Indexing

Hector Garcia-Molina

Mahmoud Sakr

Indexing

Query → Index → Block holding records → candidate records
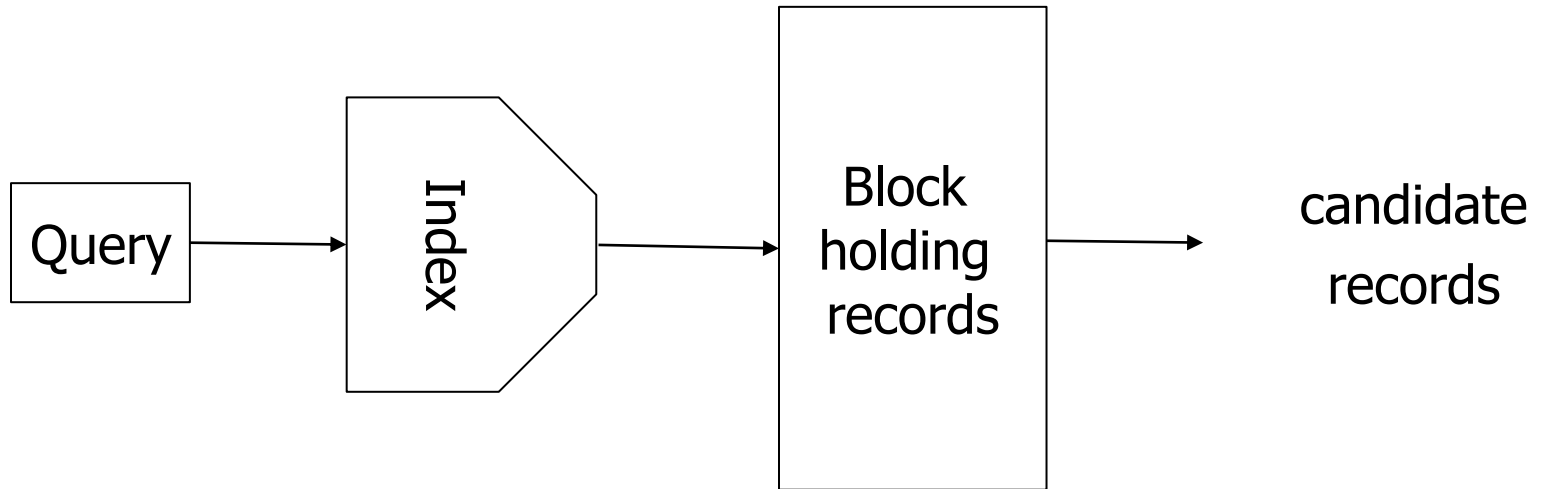
Topics

- Conventional indexes
- B-trees
- Hashing schemes

Sequential File

10	
20	

30	
40	

50	
60	

70	
80	

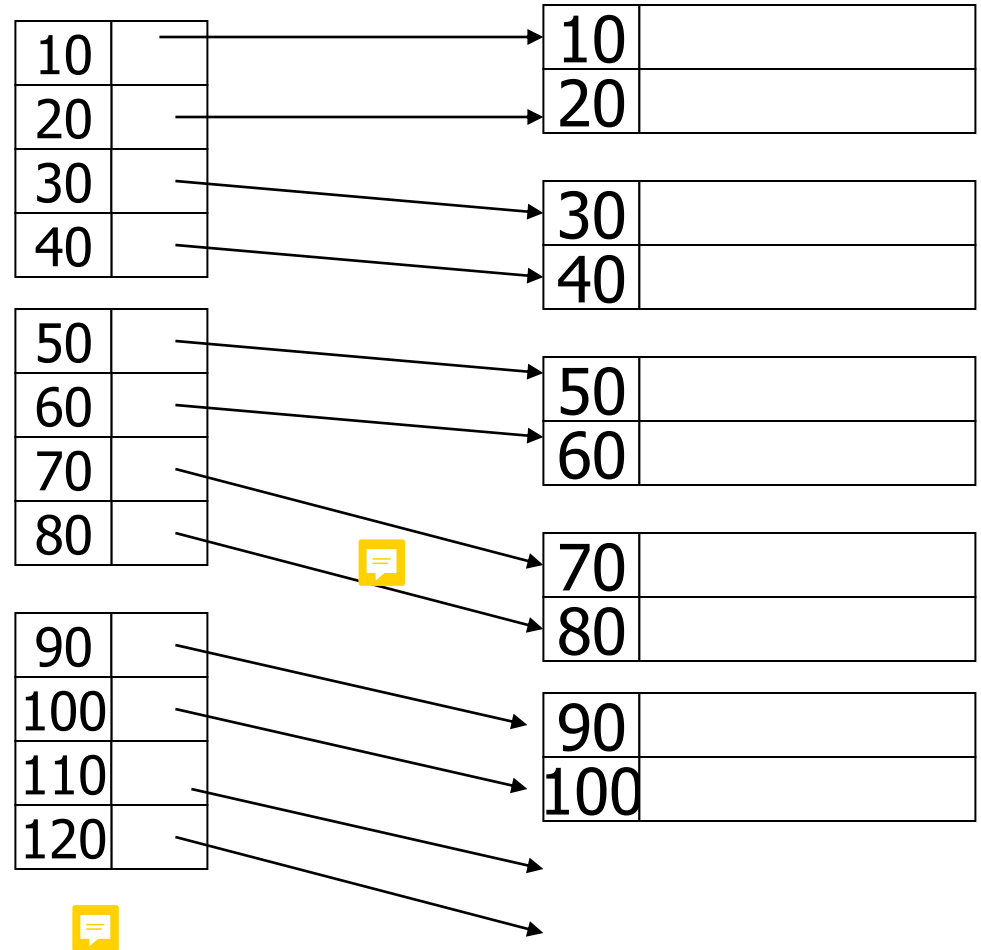| 90	
100	

Dense Index

Sequential File

Dense Index = a pointer per key

How to search for a key= 30 ?
How to search for a key= 25 ?

Can we use a dense index on a non-sequential file ?

Why querying a dense index is more efficient than querying the sequential file ?

10	
20	
30	
40	

50	
60	
70	
80	

90	
100	
110	
120	

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

Sparse Index

Sequential File

Sparse index = a pointer per block

How to search for a key= 30 ?

How to search for a key= 25 ?

Can we use a sparse index on a non-sequential file ?

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

| 10 | |
| 20 | |

| 30 | |
| 40 | |

| 50 | |
| 60 | |

| 70 | |
| 80 | |

| 90 | |
| 100 | |

Sparse 2nd level

Sequential File

10	
90	
170	
250	

330	
410	
490	
570	

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

If the first level index has so many pages, adding a second level sparse index can speed up the search.

Do we benefit from a second level dense index ?

Sparse vs. Dense Tradeoff

- <u>Sparse:</u> Less index space per record
      can keep more of index in memory
- <u>Dense:</u> Can tell if any record exists
      without accessing file

Duplicate keys

Dense index, one way to implement?

Dense index.

Can We do better ?

10	
10	
10	
20	

20	
30	
30	
30	

10	
10	

10	
20	

20	
30	

30	
30	

40	
45	
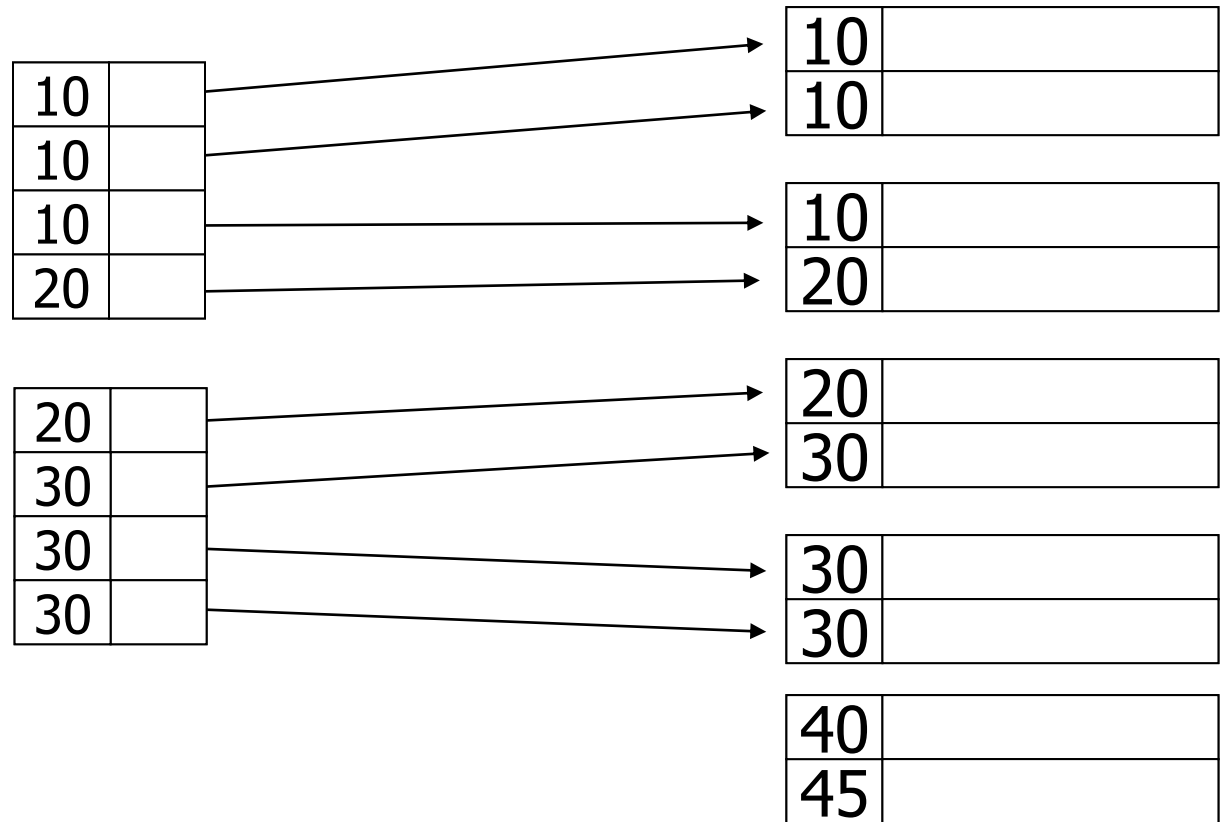
Duplicate keys

Dense index, better way?
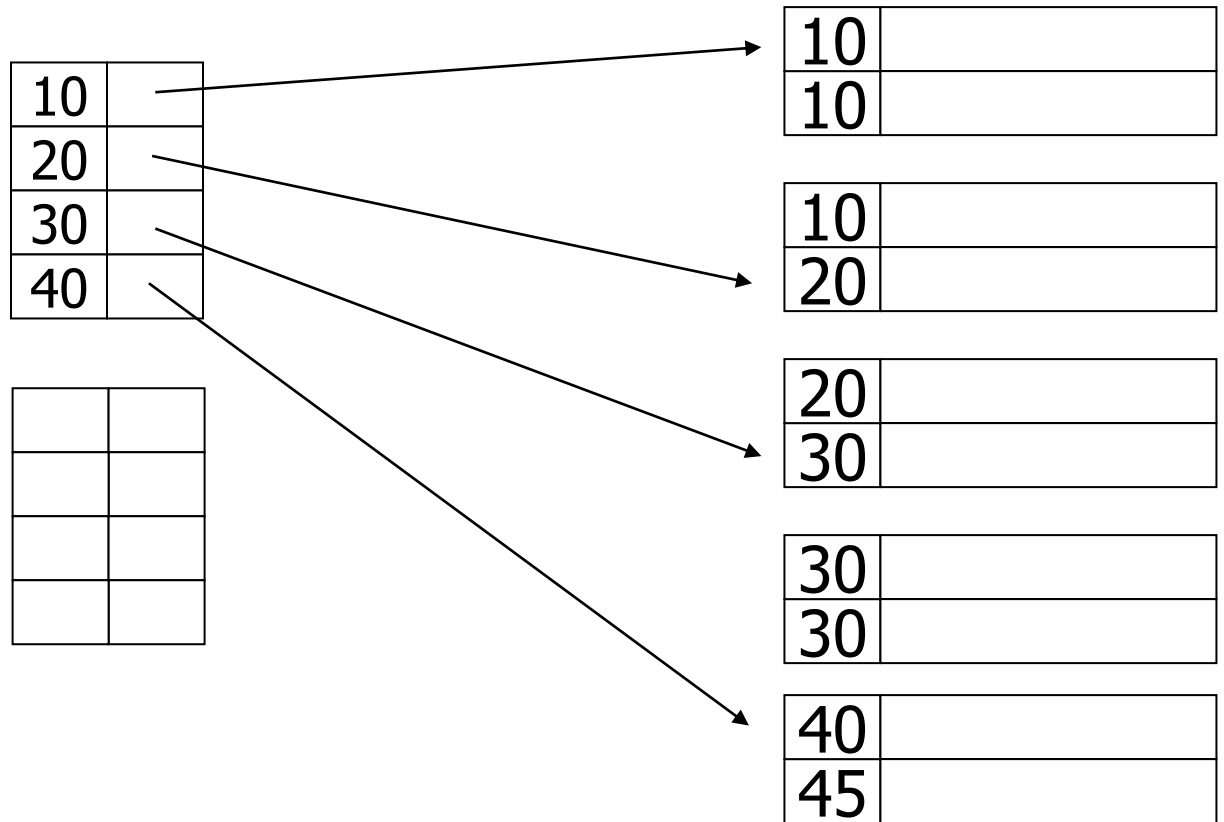
Duplicate keys

Sparse index, one way?

Sparse index - place
key from block

How to search for 30 ?

10	
10	
20	
30	

10	
10	

10	
20	

20	
30	

30	
30	

40	
45	
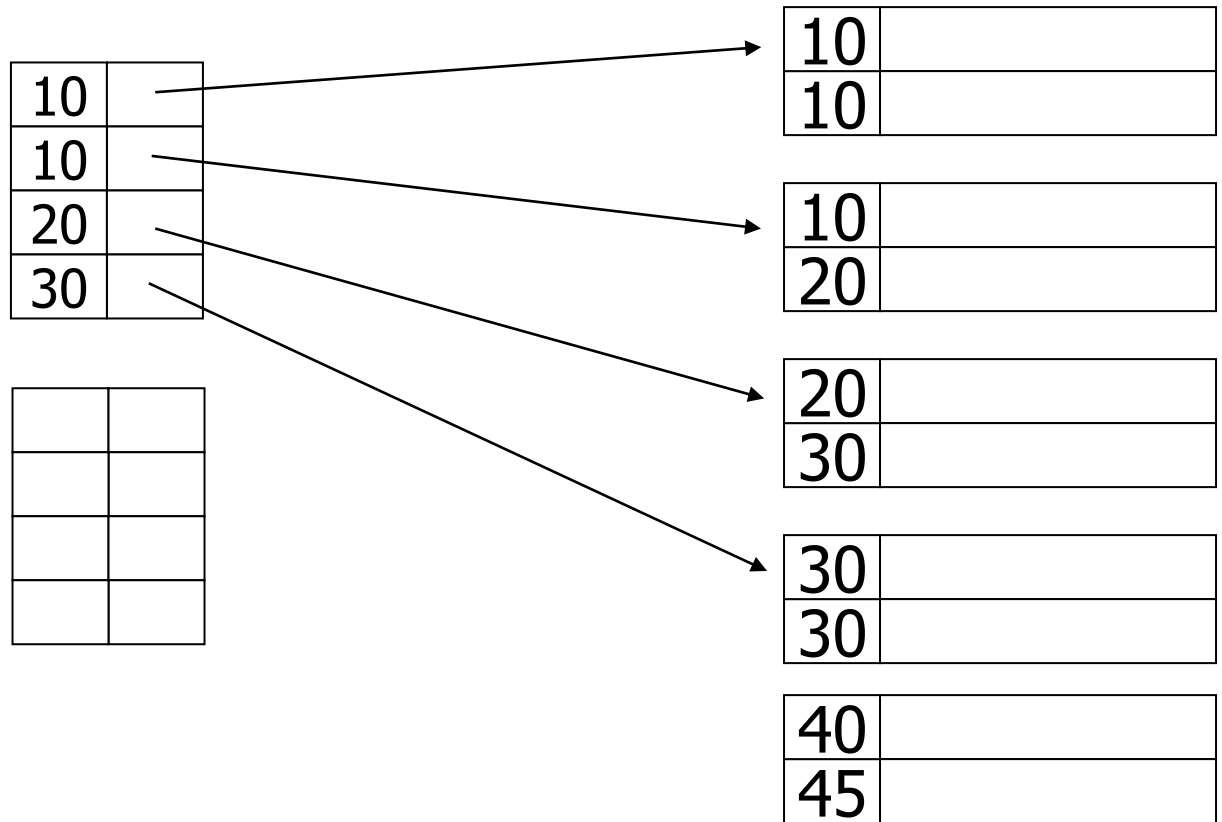
Duplicate keys

Sparse index, another way?

Sparse index - place first <mark>new</mark> key from block

How to search for 30 ?

How to search for 35 ?
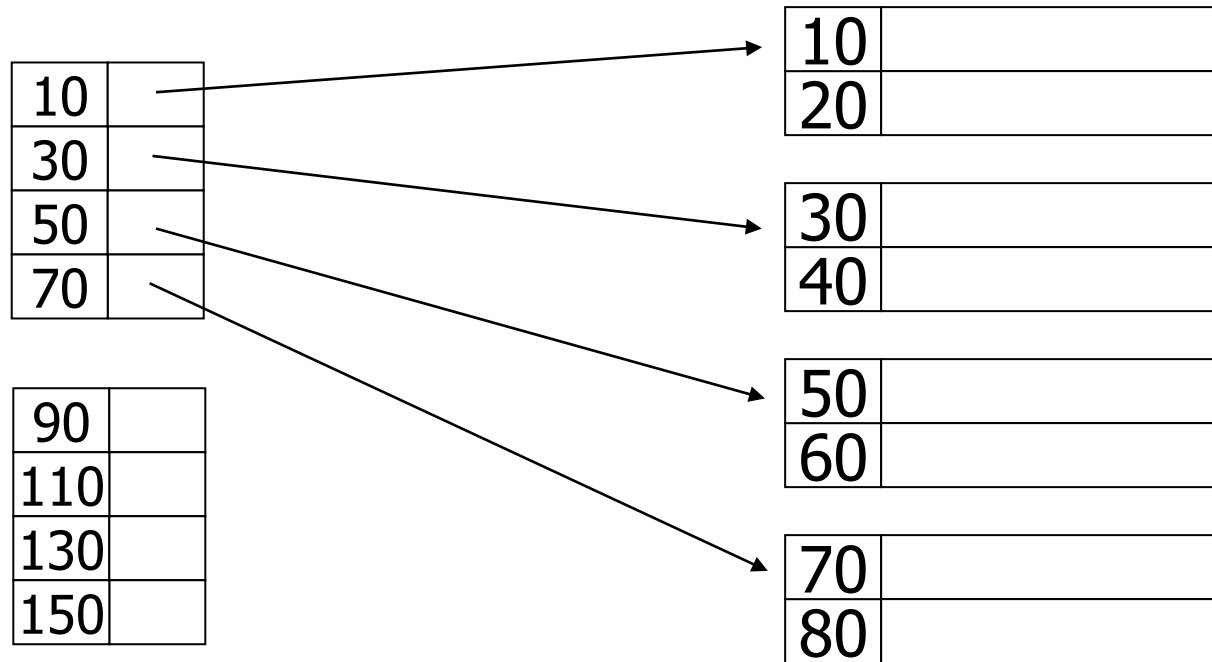
10	
20	
30	
30	

| 10 | |
| 10 | |

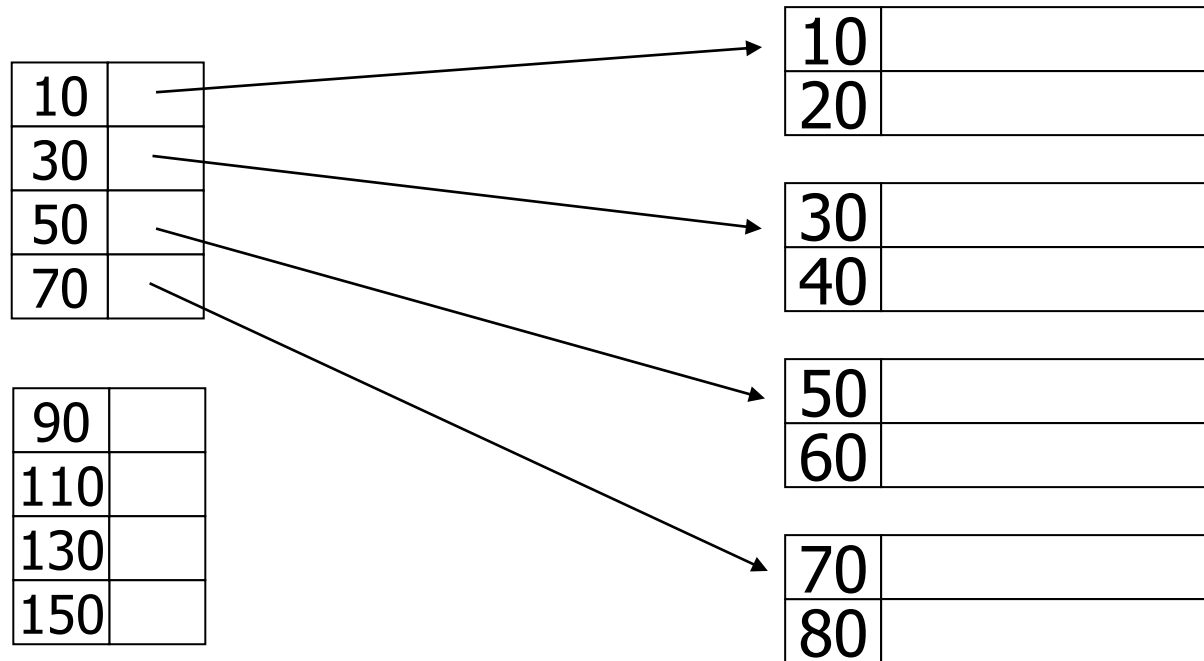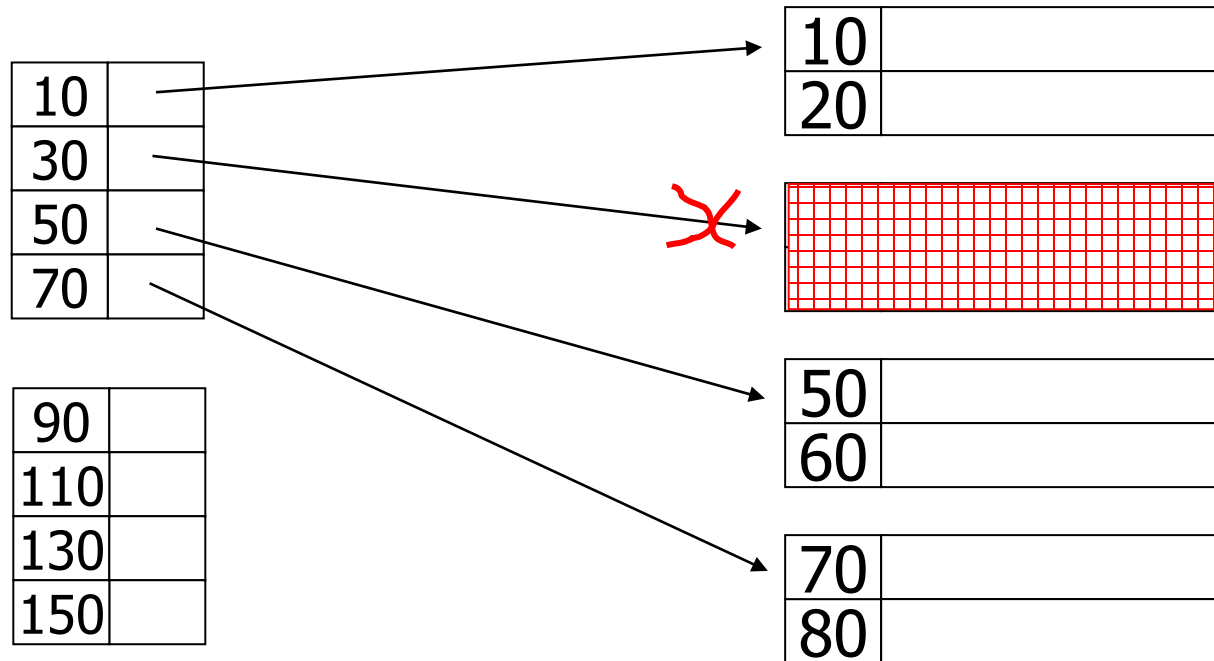| 10 | |
| 20 | |

