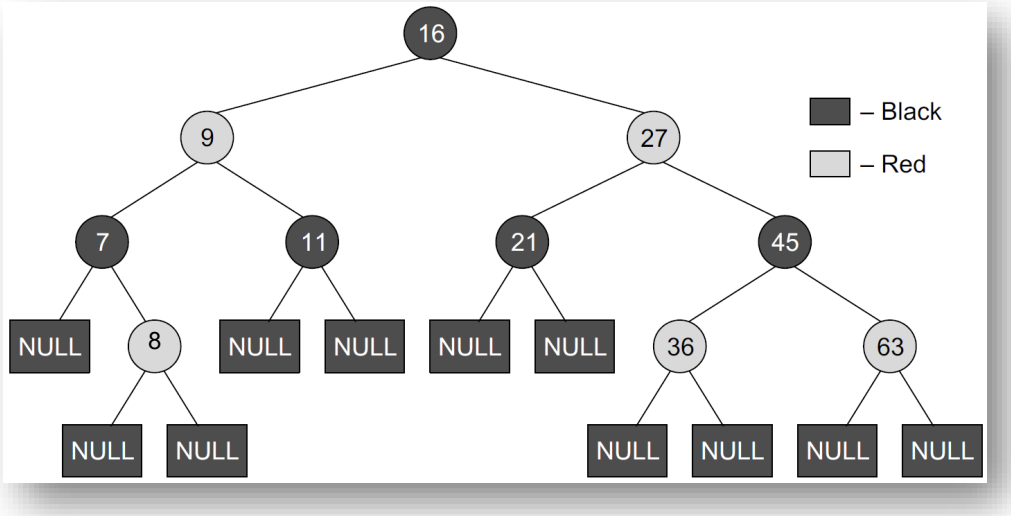
Splay Trees

Kuan-Yu Chen (陳冠宇)

Review

- Red-Black Trees
  - A red-black tree is a binary search tree in which every node has a color which is either **red** or **black**
    1. The color of a node is either red or black
    2. The color of the root node is always black
    3. All leaf nodes are black
    4. Every red node has both the children colored in black
    5. Every simple path from a given node to any of its leaf nodes has an equal number of black nodes



SPLAY Trees

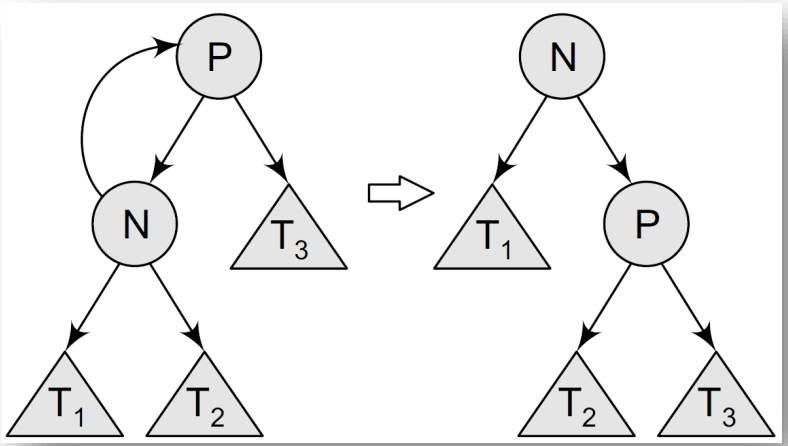
- Splay trees were invented by Daniel Sleator and Robert Tarjan, 1985

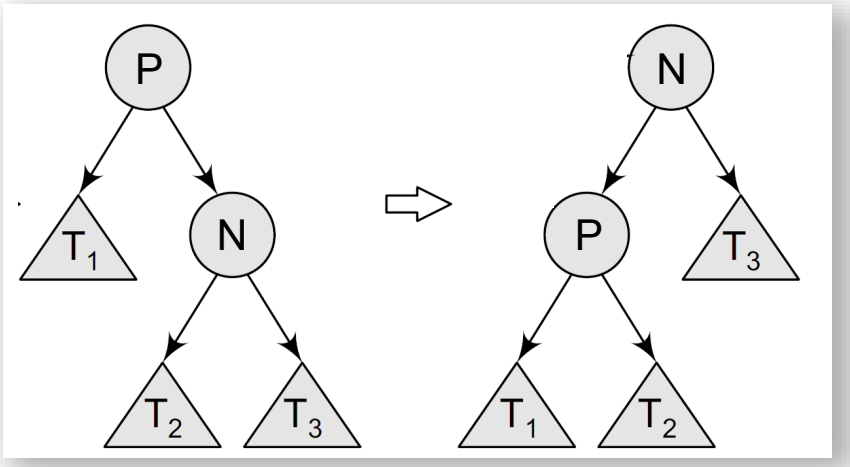
- A splay tree is a **self-balancing binary search tree** with an additional property that **recently accessed elements can be re-accessed fast**
  - A simple idea behind it is that if an element is accessed, it is likely that it will be accessed again

