ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

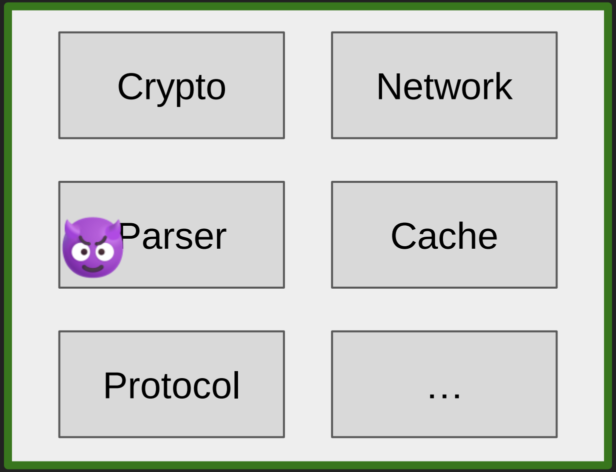
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Endokernel: A Thread Safe Monitor for Lightweight Subprocess Isolation

Fangfei Yang, Bumjin Im, Weijie Huang, Kelly Kaoudis, Anjo Vahldiek-Oberwagner, Chia-Che Tsai, Nathan Dautenhahn

Process provide isolation but when exploited enable access to the entire runtime

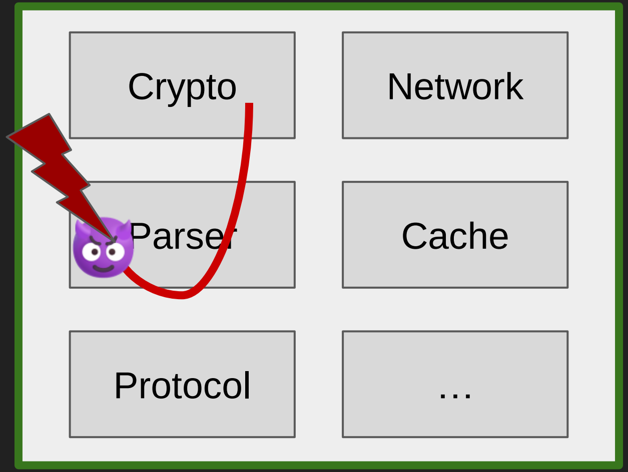
- Compartmentalize the application to improve security
- Fast, fine-grained and high-performance isolation with hardware assistance
- Monitors are used to manage the user space isolation by previous work



NGINX

Process provide isolation but when exploited enable access to the entire runtime

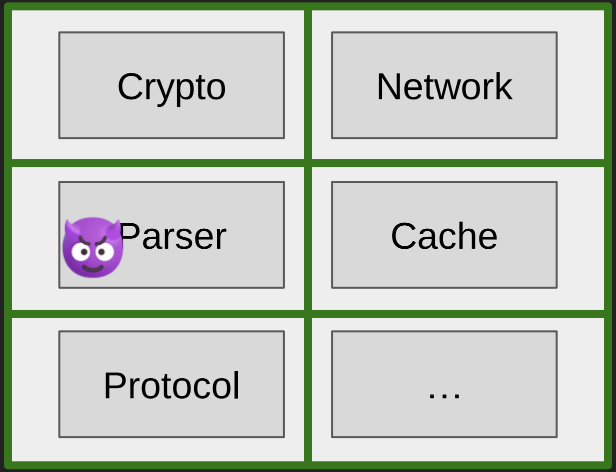
- Compartmentalize the application to improve security
- Fast, fine-grained and high-performance isolation with hardware assistance
- Monitors are used to manage the user space isolation by previous work



NGINX

Process provide isolation but when exploited enable access to the entire runtime

- Compartmentalize the application to improve security
- Fast, fine-grained and high-performance isolation with hardware assistance
- Monitors are used to manage the user space isolation by previous work



Crypto

Network

Parser

Cache

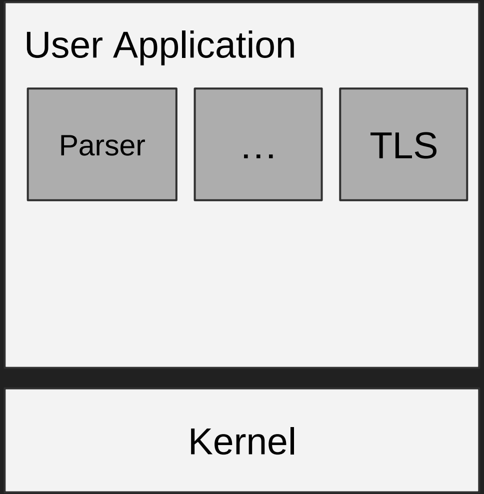
Protocol

…

NGINX

Privilege Separation with In-process Secure Monitor

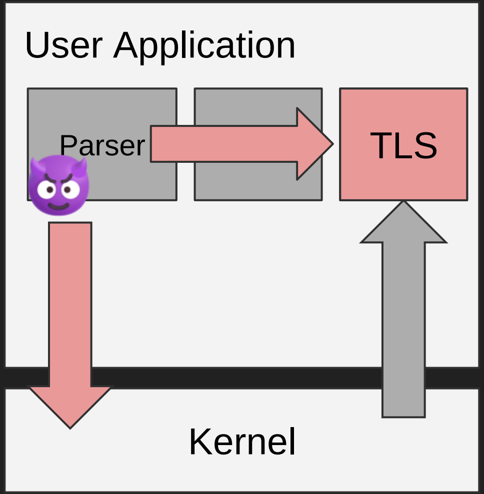
- Kernel **unaware** of isolation policy and violate the policy
  - Filtering syscalls to ensure the kernel doesn't break the isolation policy in user space
- Monitor determines whether the system call is legitimate
- BUT, making the right policy decisions in **multithreading** is harder than you think
- **=>** "Secure" Monitor is **NOT** actually secure



Secure Monitor **itself** becomes the problem!

Privilege Separation with In-process Secure Monitor

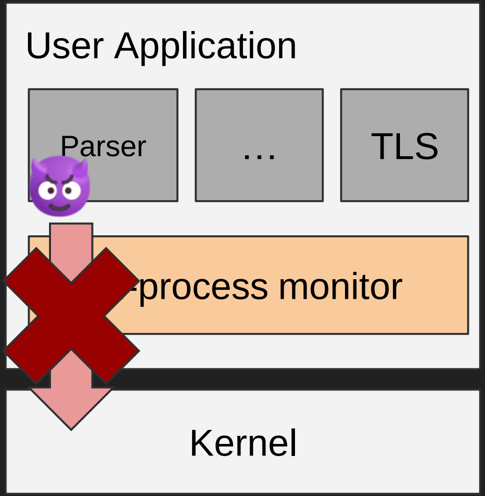
- Kernel **unaware** of isolation policy and violate the policy
  - Filtering syscalls to ensure the kernel doesn't break the isolation policy in user space
- Monitor determines whether the system call is legitimate
- BUT, making the right policy decisions in **multithreading** is harder than you think
- **=>** "Secure" Monitor is **NOT** actually secure



Secure Monitor **itself** becomes the problem!

Privilege Separation with In-process Secure Monitor

- Kernel **unaware** of isolation policy and violate the policy
  - Filtering syscalls to ensure the kernel doesn't break the isolation policy in user space
- Monitor determines whether the system call is legitimate
- BUT, making the right policy decisions in **multithreading** is harder than you think
- **=>** "Secure" Monitor is **NOT** actually secure

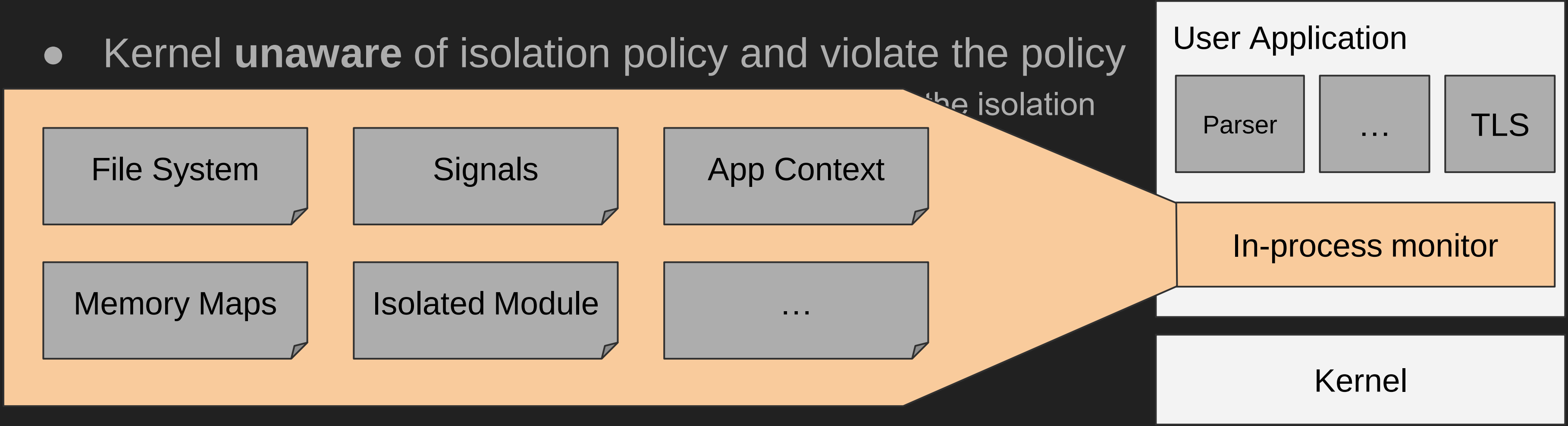

User Application
Parser
…
TLS
-process monitor
Kernel

Secure Monitor **itself** becomes the problem!

Privilege Separation with In-process Secure Monitor

- Kernel **unaware** of isolation policy and violate the policy

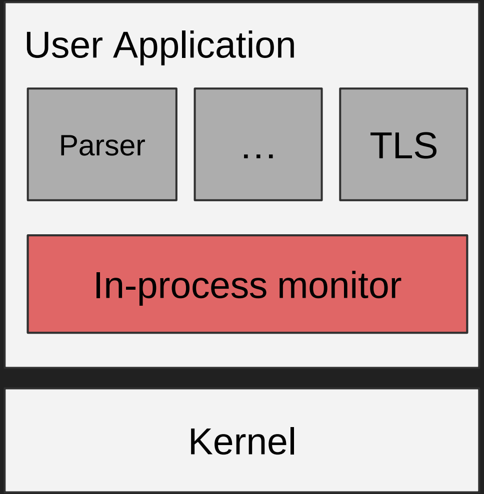
he isolation


File System
Signals
App Context
Memory Maps
Isolated Module
…
User Application
Parser
…
TLS
In-process monitor
Kernel

Secure Monitor **itself** becomes the problem!

Privilege Separation with In-process Secure Monitor

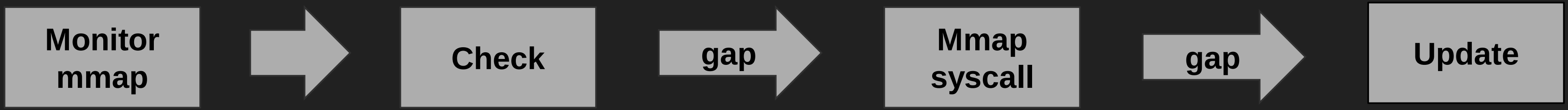
- Kernel **unaware** of isolation policy and violate the policy
  - Filtering syscalls to ensure the kernel doesn't break the isolation policy in user space
- Monitor determines whether the system call is legitimate
- BUT, making the right policy decisions in **multithreading** is harder than you think
- **=>** "Secure" Monitor is **NOT** actually secure



Secure Monitor **itself** becomes the problem!

Existing Works Fails to Secure Multi-threaded Monitor

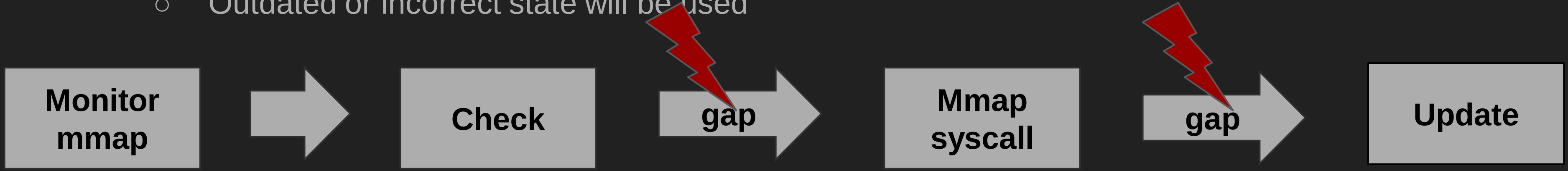
- Monitor needs **truth** about the system to make right decision
  - Which memory address belongs to whom? Is this file descriptor valid? …
- System states **changed** via syscalls and signals: easy if only **one** thread
- Gap: changes in state and updates in the monitor are **never synchronized**
  - The kernel maintains its internal consistency but not for the in-process monitor
  - Outdated or incorrect state will be used



Monitor makes decisions based on **incorrect** information!

