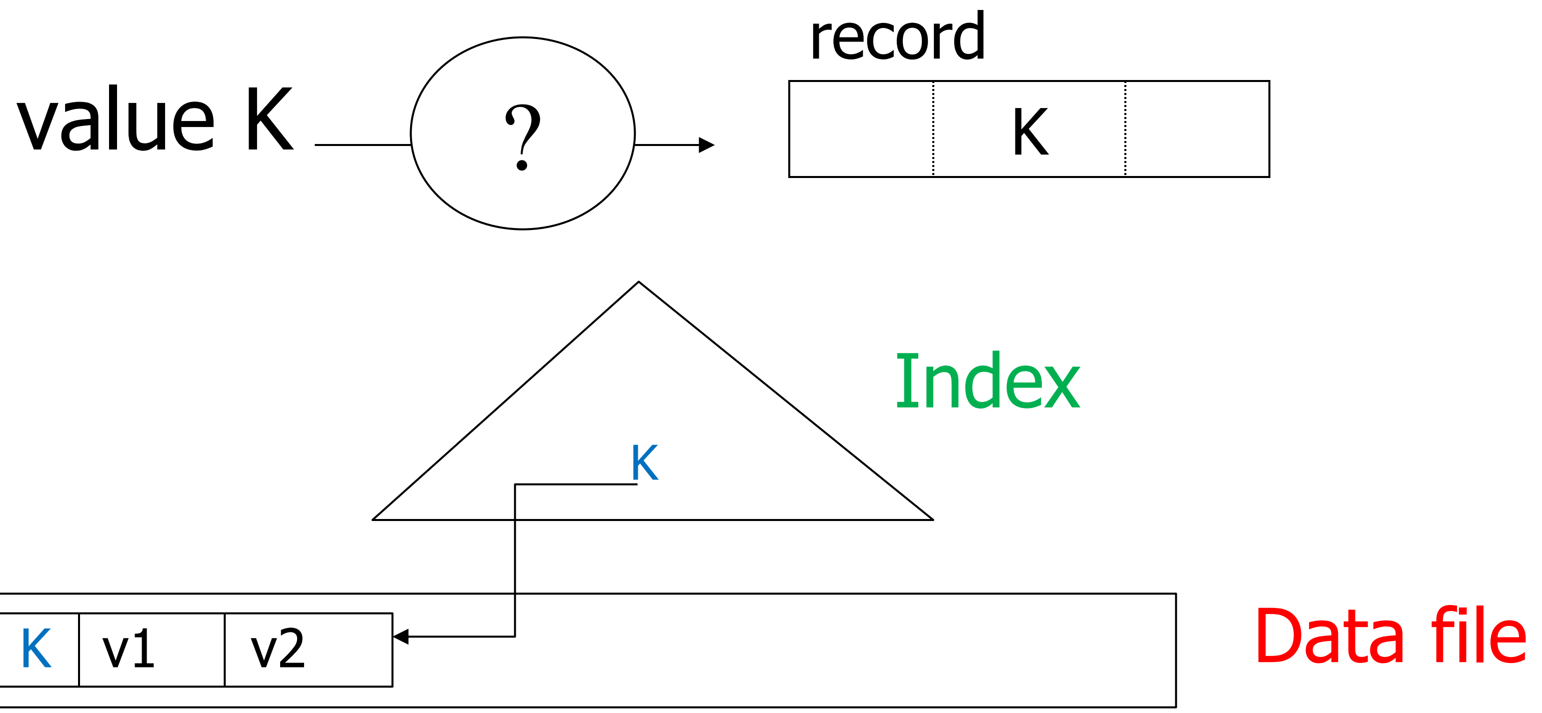
Indexing (B+ Trees)

based on lecture notes by Hector Garcia-Molina

Indexing



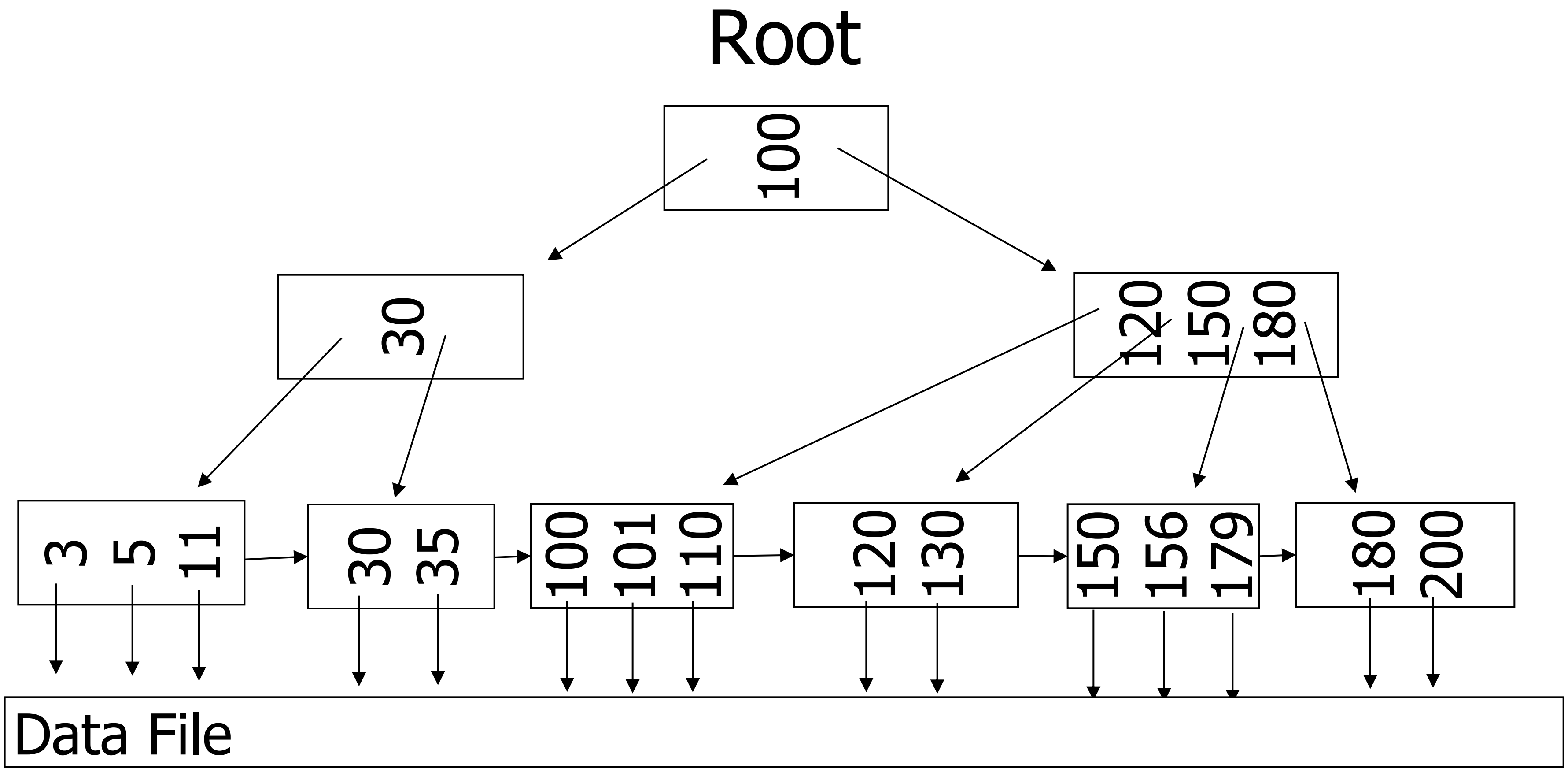
Index and Data File

- Index and Data File are separate data structures
- Index is much smaller than Data File. |Index| << |Data File|
- Index is stored in a collection of blocks in secondary memory
- Data File stored in a collection of blocks in secondary memory
- A key value K in the Index references a single or multiple records in the Data File with that key value K
- Records with the same K value are chained in the Data File (possibly in sequential blocks)
- Operations (search, insert, delete) on indexed data file require blocks from Index and Data File to be moved between primary and secondary memory
- Time complexity measured in terms of number of I/O operations

B+Tree Example

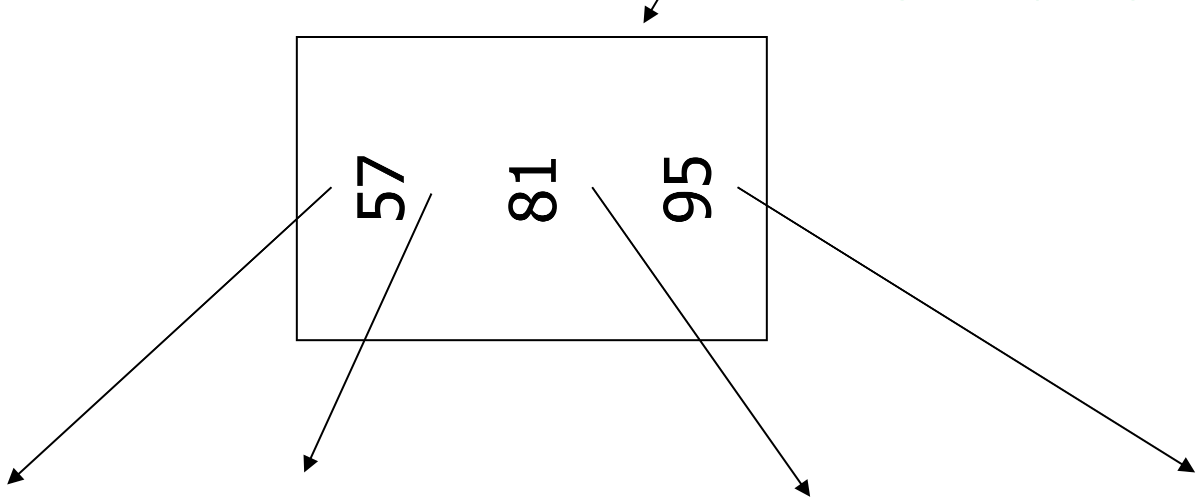
n=3

n order of B+ tree



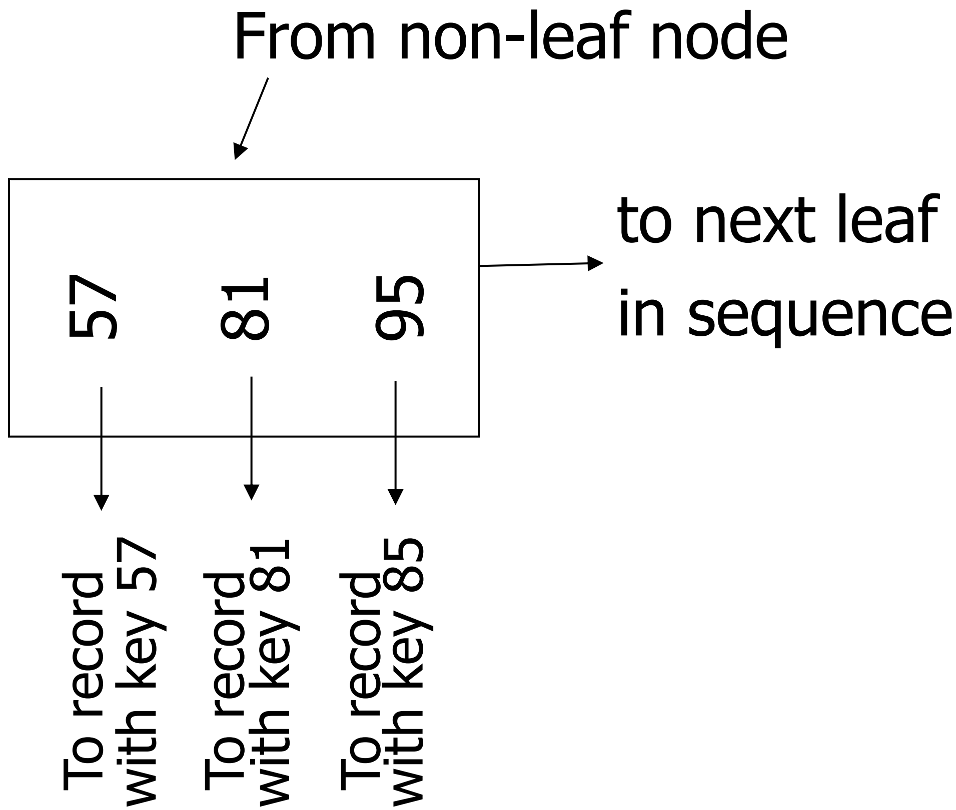
Sample non-leaf

Each node in a B+tree is a block
So n can be quite large



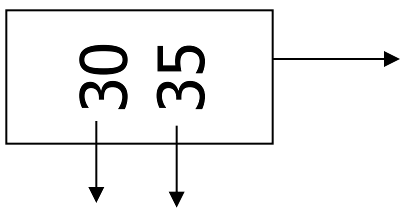
to keys	to keys	to keys	to keys
< 57	57≤ k<81	81≤k<95	≥95

