The Disjoint-Set data structure, also known as Union-Find, manages elements in non-overlapping subsets. It's primarily used for graph connectivity problems, offering two operations: Union, to merge sets, and Find, to identify a set's leader. This structure excels in tasks involving elements initially represented as separate sets, enabling efficient set combination and connectivity checks between elements. It's defined by disjoint sets (non-intersecting) and the Union-Find algorithm, which determines subset membership and merges subsets. The Disjoint-Set structure is crucial for problems requiring frequent set merges and connectivity queries, due to its near-constant-time operations.

## **How to Combine Two Sets?**

Whenever we have been given a problem that contains n elements represented as separate sets initially and we need to the perform following operations:

1. Combine two sets.
2. Find the connectivity of two given elements i.e. whether they belong to the same set or not. Then it is advisable to use the Disjoint-Set data structure to pose an efficient solution. Disjoint Set Union is sometimes also referred to as Union-Find because of its two important operations -

Disjoint Set: Two sets are called disjoint set if the intersection of two sets is ϕ i.e. NULL. A Disjoint-Set data structure that keeps records of a set of elements partitioned into several non-overlapping (Disjoint) subsets.

Union Find: Union-Find algorithm performs two very useful operations -

Find: To find the subset a particular element 'k' belongs to. It is generally used to check if two elements belong to the same subset or not.

Union: It is used to combine two subsets into one. A union query, say Union(x, y) combines the set containing element x and the set containing element y."

## **Example of Disjoint Set**

In this example, we are going to see how the union queries (series of Union operations) work in a Disjoint Set.

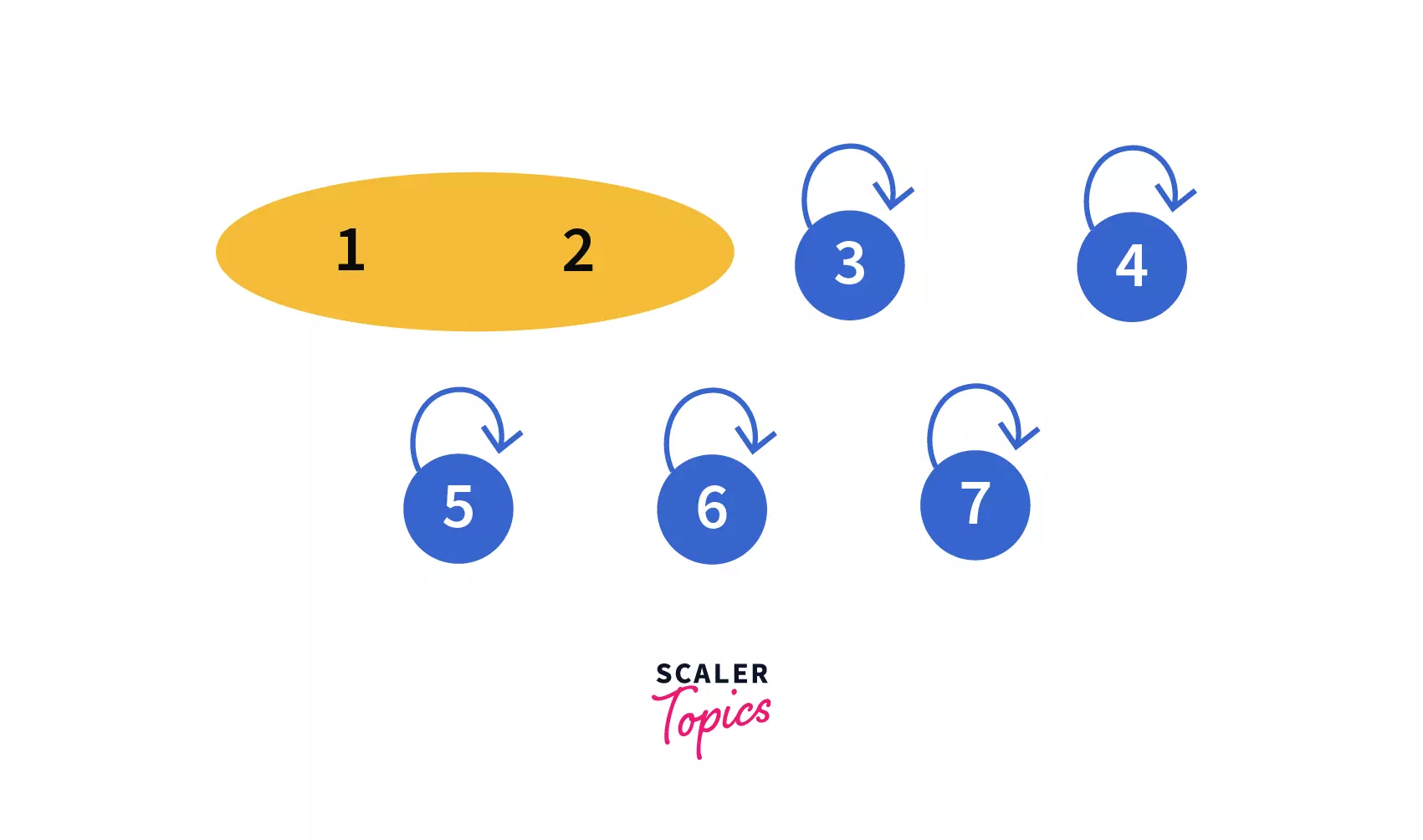
Let's assume we have 7 nodes initially, we will store them in the form of trees.Where each tree corresponds to one set and the root of the tree will be the parent/leader of the set.



