Database Design & Applications (SS ZG 518)

Dr.R.Gururaj
CS&IS Dept.

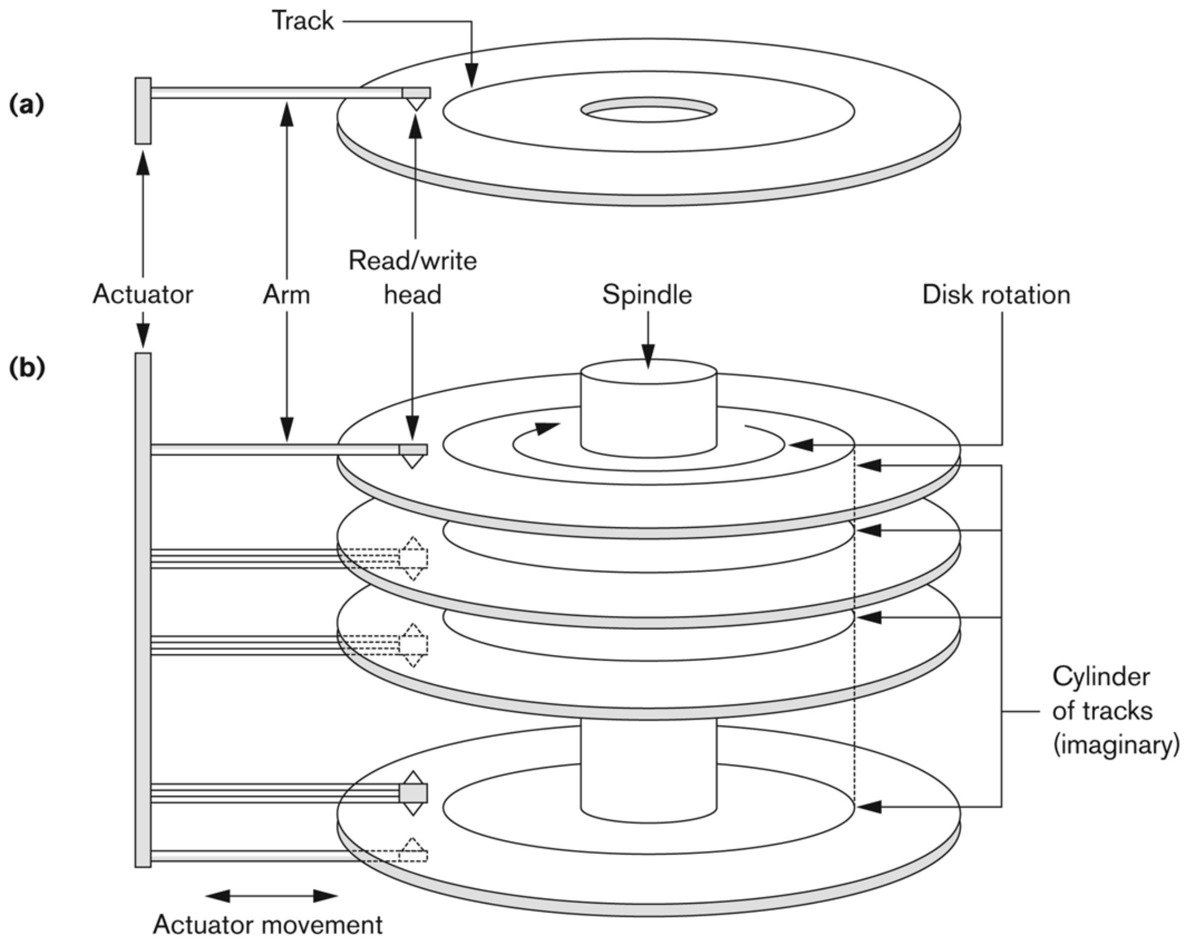
BITS Pilani
Hyderabad Campus

Disk Storage and Hashing (Ch.13)

Content

- ❑*Disk pack features*
- ❑*Records and Files*
- ❑*File operations*
- ❑*Ordered and unordered features*
- ❑ Introduction to Hashing
- ❑Internal hashing
- ❑Collision
- ❑External hashing
- ❑Static hashing
- ❑Dynamic hashing (Extendible and Linear hashing)

