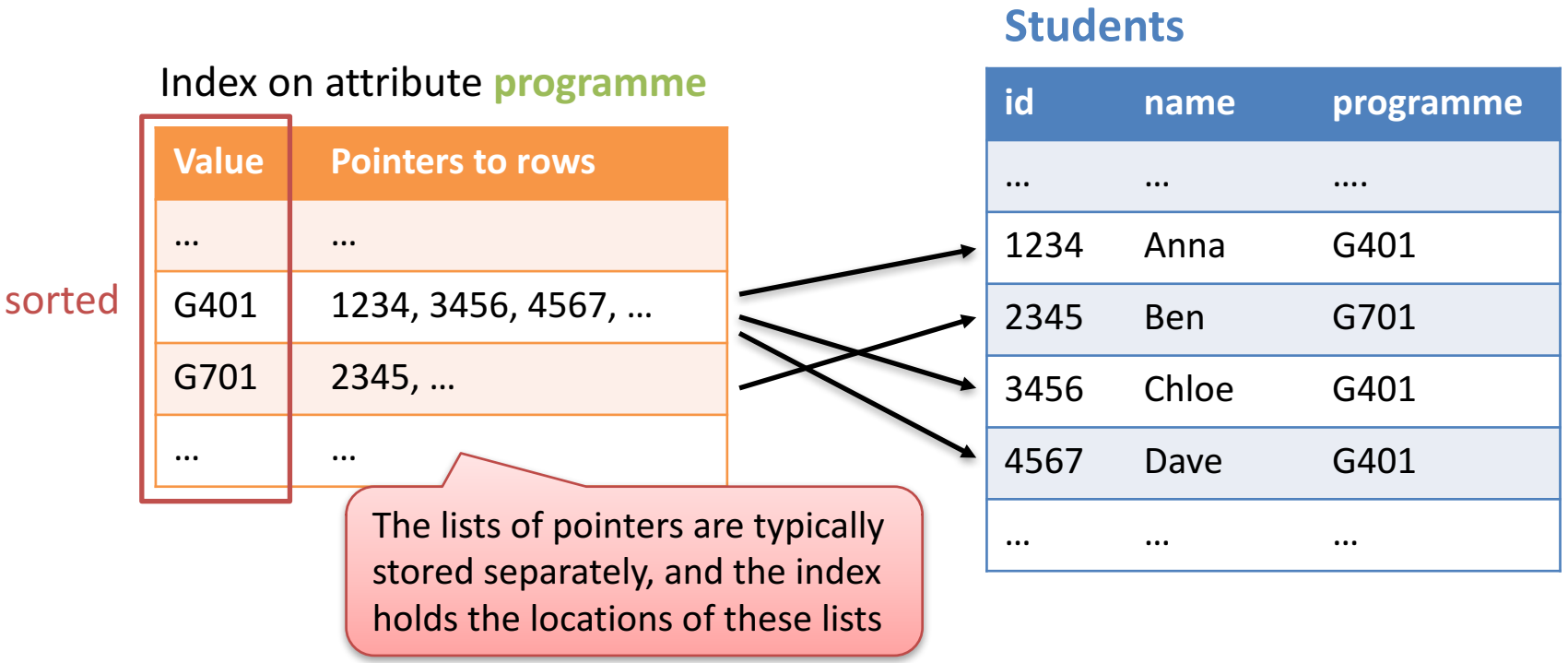
COMP207
Database Development

Lecture 15

Query Processing:
B+ Tree Indexes

Review: Index

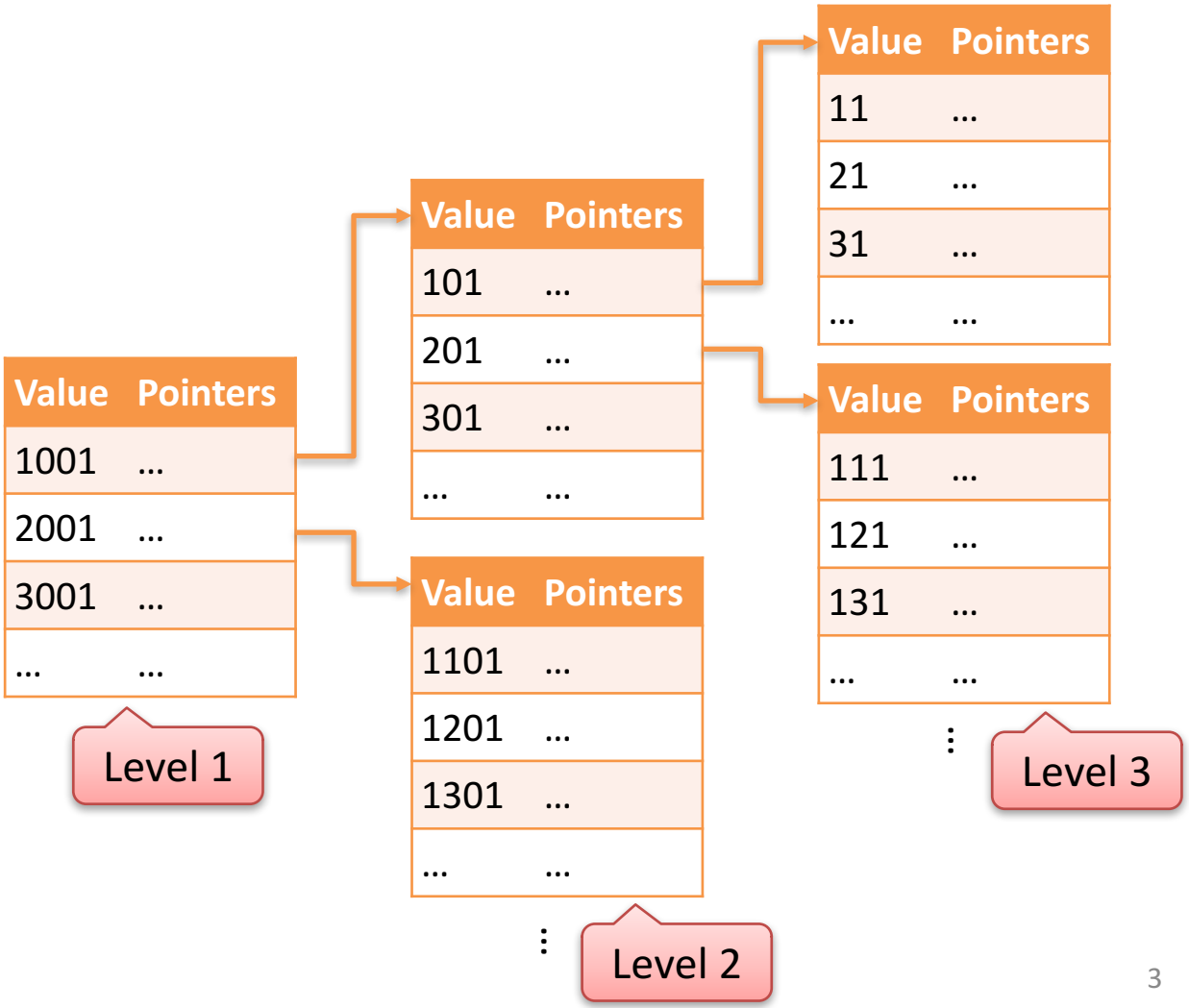
- Takes the values for one or more attributes of a relation **R**, provides quick access to the tuples in **R** with these values



- Different types & shapes

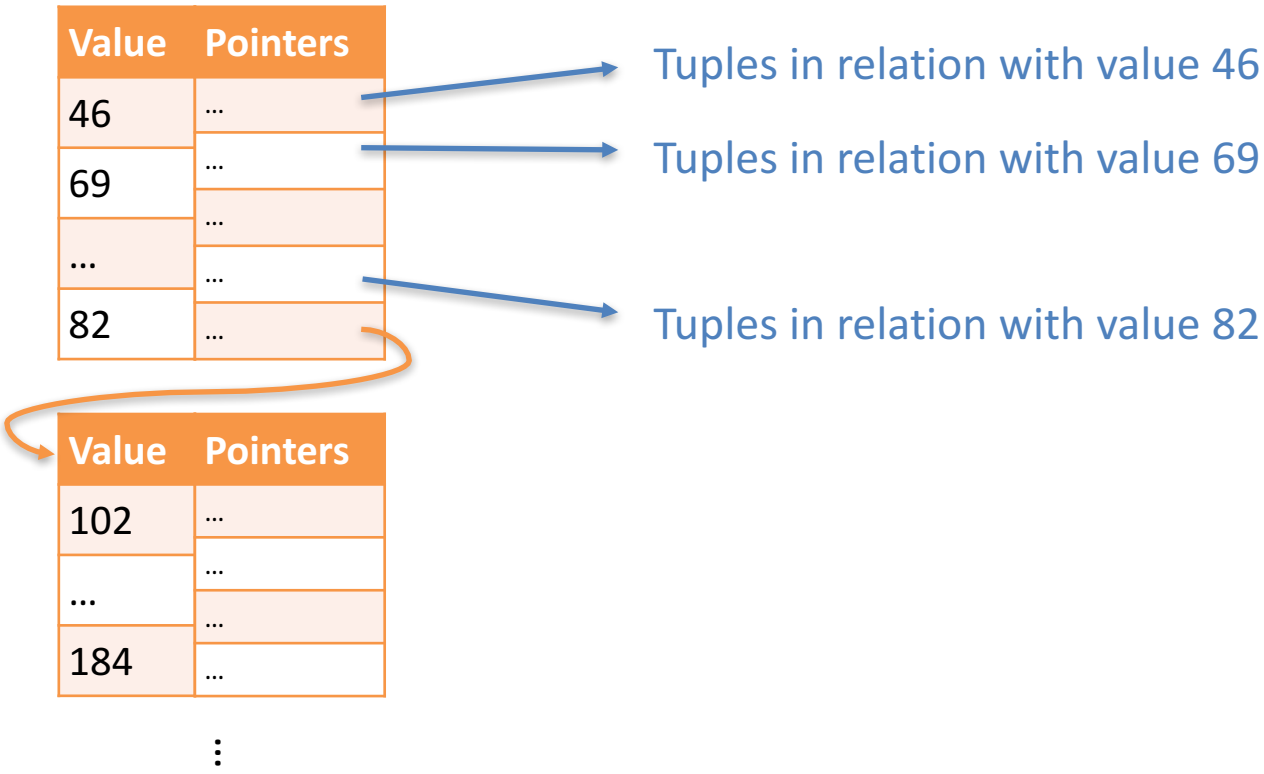
Reminder: Multi-Level Indexes

- Multi-level index: index distributed across different layers

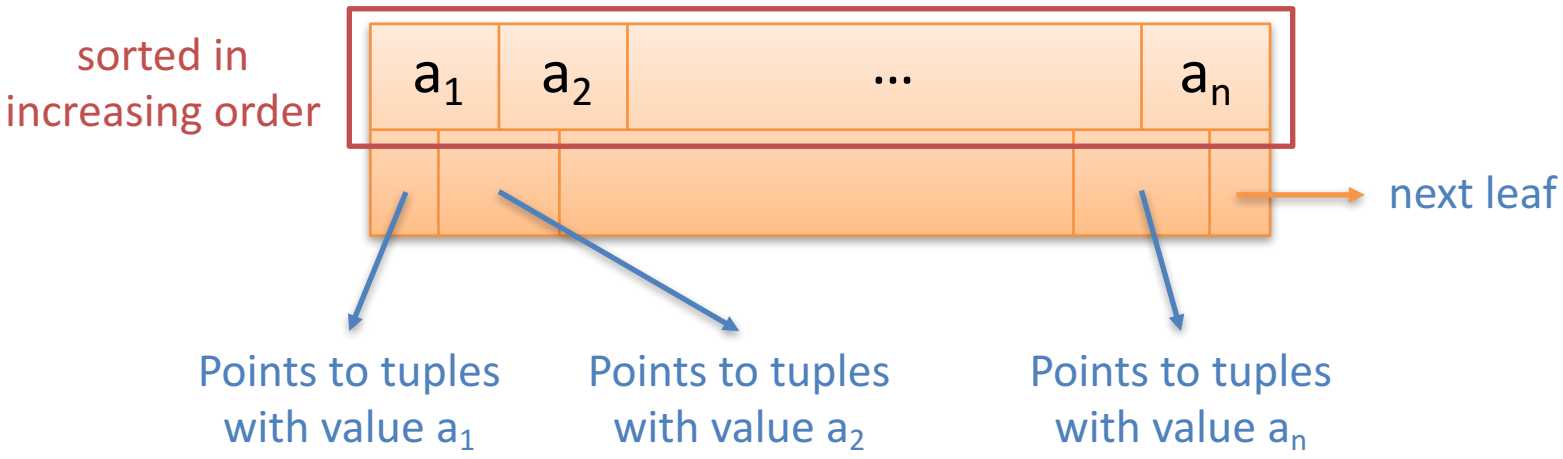


B+ Trees

- A multi-level index, but different shape than the one shown on last slide
- E.g., shape of a leaf in a B+ tree:



B+ Tree: Leaves (Idea)



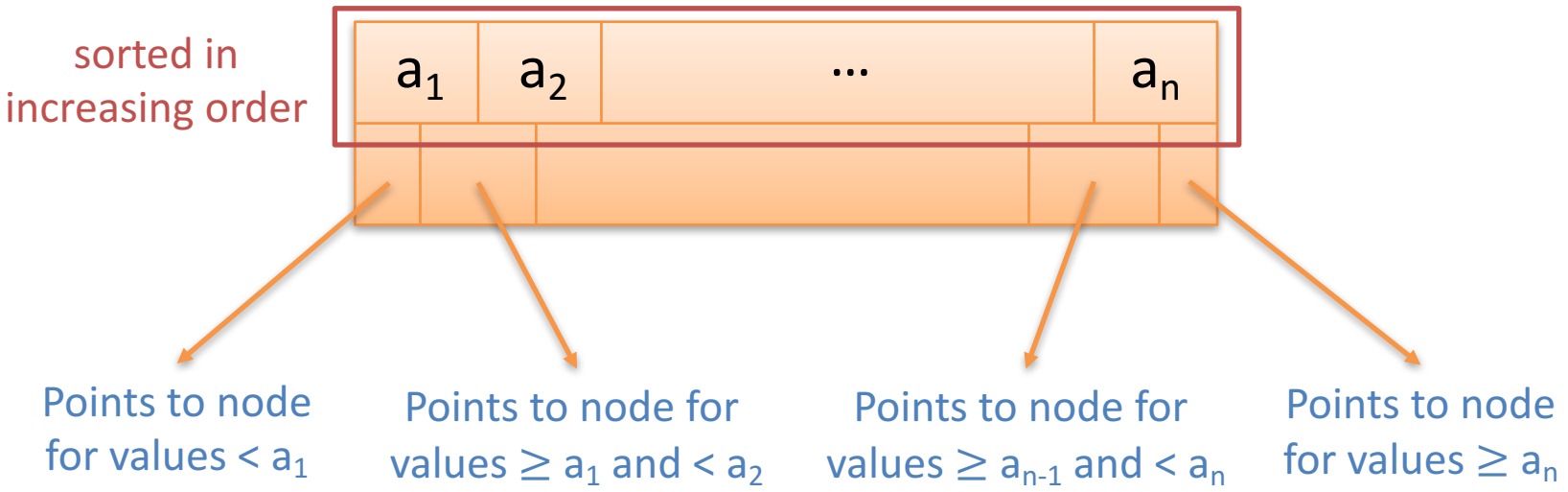
- **n** = chosen such that node fits into a single disk block
- Example:
  - Disk block size = 512 byte
  - Values: 4 byte integers
  - Pointers: 8 bytes

n = 42

$(512 \geq 42\ (8+4)+8$

$512 < 43\ (8+4)+8)$

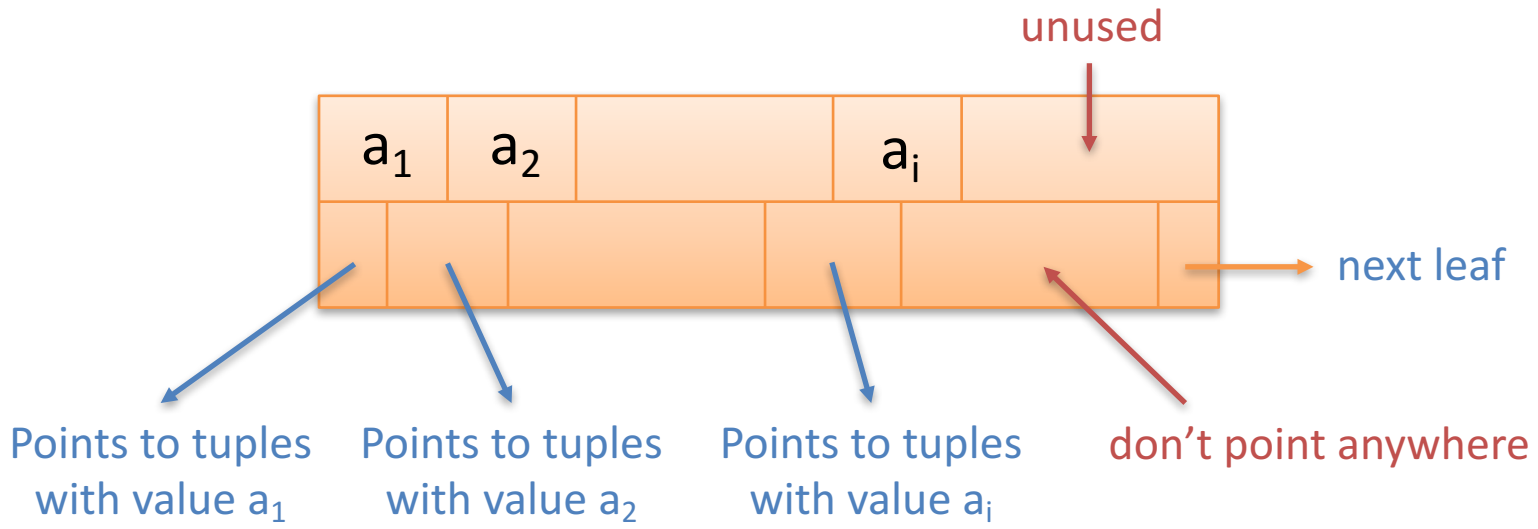
B+ Tree: Inner Nodes (Idea)



- Pointers point to B+ tree nodes at level below
- **n** = chosen as before (≥ 1), so there are ≥ 2 pointers

B+ Tree: Leaves (Actually)

