CS430: Database Systems

Dr. Chetan Jaiswal,
Department of Computer Science,
Truman State University
cjaiswal@truman.edu

Chapter 16: File Organizations

- Introduction to Files
- Types of files and file management
- Disk organization and file storage

File Organization

- **Logical structure of a file**
  - **Field: Smallest (fixed) indivisible logical unit of a file. A field holds a part of some data value**
  - **Record: A set of logically related fields. Cardinality of this set may be fixed or variable, i.e., a record size may be fixed or variable. A file, therefore, is a collection of logically related records**

A Field

A Record

SSN	Name	Age	Phone #

A File

SSN	Name	Age	Phone #

File Organization

- Logical structure of a file
  - A record can be of fixed or variable size. Most commonly fixed size records are used. The file structure has been illustrated here in a tabular form but note that a file is not just a table its records can be organized in many different ways each way defines a unique organization

File Organization

- Operations on a file

Create:	Create a new file
Write:	Write to a file
Read:	Read a part of a file
Rewind:	Go to the beginning of the file
Delete:	Remove a file from the system
Close:	Close a file (at the end of manipulation)
Open:	Open a file for processing
Modify:	Modify a field value.

File Organization

- **Relationship of a file with the storage media**
  - **Common storage media are disk, optical disk, and tape. Disk is a random storage device because a record of a file can be accessed from the place it is stored. In some cases a file acquires the attributes of the storage media on which it is stored, i.e., magnetic tape. To access a record of a file stored on a magnetic tape, unlike disk files, the entire file has to search sequentially before reaching the record**

File Organization

- **Relationship of a file with the storage media**
  - **Disk File structure: Several platters, several cylinders, several tracks on a cylinder, several sectors on a track (Sector = Block (fixed by manufacturer)).**
  - **Unit of data transfer between memory and disk: *Block*.**

File Organization

- **Relationship between logical and physical units**
  - **A fixed record size may be larger or equal to or smaller than a disk block. For storage purpose a unit called Blocking factor (Bfr) is defined:**
    - **Blocking factor - Bfr (Number of records in a disk block) = $\lfloor B/R \rfloor$ where B is disk block size and R is file record size. Thus, unused space in a block: *B - (Bfr * R)* bytes**

File Organization

- **Relationship between logical and physical units**
  - **Disk parameters: *Seek time*, *rotational latency time* and *block transfer time***
    - **Seek time: The time it takes to move the read/write arm to the correct cylinder. Average seek time is the same as the time it takes to traverse one third of the cylinders**
    - **Rotational latency time: The time the disk unit takes to move or to position the read/write head on to the beginning of the sector where the file records are stored**
    - **Block transfer time: Time to transfer a block of data from disk to RAM**

File Organization

- **Rotational latency and Block transfer time**
  - **Average rotational latency (*r*) = 1/2 of one disc revolution = 0.5 (*rpm* = No. of rotations per minute)**
  - **Time for 1 revolution =** $\frac{1}{rpm}\min$ **=** $\frac{60 \times 1000}{rpm} ms.$
  - **Time for half revolution =** $\frac{0.5 \times 60 \times 1000}{rpm} ms.$
  - **Common disk rotation speed = 3600 rpm**
  - **So the average latency time =** $\frac{0.5 \times 60 \times 1000}{3600} ms = 8.3 ms$

File Organization

Rotational latency and Block transfer time

- **Suppose *t'* = data transfer speed for formatted data (block and the gap). So the effective block transfer time *Ebt* = *B/t'*. We use IBM 3380 disk in our example to compute various parameter values. The total capacity of the disk is over 2 billion bytes**

Parameter	Definition	Value
B	Block size	2400 bytes
t	Data transfer speed	3000 bytes/ms
btt	Block transfer time	0.8ms
t'	Formatted data (interblock gap) speed	2857 bytes/ms
ebt	Effective block transfer time	0.84 ms (= B/t')
N	No. of cylinders	885 per spindle
r	Average rotational latency	8.3 ms
s	Average seek time	16 ms

File Organization

- **Data read time**
  - **Time to read 10 blocks sequentially**
    - **It is the sum of average seek time, average rotational latency time, and 10 effective transfer time. Effective block transfer time must be used, since the head must move over the interblock gap. So reading time = $s + r + 10 \times ebt = (16 + 8.3 + 8.4) = 32.7$ *ms*. Note that the *seek* time and the *rotational latency* time are significant with respect to the transfer time.**

