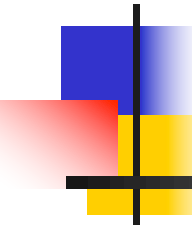
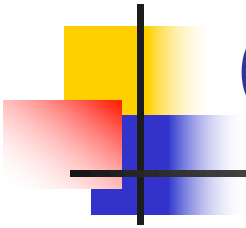


COMP231

B+-tree

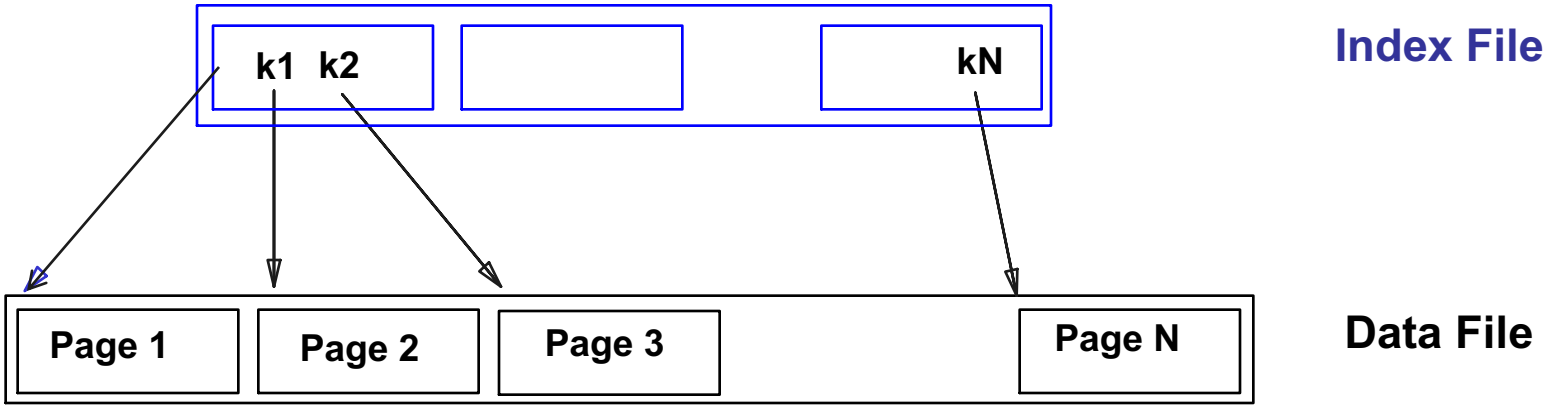


Outline

- B+-tree
- Query
- Update
  - Insertion
  - Deletion

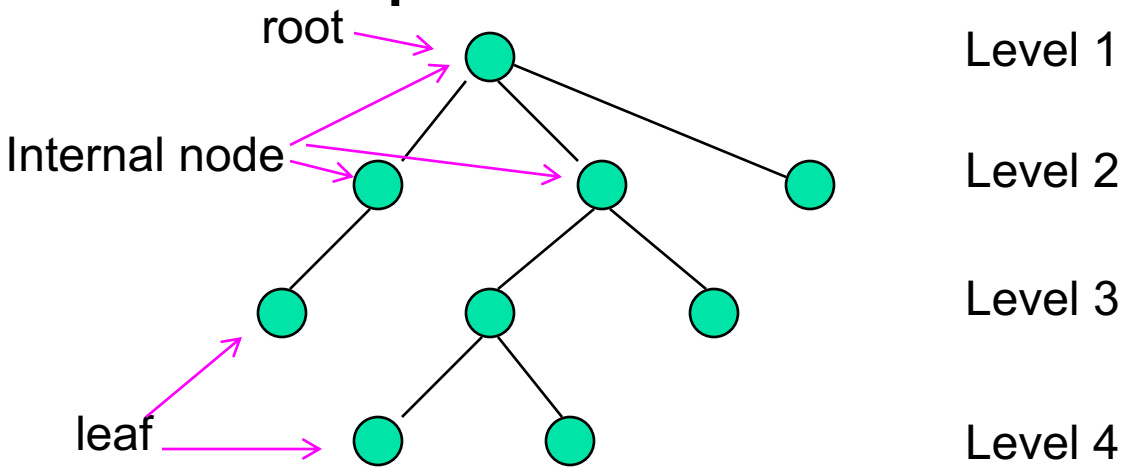
Range Searches

- *Find all students with gpa > 3.0*
  - If records are sorted on gpa, do binary search to find first such student, then scan to find others.
  - Cost of binary search can be quite high.
- Simple idea: Create an "index" file.



B+ Tree: Most Widely Used Index

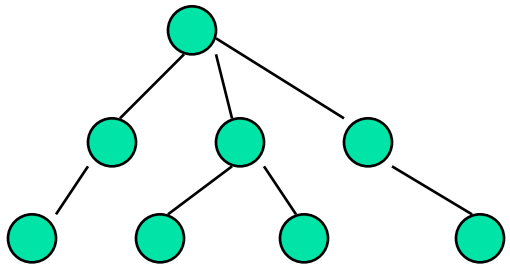
- General concept of a tree:



I. Height of a node: its distance to the root
II. If a higher level node is connected to a lower level node, then the higher level node is called a **parent** (**grandparent**, **ancestor**, etc.) of the lower level node

B+ Tree (cont.)

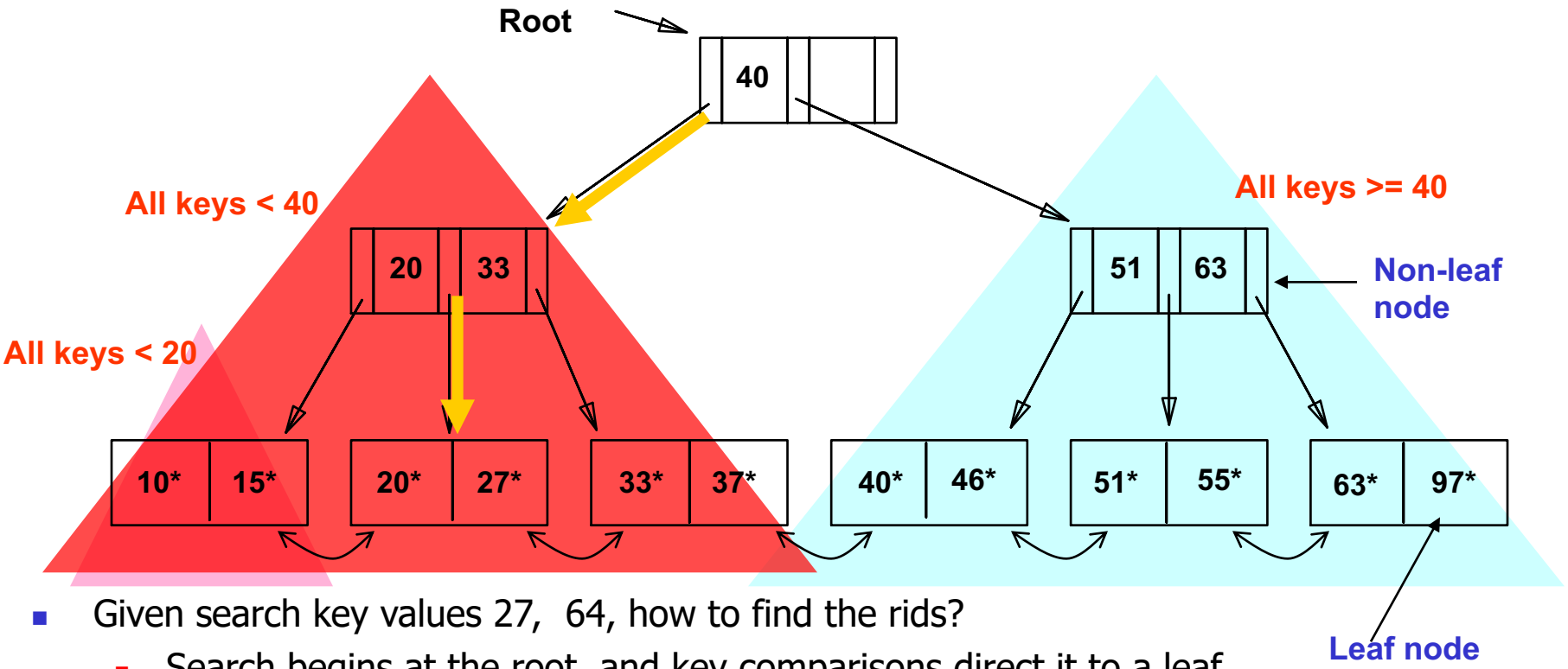
- **Balanced tree**: all leaves at the same level
  - Example:



- Structure of a B+ tree:
  - It is ***balanced:*** all leaf nodes are at the same level
  - Each node has a special structure

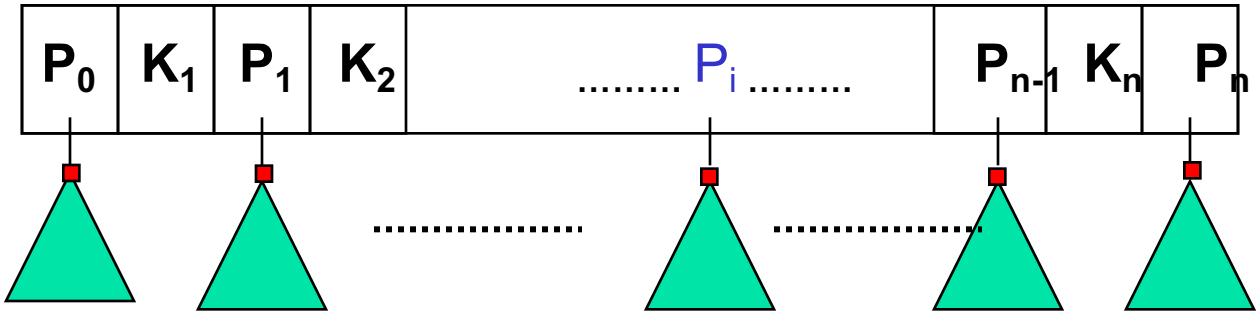
Example: A B+ tree

- In this example, each node holds at least 1 entry, and at most 2 entries

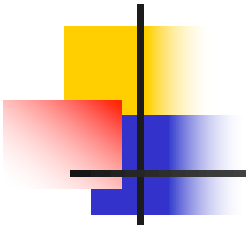


- Given search key values 27, 64, how to find the rids?
  - Search begins at the root, and key comparisons direct it to a leaf
  - Note that key values are sorted at each level

B+ tree: Internal node structure

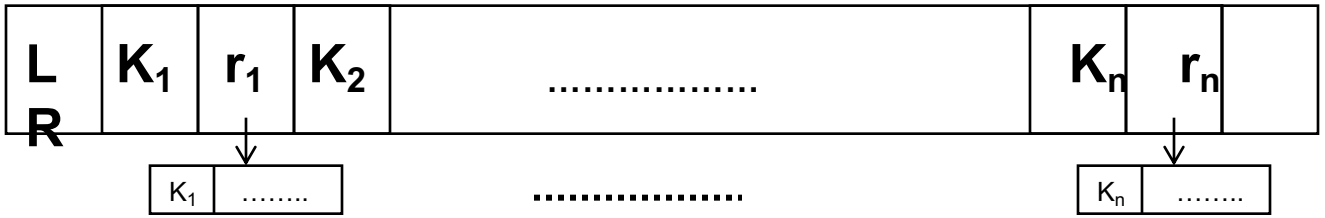


Each P_i is a pointer to a child node, each K_i is a search key value
Pointers outnumber search key values by *exactly one.*

