B+ Trees

By Satyam Kumar (160123035)

Lecture on 14 March, 2019 by Dr. Amit Awekar

B+ Trees:

B+ tree is a storage method in a tree like structure. B+ tree has one root, any number of intermediary nodes and a leaf node. Here all leaf nodes will have the actual records stored. Intermediary nodes will have only pointers to the leaf nodes and it does not have any data. Any node will have only two leaves. This is the basic of any B+ tree.

If we have to search for any record, they are all found at leaf node. Hence searching any record will take same time because of equidistance of the leaf nodes. Also they are all sorted. Hence searching a record is like a sequential search and does not take much time.

Insertion in a B+ tree:

Assumptions:

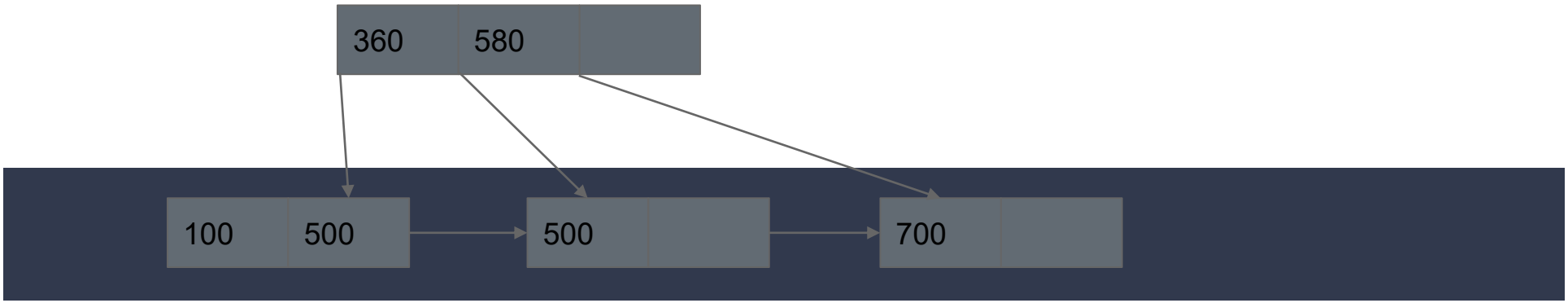
1. Everytime while splitting, more entries are given to the left than to the right.
2. While searching for node where key is to be placed, we go left on less than as well as equal to match with root while root on greater than case.

Insertion of 100: 100 is inserted by creating leaf node.

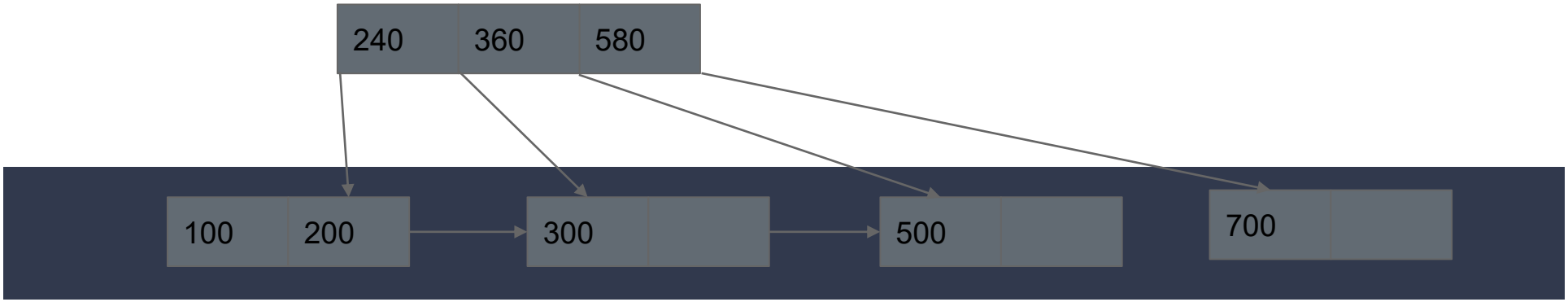
Insertion of 500: There is a vacant space available beside 100. 500 is inserted there.

Insertion of 700: No vacant space is there. We need new leaf nodes, so we do splitting following our convention of giving more entries to the left ie., into (100, 500) and (700, NULL). Any value between 500 and 700 can be chosen in place of 580(it is not actual data).

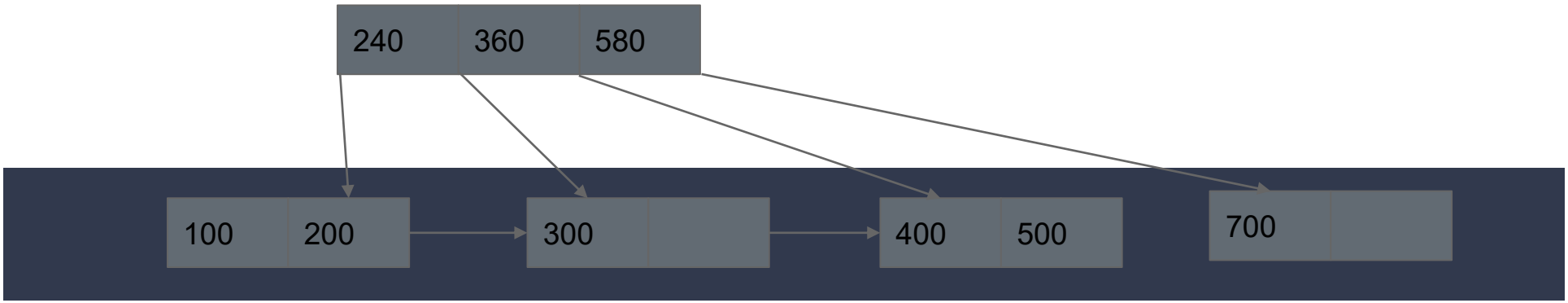
Insertion of 200: 200 needs to be inserted in the left of 580. But no vacant space is there. So we do splitting following our convention of giving more entries to the left ie., into (100, 200) and (500, NULL). Again, any value between 200 and 500 can be used in place of 360(it is not actual data).



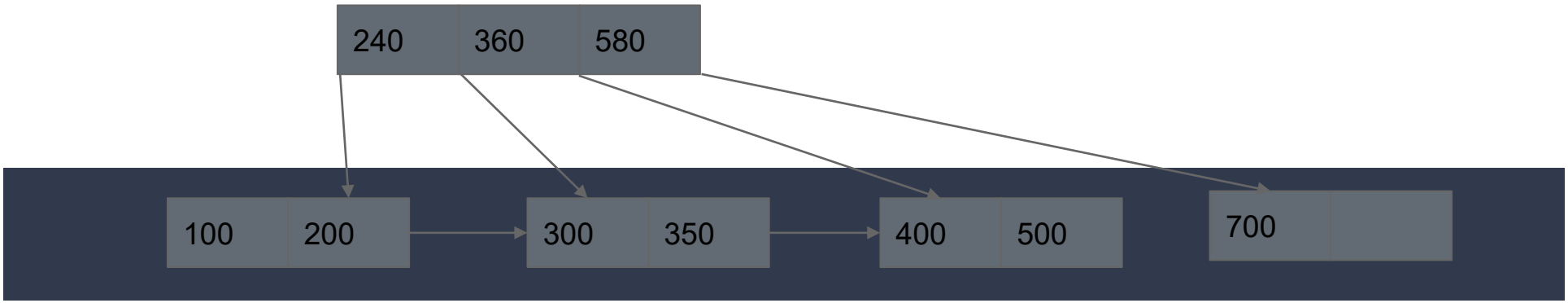
Insertion of 300: 300 needs to be inserted in the left of 350. But no vacant space is there. So we do splitting following our convention of giving more entries to the left ie., into (100, 200) and (300, NULL). Again, any value between 200 and 500 can be used in place of 240(it is not actual data).



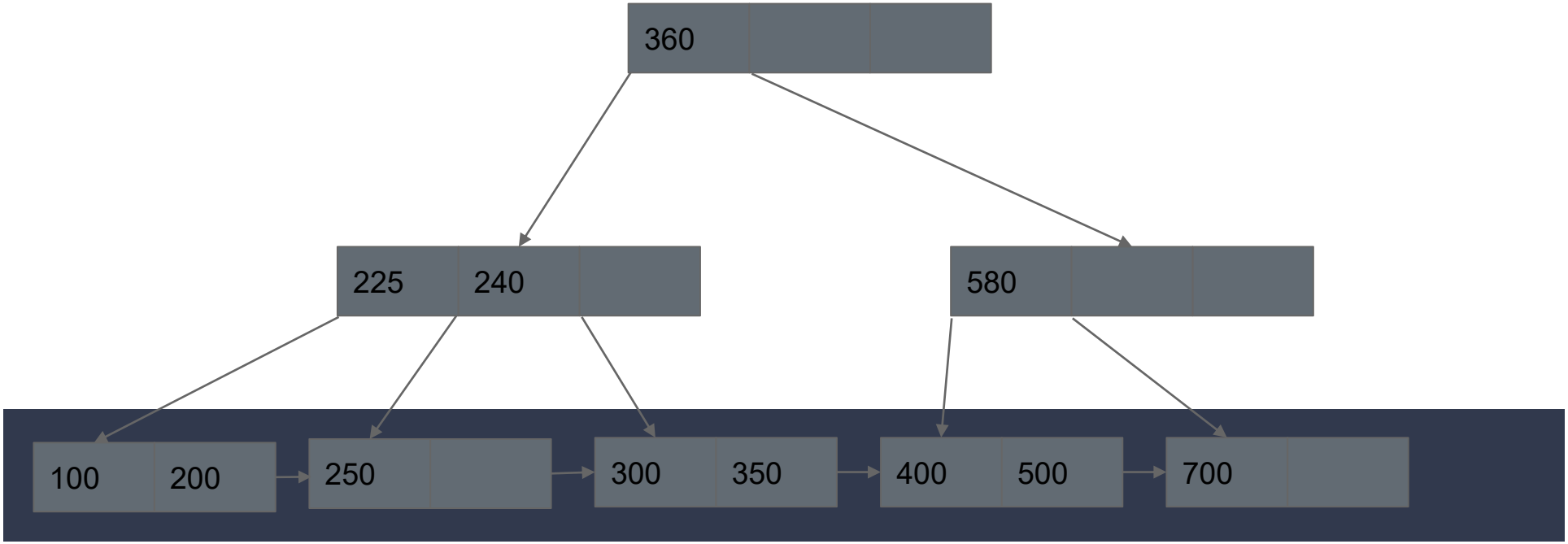
Insertion of 400: 400 needs to be inserted between 320 and 580. There is a vacant space. So, we insert 400 beside 500.



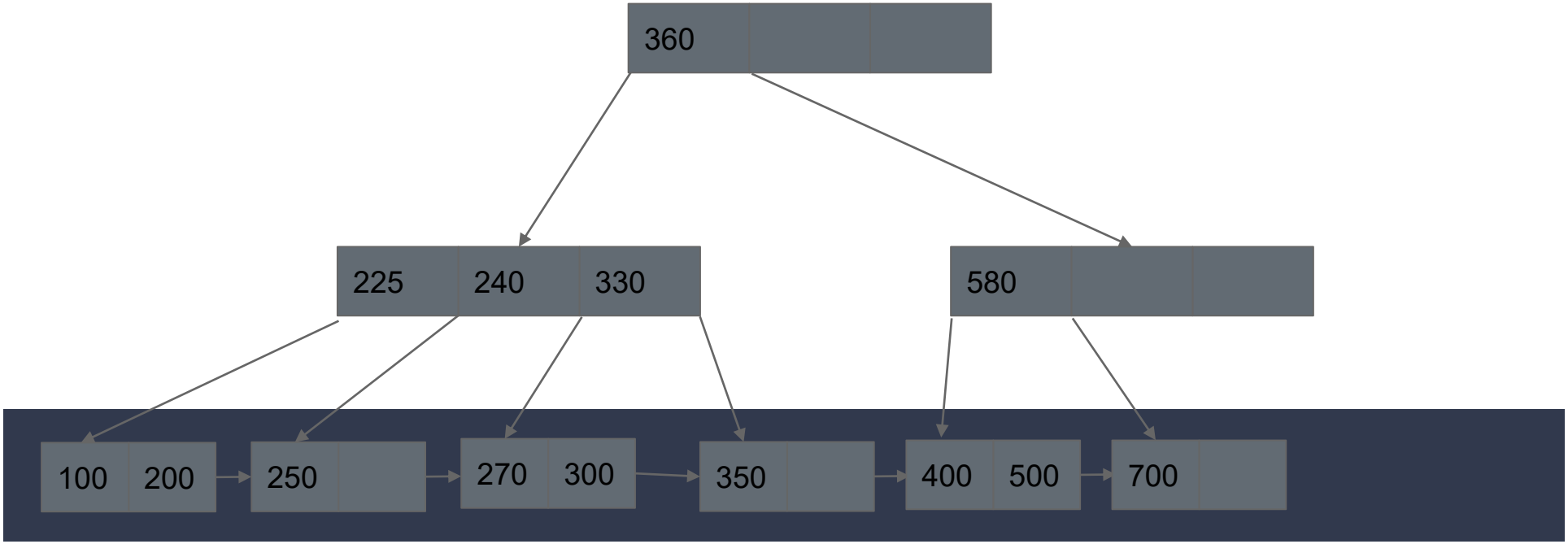
Insertion of 350: 350 needs to be inserted between 360 and 580.
There is a vacant space. So, we insert 350 beside 300.