File Organization

- **Data read time**
  - **Time to read 10 blocks randomly**
    - **In this case individual *seek* time and *rotational latency* time are required for each block. So reading time = $10 \times (s + r + btt) = 10 \times (16 + 8.3 + 0.84) = 10 \times 25.14 = 251$ *ms*. The random time for reading 10 blocks is higher than the sequential time by a factor of 8. Thus, time for reading 100 sequential blocks = 0.1083 sec. Time for reading 100 random blocks = 2.510 sec. The time for reading randomly 100 blocks is 25 times larger than time for reading 100 blocks sequentially**

File Organization

- **File organization types**
  - No single file organization is the most efficient for all types of data access. For this reason a database management system uses a number of different file organizations for storing the same set of data
  - There are basically three types of file organizations (a) Sequential, (b) Index, and (c) Random organization

File Organization

- **Sequential File**
  - **Records are written consecutively when the file is created. Records in a sequential file can be stored in two ways:**
    - **Pile file: Records are placed one after another as they arrive (no sorting of any kind)**
    - **Sorted file: Records are placed in ascending or descending values of the primary key**

File Organization

- Sequential File
  - Pile file: The total time to fetch (read) a record from a pile file requires *seek* time (*s*), *rotational latency* time (*r*), Block transfer time (*btt*), and number of blocks in the file (*B*). We also need a key field of the record (*K*). In pile file we do not know the address of a record, so we search sequentially. The possibilities are (a) the desired record may be in the first block or (*b*) it may be in some middle block or (c) it may be in the last block

File Organization

- **Sequential File**
  - **Advantages**
    - Good for batch transactions.
    - Simple to implement
    - Good for report generation, statistical computation and inventory control.
  - **Disadvantages**
    - Not good for interactive transactions
    - High overheads in file processing for simple queries

File Organization

- **Index File Organization**
  - **Index organization tries to reduce access time. It may not reduce the storage requirement of a file**
  - **Index: An index is similar to a pointer with an additional property that allows an index to identify the record it is pointing to. For example, an index to a record of employees points and identifies an employee record. This means that an index has some semantics associated with it**
  - **An index maps the *key space* to the *record space*. Index for a file may be created on the *primary* or *secondary* keys**

File Organization

- **Index File Organization**
  - Indexed file reduces record search time
  - Example
  - Sequential search
    - File size = 30,000 records. Record size = 100 bytes (R)
    - Block size = 1024 bytes (B)
    - Blocking factor = $\lfloor 1024/100 \rfloor$ =10 records per block
    - No. of blocks for the data file = 30,000/10 = 3000 blocks
    - A binary search on the data file $\lfloor \log_2 3000 \rfloor$ = 12 blocks accesses
  - Indexing
    - Index record size = 15 bytes: Primary key = 9 bytes + Block pointer = 6 bytes
    - Index file blocking factor = $\lfloor 1024/15 \rfloor$ = 68 entries. No. of blocks for index file = 3000/68 = 45
    - Binary search on index file = $\lfloor \log_2 45 \rfloor$ = 6 blocks accesses. Total no. of accesses = 6 + 1 = 7

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **This file organization closely related to the physical characteristics of the storage media. It has two parts**
    - **Index Part: Stores pointers to the actual record location on the disk. It may have several levels and each level represents some physical layout such as sector, cylinder, of the storage media.**
    - **Data Part: Holds actual data records and is made up of two distinct areas: Prime area and Overflow area. The prime area holds records of the file. The overflow area holds records when the prime area overflows.**

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Organization: Data records are stored on disk tracks in their ascending key values**

Track No.	Key	Record	Key	Record	Key	Record	Key	Record
Track 1	10	rec10	14	rec14	18	rec18	20	rec20
Track 2	26	rec26	34	rec34	41	rec41	60	rec60
Track 3	72	rec72	73	rec73	74	rec74	75	rec75
Track 4	77	rec77	78	rec78	79	rec79	82	rec82
Track 5	89	rec89	91	rec91	93	rec93	98	rec98

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - Organization: To locate a data record a Track Index is required
  - Track Index

Track No.	Highest key on the track
1	20
2	60
3	75
4	82
5	98

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Operation on file: Read record 79**
    - **Sequential access: Search sequentially from track 1, record 10. Very slow.**
    - **Semi-random access: (a) Search track index, (b) find the track that contains a key greater than 79. This takes us to track 4. Search track 4 sequentially to find record 79.**
    - **Problem: For a large file track index would be large. A large index is not convenient to manage.**
    - **Solution: Create Cylinder index to manage track index**

File Organization

- Index Sequential File (ISAM - Index Sequential Access Method)
  - Cylinder index

