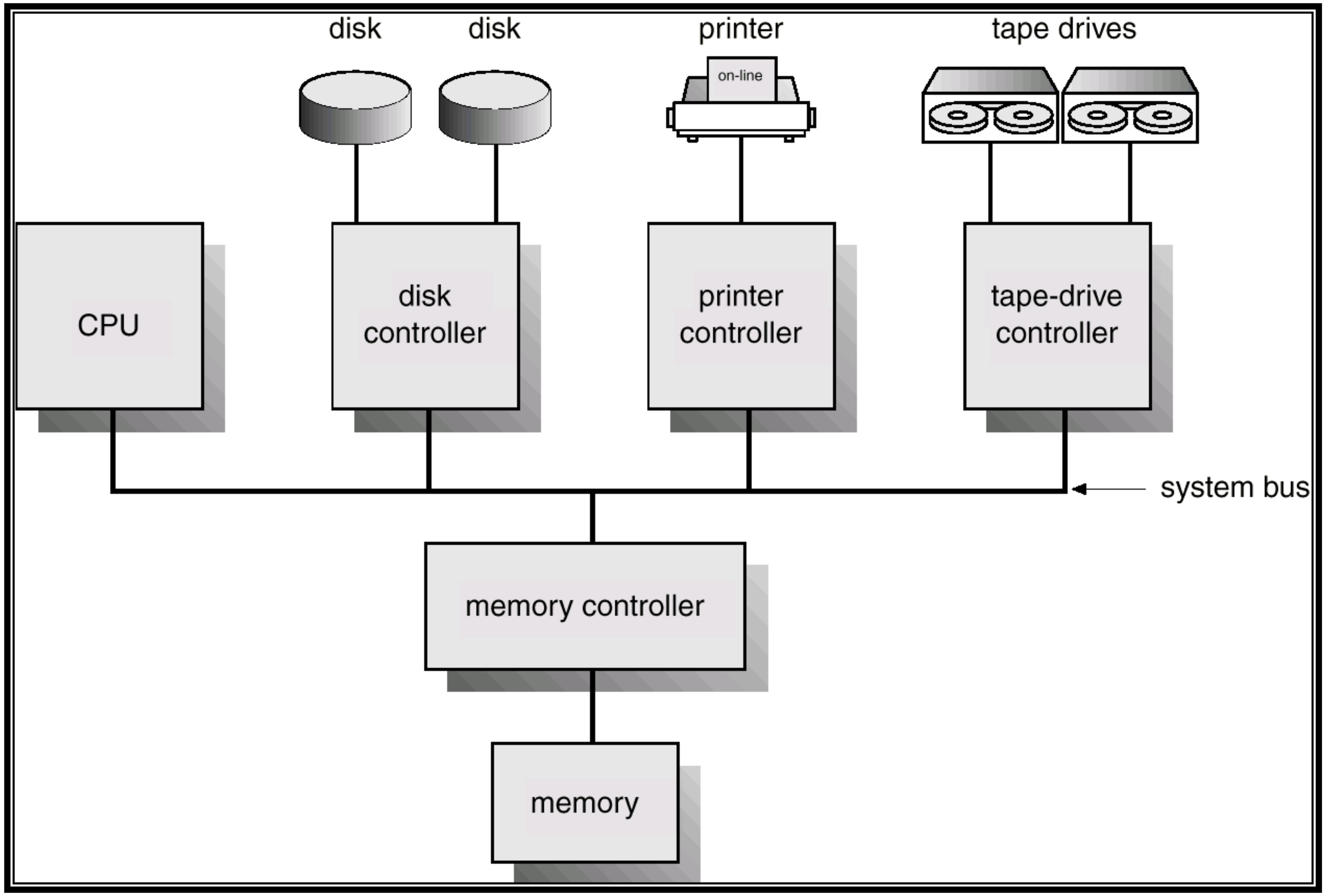
Operating Systems

Computer System Structures

- Computer System Operation
- I/O Structure
- Storage Structure
- Storage Hierarchy
- Hardware Protection
- General System Architecture

