Indexing Structures for Files

Fundamentals of database systems

Outline

- Types of Single-level Ordered Indexes
  - Primary Indexes
  - Clustering Indexes
  - Secondary Indexes
- Multilevel Indexes
- Dynamic Multilevel Indexes Using B-Trees and B+-Trees
- Indexes on Multiple Keys

Indexes as Access Paths

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries **<field value, pointer to record>**, which is ordered by field value
- The index is called an *access path* on the field.

Indexes as Access Paths (contd.)

- The index file usually occupies considerably **less** disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- Indexes can also be characterized as dense or sparse.
  - A **dense index** has an index entry for *every search key value* (and hence every record) in the data file.
  - A **sparse** (or **nondense**) **index**, on the other hand, has index entries for only some of the search values

Indexes as Access Paths (contd.)

<u>Example:</u> Given the following data file:

EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)

<u>Suppose that:</u>

- record size R=150 bytes
- block size B=512 bytes
- r = 30000 records

<u>Then, we get:</u>

- blocking factor Bfr= B div R= 512 div 150= 3 records/block
- number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks

Indexes as Access Paths (contd.)

For an index on the SSN field, assume the field size V_{SSN}= 9 bytes, assume the record pointer size P_R=7 bytes. Then:

- index entry size $R_I=(V_{SSN}+ P_R)=(9+7)=16$ bytes
- index blocking factor Bfr_I= B div R_I= 512 div 16= 32 entries/block
- number of index blocks b= (r/ Bfr_I)= (30000/32)= 938 blocks
- binary search needs $\log_2 bI$= $\log_2 938$= 10 block accesses

This is compared to an average linear search cost of:

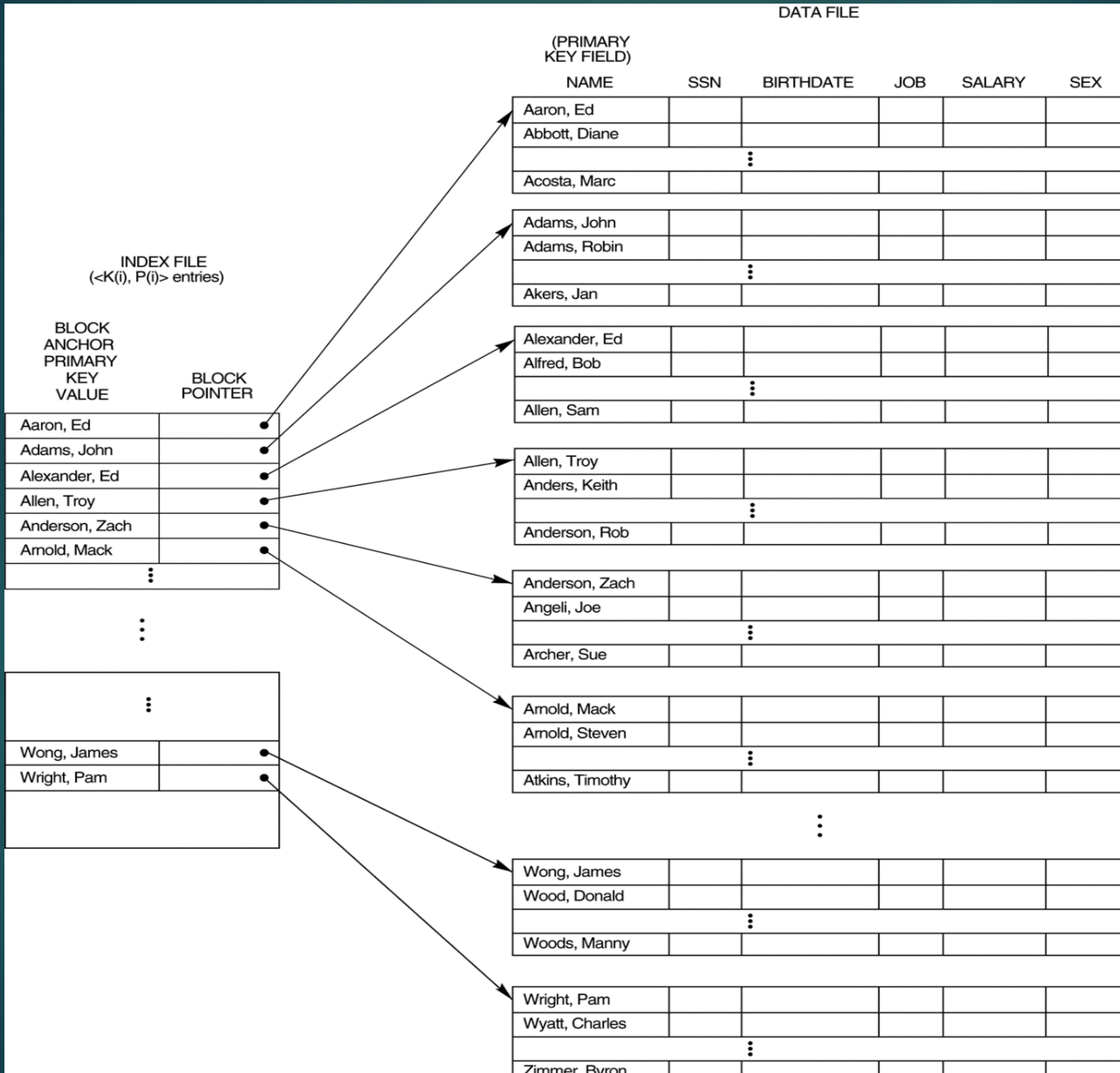
- (b/2)= 30000/2= 15000 block accesses

If the file records are ordered, the binary search cost would be:

- $\log_2 b$= $\log_2 30000$= 15 block accesses

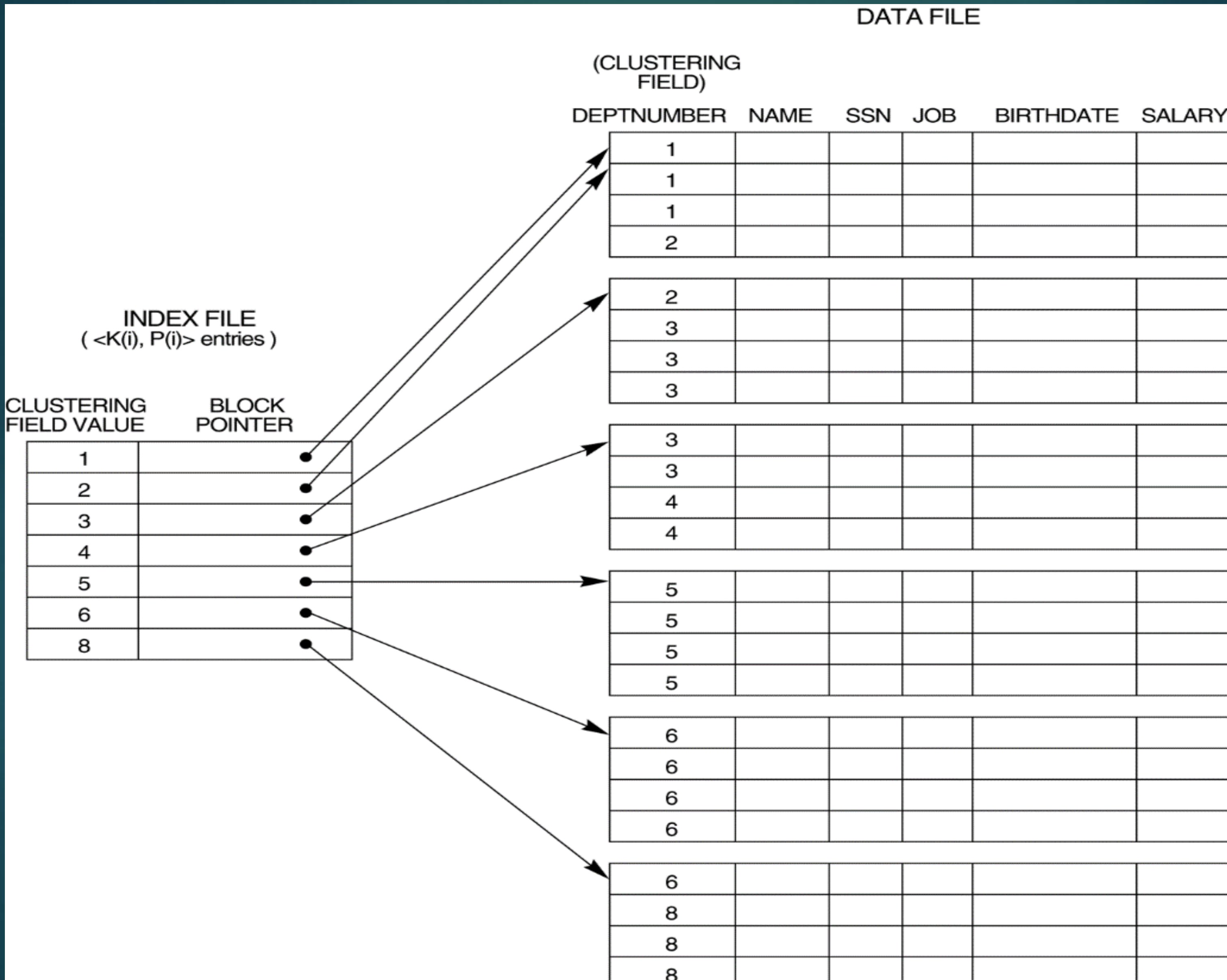
Types of Single-Level Indexes

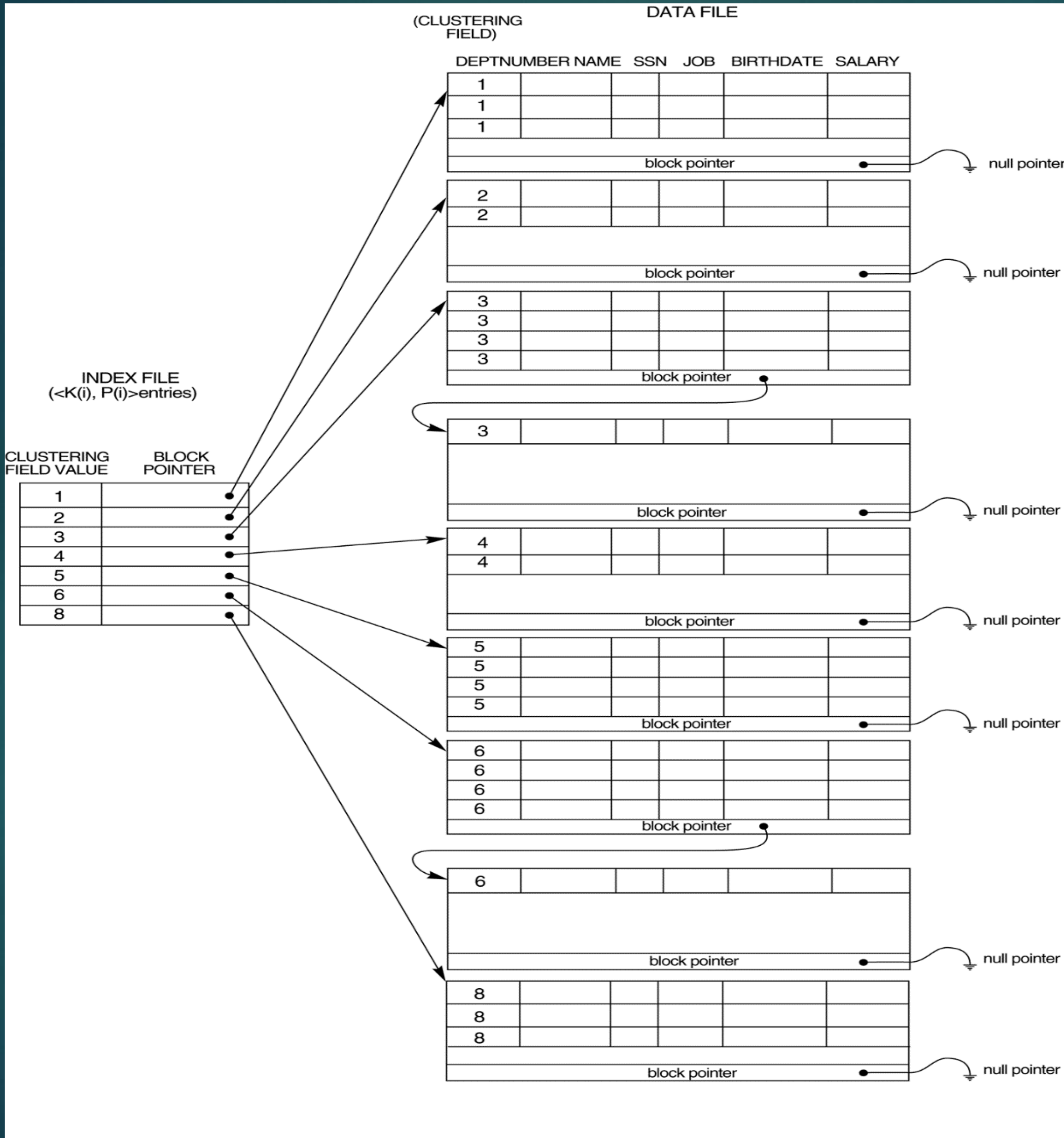
- **Primary Index**
  - Defined on an ordered data file
  - The data file is ordered on a *key field*
  - Includes one index entry *for each block* in the data file; the index entry has the **key field value** for the *first record* in the block, which is called the *block anchor*
  - A similar scheme can use the *last record* in a block.
  - A primary index is a **nondense** (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.


