Disk Storage Devices

- Preferred secondary storage device for high storage capacity and low cost.
- Data stored as magnetized areas on magnetic disk surfaces.
- A **disk pack** contains several magnetic disks connected to a rotating spindle.
- Disks are divided into concentric circular **tracks** on each disk **surface**.
  - Track capacities vary typically from 4 to 50 Kbytes or more

Disk Storage Devices (contd.)

- A track is divided into smaller **blocks** or **sectors**
  - because it usually contains a large amount of information
- The division of a track into **sectors** is hard-coded on the disk surface and cannot be changed.
  - One type of sector organization calls a portion of a track that subtends a fixed angle at the center as a sector.
- A track is divided into **blocks**.
  - The block size B is fixed for each system.
    - Typical block sizes range from B=512 bytes to B=4096 bytes.
  - Whole blocks are transferred between disk and main memory for processing.

Disk Storage Devices (contd.)

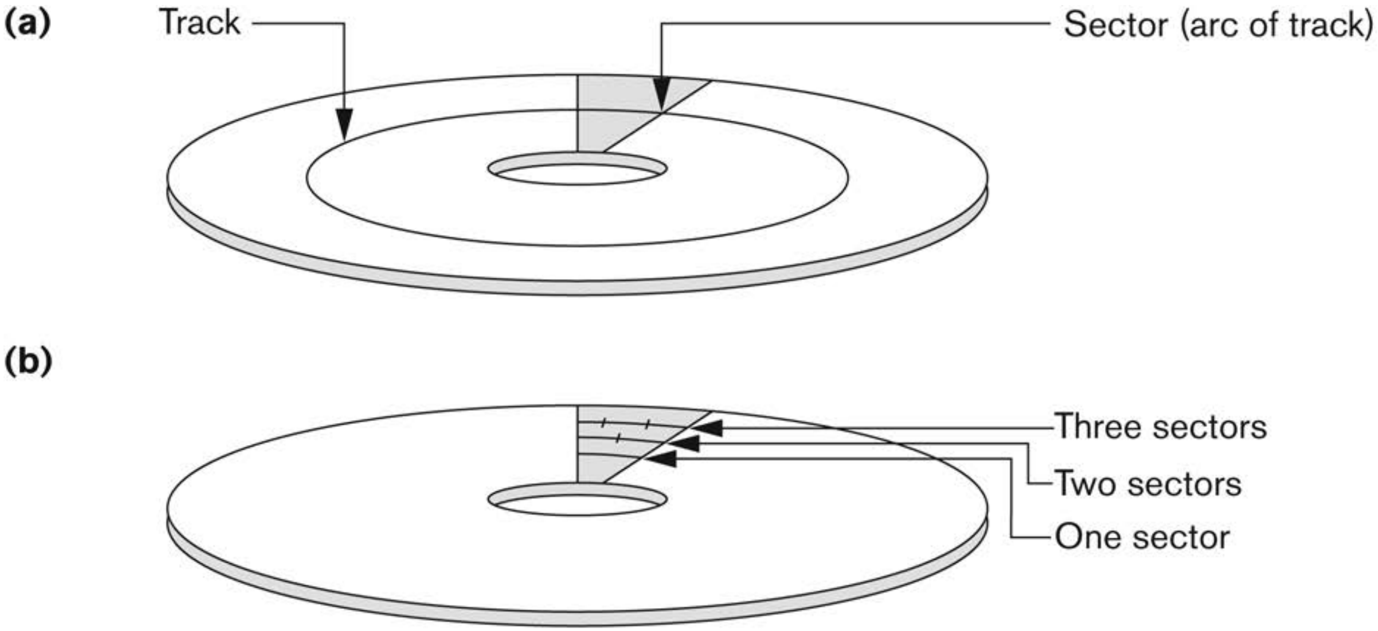


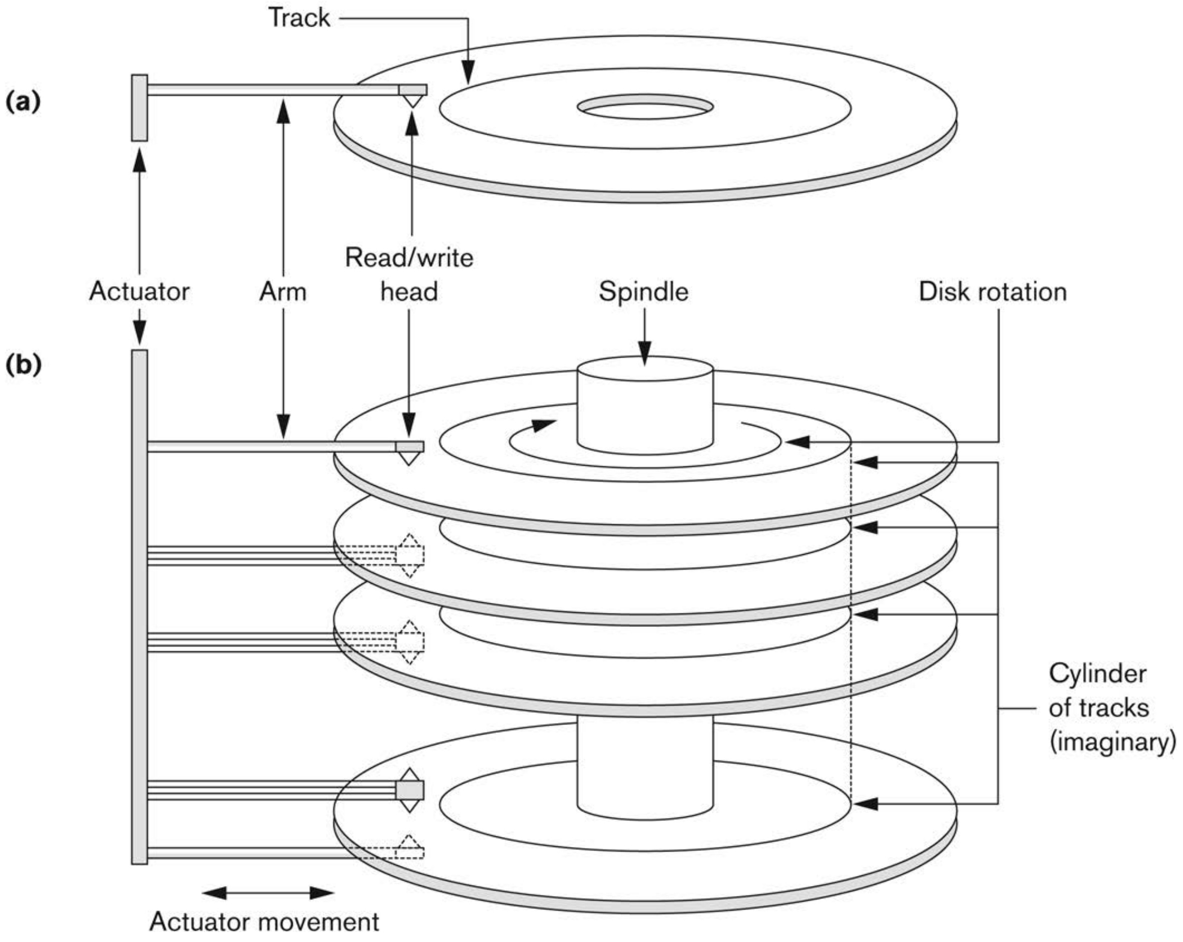
Figure 13.2
Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.

Disk Storage Devices (contd.)

- A **read-write head** moves to the track that contains the block to be transferred.
  - Disk rotation moves the block under the read-write head for reading or writing.
- A physical disk block (hardware) address consists of:
  - a cylinder number (imaginary collection of tracks of same radius from all recorded surfaces)
  - the track number or surface number (within the cylinder)
  - and block number (within track).
- Reading or writing a disk block is time consuming because of the seek time s and rotational delay (latency) **rd**.
- Double buffering can be used to speed up the transfer of contiguous disk blocks.

Disk Storage Devices (contd.)

Figure 13.1
(a) A single-sided disk with read/write hardware. (b) A disk pack with read/write hardware.



Records

- Fixed and variable length records
- Records contain fields which have values of a particular type
  - E.g., amount, date, time, age
- Fields themselves may be fixed length or variable length
- Variable length fields can be mixed into one record:
  - Separator characters or length fields are needed so that the record can be "parsed."

Blocking

- **Blocking**:
  - Refers to storing a number of records in one block on the disk.
- Blocking factor (**bfr**) refers to the number of records per block.
- There may be empty space in a block if an integral number of records do not fit in one block.
- **Spanned Records**:
  - Refers to records that exceed the size of one or more blocks and hence span a number of blocks.

