Rethinking System Audit Architectures for High Event Coverage and Synchronous Log Availability

Varun Gandhi, Sarbartha Banerjee, Aniket Agarwal,
Adil Ahmed, Sangho Lee, and Marcus Peinado

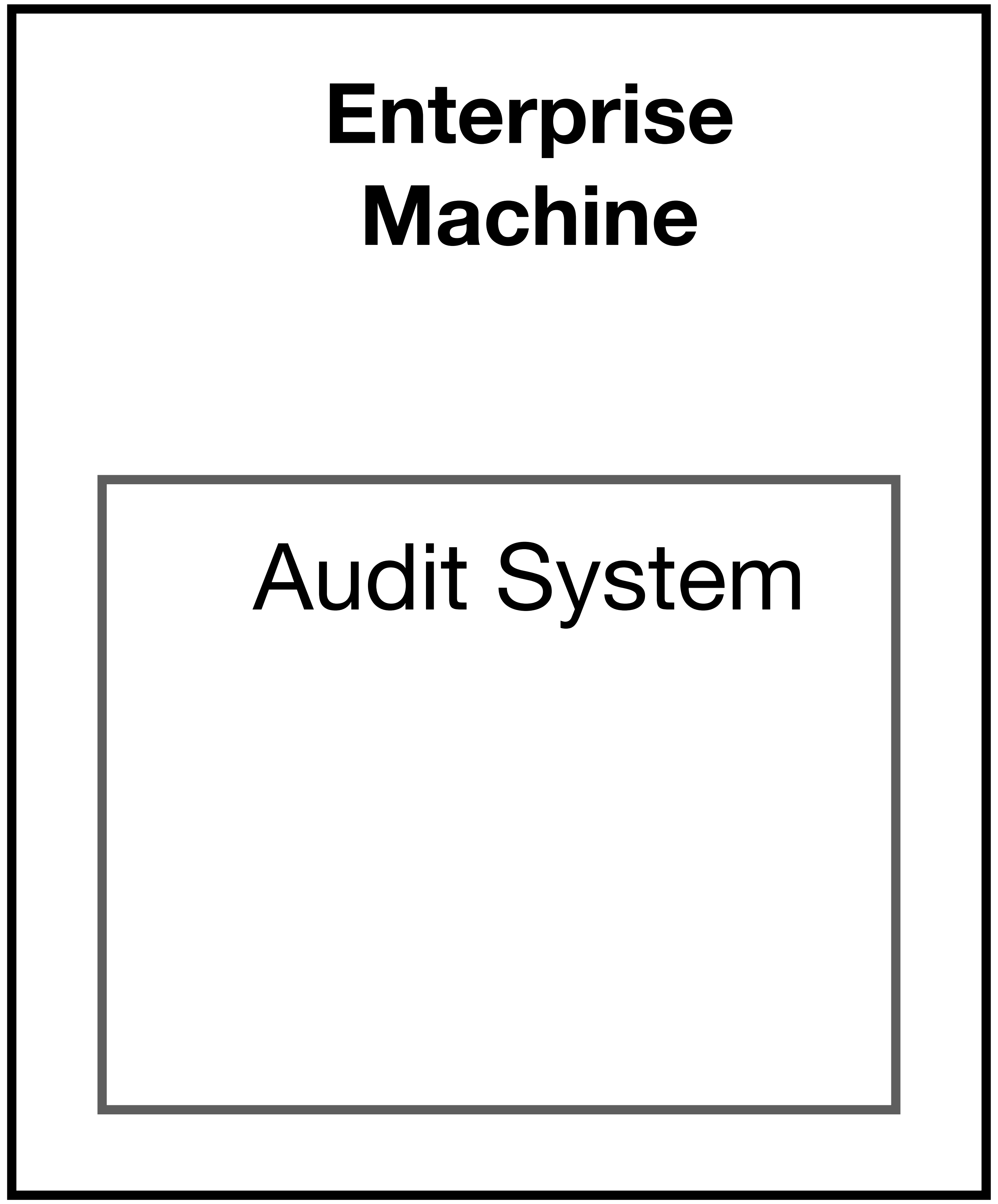
HARVARD UNIVERSITY

THE UNIVERSITY OF TEXAS AT AUSTIN

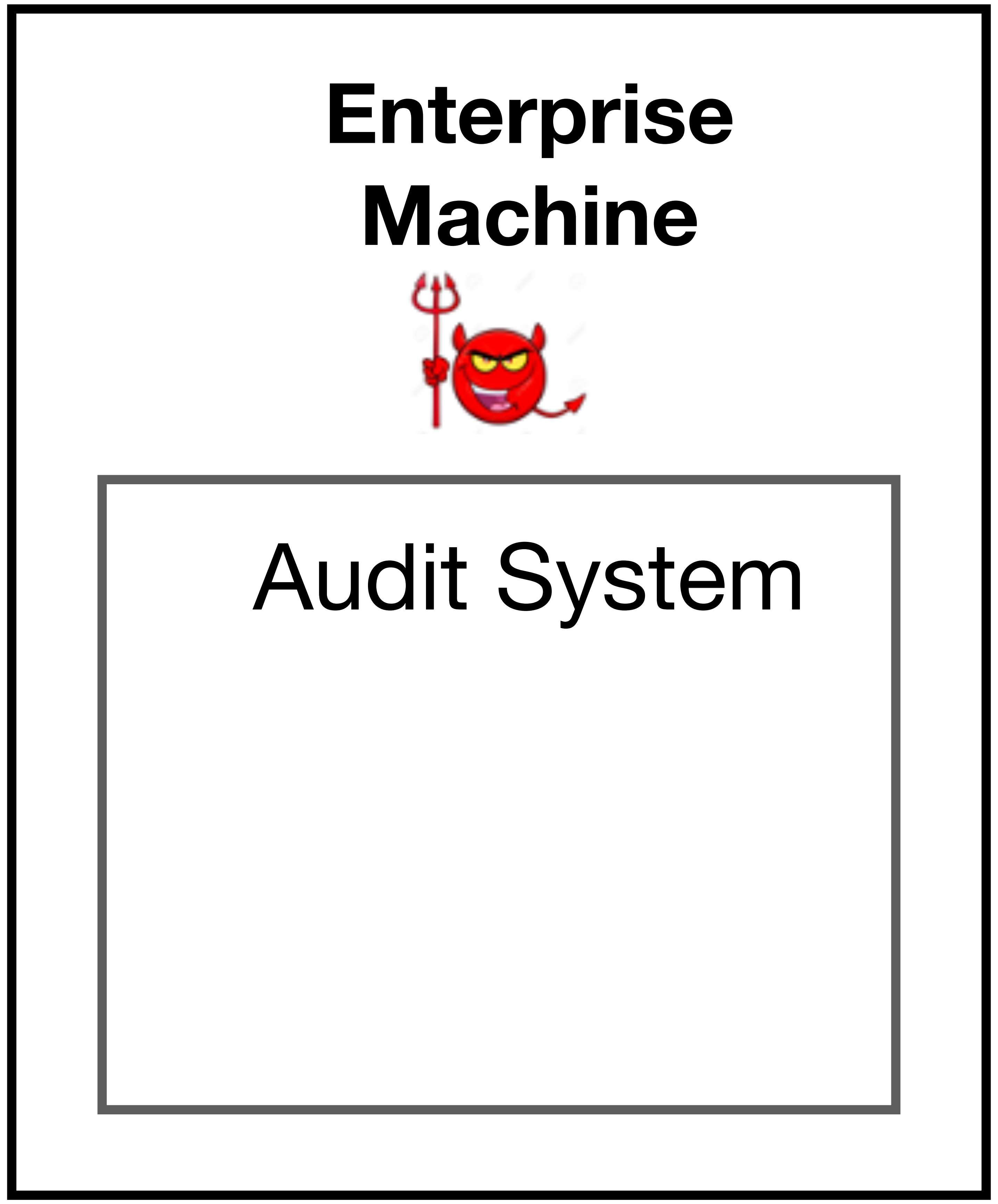
Arizona State University

Microsoft Research

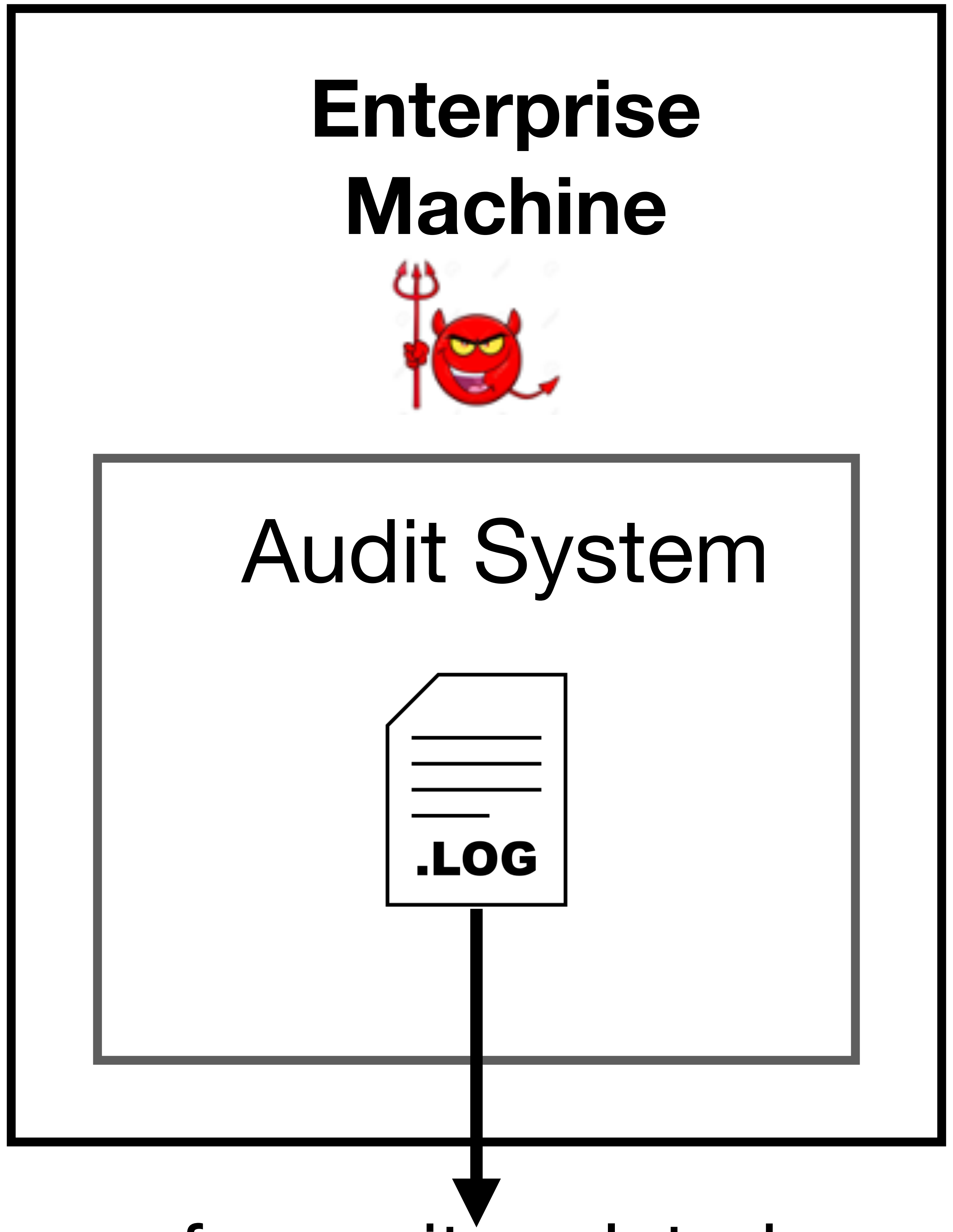
System auditing is critical for forensic analysis



System auditing is critical for forensic analysis

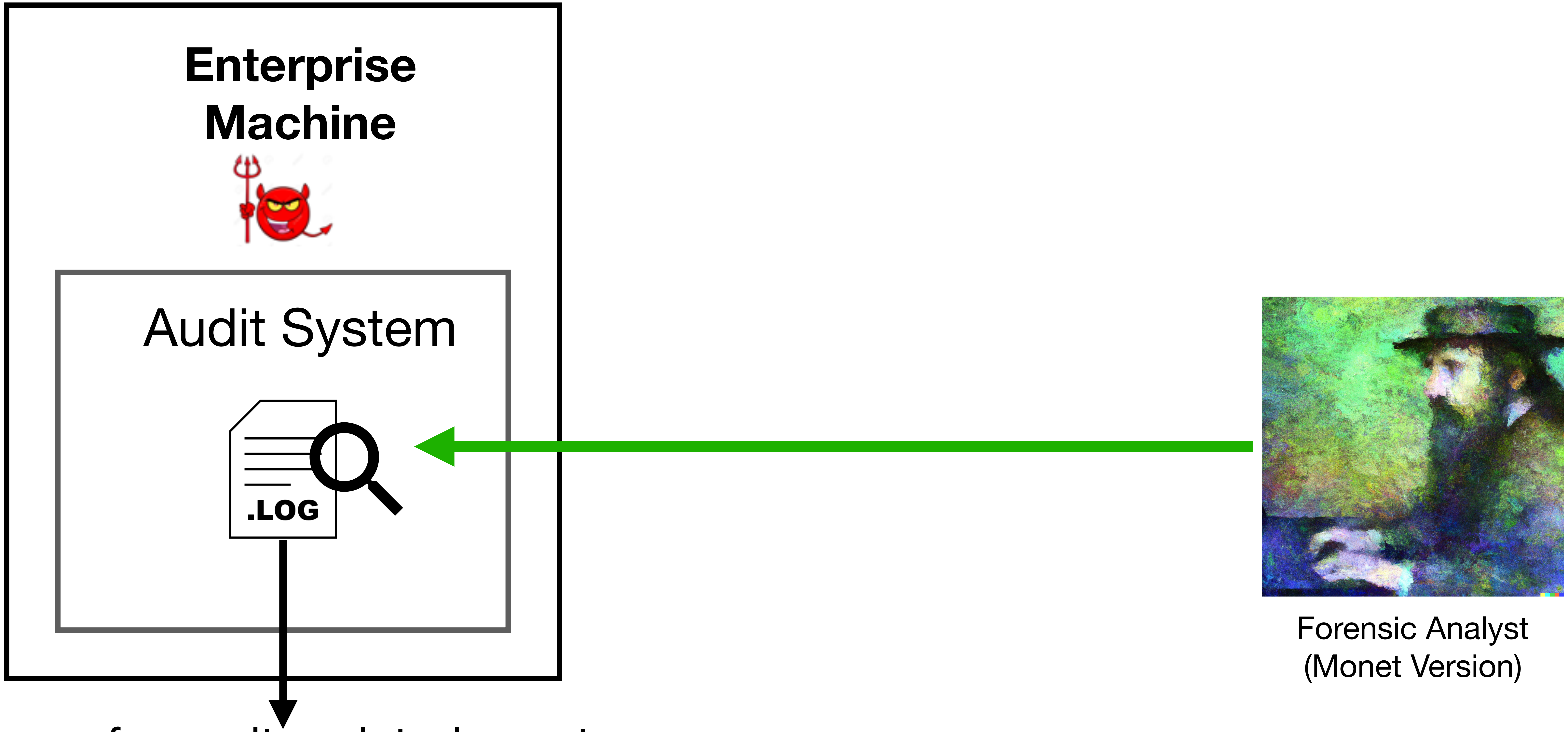


System auditing is critical for forensic analysis



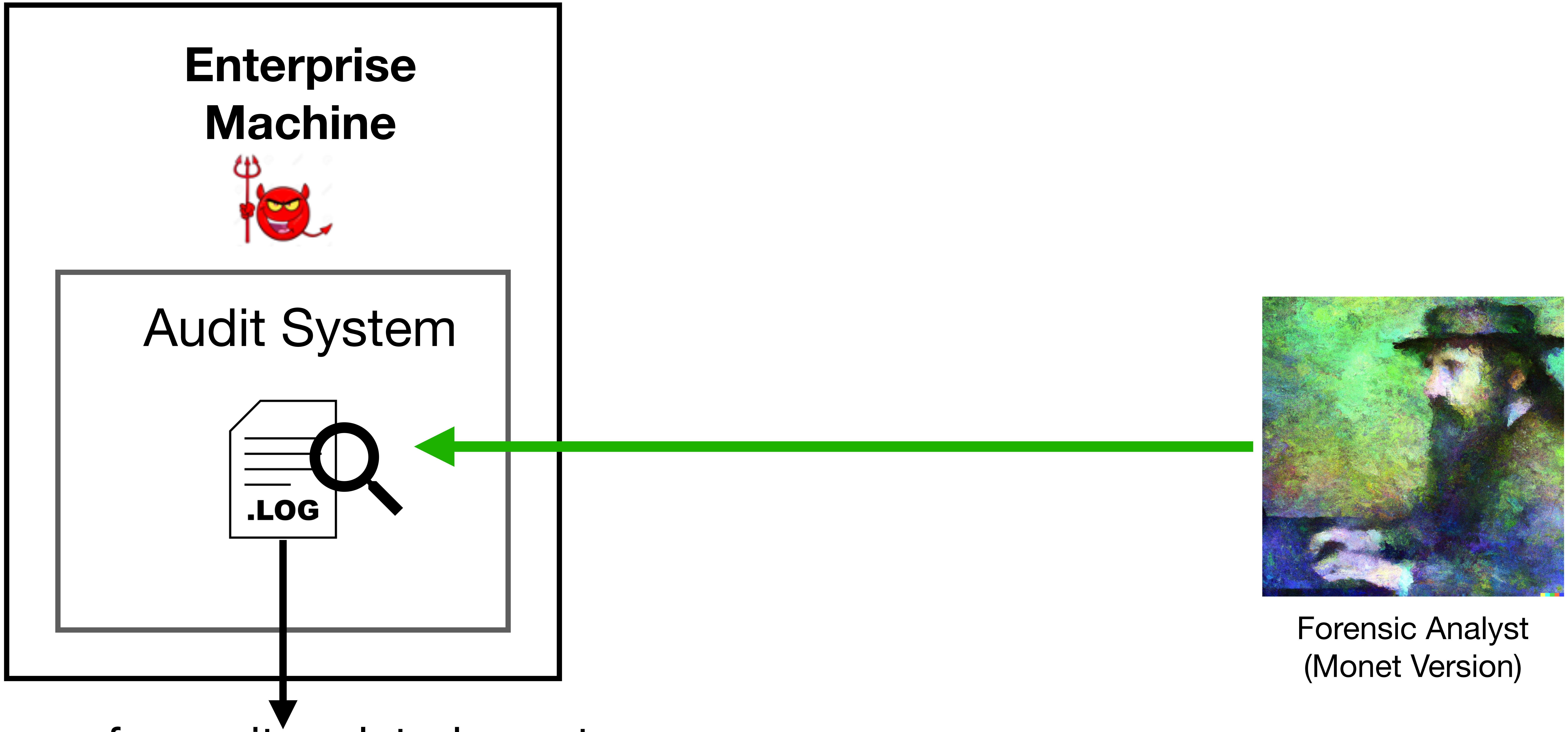
Trace of security-related events
E.g., system calls

