

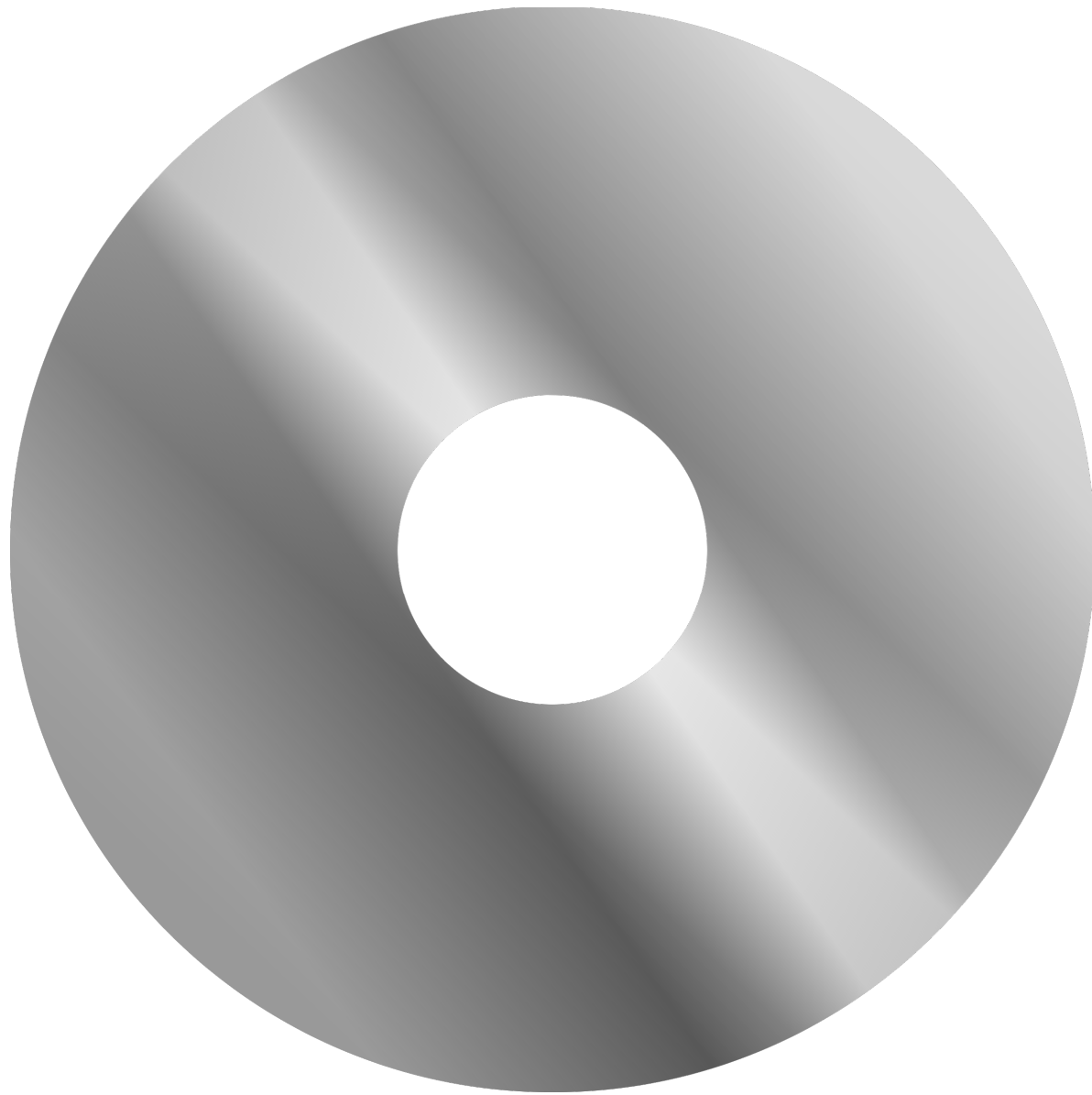
Storage and File Structure

Types of Storage Device

- Floppy diskette.
 - Hard drive.
 - Magnetic strip.
 - SuperDisk.
 - Tape cassette.
 - Zip diskette.
- **Solid-state storage** (sometimes abbreviated as **SSS**) is a type of non-volatile computer storage that stores and retrieves digital information using only electronic circuits, without any involvement of moving mechanical parts.

Magnetic Disks

- **Read-write head**
 - Positioned very close to the platter surface (almost touching it)
 - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
 - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**.
 - A sector is the smallest unit of data that can be read or written.
 - Sector size typically 512 bytes
 - Typical sectors per track: 500 (on inner tracks) to 1000 (on outer tracks)
- To read/write a sector
 - disk arm swings to position head on right track
 - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
 - multiple disk platters on a single spindle (1 to 5 usually)
 - one head per platter, mounted on a common arm.
- **Cylinder** i consists of i^{th} track of all the platters



