ILLINOIS TECH

College of Computing

CS 450 Operating Systems
Swapping & Intro to IO

Yue Duan

Recap: Swapping

- OS goal: Support multiple processes
  - Single process with very large address space
  - Multiple processes with combined address spaces
- User code should be independent of amount of physical memory
  - Correctness, if not performance
- Virtual memory: OS provides illusion of more physical memory
- Reality: many processes, limited physical memory

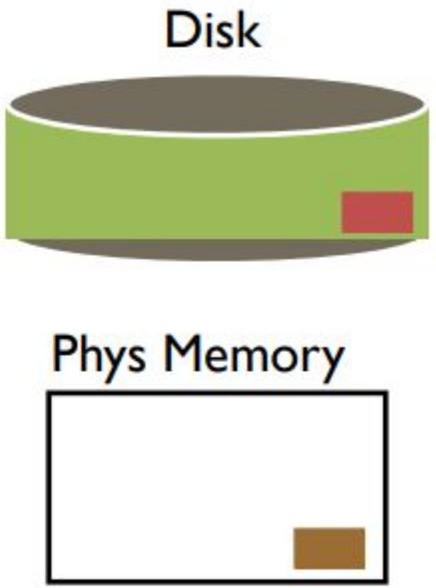
Recap: Swapping

- Idea: OS keeps unreferenced pages on disk
  - Slower, cheaper backing store than memory
- Process can run when not all pages are loaded into main memory
- OS and hardware cooperate to make large disk seem like memory
  - Same behavior as if all of address space in main memory
- Requirements:
  - OS must have **mechanism** to identify location of each page in address space ===> in memory or on disk
  - OS must have **policy** to determine which pages live in memory and which on disk

Virtual Address Space Mechanisms

- Each page in virtual address space maps to one of three locations:
  - Physical main memory: Small, fast, expensive
  - Disk (backing store): Large, slow, cheap
  - Nothing (error): Free
- Extend page tables with an extra bit: present
  - permissions (r/w), valid, present
  - Page in memory: present bit set in PTE
  - Page on disk: present bit cleared
    - PTE points to block on disk
    - Causes trap into OS when page is referenced
    - Trap: page fault

Virtual Address Space Mechanisms



PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

What if access vpn 0xb?

Virtual Address Space Mechanisms

- First, hardware checks TLB for virtual address
- if TLB hit, address translation is done; page in physical memory
- Else
  - Hardware or OS walk page tables
  - If PTE designates page is present, then page in physical memory (i.e., present bit is cleared)
  - Else
    - Trap into OS (not handled by hardware)
    - OS selects victim page in memory to replace
      - Write victim page out to disk if modified (use dirty bit in PTE)
    - OS reads referenced page from disk into memory
    - Page table is updated, present bit is set
    - Process continues execution

Swapping Policies

- Goal: Minimize number of page faults
  - Page faults require milliseconds to handle (reading from disk)
  - Implication: Plenty of time for OS to make good decision
- OS has two decisions
  - Page selection
    - When should a page (or pages) on disk be brought into memory?
  - Page replacement
    - Which resident page (or pages) in memory should be thrown out to disk?

Page Selection

- **Demand paging**:
  - Load page only when page fault occurs
  - Intuition: Wait until page must absolutely be in memory
  - When process starts: No pages are loaded in memory
  - Problems: Pay cost of page fault for every newly accessed page
- **Prepaging** (anticipatory, prefetching):
  - Load page before referenced
  - OS predicts future accesses (**oracle**) and brings pages into memory
  - Works well for some access patterns (e.g., sequential)

Page Selection

- Hints: Combine above with user-supplied hints about page references
  - User specifies: may need page in future, don’t need this page anymore, or sequential access pattern
  - Example: madvise() in Unix
    - `madvise - give advice about use of memory`

Page Replacement

- Which page in main memory should selected as victim?
  - Write out victim page to disk if modified (dirty bit set)
  - If victim page is not modified (clean), just discard
- **OPT**: Replace page **not used for longest time in future**
  - Advantages: Guaranteed to minimize number of page faults
  - Disadvantages: Requires that OS predict the future; Not practical, but good for comparison

