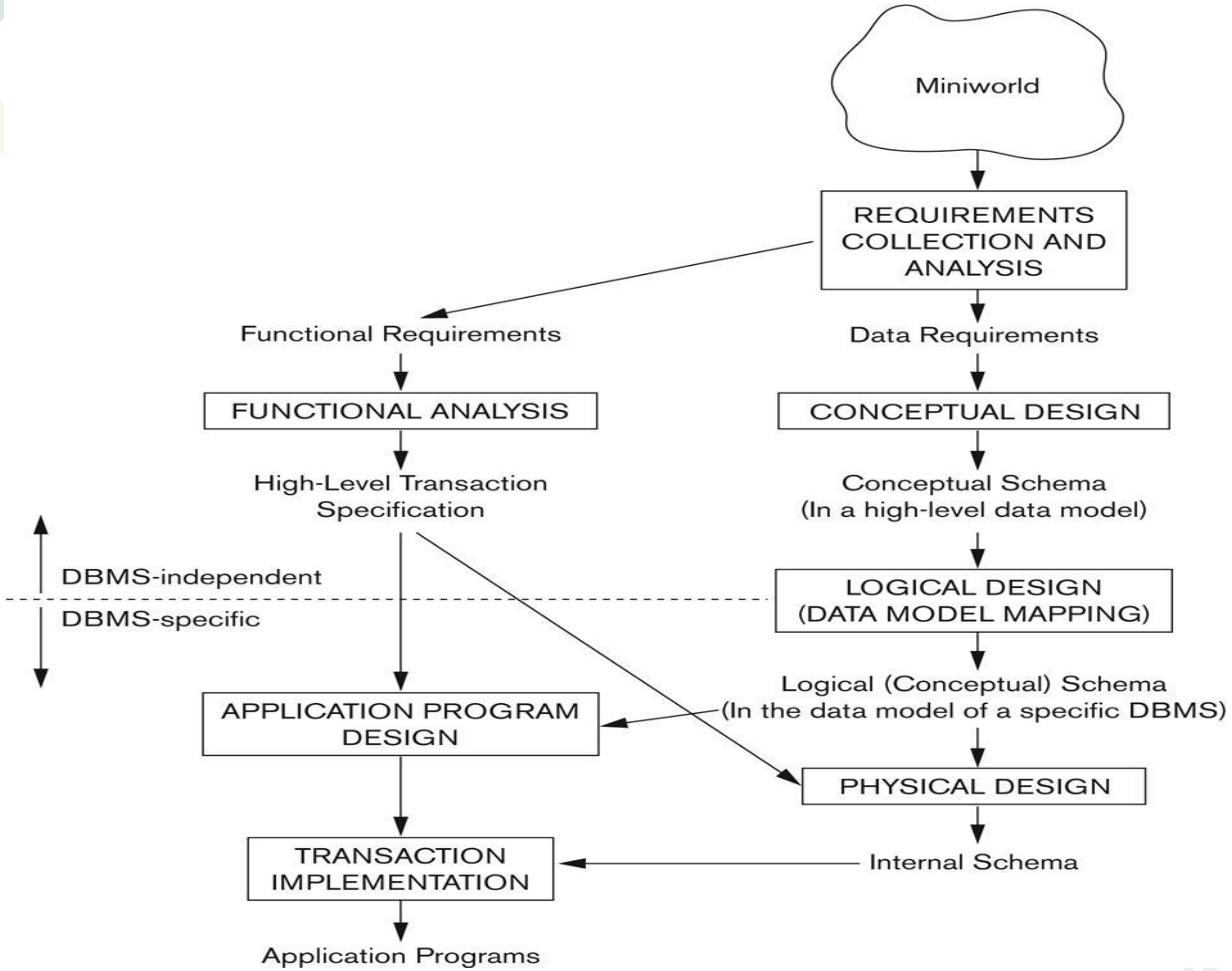
Chapter 8:

Data Storage, Indexing Structures for Files

Overview of Database Design Process



Outline

- Data Storage
  - Disk Storage Devices
  - Files of Records
  - Operations on Files
  - Unordered Files
  - Ordered Files
  - Hashed Files
  - RAID Technology
- Indexing Structures for Files
  - Types of Single-level Ordered Indexes
  - Multilevel Indexes
  - Dynamic Multilevel Indexes Using B-Trees and B+-Trees
  - Indexes on Multiple Keys

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
- 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.)

