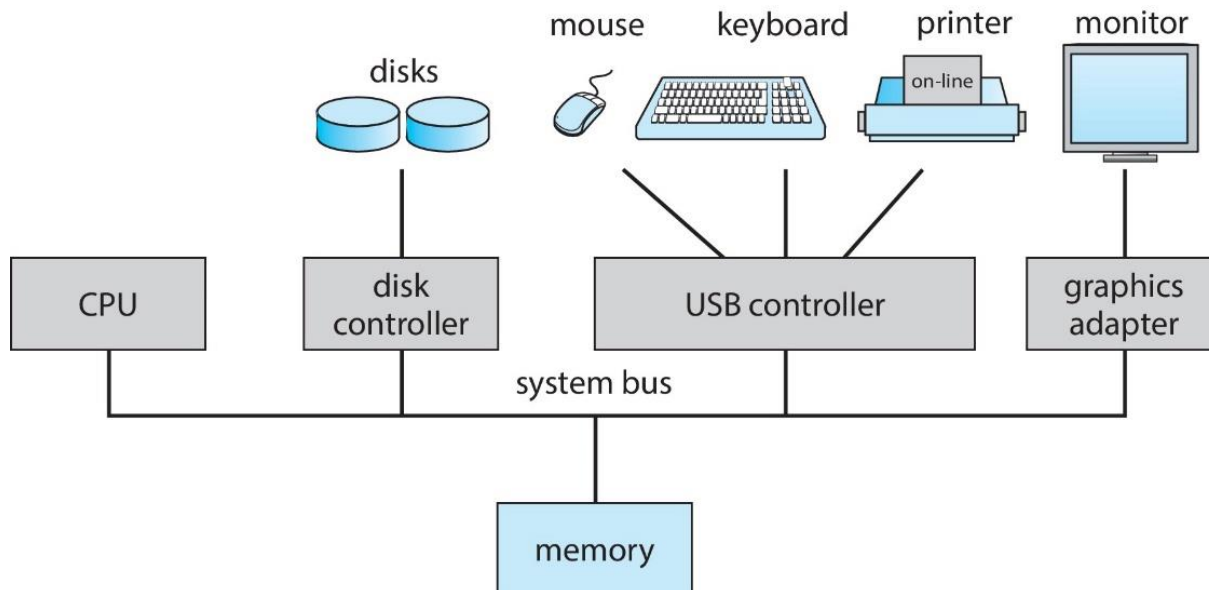


Introduction to Operating Systems (Part 2) - Computer System Structures (Interrupts, DMA, Dual- mode)

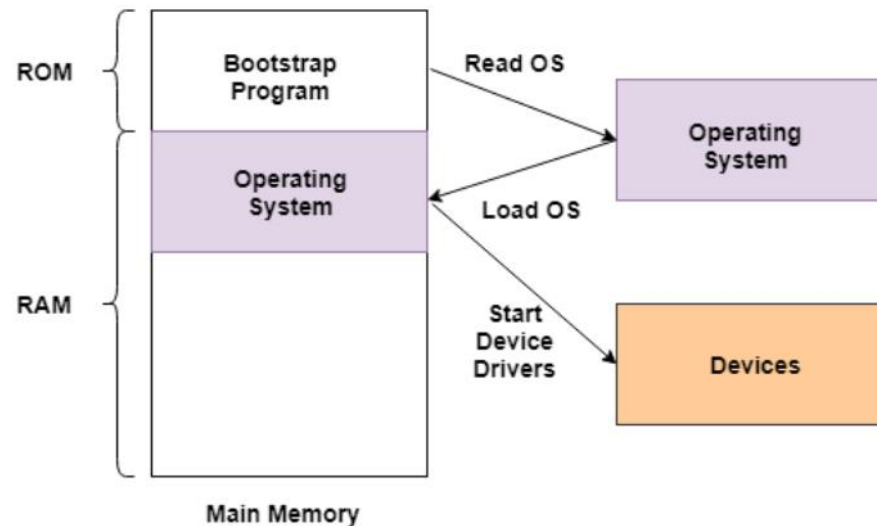
Computer System Organization

- Computer-system operation
 - One or more CPUs, device controllers connect through common **bus** providing access to shared memory
 - Concurrent execution of CPUs and devices competing for memory cycles



Computer Startup

- BIOS
- BIOS performs POST (*Power On Self Test*)
- Bootstrap program



Computer Startup (cont.)

- The init process
 - Ancestor of all other processes.
 - Processes either directly started by init or by one of its child processes.
 - Centrally configured in the `/etc/inittab` file where runlevels are defined.
- Few examples of boot loader
 - GNU GRUB, LILO, BURG, Syslinux
- Sample bootstrap program
 - <https://docs.oracle.com/cd/E19048-01/chorus4/806-0615/6j9v61fut/index.html>

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**

Interrupts

- Interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention.
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