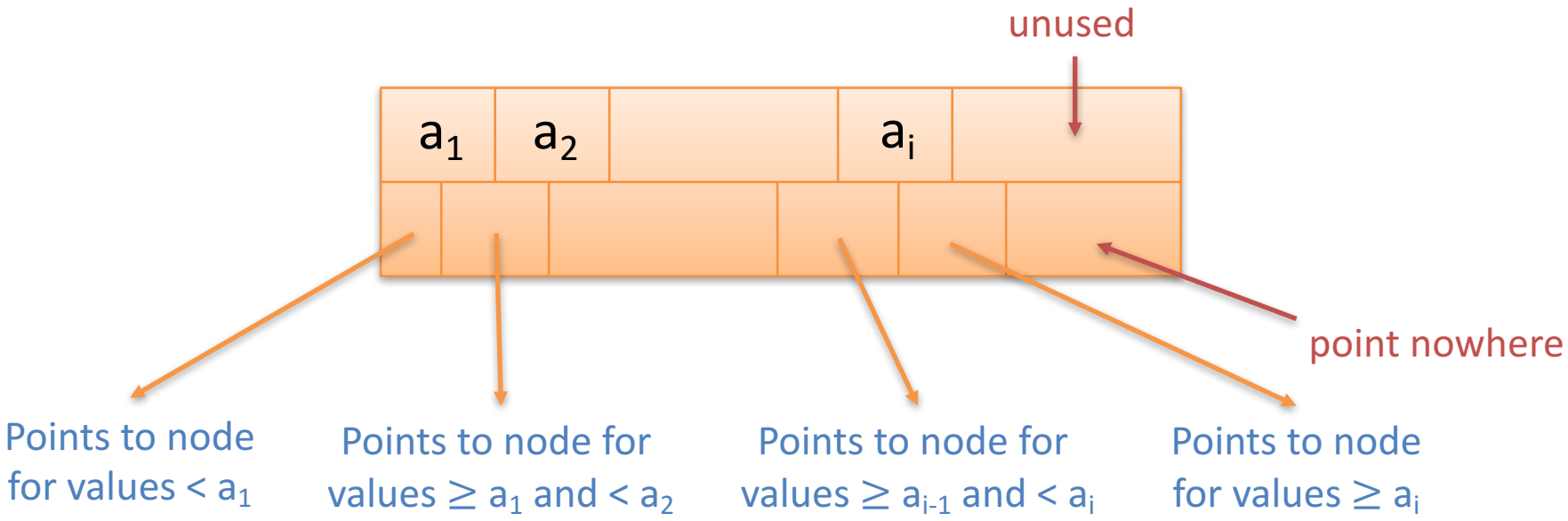
- Not all of the fields have to be used
- Fields are filled from left to right



- Ensure that at least $\left\lfloor \frac{n+1}{2} \right\rfloor$ pointers are used

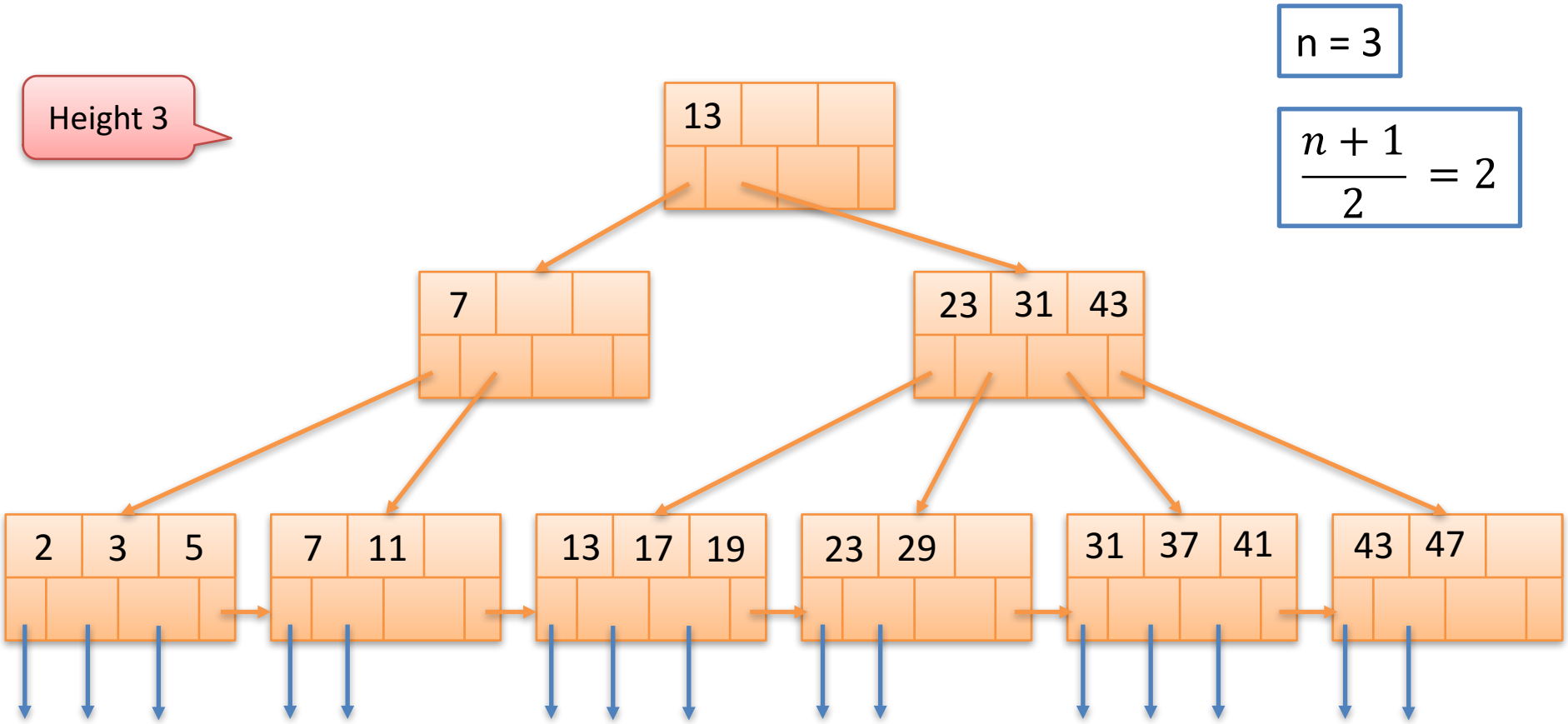
B+ Tree: Inner Nodes (Actually)

- Similar as for leaves:



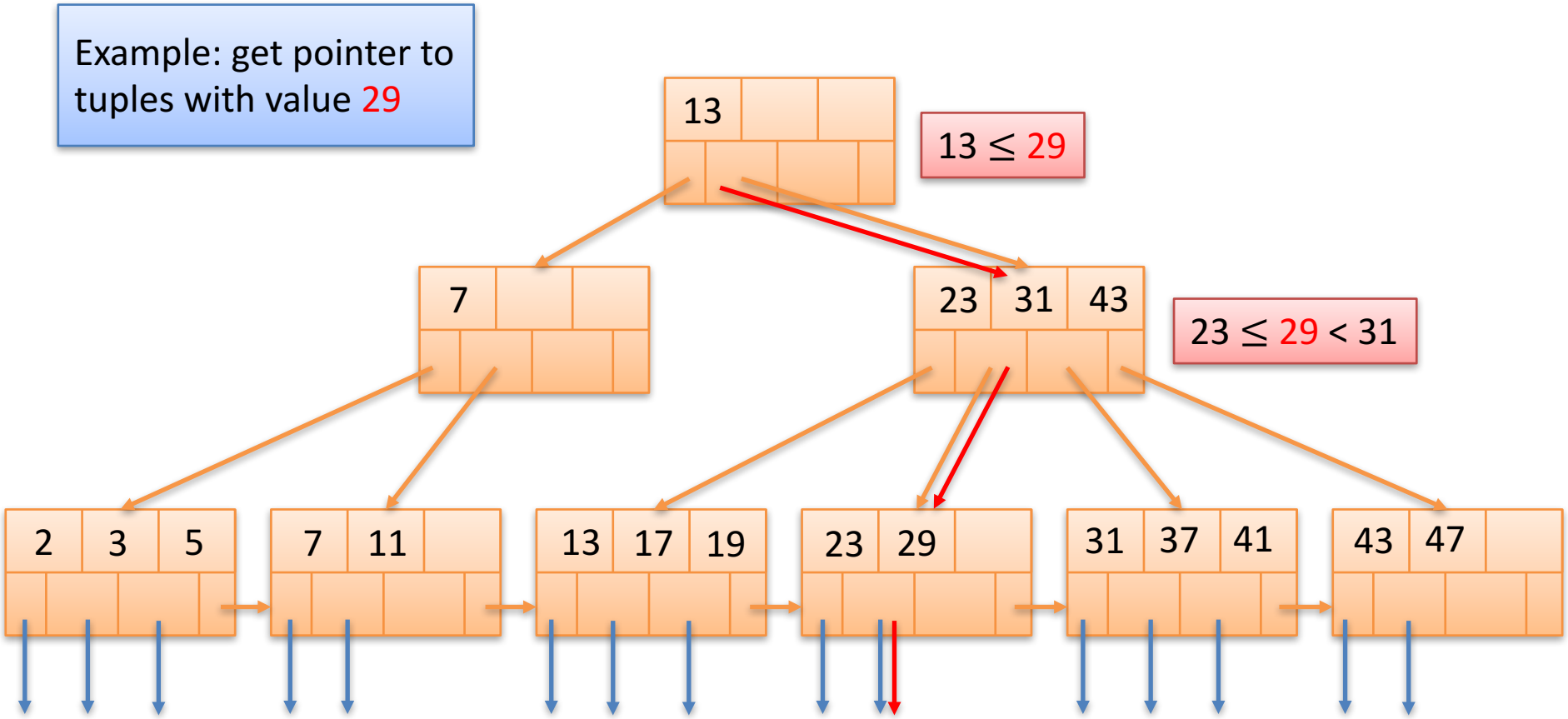
- Ensure that at least $\left\lceil \frac{n+1}{2} \right\rceil$ pointers are used
- Exception: root must use ≥ 2 pointers

A B+ Tree Index



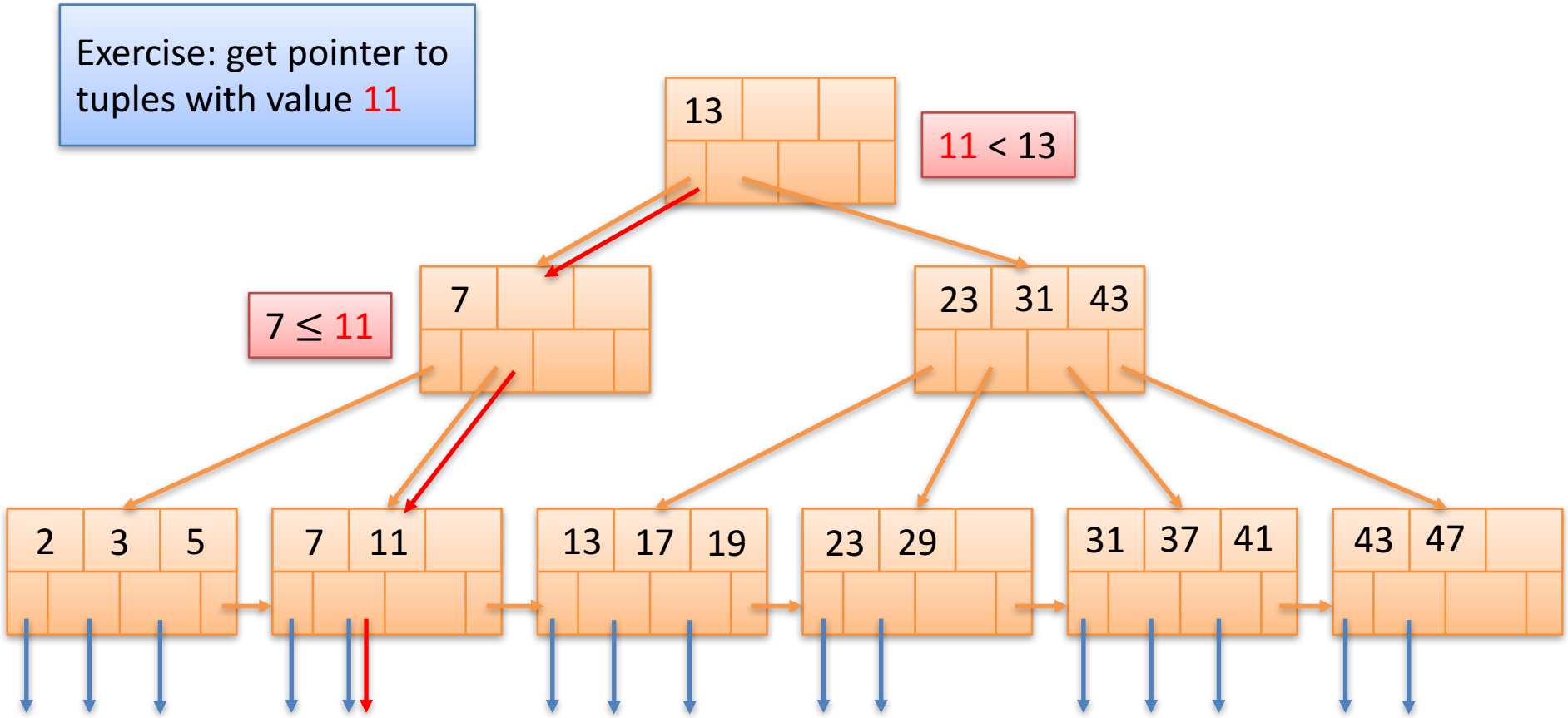
(Example from ”Database Systems: The Complete Book”)

Looking Up Values



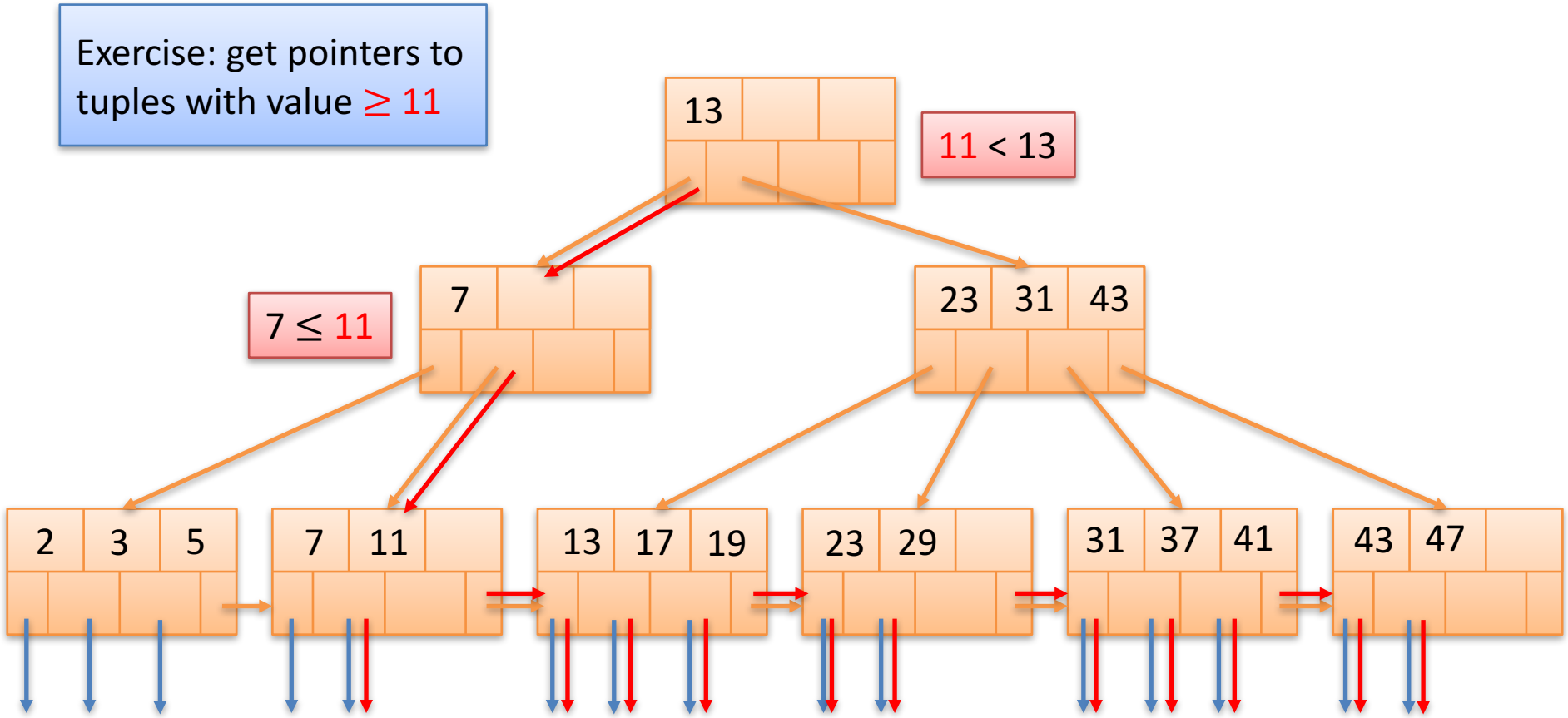
(Example from ”Database Systems: The Complete Book”)

Exercise



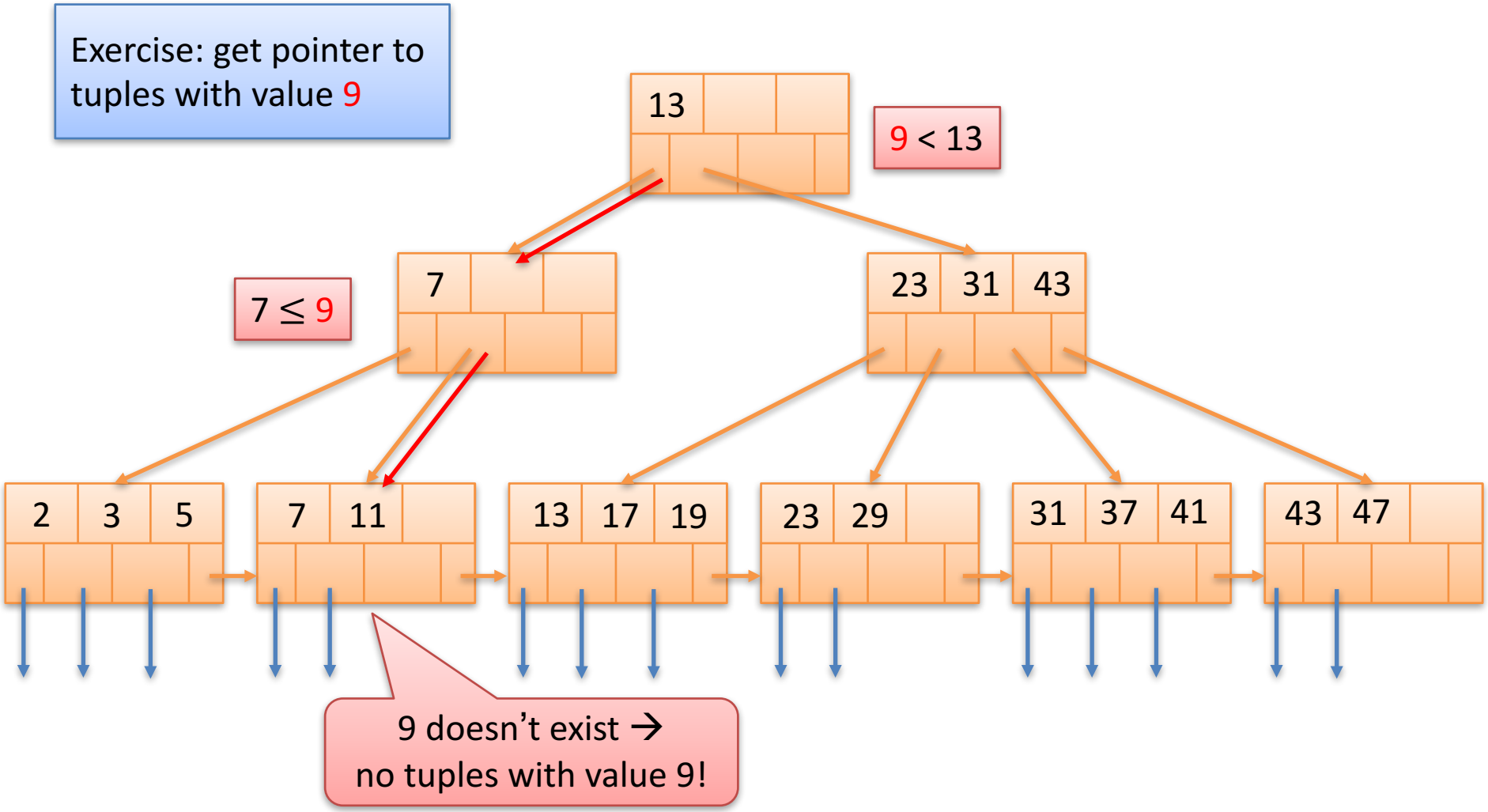
(Example from ”Database Systems: The Complete Book”)

