# Discussion #5

Diwanshu, Elena

July 8, 2020

# Logistics

- HW3 due 7/12, 11:59 PM.
- Checkpoint 3 due 7/15, 11:59 PM.

# Outline

- B+ Tree → Insertion, Deletion.
- Buffer Manager → LRU, MRU

# Why B+ Tree?

→ Most queries can be executed more quickly if the values are stored in order.

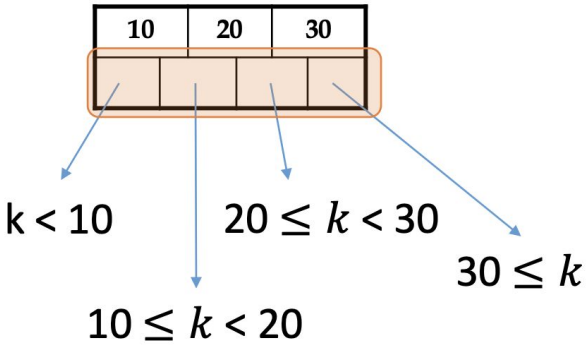
→ But it's not practical to hope to store all the rows in the table one after another, in sorted order, because this requires rewriting the entire table with each insertion or deletion of a row.

# Requirements of B+ Tree

- Two types of nodes: index (internal) nodes and data (leaf) nodes. Each node is one disk page.
- Each non-leaf (“internal”) node has **d ≤ m ≤ 2d** entries
  - Minimum **50%** occupancy
- Root node has **1 ≤ m ≤ 2d** entries
- We call **d** the **order** of the tree

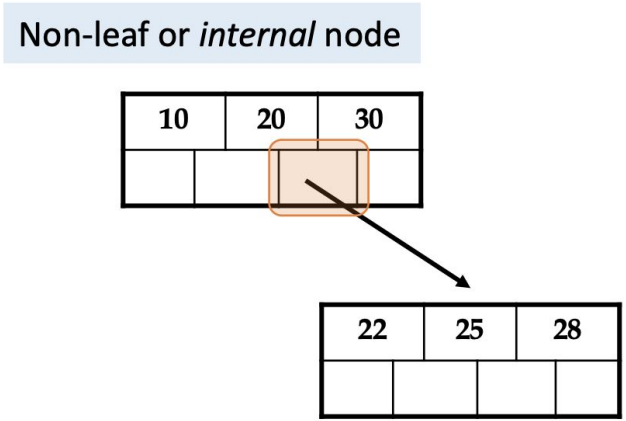
# B+ Tree

- The **n** entries in a node define **n+1** ranges.

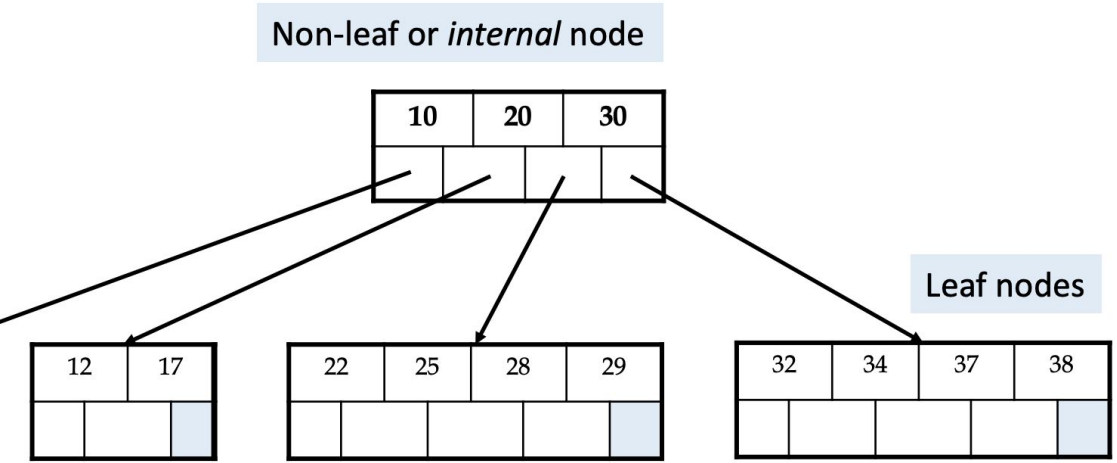


# B+ Tree

- For  each range, in a non-leaf node, there is a pointer to another node with entries in that range.

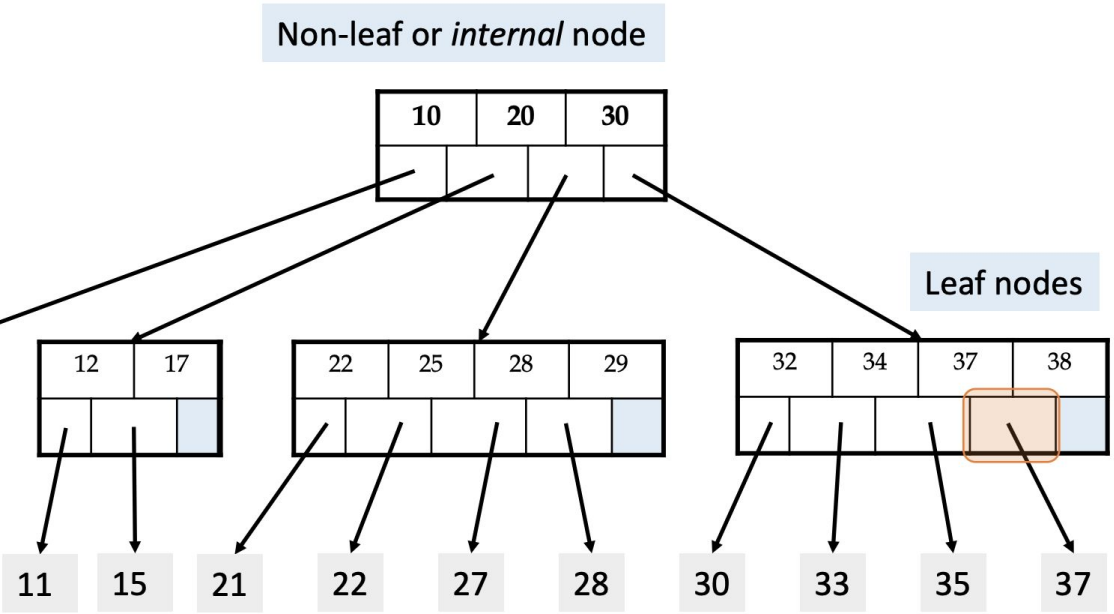


# B+ Tree




- Leaf nodes also have between **d and 2d** entries, and are different in that:

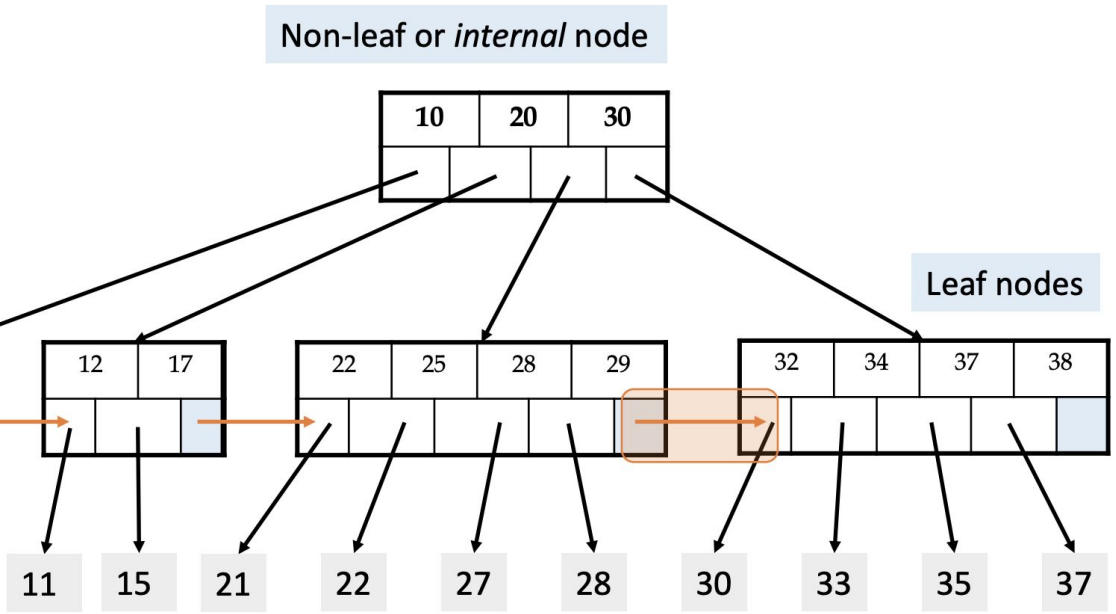
# B+ Tree




- Leaf nodes also have between **d and 2d** entries, and are different in that:

1. Their entry slots contain pointers to data records.

# B+ Tree



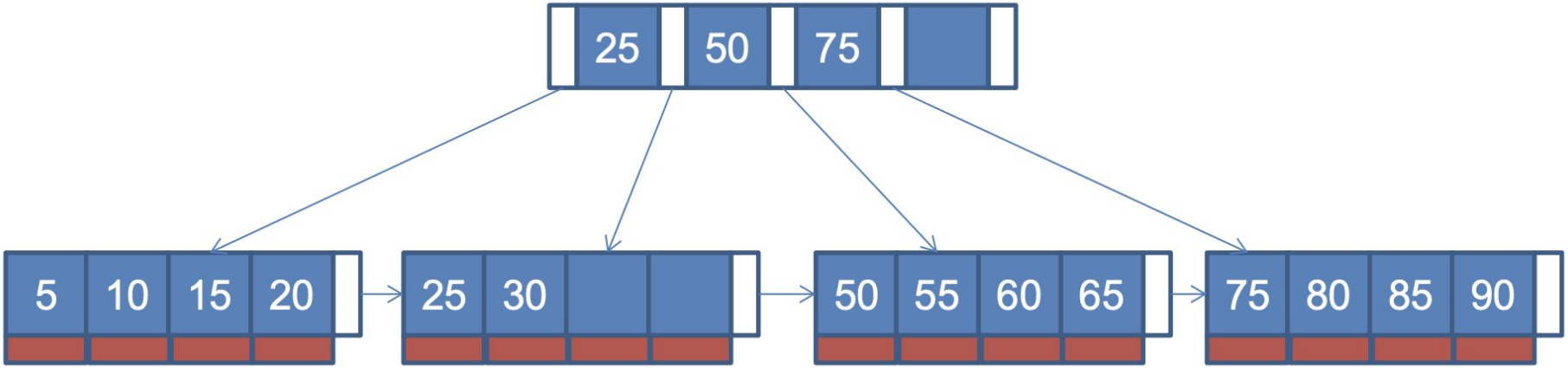

- Leaf nodes also have between **d and 2d** entries, and are different in that:

1. Their entry slots contain pointers to data records.
2. They contain a pointer to the next leaf node as well, for faster sequential traversal.

# Inserting

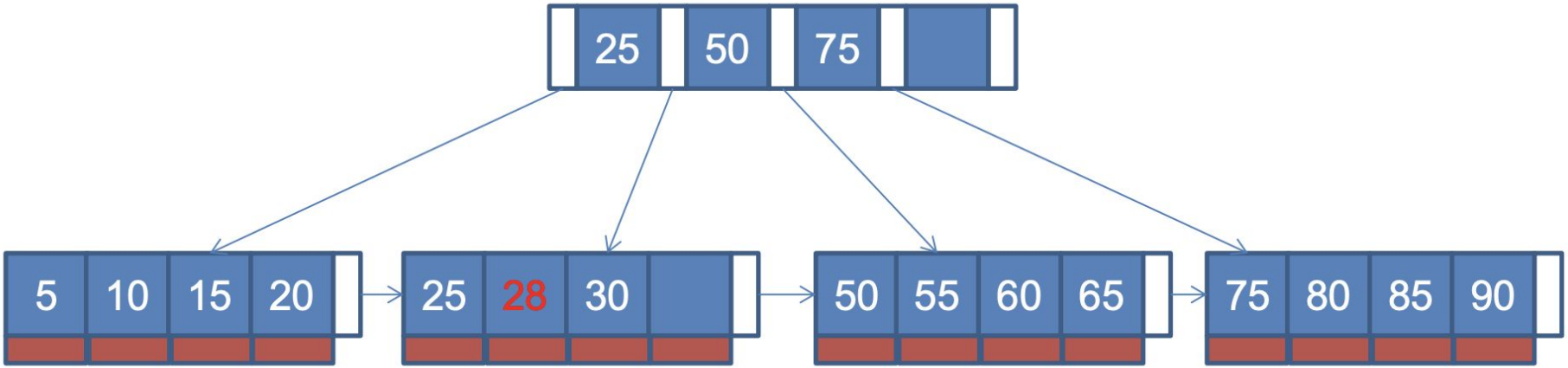
- Find leaf where new key **x** belongs
- If leaf is not full then add **x** to leaf
- If leaf is full then split leaf and push splitting key up to parent
  - If parent is full then split parent and push splitting key up
  - If we need to add a key to the root and the root is full then split the root and create a new parent as root