BANERJEE; Dept of CSE;
partha.banerjee@juet.ac.in



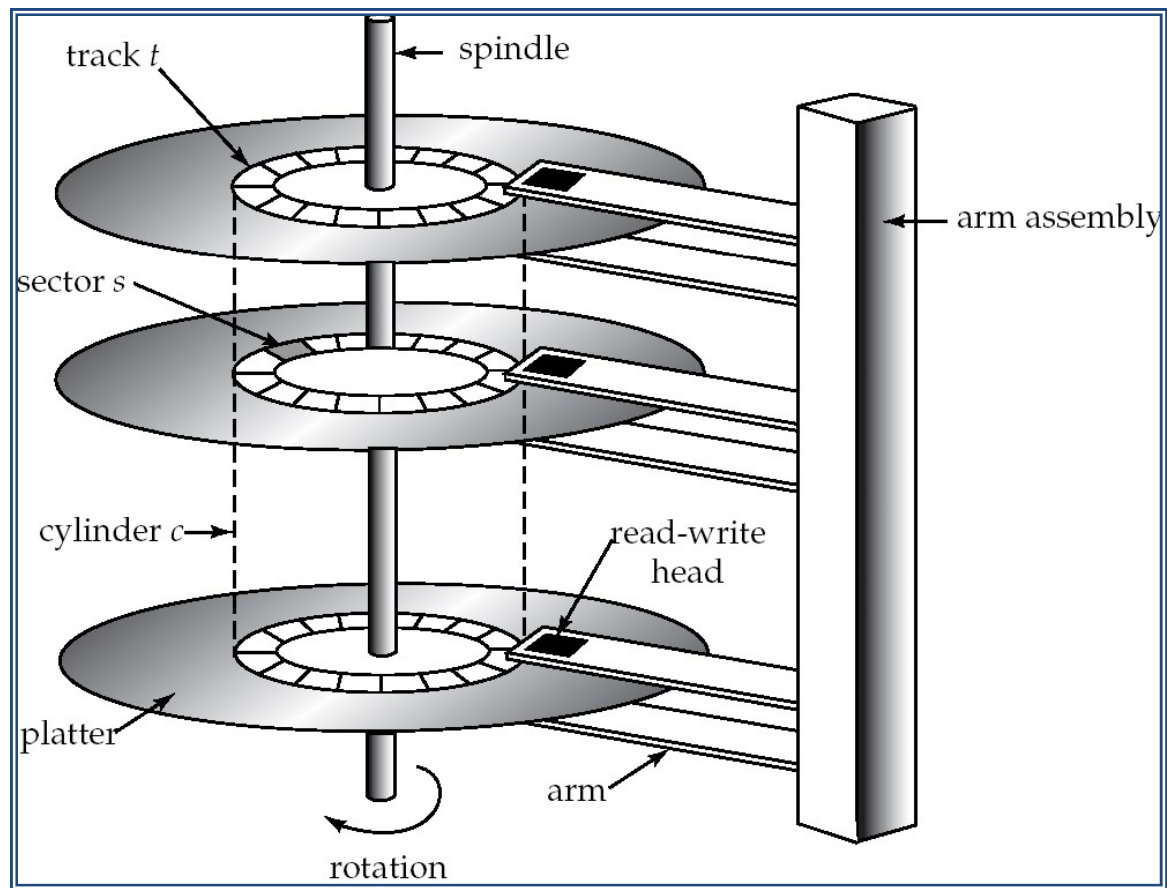
BANERJEE; Dept of CSE;
partha.banerjee@juet.ac.in

