

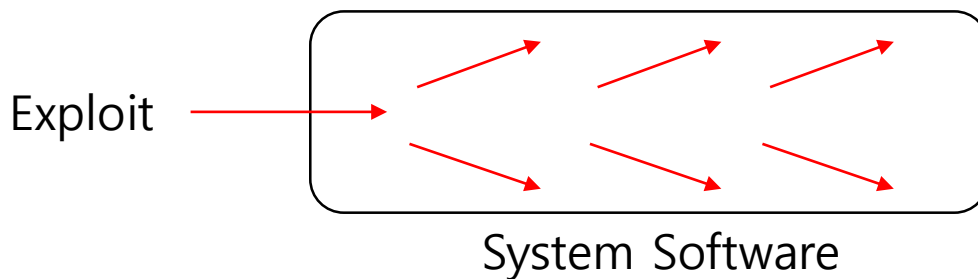
Dynamic Virtual Address Range Adjustment for Intra-Level Privilege Separation on ARM

Yeongpil Cho, Donghyun Kwon, Hayoon Yi, Yunheung Paek

Seoul National University

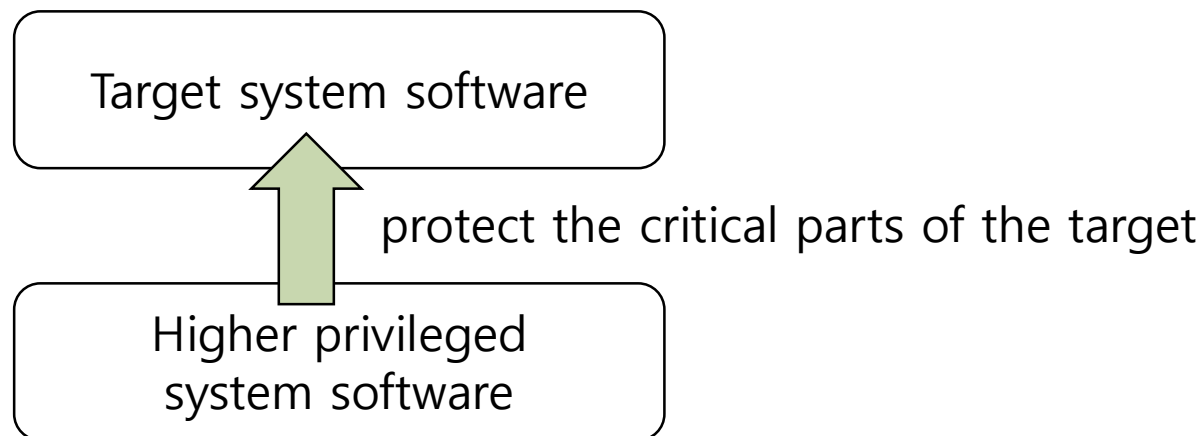
System Software

- For example,
 - Operating System
 - Hypervisor
- System Software plays roles of
 - Resource Manager
 - Trusted Computing Base
- A variety of system software has the monolithic design
 - so, the entire can suffer from small exploits.



Privilege Separation

- One of fundamental security principles
 - Protect security critical parts by separating from the others
 - ex) key management, page table management, system monitoring, ...
- How to enforce this principle to system software?
 - Relying on higher privileged entity

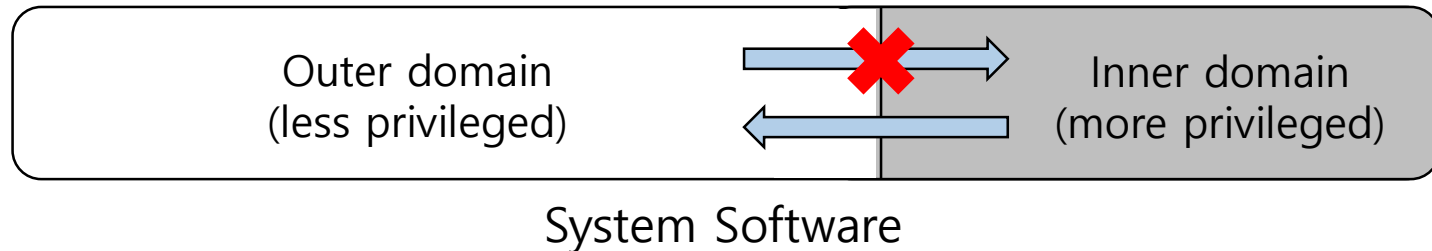


One fundamental question

“ how to enforce this principle to the higher privileged system software? ”

Intra-Level Privilege Separation

- ◉ Divide the monolithic body of system software into the outer domain and inner domain
 - Two domains run at the physically same but logically different privilege level
 - Two domains have asymmetric memory view



- ◉ Two core mechanisms of intra-level privilege separation
 - **intra-level isolation mechanism**
 - prevent the outer domain from accessing the inner domain
 - **domain switching mechanism**
 - transfer control between the outer and inner domains

Motivation of our work

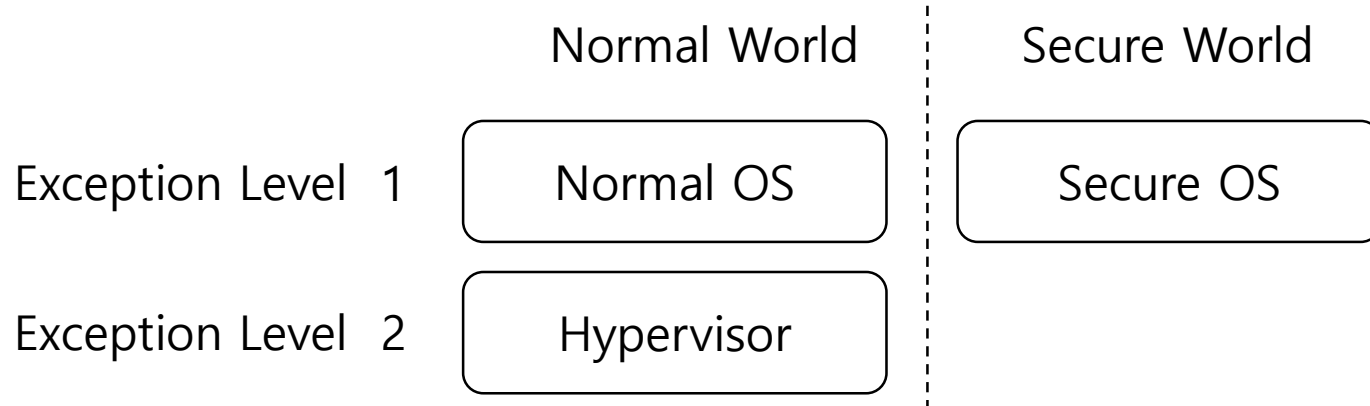
- To efficiently and securely implement two core mechanisms, a hardware feature for memory protection is used
- The type of system software that can be supported is determined by the used hardware features.

