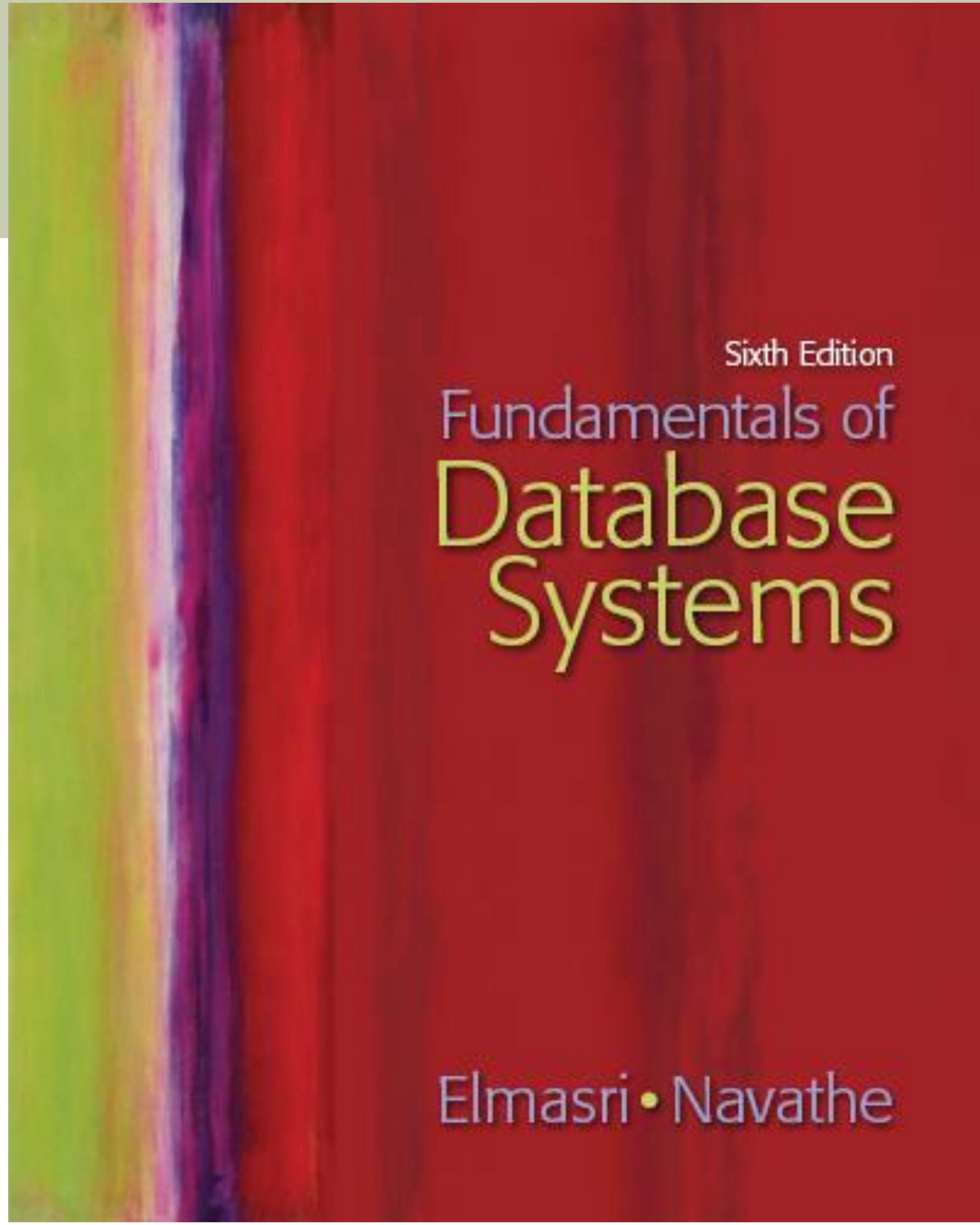
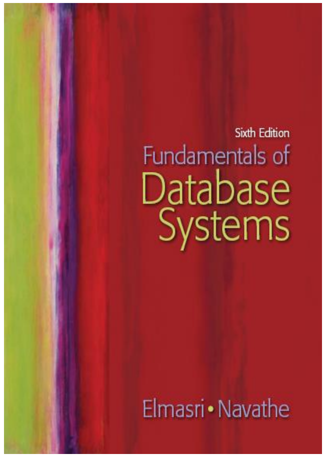

Sixth Edition
Fundamentals of
Database
Systems
Elmasri • Navathe

Chapter 17-1

Indexing Structures for Files



Chapter 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.

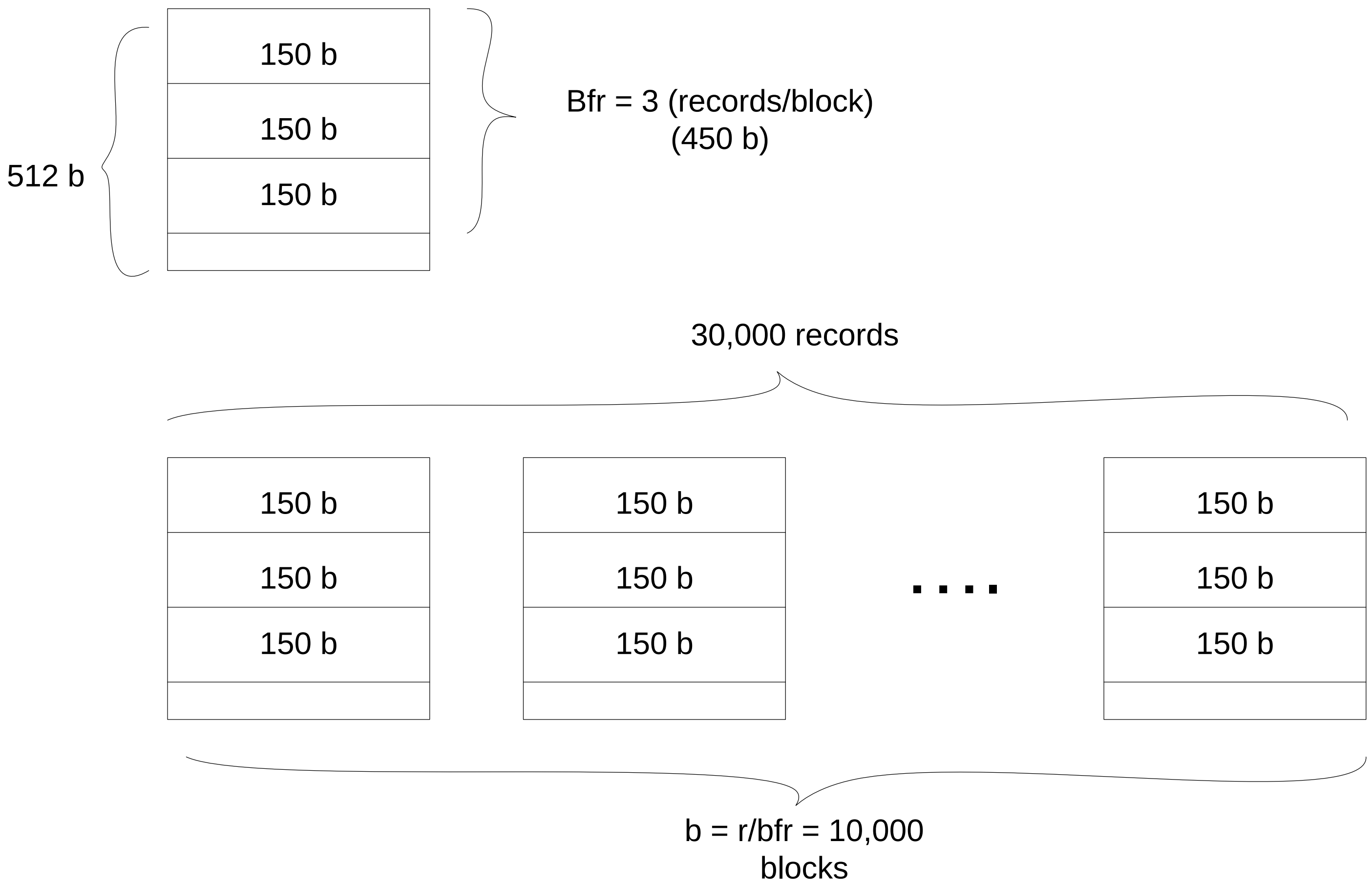
Indexes as Access Paths (contd.)

- 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

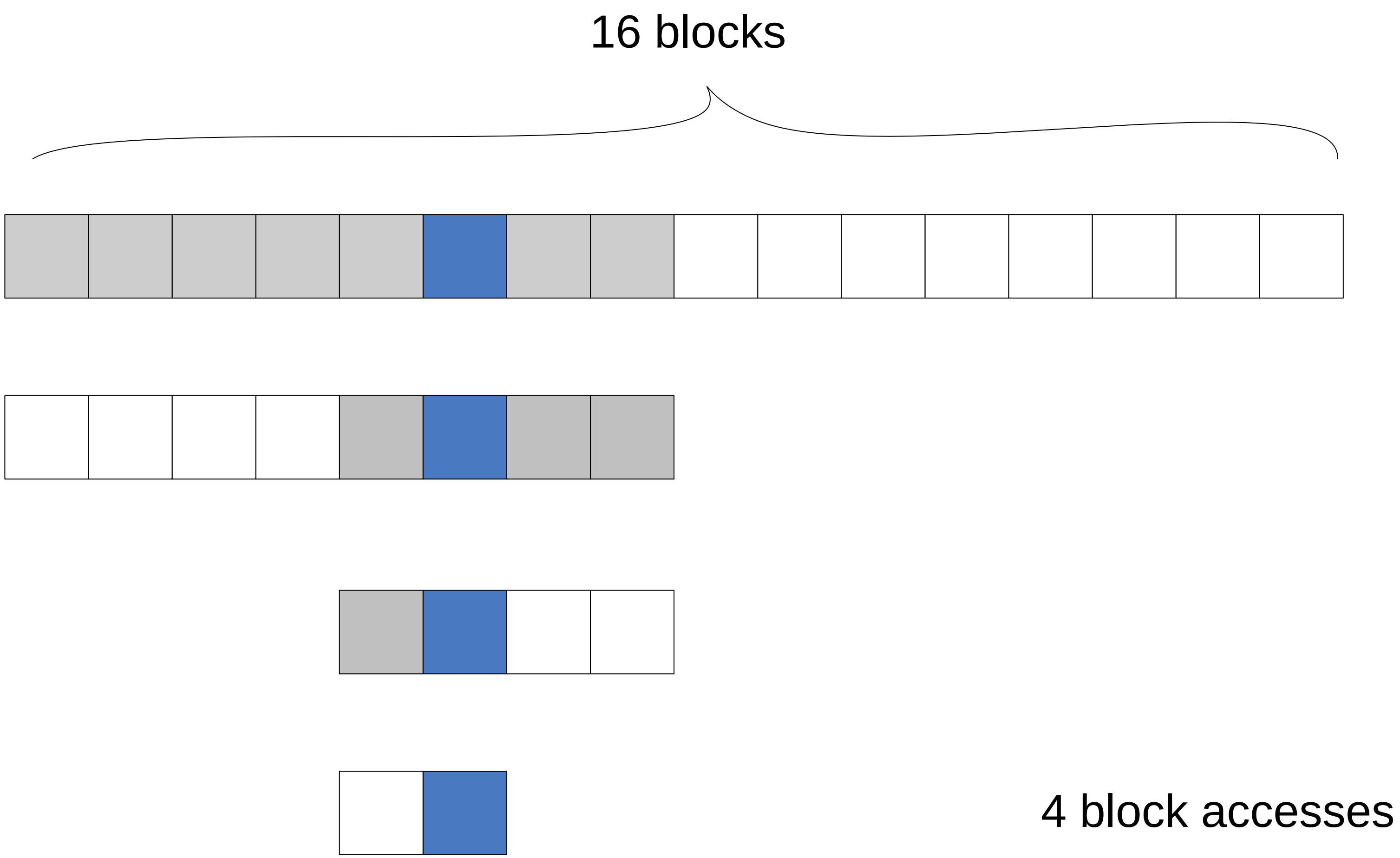
Indexes as Access Paths (contd.)