Insertion of 250: 250 needs to be inserted between 360 and 580. But there is no vacant space. So we split into (100,200) and (250,NULL) with 225 in the above level, but we'll have to split here also since it is full. 360 is pushed to make root (350,NULL,NULL) at higher level.

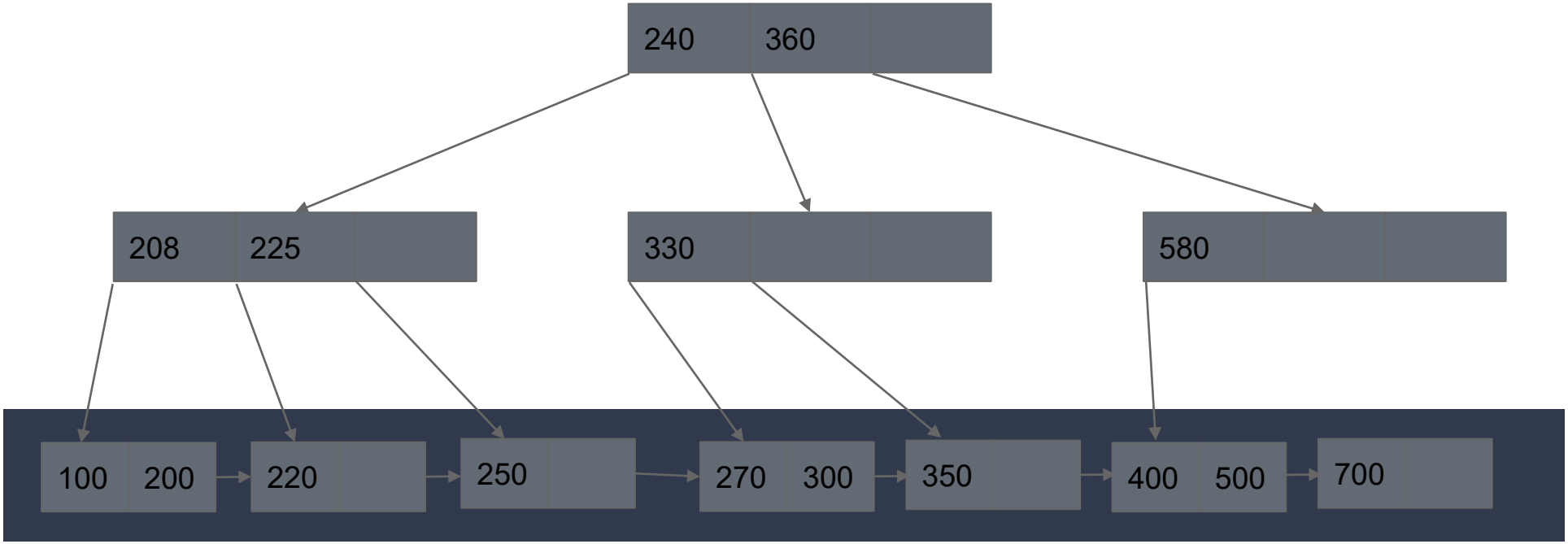


Insertion of 270: 270 needs to be inserted between 250 and 300. But there is no vacant space. So we split into (270,300) and (350,NULL) with 330 in the above tree level.



Insertion of 220: 220 needs to be inserted between 250 and 300. But there is no vacant space. So we split into (100,200) and (200,NULL) with 208 in the above level, but we'll have to split here also since it is full. 240 is pushed to make root (350,NULL,NULL) at higher level and (208, 225, NULL) and (330, NULL, NULL) are the results of the split.

And below is what we get after inserting all the given data.



Pros:

- As the file grows in the database, the performance remains the same. This is because all the records are maintained at leaf node and all the nodes are equidistant from root. In addition, if there is any overflow, it automatically re-organizes the structure.
- Leaf node allows only partial/ half filled, since records are larger than pointers.

Cons:

- If there is any rearrangement of nodes while insertion or deletion, then it would be an overhead. It takes little effort, time and space. But this disadvantage can be ignored compared to the speed of traversal

B+ Trees

By Rajat Paliwal (160123046)

Lecture on 14 March, 2019 by Dr. Amit Awekar

h1	h0				
000	00	2	8		S
001	01	33	17		
010	10	4	6	10	
011	11				

Proceeding in a similar Way.

B+ Trees :

A **B+ tree** is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children

A B+ tree can be viewed as a B-tree in which each node contains only keys (not key–value pairs), and to which an additional level is added at the bottom with linked leaves.

The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context — in particular, filesystems. This is primarily because unlike binary search trees, B+ trees have very high fanout (number of pointers to child nodes in a node,[1] typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

1. **While splitting, more entries are given to the left than right.**

2. **While searching for node where key is to be placed, we go left for less than and equal to case with root and to the right if greater.**

3. **Leaf Nodes have Even Number of keys and others have odd.**
4. **B+ trees inherit the B tree properties**

B+ Tree Insertion :

Algorithm :

- Perform a search to determine what bucket the new record should go into.
- If the bucket is not full (at most entries after the insertion), add the record.
- Otherwise, *before* inserting the new record
  - split the bucket.
    - original(Left)node has ⌈(L+1)/2⌉ items
    - new node has ⌊(L+1)/2⌋ items
  - Move ⌈(L+1)/2⌉ -th key to the parent, and insert the new node to the parent.
  - Repeat until a parent is found that need not split.
- If the root splits, treat it as if it has an empty parent and split as outline above.

B-trees grow at the root and not at the leaves.

Eg:

Insert 45 , 77 , 60 , 33 , 100 , 14 , 11 , 55

Insert 45

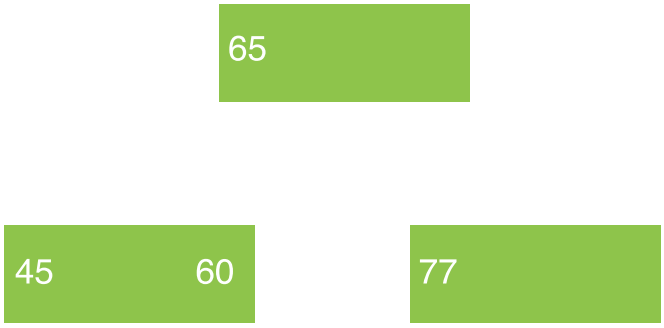
Simply insert 45 in leaf node.

Insert 77

Simply insert 77 in leaf node.

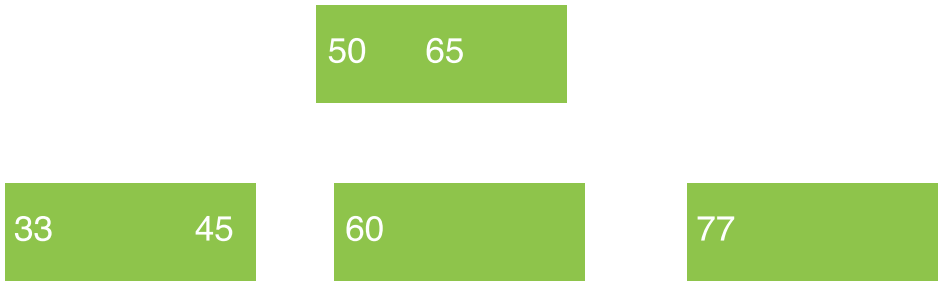
Insert 60

Split the leaf node and add a parent node with appropriate separator(65).



Insert 33

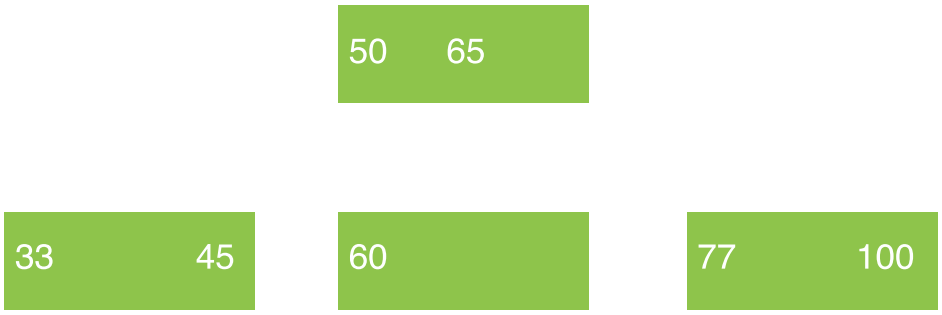
33 needs to be at the left of 65 so split is required at leaf level and add an appropriate separator(50).



Insert 100

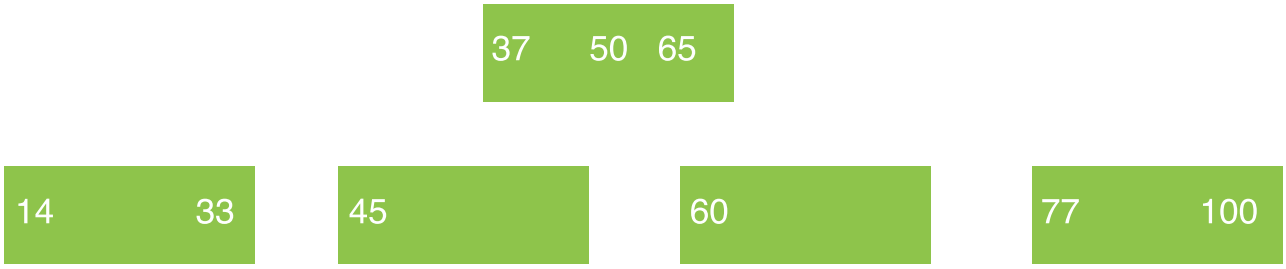
Simply insert 100 in the appropriate leaf node after travelling the tree.

As there is vacancy there.



Insert 14

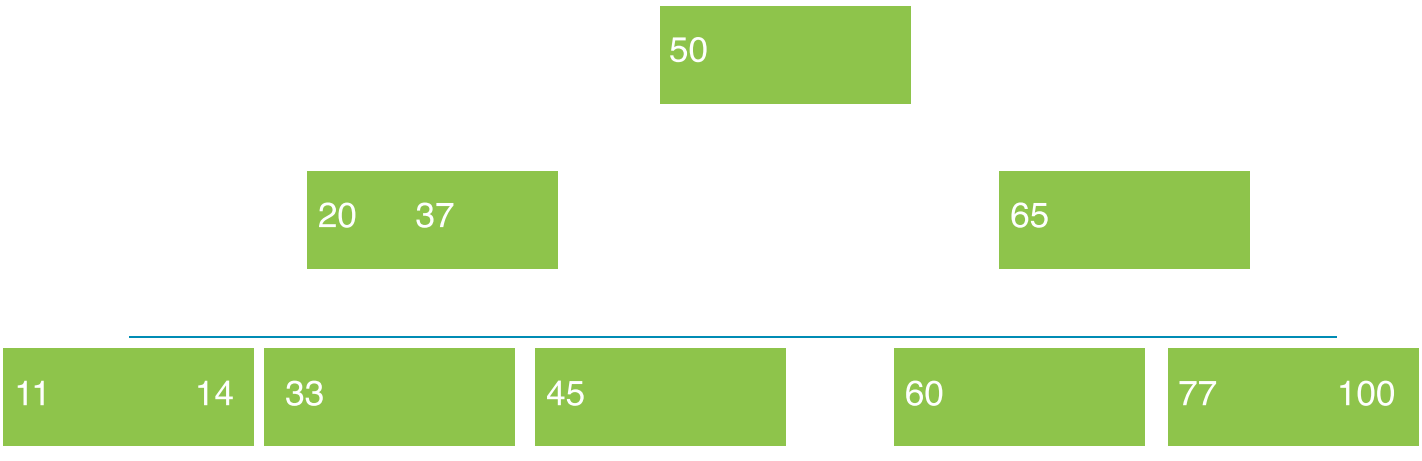
14 needs to be at the left of 50 and vacancy is not there so split is required at leaf level and add an appropriate separator(37).