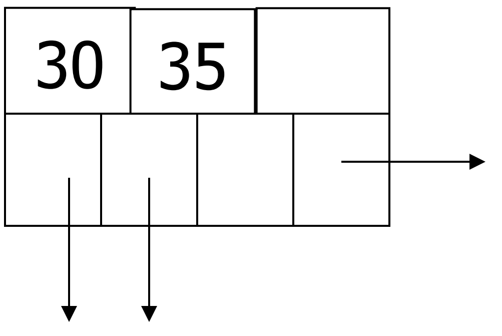
Sample leaf node:



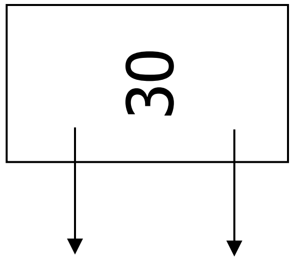
In textbook's notation n=3

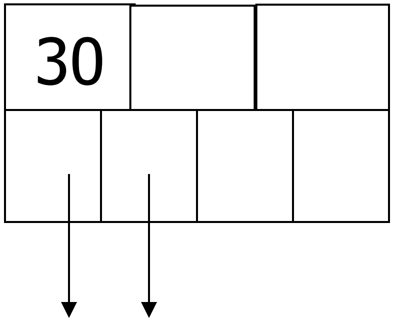
Leaf:





Non-leaf:





Size of nodes: $\left\{ \begin{array}{l} \text{n+1 pointers} \\ \text{n keys} \end{array} \right.$ (fixed)

Observe that a pointer is a block address

$$(n+1) * |blockaddress| + n * |key| \leq blocksize$$

$$n \leq \frac{blocksize - |blockaddress|}{|blockaddress| + |key|} \leq \frac{blocksize}{|blockaddress| + |key|}$$

Example: determination of n

- blocksize = 4096 bytes
- |blockaddress| = 8 bytes
- |key| = 9 bytes
- $n \leq \frac{blocksize - |blockaddress|}{|blockaddress| + |key|}$
- Thus n is maximally 240
- |blockaddress| = 8 bytes permits $2^{32} =$ 4,294,967,296 blocks to be referenced
- |blockaddress| = 10 bytes permits 1 trillion blocks to be referenced

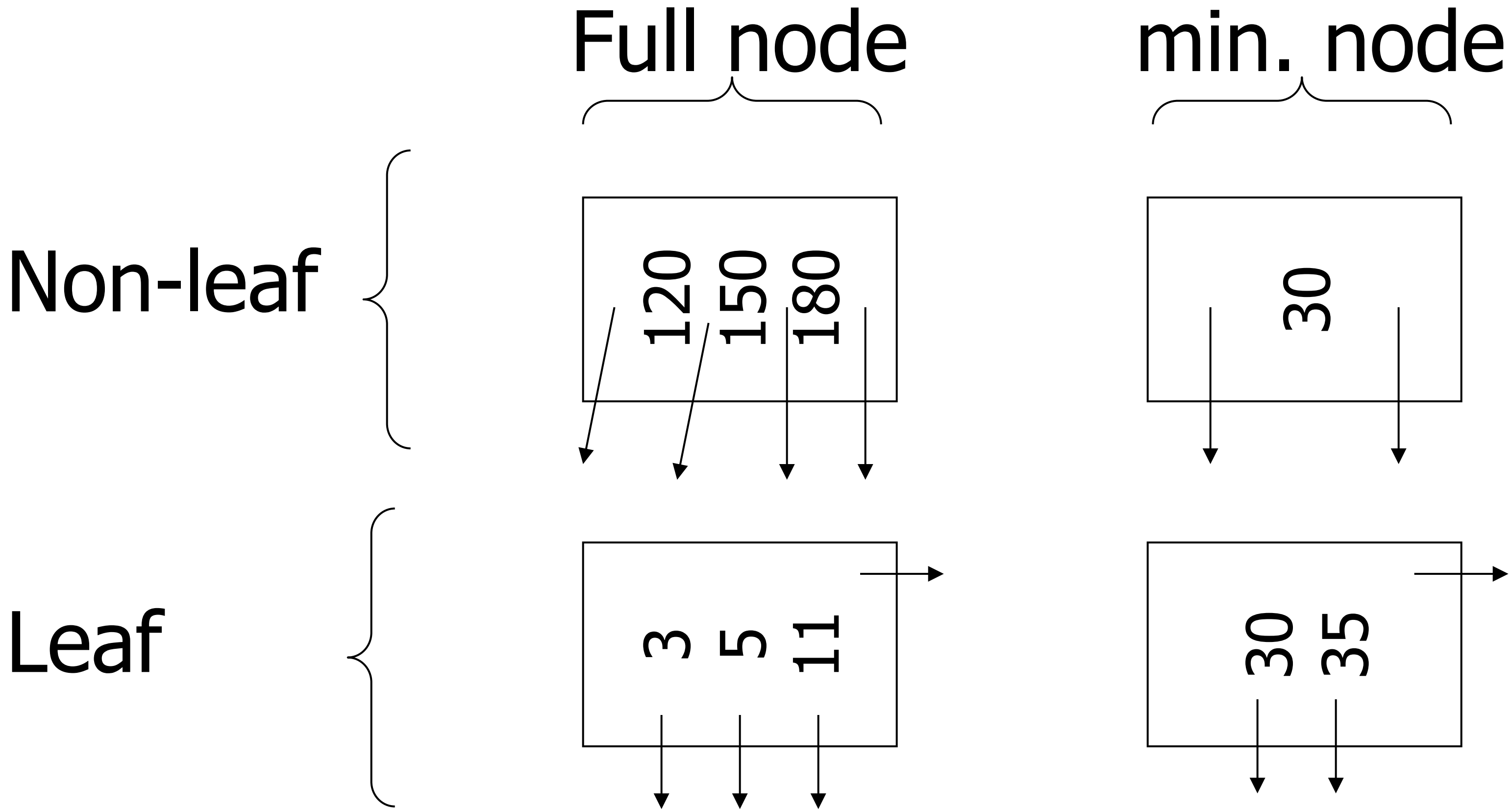
Don't want nodes to be too empty

- Use at least

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

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

n=3



<u>B+tree rules</u> tree of order *n*

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

(2) Pointers in leaves point to records
except for “next leaf pointer”

(3) Root must have at least one key
and two pointers

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

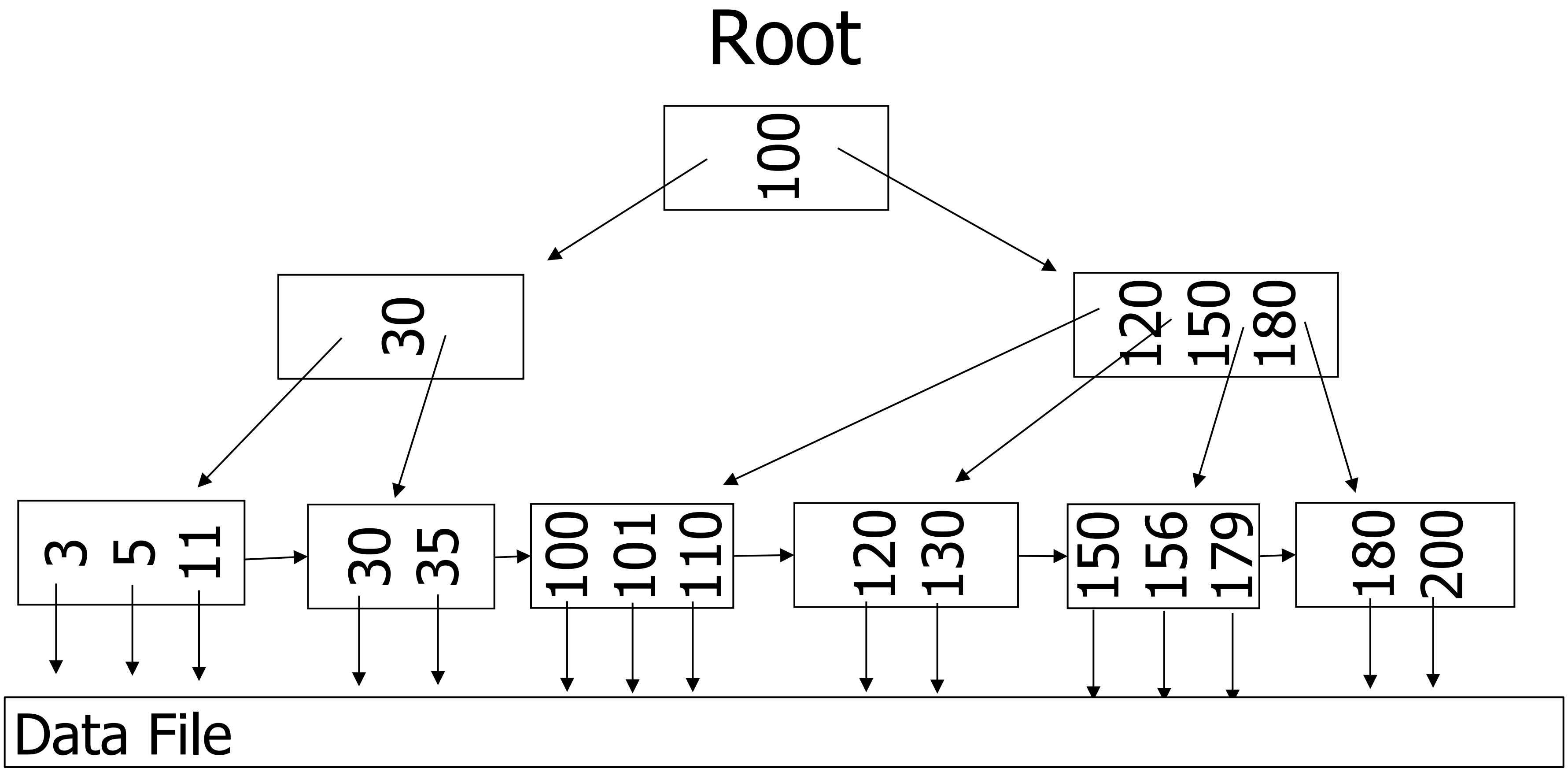
	Max ptrs	Max keys	Min ptrs	Min keys
Non-leaf (non-root)	n+1	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Leaf (non-root)	n+1	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	n+1	n	2	1

