

Advanced Database Technology
Anna Östlin Pagh and Rasmus Pagh
IT University of Copenhagen
Spring 2004

February 19, 2004

INDEXING I

Lecture based on [GUW, 13.0-13.2]

Slides based on
Notes 04: Indexing
for Stanford CS 245, fall 2002
by Hector Garcia-Molina

Today

- Why indexing?
- Conventional indexes (dense/sparse)
- Multi-level indexes
- Secondary indexes
- **Next time:** B-tree and hash indexes

Why indexing?

- Common queries involve conditions on the values of attributes, e.g.
`SELECT * FROM R WHERE a=11`
`SELECT * FROM R WHERE 0<=b and b<42`
- **indexing** an attribute (or set of attributes) speeds up finding tuples with specific values.
(And gives other speed-ups as well.)
- Conceptually similar to index in a book.

Problem session

- Consider an index data structure "similar to" the index in a book.
- How many steps does it take to find the occurrences of a specific term?
- What about the number of I/Os?
- Does your encyclopaedia have an index?

Sequential File

10	
20	

30	
40	

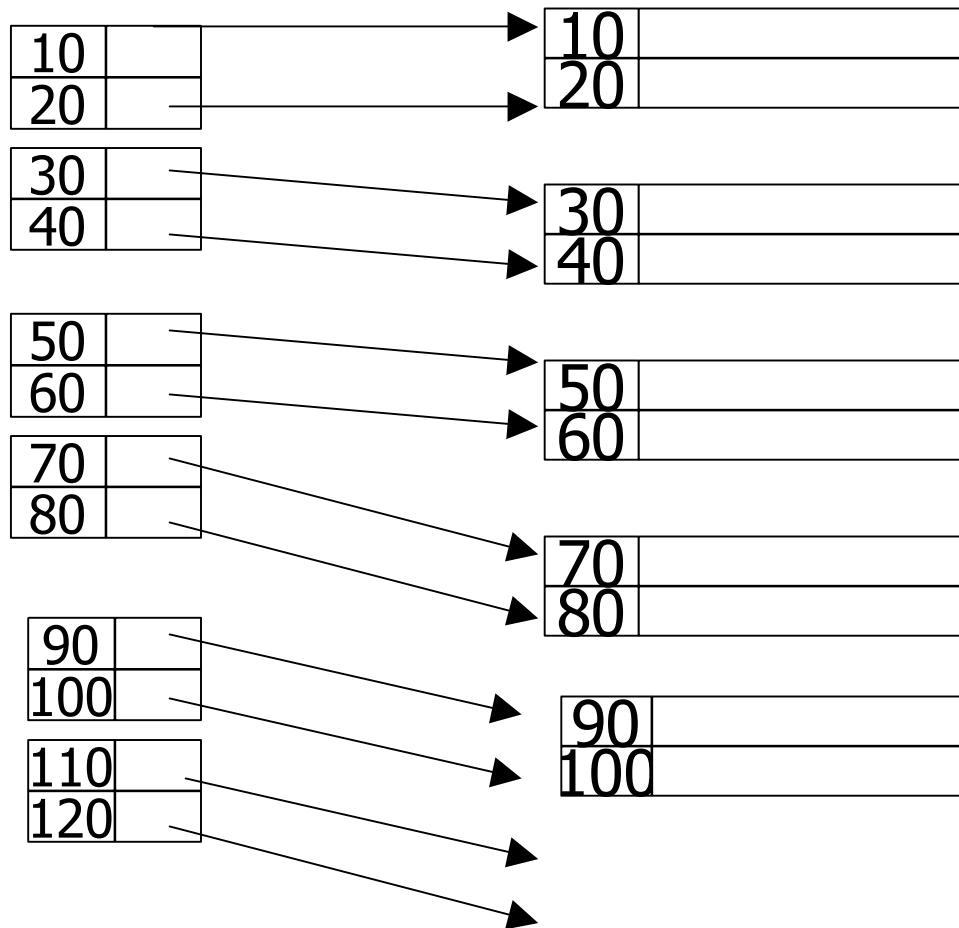
50	
60	

70	
80	

90	
100	

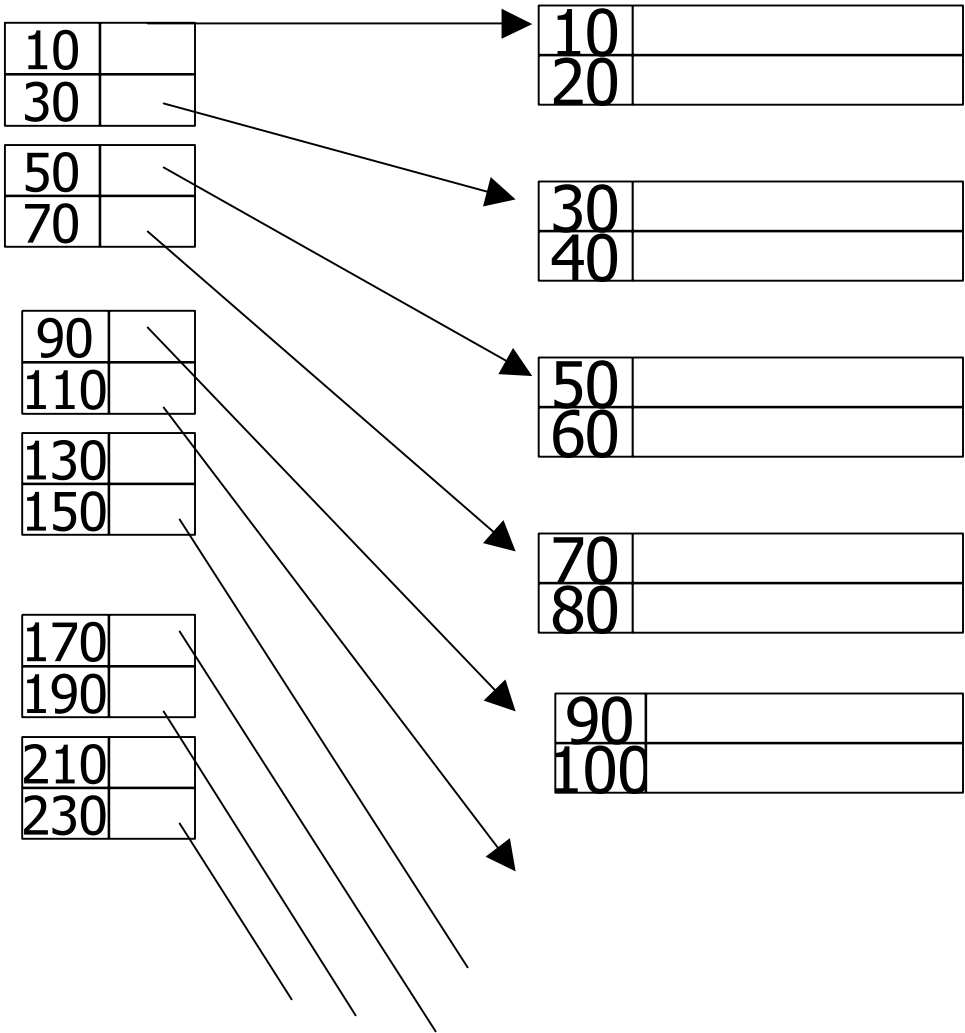
Dense Index

Sequential File



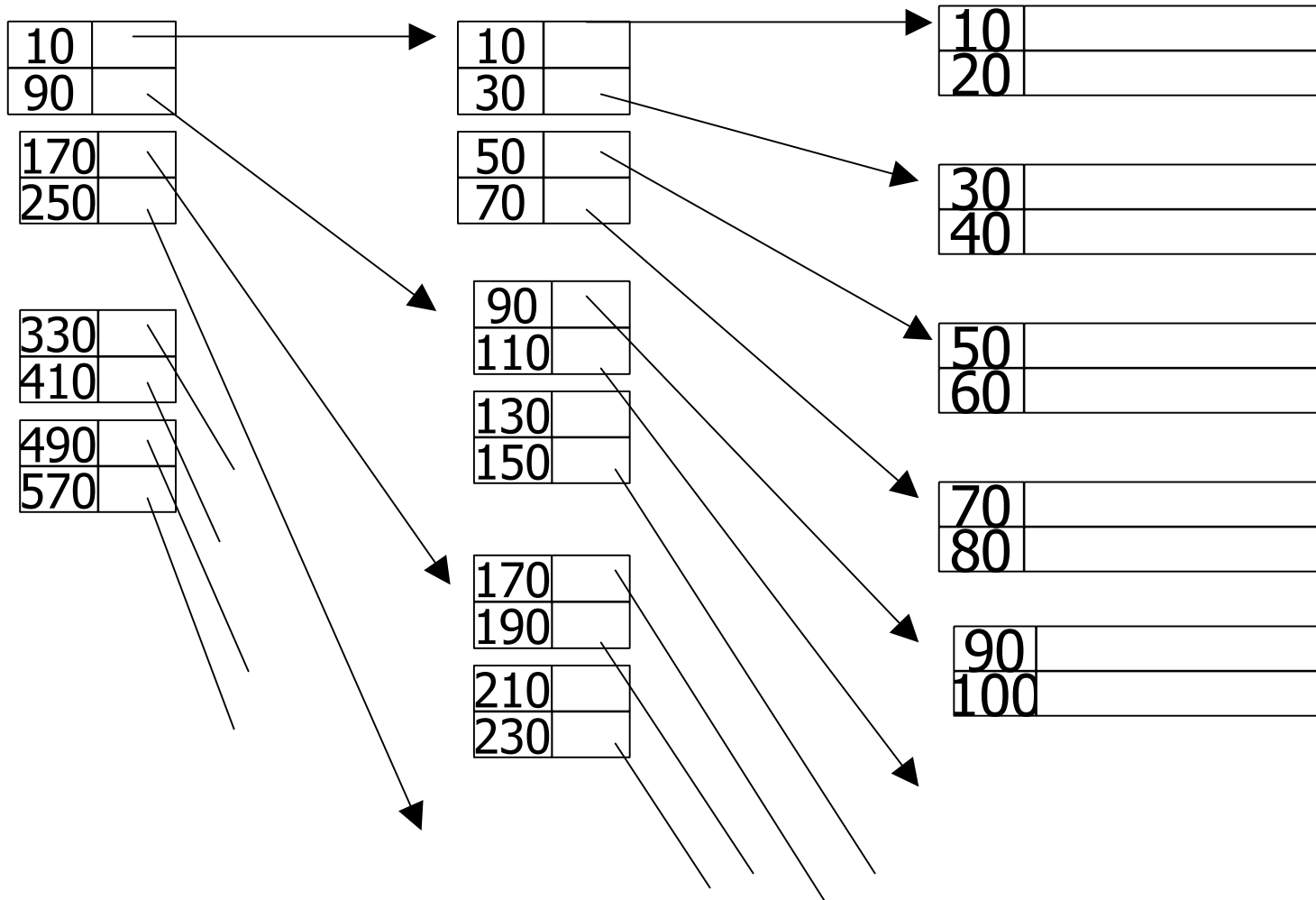
Sparse Index

Sequential File



Sparse 2nd level

Sequential File



Sparse vs. Dense Trade-off

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

(Later:

- sparse better for insertions
- dense needed for secondary indexes)

Summary of terms

- **Index on sequential file**
- **Search key** (can be \neq primary key)
- **Primary index** (on sequencing field)
 - secondary index works on other fields
- **Dense index** (all search key values in)
- **Sparse index** (one search key/ block)
- **Multi-level index** (index on index)

Next:

- Duplicate keys
- Deletion/Insertion
- Secondary indexes

Duplicate keys

10	
10	

10	
20	

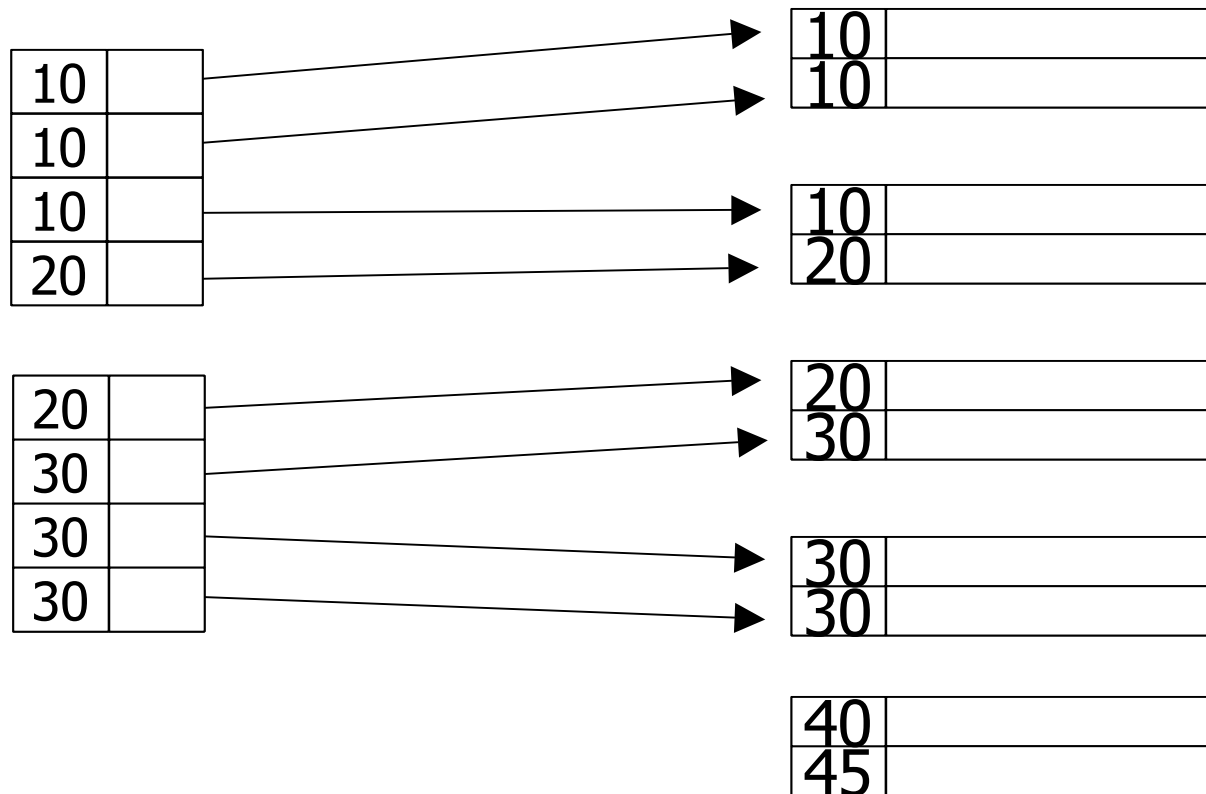
20	
30	

30	
30	

40	
45	

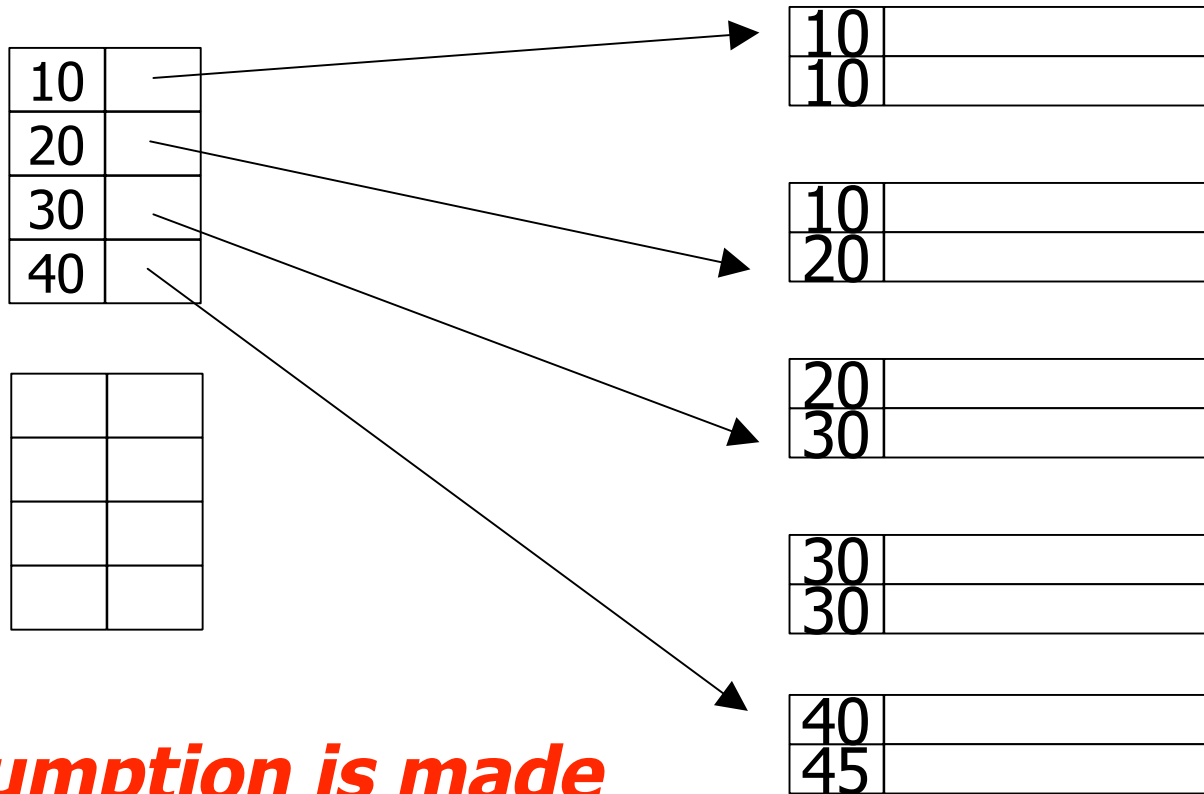
Duplicate keys

Dense index, one way to implement?



Duplicate keys

Dense index, better way?



What assumption is made here?

Duplicate keys

Sparse index, one way?

careful if looking
for 20 or 30!

10	—
10	—
20	—
30	—

10	
10	

10	
20	

20	
30	

30	
30	

40	
45	

Duplicate keys

Sparse index, another way?

– place first new key from block

should
this be
40?

