Database Management Systems (DBMS)

Lec 22: Query processing and optimization (Contd.)

Ramesh K. Jallu

IIIT Raichur

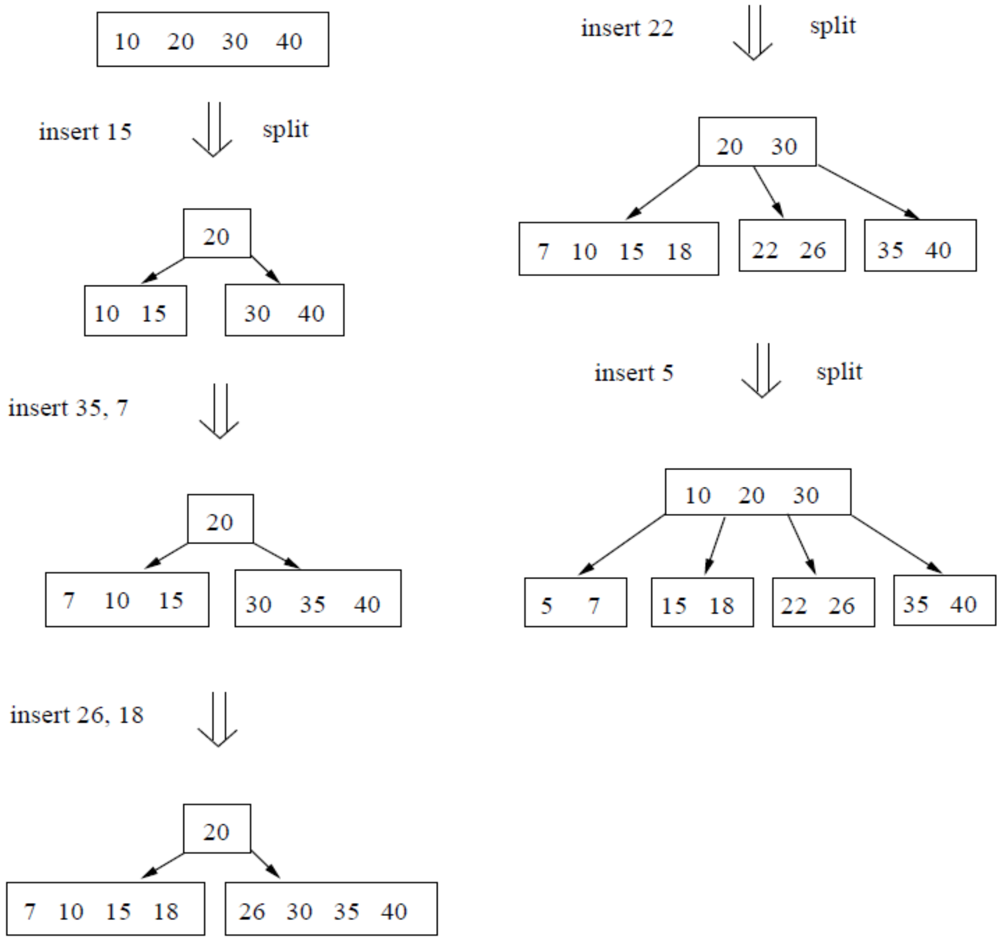
Date: 20/04/21

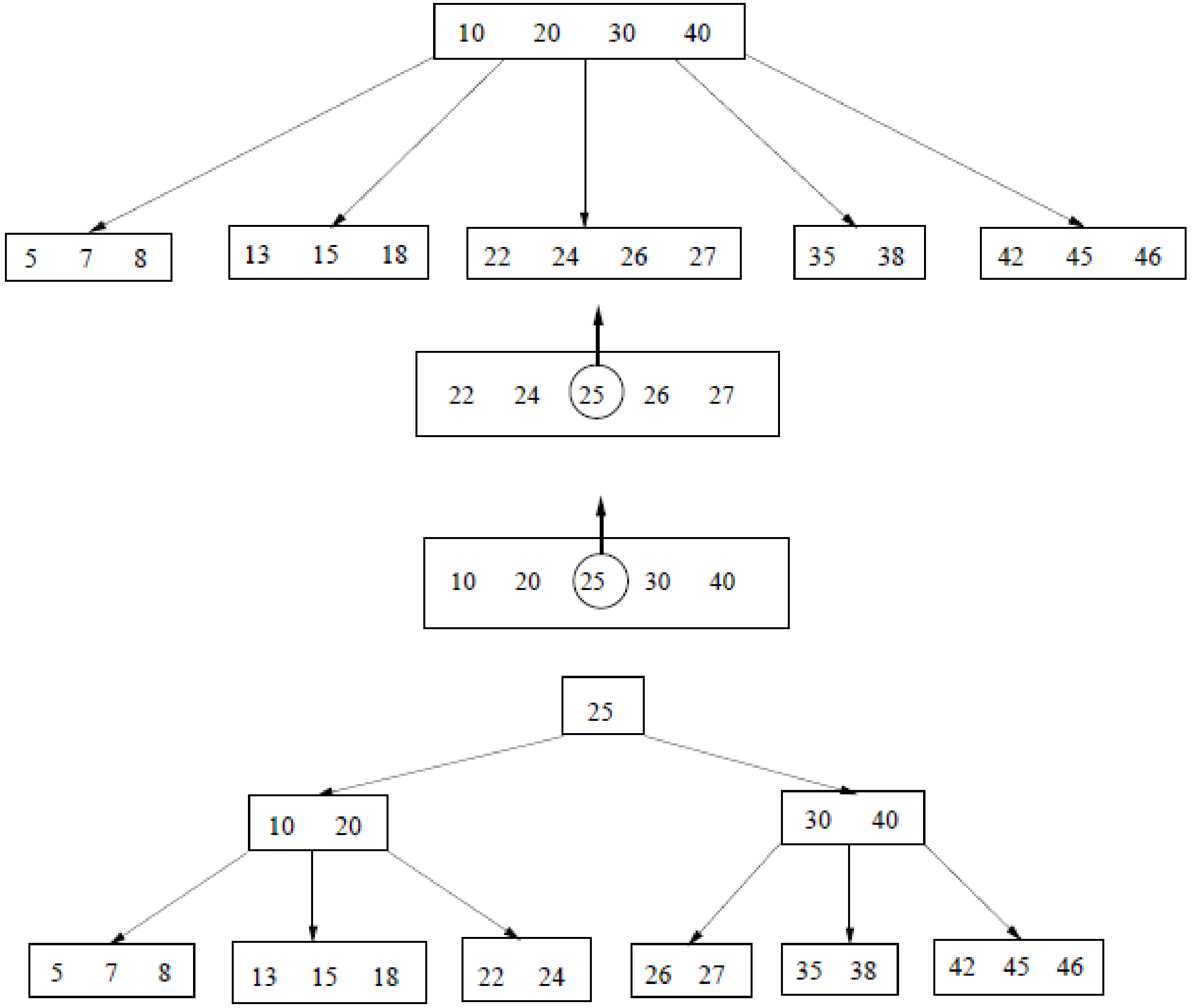
