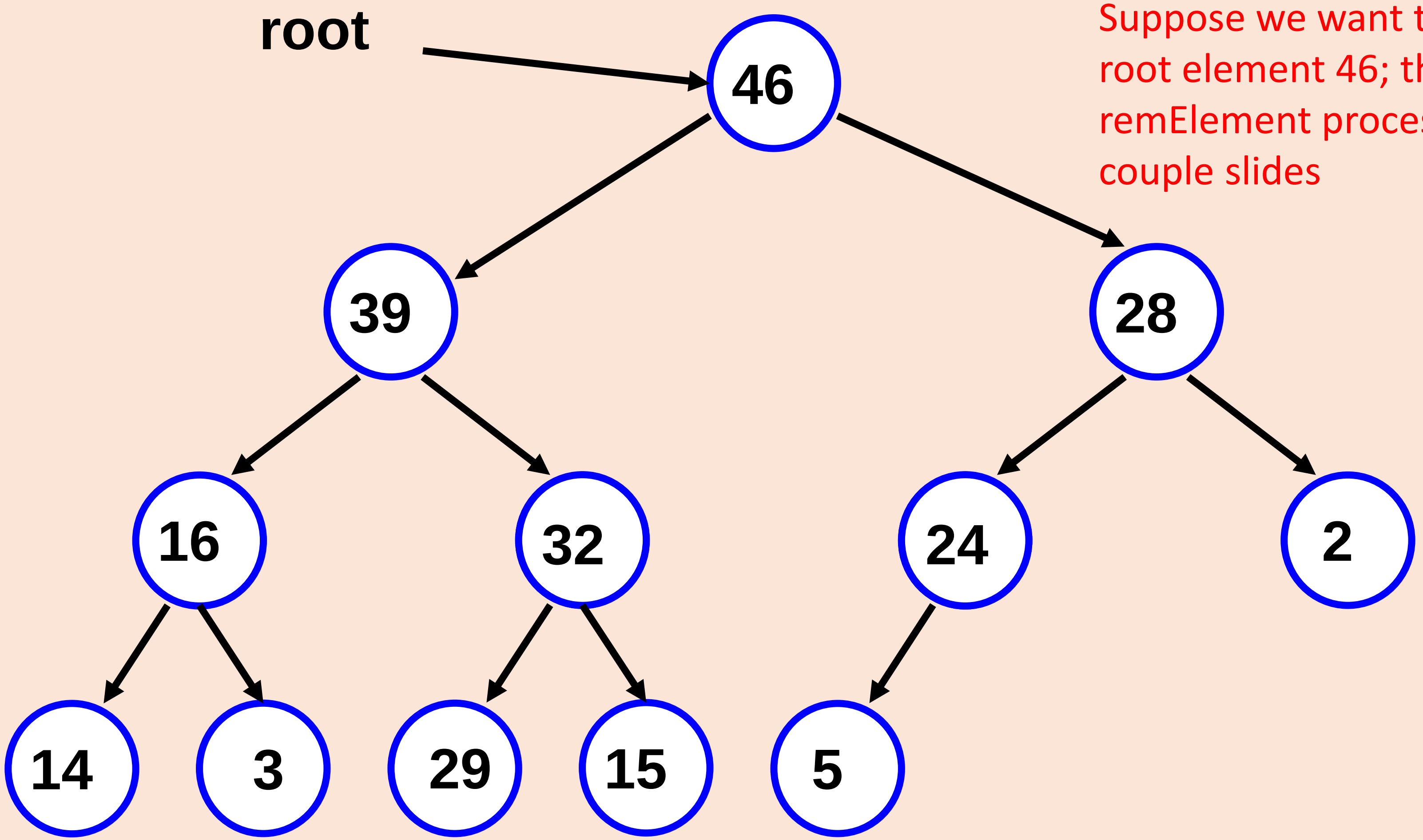
Heapify (ReheapDown) or Dequeue process

Dequeue

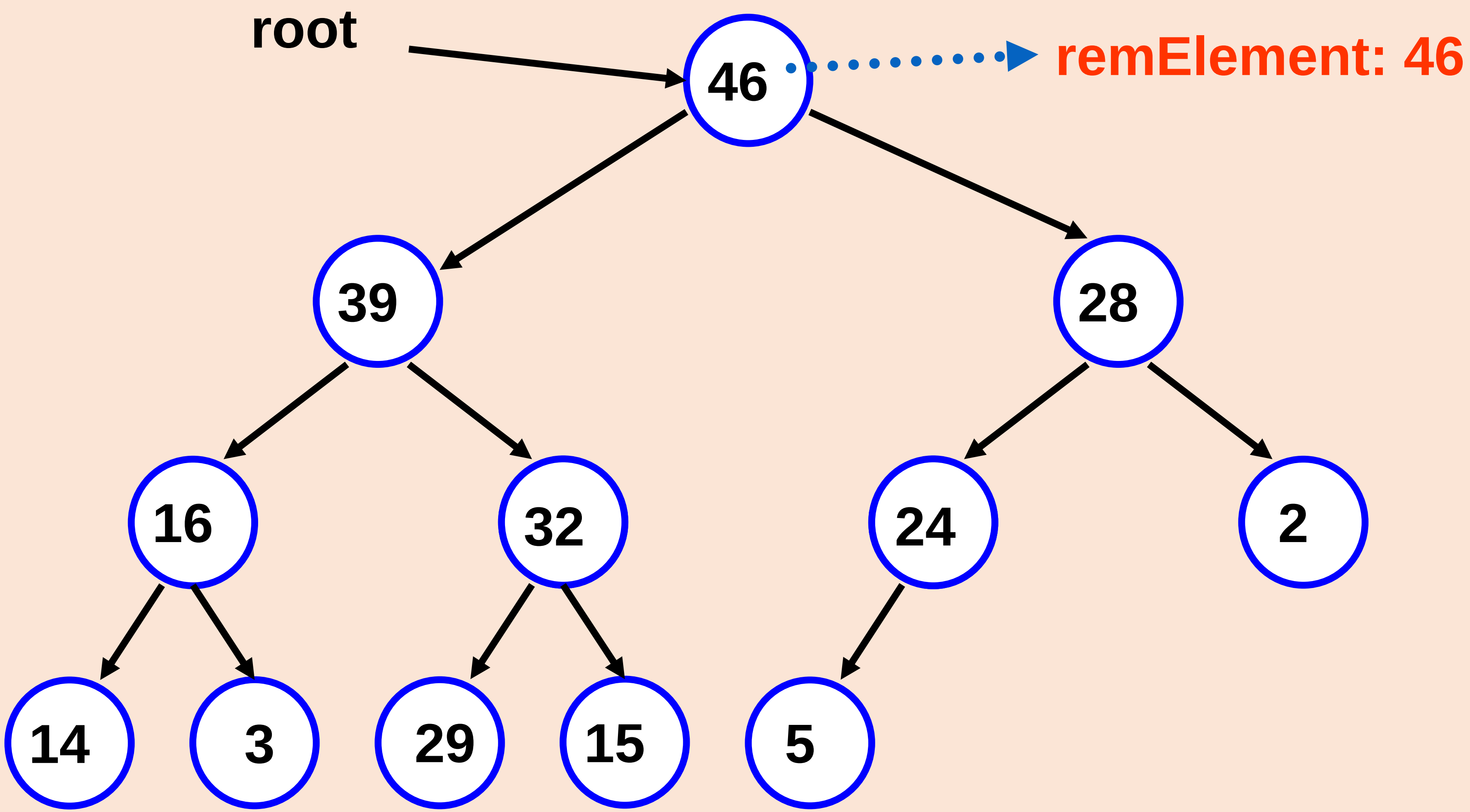
- Dequeuing (removing) the object with the greatest value appears to be a O(1) operation
- However, after removing the object, we must turn the resultant structure into a heap again, for the next dequeue.
- Fortunately, it only takes O(lg n) time to turn the structure back into a heap again (which is why dequeue in a heap is a O(lg n) operation

Dequeue (cont.)

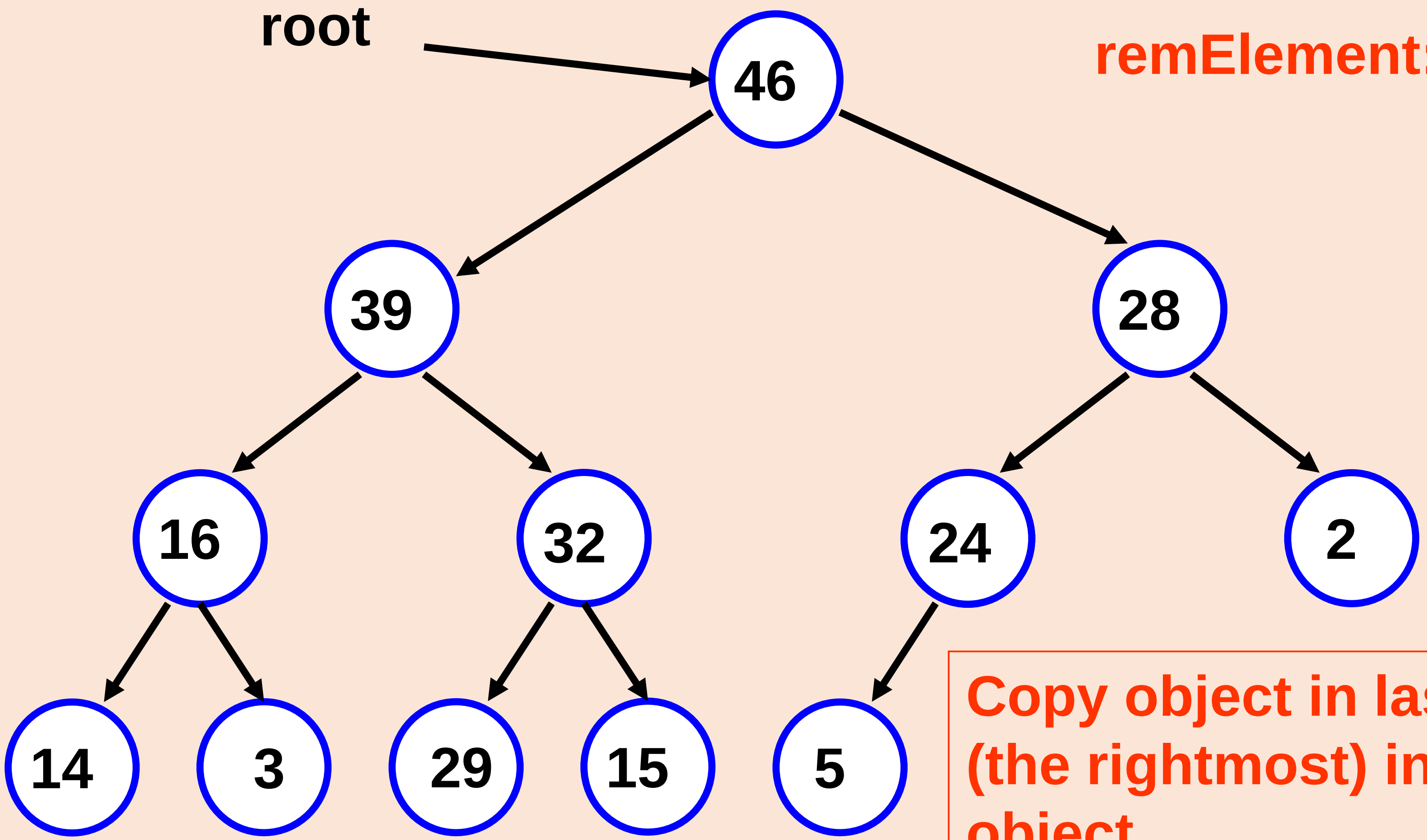


Suppose we want to remove the root element 46; the function remElement processes in next couple slides

Dequeue (cont.)



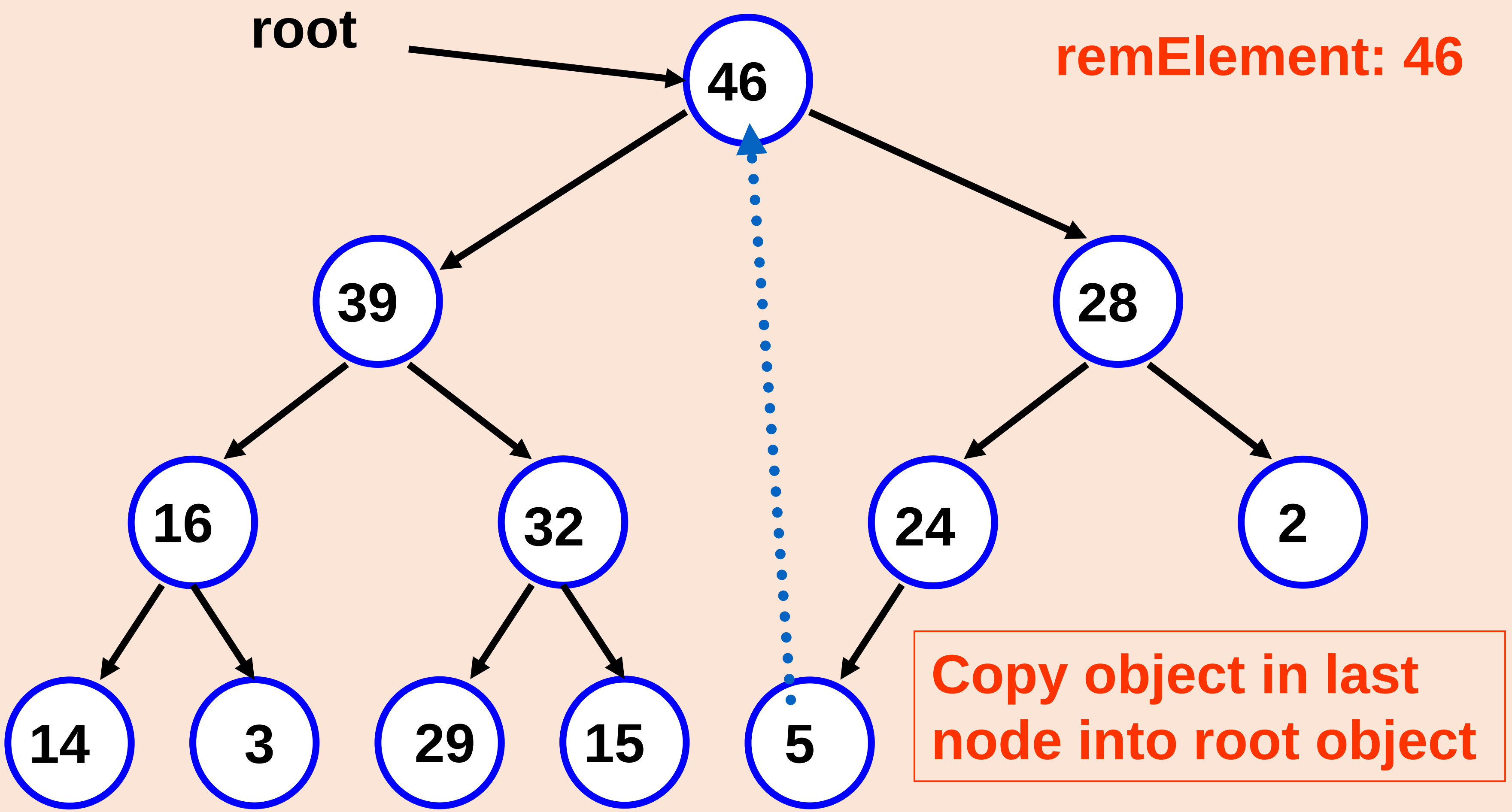
Dequeue (cont.)



remElement: 46

Copy object in last node (the rightmost) into root object

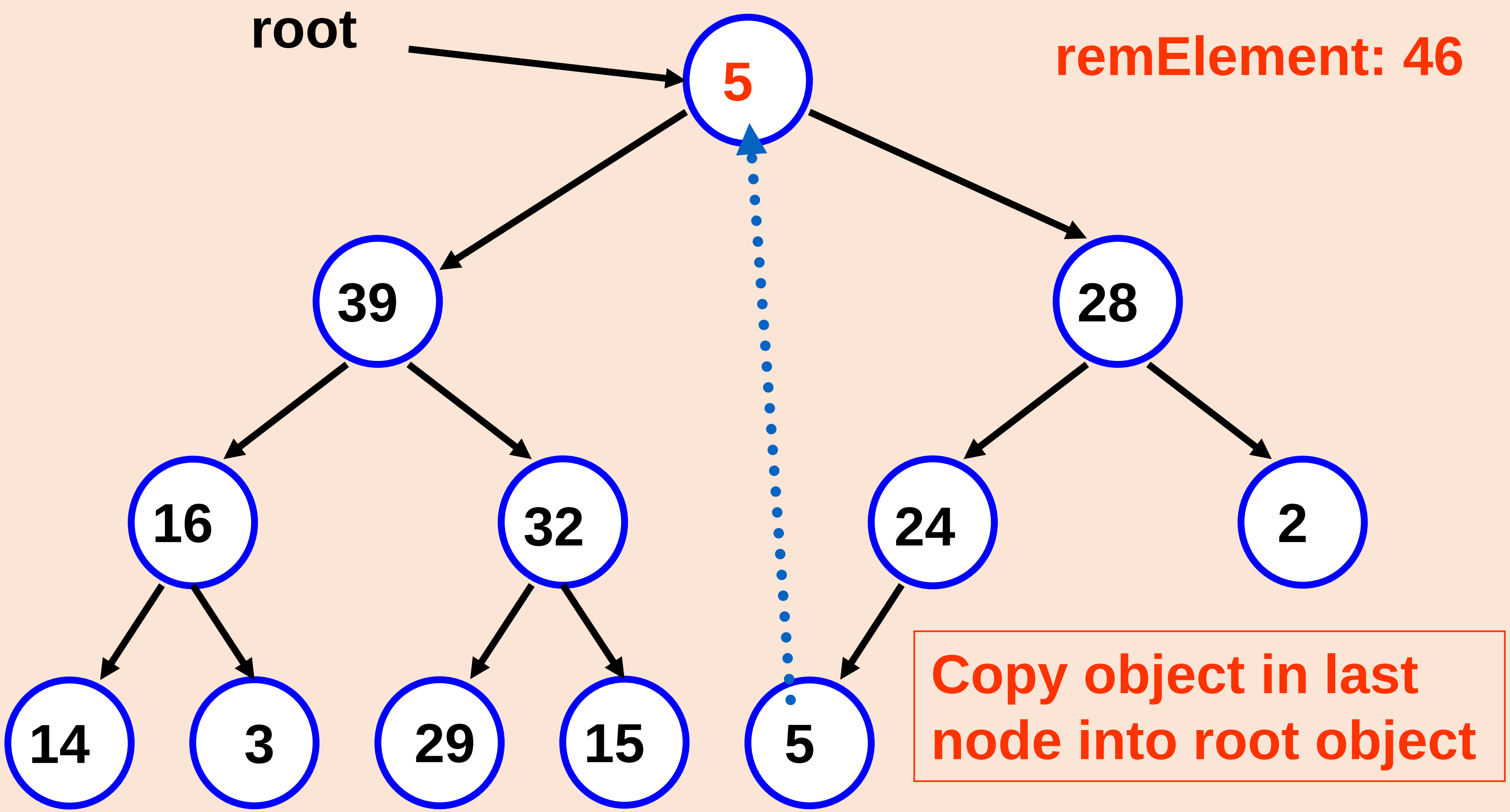
Dequeue (cont.)



