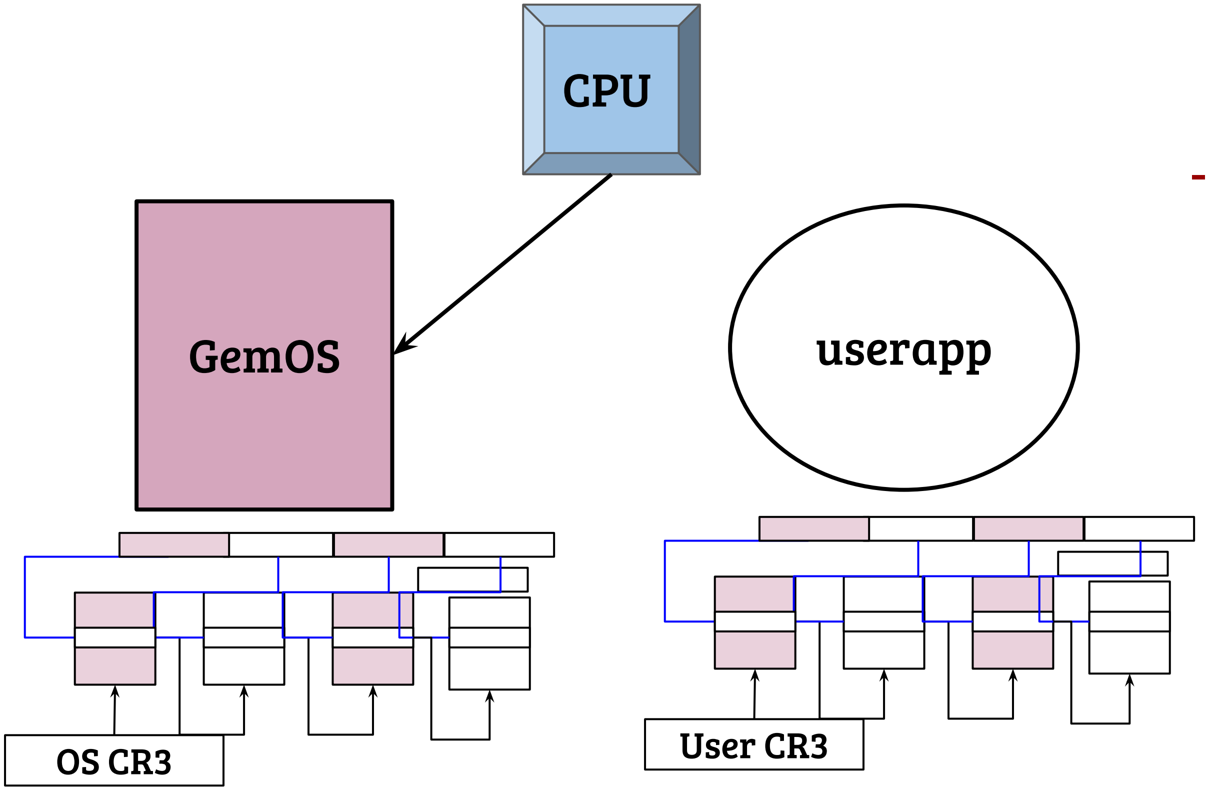
Operating Systems

User contexts and system calls

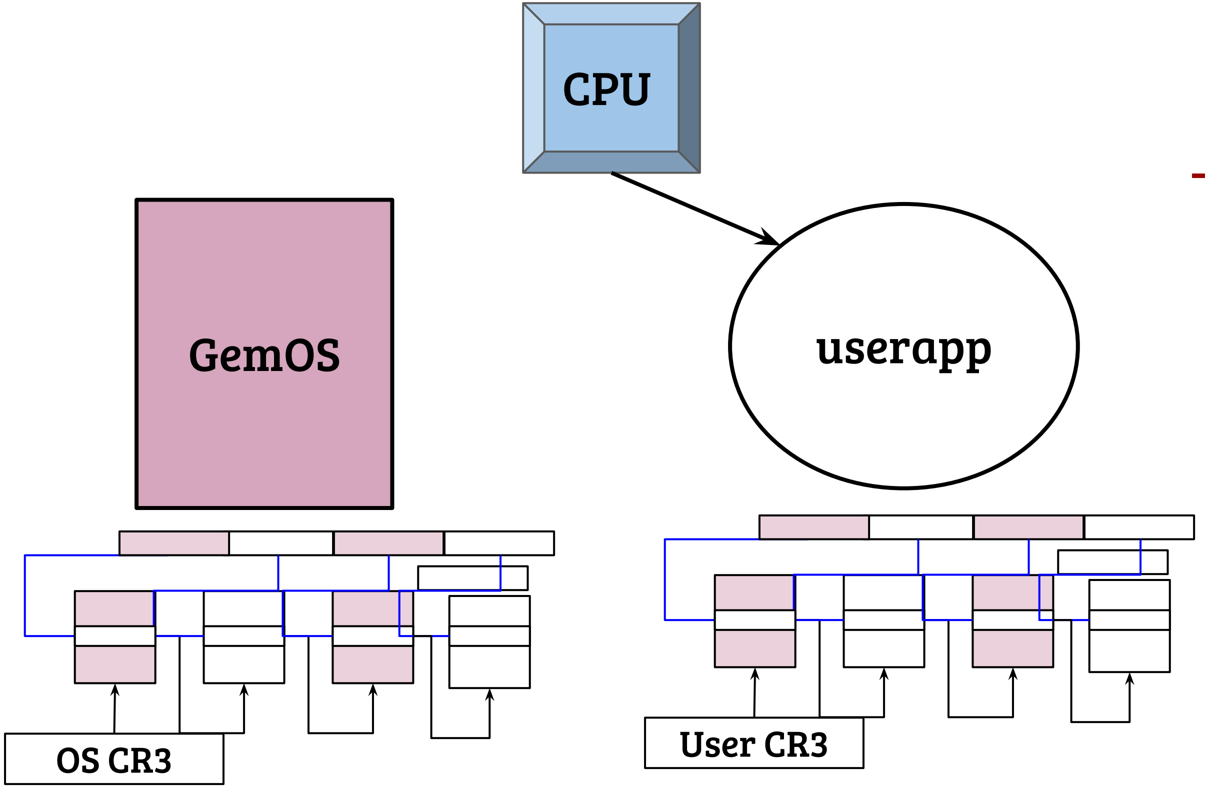