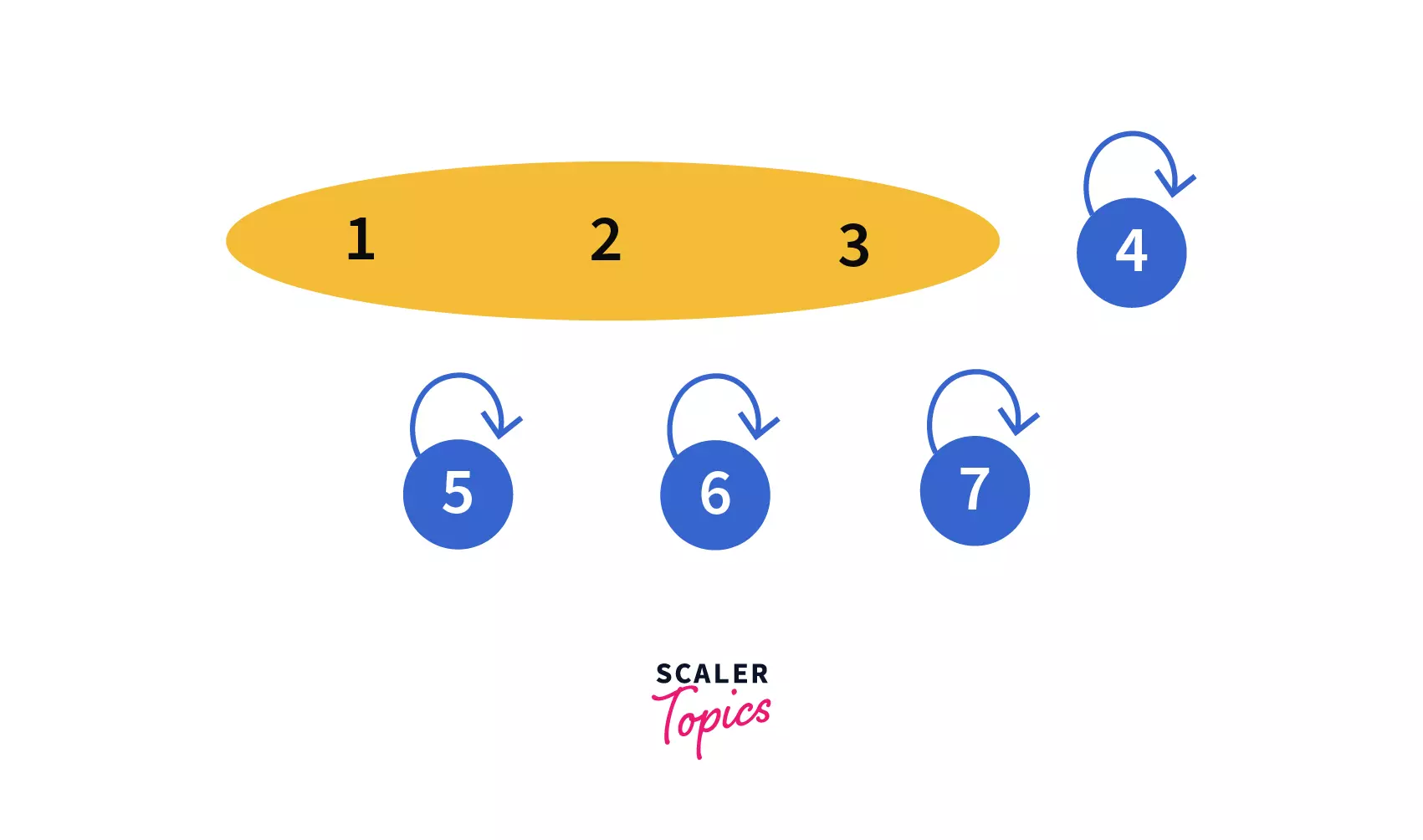
At first, you can see that all the seven nodes are parents of themselves. In simple words we can say, we have seven different trees containing 1 element each i.e. the root of the tree.