Solution	Architecture	Key Hardware Feature	Target System Software
HyperSafe [S&P'10]	x86 64-bit	Write-Protection	Hypervisor, Normal OS*
Nested Kernel [ASPLOS'15]	x86 64-bit	Write-Protection	Normal OS, Hypervisor*
SKEE [NDSS'16]	ARM 32-bit	TTBCR	Normal OS, Secure OS*
SKEE [NDSS'16]	ARM 64-bit	Extended Paging	Normal OS

* : not mentioned in the paper, but can be supported with the same technique

Motivation of our work

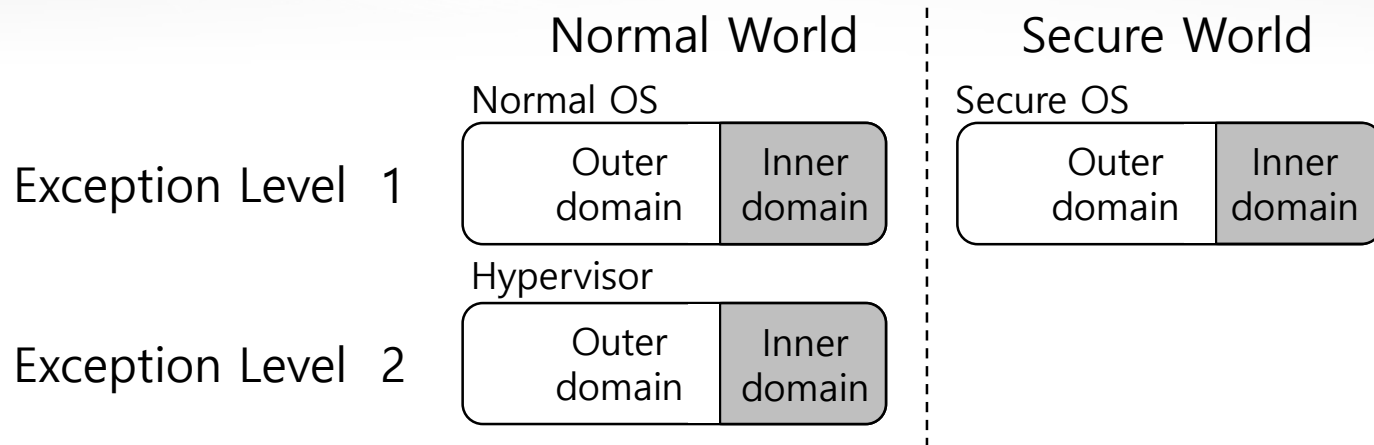
- ARM's 64-bit architecture (a.k.a AArch64) is widely used in mobile devices
- Various types of system software coexist to enrich the functionality of the AArch64-based devices
 - Exception Level has the same meaning as Privilege Level



- Not only normal OS but also secure OS and hypervisor suffer from exploits

Hilps

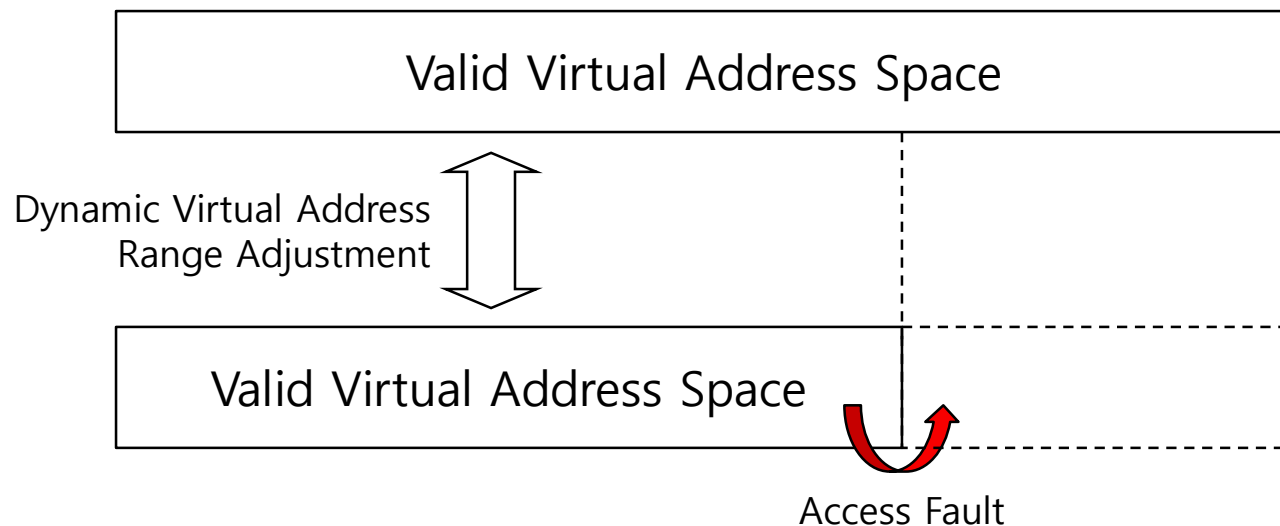
- A technique that can enforce intra-level privilege separation to a variety of system software on AArch64



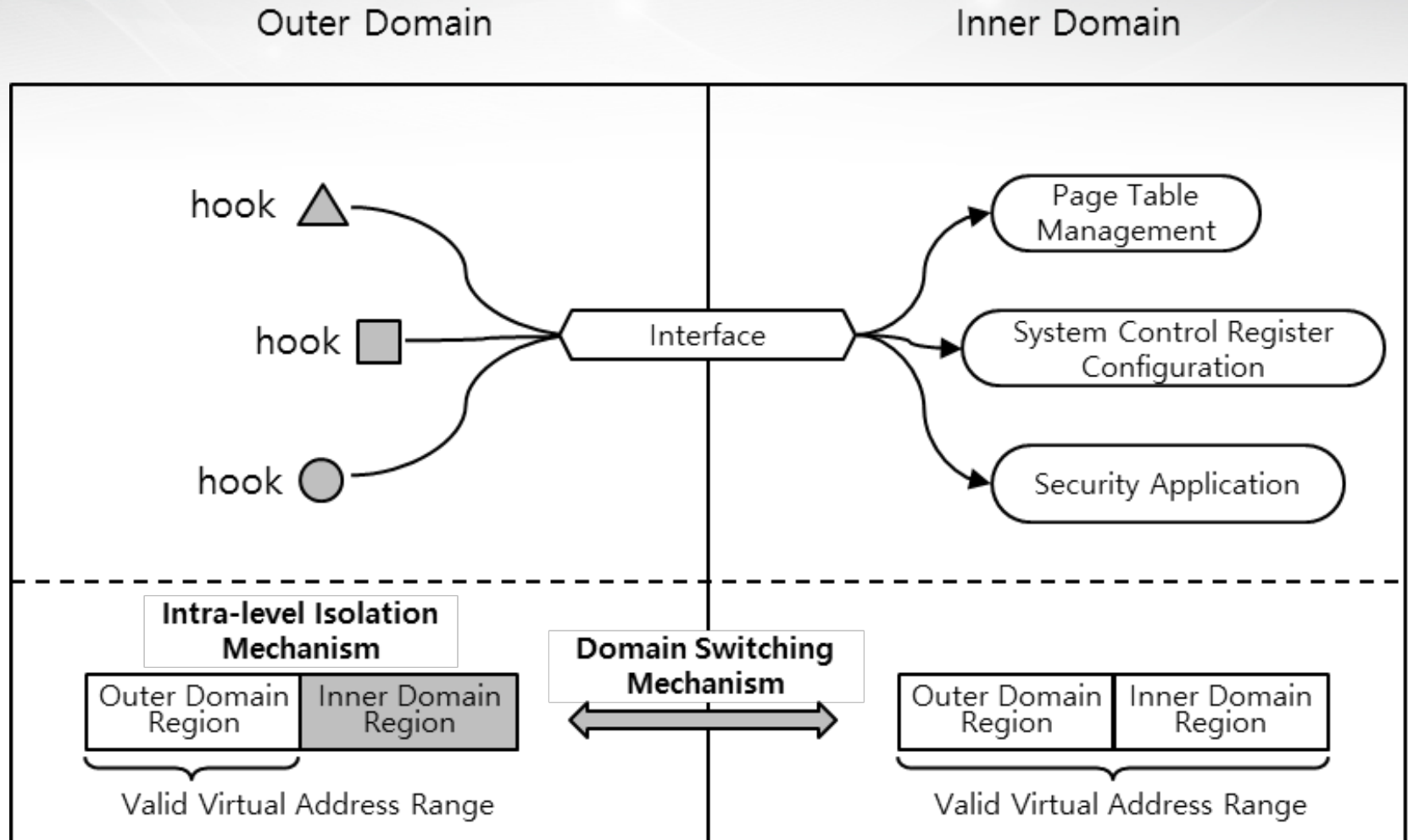
- To achieve the goal of Hilps, two core mechanisms for intra-level isolation and domain switching must be applicable regardless of exception level

Hilps

- AArch64 contains a hardware feature that allows dynamically adjusting valid virtual address range at each exception level
 - Hilps can create a special memory region that is temporally hidden from other memory regions