Cylinder index

Cylinder	Highest key
1	82
2	163
3	259

Track index for cylinder 1

Track	Highest key
1	20
2	60
3	75
4	82

Track index for cylinder 2

Track	Highest key
1	107
2	122
3	148
4	163

Track index for cylinder 3

Track	Highest key
1	201
2	210
3	223
4	259

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Operation on file: Find record with key value 248**
  - **Steps**
    - **Search Cylinder index to find correct cylinder → Cylinder 3**
    - **Search track index for cylinder 3 → Track 4**
    - **Search track 4 (sequentially) to find record 248**

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Problem: Cylinder index may become very large**
  - **Example**
    - File size = 5000 tracks of data. Disk: 3350 (IBM)
    - 16 platters/volume = 30 surfaces/volume
    - 555 tracks/surface = 555 cylinders/volume
    - A cylinder has 30 tracks, the entire file will need 5000/30 =167 cylinders. Apart from the file, there are other controlling information such as track, cylinder, and master indexes, therefore, only 25 tracks are used for data and the rest are for control information
    - **Result**: 5000/25 = 200 cylinders for indexes and data, which means there are 200 entries in a cylinder index
    - Each entry of index record may be large. The size of cylinder index might grow with the file size. To reduce the search time for a cylinder index, a master index is created, which holds the address of cylinders storing the records

File Organization

- Index Sequential File (ISAM - Index Sequential Access Method)
  - Overflow area: It accommodates overflow record from the prime area. If the record key indicates that the record must go on a track, which is full then some records from this track is moved to the overflow area. With overflow area the track index takes the following form:

Track No.	Highest key on track	Highest key in overflow area	Addr. of 1st overflow
1	20	20	Null
2	60	60	Null
3	75	75	Null
4	82	82	Null
5	98	98	Null

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **File Processing**
    - **Deletion: Simple. Find the desired record. Put a special marker into the record to indicate that it has been deleted.**
    - **Insertion: Cumbersome.**
    - **Example: Insert record 55.**
    - **Destination: Track 2. Track 2 is full. Move record 60 to the overflow area. Modify the track index.**

10	14	18	20
26	34	41	**55**
72	73	74	75
77	78	79	82
89	91	93	98

1	20	20	Null
2	55	60	Address of Rec. **60**
3	75	75	Null
4	82	82	Null

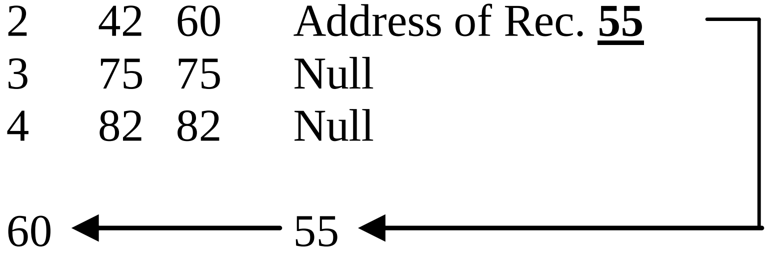
60

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **File Processing; Add record 42**
    - **Result Prime Area Track Index. Other records can be inserted similarly. When the overflow area is full then records are inserted in overflow area on the disk,**

10	14	18	20
26	34	41	**42**
72	73	74	75
77	78	79	82
89	91	93	98

1	20	20	Null
2	42	60	Address of Rec. **55**
3	75	75	Null
4	82	82	Null



File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Implementation**
    - **The cylinder index is usually kept in the memory. The cylinder index gives the address of the correct track index, which is located on the disk. The track index is fetched and the address of the record is found**
    - **Performance**
    - **Time to fetch a record**

$$\underbrace{T_F = r + s + btt}_{\text{(fetch track index)}} + \underbrace{r + btt}_{\text{(fetch the block to RAM)}}$$

File Organization

- **Index Sequential File (ISAM - Index Sequential Access Method)**
  - **Class Discussion and practice**
    - **With addition of new records file degraded in performance and reorganization becomes costly. Further, the file organization is totally dependent on the disk structure. If the disk crashes then the entire file has to be recreated on a new disk. This is why ISAM is not used that much.**

