

# **Database Development and Design (CSE210)**

## **Lecture 2: Physical Storage Media and File Organisation**

Wei Wang

CSSE

# Learning Outcomes

- Overview of Physical Storage Media
- Magnetic Disks
- Storage Access
- File Organisation
- Organisation of Records in Files

# Building a Database: High-Level

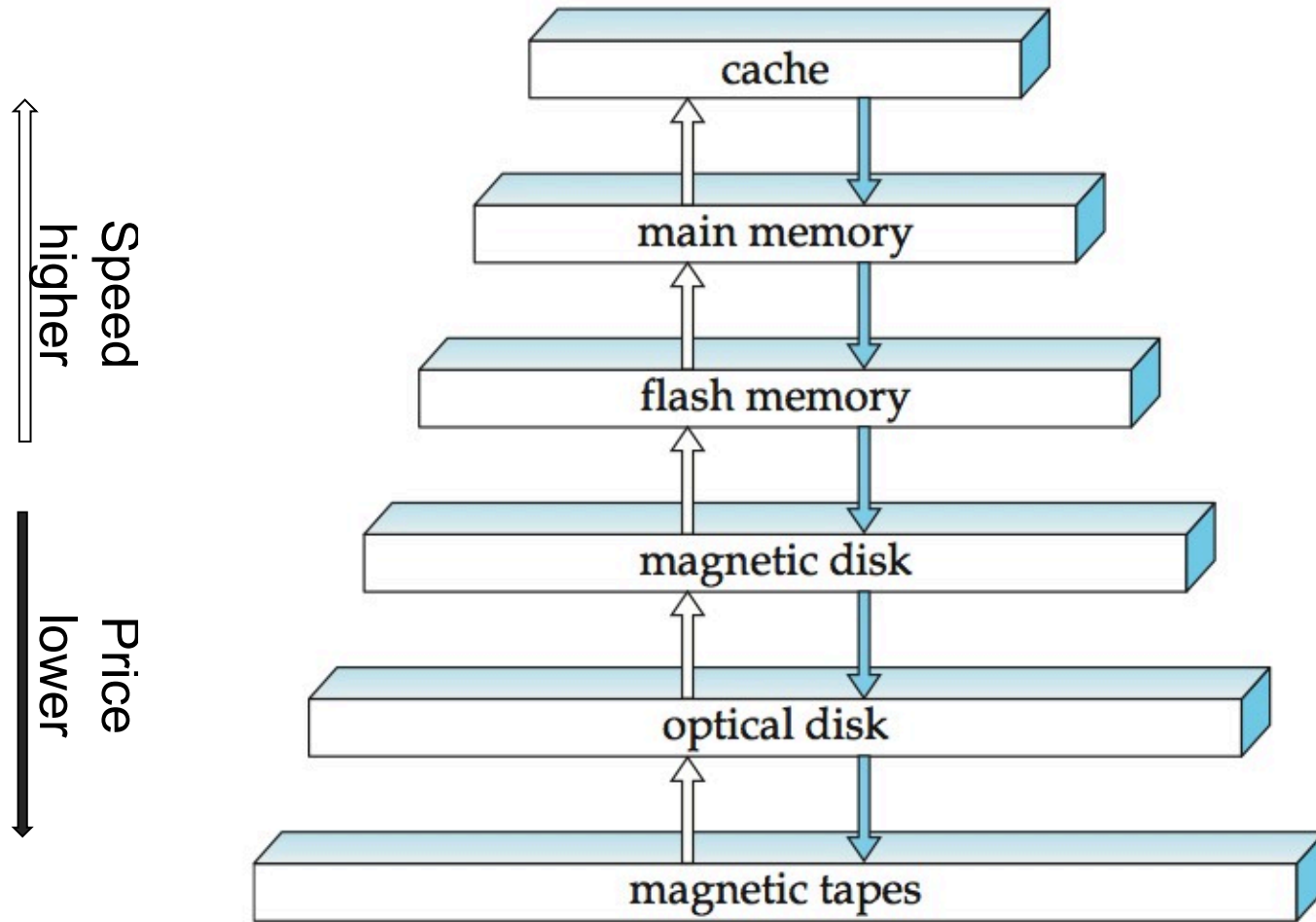
- Design conceptual schema using a data model.
  - Design relational schema, e.g., student(sid, name)
- Data Definition Language (DDL)
  - `CREATE TABLE student (sid char(8) primary key, name varchar(32))`
- Data Manipulation Language (DML)
  - `INSERT INTO student VALUES ('00112233', 'Paul')`

