# SECONDARY STORAGE DEVICES: MAGNETIC DISKS

## Contents of today's lecture:

- Secondary storage devices
- Organization of disks
- Organizing tracks by sector
- Organizing tracks by blocks
- Nondata overhead
- The cost of a disk access
- Disk as a bottleneck

**Reference:** Folk, Zoellick and Riccardi, File Structures, 1998. Sections 3.1.

# Secondary Storage Devices

Since secondary storage is different from main memory we have to understand how it works in order to do good file designs.

Two major types of storage devices:

- Direct Access Storage Devices (DASDs)
  - Magnetic Disks

    Hard Disks (high capacity, low cost per bit)

    Floppy Disks (low capacity, slow, cheap)
  - Optical Disks

    CD-ROM = Compact Disc, read-only memory

  (Read-only/write once, holds a lot of data, cheap reproduction)
- Serial Devices
  - Magnetic tapes (very fast sequential access)

# Organization of Disks

From now on we will use "disks" to refer to hard disks.
How disk drivers work

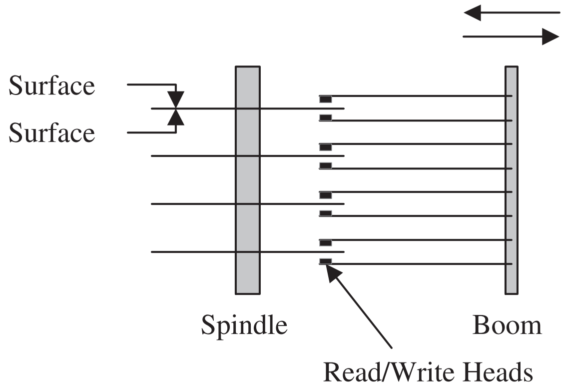
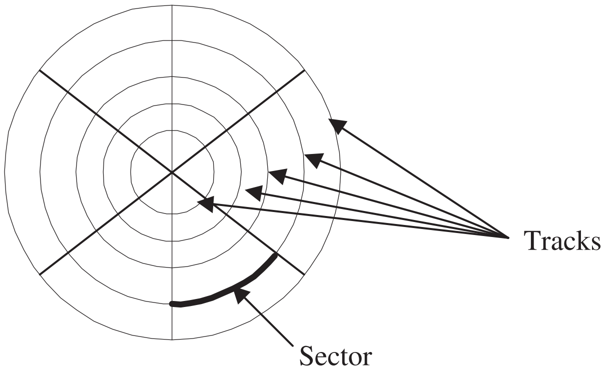



Figure 1: Disk drive with 4 platters and 8 surfaces

Looking at a surface:



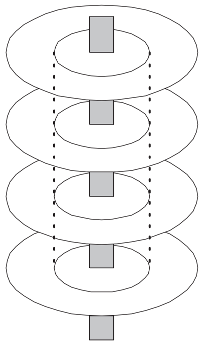

- Disk contains concentric **tracks**
- Tracks are divided into **sectors**
- A **sector** is the smallest addressable unit in a disk

When a program reads a byte from the disk, the operating system locates the surface, track and sector containing that byte, and reads the entire sector into a special area in main memory called buffer.

The bottleneck of a disk access is moving the read/write arm. So, it makes sense to store a file in tracks that are below/above each other in different surfaces, rather than in several tracks in the same surface.

**Cylinder** = the set of tracks on a disk that are directly above/below each other.

A cylinder



All the information on a cylinder can be accessed without **moving the read/write arm (Seeking)**.

number of cylinders = number of tracks in a surface
track capacity = number of sector per track x bytes per sector
cylinder capacity = number of surfaces x track capacity
drive capacity = number of cylinders x cylinder capacity

Solve the following problem:

File characteristics:

- Fixed-length records
- Number of records = 50,000 records
- Size of a record = 256 bytes

Disk characteristics:

- Number of bytes per sector = 512
- Number of sectors per track = 63
- Number of tracks per cylinder = 16
- Number of cylinders = 4092

**Q: How many cylinders are needed?**

A:
2 records per sector
$2 \times 63$ records per track
$16 \times 126 = 2{,}016$ records per cylinder
number of cylinders $= \frac{50{,}000}{2{,}016} \sim 24.8$ cylinders

Note: A disk might not have this many physically contiguous cylinders available. This file may be spread over hundreds of cylinders.