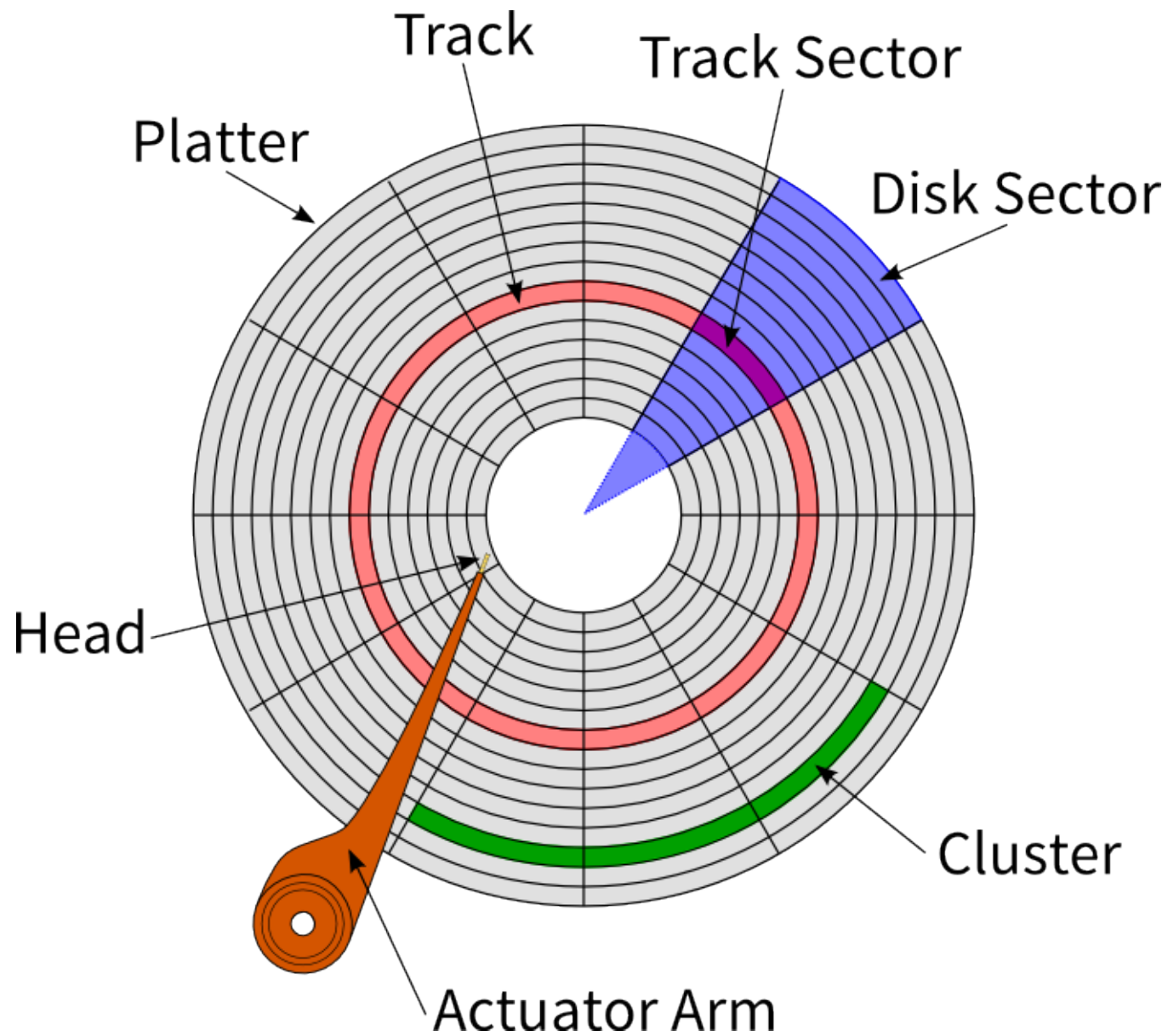
Performance Measures of Disks

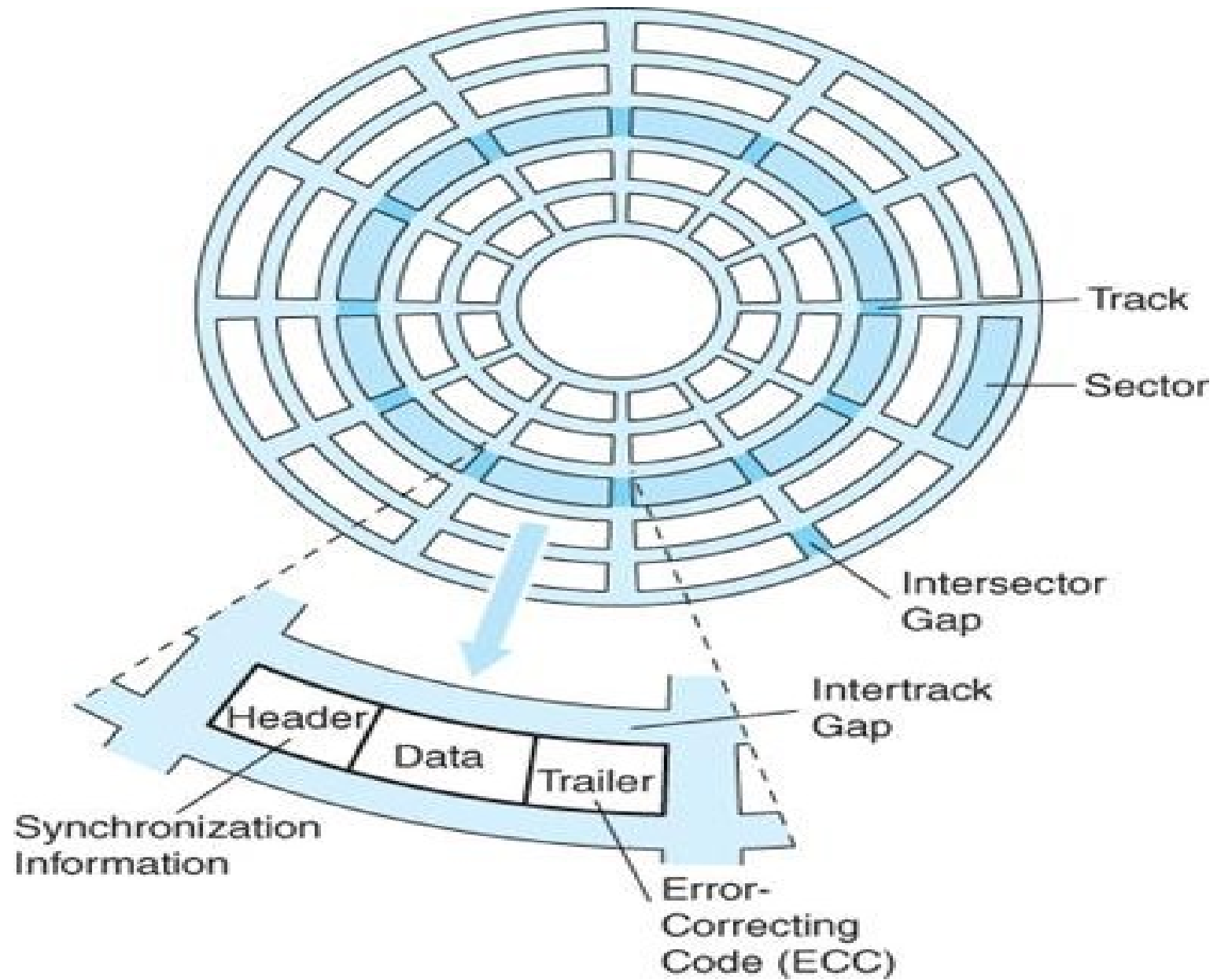
- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
 - **Seek time** – time it takes to reposition the arm over the correct track.
 - Average seek time is 1/2 the worst case seek time.
 - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
 - 4 to 10 milliseconds on typical disks
 - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
 - Average latency is 1/2 of the worst case latency.
 - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
 - 25 to 100 MB per second max rate, lower for inner tracks
 - Multiple disks may share a controller, so rate that controller can handle is also important
 - E.g. ATA-5: 66 MB/sec, SATA: 150 MB/sec, Ultra 320 SCSI: 320 MB/s
 - Fiber Channel (FC2Gb): 256 MB/s
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
 - Nowadays, typically 3 to 5 years

Magnetic Hard Disk Mechanism

- It is worth taking a look at how magnetic disks work.
 - After all they are the place where the **databases** are stored!





