B+ and B Tree Indexes

- **Two Types of Tree File Structures**

  B^+ Trees
  B Trees

- **B^+ Tree Index Files**

  The B^+ Tree file structure (or a version of it) is the most widely used in database systems.

  B^+ Tree index structure takes the form of a balanced tree in which every path from the root node of the tree to a leaf node is of the same length.

  Each node in the tree has between $\lceil n/2 \rceil$ and n child nodes, where n is fixed for a particular tree.

  n refers to the number of pointers a node can hold and $\lceil x \rceil$ refers to the ceiling of x or the smallest integer not less than x, i.e. we round upward. The number of pointers, n, for a B+ tree is defined by the software designers (i.e. designers of database software such as Oracle, Access, etc.).

  The B^+ Tree structure requires some overhead for insertion and deletion as well as some additional space.

  However, the additional overhead is justified for files that require frequent modification since the cost of file reorganization is reduced in this approach.

A typical node in a B^+ Tree is as shown on the next page:

B+ and B Tree Indexes (Cont.)

P_1	K_1	P_2	K_2	……	P_{n-1}	K_{n-1}	P_n

A node can hold up to n-1 key values K_1, K_2, … K_{n-1}, and n pointers P_1, P_2, … P_n.

The key values in a node are always maintained in a sorted order, i.e. $K_1 < K_2 < \ldots < K_{n-1}$

The range of key values in a node cannot overlap with the range of key values in another node.

That is, if a node has key values K_1 through K_5 and another node has key values K_6 through K_{10}, then K_1, K_2, … K_5 should be less than K_6, K_7, … K_{10}.

In a B^+ Tree, a key value can appear in a maximum of two levels: the leaf node level and one of the non-leaf node levels (including the root node).

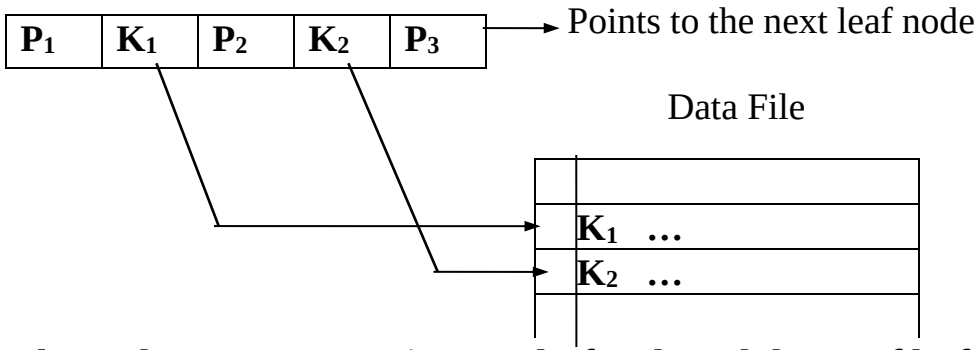
- **Characteristics of Leaf Nodes**

  Each pointer P_i in a leaf node points to a record in a data file with a key value K_i.

  The additional pointer P_n in a leaf node points to the next leaf node at the same level. This allows for efficient sequential processing of the file.

B+ and B Tree Indexes (Cont.)

- **Characteristics of Leaf Nodes (Cont.)**



Every key value must appear in some leaf node and the set of leaf nodes in effect form a dense index.

During construction of a B+ Tree, a leaf node can hold only $\lceil (n-1)/2 \rceil$ key values. But while inserting key values into an existing tree (after the initial construction is over), a leaf node can be filled up completely.

- **Characteristics of Non-Leaf Nodes**

The structure of non-leaf nodes is similar to that of leaf nodes, except that:

1. **The pointers point to the nodes below and not to data records in the data file.**
2. **Non-leaf nodes are not chained together across as in the case of leaf nodes.**
3. **Non-leaf nodes can be filled up with key values during construction until they have to be split up when there is no space in the node for inserting more key values**

If a non-leaf node contains m pointers ($1< i < m$), pointer P_i should point to the subtree containing key values less than K_i and greater than or equal to K_{i-1}.

B+ and B Tree Indexes (Cont.)

- **Characteristics of Non-Leaf Nodes (Cont.)**

Pointer P_m should point to the subtree containing those key values greater than or equal to K_{m-1}, and pointer P_1 should point to the part of the subtree containing those key values less than K_1.

Each node must hold at least $\lceil n/2 \rceil$ pointers and the root node must have at least 2 pointers.

- **Constructing a B^+ Tree**

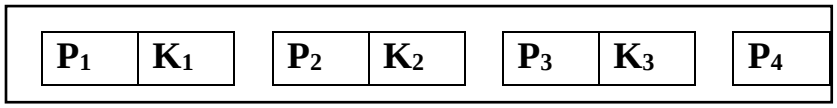
The set of search key values for which a B^+ tree should be constructed should be available first.

The first step in constructing a B+ tree is to sort the key values from low to high values.

Example: Construct a B+ tree using the following key values: 31, 2, 17, 11, 29, 5, 19, 7, 23, 3
Key values sorted from high to low are: 2, 3, 5, 7, 11, 17, 19, 23, 29, 31
Assume the no. of pointers in each node = 4, i.e. n = 4 (No. of pointers will always be given)

During construction, when the no. of key values in a leaf node exceed $\lceil (n-1)/2 \rceil$ key values, i.e. $\lceil (4-1)/2 \rceil = 2$, insert the next key value into a new leaf node, and connect the two leaf nodes together across.

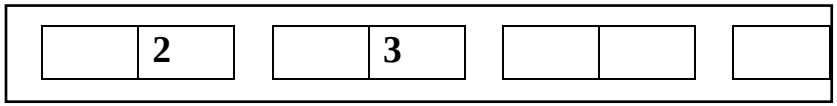
Then insert the new key value into the parent node. If the parent node does not exist, then create it first. Connect the pointers from the parent node to the leaf nodes.



P_1, P_2, P_3, P_4 are pointers
K_1, K_2 and K_3 are key values

B+ and B Tree Indexes (Cont.)

Begin construction by inserting the key value 2, and then insert the next key value 3 into a leaf node.



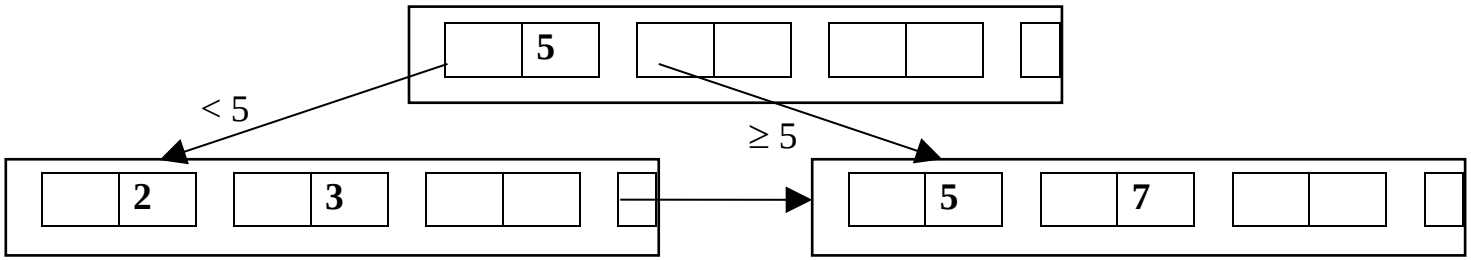
Since ⌈(4-1)/2⌉ = 2 key values, the next value to be inserted should go into a new leaf node.