# Where and How is all the information stored?



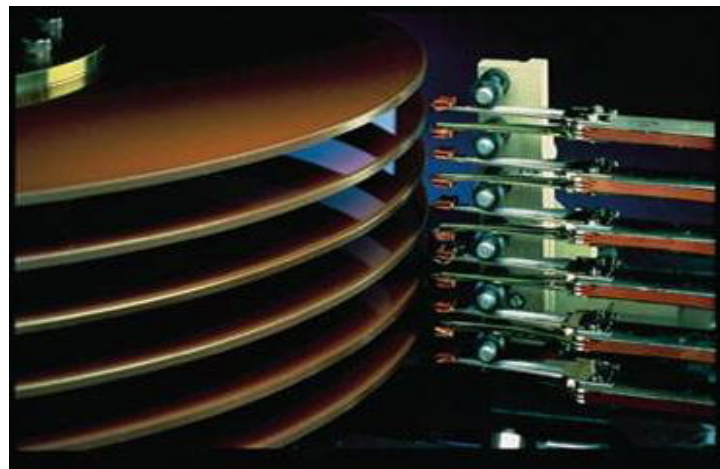
- Metadata: tables, attributes, data types, constraints
- Data (records)
- Transaction logs
- Indices
- Statistical data
- etc...

# Physical Storage Media

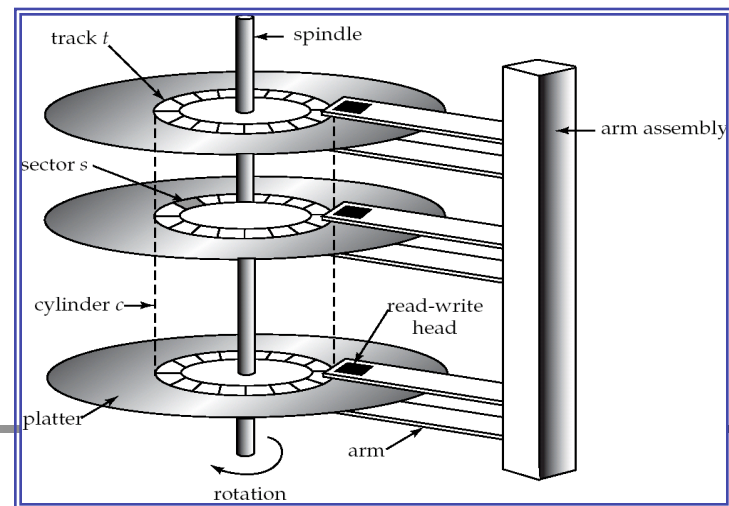


# Main Memory and Magnetic Disks

## Physical Look



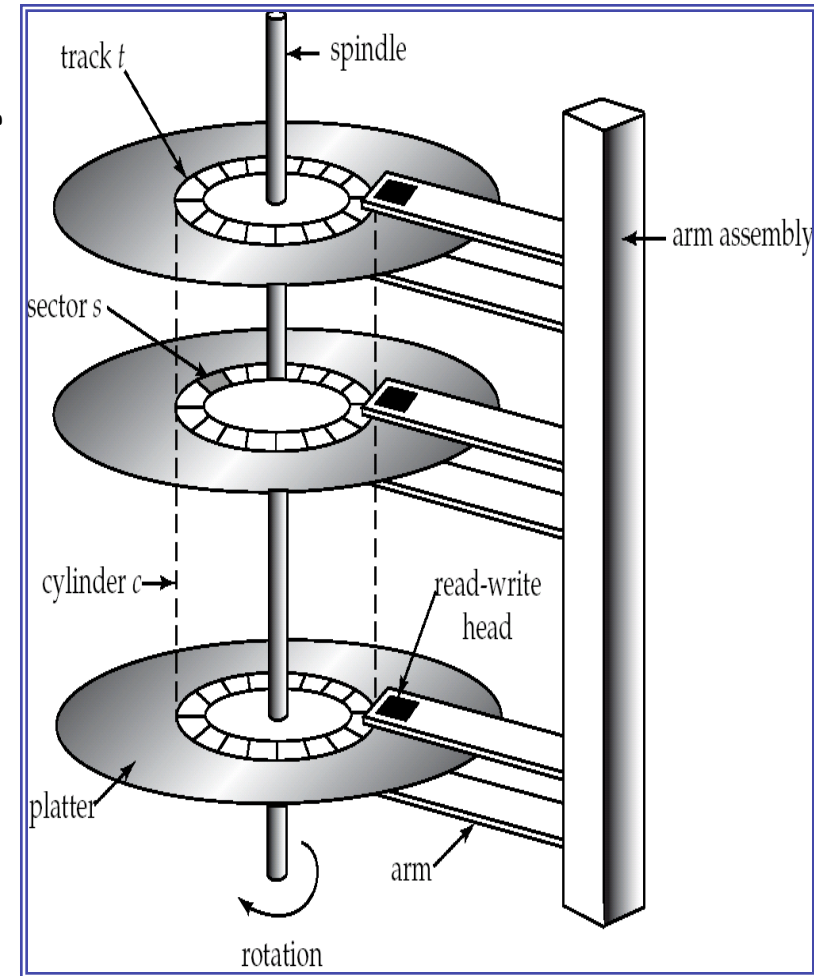
