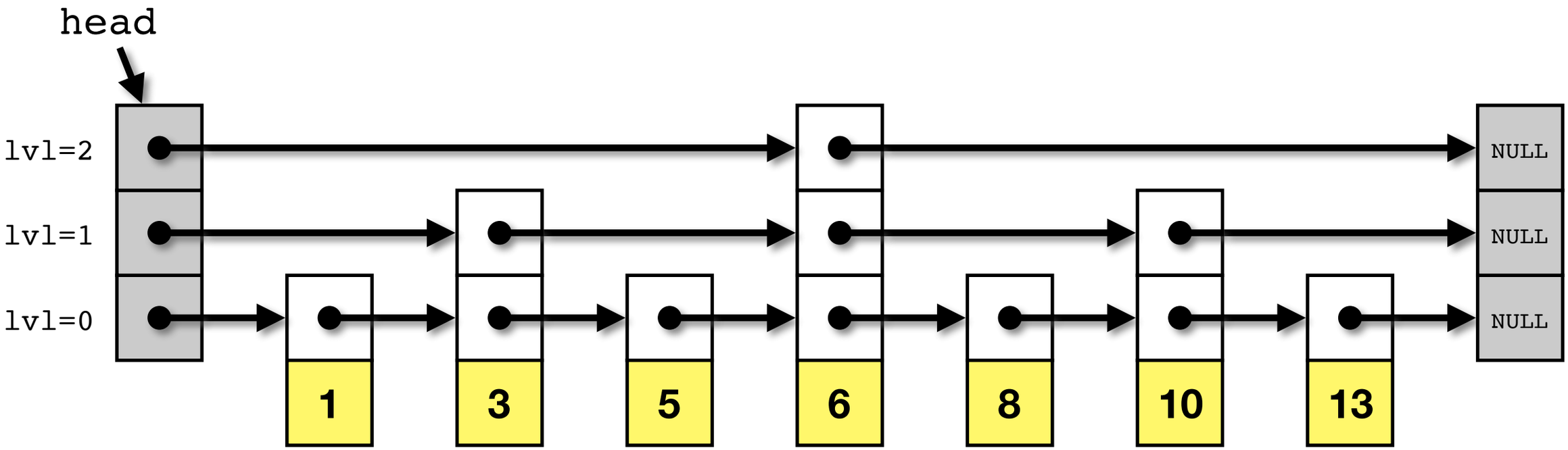
CS350: Data Structures

Skip Lists

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

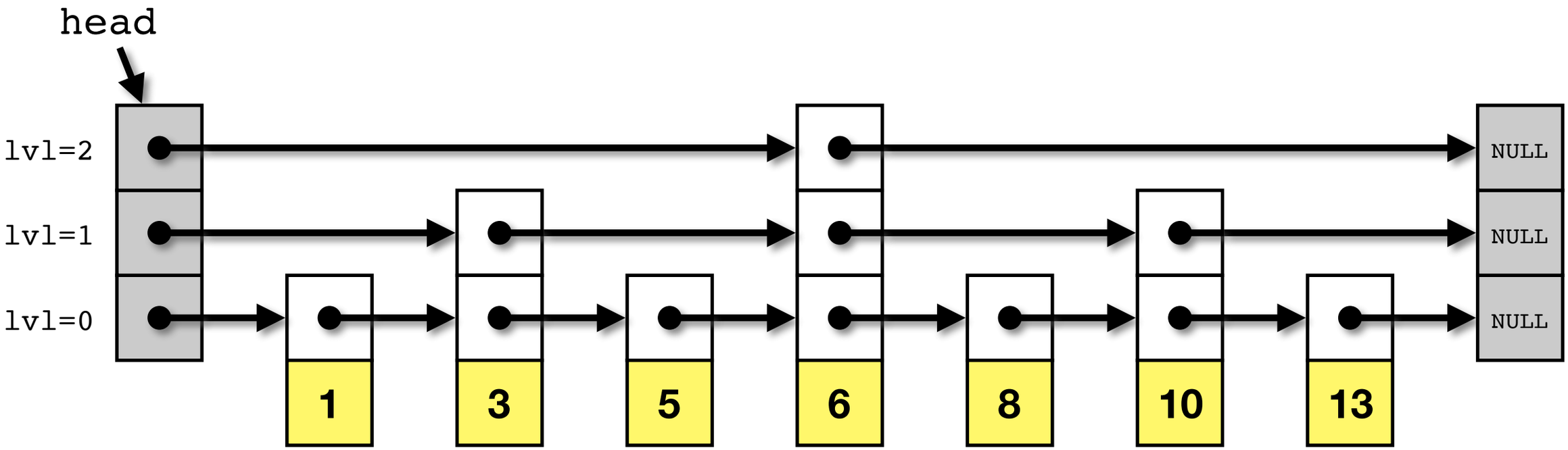
Skip List Introduction

- **A data structure used for storing a sorted list of elements using layers of linked lists**
  - Bottom layer is a standard, ordered linked list
  - Upper layer linked lists create ‘shortcuts’ or ‘fast lanes’ from one location of the list to another
  - Higher levels traverse greater portions of the list than lower levels



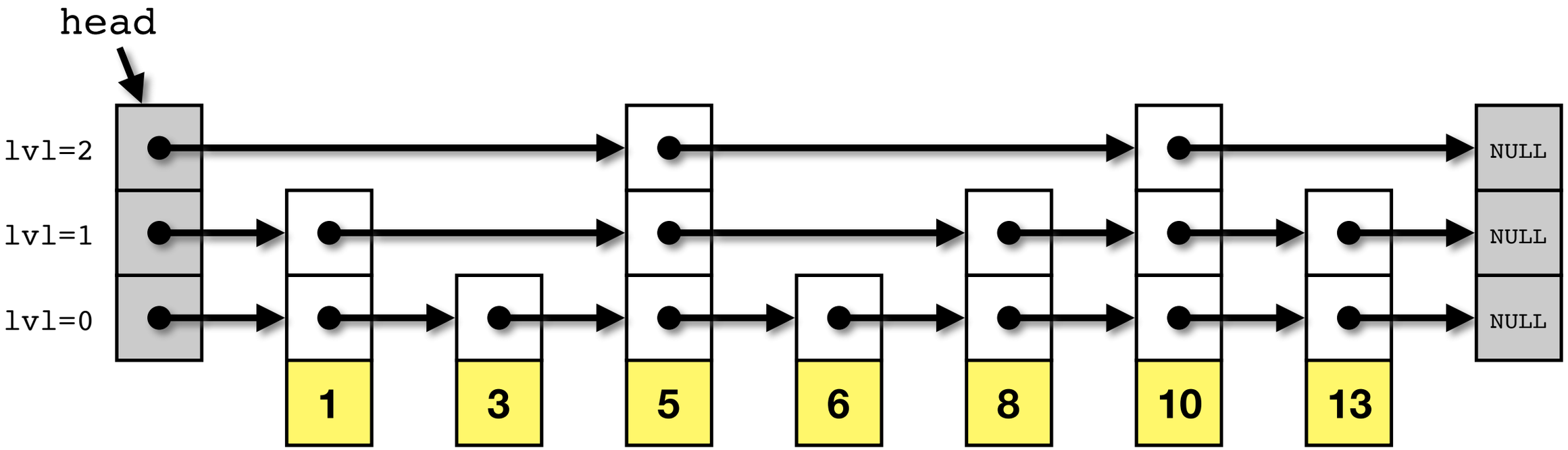
Skip List Introduction

- **The example skip list below shows the skip list divided perfectly in halves, quarters, etc.**



Skip List Introduction

- **In practice, skip list node heights are distributed randomly throughout the skip list**
  - Helps us avoid the need to ‘rebalance’ the list when insertions or deletions occur



Skip Lists

- **Operations on skip lists are comparable in efficiency to balanced binary search trees (e.g. AVL trees, Red-Black trees)**
  - Insertion, Deletion, and Search operations all run in O(log N) time

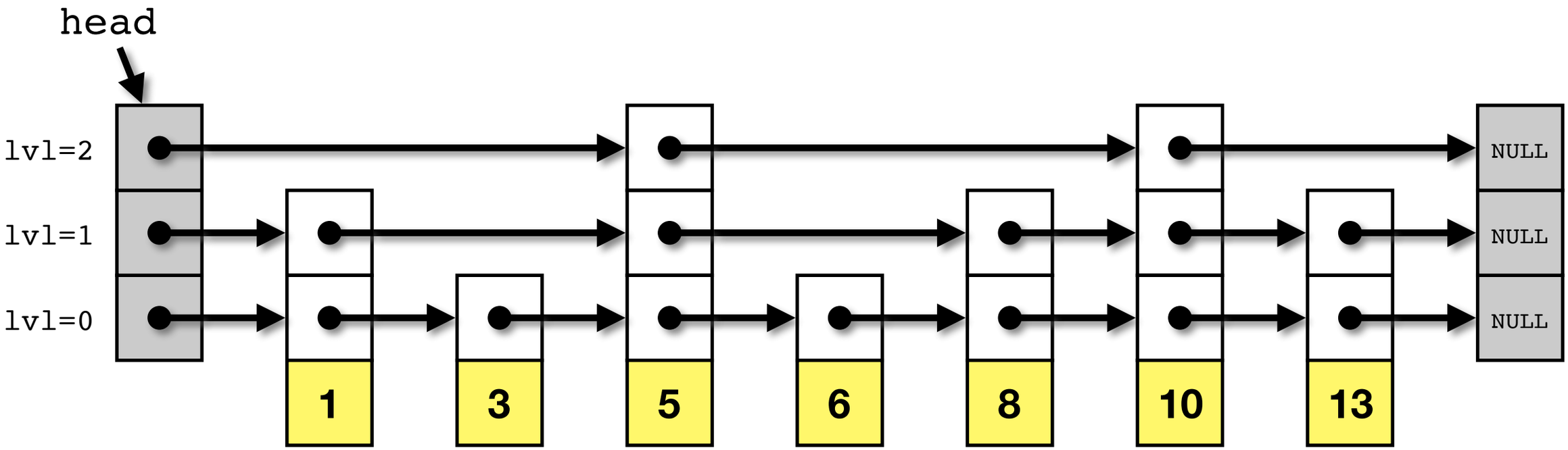
Skip List Search

Skip List Searching

- **To search for an element in a skip list:**
  - Start at the highest level at the head of the skip list and begin moving horizontally through the list
    - If the data in the next node is less than the desired value, move to the next node and continue the search
    - If the data in the next node is greater than or equal to the desired value, drop down one level and continue the search
    - When you reach the bottommost level, move forward one node
    - If the data in the current node is the desired data then it was found, otherwise, the data is not in the list