Recap

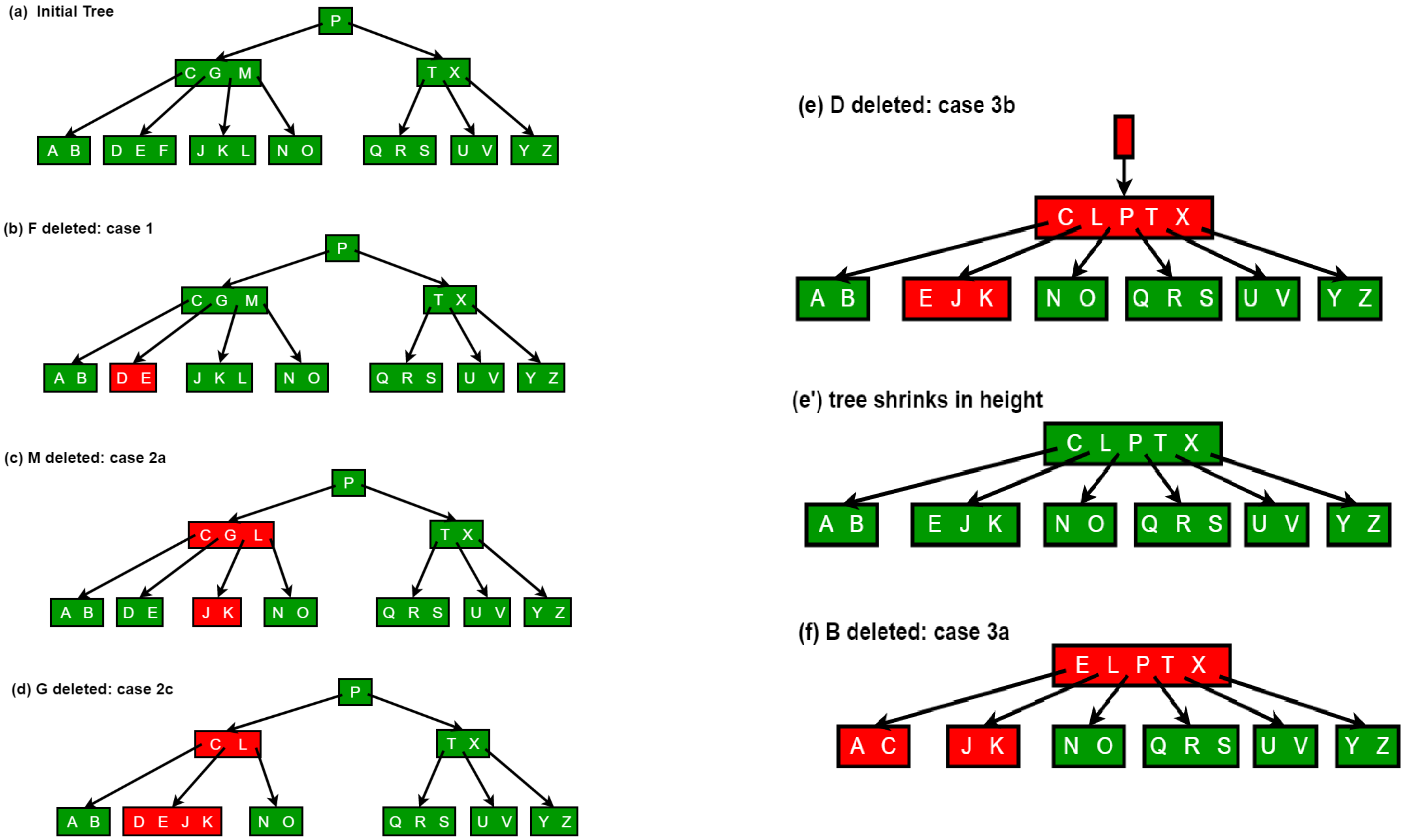
- Multilevel indexing
- Search trees
- B-tree

B-tree (Insertion)

* Let $p = 5$; input: 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5.







