LECTURE-25

M-way Trees, B+ Trees

For Poll Ev

CS202: Data Structures (Fall 2025)

Dr Maryam Abdulghafur, Momina Khan
Department of Computer Science, SBASSE

Agenda

- Binary Trees to M-ary Trees
- Motivation for M >> 2
- Case for B+ Trees
- An insight into the workings of B+ Trees

Binary Tree recap

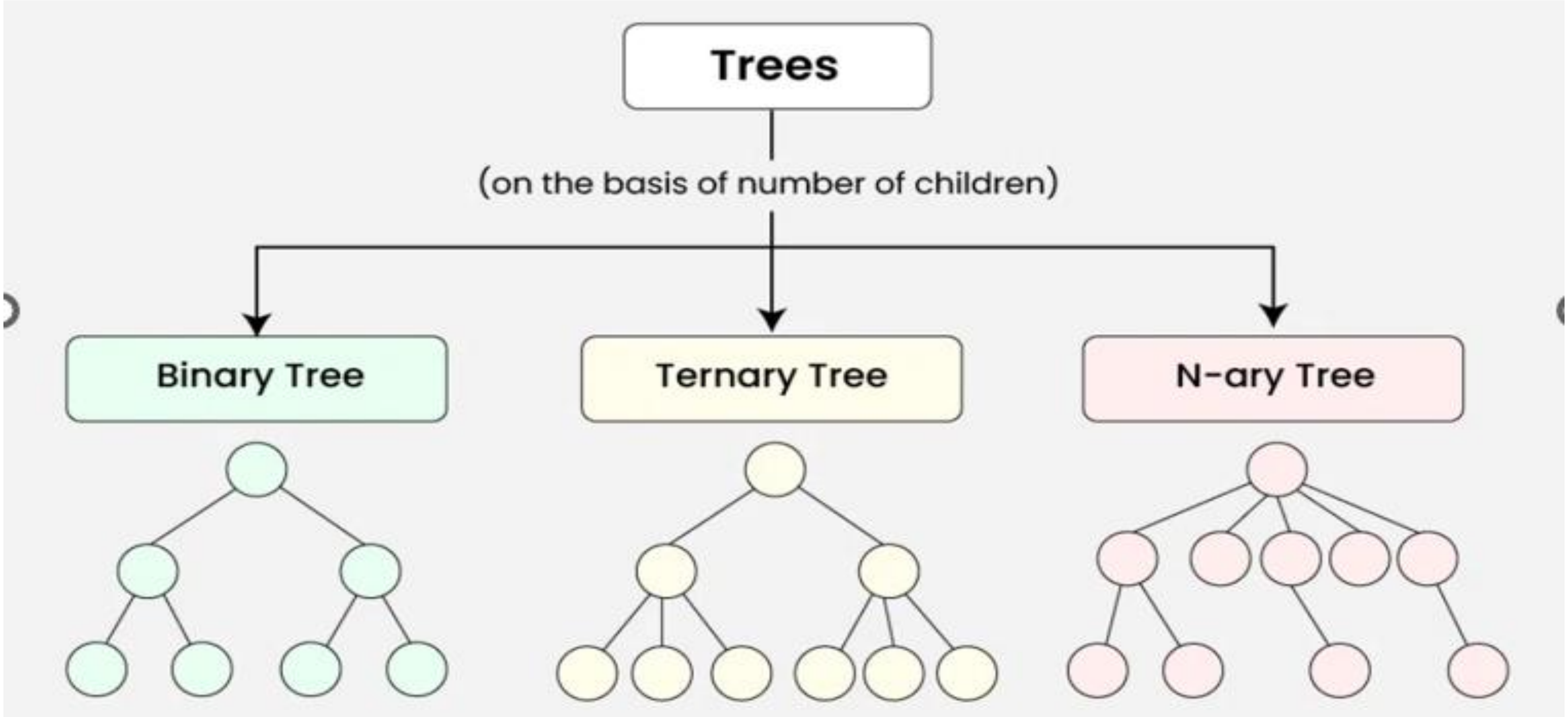
- Binary Search Trees are used to provide Binary Search in linked structures. It has a branching factor of 2!

