B+ Tree Recap

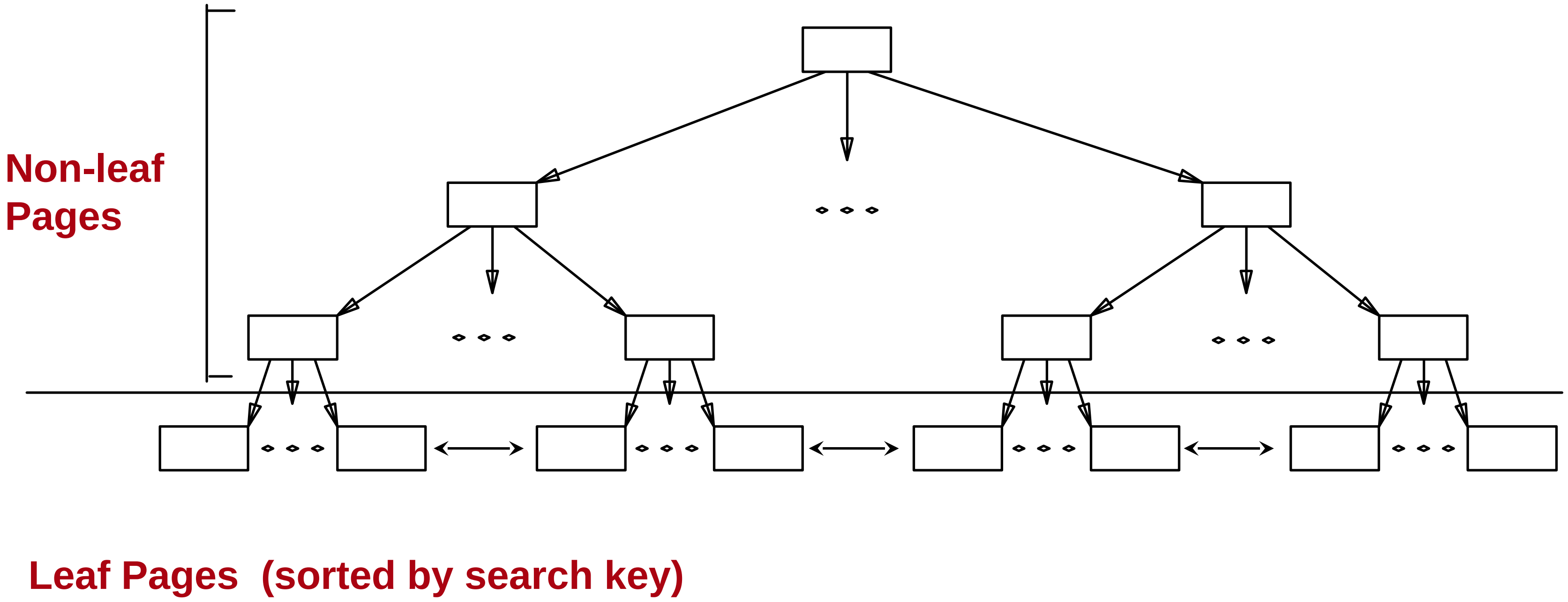
B Trees

- multiple keys per node, ensure each node (except root) is at least half full
- invented for disk-based storage, but still heavily used with DRAM
- B stands for Boeing, balanced, block, or Bayer (not: binary)

B+ Trees

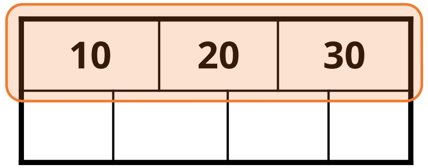
- B+-tree: variant of B-tree where inner nodes only store pointers, no values
- leaf nodes are also often chained to simplify range scan

B+ Tree Index



- Leaf pages contain data entries, and are (optionally) chained (prev & next)
- Non-leaf pages have NO data entries

B+ Tree Basics



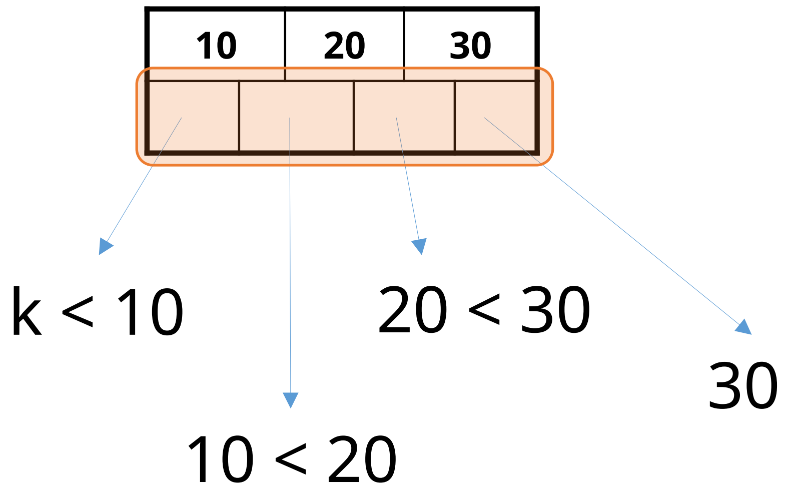
Parameter ***d*** = the order

Each *non-leaf ("interior")* ***node*** has [d,2d] ***entries***

- ***Minimum 50% occupancy***

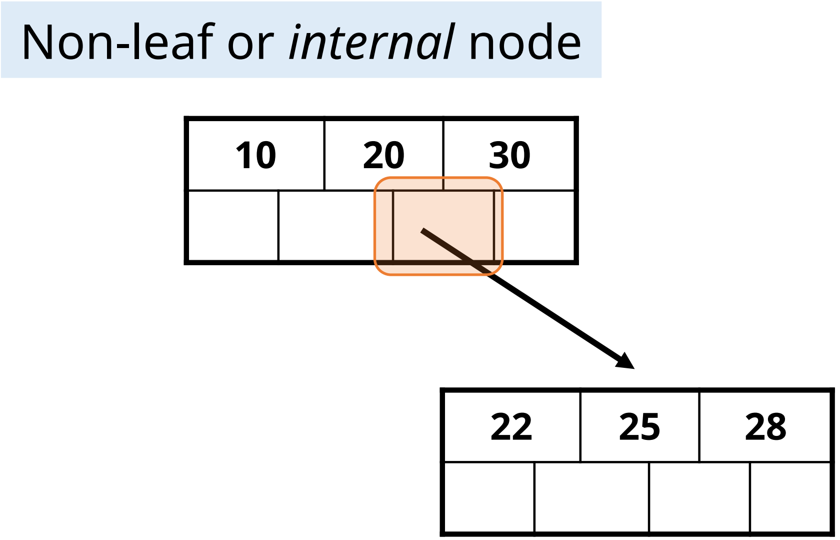
Root ***node*** has [1,2d] ***entries***