Computer-System Architecture



Computer-System Operation

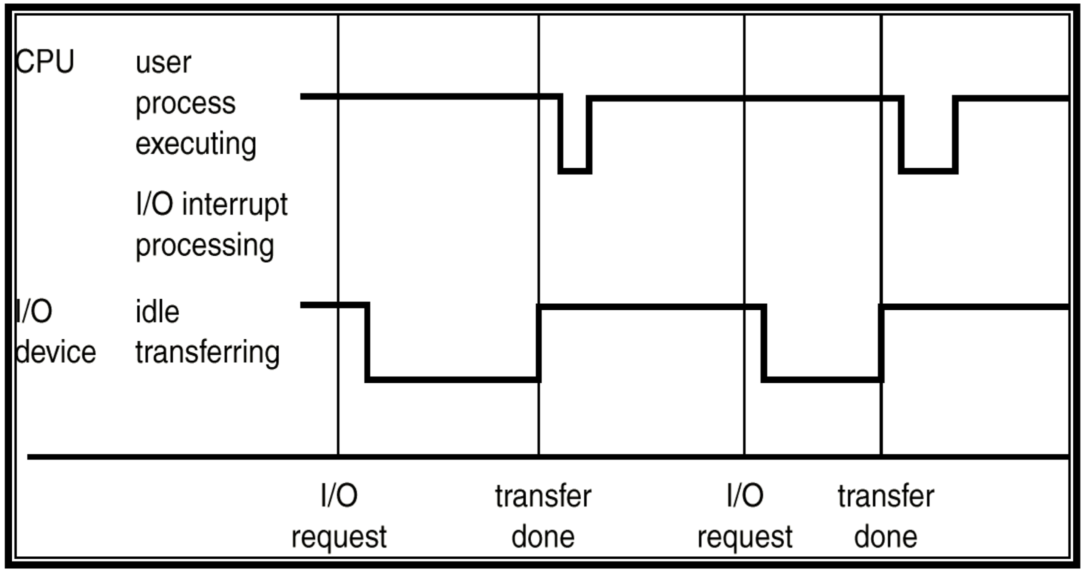
- I/O devices and the CPU can execute concurrently **– due to multi-programming & interrupt schemes and use of controllers - not a new idea: IBM's I/O Channels - 1960's.**
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer **I/O -> buffer is slow, but buffer -> CPU is fast.**.
- CPU moves data from/to main memory to/from local buffers **– in absence of DMA**
- 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*.
- **Device controller & CPU executes concurrently: controller: controller fills buffer, CPU empties buffer.**

Common Functions of Interrupts

- 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.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt* **vs "nested interrupts".**
- A *trap* is a software-generated interrupt caused either by an error or a user request - **ex: system call is software generated (in the source code). – example the int instruction in DOS assembly language.**
- An operating system is *interrupt* driven.

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - ✓ *Polling* **– actually done in a *"non-interrupt"* situation**
  - ✓ *vectored* interrupt system **– goes directly to ISR vs. a common jump location to analyze the interrupt before processing it.**
- Separate segments of code determine what action should be taken for each type of interrupt
- **Interrupt Vector addresses a particular a particular address in the IVT, which points to an *interrupt handling routine* (ISR).**
  - ✓ **Uses the IRQ lines in the control bus, and the Programmable Interrupt Controller (PIC)**

Interrupt Time Line For a Single Process Doing Output



←--User process
←--ISR move data from buffer to user process

←------------------Data xfr

Synchronous: After I/O starts, control returns to user program only upon I/O completion **… no CPU action on user behalf until I/O complete… Two approaches:**

- Interrupts are part of the architecture:
  - ✓ **User executes a "wait" instruction which blocks the user until an interrupt is issued to indicate that that the I/O is complete**
  - ✓ **The user goes into a wait loop until an interrupt is issued to indicate that that the I/O is complete**
  - ✓ **Both cases are an inefficient use of interrupts**

- No interrupts in the architecture.
  - ✓ **When the user makes the I/O request, the device driver will poll a "I/O complete" bit in the port until it indicates that I/O is complete, at which time control is returned to the user.**
  - ✓ **This can be done even it interrupts are part of the architecture, if the I/O wait time is anticipated to be very short, and context switching will be avoided**.

- Typically. one I/O request is outstanding at a time, no simultaneous I/O processing **… disable interrupts?**

