LECTURE-26

B+ Trees and Bloom Filters

CS202: Data Structures (Fall 2025)

Dr Maryam Abdulghafur, Momina Khan

Department of Computer Science, SBASSE

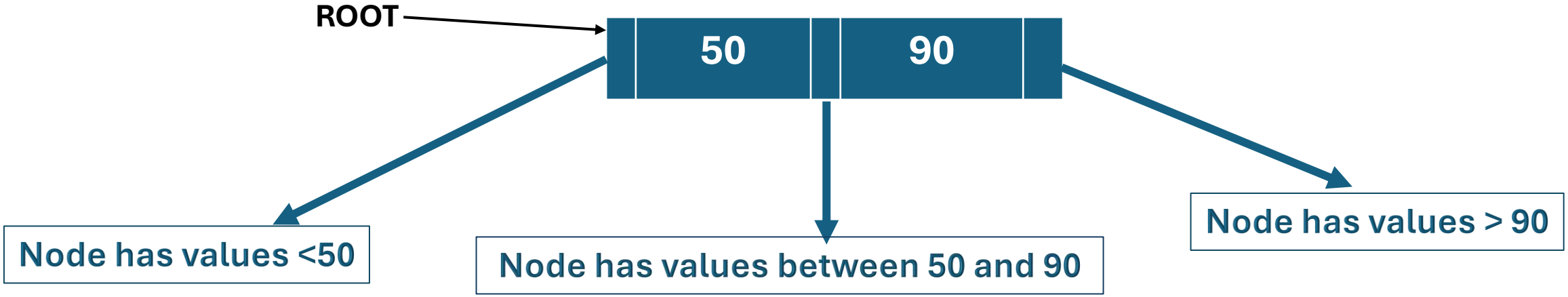
Agenda

- B+ Trees
- Case for Bloom Filters
- Bloom Filter operations and applications

3-ary search tree recap

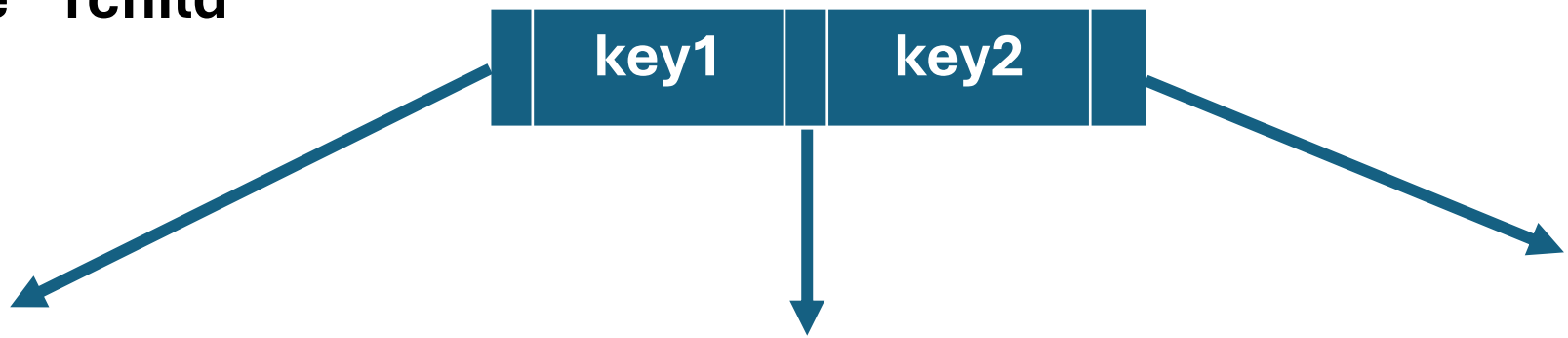
- 2-3 **Search** Trees are used to provide Binary Search in linked structures. It has a branching factor of 3! (more efficient than Binary Search)

