ECE437/CS481

INTRODUCTION TO OS
COMPUTER ARCHITECTURE SUPPORT

Xiang Sun

The University of New Mexico

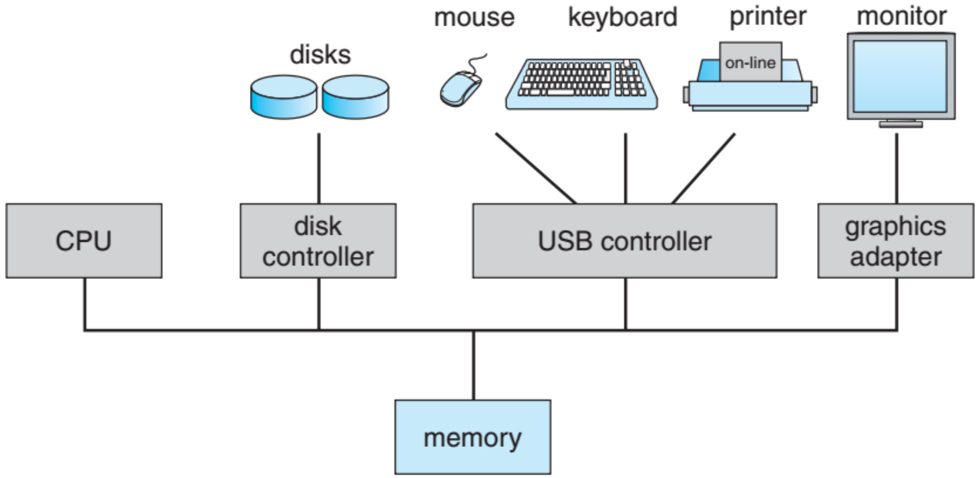
❑ I/O devices

➢ Normally, an I/O device consists of a mechanical component and an electronic component, where the electronic component is called the device controller.

- ✓ Each device controller runs a specific firmware.
- ✓ A device controller is in charge of a particular device type.
- ✓ Act as an interface/intermediary between a device and a device driver.

➢ With help of device controller, CPU can conduct asynchronous I/O.

- ✓ Synchronous I/O: CPU execution waits while I/O proceeds.
- ✓ Asynchronous I/O: I/O proceeds concurrently with CPU execution.

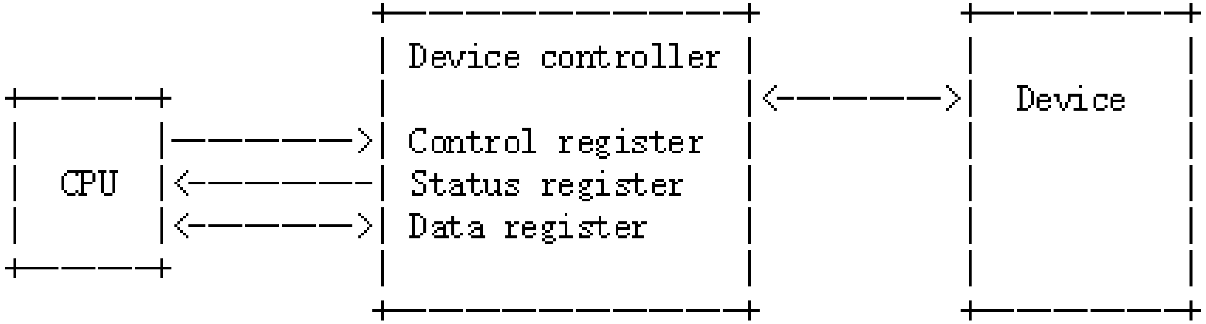


I/O Devices & OS

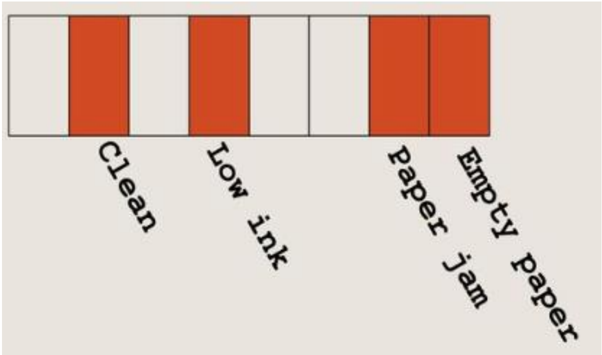
❑ Different types of I/O devices

- Character device
  - ✓ sending and/or receiving single characters
  - ✓ typical devices: mouse, keyboard, printer
- Block device
  - ✓ sending and receiving entire blocks of data
  - ✓ typical devices: disk drives, flash memory
- Network device
  - ✓ sending and receiving packets
  - ✓ typical devices: network card

❑ Three types of registers in a device controller



- **Status register**
  - ✓ provides status information (e.g., busy, ready)
  - ✓ read-only



- **Configuration and control registers**
  - ✓ enables CPU to configure and control the device
  - ✓ bits in a configuration register may be write-only
  - ✓ bits in a control register can be both read and written

