CS350: Data Structures

B-Trees

James Moscola
Department of Physical Sciences
York College of Pennsylvania

Introduction

- **All of the data structures that we've looked at thus far have been memory-based data structures**

- **What if we have more data than we can store in main system memory?**

- **What if we need to store a data structure on disk?**
  - Are the data structures we've talked about thus far suitable for disk?
  - What features of previous data structures might need to be altered?

Memory Access vs. Disk Access

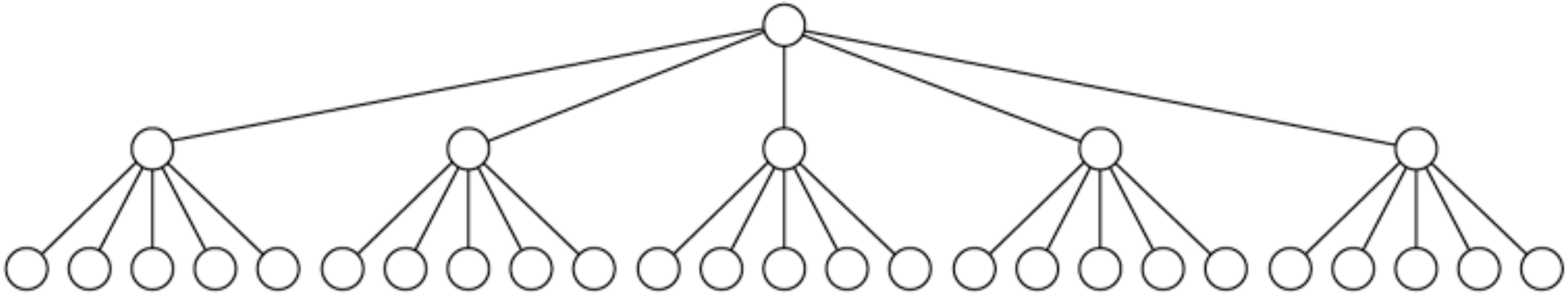
- **Following pointers in memory is a relatively quick operation when compared to following pointers on disk**
- **Hard disk drives are very slow**
  - A 7200 RPM disk drive rotates 7200 times in one minute (each revolution takes 1/120 of a second ... ~8.3 ms)
  - On average, must spin the disk half way to get to the desired data
  - For this example, can do about 120 disk accesses per second
  - A 3 GHz processor can compute the result of ~3 Billion instructions per second (6 Billion if dual core)
  - A disk access is about 25 Million times SLOWER than a modern processor (i.e. the processor can execute 25 Million instructions in the time that it takes one disk access to finish)

Memory Access vs. Disk Access

- **Storing large amounts of data in a binary tree on disk could take very long to access**
  - Given a perfectly balanced binary tree, the number of disk accesses to search for an element on disk is O(log N)
    - May take on the order of several seconds
  - Given an unbalanced binary tree (worst case) the number of disk accesses to search for an element on disk is O(N)
    - Example: A tree with 10 Million nodes stored on disk
      - May require 10 Million disk accesses to find last node. At 8.3ms/disk access, that's 83 million ms (~23 hours)

B-Tree Idea

- **Want to reduce the number of disk accesses to a small number (i.e. 3 or 4)**
    - Increase the fanout of a tree (i.e. increase the # of children at each node)
- **Reduce the height of the tree**
    - Reduces the number of disk accesses required to traverse tree
- **Requires extra computation, but processor time is very cheap compared to disk access time**



B-Trees

- **The number of children that a node may have is M (an M-ary tree)**
- **Each node requires M-1 keys to determine which branch to take when traversing the tree**
  - In a binary tree M=2, and therefore it requires only 1 key at each node to determine which child pointer to follow

B-Tree Properties

- **A B-tree of order M is an M-ary tree with the following properties:**
  (1) Data items are stored at leaf nodes

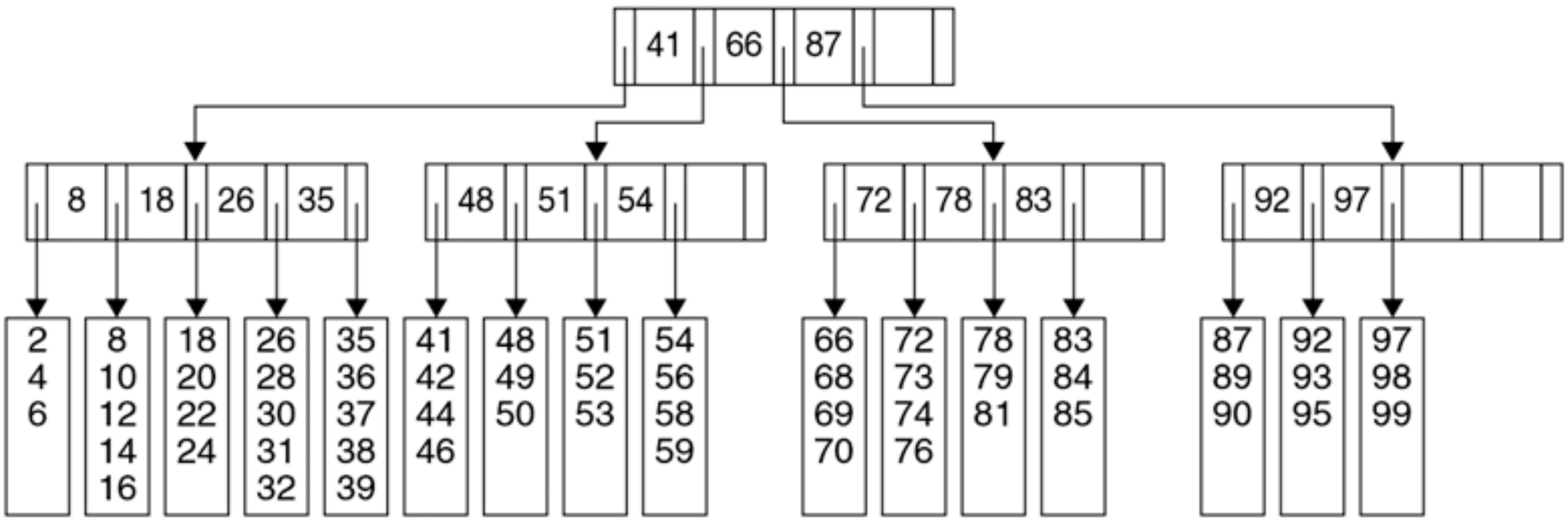
  (2) Non-leaf nodes store as many as $M-1$ keys to guide the searching process; key i represents the smallest key in subtree $i+1$