- Sarting Point in any tree is the root, from there you follow pointers to any node following random memory accesses.

- A Balanced BST has a good spread that keeps the height of the tree approx. O(log(N))

- A perfect Binary Search Tree has the least height for any shape of a Binary search Tree with same keys and same number of nodes.

Introducing Trees with different branching factors



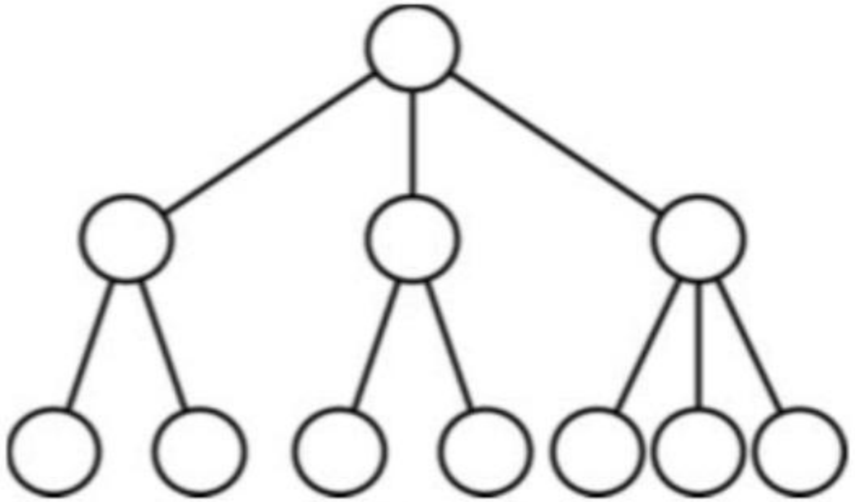
What do M-ary **Search** Trees look like?

2-3 Trees (OR a 3-ary Tree)

Definition:

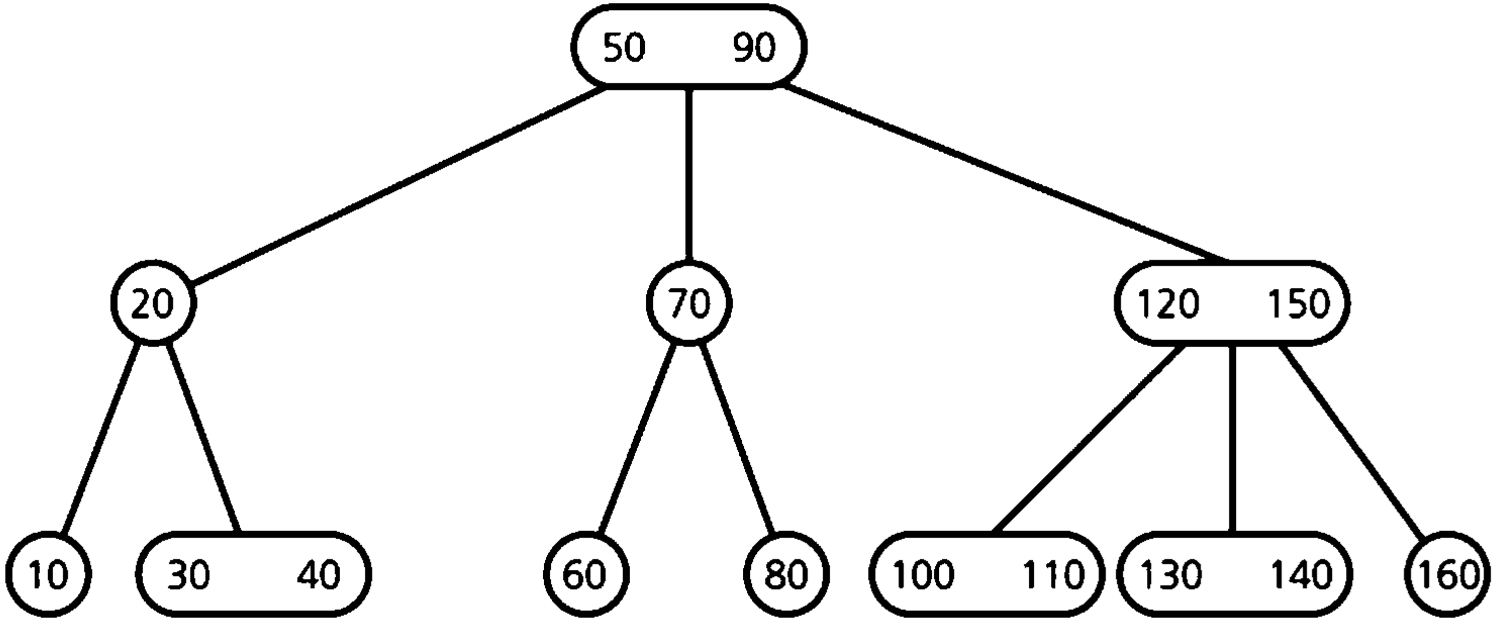
A 2-3 tree is a tree in which each internal node has either two or three children, and all leaves are at the same level.