## Logical View



# Magnetic Disks






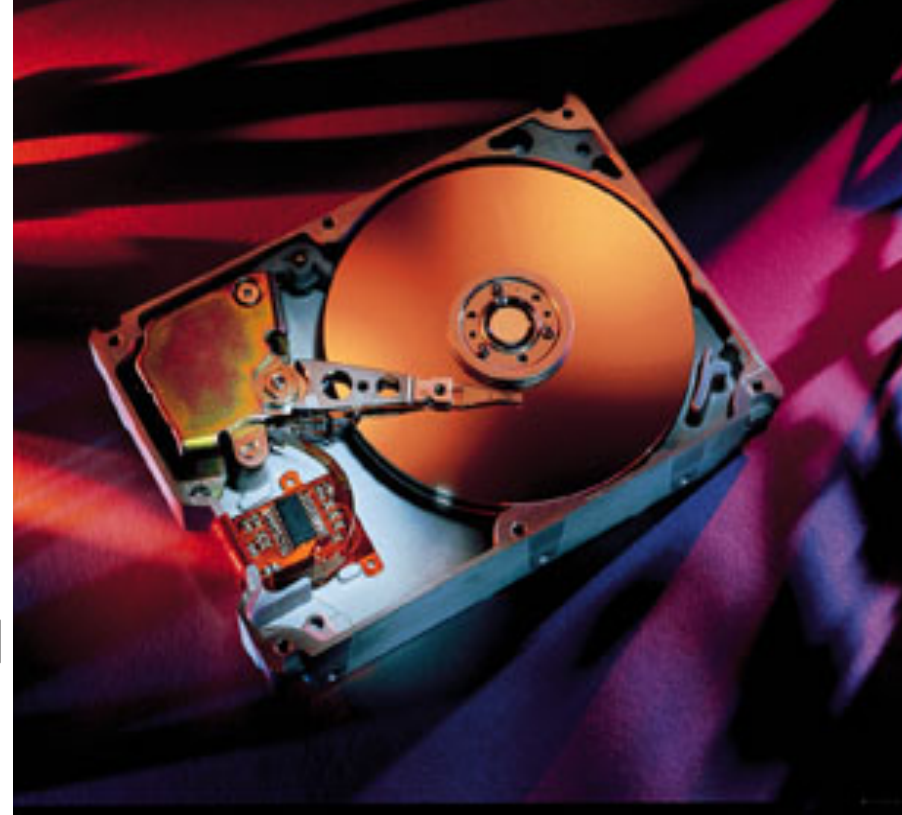
- Read-write head
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
- Each track is divided into **sectors**.
- A sector is the **smallest unit** of data that can be read or written.
- **Cylinder**  $i$  consists of  $i$ th track of all the platters