| 20 | |
| 30 | |

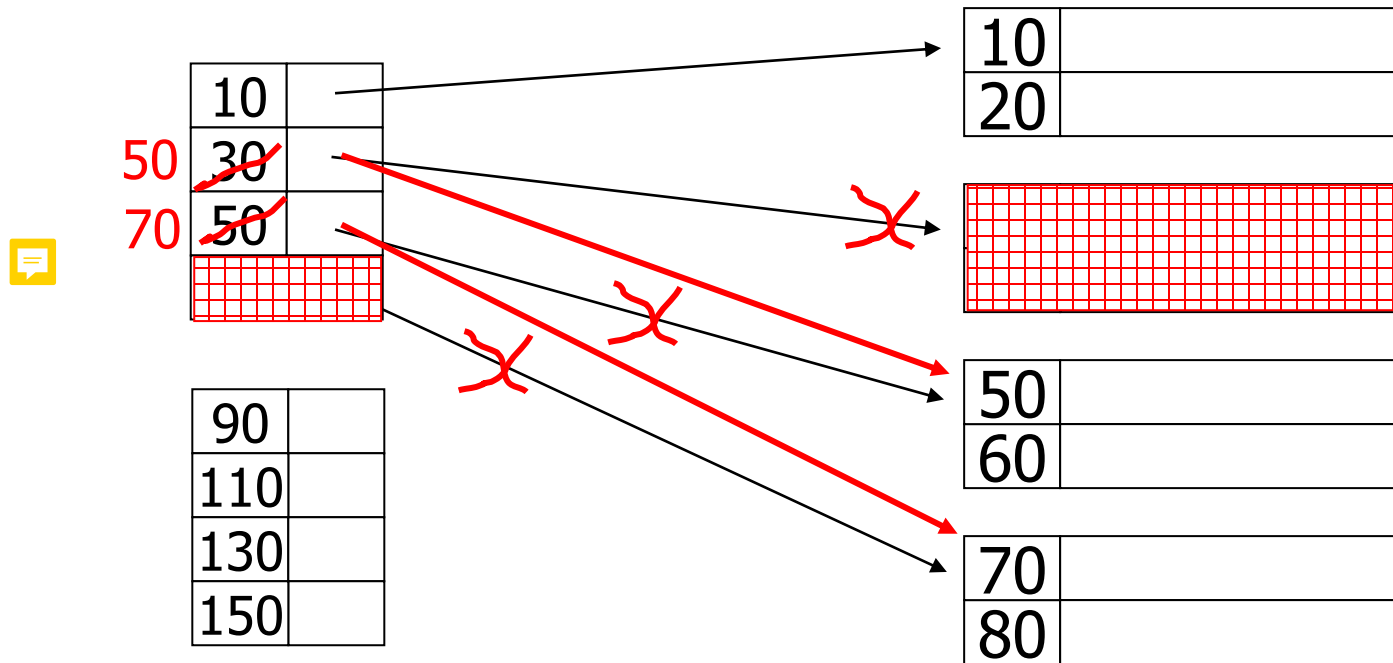| 30 | |
| 30 | |

| 40 | |
| 45 | |

Deletion from sparse index

Deletion from sparse index
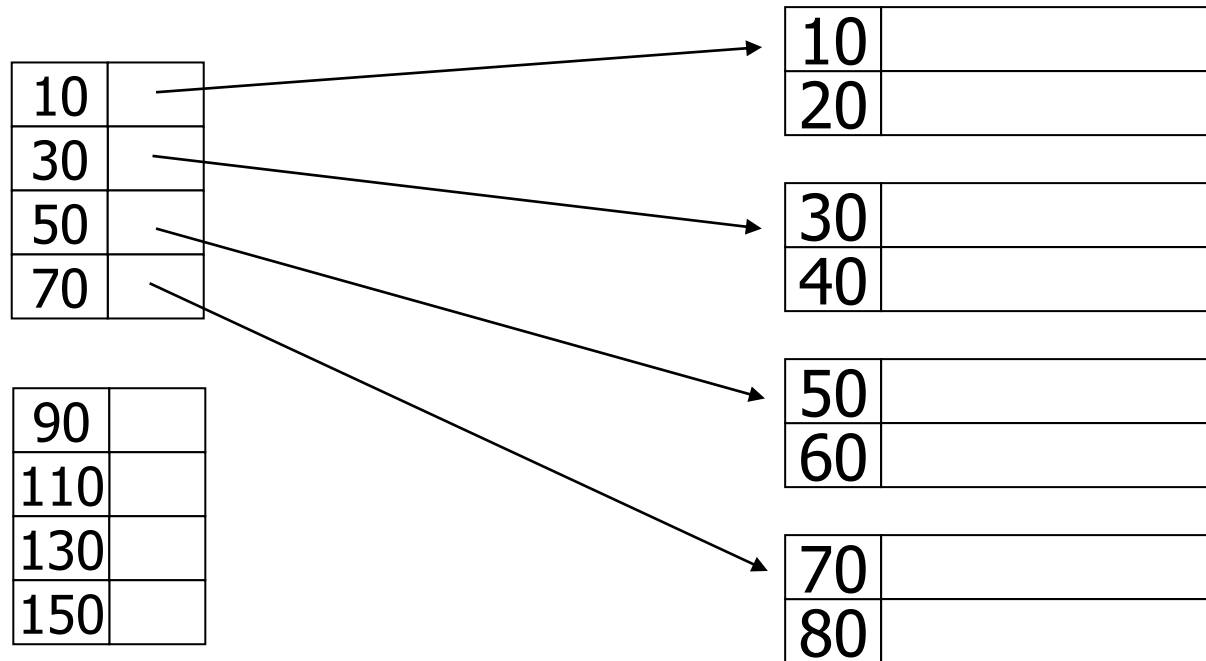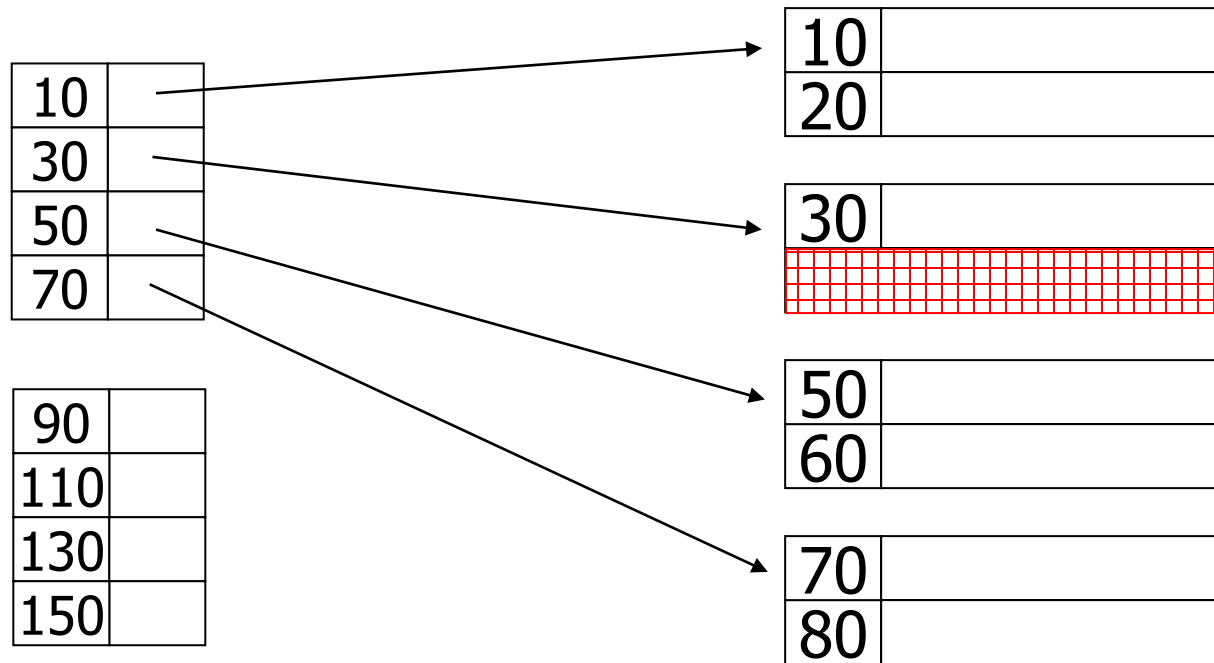
– delete record 40

Deletion from sparse index

 – delete record 40

Deletion from sparse index

– delete record 30

Deletion from sparse index

– delete record 30

Deletion from sparse index

– delete records 30 & 40

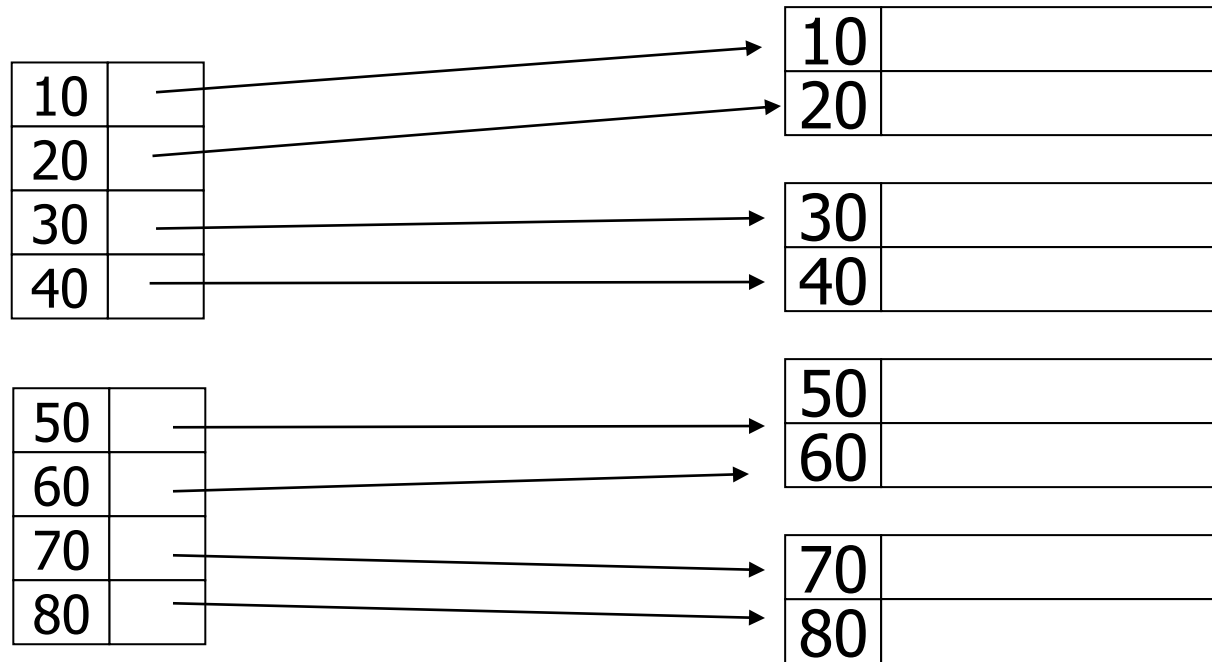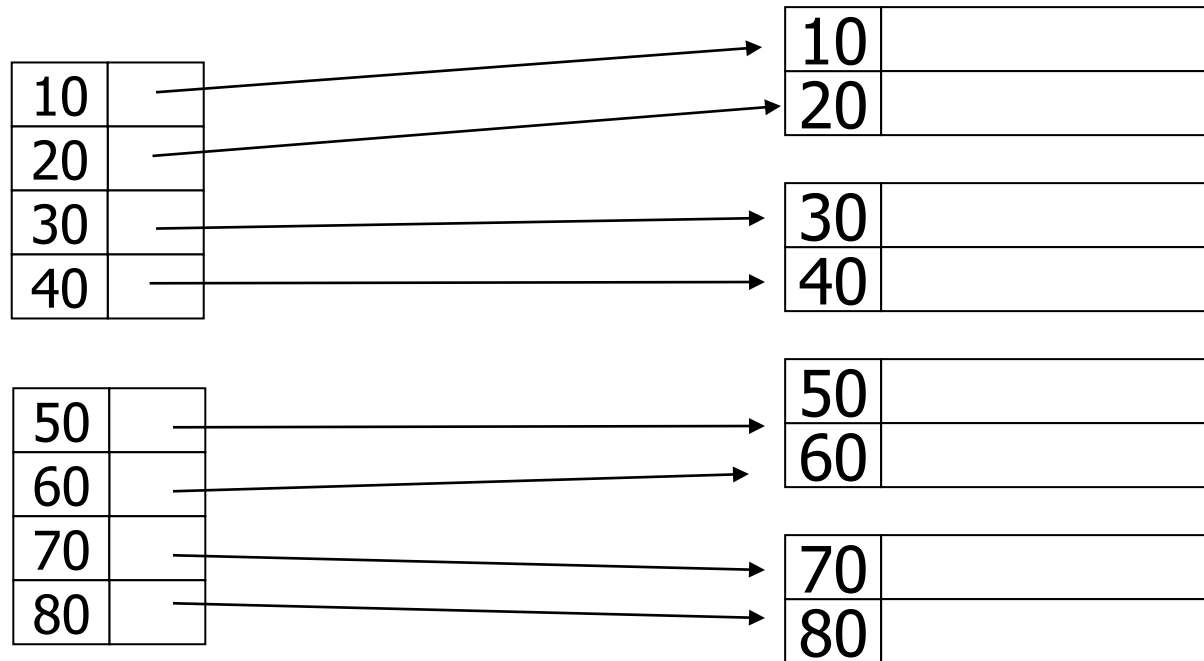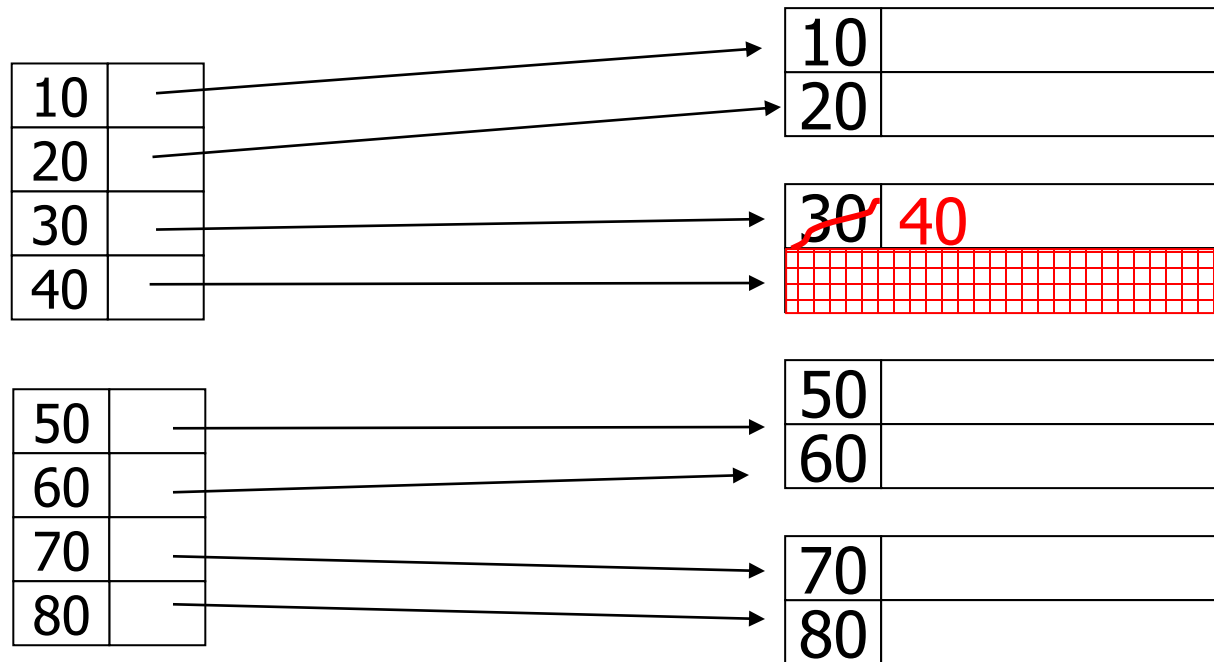# Deletion from sparse index

– delete records 30 & 40

10	
30	
50	
70	

90	
110	
130	
150	

10	
20	

50	
60	

70	
80	
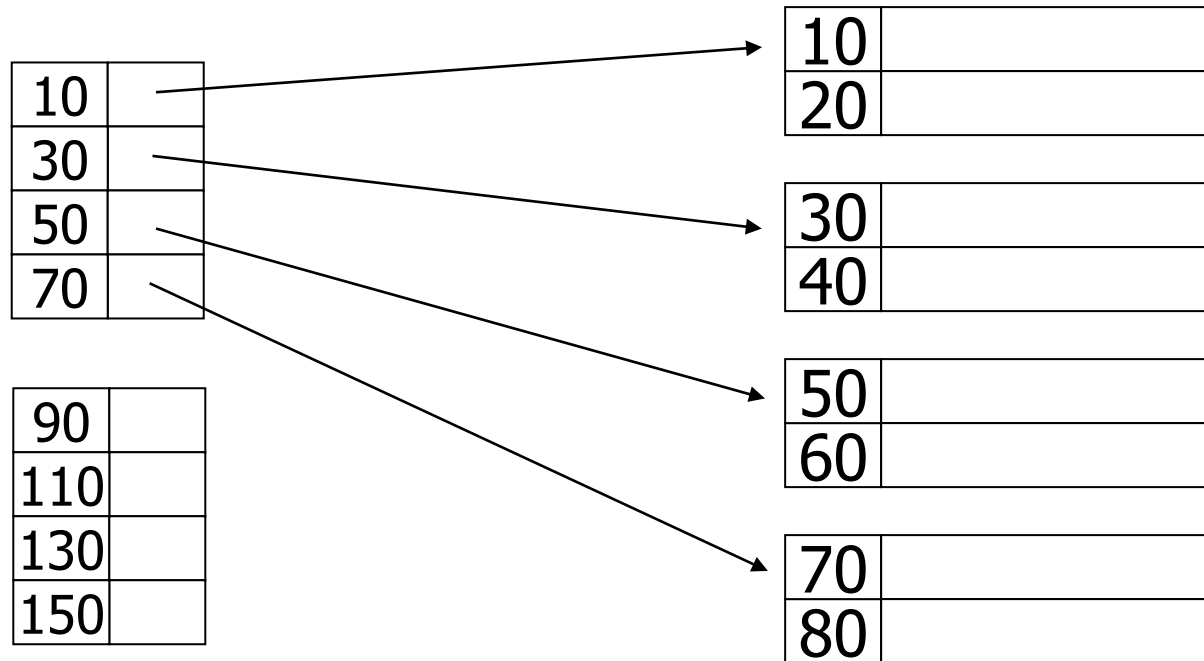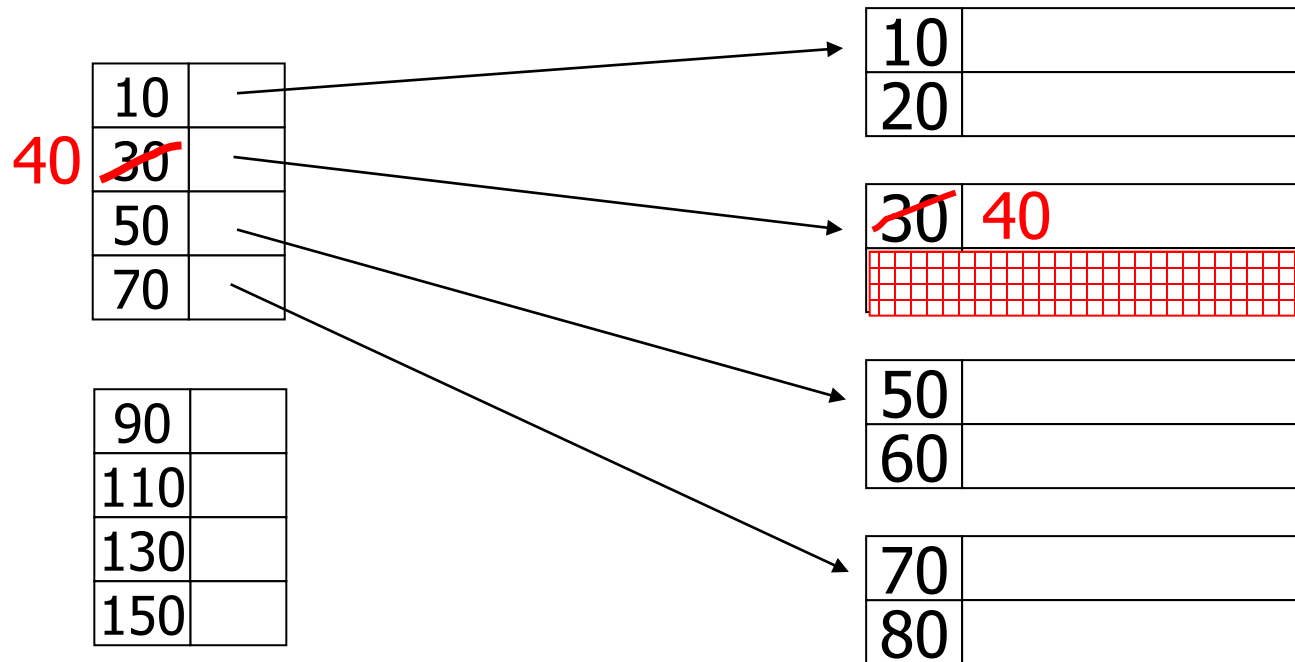
Deletion from sparse index