Page Replacement

- **FIFO**: Replace page that has been in memory the longest
  - Intuition: First referenced long time ago, done with it now
  - Advantages: Fair: All pages receive equal residency; Easy to implement
  - Disadvantage: Some pages may always be needed
- **LRU**: Replace page not used for longest time in past
  - Intuition: Use past to predict the future
  - Advantages: With locality, LRU approximates OPT
  - Disadvantages:
    - Harder to implement, must track which pages have been accessed
    - Does not handle all workloads well

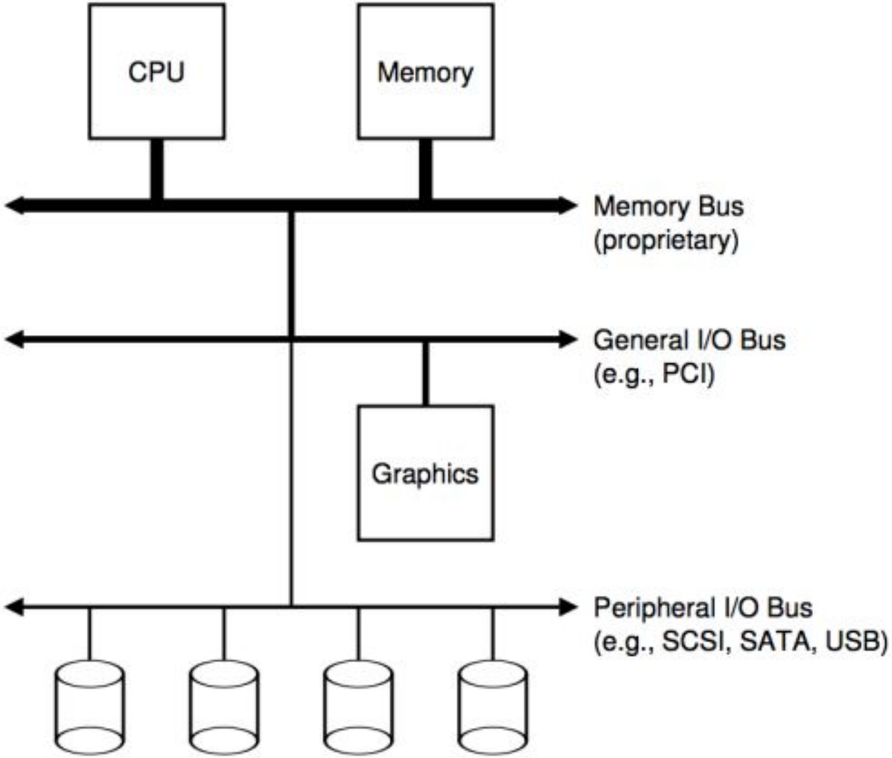
Comparison

- Add more physical memory, what happens to performance?
- LRU, OPT:
  - Guaranteed to have fewer (or same number of) page faults
  - Smaller memory sizes are guaranteed to contain a subset of larger memory sizes
  - Stack property: smaller cache always subset of bigger
- FIFO:
  - Usually have fewer page faults
  - Belady's anomaly: May actually have more page faults!

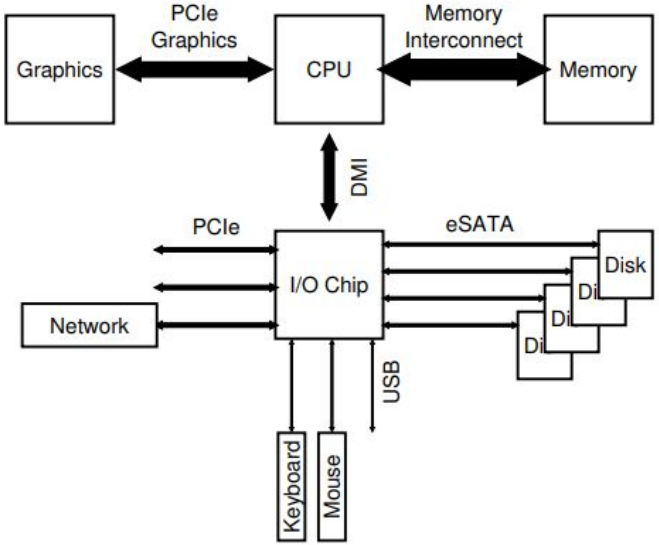
I/O Devices

- Three conceptual pieces
  - 1. Virtualization
    - Make each application believe it has each resource to itself CPU and Memory
  - 2. Concurrency
    - Provide mutual exclusion, ordering
  - 3. Persistence

