B+ Trees

- What are B+ Trees used for
- What is a B Tree
- What is a B+ Tree
- Searching
- Insertion
- Deletion

What are B+ Trees Used For?

- When we store data in a table in a DBMS we want
  - Fast lookup by primary key
    - Just this – hashtable O(c)
  - Ability to add/remove records on the fly
    - Some kind of dynamic tree **on disk**
  - Sequential access to records (physically sorted by primary key on disk)
    - Tree structured keys (hierarchical index for searching)
    - Records all at leaves in sorted order

What is a B+ Tree?

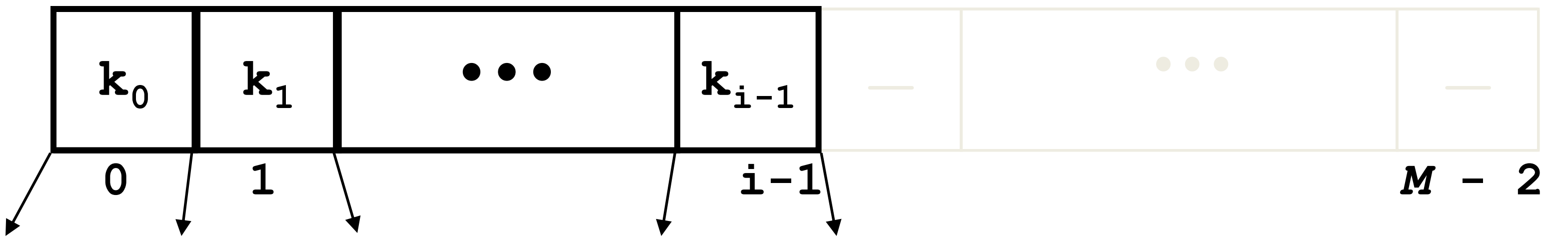
- A variation of B trees in which
  - internal nodes contain only search keys (no data)
  - Leaf nodes contain pointers to data records
  - Data records are in sorted order by the search key
  - All leaves are at the same depth

Definition of a B+Tree

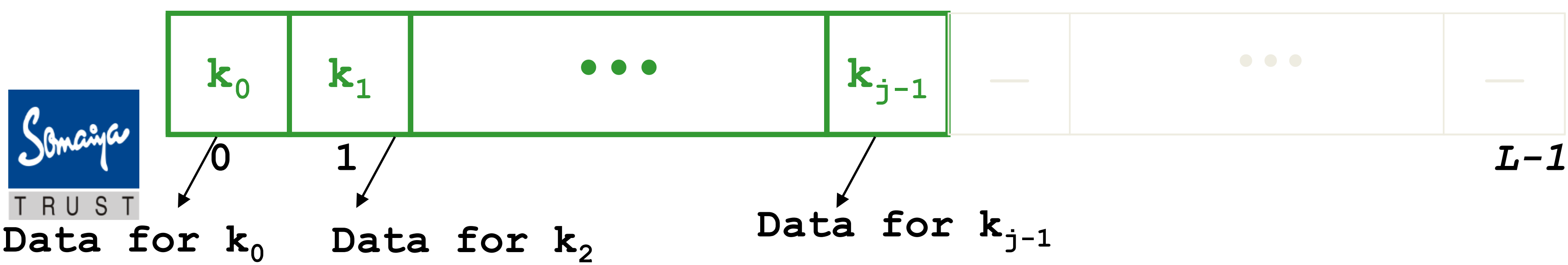
A B+ tree is a balanced tree in which every path from the root of the tree to a leaf is of the same length, and each non-leaf node of the tree has between [M/2] and [M] children, where n is fixed for a particular tree.

