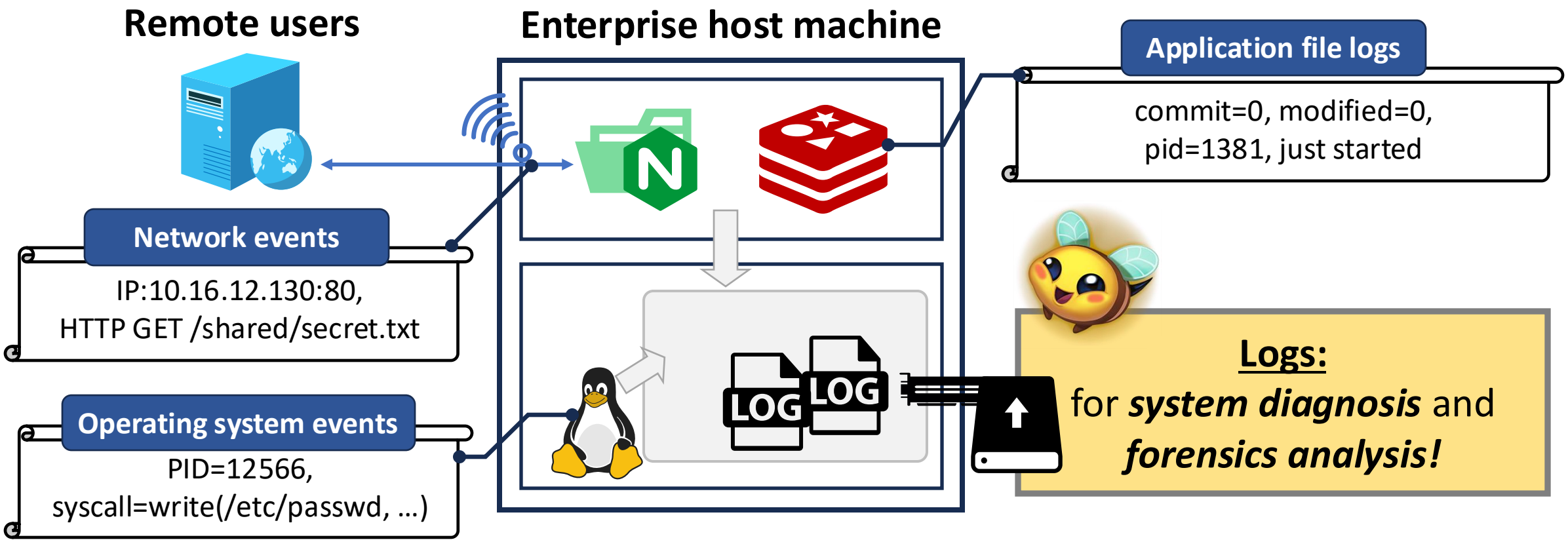
The HITCHHIKER's Guide to High-Assurance System Observability Protection with Efficient Permission Switches

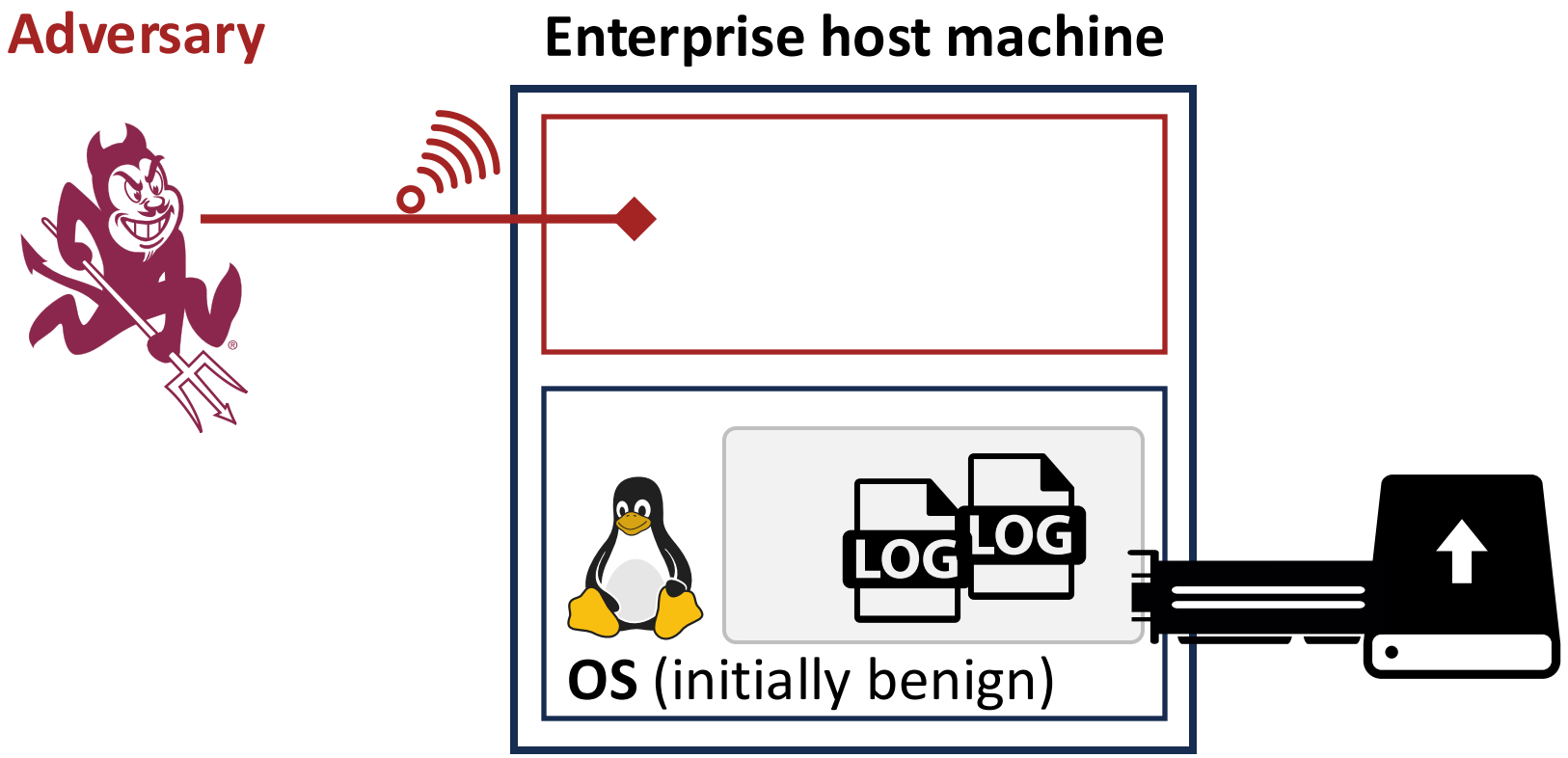
Chuqi Zhang, Jun Zeng, Yiming Zhang,
Adil Ahmad, Fengwei Zhang, Hai Jin, and Zhenkai Liang

ACM CCS, October 2024
Salt Lake City, U.S.A.

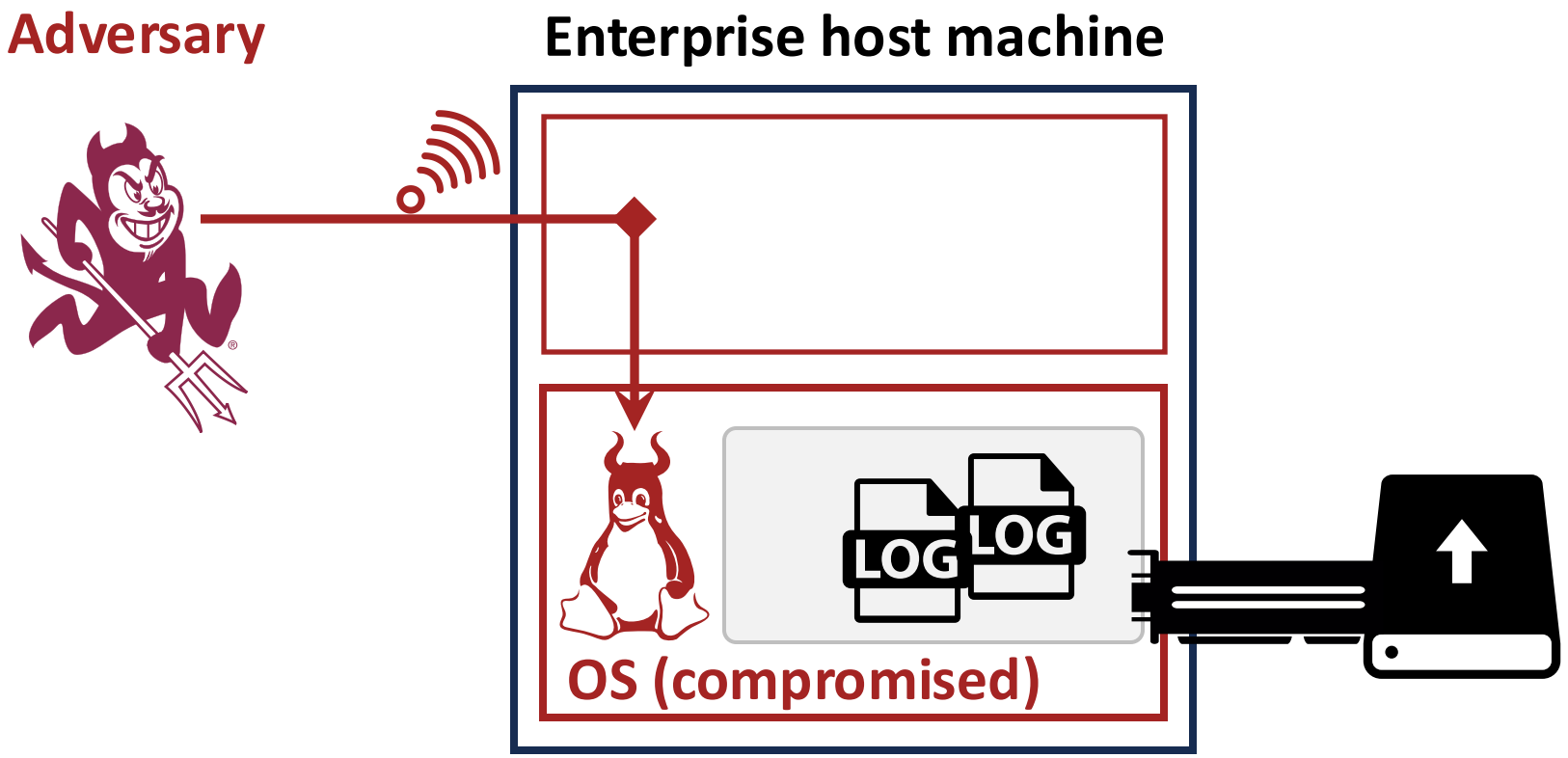
Observability captures a system's historical states



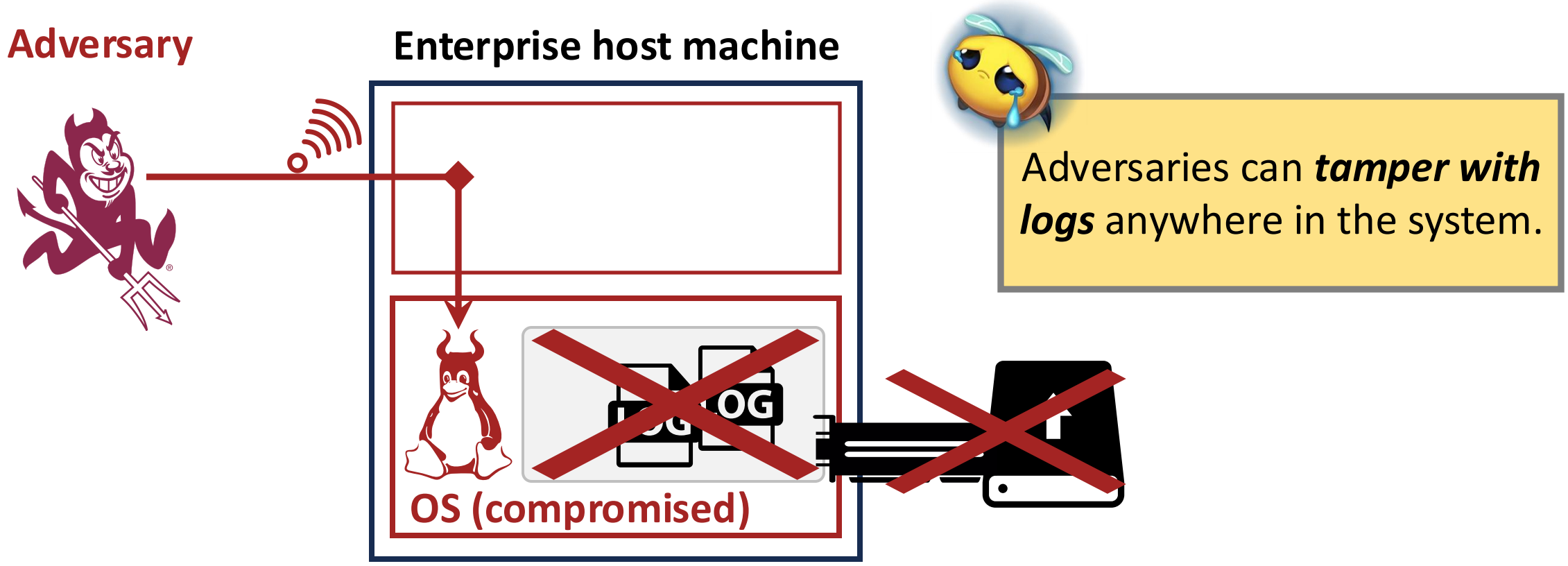
Logs are vulnerable when the OS is compromised



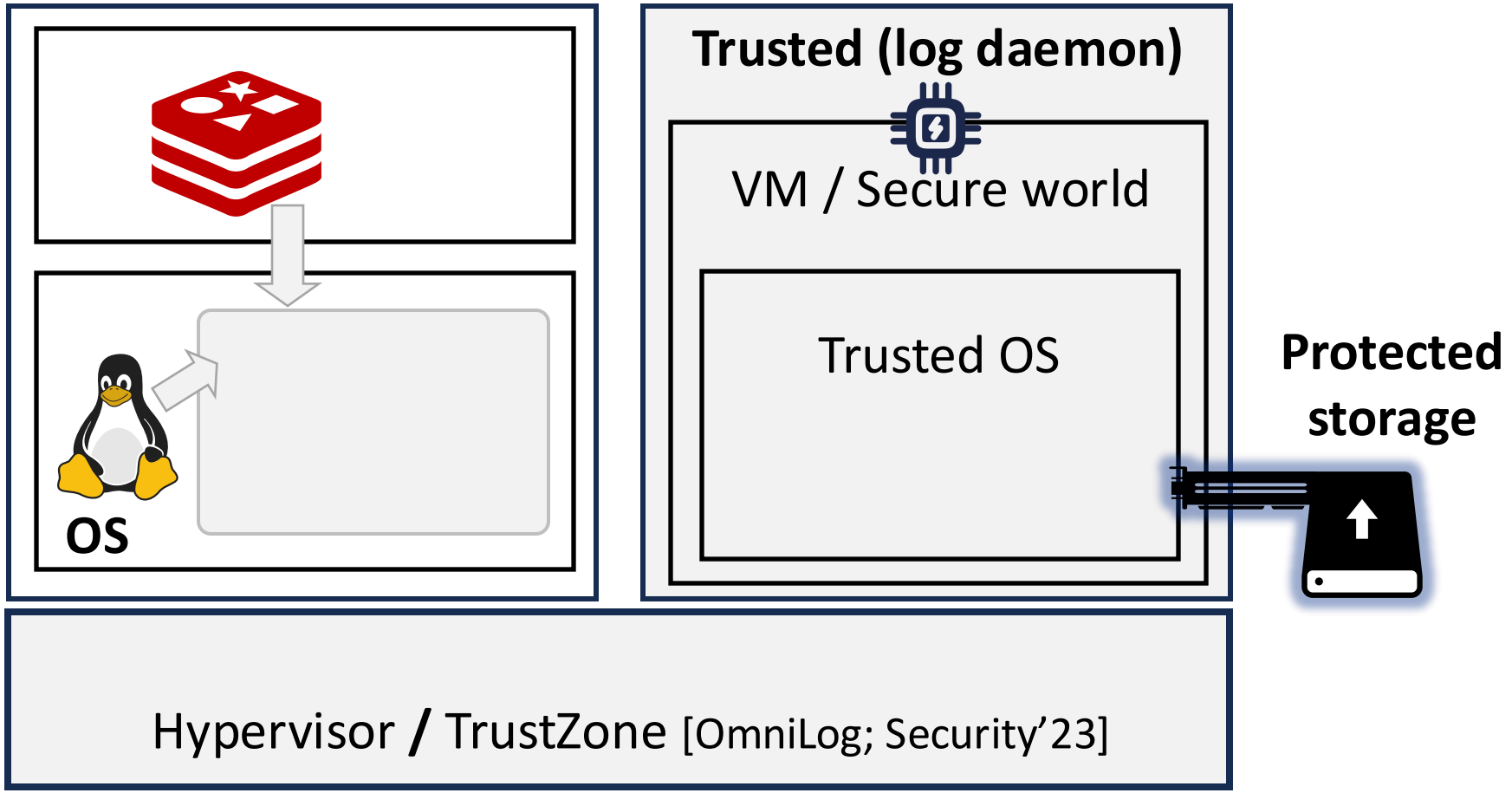
Logs are vulnerable when the OS is compromised