The empty key value space in the leaf node is for inserting key values in the future.

Next insert key value 5 and then insert 7. This requires creating a new leaf node. Notice the key value goes into the new leaf node and a parent node has also been created.

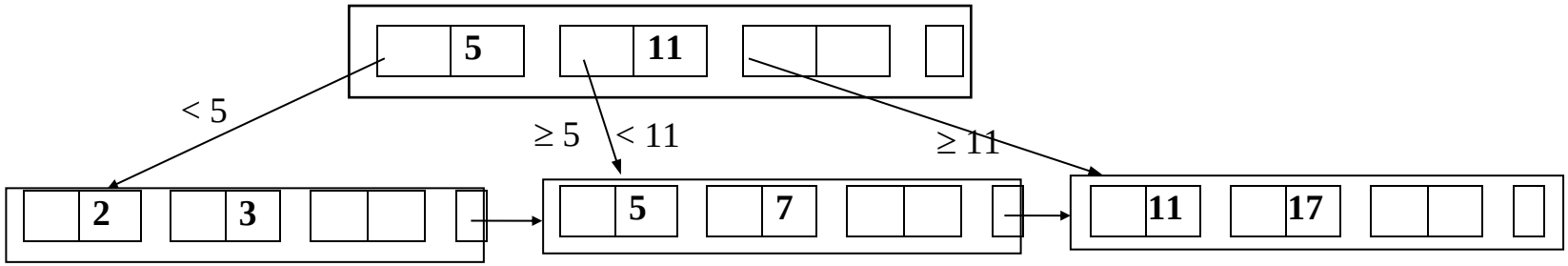
The new key value 5 is inserted into the parent node and the parent has been connected to the two leaf nodes with pointers.

Note the pointer to the left of the key value 5 in the parent node points to a leaf node with key values <5 and the pointer to the right of the key value 5 points to the leaf node with values ≥ 5

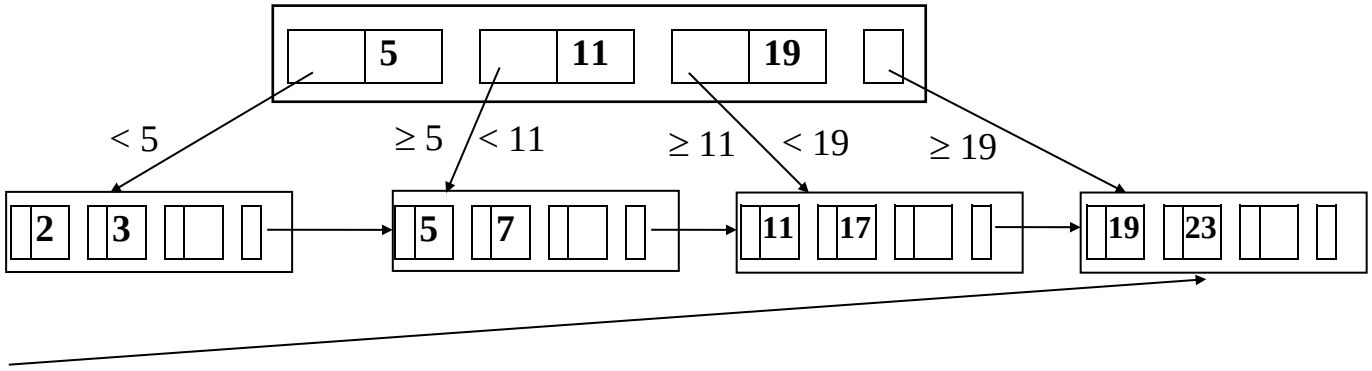


B+ and B Tree Indexes (Cont.)

For inserting 11, we must once again create a new leaf node and note the new key value 11 has been inserted into the existing parent node. We can also Insert 17 into the third leaf node.



To Insert 19, we must once again crate a new leaf node and repeat the procedure. <u>But note that the non-leaf node (parent node) is allowed to be filled with key values and need not be split up like a leaf node.</u>



Now we want to insert 29, but the non-leaf node (the root node) is full.

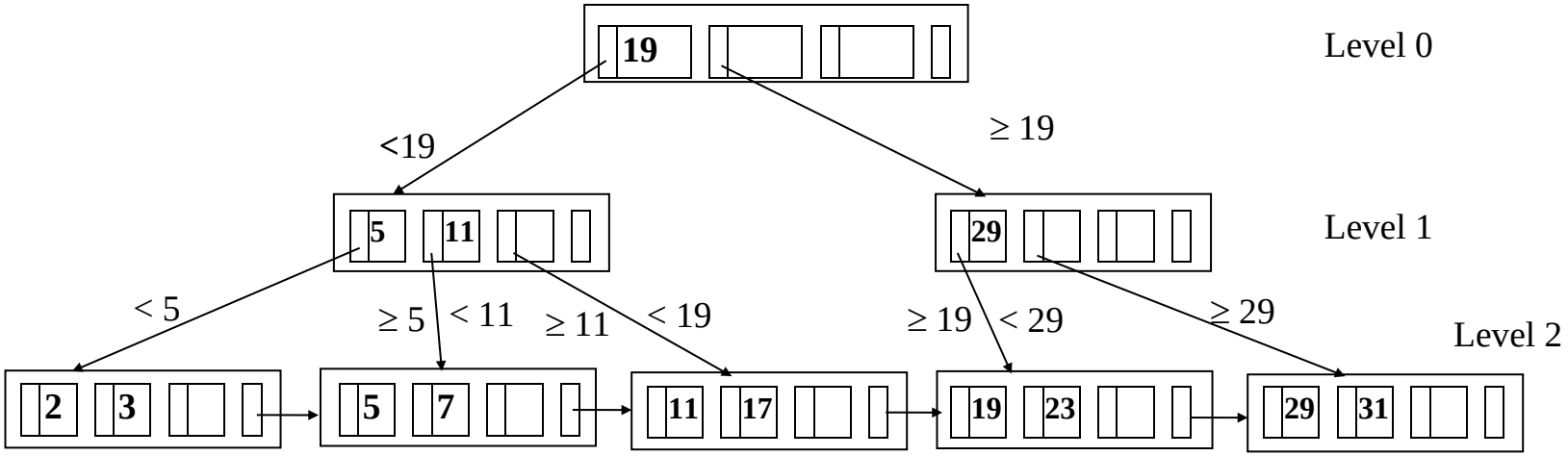
B+ and B Tree Indexes (Cont.)

Therefore, we will have to create new non-leaf nodes.

But to split a non-leaf node when it is full, the $\lceil (n-1)/2 \rceil$ rule should be applied.

$\lceil (n-1)/2 \rceil$ =2 key values should be kept in the current node. Therefore, the key values 5 and 11 are kept in the left non-leaf node in Level 1 and 19 is inserted into the root node.

The new key value 29 is inserted into the new leaf node and copied onto its parent node (non-leaf node).



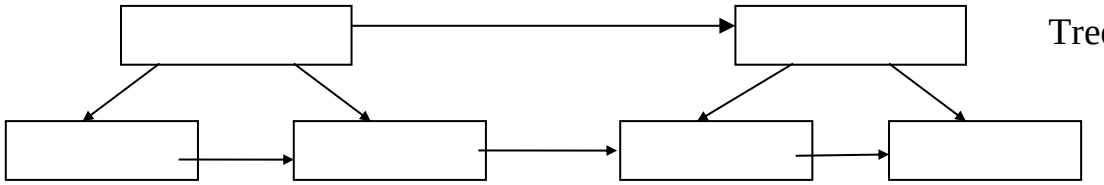
- **We do not link internal nodes or non-leaf nodes across because we want to search hierarchically from top to bottom by splitting the tree into two non-overlapping halves. So when an internal node is split, the key value split is moved up to the parent node and not to a node at the same level.**

B+ and B Tree Indexes (Cont.)