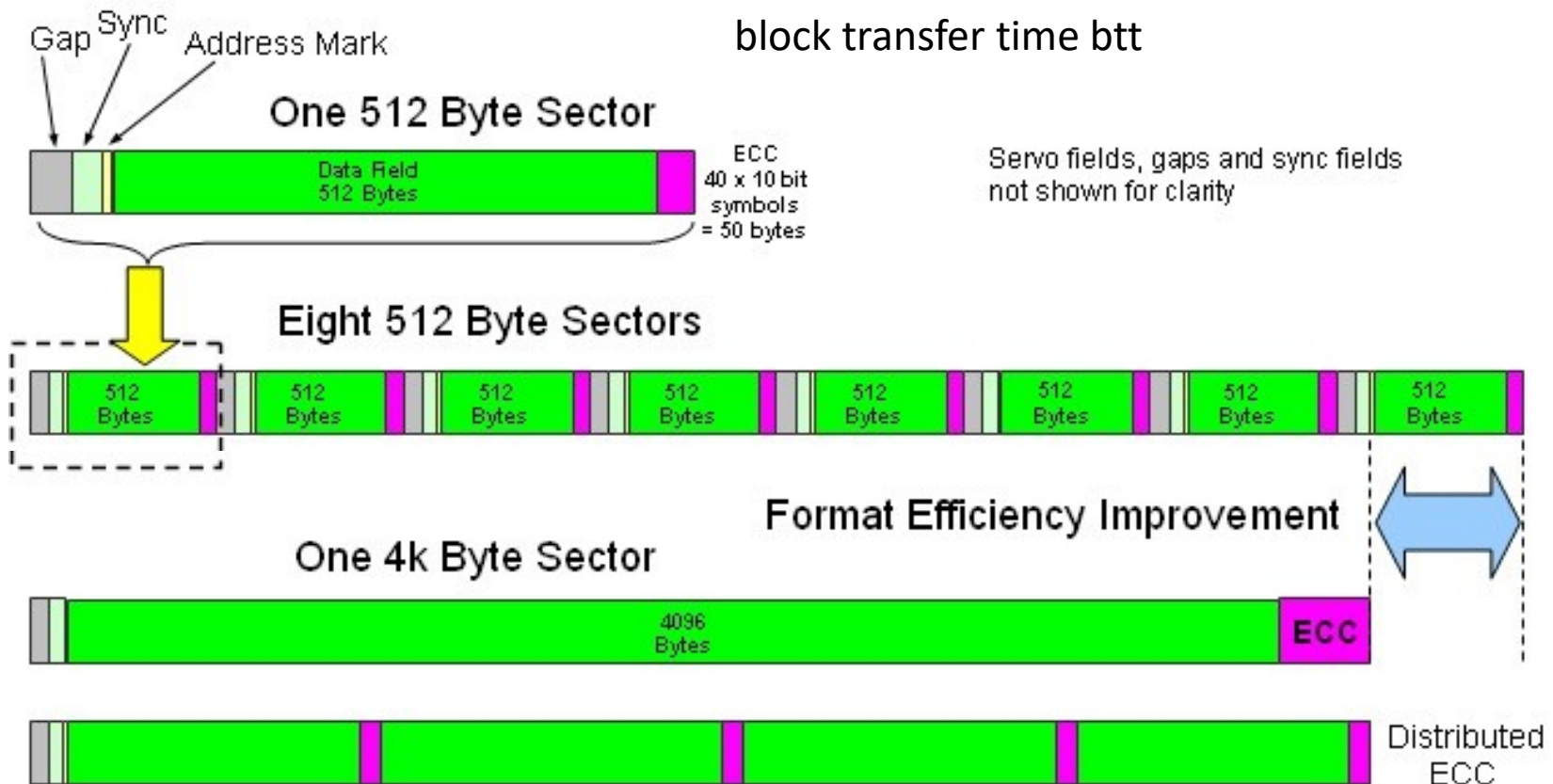


Optimization of Disk-Block Access

- **Block** – a contiguous sequence of sectors from a single track
 - data is transferred between disk and main memory in blocks
 - sizes range from 512 bytes to several kilobytes
 - Smaller blocks: more transfers from disk
 - Larger blocks: more space wasted due to partially filled blocks
 - Typical block sizes today range from 4 to 16 kilobytes
 - We'll see how this is important for database storage structure
- **Disk-arm-scheduling** algorithms order pending accesses to tracks so that disk arm movement is minimized
 - **elevator algorithm** : move disk arm in one direction (from outer to inner tracks or vice versa), processing next request in that direction, till no more requests in that direction, then reverse direction and repeat

Optimization of Disk Block Access (Cont.)

- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
 - E.g. Store related information on the same or nearby cylinders.
 - Files may get **fragmented** over time
 - E.g. if data is inserted to/deleted from the file
 - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
 - Sequential access to a fragmented file results in increased disk arm movement
 - Some systems have utilities to **defragment** the file system, in order to speed up file access



Q1. Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec.

1. What is the capacity of a track in bytes? What is the capacity of each surface?

What is the capacity of the disk?