10	
20	
30	
30	

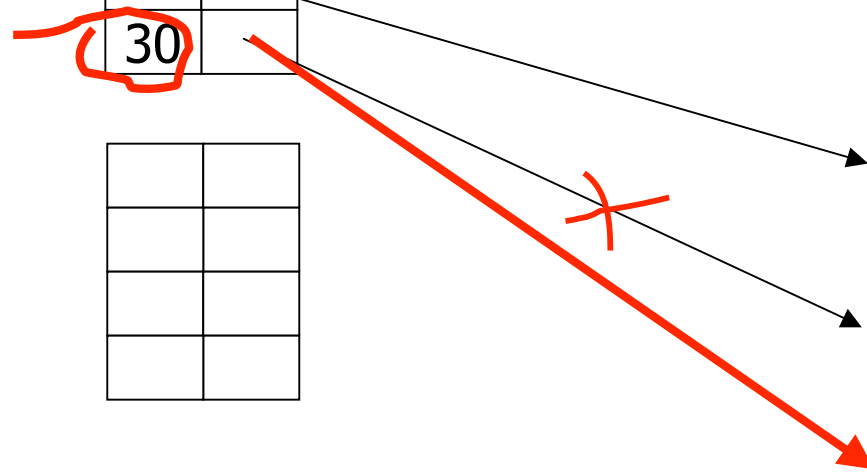
10	
10	

10	
20	

20	
30	

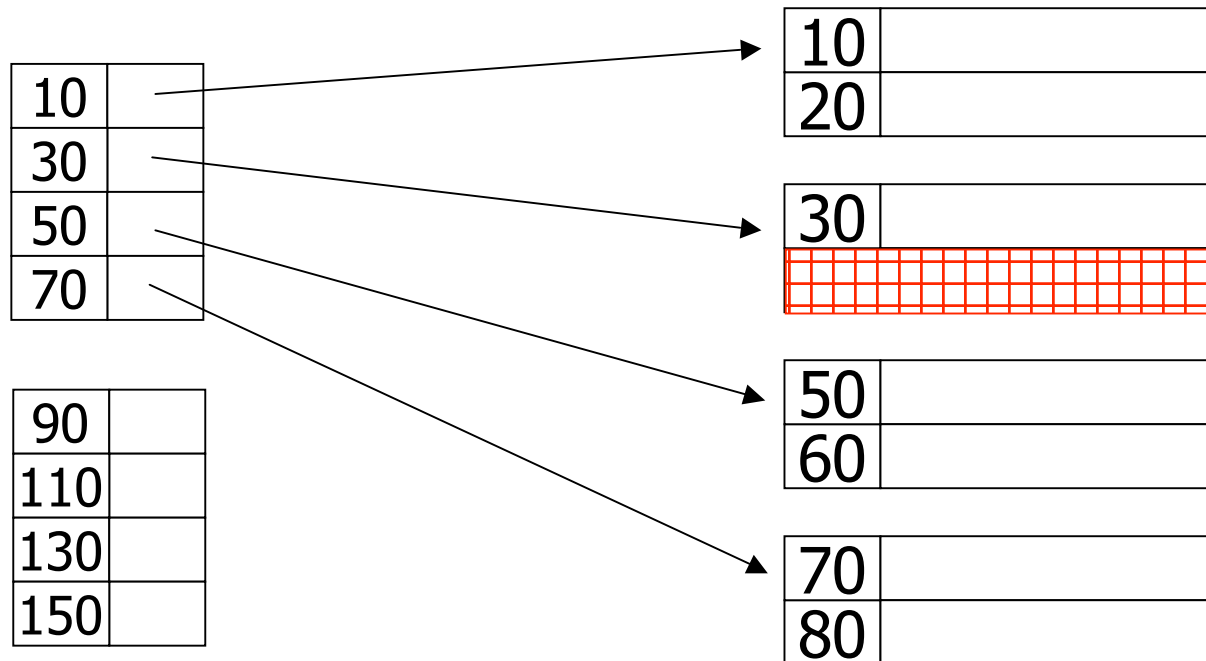
30	
30	

40	
45	



