B-Link trees

Concurrency

- is hard

- Use latches (short-term “locks”), not strict-2PL locks

- Readers go fast: no latch acquisition, no waiting

- Implication: writers must leave tree in valid state *at any point* - readers don’t respect latches!

Concurrency

- Processes go down and to the right
- Insertions expand the tree “rightward” from point of insertion
    - Might also expand parents “rightward” if recursive calls required

Reads

- Standard B-tree recursive read, but also check high-key
- while not leaf: check node
    - search item > high key? go right
    - otherwise, pick pointer as normal and go down

Writes

- No split - same as normal insert (latch and insert to leaf)

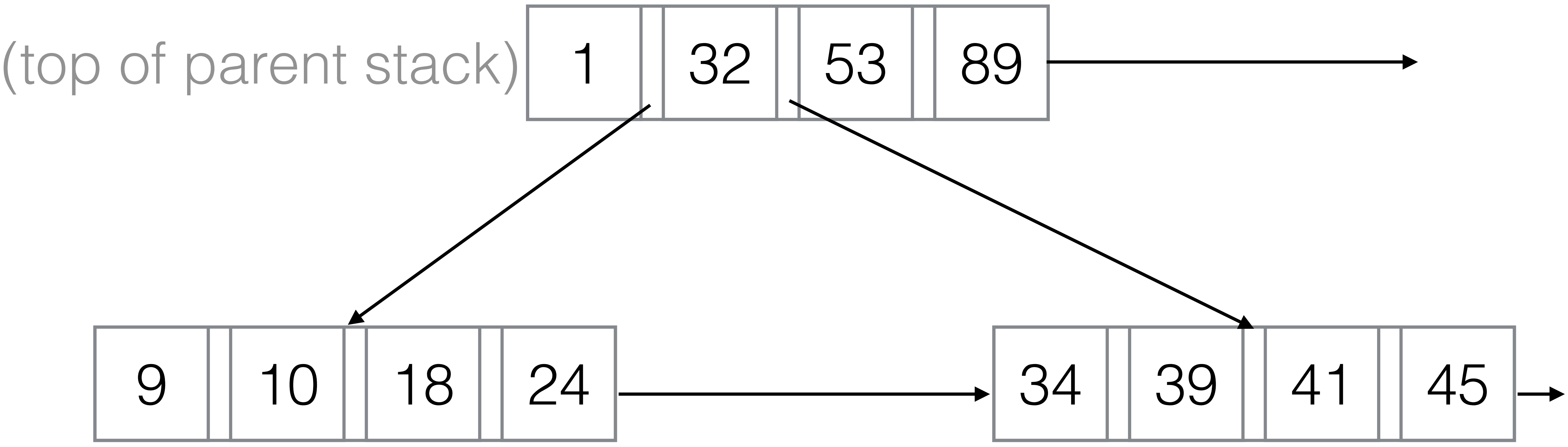
- Splits can be recursive - might need to set many latches!

    - Want to minimize # latches held at any point (will achieve maximum of 3)