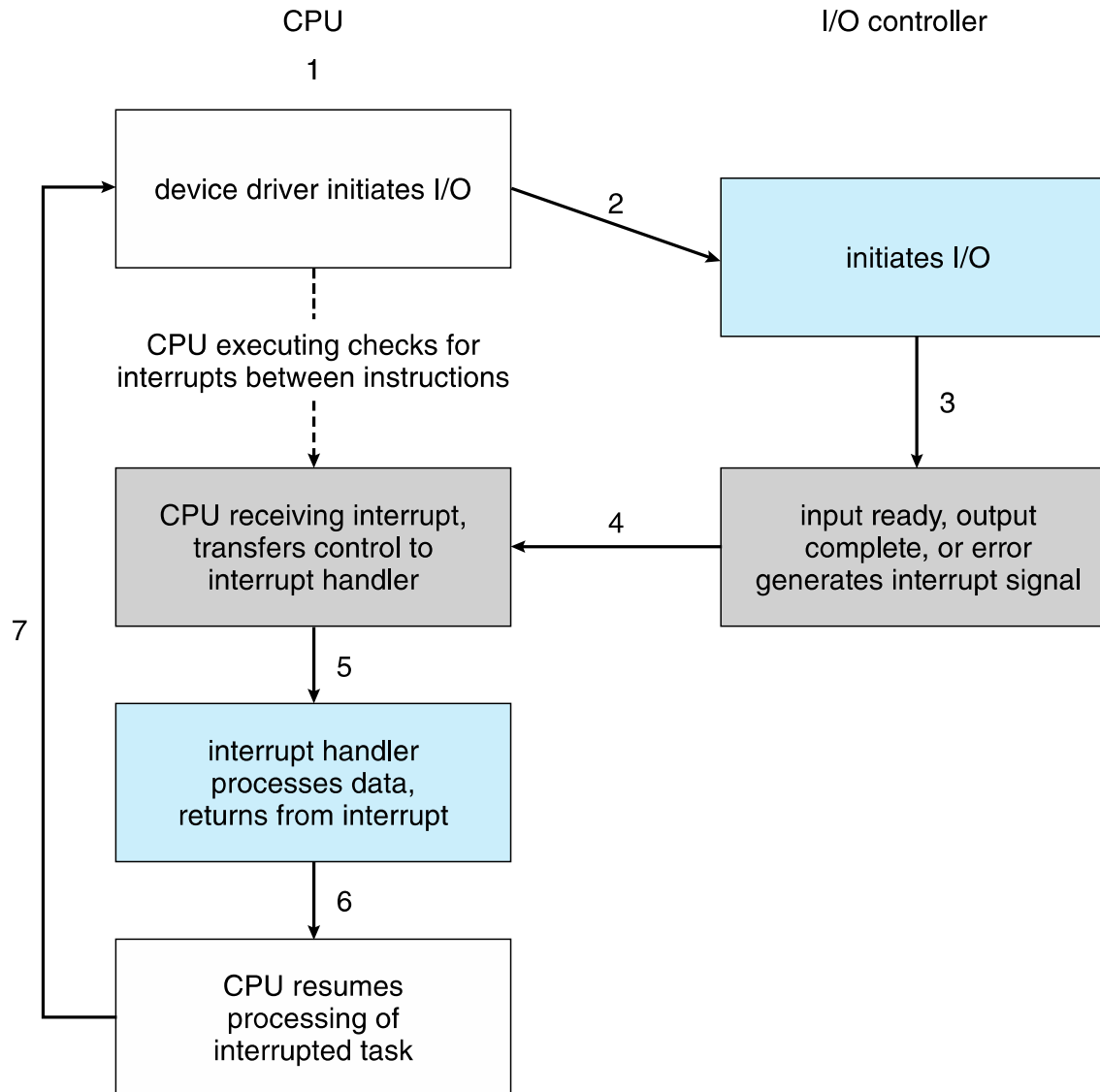
Interrupts

- Sequence of events involved in handling an IRQ
 - Devices raise an IRQ.
 - Processor interrupts the program currently being executed.
 - Device is informed that its request has been recognized and the device deactivates the request signal.
 - The requested action is performed.
 - Interrupt is enabled and the interrupted program is resumed.

Interrupt Handling

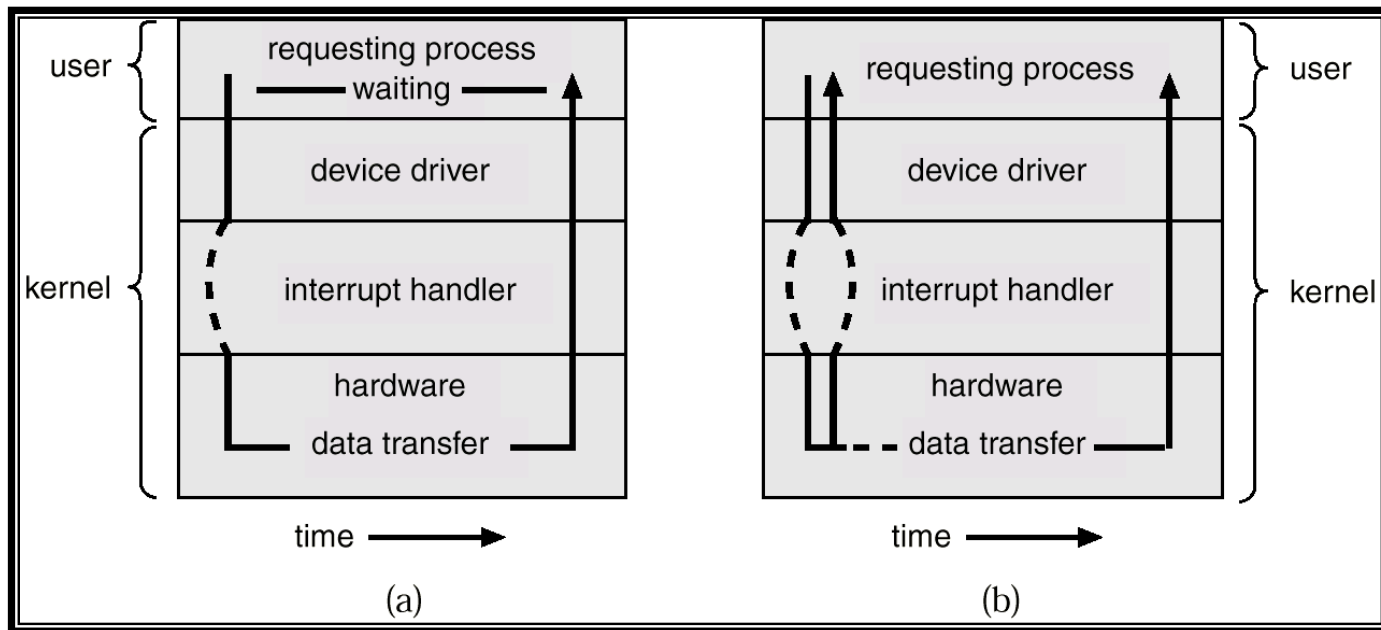
- The operating system preserves the state of the CPU by storing the registers and the program counter
- Determines which type of interrupt has occurred.
- Separate segments of code determine what action should be taken for each type of interrupt.
- Handling Multiple Devices:
 - Polling
 - Vectored Interrupts
 - Interrupt Nesting