B+ Tree Basics



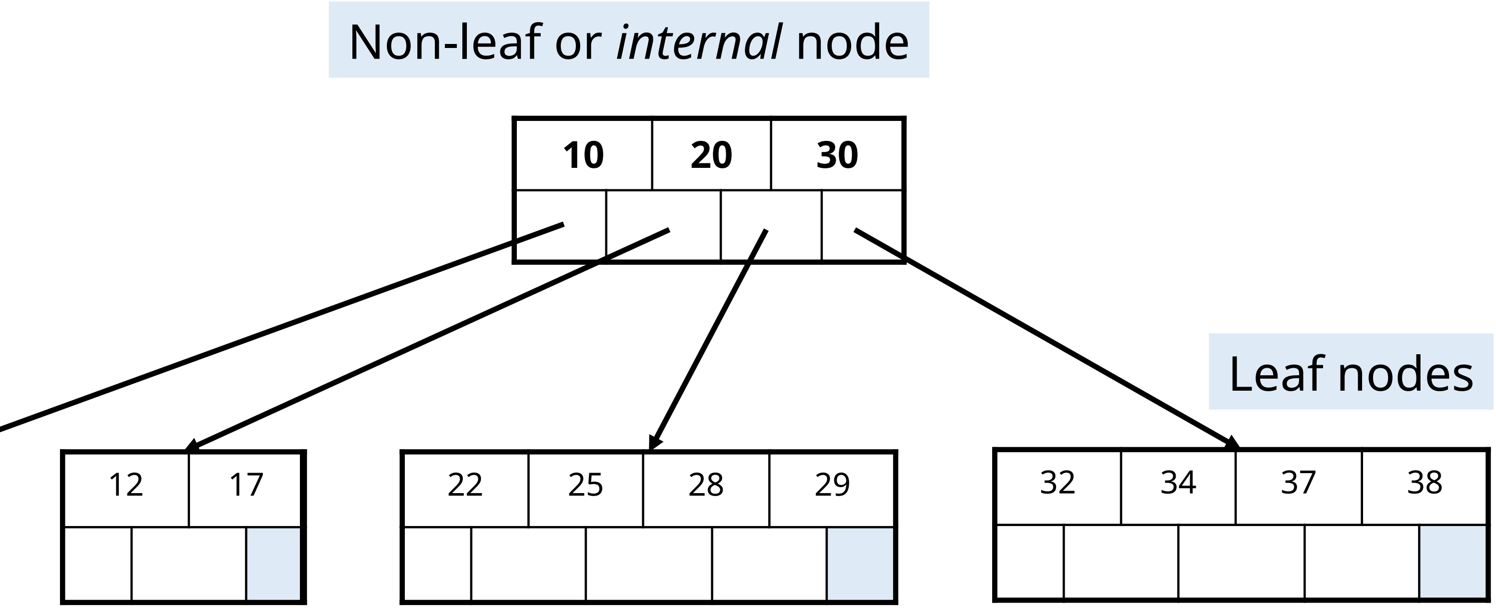
The *n* entries in a node define *n+1* ranges

B+ Tree Basics



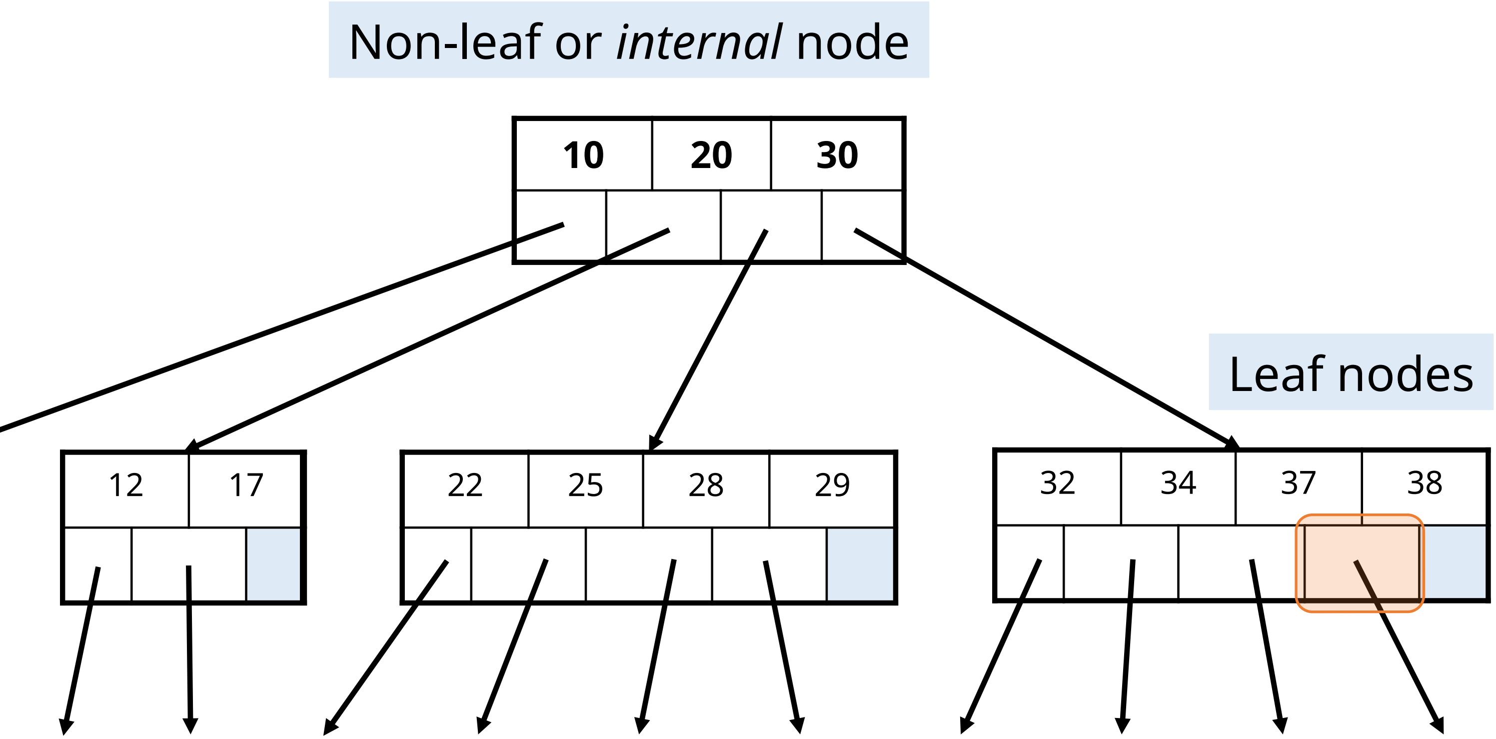
For each range, in a *non-leaf* node, there is a **pointer** to another node with entries in that range