(a) Initial Tree

(b) F deleted: case 1

(c) M deleted: case 2a

(d) G deleted: case 2c

(e) D deleted: case 3b

(e') tree shrinks in height

(f) B deleted: case 3a

B$^+$-tree

- In a B-tree, every value of the search key appears at some level in the tree, along with a data pointer

- In a B$^+$-tree, data pointers are stored *only at the leaf nodes* of the tree
  - I.e., internal nodes do not store record/block pointers; only leaf nodes do

- The leaf nodes have an entry for *every* value of the search field, along with a data/block pointer to the record if the search field is a ***key field***

- For a ***nonkey*** search field, the pointer points to a block containing pointers to the data file records, creating an extra level of indirection

B$^+$-tree (Contd.)

- The leaf nodes of the B$^+$-tree are linked to provide ordered access on the search field to the records

- B$^+$-tree is used to implement multilevel indexing
  - The leaf nodes are similar to the first (base) level of an index
  - Internal nodes of the B$^+$-tree correspond to the other levels

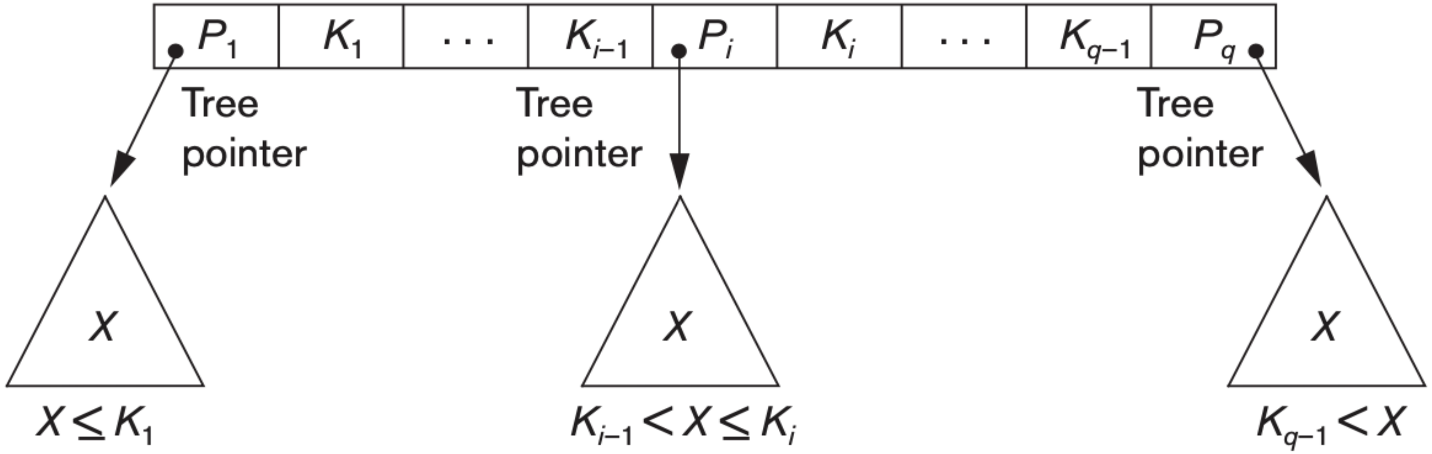
- Some search field values from the leaf nodes are ***repeated*** in the internal nodes of the B$^+$-tree to guide the search

B^+-tree: Internal node structure

- Each internal node is of the form $<P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q>$ where $q \leq p$ and each P_i is a **tree pointer**

- Within each internal node, $K_1 < K_2 < \ldots < K_{q-1}$

- For all search field values X in the subtree pointed at by P_i, we have $K_{i-1} < X \leq K_i$ for $1 < i < q$; $X \leq K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$



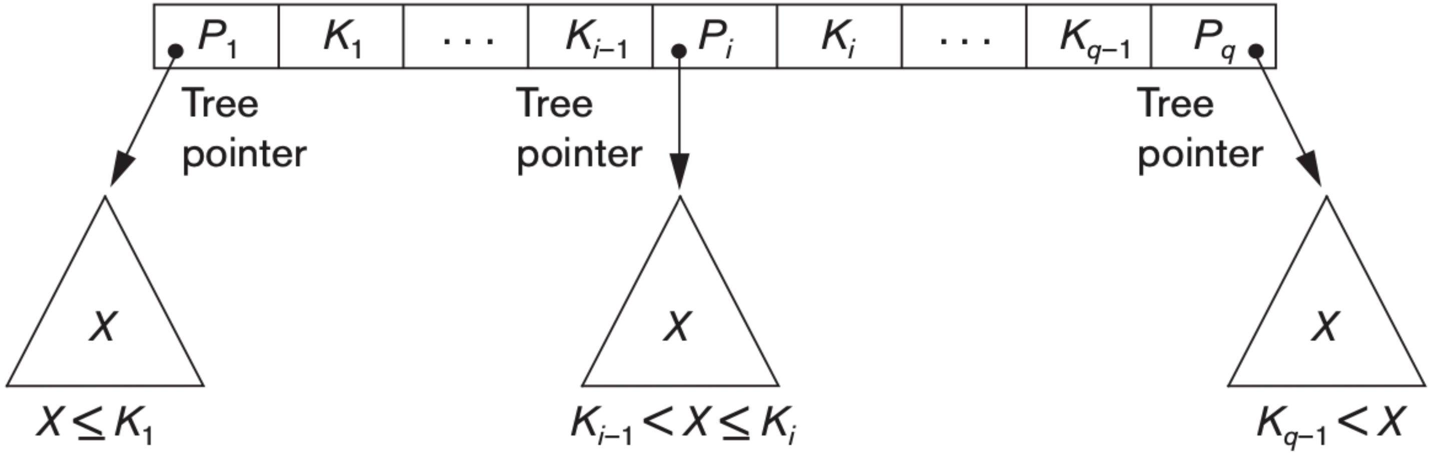
B$^+$-tree: Internal node structure (Contd.)