- Example: Given the following data file
  - EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)
- Suppose that:
  - record size R=150 bytes block size B=512 bytes   r=30000 records
- Then, we get:
  - blocking factor Bfr= $\lfloor B/R \rfloor = \lfloor 512/150 \rfloor$ = 3 records/block
  - number of file blocks b= $\lceil r/Bfr \rceil = \lceil 30000/3 \rceil$ = 10000 blocks
- 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 $Bfr_I = \lfloor B/R_I \rfloor = \lfloor 512/16 \rfloor$ = 32 entries/block
  - number of index blocks $b_I = \lceil r/ Bfr_I \rceil = \lceil 30000/32 \rceil$ = 938 blocks
  - binary search needs $\log_2 b_I = \log_2 938$= 10 block accesses
  - This is compared to an average linear search cost of:
    - $\lceil b/2 \rceil$ = 10000/2= 5000 block accesses
  - If the file records are ordered, the binary search cost would be:
    - $\lceil \log_2 b \rceil = \log_2 10000$= 14 block accesses

Indexes as Access Paths (contd.)



Indexes as Access Paths (contd.)



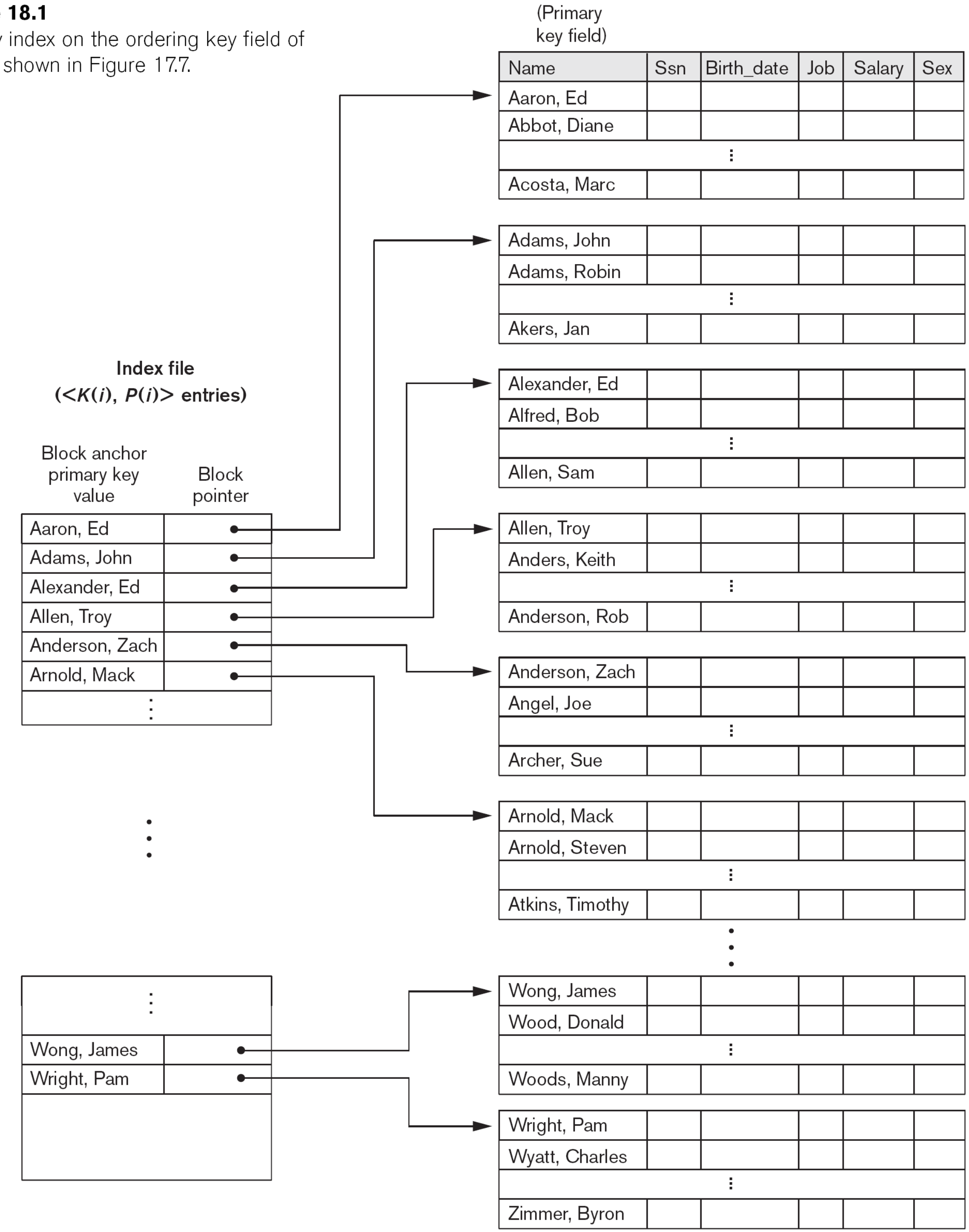
Types of Single-Level Indexes

- 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.

Primary index on the ordering key field

- FIGURE 17.1 Primary index on the ordering key field of the file shown in Figure 16.7.

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



Example 1.

- Suppose that:
  - record size R=100 bytes   block size B=1024 bytes r=30000 records
  - file records are fixed and unspanned
- Then, we get:
  - **blocking factor** Bfr= $\lfloor B/R \rfloor = \lfloor 1024/100 \rfloor$ = 10 records/block
  - **number of file blocks** b= $\lceil r/Bfr \rceil$= (30000/10)= 3000 blocks
  - binary search on data file
    - $\lceil \log_2 b \rceil = \lceil \log_2 3000 \rceil$= **12 block accesses**
- For an index on the SSN field, assume the field size V_{SSN}=9 bytes, assume the record pointer size P_R=6 bytes. Then:
  - index entry size $R_I=(V_{SSN}+P_R)$=(9+6)=15 bytes
  - index blocking factor $Bfr_I = \lfloor B/R_I \rfloor = \lfloor 1024/15 \rfloor$ = 68 entries/block
  - the total number of index entries r_I=3000 (why?)
  - number of index blocks $b_I = \lceil r_I / Bfr_I \rceil = \lceil 3000/68 \rceil$= 45 blocks
  - **binary search** needs $\lceil \log_2 b_I \rceil = \lceil \log_2 45 \rceil$ = **6 block accesses**
  - Total we need **7 = 6 + 1 block access** (1 for data file)

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.

A Clustering Index Example

- FIGURE 17.2
  A clustering index on the DEPTNUMBER ordering non-key field of an EMPLOYEE file.