– delete records 30 & 40

Deletion from dense index

Deletion from dense index

– delete record 30

Deletion from dense index
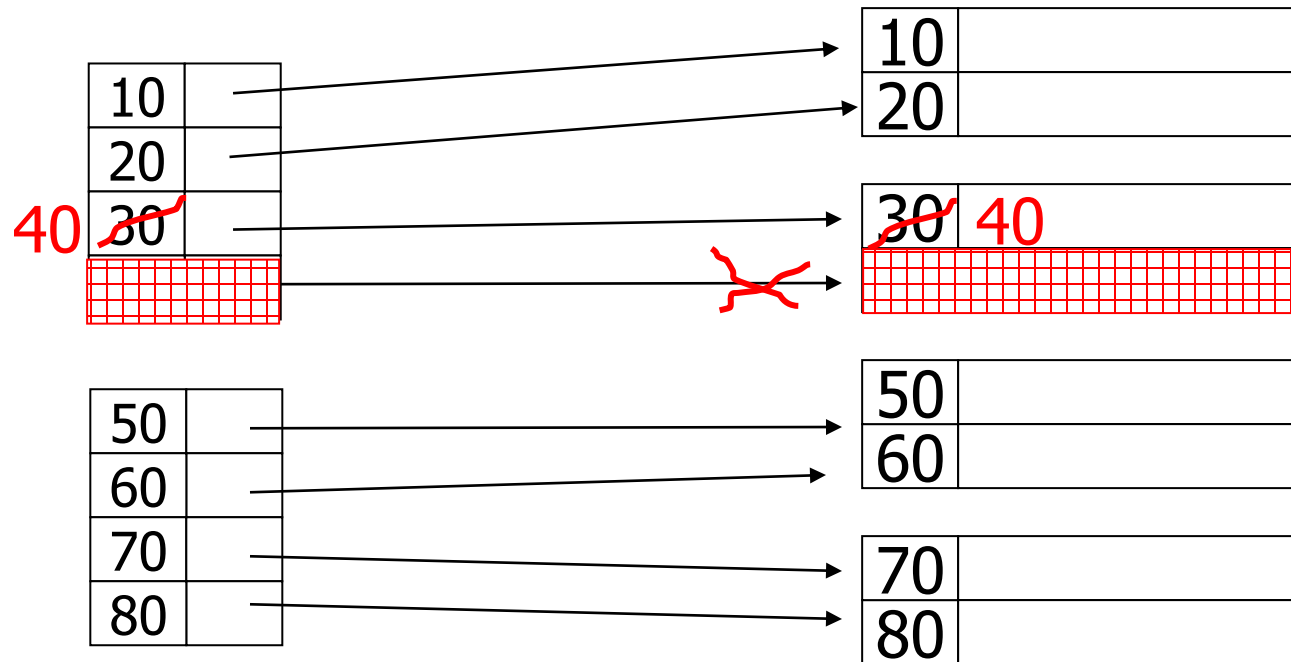
– delete record 30

Deletion from dense index
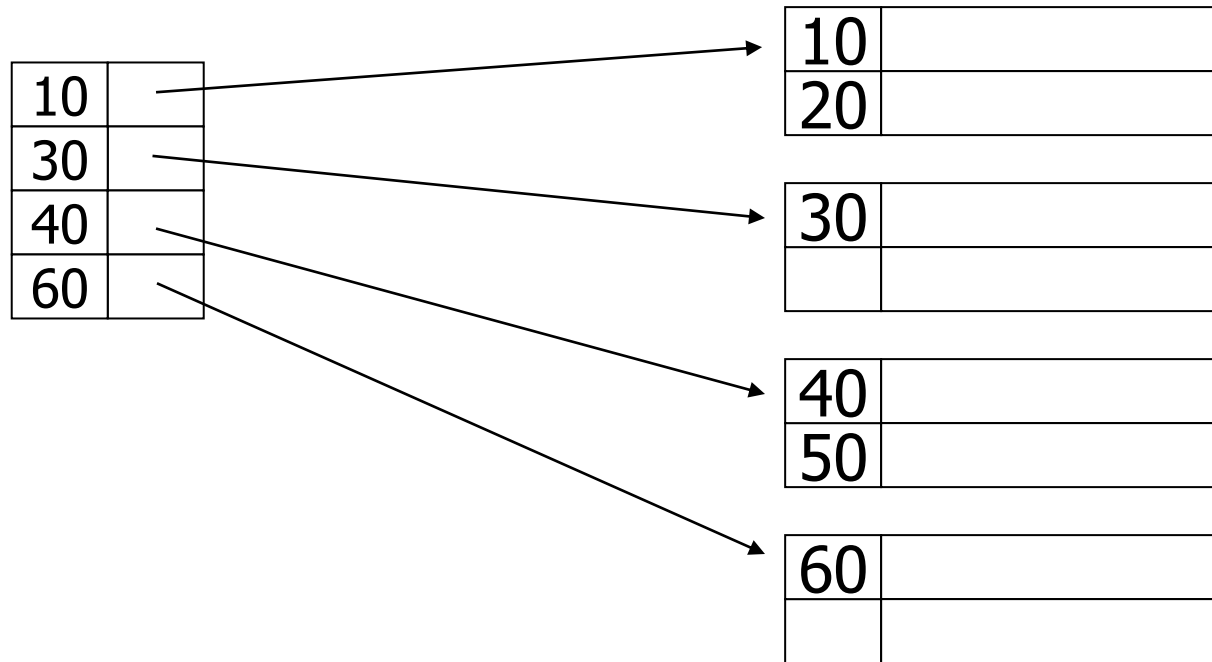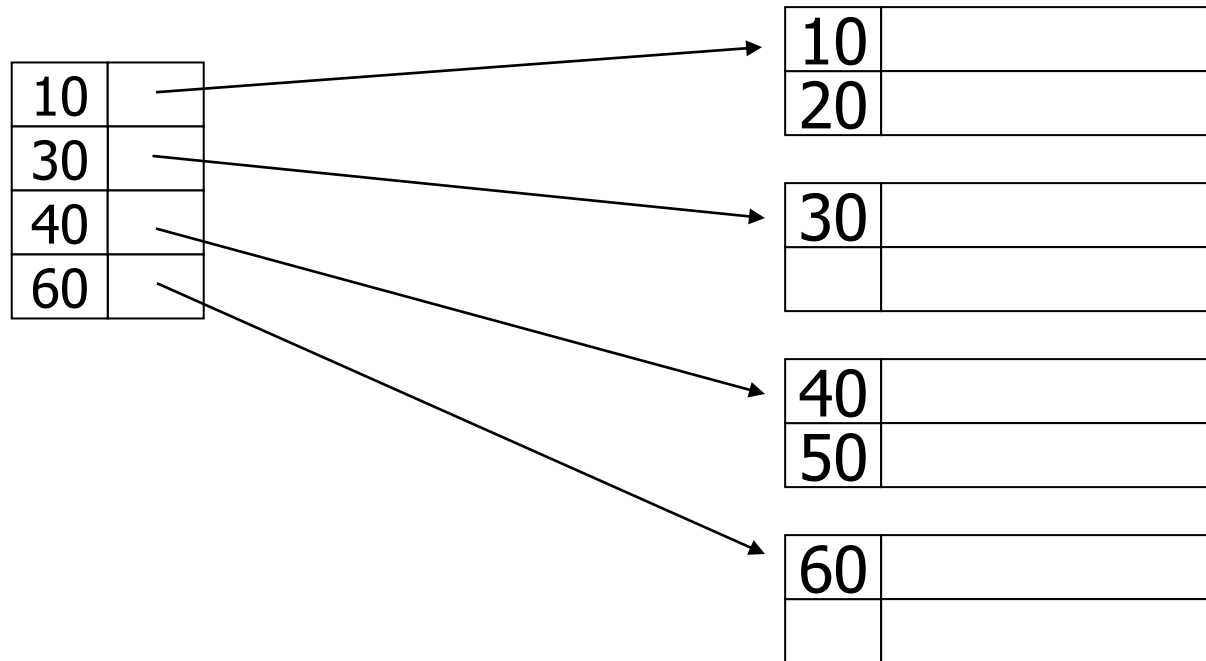
 – delete record 30

10	
20	
40 30̶	

10	
20	

30̶ 40	

50	
60	
70	
80	

50	
60	

70	
80	

Insertion, sparse index case

10	
30	
40	
60	

10	
20	

30	

40	
50	

60	

Insertion, sparse index case

– insert record 34

10	
30	
40	
60	

10	
20	

30	

40	
50	

60	

Insertion, sparse index case

– insert record 34

10	
30	
40	
60	

10	
20	

30	
34	

40	
50	

60	
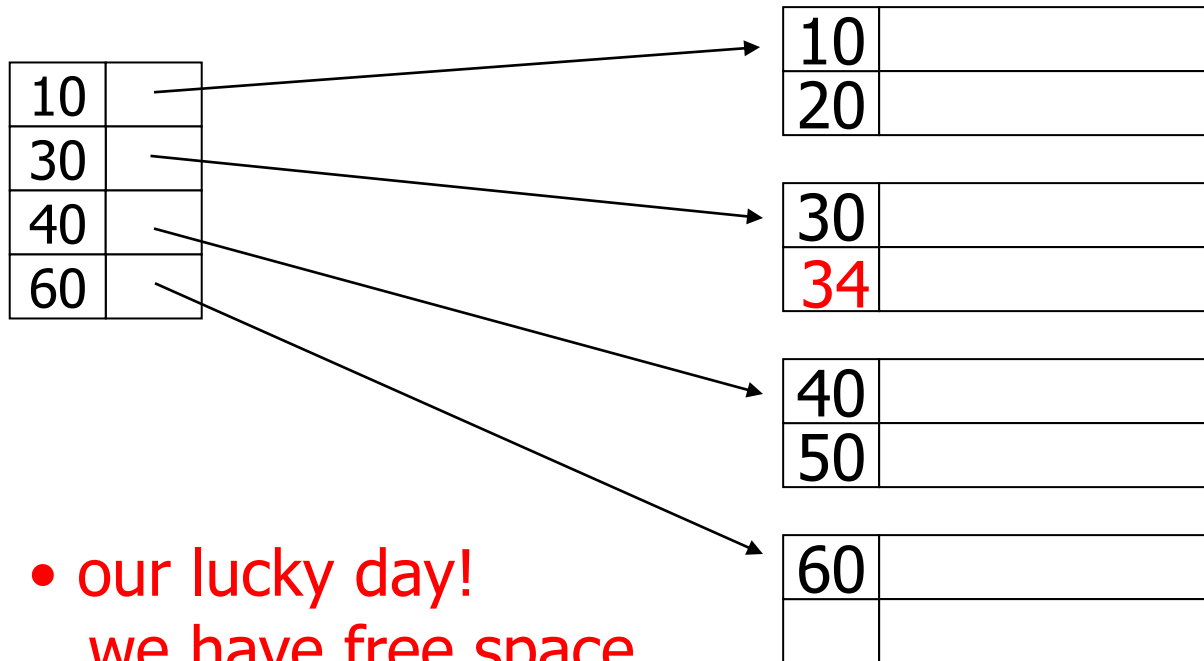
• our lucky day!
  we have free space
  where we need it!

Insertion, sparse index case
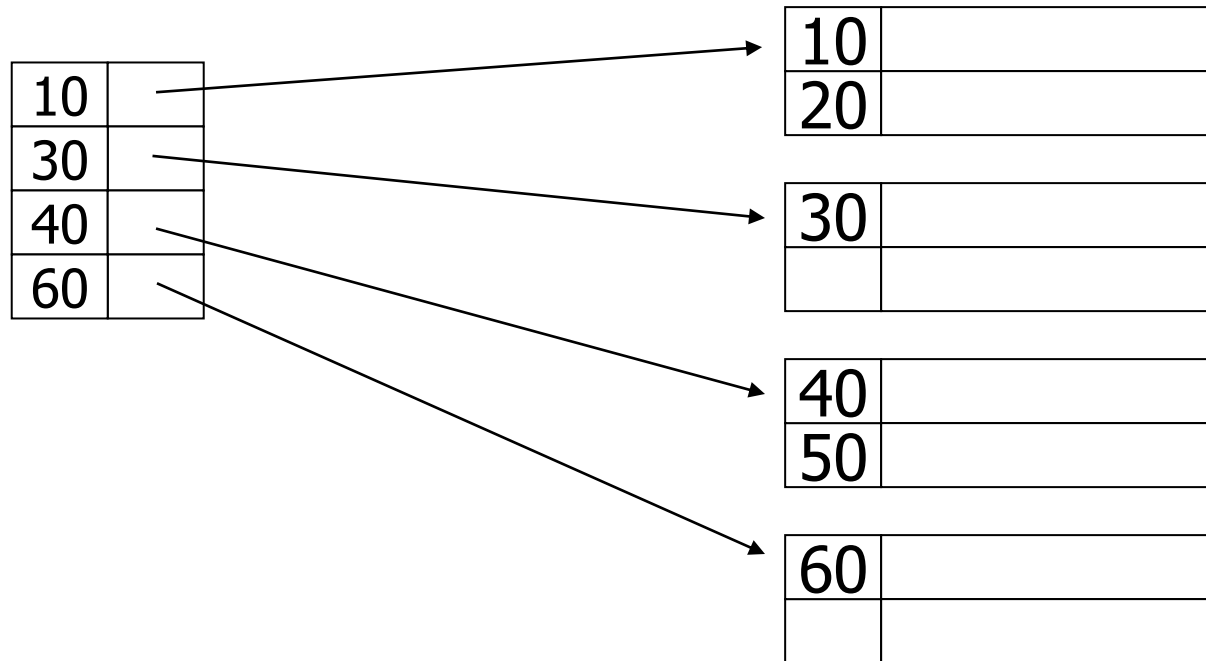
– insert record 15

10	
30	
40	
60	

10	
20	

30	

40	
50	

60	

Insertion, sparse index case
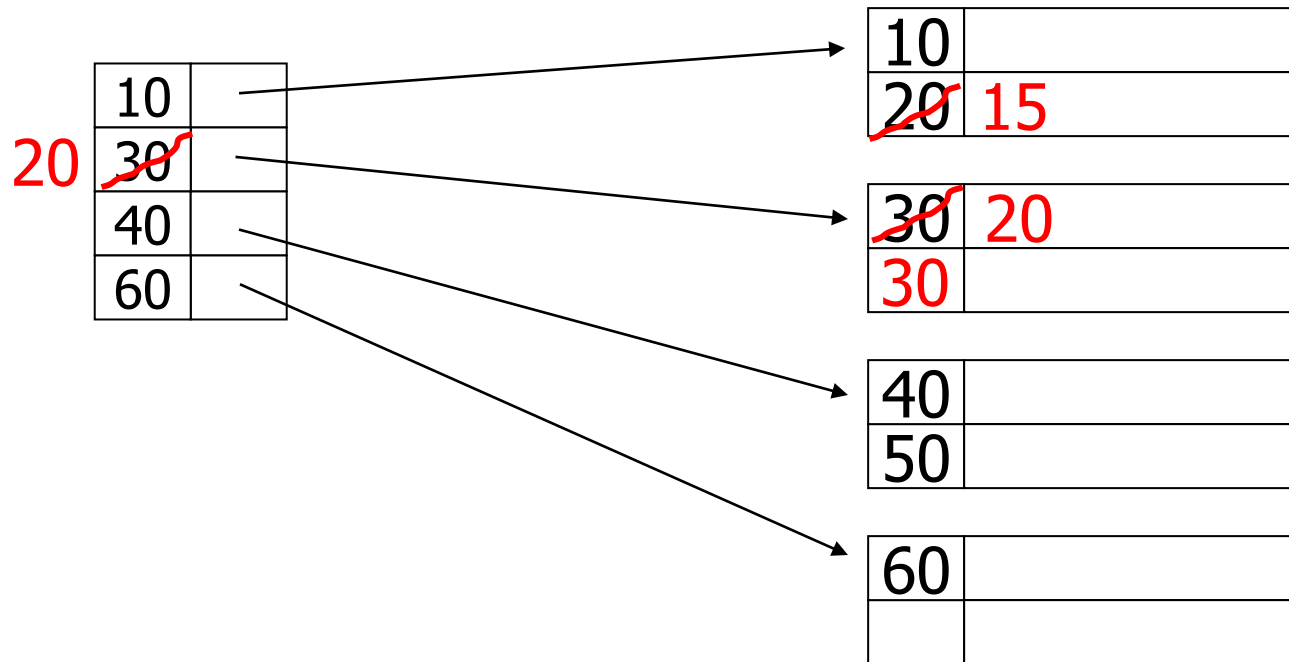
– insert record 15

Insertion, sparse index case
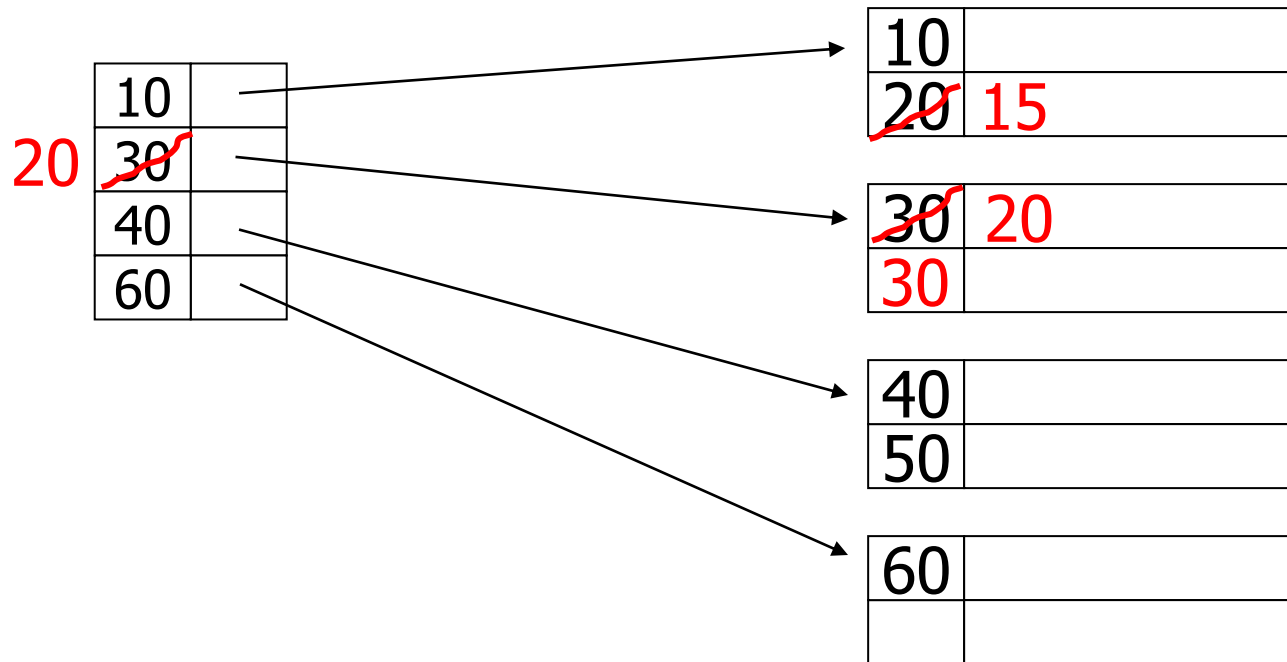
– insert record 15

Insertion, sparse index case

   – insert record 25
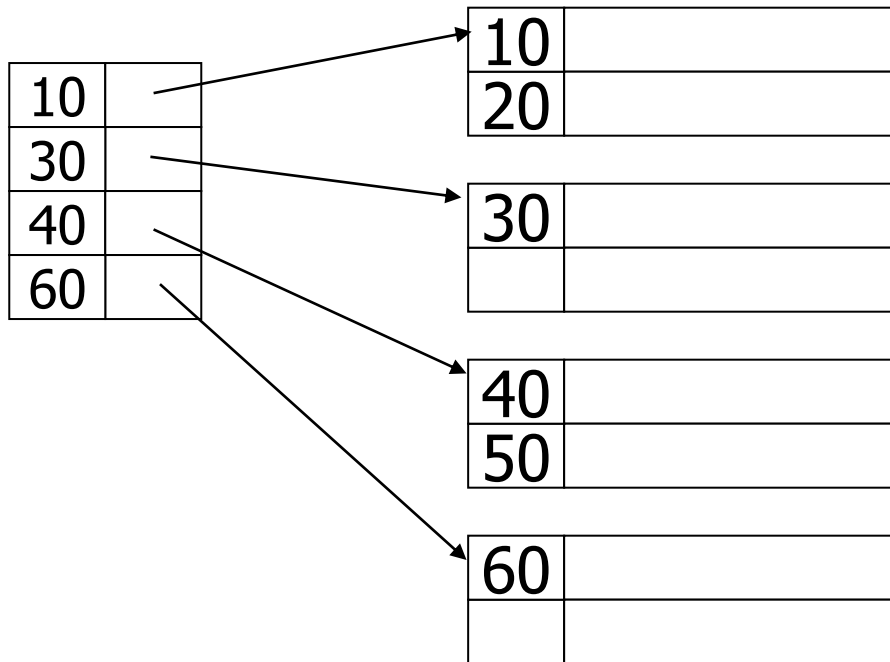
Insertion, dense index case

- Similar

- Often more expensive . . .