Debadatta Mishra, CSE, IITK

Mysteries of assignment-1



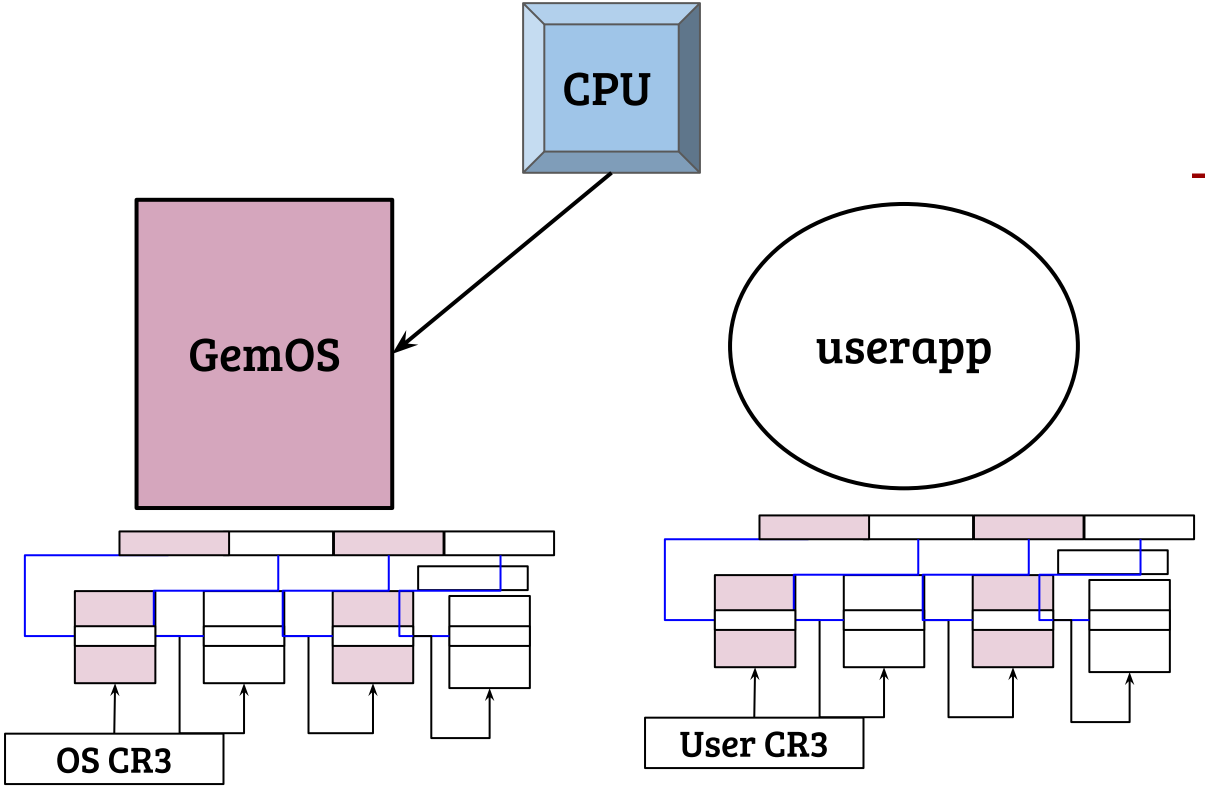
- Setup the page table for “userapp” while in GemOS context