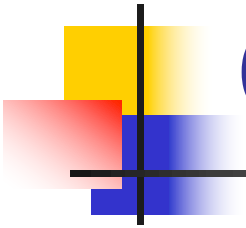


- Requirements:
  - K1 < K2 < … < Kn
  - If the node is not the root, we require $d \leq n \leq 2d$ where d is a pre-determined value for this B+ tree, called its ***order***
  - If the node is the root, we require $1 \leq n \leq 2d$
  - For any search key value K in the subtree pointed by Pi,
    - If Pi = P0, we require that K < K1
    - If Pi = P1, …, Pn-1, we require that $Ki \leq K < Ki+1$
    - If Pi = Pn, we require $Kn \leq K$

B+ tree: leaf node structure



- Each r_i is a pointer to a record that contains search key value K_i
- L points to the left neighbor, and R points to the right neighbor
- $K_1 < K_2 < \ldots < K_n$
- We require $d \leq n \leq 2d$ where d is the order of this B+ tree

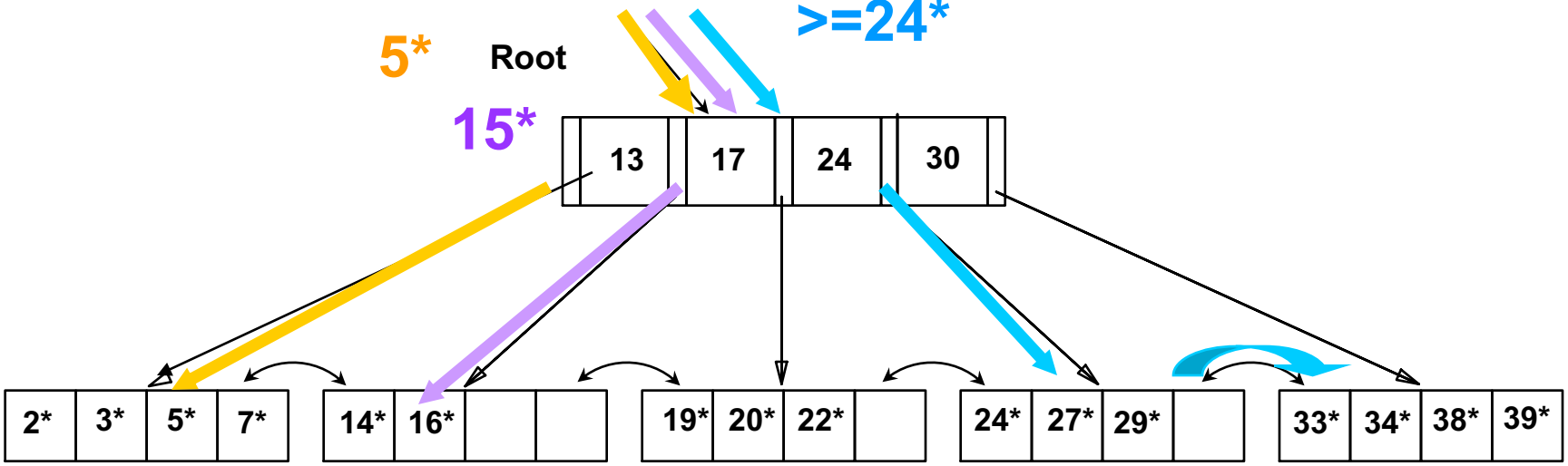


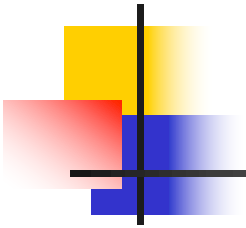
Outline

- B+-tree
- Query
- Update
  - Insertion
  - Deletion

Example: a B+ tree with order 2

- Search for 5*, 15*, all data entries >= 24* ...
- The last one is a range search, we need to do the sequential scan, starting from the first leaf containing a value >= 24.





Height/Depth of B+-tree

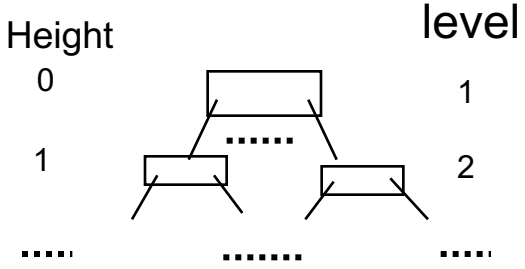
- What is the minimum height of the B+-tree?

- What is the maximum height of the B+-tree?

Minimum height of the B+-tree

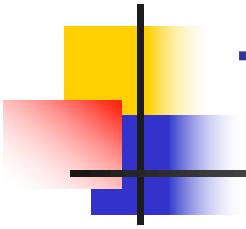
- In general nodes are pages
- Let H be the height of the B+ tree: need to read H+1 pages to reach a leaf node
- Let F be the number of pointers in a node (for internal node, called *fanout*)
  - Level 1 = 1 page = F^0 page
  - Level 2 = F pages = F^1 pages
  - Level 3 = Fx F pages = F^2 pages
  - Level H+1 = …….. = F^H pages (i.e., leaf nodes)
  - Suppose there are D data entries. So there are at least D/(F-1) leaf nodes
  - D/(F-1) <= F^H. That is, H >= $\log_F(\frac{D}{F-1})$

F = 2d+1

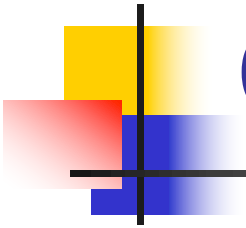


Note that data entry is one less than pointers

Maximum height of the B+-tree

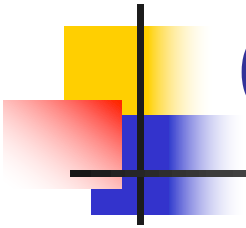


- What is the maximum height of the B+-tree?



Cost of Searching a key

- Usually, each node occupies a page
- Cost of searching a key = H + 1 (pages)



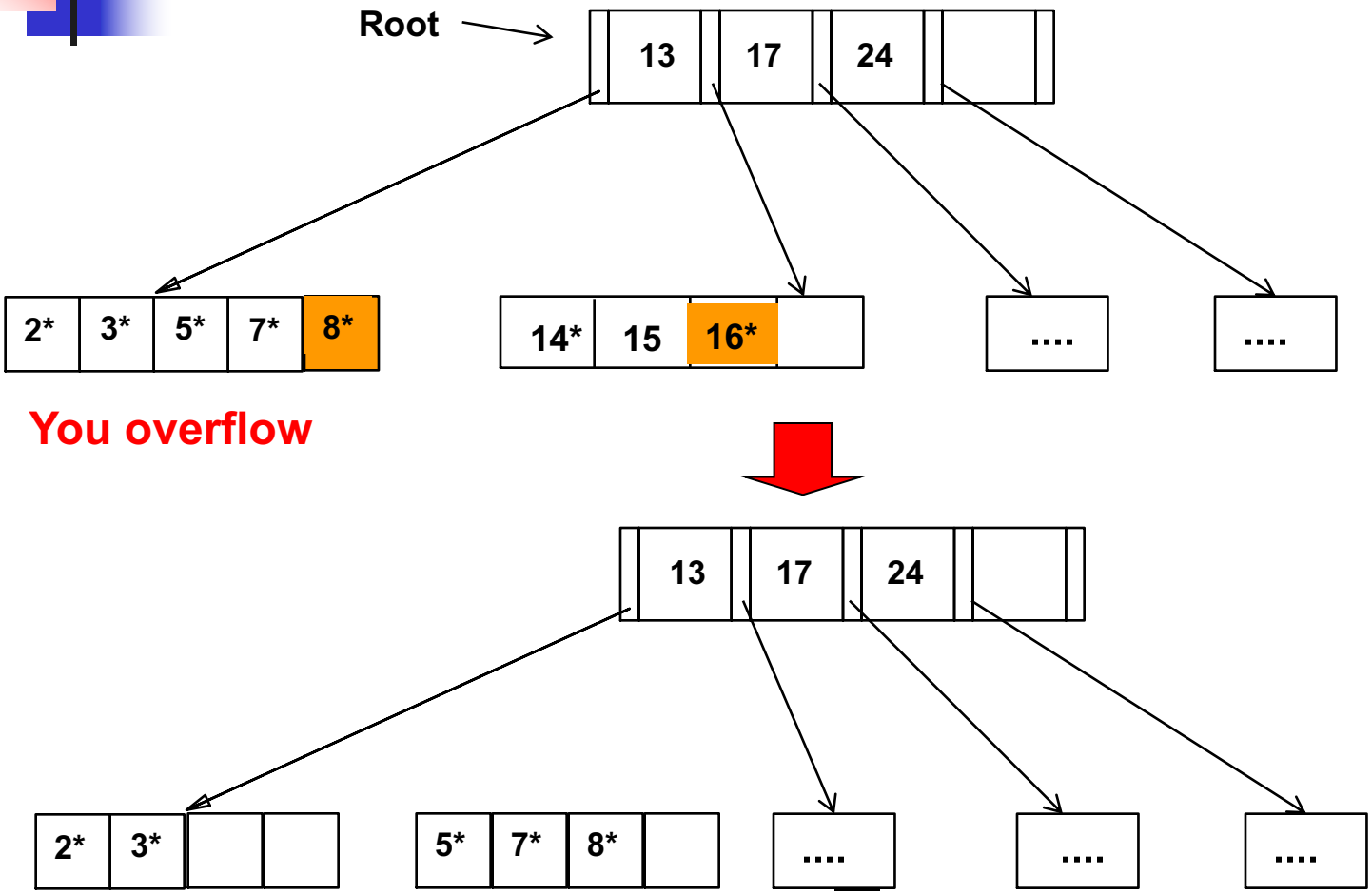
Outline

- B+-tree
- Query
- Update
  - Insertion
  - Deletion

Inserting a Data Entry into a B+ Tree

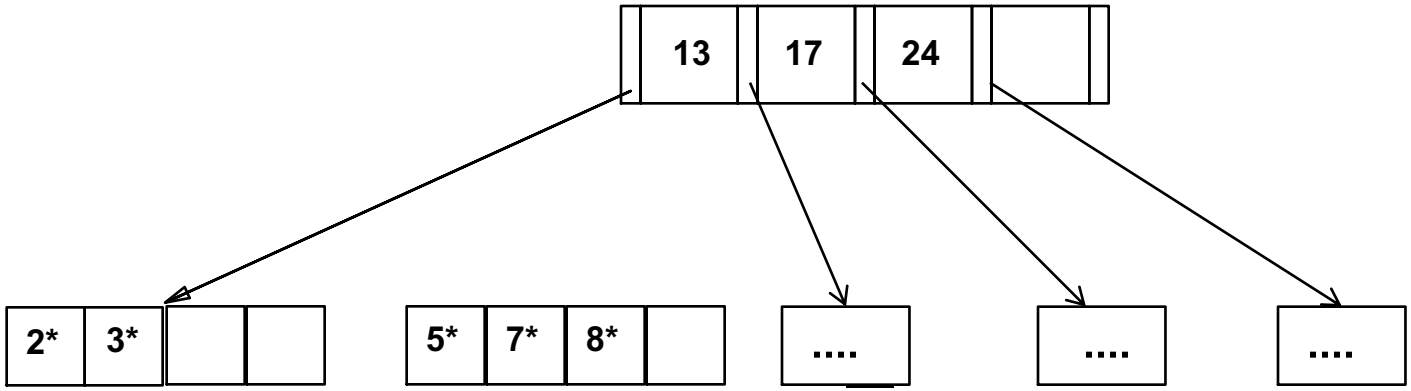
- Find correct leaf *L.*
- Put data entry onto *L*.
  - If *L* has enough space, *done*!
  - Else, must *split* *L (into L and a new node L2)*
    - Redistribute entries evenly, put middle key in L2
    - **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To split index node, redistribute entries evenly,
  - but **push up** middle key. (Contrast with leaf splits.)
- Splits "grow" tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top.*