Existing Works Fails to Secure Multi-threaded Monitor

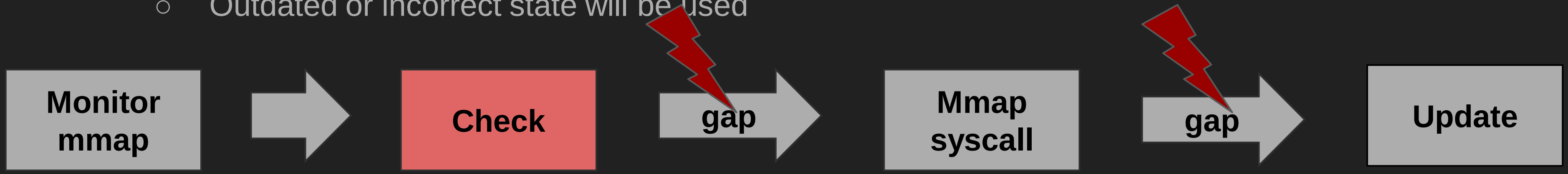
- Monitor needs **truth** about the system to make right decision
  - Which memory address belongs to whom? Is this file descriptor valid? …
- System states **changed** via syscalls and signals: easy if only **one** thread
- Gap: changes in state and updates in the monitor are **never synchronized**
  - The kernel maintains its internal consistency but not for the in-process monitor
  - Outdated or incorrect state will be used



Monitor makes decisions based on **incorrect** information!

Existing Works Fails to Secure Multi-threaded Monitor

- Monitor needs **truth** about the system to make right decision
  - Which memory address belongs to whom? Is this file descriptor valid? …
- System states **changed** via syscalls and signals: easy if only **one** thread
- Gap: changes in state and updates in the monitor are **never synchronized**
  - The kernel maintains its internal consistency but not for the in-process monitor
  - Outdated or incorrect state will be used

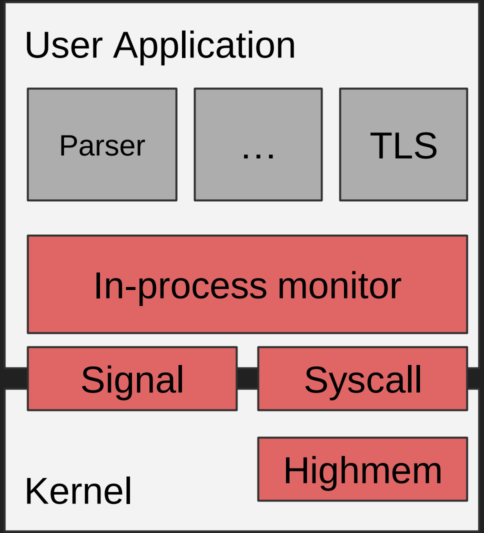


Monitor makes decisions based on **incorrect** information!

Endokernel Design

Challenge: The kernel does not cooperate with the monitor

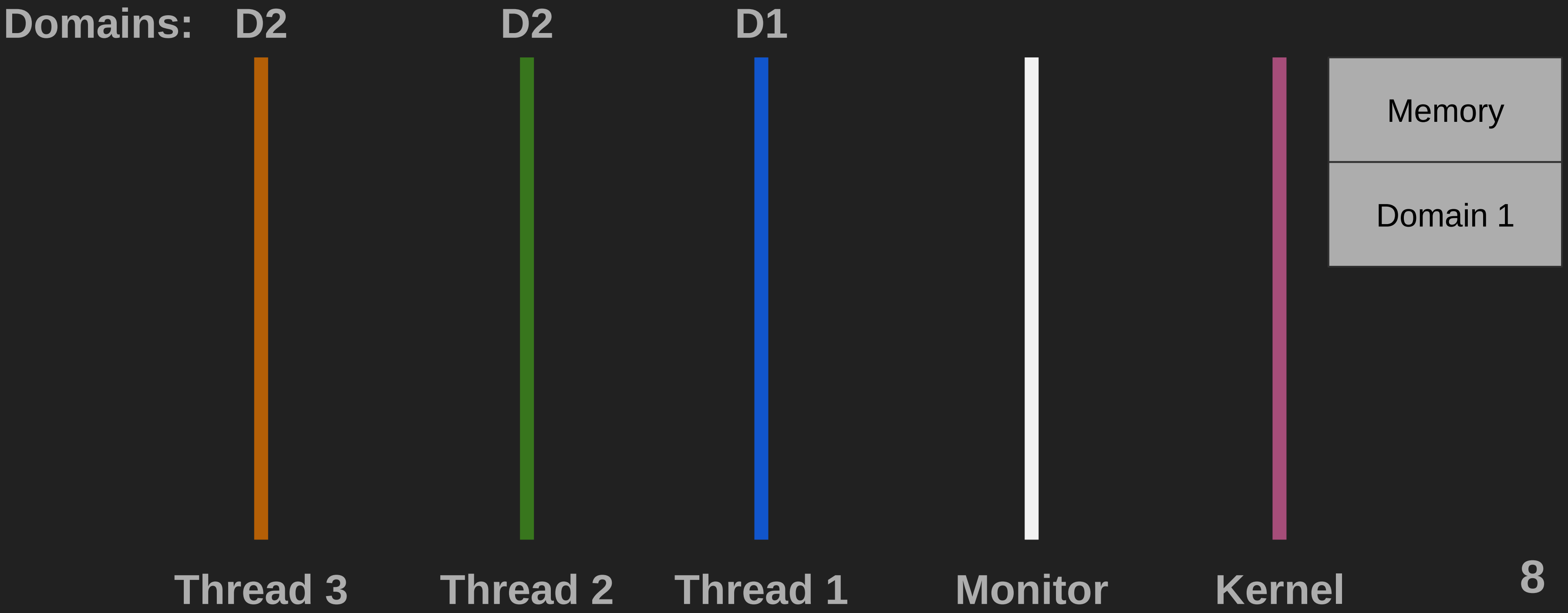
- General Syscalls: memory metadata, file descriptors
  - open/read/write/mmap/mprotect/…
  - States change before/after syscalls
- Signals: Kernel-involved context switches
  - Signal delivery and sigreturn can alter control flow and privilege
- Highmem: access physical memory and bypass checks
  - Hidden, complex, delayed and overlooked
  - Requires case-by-case analysis and solutions



Endokernel – Build a thread-safe monitor!

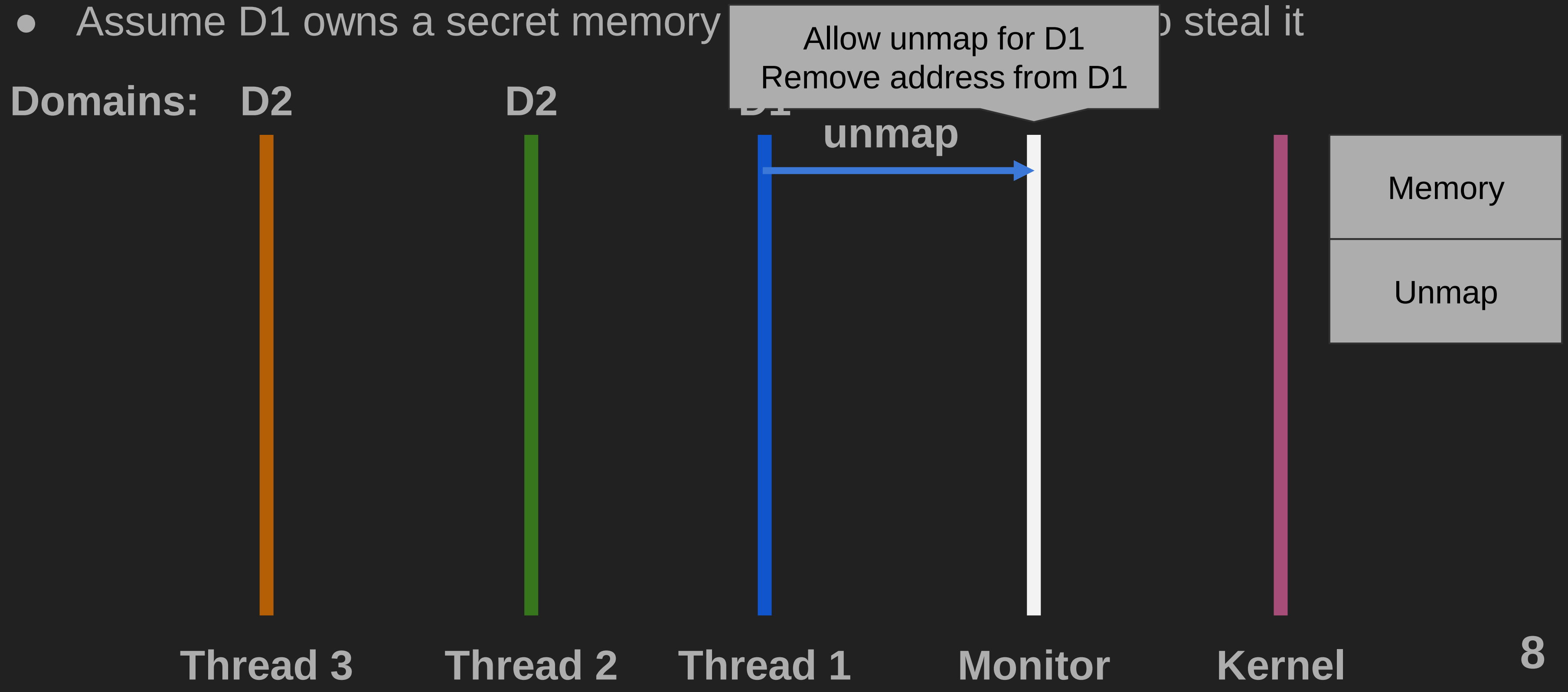
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



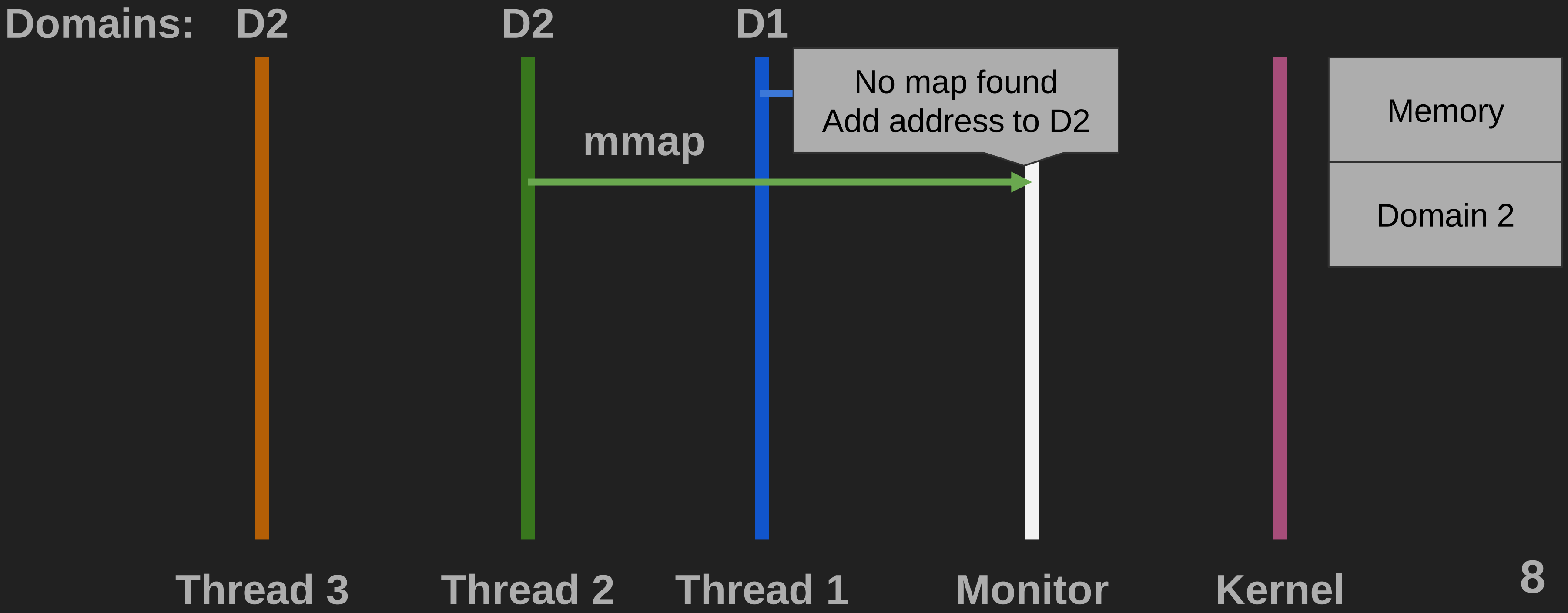
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory steal it