Mysteries of assignment-1



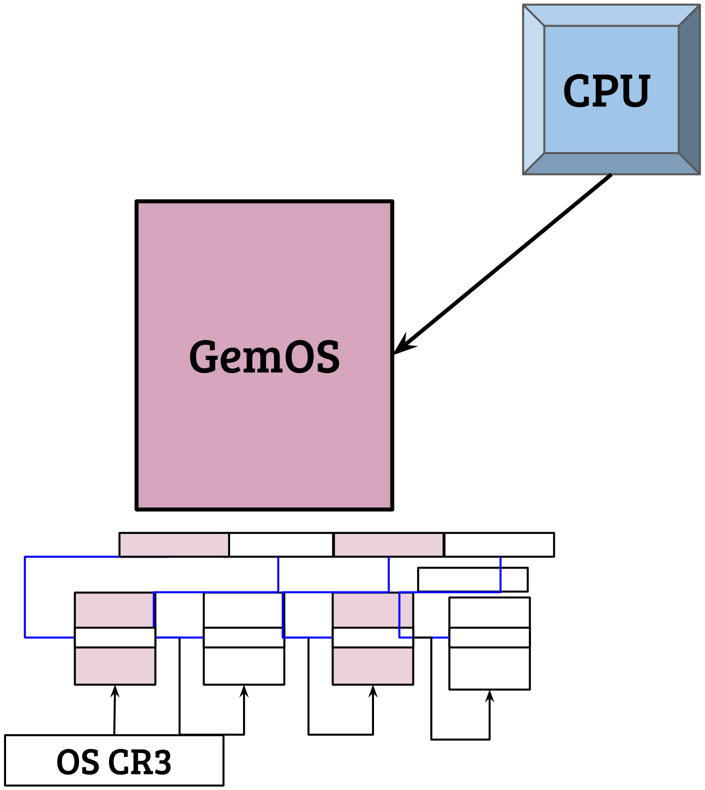
- CPU executes “userapp”, how this switch happened?

Mysteries of assignment-1



- CPU executes GemOS context, How it returned?

Mysteries of assignment-1



- Cleanup “userapp”, you know how!

Summary of hacks

- ➔ “userapp” is not executing in user mode (ring-3)
- ➔ Return address (of GemOS) is placed into the stack of “userapp”
- ➔ CR3 switched on launch and return
- ➔ Summary: Execute a function using a different page table layout

User context

- Executes in user mode with lower privileges
- Should not crash the system!
  - Due to buggy code
  - Accessing sensitive resources (like CR3)
- Can use OS services to
  - Carry out sensitive operations (expand memory)
  - Operate on I/O devices

Assignment-2 (will be out shortly)

- This time, it is a user mode context
- Some syscalls are implemented, you will implement some
- Exception handling in OS
    - Floating point exception
    - Page fault exception

What is a process?

- Classical definition: *A program in execution is called a process*
  - LOAD, EXECUTE, EXIT
- Program is persistent while process is volatile
  - Program is identified by an executable, process by PID
- Program → Process   (1 to N)
  - Many concurrent processes can execute the same program

What is a process?

- ➔ *A program in execution is called a process*
  - ◆ LOAD (program) → Process, EXECUTE (Process), EXIT (Process)
- ➔ Program is persistent while process is volatile
  - ◆ Program is identified by an executable, process by PID
- ➔ Program → Process   (1 to N)
  - ◆ Many concurrent processes can execute the same program

- ➔ Can we call every execution entity a process?
- ➔ What are the hardware and software states of a process?