Example 1: Inserting 16*, 8* into Example B+ tree

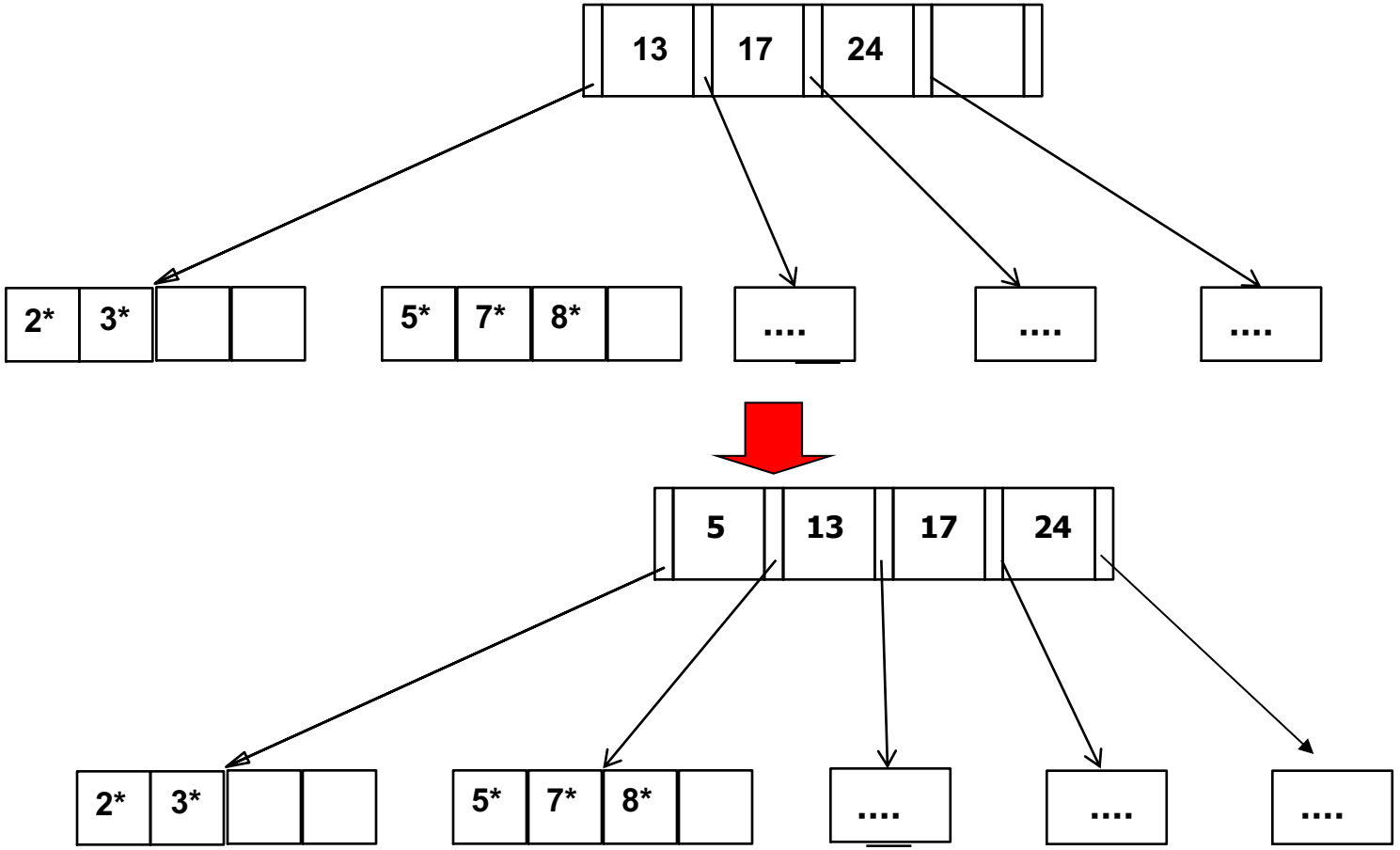


One more child generated, must add one more pointer to its parent, thus one more key value as well.

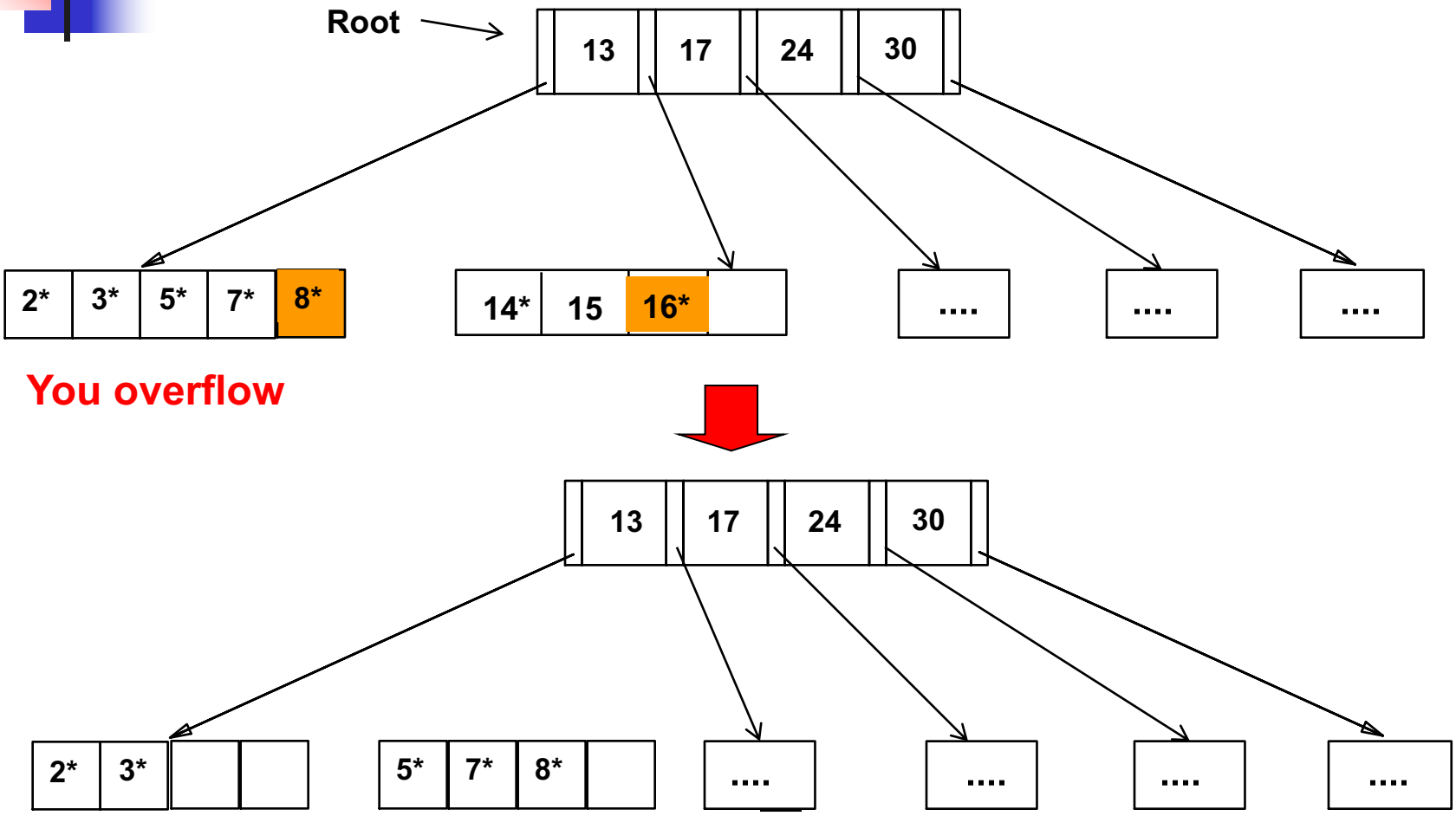
Example 1: Inserting 16*, 8* into Example B+ tree