# Disk Drive Read/Write Operation

- Disk arm swings to  position head on the right track;
- Platter spins  continually to find the right sector;
- Data is read/written  as sector passes under head.
- Read/Write Need time!




Swing for track and spin for sector



# Access Time





- Access time is the time it takes from when a read or write a block request is issued to when data transfer begins (**or ends**).
- Access time = **Seek time** + **Rotational latency** + (**Transfer time**) 
- Seek time - time it takes to position the arm over the **right track**.
  - Average seek time is about  $\frac{1}{2}$  of the worst case seek time (e.g., from the innermost to the outermost)
- Rotational latency - time it takes for the **sector** to be accessed to appear under the head.
  - Average latency is  $\frac{1}{2}$  of the worst case latency (e.g., nearly 360 degree rotation)
- Transfer time - time to actually read/write the data in the sector once the head is positioned, that is, the time for the disk to rotate over the sectors.
  - In most cases, transfer time is much less than the seek time and rotational latency.

# Example

- Given a disk with the following characteristics:
  - There are  $2^{14}=16,384$  tracks per surface
  - There are  $2^7=128$  sectors per track
  - There are  $2^{12}=4,096$  bytes per sector
  - The disk rotates at 7,200rpm (rounds per minute); i.e., it makes one rotation in 8.33 milliseconds
  - To move the head arm between cylinders or tracks, it takes one millisecond to start and stop, plus one additional millisecond for every 1,000 cylinders travelled.
- Questions:
  - what is the time to take one track movement?
  - what is the time to move the head from innermost track to outmost track?

# Disk 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

# Example cont'd

- Given a disk with the following characteristics:
  - There are  $2^{14}=16384$  tracks per surfaces
  - There are  $2^7=128$  sectors per track
  - There are  $2^{12}=4096$  byte per sector
  - The disk rotates at 7200rpm; i.e., it makes one rotation in 8.33 milliseconds
  - To move the head arm between cylinders (tracks) take one milliseconds to start and stop, plus one additional millisecond for every 1000 cylinders travelled.
- Questions:
  - Assume that there is no gap between sectors; and each block occupies 4 sectors. what is the minimum time to read a block ?
    - Best case: seek time = 0; rotational latency = 0
    - $8.33*(4/128) = 0.2603$  milliseconds
  - What is the maximum time and average time to read a block? (Exercise)

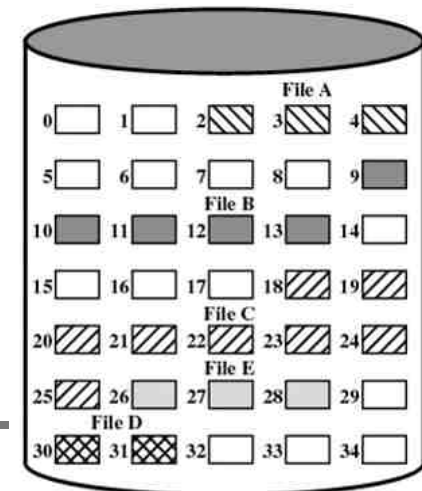
# Optimisation of Disk-Block Access



- Disk-arm-scheduling algorithms order pending accesses to tracks so that disk arm movement is minimised. 
- The elevator algorithm: 
  - move disk arm in one direction (from outer to inner tracks or vice versa),
  - process next request in that direction, till no more requests in that direction,
  - then reverse direction and repeat