B+ Tree Basics



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

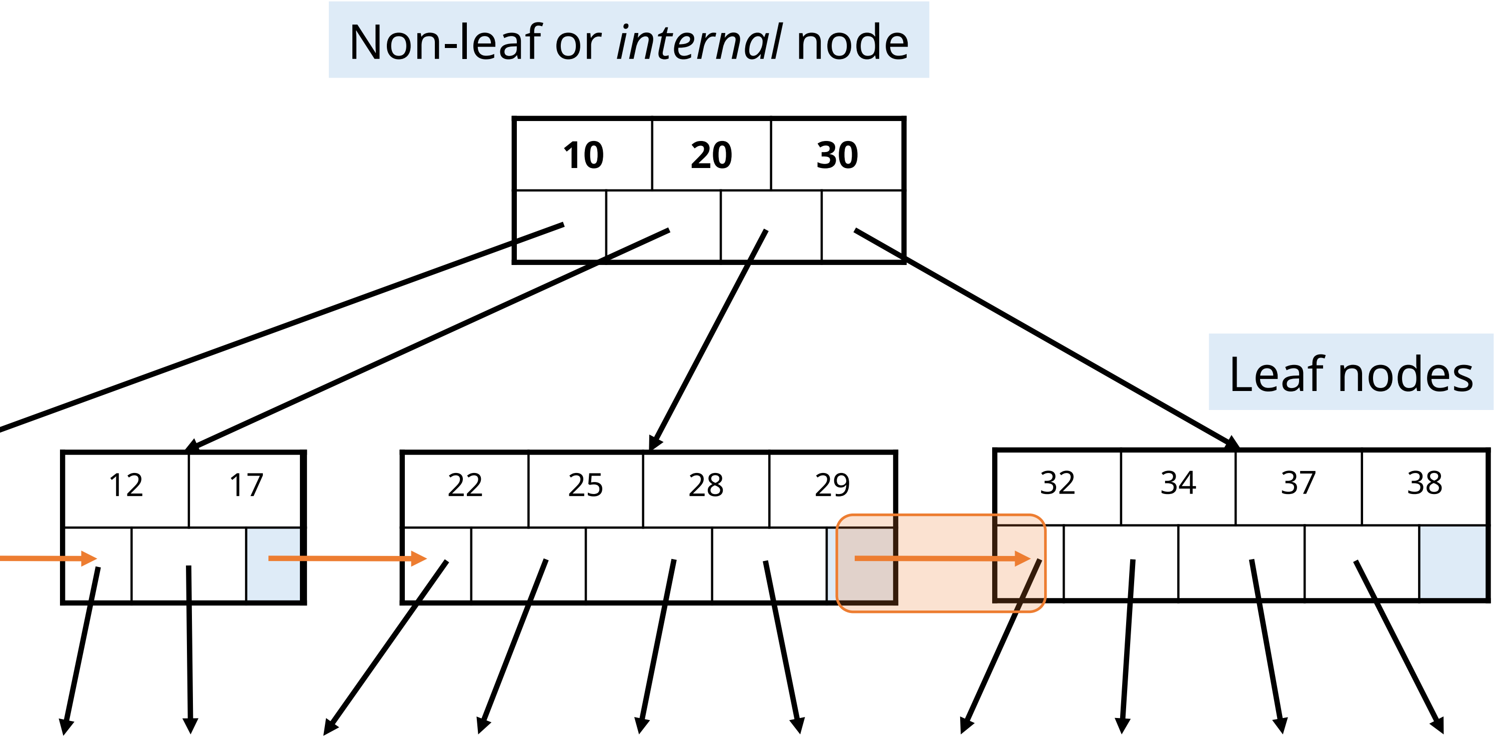
B+ Tree Basics



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

Their entry slots contain recond ID to data records

B+ Tree Basics

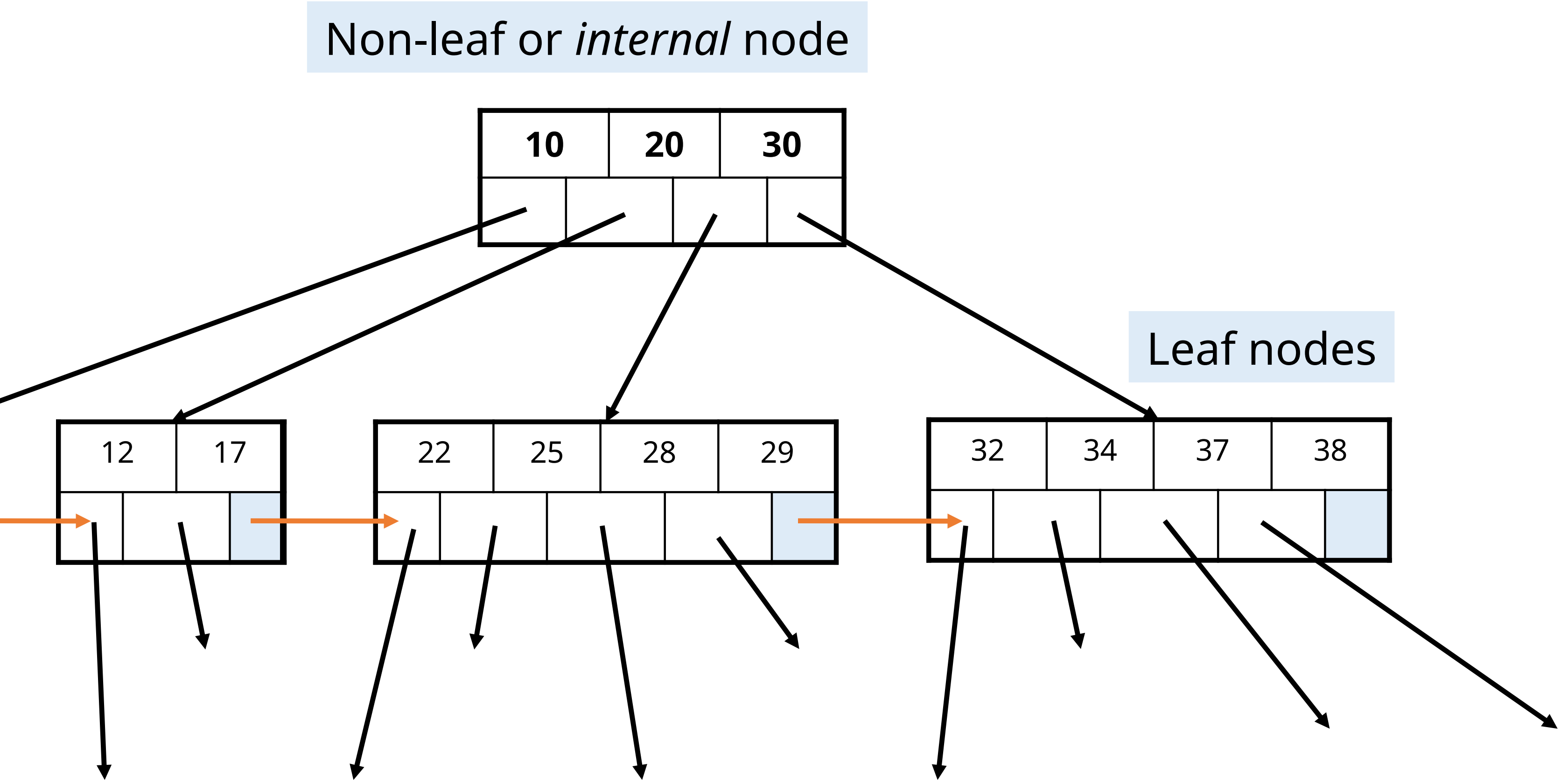


Leaf nodes also have between *d* and *2d* entries, and are different in that:

Their entry slots contain pointers to data records

They contain a pointer to the next leaf node as well, ***for faster sequential traversal***

