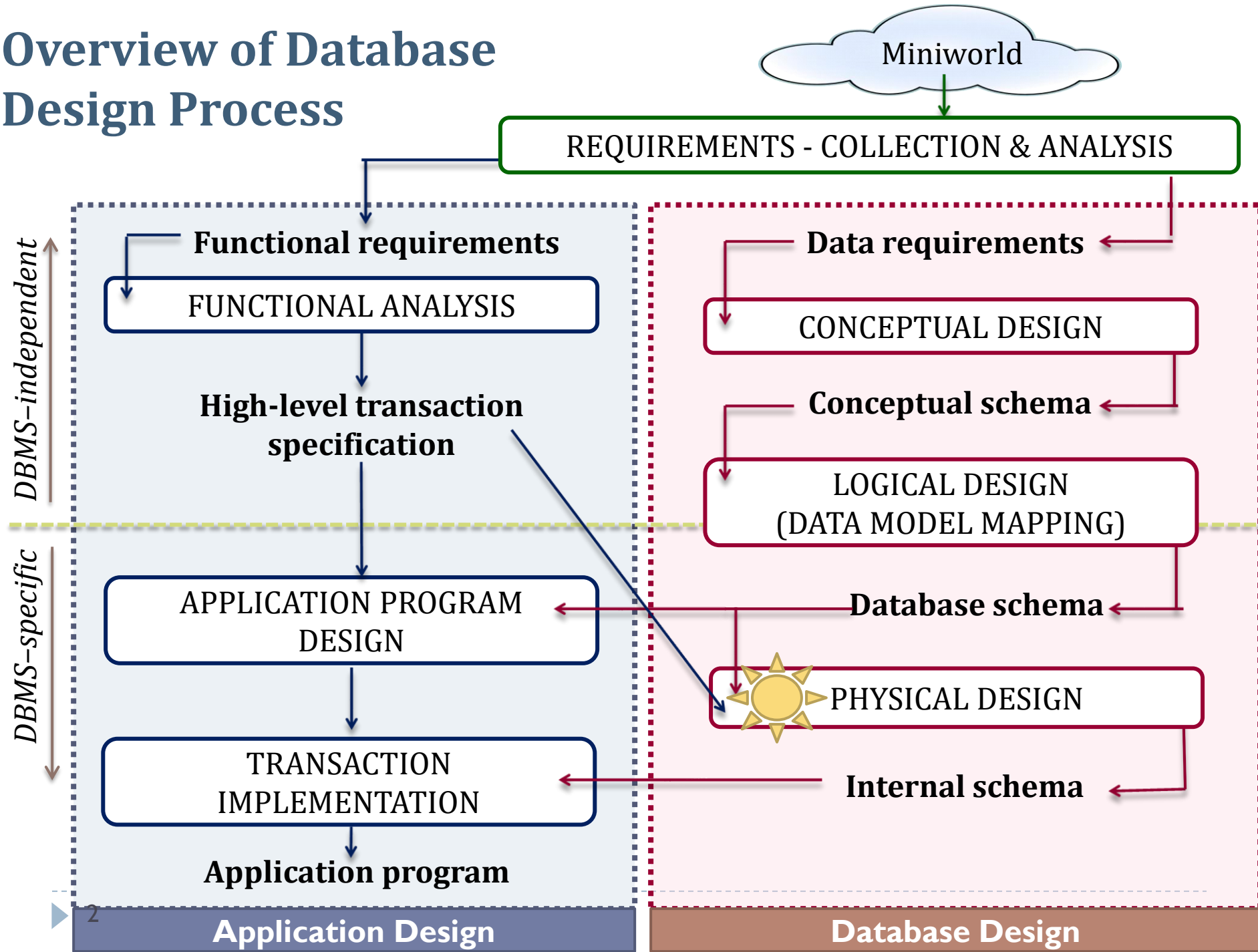




# **Data Storage, Indexing Structures for Files**

Chapter 8

# Overview of Database Design Process



# Contents

## 1 Data Storage

### 1.1 Disk Storage Devices

### 1.2 Files of Records

### 1.3 Operations on Files

### 1.4 Unordered Files & Ordered Files & Hashed Files

### 1.5 RAID Technology and Storage Area Networks

## 2 Indexing Structures for Files

### 2.1 Types of Single-level Ordered Indexes

### 2.2 Multilevel Indexes

### 2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# Contents

## **1 Data Storage**

### **1.1 Disk Storage Devices**

### 1.2 Files of Records

### 1.3 Operations on Files

### 1.4 Unordered Files & Ordered Files & Hashed Files

### 1.5 RAID Technology and Storage Area Networks

## **2 Indexing Structures for Files**

### 2.1 Types of Single-level Ordered Indexes

### 2.2 Multilevel Indexes

### 2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

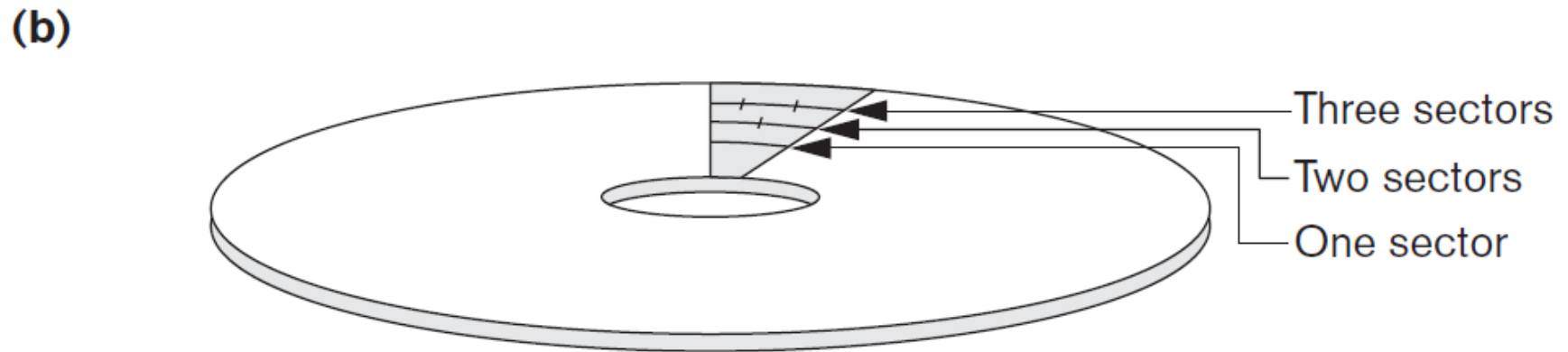
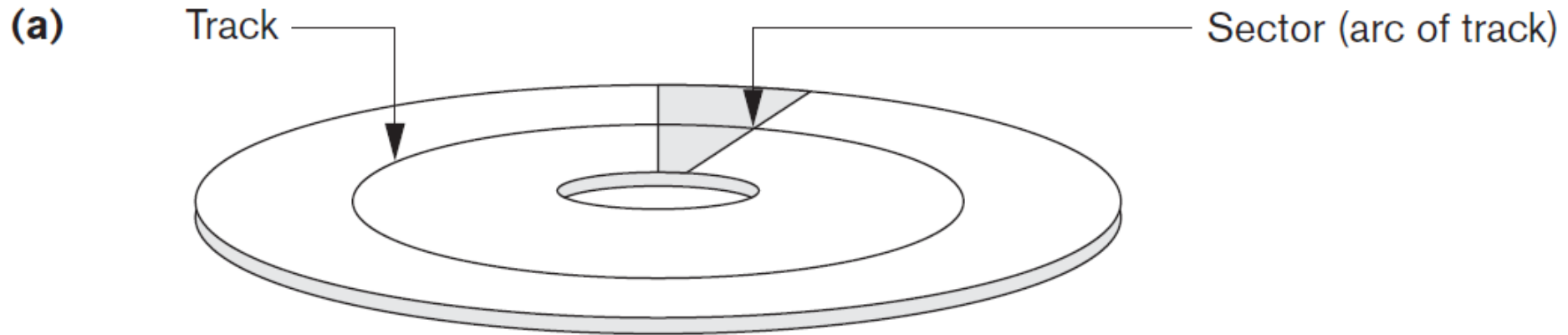
# Disk Storage Devices

---

- ▶ 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.

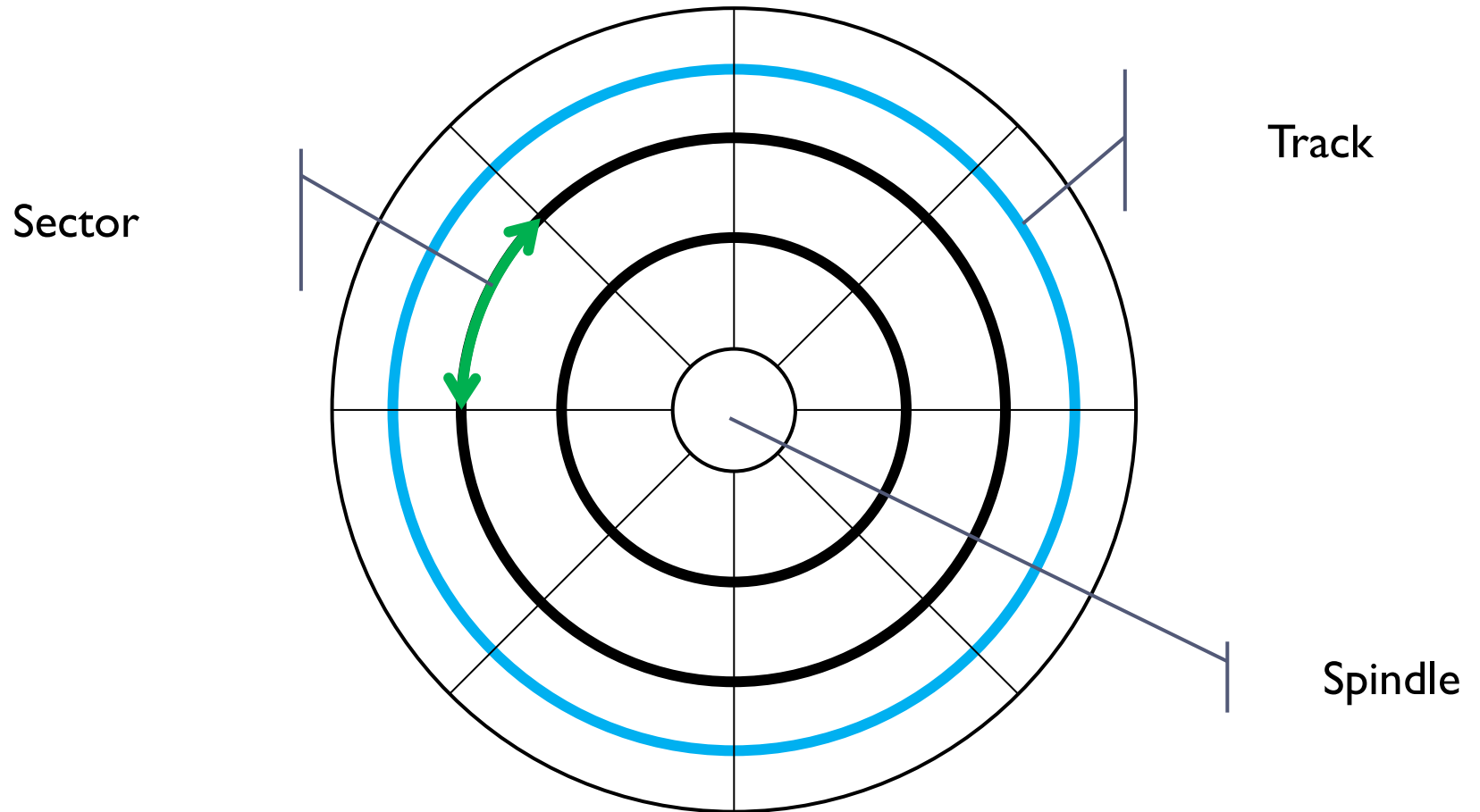
# Disk Storage Devices (cont.)