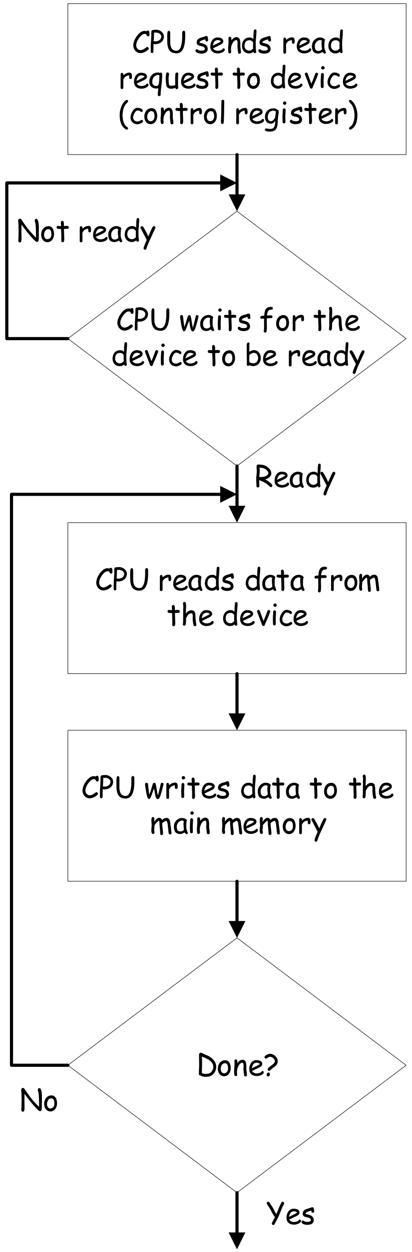
- **Data registers**
  - ✓ read data from or send data to the I/O device

- Three ways of managing data transfers
  - Programmed I/O (PIO)
  - Interrupt-driven I/O
  - Direct memory access (DMA)

❑ Programmed I/O

---Assume that the CPU tries to transfer data from the hard disk to the main memory

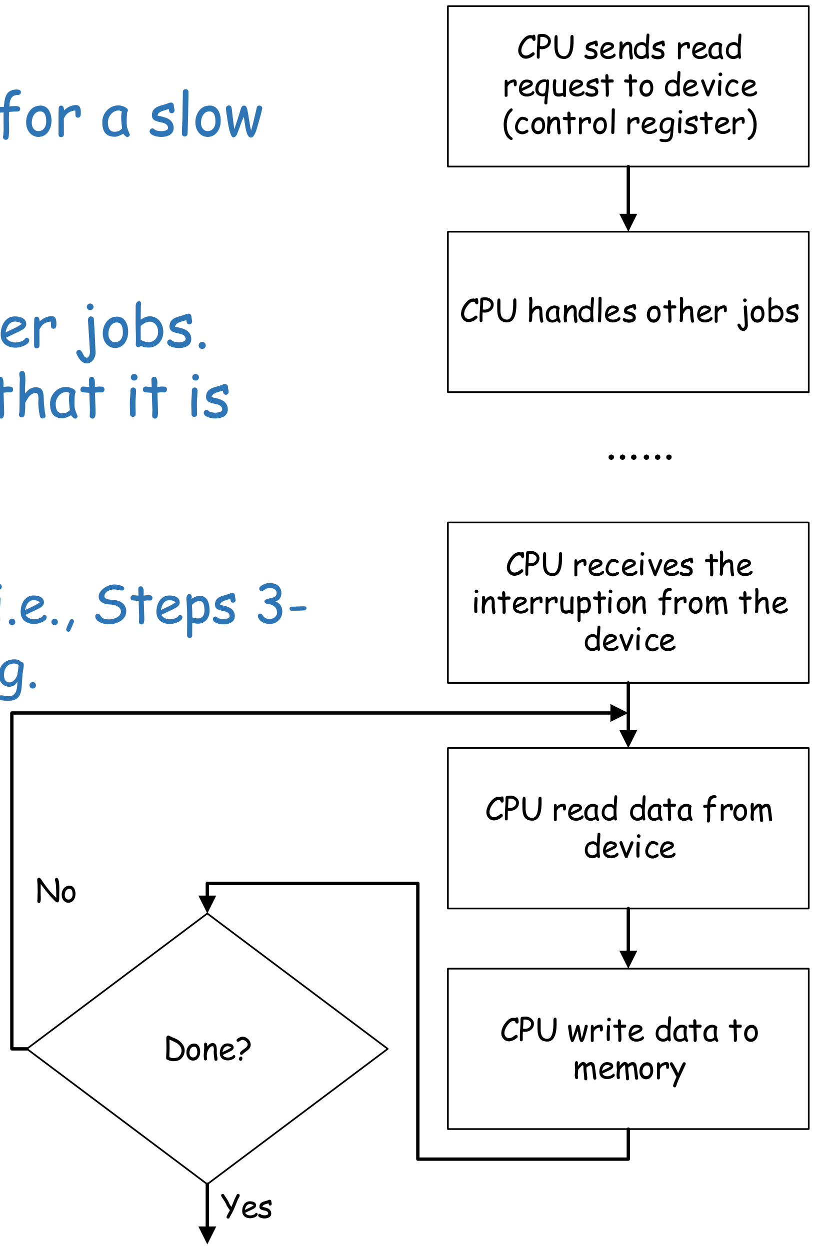
1) The CPU issues a request to the control register of the hard disk.
2) The CPU waits the hard disk to become ready (e.g., to move the disk head to the right place).
   - The CPU continuously checks the status register of the hard disk to see if it is ready-----polling
   - The CPU cannot do anything but wait
3) The CPU reads data from the hard disk.
4) The CPU writes data to the memory.
5) Step 3&4 continuous until all the data are transferred to the memory.