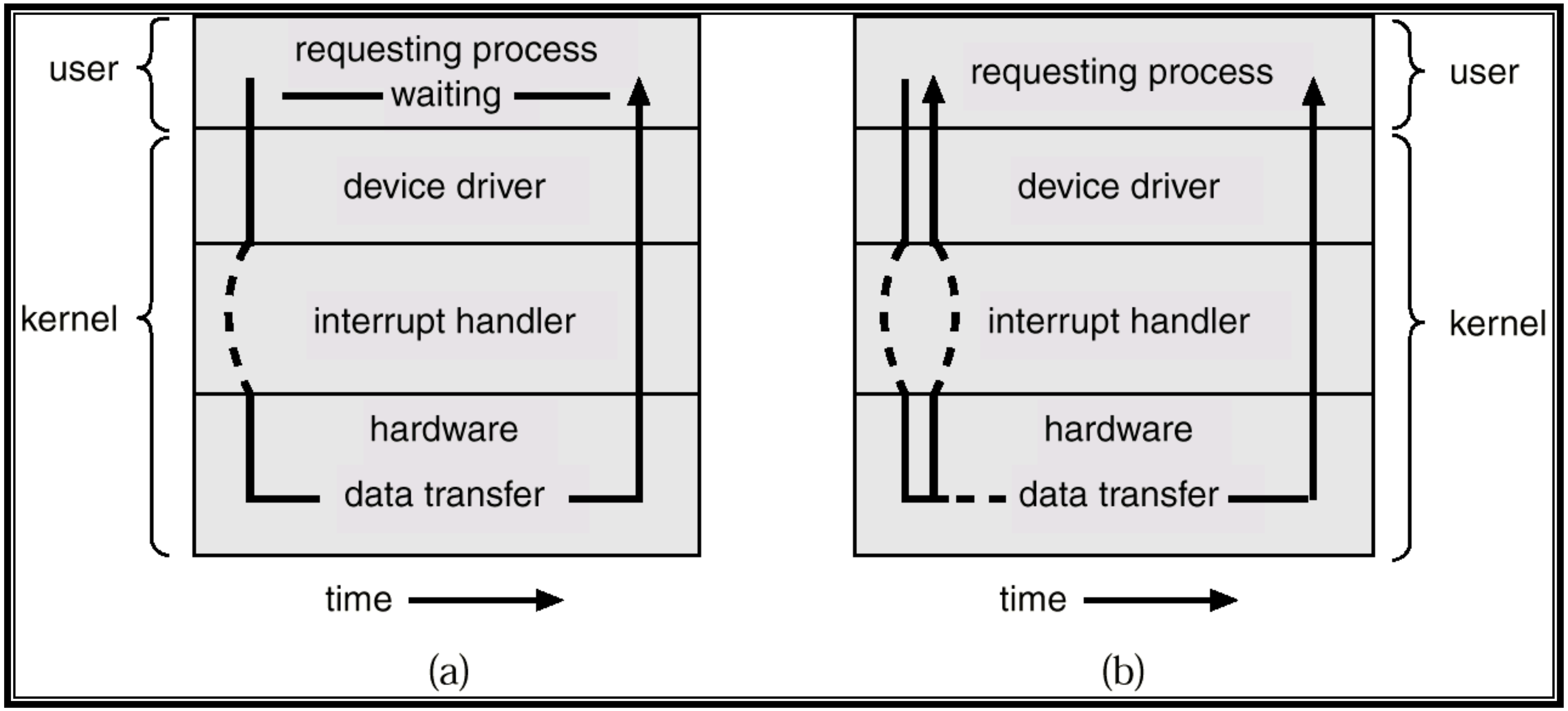
Asynchronous(2 modes: current user proc goes on vs some other process goes on): After I/O starts, control returns to user program without waiting for I/O completion **– if user cannot continue w/o results, OS can switch to another user process.**

- ✓ **This action is generally accomplished using a** *System call* – A good design would allow another user process to run if the requesting process could not run without the results of the I/O request (a task switch).
- ✓ *Device-status table* contains entry for each I/O device indicating its type, address, and state **– queue up processes waiting for the device.**
- ✓ Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