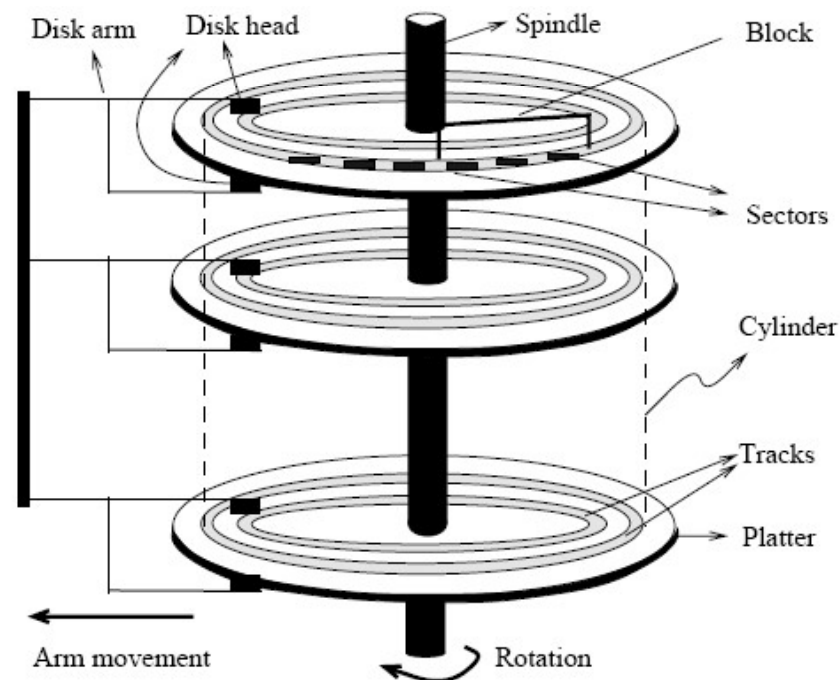
2. How many cylinders does the disk have?

3. Give examples of valid block sizes. Is 256 bytes a valid block size? 2048? 51200?

4. If the disk platters rotate at 5400 rpm (revolutions per minute), what is the maximum rotational delay?

a. What is the average rotational delay?

5. If one track of data can be transferred per revolution, what is the transfer rate?



Answer Q1.:

1. $\text{bytes/track} = \text{bytes/sector} * \text{sectors/track} = 512 * 50 = 25\text{K}$
 $\text{bytes/surface} = \text{bytes/track} * \text{tracks/surface} = 25\text{K} * 2000 = 50,000\text{K}$
 $\text{bytes/disk} = \text{bytes/surface} * \text{surfaces/disk} = 50,000\text{K} * 5 * 2 = 500,000\text{K}$
2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 is. 51200 is not a valid block size in this case because block size cannot exceed the size of a track, which is 25600 bytes.
4. If the disk platters rotate at 5400rpm, the time required for one complete rotation, which is the maximum rotational delay, is:
 $(1/5400) * 60 = 0.011$ seconds.
 - a. The average rotational delay is half of the rotation time, 0.0055 seconds.
5. The capacity of a track is 25K bytes. Since one track of data can be transferred per revolution, the data transfer rate is:
 $25\text{K}/0.011 = 2,250\text{Kbytes/second}$

Q2. Consider a hard disk that uses Advanced Format with sector size of 4096 bytes (4KB), 1000 tracks per surface, 50 sectors per track, five (5) double-sided platters, and average seek time of 8 msec. Suppose that the disk platters rotate at 7200 rpm (revolutions per minute).

Suppose also that a block (page) of 4096 bytes is chosen. Now, consider a file with 150,000 records of 100 bytes each that is to be stored on such a disk and that no record is allowed to span two blocks. Also, no block can span two tracks.

1. How many records fit onto a block?
2. How many blocks are required to store the entire file? If the file is arranged “sequentially on the disk, how many cylinders are needed?
3. How many records of 100 bytes each can be stored using this disk?
4. What time is required to read a file containing 100,000 records of 100 bytes each sequentially? You can make an assumption about how long it takes moving the heads from one cylinder to the next here.
5. What is the time required to read a file containing 100,000 records of 100 bytes each in a random order? To read a record, the block containing the record has to be fetched from the disk. Assume that each block request incurs the average seek time and rotational delay. You need also to include the transfer time for each block.

Answer Q2.:

1. $4096 / 100 = 40.96$. Which means that 40 records can fit in a block.

2. The file requires $150,000 \text{ records} / 40 \text{ records per block} = 3750 \text{ blocks}$ to be saved. In our case one block is equal to one sector(4KB)

1 track holds 50 blocks

10 tracks (two for each platter), which is one cylinder holds 500 blocks.

The file is 3750 blocks. Thus $3750/500 = 7.5$ cylinders. So 8.

3. 1 block can hold 40 records.

The disk has $50 * 1000 * 5$ number of blocks (blocks per track * tracks * surfaces)

Thus, this disk can store $40 * 50 * 1000 * 10 = 20\text{M}$ records.

4. A file containing 100,000 records of 100 bytes is stored in $100,000/40 = 2500 \text{ blocks}$.

From answer 1, we have that 1 cylinder holds 500 blocks. Our file will be saved in $2500/500 = 5 \text{ cylinders}$.

In order to read one track, we need $60\text{s}/7200(\text{rpm}) = 0.0083 \text{ seconds}$. In order to read one cylinder, we need $10 * 0.0083 = 0.083 \text{ seconds}$. We need to read 5 cylinders, so the total time will be $5 * 0.083 = 0.415 \text{ seconds}$. We assume that the time for moving from one cylinder to another is very small. The average seek time is $8\text{msec} = 0.008 \text{ seconds}$ and the average rotational delay is 0.00415 seconds . Therefore, the total time is $0.415 + 0.00415 + 0.008 = 0.42715 \text{ seconds}$.

