CSEN 604: Databases II

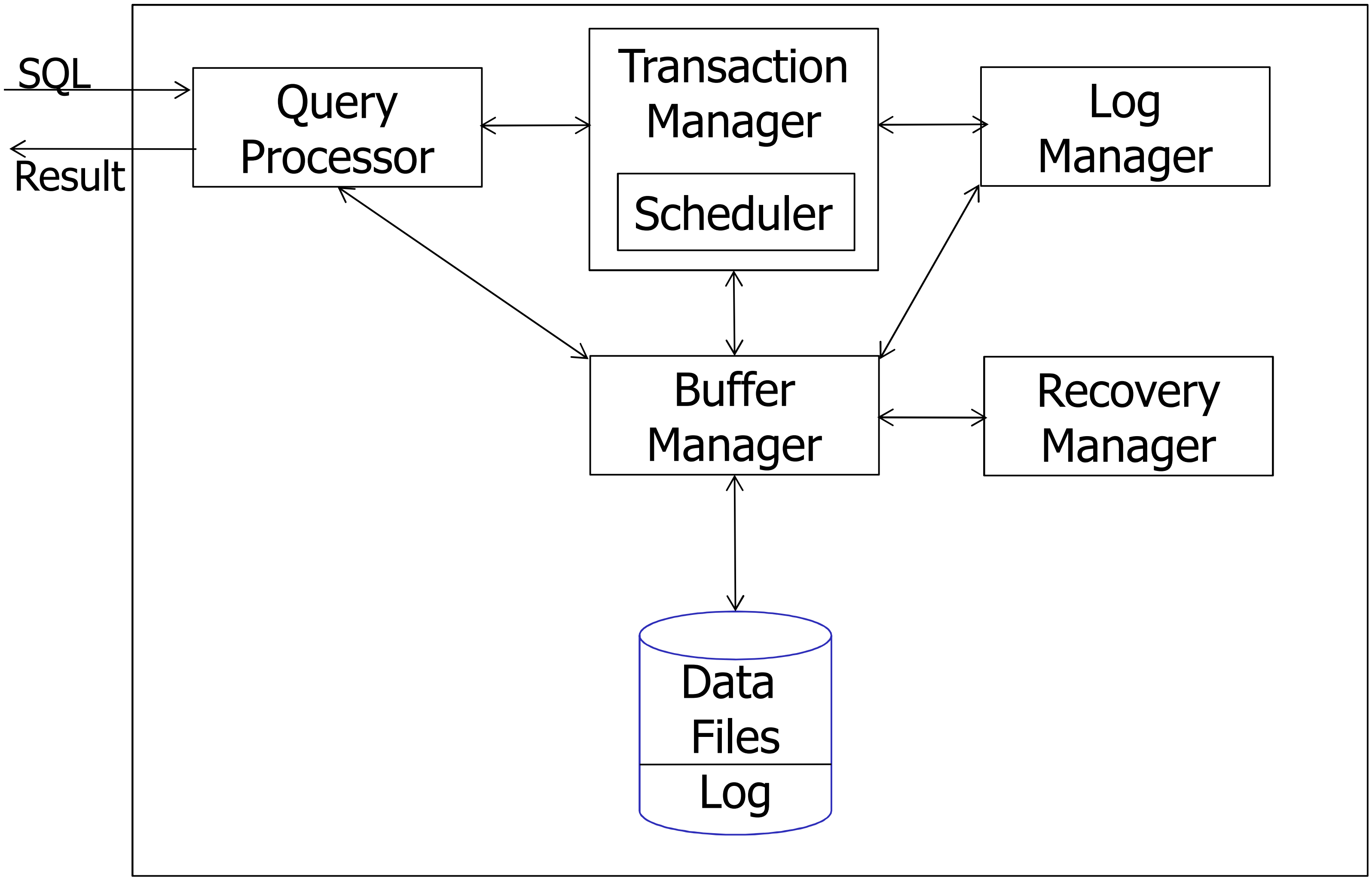
Lecture 2

Dr. Wael Abouelsaadat
wael.abouelsaadat@guc.edu.eg
Office: C7.208

Office Hour is 4th slot Saturday or you can email for appointment

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the book: *Database Systems; the Complete Book*

DBMS Architecture

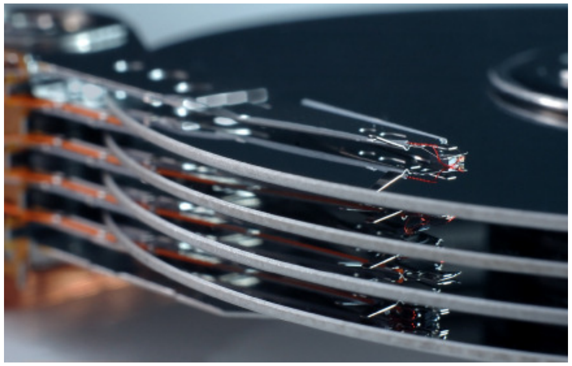


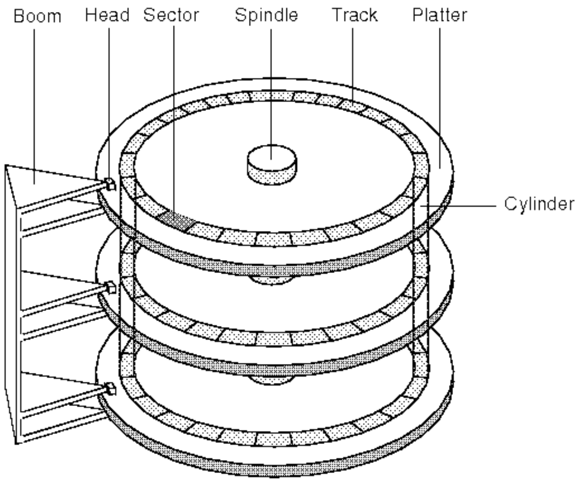
Topics

- Conventional Indexes
- B-trees

Page as a Storage Unit

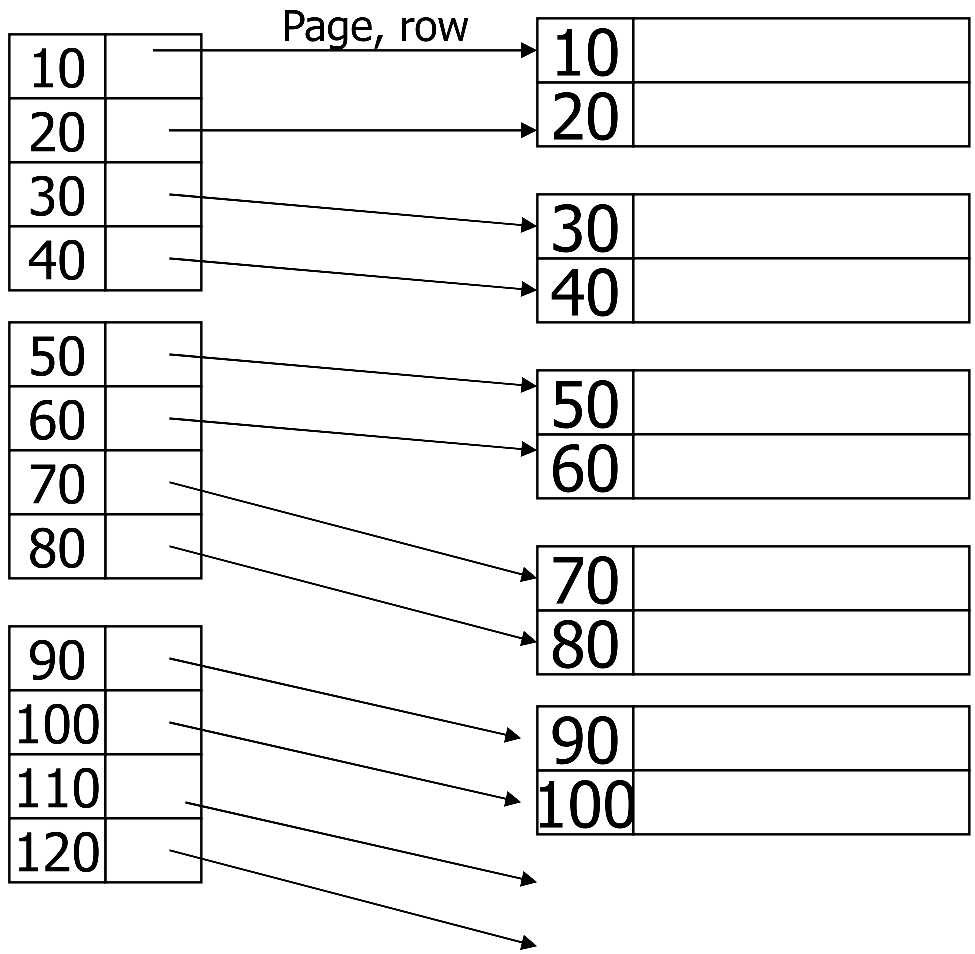
- Historically a page ranged from 16KB to 64KB
- What you can read without moving the head on the HD.

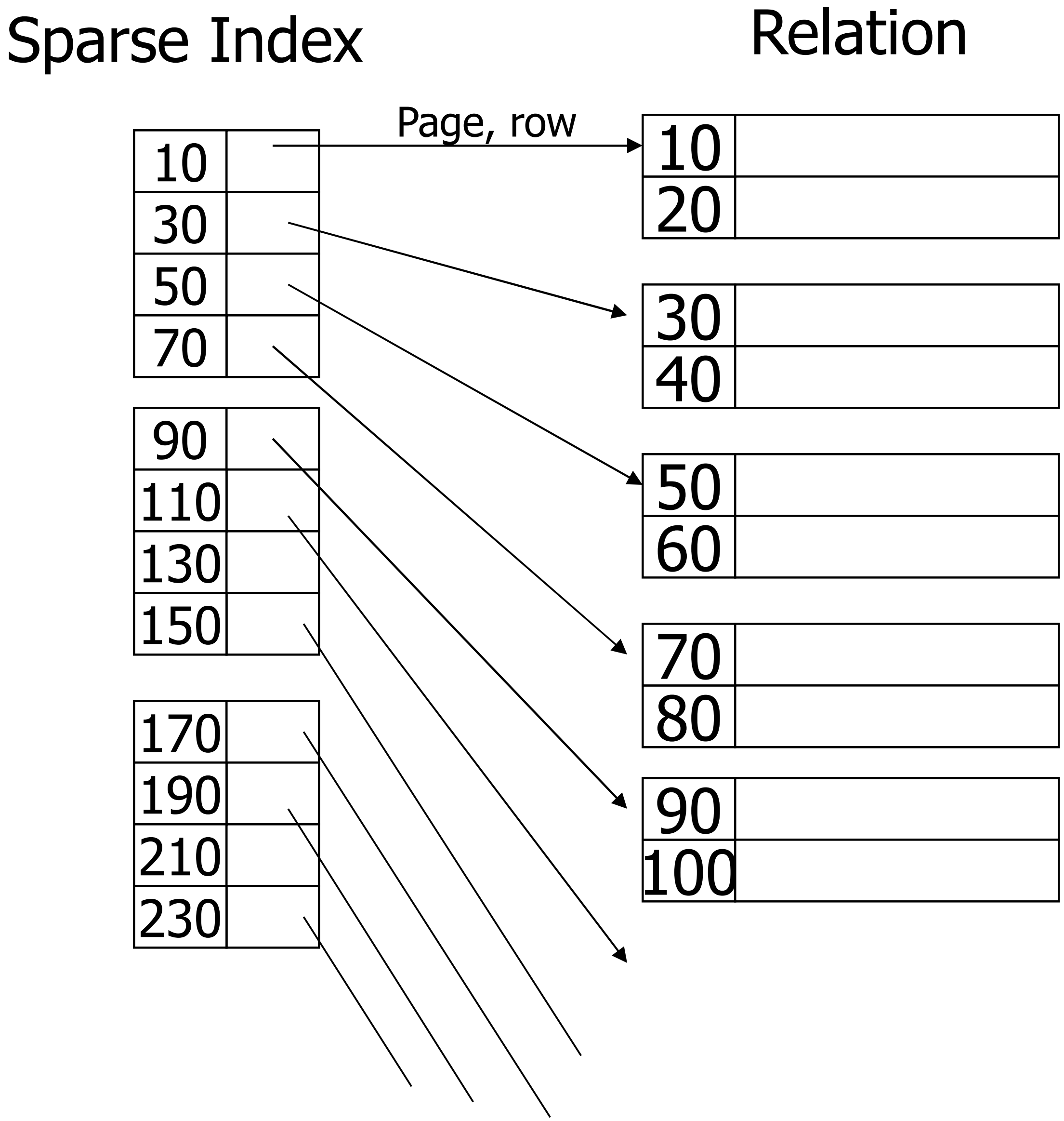


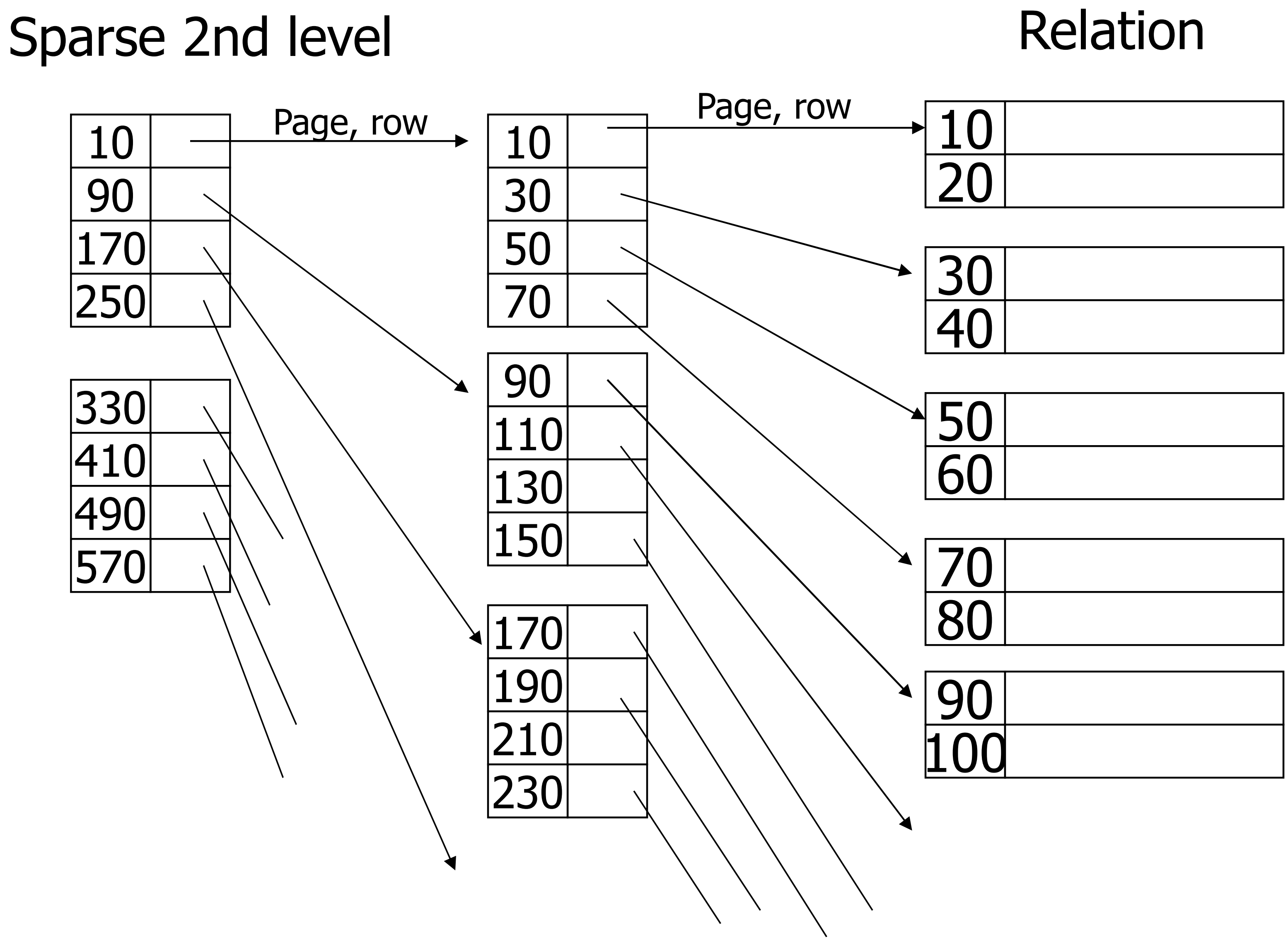


Dense Index

Relation




Sparse Index
Relation
Page, row
10
30
50
70
90
110
130
150
170
190
210
230
10
20
30
40
50
60
70
80
90
100

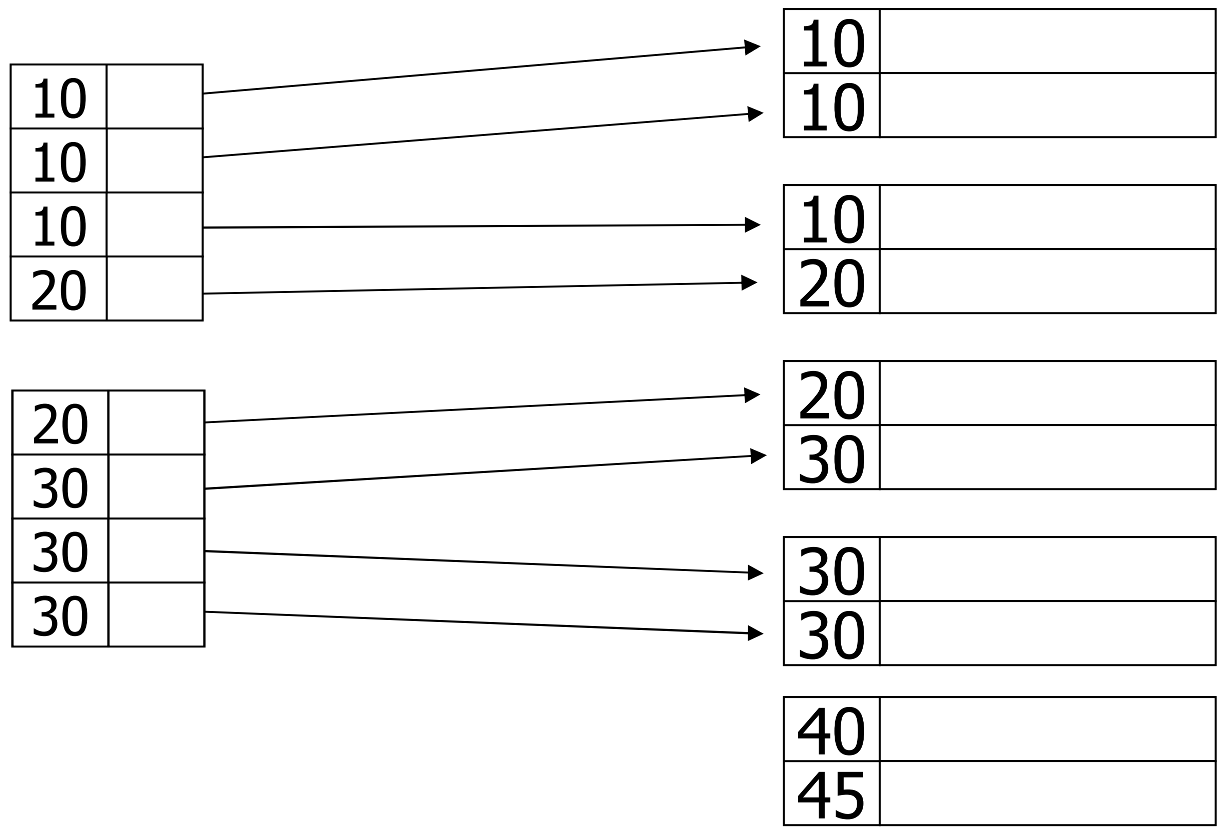

Sparse 2nd level
Relation
Page, row
Page, row
10
90
170
250
330
410
490
570
10
30
50
70
90
110
130
150
170
190
210
230
10
20
30
40
50
60
70
80
90
100

Sparse vs. Dense Tradeoff

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

Duplicate keys

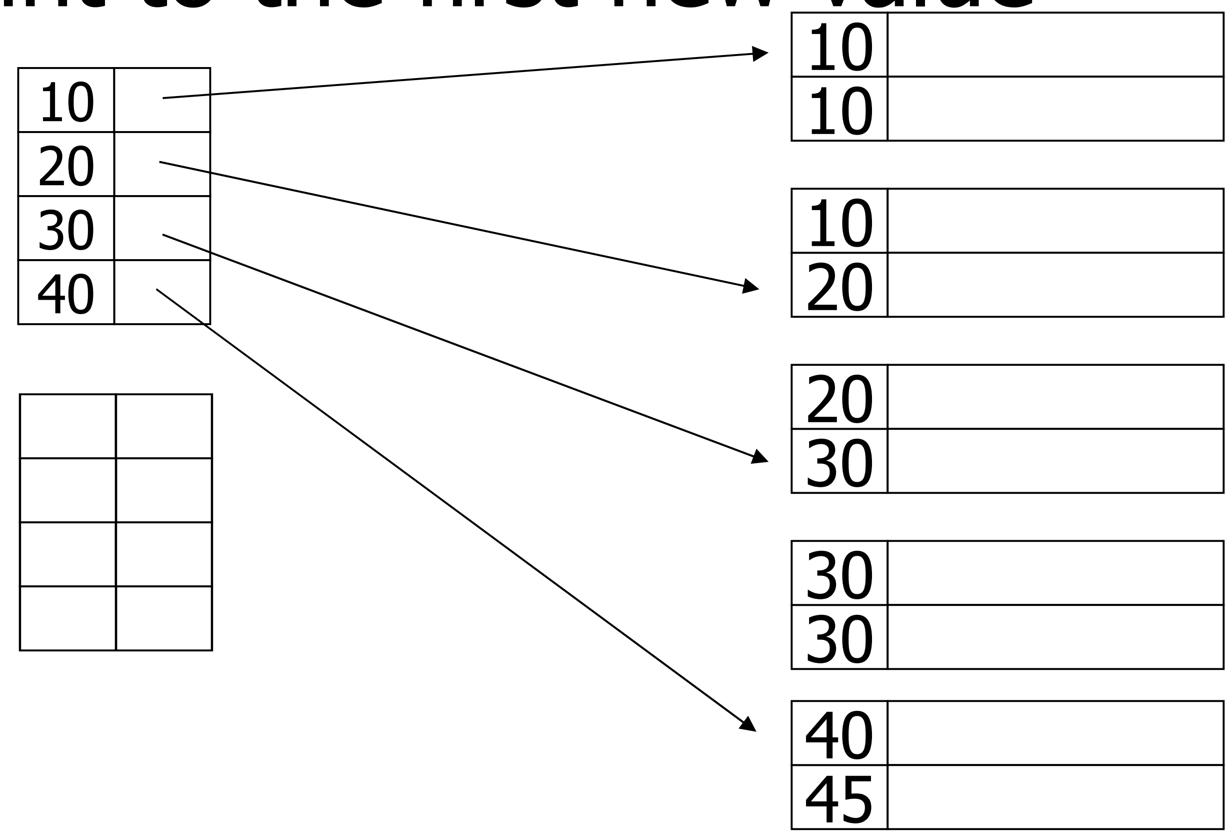
Dense index, one way to implement?