- Interrupt-driven I/O
---Solving the problem of the processor having to wait for a slow device to be ready.

1) Instead of waiting, the CPU continues to handle other jobs. The hard disk send an interrupt to inform the CPU that it is ready.

2) The data transfer is still the same as programmed I/O (i.e., Steps 3-5) --- The CPU still has to involve in the data transferring.



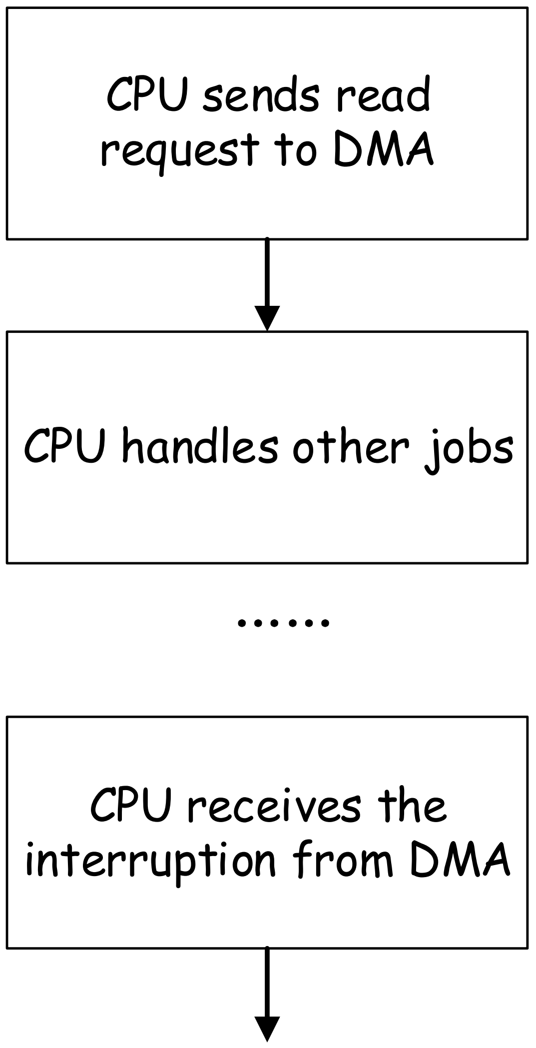
- Direct memory access (DMA)
  ---benefit the CPU for bulk data transfer

  - DMA is a simple controller which does most of the functions (related to data transferring) that the CPU would otherwise have to handle.