Insert 11

11 needs to be at the left of 37 and vacancy is not there so split is required at leaf level and add an appropriate separator(20).

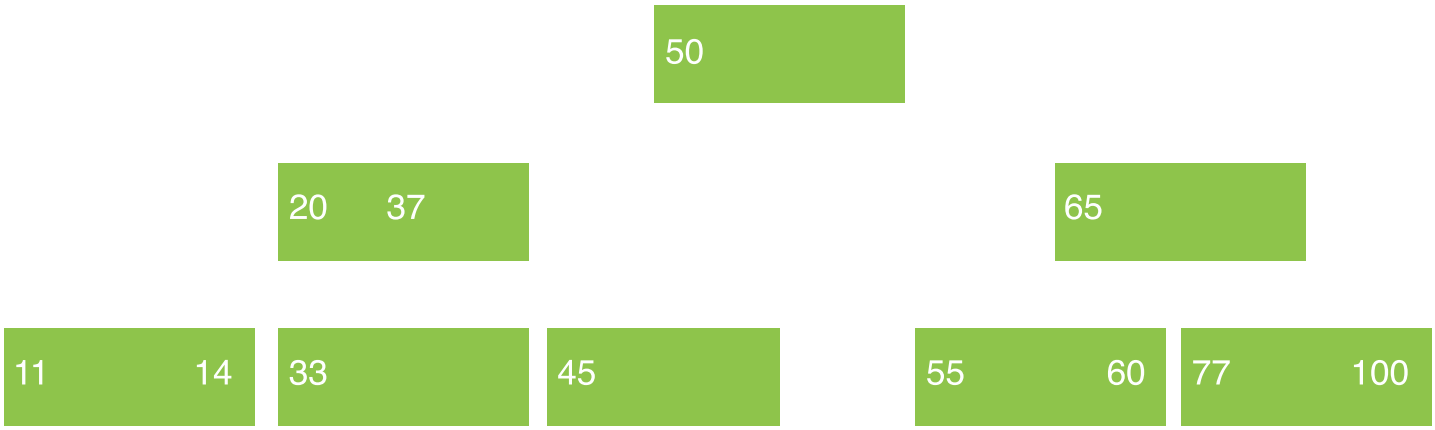
A new root is created as the parent node is full.



Insert 55

Simply insert 55 in the appropriate leaf node after travelling the tree.

As there is vacancy there.



B+ trees over B trees ?

In order, to implement dynamic multilevel indexing, B-tree and B+ tree are generally employed. The drawback of B-tree used for indexing, however is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree. Thus, the structure of leaf nodes of a B+ tree is quite different from the structure of internal nodes of the B+ tree. It may be noted here that, since data pointers are present only at the leaf nodes, the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to access them. Moreover, the leaf nodes are linked to provide ordered access to the records. The leaf nodes, therefore form the first level of index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the searching of a record.

B+ Trees

By Shubham Goel (160101083)

Lecture on 14 March, 2019 by Dr. Amit Awekar

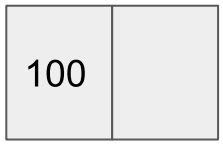
B+ Tree

A B+ tree is a data structure often used in the implementation of database indexes. Each node of the tree contains an ordered list of keys and pointers to lower level nodes in the tree. These pointers can be thought of as being between each of the keys. In a **B+ tree**, data is only found in leaf nodes. Deletion of leaf nodes is easy. **B+ trees** store redundant search key but **B tree** has no redundant value.

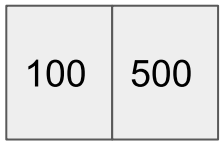
B+ Tree Insertion (Conventions followed)

1. Maximum number of keys in leaves (where data is stored) is 2 while maximum number of keys in non leaf nodes is 3.
2. Everytime while splitting, more entries are given to the right than to the left.
3. While searching for node where key is to be placed, we go left on less than root while right on greater than as well as equal to case.
4. Whenever a new entry is to be added to a full leaf node, the decision where to split is influenced by the above rule while the new value to be inserted in the tree above is 3/4th of the numbers where we are splitting.If 3/4th is greater than right value than right value is considered.

INSERT 100 : 100 is inserted into the leaf node.



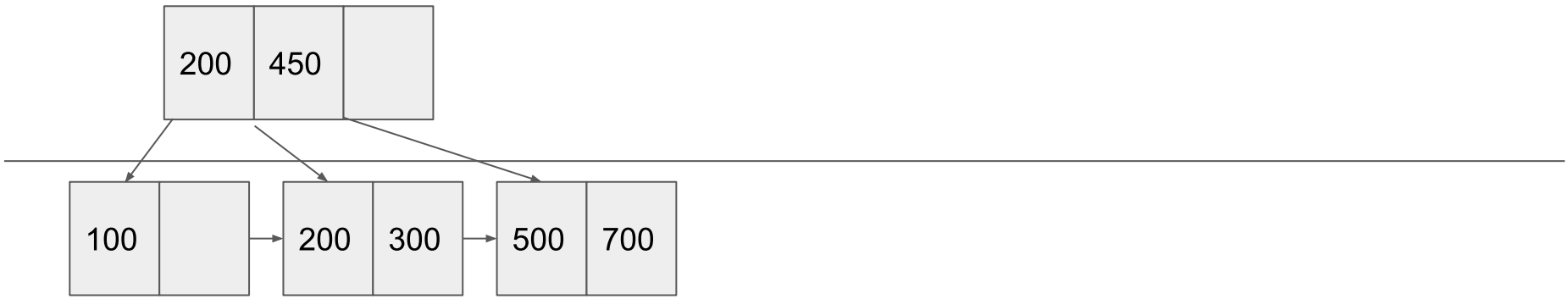
INSERT 500 : 500 is entered in empty space.



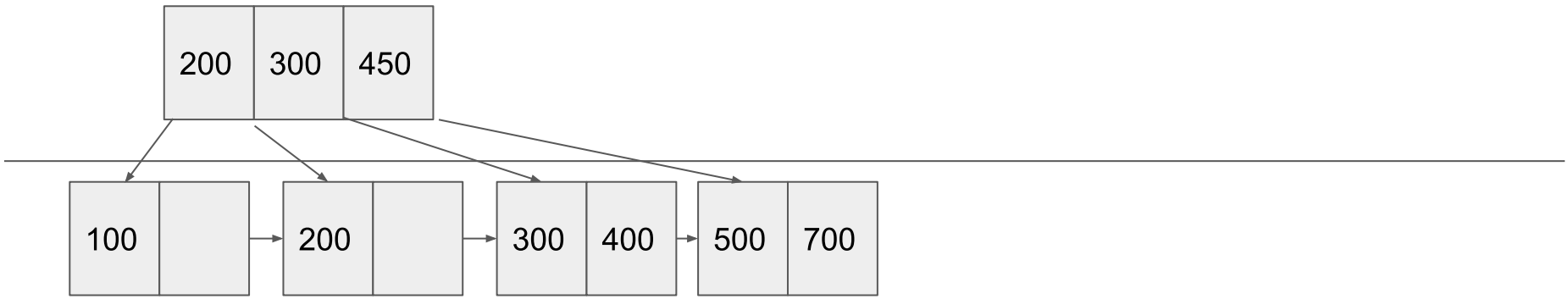
INSERT 700 : When 700 is inserted overflow takes place and new leaf node is entered. By splitting conveention left leaf nodes get 100 while right leaf node gets 500 and 700. While parent element is 3/4(100+500)=450.

INSERT 200 : 200 must go in leaf node (100,NULL) by convention of B-Tree and there is an empty space so 200 is inserted.

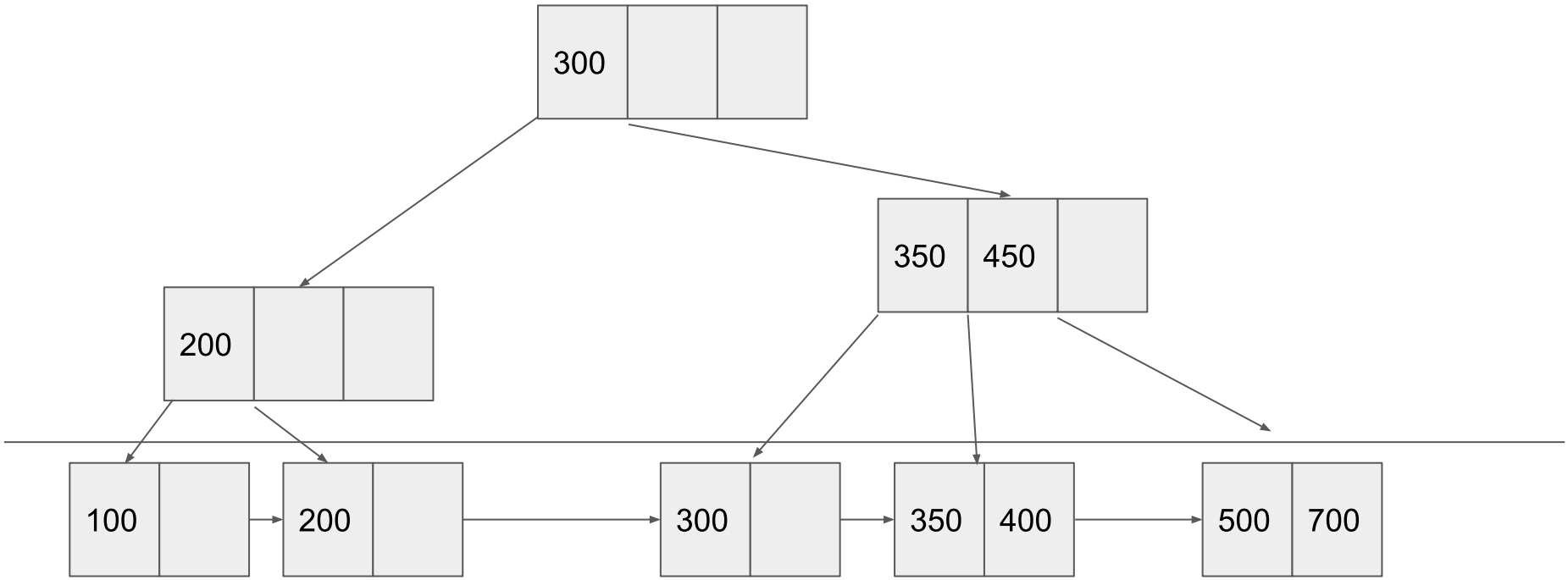
INSERT 300 : Now as we need to insert 300 but the node in which it should be inserted is not empty. Hence splitting occurs and leaf node (100,NULL) and (200,300) is formed. With root as 3/4(100+200)= 225.But 225 is greater than 200. So instead 200 is considered.