Duplicate keys

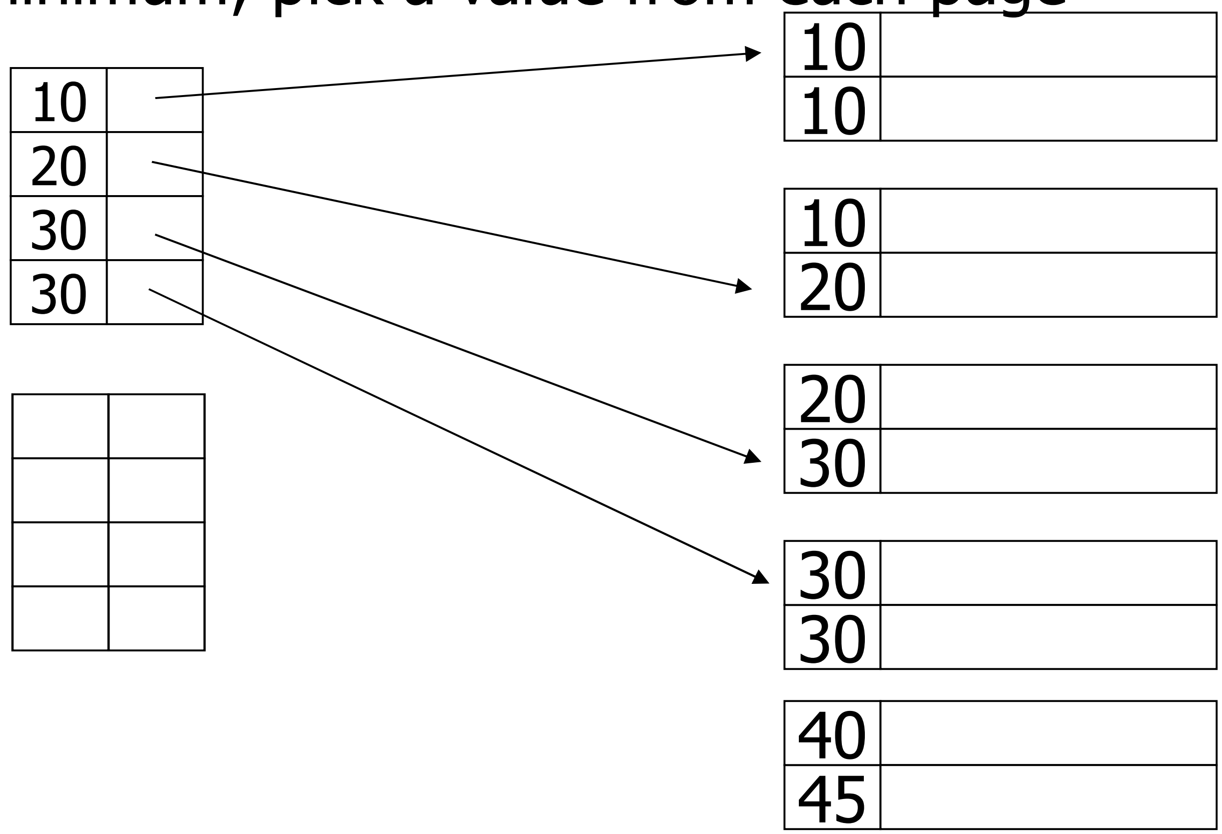
Dense index, better way?

Point to the first new value



Duplicate keys
Dense index

At minimum; pick a value from each page

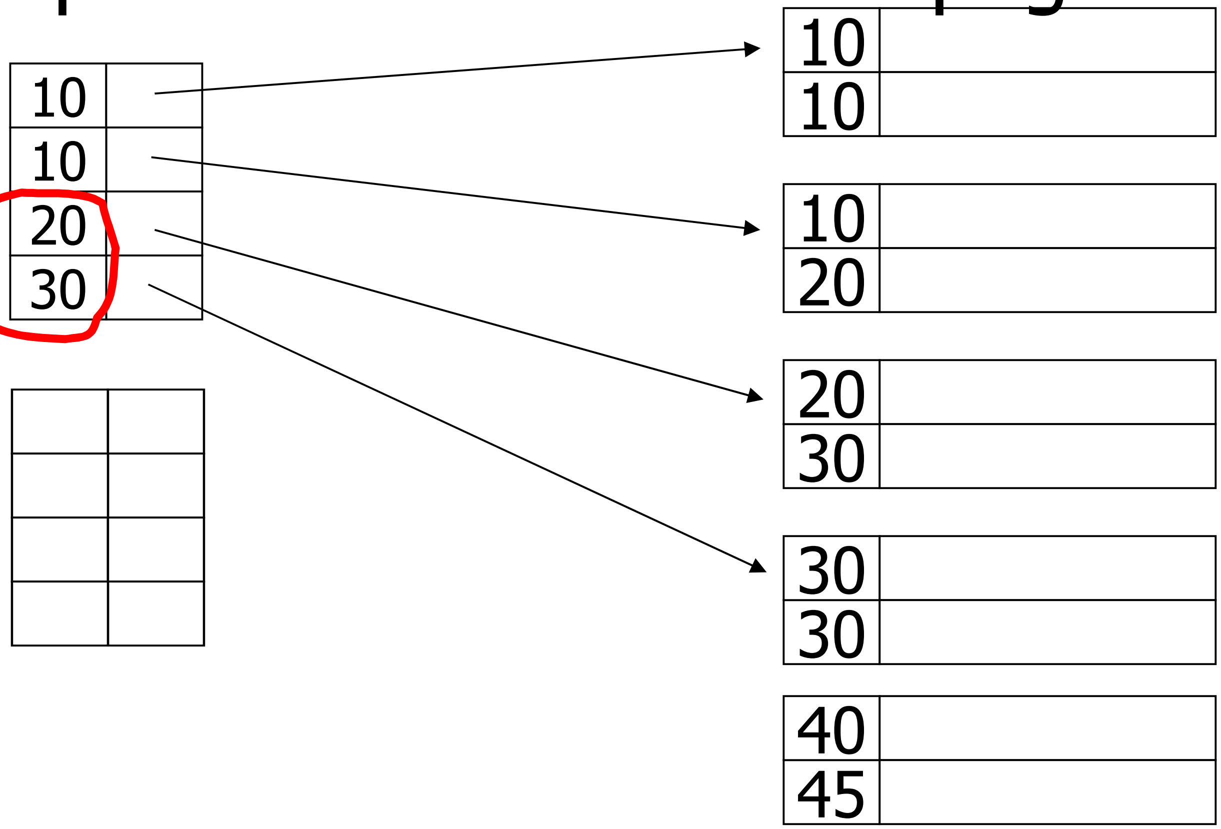


Duplicate keys

Sparse index.

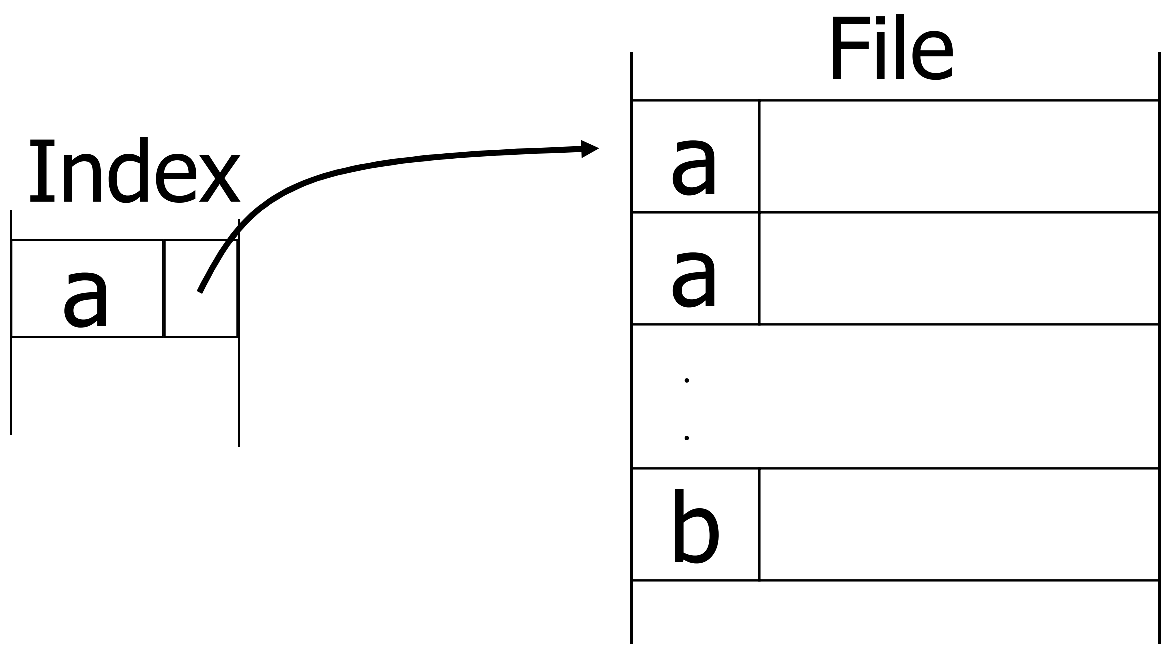
Also pick a value from each page

careful if looking for 20 or 30!

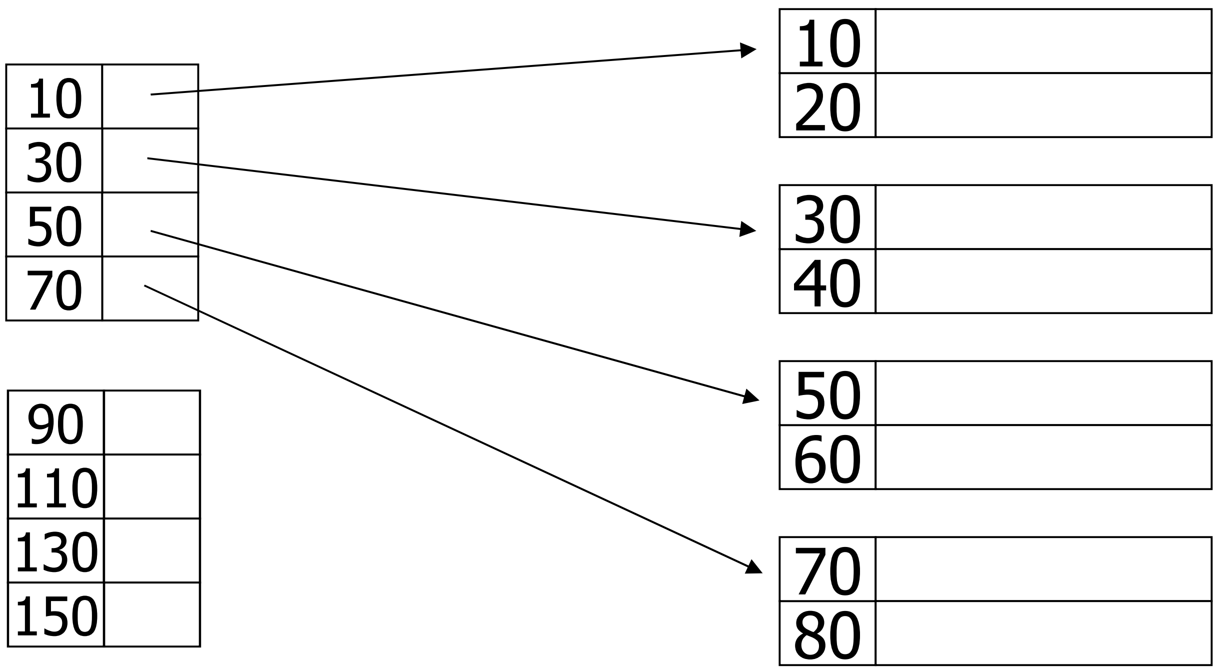


Summary: Duplicates, primary index

- Index may point to first instance of each value only
- Each page must be accessed from index