    - Want to avoid holding latches during I/O, if possible

Writes - split

0. Search to get to the correct leaf node, keeping stack of parents

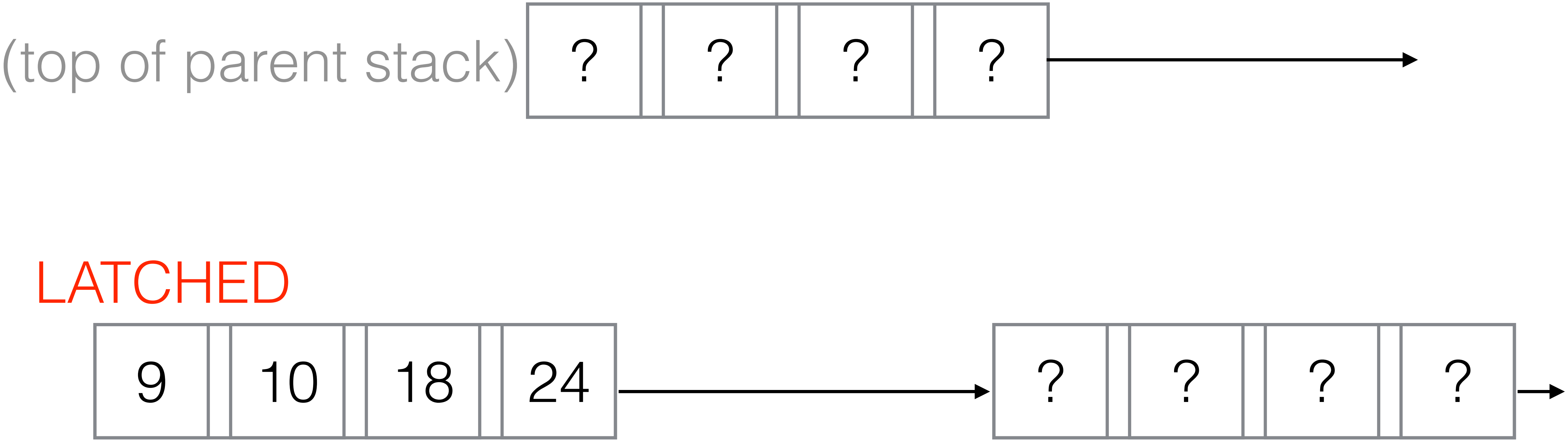


note: high keys not shown

Insert 15

Writes - split

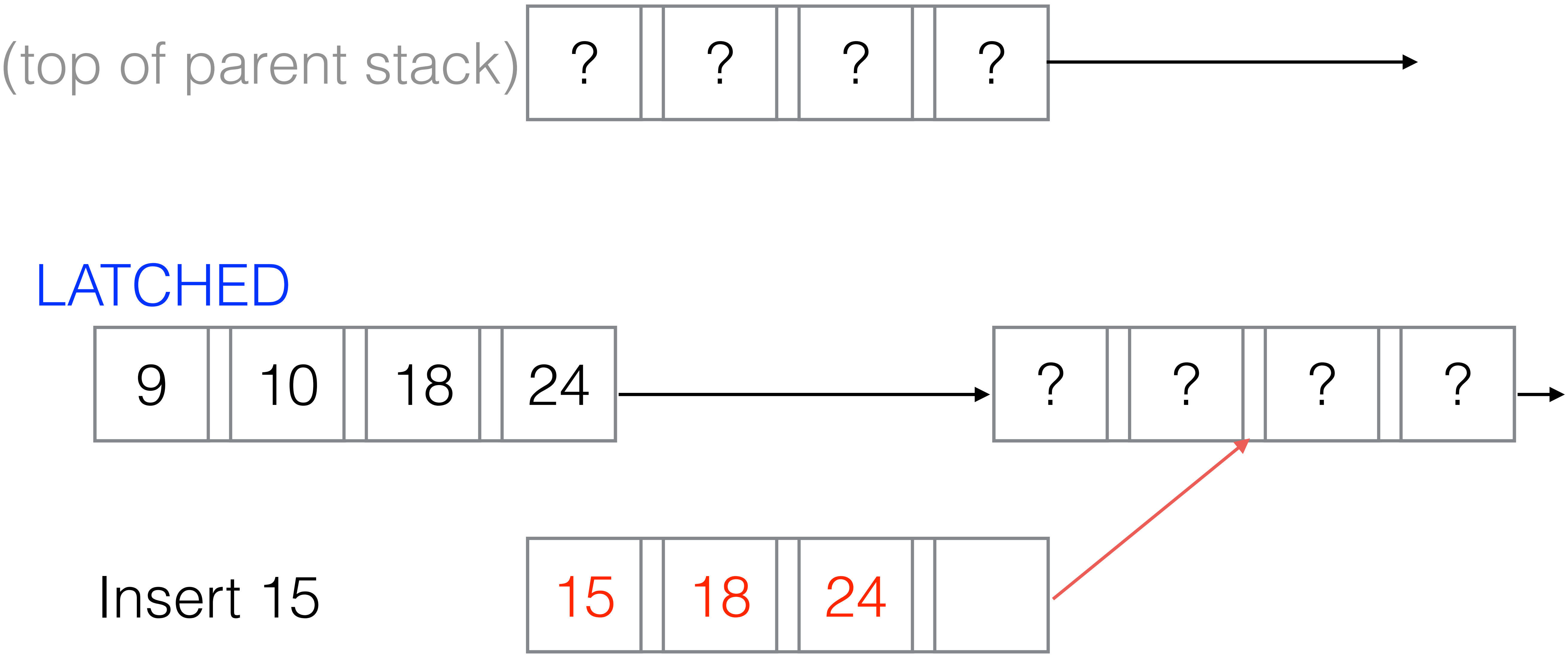
1. Acquire latch on node to be split.



Insert 15

Writes - split

2. Create and write new twin node.
Note that *it is not yet visible*! Nothing points to it.



(top of parent stack)

? ? ? ?

