Data Storage and Query Answering

Indexing and Hashing (3)

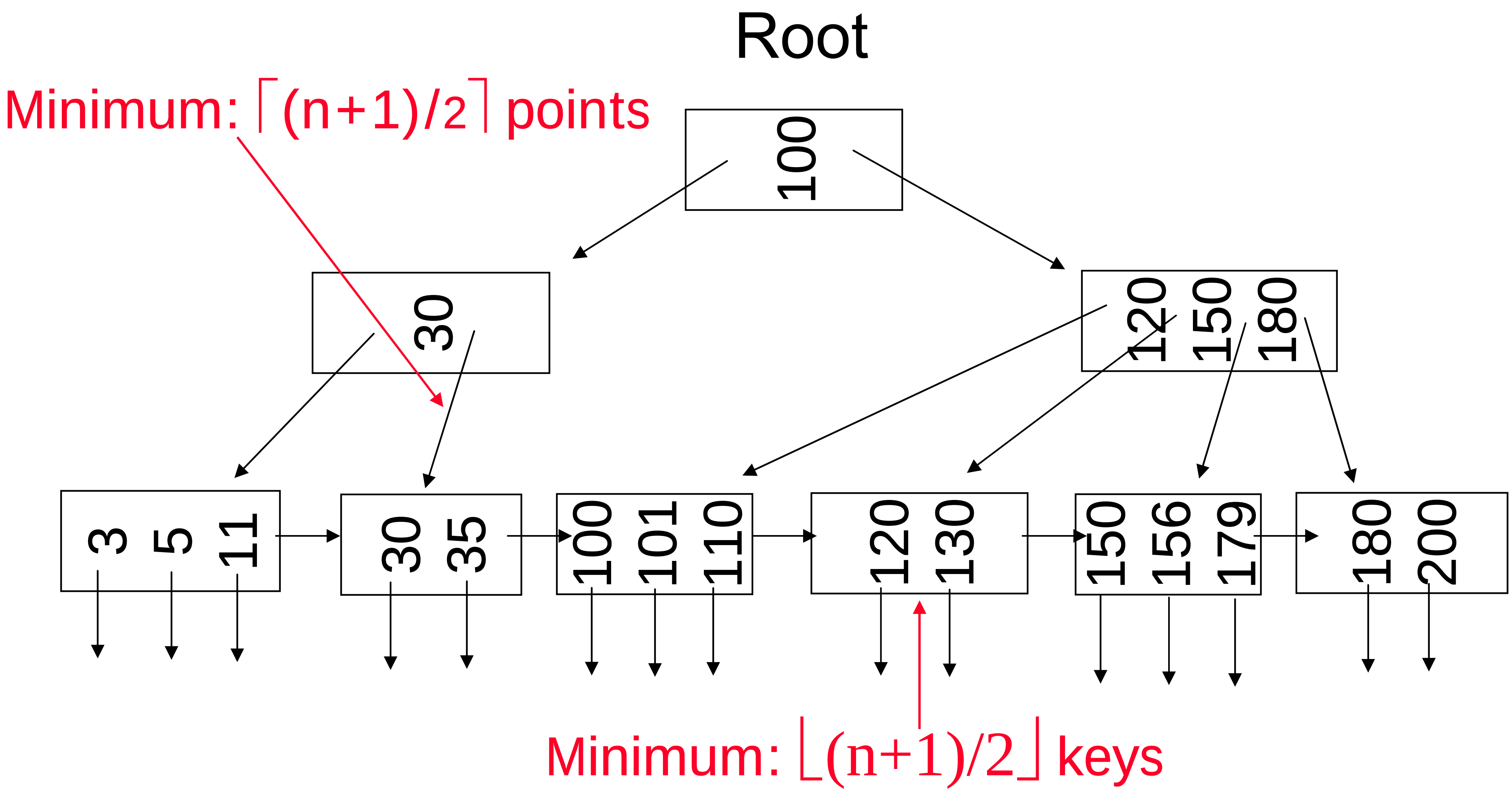
B+ Tree Rules

- (1) Each node is a disk block
  - An I/O access retrieves the whole block into main memory
- (2) All leaves at same lowest level (balanced tree)
- (3) Pointers in leaves point to records, except for "sequence pointer"
- (4) Number of pointers/keys for B+ tree (order n)