Skip List Searching

- **Example -- `find` the node with a key value 6**



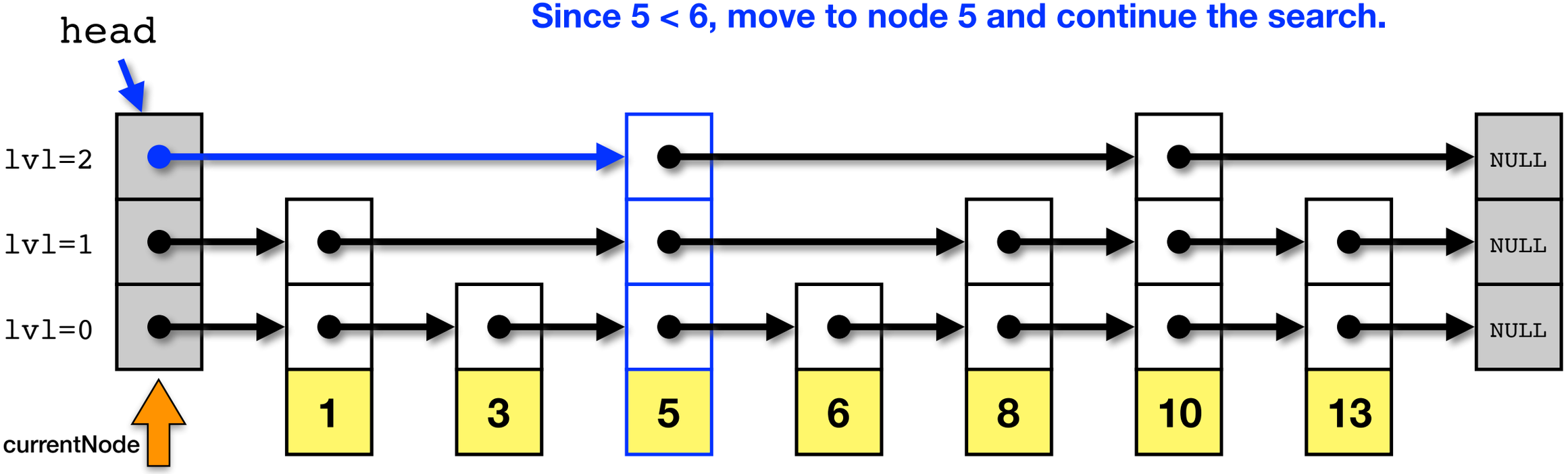
Skip List Searching

- **Example -- `find` the node with a key value 6**

(1) Start at highest level of the head node and check data value at node.next

(2) if node.next.data < desired value move to next node otherwise, drop down one level at current node.

Since 5 < 6, move to node 5 and continue the search.

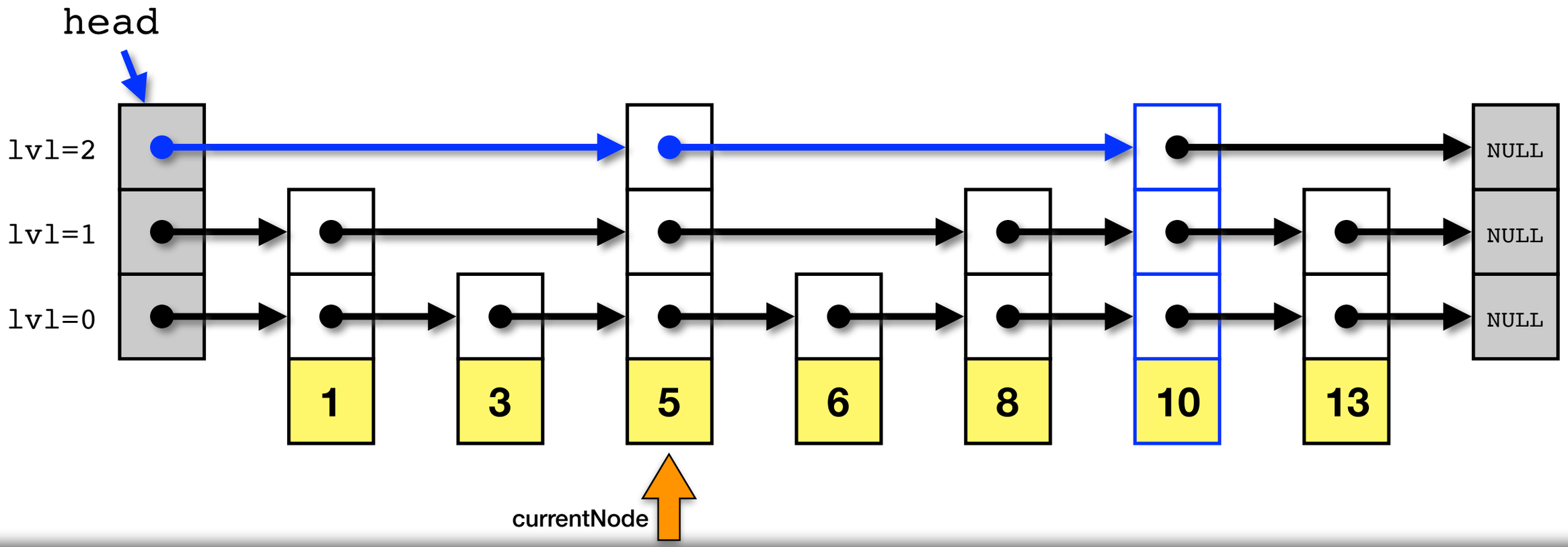


Skip List Searching

- **Example -- `find` the node with a key value 6**

(3) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

Since 10 !< 6, stay at node 5,
drop down one level and continue the search.

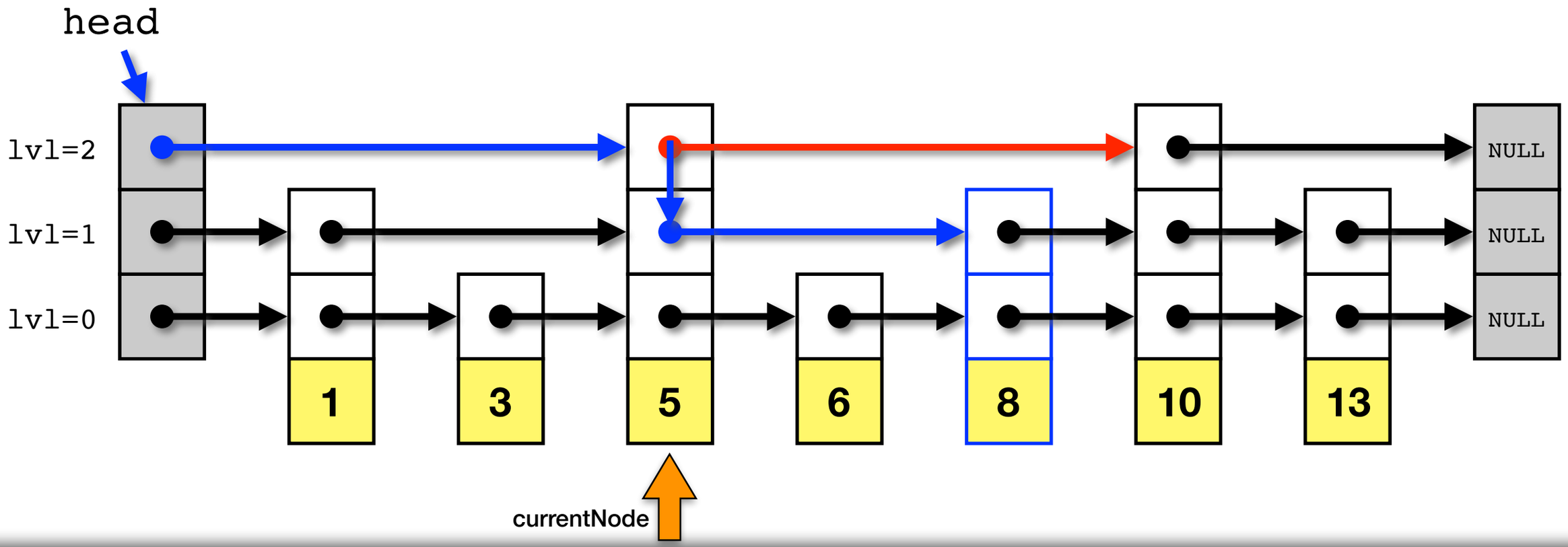


Skip List Searching

- **Example -- `find` the node with a key value 6**

(4) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

Since 8 !< 6, stay at node 5,
drop down one level and continue the search.