Files of Records

- A **file** is a *sequence* of records, where each record is a collection of data values (or data items).
- A **file descriptor** (or **file header**) includes information that describes the file, such as the *field names* and their *data types*, and the addresses of the file blocks on disk.
- Records are stored on disk blocks.
- The **blocking factor bfr** for a file is the (average) number of file records stored in a disk block.
- A file can have **fixed-length** records or **variable-length** records.

Blocking Factor Calculation

- Name: 16 char 16B
- Age: int 4B
- Major: 4 char 4B
- GPA: float 4B
- Record size: 28B
- Block size: 4096B
- Bfr: floor (4096/28) = floor (146.28) = 146

Files of Records (contd.)

- File records can be **unspanned** or **spanned**
  - **Unspanned**: no record can span two blocks
  - **Spanned**: a record can be stored in more than one block
- The physical disk blocks that are allocated to hold the records of a file can be *contiguous, linked, or indexed*.
- In a file of fixed-length records, all records have the same format. Usually, unspanned blocking is used with such files.
- Files of variable-length records require additional information to be stored in each record, such as **separator characters** and **field types**.
  - Usually spanned blocking is used with such files.

Operation on Files

- Typical file operations include:
  - **OPEN**: Readies the file for access, and associates a pointer that will refer to a *current* file record at each point in time.
  - **FIND**: Searches for the first file record that satisfies a certain condition, and makes it the current file record.
  - **FINDNEXT**: Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
  - **READ**: Reads the current file record into a program variable.
  - **INSERT**: Inserts a new record into the file & makes it the current file record.
  - **DELETE**: Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
  - **MODIFY**: Changes the values of some fields of the current file record.
  - **CLOSE**: Terminates access to the file.
  - **REORGANIZE**: Reorganizes the file records.
    - For example, the records marked deleted are physically removed from the file or a new organization of the file records is created.
  - **READ_ORDERED**: Read the file blocks in order of a specific field of the file.

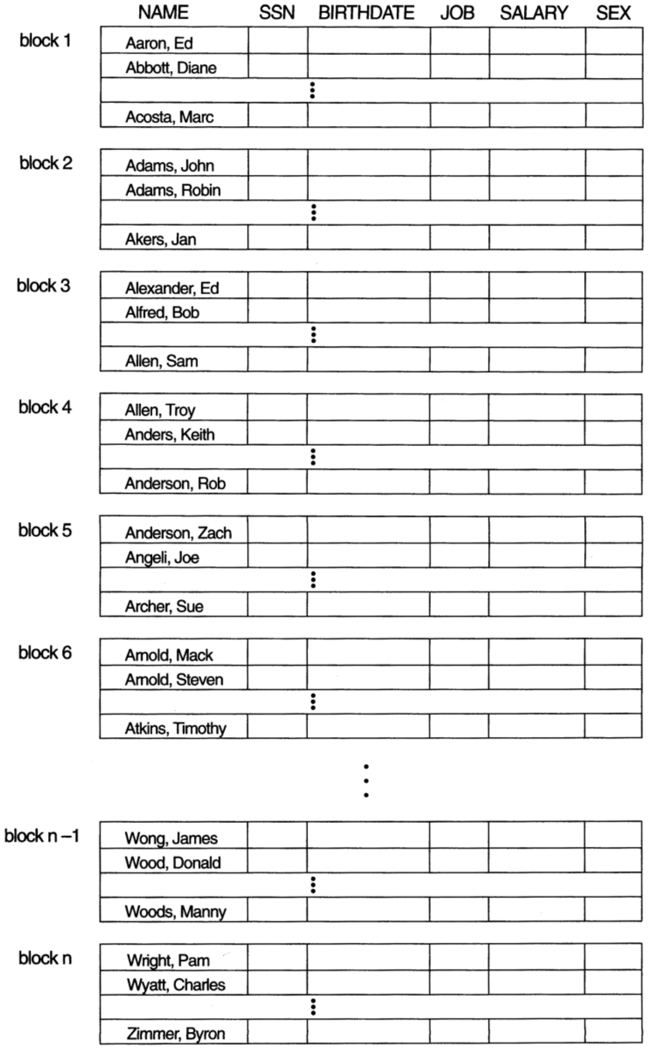
Unordered Files

- Also called a **heap** or a **pile** file.
- New records are inserted at the end of the file.
- A **linear search** through the file records is necessary to search for a record.
  - This requires reading and searching half the file blocks on the average, and is hence quite expensive.
- Record insertion is quite efficient.
- Reading the records in order of a particular field requires sorting the file records.