Answer Q2.: Continued..

5. A file containing 100,000 records of 100 bytes is stored in $100,000/40 = 2500$ blocks
For any block of data, average access time = avg seek time + avg rotational delay +
transfer time Average seek time = 0,008s and average rotational-delay = 0,00415s .

To find transfer time:

Each track has 50 blocks. To read all 50 blocks it takes 0,0083 seconds. To read just one
takes $0,0083/50 = 0,000166$ seconds

As a result, the total time is $(0,008 + 0,00415 + 0,000166) * 2500 = 30.79$ seconds.

Q3. Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size $B=512$ bytes, interblock gap size $G=128$ bytes, number of blocks per track=20, number of tracks per surface=400. A disk pack consists of 15 double-sided disks.

- (a) What is the total capacity of a track and what is its useful capacity (excluding interblock gaps)?
- (b) How many cylinders are there?
- (c) What is the total capacity and the useful capacity of a cylinder?
- (d) What is the total capacity and the useful capacity of a disk pack?
- (e) Suppose the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what is the transfer rate in bytes/msec and the block transfer time btt in msec? What is the average rotational delay r_d in msec? What is the bulk transfer rate (see Appendix B)?
- (f) Suppose the average seek time is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block given its block address?
- (g) Calculate the average time it would take to transfer 20 random blocks and compare it with the time it would take to transfer 20 consecutive blocks using double buffering to save seek time and rotational delay.

Answer Q3.:

(a) Total track size = $20 * (512+128) = 12800$ bytes = 12.8 Kbytes

Useful capacity of a track = $20 * 512 = 10240$ bytes = 10.24 Kbytes

(b) Number of cylinders = number of tracks = 400

(c) Total cylinder capacity = $15*2*20*(512+128) = 384000$ bytes = 384 Kbytes

Useful cylinder capacity = $15 * 2 * 20 * 512 = 307200$ bytes = 307.2 Kbytes

(d) Total capacity of a disk pack = $15 * 2 * 400 * 20 * (512+128)$
= 153600000 bytes = 153.6 Mbytes

Useful capacity of a disk pack = $15 * 2 * 400 * 20 * 512 = 122.88$ Mbytes

(e) Transfer rate $tr = (\text{total track size in bytes}) / (\text{time for one disk revolution in msec})$

$tr = (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512$ bytes/msec

block transfer time $btt = B / tr = 512 / 512 = 1$ msec

average rotational delay $rd = (\text{time for one disk revolution in msec}) / 2 = 25 / 2$
= 12.5 msec

bulk transfer rate $btr = tr * (B / (B+G)) = 512 * (512 / 640) = 409.6$ bytes/msec

(f) average time to locate and transfer a block = $s + rd + btt = 30 + 12.5 + 1 = 43.5$ msec

(g) time to transfer 20 random blocks = $20 * (s + rd + btt) = 20 * 43.5 = 870$ msec

time to transfer 20 consecutive blocks using double buffering = $s + rd + 20 * btt$
= $30 + 12.5 + (20 * 1) = 62.5$ msec

(a more accurate estimate of the latter can be calculated using the bulk transfer rate as follows: time to transfer 20 consecutive blocks using double buffering
= $s + rd + ((20 * B) / btr) = 30 + 12.5 + (10240 / 409.6) = 42.5 + 25 = 67.5$ msec)

Q4. A file has $r=20000$ STUDENT records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), MAJORDEPTCODE (4 bytes), MINORDEPTCODE (4 bytes), CLASSCODE (4 bytes, integer), and DEGREEPROGRAM (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk whose parameters are given in Exercise 4.18.

- (a) Calculate the record size R in bytes.
- (b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.
- (c) Calculate the average time it takes to find a record by doing a linear search on the file if (i) the file blocks are stored contiguously and double buffering is used, and (ii) the file blocks are not stored contiguously.
- (d) Assume the file is ordered by SSN; calculate the time it takes to search for a record given its SSN value by doing a binary search.

Answer Q4.:

(a) $R = (30 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 4 + 3) + 1 = 113$ bytes

(b) $bfr = \text{floor}(B / R) = \text{floor}(512 / 113) = 4$ records per block

$b = \text{ceiling}(r / bfr) = \text{ceiling}(20000 / 4) = 5000$ blocks

(c) For linear search we search on average half the file blocks = $5000/2 = 2500$ blocks.

i. If the blocks are stored consecutively, and double buffering is used, the time to read 2500 consecutive blocks

$$= s + rd + (2500 * (B / btr)) = 30 + 12.5 + (2500 * (512 / 409.6))$$

$$= 3167.5 \text{ msec} = 3.1675 \text{ sec}$$

(a less accurate estimate is $= s + rd + (2500 * btt) = 30 + 12.5 + 2500 * 1 = 2542.5 \text{ msec}$)

ii. If the blocks are scattered over the disk, a seek is needed for each block, so the time

$$\text{is: } 2500 * (s + rd + btt) = 2500 * (30 + 12.5 + 1) = 108750 \text{ msec} = 108.75 \text{ sec}$$

(d) For binary search, the time to search for a record is estimated as:

$$\text{ceiling}(\log_2 b) * (s + rd + btt)$$

$$= \text{ceiling}(\log_2 5000) * (30 + 12.5 + 1) = 13 * 43.5 = 565.5 \text{ msec} = 0.5655 \text{ sec}$$