Interrupt-drive I/O Cycle



I/O Structure

- Two methods for handling I/O
 - After I/O starts, control returns to user program only upon I/O completion
 - After I/O starts, control returns to user program without waiting for I/O completion

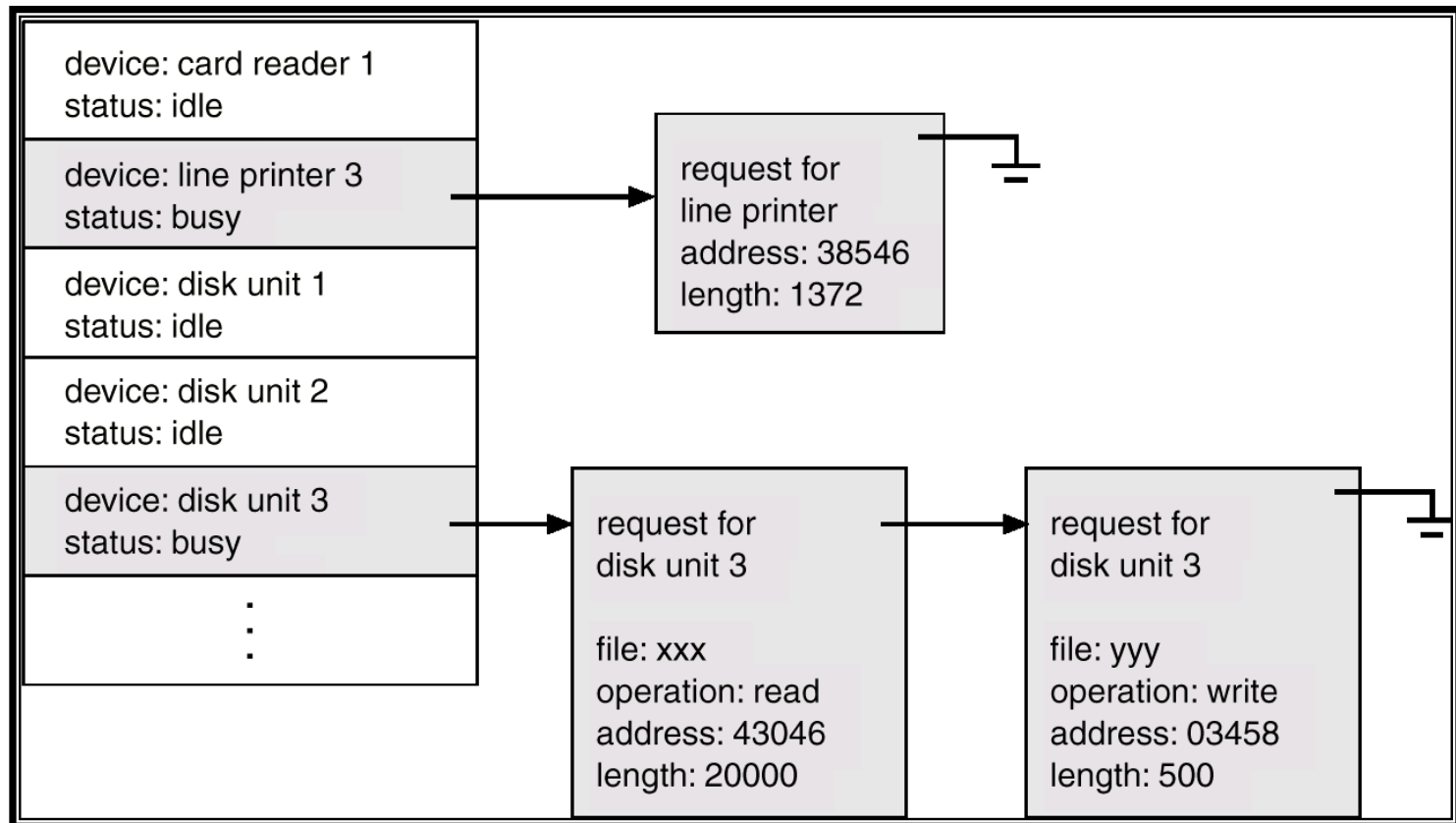


I/O Structure (Cont.)