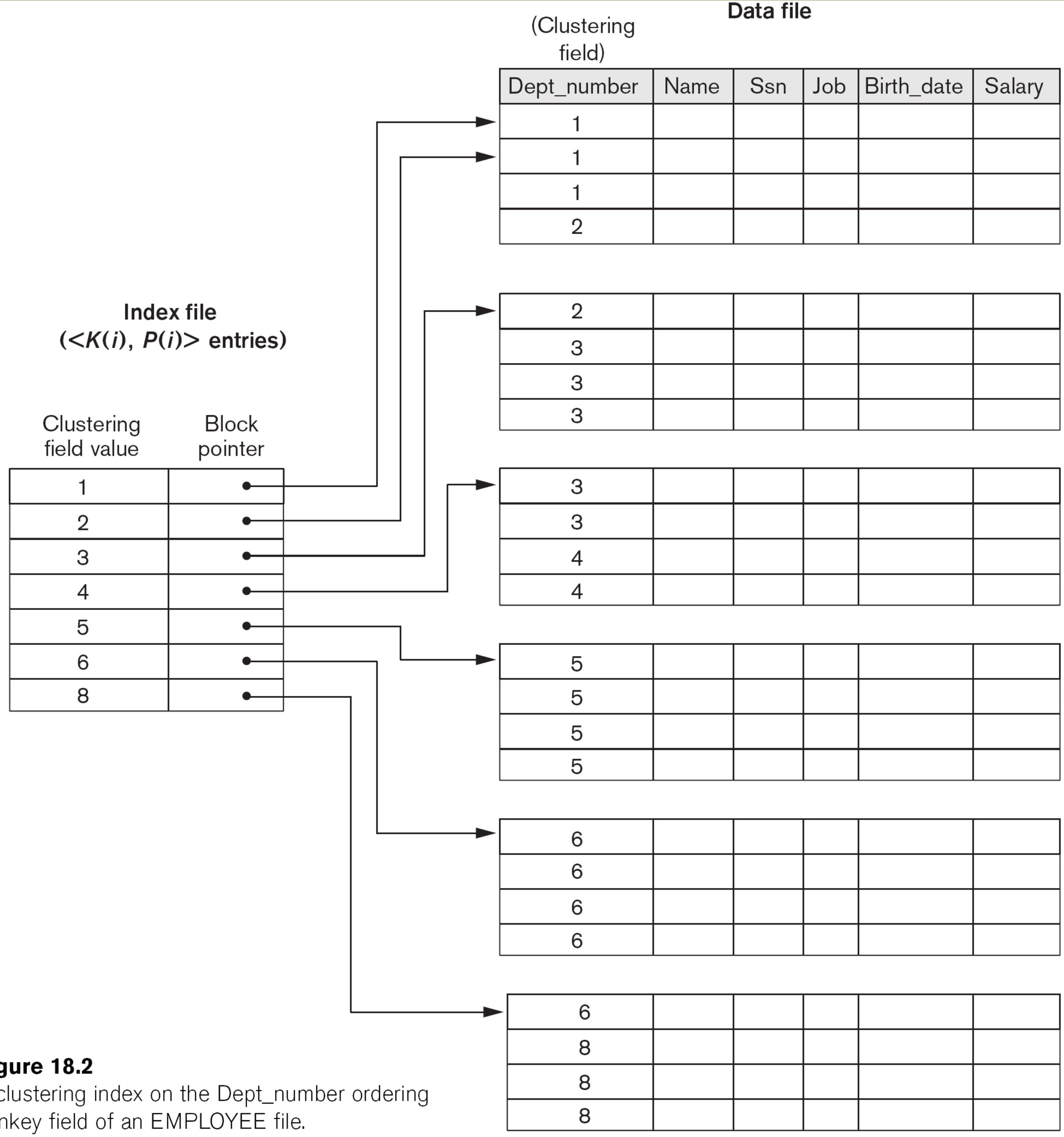
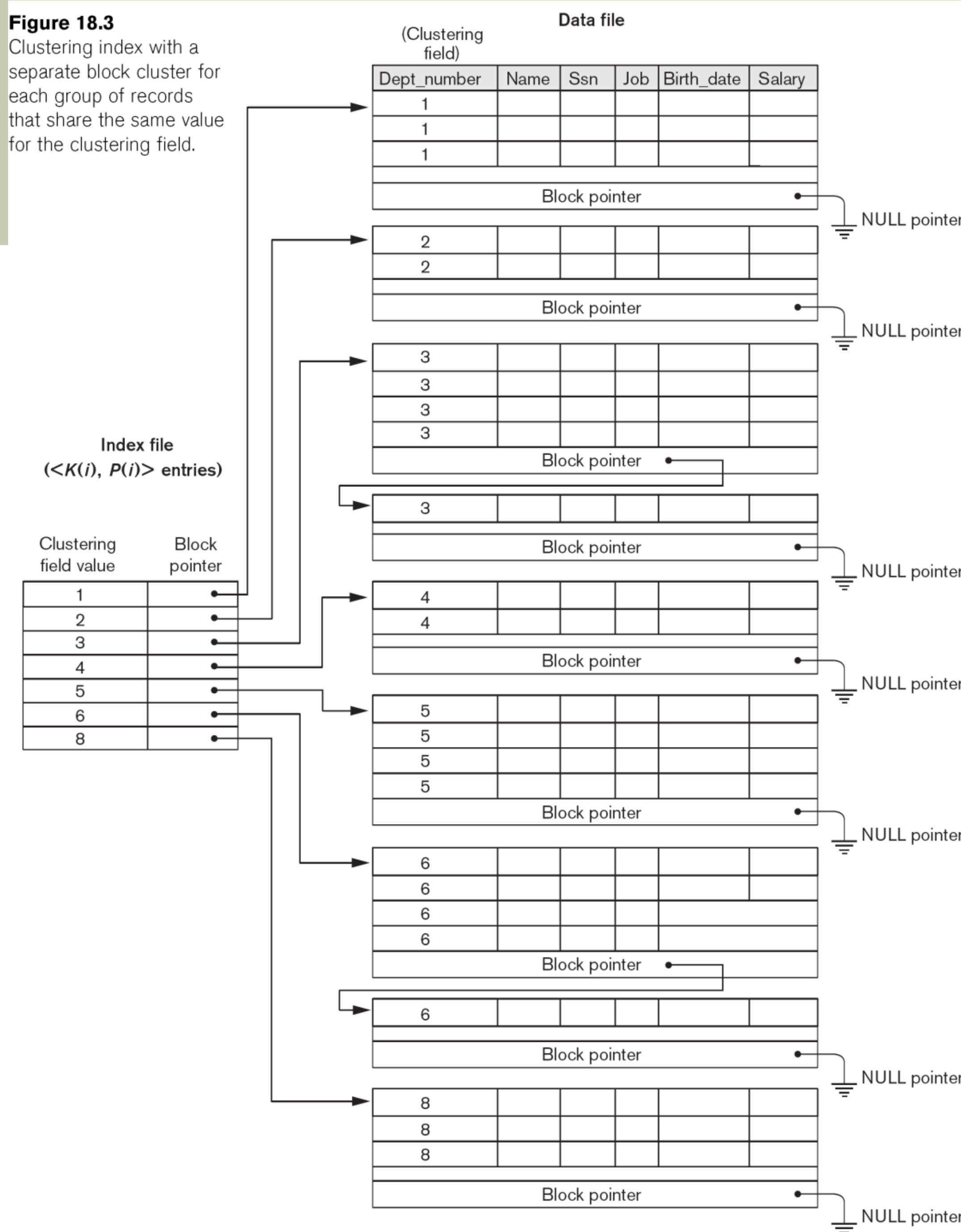


Figure 18.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

Another Clustering Index Example

- FIGURE 17.3 Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.



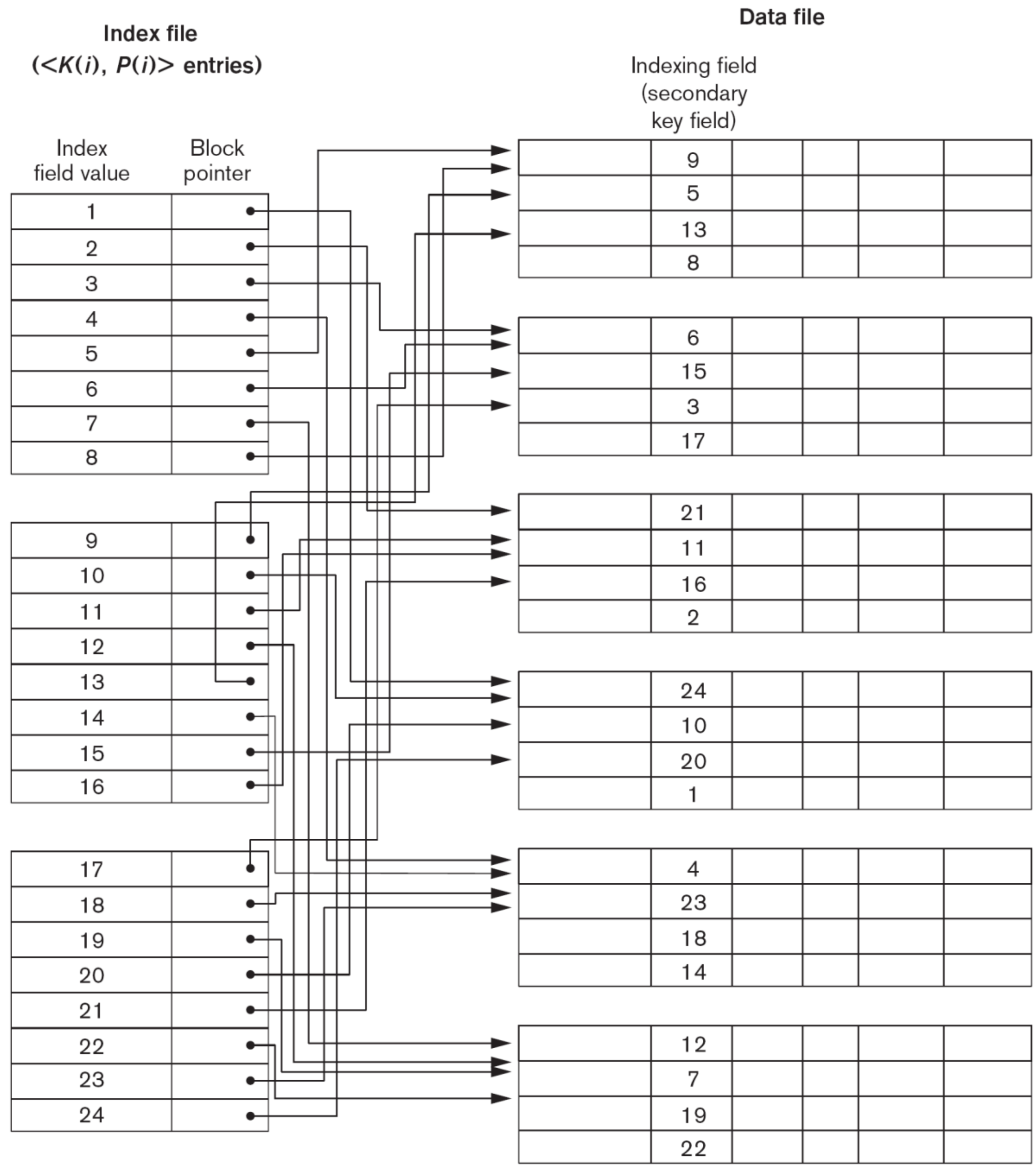
Types of Single-Level Indexes

- 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 non-key with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some **non-ordering 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*

Example of a Dense Secondary Index

- FIGURE 17.4
  A dense secondary index (with block pointers) on a non-ordering key field of a file.