remElement: 46

Copy object in last node into root object

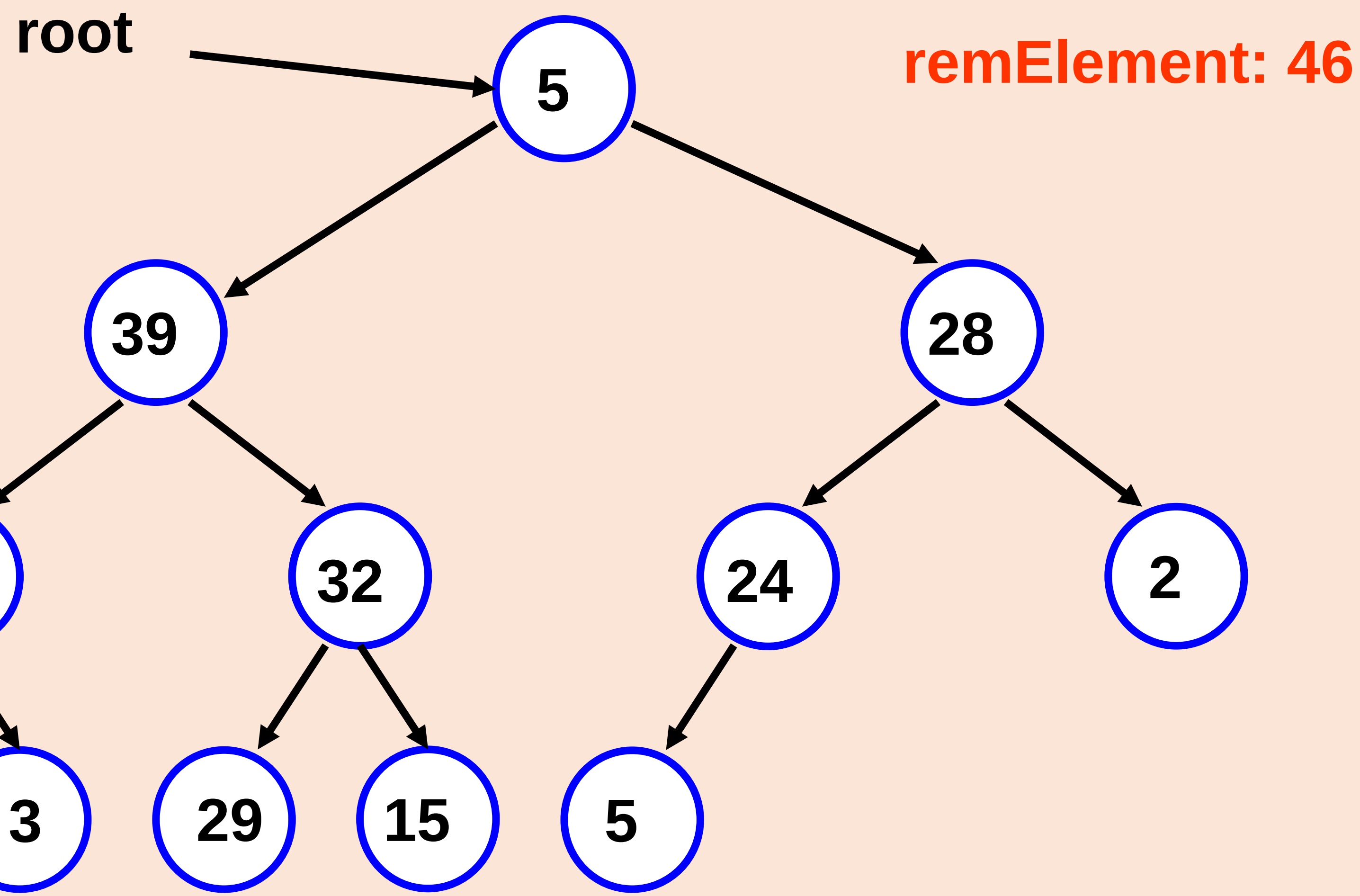
Dequeue (cont.)



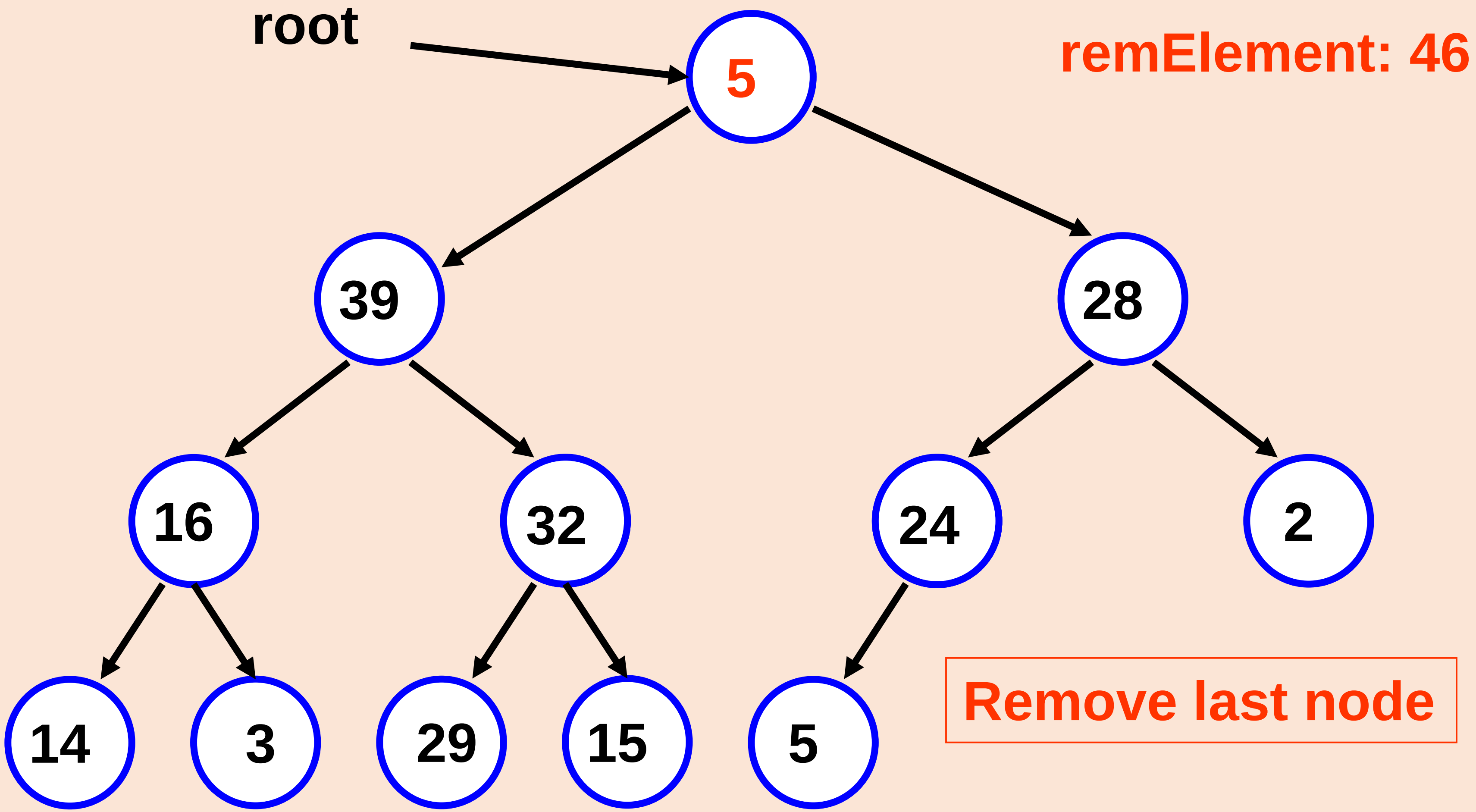
remElement: 46

Copy object in last node into root object

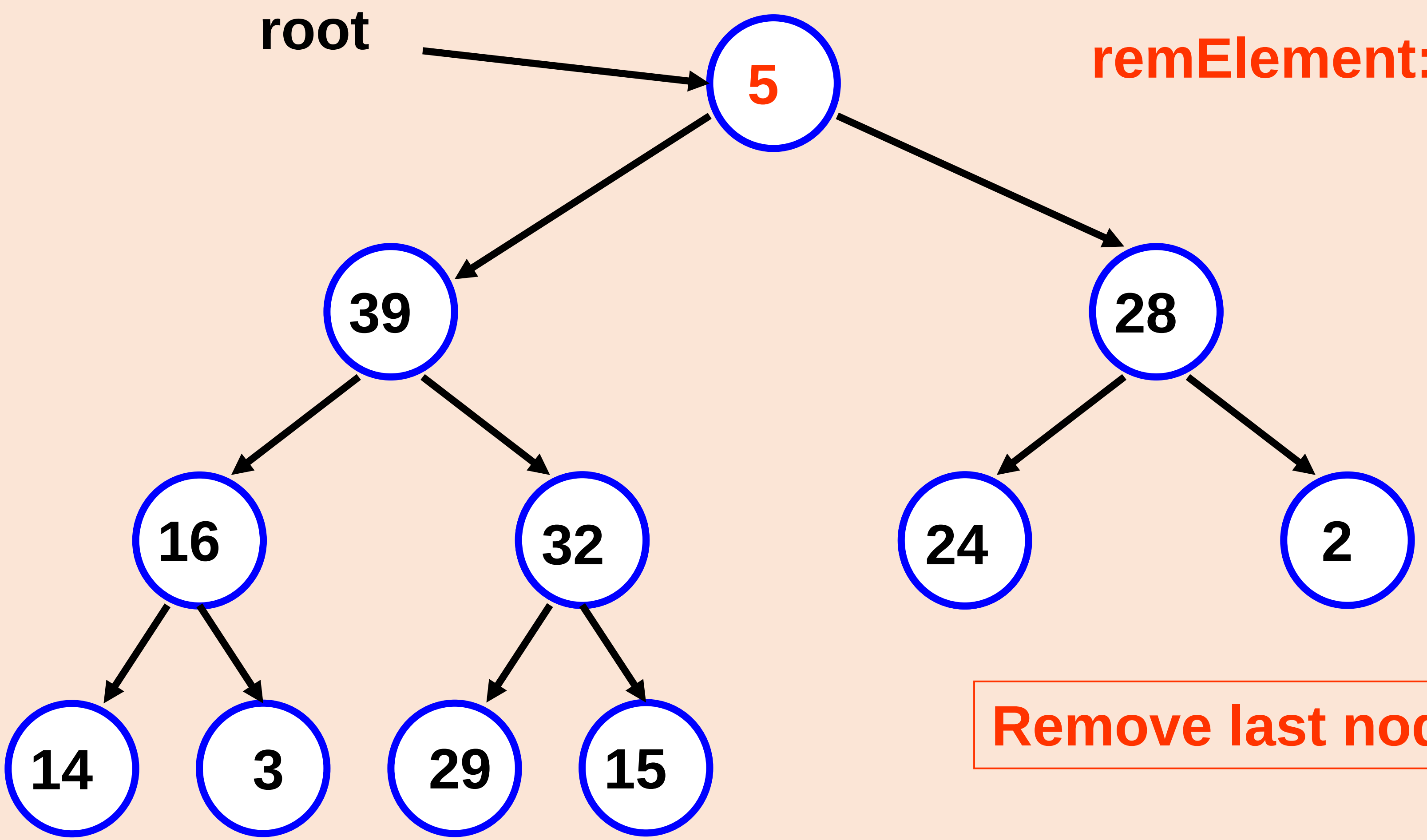
Dequeue (cont.)



Dequeue (cont.)



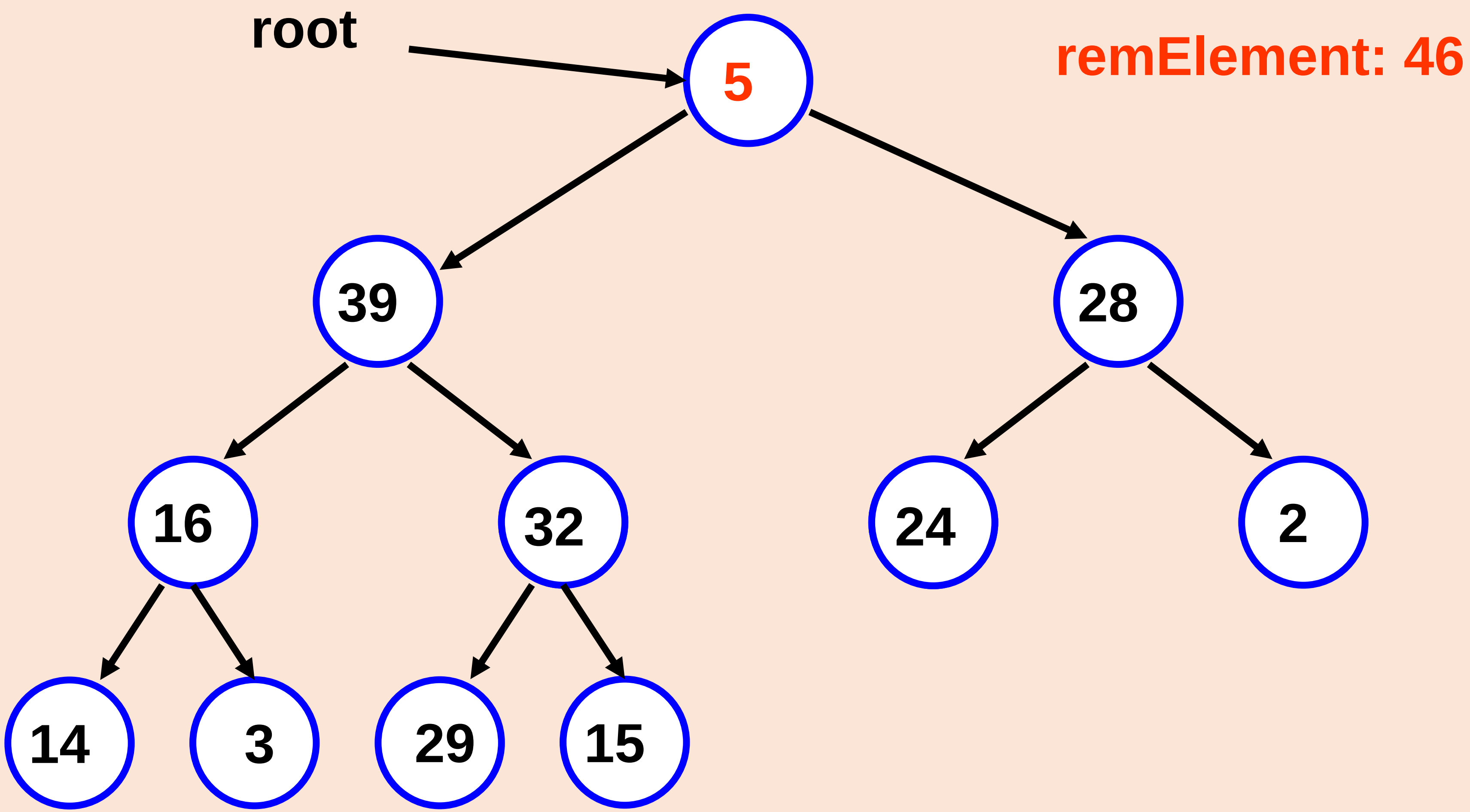
Dequeue (cont.)



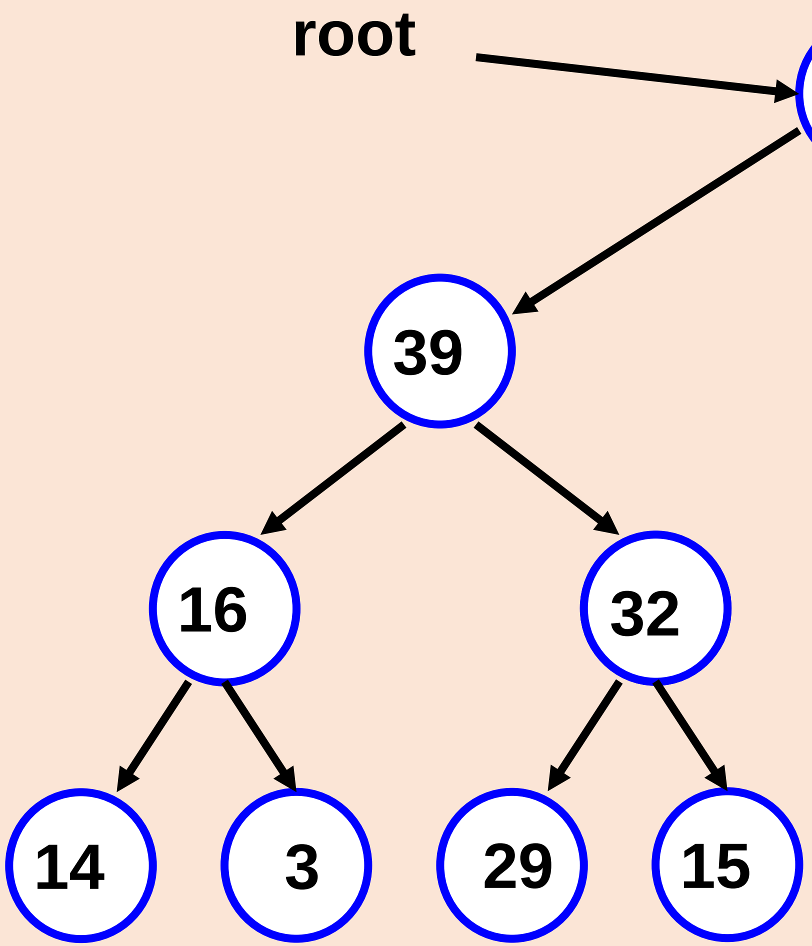
remElement: 46

Remove last node

Dequeue (cont.)



