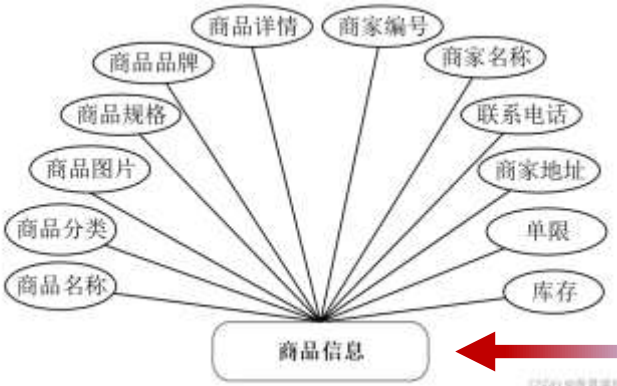
Data Structures and Algorithms

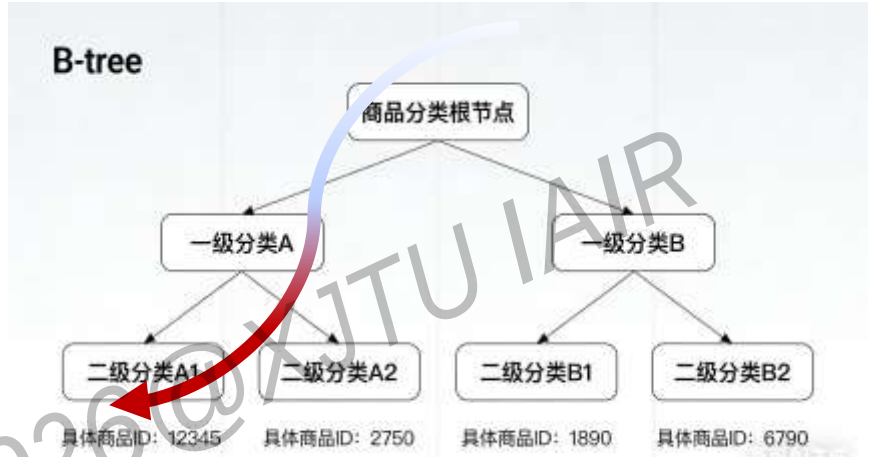
Lecture 07 – Mass Products Database Management (B-Tree, 2/3 Tree, Red-Black and B+ Tree)

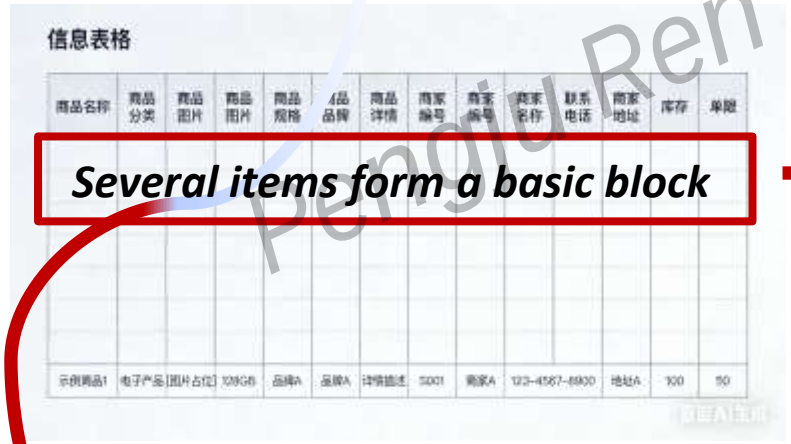
Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