- After I/O starts, control returns to user program only upon I/O completion
 - Wait instruction idles the CPU until the next interrupt
 - Wait loop (contention for memory access)
 - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- After I/O starts, control returns to user program without waiting for I/O completion
 - **System call** – request to the OS to allow user to wait for I/O completion
 - **Device-status table** contains entry for each I/O device indicating its type, address, and state
 - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

I/O Structure (Cont.)

- Device-status table



I/O Techniques

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.
- Three techniques are possible for I/O operations:
 - Programmed I/O
 - Interrupt-driven I/O
 - Direct Memory Access (DMA).

Programmed I/O

- The I/O module performs the requested action then sets the appropriate bits in the I/O status register
- The processor periodically checks the status of the I/O module until it determines the instruction is complete
- With programmed I/O the performance level of the entire system is severely degraded

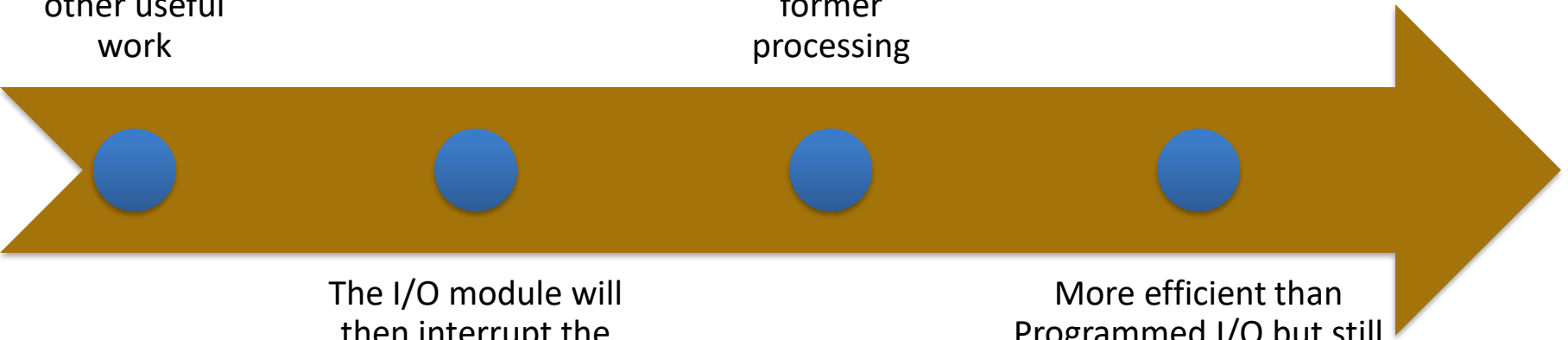
Interrupt-Driven I/O

Processor issues an I/O command to a module and then goes on to do some other useful work

The processor executes the data transfer and then resumes its former processing

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor

More efficient than Programmed I/O but still requires active intervention of the processor to transfer data between memory and an I/O module



Interrupt-Driven I/O Drawbacks

- Transfer rate is limited by the speed with which the processor can test and service a device
- The processor is tied up in managing an I/O transfer
 - a number of instructions must be executed for each I/O transfer

Direct Memory Access (DMA)

- Performed by a separate module on the system bus or incorporated into an I/O module.
- When the processor wishes to read or write a block of data, it issues a command to the DMA module containing:
 - Whether a read or write is requested
 - The address of the I/O device involved
 - The starting location in memory to read data from or write data to
 - The number of words to be read or written

DMA Controller

- DMA controller is a hardware unit that allows I/O devices to access memory directly without the participation of the processor.
 - processor is involved only at the beginning and end of the transfer
 - processor executes more slowly during a transfer when processor access to the bus is required
- More efficient than interrupt-driven or programmed I/O

DMA Controller

- The DMA controller transfers the data in three modes:
 - Burst mode
 - Cycle stealing mode
 - Transparent mode

