Tree structures

B-tree

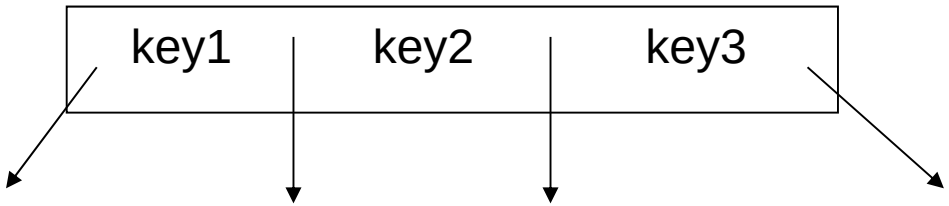
Agenda In this lesson

- B-tree
  - Definition
  - Properties
  - Operations
  - Example
  - Implementation
  - Application areas
- **Efficiency** is gained at the cost of unused space in the indexes
- **Confusions** to avoid!
- **Be aware** of the **terminology** used for B-trees
- **Be aware** of the "family" of B-trees:
  - B-tree
  - B^+-Tree
  - B*-Tree

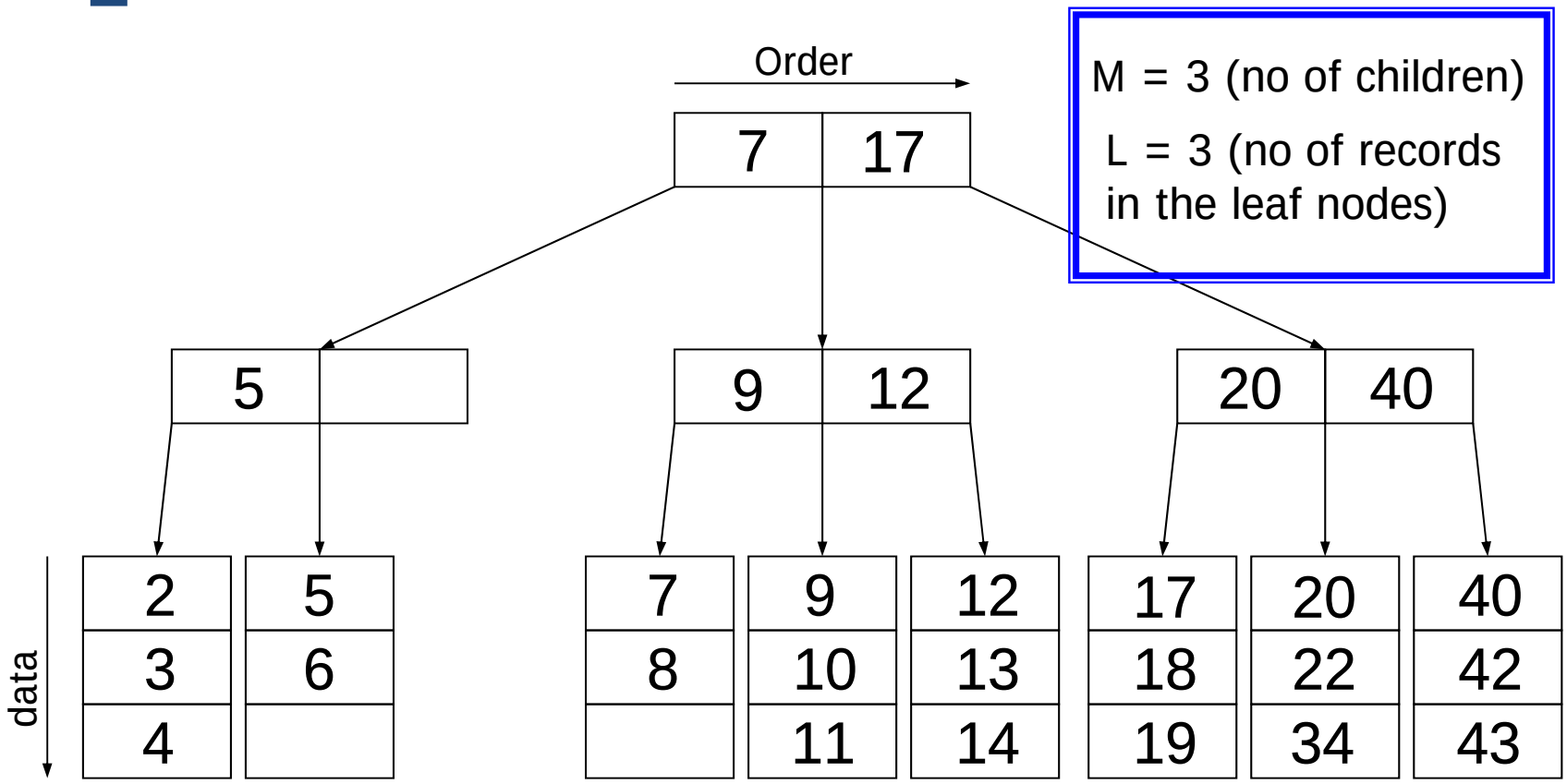
B-tree - Definition

- B-tree is an abbreviation for **BALANCED TREE**
- B-tree is completely balanced
- B-tree can have different "**degrees**" (or "**order**" not to be confused with order in a sequence), from **3** and upwards