Now we will perform the following union queries one by one on them --

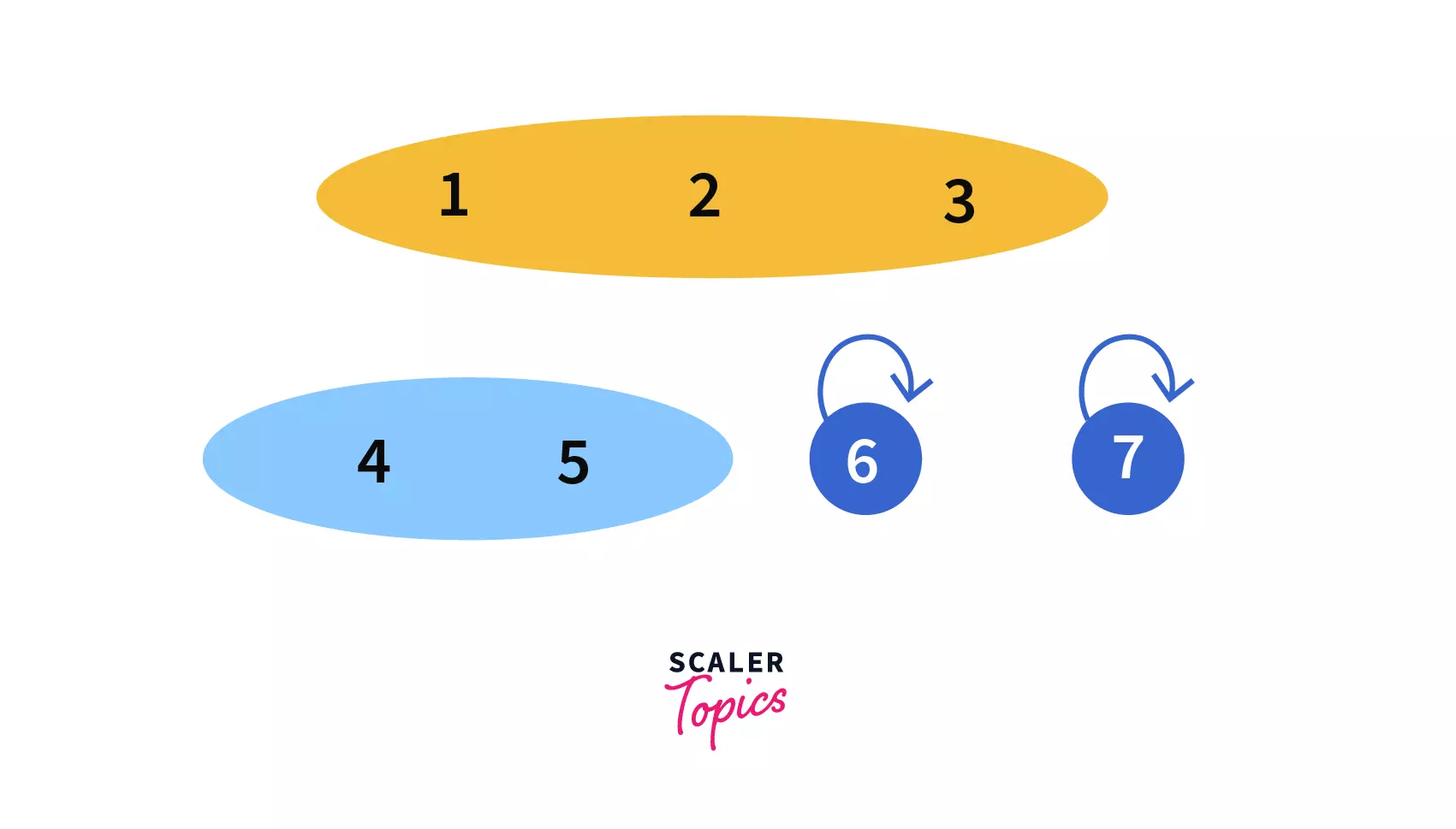
* Union(1, 2)
* Union(2, 3)
* Union(4, 5)
* Union(6, 7)
* Union(5, 6)
* Union(2, 6)
* In our first query *Union(1, 2)* we need to join two sets i.e. *1 and 2* into one.  
  After performing the query we can see that, 1 and 2 are in the same set with parent/leader as 1 so our disjoint set will look like -