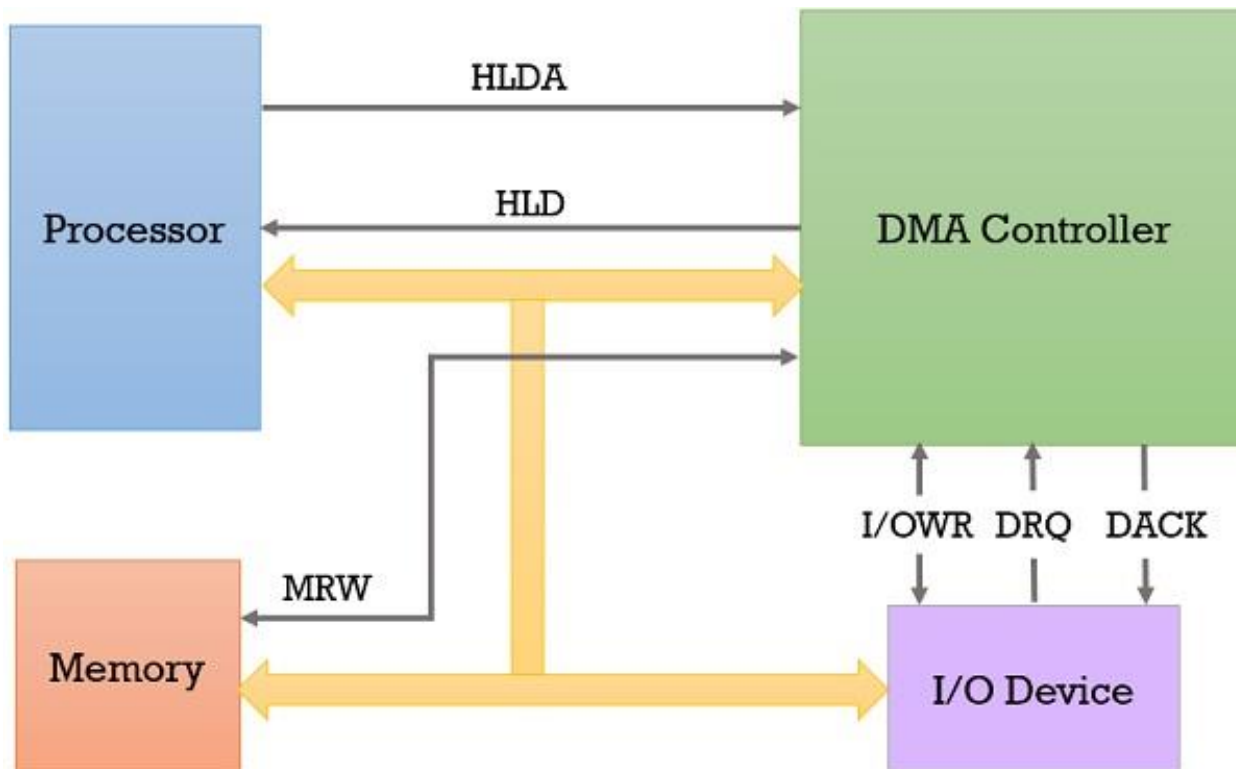
DMA Controller

DRQ: DMA request

HLDA: Hold acknowledgement

HLD: Hold request

DACK: DMA acknowledgement to I/O



DMA Controller Data Transfer

Storage Structure

- Main memory – only large storage media that the CPU can access directly
 - Random access
 - Typically volatile
 - Typically random-access memory in the form of Dynamic Random-access Memory (DRAM)
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity

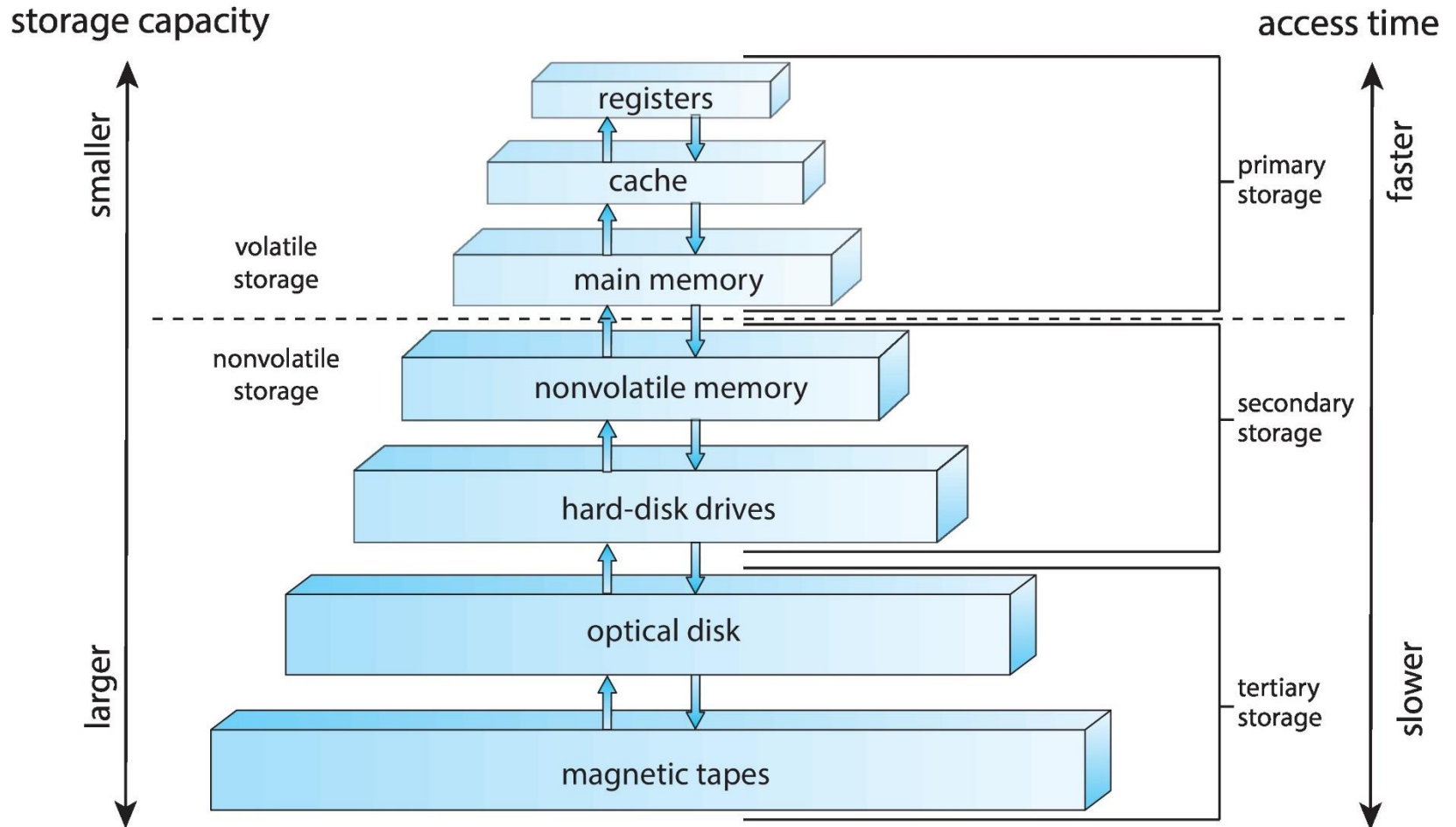
Storage Structure (Cont.)