DATA FILE
(PRIMARY KEY FIELD)
NAME
SSN
BIRTHDATE
JOB
SALARY
SEX
Aaron, Ed
Abbott, Diane
Acosta, Marc
Adams, John
Adams, Robin
Akers, Jan
Alexander, Ed
Alfred, Bob
Allen, Sam
Allen, Troy
Anders, Keith
Anderson, Rob
Anderson, Zach
Angeli, Joe
Archer, Sue
Arnold, Mack
Arnold, Steven
Atkins, Timothy
Wong, James
Wood, Donald
Woods, Manny
Wright, Pam
Wyatt, Charles
Zimmer, Byron
INDEX FILE
(<K(i), P(i)> entries)
BLOCK ANCHOR PRIMARY KEY VALUE
BLOCK POINTER
Aaron, Ed
Adams, John
Alexander, Ed
Allen, Troy
Anderson, Zach
Arnold, Mack
Wong, James
Wright, Pam

Types of Single-Level Indexes

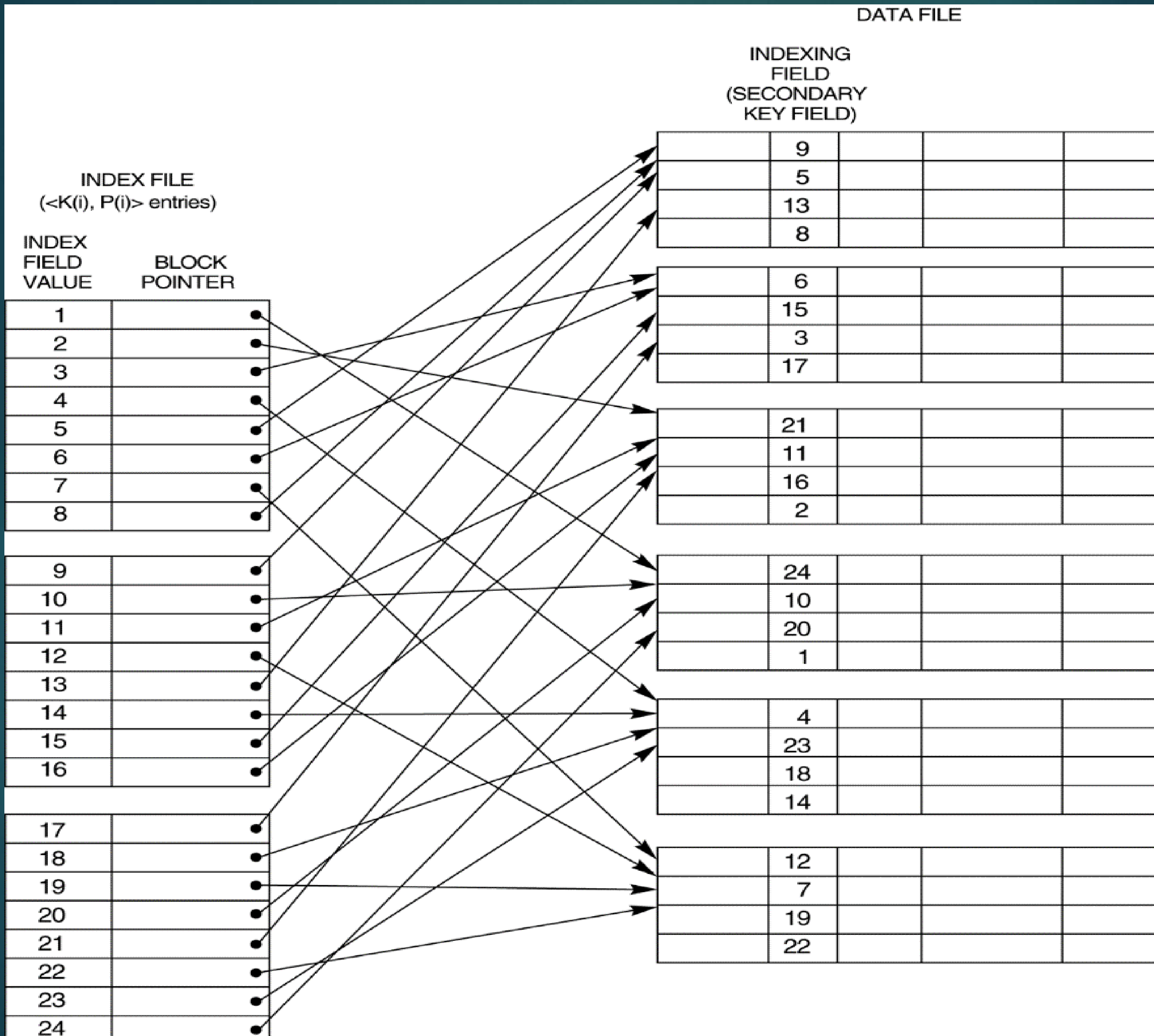
- **Clustering Index**
  - Defined on an ordered data file
  - The data file is ordered on a ***non-key*** *field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
  - Includes **one index entry *for each distinct value*** of the field; the index entry points to the first data block that contains records with that field value.
  - It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.

