Chapter 8

Indexing

Indexing

8.1 The Concept of Indexing

8.2 Disk Storage

8.3 The B-Tree Index

8.4 Clustered and Non-Clustered Indexes

8.5 A Hash Primary Index

8.6 Throwing Darts at Random Slots

8.1 The Concept of Indexing

- **Index**
  - an index is like a card catalog in a library.
  - Each card (entry) has:

    (keyvalue, row-pointer)

    - keyvalue is for lookup
    - row-pointer (ROWID) is enough to locate row on disk (one I/O)

8.1 The Concept of Indexing

- **Index: (keyvalue, row-pointer)**
  - **Entries are placed in Alphabetical order by lookup key in "B-tree" (usually), also might be hashed.**
    - **An index is a lot like memory resident structures.**
    - **But index is disk resident. Like the data itself, often won't all fit in memory at once.**
  - **Index can be regarded as a table with two columns: keyvalue and row-pointer**

8.1 The Concept of Indexing

- **Figure 8.1: the SQL Create Index Statement**

CREATE [UNIQUE] INDEX indexname ON tablename (colname [ASC | DESC] { , colname [ASC | DESC] ...});
DROP INDEX indexname;

- **UNIQUE**
  - **keyvalue and row is one-to-one**
  - **we can use UNIQUE INDEX to implement UNIQUE constraints in Create Table Statement.**

8.1 The Concept of Indexing

- **Example 8.1.1**

  create index citiesx on customers(city);

  - **each index key value (i.e., city name) in the citiesx index can correspond to a large number of different customers rows.**

8.1 The Concept of Indexing

- **Example 8.1.2**

  create unique index cidx on customers(cid);

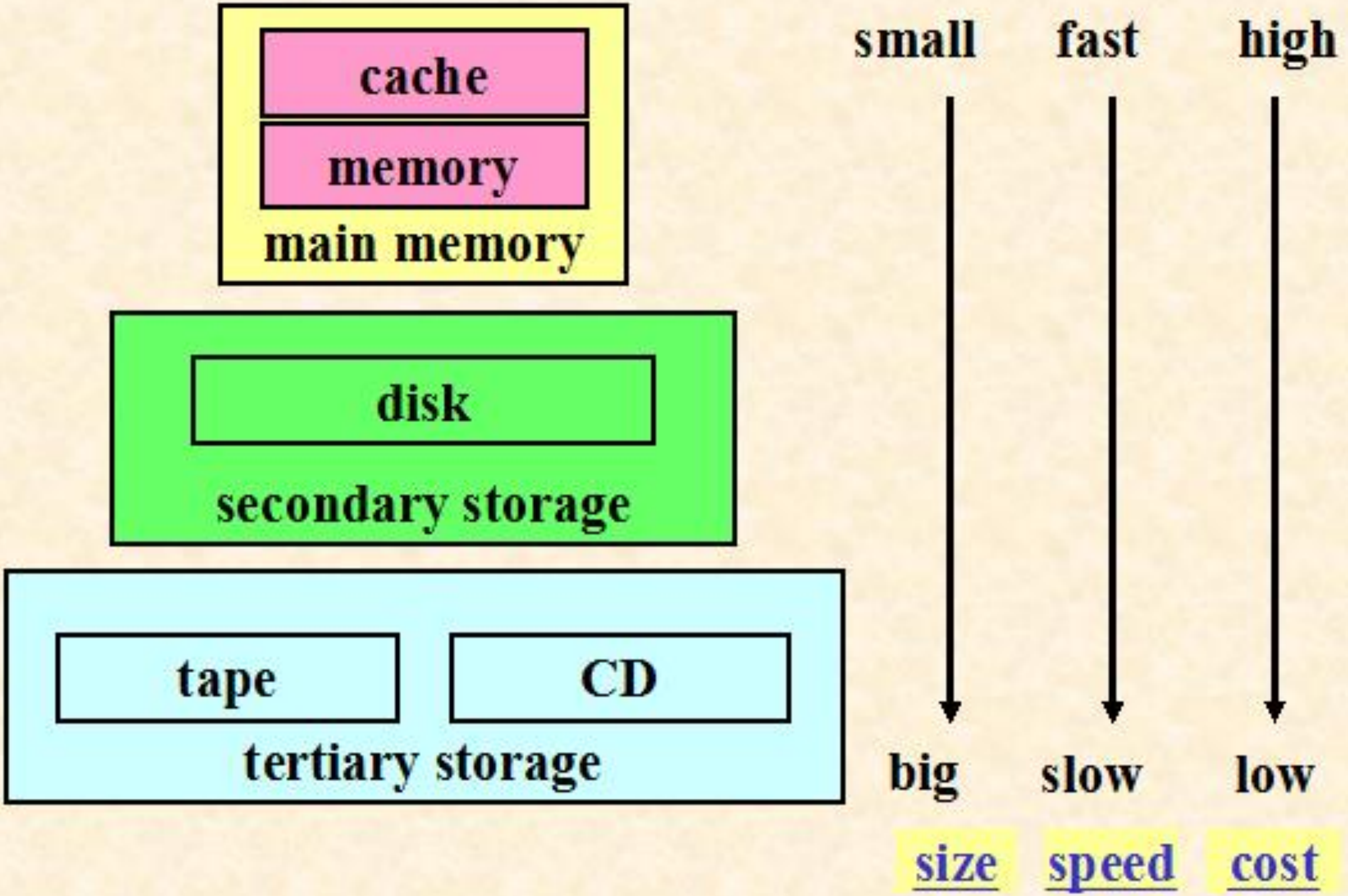
  - **each index key value in the cidx index only correspond to a customers rows.**
    - **Index key is quite different from relational concept of primary key or candidate key.**
    - **But, we can use UNIQUE INDEX and NOT NULL to implement primary key or candidate key.**

8.1 The Concept of Indexing

- **Indexing**
  - **After being created, index is sorted and placed on disk.**
    - **Sort is by column value asc or desc, as in SORT BY description of Select statement.**
  - **NOTE**
    - **LATER CHANGES to a table are immediately reflected in the index, don't have to create a new index.**

8.2 Disk Storage

- **Database Storage**
  - **Computer memory**
    - **very fast but Volatile (挥发性的) storage.**
  - **Disk storage**
    - **very slow but non-volatile and very cheap.**


cache
memory
main memory
disk
secondary storage
tape
CD
tertiary storage
small
fast
high
big
slow
low
size
speed
cost

8.2 Disk Storage

- **Disk Access**
  - **Seek time .008 seconds**
    - **The disk arm moves in or out to the proper cylinder position.**
  - **Rotational latency .004 seconds**
    - **The disk platter rotates to the proper angular position.**
  - **Transfer time .0005 seconds**
    - **The disk arm reads/writes the disk page on the appropriate surface.**
  - **access memory: 10^{-8}~10^{-7} seconds**

8.2 Disk Storage

- **Memory Buffer (Figure 8.2, pg. 340)**
  - **Read pages into memory buffer so can access them.**
    - **Once to right place on buffer, transfer time is cheap.**
  - **Everytime want a page from disk, hash on dkpgaddr, h(dkpgaddr) to entry in Hashlookaside table to see if that page is already in buffer.**