Example 1: Inserting 16*, 8* into Example B+ tree

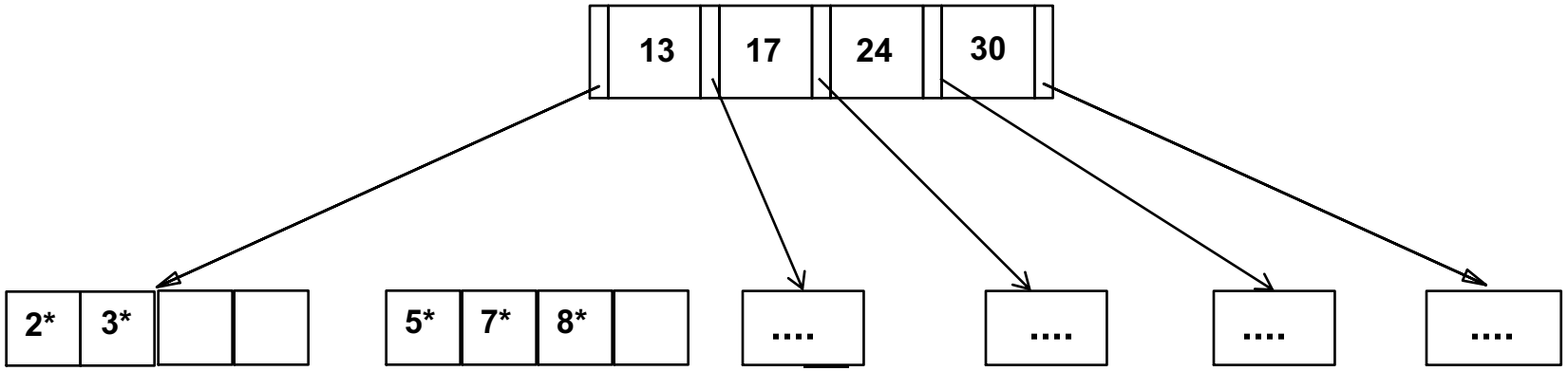


Example 2: Inserting 16*, 8* into Example B+ tree

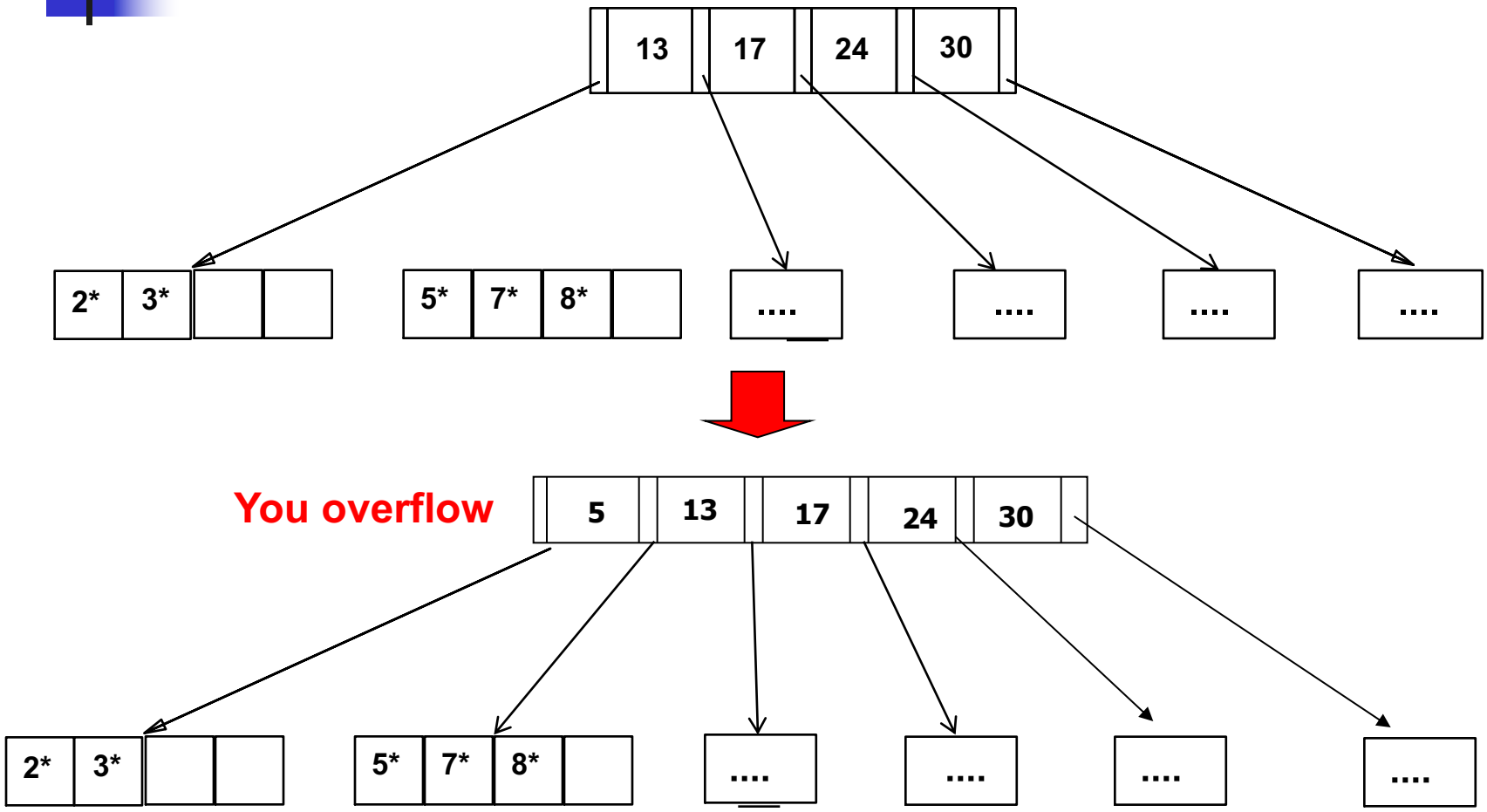


One more child generated, must add one more pointer to its parent, thus one more key value as well.

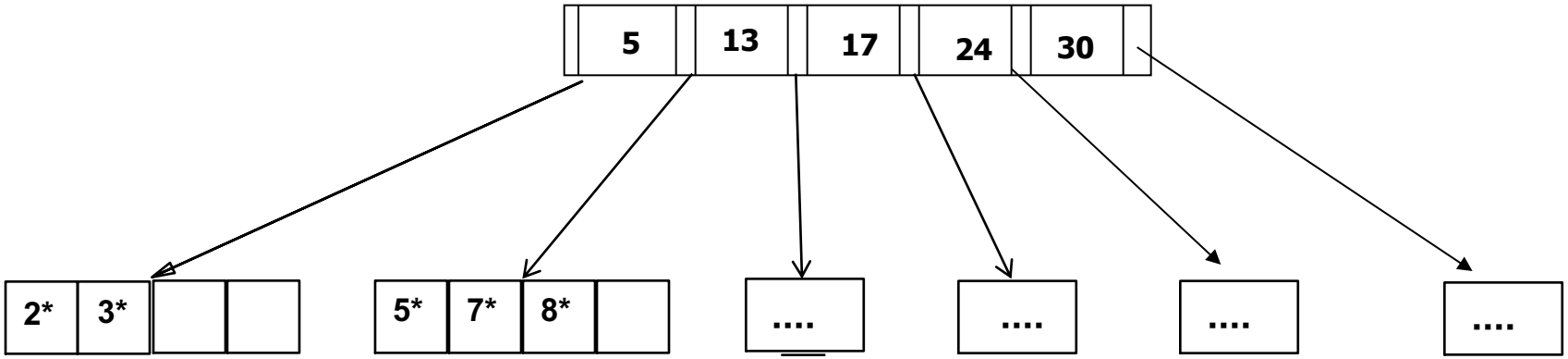
Example 2: Inserting 16*, 8* into Example B+ tree



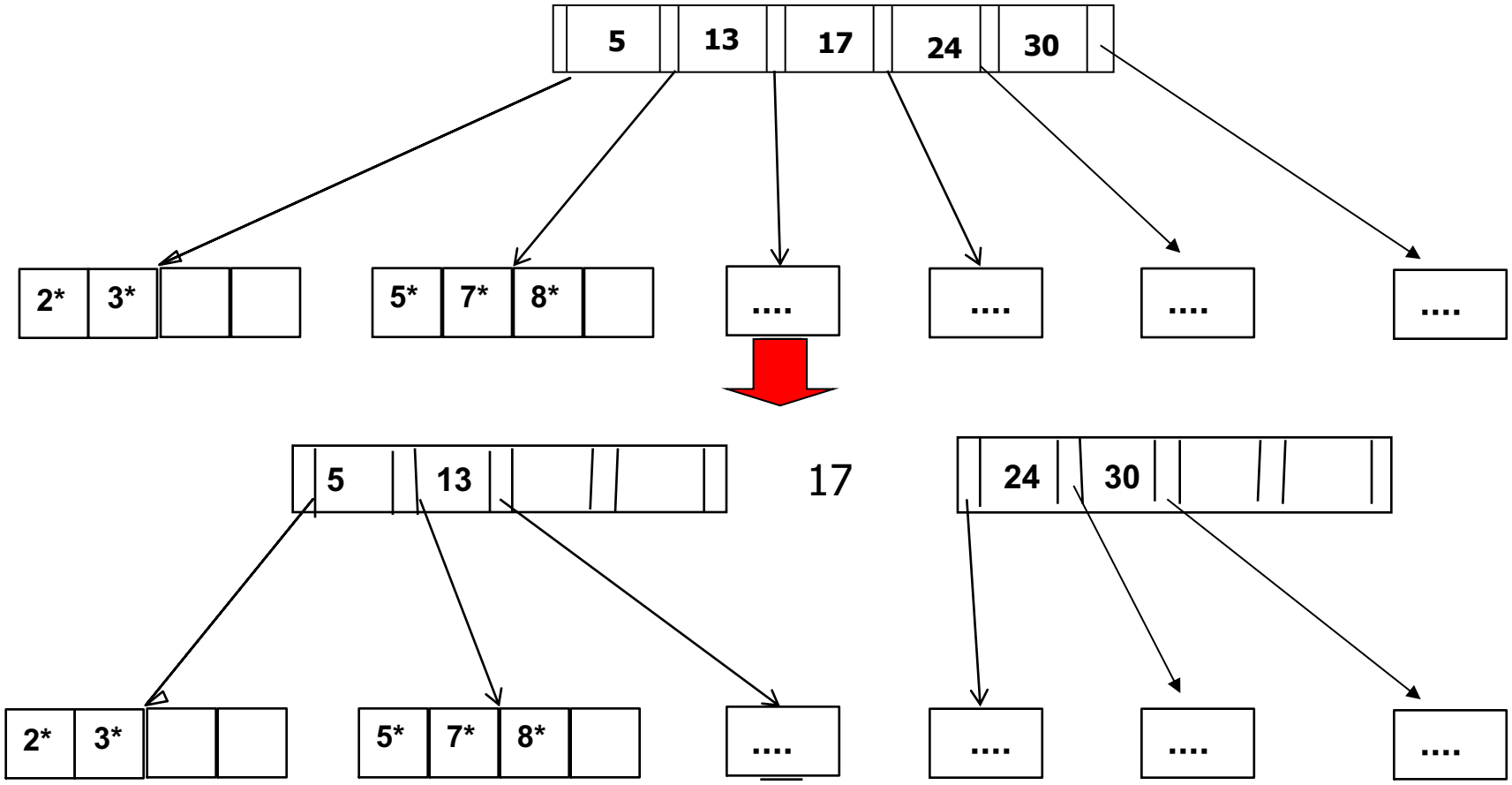
Example 2: Inserting 16*, 8* into Example B+ tree