# Organizing Tracks by Sector

## The Physical Placement of Sectors

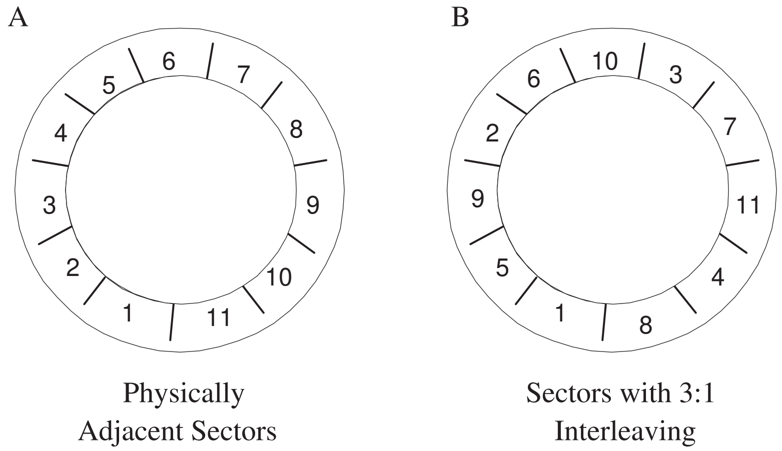



Figure 2: Interleaving

- Files are often stored in adjacent sectors, but "logically adjacent" does not necessarily mean "physically adjacent"
- **For some disks** : when it reads a sector, it takes some time to get ready to read the next sector. **Solution: interleaving**

**Example:** Suppose we need to read consecutively the sectors of a track in order: sector 1, sector 2, sector 3, ..., sector 11. Suppose the disk cannot read consecutive sectors.

- Without interleaving (Figure 2-A):
  How many revolutions to read the disk?
- With 3:1 interleaving (Figure 2-B):
  How many revolutions to read the disk?

(Now most disk controllers are fast and need no interleaving)

# Clusters, Extents and Fragmentation

The **file manager** is the part of the operating system responsible for managing files. The file manager maps the **logical parts** of the file into their **physical location**.

A **cluster** is a fixed number of contiguous sectors (if there is interleaving these sectors are not physically contiguous).
The **file manager** allocates an integer number of **clusters** to a file.

Ex: Sector size: 512 bytes,    Cluster size: 2 sectors

If a file contains 10 bytes, a cluster is allocated (1024 bytes). There may be unused space in the last cluster of a file. This unused space contributes to internal fragmentation.

**Clusters** are good since they improve sequential access: reading bytes sequentially from a cluster can be done in one revolution, seeking only once.
The file manager maintains a file allocation table (FAT) containing for each cluster in the file its location in disk.

An **extent** is a group of contiguous clusters. If a file is stored in a single extent then seeking (movement of read/write head) is done only once. If there is not enough contiguous clusters to hold a file, the file is divided into 2 or more extents.

# Fragmentation

1) Due to records not fitting exactly in a sector

Ex: Record size = 200 bytes, Sector size = 512 bytes
To avoid that a record span 2 sectors we can only store 2 records in this sector (112 bytes go unused per section ). This extra unused space contributes to fragmentation.
The alternative is to let a record span two sectors, but then two sectors must be read when we need to access this record.

2) Due to the use of clusters

If the file size is not a multiple of the cluster size, then the last cluster will be partially used.

**Choice of cluster size:** some operating systems allow the system administrator to choose cluster size.

When to use **large cluster size**?
When disk contain large files likely to be processed sequentially.
Ex: Updates in a master file of bank accounts (in batch mode)

What about **small cluster size**?
When disks contain small files and/or files likely to be accessed randomly
Ex: Online updates for airline reservation.

# Organizing Tracks by Blocks

- Disk tracks may be divided into user-defined **blocks** rather than into sectors.
  (Note: Here, "block" is not used as a synonym of sector or group of sectors)
- The amount transferred in a single I/0 operation can vary depending on the needs of the software designer (not hardware)
- A block is usually organized to contain an integral number of logical records.

**Blocking Factor** = number of records stored in each block in a file.

No internal fragmentation, no record spanning two blocks.

A block typically contains subblocks:

- Count subblock: contains the number of bytes in a block.
- Key subblock (optional): contains the key for the last record in the data subblock (the disk controller can search for key without loading it in main memory)
- Data subblock: contains the records in this block.