  (3) The root is either a leaf or has between 2 and M children

  (4) All non-leaf nodes (except the root) have between $\lceil M/2 \rceil$ and M children

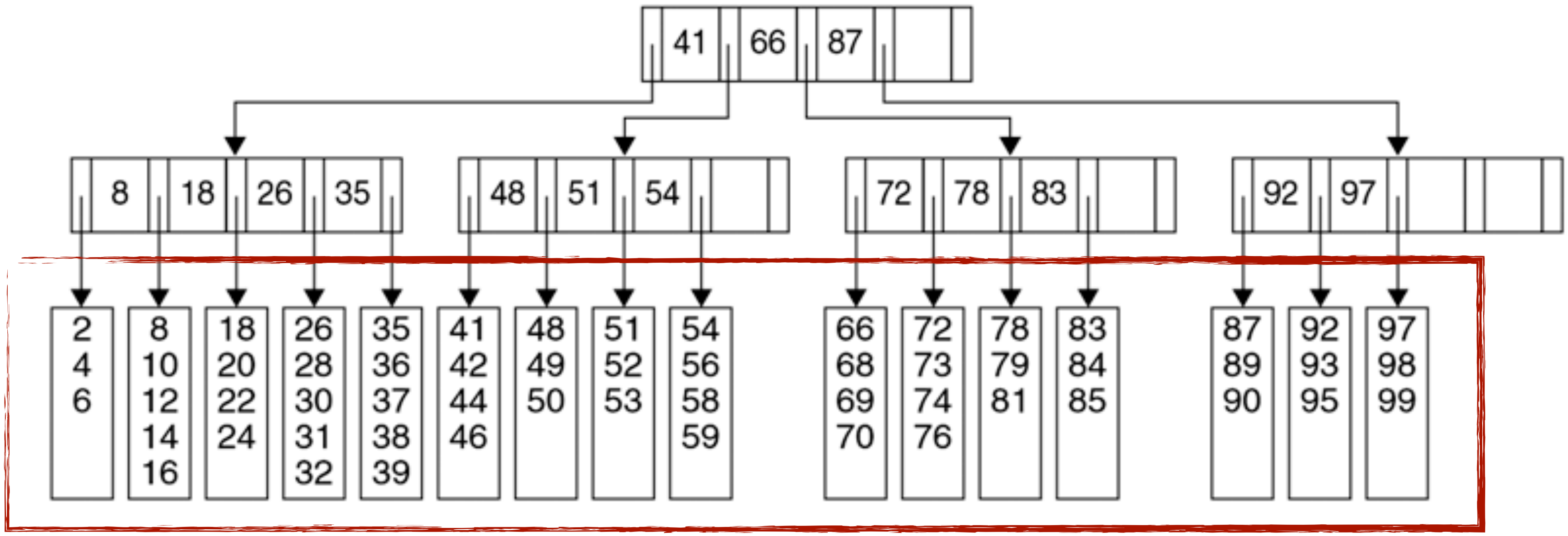
  (5) All leaf nodes are at the same depth and have between $\lceil L/2 \rceil$ and L data items, for some value L

B-Tree Example



B-Tree Properties

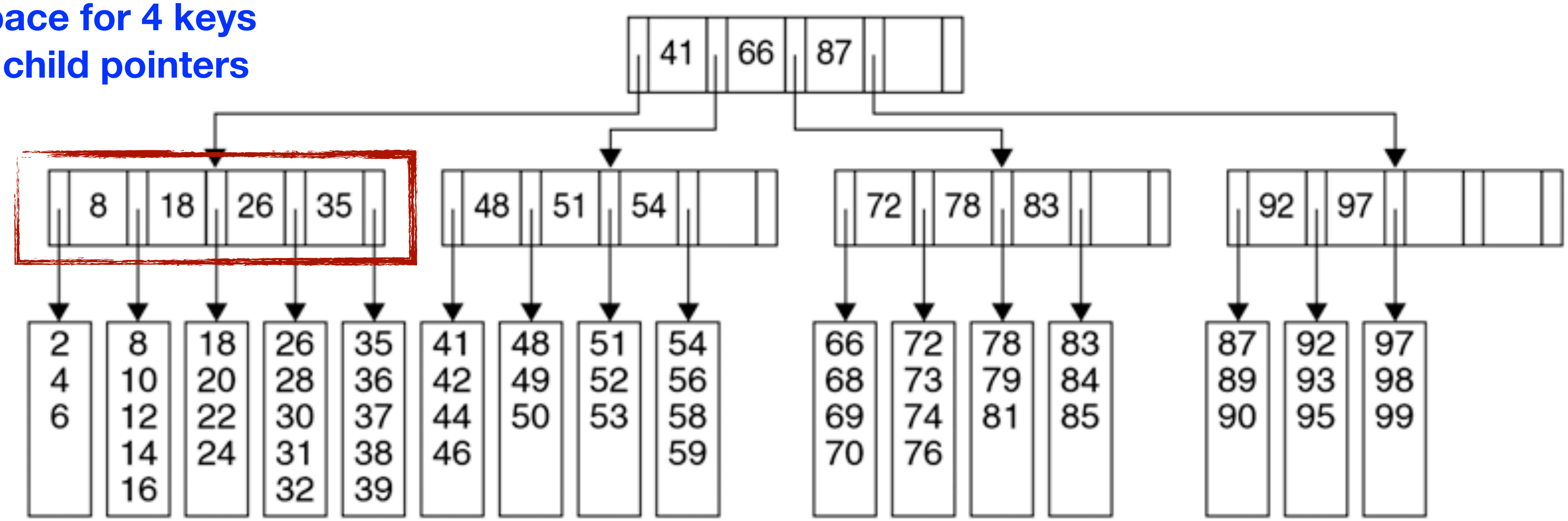
(1) All data items are stored in the leaf nodes