Figure 18.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



Example 2

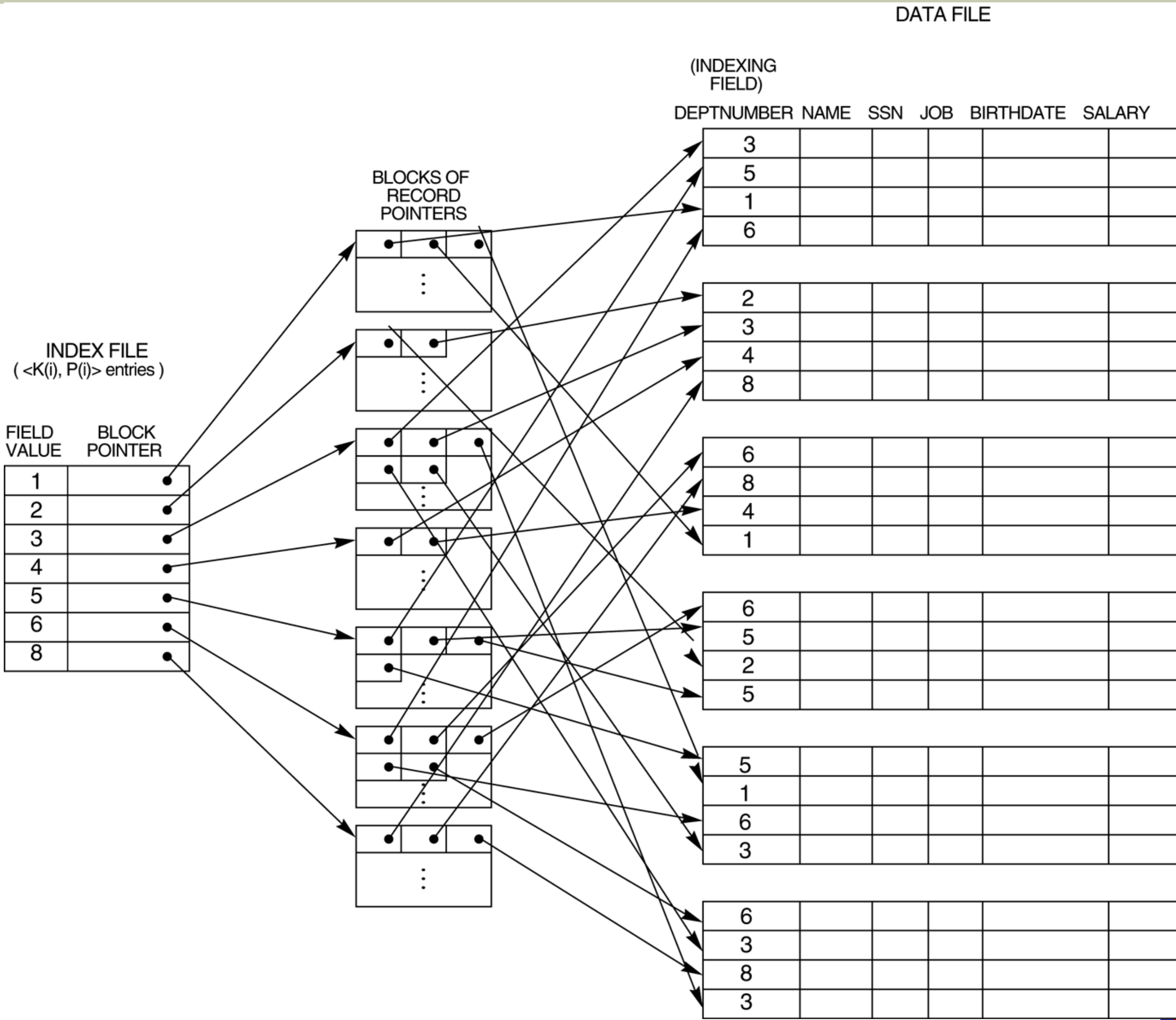
- r=30,000 fixed-length records, R=100 bytes, B=1,024 bytes, and b= 3000 blocks.
  - Linear search: b/2 = 3000/2 = 1500 block accesses
- Secondary index on a nonordering key field: V=9 bytes, and P=6 bytes
  - R_i = (9+6) = 15 bytes
  - $bfr_i = \lfloor B/R_i \rfloor = \lfloor 1024/15 \rfloor$ = 68 entries/block
  - # of index entries, r_i = r since dense
  - $b_i = \lceil r_i/bfr_i \rceil = \lceil 30{,}000/68 \rceil$ = 442 blocks
  - Binary search needs $\lceil \log_2 b_i \rceil = \lceil \log_2 442 \rceil$ = 9 block accesses
  - Total block accesses = 9 + 1 = 10

For nonkey, non-ordering field

- Option 1:
  - Duplicate index entries with the same K(i) values. Dense index
- Option 2:
  - Variable length records for the index entries (repeating pointer): e.g. <P(i,1), …, P(i,k)> for K(i)
- Option 3:
  - Create extra level to handle the multiple pointers
  - See next slide

An Example of a Secondary Index

- FIGURE 17.5
  A secondary index (with recorded pointers) on a non-key field implemented using one level of indirection so that index entries are of fixed length and have unique field values.



Types of Indexes Based on The Properties of the Indexing Field

	Index Field Used for Ordering the File	Index Field Not Used for Ordering the File
Indexing field is key	Primary Index	Secondary Index (Key)
Indexing field is nonkey	Clustering Index	Secondary Index (NonKey)

	Ordered	Non-ordered
Key	Primary Index	Secondary Index (Key)
NonKey	Clustering Index	Secondary Index (NonKey)

Properties of Index Types

Table 18.2 Properties of Index Types

Type of Index	Number of (First-level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

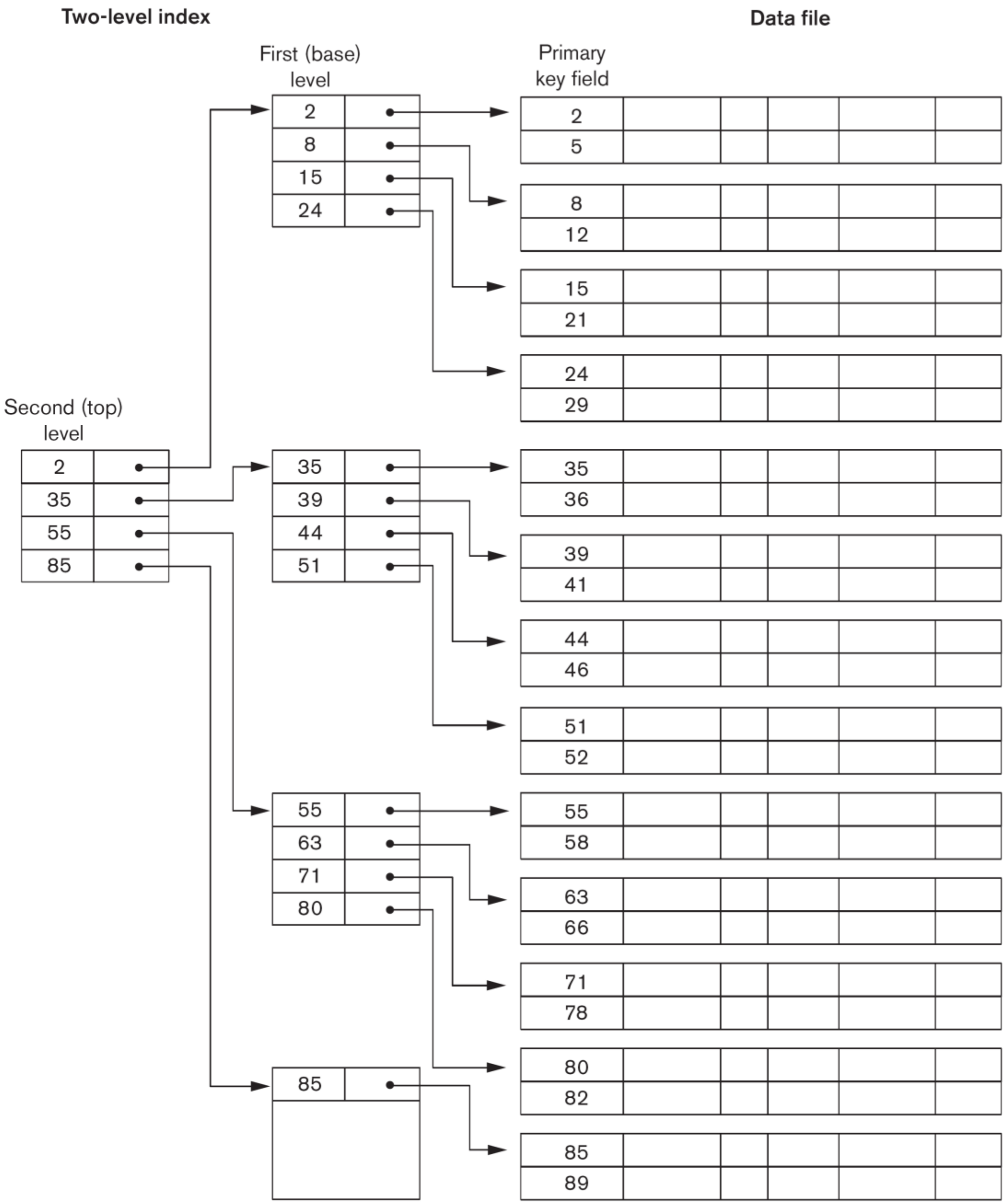
Multi-Level Indexes

- 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

A Two-level Primary Index

- FIGURE 17.6
  A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.

Figure 18.6
A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.



Example 3

- Convert Example 2 into a multilevel index
  - bfr_i = fo (fan-out) = 68
  - # of first-level blocks b_1 = 442 blocks
  - # of second-level blocks $b_2 = \lceil b_1/fo \rceil = \lceil 442/68 \rceil$ =7 blocks
  - # of third-level blocks $b_3 = \lceil b_2/fo \rceil = \lceil 7.68 \rceil$ = 1 block
  - Therefore, third level is top level (t=3)
  - Total block accesses = t+1 = 4 block accesses