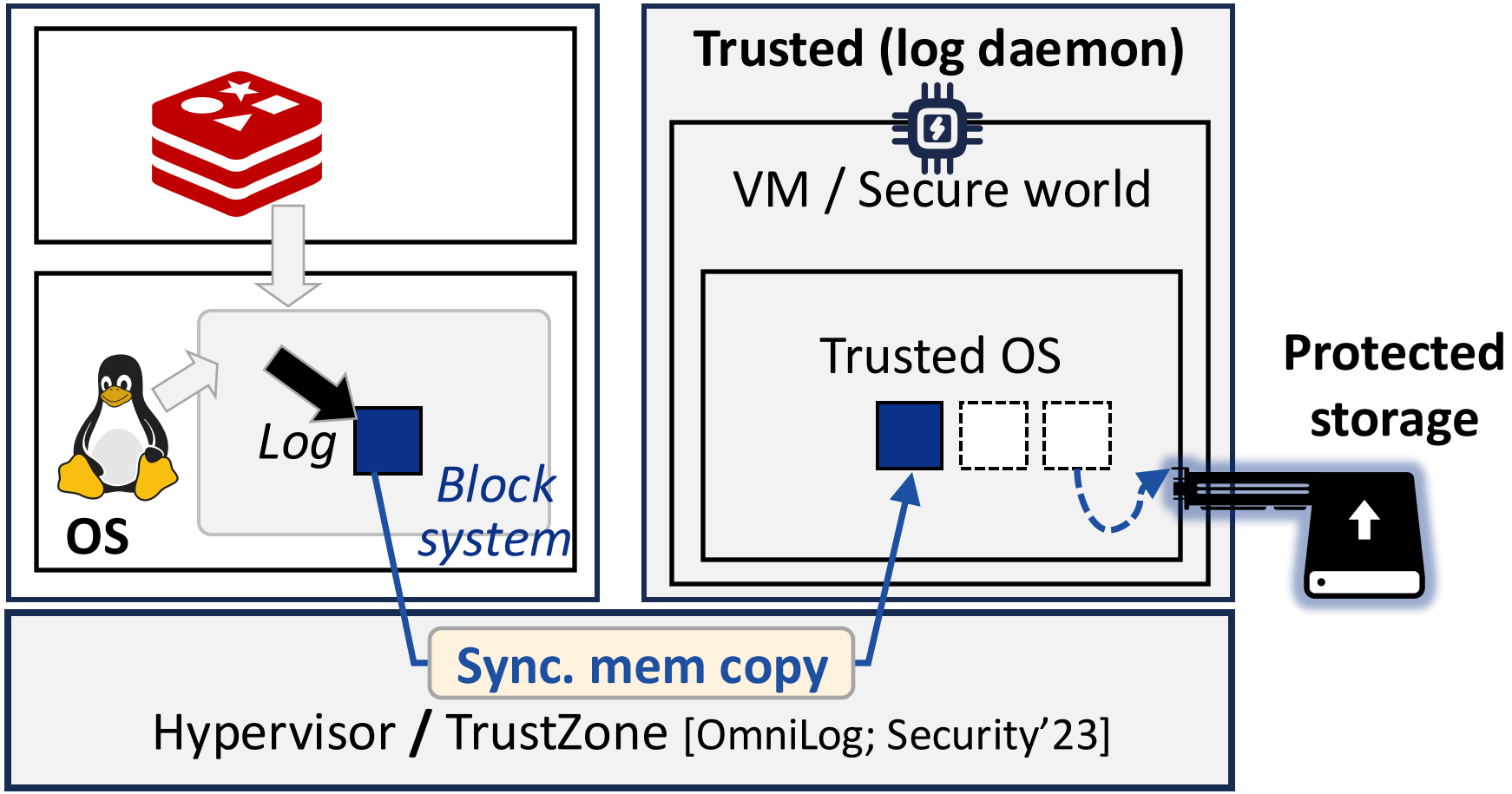
Logs are vulnerable when the OS is compromised



Prior work:
in-memory protection with eventual persistence

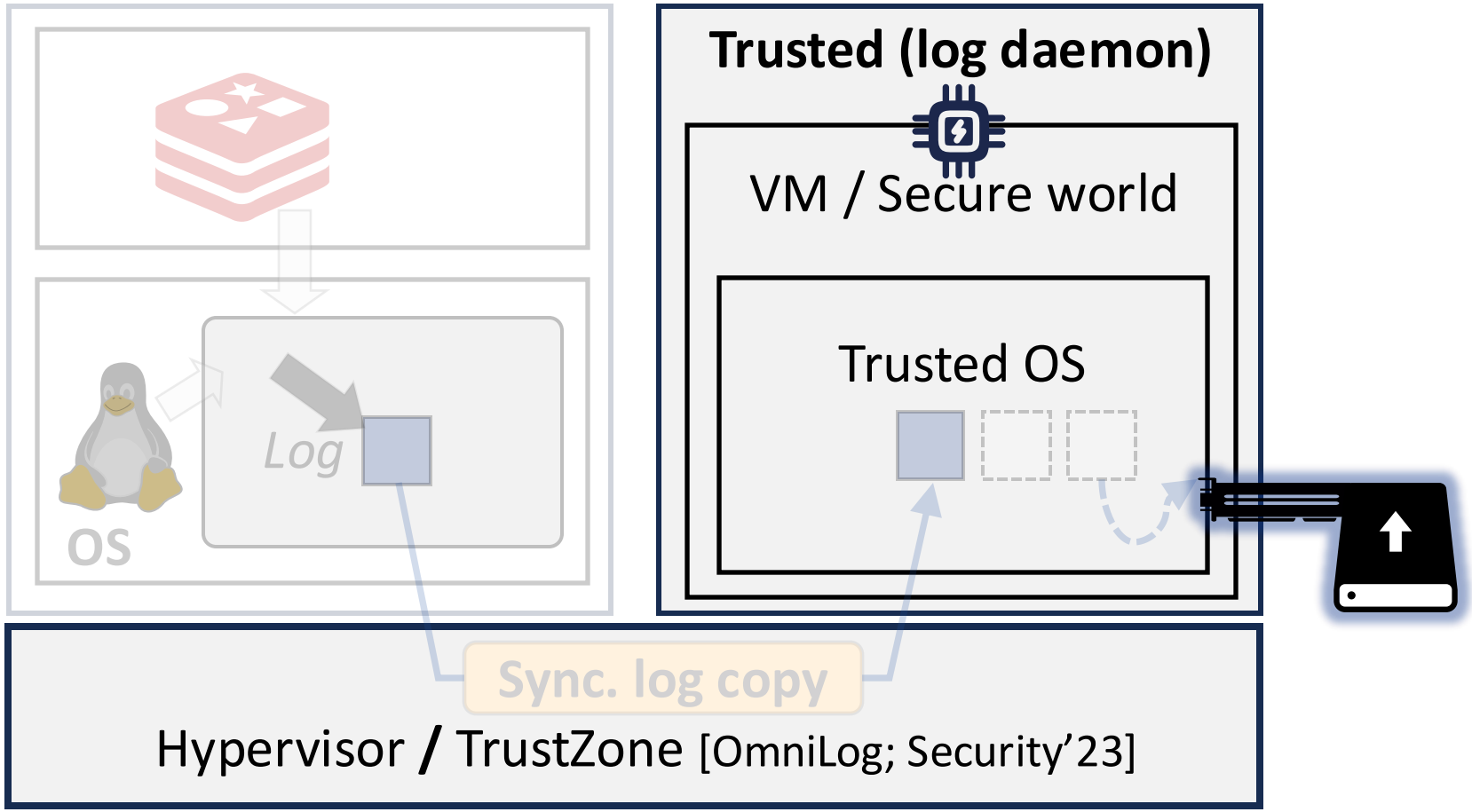


Prior work:
in-memory protection with eventual persistence



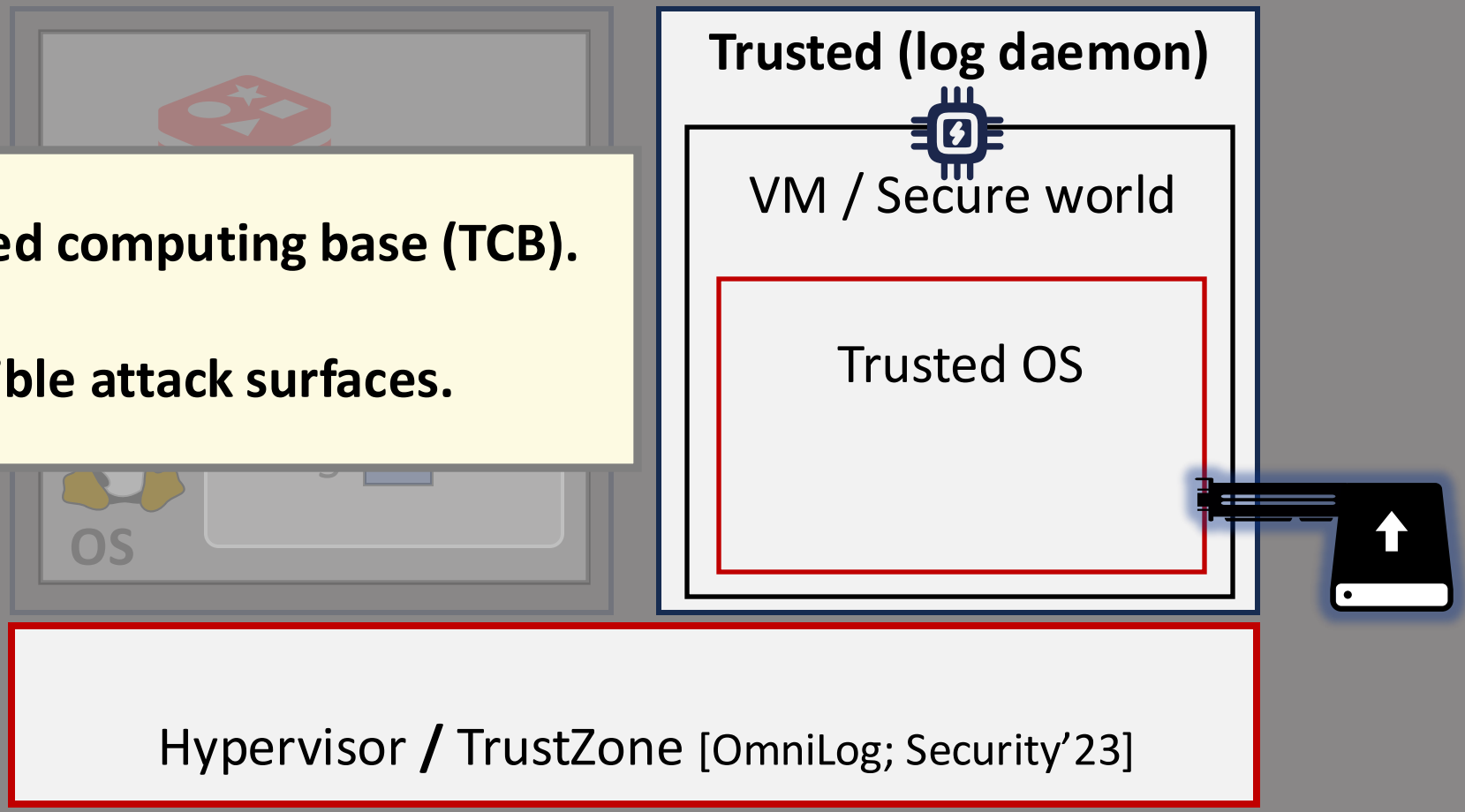
Problem 1:

Low-assurance off-the-shelf environment



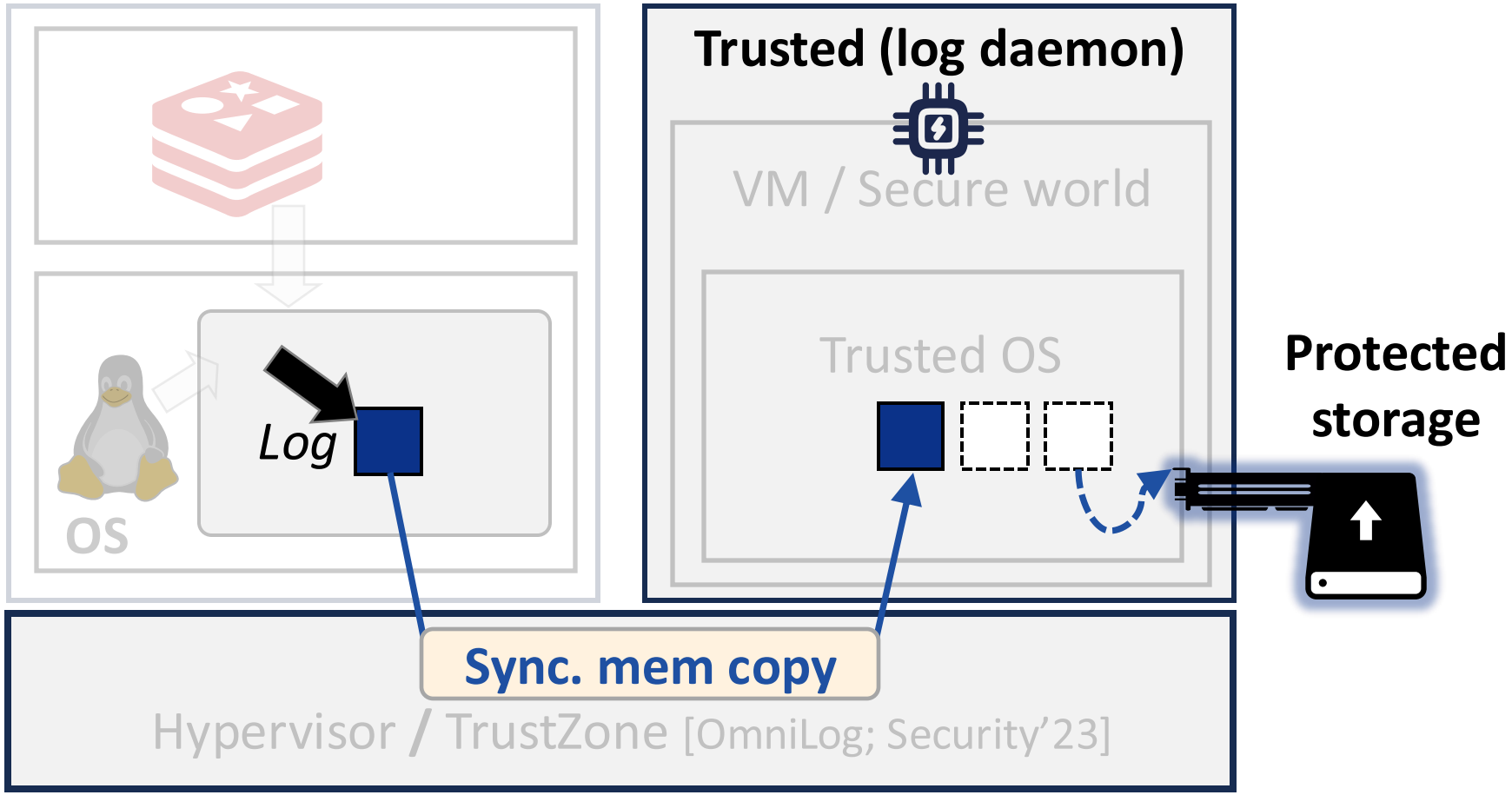
Problem 1:
Low-assurance off-the-shelf environment