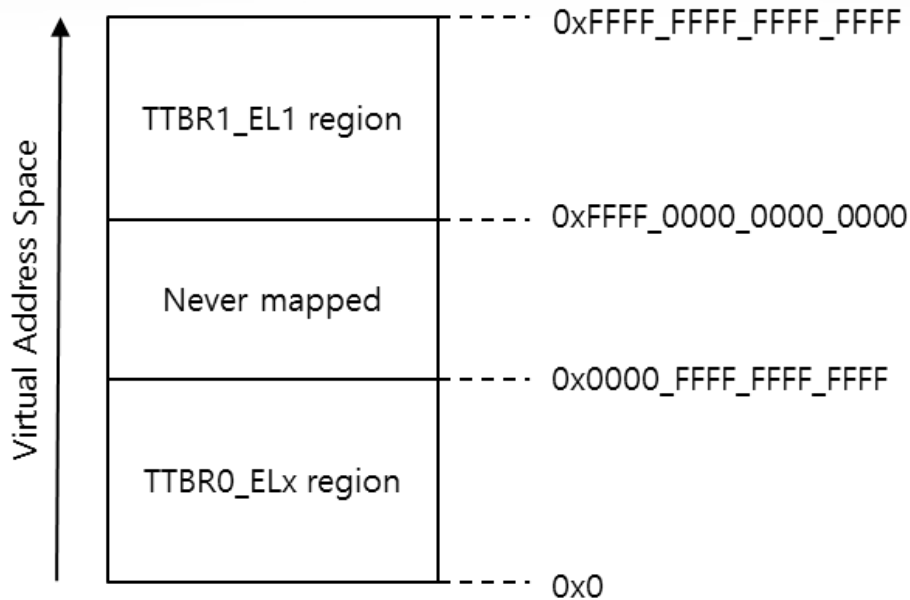


Hilps



Background: Address Translation on AArch64

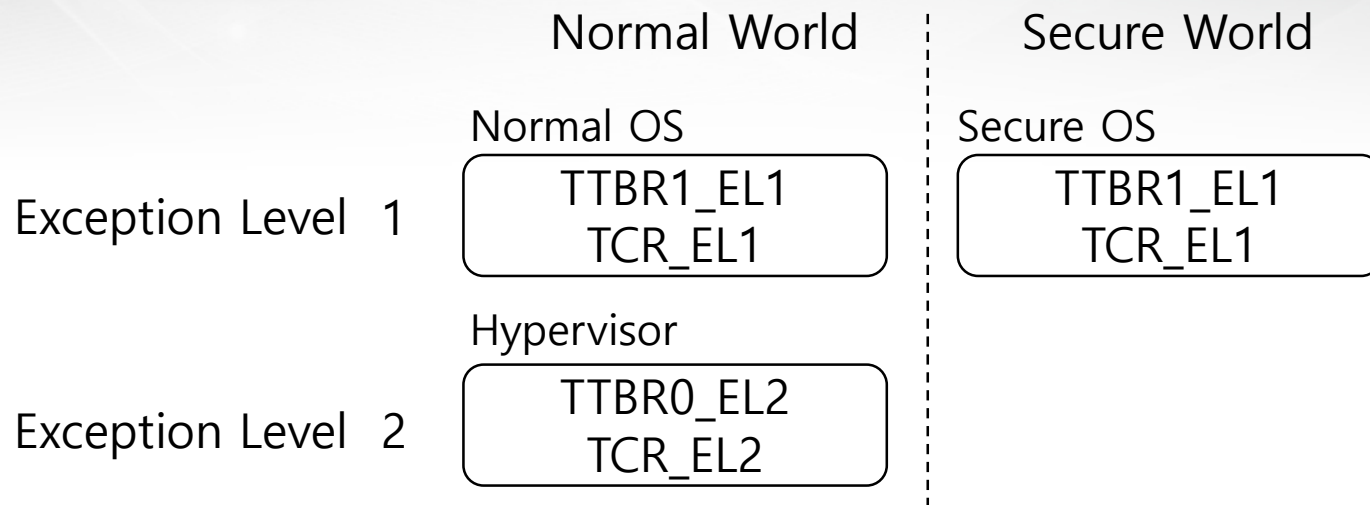
- System control registers for address translation
 - TTBRx_ELx (Translation Table Base Register)
 - points to the base address of the current page table



- TCR_ELx (Translation Control Register)
 - controls address translation

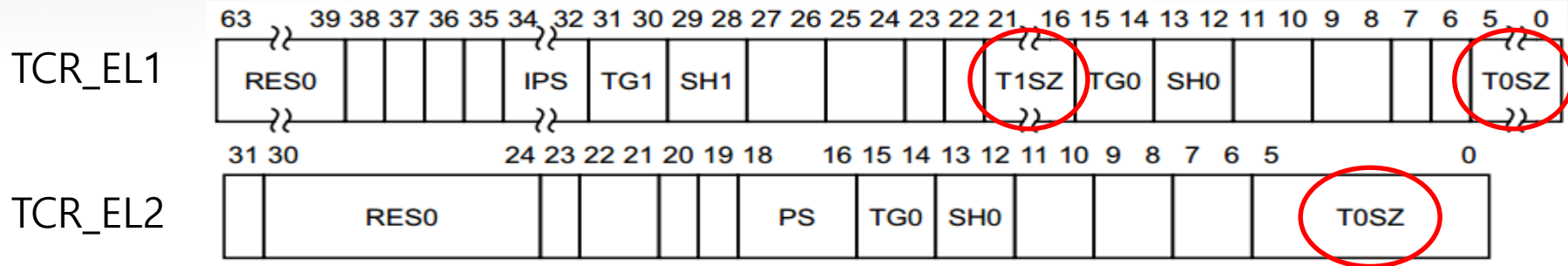
Background: Address Translation on AArch64

- Each exception level has its own control registers



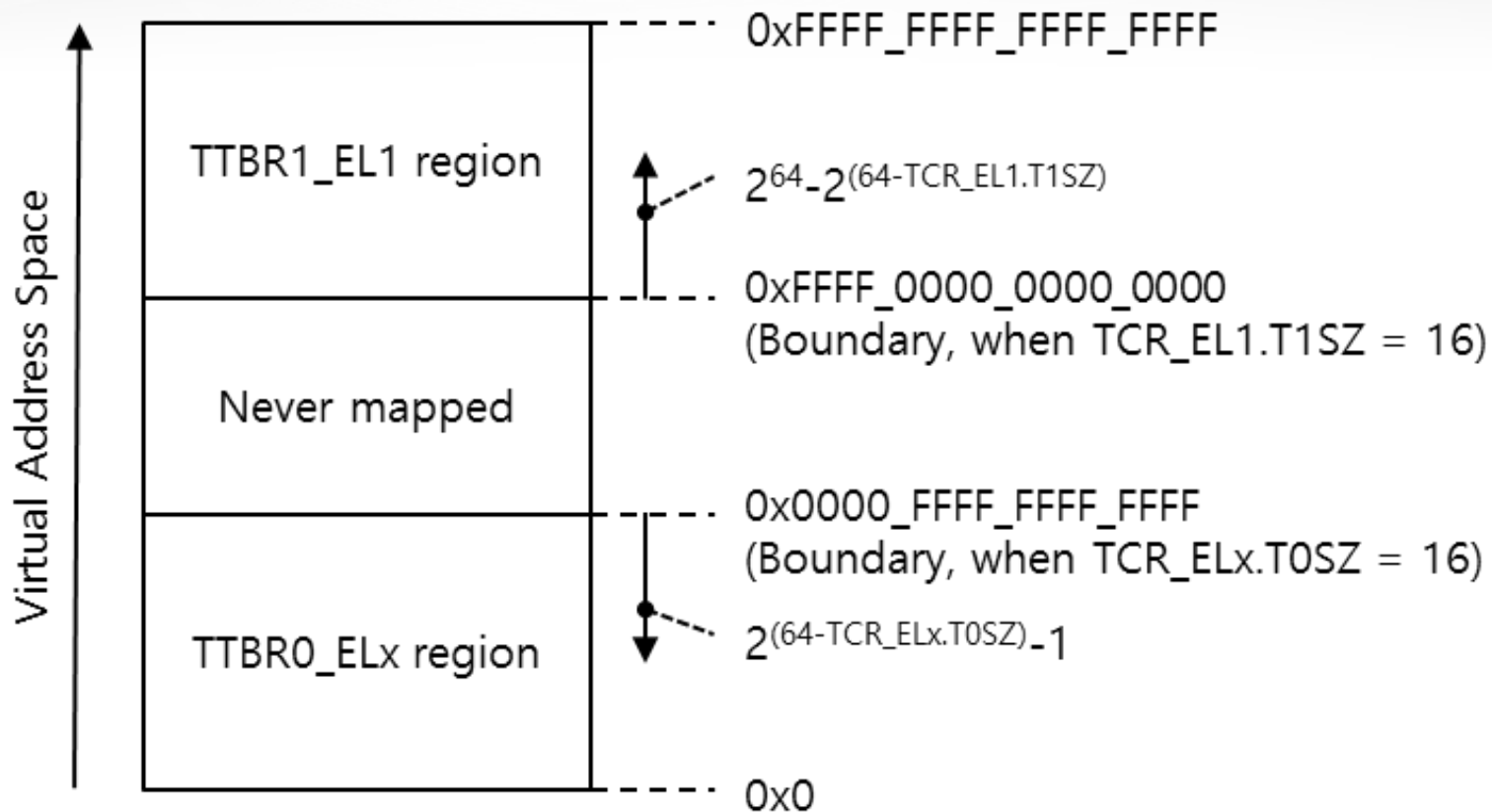
Background: Virtual Address Range Adjustment