- For many **non-uniform** sequences of operations, splay trees perform better than other search trees

SPLAY Trees – Splaying.

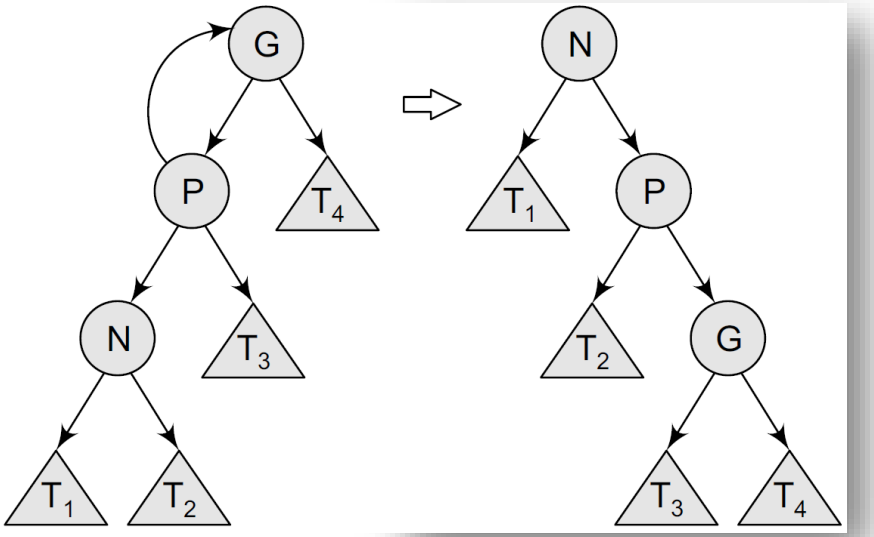
- Splaying is performed on the node N to move it to the root
  - Zig Step
    - The zig operation is done when P (the parent of N) is the root of the splay tree

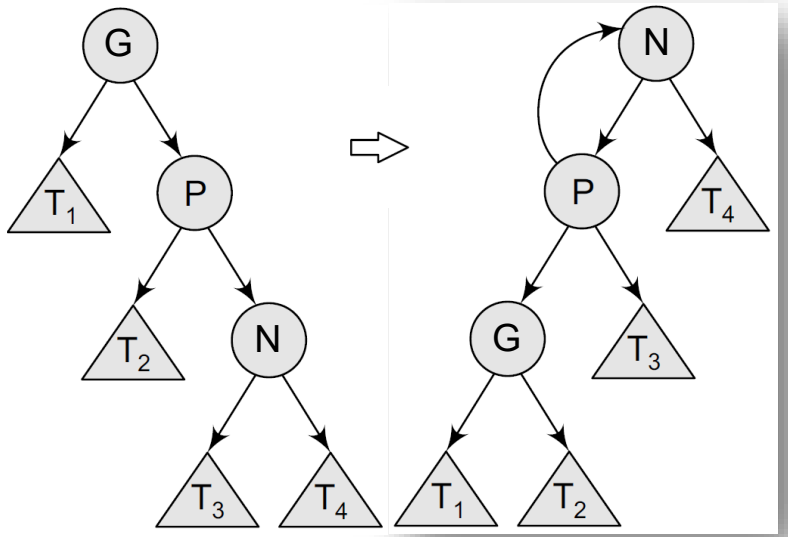




SPLAY Trees – Splaying..