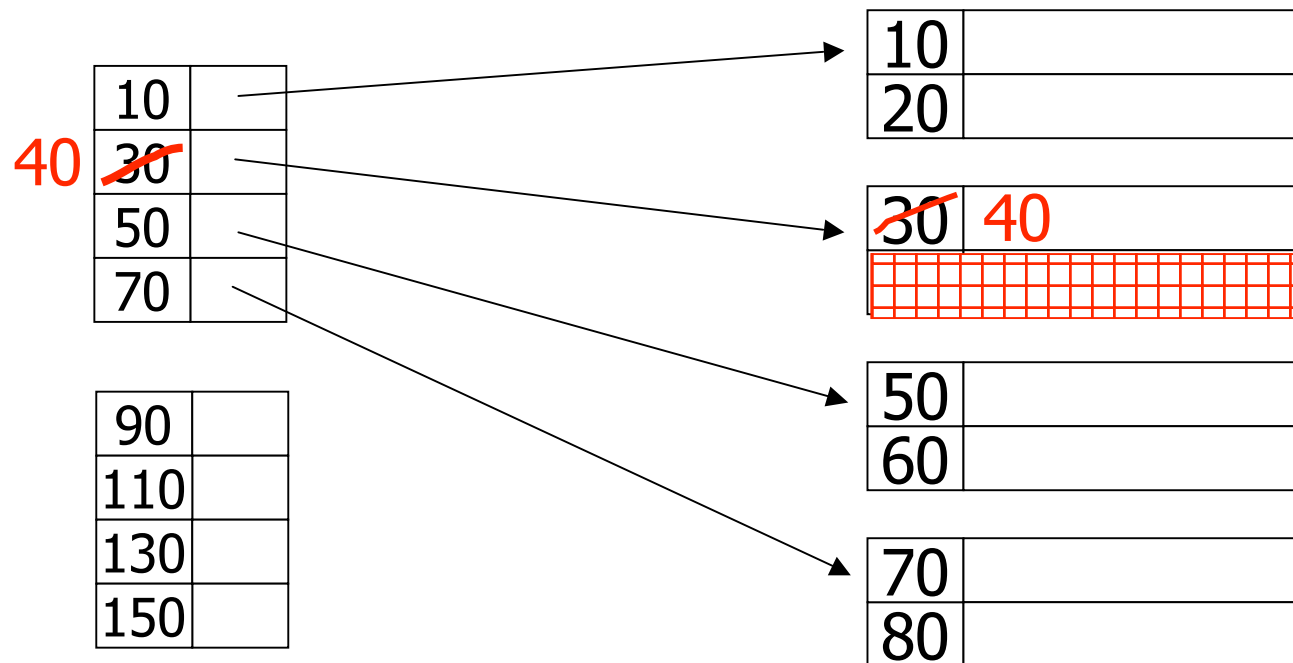
Deletion from sparse index

- delete record 40



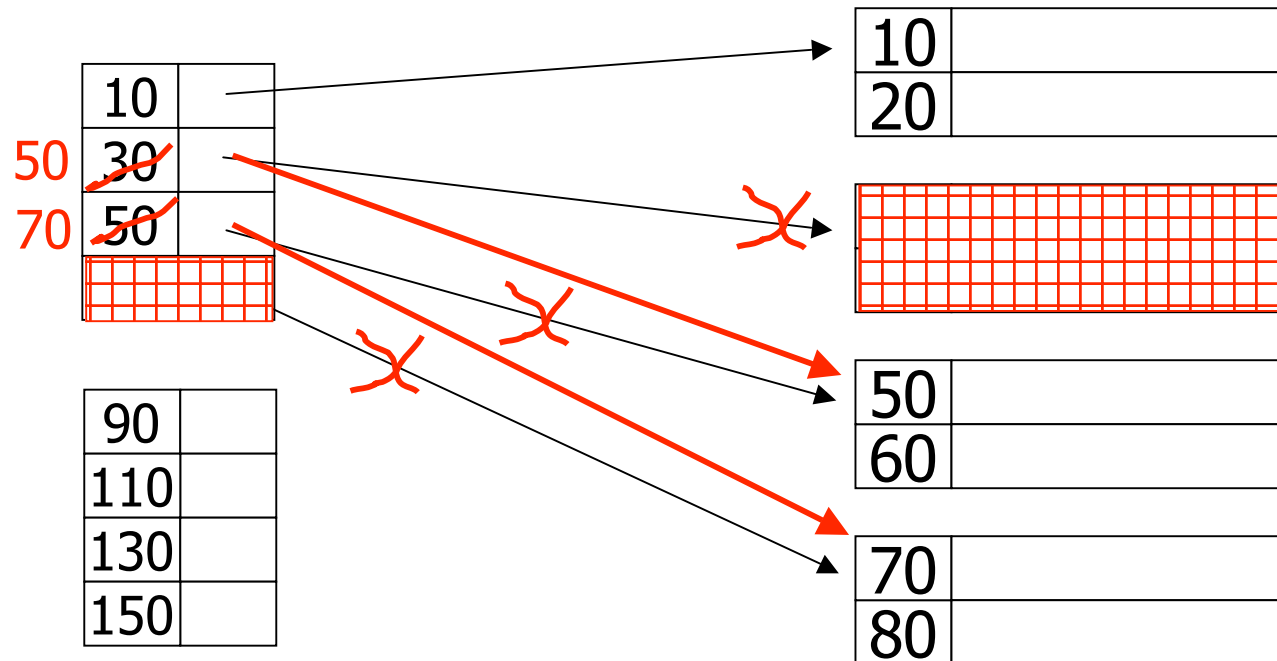
Deletion from sparse index

- delete record 30



