INTEL SGX

COMPUTATIONAL BACKGROUND

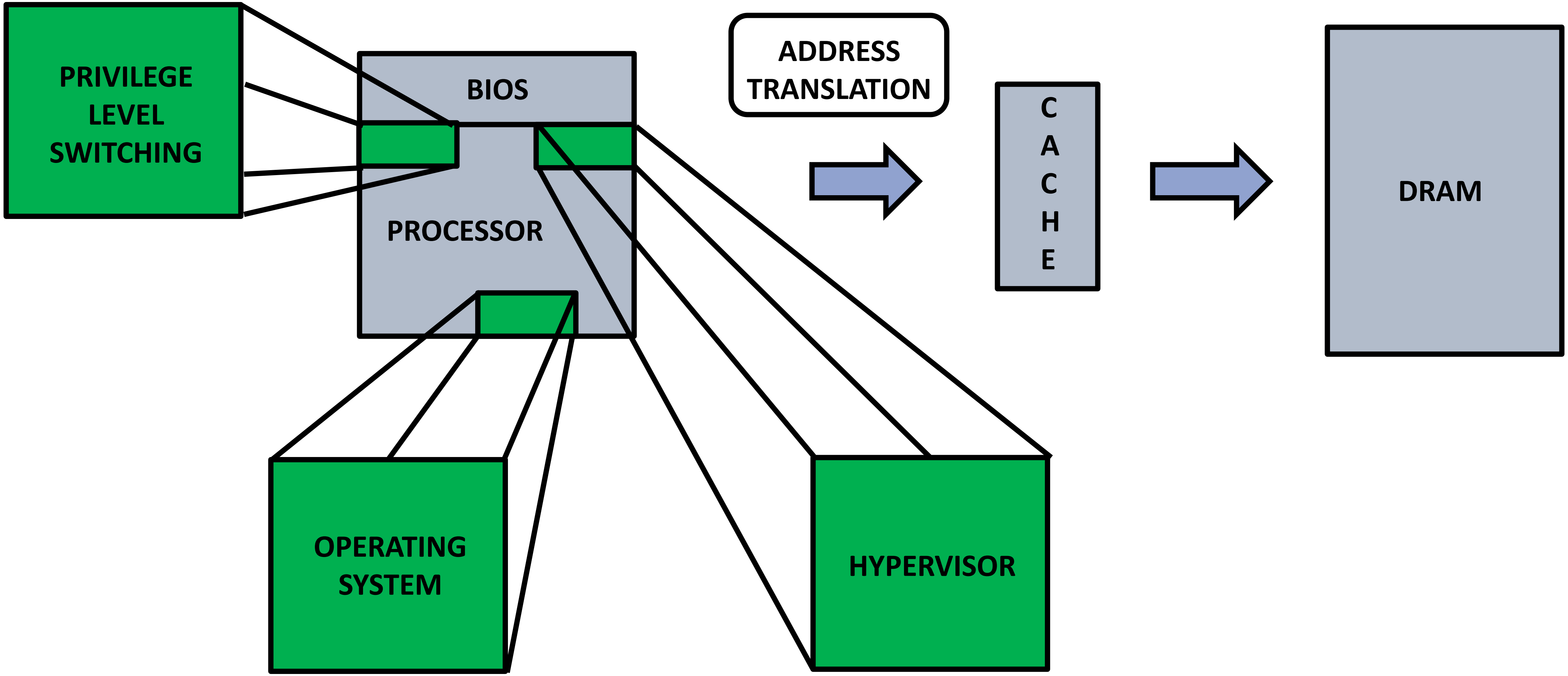
HAMZA OMAR

SECTIONS

- **OVERVIEW.**
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- Execution Contexts.
- Segment Registers.
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.

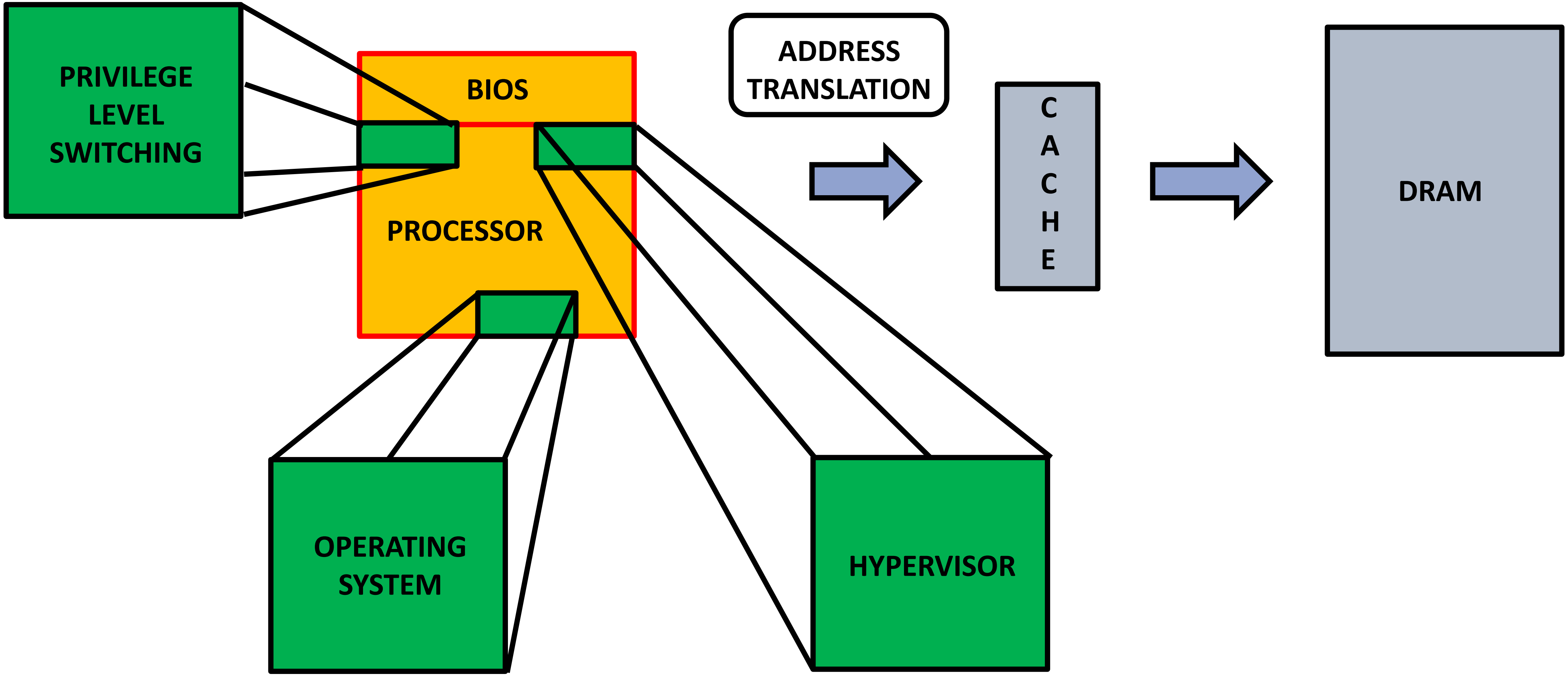
OVERVIEW

- Summarizing general architectural principles behind Intel’s most popular computer processors.
- We will discuss Intel’s 64-bit version of x86 processor designed by AMD, also known as AMD64.
- Intel computers typically run **Operating systems** and **Hypervisors**.
- **Operating System:** Allocates the computer’s resources to the running processes.
- **Hypervisor:** Partitions the computer’s resources between the operating system instances running on the computer. It exposes fixed number of virtual CPUs (vCPUs) to each operating systems.
- System software uses **virtualization techniques** like address translation and setting software privilege levels, to isolate each piece of software that it manages (process or operating system) from the rest of the software running on the computer.
- **Execution Core VS DRAM.** (caches)
- Intel’s Software Development Manual (SDM).


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

SECTIONS

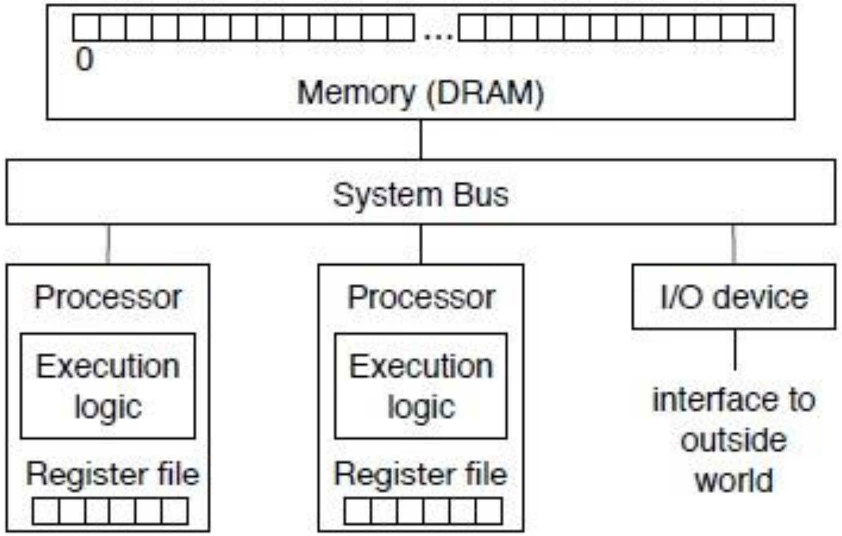
- Overview.
- **COMPUTATIONAL MODEL.**
- Software Privilege Levels.
- Address Translation.
- Execution Contexts.
- Segment Registers.
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