Exercise 2



(Example from ”Database Systems: The Complete Book”)

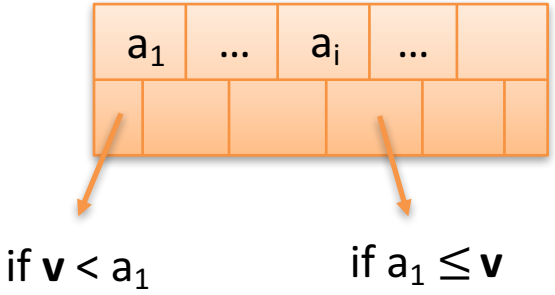
Exercise 3



(Example from "Database Systems: The Complete Book")

Looking Up A Value: Summary

- Goal: find the pointer to the rows with value **v**
- Procedure:
  - Start at the root of the B+ tree
  - While the current node is a non-leaf node:
    - If **v** < a_1, proceed to the first child of the node
    - Otherwise find the **largest i** with $a_i \leq$ **v** and proceed to the associated child node
  - If the current node is a leaf:
    - If **v** occurs in the leaf, follow the associated pointer
    - If **v** does not occur in the leaf, return "**v** does not exist in index"

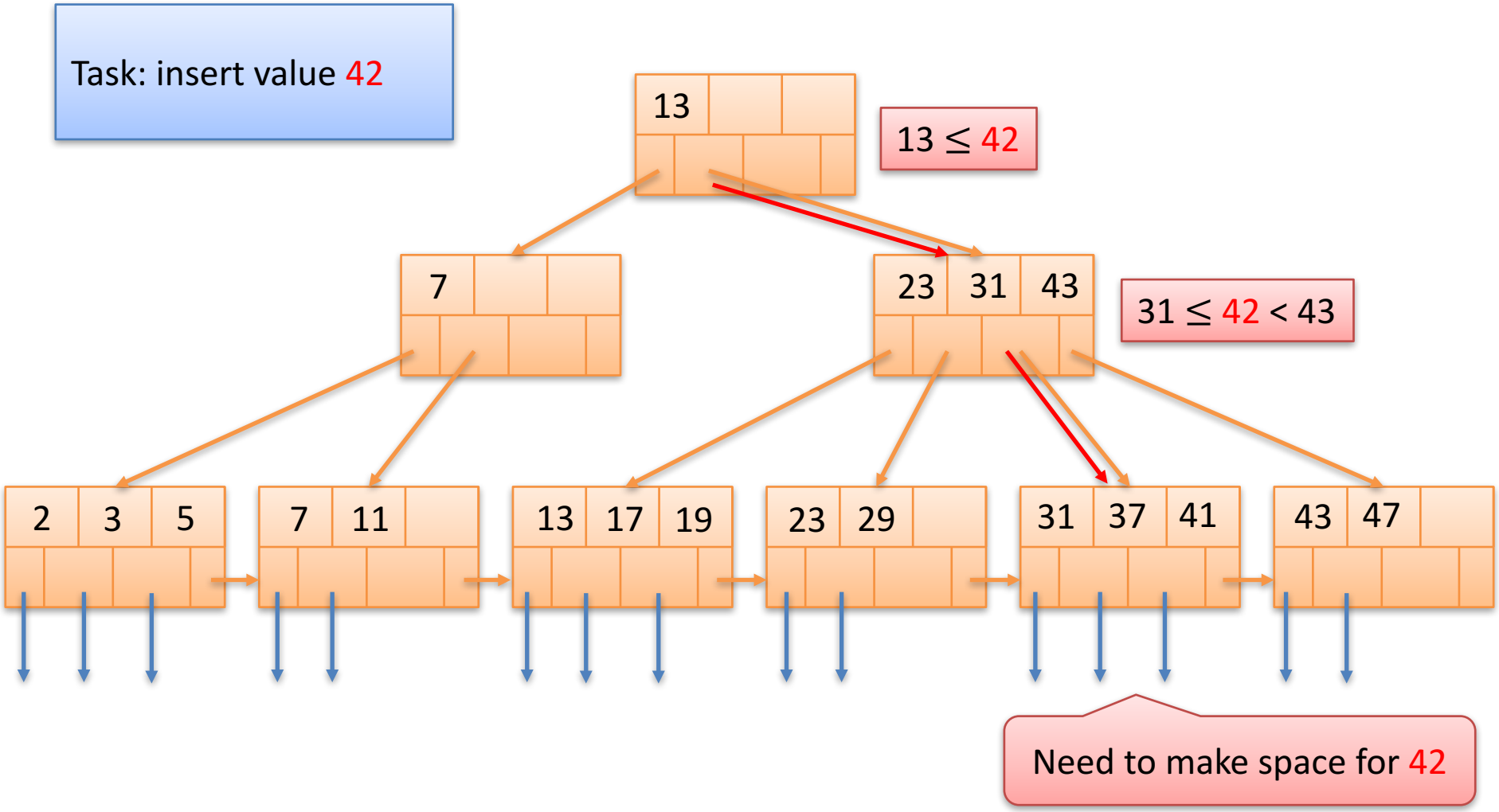


- Running time: $O(h \times \log_2 n)$ "real" running time $O(h \times D)$

Height of the B+ tree

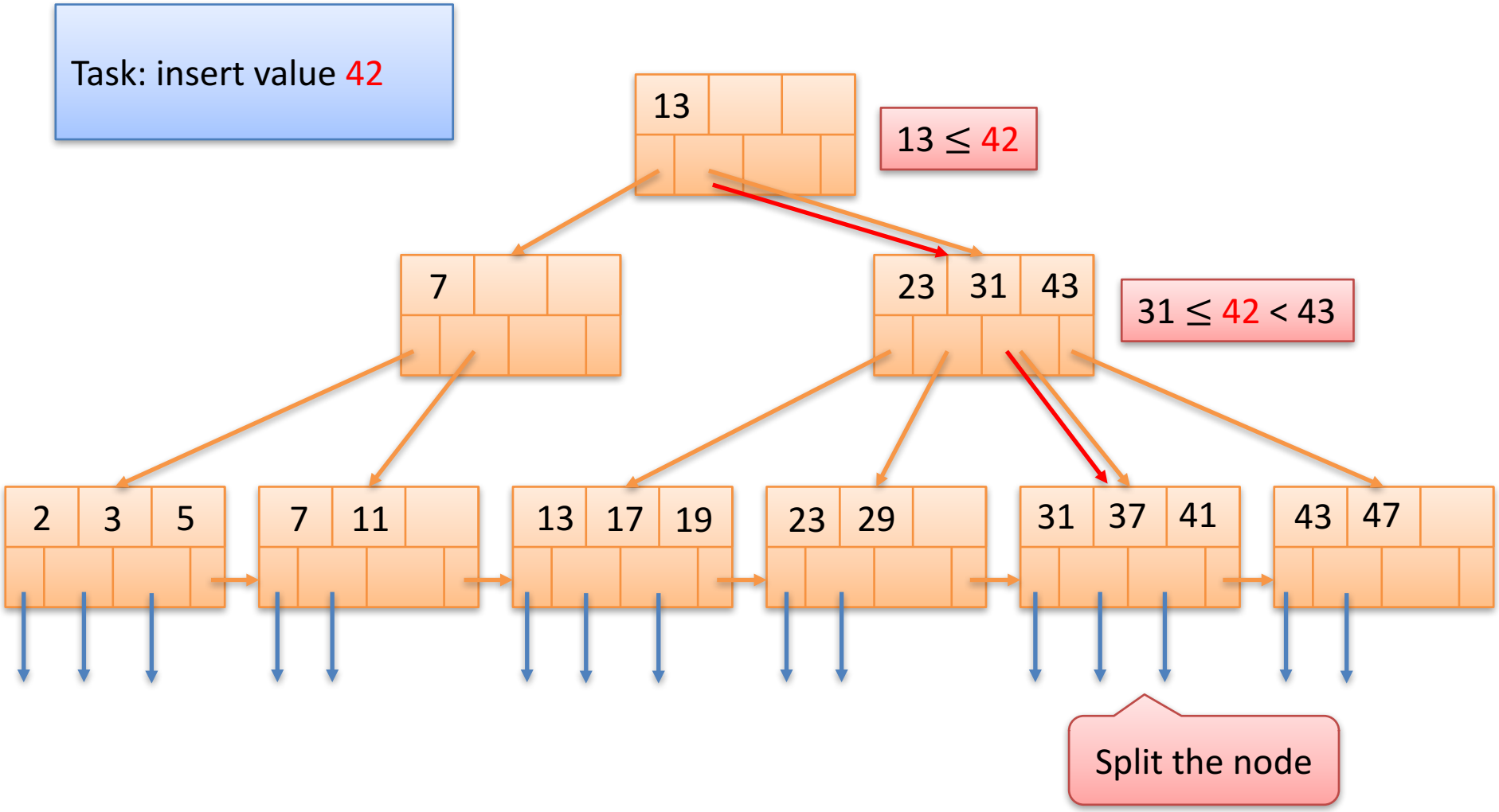
Time for a disk operation

Insertions of Value/Pointer Pairs



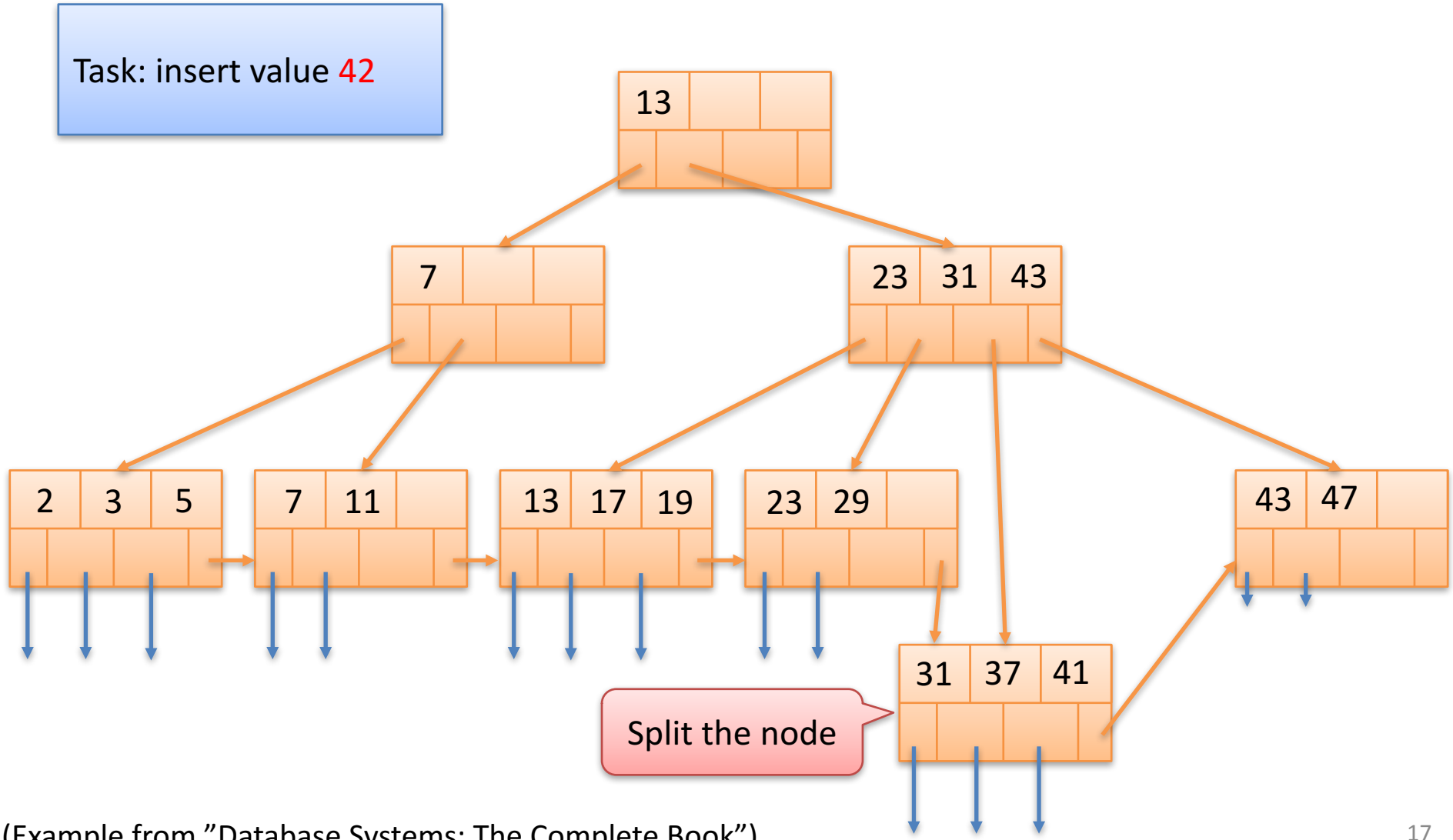
(Example from ”Database Systems: The Complete Book”)

Insertions of Value/Pointer Pairs



(Example from ”Database Systems: The Complete Book”)

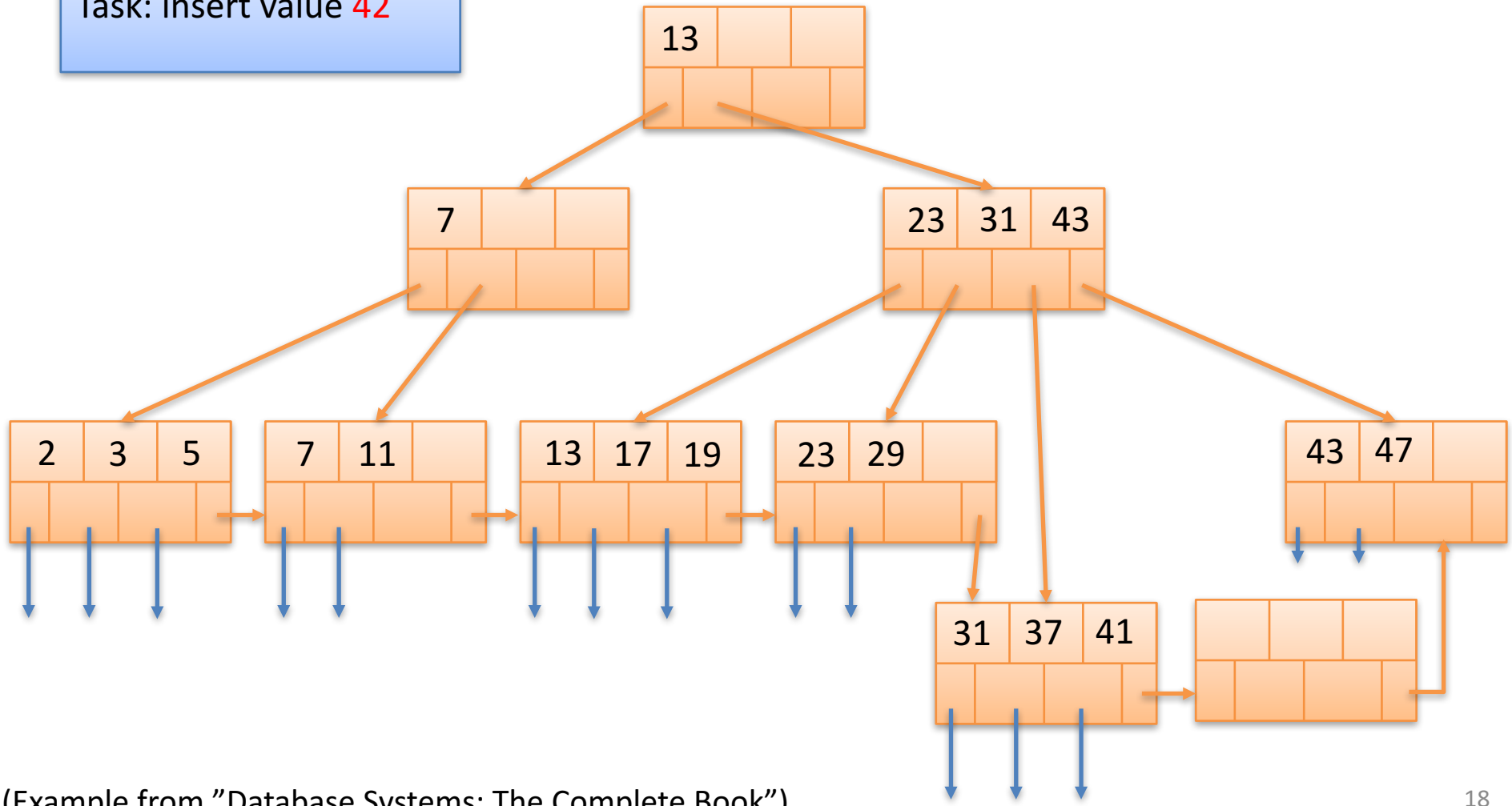
Insertions of Value/Pointer Pairs



(Example from ”Database Systems: The Complete Book”)