System auditing is critical for forensic analysis



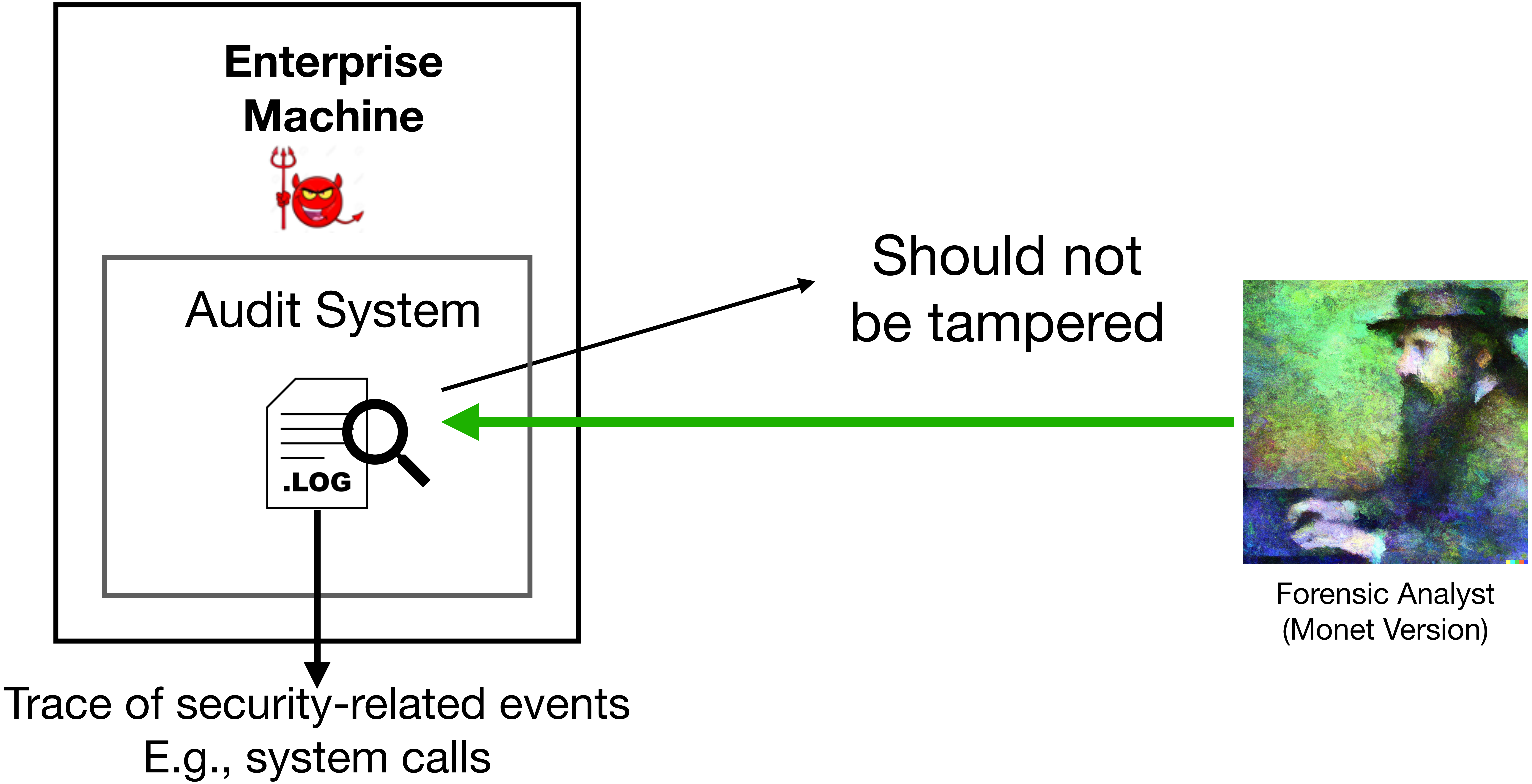
Trace of security-related events
E.g., system calls

What makes a log effective for forensic analysis?

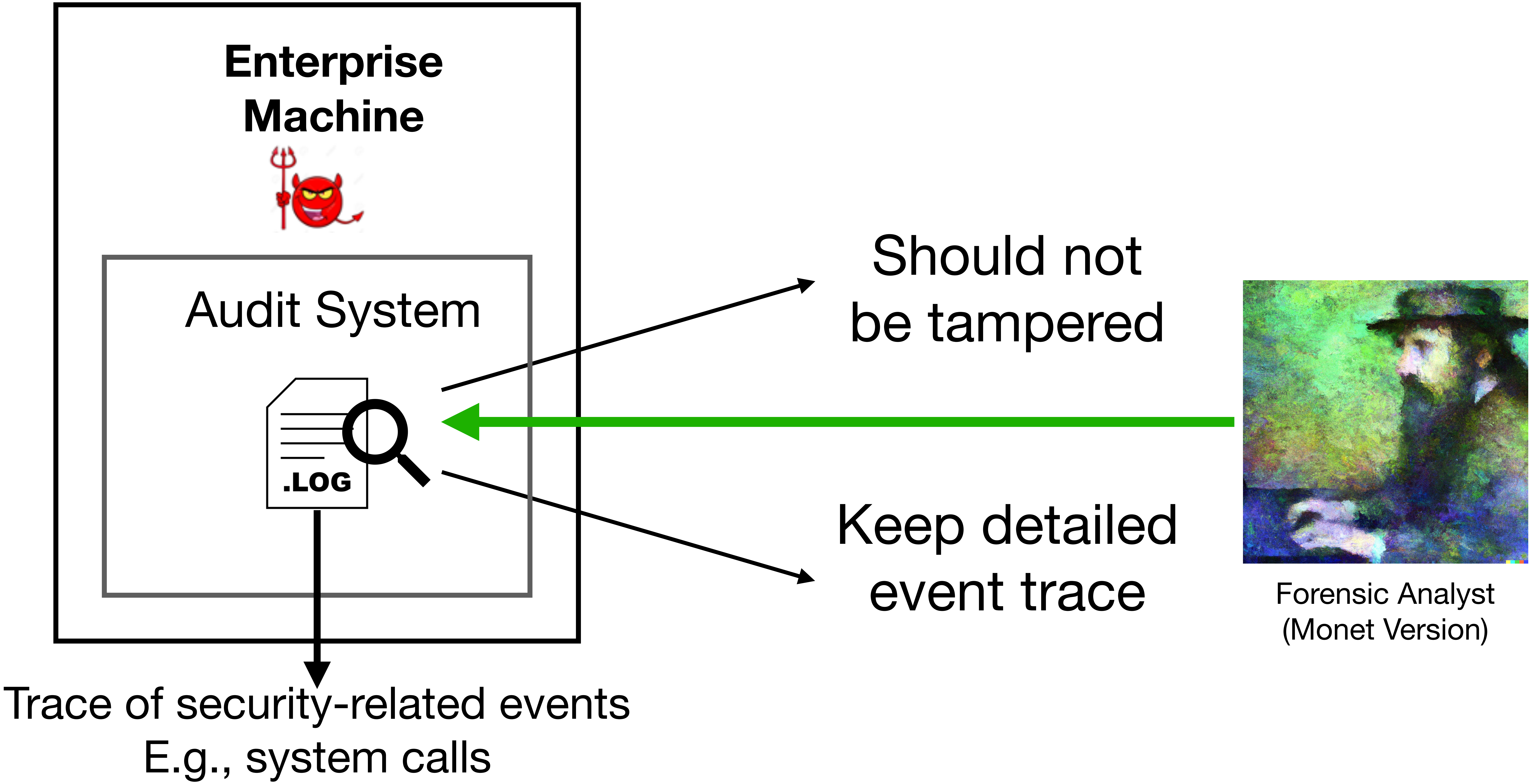


Trace of security-related events
E.g., system calls

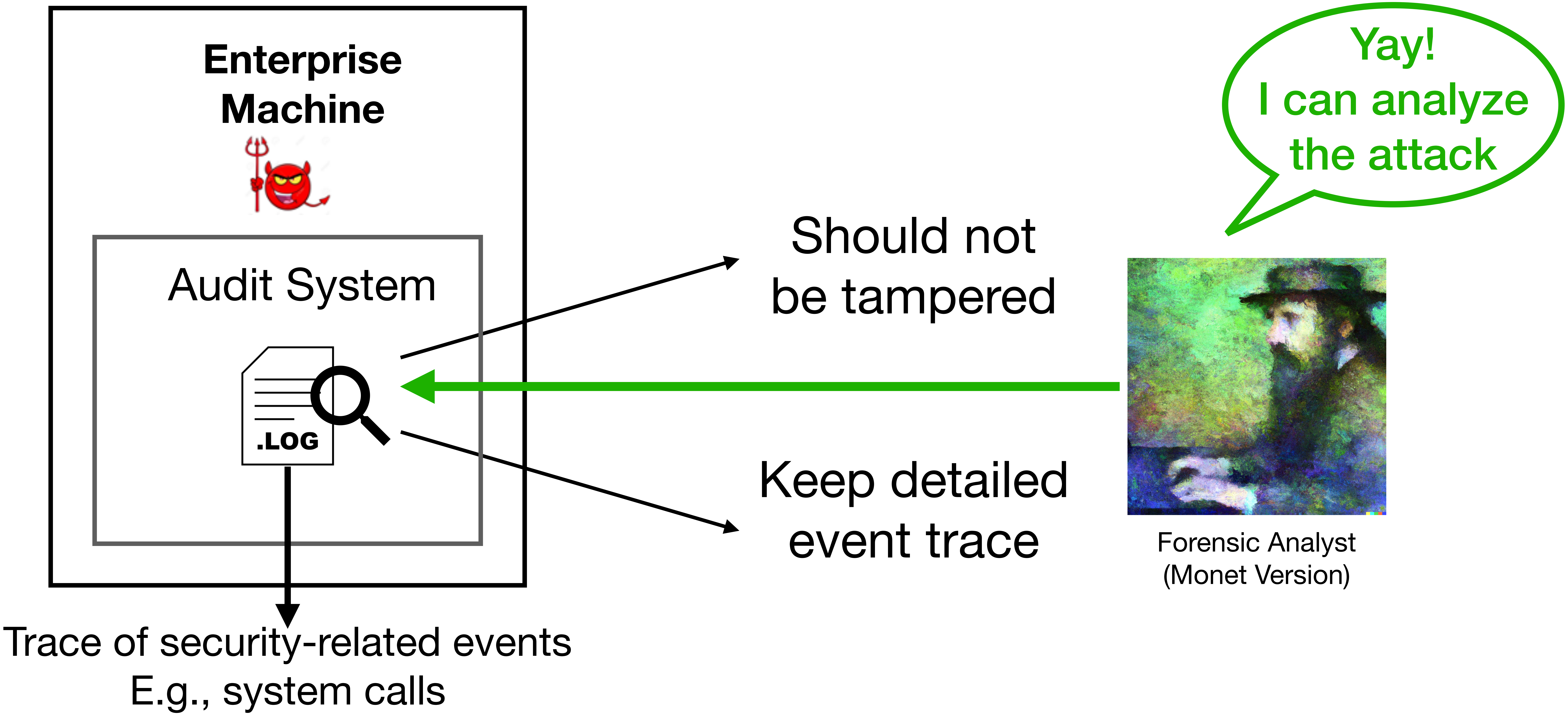
What makes a log effective for forensic analysis?



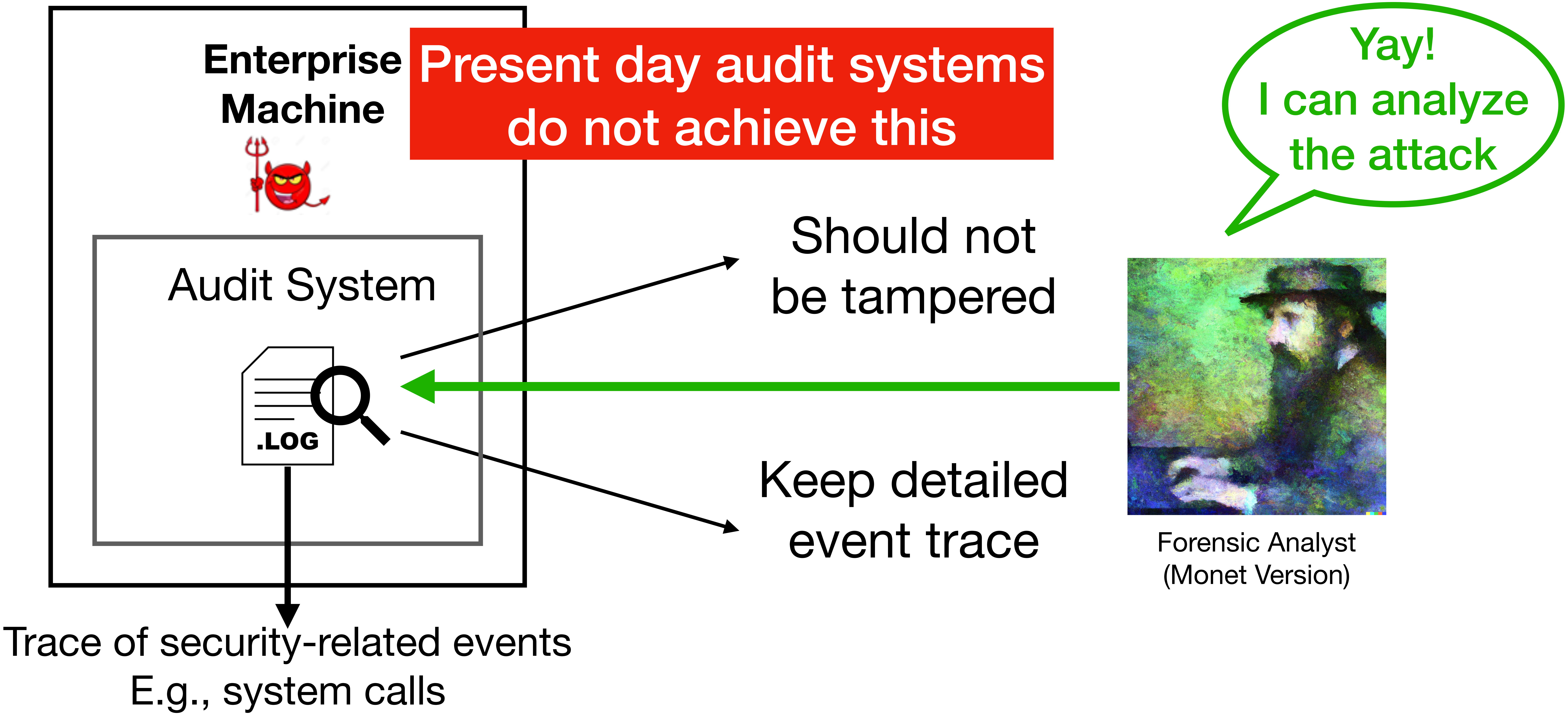
What makes a log effective for forensic analysis?