Problem Solving：JD products database management







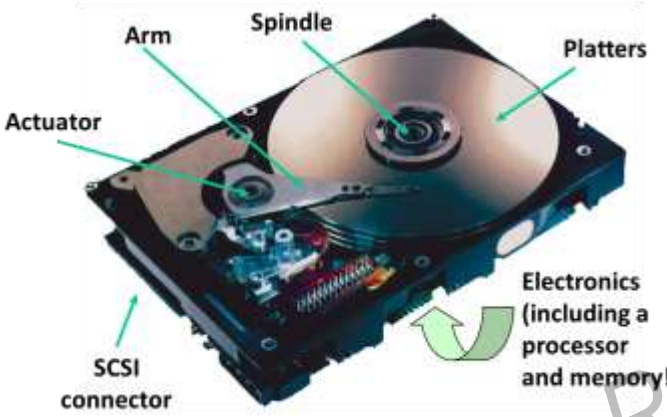
Size of each item at least 256Bytes

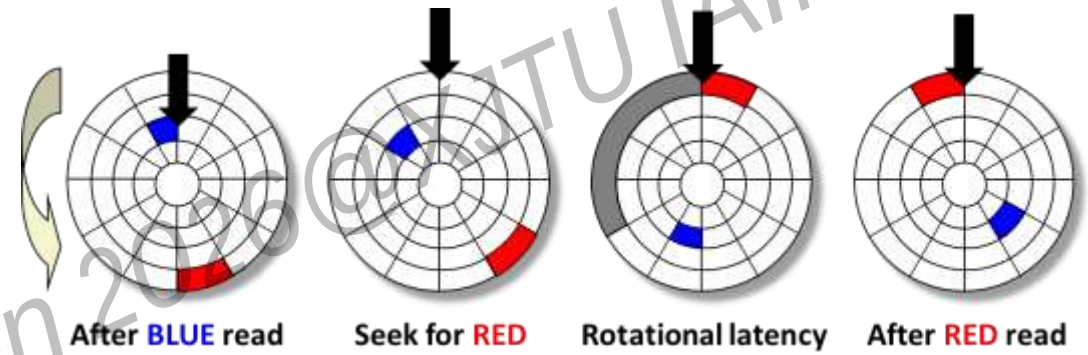
Hard Disk is arranged in Page (4KB), therefore 16 items/page

Background Magnetic Disk

Average time to access some target sector approximated by:

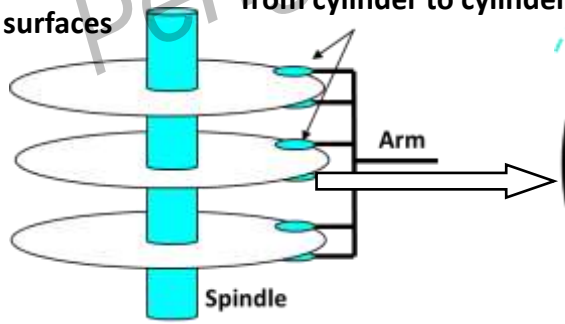
$$T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$$



