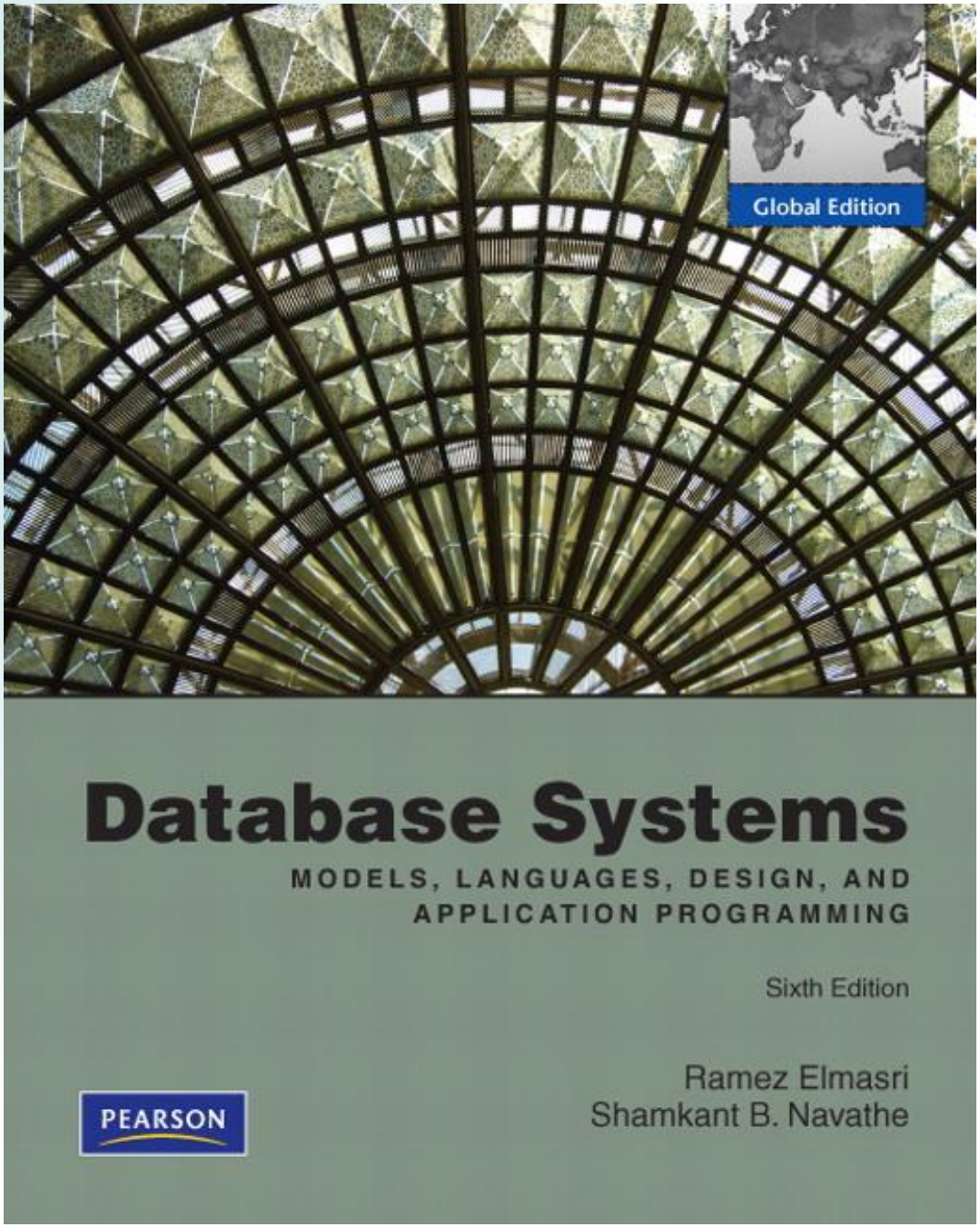
Chapter 16

Database File Organizations: Unordered, Ordered, and Hashed Files of Records



Addison-Wesley
is an imprint of
PEARSON

Index

- Records / Blocking
- Files of Records
- Operations on Files
- Unordered Files
- Ordered Files
- Hashed Files
  - Dynamic and Extendible Hashing Techniques