	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	1	1

B+ Tree Example

B+ Tree Example n=3



B+ Trees

Insertions

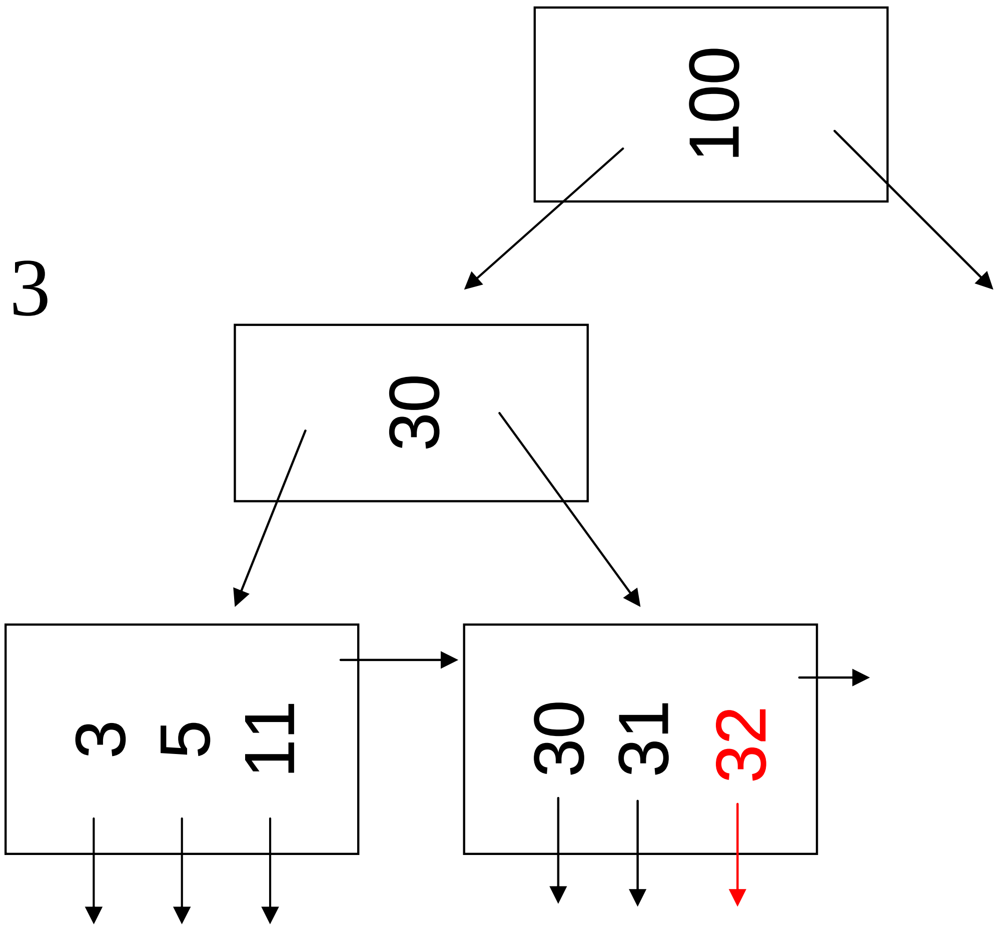
- Always insert in corresponding leaf.
- Tree grows bottom-up.
- Four different cases:
  - Space available in leaf,
  - Leaf overflow,
  - Non-leaf overflow,
  - New root.