This slide assume that the tree has at least two levels

B+Tree Search

n=3

n order of B+ tree



Insert into B+tree

(a) simple case
- space available in leaf

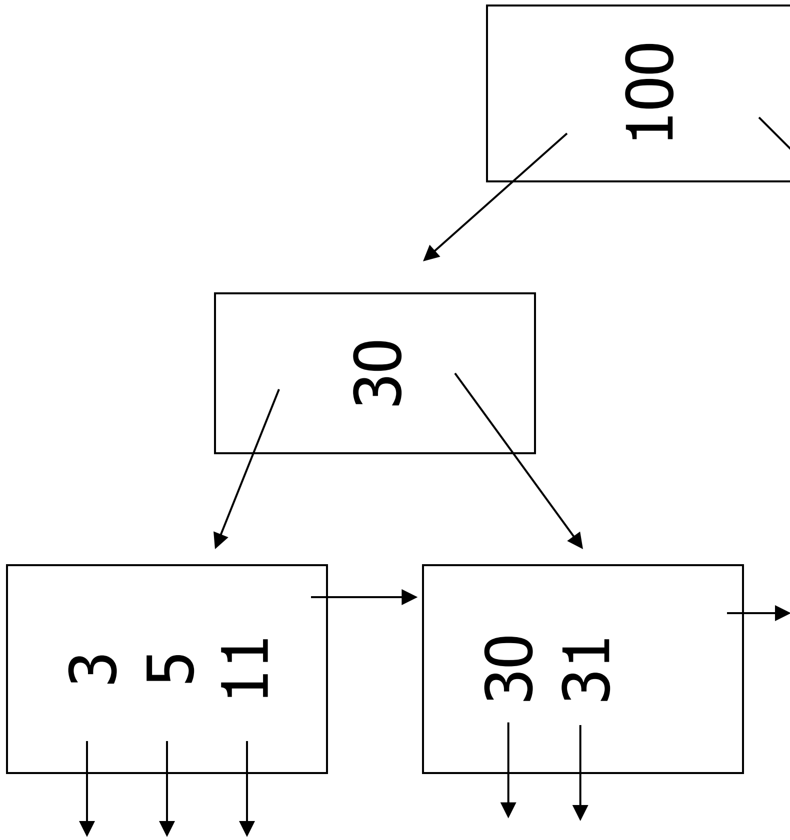
(b) leaf overflow

(c) non-leaf overflow

(d) new root

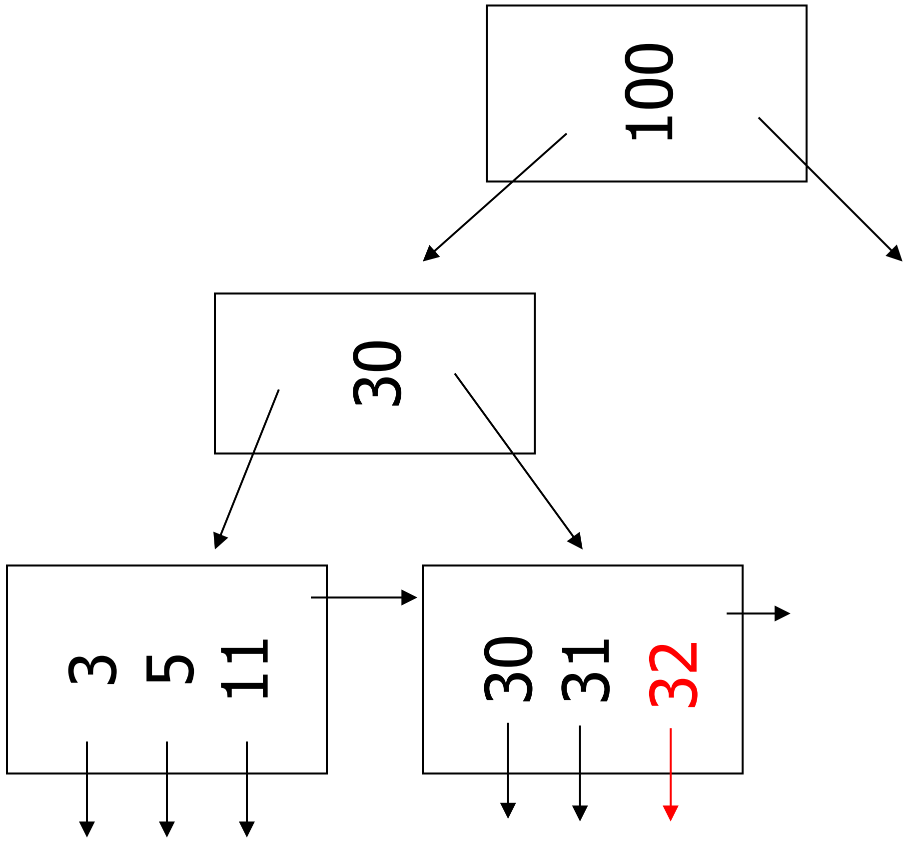
(a) Insert key = 32

n=3



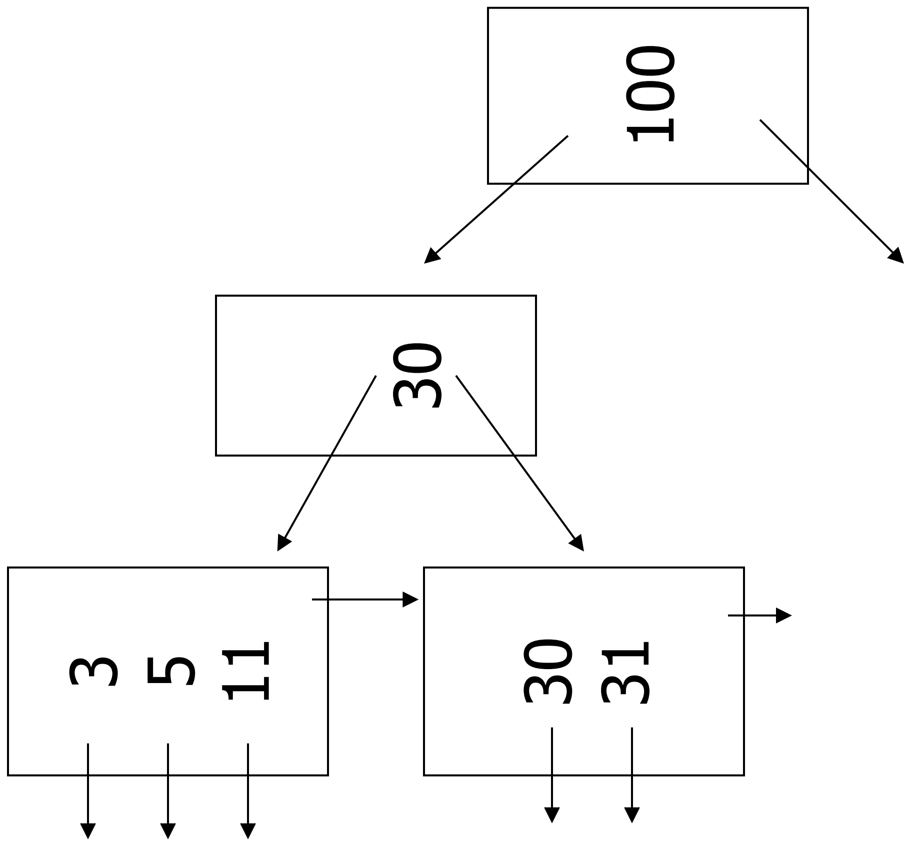
(a) Insert key = 32