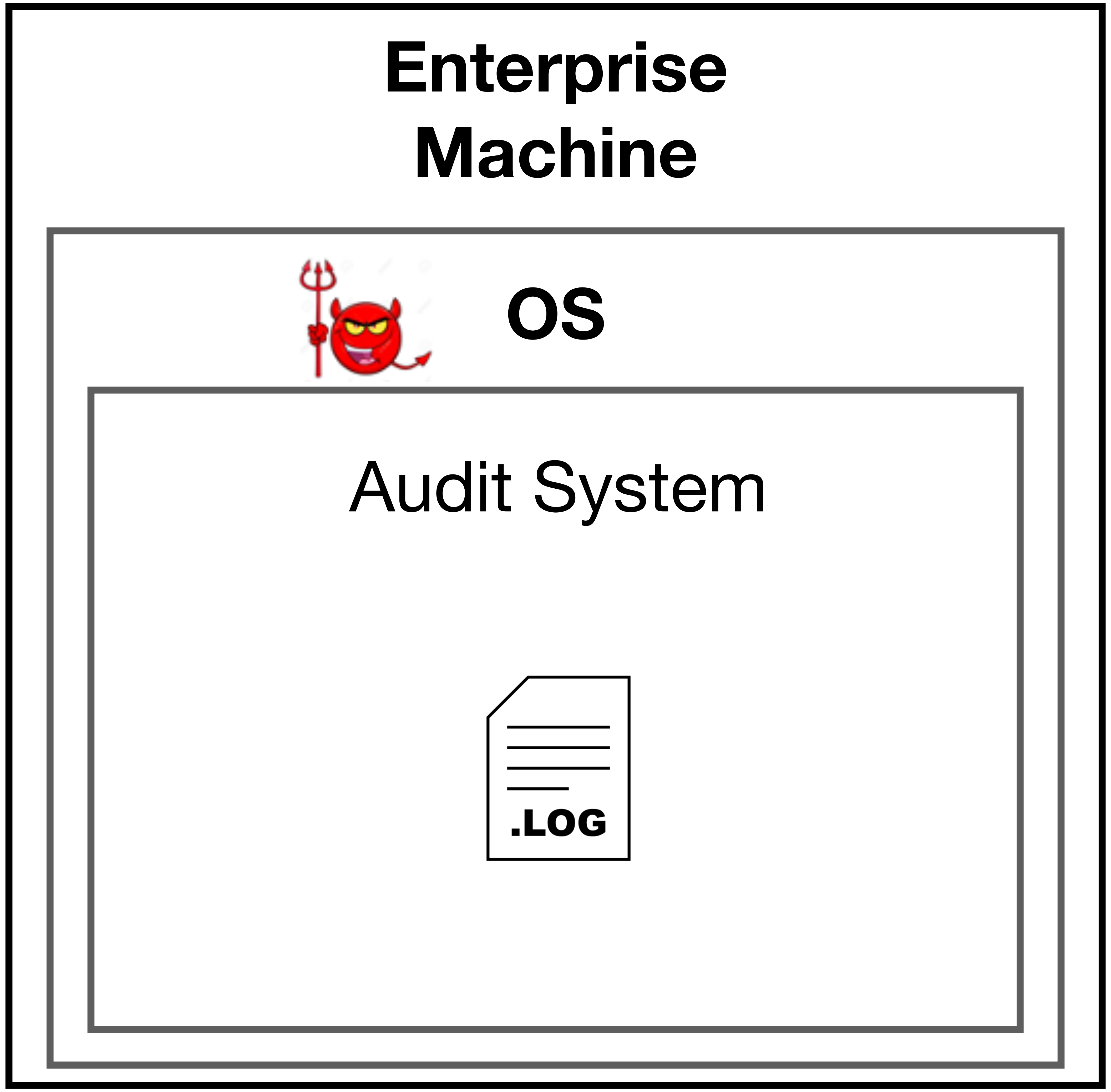
What makes a log effective for forensic analysis?



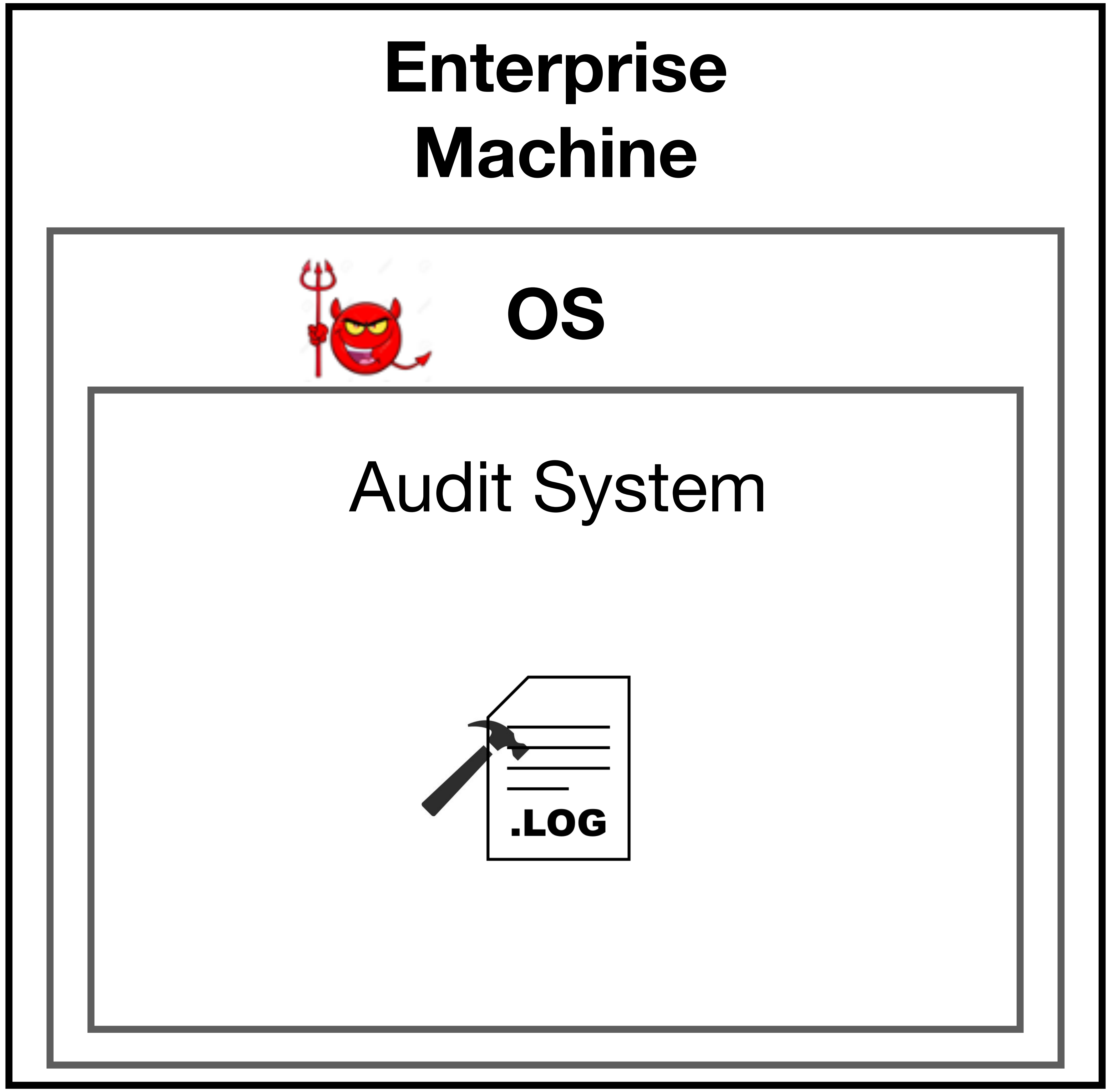
What makes a log effective for forensic analysis?



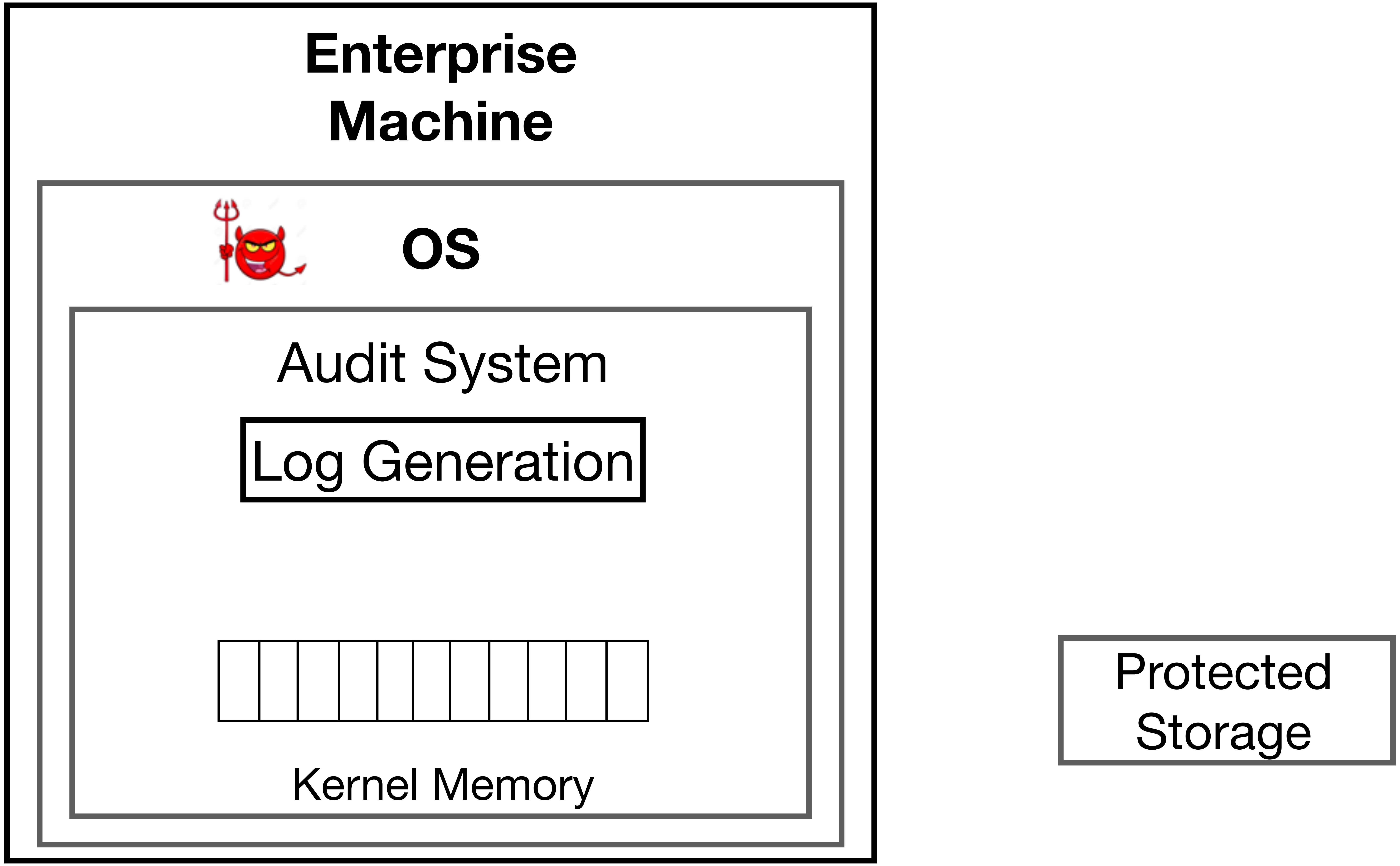
Kernel exploits can tamper audit logs



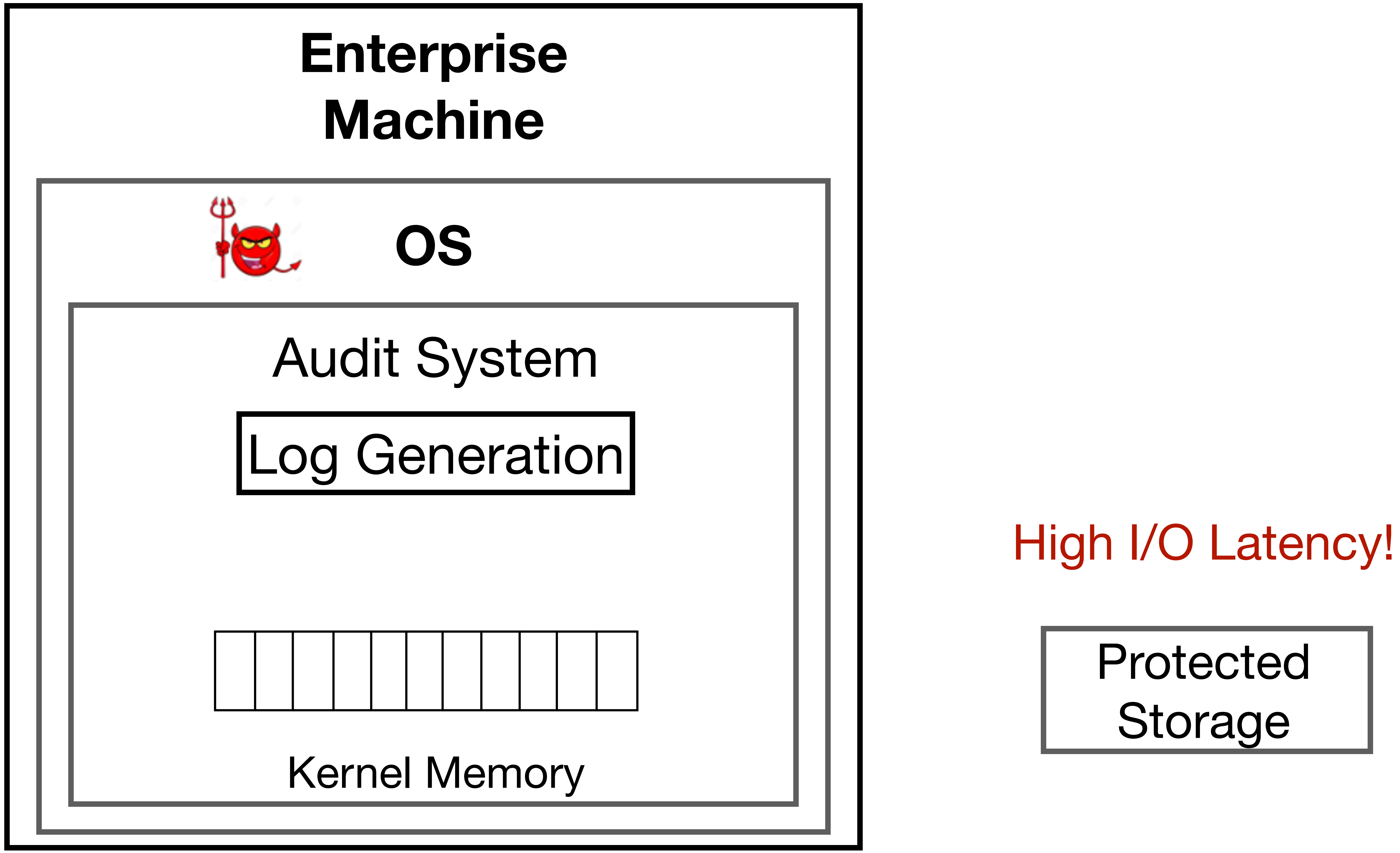
Kernel exploits can tamper audit logs



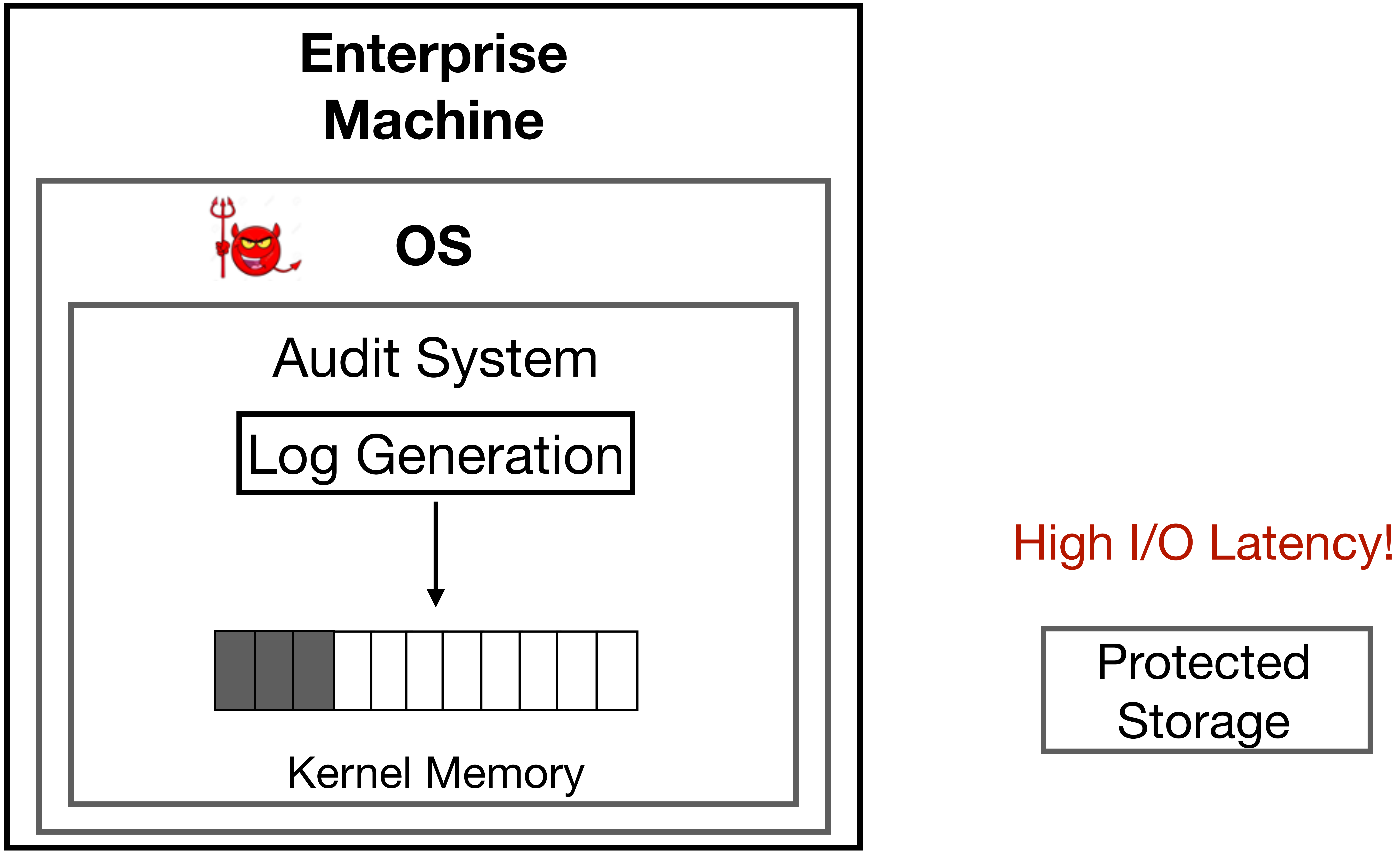
Current systems don't protect all logs up to the compromise