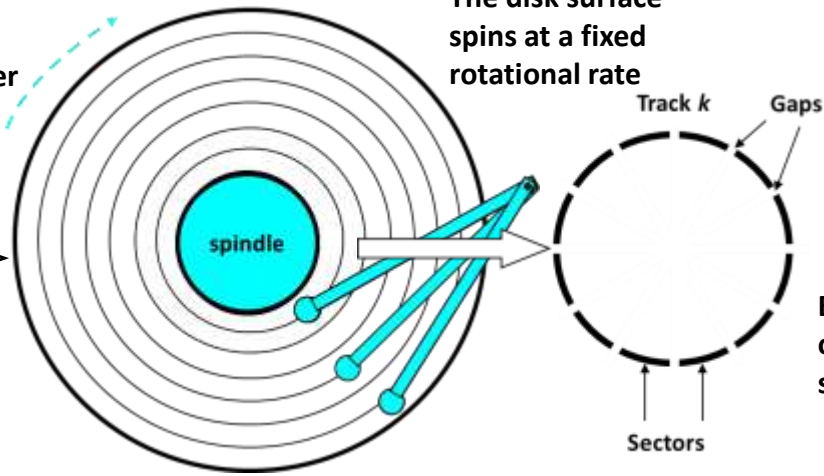


Disks consist of platters, each with two surfaces

Read/write heads move in unison from cylinder to cylinder



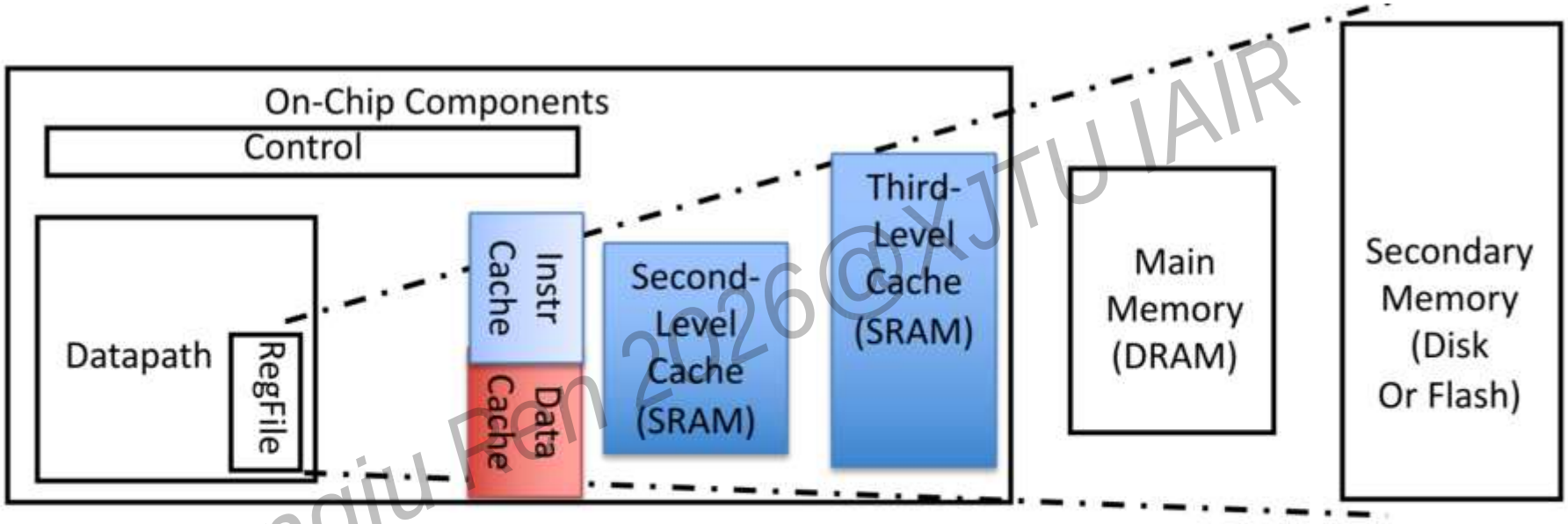
The disk surface spins at a fixed rotational rate



Each surface consists of concentric rings called tracks

Each track consists of sectors separated by gaps

Typical Memory Hierarchy



Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost/bit:	highest	←			lowest

What is a B Tree

A B-Tree is a self-balancing multi-way search tree designed for disk or block storage devices. A B-Tree of order m (m ≥ 2) satisfies the following recursive definition:

- Every internal node except the root

$$B \leq \#\,child < 2B \quad and \quad B\text{-}1 \leq \#\,key < 2B - 1$$

$$\lceil m/2 \rceil \leq \#\,child < m \quad and \quad \lceil m/2 \rceil - 1 \leq \#\,key < m - 1$$

- The root node: if it is not a leaf, it must have at least 2 child nodes (i.e., at least 1 key); if the tree is empty, the root can be a leaf with no keys.
- All leaf nodes are **at the same level (perfectly balanced)**.
- Keys inside a node are **stored in ascending order**, and for an internal node, the key ranges of its child nodes are partitioned by its keys:
  - All keys in the i-th child node lie between the *(i−1)-th* key and the i-th key (with the 0-th key treated as −∞ and the last key as +∞).

What is a 2-3 Tree

A 2-3 tree is a self-balancing multi-way search tree. It satisfies the following properties: *(Actually it is a B-Tree with b=2 or m=3)*

Node types:

- ***2-node*: Contains one key and two children.**
  Keys in the left subtree < key < keys in the right subtree.
- ***3-node*: Contains two keys (denoted α and β, with $\alpha < \beta$) and three children.**
  Keys in the left subtree < α, keys in the middle subtree are between α and β, and keys in the right subtree > β.