LATCHED

9 10 18 24

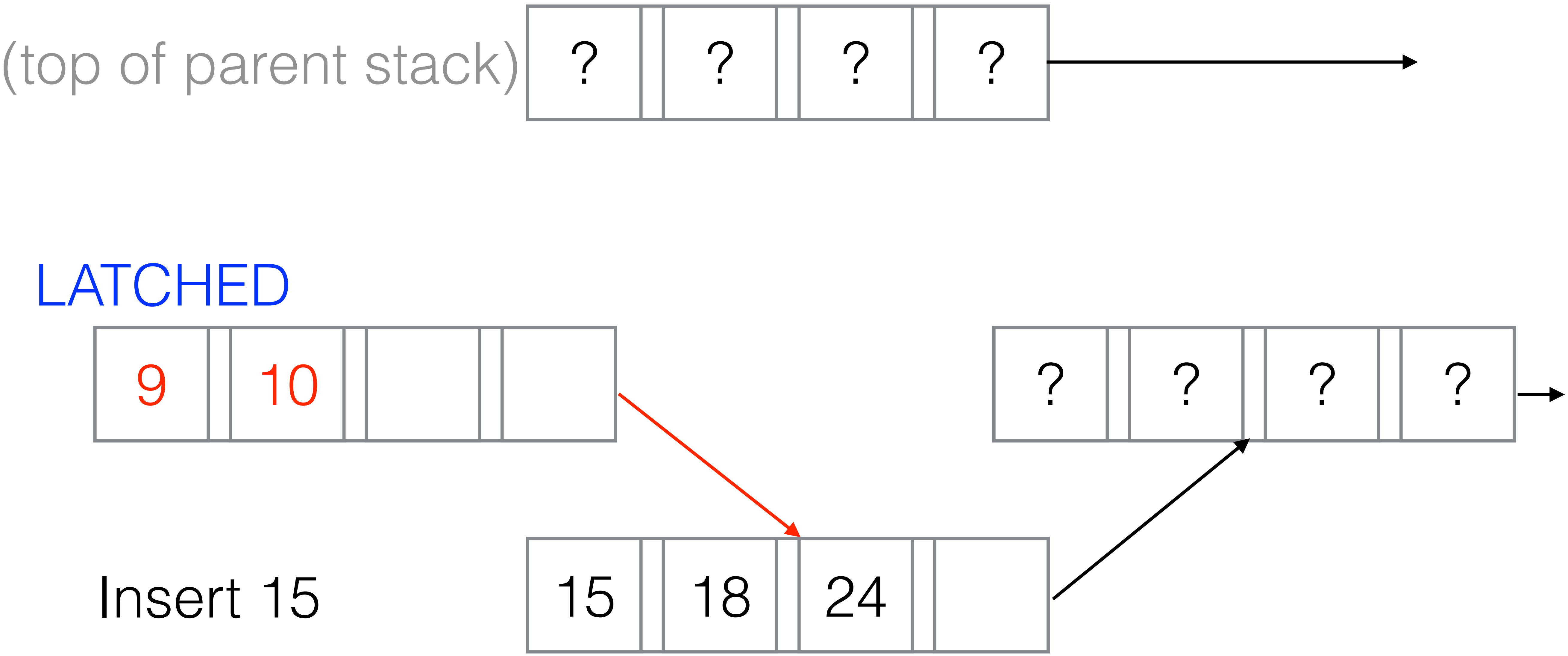
? ? ? ?

Insert 15

15 18 24

Writes - split

3. Write updated original twin node.



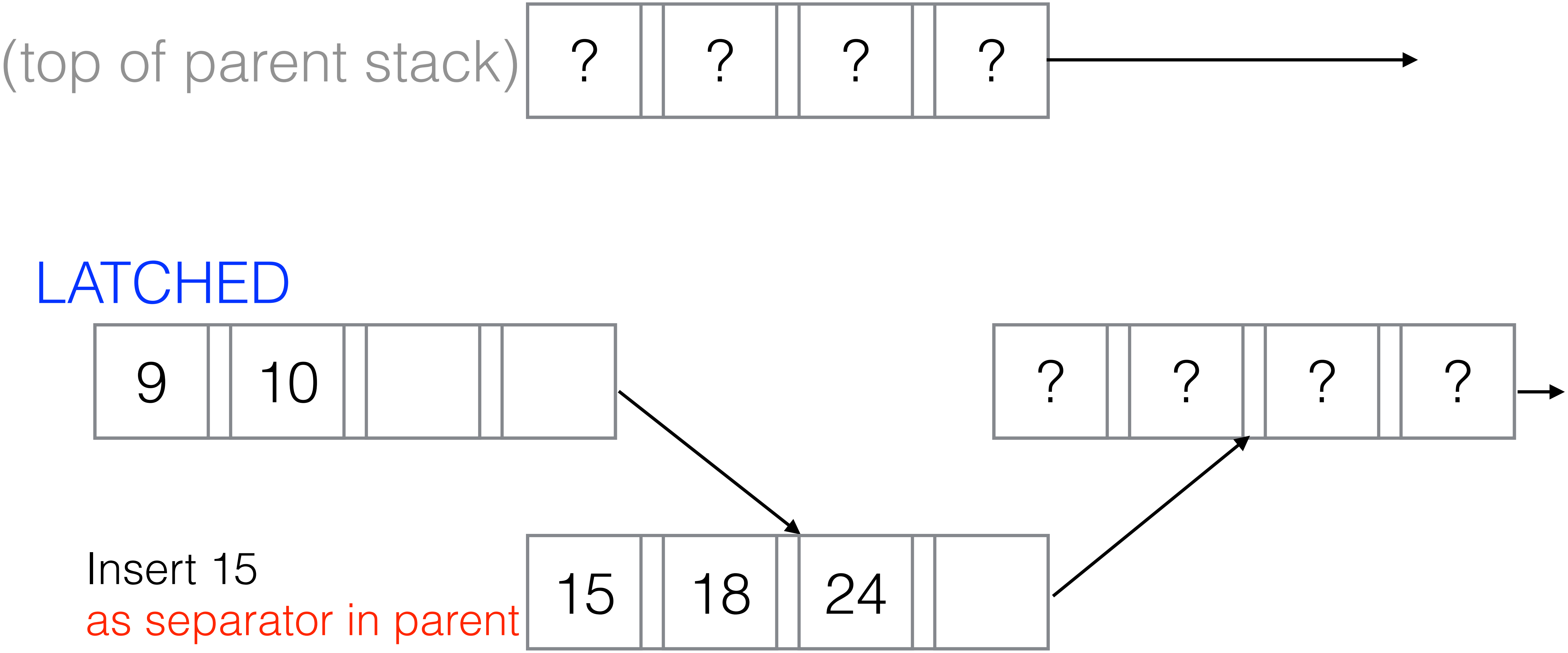
(top of parent stack)

