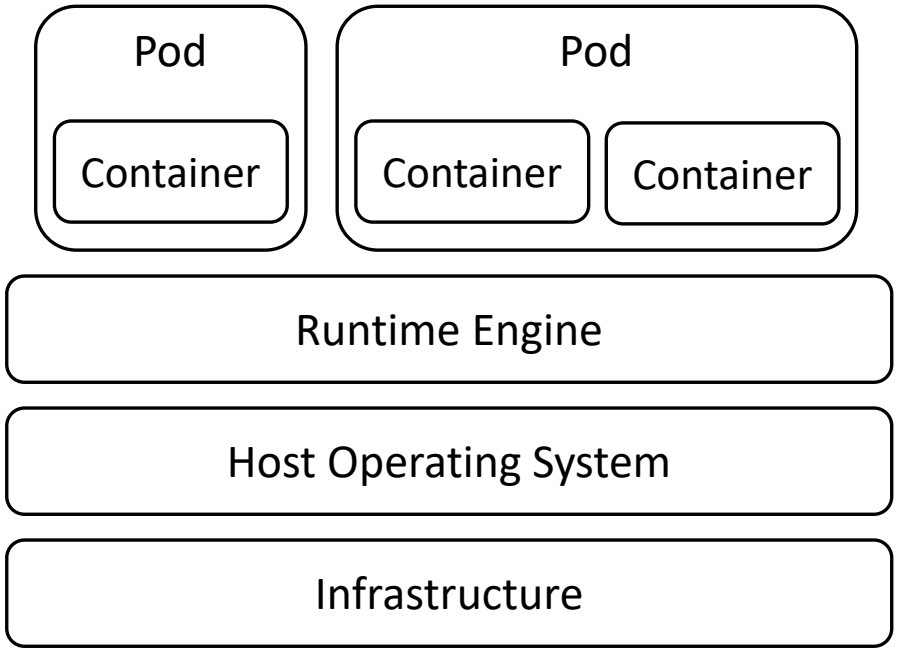
Secure Namespaced Kernel Audit for Containers

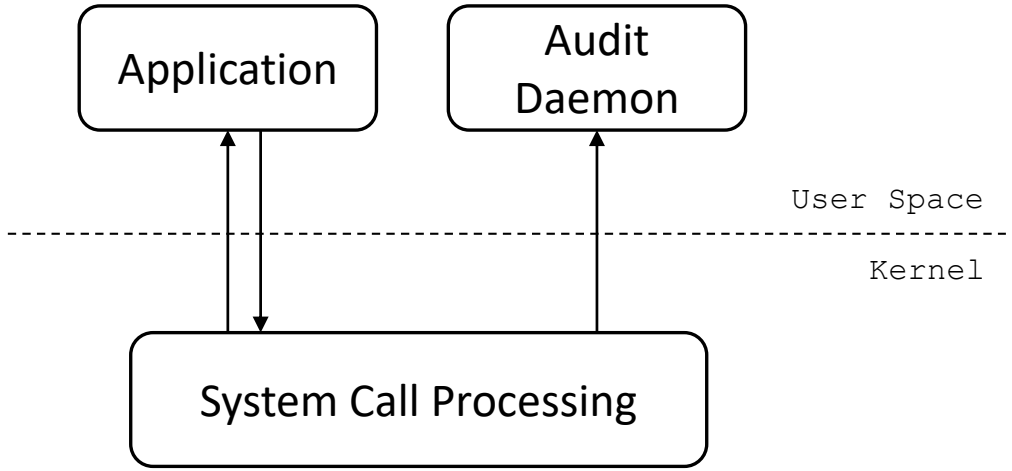
SOO YEE LIM (UBC), BOGDAN STELEA (UOB), XUEYUAN HAN (HARVARD), THOMAS PASQUIER (UBC)

EMAIL ADDRESS: SOOYEE@CS.UBC.CA

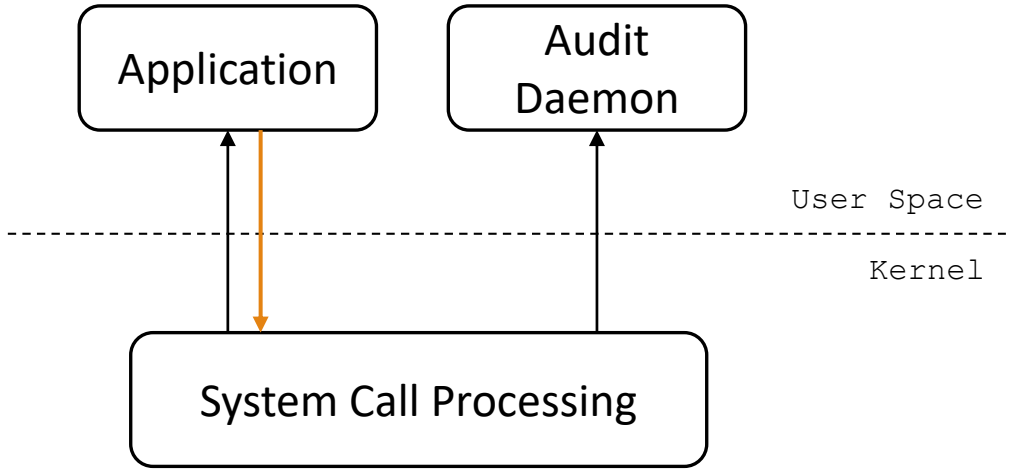
Cloud Containers (Kubernetes)



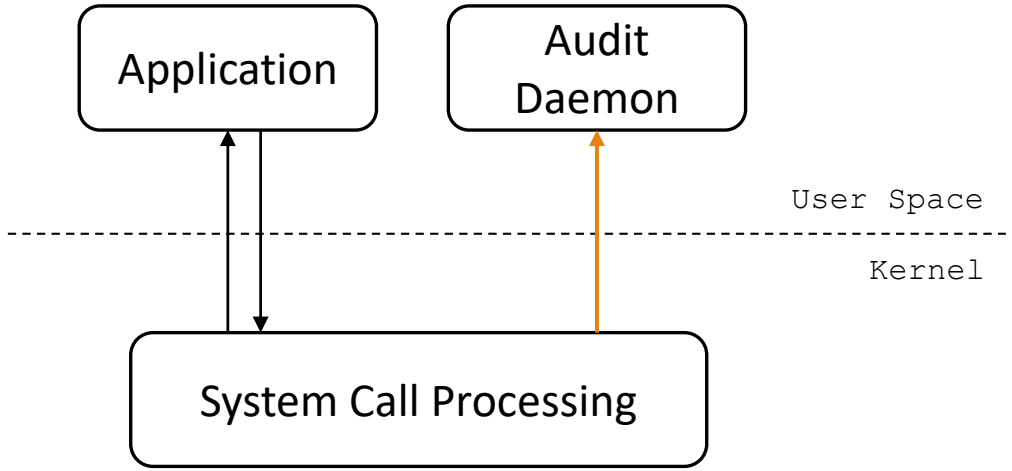
Host Audit Tool (e.g., Linux Audit Framework)



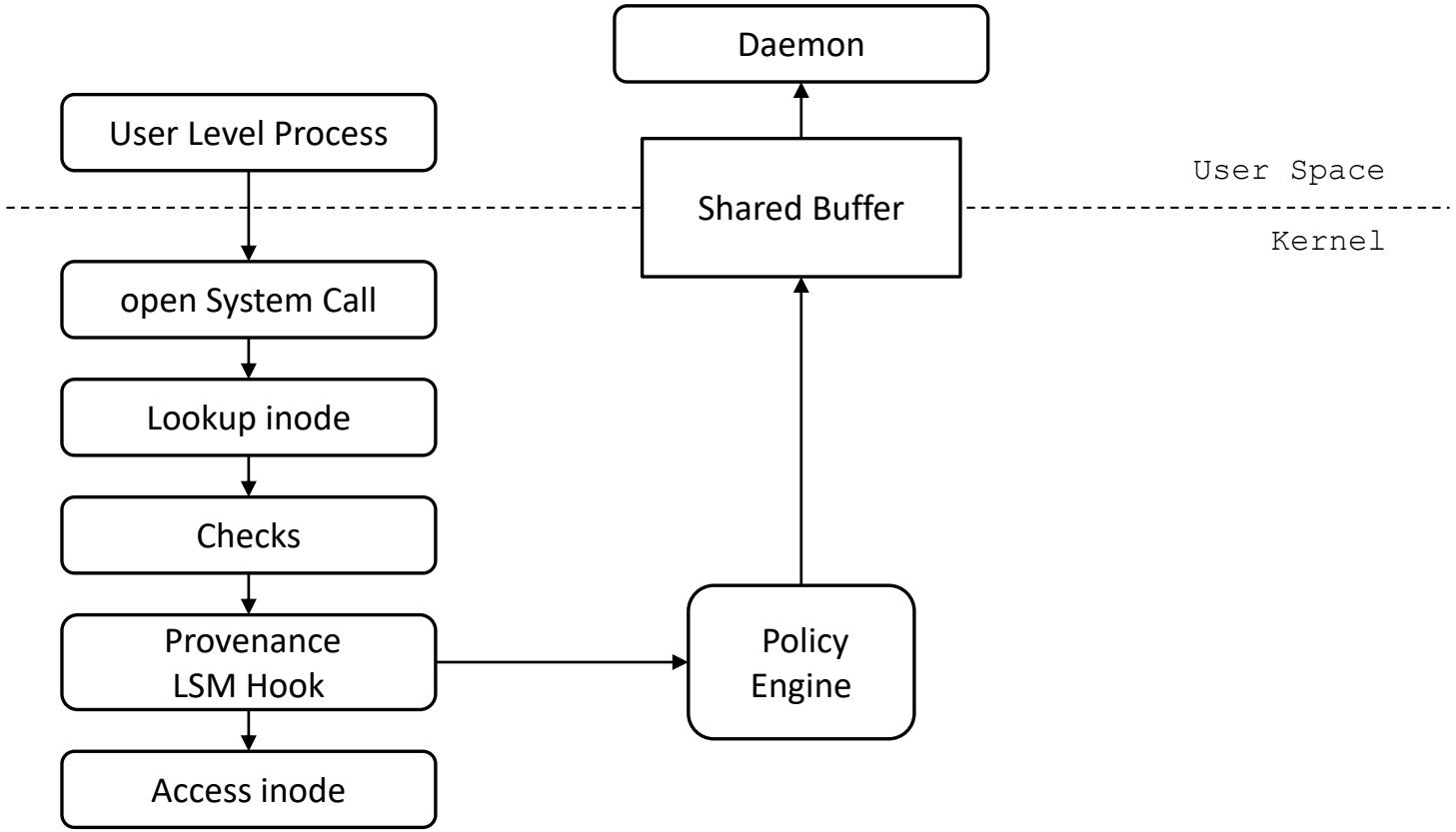
Host Audit Tool (e.g., Linux Audit Framework)