$\boxed{n=3}$



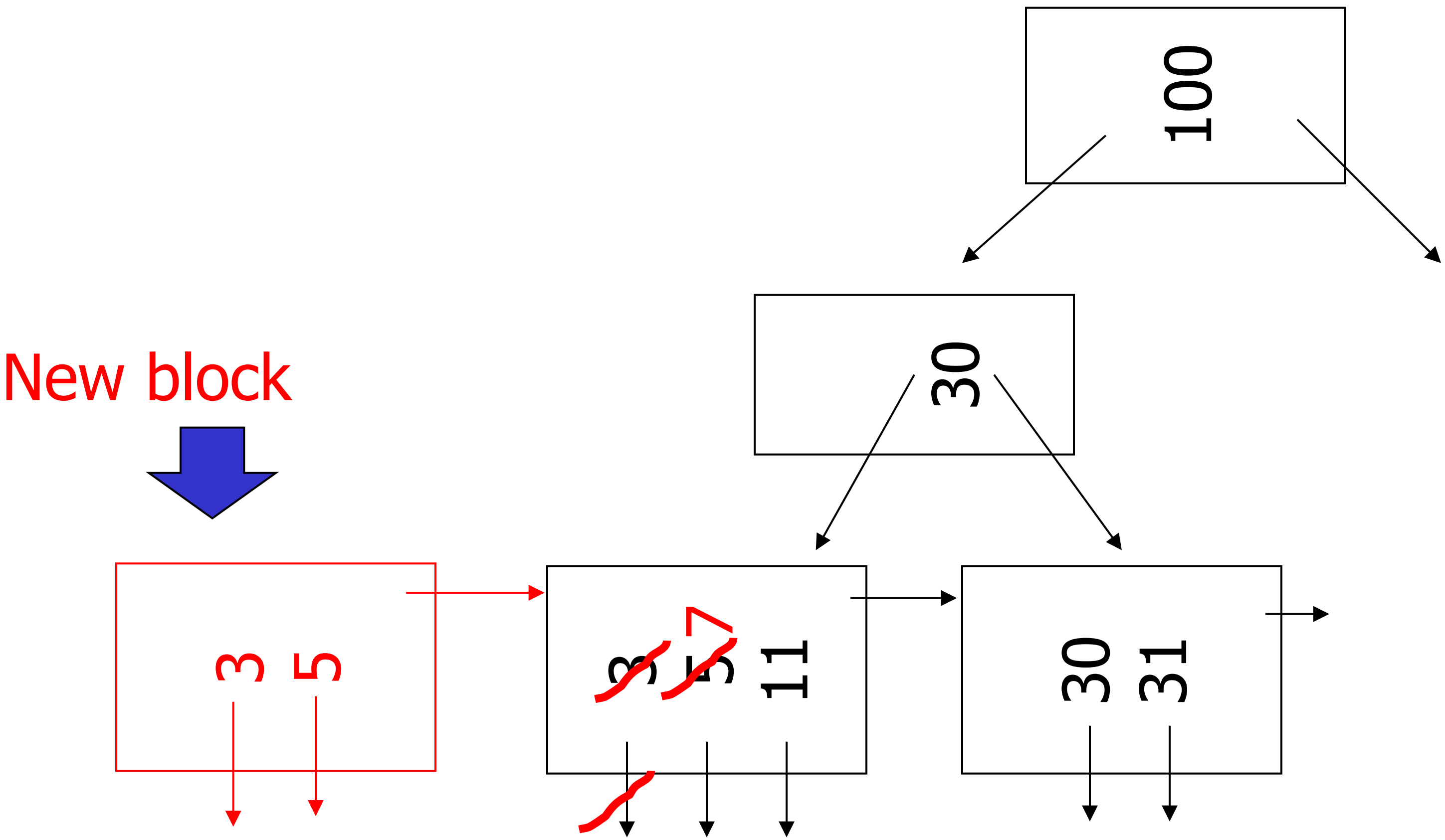
(a) Insert key = 7

n=3



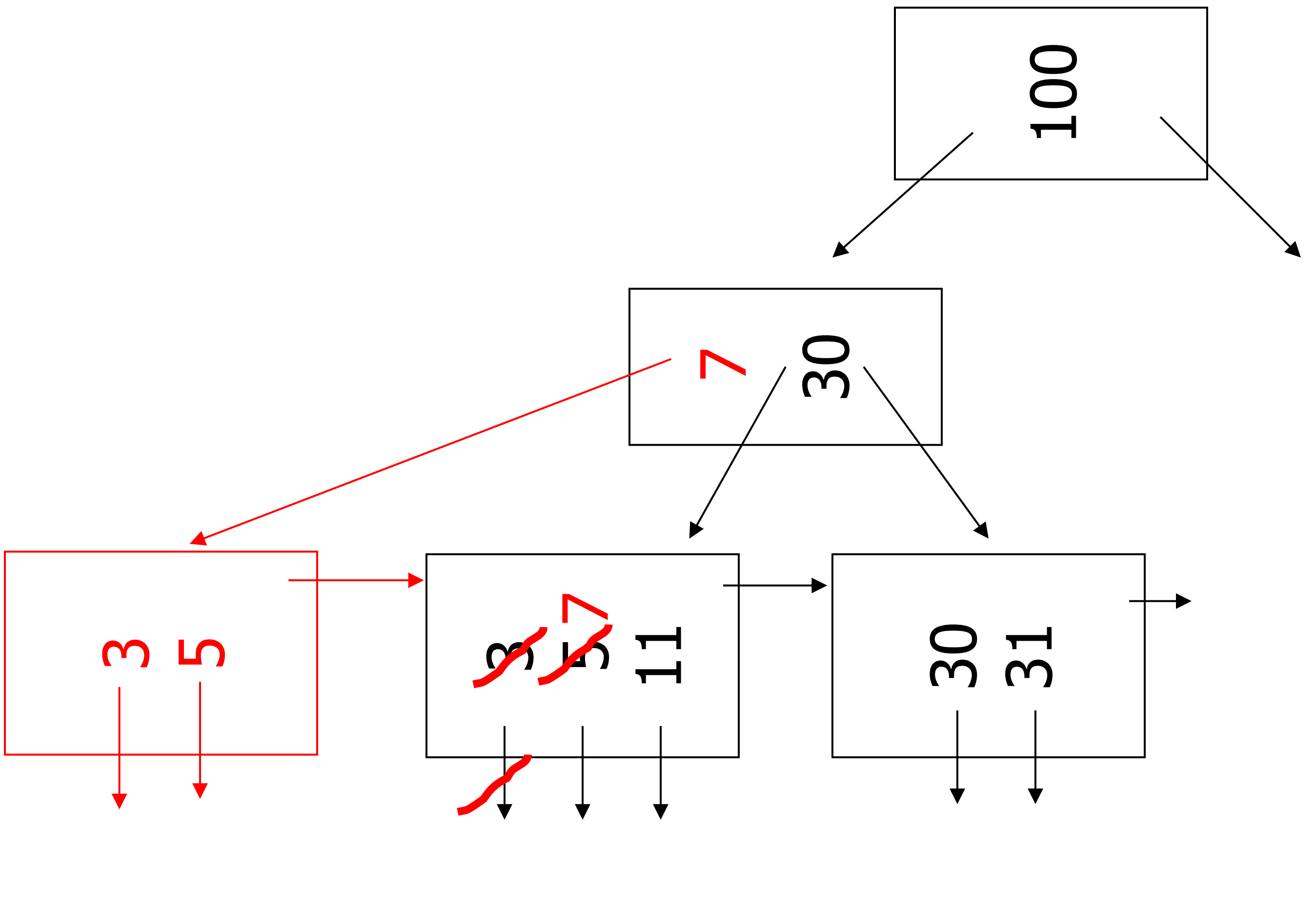
(a) Insert key = 7

n=3



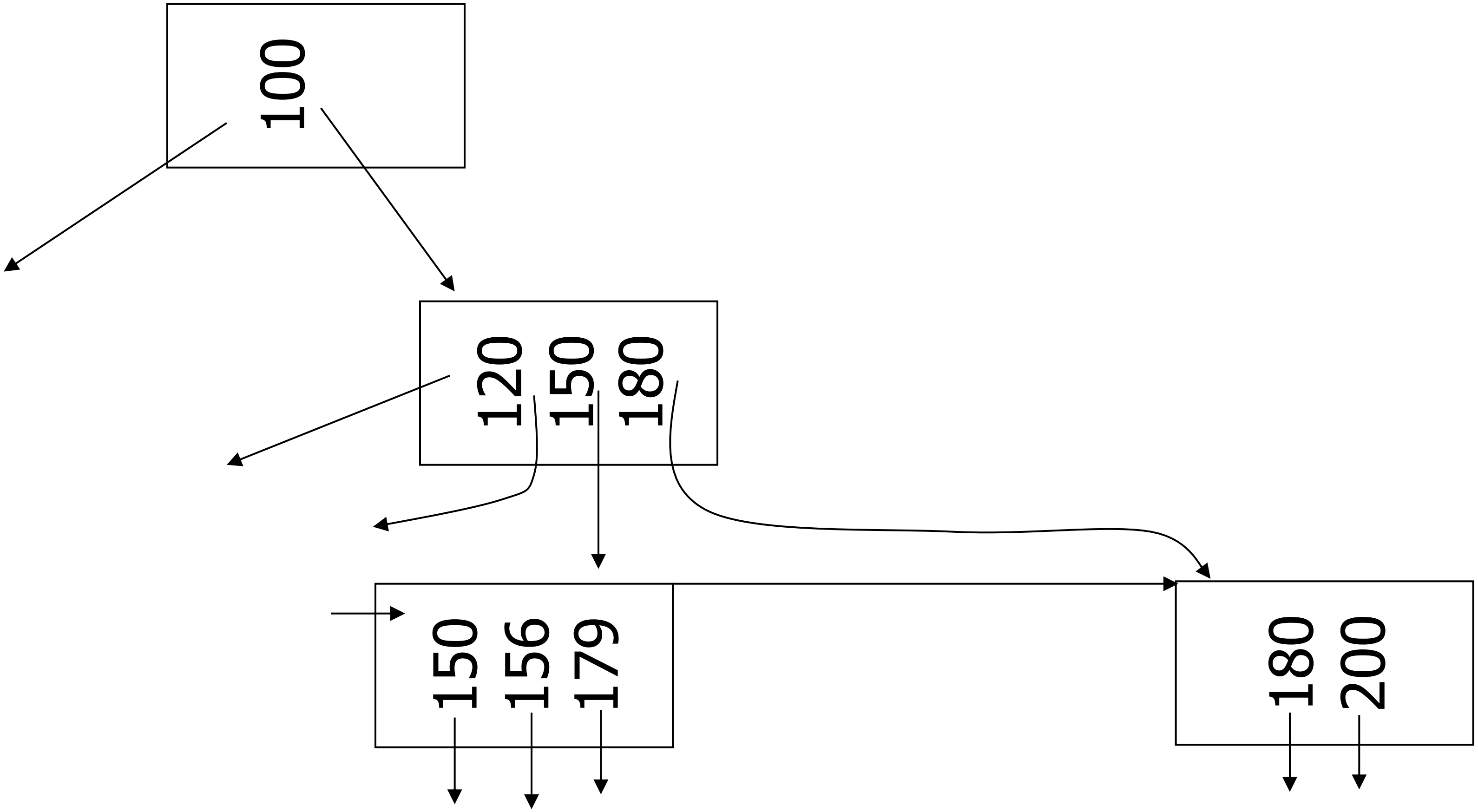
(a) Insert key = 7

n=3



(c) Insert key = 160

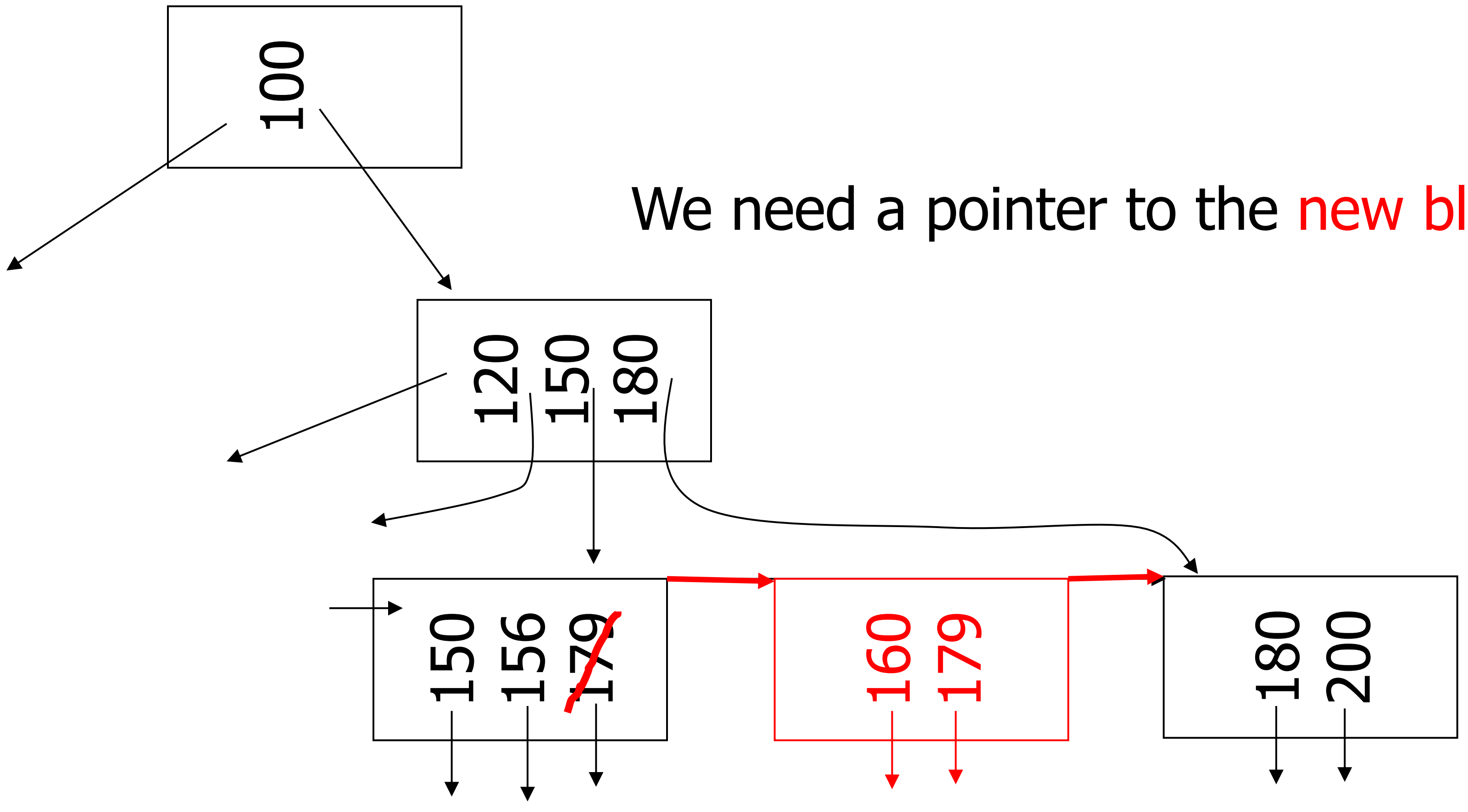
n=3



(c) Insert key = 160

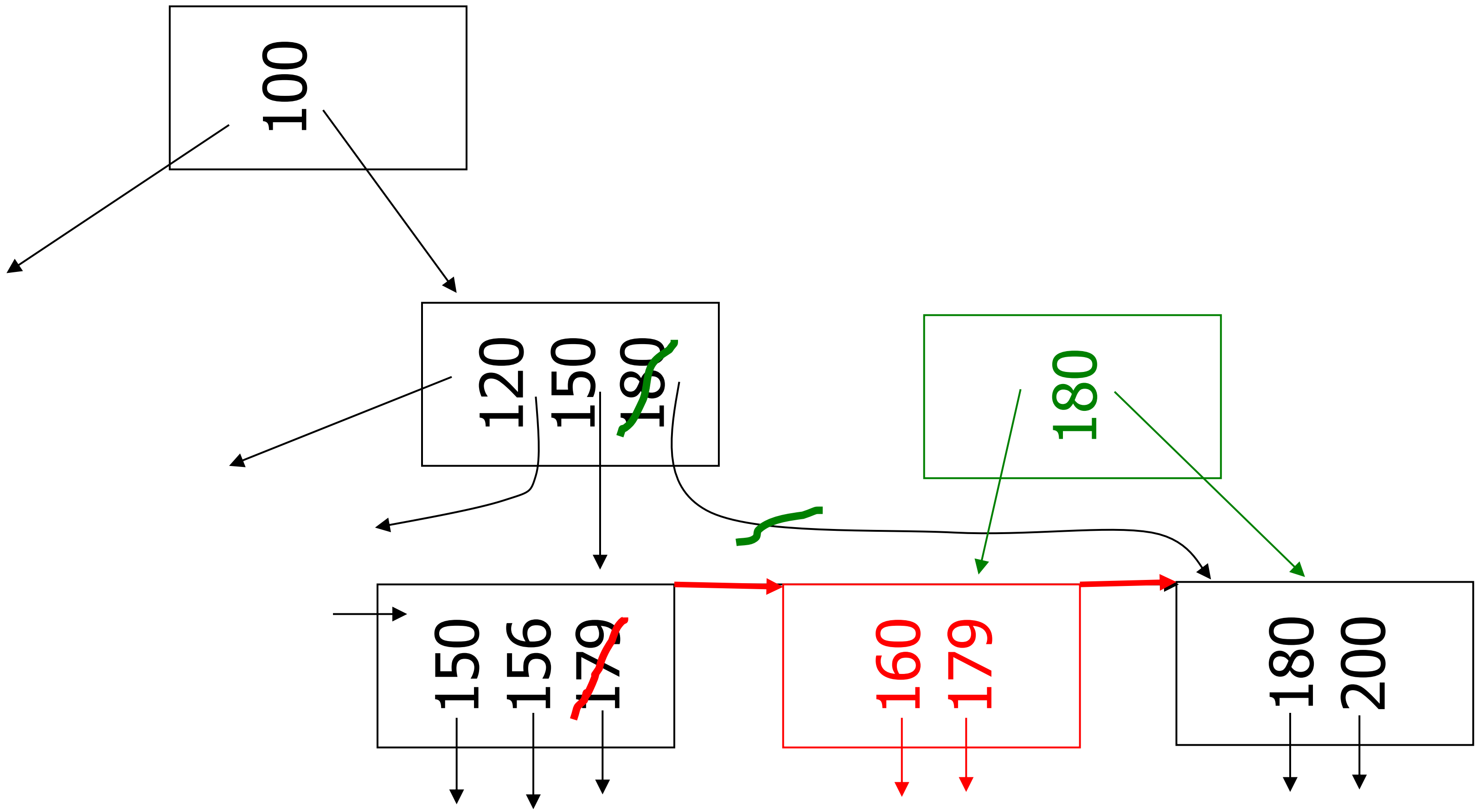
n=3

We need a pointer to the new block!



