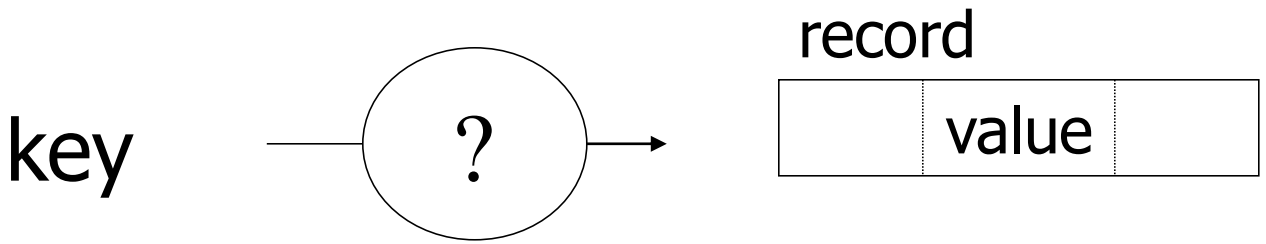
Notes 4: B+ Tree

Peter Bailis

Chapter 4

Indexing



Index is a data structure for search.

Index Types

- B-Trees *(covered next)*
  - Very good for sorted data, range search
  - *We will look at a variant called **B+ Trees***
- Hash Tables

Topics

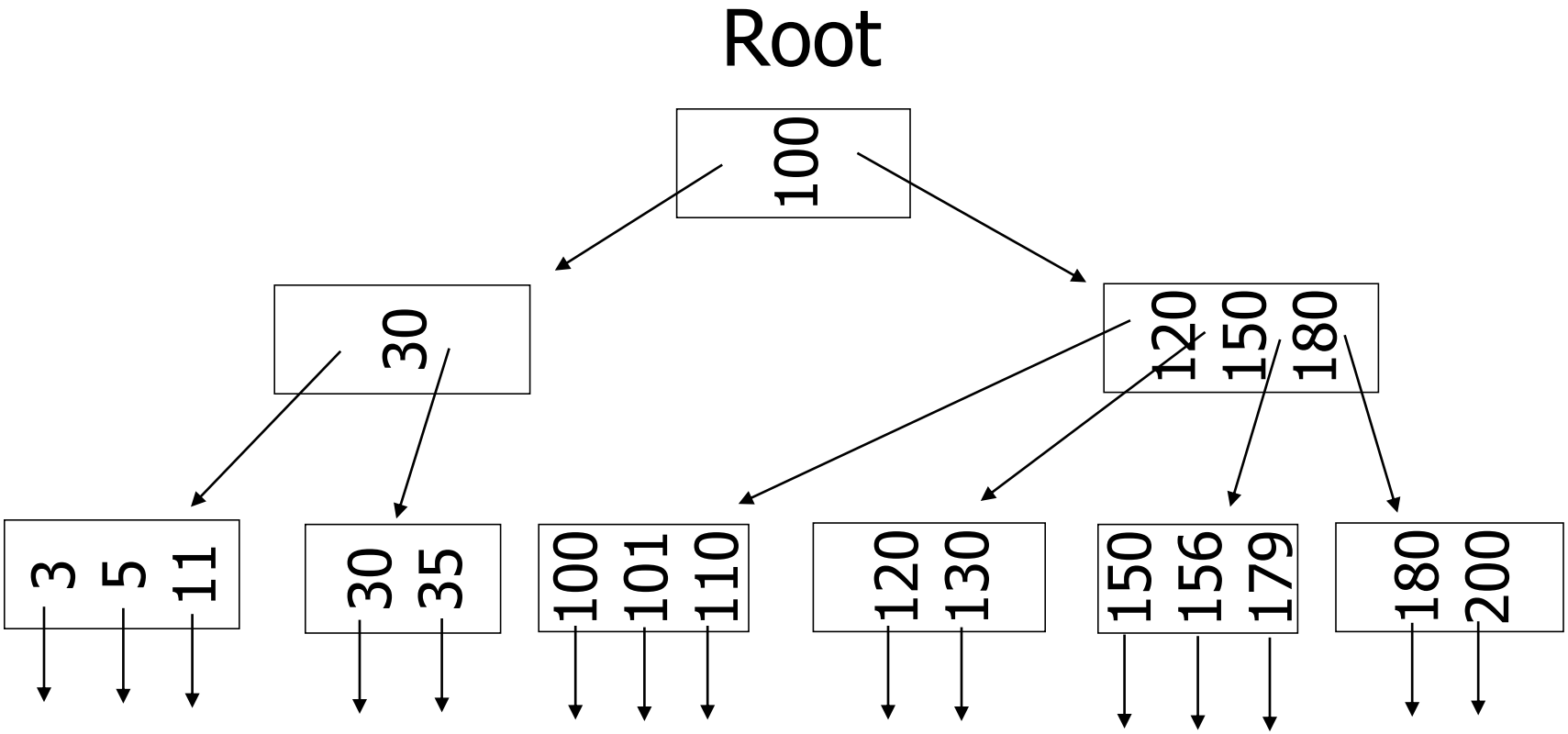
- B-trees
- Hashing schemes

Outline:

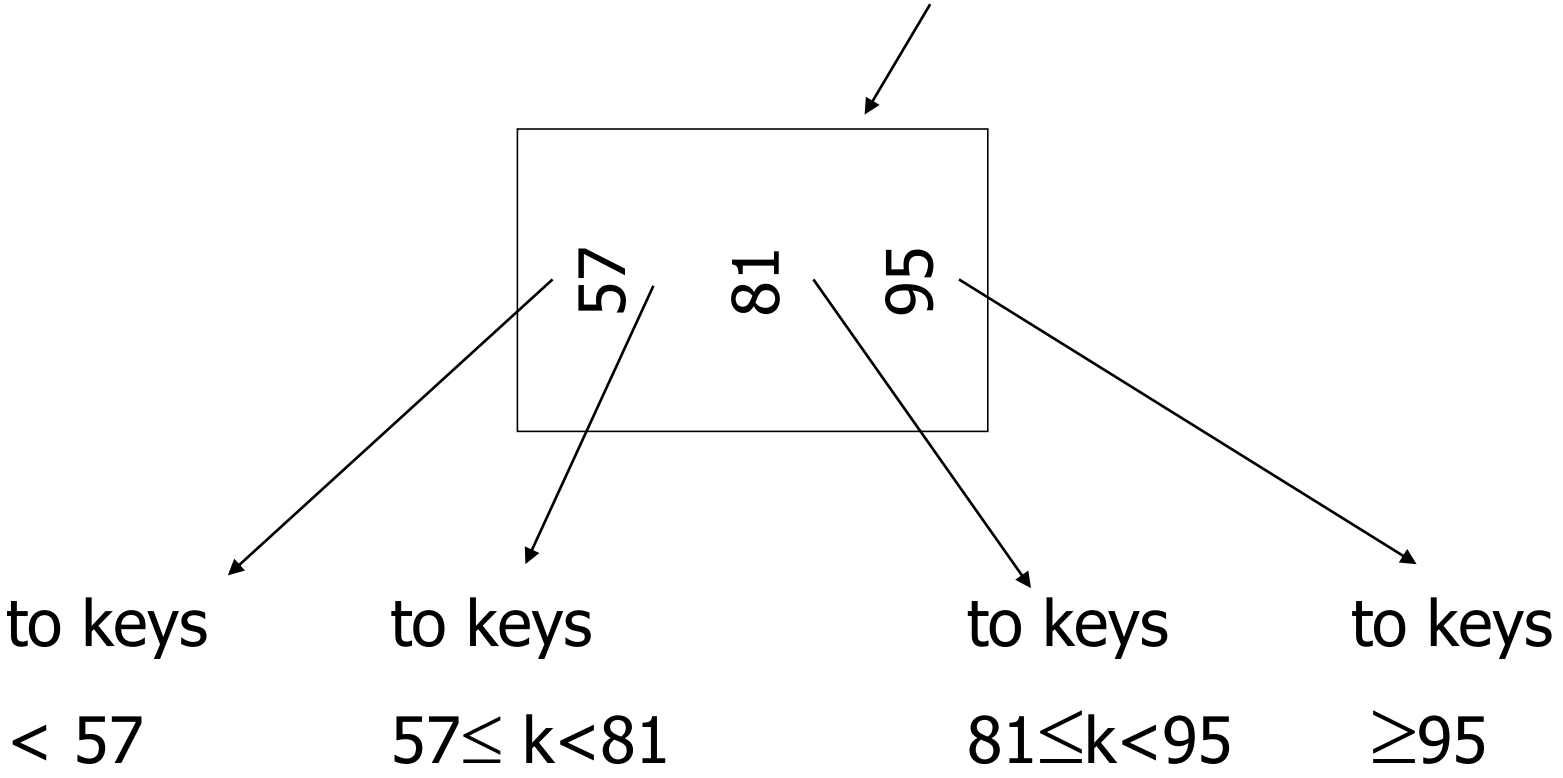
- B-Trees $\Rightarrow$ NEXT
- Hashing schemes

Note: This index is called B+ tree, Sometimes just call it B-tree.

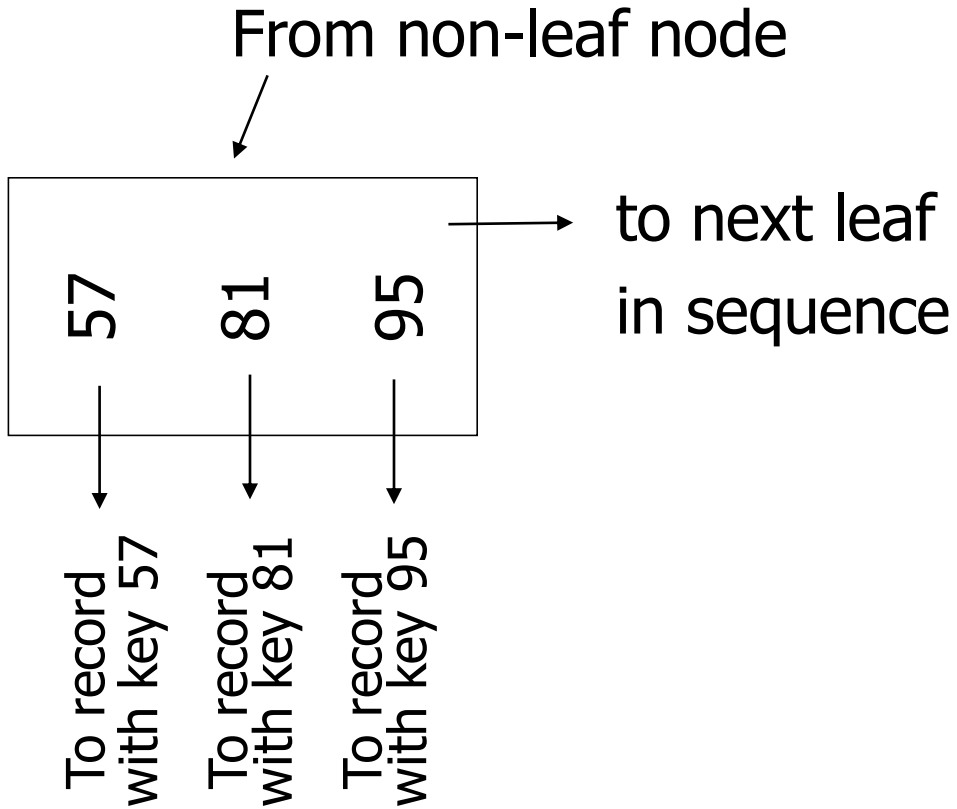
B+Tree Example n=3



Sample non-leaf

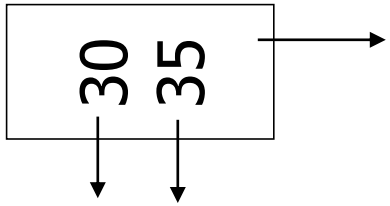


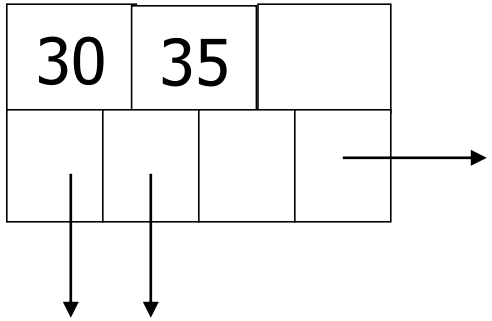
Sample leaf node:



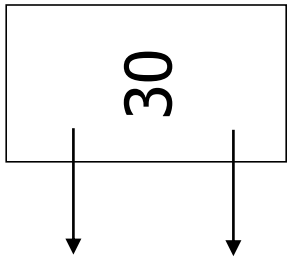
In textbook's notation n=3

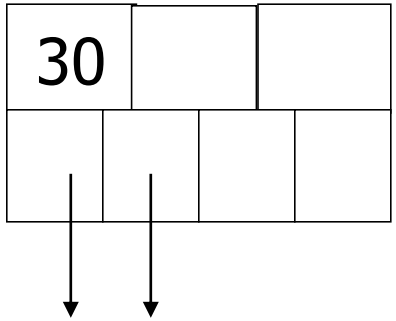
Leaf:





Non-leaf:





Size of nodes:

- n+1 pointers
- n keys

(fixed)

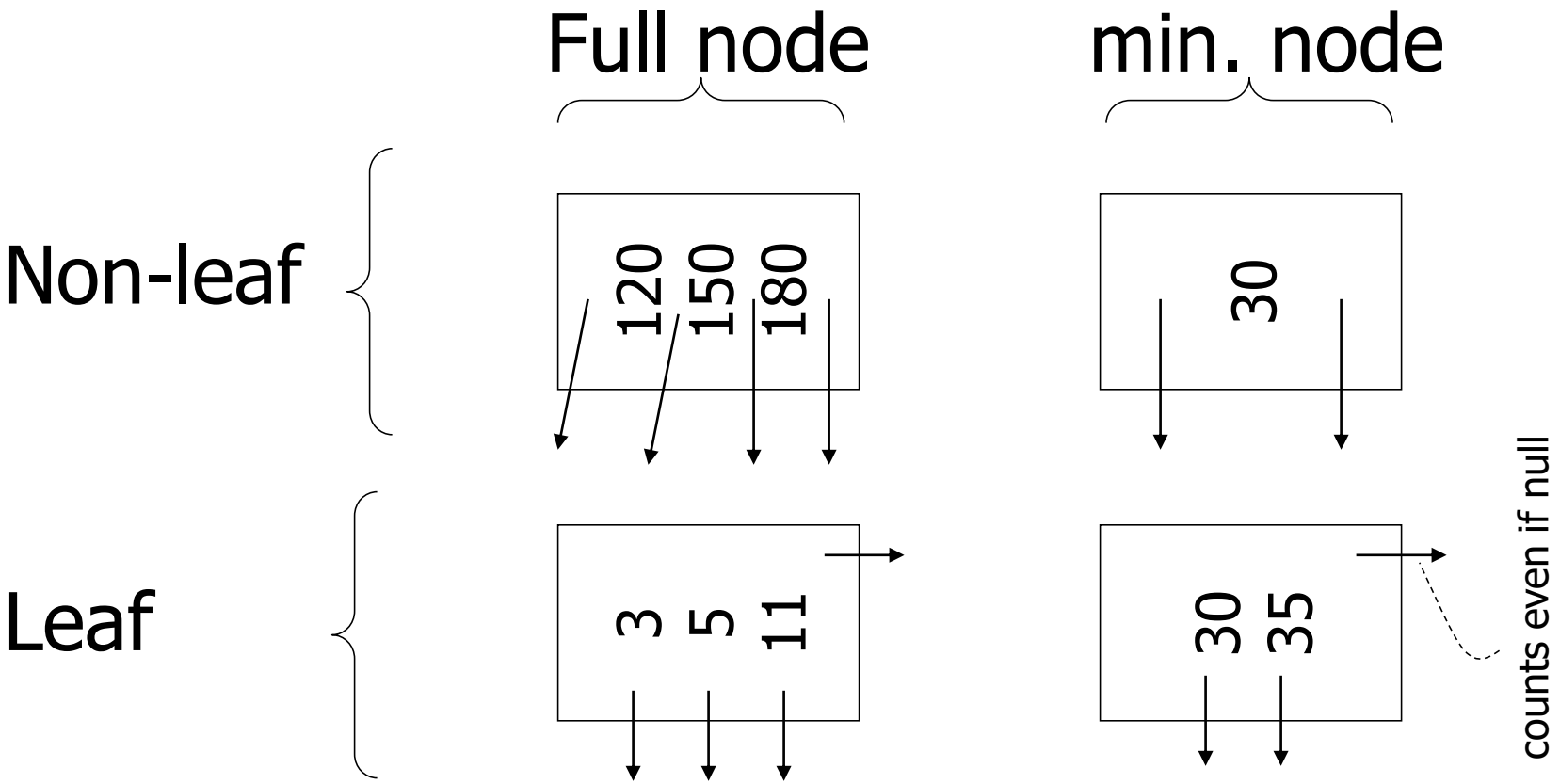
Don't want nodes to be too empty

- Use at least

  Non-leaf: $\lceil (n+1)/2 \rceil$ pointers

  Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data

n=3


Full node
min. node
Non-leaf
Leaf
120
150
180
30
3
5
11
30
35
counts even if null

B+tree rules — tree of order *n*

(1) All leaves at same lowest level
(balanced tree)

(2) Pointers in leaves point to records
except for "sequence pointer"

(3) Number of pointers/keys for B+tree

	Max ptrs	Max keys	Min ptrs→data	Min keys
Non-leaf (non-root)	n+1	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Leaf (non-root)	n+1	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	n+1	n	2*	1

* When there is only one record in the B-tree, min pointers in the root is 1 (the other pointers are null)