System Architecture

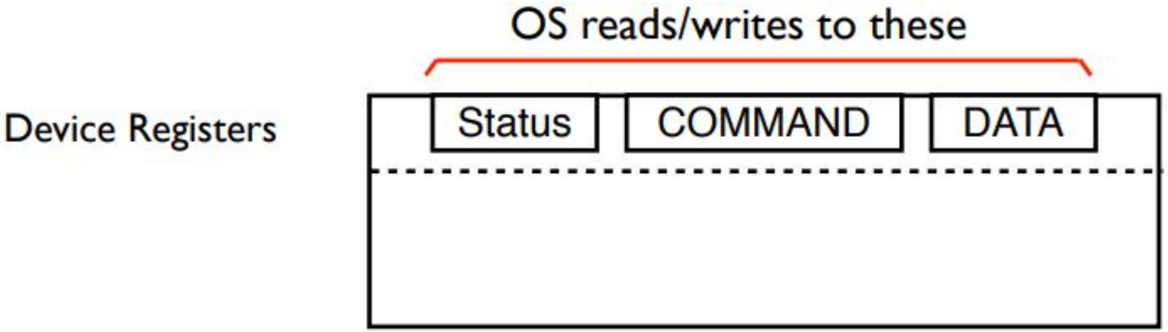


System Architecture



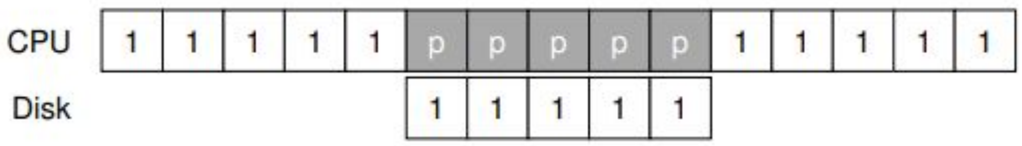
- Modern systems increasingly use specialized chipsets and faster point-to-point interconnects to improve performance