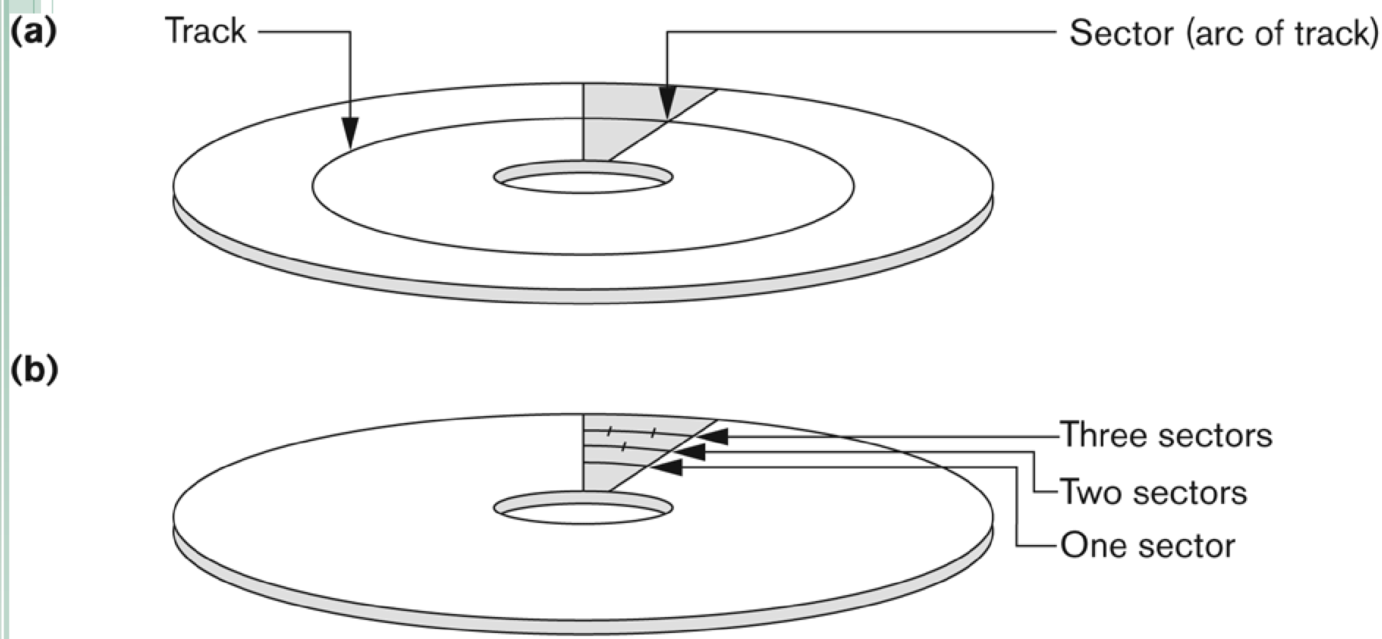


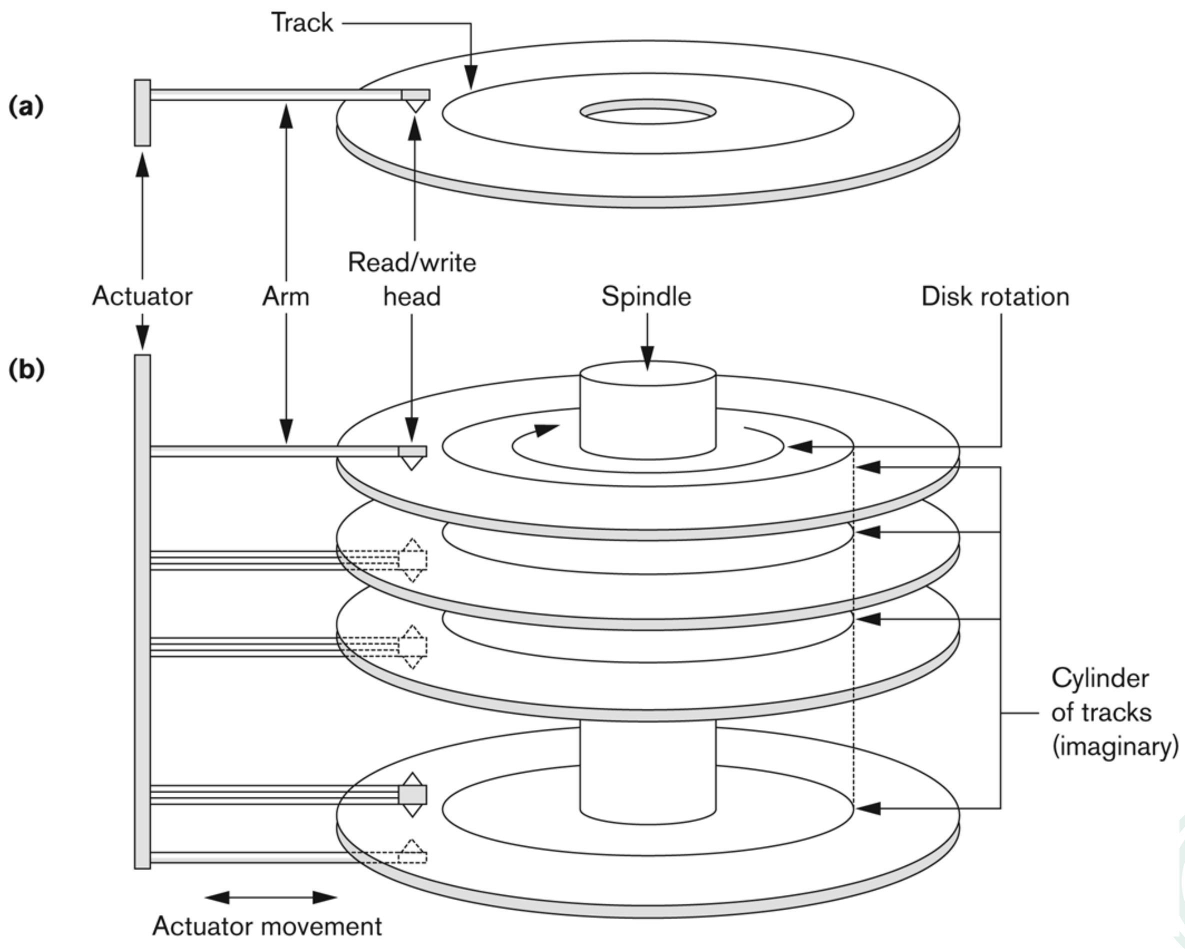
Figure 13.2
Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.

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.

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

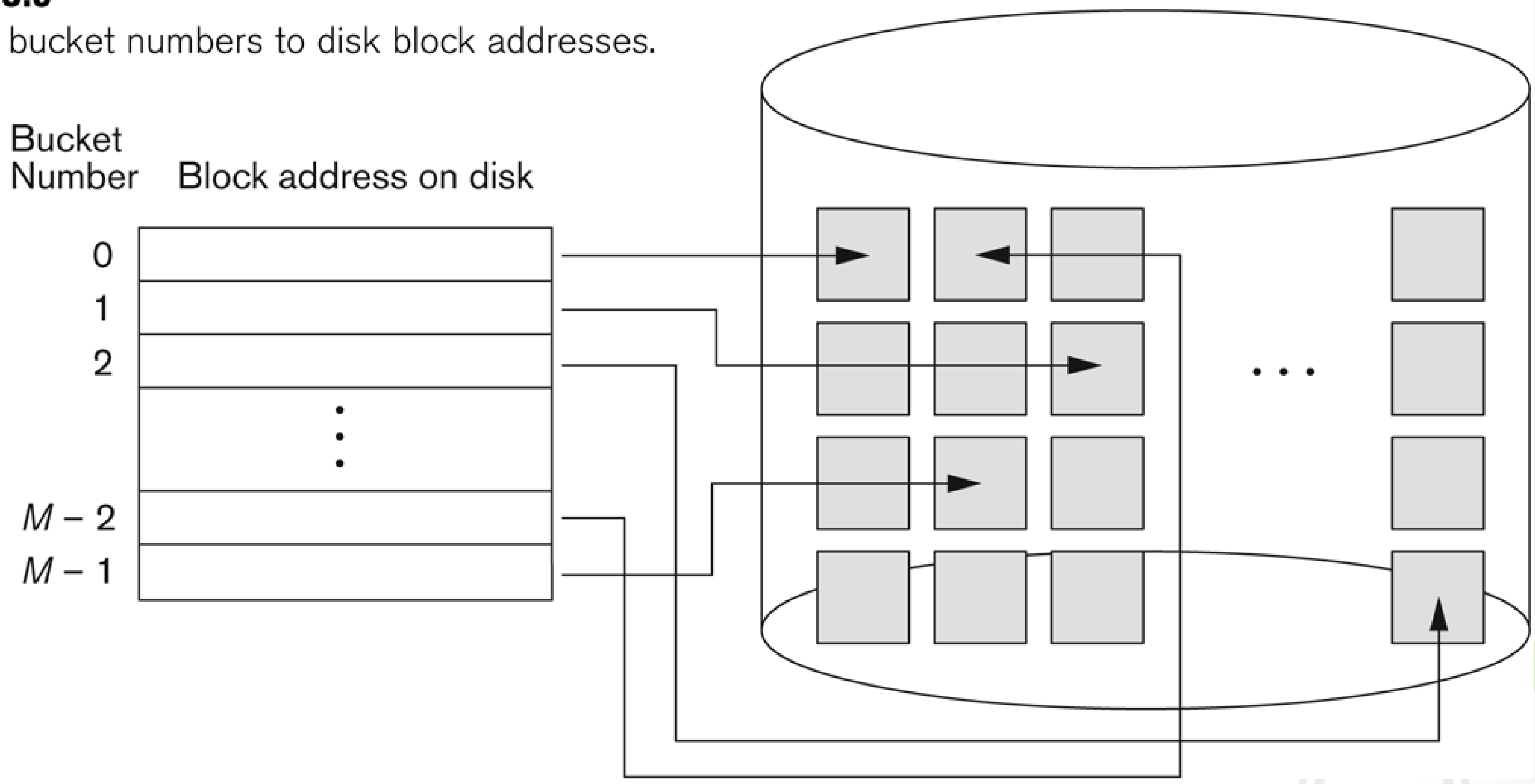
TYPE OF ORGANIZATION	ACCESS/SEARCH METHOD	AVERAGE TIME TO ACCESS A SPECIFIC RECORD
Heap (Unordered)	Sequential scan (Linear Search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary Search	$\log_2 b$

Hashed Files

- Hashing for disk files is called **External Hashing**
- The file blocks are divided into M equal-sized **buckets**, numbered $bucket_0$, $bucket_1$, ..., $bucket_{M-1}$.
  - 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 (contd.)

Figure 13.9
Matching bucket numbers to disk block addresses.



Hashed Files (contd.)

- To reduce overflow records, a hash file is typically kept 70-80% full.
- The hash function h should distribute the records uniformly among the buckets
  - Otherwise, search time will be increased because many overflow records will exist.
- Main disadvantages of static external hashing:
  - Fixed number of buckets M is a problem if the number of records in the file grows or shrinks.
  - Ordered access on the hash key is quite inefficient (requires sorting the records).