# Inserting



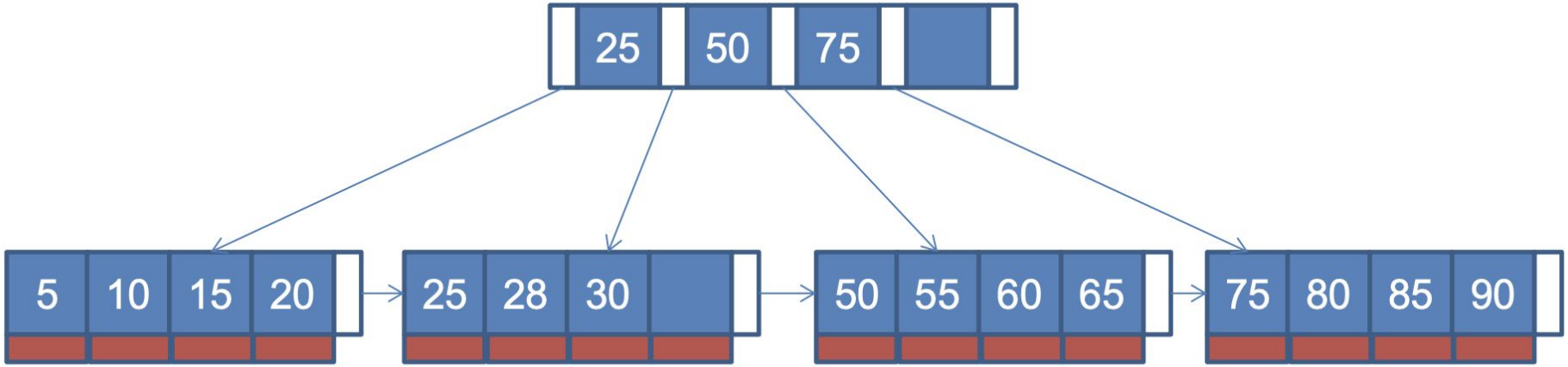

- Insert 28
  - Leaf is not full so we can just add it

# Inserting



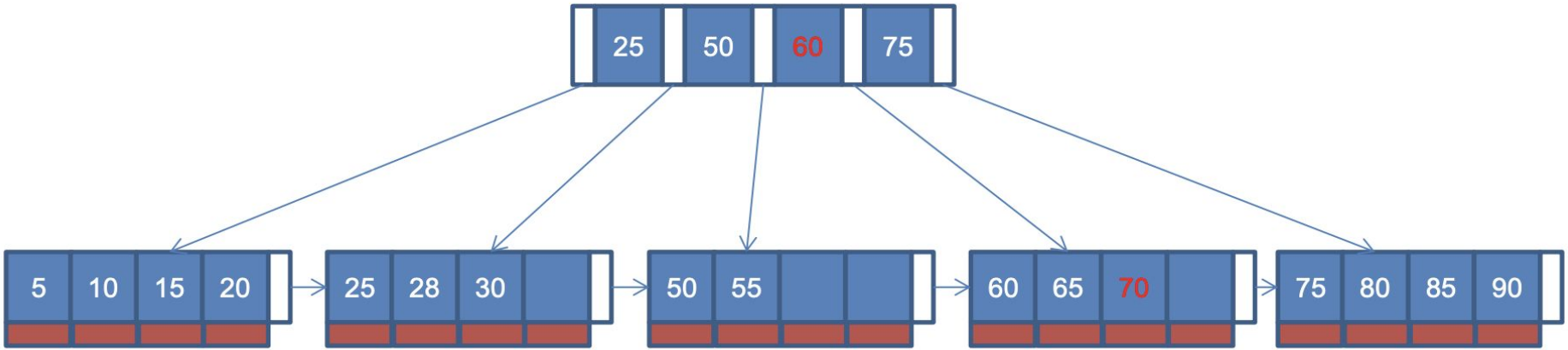

- Insert 28
  - Leaf is not full so we can just add it

# Inserting



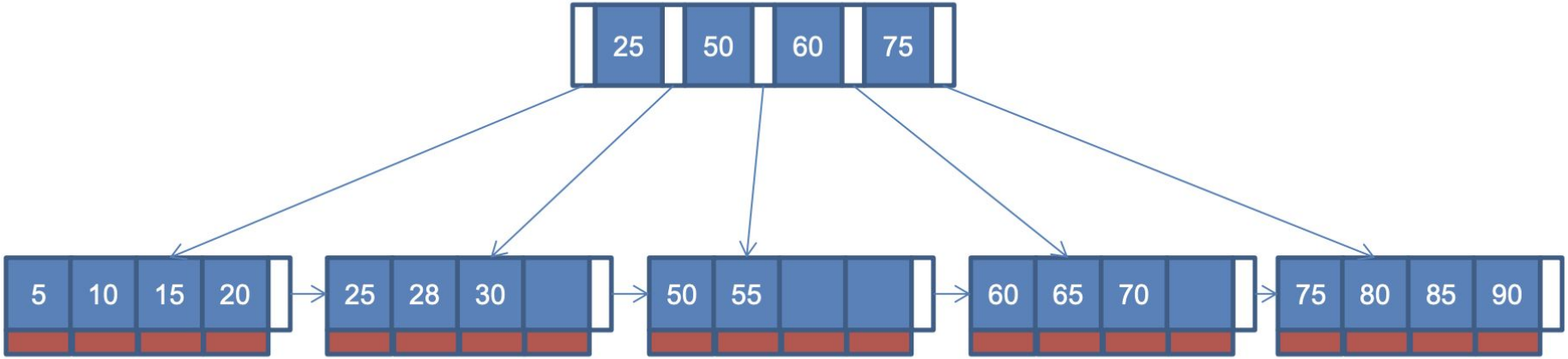

- Insert 70
  - Should go in leaf with (50, 55, 60, 65), but leaf is full → Need to split the leaf and then insert.