COMPUTATIONAL MODEL

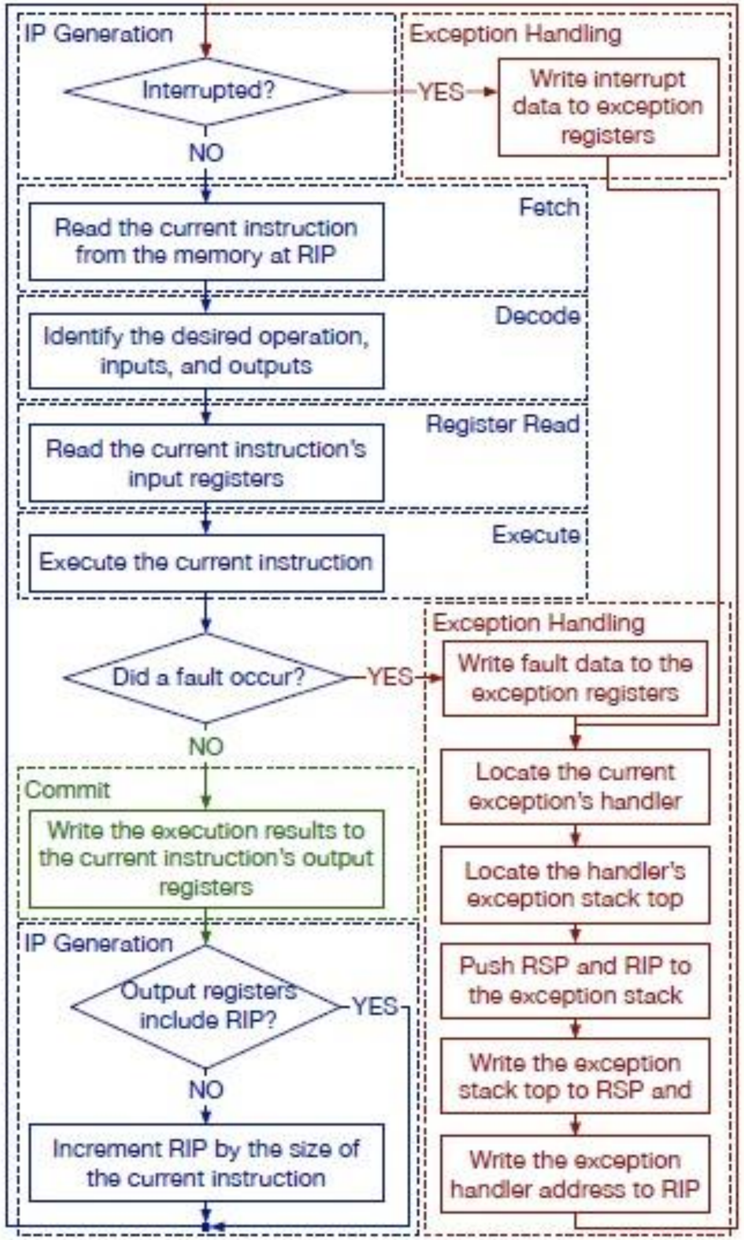
- In a computer system, the processor(s), DRAM and all the outside world interfaces communicate with each other through a common system bus (broadcast network).

- During each clock cycle, at most one of the devices connected to the system bus can send a message **(Read-Request/Read-Response).** Each device attached to the bus decodes the operation codes and addresses of all the messages sent on the bus and ignores when they do not require it.



COMPUTATIONAL MODEL

- Computation like ADD A, B, C.
- **RIP** (Instruction Pointer Register): Stores Next Address.
- **RSP** (Stack Pointer Register): Address of Top Most Element.
- INTEL uses variable sized instruction encoding.
- In case of faults, **Exception handler** is called.
- What about I/O devices and Interrupts ?



IP Generation

Interrupted?

YES

NO

Exception Handling

Write interrupt data to exception registers

Fetch

Read the current instruction from the memory at RIP

Decode

Identify the desired operation, inputs, and outputs

Register Read

Read the current instruction's input registers

Execute

Execute the current instruction

Did a fault occur?

YES

NO

Exception Handling

Write fault data to the exception registers

Locate the current exception's handler

Locate the handler's exception stack top

Push RSP and RIP to the exception stack

Write the exception stack top to RSP and

Write the exception handler address to RIP

Commit

Write the execution results to the current instruction's output registers

IP Generation

Output registers include RIP?

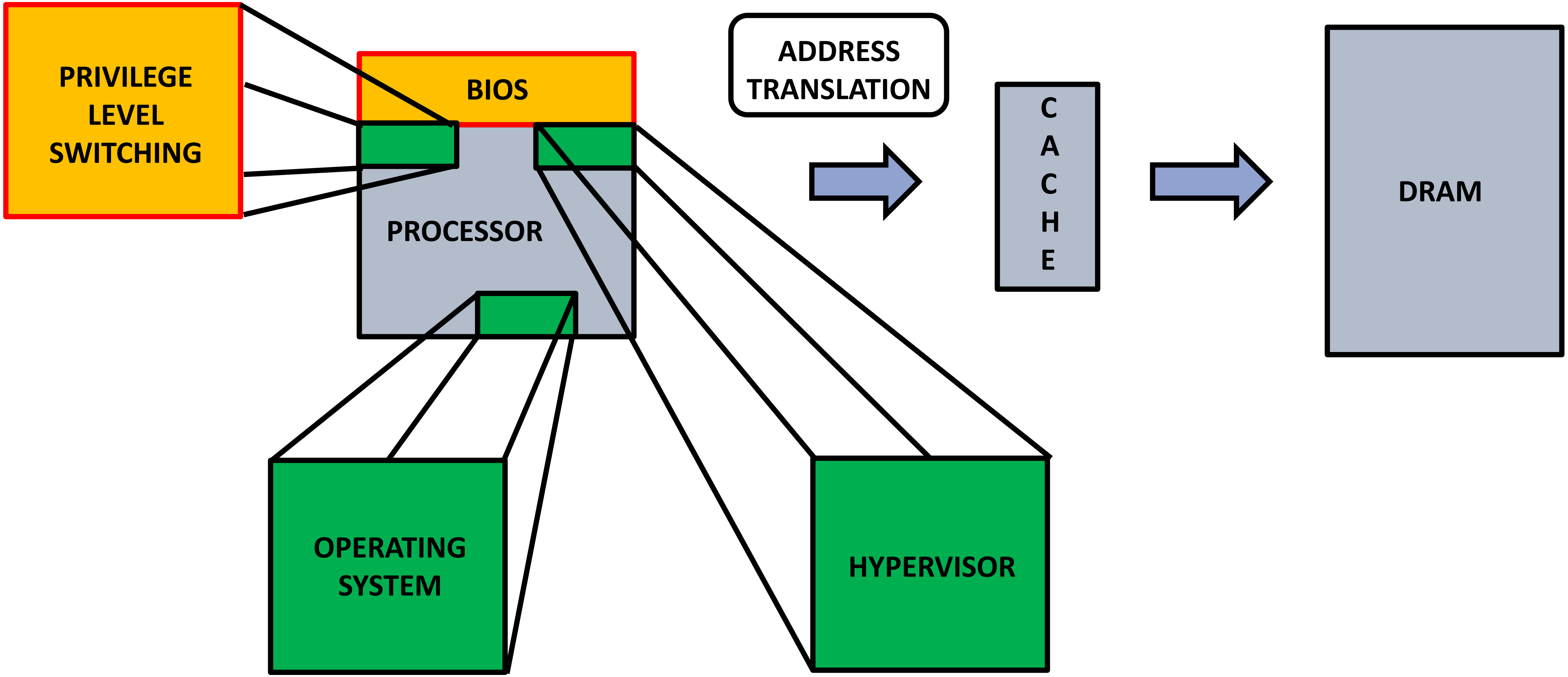
YES

NO

Increment RIP by the size of the current instruction

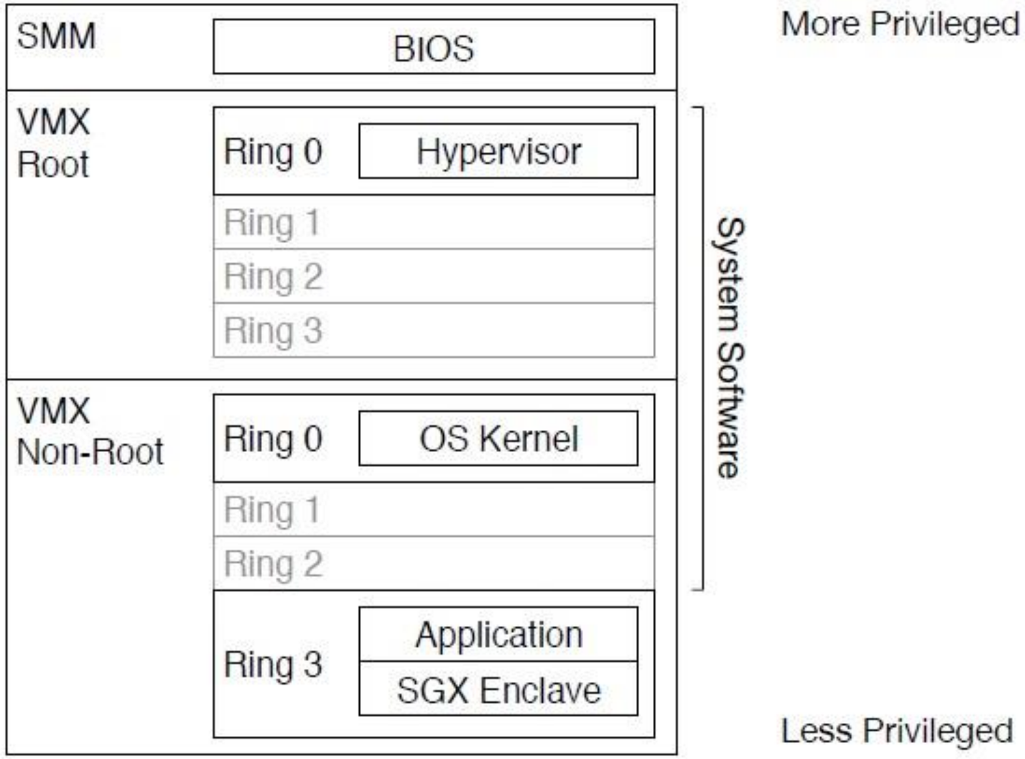
SECTIONS

- Overview.
- Computational Model.
- **SOFTWARE PRIVILEGE LEVELS.**
- Address Translation.
- Execution Contexts.
- Segment Registers.
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