Hashed Files - Overflow handling

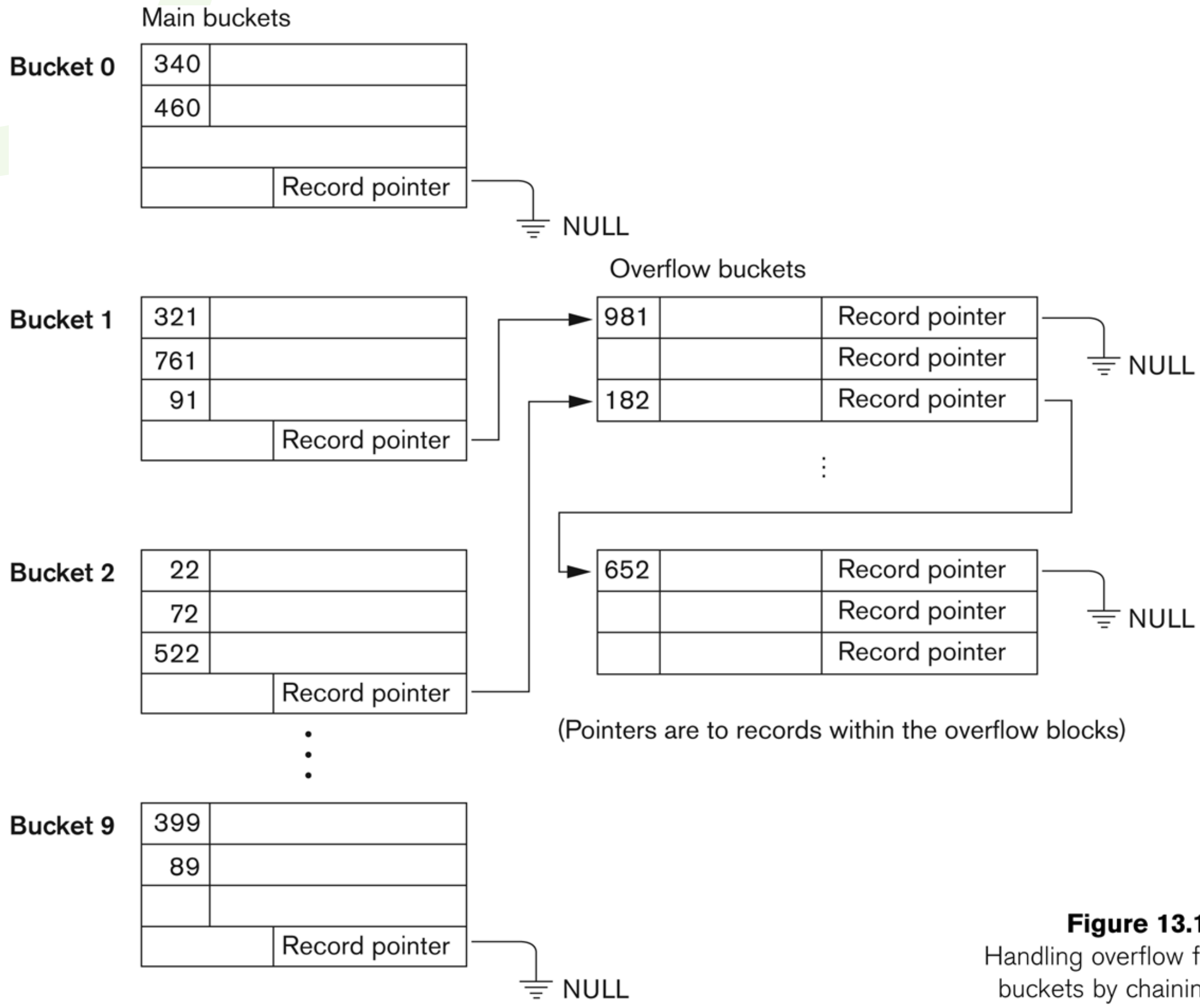


Figure 13.10
Handling overflow for buckets by chaining.

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 (contd.)

- 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.

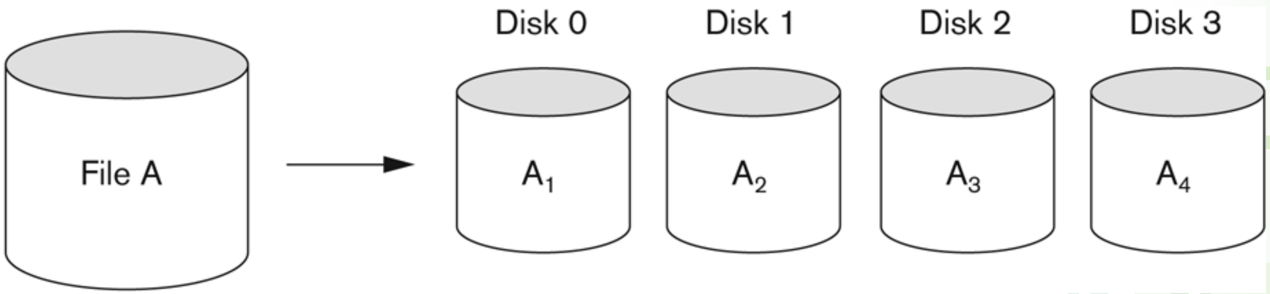
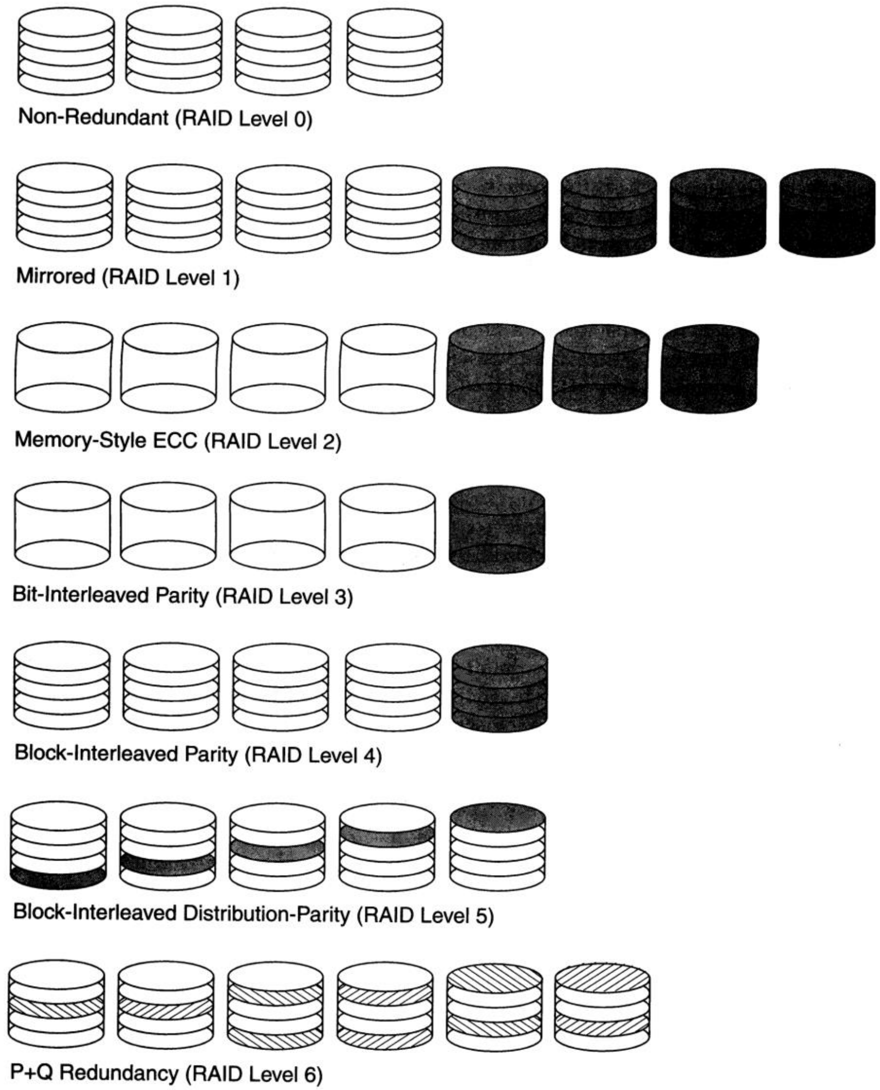


Figure 13.12
Data striping. File A is striped across four disks.

Use of RAID Technology (contd.)



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 (contd.)

- 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 non-disruptive 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.

Outline

- Disk Storage, Basic File Structures, and Hashing
  - Disk Storage Devices
  - Files of Records
  - Operations on Files
  - Unordered Files
  - Ordered Files
  - Hashed Files
  - RAID Technology
- Indexing Structures for Files
  - Types of Single-level Ordered Indexes
  - Multilevel Indexes
  - Dynamic Multilevel Indexes Using B-Trees and B+-Trees
  - Indexes on Multiple Keys

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 (contd.)