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



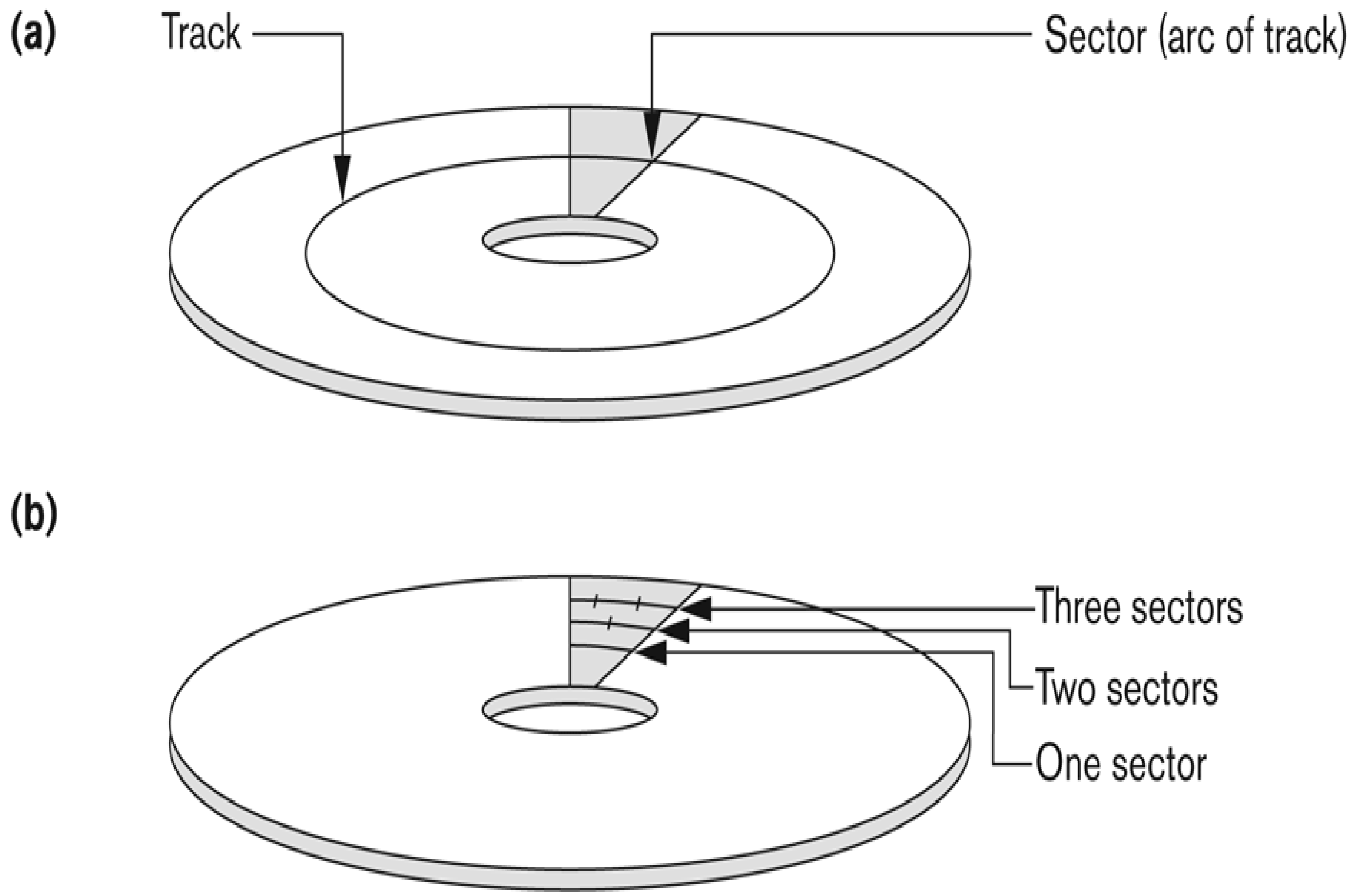


Figure 13.2
Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.

- *Seek time* (time to position the head on required track)

  3-7msec and

- *Rotational delay* (*latency*) – time to position at the beginning of the required block **rd**. 3-4 msec with 15000rpm

- And Block transfer time Block transfer time. Smaller than above two.

Problem-1

(Storage Organization)

Following are the specifications of an external storage Diskpack.

Block size is 1024 Bytes. The number of blocks per track are 16; and there are 100 tracks on each surface. Further the disk pack has 12 double-sided disks.

Now answer the below questions.

What is the total capacity of a track.

How many cylinders are there?

What is the total capacity of each cylinder?

What is the total capacity of the Diskpack?

Files and Records

- A **file** is a *sequence* of records, where each record is a collection of data values (or data items).
- 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.

Record Organization

- File records can be unspanned or spanned
- File operations

File Organization

- Unordered Files (heap)
- Ordered Files (sequential)

Hashing

Hashing technique is an alternative to indexing, for fast retrieval of data records based on search key.

The search field is called as *hash field* of the file.

In most cases the hash field is also a key field of the file, in which case it is called as *hash key*.

The basic idea of hashing is that a hash function *h*, when supplied a hash field value *K* of a record produces the address *B* of the disk block that contains the record with specified key value.