B-Tree Properties

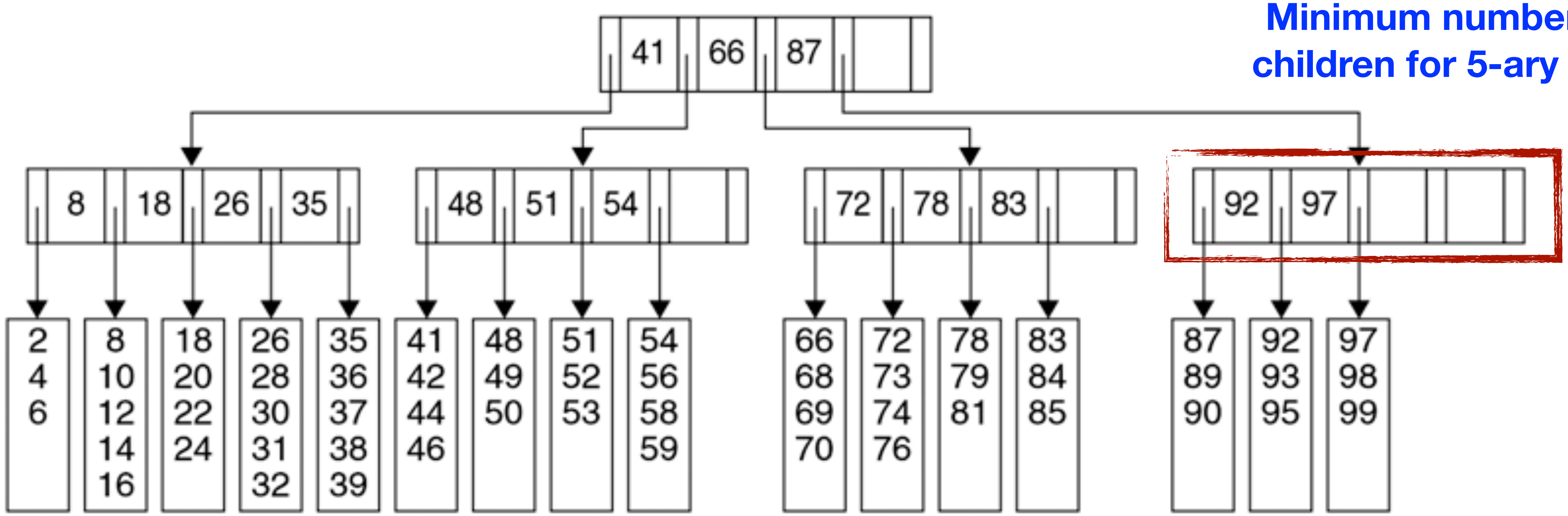
(2) Non-leaf nodes store as many as *M*-1 keys to guide the searching process; key *i* represents the smallest key in subtree *i*+1

5-ary tree, each node has space for 4 keys and 5 child pointers



B-Tree Properties

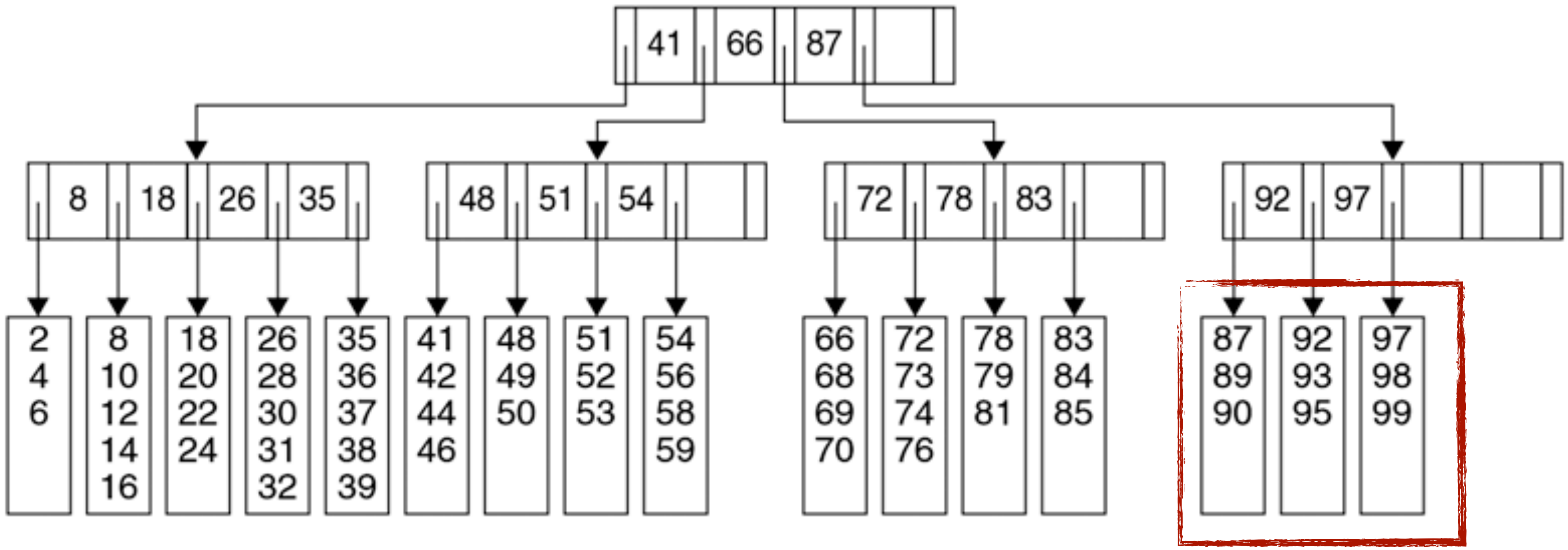
(4) All non-leaf nodes (except the root) have between ⌈*M*/2⌉ and *M* children