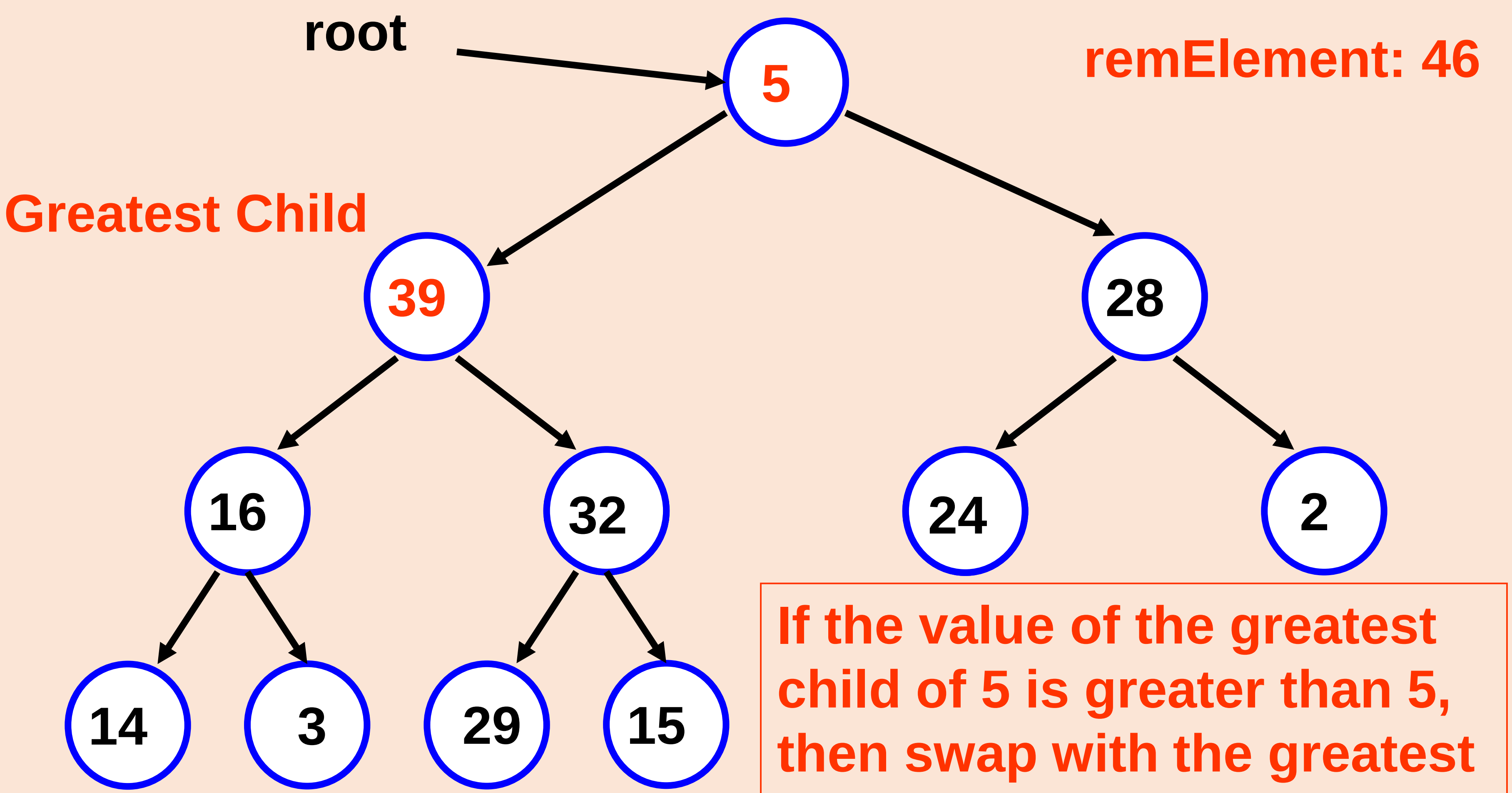
Dequeue (cont.)



remElement: 46

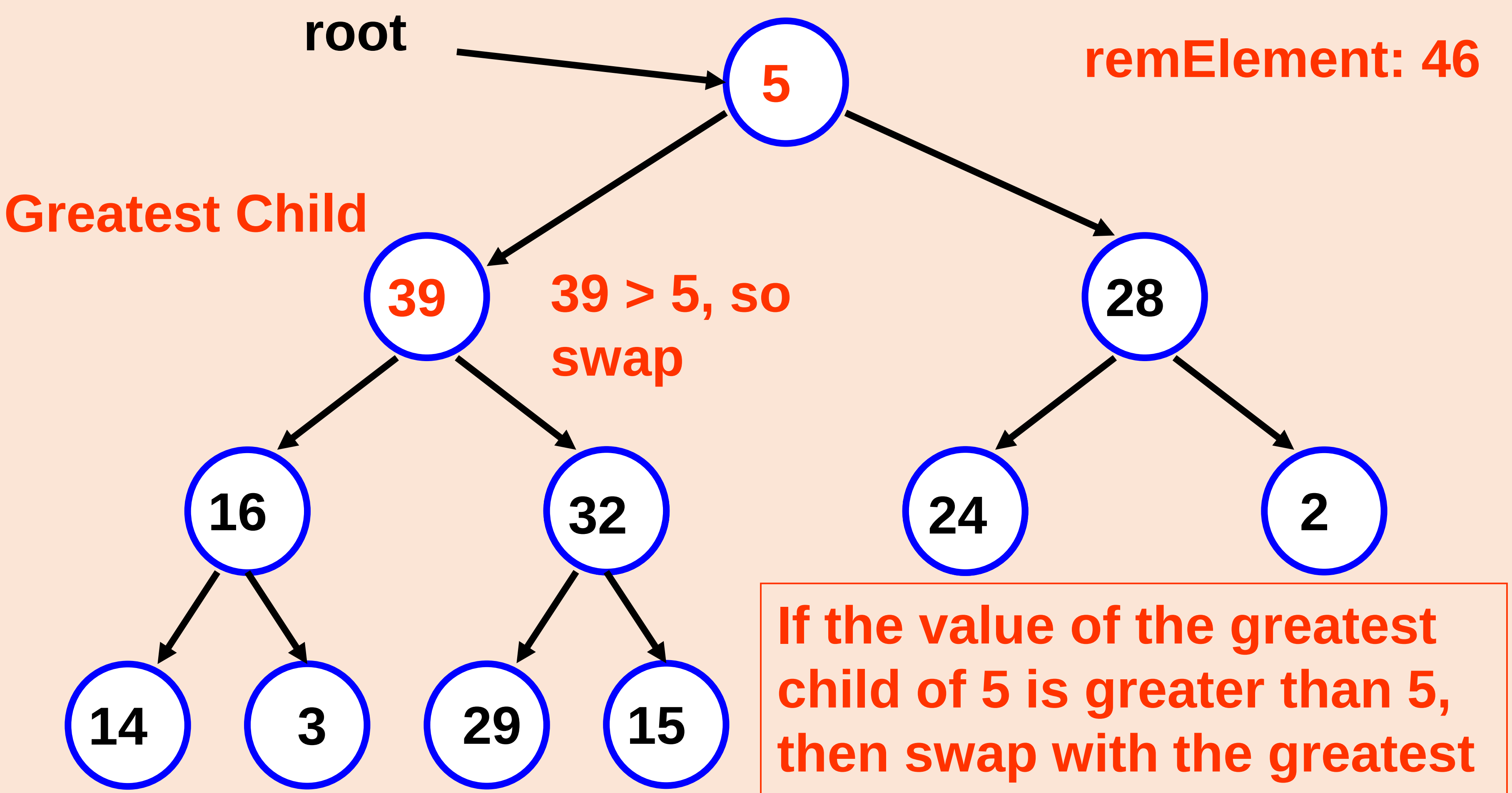
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



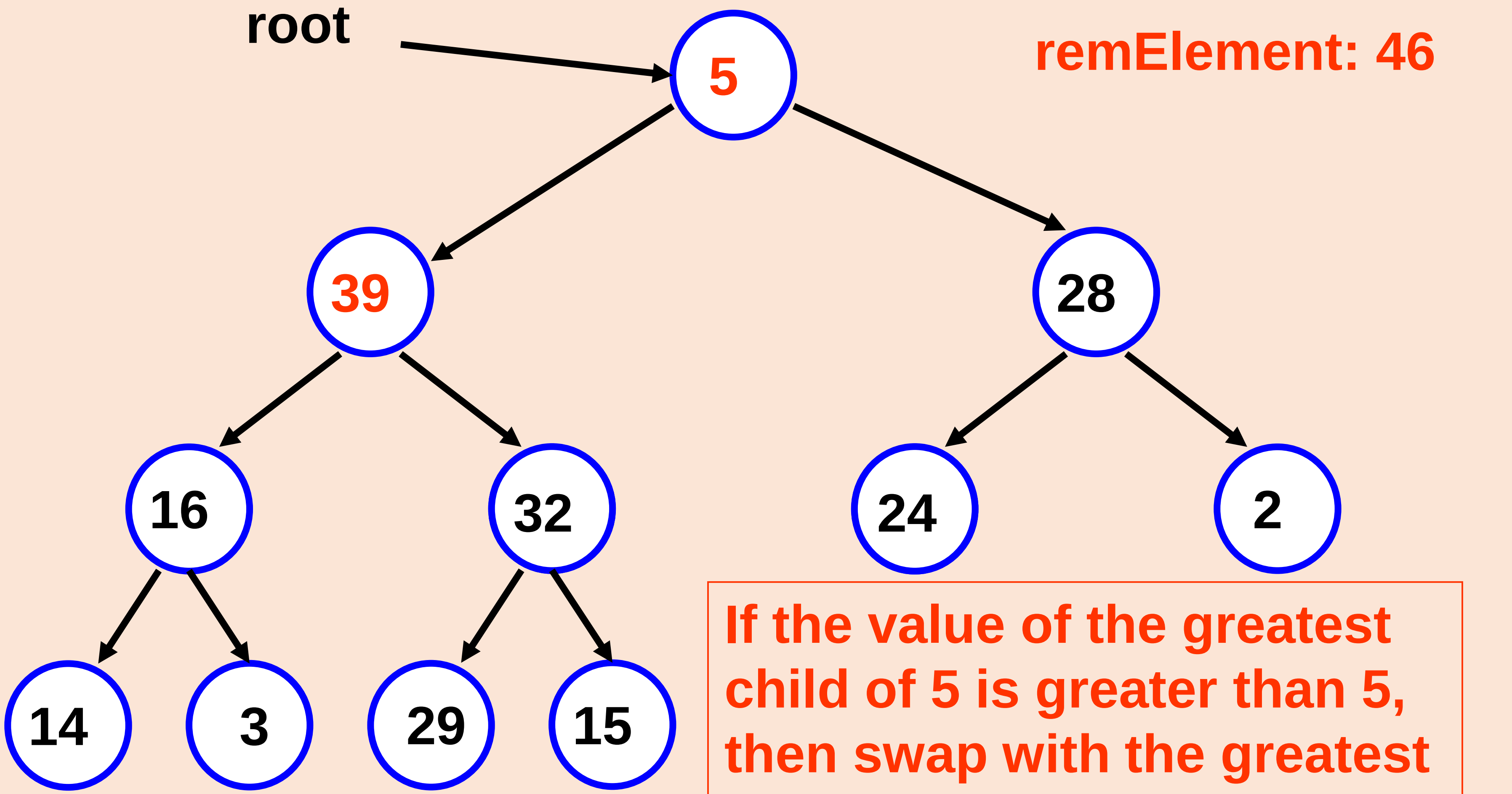
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



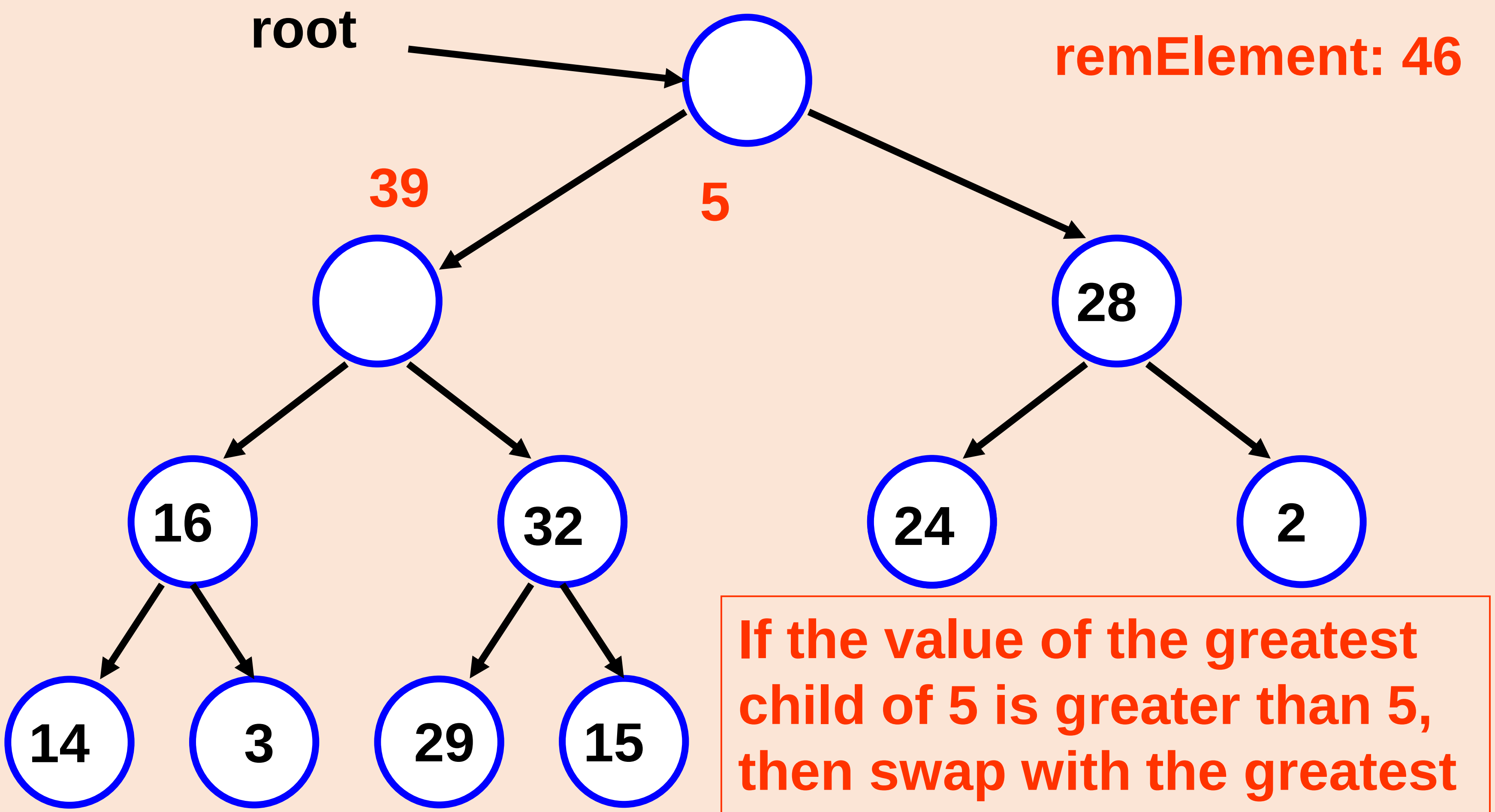
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