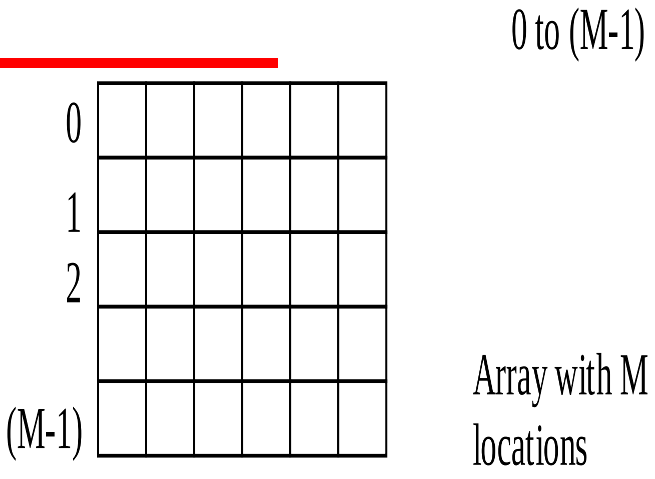
$$\underbrace{\boldsymbol{h}}_{\text{Hash function}} \;:\; \underbrace{\boldsymbol{K}}_{\text{Key value}} \rightarrow \underbrace{\mathbf{B}}_{\text{Disk block address}}$$

Once the disk block is known, the actual search for the record within the block is carried out in main memory buffer.

For most records we require only one block access.

Internal Hashing

Used for internal files.
A hash table is implemented through use of an array of records.



The most common hash function used is *h*(*k*) = K mod M
This gives the index of the location in the array.
For example- if M = 10 key value is 24
K mod M
$\Rightarrow$ 24 mod 10 = 4
Hence the record with key value 24 will be stored in 5th location of the array. If two or more records are hashed to same location it is called as *collision*. Then we need to find some other location for the new record. This process is known as *collision resolution*.

There are two methods for collision resolution