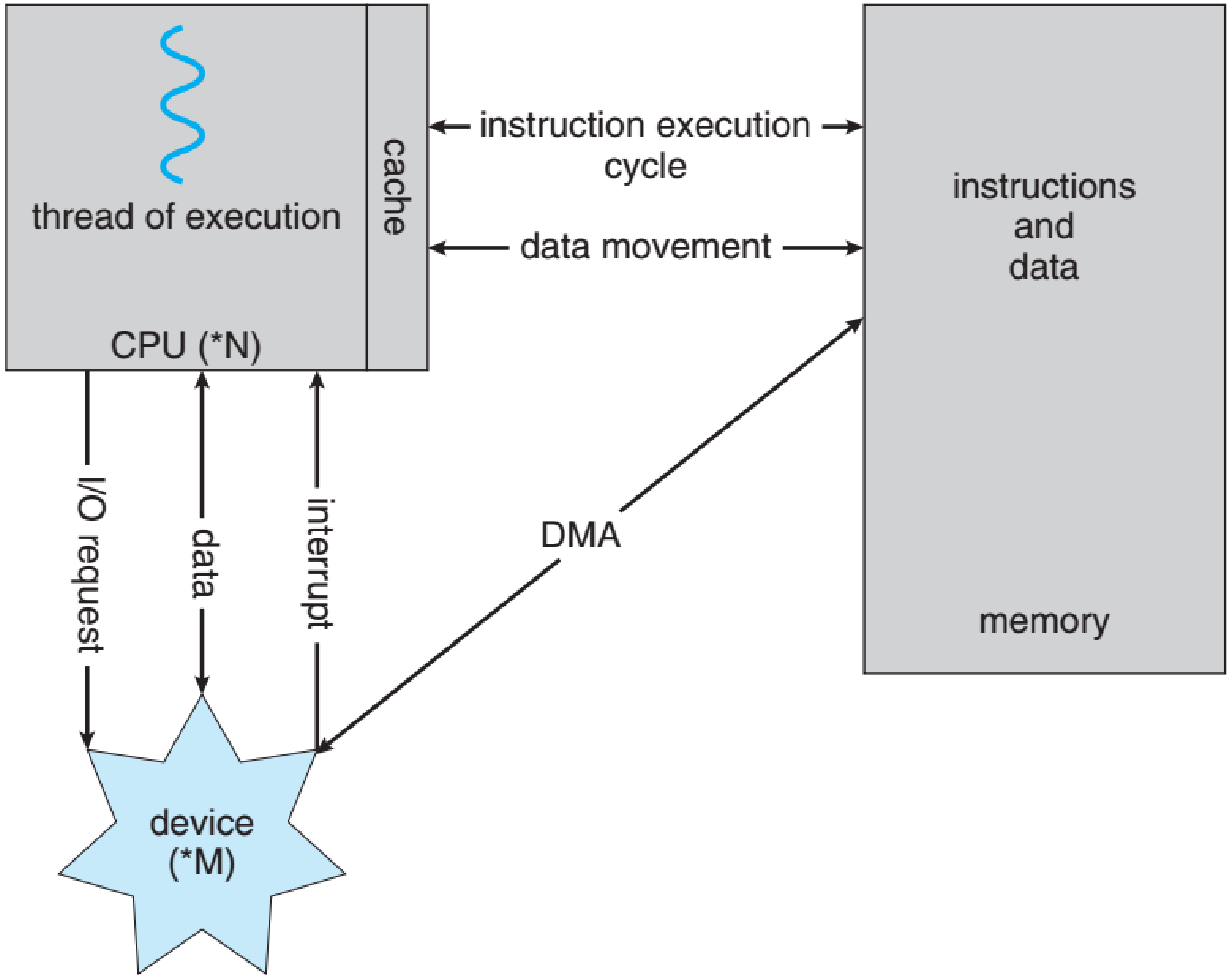
1) The CPU asks the DMA controller to transfer data between the hard disk and the memory. After that, the CPU continues to handle other jobs.

2) The DMA controller manages the data transfer between the hard disk and memory.

3) Once finished, the DMA controller sends an interrupt.



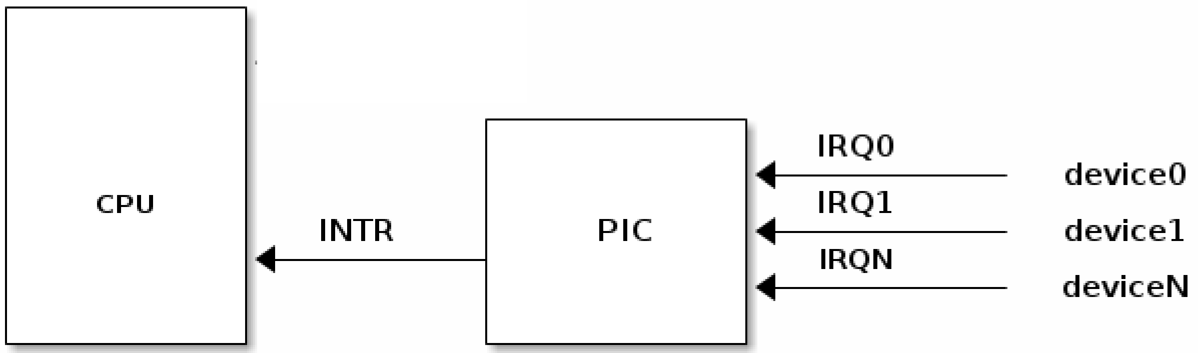
- Interplay among CPU, memory, and I/O devices