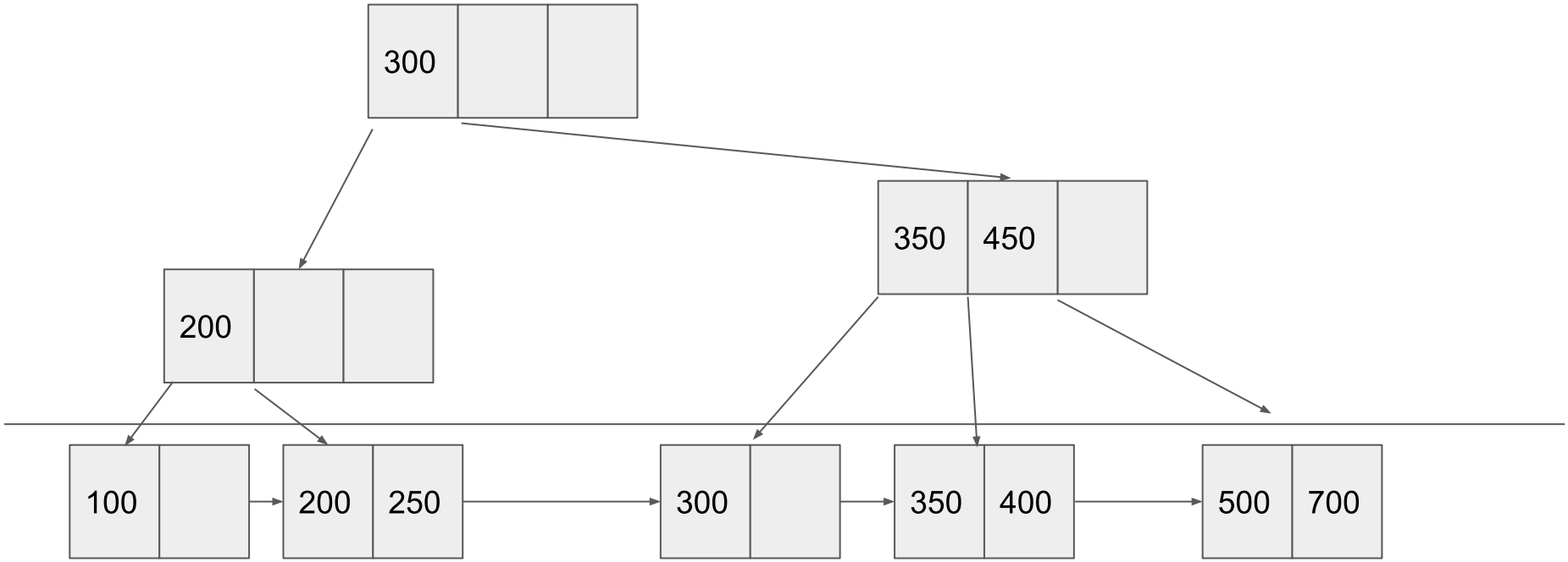
INSERT 400 : Now as we need to insert 300 but the node in which it should be inserted is not empty. Hence splitting occurs and leaf node (200,NULL) and (300,400) is formed. With root as 3/4(300+200)= 375. But 375 is greater than 300. So instead 300 is considered.



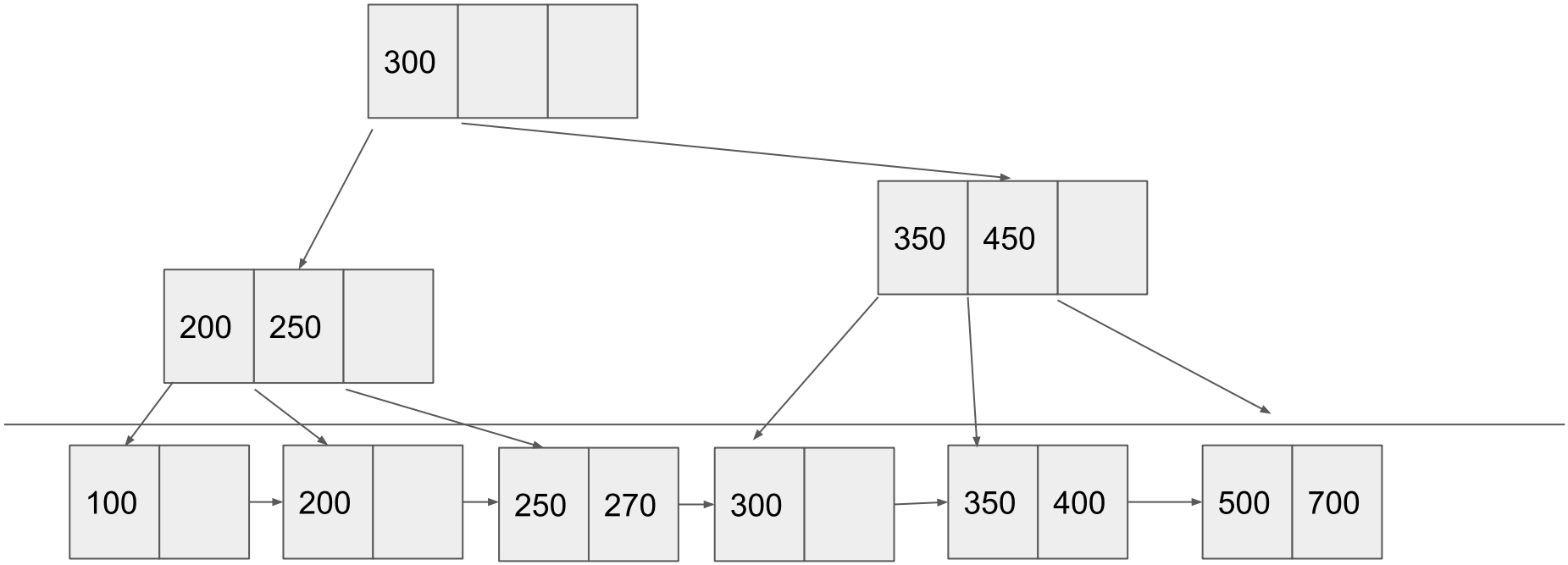
INSERT 350 : Now as we need to insert 350 but the node in which it should be inserted is not empty. Hence splitting occurs and leaf node (300,NULL) and (350,400) is formed. With root as 3/4(300+350)= 487.5 .But 487.5 is greater than 350. So instead 350 is considered.Now the parent node is also filled so a new parent node is formed and 300 is transferred to it as done in B-Tree .



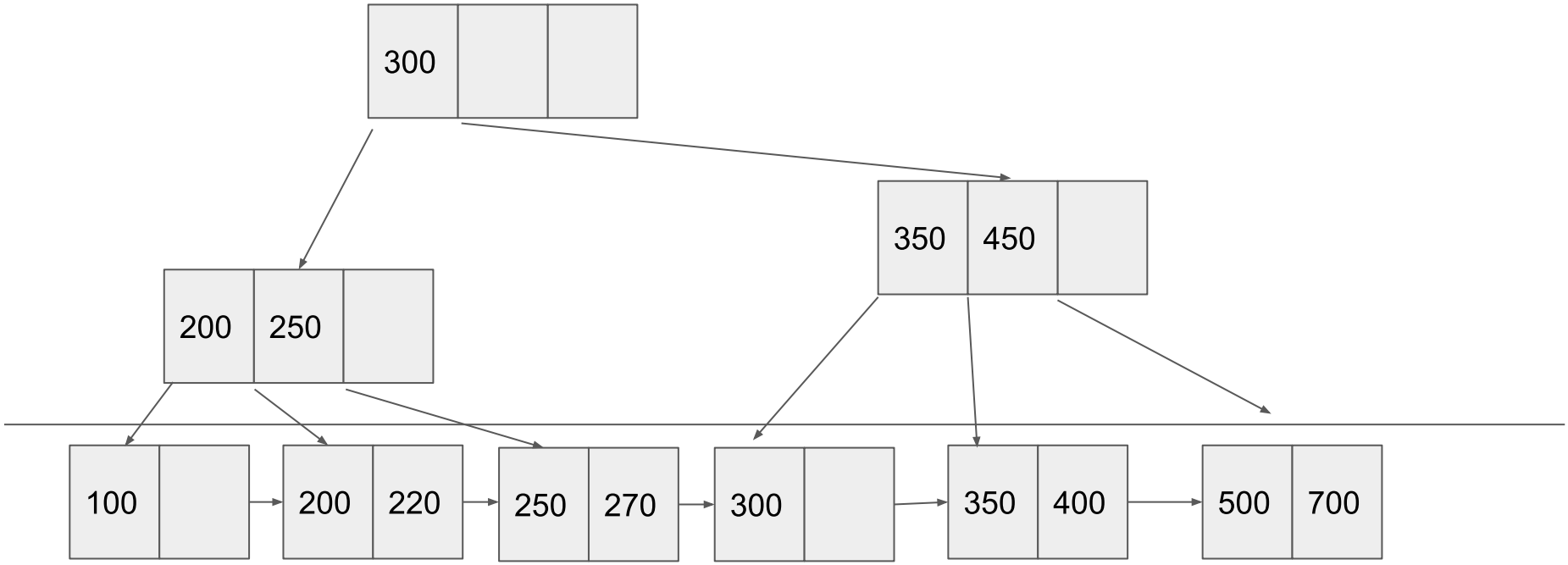
INSERT 250 : 250 must go in leaf node (200,NULL) by convention of B-Tree and there is an empty space so 250 is inserted.



INSERT 270 : Now as we need to insert 270 but the node in which it should be inserted is not empty. Hence splitting occurs and leaf node (200,NULL) and (250,2700) is formed. With root as 3/4(100+200)=337.5.But 337.5 is greater than 250. So instead 250 is considered.



INSERT 220 : 220 must go in leaf node (200,NULL) by convention of B-Tree and there is an empty space so 220 is inserted.



B+ Trees

By Shaurya Gomber (160101086)

Lecture on 14 March, 2019 by Dr. Amit Awekar

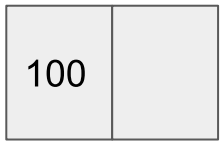
B+ Tree

A B+ tree can be viewed as a B-tree in which each node contains only keys (not key–value pairs), and to which an additional level is added at the bottom with linked leaves. B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations. In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.

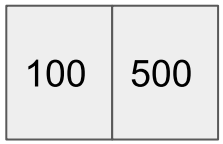
B+ Tree Insertion (Conventions followed)

1. Maximum number of keys in leaves (where data is stored) is 2 while maximum number of keys in non leaf nodes is 3.
2. Everytime while splitting, more entries are given to the left than to the right.
3. While searching for node where key is to be placed, we go left on less than as well as equal to match with root while root on greater than case.
4. Whenever a new entry is to be added to a full leaf node, the decision where to split is influenced by the above rule while the new value to be inserted in the tree above is average of the 2 numbers where we are splitting.

INSERT 100 : A leaf node is made and 100 is inserted.

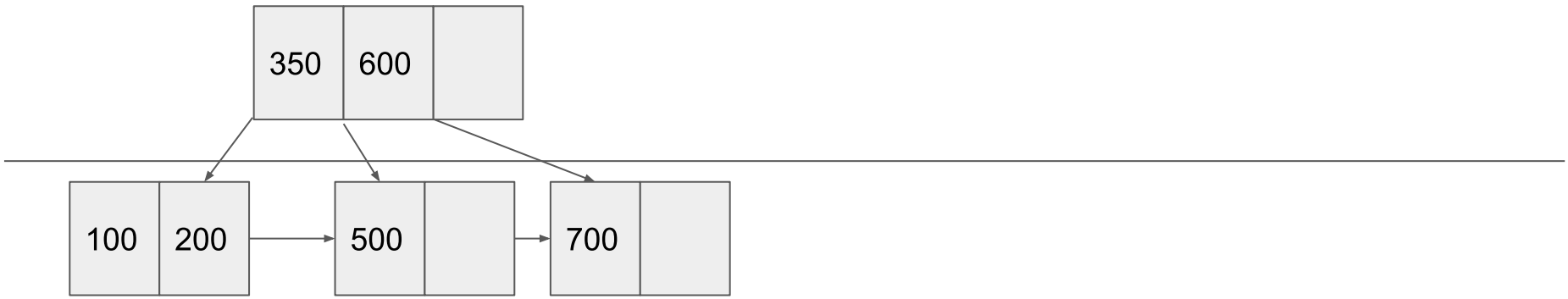


INSERT 500 : 500 is entered in vacant space besides 100 in the leaf node.

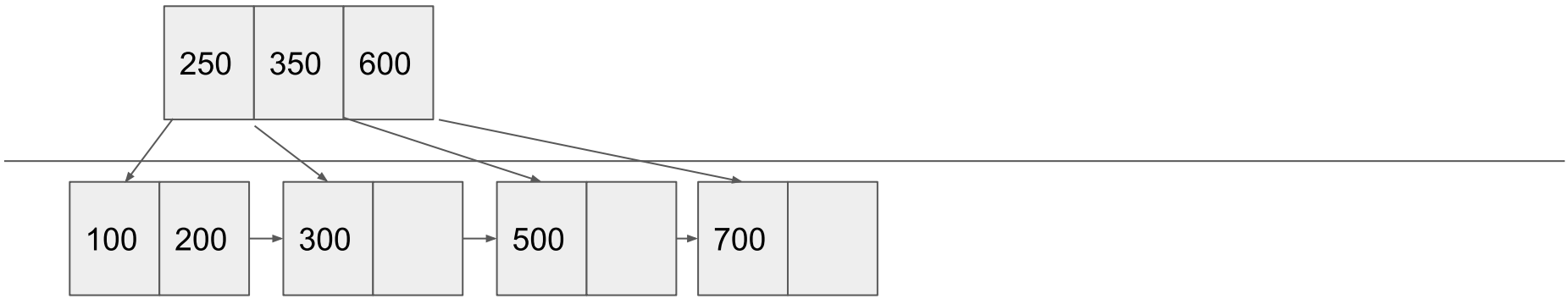


INSERT 700 : Now 700 is to be inserted but leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,500) and (700,NULL) with value to be inserted in the above tree as (500+700)/2 = 600, which is handled simply as a B-tree insertion.