- Each internal node has at most p tree pointers

- Each internal node, except the root, has at least $\lceil (p/2) \rceil$ tree pointers

- The root node has at least two tree pointers if it is an internal node

- An internal node with q pointers, $q \leq p$, has $q - 1$ search field values

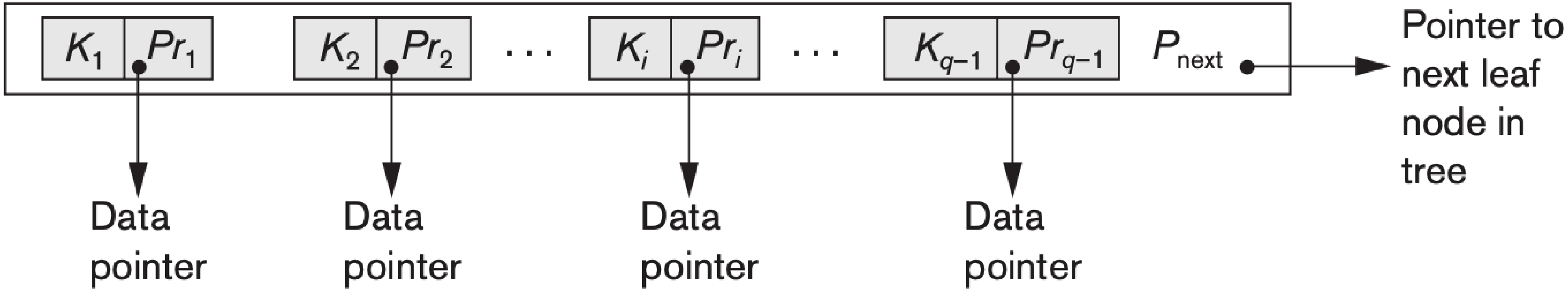


B^+-tree: Leaf node structure

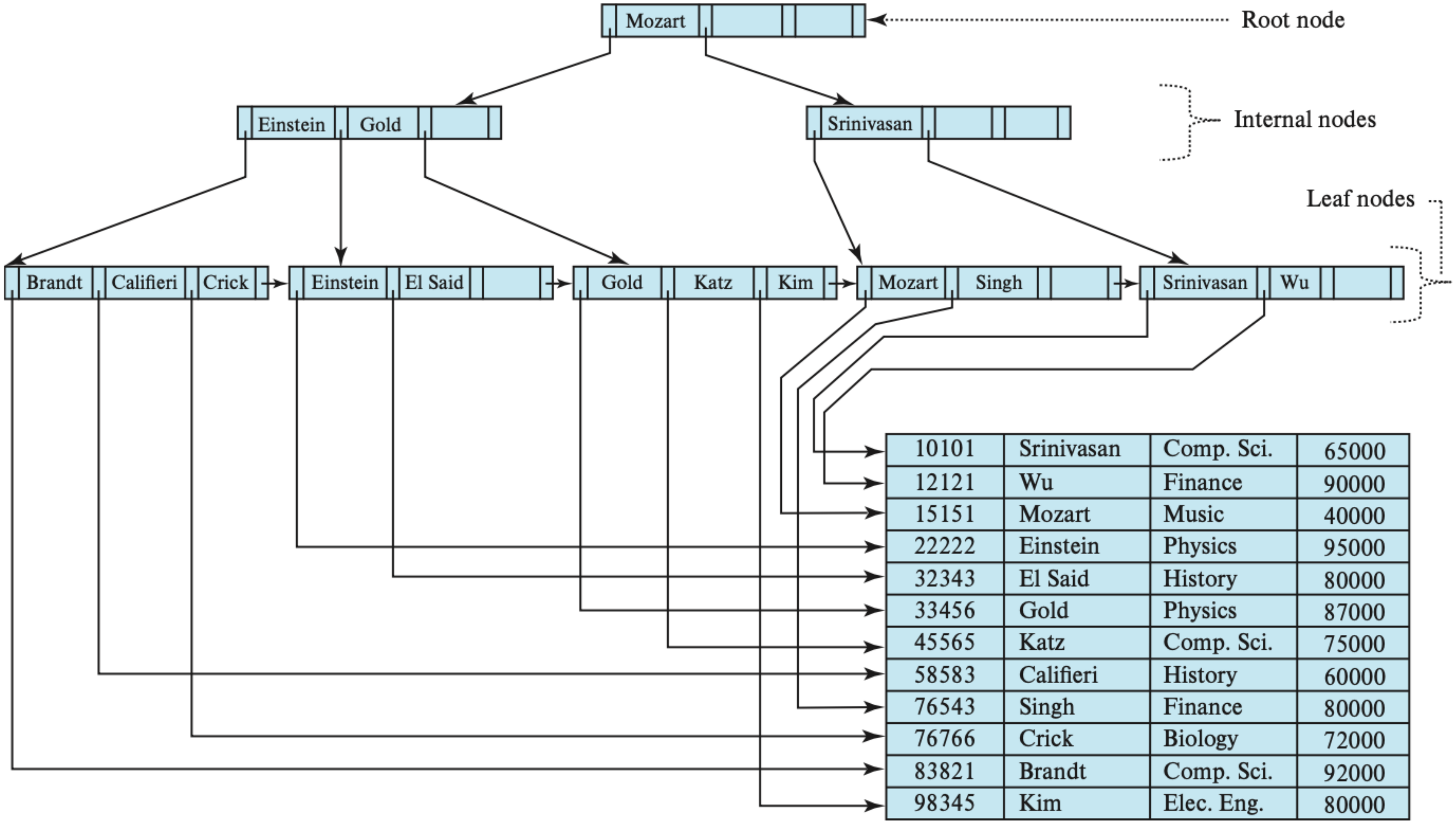
- Each leaf node is of the form $<<K_1, Pr_1>, <K_2, Pr_2>, \ldots , <K_{q-1}, Pr_{q-1}>, P_{\text{next}}>$ where $q \leq p$, each Pr_i is a data pointer, and P_{next} points to the next *leaf node*