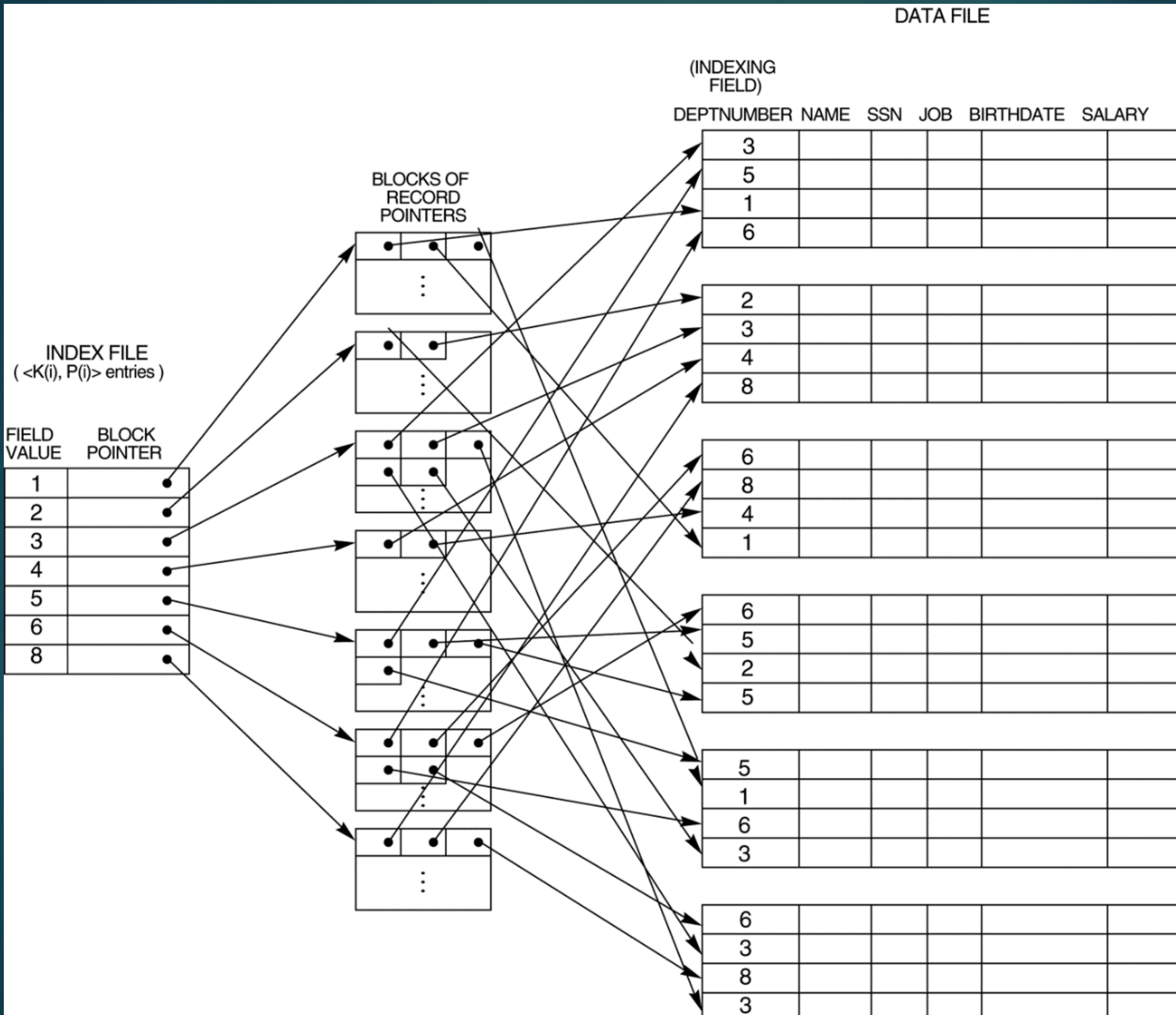

DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER
NAME
SSN
JOB
BIRTHDATE
SALARY
1
1
1
2
2
3
3
3
3
3
4
4
5
5
5
5
6
6
6
6
6
8
8
8
INDEX FILE
(<K(i), P(i)> entries)
CLUSTERING FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8


DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER NAME SSN JOB BIRTHDATE SALARY
1
1
1
block pointer
null pointer
2
2
block pointer
null pointer
3
3
3
3
block pointer
3
block pointer
null pointer
4
4
block pointer
null pointer
5
5
5
5
block pointer
null pointer
6
6
6
6
block pointer
6
block pointer
null pointer
8
8
8
block pointer
null pointer
INDEX FILE
(<K(i), P(i)>entries)
CLUSTERING FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8

Types of Single-Level Indexes

- **Secondary Index**
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a **candidate key** and has a unique value in every record, or a nonkey with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some *nonordering field* of the data file that is an *indexing field*.
    - The second field is either a *block* pointer or a *record* pointer. There can be *many* secondary indexes (and hence, indexing fields) for the same file.
  - Includes ***one entry for each record*** in the data file; hence, it is a ***dense*** *index*


DATA FILE
INDEXING FIELD (SECONDARY KEY FIELD)
INDEX FILE
(<K(i), P(i)> entries)
INDEX FIELD VALUE
BLOCK POINTER
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
9 5 13 8
6 15 3 17
21 11 16 2
24 10 20 1
4 23 18 14
12 7 19 22