  (the definition can vary between textbooks)
- The **degree** determines
  - the number of children per parent
  - the number of search keys per node (max degree-1)
- E.g. **degree** 4



B^+-tree – An Example degree (order) 3



B-tree Properties

- A B-tree has the following properties & invariants
  - M-ary tree (cf. Binary, Ternary, Quaternary, etc) where

  - **The root is either a leaf or has between 2 and m children**
  - All non-leaf nodes contain between **1 and M - 1 search keys**
  - All non-leaf nodes have between **Ceil(M/2) and M children**
  - **All leaves are at the same depth**

  - Ceil is a mathematical function - "Ceiling" which converts a real number to the nearest integer above
  - Example: degree 3 ➔
    - All non-leaf nodes have 1 or 2 keys;
    - All non-leaf nodes contain between 2 and 3 children

B-tree Properties (contd)

- Data (information) is stored in the leaves
  - **All leaf nodes are on the same level**
  - There is space for **L elements** in each leaf node
  - each node holds between **Ceil(L/2) and L children**
  - Data elements are stored in a sorted sequence, usually an array
  - Since the data elements are sorted, a **binary search** may be used to find each element

B-tree Properties (contd)

- What does ”search key” mean?
  - A search key specifies where in the tree the data value is to be found
  - In a BST – less than➔ left, greater than ➔ right
  - In a B-tree we have a generalisation of ” less than” and ”greater than” to ”**less than**”, ”**between**” and ”**greater than**”

B-tree Properties (contd)

- Search keys continued
  - For a B-tree with **M = 4** there are **3 search keys** in each **non-leaf** node
  - Suppose these keys are: **10, 20, 30**
  - Data elements may thus be stored in four ways
    - **x<10, 10≤x<20, 20≤x<30, 30<=x**
    - Less than 10 and greater than 30
    - between 10 and 20, and between 20 and 30

B-tree - Operations

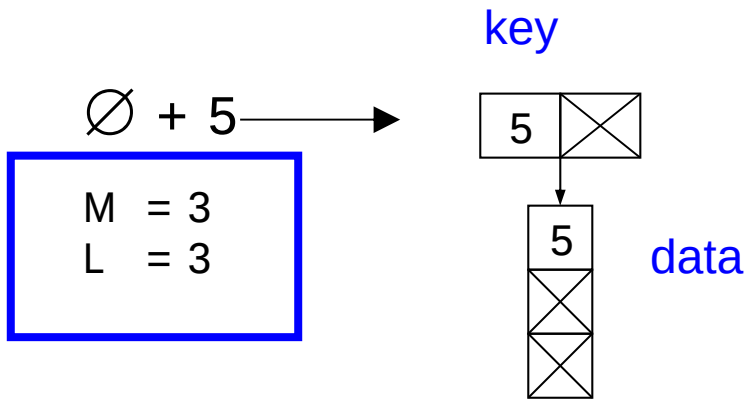
Operation	In	Out
Create		**BT**
Add	**BT** x **v**	**BT**
Remove	**BT** x **v**	**BT**
Find	**BT** x **v**	**R**
IsEmpty	**BT**	True or false

B^+-tree - Construction

- How is a B^+-tree constructed?
  - As with a ”normal” tree we begin from the **empty case** and build up the tree node for node
  - This is done using the **Add** operation

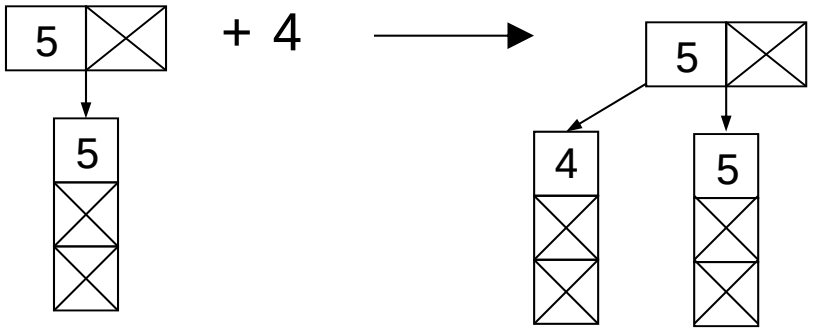
B+-tree – Construction (contd)