B+ Trees

Insertions

- Space available in leaf

n = 3



Insert key 32

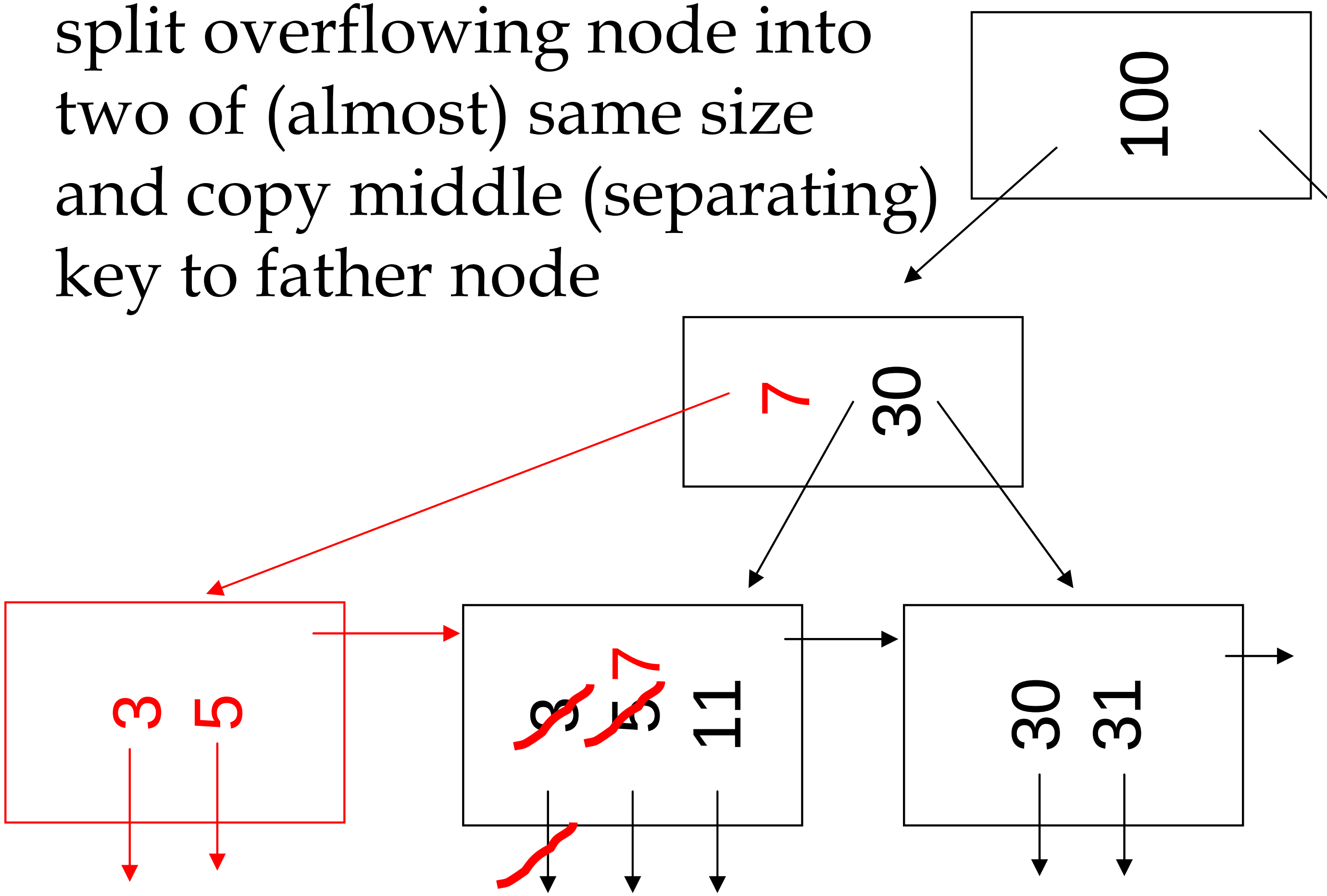
B+ Trees

Insertions

- Leaf overflow
  split overflowing node into two of (almost) same size and copy middle (separating) key to father node