Multi-Level Indexes

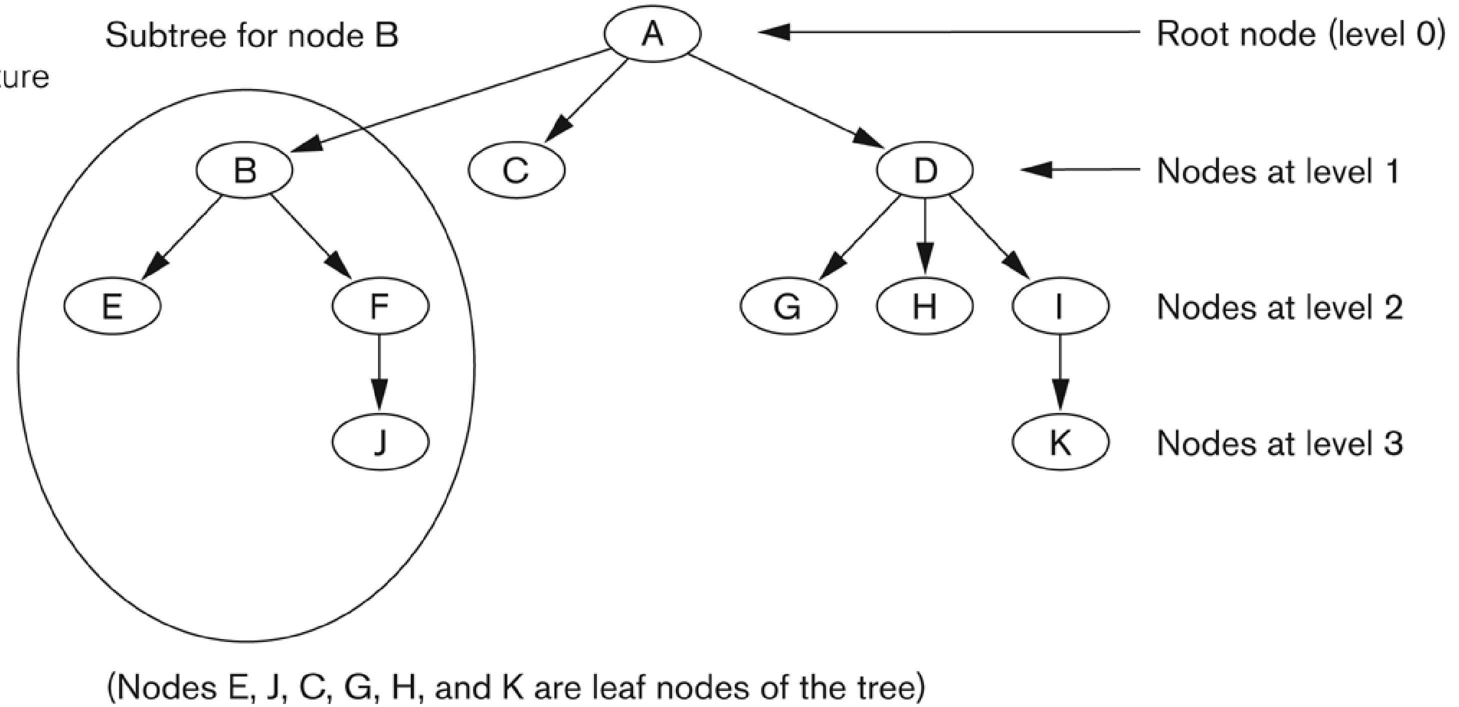
- Such a multi-level index is a form of *search tree*
  - However, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file*.
  - Dynamic multilevel index: leaves some space in each of its block for inserting new entries
  - That is called B-tree or B^+-tree

Dynamic Multilevel Indexes

- Tree data structure
  - A tree is formed of nodes
  - Each node has one parent node (except root) and several child nodes.
  - A root does not have parent node
  - A leaf does not have child node
  - A subtree of a node consists of that node and all its descendant nodes

Example of a tree data structure

Figure 14.7
A tree data structure that shows an unbalanced tree.



Search Tree

- A search tree of order p is a tree such that
  - Each node contains at most p-1 search values, and
  - P pointers in the order of $<P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q>$
    (P_i is pointer to a child node, and K_i is a search value)
  - Two constraints must hold at all times on the search tree
    1. Within each node $K_1 < K_2 < \ldots < K_{q-1}$
    2. For all values X in the subtree pointed at by P, we have $K_{i-1} < X < K_i$ for $1<i<q$; $X < K_i$ for $i=1$, and $K_{i-1} < X$ for $i=q$

A Node in a Search Tree with Pointers to Subtrees below It

- FIGURE 17.8

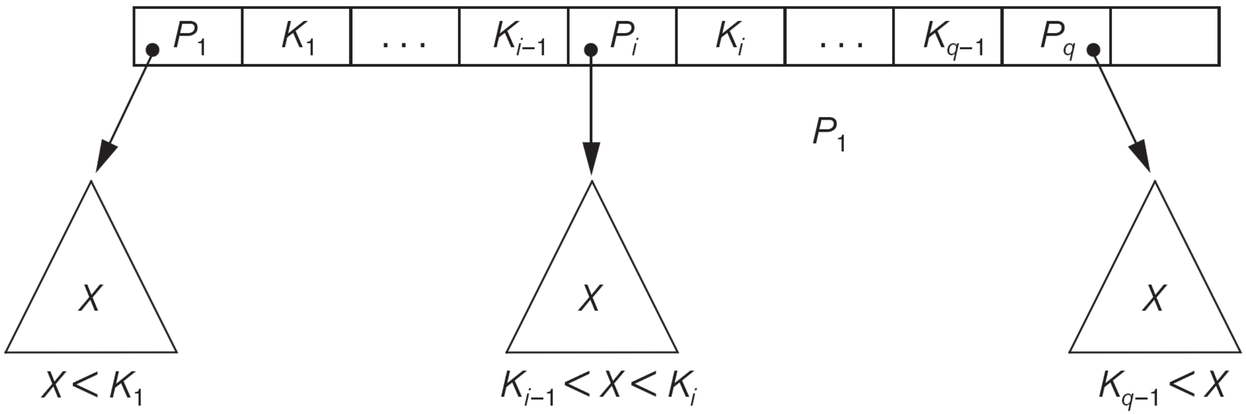
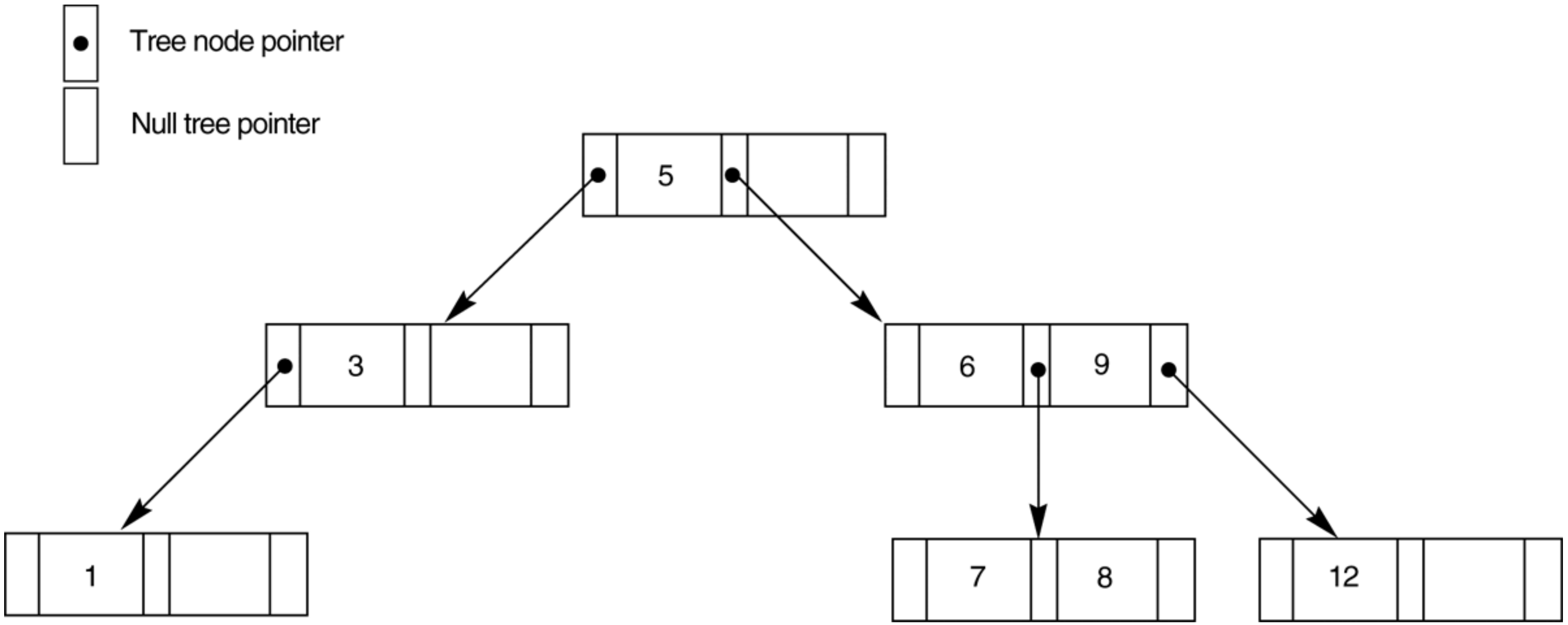


Figure 18.8
A node in a search tree with pointers to subtrees below it.

FIGURE 17.9
A search tree of order p = 3.



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

- Most multi-level indexes use B-tree or B^+-tree data structures because of the insertion and deletion problem
  - This leaves 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