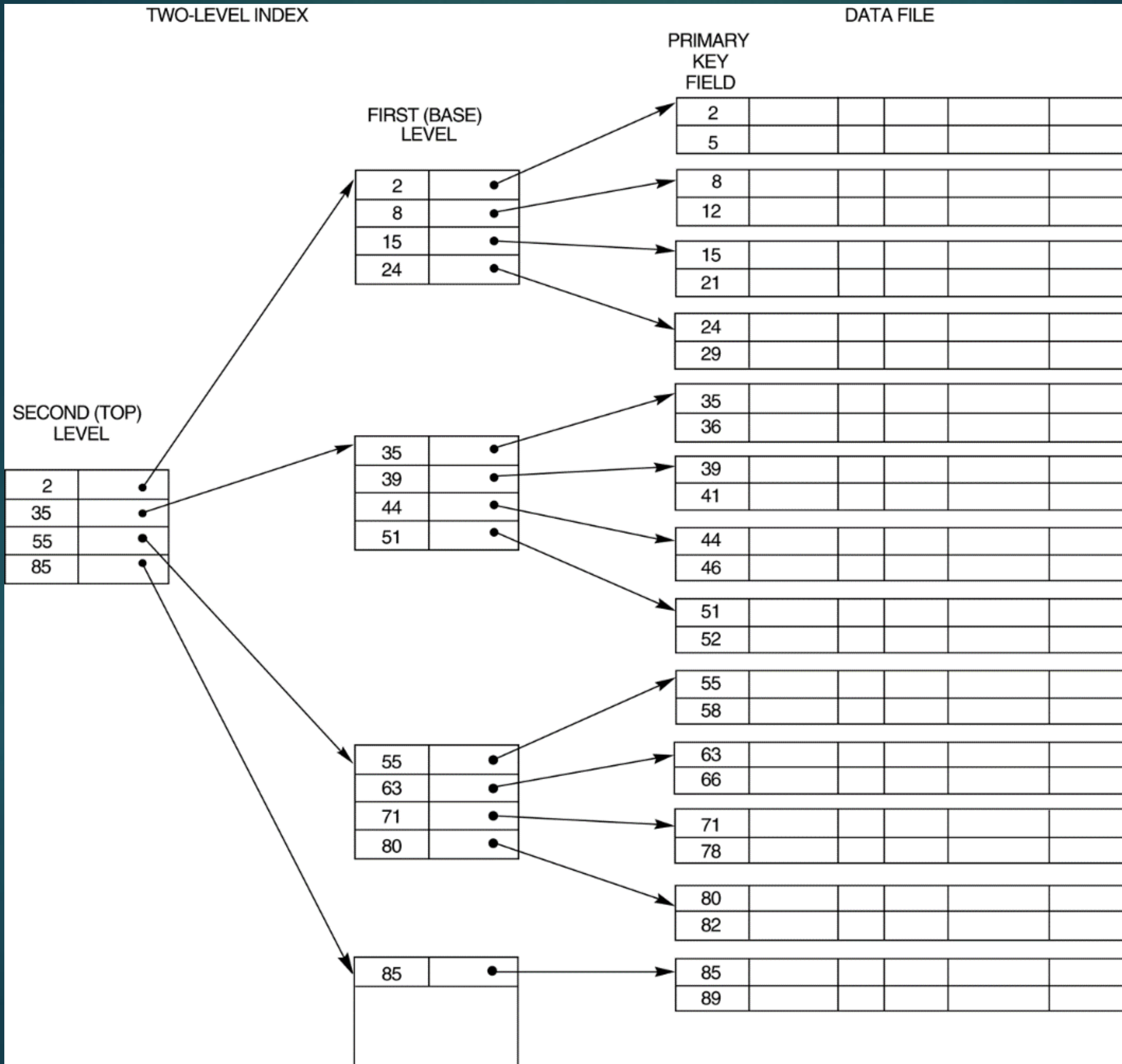

DATA FILE
(INDEXING FIELD)
DEPTNUMBER NAME SSN JOB BIRTHDATE SALARY
3
5
1
6
2
3
4
8
6
8
4
1
6
5
2
5
5
1
6
3
6
3
8
3
BLOCKS OF RECORD POINTERS
INDEX FILE
(<K(i), P(i)> entries)
FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8

Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself* ; in this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index*.

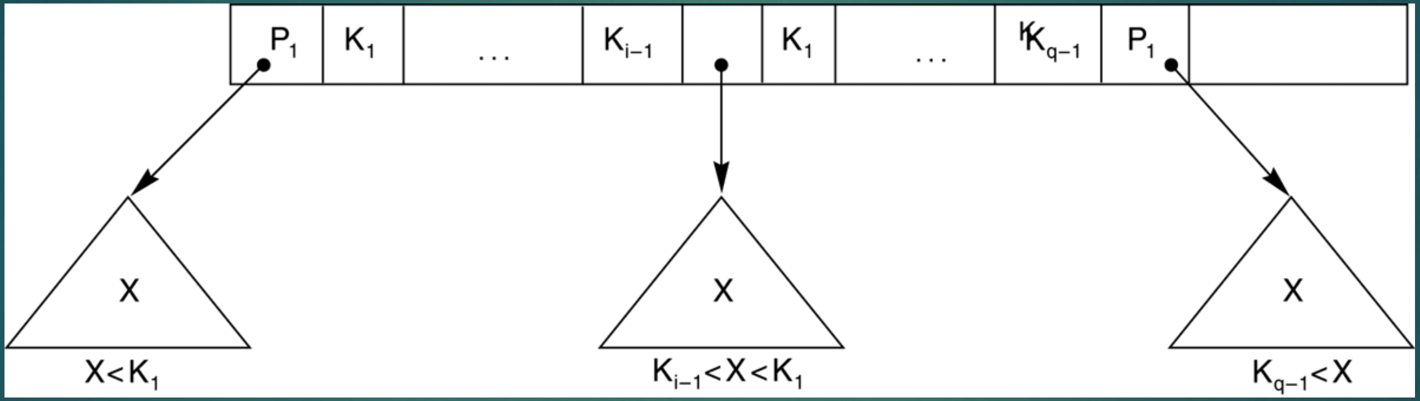
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block

- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

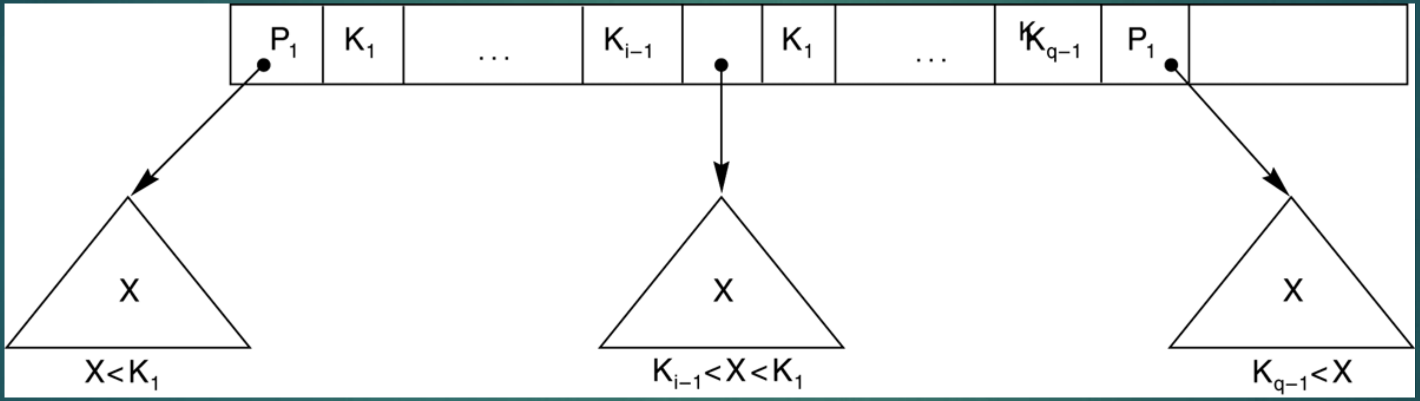

TWO-LEVEL INDEX
DATA FILE
PRIMARY KEY FIELD
FIRST (BASE) LEVEL
SECOND (TOP) LEVEL
2
35
55
85
2
8
15
24
35
39
44
51
55
63
71
80
85
2
5
8
12
15
21
24
29
35
36
39
41
44
46
51
52
55
58
63
66
71
78
80
82
85
89

Multi-Level Indexes

- Such a multi-level index is a form of *search tree* ; however, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file.*



A node in a search tree with pointers to subtrees below it.



Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- Because of the insertion and deletion problem, most multi-level indexes use B-tree or B+-tree data structures, which leave space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full

