Indexing

Presented by

Dr. Md. Abir Hossain

Dept. of ICT

MBSTU

Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- There are two basic kinds of indices:
  - Ordered indices: Based on a sorted ordering of the values.
  - Hash indices. Based on a uniform distribution of values across a range of buckets..
- **Search Key** - attribute to set of attributes used to look up records in a file. Ex. primary key, candidate key, and superkey
- An **index file** consists of records (called **index entries**) of the form

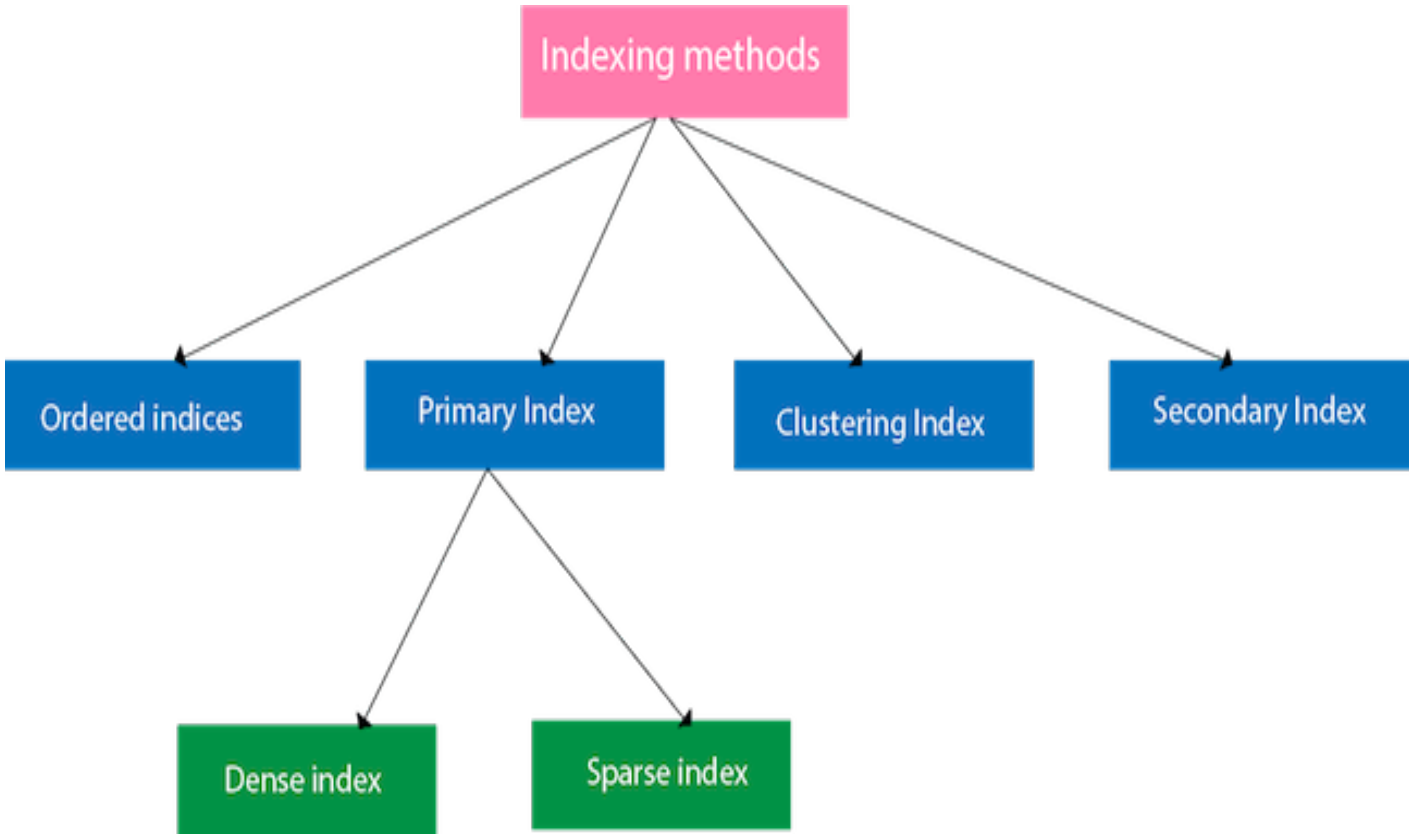
search-key	pointer

- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Index Evaluation Metrics

- **Access type** – The types of access that are supported efficiently. Access types can include finding records with a specified attribute value
- **Access time -** The time it takes to find a particular data item, or set of items.
- **Insertion time –** The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data item, as well as the time it takes to update the index structure.
- **Deletion time -** The time it takes to delete a data item. This value includes the time it takes to find the item to be deleted, as well as the time it takes to update the index structure.
- **Space overhead -** The additional space occupied by an index structure.

Indexing Methods

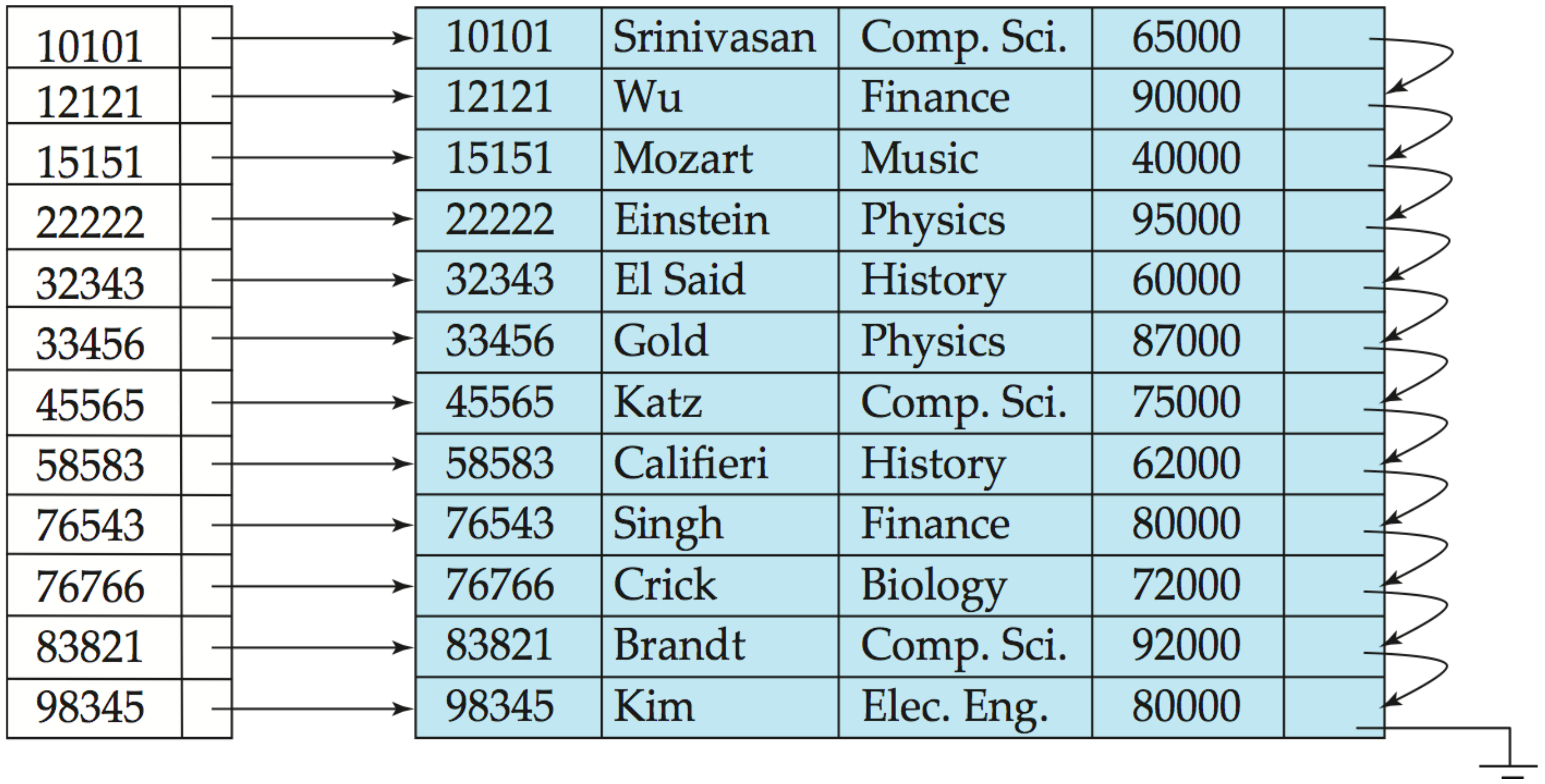


Indexing Methods(Cont.)

- In an **ordered index, t**he indices are usually sorted to make searching faster.
- **Primary index:**
- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.
- Index-sequential file**:** ordered sequential file with a primary index.

Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation
- For the first data record, the index record contains the search-key value and a pointer.
- The rest of the records with the same search-key value would be stored sequentially.

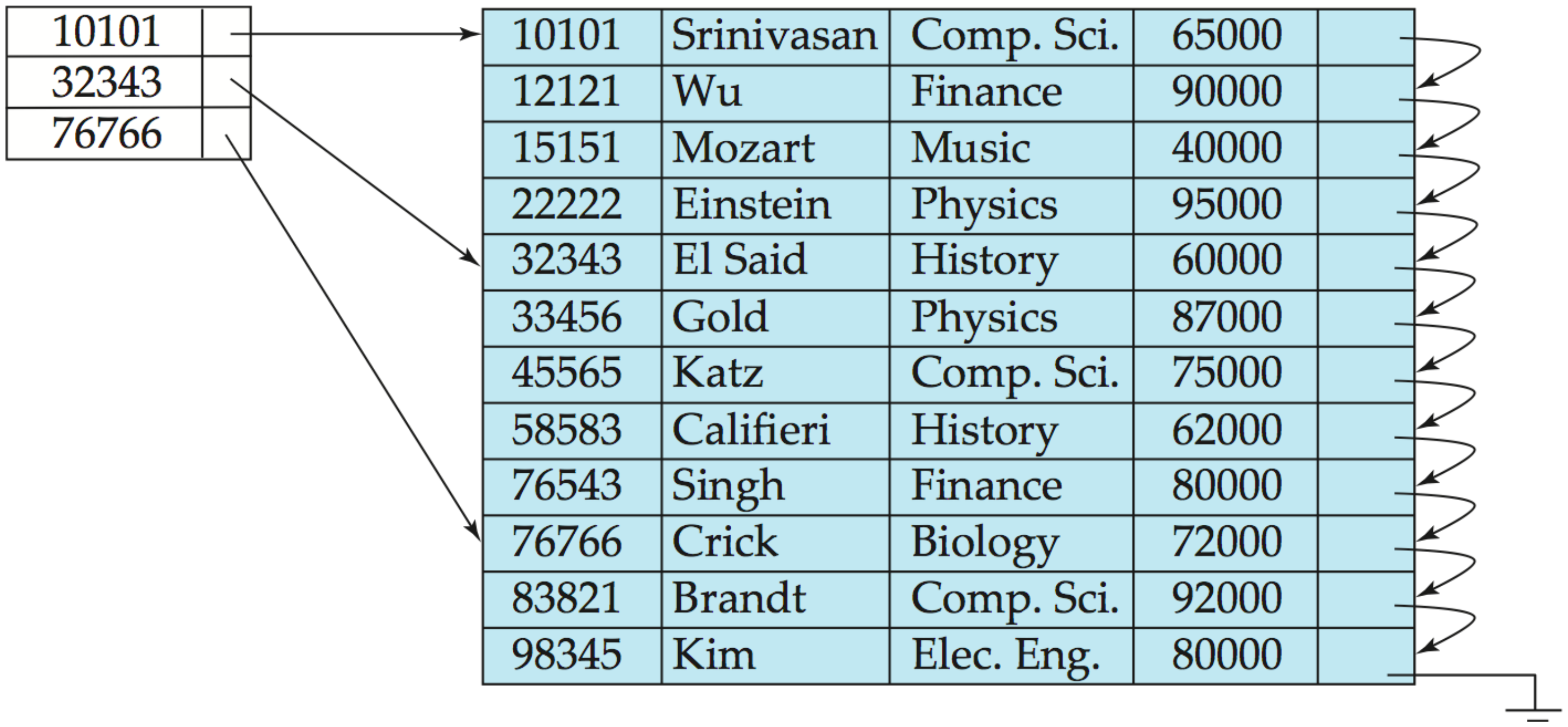


10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sparse Index Files

- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key



10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

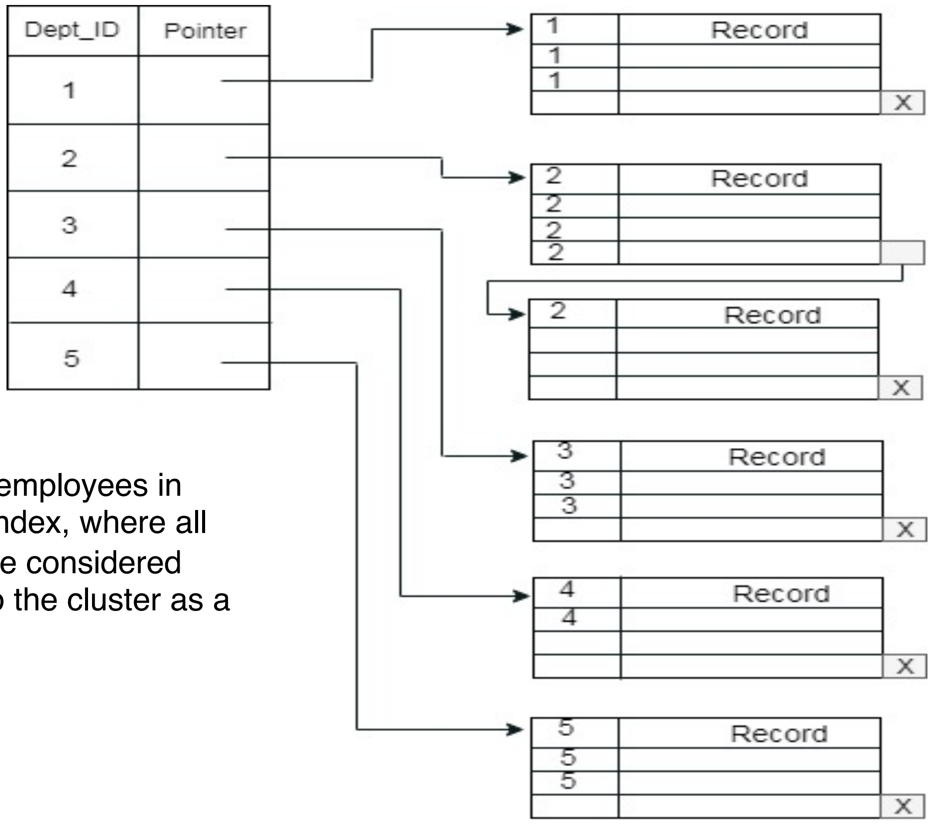
Sparse Index Files (Cont.)

- Compared to dense indices:
  - Less space and less maintenance overhead for insertions and deletions.
  - Generally slower than dense index for locating records.
- **Good tradeoff**: sparse index with an index entry for every block in file, corresponding to least search-key value in the block.

Indexing Methods(Cont.)

- **Clustering Index**
- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Clustering Index

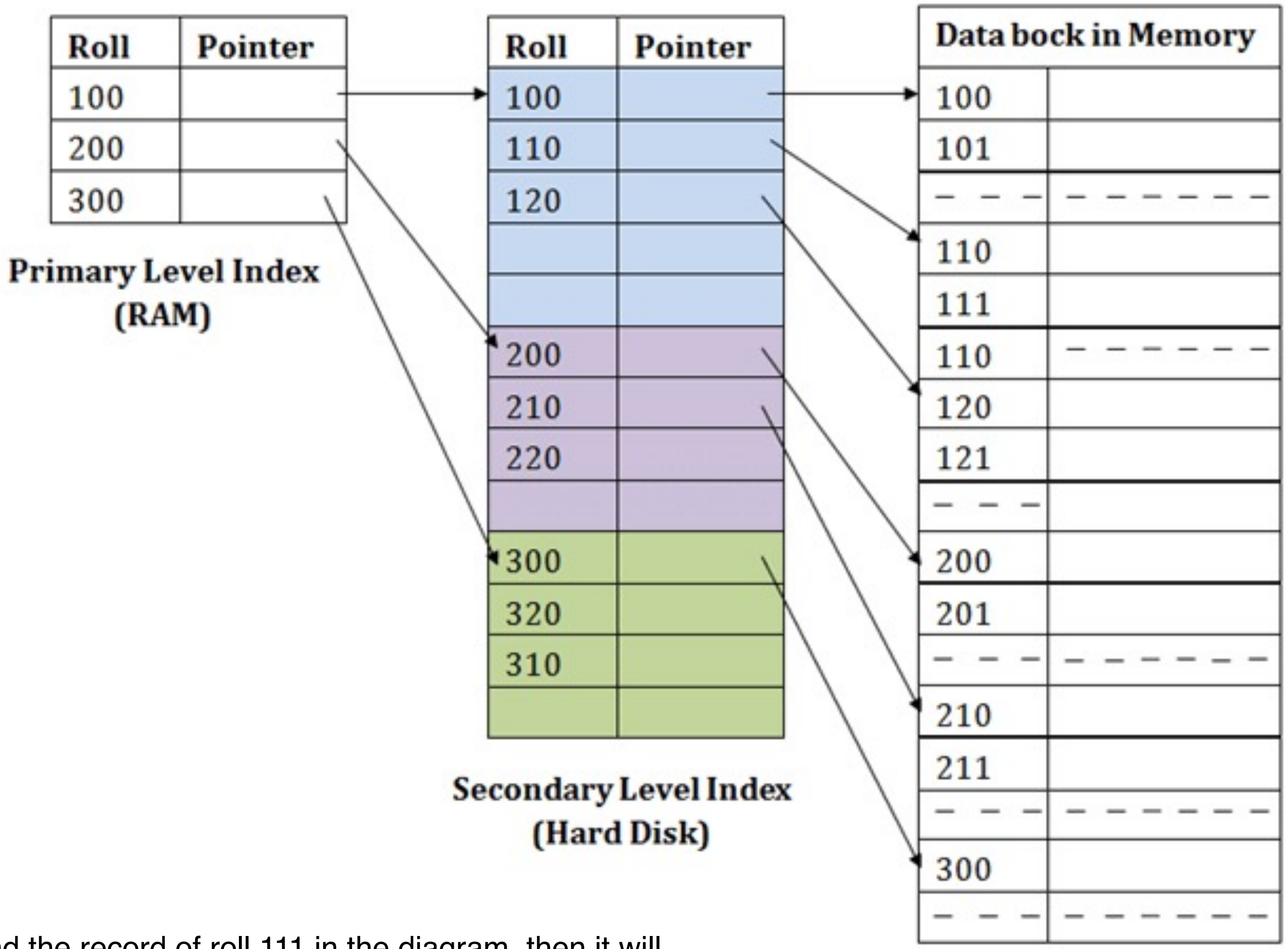


Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.