- **2-node:** a node with two children
- **3-node:** a node with three children



An example of a 2-3 tree

What do 2-3 **Search** Trees look like?



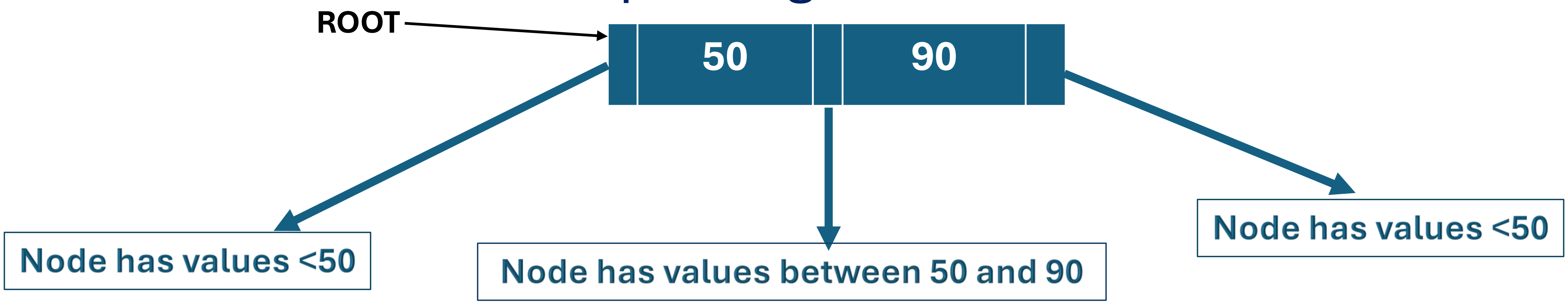
Q1. How would you define a node for such a tree?

Q2. Can you do Binary Search in such a tree?

2-3 **Search** Trees

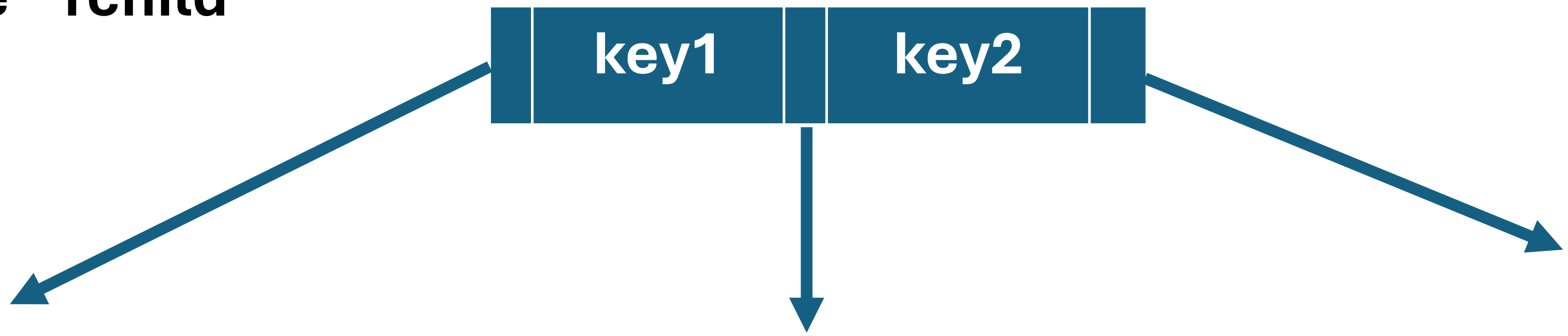
- 2-3 **Search** Trees are used to provide Binary Search in linked structures. It has a branching factor of 3! (more efficient than Binary Search)