Interrupts in OS

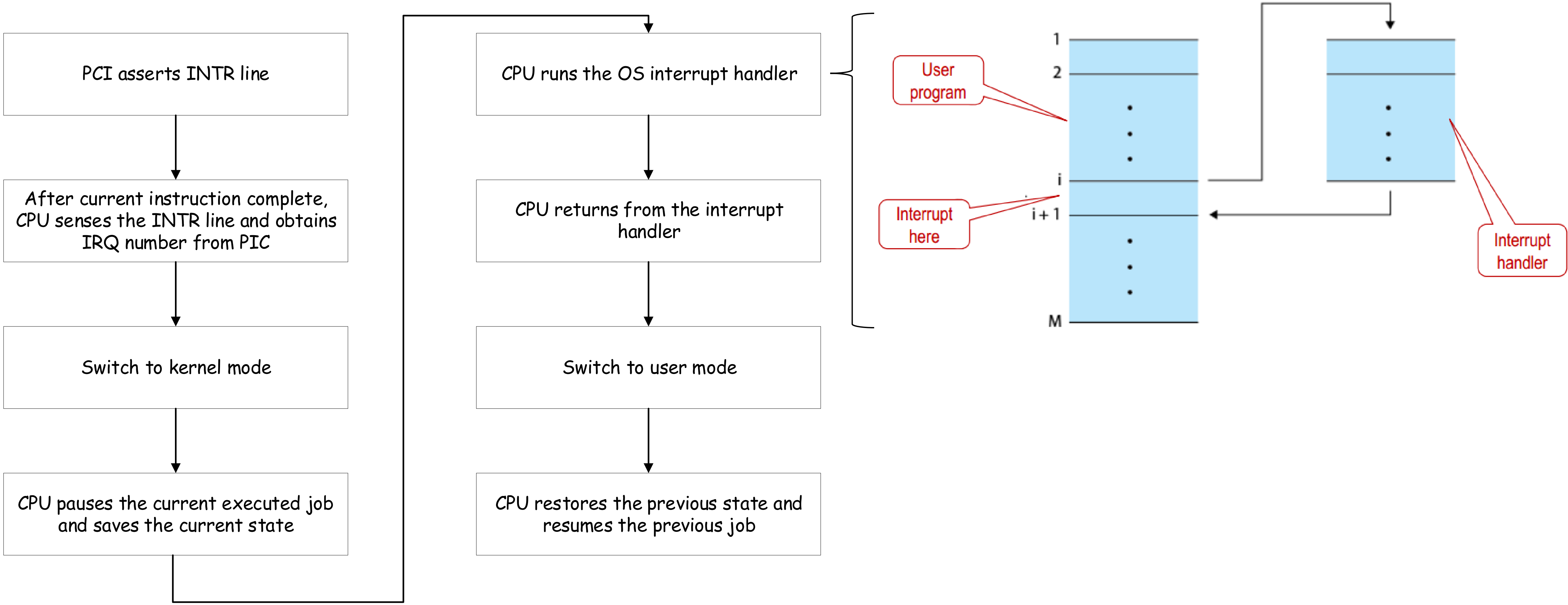
❑ Interrupt handling process

- I/O devices are connected to a Programmable Interrupt Controller (PIC) via a dedicated interrupt request (IRQ) line.



- When an I/O device asserts its IRQ line, PIC will know which device sends the interrupt. Accordingly, PIC will assert the INTR line to inform CPU.

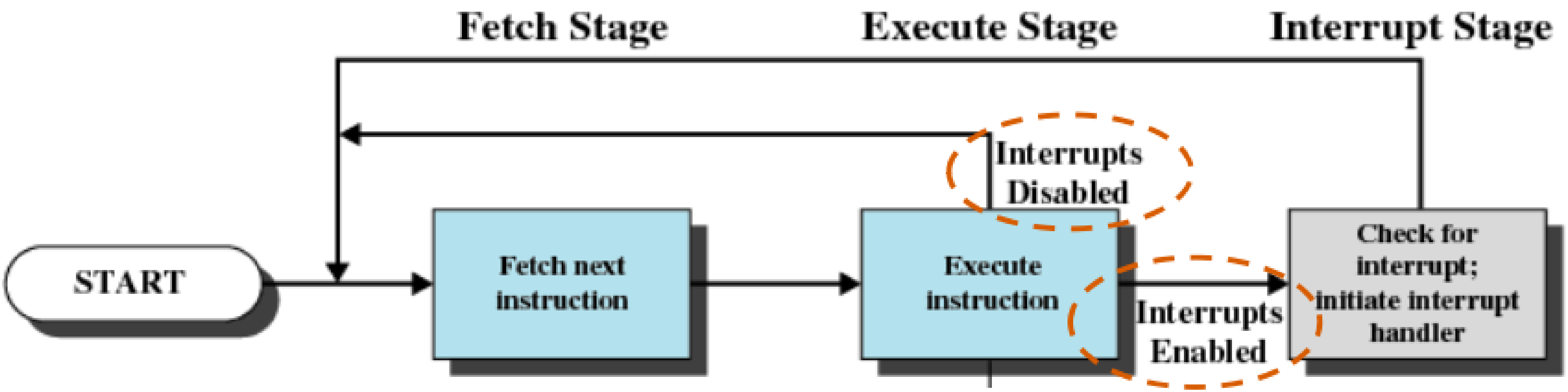
❑ Interrupt handling process



Interrupts in OS

- Interrupts
  - Interrupts can be generated by
    - ✓ Hardware: I/O devices (e.g., click of mice), a timer, etc.
    - ✓ Software: incurring errors and exception (e.g., divide by zero, arithmetic overflow). Note that a software-generated interrupt is also called a trap.
  - Interrupts have different priorities
    - ✓ If two interrupts arrives at the PIC simultaneously, the PIC will forward the interrupt with a higher priority.

- Interrupts
  - When the CPU is handling an interrupt
    - ✓ Other incoming interrupts can be disabled, waiting in a queue, i.e., non-preemptive.
    - ✓ Other incoming interrupts can be enabled, interrupting the executed interrupt, i.e., preemptive.



Interrupts in OS

- Interrupt handler vs function call
  - Function call: the instructions are executed from a new address.