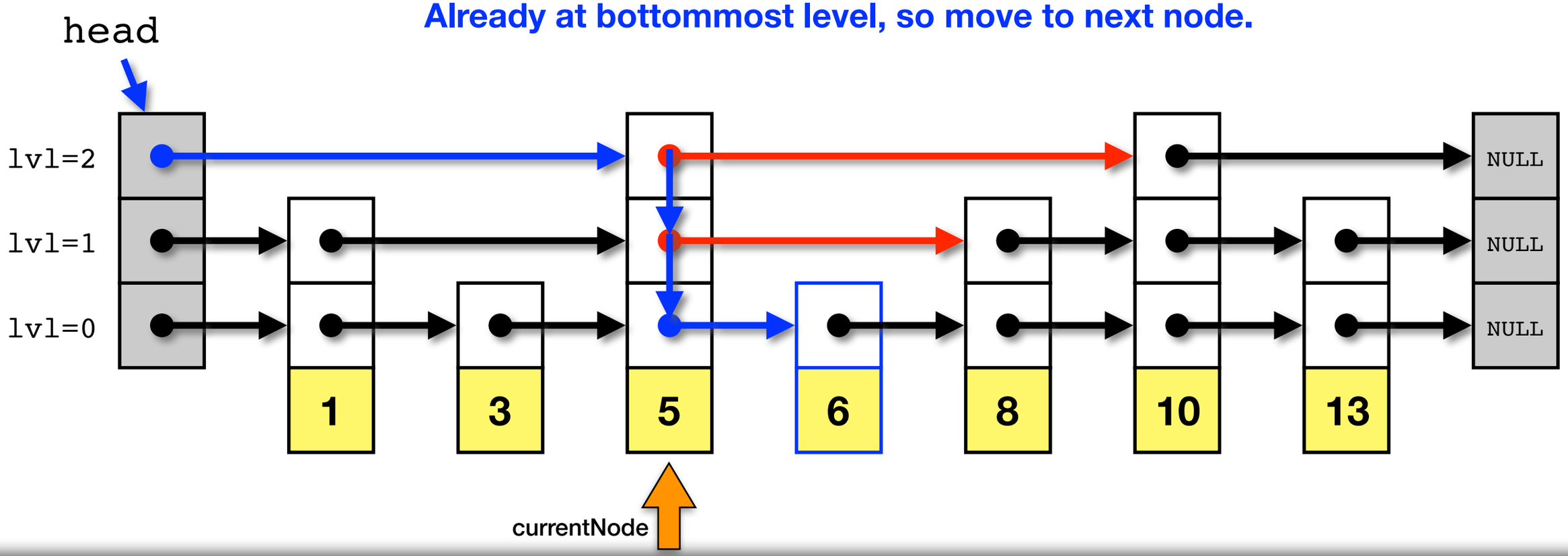


Skip List Searching

- **Example -- `find` the node with a key value 6**

(5) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

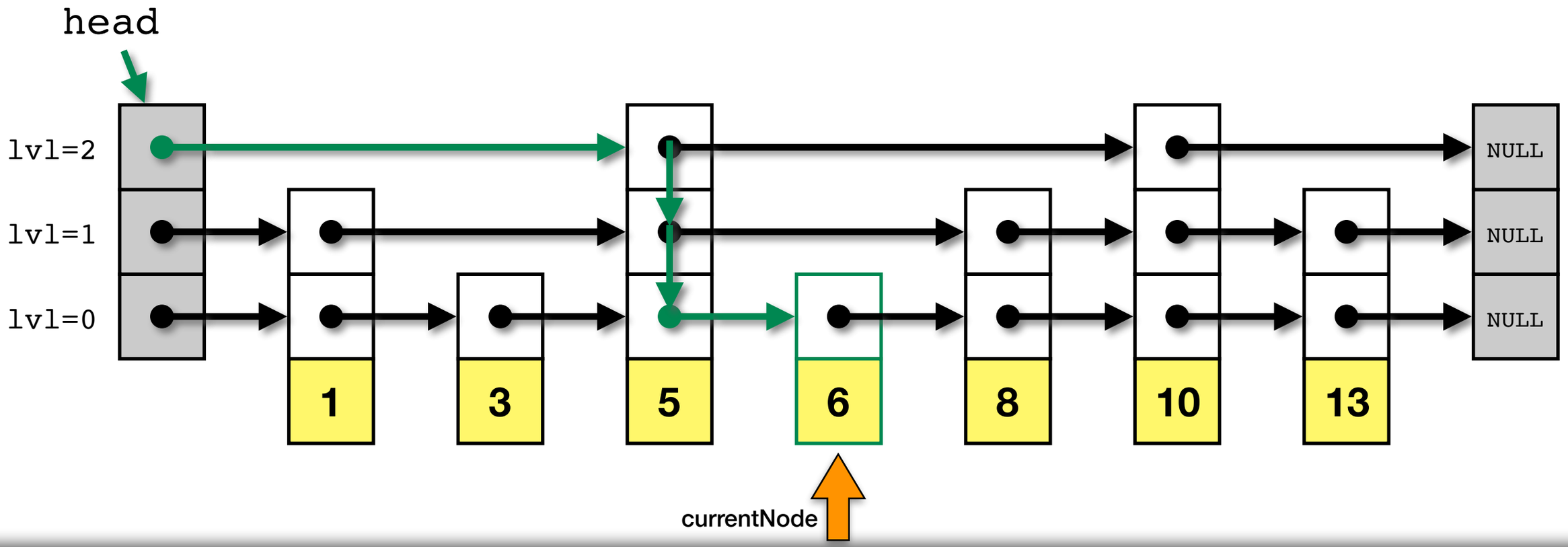
Since 6 !< 6, stay at node 5,
drop down one level and continue the search.
Already at bottommost level, so move to next node.



Skip List Searching

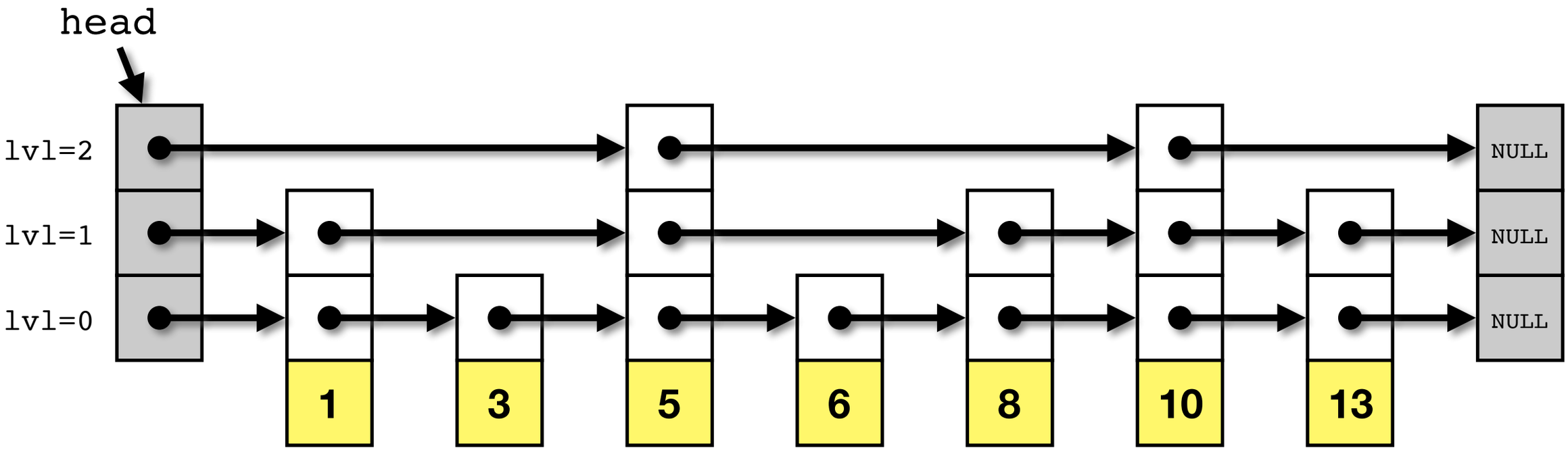
- **Example -- `find` the node with a key value 6**

(6) Check contents of current node to see if desired data was found. In this example, it was.



Skip List Searching

- **Another Example -- `find` the node with a key value 8**



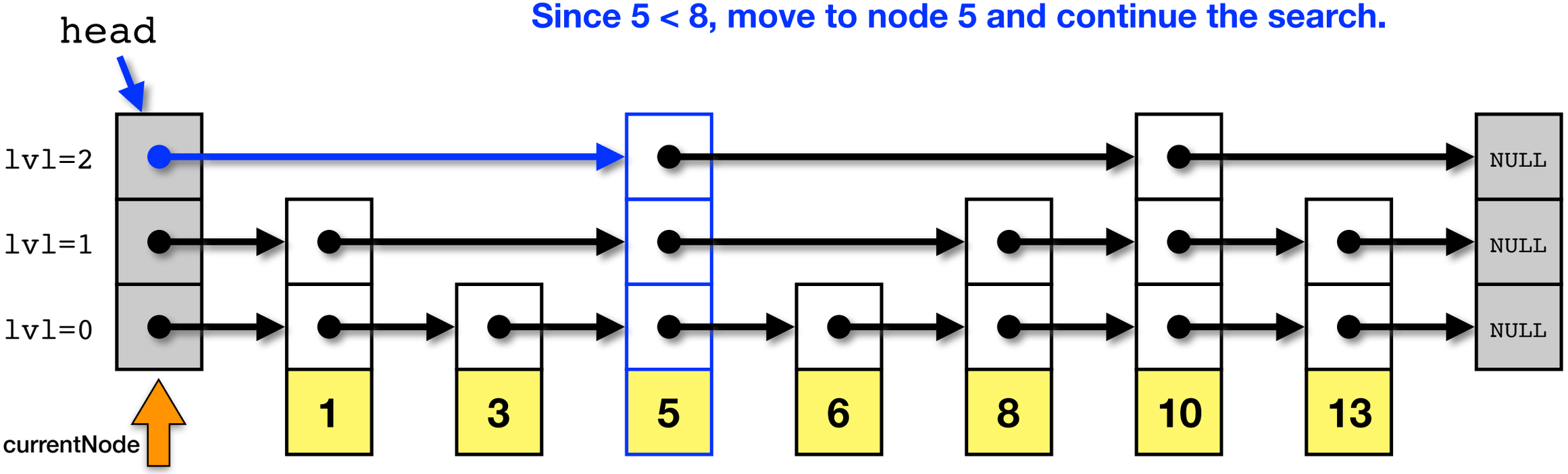
Skip List Searching

- **Another Example -- `find` the node with a key value 8**

(1) Start at highest level of the head node and check data value at node.next

(2) if node.next.data < desired value move to next node otherwise, drop down one level at current node.

Since 5 < 8, move to node 5 and continue the search.

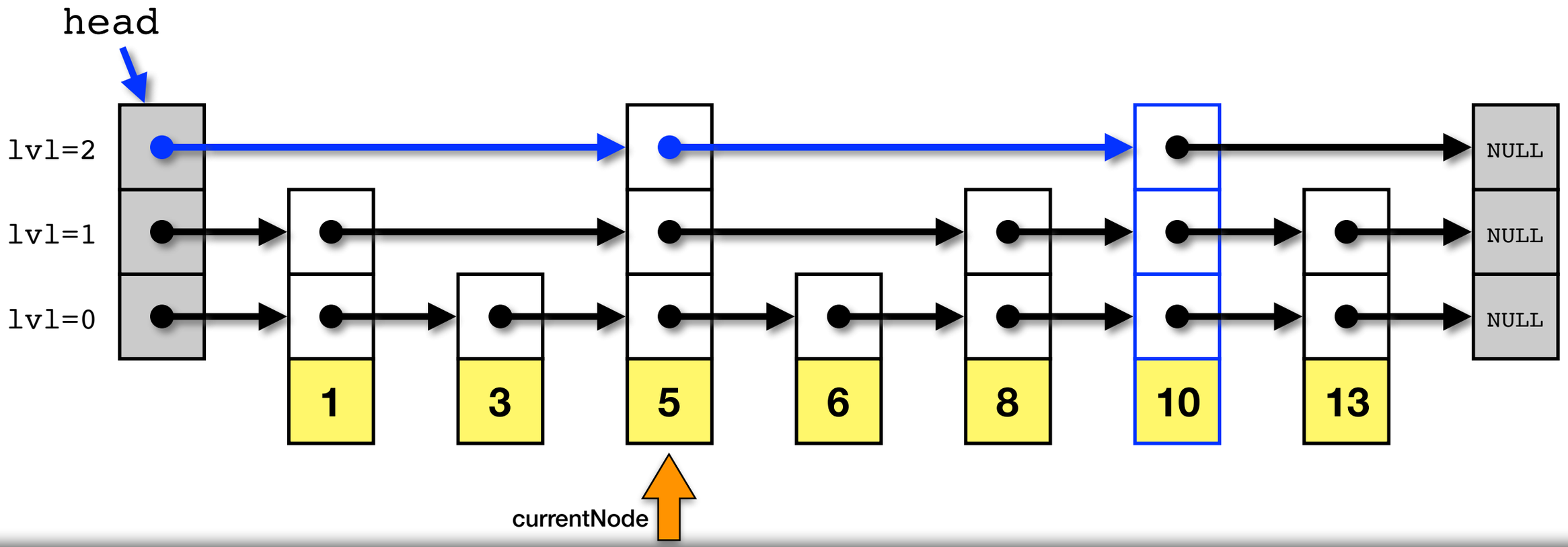


Skip List Searching

- **Another Example -- `find` the node with a key value 8**

(3) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

Since 10 !< 8, stay at node 5,
drop down one level and continue the search.