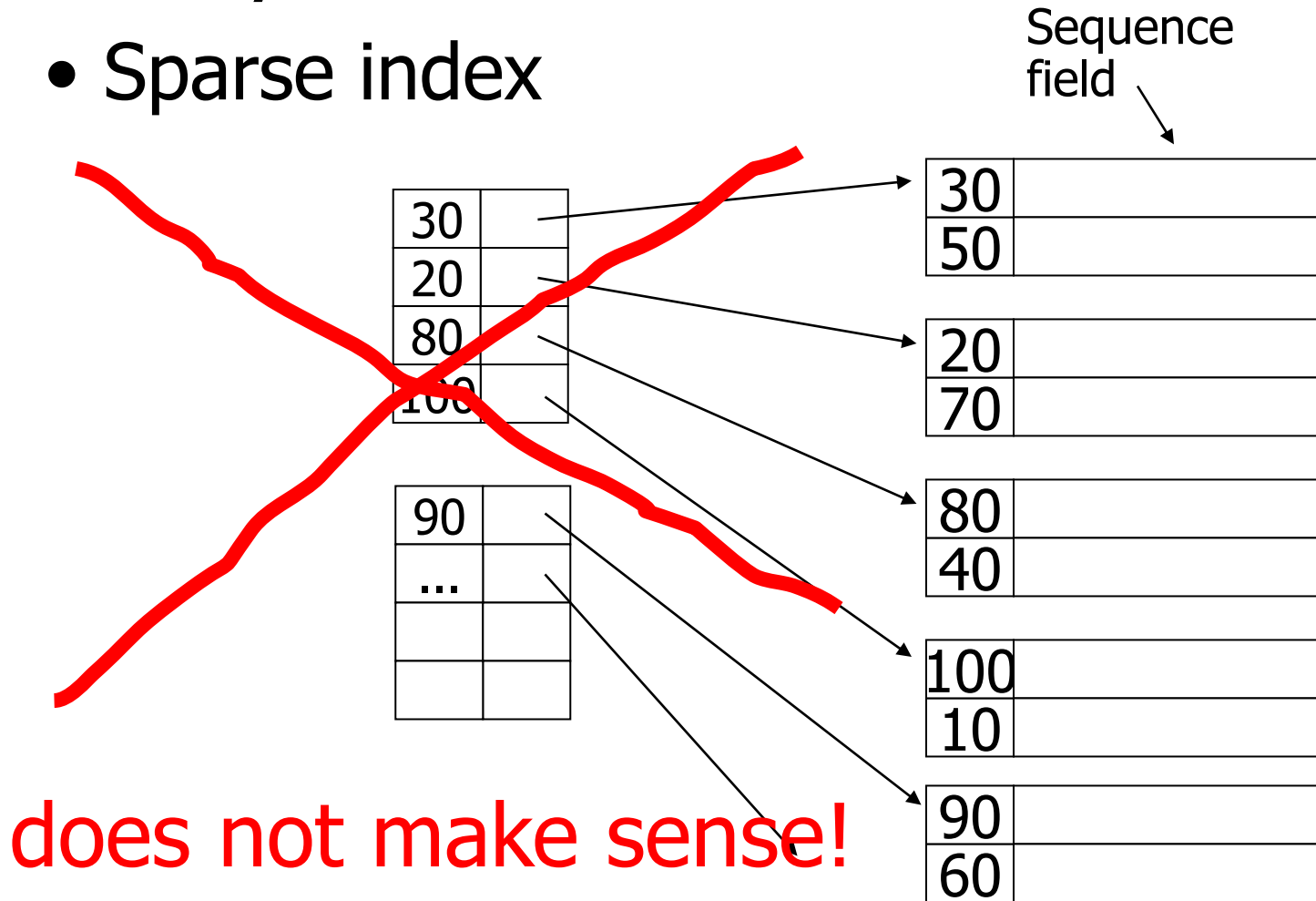# Secondary indexes

Sequence
field

| 30 | |
| 50 | |

| 20 | |
| 70 | |

| 80 | |
| 40 | |

| 100 | |
| 10 | |

| 90 | |
| 60 | |
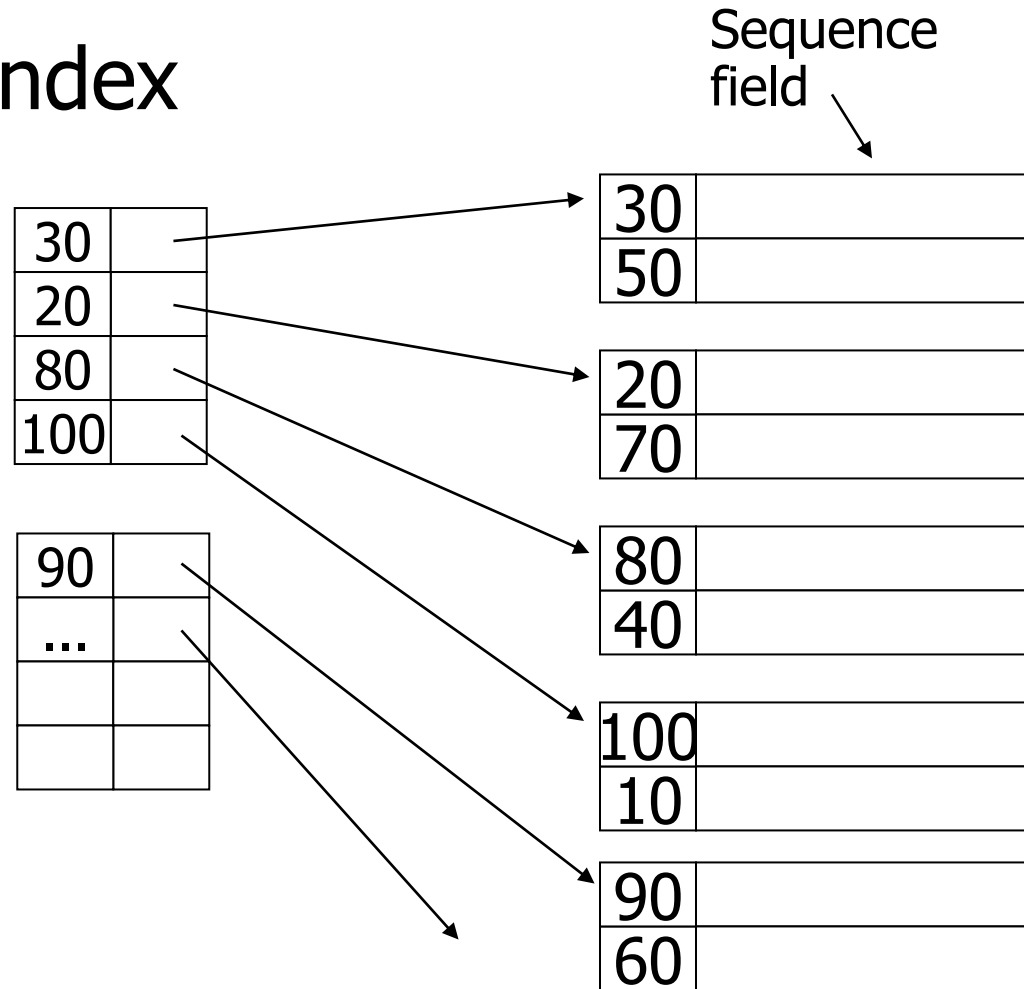
Secondary indexes

- Sparse index

Sequence field

30	
20	
80	
100	

90	
...	

30	
50	

20	
70	

80	
40	

100	
10	

90	
60	

Secondary indexes

- Sparse index

Sequence field

30	
20	
80	
100	

90	
...	

| 30 | |
| 50 | |

| 20 | |
| 70 | |

| 80 | |
| 40 | |

| 100 | |
| 10 | |

| 90 | |
| 60 | |

does not make sense!

Secondary indexes

- Dense index

Sequence field

| 30 | |
| 50 | |

| 20 | |
| 70 | |

| 80 | |
| 40 | |

| 100 | |
| 10 | |

| 90 | |
| 60 | |

Secondary indexes

- Dense index

Sequence field

10	
20	
30	
40	

50	
60	
70	
...	

| 30 | |
| 50 | |

| 20 | |
| 70 | |

| 80 | |
| 40 | |

| 100 | |
| 10 | |

| 90 | |
| 60 | |

37

Secondary indexes

- Dense index

Sequence field

10	
20	
30	
40	

50	
60	
70	
...	

10	
50	
90	
...	

sparse
high
level

30	
50	

20	
70	

80	
40	

100	
10	

90	
60	
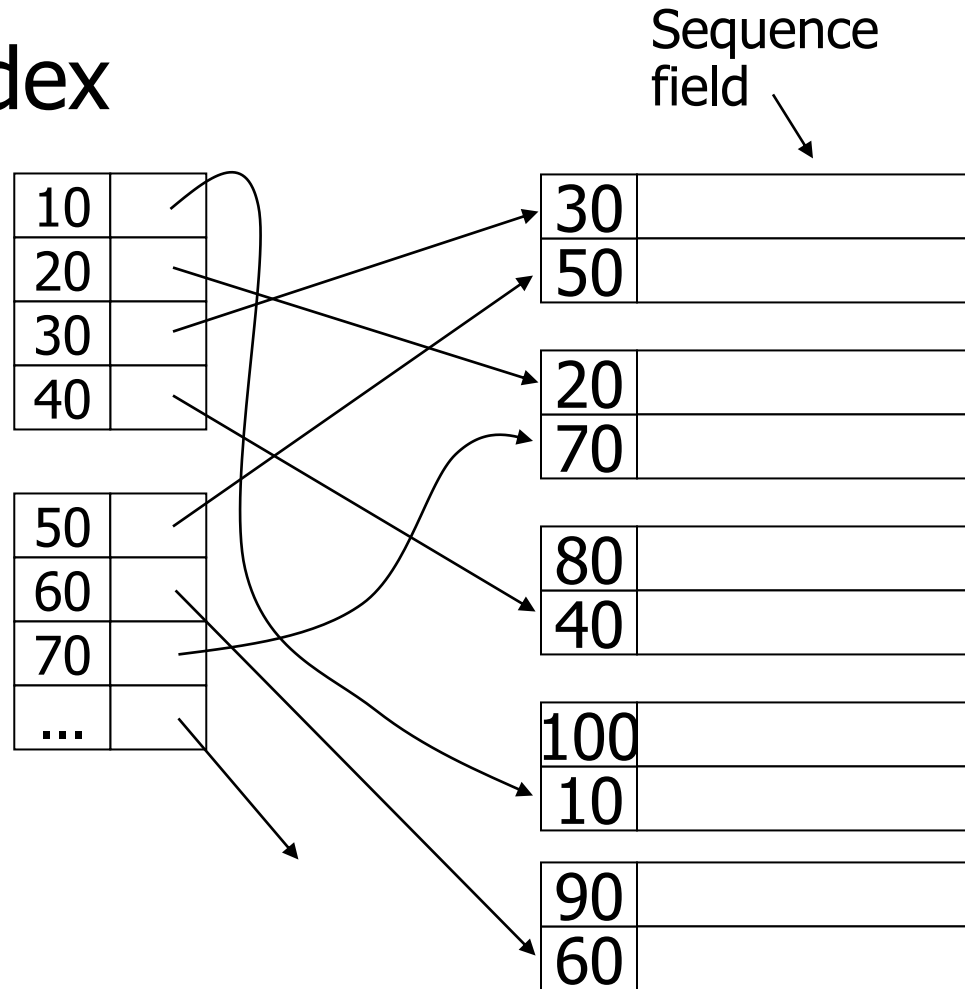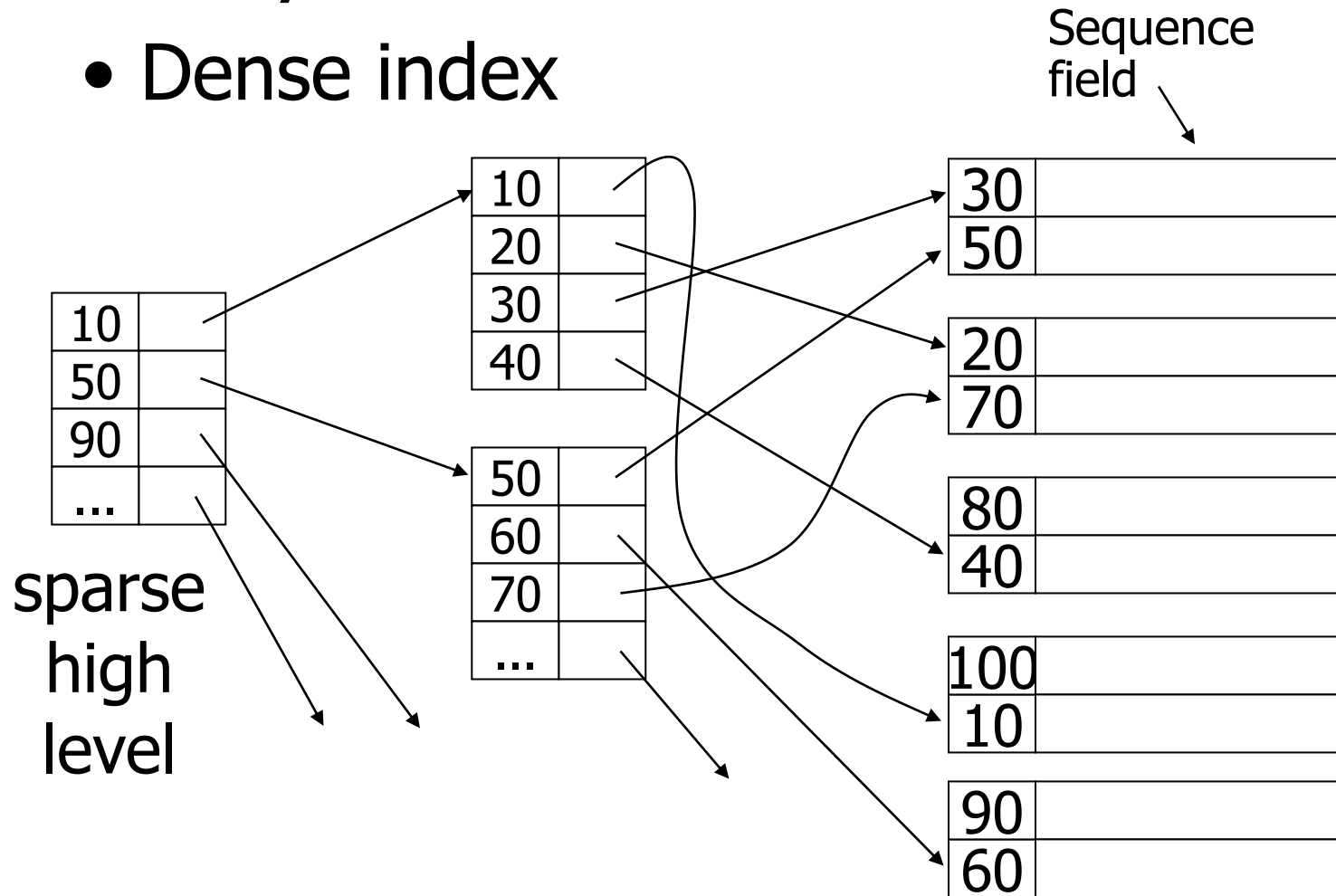
With secondary indexes:

- Lowest level is dense
- Other levels are sparse

Duplicate values & secondary indexes

20	
10	

20	
40	

10	
40	

10	
40	

30	
40	

Duplicate values & secondary indexes
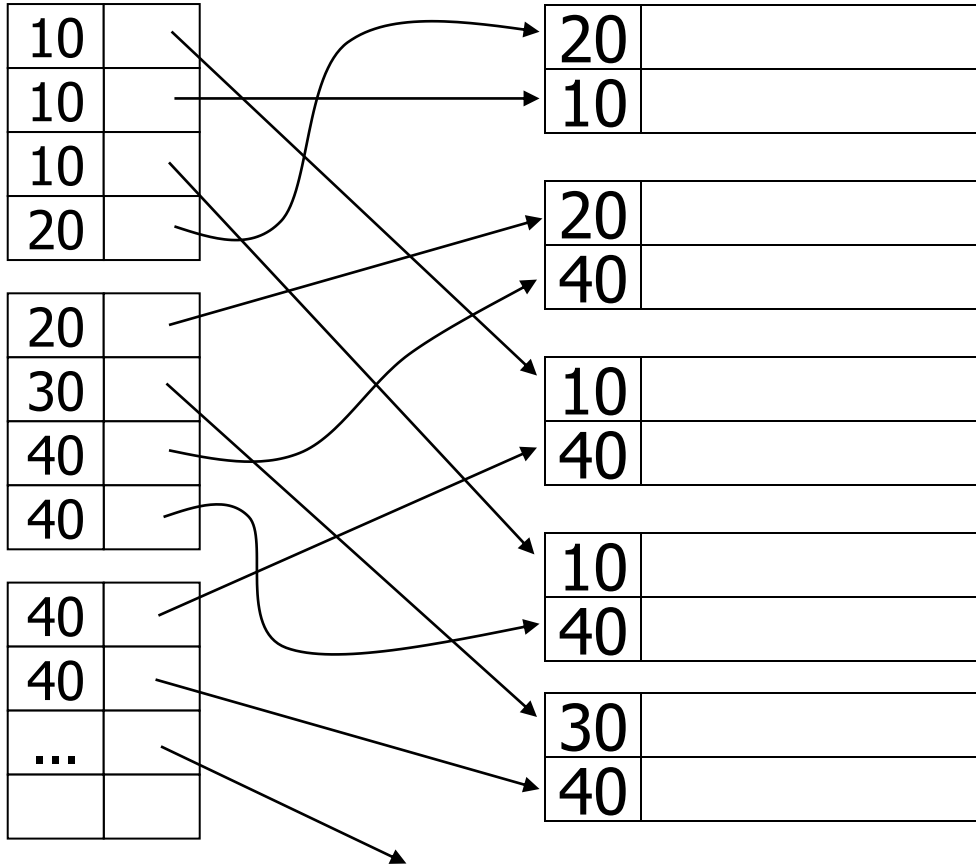
one option…

Duplicate values & secondary indexes

one option...

Problem:
excess overhead!
- disk space
- search time

10	
10	
10	
20	

20	
30	
40	
40	

40	
40	
...	

20	
10	

20	
40	

10	
40	

10	
40	

30	
40	
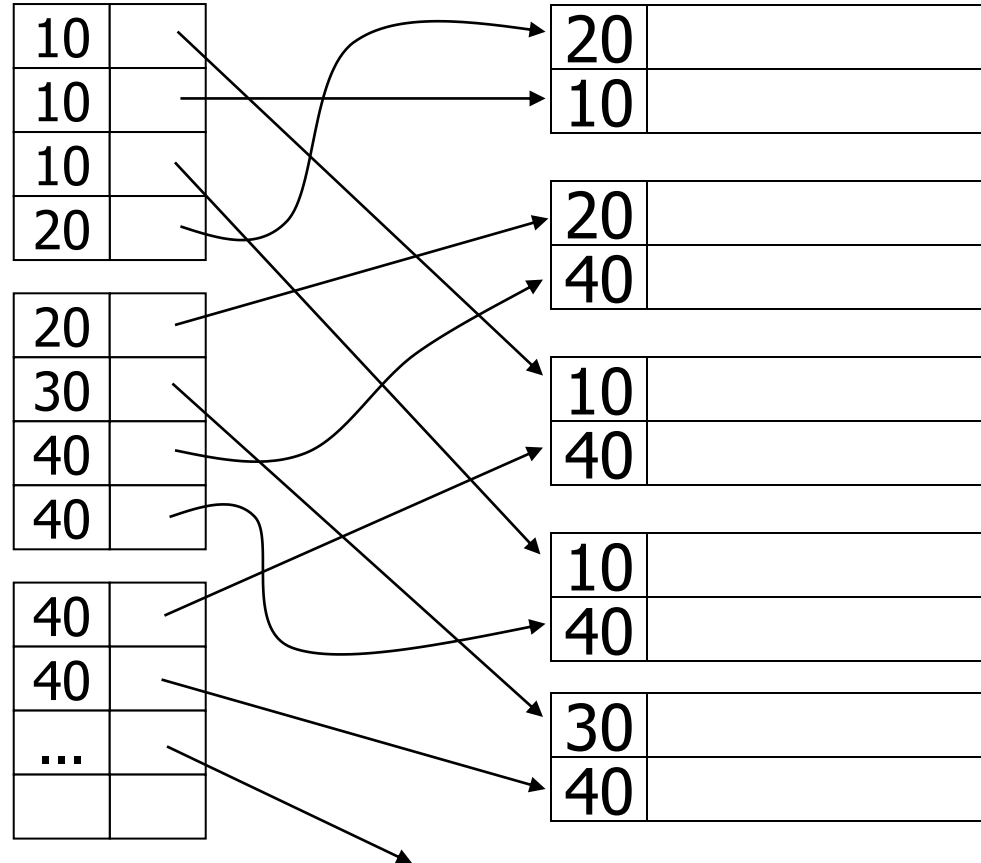
Duplicate values & secondary indexes
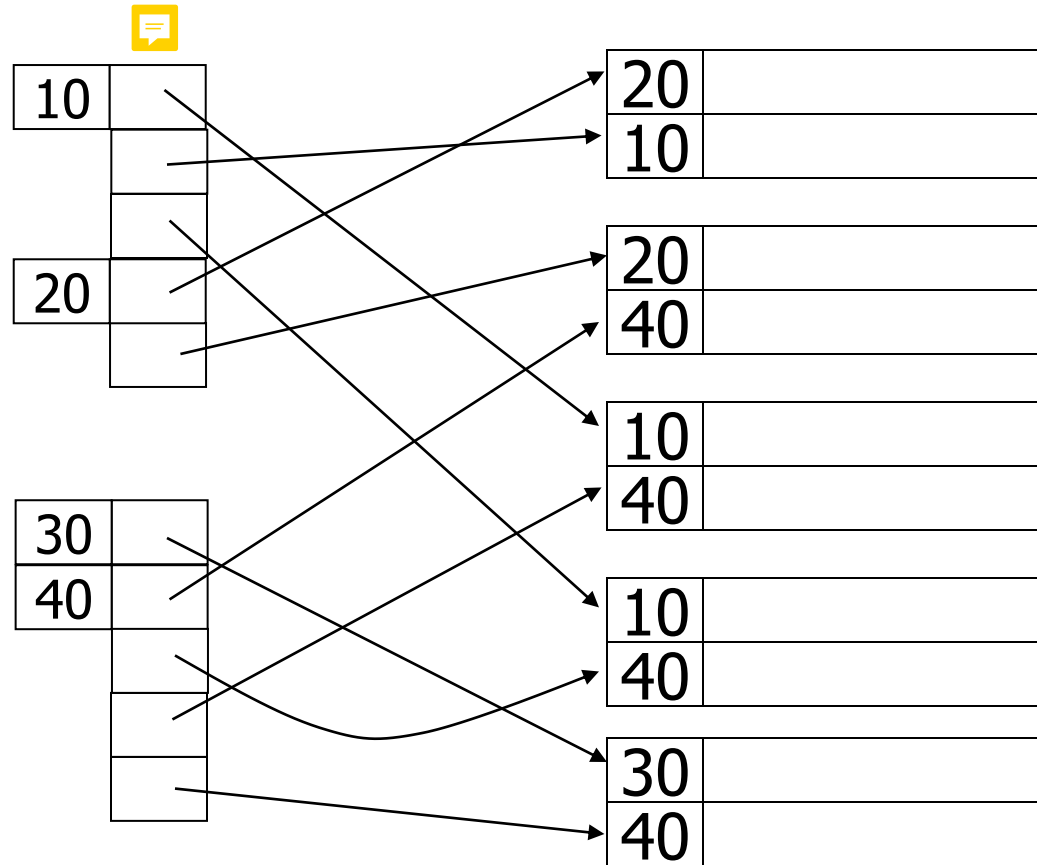
another option…