- **Common mistakes in drawing a B+ tree**

  The B+ tree must always have only one root node. There is no root node in the tree below.

  Non-leaf nodes must not be joined across. The example shows non-leaf nodes joined across



Tree needs only one root node

- **Searching for a particular key value and a record in an existing B+ tree**

  Example: Find the record with the key value 11 in the B+ tree constructed earlier.

  Search the root node first. Compare 11 with the key values in the root node.

  Since 11 is less than 19, we want to follow the pointer to the left of 19 and go to the level below (level 1)

  In level 1, we will compare 11 with the key values. Since there is a key value 11 there, we want to follow the pointer to its right to the leaf node below that contains key values ≥ 11.

  In the leaf node in level 2, we will search for the key value 11 and follow the pointer to its left to the data file containing the record with the key value 11.

B+ and B Tree Indexes (Cont.)

- **How can we go about retrieving all the records in a data file managed using a B+ tree**

  We will start with the root node. Follow the left most pointer to the next level and do the same in the next level and so on until we reach the leaf node level.

  Then from the left most leaf node, follow the pointers to the data file and retrieve the records.

- **Other Observations**

  All key values must be represented in the leaf nodes of a B+ tree.

  A key value can exist in only 2 levels in a B^+ tree and in not more than 2 levels.

  The root node should clearly separate the index structure into subtrees that do not overlap.

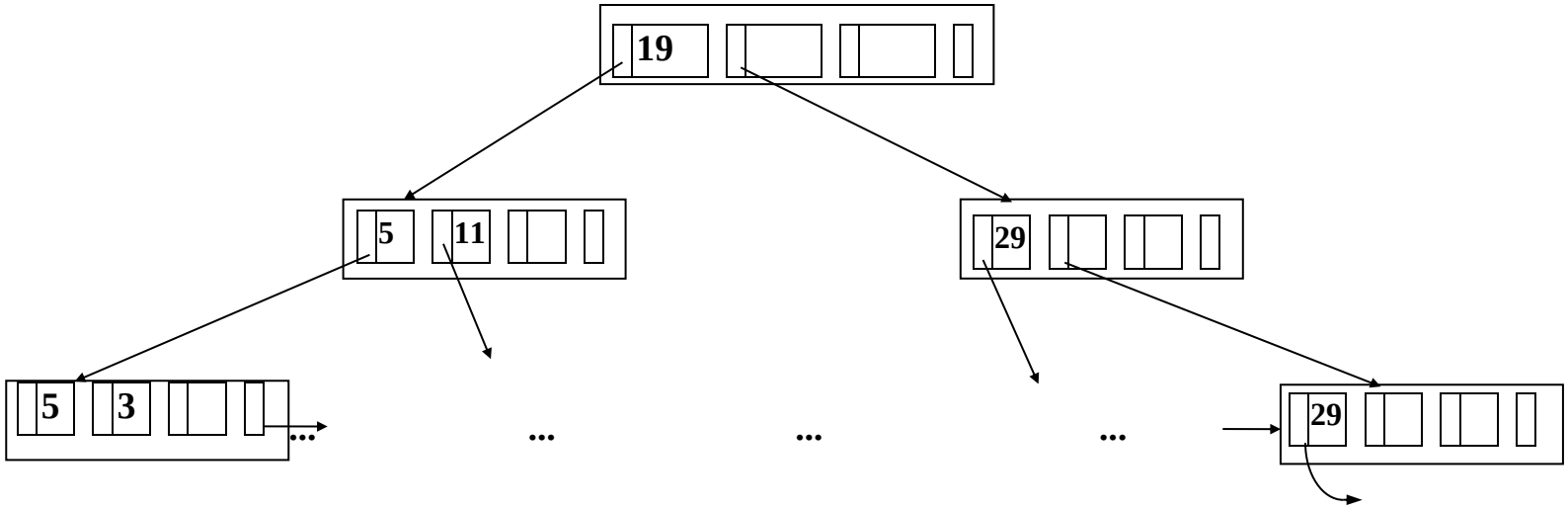
  The root node must have at least 2 pointers connecting at least 2 nodes below.

- **Deleting a record from an existing B^+ tree index structure**

  Delete 31:

  This is easy. All that we have to do is to delete the pointer to the particular record from the leaf node containing the key value 29 as shown in the figure below.

B+ and B Tree Indexes (Cont.)



Not all the leaf nodes are shown in the above figure to avoid repetition.

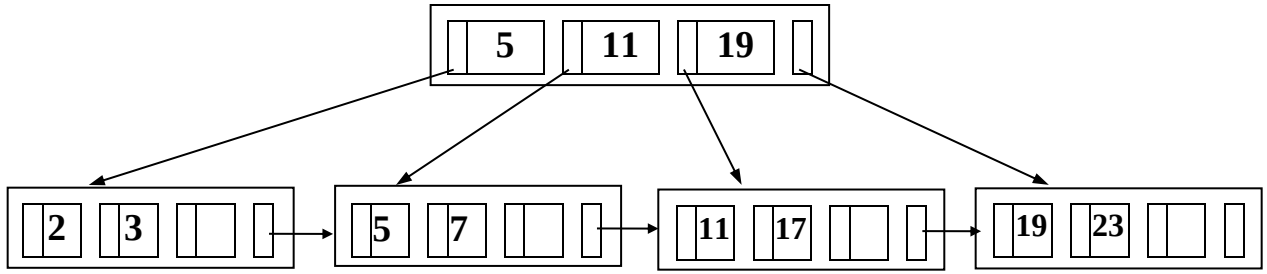
Next Delete 29:

If you delete 29, the entire leaf node has to be deleted and the pointer from the parent node.

Each non-leaf node should have at least $\lceil (n-1)/2 \rceil$ pointers connected to the nodes below.

Therefore, the tree has to collapse back again as follows:

B+ and B Tree Indexes (Cont.)



Now, if we delete 19 from the above structure, what will happen to the B+ Tree structure?

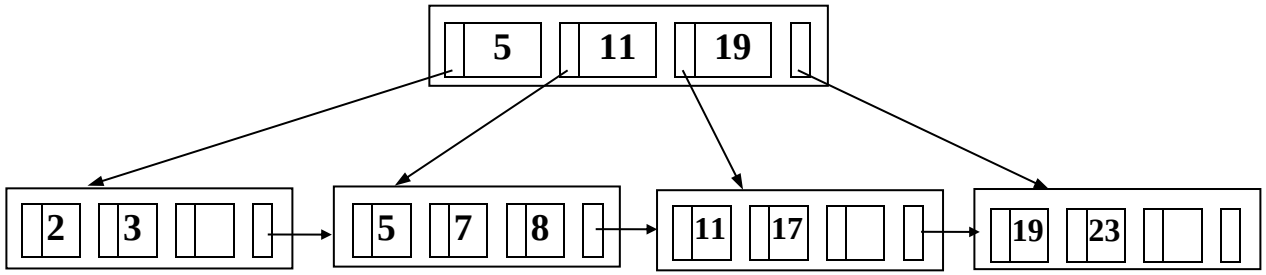
If you delete 5, what will happen to the B+ Tree structure?