Records

- Records contain **fields** which have values of a particular type
  - E.g., amount, date, time, age
- Fixed and variable length records
  - *Oracle???*
- 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”

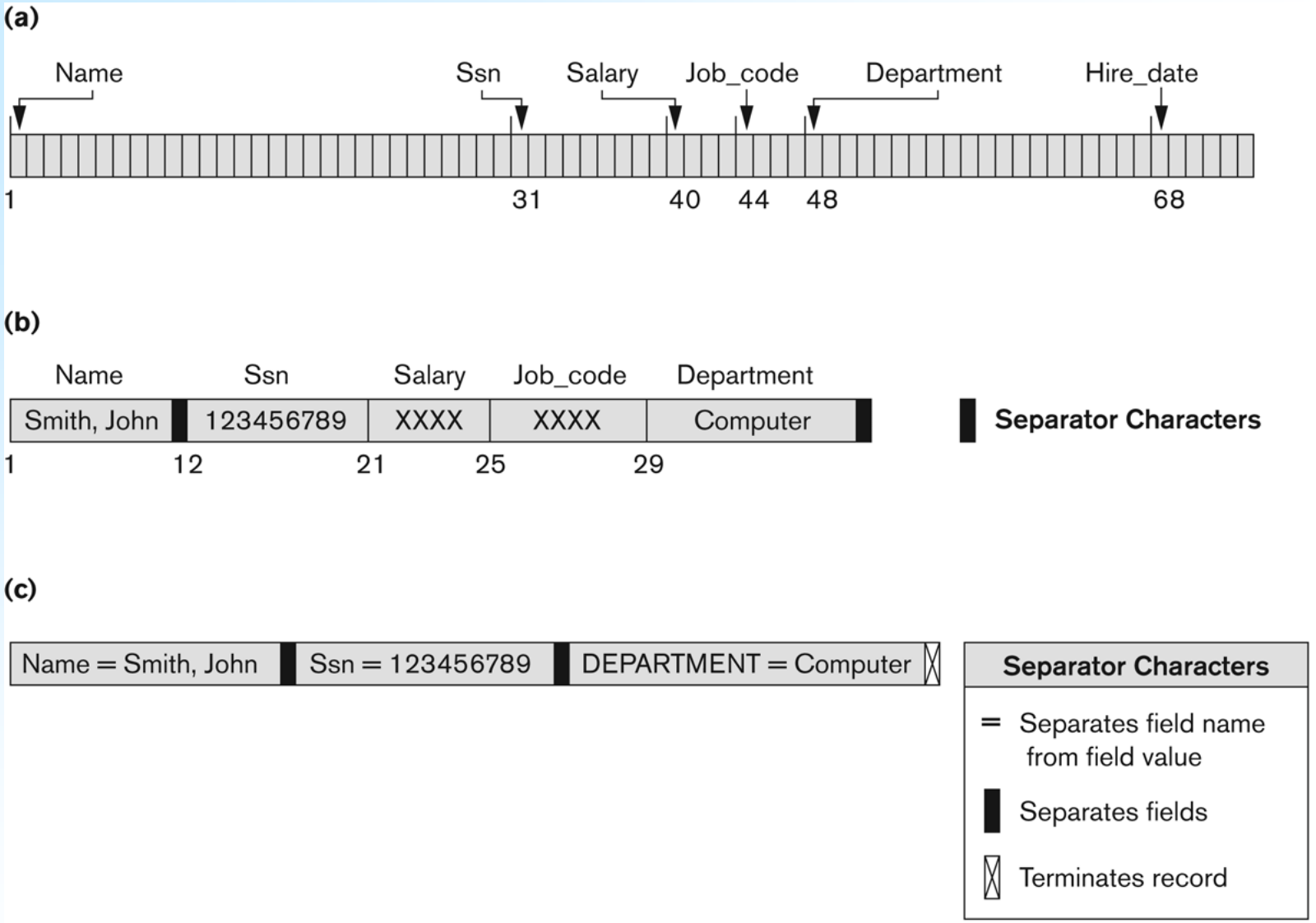


Figure 13.5

Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

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

Oracle???