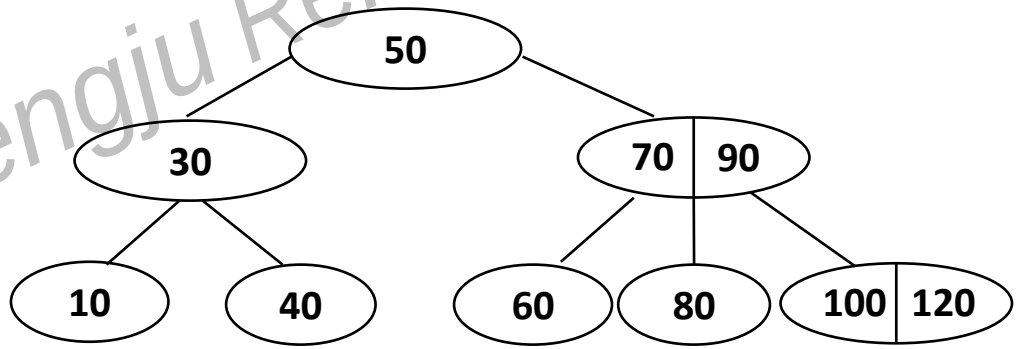
Properties of 2-3 Tree

- **In-order traversal is sorted**
  An in-order traversal of the tree yields the keys in ascending order.
- **All leaves are at the same level**
  The path length from the root to any leaf is identical; hence the tree is ***perfectly balanced***.
- **Key count restriction**
  A node can contain either 1 key (2-node) or 2 keys (3-node). During insertion a node may temporarily hold 3 keys, but it is immediately *split to restore the invariant*.

Basic Operations of 2-3 Tree

■ Search

- Start at the root. At each node, compare the search key with the key(s) in the node to decide which subtree to follow. Repeat until the key is *found* or a leaf is reached (*not found*).
- **Time complexity**: $O(\log n)$ because the height is always between $\log_2 n$ and $\log_3 n$.



Search node 80 and 85

Basic Operations of 2-3 Tree

■ Insertion

- Locate the leaf where the key should be inserted.
- If the leaf is a **2-node**, simply turn it into a **3-node** (add the key).
- If the leaf is already a **3-node**, temporarily turn it into a **4-node** (three keys), then **split** it:
  - Promote the middle key to the parent.
  - Split the node into two 2-nodes.
  - If the parent becomes a 4-node, repeat the split upward. If the root splits, the tree height increases by one.
- **Key point**: Splitting propagates upward, ensuring all leaves stay at the same level.