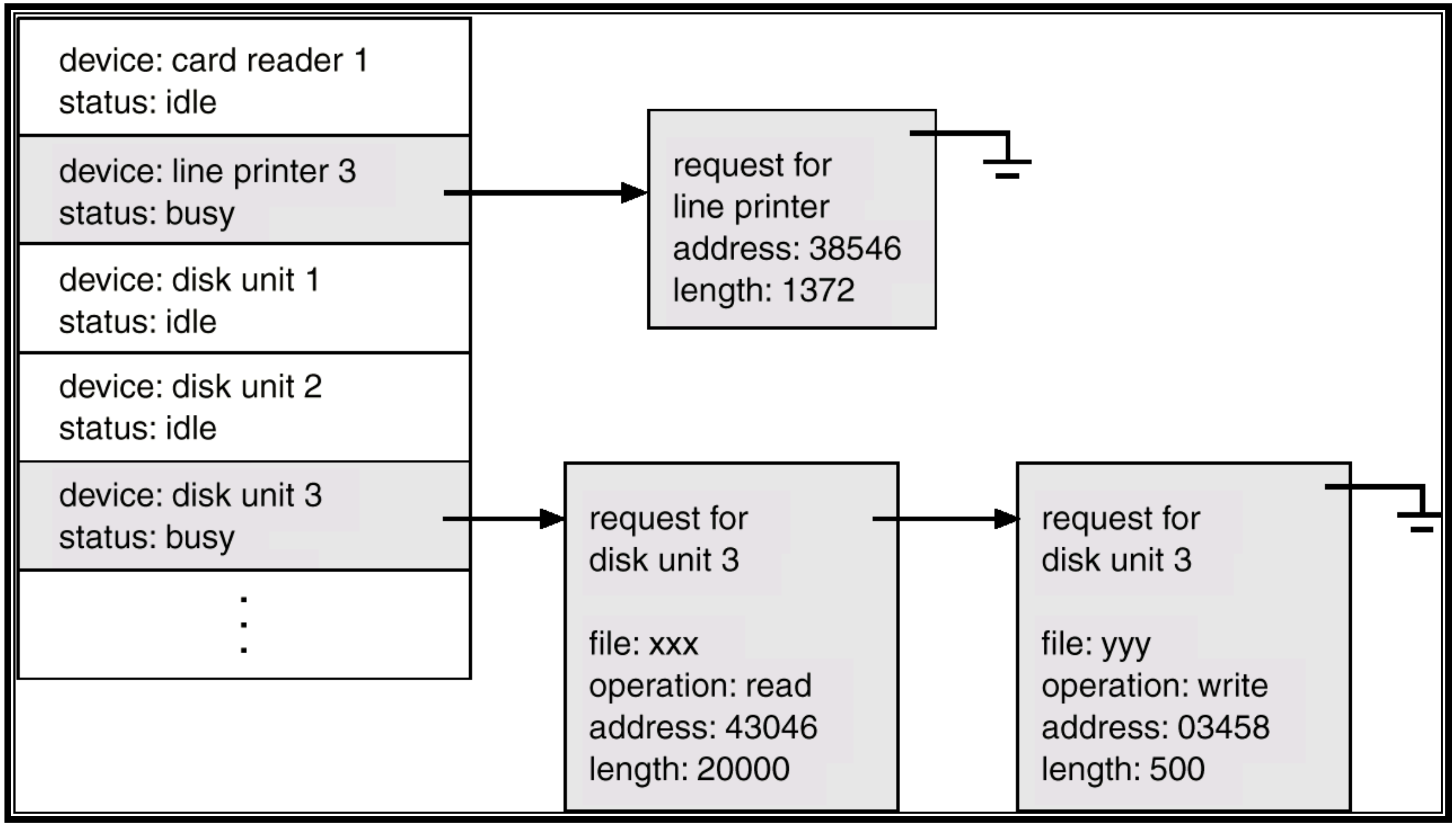
Two I/O Methods

Synchronous

Asynchronous



Device-Status Table



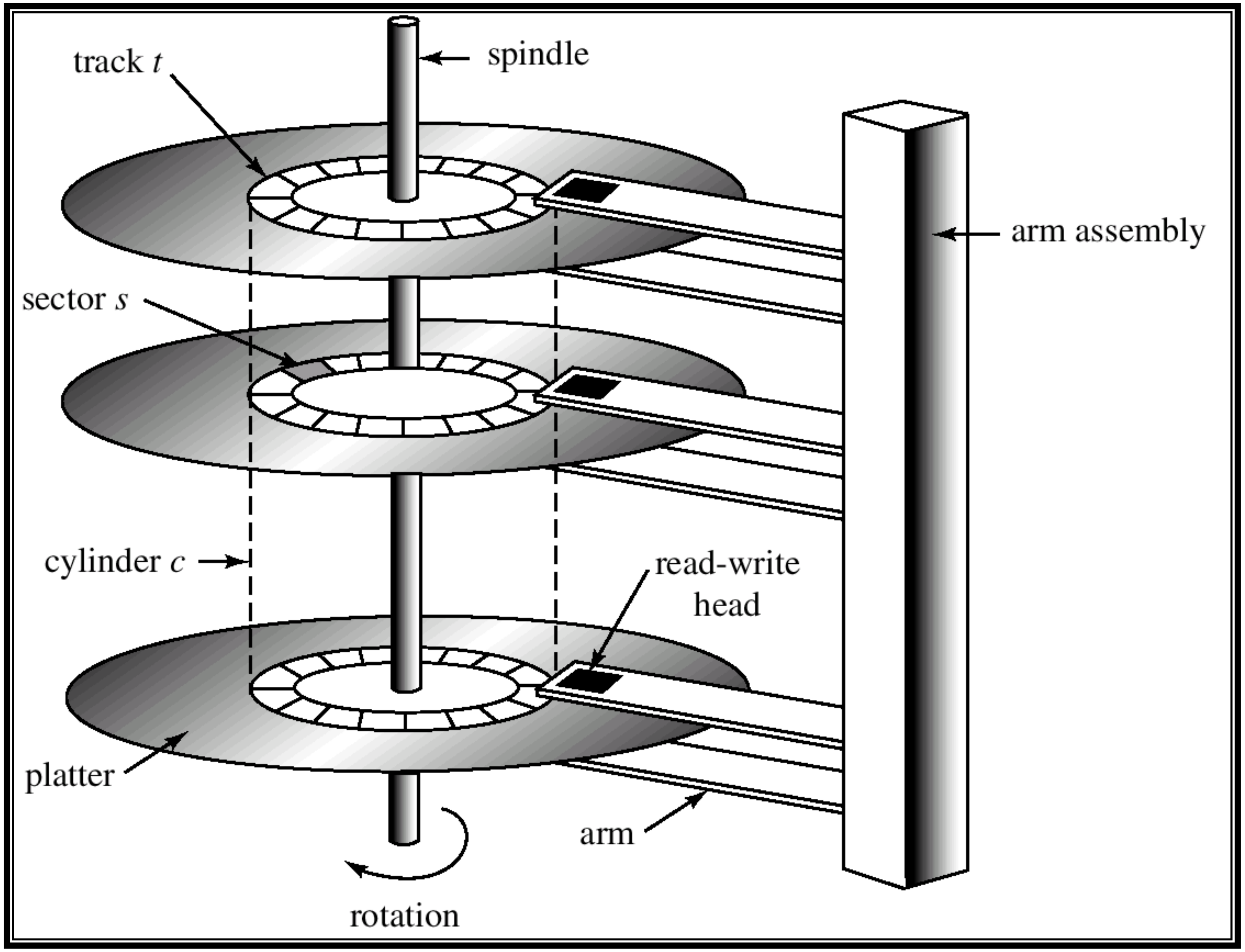
Return to process making a request when the request is completed via an interrupt.