---



# Disk Storage Devices (cont.)

---



# Disk Storage Devices (cont.)

---

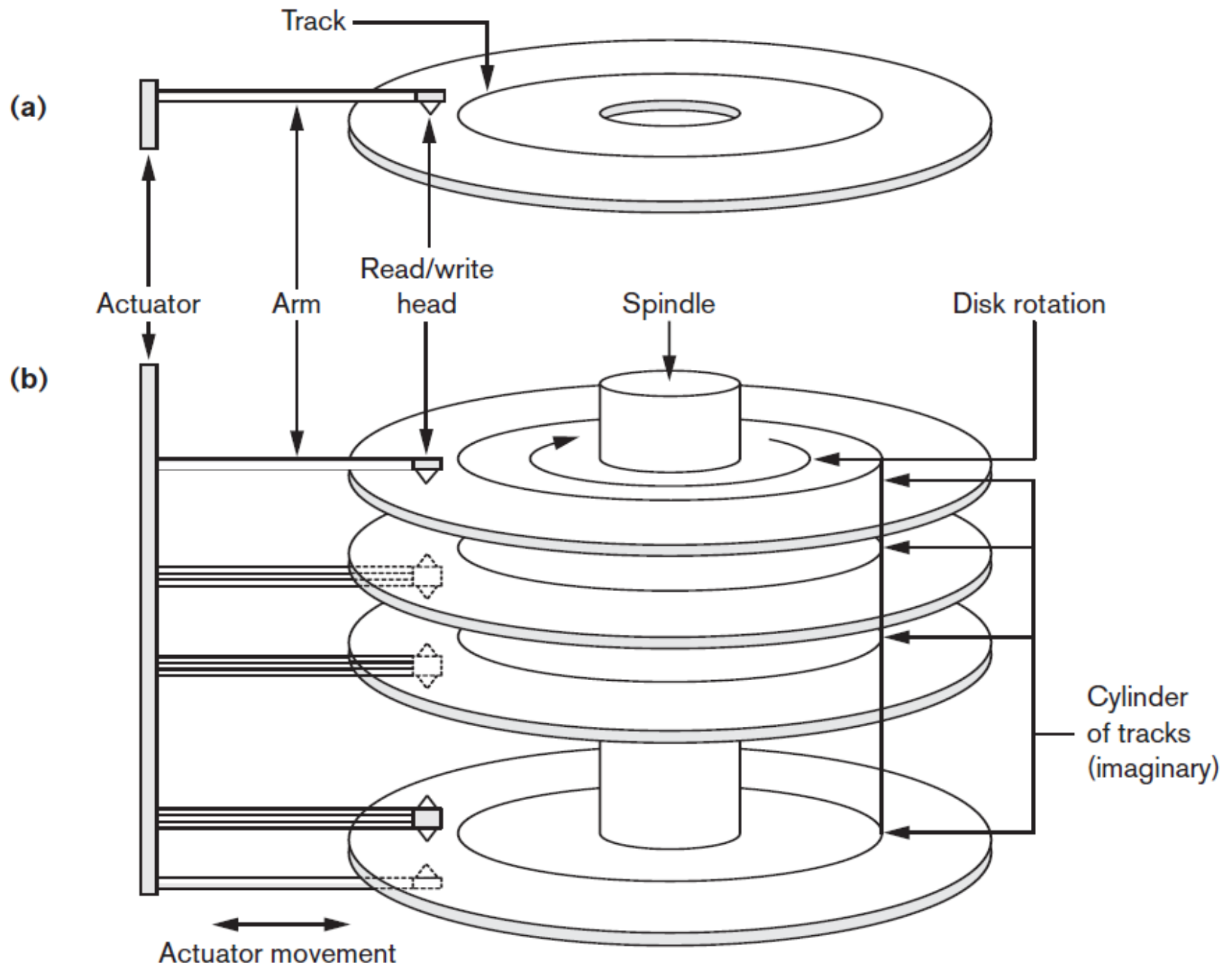
- ▶ A track is divided into smaller **blocks** or **sectors**.
  - ▶ because a track usually contains a large amount of information .
- ▶ A track is divided into **blocks**.
  - ▶ The block size  $B$  is fixed for each system.
    - ▶ Typical block sizes range from  $B=512$  bytes to  $B=4096$  bytes.
  - ▶ Whole blocks are transferred between disk and main memory for processing.



# Disk Storage Devices (cont.)

---

- ▶ 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.



# Contents

## **1 Data Storage**

1.1 Disk Storage Devices

### **1.2 Files of Records**

1.3 Operations on Files

1.4 Unordered Files & Ordered Files & Hashed Files

1.5 RAID Technology and Storage Area Networks

## **2 Indexing Structures for Files**

2.1 Types of Single-level Ordered Indexes

2.2 Multilevel Indexes

2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

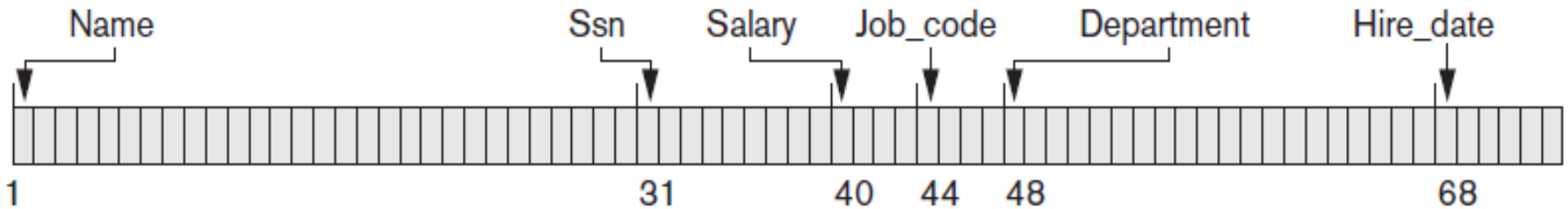
# Records

---

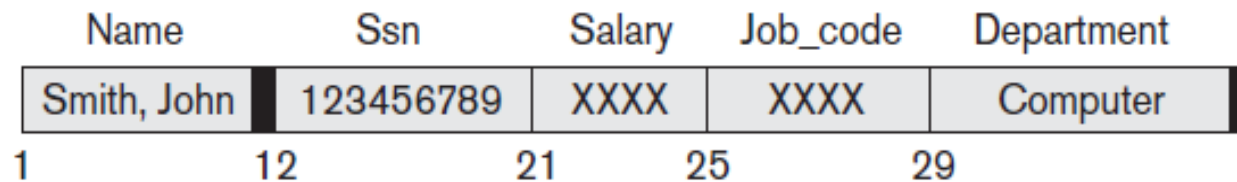
- ▶ Fixed and variable length records.
- ▶ Records contain fields which have values of a particular type.
  - ▶ E.g., amount, date, time, age.
- ▶ Fields themselves may be fixed length or variable length.
- ▶ Variable length fields can be mixed into one record:
  - ▶ Separator characters or length fields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

# Records (cont.)

(a)



(b)



(c)



Separator Characters	
=	Separates field name from field value
█	Separates fields
⌞	Terminates record

(a) A fixed-length record with 6 fields and size of 71 bytes.

(b) A record with 2 variable-length fields and 3 fixed-length fields.

(c) A variable-field record with 3 types of separator characters.

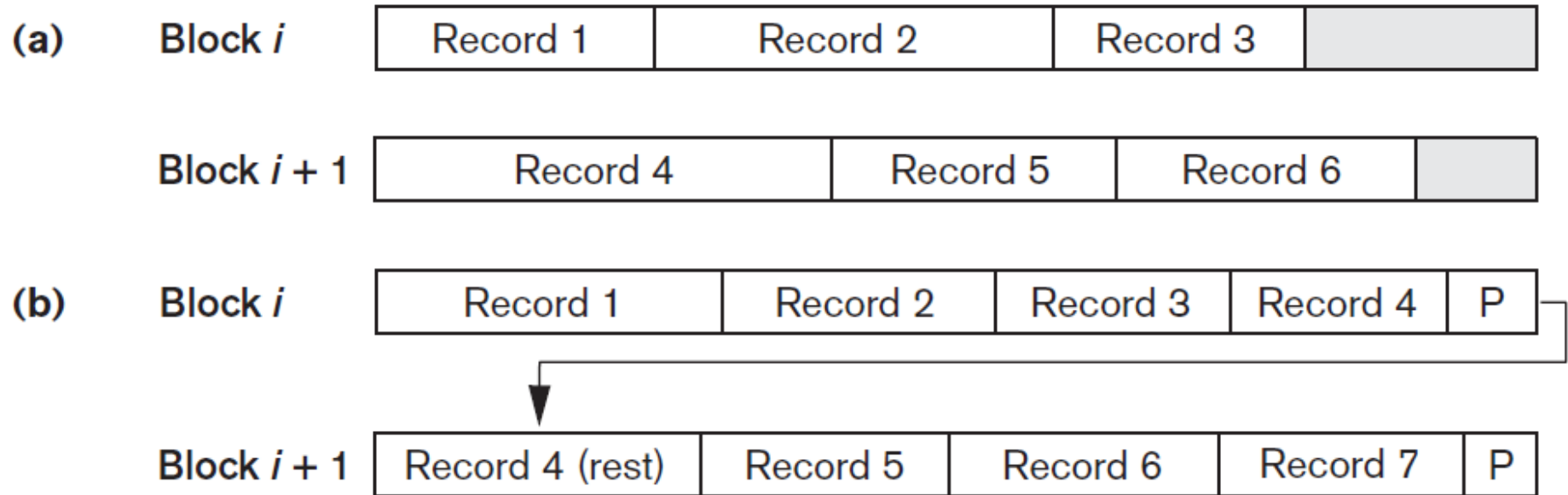
# Blocking

---

- ▶ **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:** refer to records that exceed the size of one or more blocks and hence span a number of blocks.

# Blocking (cont.)

---



(a) Unspanned records

(b) Spanned records

# 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.



## Files of Records (cont.)

---

- ▶ 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.