Insert into B+tree

(a) simple case
  - space available in leaf

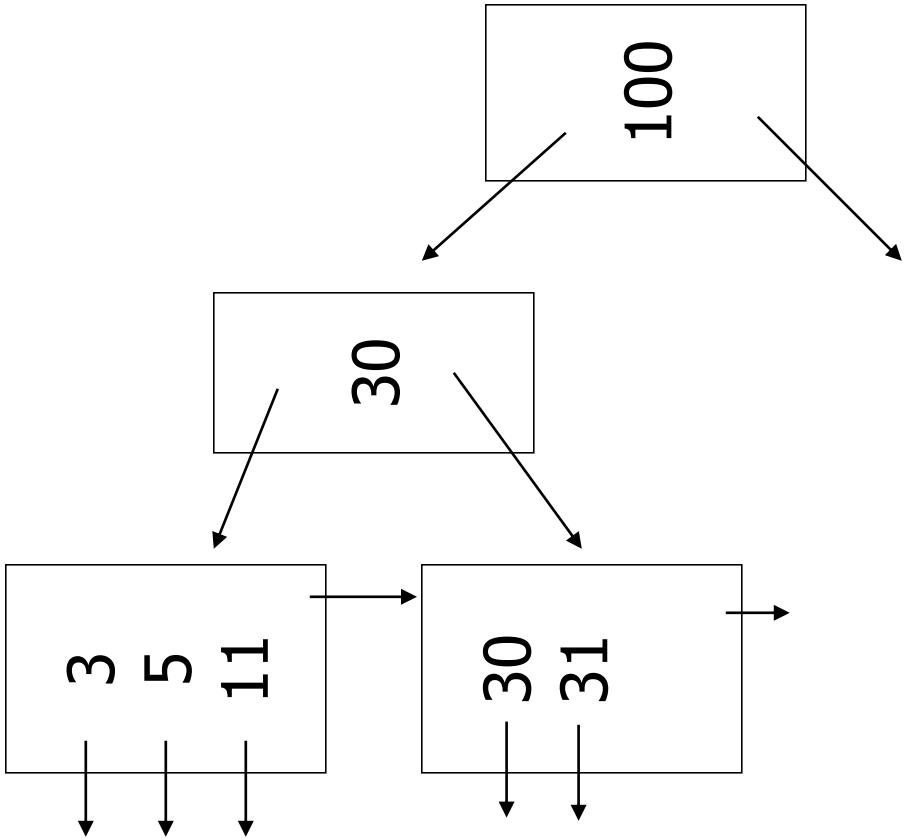
(b) leaf overflow

(c) non-leaf overflow

(d) new root

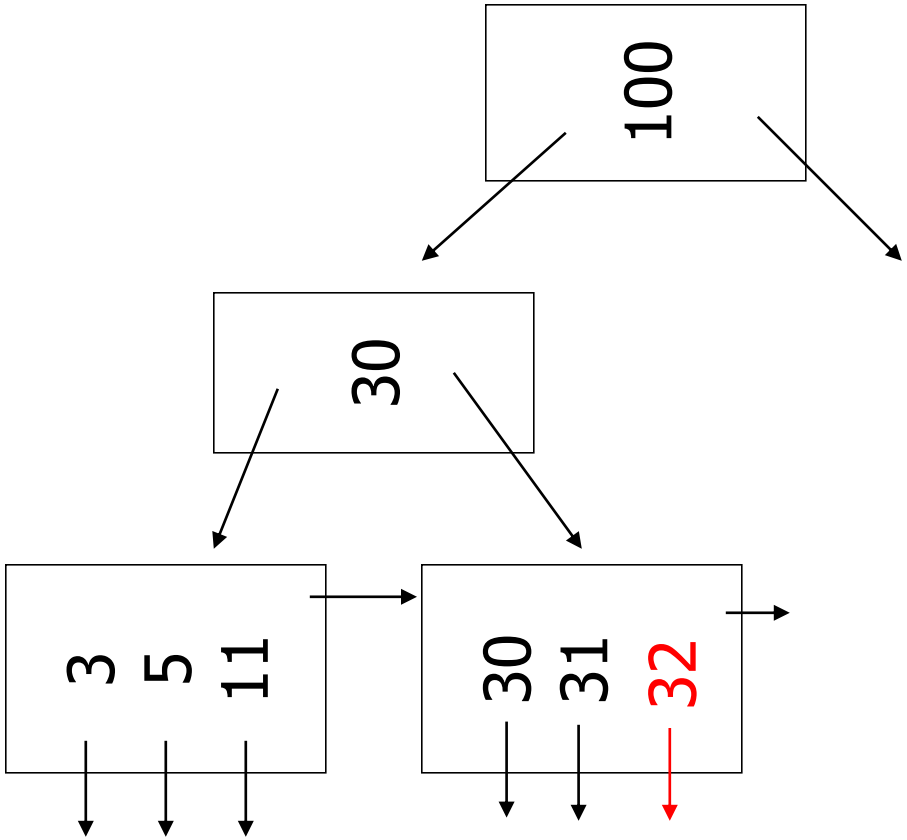
(a) Insert key = 32

n=3



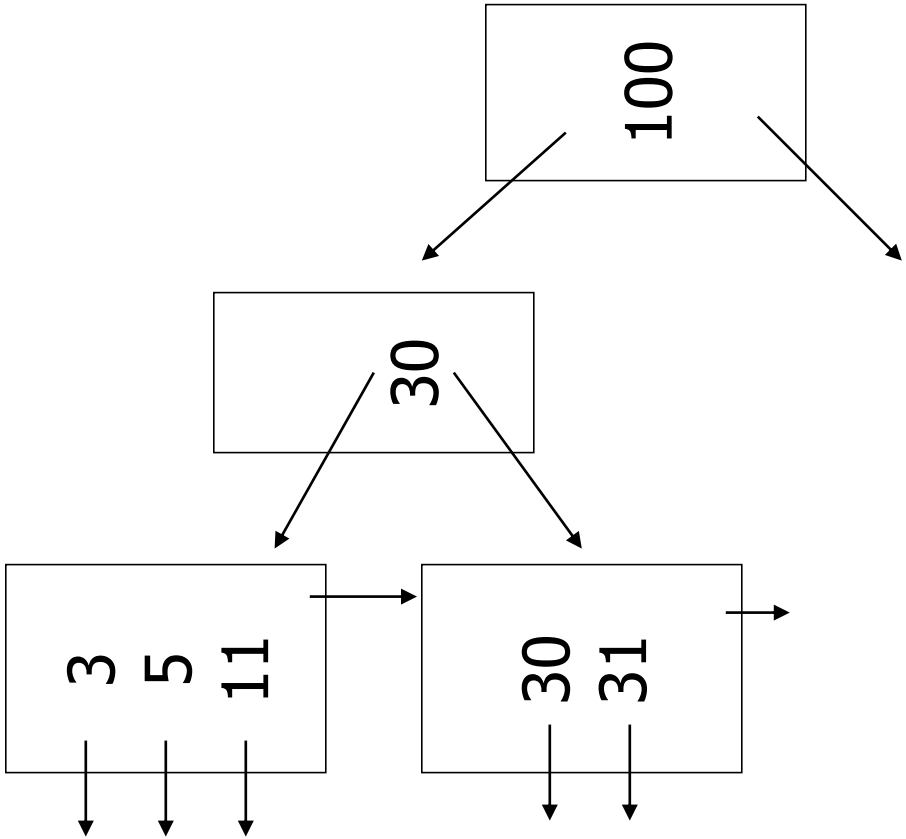
(a) Insert key = 32

n=3



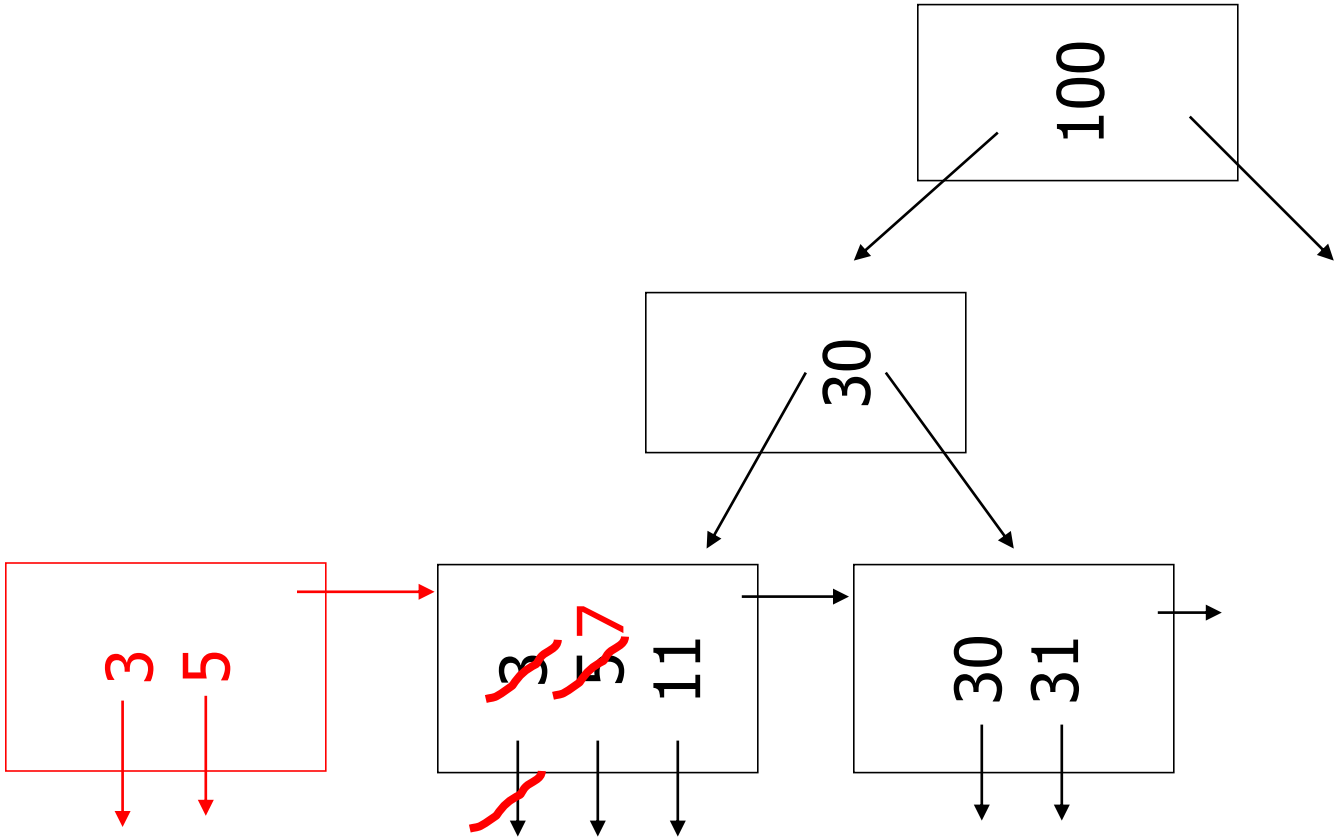
(a) Insert key = 7

n=3



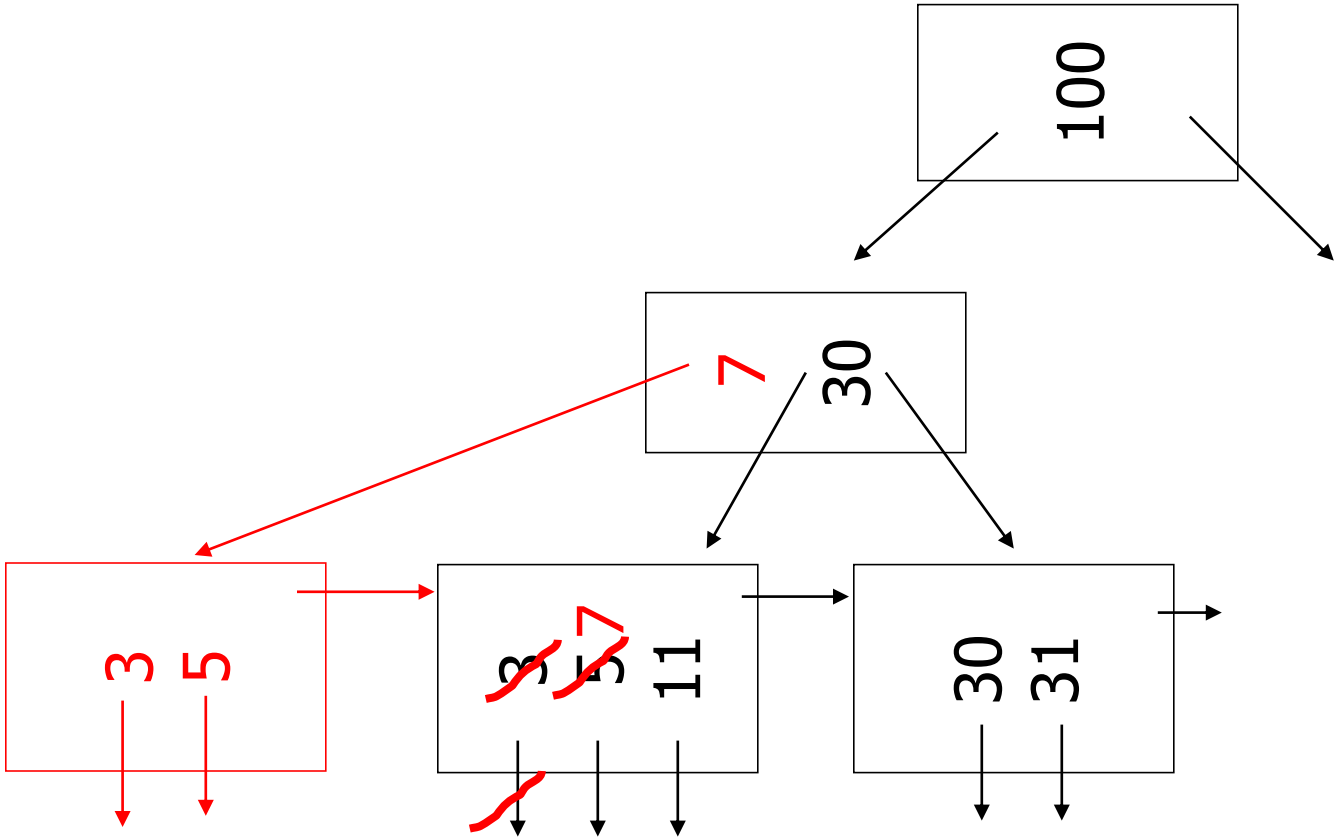