- **Hard Disk Drives (HDD)** – rigid metal or glass platters covered with magnetic recording material
 - Disk surface is logically divided into tracks, which are subdivided into sectors
 - The disk controller determines the logical interaction between the device and the computer
- **Non-volatile memory (NVM)** devices– faster than hard disks, nonvolatile
 - Various technologies
 - Becoming more popular as capacity and performance increases, price drops

Storage Hierarchy

- Storage systems organized in hierarchy
 - Speed
 - Cost
 - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
 - Provides uniform interface between controller and kernel

Storage-Device Hierarchy



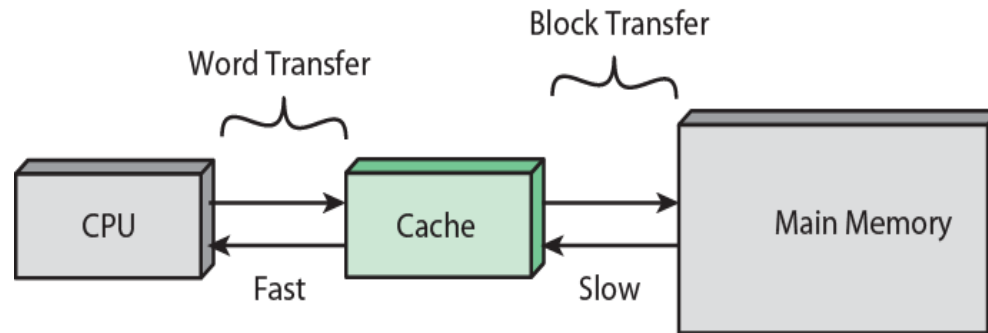
Characteristics of Various Types of Storage

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 DRAM	CMOS DRAM	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

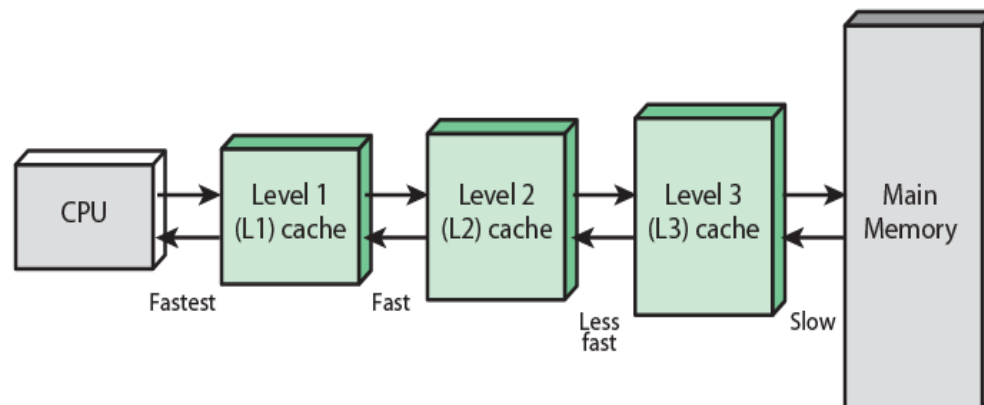
Caching

- Use of high-speed memory to hold recently-accessed data.
- Requires a *cache management* policy.
- Caching introduces another level in storage hierarchy. This requires data that is simultaneously stored in more than one level to be *consistent*.

Cache Organization



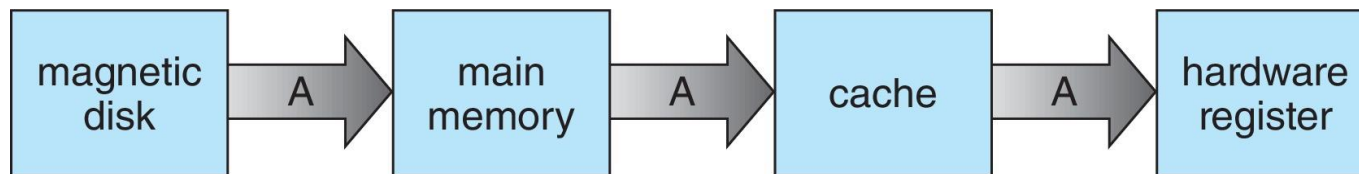
(a) Single cache



(b) Three-level cache organization

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache.
- Cache Write Policies:
 - Write back
 - Write through

Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

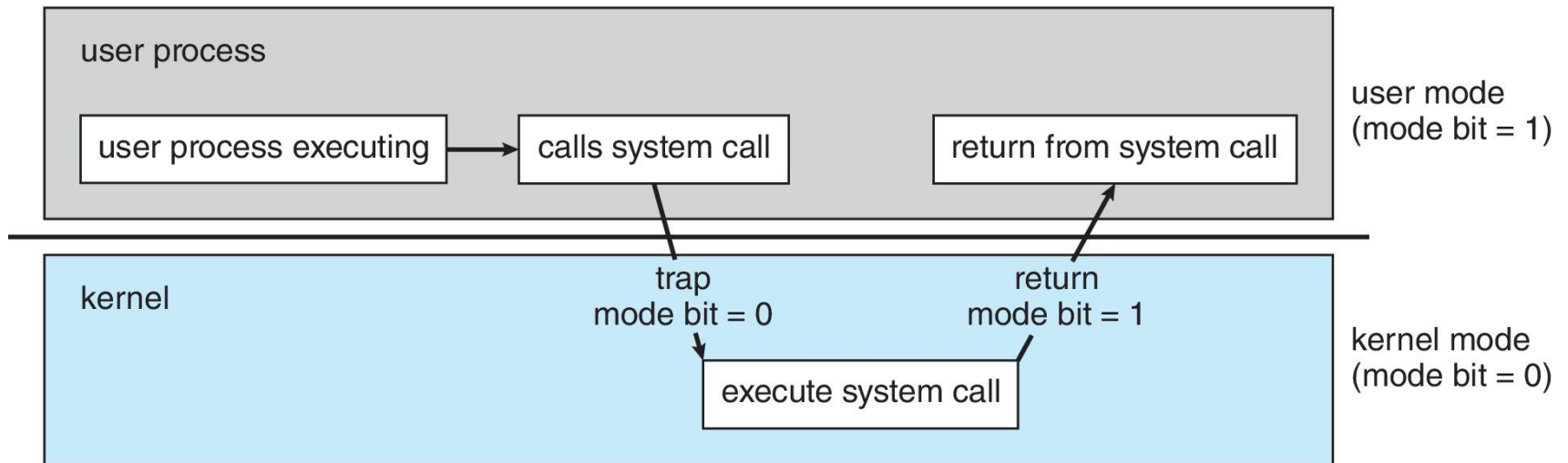
Dual-mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
 - **User mode** and **kernel mode (monitor or system mode)**
- **Mode bit** provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code.
 - When a user is running → mode bit is “user”
 - When kernel code is executing → mode bit is “kernel”

Dual-mode Operation (cont.)

- When an interrupt or fault occurs hardware switches to monitor mode.
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
 - System call changes mode to kernel, return from call resets it to user
- Privileged instructions can be issued only in monitor mode.

