Published in The 41st IEEE Symposium on Security and Privacy (Oakland'20)

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

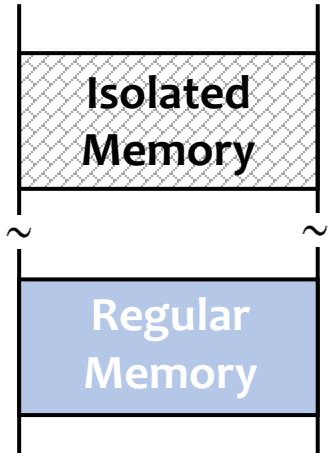
- **Memory corruption defenses need to keep their metadata safe.**
  - The safe region in CPI, the shadow stack in CFI, the randomization secrets in …
  - The software-based randomization method has been proven to be vulnerable. ⚠

- **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.

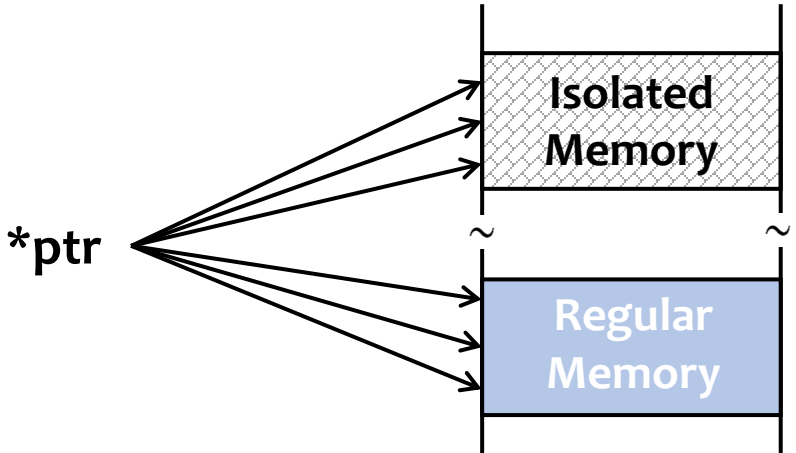
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



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. ⚠

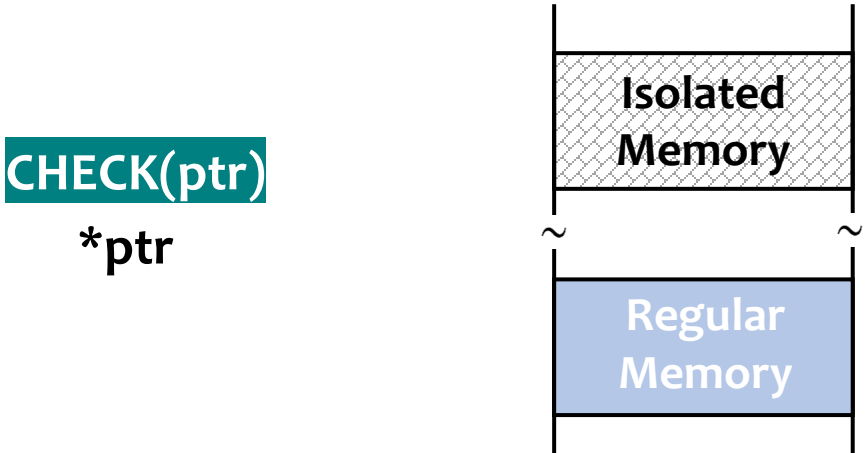
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



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. ⚠

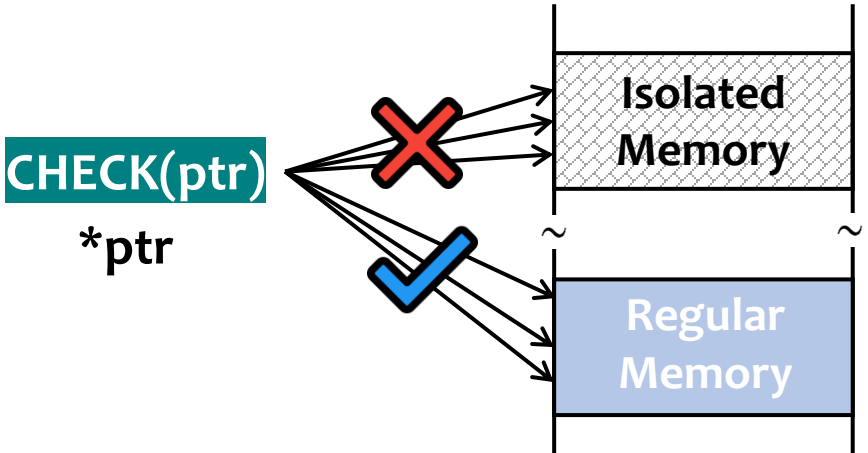
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



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.

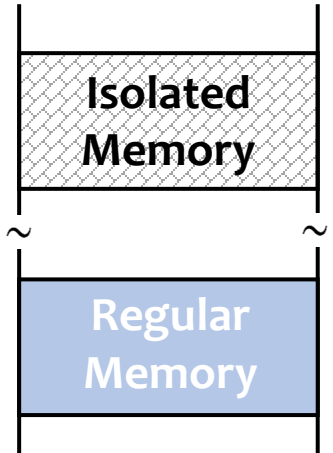
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.



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.

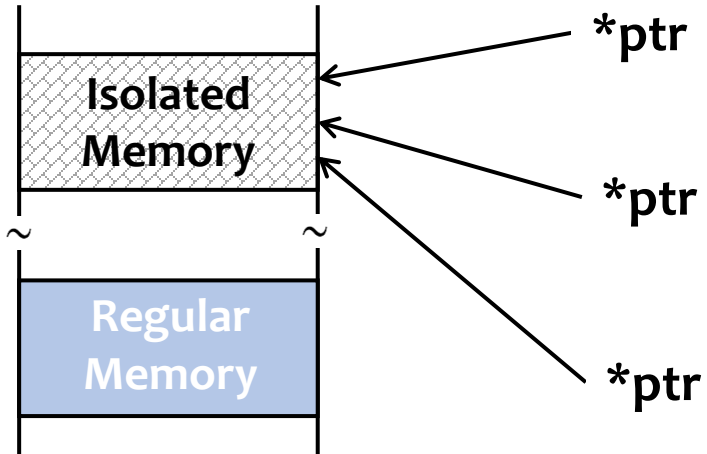
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.



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. ⚠

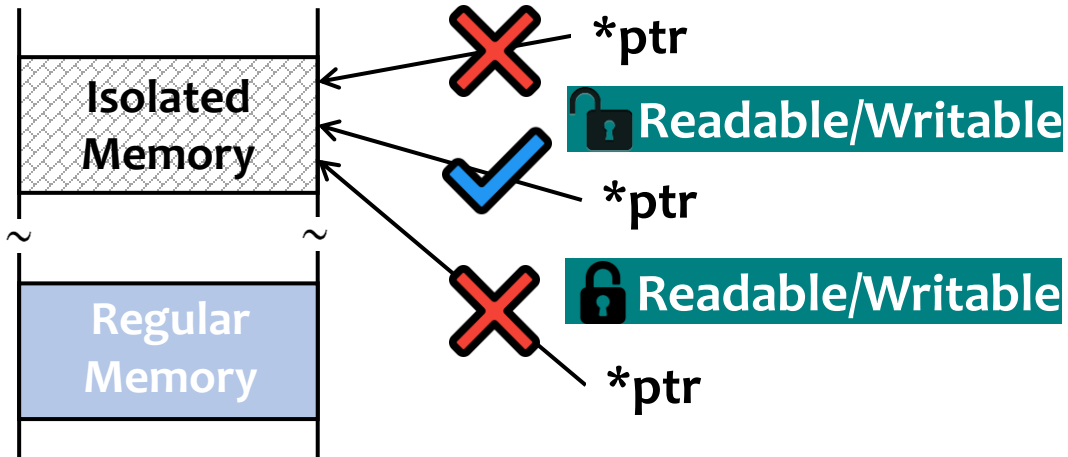
- **The strict memory isolations for the metadata in defenses are needed.**
  - Intel MPX uses bounds checks for isolation.
  - Intel MPK changes permissions of pages.



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.

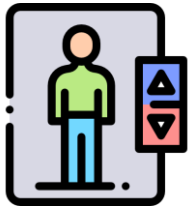
Threat Model

- **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.

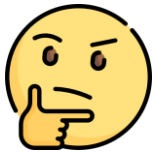
The user-mode hardware features are not fast.

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.

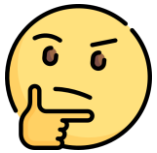
How about the privileged hardware ?



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.