File Organization

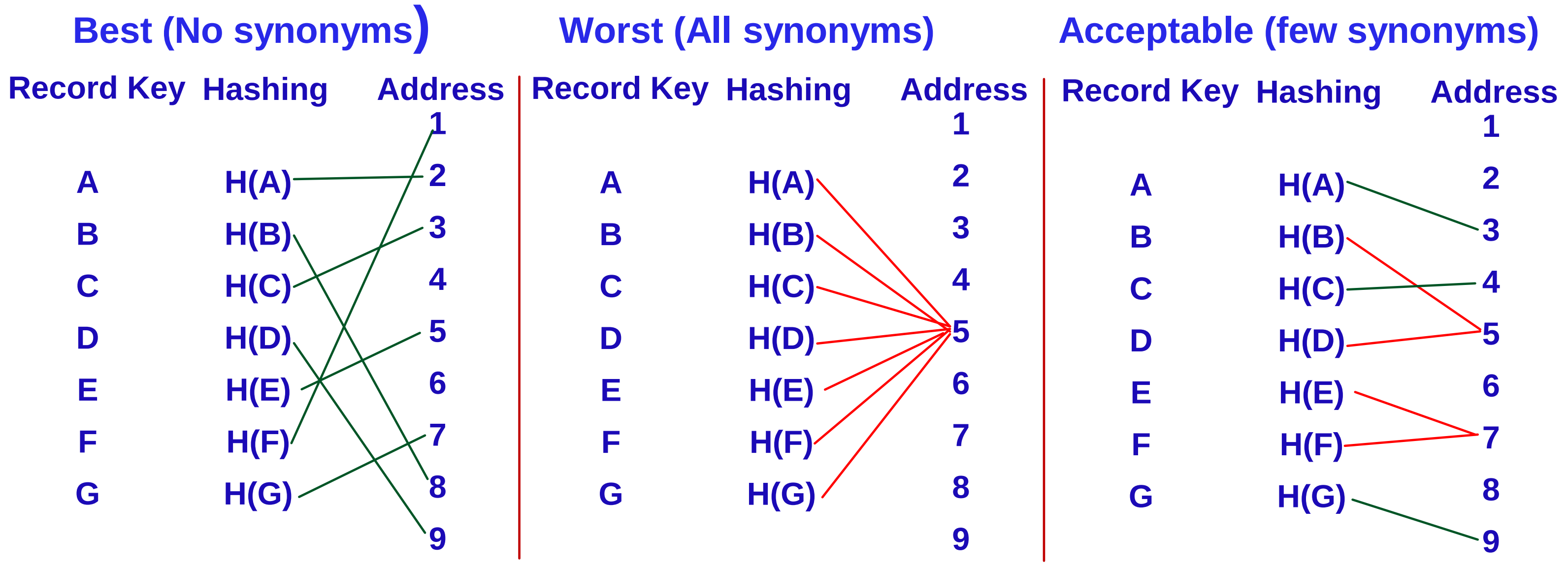
- **Direct File**
  - **Indexing provides the most powerful and flexible tools for organizing files. However, (a) indexes take up space and time to keep them organized and (b) the amount of I/O increases with the size of index. This problem can be minimized by direct file organization where the address of the desired record can be found directly (no need of indexing or sequential search). Such file are created using some *hashing* function so they are called *hashing organization* or *hashed files***
  - **https://msdn.microsoft.com/en-us/library/dn133190.aspx**
  - **http://docs.mongodb.org/manual/tutorial/shard-collection-with-a-hashed-shard-key/**

File Organization

- Hashing
  - Two types of hashing
    - Static: The address space size is predefined and does not grow or shrink with file.
    - Dynamic: The address space size can grow and shrink with file.
  - Key Space: Set of Primary keys.
  - Address Space: Set of home addresses.
    - Hash function (Primary Key) → Home address of the record.
    - Home address → file manager → disk address.
  - Requirements
    - A predefined set of home addresses (address space)
    - A hashing function.
    - Primary key (Hashed key).

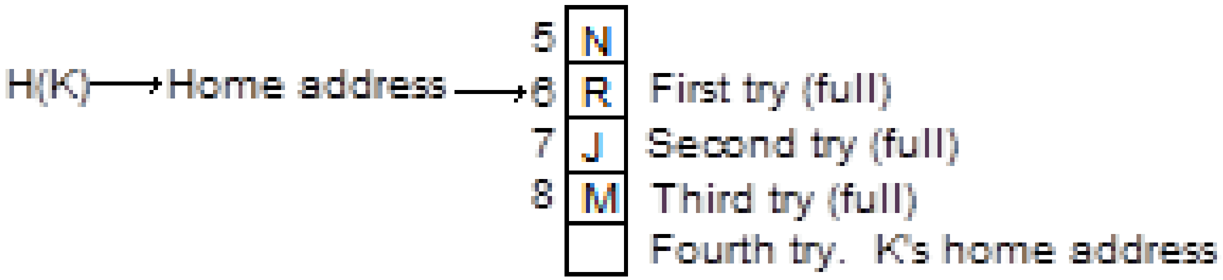
File Organization

- **Hashing**
  - **There is a unique relationship between the key and the record address. Any add, delete or change in a record must not break this relationship. If this bond is broken then the record is no longer accessible by hashing**
  - **Address Distribution**