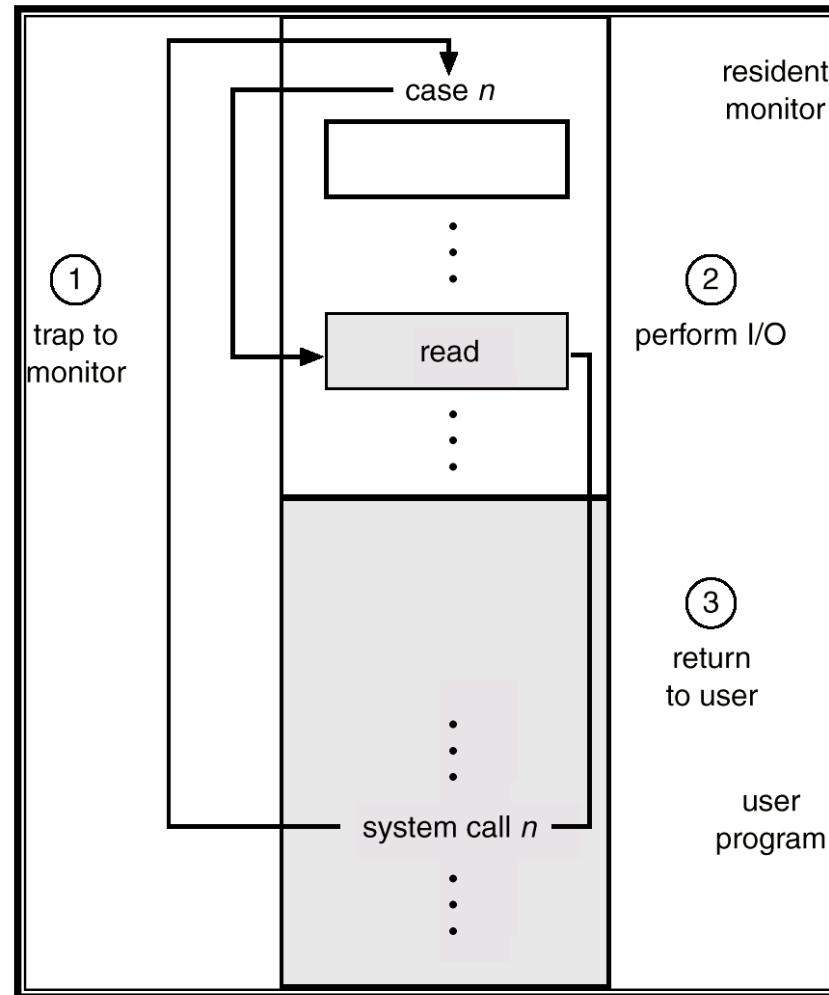
Transition from User to Kernel Mode



I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

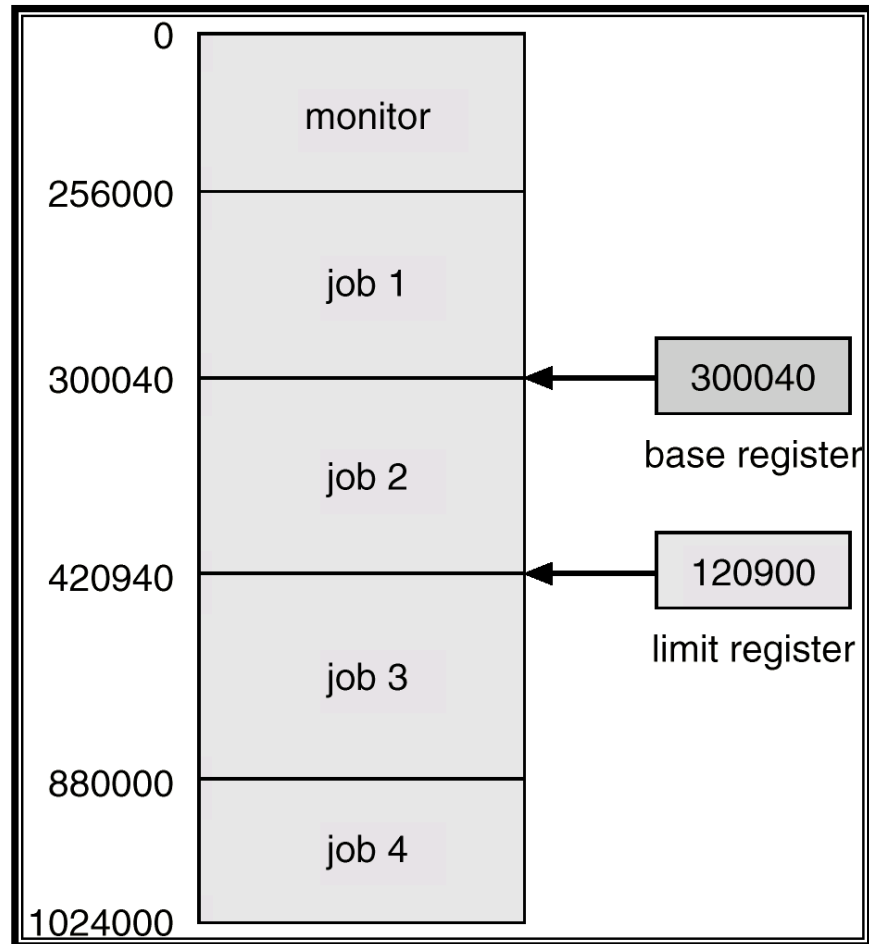
Use of A System Call to Perform I/O



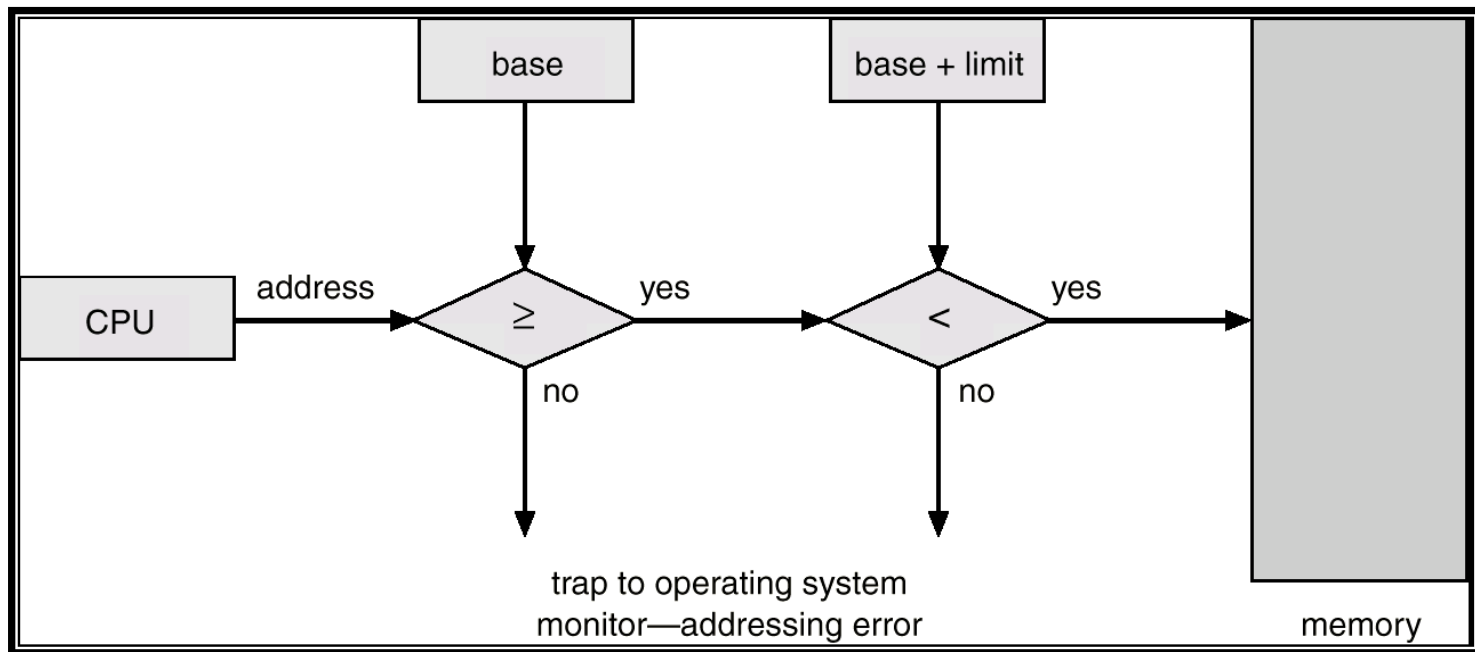
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **Base register** – holds the smallest legal physical memory address.
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected.

Use of A Base and Limit Register



Hardware Address Protection



Memory Protection (cont.)

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
- The load instructions for the *base* and *limit* registers are privileged instructions.

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
 - Timer is decremented every clock tick.
 - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.