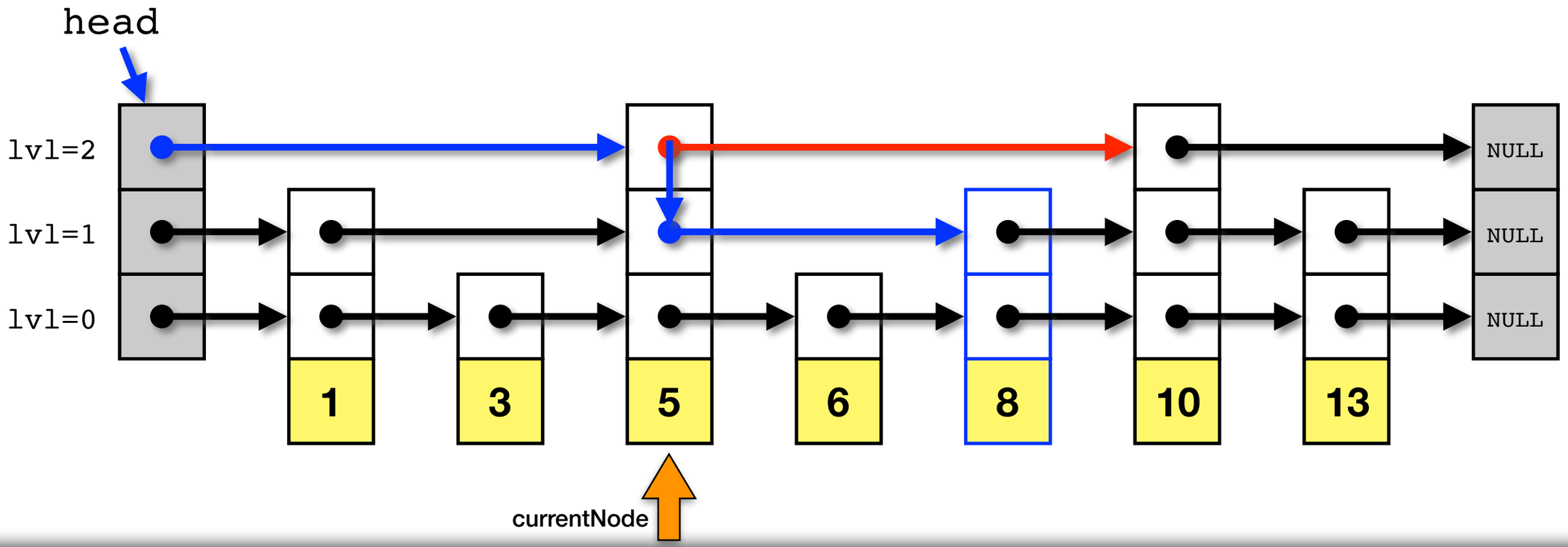


Skip List Searching

- **Another Example -- `find` the node with a key value 8**

(4) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

Since 8 !< 8, stay at node 5,
drop down one level and continue the search.

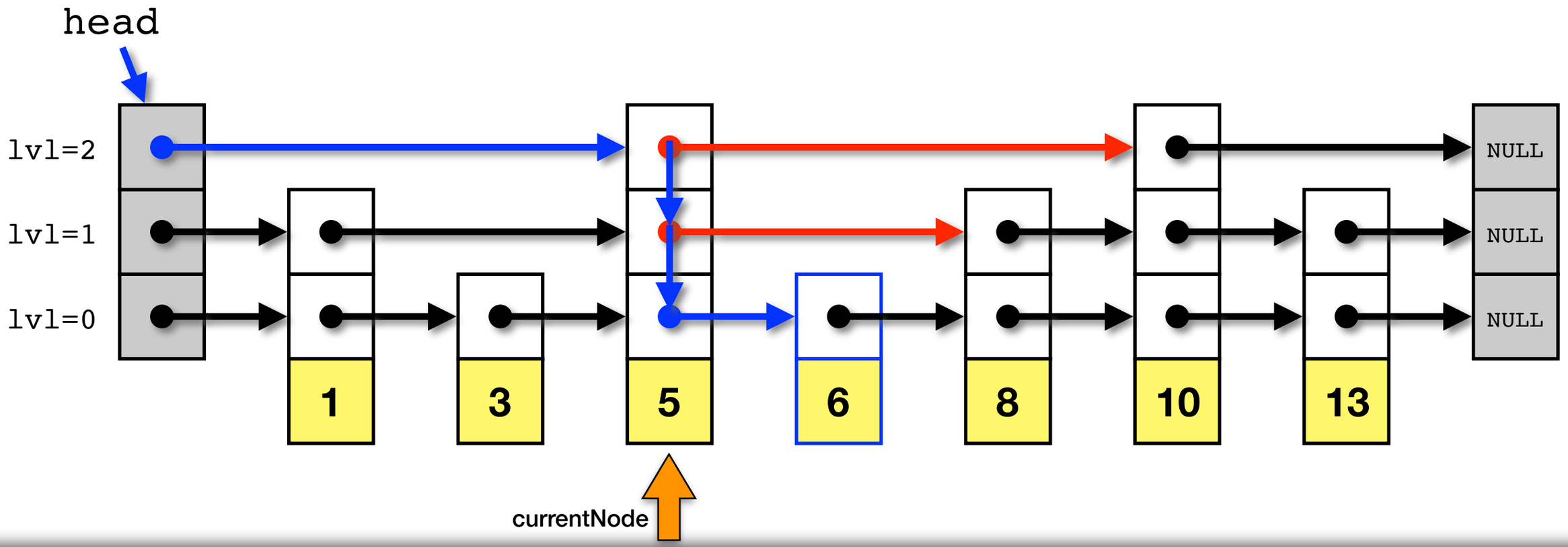


Skip List Searching

- **Another Example -- `find` the node with a key value 8**

(5) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

Since 6 < 8, move to node 6 and continue the search.

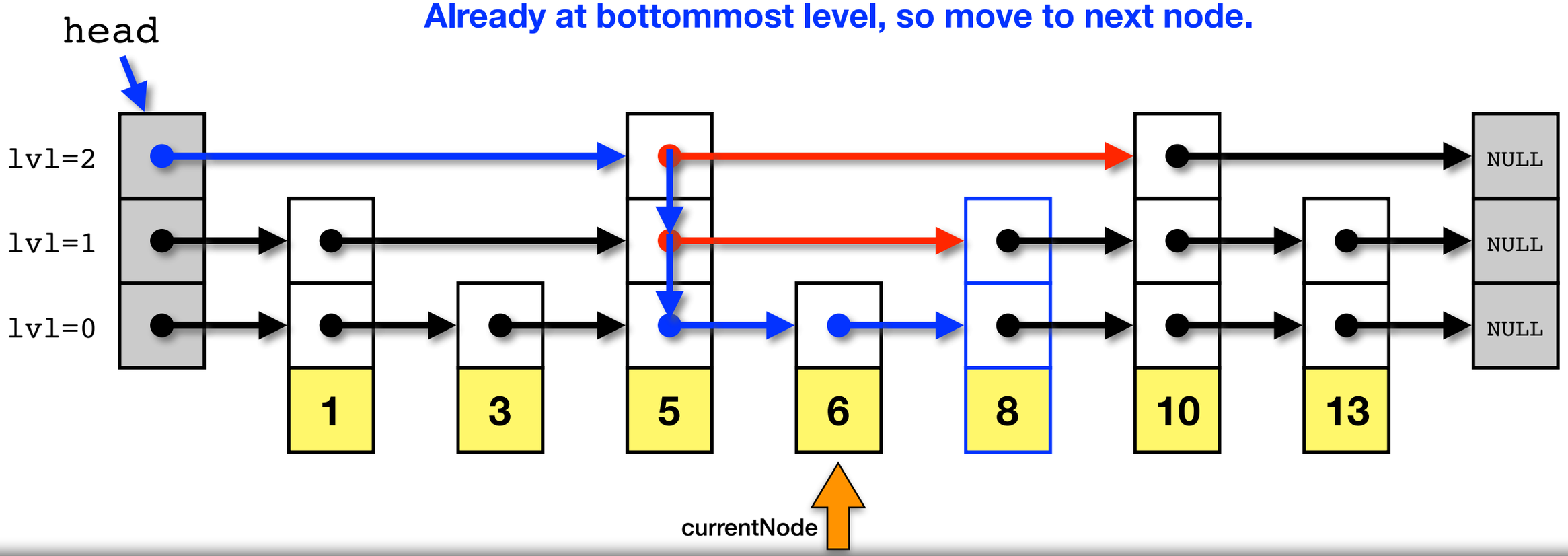


Skip List Searching

- **Another Example -- `find` the node with a key value 8**

(6) if node.next.data < desired value move to next node
otherwise, drop down one level at current node.

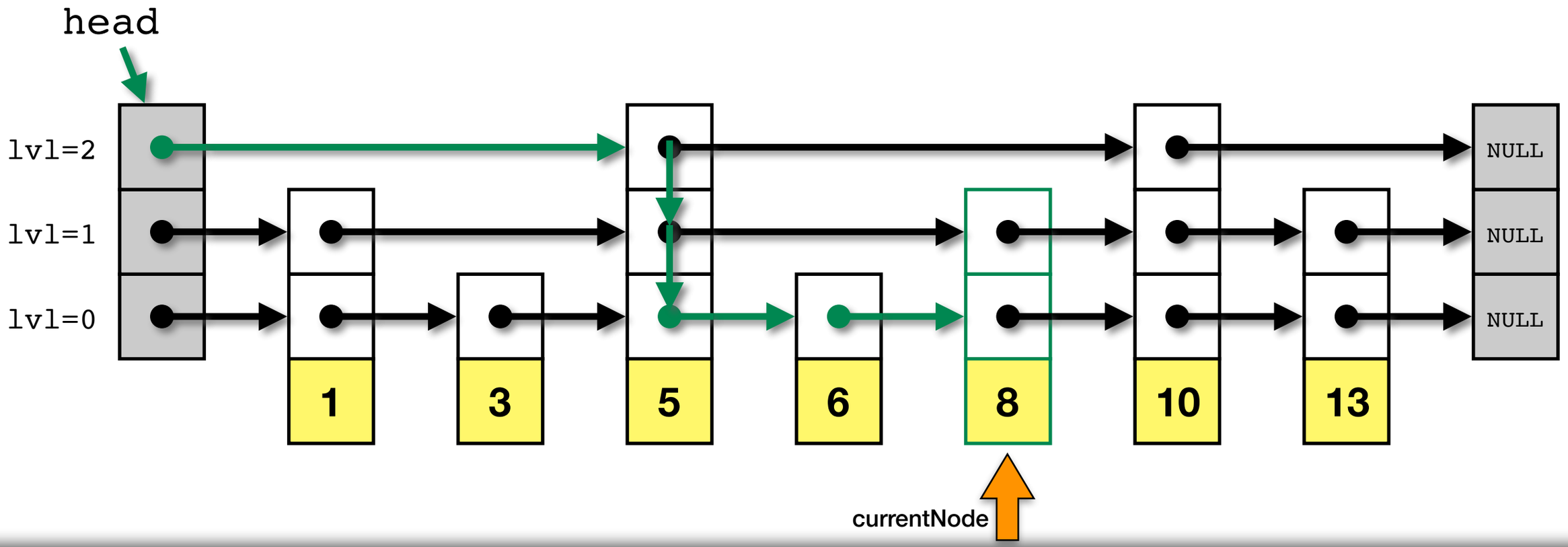
Since 8 !< 8, stay at node 6,
drop down one level and continue the search.
Already at bottommost level, so move to next node.