File Organization

- Hashing
  - Collision
    - When a hashing function generates the same home address for more than one record
  - Solutions for collision
    - Progressive overflow (also called open addressing)



File Organization

- **Progressive overflow problems**
  - **What happens if there is a search for a record, but the record was never placed in the file?**
  - **What happens if a record is hashed at a filled address and only the predecessor location is empty?**
  - **These problems increase the average search length.**

File Organization

- **Progressive overflow problems**
  - **Example: Consider the following keys and their home addresses:**

Key	Home address	RAM address		No. of accesses to retrieve a record
Adams	20 (hashed)	0		
Bates	21 (hashed)	--		
Cole	21 (hashed)	20	Adams	1
Dean	22 (hashed)	21	Bates	1
Evans	20 (hashed)	22	Cole	2
		23	Dean	2
		24	Evans	5
		25		

Average search length = (1 + 1 + 2 + 2 + 5)/5 = 2.2. Acceptable average search length = 2.0. Anything higher than this is regarded high.

File Organization

- Problem in searching a record in Hashing
  - Example
    - Four keys: Adams, Jones, Morris and Smith. Collisions: Adam and Smith at address 5, and Jones and Morris at 6.
    - The following diagram illustrates the setup

Record	Home address	Actual address
Adam	5	5
Jones	6	6
Morris	6	7
Smith	5	8

Now suppose Morris is deleted, leaving an empty space. A search for Smith will starts at address 5, then looks at addresses 6 and 7. Since address 7 is now empty, it is reasonable for the program to conclude that Smith's record is not in the file

File Organization

- **Solution**
  - Mark the deleted record with a tombstone. The search for a record will not stop at a tombstone. This solution makes insertion difficult
  - Example: Record deleted - Morris. Insert record: Smith
    - The program gets the address 5 and tries to find an empty slot. During the search it encounters a tombstone. The record Smith is inserted here
  - Problem: The File has duplicate Smith
    - Solution: The program must search the entire file and then go back to the first tombstone and insert the record
  - Another problem with a tombstone is it should not be inserted when the last record or a record, which does not have any successor, is deleted

File Organization

- **Progressive overflow problems**
  - **Improvement (reduce the average search length): by storing more than one record at a single home address. Storing records in Buckets**
    - **Bucket: A logical unit of storage where more than one records are stored and the desired set of records is retrieved in one disk access (one I/O)**
  - **The home addresses of these records are the same. On sector-addressing disks, a bucket typically consists of one or more sectors. On block-addressing disks a bucket might be a block (defined by the manufacturer)**

File Organization

- **Bucket solution**
  - **Improvement**

Key	Home address
Green	30
Hall	30
Jenks	32
King	33
Land	33
Marx	33

A set of buckets

30	Green	Hall		bucket
31				bucket
32	Jenks			bucket
33	King	Land	Marks	bucket

File Organization

- **Dynamic Hashing (Extensible Files also called Linear Hashing)**
  - **Concept: expand the file on need basis**
  - **Expansion approach: split the file space, that is, acquire extra space and integrate it to the existing set up.**

File Organization

- Virtual Hashing
  - Uses multiple hashing functions. These functions are related and the selection of the function depends upon the level of bucket split.
  - Assumption
    - N = Number of buckets
    - Home address is generated by = Key mod N (Hashing function)
  - The record goes into a bucket. When the bucket is full then to accommodate an insert do the following:
    - Split the bucket. Use a different but related hashing function to distribute the records of the old bucket between the old and the new buckets
    - The desired record is inserted in these buckets using the current hashing function.

File Organization

- **Virtual Hashing: An example**
  - **Assume**
    - **N = 100, C = 4 (bucket size). Record Keys: 508, 17308, 208, 408.**
    - **Hashing function: address = *key mod 100***
    - **Records hashed into the bucket as follows:**