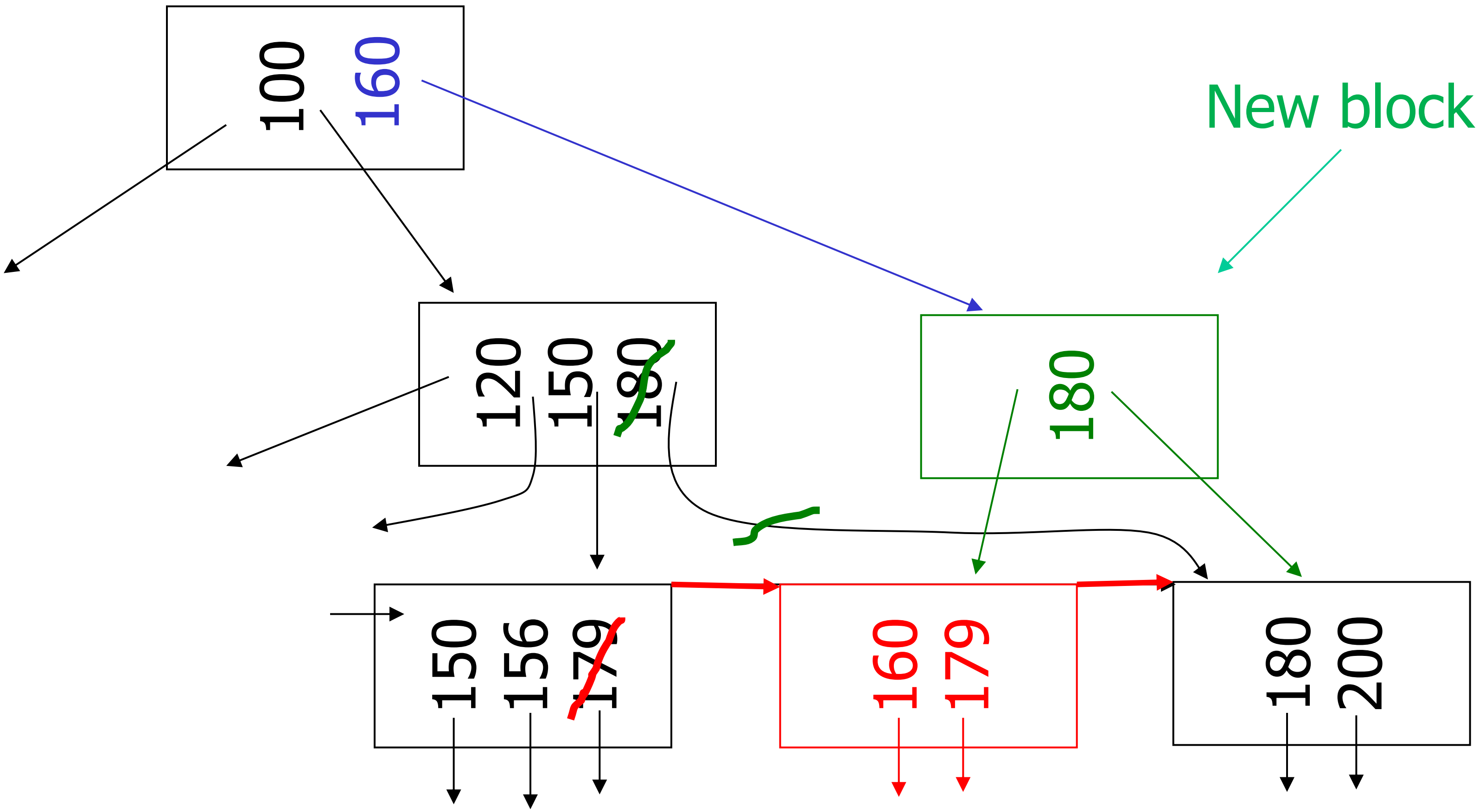
(c) Insert key = 160

n=3



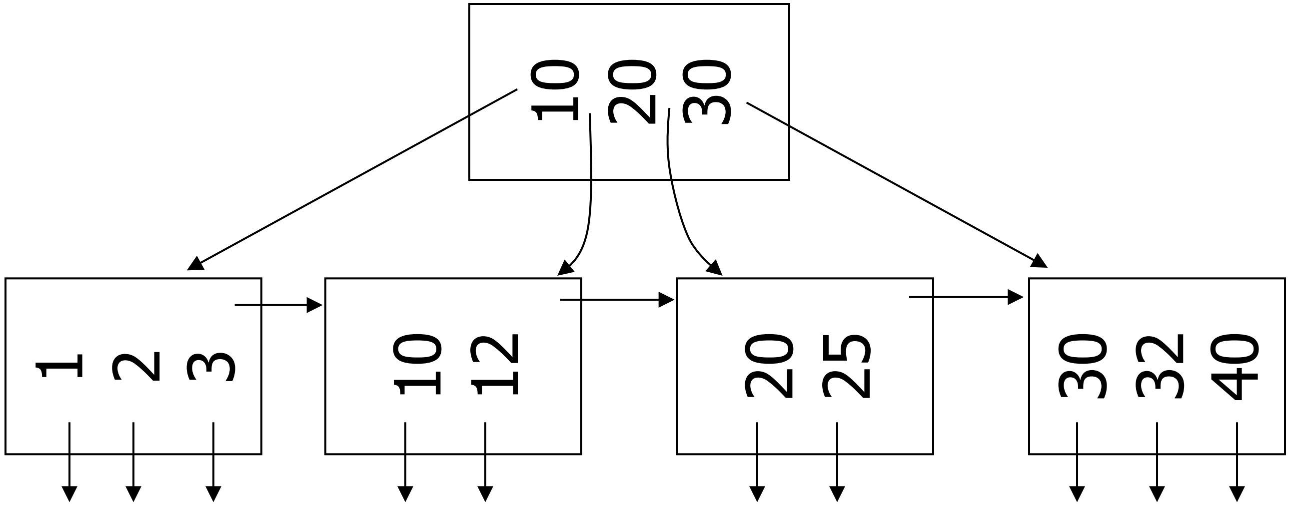
(c) Insert key = 160

n=3



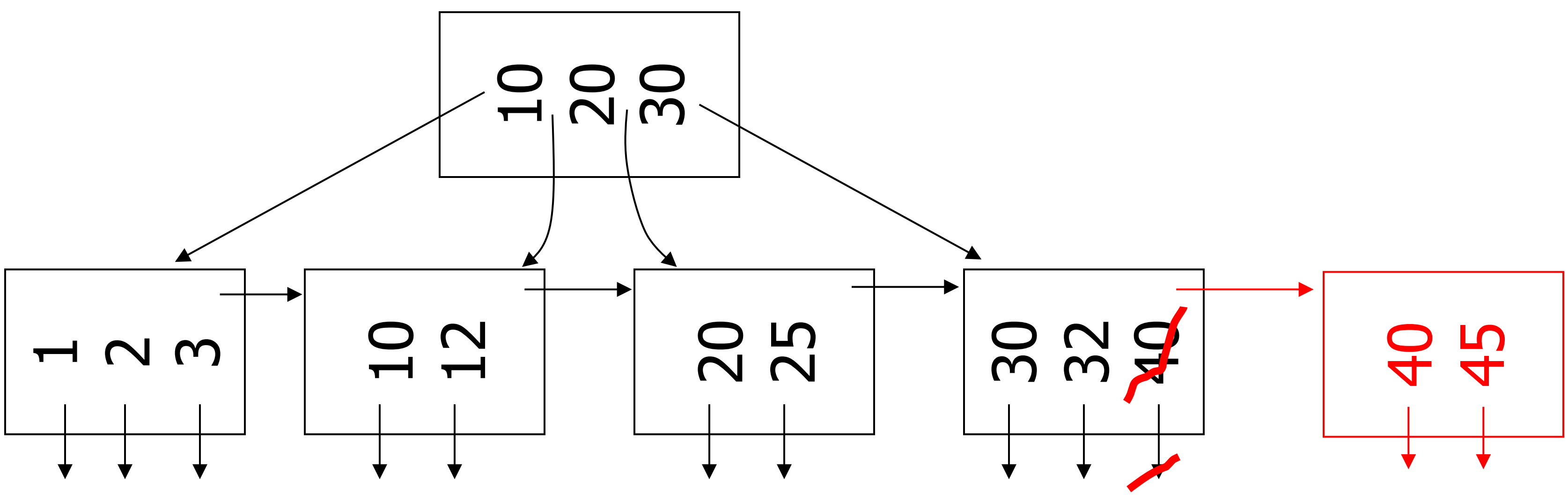
(d) New root, insert 45

n=3



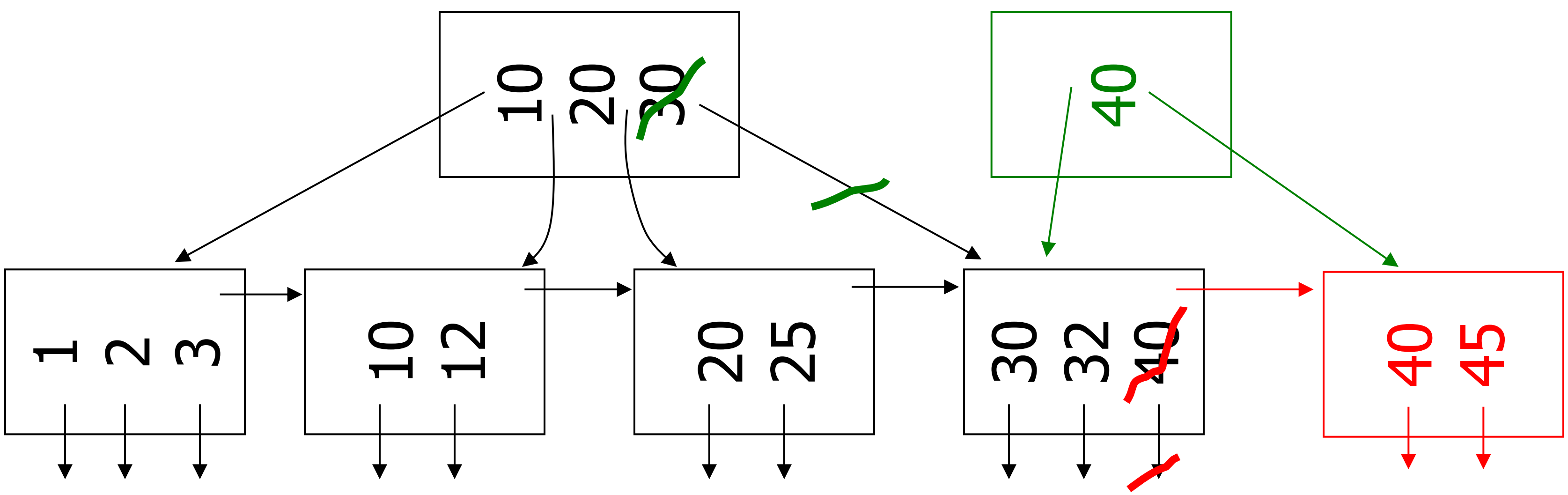
(d) New root, insert 45

n=3



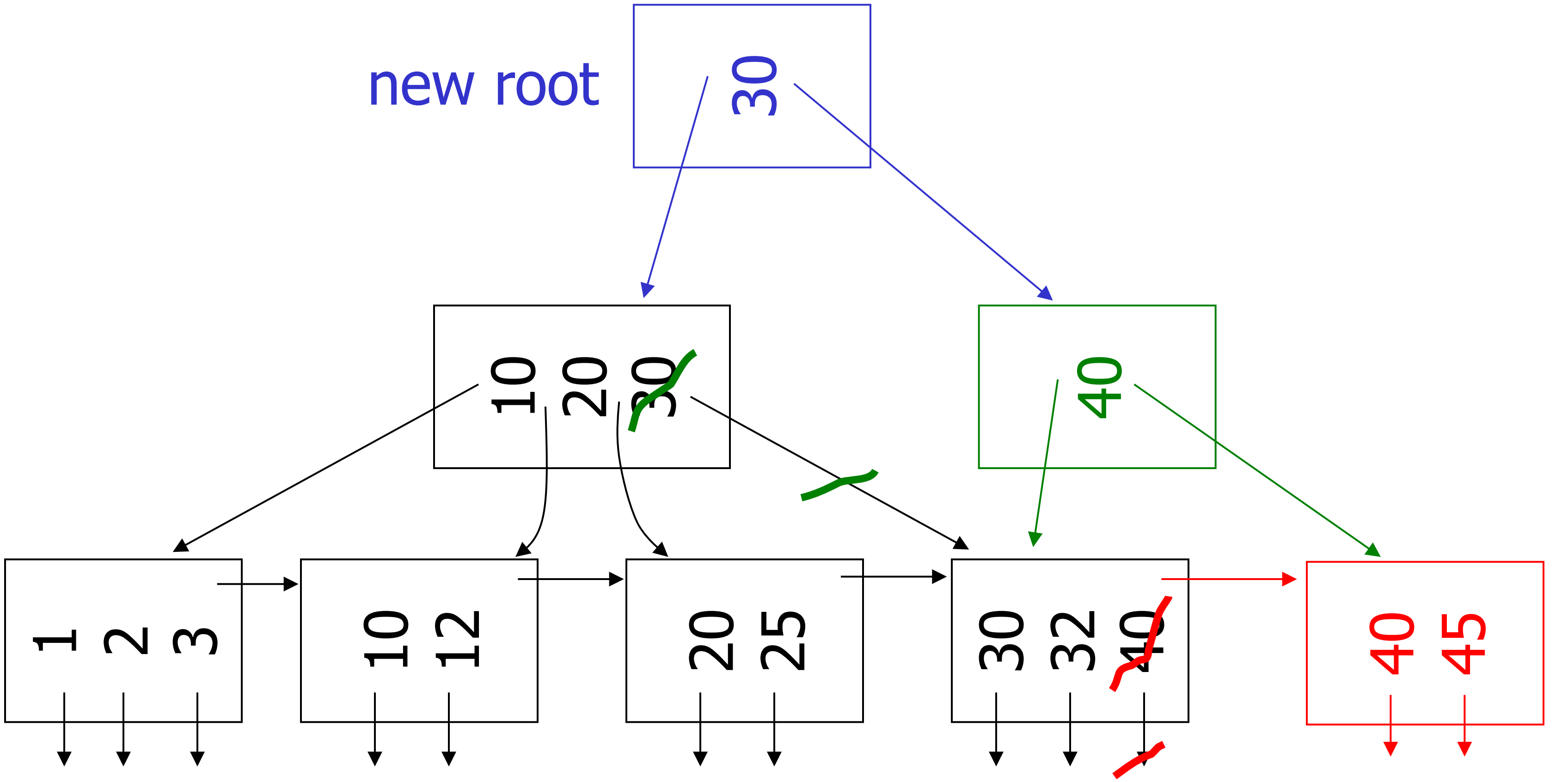
(d) New root, insert 45

n=3



(d) New root, insert 45

n=3



Deletion from B+tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

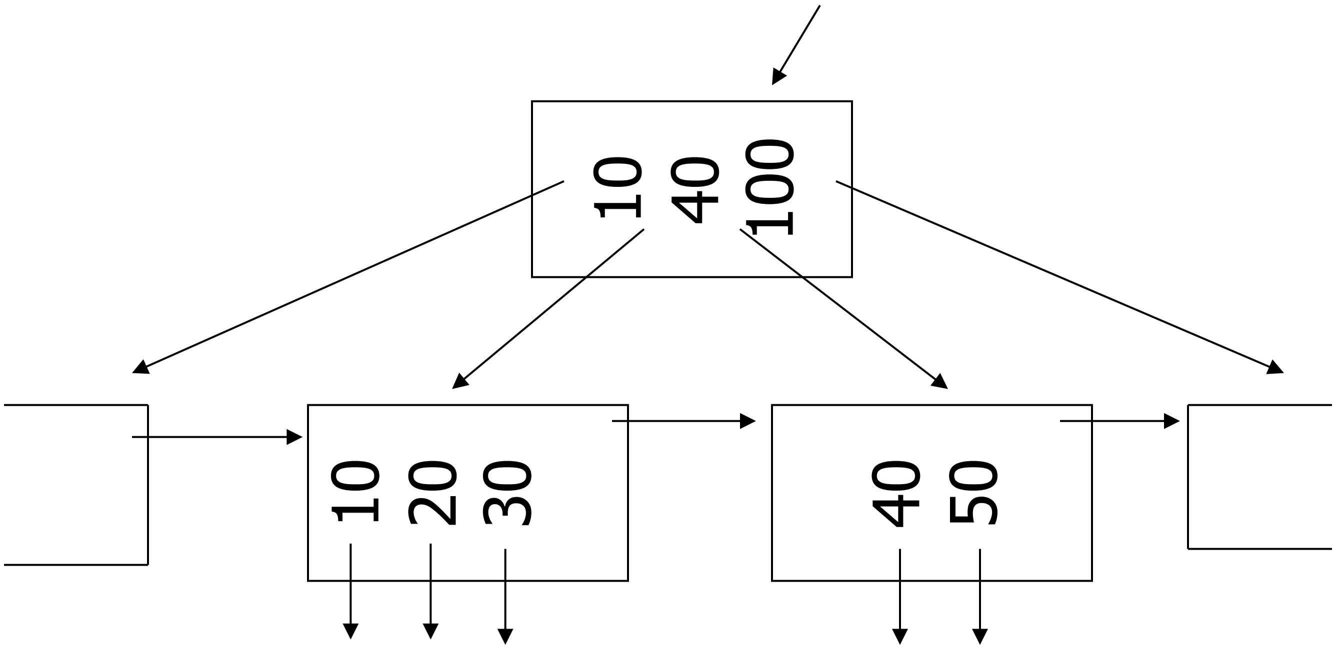
(c) Re-distribute keys

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

(b) Coalesce with sibling

– Delete 50