B+ Tree Basics

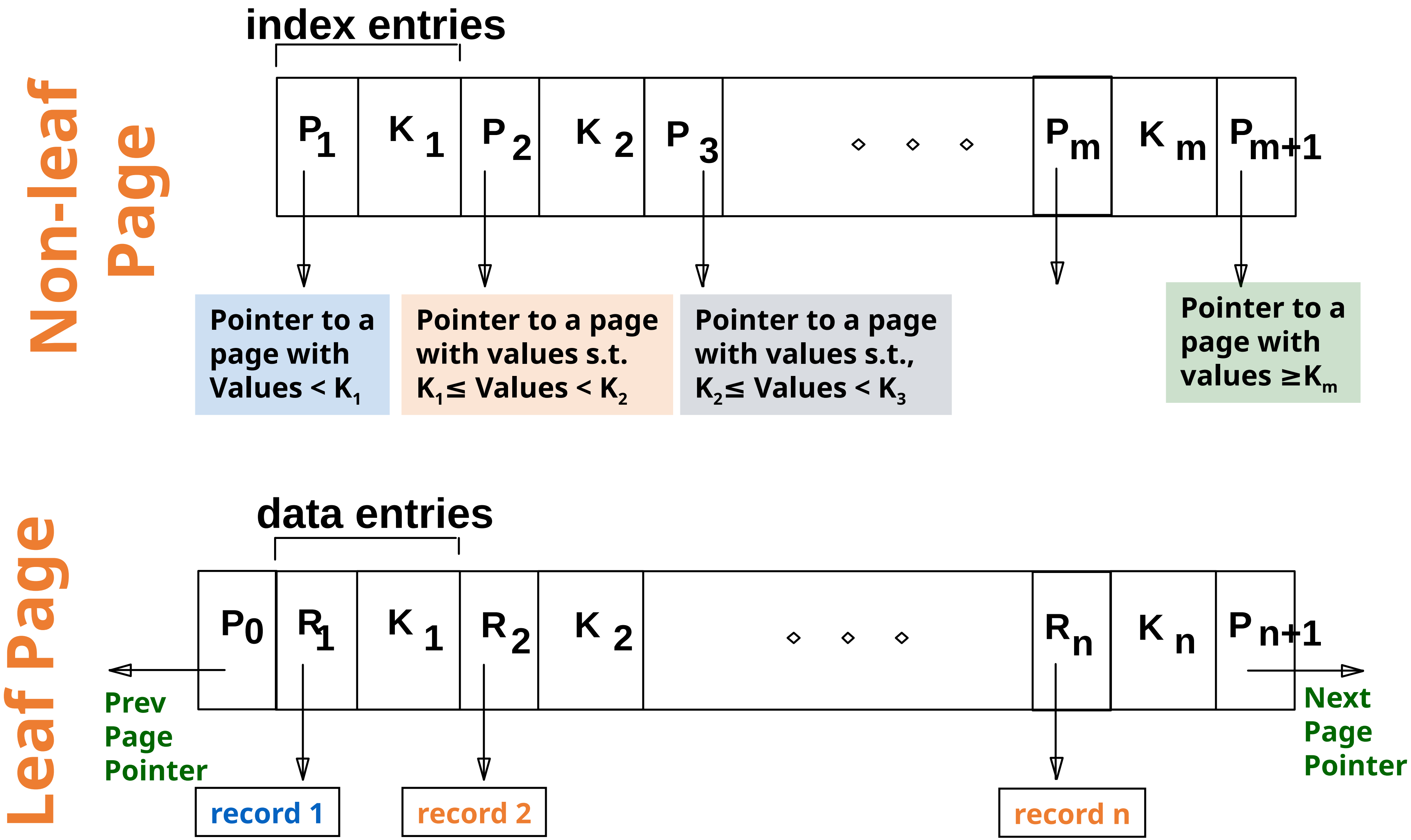


Note that the pointers at the leaf level will be to the actual data records (rows).

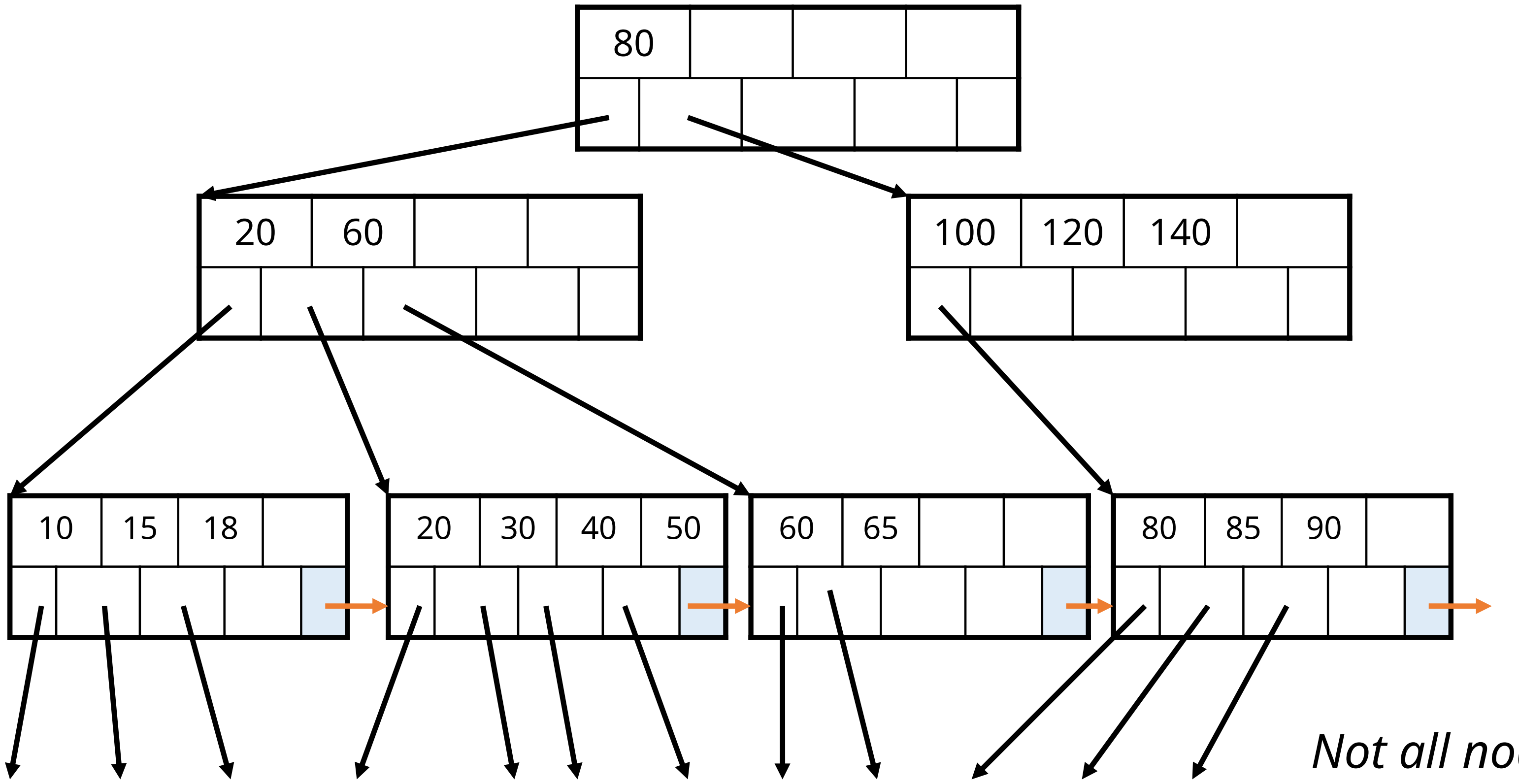
We might truncate these for simpler display (as before)…

Height = 1

B+ Tree Page Format

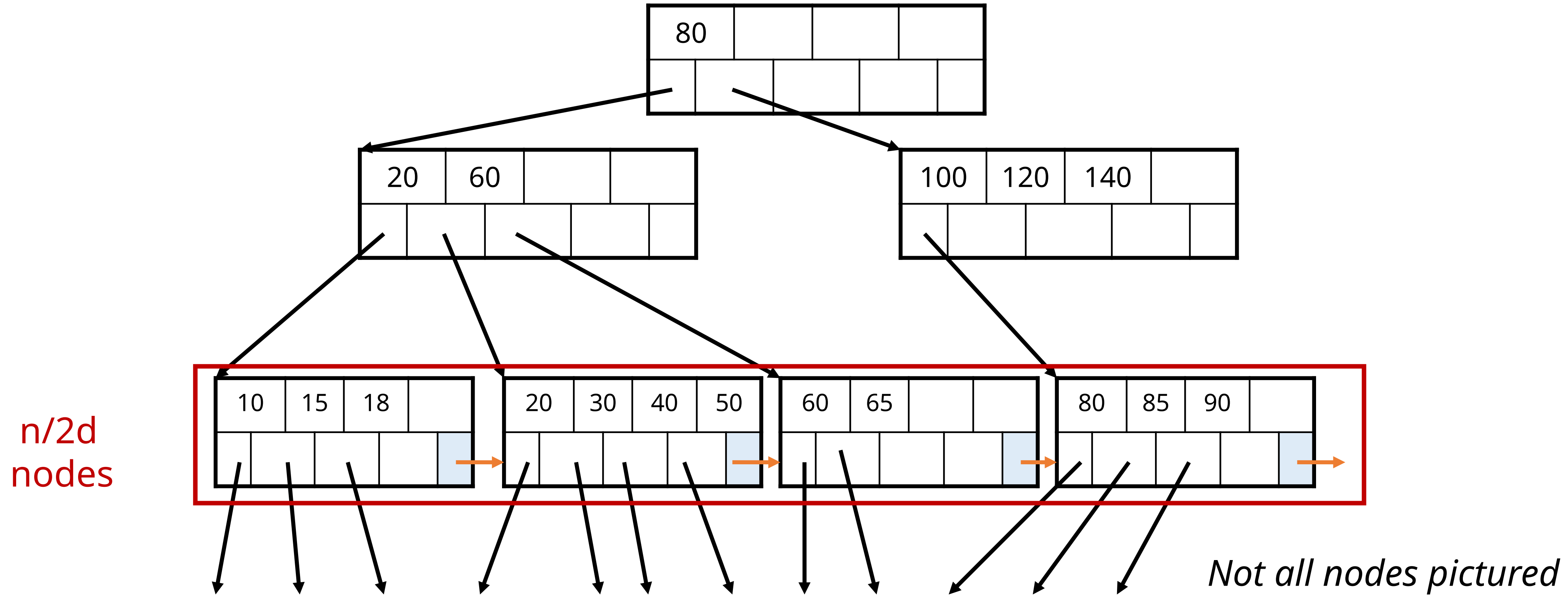


Tree height?