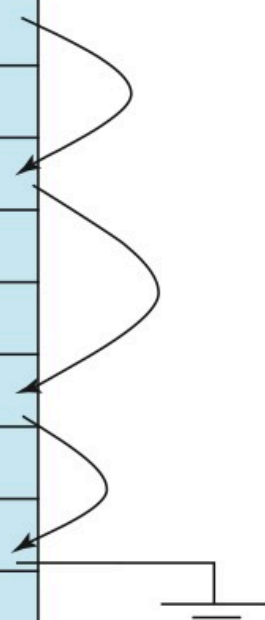
# File Organisation

- File organisation - optimise block access time by organising the blocks to correspond to how data will be accessed
  - e.g. store related information on the same or nearby cylinders.
- The database is stored as a collection of files.
  - Logically, each file is a sequence of records; a record is a sequence of fields.
  - Physically, each file is partitioned into blocks (pages).
- Assumption:
  - no record is larger than a block, i.e., each record is entirely contained in a single block.
- File Organisation
  - Fixed-length records
  - Variable-length records



# Fixed-length Records

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000





# Fixed-length Records cont'd

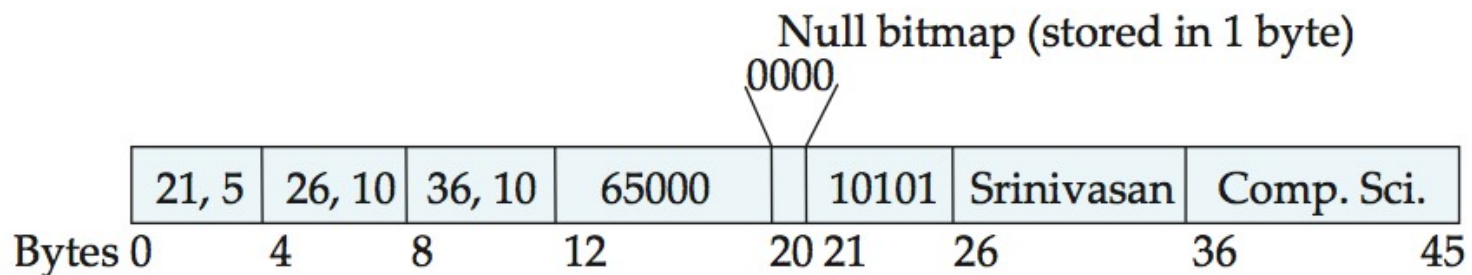
- Allocate only as many records to a block as would fit entirely in the block
- Undesirable to delete records (requires additional block accesses)
- Solutions:
  - allocate a certain number of bytes as a **file header**.
  - deleted records form a linked list, which is often referred to as a **free list**.
  - on insertion of a new record, we use the record pointed to by the header.
    - We change the header pointer to point to the next available record.
    - If no space is available, we add the new record to the end of the file.

# Variable-length Records

- Representation of a record with variable-length attributes has two parts:
  - an initial part with fixed length attributes, and
  - data for variable-length attributes
- Variable-length attributes are represented in the initial part of the record by a pair (offset, length)
  - offset denotes where the data for that attribute begins within the record
  - length is the length in bytes of the variable-sized attribute.

# Variable-length Records cont'd

- An instructor record:
  - The first three attributes ID, name, and department are variable-length strings
  - The fourth attribute salary is a fixed-sized number.
  - Note the use of Null Bitmap.