8.2 Disk Storage

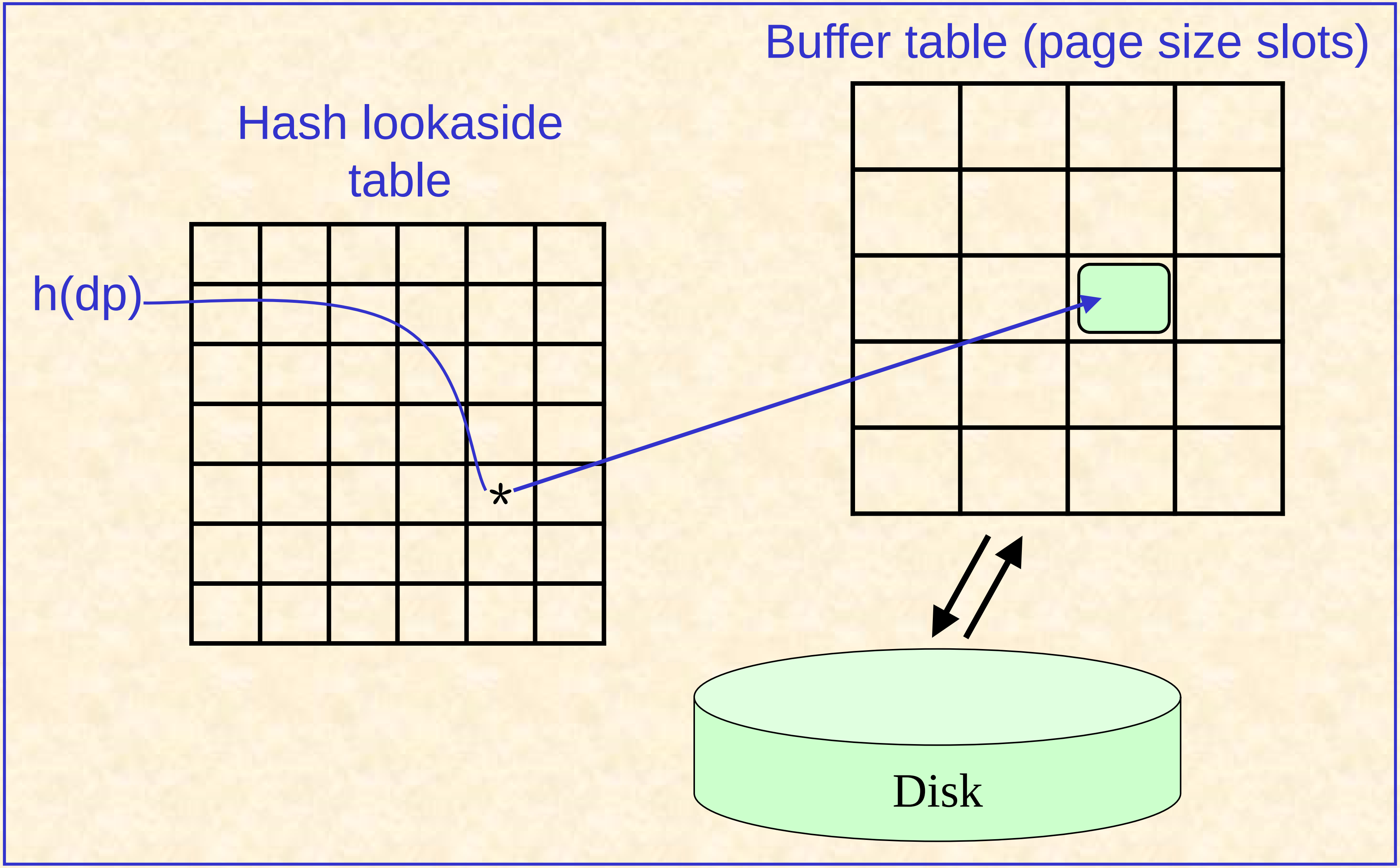


Figure 8.2 Disk Page Buffering and Lookaside

8.2 Disk Storage

- **Memory Buffer (cont.)**
  - **Advantages**
    - **saved disk I/O**
    - **find something for CPU to do while waiting for I/O.**
      - **This is one of the advantages of multi-user timesharing.**
      - **Can do CPU work for other users while waiting for this disk I/O.**

Create Tablespace in ORACLE

- Figure 8.3, pg. 342
- Tablespace
  - made up of OS files cross

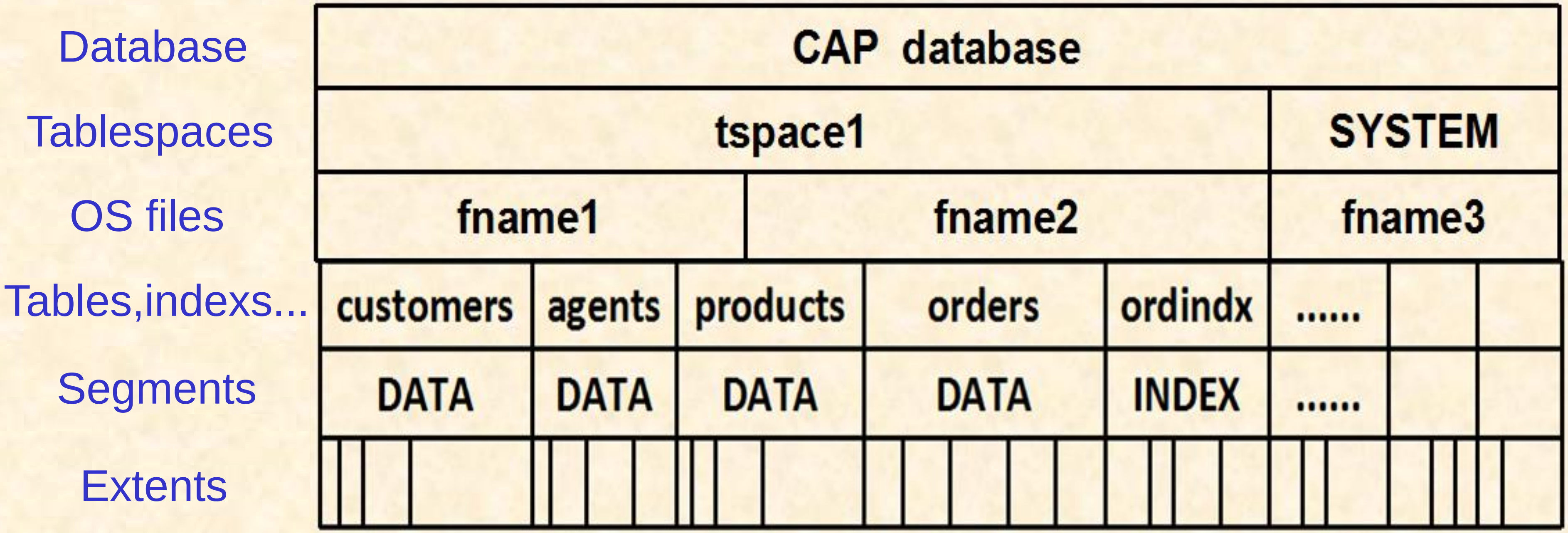


Figure 8.3 Database Storage Structures

8.2 Disk Storage

- **Data Storage Pages and Row Pointers (Figure 8.6, pg. 345)**
  - **A row on most architectures is a contiguous sequence of bytes. N rows placed on one page (called a *Block* in ORACLE).**
  - **Header info names the kind of page (data segment, index segment), and the page number (in ORACLE the OS file and page number).**