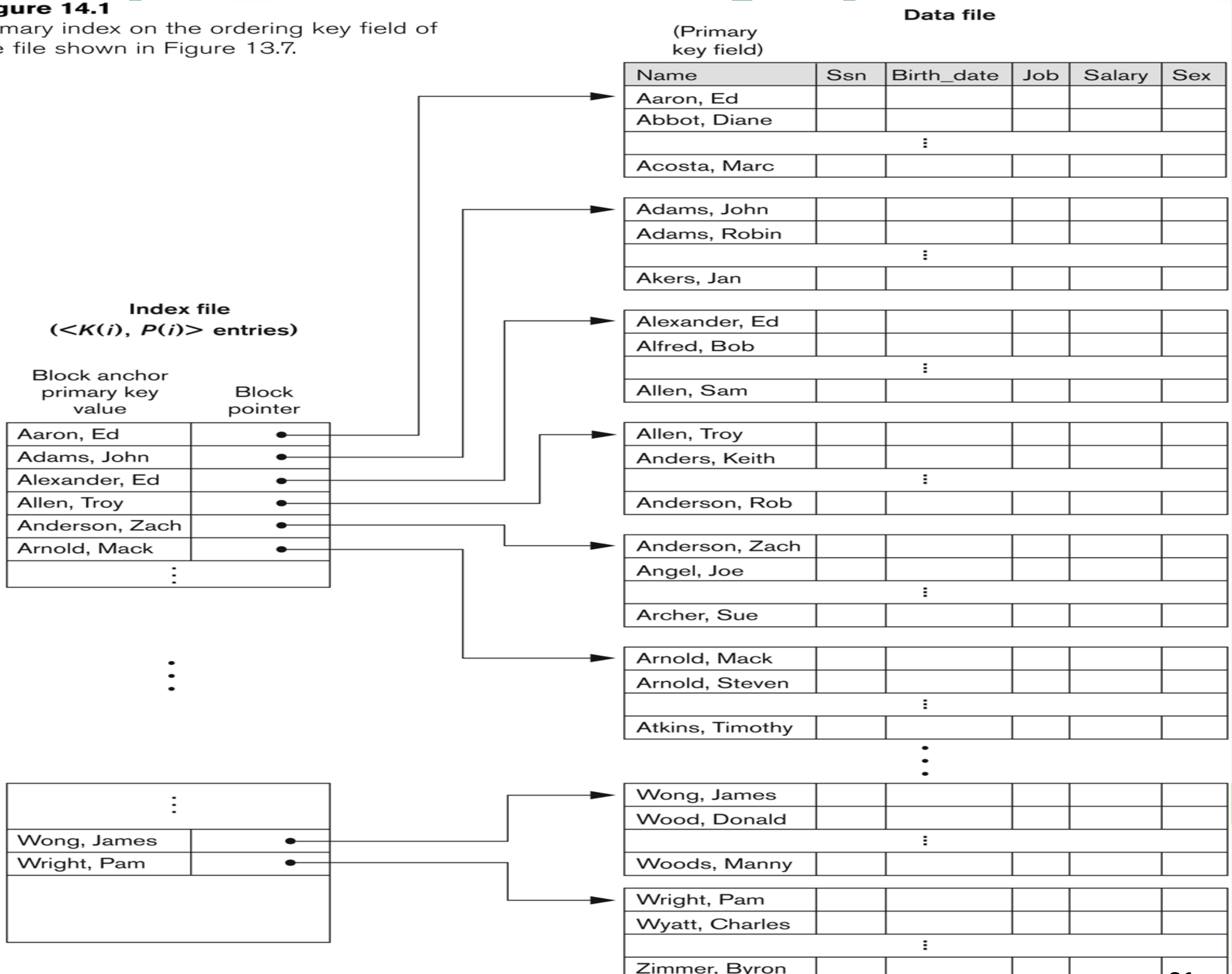
- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- Indexes can also be characterized as dense or sparse
  - A **dense index** has an index entry for every search key 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

Types of Single-Level Indexes

- Primary Index
  - Defined on an ordered data file
  - The data file is ordered on a **key field**
  - Includes one index entry *for each block* in the data file; the index entry has the key field value for the *first record* in the block, which is called the *block anchor*
  - A similar scheme can use the *last record* in a block.
  - A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.

Primary index on the ordering key field

Figure 14.1
Primary index on the ordering key field of the file shown in Figure 13.7.



Types of Single-Level Indexes

- Example: Given the following data file:

  EMPLOYEE(NAME,SSN, ADDRESS,JOB,SAL,...)
- Suppose that:
  - record size: R= 150 bytes
  - block size: B= 512 bytes
  - Number of records: r = 30000 records
- Then, we get:
  - blocking factor Bfr = $\lfloor (B/R) \rfloor = \lfloor (512/150) \rfloor$ = 3 records/block
  - number of file blocks b= $\lceil (r/Bfr) \rceil = \lceil (30000/3) \rceil$ =10000 blocks

Types of Single-Level Indexes

- For a **primary index** on the **ordering key field** SSN, assume the field size V_{SSN}= 9 bytes and the block pointer size P = 6 bytes. Then:
  - 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 index blocks $b_I=\lceil (b/Bfr_I) \rceil = \lceil (10000/34) \rceil = 295$ blocks
  - binary search needs $\log_2 b_I = \log_2 295 = 9$ block accesses
  - To search for a record using the index, we need one additional block access to the data file for a total of 9+1 = 10 block accesses
- This is compared to an average cost of:
  - Linear search: $\lceil (b/2) \rceil = \lceil 10000/2 \rceil = 5000$ block accesses
  - The binary search: $\lceil \log_2 b \rceil = \lceil \log_2 10000 \rceil =14$ block accesses

Types of Single-Level Indexes

- Clustering Index
  - Defined on an ordered data file
  - The data file is ordered on a *non-key field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
  - Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
  - It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.