- Search: Sarting point is the root, from there you follow a path as can be seen in the simple diagram below.

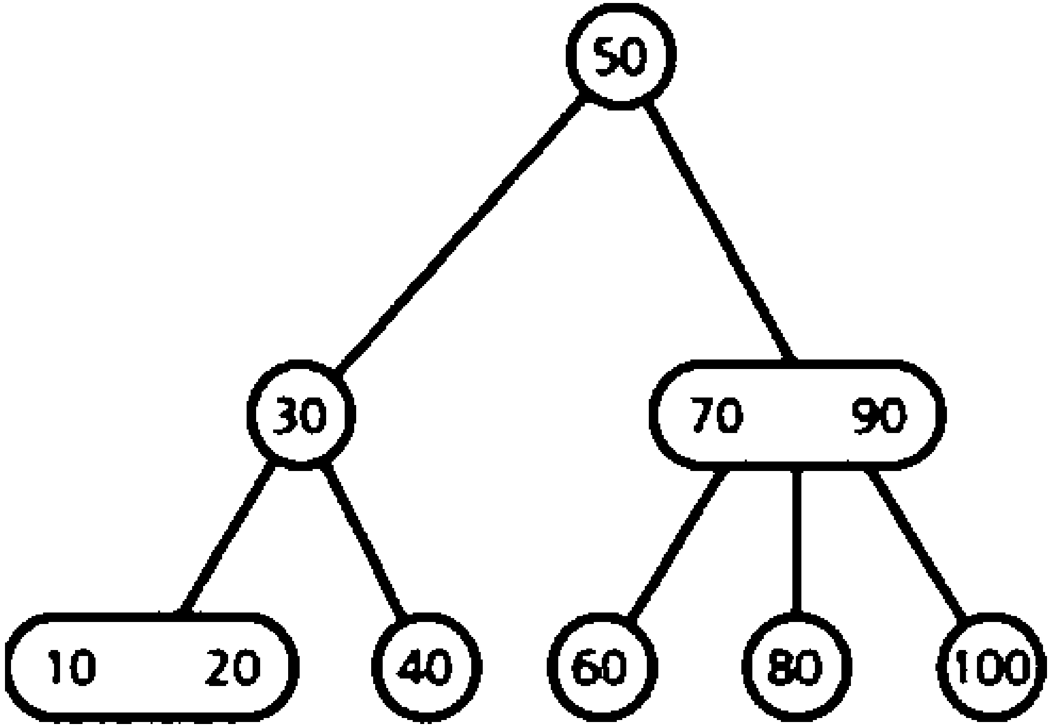


2-3 **Search** Trees Node Structure

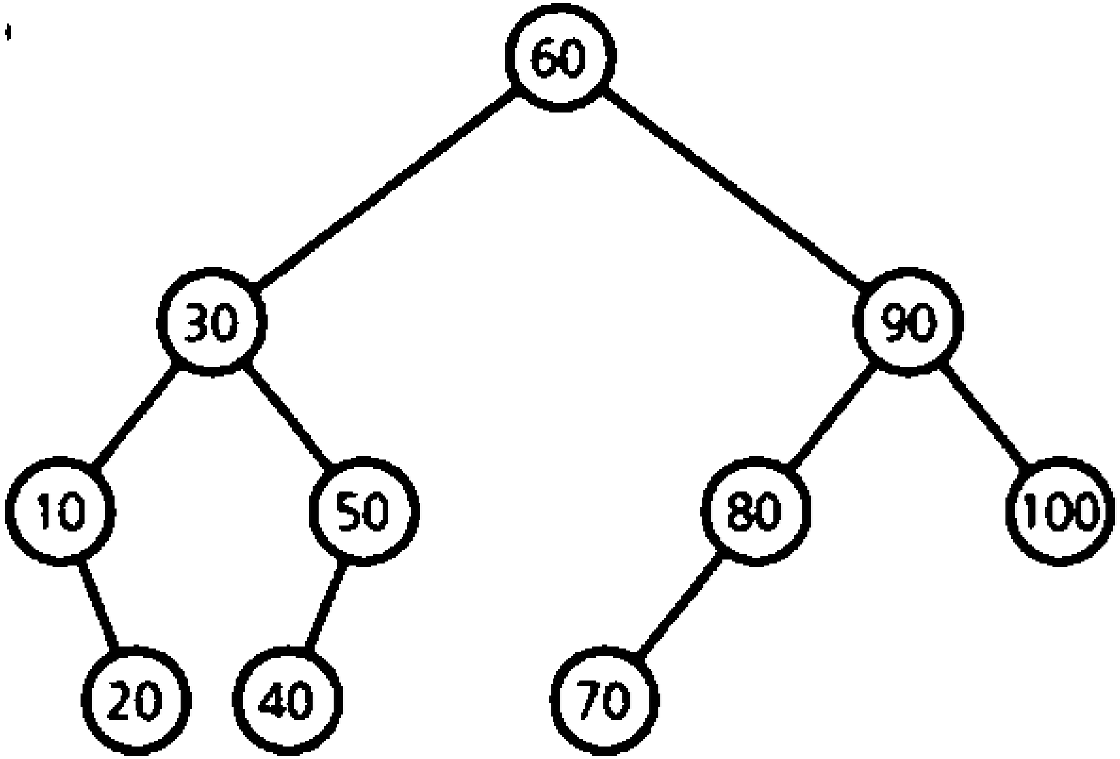
```
Class 3Node<T>
{
        T key1
        T key2

        3Node * lchild
        3Node * mchild
        3Node * rchild

}
```



