SEIMI: Efficient and Secure SMAP-Enabled Intra-process Memory Isolation

Zhe Wang[1], Chenggang Wu[1], Mengyao Xie[1], Yinqian Zhang[2], Kangjie Lu[3], Xiaofeng Zhang[1], Yuanming Lai[1], Yan Kang[1], and Min Yang[4]

[1]Institute of Computing Technology, Chinese Academy of Sciences,

[2]The Ohio State University, [3]University of Minnesota at Twin-Cities, [4]Fudan University

Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …

Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable. 

Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable. 
- **The strict memory isolations for the metadata in defenses are needed.**

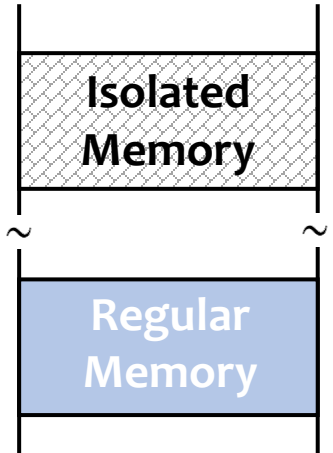
Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable. 
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.

Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

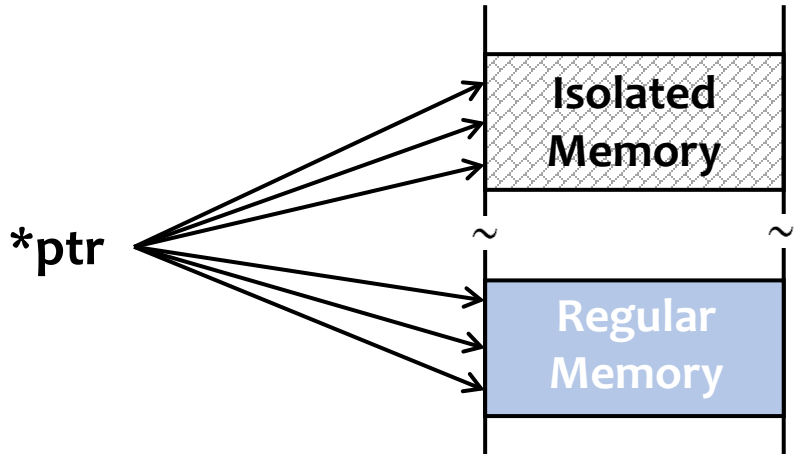
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

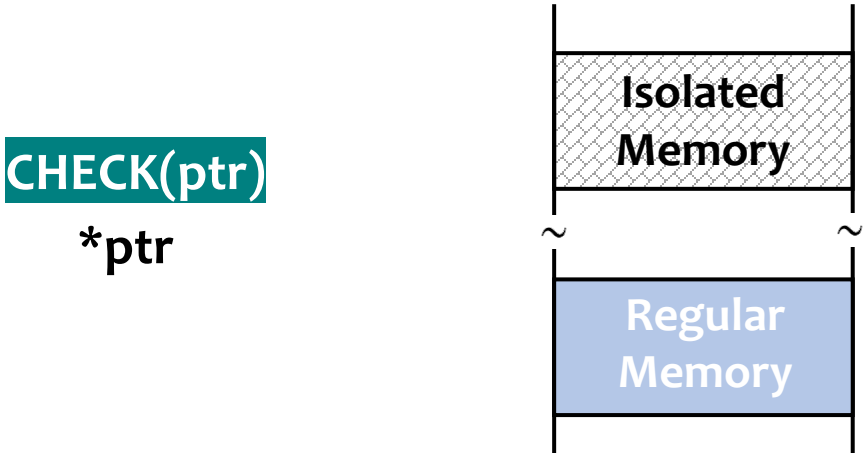
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

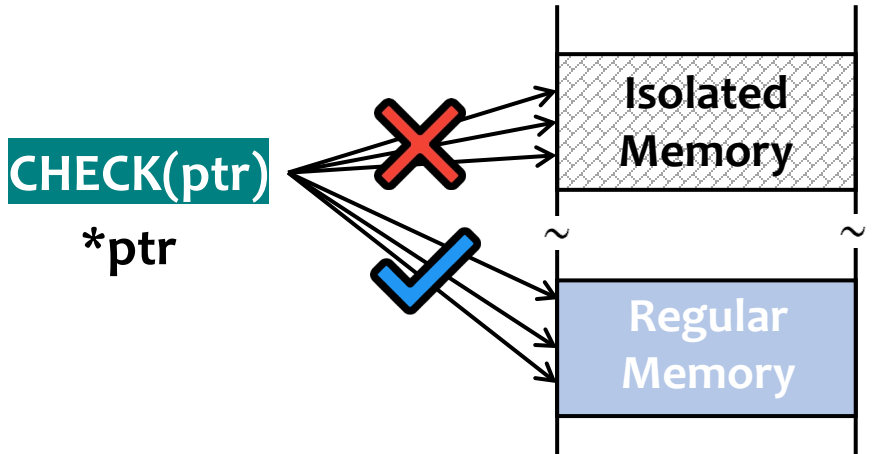
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

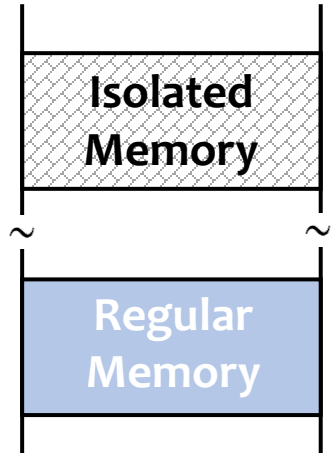
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

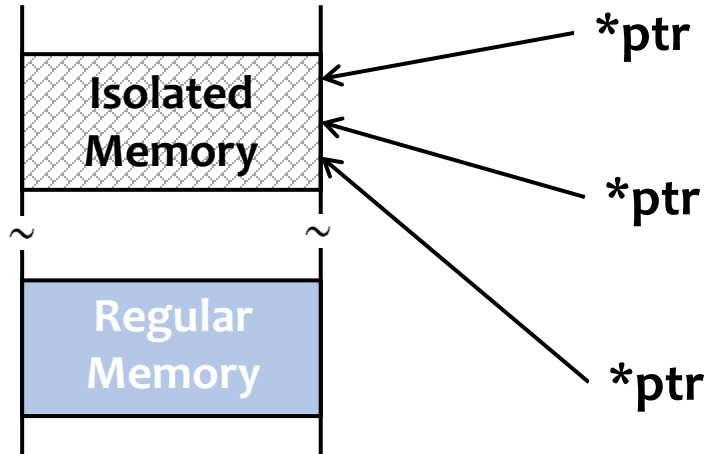
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

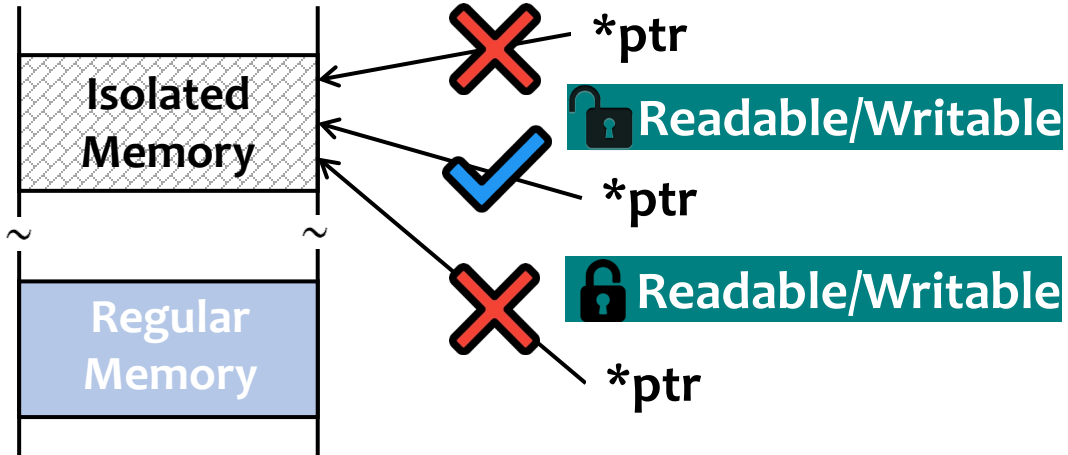
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable.

- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.



Intra-process Memory Isolation

- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable. 
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.

But they are not efficient enough as we expect.

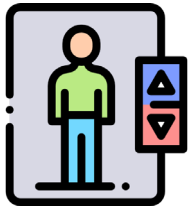
Threat Model

- **We consider a defense that protects a vulnerable application against memory corruption attacks.**
  - Web servers, databases or browsers.

- **The design of this defense is secure:**
  - Breaking memory isolation is a prerequisite for compromising the defense (e.g., attackers cannot hijack the control flow before it).

- **Attackers' capabilities:**
  - Arbitrary read and write by exploiting memory corruption vulnerabilities.

Outline



Motivation

High-level Design

Approach Overview

SEIMI System

Evaluation

Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are not fast enough for isolating in the user-mode process.

Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are not fast enough for isolating in the user-mode process.

Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are not fast enough for isolating in the user-mode process.

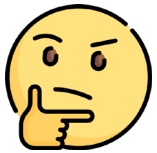
The user-mode hardware features are not fast.

Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are not fast enough for isolating in the user-mode process.

The user-mode hardware features are not fast.

How about the privileged hardware feature ?

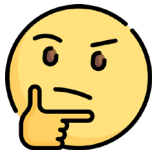


Motivation

- **Problem:**
  - Hardware-assisted memory isolations could achieve better performance.
  - But existing methods are not fast enough for isolating in the user-mode process.

The user-mode hardware features are not fast.

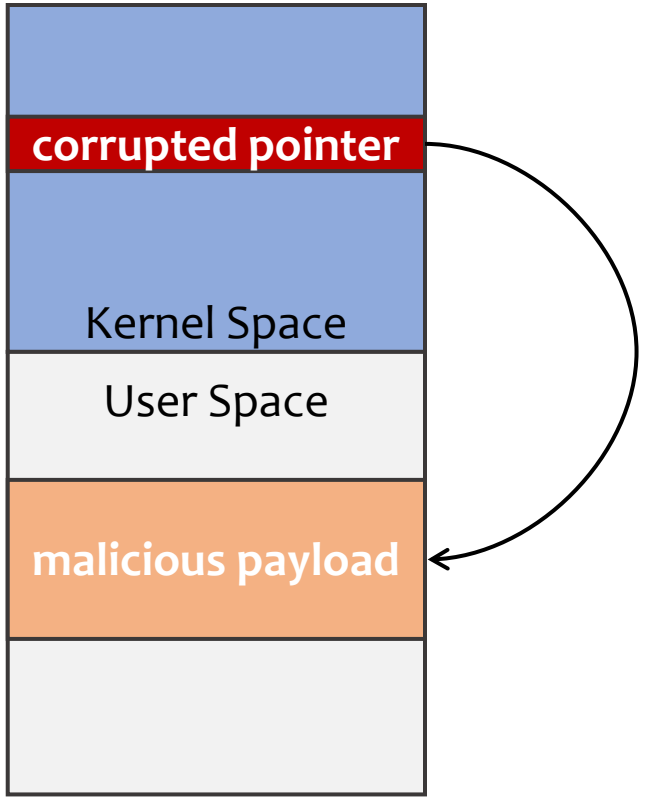
How about the privileged hardware feature ?



Is there a privileged hardware feature which is more efficient than Intel MPX/MPK for the memory isolation ???

Motivation —— SMAP in Processors 101

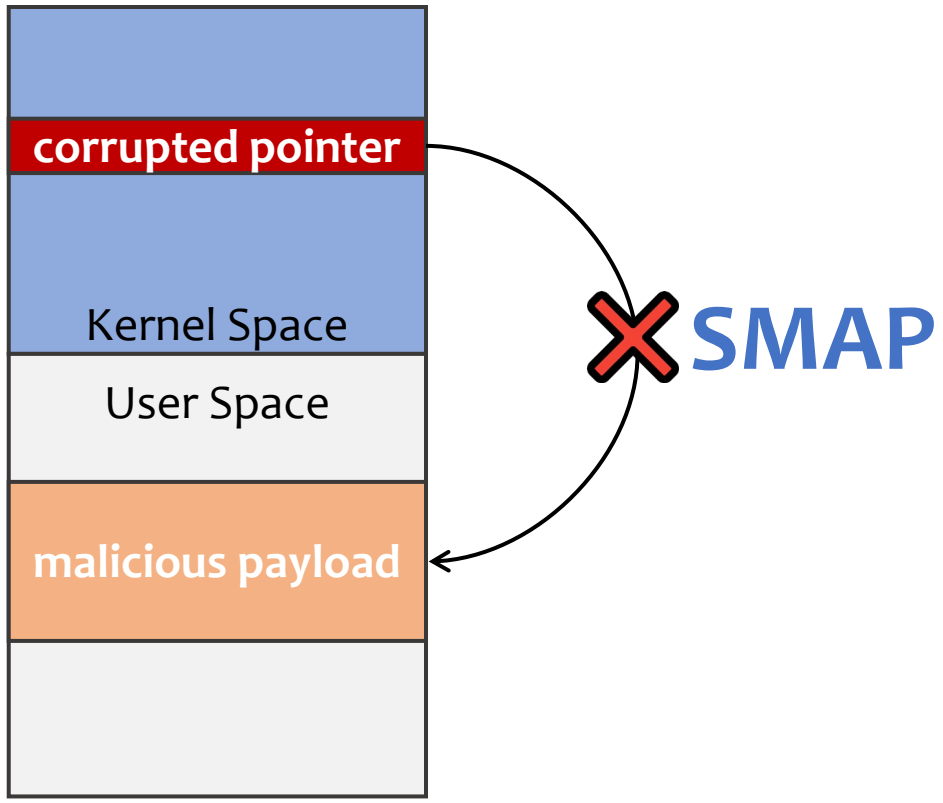
- To prevent the kernel from inadvertently accessing malicious data in user space,
  - dereferencing a corrupted data pointer



Motivation —— SMAP in Processors 101

- To prevent the kernel from inadvertently accessing malicious data in user space,
  - dereferencing a corrupted data pointer

- Intel and AMD provide the **Supervisor-mode Access Prevention (SMAP)** hardware feature to disable the kernel access to the user space memory.



Motivation —— SMAP in Processors 101

- **Supervisor-mode Page (S-page)** vs. **User-mode Page (U-page)**
  - Divided by the U/S bit in the page table entry.

Motivation —— SMAP in Processors 101

- **Supervisor-mode Page (S-page)** vs. **User-mode Page (U-page)**
  - Divided by the U/S bit in the page table entry.

- **SMAP disallows the code access to the U-page in the supervisor-mode.**
  - S-mode is short for supervisor-mode (ring 0-2).
  - U-mode is short for user mode (ring 3).

Motivation —— SMAP in Processors 101

- **Supervisor-mode Page (S-page)** vs. **User-mode Page (U-page)**
  - Divided by the U/S bit in the page table entry.

- **SMAP disallows the code access to the U-page in the supervisor-mode.**
  - S-mode is short for supervisor-mode (ring 0-2).
  - U-mode is short for user mode (ring 3).

	Ring 0	Ring 1	Ring 2	Ring 3
Privileged Instruction Fetch	✓	✗	✗	✗
S-page Access Permission	✓	✓	✓	✗
U-page Access Permission	✓	✓	✓	✓

SMAP is disabled

Motivation —— SMAP in Processors 101

- **Supervisor-mode Page (S-page) vs. User-mode Page (U-page)**
  - Divided by the U/S bit in the page table entry.

- **SMAP disallows the code access to the U-page in the supervisor-mode.**
  - S-mode is short for supervisor-mode (ring 0-2).
  - U-mode is short for user mode (ring 3).

	Ring 0	Ring 1	Ring 2	Ring 3
Privileged Instruction Fetch	✓	✗	✗	✗
S-page Access Permission	✓	✓	✓	✗
U-page Access Permission	✗	✗	✗	✓

SMAP is enabled

Motivation —— SMAP in Processors 101

- **X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.**
  - When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

Motivation —— SMAP in Processors 101

- **X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.**
  - When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

- **POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.**
  - **popfq** could be execute in S-mode (ring 0-2).
  - **stac/clac** are privileged instructions that can only be execute in ring 0.

Motivation —— SMAP in Processors 101

- **X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.**
  - When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

- **POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.**
  - popfq could be execute in S-mode (ring 0-2).
  - stac/clac are privileged instructions that can only be execute in ring 0.

Instructions	Cycles	Description
wrpkru	**18.9**	Update the access right of a pkey in **Intel MPK**
popfq	**22.4**	Pop stack into the RFLAGS register.
stac/clac 	**8.6**	Set/Clear the AC flag in the RFLAGS register.

Motivation —— SMAP in Processors 101

- **X86 processors provide a RFLAGS.AC flag to disable/enable SMAP.**
  - When the RFLAGS.AC flag is set in S-mode, SMAP is disabled.

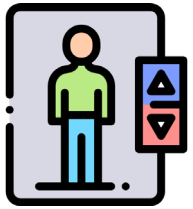
- **POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.**
  - popfq could be execute in S-mode (ring 0-2).
  - stac/clac are privileged instructions that can only be execute in ring 0.

Instructions	Cycles	Description
wrpkru	**18.9**	Update the access right of a pkey in **Intel MPK**
popfq	**22.4**	Pop stack into the RFLAGS register.
stac/clac 	**8.6**	Set/Clear the AC flag in the RFLAGS register.

Intel SMAP is more efficient than Intel MPK for controlling memory access permission.