LATCHED

Insert 15

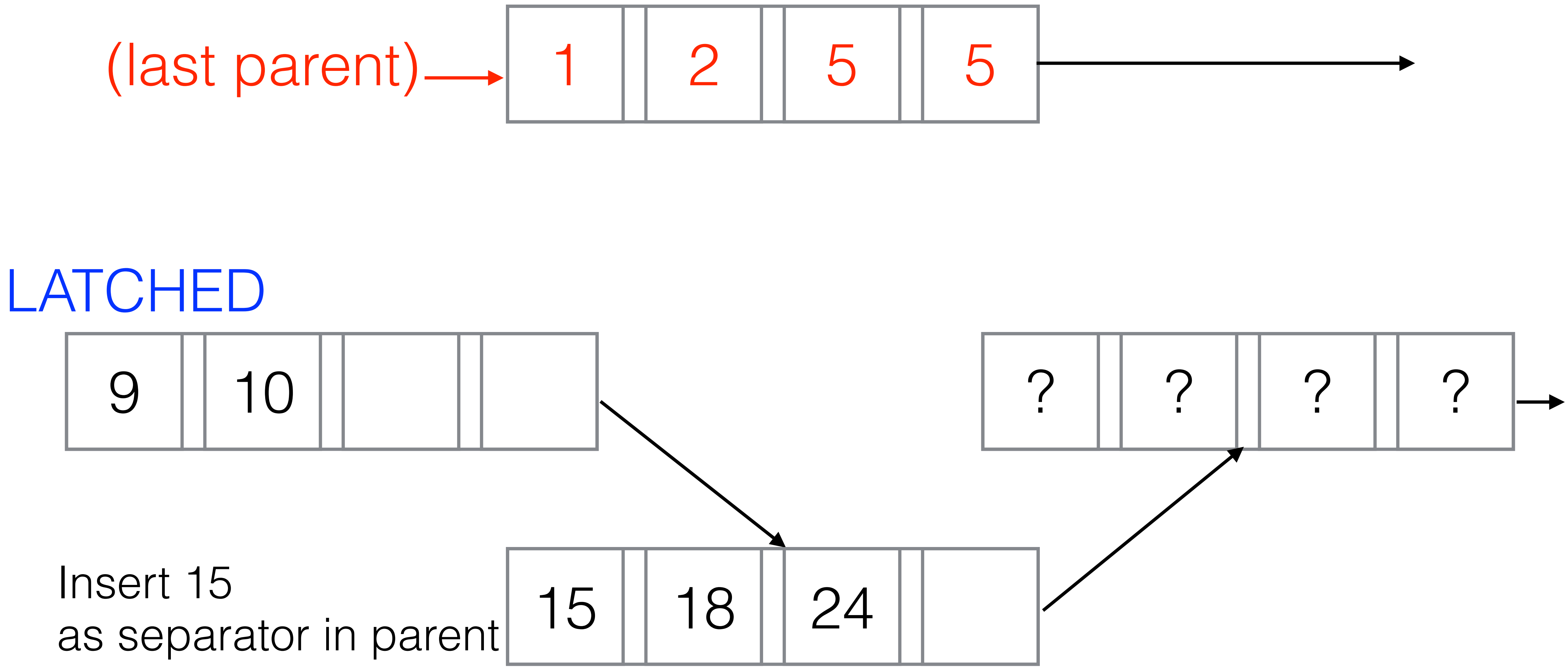
Writes - split

At this point, tree is "consistent", but if we stop now it we eventually degenerate into a linked list - need to add pointer from parent to new node!



Writes - split

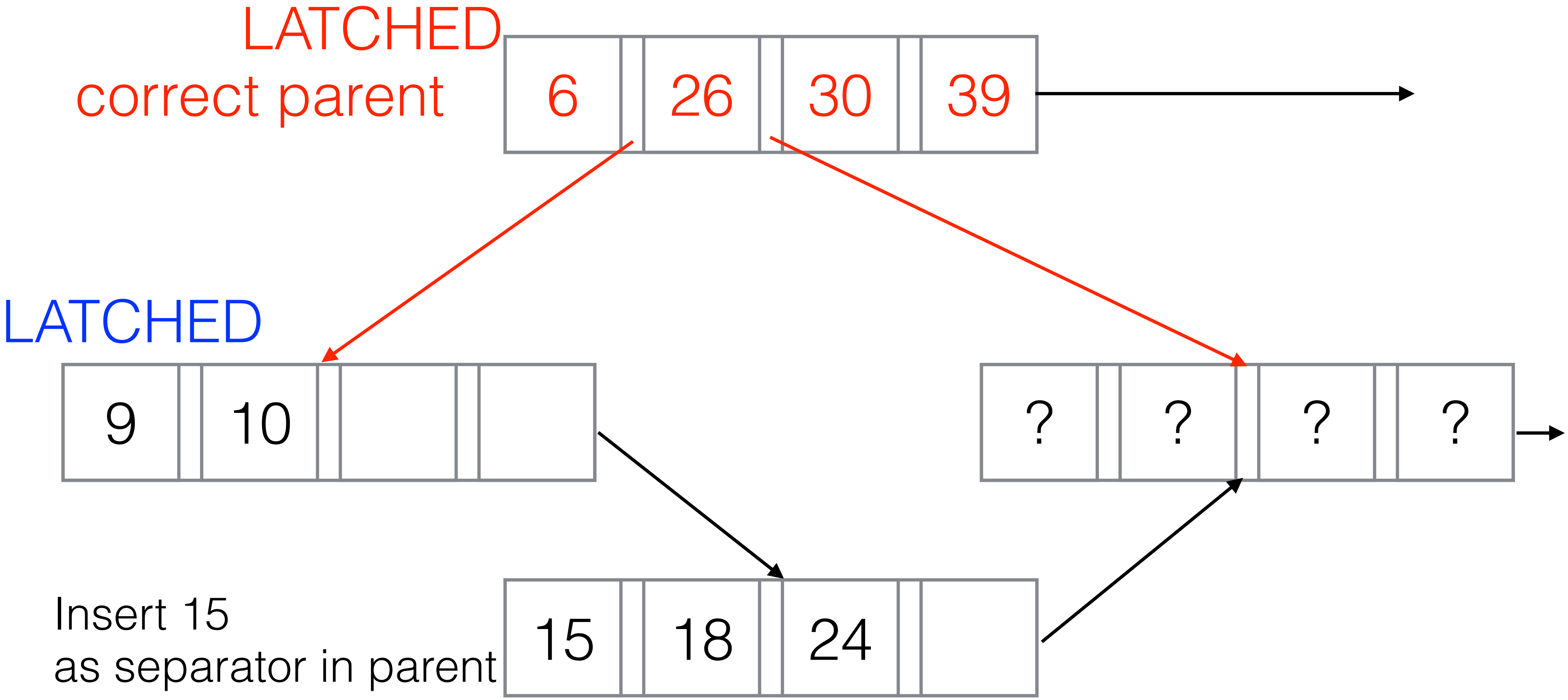
4. Re-read parent (found by popping off from our stack).
Parent might have changed! Want *correct* parent pointing to our node.