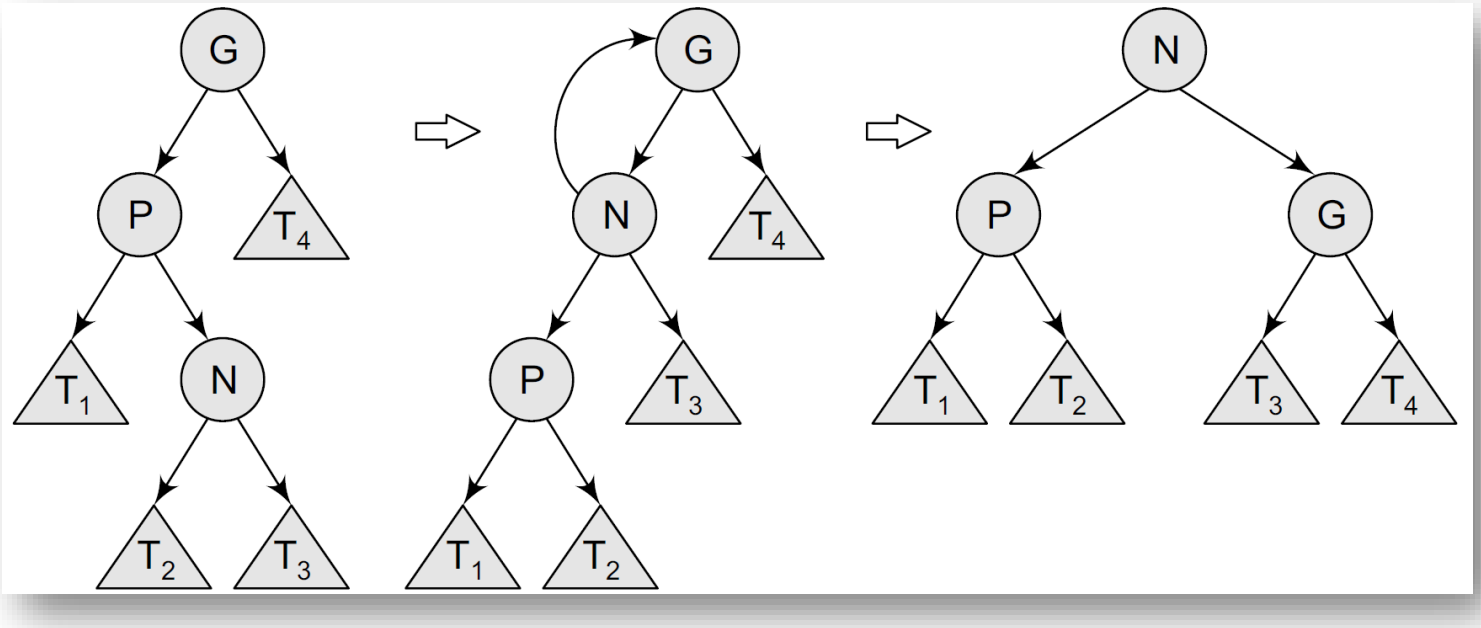
- Splaying is performed on the node N to move it to the root
  - Zig-zig Step
    - The zig–zig operation is performed when P is not the root
    - Besides, N and P are either both right or left children of their parents