- **Large trusted computing base (TCB).**
- **Non-negligible attack surfaces.**

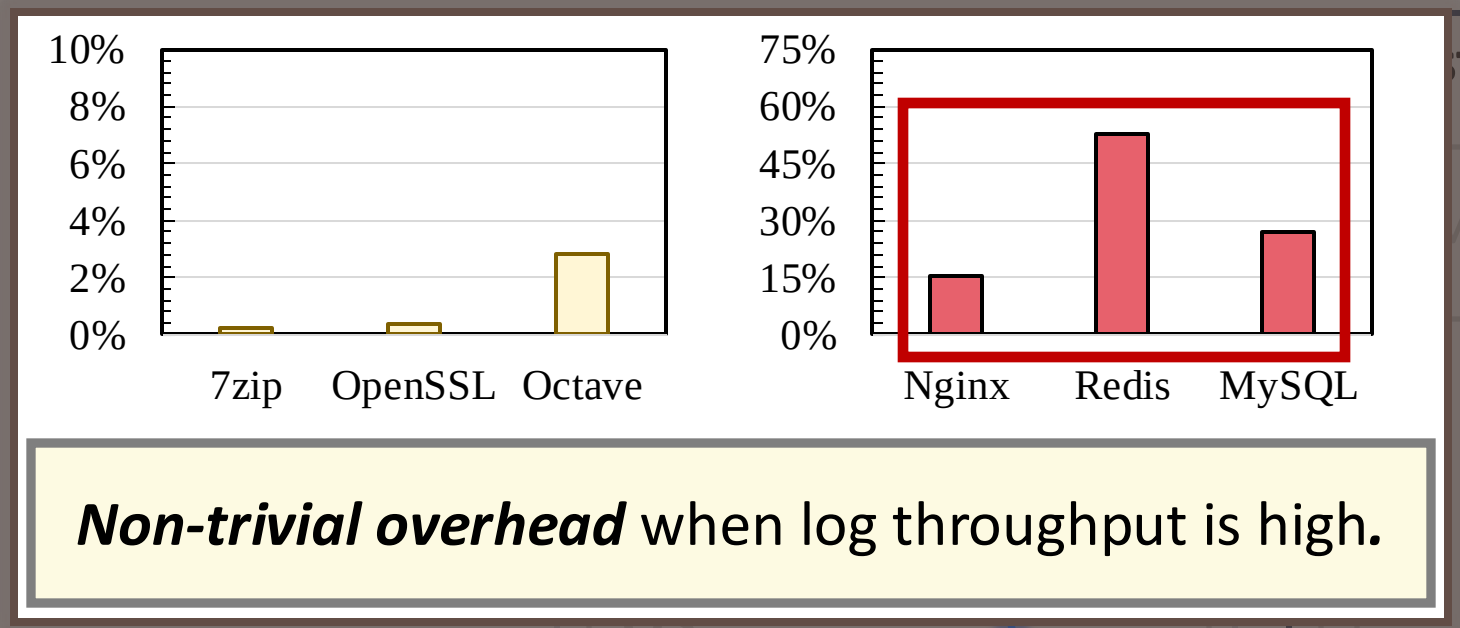


Problem 2:

High synchronous protection overhead

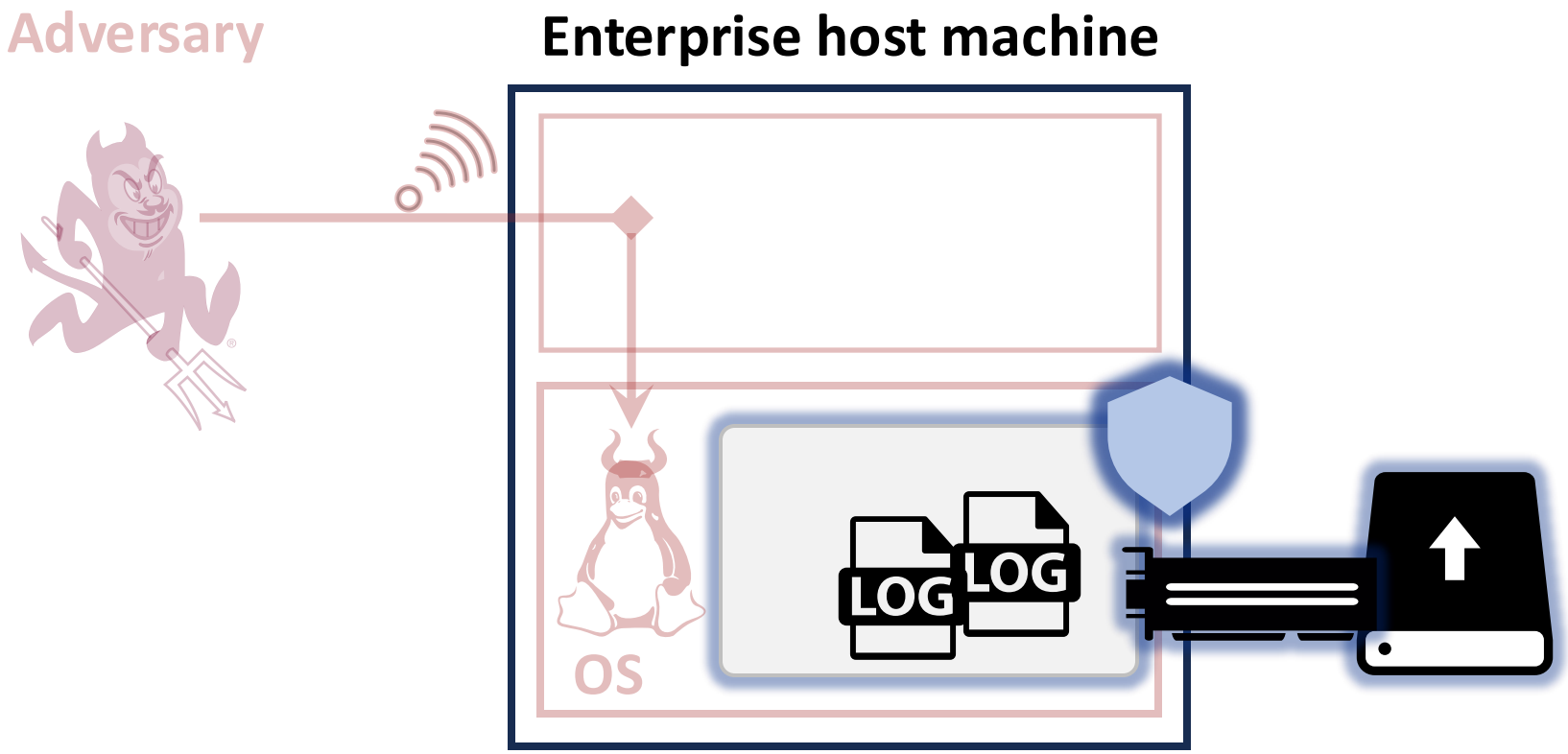


Problem 2:
High synchronous protection overhead

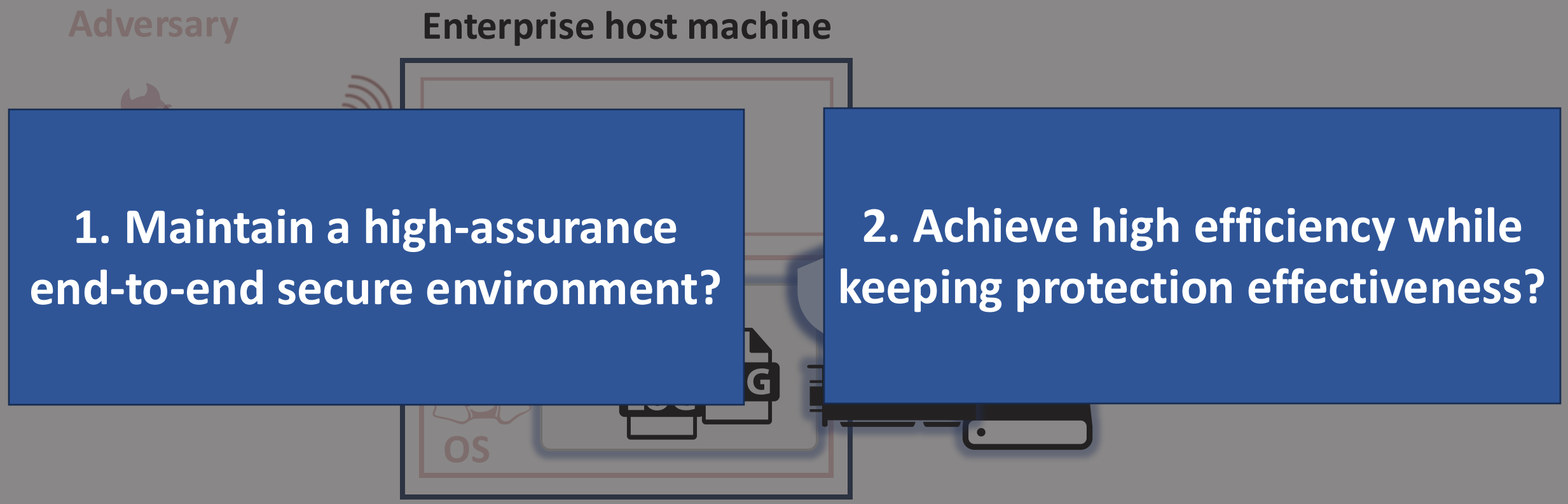


Non-trivial overhead when log throughput is high.

Research questions we ask



Research questions we ask

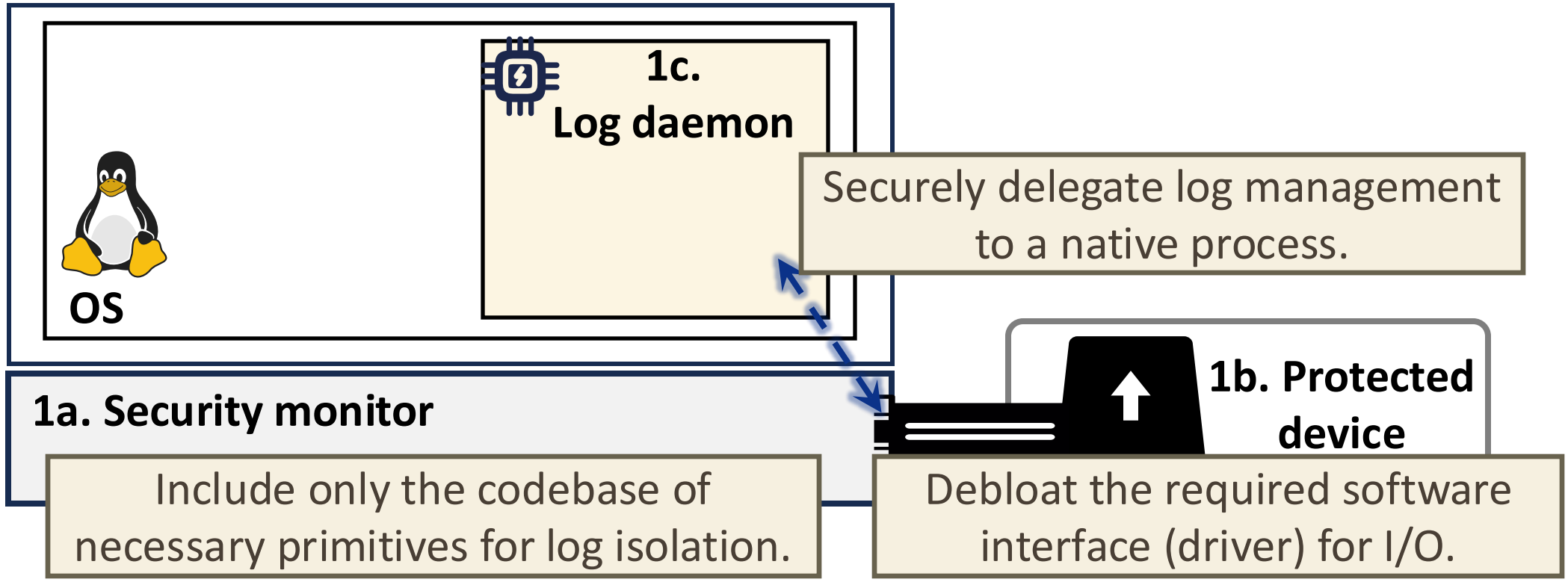


1. Maintain a high-assurance end-to-end secure environment?

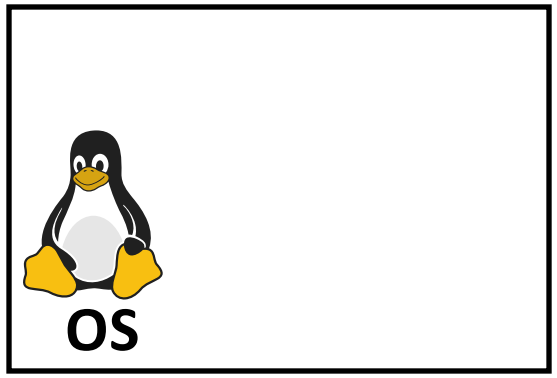
2. Achieve high efficiency while keeping protection effectiveness?

Requirements for high-assurance secure environment

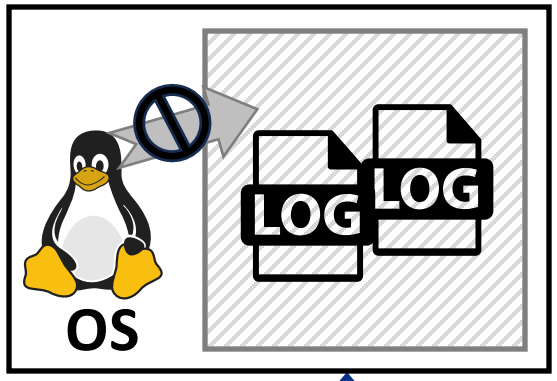
<u>Principled approach</u> to redesigning software components.



1a. Codebase with only required isolation primitives

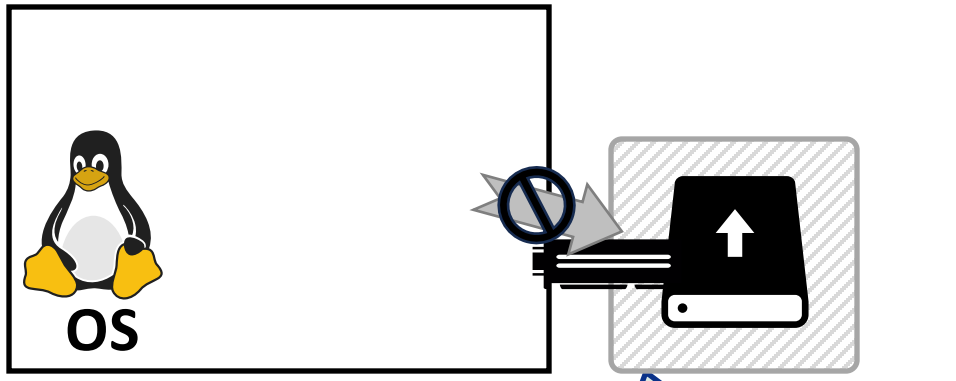


1a. Codebase with only required isolation primitives



Log integrity:
an **isolated memory** region.