- How does **Add** work?
  - Empty Tree + 5
  - Key + data

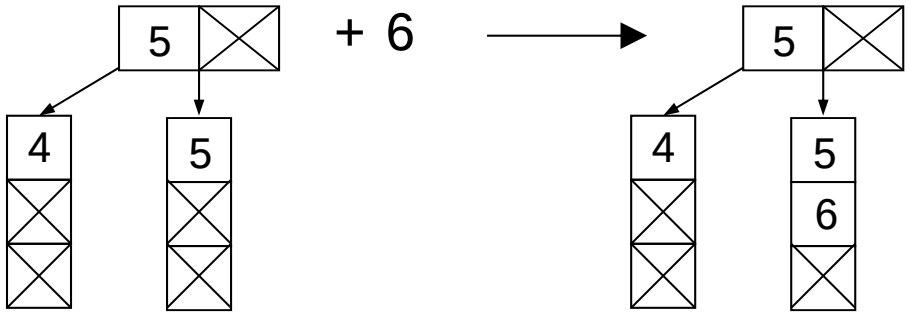


- Invariant violation!!!! 1 barn < **Ceil(M/2) = 2**
- The root node is an exception
  - otherwise it would be impossible to create a B-tree from scratch

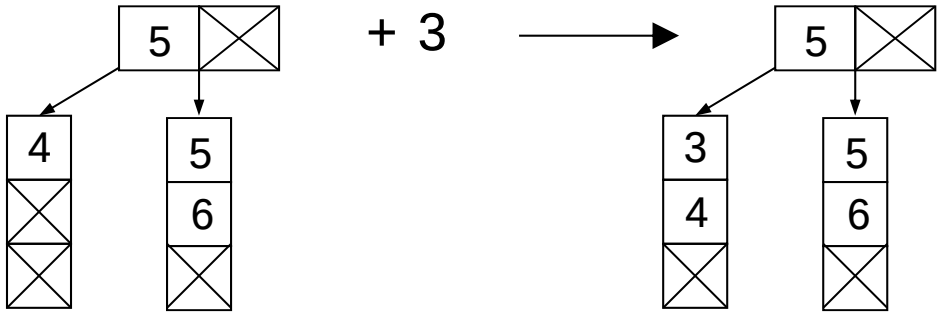
B$^+$-tree – Construction (contd)



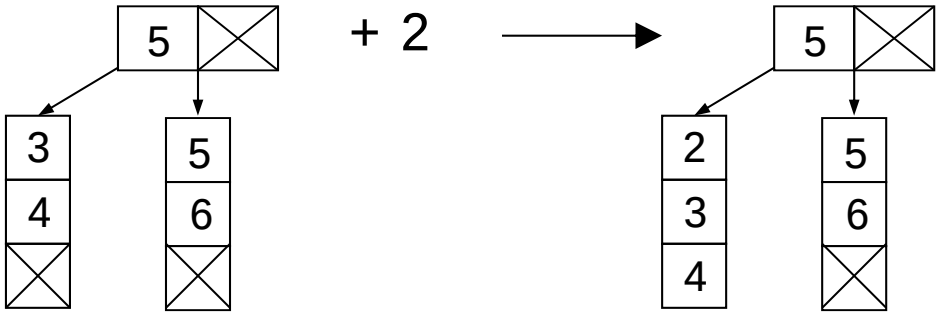
B-tree – Construction (contd)



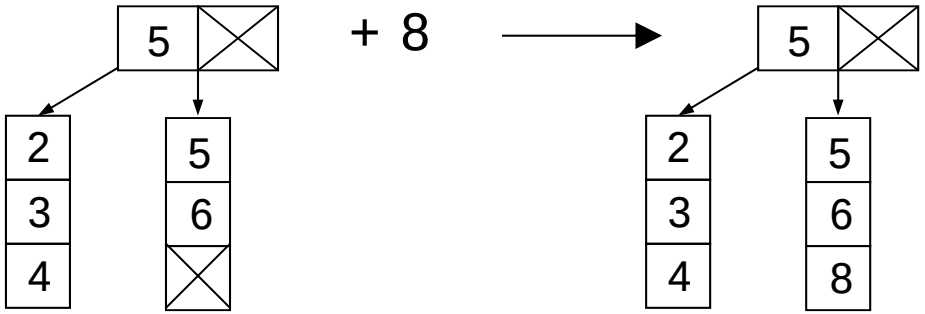
B$^+$-tree – Construction (contd)



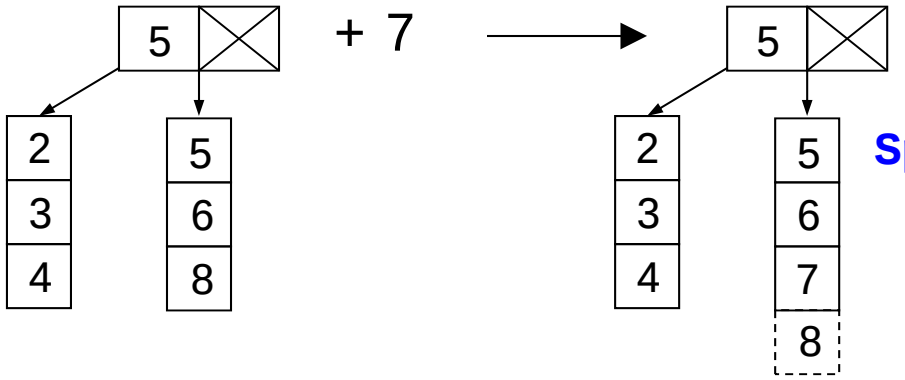
B$^+$-tree – Construction (contd)