# Inserting



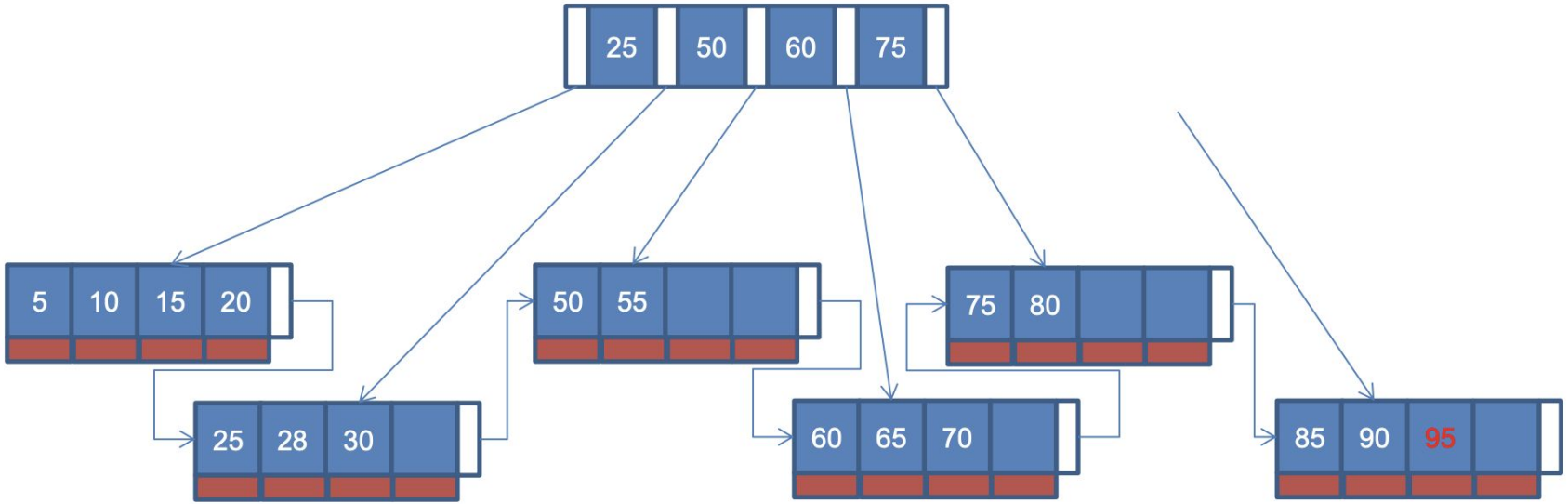

- Insert 70
  - Should go in leaf with (50, 55, 60, 65), but leaf is full → Need to split the leaf and then insert → **60 is pushed up to the parent**

# Inserting



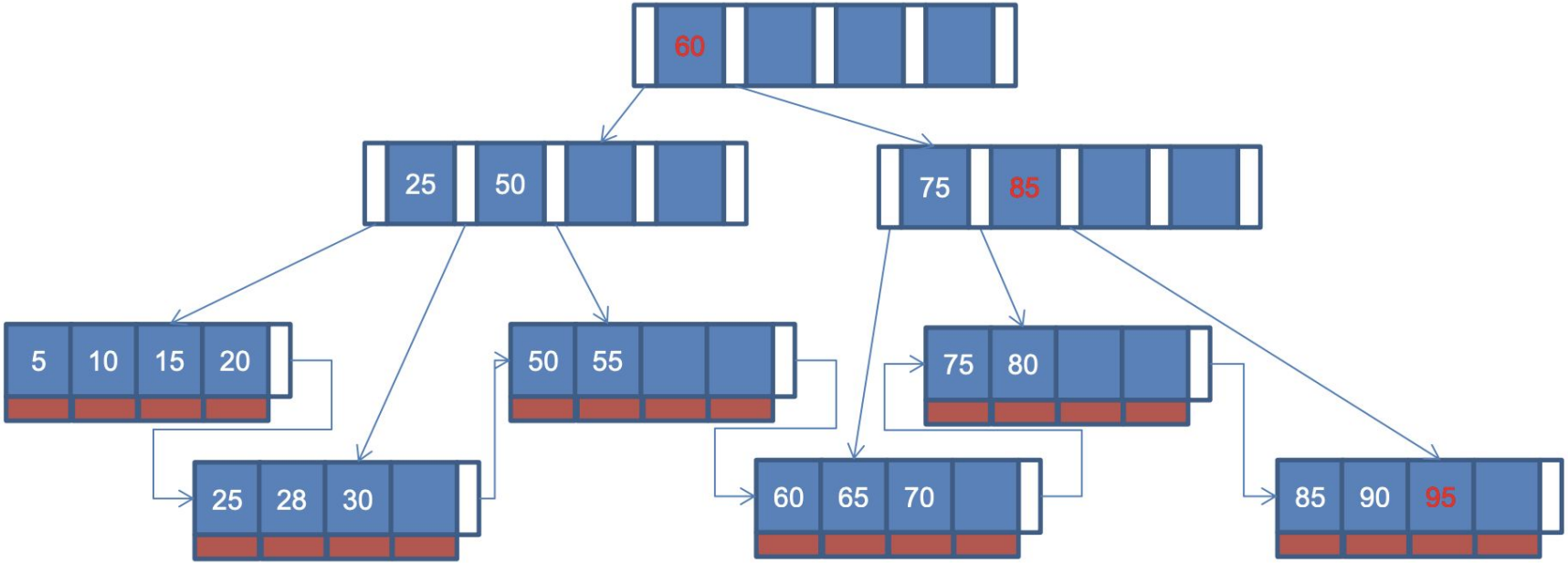

- Insert 95
  - Should go in leaf with (75, 80, 85, 90), but leaf is full → Need to split the leaf and then insert
  - 85 is pushed up to the parent, but parent is full! → Need to split parent node then insert 85

# Inserting



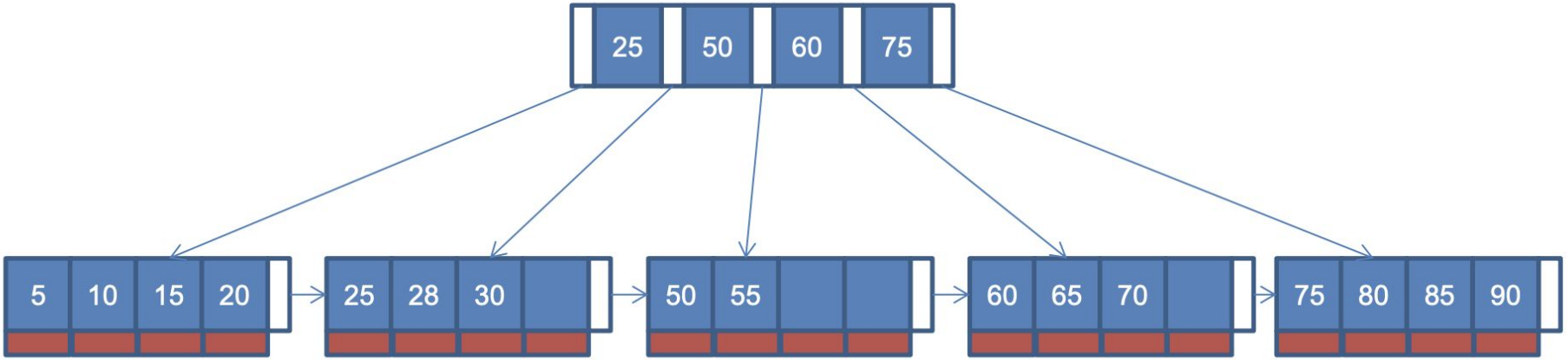

- Insert 95
  - Should go in leaf with (75, 80, 85, 90), but leaf is full → Need to split the leaf and then insert
  - 85 is pushed up to the parent, but parent is full! → Need to split parent node then insert 85