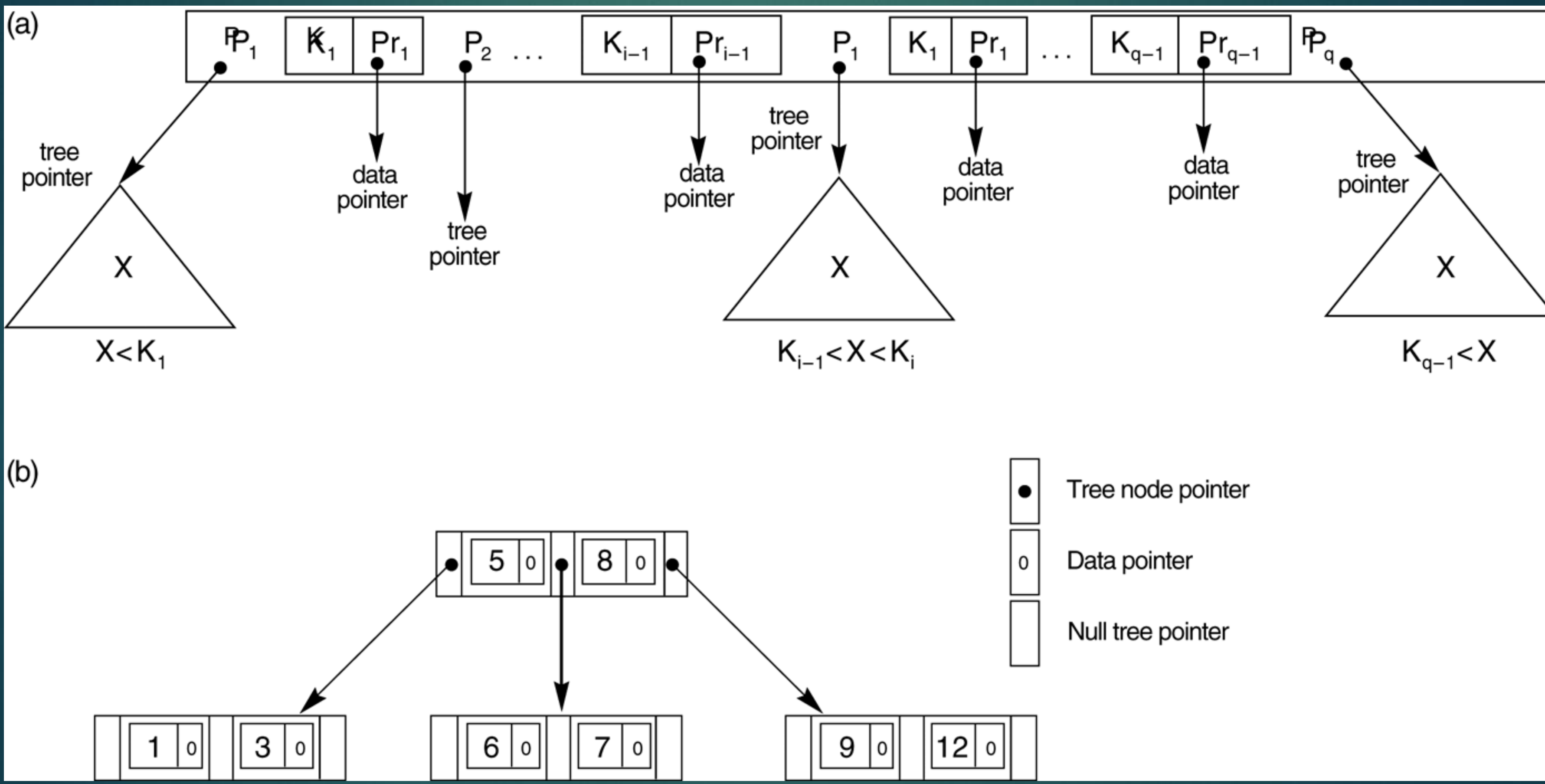
Dynamic Multilevel Indexes Using B-Trees and B+-Trees (contd.)

- An insertion into a node that is not full is quite efficient; if a node is full the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- A deletion is quite efficient if a node does not become less than half full
- If a deletion causes a node to become less than half full, it must be merged with neighboring nodes

Difference between B-tree and B+-tree

- In a B-tree, pointers to data records exist at all levels of the tree
- In a B+-tree, all pointers to data records exists at the leaf-level nodes
- A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree

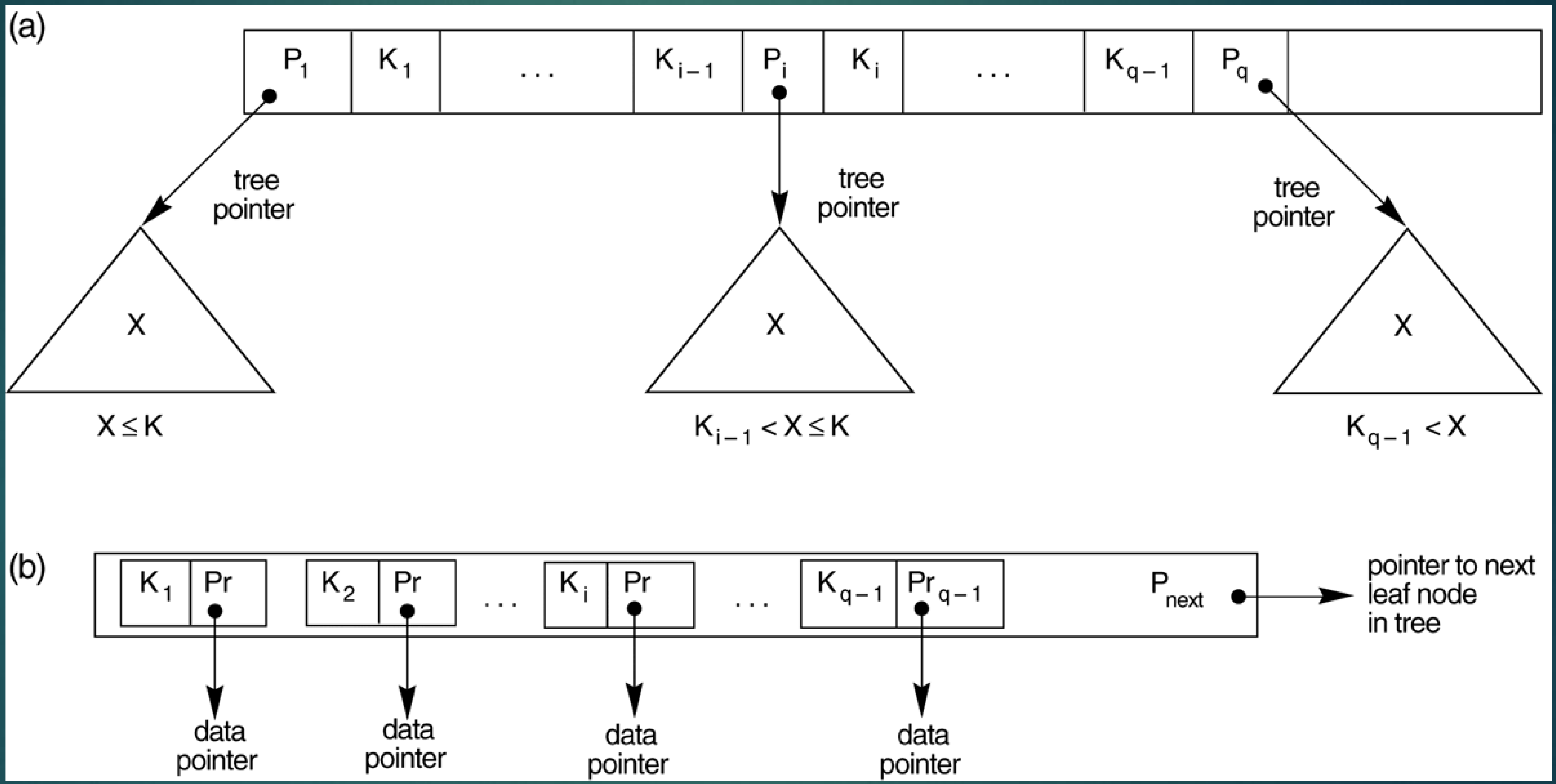
B-tree structures.
(a) A node in a B-tree with q – 1 search values.
(b) A B-tree of order p = 3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