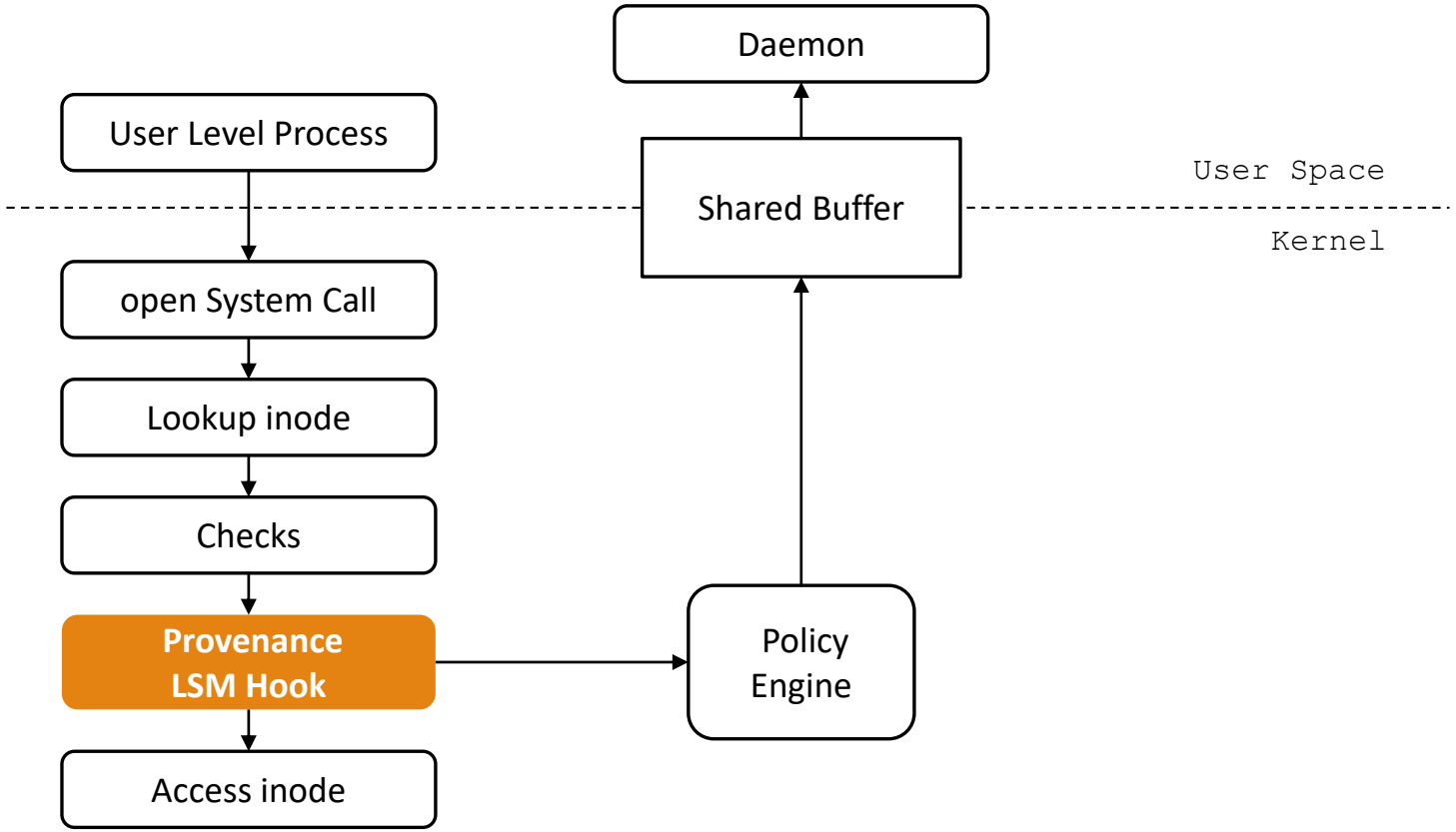
Host Audit Tool (e.g., Linux Audit Framework)



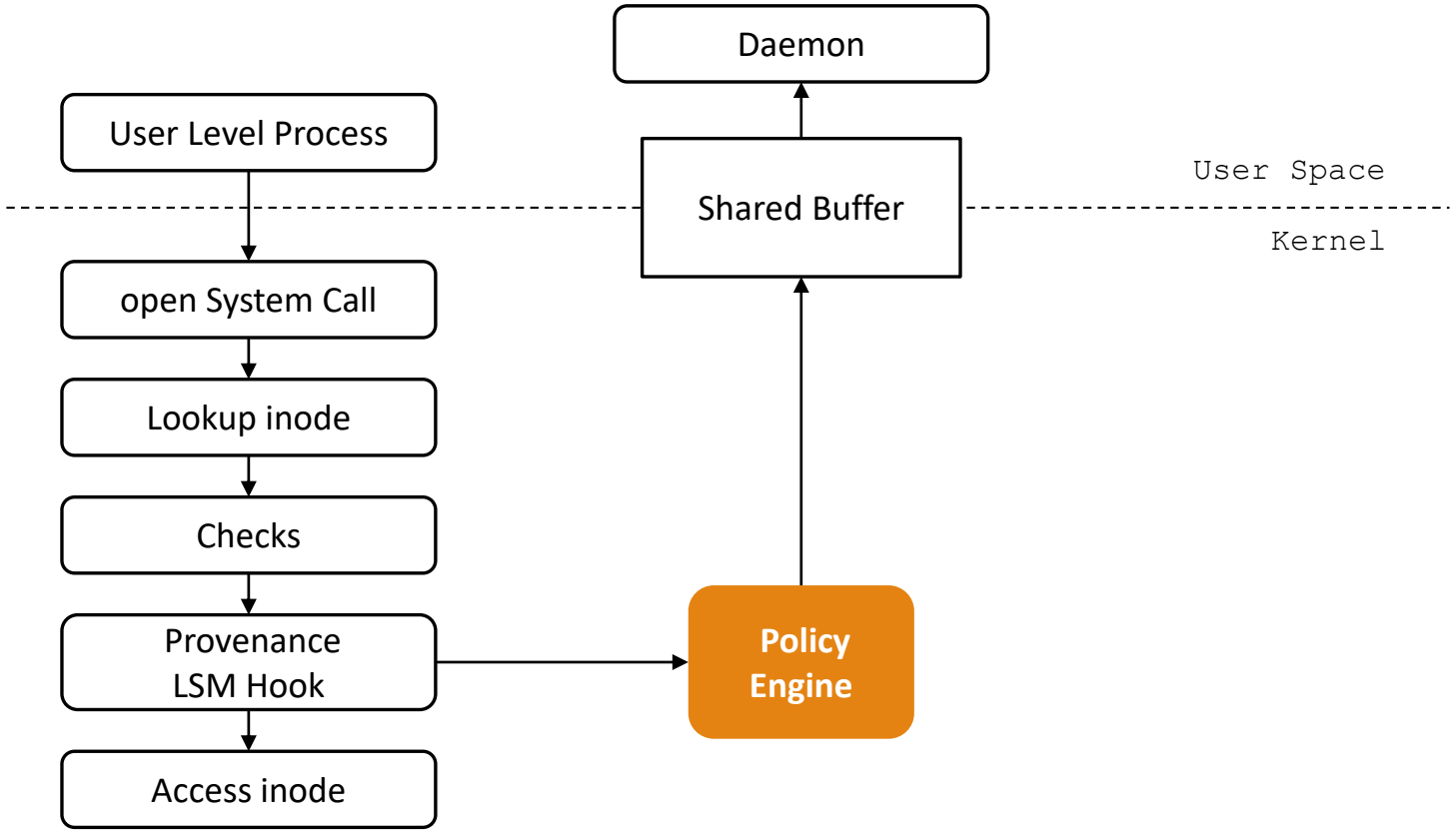
Reference Monitor-Based Audit Tool (e.g., Camflow, Hi-Fi)



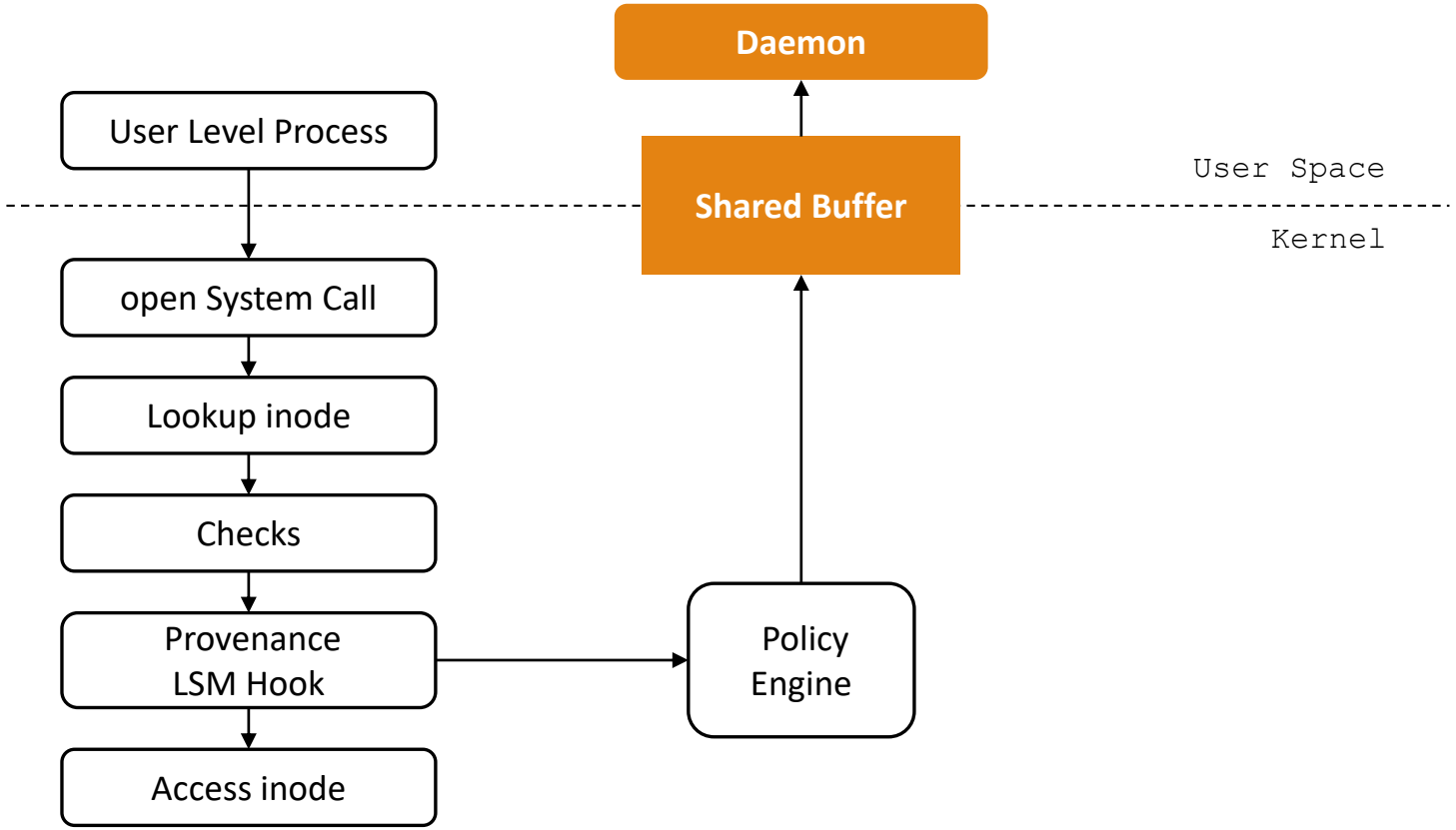
Reference Monitor-Based Audit Tool (e.g., Camflow, Hi-Fi)