CPU perspective of an execution context (revisited)

“A task is a unit of work that a processor can dispatch, execute, and suspend. It can be used to execute a program, a task or process, an operating-system service utility, an interrupt or exception handler, or a kernel or executive utility. ”

---Intel Software Developer Manual 3A, Ch7

Important aspects of an execution context

- State of GPRs
- State of FLAGS, RIP → Current execution state
- CR3 → Memory partitioning information
- Current execution space (CS, SS, DS) → Defines privilege level
- Stack pointers for ring (0 - 2) → useful when privilege level changes
    - Why change RSP when switch from user to OS?

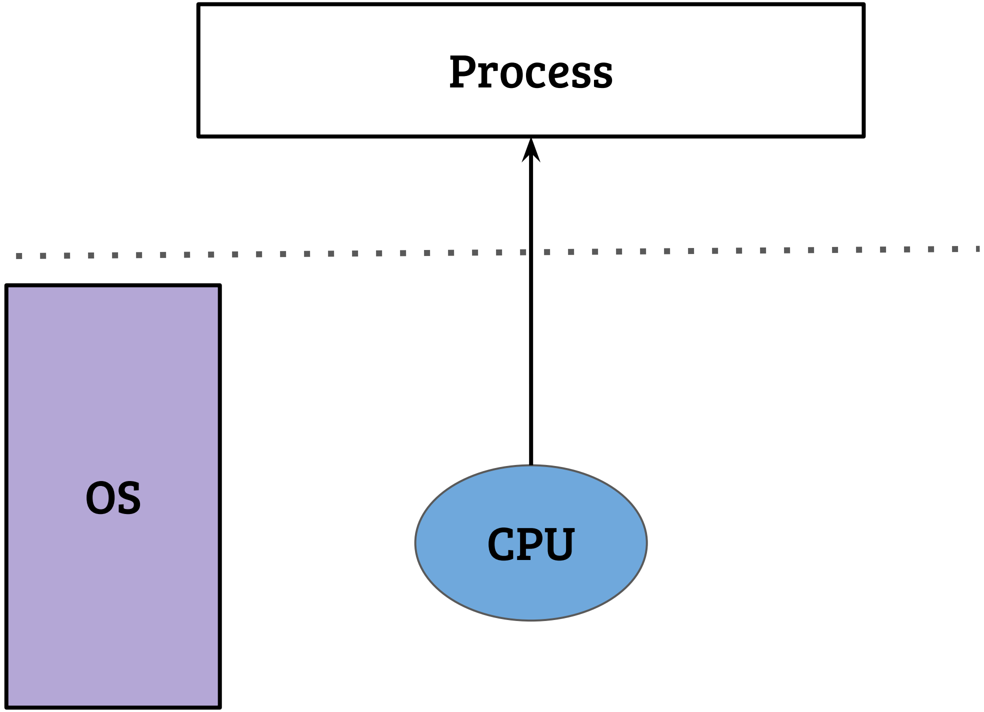
Process and hardware context

- GPRs including stack pointer for ring-3
- Flags
- Instruction pointer
- Memory partitioning information, privilege
    - CR3, Segment registers
- Stack pointer for ring-0

Software state (in OS)