- **Insertion into an existing (already constructed) B^+ Tree**

  Unlike construction, during insertion we can insert key values and fill up leaf nodes.

  Once a node is full, if we have to insert a new value then the node has to be split using the $\lceil (n-1)/2 \rceil$ rule.

  Insert 8 into the existing B^+ tree above: The result of the insertion is simple and is as follows:



B+ and B Tree Indexes (Cont.)

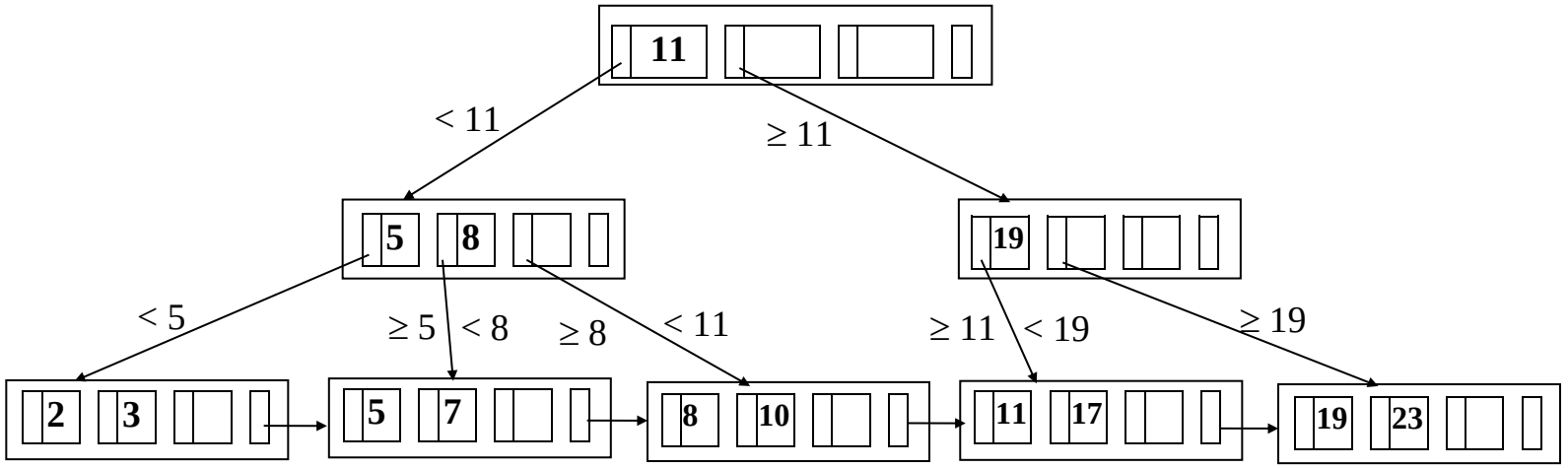
Insert 10 into the previous B+ Tree structure:

We can either attempt to insert into the second leaf node from left or we can insert it into the third leaf node from left.

Both have certain consequences. <u>But for the sake of consistency in this course, let us attempt to insert a key value into the node with a smaller value than the value to be inserted.</u>

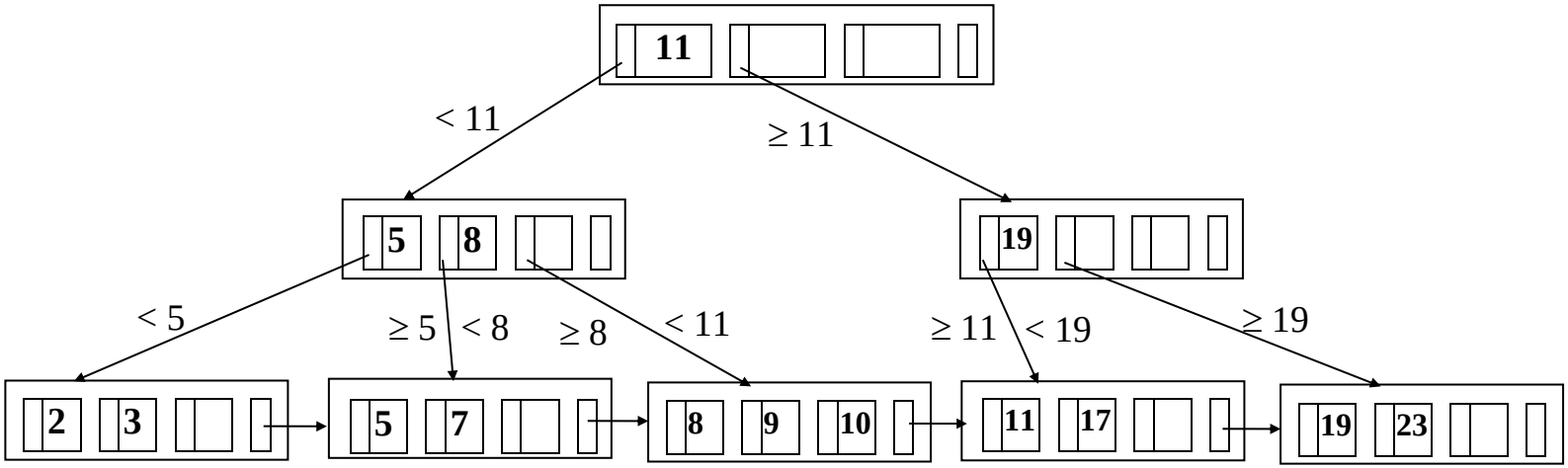
That is, insert 10 into the second leaf node from left which has the key value 8 (smaller than 10).

The resulting B+ structure will be as follows:



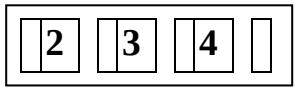
B+ and B Tree Indexes (Cont.)

Insert 9 into the previous structure. This is simple and 9 is inserted between 8 and 10 in the second leaf node. Note how the key value 10 has been moved to the right to occupy the empty key value space.



Insert 4:

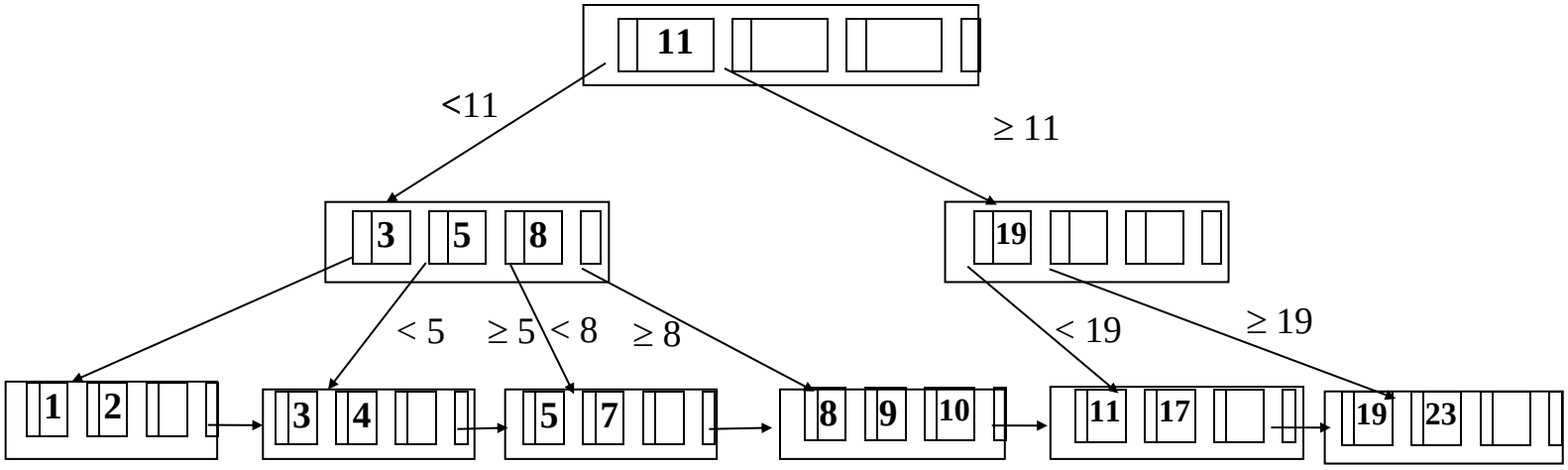
The first leaf node from the left will become as follows after the insertion.