Indexing Methods(Cont.)

- **Secondary Index**
- In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
- Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).

Secondary Index



For example:
If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

B^+-Tree Index Files

B^+-tree indices are an alternative to indexed-sequential files.

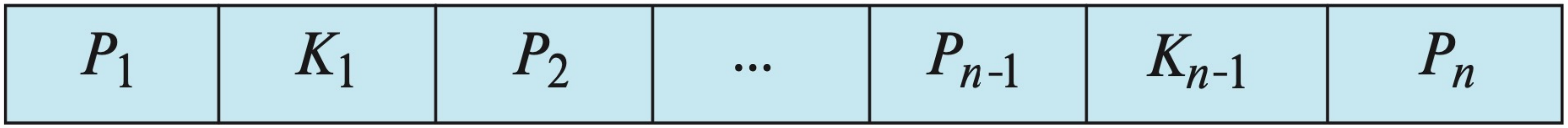
- Disadvantage of indexed-sequential files
  - performance degrades as file grows, since many overflow blocks get created.
  - Periodic reorganization of entire file is required.
- Advantage of B^+-tree index files:
  - automatically reorganizes itself with small, local, changes in the face of insertions and deletions.
  - Reorganization of entire file is not required to maintain performance.
  - B+-tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length.
- (Minor) disadvantage of B^+-trees:
  - extra insertion and deletion overhead, space overhead.
  - Since nodes may be as much as half empty, there is some wasted space.

B^+-Tree Index Files (Cont.)

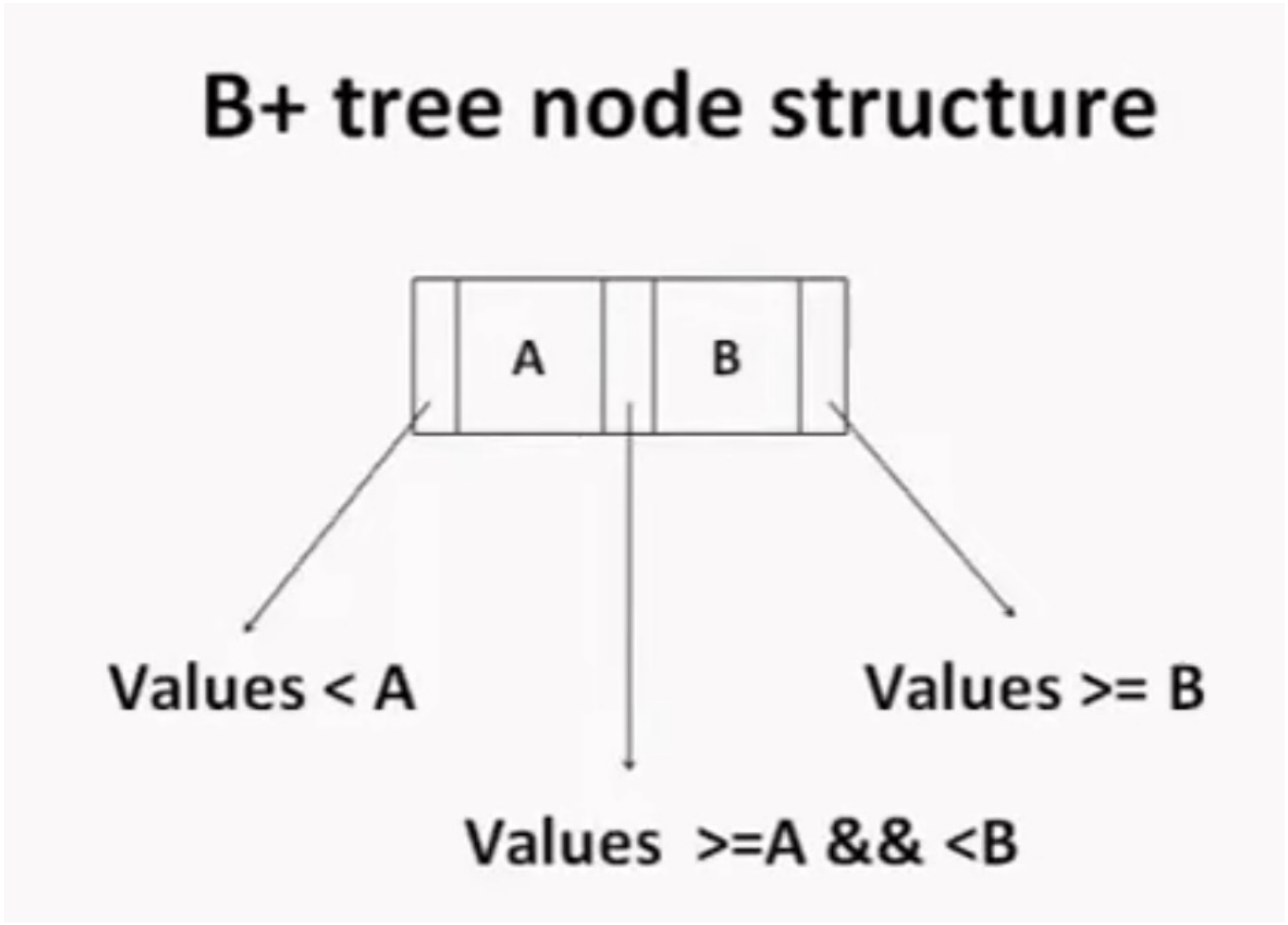
A B+-tree index is a multilevel index, but it has a structure that differs from that of the multilevel index-sequential file. A B^+-tree is a rooted tree satisfying the following properties:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
- A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values.

- A typical B+-tree contains up to n − 1 search-key values $K_1, K_2, \ldots, K_{n-1}$, and n pointers $P_1, P_2, \ldots, P_n$. The search-key values within a node are kept in sorted order; thus, if $i < j$, then $K_i < K_j$.



B$^+$-Tree Index Files (Cont.)



https://www.youtube.com/watch?v=DqcZLulVJ0M

Insertion in B+ Tree(Contd..)

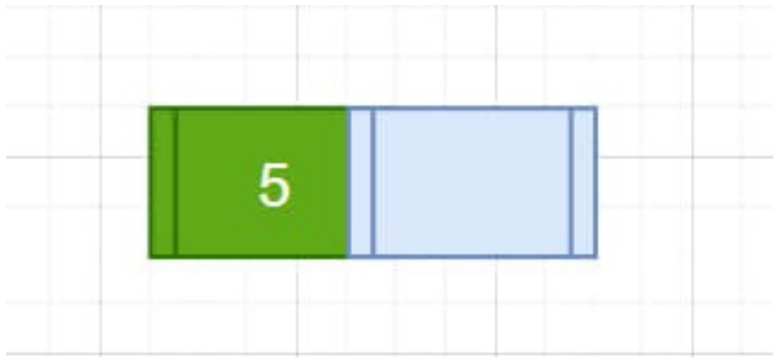
- Form a B+ tree of order m=3 by inserting the following keys sequentially:

➢ 5, 15, 25, 35, 45

Insert 5

- 5, 15, 25, 35, 45

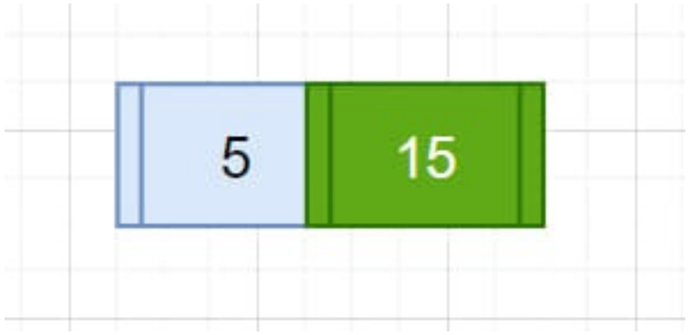
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert 15

- 5, 15, 25, 35, 45

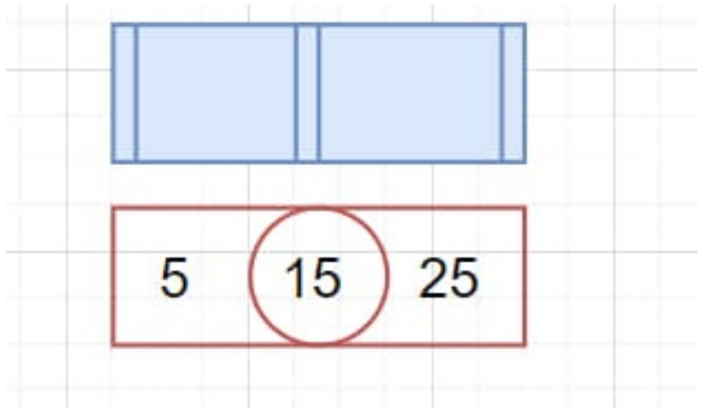
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.

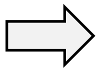


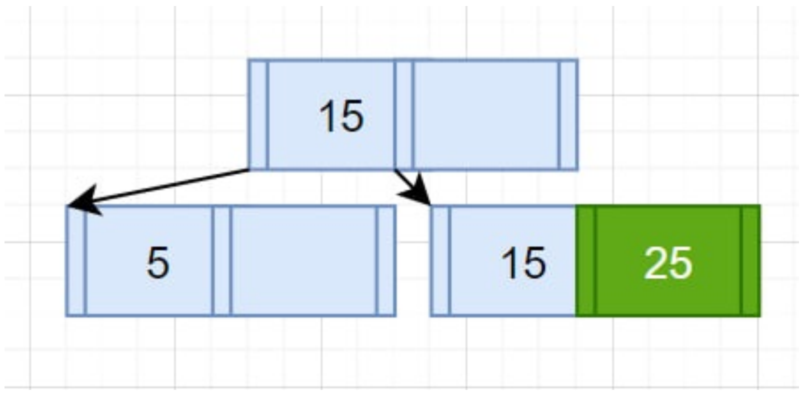
Insert 25

- 5, 15, 25, 35, 45

- We look at the leaf node it would go in and find there is no room.
- We split the node, and roll the middle value to the index node above it.



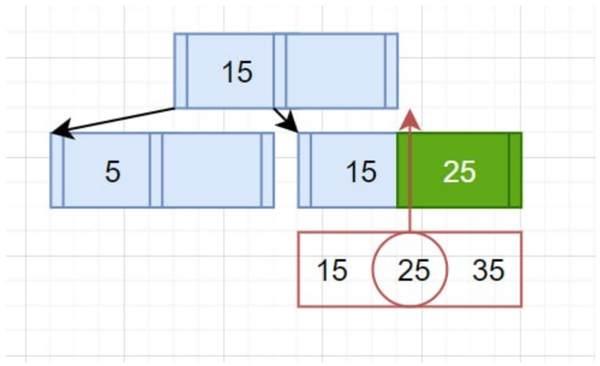


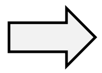


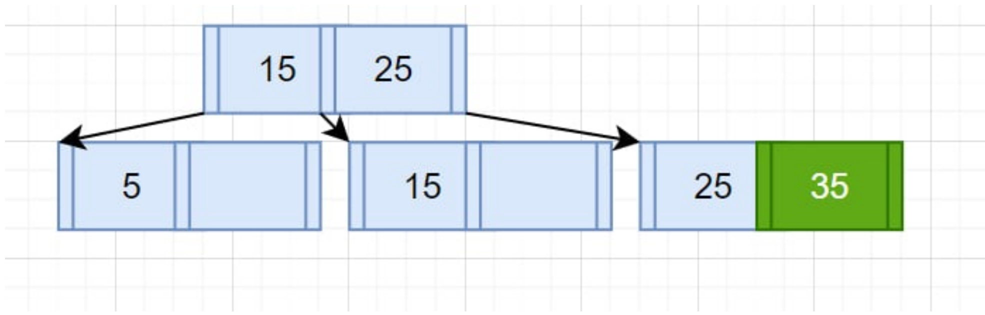
Insert 35

- 5, 15, 25, 35, 45

- We look at the leaf node it would go in and find there is no room.
- We split the node, and roll the middle value to the index node above it.



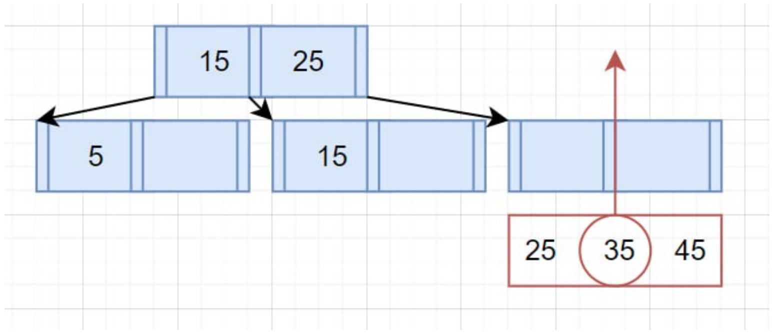


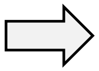


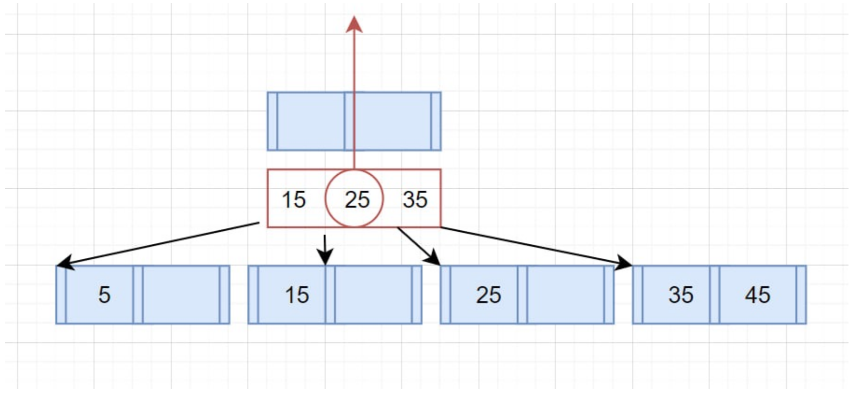
Insert 45

- ➢ 5, 15, 25, 35, 45

- We look at the leaf node it would go in and find there is no room.
- We split the node, and roll the middle value 35 to the index node above it.
- 35 go in above node and find there is no room. We split the node, and roll the middle value 25 to the index node above it.