Outline

Motivation



High-level Design

Approach Overview

SEIMI System

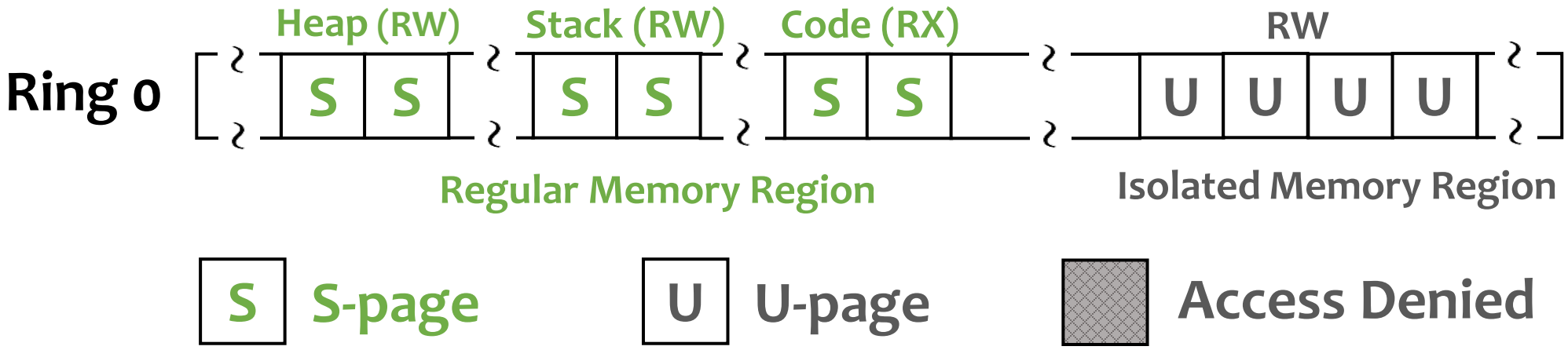
Evaluation

High-level Design —— SEIMI

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-page**s.
  - Other memory regions are set to be **S-page**s.
- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.

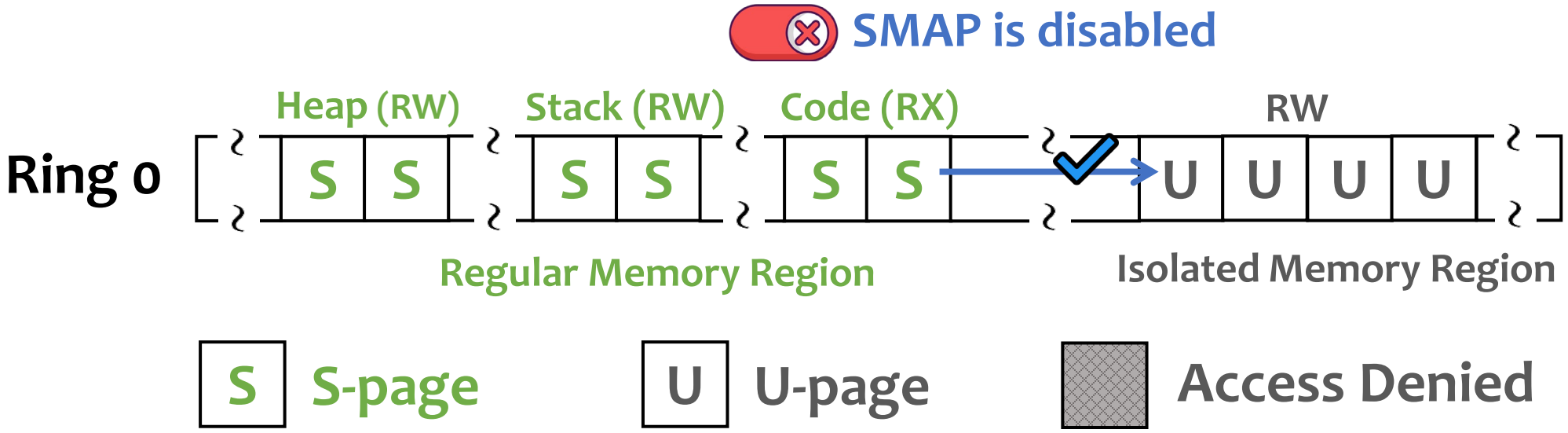
High-level Design —— SEIMI

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-page**s.
  - Other memory regions are set to be **S-page**s.
- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.



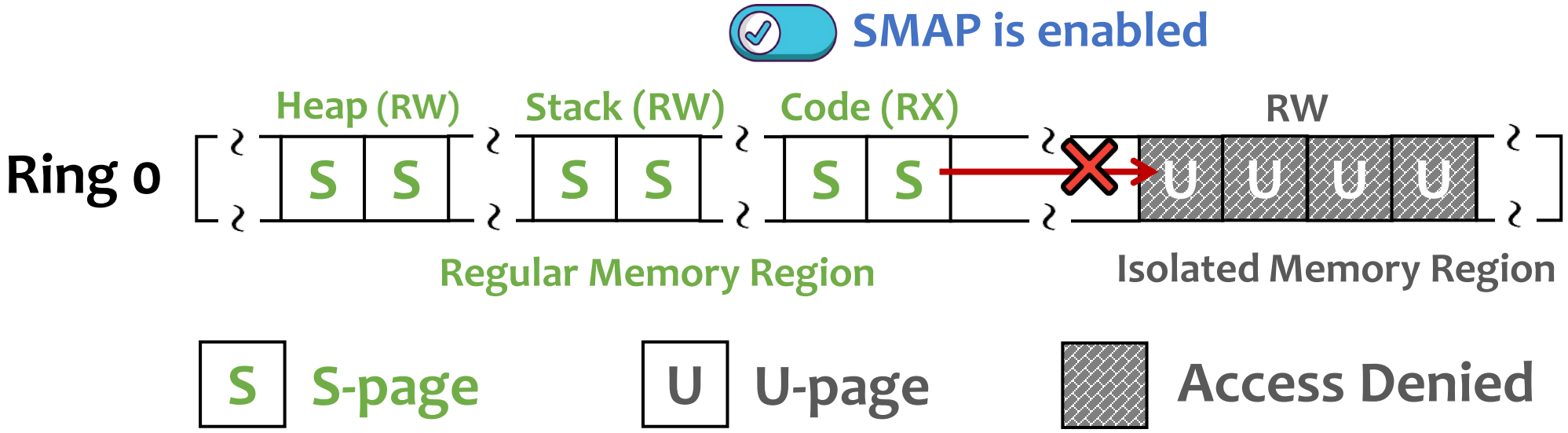
High-level Design —— SEIMI

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-page**s.
  - Other memory regions are set to be **S-page**s.
- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.



High-level Design —— SEIMI

- **The Memory Layout Setting**
  - The isolated memory region are set to be **U-page**s.
  - Other memory regions are set to be **S-page**s.
- **The Running State Setting**
  - The process runs in **ring 0**, due to the stac/clac are privileged instructions.



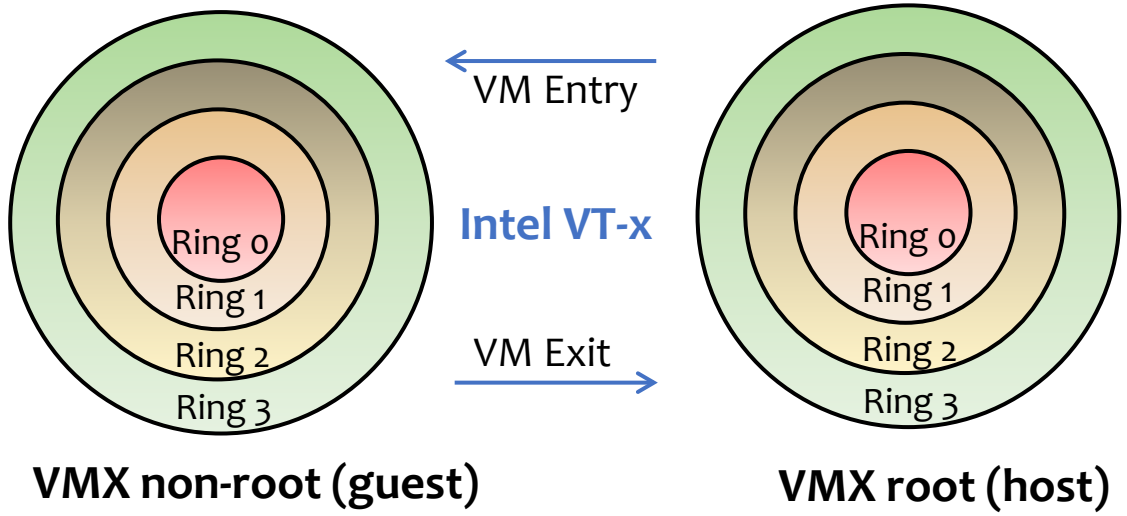
High-level Design —— SEIMI

- **Problem:**
  - Running untrusted code in ring 0 may corrupt the OS kernel.

High-level Design —— SEIMI

- **Problem:**
  - Running untrusted code in ring 0 may corrupt the OS kernel.

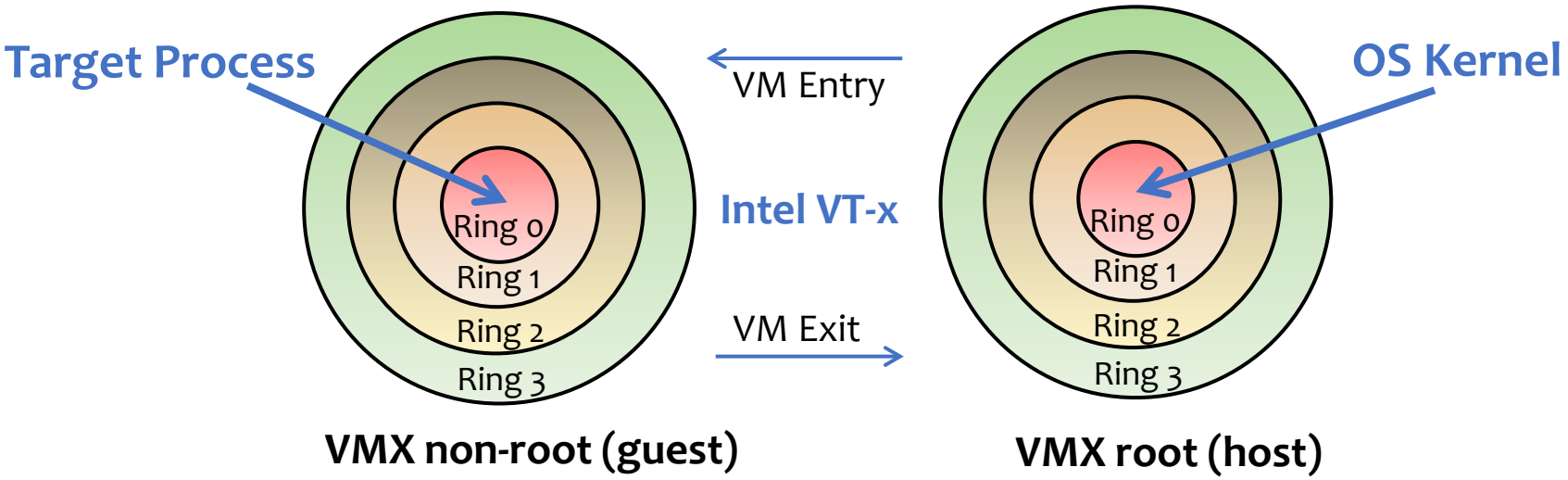
- **Our Solution** —— Placing the OS kernel in "ring -1"
  - Using the Intel VT-x technique to separate the target application and the OS kernel



High-level Design —— SEIMI

- **Problem:**
  - Running untrusted code in ring 0 may corrupt the OS kernel.

- **Our Solution** —— Placing the OS kernel in "ring -1".
  - Using the Intel VT-x technique to separate the target application and the OS kernel



High-level Design —— Challenges in SEIMI

- **C-1: Distinguishing SMAP reads and writes.**
  - Sensitive data may require only **integrity** protection.
  - Preventing reads from untrusted code can lead to **unnecessary** overhead.

High-level Design —— Challenges in SEIMI

- **C-1: Distinguishing SMAP reads and writes.**
  - Sensitive data may require only **integrity** protection.
  - Preventing reads from untrusted code can lead to **unnecessary** overhead.

- **C-2: Preventing the leaking/manipulating of the privileged data structures.**
  - In general, a guest VM needs to manage the memory, interrupts, exceptions, etc.
  - Some data structures are **privileged**, e.g., the page tables.

High-level Design —— Challenges in SEIMI

- **C-1: Distinguishing SMAP reads and writes.**
  - Sensitive data may require only **integrity** protection.
  - Preventing reads from untrusted code can lead to **unnecessary** overhead.

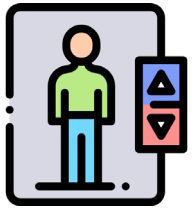
- **C-2: Preventing the leaking/manipulating of the privileged data structures.**
  - In general, a guest VM needs to manage the memory, interrupts, exceptions, etc.
  - Some data structures are **privileged**, e.g., the page tables.

- **C-3: Preventing the abusing of the privileged hardware features.**
  - Besides the stac/clac, **other** privileged instructions can also run in ring 0.

Outline

Motivation

High-level Design



Approach Overview

SEIMI System

Evaluation

Approaches Overview —— Challenge-1

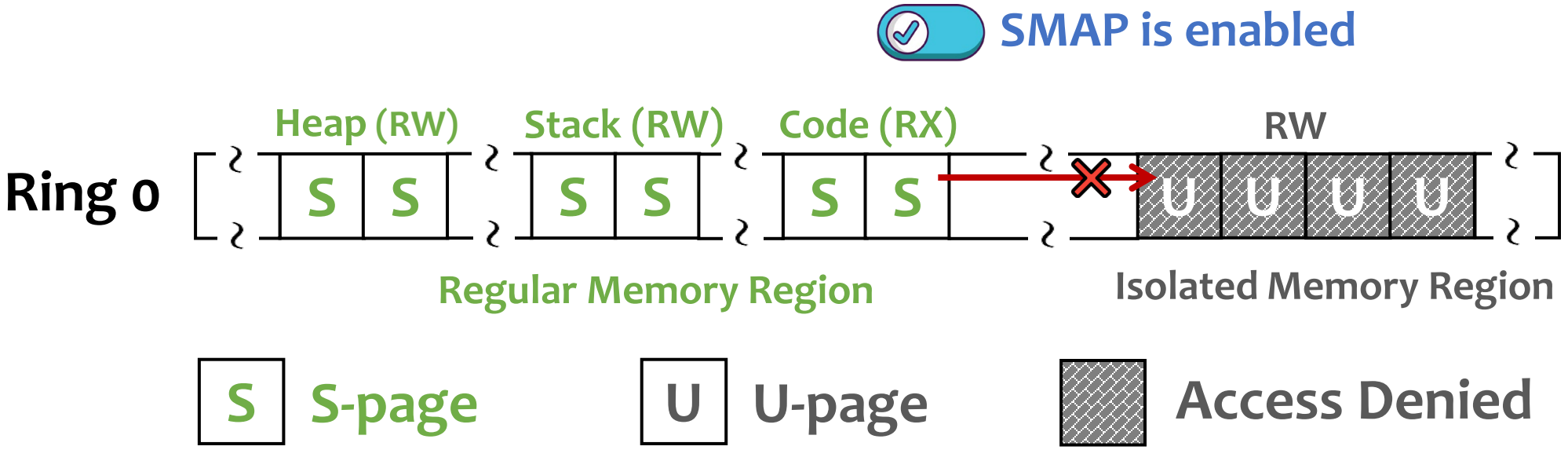
- **C-1: Distinguishing SMAP reads and writes.**

- **Solution** —— The shared-memory based read/write separation method.

Approaches Overview —— Challenge-1

- **C-1: Distinguishing SMAP reads and writes.**

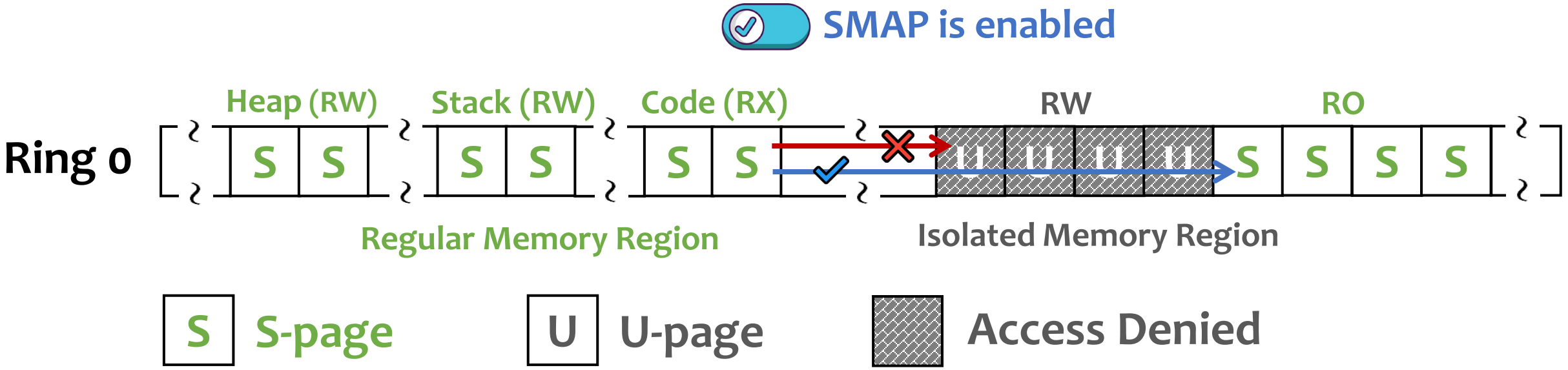
- **Solution ——** The shared-memory based read/write separation method.



Approaches Overview —— Challenge-1

- **C-1: Distinguishing SMAP reads and writes.**

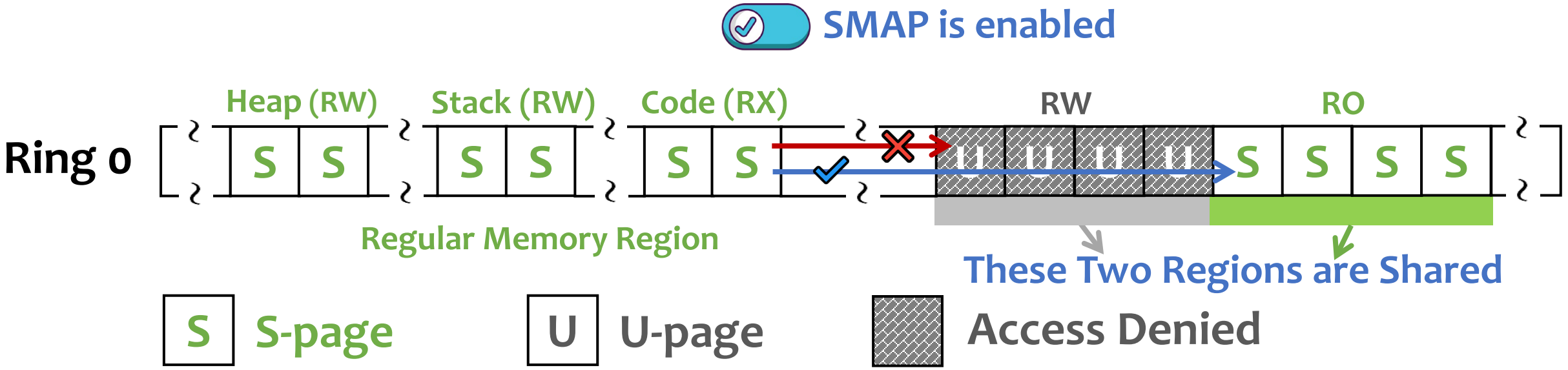
- **Solution ——** The shared-memory based read/write separation method.