Minimum number of children for 5-ary tree

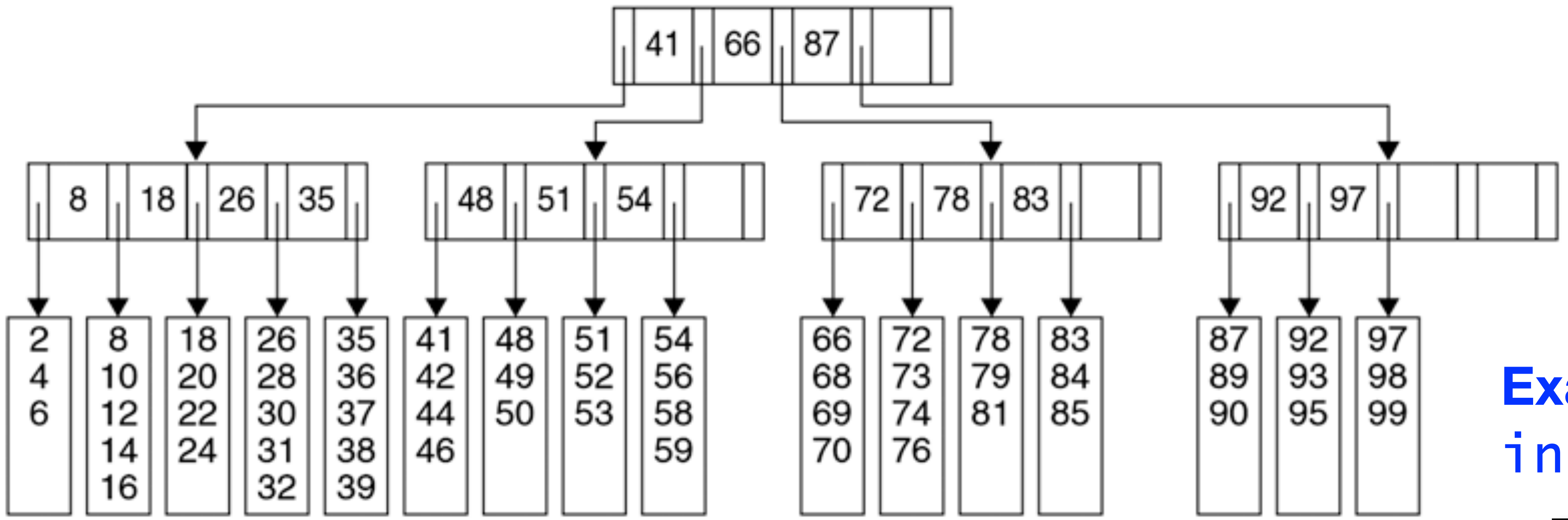
B-Tree Properties

(5) All leaf nodes are at the same depth and have between ⌈*L*/2⌉ and *L* data items, for some value *L*