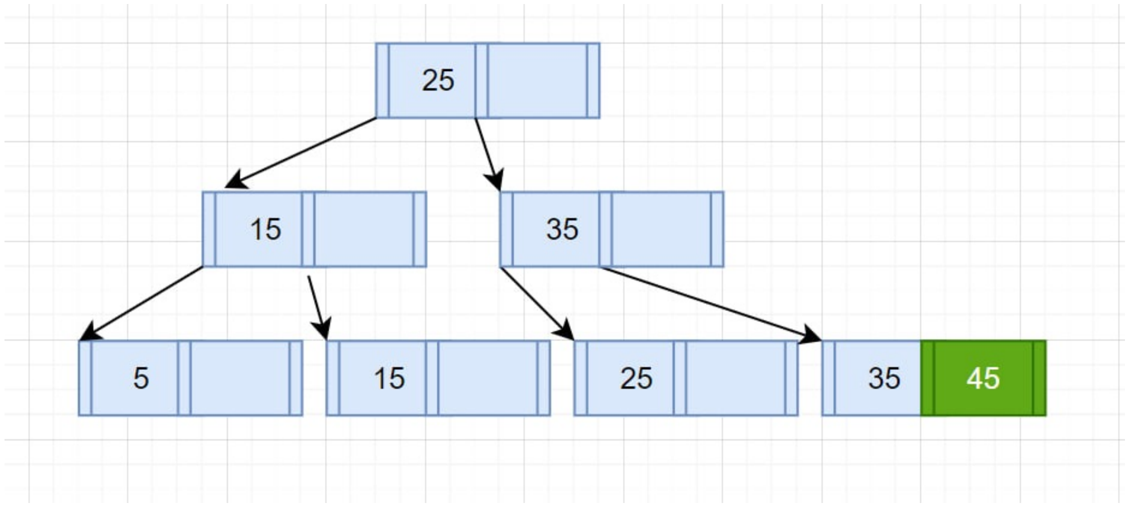




Insertion in B+ Tree

- 5, 15, 25, 35, 45

- After inserting all the given values we get the B+ tree as following:

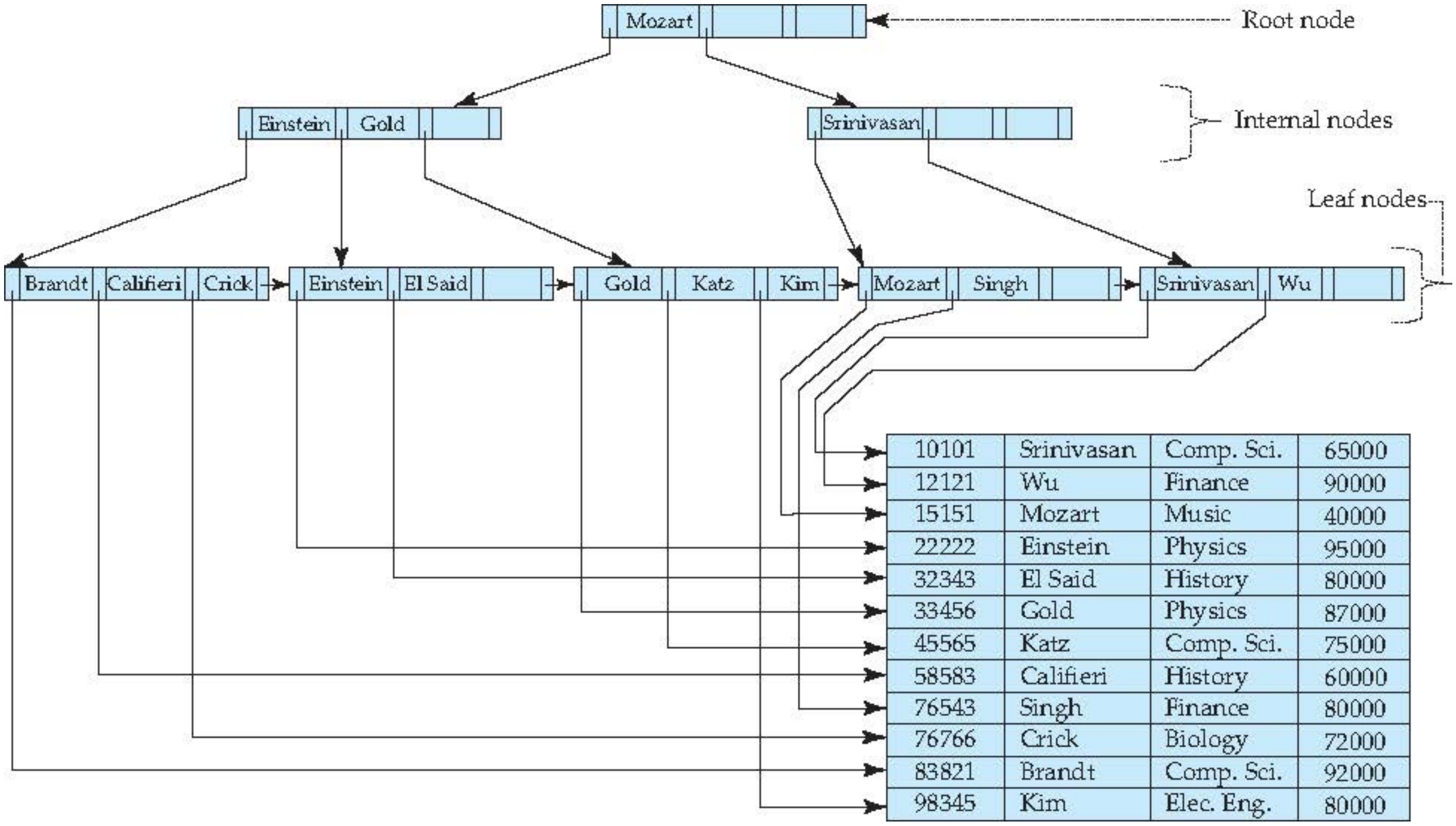


Insertion in B+ Tree

- Form a B+ tree of order m=4 by inserting the following keys sequentially:

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

Example of B$^+$-Tree (pointer n=4)

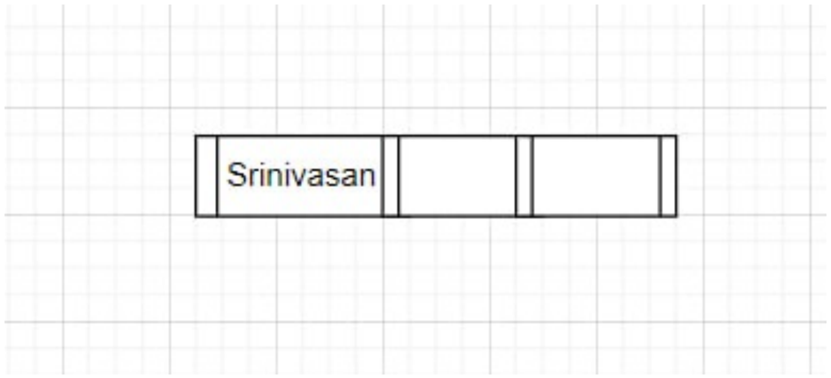


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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

Insert ‘Srinivasan’

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

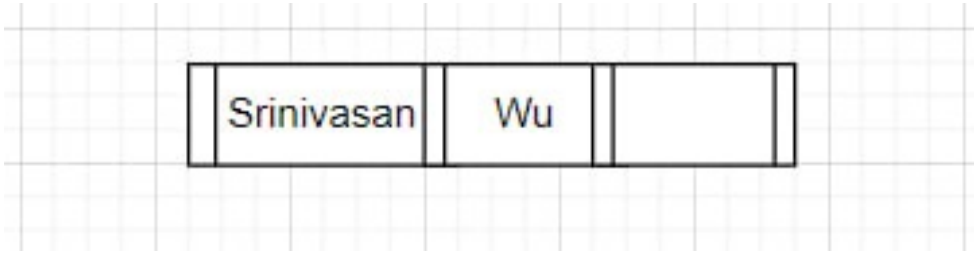
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘Wu’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

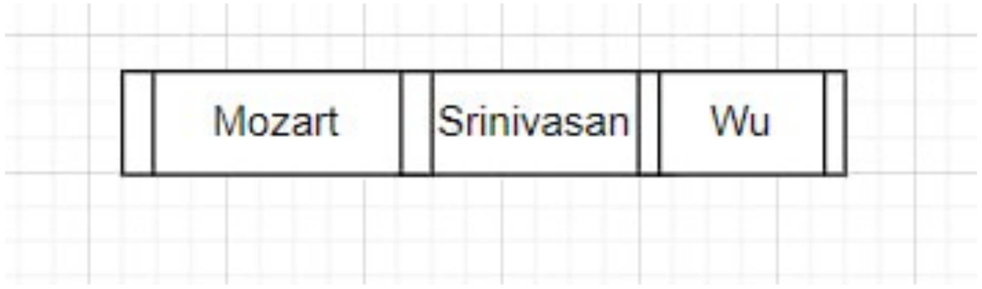
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert 'Mozart'

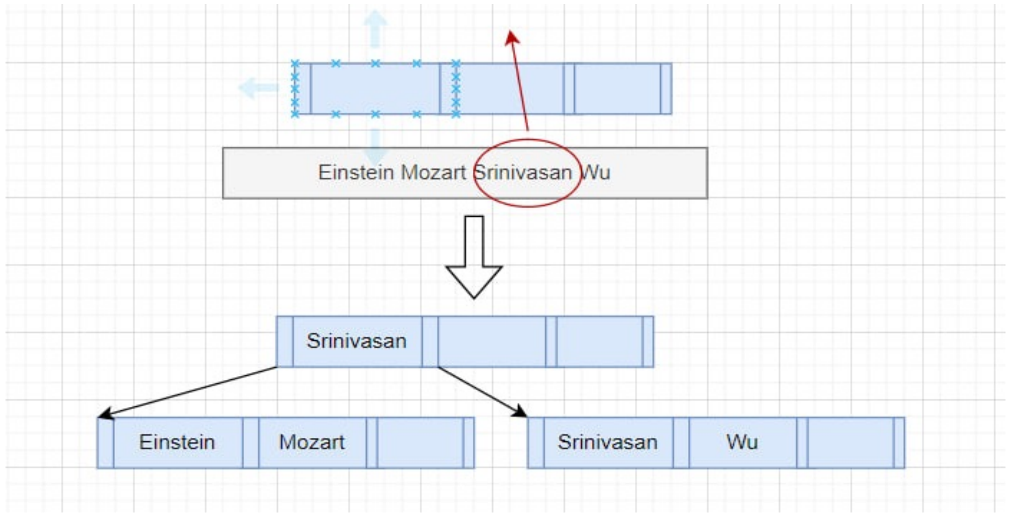
- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



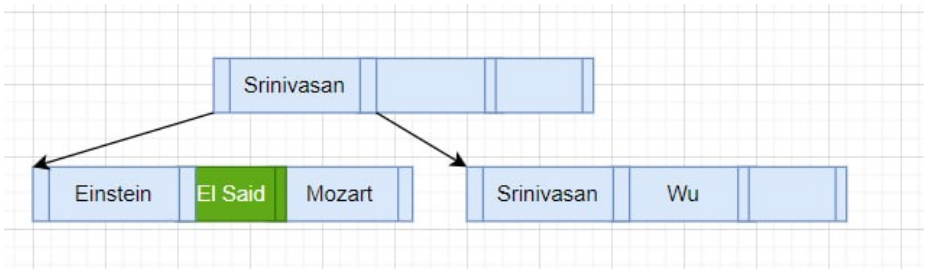
Insert Einstein

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim
- Node overflow. Split the node, and roll the middle value to the index node above it.



Insert ‘El Said’

- ➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



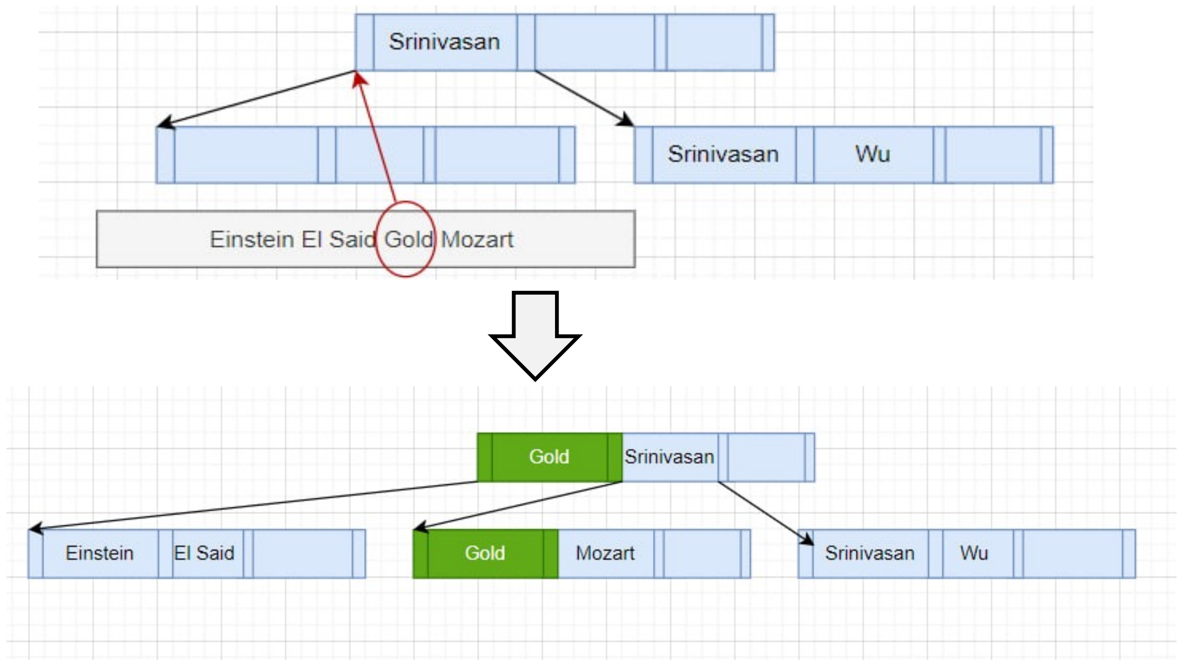
Insert ‘Gold’

- ➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look for ‘Gold’ in the leaf node it would go in and find there is no room.
- We split the node, and roll the right middle value ‘Gold’ to the index node above it.

Insert ‘Gold’

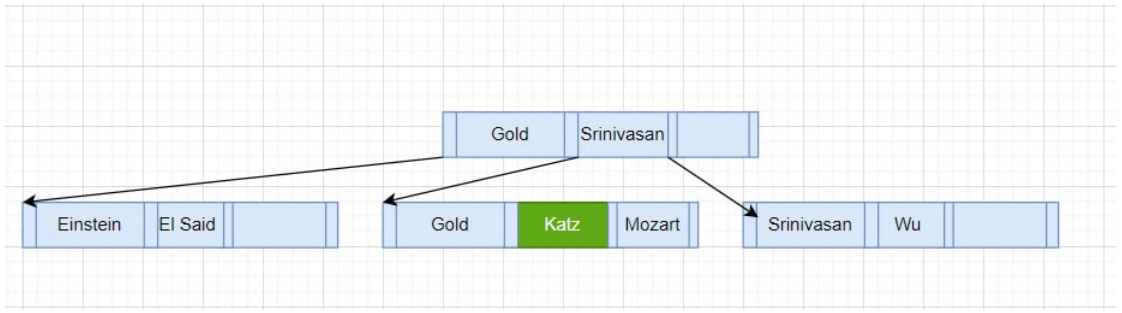
- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim



Insert ‘Katz’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

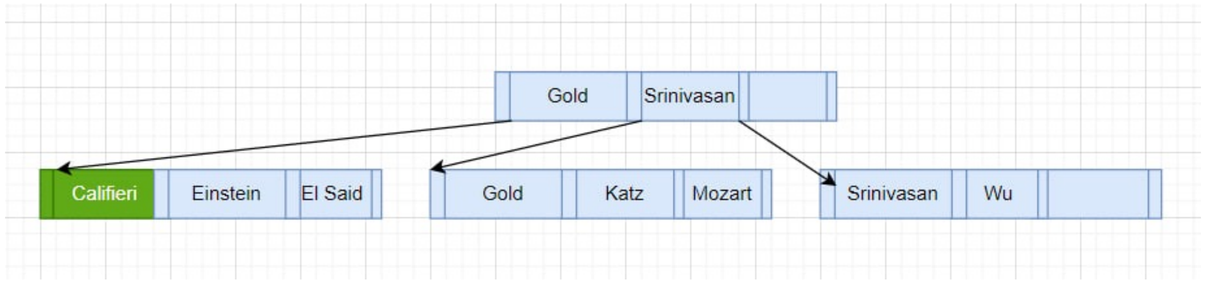
- We look at the leaf node to see if there is room for ‘Katz’.
- Finding an empty slot, we place the index in node in sorted order.



Insert 'Califieri'

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room for 'Califieri'.
- Finding an empty slot, we place the index in node in sorted order.



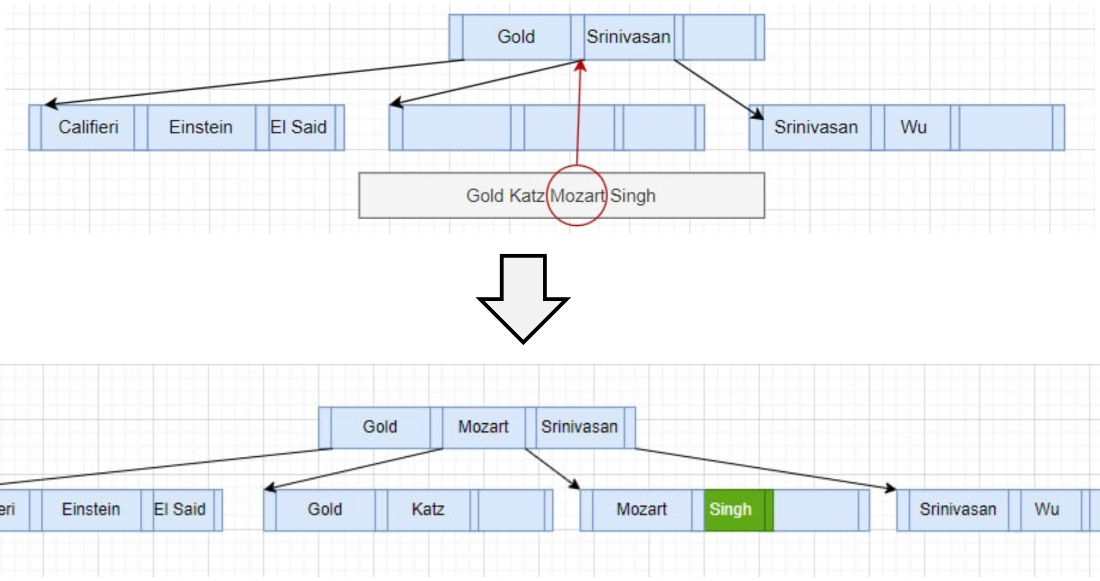
Insert ‘Singh’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look in the leaf node it would go in and find there is no room for ‘Singh’.
- We split the node, and roll the right middle value ‘Mozart’ to the index node above it.