Approaches Overview —— Challenge-1

- **C-1: Distinguishing SMAP reads and writes.**

- **Solution ——** The shared-memory based read/write separation method.



Approaches Overview —— Challenge-2

- **C-2: Preventing the leaking/manipulating of the privileged data structures.**

Approaches Overview —— Challenge-2

- **C-2: Preventing the leaking/manipulating of the privileged data structures.**

- **Observation:**
  - The operations to these structures are **only** performed when the process **accesses** the OS kernel through specific **events**, e.g., interrupts, exceptions, and system calls.

Approaches Overview —— Challenge-2

- **C-2: Preventing the leaking/manipulating of the privileged data structures.**

- **Observation:**

  - The operations to these structures are **only** performed when the process **accesses** the OS kernel through specific **events**, e.g., interrupts, exceptions, and system calls.

- **Solution:**

  - Placing the privileged data structures and their operations **into the VMX root mode**.
  - We leverage the Intel VT-x technique to force all these events to **trigger VM exits** and enter into the VMX root mode.

Approaches Overview —— Challenge-3

- **C-3: Preventing the abusing of the privileged hardware features.**

Approaches Overview —— Challenge-3

- **C-3: Preventing the abusing of the privileged hardware features.**

- **Solution:**
  - SEIMI **sanitizes** the execution of all privileged instructions in the VMX non-root mode.

Approaches Overview —— Challenge-3

- **C-3: Preventing the abusing of the privileged hardware features.**

- **Solution:**
  - SEIMI **sanitizes** the execution of all privileged instructions in the VMX non-root mode.

1 Triggering the VM exits and stopping the execution;

Approaches Overview —— Challenge-3

- **C-3: Preventing the abusing of the privileged hardware features.**

- **Solution:**
  - SEIMI **sanitizes** the execution of all privileged instructions in the VMX non-root mode.

1. Triggering the VM exits and stopping the execution;
2. Invalidating the execution effects;

Approaches Overview —— Challenge-3

- **C-3: Preventing the abusing of the privileged hardware features.**

- **Solution:**
  - SEIMI **sanitizes** the execution of all privileged instructions in the VMX non-root mode.

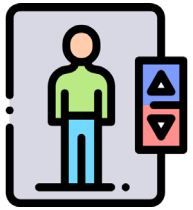
1. Triggering the VM exits and stopping the execution;
2. Invalidating the execution effects;
3. Raising processor exceptions and disabling the execution.

Outline

Motivation

High-level Design

Approach Overview



SEIMI System

Evaluation

System Overview

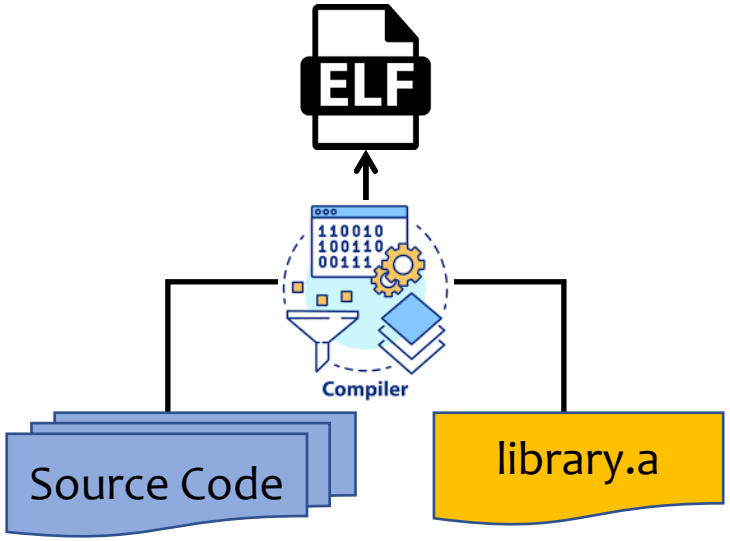
- **SEIMI is implemented on Linux/X86_64 platform.**

System Overview

- SEIMI is implemented on Linux/X86_64 platform.
- Two Phases in SEIMI —— Compilation Phase and Runtime Phase

System Overview

- **SEIMI is implemented on Linux/X86_64 platform.**
- **Two Phases in SEIMI —— Compilation Phase and Runtime Phase**

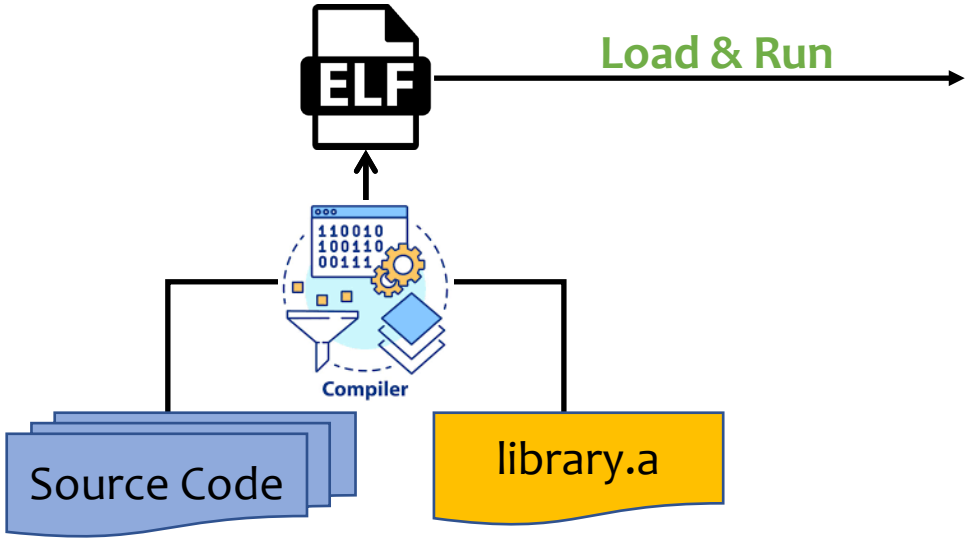


Compilation phase

Users could use the SEIMI's APIs to management the isolated memory region.

System Overview

- **SEIMI is implemented on Linux/X86_64 platform.**
- **Two Phases in SEIMI —— Compilation Phase and Runtime Phase**

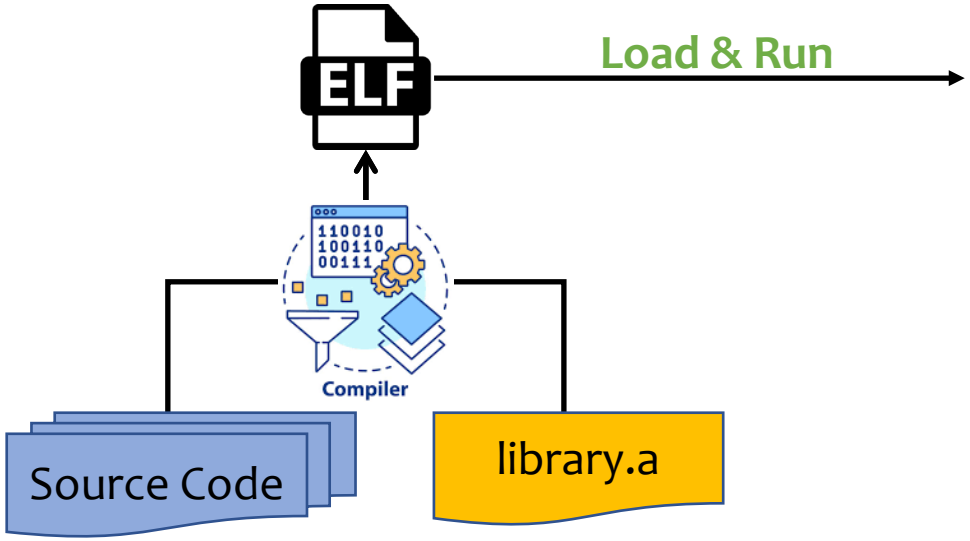


Compilation phase

Users could use the SEIMI's APIs to management the isolated memory region.

System Overview

- **SEIMI is implemented on Linux/X86_64 platform.**
- **Two Phases in SEIMI —— Compilation Phase and Runtime Phase**



Compilation phase

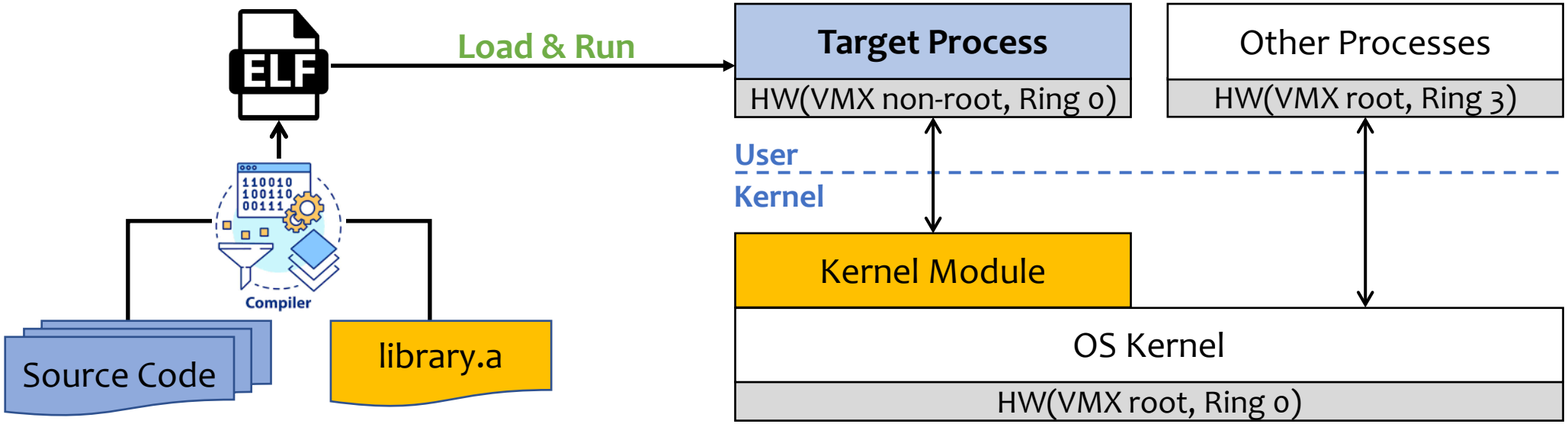
Users could use the SEIMI's APIs to management the isolated memory region.

Runtime Phase

The core of SEIMI is a kernel module which monitors the startup of the target application and places it into ring 0 of the VMX non-root mode.

System Overview

- **SEIMI is implemented on Linux/X86_64 platform.**
- **Two Phases in SEIMI —— Compilation Phase and Runtime Phase**



Compilation phase

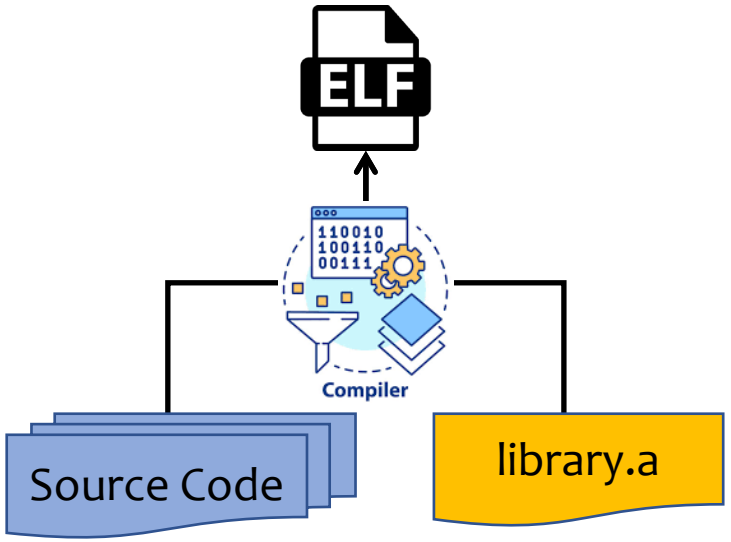
Users could use the SEIMI's APIs to management the isolated memory region.

Runtime Phase

The core of SEIMI is a kernel module which monitors the startup of the target application and places it into ring 0 of the VMX non-root mode.

SEIMI —— Compilation Phase

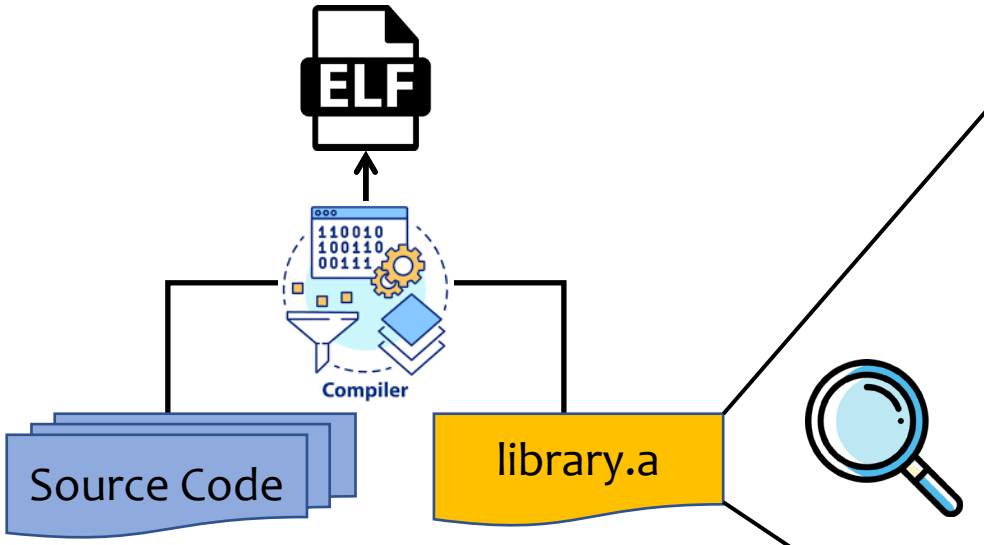
- **SEIMI provides APIs to allocate/free the isolated region, and enable/disable the SMAP.**



Compilation phase

SEIMI —— Compilation Phase

- **SEIMI provides APIs to allocate/free the isolated region, and enable/disable the SMAP.**

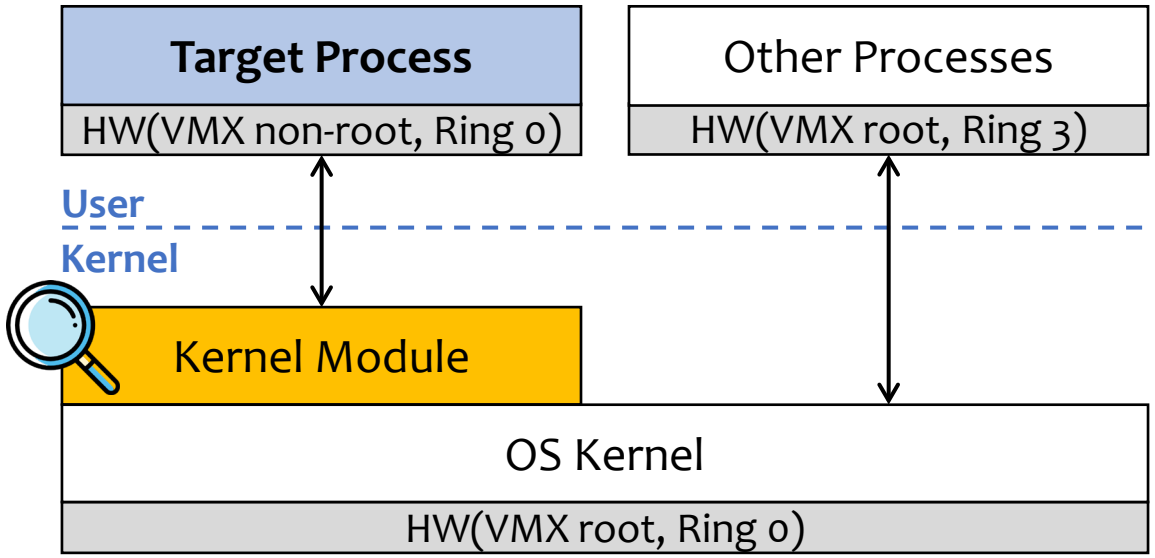


Compilation phase

API	Description
void *sa_alloc(size_t *length*, bool *need_ro*, long **offset*);	Allocate an isolated U-page region at the page granularity. If specified, it also allocates a shared isolated S-page region.
bool sa_free(void **addr*, size_t *length*);	Free an isolated U-page region with the specified length.
#define SWITCH_IN \ asm("stac\n");	Disable SMAP—access the isolated U-page region is allowed.
#define SWITCH_OUT \ asm("clac\n");	Enable SMAP—access to the isolated U-page region is denied.