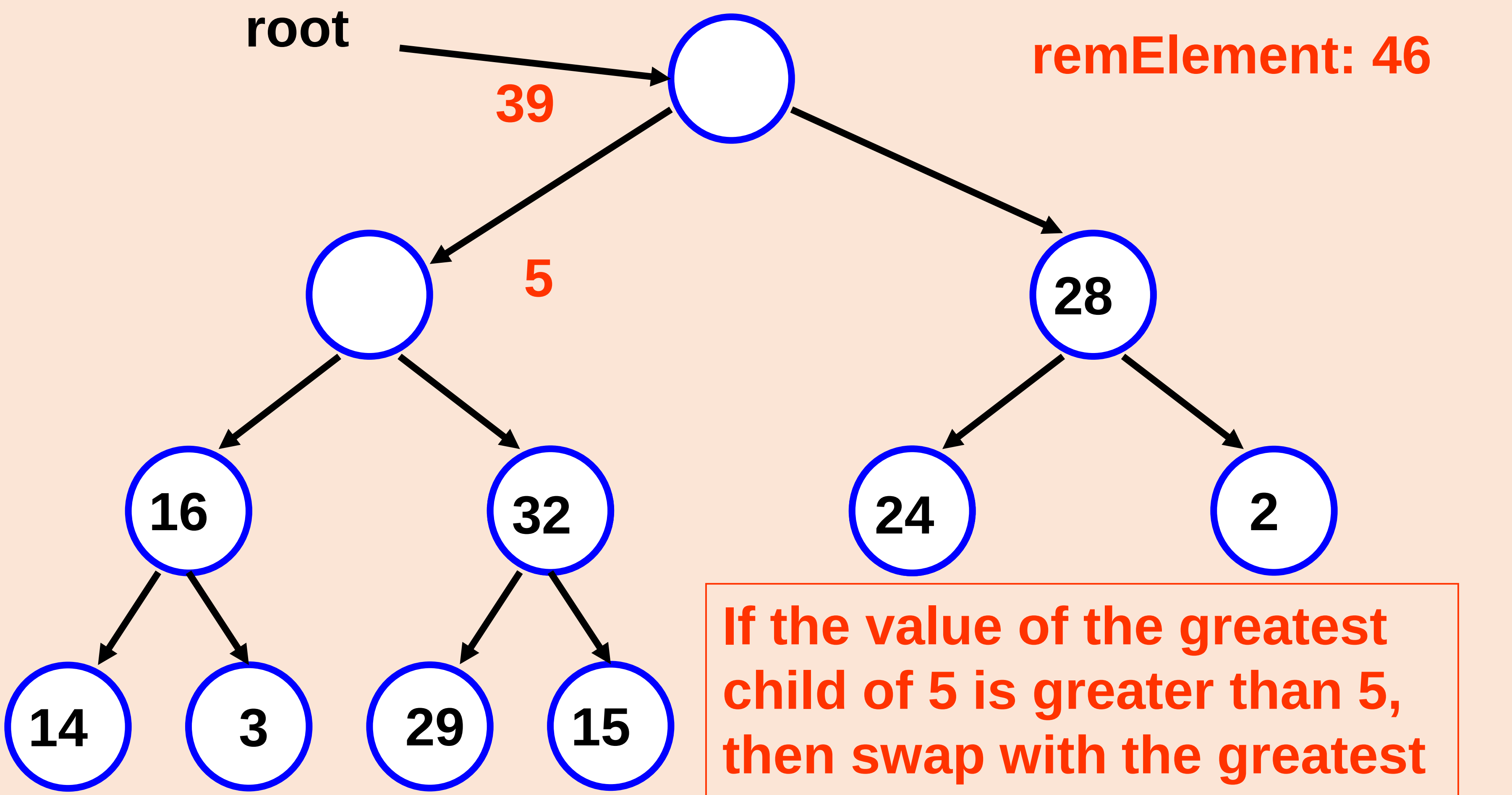
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

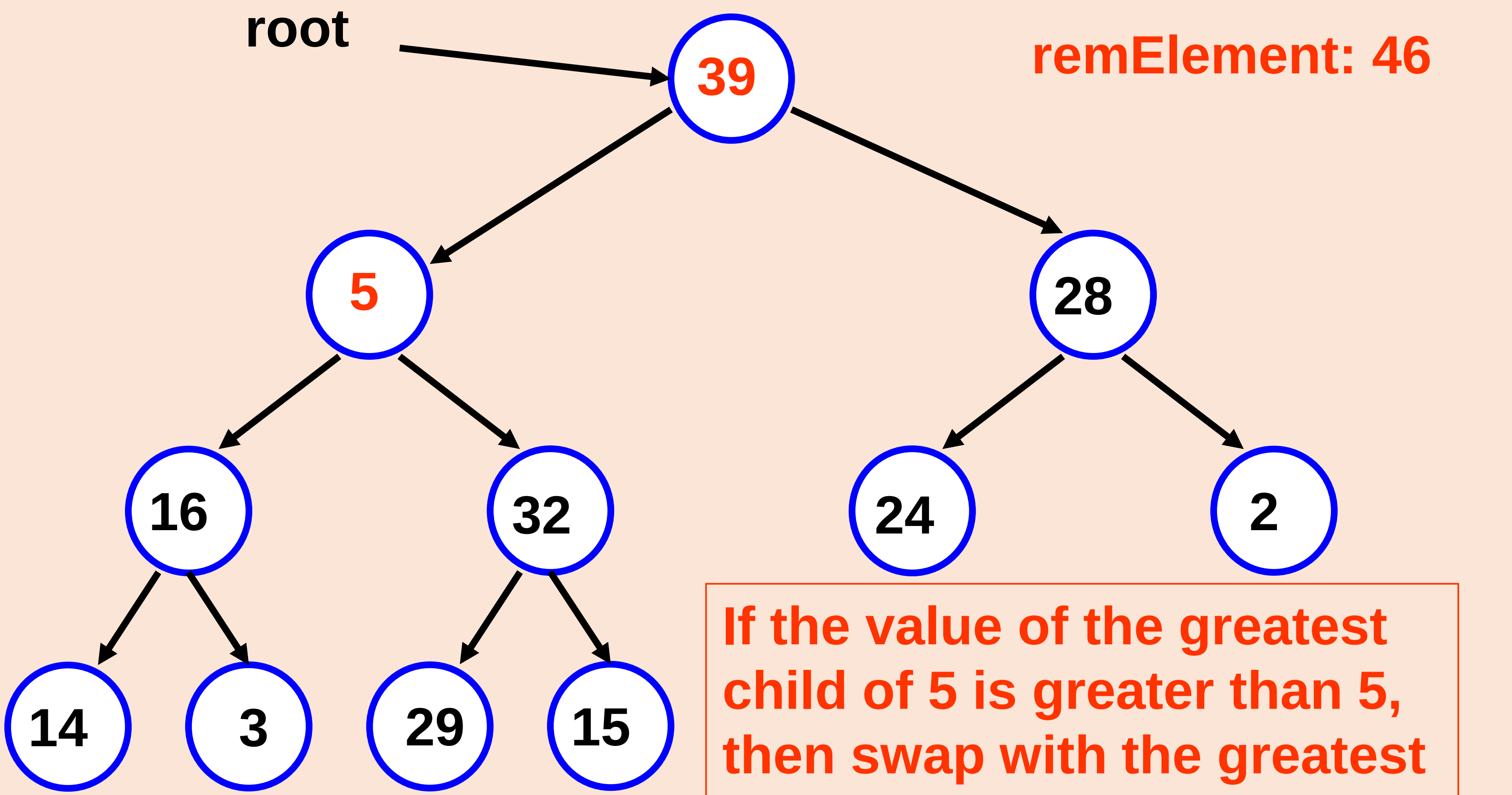
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

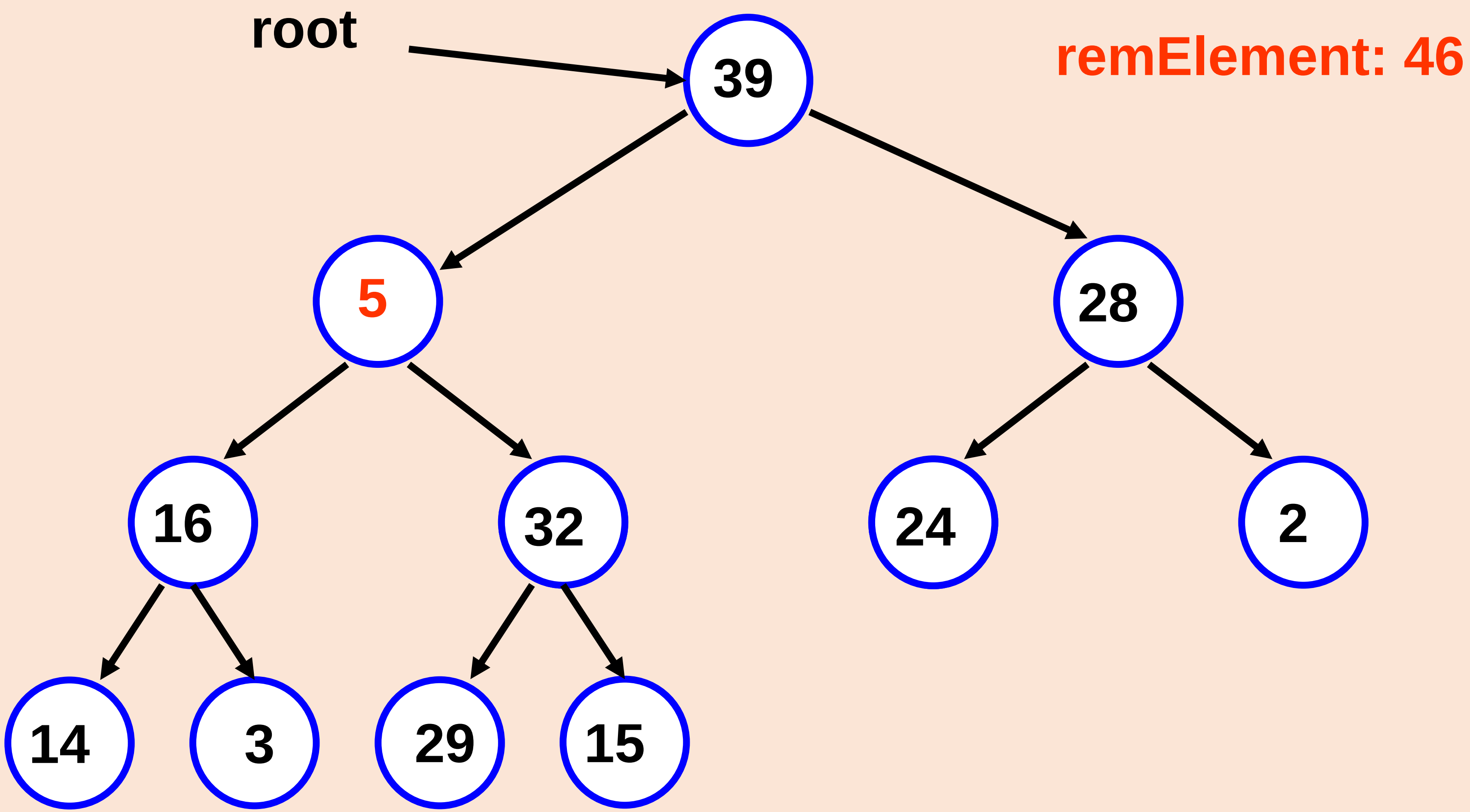
Dequeue (cont.)



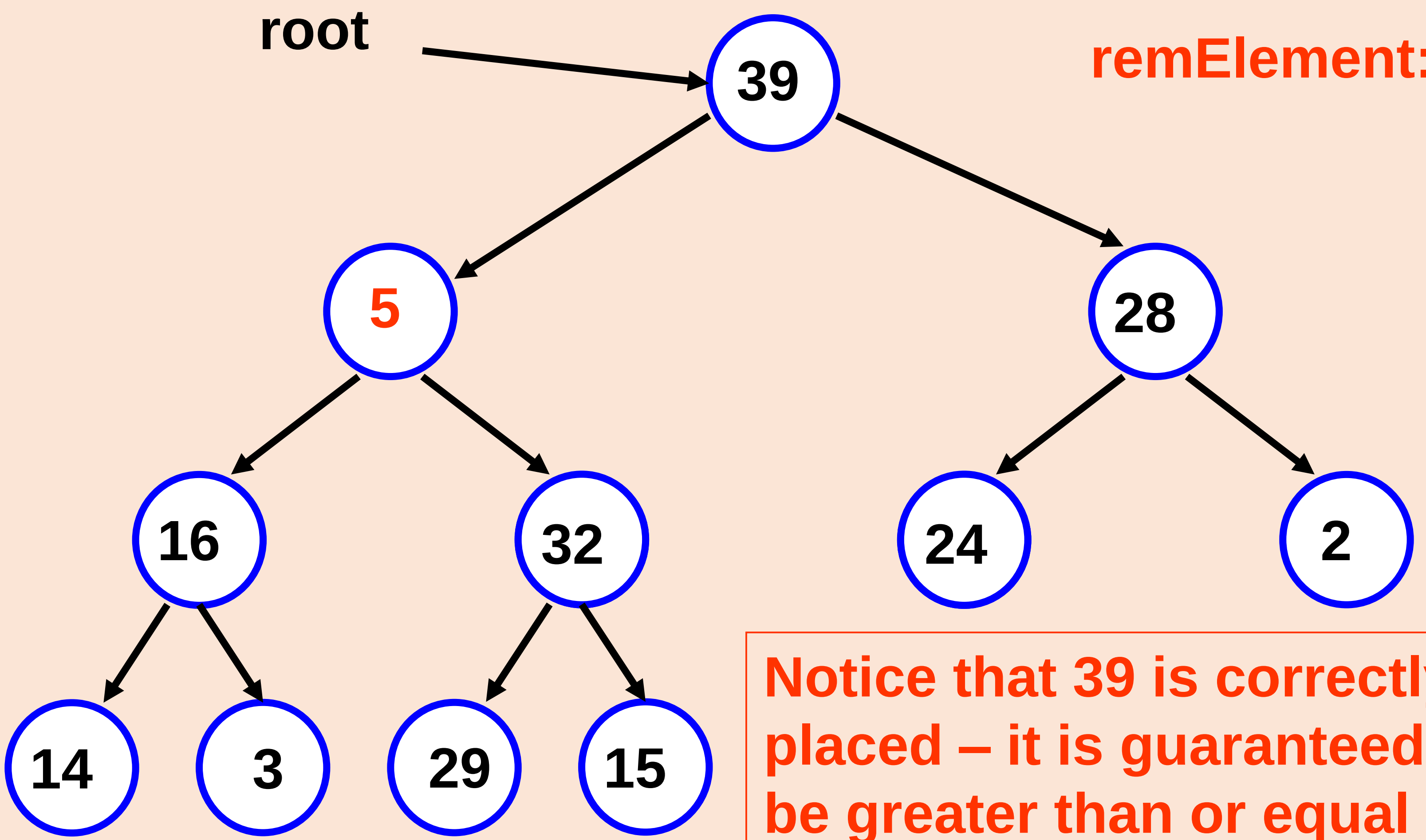
remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



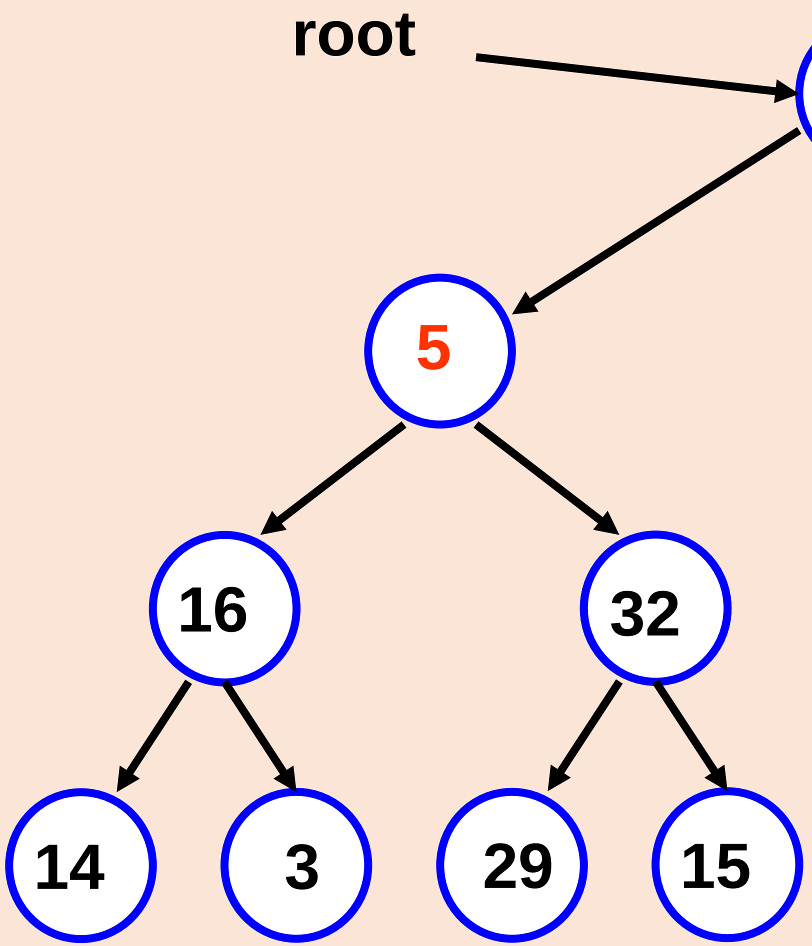
Dequeue (cont.)



remElement: 46

Notice that 39 is correctly placed – it is guaranteed to be greater than or equal to its two children

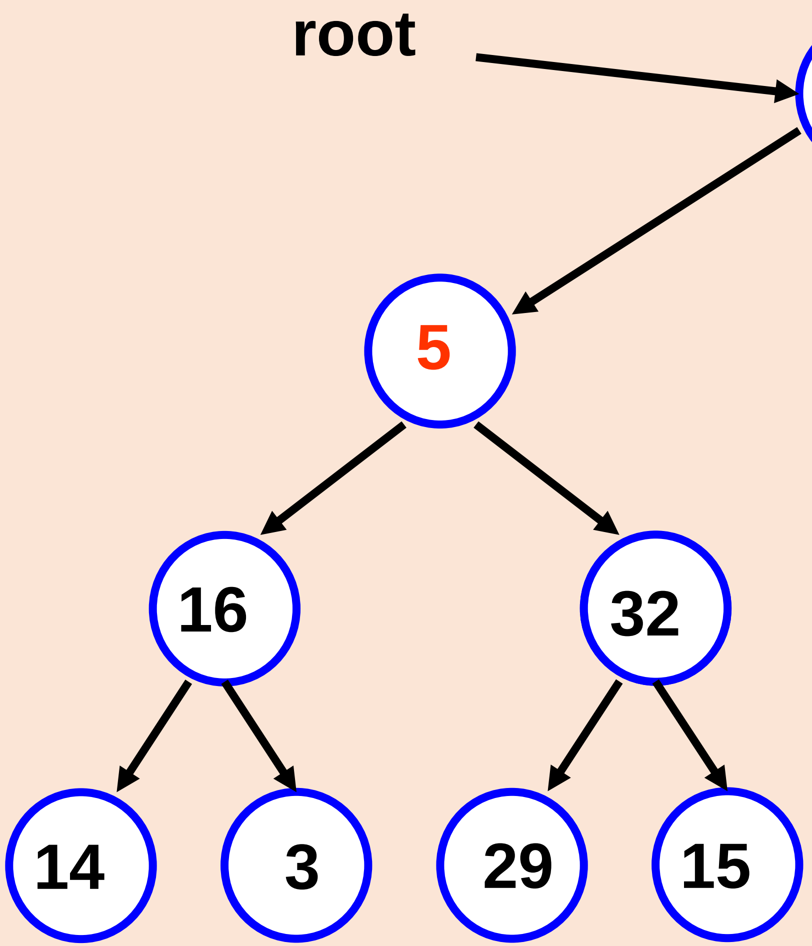
Dequeue (cont.)



remElement: 46

It was the greatest child, so it is greater than the other child (28)

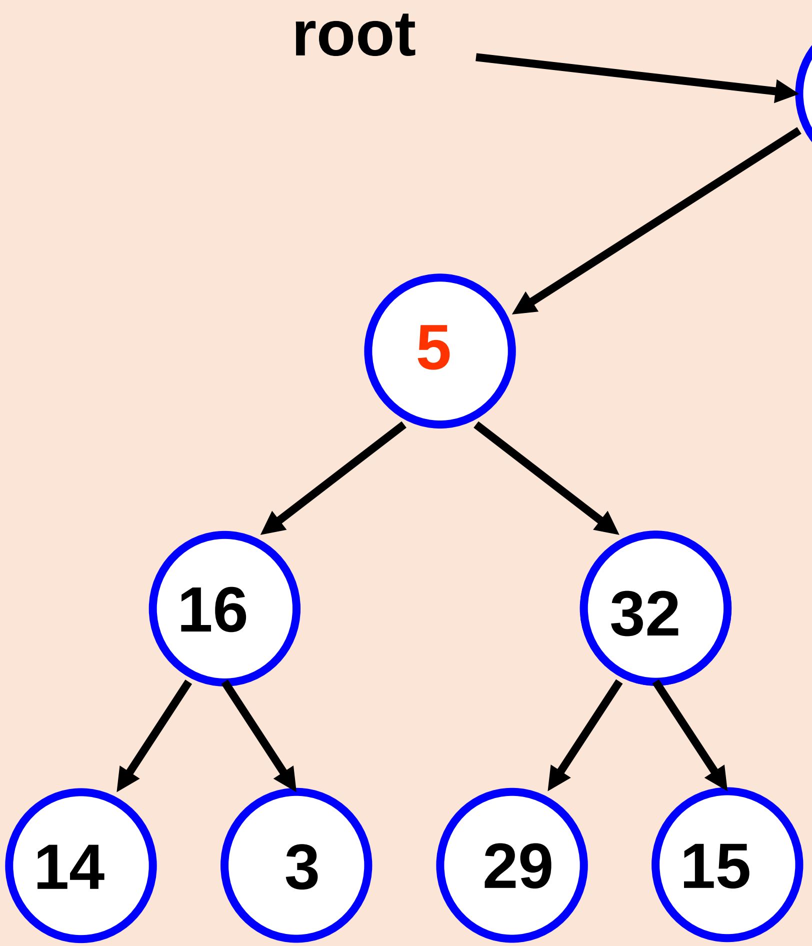
Dequeue (cont.)



remElement: 46

It is greater than the element it was swapped with (5), or else we wouldn't have swapped.