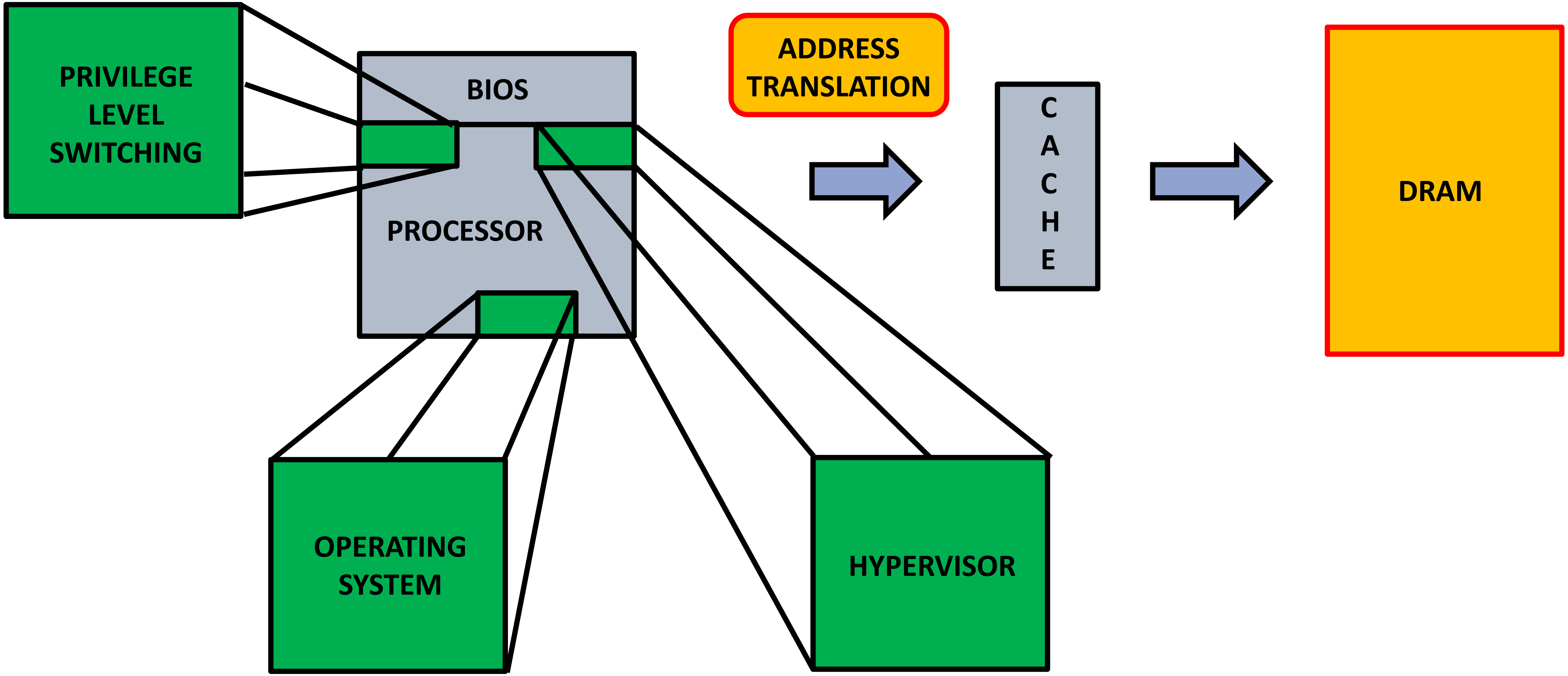
SOFTWARE PRIVILEGE LEVELS

- Commodity CPUs (x86) run software are **four** different privilege levels. So, each privilege level is more powerful than the one's below.
- Trust at higher privilege software.
- **SMM** (Motherboard Manufacturers Region).
  - ❑ Fan Control, Deep Sleep, Boot-Strapping.
- (Virtual Machine Extension) **VMX-Root**.
  - ❑ **Hypervisor** runs at **ring0** allocates resources.
  - ❑ It makes each OS to believe it is using its own CPU.
- **VMX-Non-Root.**
  - ❑ **OS into small Kernels –** at higher privilege level.
  - ❑ **Ring 3 –** Application code ,Web-server or game client.



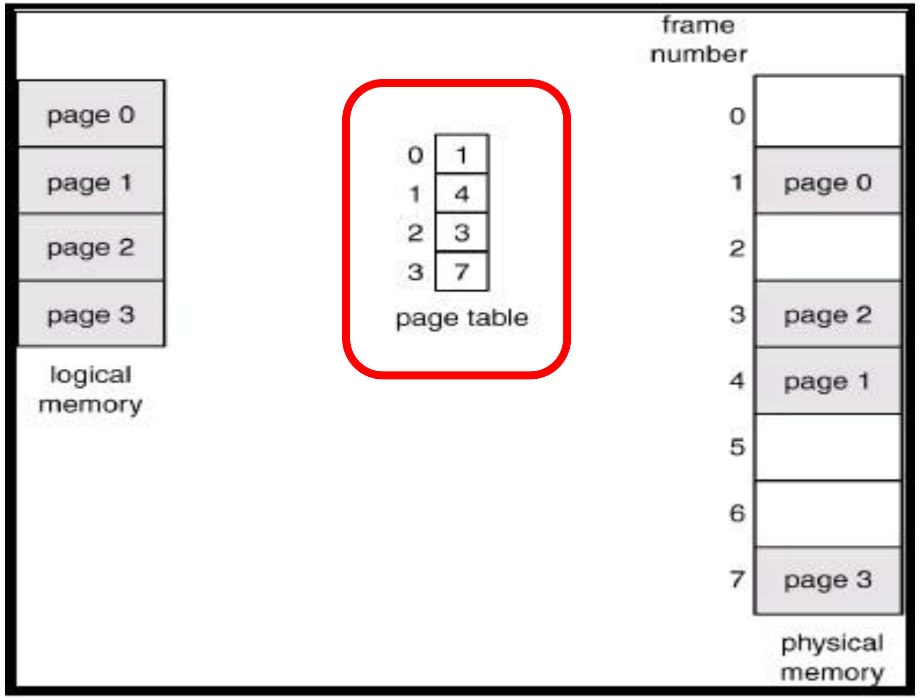
SECTIONS

- Overview.
- Computational Model.
- Software Privilege Levels.
- **ADDRESS TRANSLATION.**
- Execution Contexts.
- Segment Registers.
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
C A C H E
DRAM
OPERATING SYSTEM
HYPERVISOR

ADDRESS TRANSLATION
Pages & Page Tables:

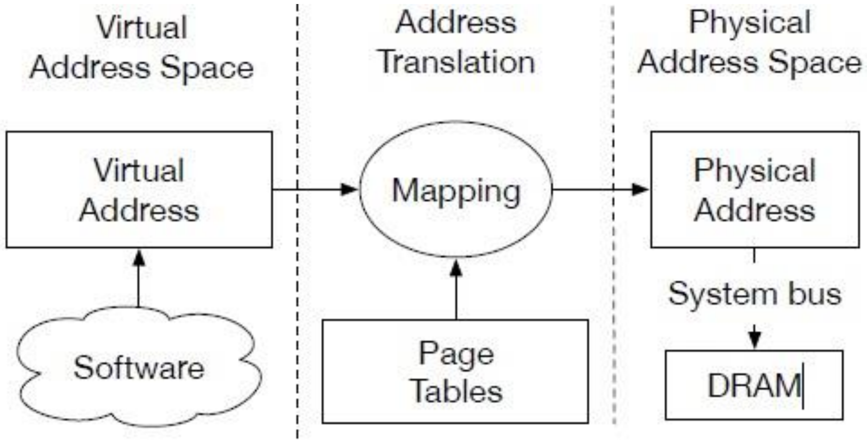
- Logical Address space is very large (64 bit has sixteen quintillion bytes). On the other hand our DRAM also becomes expensive, as the size increases when we need to store large amount of data.
- The logical address space is converted to fixed size blocks called **pages** similarly, the DRAM space is converted to **frames**.
- In practice, the process of paging becomes more complex so we extend our page tables to hierarchical page tables, where one page table points to the other page table and this process goes on until we get the actual physical address.

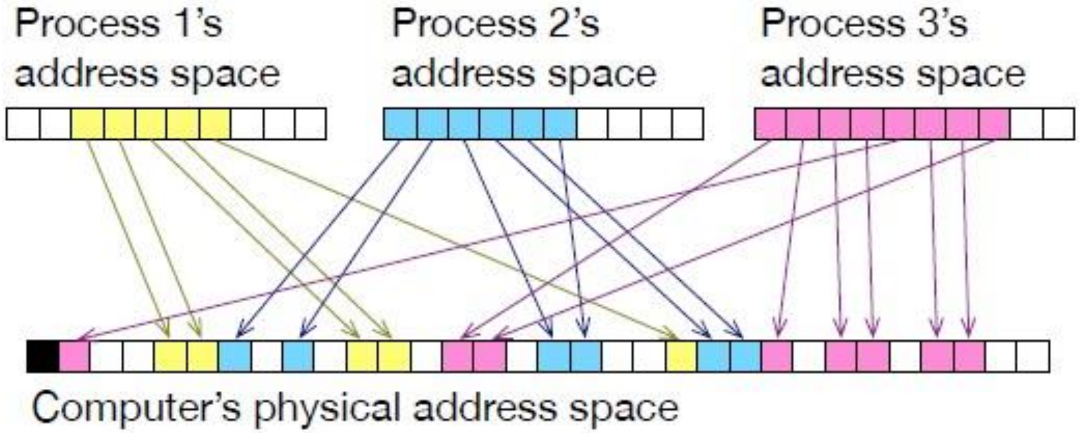


ADDRESS TRANSLATION
Concepts:

- System software relies on the CPU's address translation mechanism for implementing isolation among less privileged pieces of software (applications or operating systems).

- **Virtual Memory Abstraction**
  - ❑ Each process gets a separate virtual address space that references the memory to that process.
  - ❑ OS multiplexes the DRAM between processes, while Developers see as if it owns the entire memory.



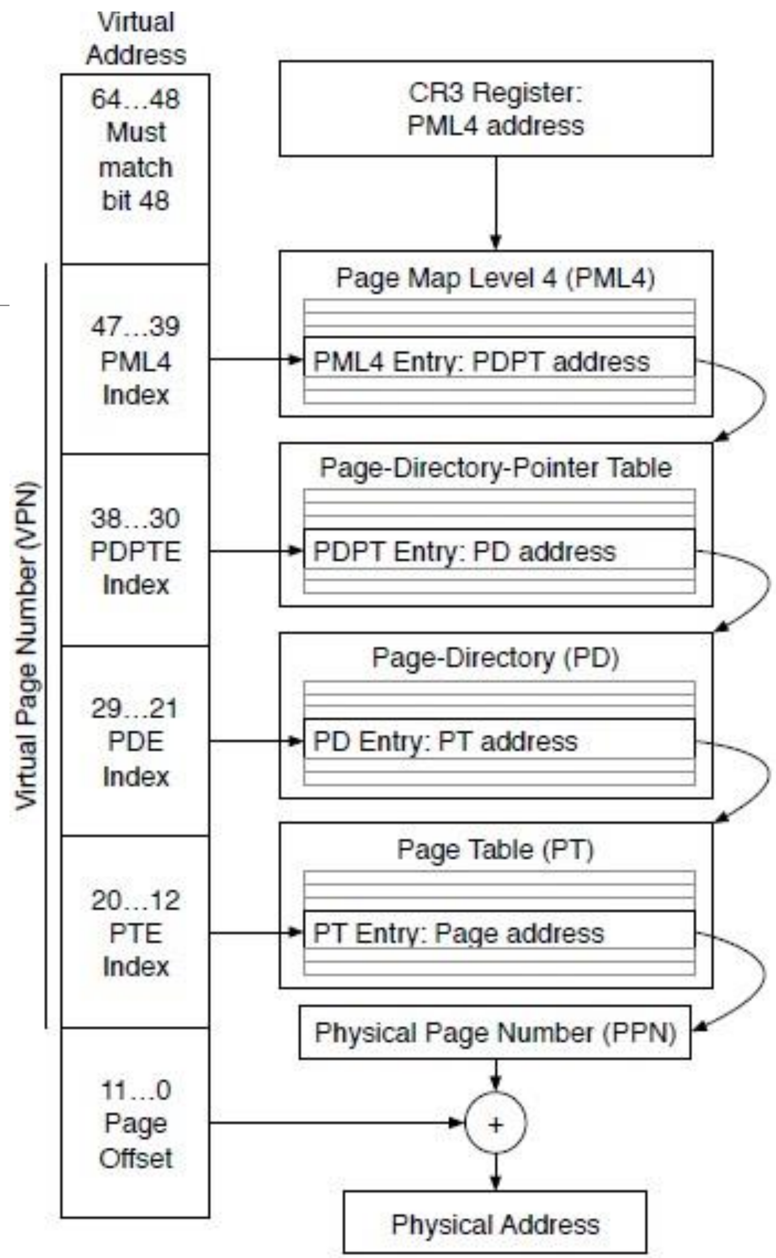

Process 1's address space
Process 2's address space
Process 3's address space
Computer's physical address space