8.2 Disk Storage

- **Figure 8.6: Data Storage Pages and Row Pointers**

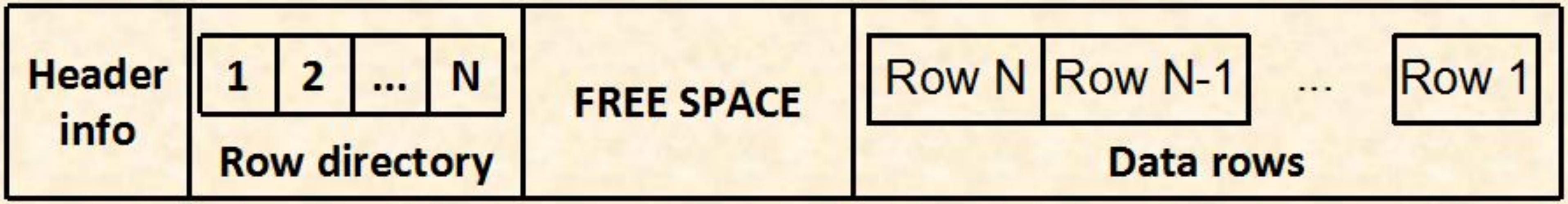
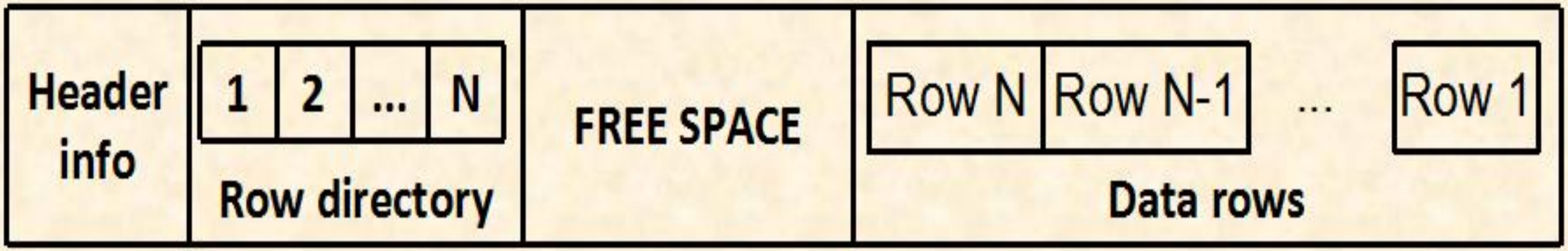
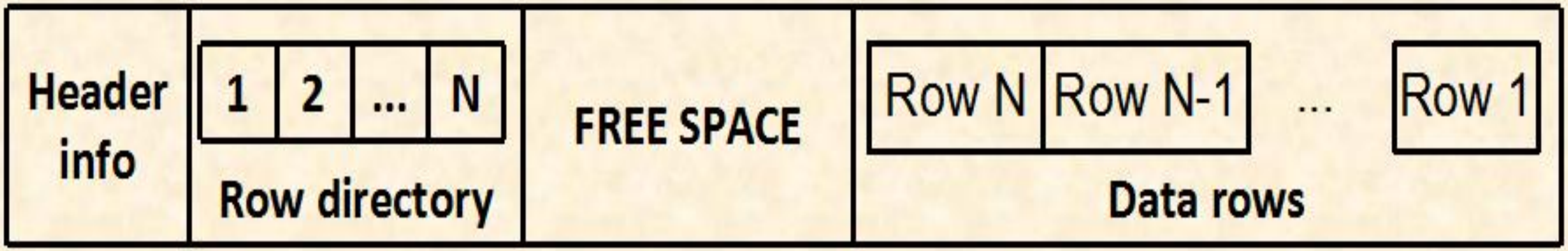


Figure 8.6 Row Layout on a Disk Page



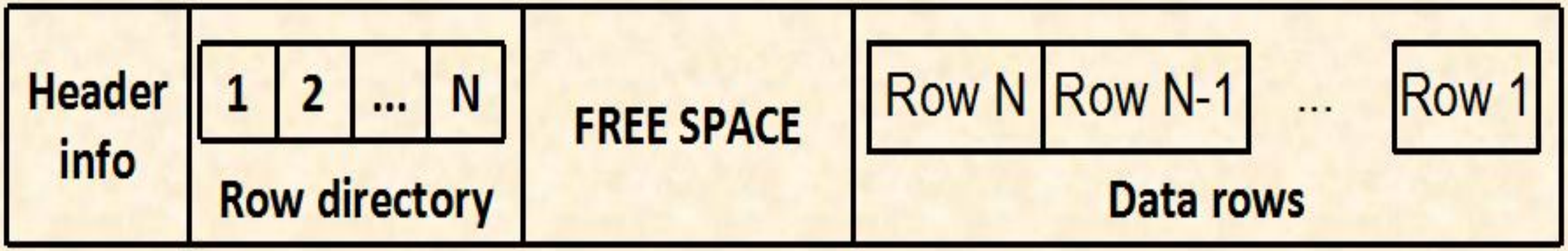
❑ROWs in page

1. Rows added right to left from right end on block.
2. Row Directory entries left to right on left after header. Give offsets of corresponding row beginnings.
3. Provide number of row slot on page.



- When new row added, tell immediately if we have free space for new directory entry and new row.
- Conceptually, all space in middle, implies when delete row and reclaim space must shift to fill in gap.
  - Also might have "Table Directory" in block when have CLUSTER.

8.2 Disk Storage



- **Disk Pointer**
  - **A row in a table can be uniquely specified with the page number (P) and slot number (S).**

- **Rows Extending Over Several Pages**
  - **Product Variations**

Indexing

8.1 The Concept of Indexing

8.2 Disk Storage

8.3 The B-Tree Index

8.4 Clustered and Non-Clustered Indexes

8.5 A Hash Primary Index

8.6 Throwing Darts at Random Slots

8.3 The B-Tree Index

- **The B-Tree**
  - **Each node of the tree**
    - **takes up a full disk page**
    - **has a lot of fanout**

❑Create Index statement in ORACLE

```
CREATE [ UNIQUE | BITMAP ] INDEX
        [schema.]indexname ON tablename
        ( colname [ ASC | DESC ]
           { , colname [ ASC | DESC ] … } )
       [ TABLESPACE tblespace ]  [ STORAGE . . . ]
       [ PCTFREE n ]  [ ...... ]
       [ NOSORT ]
```

- **UNIQUE | BITMAP**
- **ASC | DESC**
- **TABLESPACE**
- **PCTFREE**
- **NOSORT**

8.3 The B-Tree Index

- **Creating Index**
  1) reads through all rows on disk (assume N)
  2) pulls out (keyvalue, rowid) pairs for each row
  3) Get following list (<u>in order by keyvals</u>) put out on disk

(keyval$_1$, rowid$_1$) (keyval$_2$, rowid$_2$) . . . (keyval$_N$, rowid$_N$)

- If have NOSORT clause, rows are in right order, so don't have to sort.

Table S

ROWID	sno	name	dept	age
1	S_1	Lu	CS	18
2	S_2	Li	CS	17
3	S_3	Xu	MA	18
4	S_4	Lo	CS	18
5	S_5	Lin	PH	19
6	S_6	Wang	CS	17
7	S_7	Sen	MA	17
8	S_8	Shen	PH	18

Index sdx

sno	ROWID
S_1	1
S_2	2
S_3	3
S_4	4
S_5	5
S_6	6
S_7	7
S_8	8

Index ndx

name	ROWID
Li	2
Lin	5
Lo	4
Lu	1
Sen	7
Shen	8
Wang	6
Xu	3

- create unique index sdx on S(sno) NOSORT;
- create index ndx on S(name) PCTFREE 25;