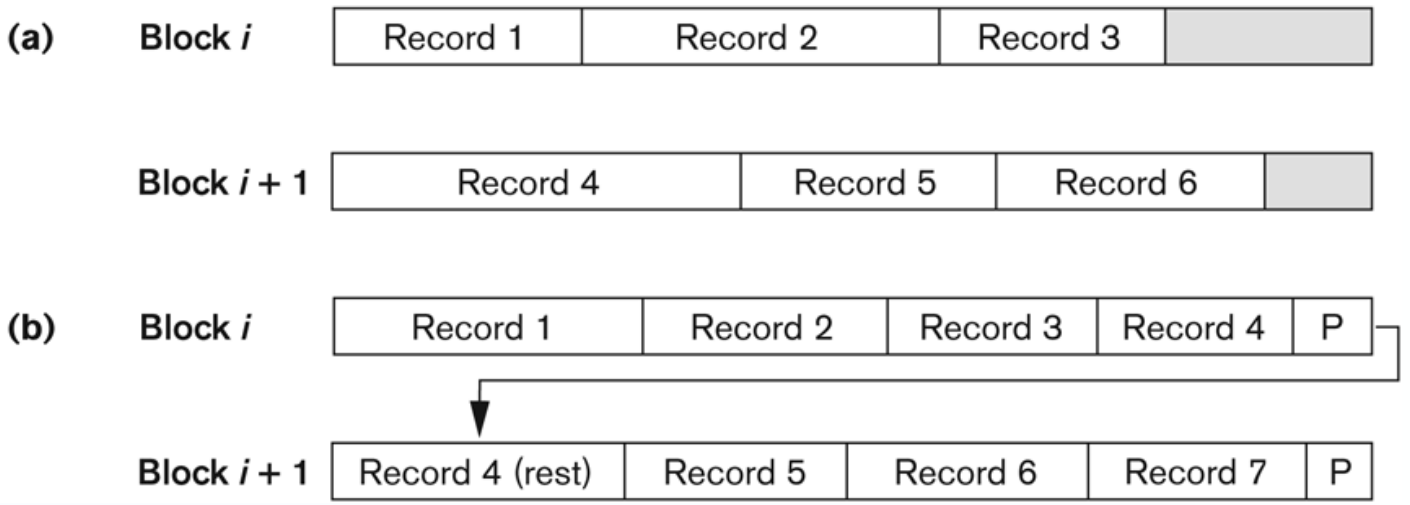
The **blocking factor *bfr*** for a file is the (average) number of file records stored in a disk block

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
  - *Oracle uses….???*

Figure 13.6
Types of record organization. (a) Unspanned. (b) Spanned.



Files of Records (cont.)

The physical disk blocks that are allocated to hold the records of a file can be *contiguous, linked, or indexed*.

Oracle uses … ???

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**

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* record at each point in time.

RESET: Makes the first record of the file the *current* record.

FIND: Searches for the first record that satisfies a certain condition, and makes it the current record.

READ: Reads the current record into a program variable.

FINDNEXT: Searches for the next record (from the current record) that satisfies a certain condition, and makes it the current record.

DELETE: Removes the current 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 record.

INSERT: Inserts a new record into the file & makes it the current record.

CLOSE: Terminates access to the file.

FIND_ORDERED: Read the file blocks in order of a specific field of the file.

REORGANIZE: Reorganizes the records.

For example, the records marked deleted are physically removed from the file or a new organization of the records is created.

Unordered Files (Heap)

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* file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.