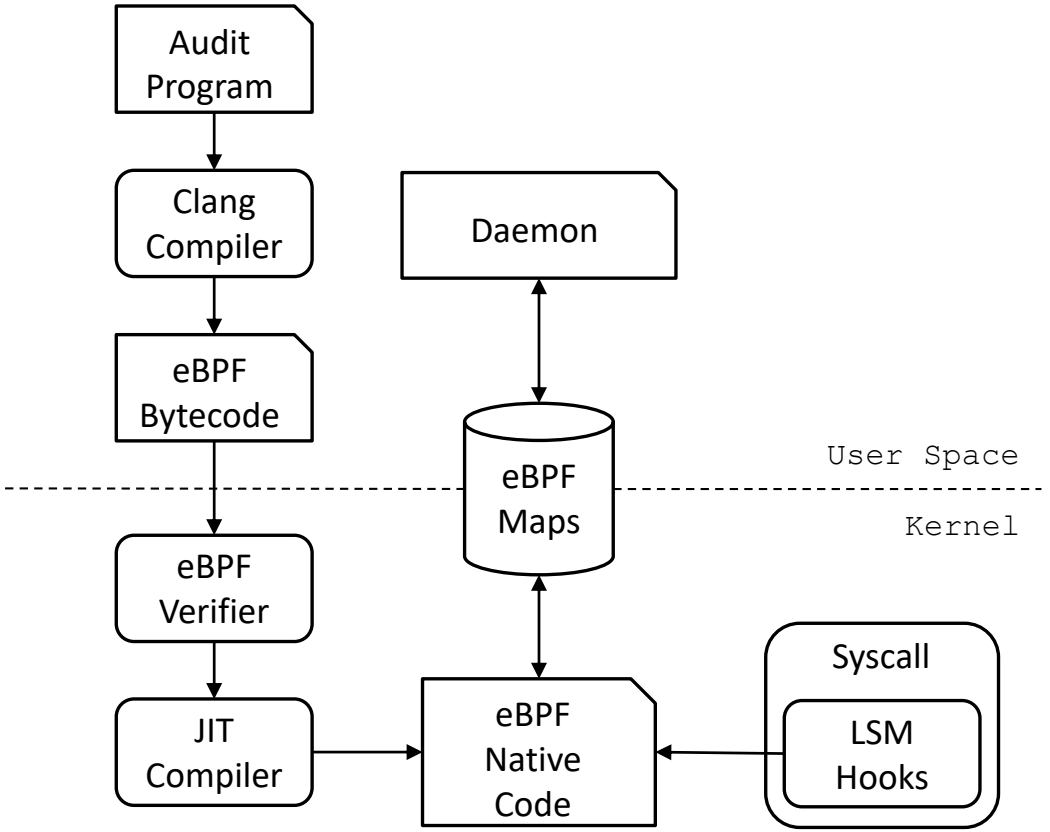
Reference Monitor-Based Audit Tool (e.g., Camflow, Hi-Fi)



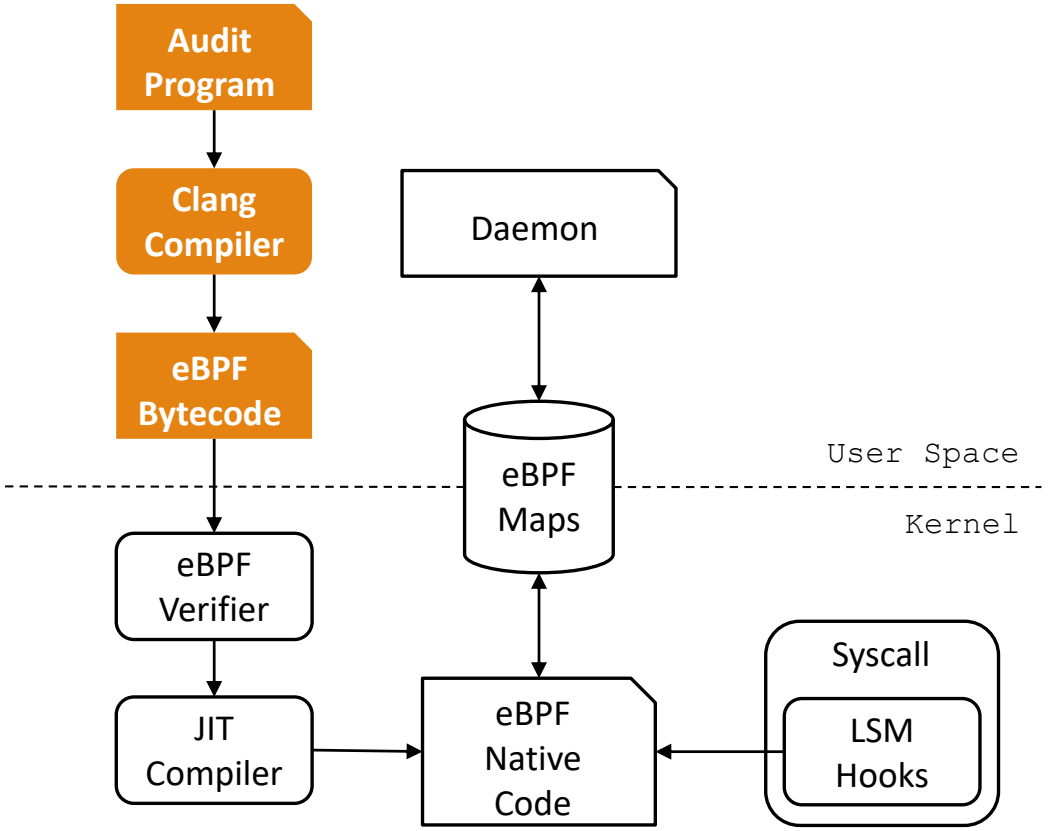
Reference Monitor-Based Audit Tool (e.g., Camflow, Hi-Fi)



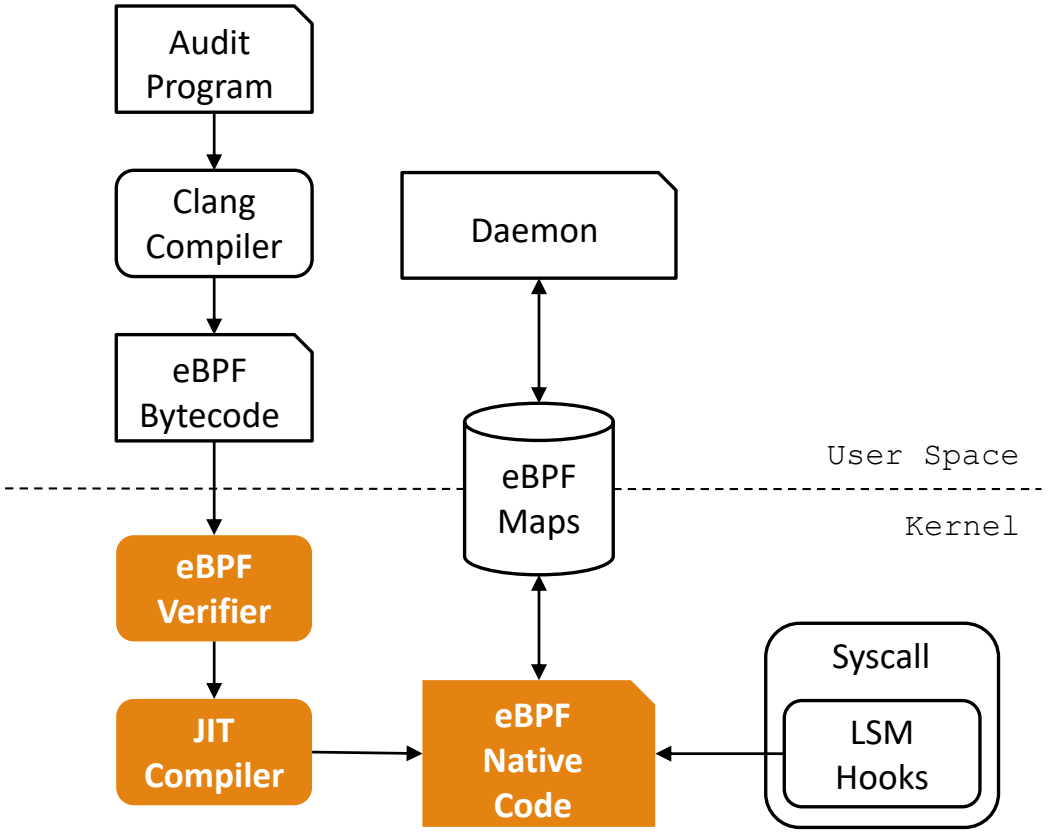
eBPF-Based Audit Tool (e.g., BPF-LSM)



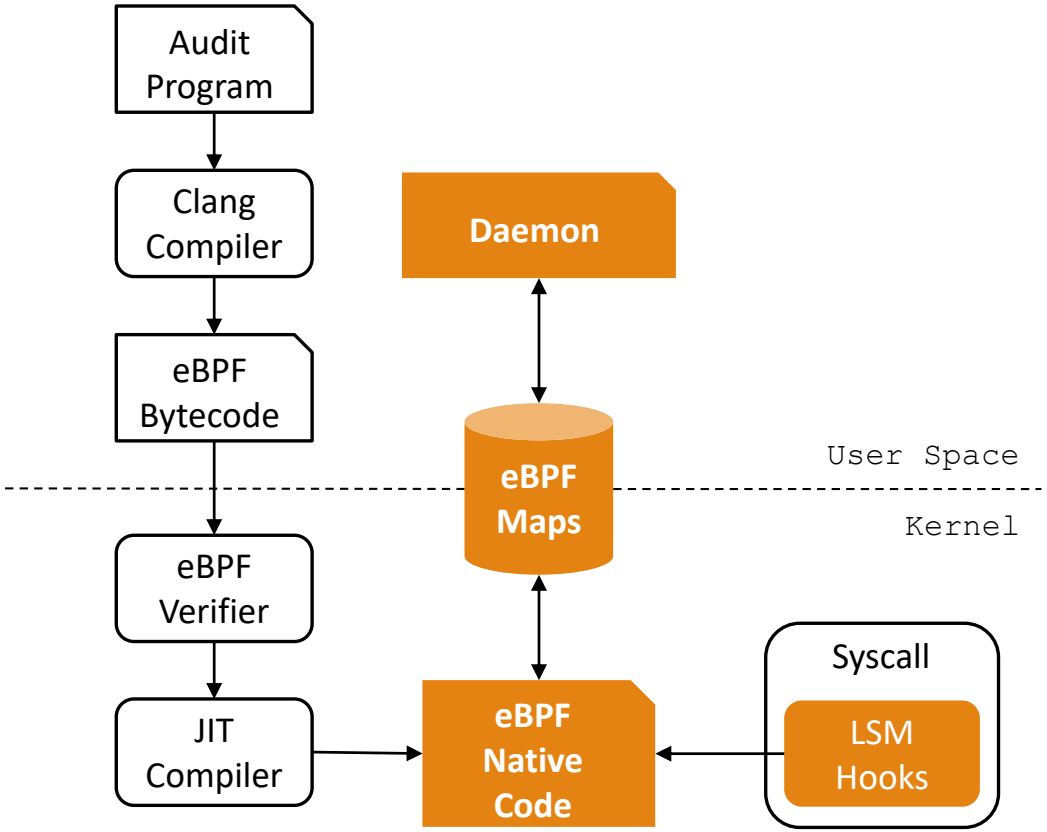
eBPF-Based Audit Tool (e.g., BPF-LSM)