Insert 15
as separator in parent

Writes - split

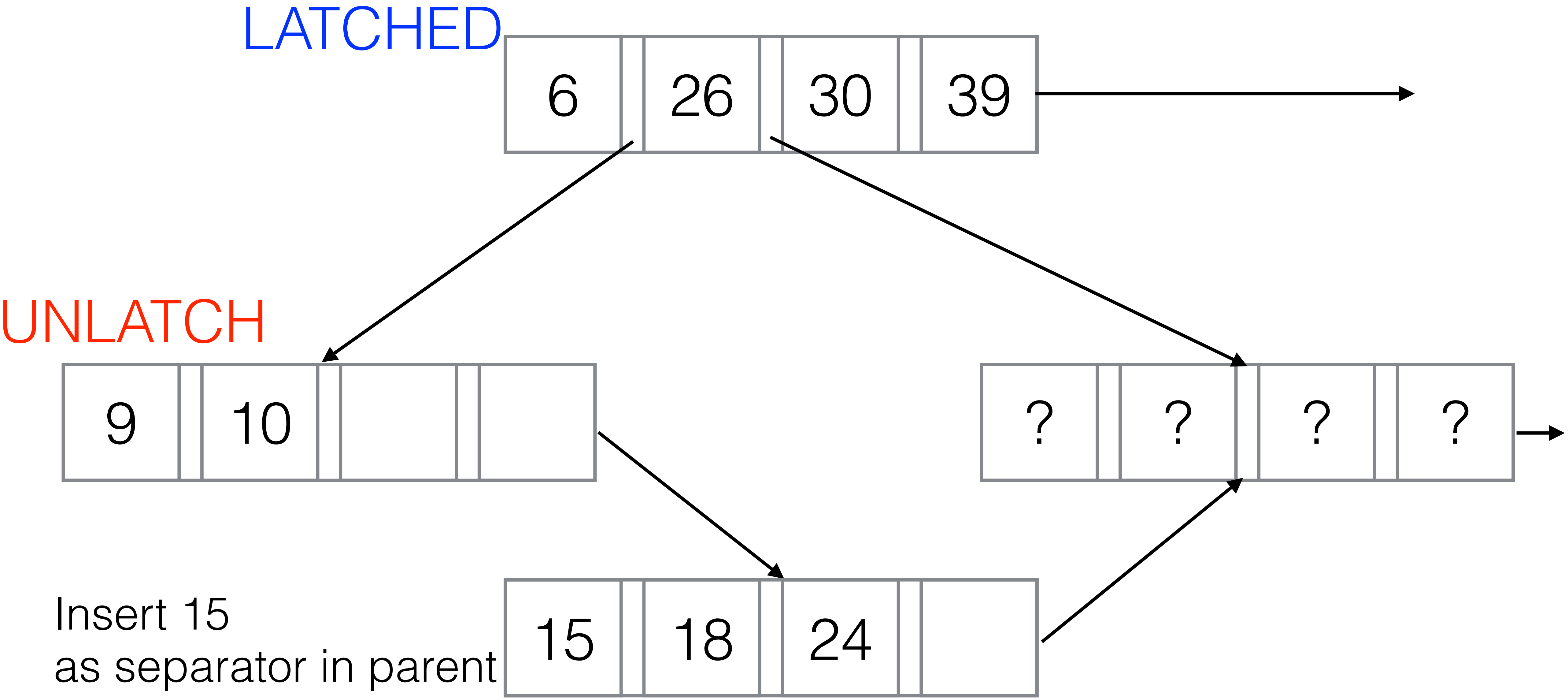
4a. While we haven't found parent pointing to us: move right. Acquire and release latches node by node. (cf. `move_right()` below for details.)