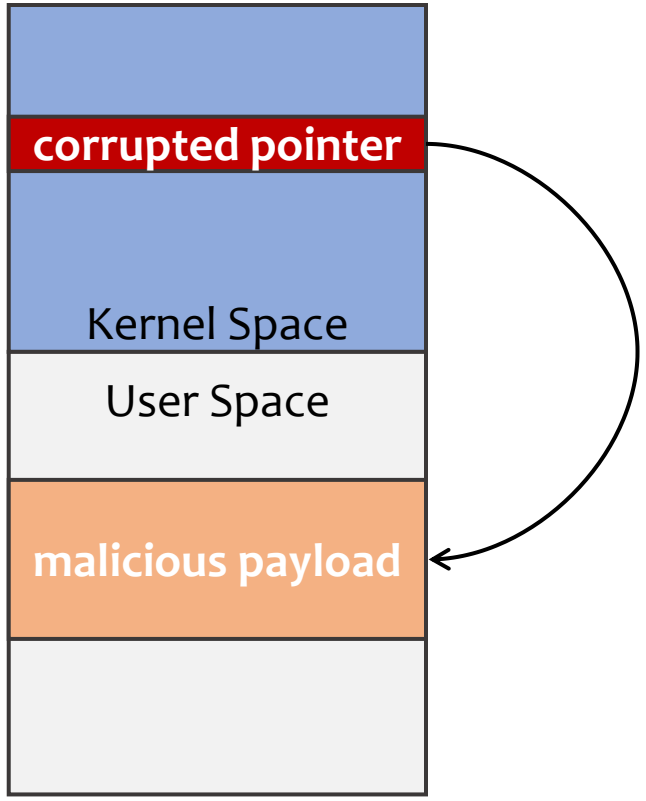


How about the privileged hardware ?

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

Motivation —— SMAP in Processors 101

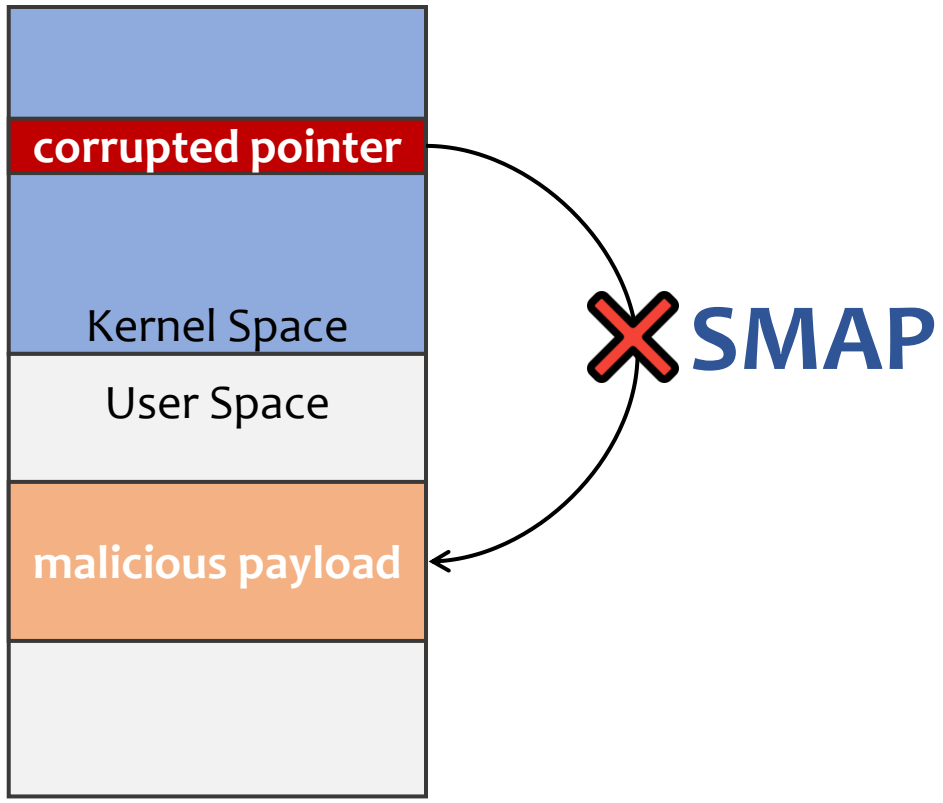
- To prevent the kernel from inadvertently accessing malicious data in user space,
  - dereferencing a corrupted data pointer



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 executed 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 executed 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.

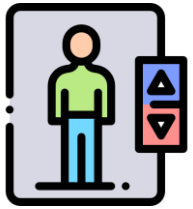
- **POPFQ and STAC/CLAC could modify the RFLAGS.AC flag.**
  - **popfq** could be executed in S-mode (ring 0-2).
  - **stac/clac** are privileged instructions that can only be execute in ring 0.

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 changing memory access permission.

Outline

Motivation



High-level Design

Approach Overview

SEIMI System

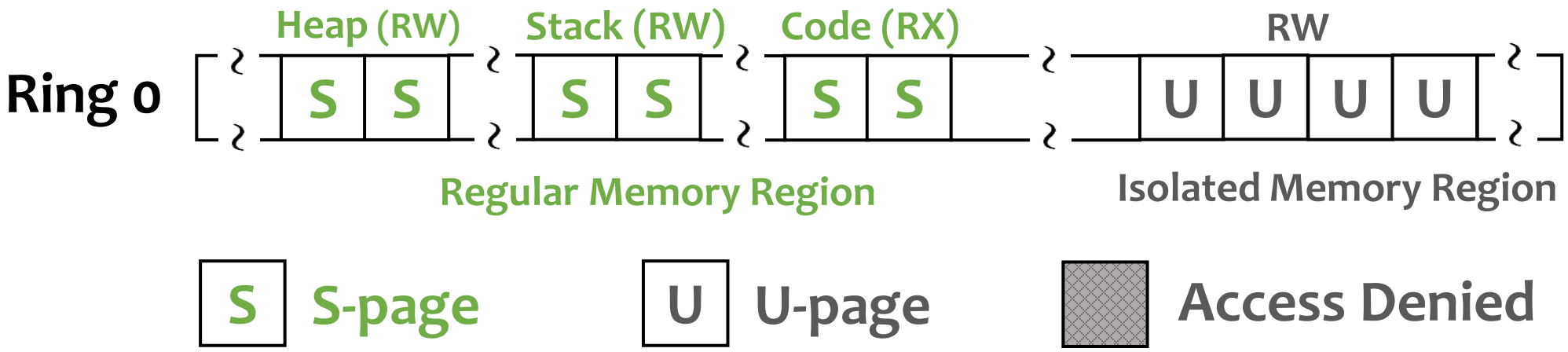
Evaluation

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.