❑ Example 8.3.1 (pg. 349) Idea of Binary Search

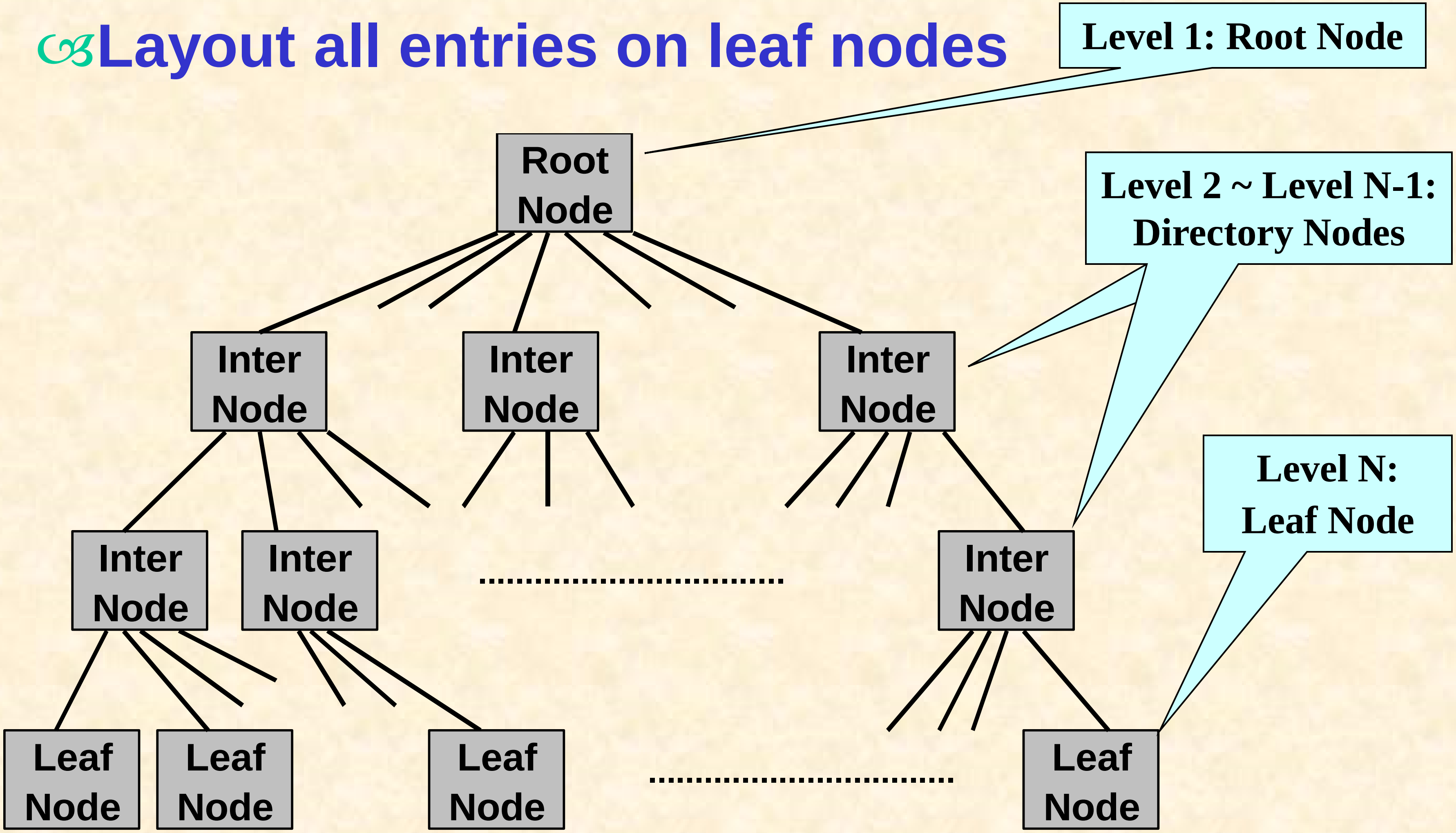
```
/* binsearch: return K so that arr[K].keyval == x,
                    or -1 if no match; * /
int binsearch(int x) {
    int probe = 3, diff = 2;
    while (diff > 0) {        /* loop until K to return * /
        if (probe <= 6 && x > arr[probe].keyval)
                probe = probe + diff;
        else   probe = probe - diff;
        diff = diff / 2;
    }    /* we have reached final K * /
    if (probe <= 6 && x == arr[probe].keyval) return probe;
    else if (probe+1 <= 6 && x == arr[probe+1].keyval)
        return probe + 1;
    else return -1;  }
```

8.3 The B-Tree Index

- **Exp 8.3.2 Binary of a Million Index Entries**
  - **number of entries: 1000000 (10^6)**
  - **size of entry: 8 bytes**
  - **size of disk page: 2k bytes**
    - **number of entries in each disk page: 250**
    - **total number of disk pages: 4000**
  - **average number of disk I/O**
    - **$\log_2 4000 + 1 \approx 13$**

8.3 The B-Tree Index

- **Example 8.3.3 B-Tree Structure**
  - Layout all entries on leaf nodes



8.3 The B-Tree Index

P_1	K_1	P_2	K_2	……	P_m	K_m	P_{m+1}

The Structure of B-Tree Node

- $K_1, K_2, \ldots, K_m$ are keyvalues, and $K_1 < K_2 < \ldots < K_m$
- **n** is maximum number of keyvalue in a node

1) **Leaf Node:** $\lfloor (n+1)/2 \rfloor \le m \le n$

- P_i is a ROWID which keyvalue is K_i $(i=1,2,\ldots,m)$
- P_{m+1} is a pointer to next leaf node

8.3 The B-Tree Index

P_1	K_1	P_2	K_2	……	P_m	K_m	P_{m+1}

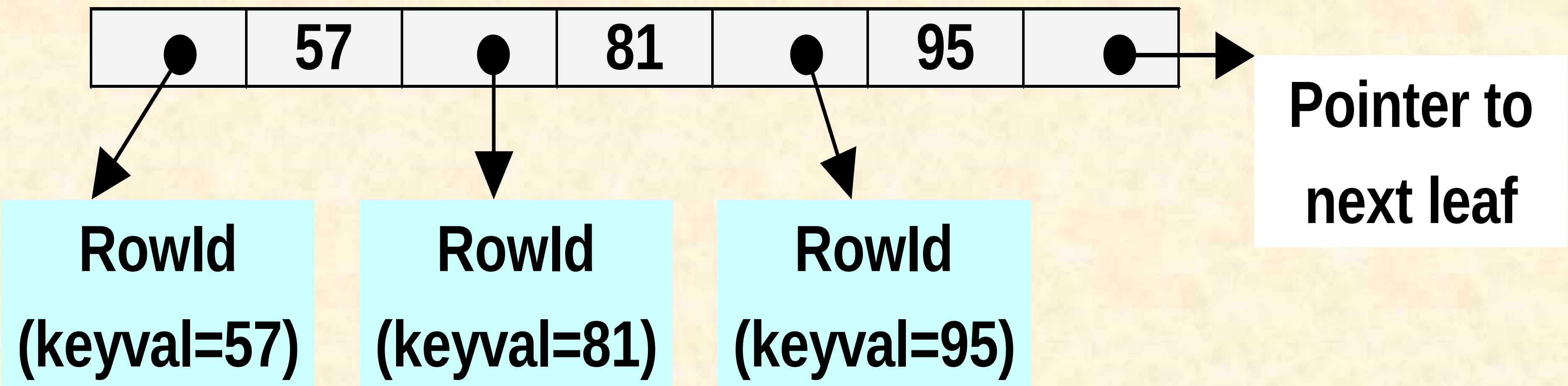
2) Root Node: $1 \le m \le n$

3) Inter Node：$\lceil (n-1)/2 \rceil \le m \le n$

- P_i is a pointer to child node (root node of sub-tree T_i)
- for each keyvalue K on sub-tree T_i, we have
  - If $i = 1$: $K < K_1$
  - If $1 < i \le m$: $K_{i-1} \le K < K_i$
  - If $i = m+1$: $K \ge K_m$