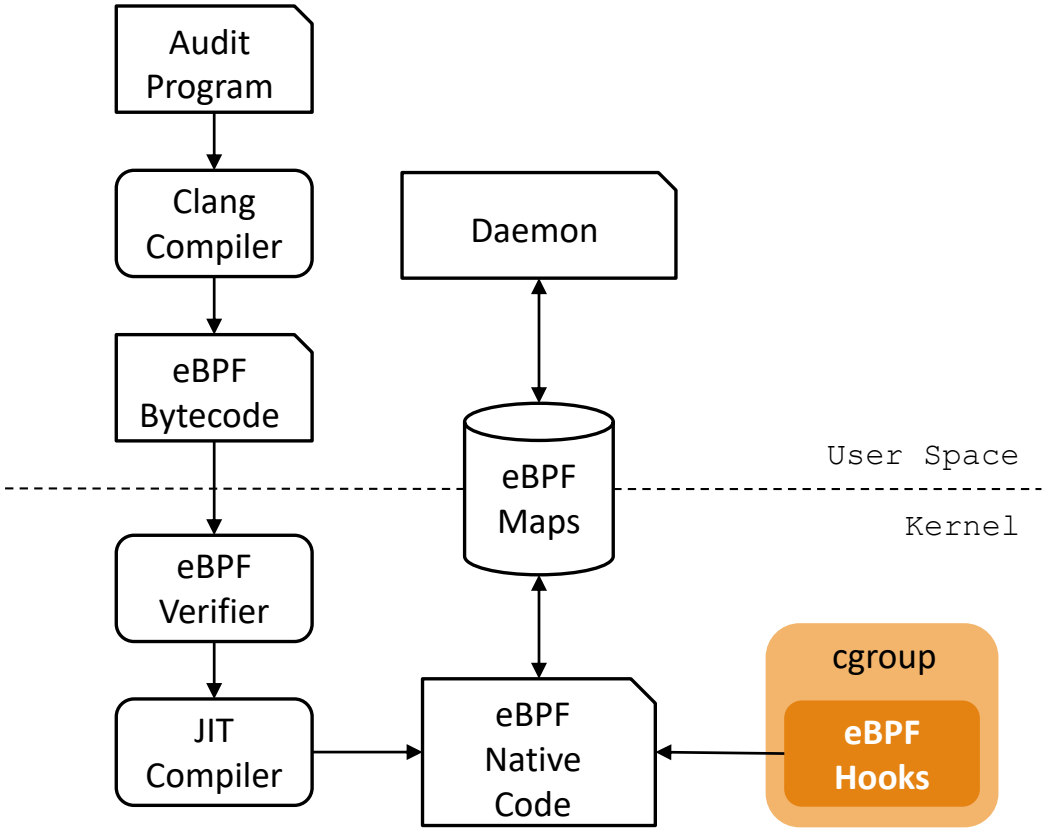
eBPF-Based Audit Tool (e.g., BPF-LSM)



eBPF-Based Audit Tool (e.g., BPF-LSM)



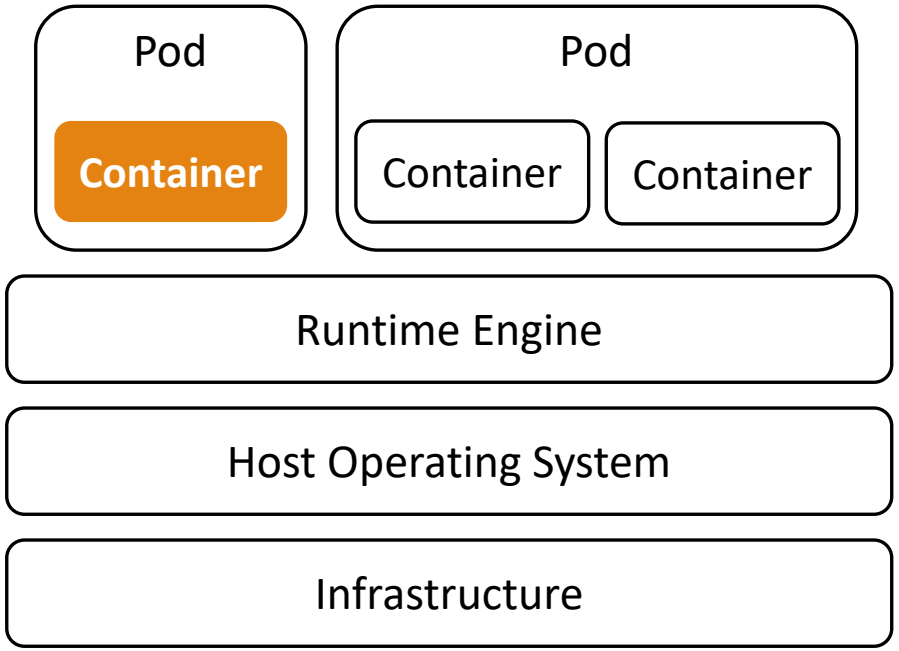
Cgroup eBPF


Audit Program
Clang Compiler
eBPF Bytecode
eBPF Verifier
JIT Compiler
Daemon
eBPF Maps
User Space
Kernel
eBPF Native Code
cgroup
eBPF Hooks

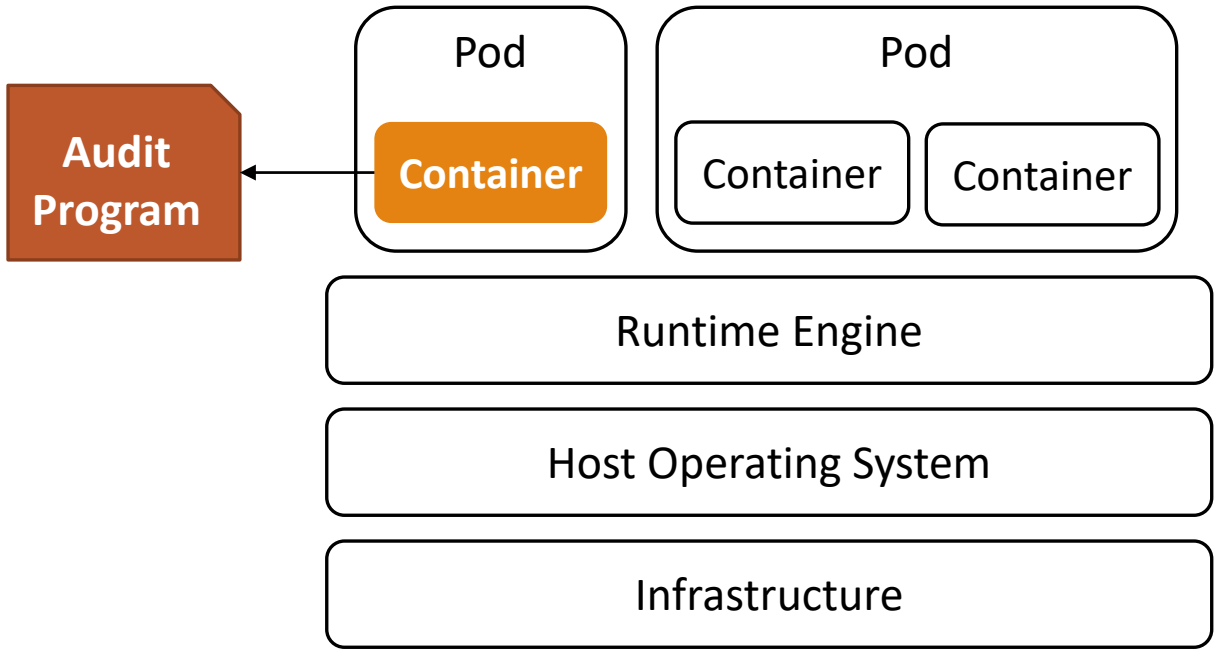
Our Objective

Feature	Host Audit Tool	Reference Monitor-Based Audit Tool	Cgroup eBPF	eBPF-Based Audit Tool (BPF-LSM)	Our Solution (saBPF)
Provides completeness guarantee in the generated logs	✗	✓	✗	✓	✓
Requires minimal host kernel modification.	✓	✗	✓	✓	✓
Supports policy specification at a container-granularity.	✗	✗	✓	✗	✓