n = 3

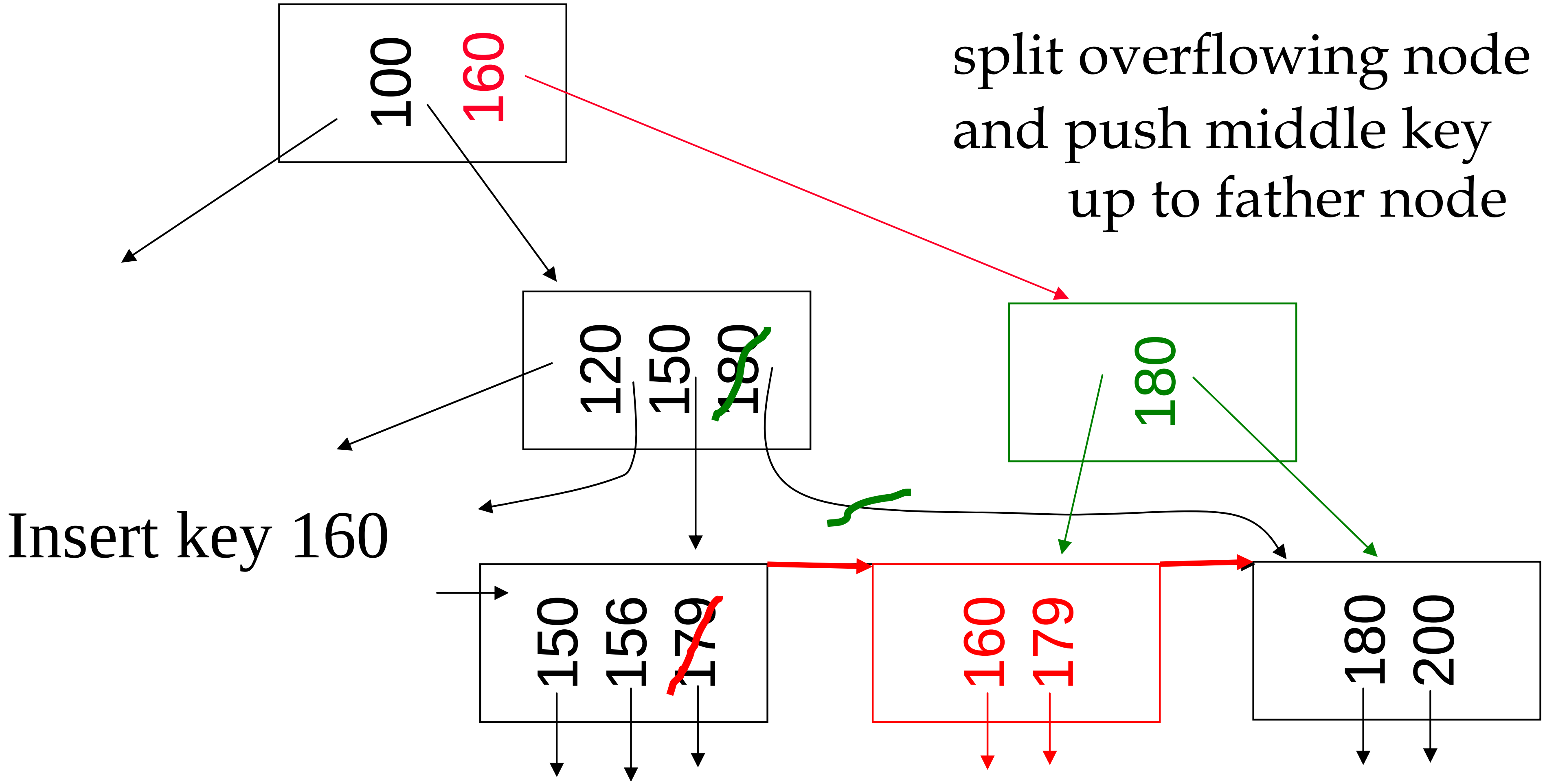
Insert key 7



B+ Trees

Insertions

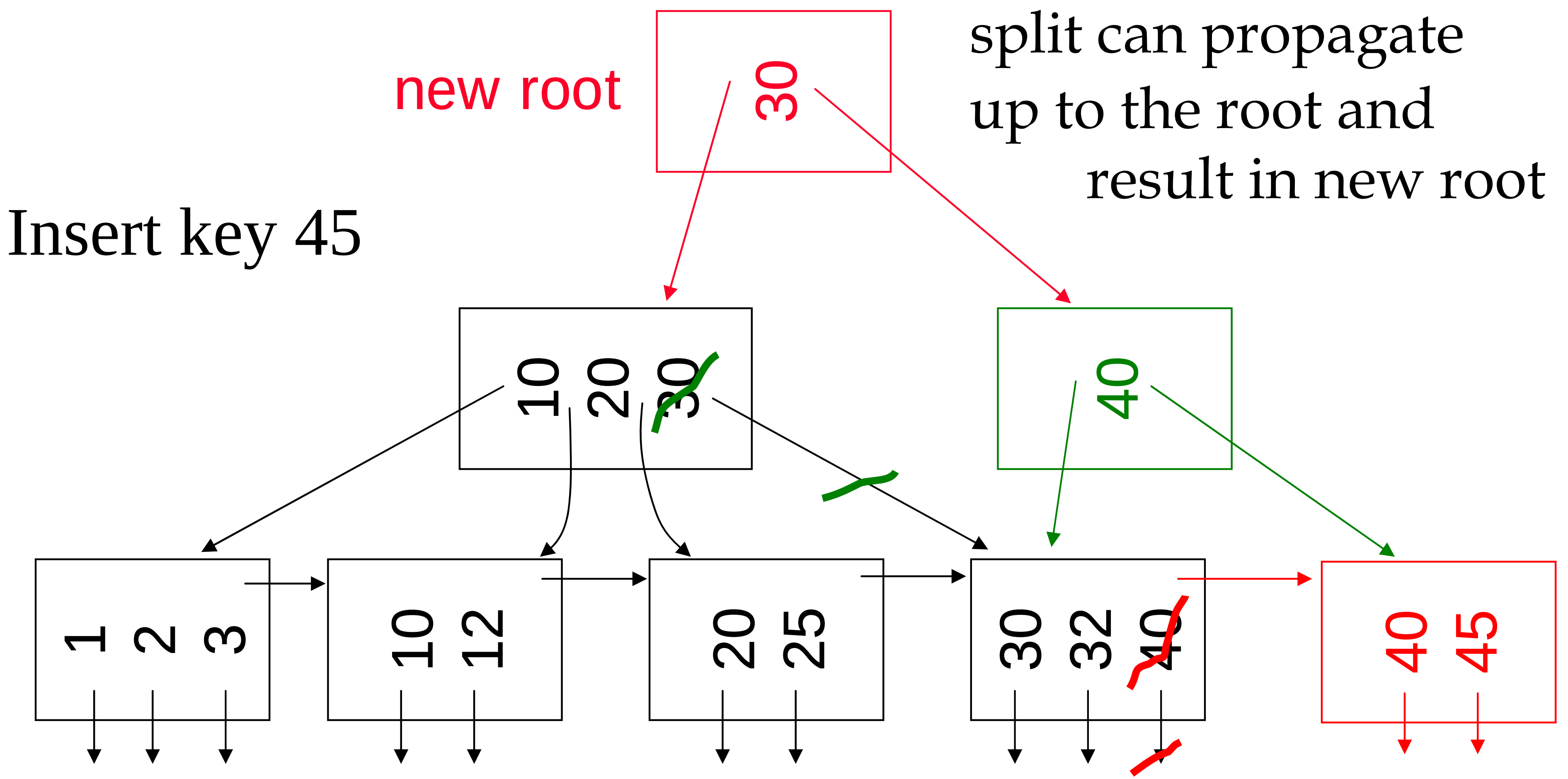
- Non-leaf overflow



B+ Trees

Insertions

- New root



B+ Trees

Deletions

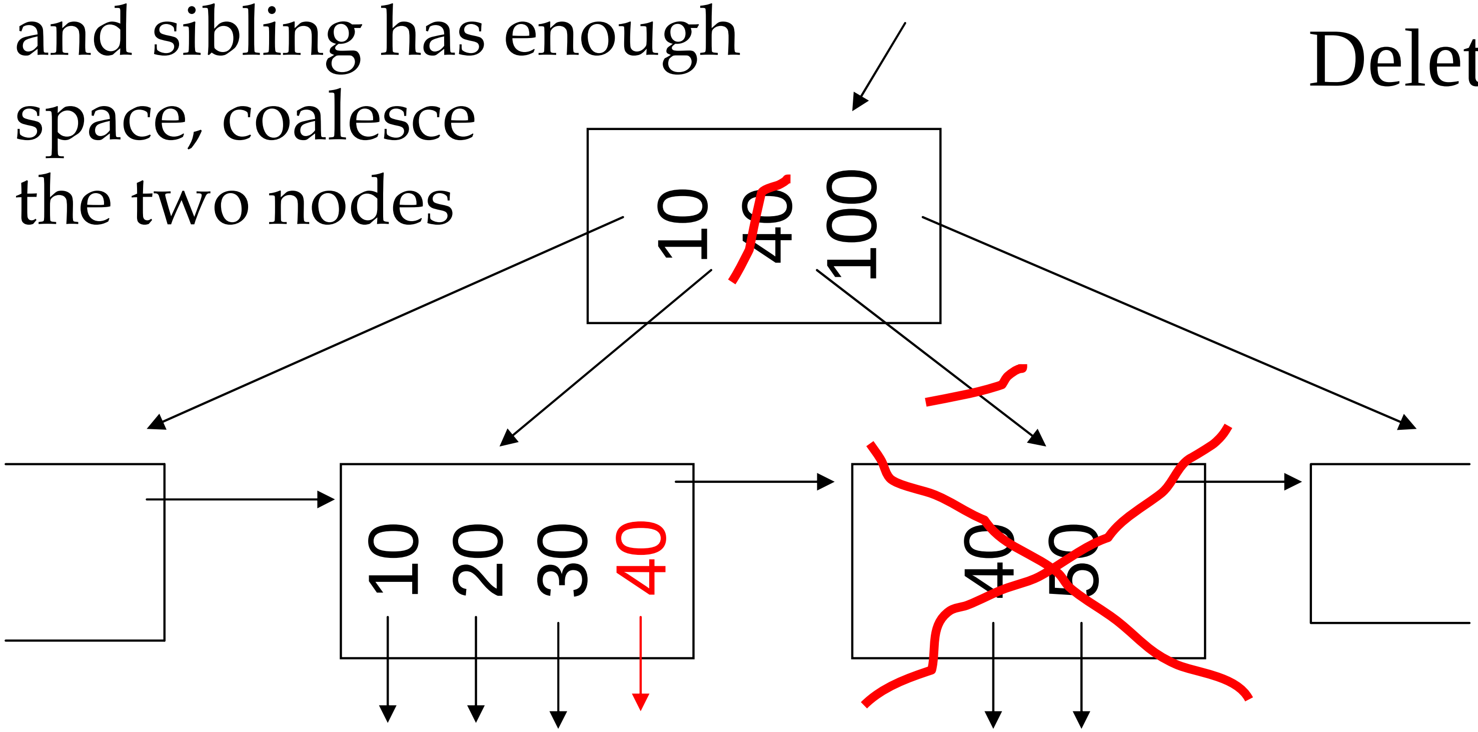
- Locate corresponding leaf node.
- Delete specified entry.
- Four different cases:
  - Leaf node has still enough entries,
  - Coalesce with neighbor (sibling),
  - Re-distribute keys,
  - Coalesce or re-distribute at non-leaf.