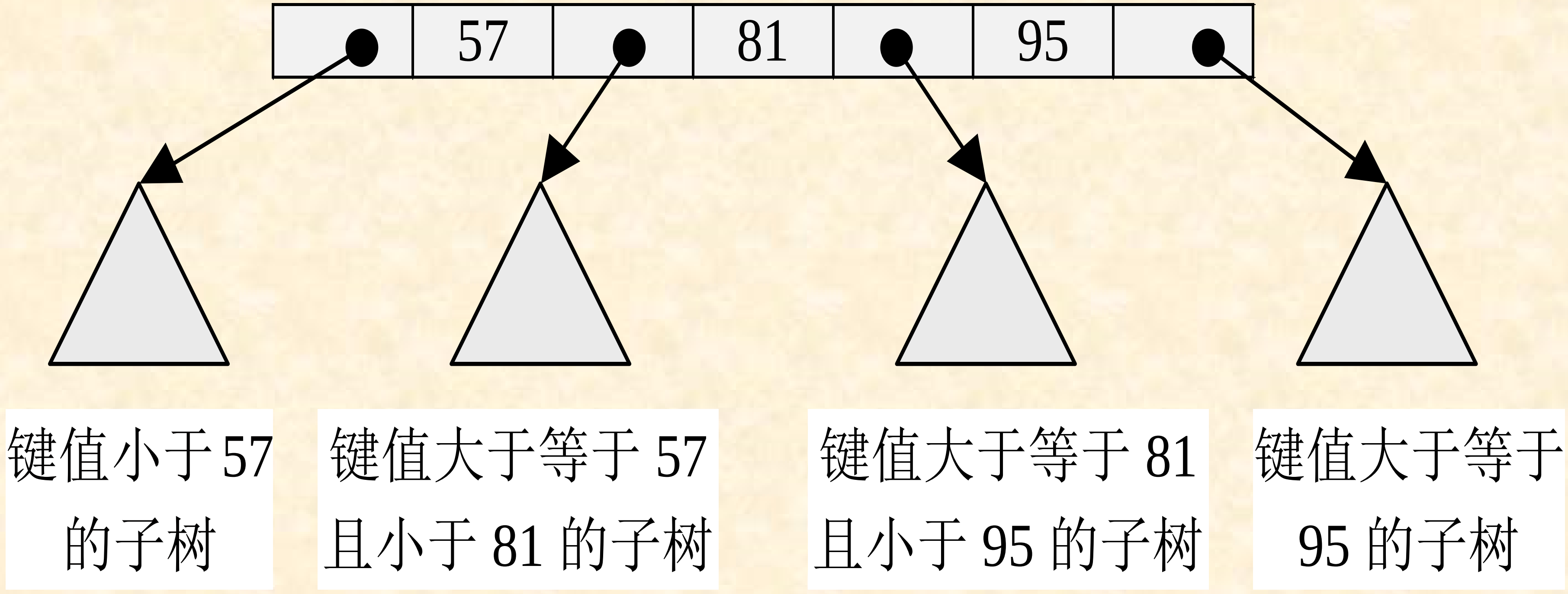
8.3 The B-Tree Index

- **Example: leaf node (n= 3)**



8.3 The B-Tree Index

- **Example:** **directory or root node (n= 3)**



秩为3的B⁺树的某个内部节点

Example 8.3.3 (cont.)

- number of entries: 1000000 (10^6)
- size of entry: 8 bytes
- size of disk page: 2k bytes
  - number of entries in each node(disk page)
    - 2k / 8 ≈ 250
  - total number of leaf node
    - 1000000 / 250 = 4000
  - total number of directory node:
    - 4000 / 250 = 16
  - root node

– average number of disk I/O
  - 3 (depth of the B-Tree) + 1 = 4

    (root node) + (directory node) + (leaf node)

8.3 The B-Tree Index

- **How to create an index**
  - **sort all entries on leaf nodes**
  - **create directory, and directory to directory**
- **Fanout f of B-Tree**
  - **the maximum number of entries on a B-Tree node**
- **Depth of B-Tree**
  - **If total number of rows is N, then**

    depth = log $_f$ (N)

8.3 The B-Tree Index

- **Dynamic changes in the B-tree**
  - **Figure 8.12 (pg. 353)**
    - **Insert following key values into B-Tree (n = 3)**

      7, 96, 41, 39, 88, 65, 55, 62

  - **Idea of [PCTFREE n]**

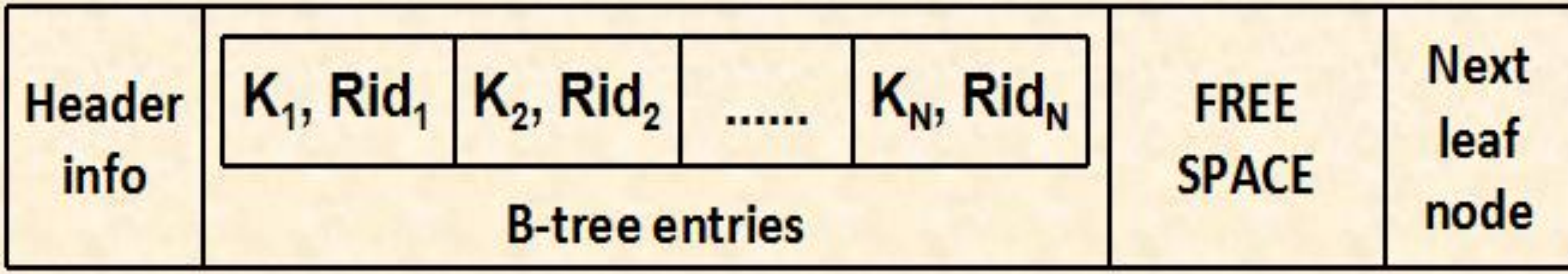
- Def 8.3.1: **Properties of the B-Tree (B^+ Tree)**
  1) Every node is disk-page sized and resides in a well-defined location on the disk.
  2) Nodes above the leaf level contain directory entries, with (n–1) separator keys and n disk pointers to lower-level B-tree nodes.
  3) Nodes at the leaf level contain entries with (keyval, ROWID) pairs pointing to individual rows indexed.
  4) All nodes below the root are at least half full with entry information.
     - This is not often enforced after multiple deletes.
  5) The root node contains at least two entries (one keyvalue).
     - except when only one row is indexed and the root is a leaf node.

8.3 The B-Tree Index

- **Algorithm on B-Tree**
  - **search**
  - **insert**
  - **delete**
    - **update = delete + insert**

P_1	K_1	P_2	K_2	......	P_m	K_m	P_{m+1}

❖ Figure Layout of a B-tree Leaf-Level Node with Unique key Values in table



❖ Figure 8.13 Layout of a B-tree Leaf-Level Node with Unique key Values in index, but maybe multi-rows in table

8.3 The B-Tree Index

- **The ORACLE Bitmap Index**
  - **A bitmap index uses ONE bitmap for each distinct keyval.**
    - **A bitmap *takes the place of a ROWID list*, specifying a set of rows.**
      - **one bit vs a ROWID (row)**
        - **1: the row have the keyval**
        - **0: the row don't have the keyval**
- **Example**
  - **Figure 8.18 (pg. 360)**

8.4 Clustered and Non-Clustered Indexes

- **The idea of a clustered index**
  - **The rows of the table are in the same order as the index entries — by keyvalue.**
    - **In most database products, default placemen of rows on data pages on disk is in order by load or by insertion (heap).**
      - **Figure 8.19 (pg. 362)**