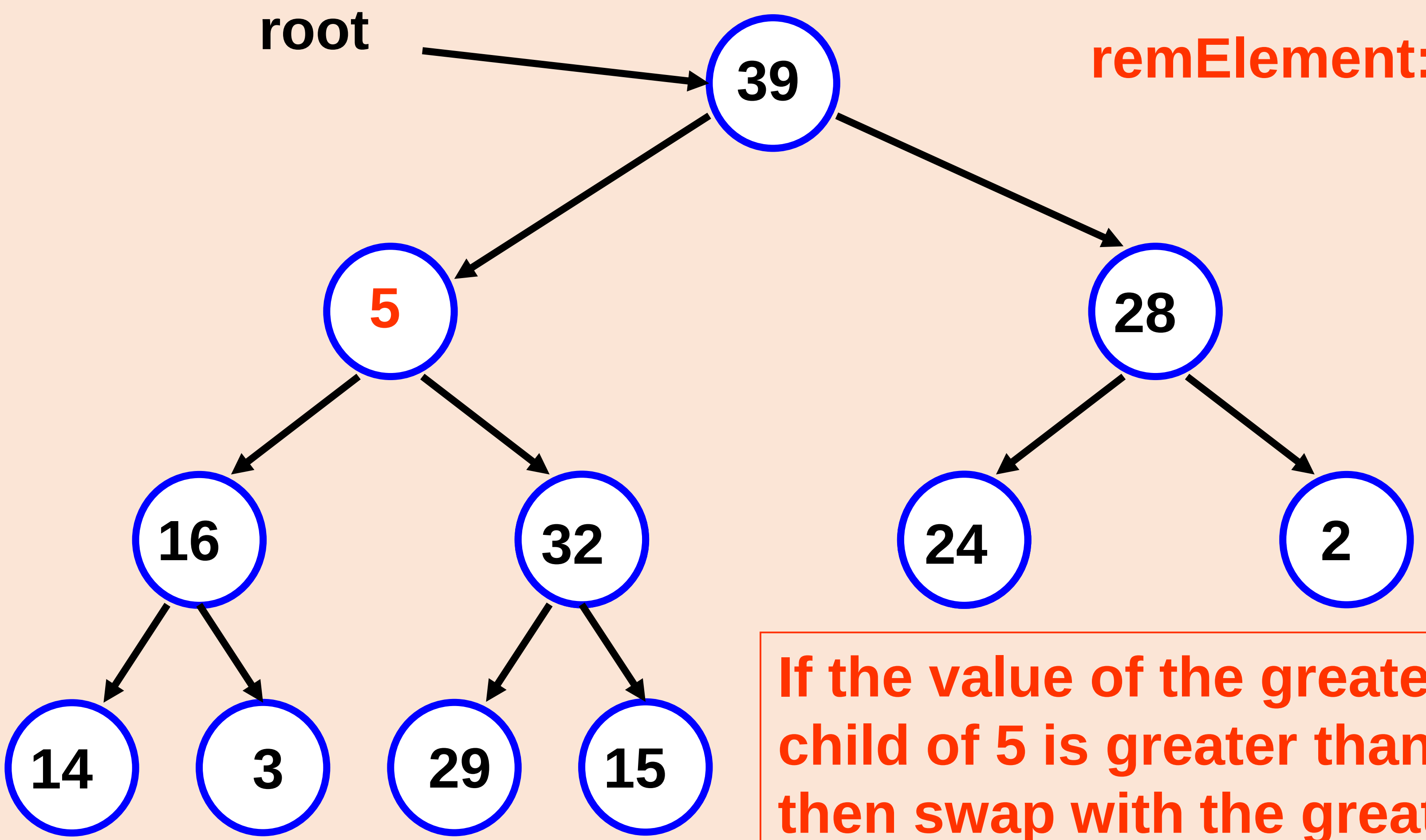
Dequeue (cont.)



remElement: 46

At this point, we repeat the process, using 5 as the parent.

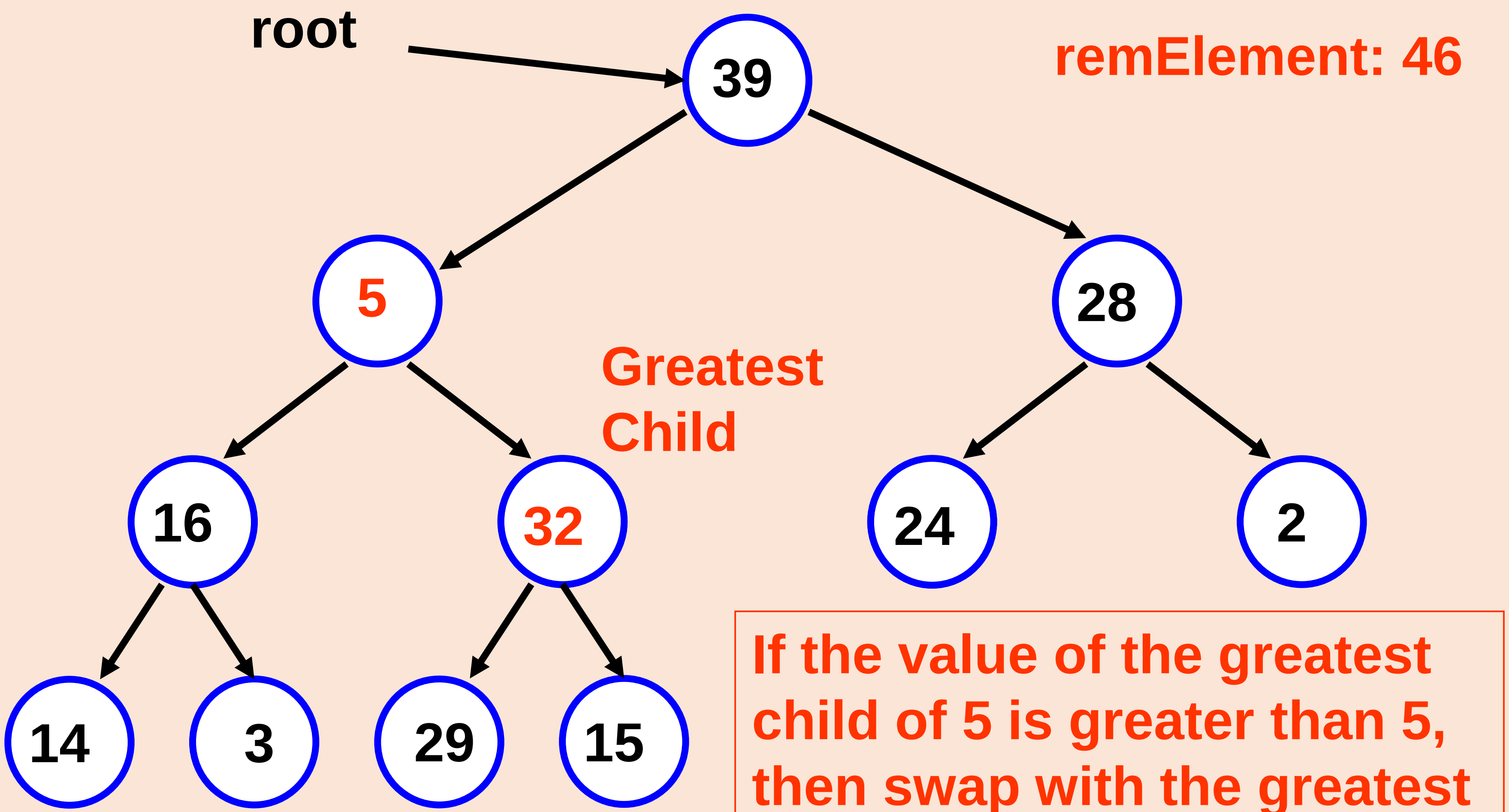
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)

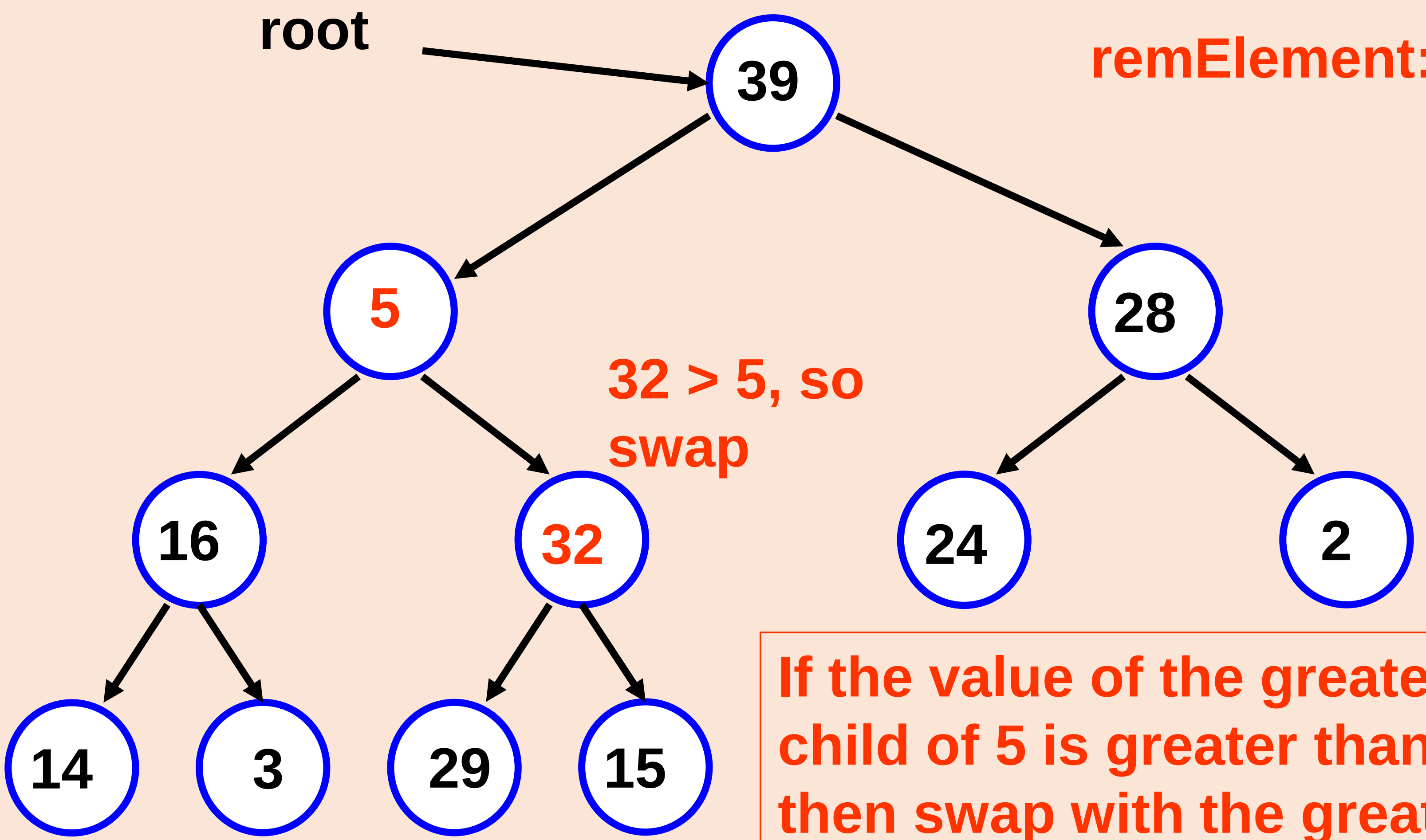


remElement: 46

Greatest Child

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)