Duplicate values & secondary indexes

another option…

Problem: variable size records in index!
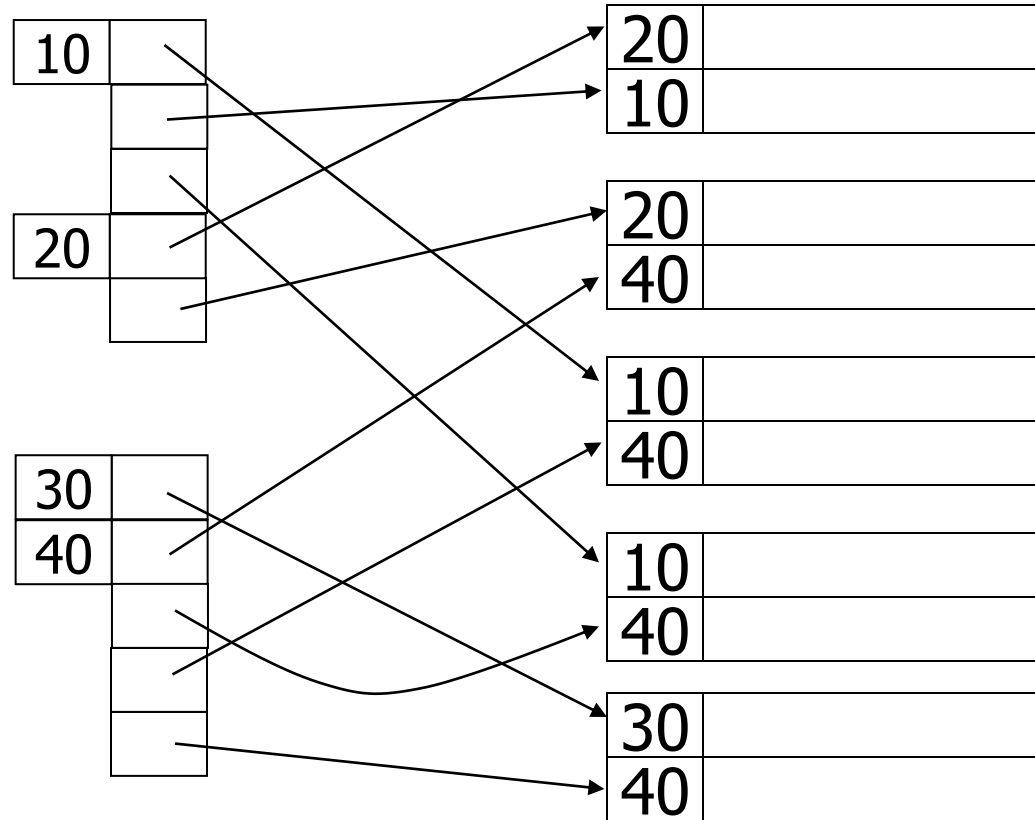
Duplicate values & secondary indexes
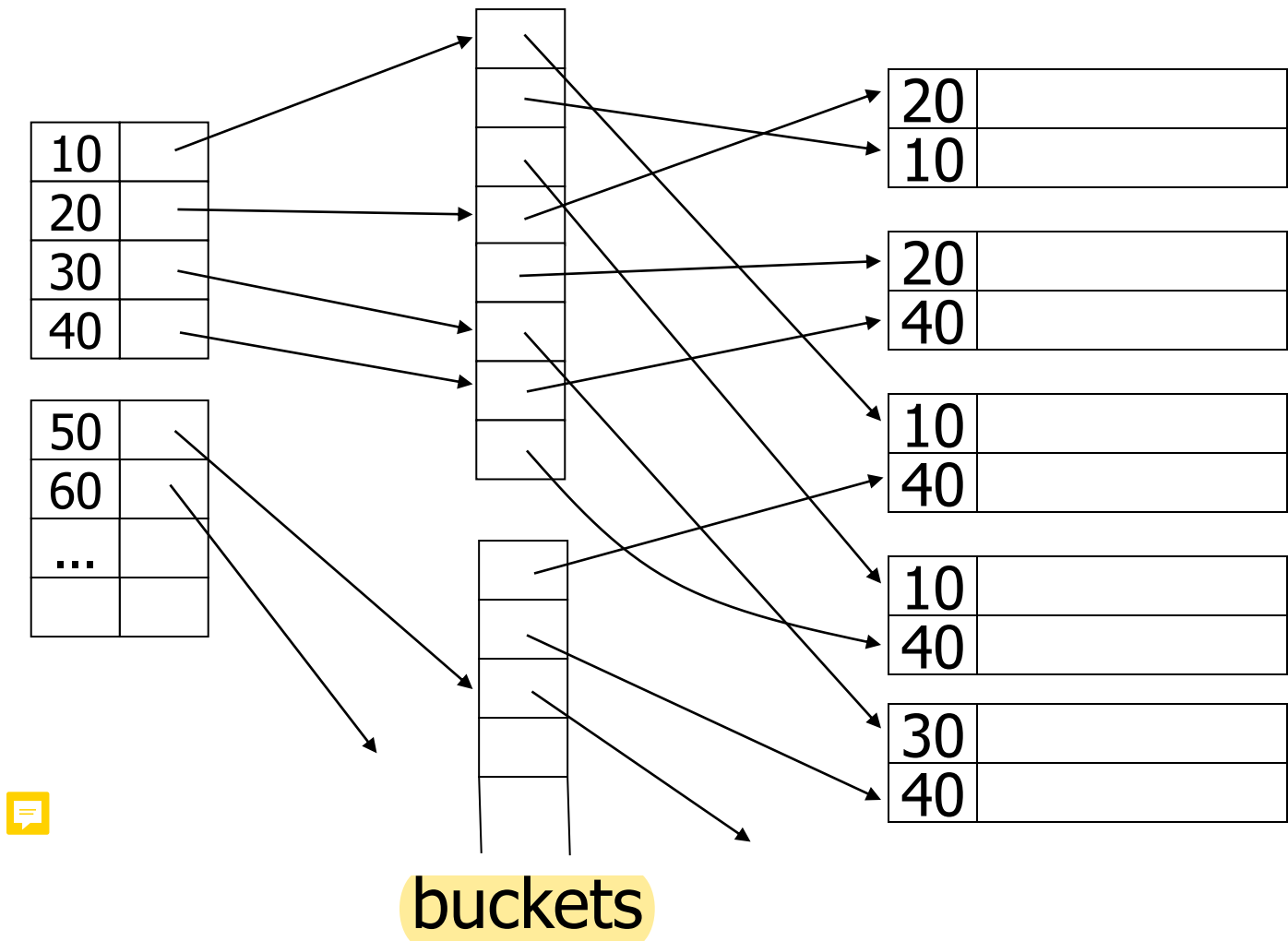
buckets

Why "bucket" idea is useful

Indexes	Records
Name: primary	EMP (name,dept,floor,...)
Dept: secondary	
Floor: secondary	

Query: Get employees in
(Toy Dept) ∧ (2nd floor)

Dept. index   EMP   Floor index

Toy   2nd

Query: Get employees in
(Toy Dept) ∧ (2nd floor)

Dept. index   EMP   Floor index

Toy   2nd
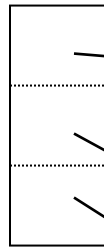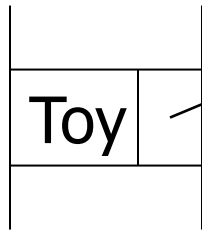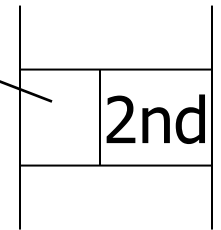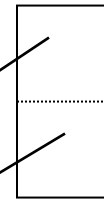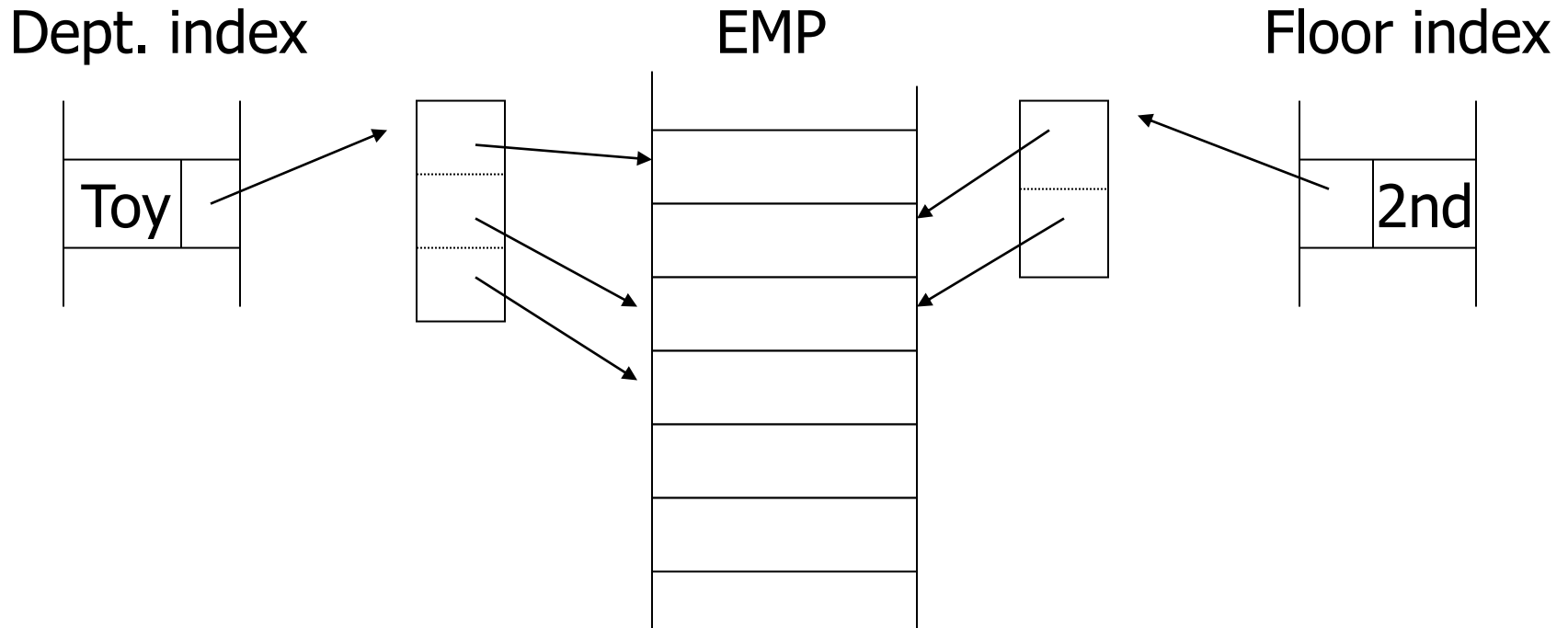
→ Intersect toy bucket and 2nd Floor
  bucket to get set of matching EMP's

Summary so far

- Conventional index
  - Basic Ideas: sparse, dense, multi-level...
  - Duplicate Keys
  - Deletion/Insertion
  - Secondary indexes

Conventional indexes

Advantage:

       - Simple
       - Index is sequential file
              good for scans

Disadvantage:

       - Inserts expensive, or
       - Lose sequentiality & balance

Example

Index (sequential)

10	→
20	→
30	→

continuous

40	→
50	→
60	→

free space

70	→
80	→
90	→

Example   Index (sequential)

10	→
20	→
30	→
33	

continuous

40	→
50	→
60	→

free space

70	→
80	→
90	→

39	
31	
35	
36	

32	
38	
34	
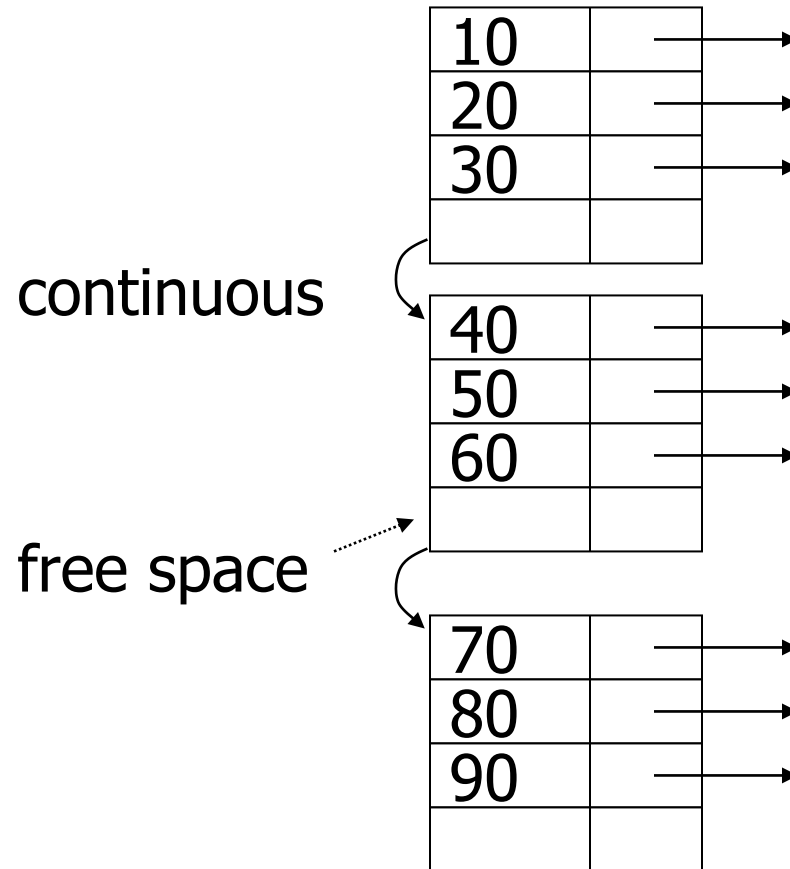
overflow area
(not sequential)
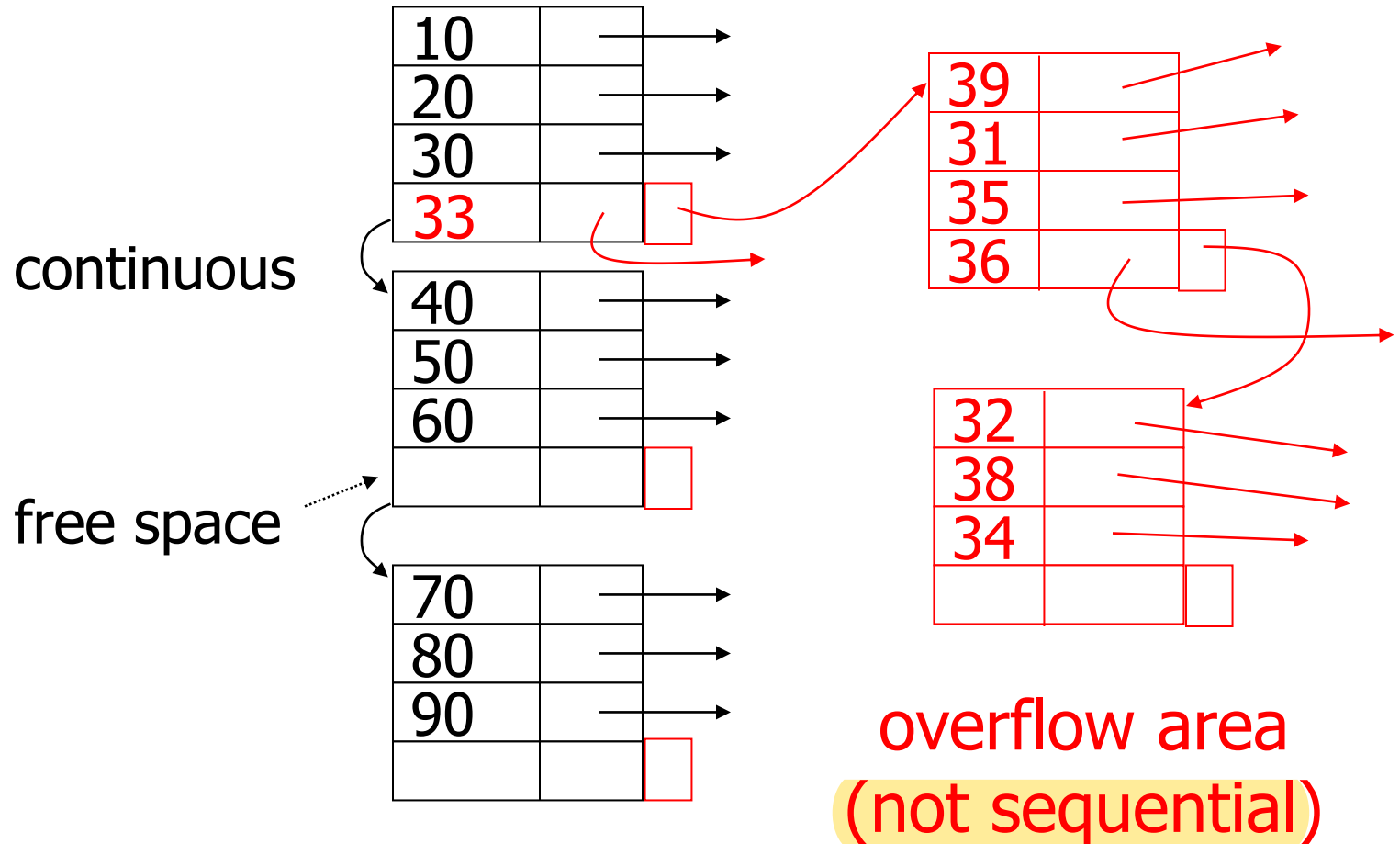
Outline:

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

- NEXT: Another type of index
  - Give up on sequentiality of index
  - Try to get "balance"

B+Tree Example

n=3
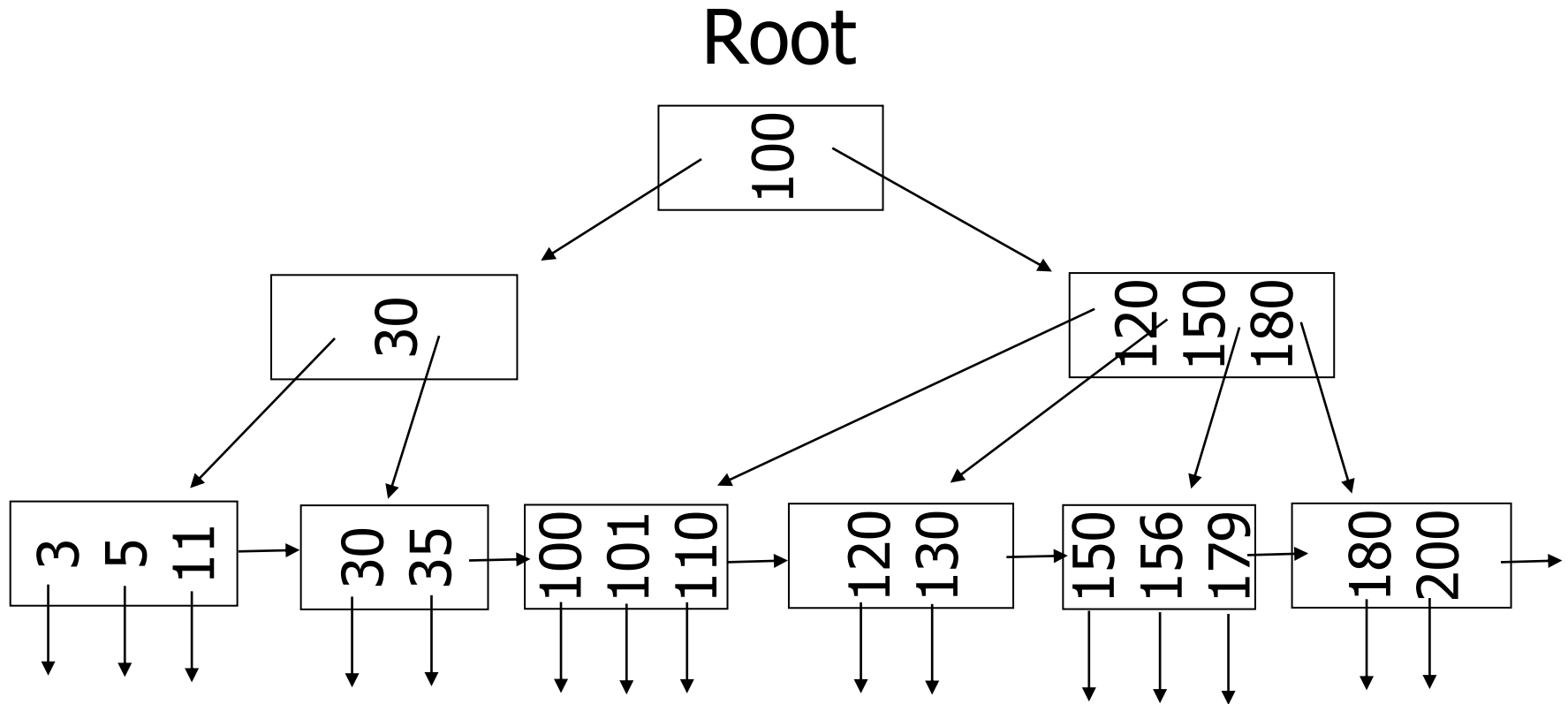
Root

| 100 |

| 30 |

| 120 | 150 | 180 |

| 3 | 5 | 11 | → | 30 | 35 | → | 100 | 101 | 110 | → | 120 | 130 | → | 150 | 156 | 179 | → | 180 | 200 | →

Sample non-leaf

57 81 95

to keys   to keys   to keys   to keys

< 57   57≤ k<81   81≤k<95   ≥95

Sample leaf node:

From non-leaf node

| 57 | 81 | 95 | → to next leaf in sequence |

To record with key 57

To record with key 81

To record with key 85
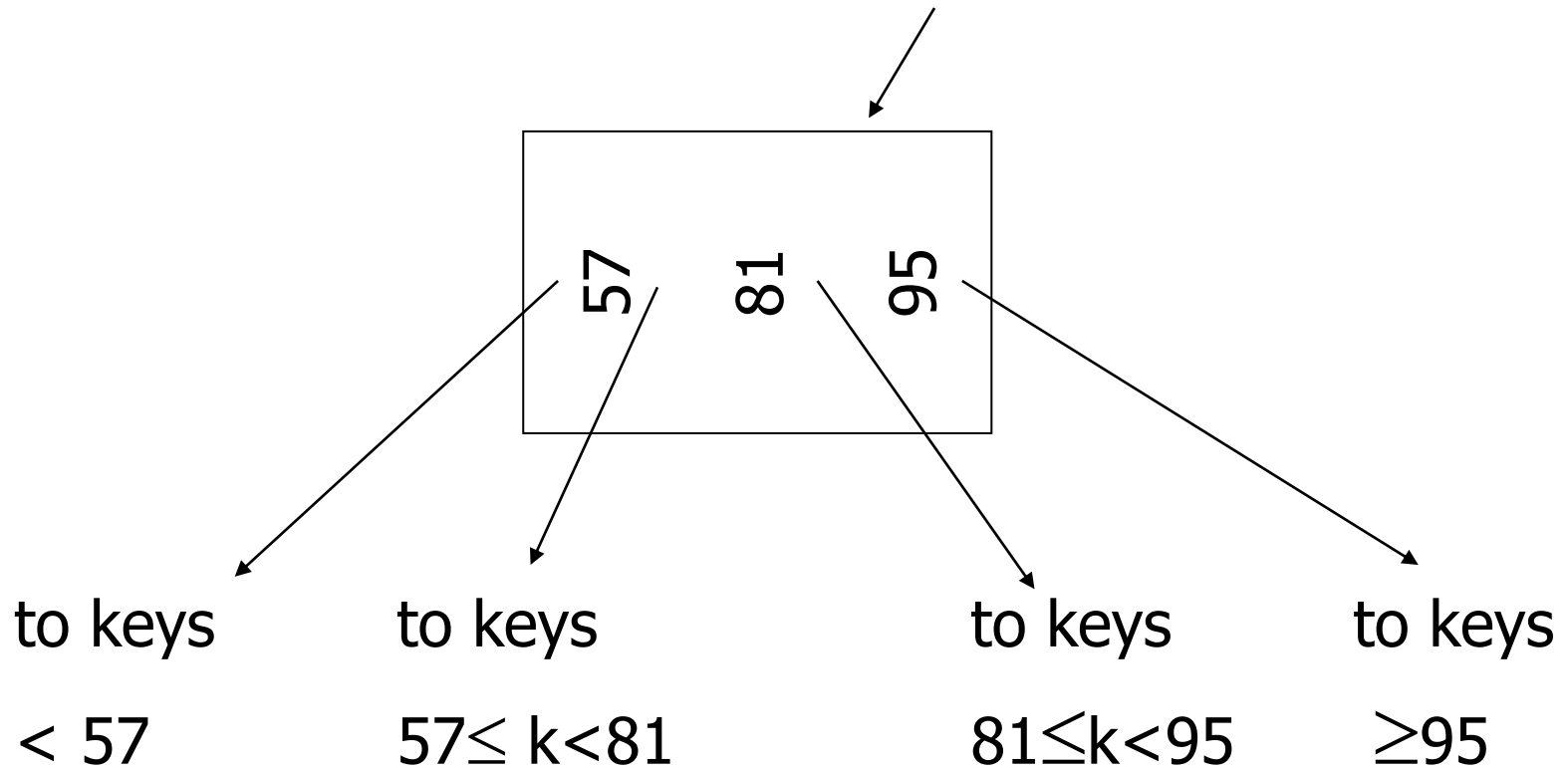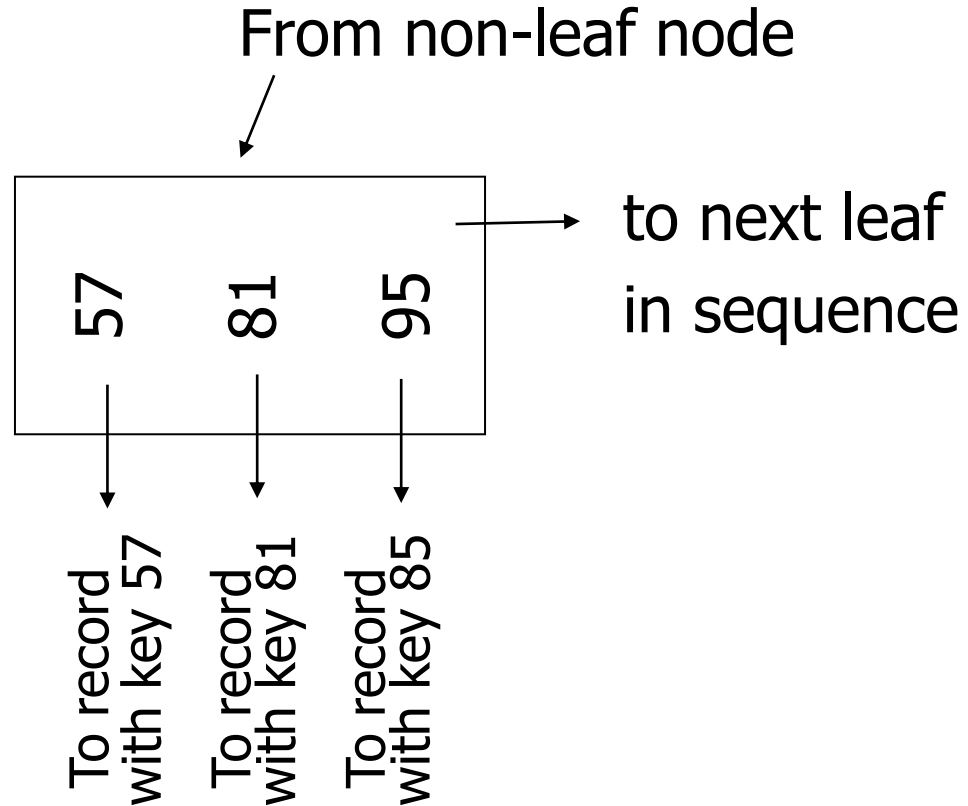
Size of nodes: n+1 pointers

n keys   <u>(fixed)</u>

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

| 120 | 150 | 180 |

30

Leaf

| 3 | 5 | 11 |

| 30 | 35 |
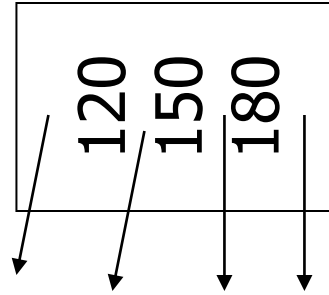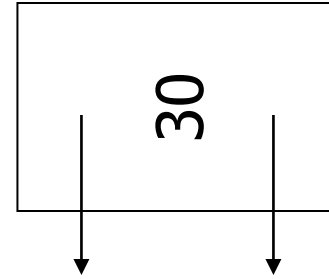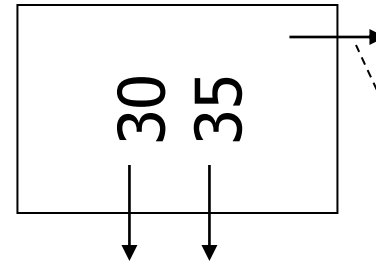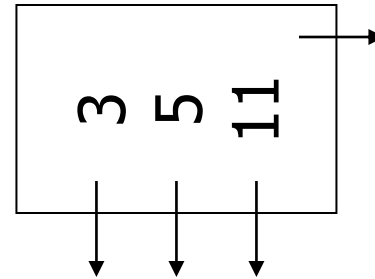
counts even if null

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	1	1

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

100

30

3
5
11

30
31
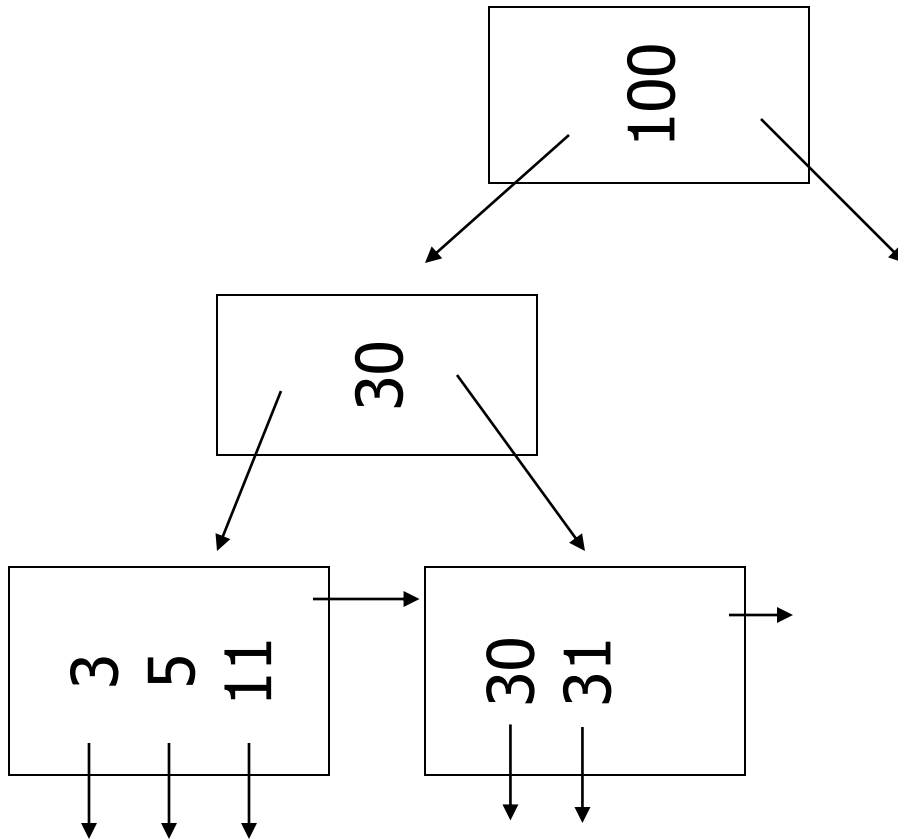
(a) Insert key = 32
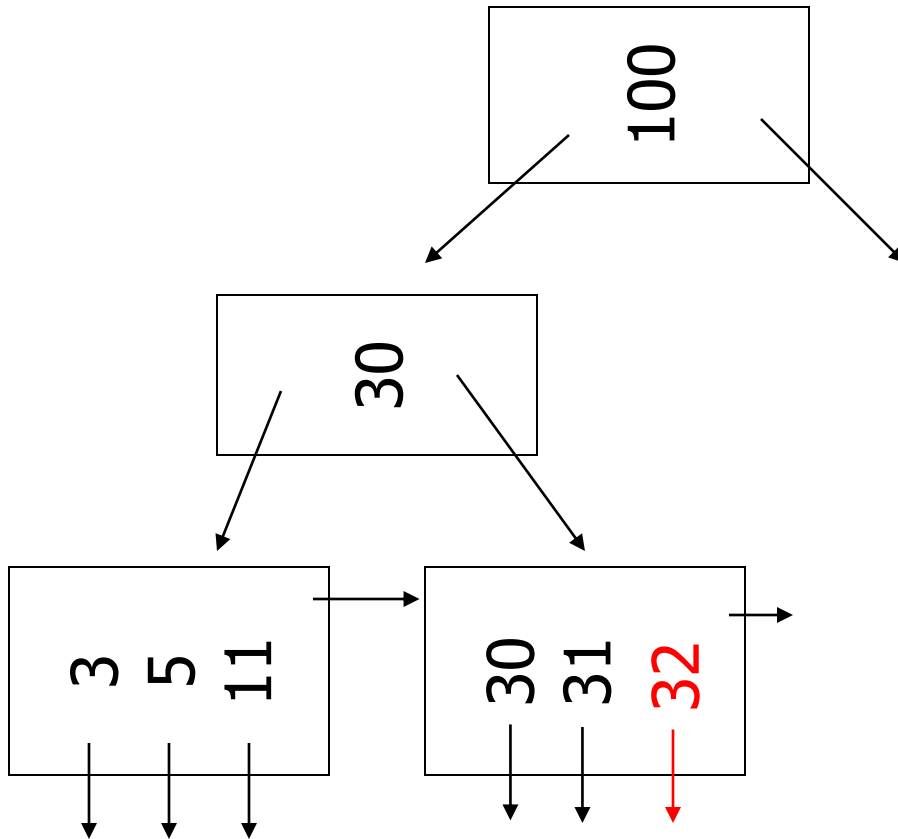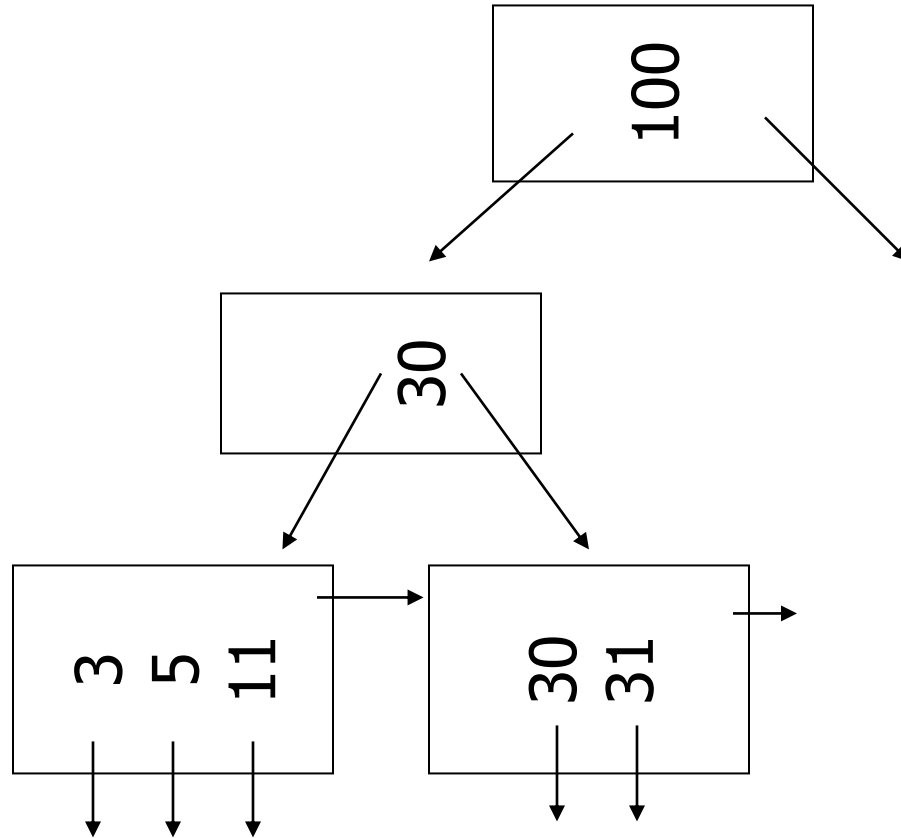
(a) Insert key = 7

(a) Insert key = 7
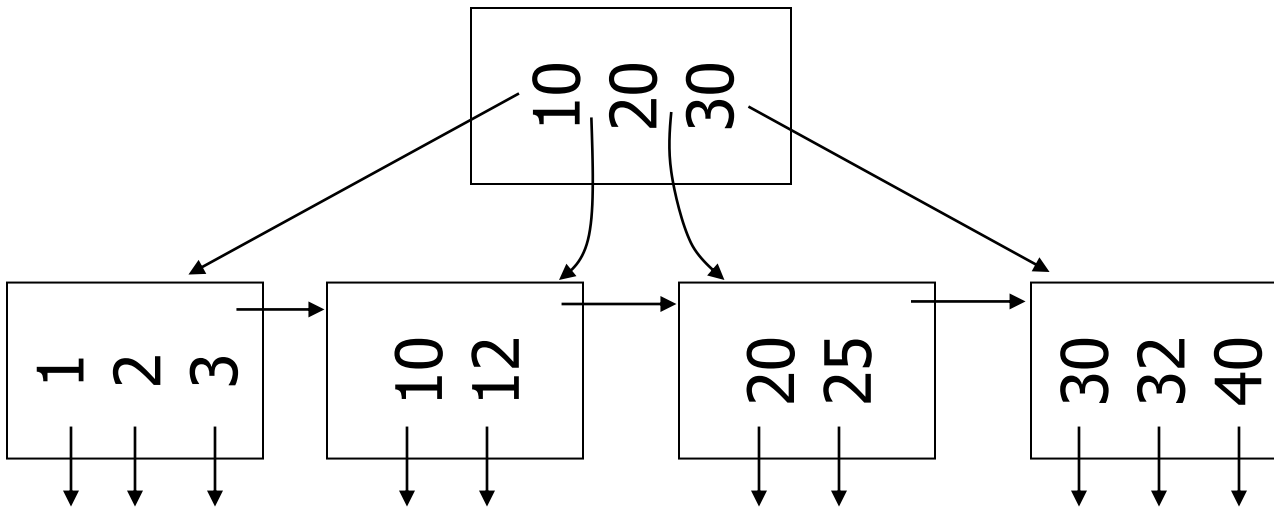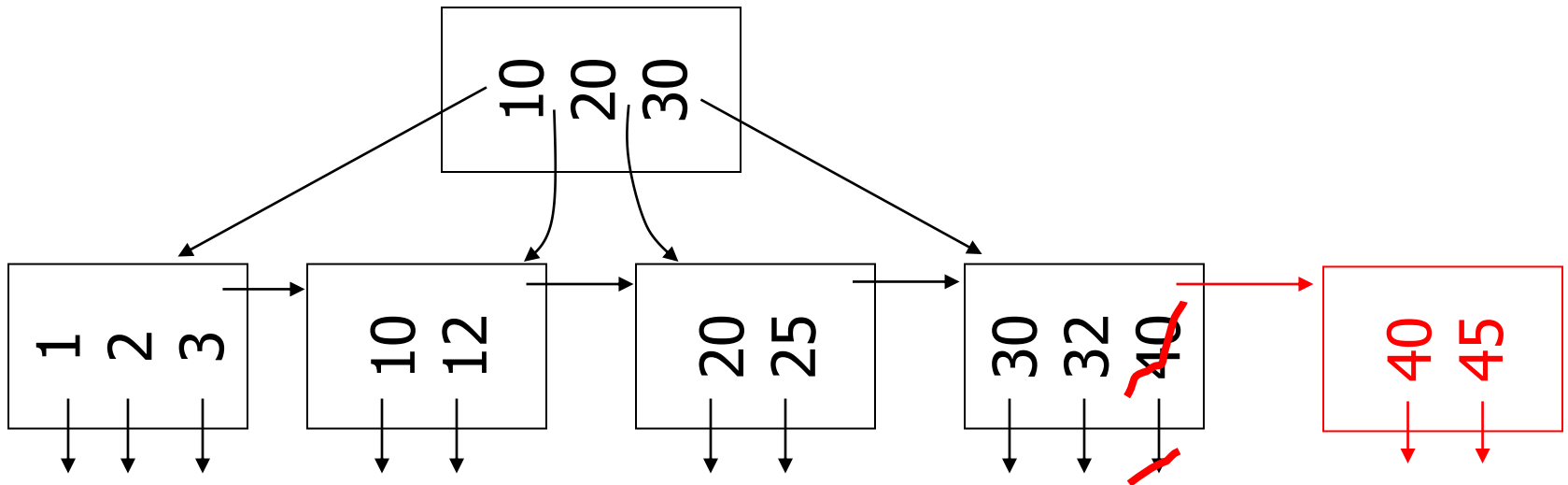
n=3