# Contents

## **1 Data Storage**

1.1 Disk Storage Devices

1.2 Files of Records

### **1.3 Operations on Files**

1.4 Unordered Files & Ordered Files & Hashed Files

1.5 RAID Technology and Storage Area Networks

## **2 Indexing Structures for Files**

2.1 Types of Single-level Ordered Indexes

2.2 Multilevel Indexes

2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees



# Operation on Files

---

- ▶ **OPEN:** Reads 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, and makes it the current file record.

## Operation on Files (cont.)

---

- ▶ **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.

# Contents

## **1 Data Storage**

1.1 Disk Storage Devices

1.2 Files of Records

1.3 Operations on Files

## **1.4 Unordered Files & Ordered Files & Hashed Files**

1.5 RAID Technology and Storage Area Networks

## **2 Indexing Structures for Files**

2.1 Types of Single-level Ordered Indexes

2.2 Multilevel Indexes

2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# 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 (cont.)

	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 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 17.2** Average Access Times for a File of  $b$  Blocks under Basic File Organizations

---

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

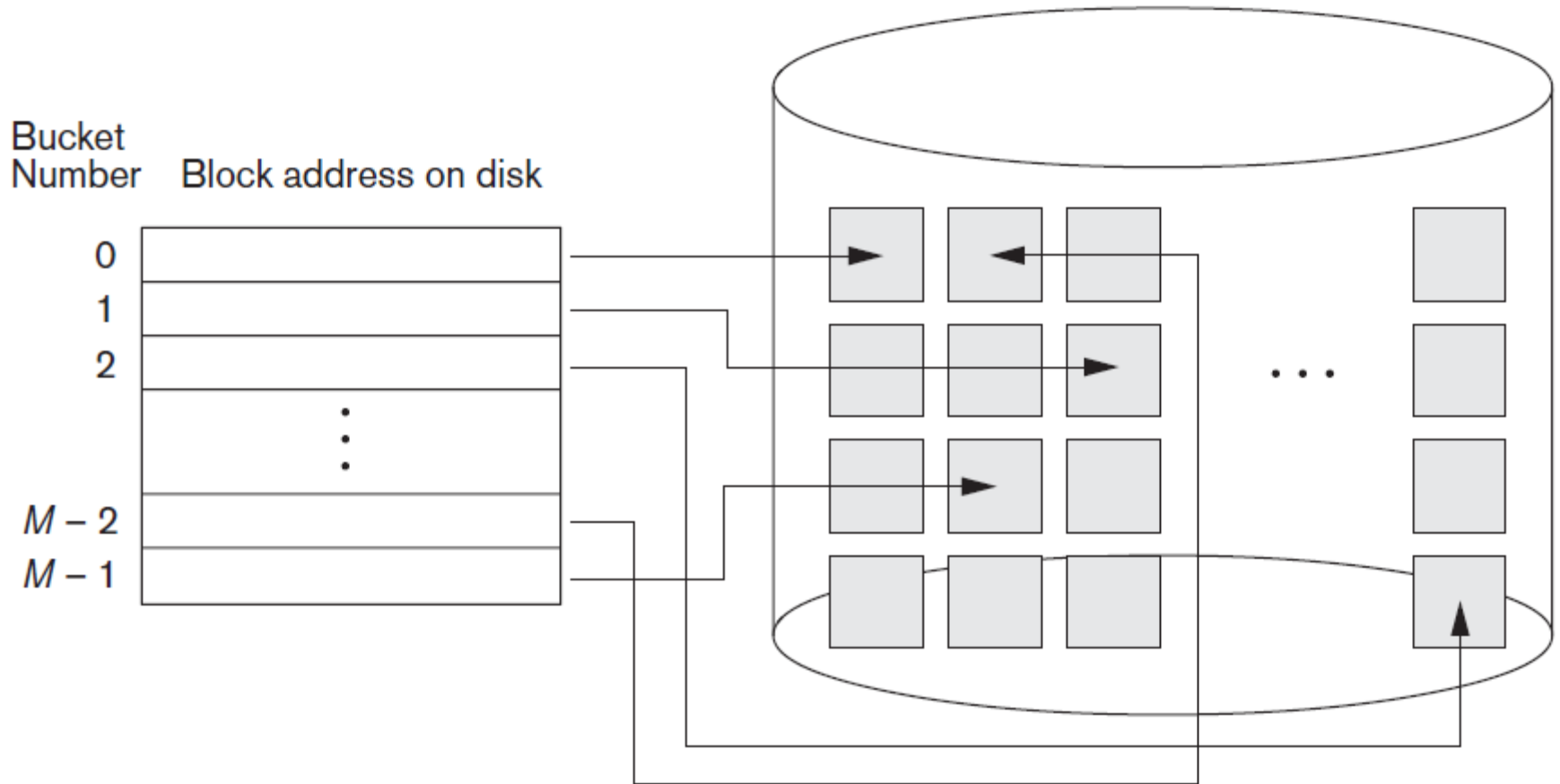
---

# Hashed Files

---

- ▶ Hashing for disk files is called **External Hashing**.
- ▶ The file blocks are divided into  $M$  equal-sized **buckets**, numbered  $\text{bucket}_0, \text{bucket}_1, \dots, \text{bucket}_{M-1}$ .
  - ▶ Typically, a bucket corresponds to one (or a fixed number of) disk block.
- ▶ One of the file fields is designated to be the **hash key** of the file.
- ▶ The record with hash key value  $K$  is stored in bucket  $i$ , where  $i=h(K)$ , and  $h$  is the **hashing function**.
- ▶ Search is very efficient on the hash key.
- ▶ Collisions occur when a new record hashes to a bucket that is already full.
  - ▶ An overflow file is kept for storing such records.
  - ▶ Overflow records that hash to each bucket can be linked together

# Hashed Files (cont.)



# Hashed Files (cont.)

- ▶ There are numerous methods for collision resolution, including the following:
  - ▶ **Open addressing:** Proceeding from the occupied position specified by the hash address, the program **checks the subsequent positions in order until an unused** (empty) position is found.

- ▶  $h(K) = K \bmod 7$

0	1	2	3	4	5	6
	1		3	11		6

- ▶ Insert 8

	1	8	3	11		6
--	---	---	---	----	--	---

- ▶ Insert 15

	1	8	3	11	15	6
--	---	---	---	----	----	---

- ▶ Insert 13

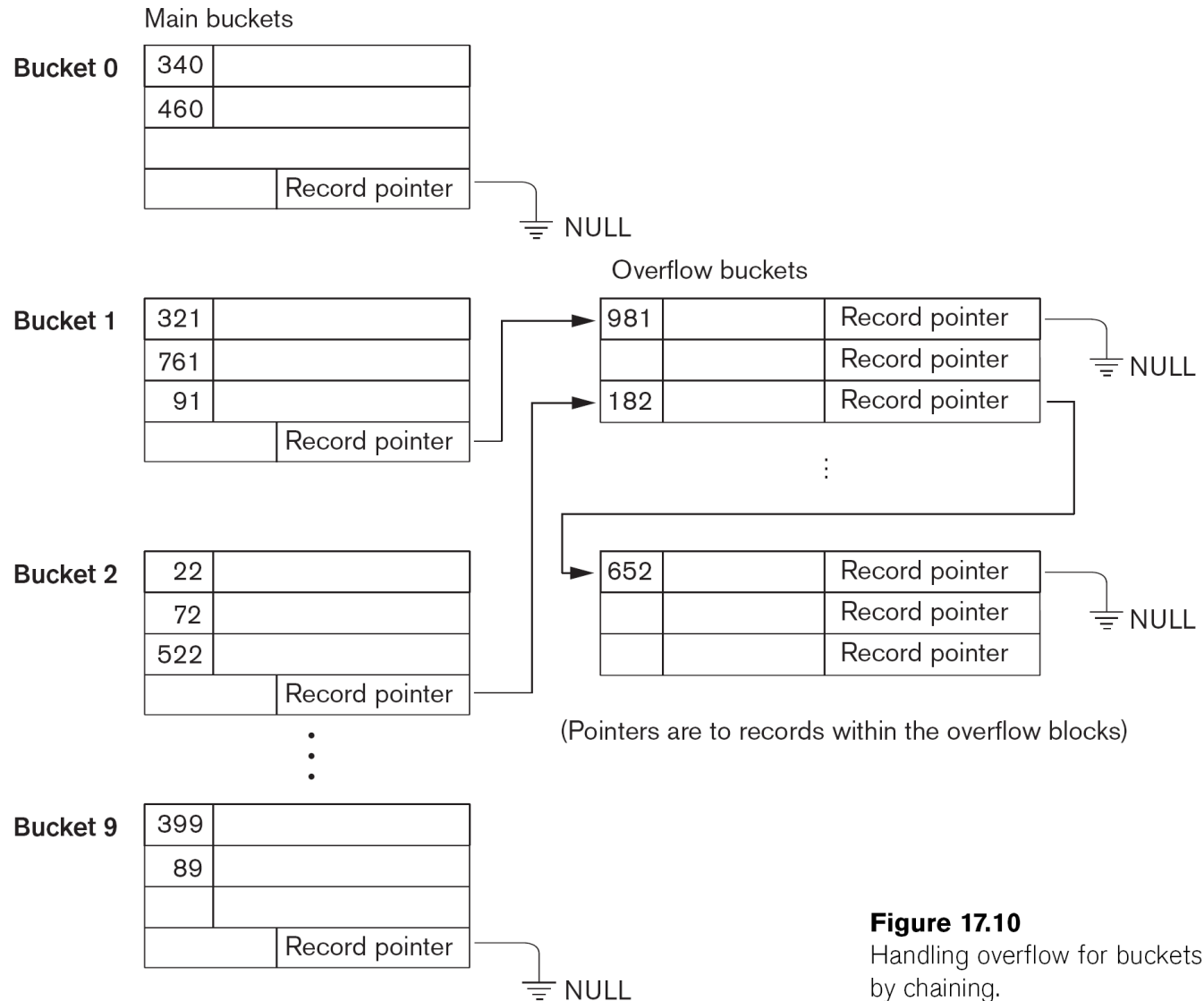
13	1	8	3	11	15	6
----	---	---	---	----	----	---

# Hashed Files (cont.)

---

- ▶ There are numerous methods for collision resolution, including the following:
  - ▶ **Chaining:**
    - ▶ Various overflow locations are kept: extending the array with a number of overflow positions.
    - ▶ A pointer field is added to each record location.
    - ▶ A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.
  - ▶ **Multiple hashing:**
    - ▶ The program applies a second hash function if the first results in a collision.
    - ▶ If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

# Hashed Files (cont.) - Overflow handling



**Figure 17.10**

Handling overflow for buckets by chaining.