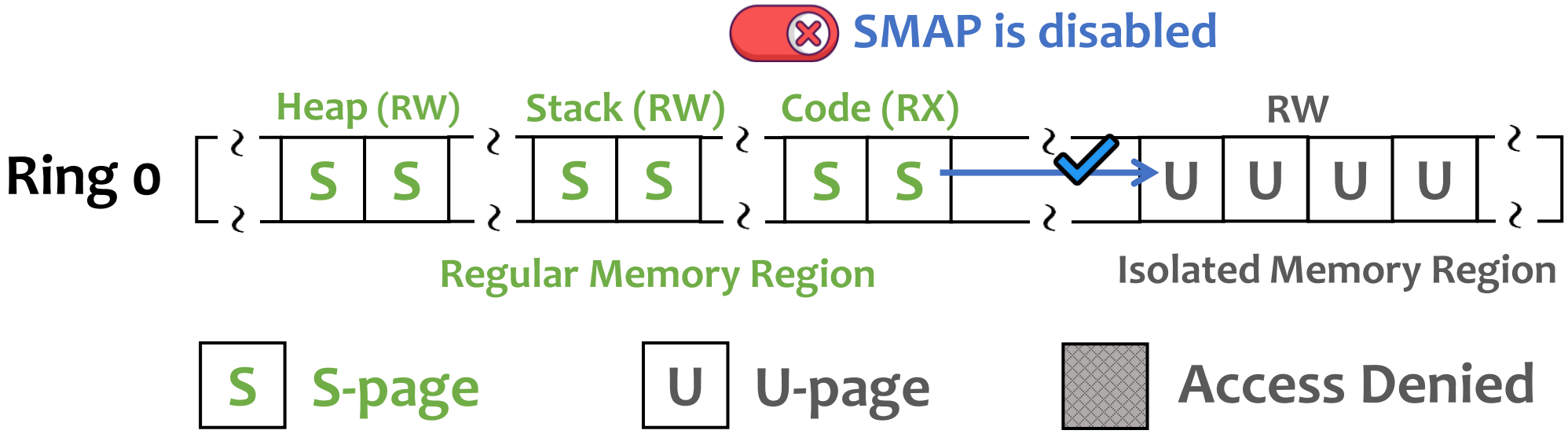
High-level Design —— SEIMI

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



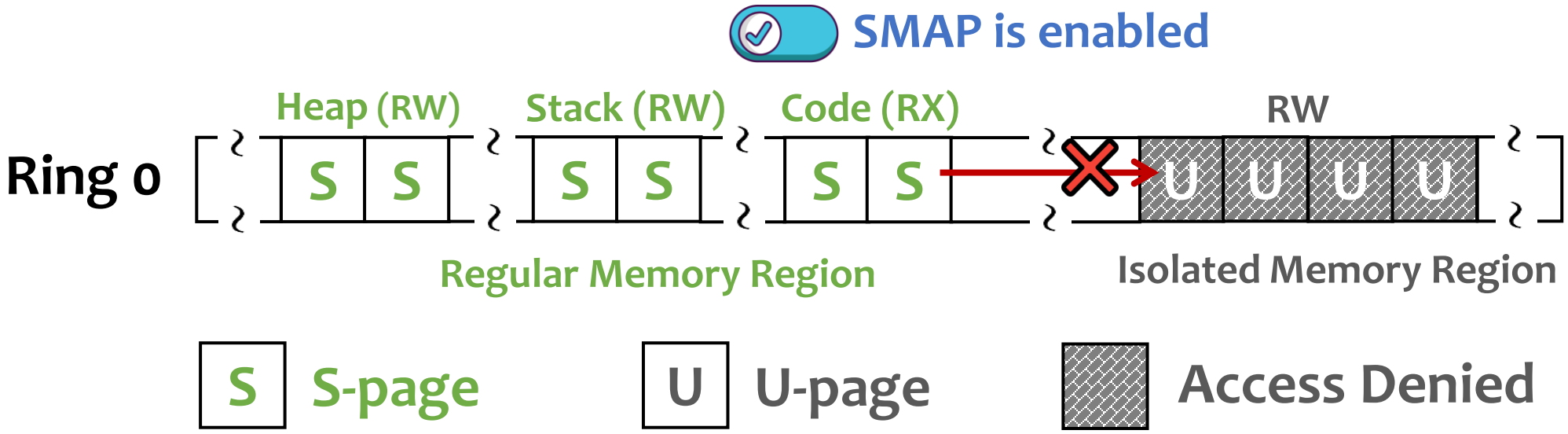
High-level Design —— SEIMI

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



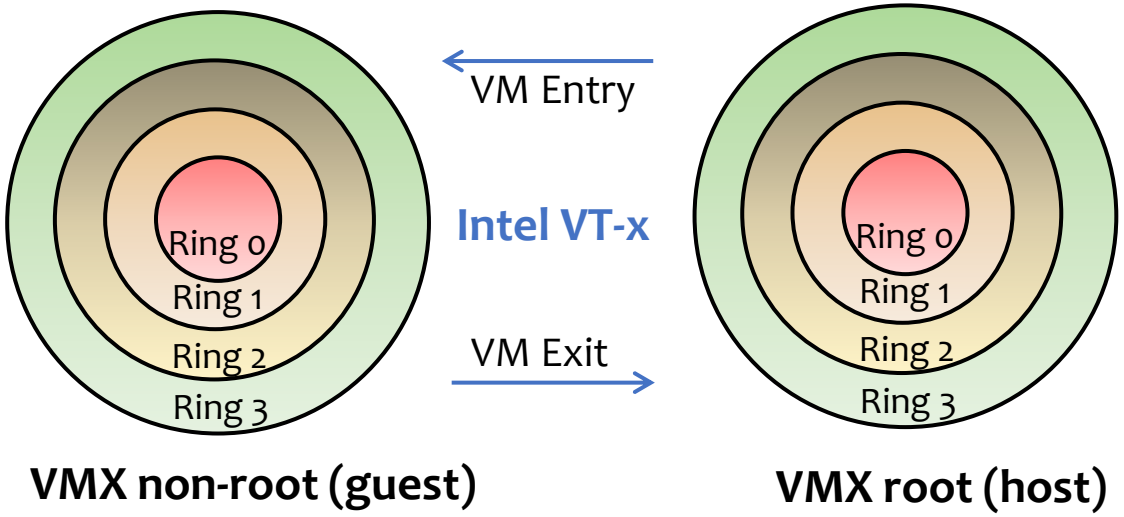
High-level Design —— SEIMI

- **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.

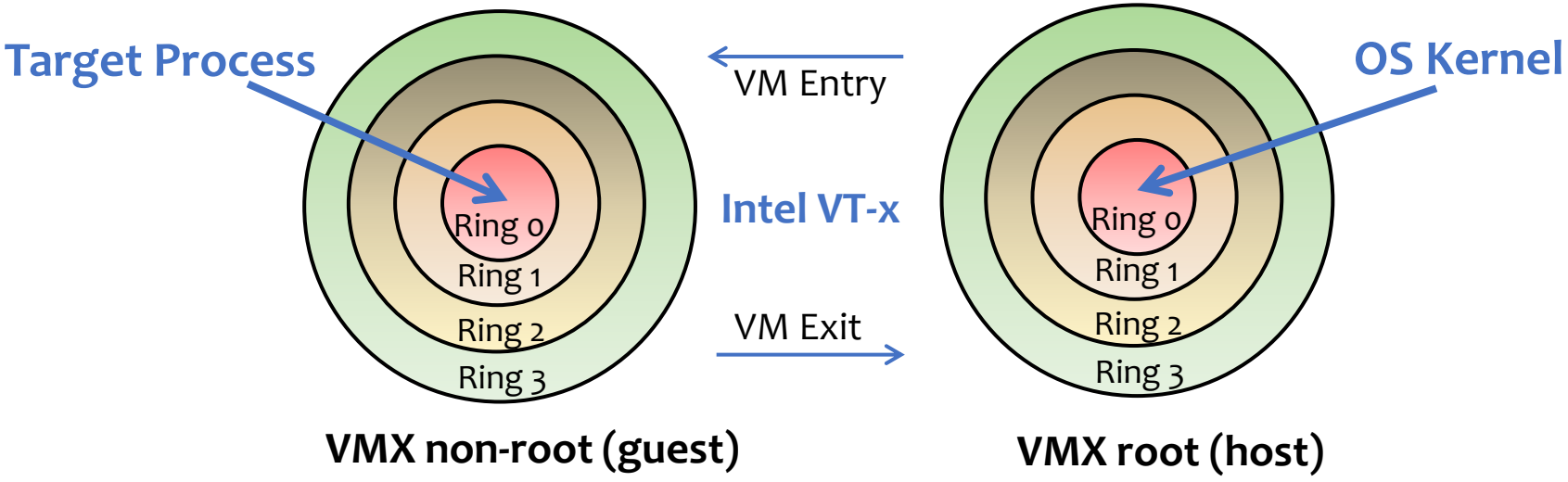
- **Our Solution** —— Placing the OS kernel in "ring -1"
  - Using the Intel VT-x technique to separate the target application and the OS kernel



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.

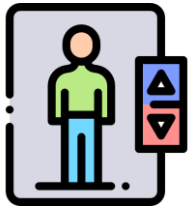
- **C-2: Preventing the leaking/manipulating of the privileged data structures.**
  - In general, a guest VM needs to manage the memory, interrupts, exceptions, etc.
  - Some data structures are **privileged**, e.g., the page tables.

- **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

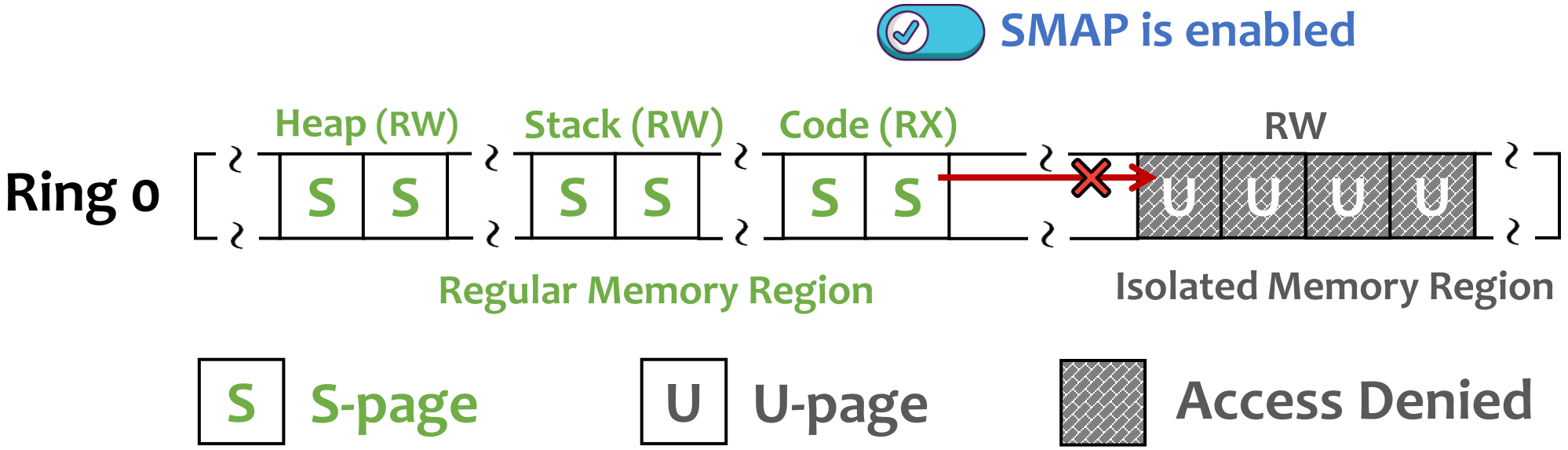
- **C-1: Distinguishing SMAP reads and writes.**