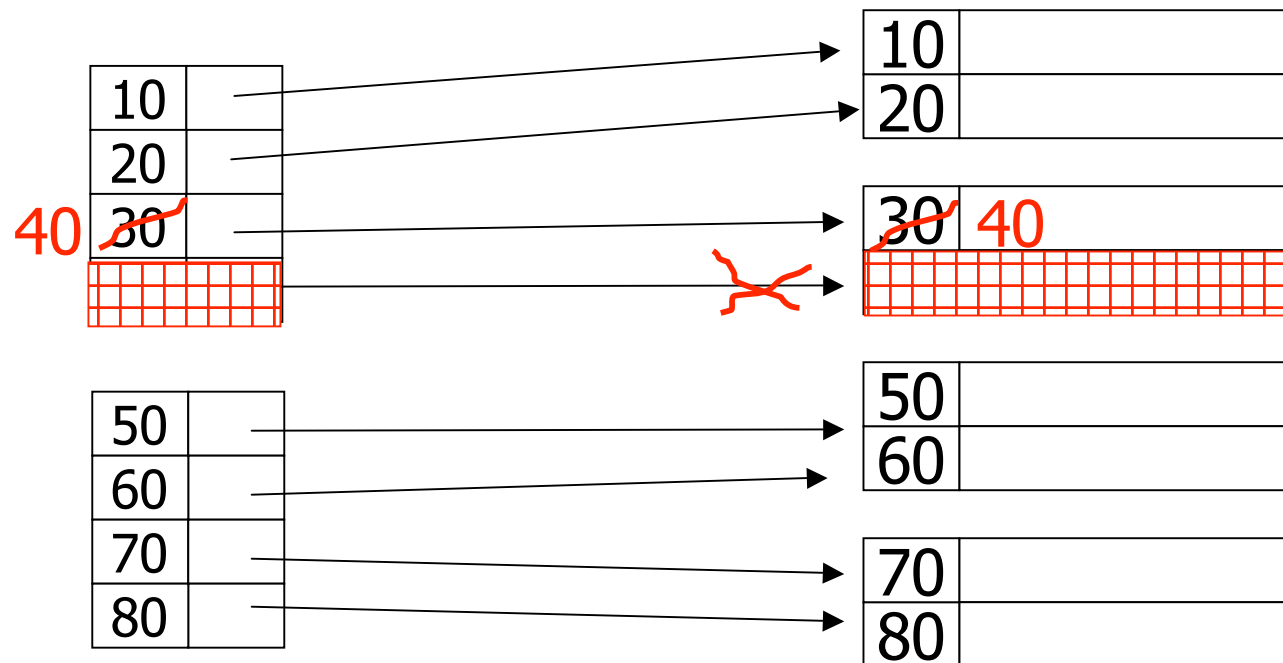
Deletion from sparse index

- delete records 30 & 40



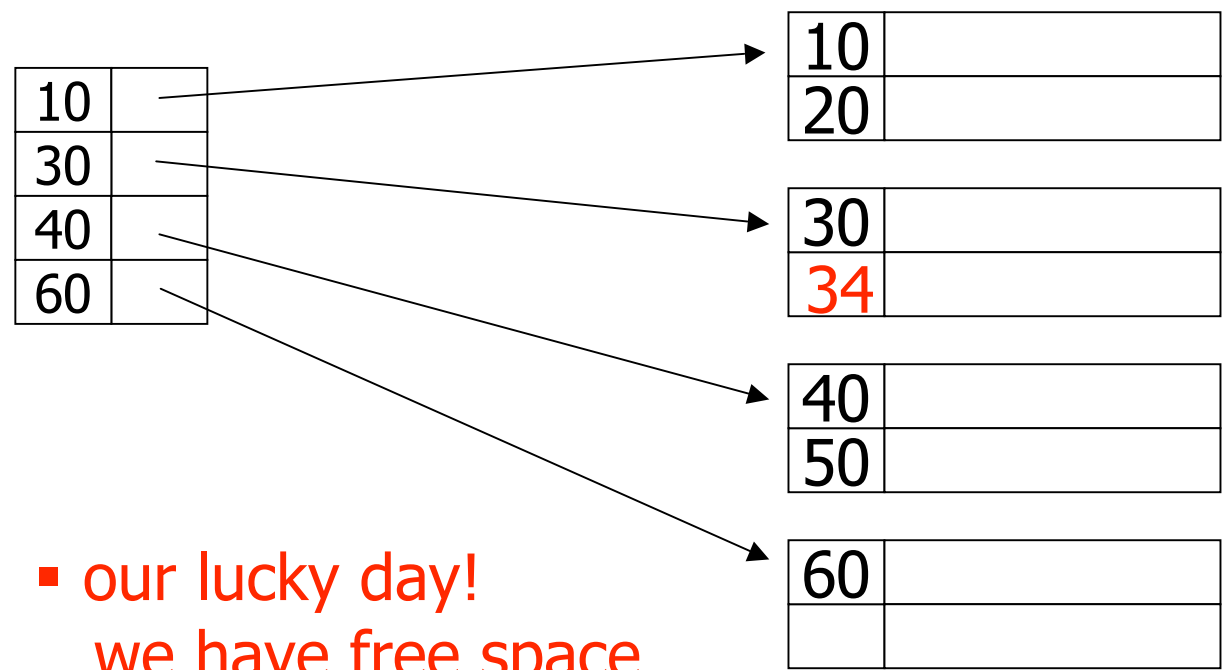
Deletion from dense index