Insertions of Value/Pointer Pairs

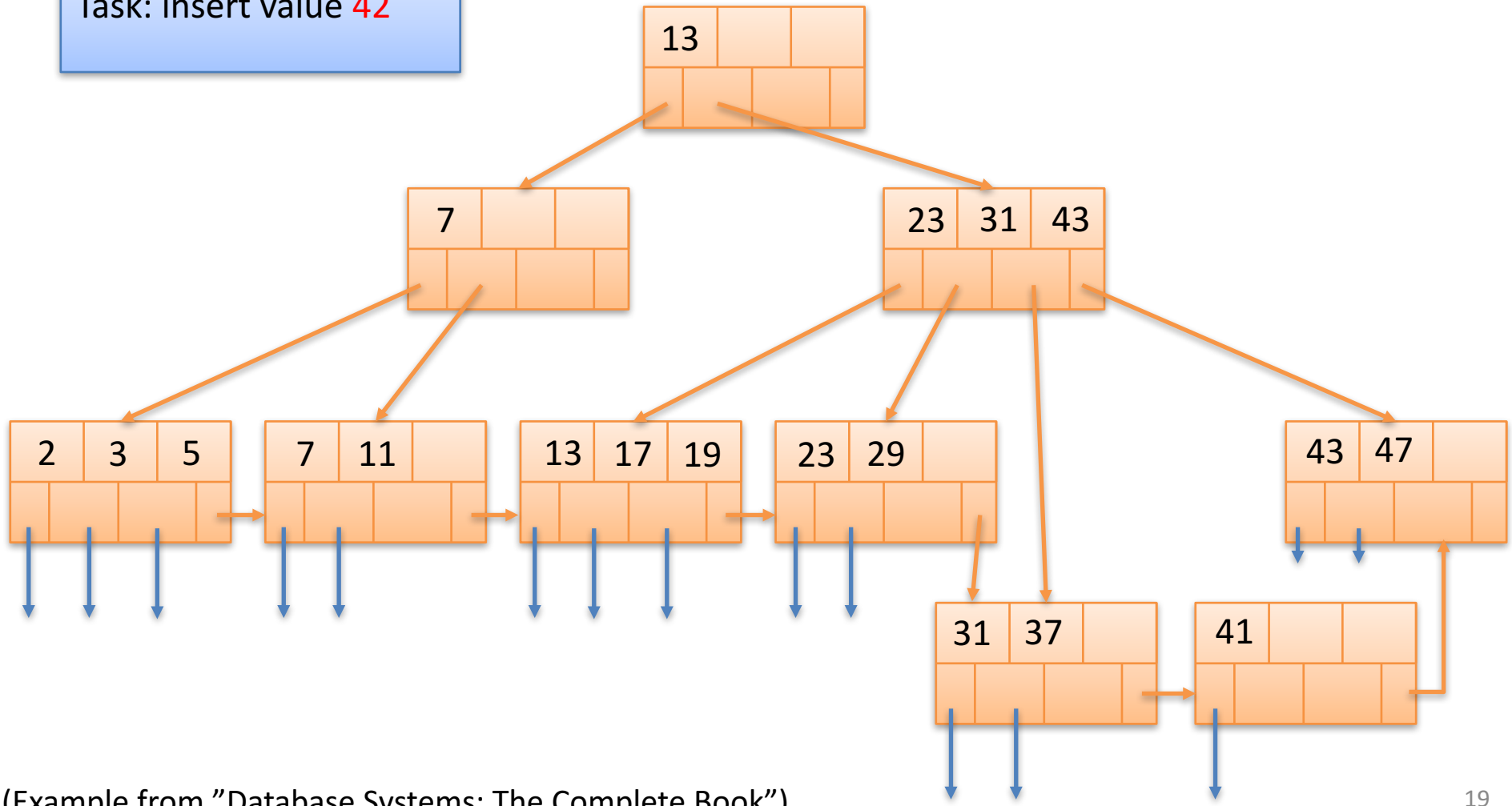
Task: insert value 42



(Example from ”Database Systems: The Complete Book”)

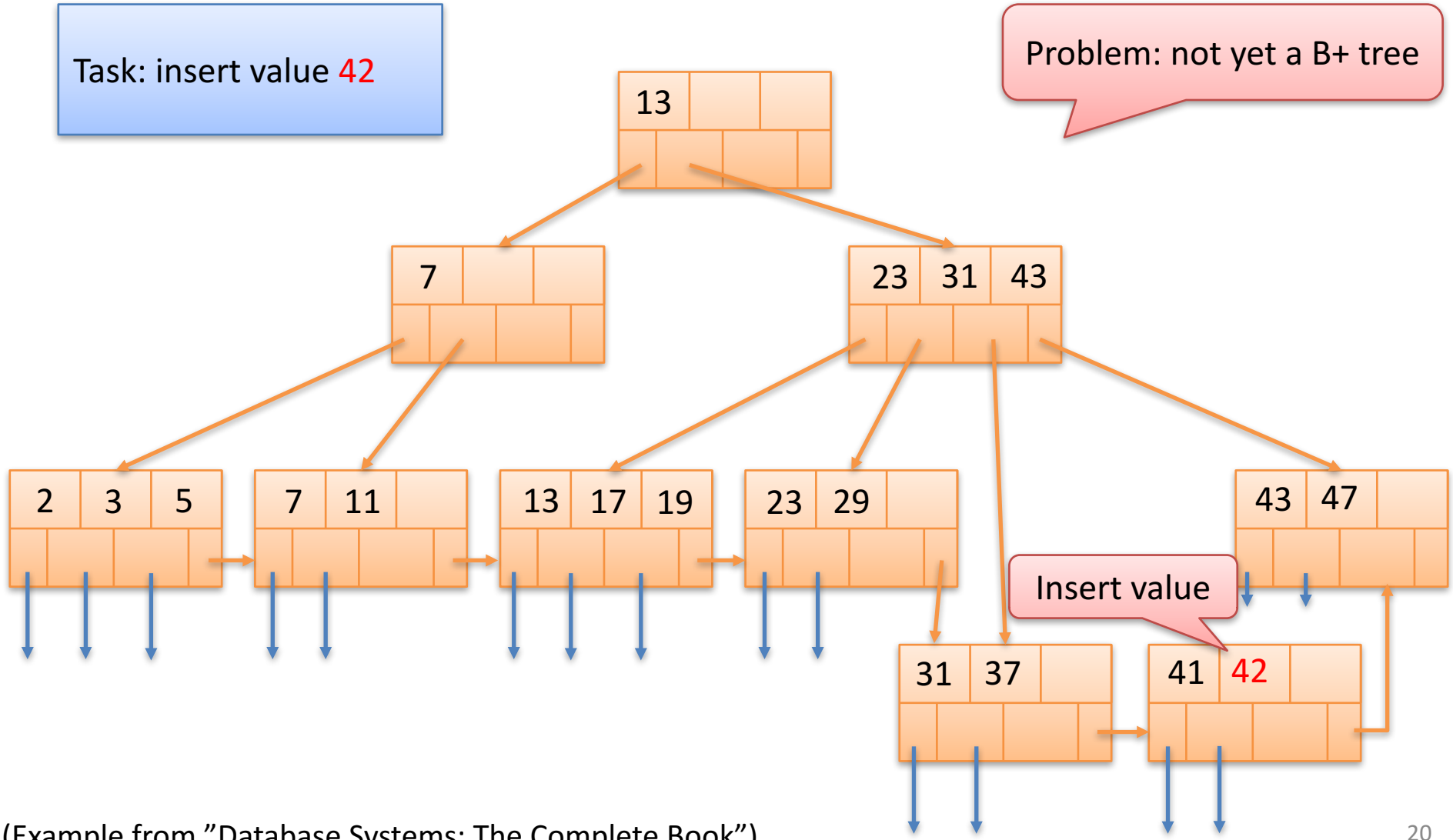
Insertions of Value/Pointer Pairs

Task: insert value 42



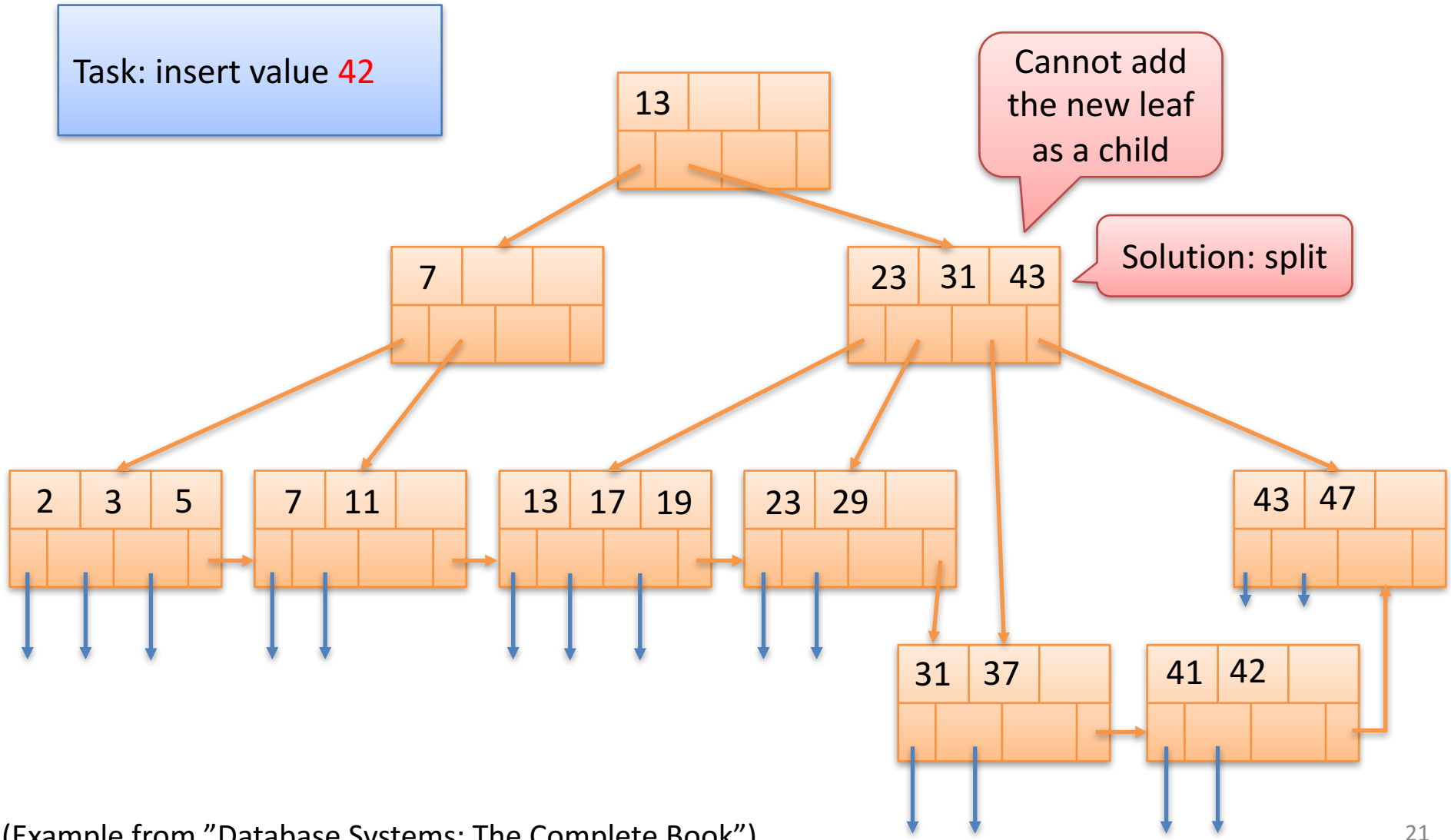
(Example from ”Database Systems: The Complete Book”)

Insertions of Value/Pointer Pairs



(Example from ”Database Systems: The Complete Book”)

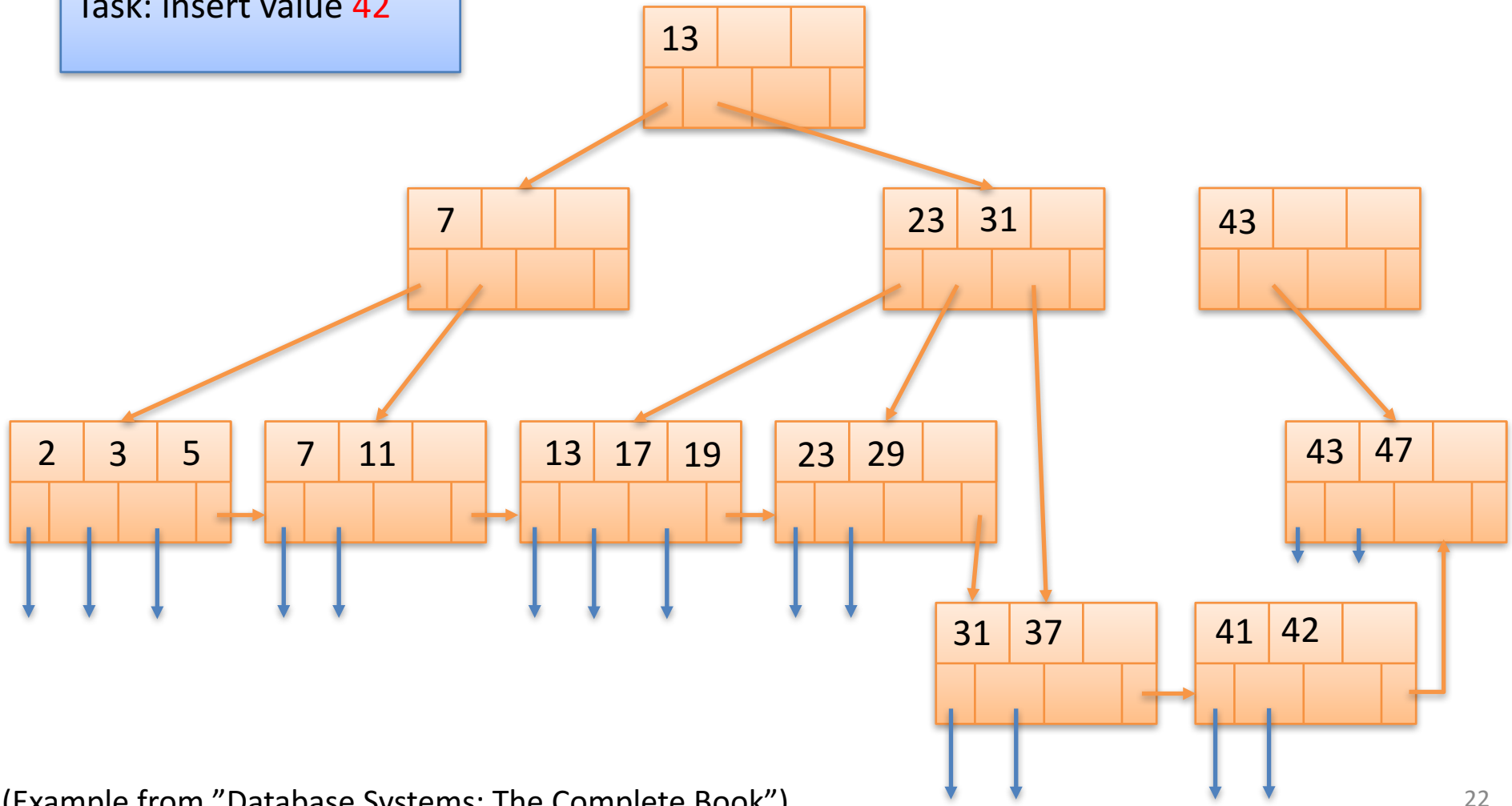
Insertions of Value/Pointer Pairs



(Example from ”Database Systems: The Complete Book”)

Insertions of Value/Pointer Pairs

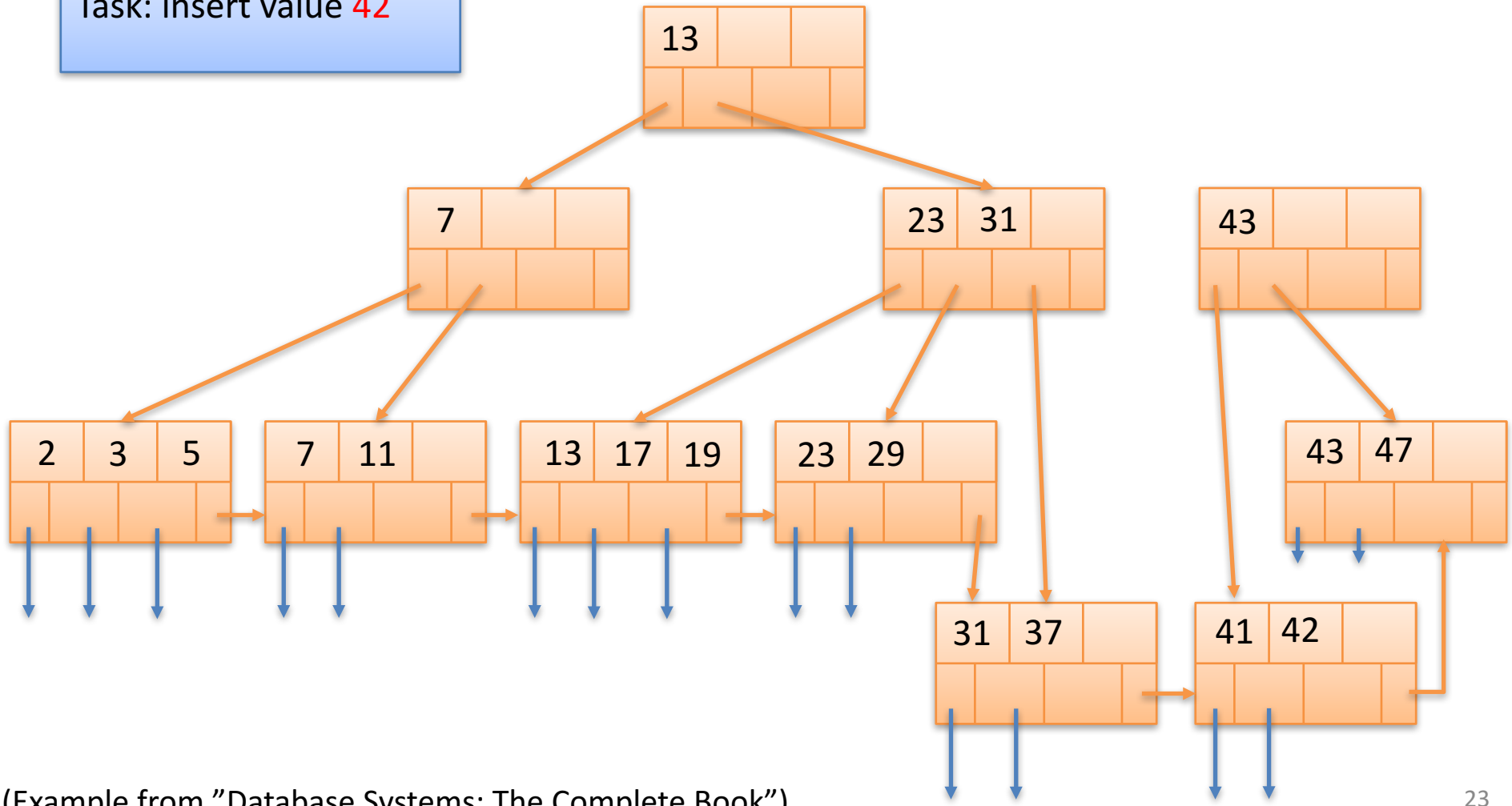
Task: insert value 42



(Example from ”Database Systems: The Complete Book”)