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

Approaches Overview —— Challenge-1

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

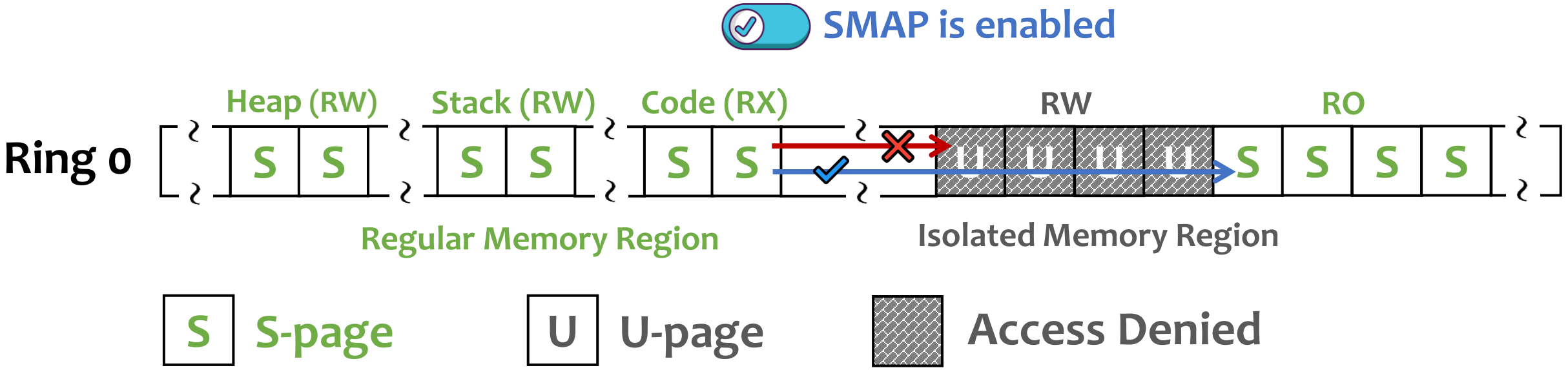
- **Solution** —— The shared-memory based read/write separation method.



Approaches Overview —— Challenge-1

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

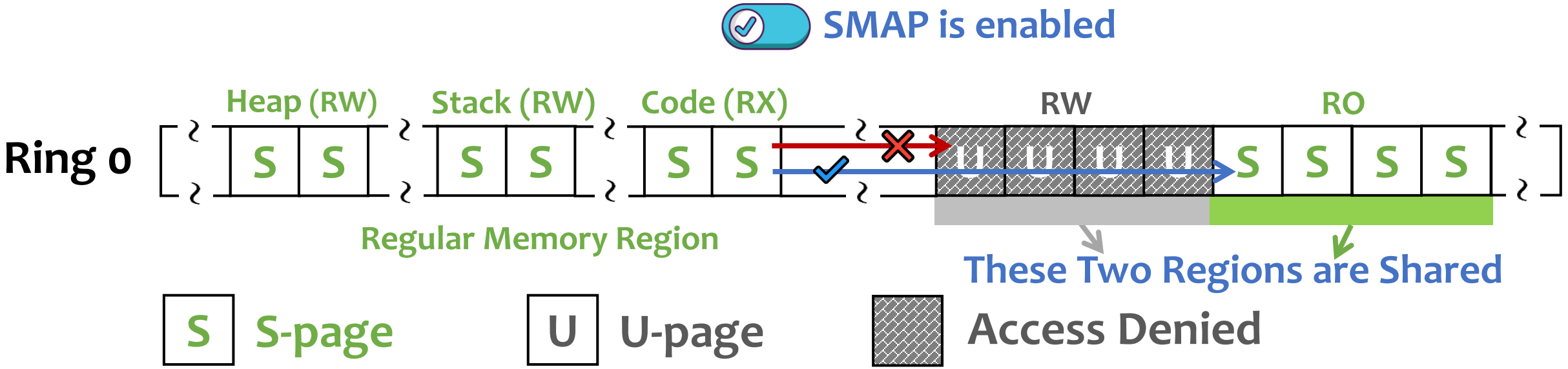
- **Solution** —— The shared-memory based read/write separation method.



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:**

1. We identify **all privileged instructions** in the 64-Bit mode of X86_64.

Approaches Overview —— Challenge-3

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

- **Solution:**

1. We identify **all privileged instructions** in the 64-Bit mode of X86_64.

2. Also, identifying the instructions that will **change the behaviors in different rings**.

Approaches Overview —— Challenge-3

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

- **Solution:**

1. We identify **all privileged instructions** in the 64-Bit mode of X86_64.

2. Also, identifying the instructions that will **change the behaviors in different rings**.

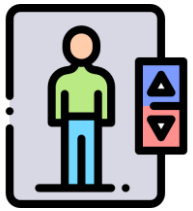
3. SEIMI **sanitizes** the execution of these instructions in the VMX non-root mode by using multiple techniques.

Outline

Motivation

High-level Design

Approach Overview



SEIMI System

Evaluation

System Overview

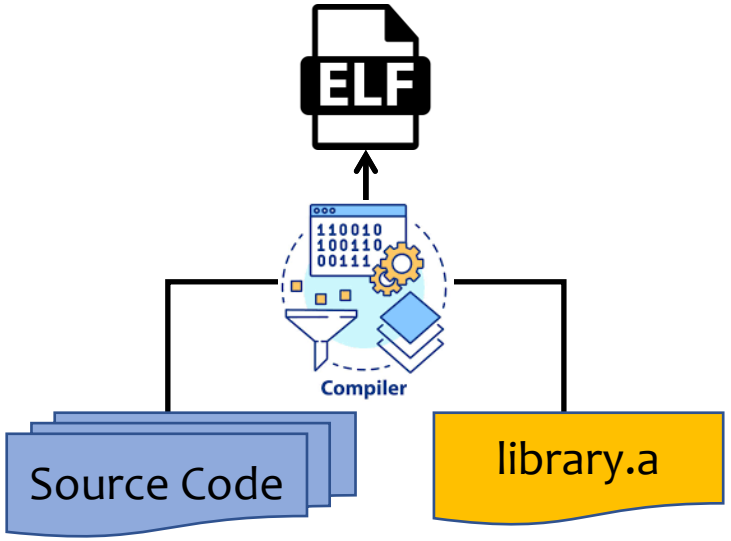
- **SEIMI is implemented on Linux/X86_64 platform.**

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**

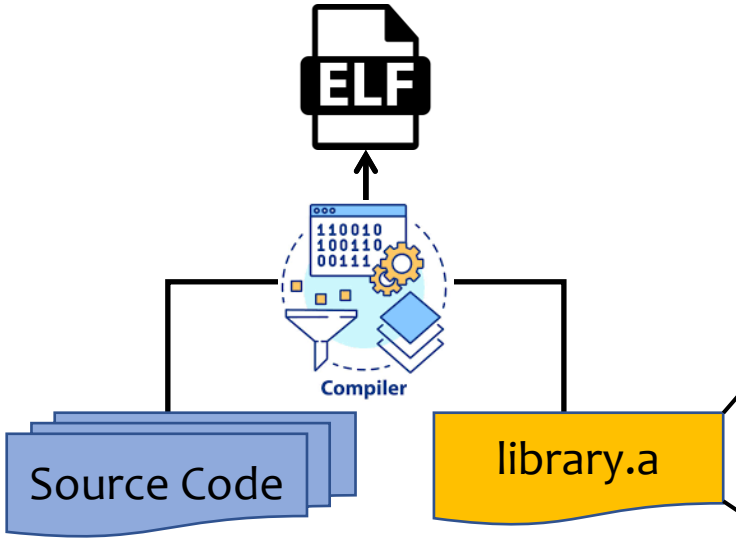


Compilation phase

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

SEIMI —— Compilation Phase

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

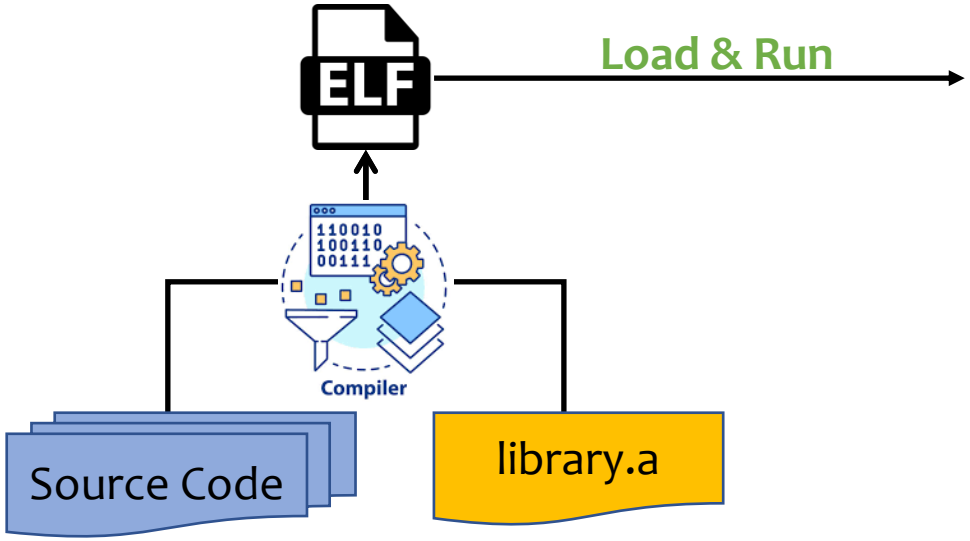


Compilation phase

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.