Comparing BSTree and 2-3 Tree with same values

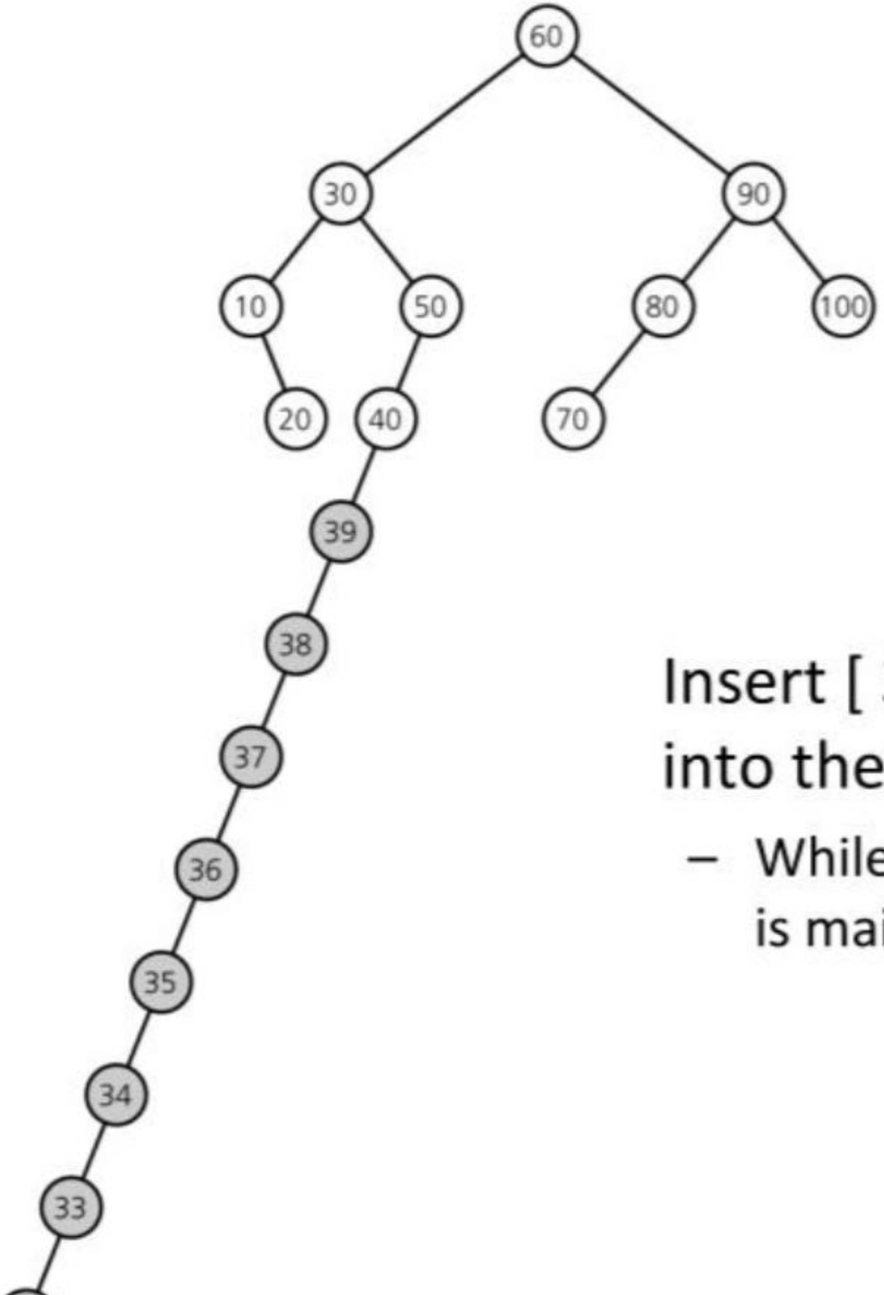


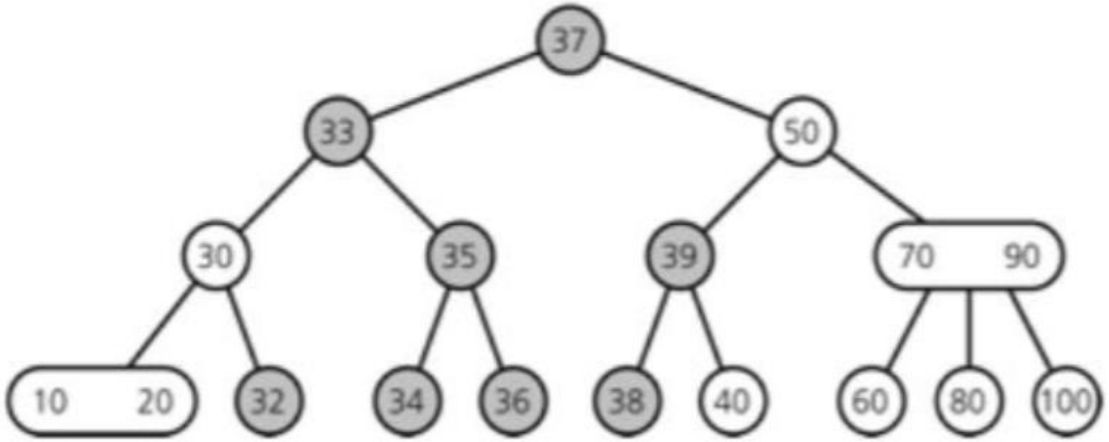
A 2-3 tree with the same elements



A balanced binary search tree