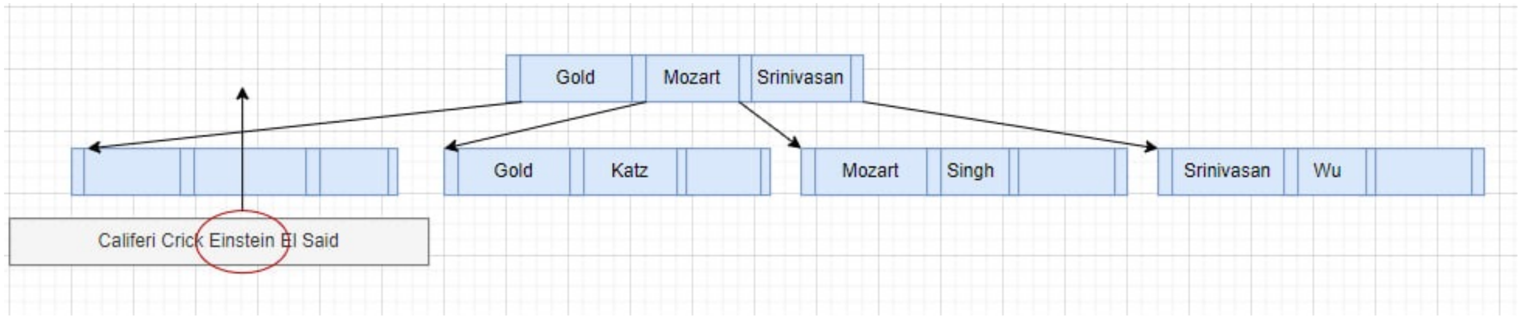
Insert 'Singh'

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim



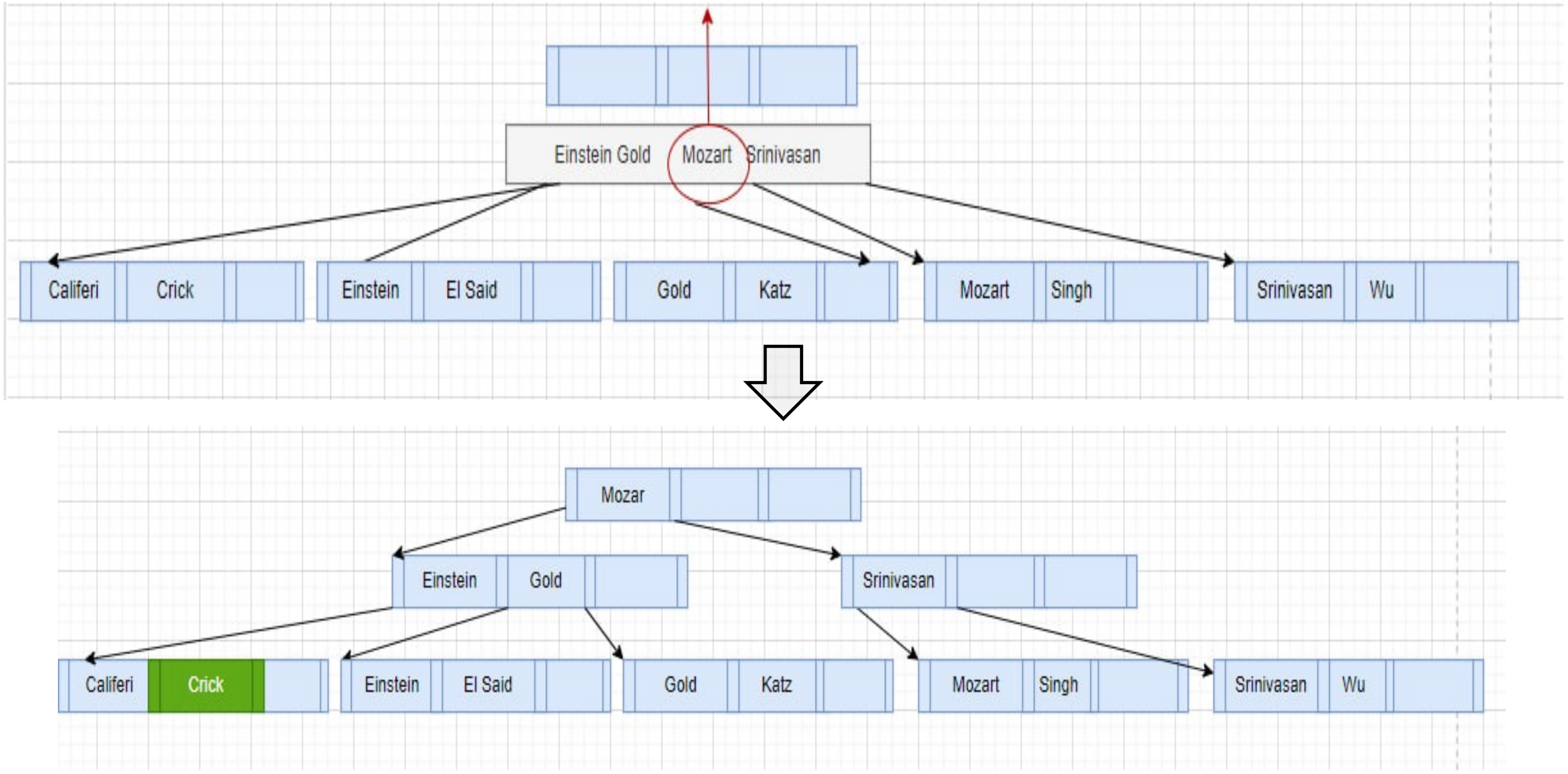
Insert ‘Crick’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim
- We look in the leaf node it would go in and find there is no room for ‘Crick’.
- We split the node, and roll the right middle value ‘Einstein’ to the index node above it.



Insert 'Crick'

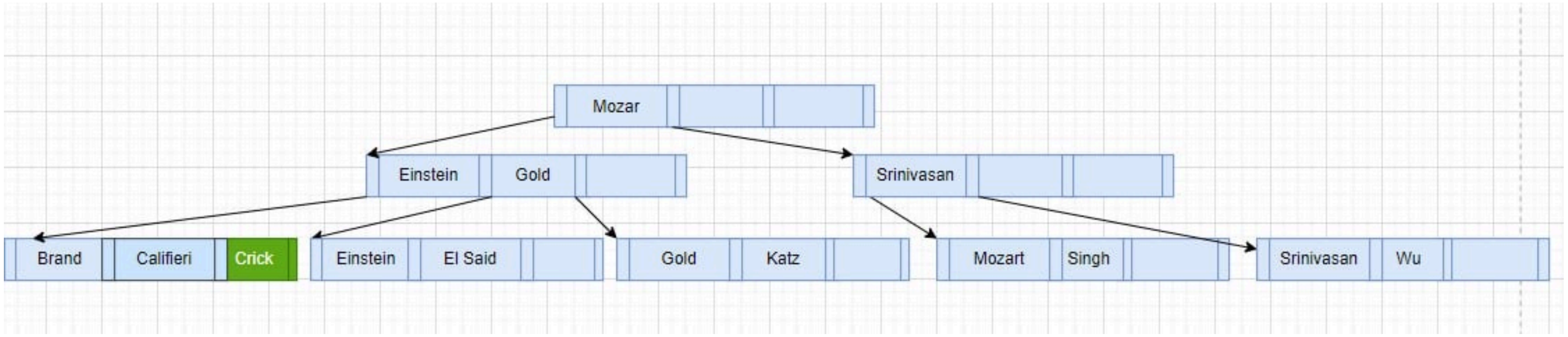
- 'Einstein' in the above node would go in and find there is no room.
- We split the node, and roll the right middle value 'Mozart' to the index node above it.



Insert ‘Brandt’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

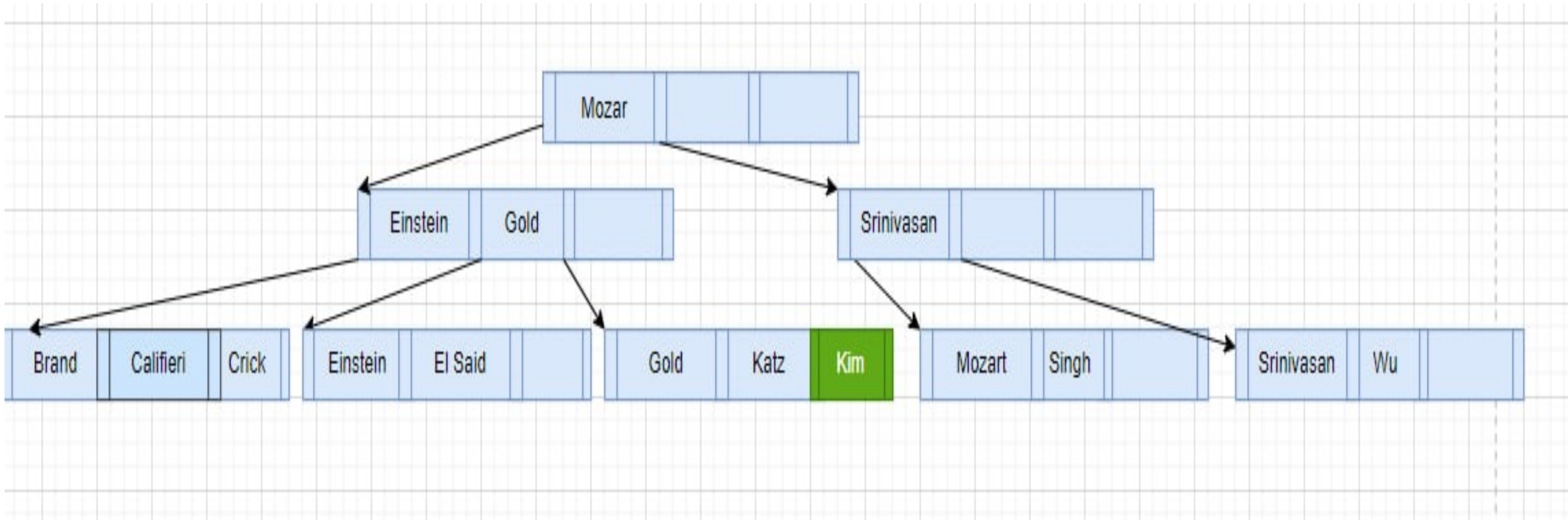
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



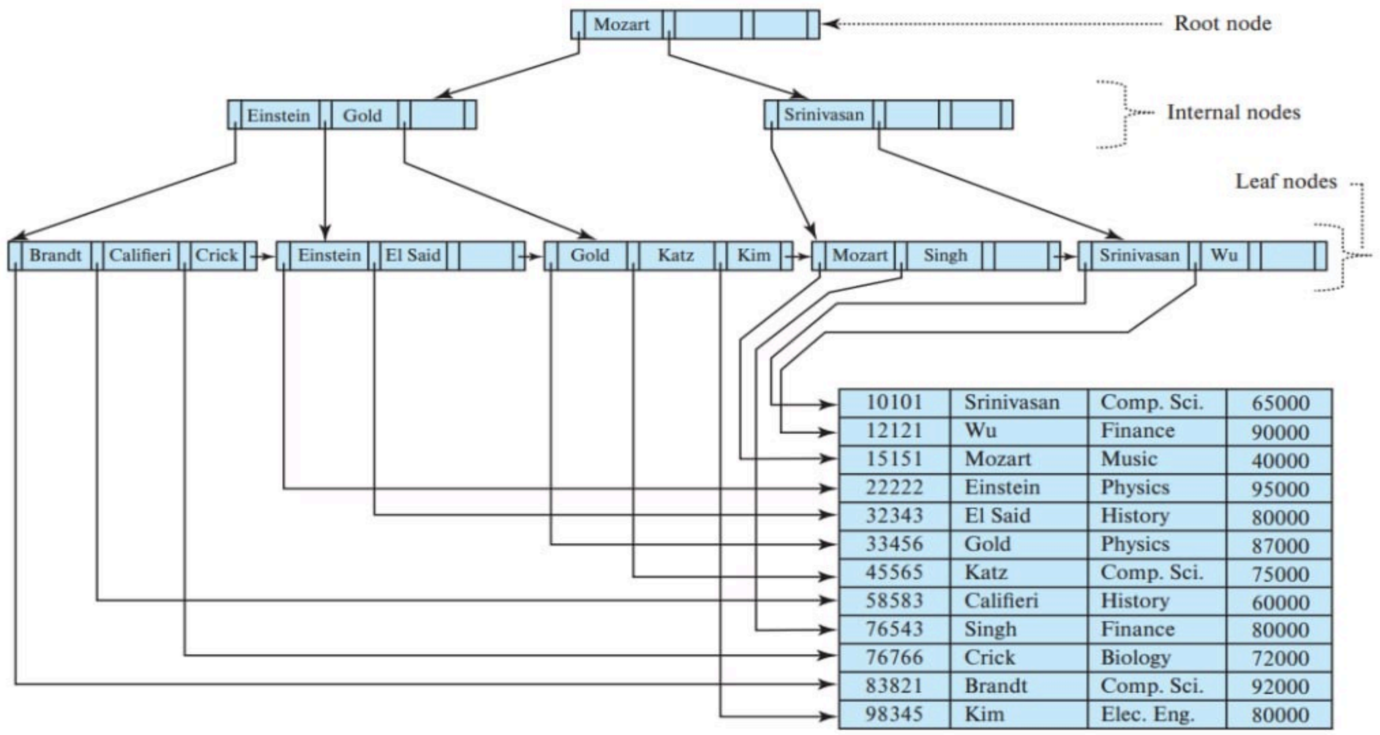
Insert ‘Kim’

- ➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room for ‘Kim’.
- Finding an empty slot, we place the index in node in sorted order.



B+ Tree

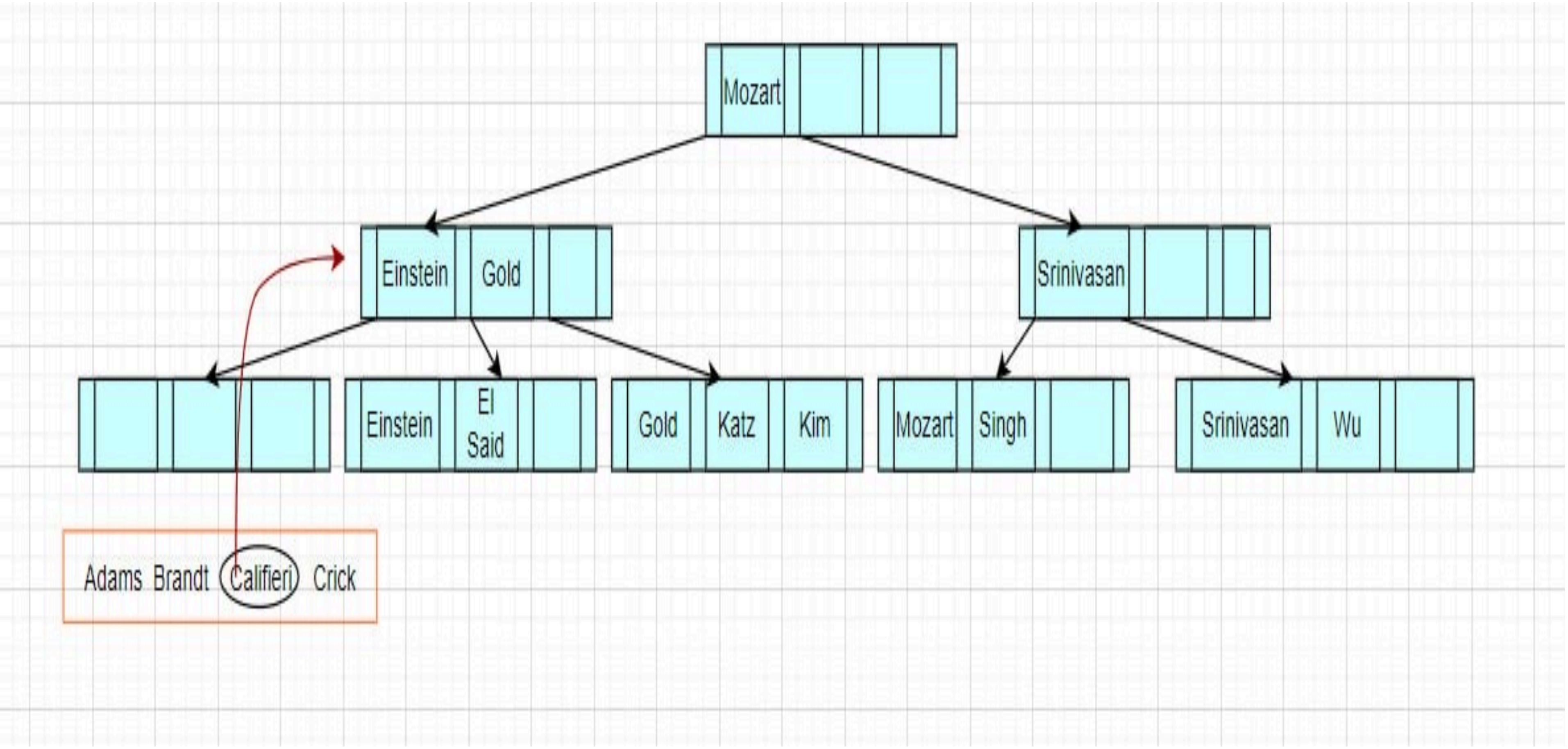


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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

Figure 14.9 B$^+$-tree for *instructor* file ($n = 4$).

Insert ‘Adams’

- We look in the leaf node it would go in and find there is no room for ‘Crick’.
- We split the node, and roll the right middle value ‘Califieri’ to the index node above it.



Insertion(Contd..)