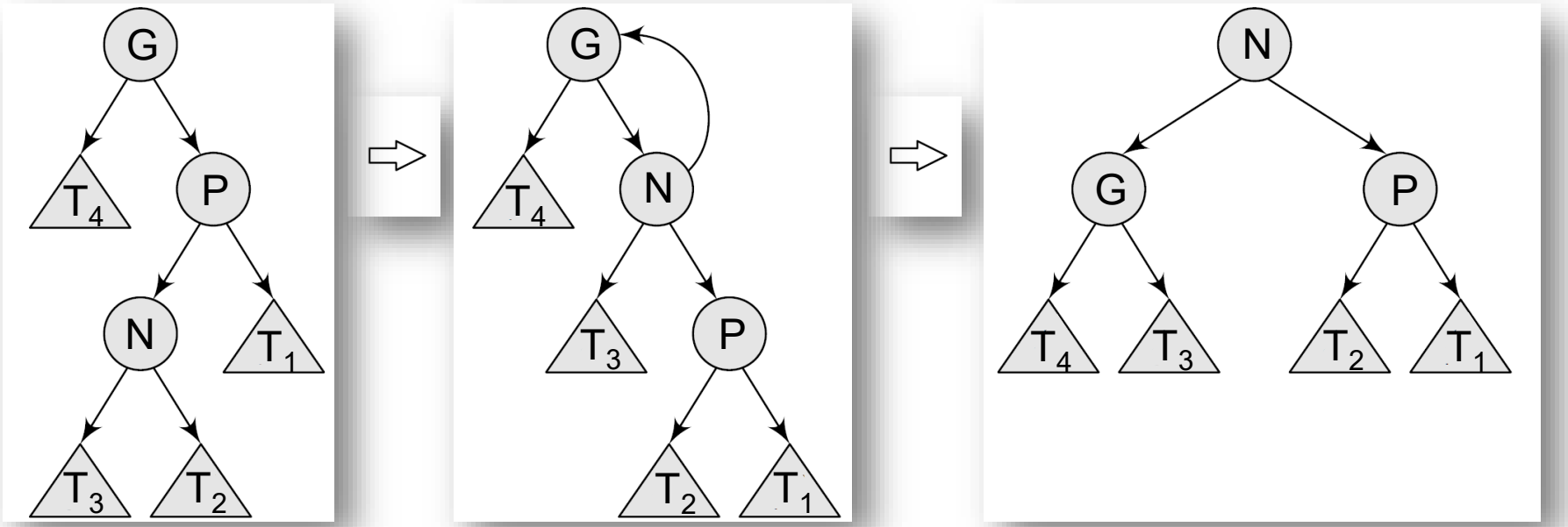
SPLAY Trees – Splaying...

- Splaying is performed on the node N to move it to the root
  - Zig-zag Step
    - The zig–zag operation is performed when P is not the root
    - In addition to this, N is the right child of P and P is the left child of G or vice versa



SPLAY Trees – Splaying...

- Splaying is performed on the node N to move it to the root
  - Zig-zag Step
    - The zig–zag operation is performed when P is not the root
    - In addition to this, N is the right child of P and P is the left child of G or vice versa

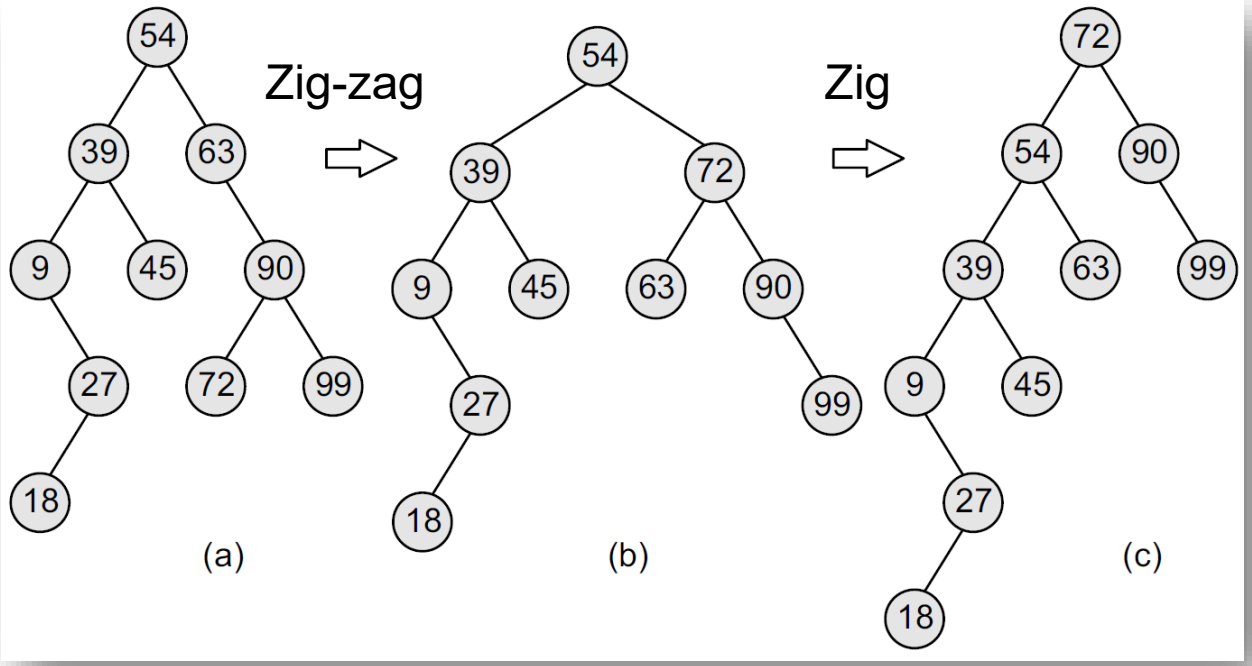


SPLAY Trees – Search

- For searching a particular node *N*
  - If the node is present in the splay tree
    1. A pointer to *N* is returned (return YES!)
    2. Splay the node
  - If the search is unsuccessful (the splay tree does not contain)
    1. A pointer to the null node is returned (return NULL)
    2. Splay the tree at the last non-null node reached during the search