Insert 15
as separator in parent

Writes - split

5. We can finally release our latch (why?).



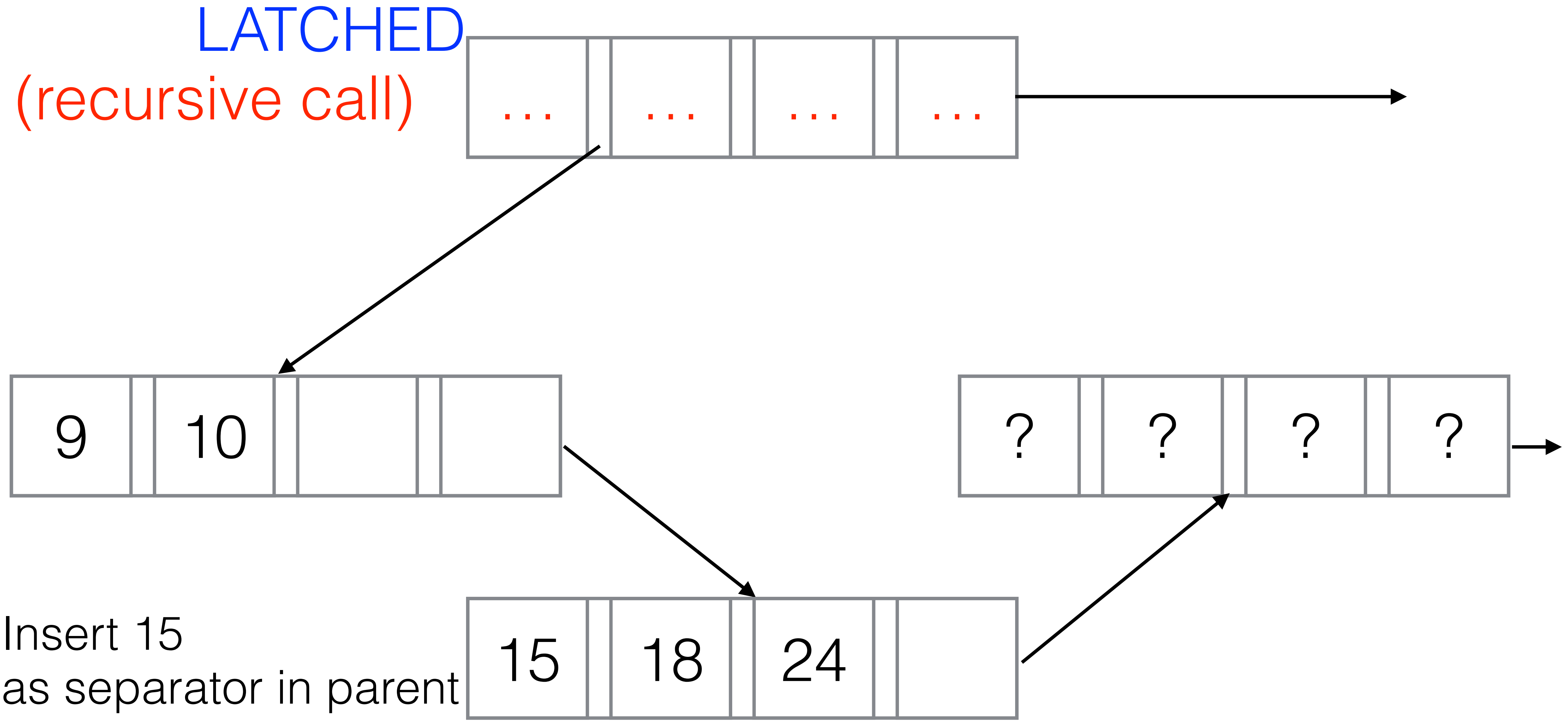
LATCHED

UNLATCH

Insert 15
as separator in parent

Writes - split

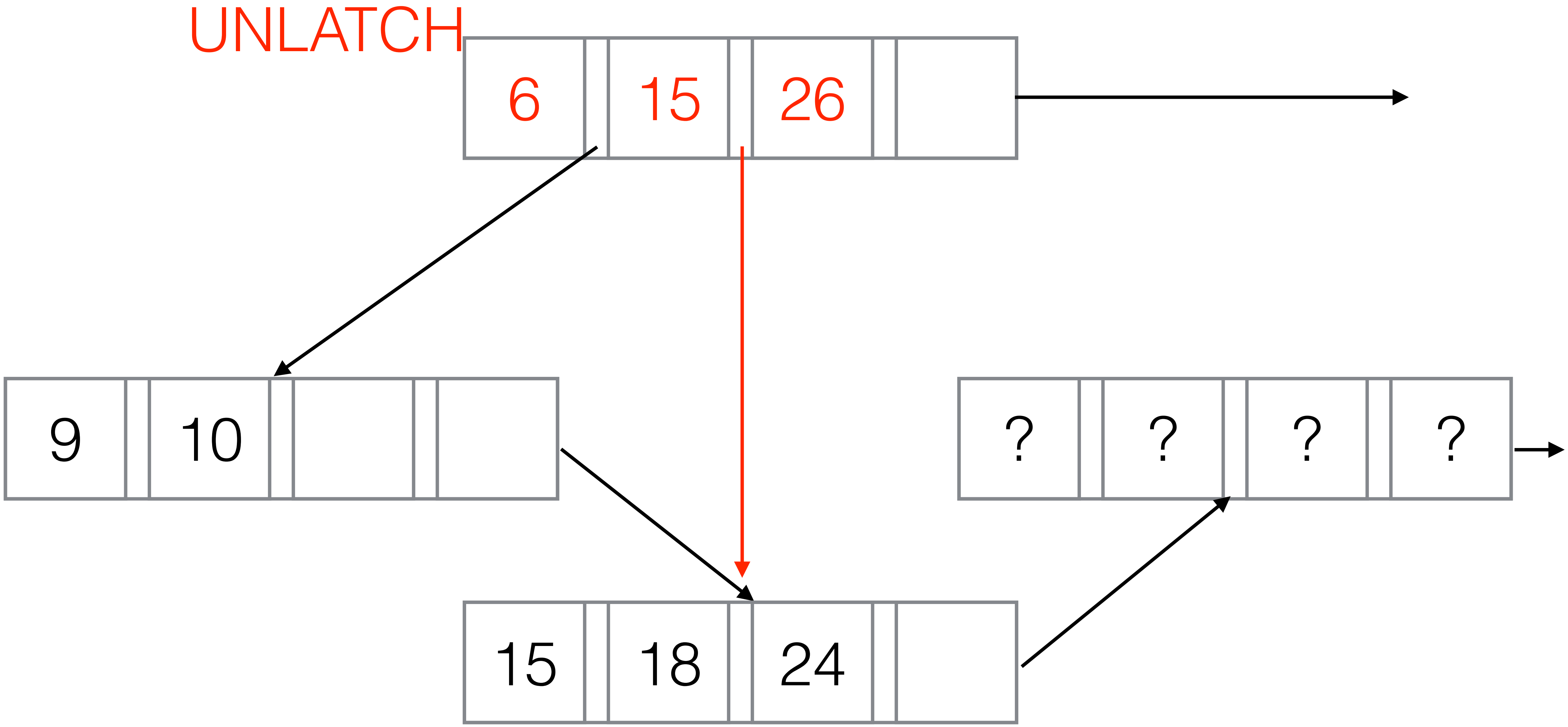