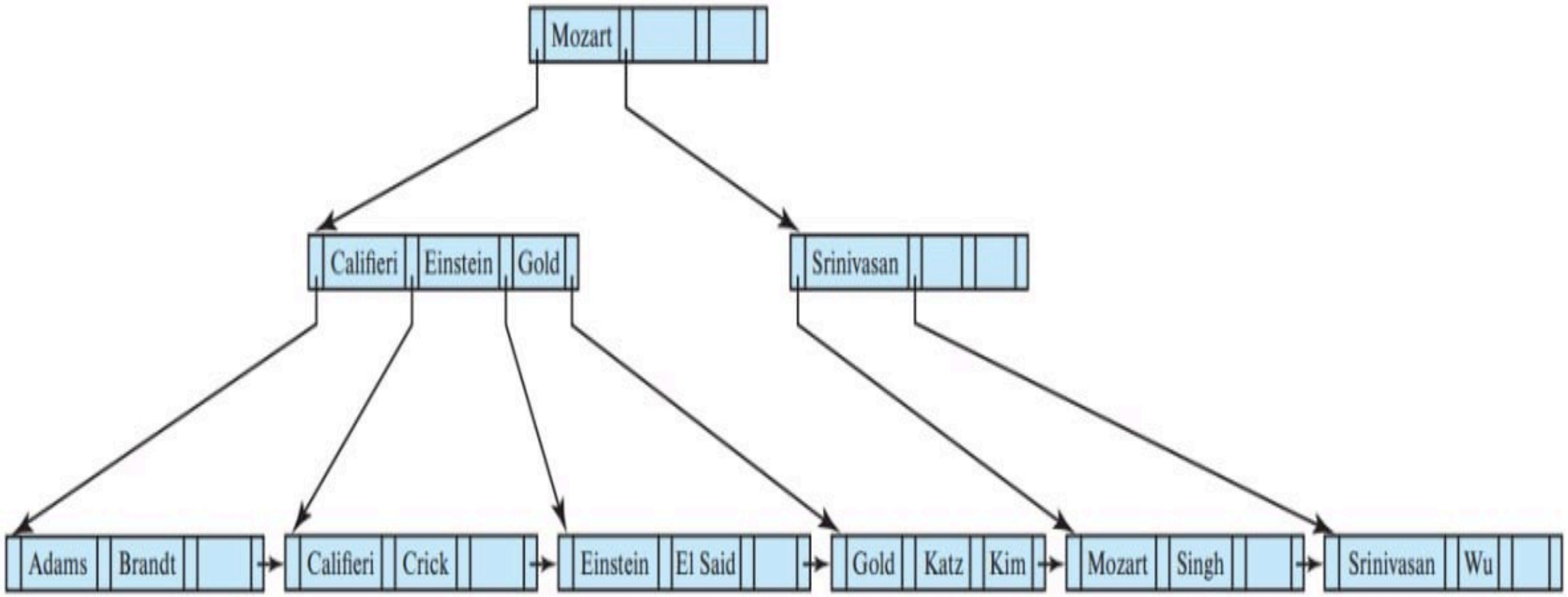
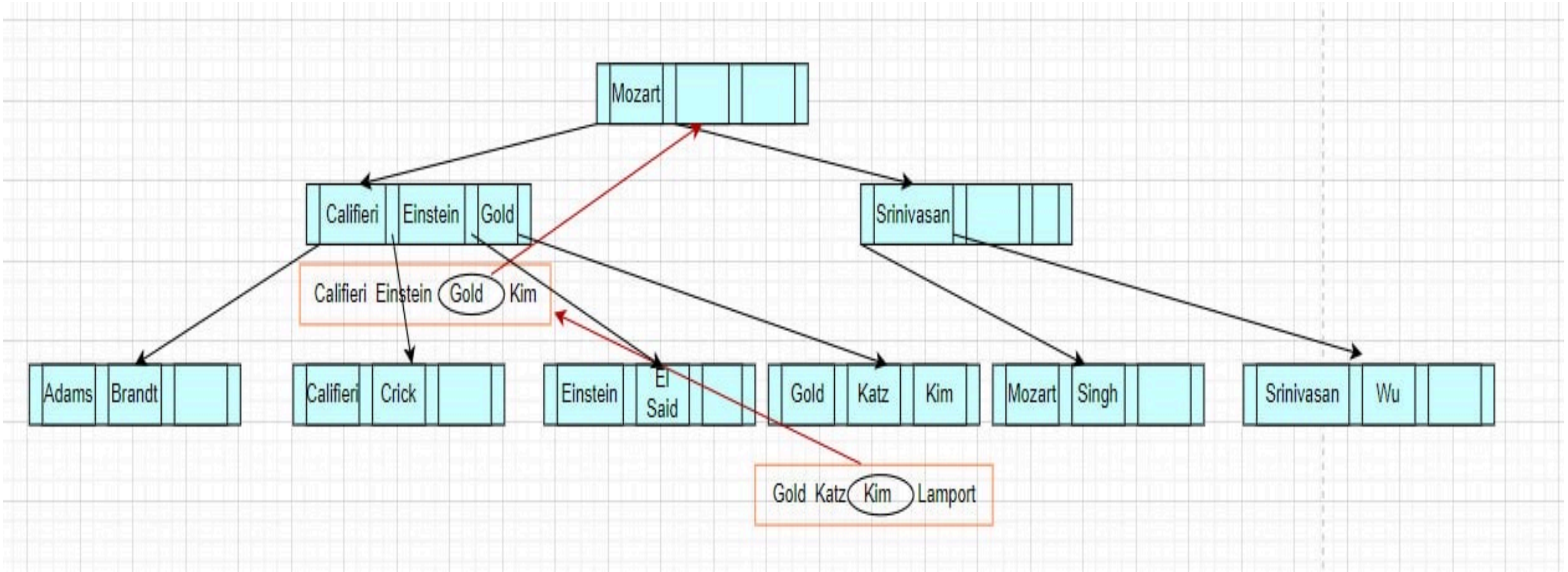


Figure 14.14 Insertion of "Adams" into the B^{+}-tree of Figure 14.9.

Insert ‘Lamport’

- We look at the leaf node it would go in and find there is no room for ‘Lamport’.
- We split the node, and roll the middle value ‘Kim’ to the index node above it.
- ‘Kim’ go to the above node and find there is no room. We split the node, and roll the middle value ‘Gold’ to the index node above it.



Insertion

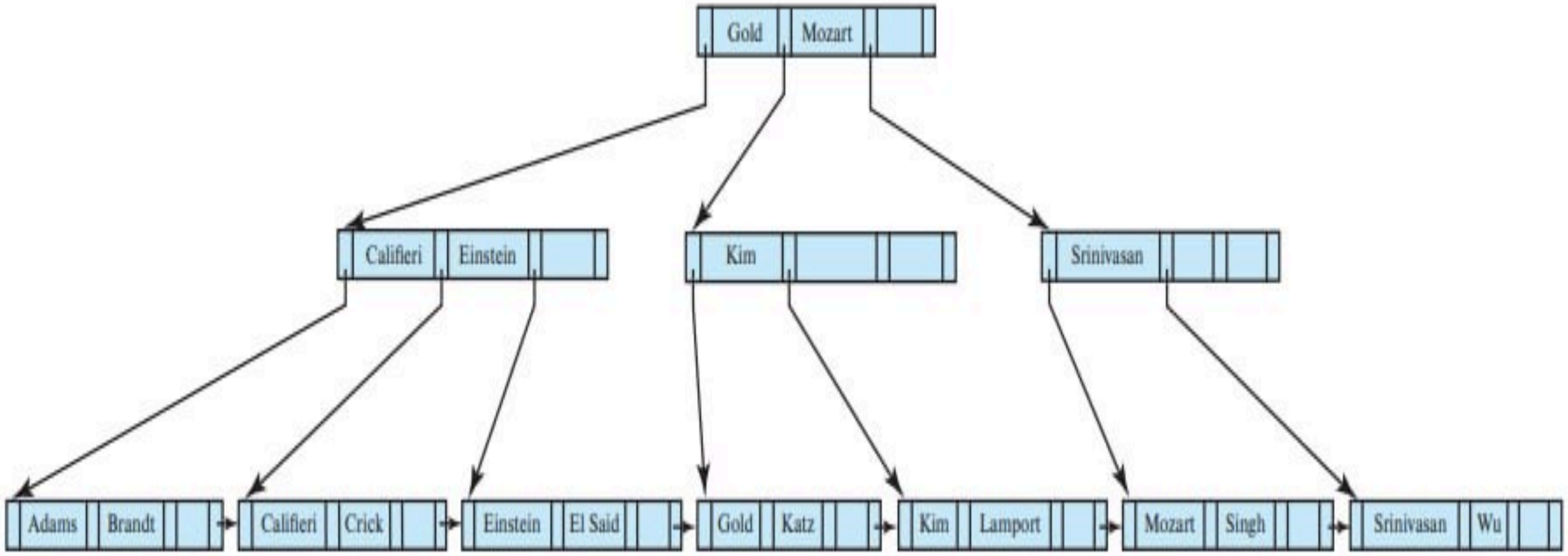


Figure 14.15 Insertion of "Lamport" into the B^+-tree of Figure 14.14.

Insertion in B+ Tree

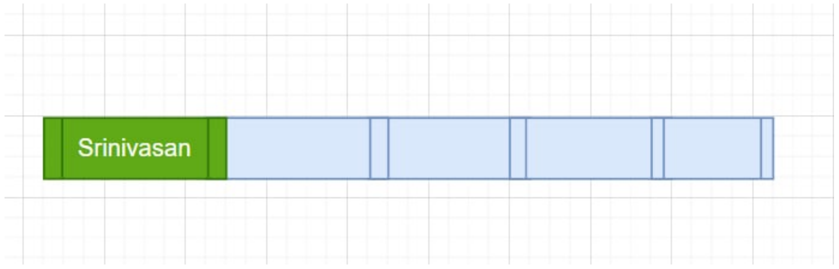
- Form a B+ tree of order m=6 by inserting the following keys sequentially:

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

Insert 'Srinivasan'

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

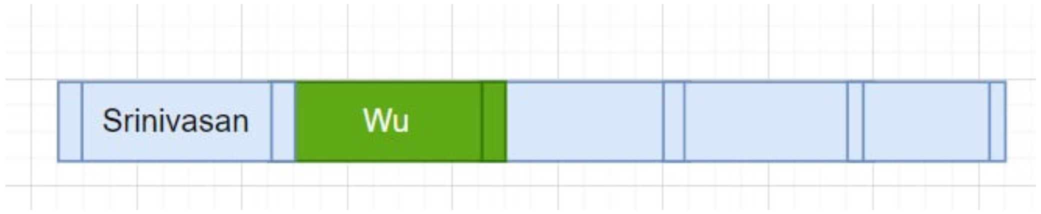
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert 'Wu'

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

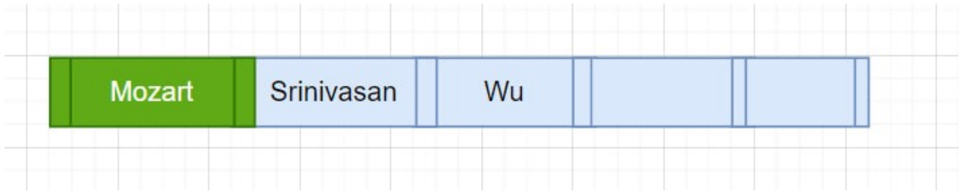
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘Mozart’

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

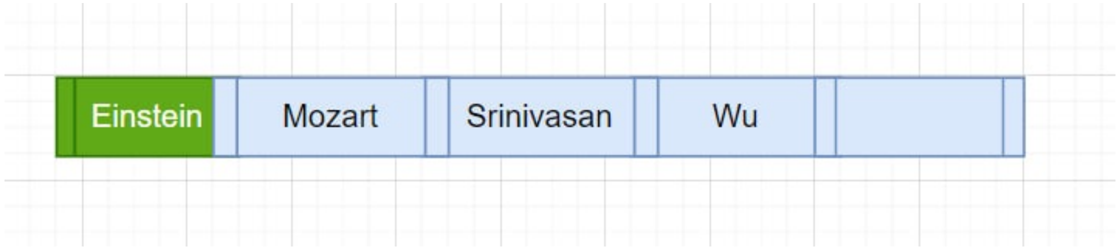
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert Einstein

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

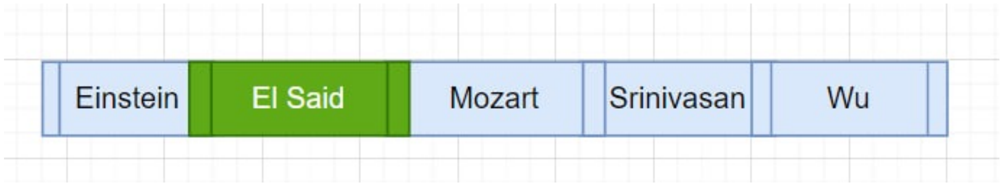
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘El Said’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



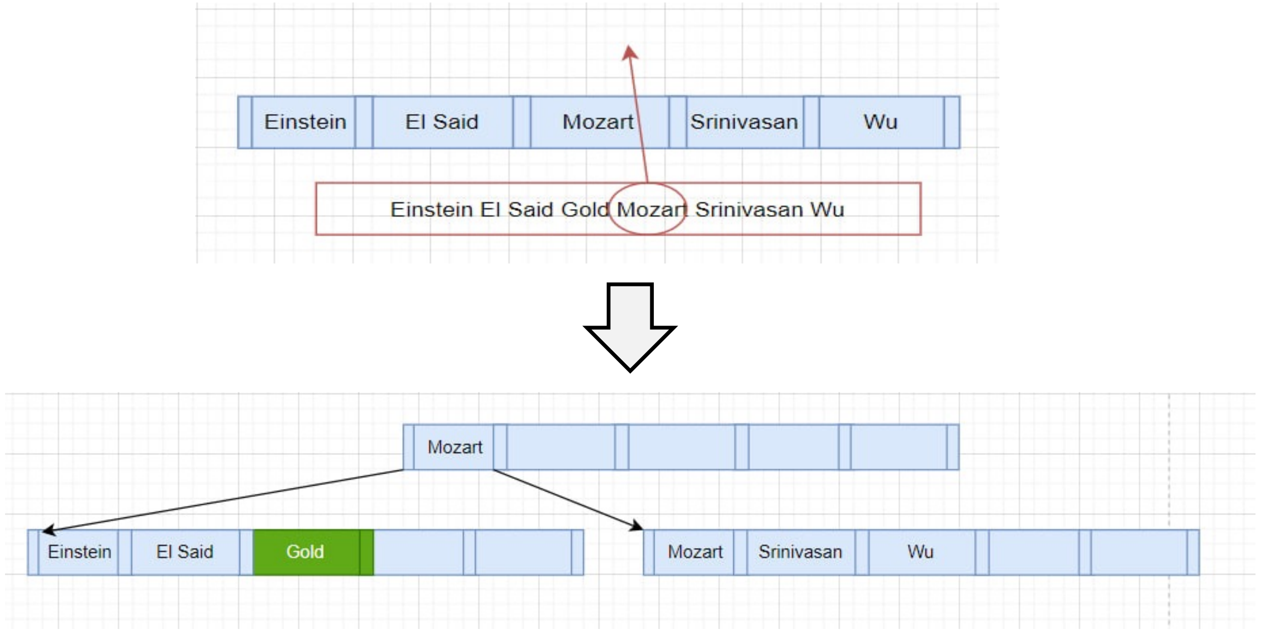
Insert ‘Gold’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look in the leaf node it would go in and find there is no room for ‘Gold’.
- We split the node, and roll the right middle value ‘Mozart’ to the index node above it.

Insert 'Gold'

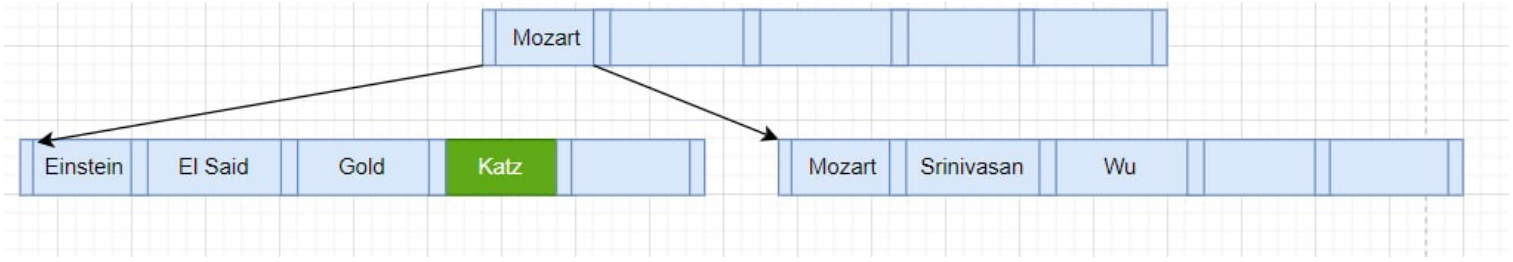
- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim



Insert 'Katz'

➢Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

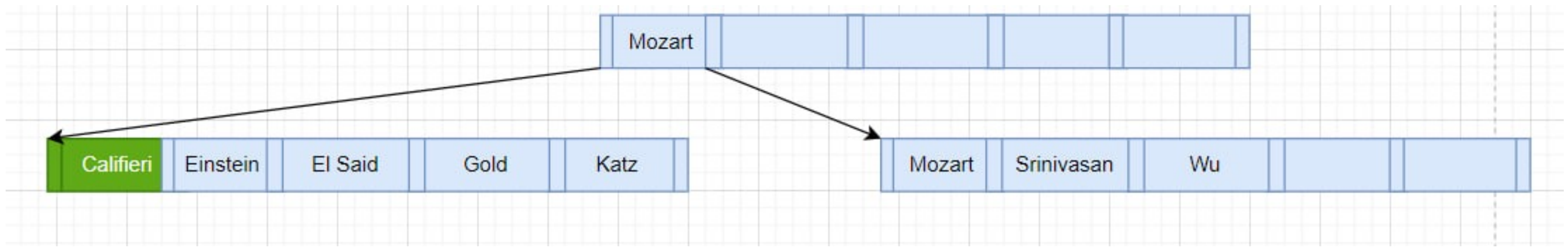
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘Califieri’

- ➢Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

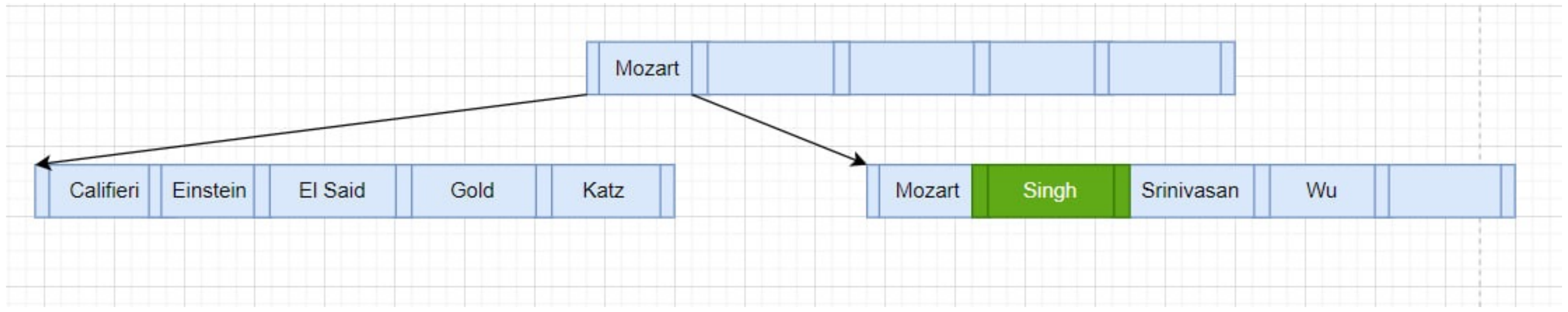
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘Singh’

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



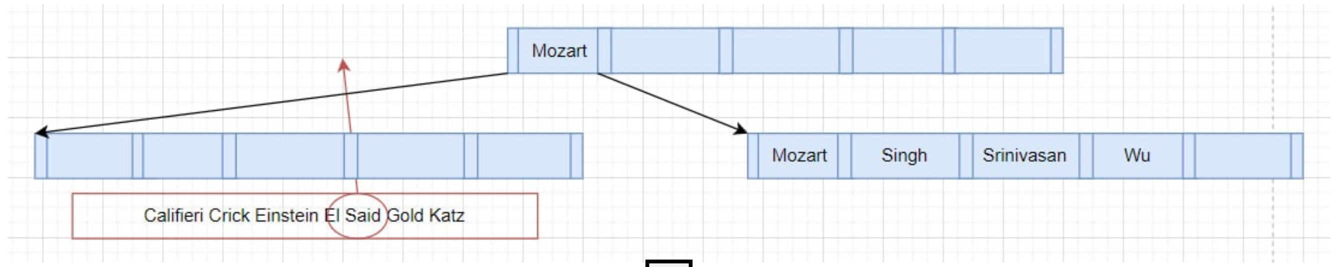
Insert ‘Crick’

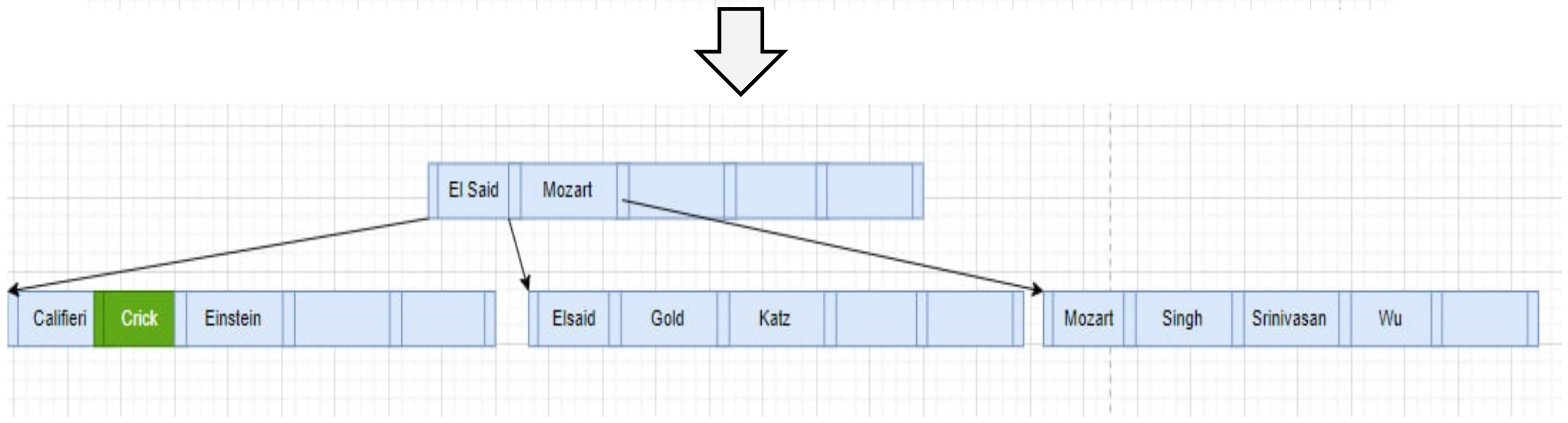
- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look in the leaf node it would go in and find there is no room for ‘Crick’.
- We split the node, and roll the right middle value ‘El Said’ to the index node above it.

Insert ‘Crick’

- Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

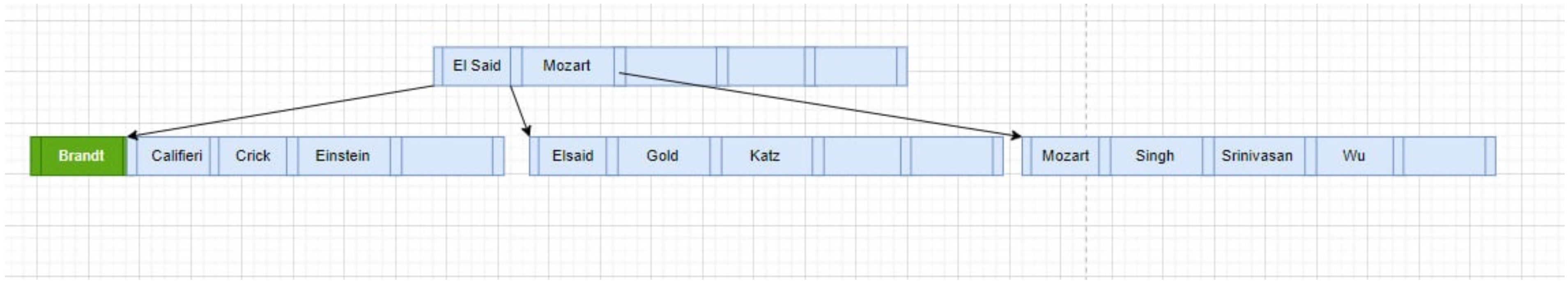




Insert ‘Brandt’

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

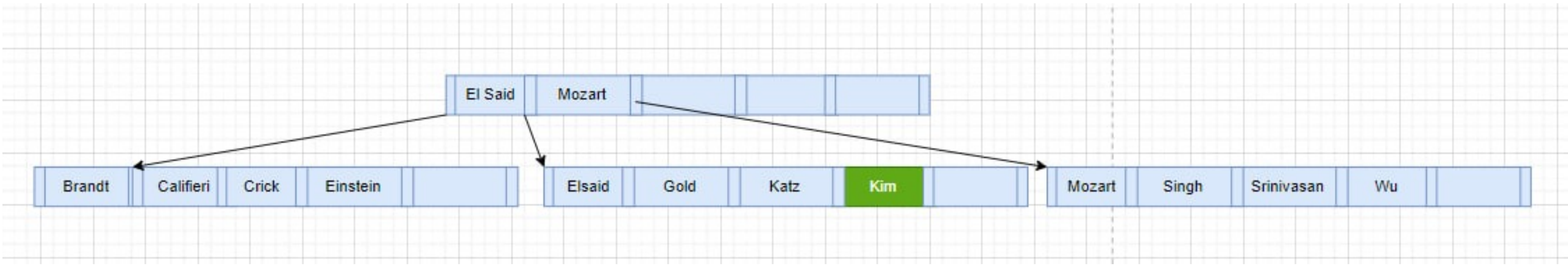
- We look at the leaf node to see if there is room.
- Finding an empty slot, we place the index in node in sorted order.



Insert ‘Kim’

➢ Srinivasan, Wu, Mozart, Einstein, El Said, Gold, Katz, Califieri, Singh, Crick, Brandt, Kim

- We look at the leaf node to see if there is room for ‘Kim’.
- Finding an empty slot, we place the index in node in sorted order.

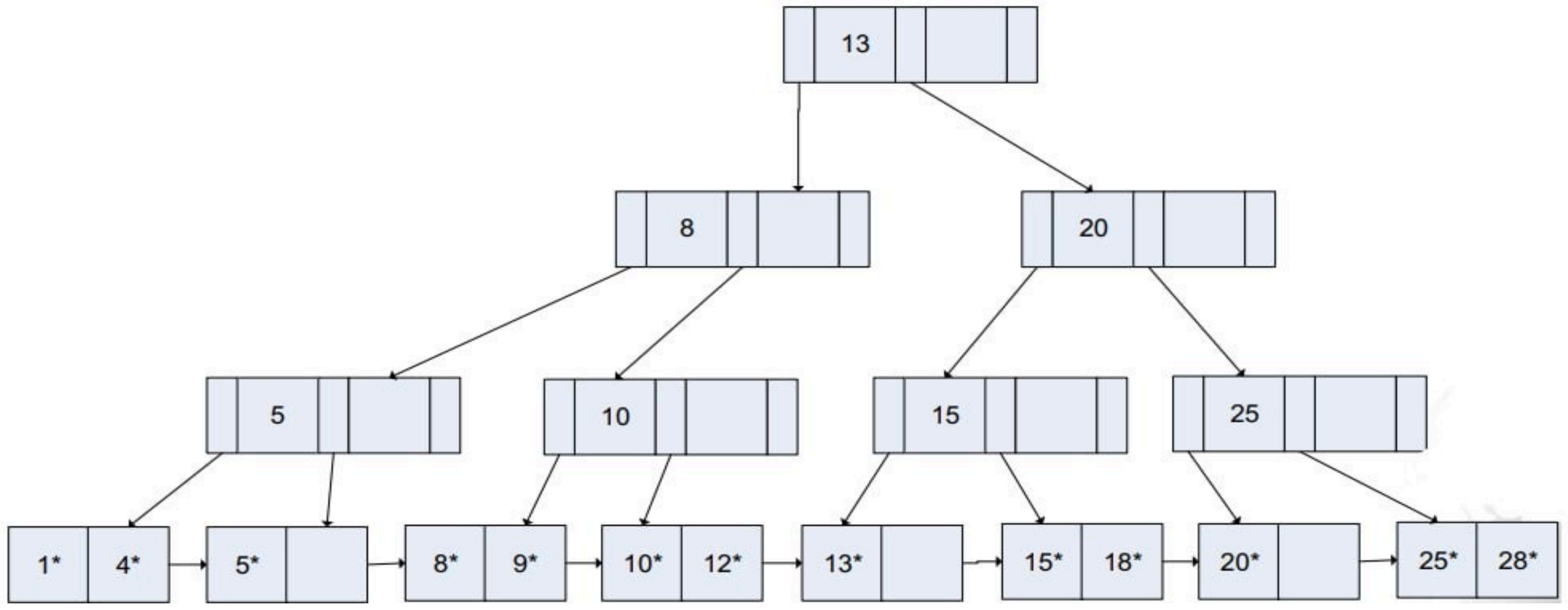


Searching in B+ Tree

- Start at the Root: Begin searching at the root.
- Compare Keys: Compare the search key with the current node's keys.
- Navigate to Child: Move to the appropriate child based on the comparison. If compared key is smaller then move to left child else move to right child.
- Repeat Until Leaf: Keep moving down the tree until reaching a leaf.
- Search in Leaf: Look for the key in the leaf node.
- Return Result: If found, return the associated value; if not, indicate the key isn't in the tree.

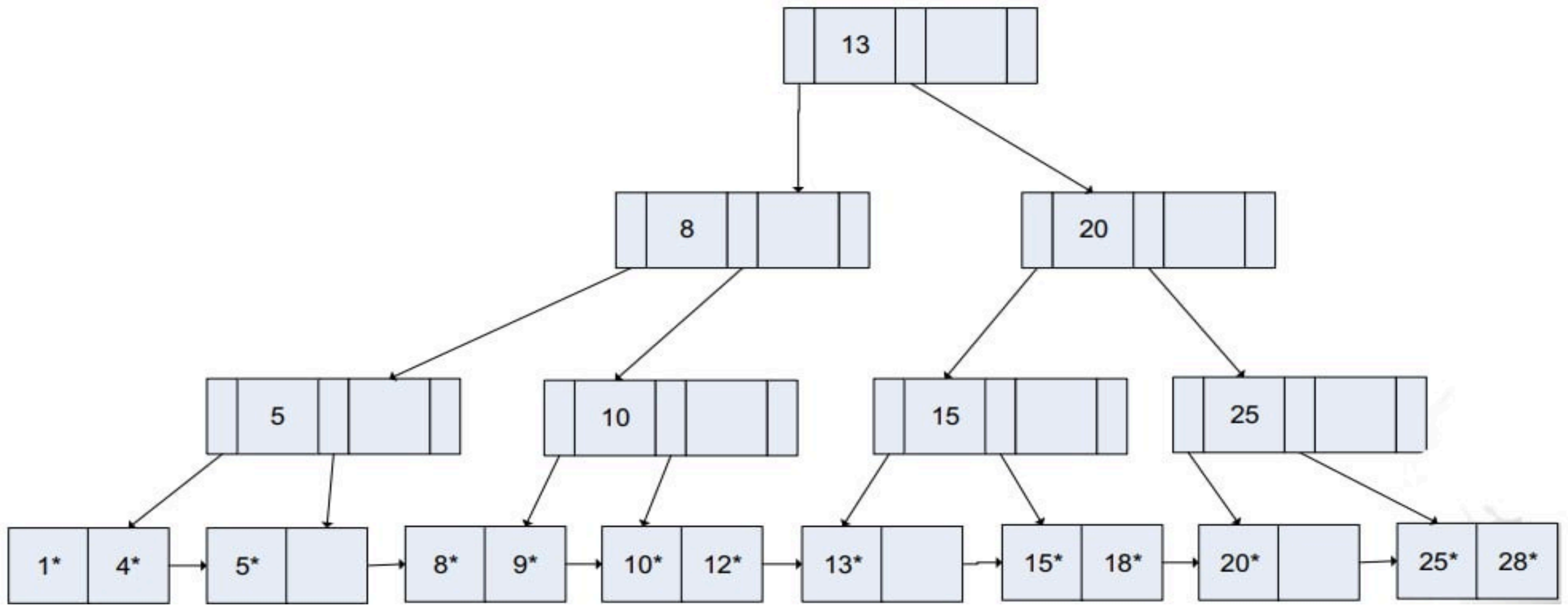
Searching(Contd..)

- Search 20 from the below tree.



Searching

- Search 10 from the below tree.

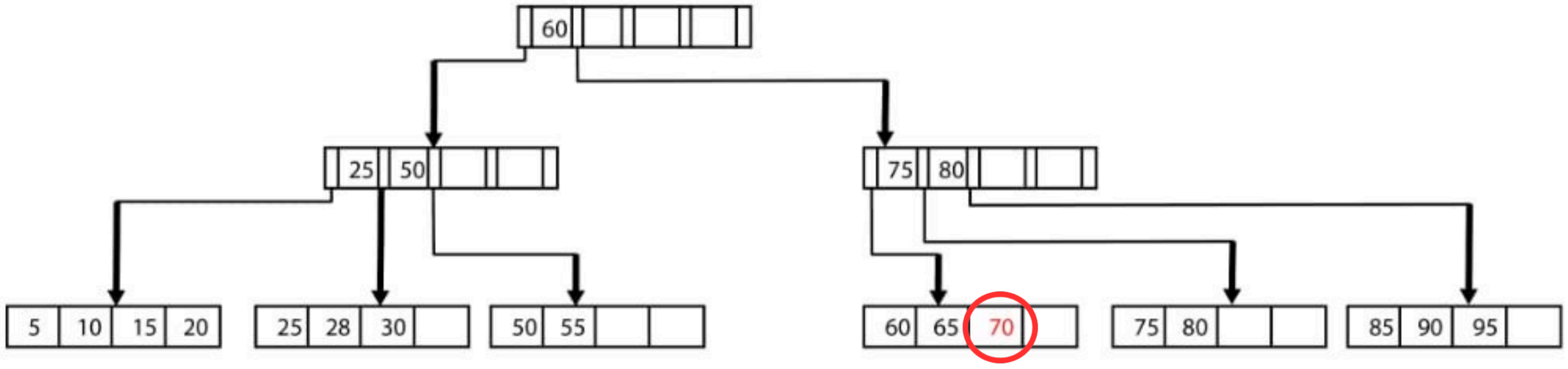


Deletion from B+ Tree

When deleting keys from a B+ tree:

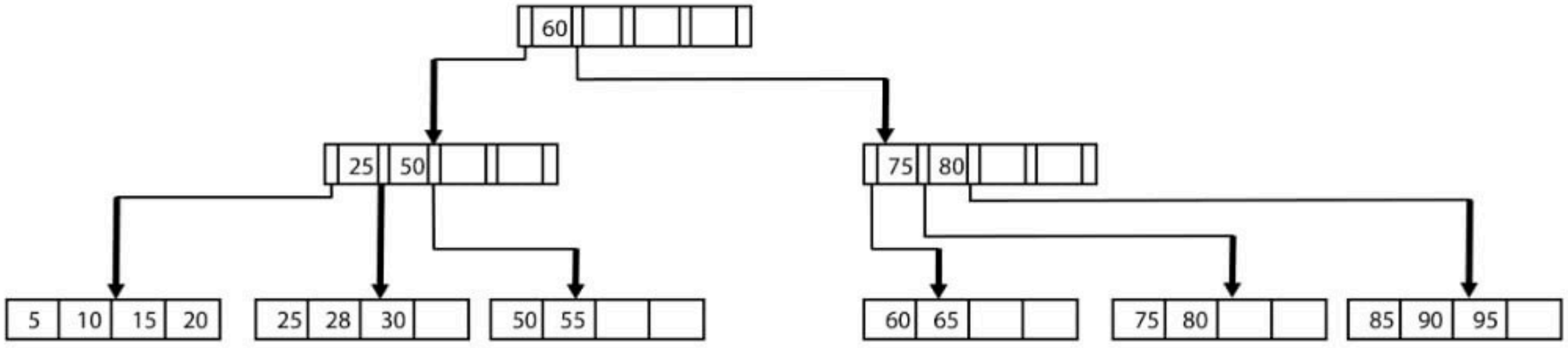
1. Find the Key: Locate the key to delete.
2. Handle Leaf Nodes: If the key is in a leaf, remove it. If the leaf becomes too empty, balance it.
3. Handle Internal Nodes: If the key is in an internal node, replace it with the largest key from the left or smallest from the right subtree, and recursively delete it from the leaf.
4. Rebalance the Tree: After deletion, ensure the tree remains balanced by merging or redistributing nodes.
5. Update Parent Keys: Update parent keys to reflect changes in the subtree structure.
6. Adjust Root Node: If the root becomes empty, adjust the tree's structure.
7. Maintain B+ Tree Rules: Throughout, maintain rules like the minimum/maximum keys per node and sorted order.