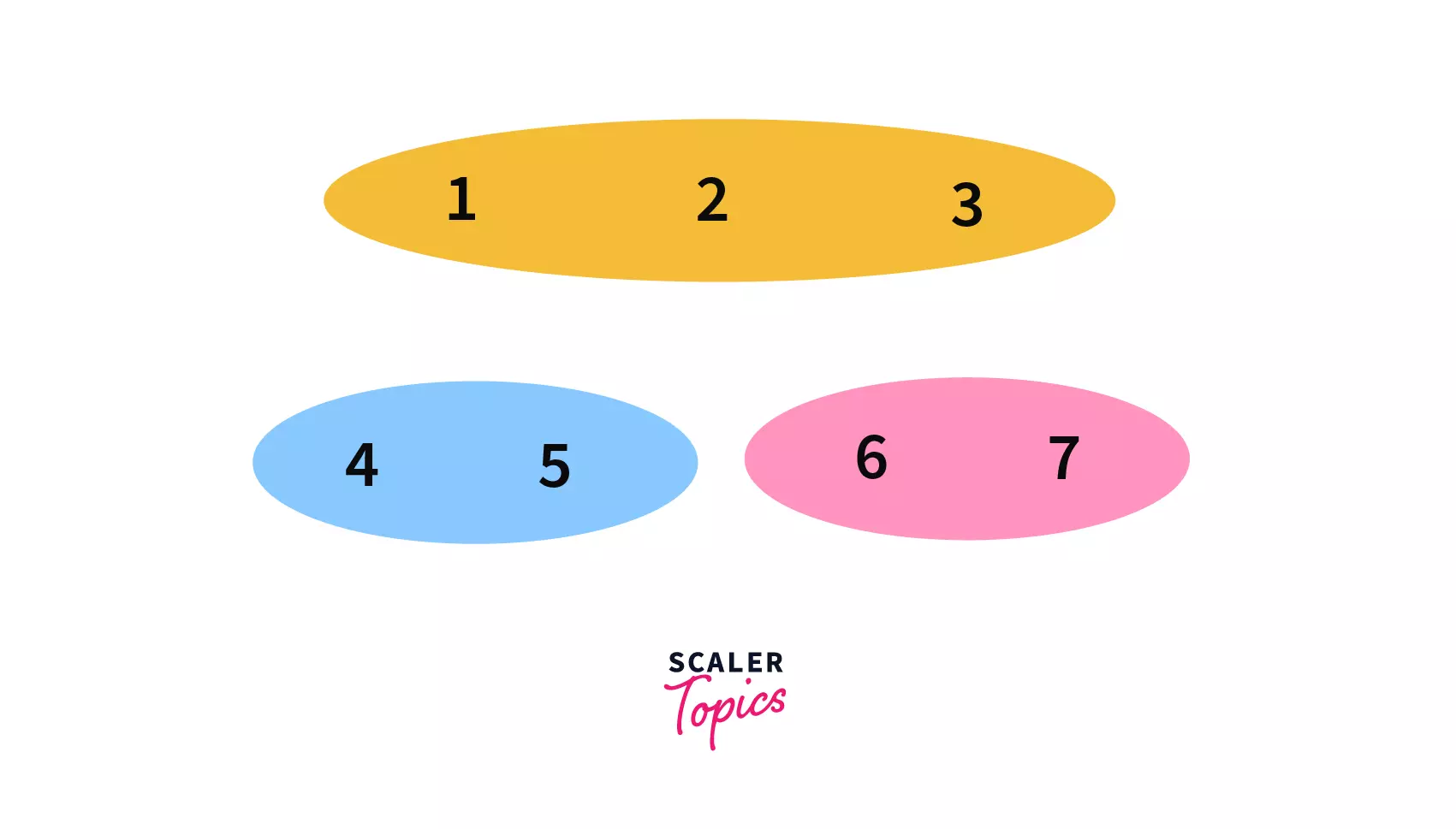
* In our second query, *Union(2, 3)* we need to join the sets which contain elements 2 and 3. After performing the query we can see that, 1, 2, and 3 are clubbed into one set so our disjoint set will look like -