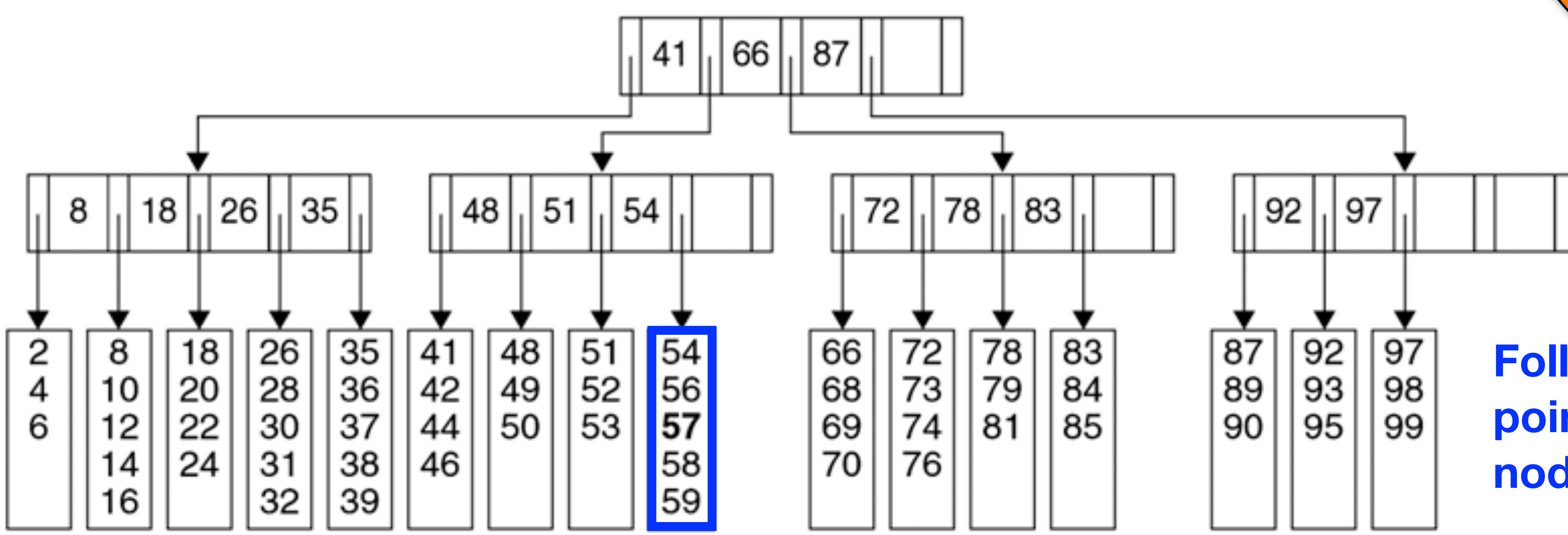


In this example, L=5 so each leaf node must have between 3 and 5 data items

B-Tree Insertion



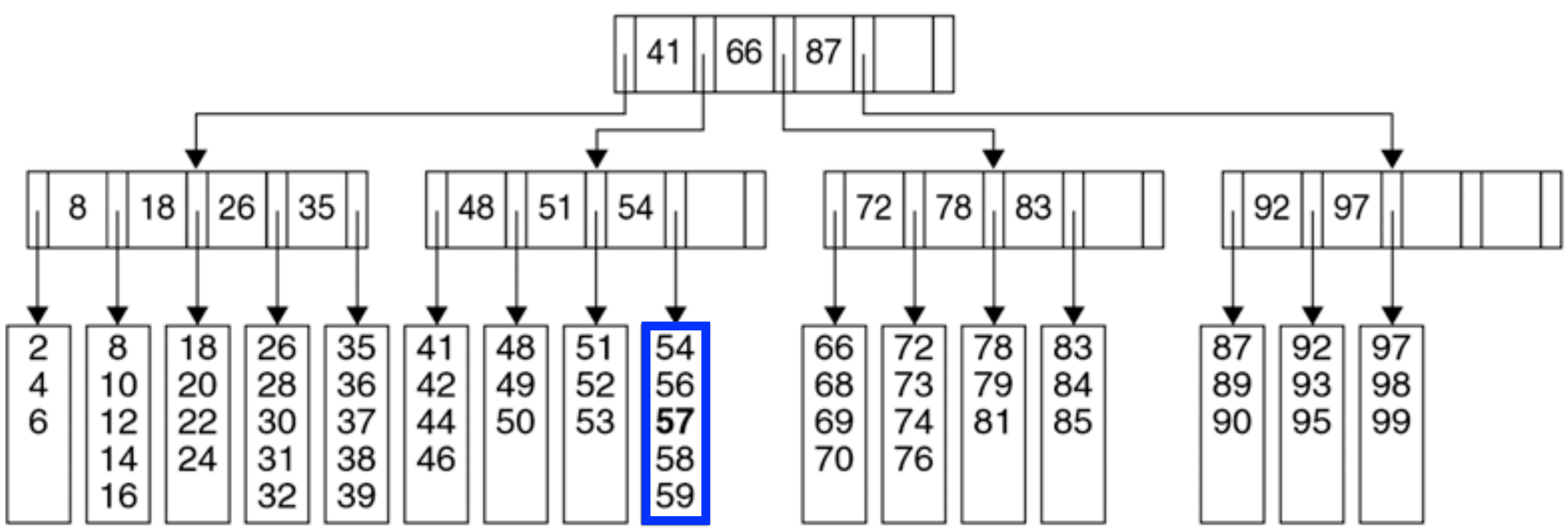
Example of insertion:
`insert(57)`



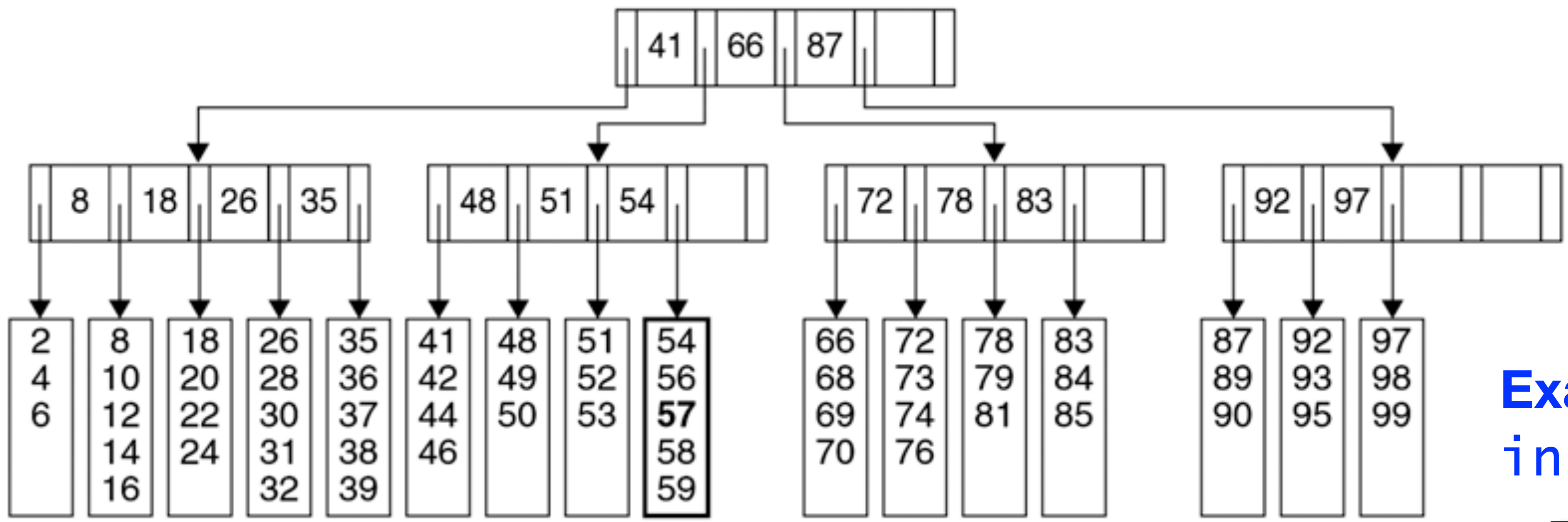
Follow tree to insertion point and insert into leaf node if space is available

B-Tree Insertion

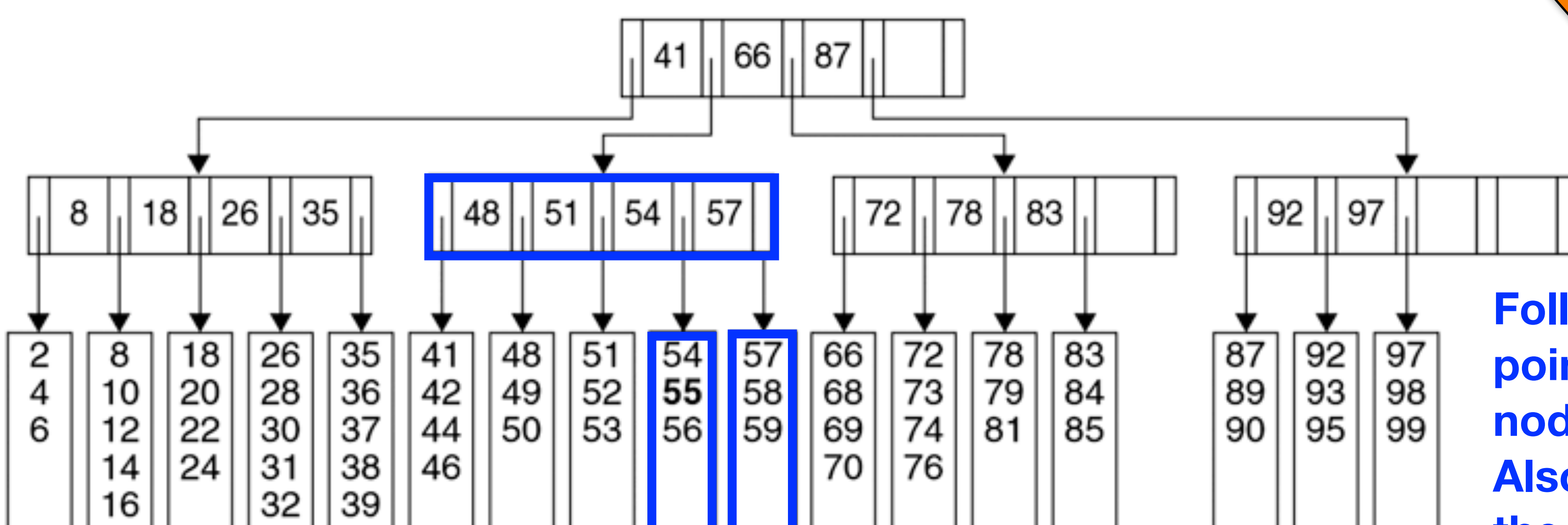
- **What if there is no space available in leaf node during insertion?**
  - The leaf node is split into two separate leaf nodes and the existing keys are redistributed between the new leaf nodes
- **Consider inserting the value 55 into the following tree**
  - The leaf node where 55 should be inserted is full