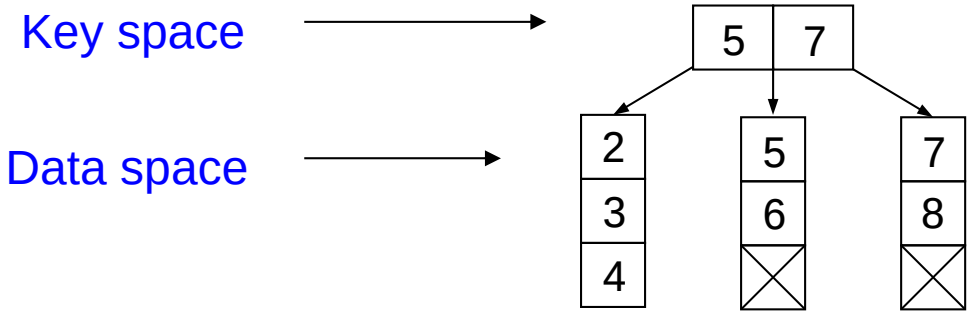
B$^+$-tree – Construction (contd)



B+-tree – Construction (contd)

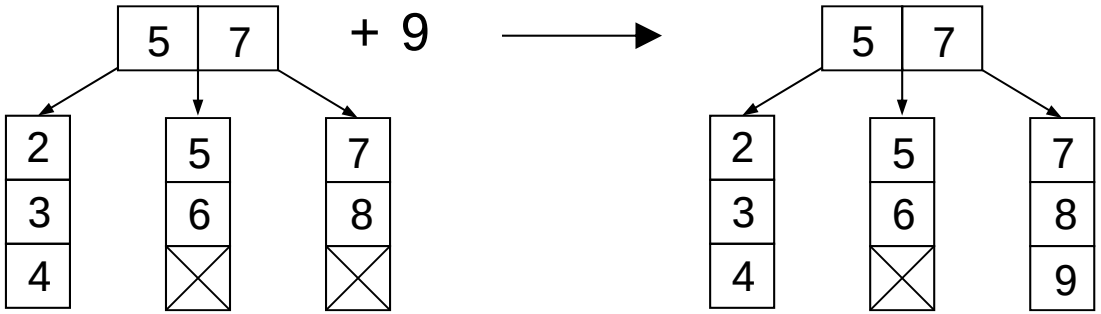


Split required, 4 > L = 3

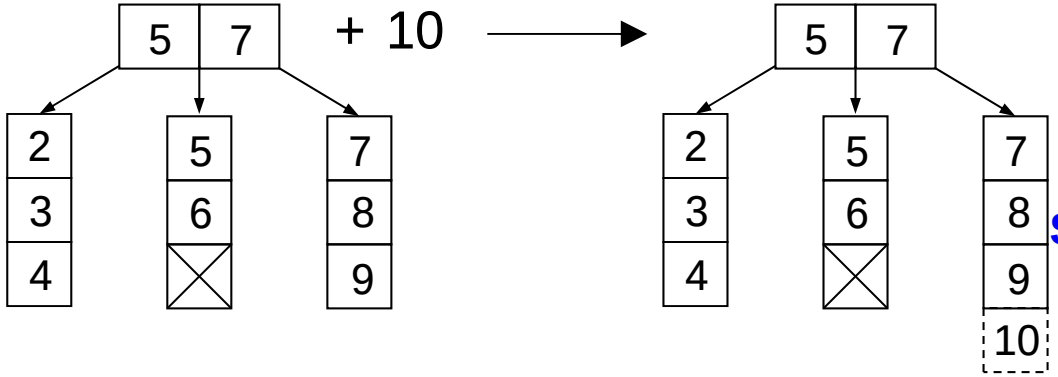


NB the search key value (7) is also moved up (promoted) to the parent!