(a) Insert key = 7

n=3



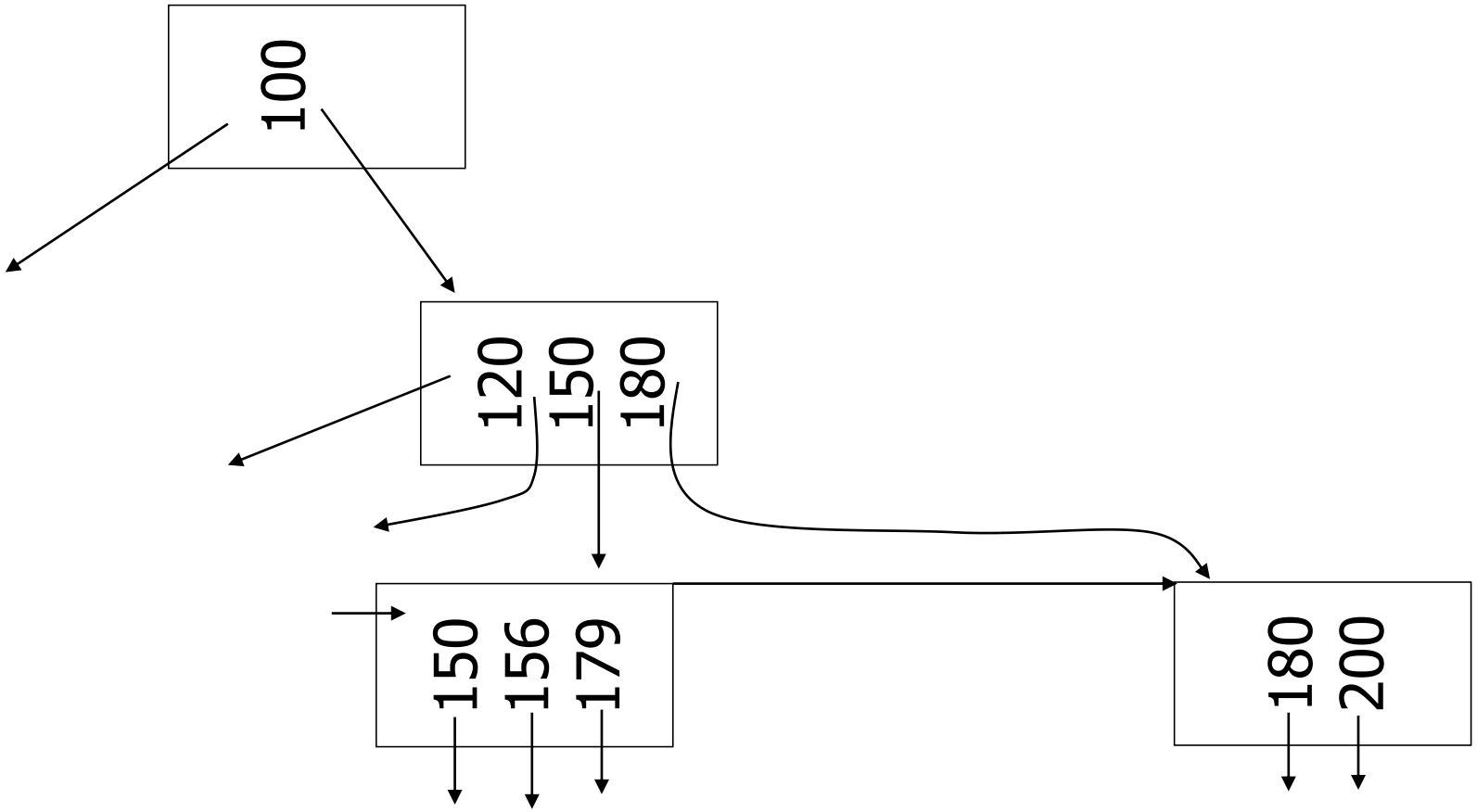
(a) Insert key = 7

n=3



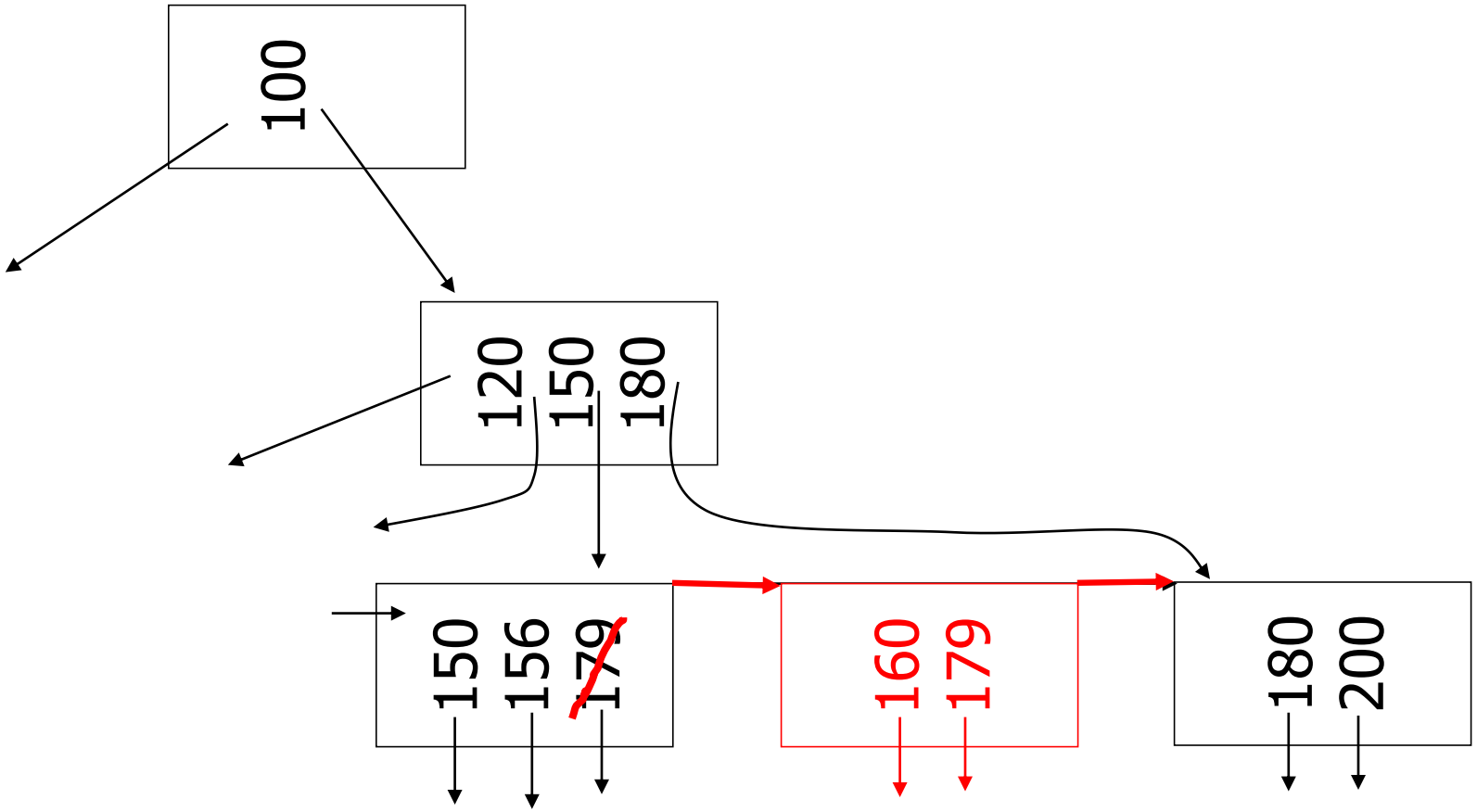
(c) Insert key = 160

n=3



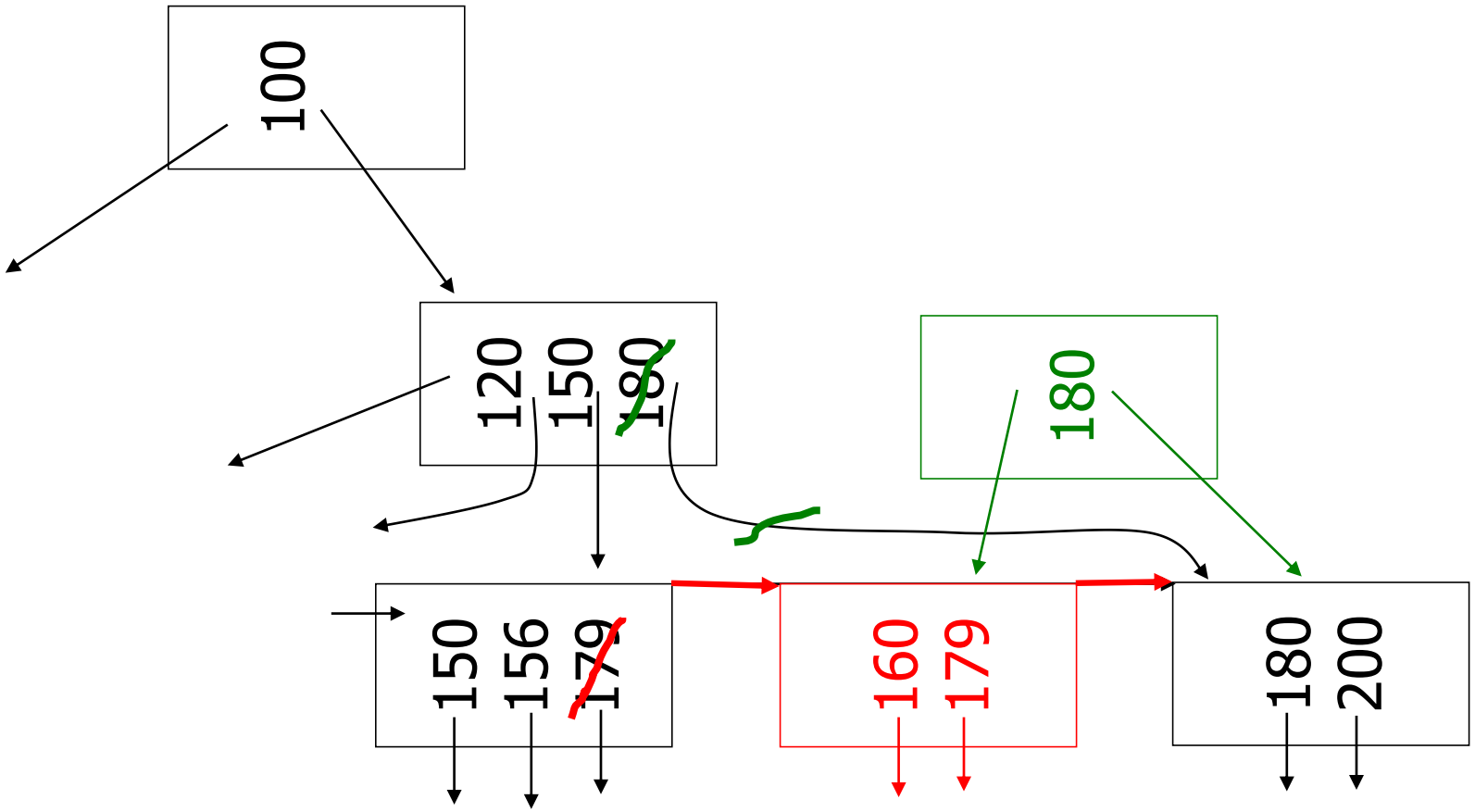
(c) Insert key = 160

n=3



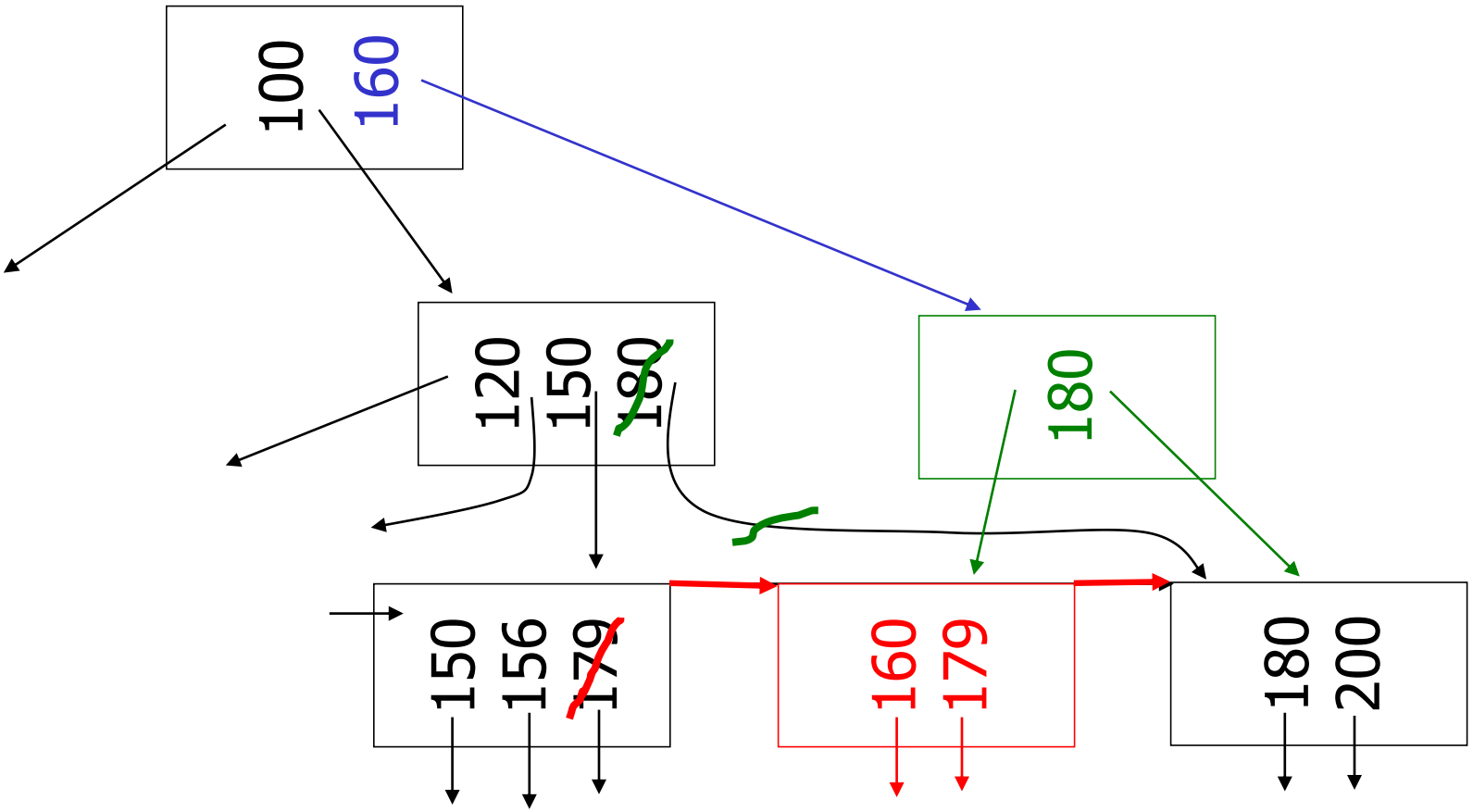
(c) Insert key = 160

n=3



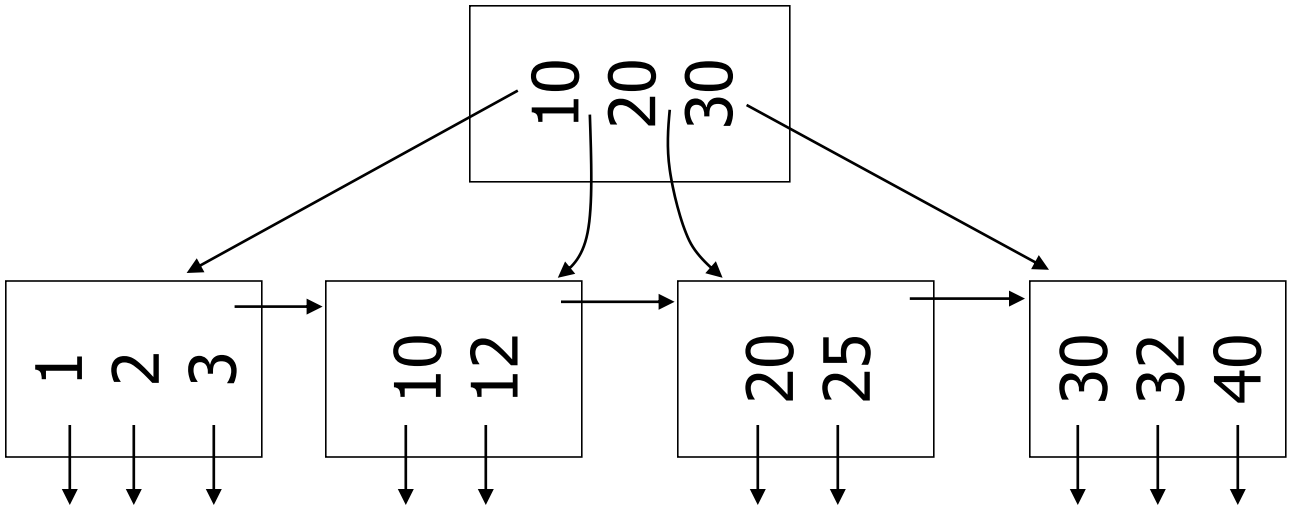
(c) Insert key = 160

n=3