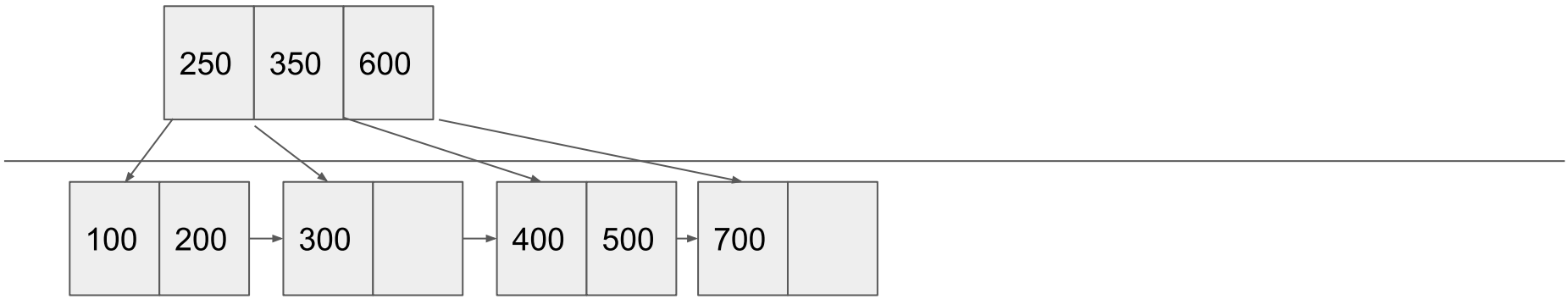
INSERT 200 : Now 200 is to be inserted but leaf node that it should go to is (100,500) as it is less than 600 but a leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,200) and (500,NULL) with value to be inserted in the above tree as (500+200)/2 = 350, which is handled simply as a B-tree insertion.



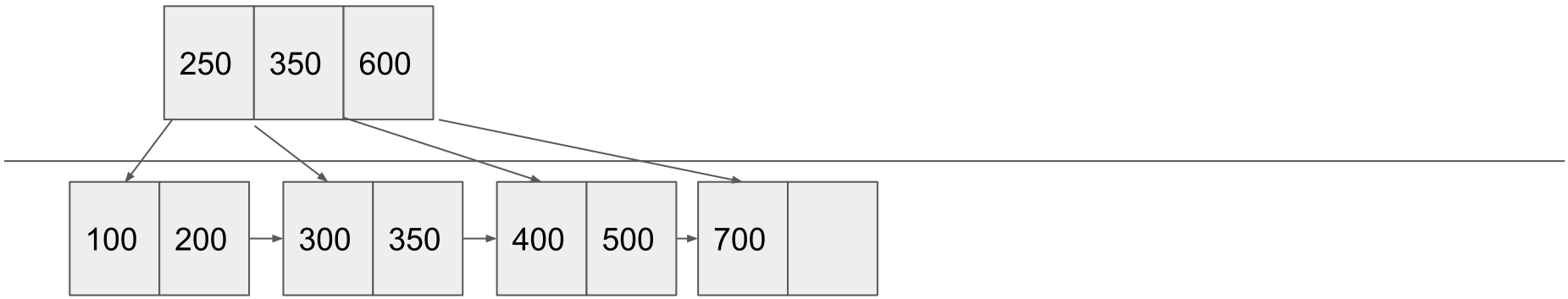
INSERT 300 : Now 300 is to be inserted but leaf node that it should go to is (100,200) as it is less than 350 but a leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,200) and (300,NULL) with value to be inserted in the above tree as (300+200)/2 = 250, which is handled simply as a B-tree insertion.



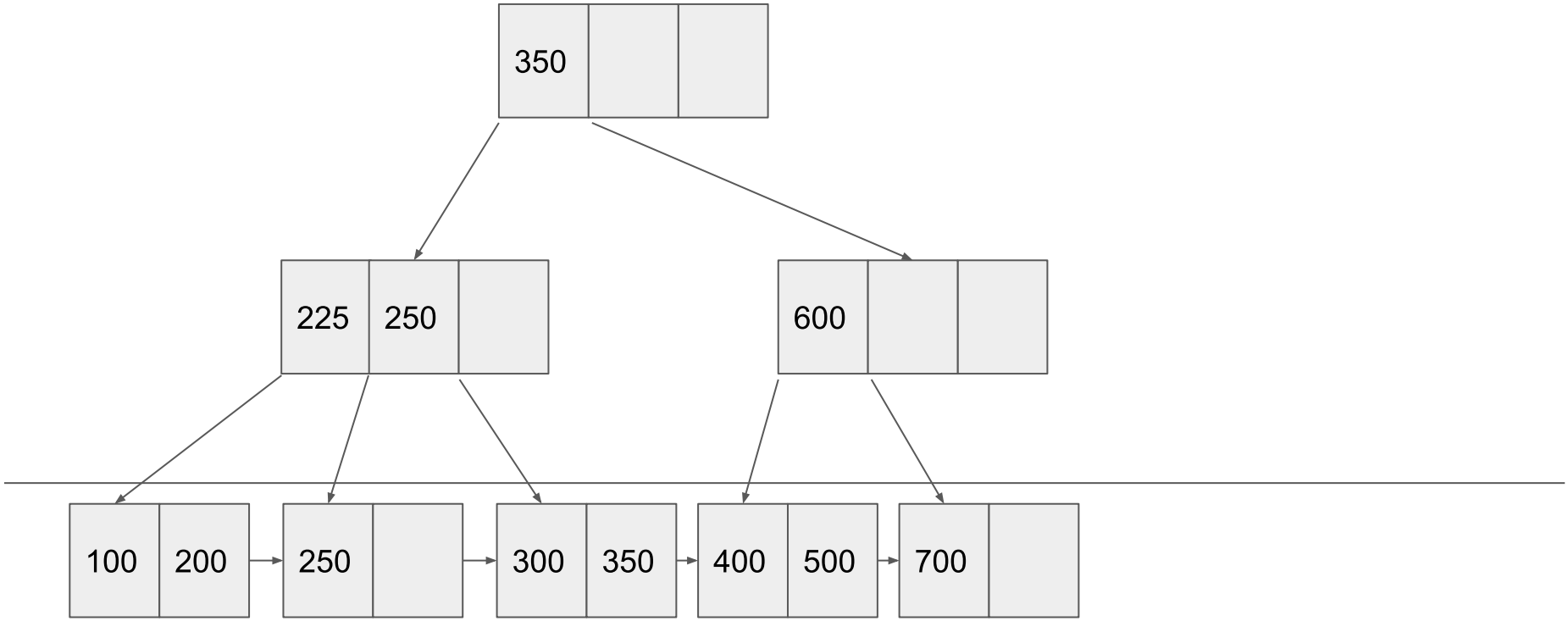
INSERT 400 : As 400 lies between 350 and 600, 400 will go to the leaf (500,NULL) and as one space is there, the leaf becomes (400,500).



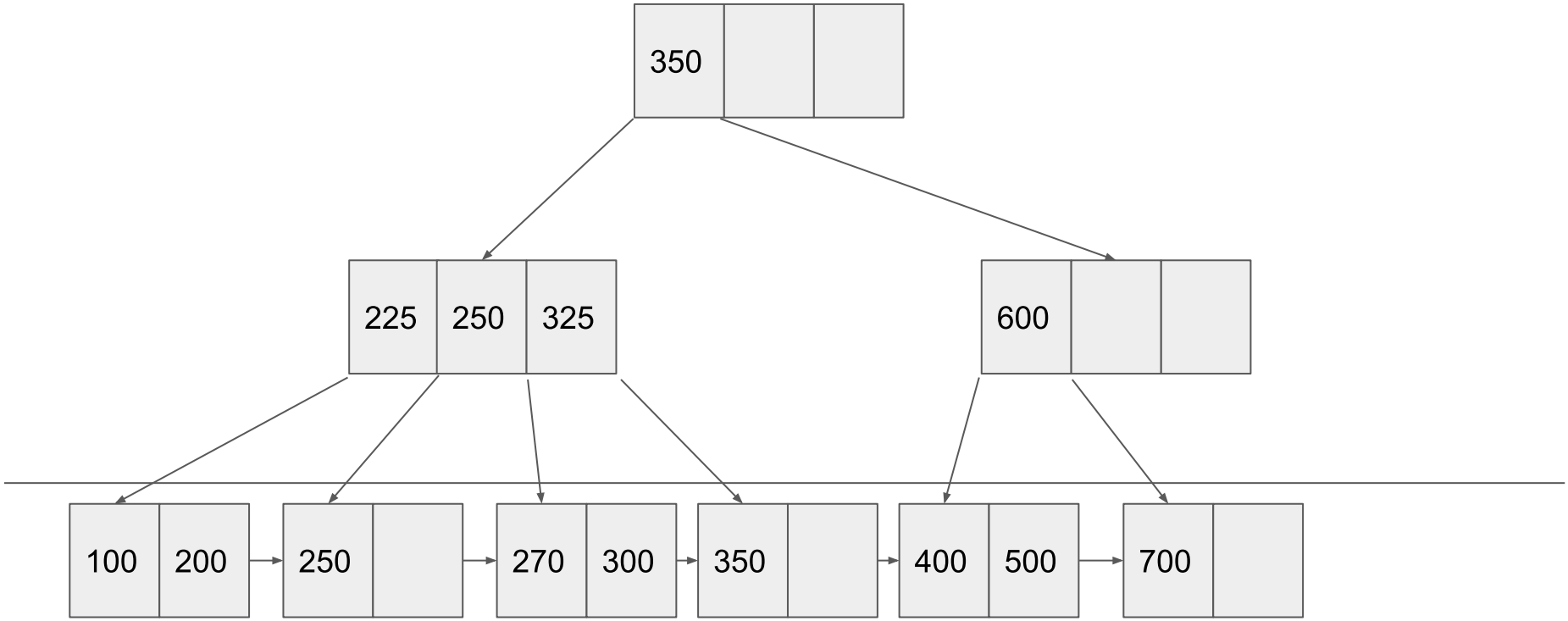
INSERT 350 : As 350 lies between 250 and 350 and our convention states while searching on less than or equal to we go left so 350 will go to the leaf (300,NULL) and as one space is there, the leaf becomes (300,350).



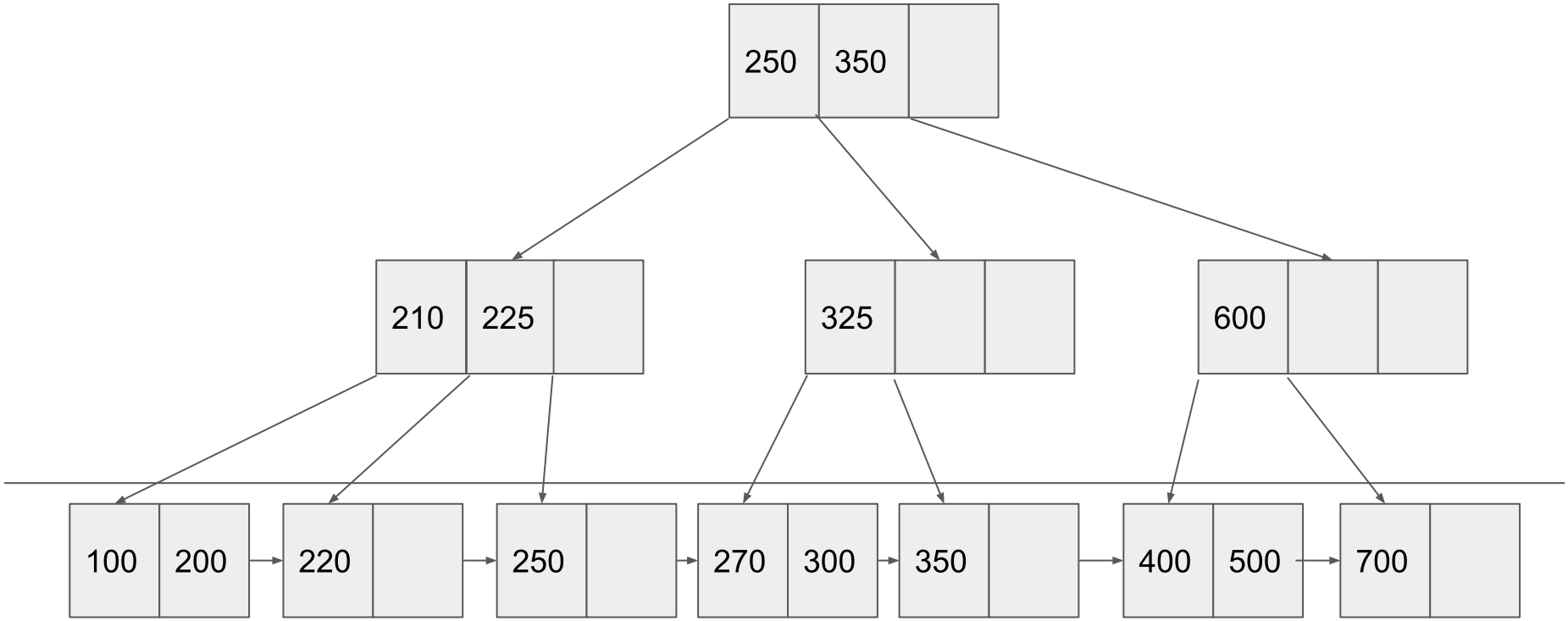
INSERT 250 : As 250 is less than equal to 250, it would be stored in (100,200) but it is full so we split giving left more as (100,200) and (250,NULL) with (200+250)/2=225 to be inserted in the B-tree above but the tree node has 3 values and so we split it again giving more data elements to left i.e (225,250,NULL) and (600,NULL,NULL) and 350 is pushed to make the new root (350,NULL,NULL).