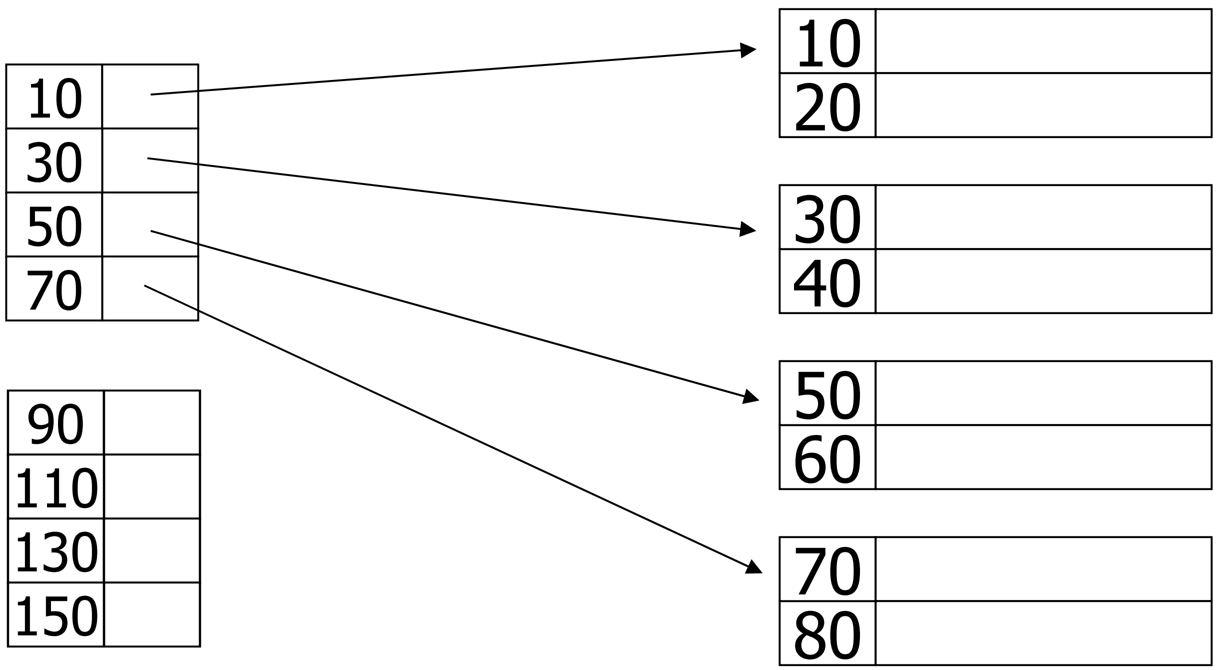


Deletion from sparse index



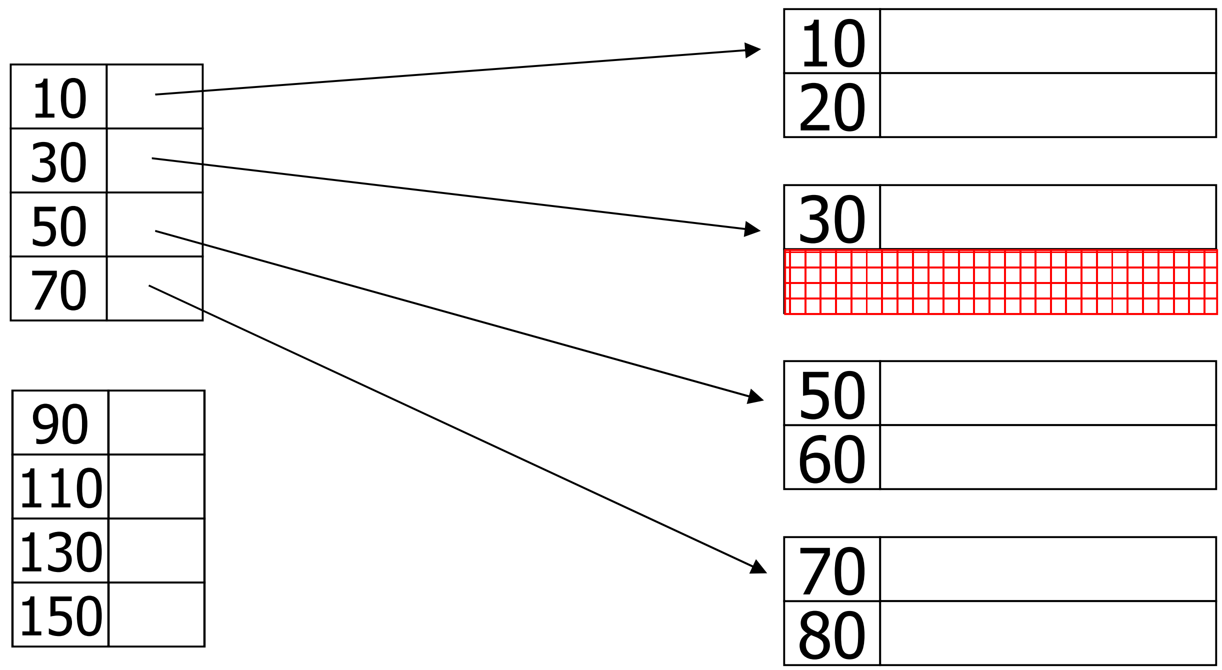
Deletion from sparse index

– delete record 40



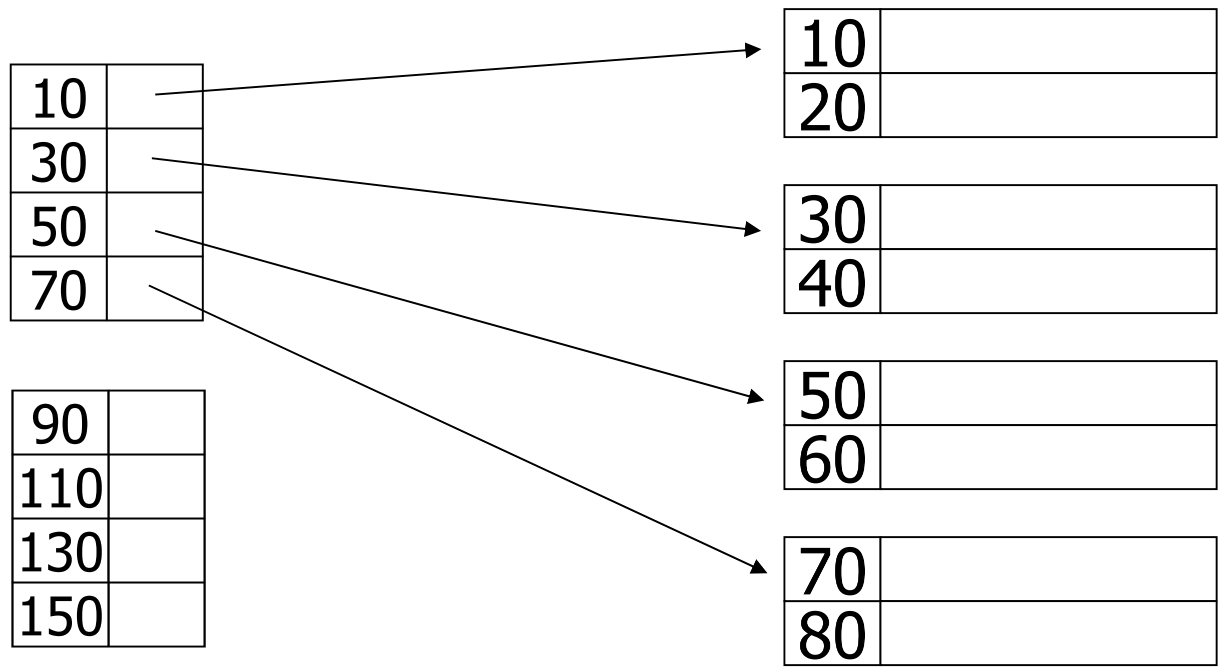
Deletion from sparse index

– delete record 40



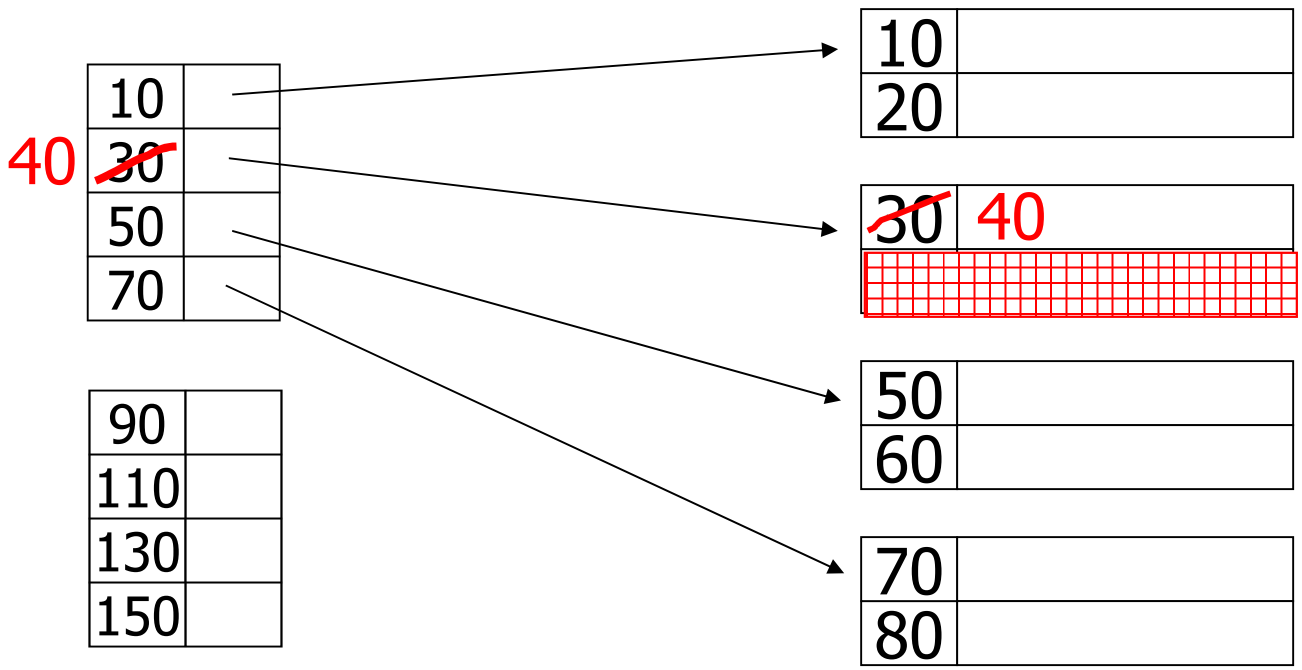
Deletion from sparse index

– delete record 30



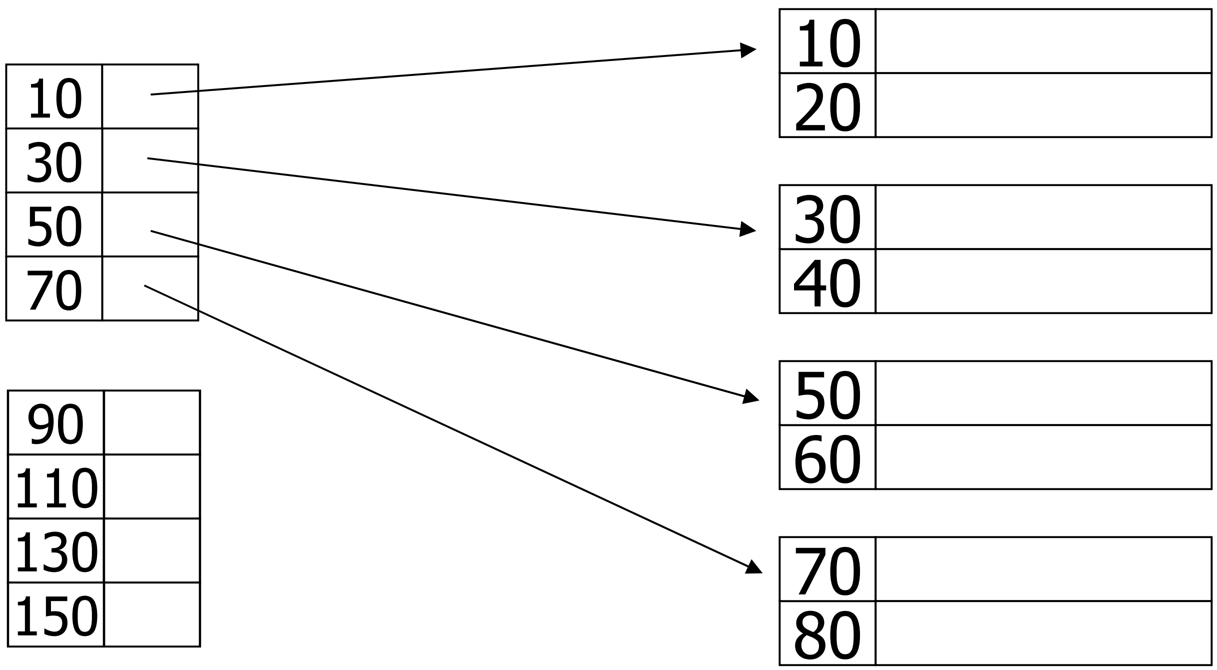
Deletion from sparse index

– delete record 30



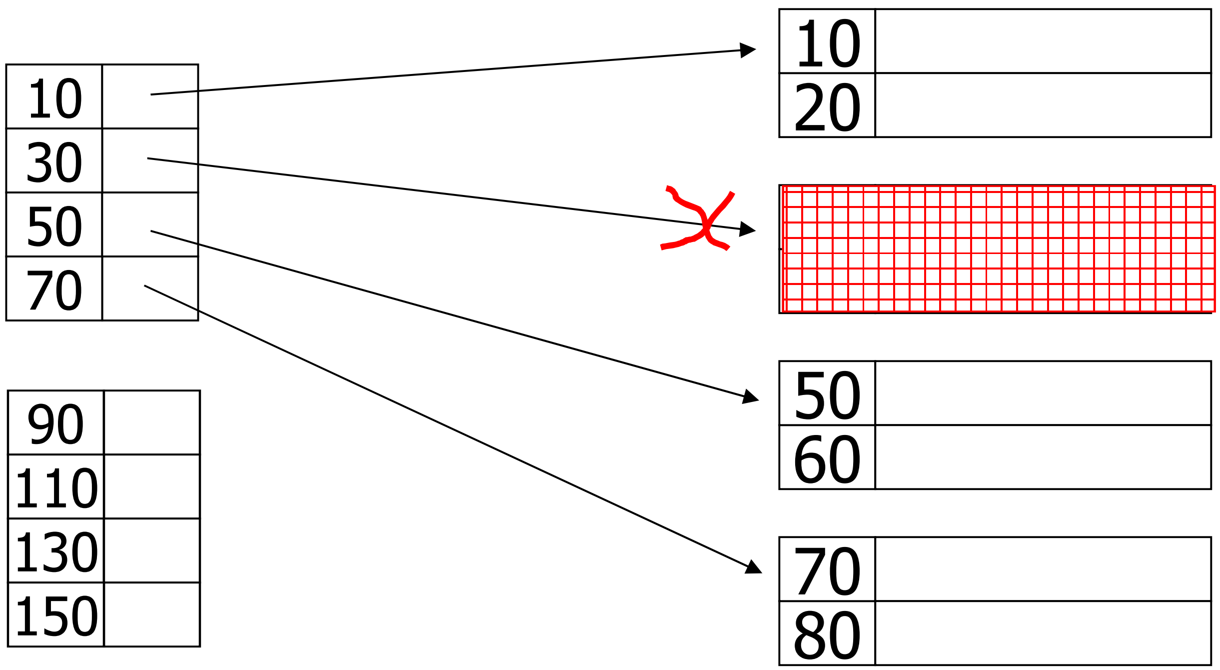
Deletion from sparse index

– delete records 30 & 40



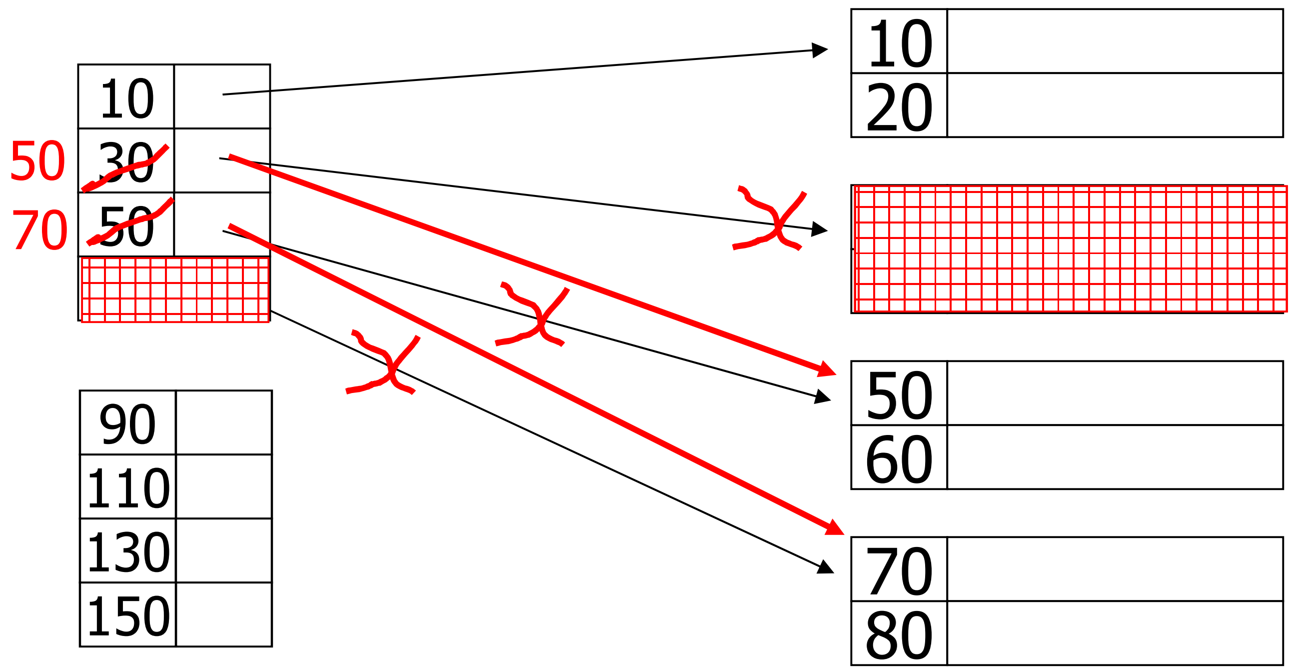
Deplete from sparse index

- delete records 30 & 40

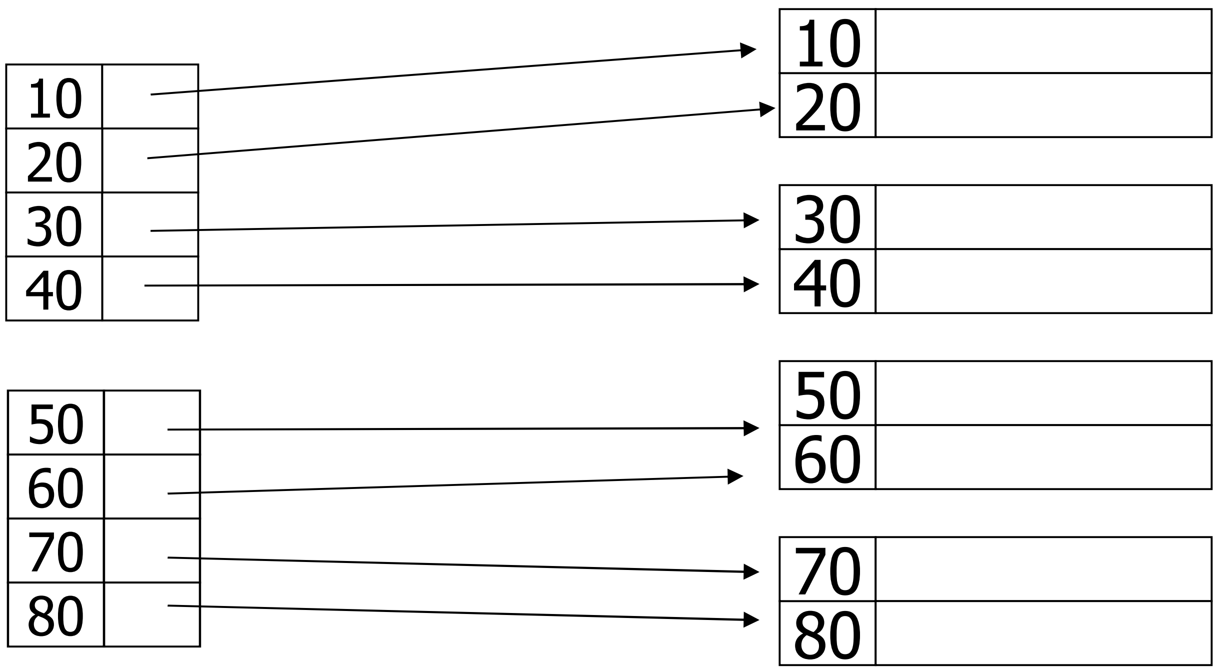


Deliberation from sparse index

– delete records 30 & 40

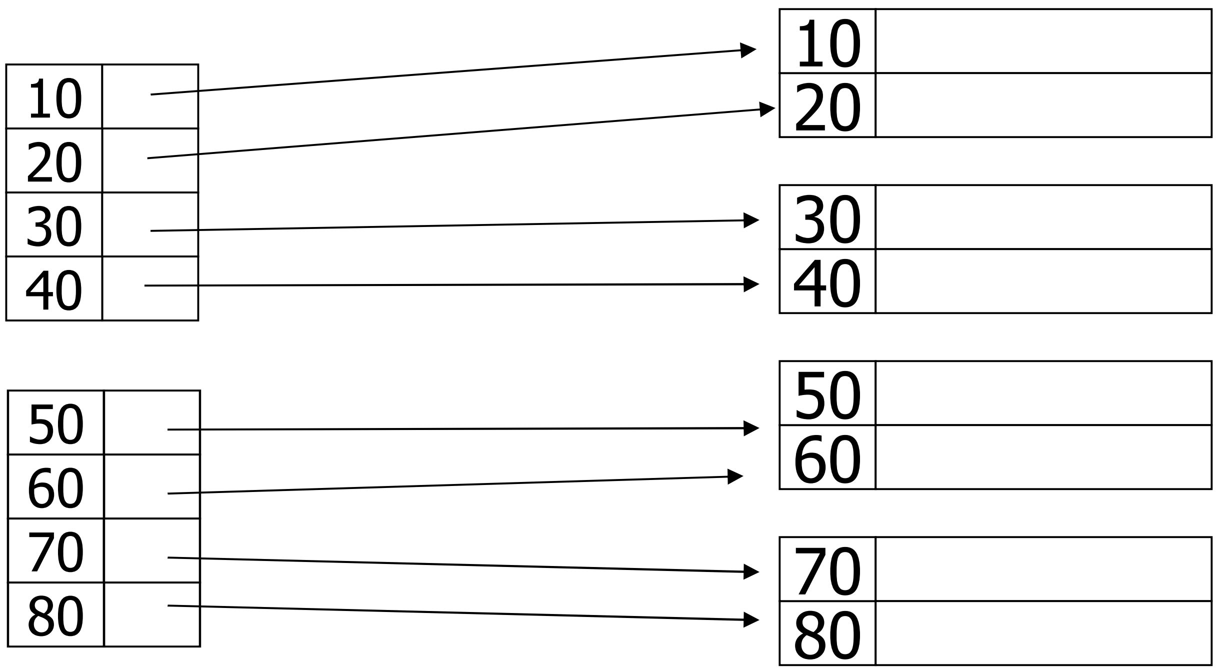


Deletion from dense index



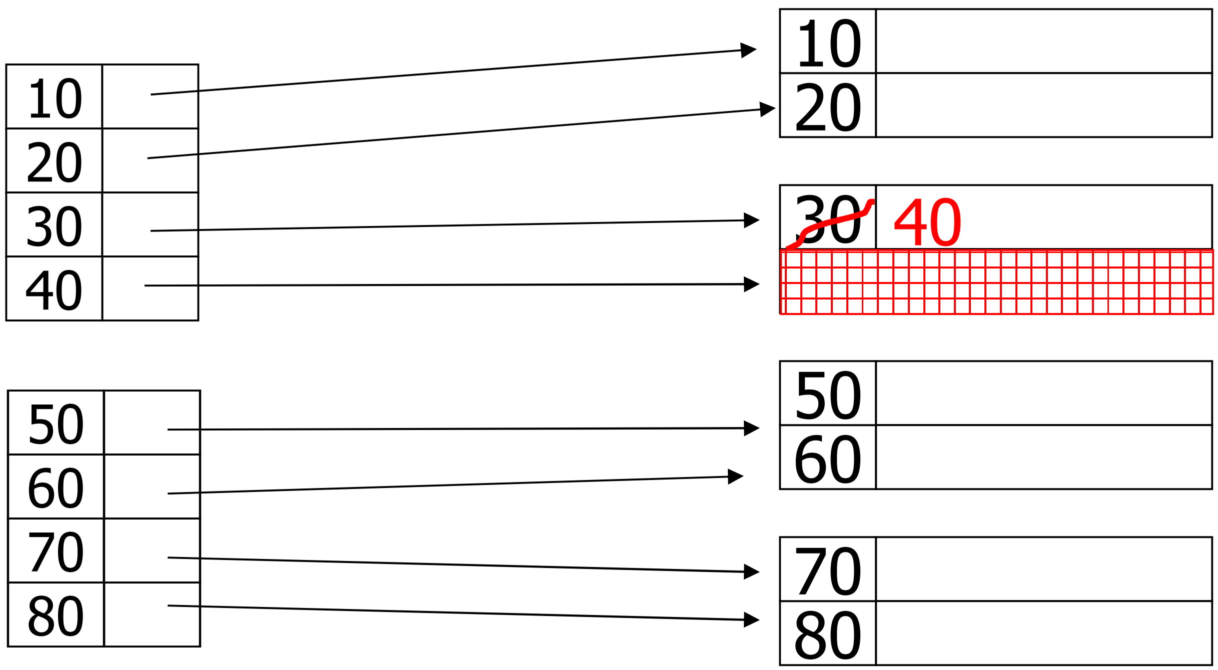
Deletion from dense index

– delete record 30



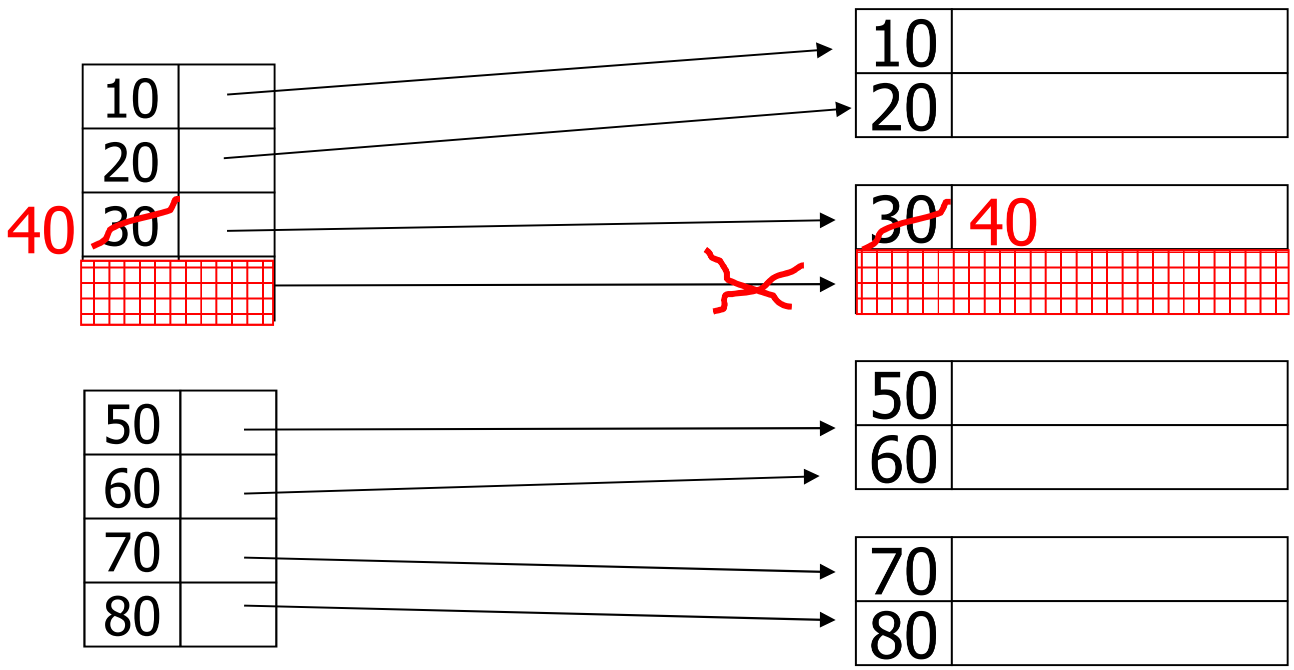
Deletion from dense index

– delete record 30

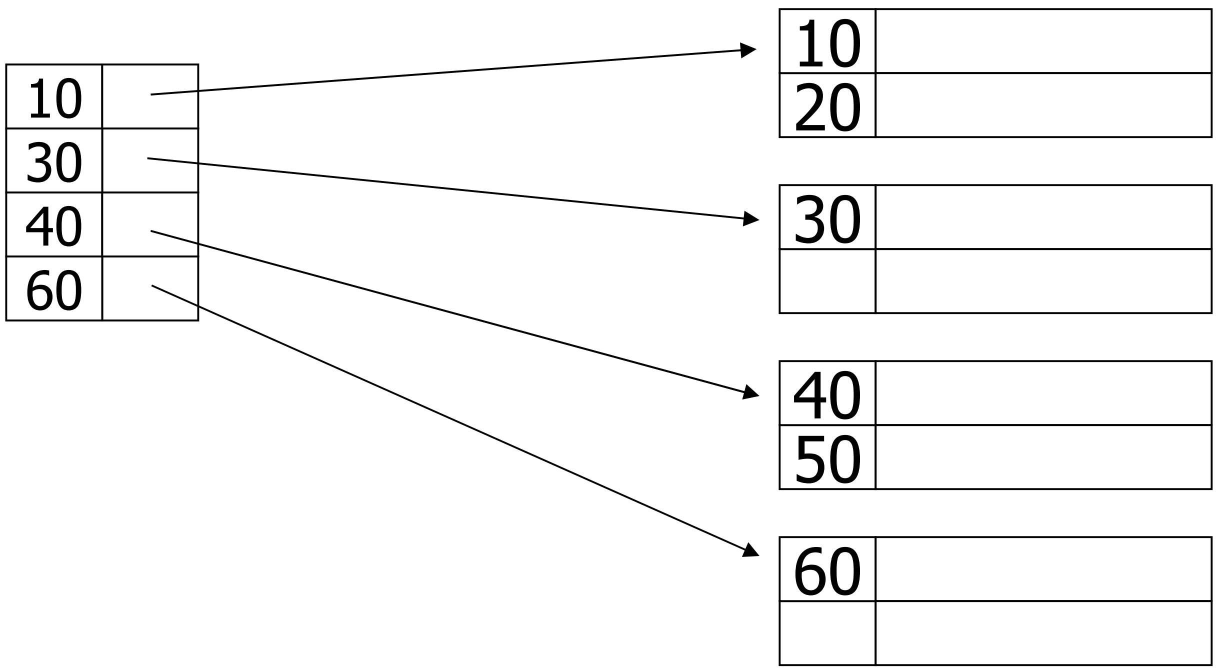


Deletion from dense index

– delete record 30

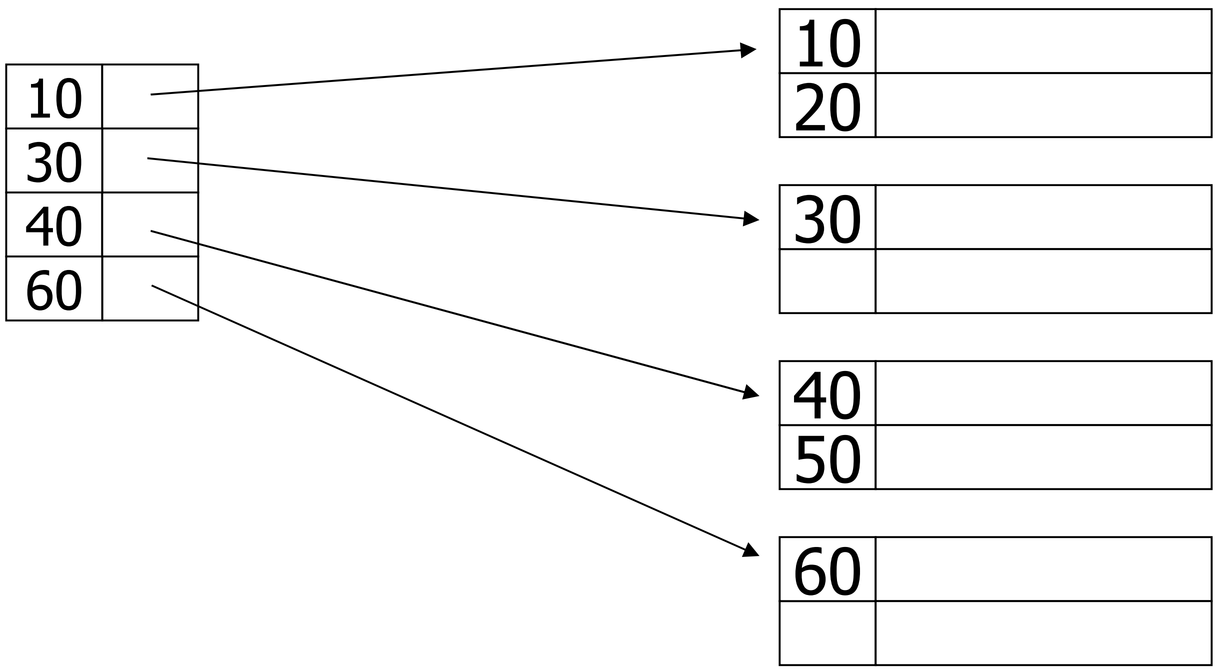