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



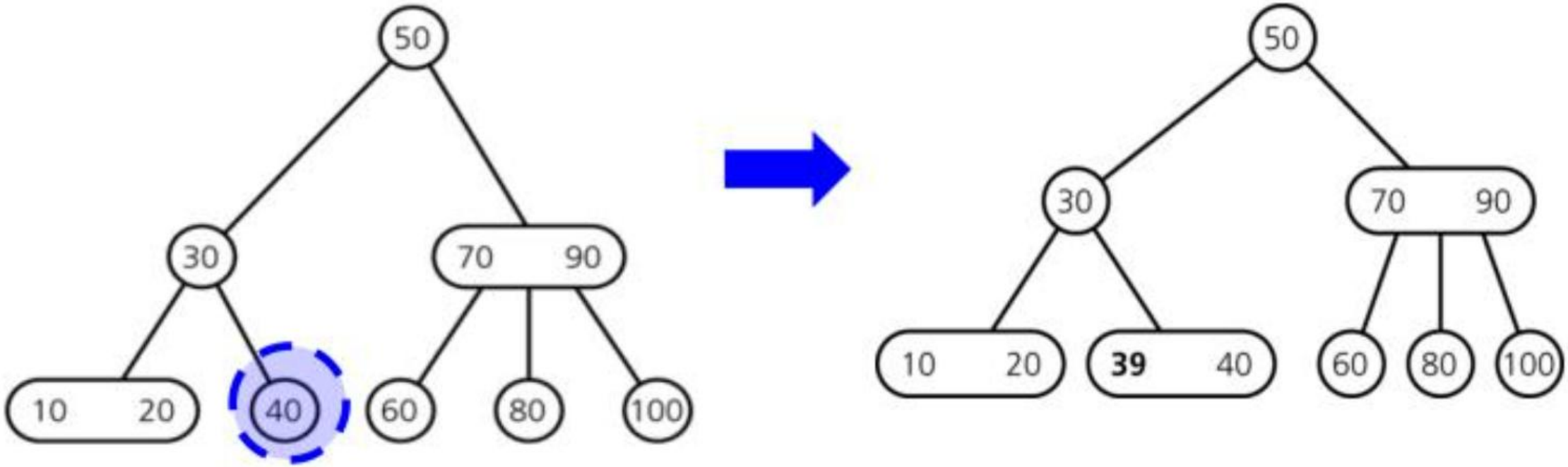
2-3 **Search** Trees Node Structure

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

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

}
```



Inserting into a 2-3 Tree -- Example

Starting from the following tree, insert [39 38 37 36 35 34 33 32]

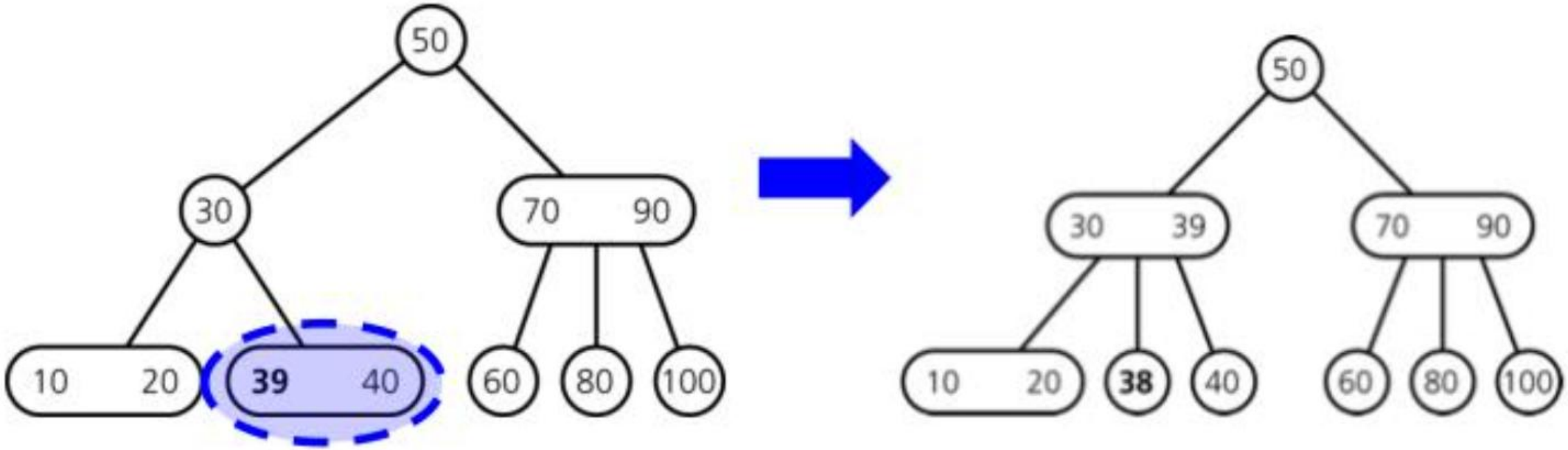


Insert 39

- Find the node into which you can put 39

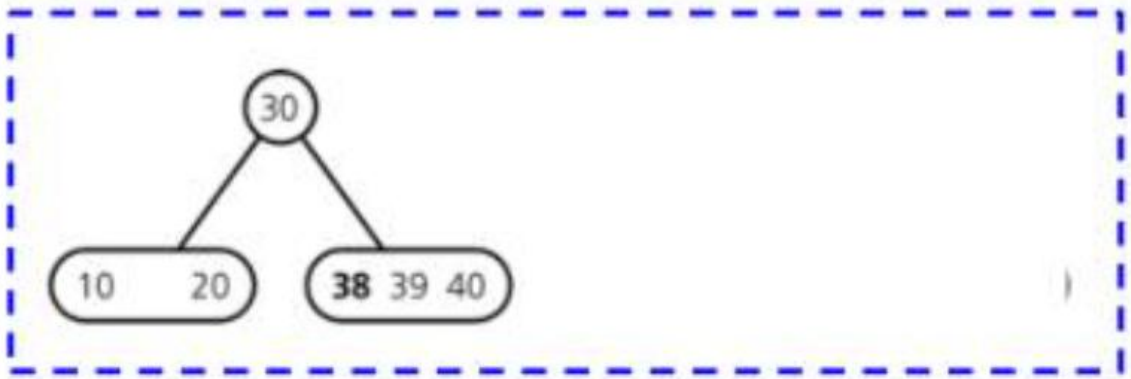
Insertion into a 2-node leaf is simple

Inserting into a 2-3 Tree -- Example



Insert 38

- Find the node into which you can put 38



Insertion into a 3-node causes it to divide

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** is **very** slow when compared to accesses in **RAM!**

B+ Trees

Goal:

B+ Trees have to **HDD/SSD** friendly!

Hint: Random memory accesses are very expensive.

What do you think will make a tree Disk friendly? A large branching factor or a small branching factor? What will be the effect on the tree of a either?

Memory Access **Secondary Storage**

Since memory access on DISK are **very slow**, every memory access made is served as a Block of memory typically 64 MB.

For example: you might has asked for **4** bytes from Disk but the Disk Controller will return these 4 bytes and also the following **(64 MB – 4B) .**

What should the size of one B+ tree node? Should it be smaller, greater or equal to 64MB? What advantage and disadvantage do we have of keeping a node big?

Structure of a B+ Tree

- **Internal Nodes:**
  - All internal nodes only carry keys of different objects.
    No value fields!

- **Leaf Nodes:**
  - Leaf nodes carry the objects (key and value fields)!

All Leaf nodes are connected via a next pointer. Like forming a linked list of leaf nodes!

Structure of a B+ Tree

Internal Nodes:

Every node has keys in ascending order. First key is the smallest key of the second child and so on …

Search is the same as in any search tree, All pointers in the node lead to a node that holds values in a range as in the 2-3 Tree

B+ tree is an *almost perfect* tree with shape of the tree always perfect but nodes in the tree do not necessarily have all slots always filled with keys.

All Leaf nodes are connected via a next pointer. Like forming a linked list of leaf nodes!

Structure of a B+ Tree

Internal Nodes:

Every node has keys in ascending order. First key is the smallest key of the second child and so on …

Search is the same as in any search tree, All pointers in the node lead to a node that holds values in a range as in the 2-3 Tree