- delete record 30



Insertion (sparse index case)

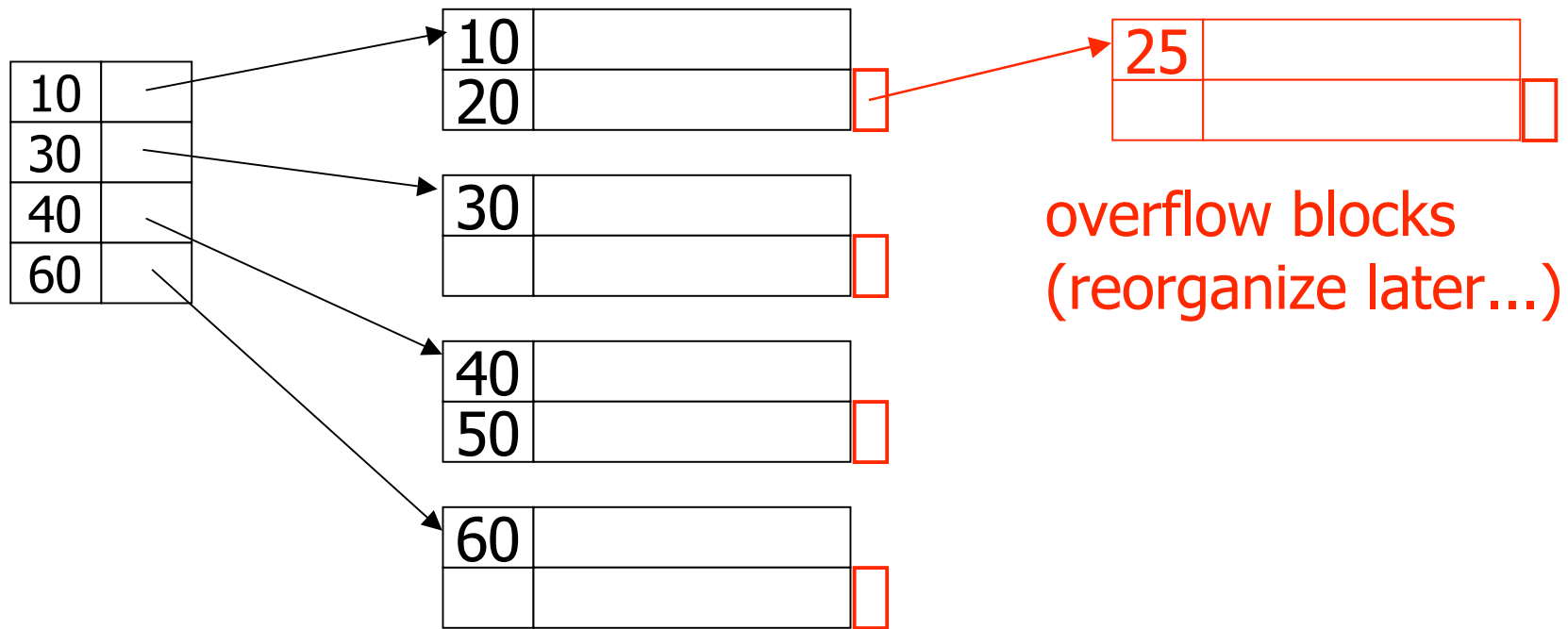
- insert record 34



- our lucky day!
we have free space
where we need it!