SEIMI —— Runtime Phase

- **The core of SEIMI is a kernel module, includes three key components.**



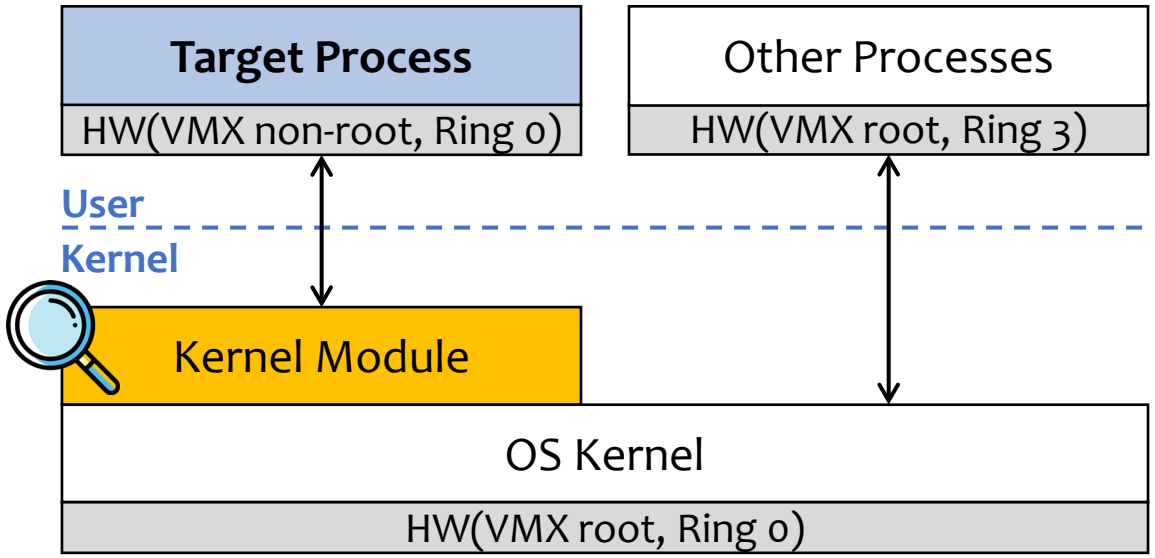
Runtime Phase

SEIMI —— Runtime Phase

- **The core of SEIMI is a kernel module, includes three key components.**

1 Memory Management Component

– Configures the regular/isolated memory region.



Runtime Phase

SEIMI —— Runtime Phase

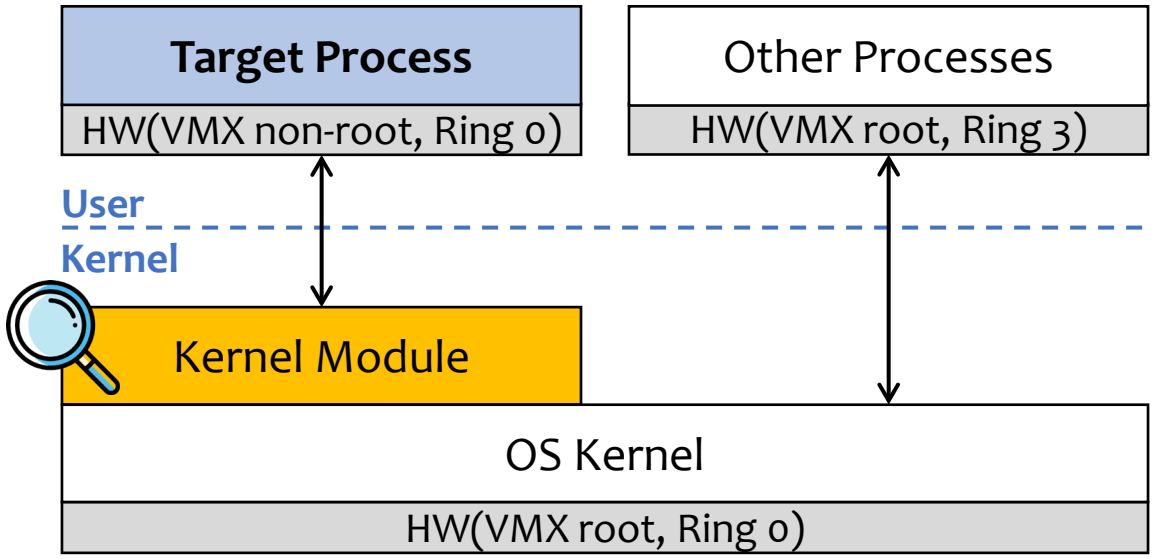
- **The core of SEIMI is a kernel module, includes three key components.**

1 Memory Management Component

– Configures the regular/isolated memory region.

2 Privileged Instructions Prevention Component

– Prevents these instructions from being abused.



Runtime Phase

SEIMI —— Runtime Phase

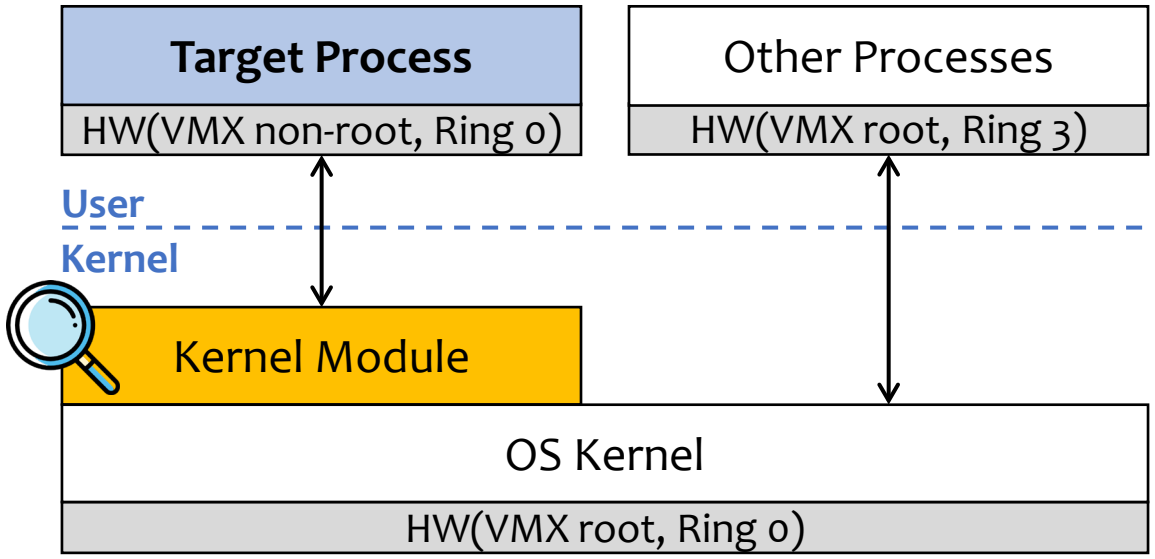
- **The core of SEIMI is a kernel module, includes three key components.**

1 Memory Management Component

– Configures the regular/isolated memory region.

2 Privileged Instructions Prevention Component

– Prevents these instructions from being abused.



Runtime Phase

3 Events Redirection Component

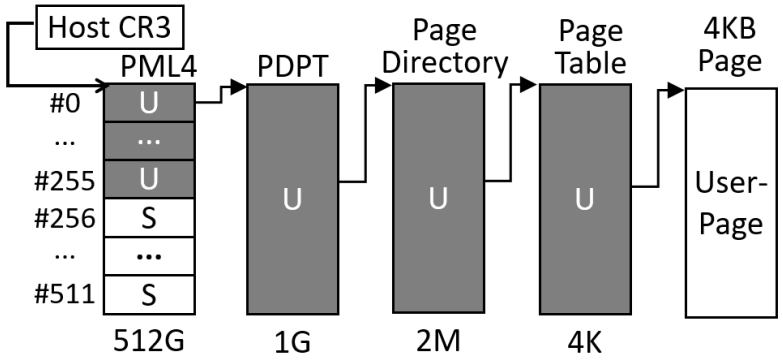
– Handles system calls, interrupts, exceptions, and Linux signals.

SEIMI —— Memory Management Component

- **A shadow mechanism for (only) page-table root.**
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.

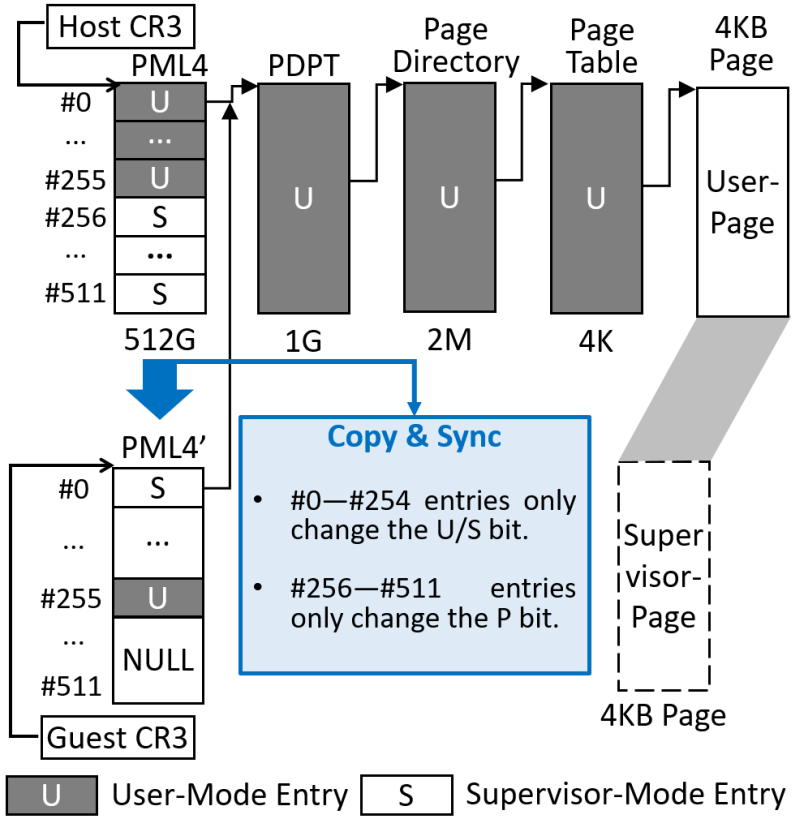
SEIMI —— Memory Management Component

- **A shadow mechanism for (only) page-table root.**
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.



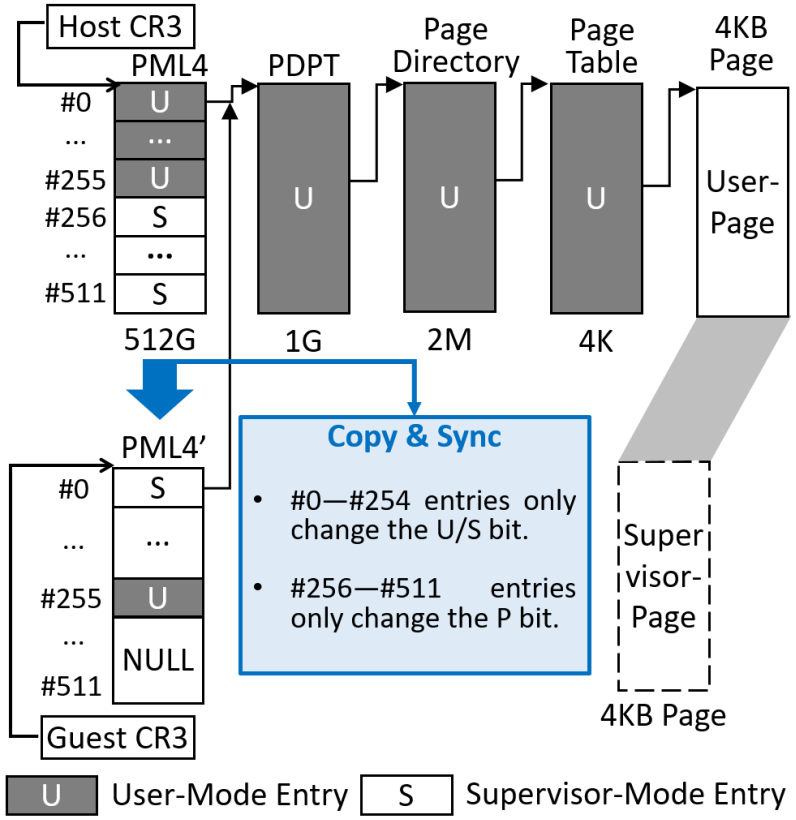
SEIMI —— Memory Management Component

- **A shadow mechanism for (only) page-table root.**
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.



SEIMI —— Memory Management Component

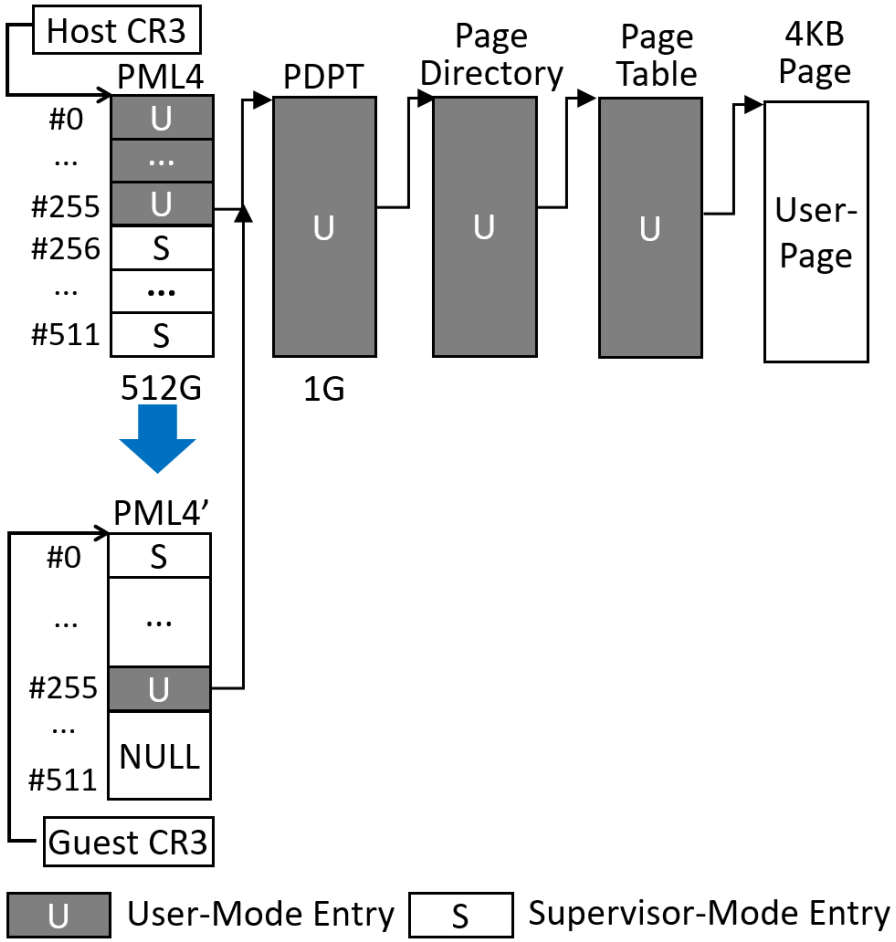
- **A shadow mechanism for (only) page-table root.**
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.



Entries in PML4	Size(TB)	Description	Type
#0 ~ #254	127.5	Regular Memory	S-page
#255	0.5	Isolated Memory	U-page
#255 ~ #511	128.0	Kernel Space	NULL

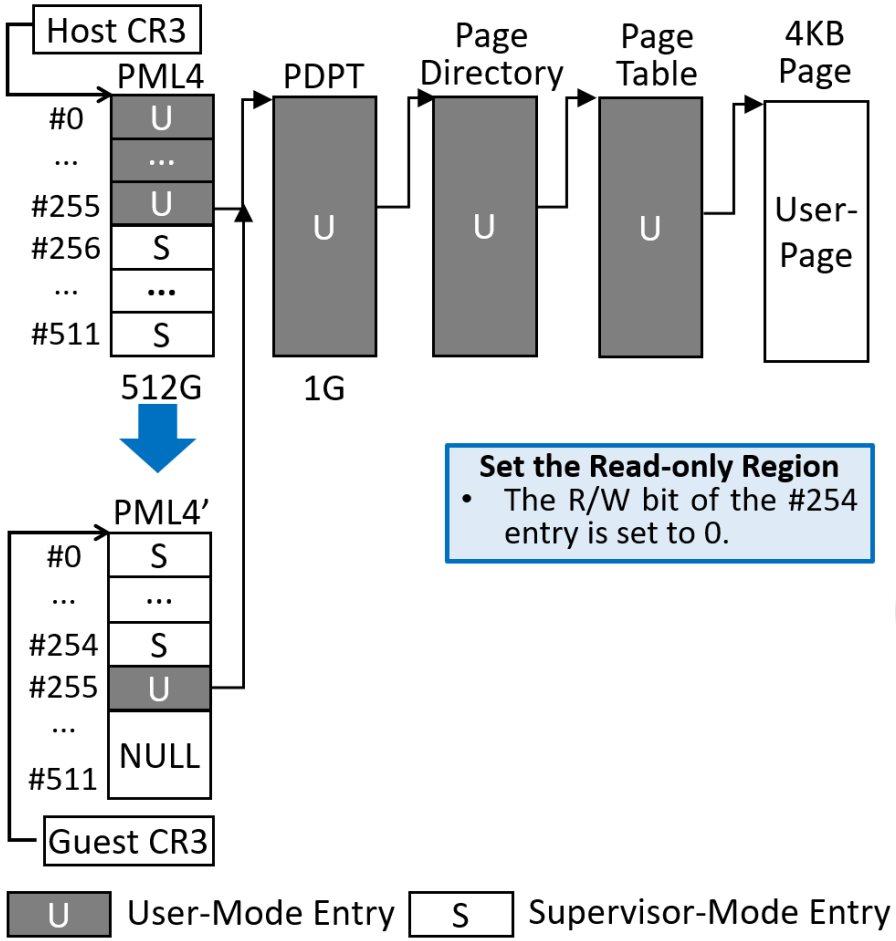
SEIMI —— Memory Management Component

- **Support the read-only isolated S-page memory region.**
  - Flipping the R/W bit to set the read-only permission neatly.