Ordered Files

- Also called a **sequential** file.
- File records are kept sorted by the values of an *ordering field*.
- Insertion is expensive: records must be inserted in the correct order.
  - It is common to keep a separate unordered *overflow* (or *transaction*) file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.
- A **binary search** can be used to search for a record on its *ordering field* value.
  - This requires reading and searching $\log_2$ of the file blocks on the average, an improvement over linear search.
- Reading the records in order of the ordering field is quite efficient.

Ordered Files (contd.)



	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
			⋮			
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
			⋮			
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
			⋮			
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
			⋮			
	Anderson, Rob					
block 5	Anderson, Zach					
	Angeli, Joe					
			⋮			
	Archer, Sue					
block 6	Arnold, Mack					
	Arnold, Steven					
			⋮			
	Atkins, Timothy					
			⋮			
block n –1	Wong, James					
	Wood, Donald					
			⋮			
	Woods, Manny					
block n	Wright, Pam					
	Wyatt, Charles					
			⋮			
	Zimmer, Byron					

Average Access Times

- The following table shows the average access time to access a specific record for a given type of file

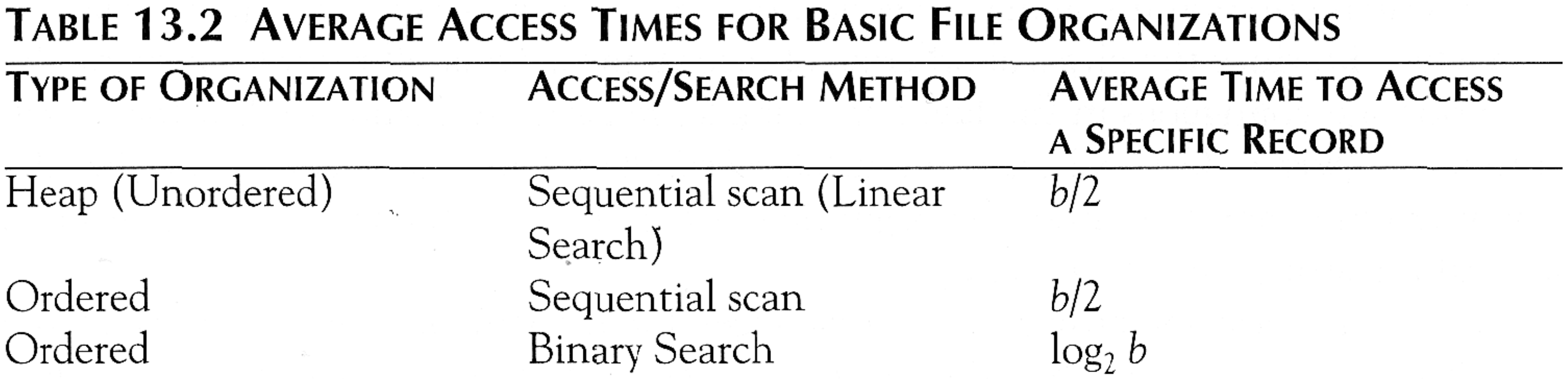


TABLE 13.2 AVERAGE ACCESS TIMES FOR BASIC FILE ORGANIZATIONS

TYPE OF ORGANIZATION	ACCESS/SEARCH METHOD	AVERAGE TIME TO ACCESS A SPECIFIC RECORD
Heap (Unordered)	Sequential scan (Linear Search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary Search	$\log_2 b$

Example

- Block: 4096B; Rec_Size: 28B;
- Bfr: floor (4096/28) = 146 records/block
- If 100,000 records
  - Numblocks = ceiling (100,000/146) = 685 blocks
  - Linear search = ceiling(685/2) = 343 block reads
    Binary Search = ceiling ($\log_2 685$) = 10 block reads
- If 10,000,000 records
  - Numblocks = ceiling (10,000,000/146) = 68,494
  - Linear search = ceiling(68,494/2) = 34,247 block reads
  - Binary Search = ceiling ($\log_2 685$) = 17 block reads