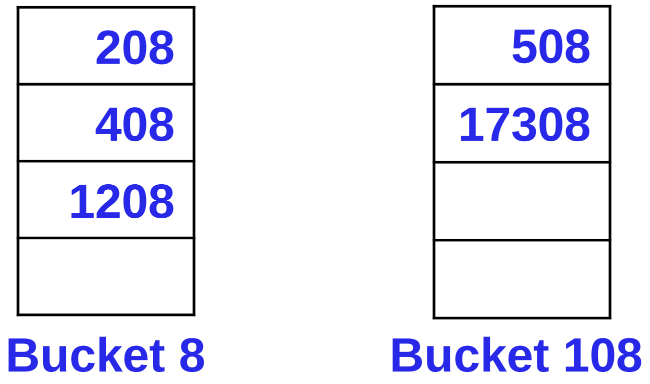
508
17308
208
408

Bucket 8
This bucket is full

File Organization

Virtual Hashing: An example

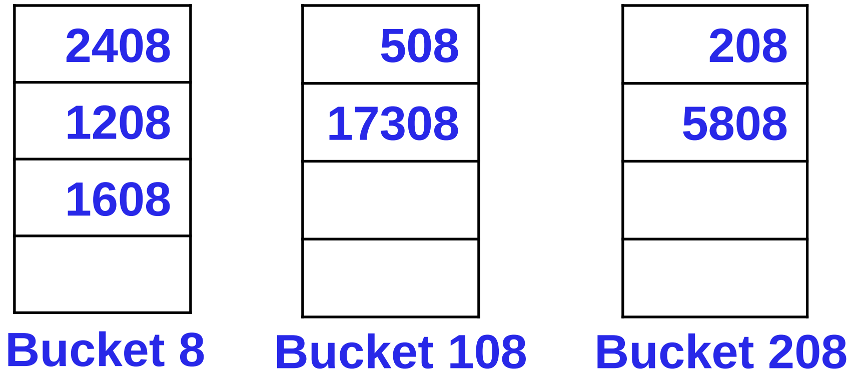
- The bucket is full. Insert record: 1208. The insertion forces a split. A new bucket is acquired on the disk. The reorganization of these records is done first in RAM and then they are written to the disk buckets (old and new).
- The five records will be rehashed using function: *address = key mod 2N (key mod 200)*
- The following diagram shows the state of the file after the distribution of the four old records and the insertion of new record with key 1208



File Organization

Virtual Hashing: An example

- Insert record 1608 into bucket 8 and then insert record 5808
- Cannot accommodate 5808 in bucket 8 so split
- Use hashing: *address = key mod 4N (key mod 400)*

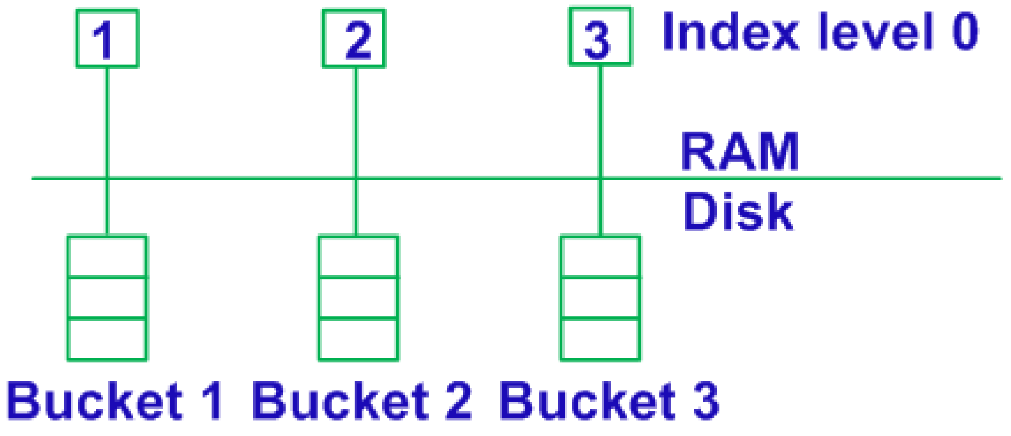


File Organization

- Virtual Hashing
  - In general, if we are splitting a bucket that has been involved as destination of S previously splits, we use the function: *address = key mod* $(2^{s+1} \times N)$
  - Disadvantages
    - Waste of space (unused buckets after each split)
    - If two buckets *n* and *n+j* are in use then all buckets between *n* and *n+j* must be available to the algorithm for inserting records
    - The history of insertion must be saved to access a record. This is because many related hashing functions are used in the search operation