Build a 2-3 tree by inserting keys in order: 10, 20, 30, 40, 50,25,35

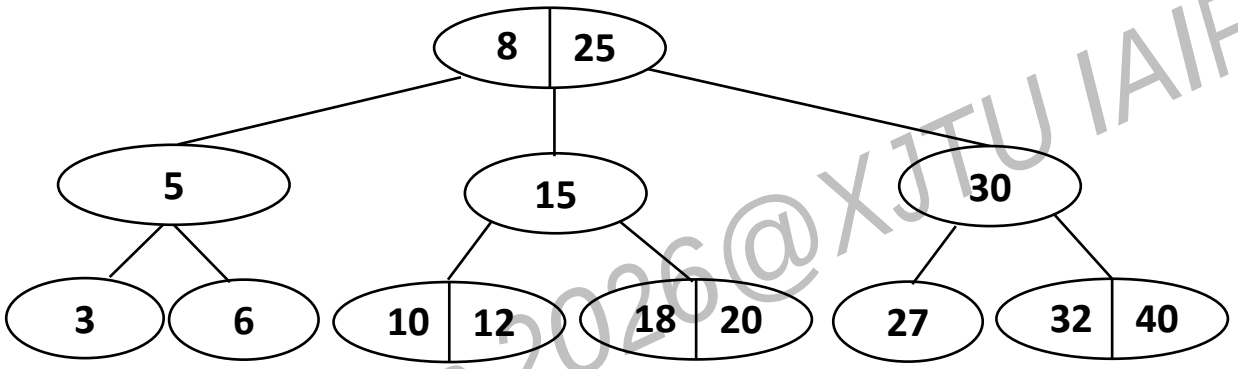
Basic Operations of 2-3 Tree

■ Deletion

- Deletion is more complex, the goal is to restore the 2-3 tree invariants after removal.
- If the key to delete is not in a leaf, replace it with its *in-order predecessor or successor* (which is always in a leaf), then delete that key from the leaf.
- When deleting from a leaf:
  - If the leaf is a **3-node**, simply remove the key → it becomes a 2-node. Done.
  - If the leaf is a **2-node**, after deletion the node becomes empty. Repair by **borrowing** from a sibling (if sibling is a 3-node) or **merging** with a sibling (if sibling is a 2-node), possibly propagating the fix upward.

Reconstruct the 2-3 tree by deleting keys in order: 40, 30, 25, 10

One more example of deletion of 2-3 Tree



Reconstruct the 2-3 tree by deleting keys in order: 30, 32, 18, 20, 15, 25, 40

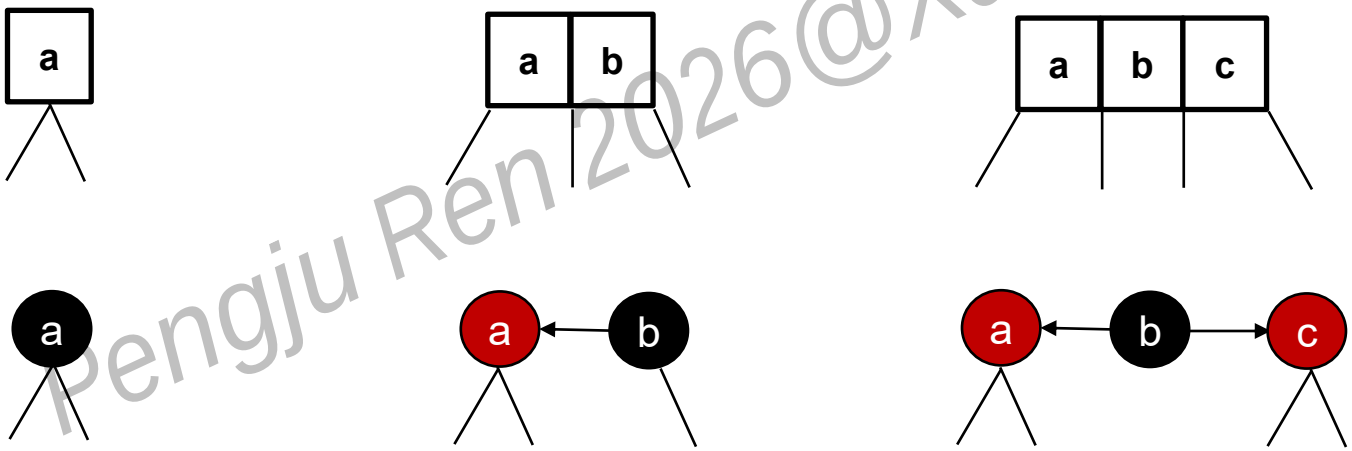
What is a Red-Black Tree

A red-black tree is a binary search tree that satisfies the following five properties:

- Every node is either *red* or *black*.
- The root is *black*.
- Every leaf (NIL null node) is *black*.
- If a node is *red*, then both its children are *black*. (i.e., no two consecutive red nodes on any path.)
- For each node, all simple paths from that node to descendant leaves contain the same number of black nodes. (This number is called the black-height of the node.)

Relationship of 2-3-4 Tree and Red-Black Tree

The **red-black tree** is equivalent to a **2-3-4 tree** (a B-tree of order 4): each black node together with its red children corresponds to one node in the 2-3-4 tree.