- Within each leaf node, $K_1 \leq K_2 \ldots , K_{q-1}$
- Each leaf node has at least $\lceil (p/2) \rceil$ values

- All leaf nodes are at the same level



B$^+$-tree: Example



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

B^{+}-tree: Order of nodes

- Because the structures for internal and for leaf nodes of a B+-tree are different, the order *p* can be different

- Suppose that the search key field is $K = 9$ bytes long, the block size is $B = 512$ bytes, a record pointer is $Pr = 7$ bytes, and a tree pointer is $P = 6$ bytes

$$(p * P) + ((p - 1) * K) \leq B$$
$$(p * 6) + ((p - 1) * 9) \leq 512$$
$$(15 * p) \leq 521;\ p \leq 34$$

$$(p_{\mathrm{leaf}} * (Pr + K)) + P \leq B$$
$$(p_{\mathrm{leaf}} * (7 + 9)) + 6 \leq 512$$
$$(16 * p_{\mathrm{leaf}}) \leq 506;\ p_{\mathrm{leaf}} \leq 31$$

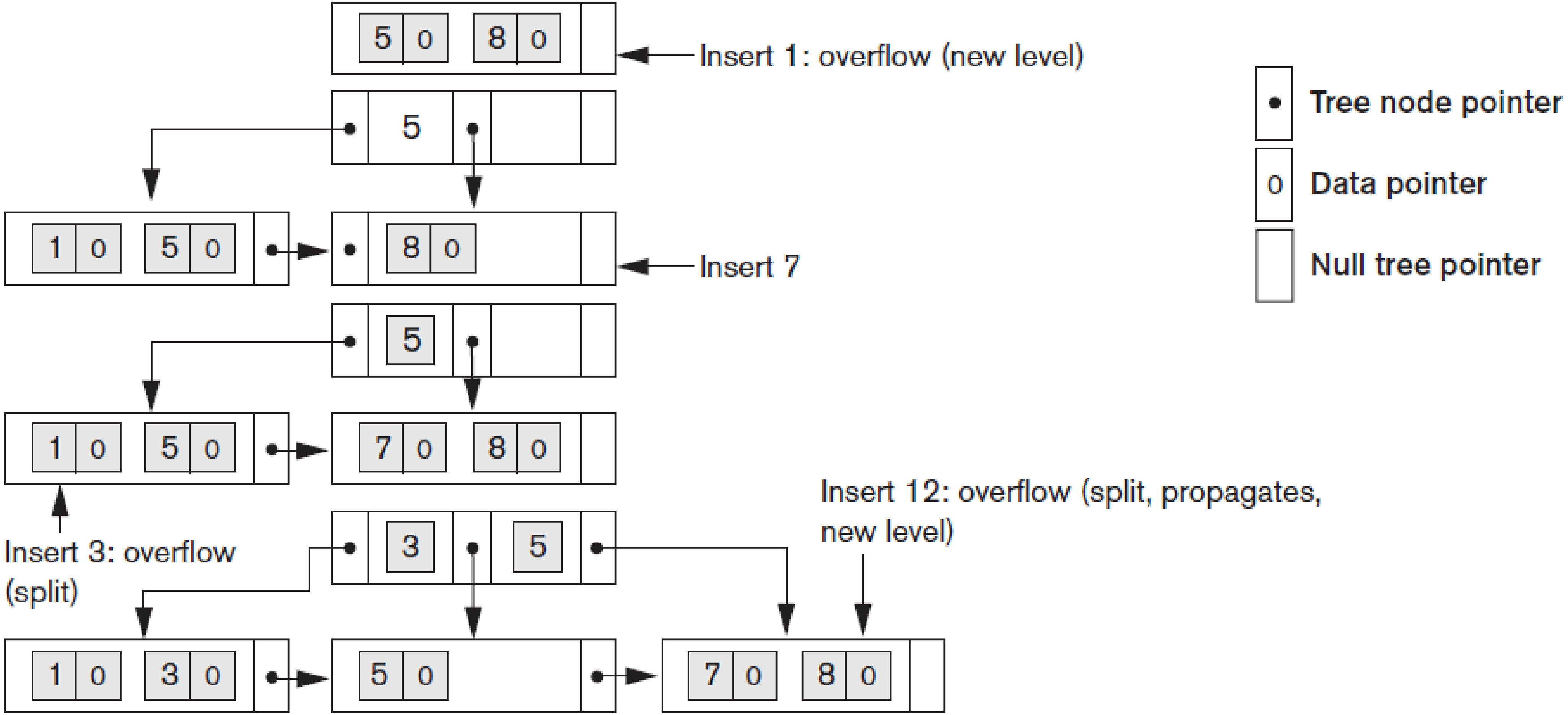
$$(p * P) + ((p - 1) * K) + (p - 1) * Pr \leq B$$
$$(p * 6) + ((p - 1) * 16) \leq 512$$
$$(22 * p) \leq 528;\ \mathrm{p} \leq 24$$

B$^+$-tree: Insertion

- **Splitting a leaf node because of overflow**
  - When a leaf node is full and a new value is inserted there, the node overflows and must be split
  - The first $j = [(p_{leaf}+1)/2]$ entries are kept in the original node, and the remaining entries are moved to a new leaf node
  - The j^{th} value is replicated in the parent internal node
  - If the parent internal is full, the new value (the j^{th} value in above step) will cause it to overflow also, so it must be split