Insertion, sparse index case



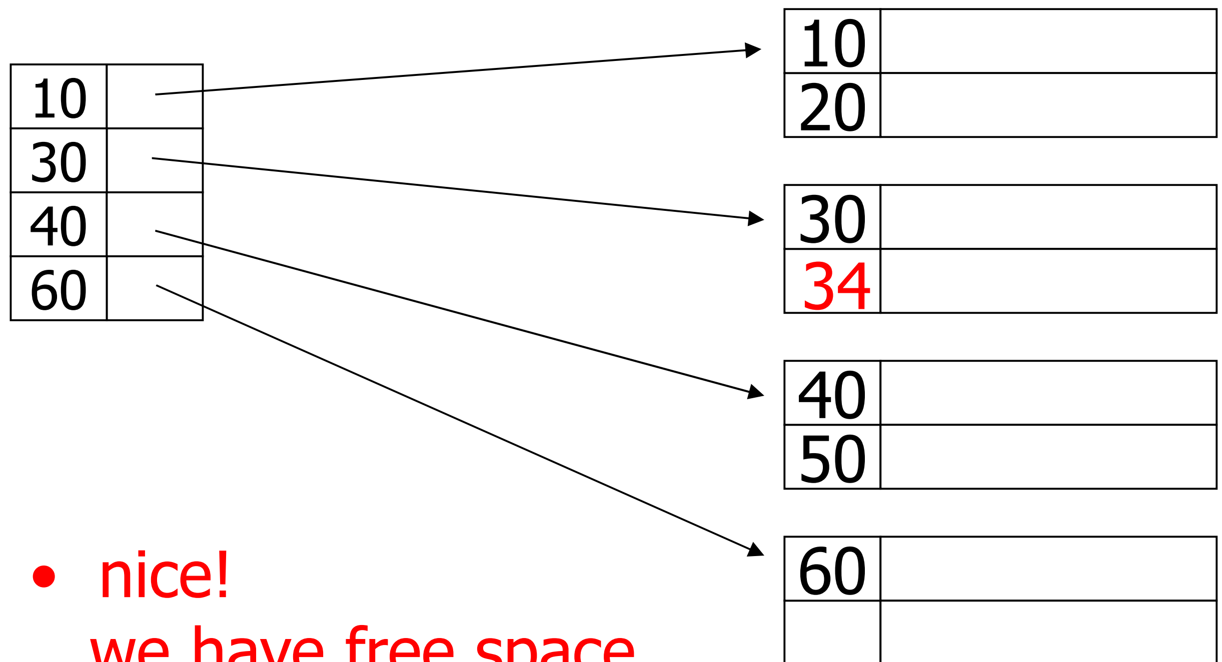
Insertion, sparse index case

– insert record 34



Insertion, sparse index case

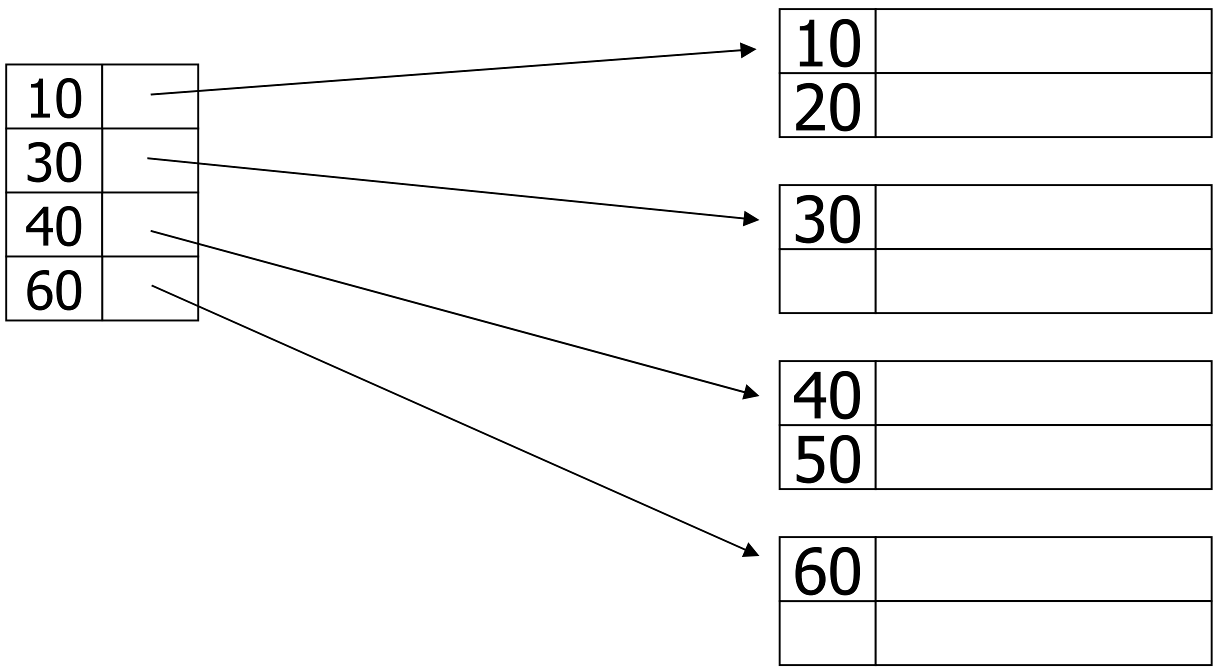
– insert record 34



- nice!
  we have free space
  where we need it!

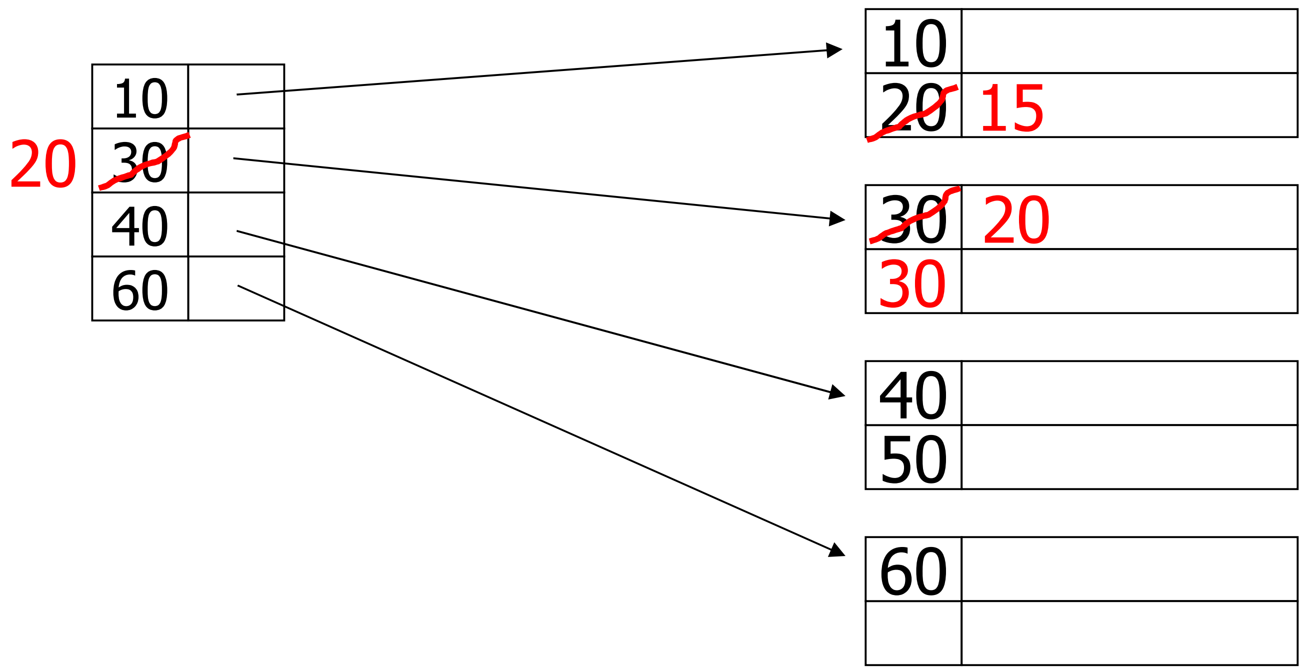
Insertion, sparse index case

– insert record 15



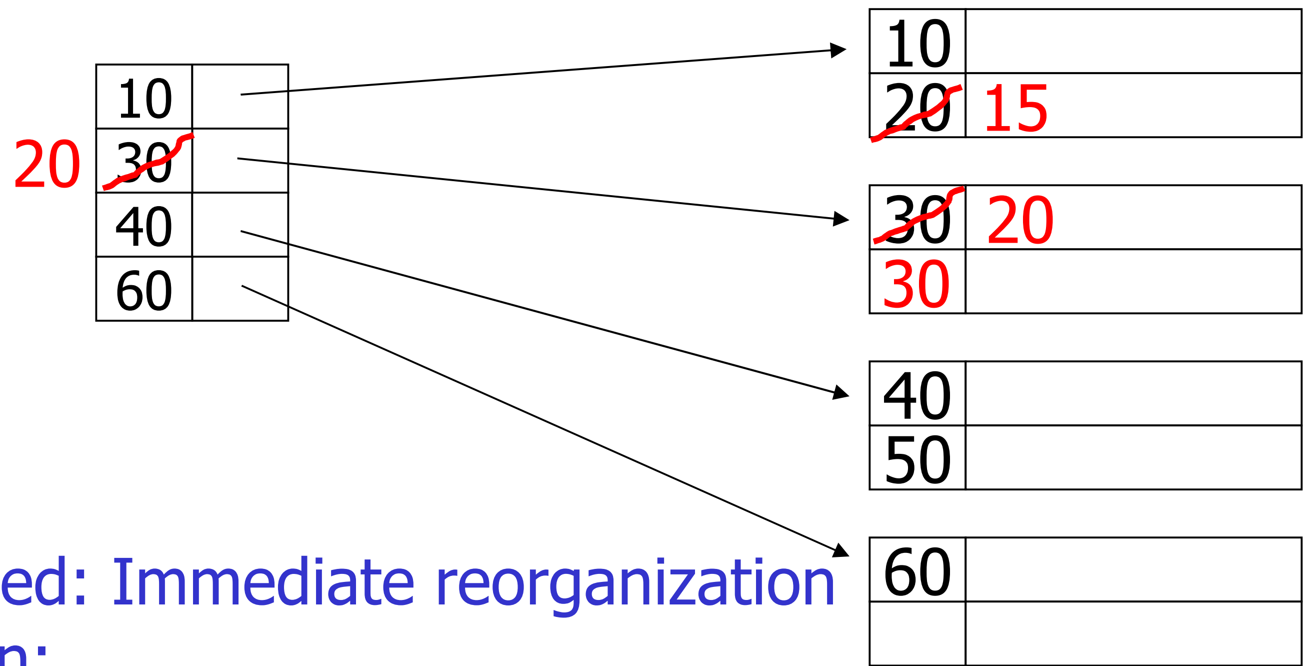
Insertion, sparse index case

– insert record 15



Insertion, sparse index case

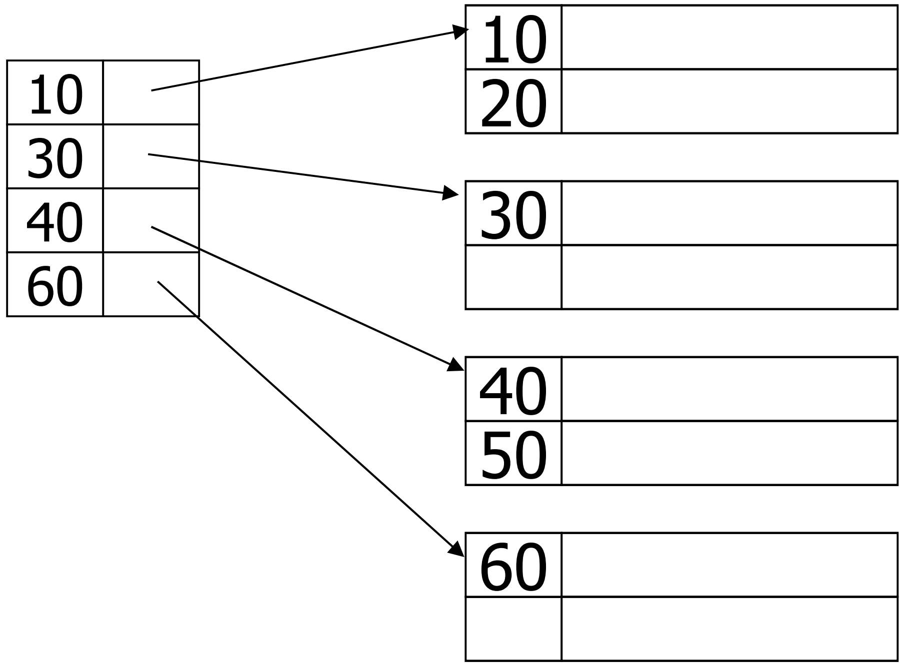
– insert record 15



- Illustrated: Immediate reorganization
- Variation:
  - insert new block (chained)

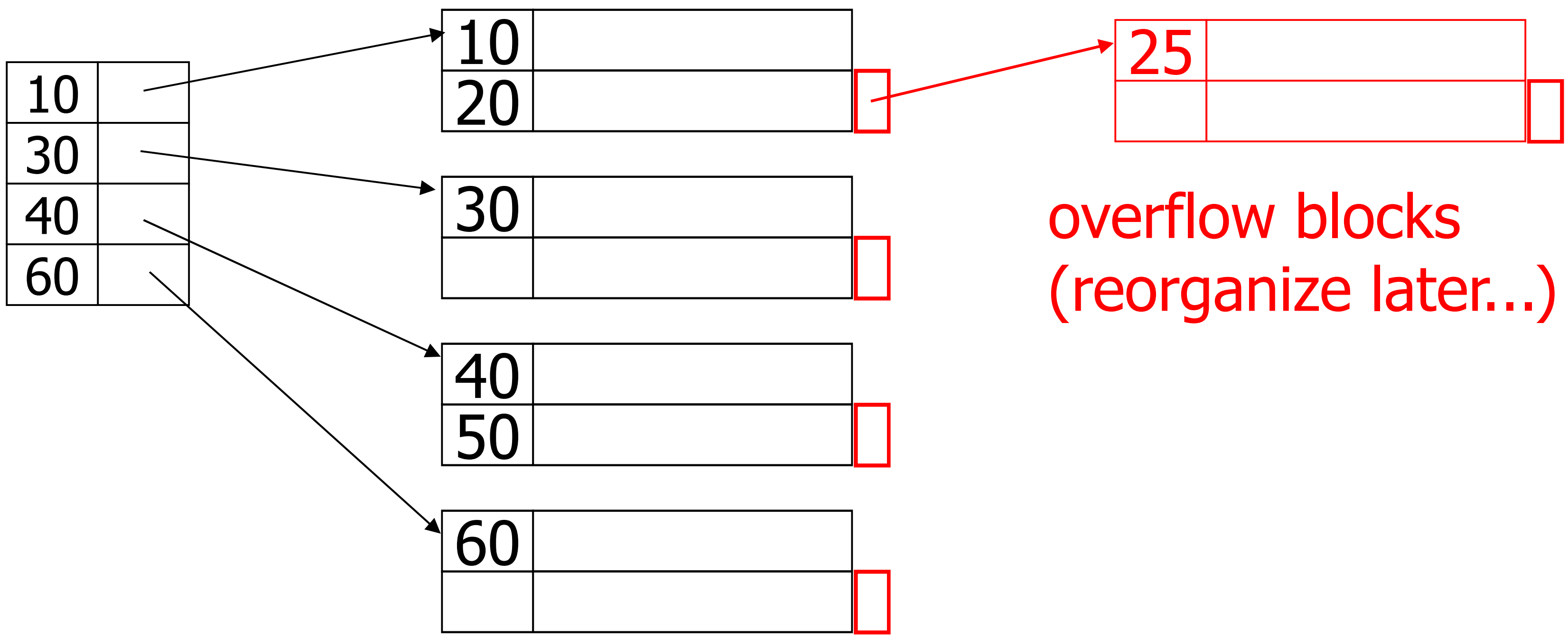
Insertion, sparse index case

– insert record 25



Insertion, sparse index case

– insert record 25

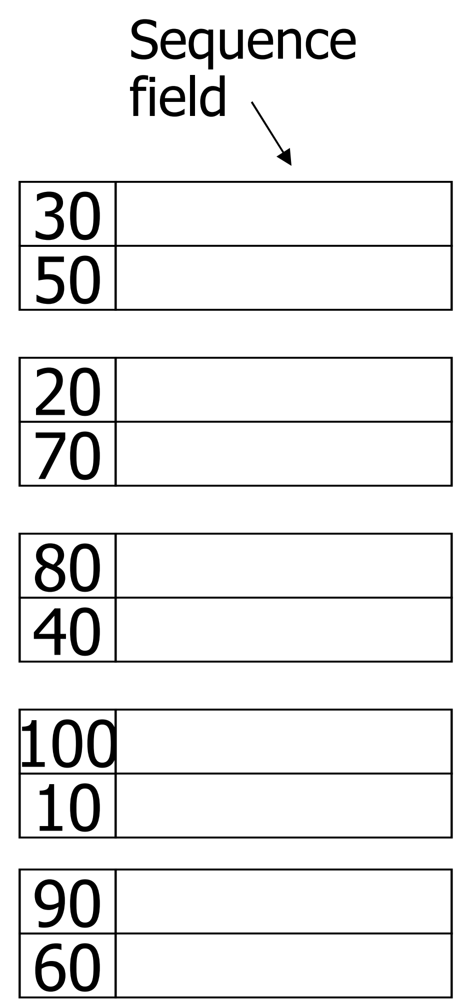


overflow blocks
(reorganize later...)

Insertion, dense index case

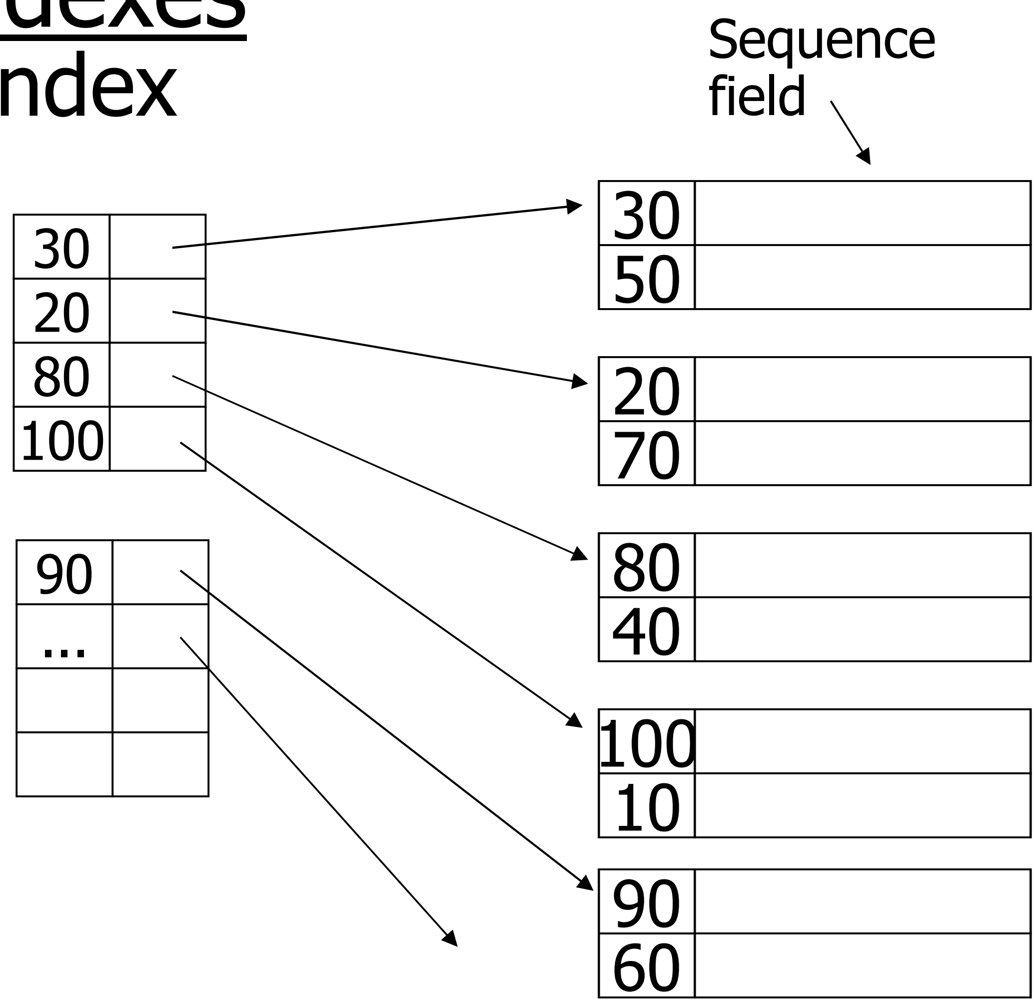
- Similar
- Often more expensive . . .

Secondary indexes



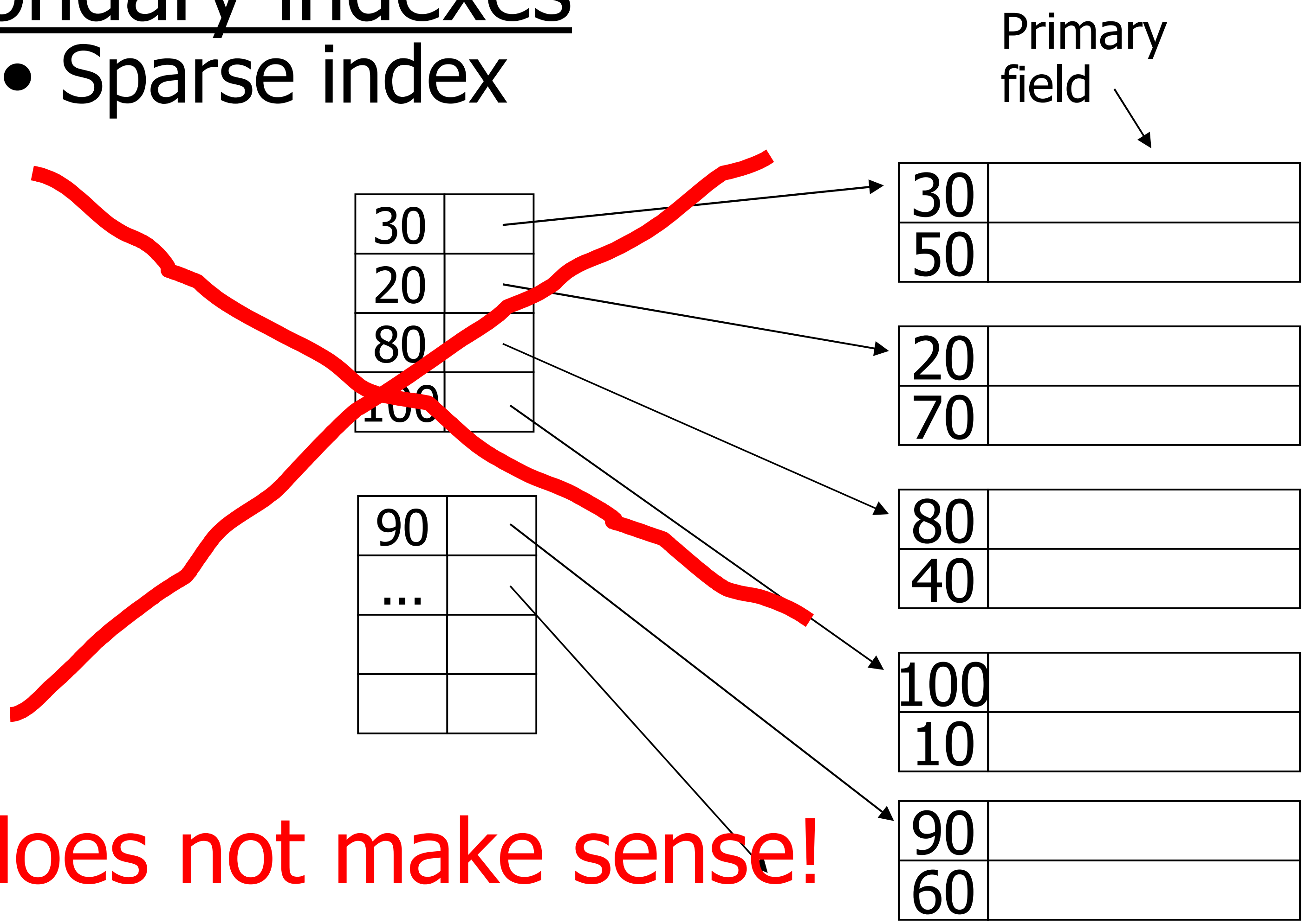
Secondary indexes

- Sparse index



Secondary indexes

- Sparse index



does not make sense!