System Overview

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

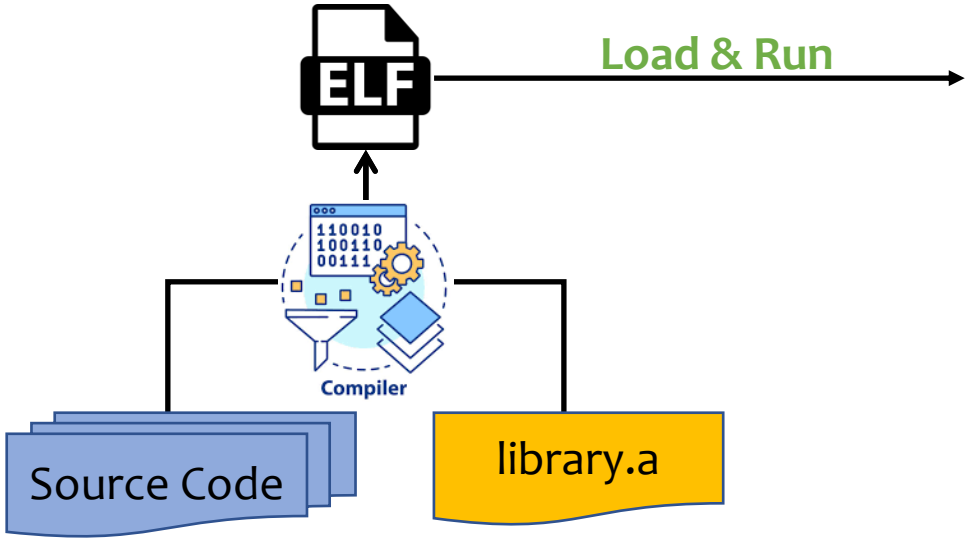


Compilation phase

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

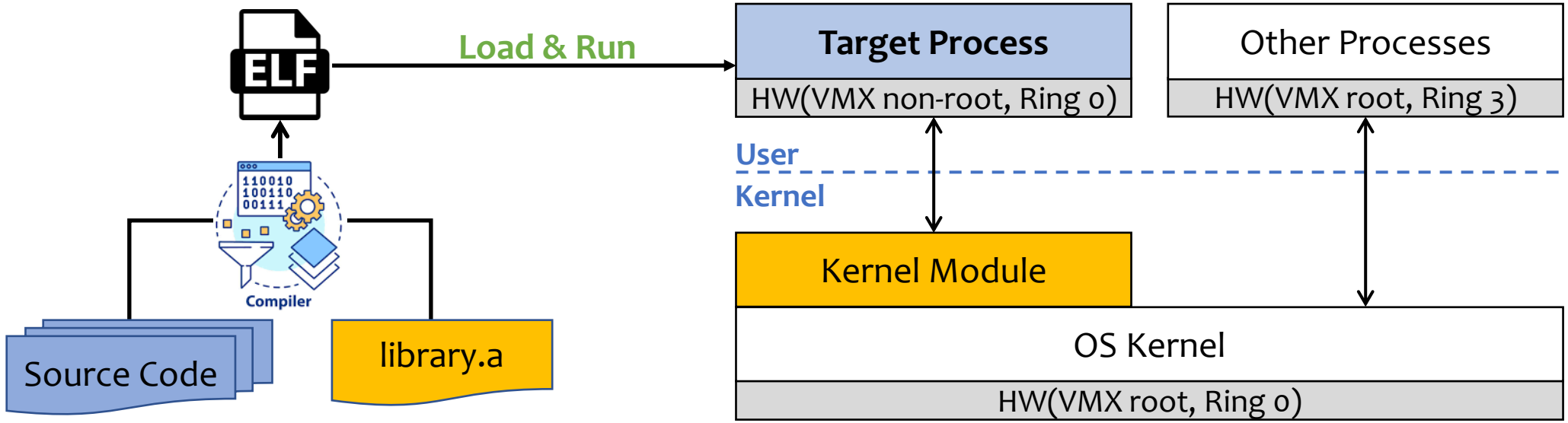
Users could use the SEIMI's APIs to management the isolated memory region.

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

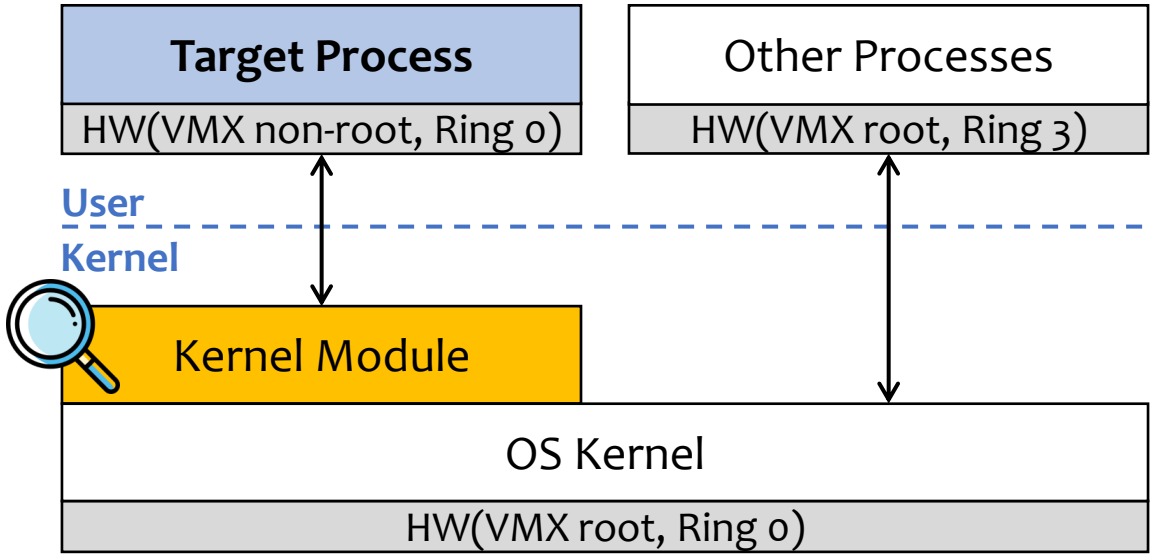
Users could use the SEIMI's APIs to management the isolated memory region.