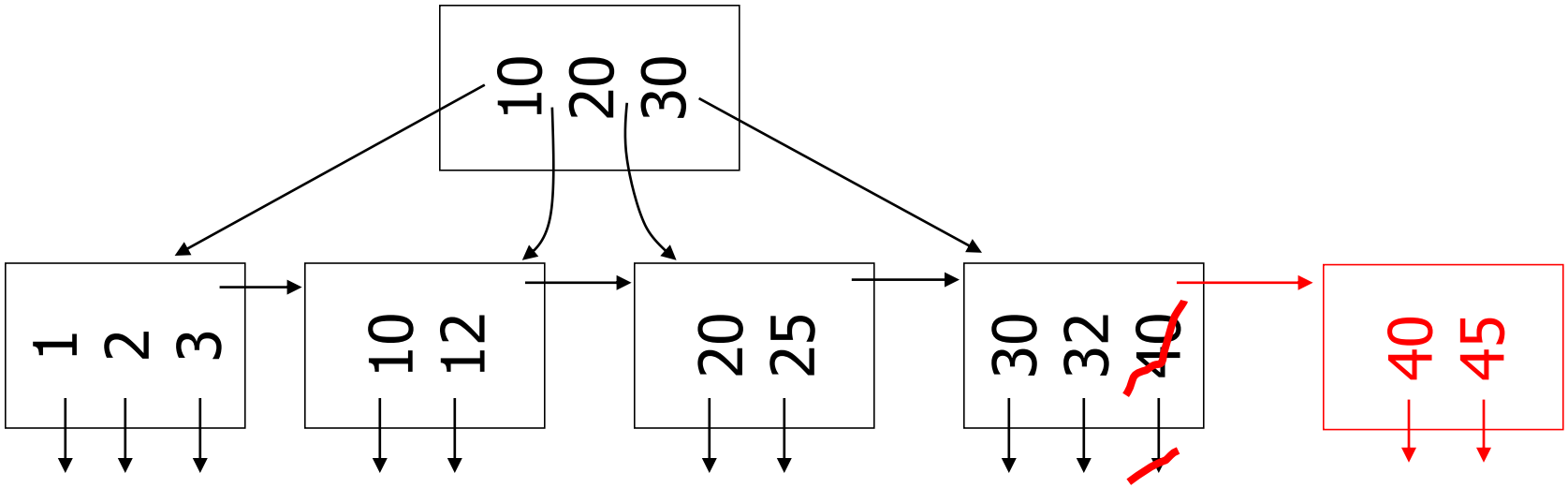
(d) New root, insert 45

n=3



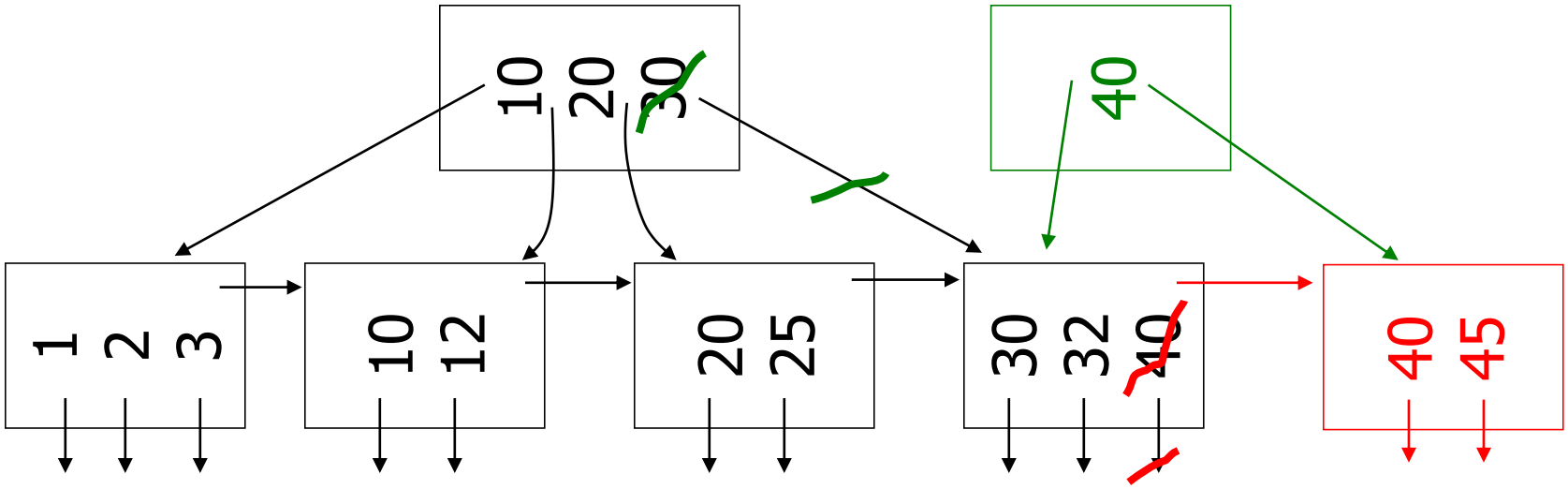
(d) New root, insert 45

n=3



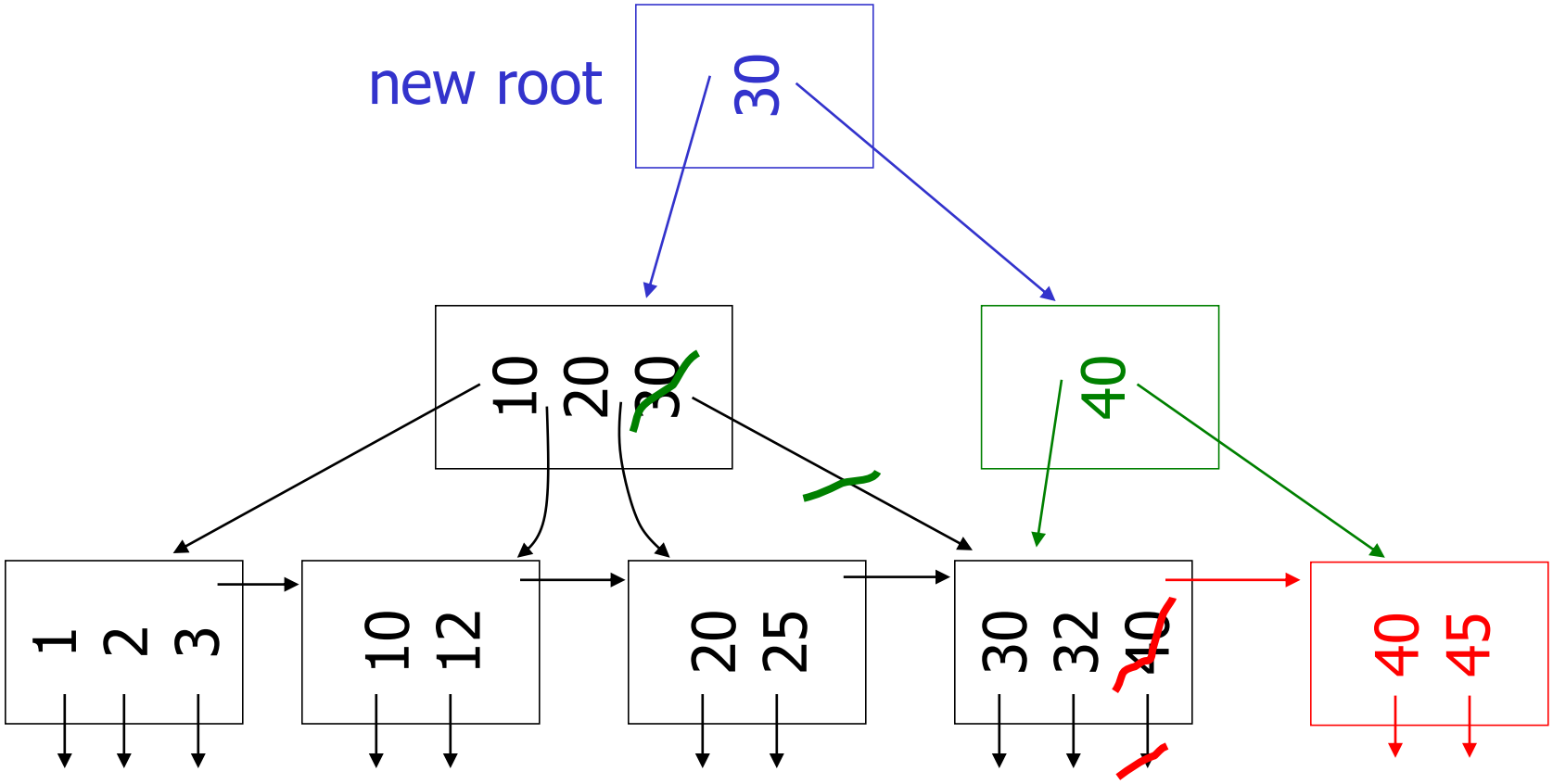
(d) New root, insert 45

n=3



(d) New root, insert 45

n=3



Deletion from B+tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

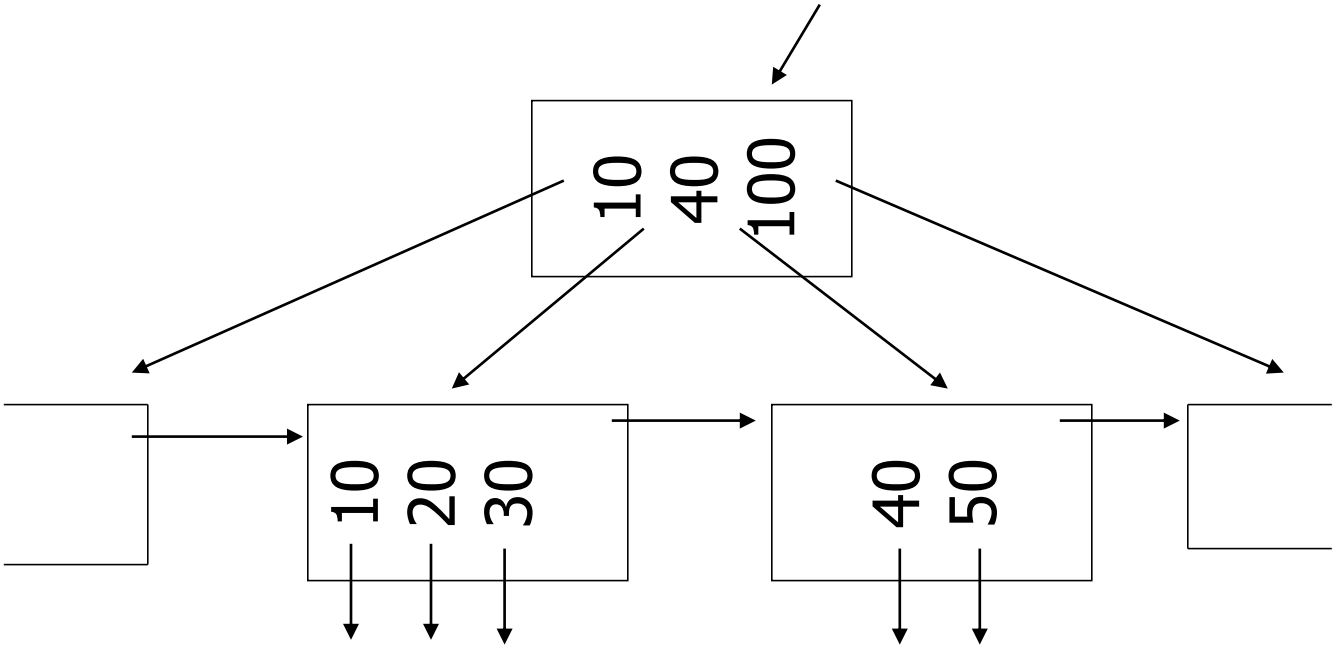
(c) Re-distribute keys

(d) Cases (b) or (c) at non-leaf