Secondary indexes

- Dense index

Sequence field

30	
50	

20	
70	

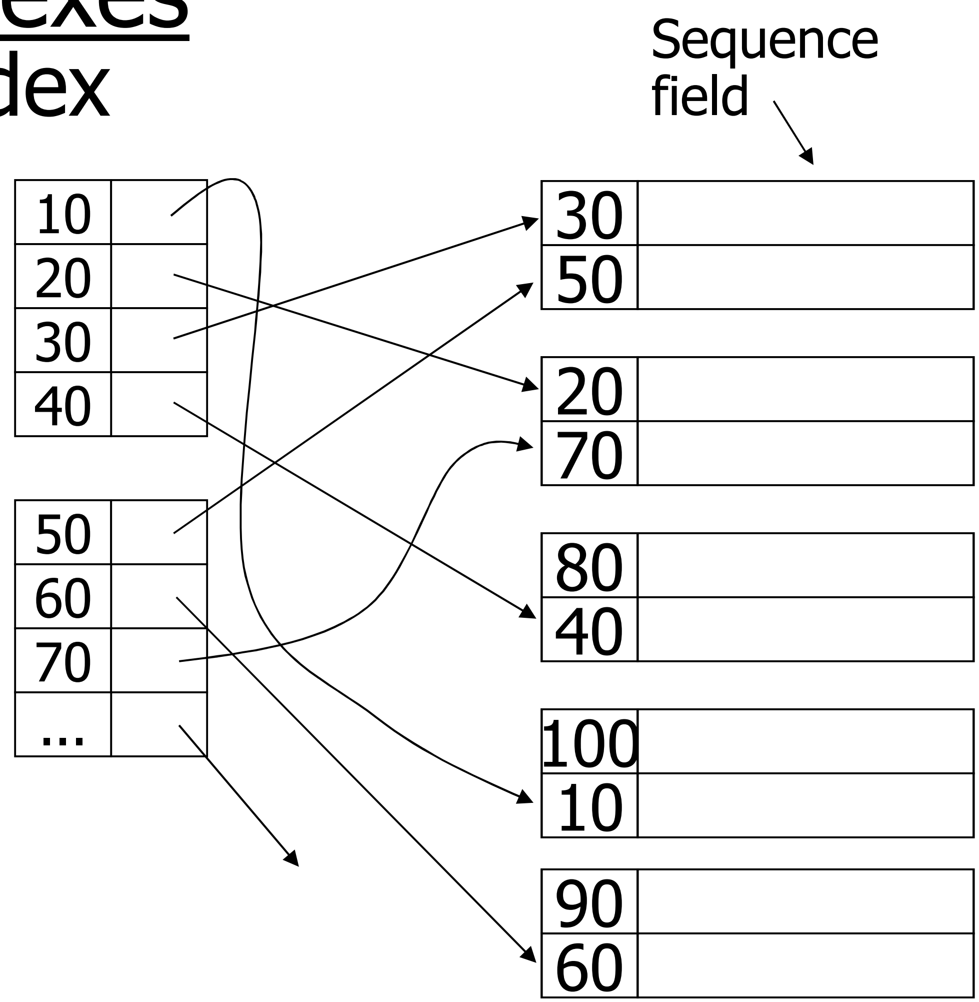
80	
40	

100	
10	

90	
60	

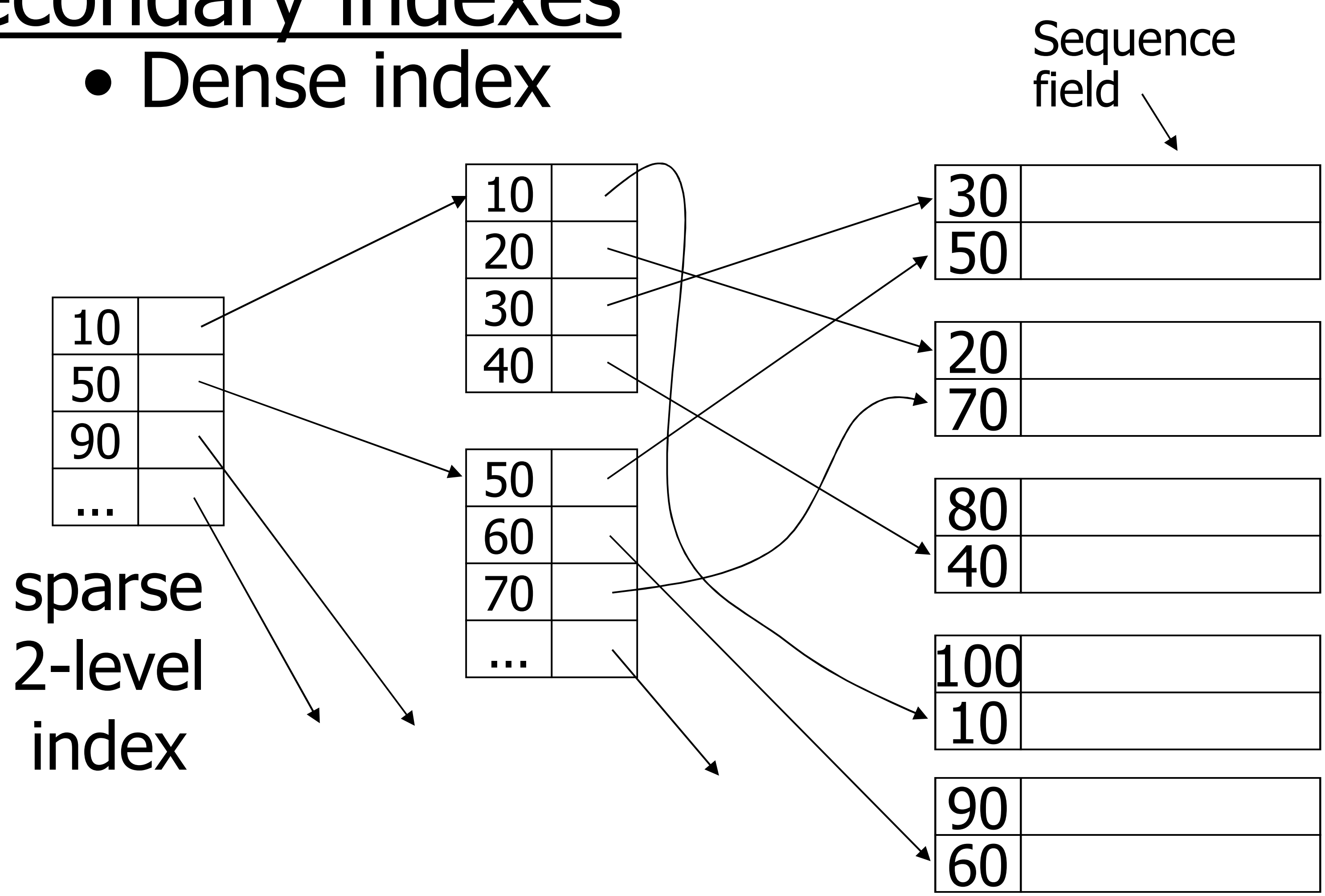
Secondary indexes

- Dense index



Secondary indexes

- Dense index



With secondary indexes:

- Lowest level is dense
- Other levels are sparse

Duplicate values & secondary indexes

20	
10	

20	
40	

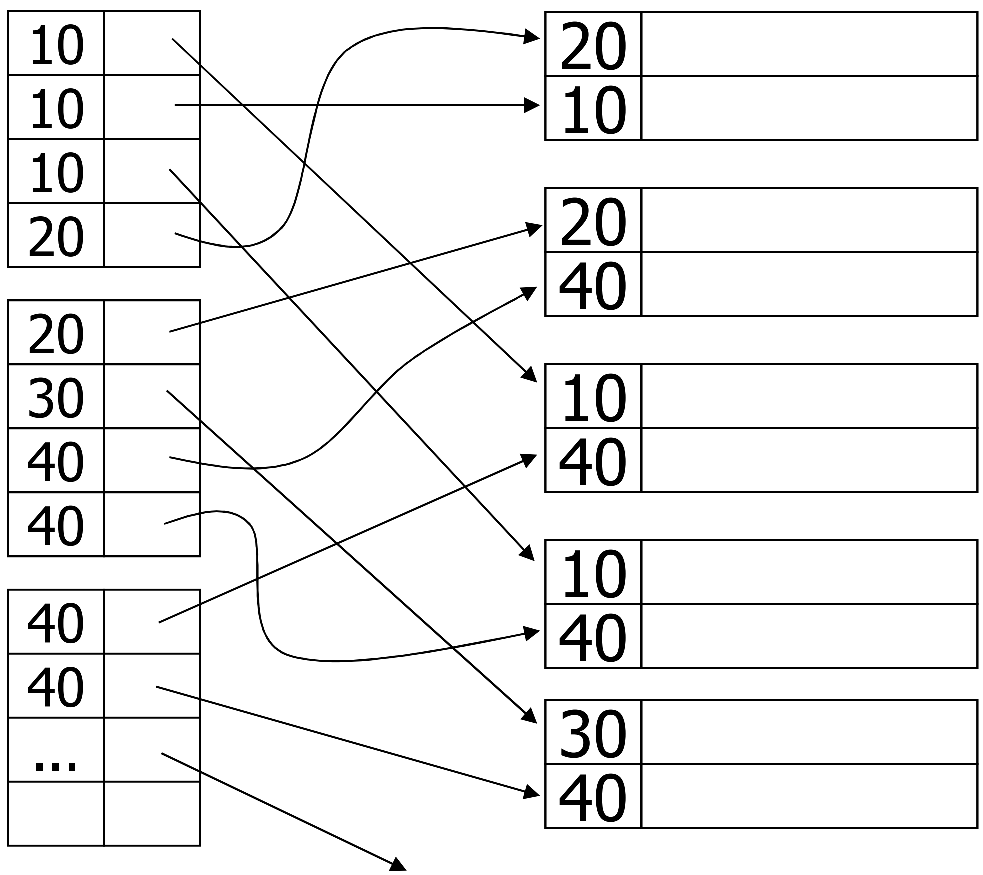
10	
40	

10	
40	

30	
40	

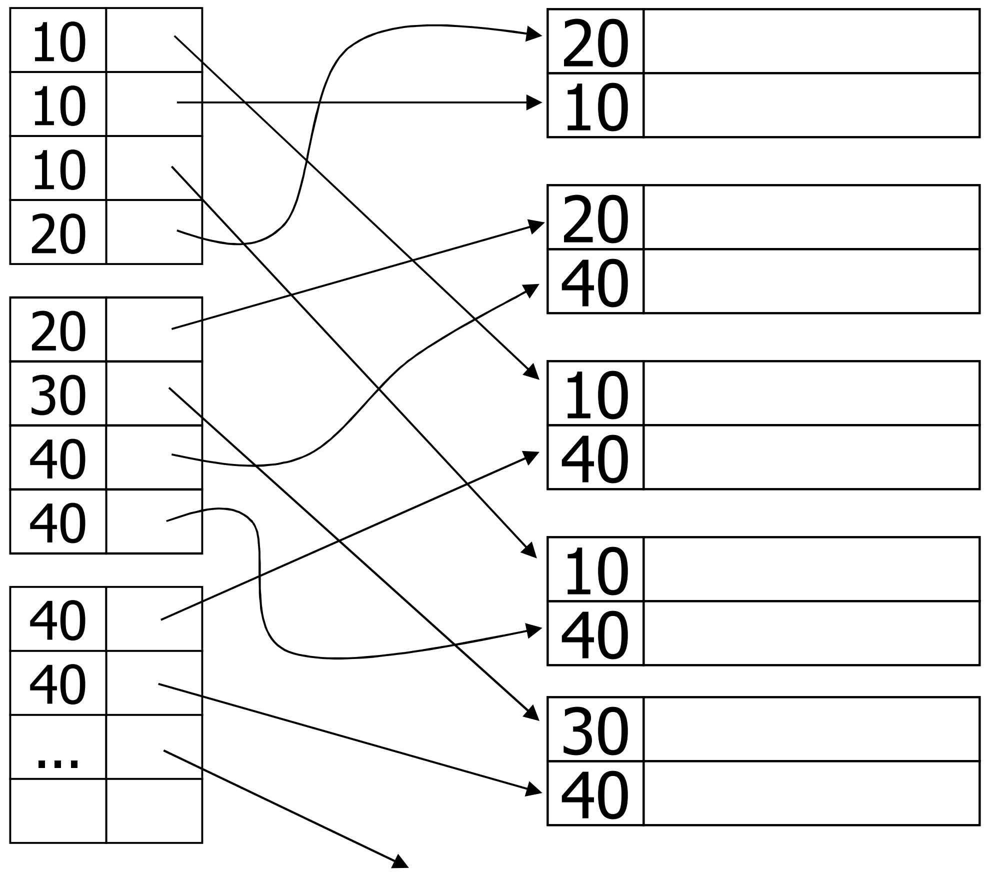
Duplicate values & secondary indexes

one option...



Duplicate values & secondary indexes

one option...



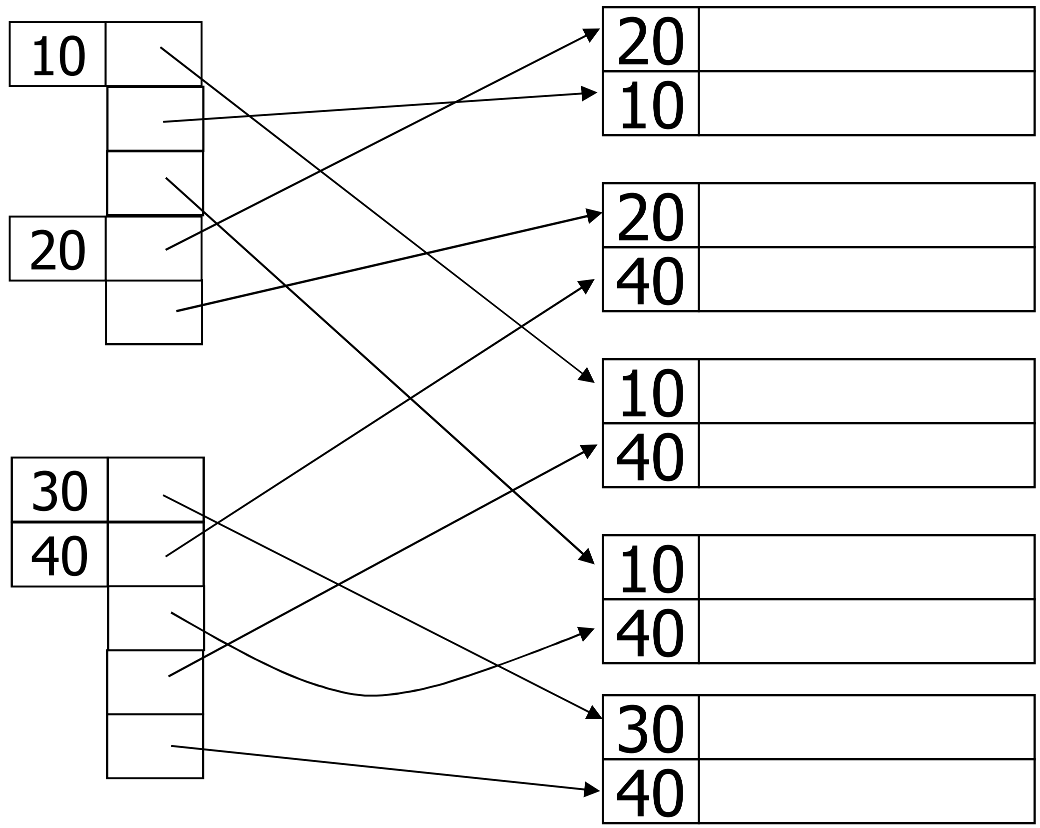
Problem:

excess overhead!

- disk space
- search time

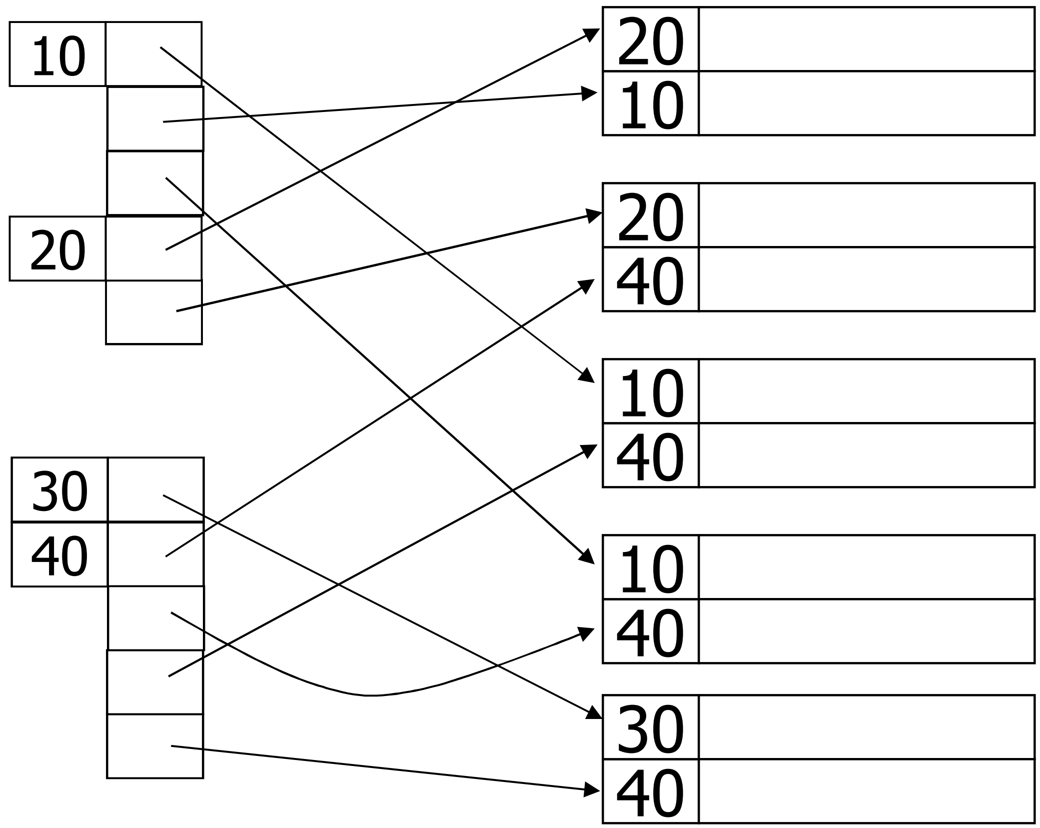
Duplicate values & secondary indexes

another option...



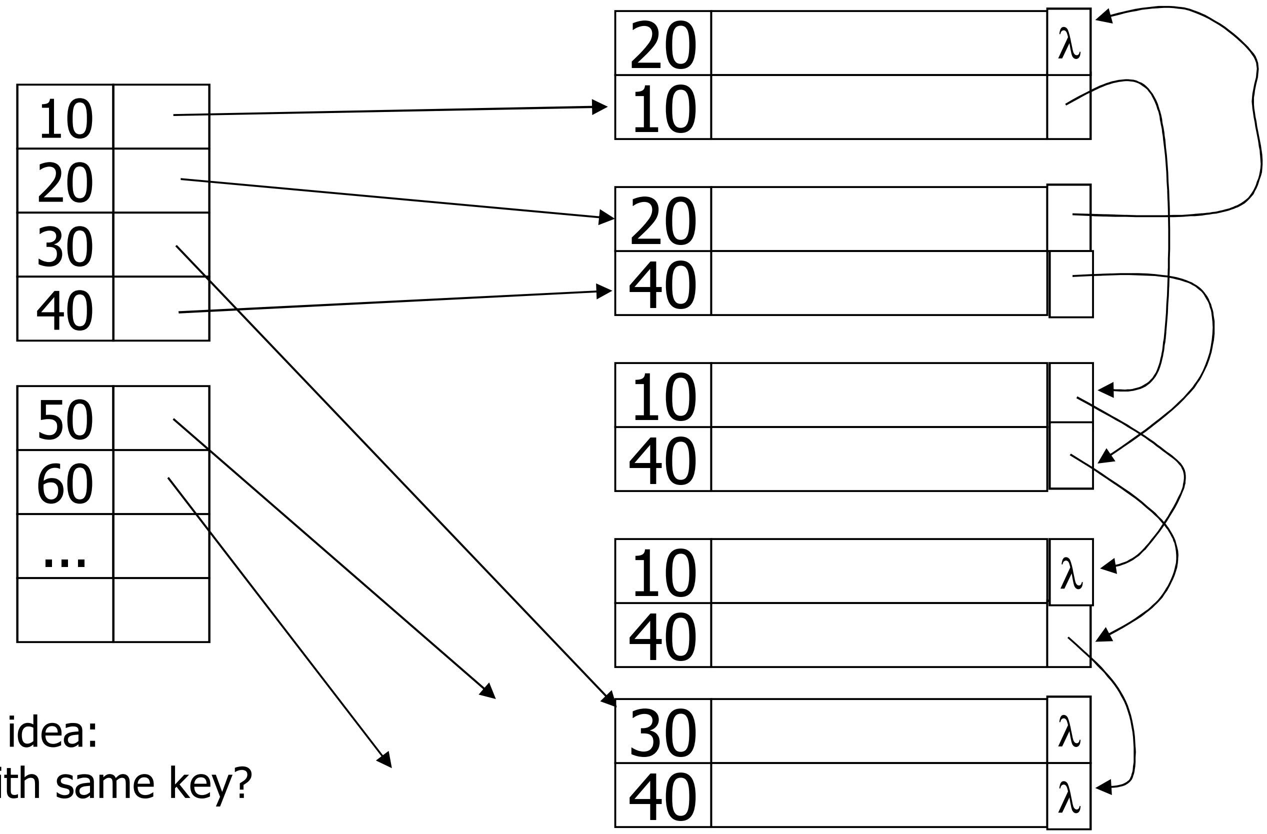
Duplicate values & secondary indexes

another option: clustering...