INSERT 270 : As 270 is less than 350 and greater than 250, 270 should go to (300,350) but as it is full, we use our left convention to make (270,300) and (350,NULL) and (300+350)/2=325 is inserted in the tree above and the node (225,250,NULL) becomes (225,250,325).



INSERT 220 : As 220 is less than 350 and 225, it should go to (100,200) but as it is full, going by our left convention it is split to (100,200) and (220,NULL) and (200+220)/2=210 to be inserted to the node above i.e (225,250,325) but as it is also full, again we use our left convention to split it into (210,225,NULL) and (325,NULL,NULL) while 250 is shifted to the node above (root) and the root is updated to (250,350,NULL).



B+ Trees

By Archit Jugran (160101087)

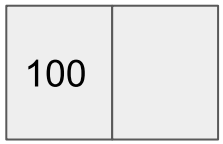
Lecture on 14 March, 2019 by Dr. Amit Awekar

B+ Tree - B+ Tree is an extension of B **Tree** which allows efficient insertion, deletion and search operations. In B **Tree**, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in **B+ tree**, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values

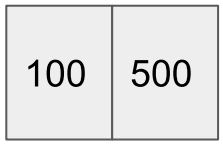
Convention followed in the examples :

1. Maximum number of keys in leaves is 2 while maximum number of keys in non leaf nodes is 3.
2. While splitting, more entries are given to the left than to the right.
3. While searching for node where key is to be placed, we go left on less than as well as equal to match with root while root on greater than case.
4. Whenever a new entry is to be added to a full leaf node, the decision where to split is influenced by the above rule while the new value to be inserted in the tree above is a random value between the two values on which split is made

INSERT 100 : A leaf node is made and 100 is inserted.

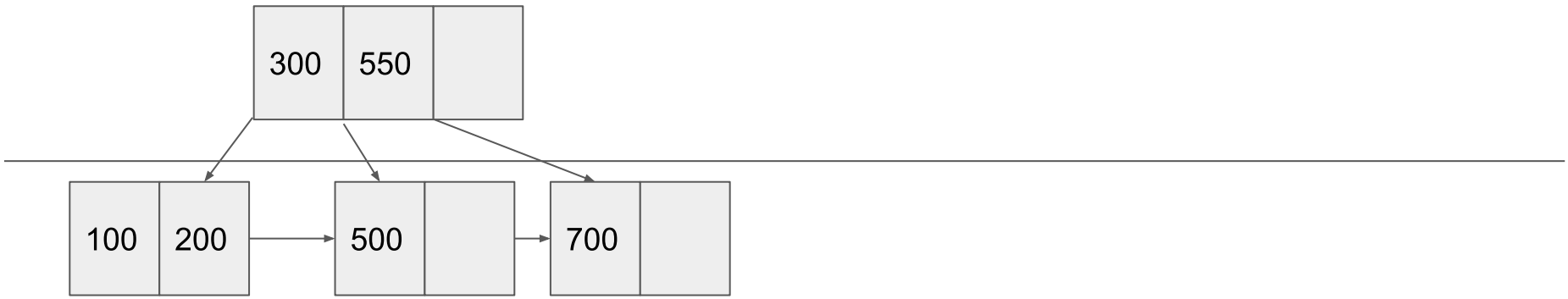


INSERT 500 : 500 is entered in vacant space besides 100 in the leaf node.

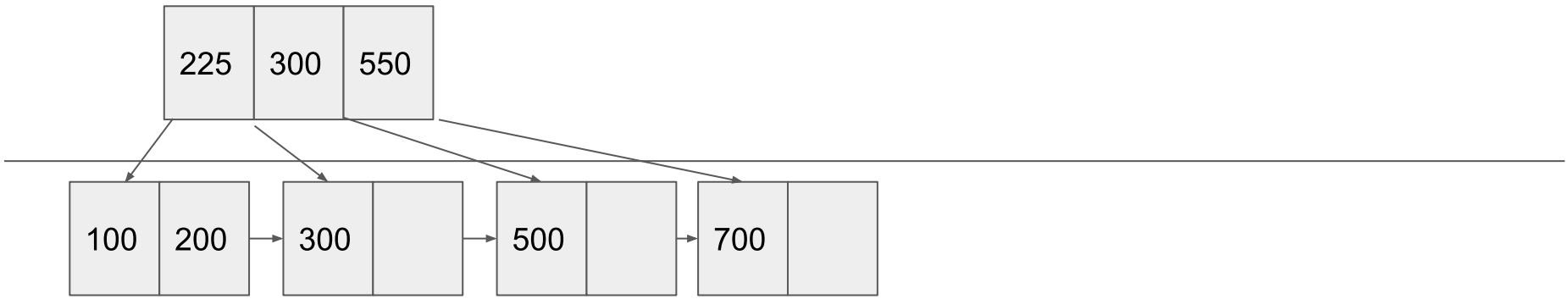


INSERT 700 : Now 700 is to be inserted but leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,500) and (700,NULL) with value inserted above (separator value) = a number between 500 and 700 = 550

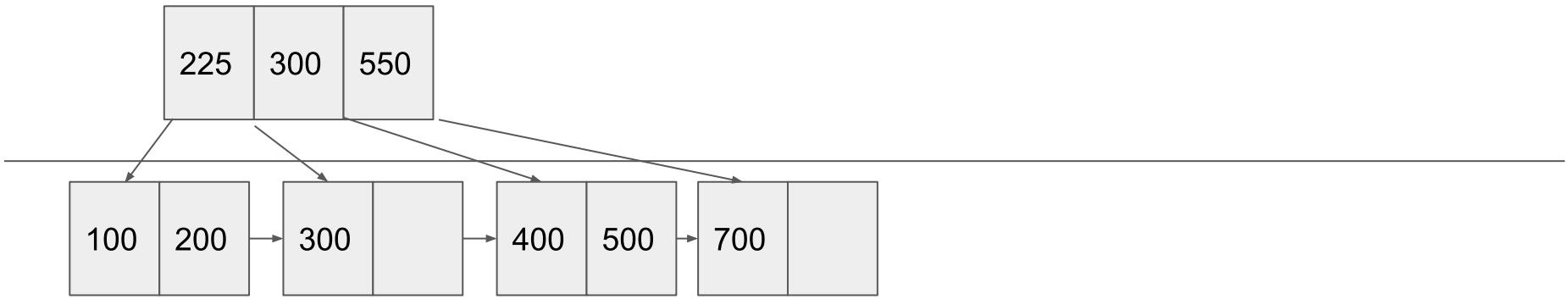
INSERT 200 : Now 200 is to be inserted but leaf node that it should go to is (100,500) as it is less than 550 but a leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,200) and (500,NULL) with value to be inserted in the above tree as 300



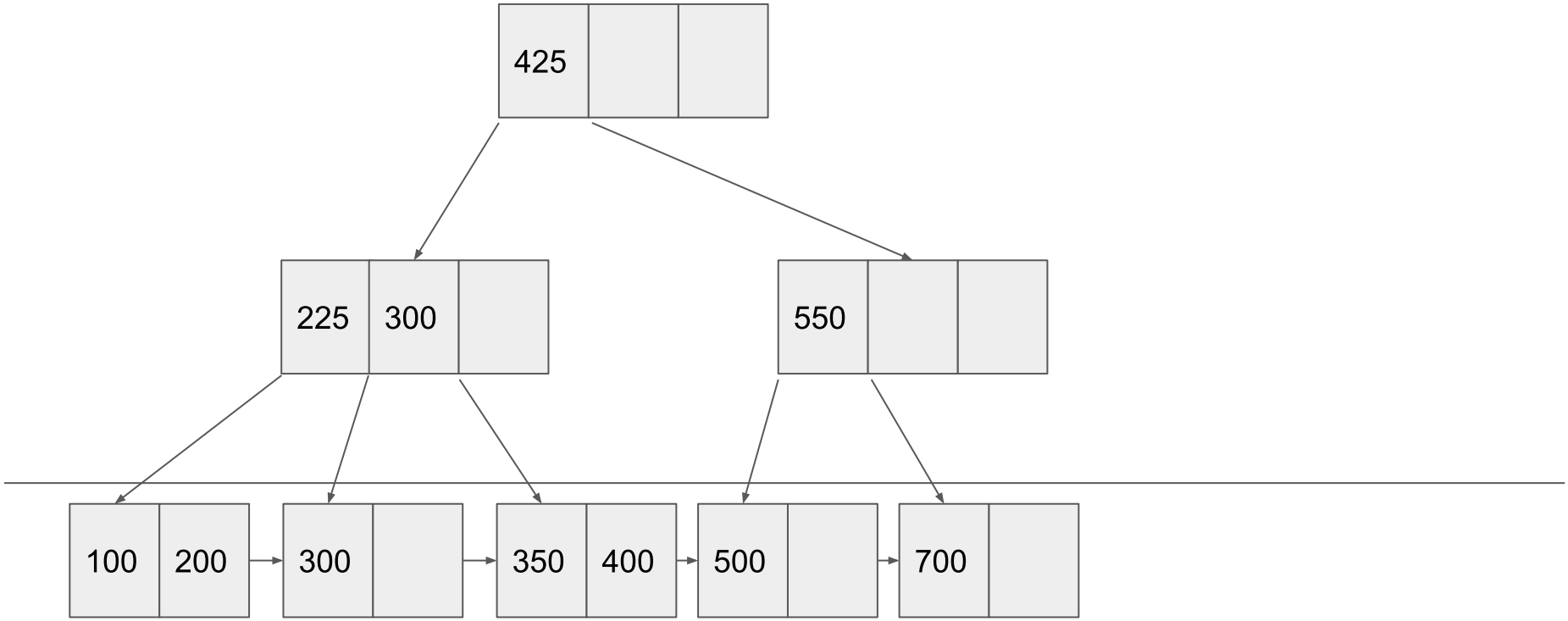
INSERT 300 : Now 300 is to be inserted but leaf node that it should go to is (100,200) as it is less than 300 but a leaf node can hold only 2 values and as our convention gives more entries to left, we split the leaf node into (100,200) and (300,NULL) with value to be inserted in the above tree as 225