Example

- Searching 81 for a given splay tree

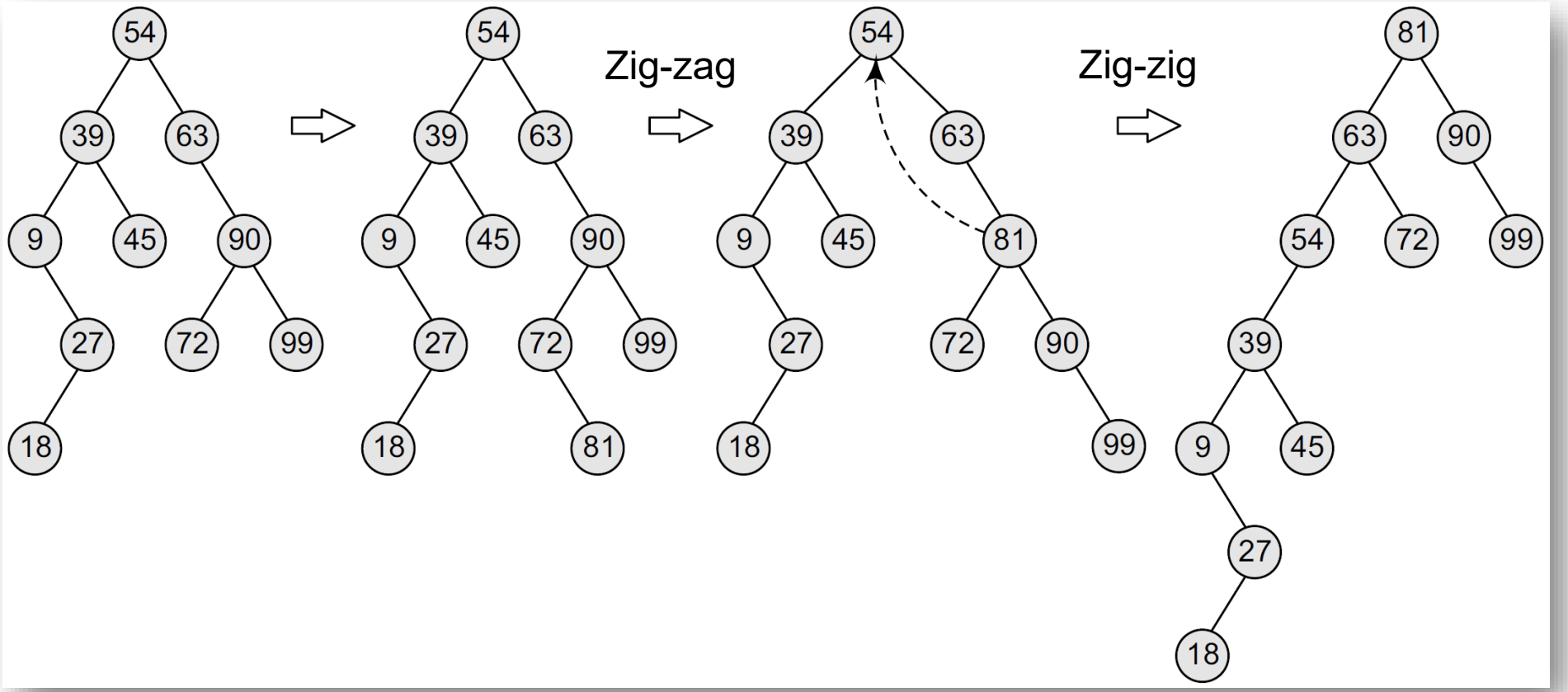


SPLAY Trees – Insertion

- The steps performed to insert a new node N in a splay tree can be given as follows
  - Search N in the splay tree
    1. If the search is successful, splay at the node N
    2. If the search is unsuccessful
       - Add the new node N in such a way that it replaces the NULL pointer reached during the search by a pointer to a new node N
       - Splay the tree at N