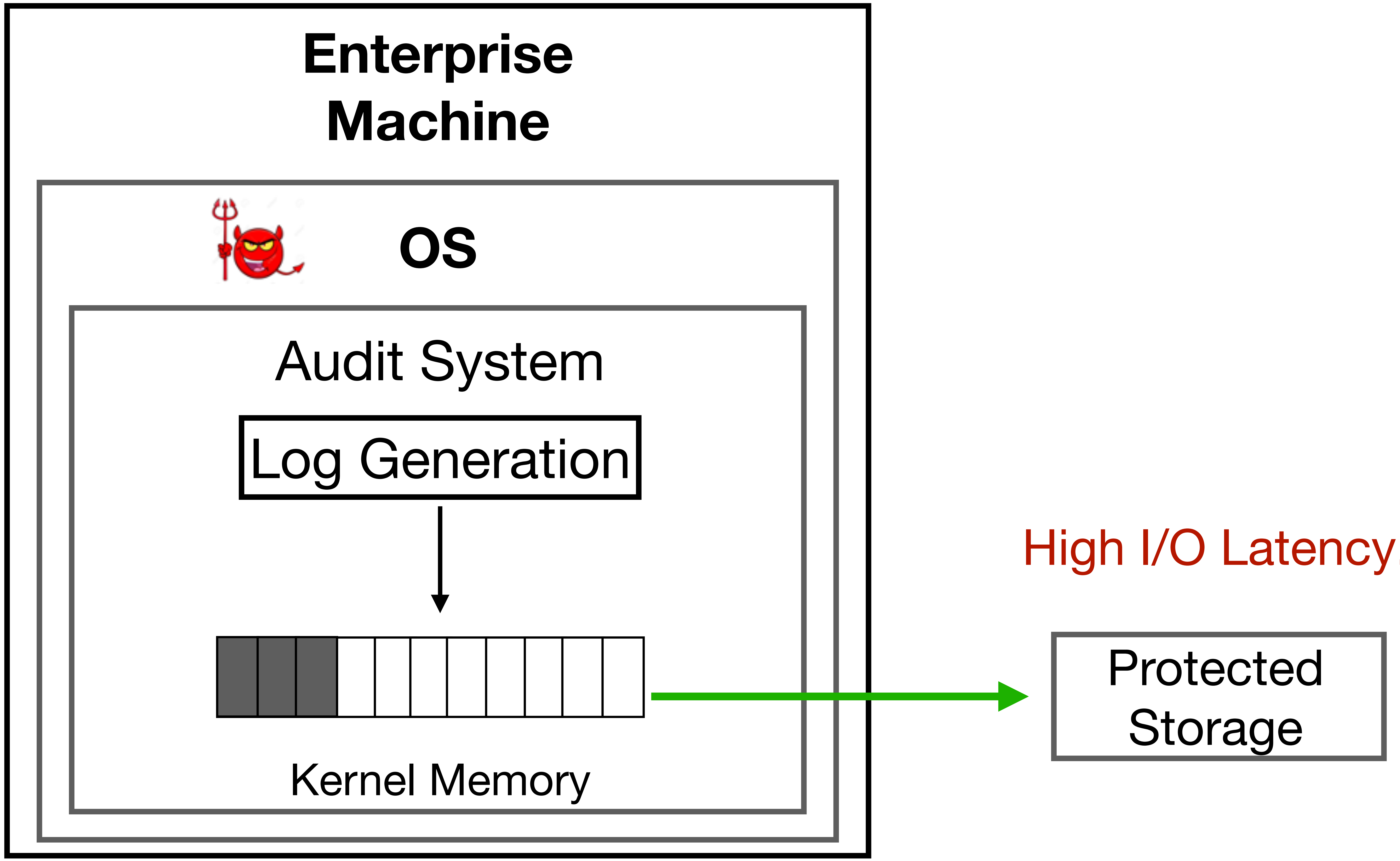
Current systems don’t protect all logs up to the compromise



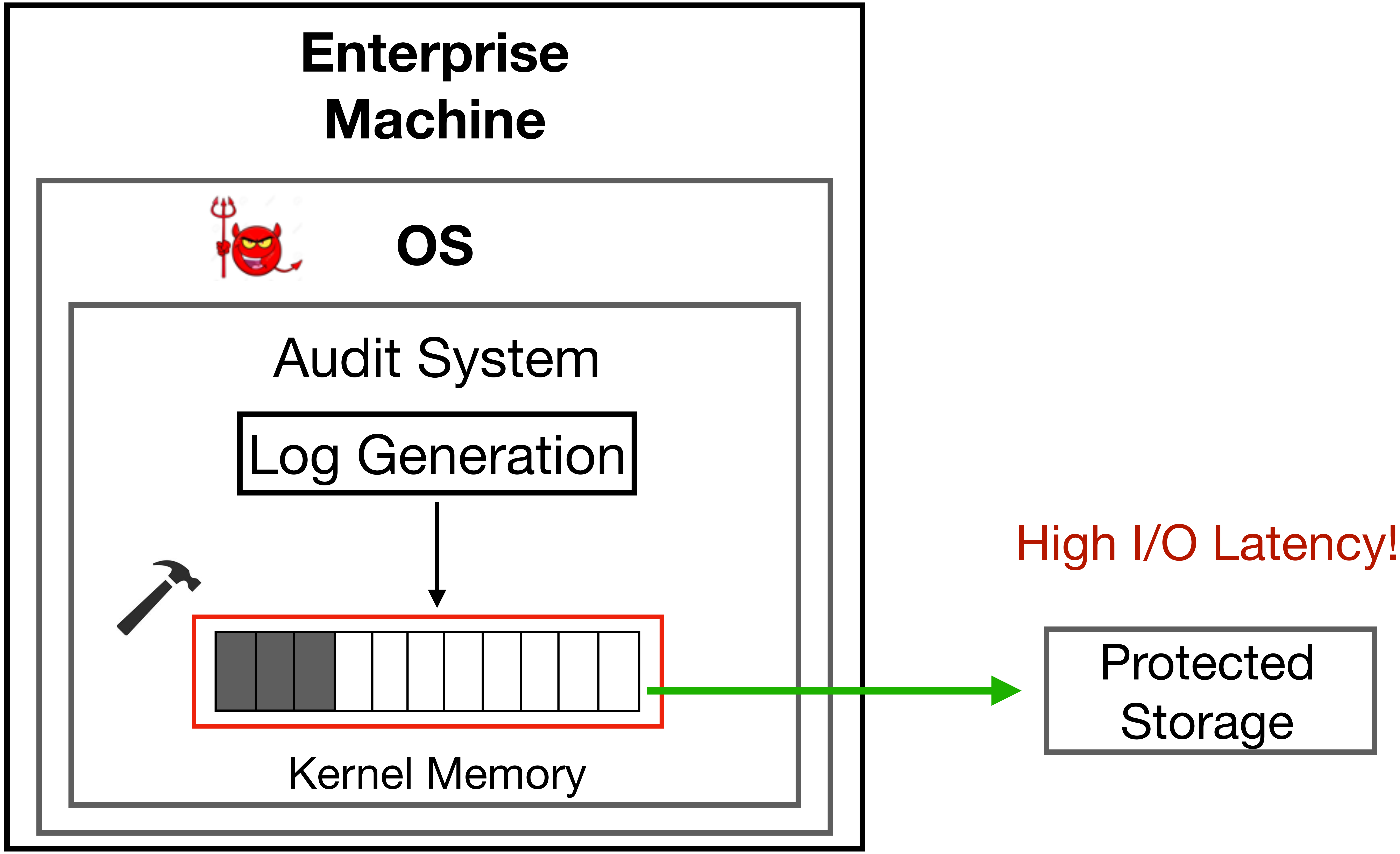
Current systems don't protect all logs up to the compromise



Current systems don't protect all logs up to the compromise



Current systems don't protect all logs up to the compromise



Audit systems also don’t keep a detailed trace of events

Audit systems also don’t keep a detailed trace of events

- Rulesets determine what system call events get logged

Audit systems also don’t keep a detailed trace of events

- Rulesets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia

Audit systems also don’t keep a detailed trace of events

- Rulesets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

Audit systems also don't keep a detailed trace of events

- Rulesets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

	MITRE	NISPOM	STIG	MSD	Academic
Average	14.7%	12.4%	9.8%	10.2%	15.4%
0% Coverage	60/164	77/164	77/164	80/164	11/164

Audit systems also don't keep a detailed trace of events

- Rulesets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

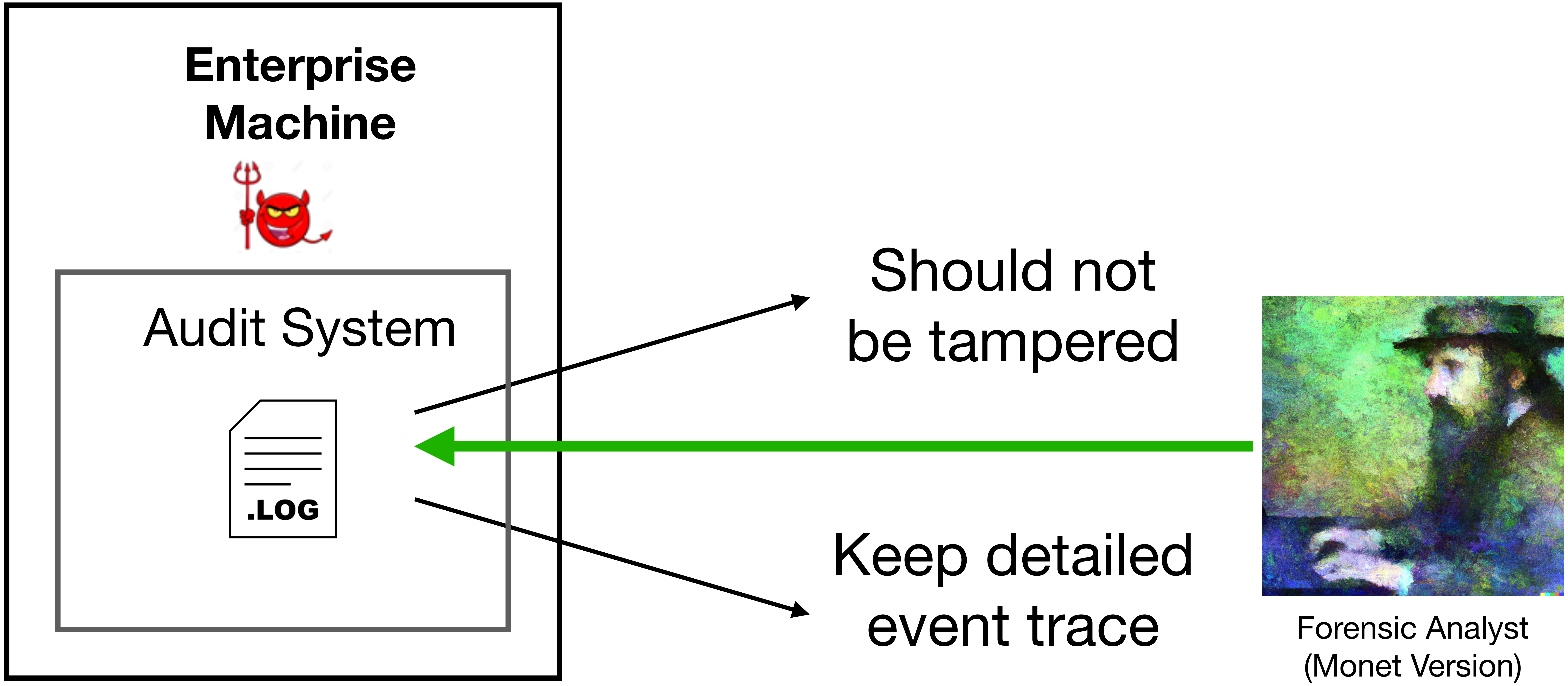
	MITRE	NISPOM	STIG	MSD	Academic
Average	14.7%	12.4%	9.8%	10.2%	15.4%
0% Coverage	60/164	77/164	77/164	80/164	11/164

Audit systems also don't keep a detailed trace of events

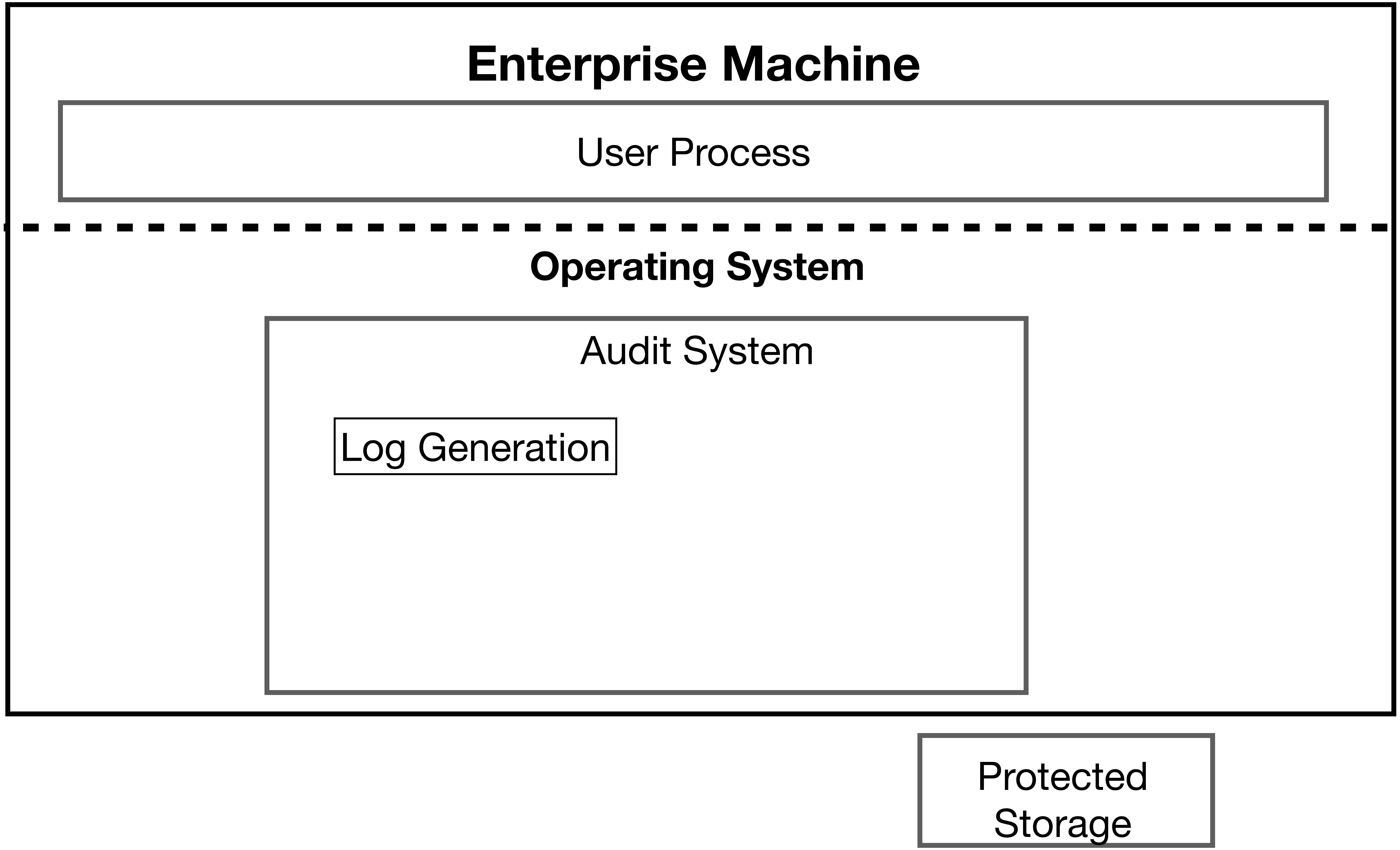
- Rulesets determine what system call events get logged
- A set of rulesets exist as standards across government, industry, and academia
- Analyzed 164 PoC kernel exploits on a subset of standard rule sets

	MITRE	NISPOM	STIG	MSD	Academic
Average	14.7%	12.4%	9.8%	10.2%	15.4%
0% Coverage	60/164	77/164	77/164	80/164	11/164