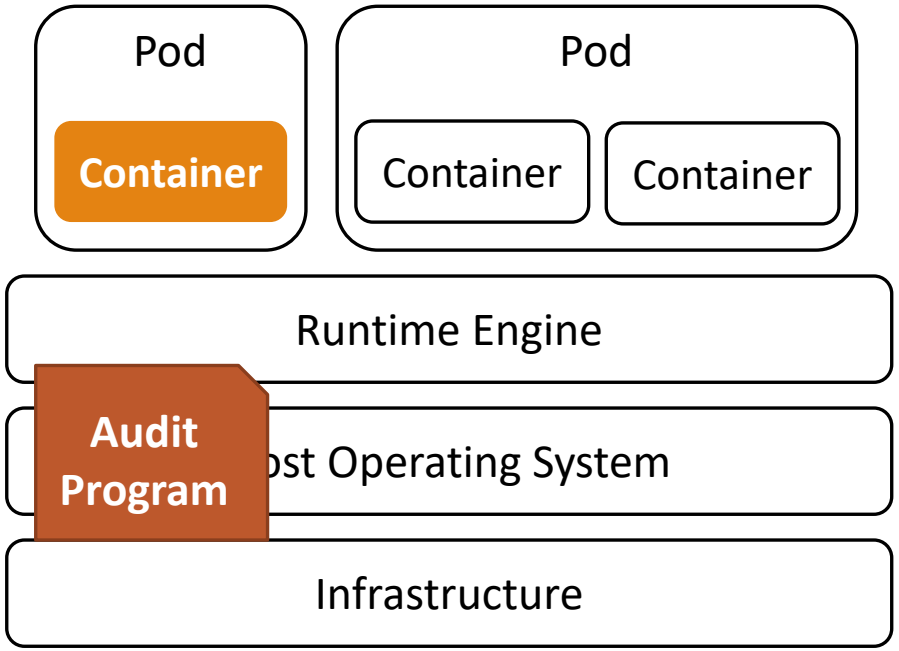
Our Solution: saBPF



Our Solution: saBPF

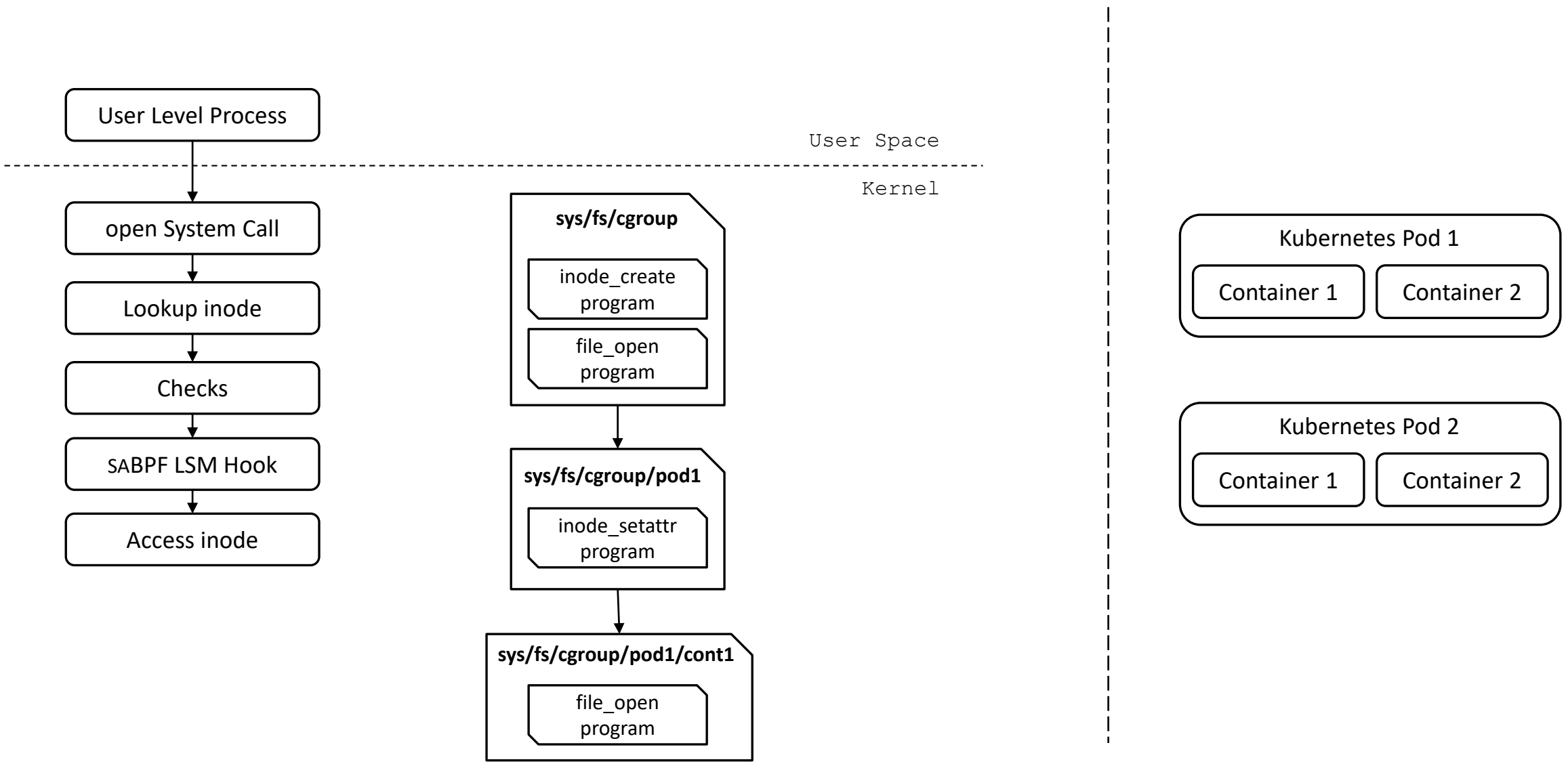


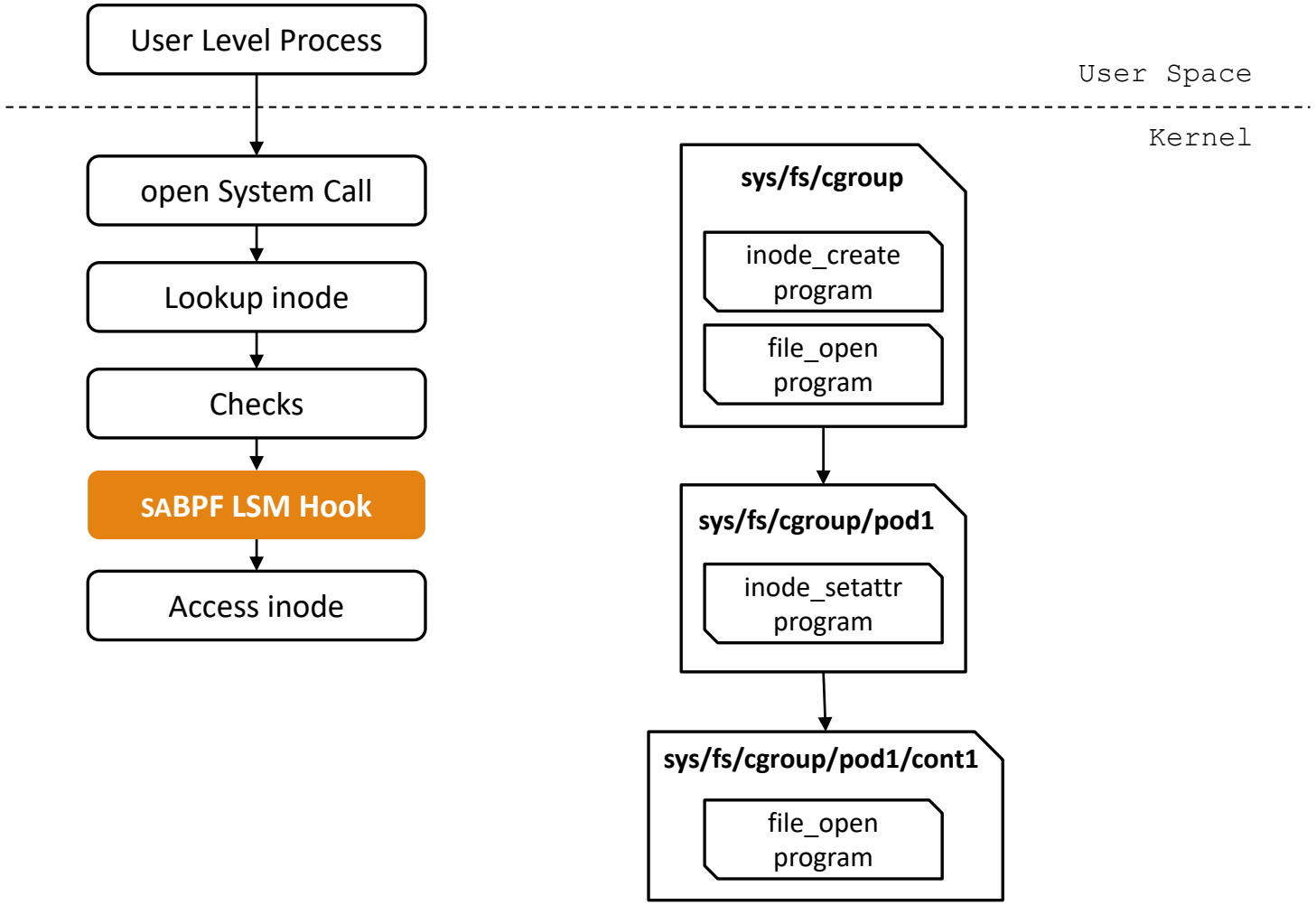
Our Solution: saBPF

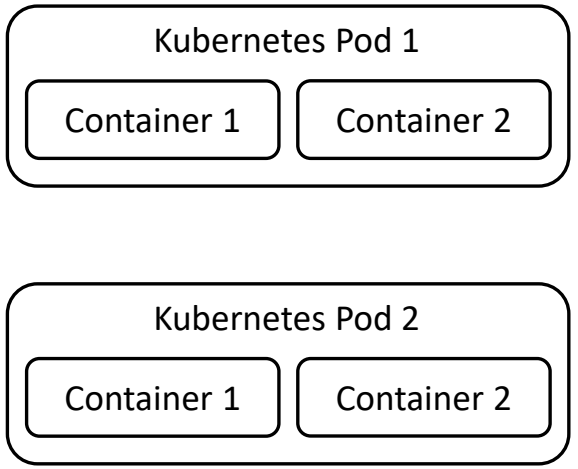


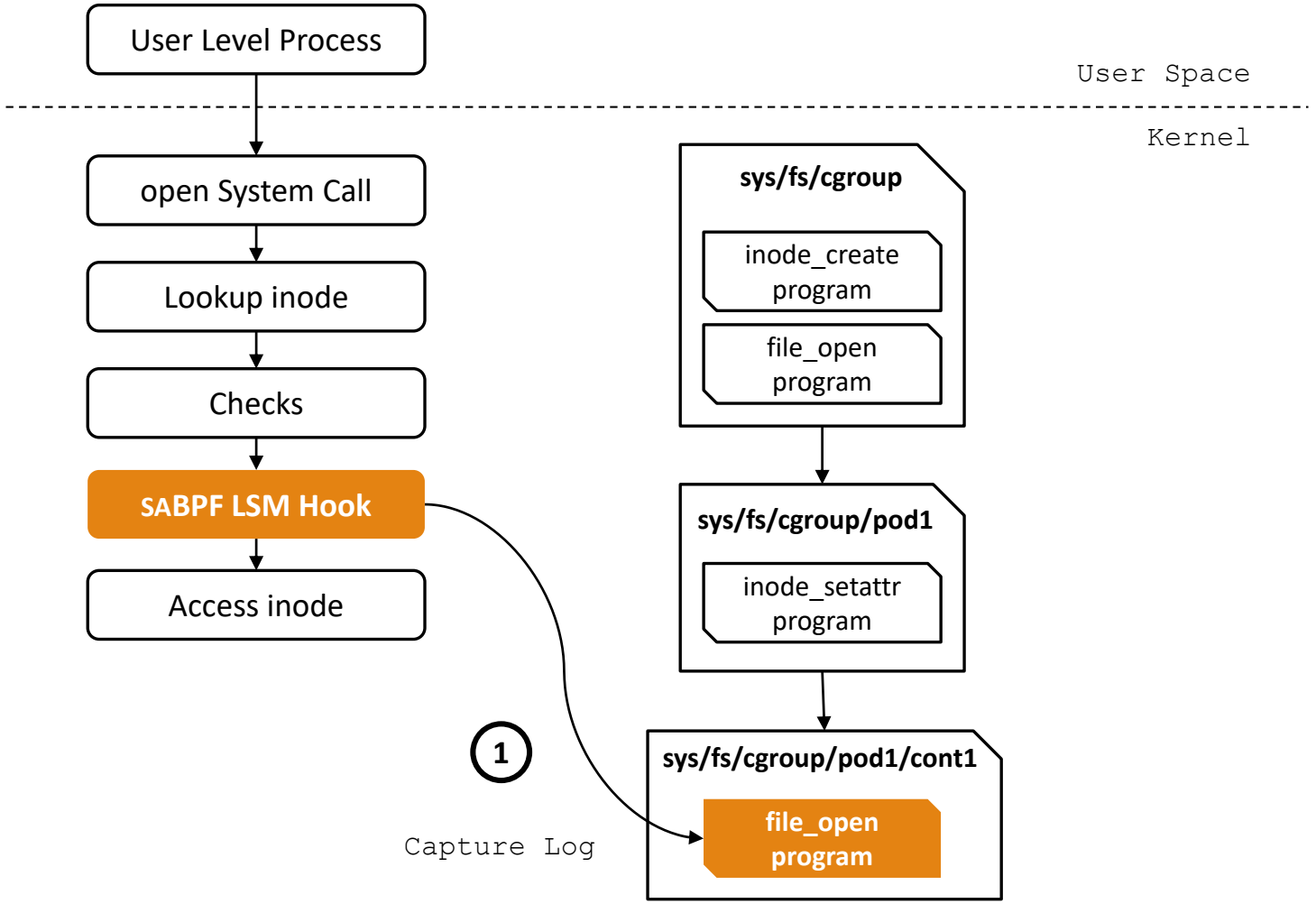
Core Components of saBPF

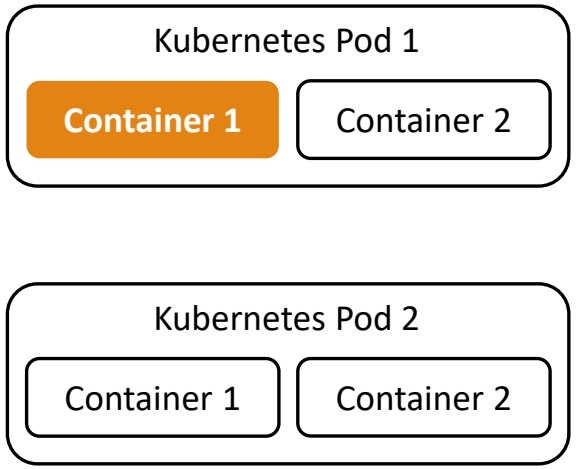
- extended Berkeley Packet Filter (eBPF)
- Linux Security Modules (LSM)
- Linux Namespaces, particularly cgroup

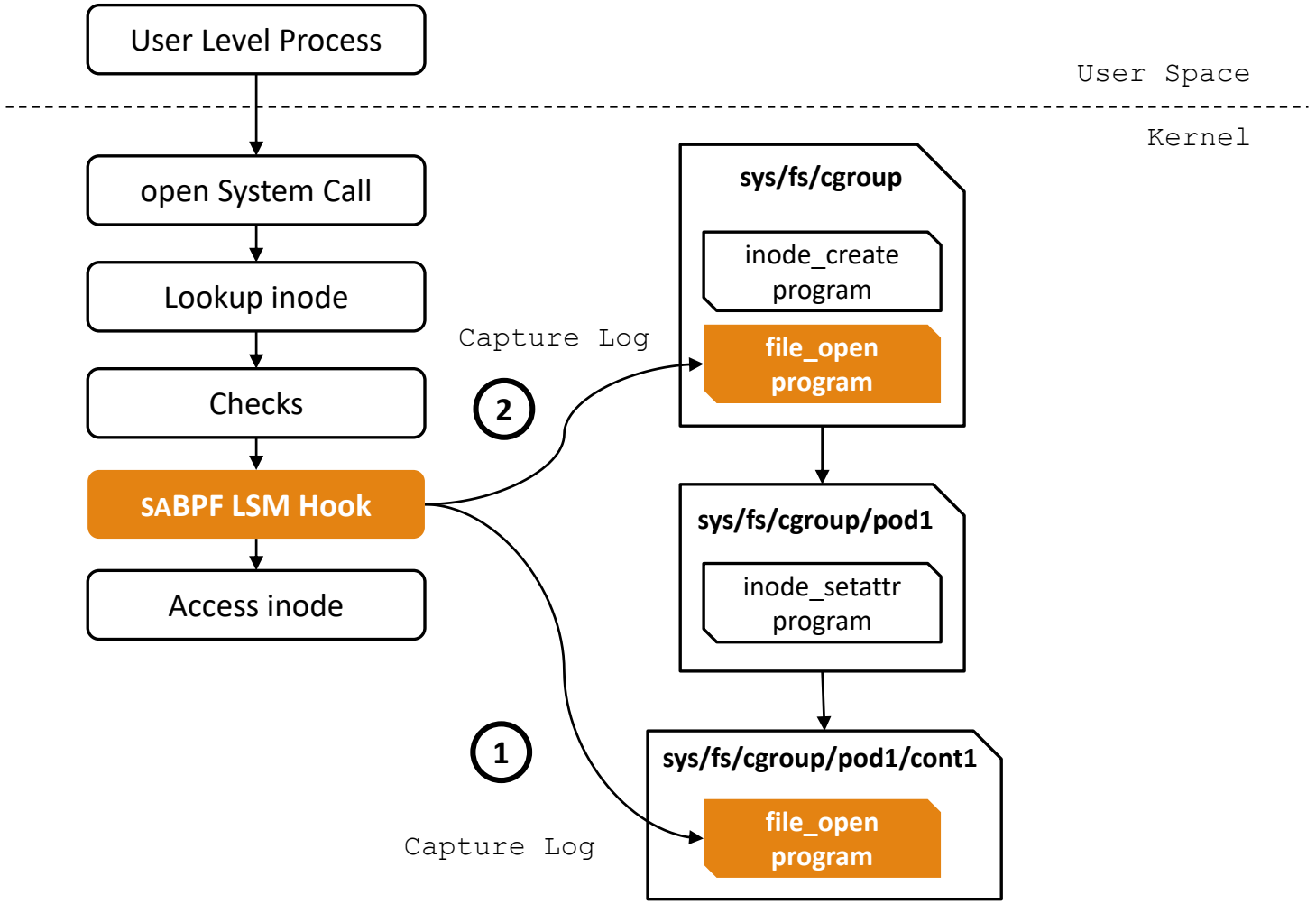

User Level Process
User Space
Kernel
open System Call
Lookup inode
Checks
SABPF LSM Hook
Access inode
sys/fs/cgroup
inode_create program
file_open program
sys/fs/cgroup/pod1
inode_setattr program
sys/fs/cgroup/pod1/cont1
file_open program
Kubernetes Pod 1
Container 1
Container 2
Kubernetes Pod 2
Container 1
Container 2

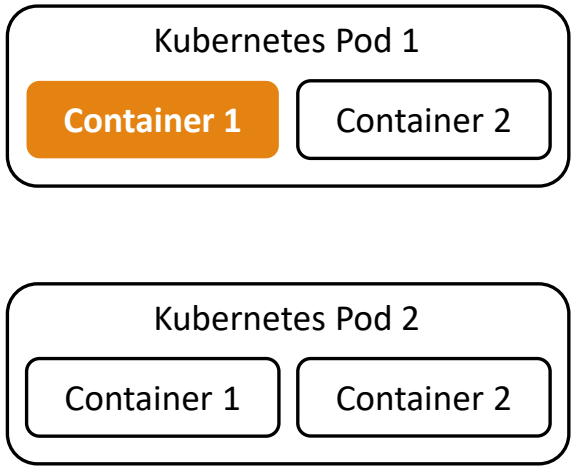

User Level Process
User Space
Kernel
open System Call
Lookup inode
Checks
SABPF LSM Hook
Access inode