B$^+$-tree – Construction (contd)



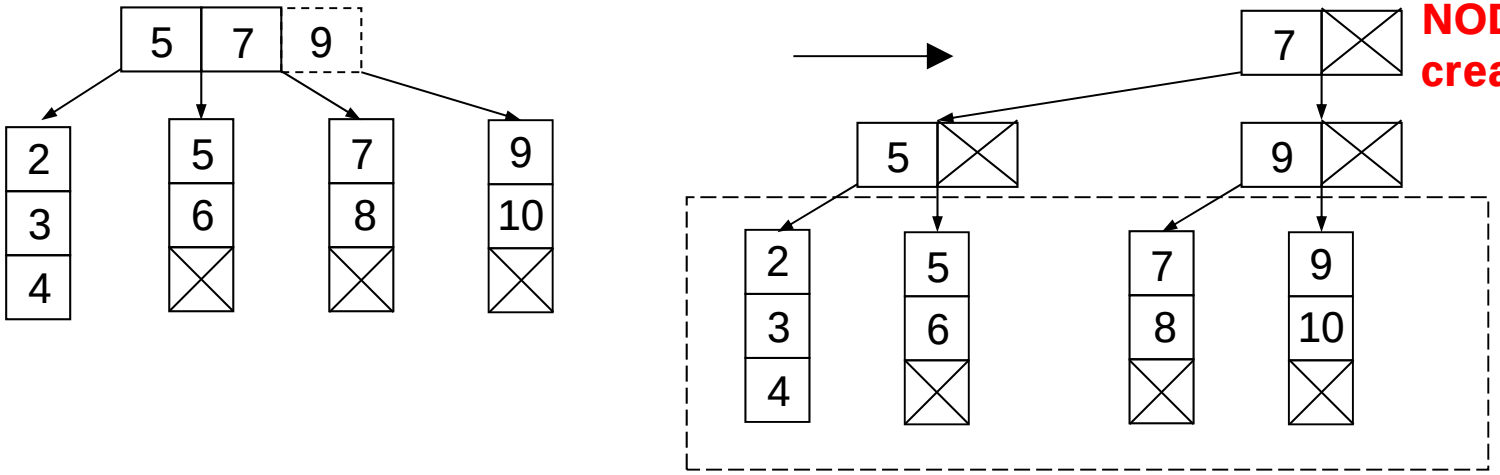
B$^+$-tree – Construction (contd)