1a. Codebase with only required isolation primitives

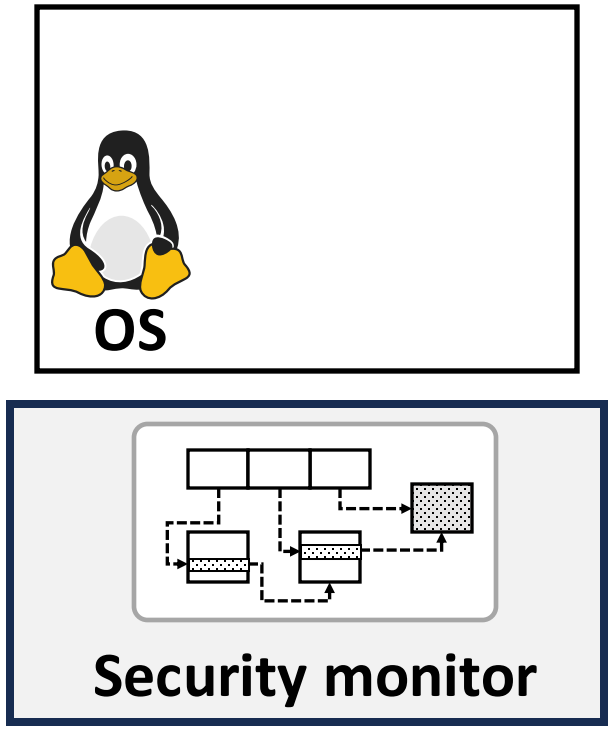


Log availability:
an **isolated** peripheral **device**.

1a. Codebase with only required isolation primitives

Memory isolation:

Stage-2 Page Table (S2PT) /
Granule Protection Table (GPT)



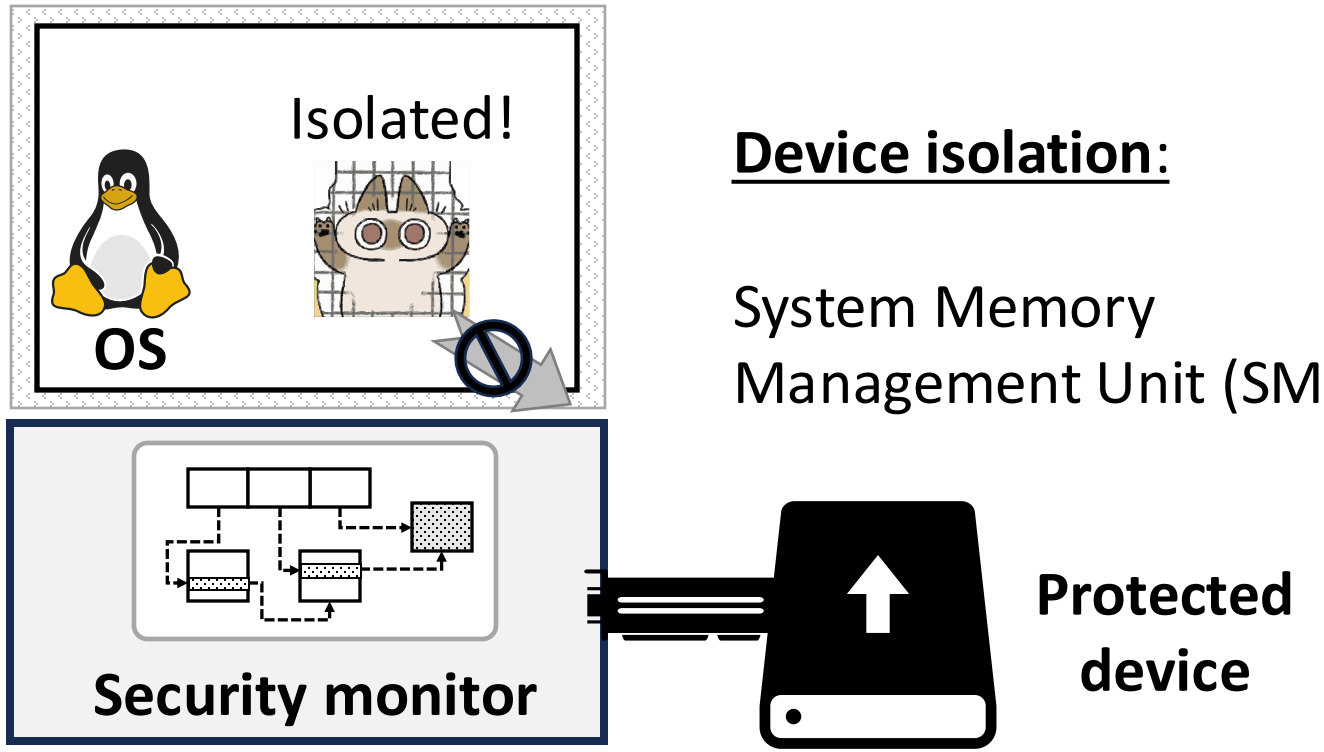
Device isolation:

System Memory
Management Unit (SMMU)

1a. Codebase with only required isolation primitives

Memory isolation:

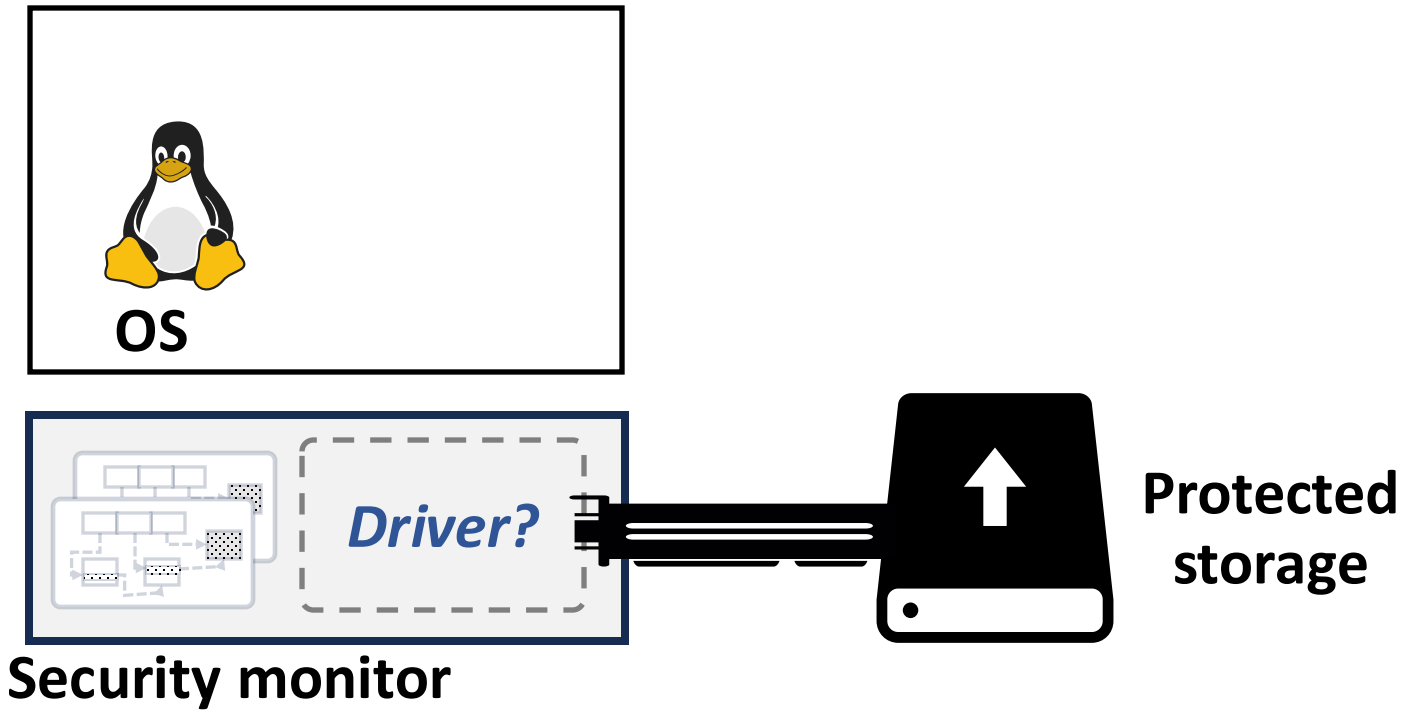
Stage-2 Page Table (S2PT) /
Granule Protection Table (GPT)



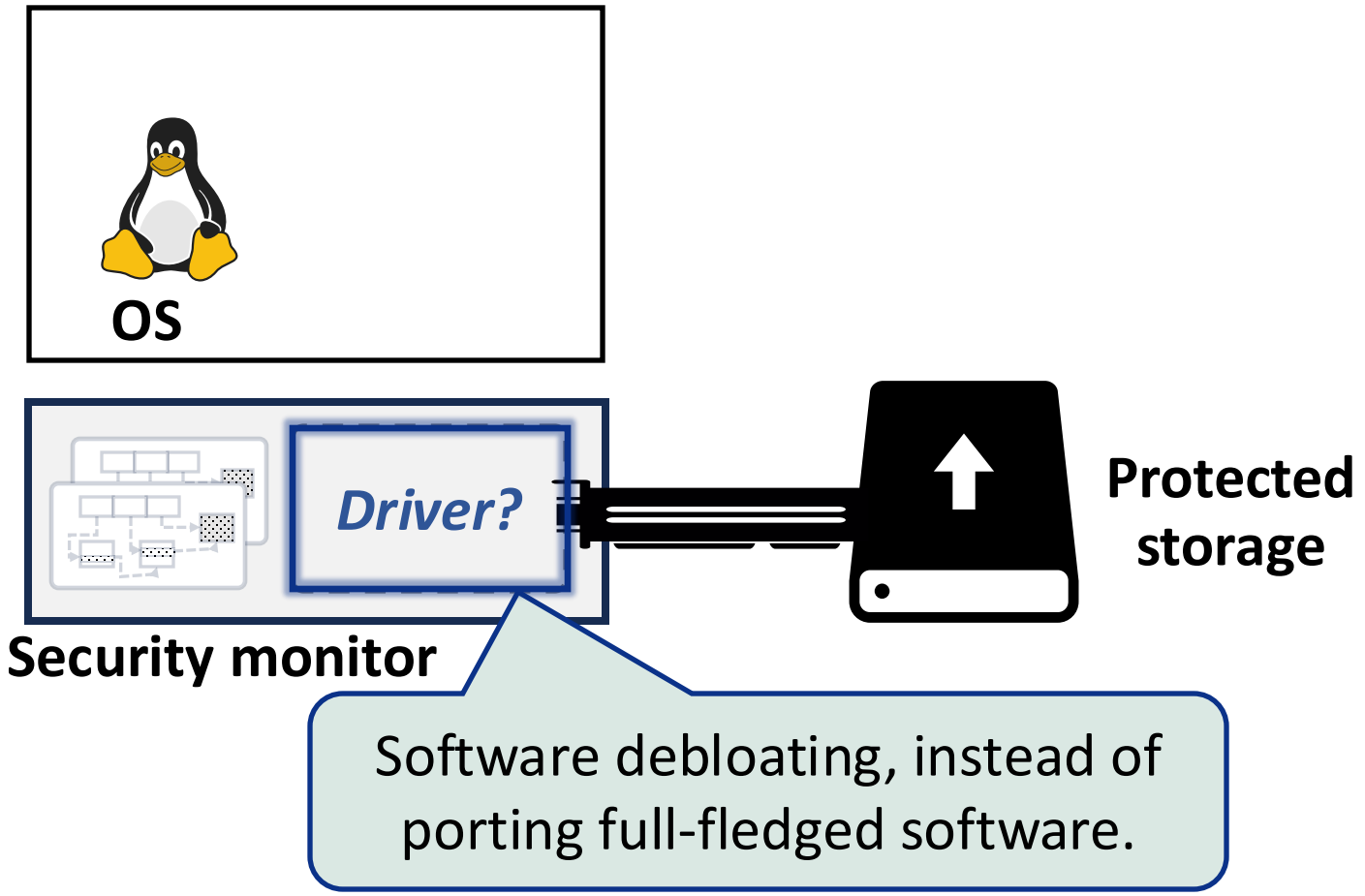
Device isolation:

System Memory
Management Unit (SMMU)

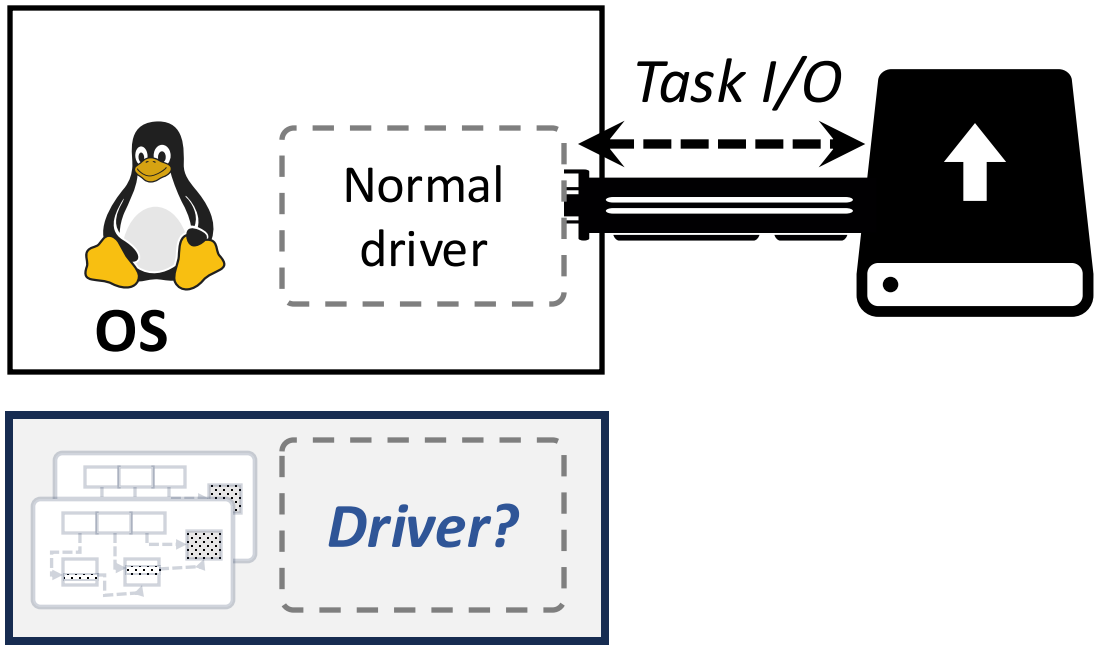
1b. Debloat software driver interface by record-replay



1b. Debloat software driver interface by record-replay

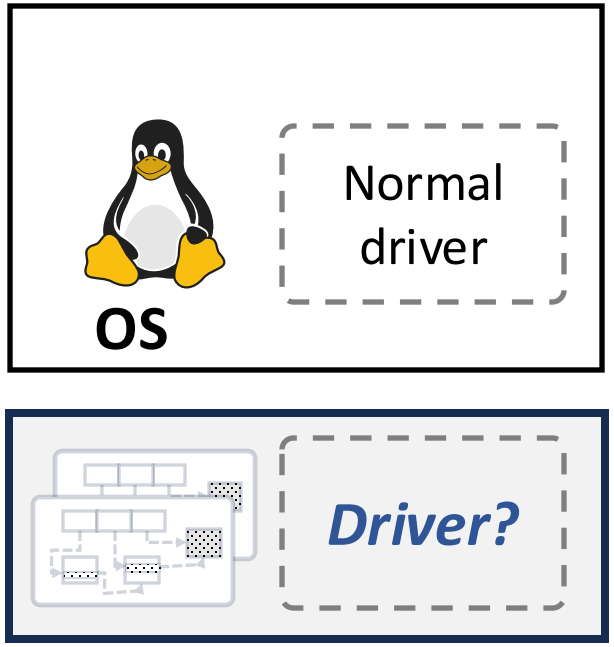


1b. Debloat software driver interface by record-replay



Sample I/O tasks: record driver-device Interactions.