Open addressing: When collision occurs try with alternate cells until an empty cell is formed.
Chaining: for this various overflow locations are kept by extending the array by number of overflow positions. A pointer field is added to each record location. Collision is resolved by allocating an unused overflow position.
Multiple hashing: We apply a second hash function if the first hashing results in a collision.

The goal of a good hashing function is to distribute the records uniformly over the address space so as to minimize collisions while not leaving many unused locations.

External Hashing

Hashing used for disk files is called as *external hashing*. The disk block contains records. A single disk block or cluster of contiguous blocks is known as a *bucket*.
The hashing function maps a key value into a relative bucket number. A table maintained in the file header converts the bucket number into the corresponding disk block address, as shown in the figure below.

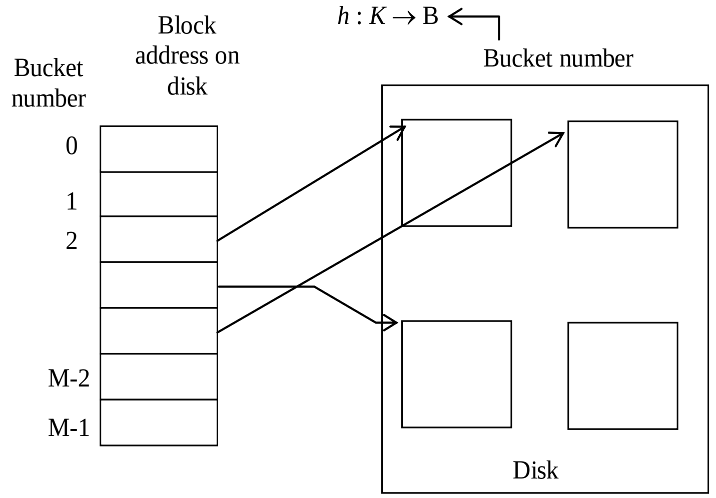
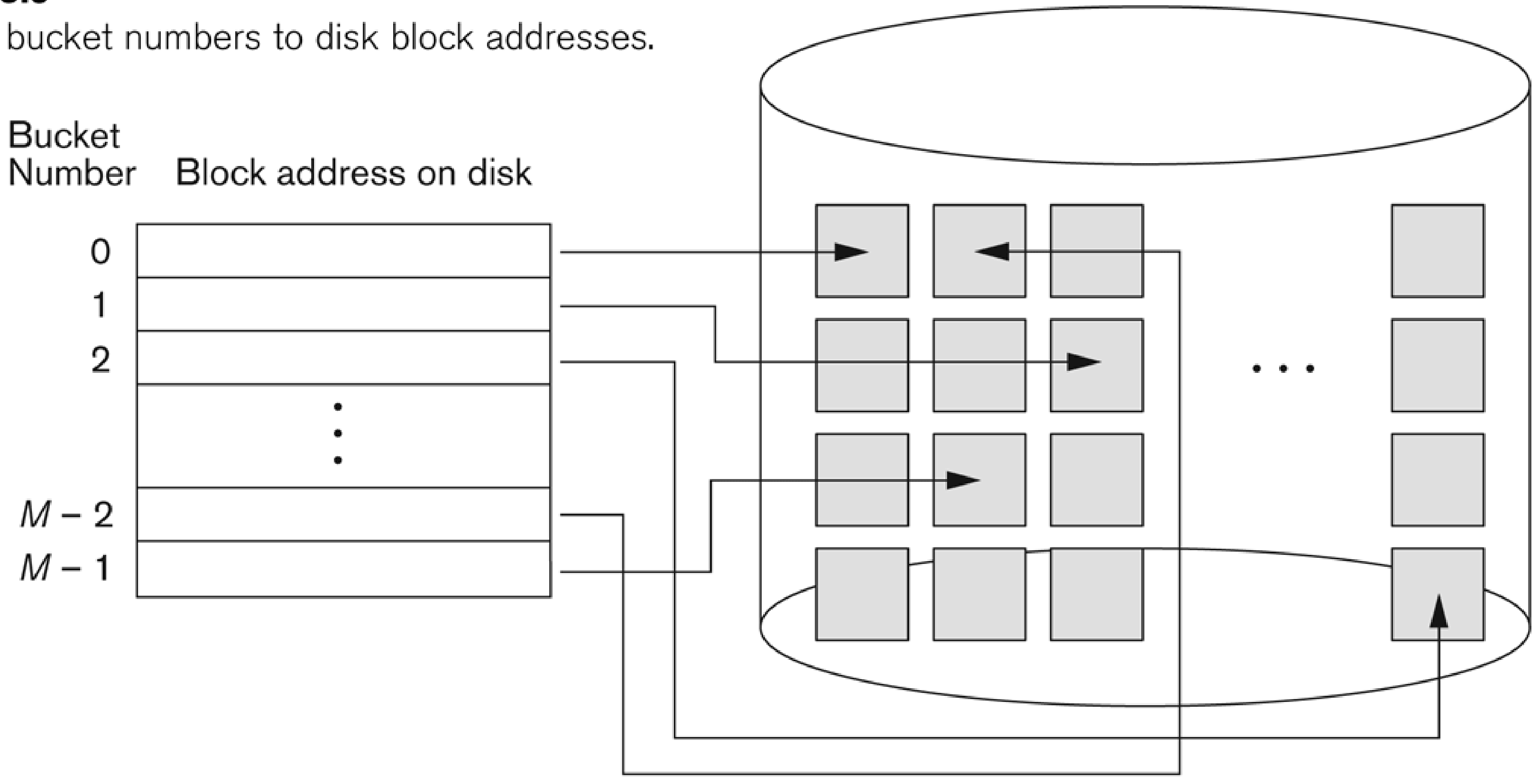


