



Tree-structured Indexing and Hash-based Indexing

Group 3

**Consider a file of student records
sorted by gpa.**

**Find all students with gpa
higher than 8.0**

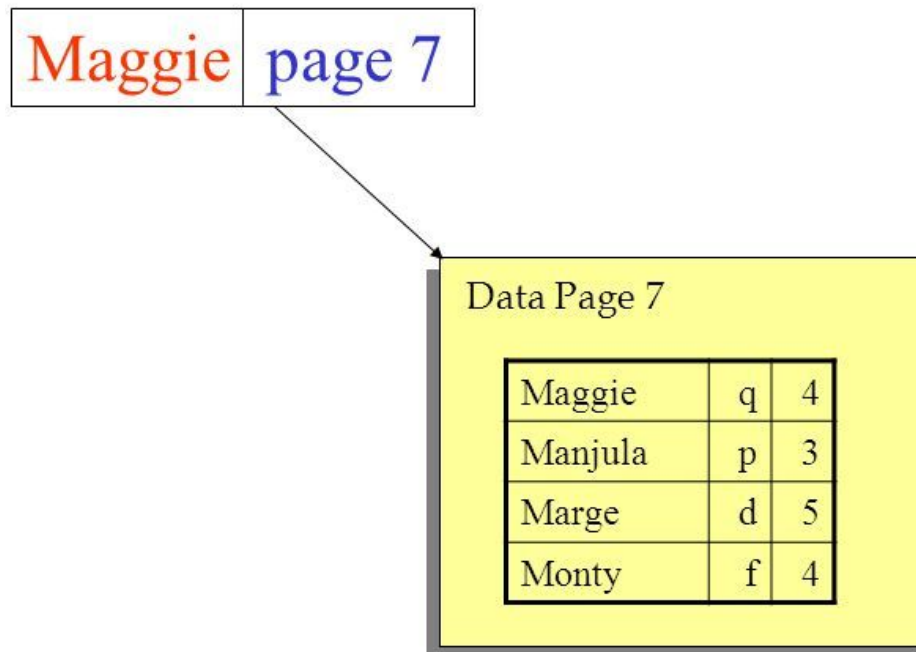
—

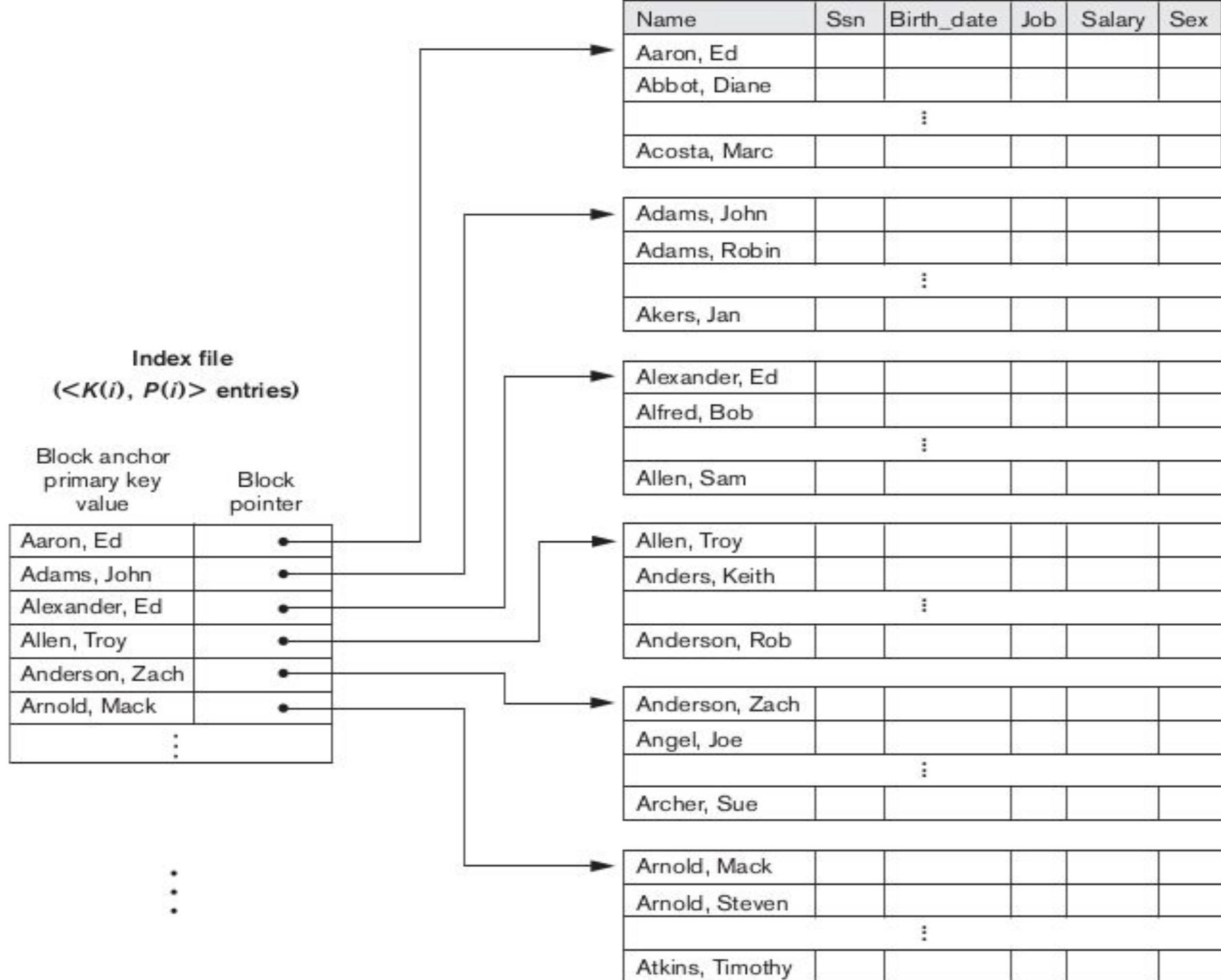
Indexed Sequential Access Method

ISAM is a static index structure that is effective when the file is not frequently updated, but it is unsuitable for files that grow and shrink a lot.

Format of an Index Page

An index entry is a $\langle \text{key}, \text{pointer} \rangle$ pair where key is the value of the first key on the page, and pointer points to the page.