- TxSZ-field of TCR_ELx determines the valid range of the virtual address space translated by the paired TTBRx_ELx



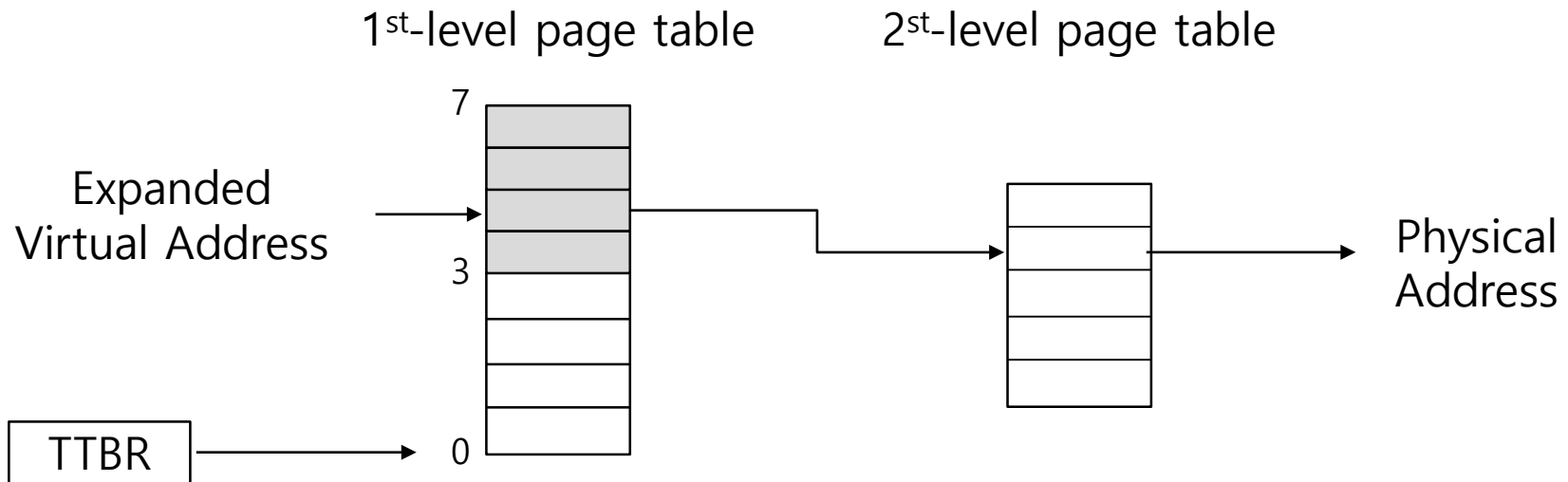
Background: Virtual Address Range Adjustment

- Change of valid virtual address range depending on TxSZ-field



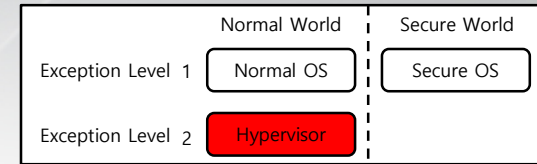
Background: Virtual Address Range Adjustment

- Typically, multi-level page tables are used for effective management
 - 1st-level page table is directly referenced by virtual address
- When valid virtual address range changes, the number of valid 1st-level page table entries also varies proportionally

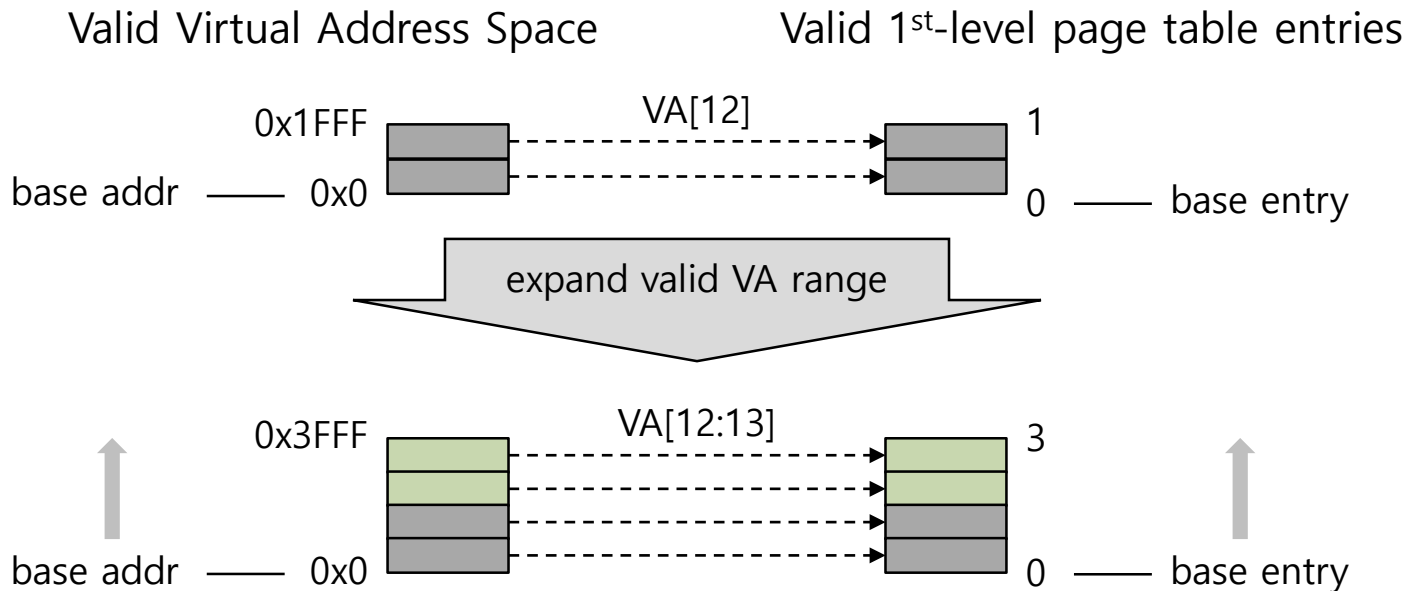


Intra-level isolation mechanism

- System software that runs with TTBR0_ELx
 - ex) Hypervisor

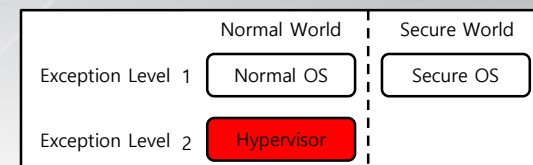


- Valid virtual address space and valid 1st-level page table entries change in the same direction