6. Insert separator (along with pointer) into parent. **May need to split that, too!** Call recursively (go back to step 1). (When do we stop?)



Insert 15
as separator in parent

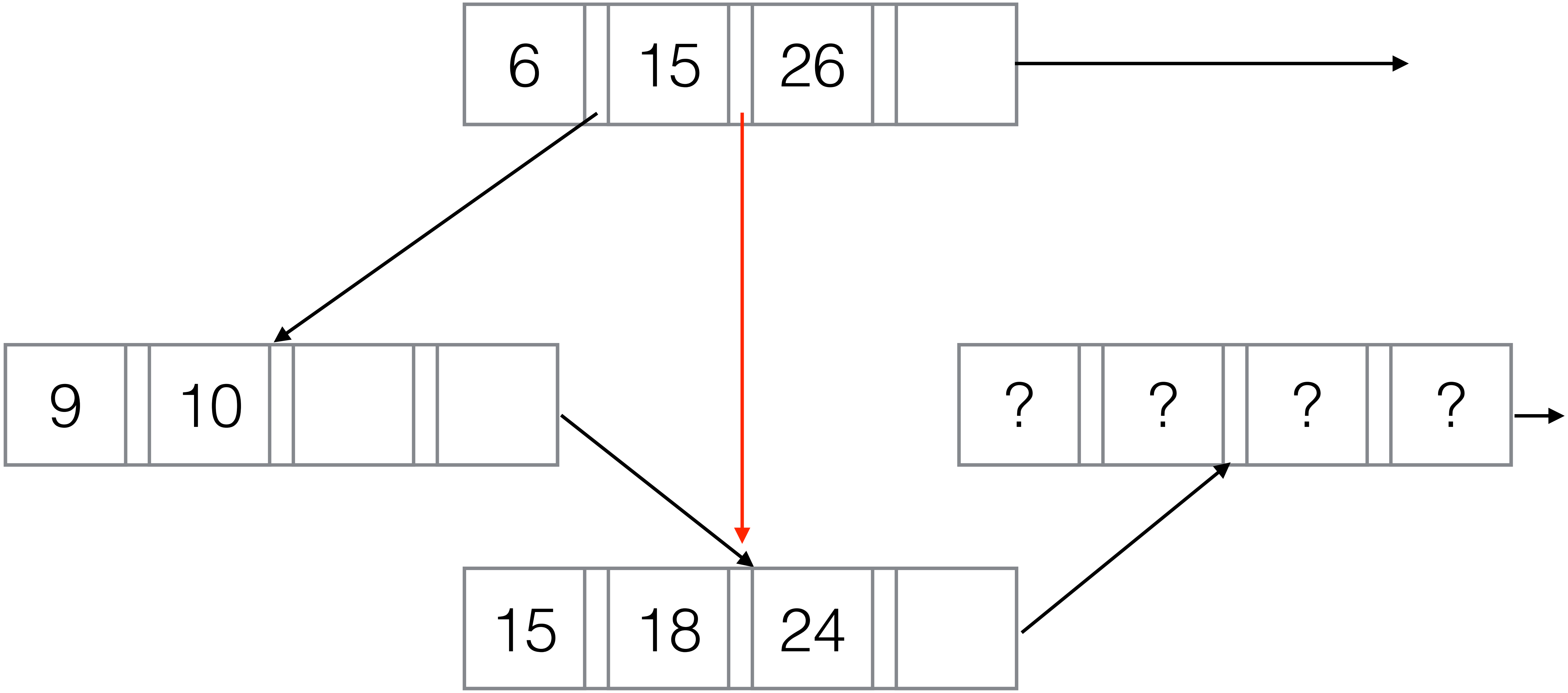
Writes - split

6. (Returned from recursive call)