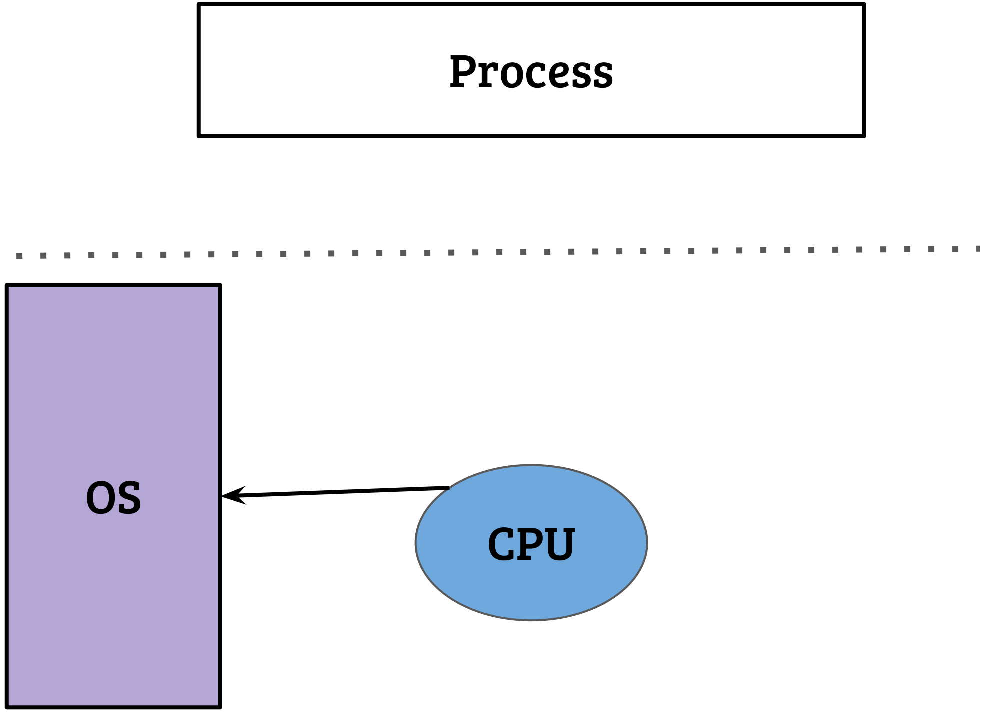
- A restorable copy of hardware state
- Process ID
- Process execution state
- Memory information
- Open files
- ….
- Different OSs name it differently
    - Process control block
    - Task

System call and exceptions



What happens when the process executes an instruction for system call or an exception occurs?

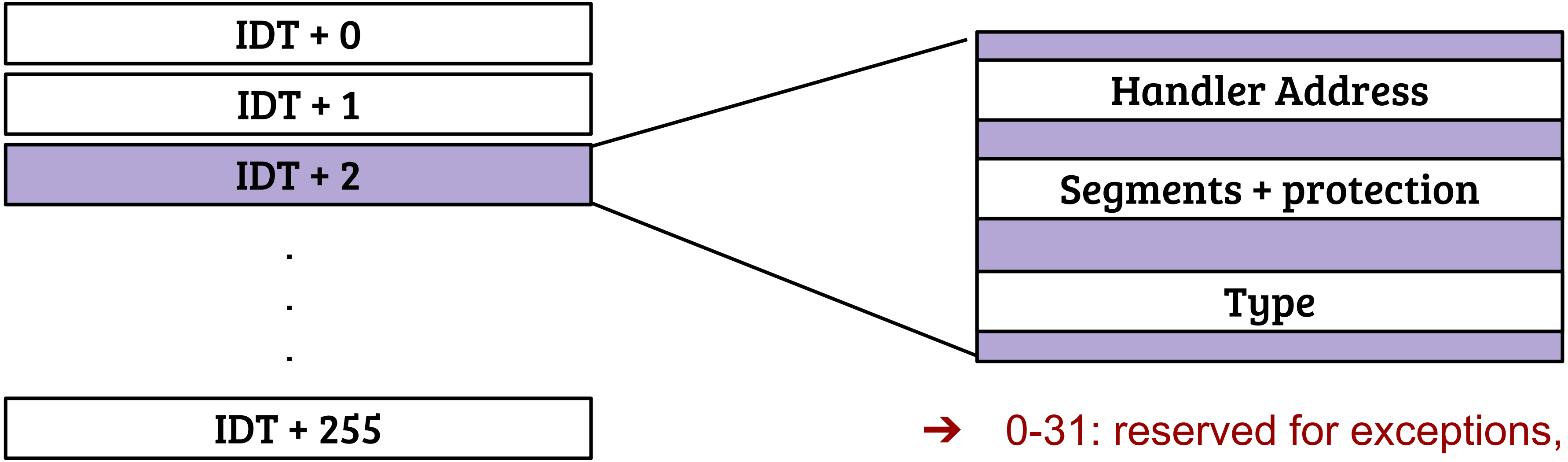
System call and exceptions



CPU starts executing "registered " system call/exception handler.