Figure 13.9
Matching bucket numbers to disk block addresses.



The above scheme is called as *static hashing* because the number of buckets allocated is fixed. This is a big constraint for files that are dynamic.

When a bucket is filled to capacity and if the new record is hashed on to the same bucket, then chaining is adopted, where a pointer is maintained in each bucket to a linked list of overflow records for the bucket.

The pointers are record pointers which include both block address and a relative record position with in that block.

Handling overflows in Static External Hashing

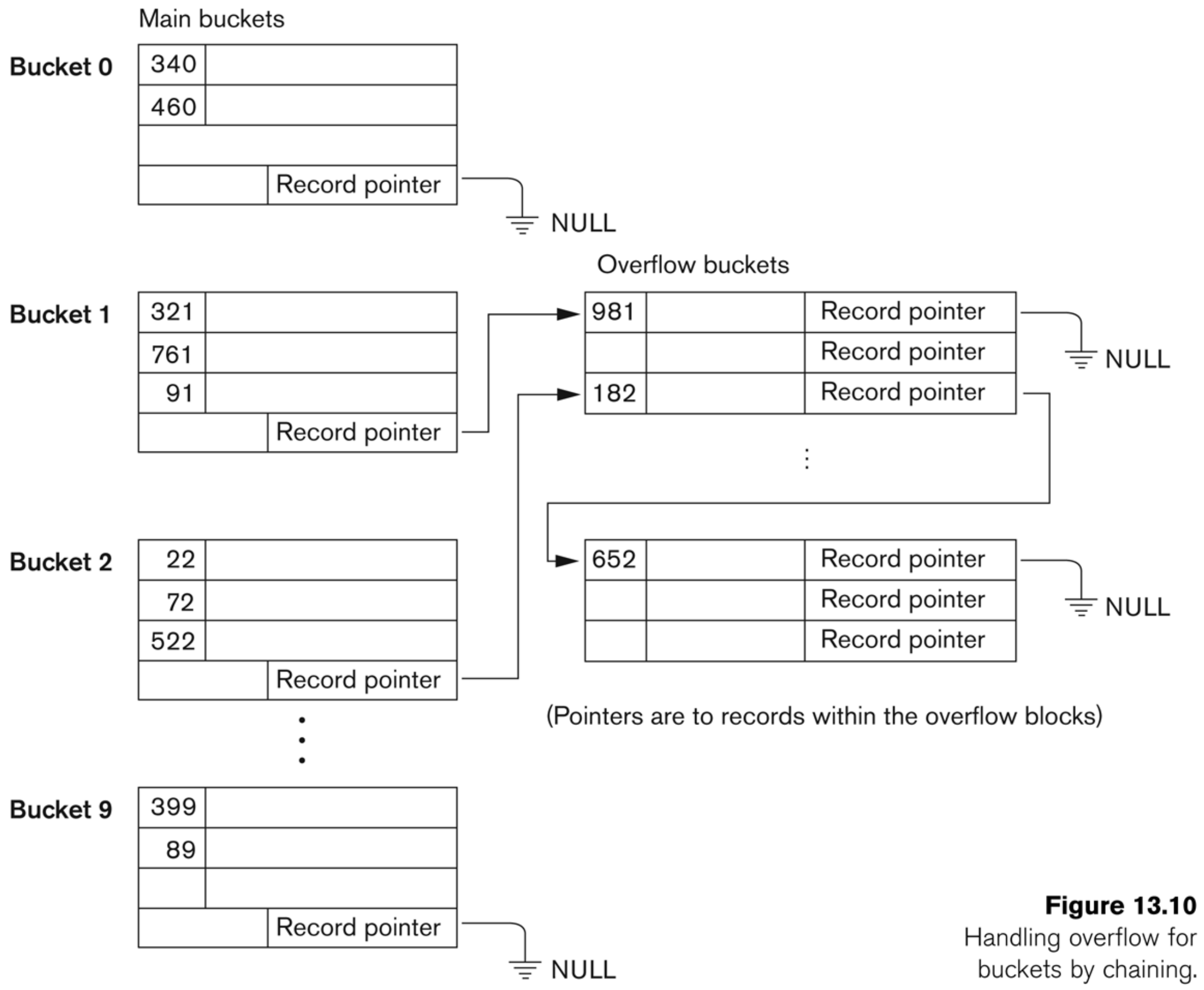


Figure 13.10
Handling overflow for buckets by chaining.

Dynamic Hashing

This scheme allows us to expand or shrink the hash address space dynamically.

Each result of applying the hash function is a nonnegative integer and hence can be represented with a binary pattern. This we call it as *hash value* of the record.

Records are distributed among the buckets based on the values of the leading bits in their hash value.

Extendible Hashing

The first technique is called as *extendible hashing*. This scheme stores a directory structure in addition to the file. This access structure is based on the result of the hash function to the search field. The major advantage of extendible hashing is that performance does not degrade because of chaining, as the file grows as we have seen in static hashing. In extendible hashing no additional space is wasted towards the allocations for future growth, but additional buckets can be allocated dynamically as needed. The only overhead in this scheme is that a directory structure needs to be searched before the buckets are accessed.