B+ Tree Nodes

- Internal node
  - Pointer (Key, NodePointer)*M-1 in each node
  - First i keys are currently in use



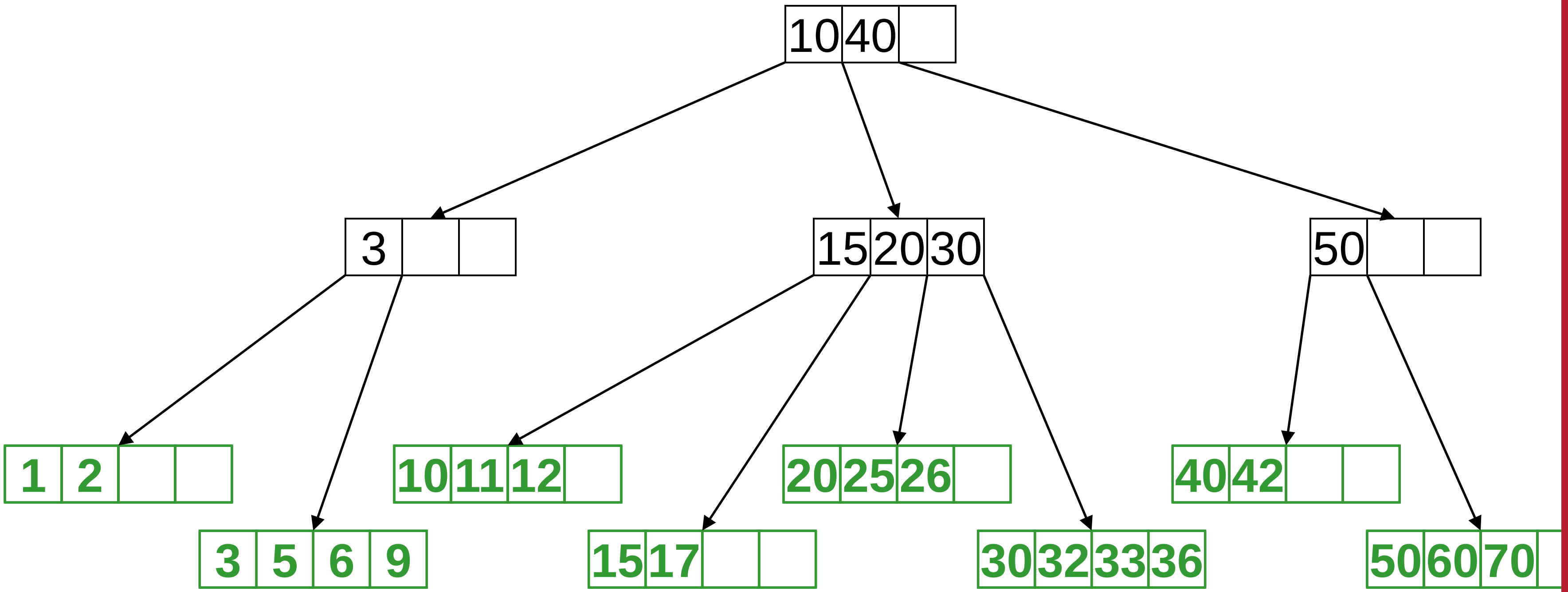
- Leaf
  - `(Key, DataPointer)* L in each node`
  - `first j Keys currently in use`



Example

B+ Tree with ***M* = 4**

`Often, leaf nodes linked together`



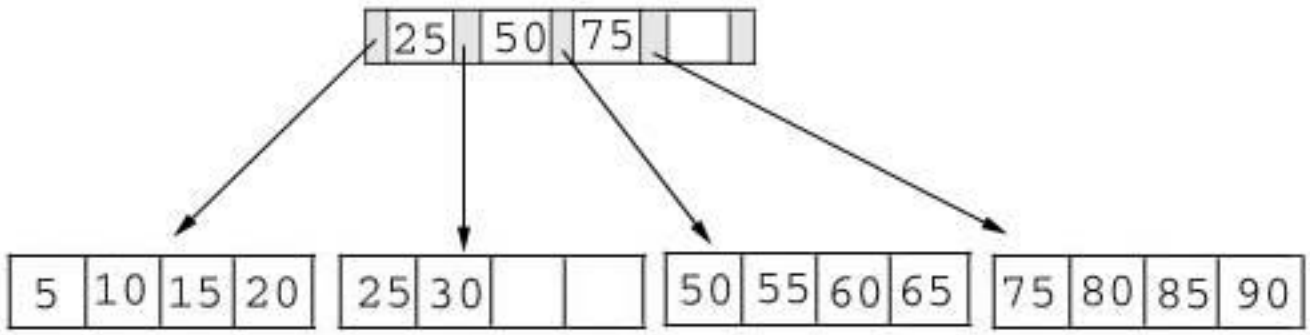
Advantages of B+ tree usage for databases

- keeps keys in sorted order for sequential traversing
- uses a hierarchical index to minimize the number of disk reads
- uses partially full blocks to speed insertions and deletions
- keeps the index balanced with a recursive algorithm
- In addition, a B+ tree minimizes waste by making sure the interior nodes are at least half full. A B+ tree can handle an arbitrary number of insertions and deletions.