Inserting into a 2-3 Tree





Insert [39 38 37 36 35 34 33 32]
into the trees given in the previous slide

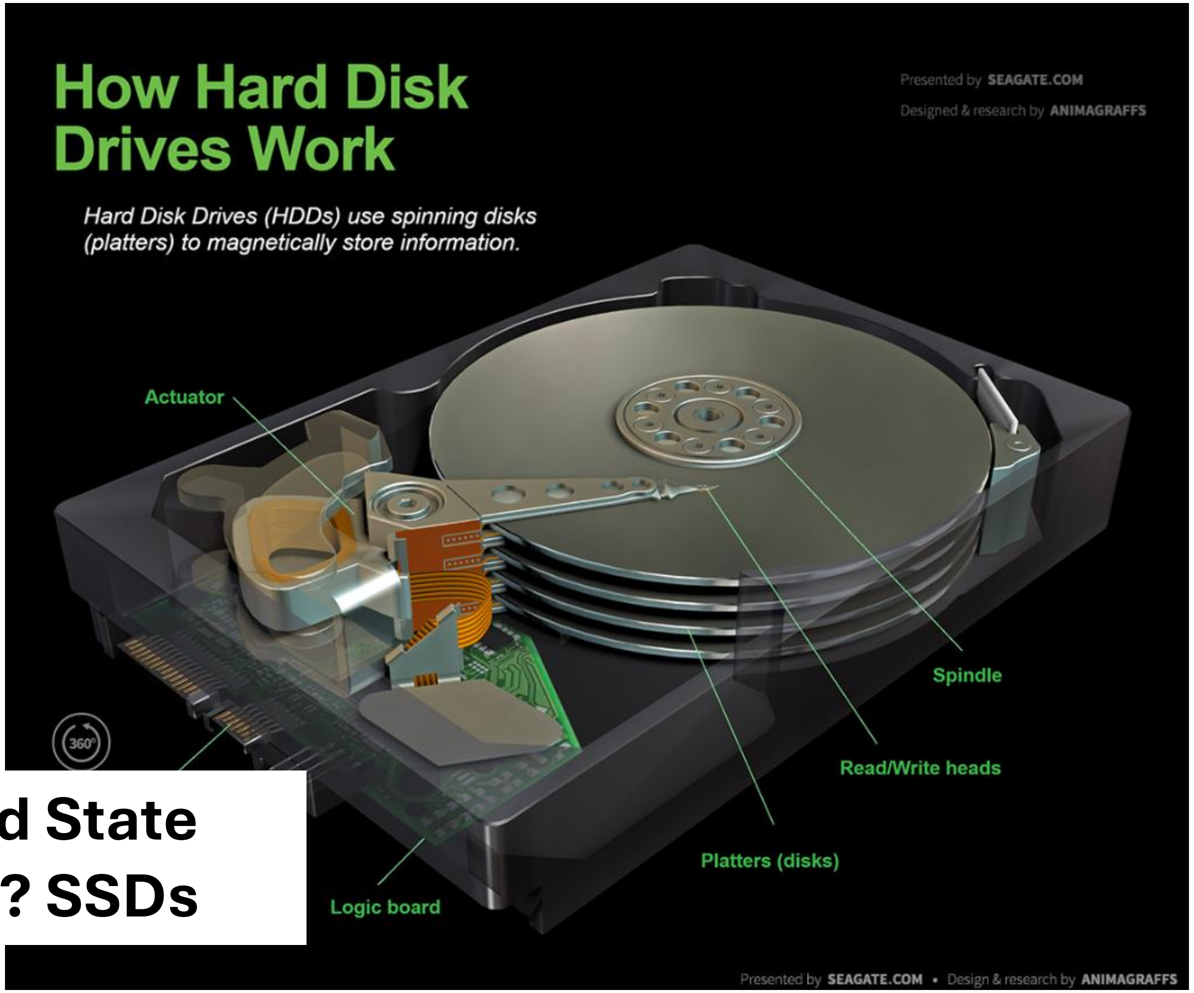
- While we insert items into a 2-3 tree, its shape is maintained

M-ary Search Trees

- **Operations**
  - Search
  - Insert
  - Delete

- **Pros**
  - **Reduced height of the tree**
- **Cons**
  - **Increased complexity of operations**