sys/fs/cgroup
inode_create program
file_open program
sys/fs/cgroup/pod1
inode_setattr program
sys/fs/cgroup/pod1/cont1
file_open program


Kubernetes Pod 1
Container 1
Container 2
Kubernetes Pod 2
Container 1
Container 2


User Level Process
User Space
Kernel
open System Call
Lookup inode
Checks
SABPF LSM Hook
Access inode
sys/fs/cgroup
inode_create program
file_open program
sys/fs/cgroup/pod1
inode_setattr program
sys/fs/cgroup/pod1/cont1
file_open program
1
Capture Log


Kubernetes Pod 1
Container 1
Container 2
Kubernetes Pod 2
Container 1
Container 2

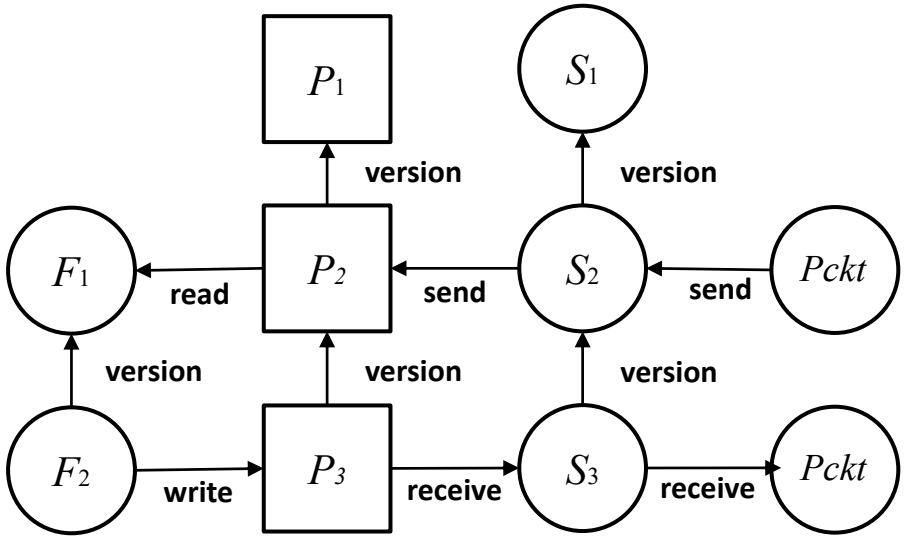

User Level Process
User Space
Kernel
open System Call
Lookup inode
Checks
SABPF LSM Hook
Access inode
Capture Log
2
1
Capture Log
sys/fs/cgroup
inode_create program
file_open program
sys/fs/cgroup/pod1
inode_setattr program
sys/fs/cgroup/pod1/cont1
file_open program


Kubernetes Pod 1
Container 1
Container 2
Kubernetes Pod 2
Container 1
Container 2