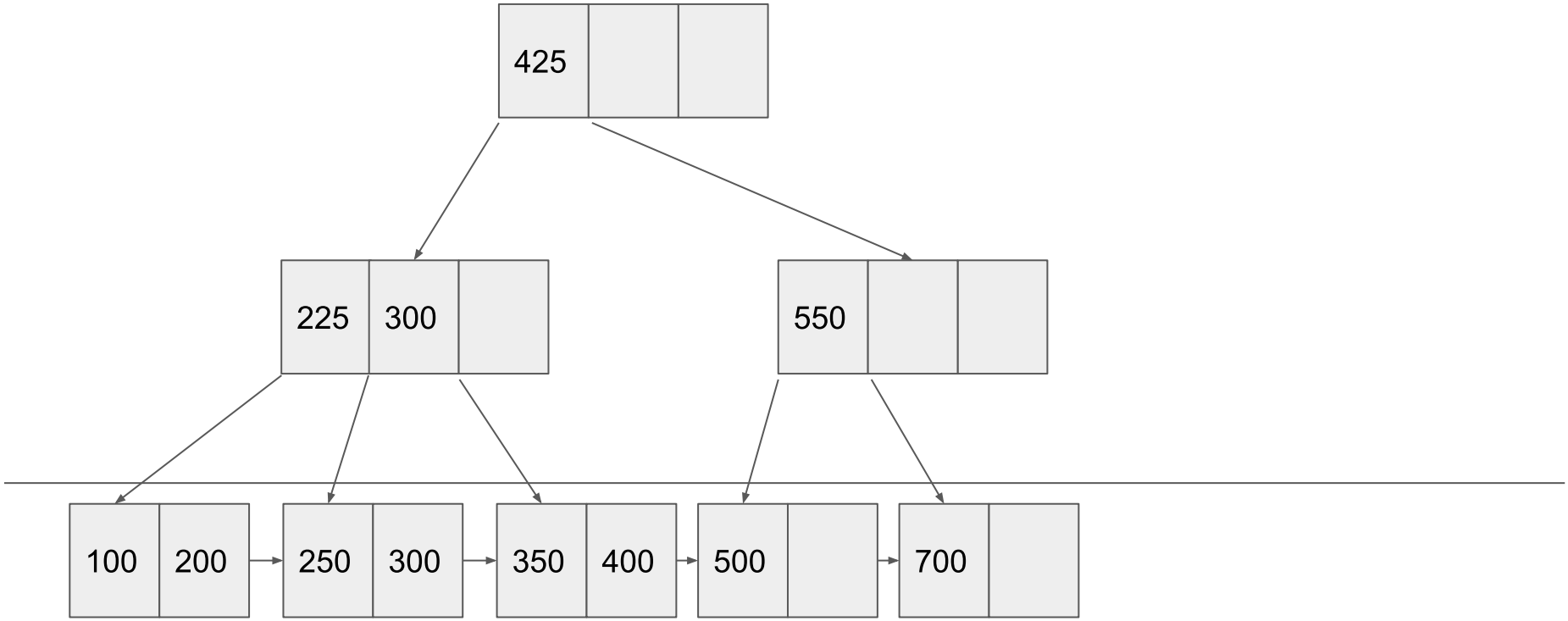
INSERT 400 : As 400 lies between 300 and 550, 400 will go to the leaf (500,NULL) and as one space is there, the leaf becomes (400,500).



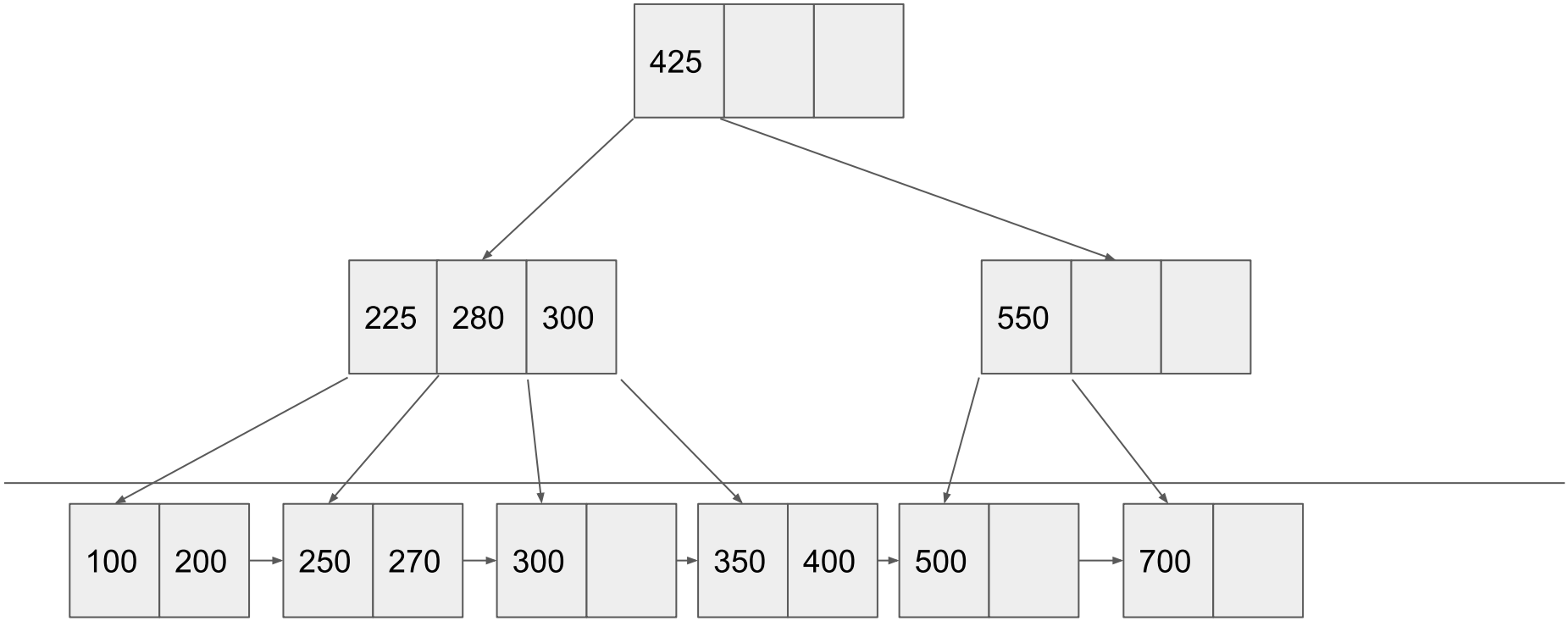
INSERT 350 : As 350 lies between 300 and 550 and leaf node (400,500) is full, so we split giving left more as (350,400) and (500,NULL) with 425 to be inserted in the B-tree above but the tree node has 3 values and so we split it again giving more data elements to left i.e (225,300,NULL) and (550,NULL,NULL) and 425 is pushed to make the new root (425,NULL,NULL).



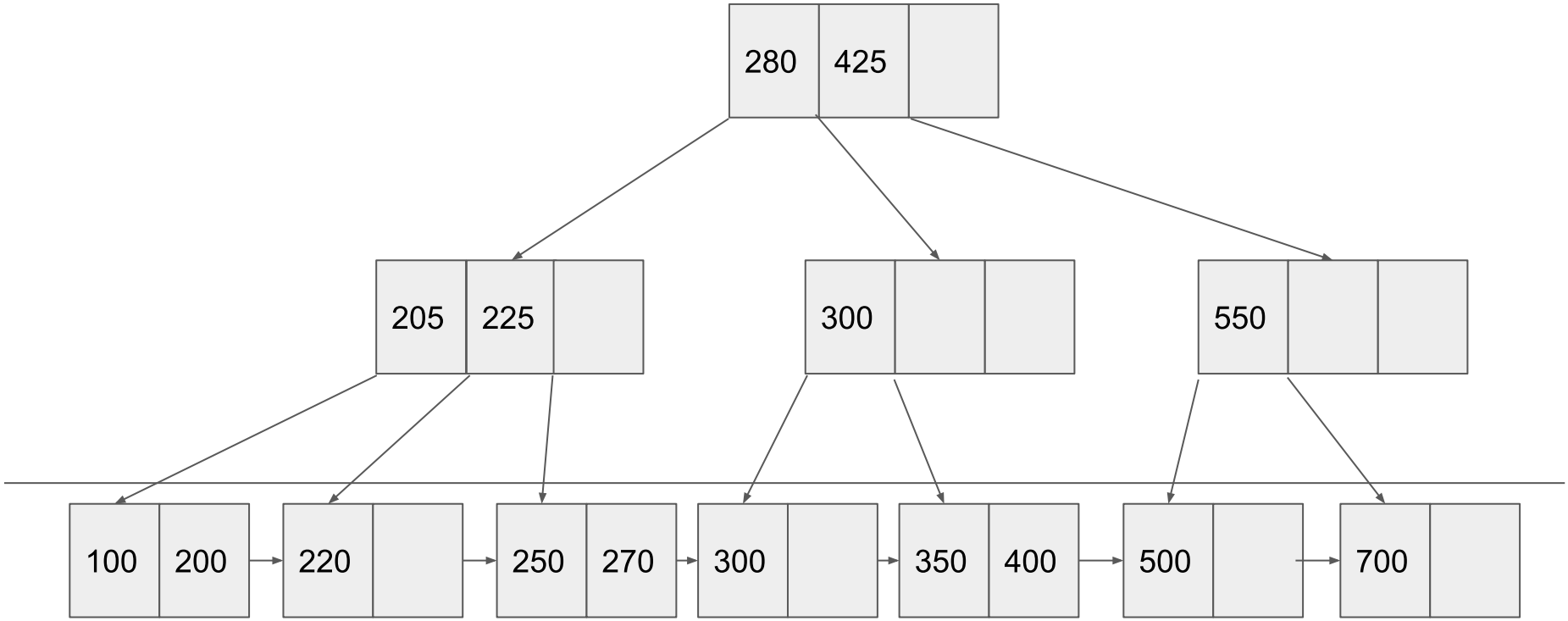
INSERT 250 : As 250 lies between 225 and 300 , we will go to the leaf (300,NULL) and as one space is there, the leaf becomes (250,300).



INSERT 270 : As 270 is less than 425, between 225 and 300, 270 should go to (250,300) but as it is full, we use our left convention to make (250,270) and (300,NULL) and 280 is inserted in the tree above and the node (225,300,NULL) becomes (225,280,300).



INSERT 220 : As 220 is less than 425 and 225, it should go to (100,200) but as it is full, going by our left convention it is split to (100,200) and (220,NULL) and 205 to be inserted to the node above i.e (225,280,300) but as it is also full, again we use our left convention to split it into (205,225,NULL) and (300,NULL,NULL) while 280 is shifted to the node above (root) and the root is updated to (280,425,NULL).



B+ Trees

By Rishabh Jain (160101088)

Lecture on 14 March, 2019 by Dr. Amit Awekar

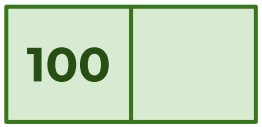
B+ TREE

B+ tree is a variation of B-tree data structure. In a B+ tree, data pointers are stored only at the leaf nodes of the tree and the path from root to each leaf is of the same length.

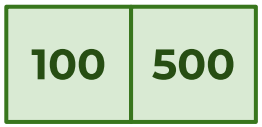
INSERTION:

- We assume that the maximum number of keys in leaves is 2 and the maximum number of keys in non leaf nodes is 3.
- During a split (in case of odd number of entries), more entries are given to the left than to the right, i.e. two to the left, and one to the right node.
- While searching for node where key is to be placed, we go left if **key <= nodeValue**, and we go right if **key > nodeValue**. {Convention}
- Whenever a new entry is to be added to a full leaf node, the decision regarding where to split is thus influenced by the above rule. The new value inserted in the tree can be any value between the two numbers.

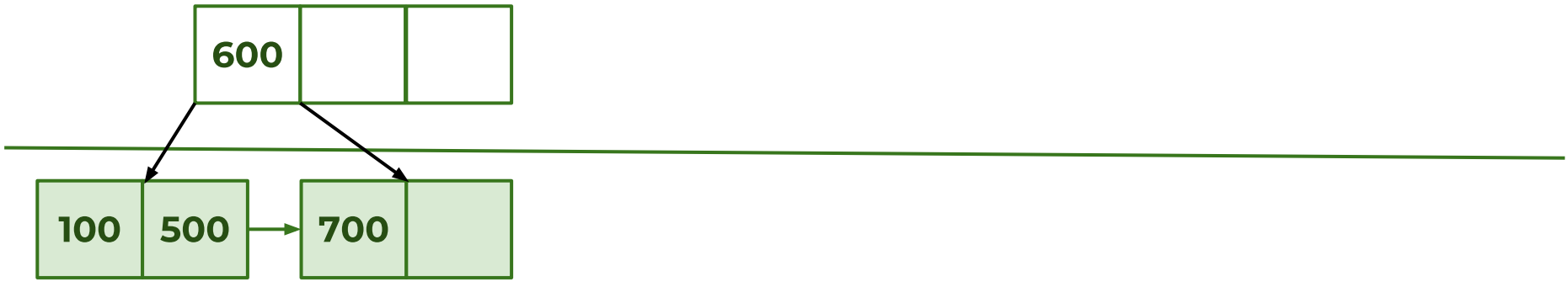
INSERT 100 : Create a new leaf node with the value 100 in one of the slots.