B+ tree is an *almost perfect* tree with shape of the tree always perfect but nodes in the tree do not necessarily have all slots always filled with keys.

All Leaf nodes are connected via a next pointer. Like forming a linked list of leaf nodes!

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

B+ Tree visualization

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Unsuccessful Searches

Data Structure	O() for key that is not in the Data Structure
Sorted Array	O(logN)
Sorted Linked List	O(N)
Unsorted Array	O(N)
AVL Tree	O(logN)

BLOOM Filters

This is a membership data structure. Its only use is to tell whether a data item is in the set or not.

As opposed to retrieving data for the object!

A **Bloom Filter**:

- Does not keep any data … it is a bit map!
- Actual data objects are kept in a separate Data Structure.
- Shares its workings and philosophy with a Hash Table!

BLOOM Filters

Actual data objects are kept in a separate Data Structure like Linked Lists or AVL Trees etc.

Bloom Filter is used as a second data structure to improve look up time for elements that are **not present**. (unsuccessful searches)

Every time an object is inserted into our data structure its key is also simultaneously inserted into the Bloom Filter.

Searches are done in the opposite order. When an object is to be searched its key is first searched in the Bloom Filter and only if it gives a positive as an answer a search is made in the corresponding data structure!

BLOOM Filters

- **Properties**:
  - **No False Negatives**: If a Bloom filter says an item is **not** in the set, it is definitely not there. This makes it safe for "does not exist" checks .
  - **Possible False Positives**: If a Bloom filter says an item **is** in the set, it might be a false alarm. The system must then perform a definitive (but more expensive) check in the data structure that stores the objects.
    - False alarms may occur if even though our search key was never added another key was inserted earlier that collides with our search key!

BLOOM Filters Application

A classic example is a web browser checking against a list of malicious URLs. These lists are quite large (about 1 million malicious URLs, the size of this file being around 25MB). The browser queries the Bloom filter first to see if the asked for URL is in the malicious URLs or not. The browser can immediately tell that it is a safe URL (not malicious) if the Bloom Filter response is negative. This way the browser can load the URL site safely without ever consulting the actual list.

BLOOM Filters Performance

Every operation is as simple as running key by hash function and getting the index into the Bloom Filter array.

Search and insert into Bloom Filter are both O(1) operations.