- **Example 8.4.1**
  - **department store – branch stores**

Figure 8.19 - Non-Clustered

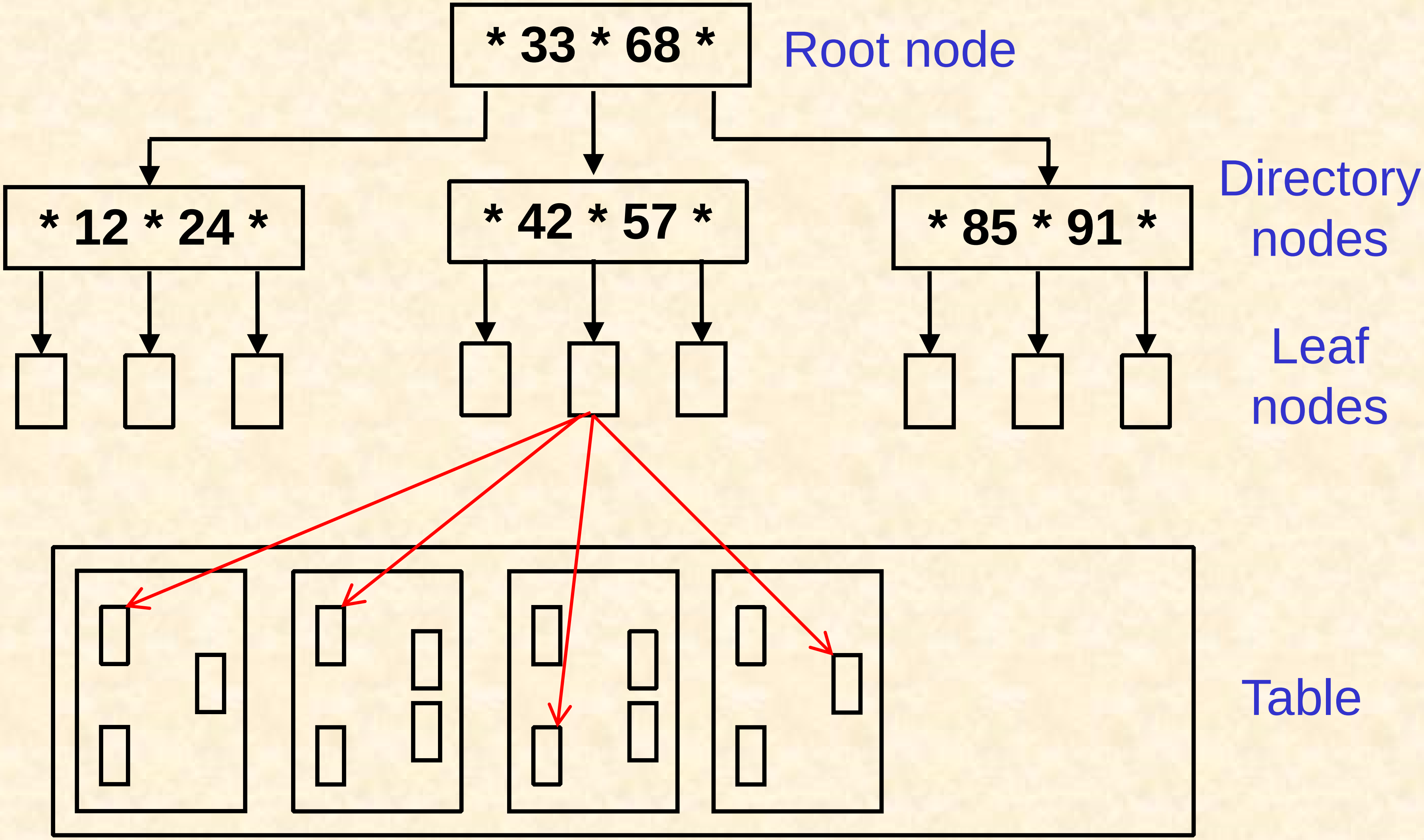


Figure 8.19 - Non-Clustered

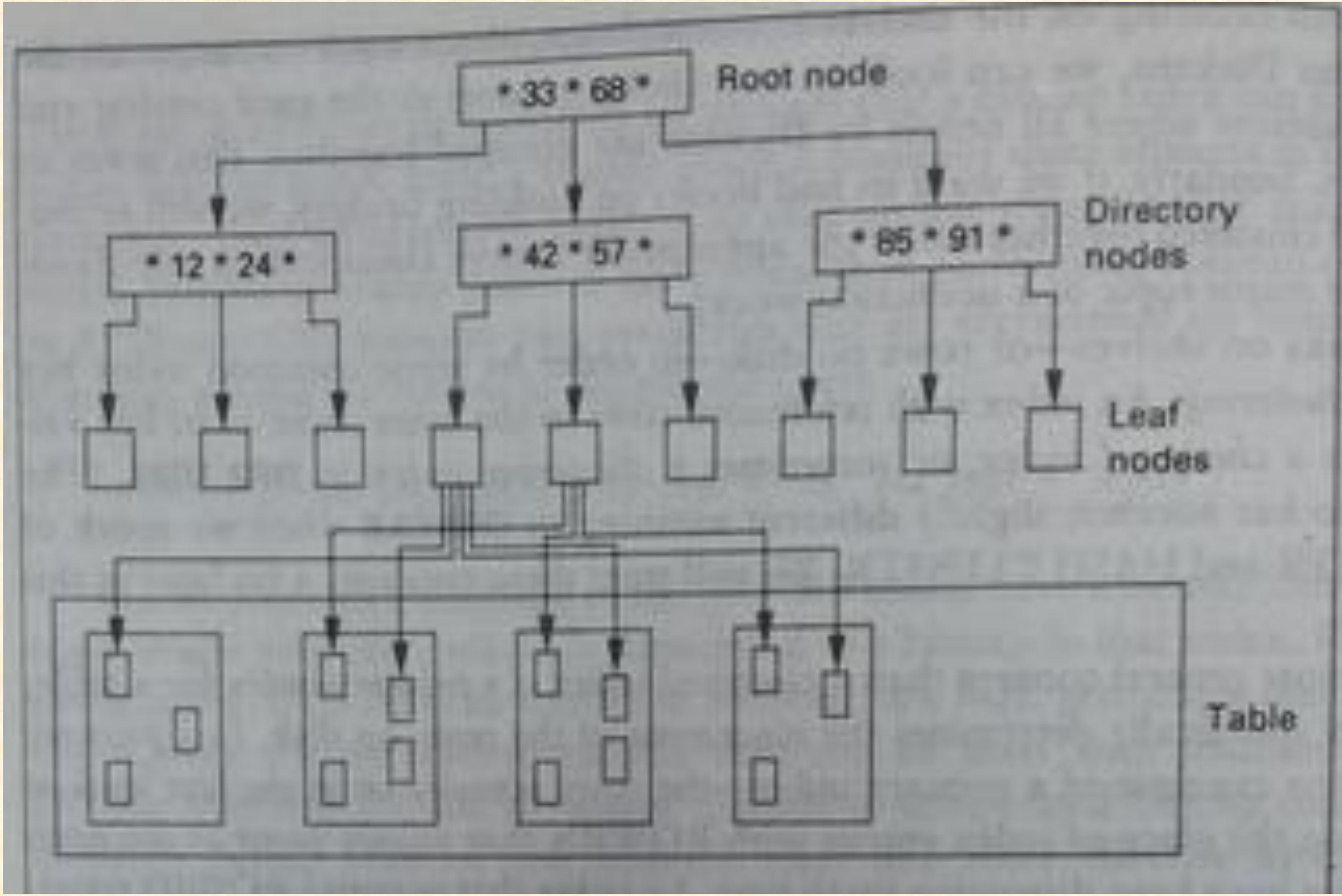
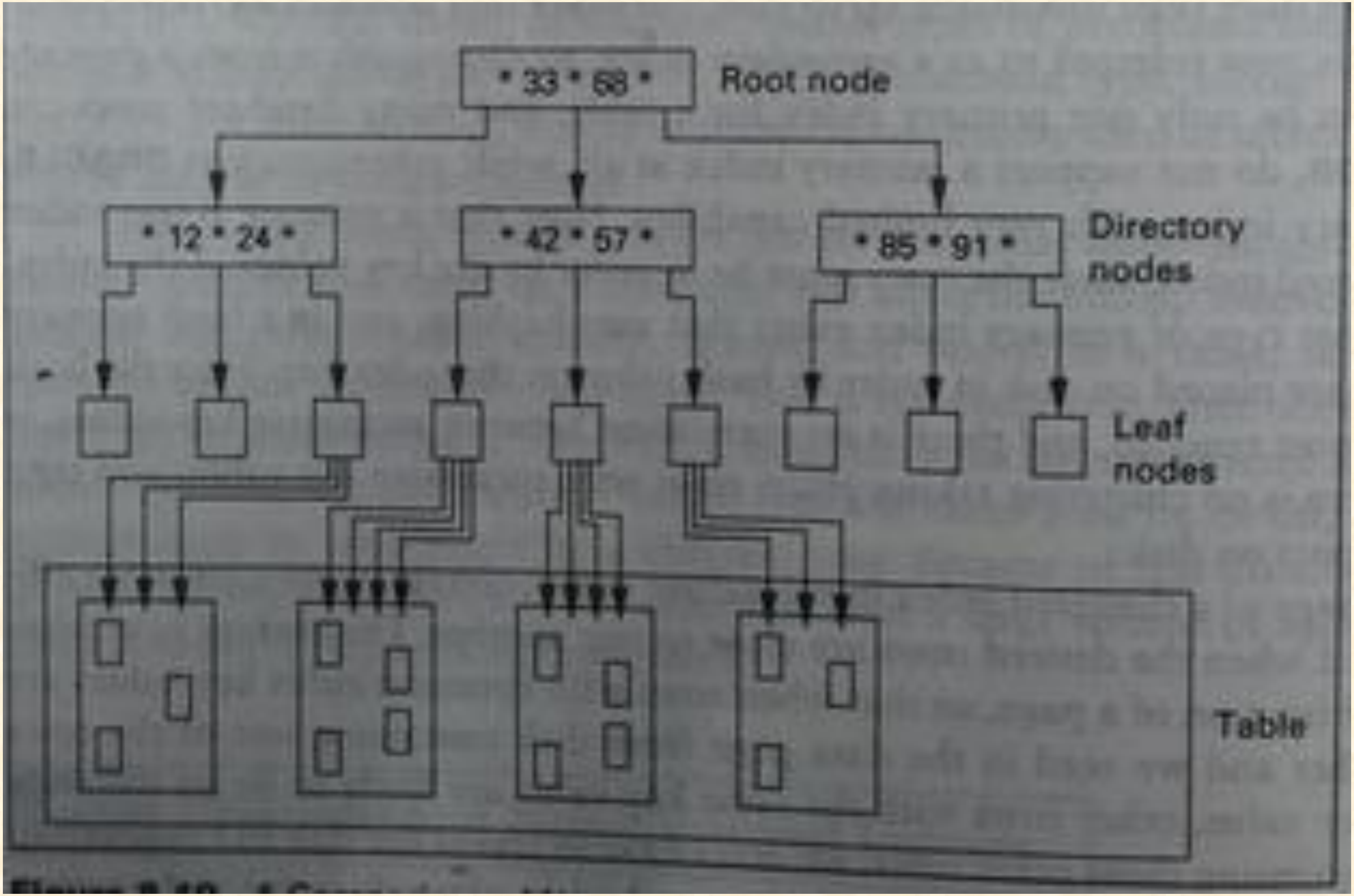