Deletions

- Coalesce with neighbor (sibling)
  if node underflows
  and sibling has enough
  space, coalesce
  the two nodes

Delete key 50

n=4



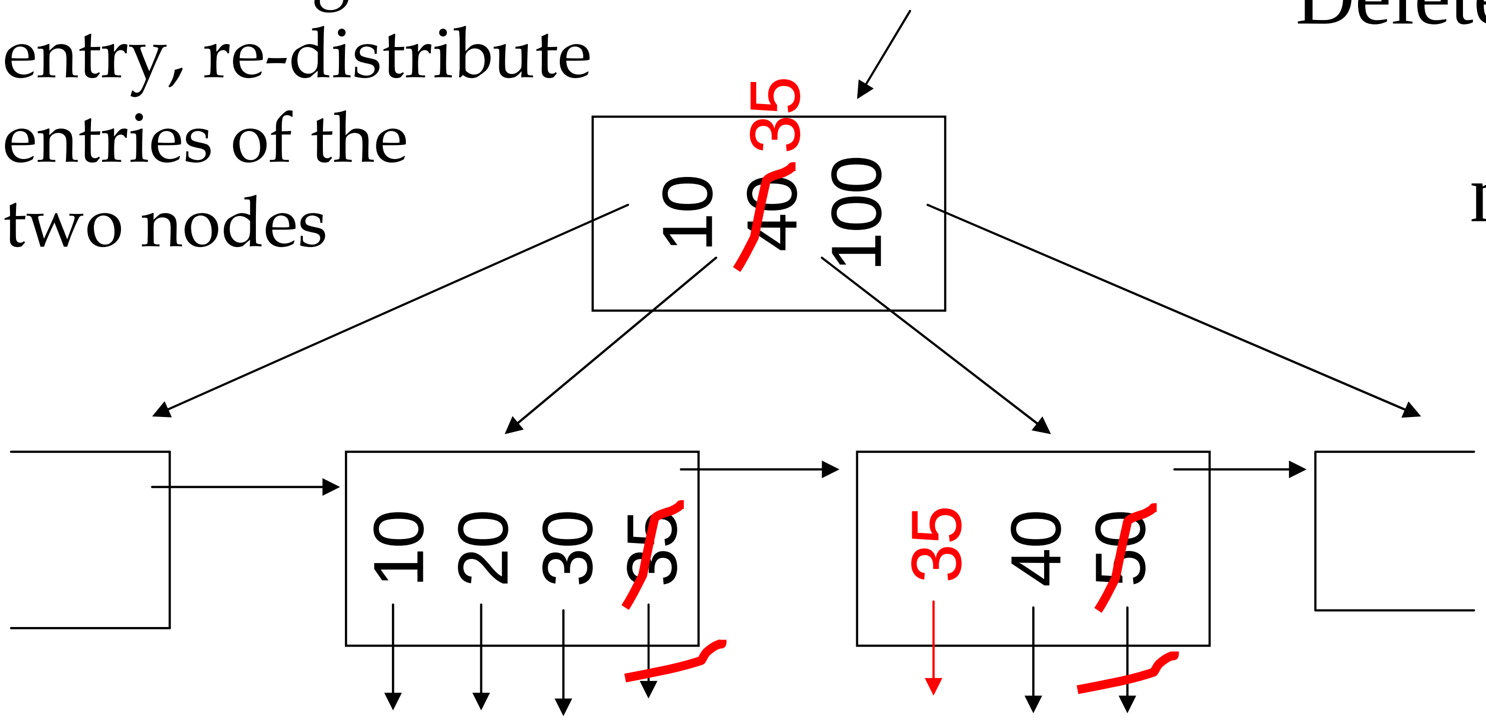
B+ Trees

Deletions

- Redistribute keys
  if node underflows and sibling has extra entry, re-distribute entries of the two nodes