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers ***blocks*** of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte**==> in a pure interrupt scheme, granularity of data xfr is typically on a byte or word basis – OK if a slow serial port – overhead is small percent, but high speed xfr, bytes are coming too fast and percent overhead is significant – leaving not much time for data xfr. Solution is DMA.**
- **DMA controller xfr's data from device buffer to main memory (via bus) in parallel with CPU operations … interrupt at end of DMA action which is a relatively large block. Interrupts now infrequent - overhead of interrupts now minimal.**
  - ✓ **Problem: "cycle stealing" - when there is bus/memory contention when CPU is executing a memory word during a DMA xfr, DMA wins out and CPU will pause instruction execution memory cycle (cycle was "stolen").**

- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – 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.
  - ✓ Disk storage is indirect access

- **Main memory & registers accessed directly via instructions. Disk storage is indirect access: must first move data to memory before CPU can access it.**
- **Memory mapped I/O – write to "special" memory locations using memory words (example – video buffer).**
- **Port I/O is similar to memory mapping – registers have address.**
- **Memory mapping allows direct access of outside Storage elements (I/O) - done in hardware.**

Moving-Head Disk Mechanism



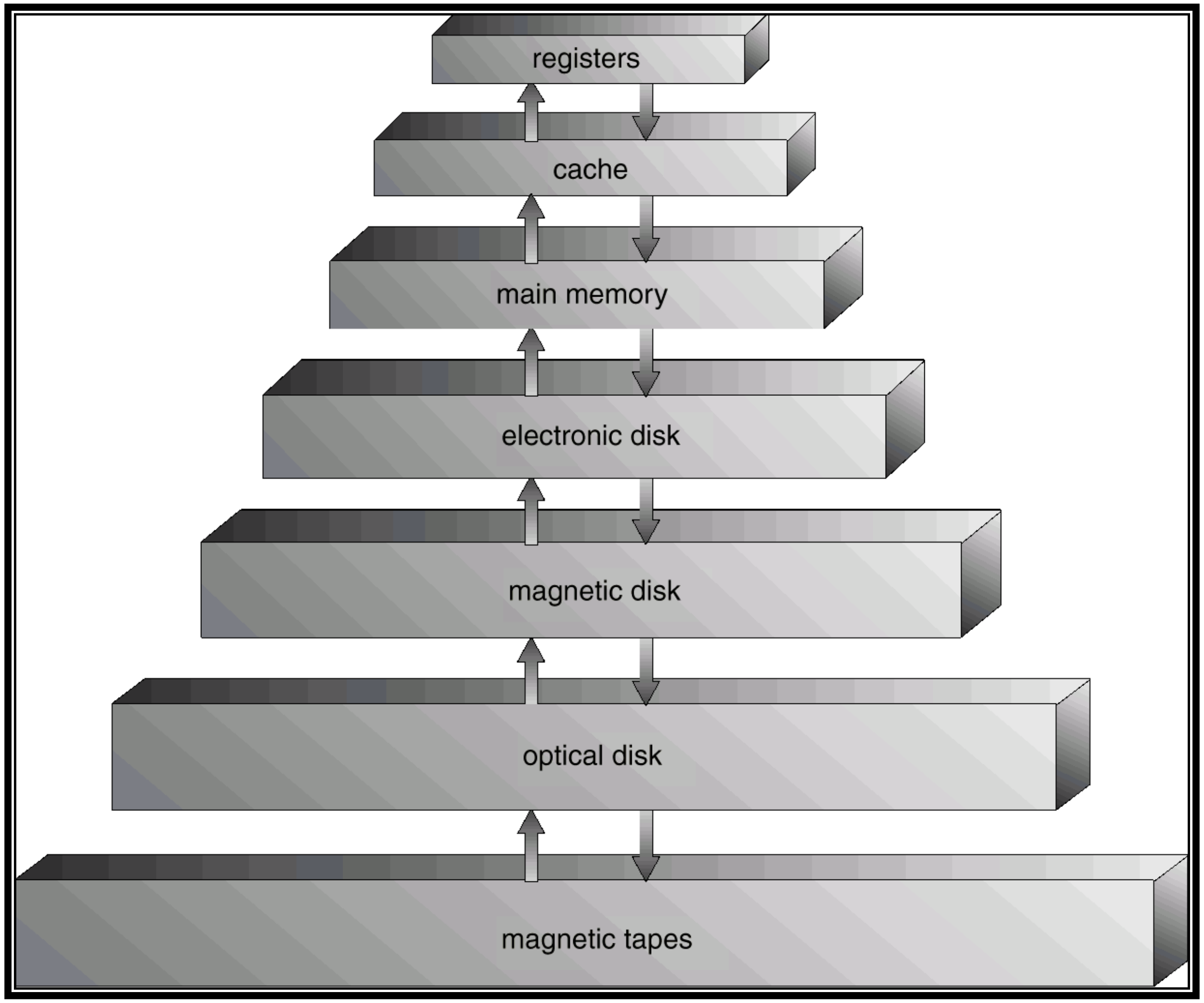
Storage Hierarchy

- Storage systems organized in hierarchy.
  - ✓ Speed
  - ✓ Cost
  - ✓ Volatility
- *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.
- “principle of locality” makes it work.