B-Tree Insertion (With Full Leaf Node)

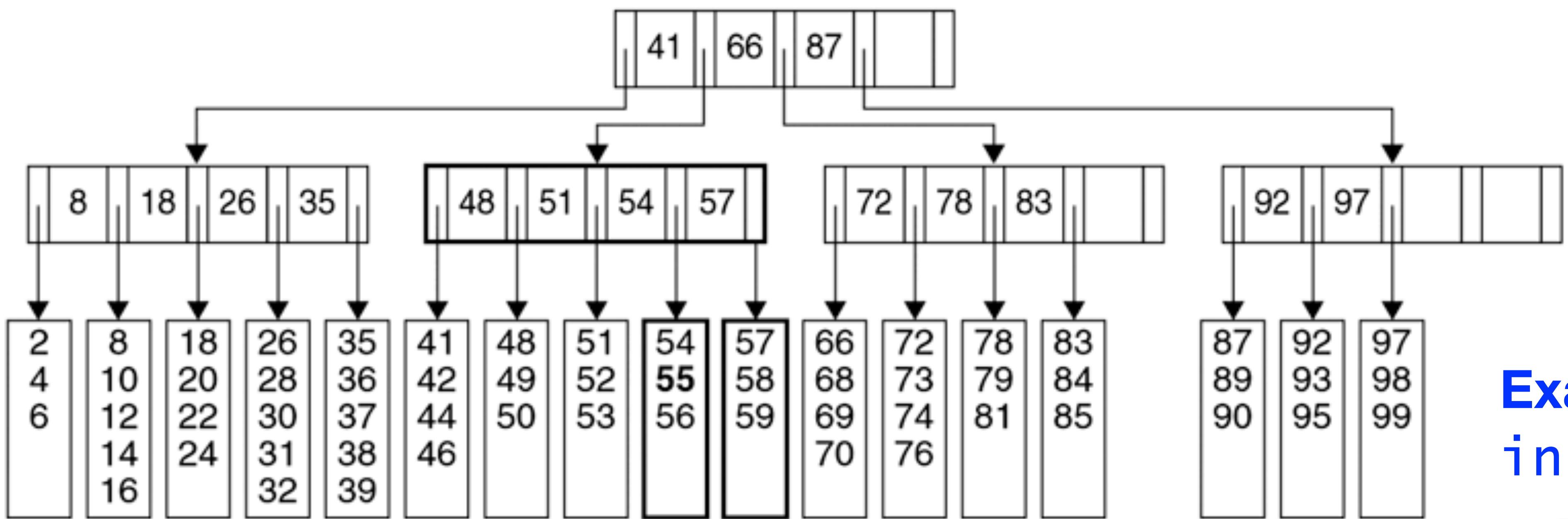


Example of insertion:
`insert(55)`



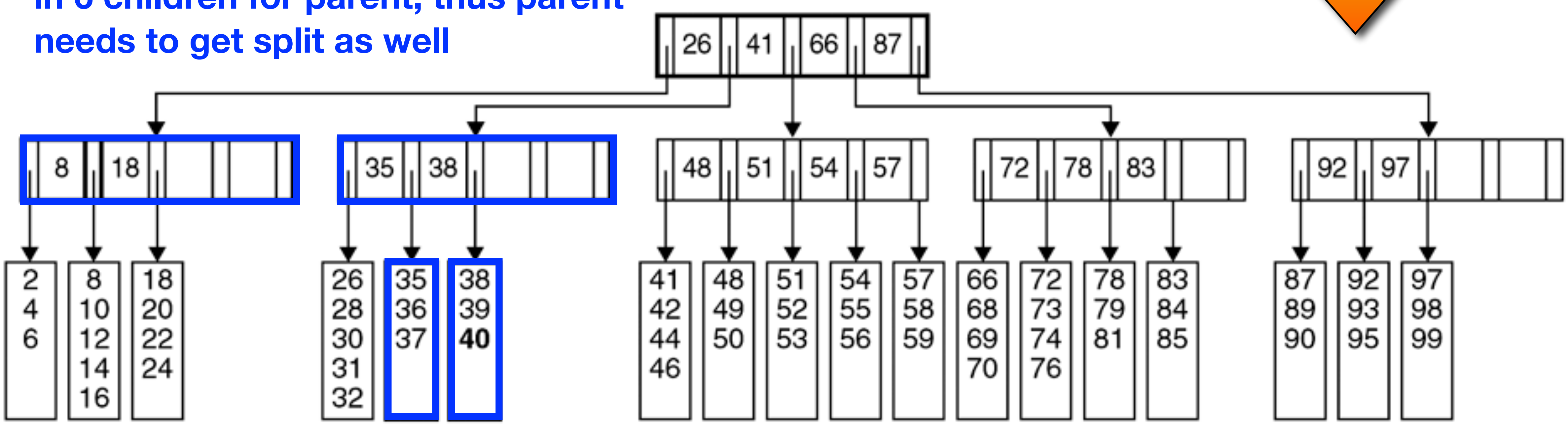
Follow tree to insertion point and split the full leaf node into two nodes. Also, update the keys in the parent node

B-Tree Insertion (With Full Leaf and Parent Nodes)



Example of insertion:
`insert(40)`

Node will need to get split resulting in 6 children for parent, thus parent needs to get split as well