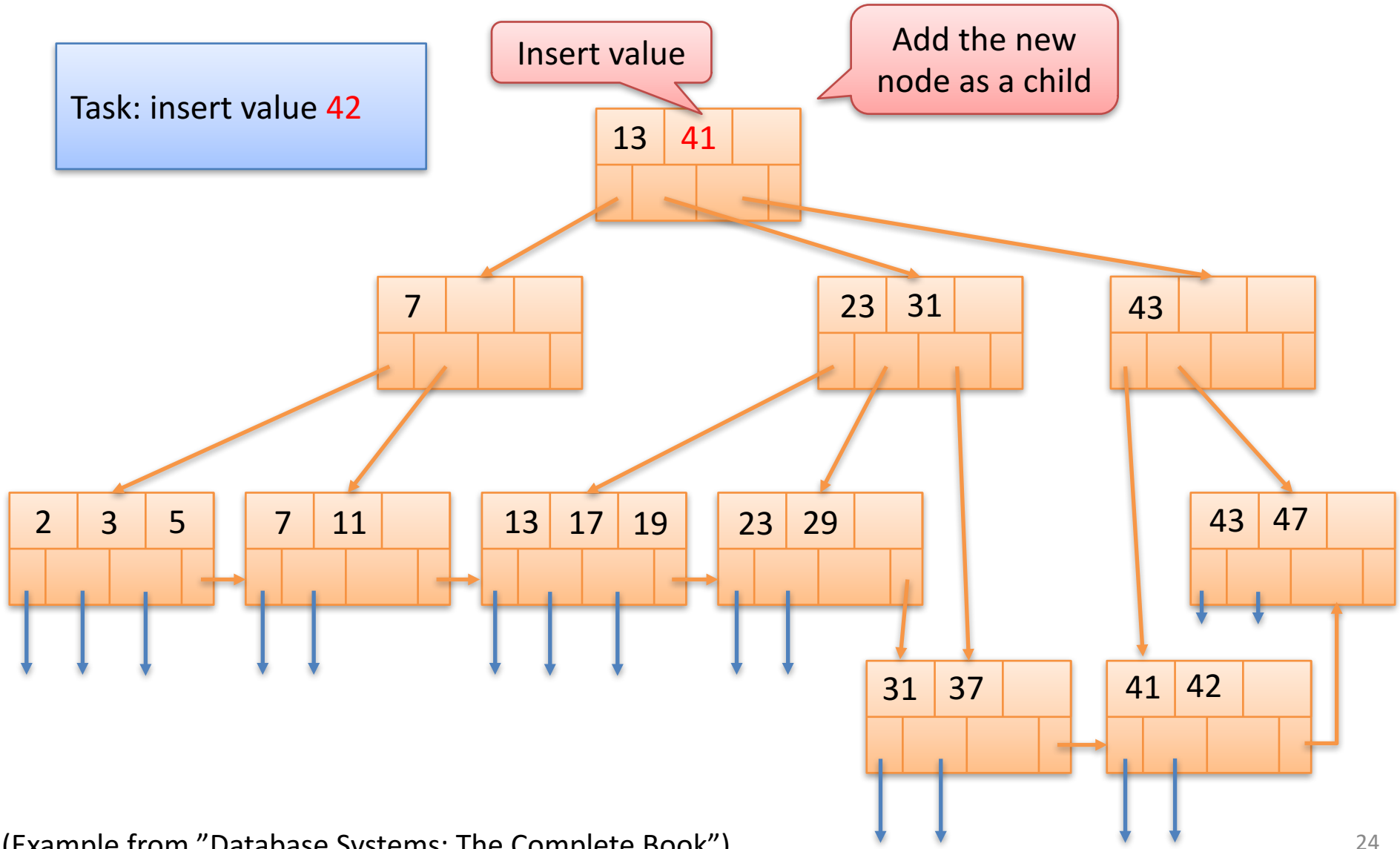
Insertions of Value/Pointer Pairs

Task: insert value 42



(Example from ”Database Systems: The Complete Book”)

Insertions of Value/Pointer Pairs



(Example from ”Database Systems: The Complete Book”)

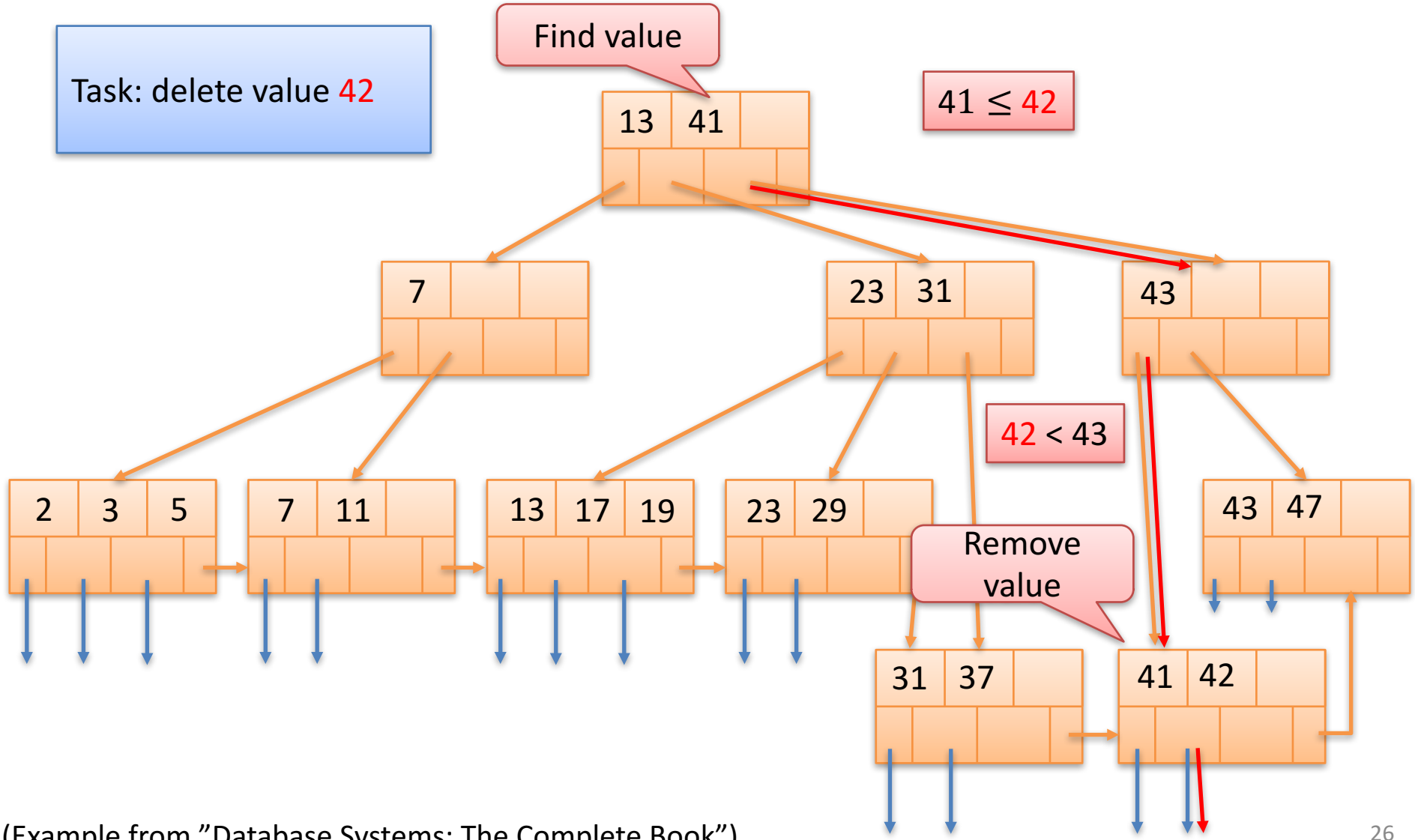
Insertion: Summary

- Goal: insert a new value/pointer pair
- Procedure:
  - Find the leaf that should contain the value
  - If the leaf is not full, insert the key value pair at a suitable location
  - If the leaf is full:
    - Split the leaf to make space for the new value/pointer pair
    - Insert the value/pointer pair
    - Connect the leaf to a suitable parent node (which might incur the creation of a new node etc.)
- The B+ tree **remains balanced**!
- Running time: $O(h \times \log_2 n)$ "real" running time $O(h \times D)$

Time for a disk operation

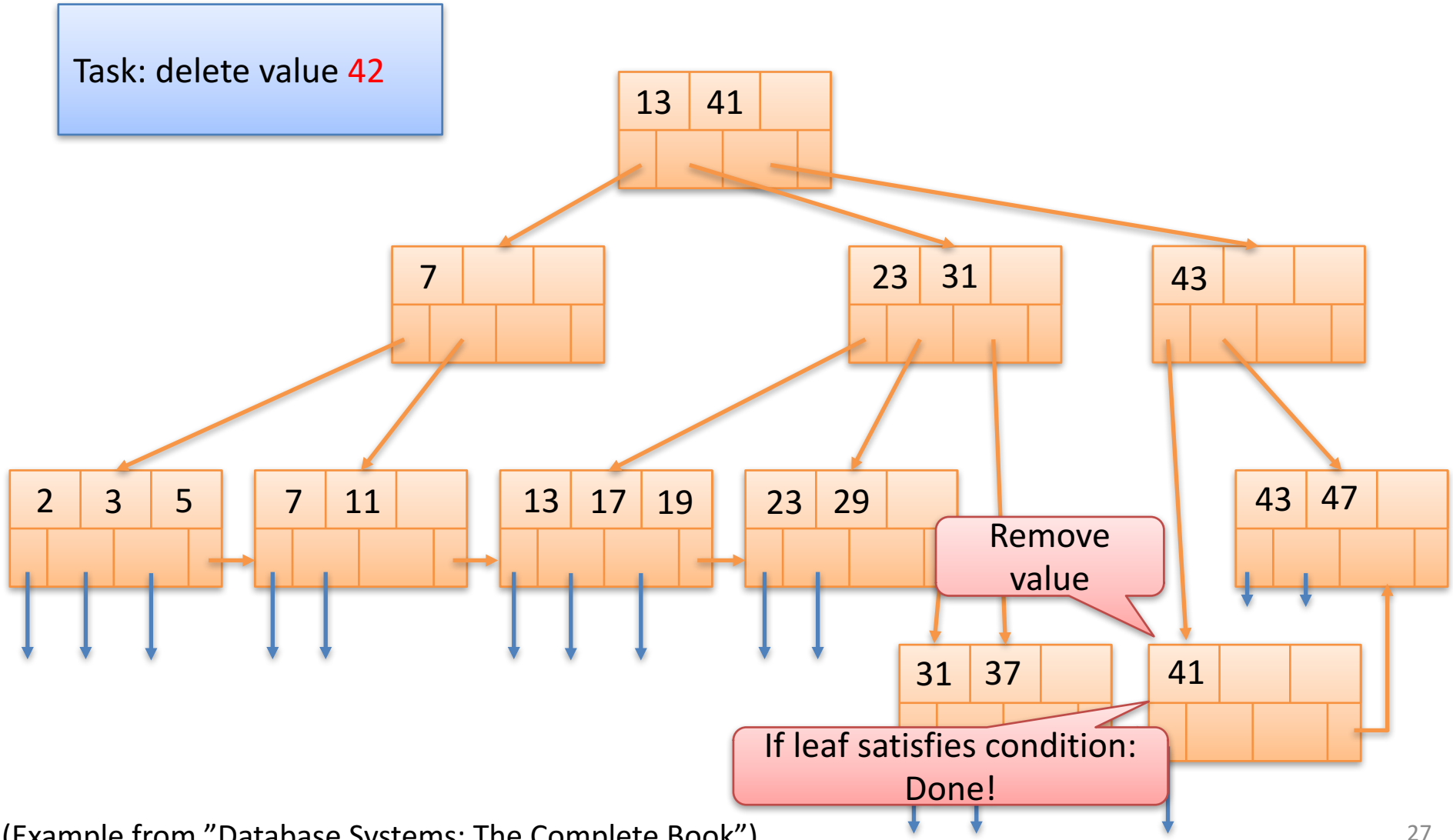
Height of the B+ tree

Deletion



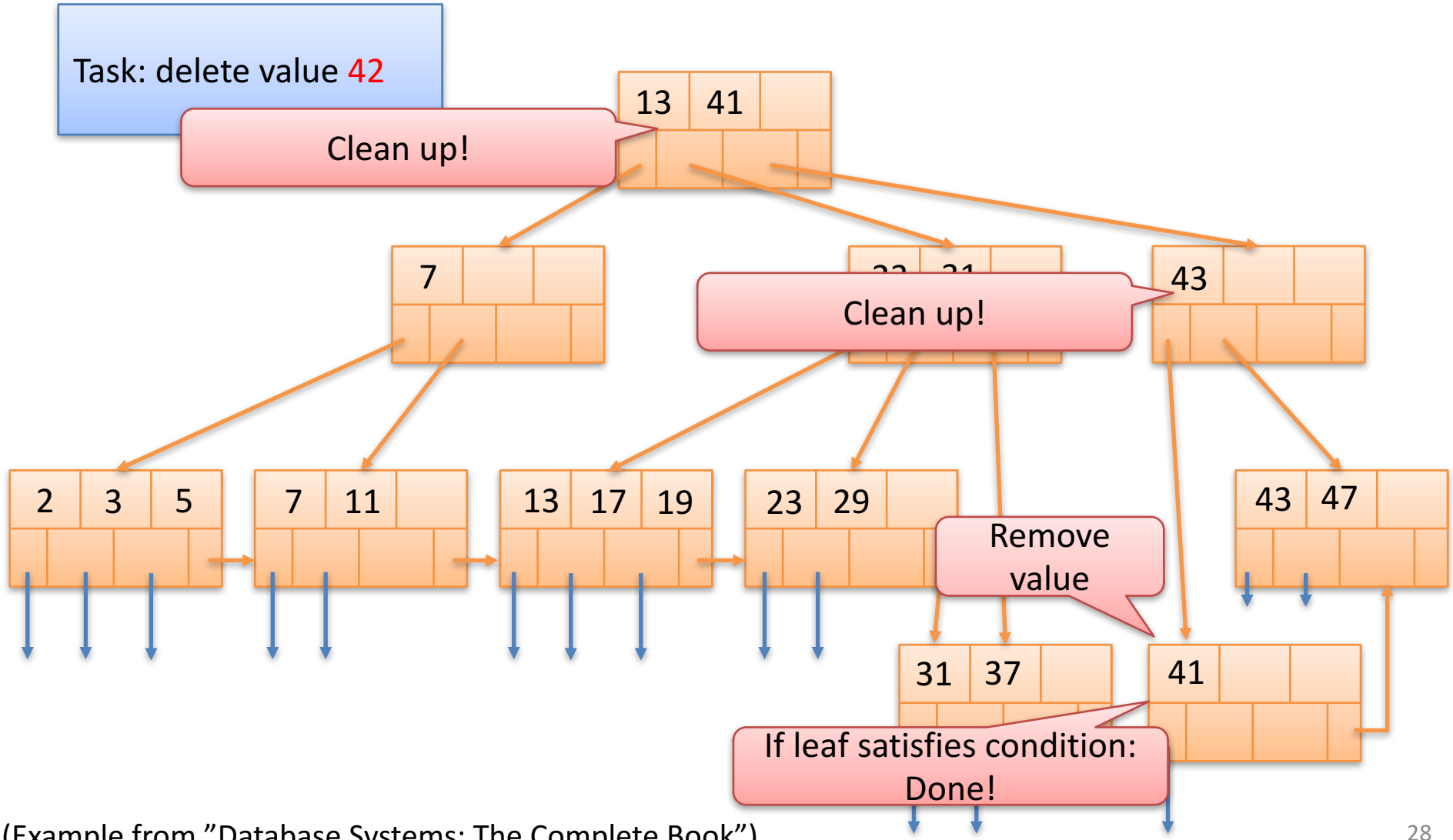
(Example from ”Database Systems: The Complete Book”)

Deletion



(Example from ”Database Systems: The Complete Book”)

Deletion

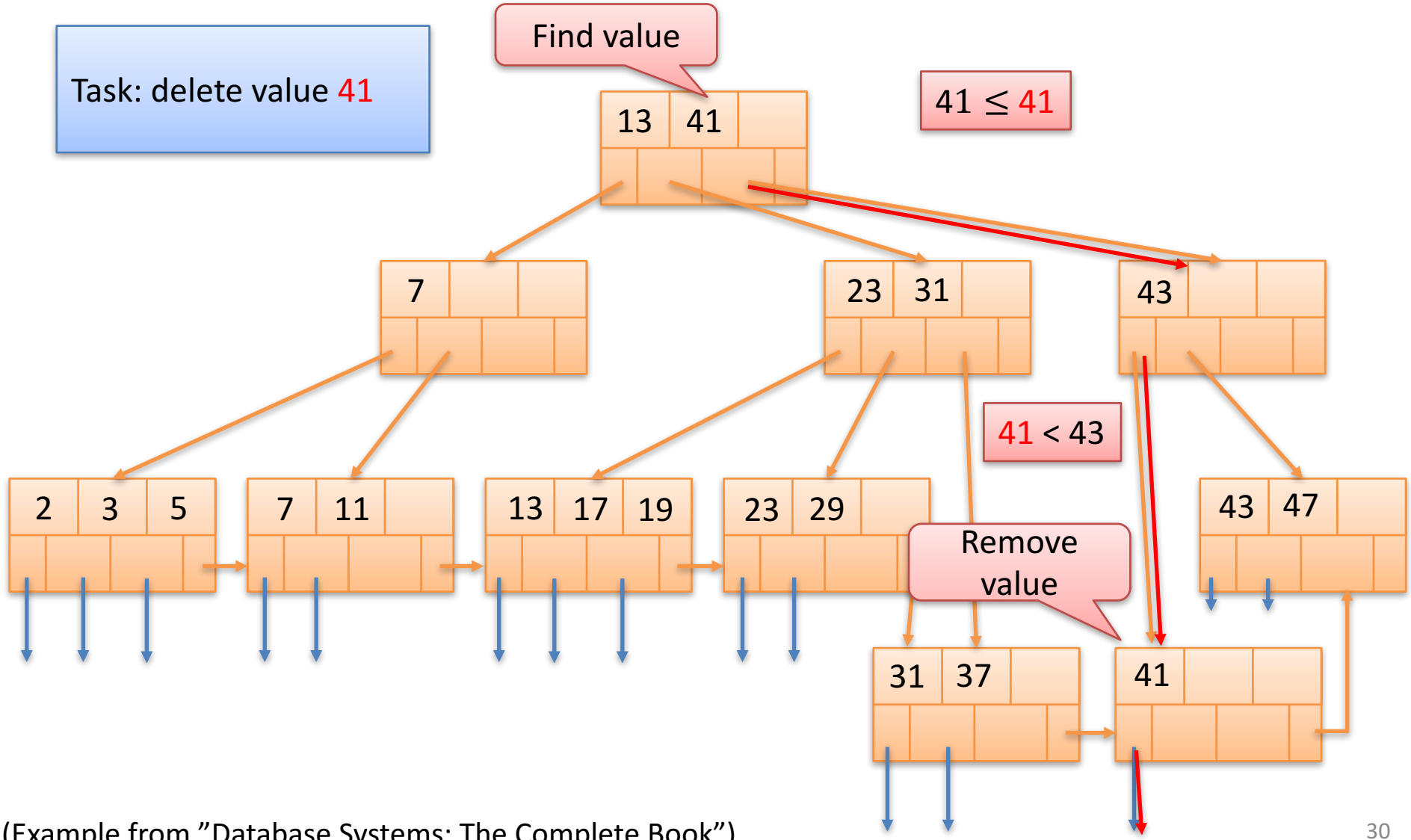


(Example from ”Database Systems: The Complete Book”)