Storage-Device Hierarchy

Fast, expensive, small

Slow cheap Large



Performance of Various Levels of Storage

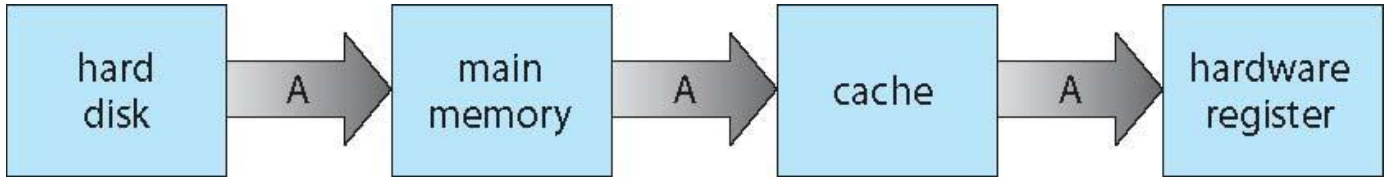
Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 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

Movement between levels of storage hierarchy can be explicit or implicit

- **Cache:** 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*.
  - ✓ **Caching implemented in hardware – integrated with the CPU.**
- **Virtual Memory: In the storage hierarchy, main memory is a cache for the the disk. This is how virtual memory is implemented using disk paging. For the most part this function is implemented in software (The OS memory management function)**
  … lots more on this later.

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
- Distributed environment situation even more complex
    - ✓Several copies of a datum can exist

Hardware Protection

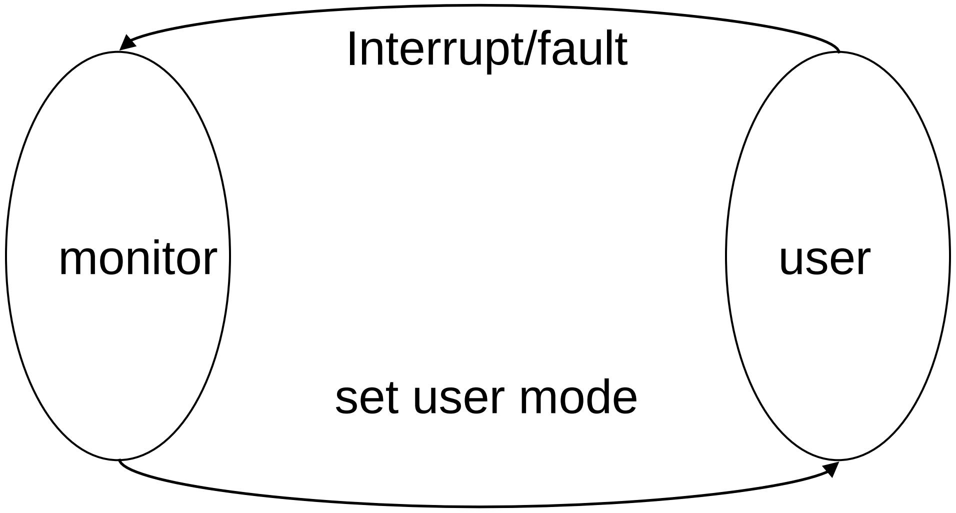
- Dual-Mode Operation– **supervisor vs user mode – privileged operations**
- I/O Protection – **all I/O instruction privileged. Keep user from getting control of computer in "user mode"**
- Memory Protection– **keep processes from accessing outside of its process space … base reg & limit reg checked in addressing instructions.**
- CPU Protection – **control the amount of time a user process is using the CPU. Use a timer … time slicing … prevents infinite loop hangs.**

Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations **- a “mode bit”.**
  1. *User mode* – execution done on behalf of a user.
  2. *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system.

Dual-Mode Operation

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.