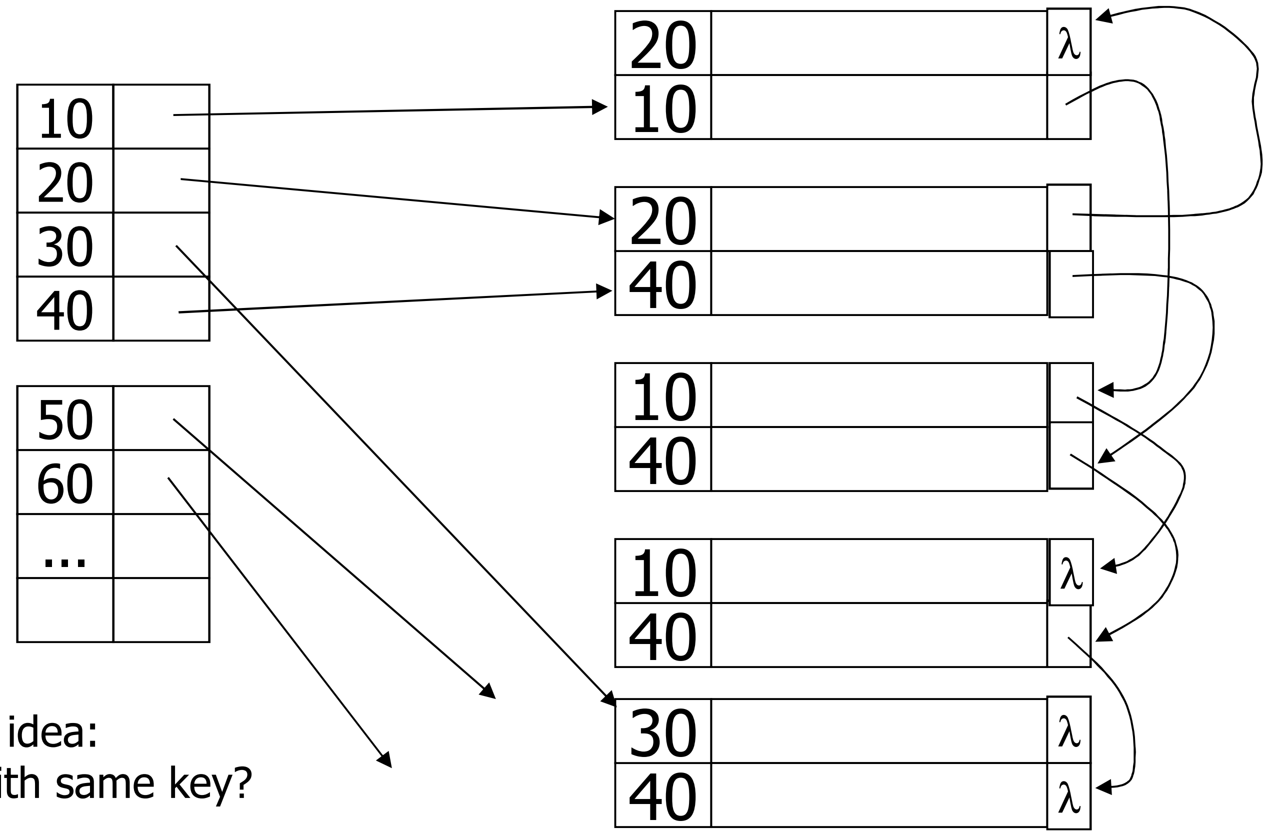
Problem: variable size records in index!

Duplicate values & secondary indexes



Another idea:
Chain records with same key?

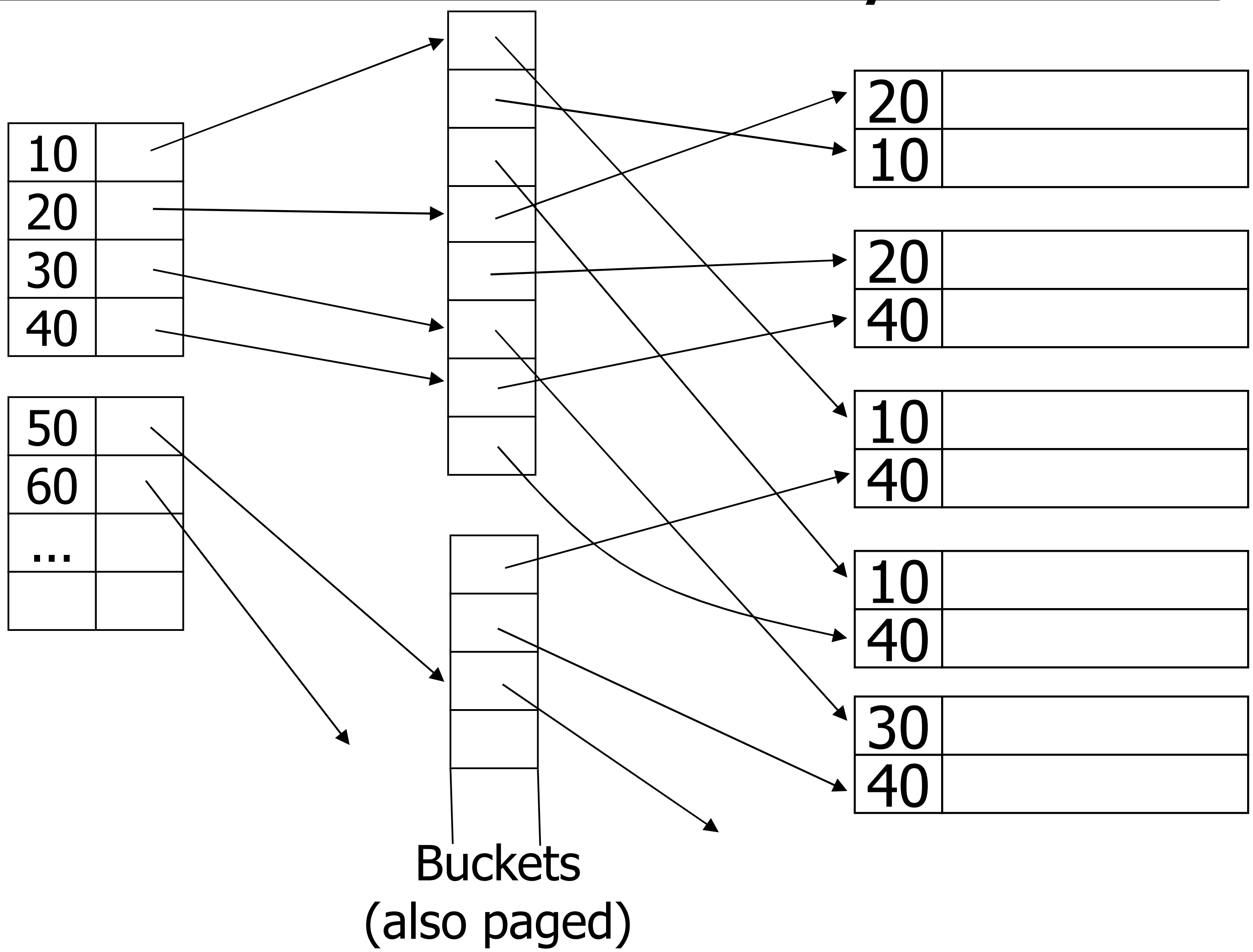
Duplicate values & secondary indexes



Another idea:
Chain records with same key?

Problems:
- Need to add fields to records
- Need to follow chain to know records

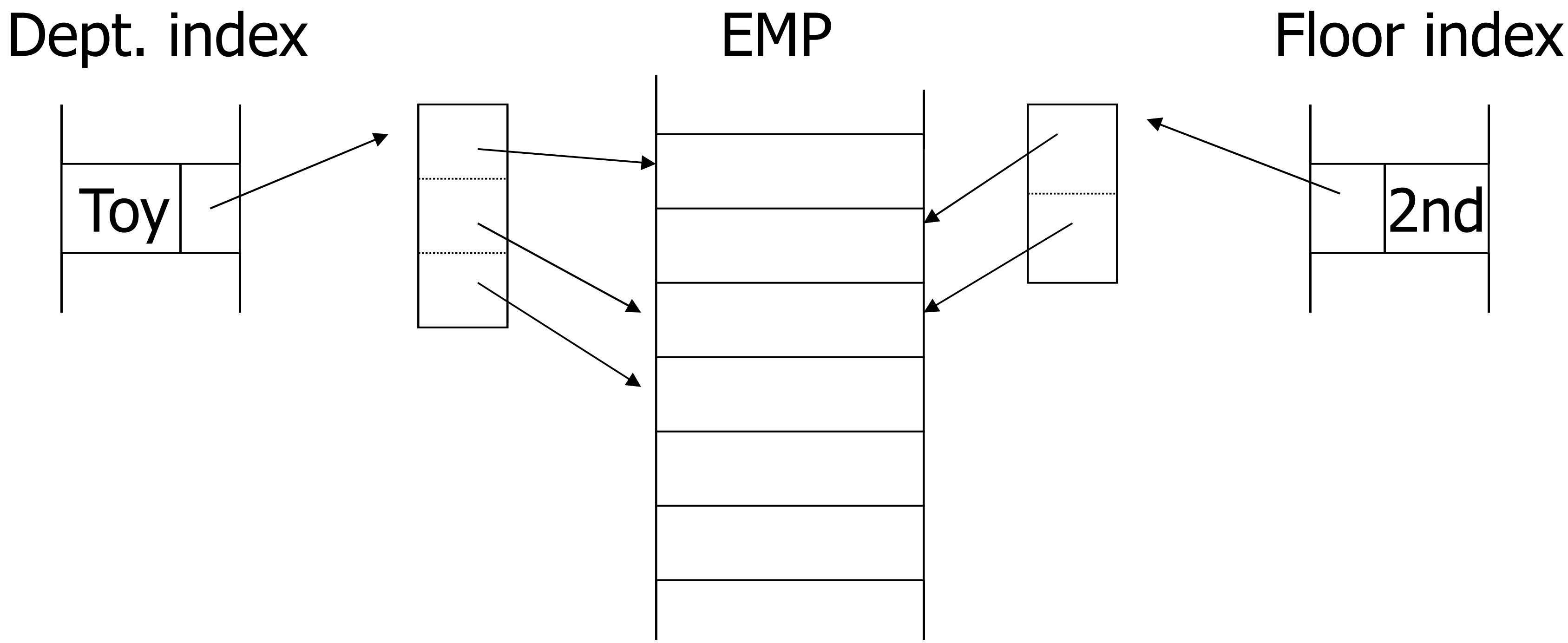
Duplicate values & secondary indexes



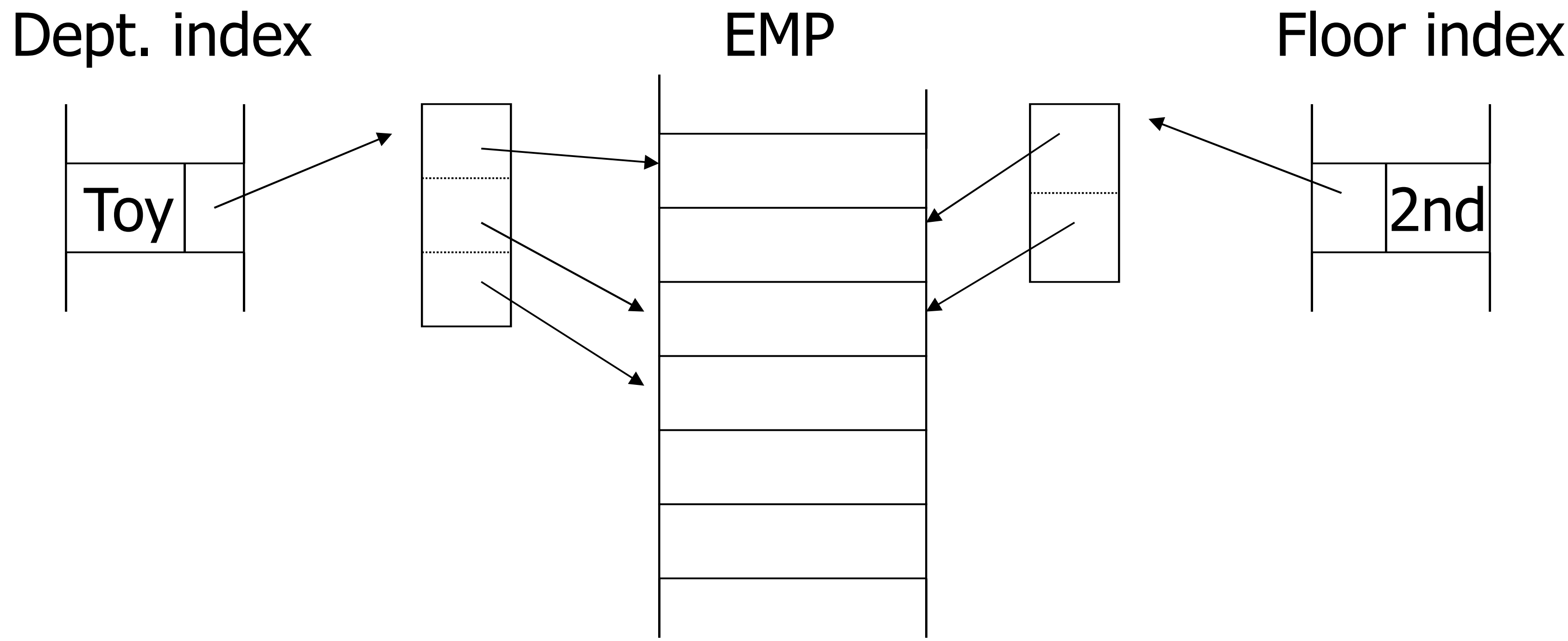
Why "bucket" idea is useful?

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

Query: Get employees in (Toy Dept) $\wedge$ (2nd floor)



Query: Get employees in (Toy Dept) $\wedge$ (2nd floor)



→ 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
    - Buckets of Postings List

Conventional indexes

Advantage:

- Simple
- Index is sequential file
  (good for scans)

Disadvantage:

- Inserts/deletes expensive,

- NEXT: Another type of index
  - Give up on sequentiality of index
  - Try to get fast insert/delete/search

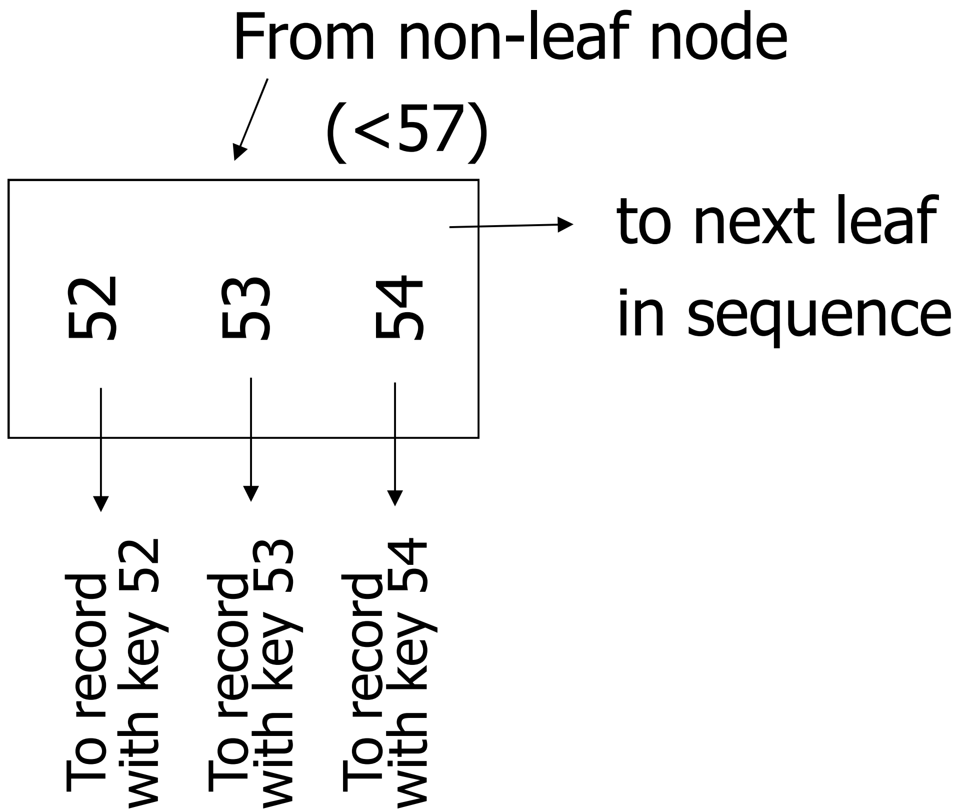
- B+ Tree
  - A data structure used to build an index
  - Just like any tree: basic building blocks;
    - Node (link in CS3)
    - Node consists of data + references to children nodes.
  - All nodes (but root) are on hard-disk → a storage oriented data structures
  - Self-balanced data structure → $O(\log n)$

B+tree rules

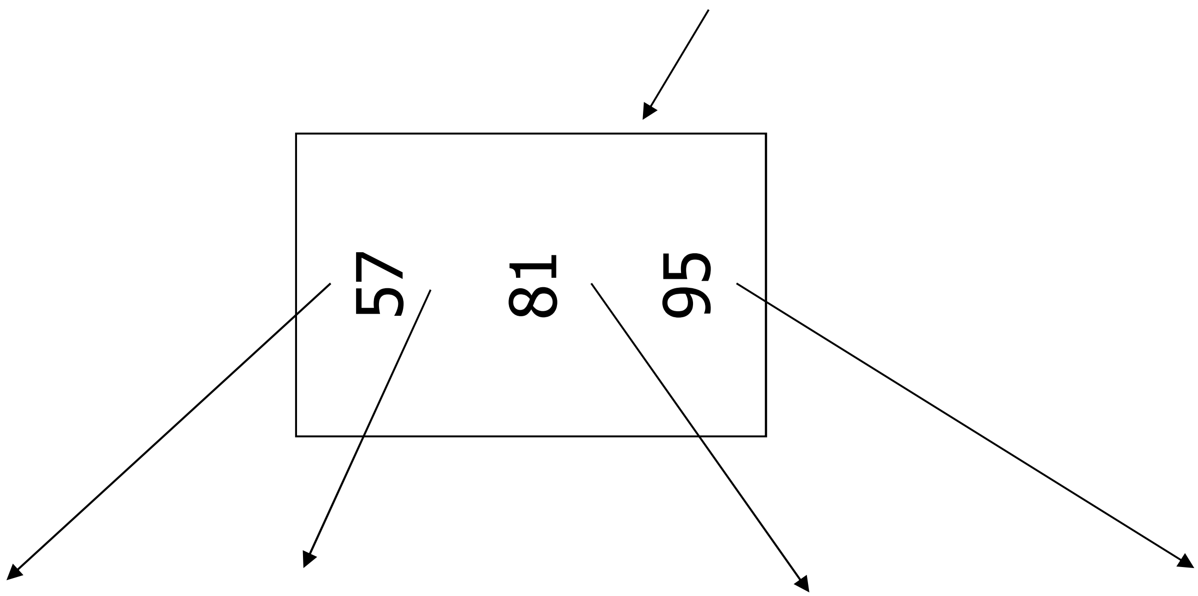
(1) All leaves at the same lowest level
(balanced tree)

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

Sample leaf node:



Sample non-leaf



to keys < 57

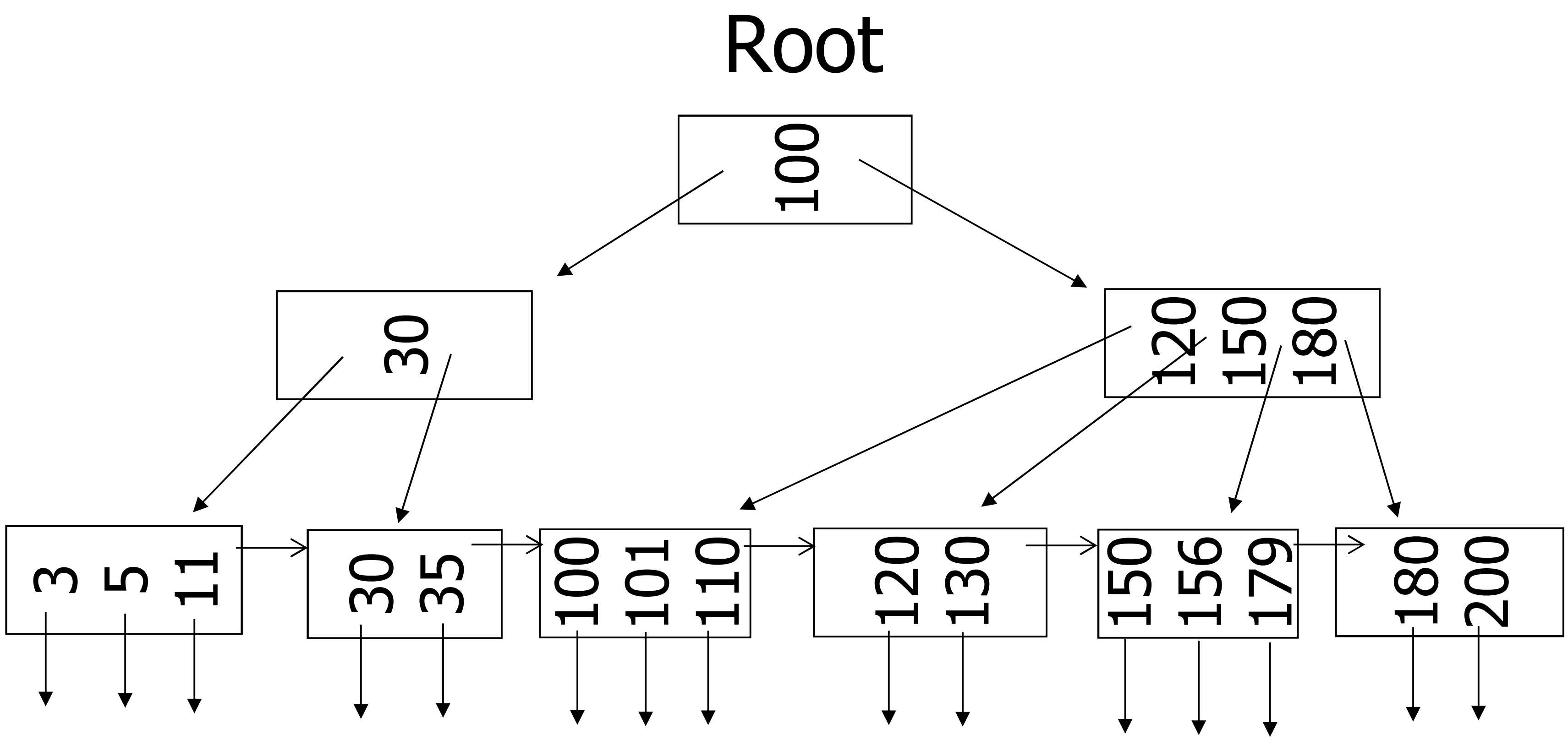
to keys $57 \leq k < 81$

to keys $81 \leq k < 95$

to keys ≥ 95

B+Tree Example

n=3



B+tree rules

(3) Relation between keys and pointers

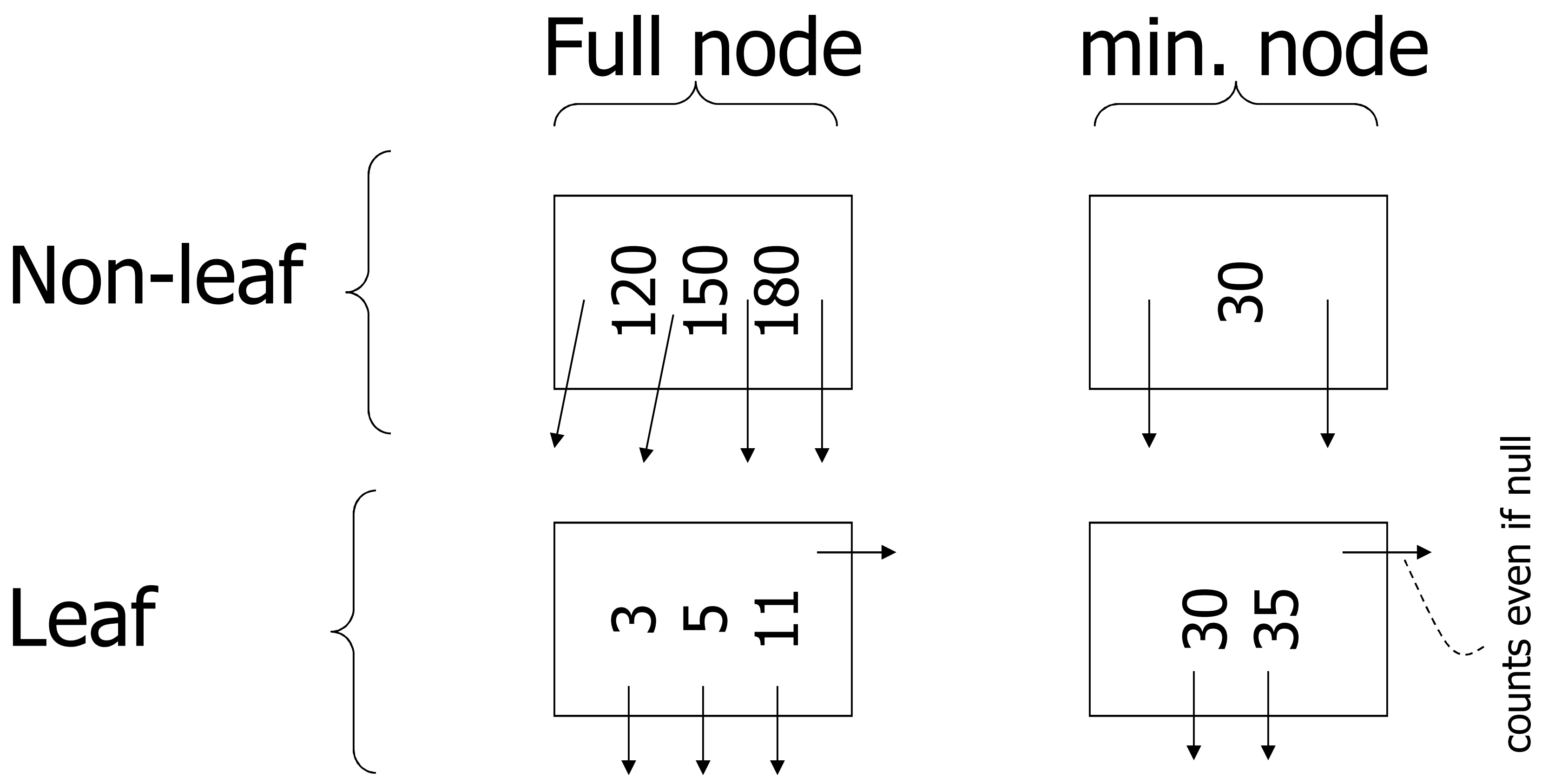
$\left\{ \begin{array}{l} \text{n keys} \\ \text{n+1 pointers} \end{array} \right.$ (fixed)

Don't want nodes to be too empty

- Use at least (minimum)

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

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

(4) 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	1	1

B+tree rules

(5) Insertion/deletion algorithm guarantee the tree is always balanced

→ e.g. of an amortized algorithm

Insert into B+tree

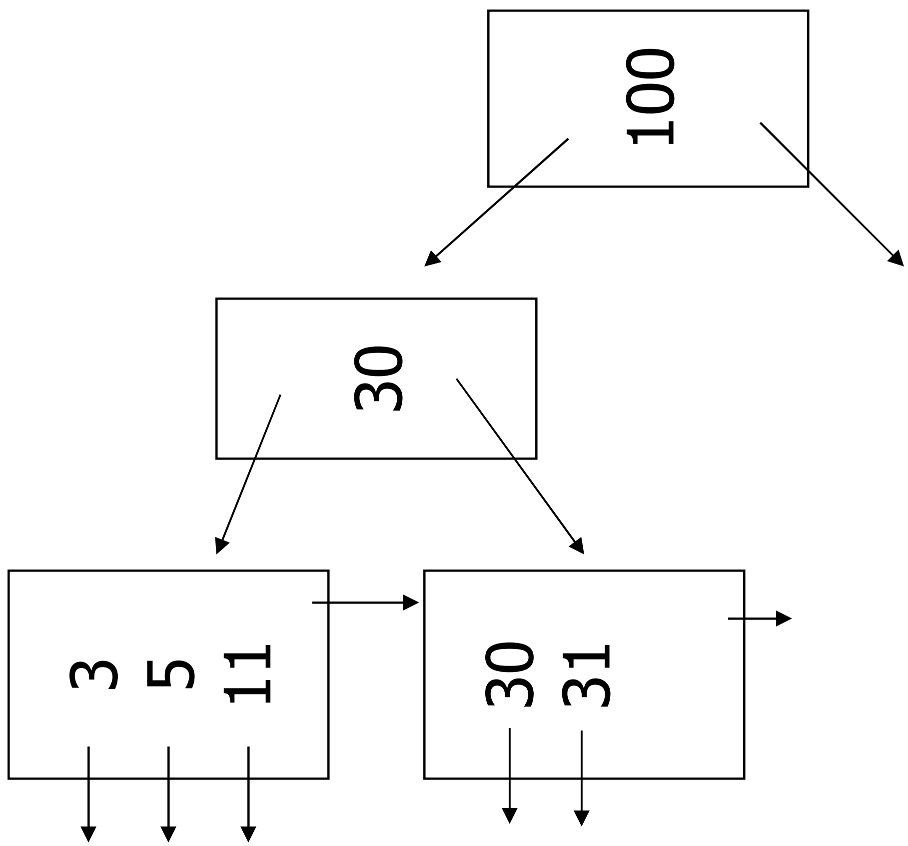
(a) simple case
- space available in leaf