Side note

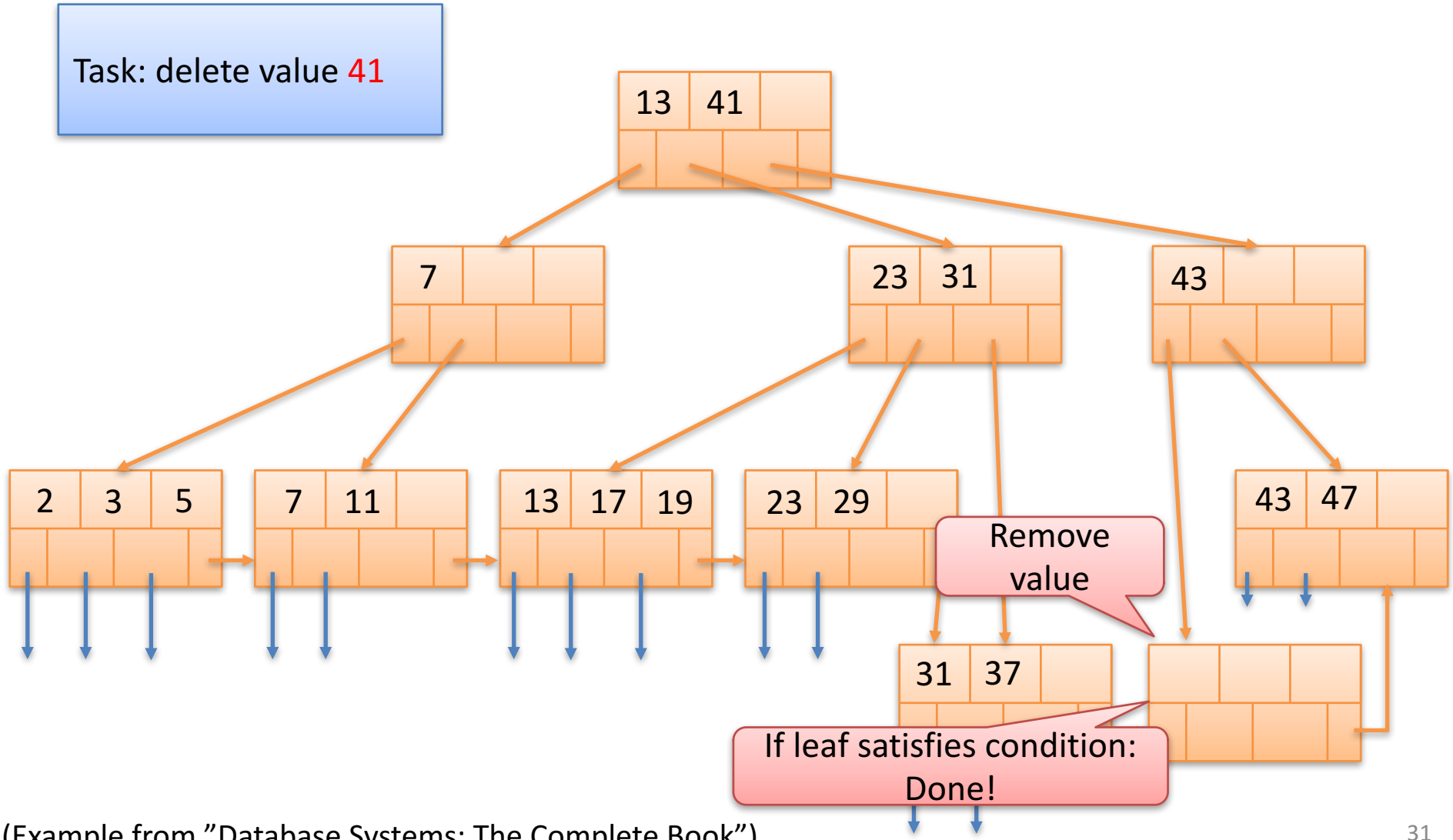
- There may be more than one B+ tree with the same leaves

Deletion part 2



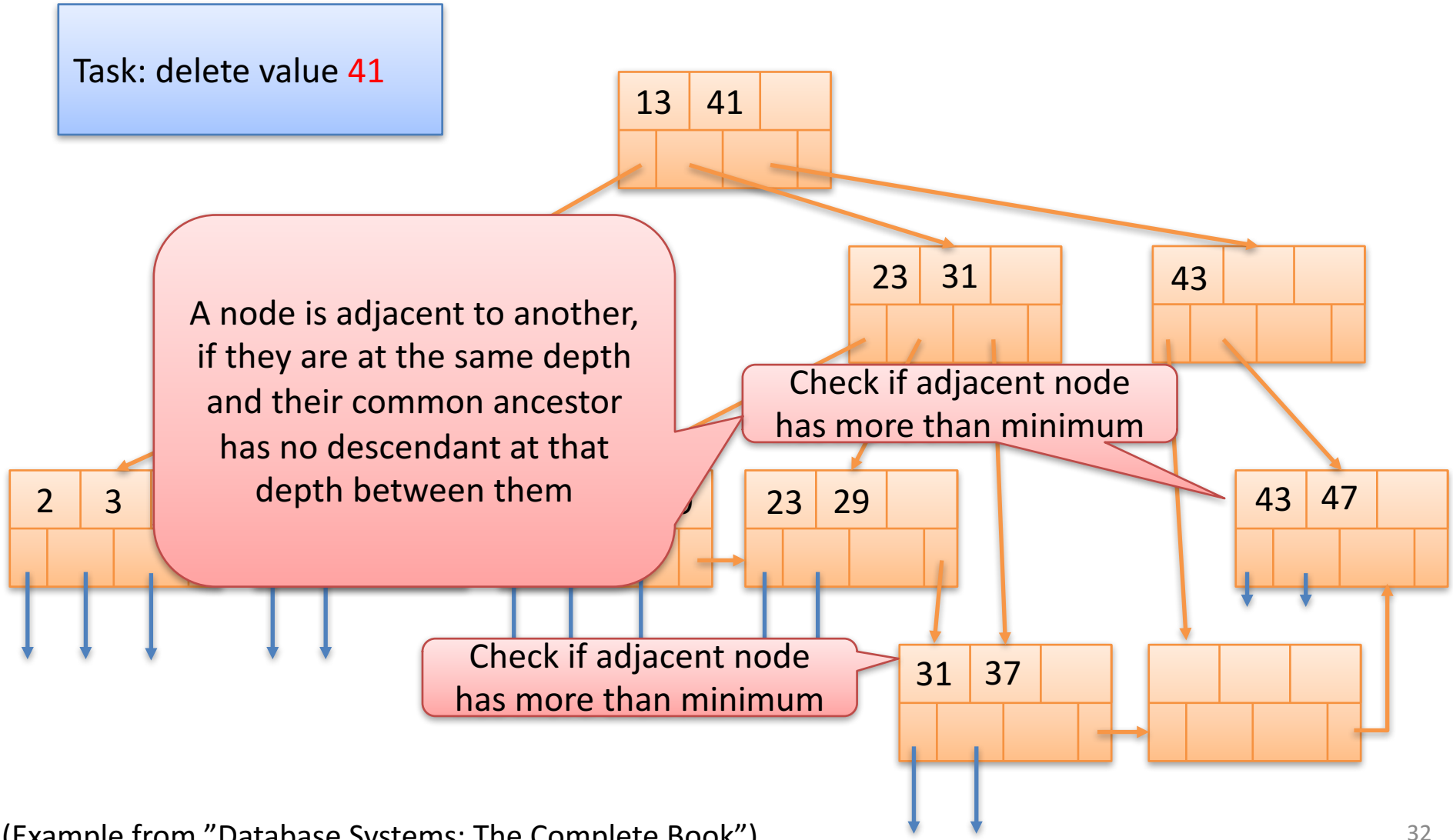
(Example from ”Database Systems: The Complete Book”)

Deletion part 2



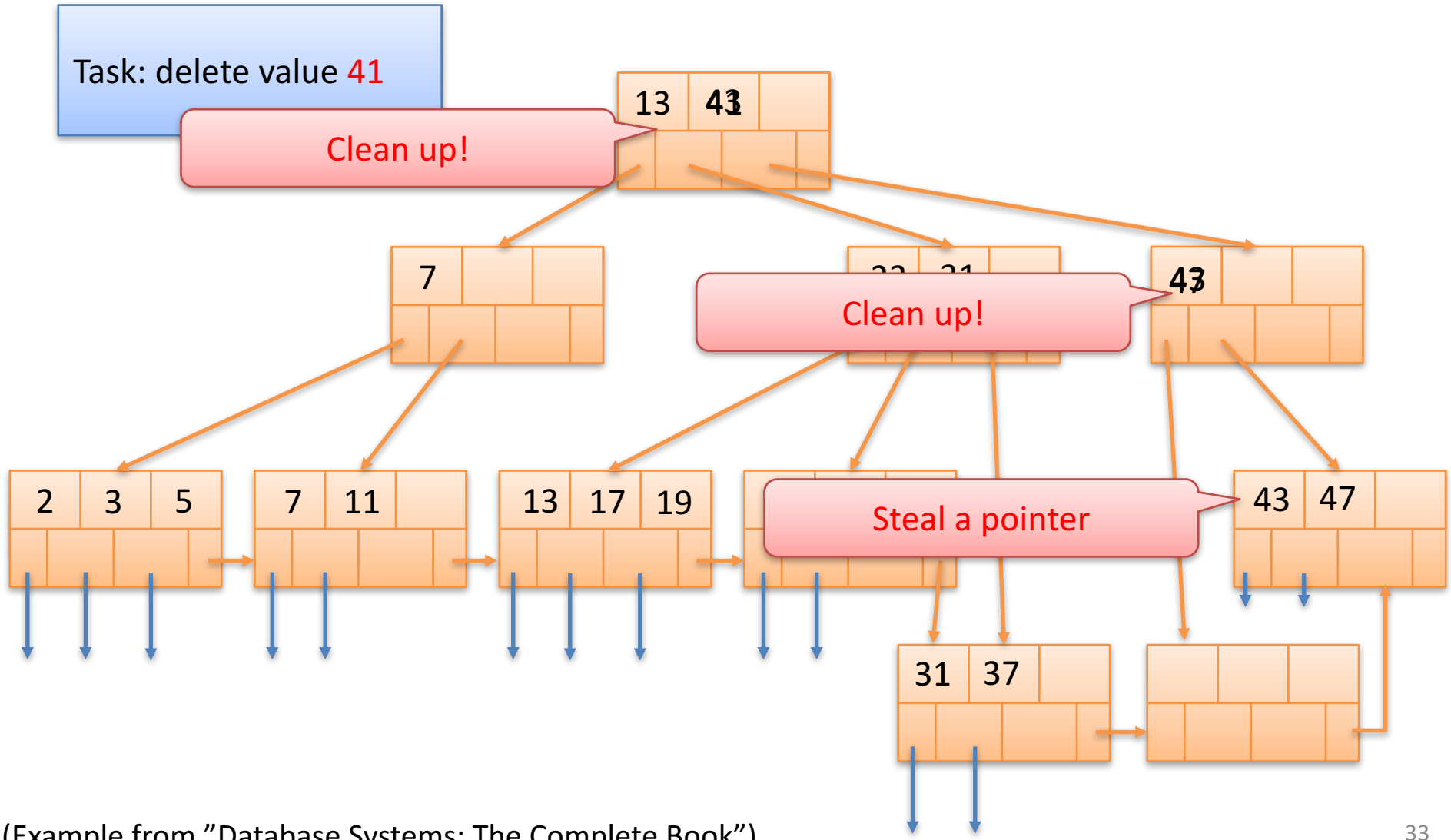
(Example from ”Database Systems: The Complete Book”)

Deletion part 2



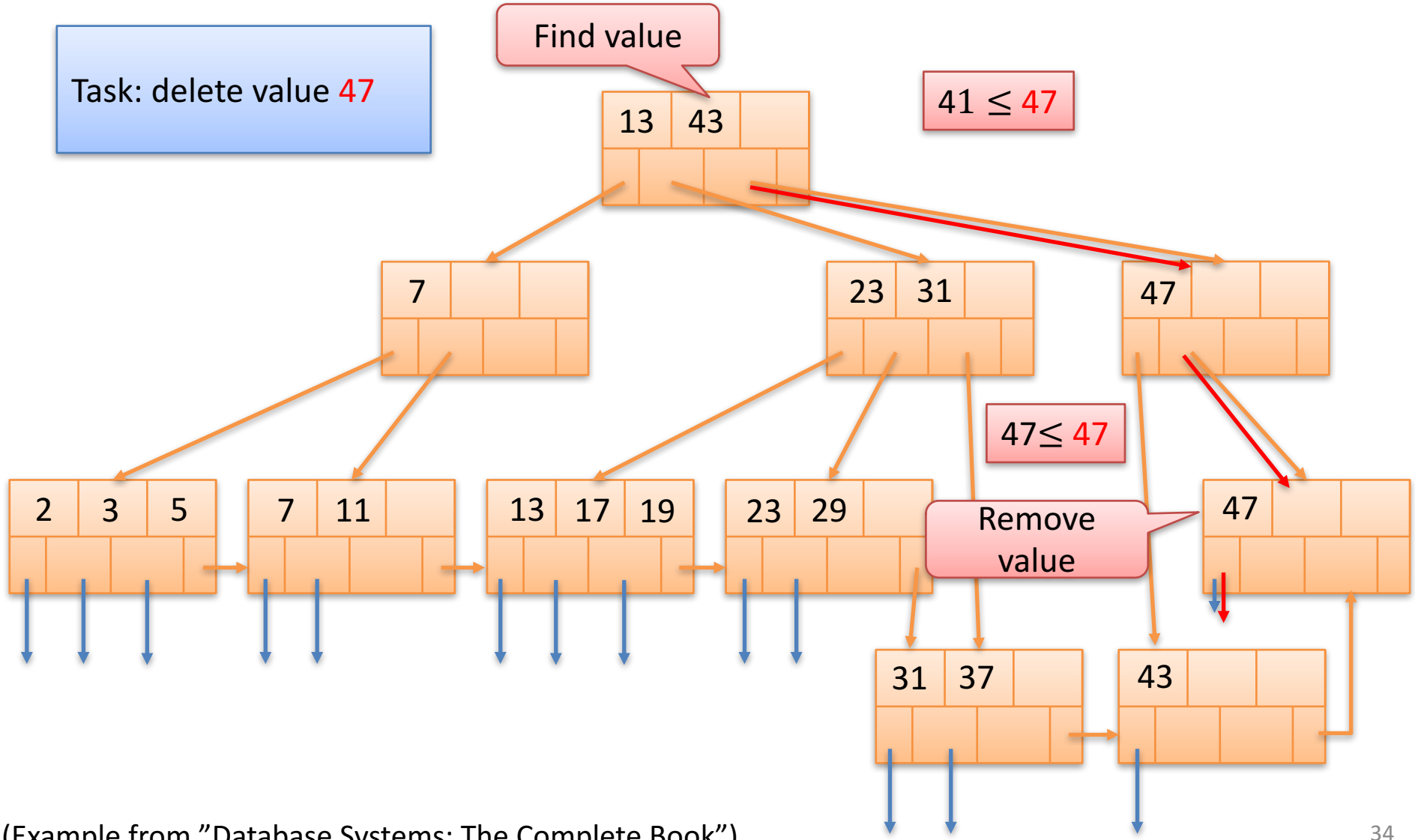
(Example from ”Database Systems: The Complete Book”)

Deletion part 2



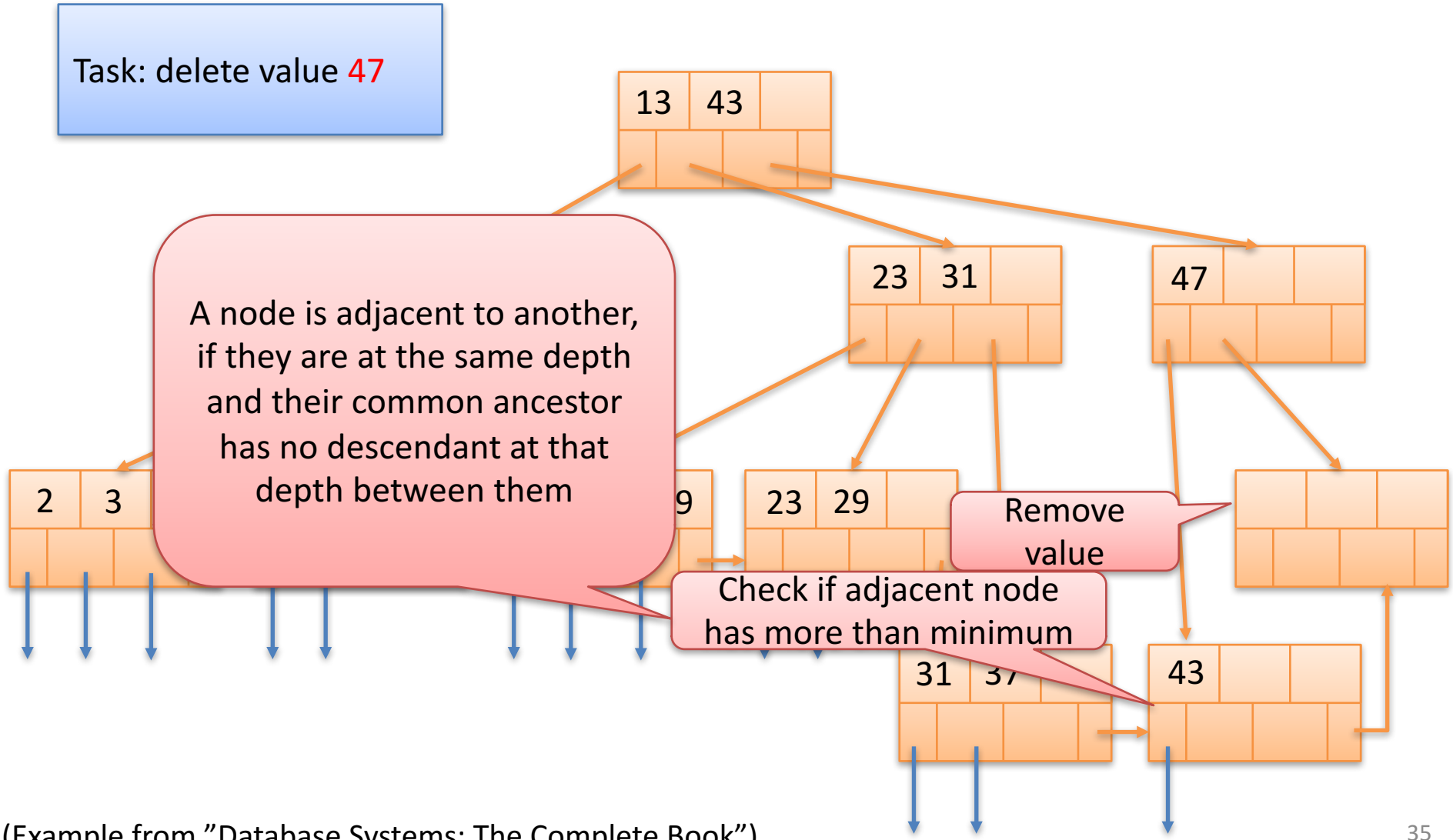
(Example from ”Database Systems: The Complete Book”)

Deletion part 3



(Example from ”Database Systems: The Complete Book”)