SEIMI —— Memory Management Component

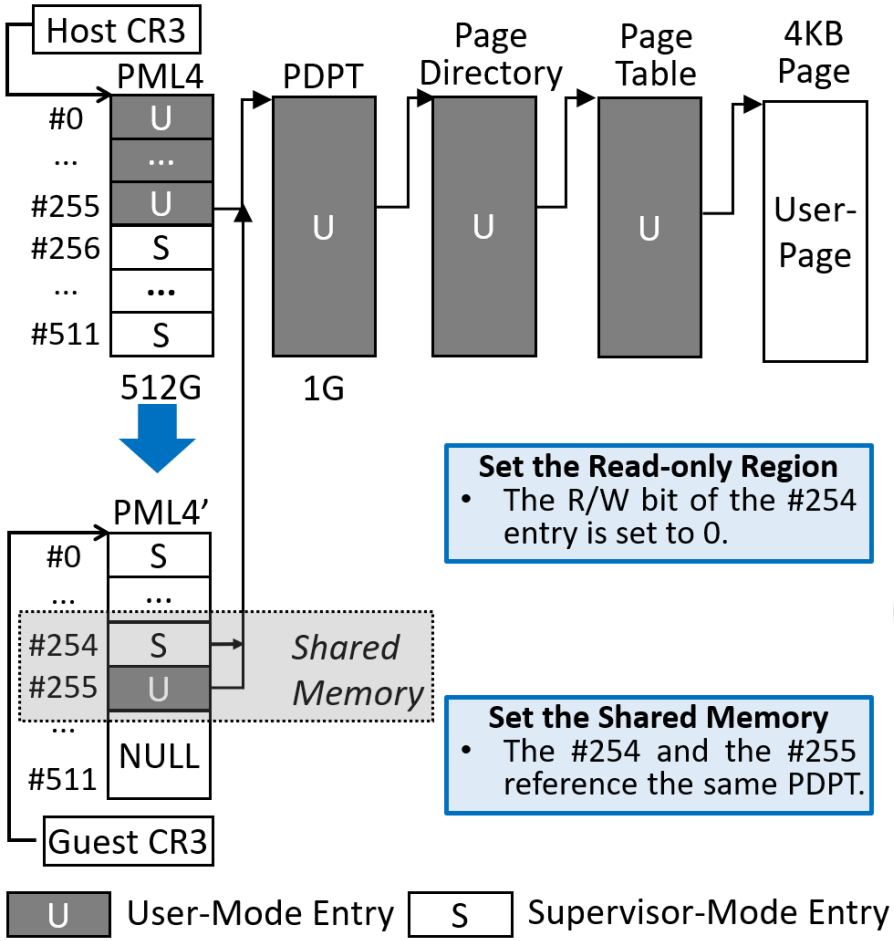
- **Support the read-only isolated S-page memory region.**
  - Flipping the R/W bit to set the read-only permission neatly.



Entries in PML4	Size(TB)	Description	Type
#0 ~ #253	127	Regular Memory	S-page
#254	0.5	Isolated Memory	S-page
#255	0.5	Isolated Memory	U-page
#255 ~ #511	128	Kernel Space	NULL

SEIMI —— Memory Management Component

- **Support the read-only isolated S-page memory region.**
  - Flipping the R/W bit to set the read-only permission neatly.



Entries in PML4	Size(TB)	Description	Type
#0 ~ #253	127	Regular Memory	S-page
#254	0.5	Isolated Memory	S-page
#255	0.5	Isolated Memory	U-page
#255 ~ #511	128	Kernel Space	NULL

SEIMI —— Privileged Instruction Prevention Component

- We identify **all privileged instructions** and the instructions that will **change the behaviors** in different rings in the 64-Bit mode of X86_64.

SEIMI —— Privileged Instruction Prevention Component

- We identify **all privileged instructions** and the instructions that will **change the behaviors** in different rings in the 64-Bit mode of X86_64.

- Our identification method:

1 Automated filtering

- We embed each instruction with **random operands** into a test program and run it in **ring 3**.
- By capturing the **#GP** and the **#UD**, we automatically and completely filter all privileged instructions.

SEIMI —— Privileged Instruction Prevention Component

- We identify **all privileged instructions** and the instructions that will **change the behaviors** in different rings in the 64-Bit mode of X86_64.

- Our identification method:

1 Automated filtering

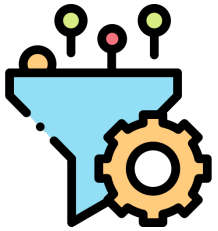
- We embed each instruction with **random operands** into a test program and run it in **ring 3**.
- By capturing the **#GP** and the **#UD**, we automatically and completely filter all privileged instructions.

2 Manual Verification

- We **manually** review the description of all X86 instructions by reading the Intel Software Developers' Manual.
- Confirm the first step is **complete**, and also find the instructions that **behave differently in ring 0 and ring 3.**

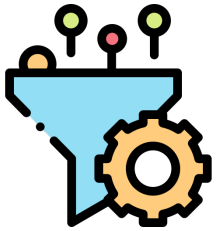
SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.



SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.

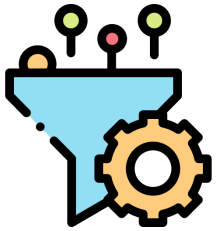


Line	Detailed Instructions	Is Privileged Instruction?
1	VM[RESUME|READ|WRITE|…], INVEPT, INVVPID	Y
2	INVD. XSETBV	Y
3	ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD…)	Y
4	RDMSR, WRMSR	Y
5	IN, OUT, IN[S|SB|SW|SD], OUT[S|SB|SW|SD]	Y
6	HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD	Y
7	LGDT, LLDT, LTR, LIDT	Y
8	MOV to/from DR0-DR7	Y
9	MOV to/from CR3, MOV to/from CR8	Y
10	MOV to/from CR0/CR4, CLTS, LMSW, SMSW	Y
11	MOV to/from CR2	Y
12	SWAPGS	Y
13	CLI, STI	Y
14	LAR, LSL. VERR, VERW	N
15	POPF, POPFQ	N
16	L[FS|DS|SS], MOV to [DS|ES|FS|GS|SS], POP [FS|GS]	N
17	Far CALL, Far RET, Far JMP	N
18	IRET, IRETD, IRETQ	Y
19	SYSEXIT, SYSRET	Y
20	XSAVES, XRSTORS, INVPCID	Y

SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.

Line	Detailed Instructions	Is Privileged Instruction?
1	VM[RESUME|READ|WRITE|…], INVEPT, INVVPID	Y
2	INVD. XSETBV	Y
3	ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD…)	Y
4	RDMSR, WRMSR	Y
5	IN, OUT, IN[S|SB|SW|SD], OUT[S|SB|SW|SD]	Y
6	HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD	Y
7	LGDT, LLDT, LTR, LIDT	Y
8	MOV to/from DR0-DR7	Y
9	MOV to/from CR3, MOV to/from CR8	Y
10	MOV to/from CR0/CR4, CLTS, LMSW, SMSW	Y
11	MOV to/from CR2	Y
12	SWAPGS	Y
13	CLI, STI	Y
14	LAR, LSL. VERR, VERW	N
15	POPF, POPFQ	N
16	L[FS|DS|SS], MOV to [DS|ES|FS|GS|SS], POP [FS|GS]	N
17	Far CALL, Far RET, Far JMP	N
18	IRET, IRETD, IRETQ	Y
19	SYSEXIT, SYSRET	Y
20	XSAVES, XRSTORS, INVPCID	Y



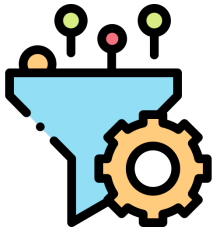
Triggering VM Exit and Stopping Execution.

- Using the Intel VT-x technique to configure the VM exits directly.

SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.

Line	Detailed Instructions		Is Privileged Instruction?
1	VM[RESUME|READ|WRITE|…], INVEPT, INVVPID	1	Y
2	INVD. XSETBV		Y
3	ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD…)		Y
4	RDMSR, WRMSR		Y
5	IN, OUT, IN[S|SB|SW|SD], OUT[S|SB|SW|SD]		Y
6	HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD		Y
7	LGDT, LLDT, LTR, LIDT		Y
8	MOV to/from DR0-DR7		Y
9	MOV to/from CR3, MOV to/from CR8		Y
10	MOV to/from CR0/CR4, CLTS, LMSW, SMSW	2	Y
11	MOV to/from CR2		Y
12	SWAPGS		Y
13	CLI, STI		Y
14	LAR, LSL. VERR, VERW		N
15	POPF, POPFQ		N
16	L[FS|DS|SS], MOV to [DS|ES|FS|GS|SS], POP [FS|GS]		N
17	Far CALL, Far RET, Far JMP		N
18	IRET, IRETD, IRETQ		Y
19	SYSEXIT, SYSRET		Y
20	XSAVES, XRSTORS, INVPCID		Y



Triggering VM Exit and Stopping Execution.

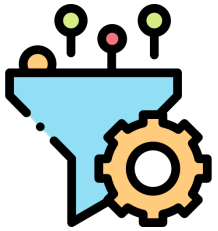
- Using the Intel VT-x technique to configure the VM exits directly.

Invalidating the Execution Effects.

- The execution does not change any state.

SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.



Line	Detailed Instructions		Is Privileged Instruction?
1	VM[RESUME|READ|WRITE|…], INVEPT, INVVPID	1	Y
2	INVD. XSETBV		Y
3	ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD…)		Y
4	RDMSR, WRMSR		Y
5	IN, OUT, IN[S|SB|SW|SD], OUT[S|SB|SW|SD]		Y
6	HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD		Y
7	LGDT, LLDT, LTR, LIDT		Y
8	MOV to/from DR0-DR7		Y
9	MOV to/from CR3, MOV to/from CR8		Y
10	MOV to/from CR0/CR4, CLTS, LMSW, SMSW	2	Y
11	MOV to/from CR2		Y
12	SWAPGS		Y
13	CLI, STI		Y
14	LAR, LSL. VERR, VERW		N
15	POPF, POPFQ		N
16	L[FS|DS|SS], MOV to [DS|ES|FS|GS|SS], POP [FS|GS]	3	N
17	Far CALL, Far RET, Far JMP		N
18	IRET, IRETD, IRETQ		Y
19	SYSEXIT, SYSRET		Y
20	XSAVES, XRSTORS, INVPCID		Y

Triggering VM Exit and Stopping Execution.

- Using the Intel VT-x technique to configure the VM exits directly.

Invalidating the Execution Effects.

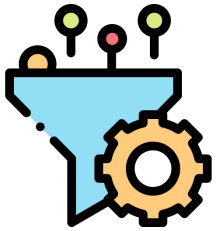
- The execution does not change any state.

Raising the Execution Exception and Stopping Execution.

- Configure the execution condition.

SEIMI —— Privileged Instruction Prevention Component

- We group them into **20 categories** based on their different functionality.



Line	Detailed Instructions		Is Privileged Instruction?
1	VM[RESUME|READ|WRITE|…], INVEPT, INVVPID	1	Y
2	INVD. XSETBV		Y
3	ENCLS(e.g., ECREATE, EADD, EINIT, EDBGRD…)		Y
4	RDMSR, WRMSR		Y
5	IN, OUT, IN[S|SB|SW|SD], OUT[S|SB|SW|SD]		Y
6	HLT, INVLPG, RDPMC, MONITOR, MWAIT, WBINVD		Y
7	LGDT, LLDT, LTR, LIDT		Y
8	MOV to/from DR0-DR7		Y
9	MOV to/from CR3, MOV to/from CR8		Y
10	MOV to/from CR0/CR4, CLTS, LMSW, SMSW	2	Y
11	MOV to/from CR2		Y
12	SWAPGS		Y
13	CLI, STI		Y
14	LAR, LSL. VERR, VERW		N
15	POPF, POPFQ		N
16	L[FS|DS|SS], MOV to [DS|ES|FS|GS|SS], POP [FS|GS]	3	N
17	Far CALL, Far RET, Far JMP		N
18	IRET, IRETD, IRETQ		Y
19	SYSEXIT, SYSRET		Y
20	XSAVES, XRSTORS, INVPCID		Y

Triggering VM Exit and Stopping Execution.

- Using the Intel VT-x technique to configure the VM exits directly.

Invalidating the Execution Effects.

- The execution does not change any state.

Raising the Execution Exception and Stopping Execution.

- Configure the execution condition.

SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.

- We choose to raise an exception during their execution and to trigger the VM exits.

SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.

- We choose to raise an exception during their execution and to trigger the VM exits.

1 **Raising the #UD (invalid opcode exception)**

- xsaves, xrstors, invpcid … via configuring the VMCS to disable the support in guest.

SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.

- We choose to raise an exception during their execution and to trigger the VM exits.

1 **Raising the #UD (invalid opcode exception)**

- xsaves, xrstors, invpcid … via configuring the VMCS to disable the support in guest.

2 **Raising the #PF (page fault exception)**

- sysexit, sysret… due to the S-page setting in all code pages.

SEIMI —— Privileged Instruction Prevention Component

Raising the Execution Exception.

– We choose to raise an exception during their execution and to trigger the VM exits.

1 **Raising the #UD (invalid opcode exception)**

– xsaves, xrstors, invpcid … via configuring the VMCS to disable the support in guest.

2 **Raising the #PF (page fault exception)**

– sysexit, sysret… due to the S-page setting in all code pages.

3 **Raising the #GP (general protection exception)**

– Segment-switching related instructions: mov to %ds, lcall…

SEIMI —— Privileged Instruction Prevention Component

- **Since the application runs in ring 0, attackers may use the segment-switching instructions to switch to any segment, we need to control them.**

SEIMI —— Privileged Instruction Prevention Component

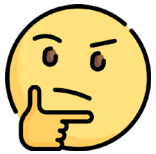
- **Since the application runs in ring 0, attackers may use the segment-switching instructions to switch to any segment, we need to control them.**

- **Problem:**
  - Intel VT-x cannot intercept these instructions that could change the segment.

SEIMI —— Privileged Instruction Prevention Component

- **Since the application runs in ring 0, attackers may use the segment-switching instructions to switch to any segment, we need to control them.**

- **Problem:**
  - Intel VT-x cannot intercept these instructions that could change the segment.



How to intercept this type of instructions ???

SEIMI —— Privileged Instruction Prevention Component

- **Observation**
  - When changing a segment register, the hardware will use the target selector to **access the segment descriptor table** to obtain the target segment information.

SEIMI —— Privileged Instruction Prevention Component

- **Observation**

  - When changing a segment register, the hardware will use the target selector to **access the segment descriptor table** to obtain the target segment information.
  - **If the segment descriptor table is empty**, the CPU will raise a general protection exception (#GP) in this process.

SEIMI —— Privileged Instruction Prevention Component

- **Observation**

  - When changing a segment register, the hardware will use the target selector to **access the segment descriptor table** to obtain the target segment information.
  - **If the segment descriptor table is empty**, the CPU will raise a general protection exception (#GP) in this process.

Solution —— Emptying out this table to intercept such instructions.

SEIMI —— Privileged Instruction Prevention Component

- **Observation**

  - When changing a segment register, the hardware will use the target selector to **access the segment descriptor table** to obtain the target segment information.
  - **If the segment descriptor table is empty**, the CPU will raise a general protection exception (#GP) in this process.

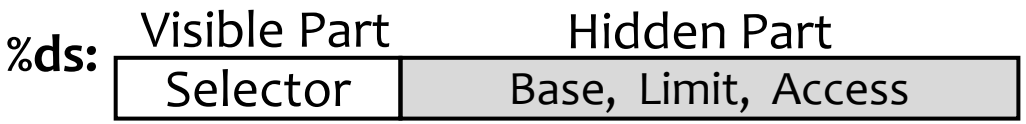
Solution —— Emptying out this table to intercept such instructions.

- **Problem:**

  - How to ensure the normal execution (segment addressing)?
  - How to ensure the correct functionality of the segment-switching instructions?

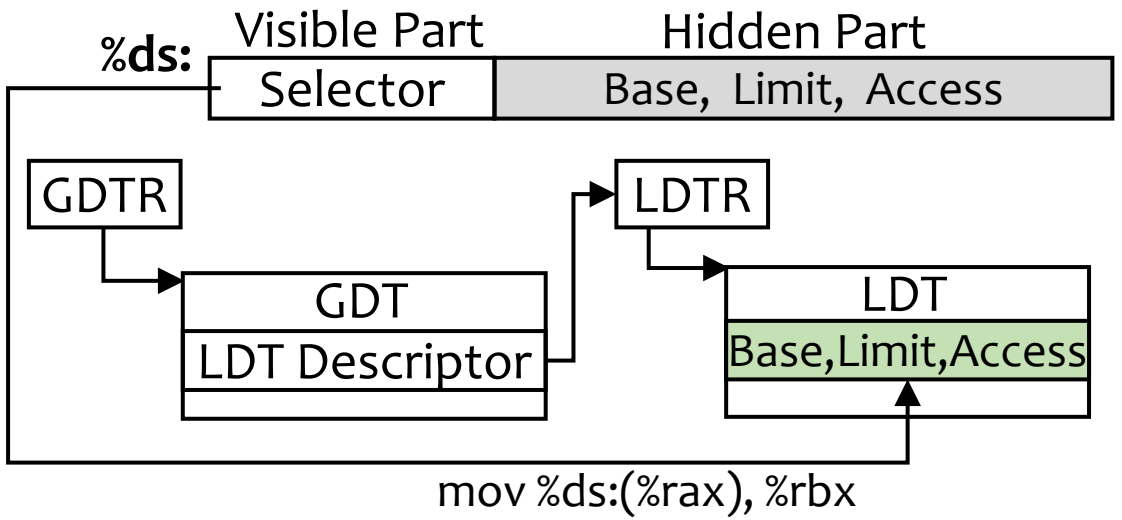
SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**