# Contents

## **1 Data Storage**

1.1 Disk Storage Devices

1.2 Files of Records

1.3 Operations on Files

1.4 Unordered Files & Ordered Files & Hashed Files

## **1.5 RAID Technology and Storage Area Networks**

## **2 Indexing Structures for Files**

2.1 Types of Single-level Ordered Indexes

2.2 Multilevel Indexes

2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# Parallelizing Disk Access using RAID Technology

---

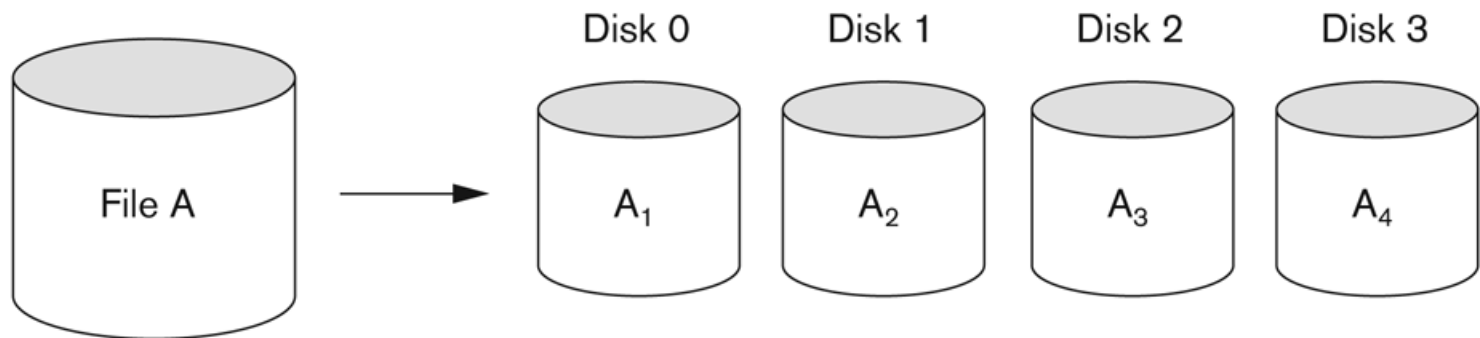
- ▶ Secondary storage technology must take steps to keep up in performance and reliability with processor technology.
- ▶ A major advance in secondary storage technology is represented by the development of **RAID**, which originally stood for **Redundant Arrays of Inexpensive Disks**.
- ▶ The main goal of RAID is to even out the widely different rates of performance improvement of disks against those in memory and microprocessors.



# RAID Technology (cont.)

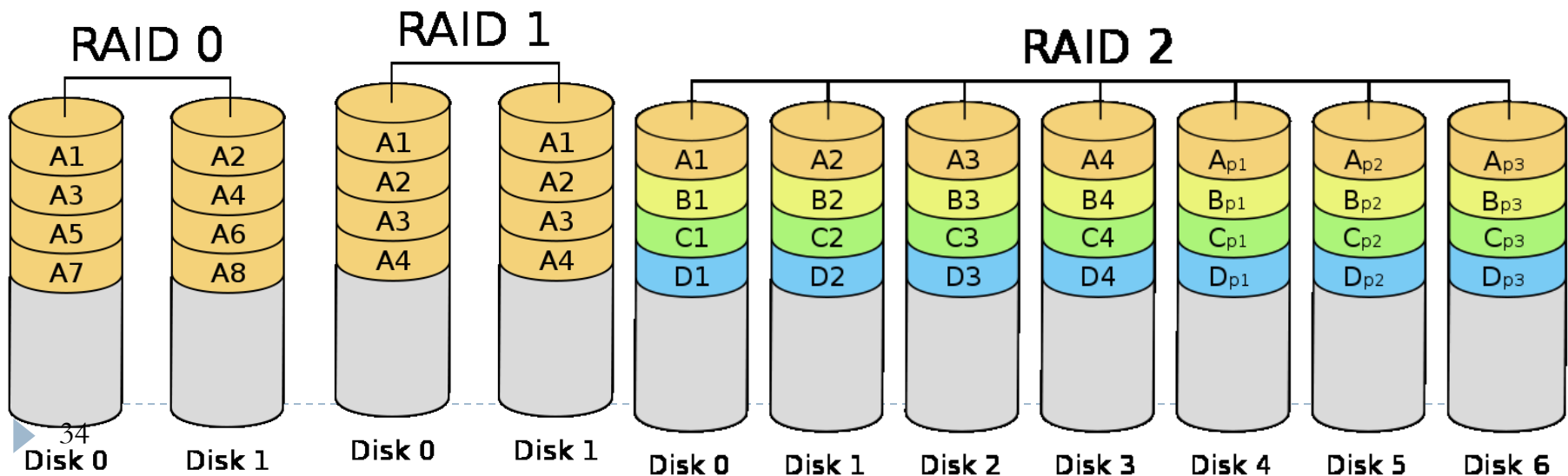
---

- ▶ A natural solution is a large array of small independent disks acting as a single higher-performance logical disk.
- ▶ A concept called **data striping** is used, which utilizes *parallelism* to improve disk performance.
- ▶ Data striping distributes data transparently over multiple disks to make them appear as a single large, fast disk.



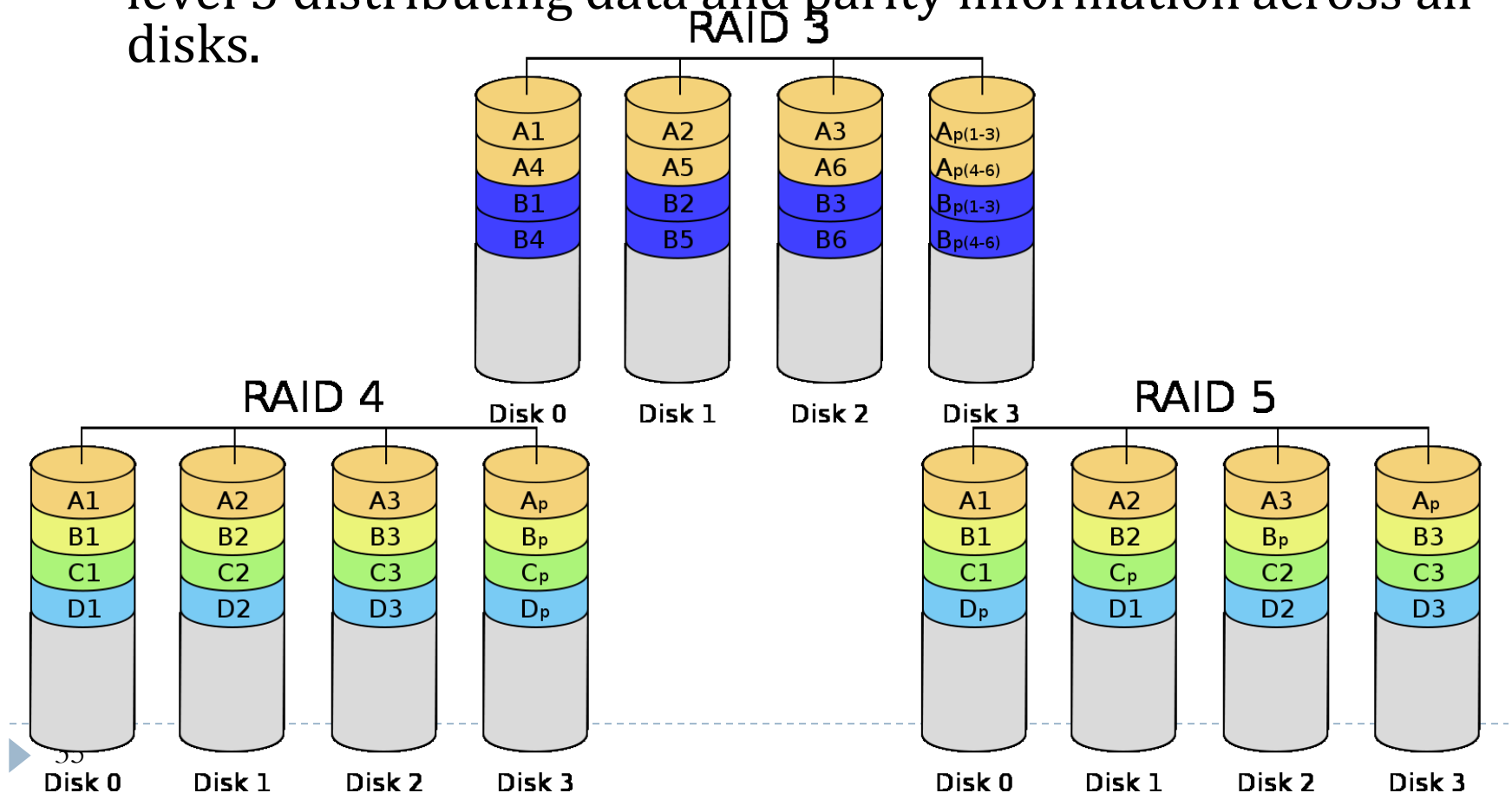
# RAID Technology (cont.)

- ▶ Different raid organizations were defined based on different combinations of the two factors of granularity of data interleaving (striping) and pattern used to compute redundant information.
  - ▶ **Raid level 0** has no redundant data and hence has the best write performance.
  - ▶ **Raid level 1** uses mirrored disks.
  - ▶ **Raid level 2** uses memory-style redundancy by using Hamming codes, which contain parity bits for distinct overlapping subsets of components. Level 2 includes both error detection and correction.



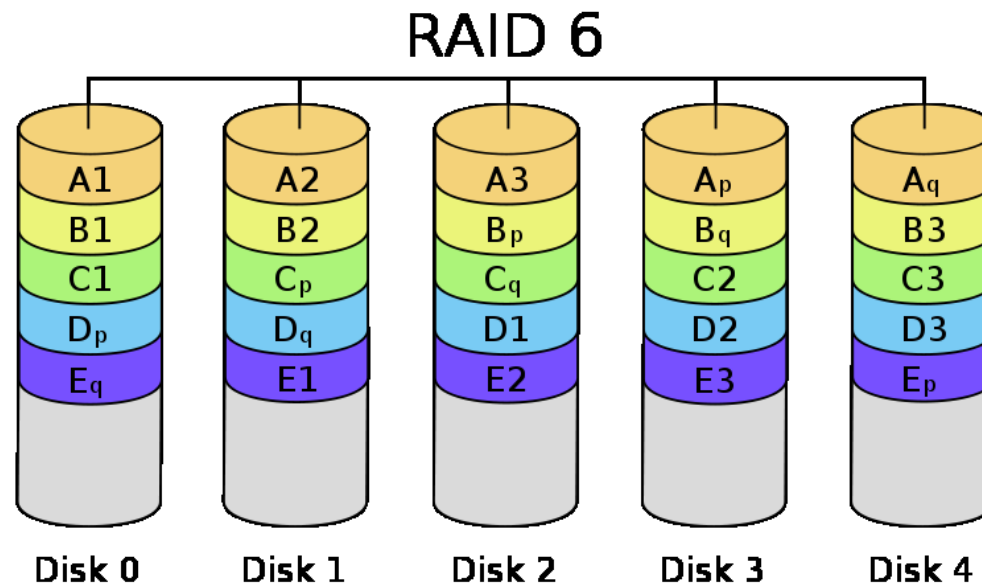
# RAID Technology (cont.)