(a) Insert key = 7

n=3

68

(c) Insert key = 160

n=3

100

120 150 180

150 156 179

180 200

(c) Insert key = 160

n=3

100

120 150 180

150 156 179

160 179

180 200

(c) Insert key = 160

n=3

100

120 150 180

180

150 156 179

160 179

180 200

71
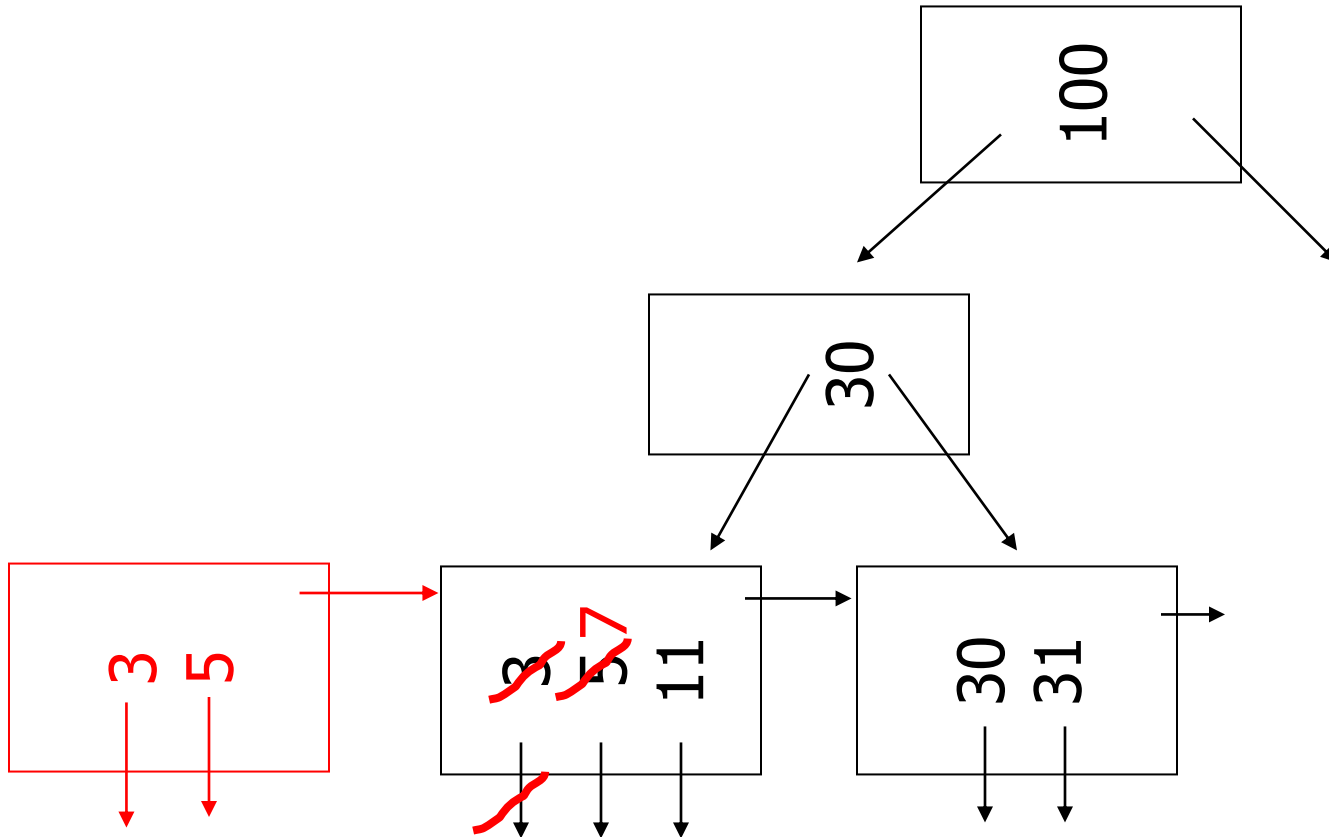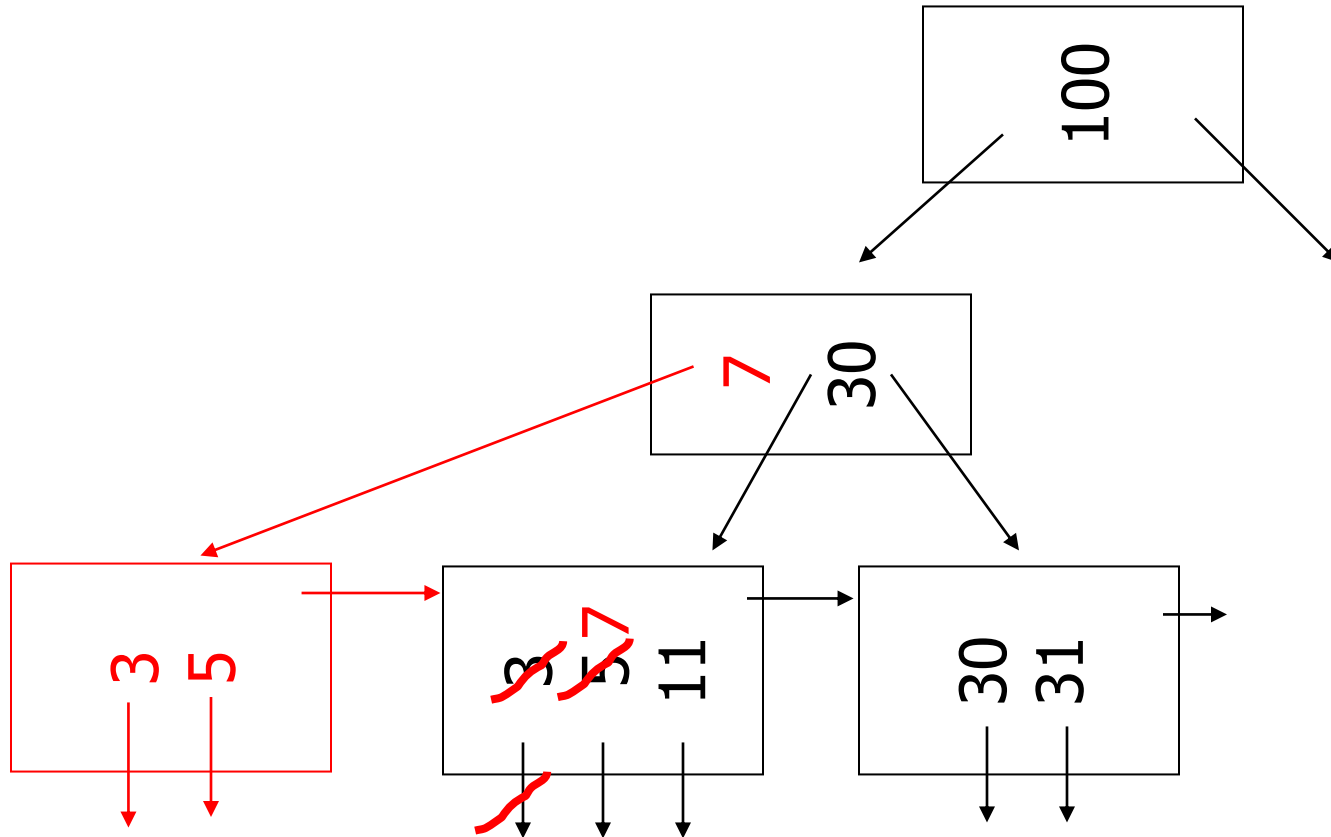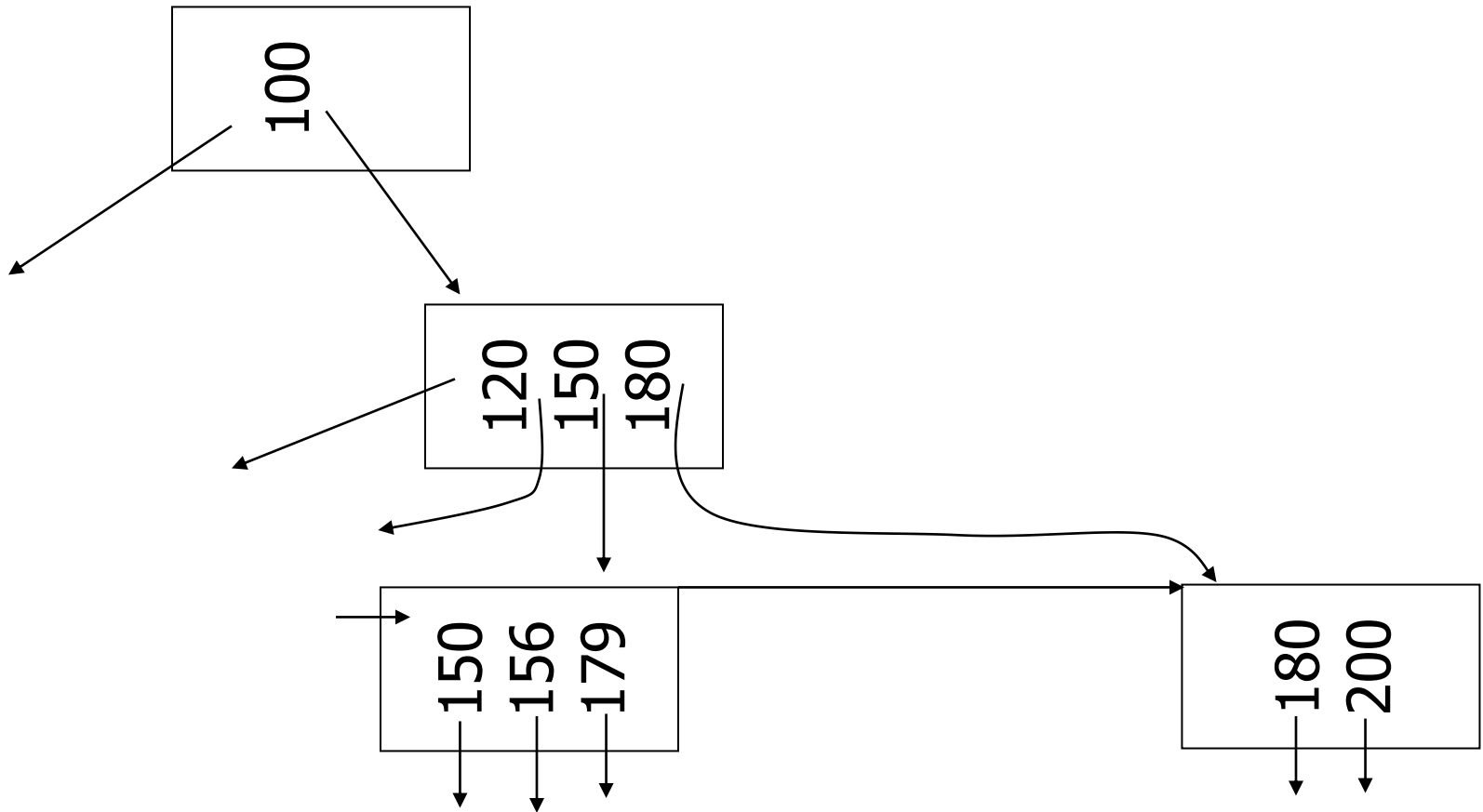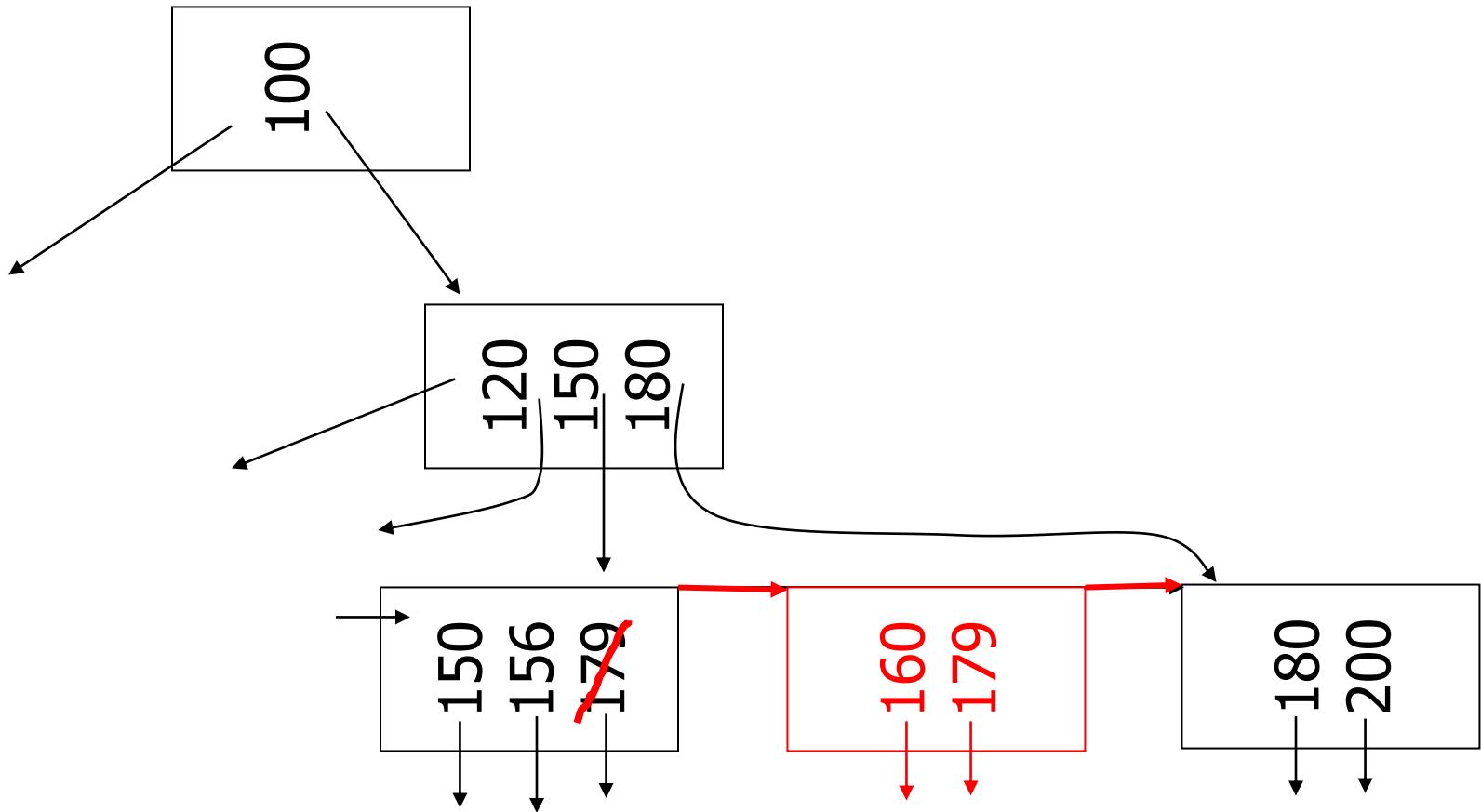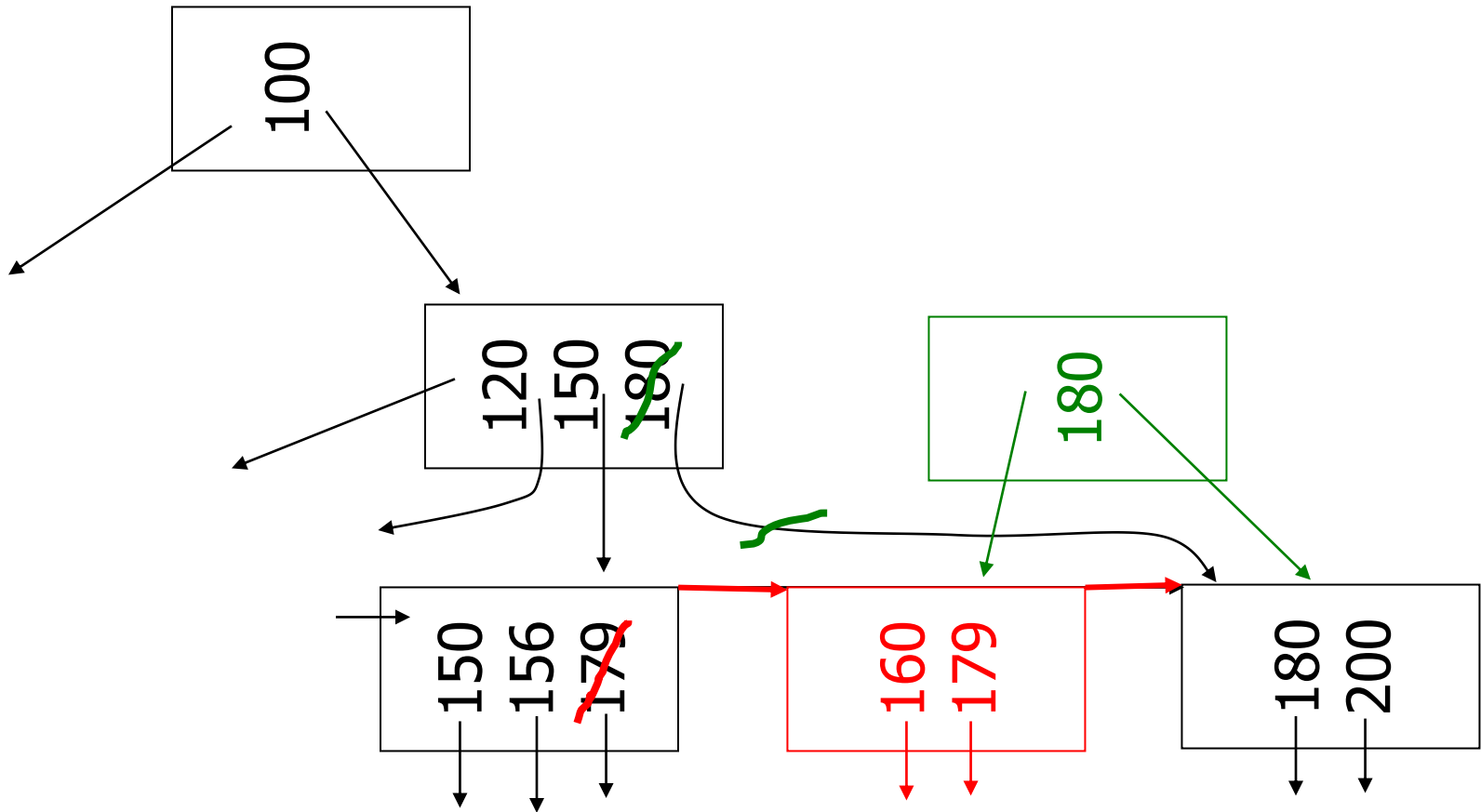
(c) Insert key = 160

n=3

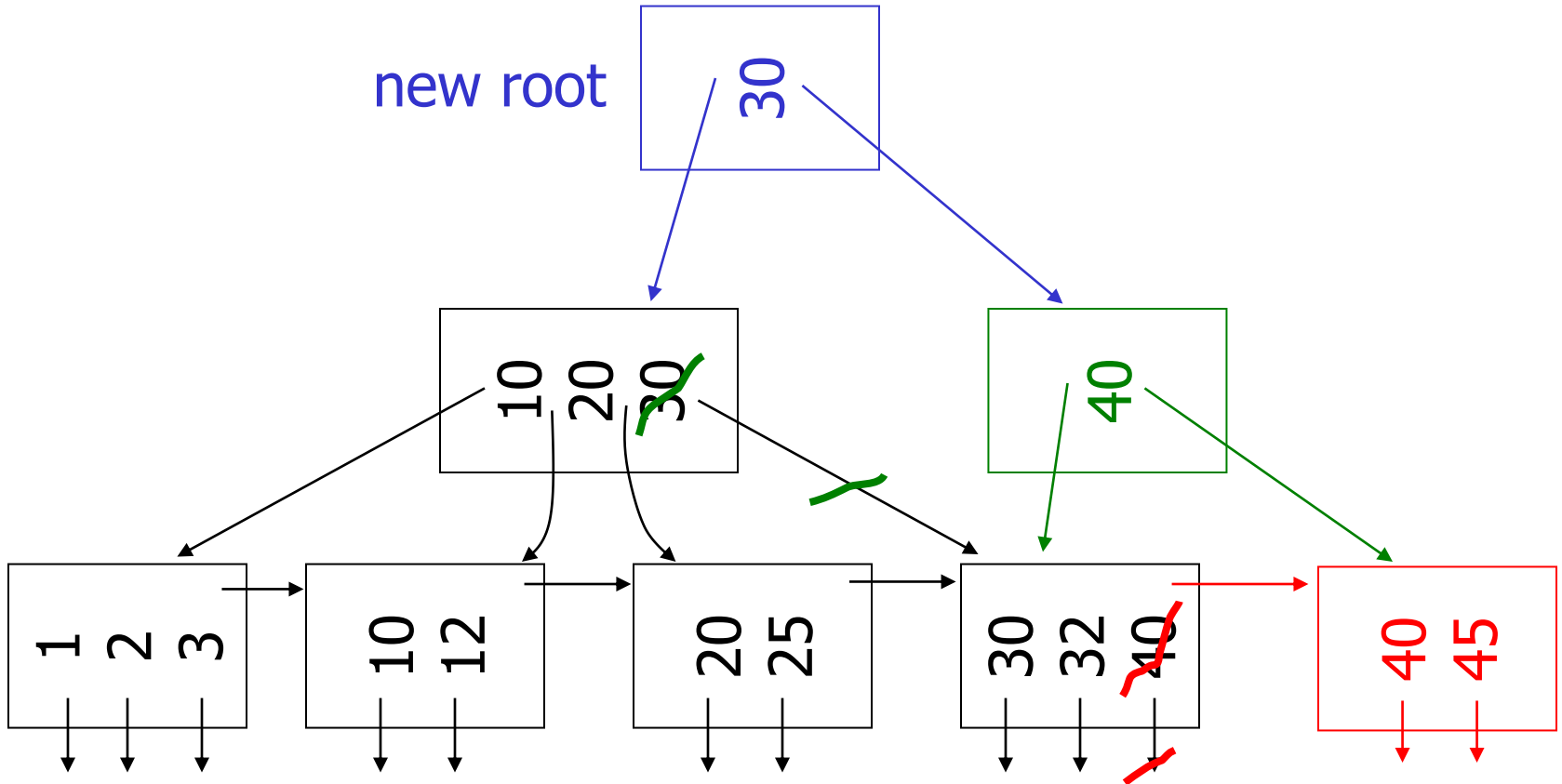# (d) New root, insert 45

n=3

(d) New root, insert 45

n=3

(d) New root, insert 45

n=3

(d) New root, insert 45

n=3

new root

30

10 20 30

40

1 2 3

10 12

20 25

30 32 40

40 45

76
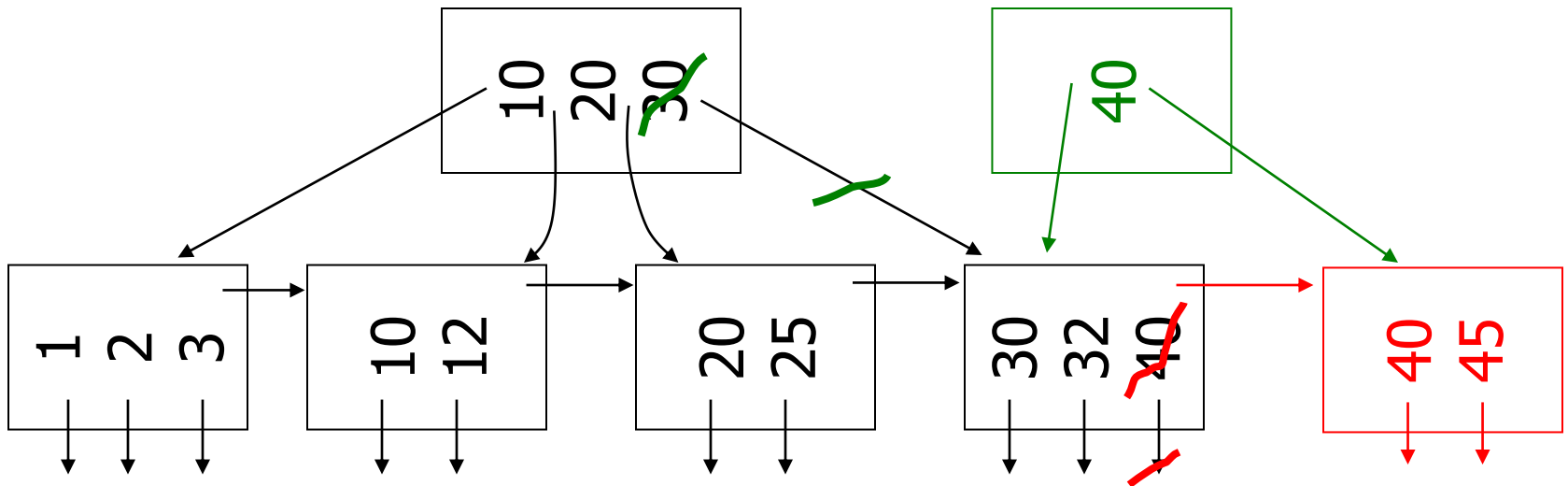
Deletion from B+tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