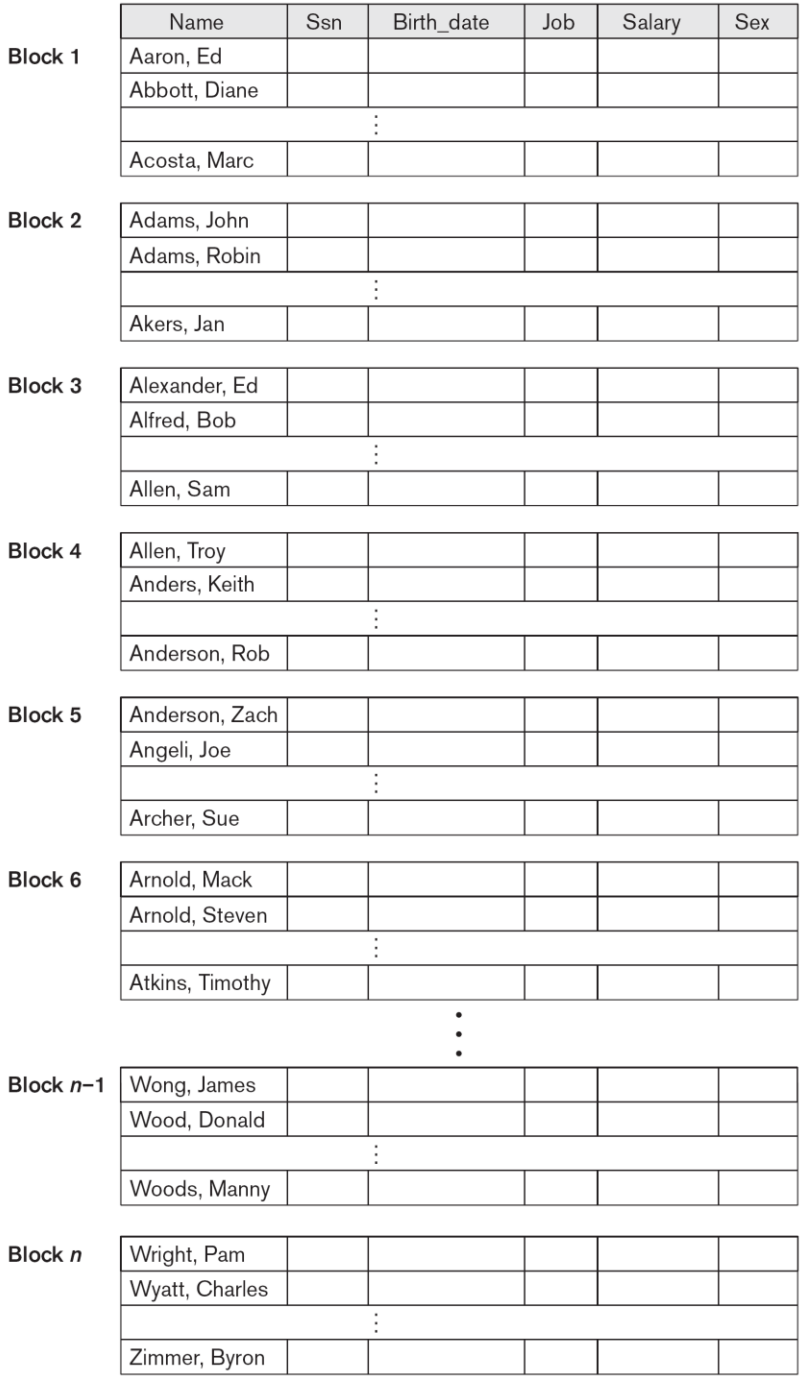
A **binary search** can be used to search for a record on its *ordering field* value.

- 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.)

Figure 16.7
Some blocks of an ordered (sequential) file of EMPLOYEE records with Name as the ordering key field.



Average Access Times

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

Table 16.2 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	$\log_2 b$

Hashing

One of the file fields is designated to be the **hash field (hash key** if the field is a key) .

The record with hash key value *K* is stored in address *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 an address that is already occupied.

Overflow handling

There are numerous methods for collision resolution, including the following:

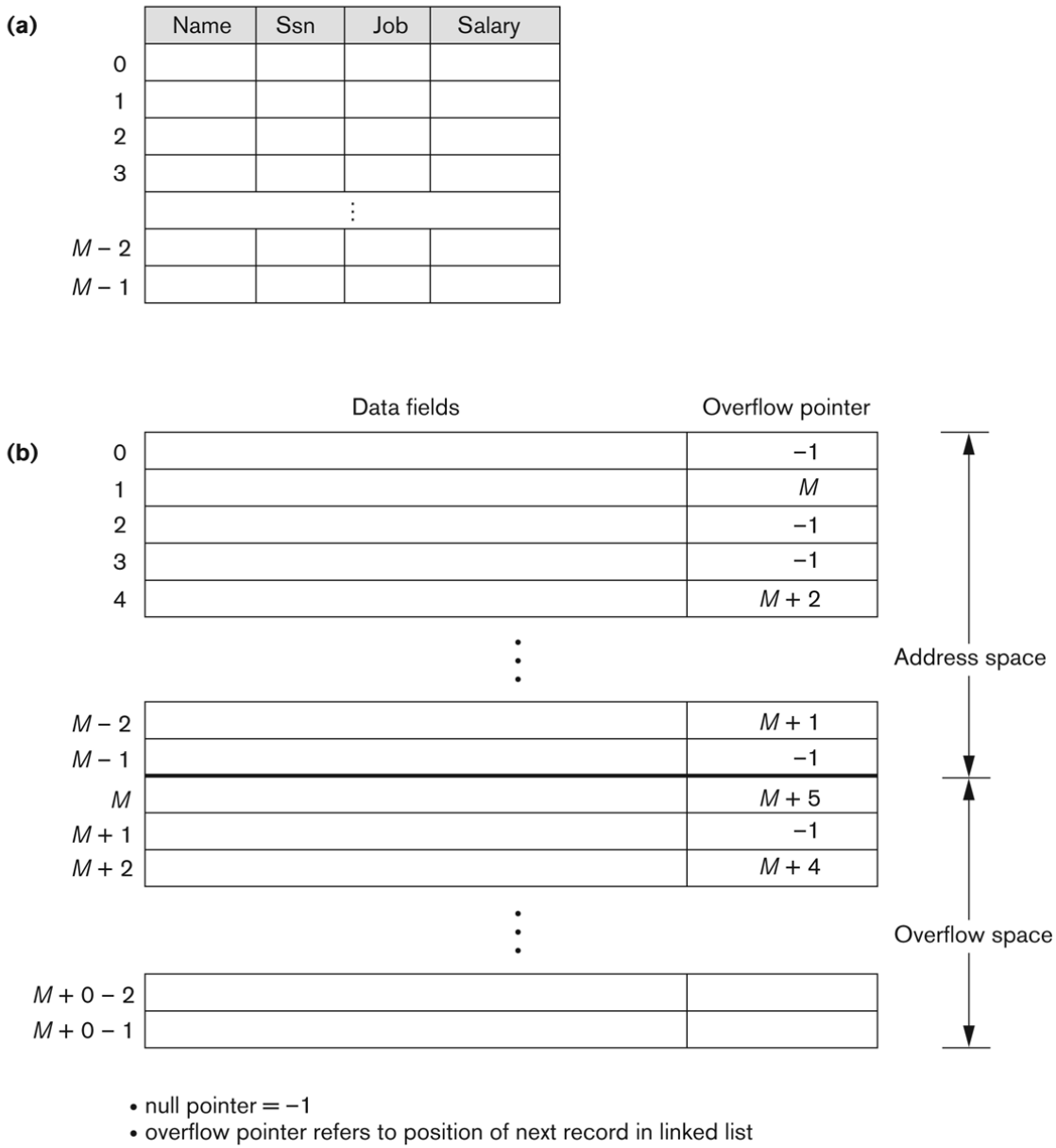
Open addressing: Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.

Chaining: For this method, various overflow locations are kept. In addition, 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.

Internal Hashing

Figure 13.8
Internal hashing data structures. (a) Array of M positions for use in internal hashing. (b) Collision resolution by chaining records.



- null pointer = –1
- overflow pointer refers to position of next record in linked list

External Hashing

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 field (hash key** if the field is a key) .

The record with hash key value *K* is stored in bucket *i*, where i=h(K), and *h* is the **hashing function**.

Collisions occur when a new record hashes to a bucket that is already full.