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 —— Runtime Phase

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



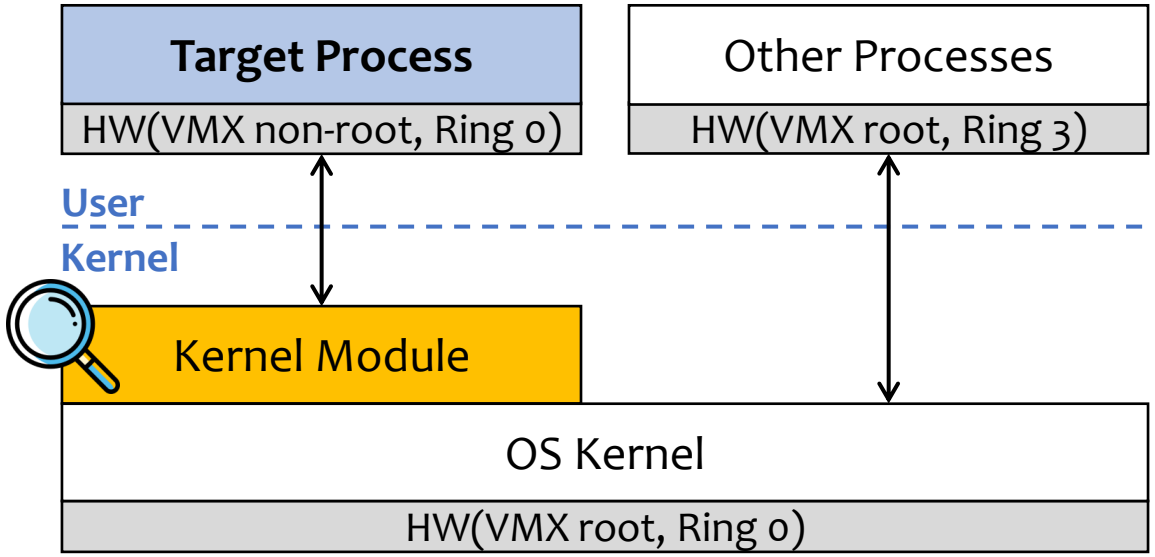
Runtime Phase

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

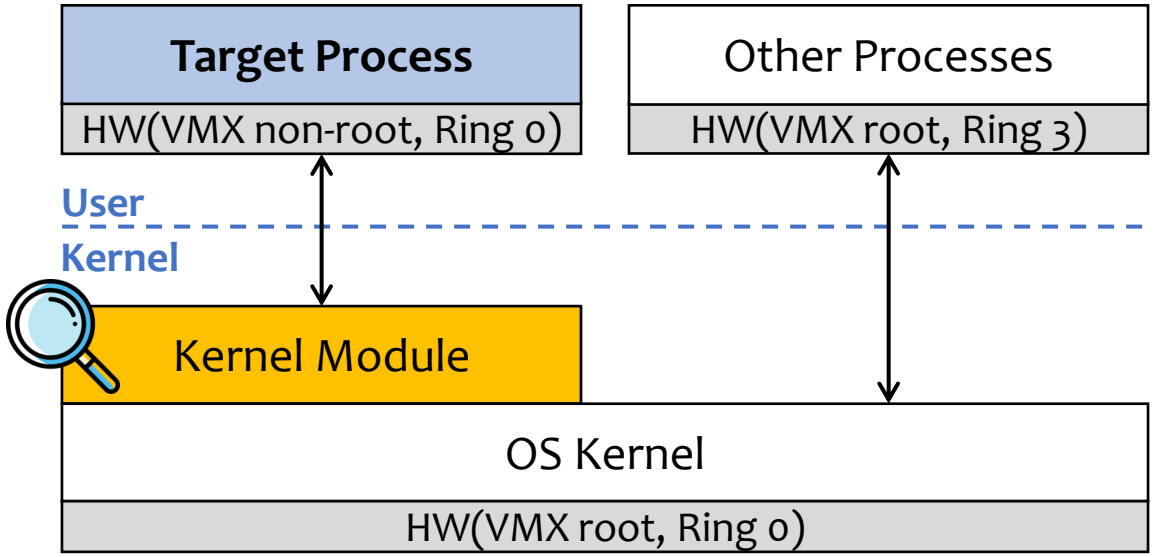
- **The core of SEIMI is a kernel module, includes three key components.**

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**

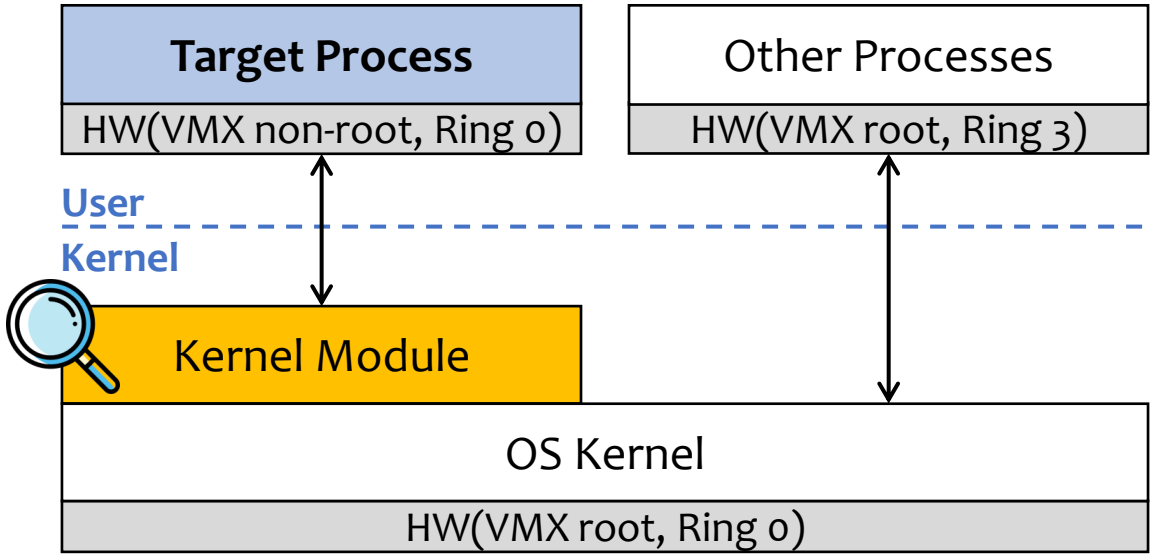
– Configures the regular/isolated memory region.

2 **Privileged Instructions Prevention Component**

– Prevents these instructions from being abused.

3 **Events Redirection Component**

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



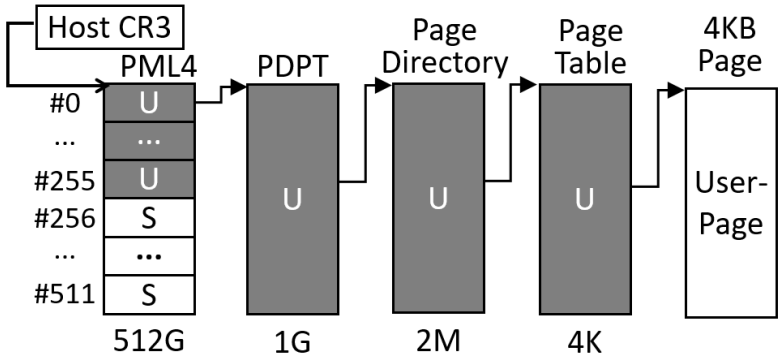
Runtime Phase

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.

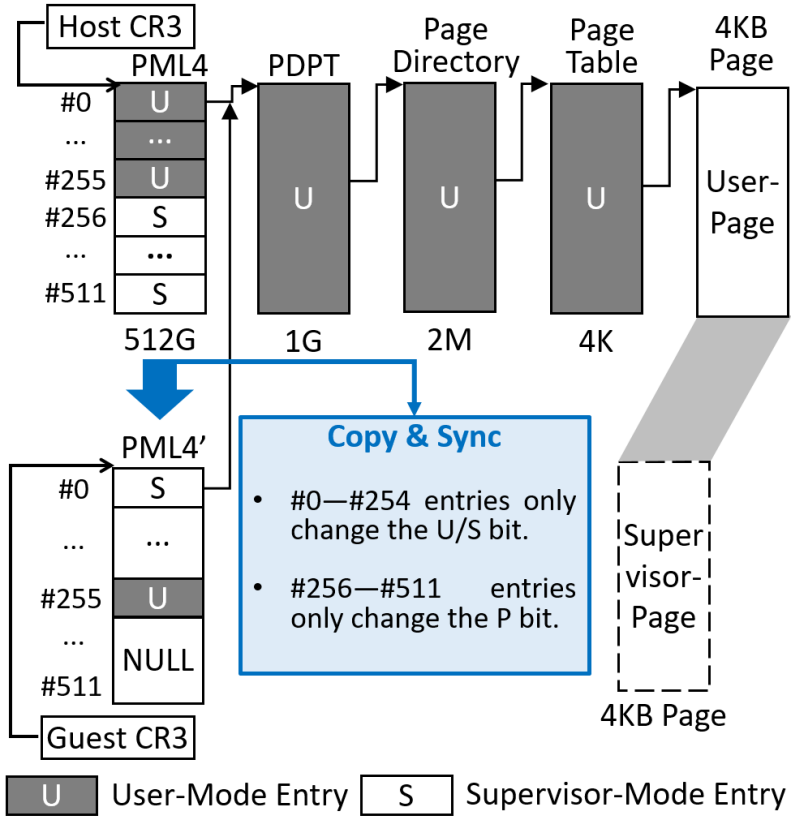
SEIMI —— Memory Management Component

- **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.