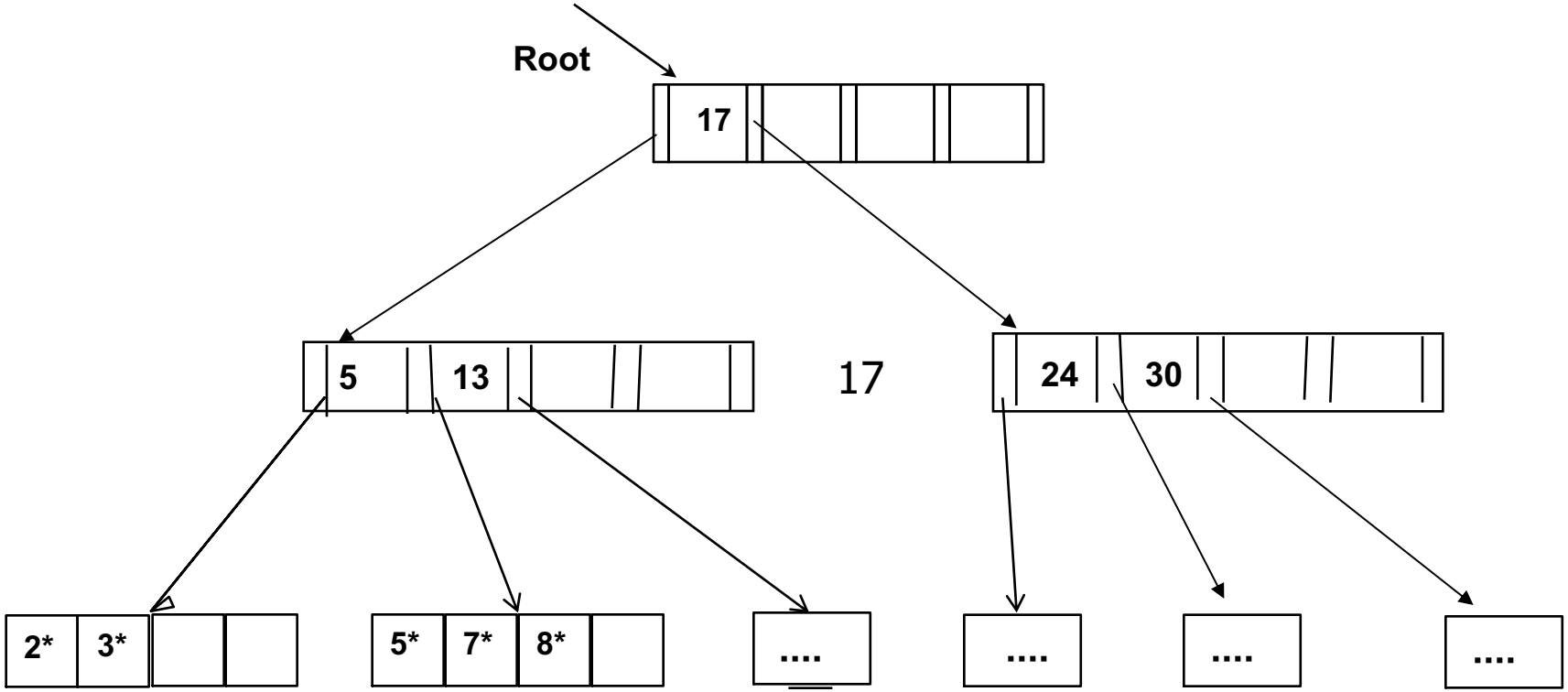
Example 2: Inserting 16*, 8* into Example B+ tree



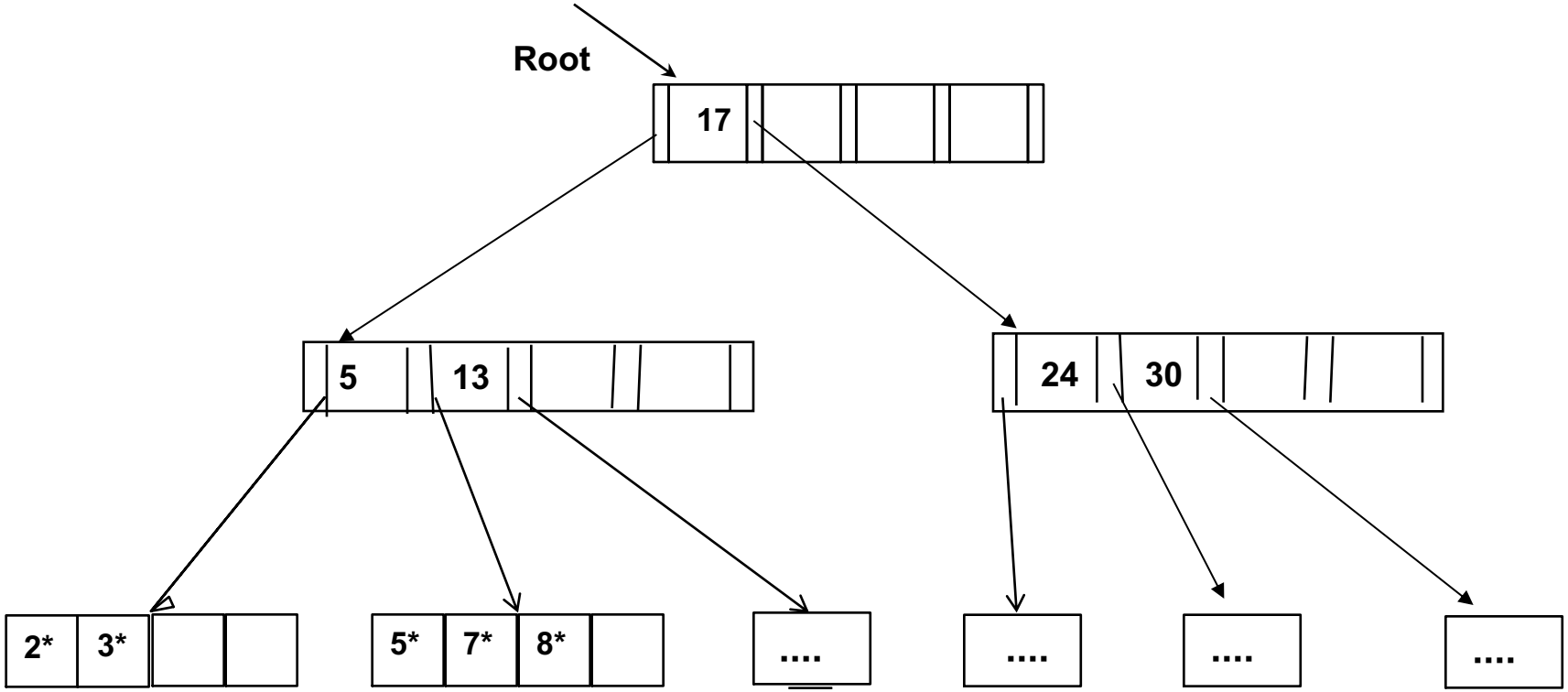
Example 2: Inserting 16*, 8* into Example B+ tree

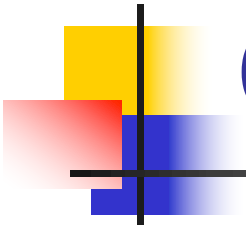


Example 2: Inserting 16*, 8* into Example B+ tree



Example 2: Inserting 16*, 8* into Example B+ tree

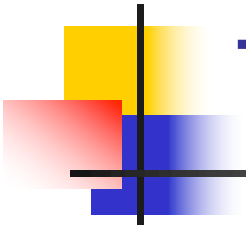




Outline

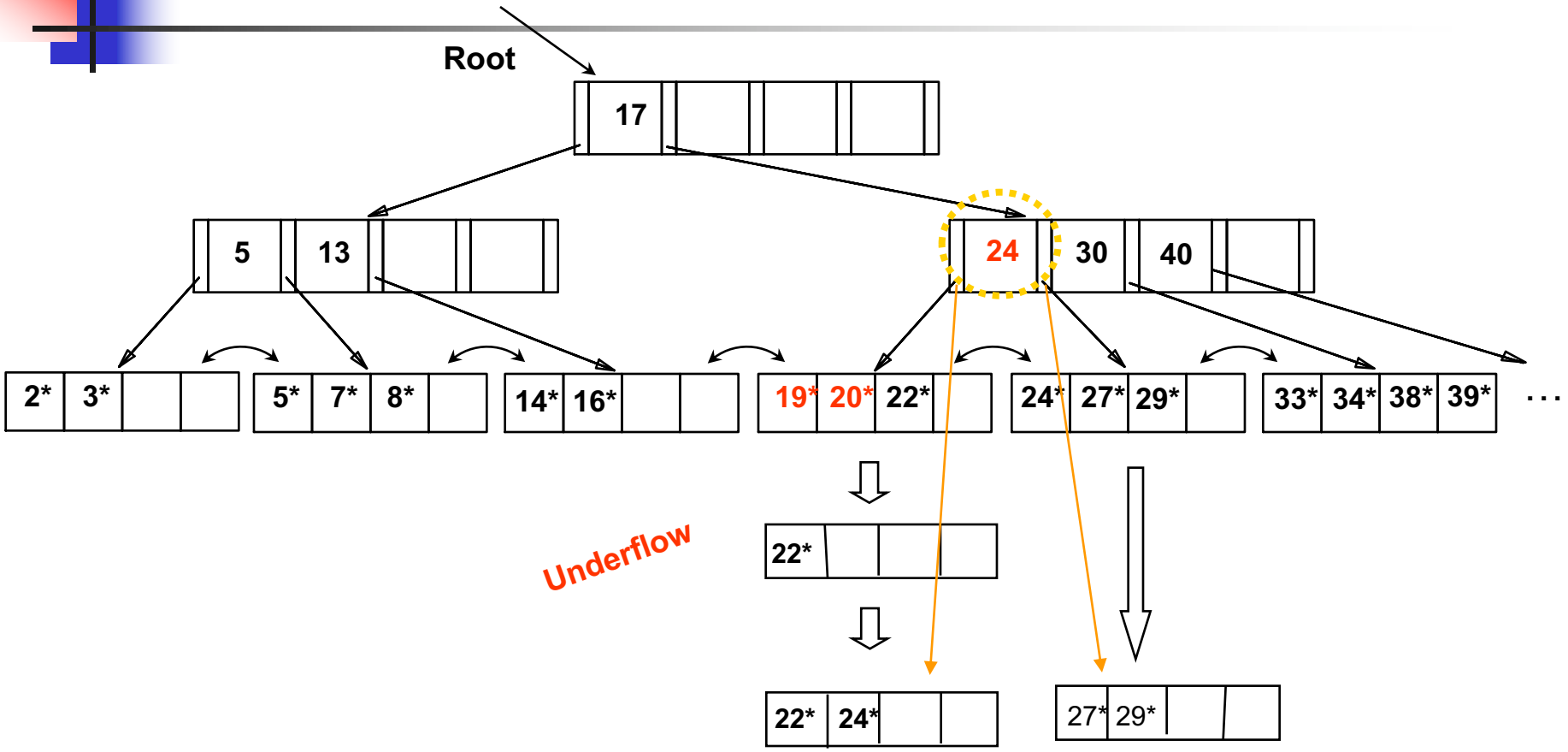
- B+-tree
- Query
- Update
  - Insertion
  - Deletion