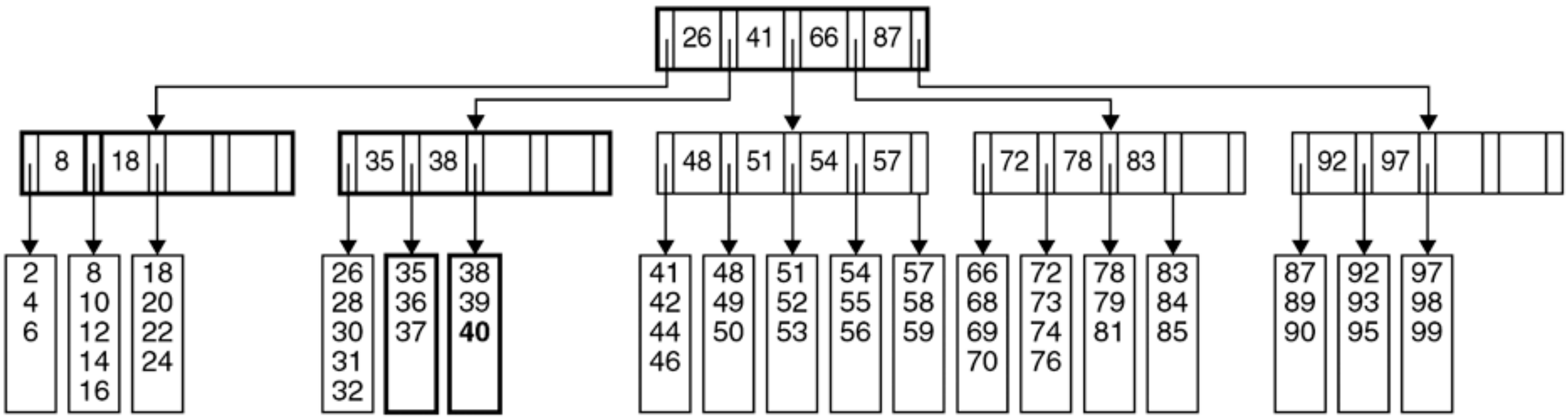
B-Tree Insertion

- **When a node is split, its parent gains a child**
- **If the parent has already reached its limit of children, then the parent must be split as well**
  - Must continue splitting all the way up the tree until a parent node is found that does not need splitting
  - If the root is split, then a new root must be created with the older, split root as its children (root is allowed to have only 2 children)

B-Tree Deletion

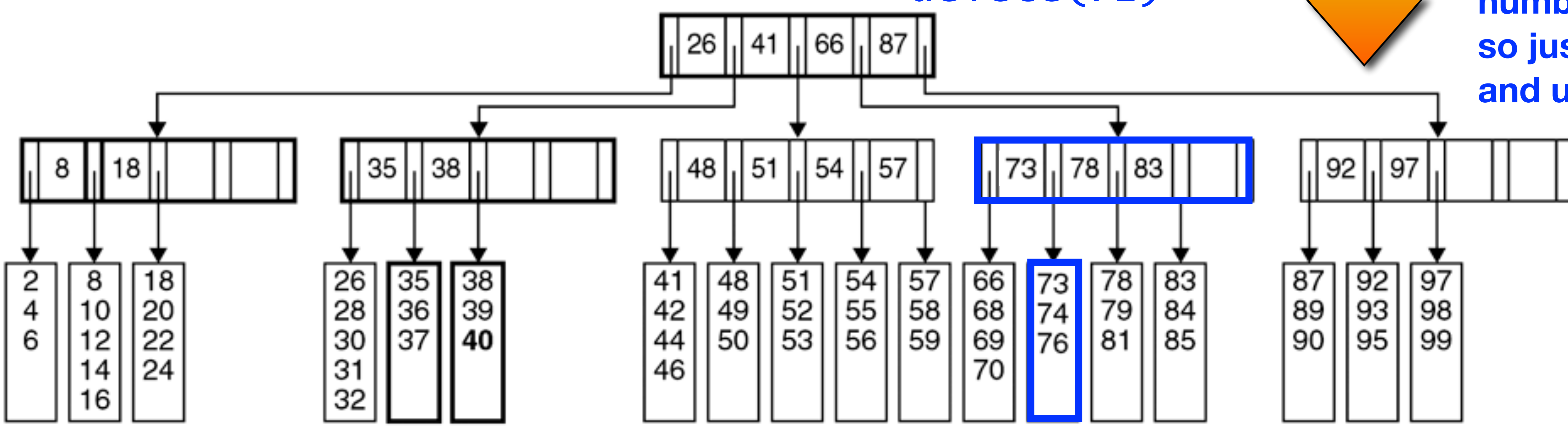
- **To delete from a B-Tree, find the item that needs to be removed and remove it**
  - May cause violation of B-Tree rules if the leaf that the item is deleted from contains only the minimum number of items
    - Remember, each node must contain between $\lceil L/2 \rceil$ and L data items
  - If a violation occurs, a node can gain the minimum number of items by adopting an item from a neighboring leaf node
    - If the neighbor is ALSO already at its minimum number of items, then the two leaf nodes can be combined to form a single full leaf node.
      - This may cause the parent to fall below the minimum number of children, in which case, the same adoption/combining strategy is used to satisfy the B-Tree properties

B-Tree Deletion

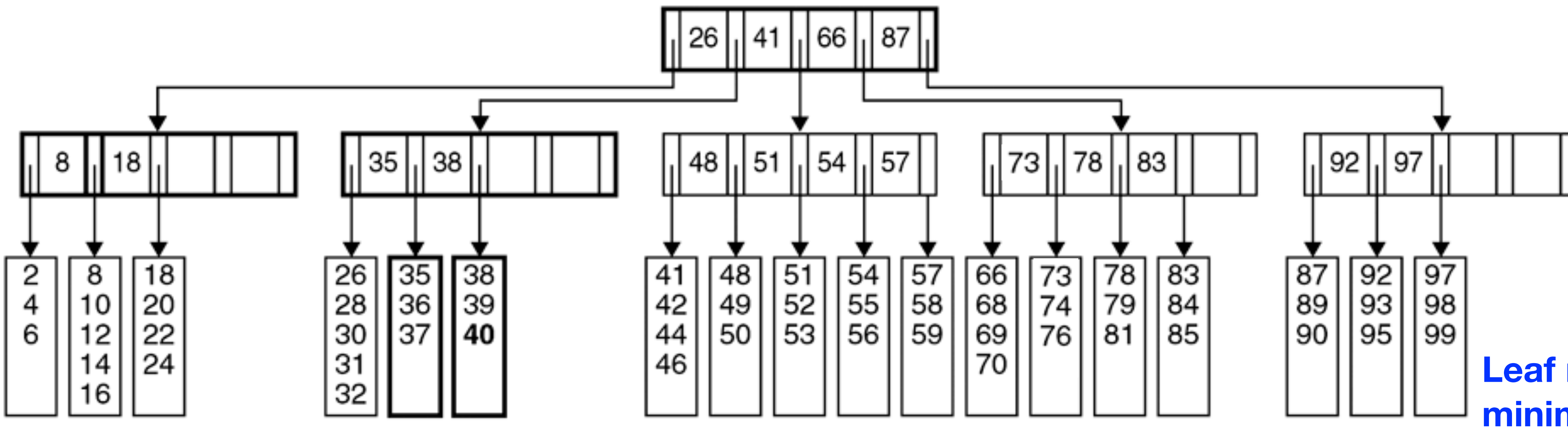


Example of deletion:
`delete(72)`

Leaf node is not at a minimum number of items, so just delete 72 and update keys.