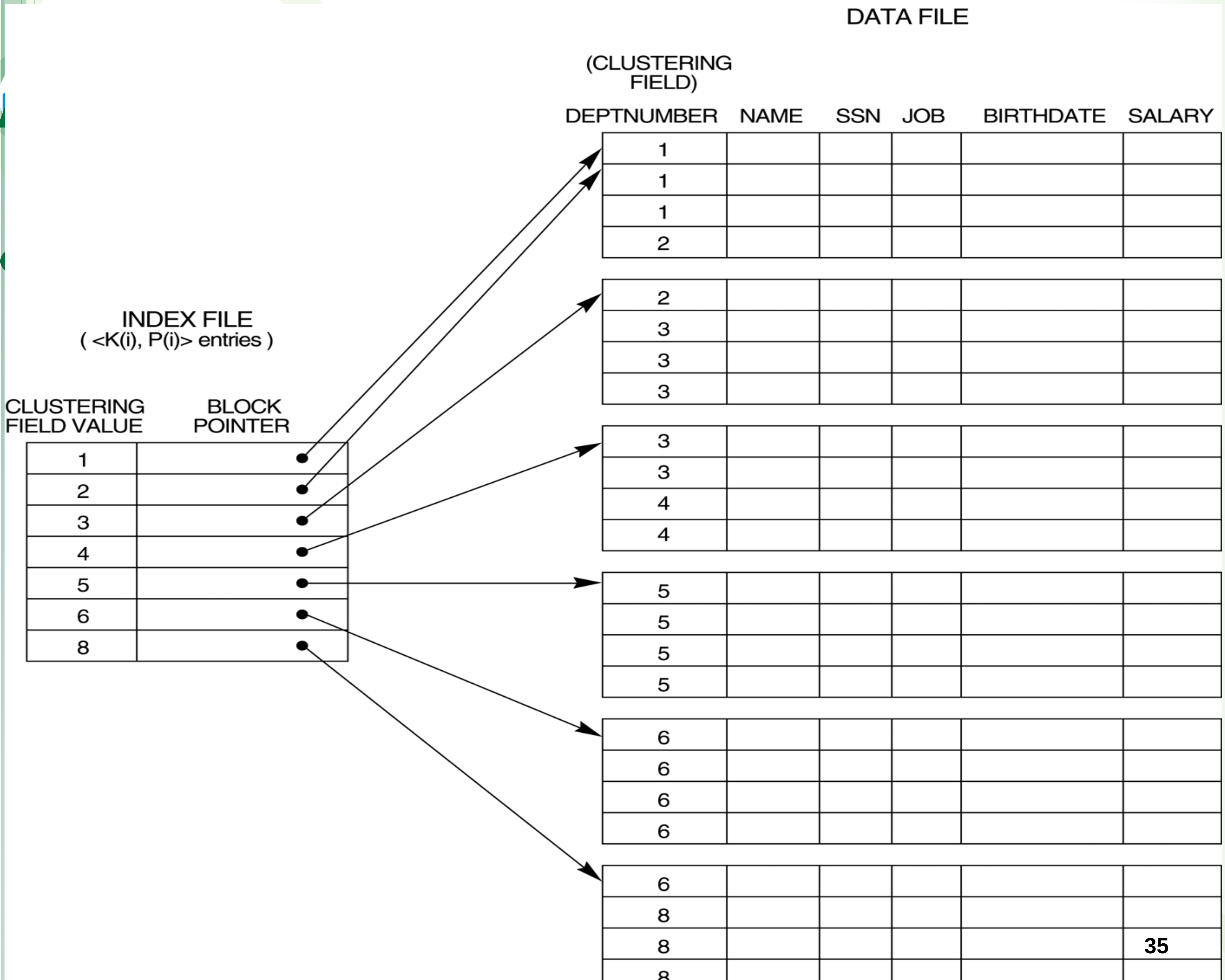
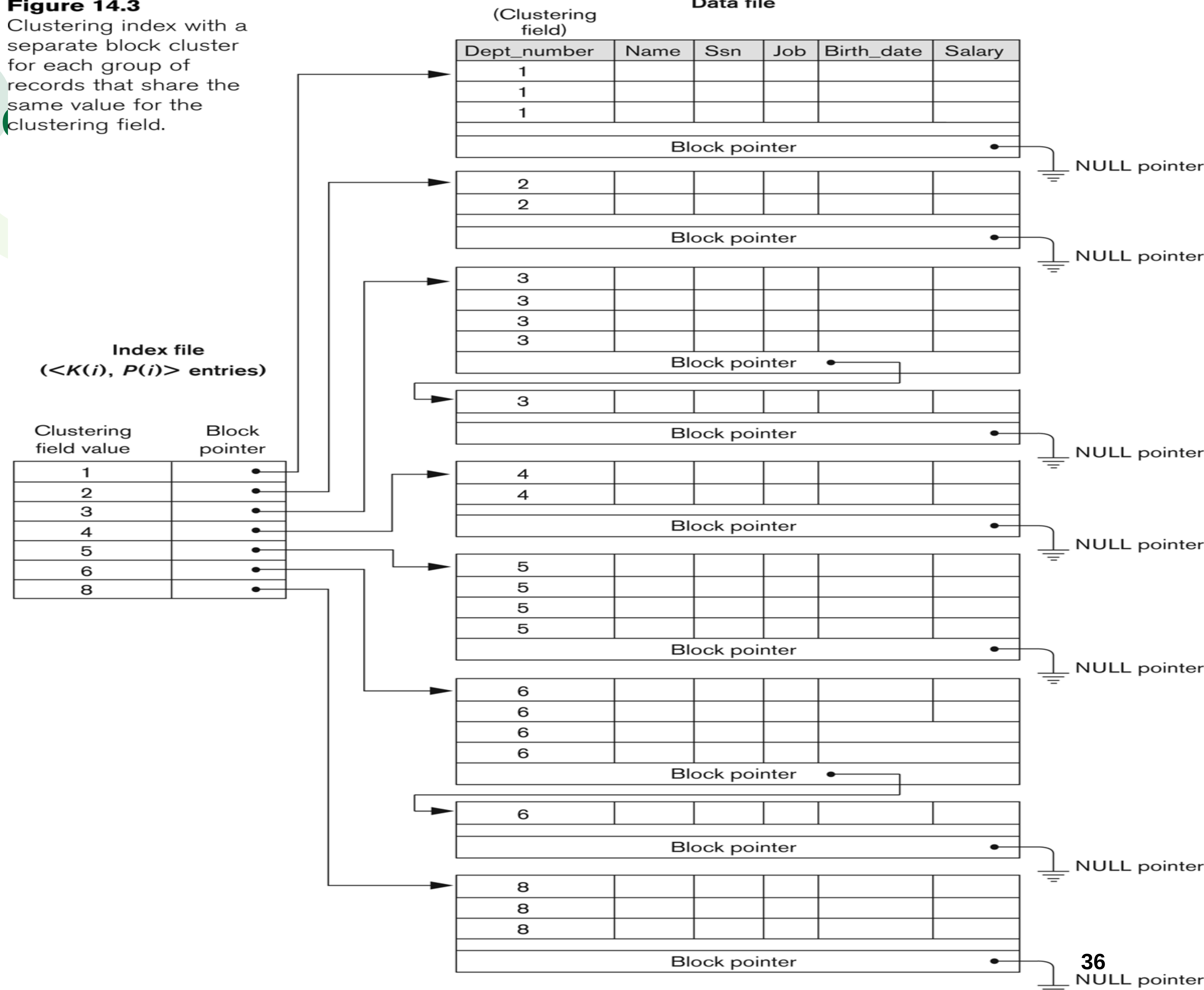
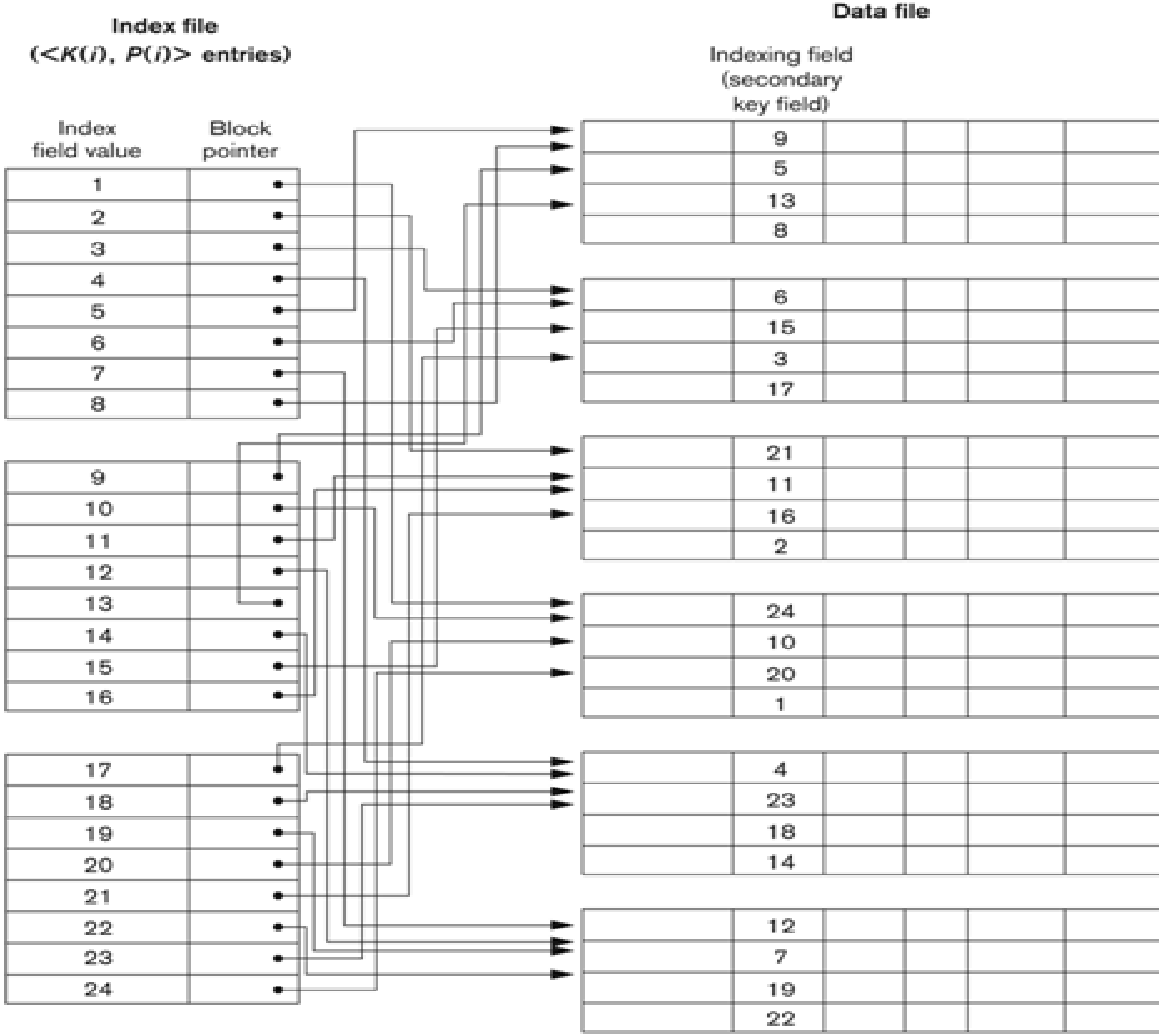

DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER
NAME
SSN
JOB
BIRTHDATE
SALARY
1
1
1
2
2
3
3
3
3
3
4
4
5
5
5
5
6
6
6
6
6
8
8
8
INDEX FILE
(<K(i), P(i)> entries)
CLUSTERING FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8

Figure 14.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.