(b) Coalesce with sibling

– Delete 50

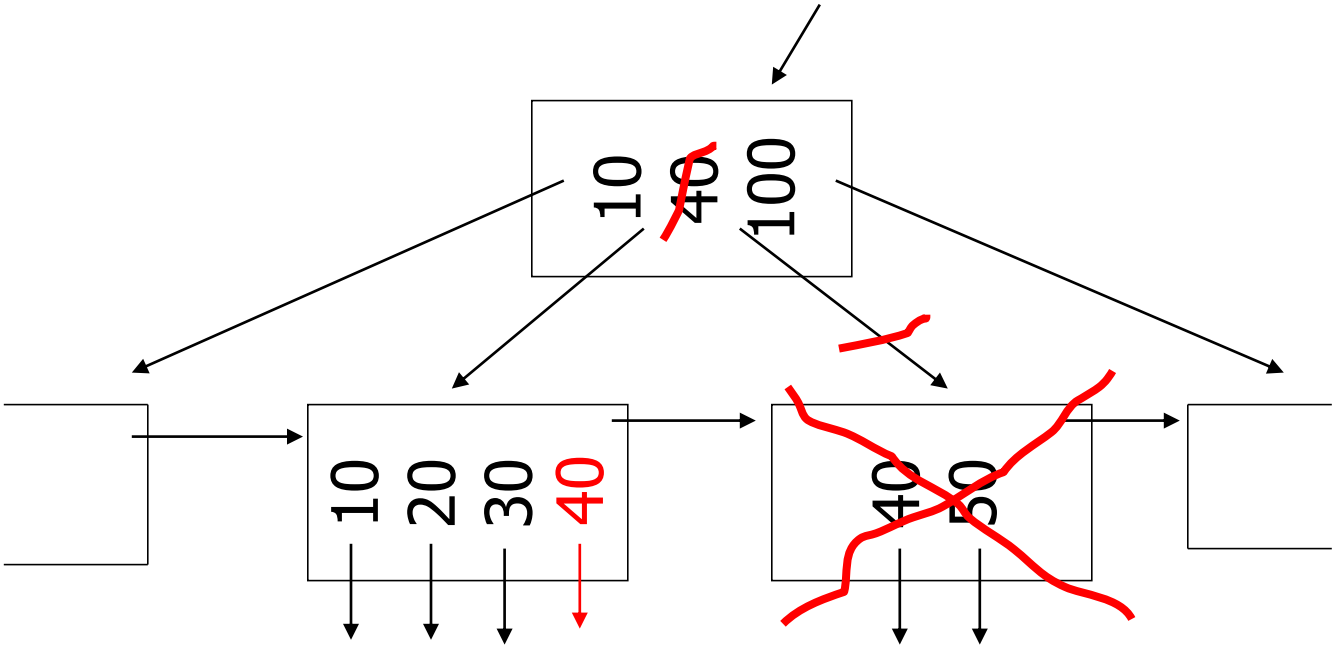
n=4



(b) Coalesce with sibling

– Delete 50

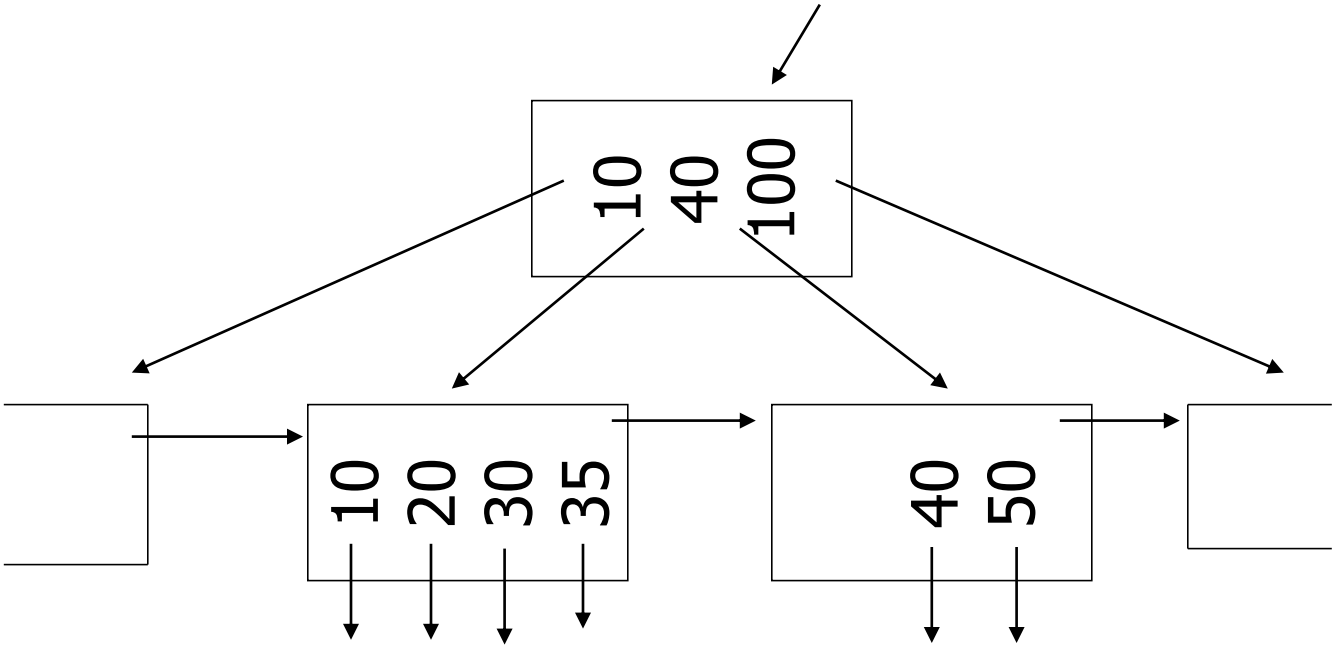
n=4



(c) Redistribute keys

– Delete 50

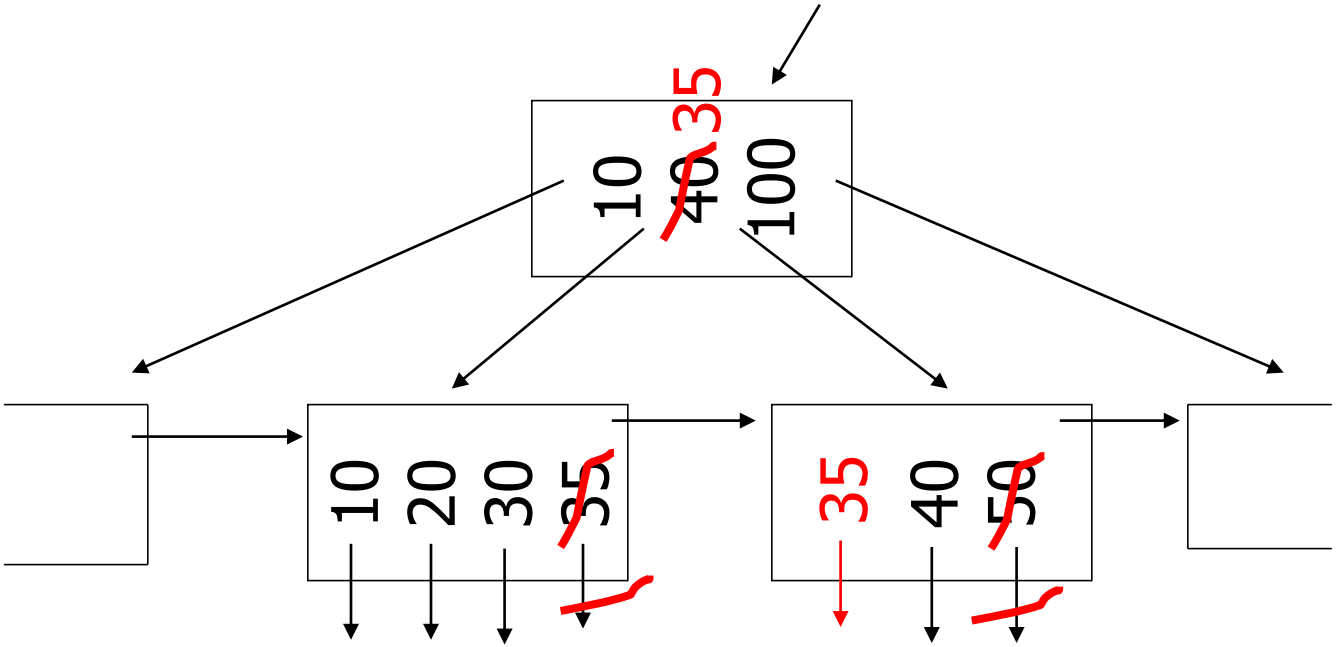
n=4



(c) Redistribute keys

– Delete 50

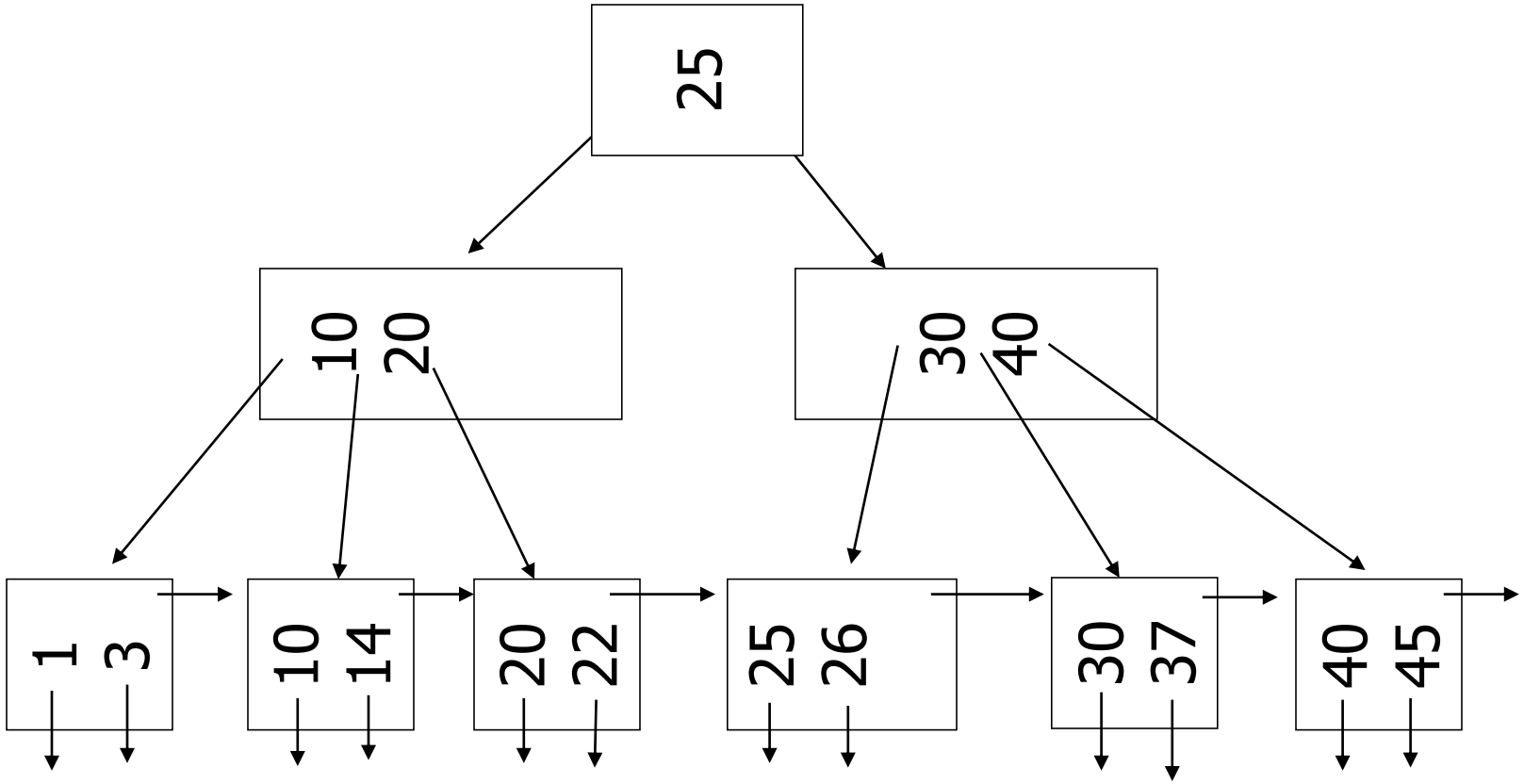
n=4



(d) Non-leaf coalesce

– Delete 37