- ▶ **Raid level 3** uses a single parity disk relying on the disk controller to figure out which disk has failed.
- ▶ **Raid levels 4 and 5** use block-level data striping, with level 5 distributing data and parity information across all disks.



# RAID Technology (cont.)

- ▶ **Raid level 6** applies the so-called  $P + Q$  redundancy scheme using Reed-Soloman codes to protect against up to two disk failures by using just two redundant disks.



# Use of RAID Technology (cont.)

---

- ▶ Different raid organizations are being used under different situations:
  - ▶ Raid level 1 (mirrored disks) is the easiest for rebuild of a disk from other disks
    - ▶ It is used for critical applications like logs.
  - ▶ Raid level 2 uses memory-style redundancy by using Hamming codes, which contain parity bits for distinct overlapping subsets of components. Level 2 includes both error detection and correction.
  - ▶ Raid level 3 (single parity disks relying on the disk controller to figure out which disk has failed) and level 5 (block-level data striping) are preferred for large volume storage, with level 3 giving higher transfer rates.
  - ▶ Most popular uses of the RAID technology currently are: Level 0 (with striping), Level 1 (with mirroring) and Level 5 with an extra drive for parity.
  - ▶ Design decisions for RAID include – level of RAID, number of disks, choice of parity schemes, and grouping of disks for block-level striping.

# Storage Area Networks

---

- ▶ The demand for higher storage has risen considerably in recent times.
- ▶ Organizations have a need to move from a static fixed data center oriented operation to a more flexible and dynamic infrastructure for information processing.
- ▶ Thus they are moving to a concept of Storage Area Networks (SANs).
  - ▶ In a SAN, online storage peripherals are configured as nodes on a high-speed network and can be attached and detached from servers in a very flexible manner.
- ▶ This allows storage systems to be placed at longer distances from the servers and provide different performance and connectivity options.

# Storage Area Networks (cont.)

---

## ▶ **Advantages of SANs are:**

- ▶ Flexible many-to-many connectivity among servers and storage devices using fiber channel hubs and switches.
- ▶ Up to 10km separation between a server and a storage system using appropriate fiber optic cables.
- ▶ Better isolation capabilities allowing nondisruptive addition of new peripherals and servers.
- ▶ SANs face the problem of combining storage options from multiple vendors and dealing with evolving standards of storage management software and hardware.

# Contents

## 1 Data Storage

### 1.1 Disk Storage Devices

### 1.2 Files of Records

### 1.3 Operations on Files

### 1.4 Unordered Files & Ordered Files & Hashed Files

### 1.5 RAID Technology and Storage Area Networks

## 2 Indexing Structures for Files

### 2.1 Types of Single-level Ordered Indexes

### 2.2 Multilevel Indexes

### 2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees





# Indexes as Access Paths

---

- ▶ A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- ▶ The index is usually specified on one field of the file (although it could be specified on several fields)
- ▶ One form of an index is a file of entries **<field value, pointer to record>**, which is ordered by field value
- ▶ The index is called an access path on the field.

## Indexes as Access Paths (cont.)

---

- ▶ The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller.
- ▶ A *binary search* on the index yields a pointer to the file record.
- ▶ Indexes can also be characterized as dense or sparse:
  - ▶ A **dense index** has an index entry for every search key value (and hence every record) in the data file.
  - ▶ A **sparse (or nondense) index**, on the other hand, has index entries for only some of the search values

# Example

Given the following data file: EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ... )

Suppose that:

- ▶ record size  $R=150$  bytes, block size  $B=512$  bytes,  $r=30000$  records
- ▶ SSN Field size  $V_{SSN}=9$  bytes, record pointer size  $P_R=7$  bytes

Then, we get:

- ▶ blocking factor:  $bfr = \lfloor B/R \rfloor = \lfloor 512/150 \rfloor = 3$  records/block
- ▶ number of blocks needed for the file:  $b = \lceil r/bfr \rceil = \lceil 30000/3 \rceil = 10000$  blocks

**For an dense index on the SSN field:**

- ▶ index entry size:  $R_i = (V_{SSN} + P_R) = (9 + 7) = 16$  bytes
- ▶ index blocking factor  $bfr_i = \lfloor B/R_i \rfloor = \lfloor 512/16 \rfloor = 32$  entries/block
- ▶ number of blocks for index file:  $b_i = \lceil r/bfr_i \rceil = (30000/32) = 938$  blocks
- ▶ search for and retrieve a record needs:  $\lceil \log_2 b_i \rceil + 1 = \lceil \log_2 938 \rceil + 1 = 11$  block accesses

- ▶ This is compared to an average linear search cost of:

$$(b/2) = 10000/2 = 5000 \text{ block accesses}$$

- ▶ If the file records are ordered, the binary search cost would be:

$$\lceil \log_2 b \rceil = \lceil \log_2 10000 \rceil = 14 \text{ block accesses}$$

# Contents

1	Data Storage
1.1	Disk Storage Devices
1.2	Files of Records
1.3	Operations on Files
1.4	Unordered Files & Ordered Files & Hashed Files
1.5	RAID Technology and Storage Area Networks
<b>2</b>	<b>Indexing Structures for Files</b>
<b>2.1</b>	<b>Types of Single-level Ordered Indexes</b>
2.2	Multilevel Indexes
2.3	Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# Types of Single-level Ordered Indexes

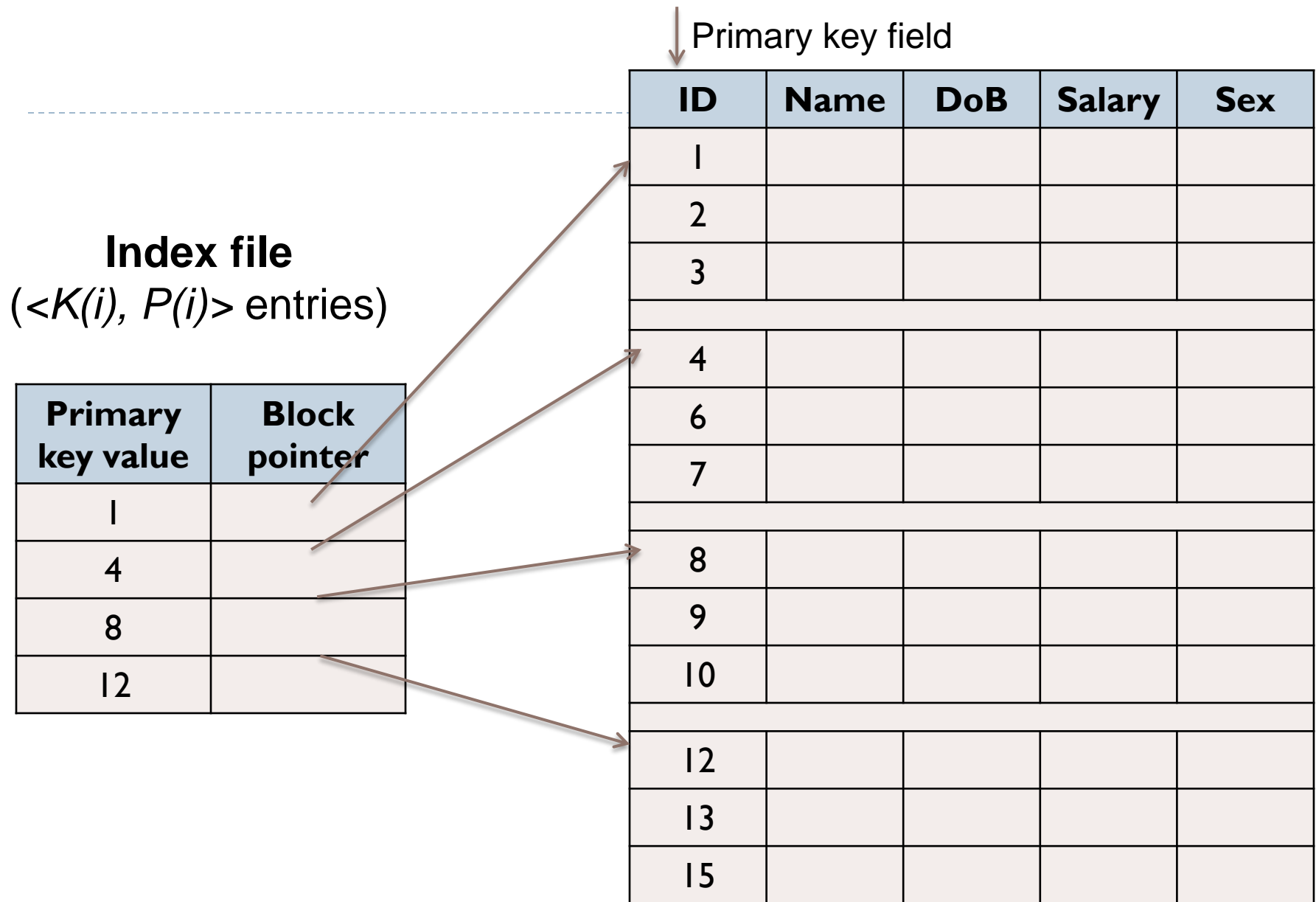
---

- ▶ Primary Indexes
- ▶ Clustering Indexes
- ▶ Secondary Indexes

# Primary Index

---

- ▶ Defined on an **ordered data file**.
  - ▶ The data file is ordered on a *key field*.
- ▶ One index entry *for each block* in the data file
  - ▶ *First record* in the block, which is called the *block anchor*
- ▶ A similar scheme can use the *last record* in a block.



# Primary Index

---

- ▶ Number of index entries?
  - ▶ Number of blocks in data file.
- ▶ Dense or Nondense?
  - ▶ Nondense
- ▶ Search/ Insert/ Update/ Delete?



# Clustering Index

---

- ▶ Defined on an **ordered data file**.
  - ▶ The data file is ordered on a *non-key field*.
- ▶ One index entry *each distinct value* of the field.
  - ▶ The index entry points to the *first data block* that contains records with that field value

↓ Clustering field

**Index file**  
( $\langle K(i), P(i) \rangle$  entries)

Clustering field value	Block pointer
1	
2	
3	
4	
5	

Dept_No	Name	DoB	Salary	Sex
1				
1				
2				
2				
2				
2				
2				
3				
3				
4				
4				
5				

Clustering field

**Index file**  
( $\langle K(i), P(i) \rangle$  entries)

Clustering field value	Block pointer
1	
2	
3	
4	
5	

Dept_No	Name	DoB	Salary	Sex
1				
1				
2				
2				
2				
2				
2				
3				
3				
4				
4				
5				

# Clustering Index

---

- ▶ Number of index entries?
  - ▶ Number of distinct indexing field values in data file.
- ▶ Dense or Nondense?
  - ▶ Nondense
- ▶ Search/ Insert/ Update/ Delete?
- ▶ At most **one primary index or one clustering index but not both.**

# Secondary index

---

- ▶ A secondary index provides a secondary means of accessing a file.
  - ▶ The data file is unordered on indexing field.
- ▶ Indexing field:
  - ▶ secondary key (unique value)
  - ▶ nonkey (duplicate values)
- ▶ The index is an ordered file with two fields.
  - ▶ The first field: *indexing field*.
  - ▶ The second field: *block* pointer or *record* pointer.
- ▶ There can be **many** secondary indexes for the same file.