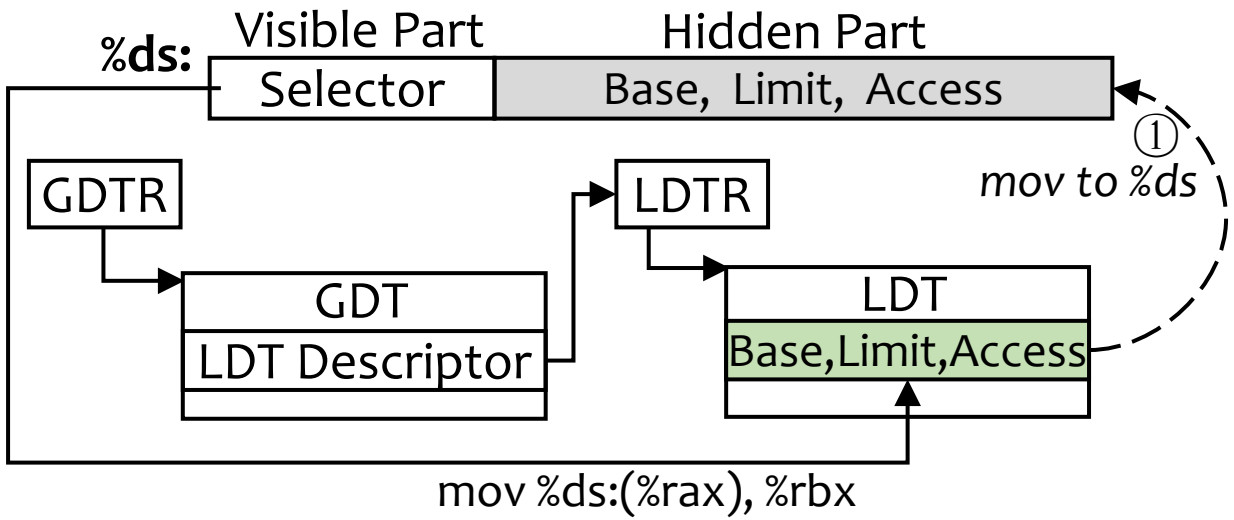
SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**



SEIMI —— Privileged Instruction Prevention Component

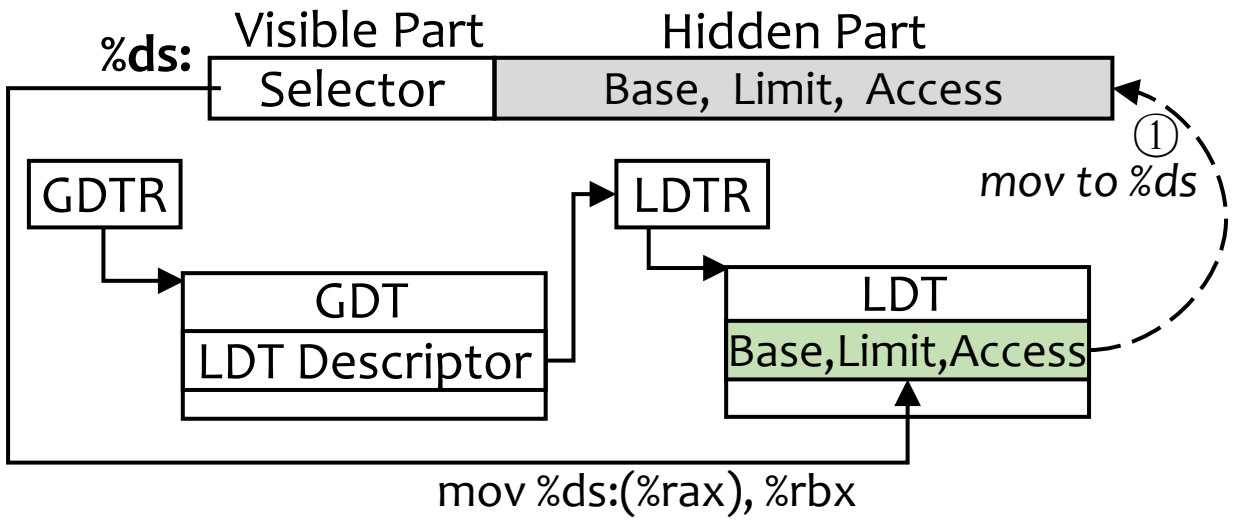
- **Segment-switching exception using descriptor cache.**



Segment-switching instruction will load the information into the descriptor cache.

SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.

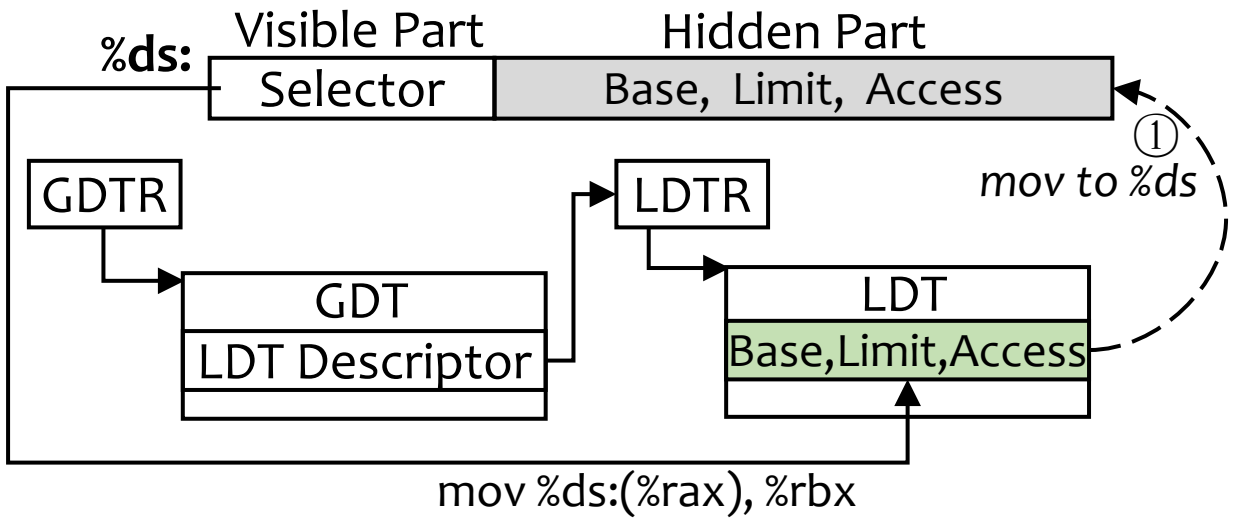


Segment-switching instruction will load the information into the descriptor cache.

SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.

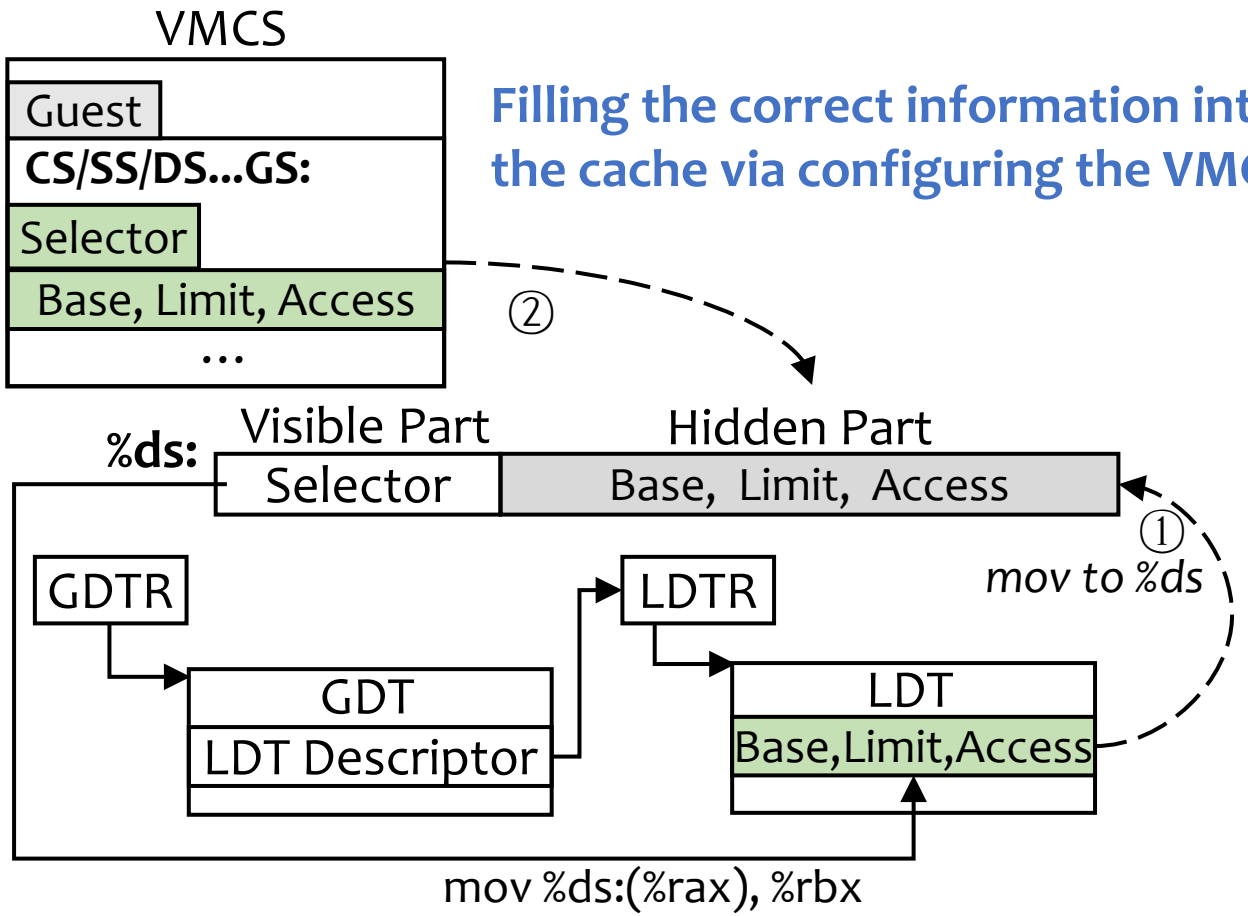
Filling the correct information into the cache via configuring the VMCS.



Segment-switching instruction will load the information into the descriptor cache.

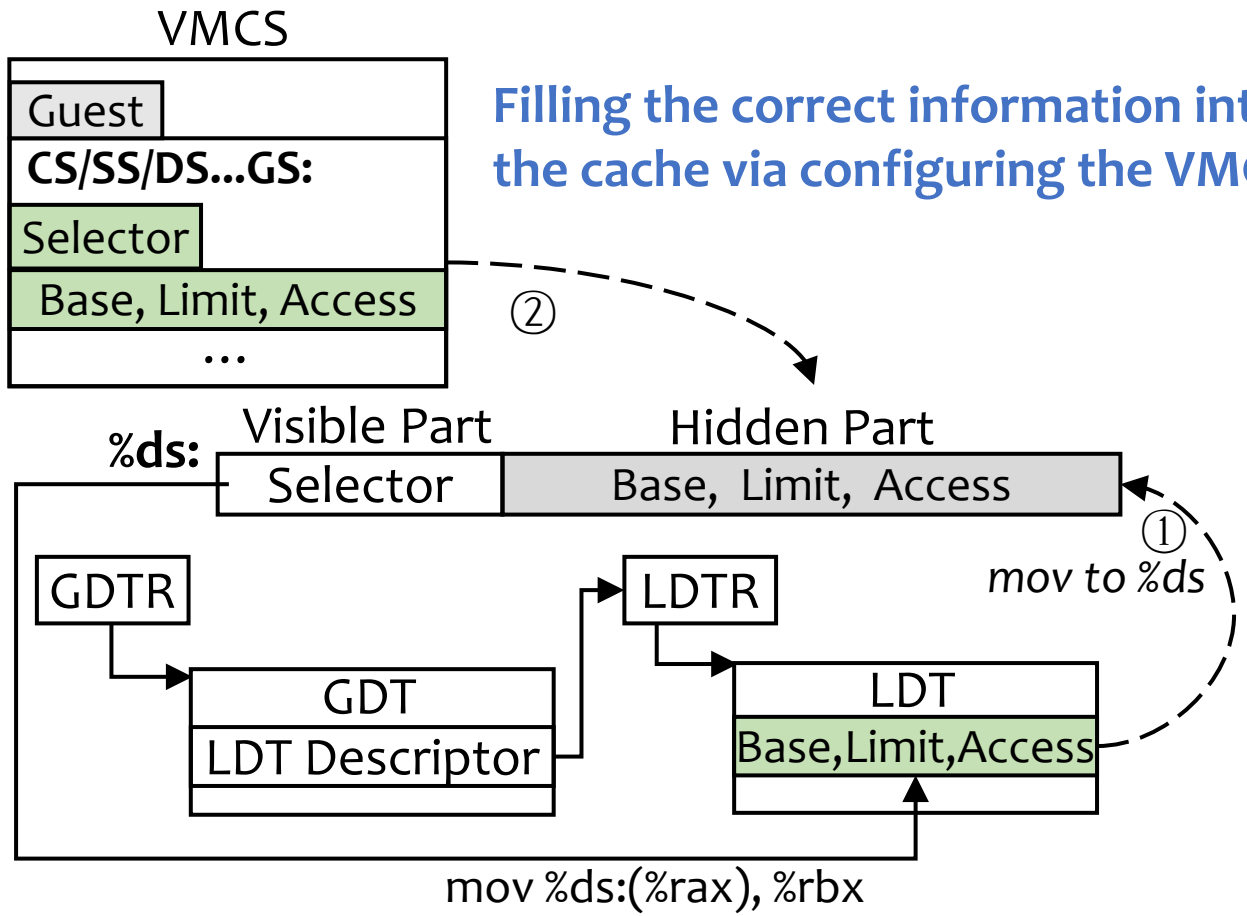
SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.



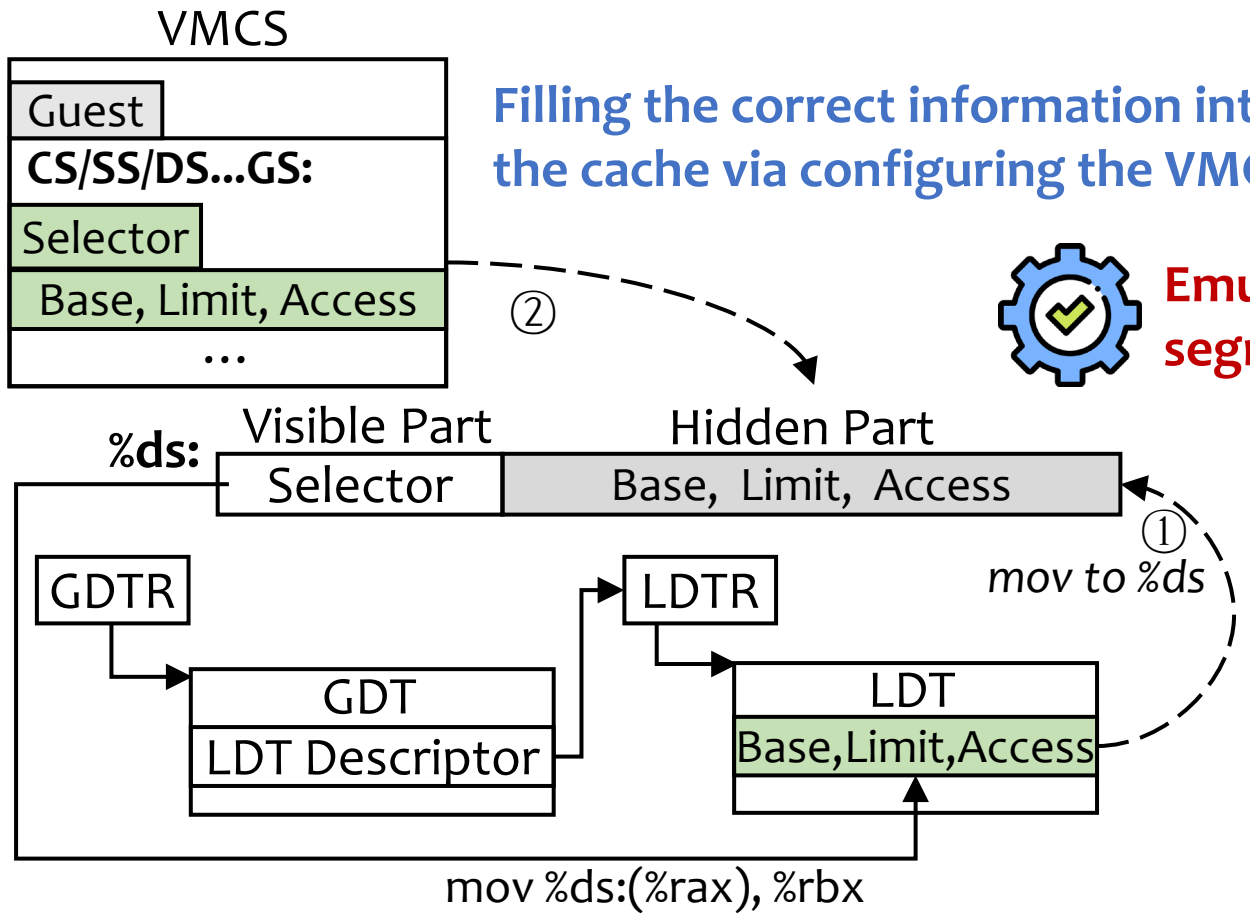
SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.



SEIMI —— Privileged Instruction Prevention Component

- **Segment-switching exception using descriptor cache.**
  - X86 allows the descriptor cache to be inconsistent with the descriptor table.



SEIMI —— Privileged Instruction Prevention Component

Invalidating the Execution Effects.

- We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.

SEIMI —— Privileged Instruction Prevention Component

Invalidating the Execution Effects.

- We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.

- **CR0/CR4-related instructions.**
  - Configure guest/host masks and read shadows in VMCS.
  - The value of the %CR0/%CR4 read is all 0.
  - Write to them does not really modify the values;

SEIMI —— Privileged Instruction Prevention Component

Invalidating the Execution Effects.

- We invalidate their execution effects, thus preventing attackers from using these instructions to obtain information or change any state.

- **CR0/CR4-related instructions.**
  - Configure guest/host masks and read shadows in VMCS.
  - The value of the %CR0/%CR4 read is all 0.
  - Write to them does not really modify the values;

- **SWAPGS, L[AR/SL], VER[R/W], CLI/STI …**
  - More details are in the paper.🔍

SEIMI —— Events Redirection Component

- **System-call Handling**
  - Convert the system calls to the hypercalls via mapping a code page.
    - Containing two instructions: **VMCALL** and **JMP *%RCX**.
    - The **IA32_LSTAR** MSR register in guest points to this page.

SEIMI —— Events Redirection Component

- **System-call Handling**
  - Convert the system calls to the hypercalls via mapping a code page.
    - Containing two instructions: **VMCALL** and **JMP *%RCX**.
    - The **IA32_LSTAR** MSR register in guest points to this page.
  - The kernel module vectors the *system_call_table* and calls the handlers.