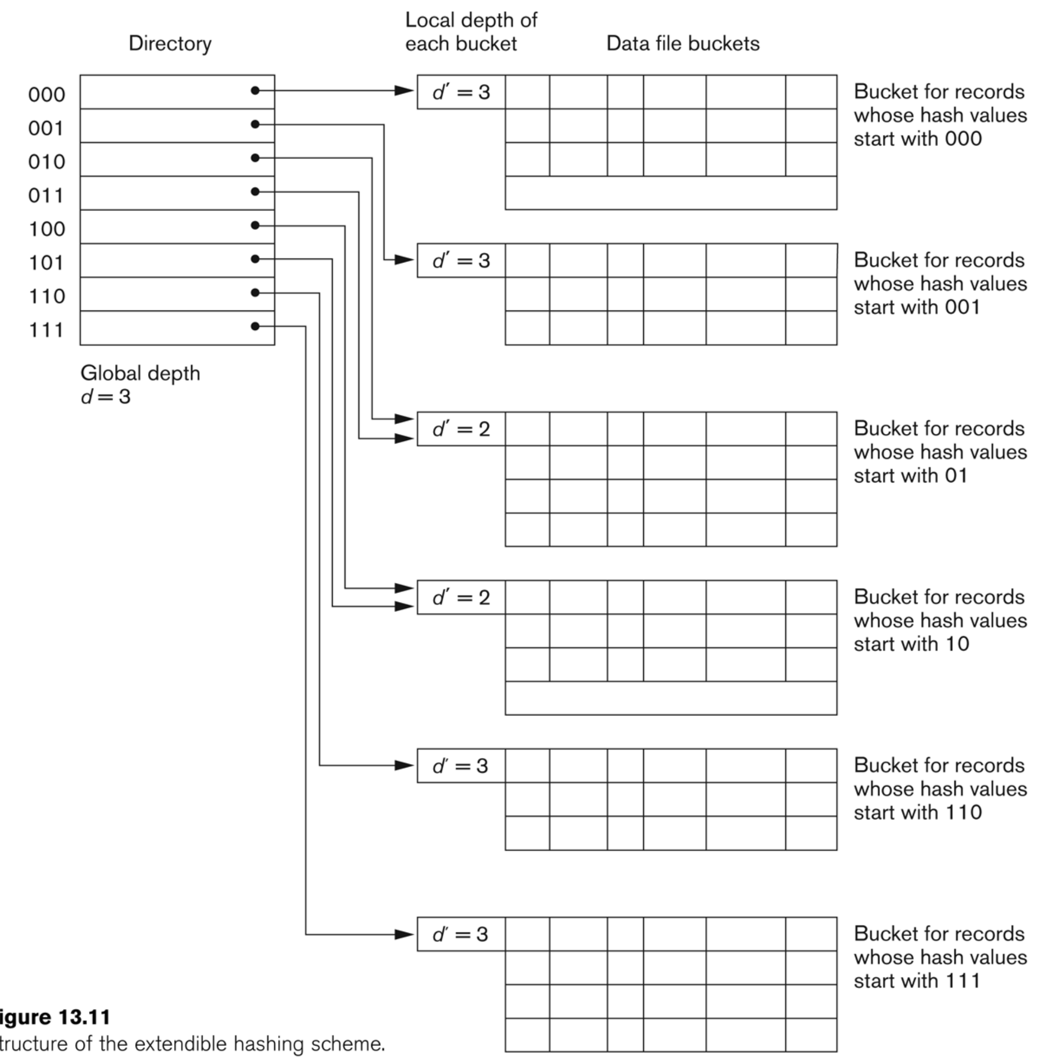

Directory
Local depth of each bucket
Data file buckets
000
001
010
011
100
101
110
111
Global depth $d = 3$
$d' = 3$
$d' = 3$
$d' = 2$
$d' = 2$
$d' = 3$
$d' = 3$
Bucket for records whose hash values start with 000
Bucket for records whose hash values start with 001
Bucket for records whose hash values start with 01
Bucket for records whose hash values start with 10
Bucket for records whose hash values start with 110
Bucket for records whose hash values start with 111

Figure 13.11
Structure of the extendible hashing scheme.

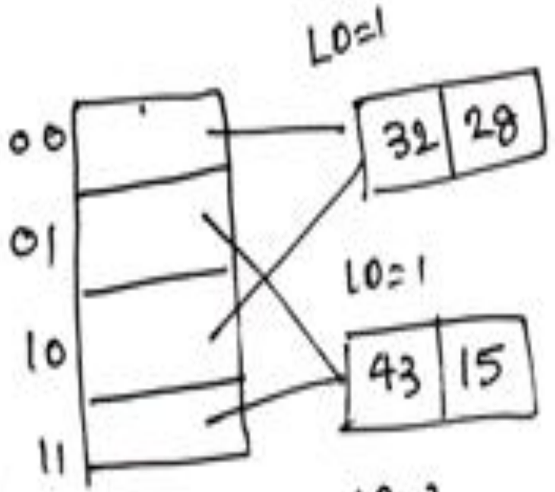
Assume that we need to load some records of the relation *EMP* into *expandable hashfiles* based on *extendible hashing technique*. Records are inserted into the file with following key field values: 32, 28, 43, 15, 66, 27, 86, 54, 35, 98, 72. The order of insertion should be same as the above. Each bucket is one disk block and each block can store maximum of three records. Show the structure of the directory at each step, and the local and global depths. Use the hash function *h(K)=K mod 10.* Start with Global depth-2, Local depth-1. Consider LSB of the hash value for directory entries.

Q4: Key value

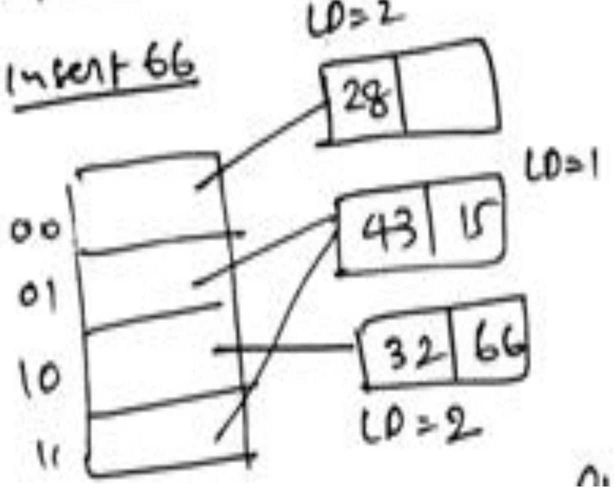
(a) Key values: 32, 28, 43, 15, 66, 27, 86, 57, 35, 98, 72.