Example

- Insert 81 into a given Splay tree

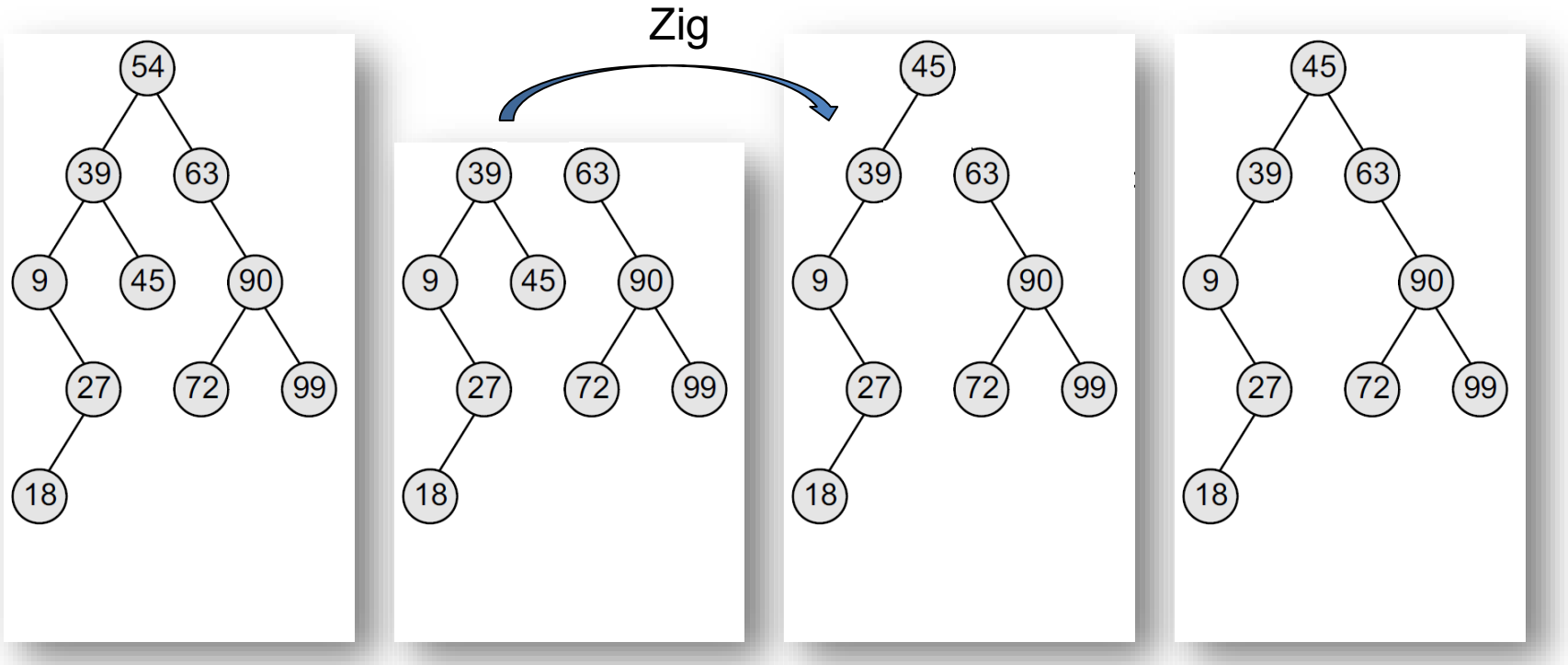


SPLAY Trees – Deletion

- To delete a node *N*, use the same method as with a binary search tree
  - Search for *N* that has to be deleted
    - If the search is **unsuccessful**, splay the tree at the last non-null node encountered during the search
    - If the search is **successful** and *N* is **not the root** node
      a) Delete *N* and replace *N* by an appropriate node
      b) Splay the parent of *N* to the top of the tree
    - If the search is **successful** and *N* is the **root** node
      a) Delete the node *N*
      b) Two sub trees are then joined
        - Splay the largest item *S* in left sub tree
        - Set the right sub tree to be the right sub tree of root *S*