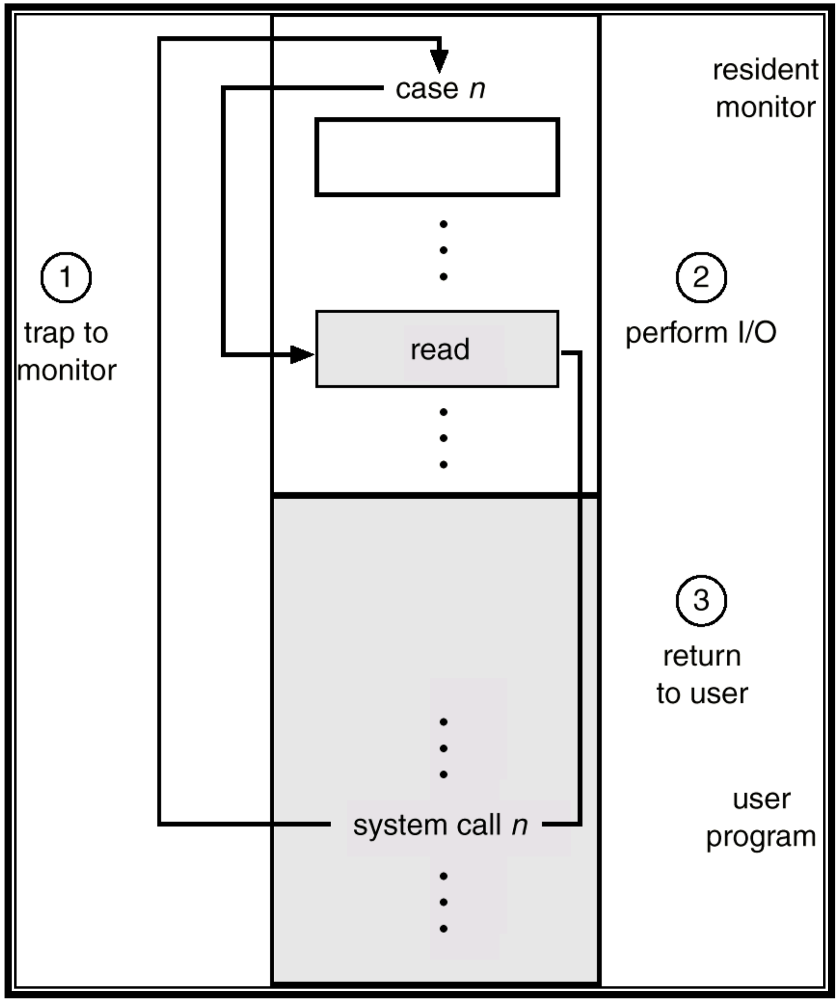


Privileged instructions can be issued only in monitor mode.

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 **… overrides dual mode (I/O) protection - need memory protection to protect IVT).**