Types of Single-Level Indexes

- Secondary Index
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some **non-ordering field** of the data file that is an indexing field.
    - The second field is either a **block** pointer or a record pointer.
    - There can be *many* secondary indexes (and hence, indexing fields) for the same file.
  - Includes one entry *for each record* in the data file; hence, it is a *dense index*


Index file
(<K(i), P(i)> entries)
Data file
Indexing field
(secondary
key field)
Index
field value
Block
pointer
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
9
5
13
8
6
15
3
17
21
11
16
2
24
10
20
1
4
23
18
14
12
7
19
22

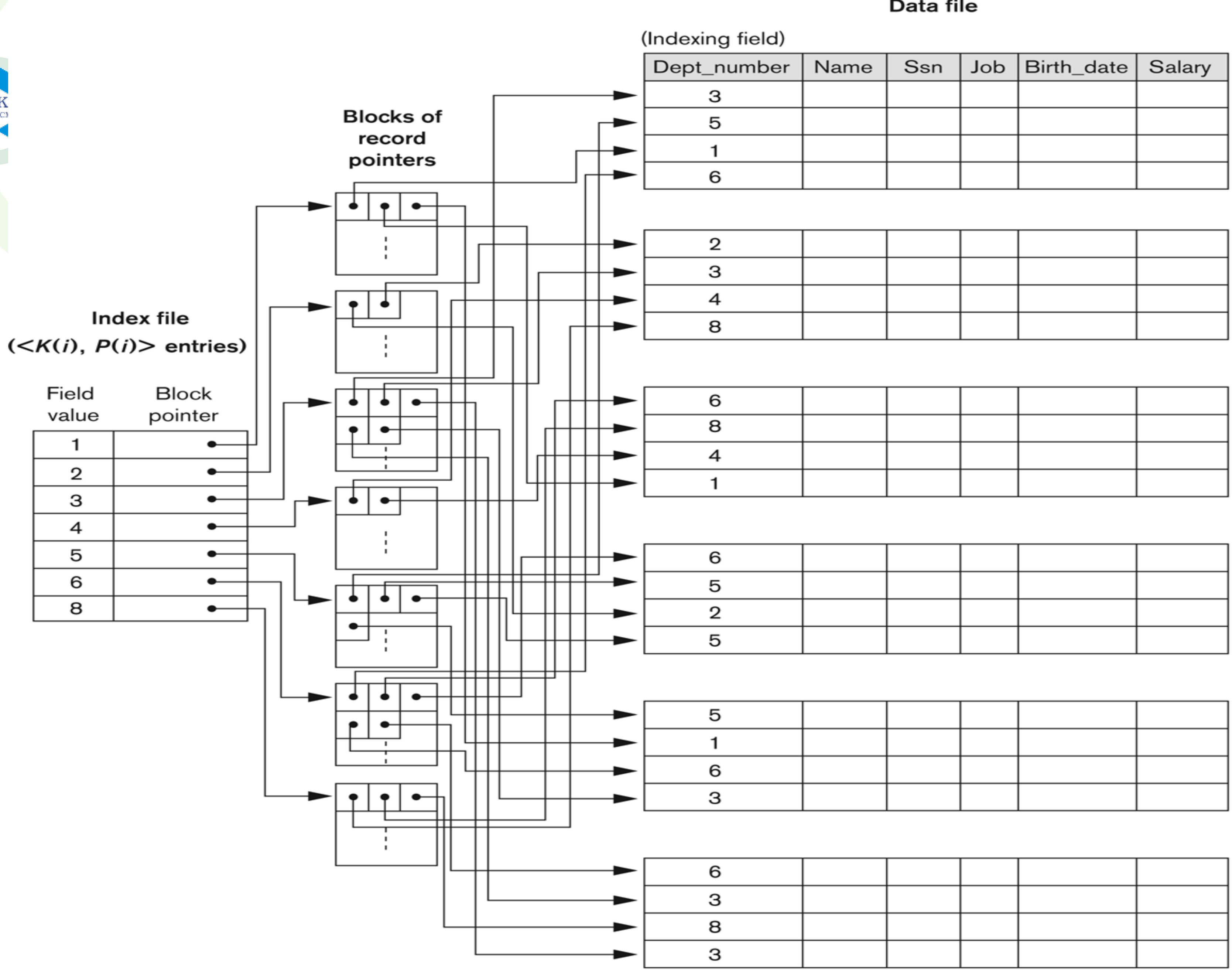


Figure 14.5
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

Properties of Index Types

TABLE 14.2 PROPERTIES OF INDEX TYPES

TYPE OF INDEX	NUMBER OF (FIRST-LEVEL) INDEX ENTRIES	DENSE OR NONDENSE	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[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or Number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.

[b]For option 1.

[c]For options 2 and 3.

Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself*;
  - In this case, 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