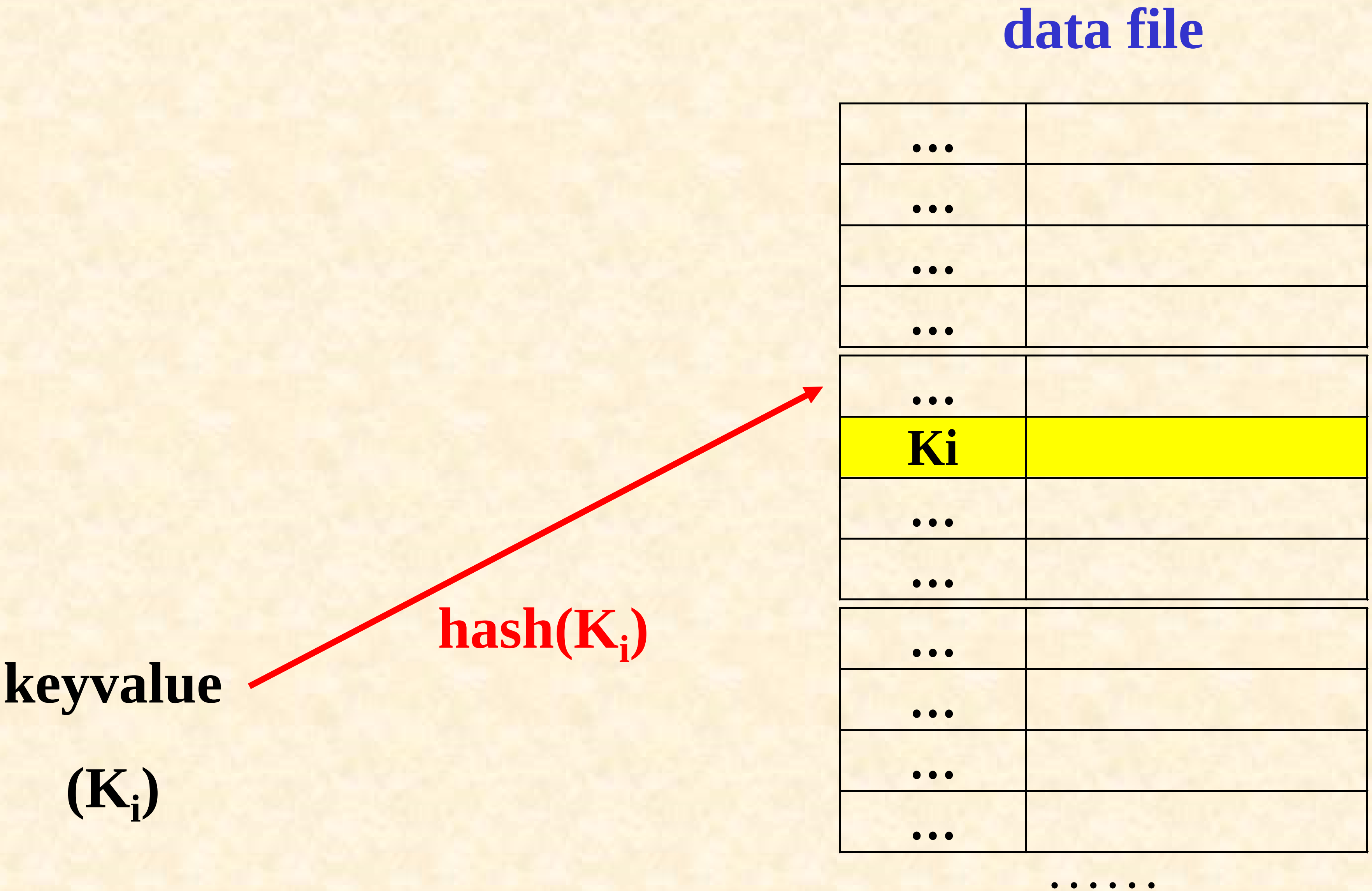
Figure 8.19 - Clustered



8.5 A Hash Primary Index

- **Idea**
  - **Rows in a table located in hash cluster are placed in pseudo-random data page slots using a hash function, and looked up the same way, often with only one I/O.**
  - **Also, no order by keyvalue**
    - **Successive keyvals are not close together, probably on entirely different pages**
      - **depends on hash function**