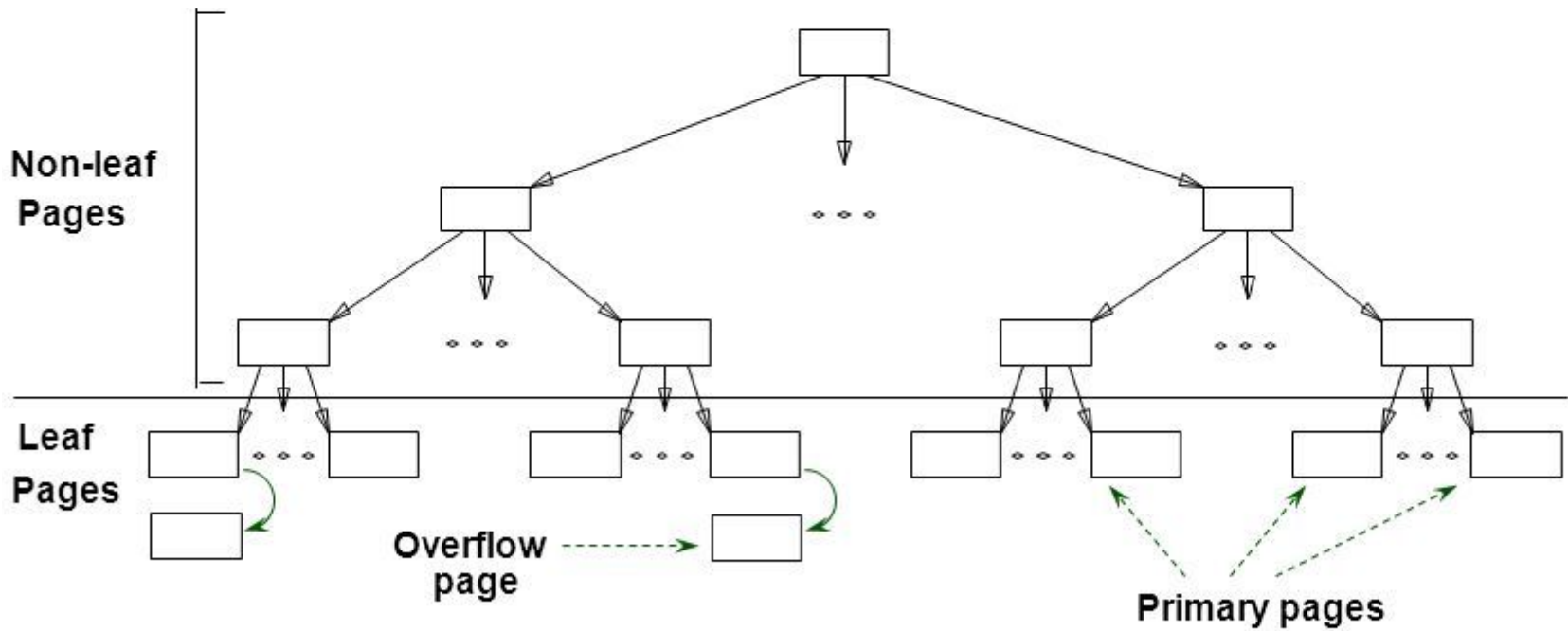
One-Level Index Structure

- As the size of an entry in the index file (key value and page id) is likely to be much smaller than the size of a page, and only one such entry exists per page of the data file, the index file is likely to be much smaller than the data file; therefore, a binary search of the index file is much faster than a binary search of the data file.
- However, a binary search of the index file could still be fairly expensive, and the index file is typically still large enough to make the operations expensive.

Why not apply the previous step of building an auxiliary structure all the collection of index records and so on recursively until the smallest auxiliary structure fits in one page?

—


ISAM Index Structure





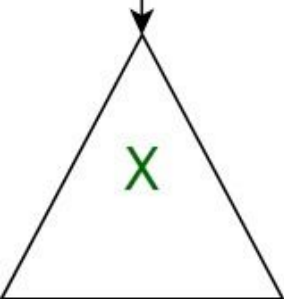


B+ Trees

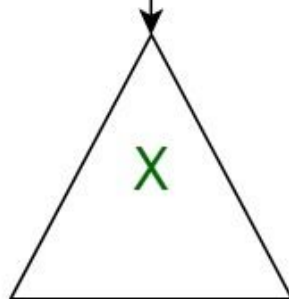
- 
- A static structure such as the ISAM index suffers from the problem that long overflow chains.
 - A balanced tree in which the internal nodes direct the search.
 - The leaf nodes contain the data entries.



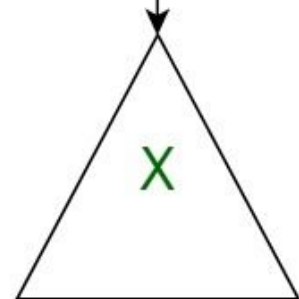
- Each internal node is of the form $\langle P_1, K_1, P_2, K_2, \dots, P_{c-1}, K_{c-1}, P_c \rangle$.
- P_i is a **tree pointer** and, each K_i is a **key value**.
- The root node has, at least two tree pointers, while the other internal nodes have at least $\lceil a/2 \rceil$ tree pointers.