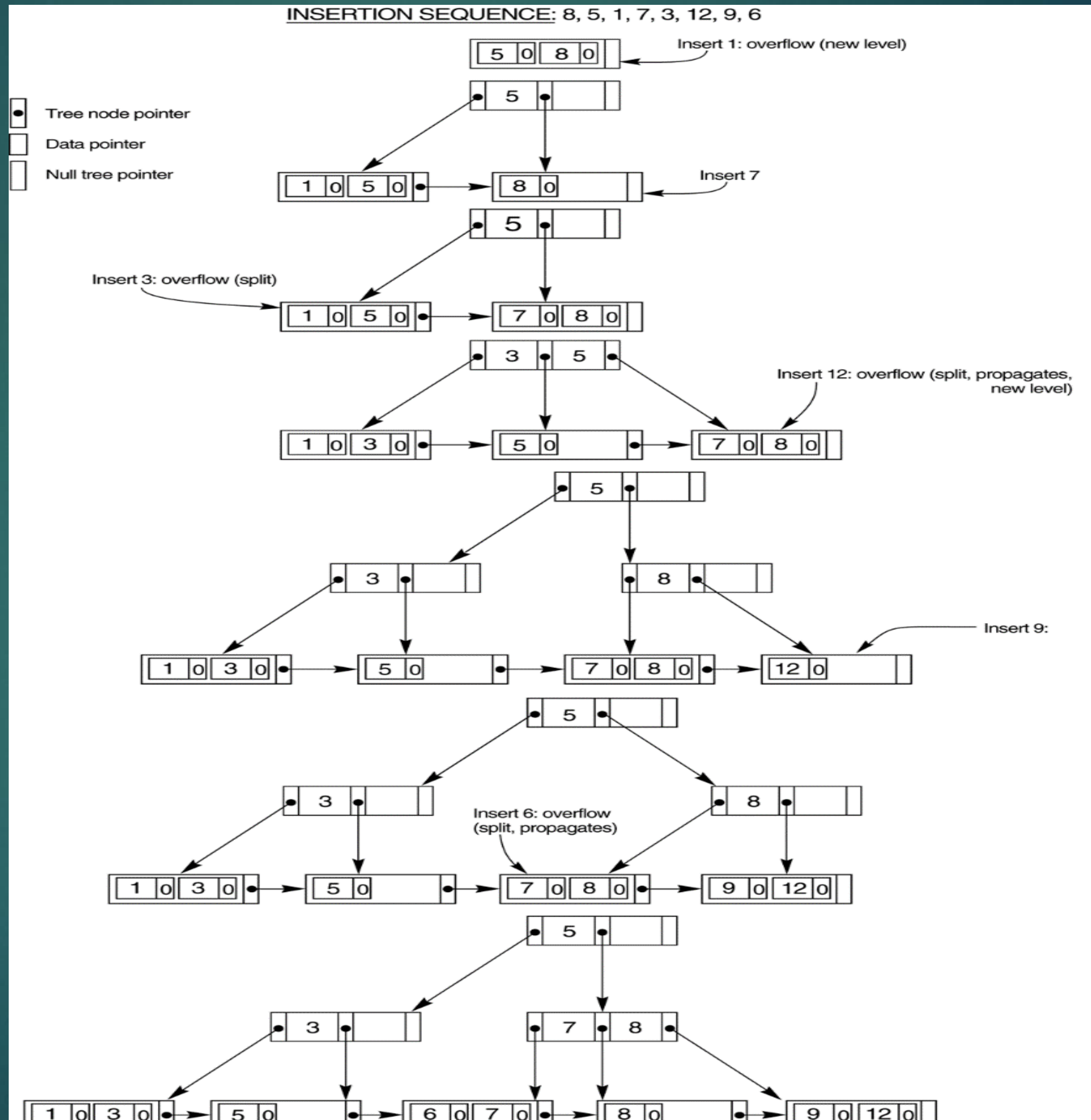
The nodes of a B+-tree.

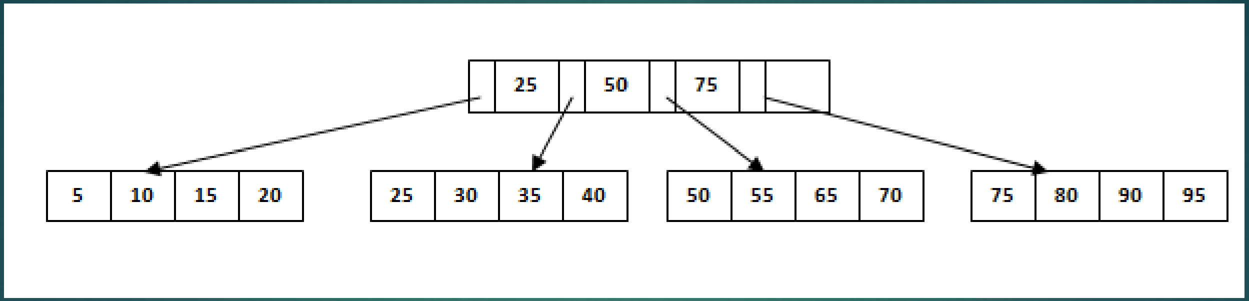
(a) Internal node of a B+-tree with q –1 search values.

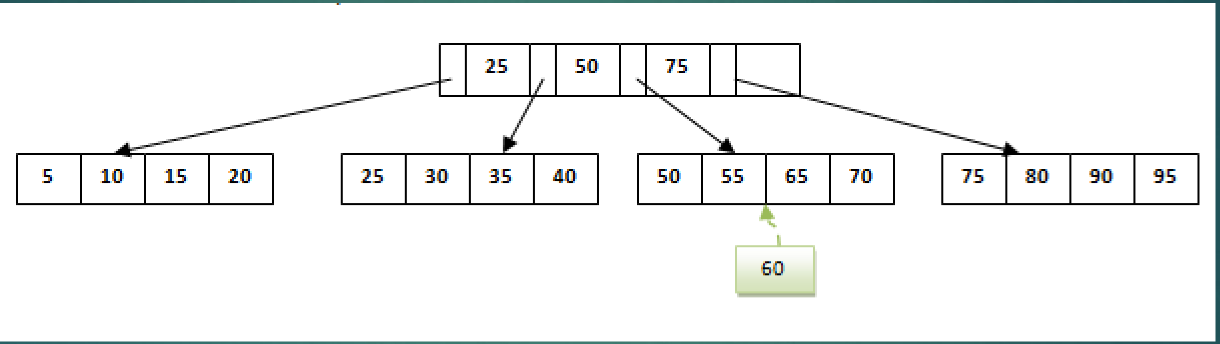
(b) Leaf node of a B+-tree with q – 1 search values and q – 1 data pointers.