ADDRESS TRANSLATION
Concepts:

- **Why need isolation ?**

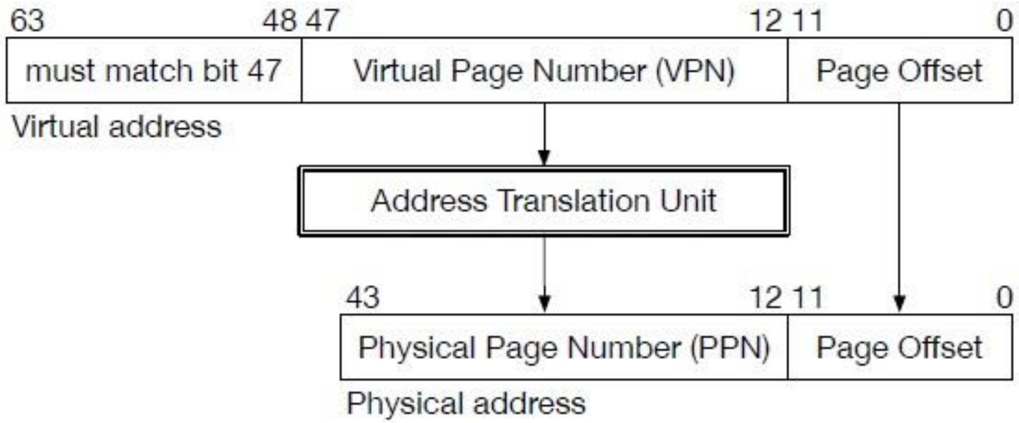
Address Translation isolates the processes from each other and prevents application code from accessing memory mapped devices directly. So application's bug cannot affect the higher level privilege applications such as OS kernel itself.

❑ In 64-bit OS**, 48-bit virtual address is mapped to physical address.**
❑ **MMU** does the Translation process.
❑ [11:0] – Unchanged. These bits partition address space into pages.
❑ [47:12] – Grouped into four 9-bit indexes for indexing to different hierarchical page tables.
❑ [47:39] – Page Map Level, [38:30] – Page Directory Pointer Table, [29:21] – Page Directory, [20:12] – Page Table.



ADDRESS TRANSLATION
VPN to PPN:

- Address translation is actually the mapping of Virtual Page Number (VPN) to Physical Page Number (PPN).



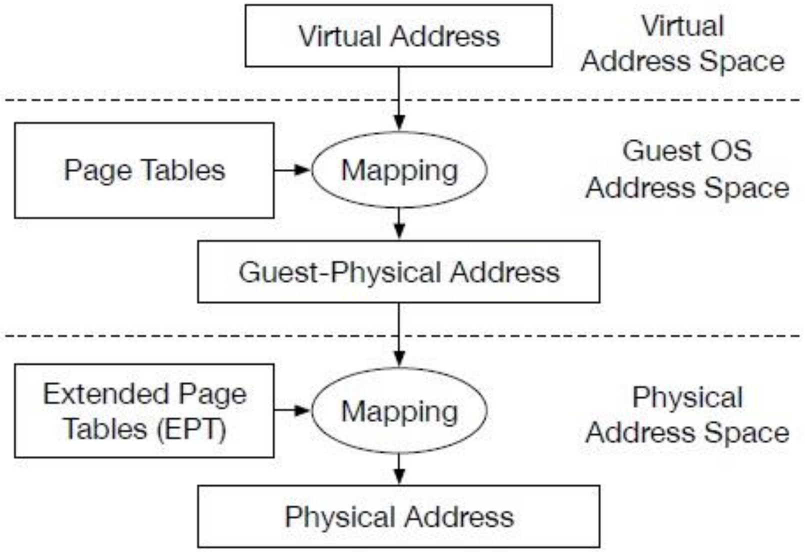
- One more advantage of using address translation is to run applications whose collective memory demand exceeds the amount of DRAM. In such cases, OS evicts infrequently used pages from DRAM to our hard-drives or SSD. This process is called **Page Swapping.**

- When an application process attempts to access a page that has been evicted, the OS "steps in" and reads the missing page back into DRAM from Disk. In order to do this, the OS might have to evict a different page from DRAM, effectively swapping the contents of a DRAM page with a disk page.

ADDRESS TRANSLATION
Virtualization:

- Hypervisor runs multiple operating systems at the same time and each OS is under an impression that it owns the entire computer's DRAM. **This creates some tension !!**
  - **SOLUTION:** Introduction of Extended Page Table.

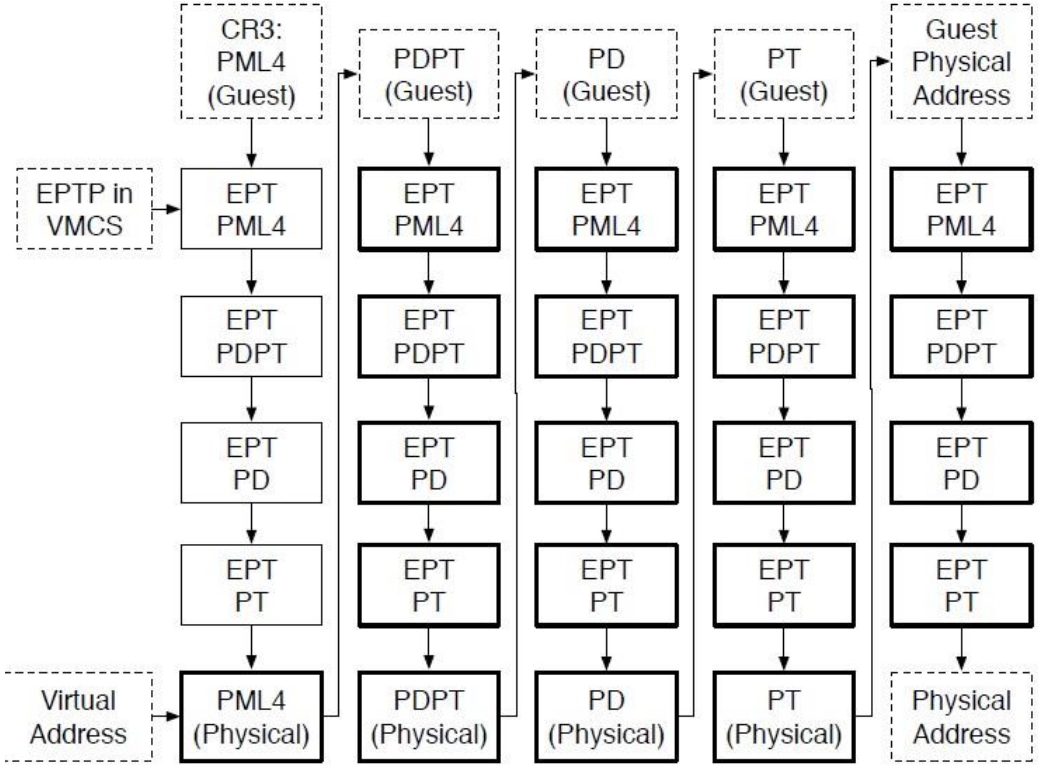
- **Extended Page Table:**
  - The hypervisor multiplexes the computer's DRAM between the operating systems' guest-physical address spaces via the second layer of address translations, which uses extended page Tables (EPT) to map guest-physical addresses to physical addresses.
  - **EPT** uses the same data structure as the page tables.
  - When caching some addresses, the guest space page table entries are traversed in horizontal while the EPT entities are traversed in vertical direction.



ADDRESS TRANSLATION
Page Table Attributes & Flags:

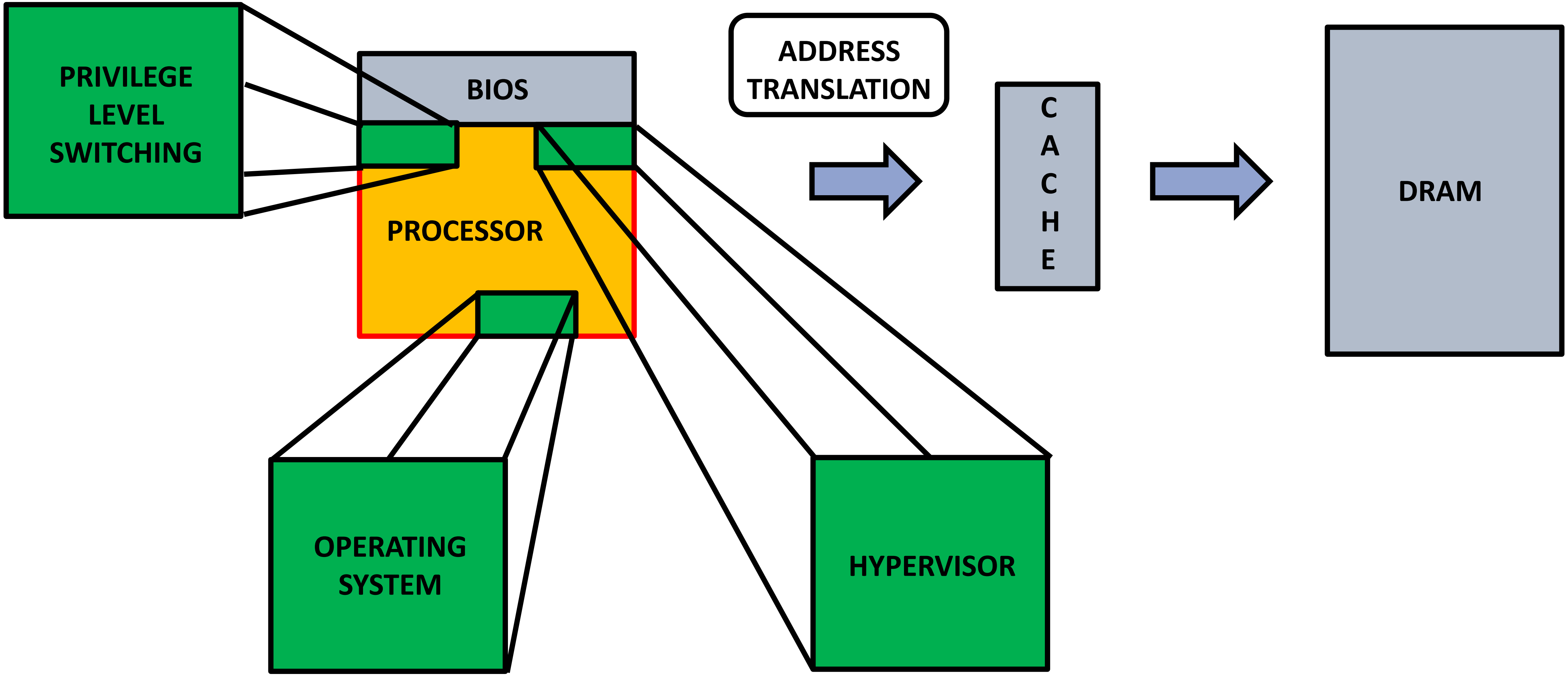
- Each level in the kernel's page table requires a full walk of the hypervisor's extended page table (EPT). A translation requires up to 20 memory accesses (the bold boxes), assuming the physical address of the kernel's PML4 is cached.