$$X \leq K_1$$



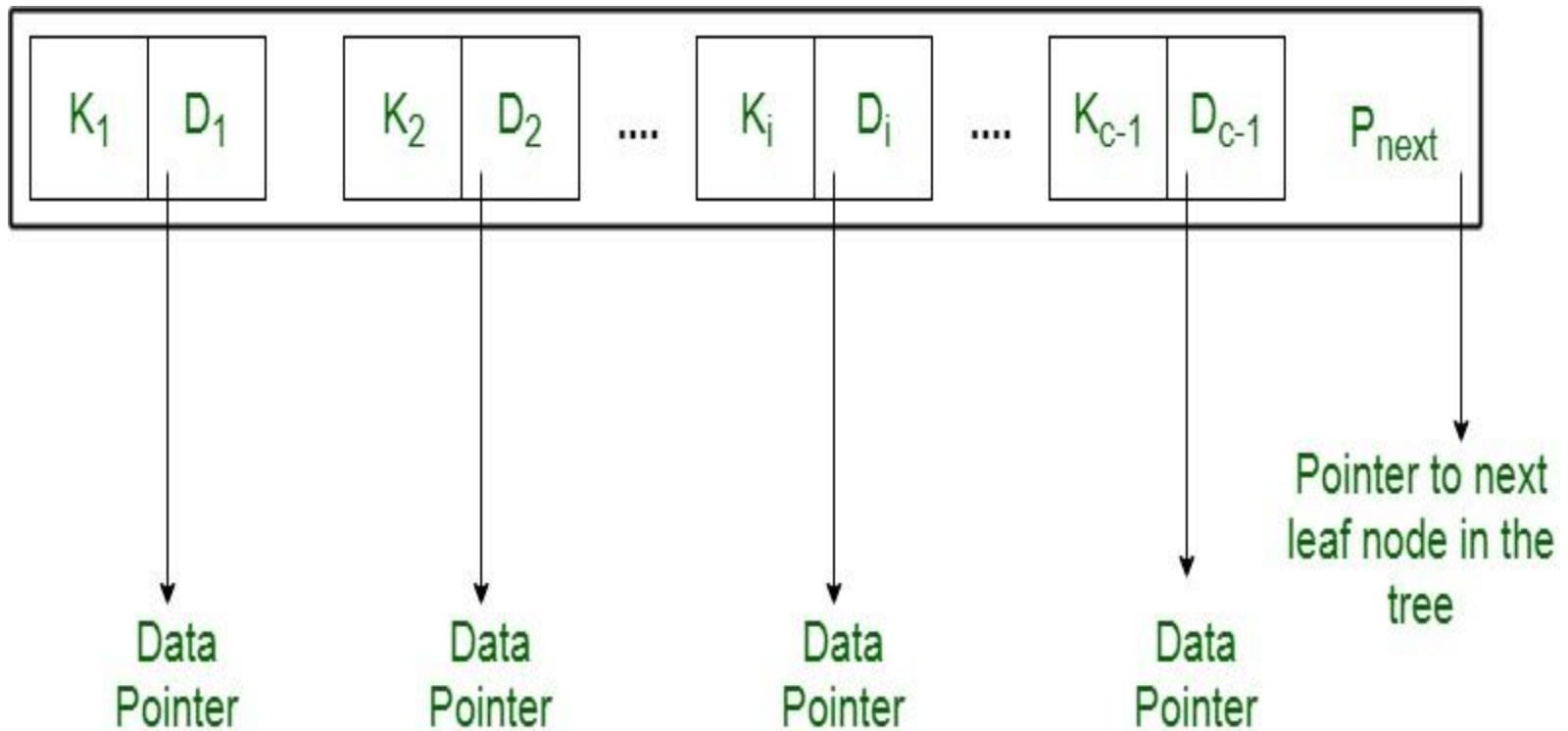
$$K_{i-1} < X \leq K_i$$



$$K_{c-1} < X$$



- Each leaf node is of the form $\langle \langle K1, D1 \rangle, \langle K2, D2 \rangle, \dots, \langle K_{c-1}, D_{c-1} \rangle, P_{next} \rangle$.
- **D_i** is a **data pointer**(points to actual point in the disk).
- **P_{next}** points to next leaf node.





Insertion

The search operation is started in the root node and it proceeds in every node x as follows:

- If x is a non-leaf node, we seek for the first router value $x.router$ which is greater than or equal to the key k searched for. After that the search continues in the node pointed by $x.ci$
- If all router values in node x are smaller than the key k searched for, we continue in the node pointed by the last pointer in x .
- If x is a leaf node, we inspect whether x is stored in this node.



B+ Tree Data Deletion

#Start at the root and go up to leaf node containing the key K

#Find the node n on the path from the root to the leaf node containing K

A. If n is root, remove K

a. if root has more than one keys, done

b. if root has only K

(i) if any of its child node can lend a node

*Borrow key from the child
and adjust child links

(ii) Otherwise merge the children
nodes it will be new root

c. If n is an internal node, remove K

(i) If n has at least $\lceil m/2 \rceil$ keys,

*done!

(ii) If n has less than $\lceil m/2 \rceil$ keys,

*If a sibling can lend a key,

-> Borrow key from the sibling

and adjust keys in n and the parent node

-> Adjust child links

*Else

-> Merge n with its sibling

-> Adjust child links

d. If n is a leaf node, remove K

(i) If n has at least $\text{ceil}(M/2)$ elements, done!