Interrupt Handler	Function Call
Random	Expected or User Programmed
Execution based on priority	Execution based on sequential order
Cannot have arguments and return values	Can have arguments and return values

```c
int add(int a, int b) {
    return a + b;
}

int main() {
    int num1 = 10;
    int num2 = 5;

    // Call the add function
    int sum = add(num1, num2);

    // Print the result
    printf("Sum: %d\n", sum);

    return 0;
}
```

```c
void send_EOI_to_PIC() {
    // Typically, you would write to a specific port to signal End of Interrupt (EOI)
    printf("End of Interrupt (EOI) signal sent to PIC.\n");
}

// Interrupt handler for keyboard input (simplified)
void keyboard_interrupt_handler() {
    // Simulate reading the keycode from the keyboard data port
    uint8_t keycode = read_from_keyboard_port();

    // Process the keycode (for example, converting it to an ASCII character)
    char ascii_char = convert_keycode_to_ascii(keycode);

    // Print the key pressed (for demonstration purposes)
    printf("Key pressed: %c\n", ascii_char);

    // Send End of Interrupt (EOI) signal to the PIC
    send_EOI_to_PIC();
}
```

System calls in OS

- System calls

  - A system call is a way in which a user program requests a service provided by the kernel of the OS.

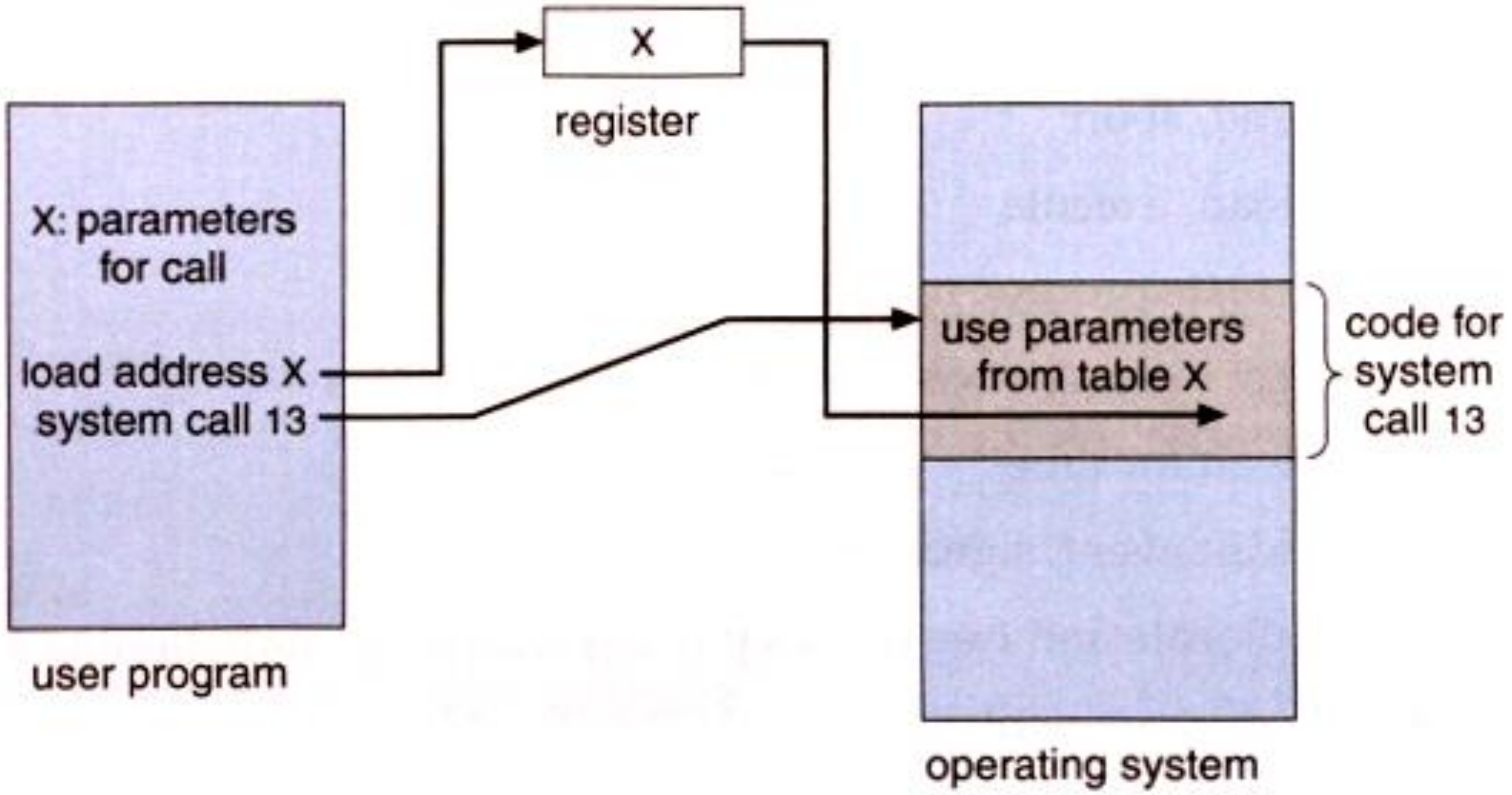
  - Types of System Calls
    - ✓ Process control: end, abort, create, terminate, allocate and free memory.
    - ✓ File management: create, open, close, delete, read file etc.
    - ✓ Device management
    - ✓ Information maintenance
    - ✓ Communication
    - ✓ Protection