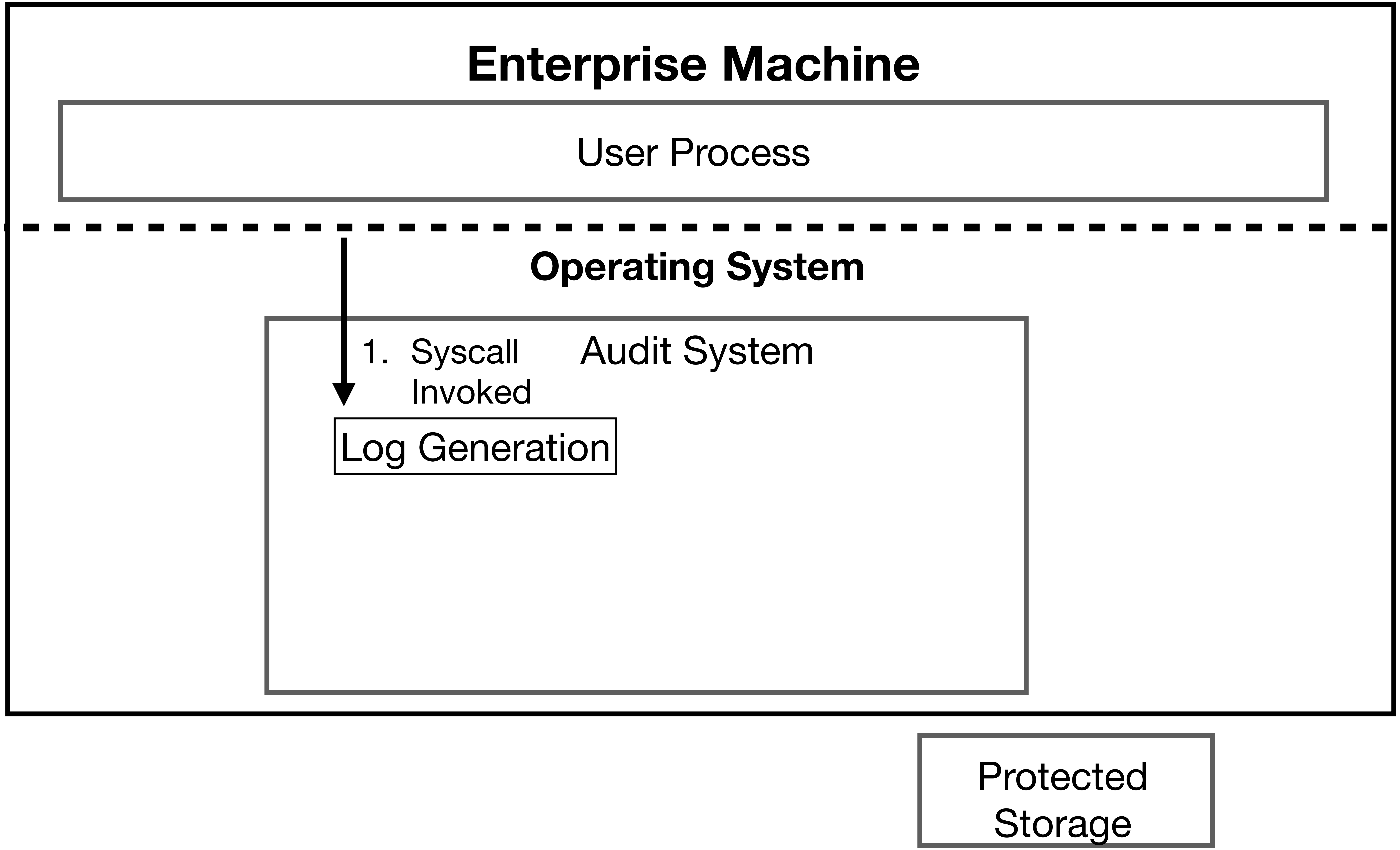
How do we build an audit system with these guarantees?



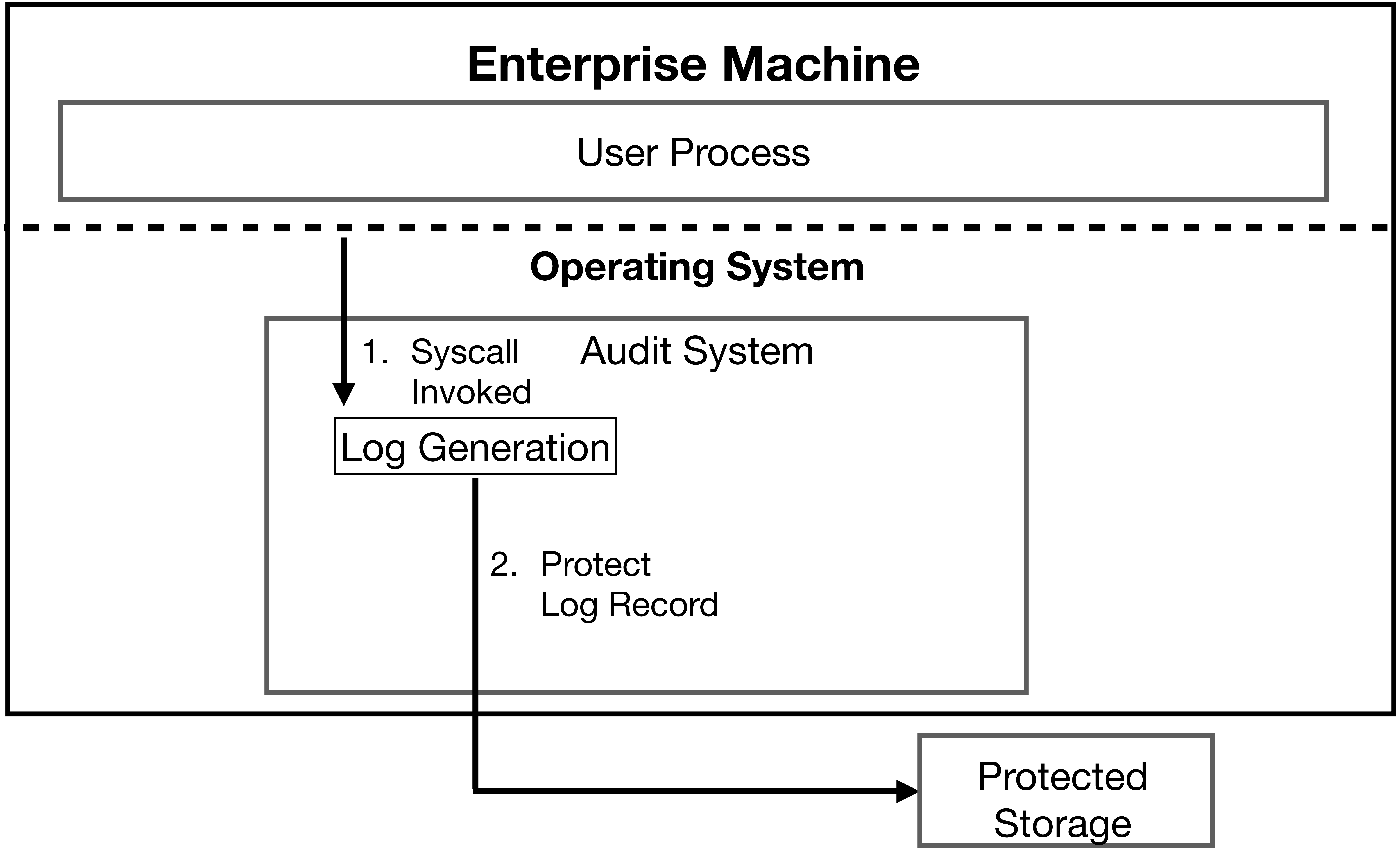
Synchronous logging prevents tampering



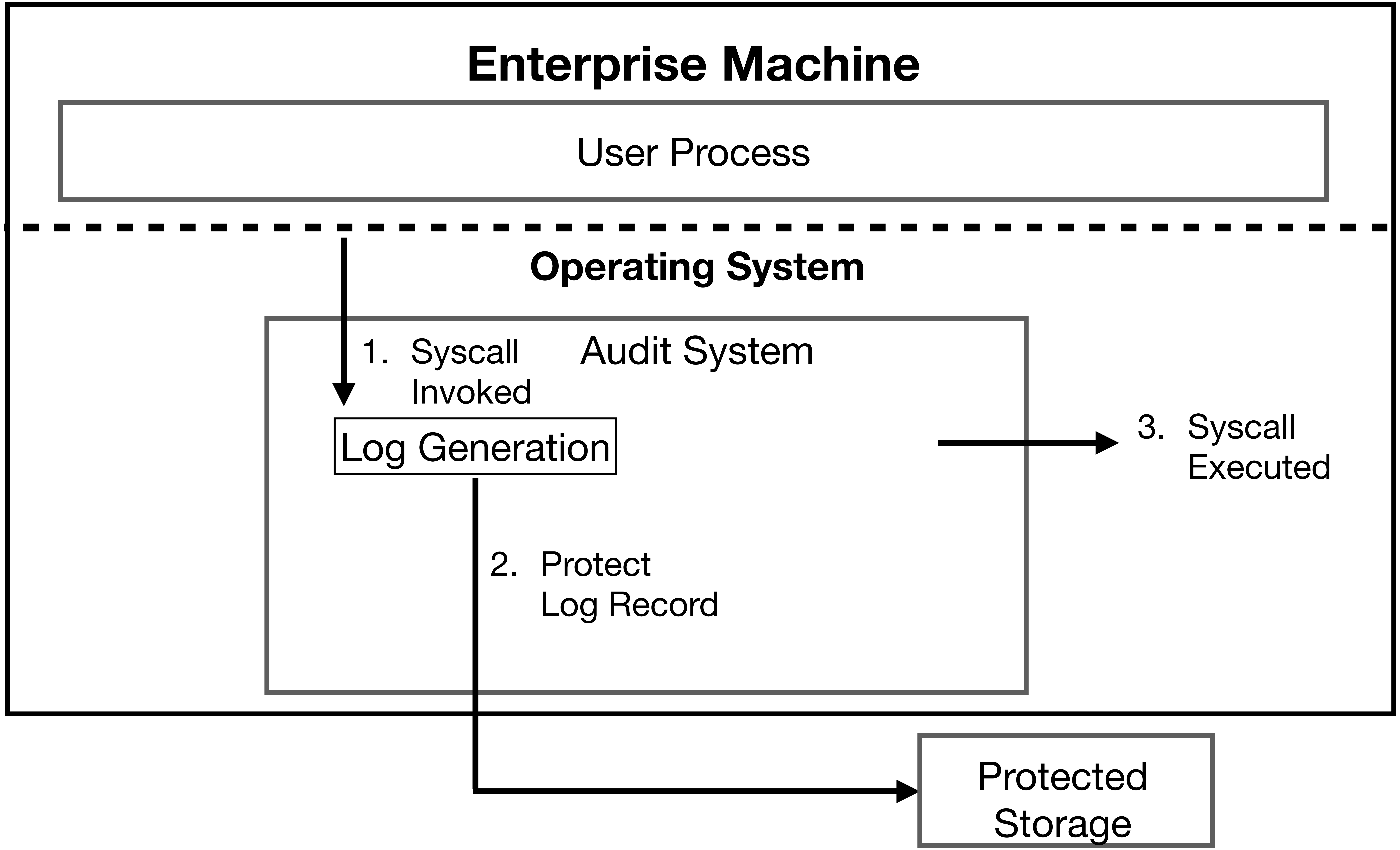
Synchronous logging prevents tampering



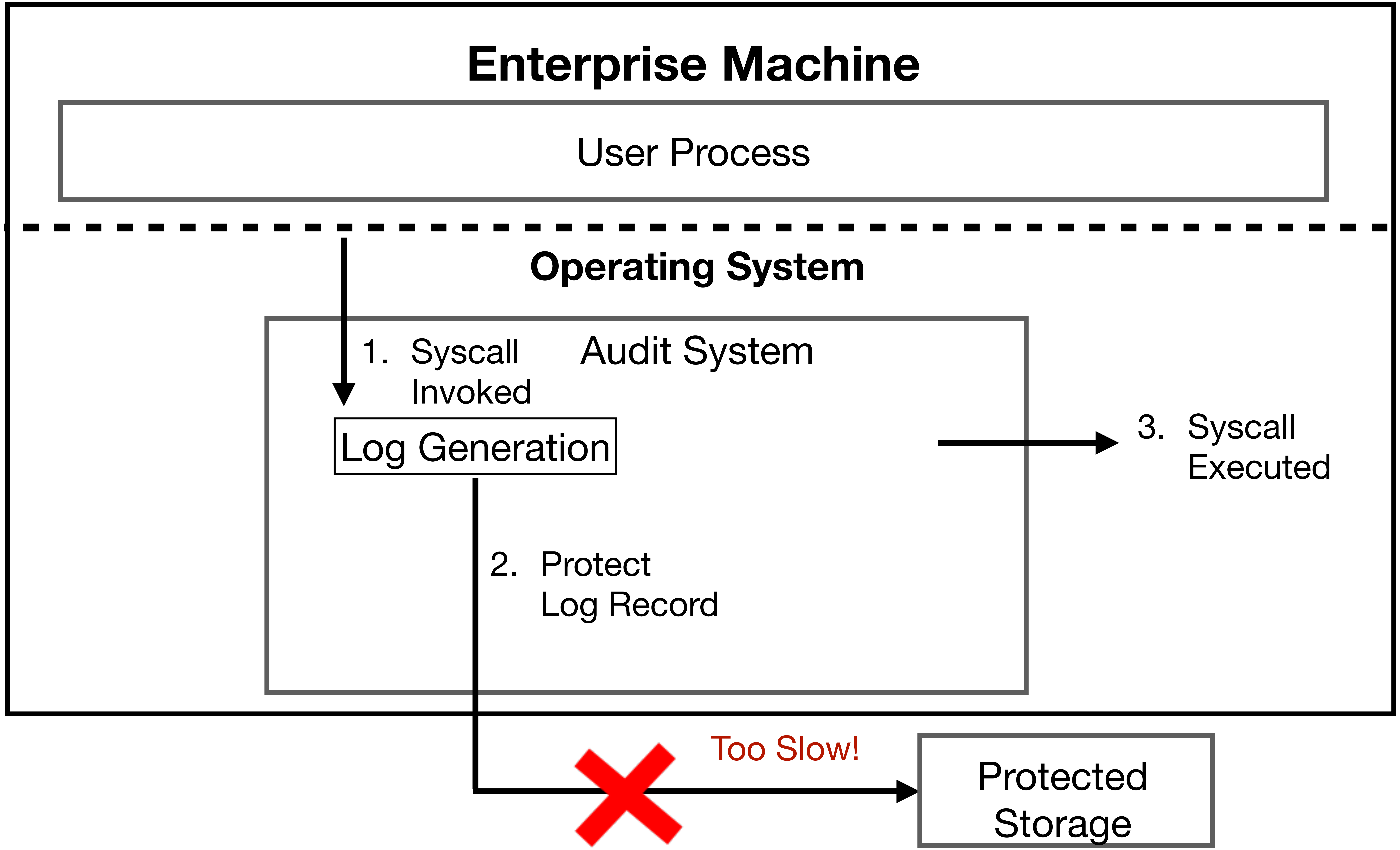
Synchronous logging prevents tampering



Synchronous logging prevents tampering



Synchronous logging prevents tampering



Logging all system calls ensures detailed event tracing

Logging all system calls ensures detailed event tracing

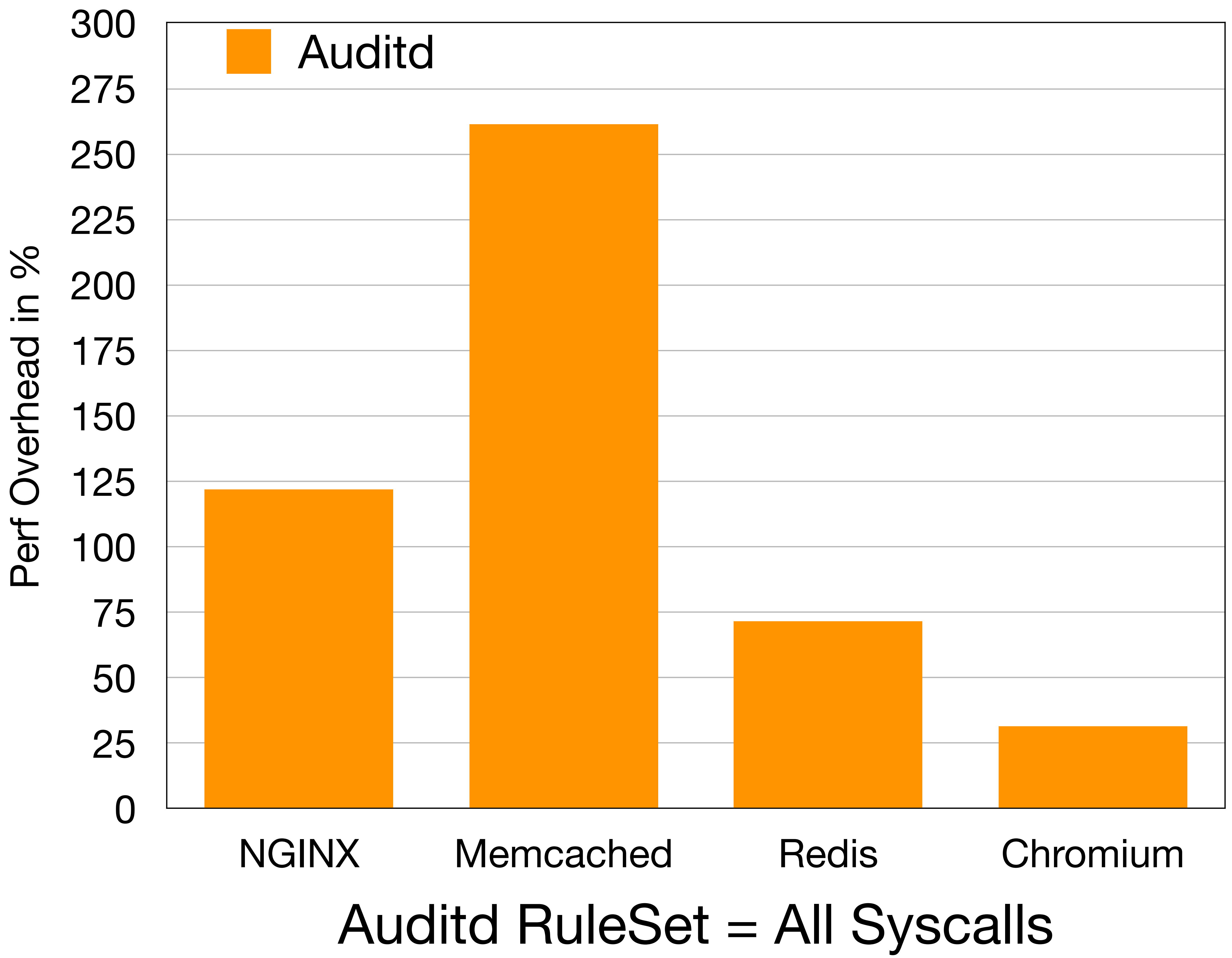
- Kernel exploits leverage a diverse set of system calls

Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
- Logging all system calls guarantees event coverage

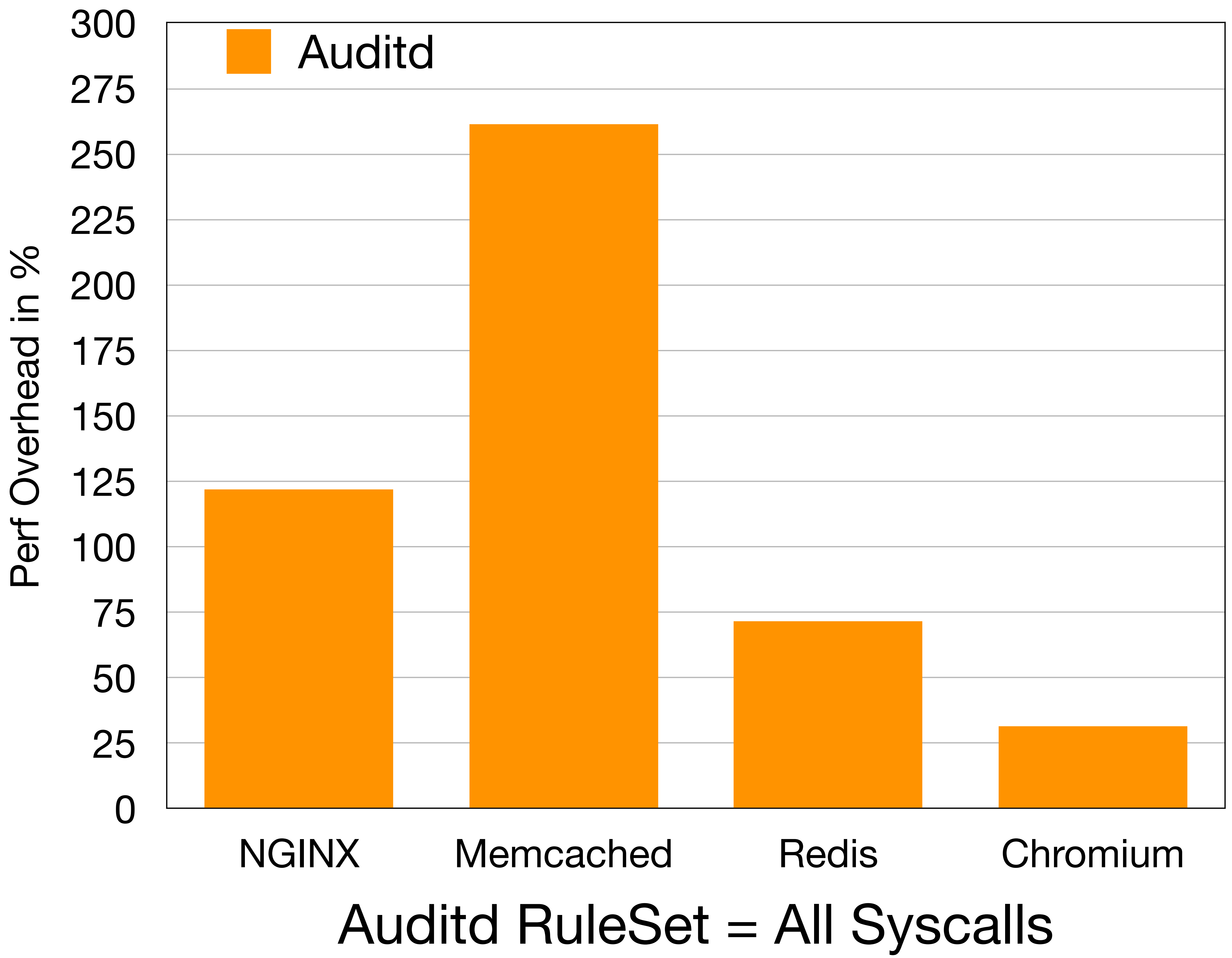
Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
- Logging all system calls guarantees event coverage
- We measured Auditd overhead when logging all system calls on real-world workloads



Logging all system calls ensures detailed event tracing

- Kernel exploits leverage a diverse set of system calls
- Logging all system calls guarantees event coverage
- We measured Auditd overhead when logging all system calls on real-world workloads
- Even for asynchronous logging, the slowdown is prohibitive



OMNILog addresses these efficiency challenges

Challenge 1: High I/O latency for synchronous logging

Challenge 2: Inefficient logging pipeline

OMNILog addresses these efficiency challenges

Challenge 1: High I/O latency for synchronous logging

- Isolating logs in memory within a protected environment and eventually persist

Challenge 2: Inefficient logging pipeline

OMNILog addresses these efficiency challenges

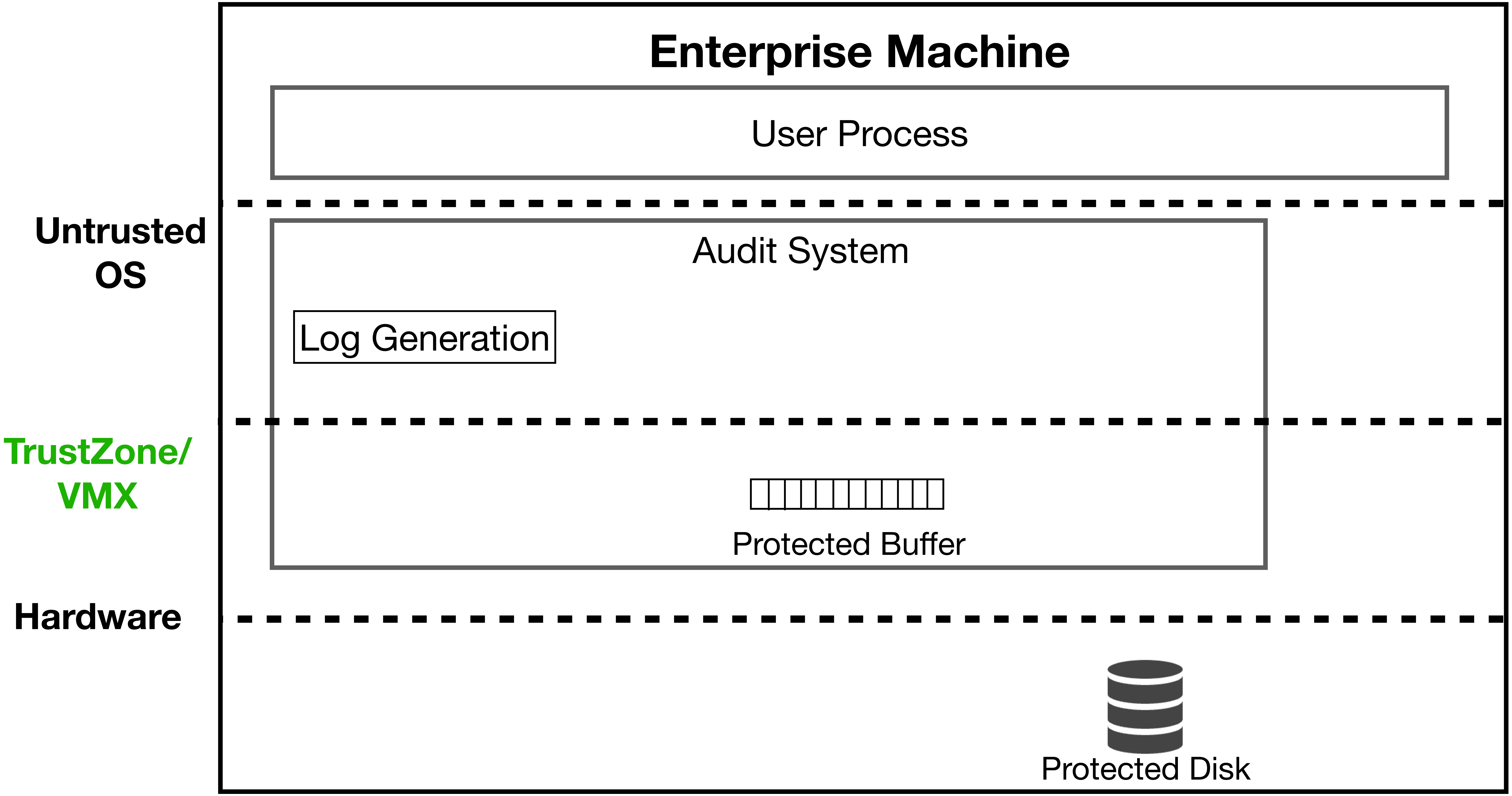
Challenge 1: High I/O latency for synchronous logging

- Isolating logs in memory within a protected environment and eventually persist

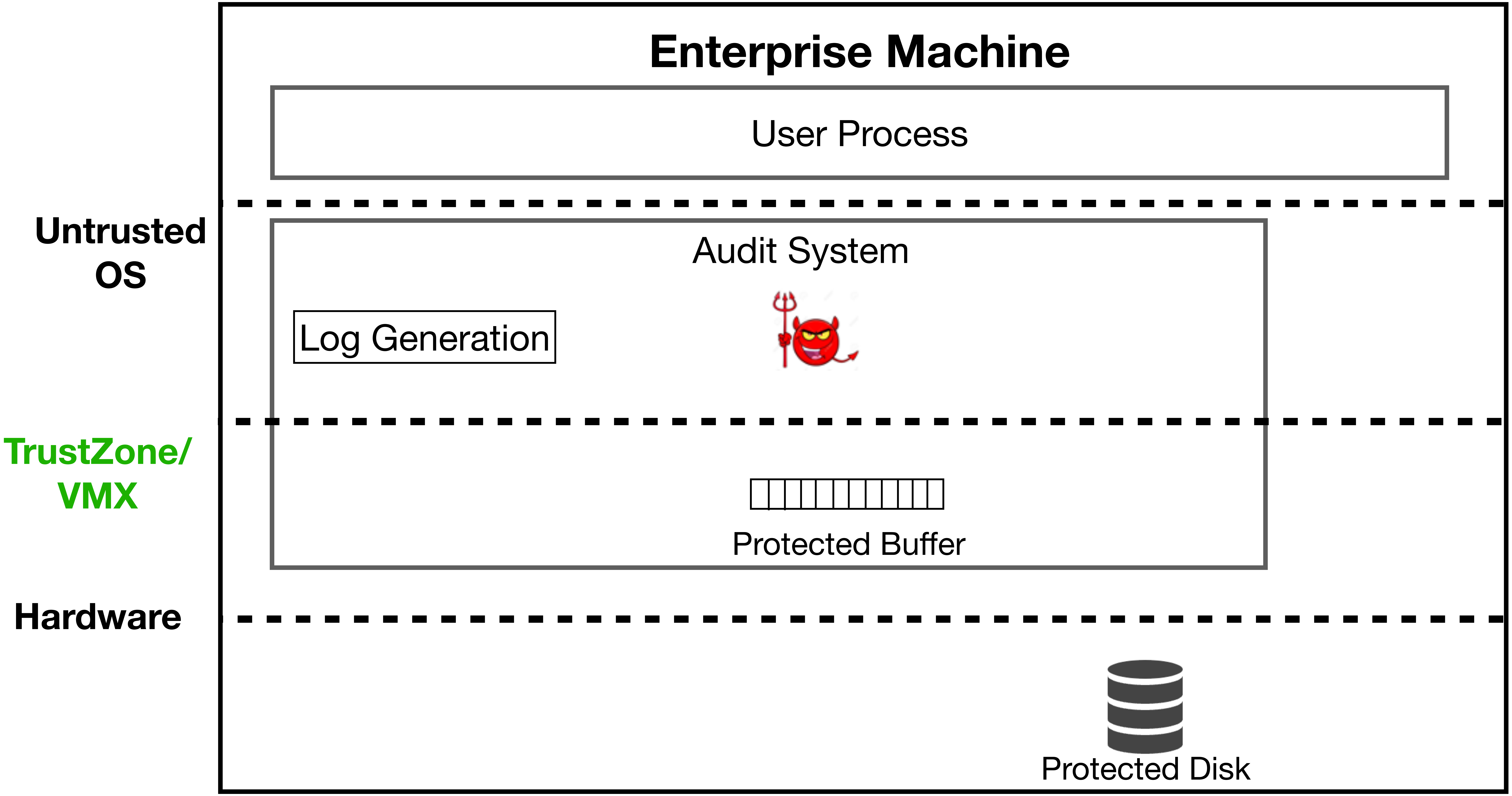
Challenge 2: Inefficient logging pipeline

- Optimizing the end-to-end pipeline from log generation to persistence

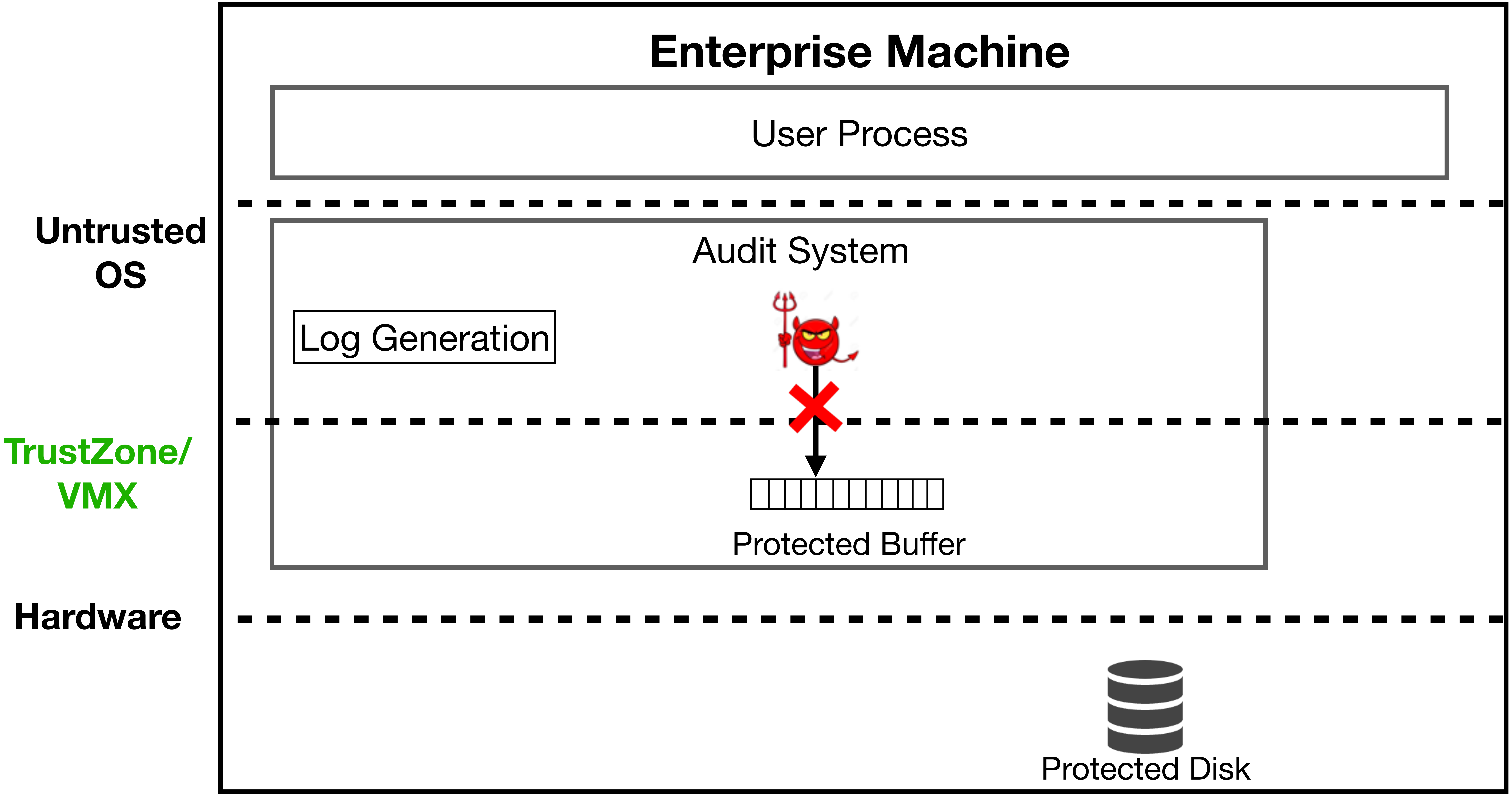
How does OMNILog build a protected environment?