Motivation for M-ary Trees



What about Solid State Storage Devices? SSDs

Problem to be solved with its constraints

- **Problem at hand:**
  - **Very Very Very** Large Data Set on which to do Binary Search

- **Constraints:**
  - Data Cannot fit into the **RAM** it must be moved to the **Secondary Storage** (HDD, SSD)

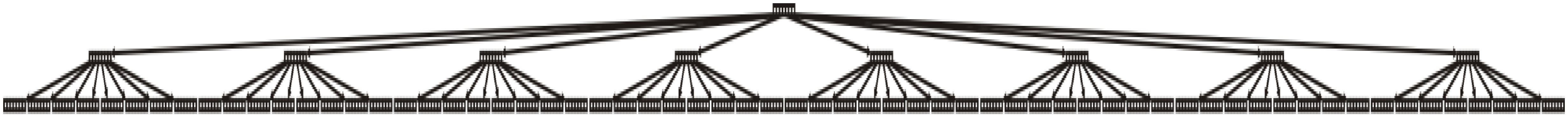
  - Memory Access Speed **Secondary Storage** vs **RAM**

Memory Access Speed **Secondary Storage** vs **RAM**

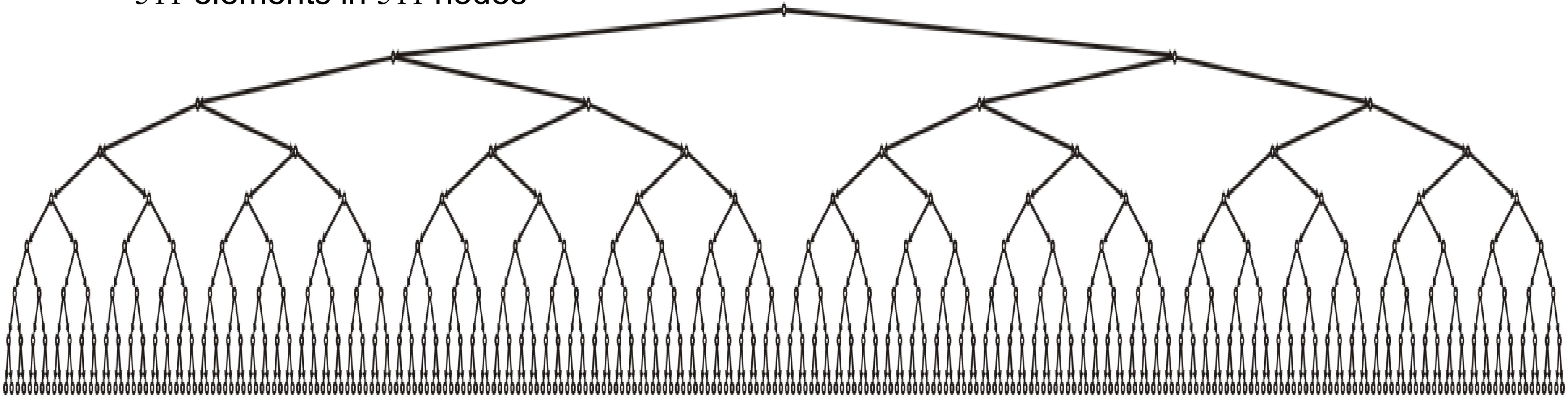
8-way tree versus binary trees

Compare:

- A perfect 8-way tree with $h = 2$
  - 511 elements in 73 nodes



- A perfect binary tree with $h = 8$
  - 511 elements in 511 nodes



Challenge!!

- Cannot fit tree of **Very Very Very** Large Data Set into the **RAM** it must be moved to the **Secondary Storage** (HDD, SSD)

- Following one path in the tree from root to leaf means accesses as many random memory locations on Disk!

- Memory Access Speed of **Secondary Storage** is much much more expensive than memory accesses in **RAM.**

Proposal!

- Use a **m-ary** tree where m (branching factor) is very large to keep the height of the tree as small as possible. (Short paths!)

How large should 'm' be??

Proposal:

Keep tree in Secondary Storage which stores and retrieves data on demand in blocks (size of block is 64MB)