SEIMI —— Events Redirection Component

- **System-call Handling**
  - Convert the system calls to the hypercalls via mapping a code page.
    - Containing two instructions: **VMCALL** and **JMP *%RCX**.
    - The **IA32_LSTAR** MSR register in guest points to this page.
  - The kernel module vectors the *system_call_table* and calls the handlers.

- **Interrupts and Exceptions Handling**
  - All these events trigger the VM exit via configuring the VMCS.
  - The kernel module checks the call gates and vectors the IDT.

SEIMI —— Events Redirection Component

- **System-call Handling**
  - Convert the system calls to the hypercalls via mapping a code page.
    - Containing two instructions: **VMCALL** and **JMP *%RCX**.
    - The **IA32_LSTAR** MSR register in guest points to this page.
  - The kernel module vectors the *system_call_table* and calls the handlers.

- **Interrupts and Exceptions Handling**
  - All these events trigger the VM exit via configuring the VMCS.
  - The kernel module checks the call gates and vectors the IDT.

- **Linux Signal Handling**
  - Check the signal queue, and switch the context via configuring the VMCS.

SEIMI —— Some Implementations

- **Memory Management Implementations**
  - Avoiding overlaps in the 254th and 255th entries.
  - Handling the VSYSCALL page.
  - Tracking updates of the PML4 page.
  - Avoiding accessing the kernel by exploiting the TLB.

- **Other Implementations**
  - Hardening system calls against confused deputy.
  - Starting and exiting the target process.
  - Supporting multi-threading and multi-processes.
  - Defeating the concurrent attacks.

SEIMI: Efficient and Secure SMAP-Enabled Intra-process Memory Isolation

Zhe Wang[1,2] Chenggang Wu[1,2] Mengyao Xie[1,2] Yinqian Zhang[3] Kangjie Lu[4]
Xiaofeng Zhang[1,2] Yuanming Lai[1,2] Yan Kang[1] Min Yang[5]

[1] *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences,*
[2] *University of Chinese Academy of Sciences,* [3] *The Ohio State University,* [4]*University of Minnesota,* [5]*Fudan University*

***Abstract*—Memory-corruption attacks such as code-reuse attacks and data-only attacks have been a key threat to systems security. To counter these threats, researchers have proposed a variety of defenses, including *control-flow integrity* (CFI), *code-pointer integrity* (CPI), and code (re-)randomization. All of them, to be effective, require a security primitive—intra-process protection of confidentiality and/or integrity for sensitive data (such as CFI's shadow stack and CPI's safe region).**

In this paper, we propose SEIMI, a highly efficient intra-process memory is[illegible]nique for memory-corruption defenses to prot[illegible]ata. The core of SEIMI is to use the c[illegible] *Access Prevention (SMAP)*, a hardwar[illegible] used for preventing the kernel fr[illegible] to achieve intra-process memory i[illegible]EIMI creatively executes the user c[illegible] addition to enabling the new desig[illegible]ry isolation, we further develop mi[illegible]nsure secure escalation of user code, e.[illegible]es to capture the potential segment operat[illegible]*irtual Machine Control Structure* (VMCS) [illegible] result of the control registers related operations. Exten[illegible]tal results show that SEIMI outperforms existing isola[illegible]ms, including both the *Memory Protection Keys* (M[illegible] scheme and the *Memory Protection Extensions* (MPX[illegible]scheme, while providing secure memory isolation.

I. Introduction

Memory-corruption attacks such as control-flow hijacking and data-only attacks have been a major threat to systems security in the past decades. To defend against such attacks, researchers have proposed a variety of advanced mechanisms, including enhanced control-flow integrity (CFI), code-pointer integrity (CPI), fine-grained code (re-)randomization, and data-layout randomization. All these techniques require a security primitive—effective intra-process memory protection of the integrity and/or confidentiality of sensitive data from potentially compromised code. The sensitive data includes critical data structures that are frequently checked against or used for protection. For example, O-CFI [39] uses a *bounds lookup table* (BLT), and CCFIR [58] uses a *safe SpringBoard* to restrict the control flow; CPI [31] uses a *safe region*, and Shuffler [55] uses a *code-pointer table* to protect the sensitive pointers; Oxymoron [6] maintains a *sensitive translation table*, and Isomeron [19] uses a table to protect randomization secrets. The effectiveness of all such techniques heavily depends on the integrity and/or confidentiality of the sensitive data.

To efficiently protect sensitive data, researchers proposed *information hiding* (IH) which stores sensitive data in a memory region allocated in a *random* address and wishes that attackers could not know the random address thus could not write or read the sensitive data. Unfortunately, recent works show that memory disclosures and side channels can be exploited to readily reduce the randomization entropy and thus to bypass the information hiding [22–24, 36, 41]. As such, even a robust IH-based defense can be defeated.

To address this problem, recent research instead opts for *practical memory isolation* which provides efficient protection with a stronger security guarantee. Memory isolation, in general, can be classified into *address-based* isolation and *domain-based* isolation. Address-based isolation checks (e.g., bound-check) *each* memory access from untrusted code to ensure that it cannot access the sensitive data. The main overhead of this method is brought by the code that performs the checks. The most efficient address-based isolation is based on Intel Memory Protection Extensions (MPX), which performs bound-checking with hardware support [30].

Domain-based isolation instead stores sensitive data in a protected memory region. The permission to accessing this region is granted when requested by the trusted code, and is revoked when the trusted access finished. However, memory accesses from untrusted code (i.e., the potentially vulnerable code that can be compromised by attackers) cannot enable the permission. The main source of the performance overhead of domain-based memory isolation is the operations for enabling and disabling the memory-access permissions. The most efficient domain-based isolation is to use Intel Memory Protection Keys (MPK) [25, 30, 40, 47].

In general, existing address-based isolation and domain-based isolation both incur non-trivial performance overhead compared to the IH-based scheme. Worse, the overhead will be significantly elevated when the workloads (i.e., the frequency of memory accesses that require bound-checking or permission switching) increase. For example, when protecting the shadow stack, the MPK-based scheme (i.e., domain-based) incurs a runtime overhead of 61.18% [40]. When protecting the safe region of CPI using the MPX-based scheme (i.e., address-

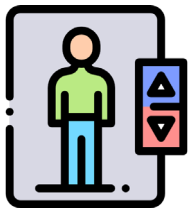
Outline

Motivation

High-level Design

Approach Overview

SEIMI System



Evaluation

Performance Evaluation

- **Defenses and Isolation Schemes:**
  - Defenses: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  - Isolation: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes

Performance Evaluation

- **Defenses and Isolation Schemes:**
  - Defenses: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  - Isolation: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes
- **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  - lmbench v3.0-a9

Performance Evaluation

- **Defenses and Isolation Schemes:**
  - Defenses: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  - Isolation: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes
- **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  - lmbench v3.0-a9
- **Macrobenchmark** —— the overheads on different isolation schemes.
  - SPEC CPU2006 C/C++ benchmark with the *ref* input.

Performance Evaluation

- **Defenses and Isolation Schemes:**
  - Defenses: O-CFI, Shadow Stack (SS), Code Pointer Integrity (CPI), and ASLR-Guard (AG)
  - Isolation: IH-based (randomization), MPX-based, MPK-based, and SEIMI-based schemes
- **Microbenchmark** —— the overheads imposed by SEIMI on kernel operations.
  - lmbench v3.0-a9
- **Macrobenchmark** —— the overheads on different isolation schemes.
  - SPEC CPU2006 C/C++ benchmark with the *ref* input.
- **Real-world applications:**
  - 4 Web servers: Nginx, Apache, Lighttpd, and Openlitespeed.
  - 4 Databases: MySQL, SQLite, Redis, and Memcached.
  - 4 JavaScript engines: ChakraCore, Google V8, JavaScriptCore, SpiderMonkey.

Microbenchmark —— lmbench

- **We run *lmbench* directly on SEIMI to only evaluate the overhead on kernel operations.**

Microbenchmark —— lmbench

- **We run *lmbench* directly on SEIMI to only evaluate the overhead on kernel operations.**

Config	null call	null I/O	stat	open close	select TCP	signal install	signal handle	fork proc	exec proc	sh proc
Native	0.21	0.26	0.57	1.23	5.35	0.27	0.99	355	870	2162
SEIMI	**0.71**	**0.82**	**1.33**	**2.58**	6.11	**0.79**	**3.02**	463	1029	2368
Slowdown	**2.4X**	**2.2X**	**1.3X**	**1.1X**	14%	**1.9X**	**2.1X**	30.4%	18.3%	9.5%

1 Latency on process-related kernel operations (in µs): smaller is better.

Microbenchmark —— lmbench

- **We run *lmbench* directly on SEIMI to only evaluate the overhead on kernel operations.**

Config	null call	null I/O	stat	open close	select TCP	signal install	signal handle	fork proc	exec proc	sh proc
Native	0.21	0.26	0.57	1.23	5.35	0.27	0.99	355	870	2162
SEIMI	**0.71**	**0.82**	**1.33**	**2.58**	6.11	**0.79**	**3.02**	463	1029	2368
Slowdown	**2.4X**	**2.2X**	**1.3X**	**1.1X**	14%	**1.9X**	**2.1X**	30.4%	18.3%	9.5%

1 Latency on process-related kernel operations (in µs): smaller is better.

Config	2p/0K	2p/16K	2p/64K	8p/16K	8p/64K	16p/16K	16p/64K
Native	2.05	2.06	3.1	8.13	12.2	8.43	12.6
SEIMI	2.46	2.45	3.6	10.1	14.8	11.52	15.9
Slowdown	20.0%	18.9%	16.1%	24.2%	21.3%	36.7%	26.2%

2 Context-switching latency (in µs): smaller is better.

Microbenchmark —— lmbench

- **We run *lmbench* directly on SEIMI to only evaluate the overhead on kernel operations.**

Config	null call	null I/O	stat	open close	select TCP	signal install	signal handle	fork proc	exec proc	sh proc
Native	0.21	0.26	0.57	1.23	5.35	0.27	0.99	355	870	2162
SEIMI	**0.71**	**0.82**	**1.33**	**2.58**	6.11	**0.79**	**3.02**	463	1029	2368
Slowdown	**2.4X**	**2.2X**	**1.3X**	**1.1X**	14%	**1.9X**	**2.1X**	30.4%	18.3%	9.5%

1 Latency on process-related kernel operations (in µs): smaller is better.

Config	2p/0K	2p/16K	2p/64K	8p/16K	8p/64K	16p/16K	16p/64K
Native	2.05	2.06	3.1	8.13	12.2	8.43	12.6
SEIMI	2.46	2.45	3.6	10.1	14.8	11.52	15.9
Slowdown	20.0%	18.9%	16.1%	24.2%	21.3%	36.7%	26.2%

2 Context-switching latency (in µs): smaller is better.

Config	0K File Create	0K File Delete	10K File Create	10K File Delete	Mmap Latency	Prot Fault	Page Fault	100fd select
Native	5.4717	4.7816	10.9	6.6214	6779	0.636	0.1593	1.016
SEIMI	6.9623	5.3421	14.5	7.4527	12500	1.038	0.2128	1.705
Slowdown	27.2%	11.7%	33.0%	12.6%	84.4%	63.2%	33.6%	67.8%

3 File & VM system latency (in µs): smaller is better.

Microbenchmark —— lmbench

- **We run *lmbench* directly on SEIMI to only evaluate the overhead on kernel operations.**

Config	null call	null I/O	stat	open close	select TCP	signal install	signal handle	fork proc	exec proc	sh proc
Native	0.21	0.26	0.57	1.23	5.35	0.27	0.99	355	870	2162
SEIMI	**0.71**	**0.82**	**1.33**	**2.58**	6.11	**0.79**	**3.02**	463	1029	2368
Slowdown	**2.4X**	**2.2X**	**1.3X**	**1.1X**	14%	**1.9X**	**2.1X**	30.4%	18.3%	9.5%

1 Latency on process-related kernel operations (in µs): smaller is better.

Config	2p/0K	2p/16K	2p/64K	8p/16K	8p/64K	16p/16K	16p/64K
Native	2.05	2.06	3.1	8.13	12.2	8.43	12.6
SEIMI	2.46	2.45	3.6	10.1	14.8	11.52	15.9
Slowdown	20.0%	18.9%	16.1%	24.2%	21.3%	36.7%	26.2%

2 Context-switching latency (in µs): smaller is better.

Config	0K File Create	0K File Delete	10K File Create	10K File Delete	Mmap Latency	Prot Fault	Page Fault	100fd select
Native	5.4717	4.7816	10.9	6.6214	6779	0.636	0.1593	1.016
SEIMI	6.9623	5.3421	14.5	7.4527	12500	1.038	0.2128	1.705
Slowdown	27.2%	11.7%	33.0%	12.6%	84.4%	63.2%	33.6%	67.8%

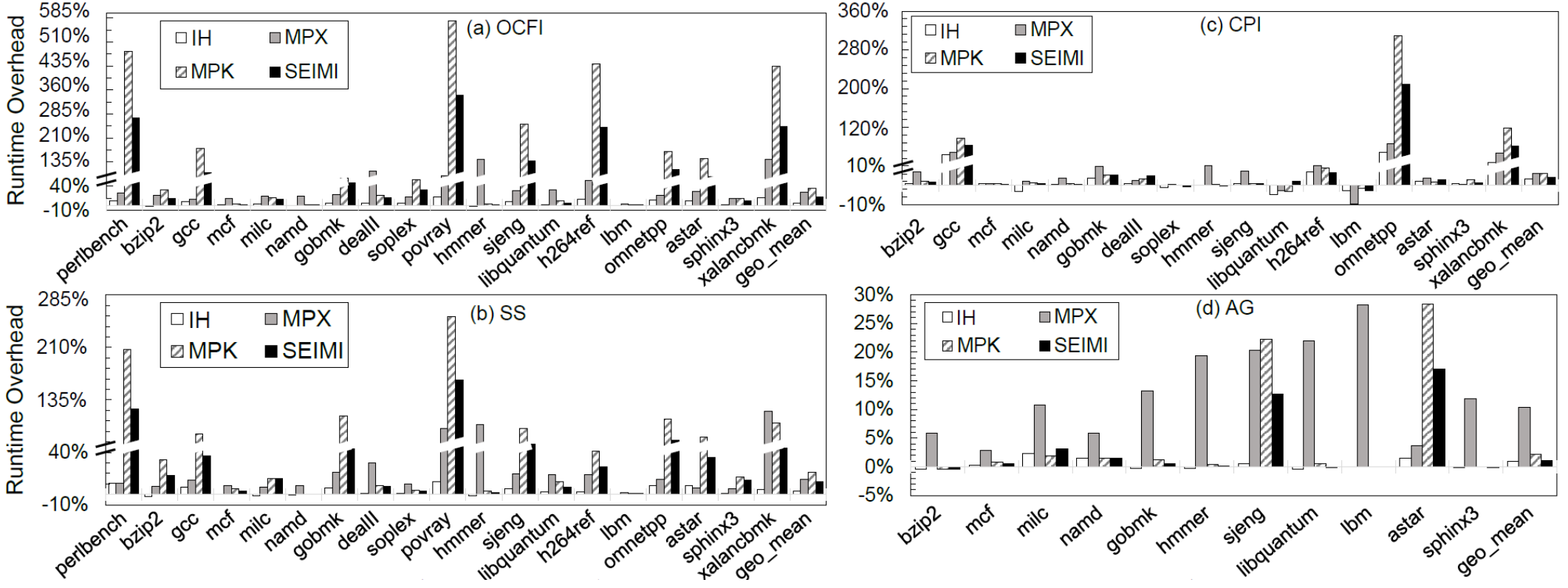
3 File & VM system latency (in µs): smaller is better.

Config	Pipe	AF UNIX	UDP	RPC/ UDP	TCP	RPC/ TCP	TCP conn
Native	5.582	9.2	9.883	14.9	13.9	17.6	22
SEIMI	7.428	11.7	11.7	20	17.6	23.9	24
Slowdown	33.1%	27.2%	18.4%	34.2%	26.6%	35.8%	9.1%

4 Local-communication latency (in µs): smaller is better.

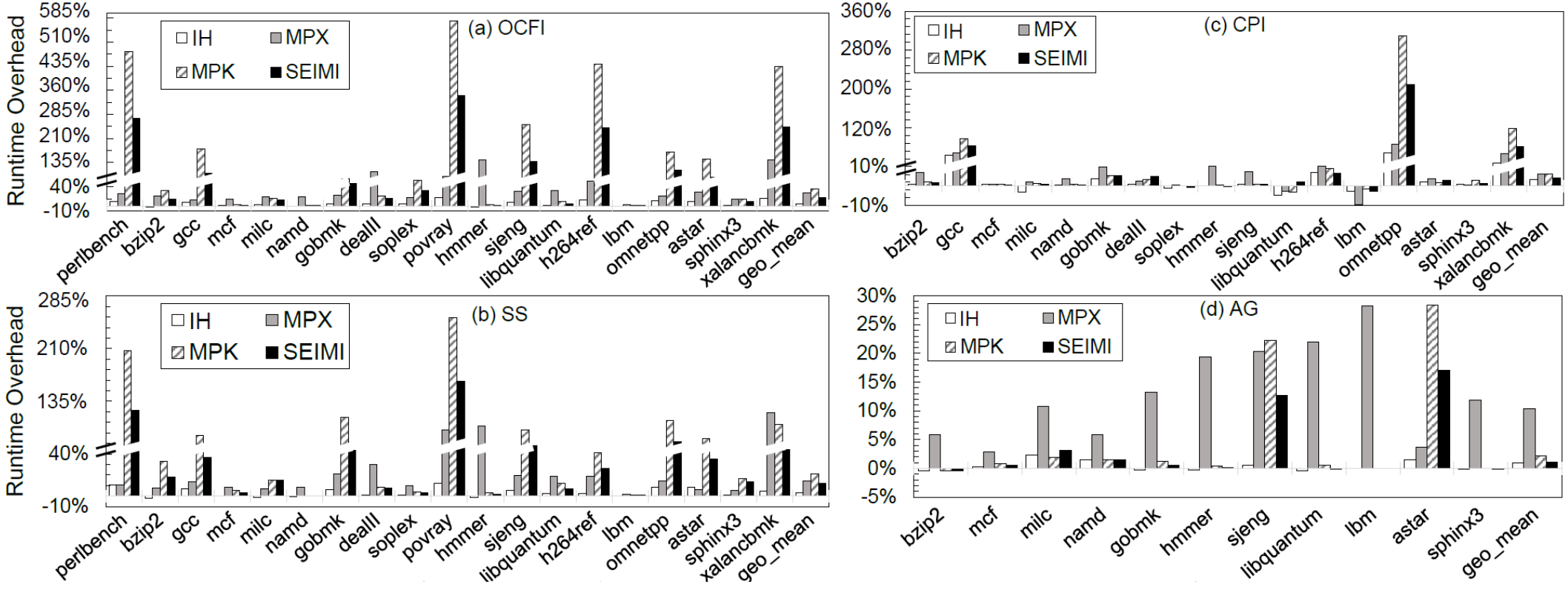
Macrobenchmark —— SPEC CPU 2006 benchmark

- Compared with the **MPX-based scheme**, **SEIMI** achieves a lower performance overhead on average, with the average reduction of **33.97%**.