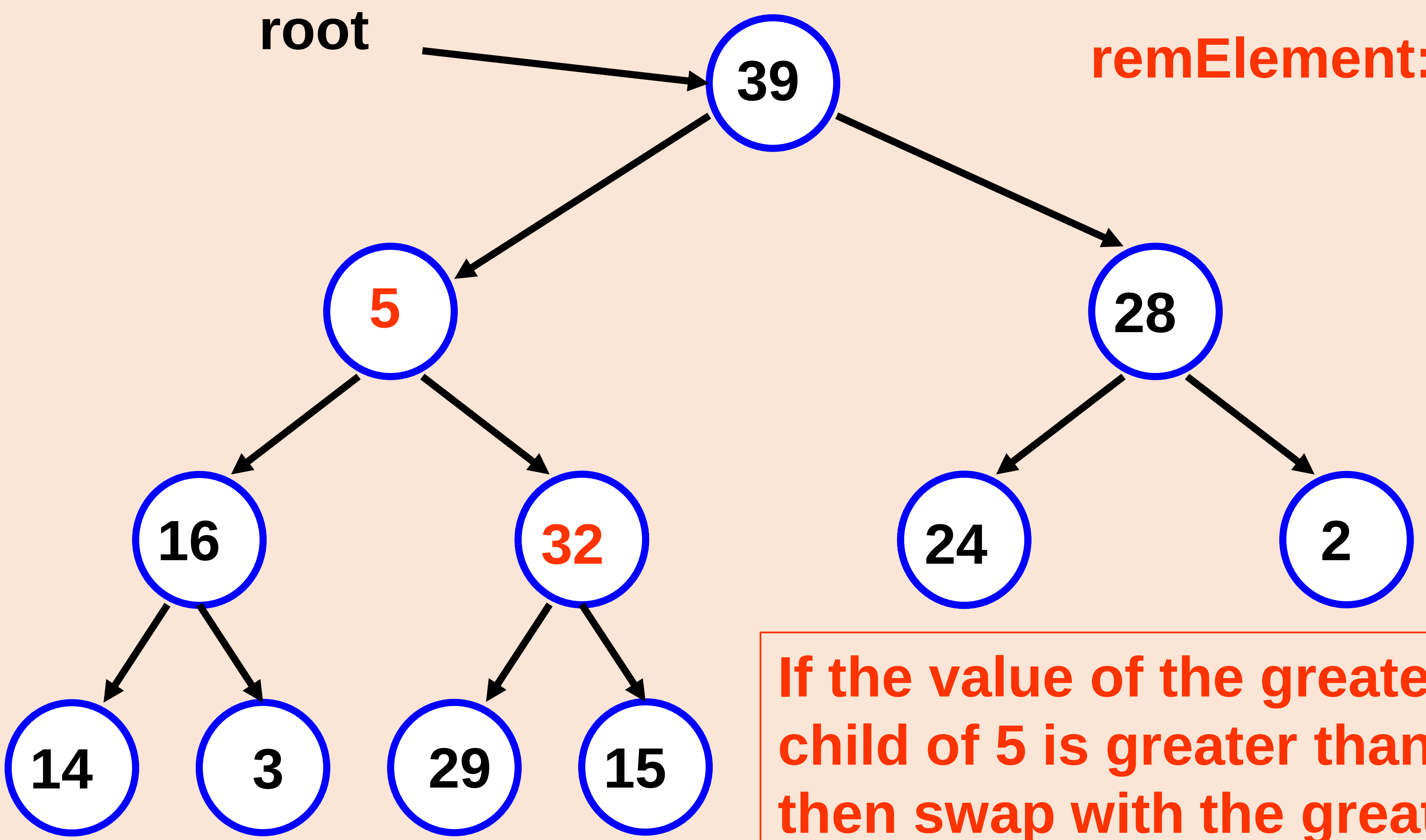


remElement: 46

32 > 5, so swap

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

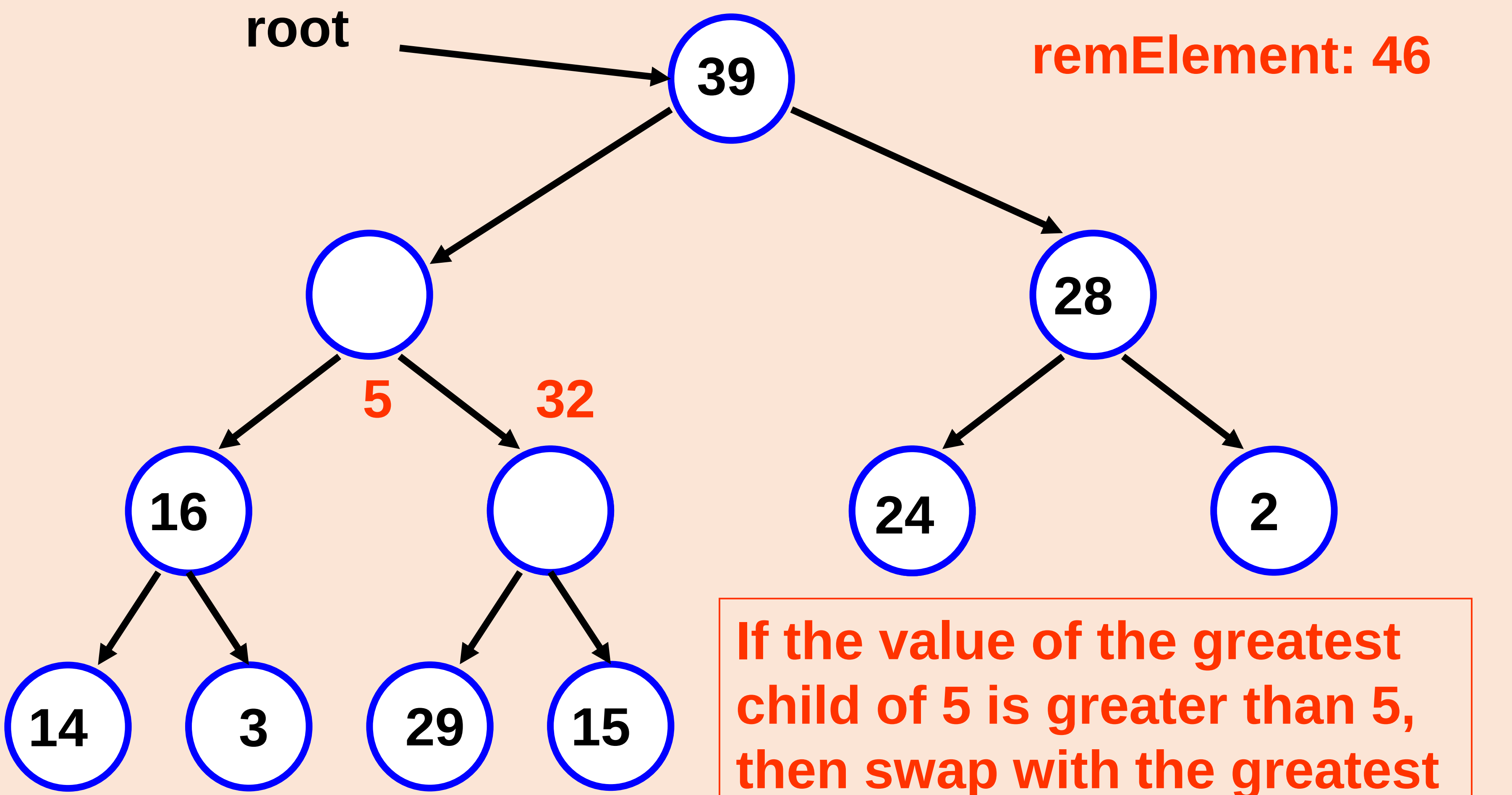
Dequeue (cont.)



remElement: 46

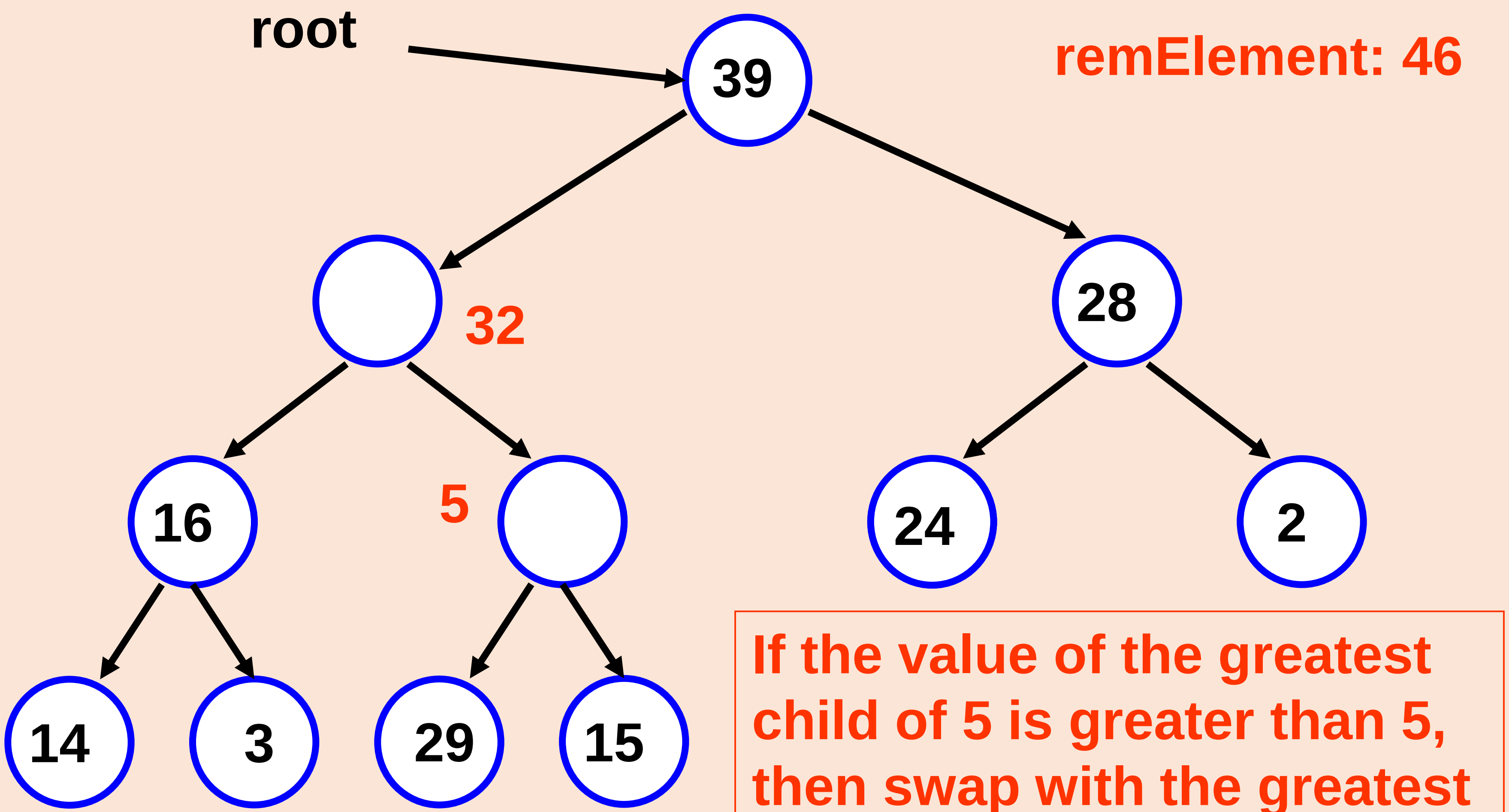
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

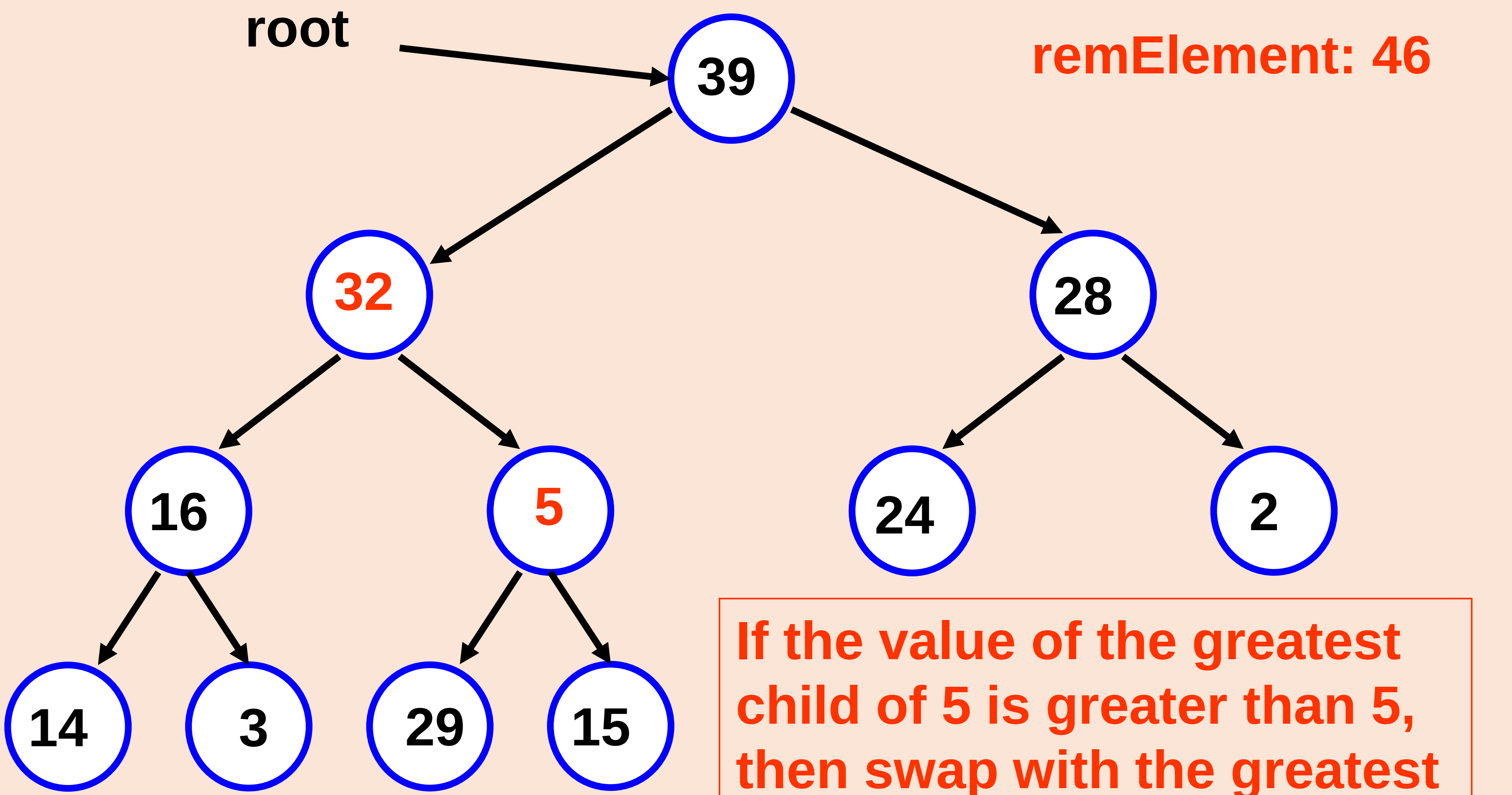
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

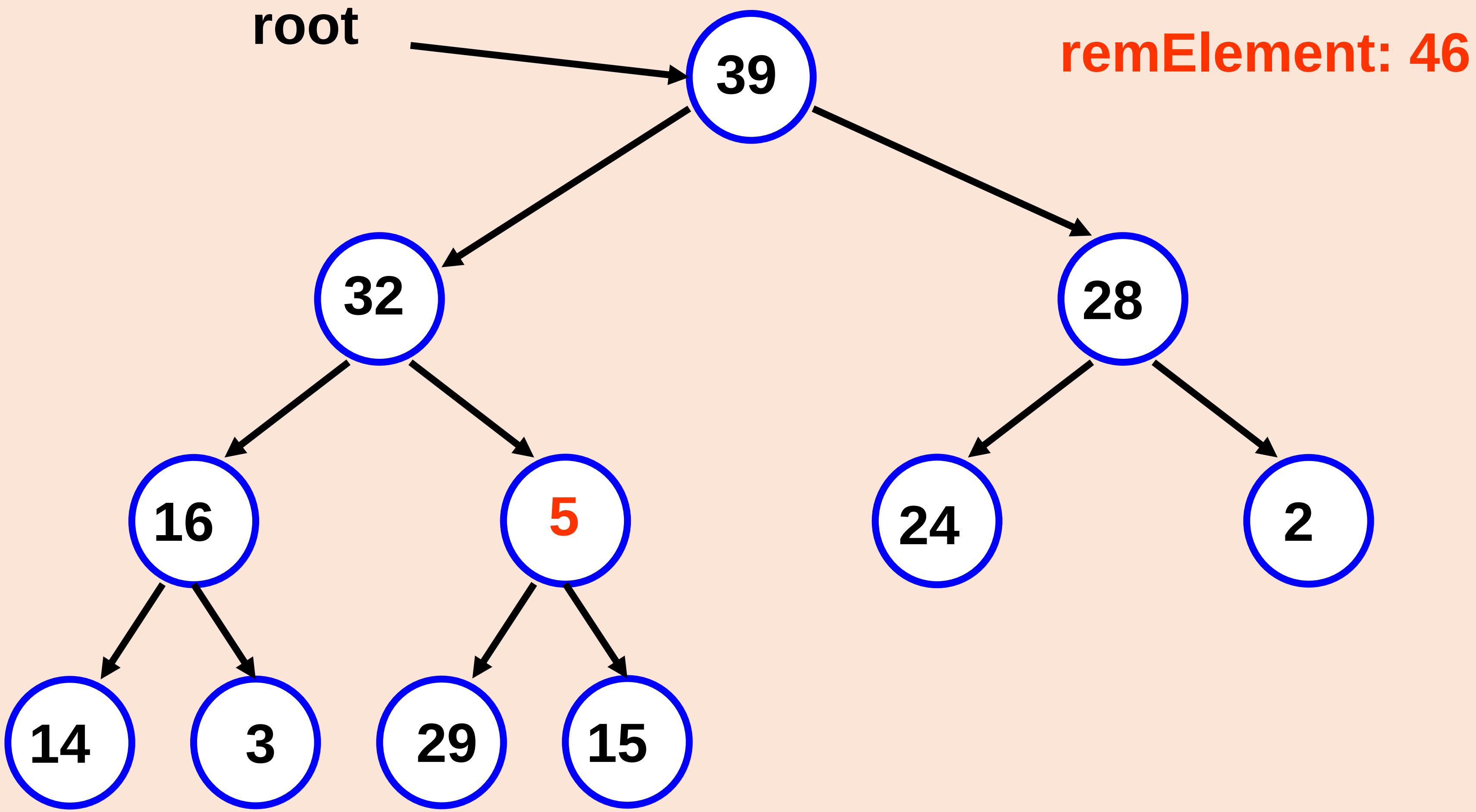
Dequeue (cont.)



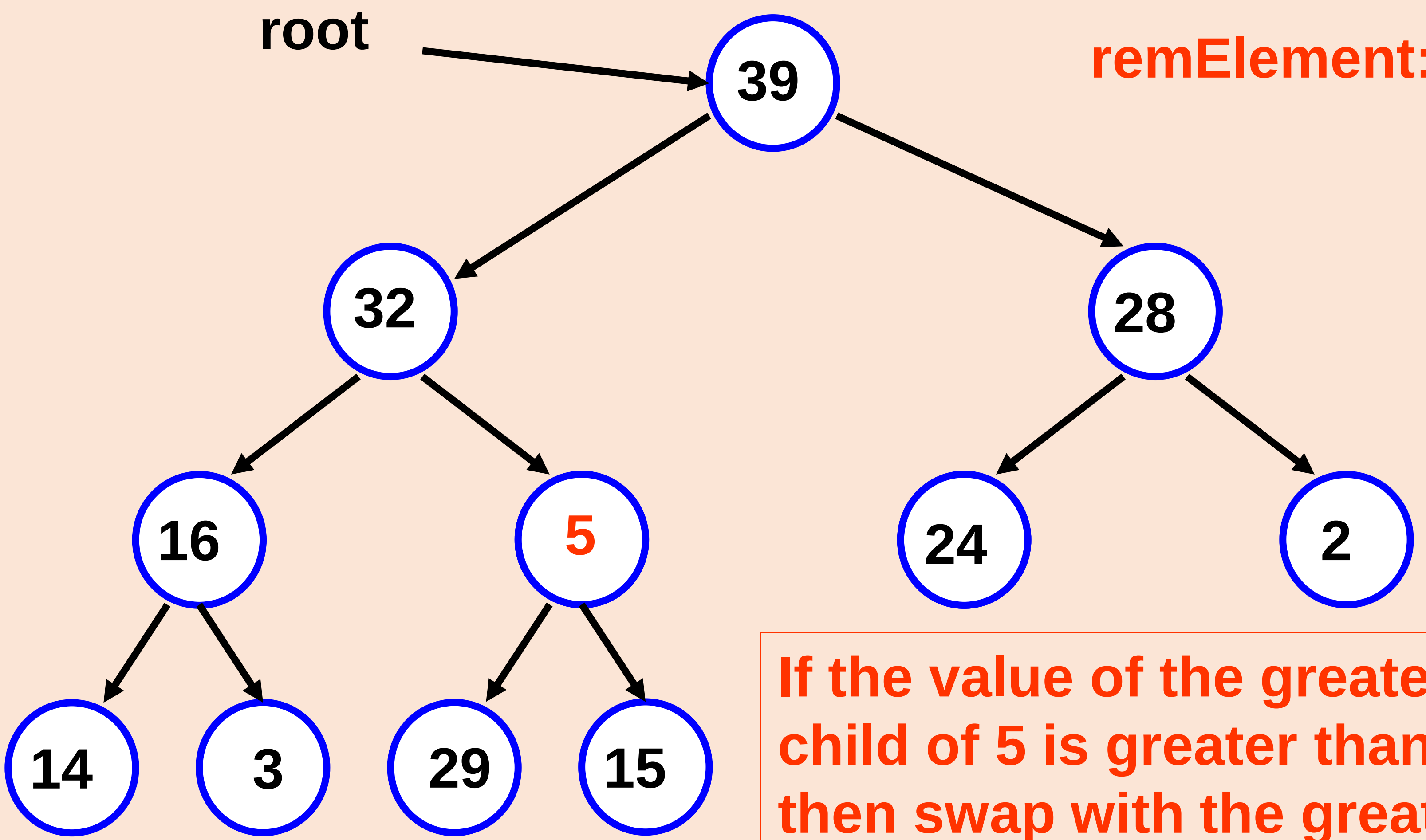
remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