# Nondata Overhead

Amount of space used for extra stuff other than data.

**Sector-Addressable Disks** = At the beginning of each sector some info is stored, such as sector address, track address, condition (if sector is defective); there is some gap between sectors.

**Block-Organized Disks** subblocks and interblock gaps is part of the extra stuff; more nondata overhead than with sector-addressing.

Example:
Disk characteristics:
- Block-addressable disk drive
- Size of track = 20,000 bytes
- Nondata overhead per block = 300 bytes

File characteristics:
- Record size = 100 bytes

**Q: How many records can be stored per track for the following blocking factors?**

(1) Blocking factor = 10

(2) Blocking factor = 60

## Solution for example:

Disk characteristics:
- Block-addressable disk drive
- Size of track = 20,000 bytes
- Nondata overhead per block = 300 bytes

File characteristics:
- Record size = 100 bytes

(1) Blocking factor = 10

Size of data subblock = 1,000
Nondata overhead = 300
# of blocks that can fit in a track = $\lfloor \frac{20{,}000}{1{,}300} \rfloor = \lfloor 15.38 \rfloor = 15$ records
# of records per track = 150 records

(2) Blocking factor = 60

Size of data subblock = 6,000
Nondata overhead = 300
# of blocks that can fit in a track = $\lfloor \frac{20{,}000}{6{,}300} \rfloor = 3$ blocks
# of records per track = 180 records

# The Cost of a Disk Access

The time to access a sector in a track on a surface is divided into:

| **Time Component** | **Action** |
|---|---|
| Seek time | Time to move the read/write arm to the correct cylinder |
| Rotational delay (or latency) | Time it takes for the disk to rotate so that the desired sector is under the read/write head |
| Transfer time | Once the read/write head is positioned over the data, this is the time it takes for transferring data |

**Example: Disk Characteristics:**

- Average seek time = 8 msec
- Average rotational delay = 3 msec
- Maximum rotational delay = 6 msecs
- Spindle speed = 10,000 rpm
- Sectors per track = 170 sectors
- Sector size = 512 bytes

**What is the average time to read one sector ?**

Transfer time = revolution time/# sectors per track = (1/10,000)min/170 = (1/10,000 x 60)/170 secs = 6/170,000 secs = 6/170 msecs $\approx$ 0.035 msecs

Average total time = average seek + average rotational delay + transfer time = 8 + 3 + 0.035 = 11.035 msecs.

## Comparing sequential access to random access

Same disk as before:
- Average seek time = 8 msec
- Average rotational delay = 3 msec
- Maximum rotational delay = 6 msecs
- Spindle speed = 10,000 rpm
- Sectors per track = 170 sectors
- Sector size = 512 bytes

File characteristics:
- Number of records = 34,000
- Record size = 256 bytes
- File occupies 100 tracks dispersed randomly

a) Reading File Sequentially
- Average seek time = 8 msecs
- Average rotational delay = 3 msecs
- Average transfer time for 1 track = 60/10,000 = 6 msecs
- Average total time per track = 8 + 3 + 6 = 17 msecs
- Total time for file = 17 msecs x 100 = 1.7 secs

b) Reading a file accessing randomly each record
Average total time to read all records = Number of records x average time to read a sector = 34,000 x 11.035 msecs $\approx$ 371.1 secs

Note: typo in page 63 of the book in a similar calculation.

# Disk as a bottleneck

Processes are often **disk-bound**, i.e. network and CPU have to wait a long time for the disk to transmit data.

Various techniques to solve this problem

1. **Multiprocessing** (CPU works on other jobs while waiting for the disk)
2. **Disk Striping**
   Putting different blocks of the file in different drives.
   Independent processes accessing the same file may not interfere with each other (parallelism).
3. **RAID** (Redundant array of independent disks)
   Ex: in an eight-drive RAID the controller breaks each block into 8 pieces and place one in each disk drive (at the same position in each drive).
4. **RAM Disk** (memory disk)
   Piece of main memory used to simulate a disk (difference: speed and volatility). Used to simulate floppy disks.
5. **Disk Cache**
   Large block of memory configured to contain pages of data from a disk (typical size = 256 KB). When data is requested from disk, check cache first. If data is not there go to the disk and replace some page already in cache with page from disk containing the data.