(b) leaf overflow

(c) non-leaf overflow

(d) new root

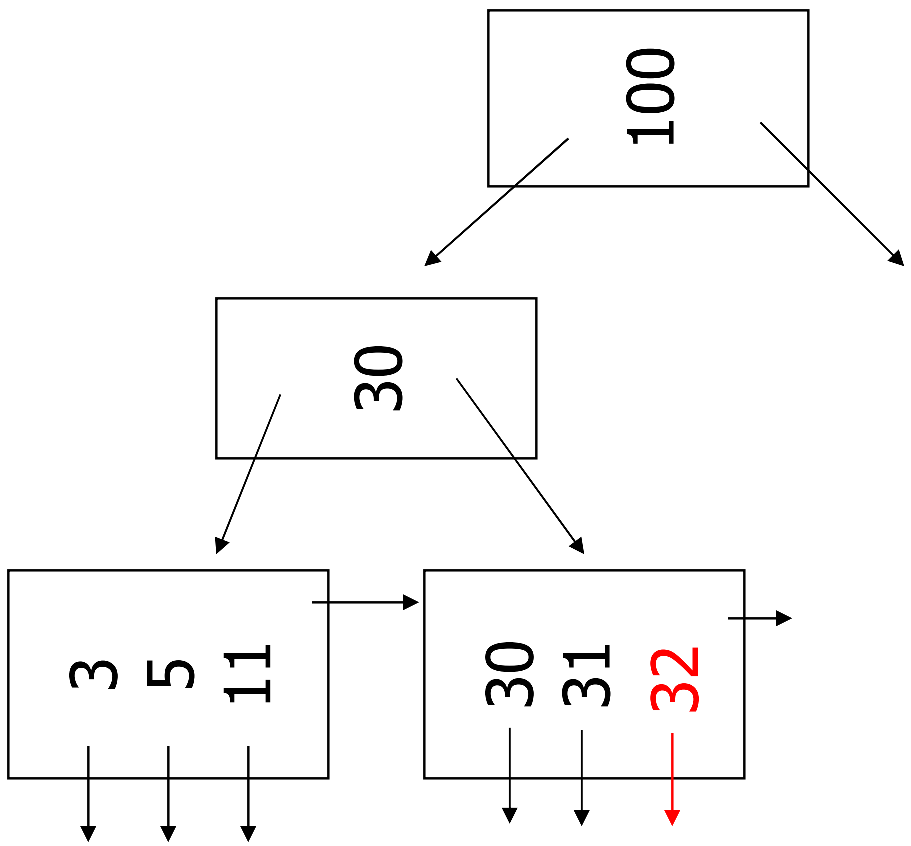
(a) Insert key = 32



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

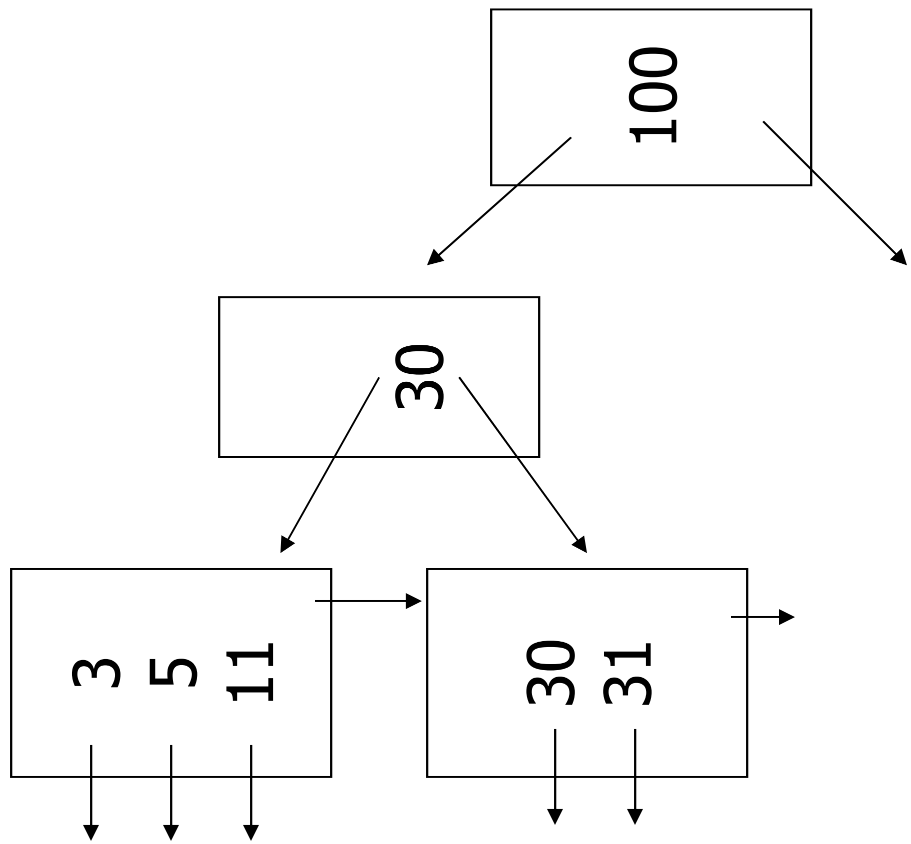
(a) Insert key = 32



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

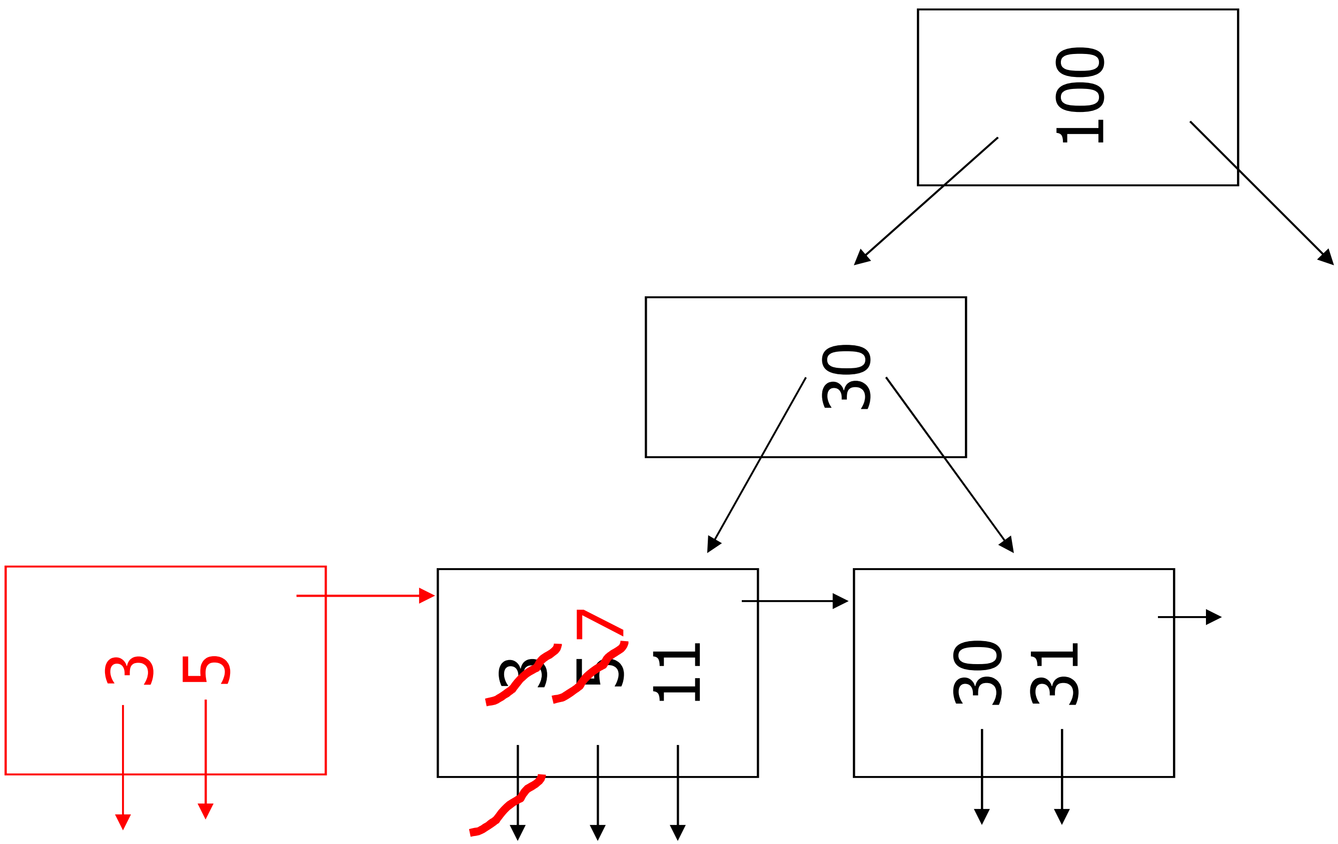
(a) Insert key = 7



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

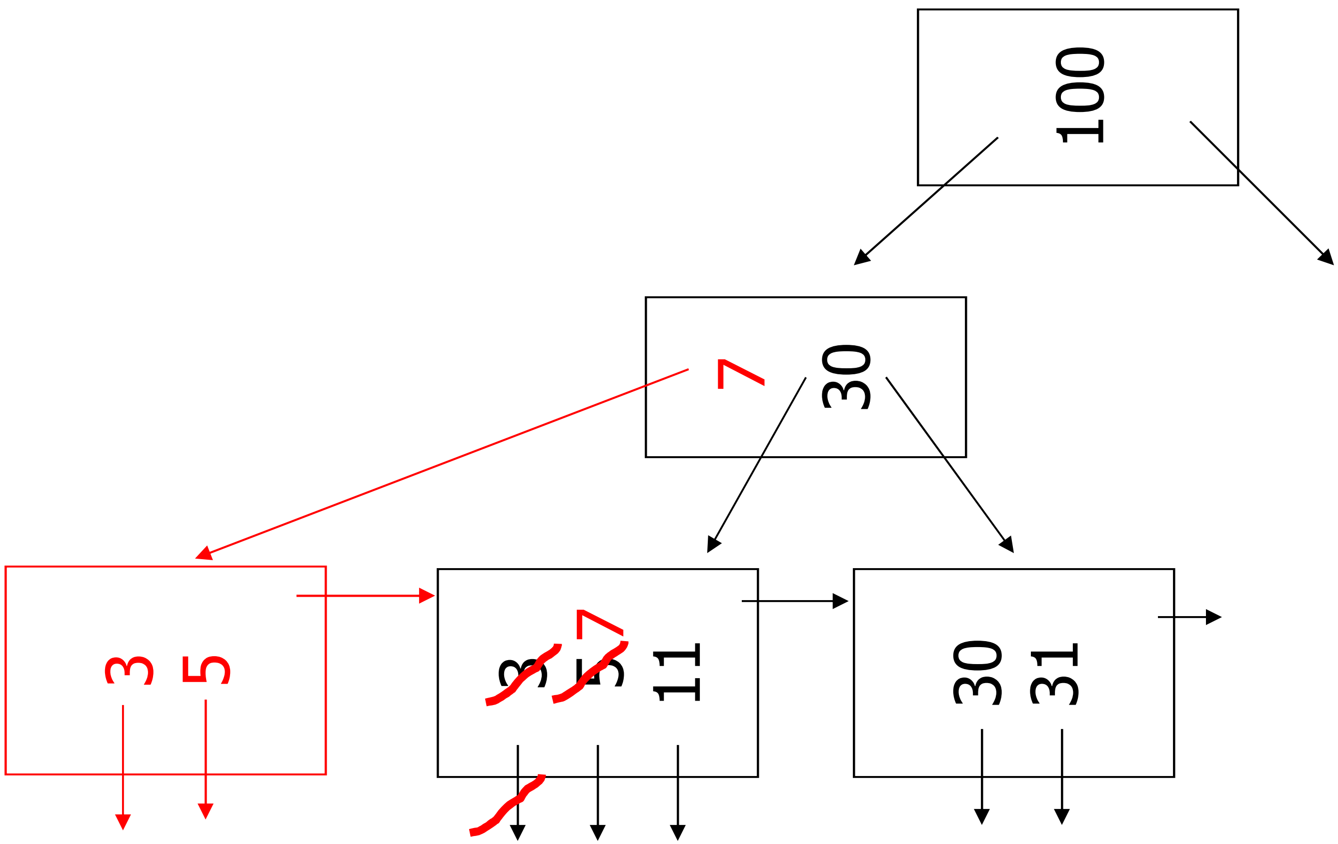
(a) Insert key = 7



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

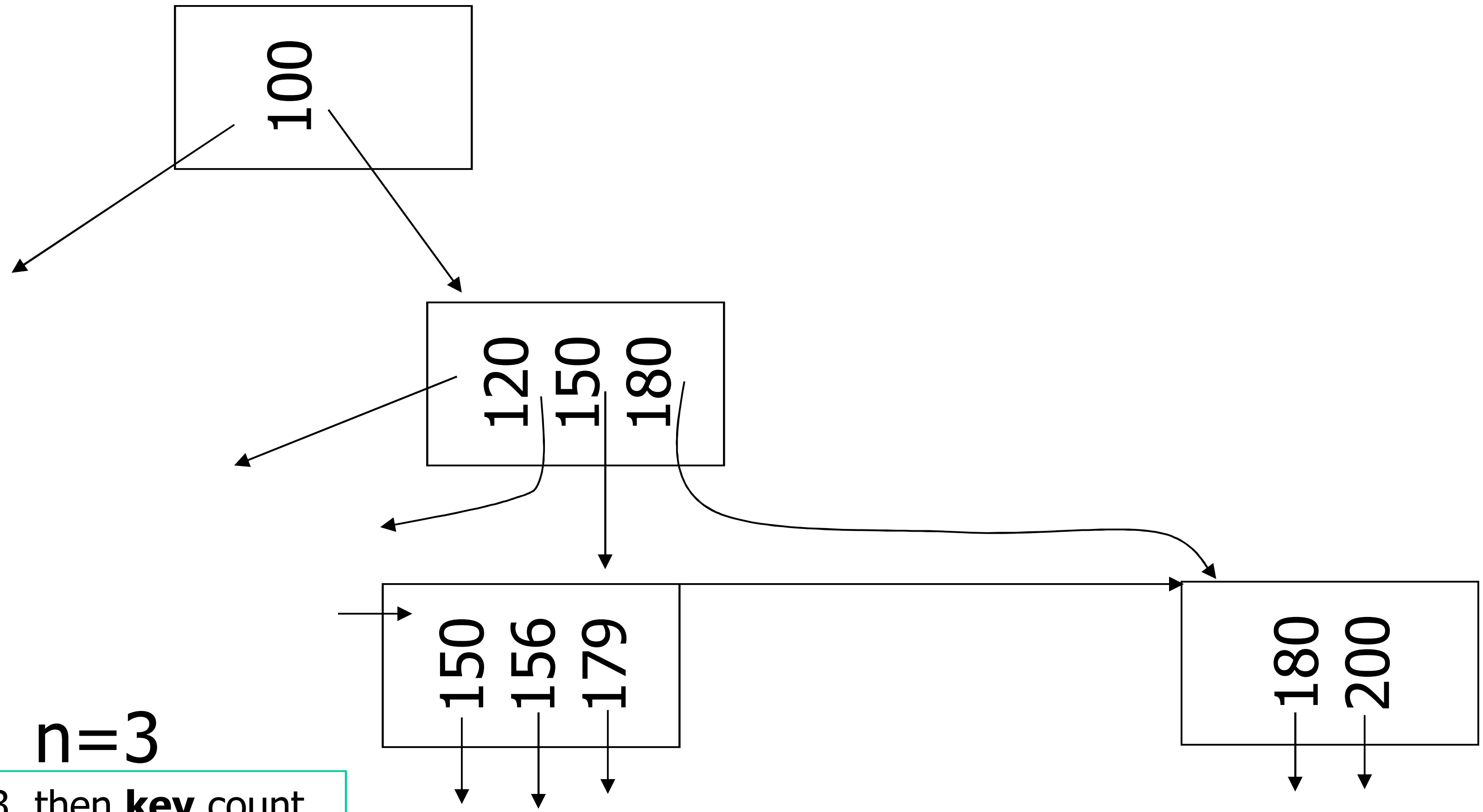
(a) Insert key = 7



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

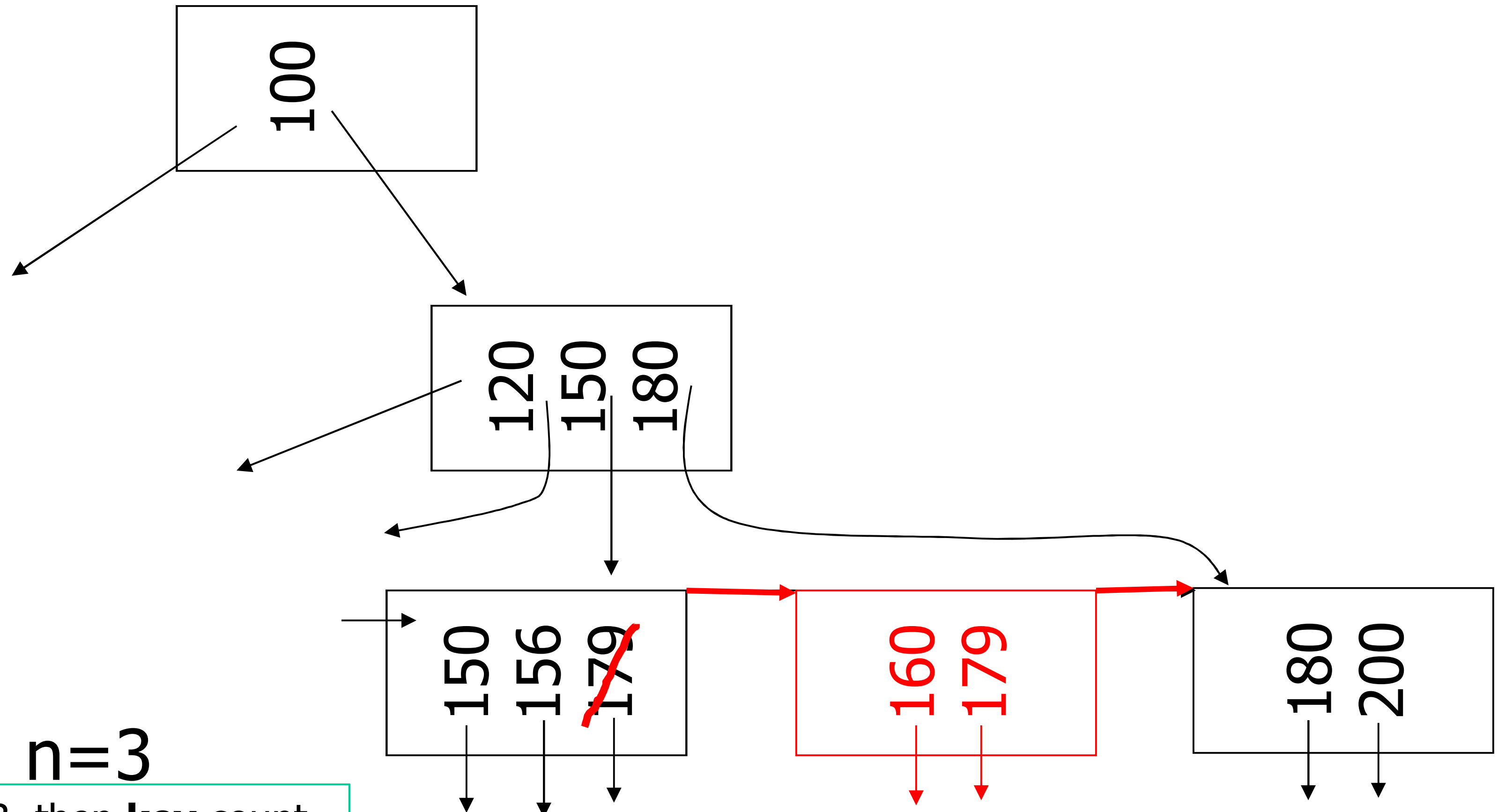
(c) Insert key = 160



n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

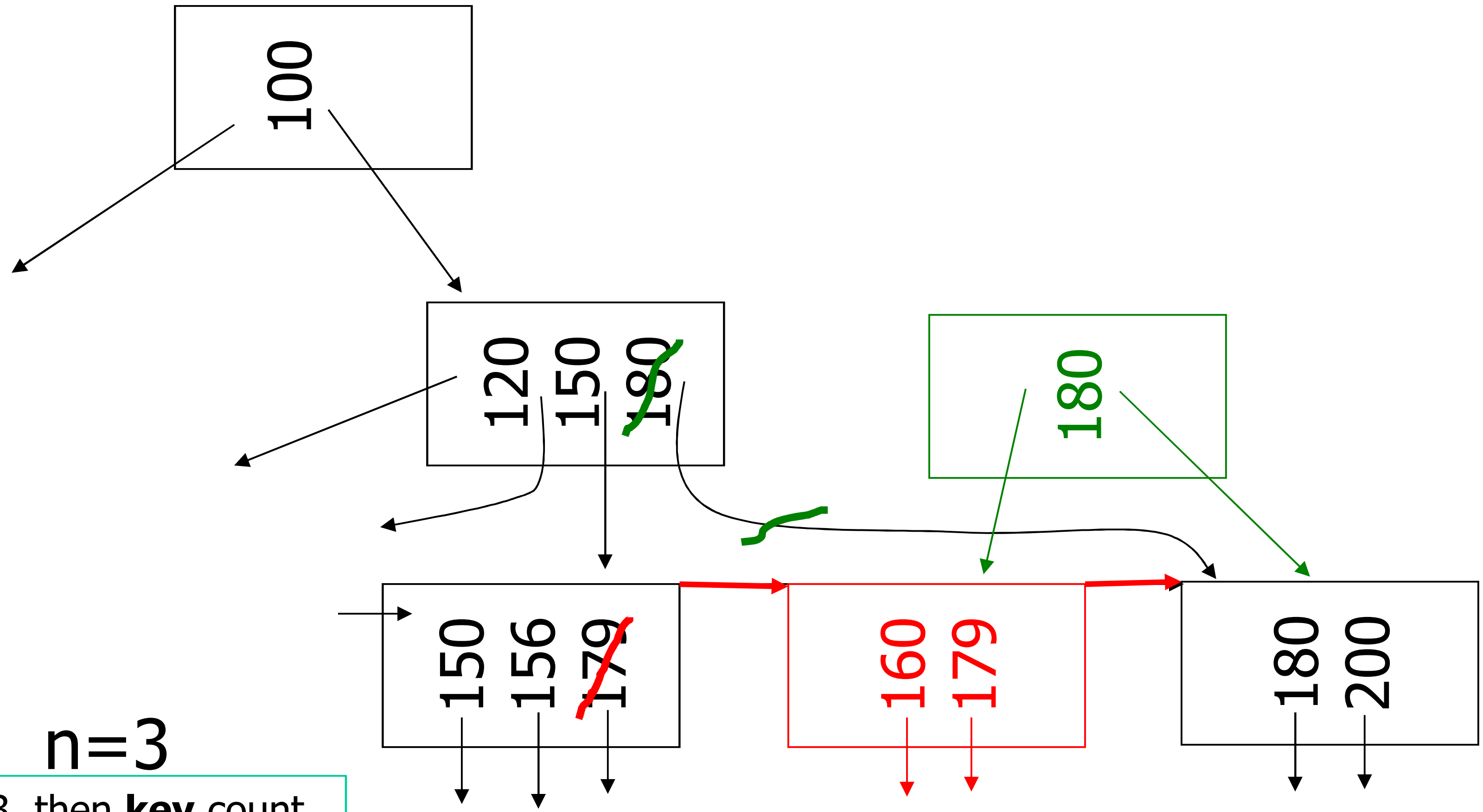
(c) Insert key = 160



n=3

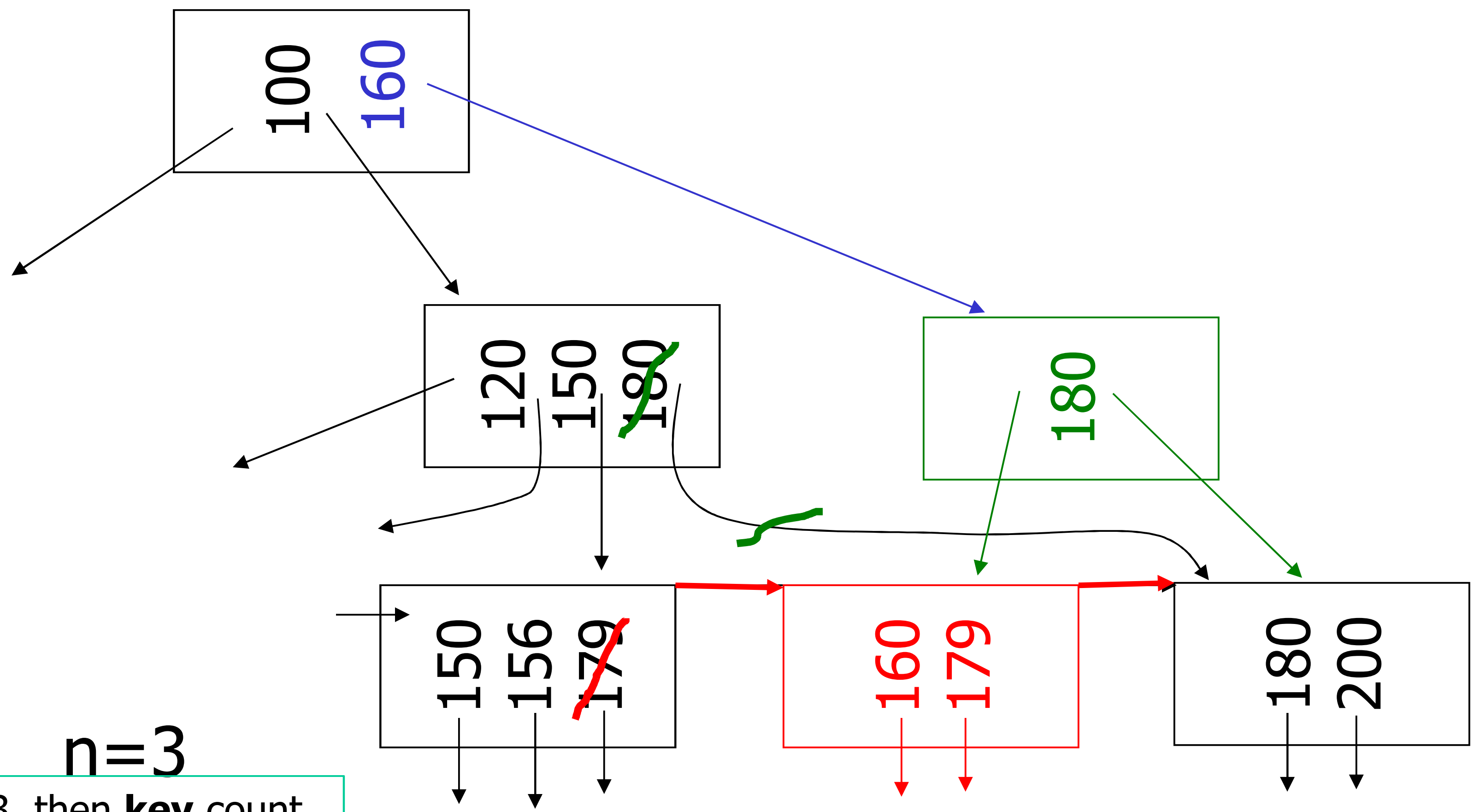
if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

(c) Insert key = 160



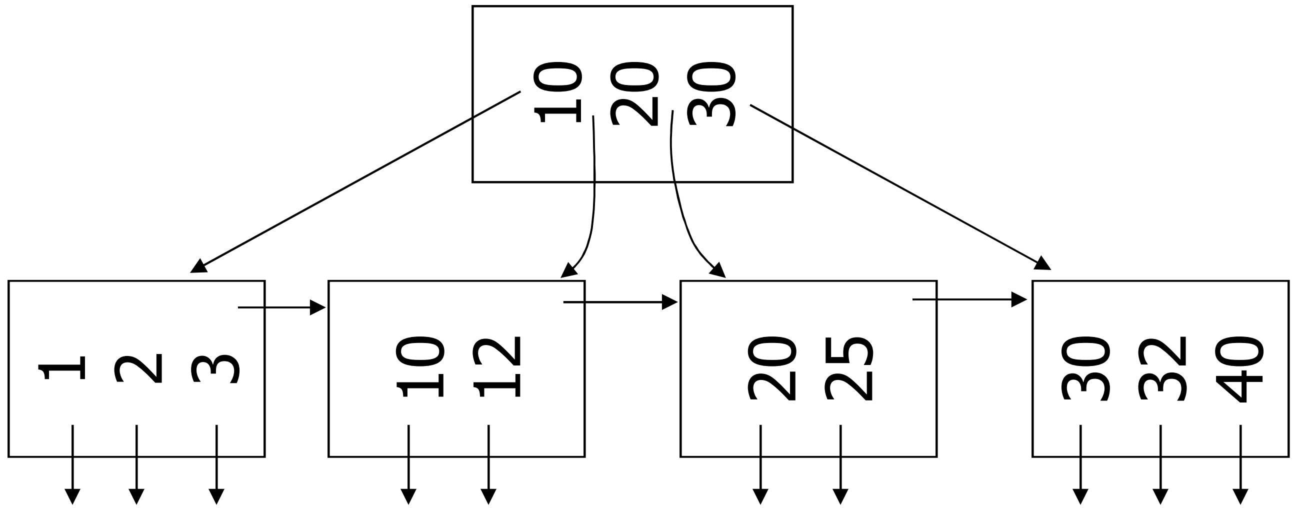
if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

(c) Insert key = 160



if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

(d) New root, insert 45



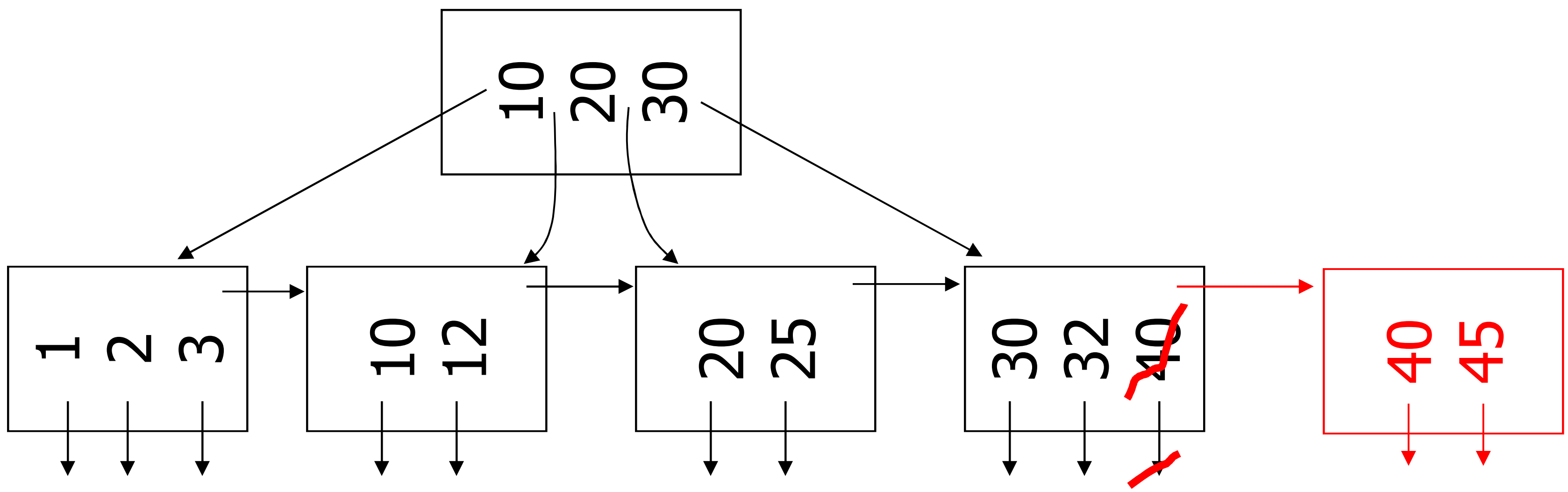
n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2

(d) New root, insert 45

n=3

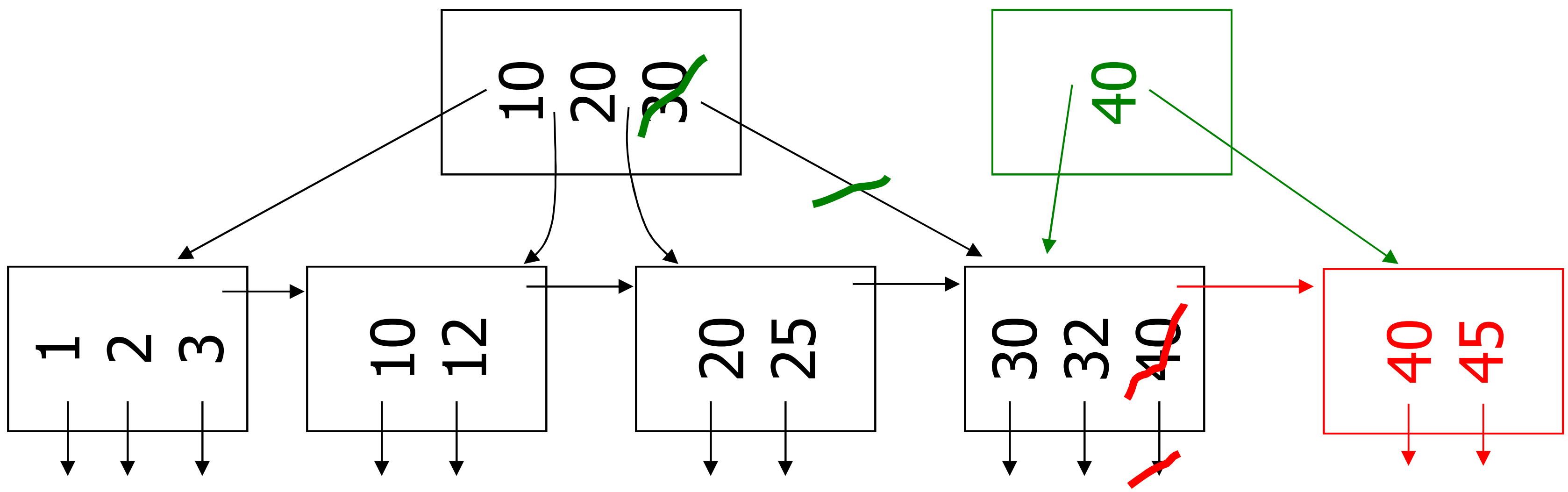
if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2



(d) New root, insert 45

n=3

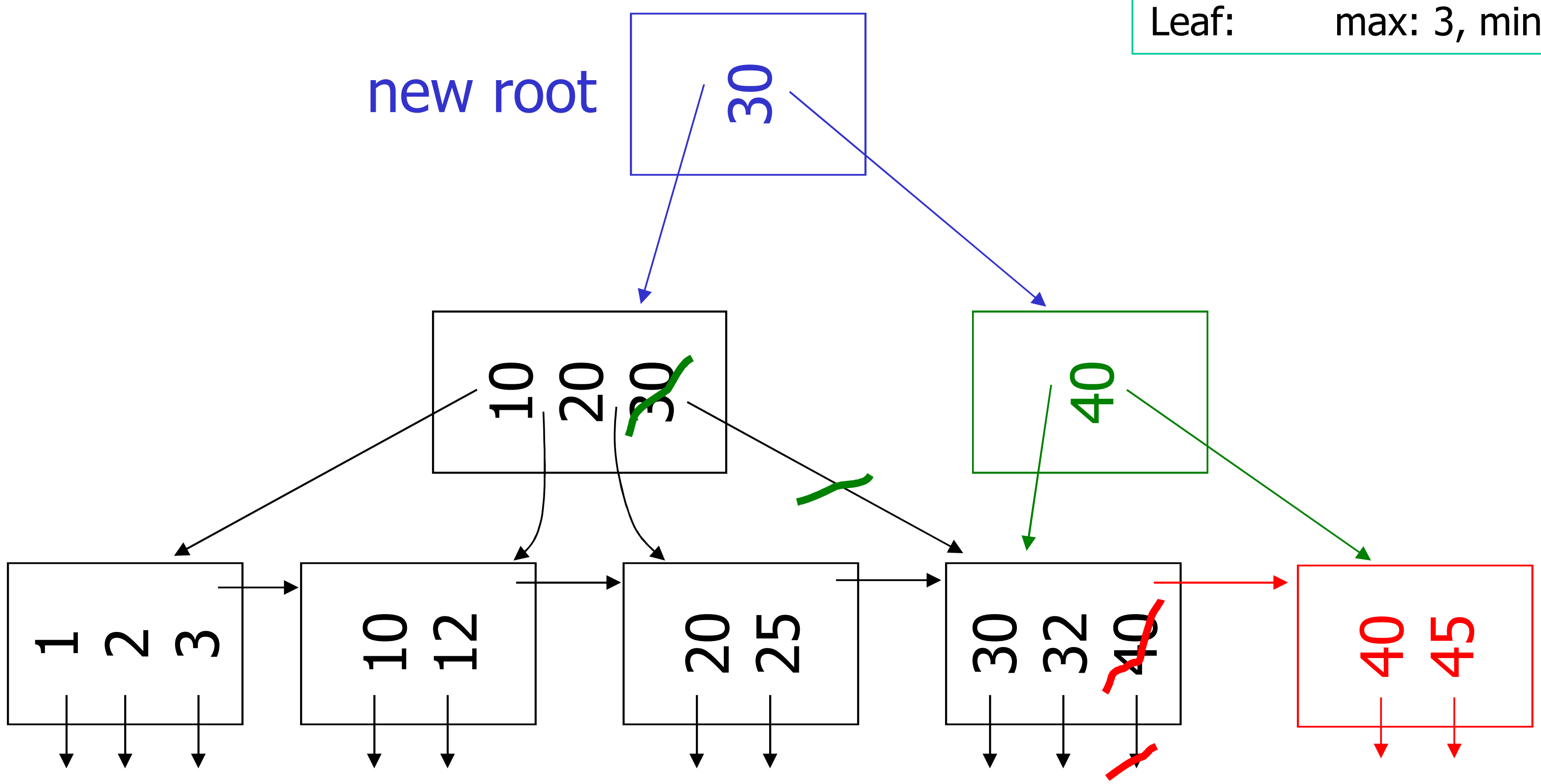
if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2



(d) New root, insert 45

n=3

if n = 3, then **key** count
Non-leaf: max: 3, min: 1
Leaf: max: 3, min: 2



Read

- Course Textbook: pages 619-648
  - dropbox:
  https://www.dropbox.com/s/fqv14g1zqhhl6k5/Chaps14-19.pdf
  - gdrive:
  https://drive.google.com/file/d/0B03SaNyIsL_2U2pmMjVrMDl4MWs/view?usp=sharing

- http://en.wikipedia.org/wiki/B+_tree