Deletion(Contd..)



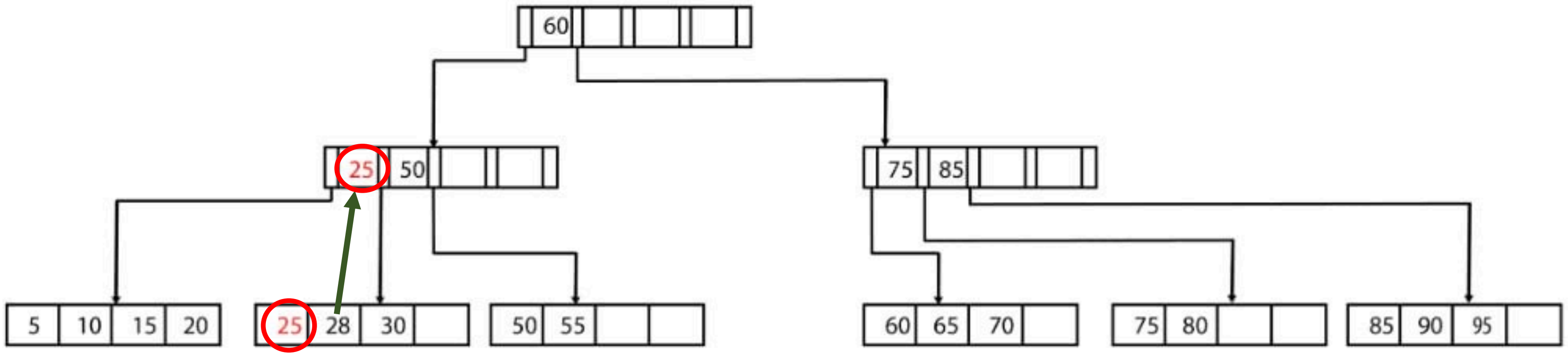
Before Deletion

Deletion(Contd..)



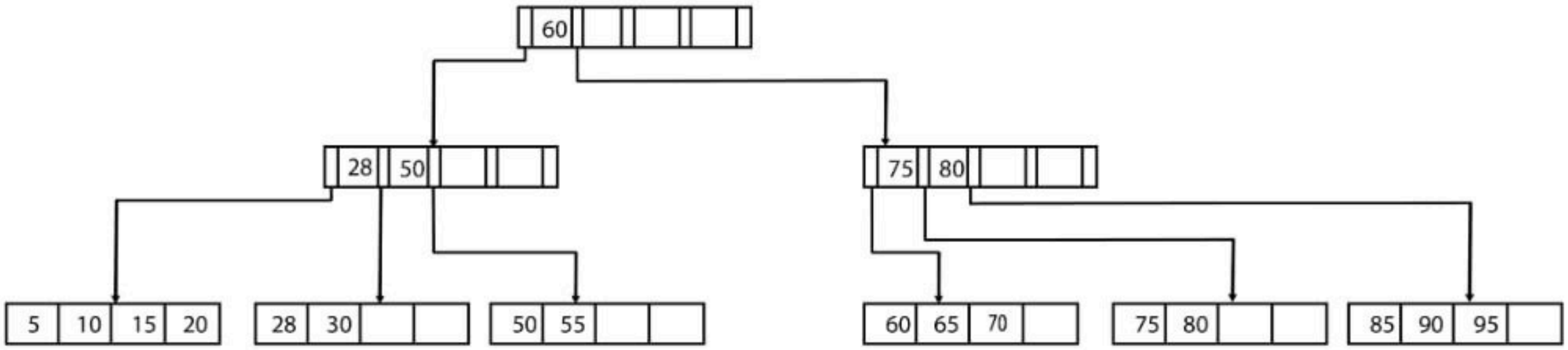
After Deletion

Deletion(Contd..)



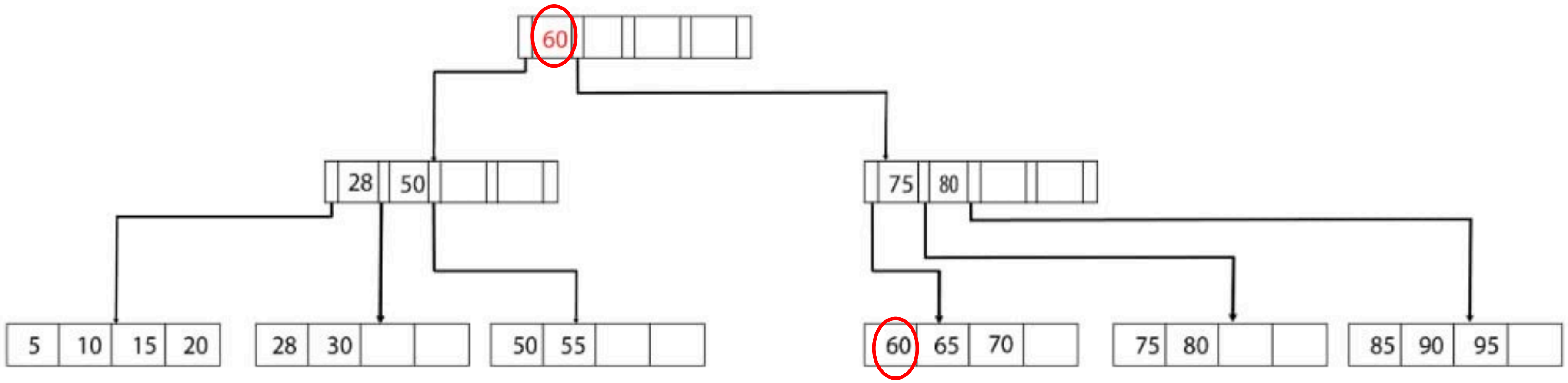
Before Deletion

Deletion(Contd..)



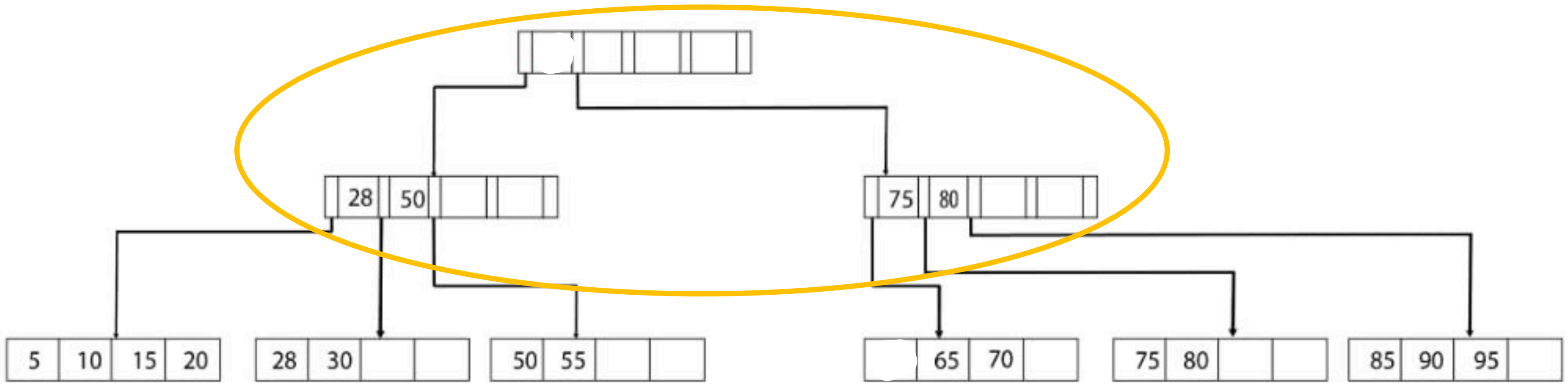
After Deletion

Deletion(Contd..)



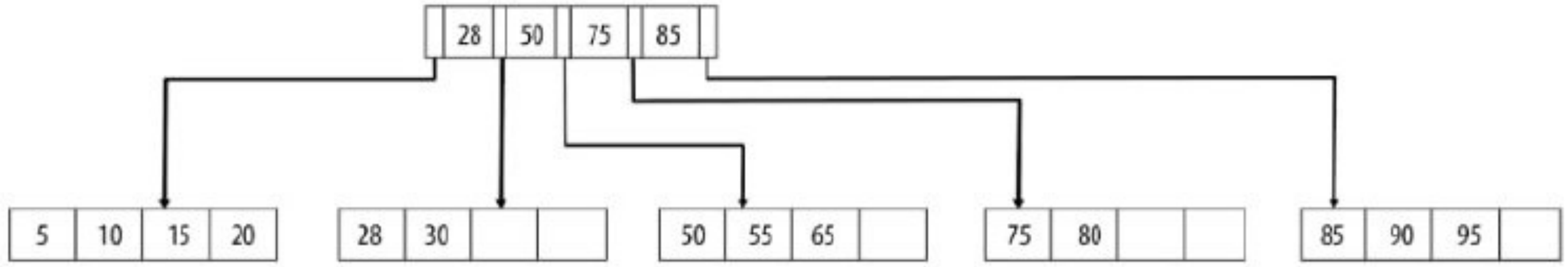
Before Deletion

Deletion(Contd..)



Before Deletion

Deletion



After Deletion

Deletion(Contd..)

- Delete 'Srinivasan' from the tree below:

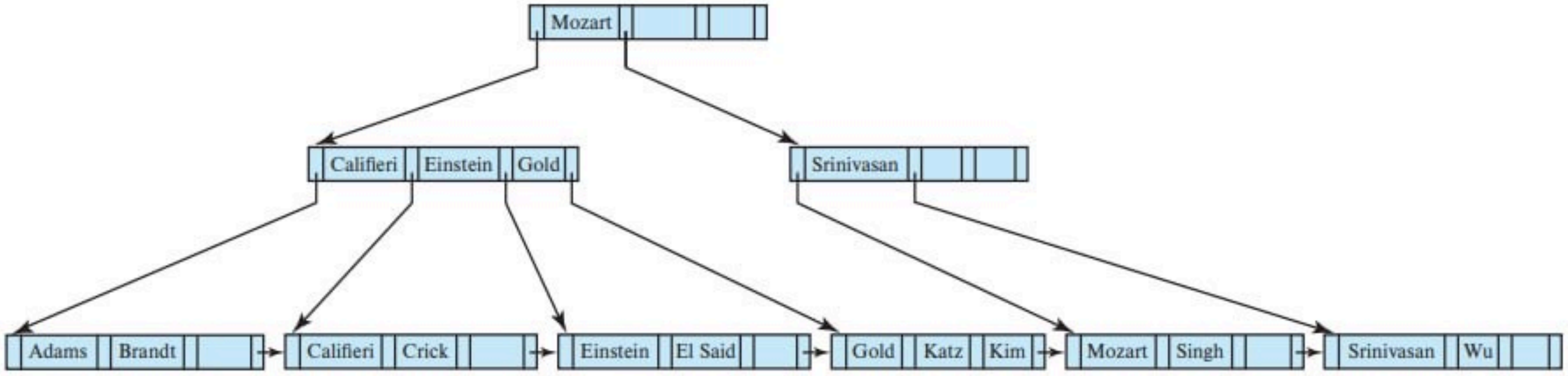
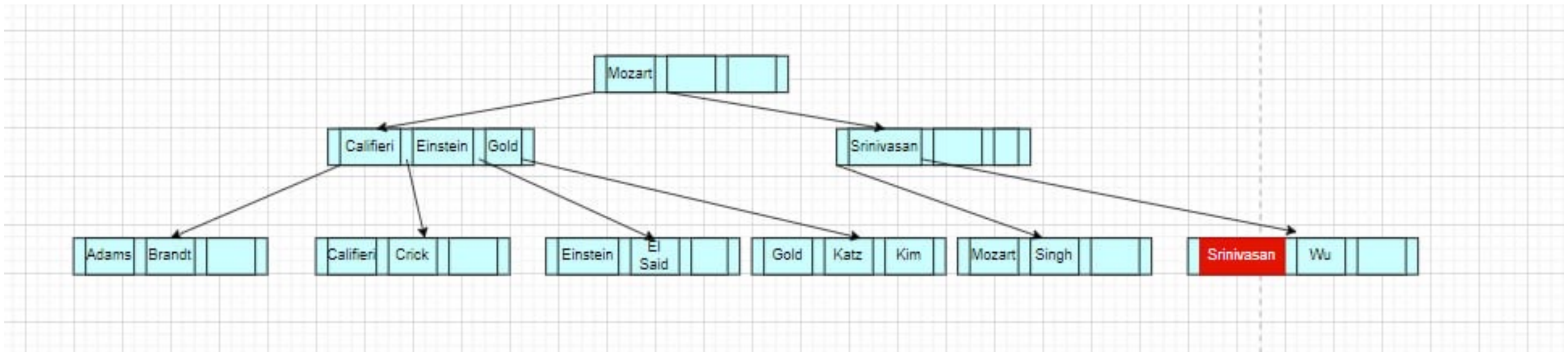
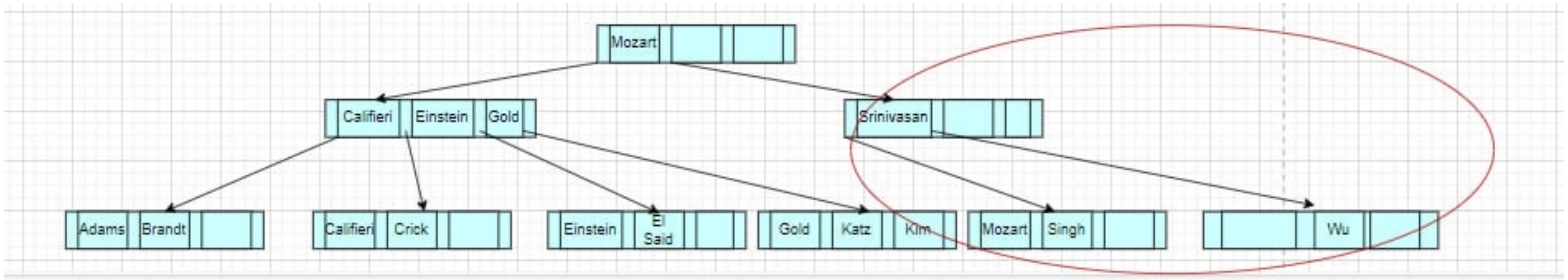


Figure 14.14

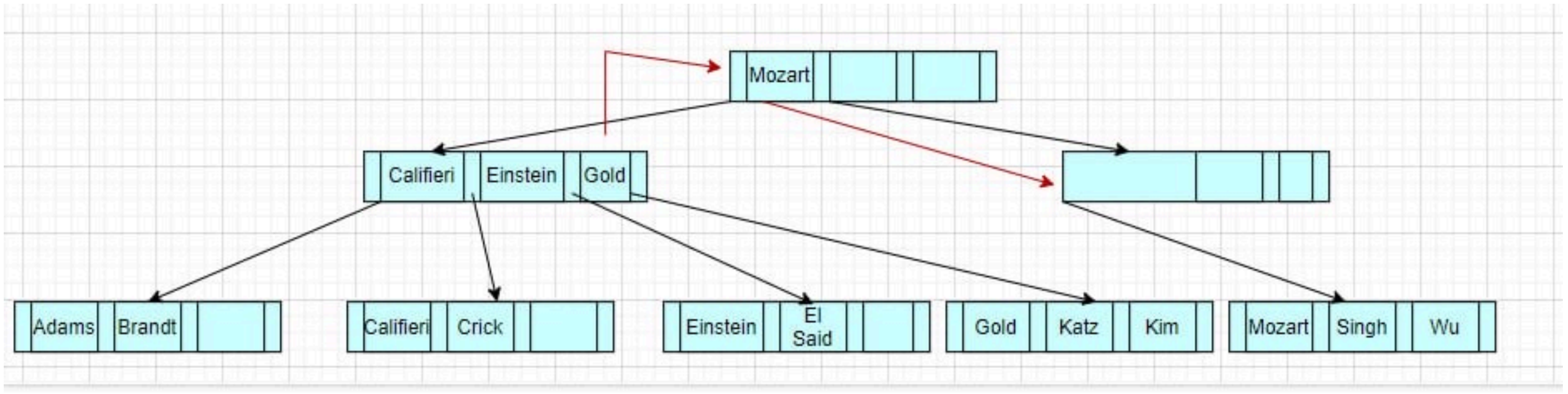
Deletion(Contd..)



Deletion(Contd..)



Deletion(Contd..)



Deletion(Contd..)