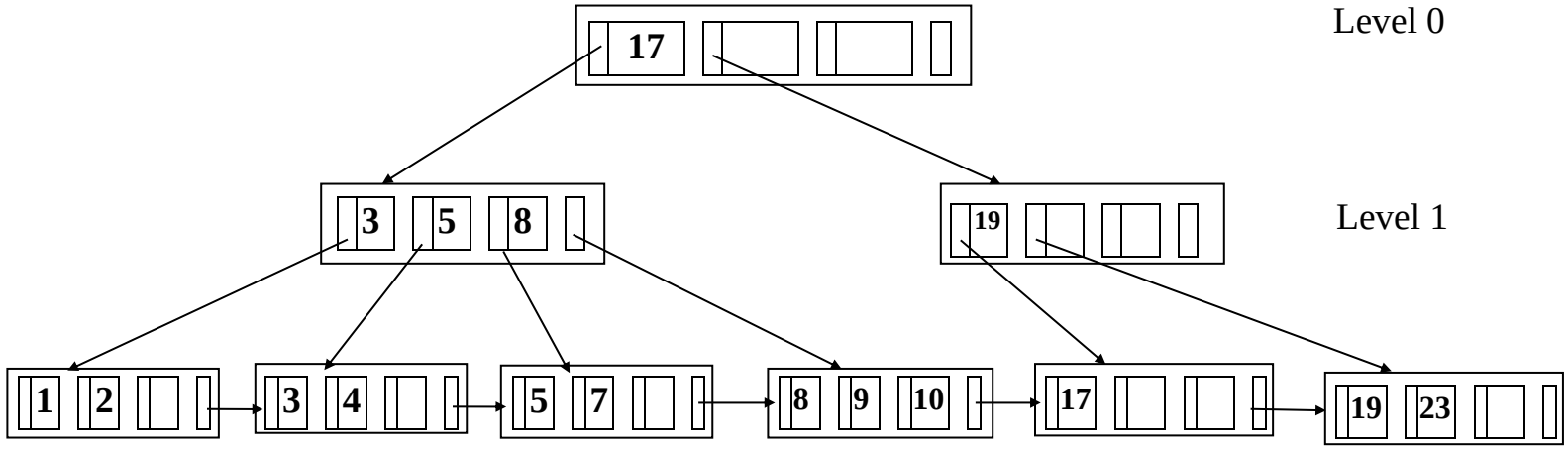
Insert 1:

Since 1 has to go before 2 and there are no other leaf nodes, the first leaf node should be split as follows:

B+ and B Tree Indexes (Cont.)



Delete 11 from the previous B+ tree structure: The most efficient solution is to replace 11 with the next key value, which is 17, in the root node as well as the leaf node as shown below.



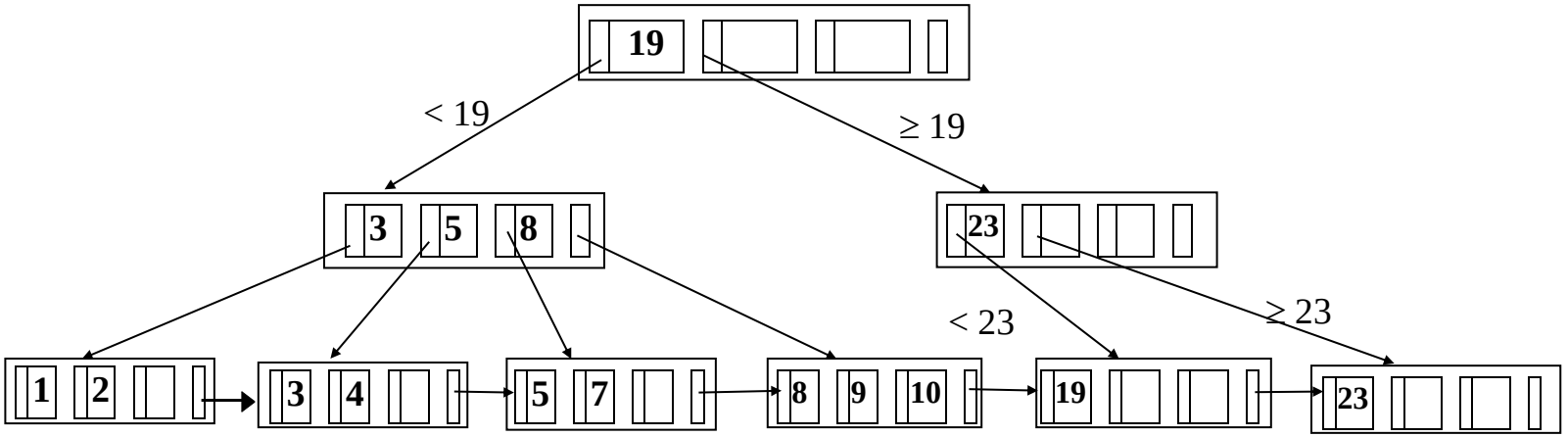
B+ and B Tree Indexes (Cont.)

What would happen if we delete 17 from the above structure?

Since 17 is in the root node, the key value must be gone from the root node.

The leaf node with the key value 17 will also be gone.

The resulting structure is shown below:



- **Key values need not be just numbers and in real life situation, key values can be letters**
- **Another Example: Construct a B^+ tree using the following key values, assuming n = 3 pointers**

  DeKalb, Aurora, Genoa, Kingston

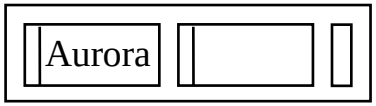
  What should be done first?

B+ and B Tree Indexes (Cont.)