Deleting a Data Entry from a B+ Tree



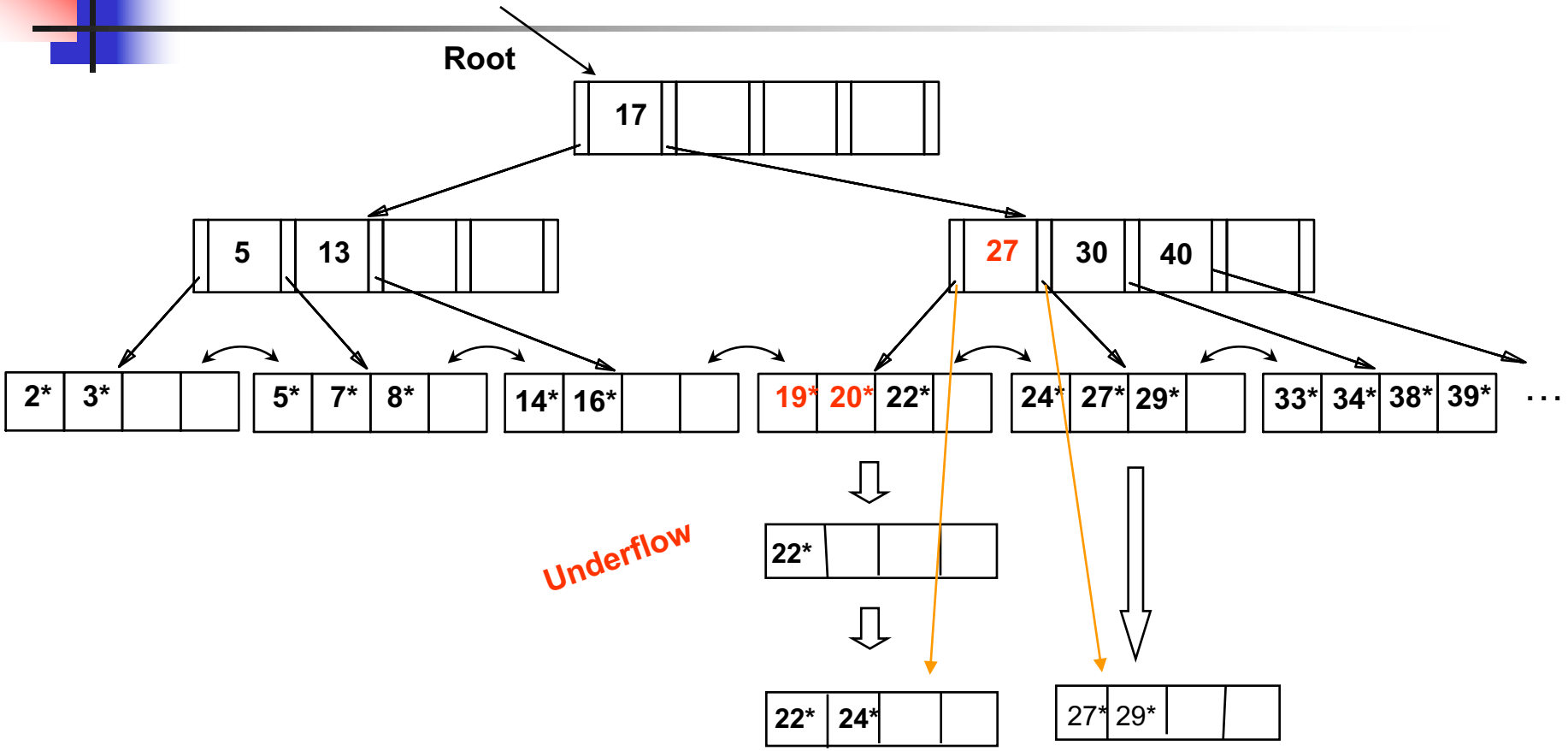
- Start at root, find leaf *L* where entry belongs.
- Remove the entry.
  - If L is at least half-full, *done!*
  - If L has only **d-1** entries,
    - Try to re-distribute, borrowing from *sibling (adjacent node with same parent as L)*.
    - If re-distribution fails, *merge* *L* and sibling.
- If merge occurred, must delete entry (pointing to *L* or sibling) from parent of *L*.
- Merge could propagate to root, decreasing height.

Ex1: Delete 19* and 20*

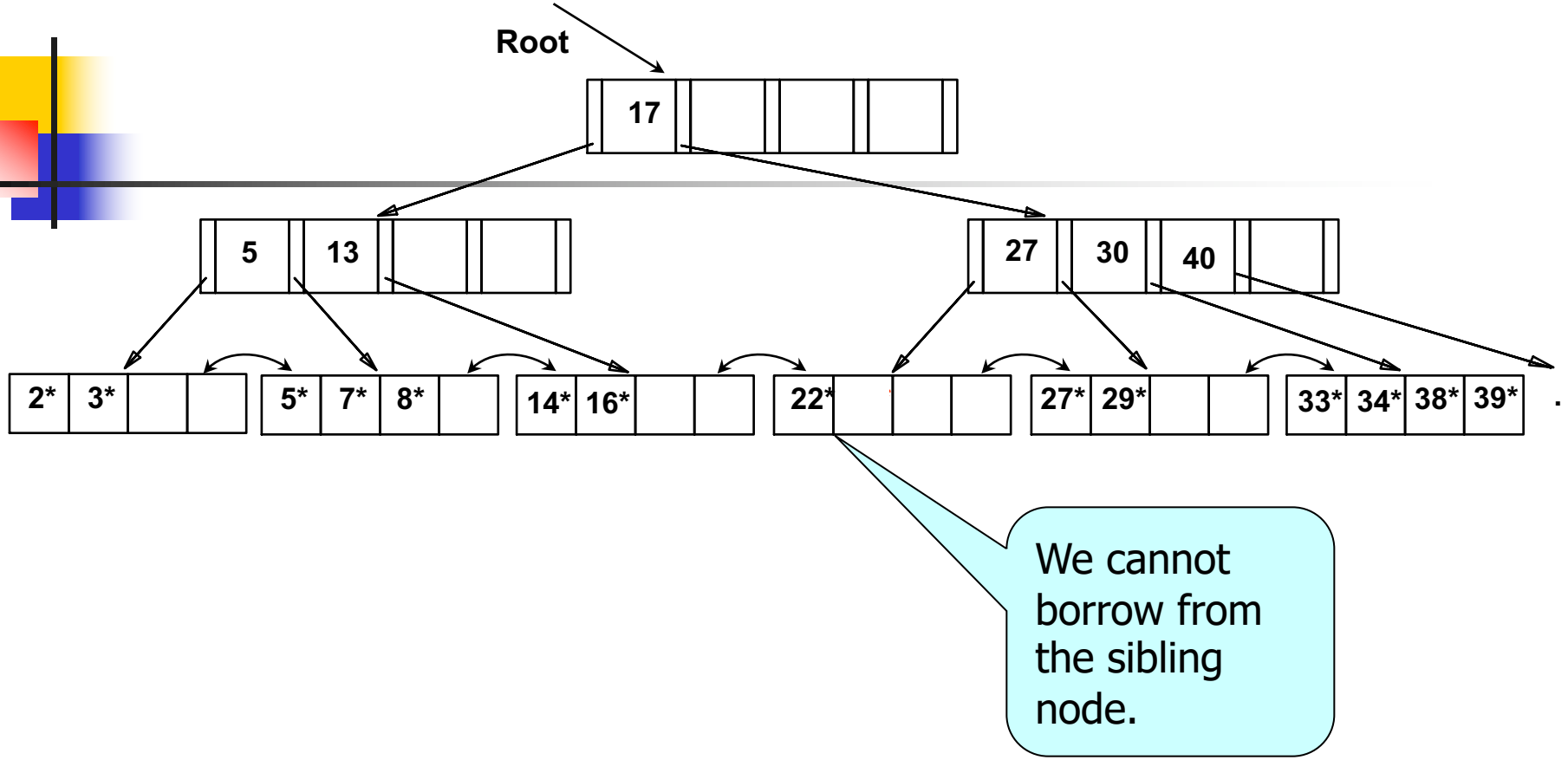


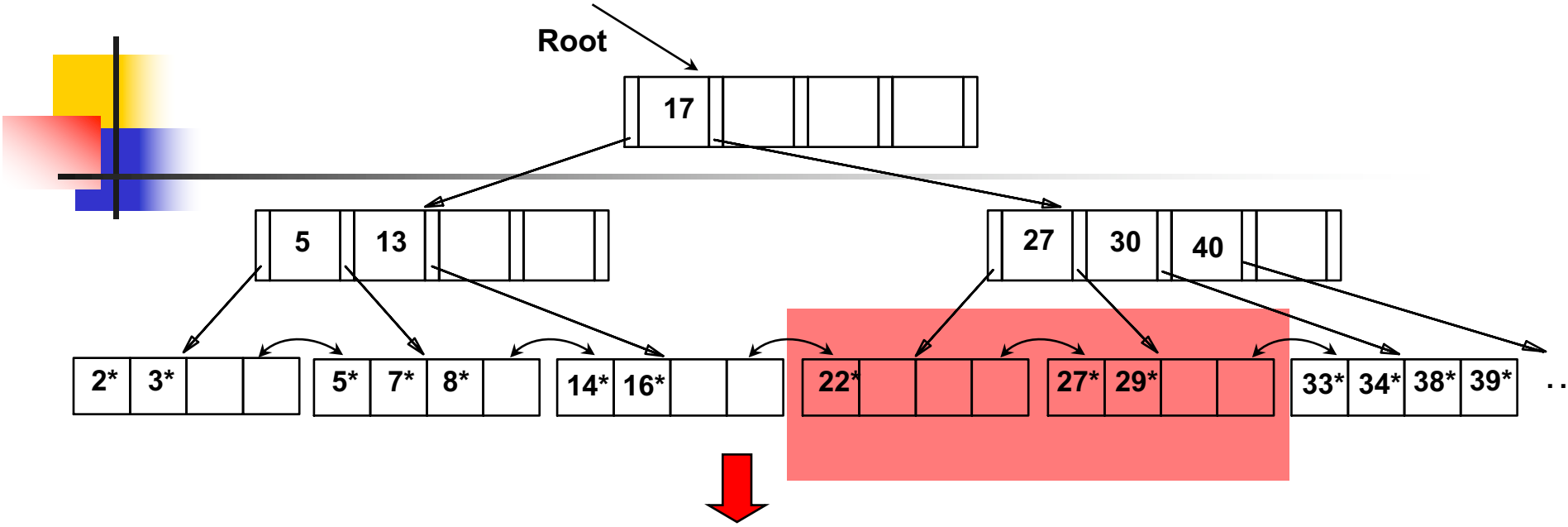
Have we still forgot something?