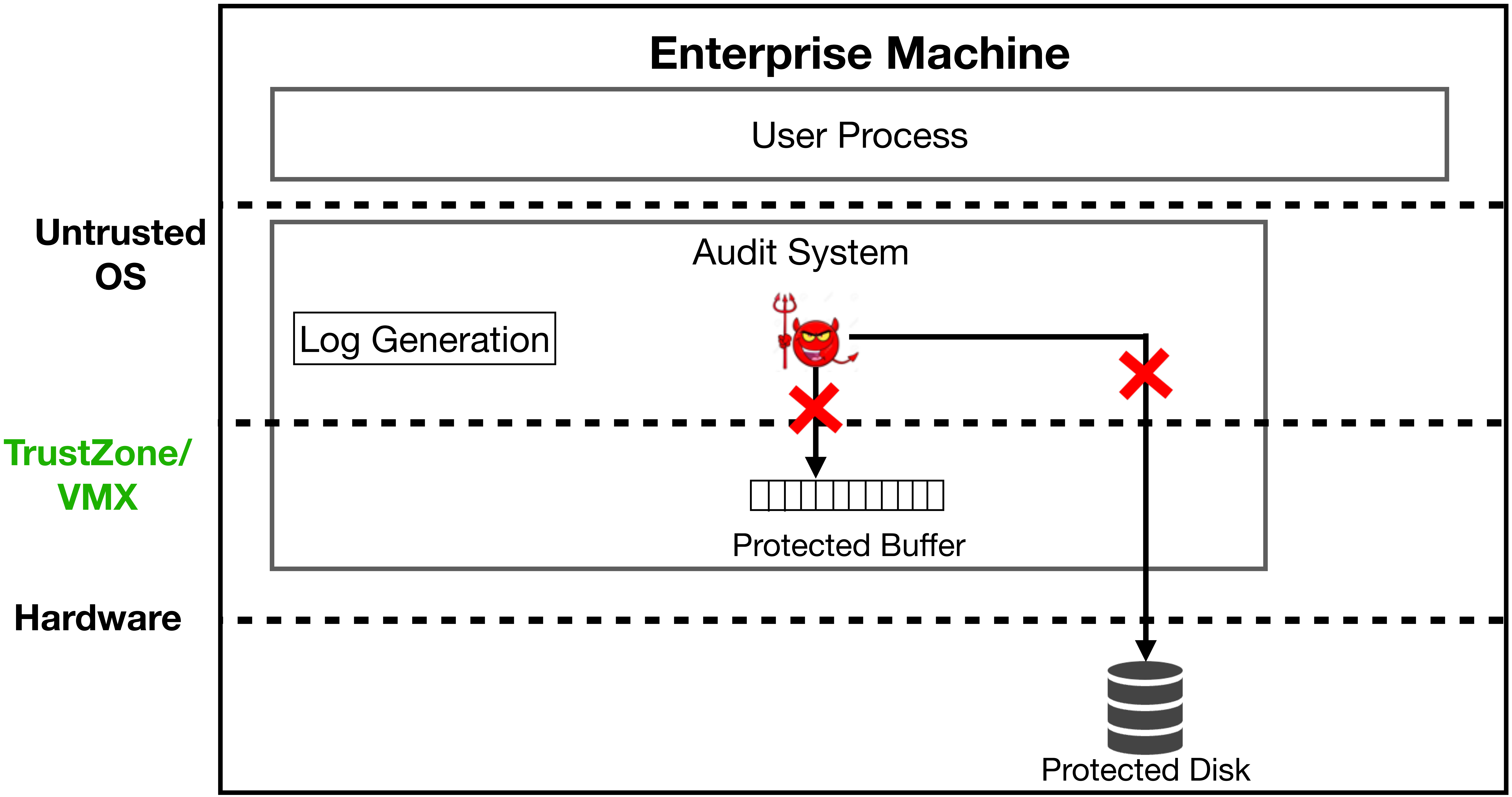
How does OMNILog build a protected environment?



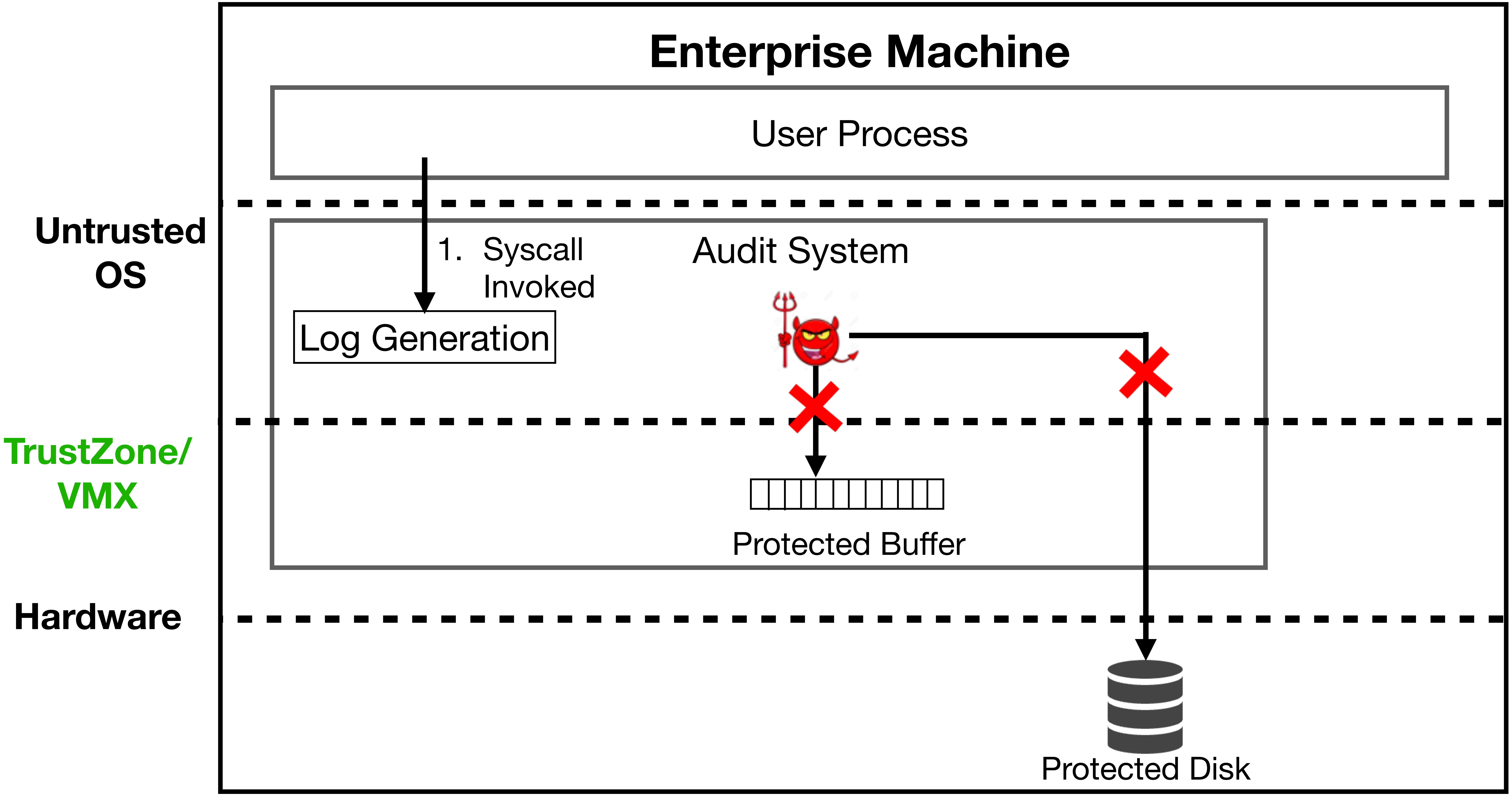
How does OMNILog build a protected environment?



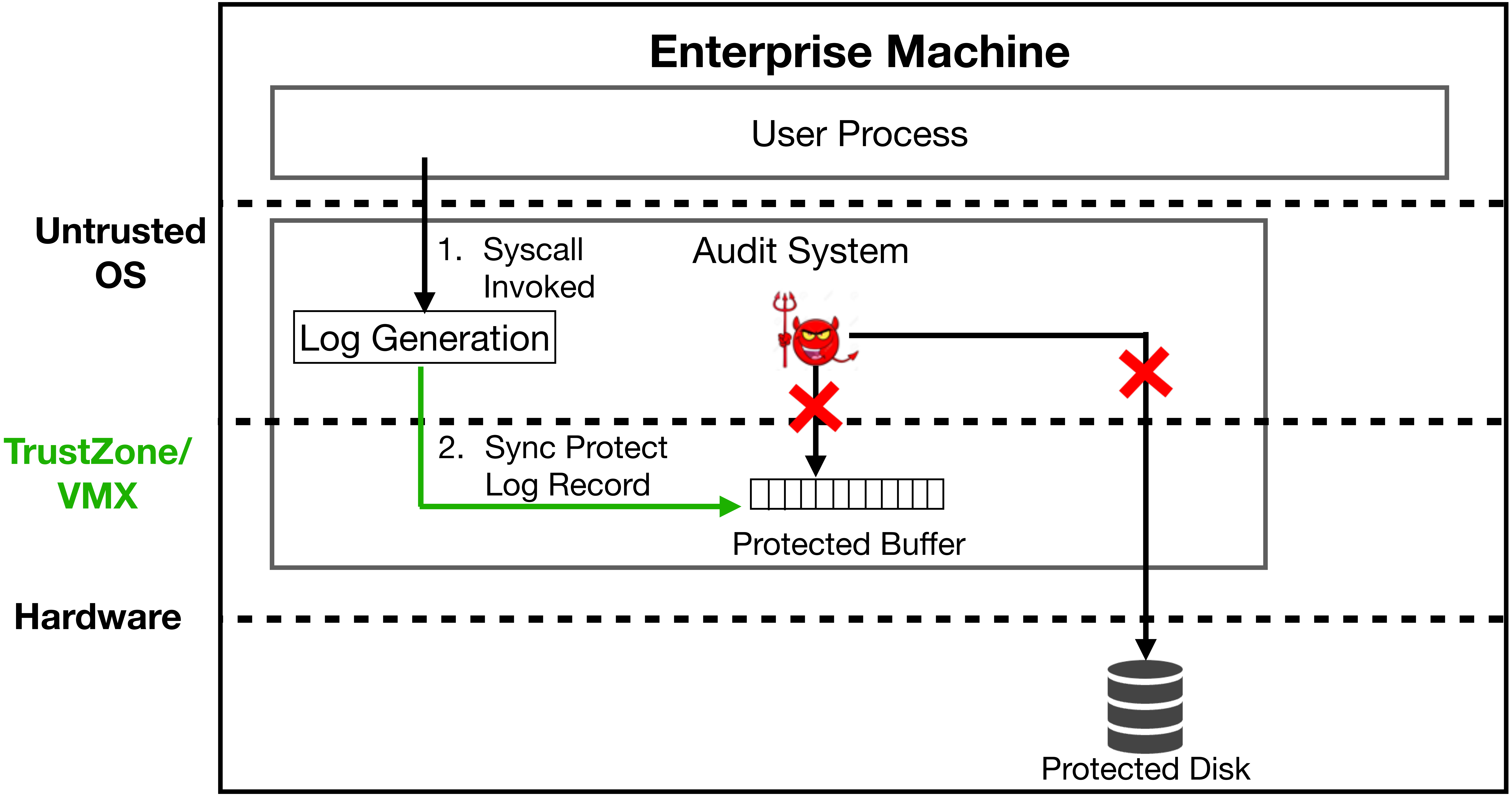
How does OMNILog build a protected environment?



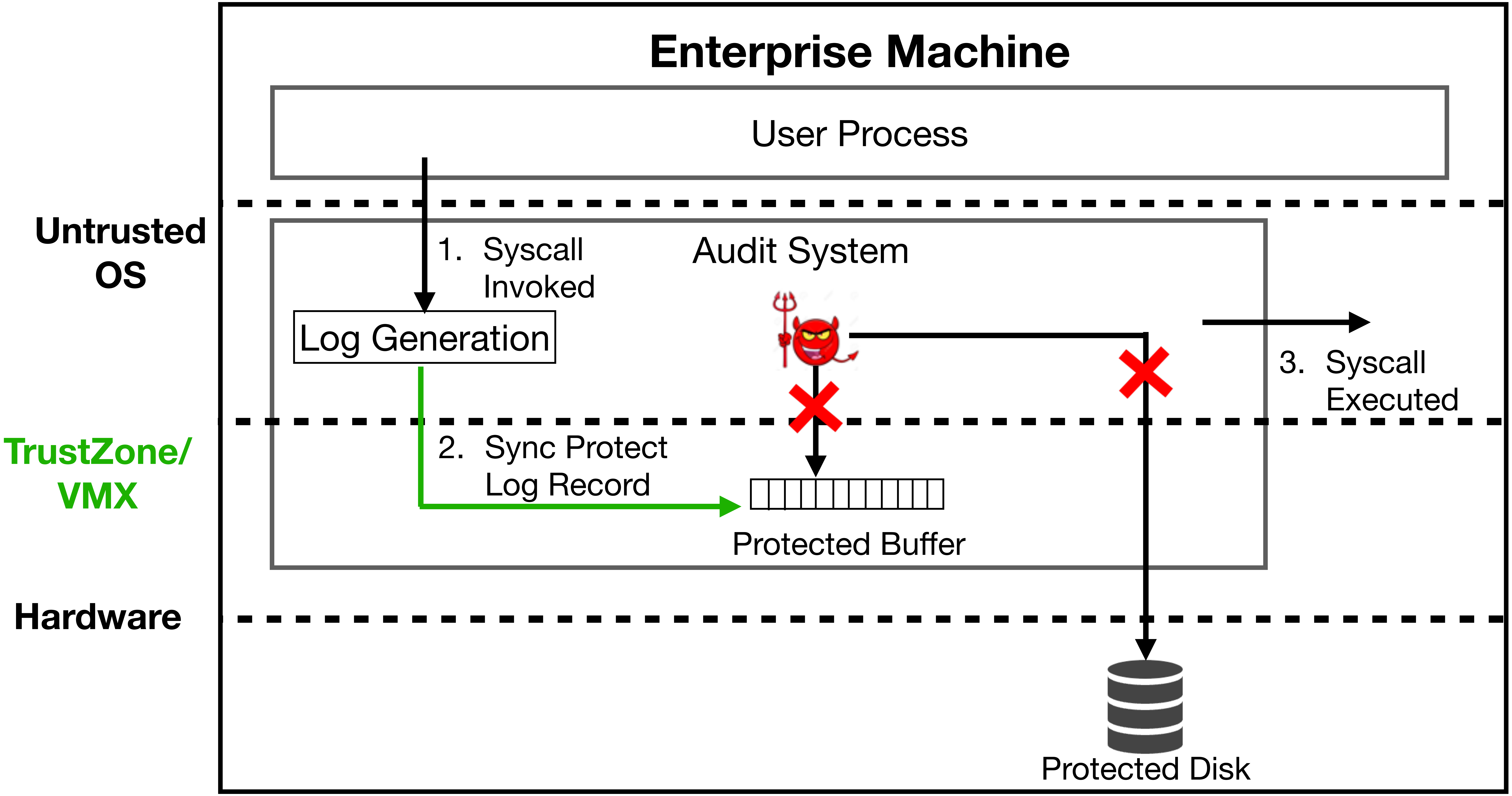
How does OMNILog build a protected environment?



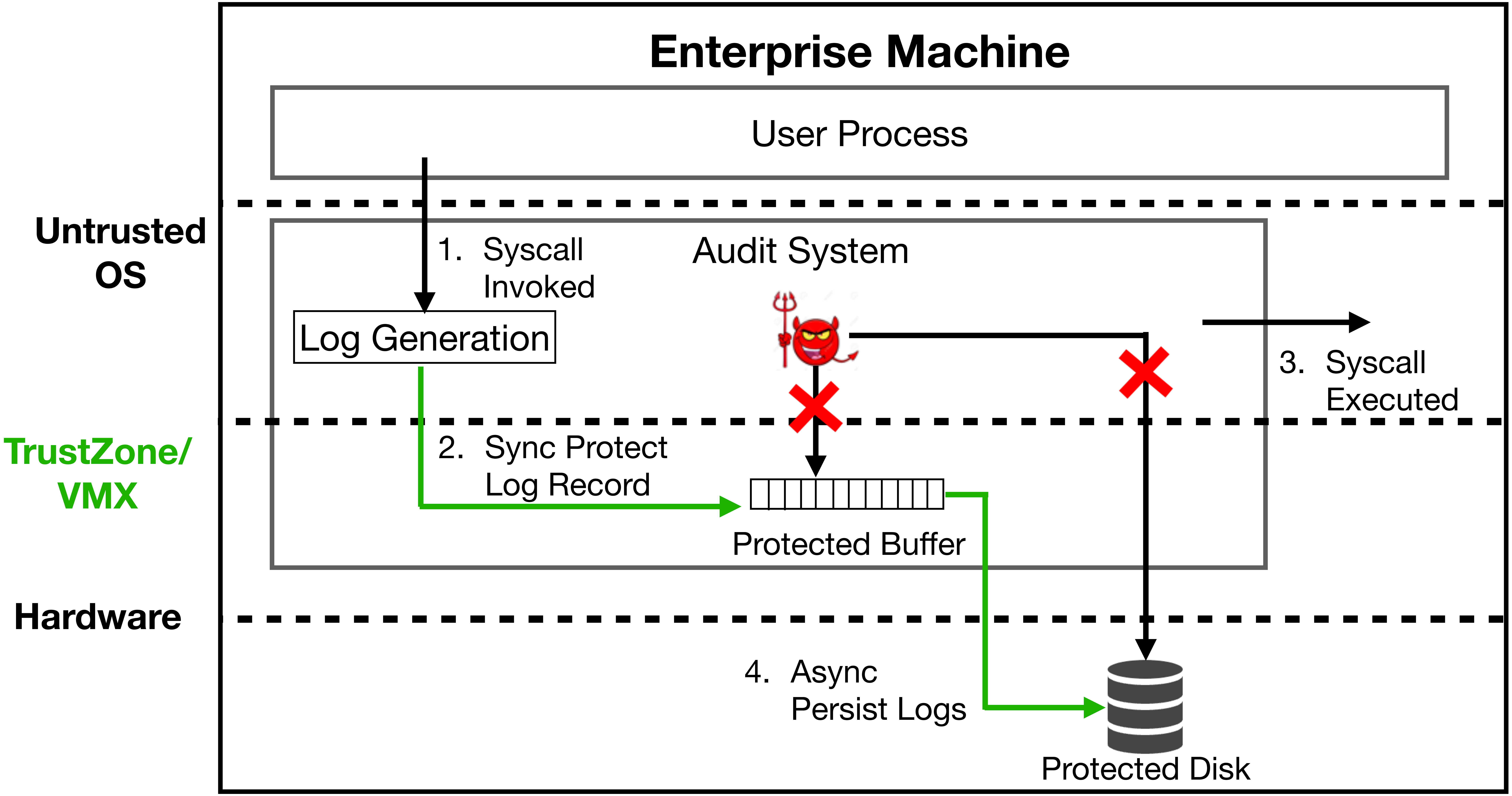
How does OMNILog build a protected environment?