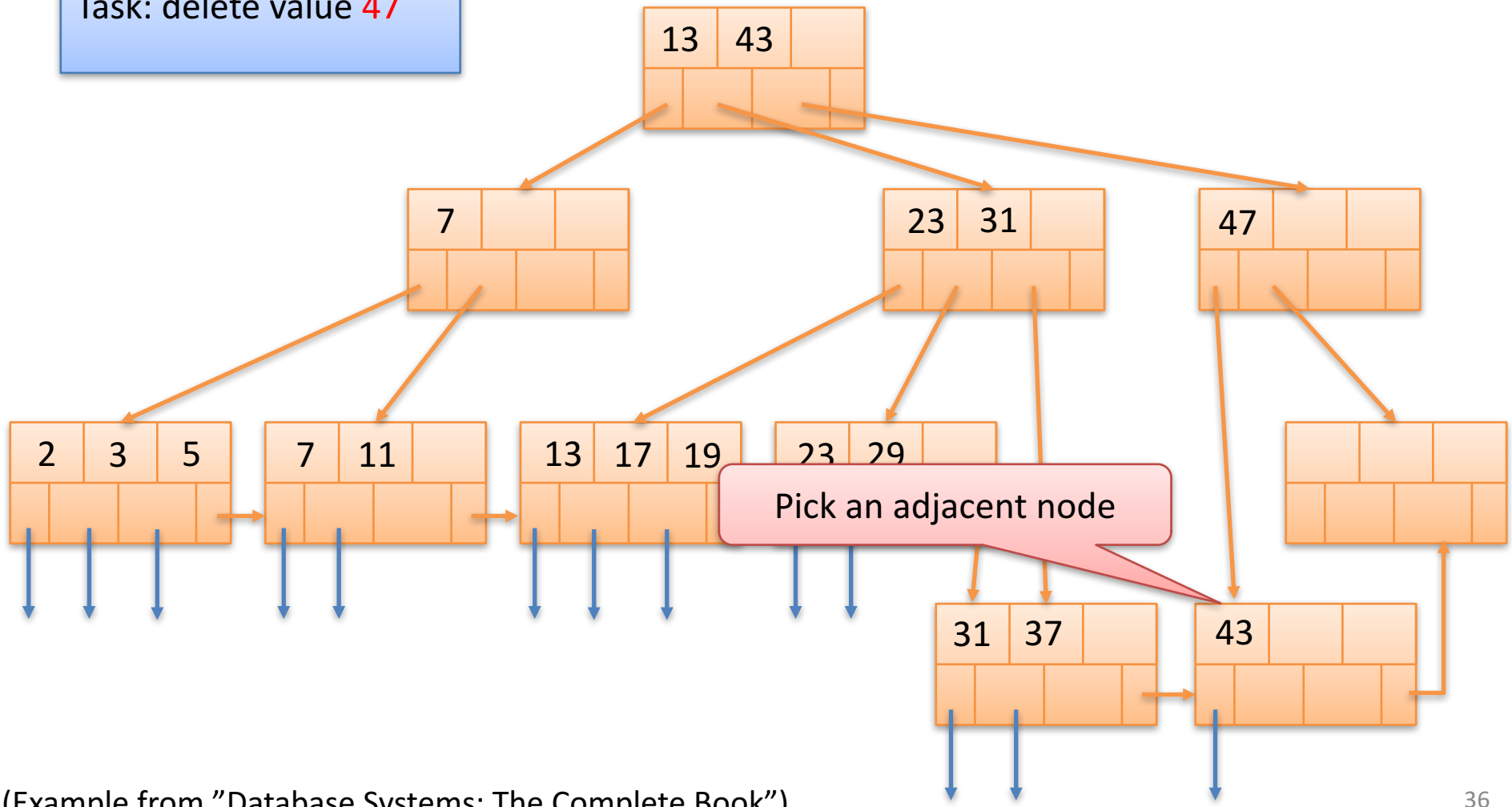
Deletion part 3



(Example from ”Database Systems: The Complete Book”)

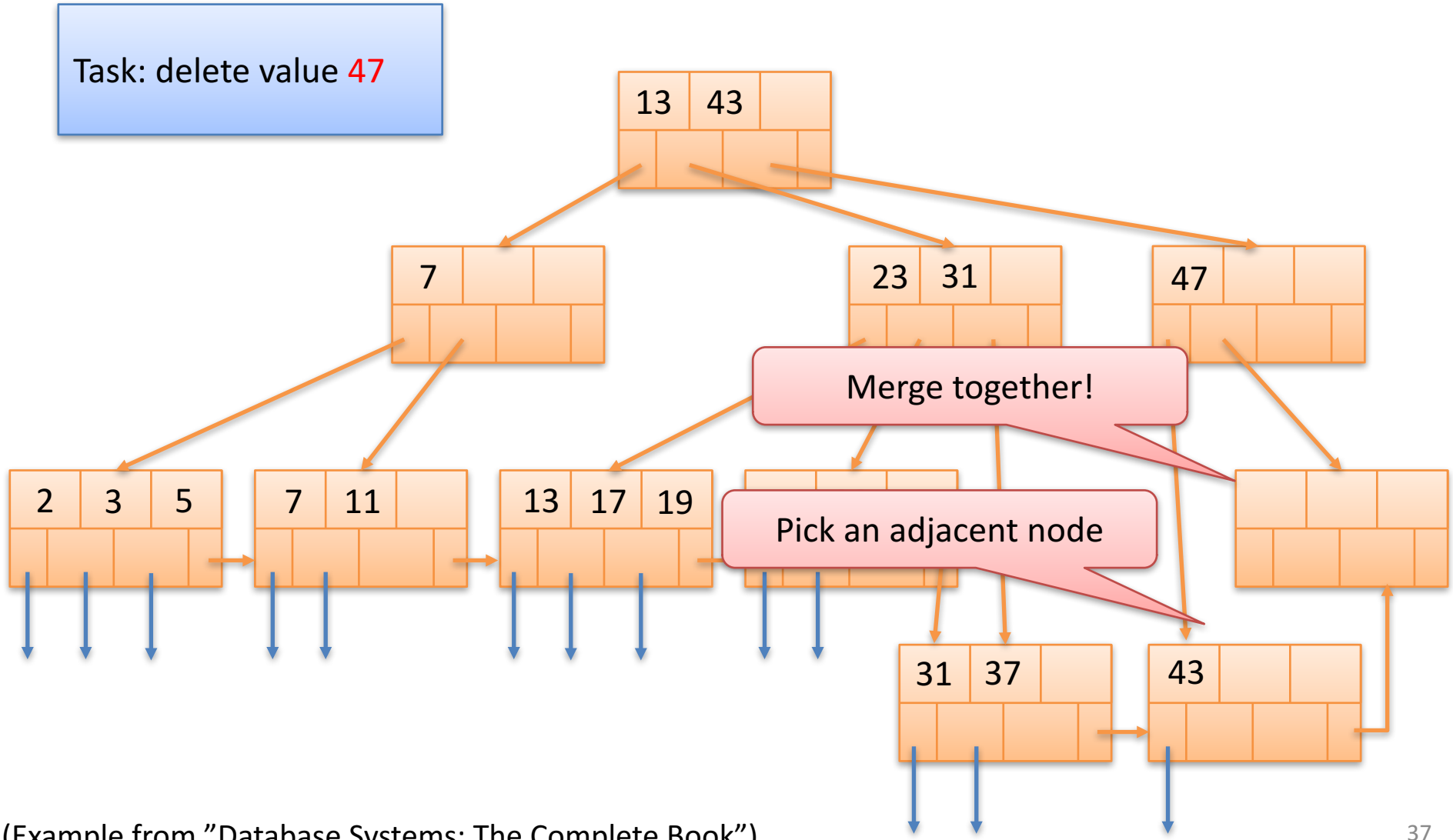
Deletion part 3

Task: delete value 47



(Example from ”Database Systems: The Complete Book”)

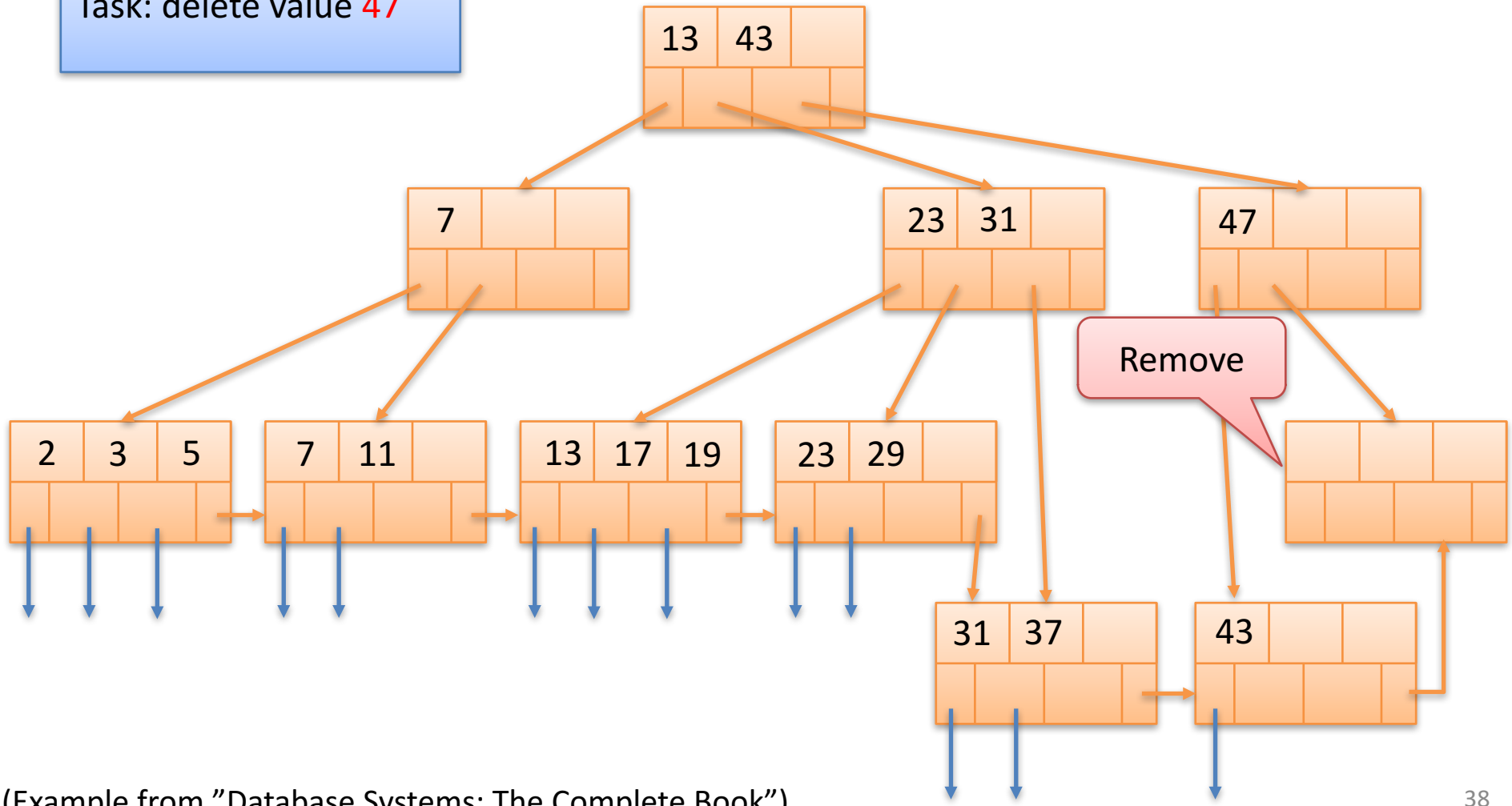
Deletion part 3



(Example from ”Database Systems: The Complete Book”)

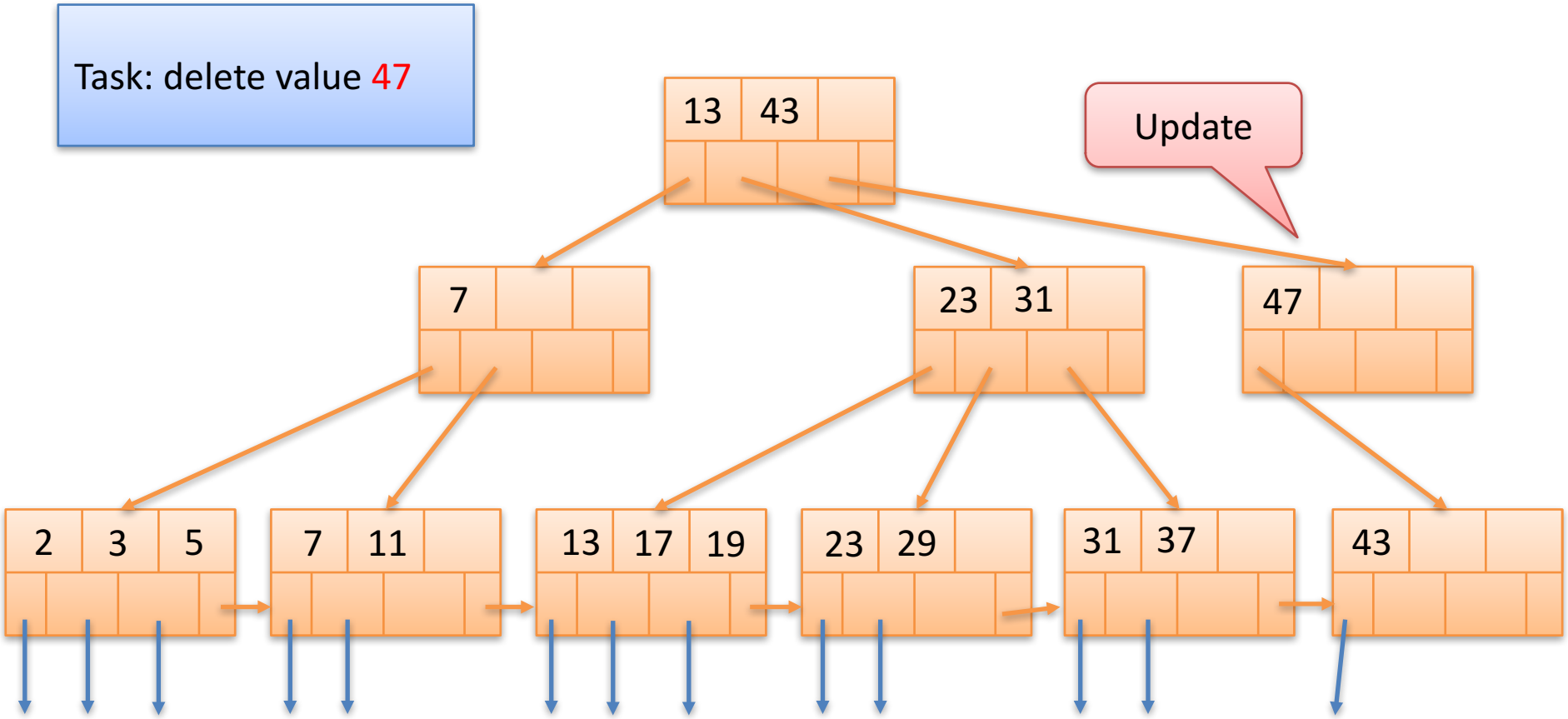
Deletion part 3

Task: delete value 47



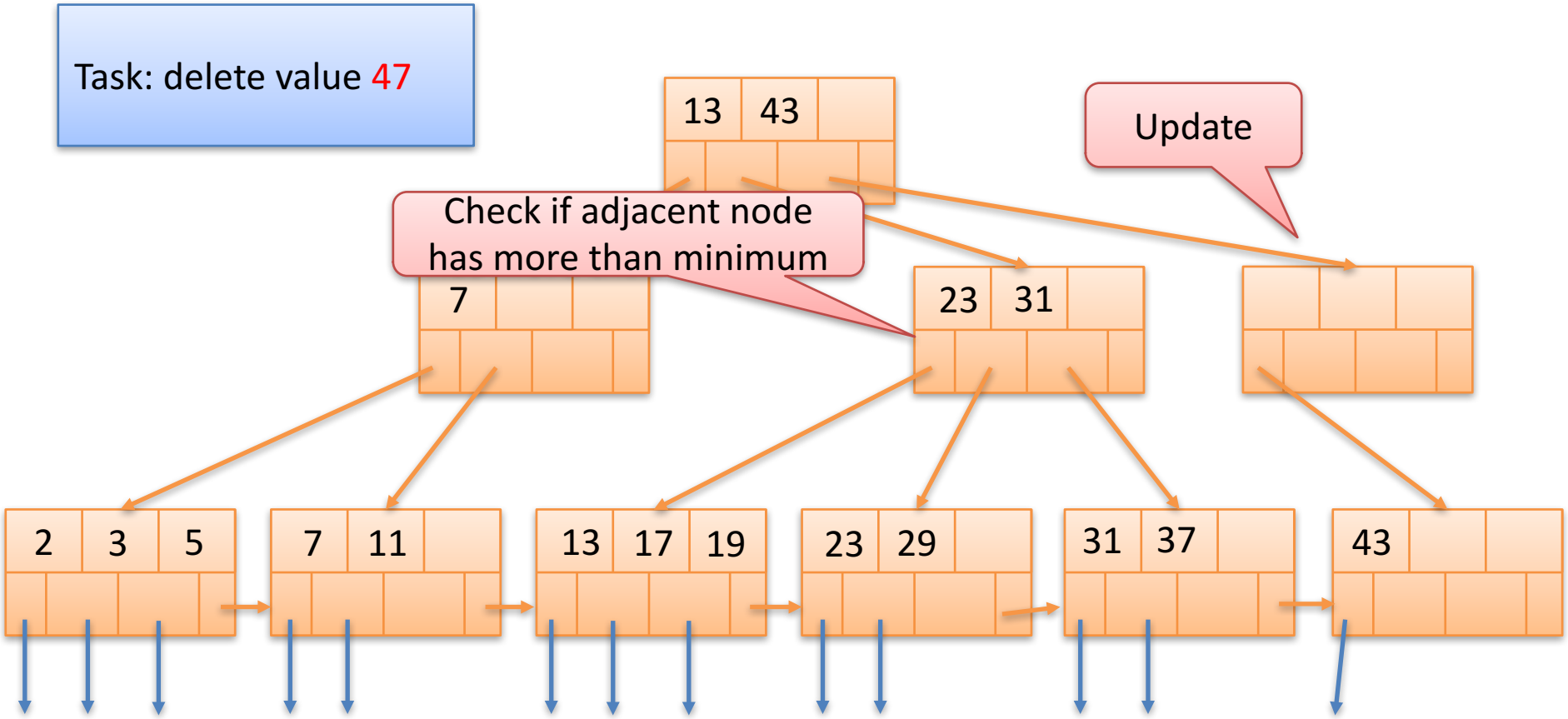
(Example from ”Database Systems: The Complete Book”)