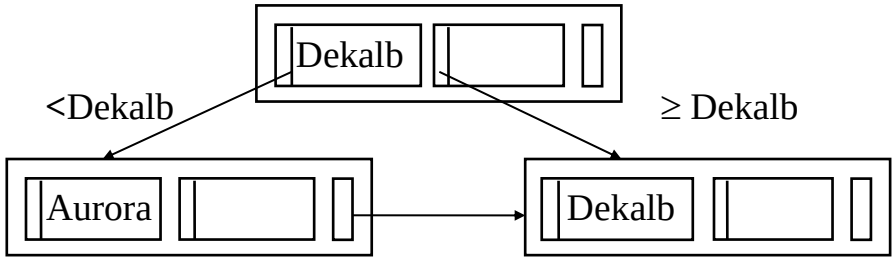
First sequence the key values in ascending order as shown below:
Aurora, DeKalb, Genoa, Kingston

Insert Aurora:

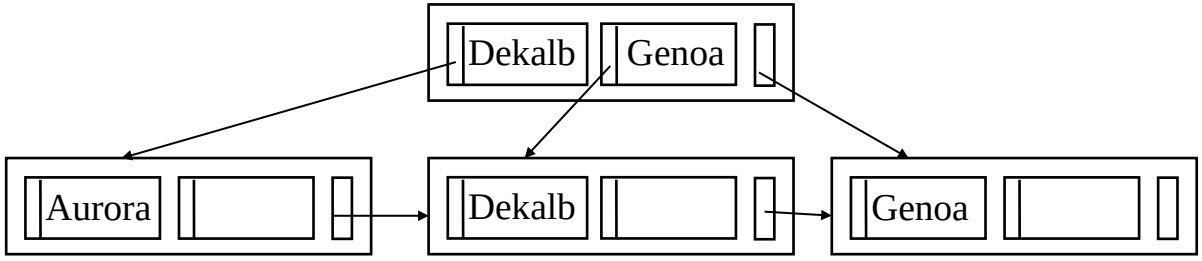
$$\left\lceil \frac{n-1}{2} \right\rceil = \left\lceil \frac{3-1}{2} \right\rceil = 1$$



Insert DeKalb:

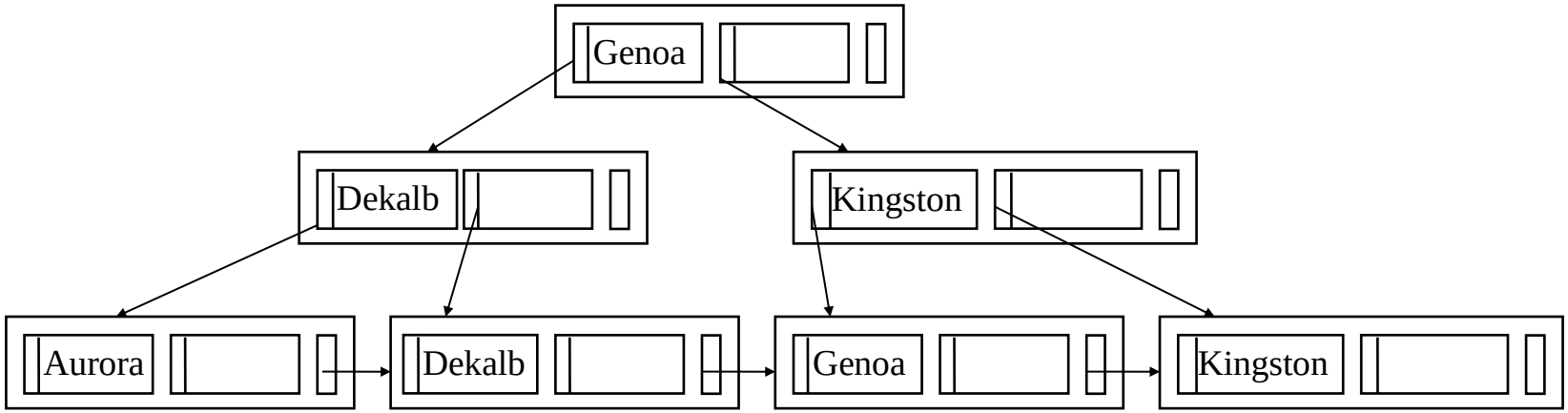


Insert Genoa:



B+ and B Tree Indexes (Cont.)

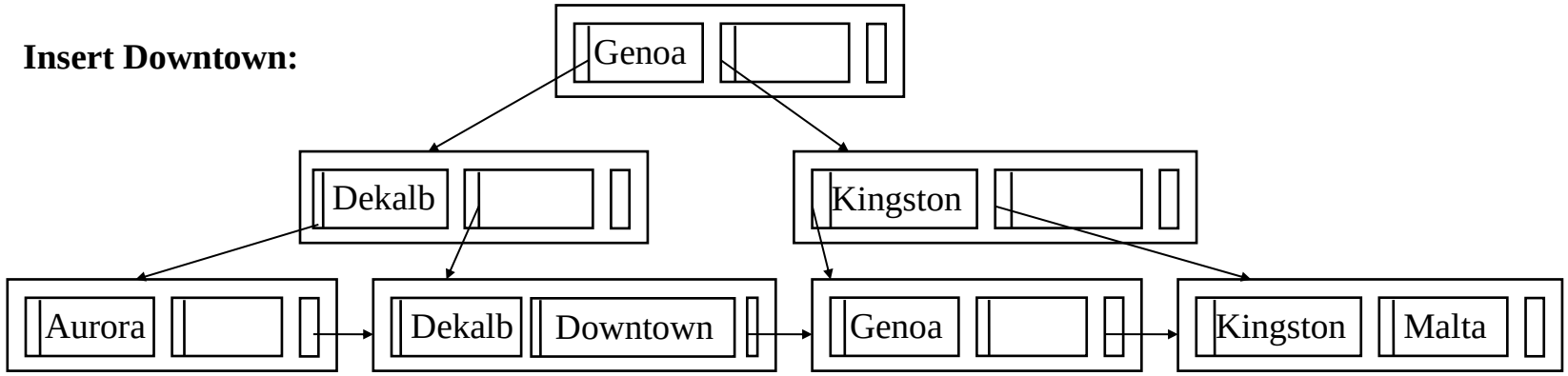
Insert Kingston:



- **Insert key values into the already constructed (existing) B^+ tree**

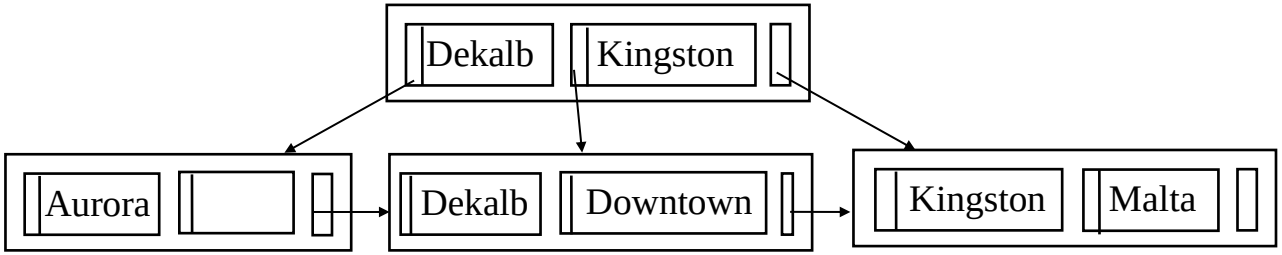
  Insert Malta: This is simple since there is an empty space after Kingston in the last leaf node.

  Insert Downtown:



B+ and B Tree Indexes (Cont.)

Delete Genoa from the B+ tree above: The tree will collapse as shown below.



- **IMPORTANT:**

For constructing a B+ tree, the key values must be sorted first from low to high

During construction, a leaf node can contain only $\lceil (n-1)/2 \rceil$ key values, and n (# of pointers) will be given

All key values must be in the leaf nodes joined across, and possibly (not more than 2) on a non-leaf node

There must be only one root node in a B+ tree, and non-leaf nodes must NOT be chained across with pointers

In a non-leaf node, every key value MUST have at least 2 pointers, one to the left and the other to the right

After construction, when inserting a key values into an existing tree, nodes can be filled up with key values

Pointer to the left of a key value on a non-leaf node must point to nodes with key values less than it

From the root node to find a key value in a leaf node, there must be the same number of steps (Balanced tree)

B+ and B Tree Indexes (Cont.)