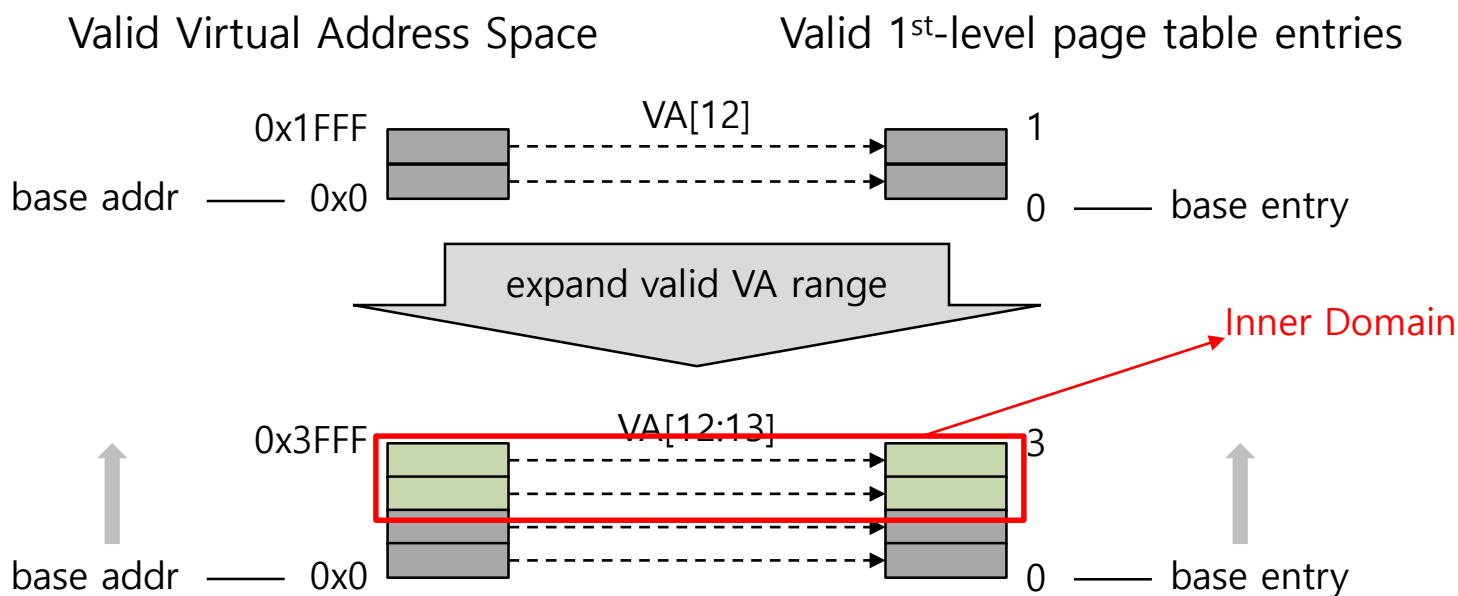


Intra-level isolation mechanism

- System software that runs with TTBR0_ELx
 - ex) Hypervisor

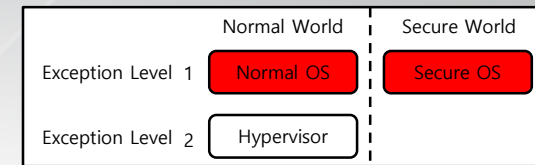


- Valid virtual address space and valid 1st-level page table entries change in the same direction

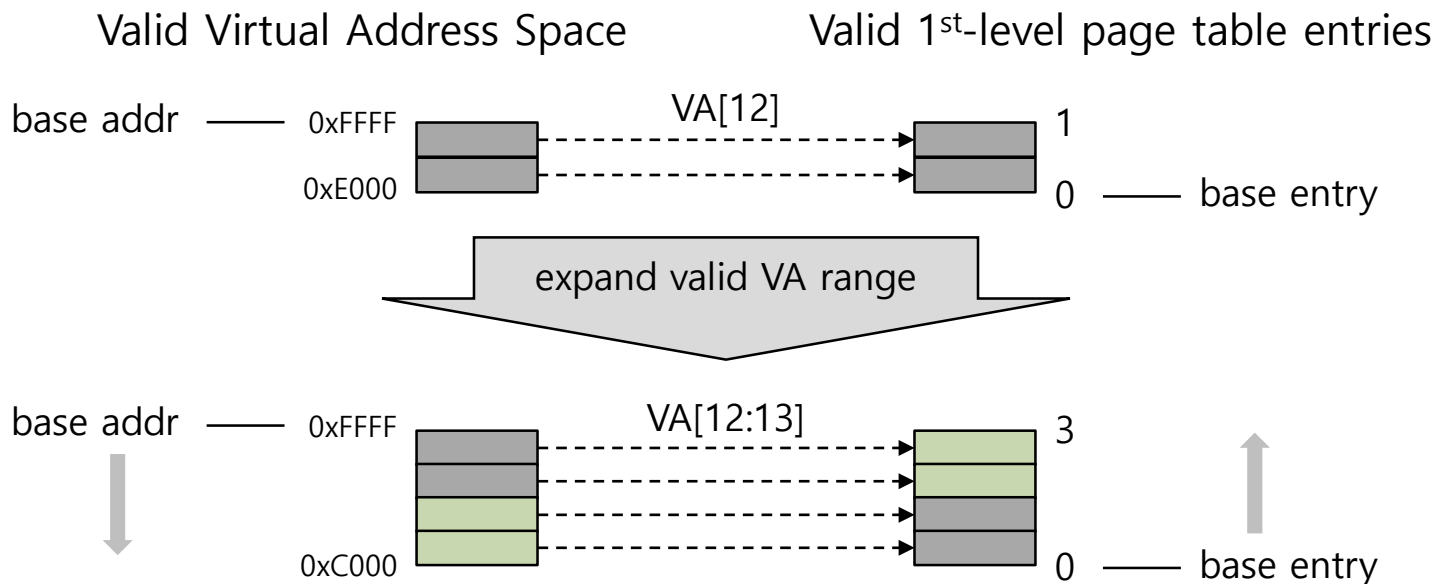


Intra-level isolation mechanism

- System software that runs with TTBR1_EL1
 - ex) Normal OS and secure OS

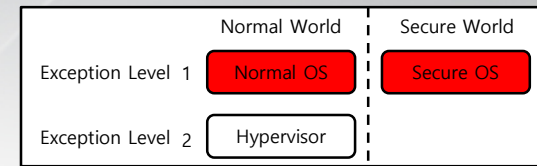


- Valid virtual address space and valid 1st-level page table entries change in the opposite direction