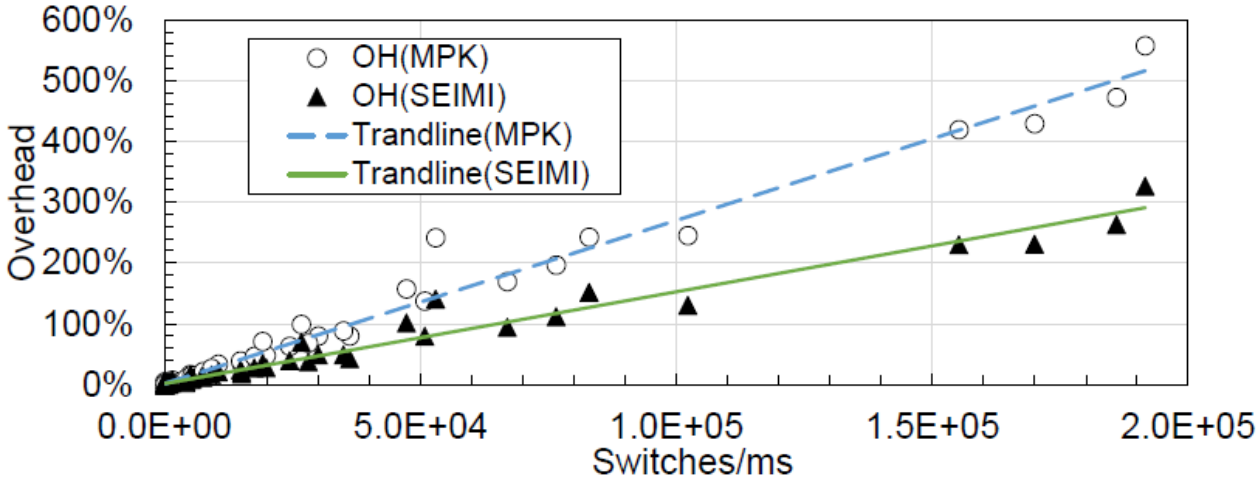
Macrobenchmark —— SPEC CPU 2006 benchmark

- Compared with the **MPX-based scheme**, **SEIMI** achieves a lower performance overhead on average, with the average reduction of **33.97%**.
- Compared to the **MPK-based scheme**, **SEIMI** is more efficient in almost all test cases, and with the average reduction of **42.3%** (maximum is **133.33%**).



Macrobenchmark —— SPEC CPU 2006 benchmark

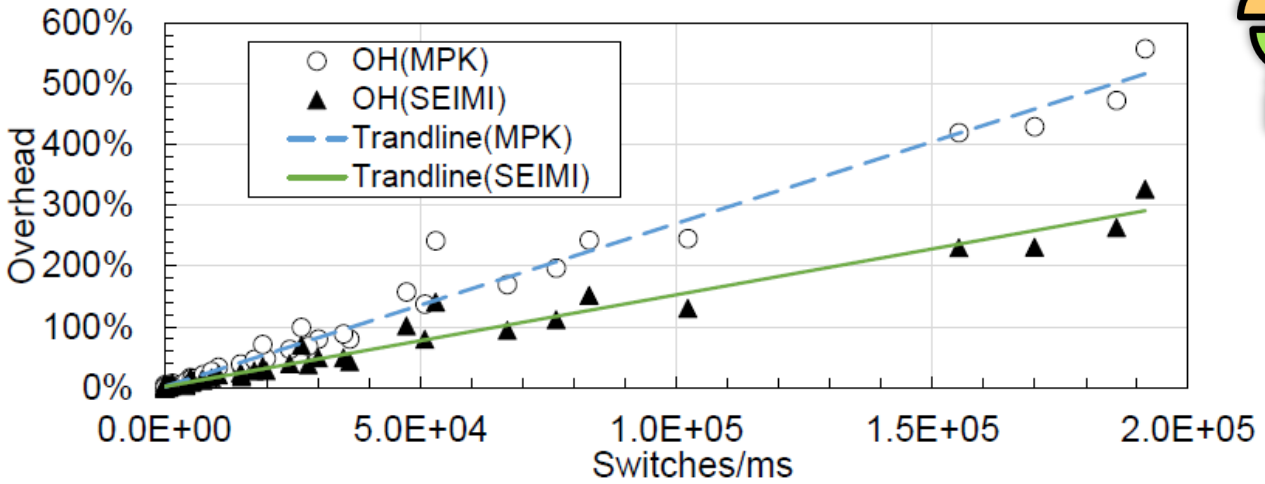
- **Performance Analysis: MPK vs. SEIMI**



The impact of permission-switching frequency on performance of MPK and SEIMI.

Macrobenchmark —— SPEC CPU 2006 benchmark

- **Performance Analysis: MPK vs. SEIMI**

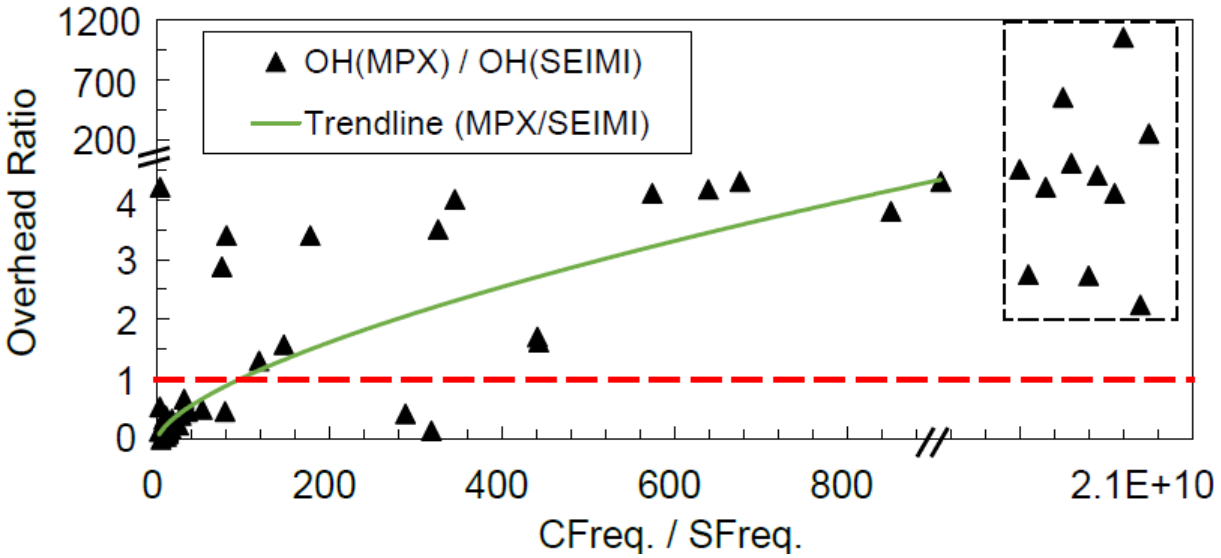


The impact of permission-switching frequency on performance of MPK and SEIMI.

Compared to **MPK**, as the access permission switching frequency increases, the performance gain of **SEIMI** becomes more apparent.

Macrobenchmark —— SPEC CPU 2006 benchmark

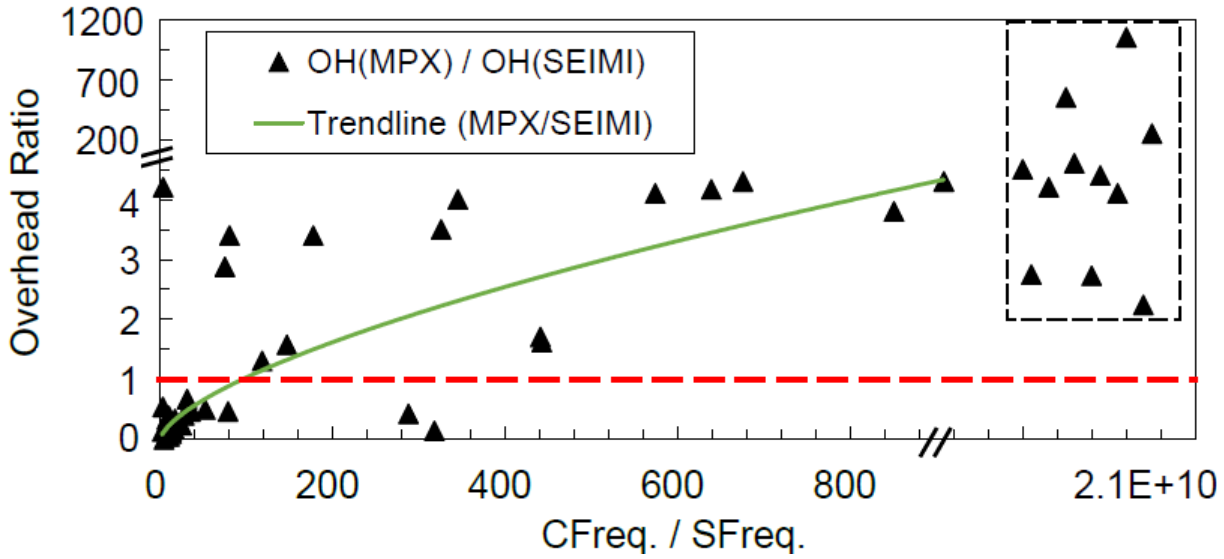
- **Performance Analysis: MPX vs. SEIMI**



The impact of bound-checking frequency (CFreq) and permission-switching frequency (SFreq) on performance.

Macrobenchmark —— SPEC CPU 2006 benchmark

- **Performance Analysis: MPX vs. SEIMI**



When the bound-checking frequency is **52** times of the access permission switching frequency, **SEIMI** is more efficient than **MPX** in most cases.

The impact of bound-checking frequency (CFreq) and permission-switching frequency (SFreq) on performance.

Real-world Applications

- **SEIMI is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.**

Applications	OCFI				SS				CPI				AG			
	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI
Nginx	1.10%	3.86%	5.32%	1.77%	1.86%	7.33%	10.49%	2.43%	0.90%	6.38%	8.95%	3.08%	0.74%	7.60%	5.27%	2.01%
Apache	1.58%	4.71%	2.82%	1.82%	1.64%	6.36%	6.83%	2.15%	1.45%	5.01%	2.58%	1.80%	—	—	—	—
Lighttpd	**2.94%**	**3.42%**	**5.74%**	**4.46%**	2.77%	6.85%	6.33%	3.78%	1.70%	6.83%	3.42%	2.46%	—	—	—	—
Openlitespeed	1.44%	5.39%	3.88%	1.61%	1.04%	1.92%	3.39%	1.42%	0.91%	2.89%	2.99%	1.38%	—	—	—	—
MySQL	1.75%	12.09%	8.08%	3.79%	3.17%	9.60%	11.99%	3.94%	—	—	—	—	—	—	—	—
SQLite	1.61%	2.11%	2.70%	1.84%	1.42%	3.46%	2.19%	1.94%	1.36%	3.11%	2.66%	2.18%	—	—	—	—
Redis	**4.51%**	**5.46%**	**13.12%**	**10.31%**	**1.18%**	**2.81%**	**5.36%**	**5.06%**	1.24%	4.47%	4.81%	3.93%	—	—	—	—
Memcached	1.64%	6.64%	7.46%	2.74%	2.38%	5.57%	8.13%	3.44%	1.04%	6.02%	7.28%	1.60%	—	—	—	—
ChakraCore	3.03%	12.09%	9.90%	4.10%	4.37%	7.92%	10.09%	5.15%	—	—	—	—	—	—	—	—
V8	2.57%	11.63%	5.04%	3.37%	2.05%	8.01%	4.05%	2.96%	—	—	—	—	—	—	—	—
JavaScriptCore	**2.22%**	**22.87%**	**39.65%**	**26.81%**	20.69%	38.34%	47.77%	31.82%	—	—	—	—	—	—	—	—
SpiderMonkey	1.75%	9.32%	7.63%	4.15%	1.84%	7.56%	7.79%	5.19%	—	—	—	—	—	—	—	—

All overheads are normalized to the unprotected applications. "—" represents the defense failed to compile or run it.

Real-world Applications

- **SEIMI is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.**
  - SEIMI is much more efficient than MPK for all 32 cases.

Applications	OCFI				SS				CPI				AG			
	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI
Nginx	1.10%	3.86%	5.32%	1.77%	1.86%	7.33%	10.49%	2.43%	0.90%	6.38%	8.95%	3.08%	0.74%	7.60%	5.27%	2.01%
Apache	1.58%	4.71%	2.82%	1.82%	1.64%	6.36%	6.83%	2.15%	1.45%	5.01%	2.58%	1.80%	—	—	—	—
Lighttpd	**2.94%**	**3.42%**	**5.74%**	**4.46%**	2.77%	6.85%	6.33%	3.78%	1.70%	6.83%	3.42%	2.46%	—	—	—	—
Openlitespeed	1.44%	5.39%	3.88%	1.61%	1.04%	1.92%	3.39%	1.42%	0.91%	2.89%	2.99%	1.38%	—	—	—	—
MySQL	1.75%	12.09%	8.08%	3.79%	3.17%	9.60%	11.99%	3.94%	—	—	—	—	—	—	—	—
SQLite	1.61%	2.11%	2.70%	1.84%	1.42%	3.46%	2.19%	1.94%	1.36%	3.11%	2.66%	2.18%	—	—	—	—
Redis	**4.51%**	**5.46%**	**13.12%**	**10.31%**	**1.18%**	**2.81%**	**5.36%**	**5.06%**	1.24%	4.47%	4.81%	3.93%	—	—	—	—
Memcached	1.64%	6.64%	7.46%	2.74%	2.38%	5.57%	8.13%	3.44%	1.04%	6.02%	7.28%	1.60%	—	—	—	—
ChakraCore	3.03%	12.09%	9.90%	4.10%	4.37%	7.92%	10.09%	5.15%	—	—	—	—	—	—	—	—
V8	2.57%	11.63%	5.04%	3.37%	2.05%	8.01%	4.05%	2.96%	—	—	—	—	—	—	—	—
JavaScriptCore	**2.22%**	**22.87%**	**39.65%**	**26.81%**	20.69%	38.34%	47.77%	31.82%	—	—	—	—	—	—	—	—
SpiderMonkey	1.75%	9.32%	7.63%	4.15%	1.84%	7.56%	7.79%	5.19%	—	—	—	—	—	—	—	—

All overheads are normalized to the unprotected applications. "—" represents the defense failed to compile or run it.

Real-world Applications

- **SEIMI is more performant than MPX-based and MPK-based schemes on protecting the real-world applications.**
  - SEIMI is much more efficient than MPK for all 32 cases.
  - SEIMI is much more efficient than MPX for 28 cases.

Applications	OCFI				SS				CPI				AG			
	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI	IH	MPX	MPK	SEIMI
Nginx	1.10%	3.86%	5.32%	1.77%	1.86%	7.33%	10.49%	2.43%	0.90%	6.38%	8.95%	3.08%	0.74%	7.60%	5.27%	2.01%
Apache	1.58%	4.71%	2.82%	1.82%	1.64%	6.36%	6.83%	2.15%	1.45%	5.01%	2.58%	1.80%	—	—	—	—
Lighttpd	**2.94%**	**3.42%**	**5.74%**	**4.46%**	2.77%	6.85%	6.33%	3.78%	1.70%	6.83%	3.42%	2.46%	—	—	—	—
Openlitespeed	1.44%	5.39%	3.88%	1.61%	1.04%	1.92%	3.39%	1.42%	0.91%	2.89%	2.99%	1.38%	—	—	—	—
MySQL	1.75%	12.09%	8.08%	3.79%	3.17%	9.60%	11.99%	3.94%	—	—	—	—	—	—	—	—
SQLite	1.61%	2.11%	2.70%	1.84%	1.42%	3.46%	2.19%	1.94%	1.36%	3.11%	2.66%	2.18%	—	—	—	—
Redis	**4.51%**	**5.46%**	**13.12%**	**10.31%**	**1.18%**	**2.81%**	**5.36%**	**5.06%**	1.24%	4.47%	4.81%	3.93%	—	—	—	—
Memcached	1.64%	6.64%	7.46%	2.74%	2.38%	5.57%	8.13%	3.44%	1.04%	6.02%	7.28%	1.60%	—	—	—	—
ChakraCore	3.03%	12.09%	9.90%	4.10%	4.37%	7.92%	10.09%	5.15%	—	—	—	—	—	—	—	—
V8	2.57%	11.63%	5.04%	3.37%	2.05%	8.01%	4.05%	2.96%	—	—	—	—	—	—	—	—
JavaScriptCore	**2.22%**	**22.87%**	**39.65%**	**26.81%**	20.69%	38.34%	47.77%	31.82%	—	—	—	—	—	—	—	—
SpiderMonkey	1.75%	9.32%	7.63%	4.15%	1.84%	7.56%	7.79%	5.19%	—	—	—	—	—	—	—	—

All overheads are normalized to the unprotected applications. "—" represents the defense failed to compile or run it.

Conclusion

- **We propose a highly efficient intra-process memory isolation technique SEIMI, which leverages the widely used hardware feature — SMAP.**

- **To avoid introducing security threats, we propose multiple new techniques to ensure the user code run in ring 0 securely.**

- **We believe that SEIMI can not only benefit existing defenses, but also open the new research direction …**
  - Enabling the efficient access to a variety of privileged hardware features, which does not require context switch, to defenses.

Any Questions ?

wangzhe12@ict.ac.cn