# Inserting



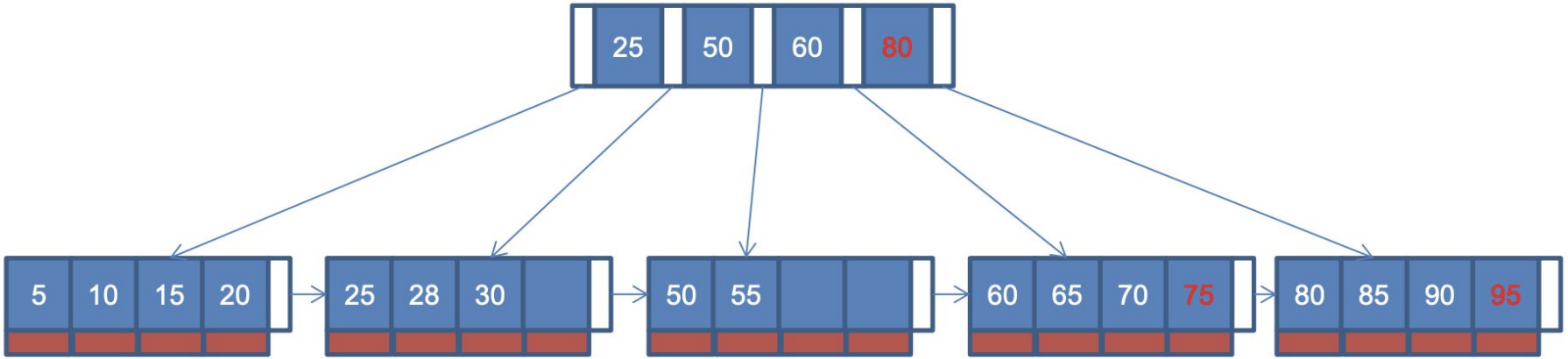

- Insert 95
  - Should go in leaf with (75, 80, 85, 90), but leaf is full → Need to split the leaf and then insert
  - 85 is pushed up to the parent, but parent is full! → Need to split parent node then insert 85
  - **60 is then pushed up to the new root**

# Inserting Alternative




- If inserting into a full node with a non-full sibling, we could shift keys from the full node to the non-full sibling
  - Requires modifying parent keys

# Inserting Alternative




- Insert 95
  - Should go in leaf with (75, 80, 85, 90), but leaf is full → Shift 75 to left sibling
  - Modify key in parent node → Insert 95 in newly opened node

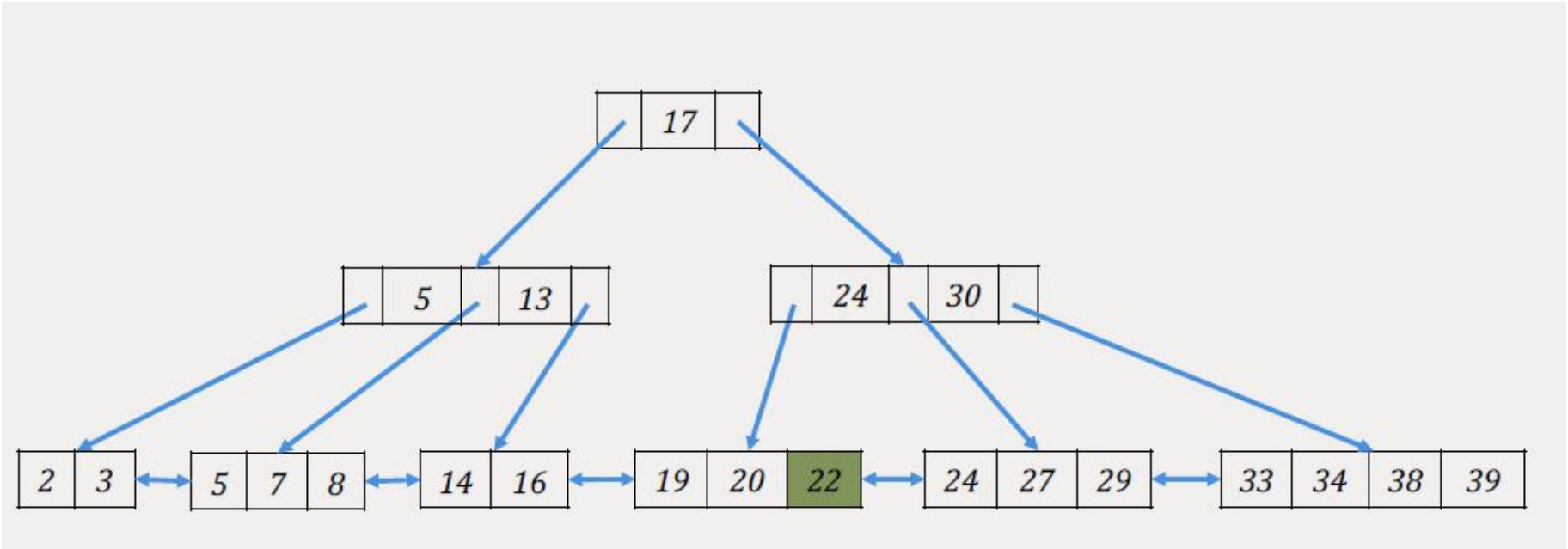
# Deletion

- Find leaf node **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
    - If re-distribution fails, **merge L** and sibling
- If a merge occurred, we must delete an entry from the parent of **L**