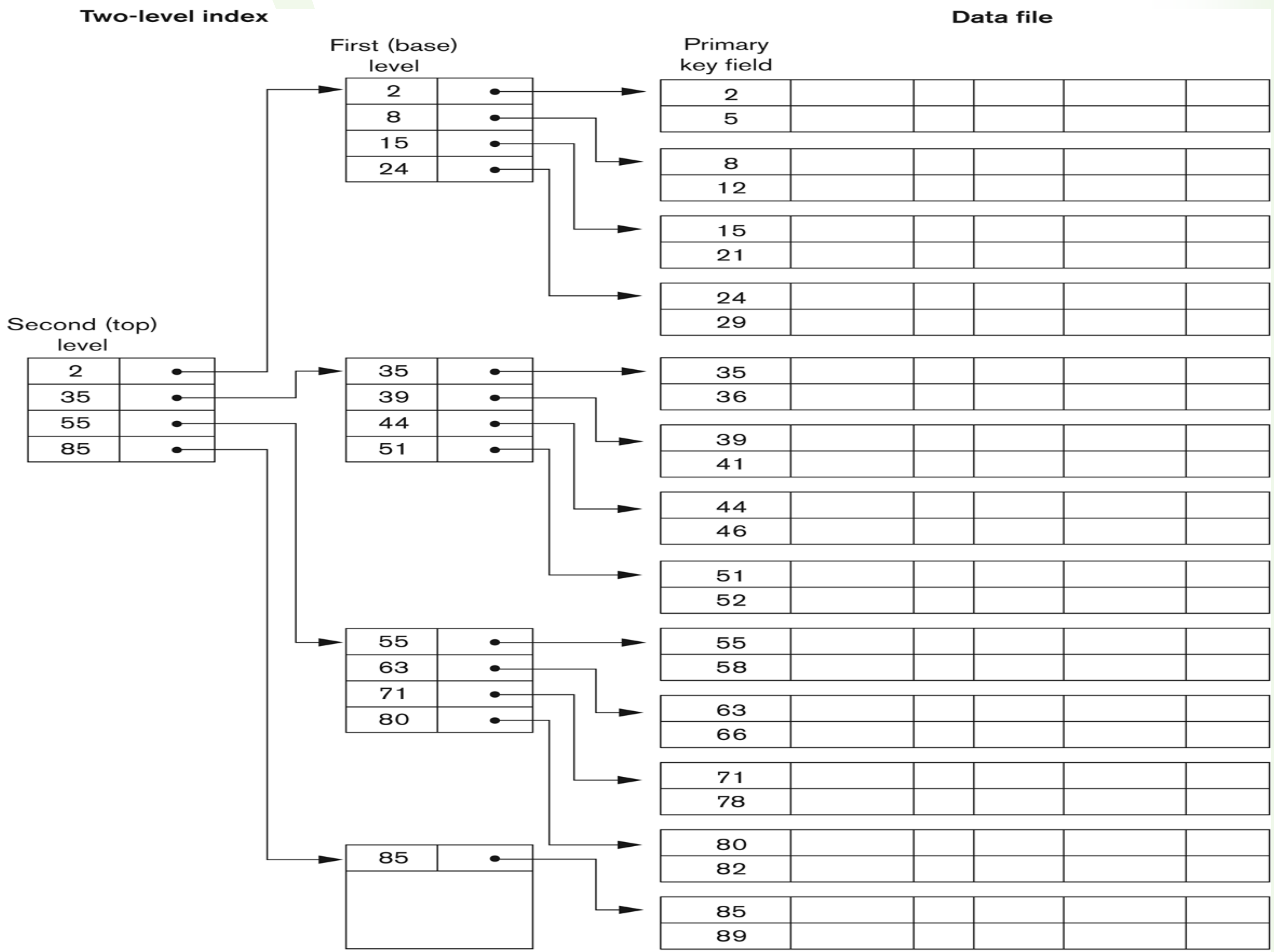


Figure 14.6
A two-level primary index resembling ISAM (Index Sequential Access Method) organization.

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

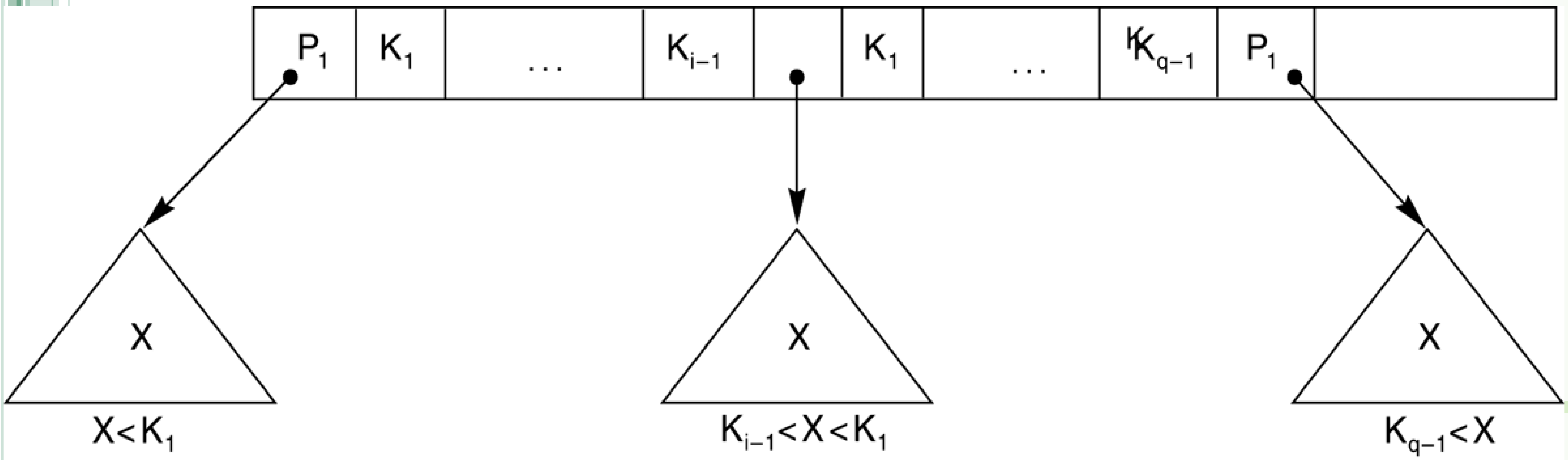
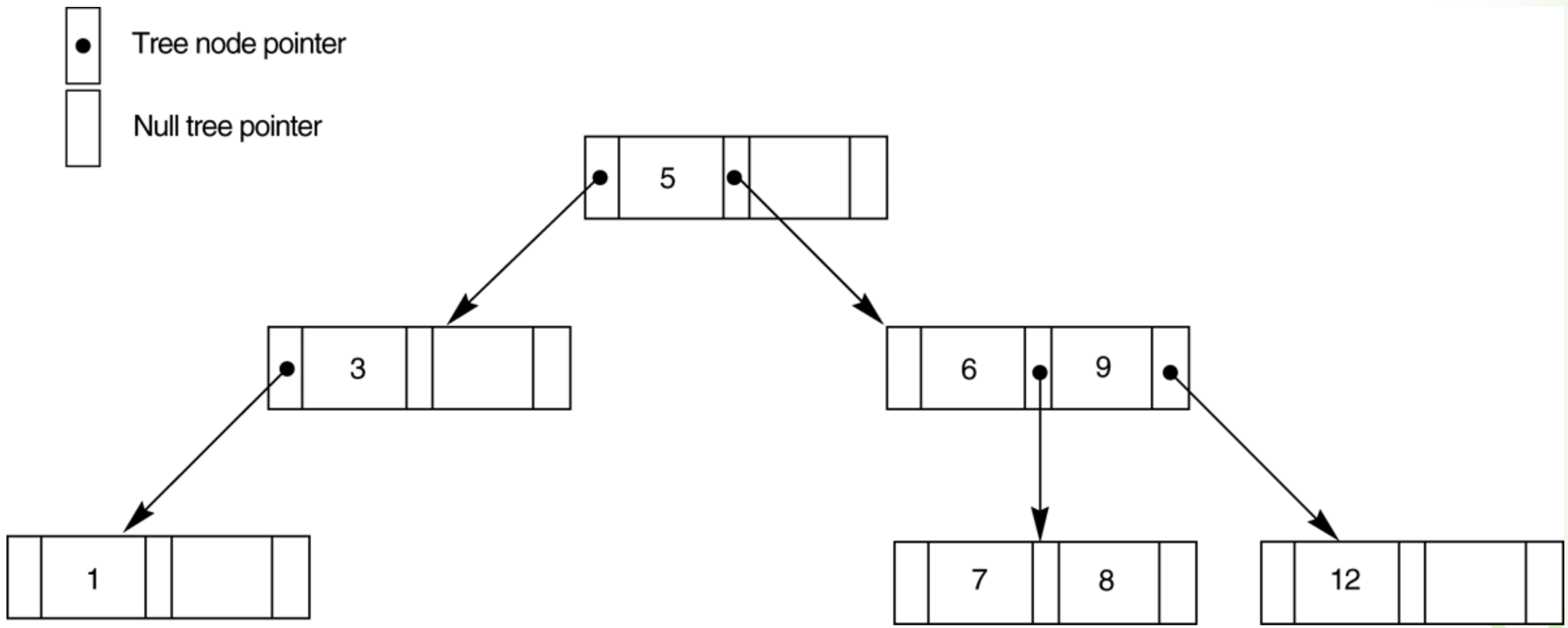


FIGURE 14.9
A search tree of order p = 3.



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 (contd.)