		Binary
32%10 =	2	0010
28%10 =	8	1000
43%10 =	3	0011
15%10 =	5	0101
66%10 =	6	0110
27%10 =	7	0111
86%10 =	6	0110
57%10 =	7	0111
35%10 =	5	0101
98%10 =	8	1000
72%10 =	2	0010

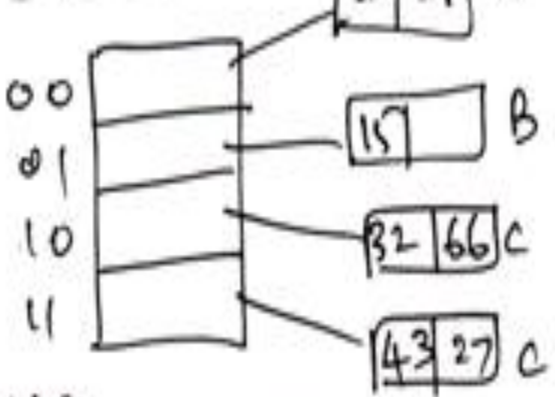
Global depth = 2



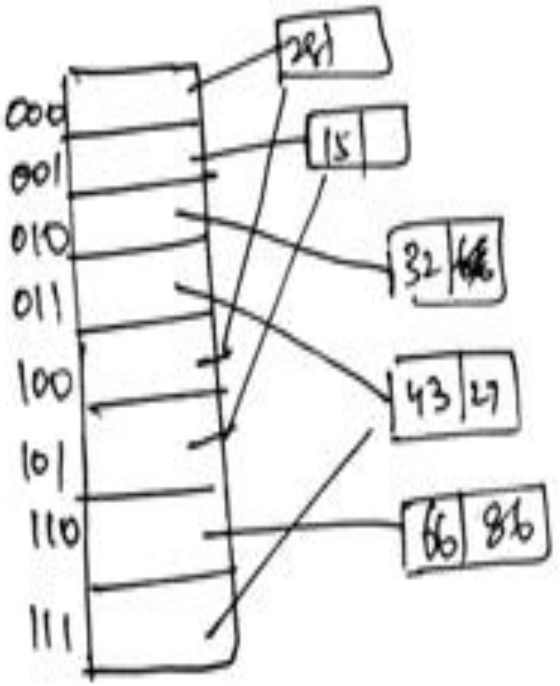
Insert 66



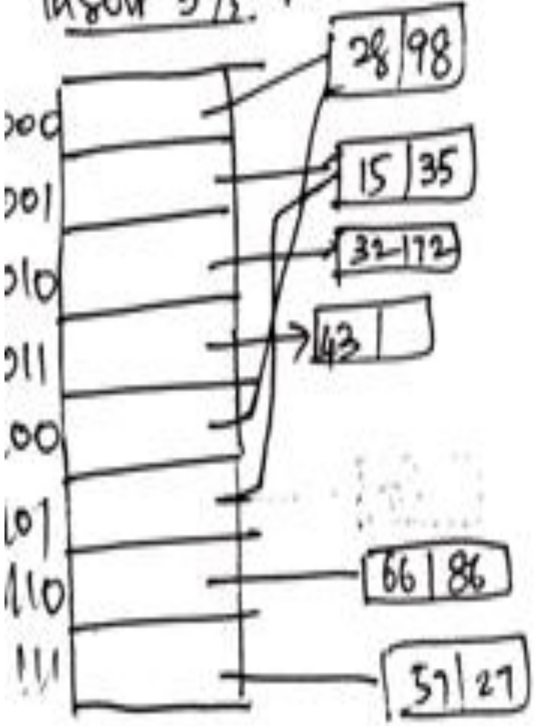
Insert 27



When insert 86 increase Global depth



After insert 57, 35, 98



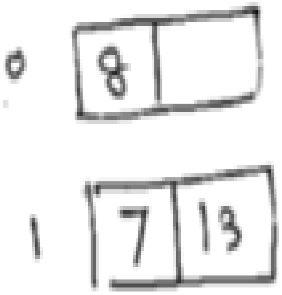
Linear Hashing

In the second scheme called *linear hashing*, no directory structure is used. Here instead of one hash function, multiple hash functions are used. When collision occurs with one hash function, the bucket that overflows is split in to two and the records in the original bucket are distributed among two buckets using the next hash function $h_{(i+1)}(k)$. Hence we have multiple hash functions.

Q5: ⓑ	8	7	13	5	22	11	42	24	16	31
k mod 2	0	1	1	1	0	1	0	1	0	1
k mod 4	0	3	1	1	2	3	2	1	0	3
k mod 8	0	7	5	5	6	3	2	5	0	7