- **FLAGS:**
  - ❑ **A –** accessed flag is set to 1 whenever MMU reads a Page table entry.
  - ❑ **D –** dirty flag is set to 1 when an entry is accessed by a memory write operations.
  - ❑ **P –** Set to 0 when ever a page is evicted from DRAM.
  - ❑ **W –** can be set to 0 to prohibit any writes.
  - ❑ **XD –** disables instruction fetches from a page.
  - ❑ **S –** supervisor flag is set to 1 to prohibit any accesses from lower privilege level applications.



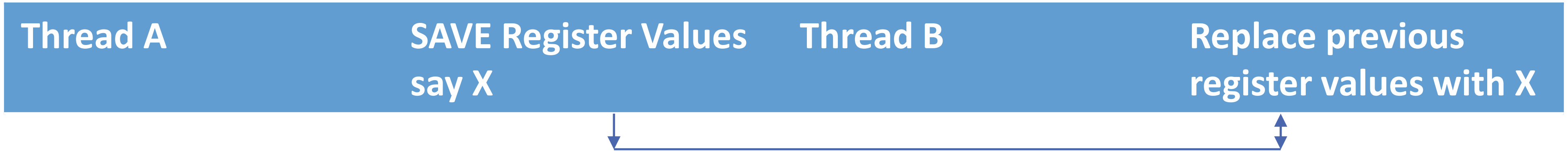
SECTIONS

- Overview.
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- **EXECUTION CONTEXTS.**
- Segment Registers.
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

EXECUTION CONTEXTS

- 64-bit architecture uses variety of CPU registers to interact with processor's feature.
- **Context Switching:** It is the procedure of storing the state of an active process for the CPU when it has to start executing a new one.



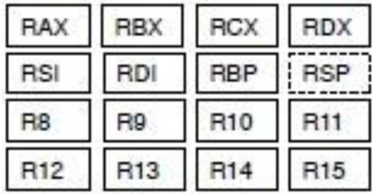
Thread A

SAVE Register Values say X

Thread B

Replace previous register values with X

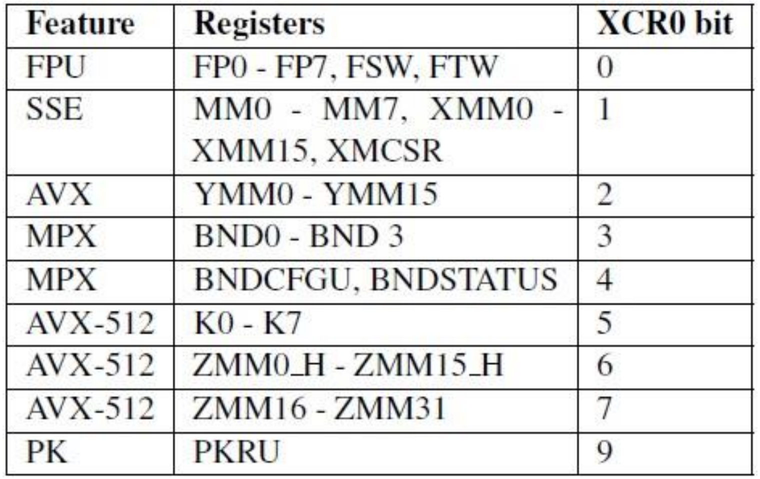
- Context switching also plays a part in executing code inside secure containers, so its design has security implications.
- Integers and memory addresses are stored in 16 general purpose registers (GPRs). The first 8 are the extended versions of 32-bit architecture. The other 8 are simply known as R9-R16.
- RSP: Stack pointer use **CALL/RETURN** functions like PUSH/POP.
- RIP: It has the address of the currently executing instruction.



RAX	RBX	RCX	RDX
RSI	RDI	RBP	RSP
R8	R9	R10	R11
R12	R13	R14	R15

EXECUTION CONTEXTS

- The Intel architecture provides a method for an OS kernel to save the values of feature-specific registers used by an application.
- OS kernel saves the feature specific register values for the application using **XSAVE** which takes in a feature requested bitmap **(RFBM).** It writes the register values used by the feature whose RFBM bits are set to 1.
- Figure shows some feature-specific Intel's architecture registers.
- Kernel knows which XSAVE bitmap to use for context switching.
  - ❑ Sets XCR0 register to feature bitmap.
  - ❑ CPU generates a fault if the application attempts to use features that are not enabled by XCR0.
  - **Task State Segment** (TSS) was designed to implement **hardware context switching**. The descriptor is stored in the **Task Register** (TR).



Feature	Registers	XCR0 bit
FPU	FP0 - FP7, FSW, FTW	0
SSE	MM0 - MM7, XMM0 - XMM15, XMCSR	1
AVX	YMM0 - YMM15	2
MPX	BND0 - BND 3	3
MPX	BNDCFGU, BNDSTATUS	4
AVX-512	K0 - K7	5
AVX-512	ZMM0_H - ZMM15_H	6
AVX-512	ZMM16 - ZMM31	7
PK	PKRU	9

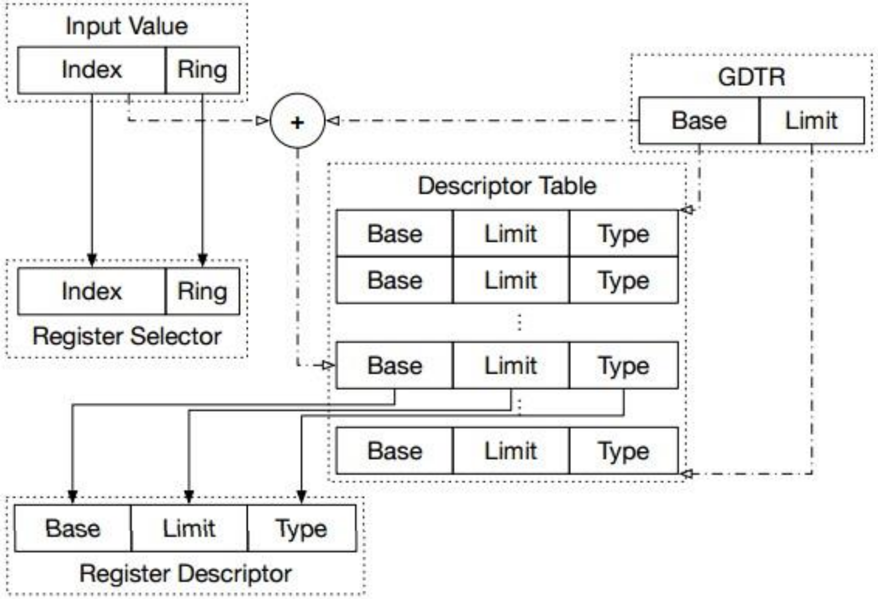
SECTIONS

- Overview.
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- Execution contexts.
- **SEGMENT REGISTERS.**
- Privilege Level Switching.
- OOO and Speculative Execution.
- Caches.

SEGMENT REGISTERS

- INTEL's architecture instructions include the implicit use of few segments which are loaded into the processor's segment registers. These registers contain the base address of these segments.
  - ❑ **CS –** Code fetches use code segment.
  - ❑ **SS –** Instructions the reference the stack use stack segment.
  - ❑ **DS/ES –** Memory references use data segment or destination segment.
  - ❑ Modern OS effectively disable segmentation by putting the entire addressable space into one segment, which is loaded in CS and one data segment which is loaded in SS, DS and ES.

- These segment registers are of 16 bits.
  - ❑Each segment register has a hidden segment descriptor, which consists of a **base address, limit,** and **type information**, such as whether the descriptor should be used for executable code or data.
  - ❑ **[15:3] :** Indexing into descriptor table.
  - ❑ **[2:0] :** Privilege level (Ring number).
  - ❑ **GDT:** Global Descriptor Table (Register).



SEGMENT REGISTERS
Hardware Context Switching:

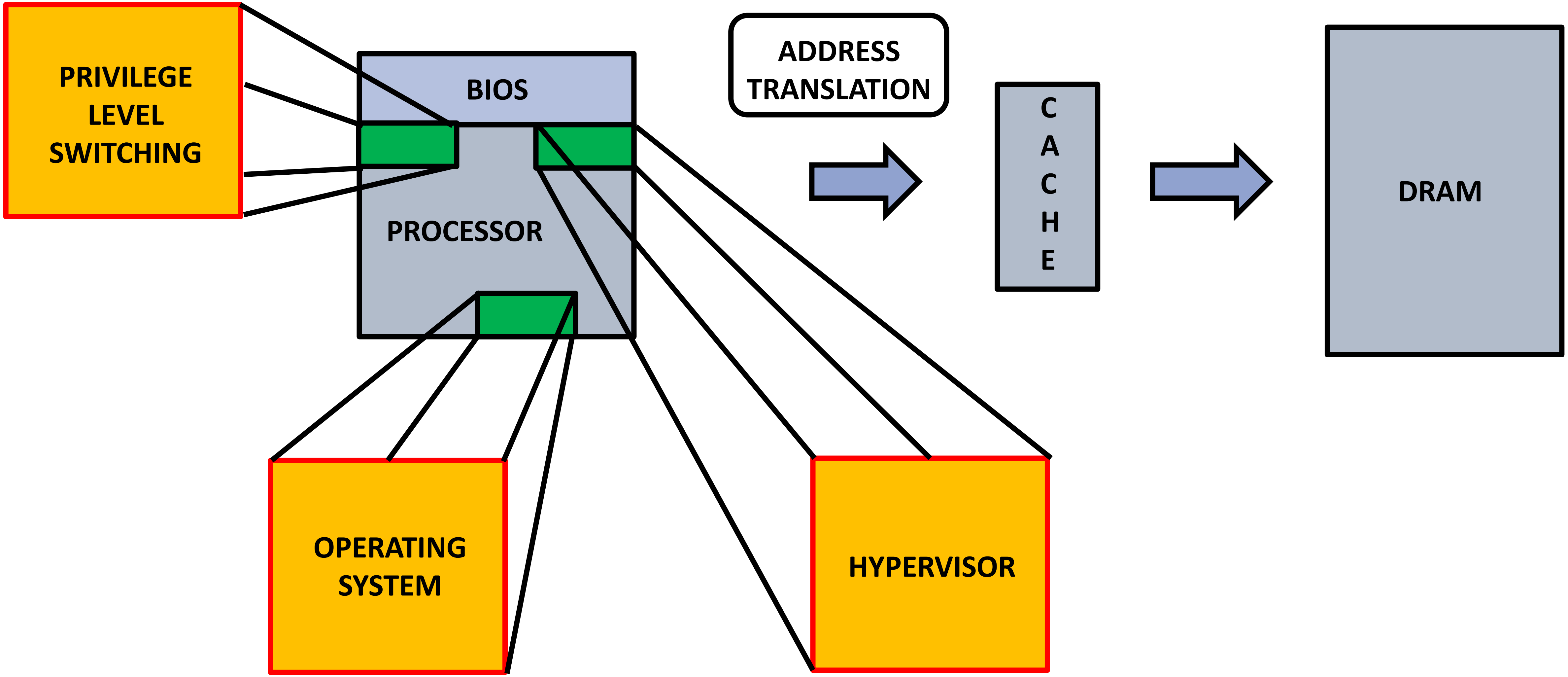
- Hardware context switching was removed from the 64-bit Intel's architecture. But **Task State Segment** (TSS) was reserved.