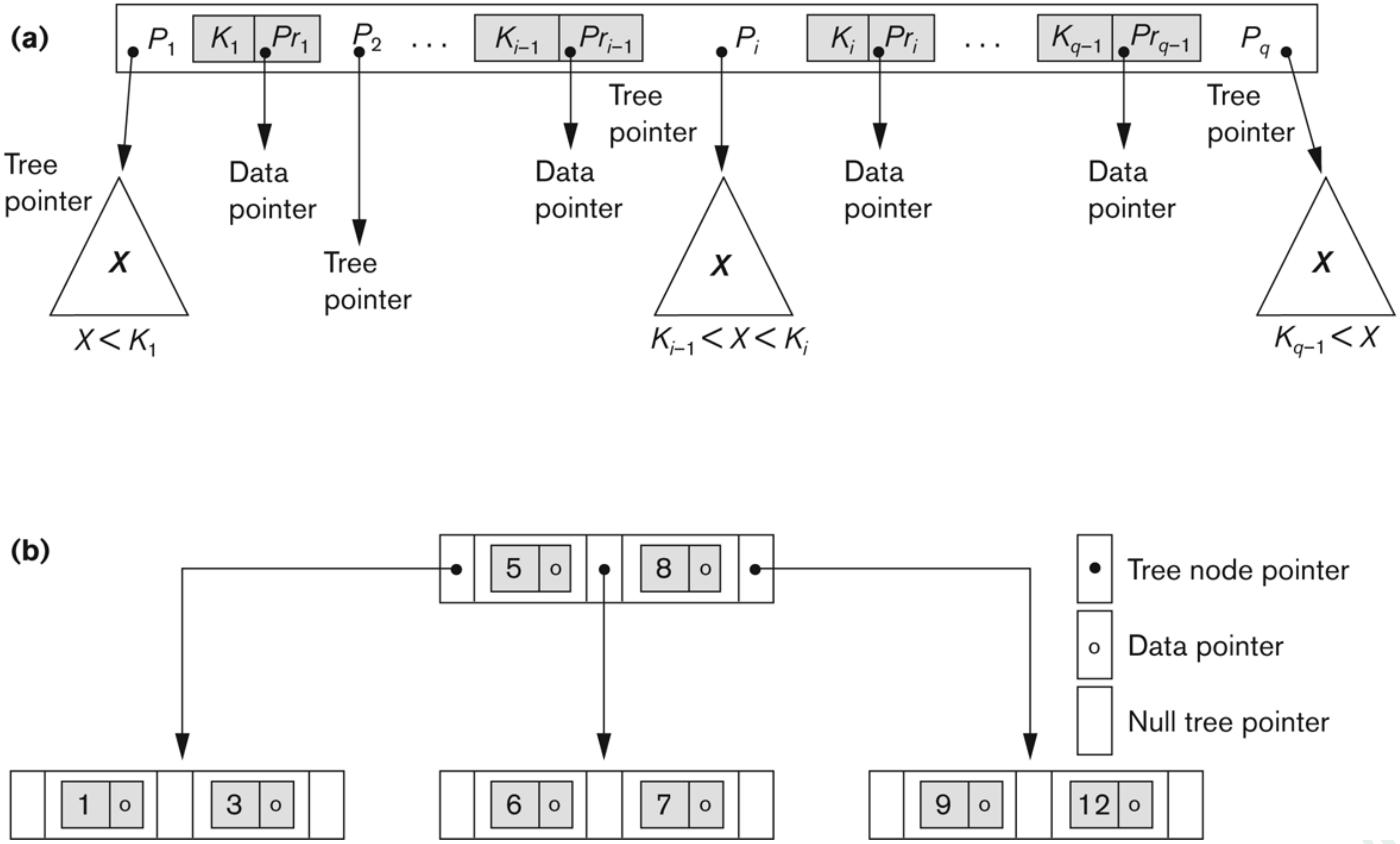
- An insertion into a node that is not full is quite efficient
  - If a node is full the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- A deletion is quite efficient if a node does not become less than half full
- If a deletion causes a node to become less than 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+-tree, all pointers to data records exists at the leaf-level nodes
- A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree

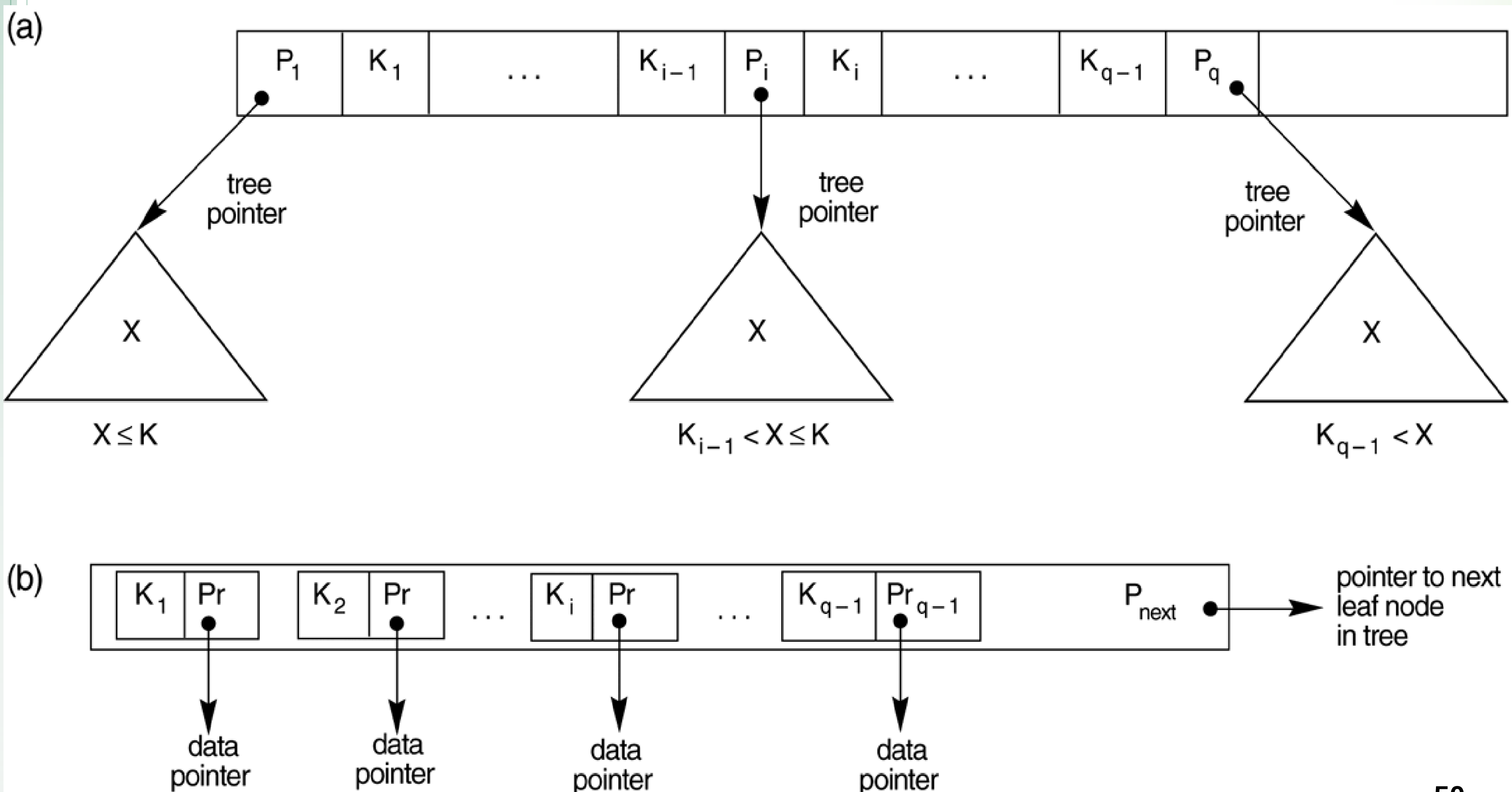
B-tree Structures

Figure 14.10
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+-tree

- FIGURE 14.11 The nodes of a B+-tree
  - (a) Internal node of a B+-tree with q –1 search values.
  - (b) Leaf node of a B+-tree with q – 1 search values and q – 1 data pointers.



Summary

- Data Storage
  - Disk Storage Devices
  - Files of Records
  - Operations on Files
  - Unordered Files
  - Ordered Files
  - Hashed Files
  - RAID Technology
- Indexing Structures for Files
  - Types of Single-level Ordered Indexes
  - Multilevel Indexes
  - Dynamic Multilevel Indexes Using B-Trees and B+-Trees
  - Indexes on Multiple Keys

BK
TP.HCM

Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), 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, real number).An additional byte is used as a deletion marker.

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.

Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), 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, real number).An additional byte is used as a deletion marker.

3. Suppose that the file is ordered by the key field Ssn Calculate

A. the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value

Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), 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, real number).An additional byte is used as a deletion marker.

4. Suppose that the file is ordered by the key field Ssn and we want to construct a primary index on Ssn. Calculate

A. The index blocking factor bfri

B. the number of first-level index entries and the number of first-level index blocks

Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), 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, real number).An additional byte is used as a deletion marker.

4. Suppose that the 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

Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), 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, real number).An additional byte is used as a deletion marker.

5. If we make it into a multilevel index (two levels).

   A. Calculate the total number of blocks required by the multilevel index;

   B. the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value