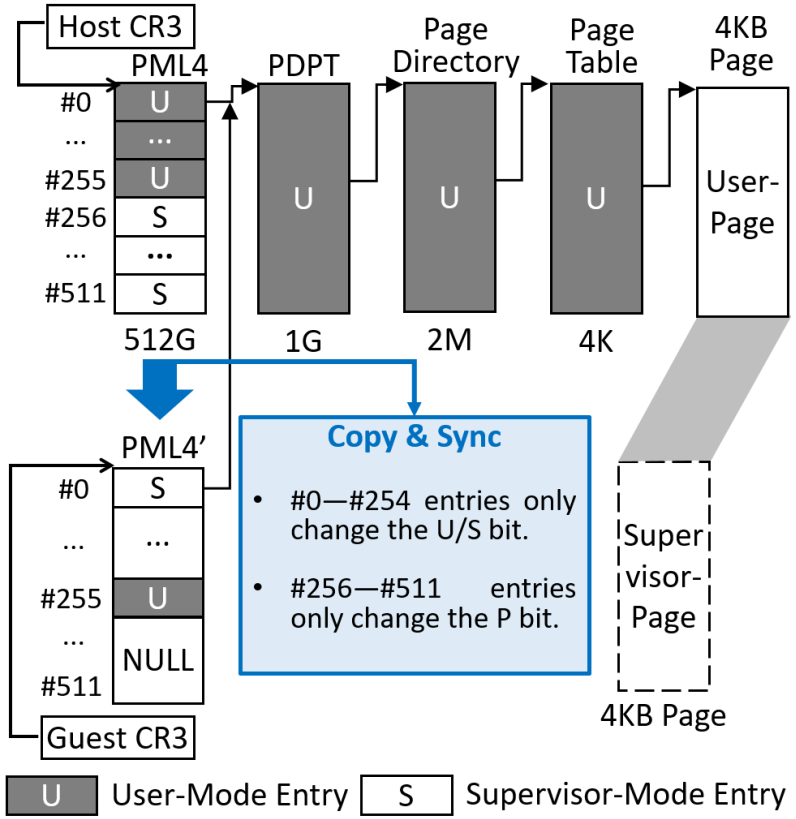
SEIMI —— Memory Management Component

- **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

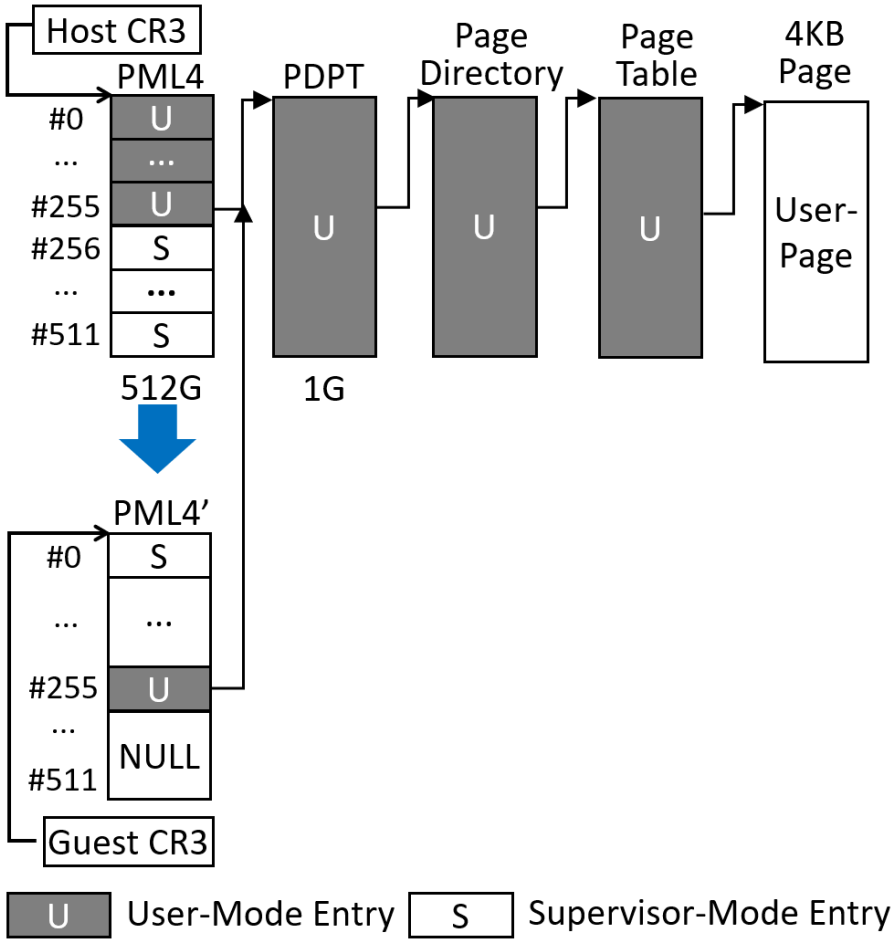
- **A shadow mechanism for (only) page-table root.**
  - The guest/host page-tables share the last three-level page table entries.
  - Flipping the U/S bit to set the U-page and S-page neatly.



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

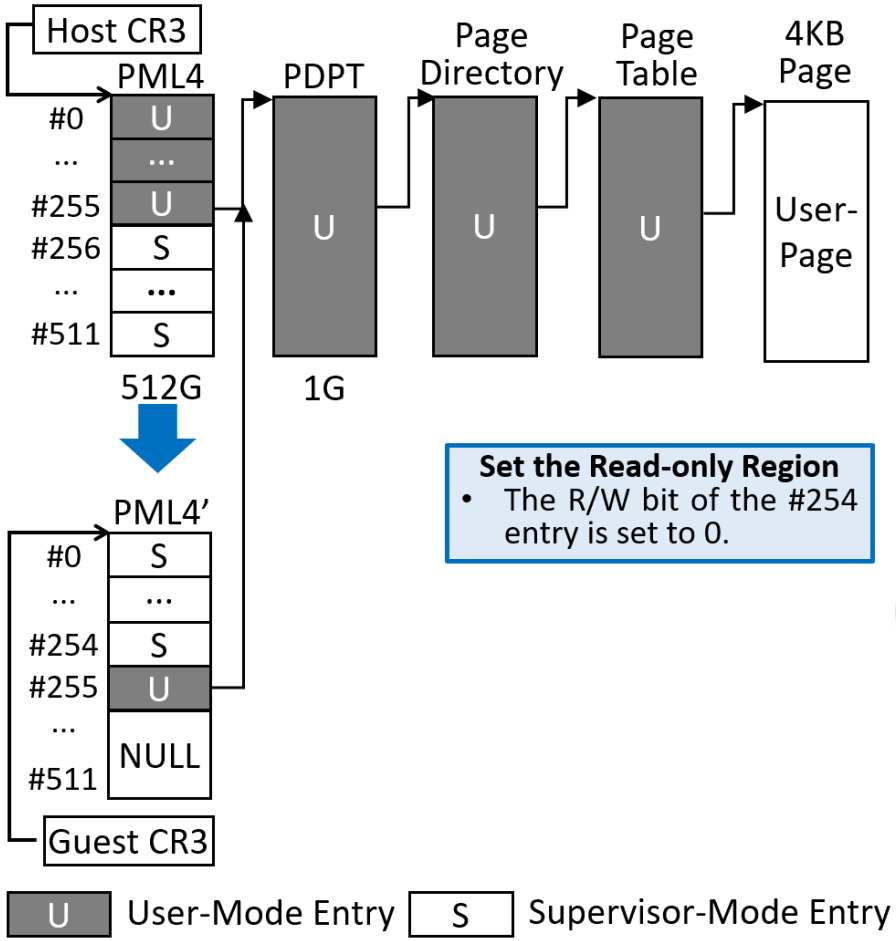
SEIMI —— Memory Management Component

- **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

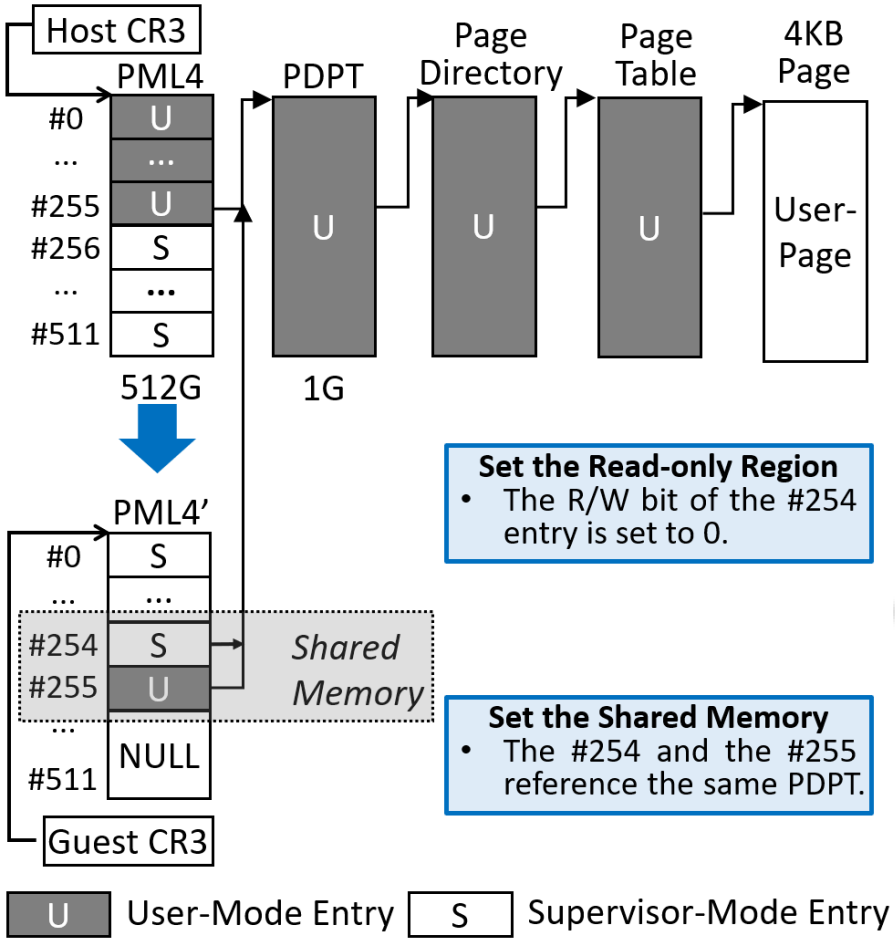
- **Support the read-only isolated S-page memory region.**
  - Flipping the R/W bit to set the read-only permission neatly.