Some Formulas

- Record length R = Sum of the sizes of all fields (in Bytes)
- Blocking factor $bfr = \text{floor}(B/R)$ [the floored value of blocks per record]
- Number of blocks needed for file = $\text{ceiling}(r/bfr)$
- Index record size $R_i = ((\text{Size of Key field}) + P)$
- Index blocking factor $bfr_i = fo = \text{floor}(B/R_i)$

INDEXING STRUCTURES FOR FILES

Q5. Consider a disk with block size $B=512$ bytes. A block pointer is $P=6$ bytes long, and a record pointer is $P R =7$ bytes long. A file has $r=30,000$ EMPLOYEE records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), DEPARTMENTCODE (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), JOBCODE (4 bytes), SALARY (4 bytes, real number). An additional byte is used as a deletion marker.

- (a) Calculate the record size R in bytes.
- (b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.
- (c) Suppose the file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate
 - i. the index blocking factor $bfri$ (which is also the index fan-out fo);
 - ii. the number of first-level index entries and the number of first-level index blocks;
 - iii. the number of levels needed if we make it into a multi-level index;
 - iv. the total number of blocks required by the multi-level index; and
 - v. the number of block accesses needed to search for and retrieve a record from the file—given its SSN value—using the primary index.

Answer Q5.:

(a) Record length $R = (30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4) + 1 = 115$ bytes

(b) Blocking factor $bfr = \text{floor}(B/R) = \text{floor}(512/115) = 4$ records per block

Number of blocks needed for file = $\text{ceiling}(r/bfr) = \text{ceiling}(30000/4) = 7500$

(c) i. Index record size $R_i = (V \text{ SSN} + P) = (9 + 6) = 15$ bytes

Index blocking factor $bfri = fo = \text{floor}(B/R_i) = \text{floor}(512/15) = 34$

ii. Number of first-level index entries $r1 = \text{number of file blocks } b = 7500$ entries

Number of first-level index blocks $b1 = \text{ceiling}(r1/bfri) = \text{ceiling}(7500/34) = 221$

Blocks

iii. We can calculate the number of levels as follows: Number of second-level

index entries $r2 = \text{number of first-level blocks } b1 = 221$ entries

Number of second-level index blocks $b2 = \text{ceiling}(r2/bfri) = \text{ceiling}(221/34) = 7$

Blocks

Number of third-level index entries $r3 = \text{number of second-level index blocks } b2 =$

7 entries

Number of third-level index blocks $b3 = \text{ceiling}(r3/bfri) = \text{ceiling}(7/34) = 1$

Since the third level has only one block, it is the top index level. Hence, the index has $x = 3$ levels

iv. Total number of blocks for the index $bi = b1 + b2 + b3 = 221 + 7 + 1 = 229$ blocks

v. Number of block accesses to search for a record = $x + 1 = 3 + 1 = 4$

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records*. A record is a sequence of fields.
 - First approach:
 - assume record size is fixed
 - each file has records of one particular type only
 - different files are used for different relations
- This case is the easiest to implement.

Fixed-Length Records

- Simple approach:
 - Store record i starting from byte $n * (i - 1)$, where n is the size of each record.
 - Record access is simple but records may cross blocks!
 - Modification: do not allow records to cross block boundaries
- Deletion of record i :
alternatives:
 - move records $i + 1, \dots, n$ to $i, \dots, n - 1$
 - move record n to i
 - do not move records, but link all free records on a *free list*

| | | | |
|----------|-------|------------|-----|
| record 0 | A-102 | Perryridge | 400 |
| record 1 | A-305 | Round Hill | 350 |
| record 2 | A-215 | Mianus | 700 |
| record 3 | A-101 | Downtown | 500 |
| record 4 | A-222 | Redwood | 700 |
| record 5 | A-201 | Perryridge | 900 |
| record 6 | A-217 | Brighton | 750 |
| record 7 | A-110 | Downtown | 600 |
| record 8 | A-218 | Perryridge | 700 |

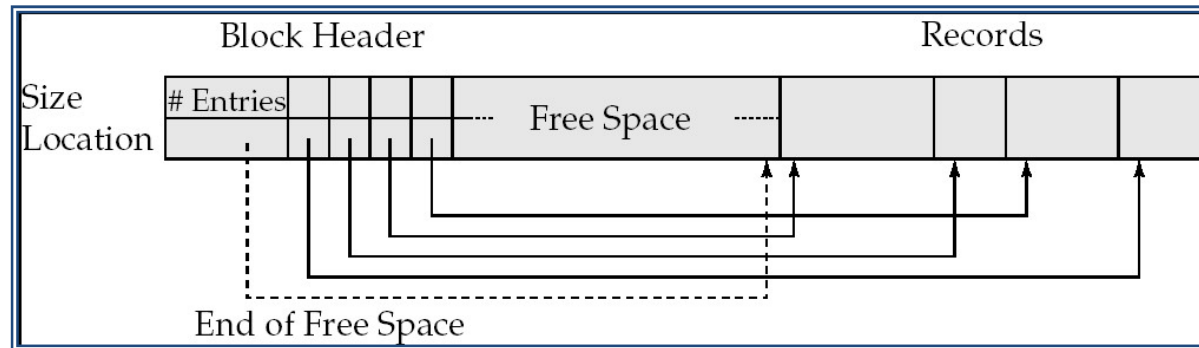
Free Lists

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as **pointers** since they “point” to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)