Kernel Mode VS. User Mode

❑ System calls

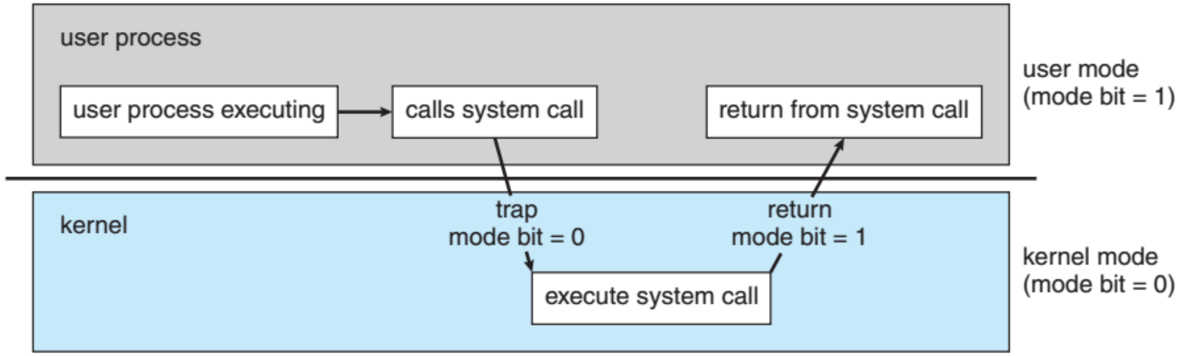
- Parameters of a system call can be passed via registers.

- If there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.



Kernel Mode VS. User Mode

- kernel mode and user mode
  - There are two modes of operations, i.e., kernel mode and user mode.
  - In user mode, the CPU is running a user's job in the memory.
  - In kernel mode, the CPU is running an interrupt handler or system call in the memory.
  - The transition from user mode to kernel mode occurs when an interrupt or system call occurs. The transition from kernel mode to user mode when the interrupt or system call is completed.
    - ✓ Mode bit is used to indicate if the process/job is in kernel mode (0) or user mode (1). This mode bit is stored in the Program Status Word (PSW) register.