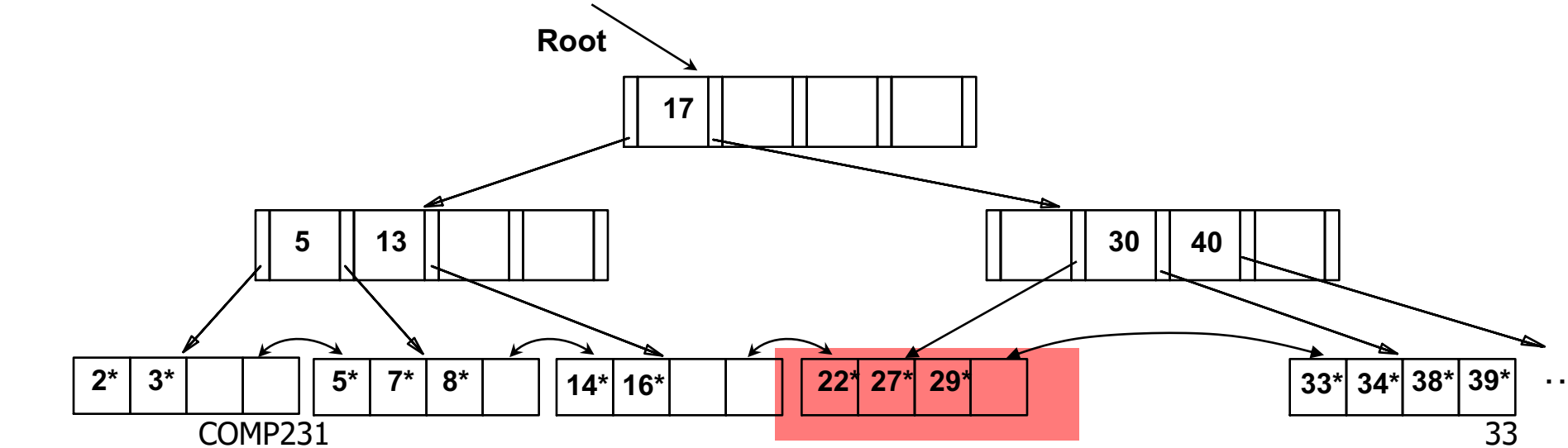
Ex1: Delete 19* and 20*

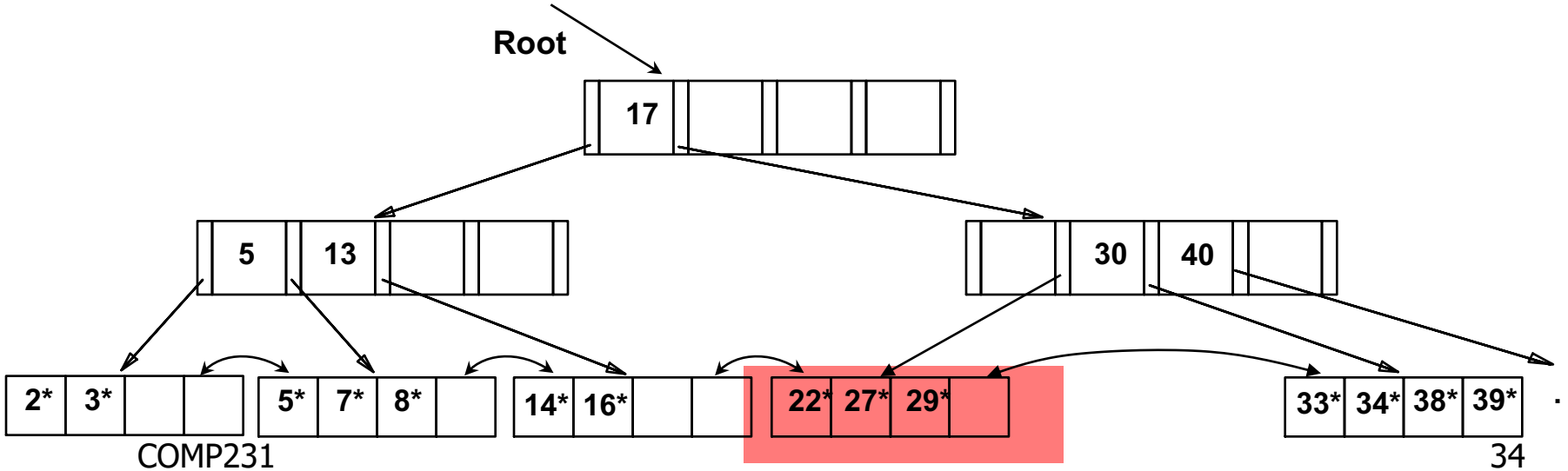


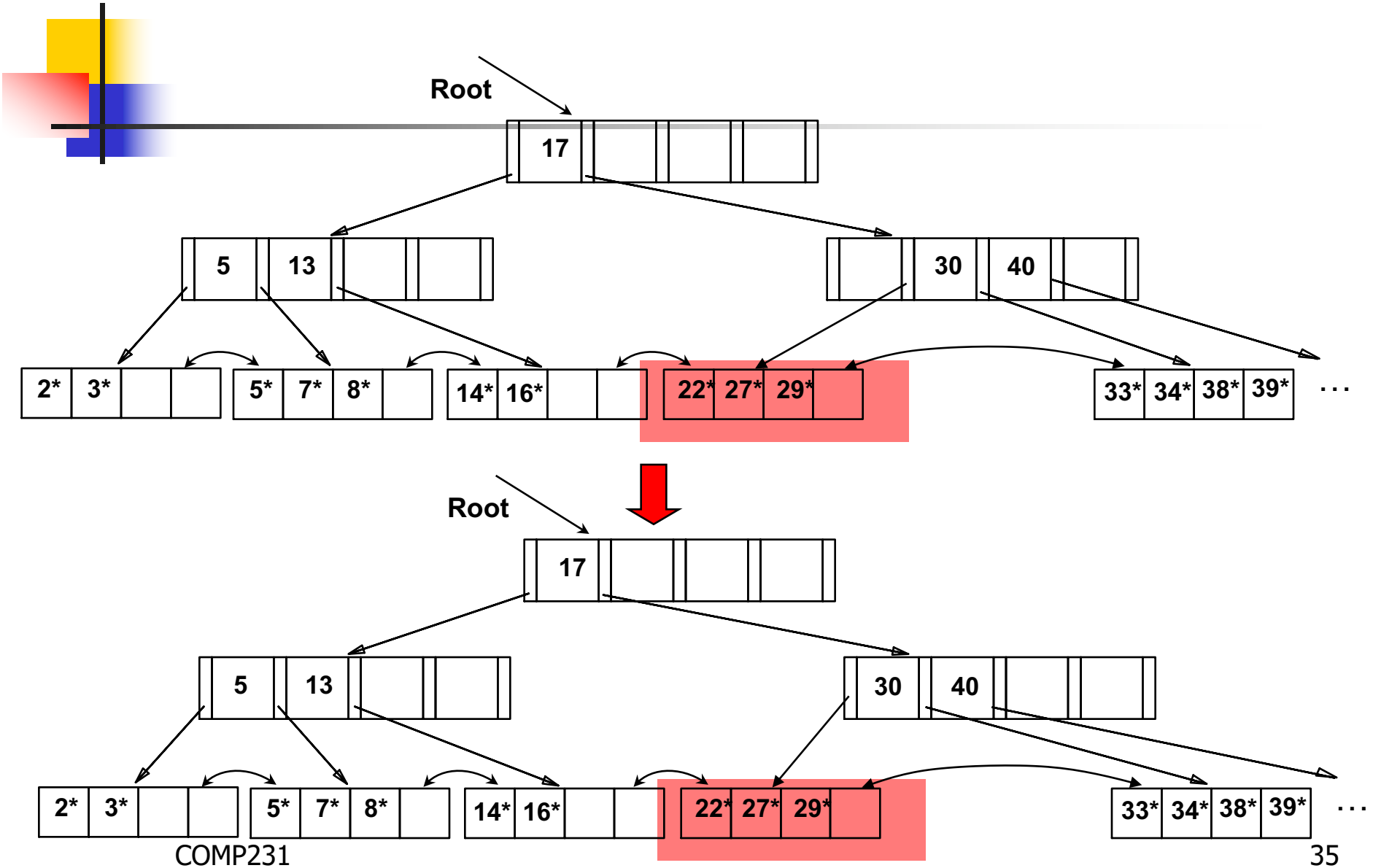
Ex2: Deleting 24*



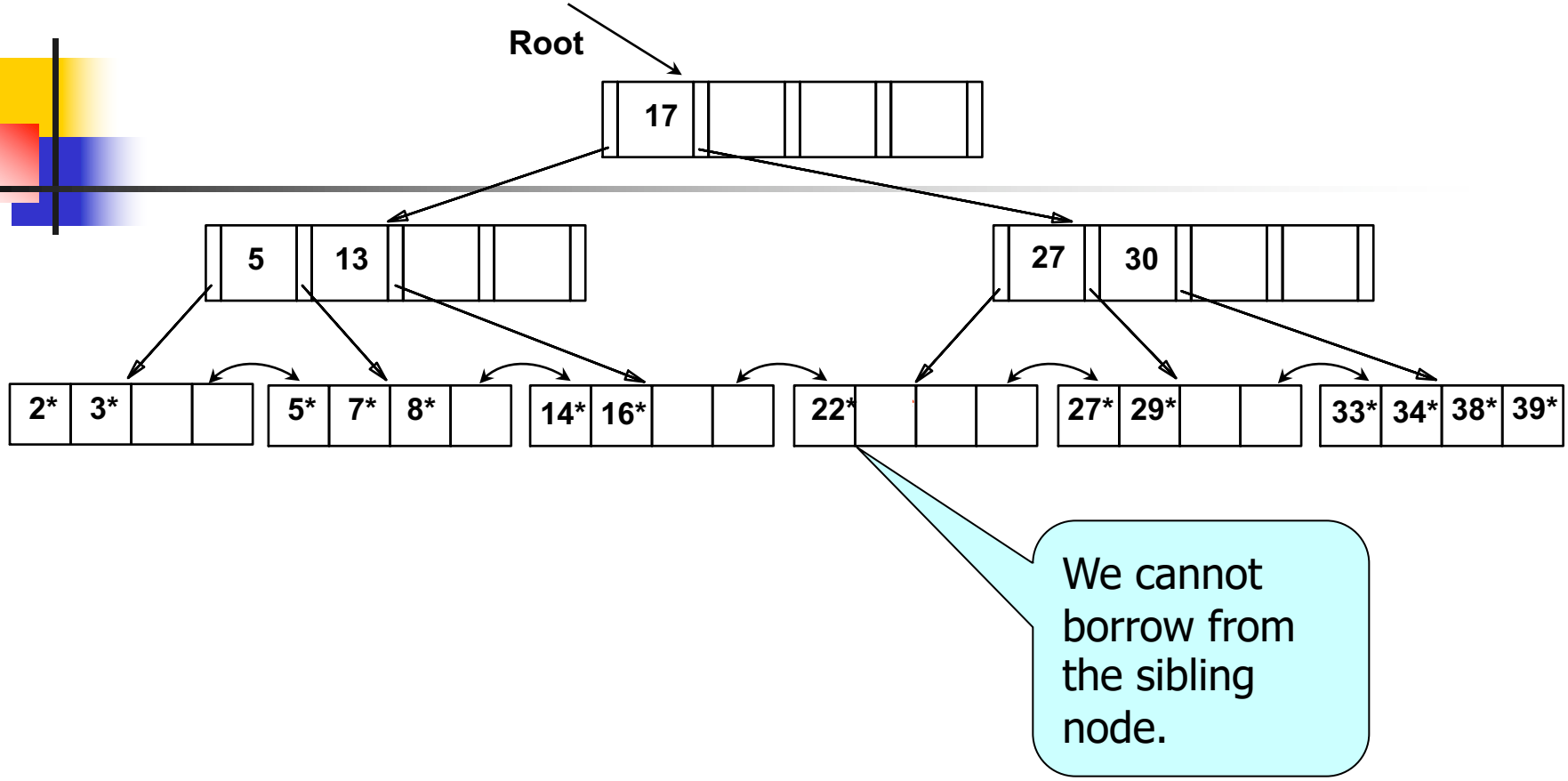

Root
17
5
13
27
30
40
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*

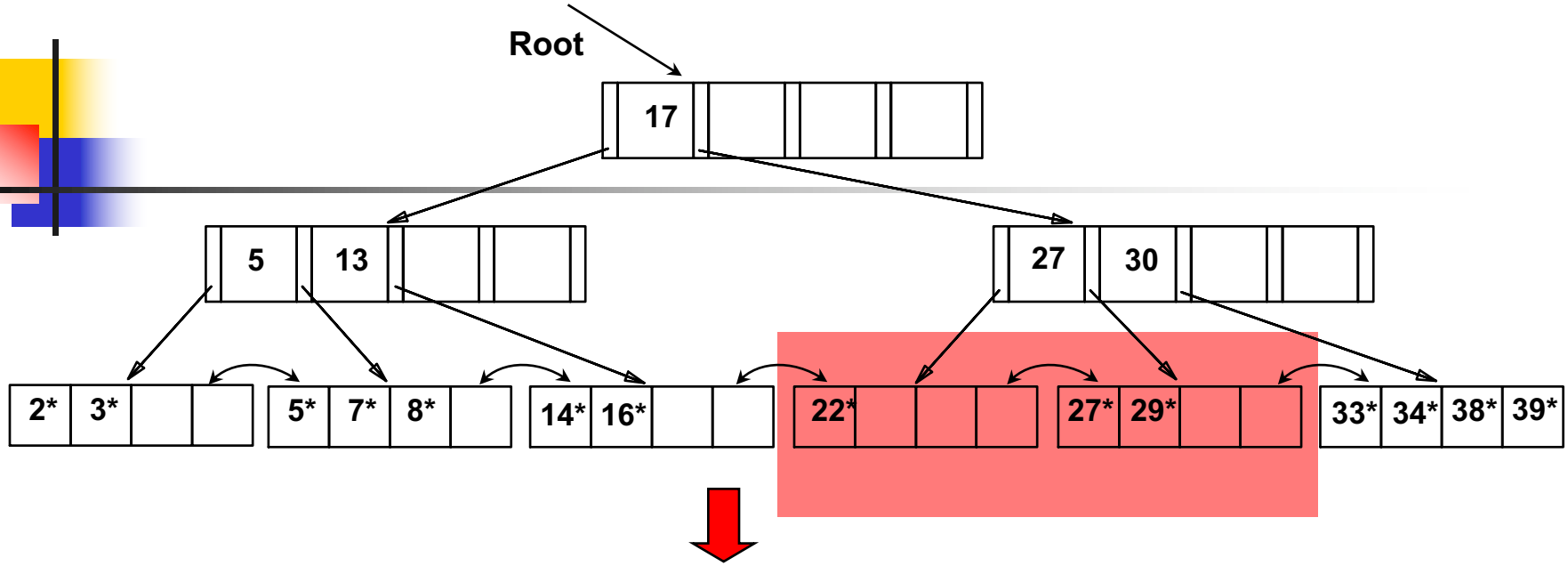

Root
17
5
13
30
40
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*

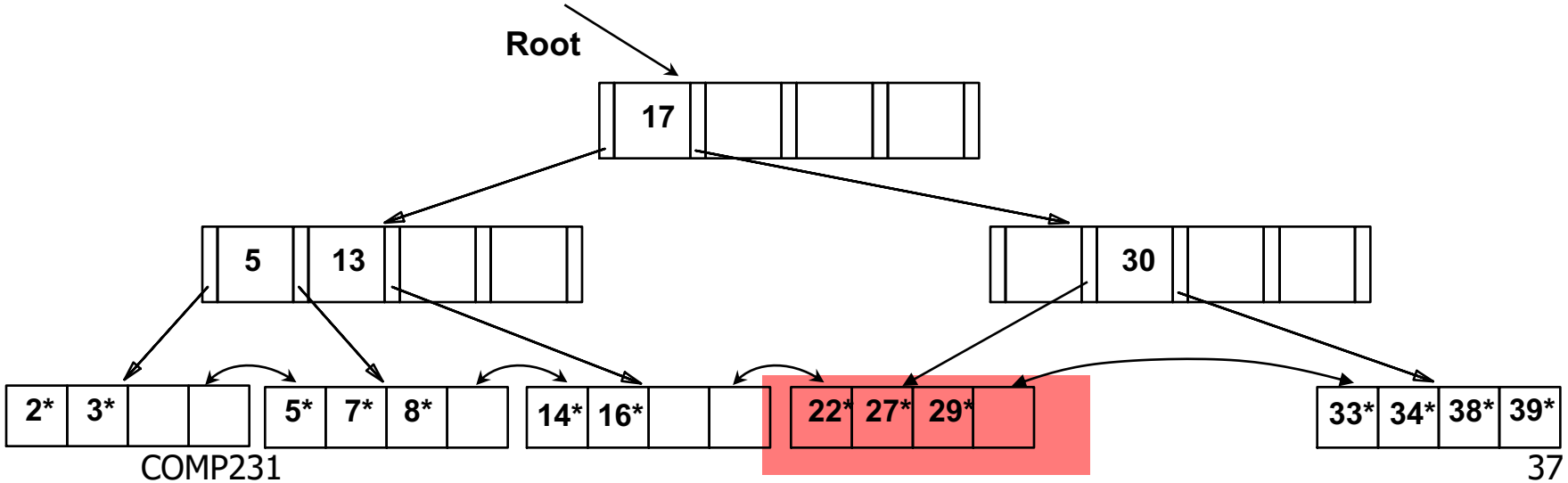

Root
17
5
13
30
40
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*

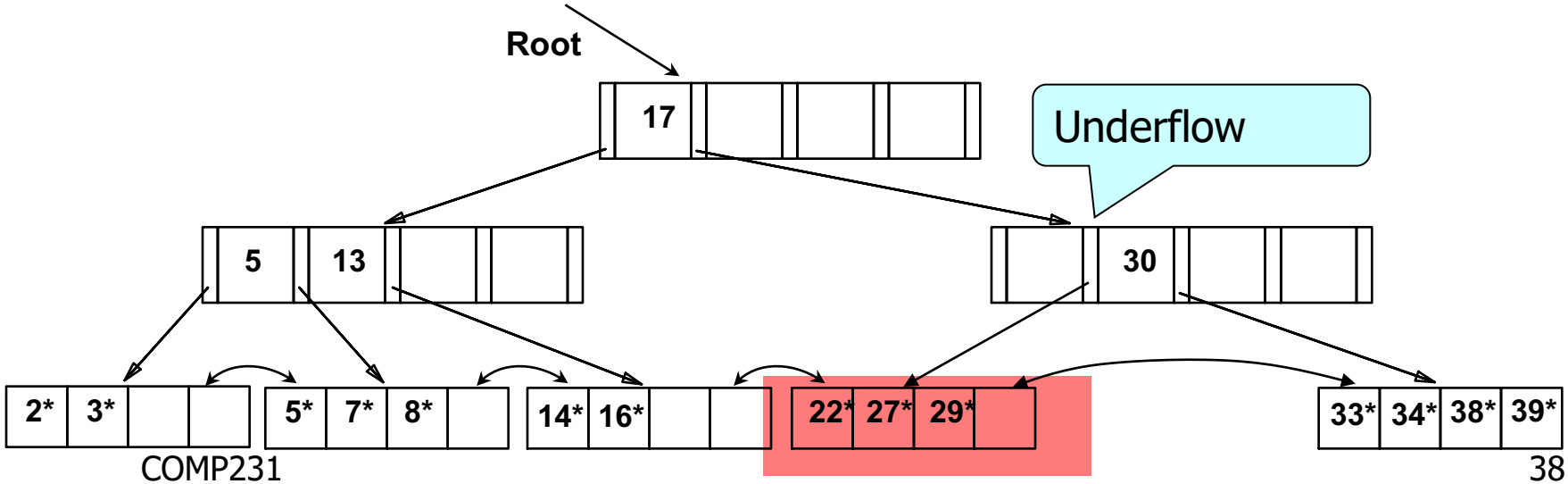