Canonical Device

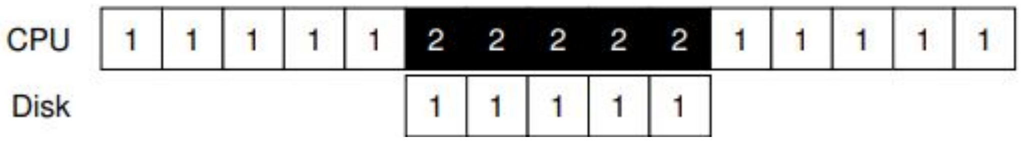



```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

Lowering CPU Overhead With Interrupts



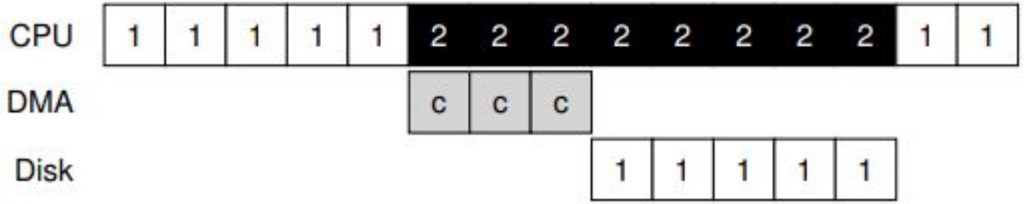
- P1 issues an I/O request
- CPU spins



- P1 issues an I/O request
- P2 on the CPU while the disk serves P1's request.

Data Movement with DMA

- P1 wishes to write some data to the disk
- CPU copies

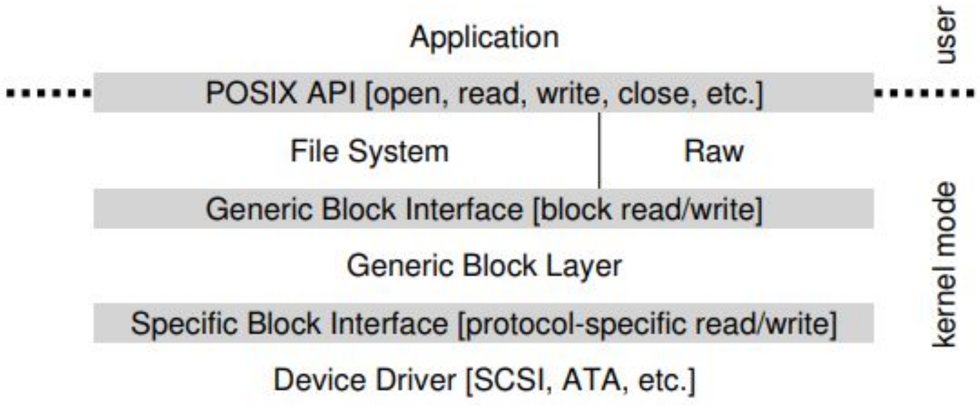


- DMA controller handles the copying
- DMV controller raises an interrupt when finishes

Programmed I/O VS. Directed Memory Access

- PIO (Programmed I/O):
  - CPU directly tells device what the data is
- DMA (Direct Memory Access):
  - Orchestrate transfers between devices and main memory without much CPU intervention
  - CPU leaves data in memory
  - Device reads data directly from memory

Device Drivers



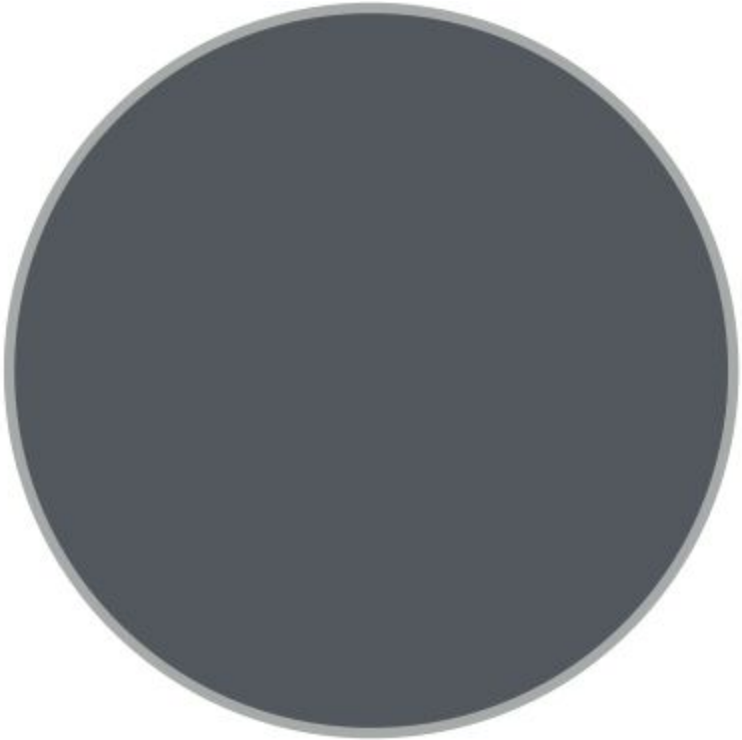
- a file system is completely oblivious to the specifics
- a raw interface to devices, which enables special applications to directly read and write blocks without using the file abstraction

Hard Disks

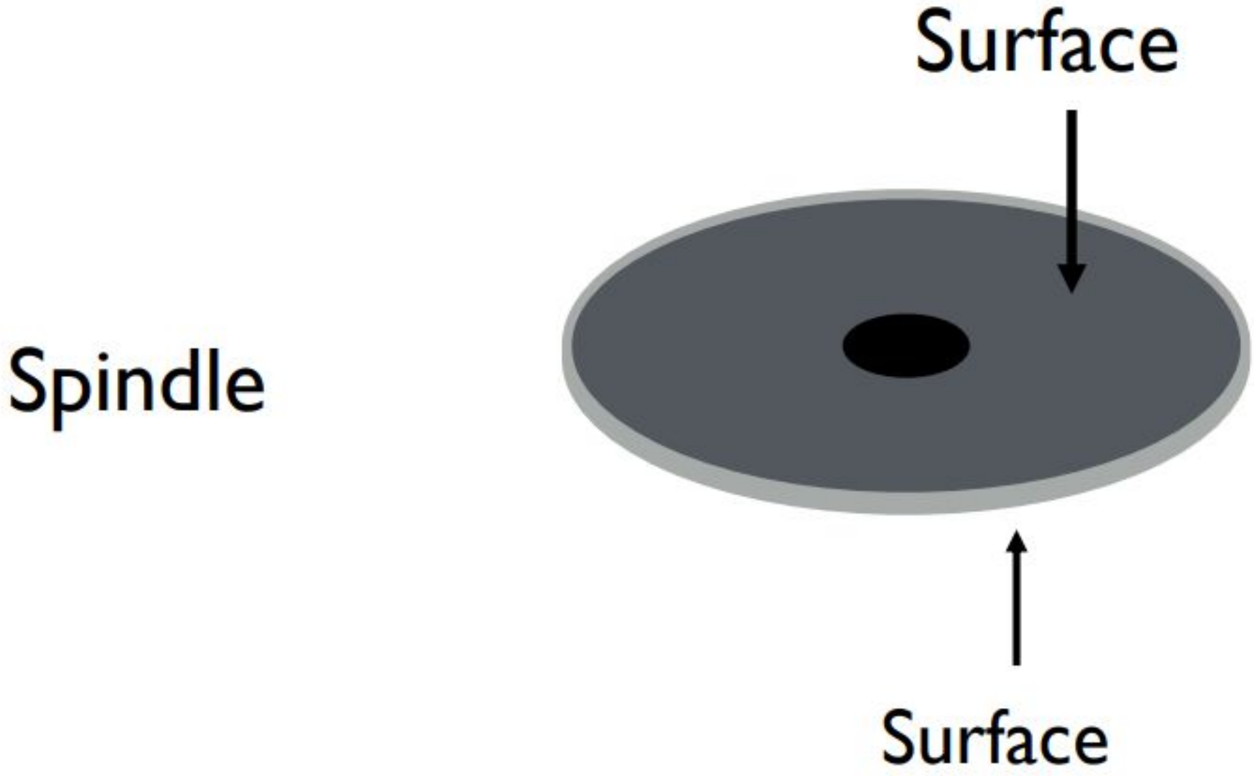
- Disk has a sector-addressable address space
  - Appears as an array of sectors
- Sectors are typically 512 bytes
- Main operations: reads + writes to sectors
- Mechanical and slow

Hard Disks

Platter

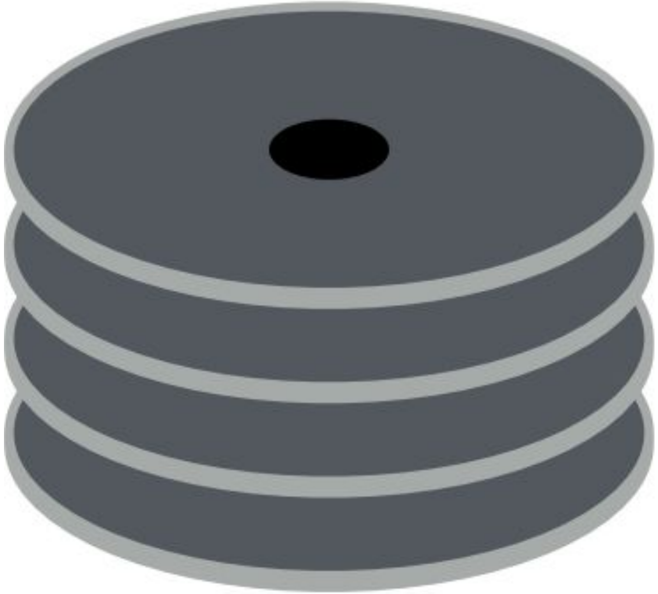


Hard Disks



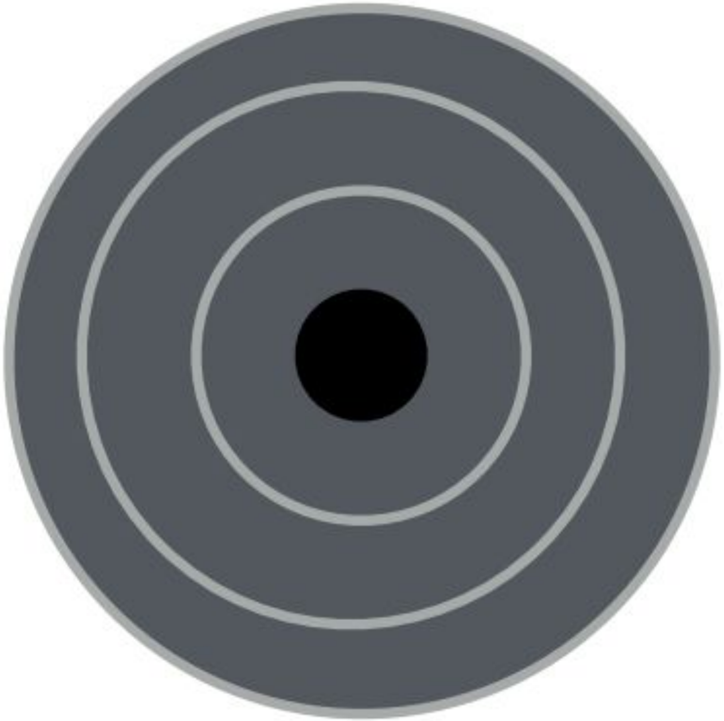
Hard Disks

- Motor connected to spindle spins platters
- Rate of rotation
  - RPM (rotation per minute)
- 10000 RPM ===>
  - 1 min == 60 sec == 60,000 ms
  - single rotation is 6 ms



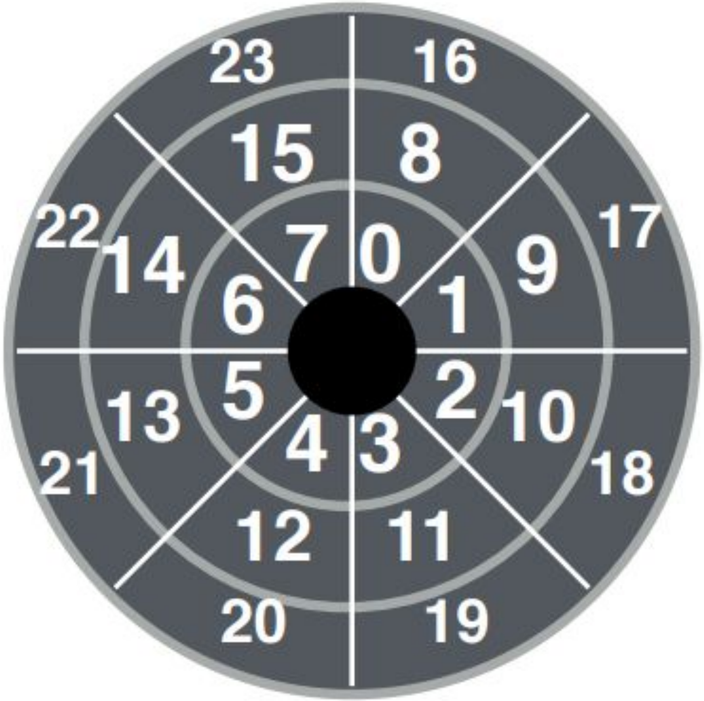
Hard Disks

- Surface is divided into rings: **tracks**
- Stack of tracks(across platters): **cylinder**



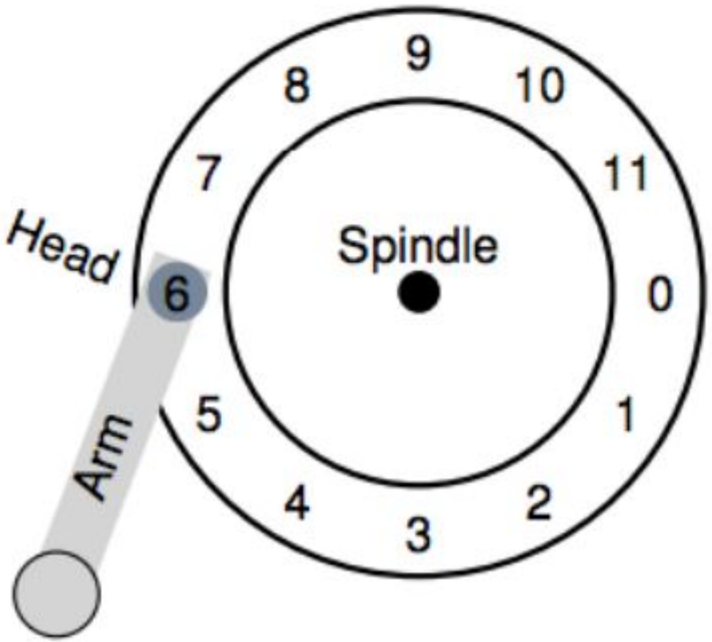
Hard Disks

- Tracks are further divided into numbered sectors



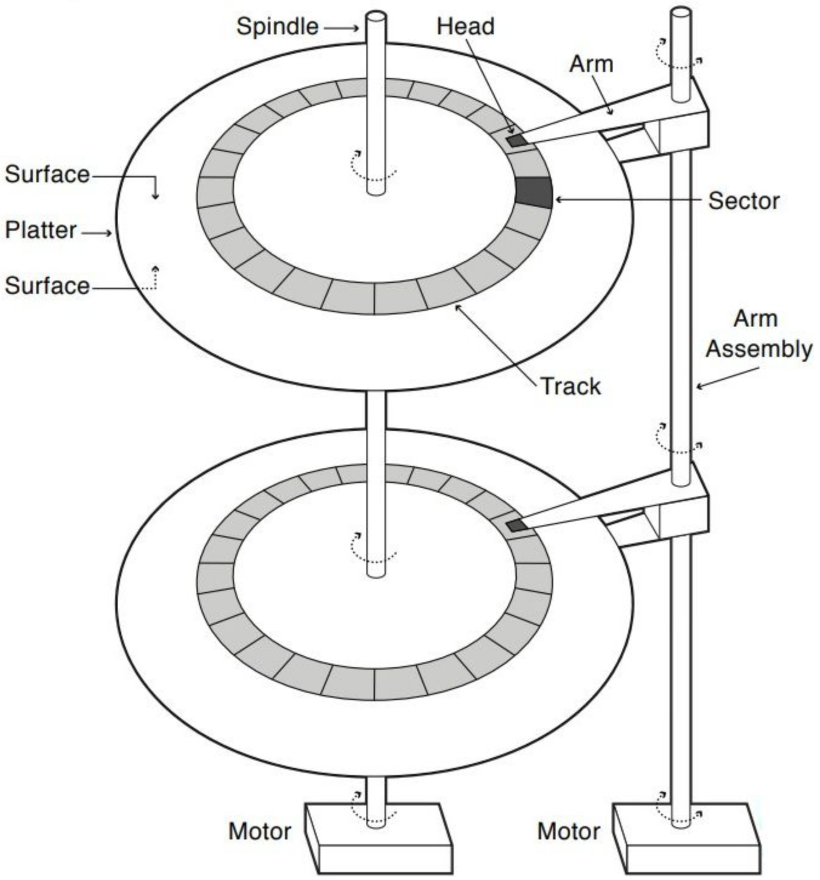
Hard Disks

- Heads on a moving arm can read from each surface.



Hard Disks

- Operations:
  - seek
  - read
  - write
- Must specify
  - cylinder # (distance from spindle)
  - head #
  - sector #
  - transfer size
  - memory address



THANK YOU!