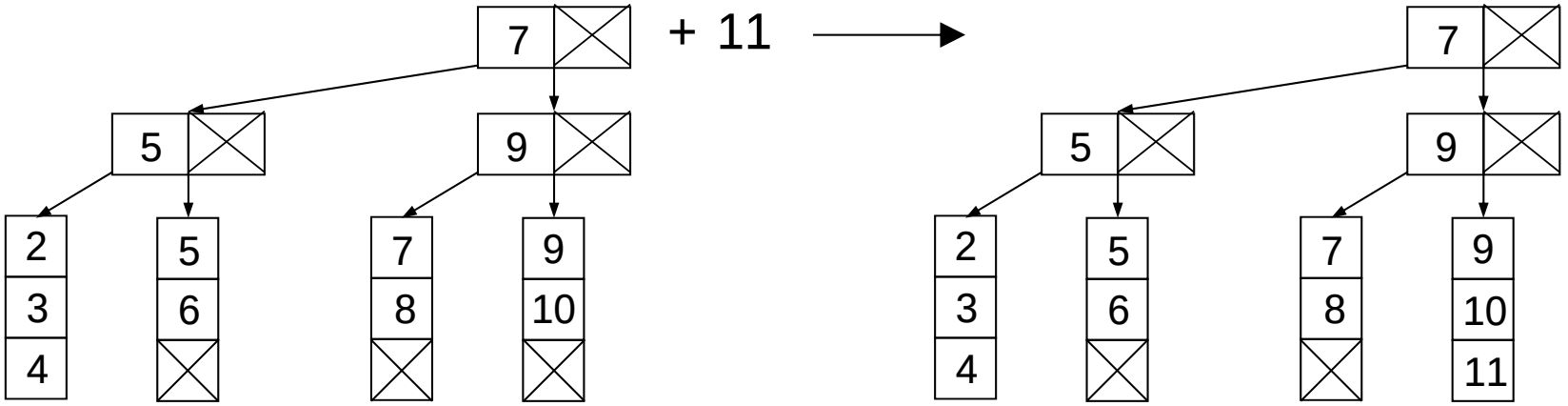
Split required, 4 > L = 3

The height of the tree is increased, 3 > M - 1 = 2

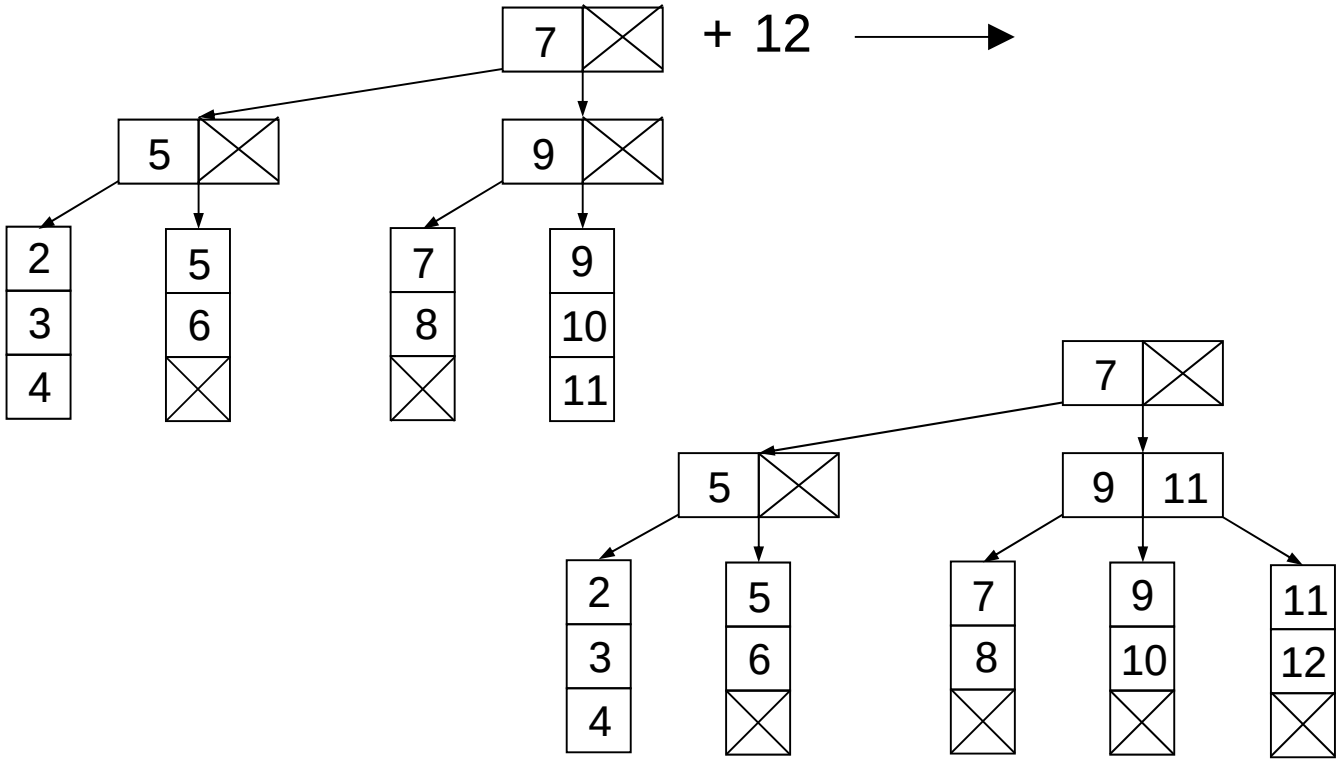
A new ROOT NODE is created!



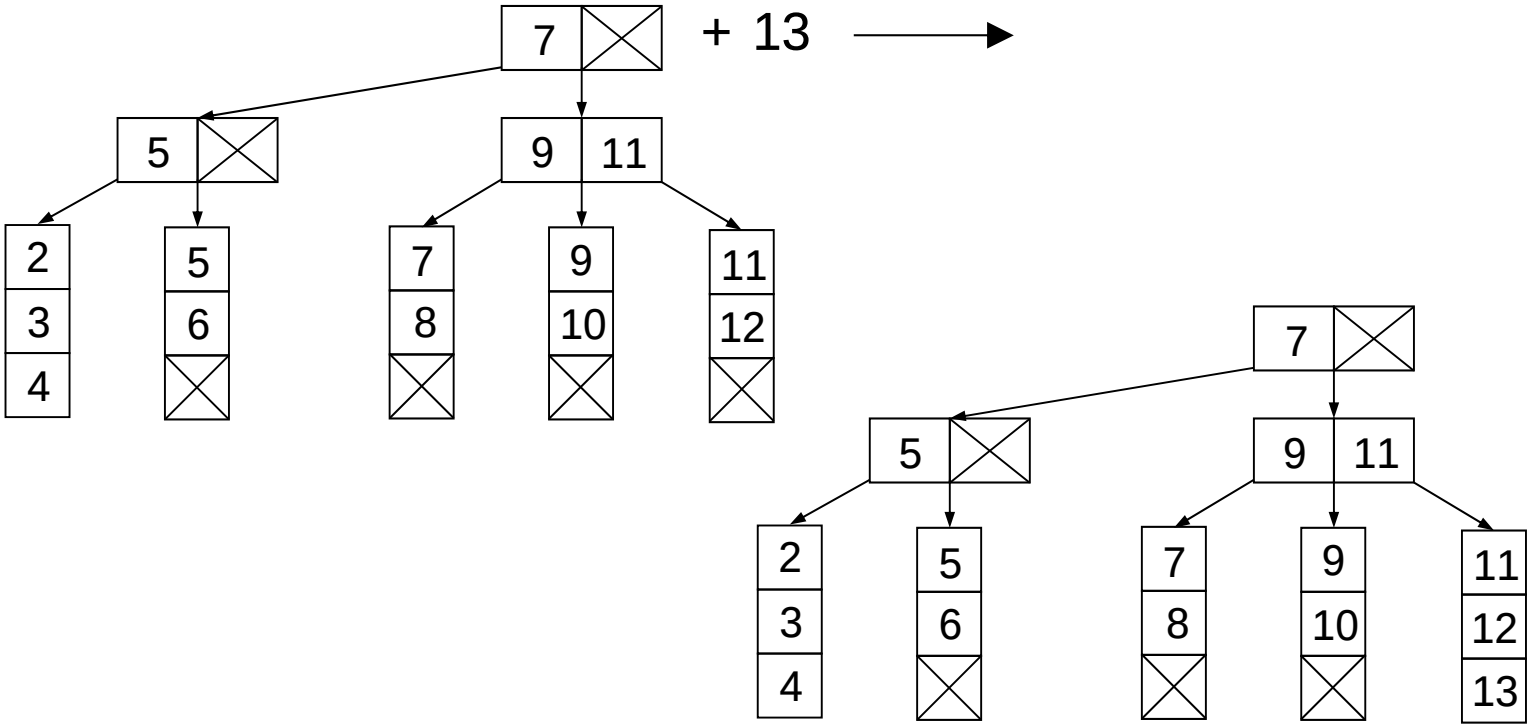
B+-tree – Construction (contd)