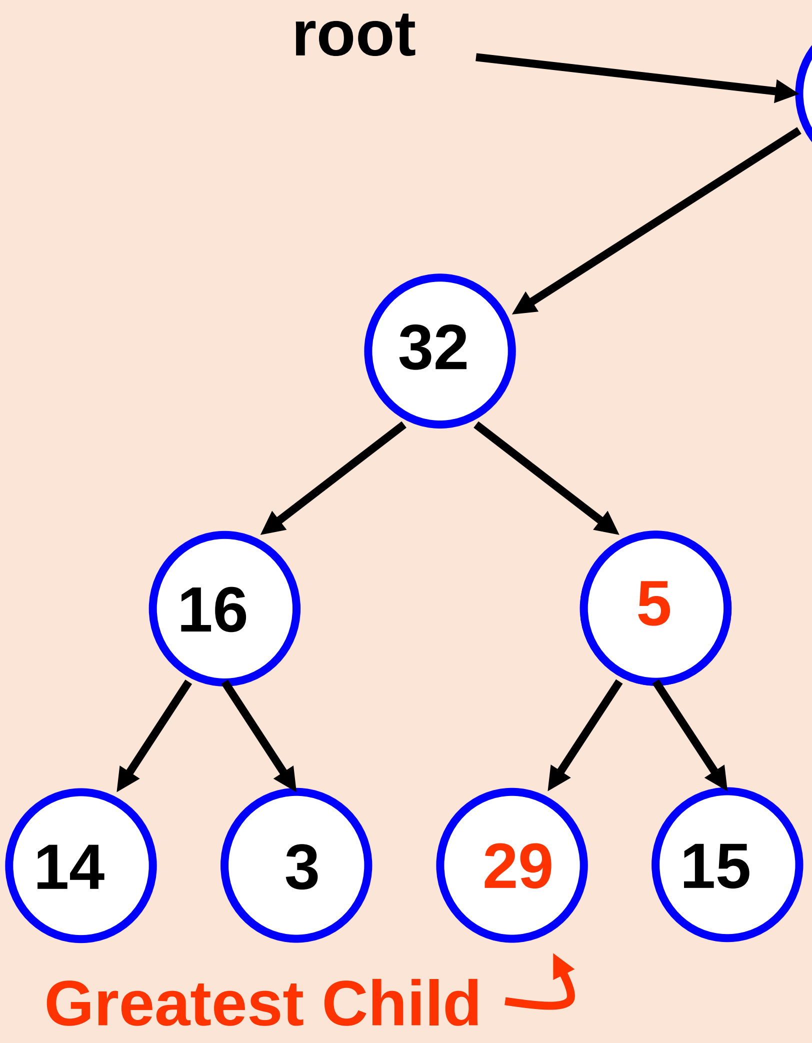
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

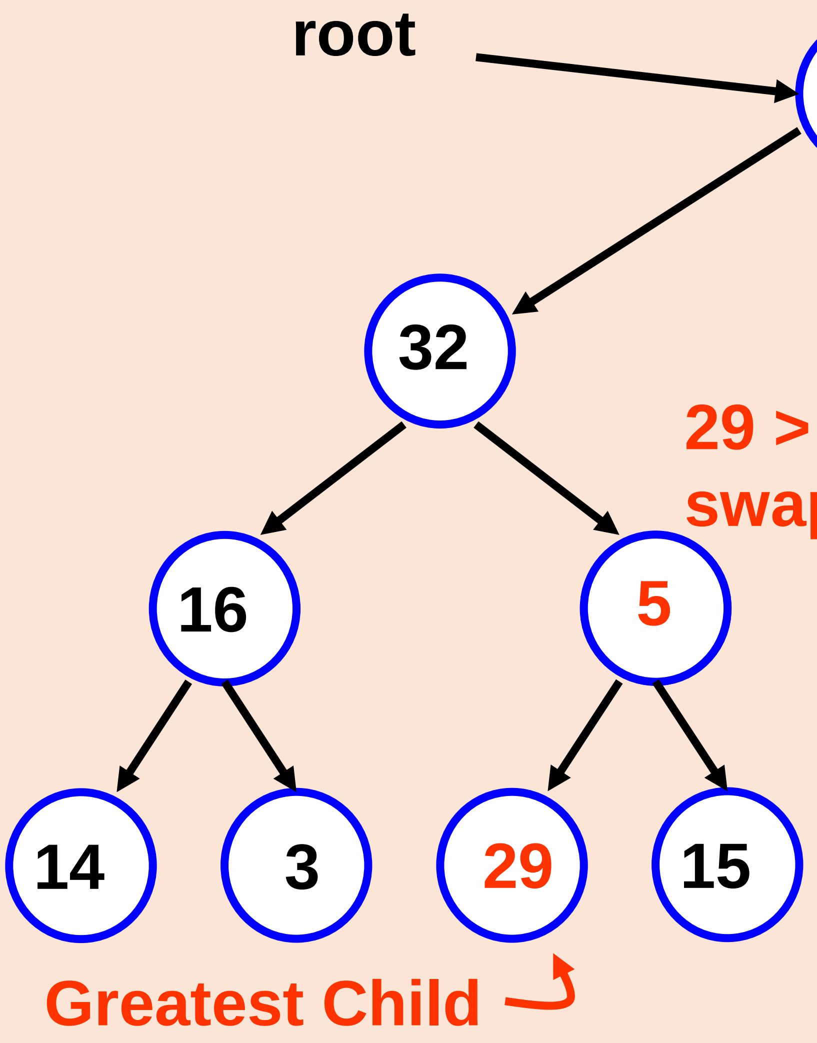
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)

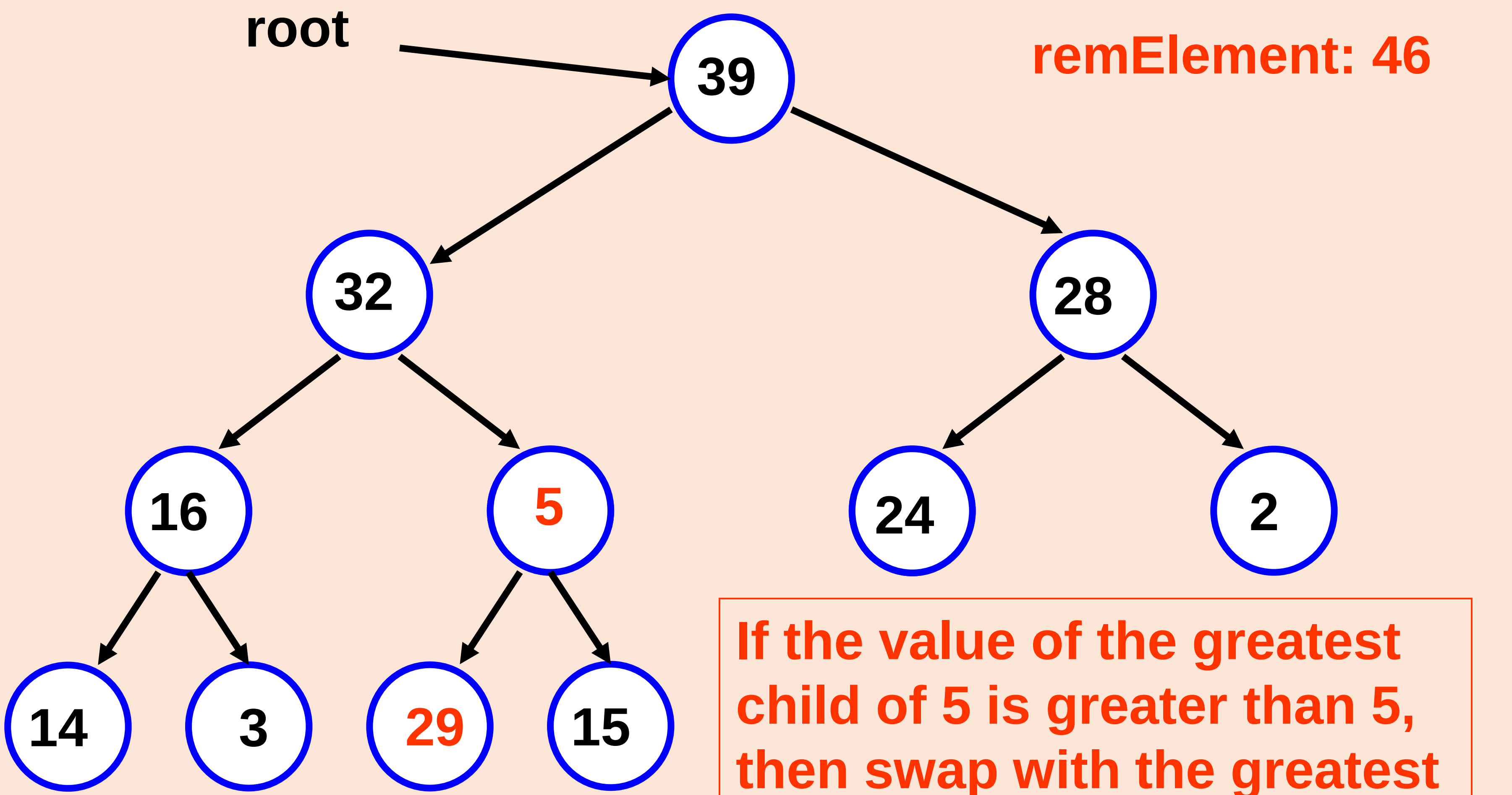


remElement: 46

29 > 5, so swap

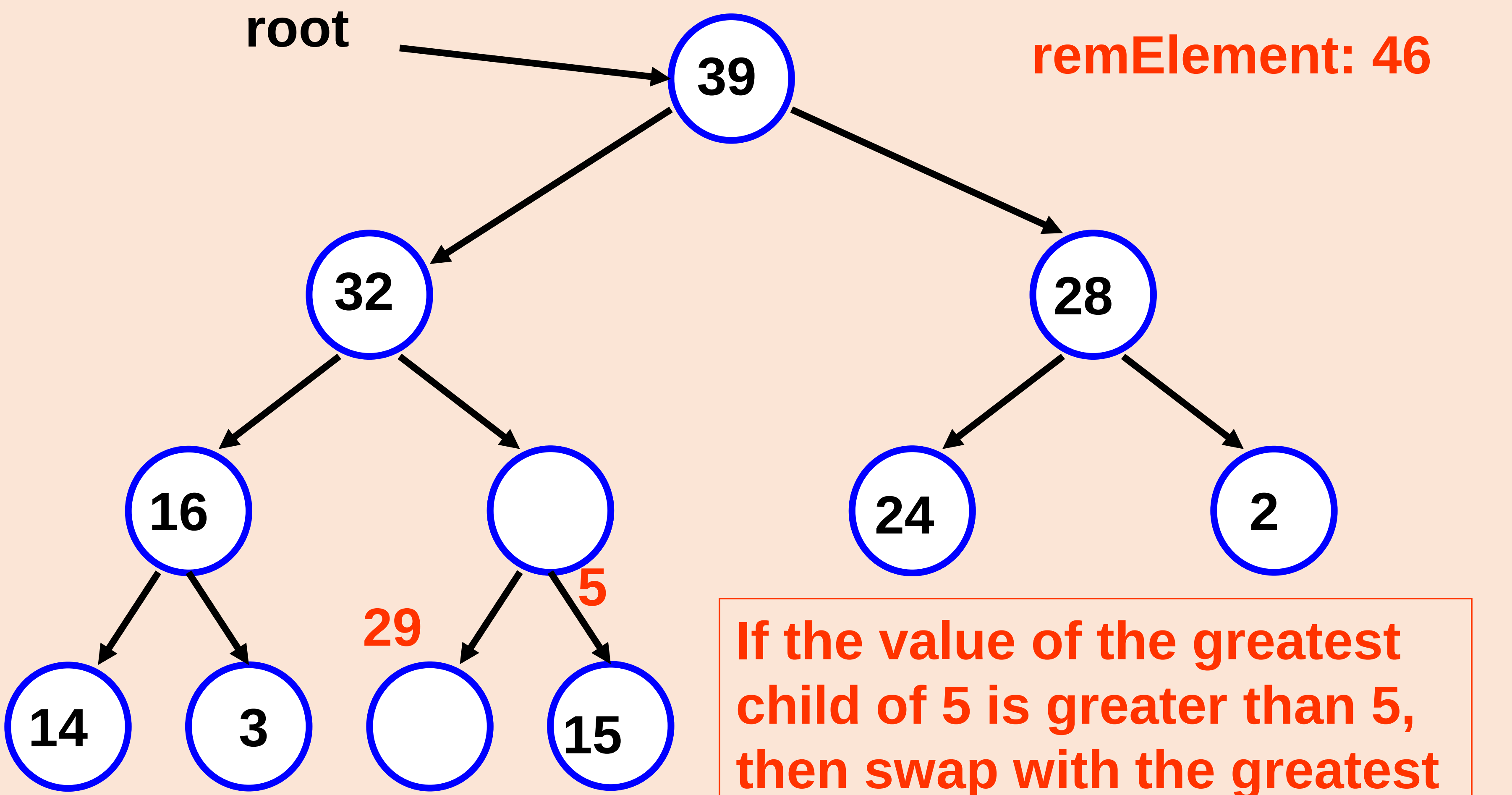
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



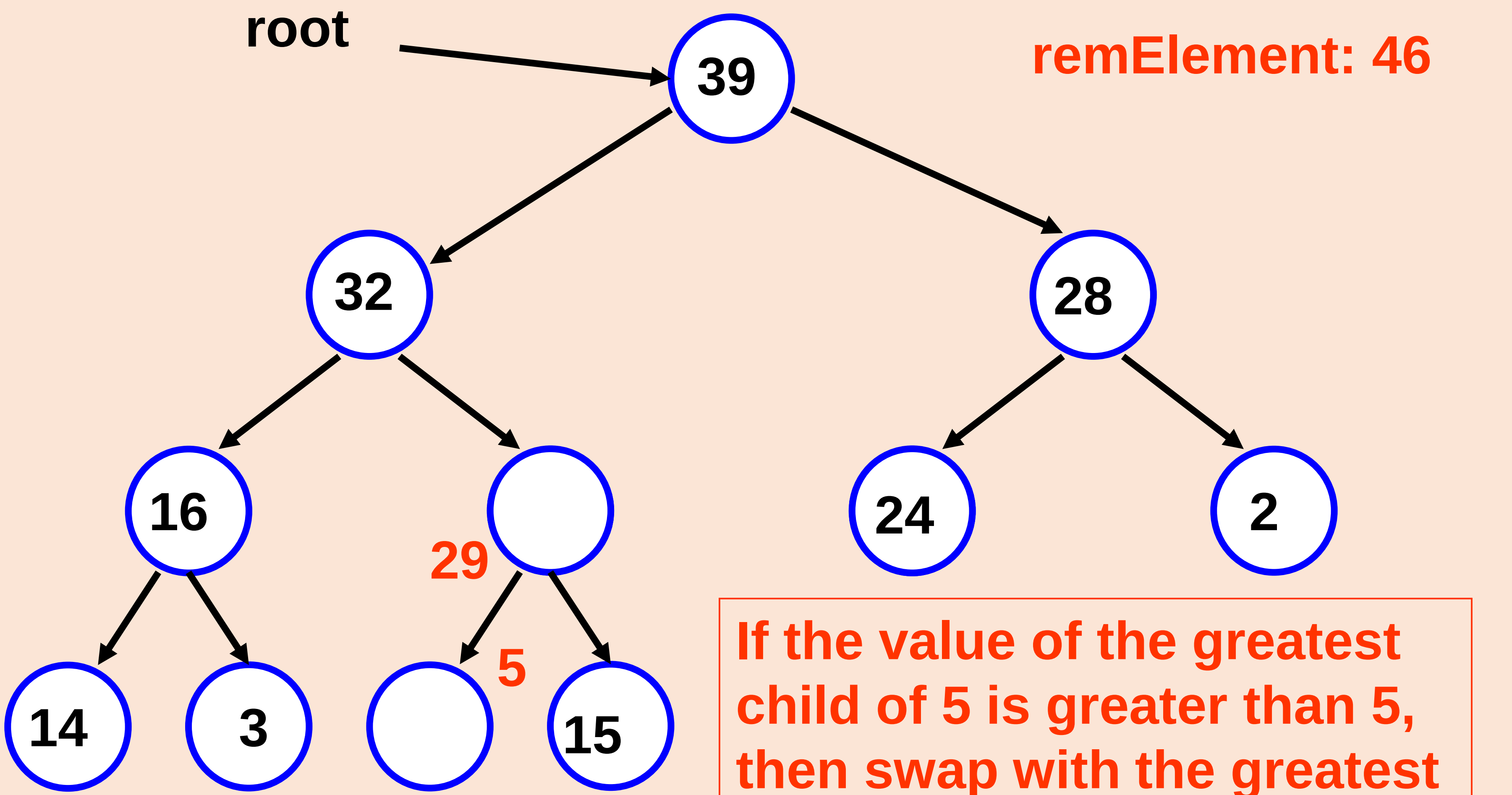
If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