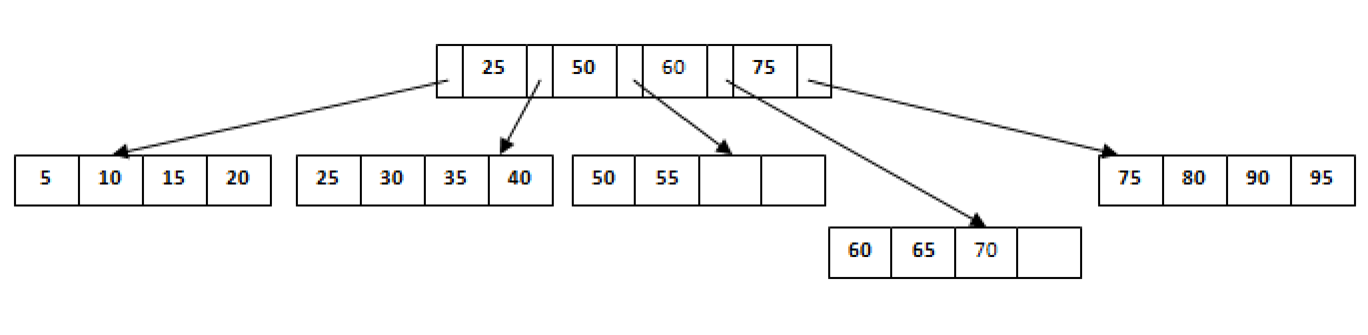


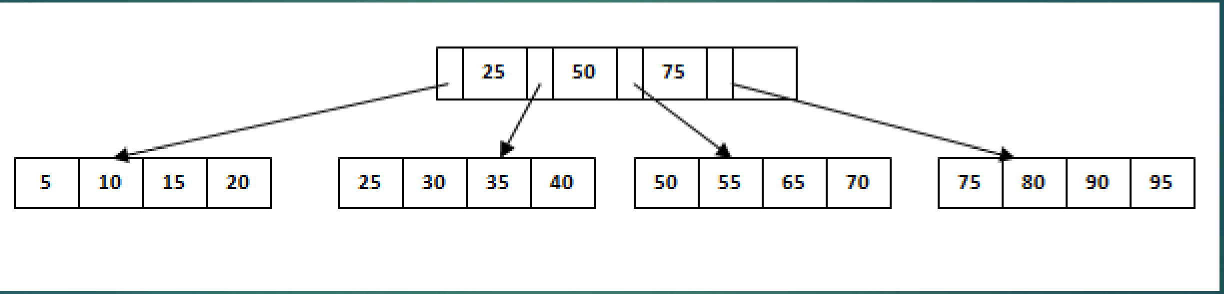
An example of insertion in a B+-tree with q = 3 and p_{leaf} = 2.

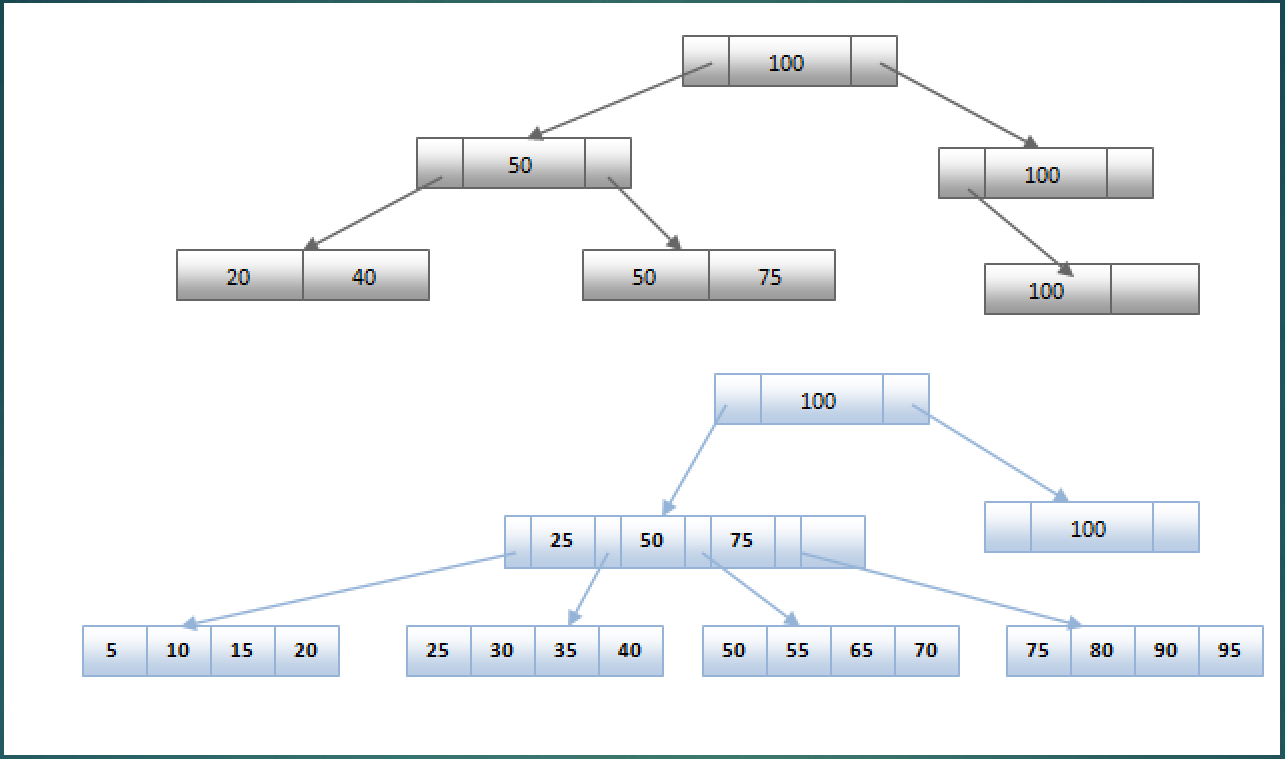



25
50
75
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95


25
50
75
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95
60


25 50 60 75
5 10 15 20
25 30 35 40
50 55
60 65 70
75 80 90 95


25 50 75
5 10 15 20
25 30 35 40
50 55 65 70
75 80 90 95


100
50
100
20
40
50
75
100
100
25
50
75
100
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95

An example of deletion from a B+-tree.