Searching

- Just compare the key value with the data in the tree, then return the result.

For example: find the value 45, and 15 in below tree.

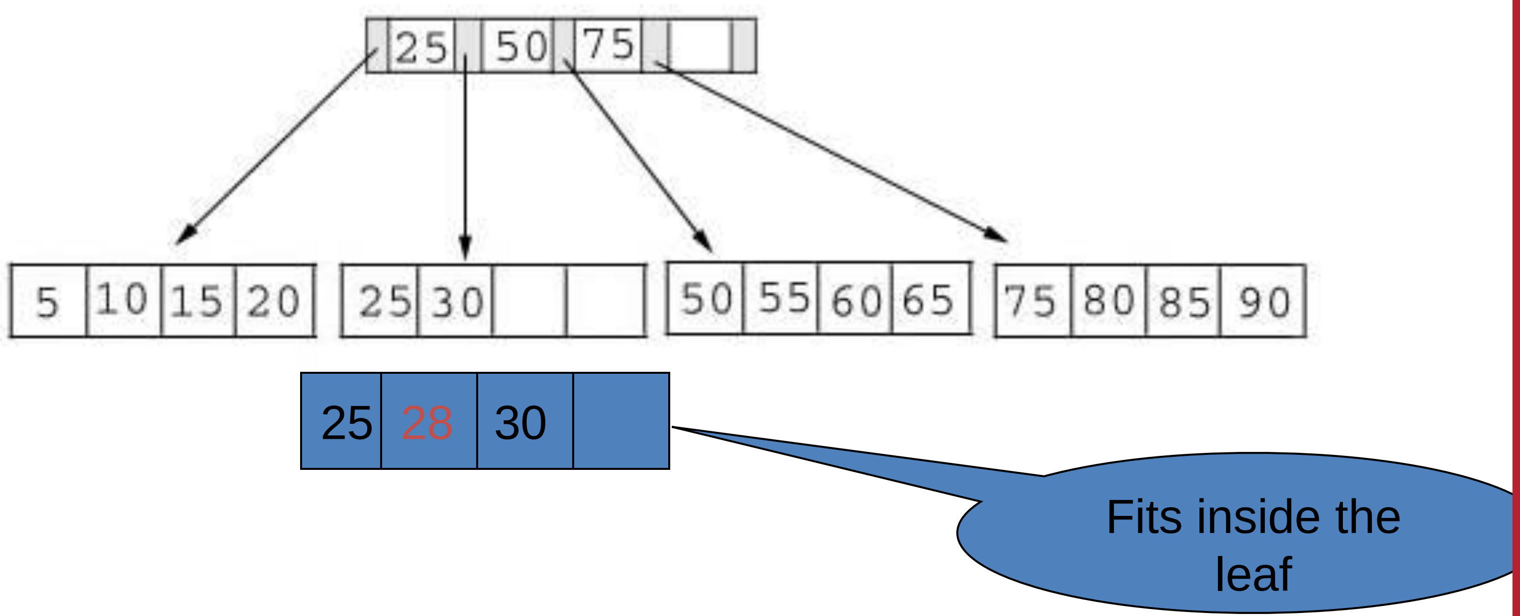


Searching

- Result:
  1. For the value of 45, not found.
  2. For the value of 15, return the position where the pointer located.