# Index file ( $\langle K(i), P(i) \rangle$ entries)

Index field value	Block pointer
3	
4	
5	
6	
8	
9	
11	
13	
15	
18	
21	
23	

# Secondary key field

	5			
	13			
	8			
	6			
	15			
	3			
	9			
	21			
	11			
	4			
	23			
	18			

...

# Secondary index on key field



# Secondary index on key field

---

- ▶ Number of index entries?
  - ▶ Number of record in data file
- ▶ Dense or Nondense?
  - ▶ Dense
- ▶ Search/ Insert/ Update/ Delete?

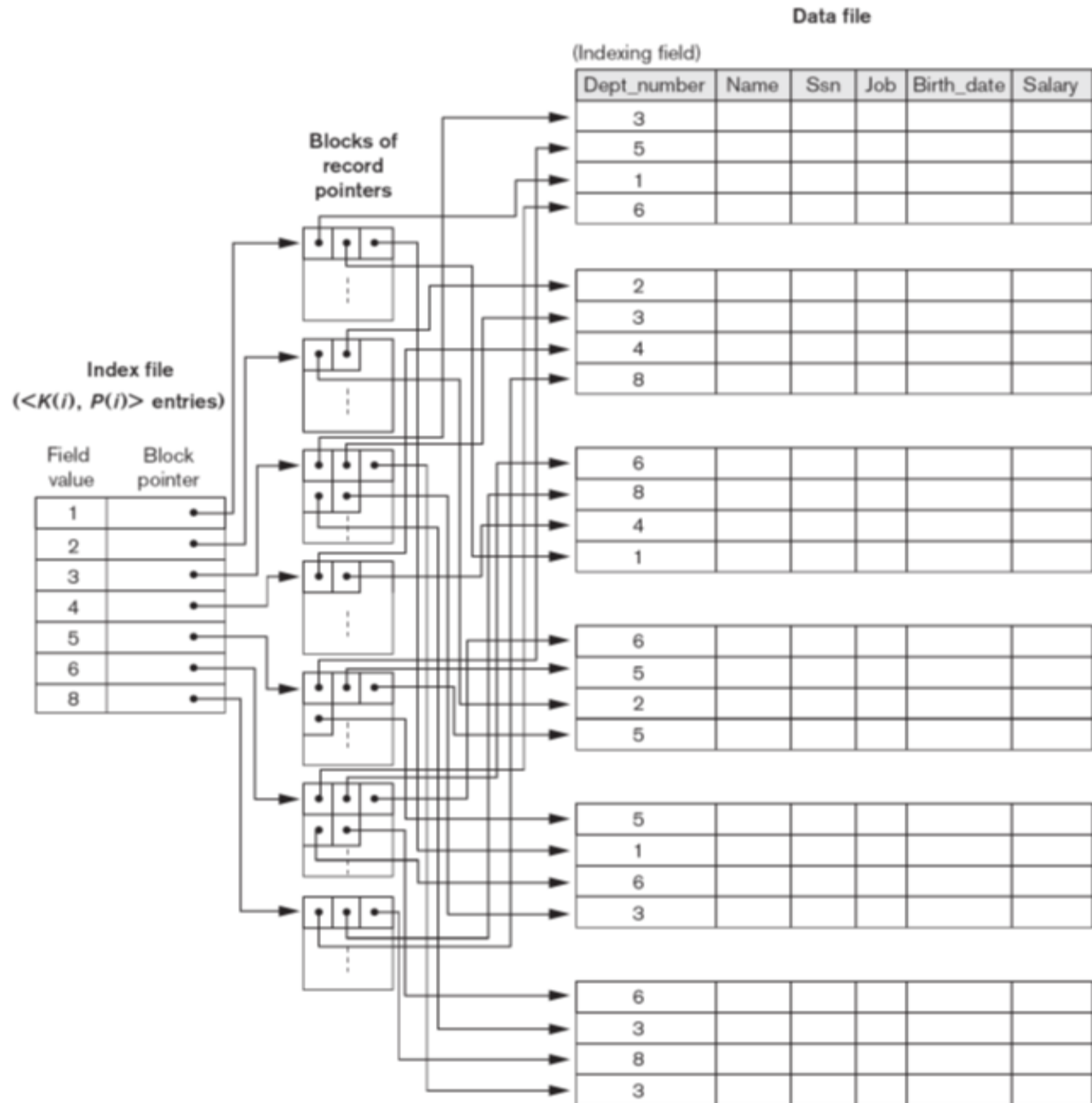
# Secondary index on non-key field

---

- ▶ **Discussion:** Structure of Secondary index on non-key field?
- ▶ Option 1: include **duplicate index entries** with the same  $K(i)$  value - one for each record.
- ▶ Option 2: keep a **list of pointers**  $\langle P(i, 1), \dots, P(i, k) \rangle$  in the index entry for  $K(i)$ .
- ▶ Option 3:
  - ▶ more commonly used.
  - ▶ one entry for each *distinct index field value* + an **extra level of indirection** to handle the multiple pointers.



► Secondary Index on non-key field:  
option 3



# Secondary index on nonkey field

---

- ▶ Number of index entries?
  - ▶ Number of records in data file
  - ▶ Number of distinct index field values
- ▶ Dense or Nondense?
  - ▶ Dense/ nondense
- ▶ Search/ Insert/ Update/ Delete?

# Summary of Single-level indexes

---

- ▶ Ordered file on indexing field?
  - ▶ Primary index
  - ▶ Clustering index
- ▶ Indexing field is Key?
  - ▶ Primary index
  - ▶ Secondary index
- ▶ Indexing field is not Key?
  - ▶ Clustering index
  - ▶ Secondary index

# Summary of Single-level indexes

---

- ▶ Dense index?
  - ▶ Secondary index
- ▶ Nondense index?
  - ▶ Primary index
  - ▶ Clustering index
  - ▶ Secondary index

# Summary of Single-level indexes

---

**Table 18.2** Properties of Index Types

Type of Index	Number of (First-level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no <sup>a</sup>
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records <sup>b</sup> or number of distinct index field values <sup>c</sup>	Dense or Nondense	No

<sup>a</sup>Yes if every distinct value of the ordering field starts a new block; no otherwise.

<sup>b</sup>For option 1.

<sup>c</sup>For options 2 and 3.

# Example

---

Given the following data file: EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ... )

Suppose that:

- ▶ record size  $R=150$  bytes, block size  $B=512$  bytes,  $r=30000$  records
- ▶ SSN Field size  $V_{SSN}=9$  bytes, block pointer size  $P=6$  bytes

Then, we get:

- ▶ blocking factor:  $bfr = \lfloor B/R \rfloor = \lfloor 512/150 \rfloor = 3$  records/block
- ▶ number of blocks needed for the file:  $b = \lceil r/bfr \rceil = \lceil 30000/3 \rceil = 10000$  blocks

**For a primary index on the ordering key field SSN:**

- ▶ index entry size:  $R_i = (V_{SSN} + P) = (9 + 6) = 15$  bytes
- ▶ index blocking factor  $bfr_i = \lfloor B/R_i \rfloor = \lfloor 512/15 \rfloor = 34$  entries/block
- ▶ number of blocks for index file:  $b_i = \lceil b/bfr_i \rceil = \lceil 10000/34 \rceil = 295$  blocks
- ▶ search for and retrieve a record needs:  $\lceil \log_2 b_i \rceil + 1 = \lceil \log_2 295 \rceil + 1 = 10$  block accesses
- ▶ This is compared to a dense index cost of: 11 block accesses

# Contents

## 1 Data Storage

### 1.1 Disk Storage Devices

### 1.2 Files of Records

### 1.3 Operations on Files

### 1.4 Unordered Files & Ordered Files & Hashed Files

### 1.5 RAID Technology and Storage Area Networks

## 2 Indexing Structures for Files

### 2.1 Types of Single-level Ordered Indexes

### 2.2 Multilevel Indexes

### 2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# Multi-Level Indexes

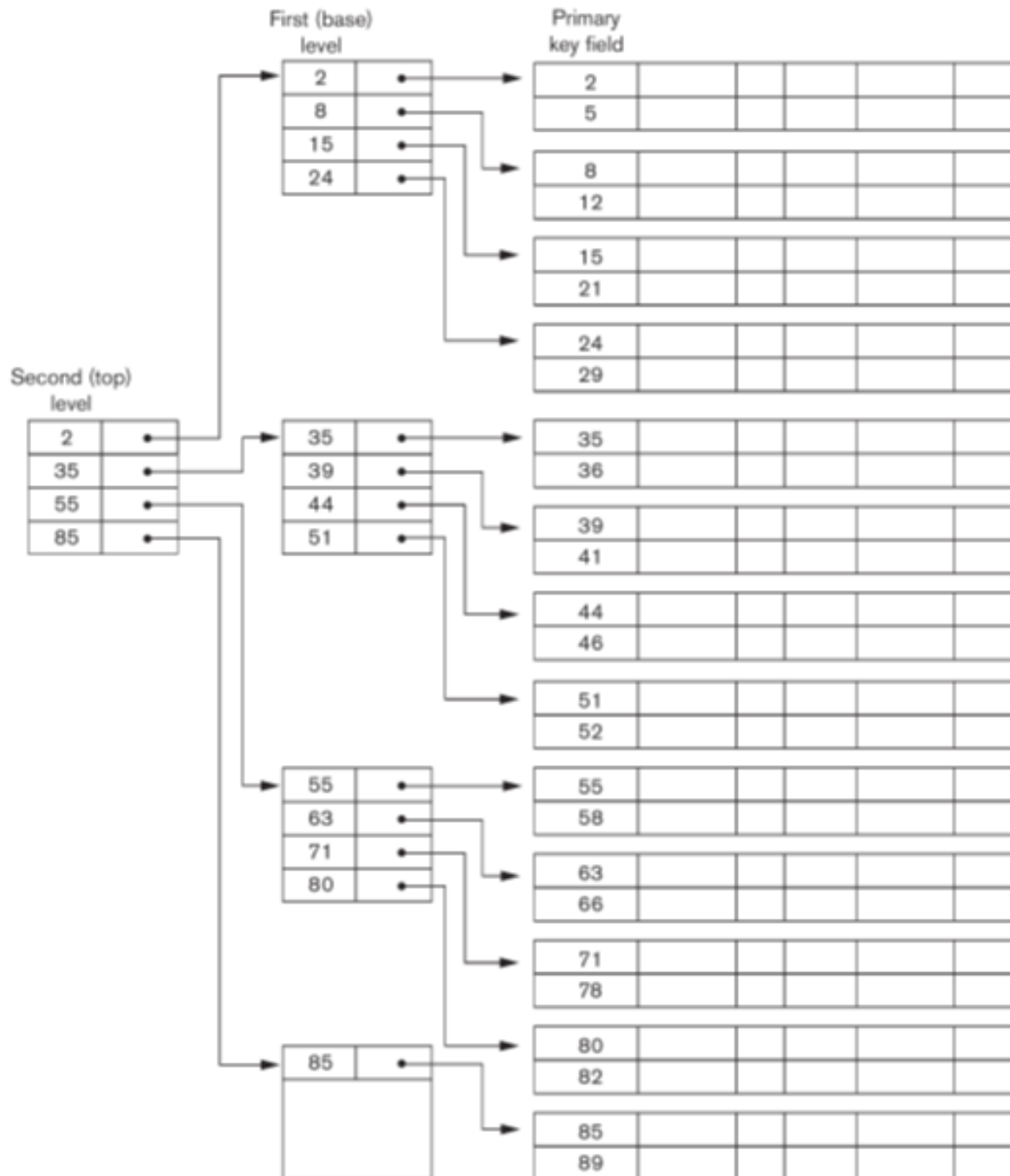
---

- ▶ Because a single-level index is an ordered file, we can **create a primary index to the index itself**.
  - ▶ The original index file is called the *first-level index* and the index to the index is called the *second-level index*.
- ▶ We can repeat the process, creating a third, fourth, ..., top level **until all entries of the top level fit in one disk block**.
- ▶ A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block.