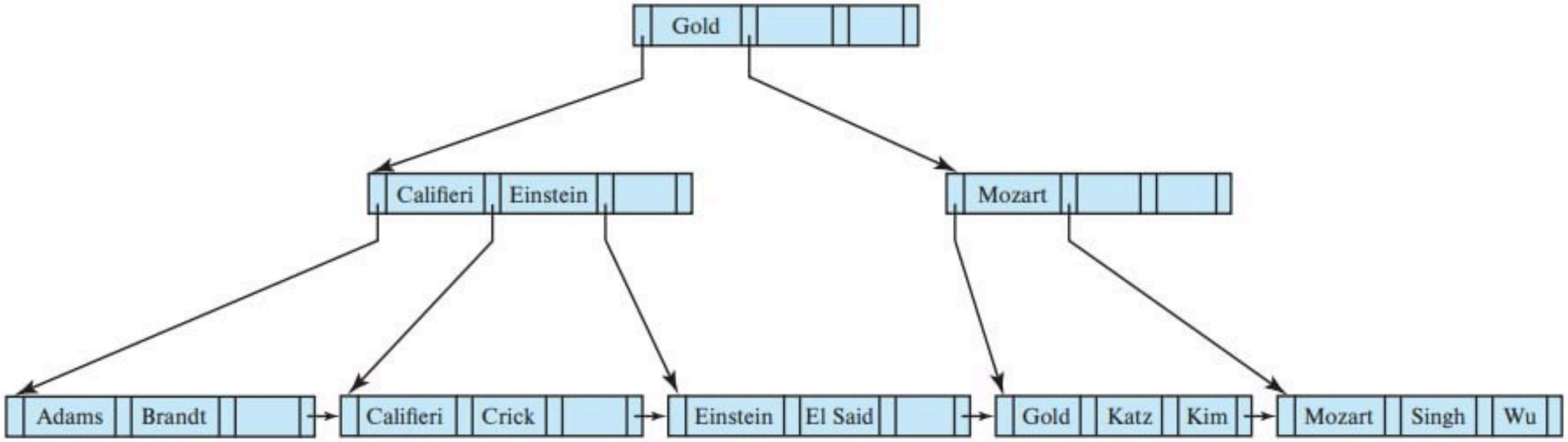
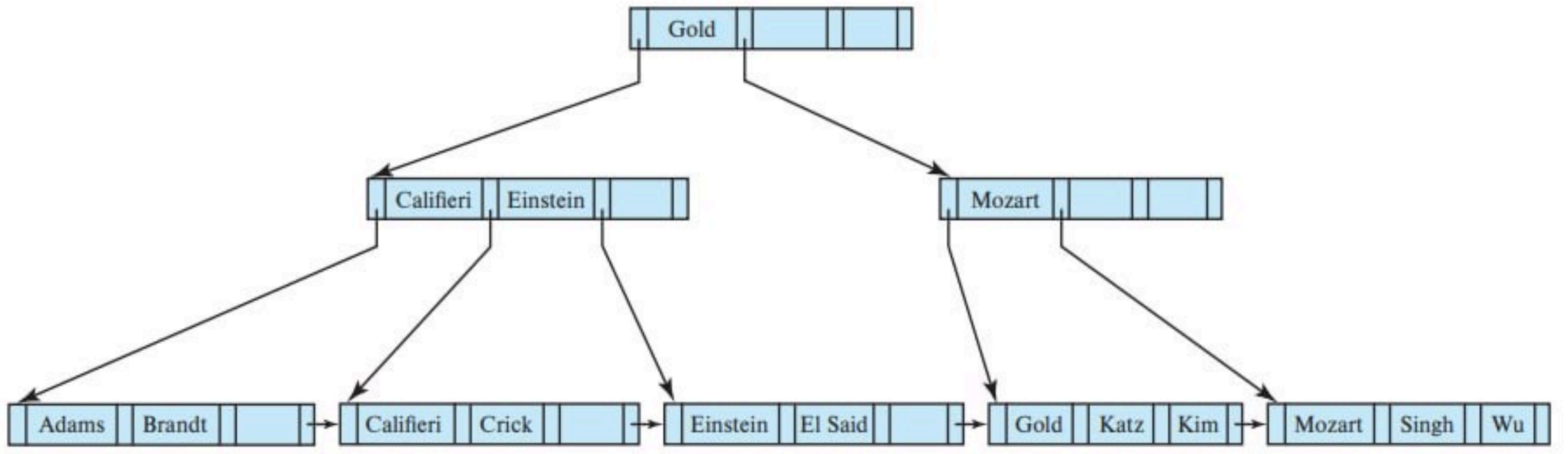


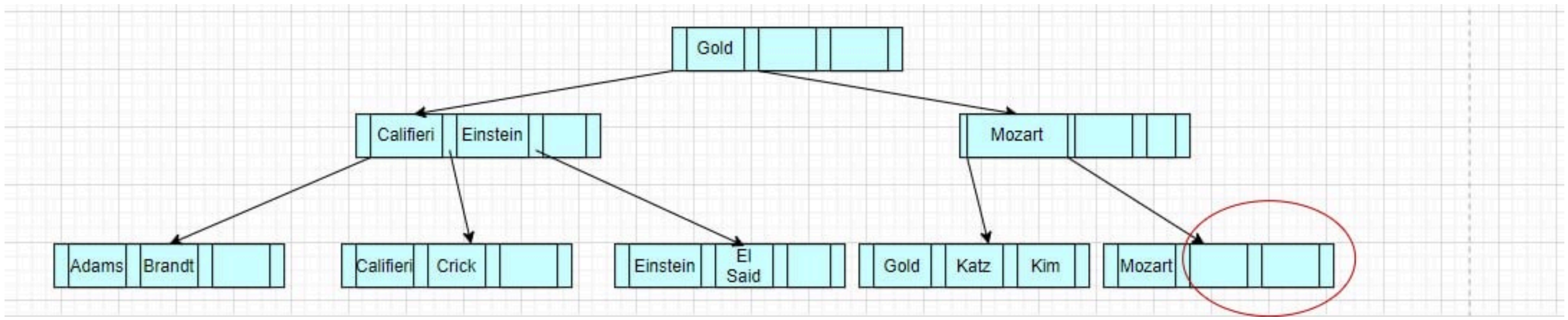
Figure 14.18 Deletion of “Srinivasan” from the B$^+$-tree of Figure 14.14.

Deletion(Contd..)

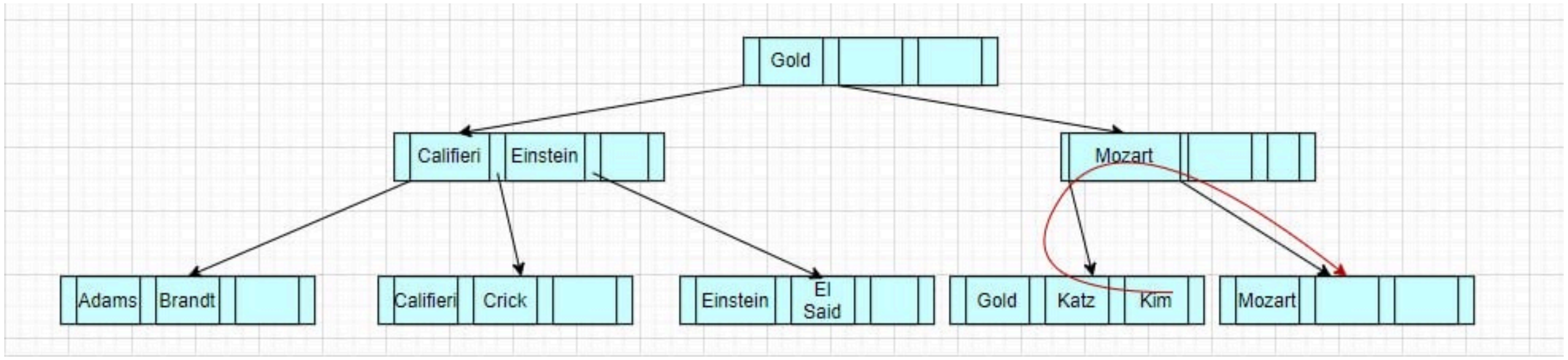
- Delete ‘Singh’ and ‘Wu’ from the below tree



Deletion(Contd..)



Deletion(Contd..)



Deletion(Contd..)

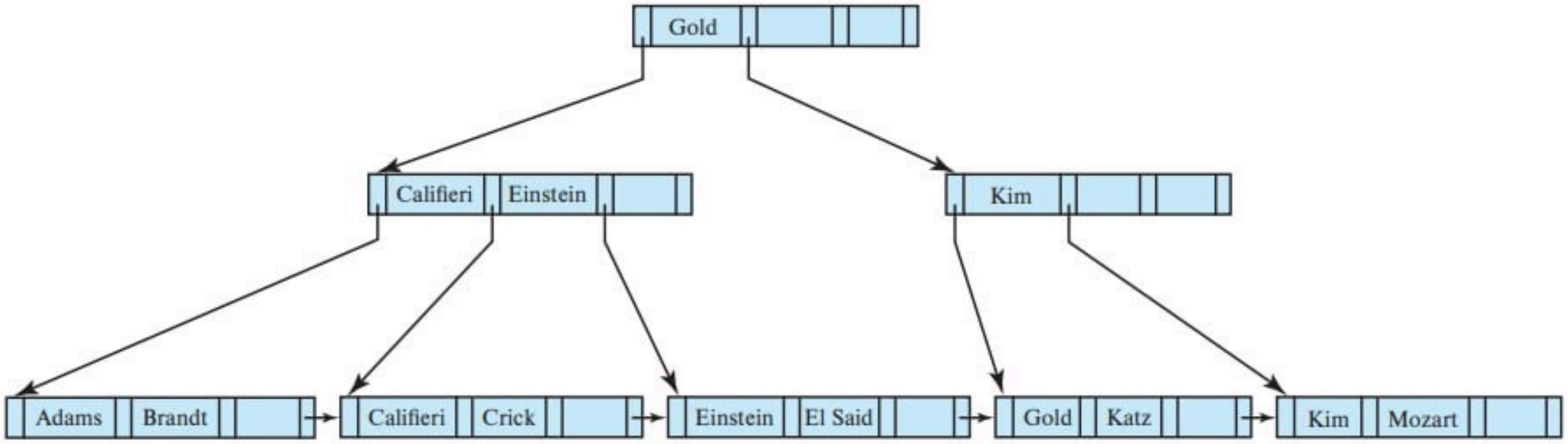
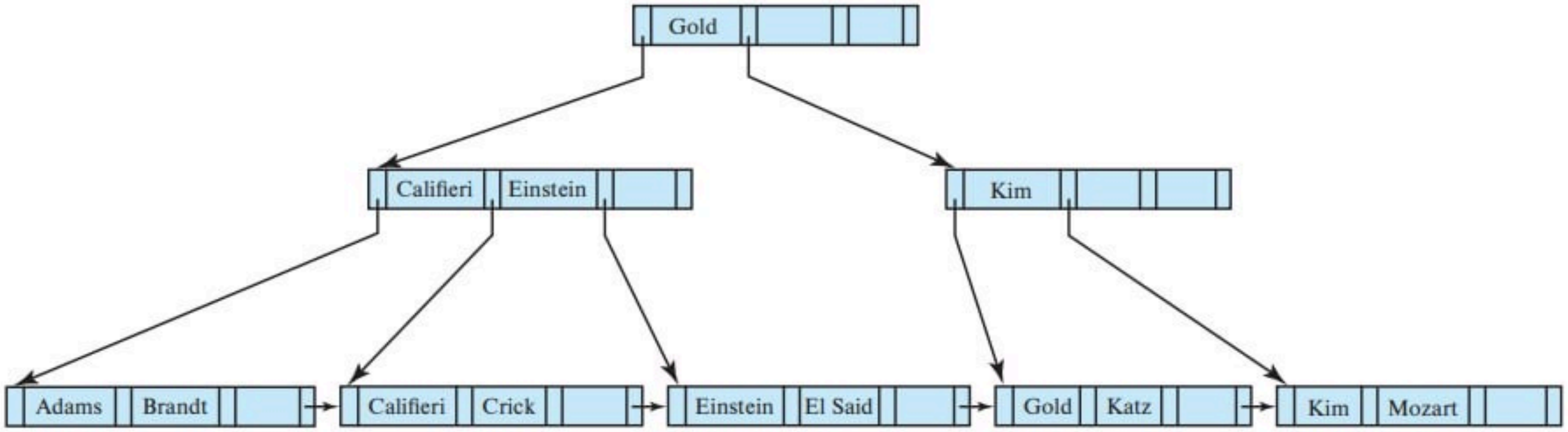


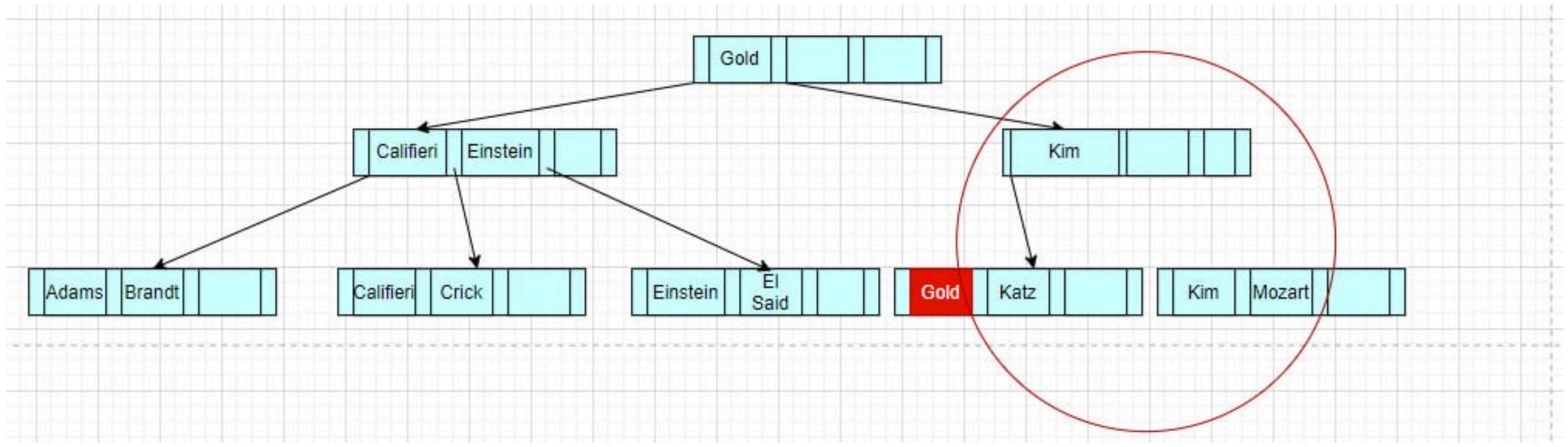
Figure 14.19 Deletion of "Singh" and "Wu" from the B$^+$-tree of Figure 14.18.

Deletion(Contd..)

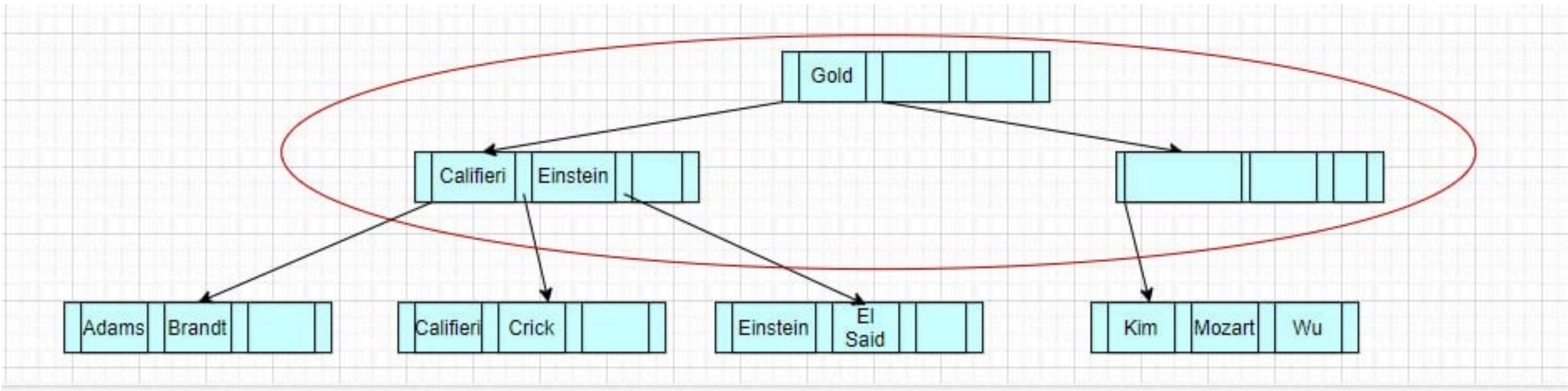
- Delete ‘Gold’ from the tree below:



Deletion(Contd..)



Deletion(Contd..)



Deletion

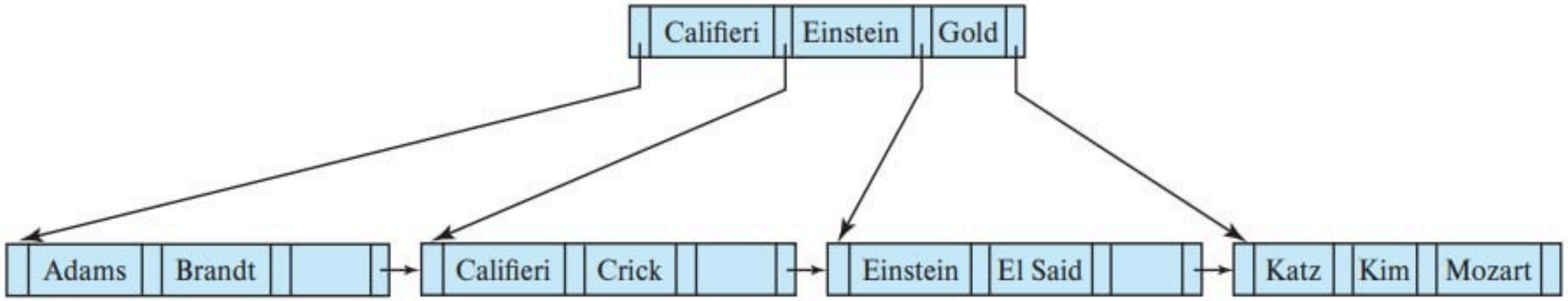


Figure 14.20 Deletion of "Gold" from the B⁺-tree of Figure 14.19.

Thank You