How does OMNILog build a protected environment?



How does OMNILog build a protected environment?



How does OMNILog optimize the logging pipeline?

Native Auditd

OMNILog

How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation

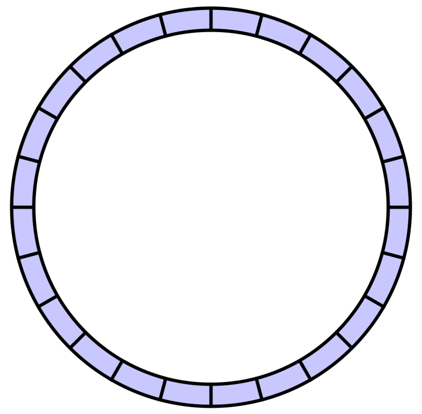
Human-Readable
~12k cycles | 1KB

OMNILog

How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation →

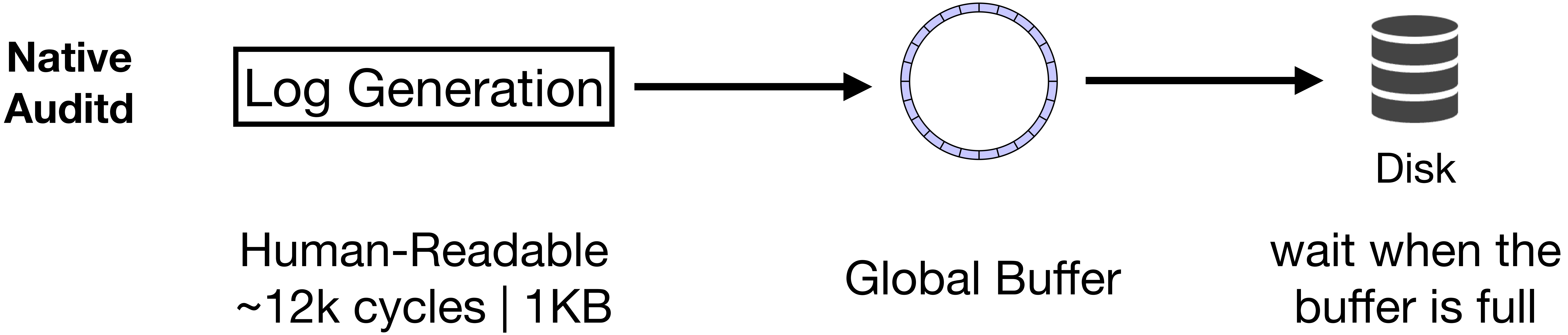


Human-Readable
~12k cycles | 1KB

Global Buffer

OMNILog

How does OMNILog optimize the logging pipeline?

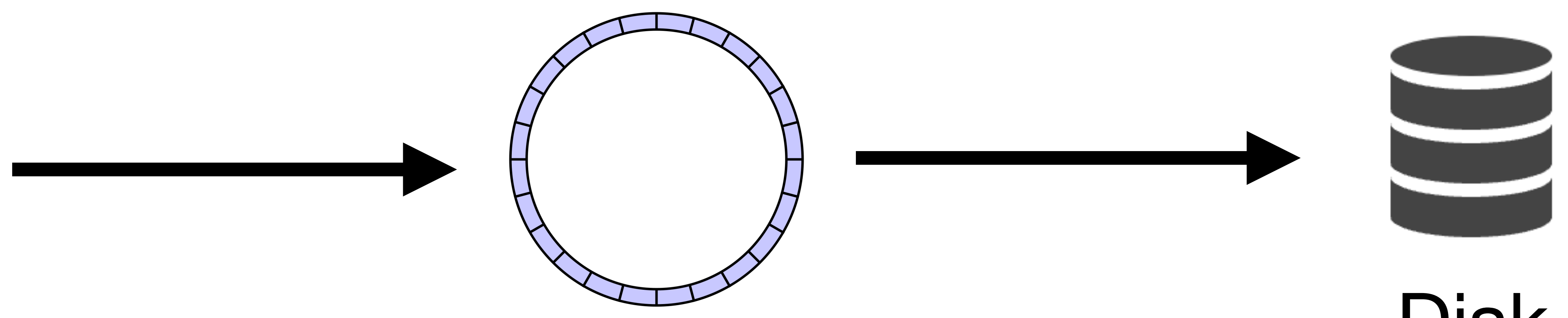


OMNILog

How does OMNILog optimize the logging pipeline?

Native Auditd

Log Generation



Disk

Human-Readable
~12k cycles | 1KB

Global Buffer

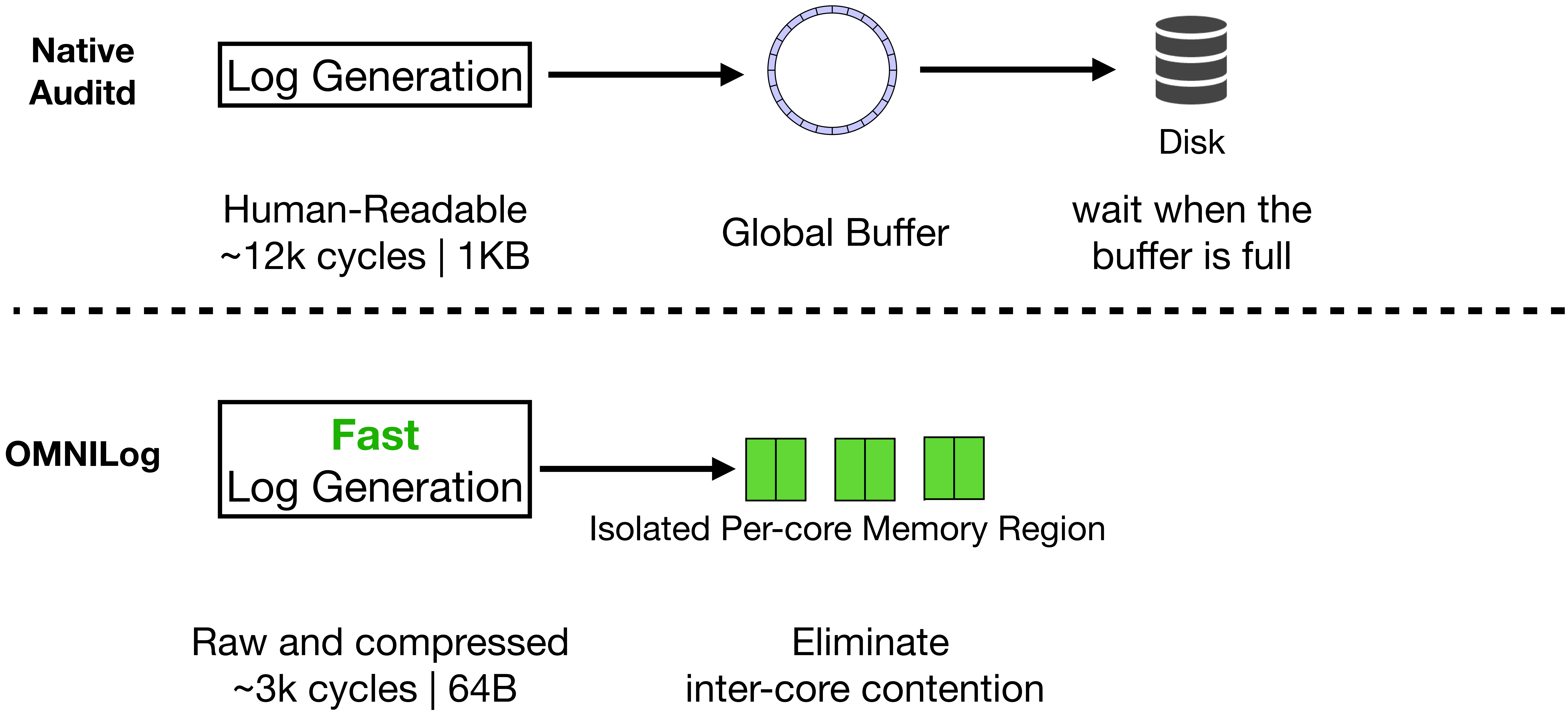
wait when the buffer is full

OMNILog

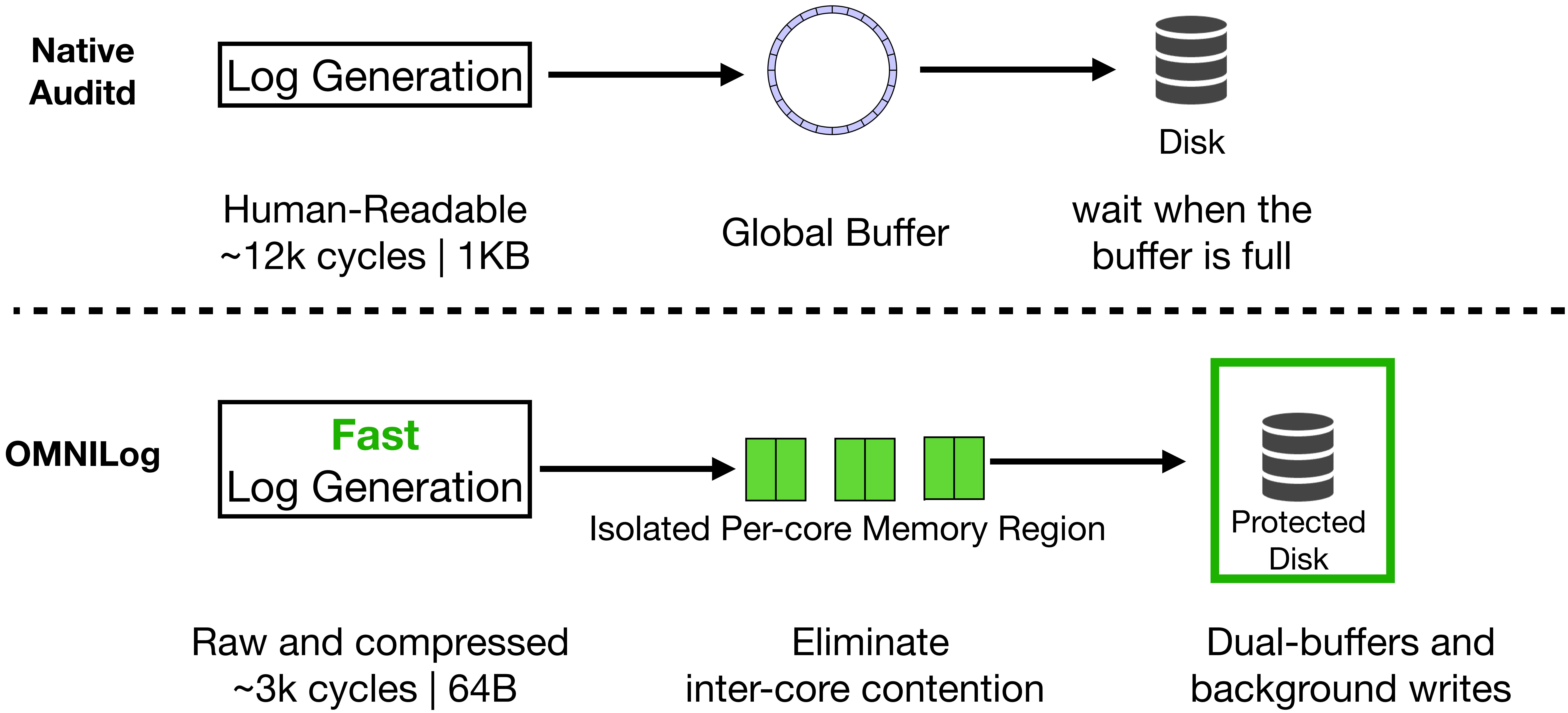
Fast
Log Generation

Raw and compressed
~3k cycles | 64B

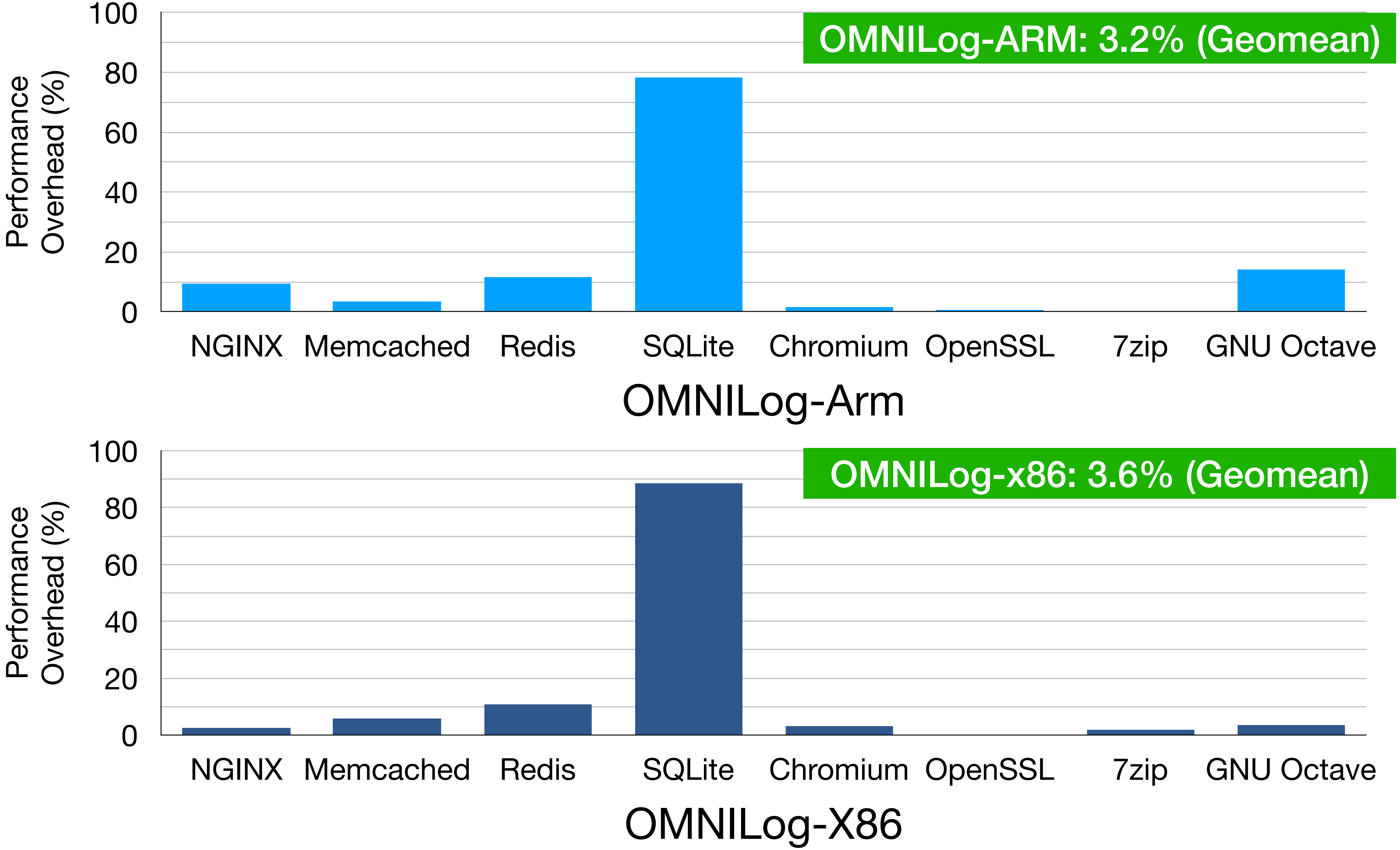
How does OMNILog optimize the logging pipeline?



How does OMNILog optimize the logging pipeline?



OMNILog incurs low overhead over native execution



Conclusion

- Current audit systems architectures:
  - Can't prevent tampering of all logs under kernel exploits
  - Can't keep a detailed trace of security-related events
- OmniLog redesigns audit architecture to:
  - Prevent all log tampering for all events
  - Keep a full trace of all syscalls executed during kernel exploits
- OMNILog's overhead compared to native execution is **~3.5%** (geomean)

Thanks!

vgandhi@g.harvard.edu