*In case the smallest key is deleted,

push up the next key

(ii) If n has less than $\text{ceil}(m/2)$ elements

*If the sibling can lend a key

-> Borrow key from a sibling

and adjust keys in n and its

parent node

*Else

-> Merge n and its sibling

-> Adjust keys in the parent

node

Hash-Based Indexing

Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure

—



Dynamic Hashing:

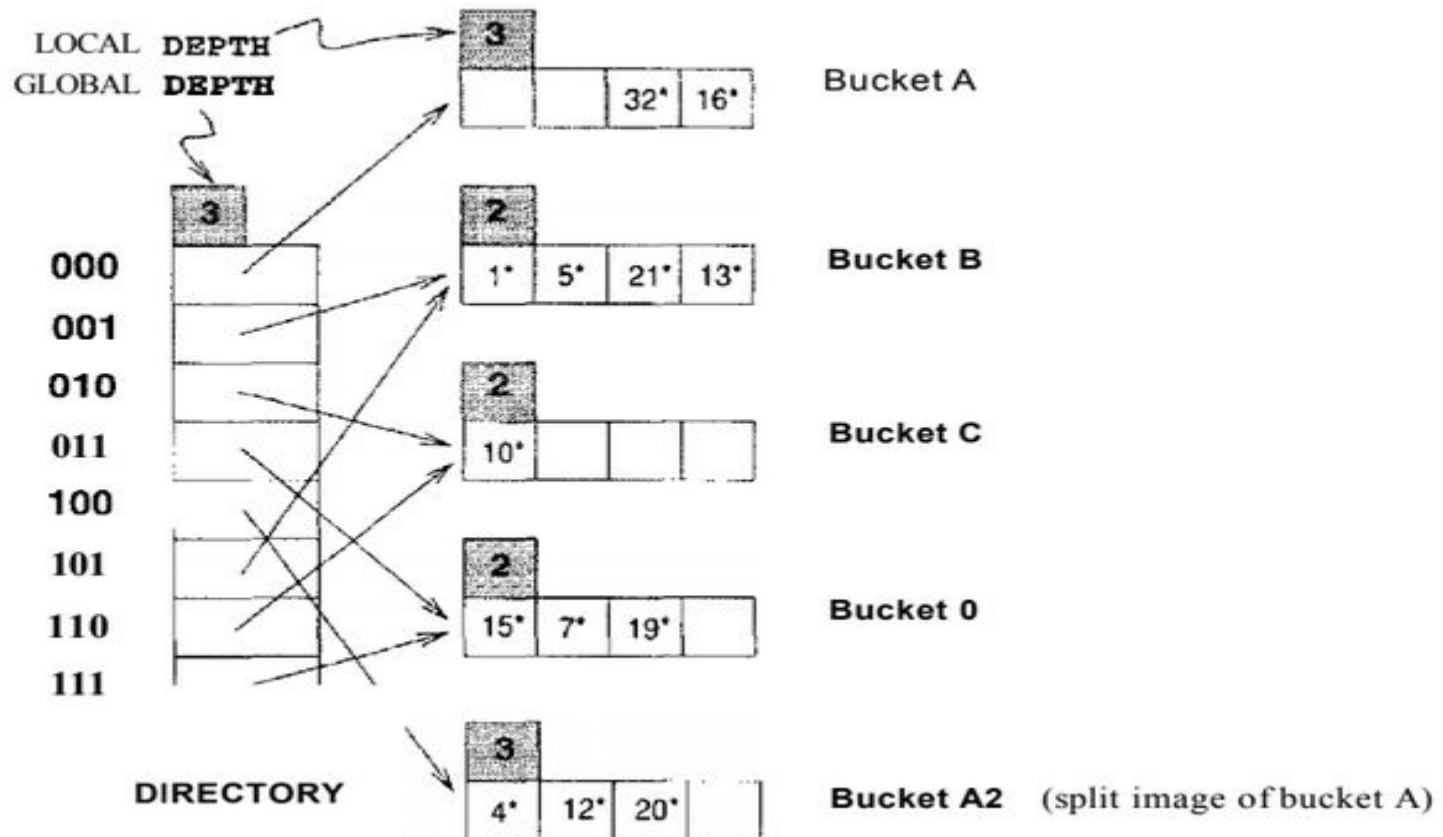
1. Extendible hashing
2. Linear Hashing



Extendible Hashing

- Directory of pointers to buckets and double the size of the number of buckets by doubling just the directory and splitting only the buckets that overflowed.
- Hashing function-binary number and interpret the last 'd' bits , 'd'-size of directory

Global and Local depth





Global Depth:

It is used to locate the data entry

Local Depth:

1. To determine directory splitting is required.
2. If local depth $>$ global depth, then directory split.

Can we use first d bits (the most significant bits) instead of the last d bits (least significant bits)?

—