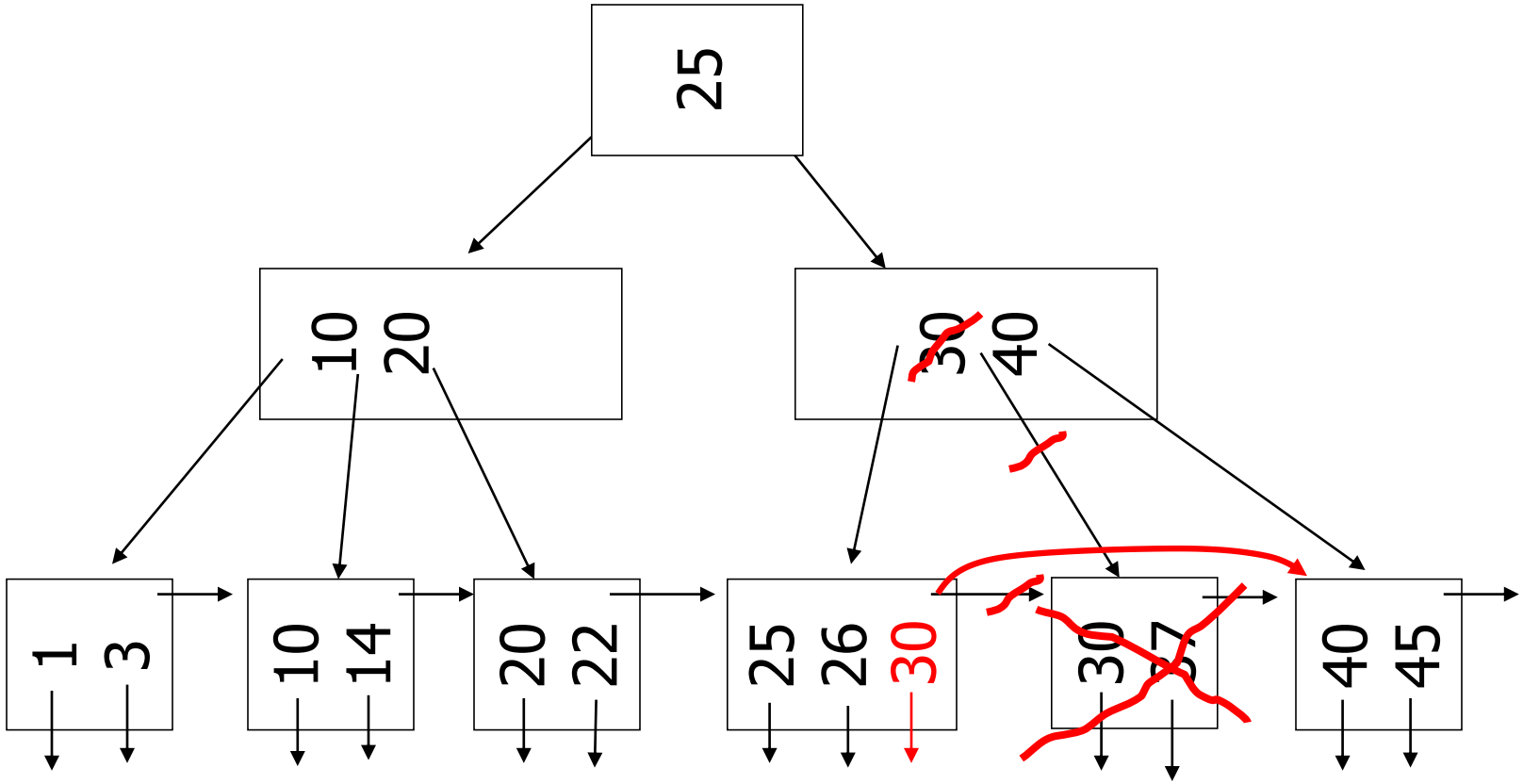
n=4



(d) Non-leaf coalesce

– Delete 37

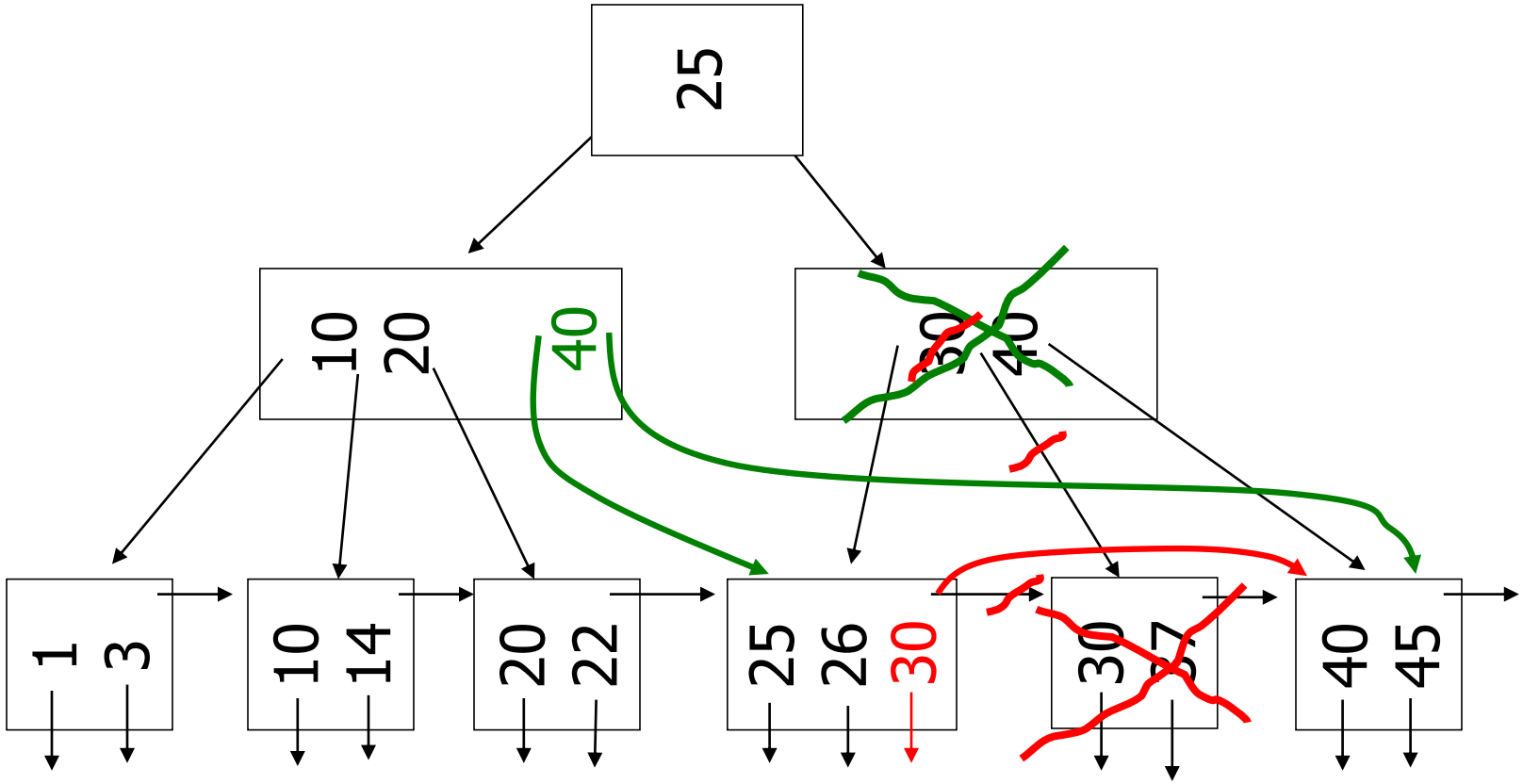
n=4



(d) Non-leaf coalesce

– Delete 37

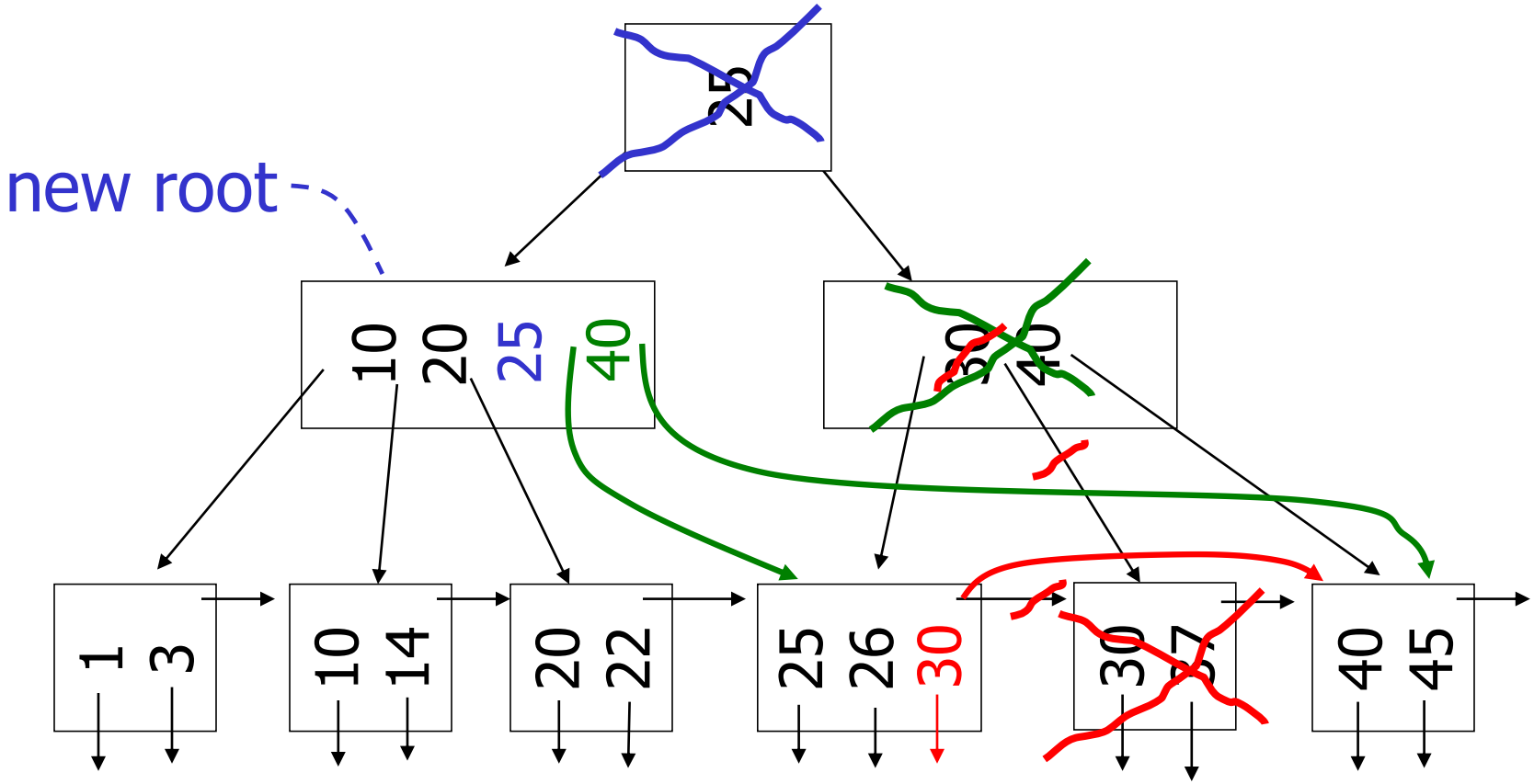
n=4



(d) Non-leaf coalesce

– Delete 37

n=4

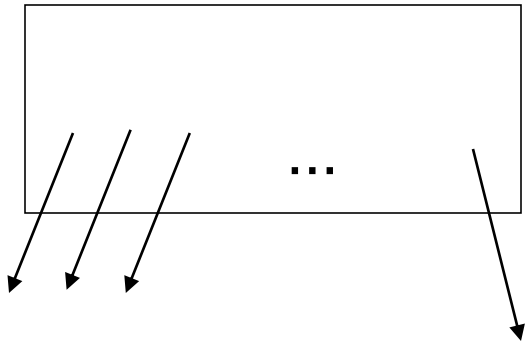


B+tree deletions in practice

- Often, coalescing is not implemented
  - Too hard and not worth it!