Skip List Searching

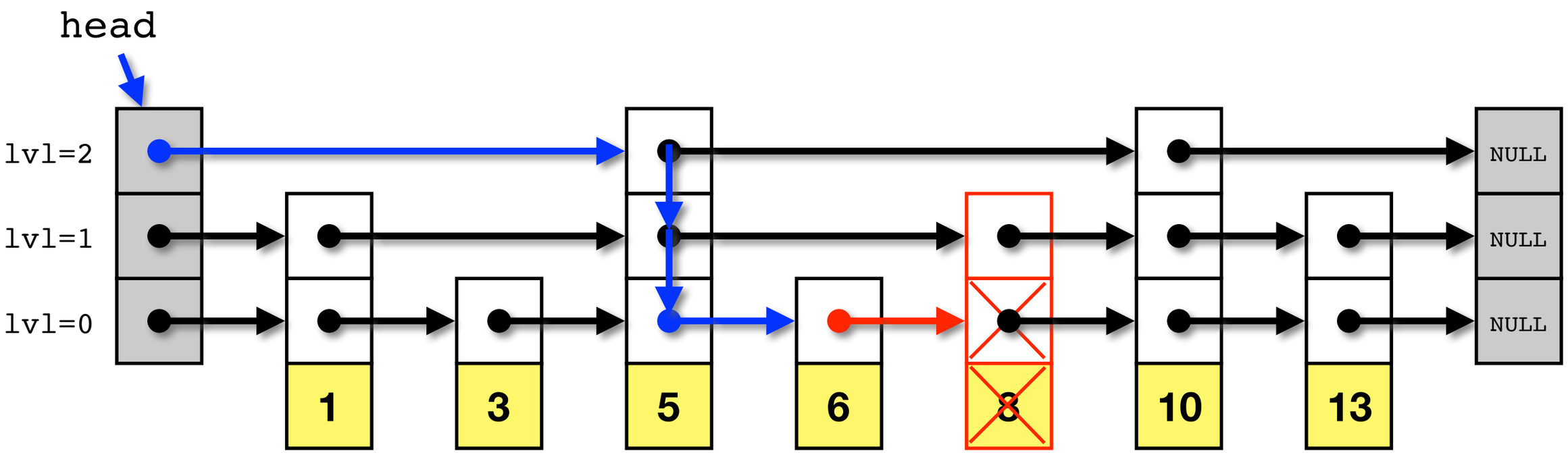
- **Another Example -- `find` the node with a key value 8**

(7) Check contents of current node
to see if desired data was found.
In this example, it was.



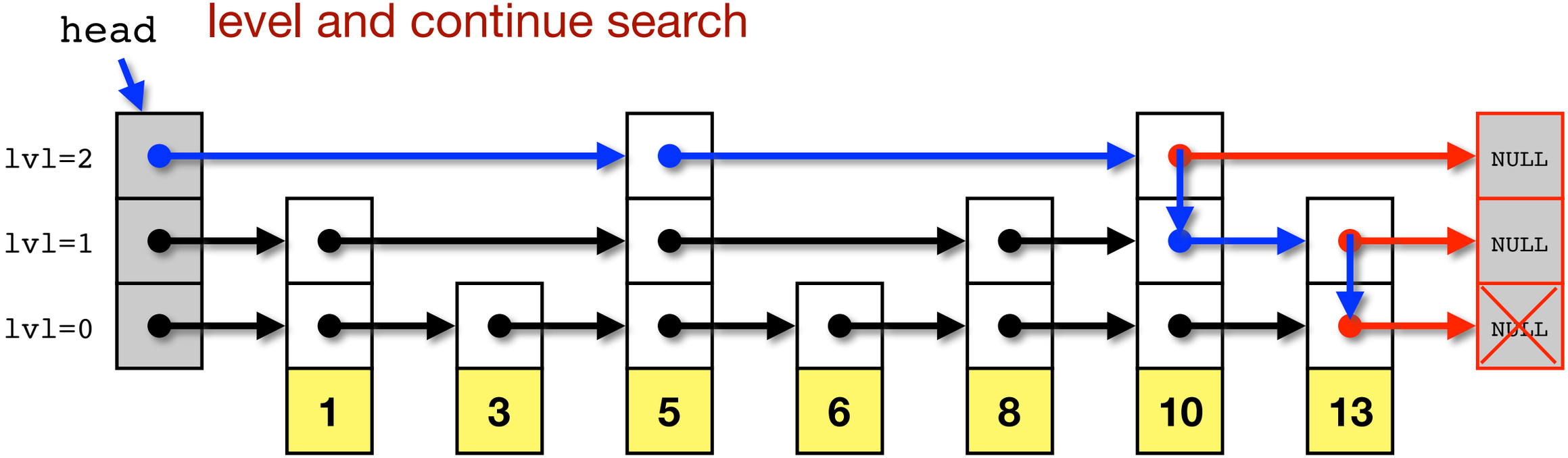
Skip List Searching

- **Example -- `find` the node with a key value 7**
  - In this example, the desired element does not exist in the list
  - Search the list just as before, but when a value is found that exceed that desired value, AND there are no more levels to drop, then the desired element doesn't exist



Skip List Searching

- **Example -- `find` the node with a key value 15**
  - Another example where the desired element does not exist in the list
  - Start with a normal search, if search reaches bottom-most level and next node is null, then element does not exist in list
    - If not at bottom-most level and node.next == NULL, then drop a level and continue search



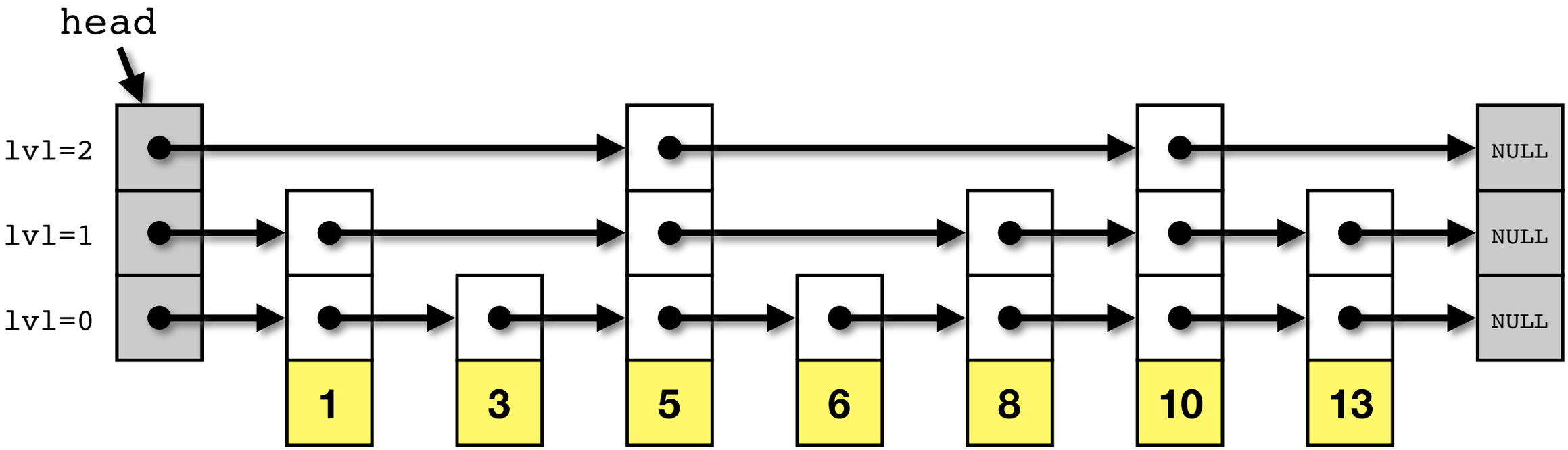
Skip List Insertion

Skip List Insertion

- **When inserting a node into a skip list, first perform a search to determine where the new node must be inserted**
  - Search for the value to be inserted
  - At each level, remember the rightmost node that is to the left where the new node will be inserted
    - i.e. Keep track of where the level is decremented during the search
  - If the value is found to already exist, don't insert duplicate
- **The height of a newly inserted node is chosen at random**
  - Flip a coin, if heads, then add an additional level to the newly inserted node, if tails then don't add anymore levels
    - Probability of 1/2 that newly inserted node will only be at the bottom-most level
    - Probability of 1/2 that newly inserted node will have one additional level (two total)
    - Probability of 1/4 that newly inserted node will have two additional levels (three total)
    - Probability of 1/8 that newly inserted node will have three additional levels (four total)
    - etc.