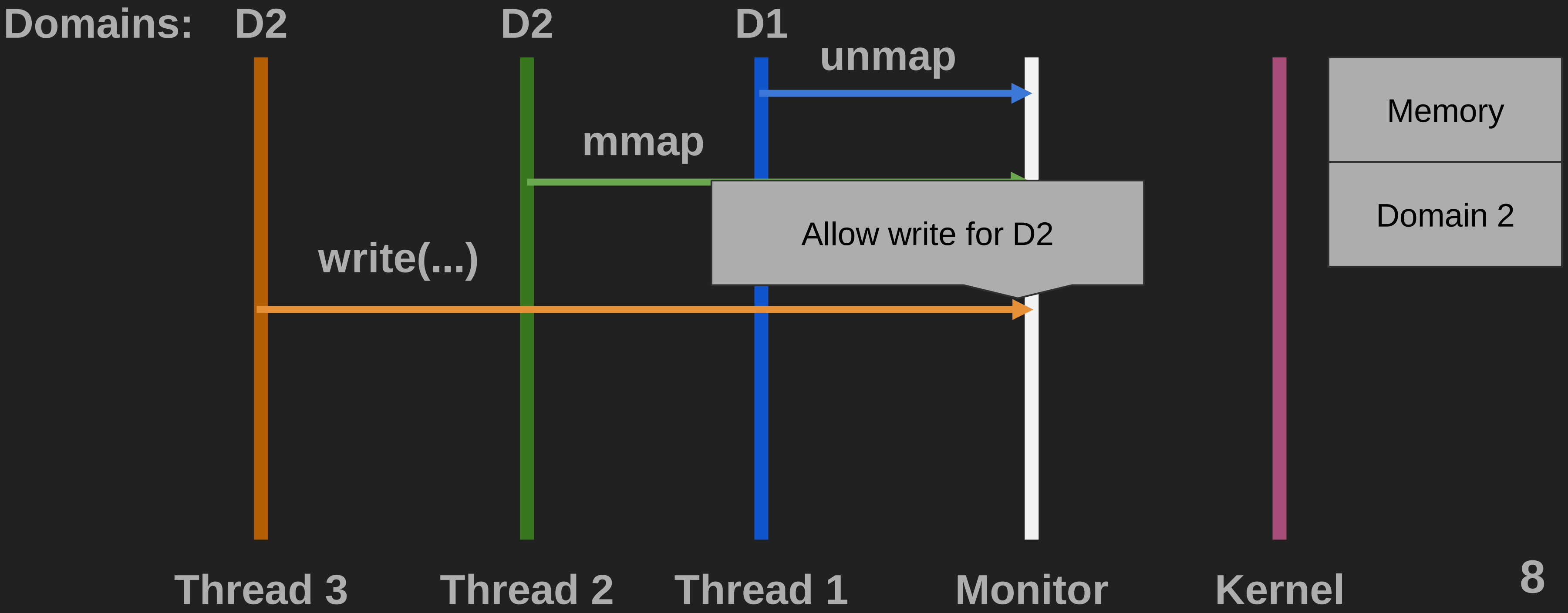
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



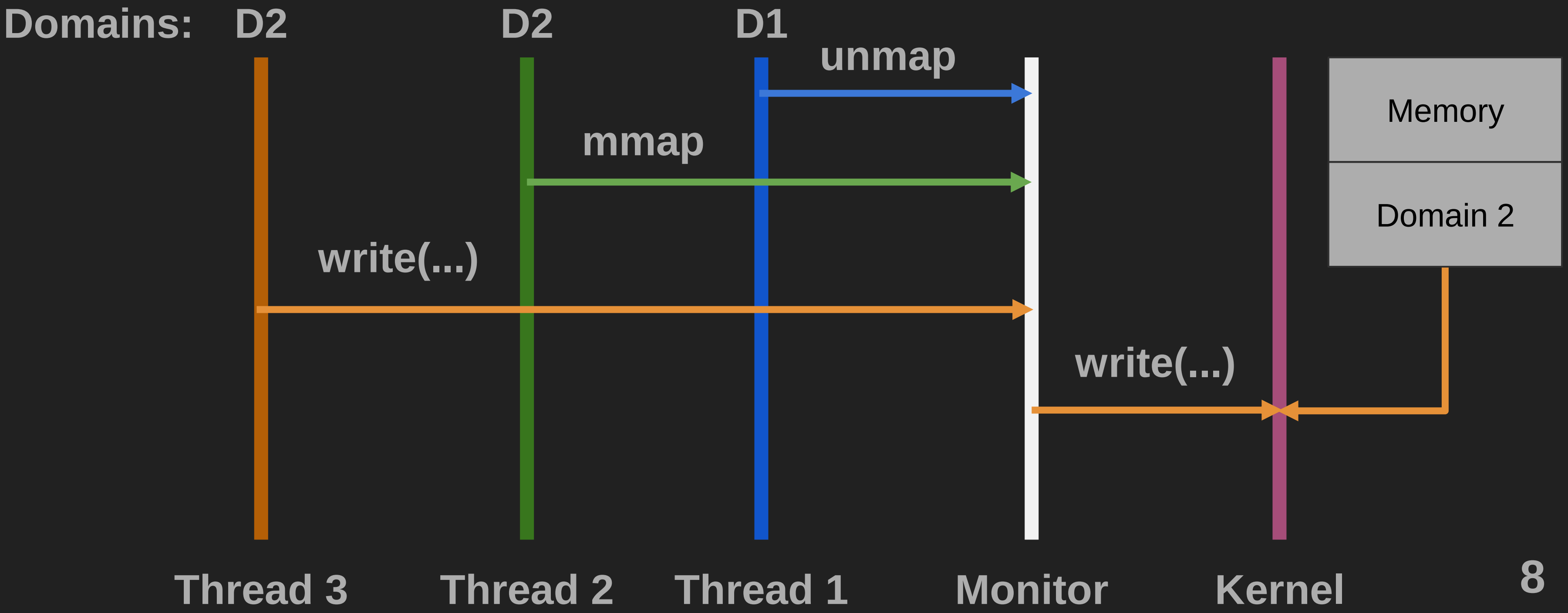
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



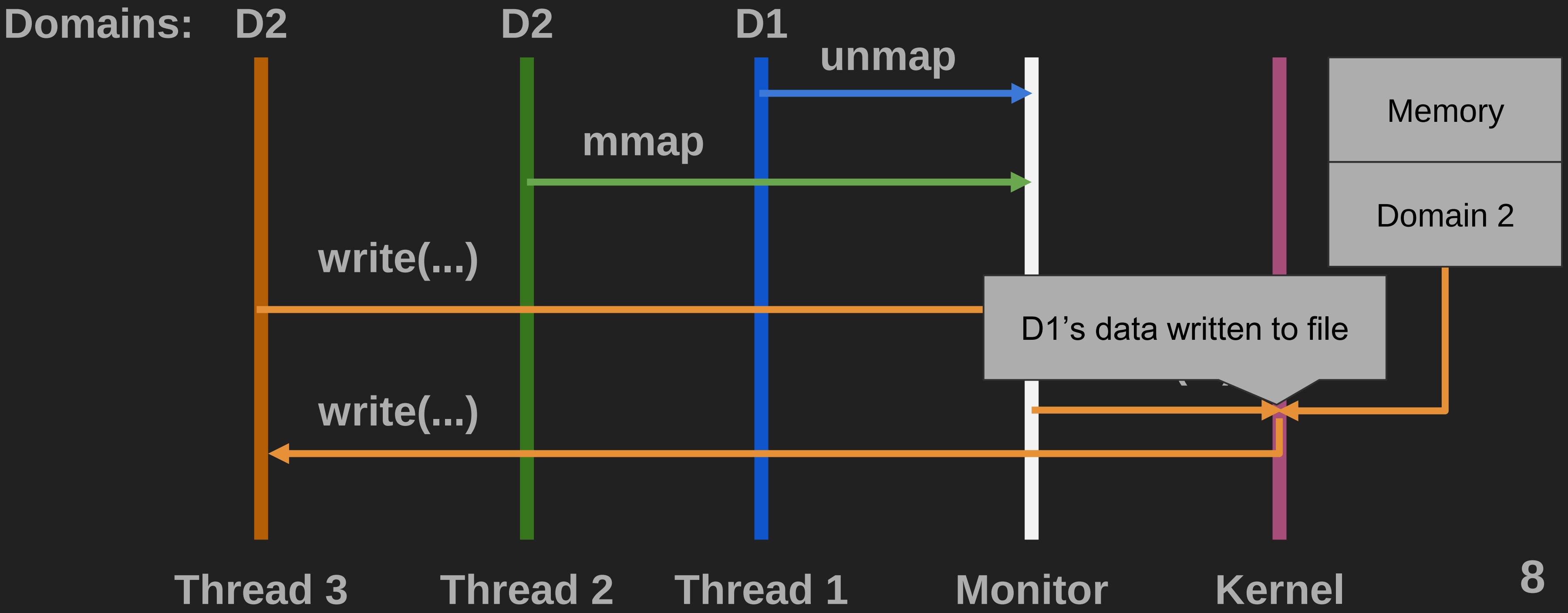
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



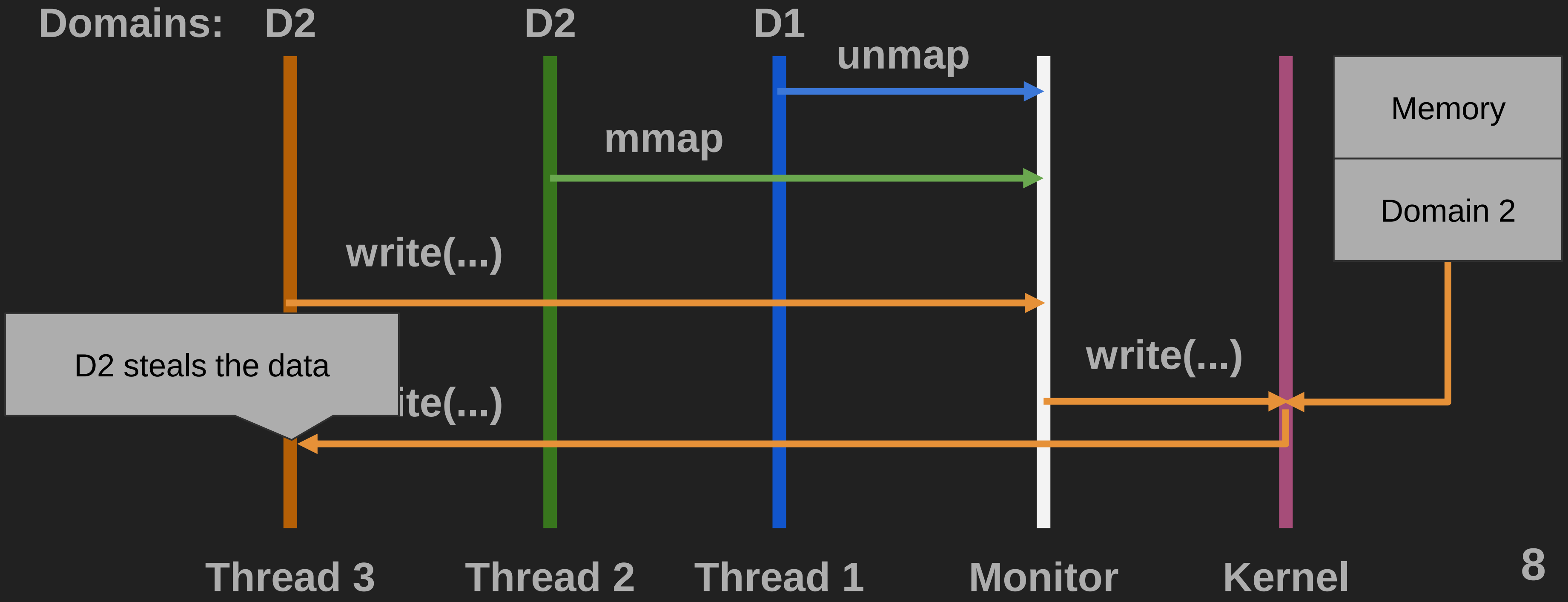
General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



General Syscalls: Monitor/Kernel Synchronization

- Assume D1 owns a secret memory region, and D2 wants to steal it



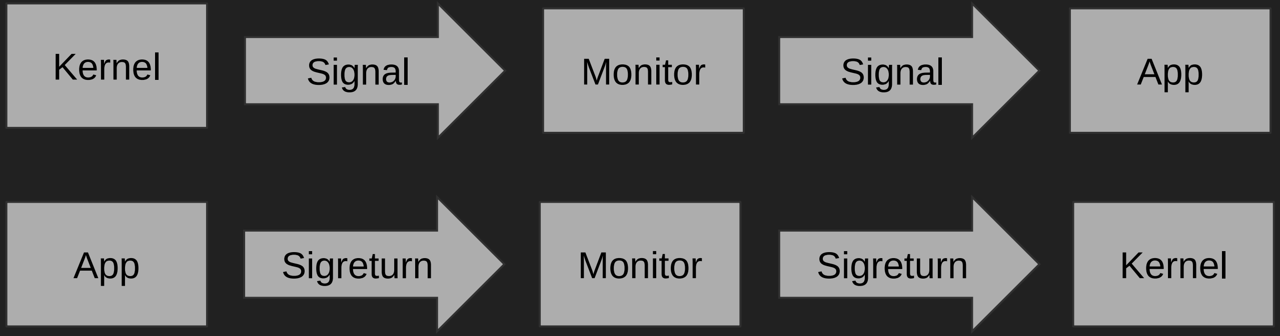
Solution: Weak Metadata Synchronization

- Tolerate inconsistencies before and after system calls; ensure they **only lead to inspection failures**
- Mark pages involved in system calls; block other calls that would change their properties while the memory is in use
- Allow concurrent invocation of system calls if they don't alter page properties
- Ensure **correct decisions** are made, even with Kernel-Endokernel inconsistencies, without violating policy.