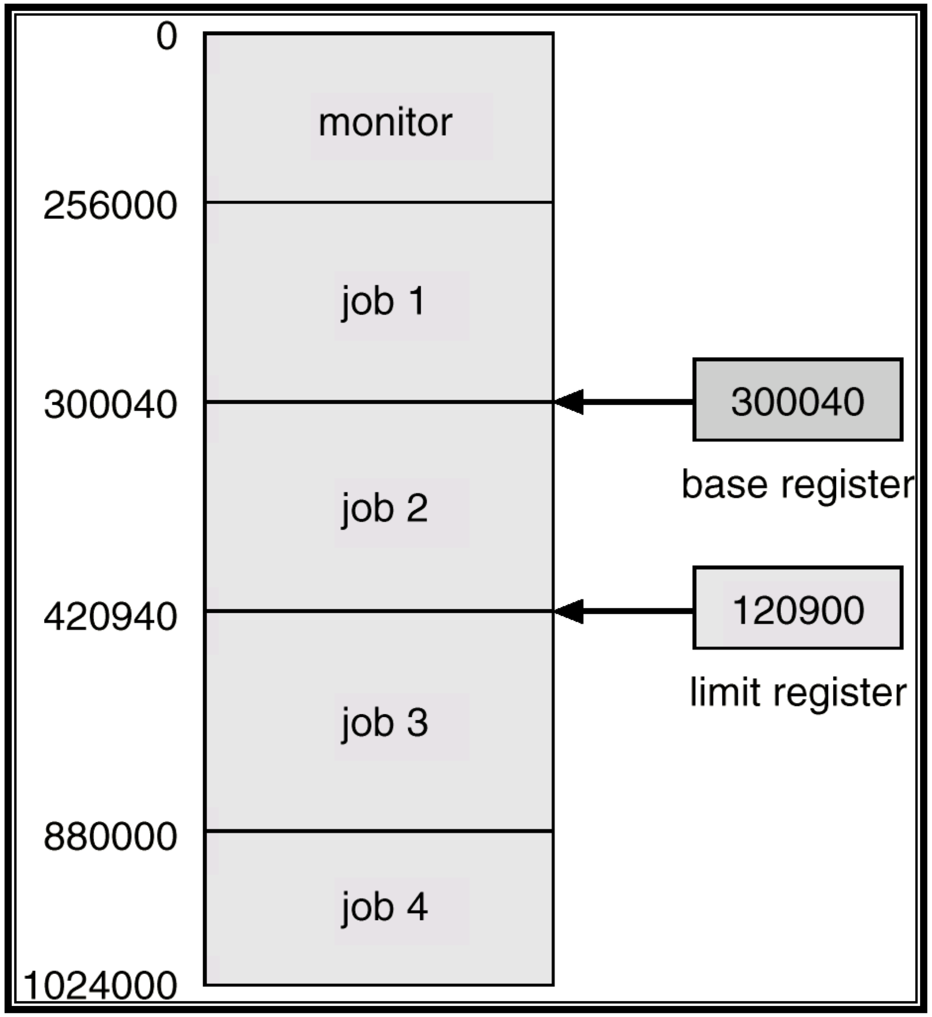
Use of A System Call to Perform I/O



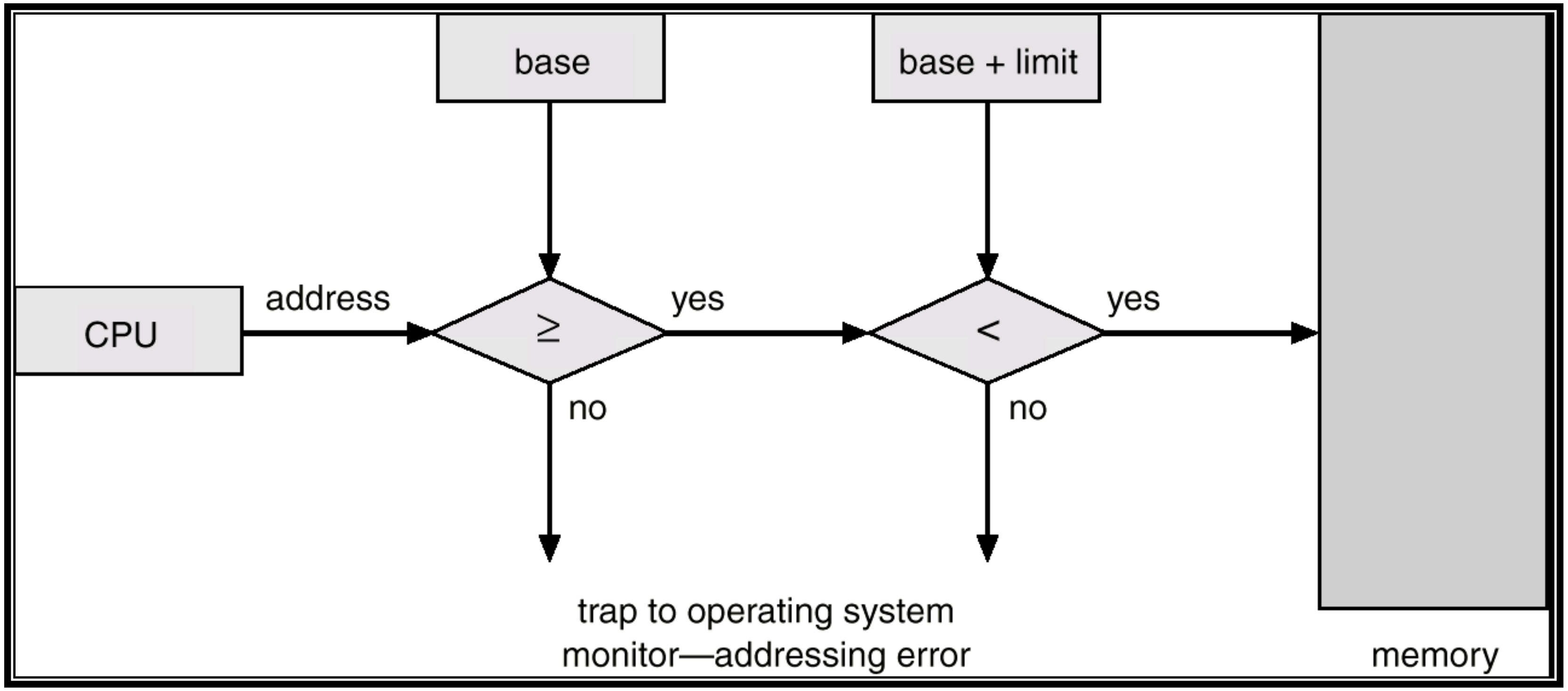
Memory Protection

- 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



- 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. **… an example of hardware protection**

- Prevent user programs from hogging the CPU
- *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 … HW protection.

Bibliography

- ❖ Silberschatz, A, Galvin, P.B, and Gagne, G., Operating System Principles, 9e, John Wiley & Sons, 2013.
- ❖ Stallings W., Operating Systems-Internals and Design Principles, 7e, Pearson Education, 2014.
- ❖ Harvey M. Deital, “Operating System”, Third Edition, Pearson Education, 2013.
- ❖ Andrew S. Tanenbaum, “Modern Operating Systems”, Second Edition, Pearson Education, 2004.
- ❖ Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

Acknowledgements

- I have drawn materials from various sources such as mentioned in bibliography or freely available on Internet to prepare this presentation.
- I sincerely acknowledge all sources, their contributions and extend my courtesy to use their contribution and knowledge for educational purpose.

Thank You!!

?