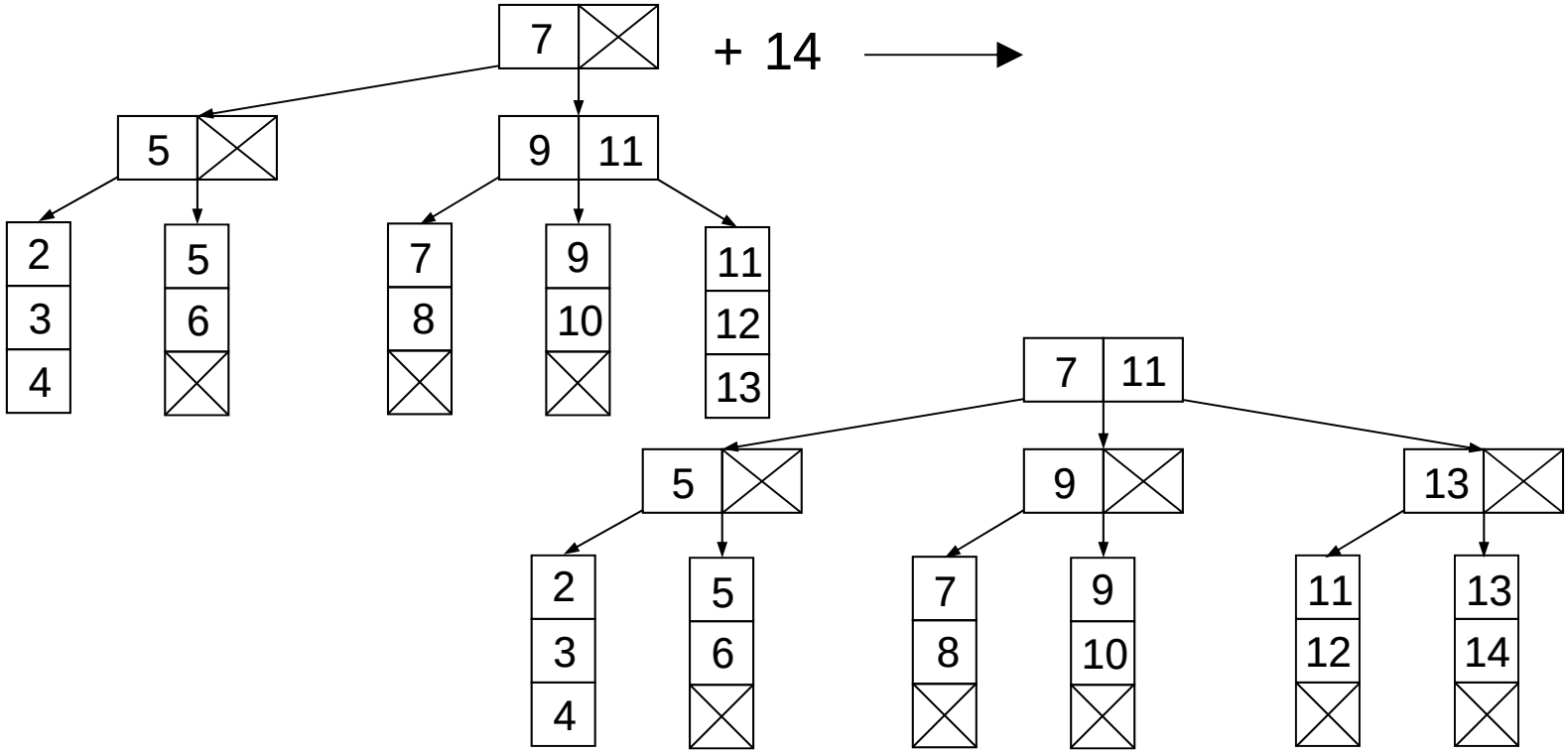
B$^+$-tree – Construction (contd)