Root
17
5
13
30
40
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*
Root
17
5
13
30
40
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*

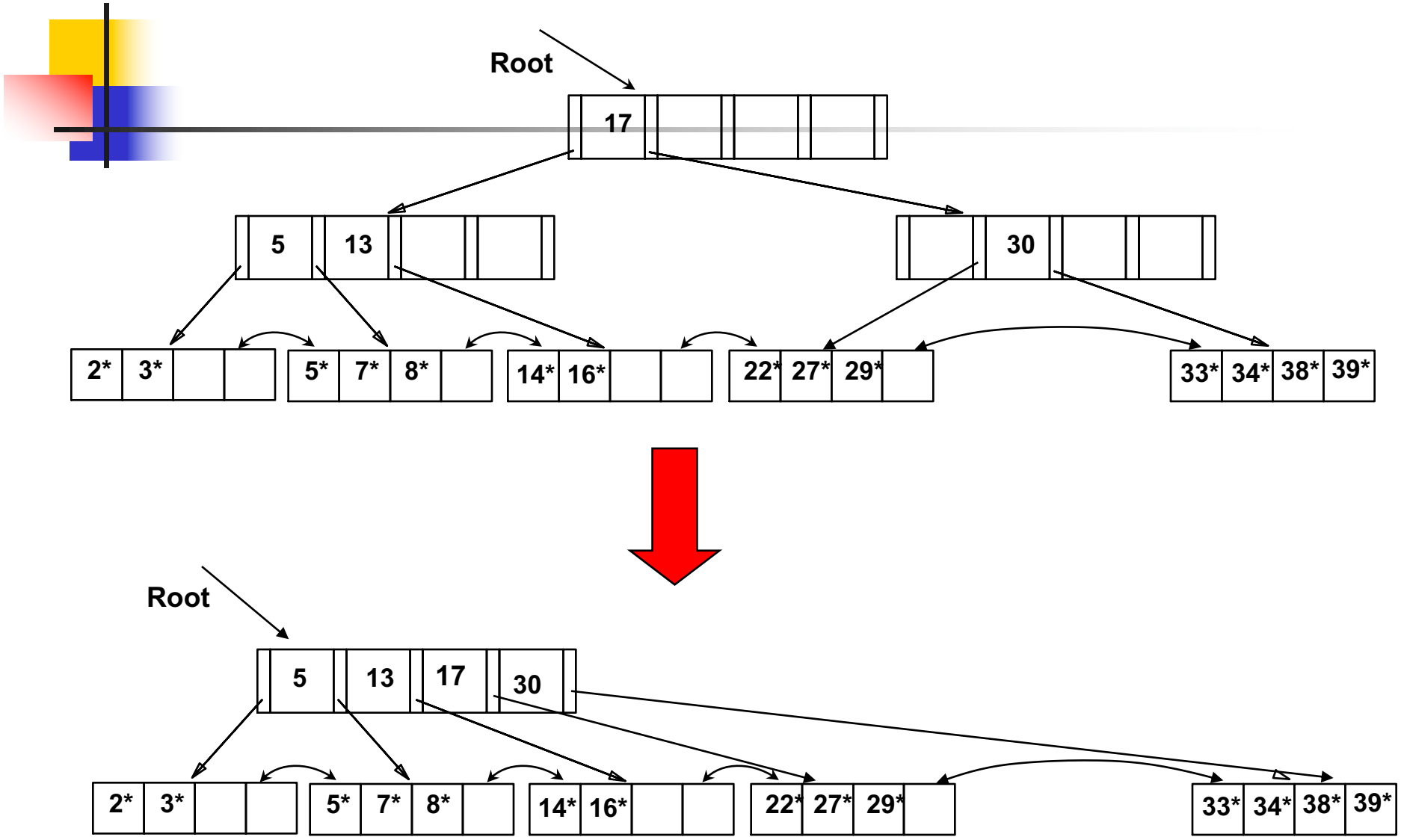
Ex3: Deleting 24*




Root
17
5
13
27
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*


Root
17
5
13
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*


Root
17
Underflow
5
13
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*


Root
17
5
13
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*
Root
5
13
17
30
2*
3*
5*
7*
8*
14*
16*
22*
27*
29*
33*
34*
38*
39*