- Now, TSS is used for I/O map, which indicates which address space of I/O can be accessed directly from different privilege level applications e.g. Ring 3.

- Modern operating systems **do not allow** application software (Ring 3) any direct access to the I/O address space, so the kernel sets up a single TSS that is loaded into Task Register during early initialization, and used to represent all applications running under the OS.

SECTIONS

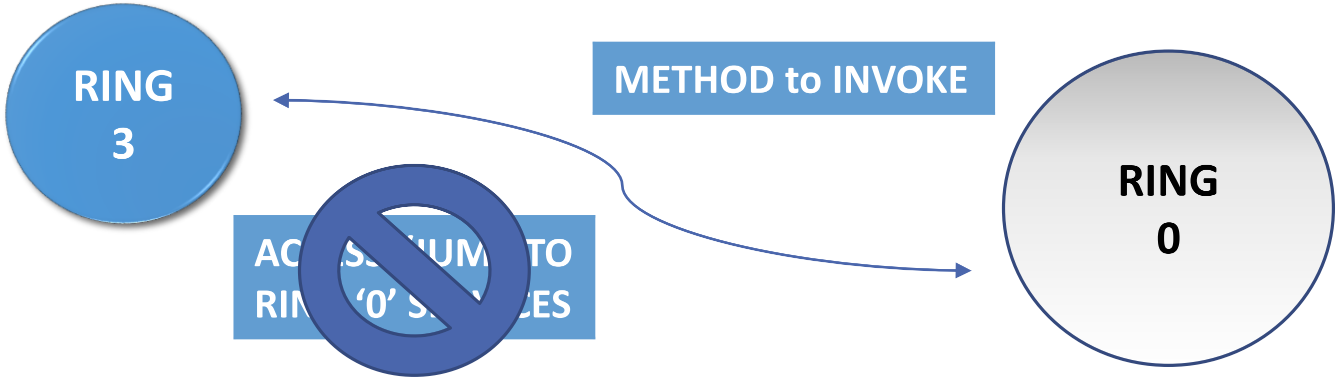
- Overview.
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- Execution contexts.
- Segment Registers.
- **PRIVILEGE LEVEL SWITCHING.**
- OOO and Speculative Execution.
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
C A C H E
DRAM
OPERATING SYSTEM
HYPERVISOR

PRIVILEGE LEVEL SWITCHING

WHY?

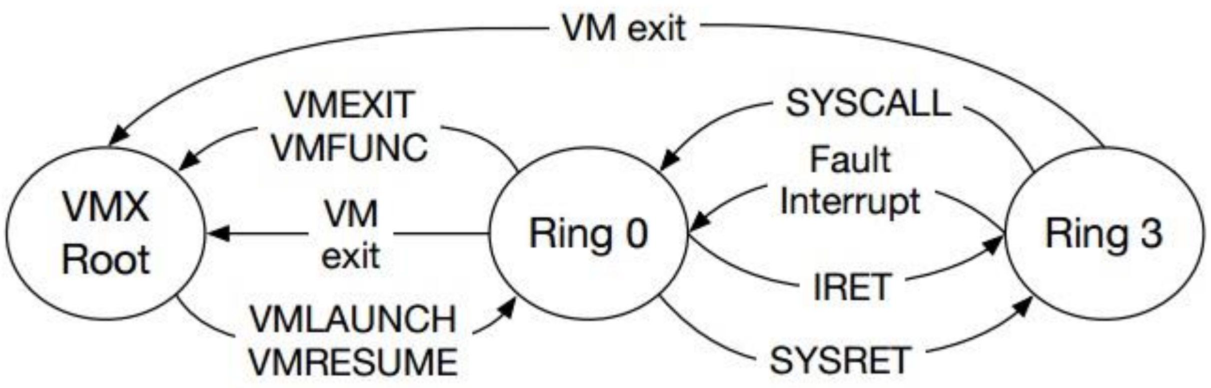
This would **compromise** the privileged software’s ability to enforce **security** and **isolation** invariants.



PRIVILEGE LEVEL SWITCHING
Secure Way:

- **CASE:** An application (Ring-3) wishes to write a file to the disk, the kernel (Ring-0) must check if the application's user has access to that file. If the ring 3 code could perform an arbitrary jump in kernel space, it would be able to skip the access check.

- **Transfer of Control:**
  - ❑ Intel's architecture includes privilege-switching mechanisms used to transfer control from less privileged software to **well-defined entry points** in more privileged software. So Ring-3 can access the services of Ring-0 but to some extent, not all services.



PRIVILEGE LEVEL SWITCHING

Control Transfer from 0 ↔ 3:

CALLS:

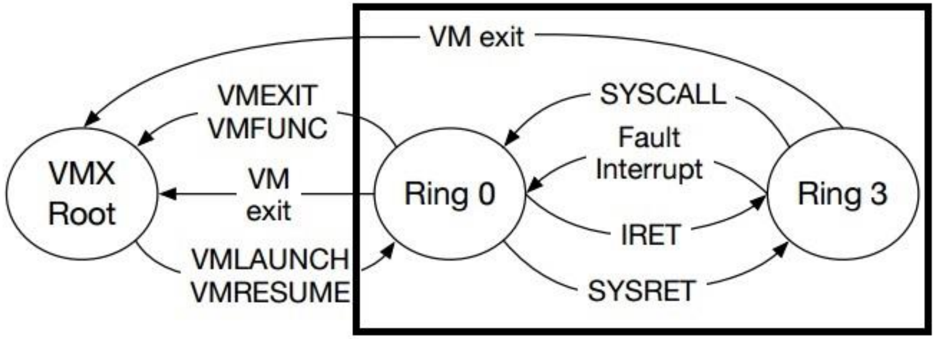
- **SYSCALL:**Ring-3 app. uses this call to gain access to specific entry points in Ring-0 region.
- **SYSRET:** Privilege level switched back to Ring-3 from Ring-0 by jumping to the address in RCX register, which is set by SYSCALL.

FAULTS:

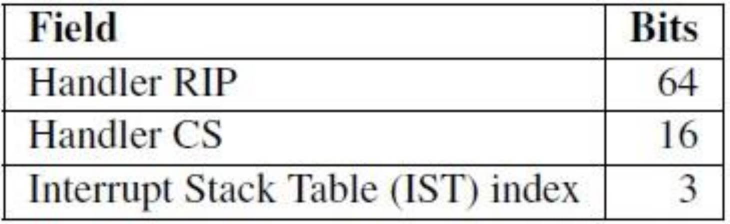
- When hardware exception occurs in ring-3 code, CPU performs a ring switch and calls the corresponding exception handler.
  - **#GP:** Performing disallowed operation, needs #GP handler.
  - **#PF:** when address translation encounters a page whose **P** flag is **0**, means page was evicted. This fault needs #PF handler which reads swapped out page back into DRAM.

IDT (Interrupt Descriptor Table):

- Exception handlers are a part of OS and their locations are in the first 32 entries of the IST.
  - When a hardware exception occurs, the execution state may be corrupted, and the current stack cannot be relied on.
  - Therefore, the CPU first uses the handler's IDT entry to set up a known good stack. **SS** is loaded with a null descriptor, and **RSP** is set to the IST value to which the **IDT** entry points.