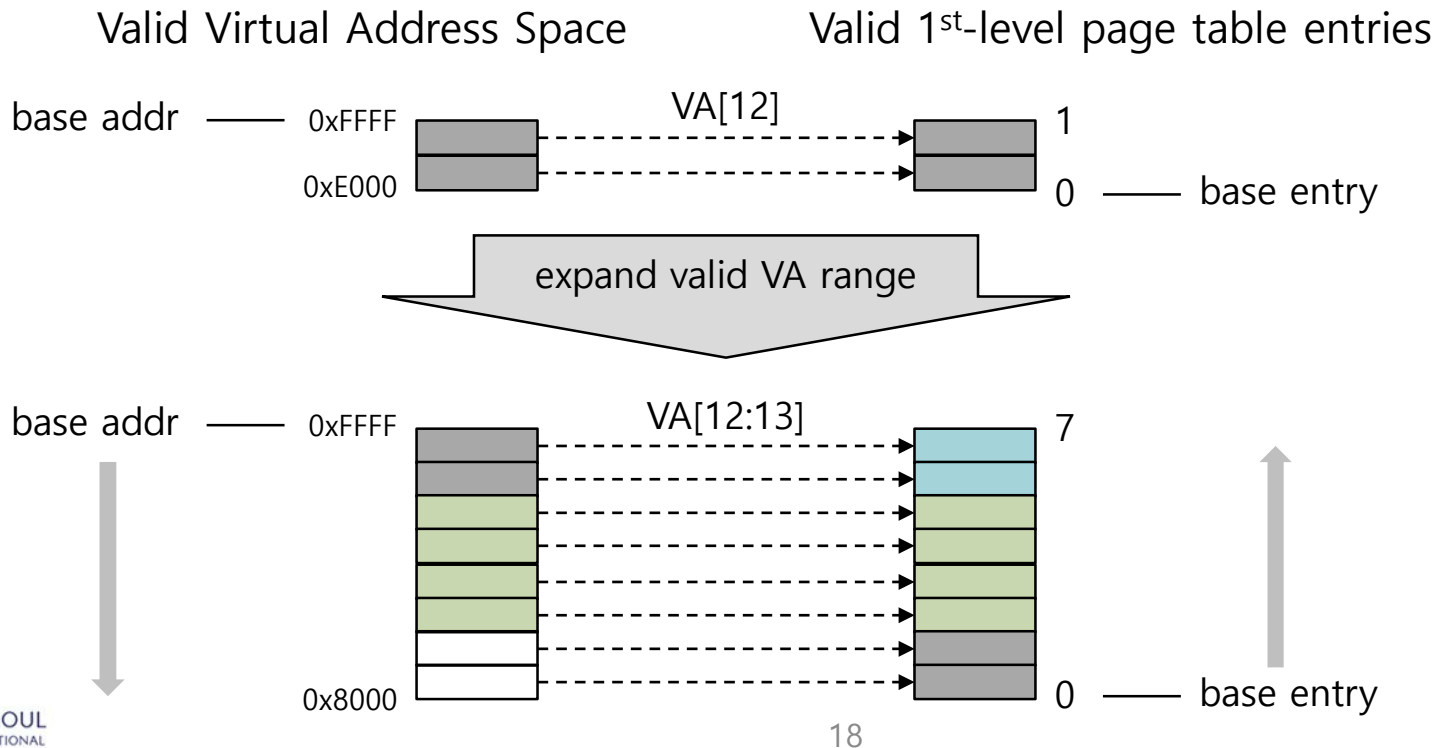


Intra-level isolation mechanism

- System software that runs with TTBR1_EL1
 - ex) Normal OS and secure OS

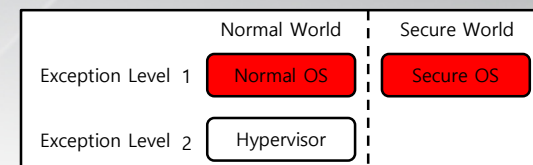


- Valid virtual address space and valid 1st-level page table entries change in the opposite direction

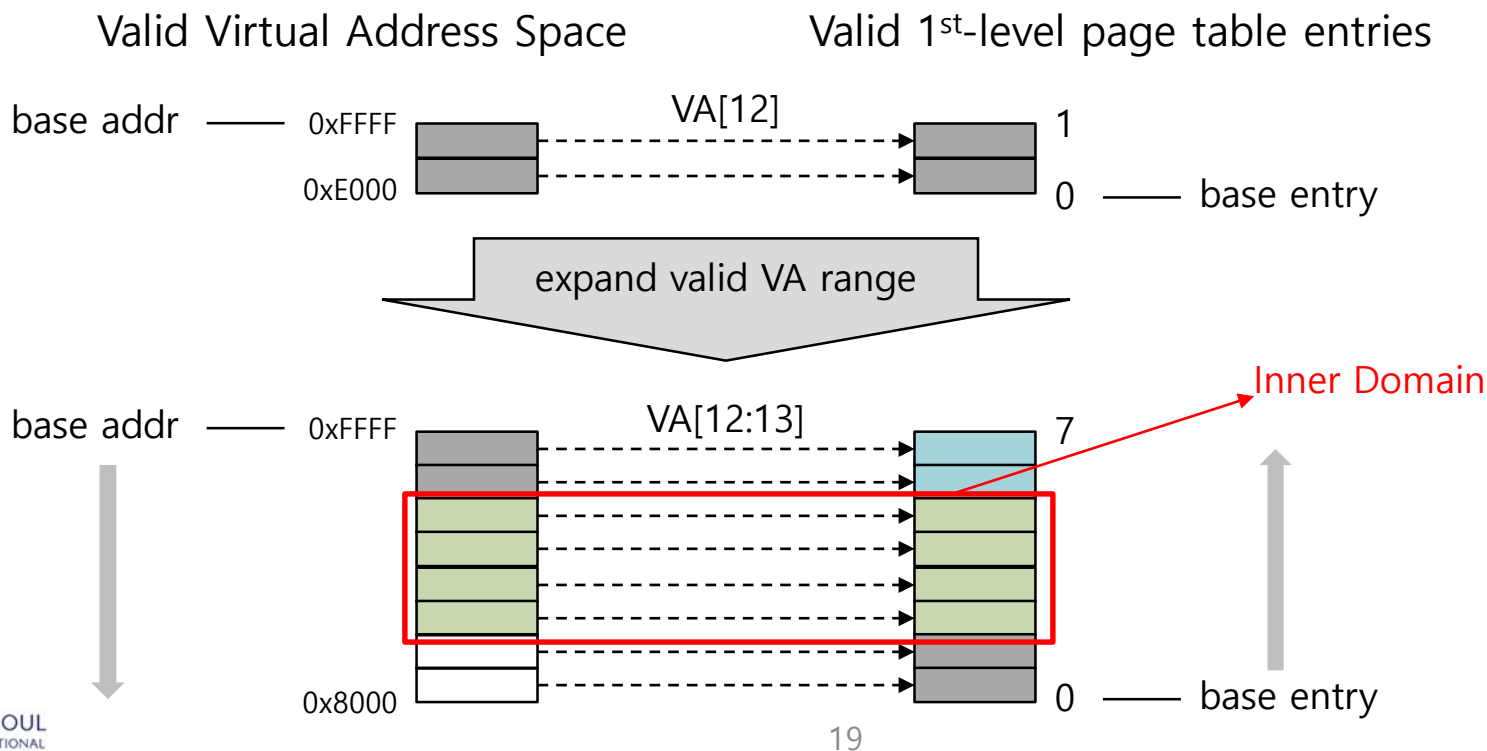


Intra-level isolation mechanism

- System software that runs with TTBR1_EL1
 - ex) Normal OS and secure OS



- Valid virtual address space and valid 1st-level page table entries change in the opposite direction



Domain switching mechanism

○ A way to enter the inner domain

<pre> mrs x5, DAIF stp x30, x5, [sp, #-16]! msr DAIFset, 0x3 1: mrs x5, tcr_el1 and x5, x5, #0xffffffffdffff orr x5, x5, #0x400000 msr tcr_el1, x5 isb mov x6, #0xc03f mov x7, #0x1b movk x6, #0xc07f, lsl #16 movk x7, #0x8059, lsl #16 and x5, x5, x6 cmp x5, x7 b.ne 1b mrs x6, mpidr_el1 ubfx x5, x6, #8, #4 and x6, x6, #0xf orr x6, x6, r5, lsl #2 add x6, x6, #1 adrp x5, InnerDomain_stack add x5, x5, x6, lsl #12 mov x6, sp mov sp, x5 str x6, [sp, #-8]! adrp x5, InnerDomain_handler blr x5 </pre>	}	<p>Disable interrupts</p>
<pre> mrs x5, tcr_el1 and x5, x5, #0xffffffffdffff orr x5, x5, #0x400000 msr tcr_el1, x5 isb </pre>	}	<p>Configure TCR to expand valid virtual address range and reveal the inner domain</p>
<pre> mov x6, #0xc03f mov x7, #0x1b movk x6, #0xc07f, lsl #16 movk x7, #0x8059, lsl #16 and x5, x5, x6 cmp x5, x7 b.ne 1b </pre>	}	<p>Verify the value of TCR</p>
<pre> mrs x6, mpidr_el1 ubfx x5, x6, #8, #4 and x6, x6, #0xf orr x6, x6, r5, lsl #2 add x6, x6, #1 adrp x5, InnerDomain_stack add x5, x5, x6, lsl #12 mov x6, sp mov sp, x5 str x6, [sp, #-8]! </pre>	}	<p>Switch to the inner domain stack</p>
<pre> adrp x5, InnerDomain_handler blr x5 </pre>	}	<p>Jump to the inner domain</p>