- **B Trees**

  B+ trees have redundant indexes. That is, some of the key values are represented both in the leaf node and non-leaf nodes.

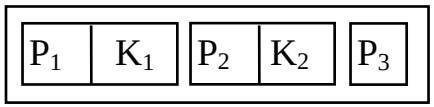
  But the B tree structure is slightly different from B+ trees.

  B tree allows a search key value to appear only once in a tree.

  Since search key values are not repeated in a B tree, we are able to store the index using fewer nodes.

  However, we need additional pointers for each search key in a non-leaf node. These additional pointers are the bucket pointers for the associated search key.

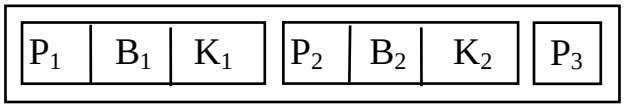
  The structure of leaf nodes in a B tree is as follows:



For n = 3 pointers

Where P_i is the pointer to the data record containing the key value K_i.

The structure of a non-leaf node is as follows and is different that of a B+ tree.



B+ and B Tree Indexes (Cont.)

P_i is the pointer to the index node below.

B_i is the pointer to the data record containing the key value K_i. and so a record with a particular key value can be located directly from a non-leaf node, unlike in the B+ tree.

- **Note: the leaf nodes of a B tree are not chained together since the non-leaf nodes can directly point to a data record and because all key values are not represented in the leaf nodes**

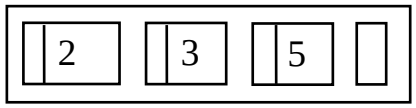
  Each leaf node during the construction of a B tree can hold up to n-1 key values, unlike B+ trees.

  If a leaf node is full, then create a new leaf node and leave $\lceil \frac{n-1}{2} \rceil$ key values in the old leaf node and move $\lceil (n-1)/2 \rceil$ +1th key value into the parent node and the remaining key values into the new leaf node.

Example: Construct a B tree for the following search key values: 31, 2, 17, 11, 29, 5, 19, 7, 23, 3

The sorted key values from low to high are: 2, 3, 5, 7, 11, 17, 19, 23, 29, 31.
Assume n = 4 pointers

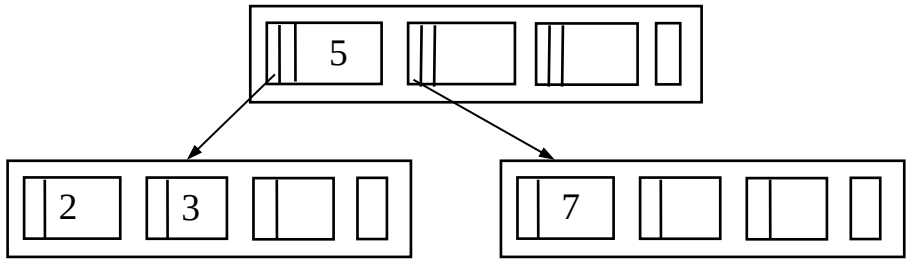
Create a leaf node with 4 pointers and insert 2, 3, 5 as shown below:



Note: The key value spaces have been filled up completely as this is a B tree. But to insert the next key value, the node must be split up.

B+ and B Tree Indexes (Cont.)

Insert 7: Since the leaf node is full, it should be now split up according to the $\lceil \frac{n-1}{2} \rceil$ rule.



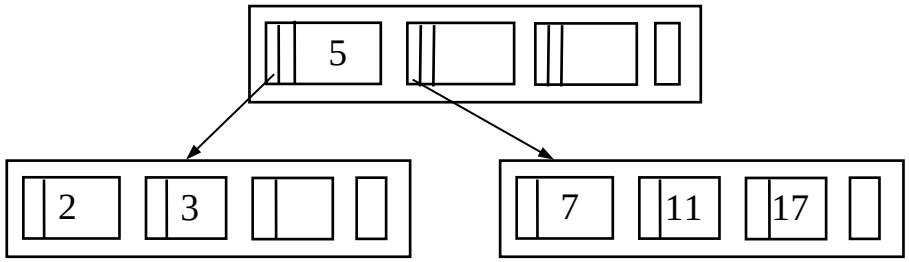
Leave $\lceil \frac{n-1}{2} \rceil$ key values in the leaf node,

i.e. $\lceil \frac{4-1}{2} \rceil = 2$

Move the $\lceil \frac{n-1}{2} \rceil$+1 th key value into the parent node.

Move the remaining key values into the new leaf node.

Insert 11, 17

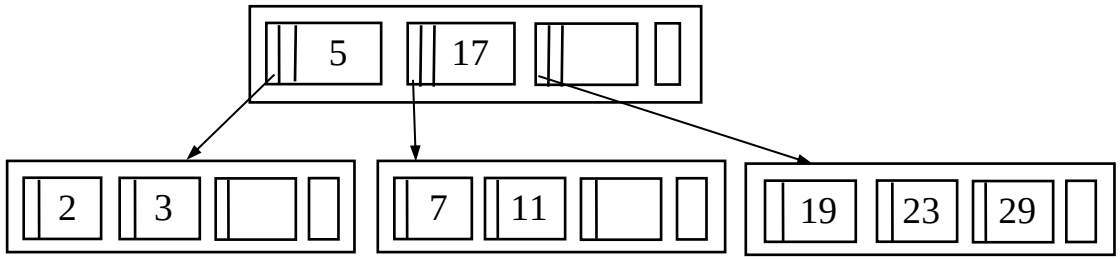


Note the following so far:

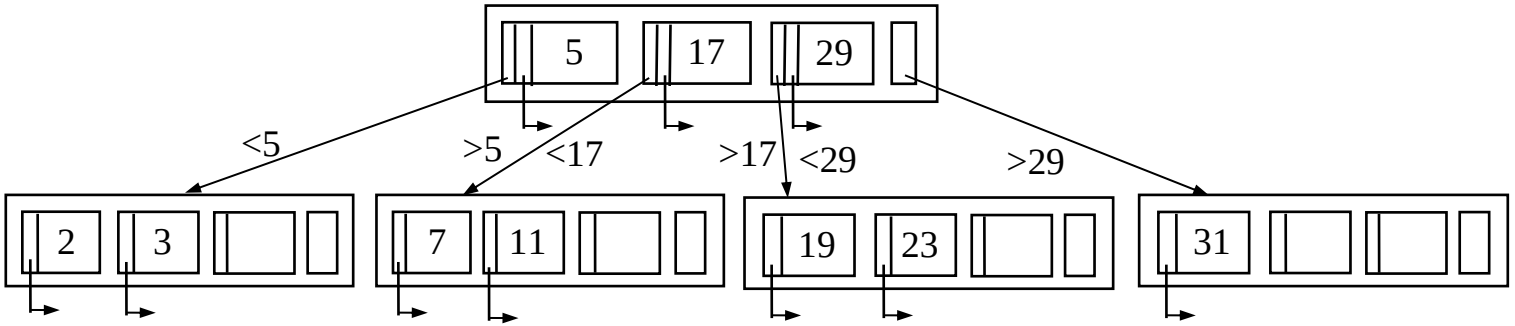
- **In the non-leaf node, a key value has two pointers to its left. The first pointer points to the node below and the second pointer points directly to the data record in the data file containing the key value**
- **The leaf node has only one pointer to the left of a key value and this pointer points to the data record in the data file containing the same key value. Leaf nodes are also NOT joined across.**

B+ and B Tree Indexes (Cont.)