Kernel Protects IDT, using **page tables,** from **Ring-3** to access it

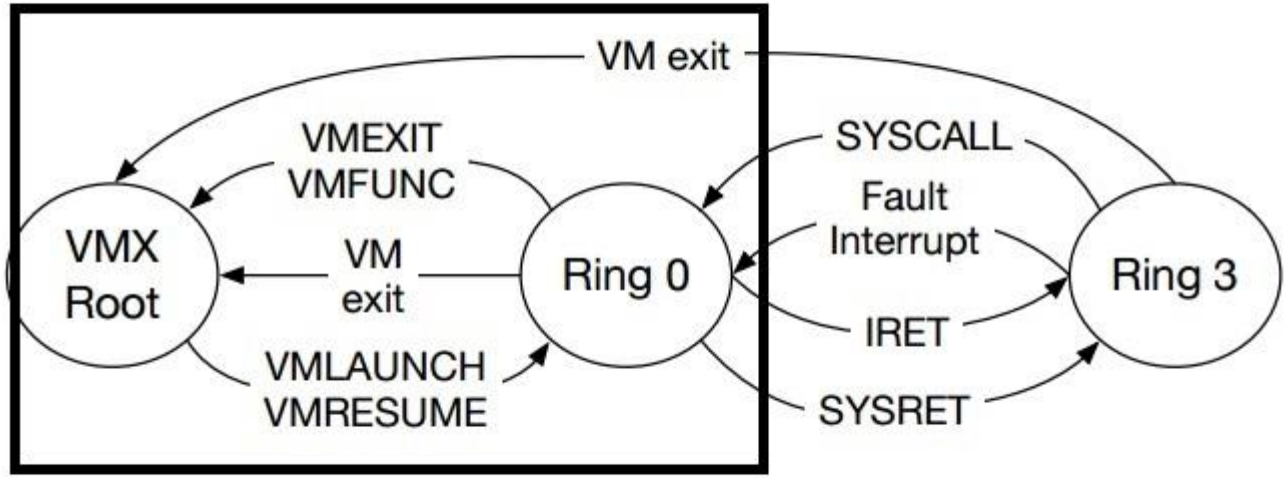


Field	Bits
Handler RIP	64
Handler CS	16
Interrupt Stack Table (IST) index	3

PRIVILEGE LEVEL SWITCHING

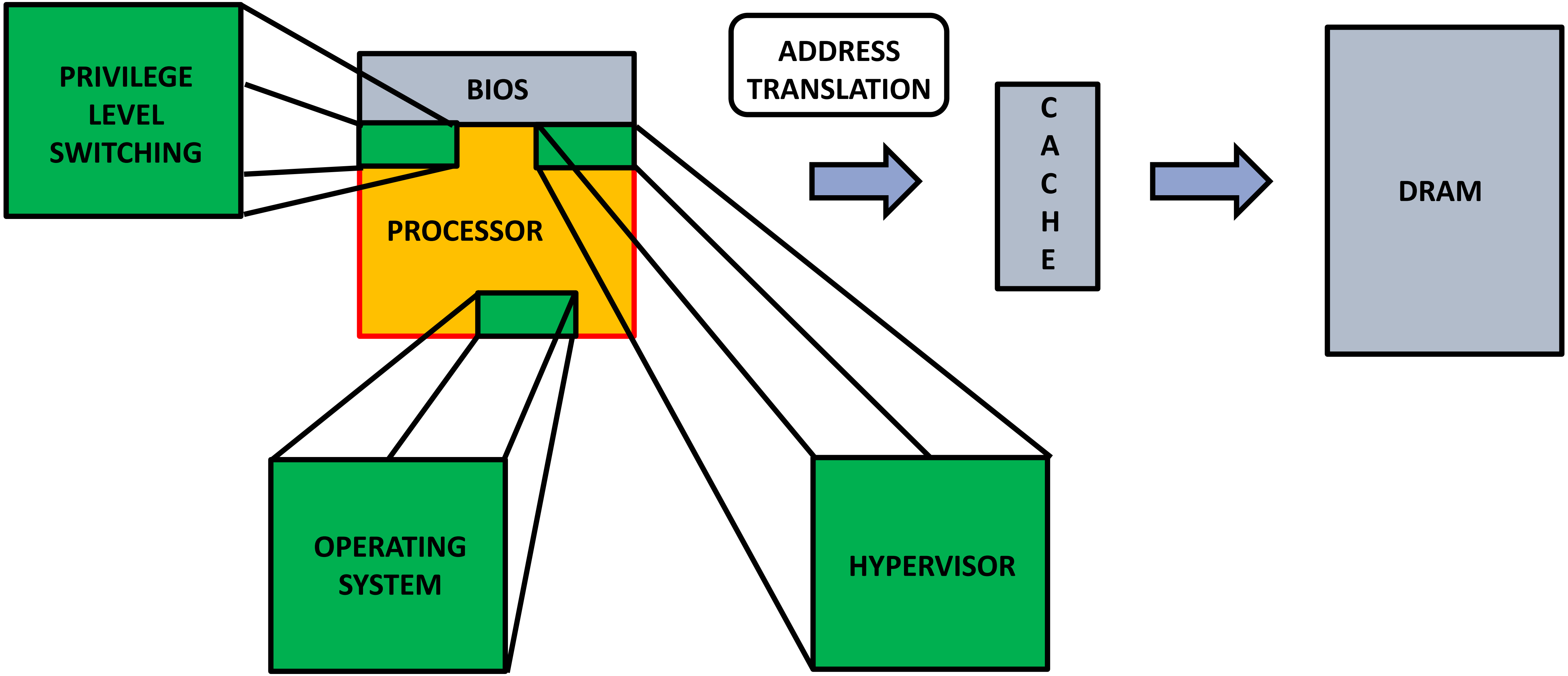
VMX Privilege Level Switching:

- **Remember!** Hypervisor uses virtualization to run multiple operating systems at the same time and manages the Virtual Machines **(VM).**
  - Hypervisor creates a control structure **(VMCS)** for each operating system instance that it wishes to run.
  - When a processor encounters an exception, hypervisor performs a **VMEXIT** for the fault to be handled.
  - Example, if a address translation encounter an entry in the page table with P flag set to 0. The CPU performs the exit for the page to be loaded in the table. When done, CPU calls a **VMLAUNCH** or **VMRESUME** function to resume the operation.



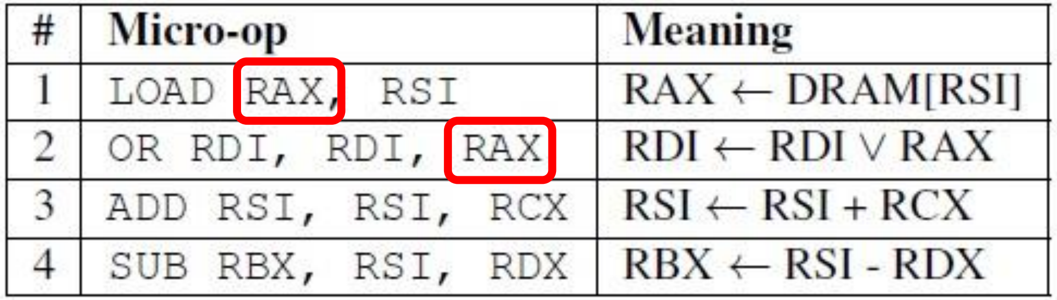
SECTIONS

- Overview.
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- Execution contexts.
- Segment Registers.
- Privilege level switching.
- **OOO AND SPECULATIVE EXECUTION.**
- Caches.


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

OUT OF ORDER + SPECULATIVE EXECUTION

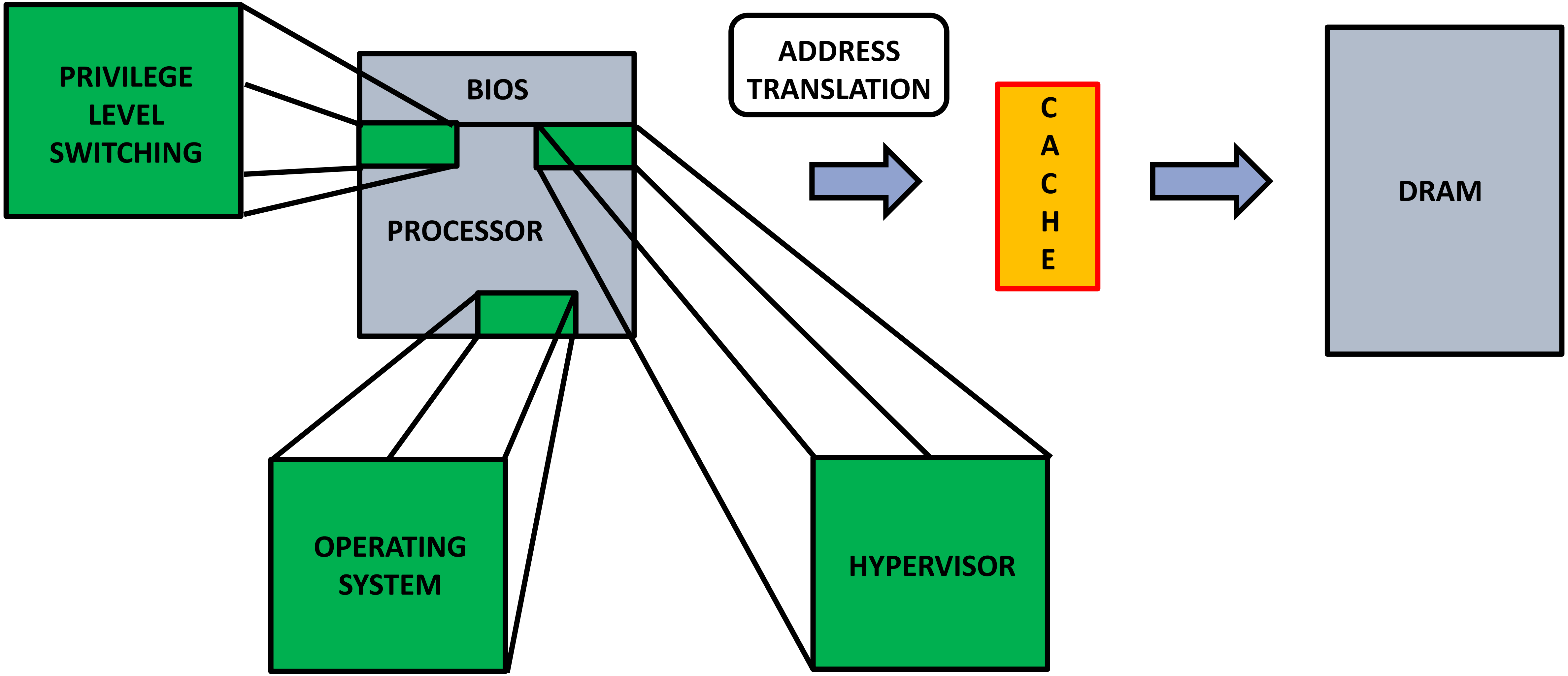
- Execution is not in order, but commits are always in order using ROB.
- Any CPU actions observed by an attacker match the execution order, so the attacker may learn some information by comparing the observed execution order with a known program order.
- Scoreboard and Tomosulo schedulers.



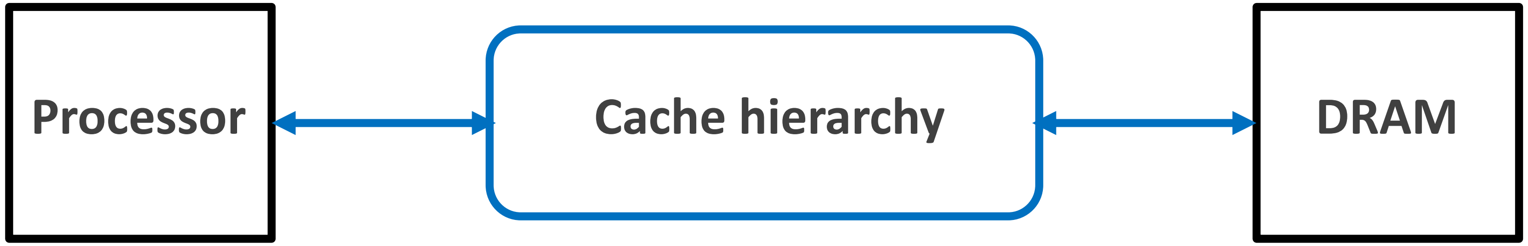
#	Micro-op	Meaning
1	LOAD RAX, RSI	RAX ← DRAM[RSI]
2	OR RDI, RDI, RAX	RDI ← RDI ∨ RAX
3	ADD RSI, RSI, RCX	RSI ← RSI + RCX
4	SUB RBX, RSI, RDX	RBX ← RSI - RDX

SECTIONS

- Overview.
- Computational Model.
- Software Privilege Levels.
- Address Translation.
- Execution contexts.
- Segment Registers.
- Privilege level switching.
- OOO and speculative execution.
- **CACHES.**