B$^+$-tree – Construction (contd)



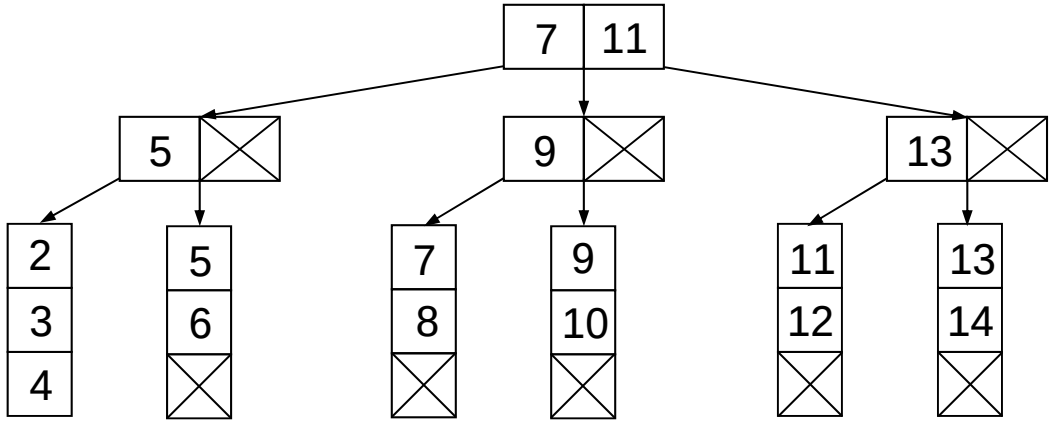
B$^+$-tree – Construction (contd)



B+-tree - remove

- This is the “reverse” of add – recall
  - **The root is either a leaf or has between 2 and m children**
  - All non-leaf nodes contain between **1 and M - 1** **search keys**
  - All non-leaf nodes have between **Ceil(M/2) and M children**
  - **All leaves are at the same depth**

remove 7



B^+-tree - Implementation

- A B-tree contains 2 search spaces
  - The **search keys**
  - The **data elements**
- The **search key space** is held in **PRIMARY memory** (faster access)
- The **data space** is held in **SECONDARY memory**

B^+-tree - Implementation

- The **data space** is usually considerable larger than the search **key space**
  - Today 1 gigabyte internal (primary) memory is usual for a server
  - Several thousand gigabytes is not uncommon for the secondary memory
  - For data collections larger than 1 gigabyte secondary memory is required

B^+-tree - Implementation

- As an example, consider a database of images where some search key has been defined
- The size of each image is 100 kilobyte
- The number of images is 2000000 ($2 \cdot 10^6$)
- Therefore 200000000 kilobyte $\approx$ 200 gigabytes are required
- If you have 1 gigabyte of primary memory, B-tree might be a useful structure in which to store the images

B^+-tree - Implementation

- The B-tree required to catalogue the images (the search key space) is much smaller that the total storage required for the images – in this case perhaps as little as 10 megabytes
- **SEARCH KEYS** - **primary memory**
- **DATA ELEMENTS** - **secondary memory**

B^+-tree – Application Areas

- The main area for B-trees is **databases** (DBs)
  - There are DBs larger than terabyte (~10^{12} bytes) and B-trees are useful in such applications
- B-trees have also been used in file systems
  - It is not uncommon that one has around 10000 files at home in a PC – some servers handle even more files
  - Hence it has been found that B-trees are even applicable in file systems

B-Tree, B$^+$-Tree, B*-Tree

- B-Tree
  - Nodes contain **keys + data**
- B$^+$-Tree (from Knuth's definition)
  - Non-leaf nodes contain **keys**
  - Load factor in key space about 50%
  - Leaf-nodes contain **data**
- B*-Tree
  - Load factor in key space about 66%

Summary: B-Tree

- Properties
  - **DEGREE** (**M**)
  - **The number of data items per leaf** (**L**)
- Operations
  - How Add and Remove work is not so trivial
- Implementation
  - **Search keys** in primary memory
  - **Data** in secondary memory
- Application Areas
  - Databases
  - File systems

- B-Tree
- B^+-tree
- B*-tree

Reference Literature

- Data Structures and Problem Solving Using C++, [Weiss]
  - sid. 709-717
- Introduction to Algorithms, [Cormen, Leiserson, Rivest]
  - sid. 381-399
- B-Tree Visualization (Program)*, Sebastian
  - ftp://ftp.cdrom.com/pub/simtelnet/win95/prog/btree10.zip[1]

[1] requires Microsoft Windows
* Very informative, recommended