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

1

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 —— 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.

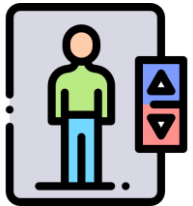
Outline

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%

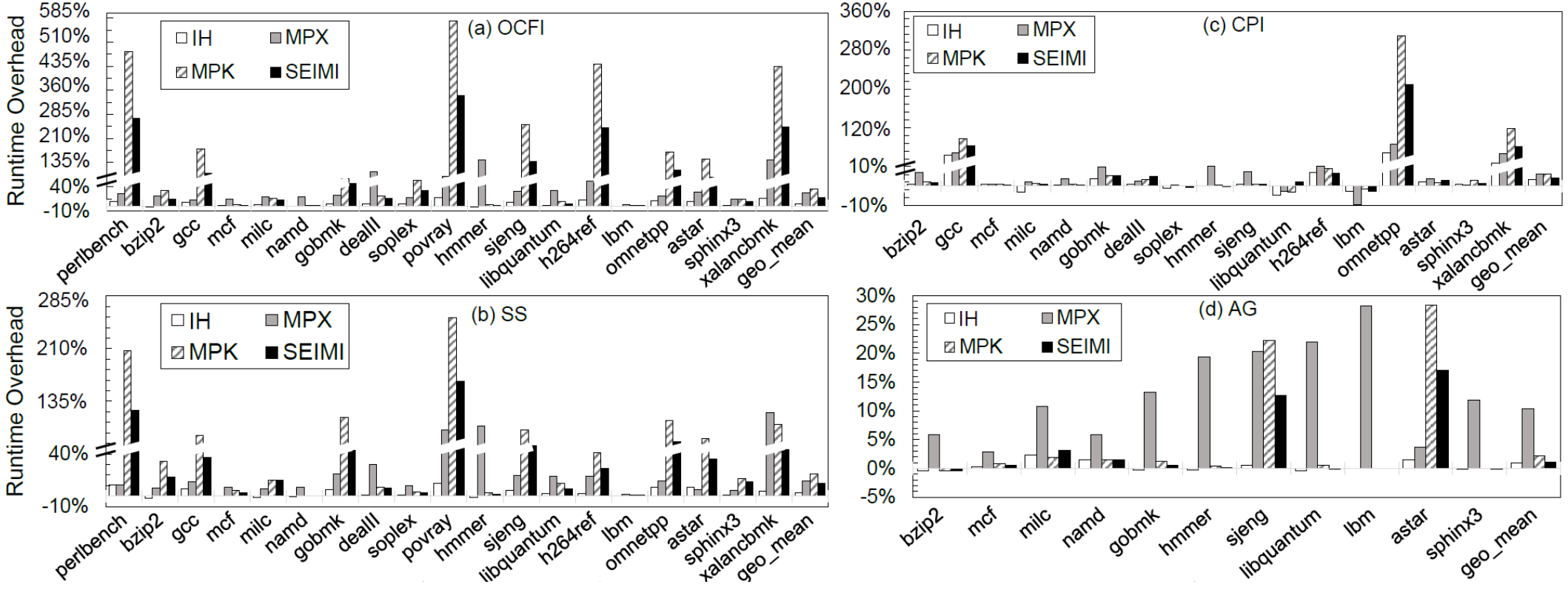
3 File & VM system latency (in µs): smaller is better.

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.

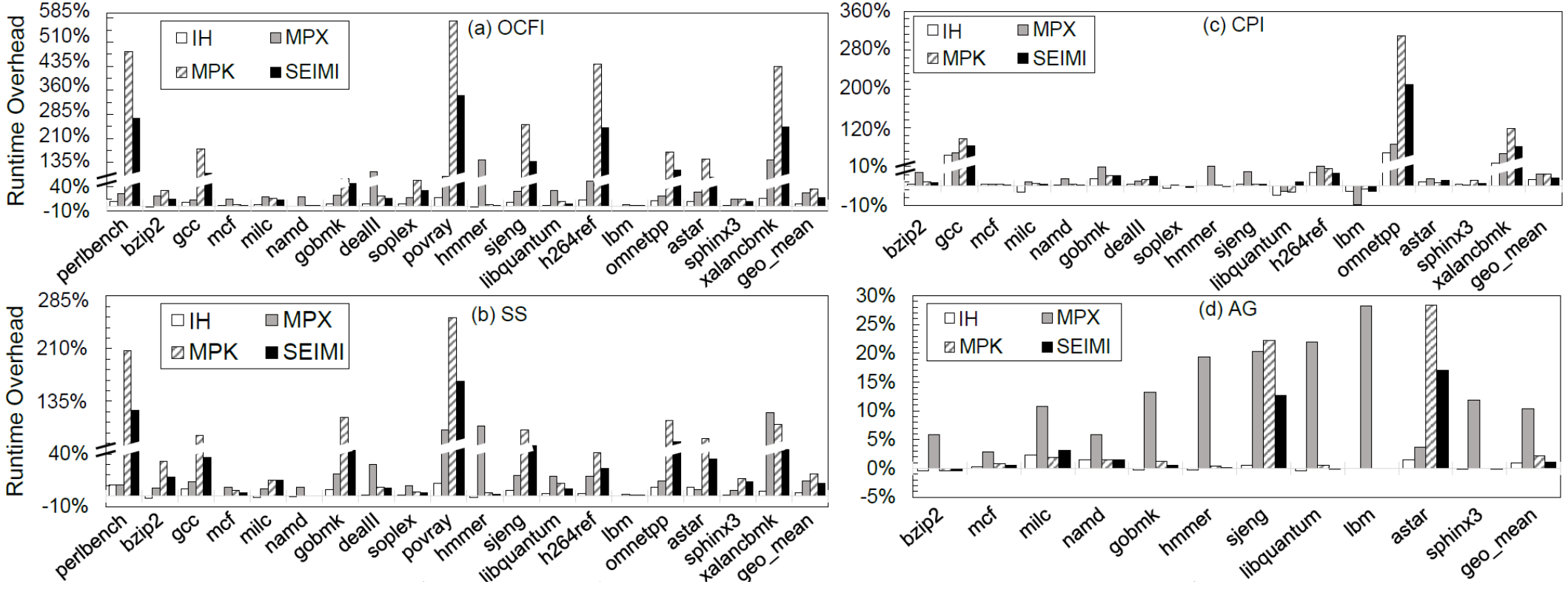
Macrobenchmark —— SPEC CPU 2006 benchmark

- 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%**).



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

Two Weaknesses but Already Solved

- **For I/O-intensive applications, SEIMI may be a double-edged sword:**
  - **The performance benefit on the isolation may be counteracted or even far less than the cost of the handling of system calls ——VM Exit is six times slower than SYSCALL.**

Two Weaknesses but Already Solved

- **For I/O-intensive applications, SEIMI may be a double-edged sword:**
  - **The performance benefit on the isolation may be counteracted or even far less than the cost of the handling of system calls ——VM Exit is six times slower than SYSCALL.**

- **SEIMI must be coupled with defenses that restricts its scenarios.**
  - **Since X86-64 ISA has variable length instructions, code alignment is critical: unintended instruction can be executed when alignment is broken.**
  - **Defenses can help to prevent the unintended instructions POPF and STAC.**

Two Weaknesses but Already Solved

- **For I/O-intensive applications, SEIMI may be a double-edged sword:**
  - **The performance benefit on the isolation may be counteracted or even far less than the cost of the handling of system calls ——VM Exit is six times slower than SYSCALL.**

- **SEIMI must be coupled with defenses that restricts its scenarios.**
  - **Since X86-64 ISA has variable length instructions, code alignment is critical: unintended instruction can be executed when alignment is broken.**
  - **Defenses can help to prevent the unintended instructions POPF and STAC.**
  - **But the binary rewriting technique is difficult to eliminate them with low runtime overhead due to the POPF is only 1-Byte.**