Skip List Insertion

- **Example -- `insert` the node with a key value 7**
- **Start with a search to determine where the node should be inserted**



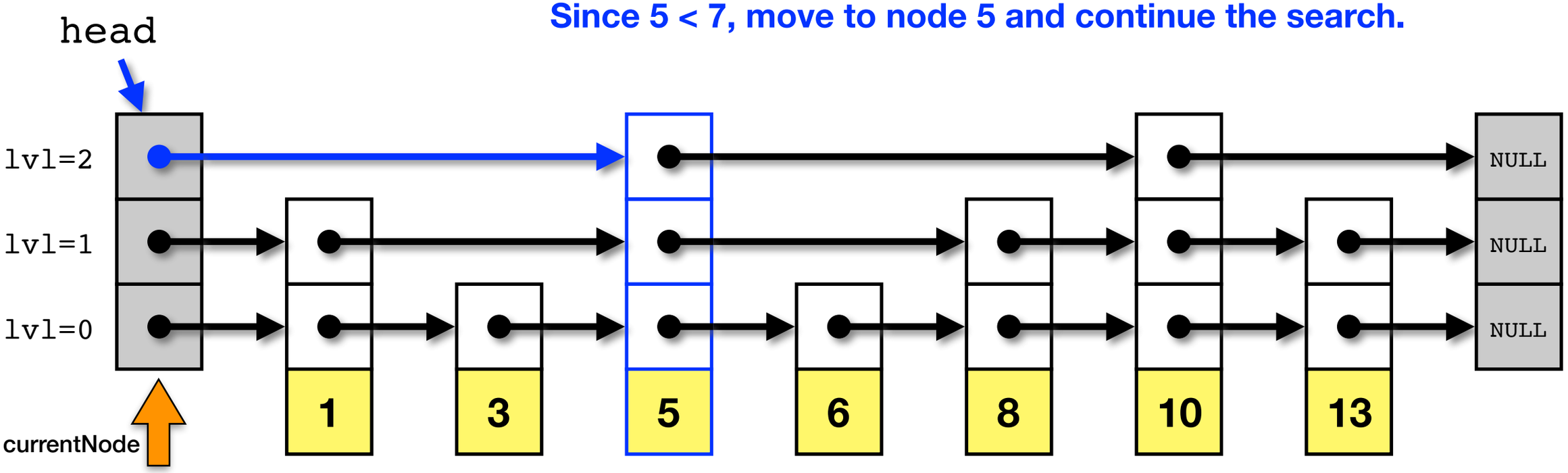
Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(1) Start at highest level of the head node and check data value at node.next

(2) if node.next.data < new value move to next node otherwise, drop down one level at current node.

Since 5 < 7, move to node 5 and continue the search.

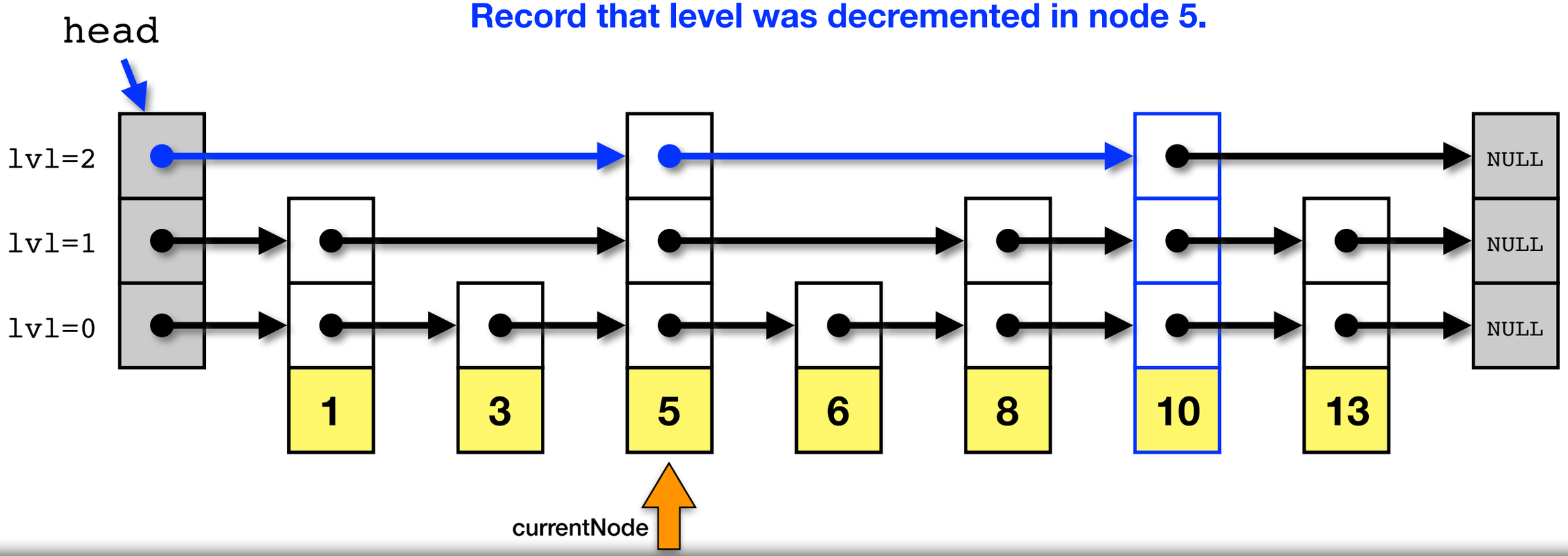


Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(3) if node.next.data < new value move to next node
otherwise, drop down one level at current node.

Since 10 !< 7, stay at node 5,
drop down one level and continue the search.
Record that level was decremented in node 5.

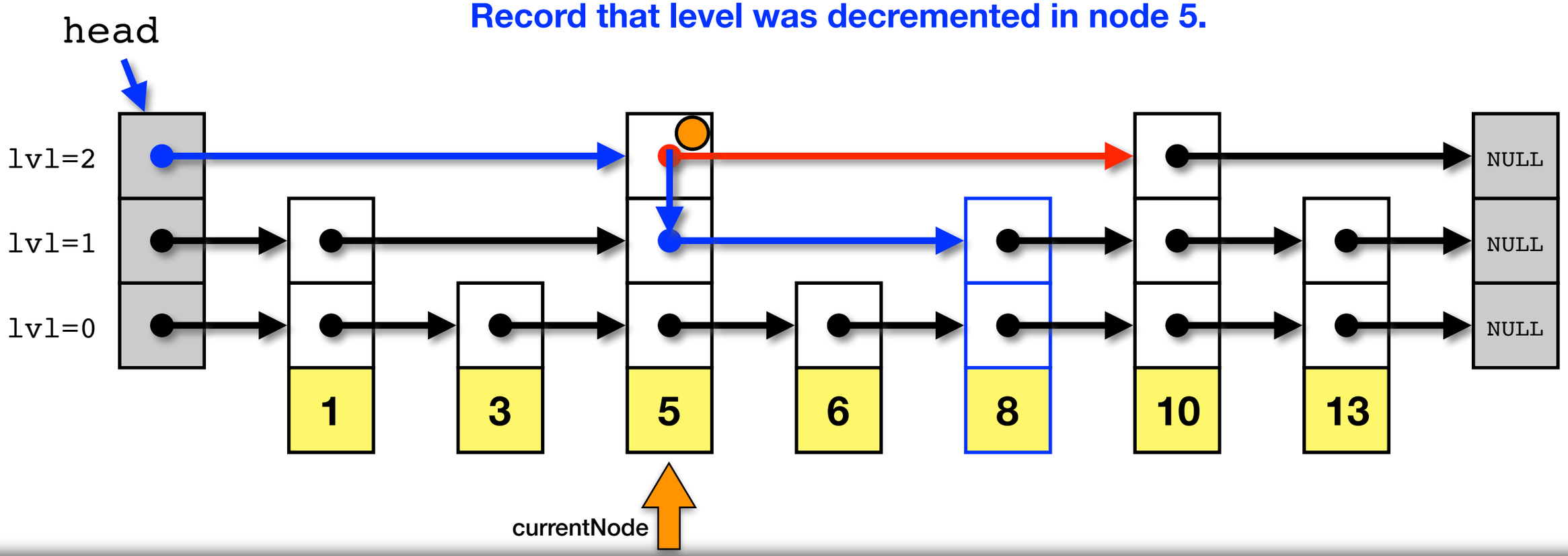


Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(4) if node.next.data < new value move to next node
otherwise, drop down one level at current node.

Since 8 !< 7, stay at node 5,
drop down one level and continue the search.
Record that level was decremented in node 5.

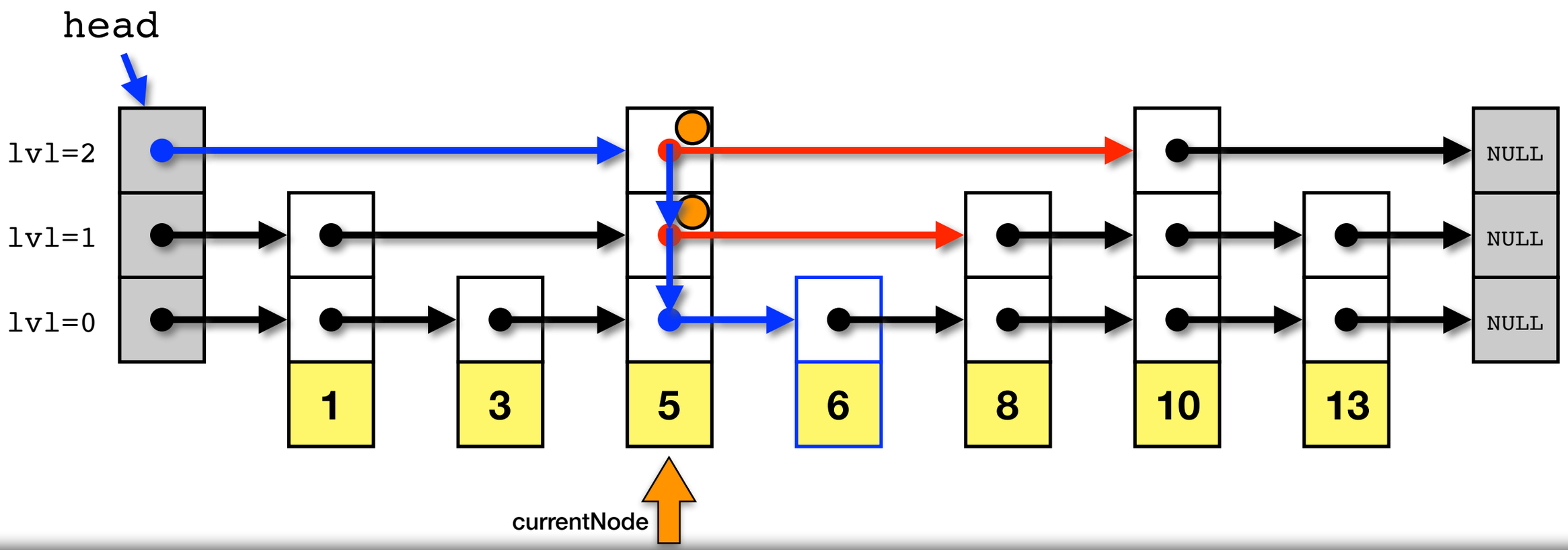


Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(5) if node.next.data < new value move to next node
otherwise, drop down one level at current node.

Since 6 < 7, move to node 6 and continue the search.