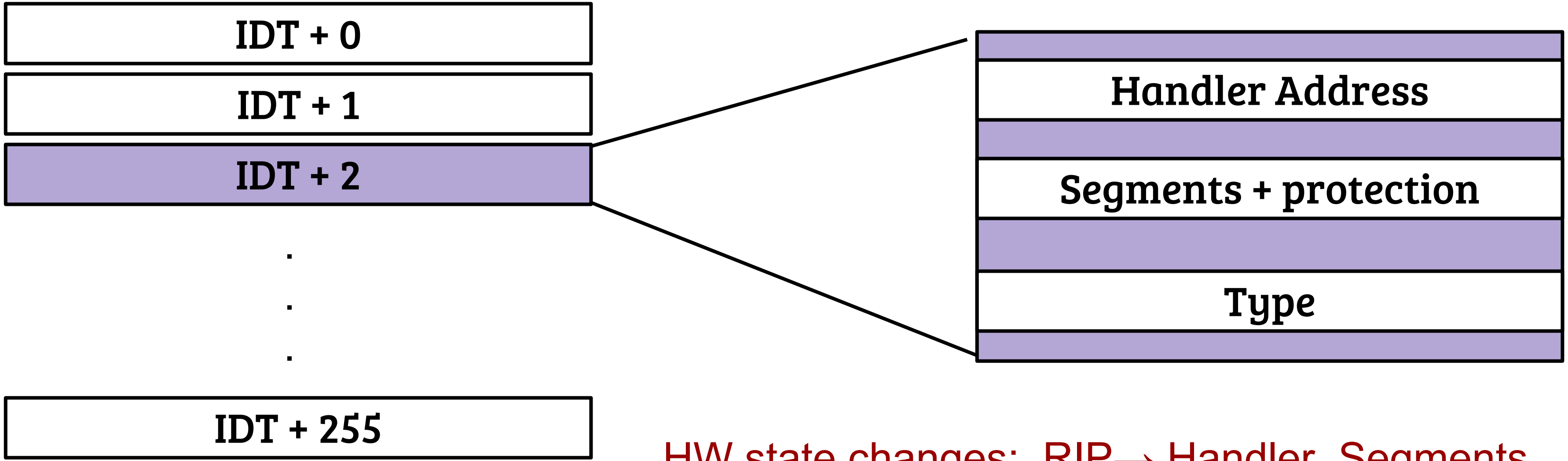
- ➔ How the handler is registered?
- ➔ What are the hardware context changes?
- ➔ What are the OS and hardware responsibilities?

X86 IDT: gateway to handlers



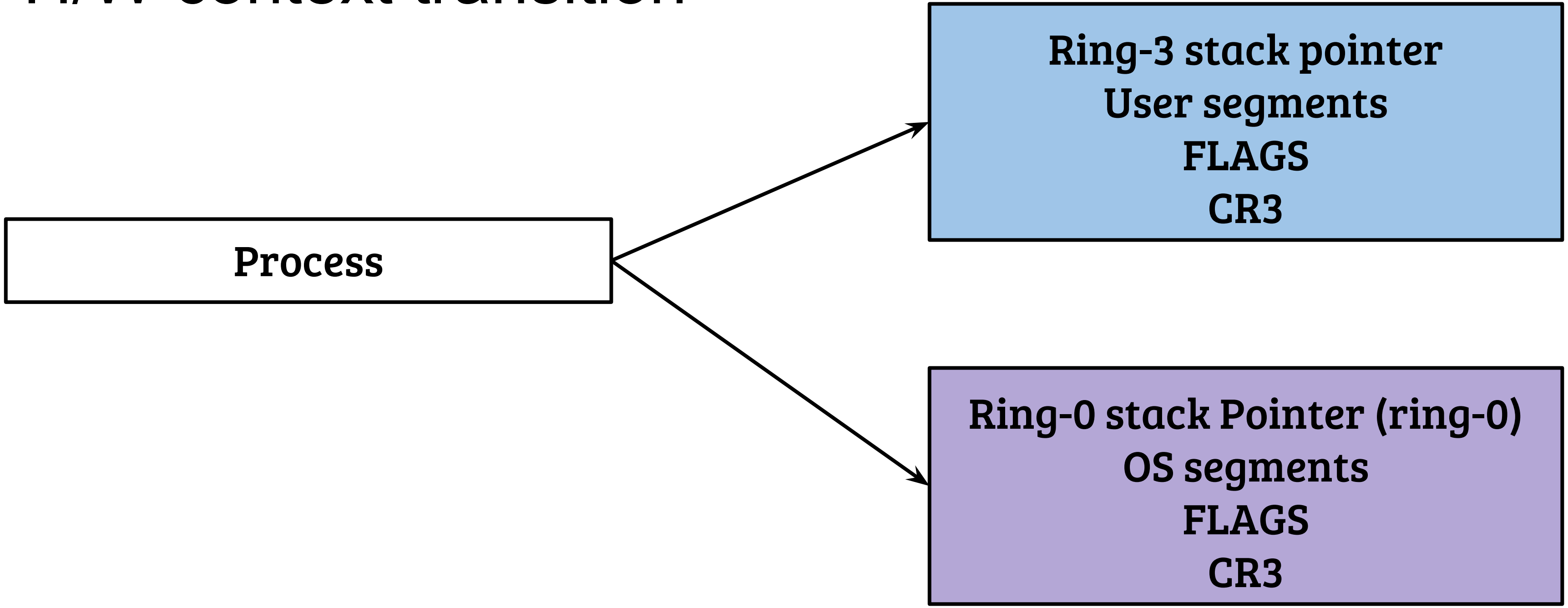
- ➔ 0-31: reserved for exceptions, traps. e.g., PF=14, DE=0
- ➔ 32-255: user defined, used for h/w and s/w interrupt handling