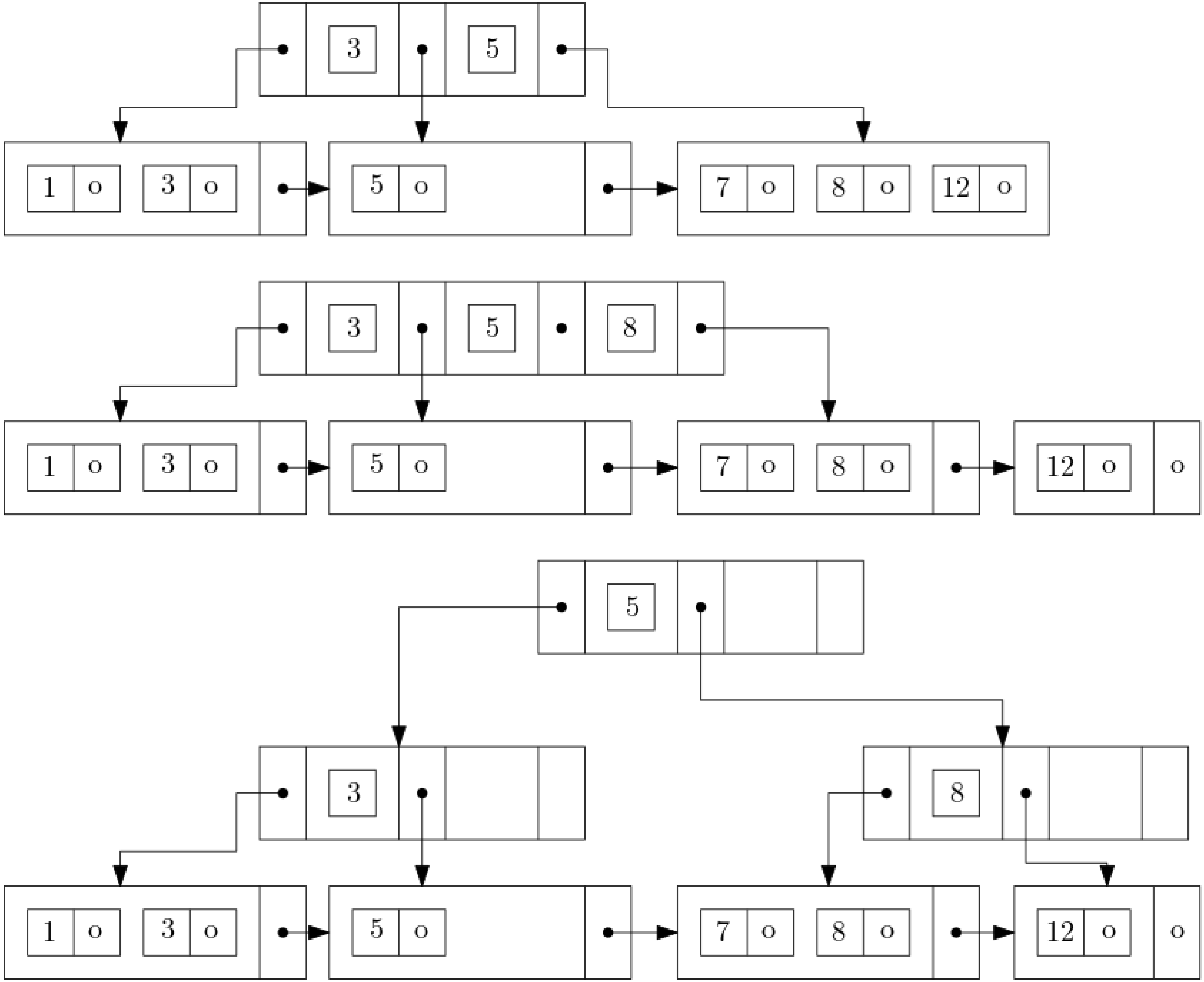
An example of insertion in a B$^+$-tree with $p = 3$ and $p_{leaf} = 2$.