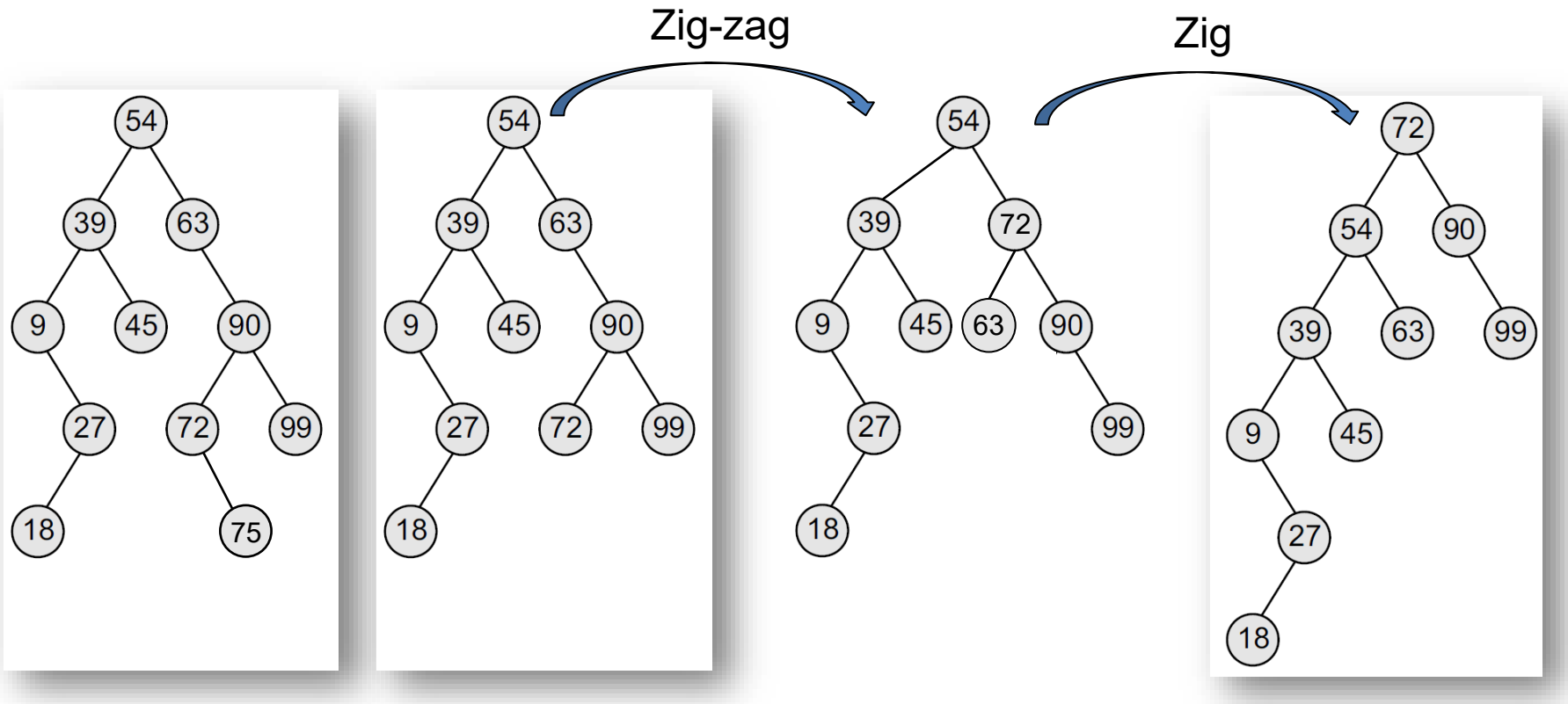
Example.