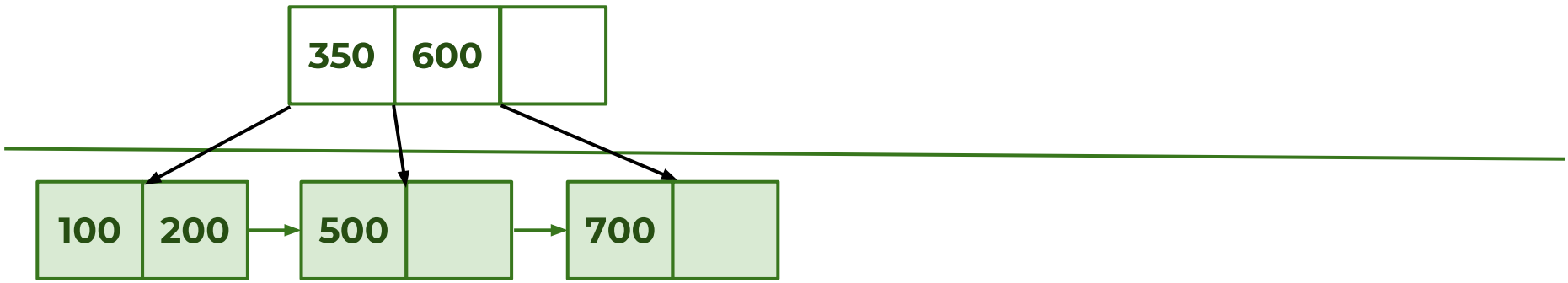
INSERT 500 : Insert value 500 into the other slot of the leaf node.



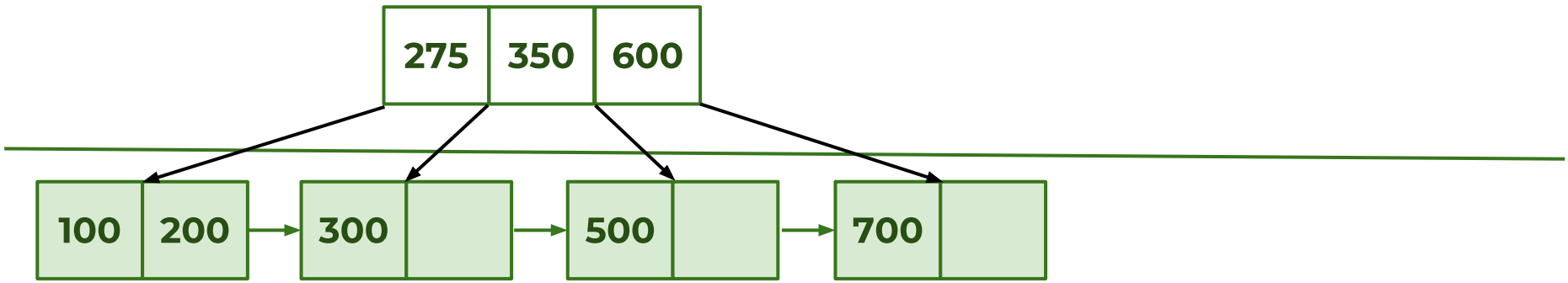
INSERT 700 : The current leaf node is full, thus we must split into **(100,500)** and **(700,NULL)** and insert the new value between 500 and 700, say **600** as a B-tree insertion.



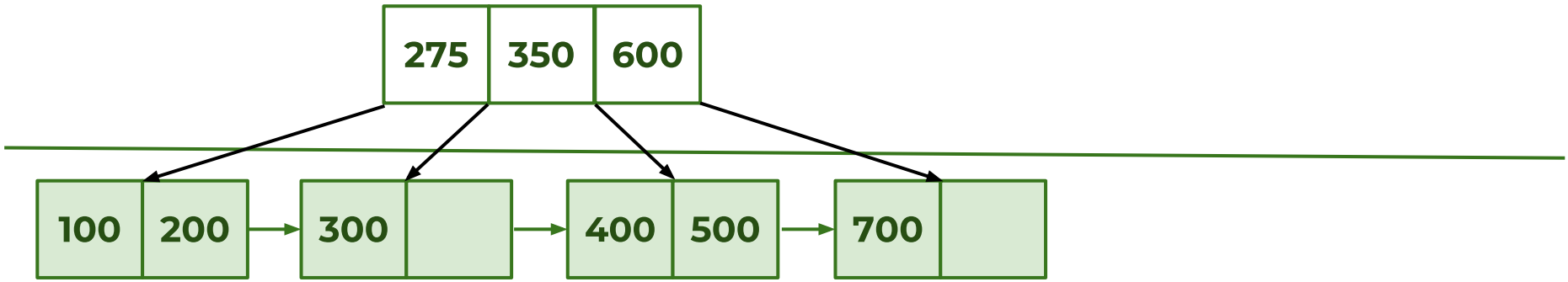
INSERT 700 : Again we see that the leaf node **[100, 200]** is full and we must split into **(100,200)** and **(500,NULL)** and insert the new value between 200 and 500, say **350** as a B-tree insertion.



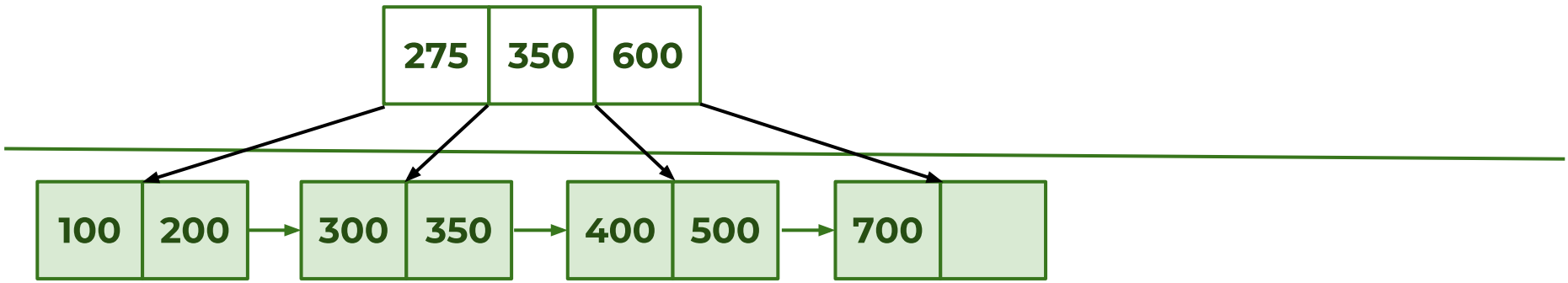
INSERT 300 : Again we see that the leaf node **[100, 200]** is full and we must split into **(100,200)** and **(300,NULL)** and insert the new value between 200 and 300, say **275** as a B-tree insertion.



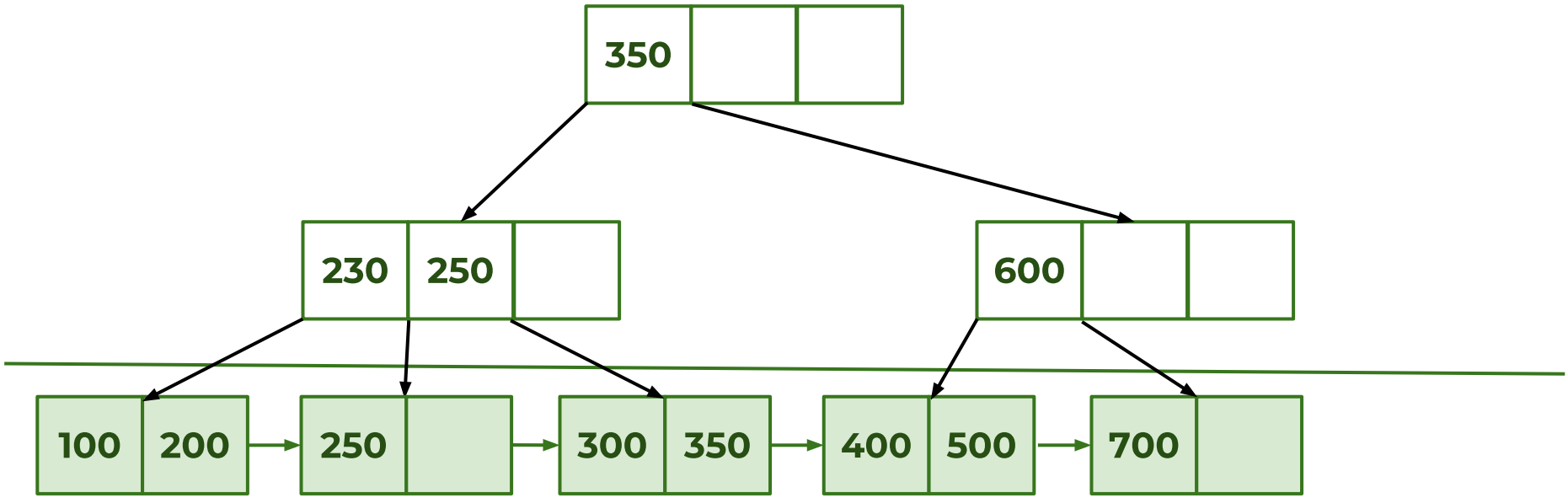
INSERT 400 : This is a simple insert. We insert the value **400** into the leaf node **(500, NULL)**.



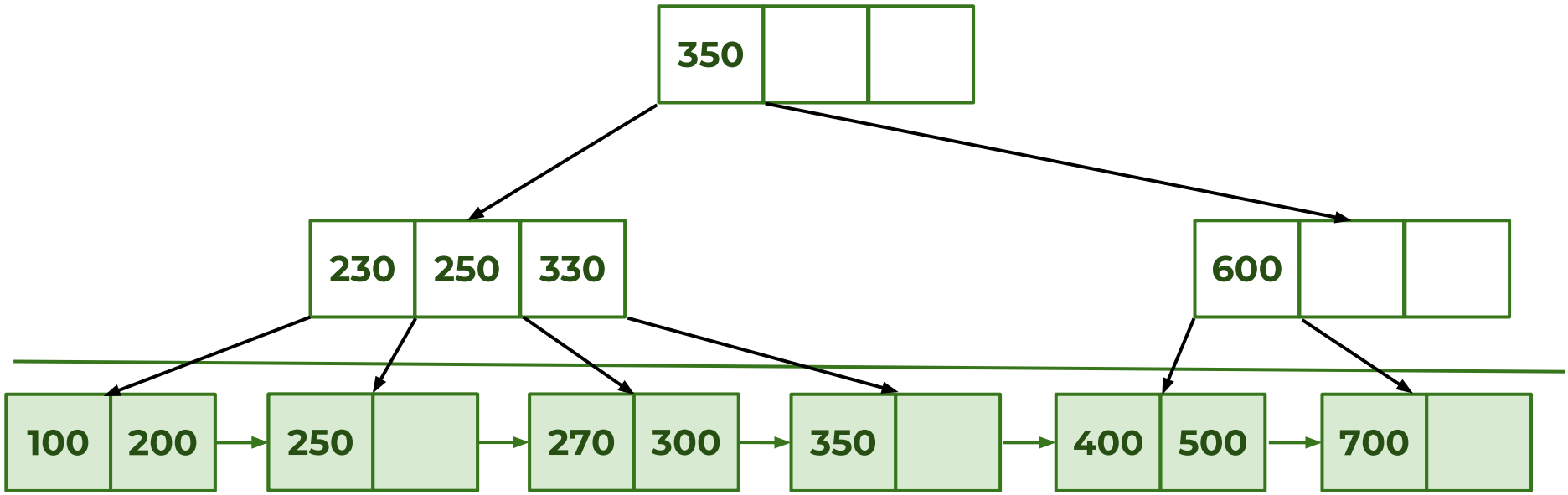
INSERT 350 : Since we go to left if the new key matches the internal node, this is also a simple insert. We insert the value **350** into the leaf node **(300, NULL)**.



INSERT 250 : We see that the value **250** would need to go to the leaf node **(100,200)** but it is full. So we split into **(100,200)** and **(250,NULL)** and a new value, say **230** is inserted into the upper level.



INSERT 270 : We see that the value **270** would need to go to the leaf node **(300,350)** but it is full. So we split into **(270,300)** and **(350,NULL)** and a new value, say **330** is inserted into the upper level.



INSERT 220 : We see that the value **220** would need to go to the leaf node **(100,200)** but it is full. So we split into **(100,200)** and **(220,NULL)** and a new value, say **210** is inserted into the upper level. But the internal node **(230,250,330)** is already full and is thus also split into **(210,230,NULL)** and **(330,NULL,NULL)** and the value **250** is pushed into the root.