## Two-level index

## Data file



**A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.**

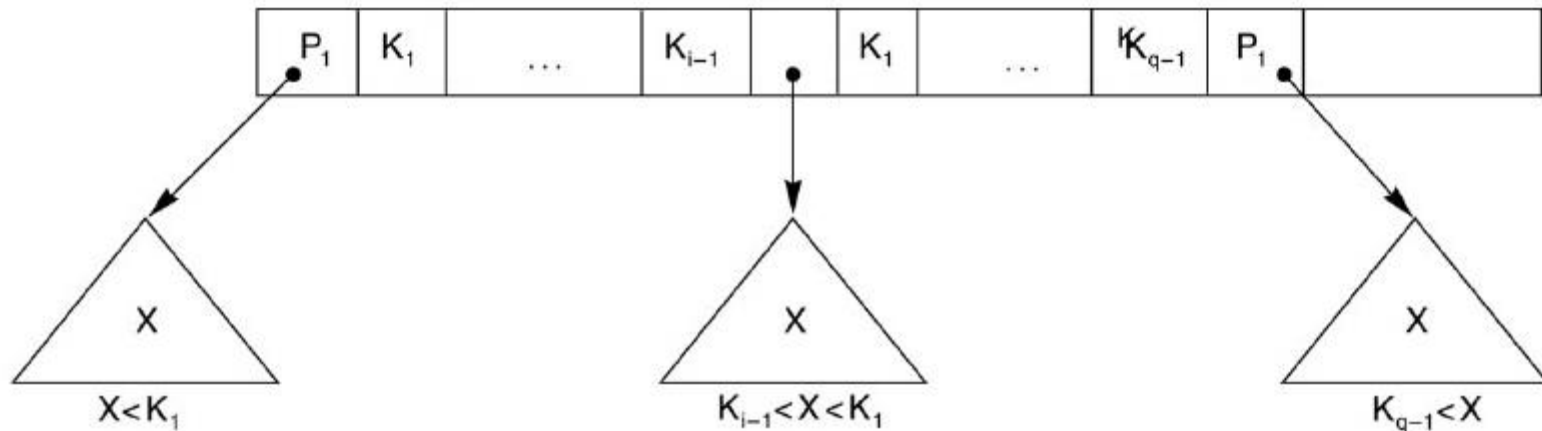
# Multi-Level Indexes

---

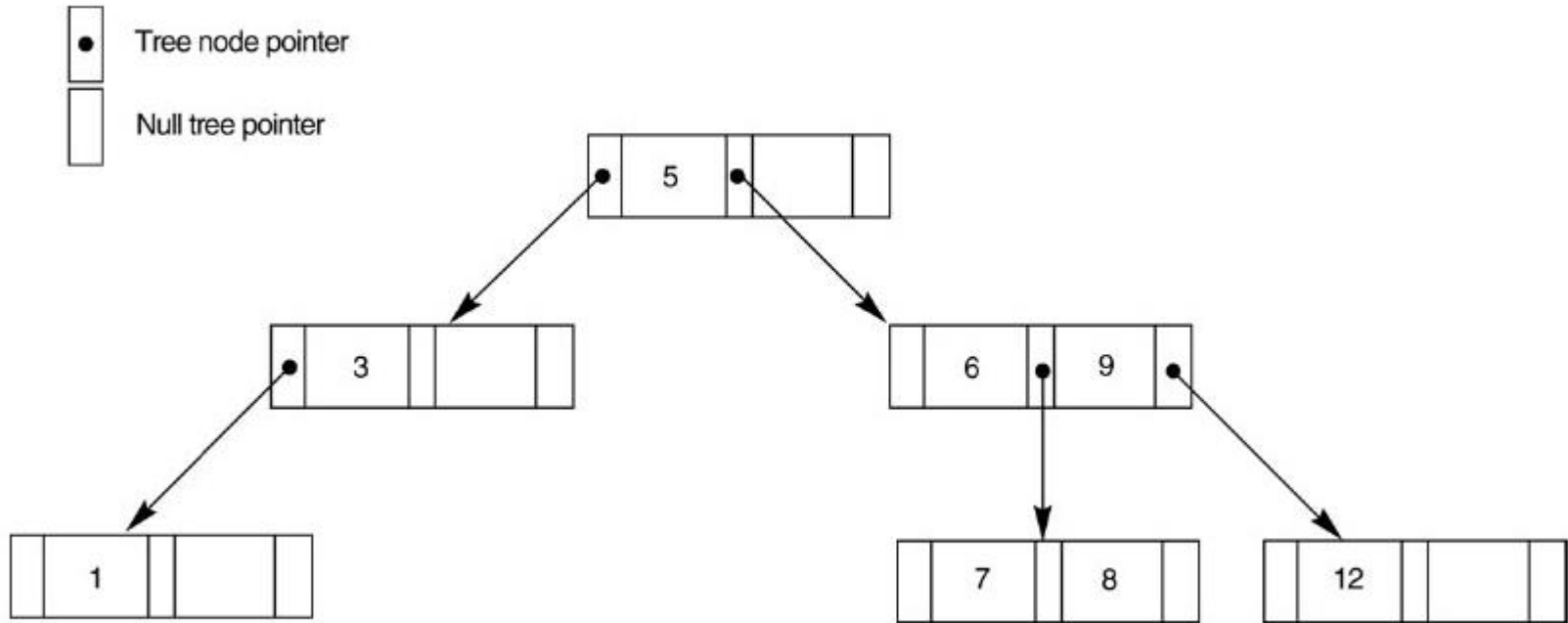
- ▶ Such a multi-level index is a form of *search tree*.
- ▶ However, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file*.

# A Node in a Search Tree with Pointers to Subtrees below It

---



# A search tree of order $p = 3$



# Contents

## 1 Data Storage

### 1.1 Disk Storage Devices

### 1.2 Files of Records

### 1.3 Operations on Files

### 1.4 Unordered Files & Ordered Files & Hashed Files

### 1.5 RAID Technology and Storage Area Networks

## 2 Indexing Structures for Files

### 2.1 Types of Single-level Ordered Indexes

### 2.2 Multilevel Indexes

### 2.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

# Dynamic Multilevel Indexes Using B-Trees and B+-Trees

---

- ▶ Most multi-level indexes use B-tree or B+-tree data structures because of the insertion and deletion problem.
  - ▶ This leaves space in each tree node (disk block) to allow for new index entries
- ▶ These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- ▶ In B-Tree and B+-Tree data structures, each node corresponds to a disk block.
- ▶ Each node is kept between half-full and completely full.

# Dynamic Multilevel Indexes Using B-Trees and B+-Trees (cont.)

---

- ▶ An insertion into a node that is not full is quite efficient.
  - ▶ If a node is full, the insertion causes a split into two nodes.
- ▶ Splitting may propagate to other tree levels.
- ▶ A deletion is quite efficient if a node does not become less than half full.
- ▶ If a deletion causes a node to become less than half full, it must be merged with neighboring nodes.

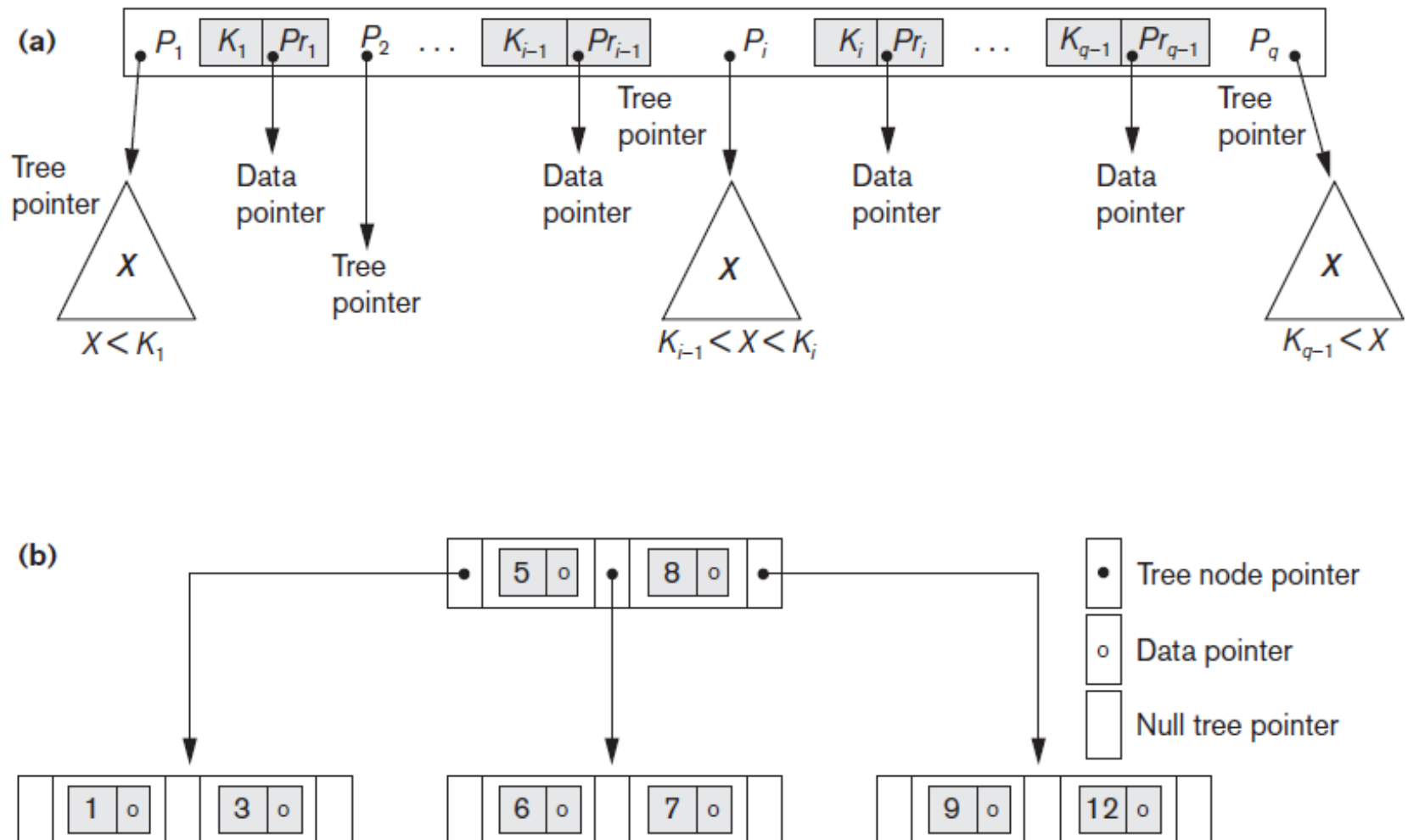
# Difference between B-tree and B+-tree

---

- ▶ In a B-Tree, pointers to data records exist at all levels of the tree.
- ▶ In a B<sup>+</sup>-Tree, all pointers to data records exist at the leaf-level nodes.
- ▶ A B<sup>+</sup>-Tree can have less levels (or higher capacity of search values) than the corresponding B-tree.