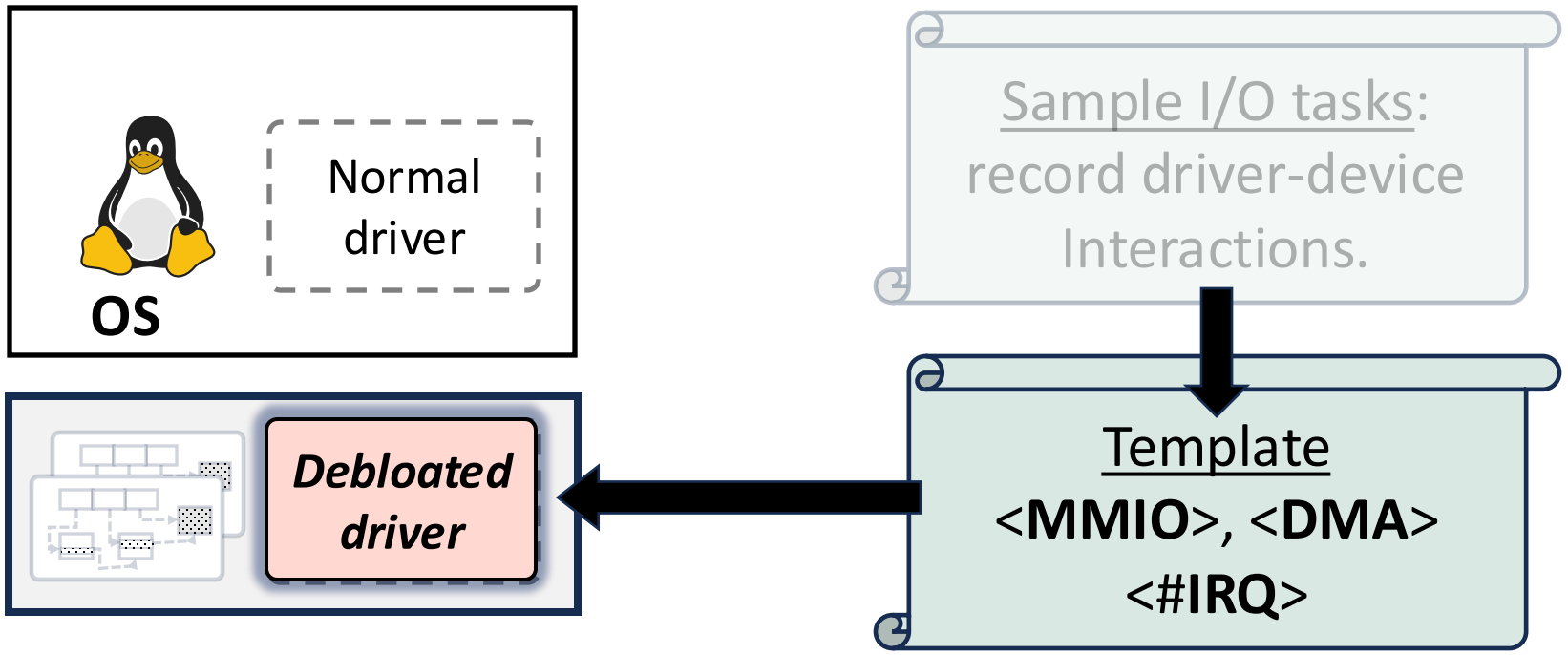
1b. Debloat software driver interface by record-replay



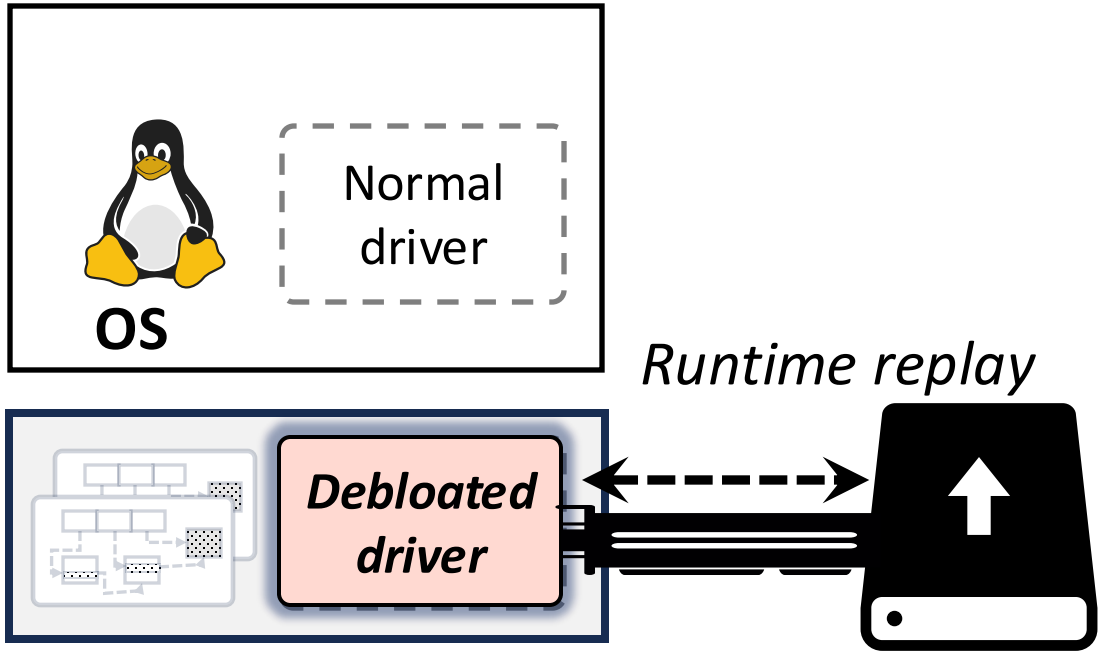
Sample I/O tasks: record driver-device Interactions.

Template
<MMIO>, <DMA>
<#IRQ>

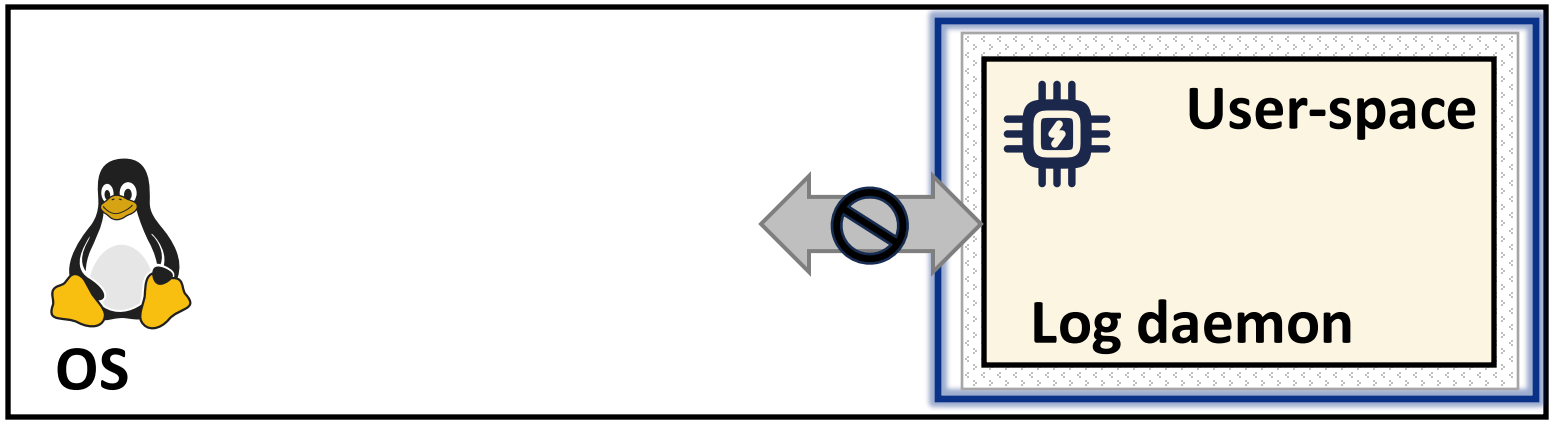
1b. Debloat software driver interface by record-replay