- Delete 54 from a given Splay tree



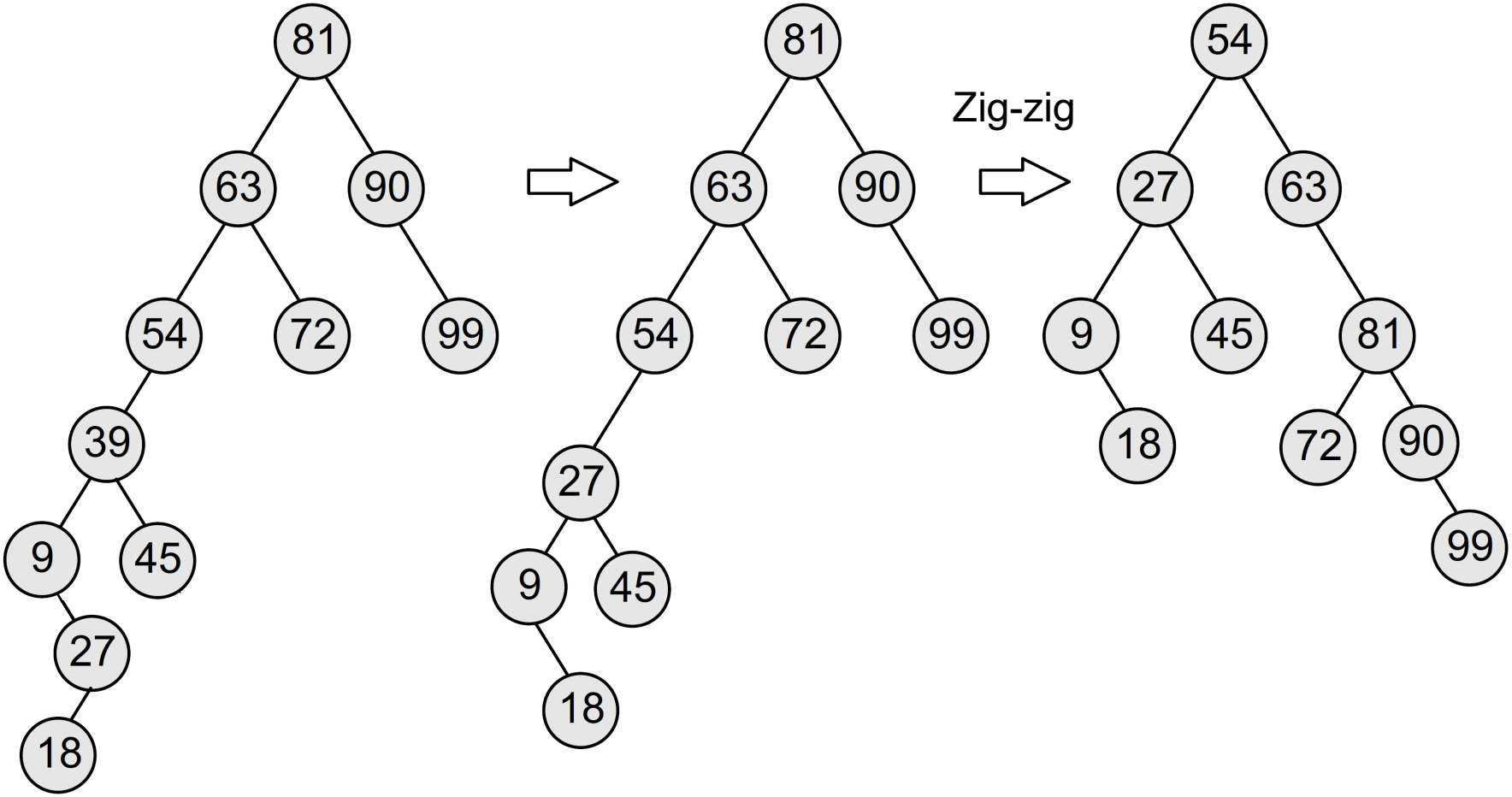
Example..

- Delete 75 from a given Splay tree



Example…

- Delete 39 from a given Splay tree



Pros and Cons.

- The advantages of using a splay tree are:
  - Splay tree is a **self-balancing** and a **self-optimizing** data structure
    - The frequently accessed nodes are moved closer to the root so that they can be accessed quickly
    - It is particularly useful for implementing **caches** and garbage collection algorithms (memory management)
  - Splay trees are considerably simpler to implement than the other self-balancing binary search trees (**red-black trees** or **AVL trees**), while their average case performance is just as efficient
  - Splay trees minimize memory requirements as they do not store any book-keeping data

Pros and Cons..

- The demerits of splay trees include:
  - While sequentially accessing all the nodes of a tree in a sorted order, the resultant tree becomes completely unbalanced
  - For uniform access, the performance of a splay tree will be considerably worse than a somewhat balanced simple binary search tree

Questions?

kychen@mail.ntust.edu.tw