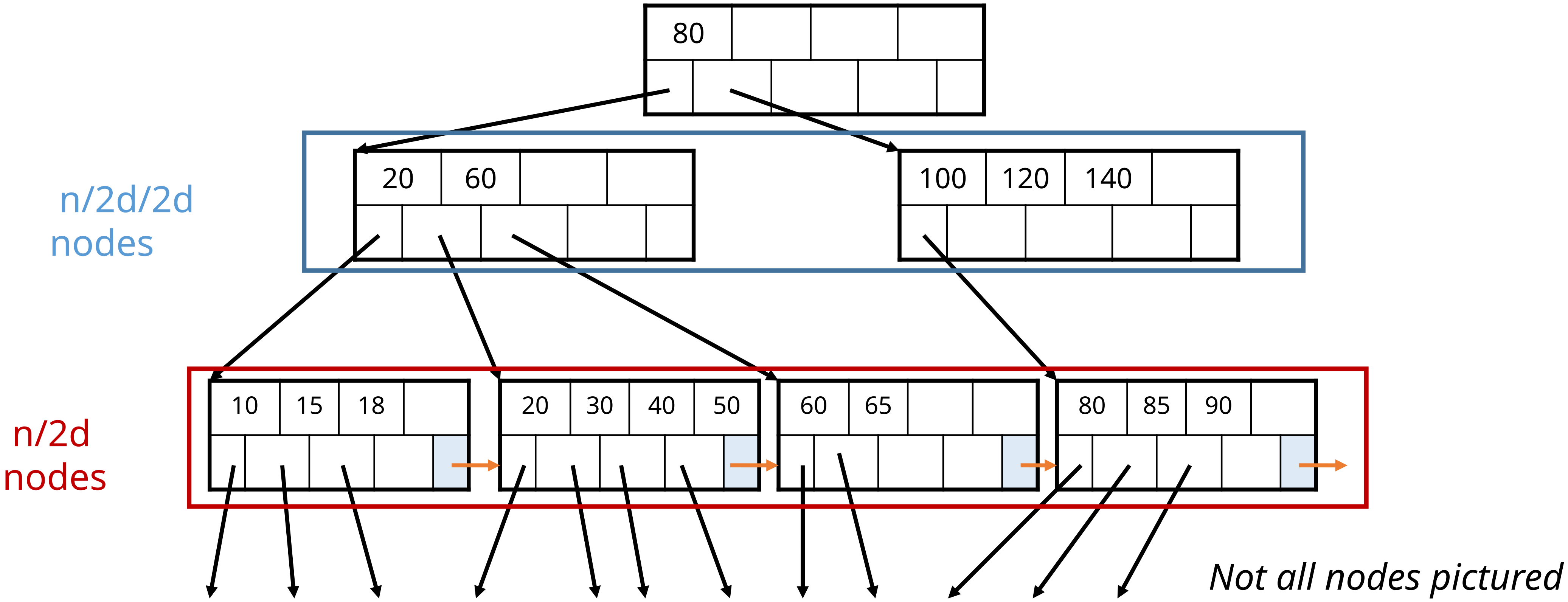
Not all nodes pictured

Tree height?

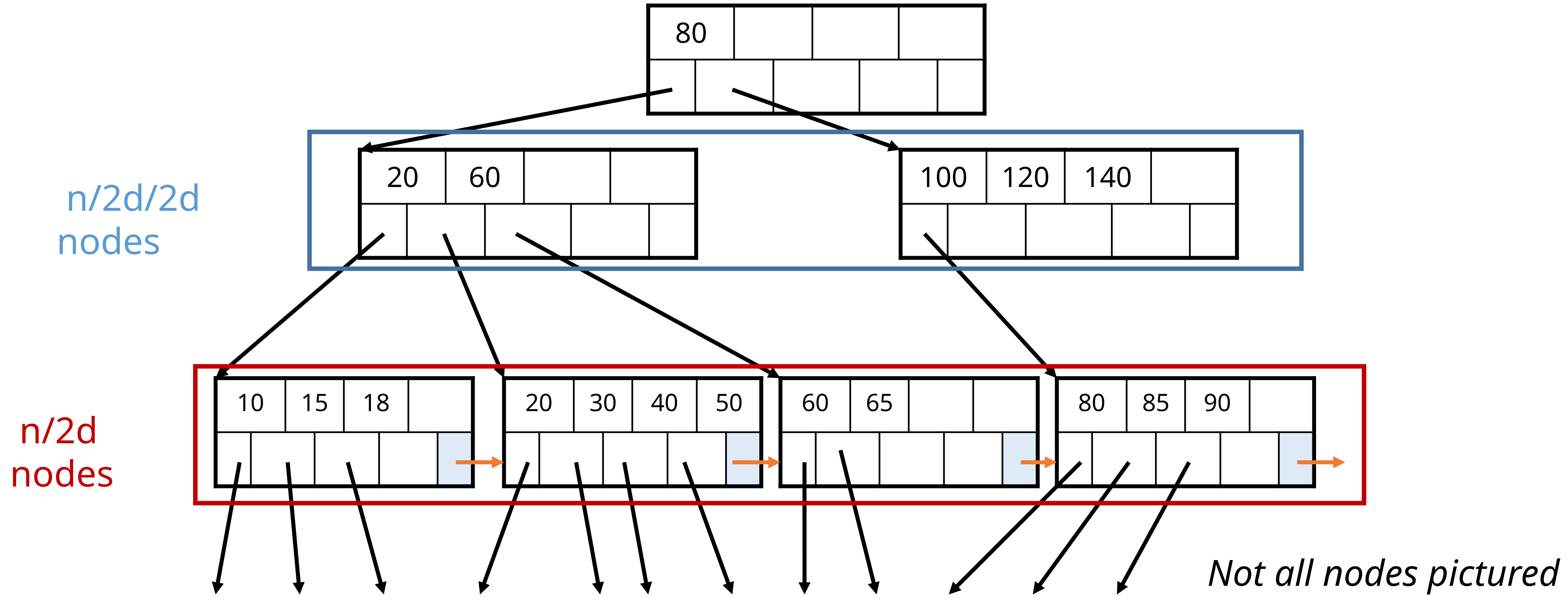


80
20
60
100
120
140
10
15
18
20
30
40
50
60
65
80
85
90
n/2d
nodes
Not all nodes pictured

Tree height?


80
20
60
100
120
140
n/2d/2d
nodes
10
15
18
20
30
40
50
60
65
80
85
90
n/2d
nodes
Not all nodes pictured

Tree height?