Domain switching mechanism

○ A way to return back to the outer domain

```
ldp x6, [sp], #8
mov sp, x6
```

} Switch to the outer domain stack

```
mrs x5, tcr_el1
and x5, x5, #0xffffffffbfffff
orr x5, x5, #0x20000
msr tcr_el1, x5
```

} Configure TCR to reduce valid virtual address range and hide the inner domain

```
mov x6, #0xc03f
mov x7, #0x1b
movk x6, #0xc07f, lsl #16
movk x7, #0x801b, lsl #16
and x5, x5, x6
cmp x5, x7
b.ne 2b
```

} Verify the value of TCR

```
ldp x30, x5, [sp], #16
msr DAIF, x5
isb
```

} Enable interrupts

```
ret
```

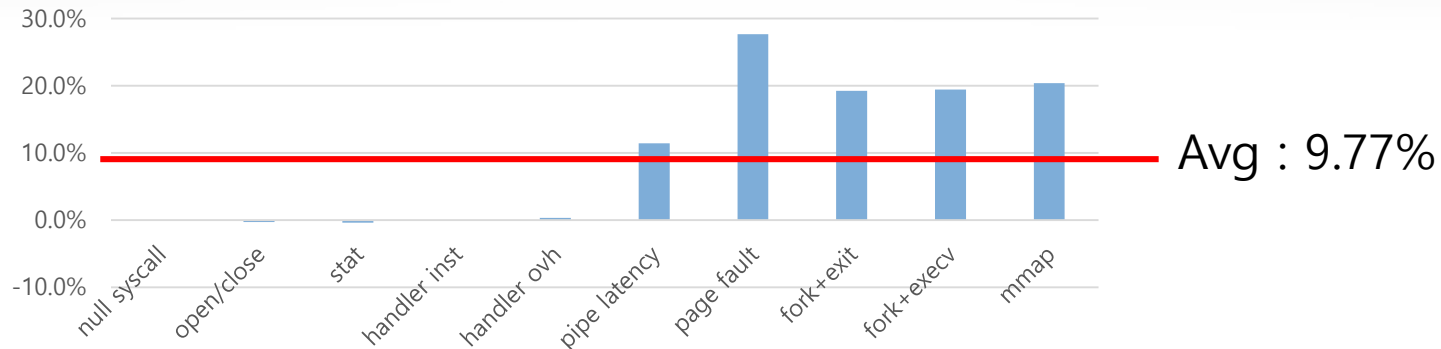
Evaluation

- ◎ V2M-Juno r1 platform
 - Cortex-A57 1.15 GHz dual-core
 - Cortex-A53 650 MHz quad-core
 - 2 GB of DRAM

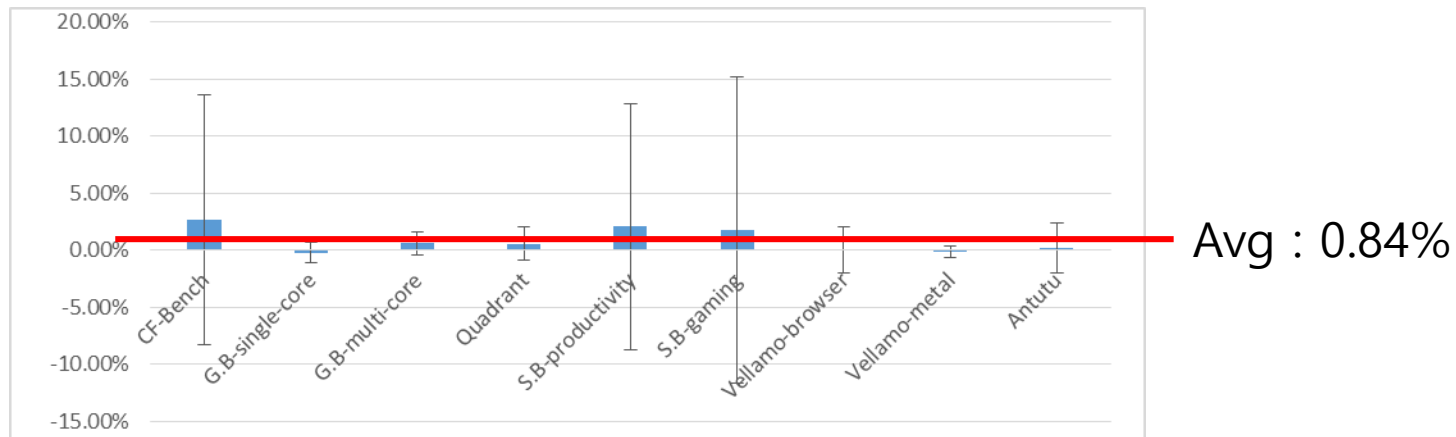
- ◎ Target
 - AArch64 Linux Kernel 3.10 of Android 5.1.1

Evaluation

- Round-Trip Cycles between the outer and inner domains
 - Big core : 424 cycles, Little core : 210 cycles
- LMbench to measure the kernel performance



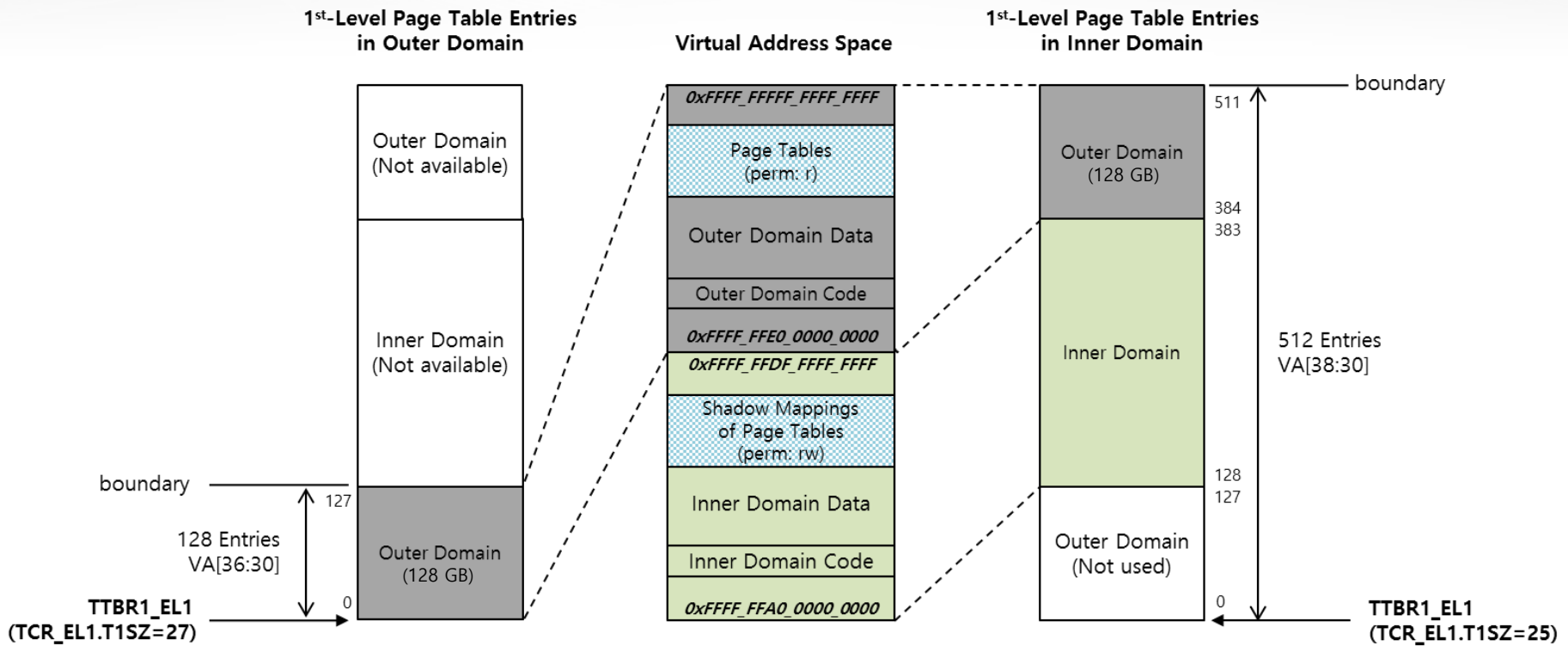
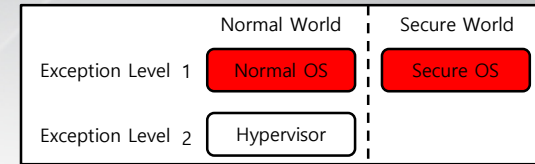
- Application benchmarks to measure the system performance