Interesting problem:

For B+tree, how large should *n* be?



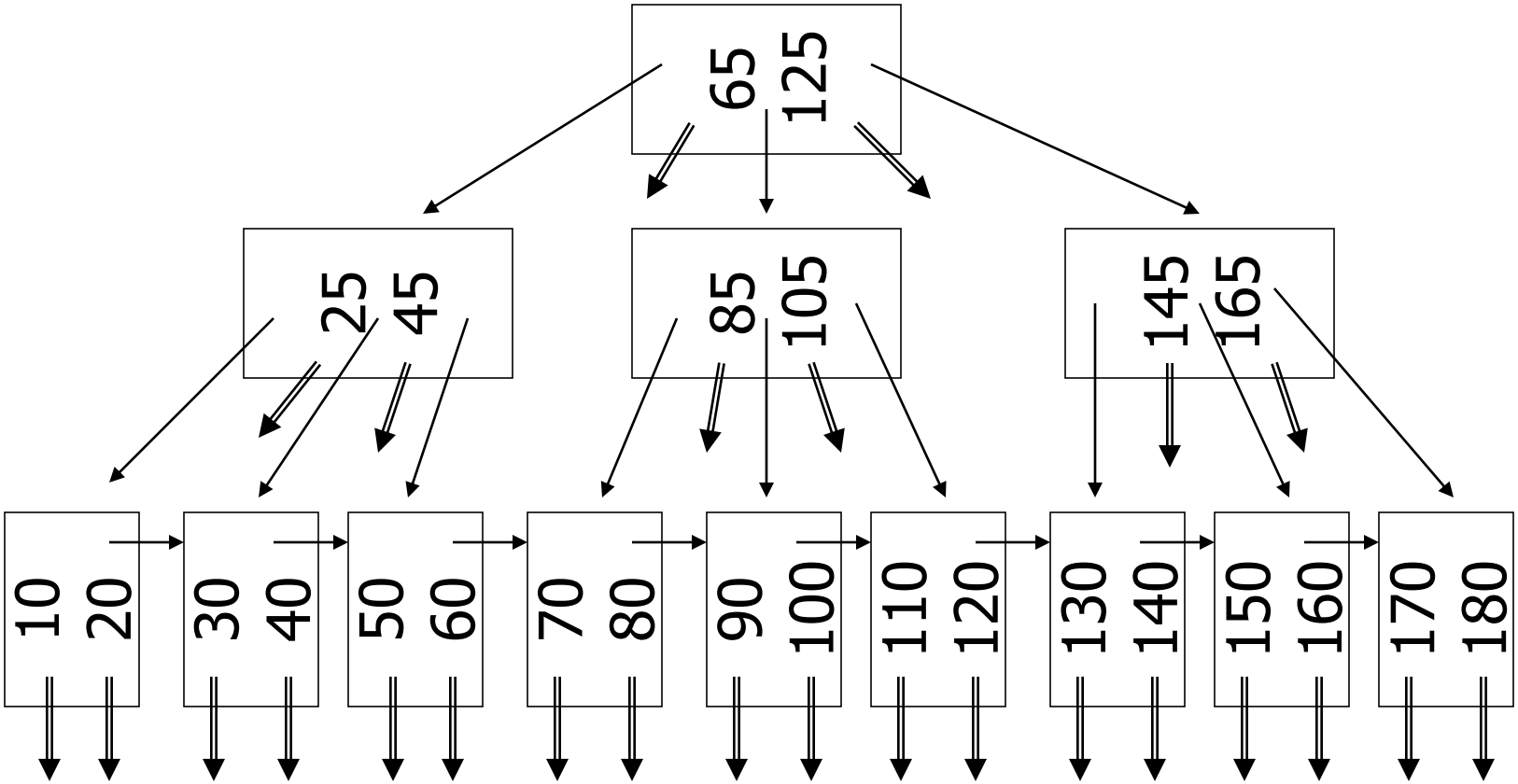
n is number of keys / node

Variation on B+tree: B-tree (no +)

- Idea:
  - Avoid duplicate keys
  - Have record pointers in non-leaf nodes

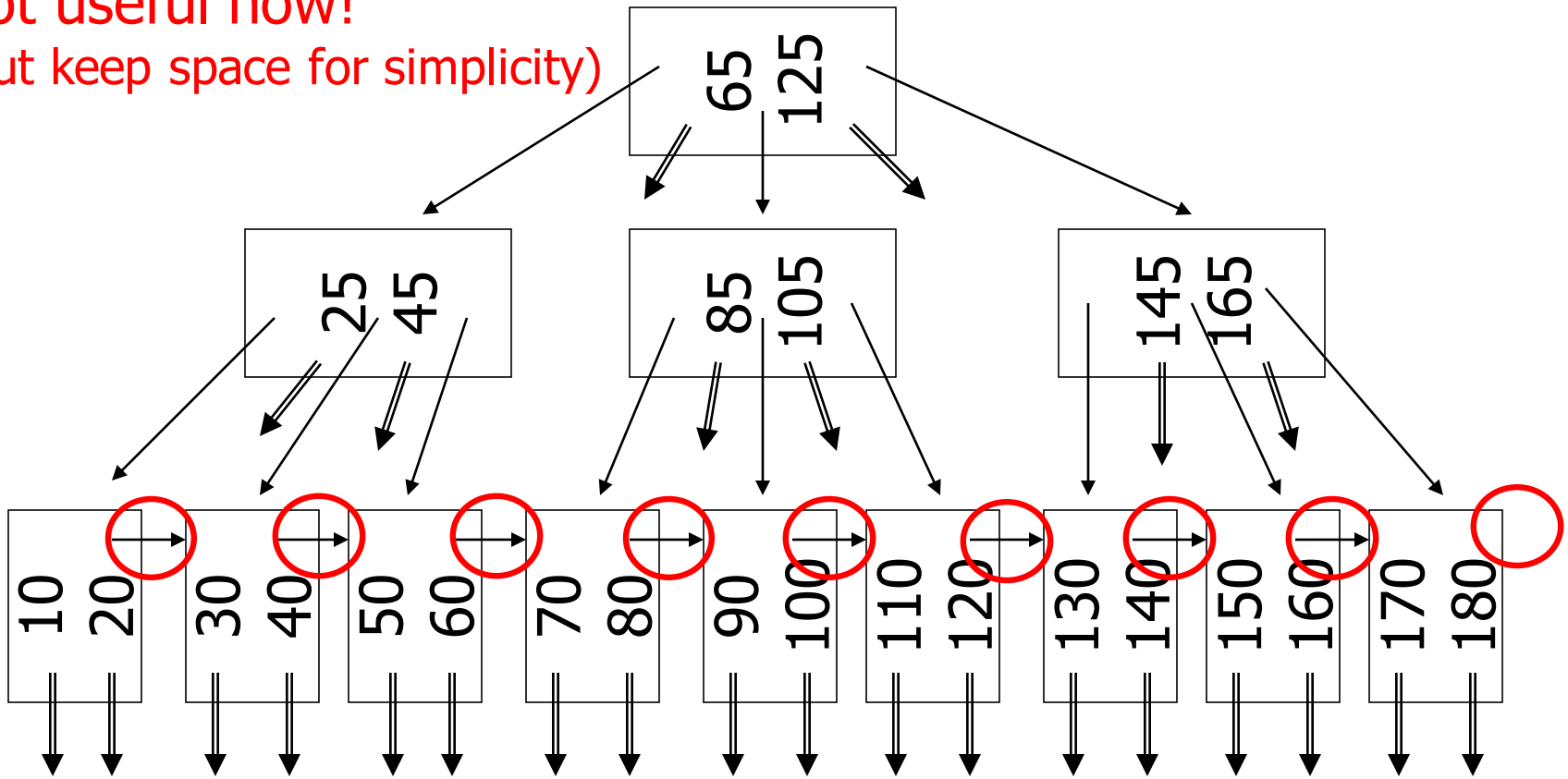
B-tree example

n=2



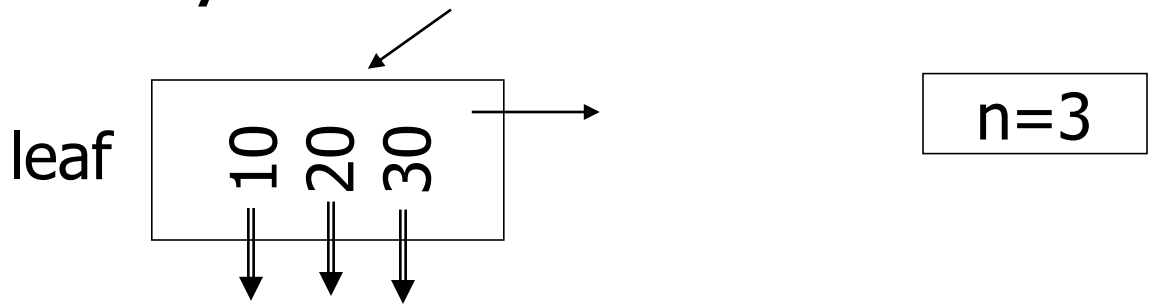
B-tree example n=2

- sequence pointers not useful now!
  (but keep space for simplicity)



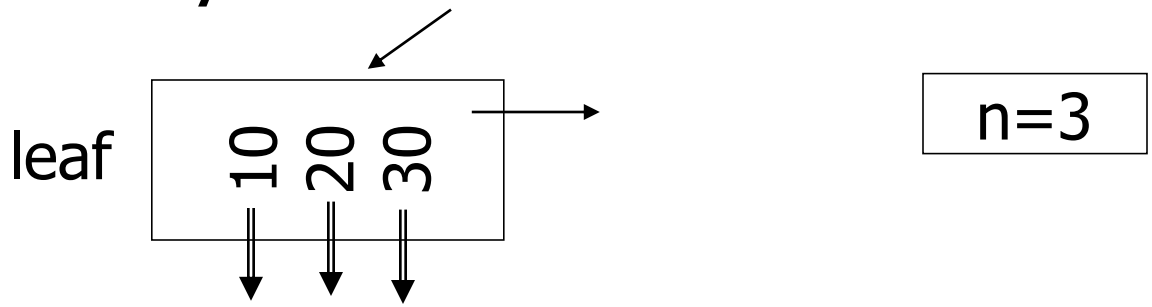
Note on inserts

- Say we insert key = 25

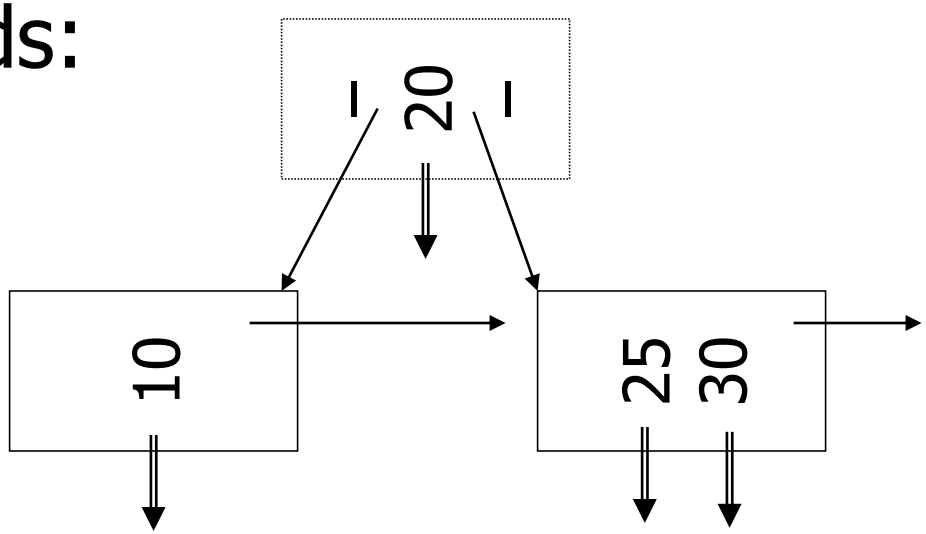


Note on inserts

- Say we insert key = 25



- Afterwards:



Outline/summary

- B trees
  - B+trees vs. B-trees
  - B+trees vs. indexed sequential
- Hashing schemes --> Next