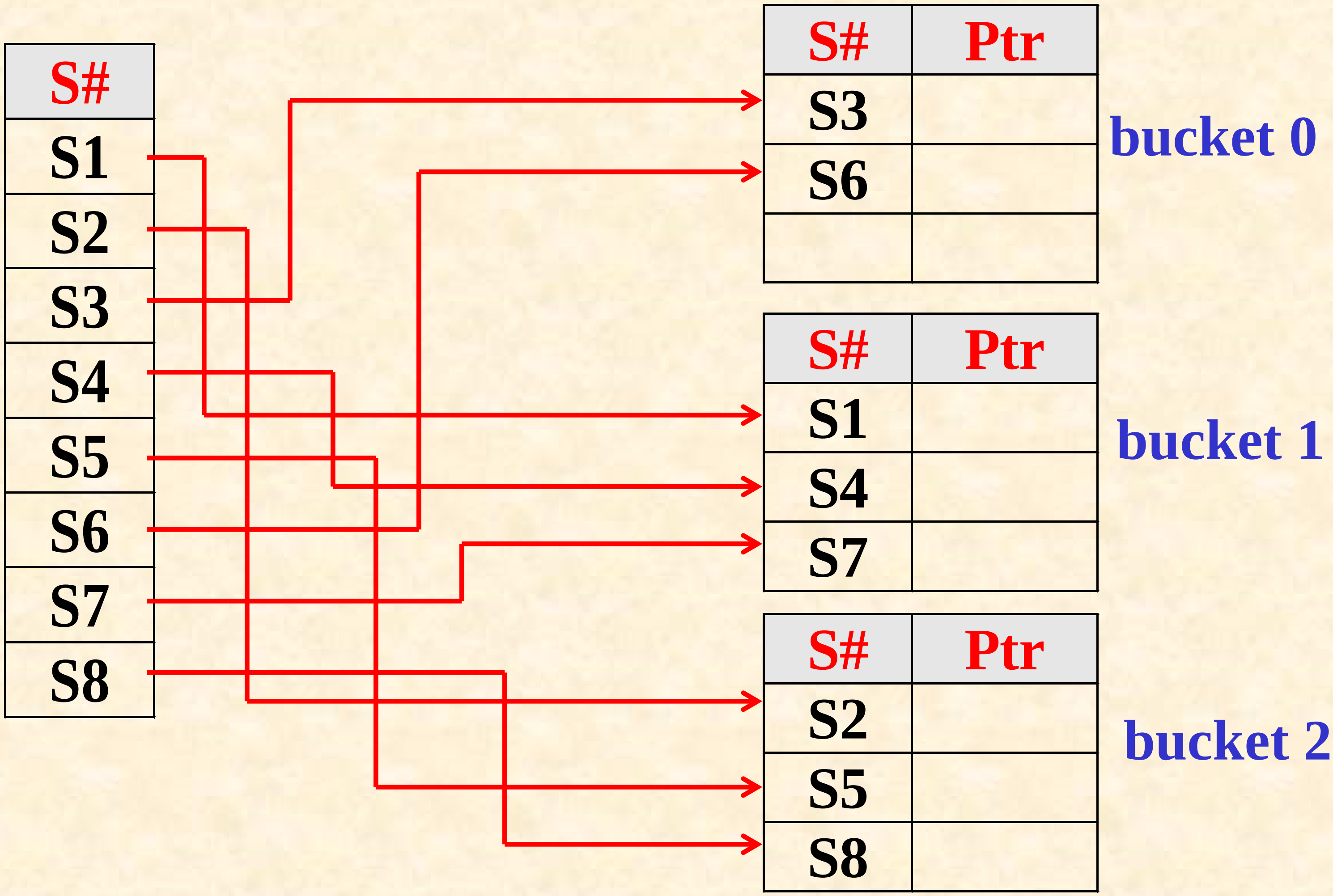
8.5 A Hash Primary Index



8.5 A Hash Primary Index

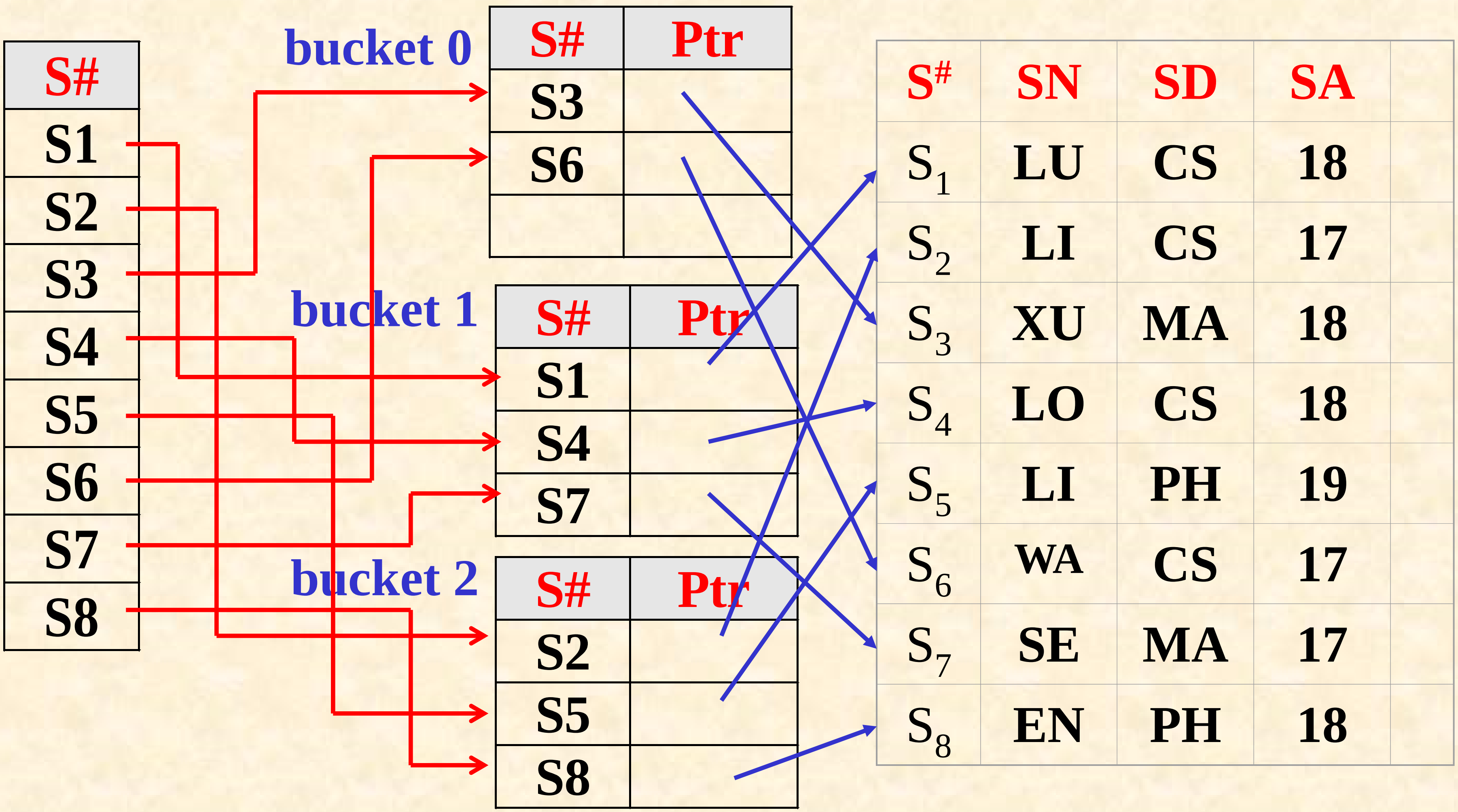
- **Tuning HASHKEYS and SIZE in a Hash Cluster**
  - **total number of slots: S**
  - **the number of slots on a disk block: B**
    - **total number of disk blocks: S/B**
  - **collision**
    - **two distinct key values hashed to same slot.**

8.5 A Hash Primary Index



Schematic Picture of a Hash (mod 3)

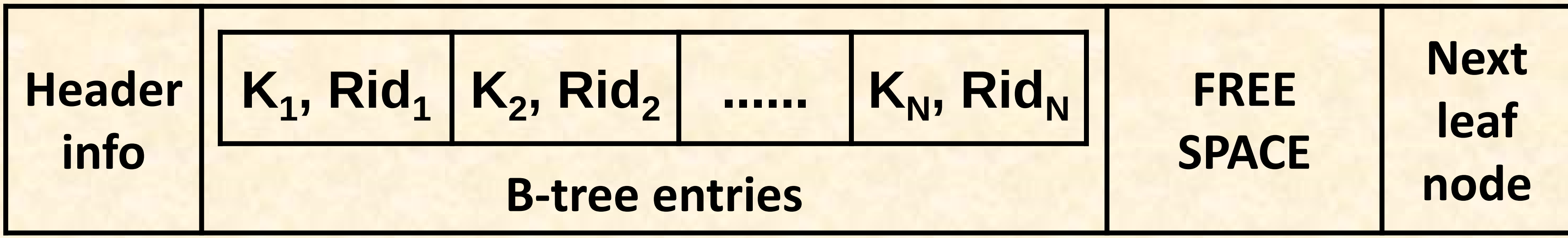
8.5 A Hash Primary Index



Schematic Picture of a Hash Index

Figure 8.13 Layout of a B-tree Leaf-Level Node with Unique key Values

Figure 8.13 Layout of a B-tree Leaf-Level Node with Unique key Values



Header info
K_1, Rid_1
K_2, Rid_2
……
K_N, Rid_N
B-tree entries
FREE SPACE
Next leaf node