Desynchronization **never** violates security policies

Signal: Intermediate States Exposed by Sigreturn

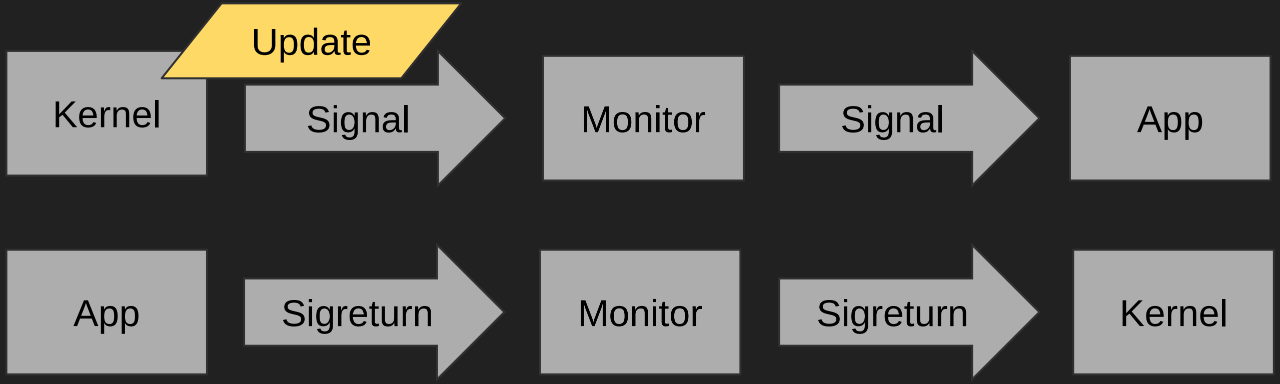
- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches


Kernel
Signal
Monitor
Signal
App

App
Sigreturn
Monitor
Sigreturn
Kernel

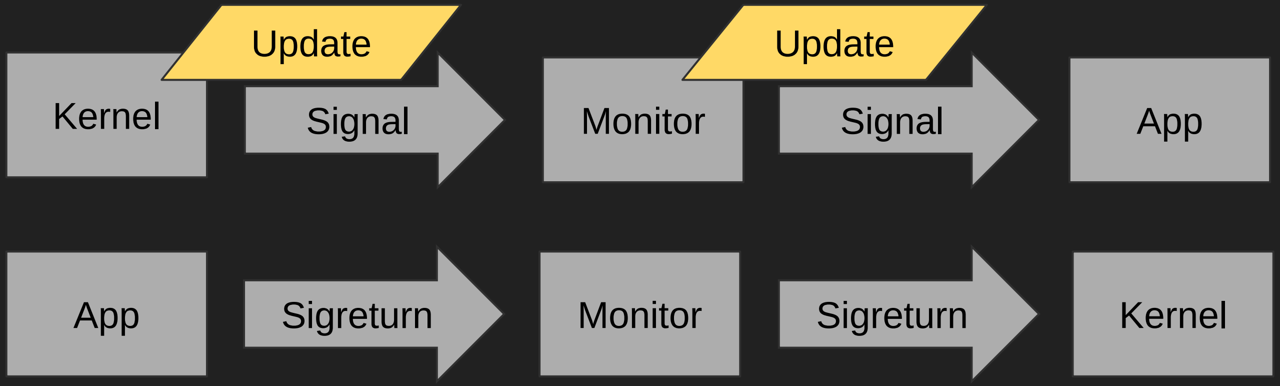
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



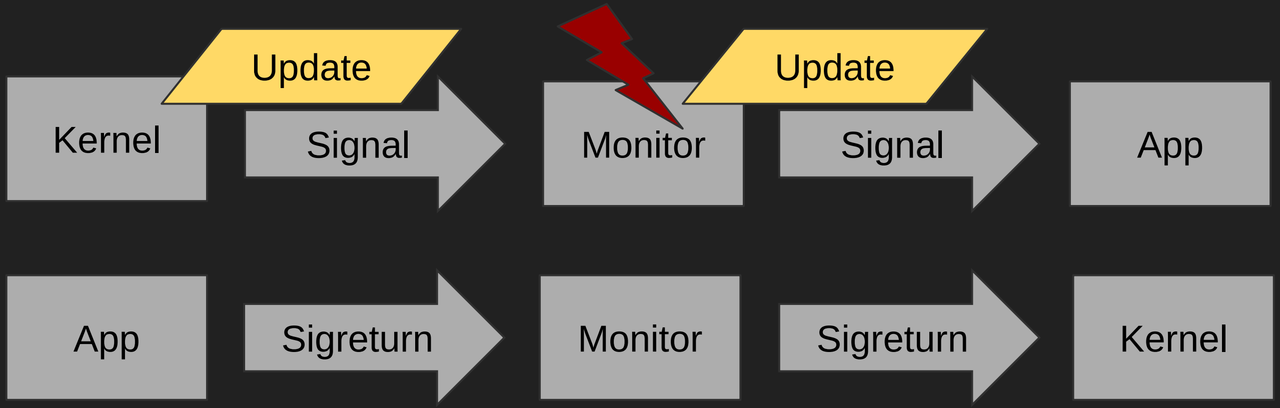
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



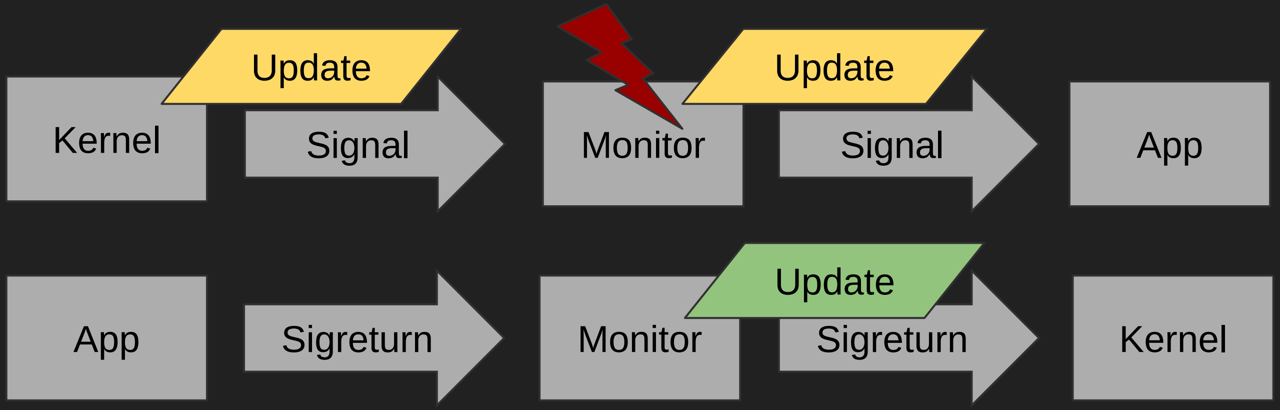
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



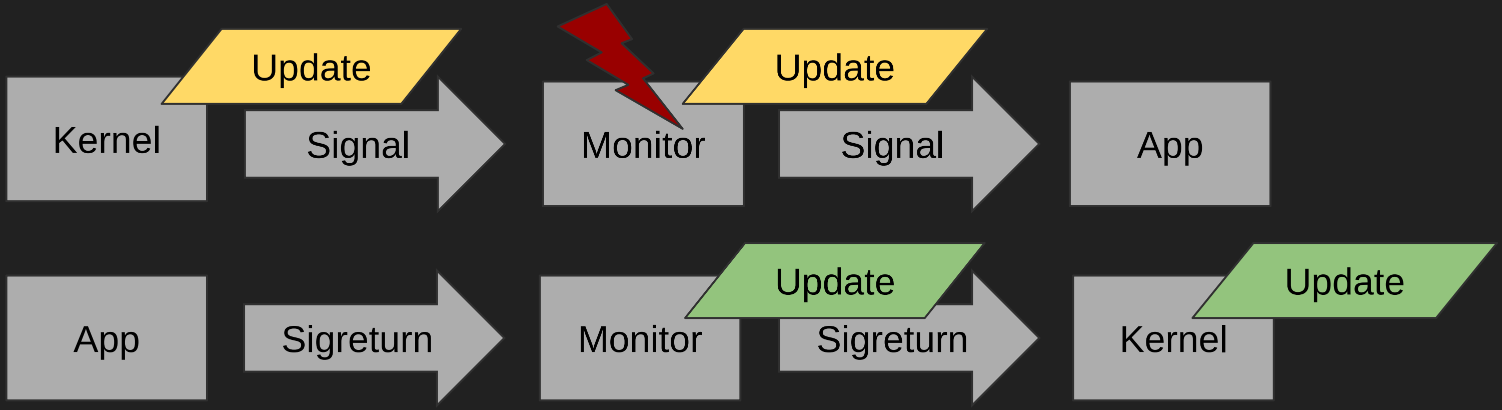
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



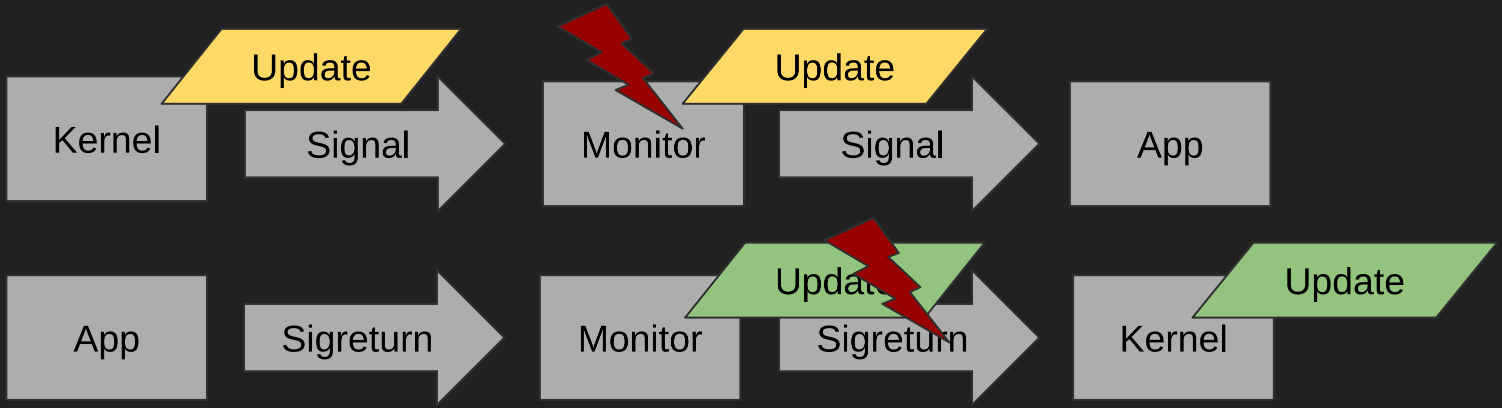
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



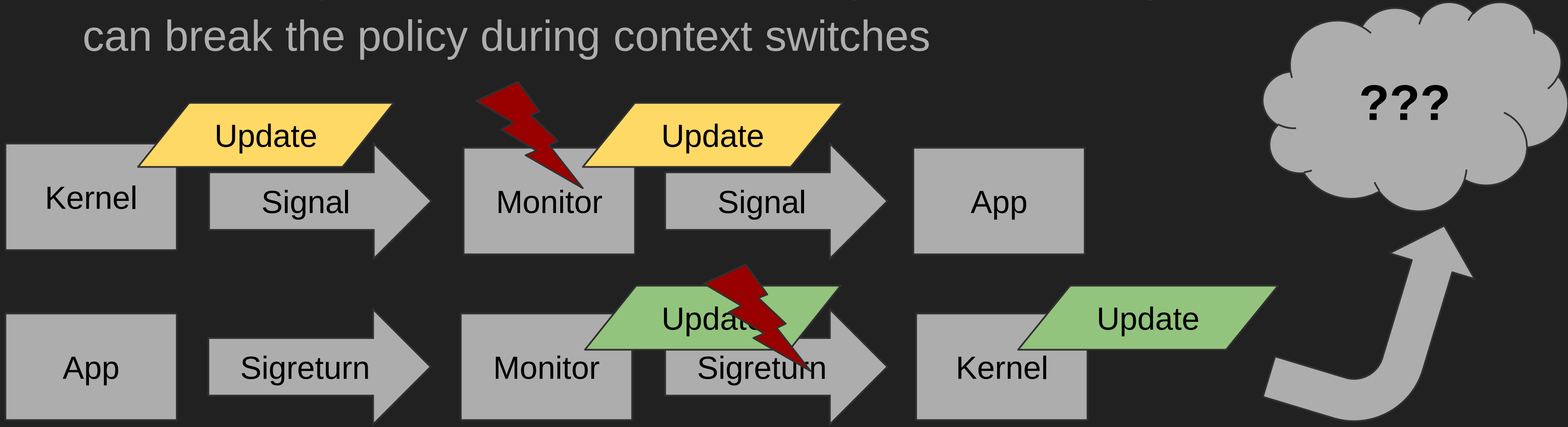
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



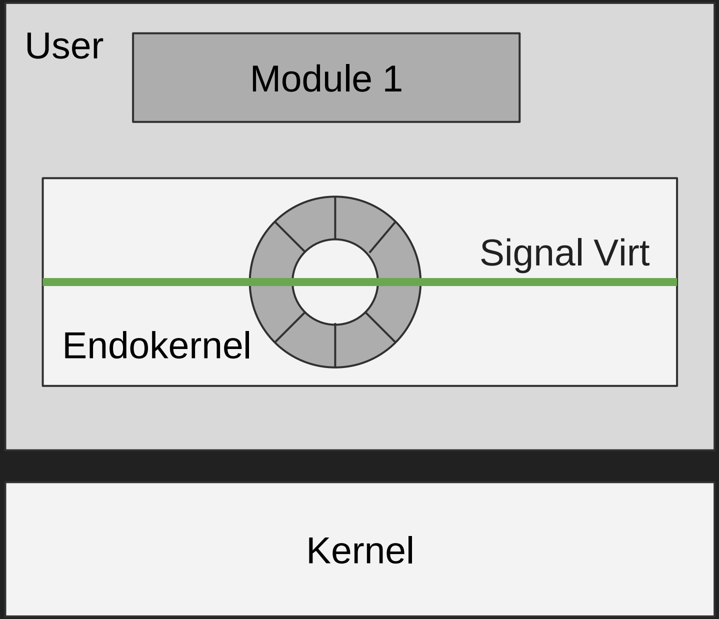
Signal: Intermediate States Exposed by Sigreturn