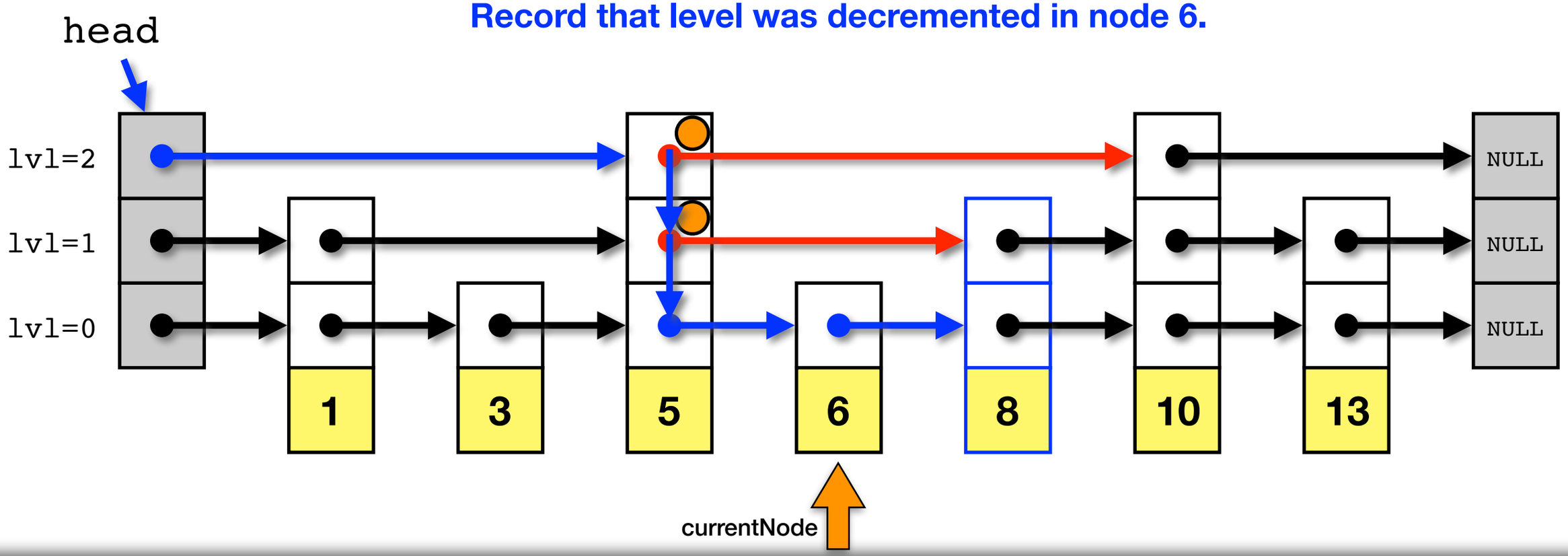


Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(6) if node.next.data < new value move to next node
otherwise, drop down one level at current node.

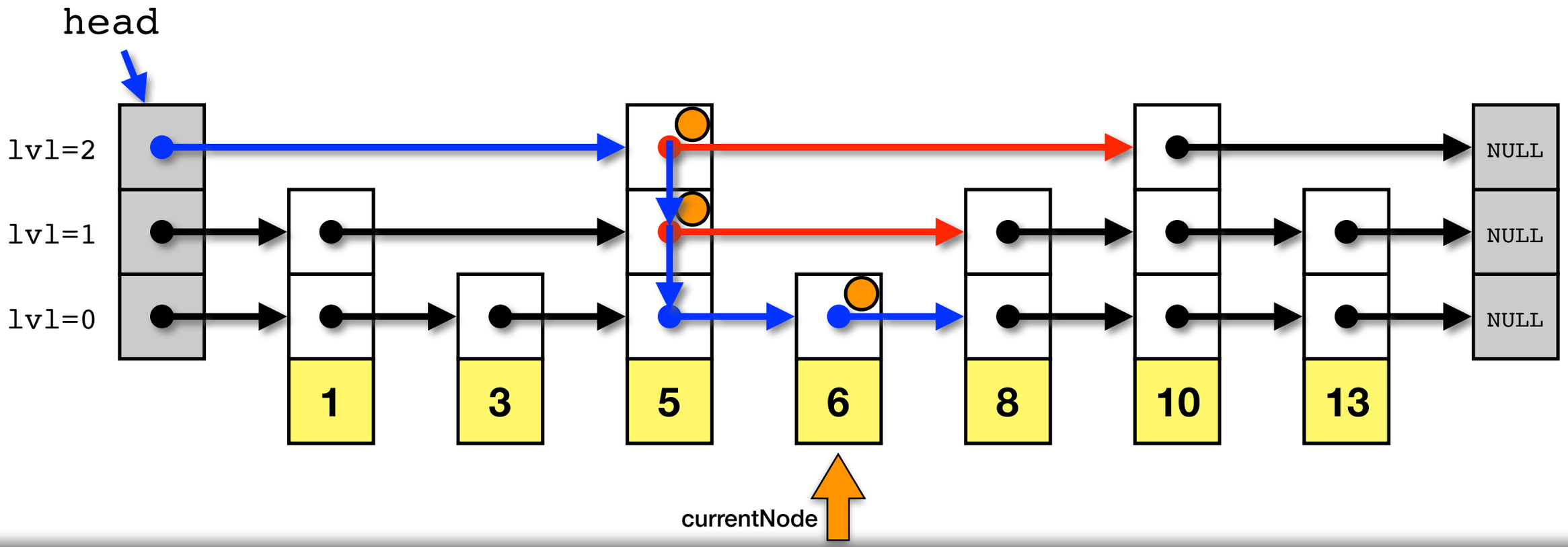
Since 8 !< 7, stay at node 6,
drop down one level and continue the search.
Record that level was decremented in node 6.



Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(7) At bottommost level, and can no longer decrement level.
Move to next node to finalize.



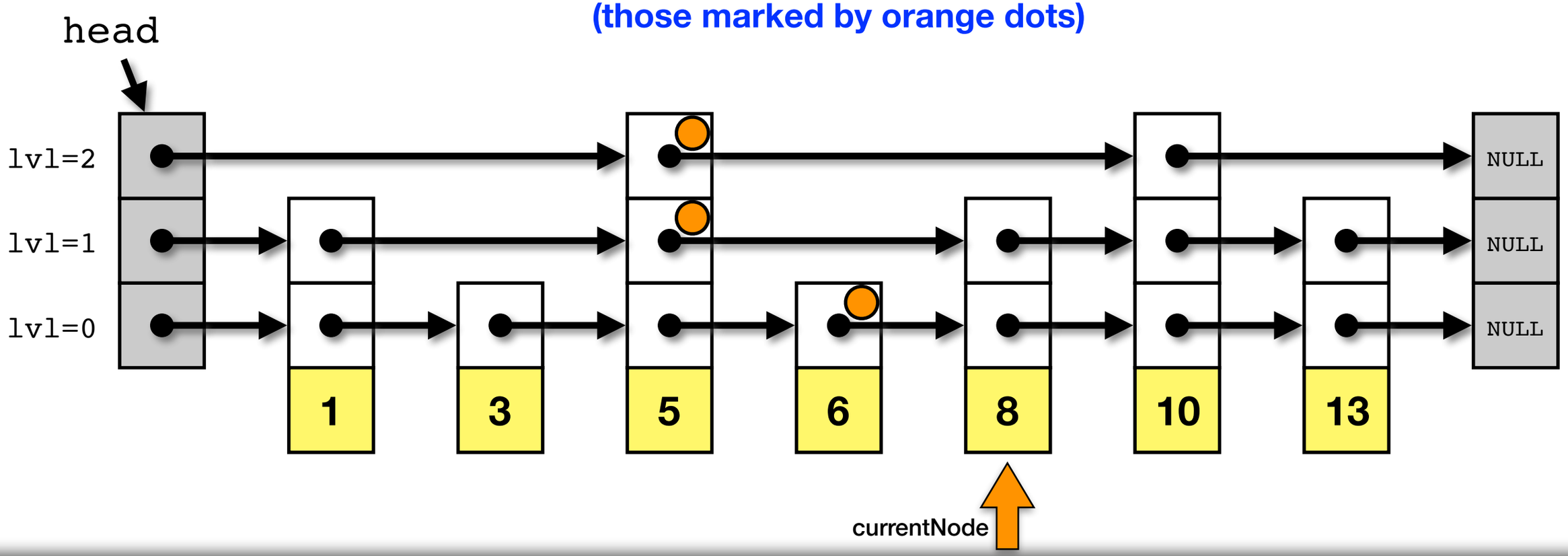
Skip List Insertion

- **Example -- `insert` the node with a key value 7**

(8) if the current node contains the same value as the new value, do nothing
otherwise, insert the new node to the left of the current node.

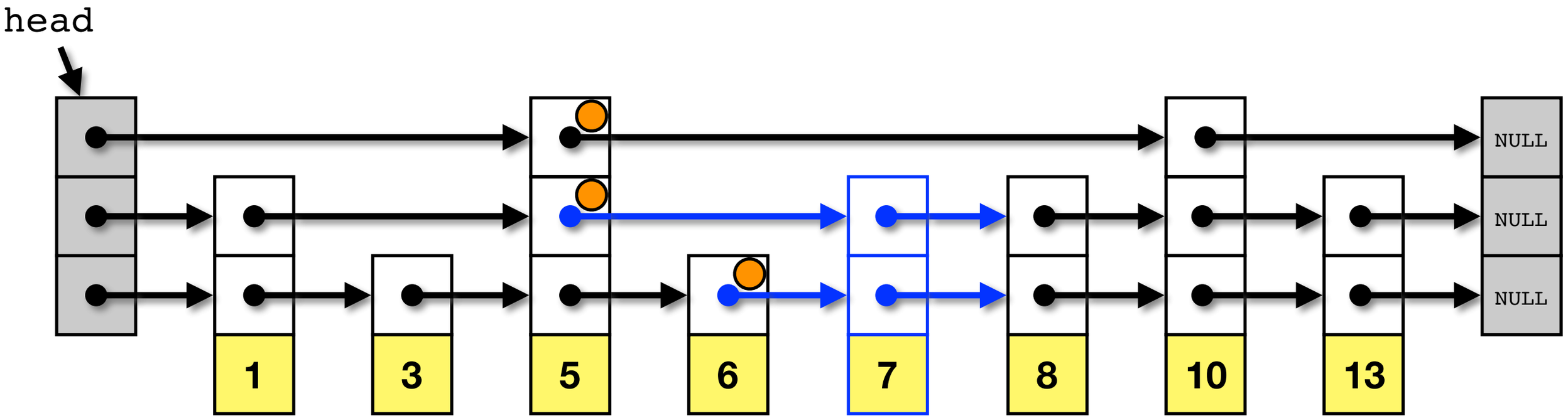
Since 8 != 7, must insert new node to left of node 8.