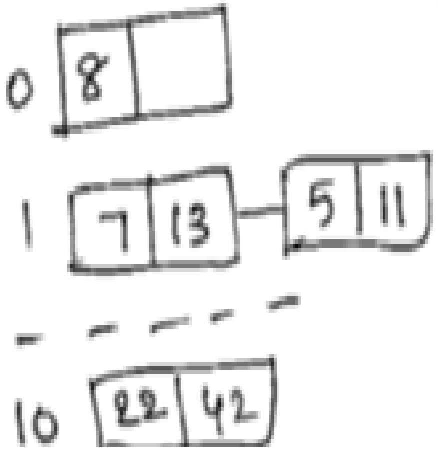
n = 0 use k mod 2

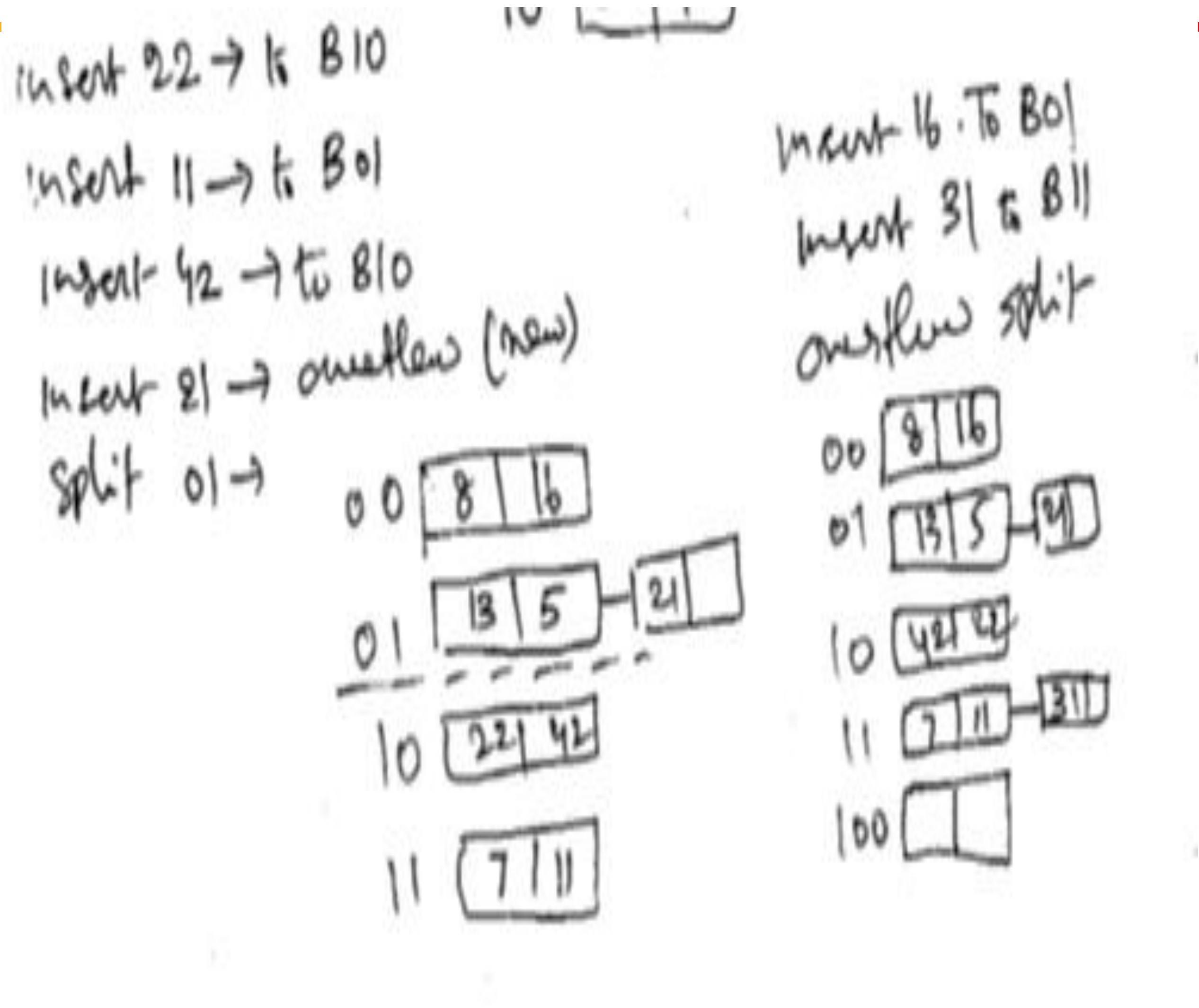
insert 8, 7, 13



When inserting 5
Collision.
new overflow is added
to Bucket-1
Now split occurs for B=0

n=1





insert 22 → to B10

insert 11 → to B01

insert 42 → to B10

insert 21 → overflow (new)

split 01 →

insert 16 . To B01

insert 31 to B11

overflow split

Summary

- ✓ *What is Disk storage*
- ✓ *Disk characteristics*
- ✓ *Disk pack structure*
- ✓ *Files and Records*
- ✓ *Ordered and unordered files*
- ✓ What is hashing
- ✓ Internal hashing
- ✓ External hashing
- ✓ What is static external hashing
- ✓ What is dynamic hashing
- ✓ How Extendible and Linear hashing techniques work