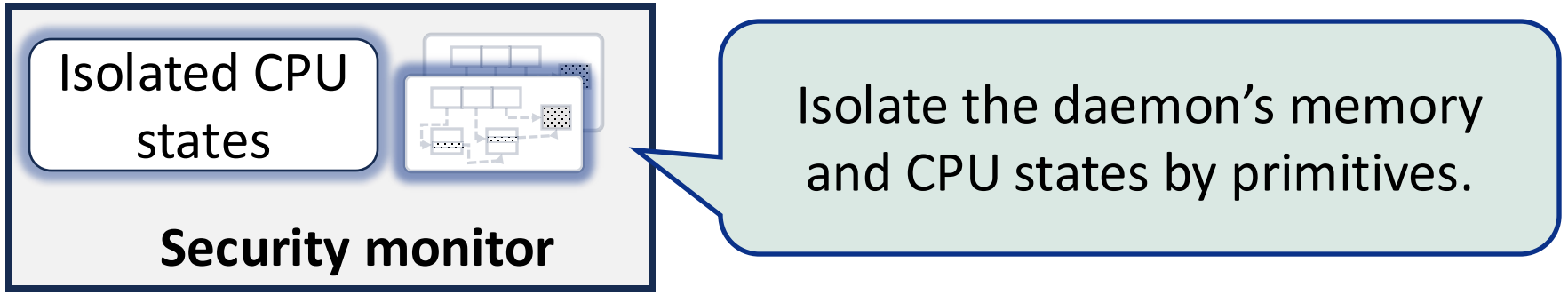


1b. Debloat software driver interface by record-replay

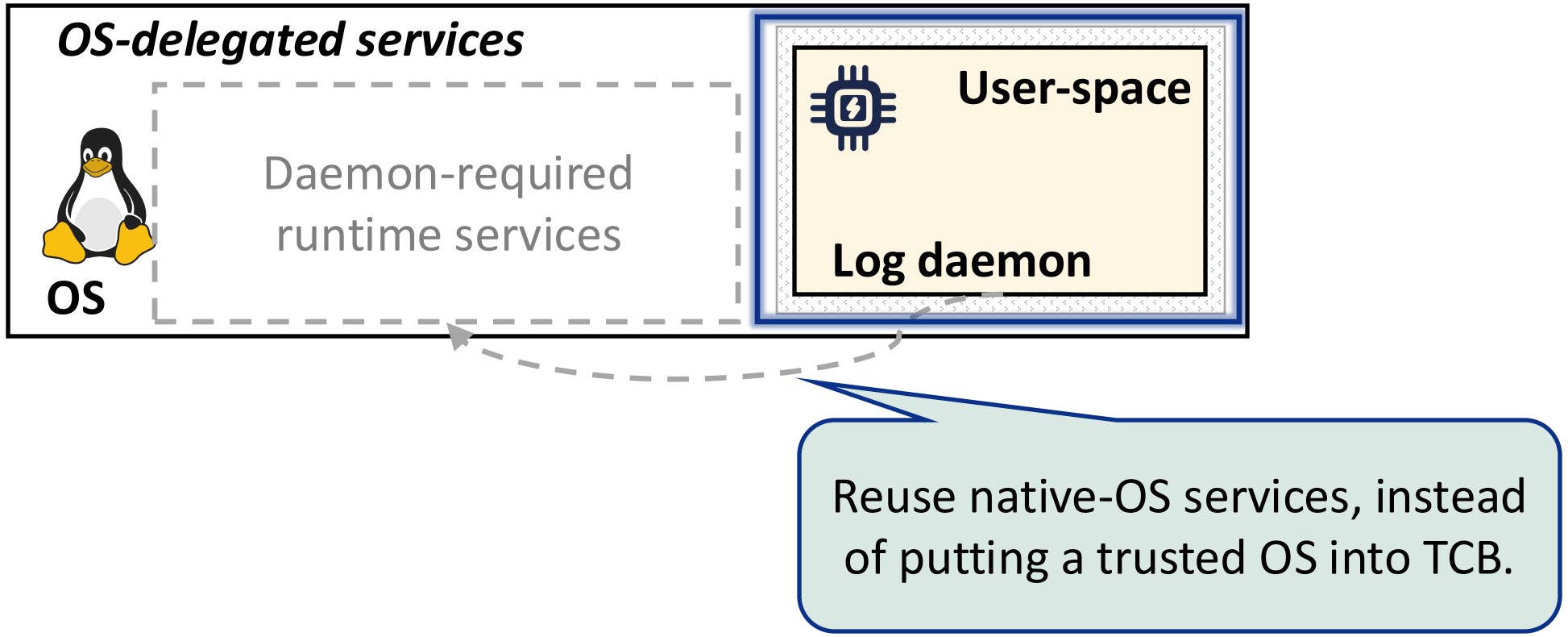


1c. Protect userspace daemon under native OS

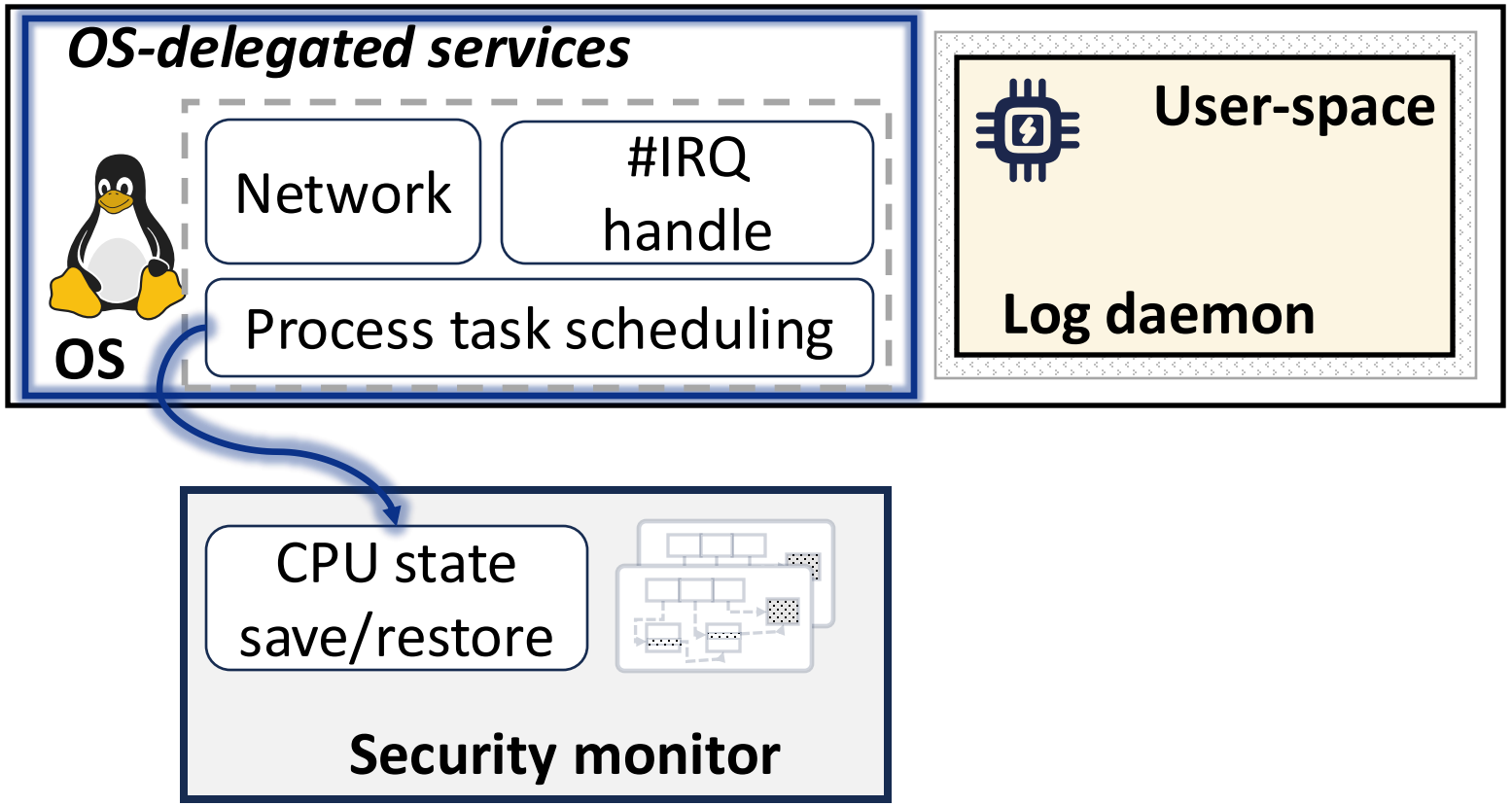




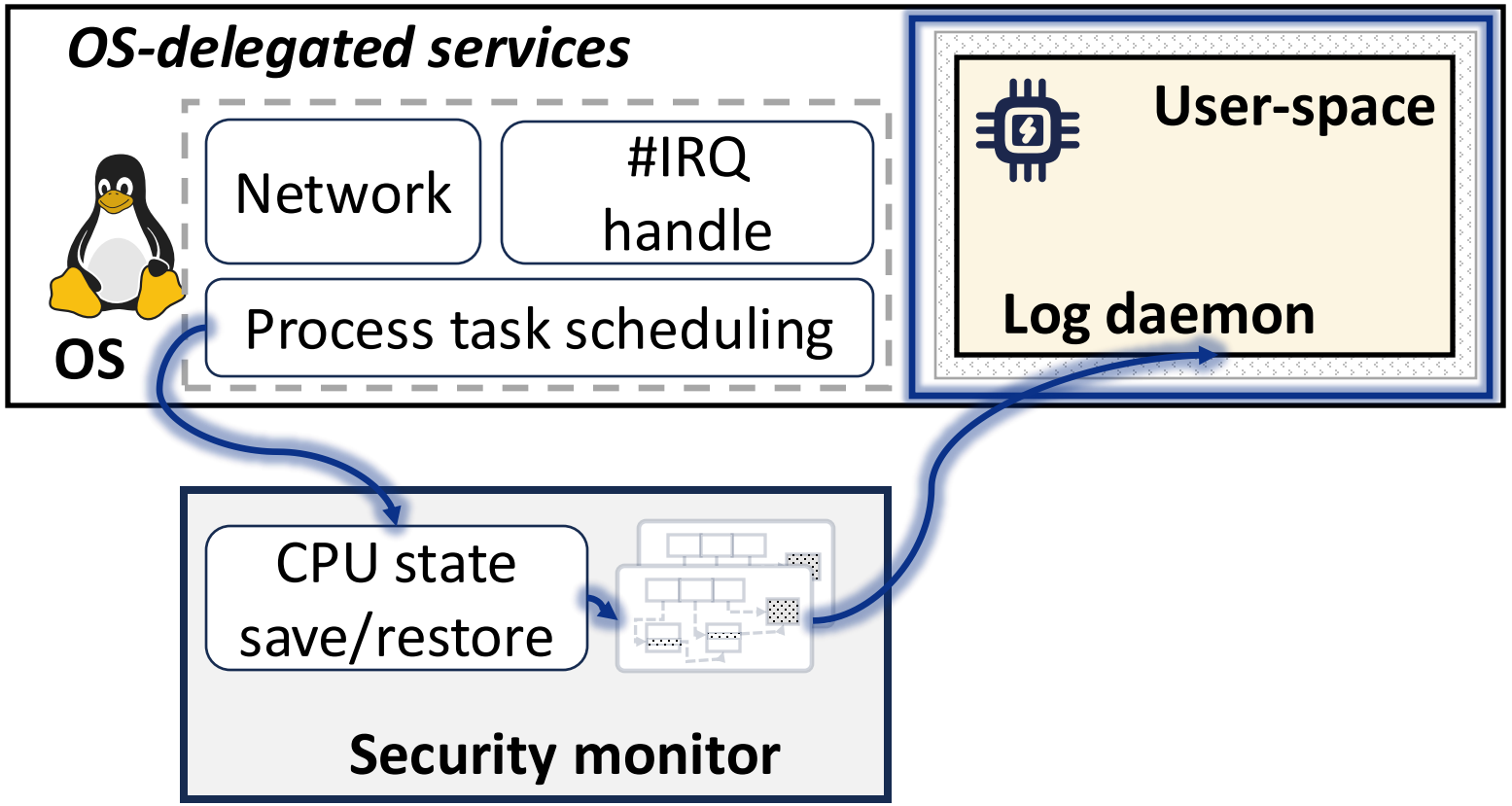
1c. Protect userspace daemon under native OS



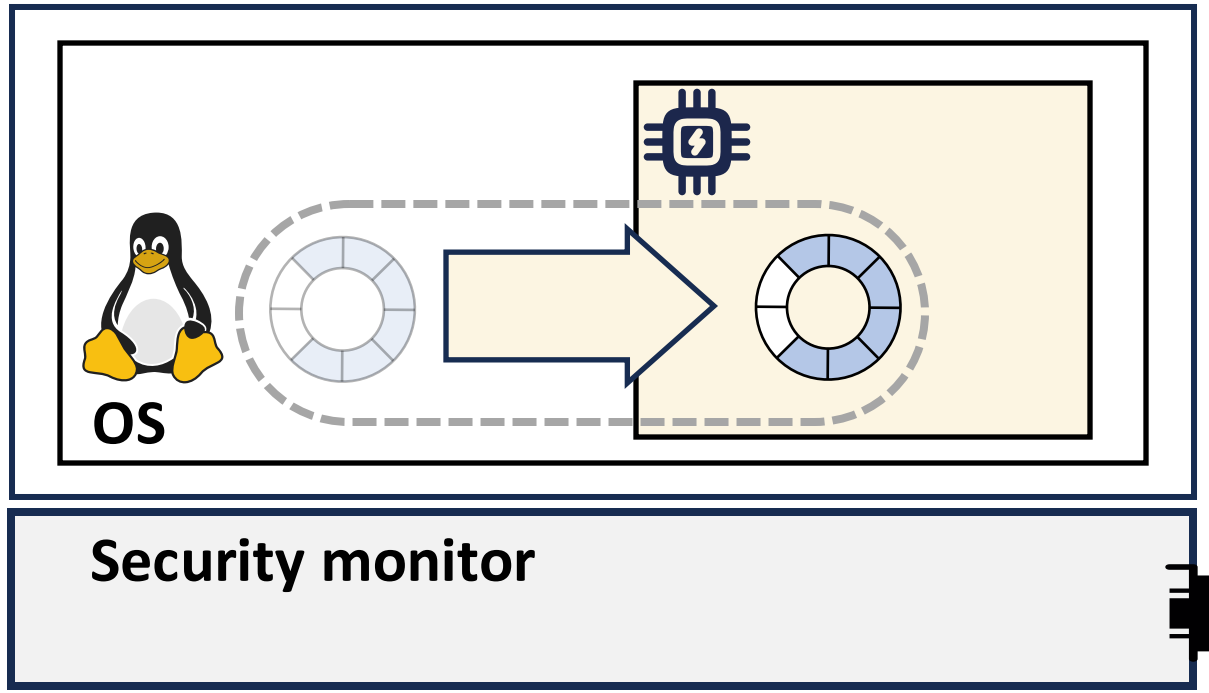
1c. Protect userspace daemon under native OS



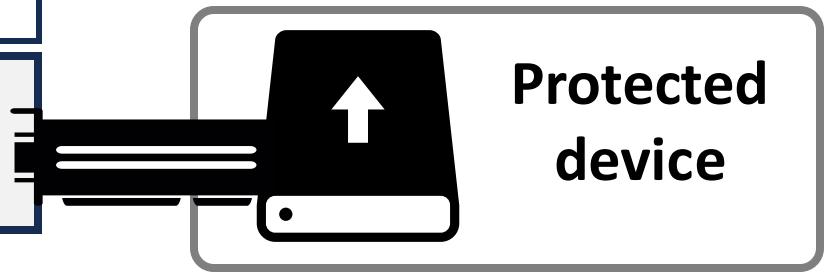
1c. Protect userspace daemon under native OS



Requirements for efficient and effective log protection

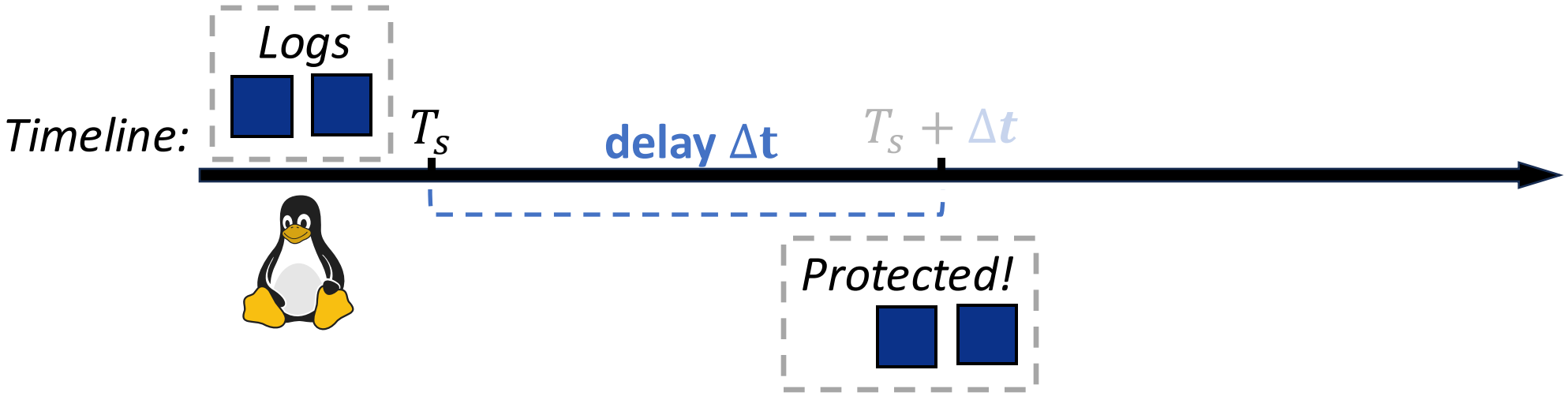


We don’t use synchronous protection due to its overhead.



Requirements for efficient and effective log protection

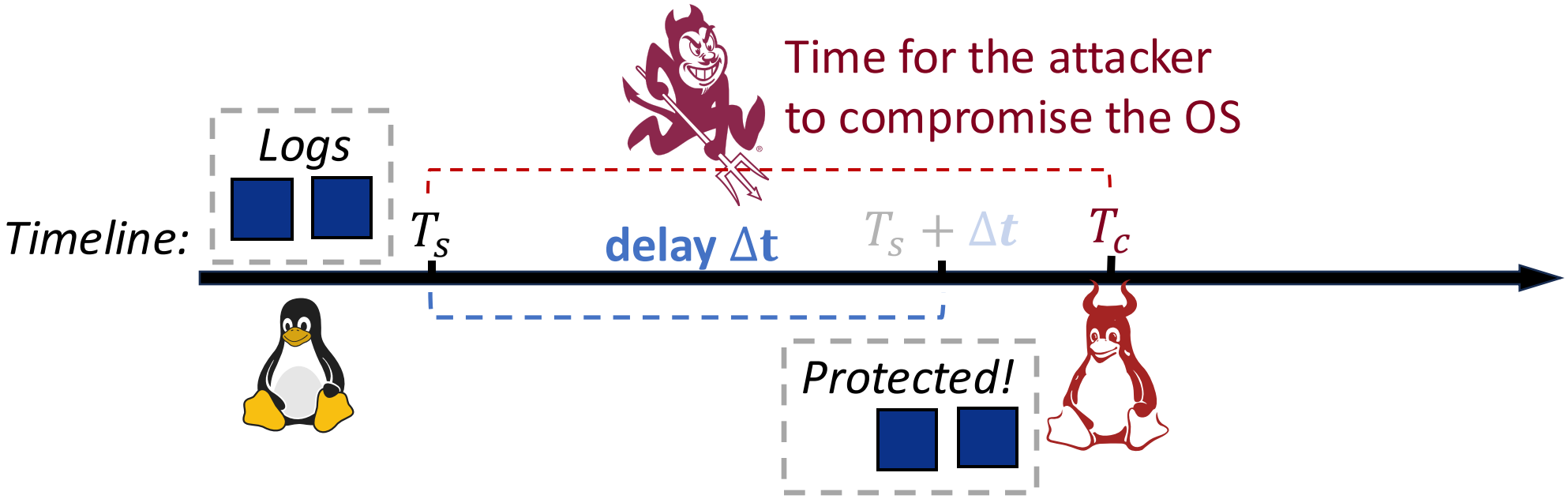
For security, log protection can not be **unbounded asynchronous.**



(Will take **an unbounded delay Δt** for async. protection)

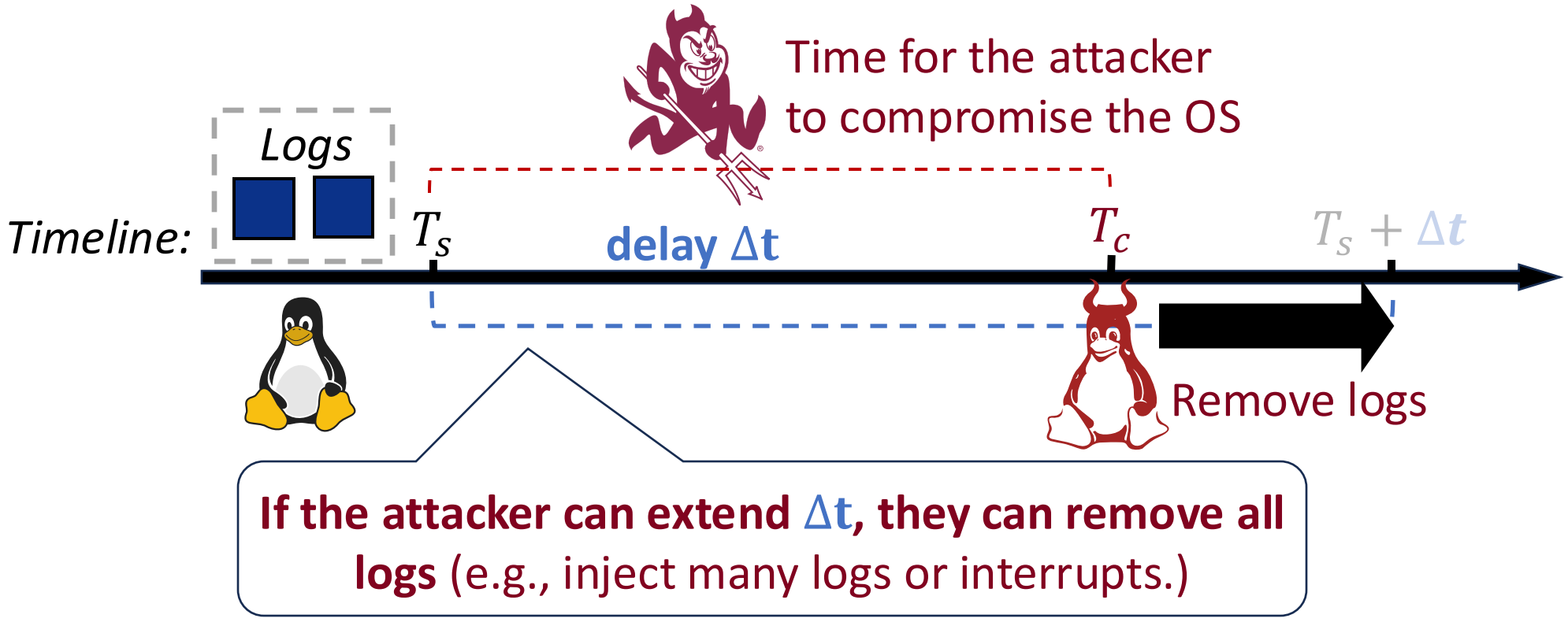
Requirements for efficient and effective log protection