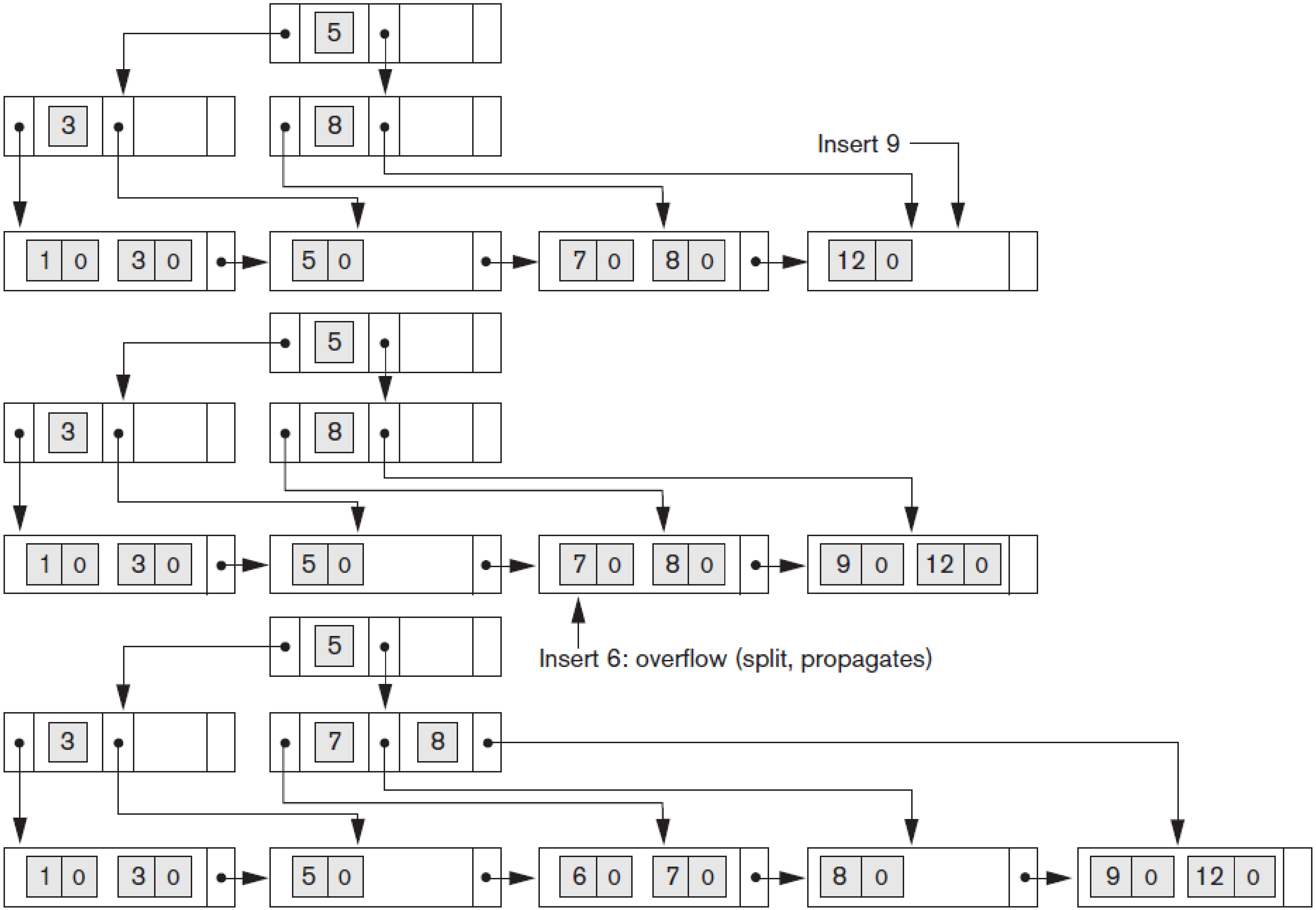
Insertion sequence: 8, 5, 1, 7, 3, 12, 9, 6



B^+-tree: Insertion (Contd.)

- **Splitting an internal node because of overflow**
  - The entries in the internal node up to P_j, where j = $\lfloor(p + 1)/2\rfloor$ are kept, while the j^{th} search value in moved to the parent, but not replicated
  - A new internal node will hold the entries from P_{j+1} to the end of the entries in the node
- This splitting can propagate all the way up to create a new root and hence a new level for the B^+-Tree


3
5
1 o
3 o
5 o
7 o
8 o
12 o
3
5
8
1 o
3 o
5 o
7 o
8 o
12 o
o
5
3
8
1 o
3 o
5 o
7 o
8 o
12 o
o



Insert 9

Insert 6: overflow (split, propagates)