Overflow Handling (Hashed Files)

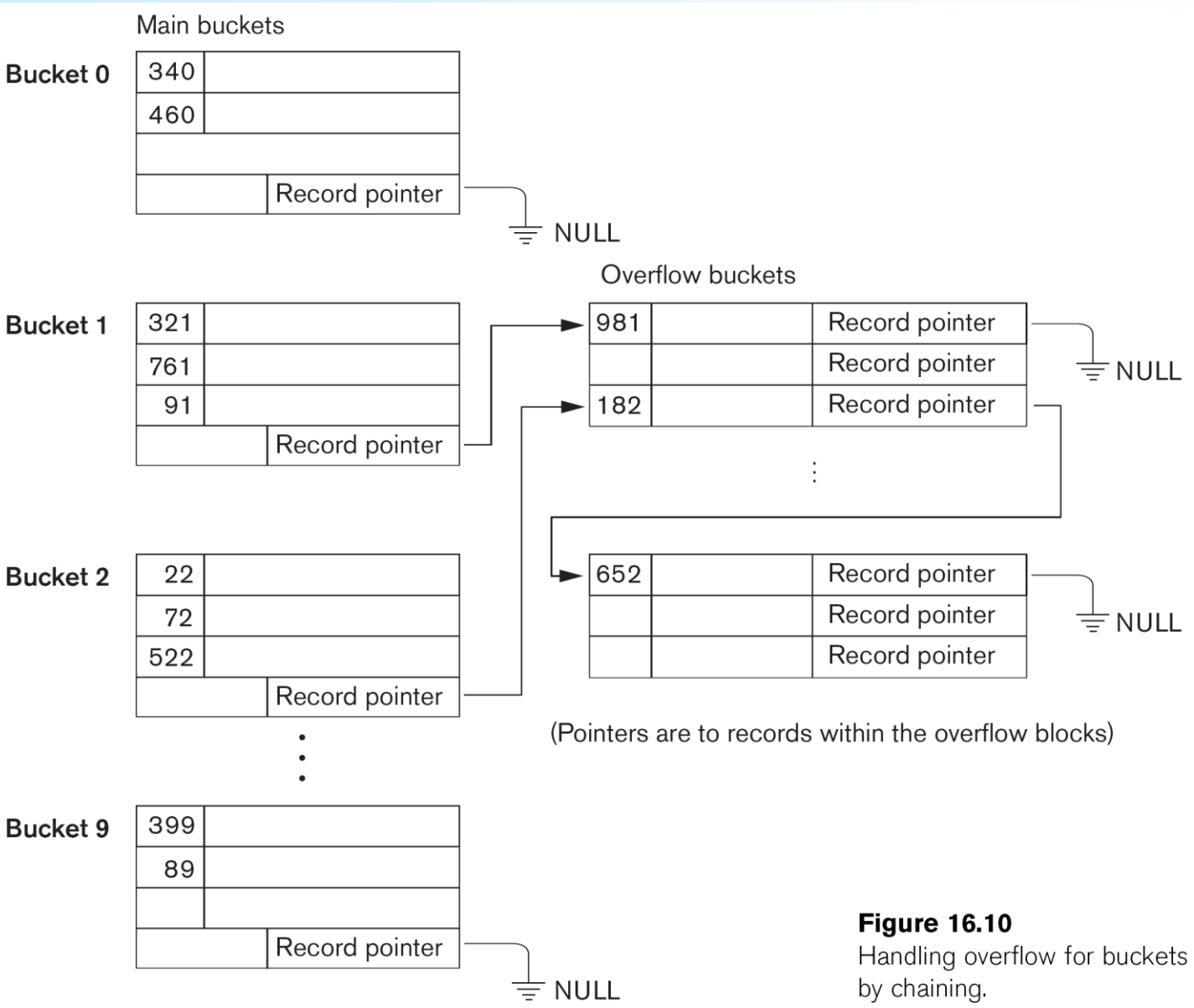


Figure 16.10
Handling overflow for buckets by chaining.

Hashed Files (cont.)

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 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).

Dynamic Hashed Files

Dynamic Hashing

- Hashing techniques adapted to allow the dynamic growth and shrinking of the number of file records.
- These techniques include the **extendible hashing** and **linear hashing.**

Extendible hashing uses the **binary representation** of the hash value h(K) in order to access a **directory**.

- The directory is an array of size 2^d where *d* is called the **global depth** (number of bits used to address)

Extendible Hashing

- The directories can be stored on disk, and they expand or shrink dynamically.
  - Directory entries point to the disk blocks that contain the stored records.
- An insertion in a bucket that is full causes the bucket to split into two buckets and the records are redistributed among the two buckets.
  - The directory is updated appropriately.
- Extendible hashing does not require an overflow area.