- Signal delivery and return are meant to switch contexts
- Different contexts have different permissions defined by the policy
- Unfortunately, the kernel cannot correctly handle these permissions, and can break the policy during context switches



Solution: Fully Virtualized Signal

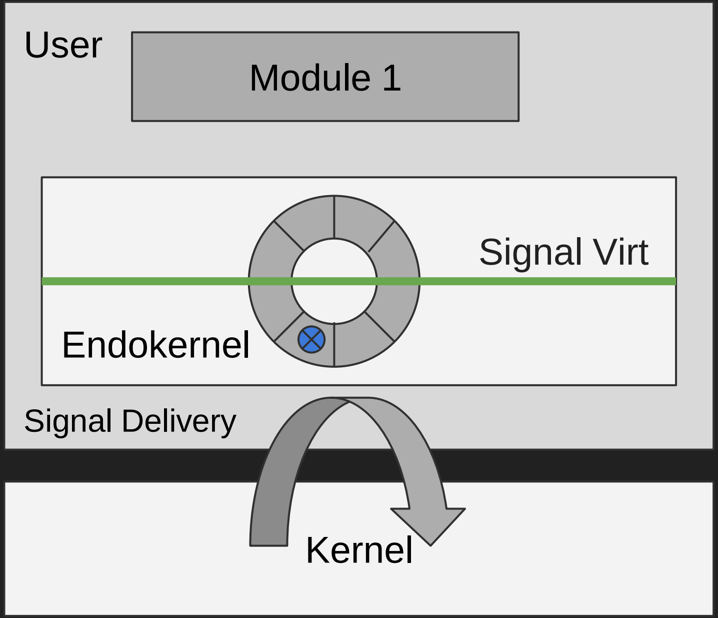
- Endokernel acts as a **middleware**
- Endokernel receives signals from the kernel
  - Stores signals in a pending queue
  - Returns control to the kernel with sigreturn
- Endokernel delivers signals to the user
  - Creates a new sigcontext and sigframe.
  - Simulates the user's sigreturn syscall



Virtualized secure and compatible signals

Solution: Fully Virtualized Signal

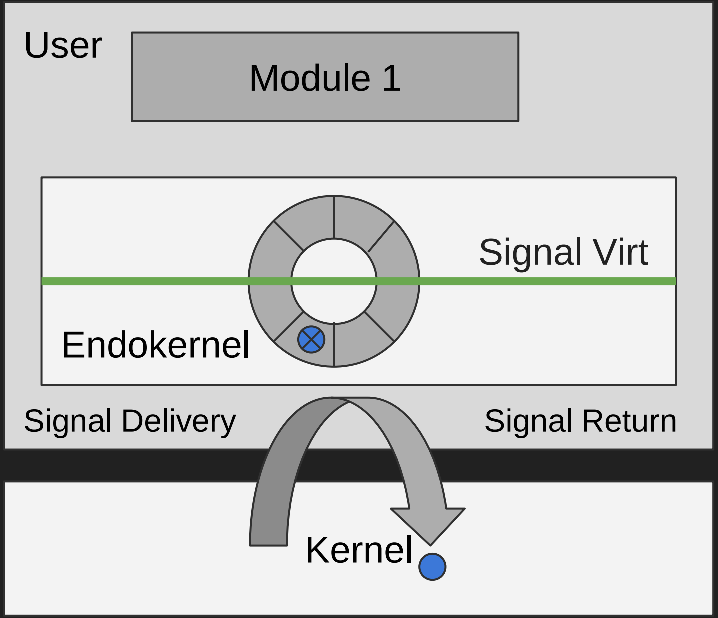
- Endokernel acts as a **middleware**
- Endokernel receives signals from the kernel
  - Stores signals in a pending queue
  - Returns control to the kernel with sigreturn
- Endokernel delivers signals to the user
  - Creates a new sigcontext and sigframe.
  - Simulates the user's sigreturn syscall



Virtualized secure and compatible signals

Solution: Fully Virtualized Signal

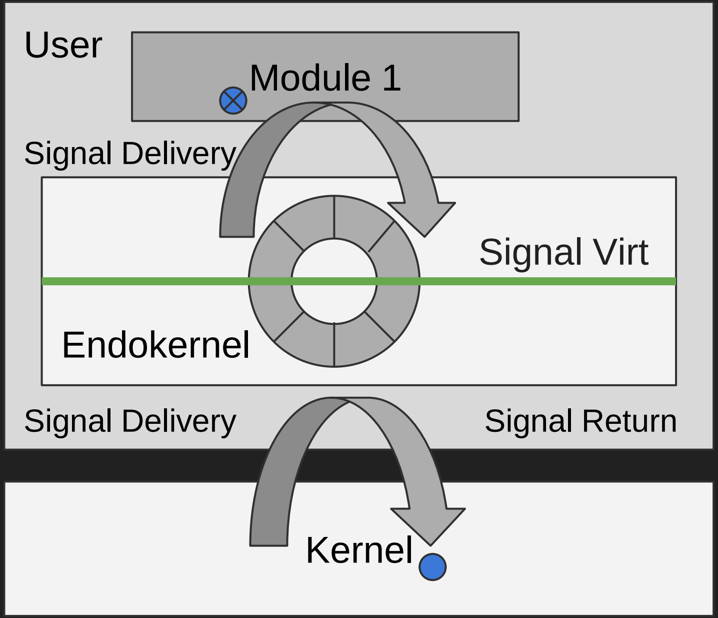
- Endokernel acts as a **middleware**
- Endokernel receives signals from the kernel
  - Stores signals in a pending queue
  - Returns control to the kernel with sigreturn
- Endokernel delivers signals to the user
  - Creates a new sigcontext and sigframe.
  - Simulates the user's sigreturn syscall



Virtualized secure and compatible signals

Solution: Fully Virtualized Signal

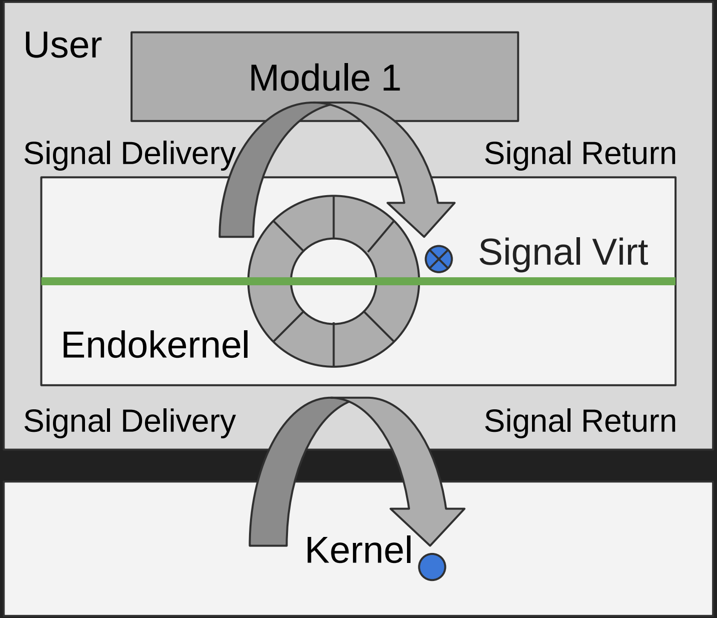
- Endokernel acts as a **middleware**
- Endokernel receives signals from the kernel
  - Stores signals in a pending queue
  - Returns control to the kernel with sigreturn
- Endokernel delivers signals to the user
  - Creates a new sigcontext and sigframe.
  - Simulates the user's sigreturn syscall



Virtualized secure and compatible signals

Solution: Fully Virtualized Signal

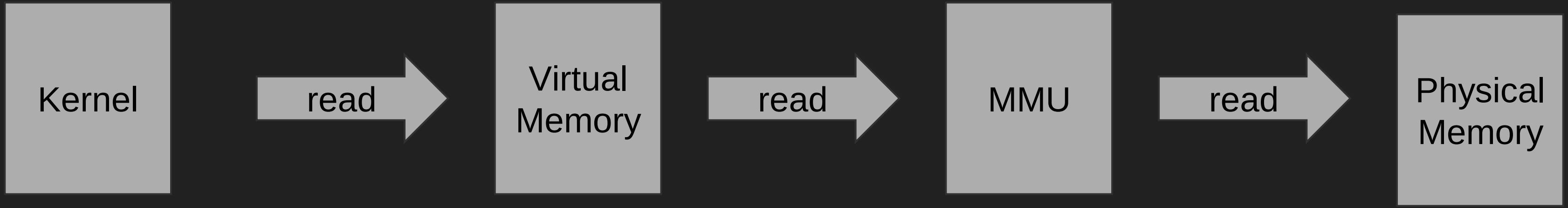
- Endokernel acts as a **middleware**
- Endokernel receives signals from the kernel
  - Stores signals in a pending queue
  - Returns control to the kernel with sigreturn
- Endokernel delivers signals to the user
  - Creates a new sigcontext and sigframe.
  - Simulates the user's sigreturn syscall



Virtualized secure and compatible signals

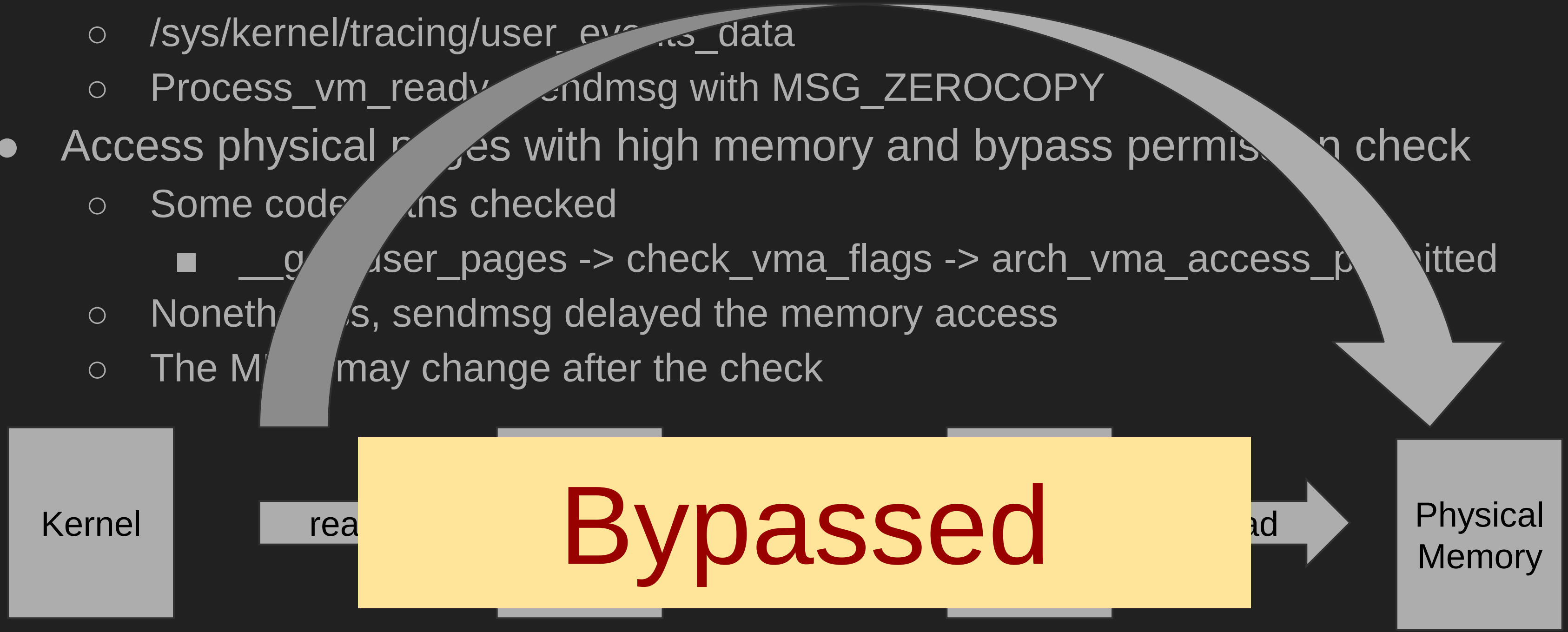
Highmem: Bypass Pattern and Delayed Memory Access

- Various triggering mechanisms
  - /sys/kernel/tracing/user_events_data
  - Process_vm_readv, Sendmsg with MSG_ZEROCOPY
- Access physical pages with high memory and bypass permission check
  - Some code paths checked
    - __get_user_pages -> check_vma_flags -> arch_vma_access_permitted
  - Nonetheless, sendmsg delayed the memory access
  - The MMU may change after the check



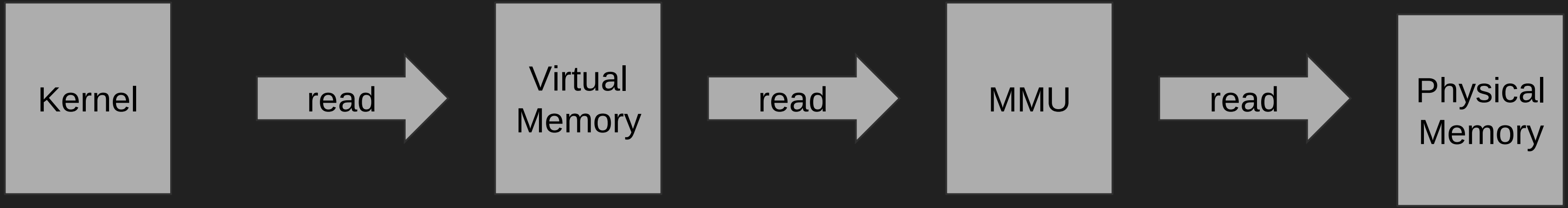
Highmem: Bypass Pattern and Delayed Memory Access