Best case tree height: $\log_{2d}(n)$

Tree height <= $\log_{d}(n)$

B+ Trees: Operations

B+ Tree operations

A B+ tree supports the following operations:

- equality search
- range search
- insert
- delete
- bulk loading

B+ Tree: Search

- start from root
- examine index entries in non-leaf nodes to find the correct child
- traverse down the tree until a leaf node is reached

B+ Tree Exact Search Animation

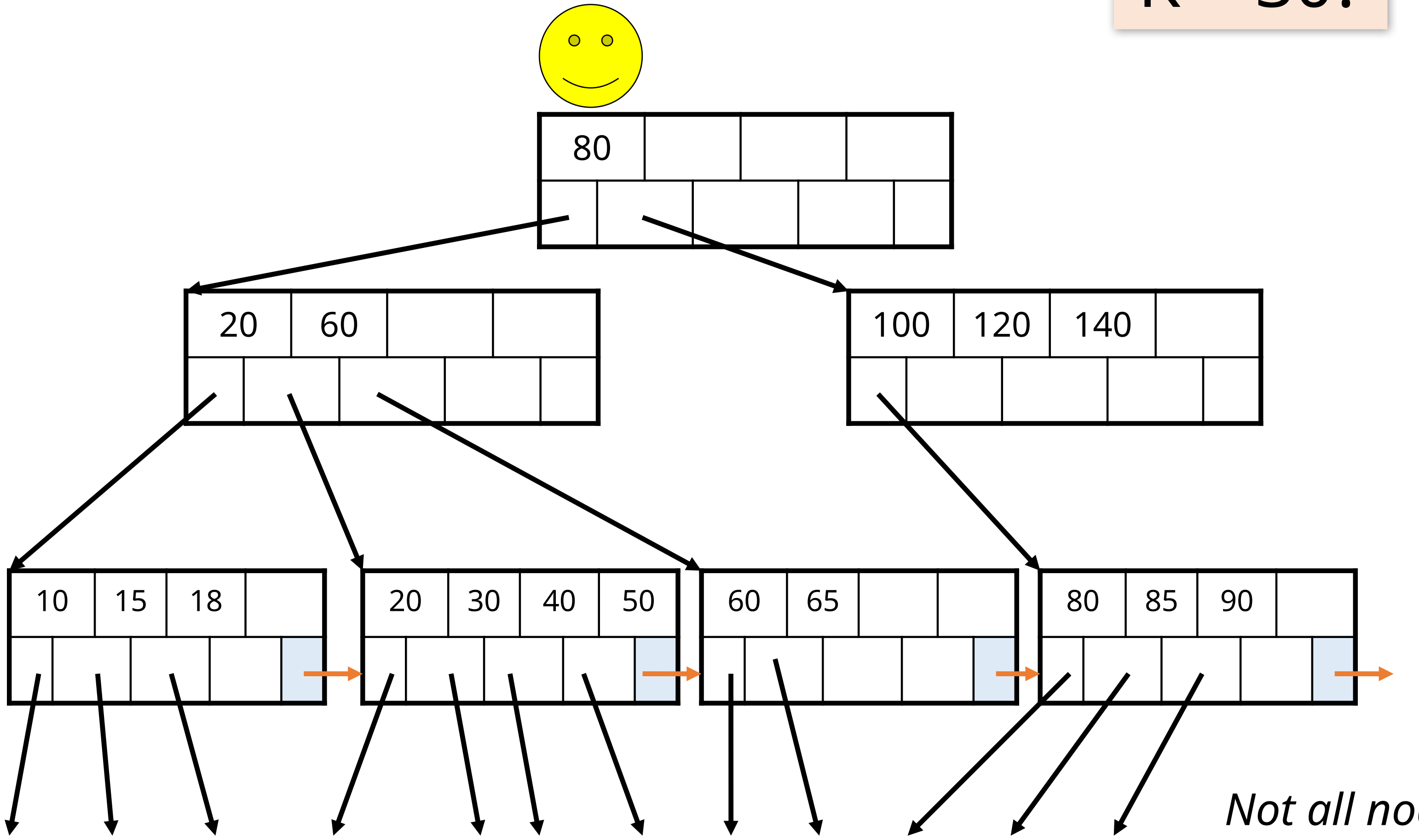
K = 30?

30 < 80

30 in [20,60)

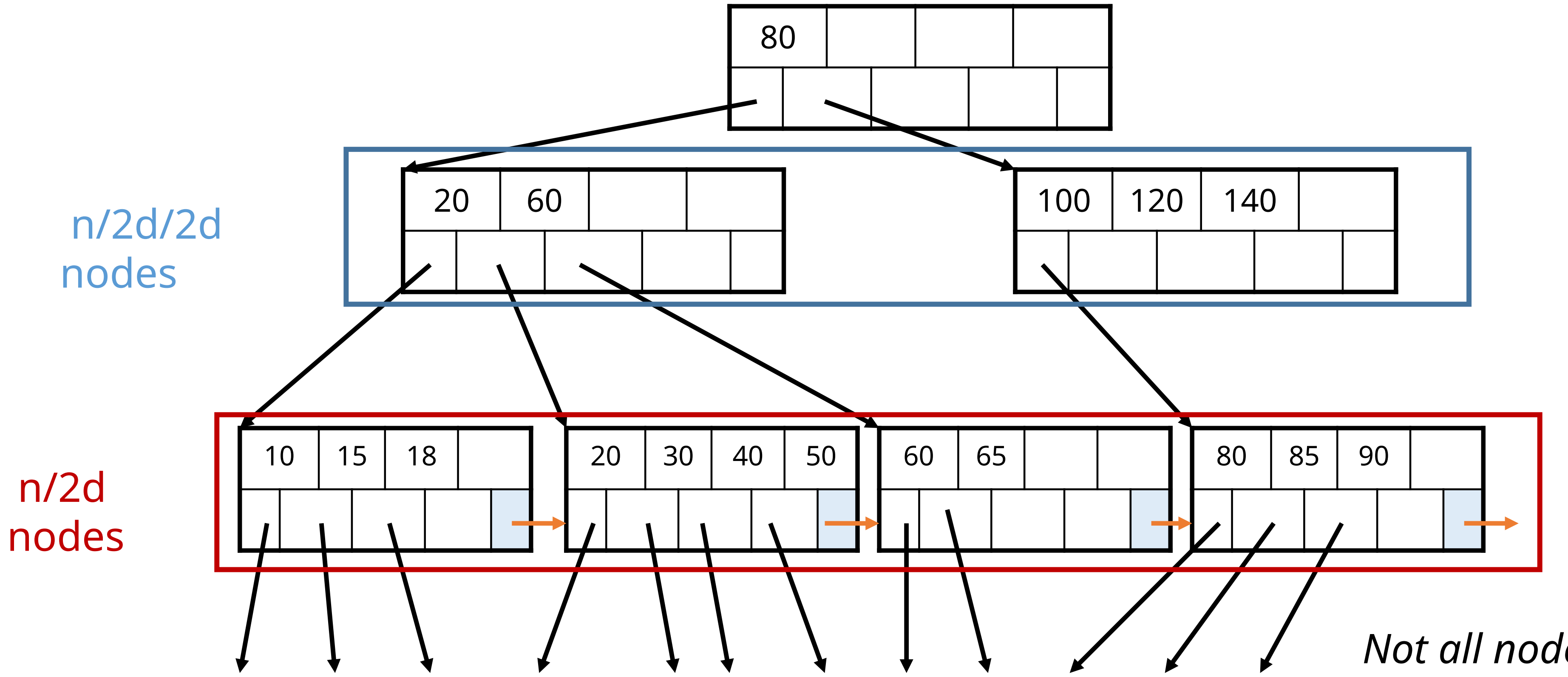
30

To the data!



Not all nodes pictured

How many comparisons for the exact search?