# Example 1

order **d = 2**

delete 22

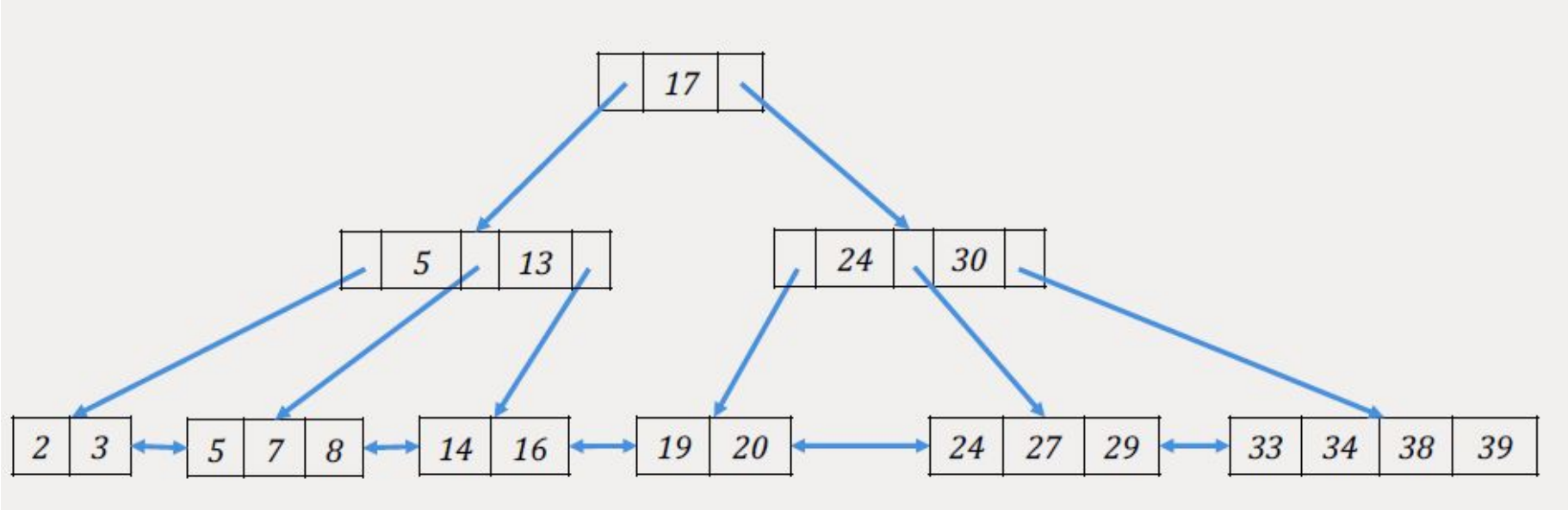



since by deleting 22 the node remains half-full, we simply remove it

# Example 1

order **d = 2**

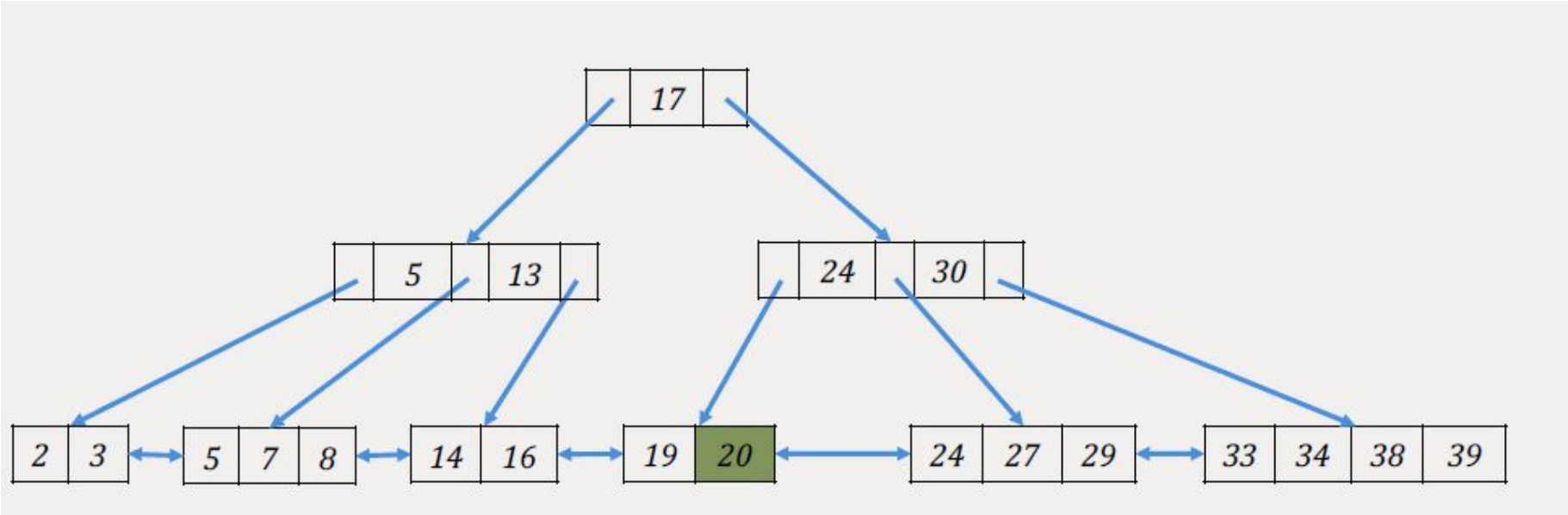
delete 22



# Example 2

order **d = 2**

delete 20

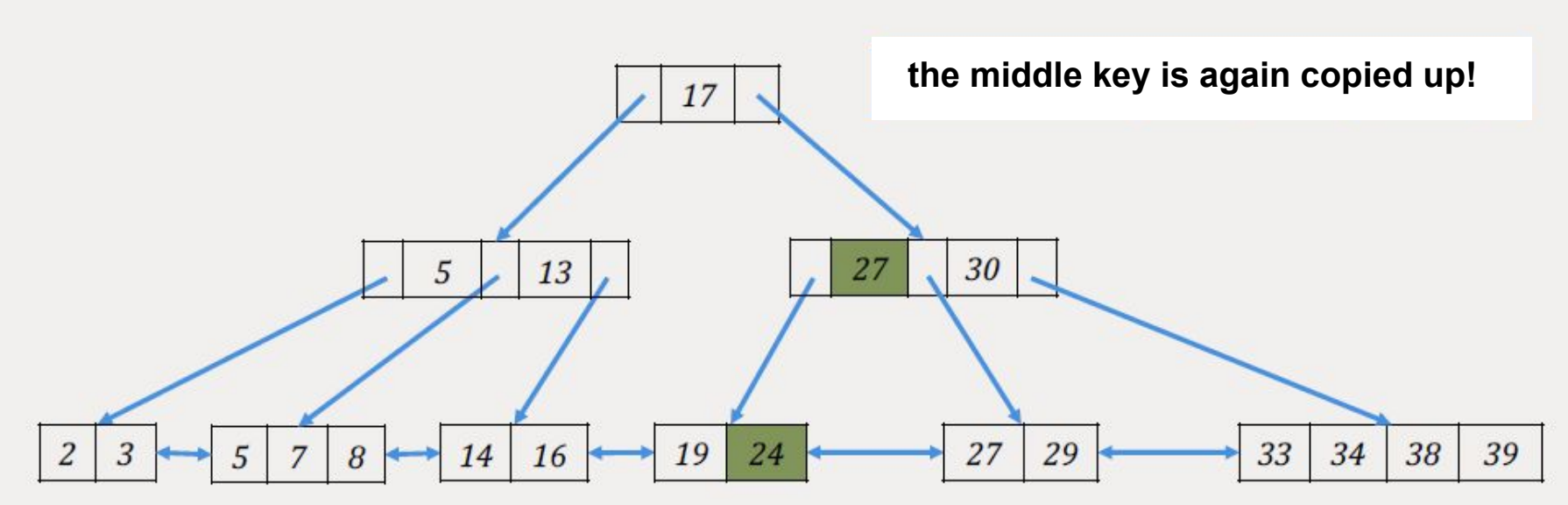



by removing 20 the node is not half-full anymore, so we attempt to redistribute!