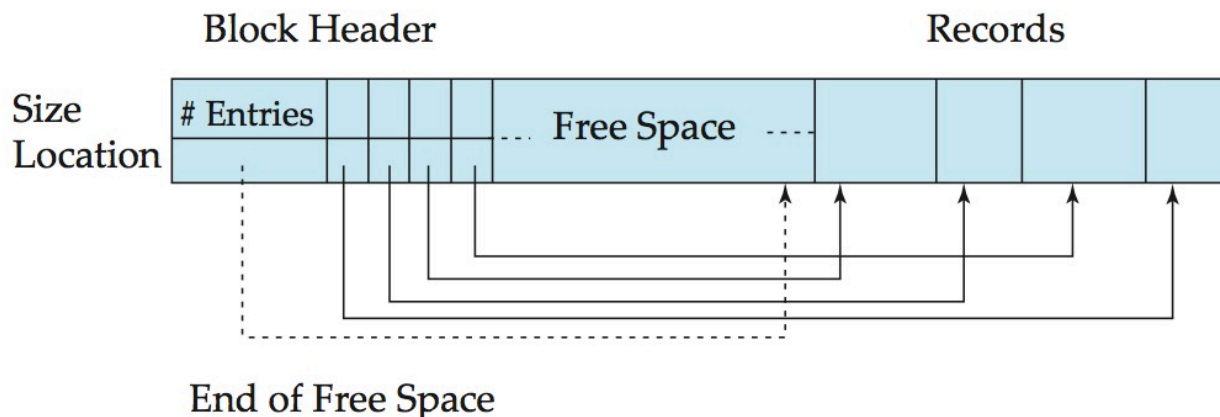


# Variable-length Records cont'd

- Storing variable-length records in a block uses the **slotted-page structure**:
  - The number of record entries in the header.
  - The end of free space in the block.
  - An array whose entries contain the location and size of each record.
- The actual records are allocated contiguously in the block, starting from the **end** of the block.
- The free space in the block is contiguous, between the final entry in the header array, and the first record.

# Slotted-page Structure

- If a record is inserted, space is allocated for it at the end of free space.
- If a record is deleted, the space that it occupies is freed. Further, the records in the block before the deleted record are moved, so that all free space is again between the final entry in the header array and the first record.
- Large objects are often represented using B+-tree file organisations (SQL supports Blob or Clob) (in future lectures)



# Organisation 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 organisation**, records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimise I/O

# Benefits VS Drawbacks

<i><b>File organisation</b></i>	<i><b>Benefits</b></i>	<i><b>Drawbacks</b></i>
Heap file	INSERTS (especially bulk inserts)	Slower in SELECTS and joining
Sequential File	in SELECTS and joining on the attribute that the table is sorted on.	in INSERTS
Hashing File	<ul style="list-style-type: none"><li>• SELECTS and joining (like sorted file)</li><li>• Faster performance with inserts than sorted file</li></ul>	Slower with inserts than heaped file



# Sequential File Organisation

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key
- 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 reorganise the file from time to time to restore sequential order

Account			
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	

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	
A-888	North Town	800	

# Multitable Clustering File Organisation

- Store several relations in one file using a multitable clustering file organisation (see next slide)
  - good for queries involving join (e.g. depositor and customer)
  - bad for queries involving only customer
  - results in variable size records
  - can add pointer chains to link records of a particular relation

# Multitable Clustering File Organisation cont'd

Depositor

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

Customer

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

Multitable clustering organisation of *customer* and *depositor*

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

# End of Lecture

## ■ Summary

- Overview of Physical Storage Media
  - Magnetic Disks: sector/tract/cylinder, block/page
  - Main memory
- Storage Access
- File Organisation
  - Fixed length/variable length records
  - Organisation of Records in Files

## ■ Reading

- chapter 10, database system concepts, 6th ed.