X86 IDT: gateway to handlers



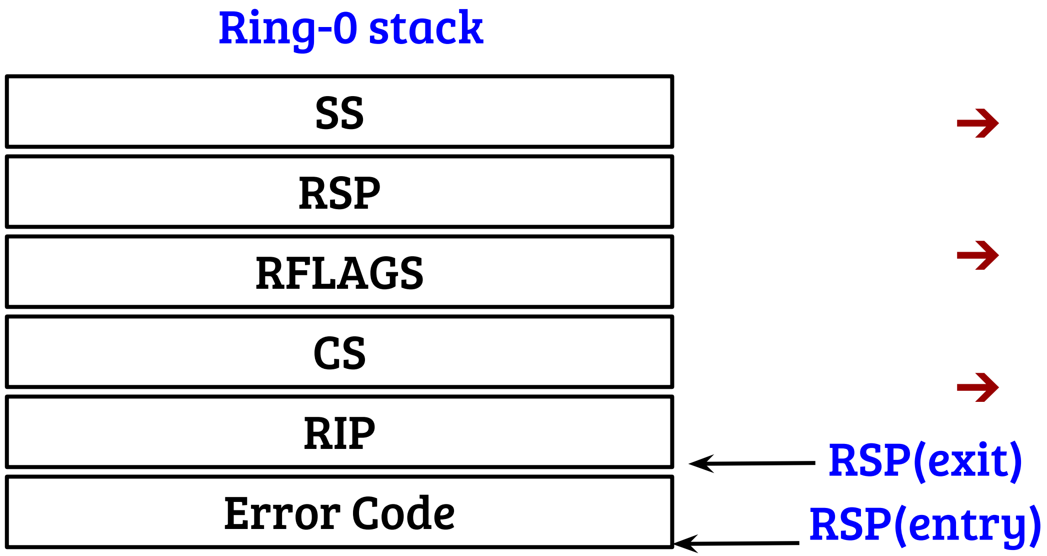
HW state changes: RIP→ Handler, Segments loaded to seg. registers, Stack pointer changed to ring-0 stack

H/W context transition



- ➔ CR3 to be switched (if at all) by the handler
- ➔ How restored after handling the fault/exception?

H/W context transition → restoration (X86_64)



Ring-0 stack

SS

RSP

RFLAGS

CS

RIP

Error Code

RSP(exit)

RSP(entry)

- ➔ At entry, the kernel stack is as shown
- ➔ At return, the RSP must be placed at saved RIP address on stack
- ➔ Invoke IRETQ to return to user space

System calls

- ➔ Classic system call implementation is a user defined entry in IDT, at IDT + 0x80
- ➔ How parameters to syscall are passed?
    - ◆ Registers
    - ◆ Stack

System calls

- ➔ Classic system call implementation is a user defined entry in IDT, at IDT + 0x80
- ➔ How parameters to syscall are passed?
  - ◆ Registers
  - ◆ Stack
- ➔ How user level pointers are accessed?
  - ◆ Using user page tables vs. OS page tables

GemOS system calls

- ➔ In GemOS, we also have registered a generic handler for all syscalls
  - ◆ Syscall number is the first parameter
  - ◆ All parameters are passed in X86 calling conventions
  - ◆ Return value is stored in EAX
- ➔ Already implemented
  - ◆ getpid( ), exit( )
- ➔ In GemOS, CR3 is not switched → convenient
- ➔ By implication → OS V to P mapping is present in each process page table