# Example 2

order **d = 2**

delete 20

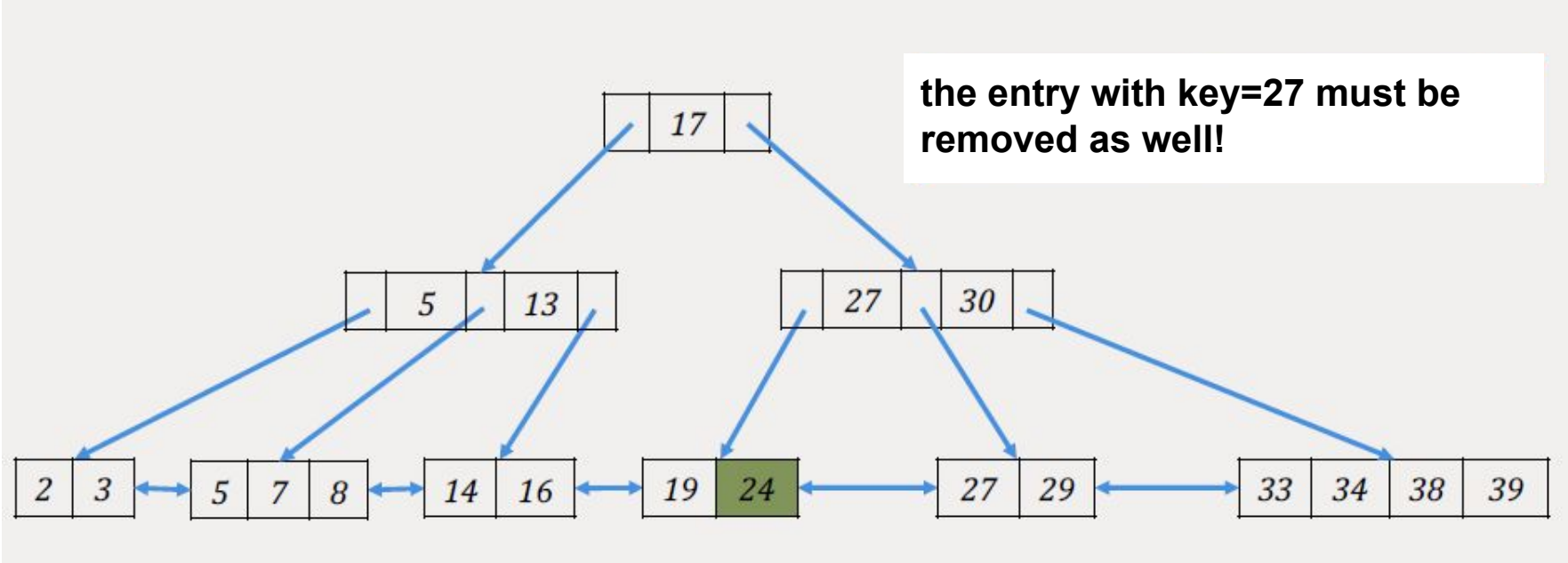



by removing 20 the node is not half-full anymore, so we attempt to redistribute!

# Example 3

order **d = 2**

delete 24



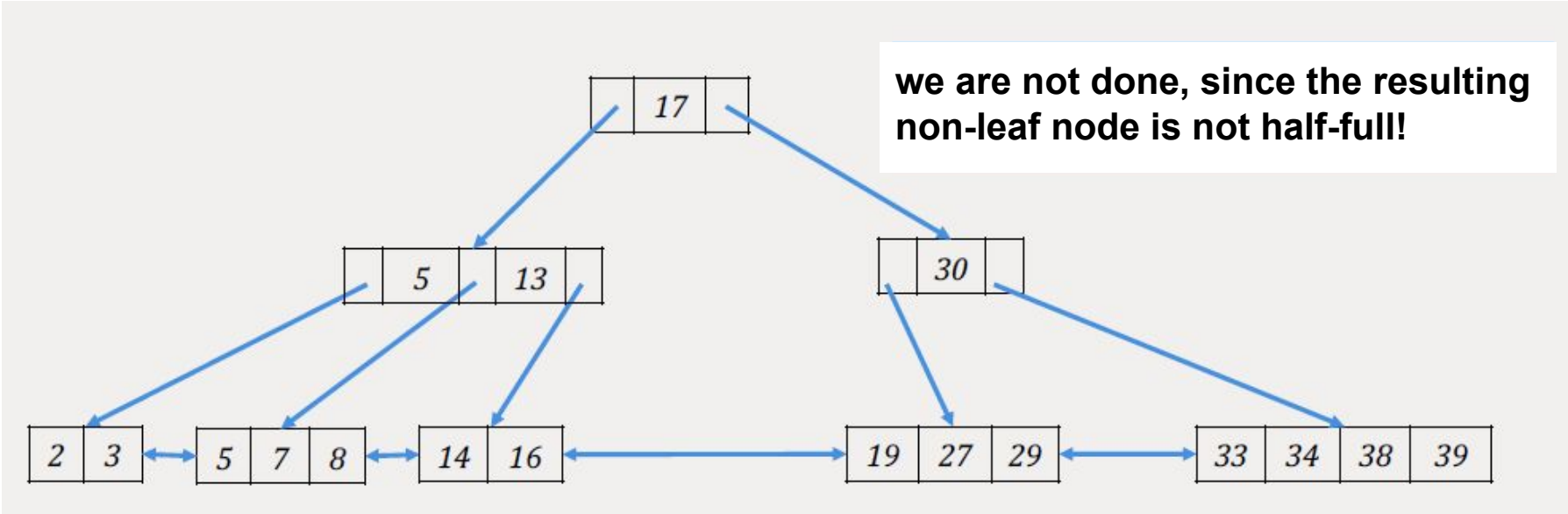

**the entry with key=27 must be removed as well!**

in this case we have to merge nodes!