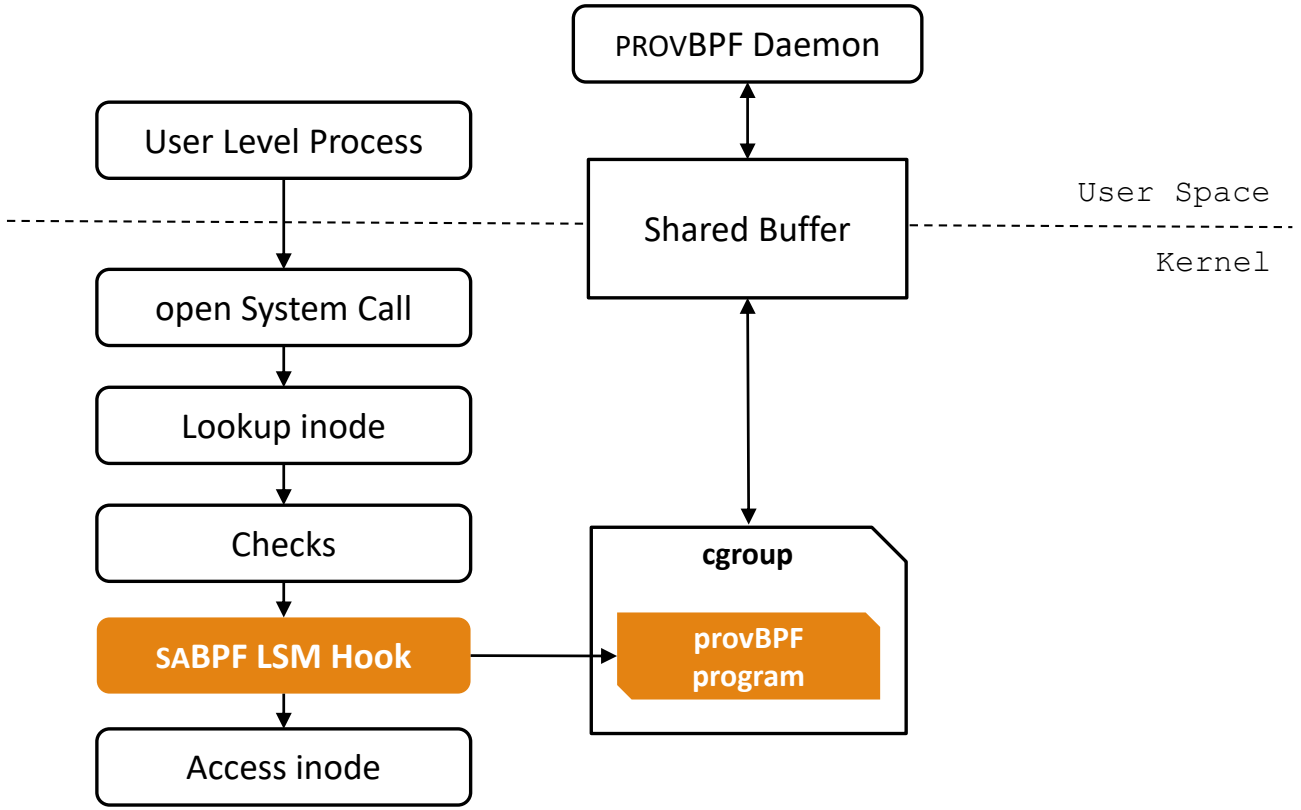
Use Cases of saBPF

- Whole-system provenance capture (ProvBPF)
- Provenance-based intrusion detection system
- Lightweight ad-hoc access control

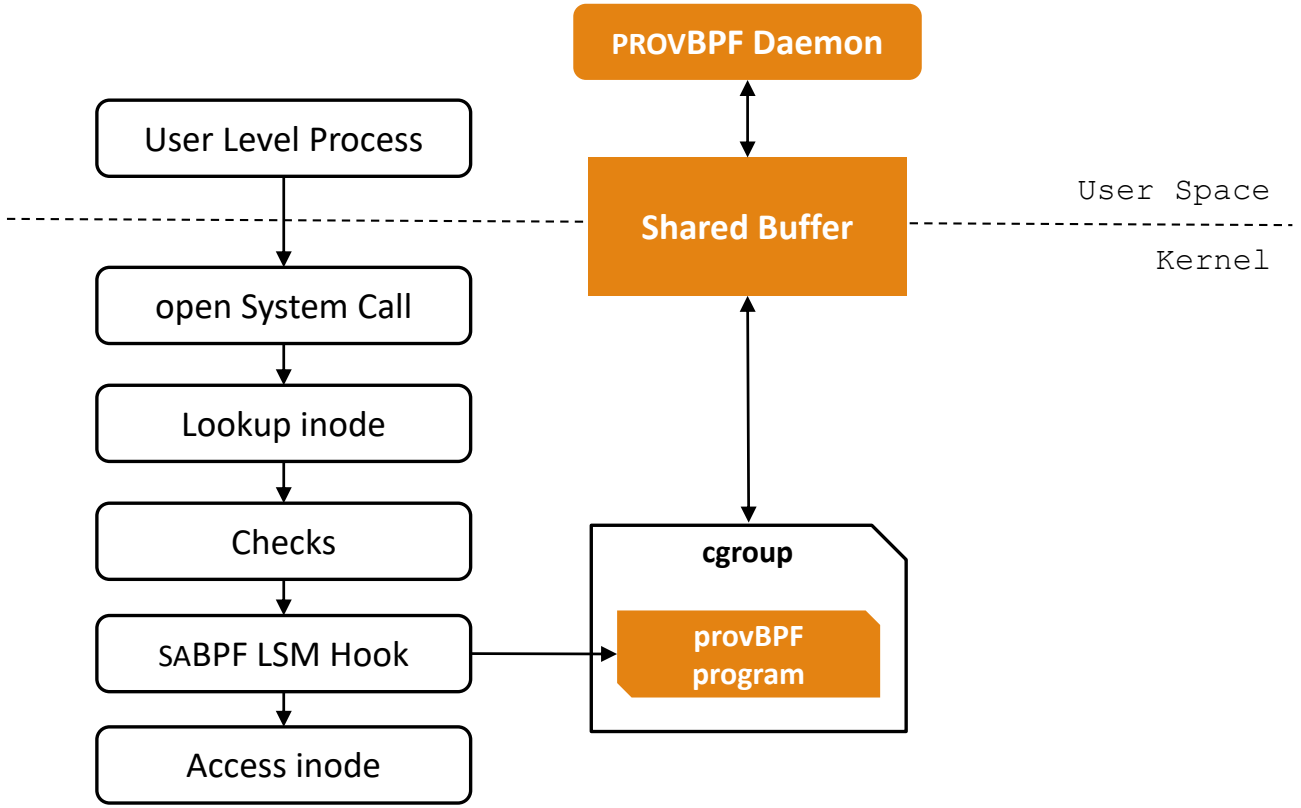
Whole-System Provenance



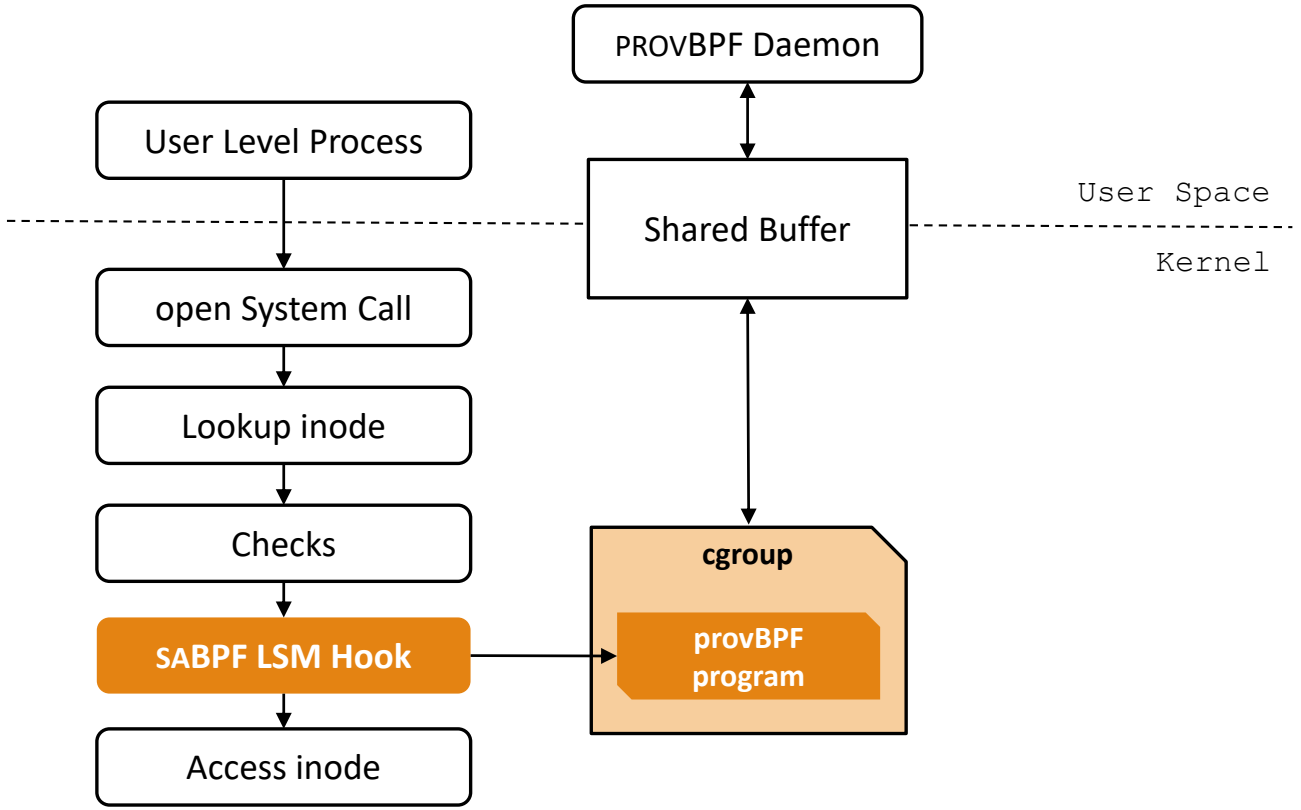
Whole-System Provenance Capture Mechanism (ProvBPF)



Whole-System Provenance Capture Mechanism (ProvBPF)