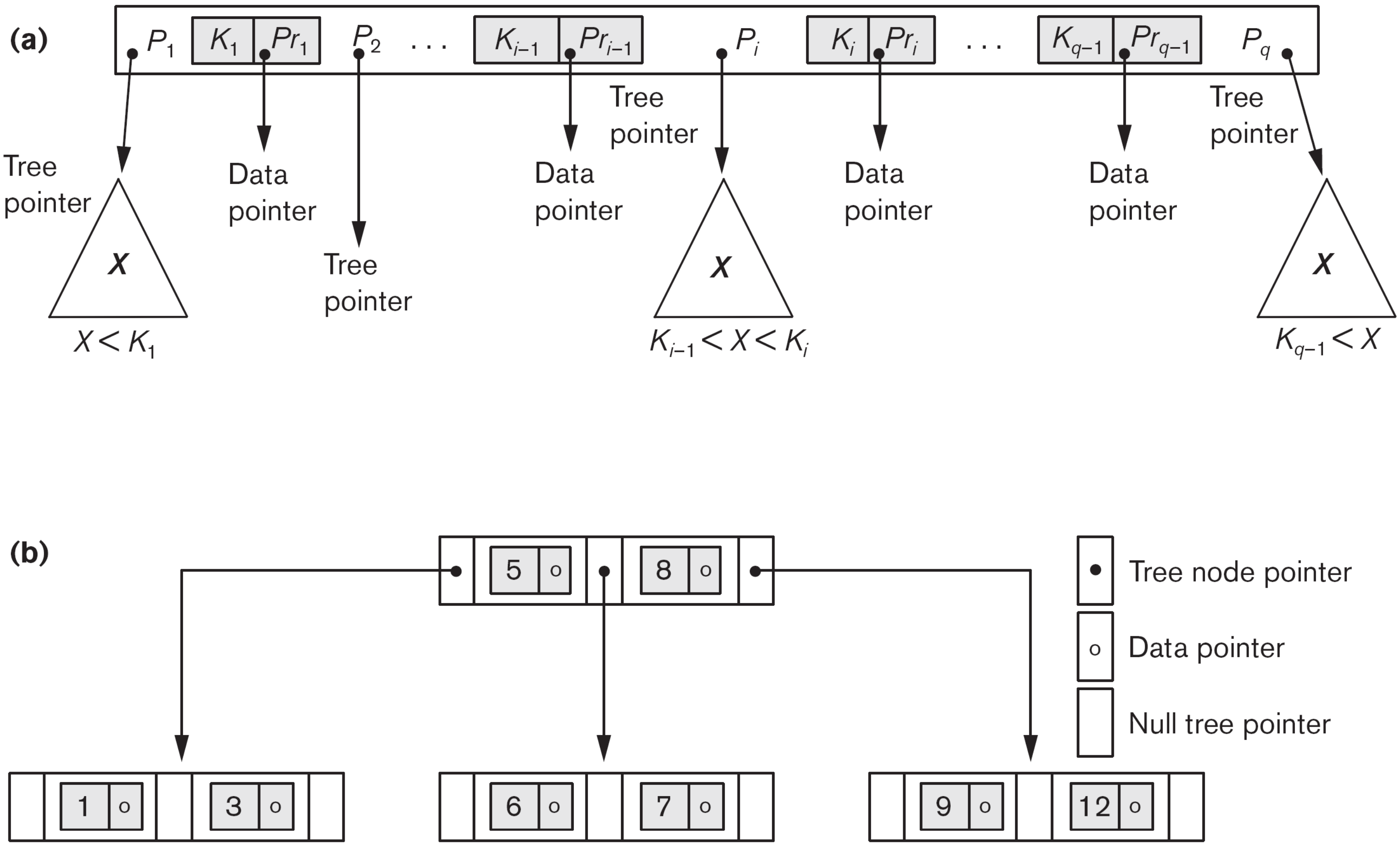


Figure 18.10
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

B-tree of order p

1. Each internal node in the B-tree
   $<P_1,<K_1,Pr_1>,P_2,<K_2,Pr_2>, \ldots, <K_{q-1},Pr_{q-1}>,P_q>$ where $q \le p$
   P_i: tree pointer, and Pr_i: data pointer
2. Within each node, $K_1<K_2<. . .<K_{q-1}$
3. For all search key field values X in the subtree
   $K_{i-1}<X<K_i$ for $1<i<q$, $X<K_i$ for $i=1$, and $K_{i-1}<X$ for $i=q$
4. Each node has at most p tree pointers
5. Each node has at least $\lceil p/2 \rceil$ tree pointers
6. A node with q tree pointers, $q \le p$, has q-1 search key field values
7. All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes

Example 4: Order p of B-tree

- Search field V=9 bytes,
- Block size B=512 bytes,
- Data pointer Pr=7 bytes, and
- Block pointer P=6 bytes
  - At most p tree pointers, p-1 data pointers, and p-1 search key fields, which should be in a single block
  - $(p*P) + ((p-1)*(Pr+V)) \leq B$

    $(p*6) + ((p-1)*(7+9)) \leq 512$

    $(22*p) \leq 528$

   Therefore, p = 23 (not 24)

Example 5: #of Blocks and Levels

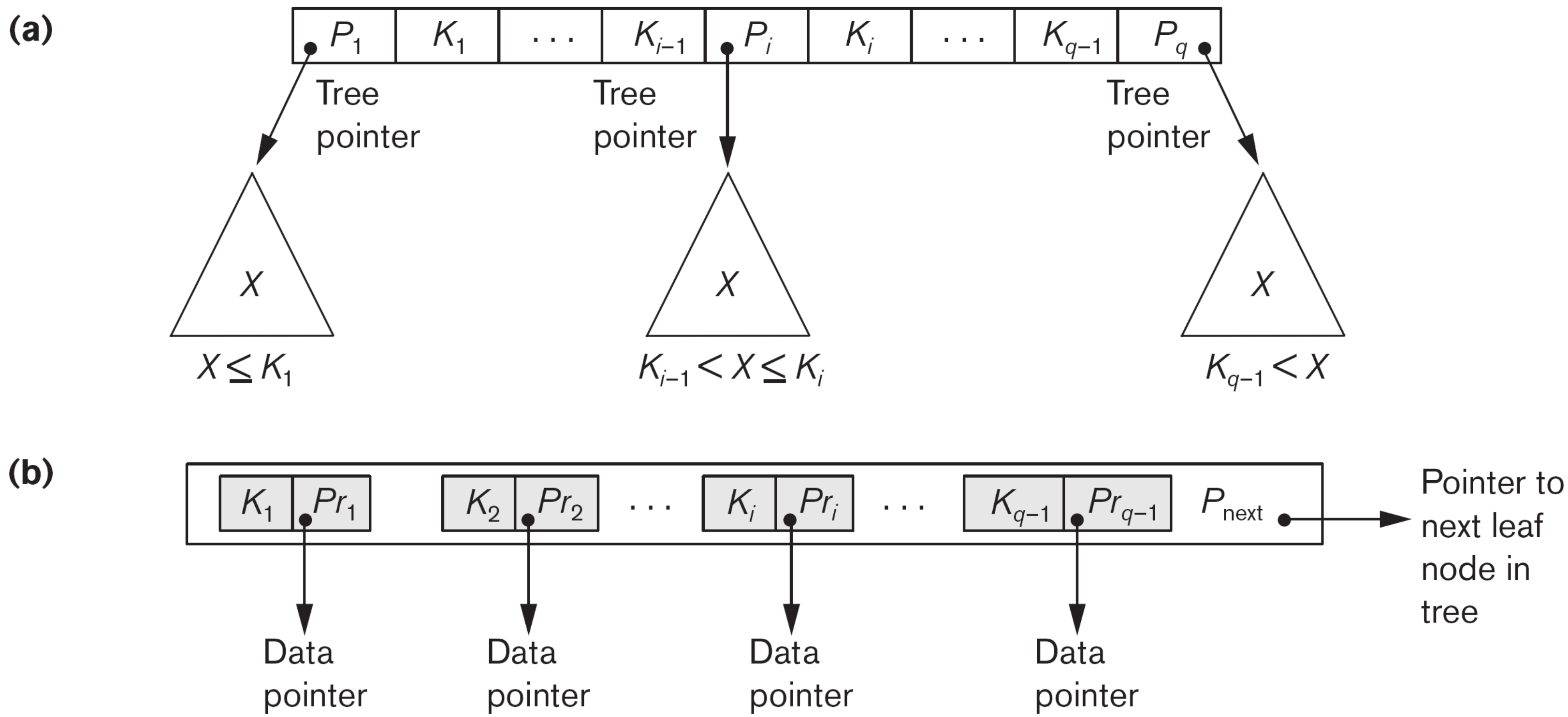
- Search field of Example 4 = nonordering key field
- Each node of B-tree is 69 % full
  - Each node will have p*0.69 = 23 * 0.69 = 16 pointers, and 15 search key field values. fo = 16

Root:	1 node	15 entries	16pointers
Level1:	16 nodes	240 entries	256 pointers
Level2:	256 nodes	3840 entries	4096 pointers
Level3:	4096 nodes	61440 entries	

  - For example, two-level (3840+240+15=4095), or three-level (65,535 entries)

The Nodes of a B$^+$-tree

Figure 18.11
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.



B^+-tree of order p (Internal nodes)

1. Each internal node is
   $<P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}\ P_q>$ where $q \leq p$,
   each P_i is a tree pointer
2. Within each internal node, $K_1 < K_2 < \ldots < K_{q-1}$
3. For all search field values X in the subtree pointed at by Pi, $K_{i-1} < X < K_i$ for $1 < i < q$, $X < K_i$ for $i=1$, and $K_{i-1} < X$ for $i=q$
4. Each internal node has at most p tree pointers
5. Each internal node has at least $\lceil p/2 \rceil$ tree pointers
6. An internal node with q pointers $q \leq p$, has q-1 search field values

B^+-tree of order p (leaf nodes)

1. Each leaf node is
   $<<K_1,Pr_1>, <K_2,Pr_2>,\ldots, <K_{q-1},Pr_{q-1}>,P_{next}>$ where $q \leq p$, each Pr_i is a data pointer, and P_{next} points to the next leaf node
2. Within each leaf node, $K_1<K_2<. . .<K_{q-1}$, $q \leq p$
3. Each Pr_i is a data pointer points to K_i search field value
4. Each leaf node has at least $\lceil p/2 \rceil$ tree pointers
5. All leaf nodes are at the same level

Example 6: Order p of B^+-tree

- Search field V=9 bytes,
- Block size B=512 bytes,
- Data pointer Pr=7 bytes, and
- Block pointer P=6 bytes
  - At most p tree pointers, and p-1 search key fields, which should be in a single block
  - $(p*P) + ((p-1)*V) \leq B$

    $(p*6) + ((p-1)*9) \leq 512$

    $(15*p) \leq 521$

    Therefore, $p = 34$
  - The leaf node order p_{leaf}

    $(p_{leaf}*(Pr+V)) + P \leq B$ $\quad$ $(p_{leaf}*(7+9) + 6) \leq 512$

    $(16* p_{leaf}) \leq 506 \rightarrow p_{leaf} = 31$

Example 7: # of Entries

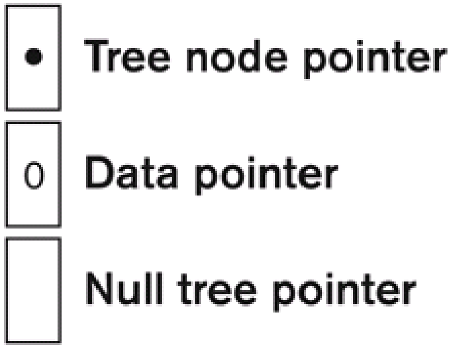
- Construct B+-tree for Example 6
- Each node of B-tree is 69 % full
  - Each node will have 34 * 0.69 = 23 pointers, and 22 search key field values.
  - Each leaf node $0.69 * p_{leaf} = 0.69 * 31 \rightarrow 21$ data record pointer