PRIVILEGE LEVEL SWITCHING
BIOS
PROCESSOR
ADDRESS TRANSLATION
CACHE
DRAM
OPERATING SYSTEM
HYPERVISOR

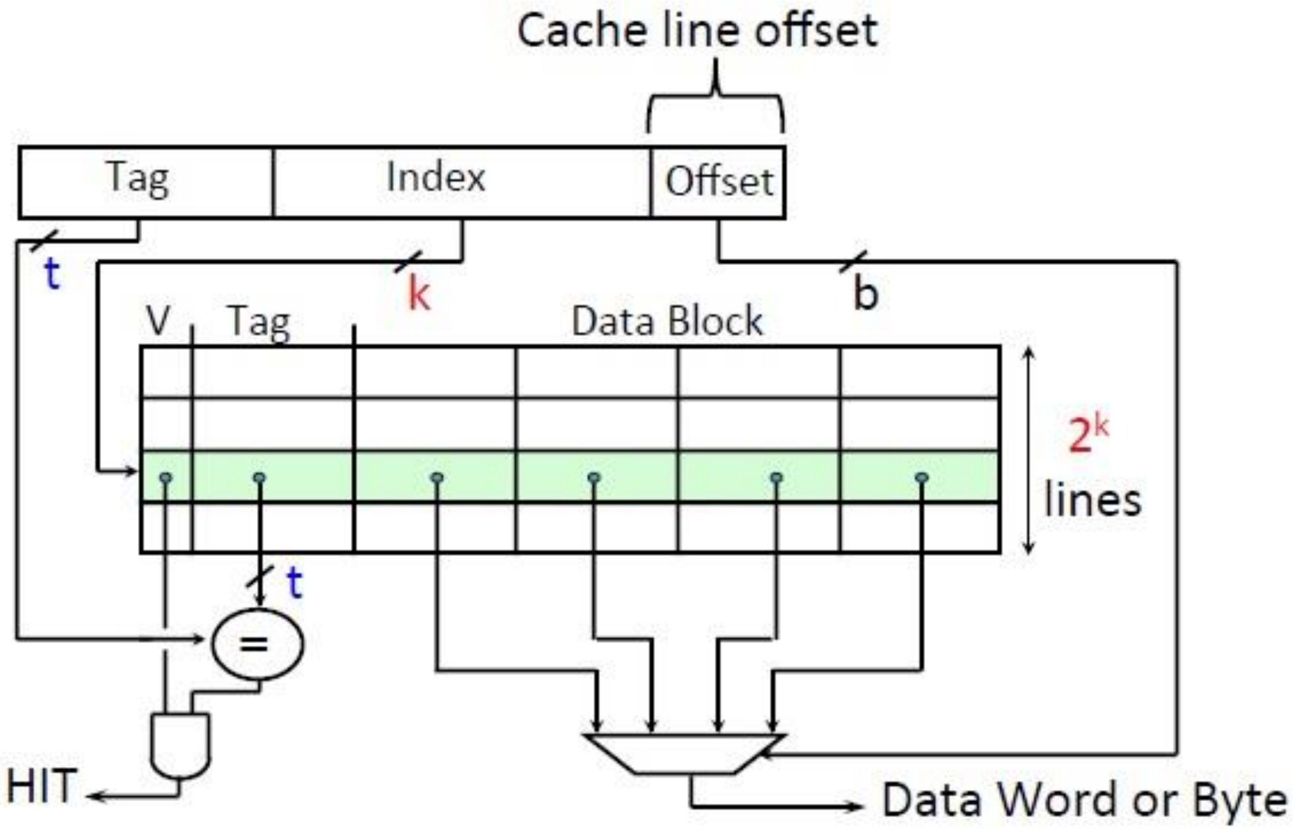
CACHES AND COHERENCE



- Types of Caches
  - Direct Mapped
  - 2, 4 - Way Associative
  - Fully Associative

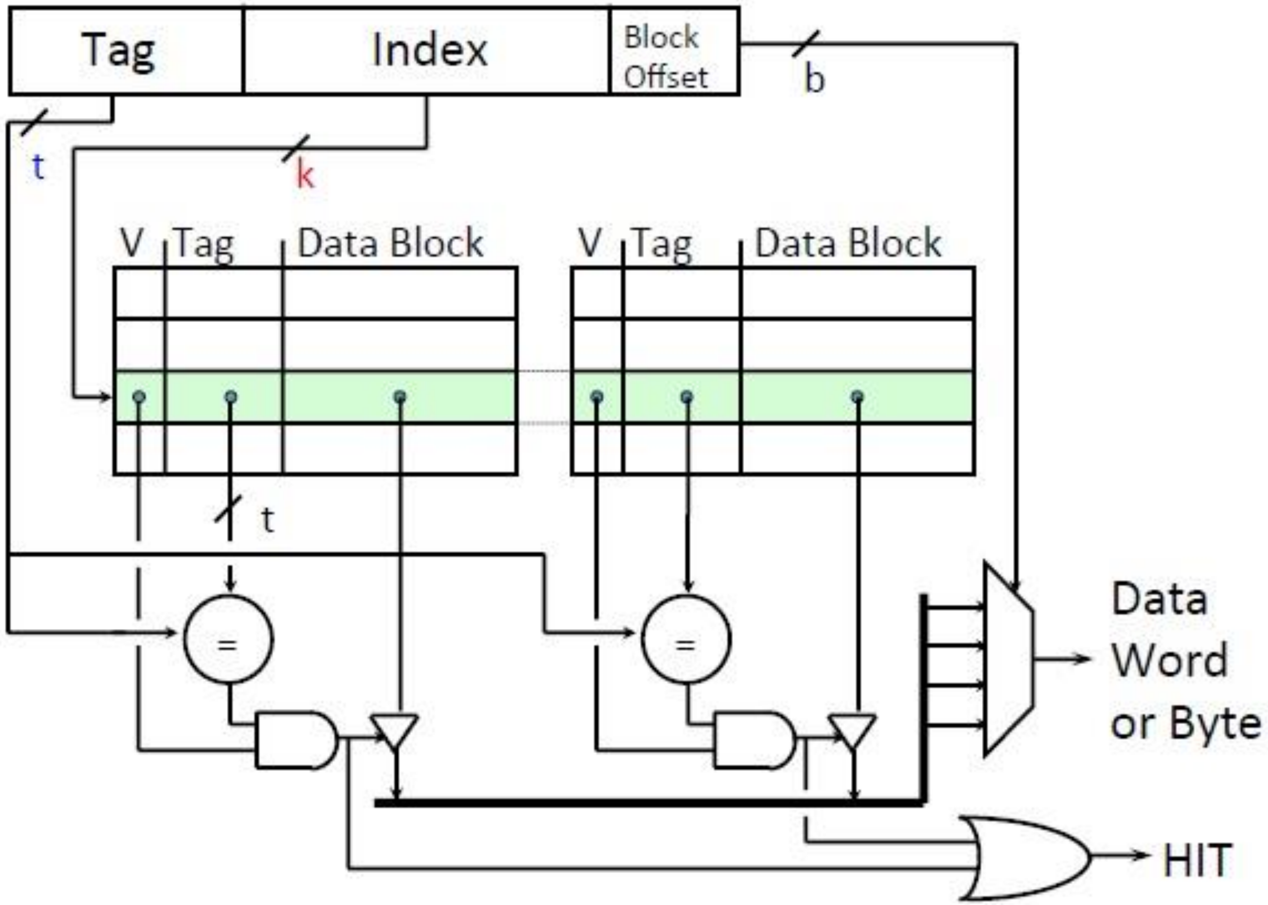
CACHES AND COHERENCE

Direct Mapped Cache

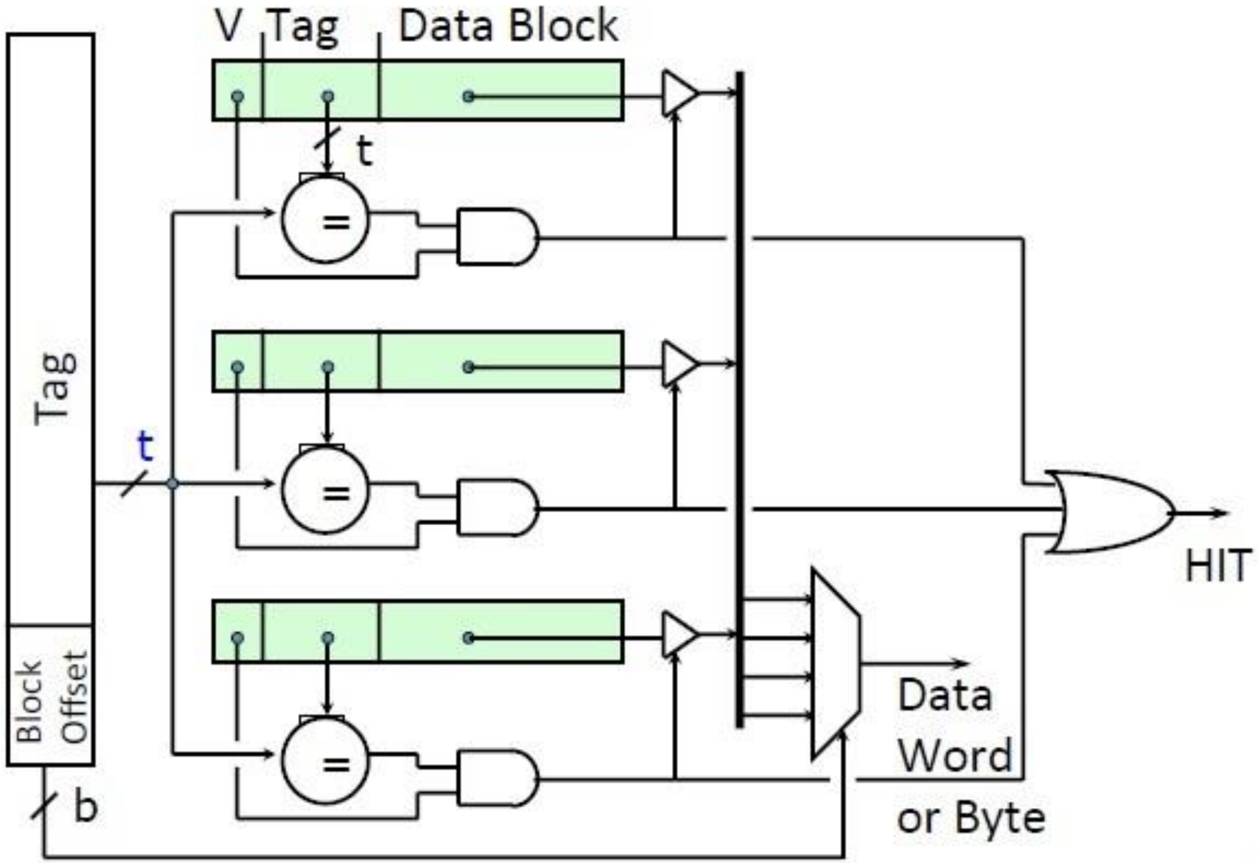


CACHES AND COHERENCE

2-Way Set Associative Cache



CACHES AND COHERENCE
Fully Associative Cache



CACHES AND COHERENCE
Cache Coherence

- Example, core 0 and core 1 both have a value ‘A = 1’ in their private caches. If core 0 updates the value of A to 5. How will core 1 know that the value of A has been changed ?
- Ensuring correctness and freshness of cache lines for each processor requires a protocol, known as cache coherence protocol.
- Cache coherence protocol introduces different states for each cache line. In MSI protocol, these states are **Modified (M), Shared (S),** and **Invalid (I).**
- When in ‘M’ state, all other cores invalidate their cache lines.
- When in ‘I’ state, the core asks for the most recently updates value. After getting it, that core comes to ‘S’ state.
- Multiple reads but single write is allowed. Meaning more than one core can be in ‘S’ state at the same time but only one core at a time can be in ‘M’ state.

A MONSTER
BIG
THANKYOU
X