- Various triggering mechanisms
  - /sys/kernel/tracing/user_ev ts_data
  - Process_vm_ready endmsg with MSG_ZEROCOPY
- Access physical p ges with high memory and bypass permis n check
  - Some code ths checked
    - __g user_pages -> check_vma_flags -> arch_vma_access_p itted
  - Noneth s, sendmsg delayed the memory access
  - The M may change after the check



Highmem: Bypass Pattern and Delayed Memory Access

- Various triggering mechanisms
  - /sys/kernel/tracing/user_events_data
  - Process_vm_readv, Sendmsg with MSG_ZEROCOPY
- Access physical pages with high memory and bypass permission check
  - Some code paths checked
    - __get_user_pages -> check_vma_flags -> arch_vma_access_permitted
  - Nonetheless, sendmsg delayed the memory access
  - The MMU may change after the check



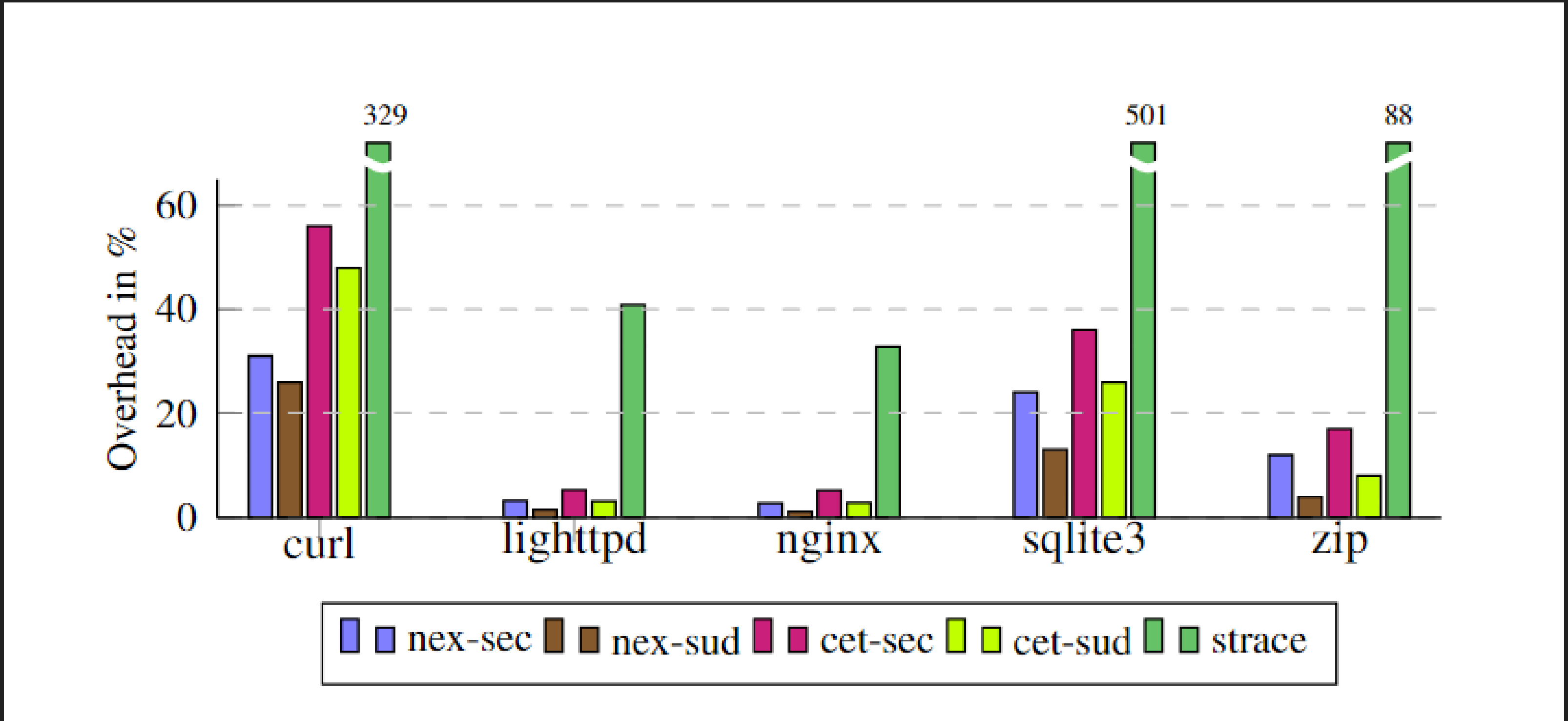
Solution: Extra Policy with Syscalls Analysis

- Traced the syscalls in the kernel that use certain APIs most of which are related to driver and ioctl
- Restrictions need to be applied based on specific use cases
  - For example, adding extra policies to prohibit the use of zero copy or prevent the memory from being unmapped
- Kernel features that improve efficiency can make in-process monitoring more challenging

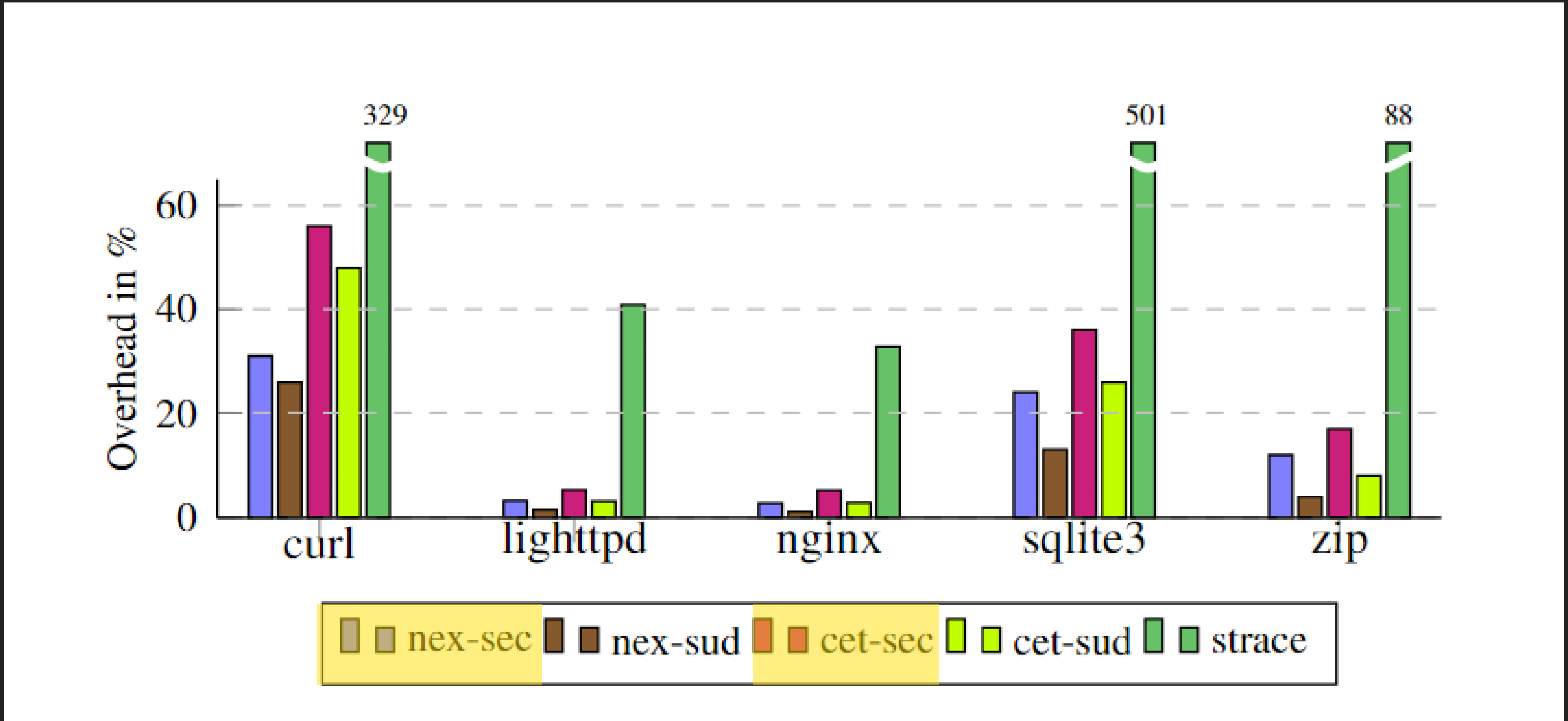
Identified **patterns**, allowing for case-by-case analysis

Evaluation

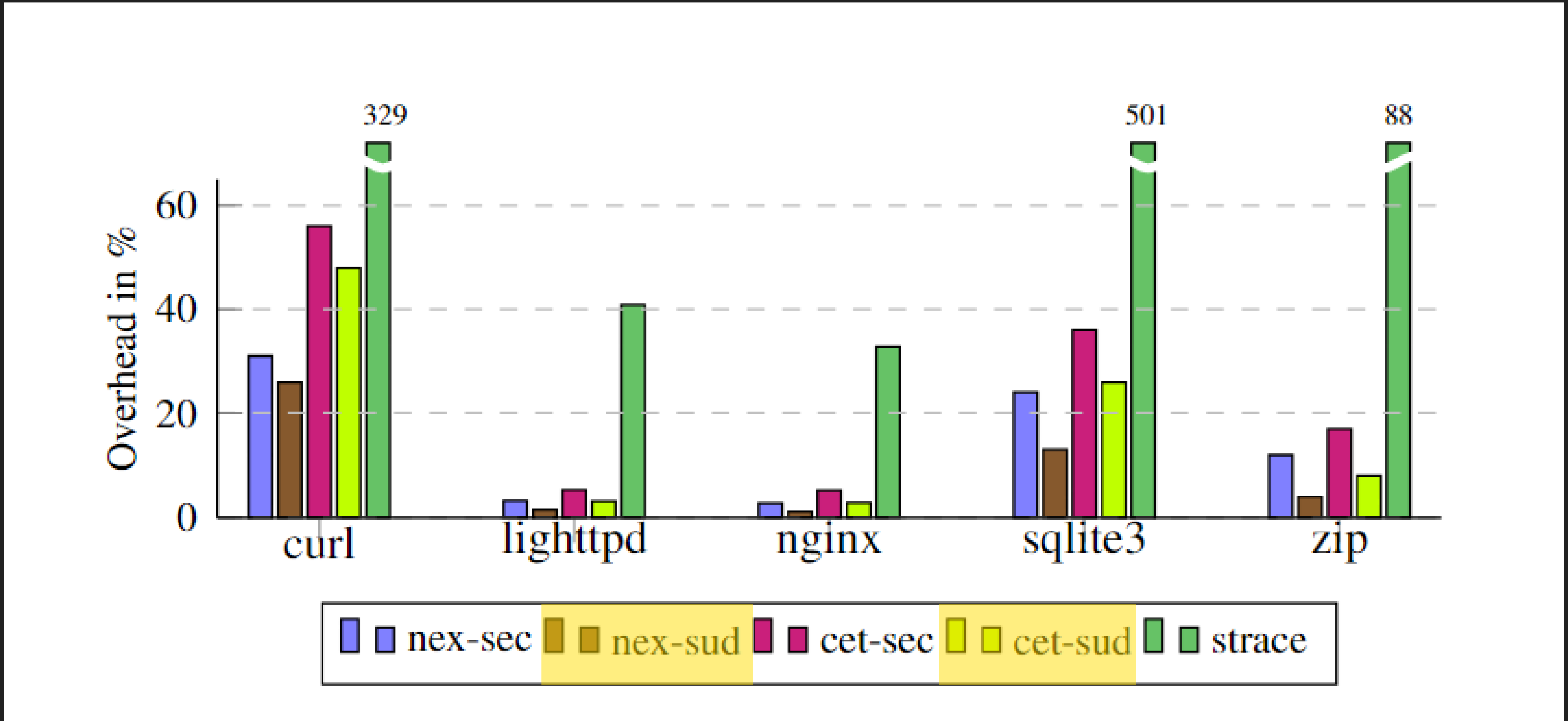
Virtualization Cost with Different Callgate Mechanism