# Example 3

order **d = 2**

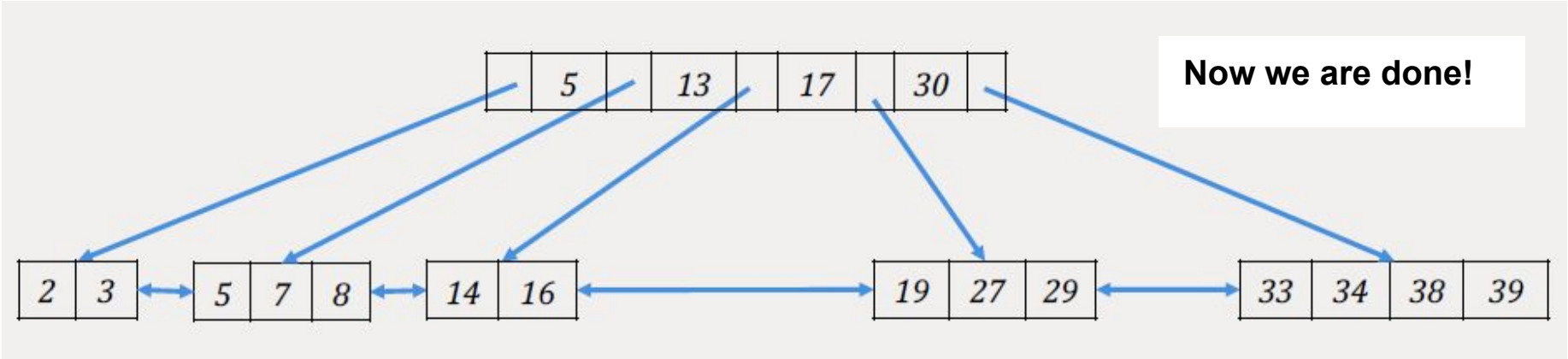
delete 24



# Example 3

order **d = 2**

delete 24




**Now we are done!**

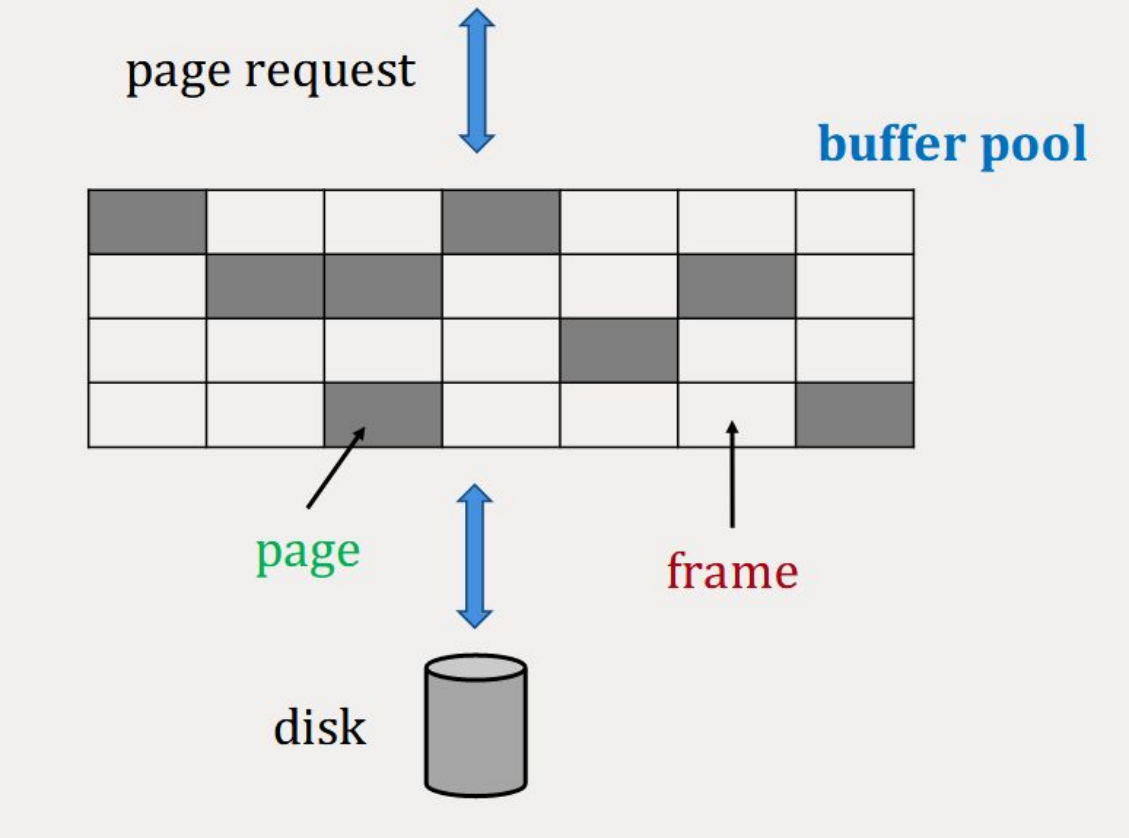
# Buffer Manager

- Data must be in RAM for the DBMS to operate on it
- All the pages may not fit into the main memory

**Buffer Manager:** responsible for bringing pages from disk to main memory as needed

- pages brought into memory are in the **buffer pool**
- the buffer pool is partitioned into **frames**: slots for holding disk pages
- page same size with frame

# Buffer Replacement Policy




How to choose a frame for replacement?

- LRU (Least Recently Used)
- MRU (Most Recently Used)
- Clock
- FIFO
- Random
- ...

# LRU

- Uses a queue of pointers to frames that have **pin_count = 0**
- A page request uses frames only from the head of the queue
- When a the **pin_count** of a frame goes to 0, it is added to the end of the queue
- MRU works in a similar way

# LRU - Least Recently Used


Page read sequence - A, B, C, A, B, C, A, B, C *(Sequential Flooding)*

**Q. Miss rate?**

| Time | Frame 1 | Frame 2 |
|---|---|---|
| T0(A) | A | |
| T1(B) | A | B |
| T2(C) | C | B |
| T3(A) | C | A |
| T4(B) | B | A |
| T5(C) | B | C |
| T6(A) | A | C |
| T7(B) | A | B |
| T8(C) | C | B |

# LRU - Least Recently Used


Page read sequence - A, A, B, B, C, C, A, B, C

**Q. Miss rate?**

| Time | Frame 1 | Frame 2 |
|---|---|---|
| T0(A) | A | |
| T1(A) | A | |
| T2(B) | A | B |
| T3(B) | A | B |
| T4(C) | C | B |
| T5(C) | C | B |
| T6(A) | C | A |
| T7(B) | B | A |
| T8(C) | B | C |

# MRU - Most Recently Used


Page read sequence - A, B, C, A, B, C, A, B, C *(Sequential Flooding)*

**Q. Miss rate?**

| Time | Frame 1 | Frame 2 |
|---|---|---|
| T0(A) | A | |
| T1(B) | A | B |
| T2(C) | A | C |
| T3(A) | A | C |
| T4(B) | B | C |
| T5(C) | B | C |
| T6(A) | B | A |
| T7(B) | B | A |
| T8(C) | C | A |

# MRU - Most Recently Used


Page read sequence - A, A, B, B, C, C, A, B, C

**Q. Miss rate?**

| Time | Frame 1 | Frame 2 |
|---|---|---|
| T0(A) | A | |
| T1(A) | A | |
| T2(B) | A | B |
| T3(B) | A | B |
| T4(C) | A | C |
| T5(C) | A | C |
| T6(A) | A | C |
| T7(B) | B | C |
| T8(C) | B | C |

Thanks!