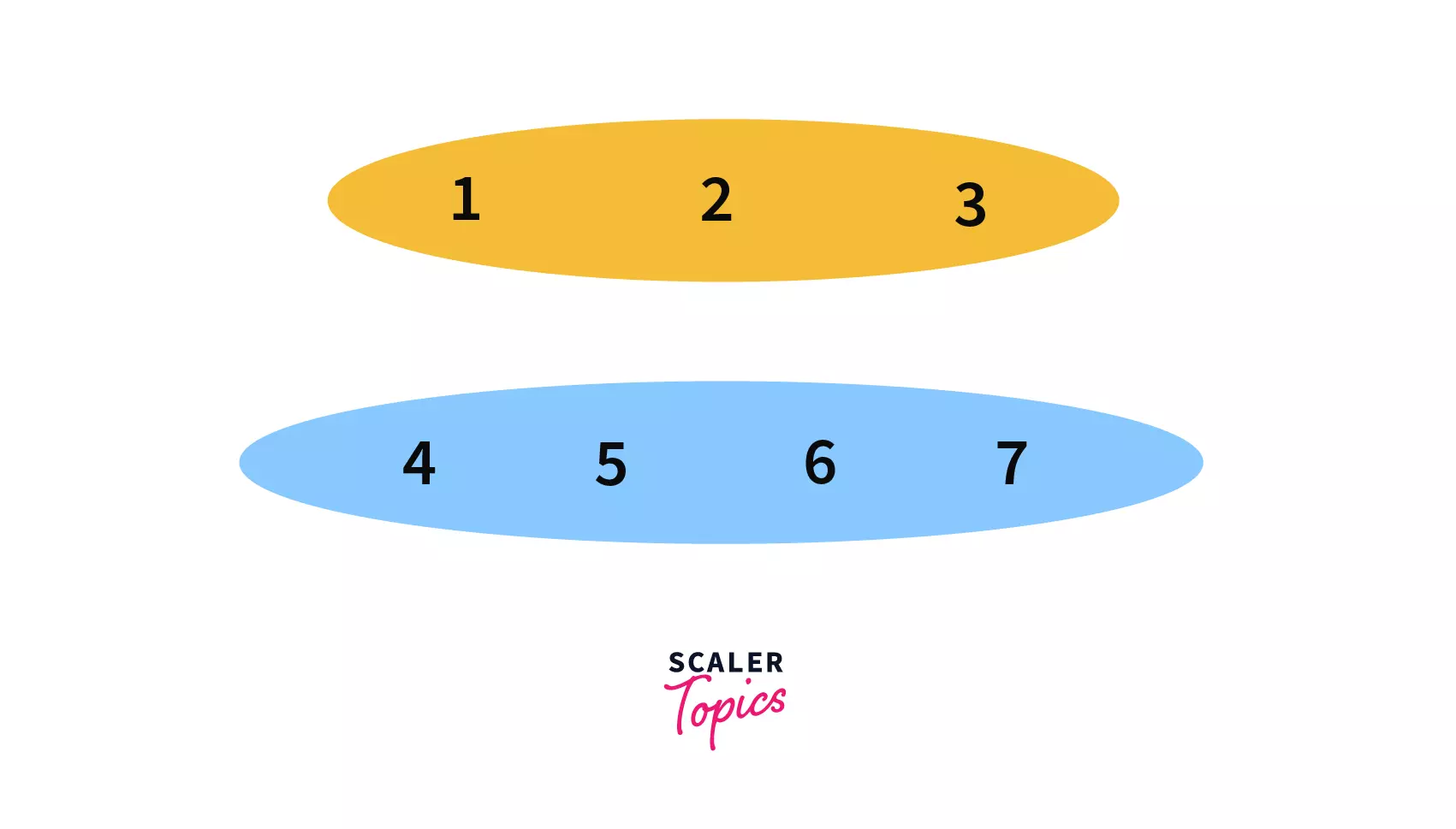
* In our third query, *Union(4, 5)* we need to join the sets which contain the elements 4 and 5. After performing the query we can see that, 4 and 5 are in the same set now with parent/leader as 4, making our disjoint set like -



* In our fourth query, *Union(6, 7)* we need to join the sets which contain the elements 6 and 7. After performing the query we can see that, 6 and 7 are in the same set now with parent/leader as 6, making our disjoint set like -



* In our fifth query, *Union(5, 6)* we need to join the sets which contain the elements 5 and 6. After performing the query we can see that, 6 and 7 are in the same set now with parent/leader as 4, making our disjoint set like -



* In our sixth query, *Union(2, 6)* we need to join the sets which contain elements 2 and 6. After performing the query we can see that, all the elements i.e. 1 to 7 are in the same set now with parent/leader as 1, making our disjoint set like