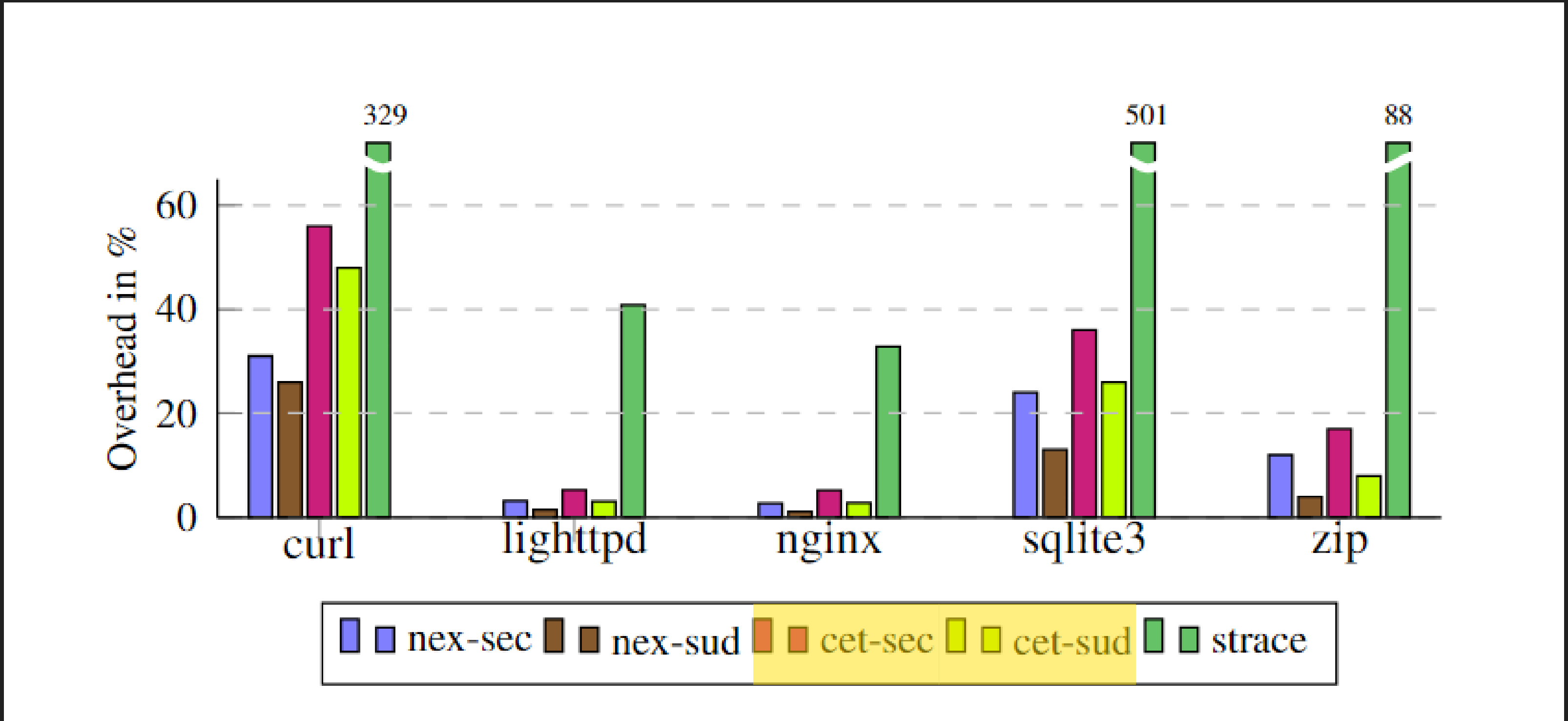
Virtualization Cost with Different Callgate Mechanism



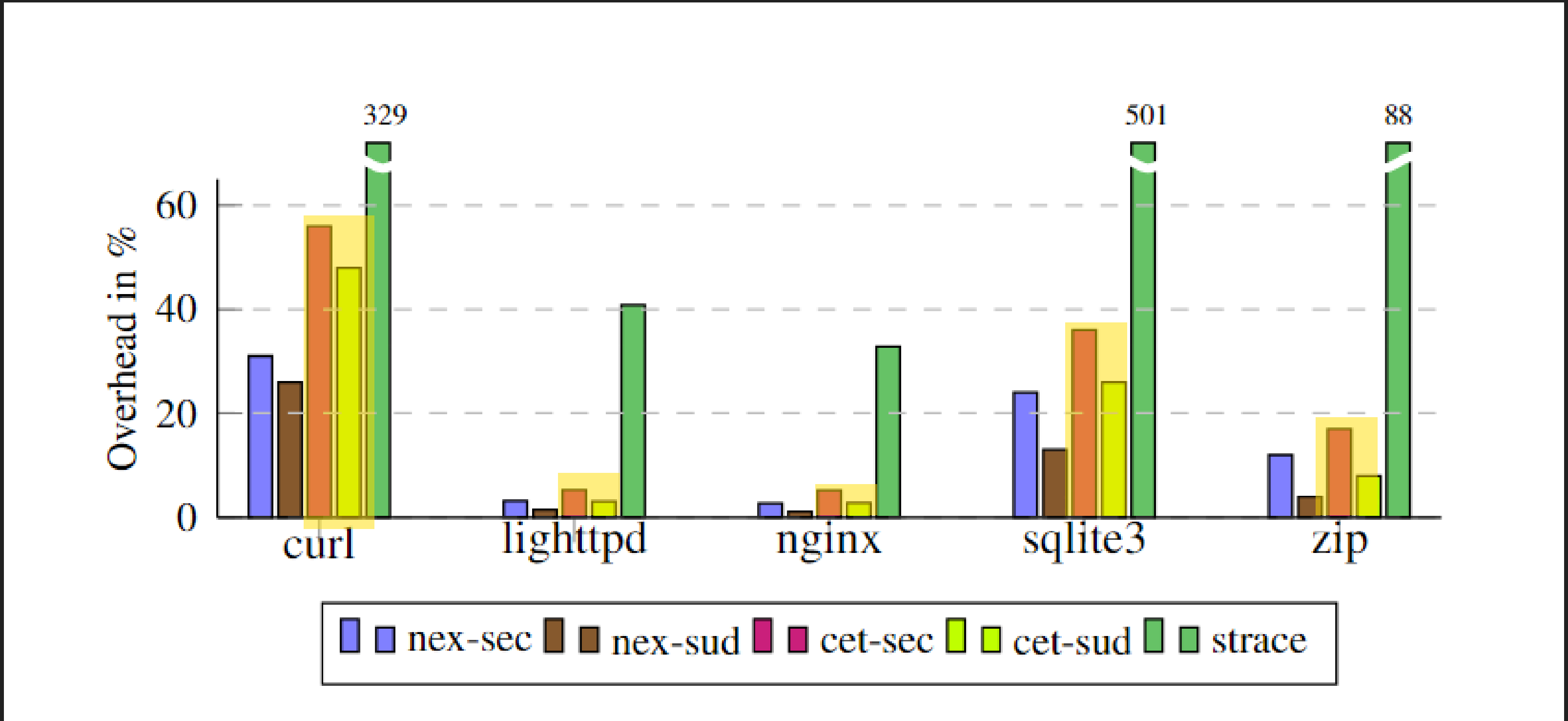
Virtualization Cost with Different Callgate Mechanism



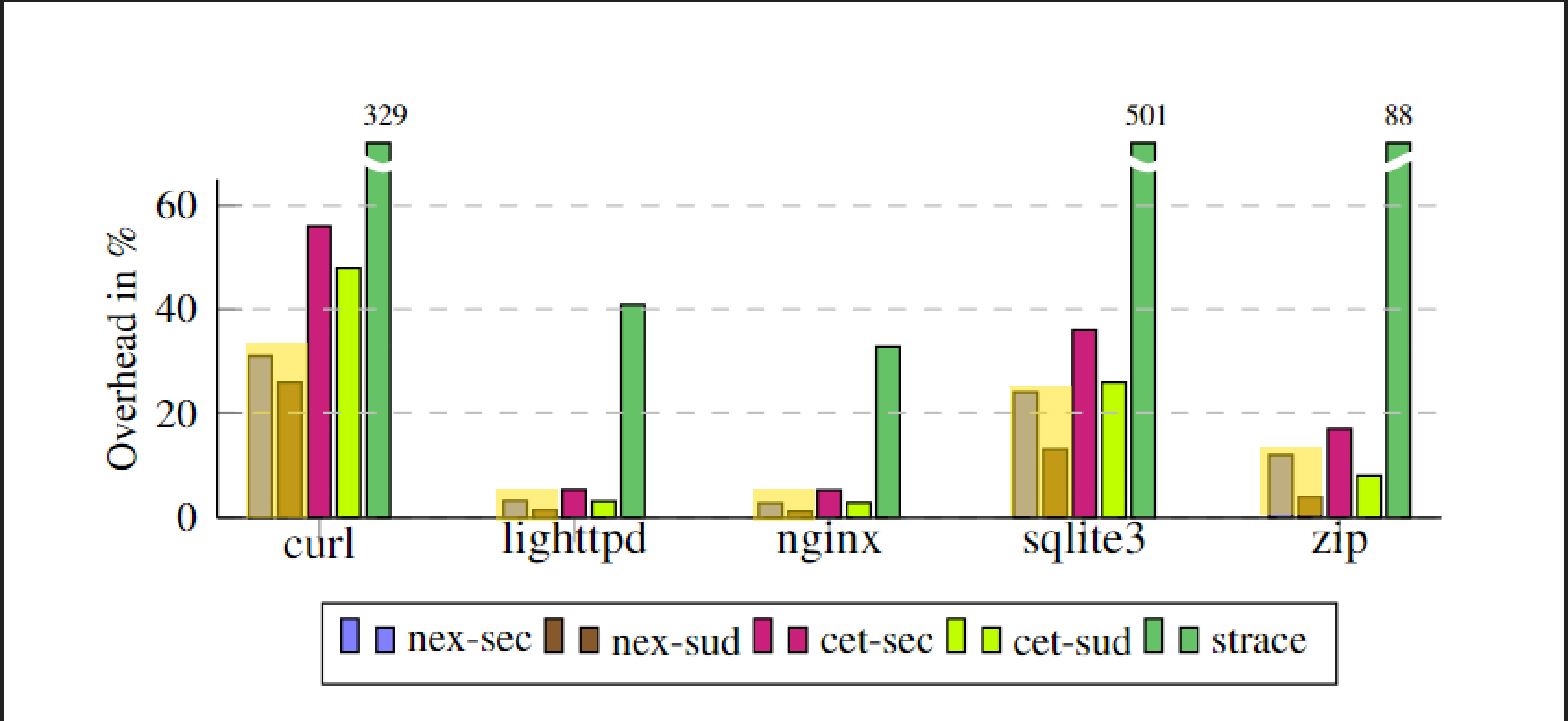
Virtualization Cost with Different Callgate Mechanism



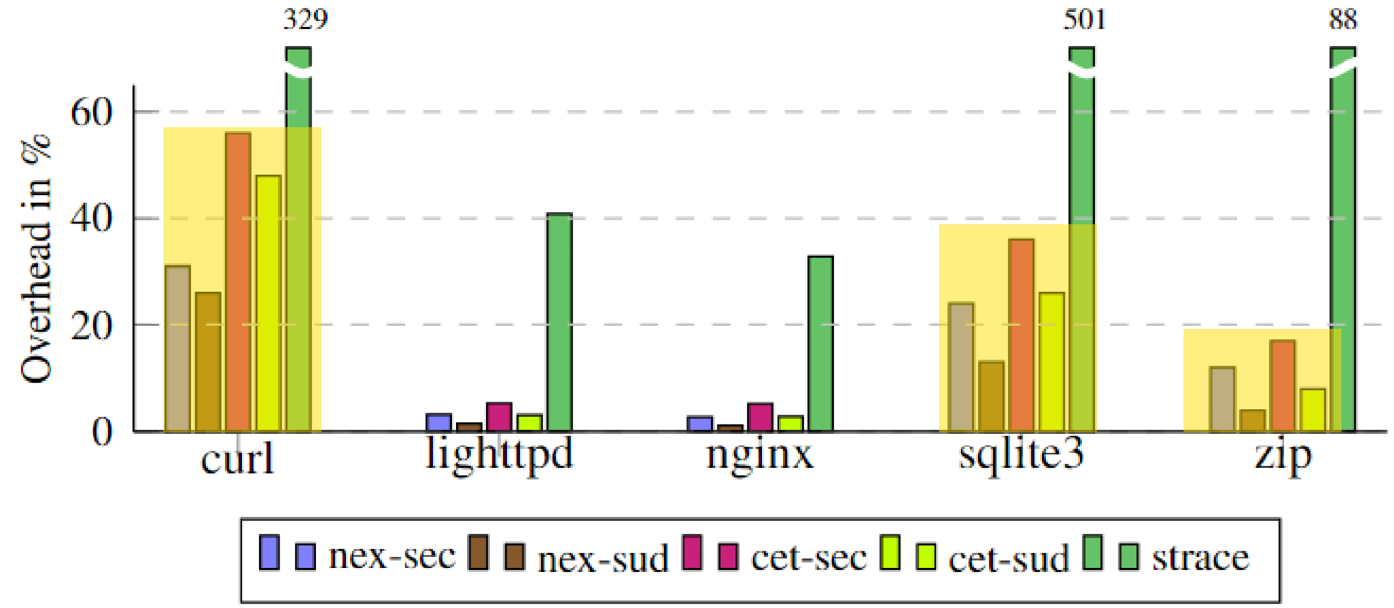
Virtualization Cost with Different Callgate Mechanism