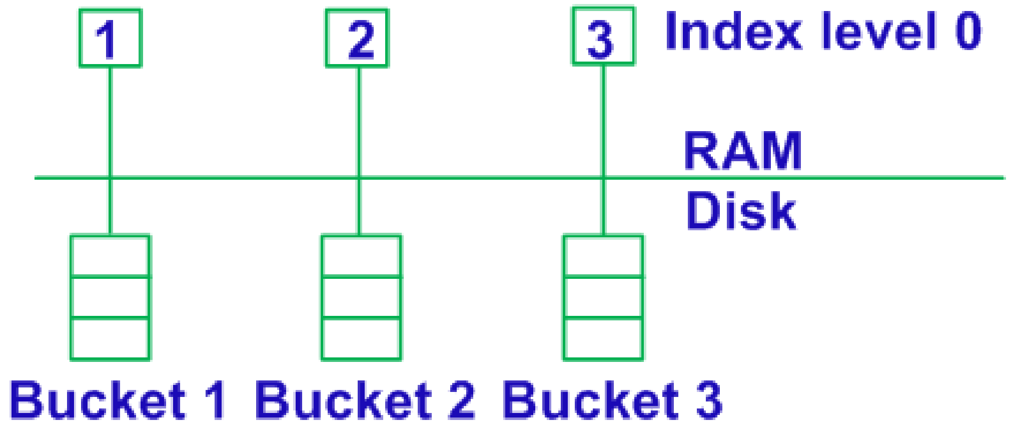
File Organization

- **Dynamic Hashing**
  - **It also uses buckets but unlike virtual hashing, they are not related. Initially files are organized in *N* buckets. These buckets reside on the disk but each of them is pointed to by an index (in RAM)**
  - **Structure: Bucket size C = 3 records**



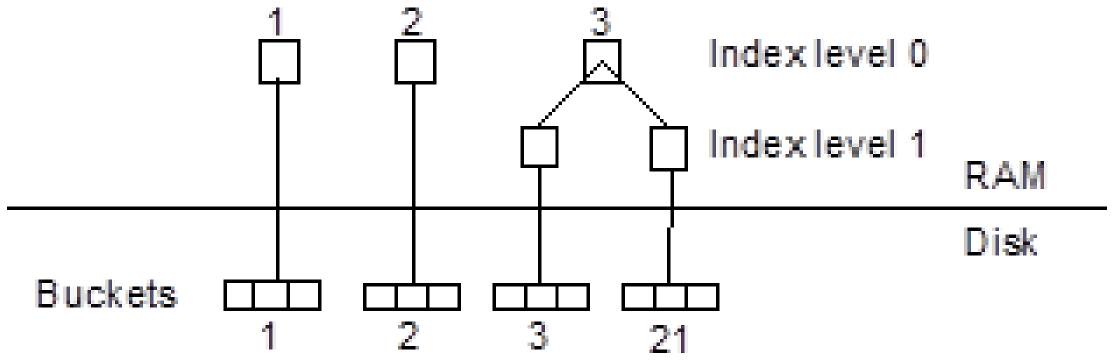
File Organization

- **Dynamic Hashing**
  - **A hash function transforms a key into an address of a level 0 index element (1 through 3 in this case). The appropriate bucket is accessed by the pointer**



File Organization

- Dynamic Hashing
  - Insert a record
    - The hashing function gives the address of an index element and the record is inserted in the bucket pointed to by that node. If the target bucket is full, a new bucket is allocated to the file. The C + 1 records are redistributed between the two buckets. These two buckets then are pointed to by the children of the original index node as shown in the following diagram



File Organization

- Dynamic Hashing
  - Two questions
    - In redistributing of records between two buckets (new bucket and old bucket), how do we decide which record goes to which bucket?
    - A record search begins from the Index level 0. When searching a record, how do we find which of the two buckets pointed to by the index contains the record?

File Organization

- Dynamic Hashing
  - Answers
    - Use a second function. This function converts a key into an arbitrarily long binary string (1's and 0's) of fixed length (predefined) as shown below. This string is referred to here as B (Key)

Key	H (Key)	B(Key)
157	2	10100…
95	1	00011…
83	1	01100…
205	2	10010…
13	1	10111…
125	1	10001…
6	1	01000…
301	1	00110…

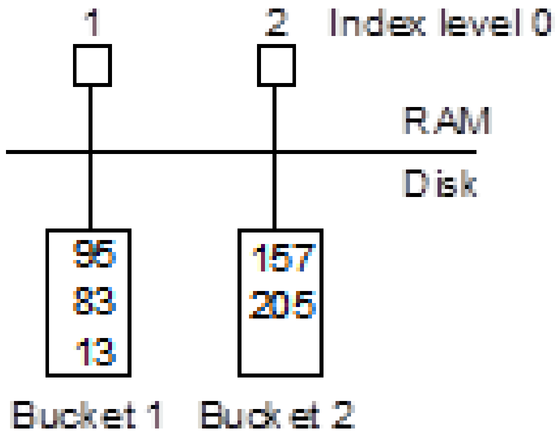
File Organization

Dynamic Hashing

How is a B (Key) used?