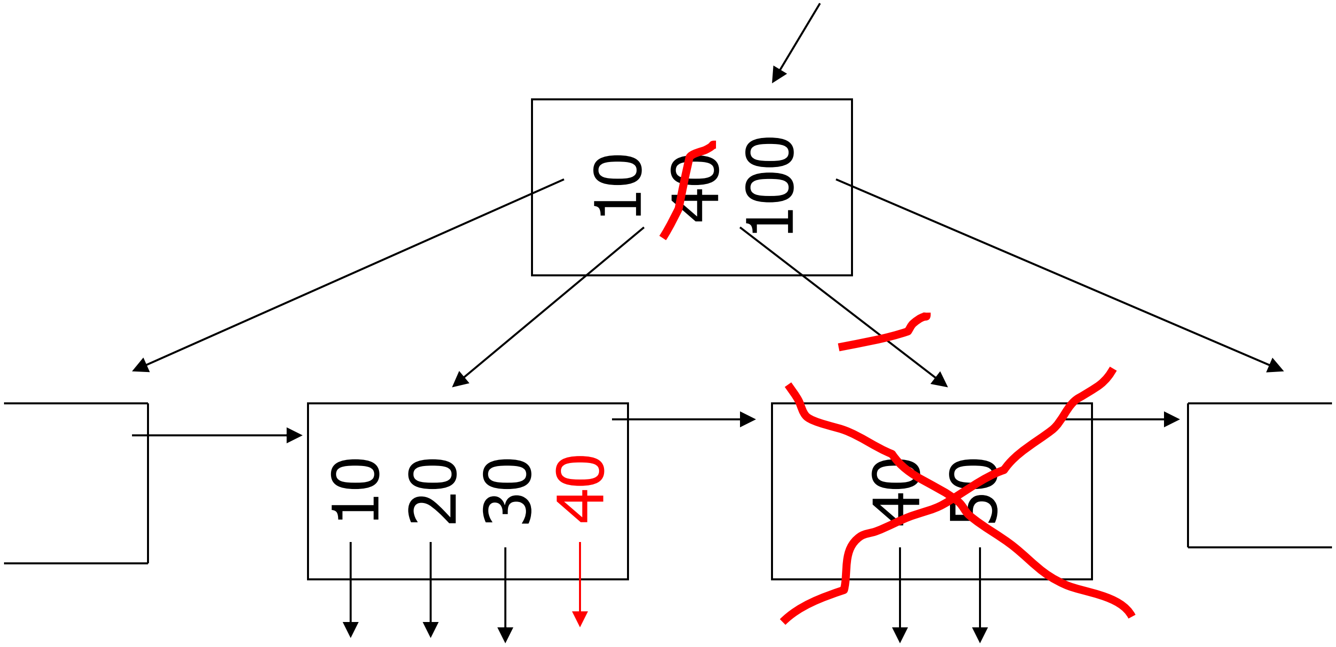
n=4



(b) Coalesce with sibling

- Delete 50

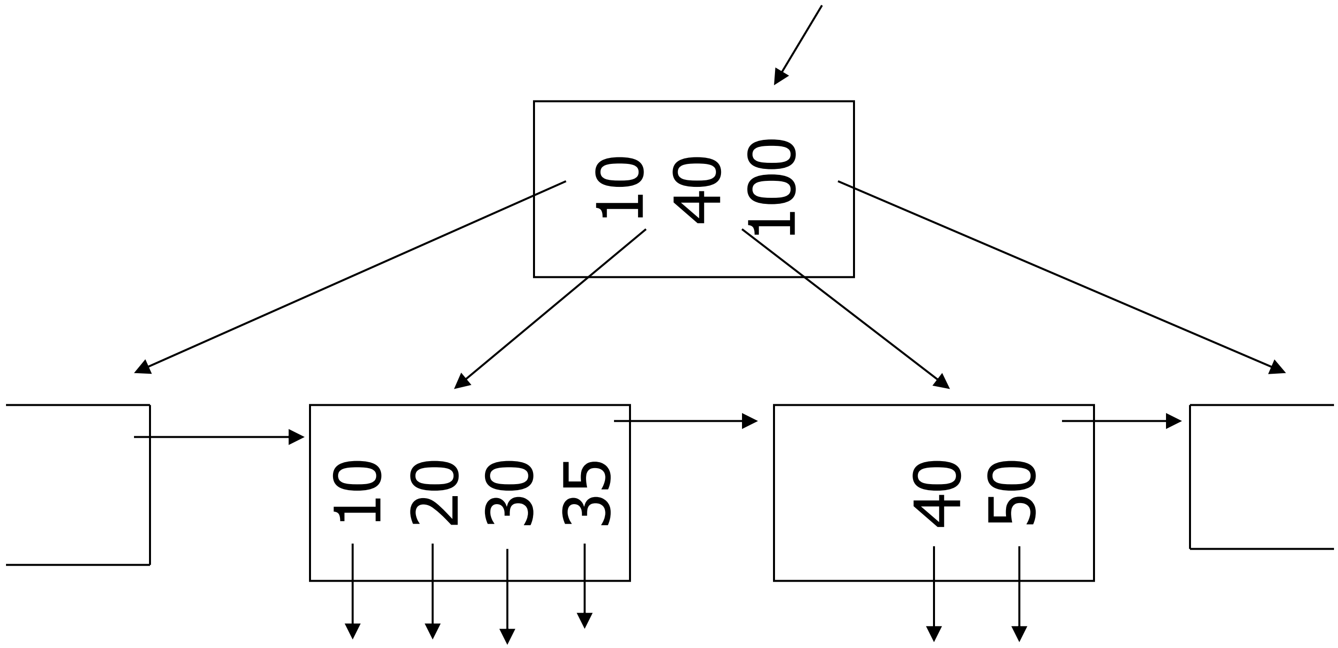
n=4



(c) Redistribute keys

– Delete 50

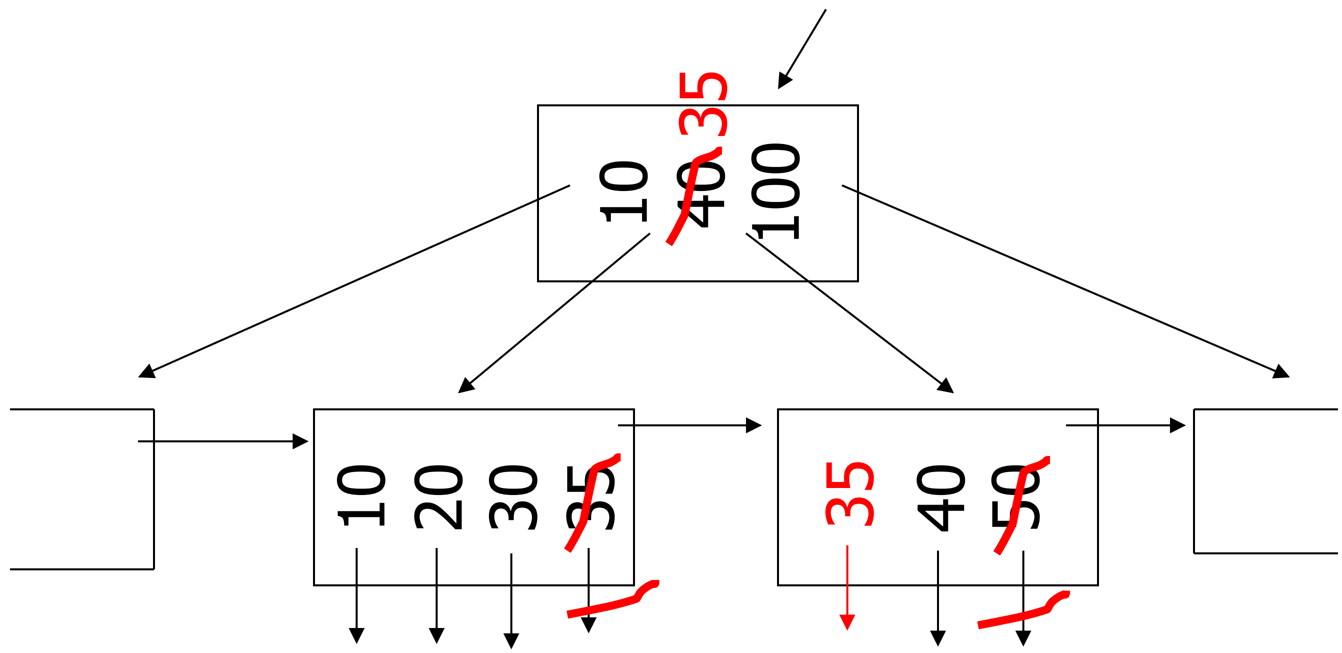
n=4



(c) Redistribute keys

n=4

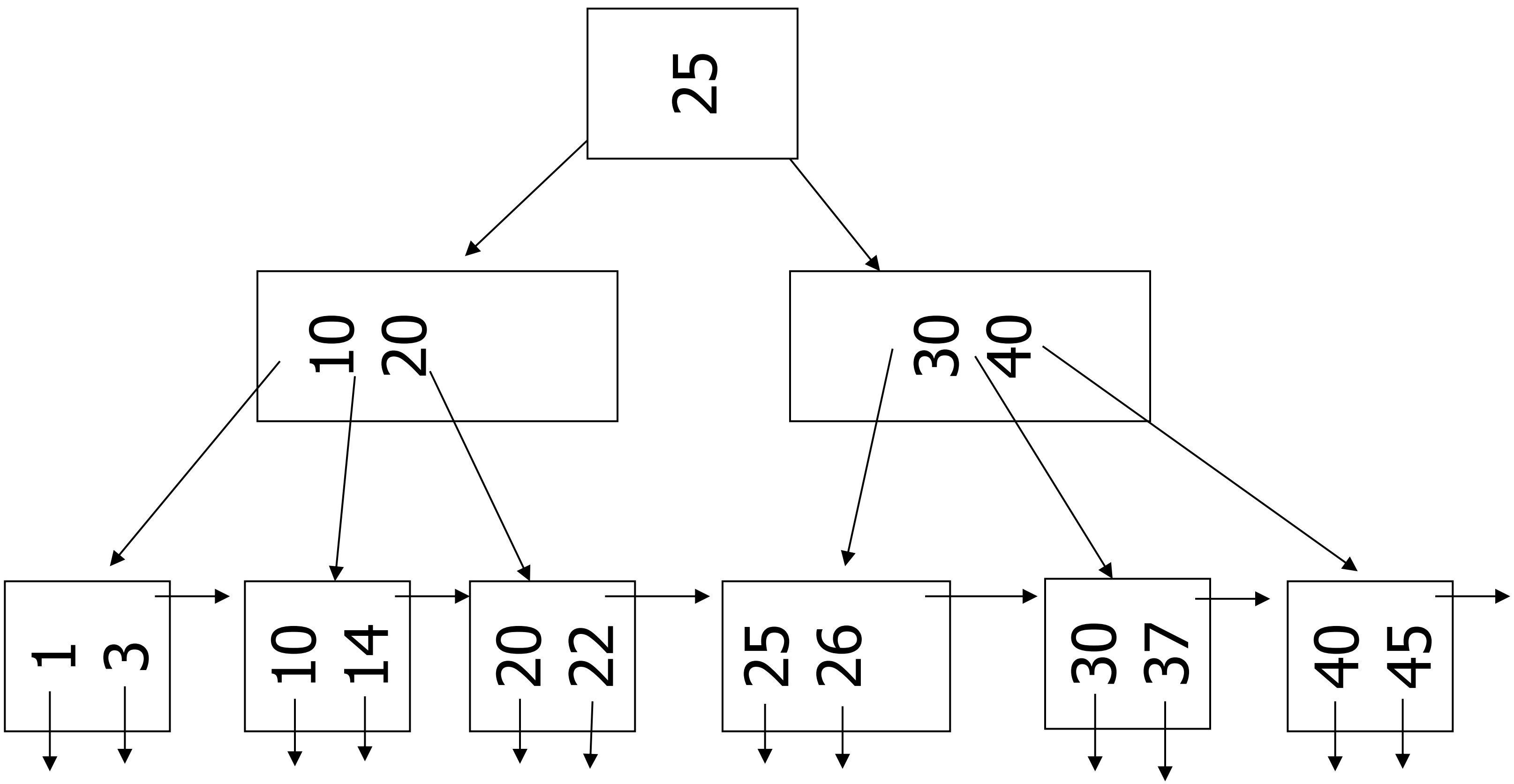
– Delete 50



(d) Non-leaf coalese

– Delete 37

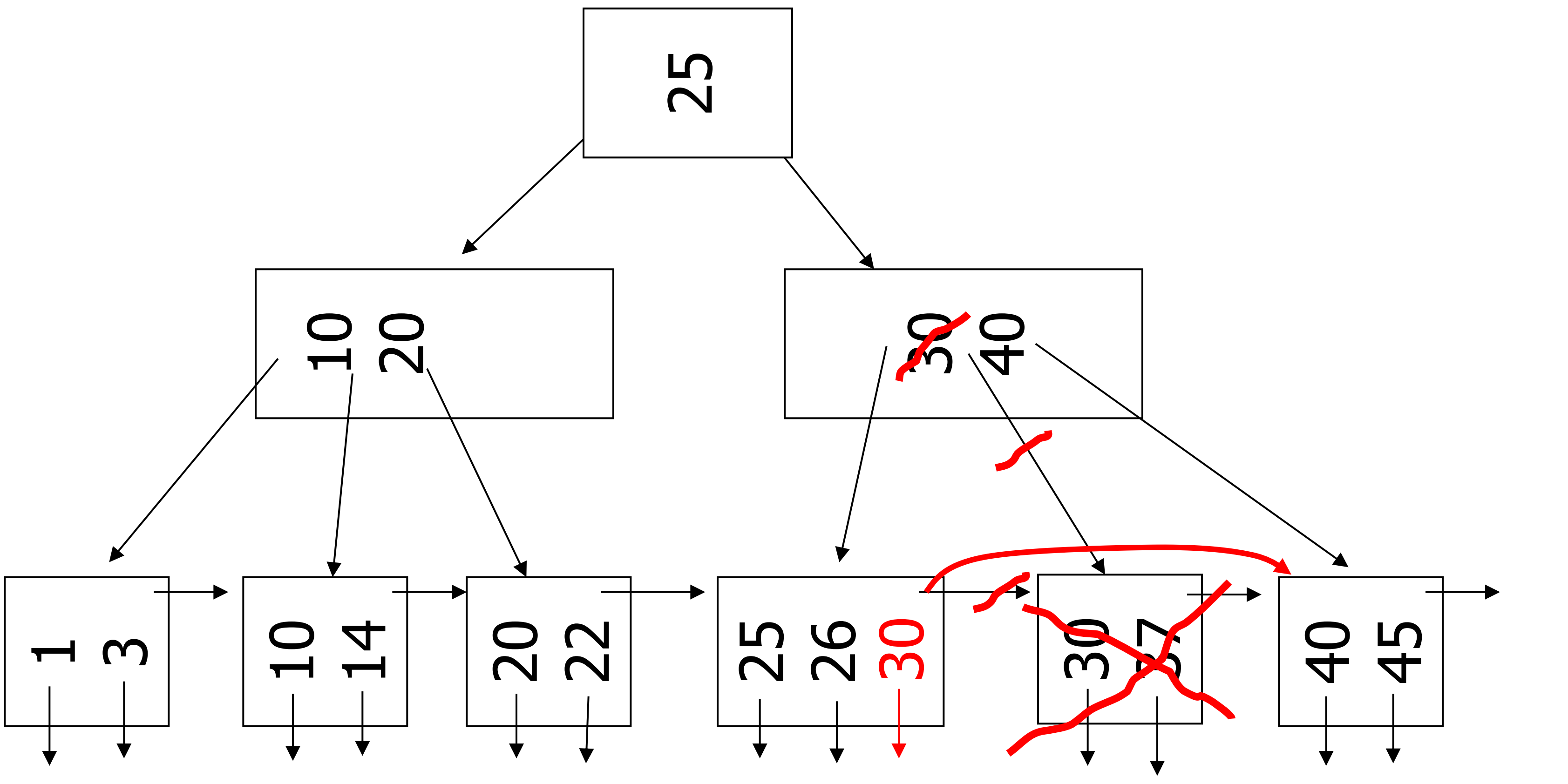
n=4



(d) Non-leaf coalese

– Delete 37

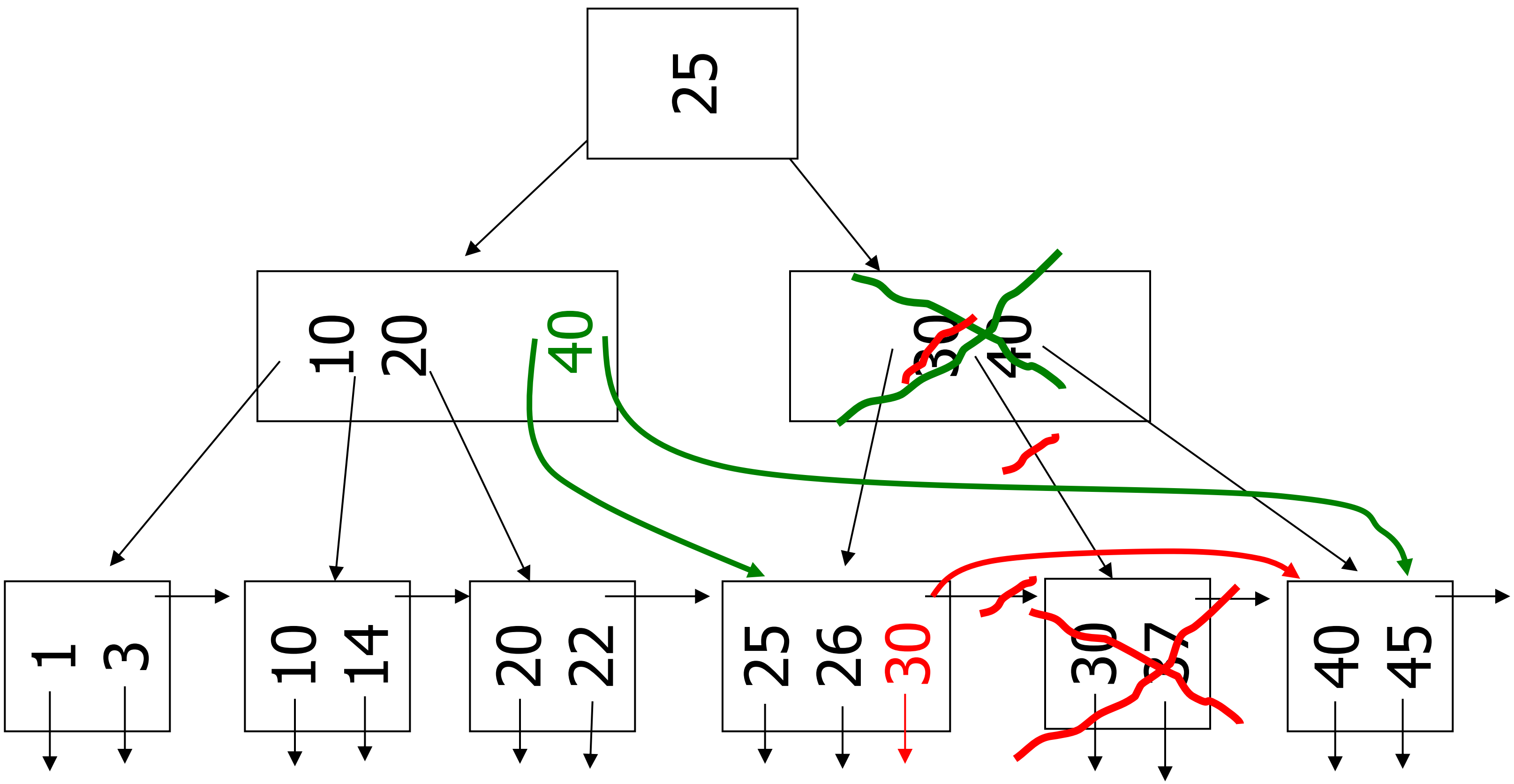
n=4



(d) Non-leaf coalese

n=4

– Delete 37

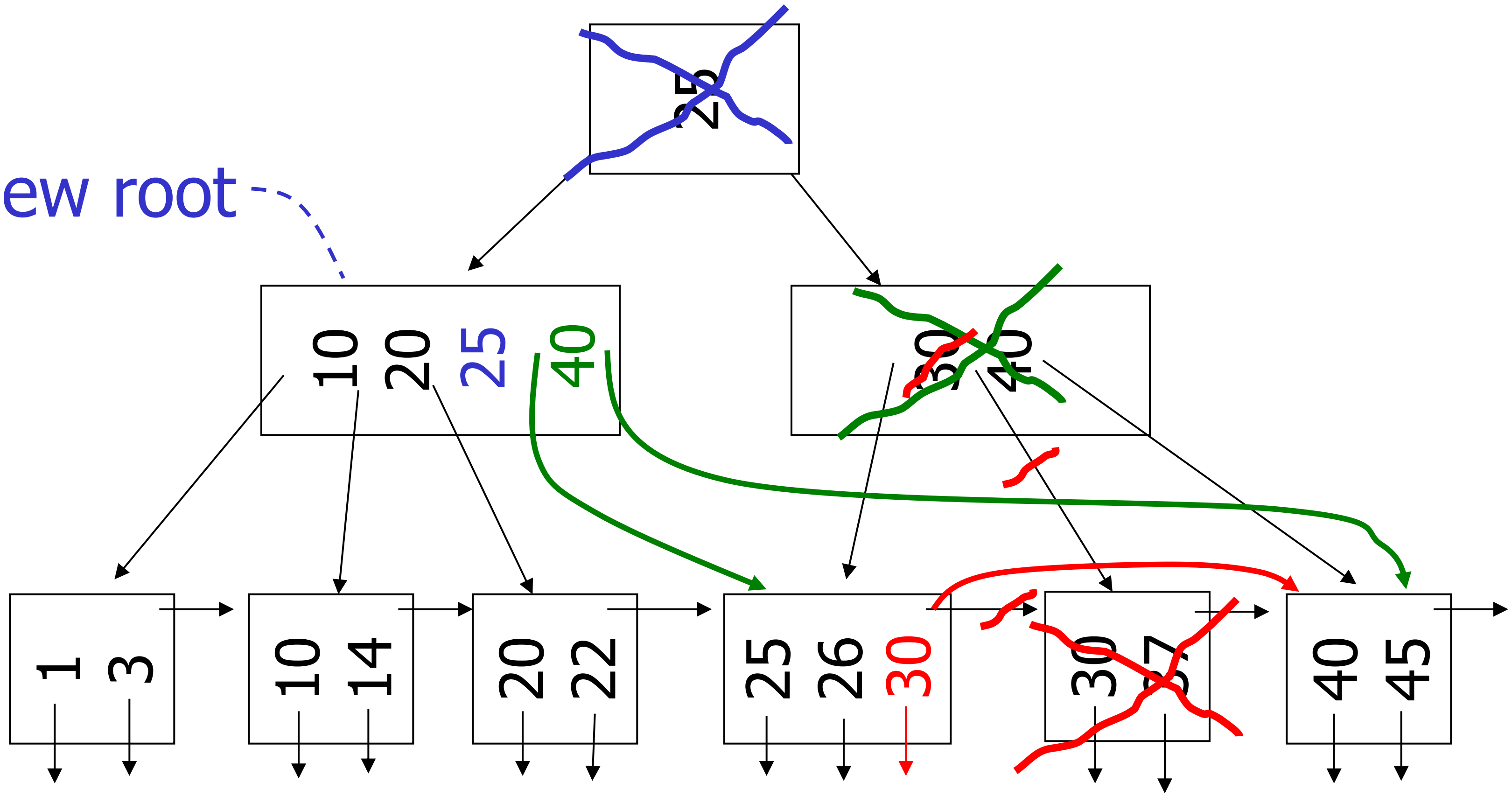


(d) Non-leaf coalese

– Delete 37

n=4

new root



Complexity analysis of search

Assumption: N blocks in the index
n order or the tree

Each search requires navigating along a path from the root to a leaf

Thus the complexity corresponds to the height h of the tree height h is maximum if the branching factor at the nodes is minimal

at the root: 2
at non-leaf nodes: essentially n/2

the root splits the tree into two trees of N/2 keys

we must find the number a such that $(\frac{n}{2})^a$ >= N/2

So $a \sim log_{n/2}$(N/2) = O(log_n(N))

If the branching factor is maximum at each nodes, the complexity analysis also gives O(log_n(N))

Complexity analysis for insert and delete

In the worst case, for both insert and delete, processing is determined by a downward phase of h steps and an upward phase also of h steps

Consequently, the complexity of insert and delete is also O(log_n(N))

To improve complexity, place the first several levels in main memory

For typical cases, search time is measured in terms of a few block I/O’s and this for very large data files

The leaf level provides a sorted list of the records in the data file.

Range searches can be accommodated: given range (k1, k2), locate the leaf holding k1 and then follow along the leaf level until reaching records with key value higher than k2