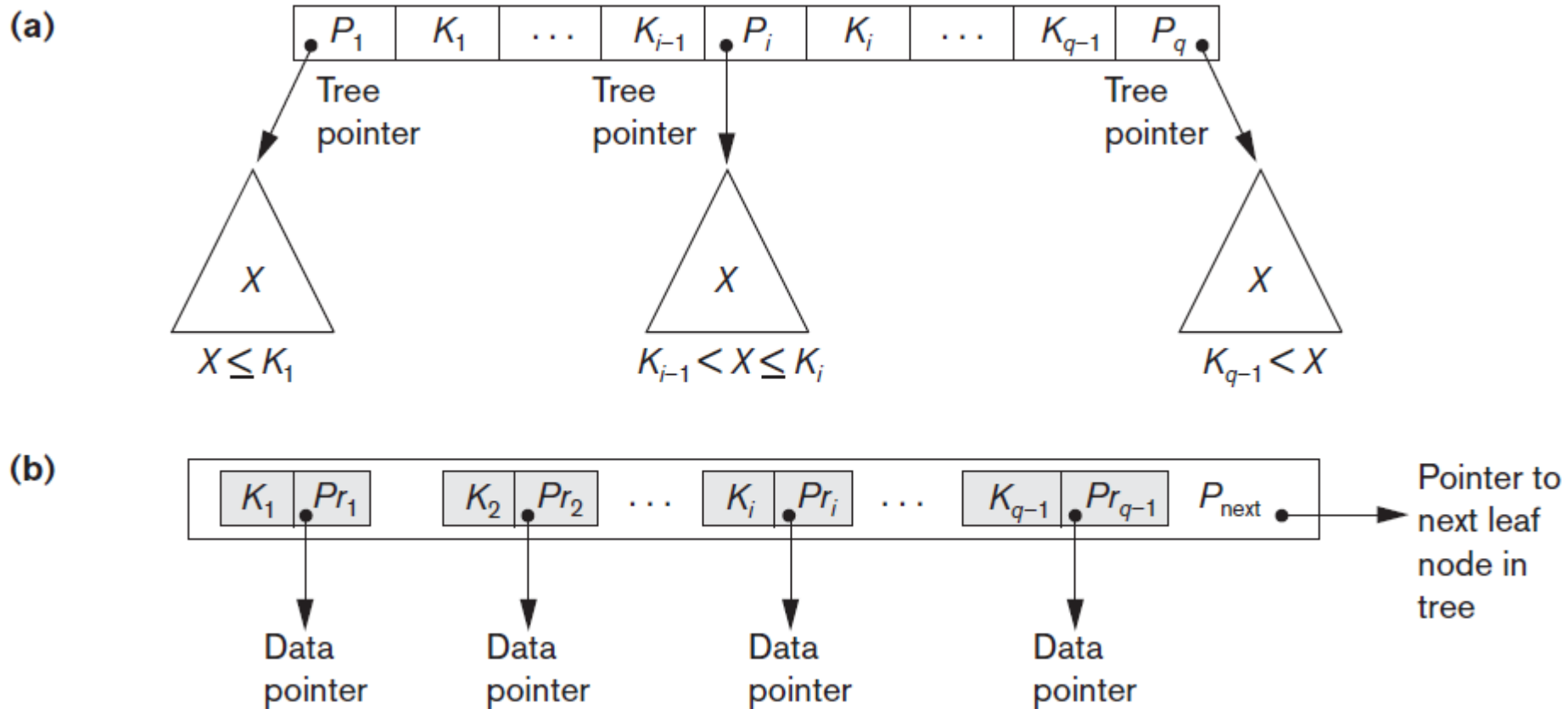
# B-tree Structures



**Figure 18.10**

B-tree structures. (a) A node in a B-tree with  $q - 1$  search values. (b) A B-tree of order  $p = 3$ . The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

# The Nodes of a B<sup>+</sup>-Tree



**Figure 18.11**

The nodes of a B<sup>+</sup>-tree. (a) Internal node of a B<sup>+</sup>-tree with  $q - 1$  search values. (b) Leaf node of a B<sup>+</sup>-tree with  $q - 1$  search values and  $q - 1$  data pointers.

# Contents

1	Data Storage
1.1	Disk Storage Devices
1.2	Files of Records
1.3	Operations on Files
1.4	Unordered Files & Ordered Files & Hashed Files
1.5	RAID Technology and Storage Area Networks
2	Indexing Structures for Files
2.1	Types of Single-level Ordered Indexes
2.2	Multilevel Indexes
2.3	Dynamic Multilevel Indexes Using B-Trees and B+-Trees



## Exercise

---

- ▶ A file has  $r = 30,000$  EMPLOYEE records of fixed length.
  - ▶ Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department\_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth\_date (8 bytes), Sex (1 byte), Job\_code (4 bytes), and Salary (4 bytes). An additional byte is used as a deletion marker.
  - ▶ Block size  $B = 512$  bytes, block pointer  $P = 6$  bytes, record pointer  $PR = 7$  bytes.
1. Calculate the **record size**  $R$  in bytes.
  2. Calculate the **blocking factor**  $bfr$  and the **number of file blocks**  $b$ , assuming an unspanned organization.

# Solution

---

1. Calculate the **record size**  $R$  in bytes.
  - ▶  $R = 30 + 9 + 9 + 40 + 10 + 8 + 1 + 4 + 4 + 1 = 116$  bytes.
2. Calculate the **blocking factor**  $bfr$  and the **number of file blocks**  $b$ , assuming an unspanned organization.
  - ▶  $bfr = \lfloor B/R \rfloor = \lfloor 512/116 \rfloor = 4$  (records/blocks)
  - ▶  $b = \lceil r/bfr \rceil = \lceil 30000/4 \rceil = 7500$  (blocks)

## Exercise (cont.)

---

- ▶ *A file has  $r = 30,000$  EMPLOYEE records of fixed length.*
  - ▶ *Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department\_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth\_date (8 bytes), Sex (1 byte), Job\_code (4 bytes), and Salary (4 bytes). An additional byte is used as a deletion marker.*
  - ▶ *Block size  $B = 512$  bytes, block pointer  $P = 6$  bytes, record pointer  $PR = 7$  bytes.*
3. This file is *ordered by* the key field SSN. Calculate the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value.

# Solution

---

3. This file is *ordered by* the key field SSN. Calculate the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value.
- ▶ Binary search
  - ▶ Number of blocks accesses:  $\lceil \log_2 b \rceil = \lceil \log_2 7500 \rceil = 13$



## Exercise (cont.)

---

- ▶ A file has  $r = 30,000$  EMPLOYEE records of fixed length.
  - ▶ Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department\_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth\_date (8 bytes), Sex (1 byte), Job\_code (4 bytes), and Salary (4 bytes). An additional byte is used as a deletion marker.
  - ▶ Block size  $B = 512$  bytes, block pointer  $P = 6$  bytes, record pointer  $PR = 7$  bytes.
4. This file is *ordered* by the key field SSN and we want to construct a primary index on SSN. Calculate:
- a) the ***index blocking factor  $bfr_i$***
  - b) the ***number of index entries*** and the ***number of index blocks***

# Solution

---

4. This file is *ordered by* the key field SSN and we want to construct a primary index on SSN. Calculate:
- a) the ***index blocking factor  $bfr_i$*** 
    - ▶ Index entry size:  $R_i = 9 + 6 = 15$  (bytes)
    - ▶ Index blocking factor  $bfr_i = \lfloor B / R_i \rfloor = \lfloor 512 / 15 \rfloor = 34$
  - b) the ***number of index entries*** and the ***number of index blocks***
    - ▶ Number of index entries:  $r_i = 7500$
    - ▶ Number of index blocks:  $b_i = \lceil 7500 / 34 \rceil = 221$

## Exercise (cont.)

---

- ▶ *A file has  $r = 30,000$  EMPLOYEE records of fixed length.*
  - ▶ *Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department\_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth\_date (8 bytes), Sex (1 byte), Job\_code (4 bytes), and Salary (4 bytes). An additional byte is used as a deletion marker.*
  - ▶ *Block size  $B = 512$  bytes, block pointer  $P = 6$  bytes, record pointer  $PR = 7$  bytes.*
4. This file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate:
- c) the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value

# Solution

---

4. This file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate:
- c) the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value.
    - ▶ Binary search on index file and 1 more blocks access to data file
    - ▶  $\lceil \log_2 b_i \rceil + 1 = \lceil \log_2 221 \rceil + 1 = 9$  blocks

## Exercise (cont.)

---

- ▶ *A file has  $r = 30,000$  EMPLOYEE records of fixed length.*
  - ▶ *Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department\_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth\_date (8 bytes), Sex (1 byte), Job\_code (4 bytes), and Salary (4 bytes). An additional byte is used as a deletion marker.*
  - ▶ *Block size  $B = 512$  bytes, block pointer  $P = 6$  bytes, record pointer  $PR = 7$  bytes.*
5. If we make it into a multilevel index (two levels).
- a) Calculate the ***total number of blocks*** required by the second index
  - b) the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value

# Solution

---

5. If we make it into a multilevel index (two levels).
- a) Calculate the ***total number of blocks*** required by the second index
- ▶ Number of 1<sup>st</sup> level index entries:  $r_{i_1} = 7500$
  - ▶ Number of 1<sup>st</sup> level index blocks:  $b_{i_1} = 221$
  - ▶ Number of 2<sup>nd</sup> level index entries:  $r_{i_2} = 221$
  - ▶ Number of 2<sup>nd</sup> level index blocks:  $b_{i_2} = \lceil 221/34 \rceil = 7$
- b) the ***number of block accesses*** needed to search for and retrieve a record from the file - given its SSN value
- ▶  $\lceil \log_2 b_{i_2} \rceil + 1 + 1 = \lceil \log_2 7 \rceil + 1 + 1 = 5$  blocks