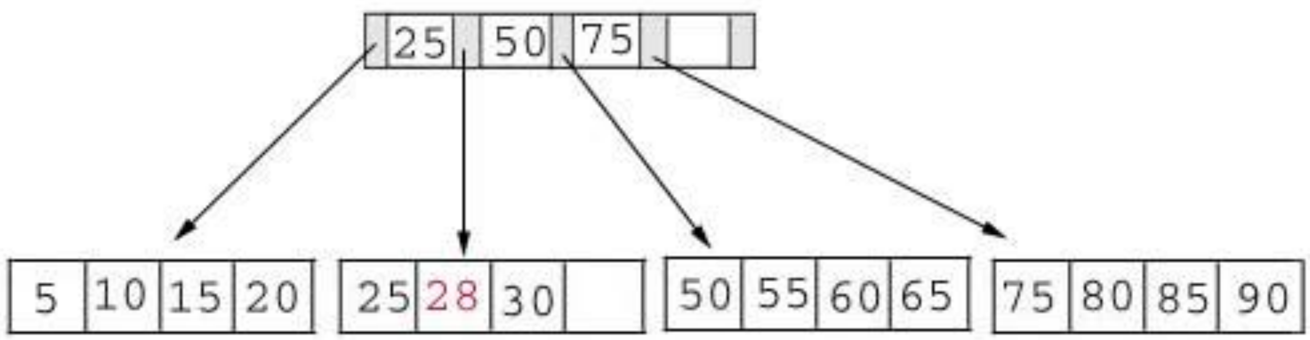
Insertion

- inserting a value into a B+ tree may unbalance the tree, so rearrange the tree if needed.
- Example #1: insert 28 into the below tree.



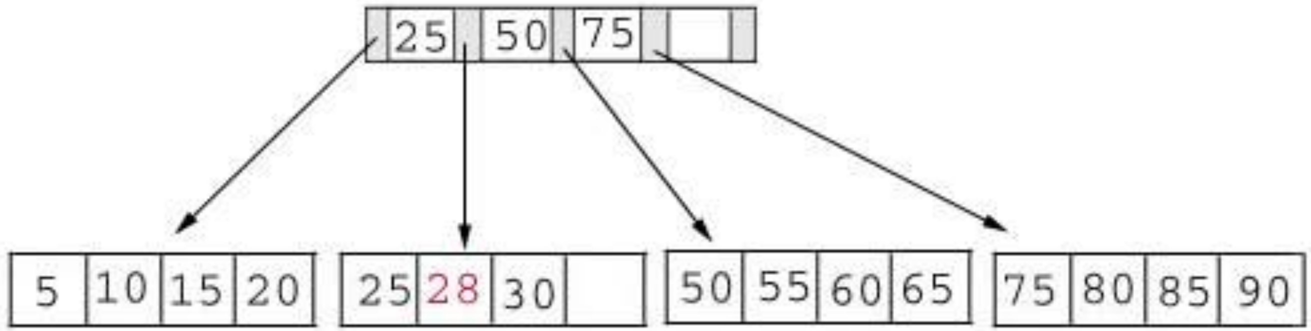
Insertion

- Result:



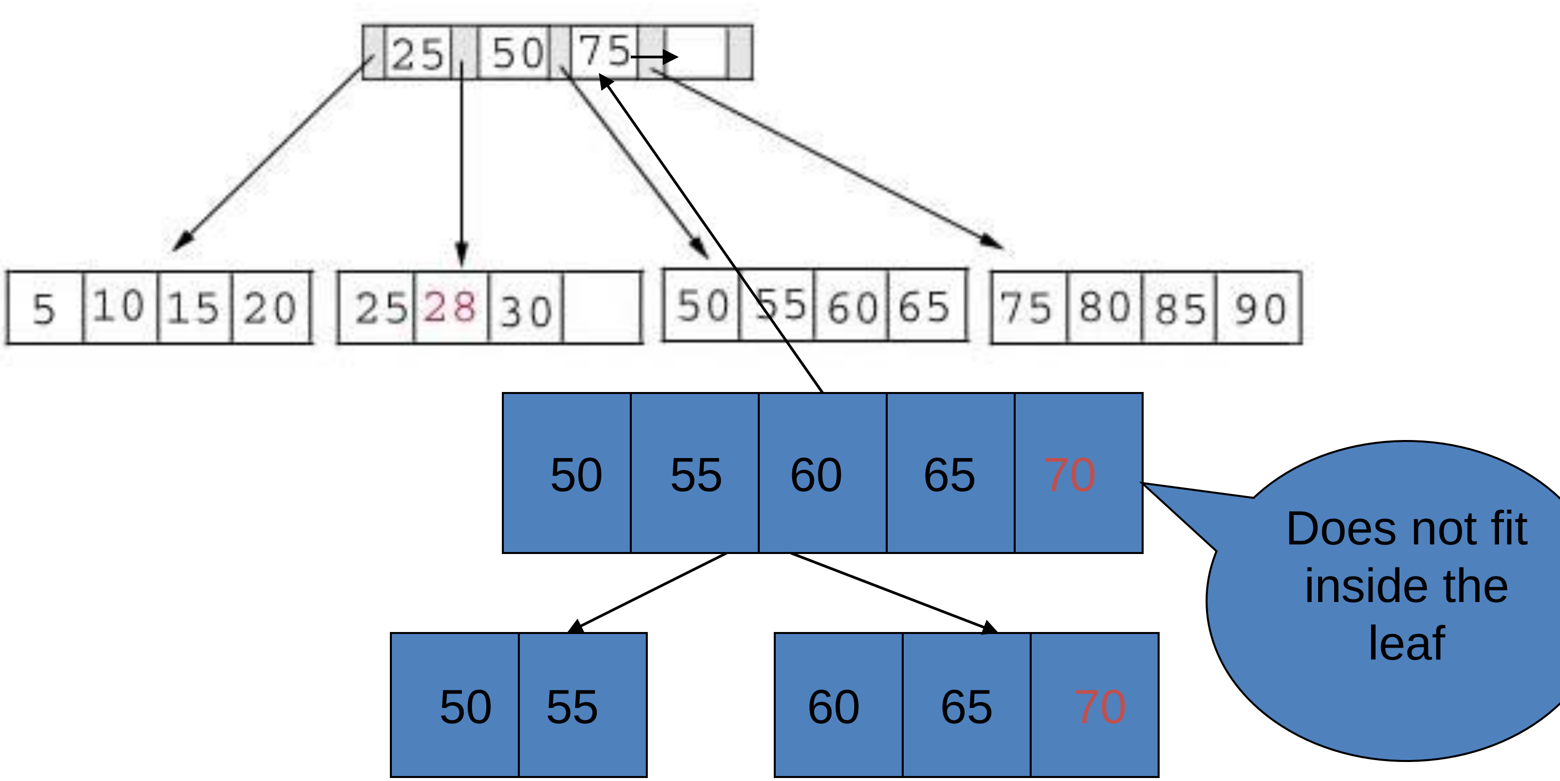
Insertion

- Example #2: insert 70 into below tree



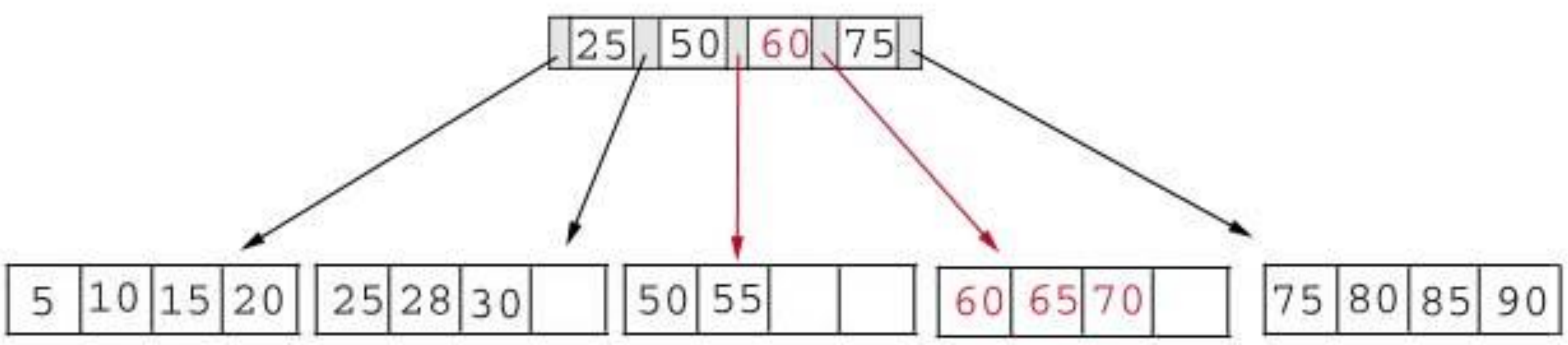
Insertion

- Process: split the leaf and propagate middle key up the tree



Insertion

- Result: chose the middle key 60, and place it in the index page between 50 and 75.



SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Insertion

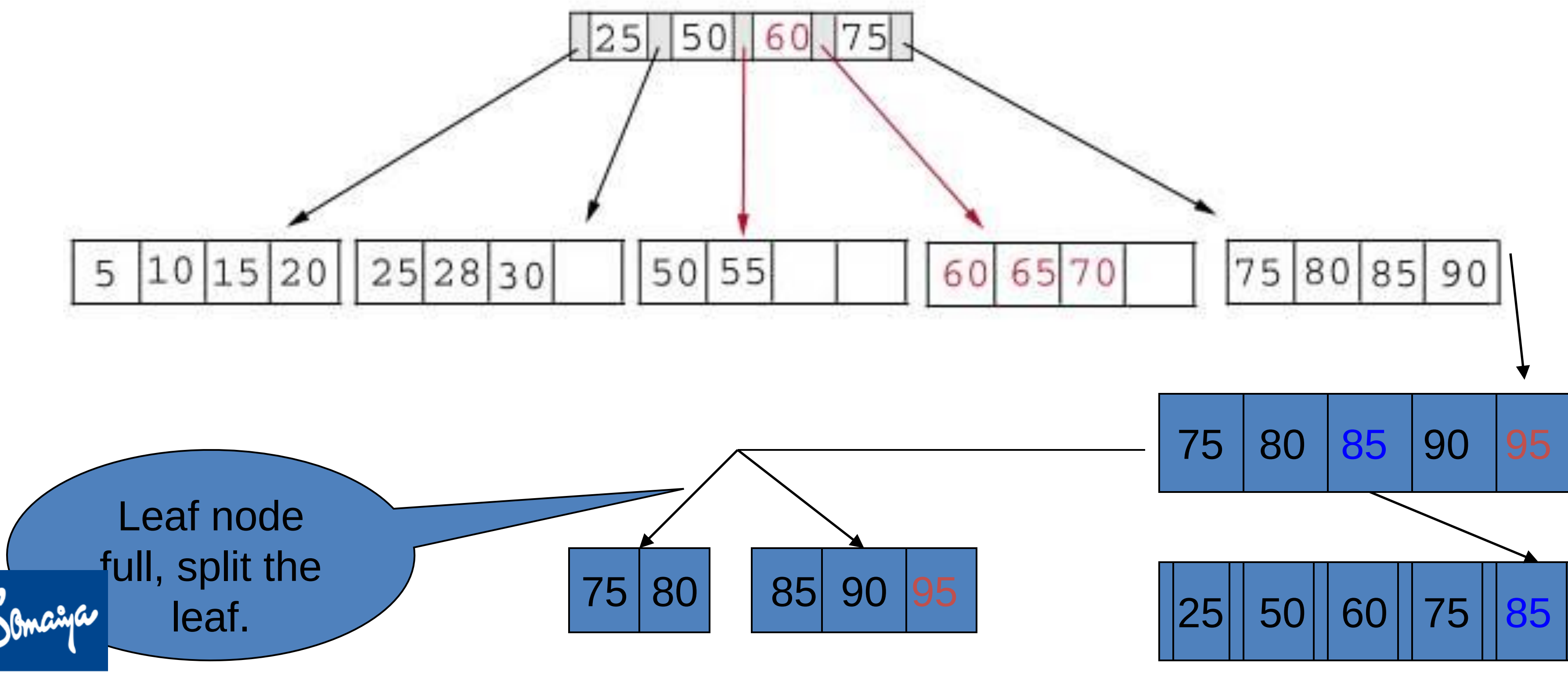
The insert algorithm for B+ Tree

Leaf Node Full	Index Node Full	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf node 2. Place Middle Key in the index node in sorted order. 3. Left leaf node contains records with keys below the middle key. 4. Right leaf node contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf node. 2. Records with keys < middle key go to the left leaf node. 3. Records with keys >= middle key go to the right leaf node. Split the index node. 4. Keys < middle key go to the left index node. 5. Keys > middle key go to the right index node. 6. The middle key goes to the next (higher level) index node. IF the next level index node is full, continue splitting the index nodes.