For security, log protection can not be **unbounded asynchronous.**

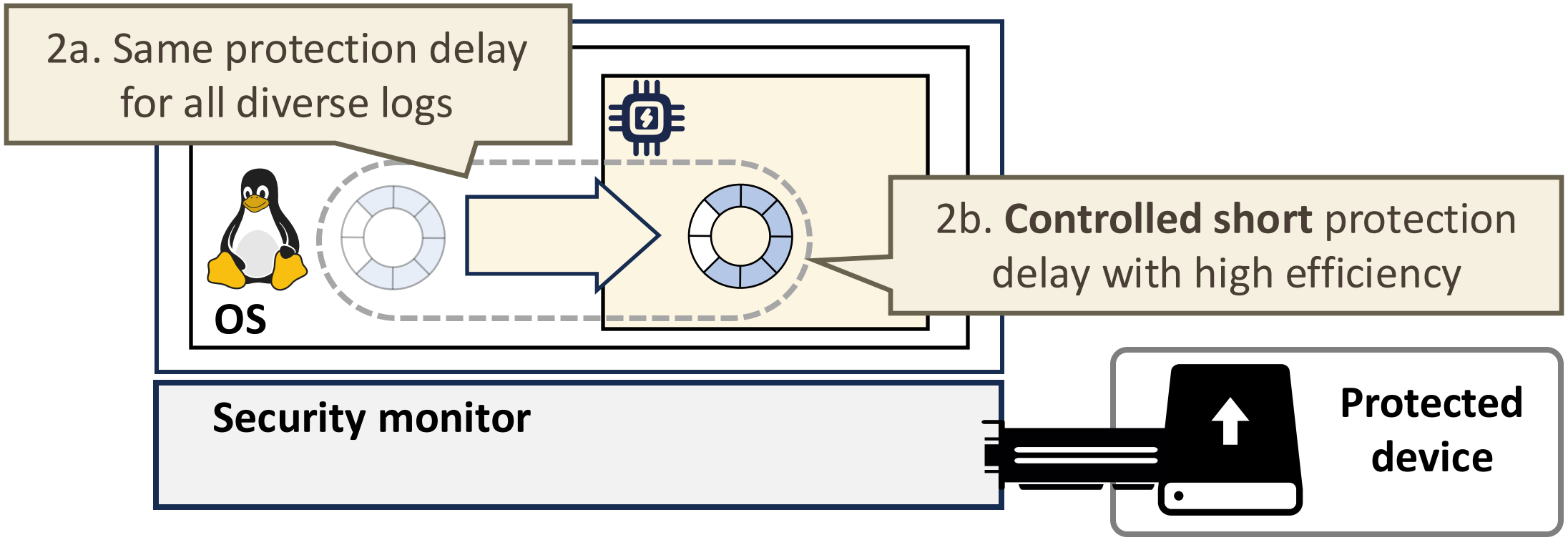


Requirements for efficient and effective log protection

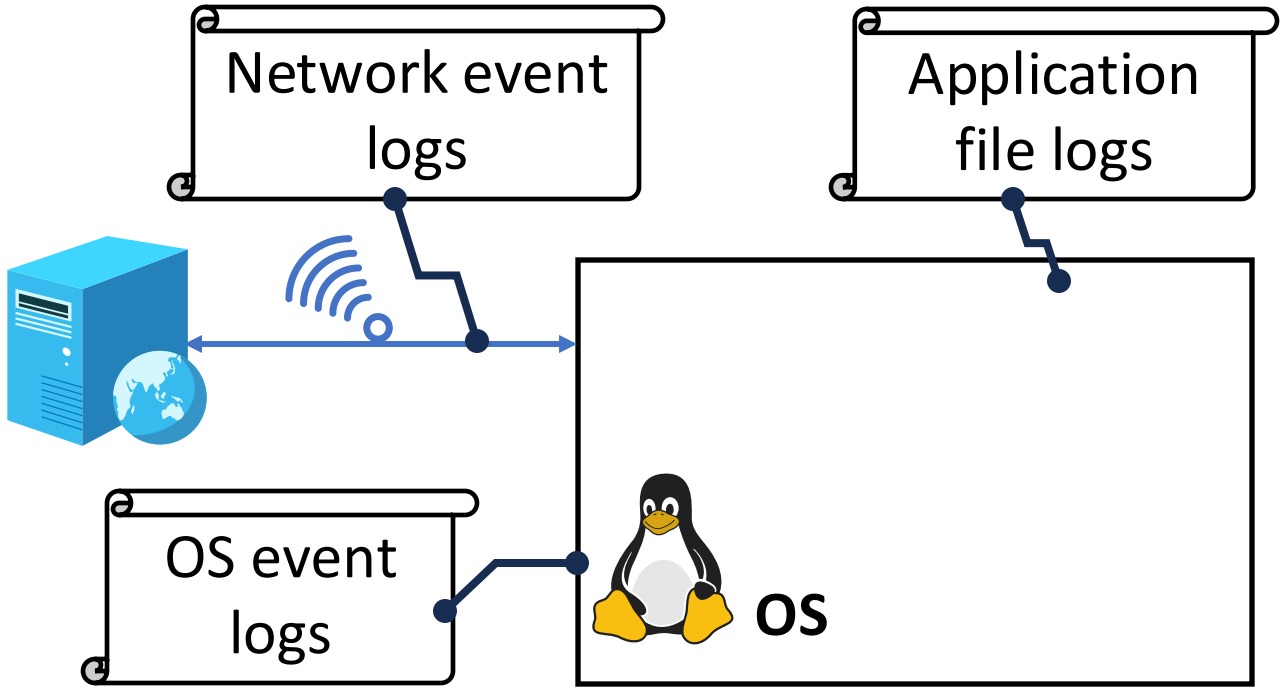
For security, log protection can not be **unbounded asynchronous.**



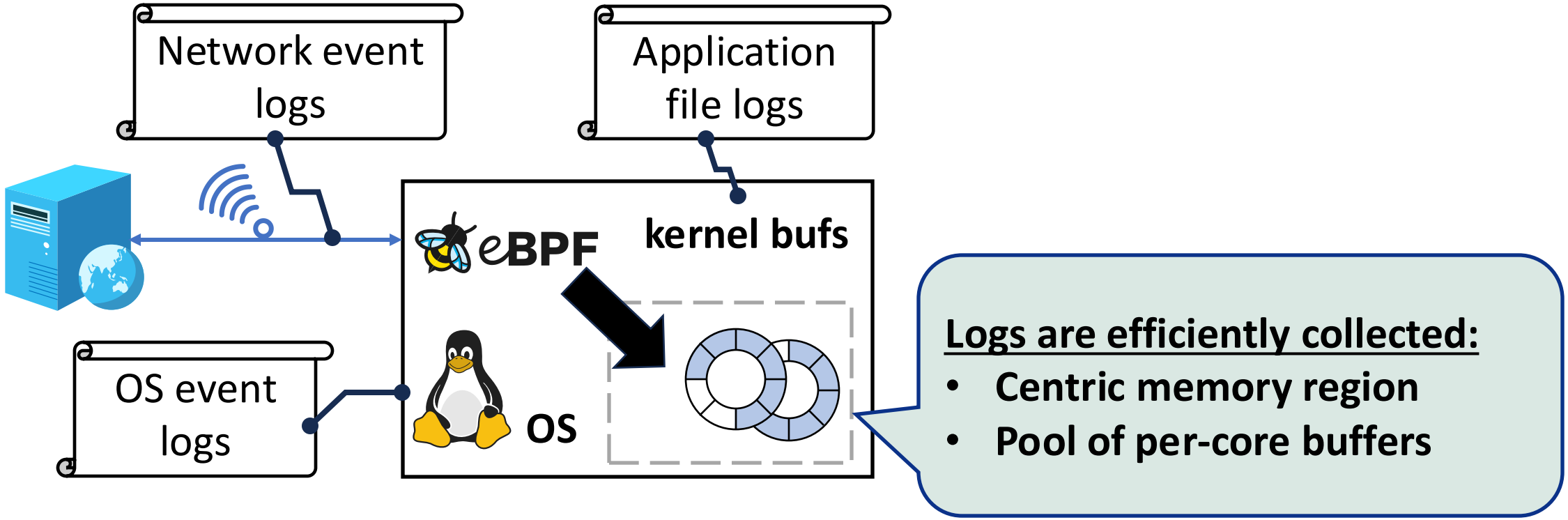
Requirements for efficient and effective log protection



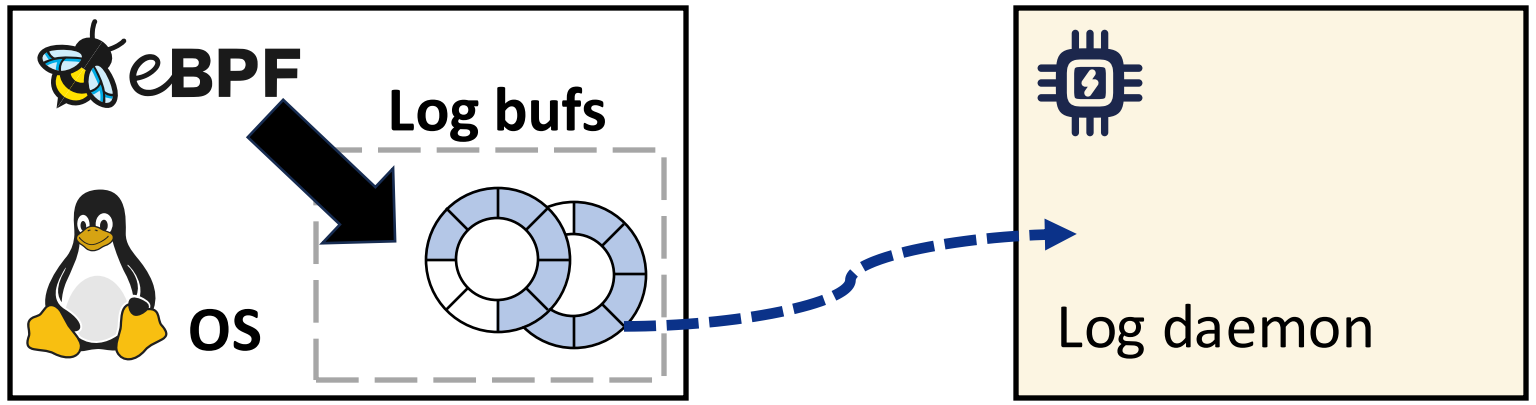
2a. Unified log collection using eBPF



2a. Unified log collection using eBPF

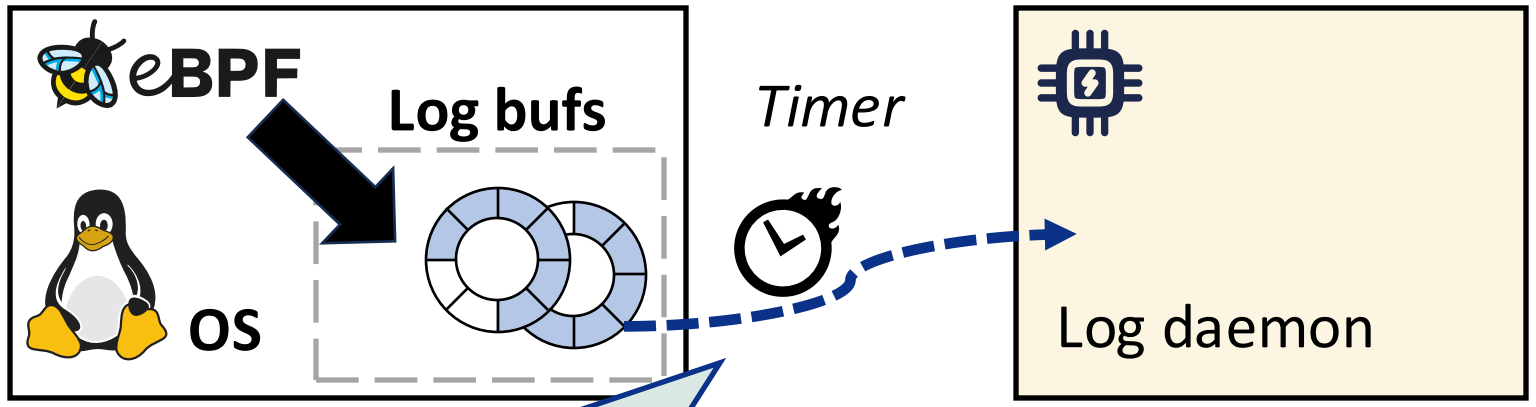


2b. Bounded hardware timer and fast permission switch



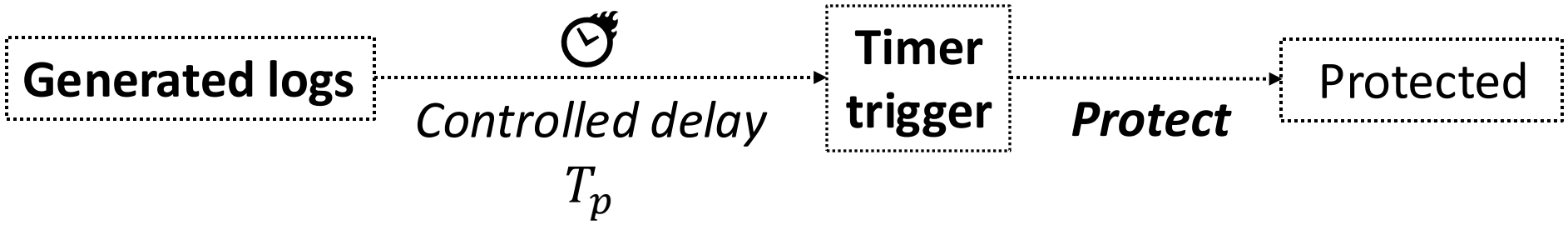
Unified logs to be protected with ***controlled delay***.

2b. Bounded hardware timer and fast permission switch

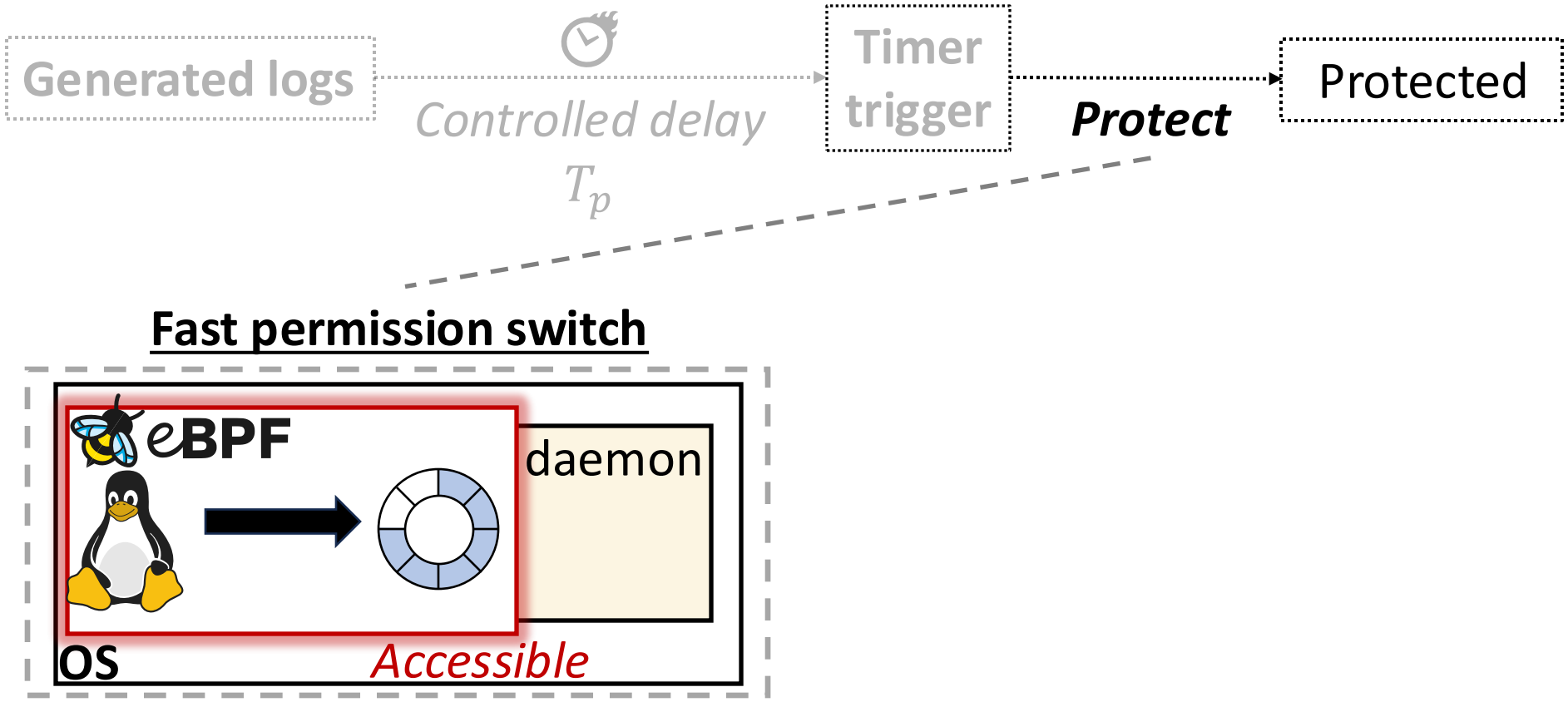


- Periodically triggers protections (timer interrupts).
- Prioritized and isolated timer interrupts.