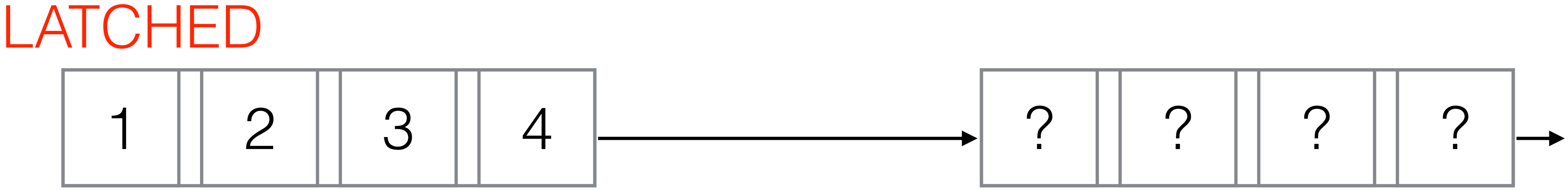
Writes - split

We’re done (finally).



Detail: `move_right()`

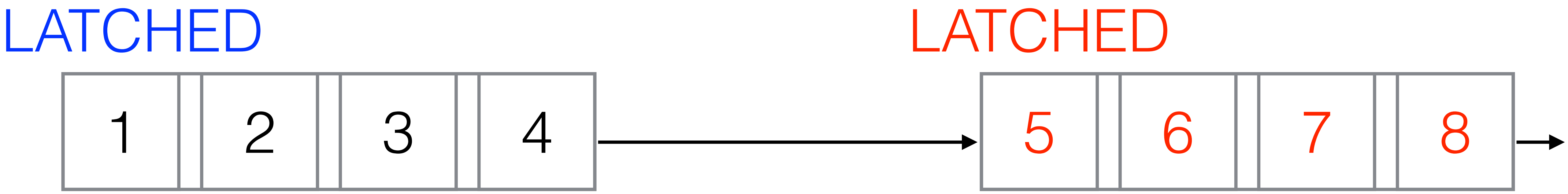
0. Acquire latch on node.



Moving right

Detail: `move_right()`

1. Latch right-pointer's node.



LATCHED

1 2 3 4

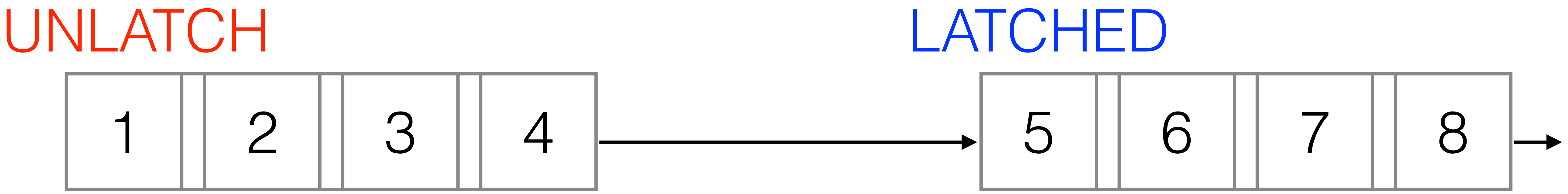
LATCHED

5 6 7 8

Moving right

Detail: `move_right()`

2. Unlatch old node.



UNLATCH

LATCHED

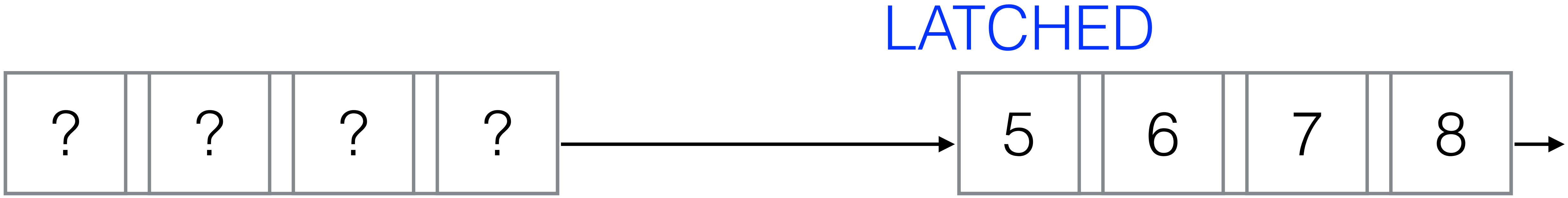
1 2 3 4

5 6 7 8

Moving right

Detail: `move_right()`

3. Done? If not, go back to step 1.



Moving right