Somaiya TRUST

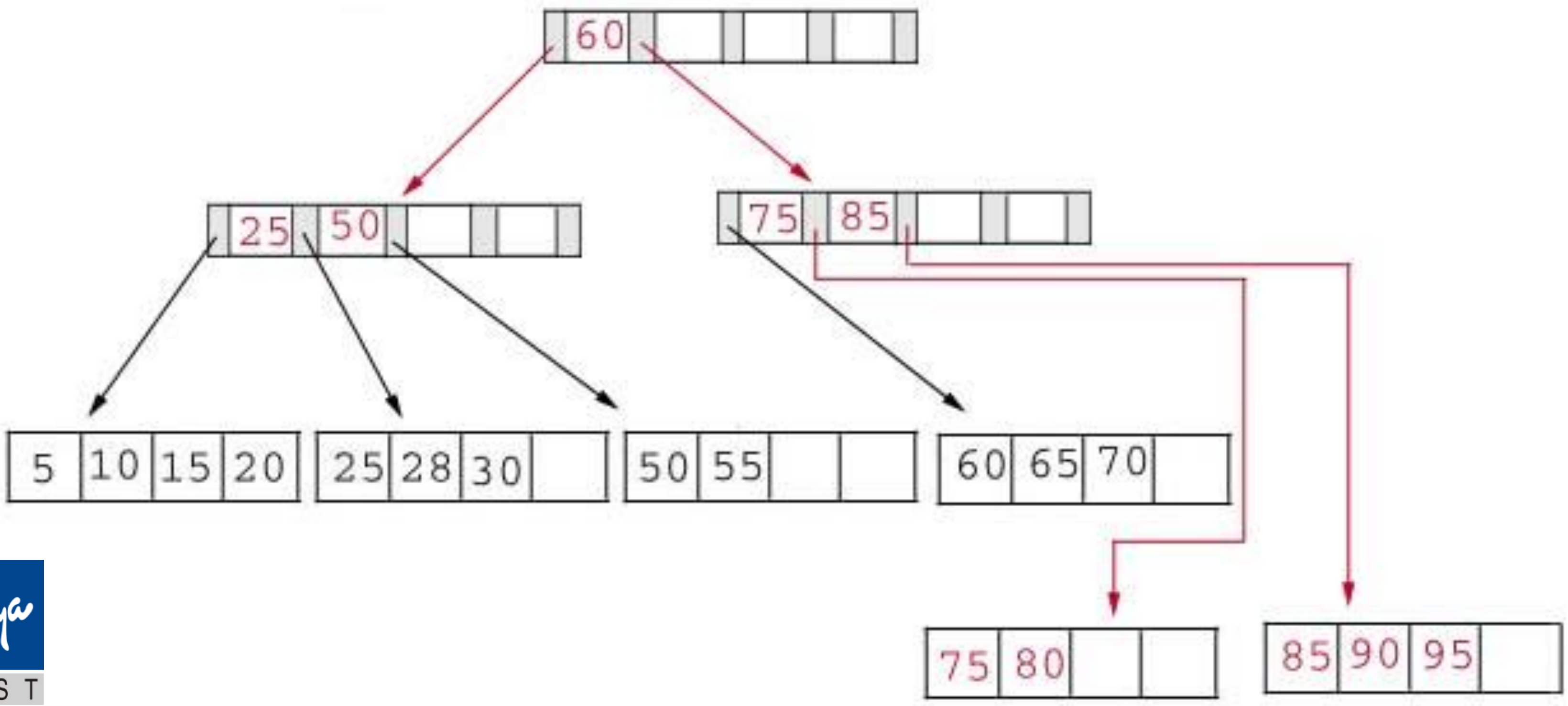
Insertion

- Exercise: add a key value 95 to the below tree.



Insertion

- Result: again put the middle key 60 to the index page and rearrange the tree.



SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Conclusion

- For a B+ Tree:
- It is "easy" to maintain its balance
  - Insert/Deletion complexity $O(\log_{M/2})$
- The searching time is shorter than most of other types of trees because branching factor is high

Somaiya TRUST

B+Trees and DBMS

- Used to index primary keys
- Can access records in $O(\log_{M/2})$ traversals (height of the tree)
- Interior nodes contain Keys only
  - Set node sizes so that the M-1 keys and M pointers fits inside a single block on disk
    - E.g., block size 4096B, keys 10B, pointers 8 bytes
    - $(8+ (10+8)*M-1) = 4096$
    - M = 228; 2.7 billion nodes in 4 levels
  - One block read per node visited

Reference

Li Wen & Sin-Min Lee, San Jose State University

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Thank you!

Somaiya
TRUST