Extendible Hashing

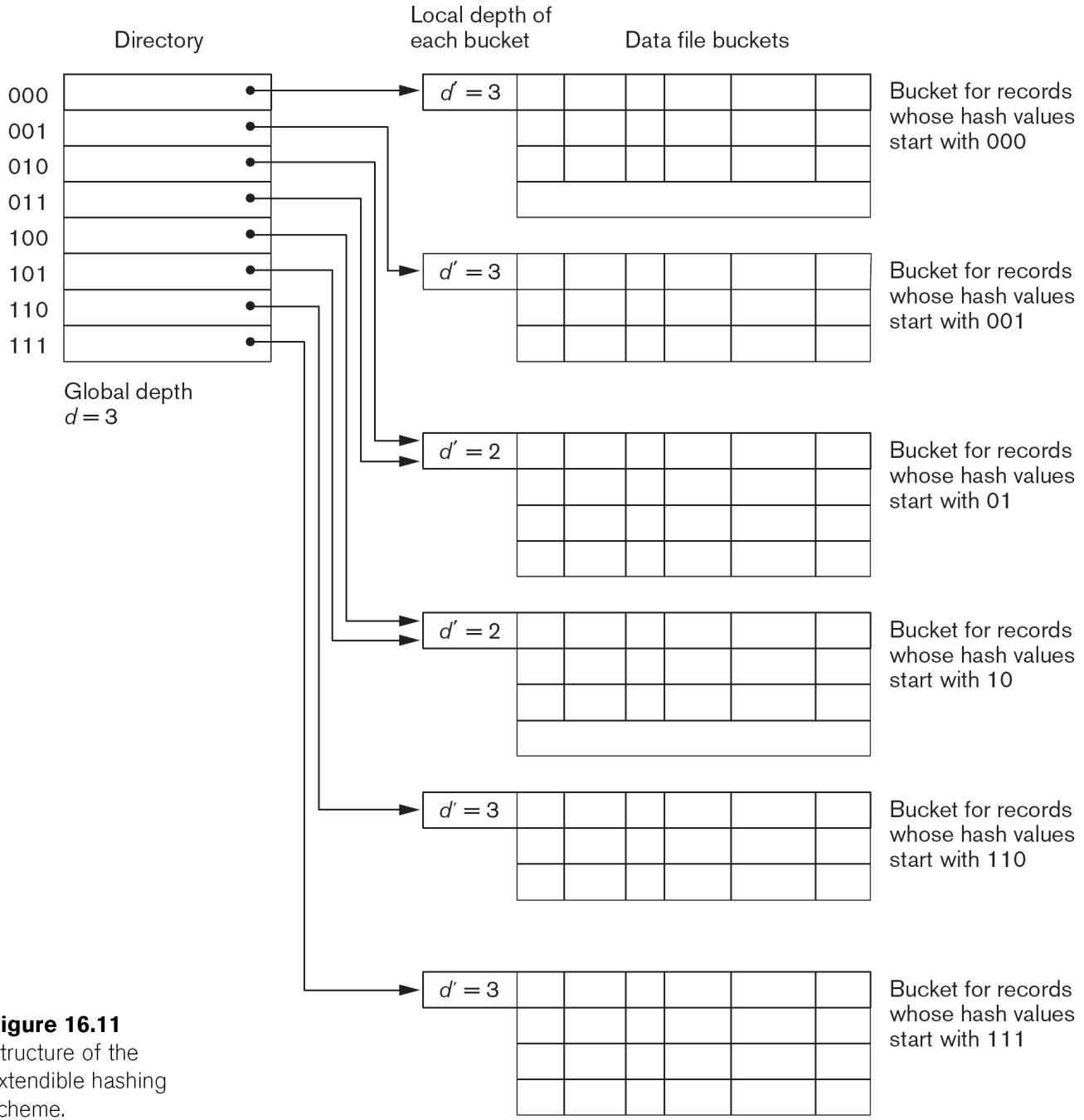


Figure 16.11
Structure of the extendible hashing scheme.