All pointers that MAY need to be updated have been recorded
(those marked by orange dots)



Skip List Insertion

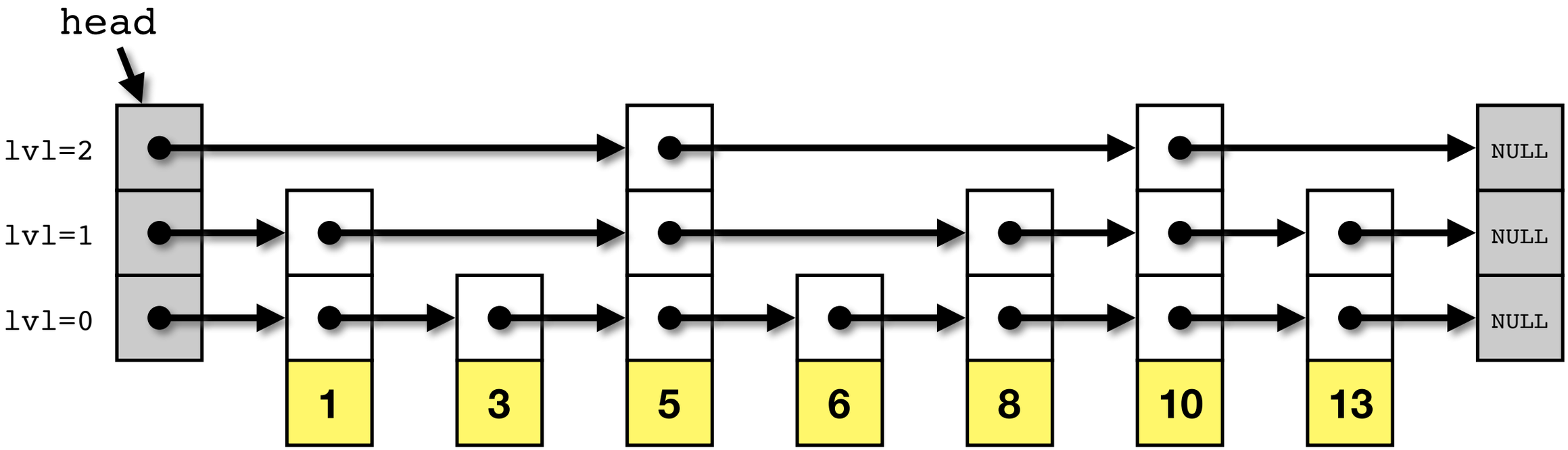
- **Example -- `insert` the node with a key value 7**
  - To **`insert`** node 7:
    - Randomly pick height for node 7
    - Markers were left behind to remember the rightmost node at each level to the left of the insertion location ... use them to update next pointers (all blue pointers must be updated)
    - Note that the topmost level of node 5 need not be updated since the newly created node 7 is shorter than node 5



Skip List Deletion

Skip List Deletion

- **For each level of which a skip list node is a part, splice the node out in the same fashion as a standard linked list**
- **Example -- `delete` the node with a key value 10**

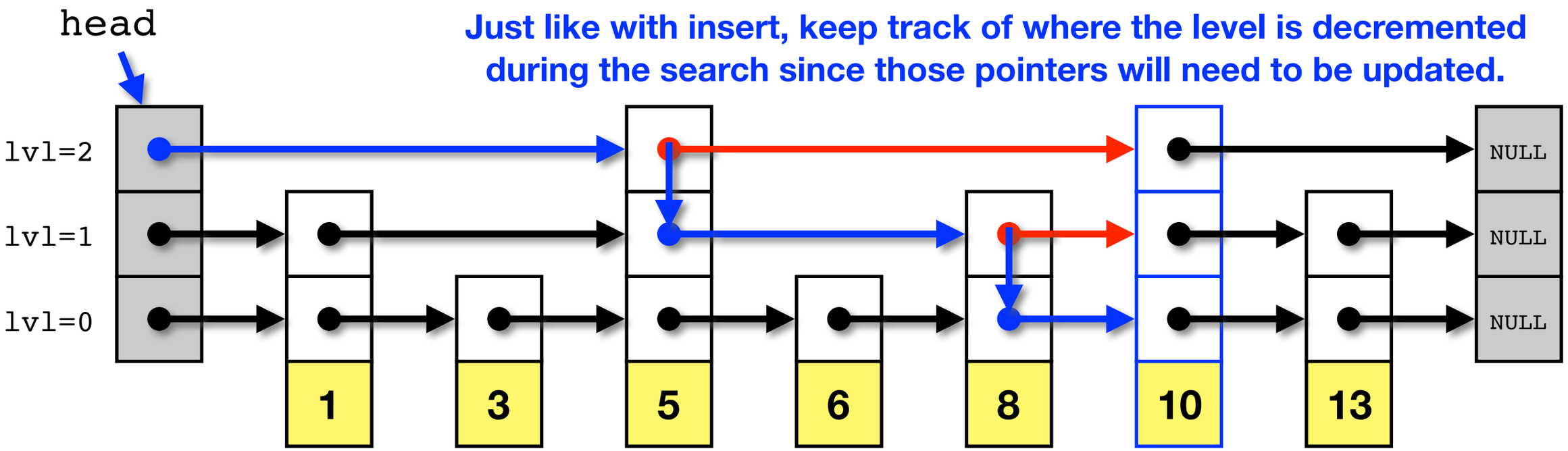


Skip List Deletion

- **For each level of which a skip list node is a part, splice the node out in the same fashion as a standard linked list**
- **Example -- `delete` the node with a key value 10**

First find the node to be deleted using the same search algorithm used for finding nodes.

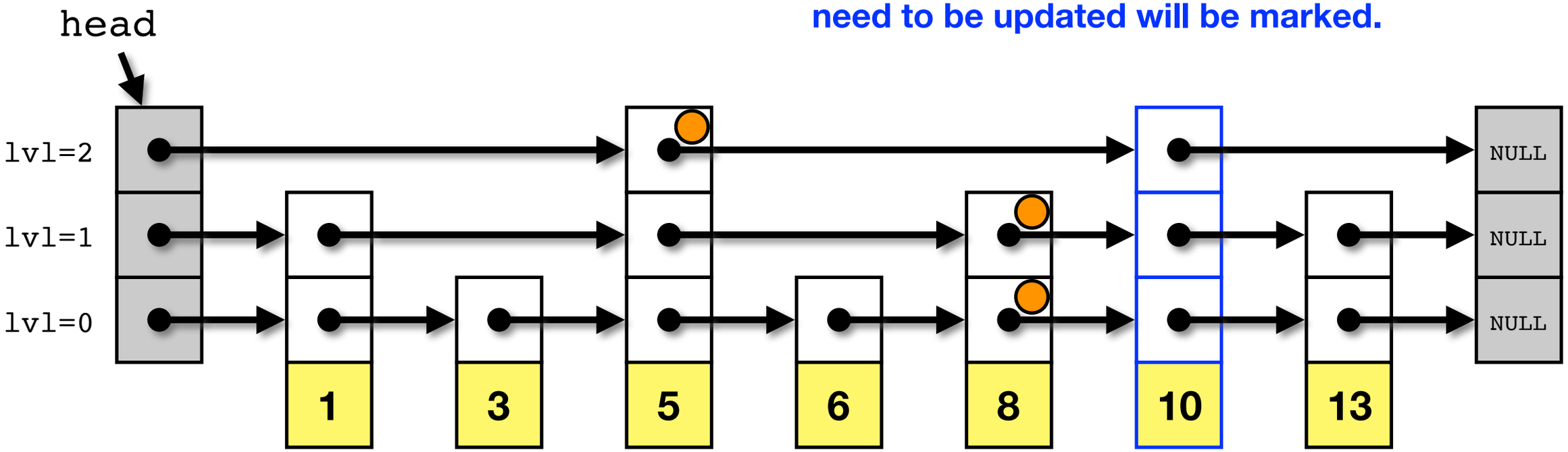
Just like with insert, keep track of where the level is decremented during the search since those pointers will need to be updated.



Skip List Deletion

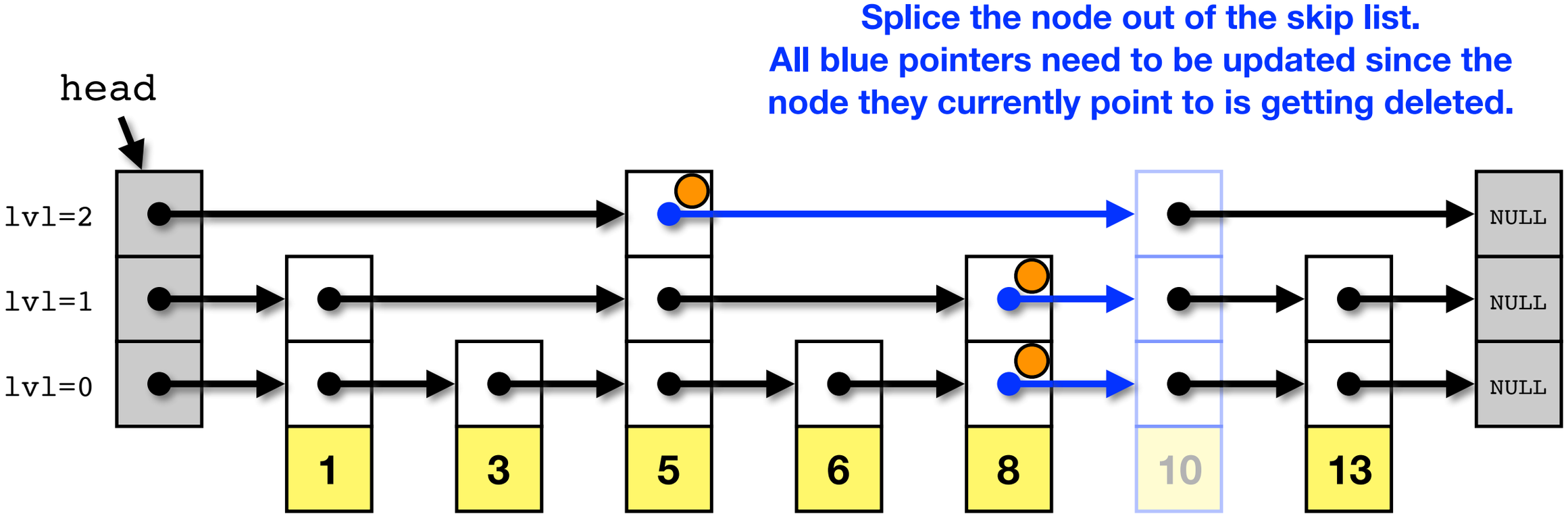
- **For each level of which a skip list node is a part, splice the node out in the same fashion as a standard linked list**
- **Example -- `delete` the node with a key value 10**

After finding the node to be deleted, all pointers that need to be updated will be marked.



Skip List Deletion

- **For each level of which a skip list node is a part, splice the node out in the same fashion as a standard linked list**
- **Example -- `delete` the node with a key value 10**



Skip List Deletion

- **For each level of which a skip list node is a part, splice the node out in the same fashion as a standard linked list**
- **Example -- `delete` the node with a key value 10**