Deletion part 3



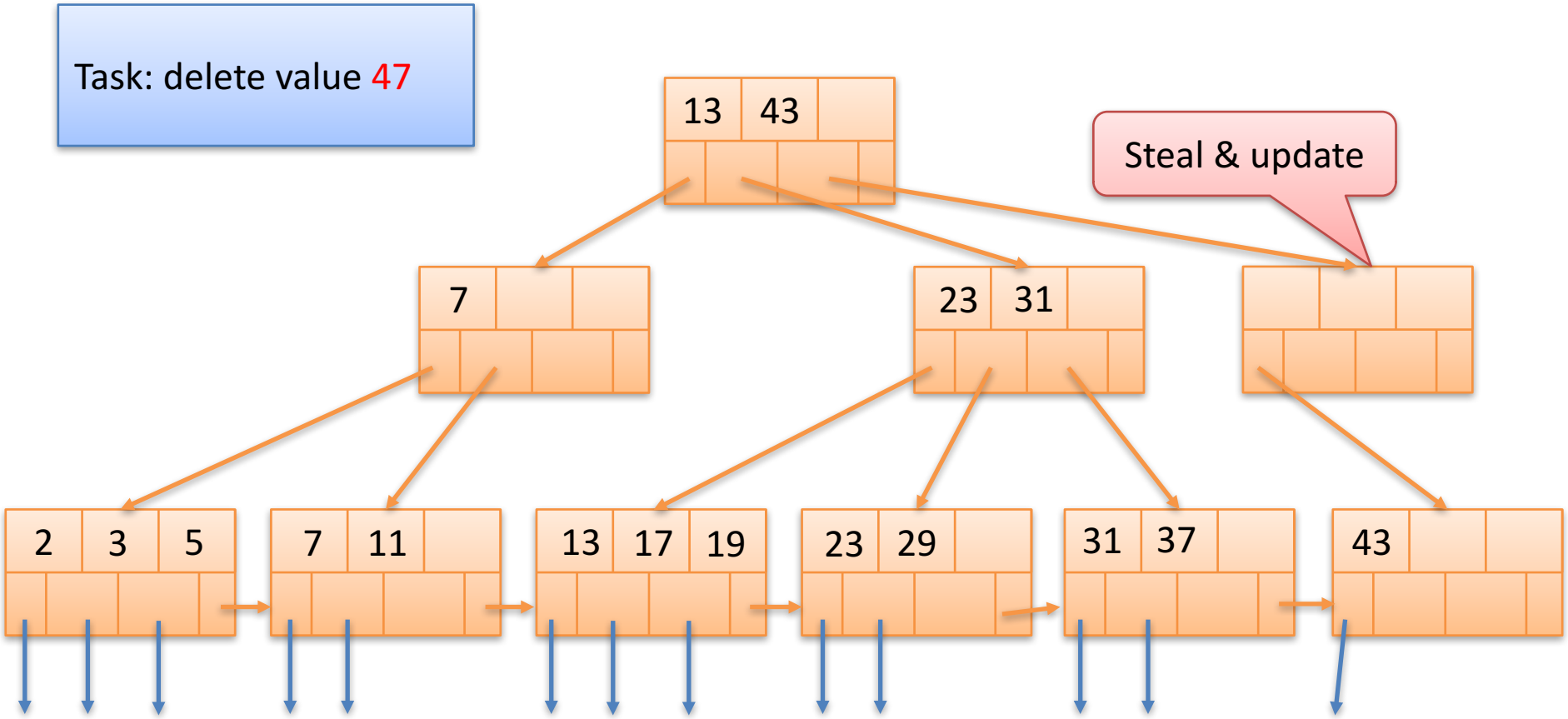
(Example from ”Database Systems: The Complete Book”)

Deletion part 3



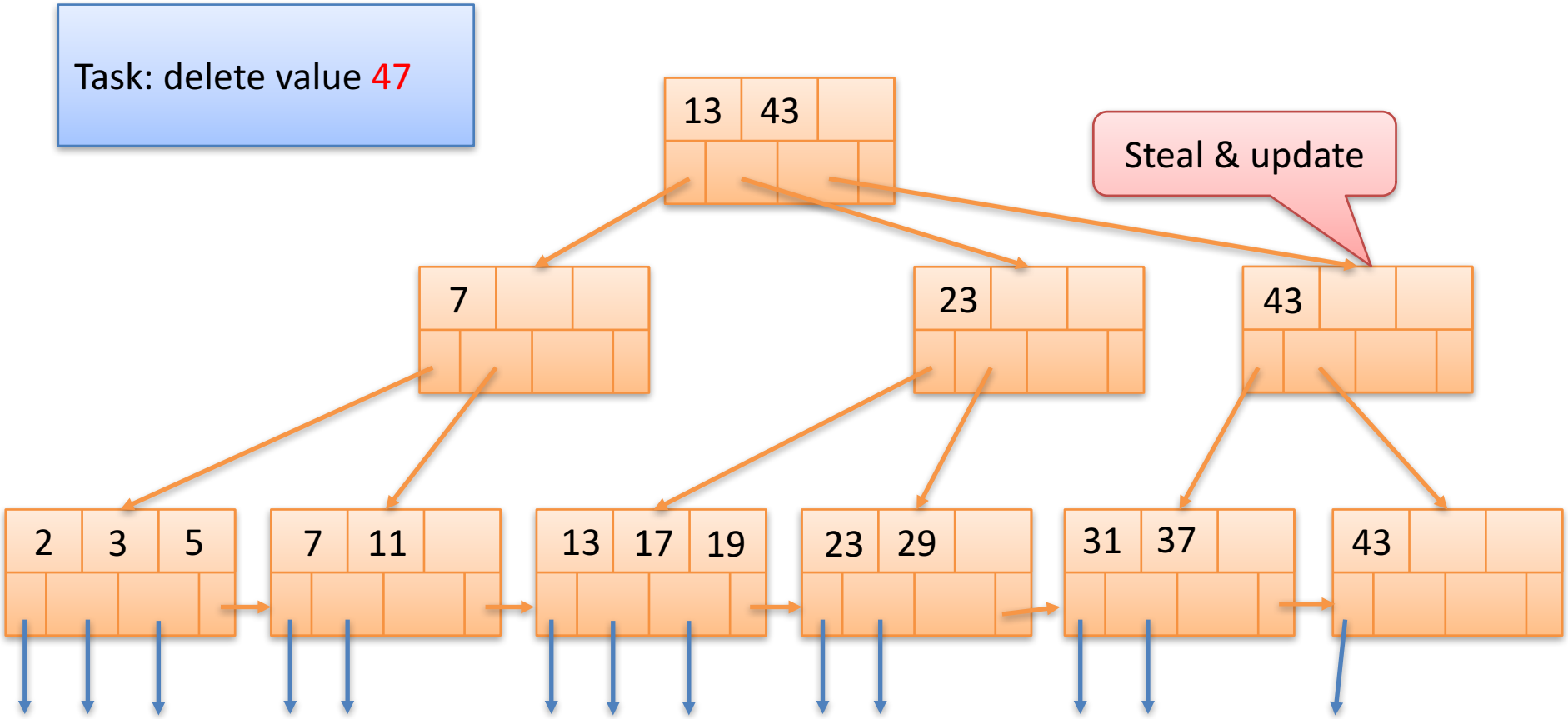
(Example from ”Database Systems: The Complete Book”)

Deletion part 3



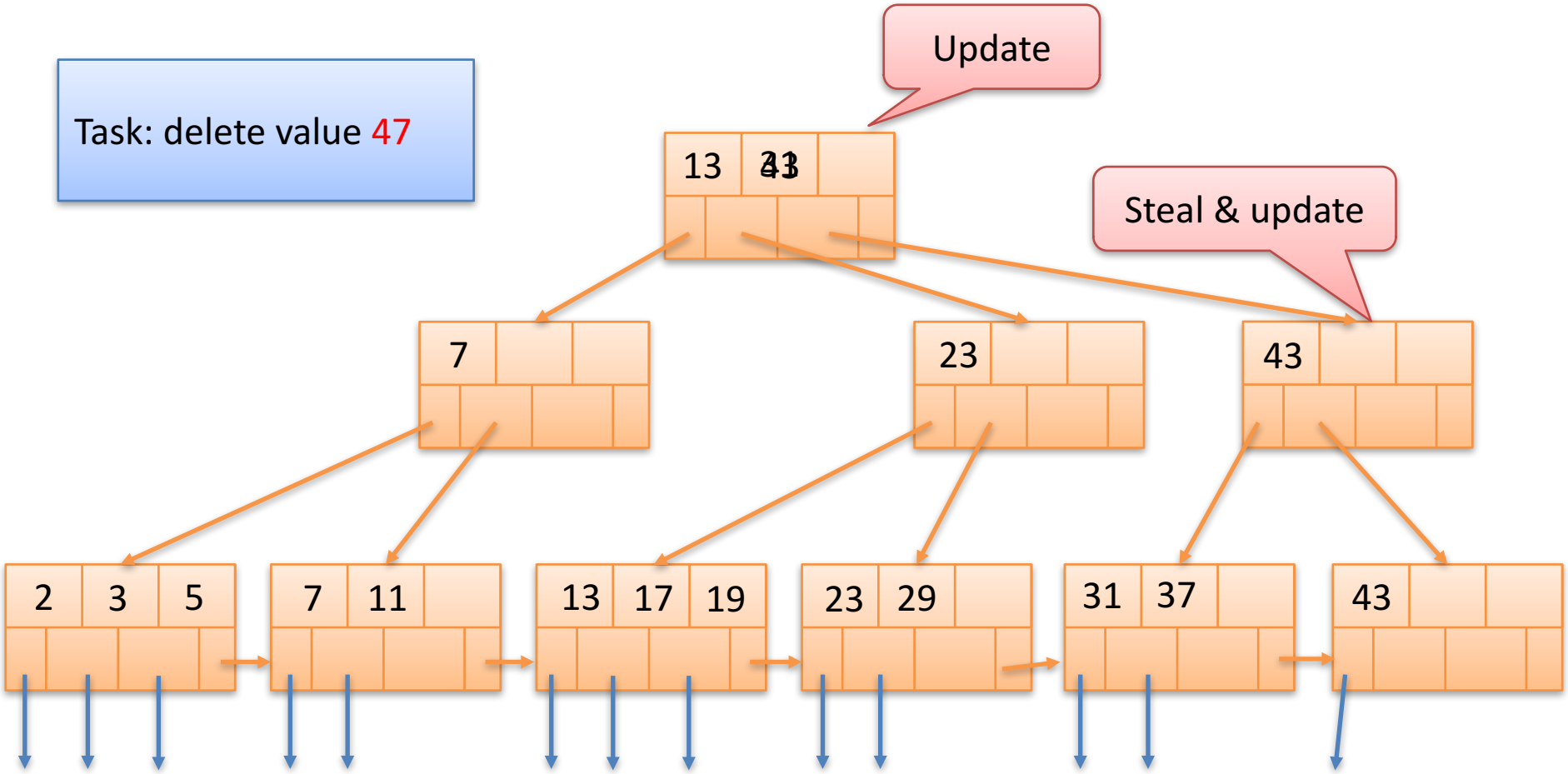
(Example from ”Database Systems: The Complete Book”)

Deletion part 3



(Example from ”Database Systems: The Complete Book”)

Deletion part 3



(Example from ”Database Systems: The Complete Book”)

Review: Deletion

- Goal: delete a value/pointer pair
- Procedure:
  - Find the leaf that should contain the value
    - If not there: Done
  - Remove the value/pointer pair
  - Let the current node C
  - Let x be $\begin{cases} 2 & if\ C\ is\ root \\ \left\lceil \frac{n+1}{2} \right\rceil & if\ C\ is\ internal\ node \\ \left\lfloor \frac{n+1}{2} \right\rfloor & if\ c\ is\ leaf \end{cases}$
  - If C has above x pointers: Fix ancestors (if necessary) and you are done
  - If C is the root but not a leaf: Remove it (and let the child of the root be the new root)
  - Otherwise, check if an adjacent node has at least $x + 1$ pointers
  - If so: take one, fix ancestors (if necessary) and you are done
  - Otherwise, merge with sibling and go to line 3 with the parent as current node
- The B+ tree **remains balanced**!
- Running time: O(h × $\log_2$ n)

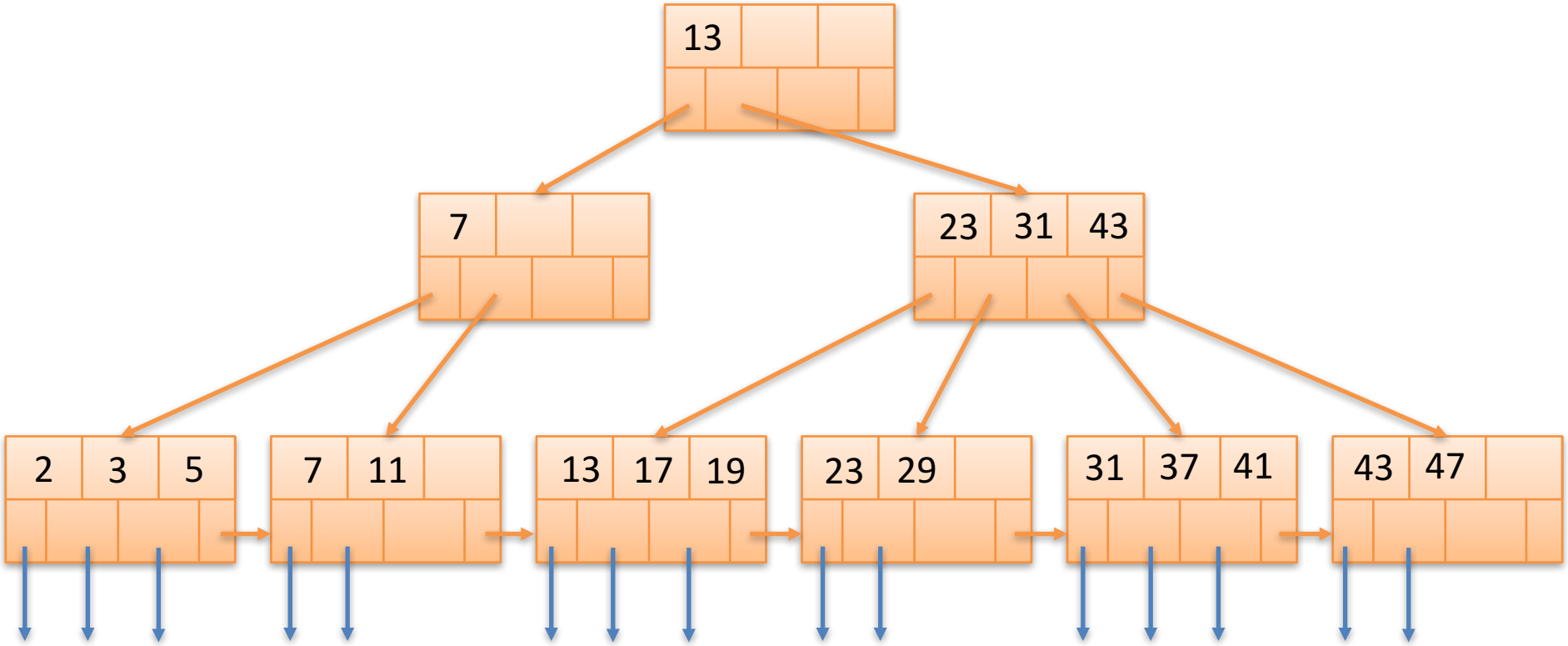
See definition slide 25

Time for a disk operation

"real" running time O(h × D)

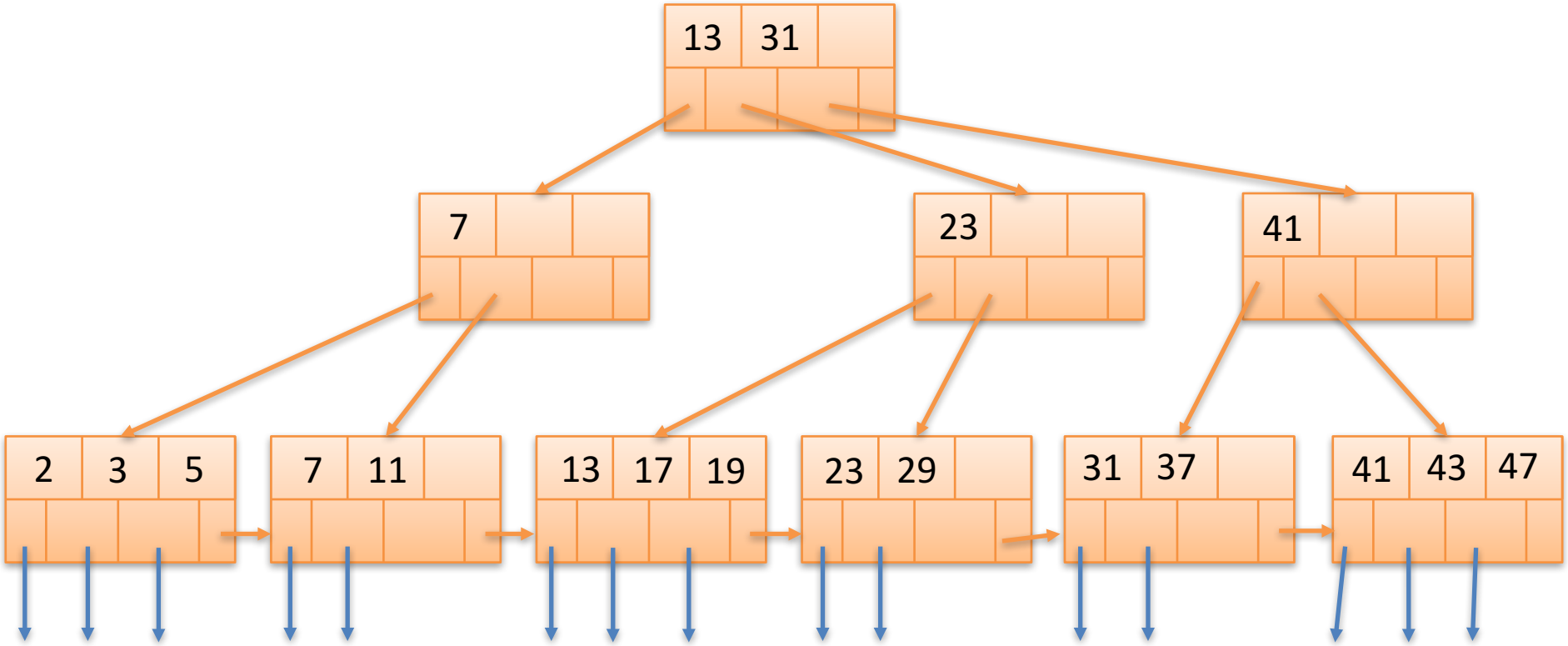
Height of the B+ tree

Before insertion + deletion of 42



(Example from ”Database Systems: The Complete Book”)

After insertion + deletion of 42



(Example from ”Database Systems: The Complete Book”)

Properties of B+ Tree Indexes

- **Fast lookups, insertions, deletions** in time:

$n \approx B$

$$O(\text{height of B+ tree} \times \log_2 n) = O(\log_n N \times \log_2 n)$$
$$= O(\log_2 N)$$

- Remain **balanced**
- **Huge capacity** even with height 3 if blocks large enough
  - Block size: 16386 bytes (16 kilobytes)
  - Values stored in index: 4 bytes
  - Pointers: 8 bytes
  - Largest n so that each B+ tree node fits into a block (i.e., $4n + 8(n+1) \leq 16386$) is 1364
  - B+ trees with height 3 can store $> n^3 = 2537716544$ values

Properties of B+ Tree Indexes

- Can be implemented **efficiently with respect to number of disk accesses**
  - Number of disk accesses typically ≈ 2 + height of B+ tree
- Most of the B+ tree can be kept in memory
  - Upper levels
  - Even with block size of 16384 bytes and n = 1364:
    - Level 1 (root) ≈ 16 KB
    - Level 2 (children of root) ≈ (n+1) × 16 KB ≈ 21 MB
    - Level 3 ≈ (n+1) × (n+1) × 16 KB ≈ 28 GB

Typically, these are the leaf nodes and can be loaded from disk on demand

Summary

- Indexes allow DBMS to **quickly find and navigate** to desired tuples of a relation
- **Can speed up computation** of selections & joins
- Many **different forms** of indexes
  - B+ trees
  - Hash tables (not covered here)
  - …
- Trade off:
  - More indexes help answer more queries faster
  - But more indexes make updates slower