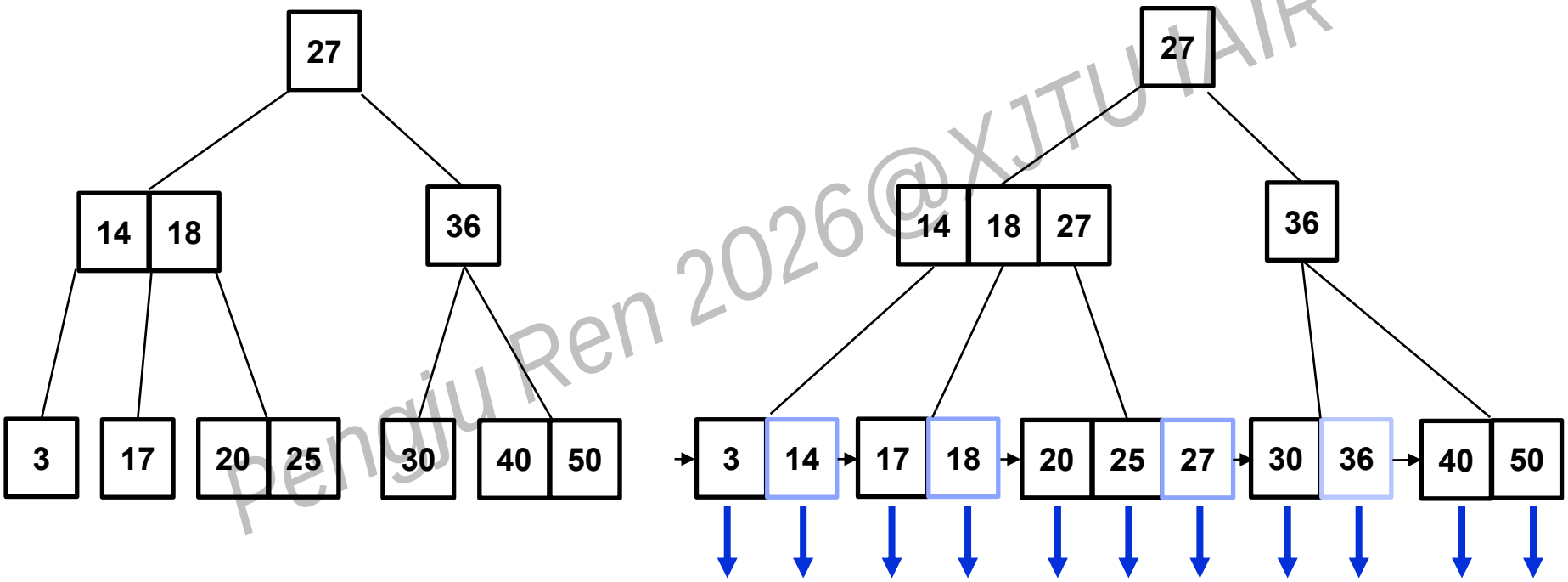


What is a B+ Tree

A B^+-Tree is a variant of the B-Tree. Its main differences lie in where *keys are stored* and *leaf-node linking*.

- **Internal nodes (index nodes):**
  Store only keys (for routing) – no actual data (or data pointers) are kept here.
- **Leaf nodes (data nodes):**
  Store all keys together with their associated actual data (or pointers to the data).
  All leaf nodes are linked in sequential order via a linked list (singly or doubly linked).
- **All leaf nodes are at the same level.**

Diff of B and B+ tree



Summary

- **B-Tree:** A multi-way balanced search tree that compresses tree height using a high branching factor; each node corresponds to one disk block, dramatically reducing I/O operations.
- **B+ Tree:** A B-Tree variant where data is stored only in leaves, which are linked by a list; internal nodes hold only keys (pure indexes), enabling highly efficient range queries – the preferred index structure for databases.
- **2-3 Tree**: A B-Tree of order 3; nodes can be 2-nodes (1 key, 2 children) or 3-nodes (2 keys, 3 children)
- **Red-Black Tree:** A nearly balanced binary search tree that uses red/black color constraints (root black, leaves black, red nodes have black children, equal black-height on all root-to-leaf paths) to guarantee that the longest path is at most twice the shortest path