Insert 19:



Insert 23, 29, 31:



<u>Remember</u>: In a B tree structure, nodes are allowed to be filled up during construction until a new key value has to be inserted into the node.

Leaf nodes are NOT chained across with pointers and a <u>key value can exist in only one node in the tree</u>.

The structure of non-leaf nodes is different than that of leaf nodes.

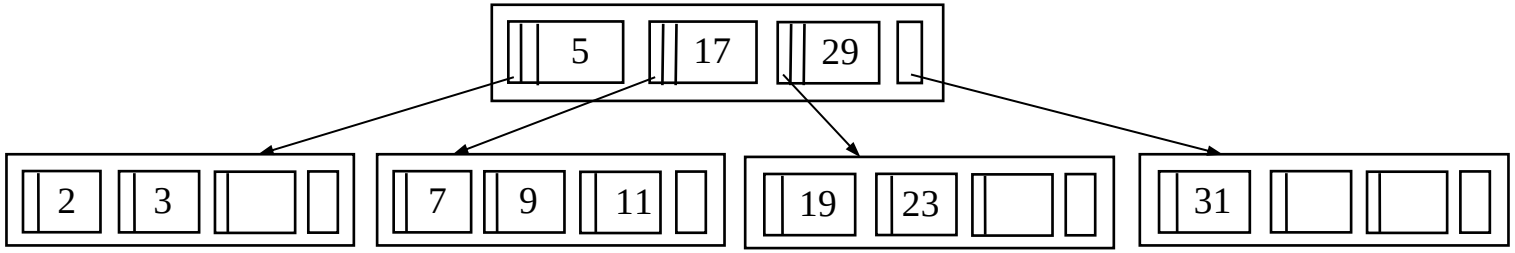
In a leaf node, a key value has only one pointer but a key value has two pointers in a non-leaf node.

B+ and B Tree Indexes (Cont.)

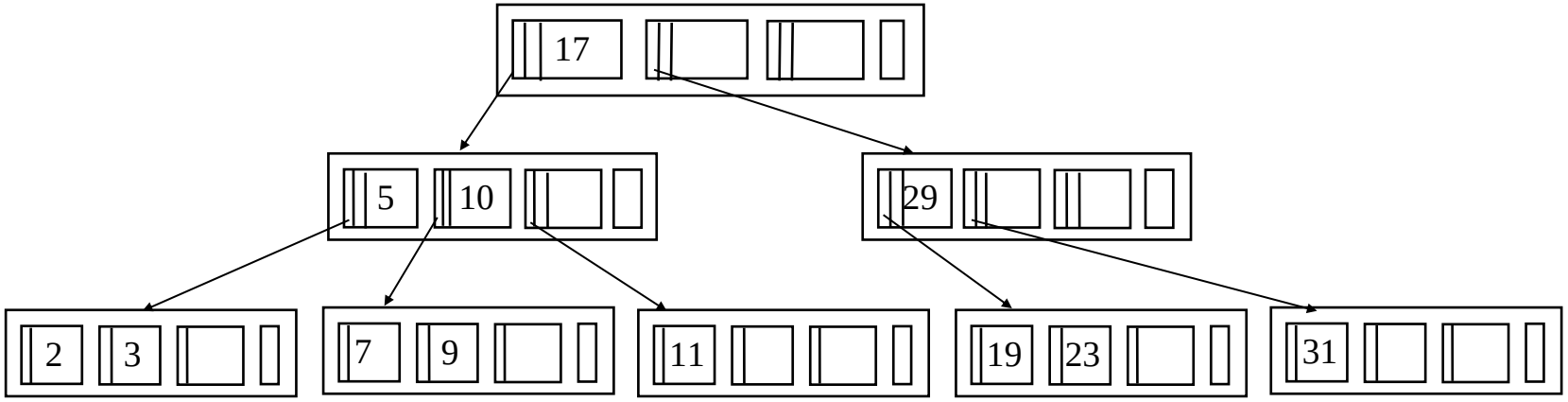
- **Inserting key values into an existing B tree**

  While inserting key values a node in B tree can be filled up completely. To insert another key value into a full node, the node has to be split up according to the $\lceil \frac{n-1}{2} \rceil$ rule.

  Insert 9:



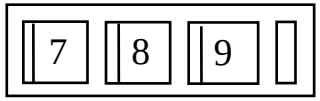
  Insert 10:



B+ and B Tree Indexes (Cont.)

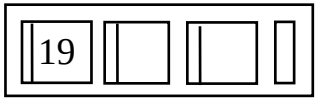
Insert 8:

The second leaf node from the left will become:



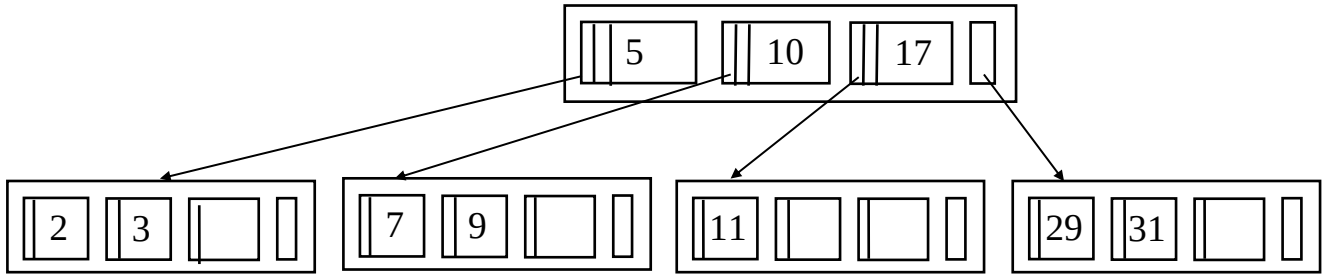
Delete 23

The second leaf node from right will become:



Delete 19:

This will cause the second leaf node from right to disappear and the tree will collapse as follows:



B+ and B Tree Indexes (Cont.)

- **IMPORTANT: Remember the following**

  1. **In a B+ tree, leaf nodes are chained across but in a B tree leaf nodes are not chained across.**

  2. **Do not assume because you can "see" there is an empty key space in a node you can insert a key value there. Think of the DB software following a consistent (programmed) approach for the operations.**

  3. **Each key value in a non-leaf node should have a pointer to its left and another pointer to right. Otherwise, you will have an imbalanced tree. Root node should have at least two points going from it.**

  4. **During construction, a leaf node can have only $\lceil (n-1)/2 \rceil$ key values in a B+ Tree. A leaf node in a B Tree can be filled up during construction.**

  5. **The leaf and non-leaf nodes are different in a B Tree as each non-leaf node has 2 pointers for each key.**

  6. **Do not draw B+ or B trees, their nodes, key values and pointers sloppily but draw them neatly.**

- **B+ and B Trees Summary**

  B^+ trees and B tree allow efficient construction, sorting, searching, and manipulation of indexes.

  The procedures applied for splitting nodes, decision on no. of pointers (n value), etc., are dependent on the software developer.

  The procedures shown in this section are for illustrative purposes only, and almost all database systems use some form of a proprietary B^+ tree indexing structure for managing data records.