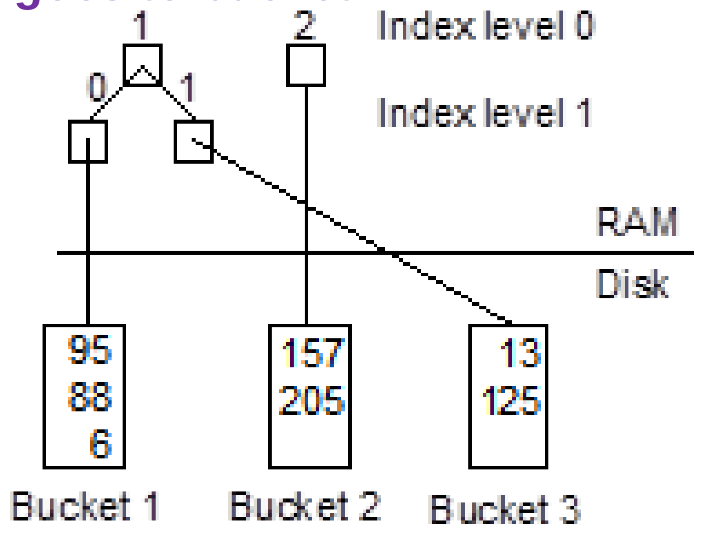
- The bits of B are used to identify the correct bucket for inserting and for searching the desired record. If the bucket being split is pointed to from a node at level I in the tree, then we use the value of the (I + 1)st digit of B (key) to determine if a record goes to the left (old) or to the right (new) bucket. The following diagram shows the insertions of the first five records

File after 5 insertions



File Organization

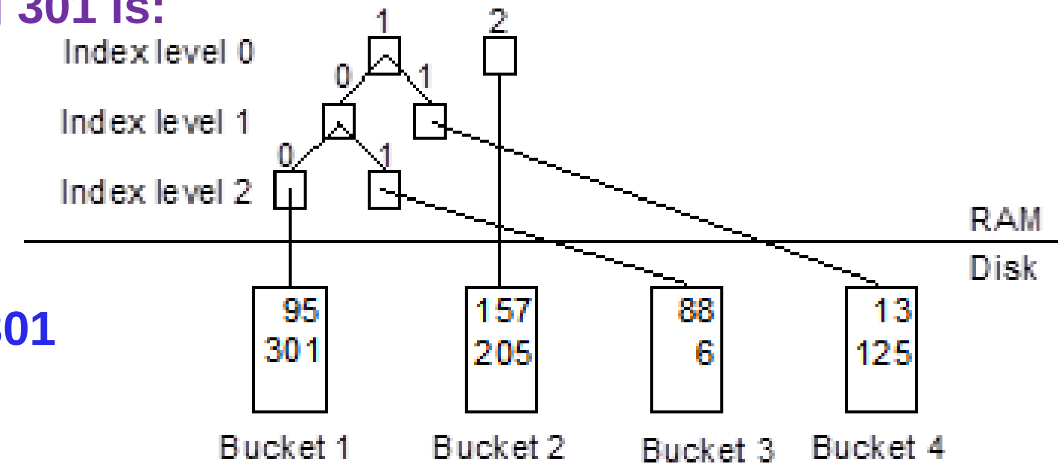
- Dynamic Hashing
  - Insert 125 and 6
    - H(125) → 1. So 125 is inserted in bucket pointed by node 1 of level 0. The bucket is full so it splits (old and new). Use the first digit in B (key) string to decide the destination bucket of 125. 95 and 88 go to the left bucket, and 13 and 125 go to the right bucket. 6 goes to bucket 1



Split after inserting 125.
No split in inserting 6

File Organization

- Dynamic Hashing
  - Insert 301
    - H(301) → 1. so its B (key) takes it to bucket 1. Bucket 1 is full so a split occurs. The bucket to be split is being pointed to by a leaf at index level 1. So in distributing the records (old) and inserting new record (301), 2nd bit (Index level 1+1 = 2) of B key is used for distribution. The structure of the file after inserting 301 is:



Split to accommodate 301

File Organization

- **Indexing and Hashing**
  - Hashing of any form is not at all good for sequential access. So a process, which requires a few logically contiguous records may not be processed efficiently and indexing is the best organization for such requirements. The performance of index organization degrades when the volume of updates and insertion increases.
  - There are a large number of different file organizations such as extendible hashing, linear hashing, B-trees group (B^+-tree, B-tree, etc.) but all are based in some form of indexing and hashing. In a database system, a number of different file organizations are used to store data. In many cases the same data is stored in multiple file organizations to improve data retrieval for different types of queries (key query, range query, mult-key queries, etc.)

Self study

- B/B+ Trees: Textbook Pg: 646 – Pg: 667 (6th Edition)
- B/B+ Trees: Textbook Pg: 617 – Pg: 638 (7th Edition)