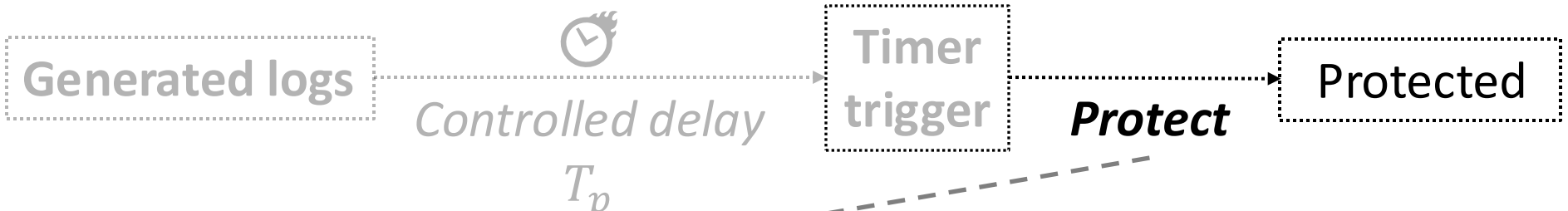
2b. Bounded hardware timer and fast permission switch



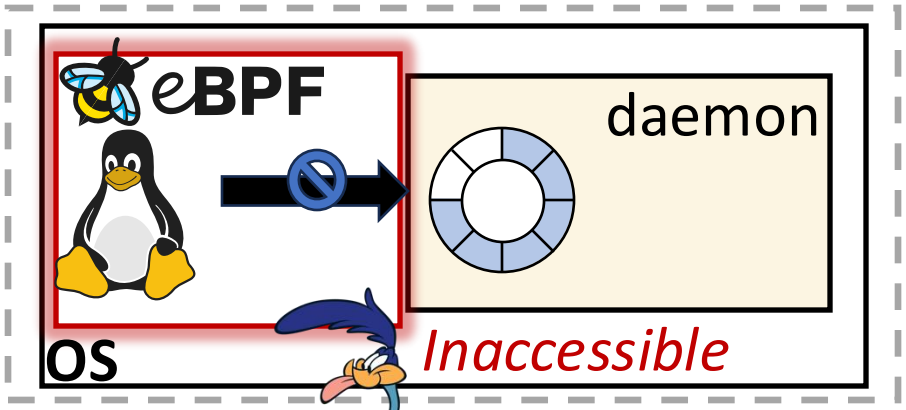
2b. Bounded hardware timer and fast permission switch



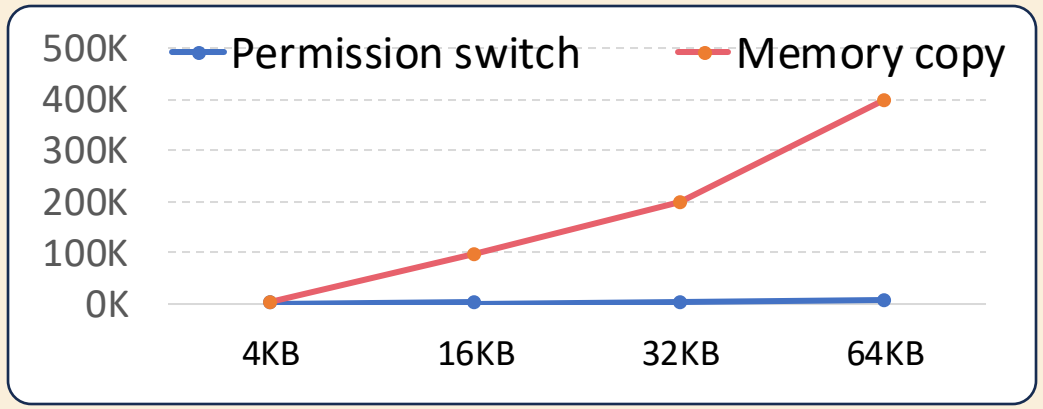
2b. Bounded hardware timer and fast permission switch



Fast permission switch



CPU cycles for batched buffer protection



Hitchhike the logs (fast)

Hitch a ride to protect logs by removing OS's memory permissions.

We evaluated HITCHHIKER

Security environment's TCB and attack surface

Log protection effectiveness

Log protection efficiency

Secure environment

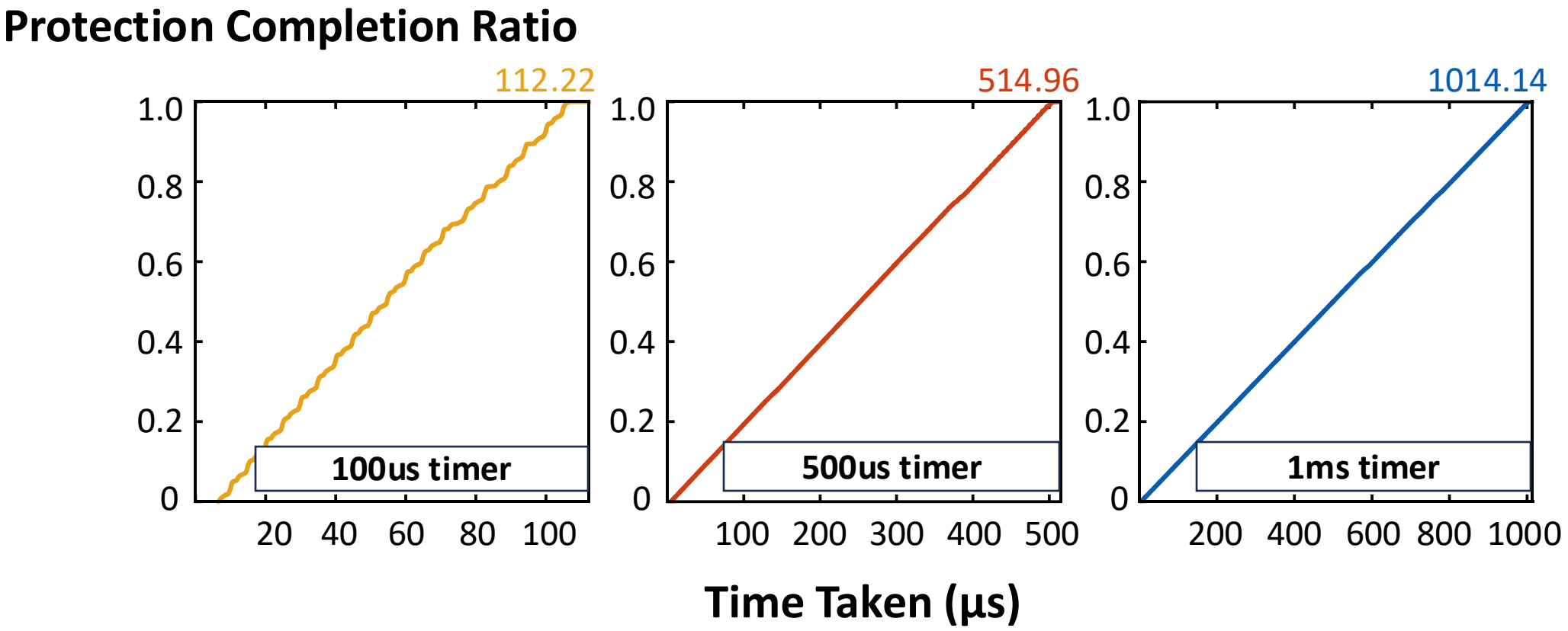
TCB: Reduced more than 10× compared to off-the-shelf environments.

Mitigated issues of off-the-shelf environment (TrustZone)

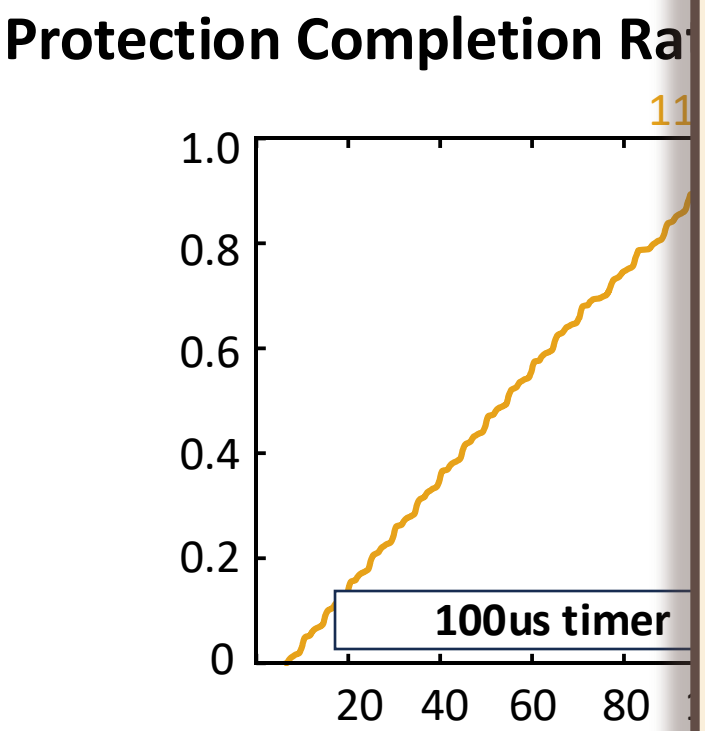
Issues	Issues
I01. SW drivers run in the TEE kernel space	I02. Wide interfaces between TEE system subcomponents
I03. Excessively large TEE TCBs	I04. TAs can map physical memory in the NW
I05. Information leaks to NW through debugging channel	I06. Absent or weak ASLR implementations
I07. No stack cookies, guard pages, or execution protection	I08. Lack of software-independent TEE integrity reporting
I09. Ill-supported TA revocation	I10. Validation bugs within the secure monitor
I11. Validation bugs within TAs	I12. Validation bugs within the trusted kernel
I13. Validation bugs in secure boot loader	I14. Bugs in memory protection
I15. Bugs in configuration of peripherals	I16. Bugs in security mechanisms
I17. Concurrency bugs (from multiple TAs)	I18. Software side-channels

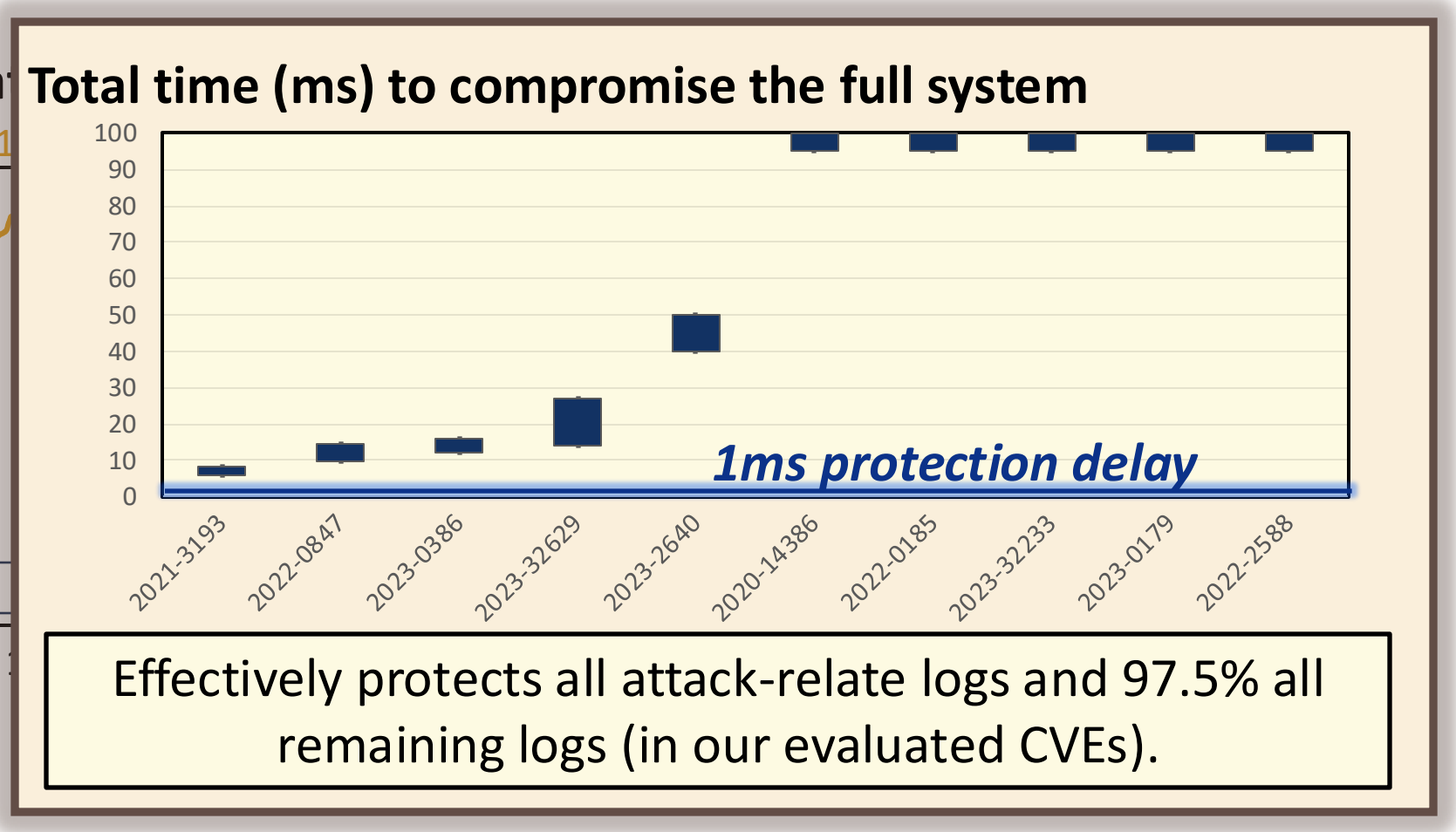
16/18 TrustZone issues were mitigated.

Effectiveness on controlled short log protection



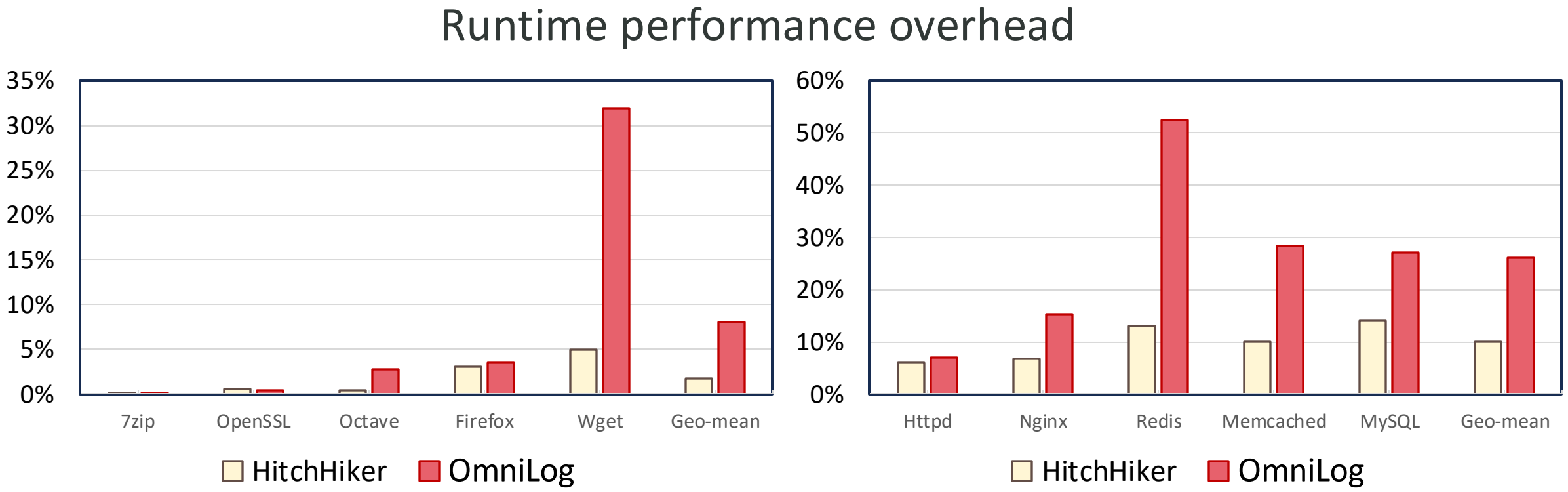
Effectiveness on controlled short log protection





Effectively protects all attack-relate logs and 97.5% all remaining logs (in our evaluated CVEs).

Performance efficiency on real-world applications



Both are efficient in systems with low log throughput.

61.9% lower overhead on high-throughput systems.

Takeaways

The state of the observability protection systems

- Low assurance in-memory environments and inefficiency under stress.

HitchHiker's design principles for observability protection

- Principled debloating strategies to designing high-assurance environments.
- Controlled protection delay using bounded timer and fast permission switches.

Code will (soon) be available at:

THANKS!