(c) Re-distribute keys

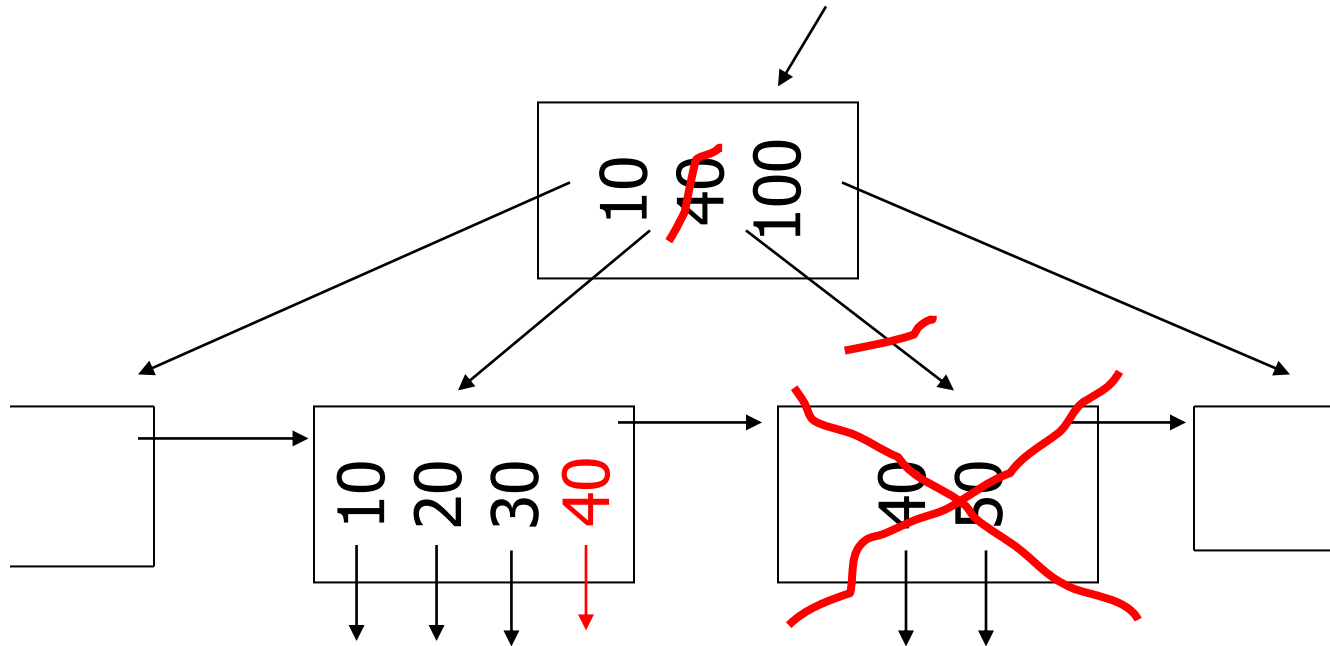(d) Cases (b) or (c) at non-leaf

(b) Coalesce with sibling
– Delete 50
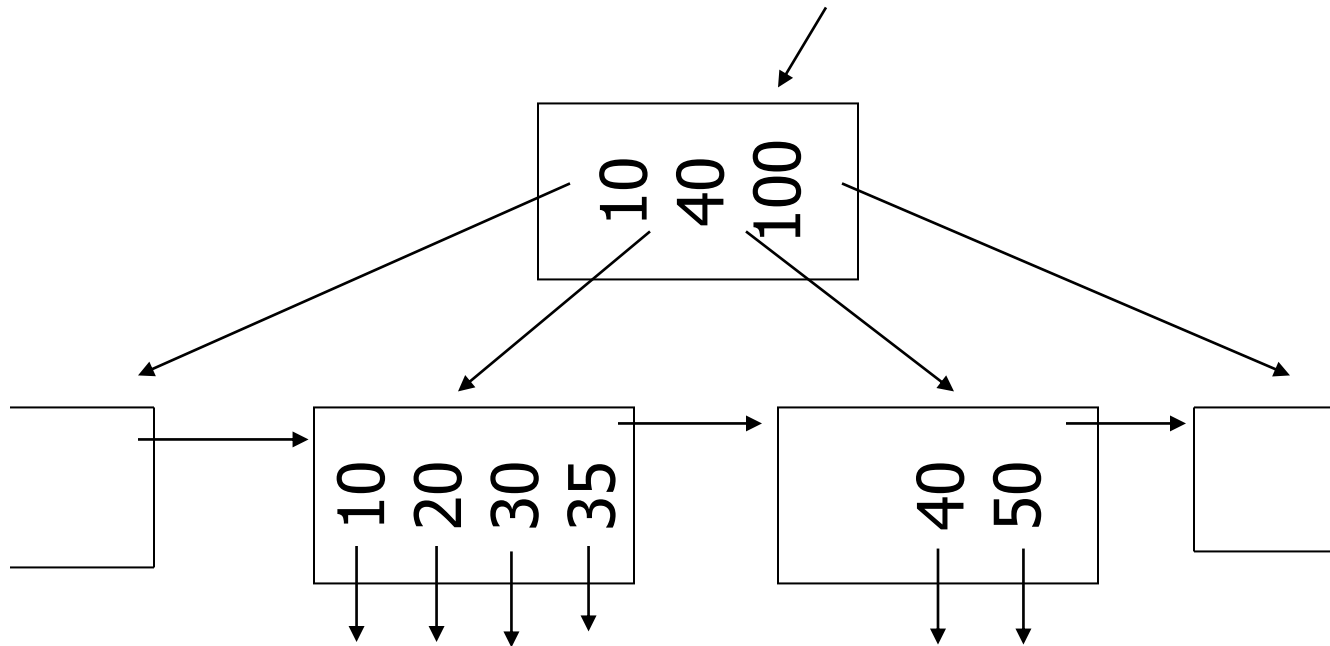
n=4
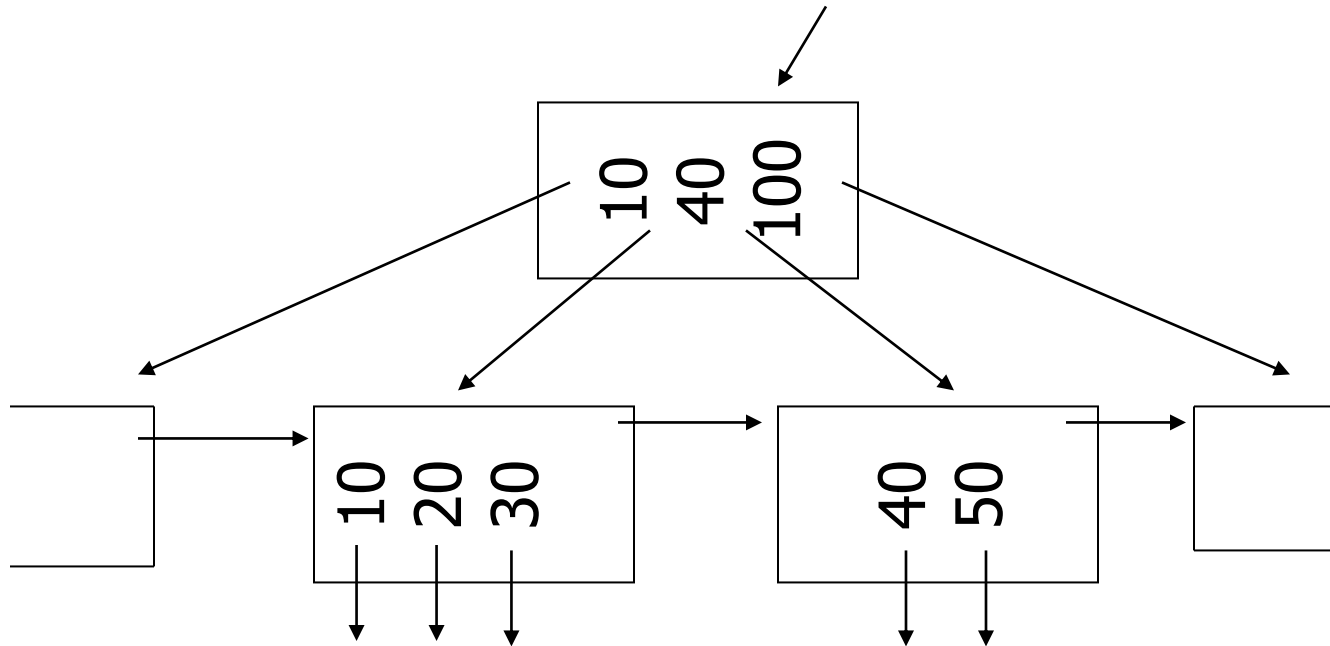
(b) Coalesce with sibling
  – Delete 50
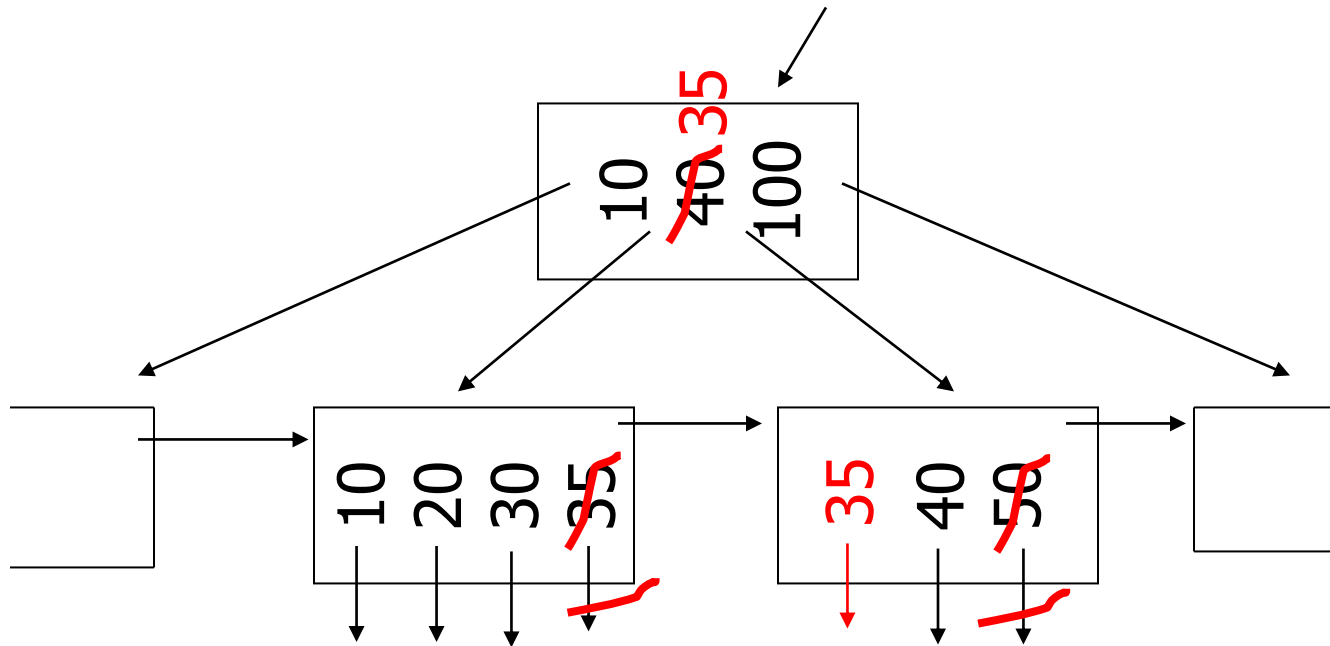
n=4

(c) Redistribute keys
  – Delete 50
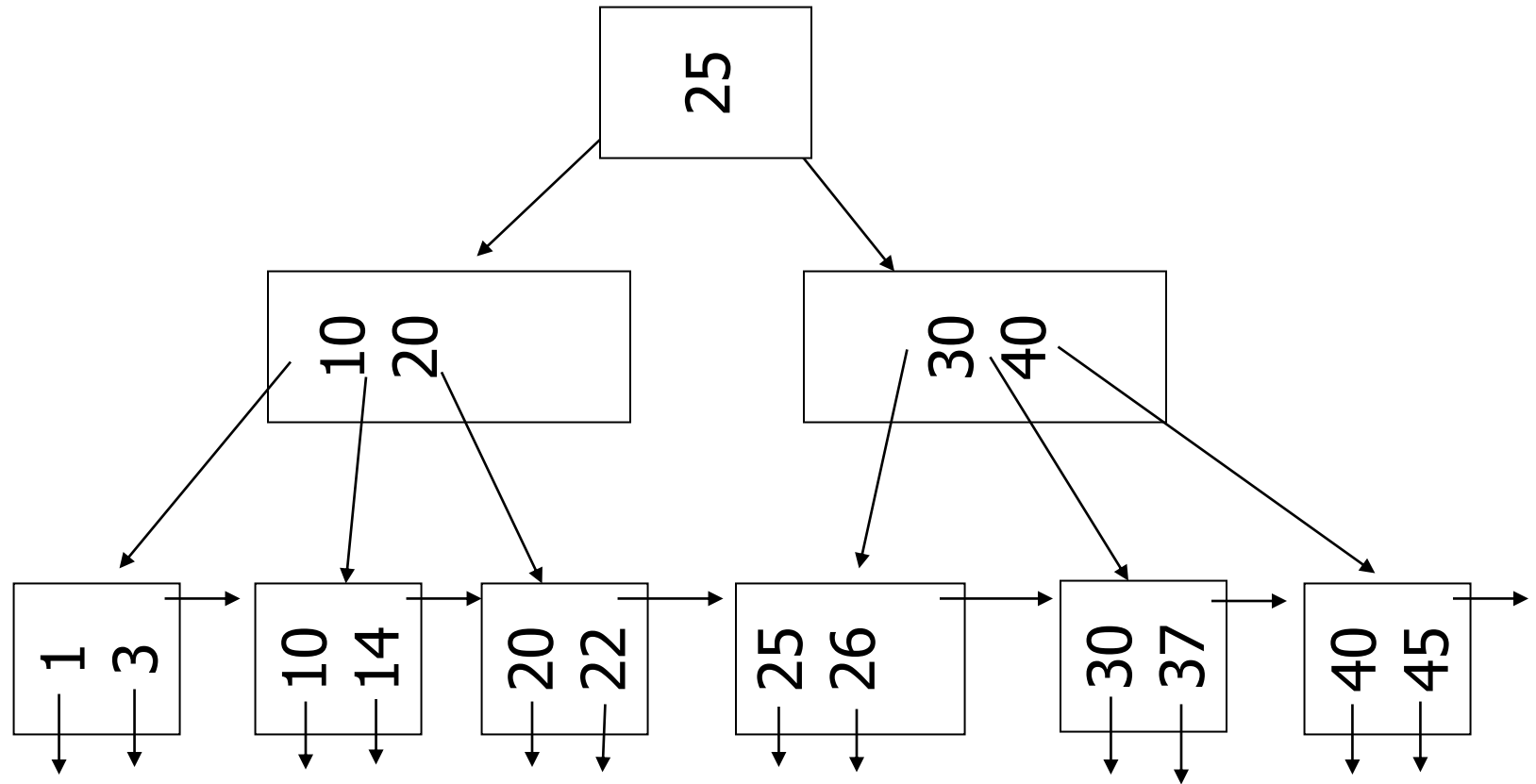
(c) Redistribute keys
– Delete 50

(d) Non-leaf coalesce
– Delete 37

$n=4$

82

(d) Non-leaf coalesce
  – Delete 37

n=4

(d) Non-leaf coalesce
– Delete 37

n=4

25

10
20

40

~~30~~ ~~40~~

1
3

10
14

20
22

25
26
30

~~30~~ ~~37~~
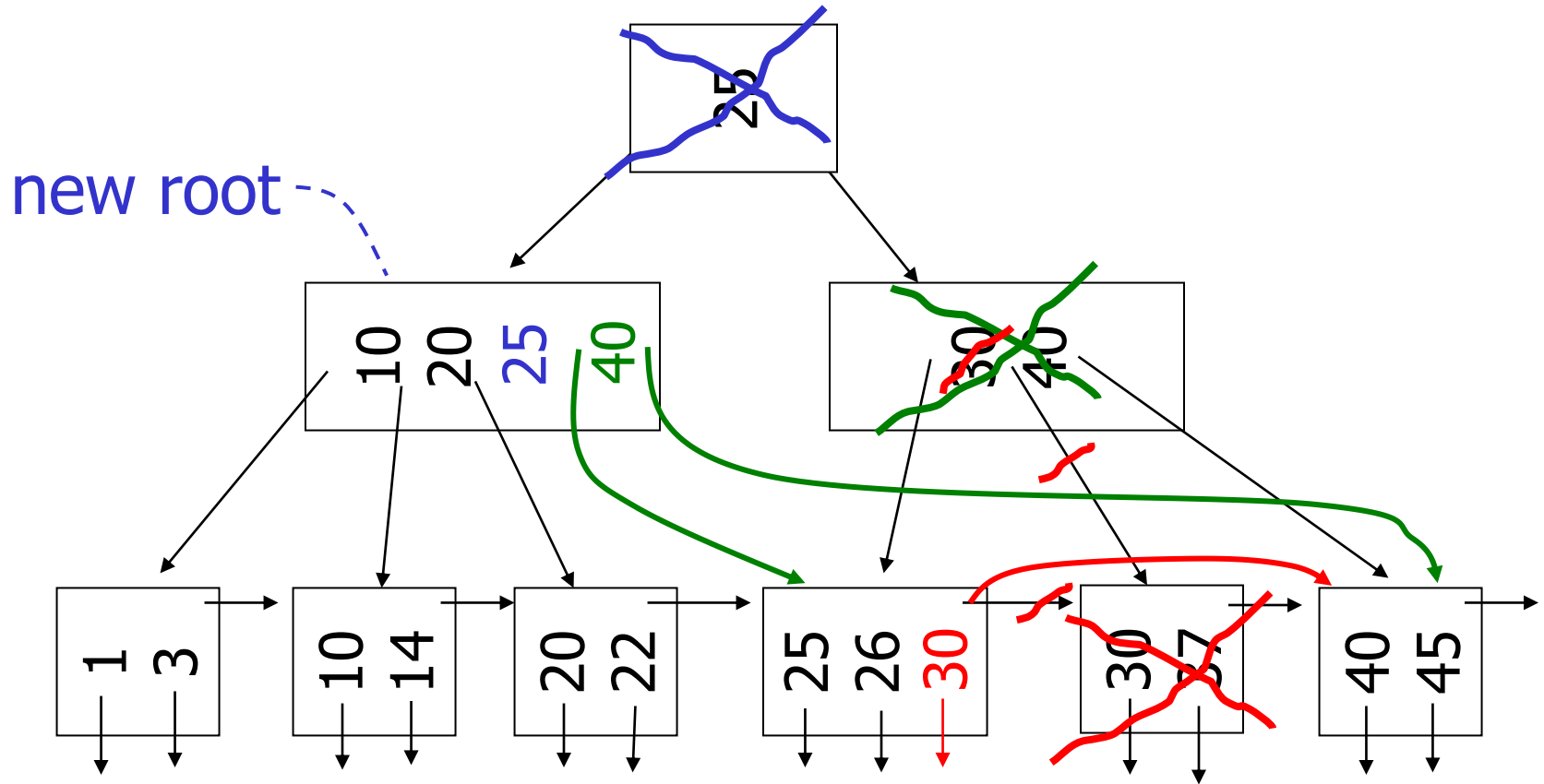
40
45

(d) Non-leaf coalesce
  – Delete 37

n=4

new root
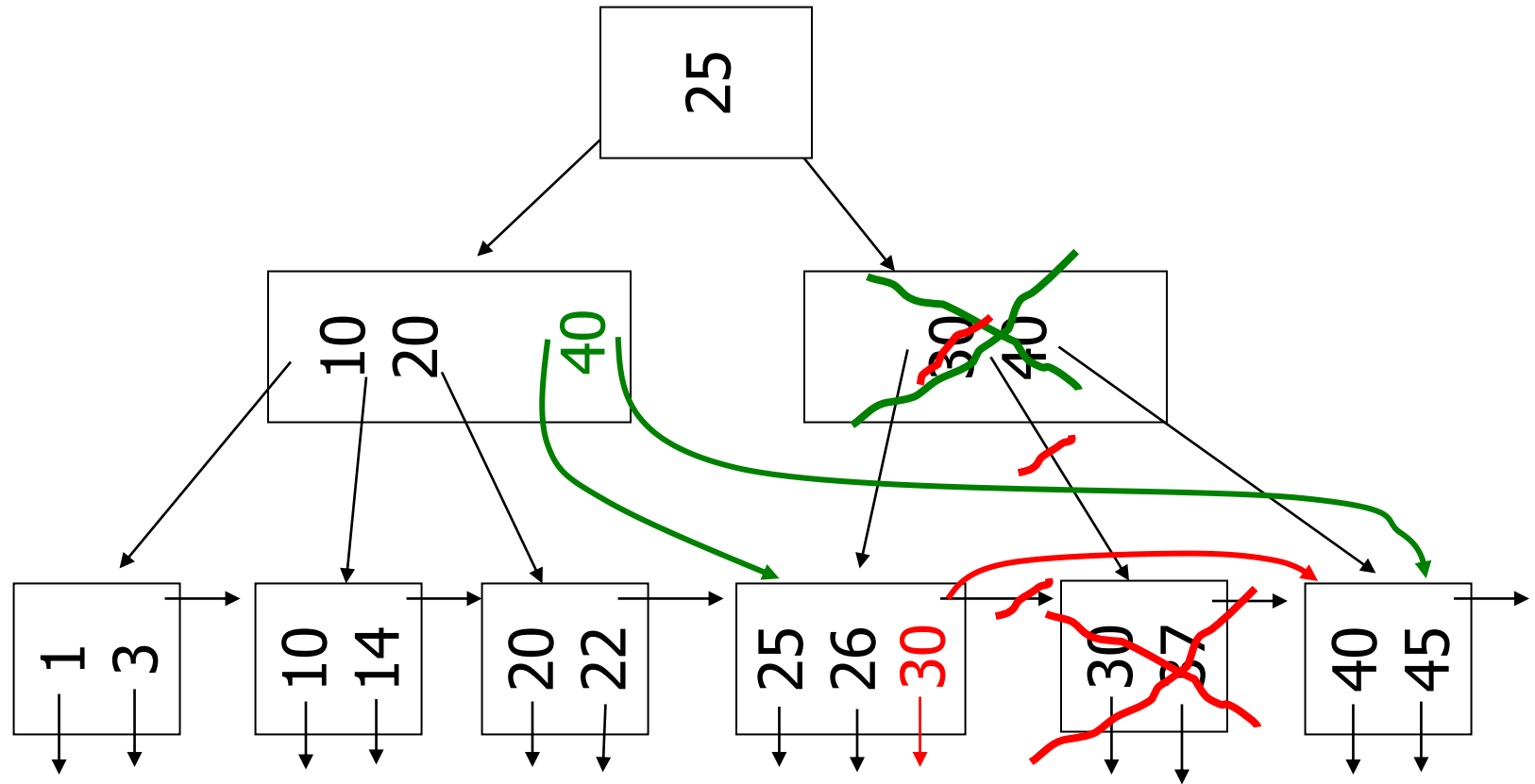
B+tree deletions in practice

– Often, coalescing is <u>not</u> implemented
  – Too hard and not worth it!

Outline/summary

- ## Conventional Indexes
  - Sparse vs. dense
  - Primary vs. secondary

- ## B trees

- ## Hashing schemes (recommended reading, not mandatory)

The slides in this lecture are taken from:

- Hector Garcia-Molina, CS 245: Database System Principles, Notes 4: Indexing.

Reading

- Héctor García-Molina, Jeffrey Ullman, and Jennifer Widom. Database Systems: The Complete Book