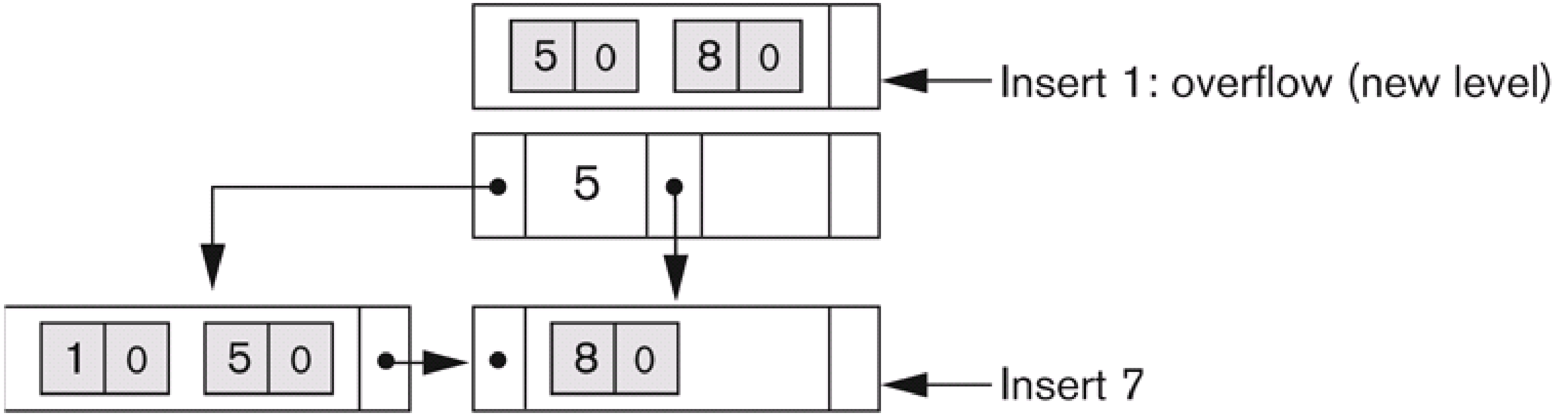
  - Root: 1 node 22 entries 23 pointers
  - Level1: 23 nodes 506 entries 529 pointers
  - Level2: 529 nodes 11,638 entries 12,167 pointers
  - Level3: 12,167 nodes 255,507 data record pointer

  - For example, three-level B+-tree 255,507 record pointers

Insertion with B^+-tree

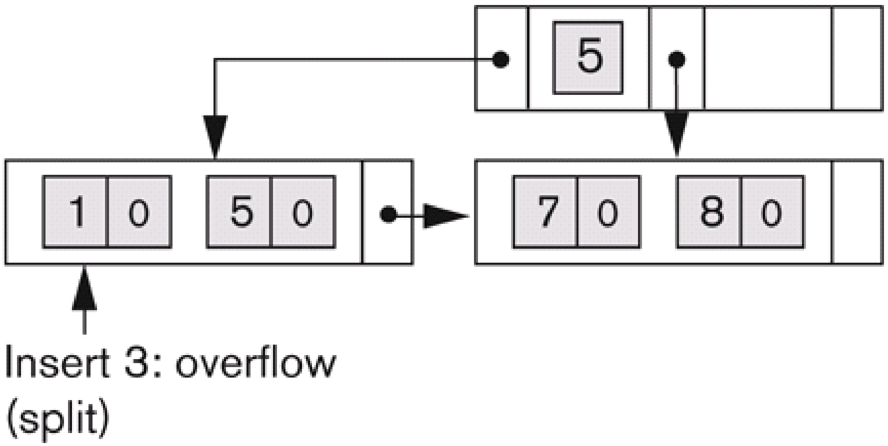
- $p = 3$ and $p_{leaf} = 2$
- Insertion sequence
  - 8, 5, 1, 7, 3, 12, 9, 6
- Insert 8, 5, 1