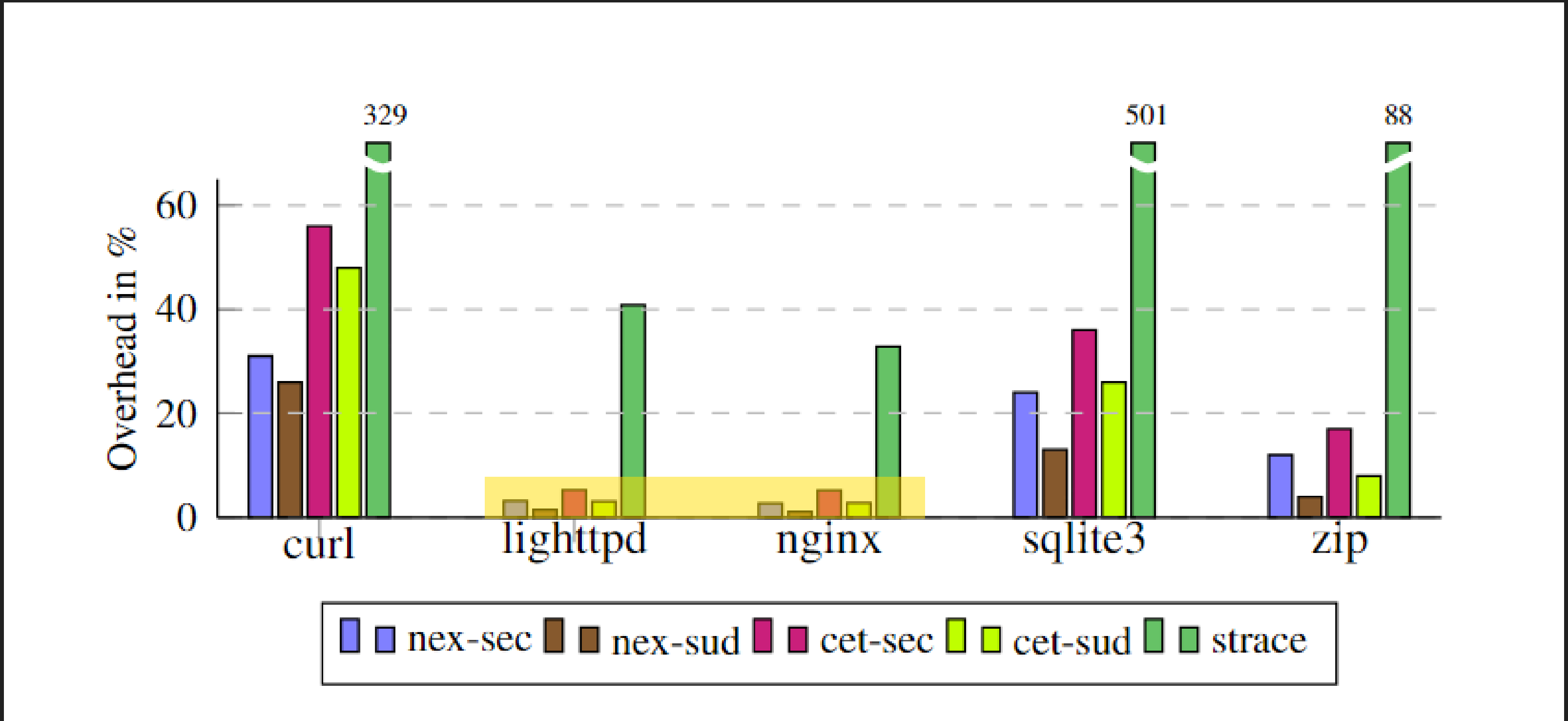
Virtualization Cost with Different Callgate Mechanism



Virtualization Cost with Different Callgate Mechanism



Virtualization Cost with Different Callgate Mechanism



Thread Scalability: Near Consistent Overhead with More Thread

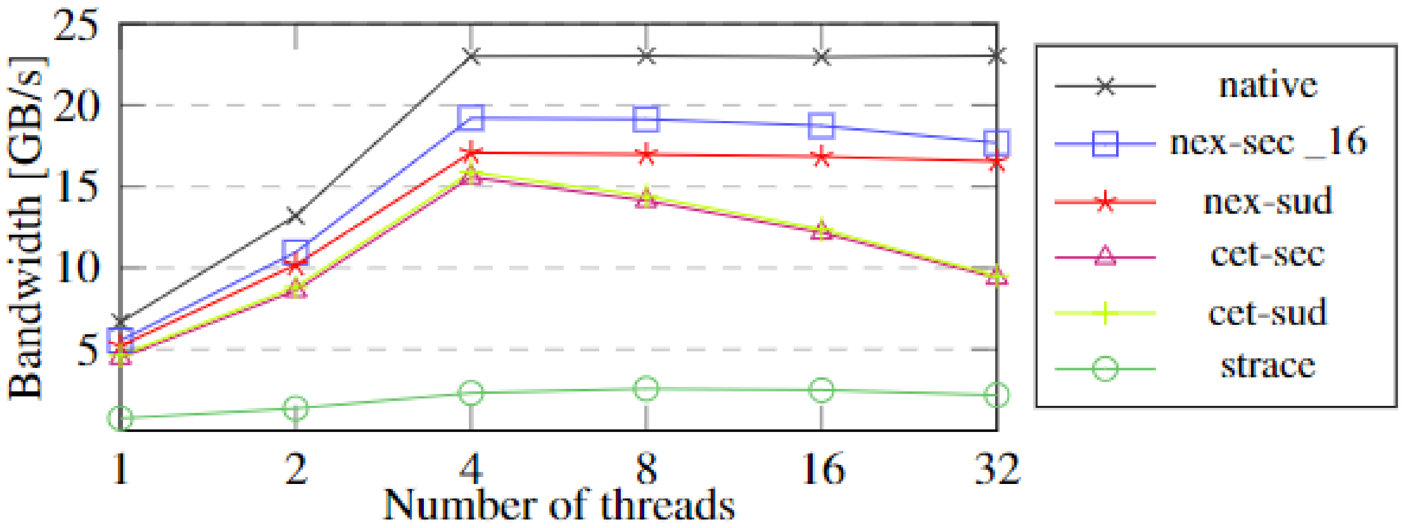


Figure 6: Random read bandwidth for diff. numbers of threads measured with `sysbench`. Std. dev. below 0.7%.

Thread Scalability: Near Consistent Overhead with More Thread

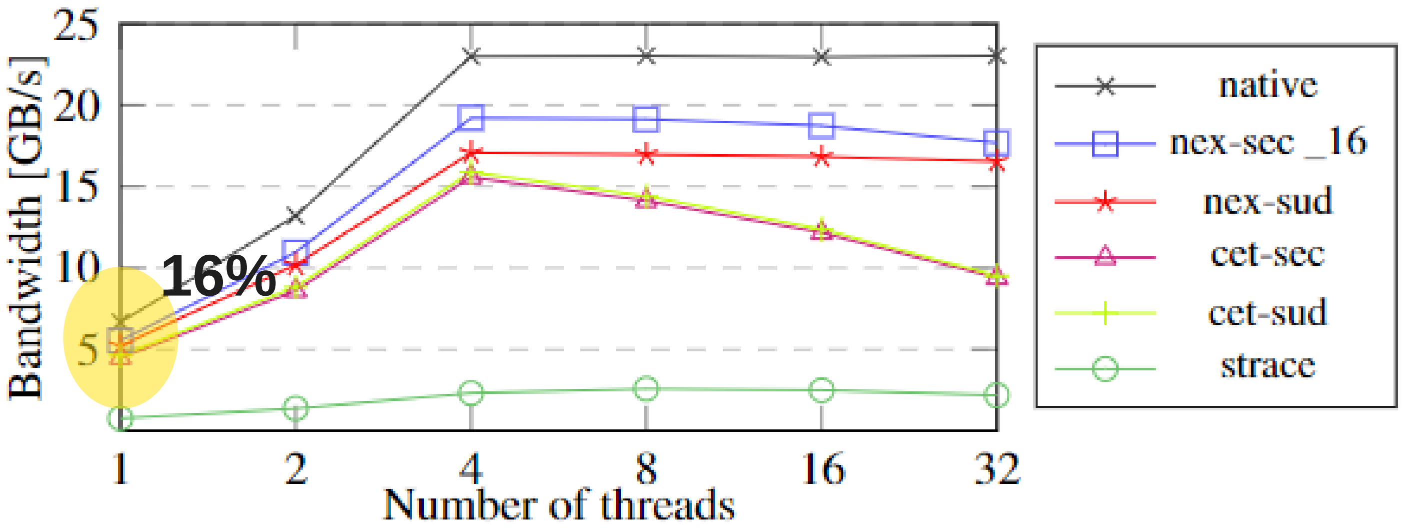


Figure 6: Random read bandwidth for diff. numbers of threads measured with `sysbench`. Std. dev. below 0.7%.

Thread Scalability: Near Consistent Overhead with More Thread

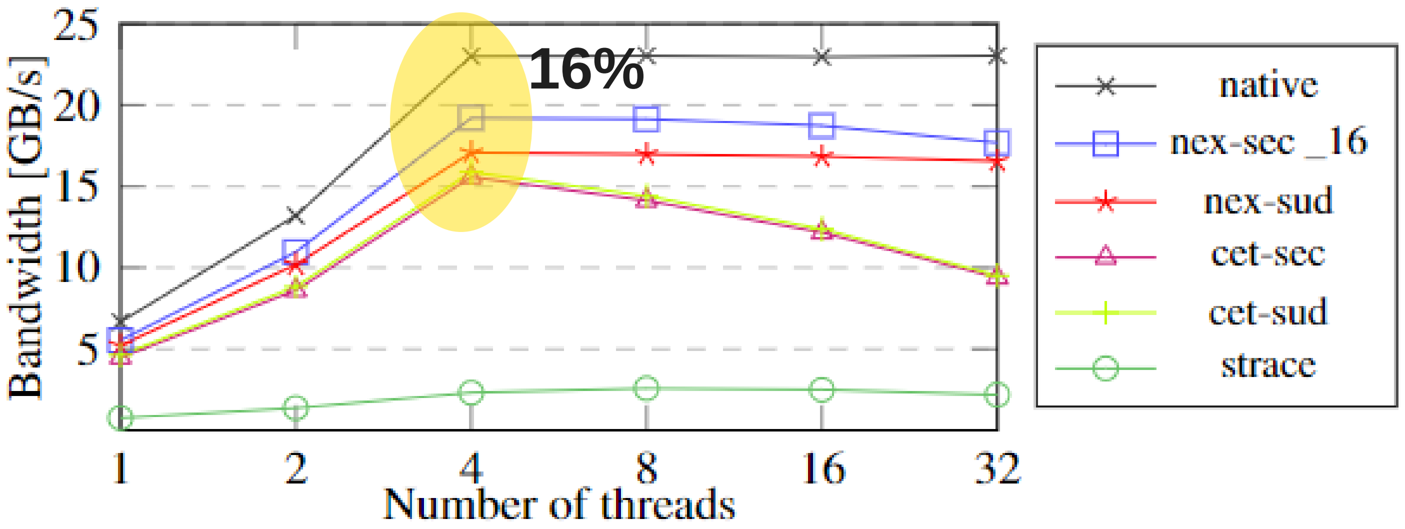


Figure 6: Random read bandwidth for diff. numbers of threads measured with `sysbench`. Std. dev. below 0.7%.

Thread Scalability: Near Consistent Overhead with More Thread

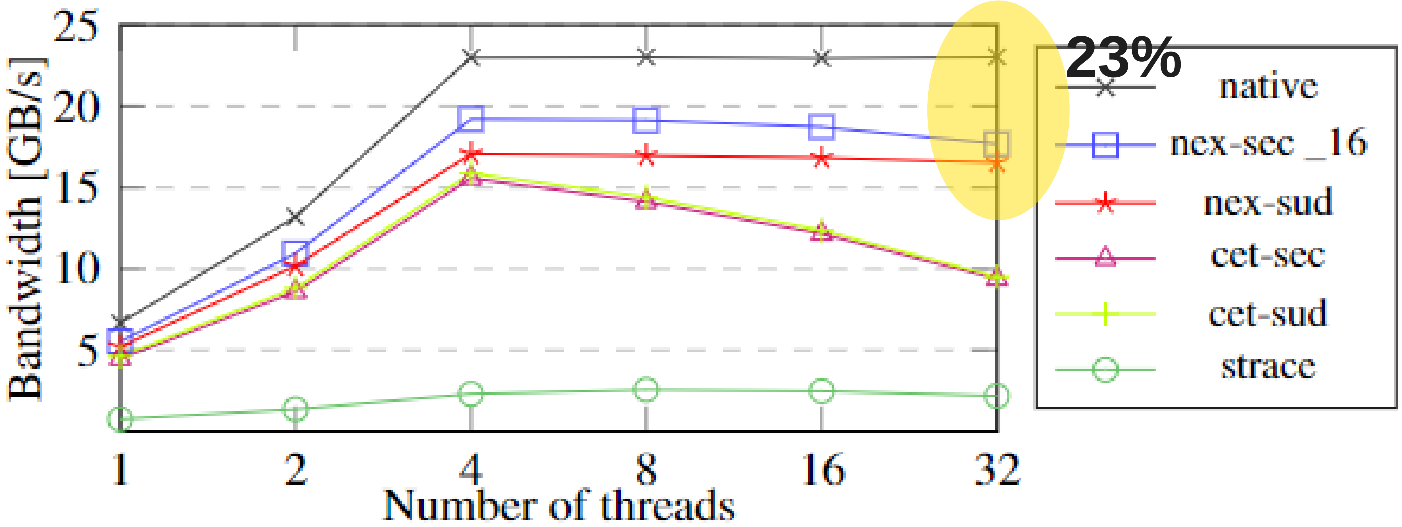


Figure 6: Random read bandwidth for diff. numbers of threads measured with `sysbench`. Std. dev. below 0.7%.

Compatibility Test for the Signal and Multithreading

- Linux Test Project (LTP) provides regression and conformance to the kernel
- The Endokernel passed **95.95%** of the LTP test cases
- The failed cases are not related to thread or signal compatibility
  - Security-related
  - Kernel Side-Effect
  - Endokernel as a secondary loading
  - Memory Layout

Takeaways

- For an in-process monitor, thread safety is **not** as simple as just adding locks
- Weak Metadata Synchronization
  - Conservative monitor state updates to achieve safe results even in cases of unsynchronized operations
- Signals Virtualization
  - Complete virtualization of signal behavior within the monitor to avoid synchronization with the kernel
- High Memory Access Bypass
  - Locating these patterns through source code analysis, enabling for case-by-case examination
- <5.5% overhead on nginx and lighttpd; ~30% overhead on curl with nex-sud
- ~23% overhead with increasing thread count
- Passes 95% of LTP tests with insignificant failed cases
- Source code: https://github.com/endokernel/test/
- Q&A