Delete key 50

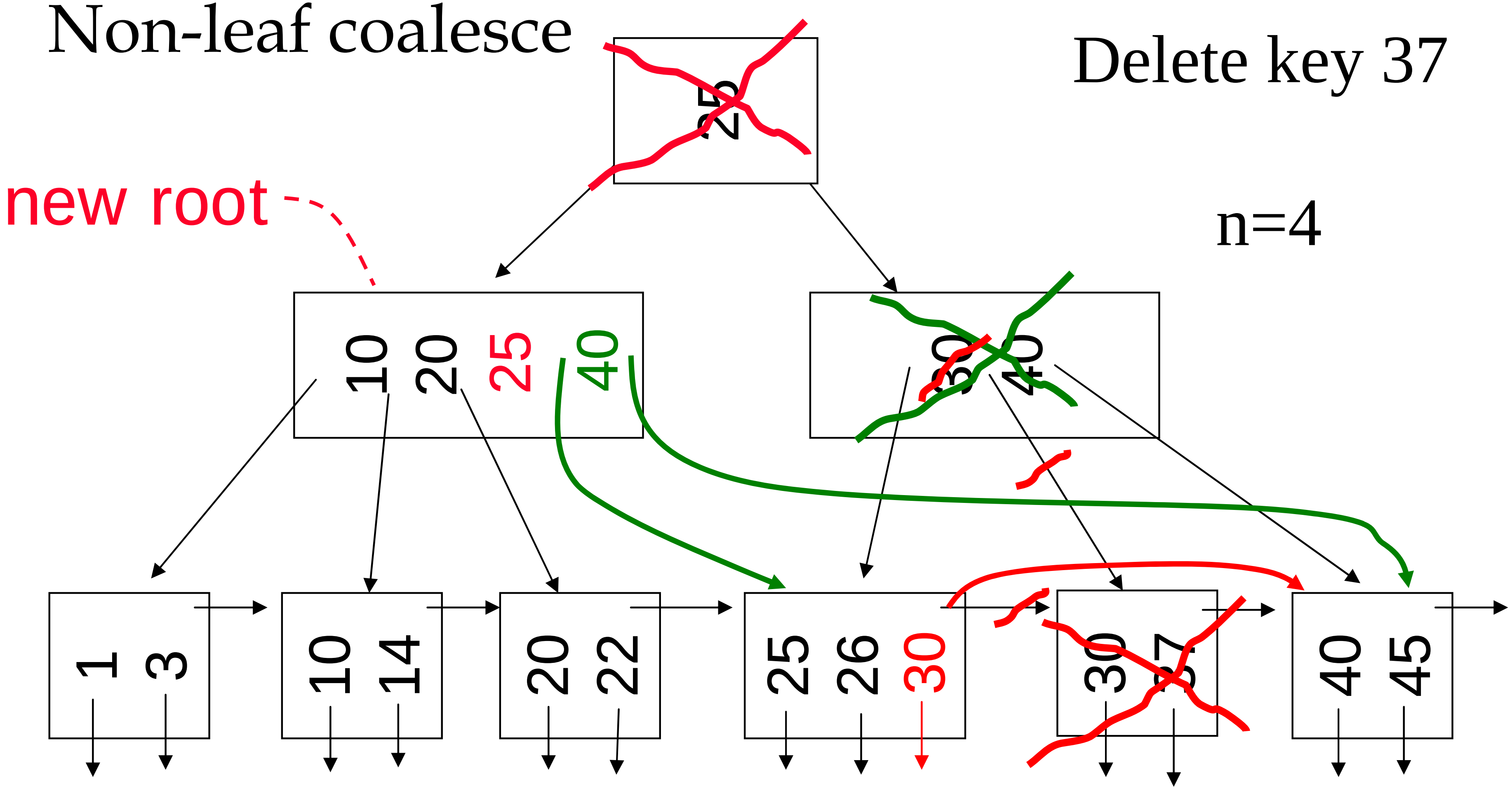
n=4



B+ Trees

Deletions

- Non-leaf coalesce



B+ Trees

B+ Trees in Practice

- Often, coalescing is not implemented.
- It is too hard and typically does not gain a lot of performance.

B+ Trees

B+ Trees in Practice

- Typical order: 200, typical space utilization: 67%, i.e., average fanout = 133.
- Typical capacities:
  - Height 4: 133^4 = 312,900,700 records,
  - Height 3: 133^3 = 2,352,637 records.
- Can often hold top levels in buffer pool:
  - Level 1 = 1 blocks = 8 Kbytes,
  - Level 2 = 133 blocks = 1 Mbyte,
  - Level 3 = 17,689 blocks = 133 Mbytes.