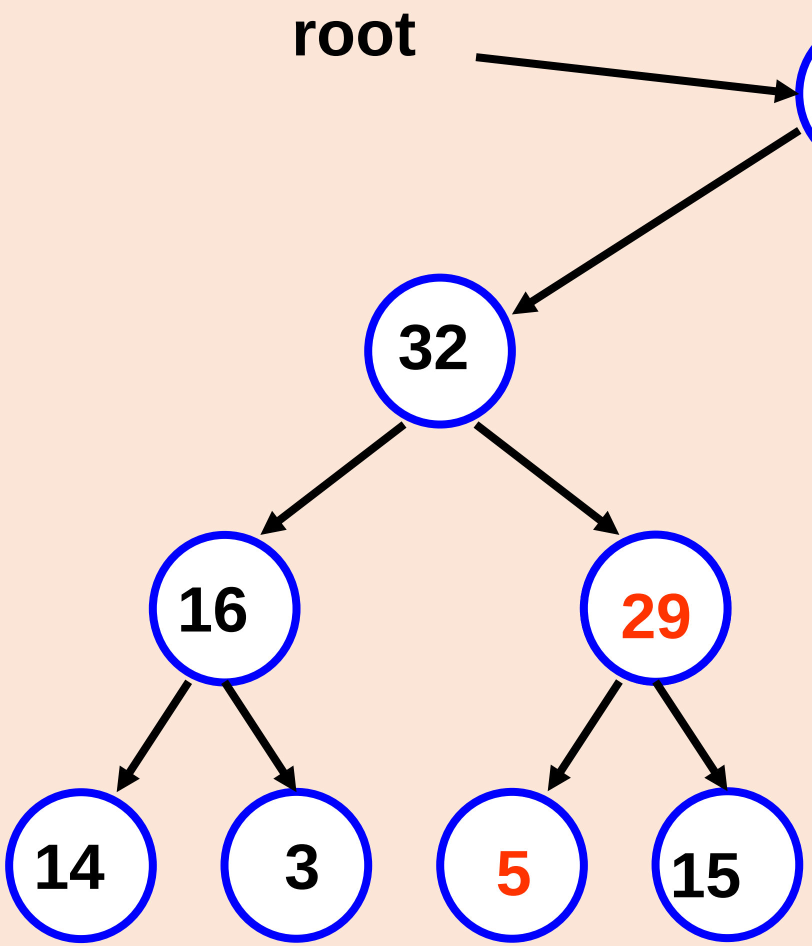
Dequeue (cont.)



remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

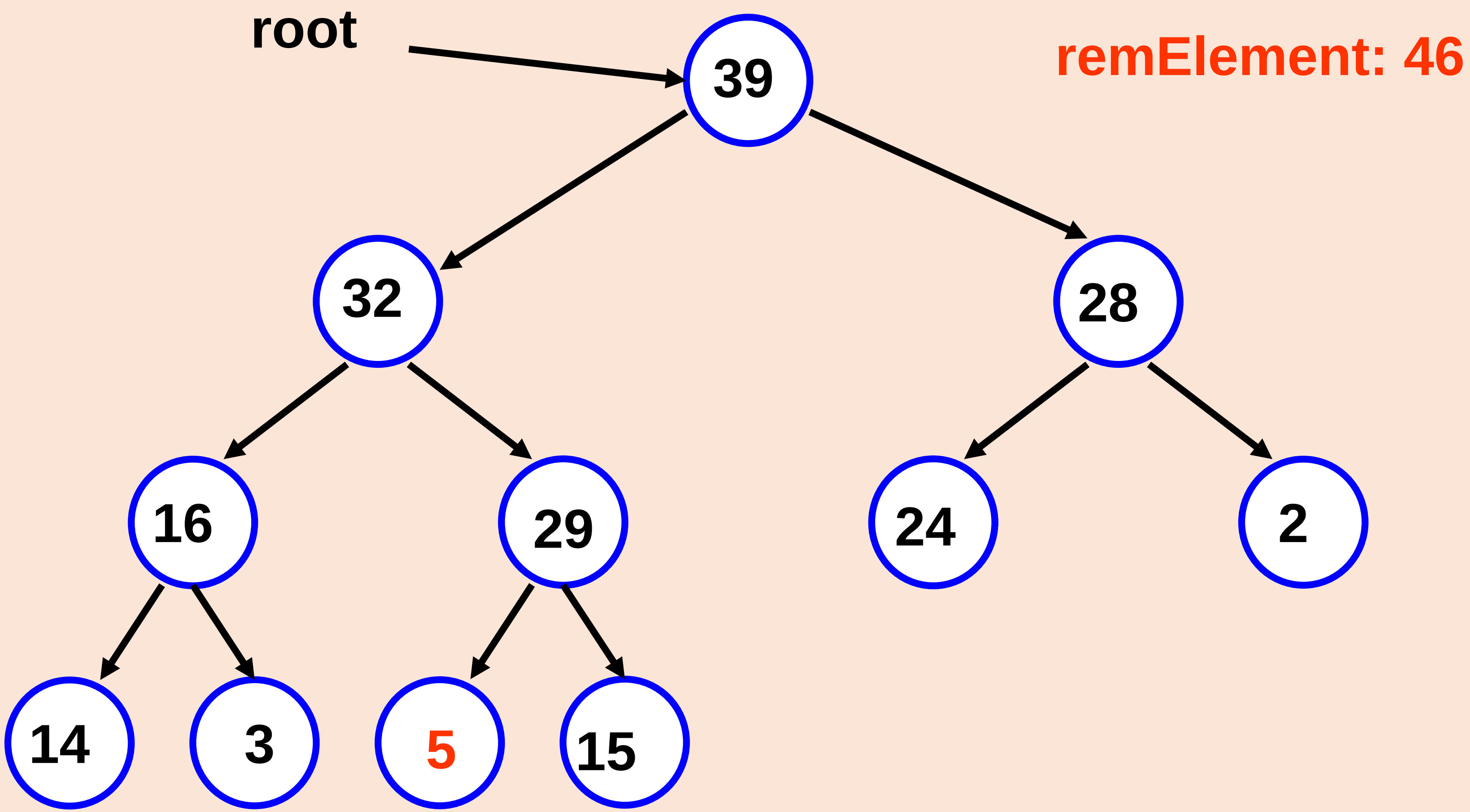
Dequeue (cont.)



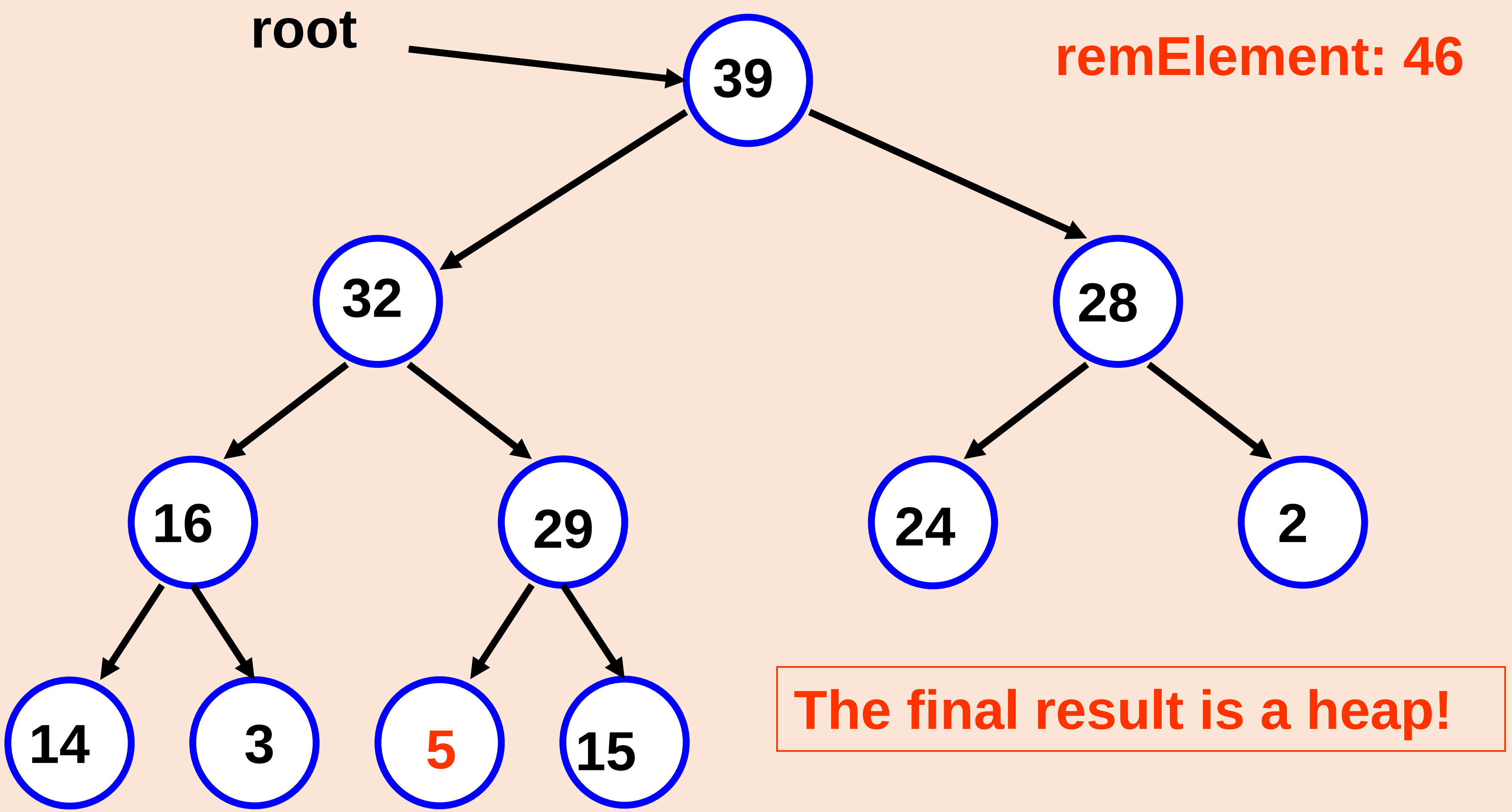
remElement: 46

If the value of the greatest child of 5 is greater than 5, then swap with the greatest child

Dequeue (cont.)



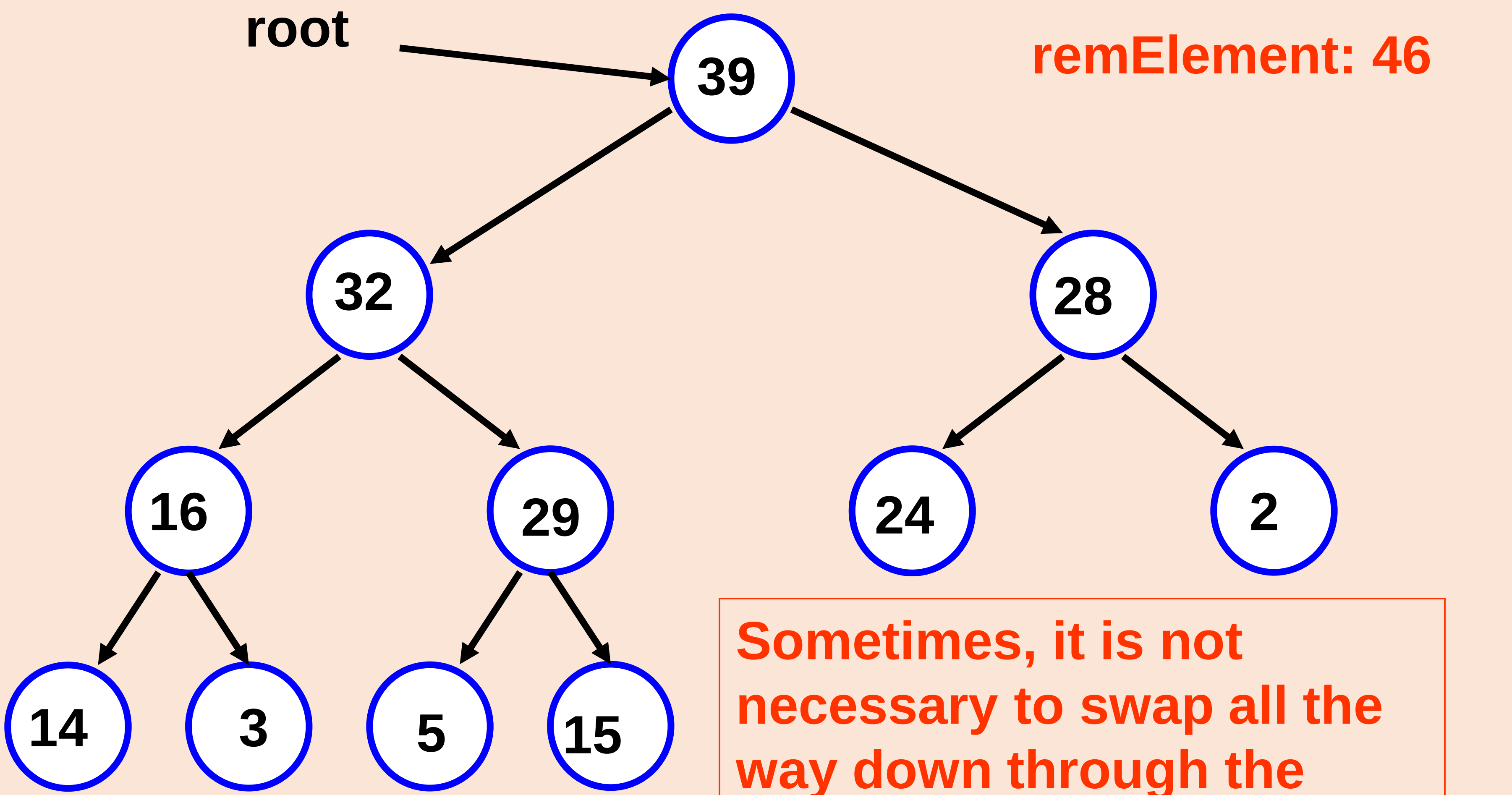
Dequeue (cont.)



remElement: 46

The final result is a heap!

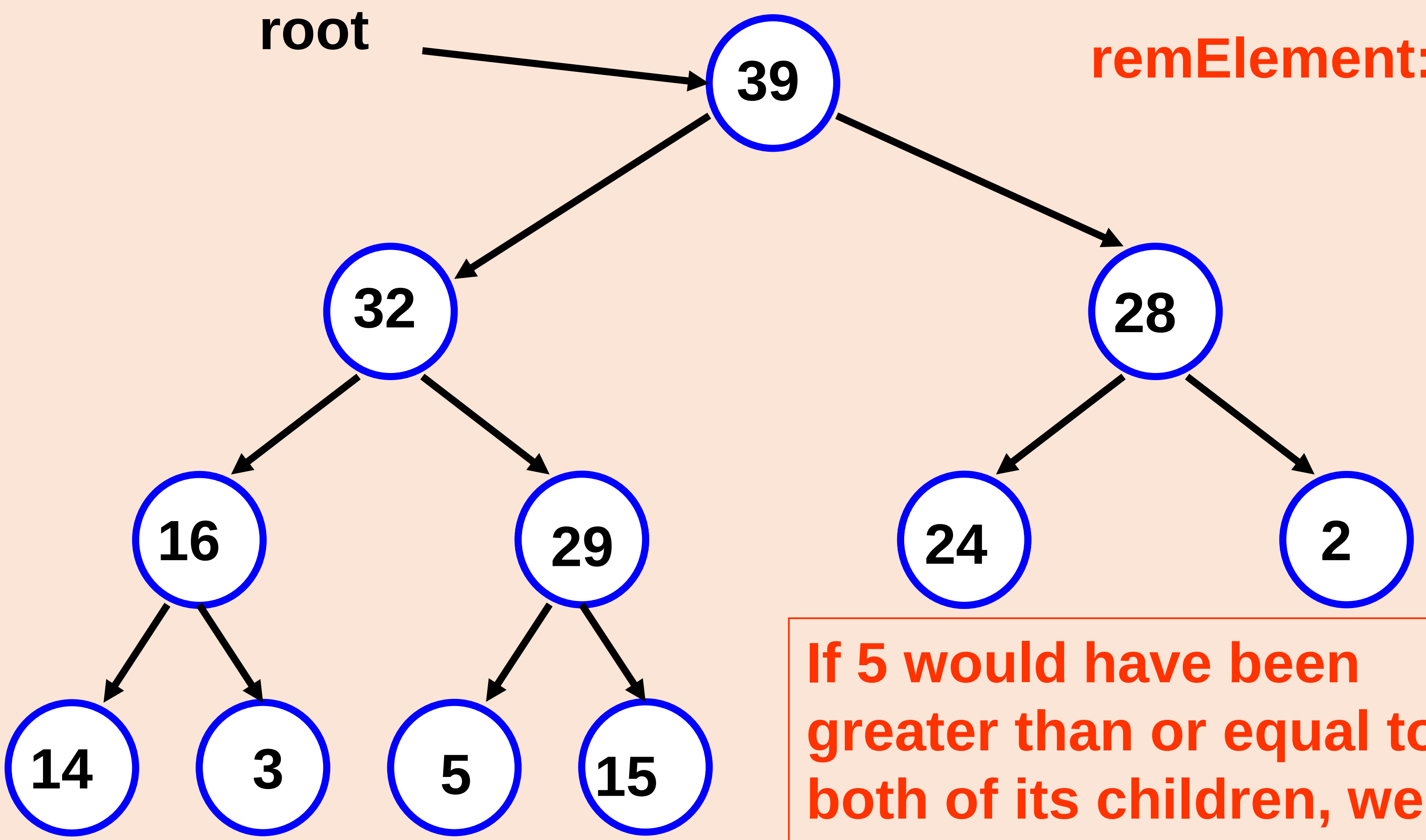
Dequeue (cont.)



remElement: 46

Sometimes, it is not necessary to swap all the way down through the heap

Dequeue (cont.)



remElement: 46

If 5 would have been greater than or equal to both of its children, we would stop there

Heapify or ReheapDown

- The process of swapping downwards to form a new heap is called ***heapifying***
- When, we heapify, it is important that the rest of the structure is a heap, except for the root node that we are starting off with; otherwise, a new heap won’t be formed
- A loop is used for heapifying; the number of times through the loop is always O(lg n) or less, which gives the O(lg n) complexity
- Each time we swap downwards, the number of nodes we can travel to is reduced by approximately half

ReheapDown Algorithm

- If the root is a leaf node, do nothing
- Find the maximum of the root’s children
- If the root is less than the max child, swap the two nodes and recurse on the max child’s index, which now contains the root’s value