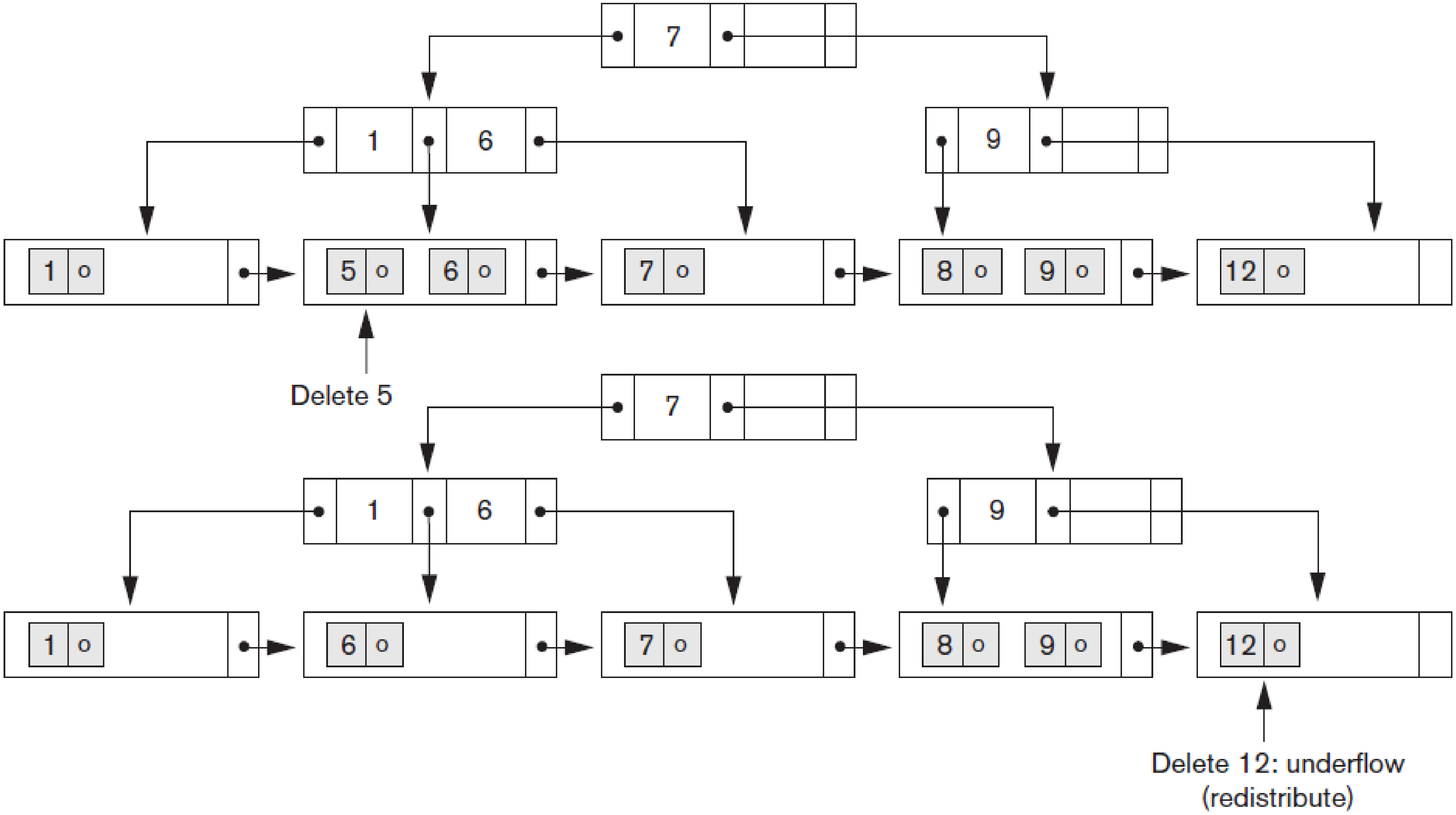
B$^+$-tree: Deletion

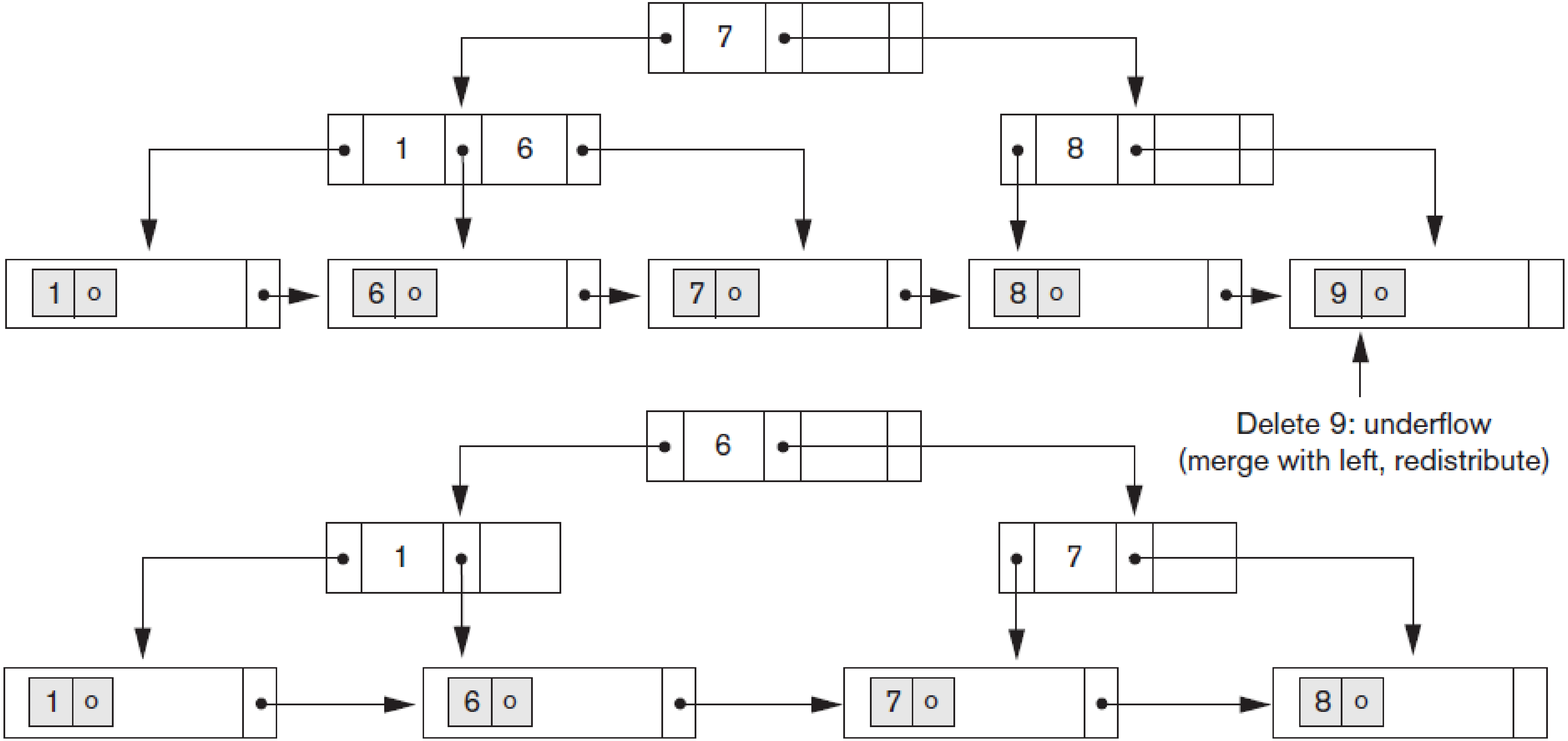
- When an entry is deleted, it is always removed from the leaf level
- If it happens to occur in an internal node, it must also be removed from there
  - In this case, the value to its left in the leaf node must replace it in the internal node
    - **Reason**: that value is now the rightmost entry in the subtree

B$^+$-tree: Deletion (Contd.)

- **Underflow in the leaf node**:
  - Try to redistribute the entries from the *left or right* sibling node, if possible
  - If the redistribution is not possible, then the three nodes are merged into two leaf nodes
    - In this case, underflow may propagate to internal nodes because one fewer tree pointer and search value are needed
    - This can propagate and reduce the tree levels

Deletion sequence: 5, 12, 9





Delete 9: underflow
(merge with left, redistribute)

Thank you!