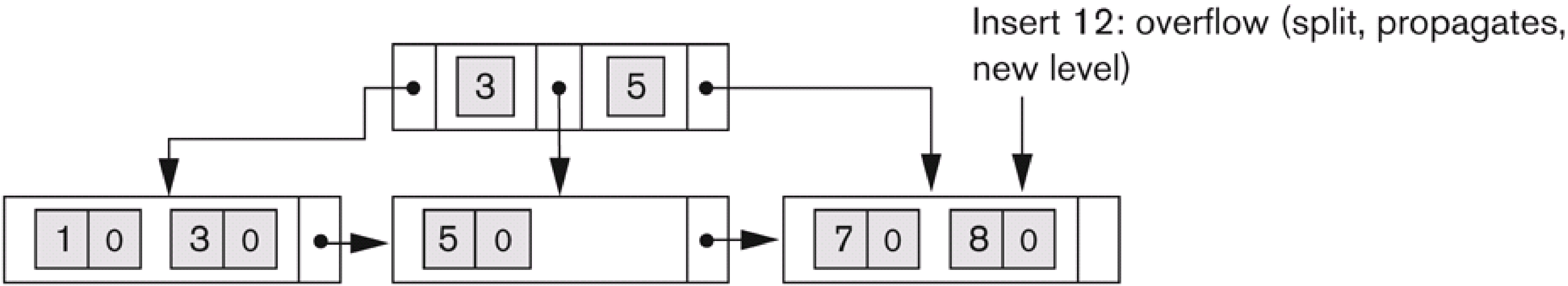
Insertion with B$^+$-tree

- Insert 7, 3



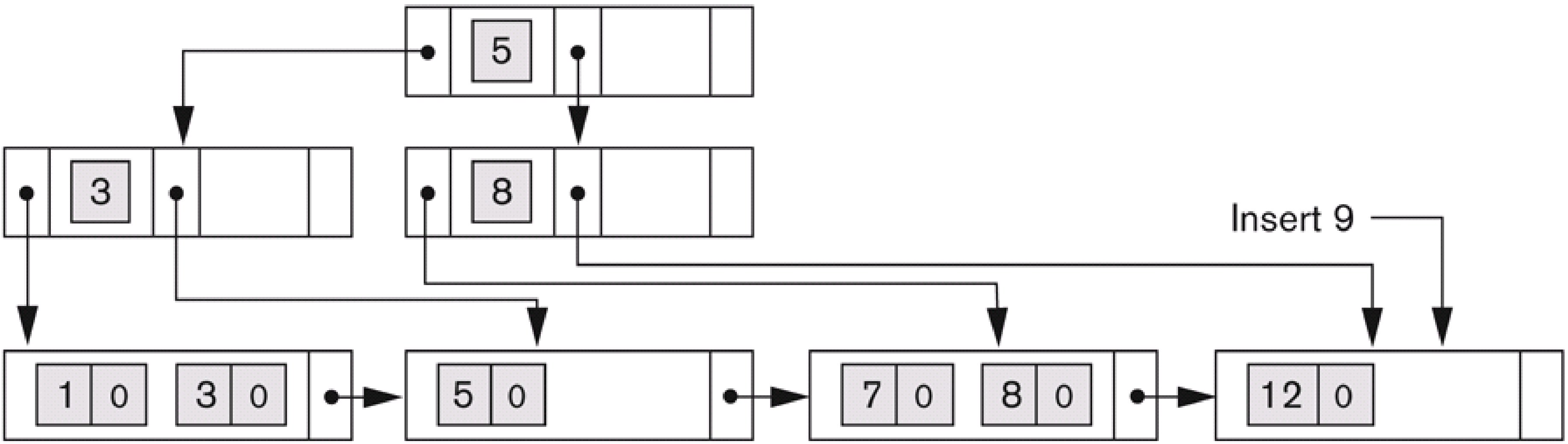
Insertion with B$^+$-tree

- Insert 12



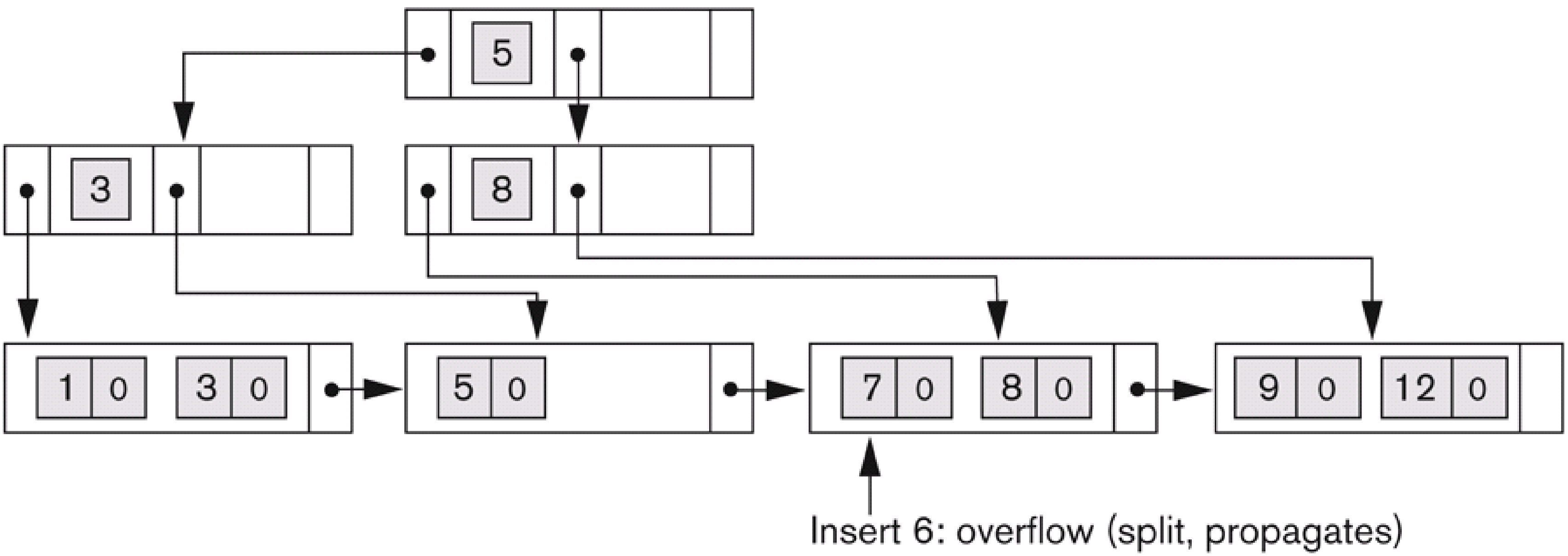
Insertion with B$^+$-tree

- Insert 9



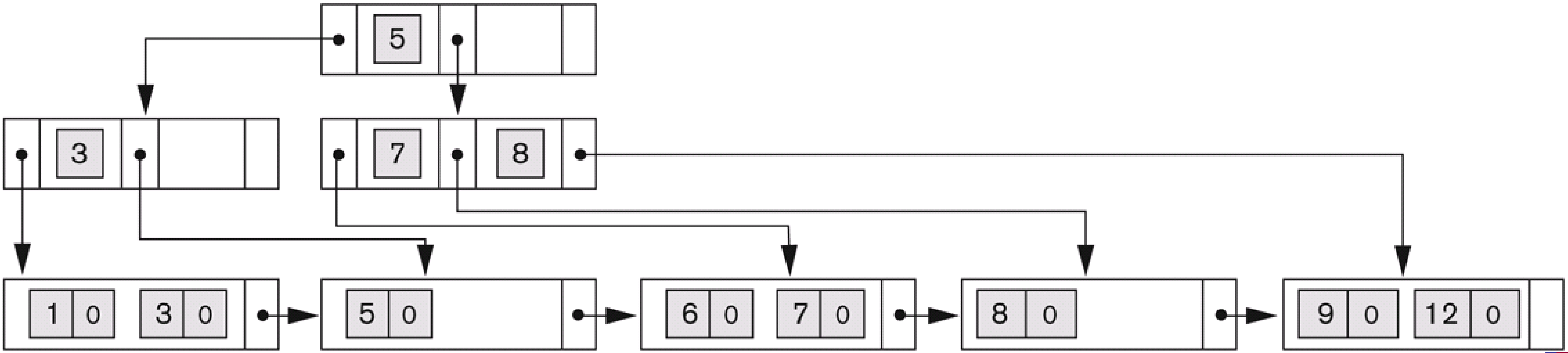
Insertion with B⁺-tree

- Insert 6



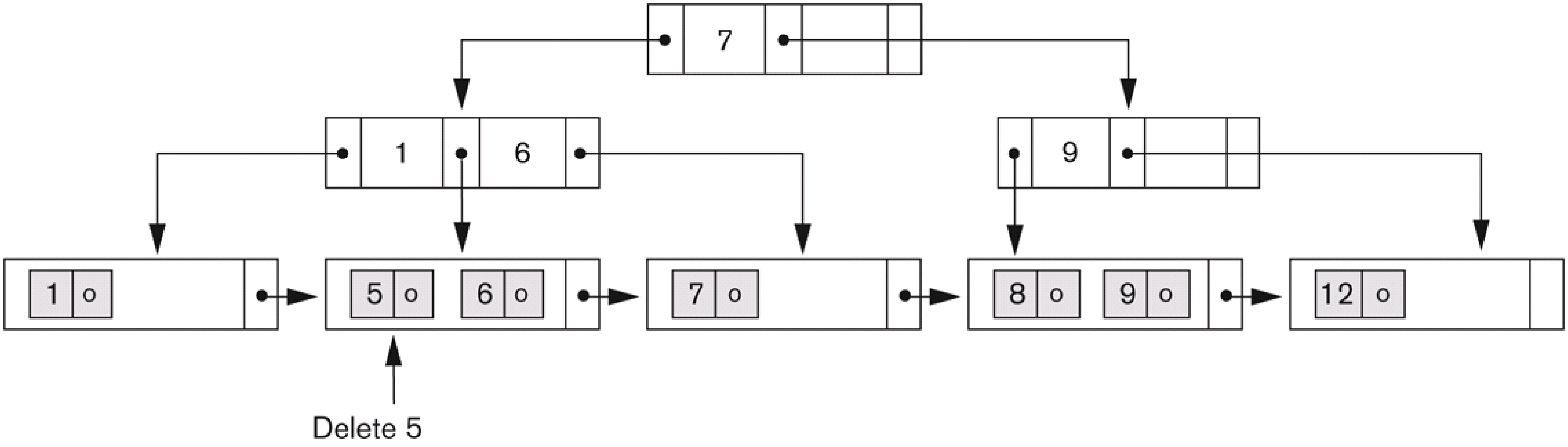
Insertion with B$^+$-tree

- Done



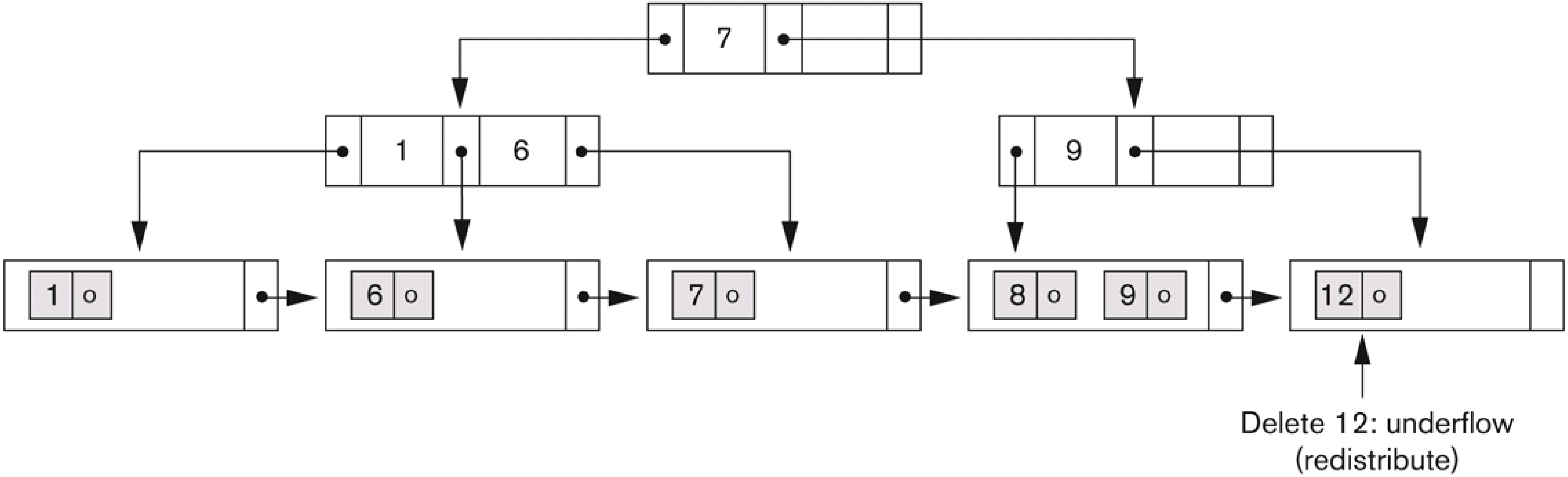
Deletion with B+-tree

- Deletion sequence: 5,12, 9
- Delete 5



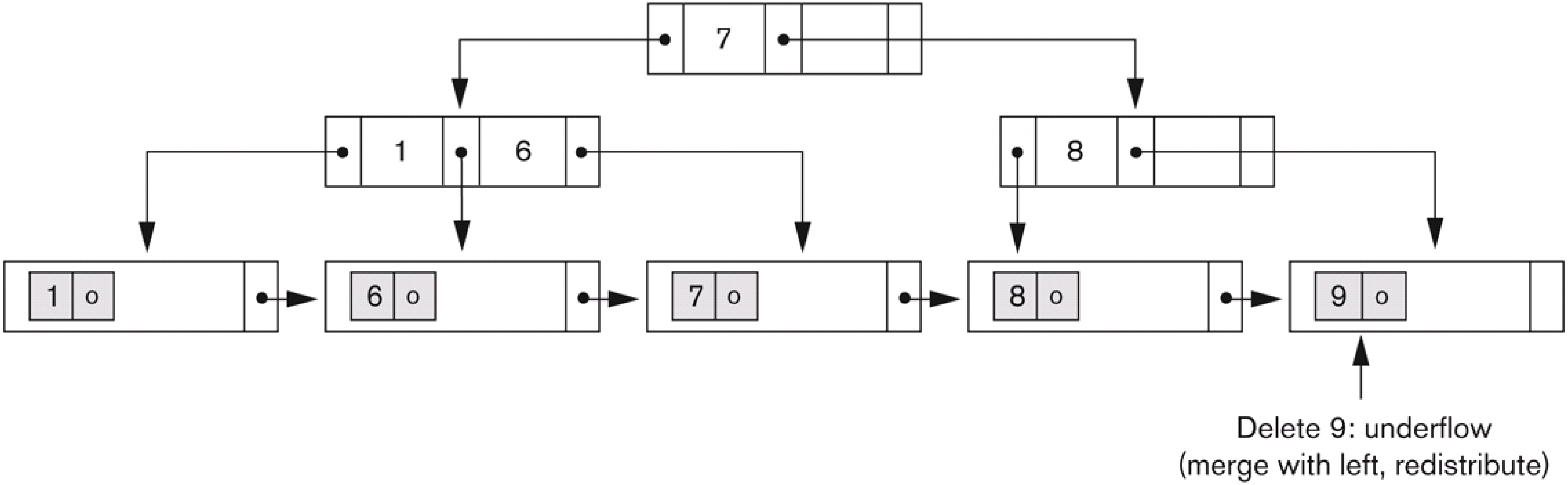
Deletion with B+-tree

- Delete 12



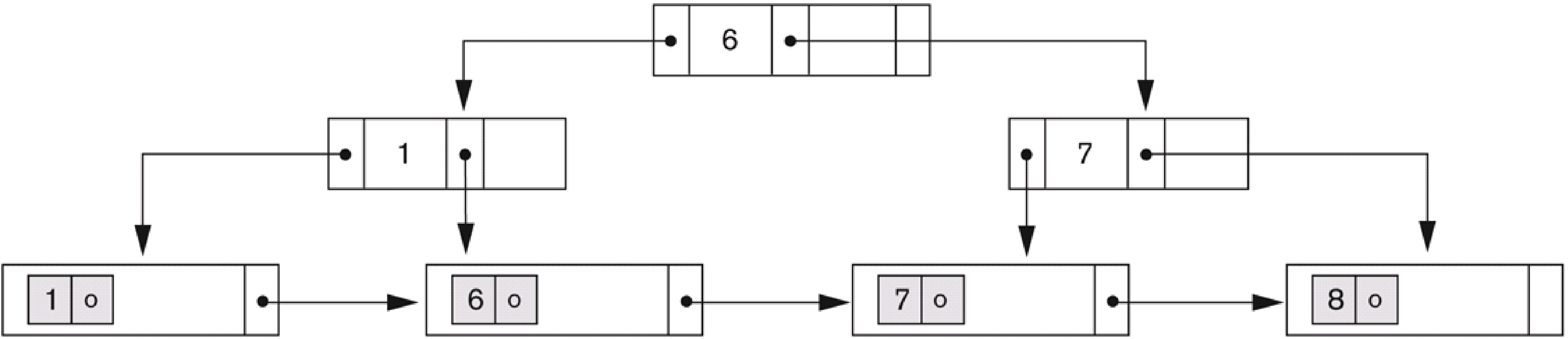
Deletion with B^+-tree

- Delete 9



Deletion with B^{+}-tree

- Done



Summary

- 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