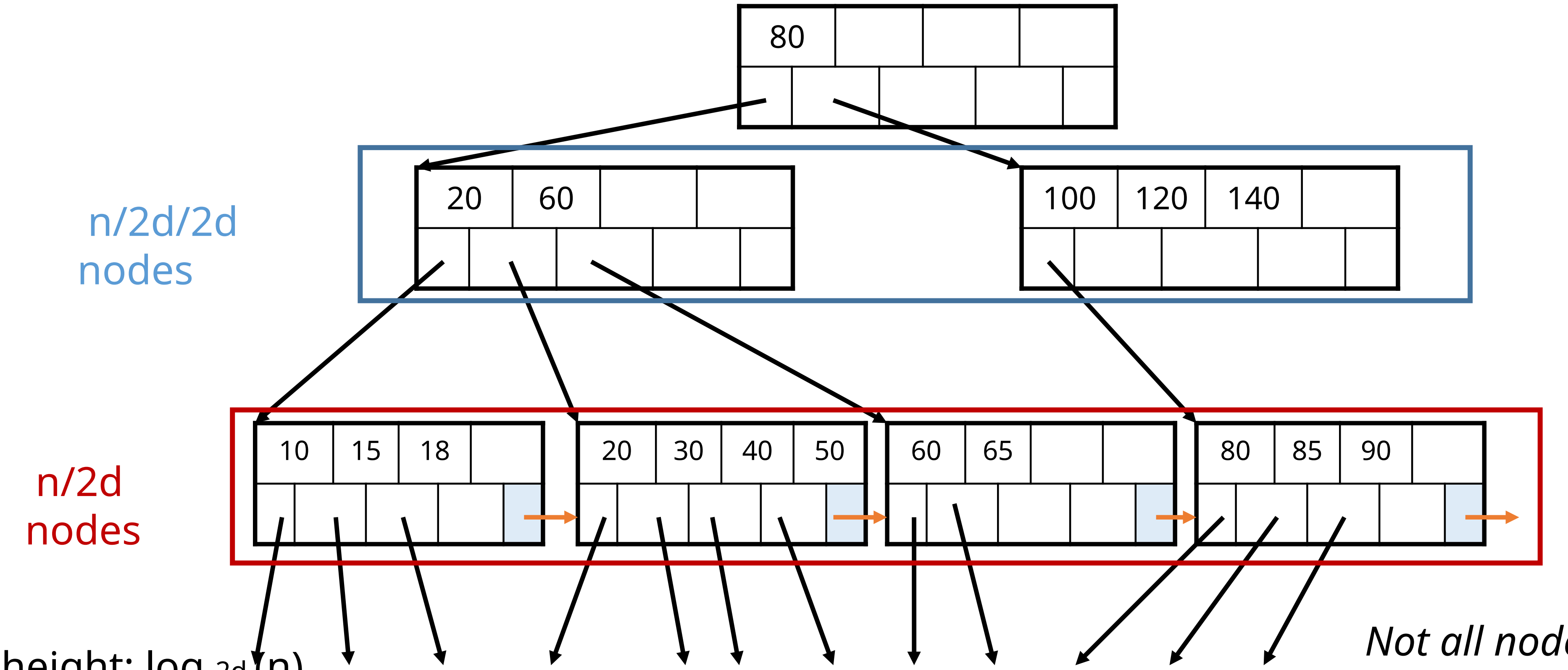


Not all nodes pictured

Best case tree height: $\log_{2d}(n)$
tree height <= $\log_{d}(n)$

How many comparisons for the exact search?

comparisons: $\log_d(n) \cdot \log_2(d) = \log_2(n)$ (like binary search)