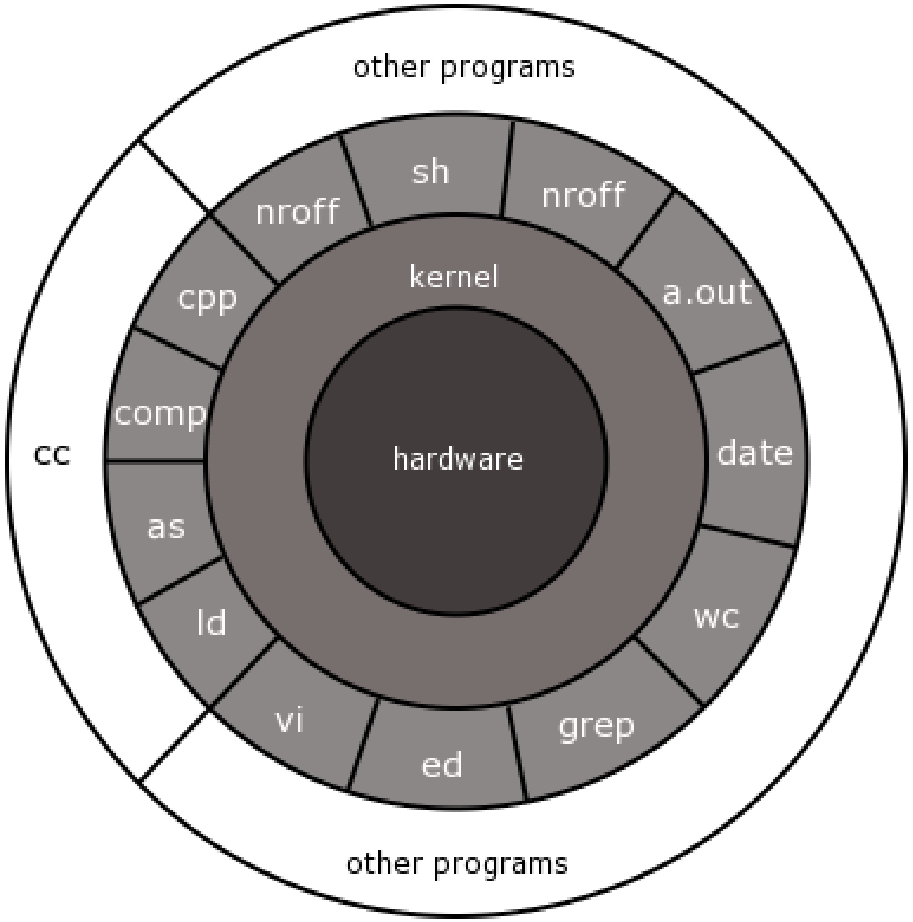


Kernel Mode VS. User Mode

- kernel mode and user mode
  - Why need two modes?
    - ✓ In the kernel mode, the executing code (e.g., interrupt handler) has unrestricted access to the underlying hardware and reference any memory address. Crashes in the kernel mode are catastrophic; they will halt the entire PC.
    - ✓ In the user mode, the executing code has no ability to directly access hardware or reference any other memory addresses. Crashes in user mode are always recoverable.
    - ✓ Essentially, having two modes improves the robustness and security of the system.

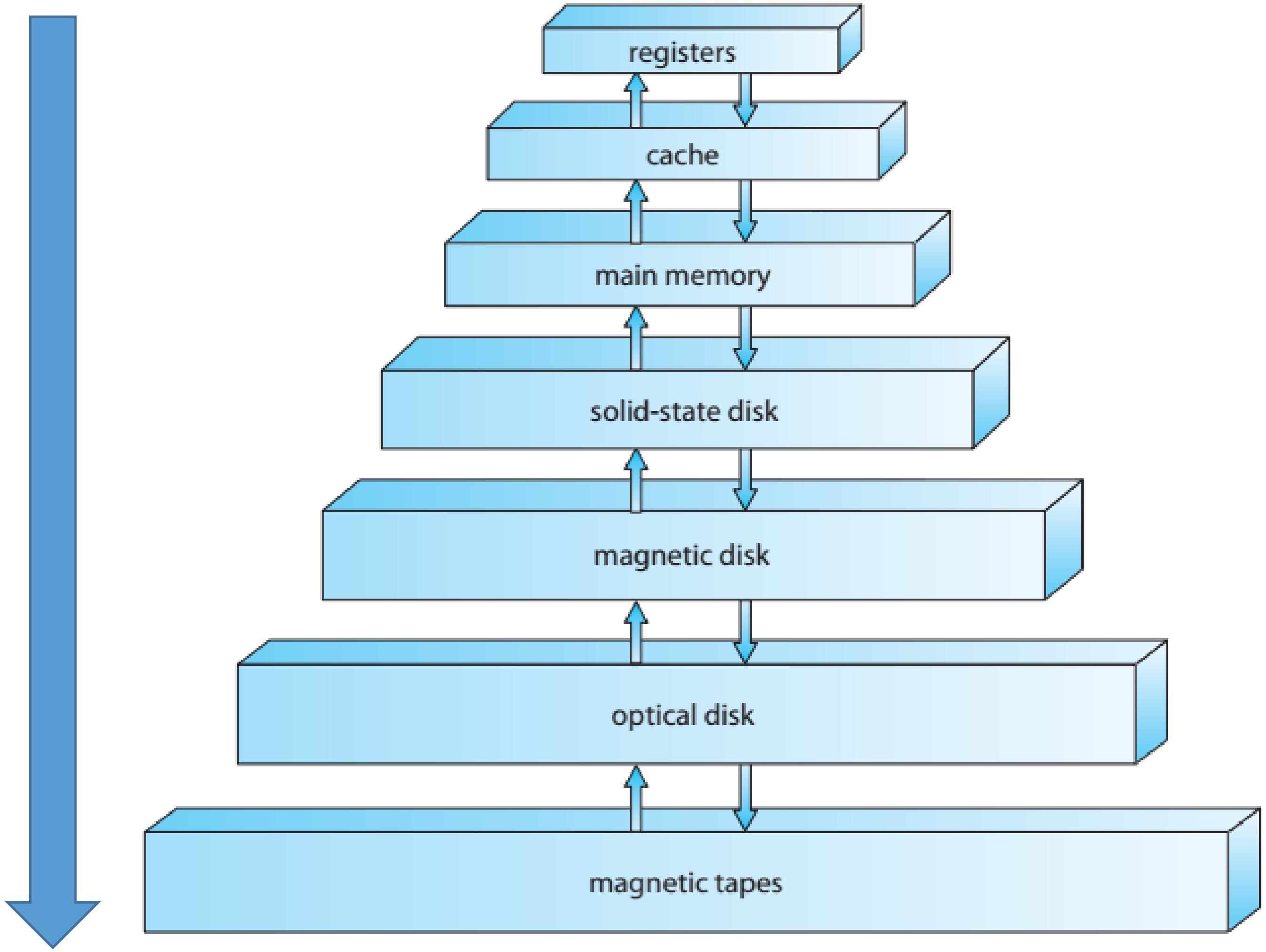
OS & Kernel

- Operating System
  - Kernel+ Utility Software
- Kernel--core component of the OS
  - Manage processes
  - Manages memory
  - Manages file systems
  - Manages access to devices
  - ……



OS & Storage Devices

- Storage systems organized in hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access delay



OS & Storage Devices

❑ Performance of Various Levels of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape