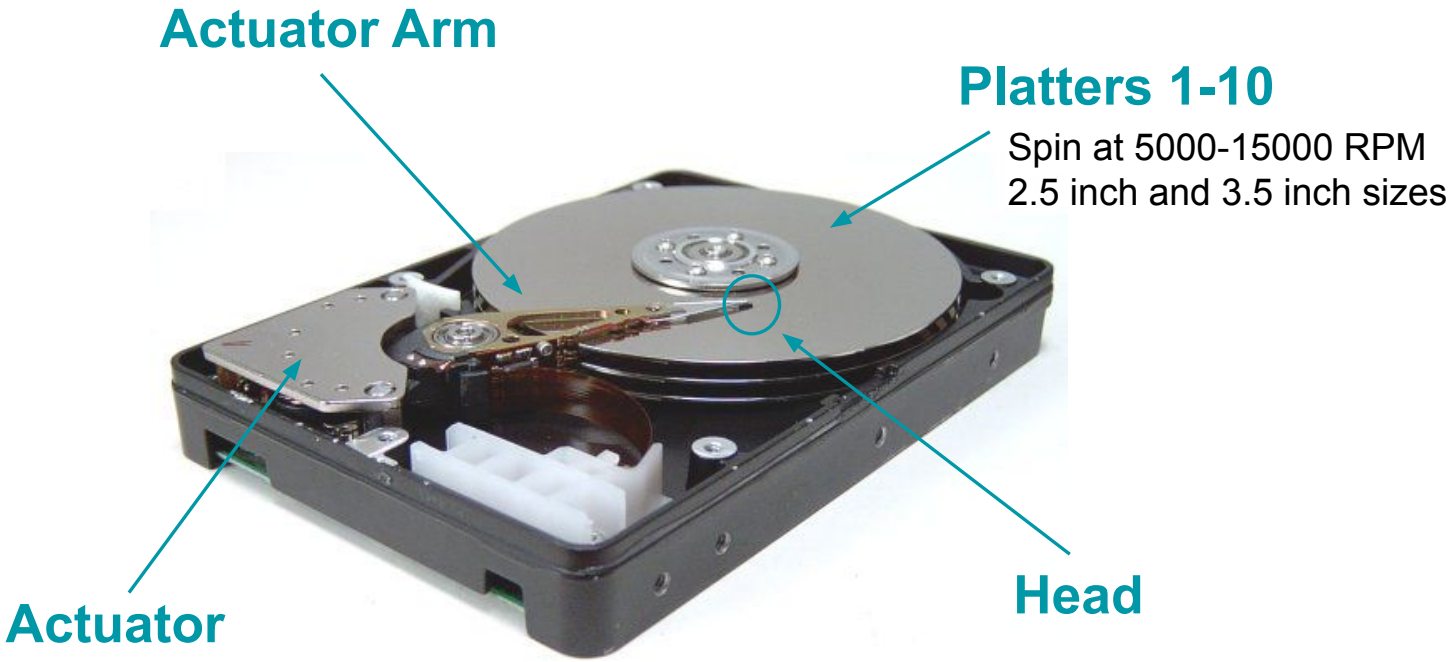
Magnetic Disks

Mendel Rosenblum

Magnetic Disks

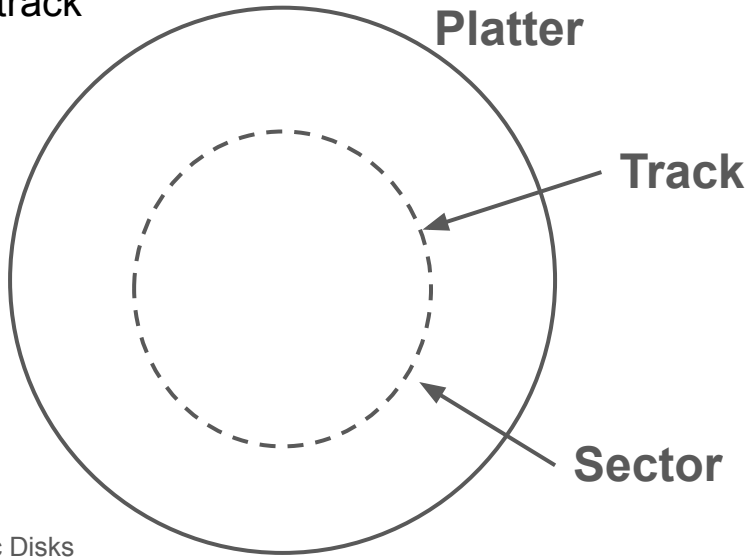
Optional readings: Operating Systems: Principles and Practice: Chapter 12.1

Hard Disk Drive



Organization of disk data

- Circular **tracks** corresponding to a particular position of a disk head
  - Typical density today: few hundred thousand to about 500,000 tracks per radial inch
- Tracks divided into 4096-byte *sectors*
  - Thousands to tens of thousands of sectors per track
- Typical total drive capacities: 500GB–30TB+
  - 1TB can store roughly 500 million pages of text

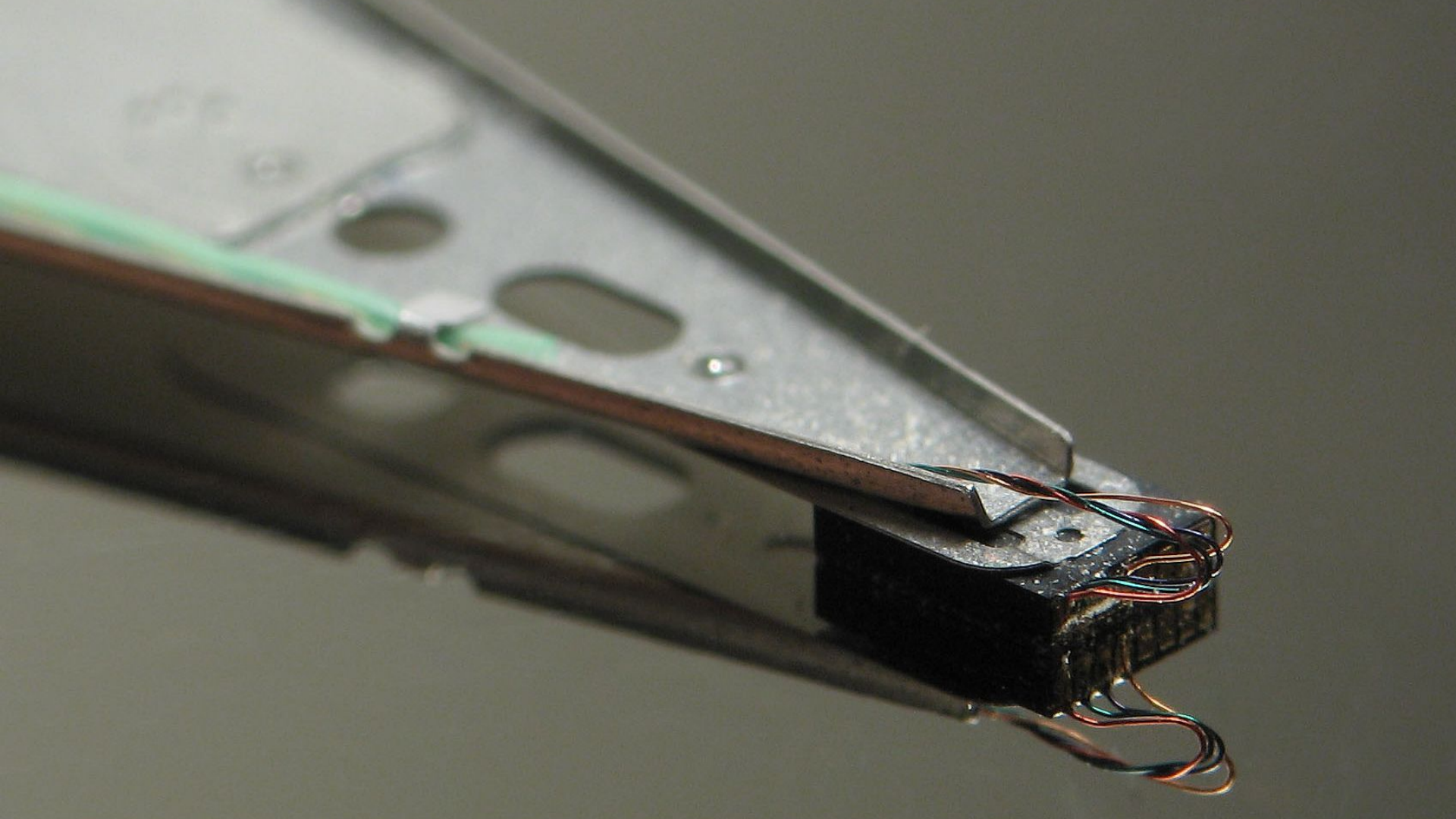


Reading and writing disks

1. **Seek**: move actuator arm to position heads over desired track
   - Typical seek time: 3-10ms
2. Select a particular head
3. ***Rotational latency:*** wait for desired sector to pass under the head
   - One-half disk rotation on average (4ms @ 7500RPM)
4. Transfer: read or write one or more sectors as they pass under the head
   - Typical transfer rates: 150–280 MBytes/sec
5. *Latency* refers to the sum of **seek** time plus **rotational latency**; typically 5-15ms

Disks are very high-tech devices

- Read-write head must get extremely close to the disk surface
  - But not touch
- Solution: the head “flies” above the disk on an ***air bearing*** created by the spinning platter dragging air with it.
  - Typical flying heights today are roughly **3–10 nm**
  - For comparison, a human hair is about **80,000–100,000 nm** thick
- Tiny contaminants can damage the disk surface or head
  - Older disks were vulnerable to catastrophic *head crashes* caused by dust or physical shock
- Modern drives are sealed in extremely clean enclosures
- When the drive powers down, heads are usually moved to a ***parking ramp*** off the platter surface
  - Older drives used a *landing zone* directly on the disk surface
- Laptop drives often included accelerometers that parked the heads if a fall was detected
- Modern HDDs also compensate for vibration and thermal expansion automatically



API for disks

- Modern disks export disk as an linear array of blocks (e.g 0, 1, … N)

```
read(startSector, sectorCount, physAddr)
```

```
write(startSector, sectorCount, physAddr)
```

In the old days the track and surface structure of the disk was visible to software:

```
read(track, surface, sector, sectorCount, physAddr)
```

Nowadays the track structure is hidden inside the disk:

- Inner tracks have fewer sectors than outer tracks
- If some sectors are bad, disk software automatically remaps them to spare sectors.

Communicating with I/O Devices

- Modern CPU/MMUs device physical address space:
  - DRAM memory: cache line load and stores
  - I/O devices: uncached load and stores
- I/O devices listen and respond to loads and stores at particular addresses
  - Called **device registers**
  - OS reads and writes device registers to control the device
- Called **memory-mapped input/output**

Device Registers

- Bits in device registers serve 3 purposes:
  - Parameters to device - CPU store instruction
    - e.g. number of first sector to read
  - Status bits from device - CPU load instruction
    - e.g. "operation complete" or "error occurred"
  - Control bits set by CPU - CPU store instruction
    - e.g. "start disk read" to initiate operations.
- Device registers don't behave like ordinary memory locations:
  - "Start operation" bit may always read as 0.
  - Bits may change without being written by CPU (e.g. "operation complete" bit).

Example disk read I/O operation

- CPU writes device registers to start operation (e.g., read sector)
  - Once operation has started, **`ready`** bit reads as zero
  - When operation is finished, device sets **`ready`** bit to one
- Remaining steps:
  - How does the OS know when the device is finished the operation?
  - Where does the bytes from the read sector go?

Device done signals to the Operating System

- **Poll**: Have OS spin reading the device's `ready` bit
  - CPU wastes time until the I/O finishes
- **Interrupt**: Have device force the CPU to trap
  - Allows CPU to do other work while devices are operating
  - Device needs attention (i..e. operation completes): it interrupts the CPU

Interrupt processing

- Device posts an interrupt to CPU
  - Trap: Switch CPU to particular address in the kernel
    - Interrupt trap vector also used by system calls, page faults, etc.
  - Operating system figures out which device interrupted, services that device
    - Acknowledge the interrupt and possibly start a new operation
  - After interrupt, return from trap
    - Execution resume from where left off
- Interrupts make the operating system much more efficient
  - Can keep many devices busy at the same time, while also running user code
  - Multi-core machines spread the interrupts around to balance loads
- Both CPU and devices after interrupt enable/disable flags

Data transfers to and from I/O device

- **Programmed Input/Output (PIO)**
  - Have CPU do loads or stores to device registers to transfer the data
  - Simple but very wasteful of CPU

- **Direct Memory Access (DMA)**
  - Device can transfer data to and from physical memory, without help from the CPU
  - CPU loads buffer address into a device register before starting operation
    - e.g., where to copy data read from disk
  - Device moves data directly to/from physical memory

Modern device interface

- Mainly use DMA with minimal programmed I/O for doorbell
- Device and OS share data structures in memory
  - Command queue for communication going from OS to device
  - Response queue for communication going from device to OS
- Example: Disk read of sector 32
  - OS build a command block in Command queue (e.g. read sector 32 into this address)
  - OS does a uncached write to a location notifying device of new command (i.e. doorbell)
  - Device uses DMA to read the command from the Command queue
  - Device uses DMA to write the sector bytes to the specified address
  - Device uses DMA to write the completion response in the Response queue
  - Device posts an interrupt to the CPU

Old-Style Magnetic Tape

System 360 Datacenter