Not all nodes pictured

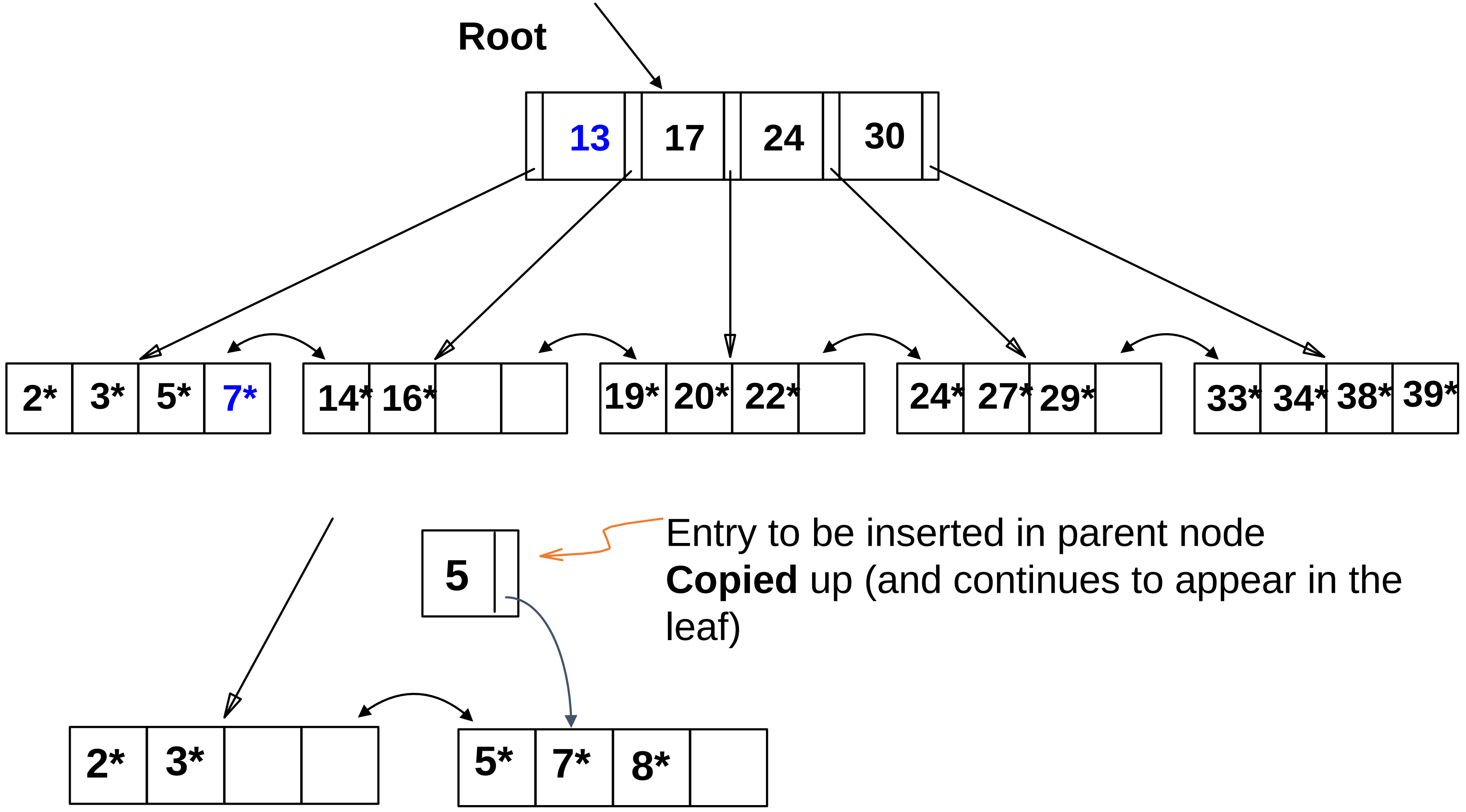
Best case tree height: $\log_{2d}(n)$

Tree height <= $\log_d(n)$

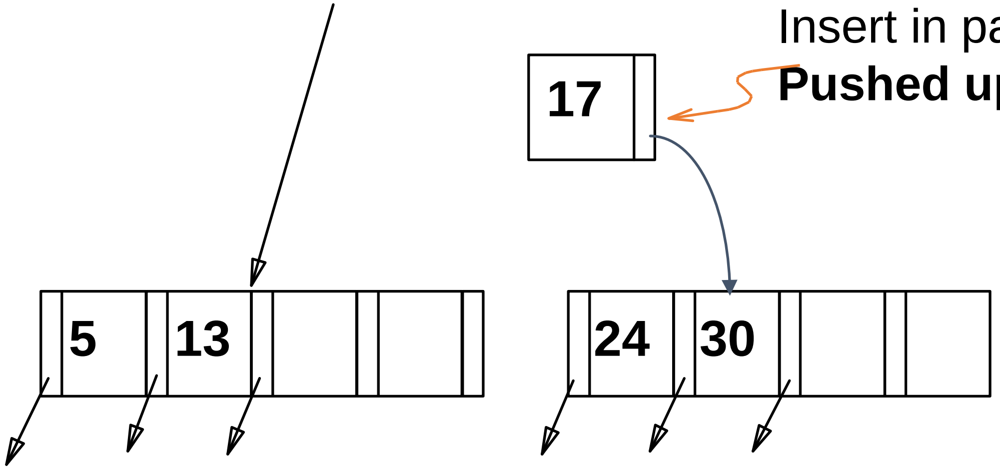
B+ Tree: Insert

- 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, **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To split non-leaf node, redistribute entries evenly, but **pushing up** the middle key. (Contrast with leaf splits.)
- Splits “grow” tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top.*

Inserting 8* into B+ Tree



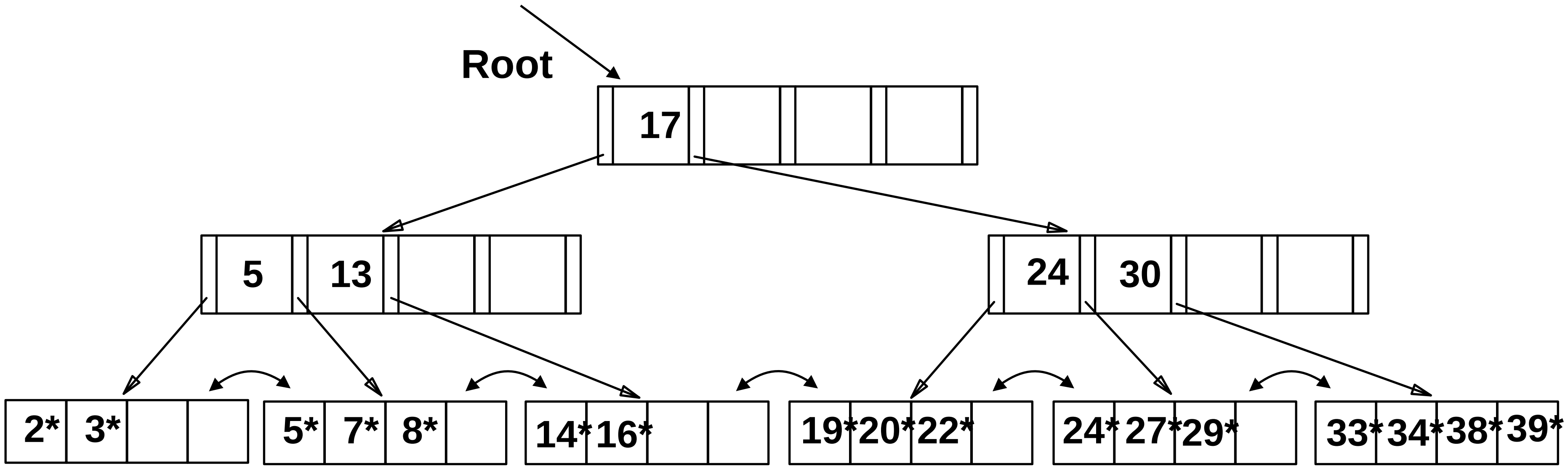
Inserting 8* into B+ Tree



Insert in parent node.
Pushed up (and only appears once in the index)

Minimum occupancy is guaranteed in both leaf and index page splits

Inserting 8* into B+ Tree



- Root was split: height increases by 1
- Could avoid split by re-distributing entries with a sibling
  - Sibling: immediately to left or right, and same parent

Deletion and Textbook Merge

- to avoid underfull nodes after deletion one must merge or rebalance
- B-tree delete is more complicated than insert
- if node is underfull (<50% fill factor):

    find a neighboring node

    if neighboring has low enough fill factor:

        merge the two nodes into one

        remove separator in parent

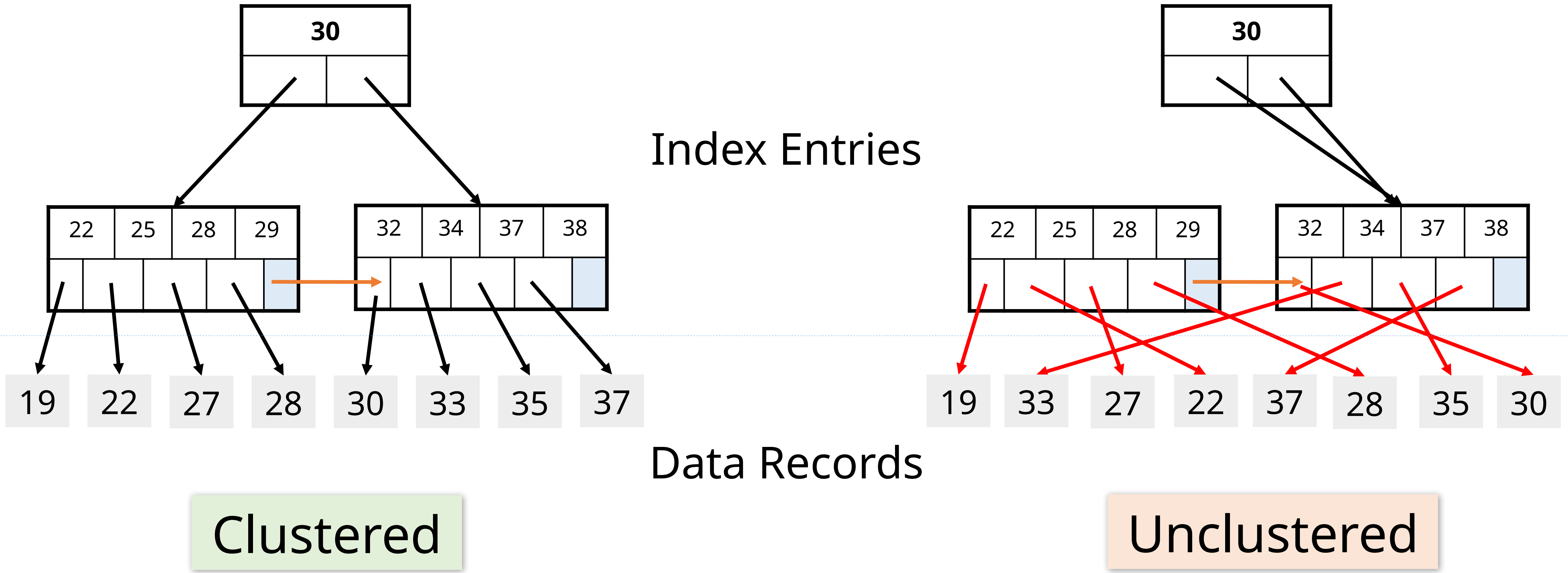
        if parent is underfull merge parent the same way

    else rebalance elements between the two nodes

Clustered Indexes

An index is ***clustered*** if the underlying data is ordered in the same way as the index's data entries.

Clustered vs. Unclustered Index



Summary

- We create **indexes** over tables in order to support ***fast (exact and range) search*** and ***insertion*** over ***multiple search keys***

- **B+ Trees** are one index data structure which support very fast exact and range search & insertion via ***high fanout***
  - ***Clustered vs. unclustered*** makes a big difference for range queries too