| | | | | | |
|----------|-------|------------|-----|--|--|
| header | | | | | |
| record 0 | A-102 | Perryridge | 400 | | |
| record 1 | | | | | |
| record 2 | A-215 | Mianus | 700 | | |
| record 3 | A-101 | Downtown | 500 | | |
| record 4 | | | | | |
| record 5 | A-201 | Perryridge | 900 | | |
| record 6 | | | | | |
| record 7 | A-110 | Downtown | 600 | | |
| record 8 | A-218 | Perryridge | 700 | | |

```
graph TD
    H[header] --> R0[record 0]
    R0 --> R1[record 1]
    R1 --> R2[record 2]
    R2 --> R3[record 3]
    R3 --> R4[record 4]
    R4 --> R5[record 5]
    R5 --> R6[record 6]
    R6 --> R7[record 7]
    R7 --> R8[record 8]
    R8 --> End[ ]
```

Variable-Length Records: Slotted Page Structure



- **Slotted page** are usually the size of a block
- Header contains:
 - number of record entries
 - end of free space in the block
 - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- (Other) pointers should not point directly to record — instead they should point to the entry for the record in header.

Variable-Length Records

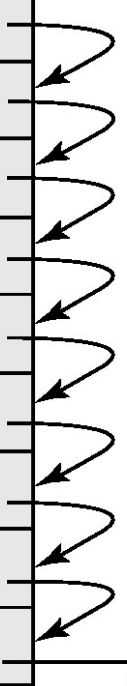
- Variable-length records arise in database systems in several ways:
 - Storage of multiple record types in a file.
 - Record types that allow variable lengths for one or more fields.
- If slotted pages are the size of a block, the issue of records spanning over more than one block is eliminated
- This limits the size of records in a database, which is usually the (default) case
 - There are special types for big records, that are treated differently (remember the **clob**s and **blob**s in Oracle?)

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
 - Motivation: store related records on the same block to minimize I/O
- The choice of a proper organization of records in a file is important for the efficiency of real databases!

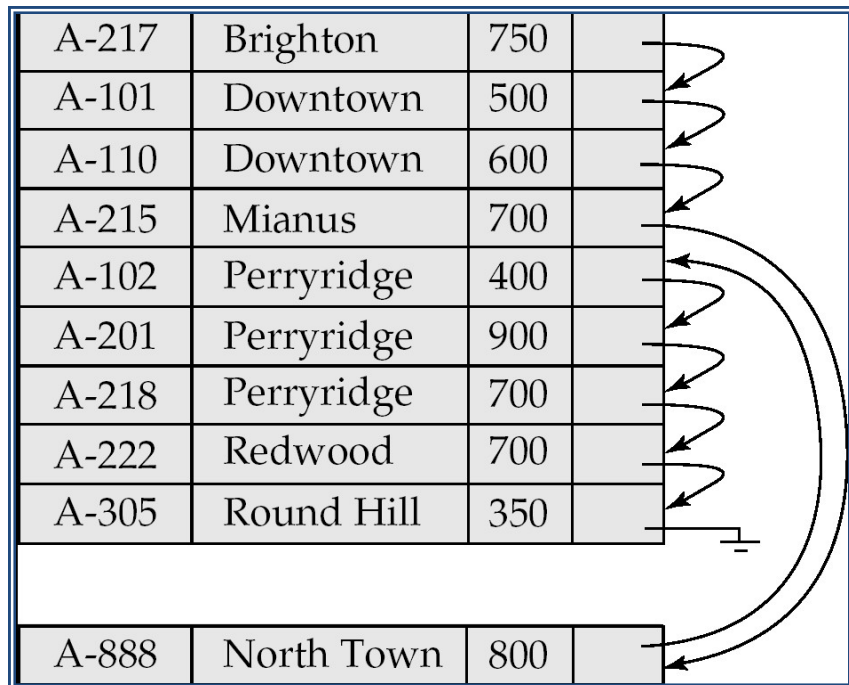
Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a **search-key**

| | | | | |
|-------|------------|-----|--|--|
| A-217 | Brighton | 750 | |  |
| A-101 | Downtown | 500 | | |
| A-110 | Downtown | 600 | | |
| A-215 | Mianus | 700 | | |
| A-102 | Perryridge | 400 | | |
| A-201 | Perryridge | 900 | | |
| A-218 | Perryridge | 700 | | |
| A-222 | Redwood | 700 | | |
| A-305 | Round Hill | 350 | | |

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion – locate the position where the record is to be inserted
 - if there is free space insert there
 - if no free space, insert the record in an **overflow block**
 - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization.

| <i>customer_name</i> | <i>account_number</i> |
|----------------------|-----------------------|
| Hayes | A-102 |
| Hayes | A-220 |
| Hayes | A-503 |
| Turner | A-305 |

| <i>customer_name</i> | <i>customer_street</i> | <i>customer_city</i> |
|----------------------|------------------------|----------------------|
| Hayes | Main | Brooklyn |
| Turner | Putnam | Stamford |

Multitable Clustering File Organization (cont.)

Multitable clustering organization of

customer

| | | |
|--------|--------|----------|
| Hayes | Main | Brooklyn |
| Hayes | A-102 | |
| Hayes | A-220 | |
| Hayes | A-503 | |
| Turner | Putnam | Stamford |
| Turner | A-305 | |

- good for queries involving *depositor* ~~customer~~, and for queries involving one single customer and his accounts
- bad for queries involving only customer
 - but one can add pointer chains to link records of a particular relation
- results in variable size records