ProvBPF: Advantages of Using saBPF

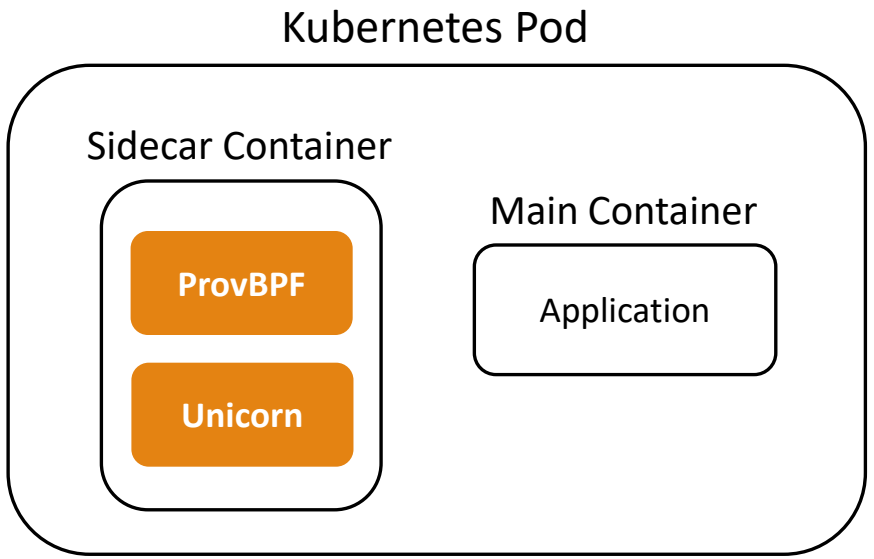


Provenance-Based Intrusion Detection System

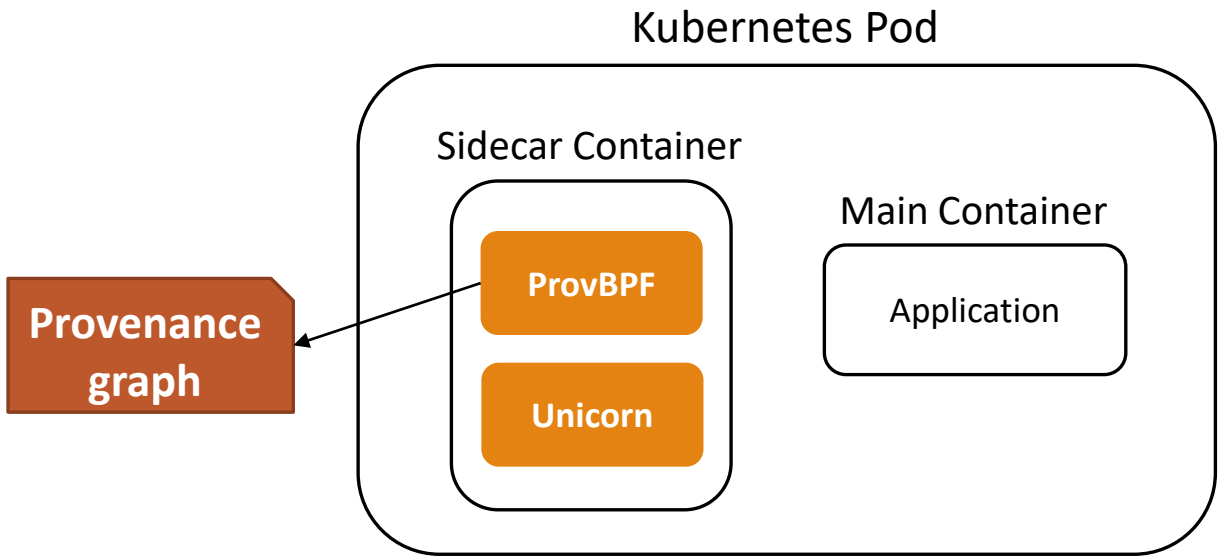
Provenance-based intrusion detection systems learn system behaviours from the provenance graph generated by benign system activity.

However, provenance capture mechanisms are typically deployed at the system-level.

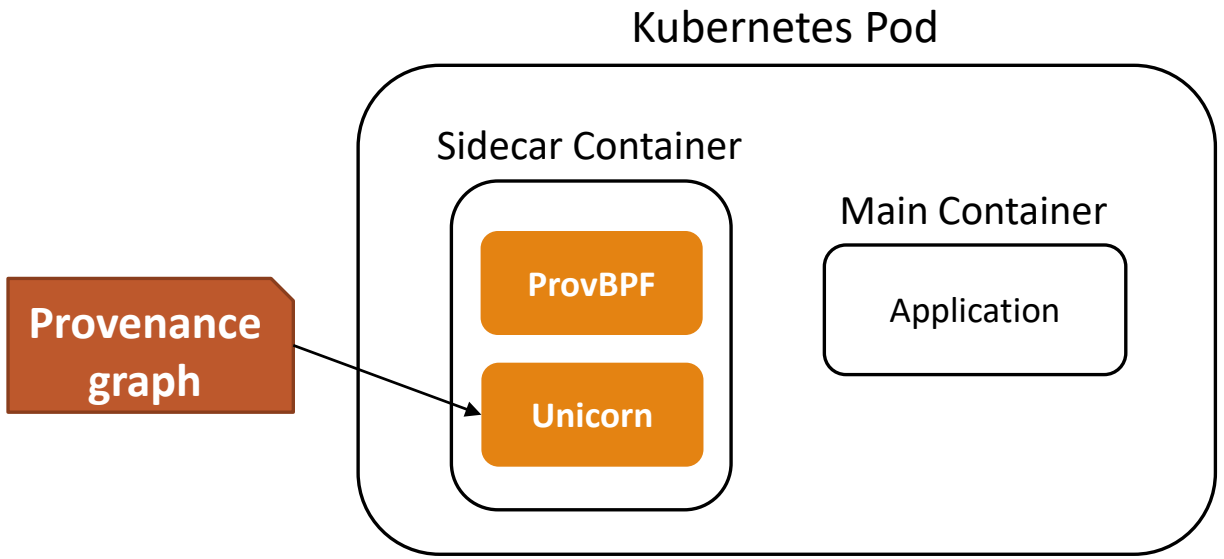
Kubernetes' Sidecar Design



Kubernetes' Sidecar Design



Kubernetes' Sidecar Design



Overhead of Namespacing

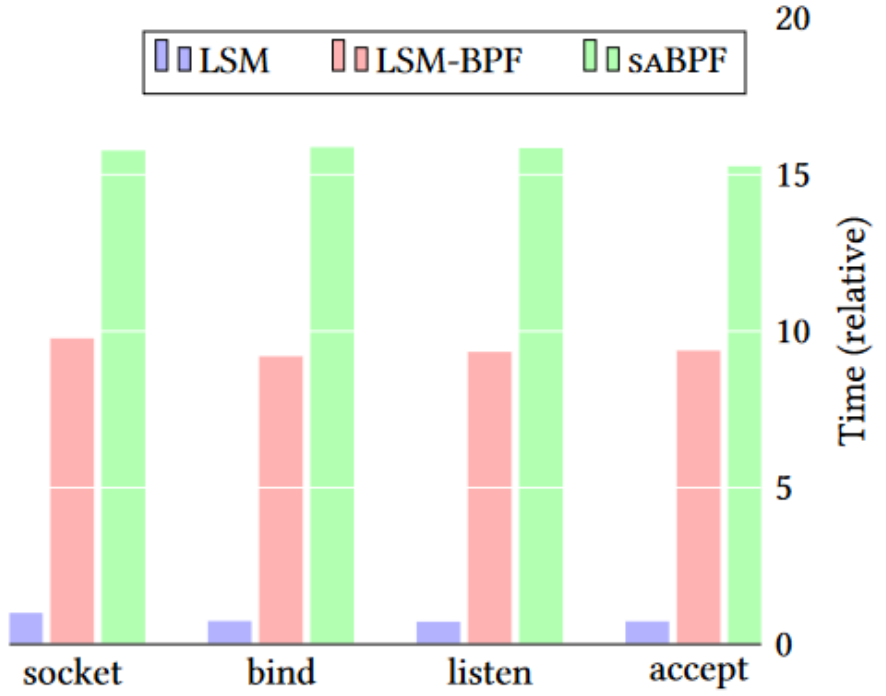
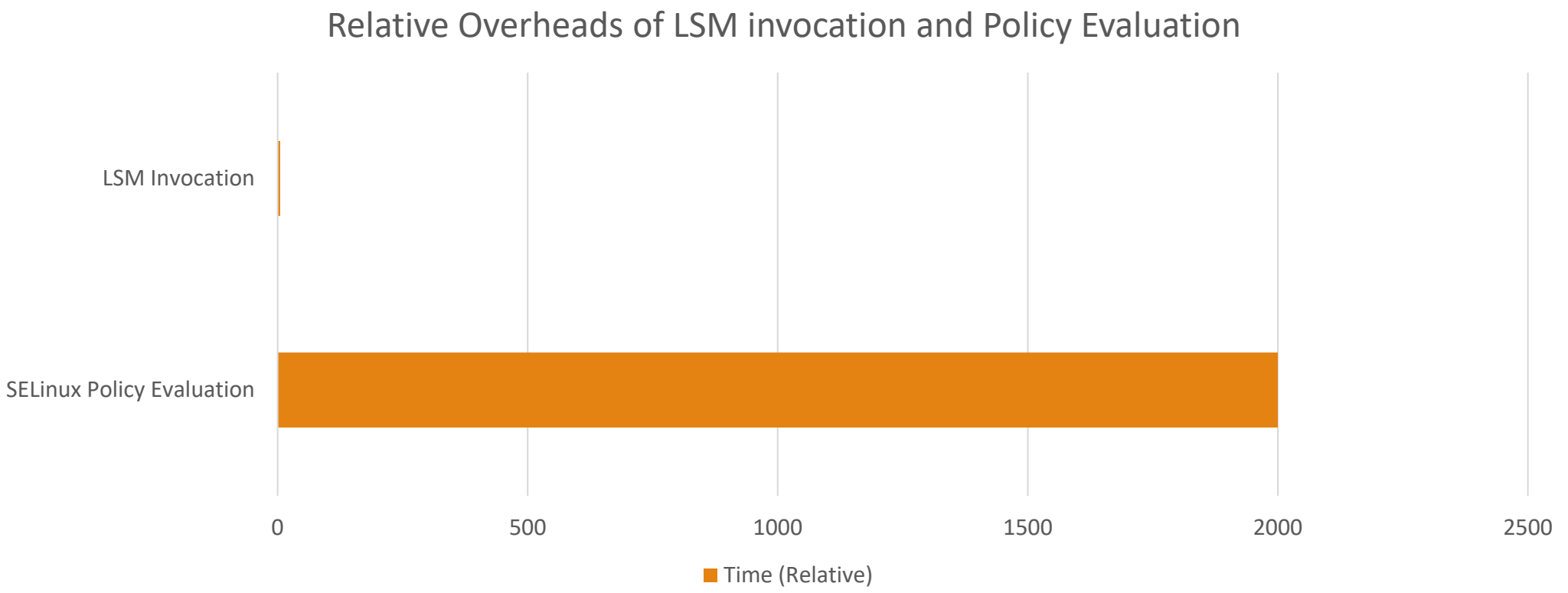


Figure 7: Overhead of the LSM, LSM-BPF, and sABPF invocation mechanisms.

Overhead of Namespacing



Evaluation Results

Test Type	vanilla	CamFlow	Overhead	ProvBPF	Overhead
Execution time (in seconds, the smaller the better)					
unpack	6.52	7.70	18%	6.59	1%
build	194.26	232.01	19%	203.70	5%
4kB to 1MB file, 10 subdirectories,4k5 simultaneous transactions, 1M5 transactions					
postmark	79.50	113.00	42%	92.50	16%

Table 5: Macrobenchmark results.

Test Type	vanilla	CamFlow	Overhead	ProvBPF	Overhead
Request/Operation per second (the higher the better)					
apache httpd	14645	10682	27%	13487	8%
redis (LPOP)	2105221	1780868	15%	1894961	10%
redis (SADD)	2073489	1721367	17%	1854162	11%
redis (LPUSH)	1630446	1401497	14%	1510000	7%
redis (GET)	2360694	1928276	18%	2102901	11%
redis (SET)	1873359	1569507	16%	1690189	10%
memcache (ADD)	44122	30444	31%	41362	6%
memcache (GET)	67895	41363	39%	62167	8%
memcache (SET)	44460	30346	32%	41355	7%
memcache (APPEND)	46730	31157	33%	43215	8%
memcache (DELETE)	67761	40735	40%	61755	9%
php	690725	613296	11%	709476	0%
Execution time (in ms, the lower the better)					
pybench	1246	1298	4%	1196	0%

Table 6: Extended macrobenchmark results.

Advantages of saBPF

Improved Performance

- Audit rule configuration occurs during eBPF compilation.

Better Maintainability

- Our provenance capture mechanism, ProvBPF, is neatly separated from the kernel, and can be built and tested independently.

Decentralized Deployment

- Each containerized environment can deploy its own LSM mechanism using saBPF without affecting the rest of the system.

Thank you!

Source code available at https://github.com/saBPF-project

Speaker: Soo Yee Lim
Affiliation: University of British Columbia
Email: sooyee@cs.ubc.ca