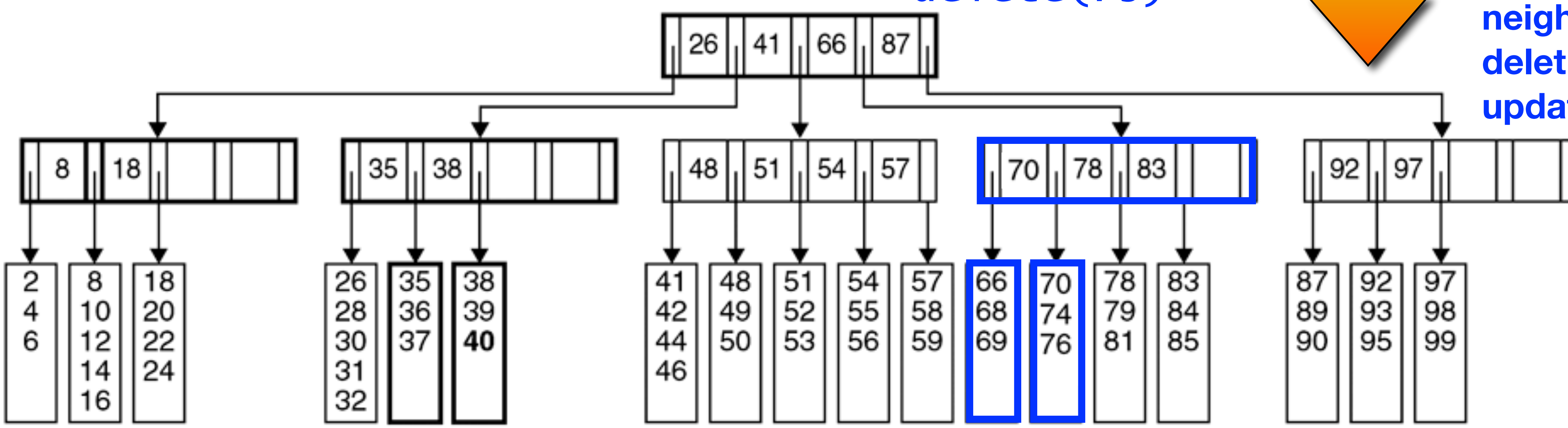


B-Tree Deletion

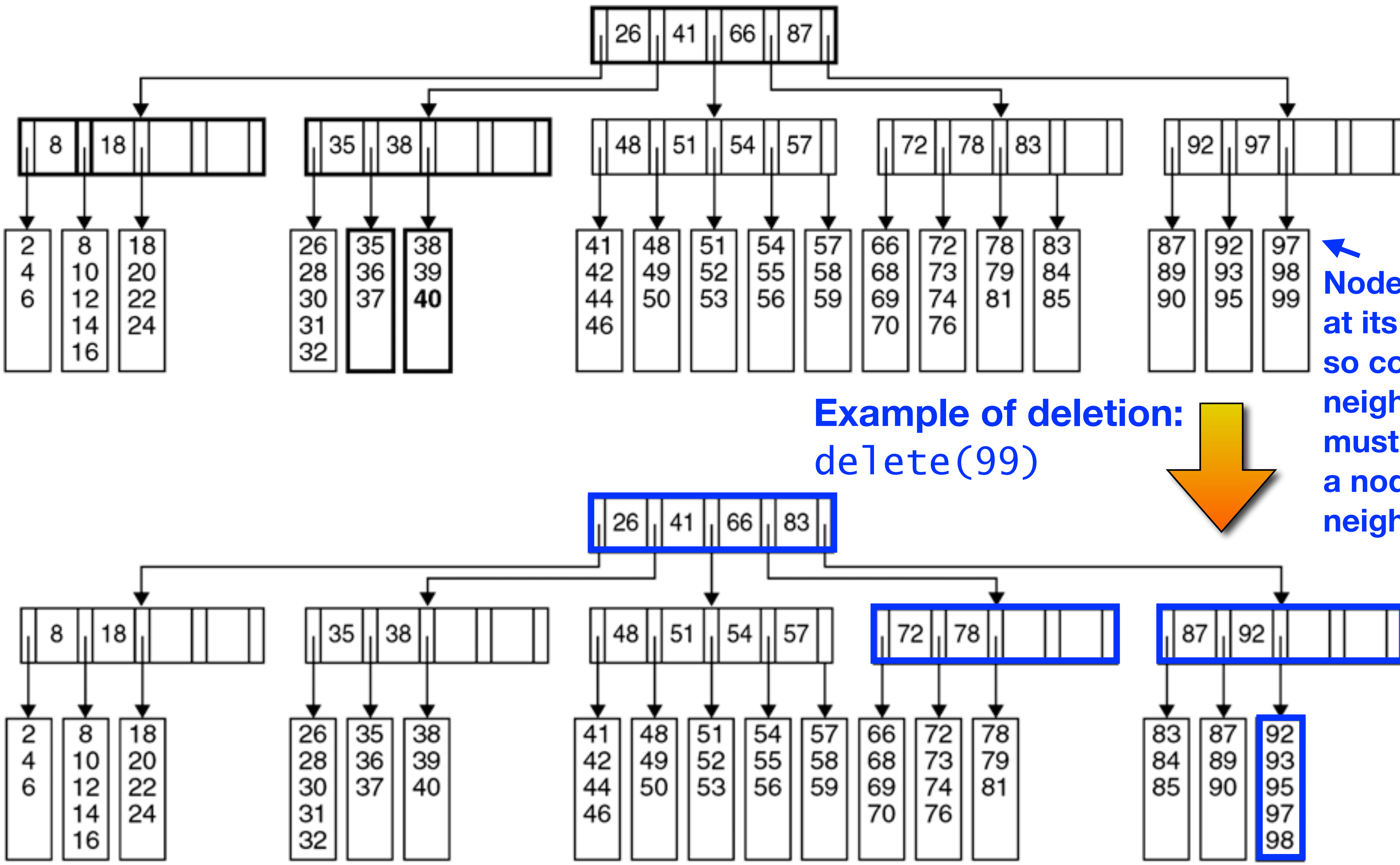


Example of deletion:
`delete(73)`

Leaf node is at a minimum, so need to adopt an item from a neighbor before deleting 73, then update keys.



B-Tree Deletion



Example of deletion:
`delete(99)`

Node is already at its minimum, so combine with neighbor. Parent must then adopt a node from its neighbor.