Thank you!

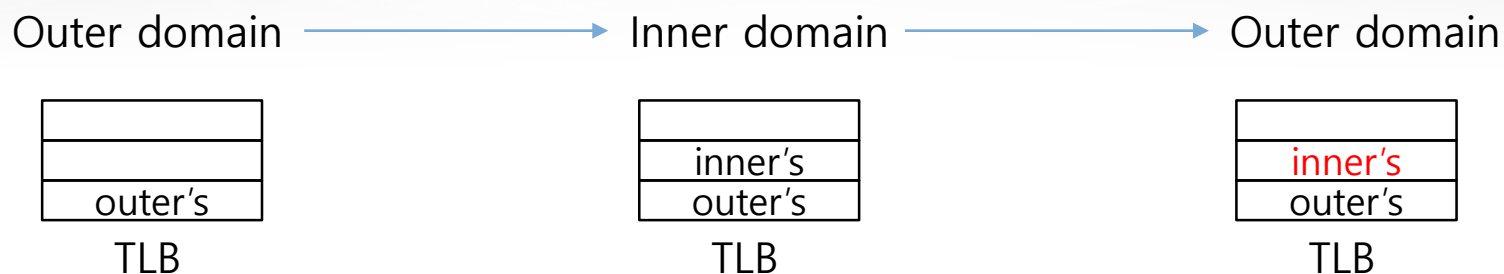


Intra-level isolation mechanism



Intra-level isolation mechanism

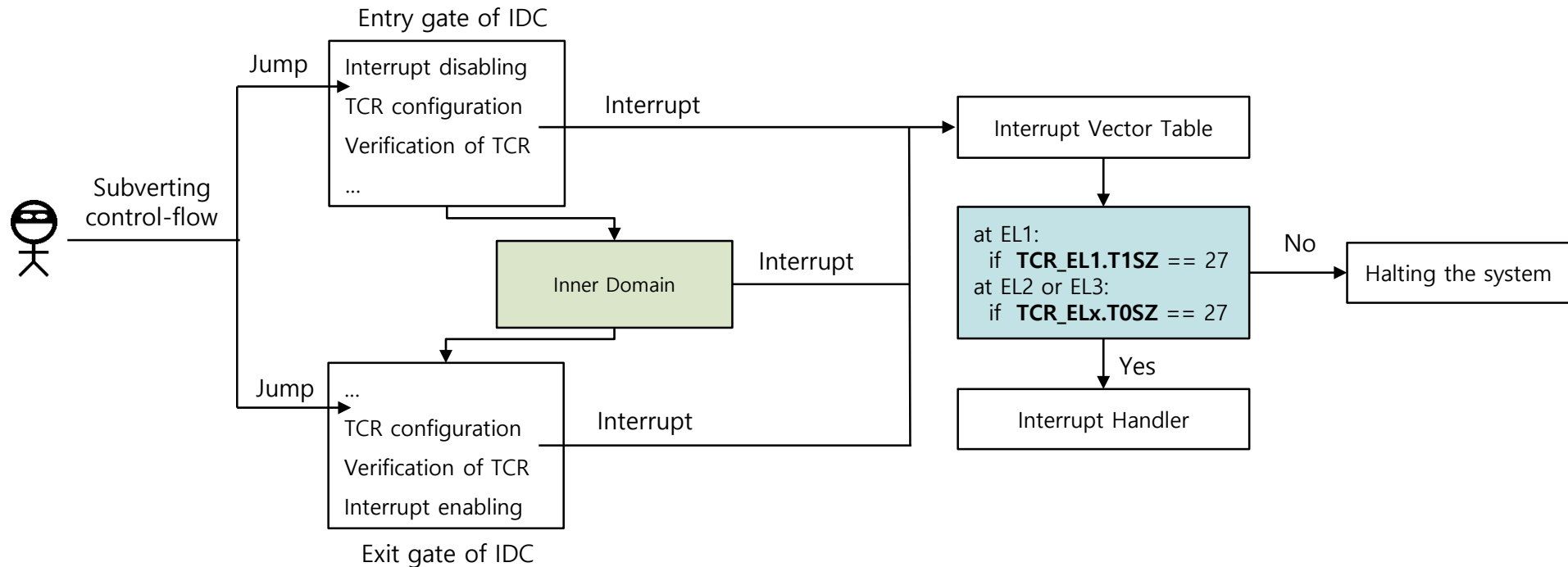
- ⦿ An attacker in the outer domain would access the inner domain through the cached TLB entries



- ⦿ How to prevent the outer domain from referencing cached TLB entries for the inner domain?
 - at Exception Level 1
 - use different ASIDs between the two domains
 - at Exception Level 2
 - invalidate cached TLB entries when switching between the two domains

Domain switching mechanism

- An attacker would cause a malicious interrupt to bypass the verification process for TCR and access to the inner domain
 - thwarting this attack by inserting a code snippet to verify the value of TCR



Efficiency of the domain switching mechanism

- Round-Trip Cycles between the outer and inner domains
 - Measured by the performance monitor provided by AArch64

	Big core		Little core	
	with ASID	with TLB inv.	with ASID	with TLB inv.
RTC	424	832	210	249

Efficiency of Hilps

LMBench to measure the kernel performance

	Big core		Little core	
	with ASID	with TLB inv.	with ASID	with TLB inv.
null syscall	0.00 %	0.00 %	2.33 %	2.33 %
open/close	-0.31 %	1.10 %	0.16 %	0.71 %
stat	-0.38 %	0.38 %	0.99 %	1.8 %
handler inst	0.00 %	1.47 %	0.00 %	0.00 %
handler ovh	0.31 %	1.84 %	-0.67 %	-0.17 %
pipe latency	11.40 %	43.48 %	6.89 %	19.10 %
page fault	27.66 %	102.13 %	31.32 %	96.44 %
fork+exit	19.20 %	61.89 %	14.57 %	44.95 %
fork+execv	19.42 %	55.34 %	12.44 %	41.71 %
mmap	20.36 %	71.85 %	11.45 %	44.35 %
average	9.77 %	33.95 %	7.95 %	25.12 %

Application benchmarks to measure the system performance

		with ASID	with TLB inv.
CF-Bench		2.68 %	12.96 %
GeekBench	single core	-0.21 %	0.31 %
	multi core	0.59 %	0.30 %
Quadrant		0.56 %	-0.02 %
Smartbench	productivity	2.07 %	-2.56 %
	gaming	1.74 %	1.32 %
Vellamo	browser	0.07 %	1.12 %
	metal	-0.13 %	0.15 %
Antutu		0.17 %	1.79 %
aberaage		0.84 %	1.71 %