Fanout: the number of pointers in a node

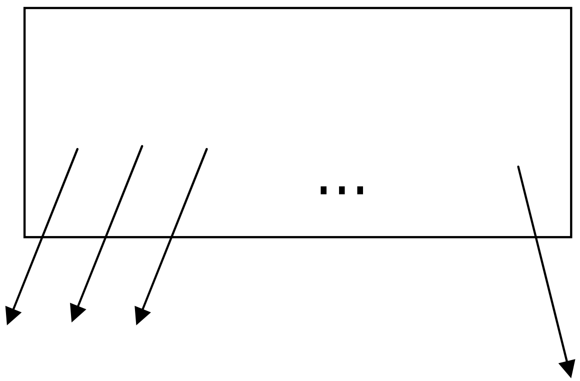
B+ Trees

B+ Trees in Practice

- Order (n) concept replaced by physical space criterion in practice (‘*at least half-full*’).
- Inner nodes can typically hold many more entries than leaf nodes.
- Variable sized records and search keys mean different nodes will contain different numbers of entries.
- Even with fixed length fields, multiple records with the same search key value (duplicates) can lead to variable-sized data entries.

Interesting Problem

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



n is number of keys / node

Sample Assumptions

(1) Time to read node from disk is
$(S+Tn)$ msec.

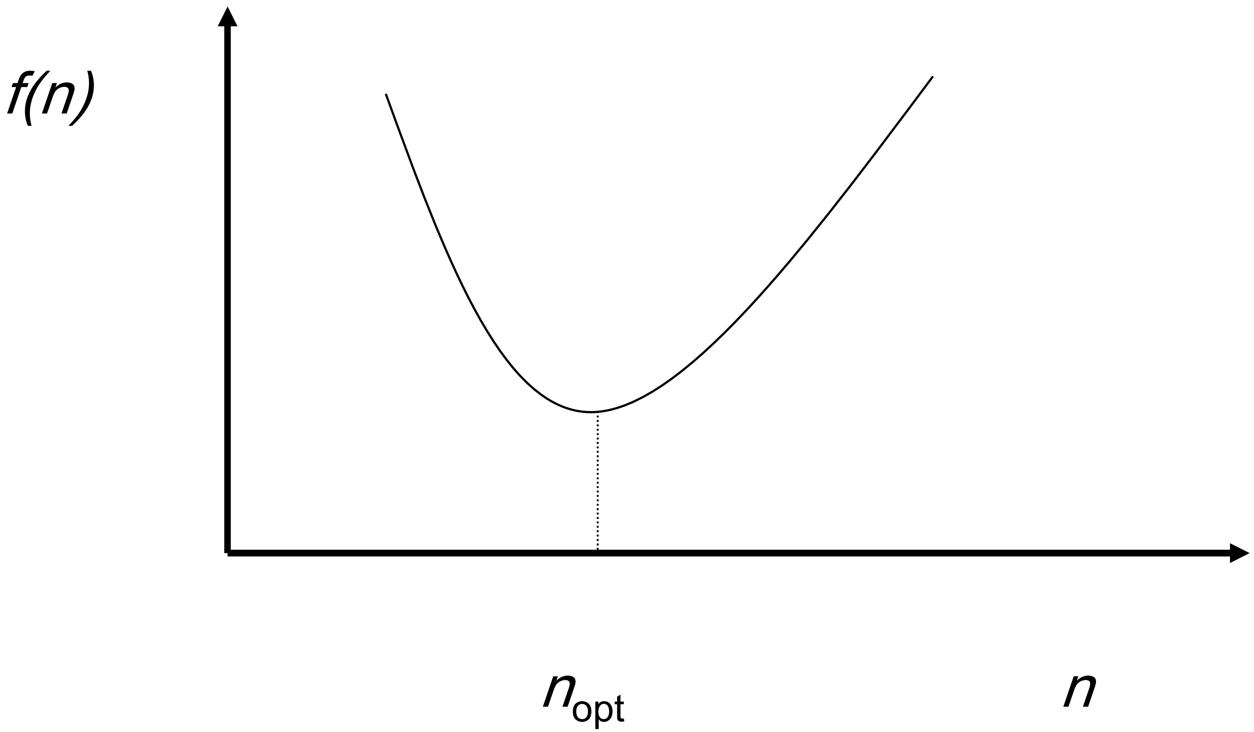
(2) Once block in memory, use binary search to locate key:

$$(a + b \, \mathrm{LOG}_2 \, n) \text{ msec.}$$

For some constants *a*,*b;* Assume a << S

(3) Assume B+ tree is full, i.e.,
\# nodes to examine is $\mathrm{LOG}_n N$
where N = # records

Get: $f(n)$ = time to find a record



FIND n_{opt} by $f'(n) = 0$

Answer is n_{opt} = "few hundred"

⇘ What happens to n_{opt} as

- Disk gets faster?
- CPU get faster?