Insertion - using overflow blocks

- insert record 25

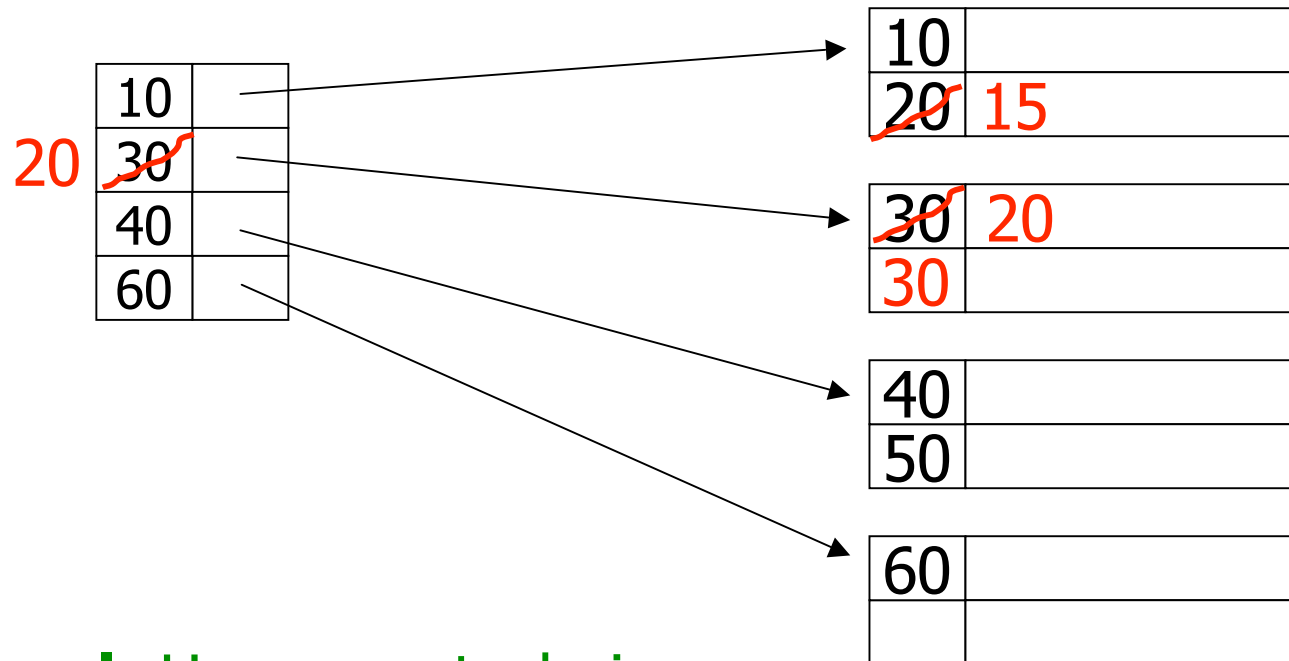


Problem:

Overflow blocks take longer to access

Insertion - immediate reorganization

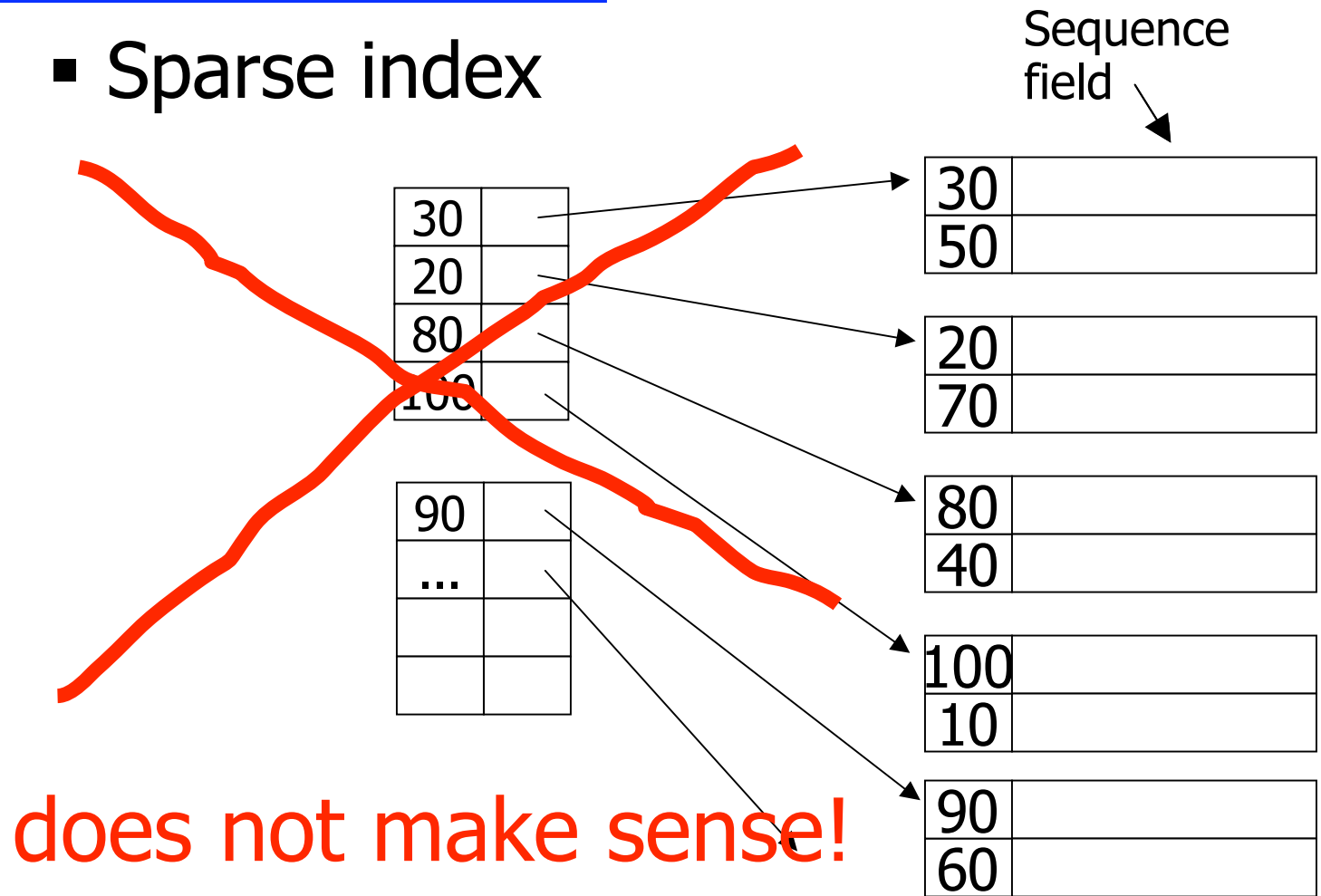
- insert record 15



- **In general:** Use same technique as for inserting in linked list.

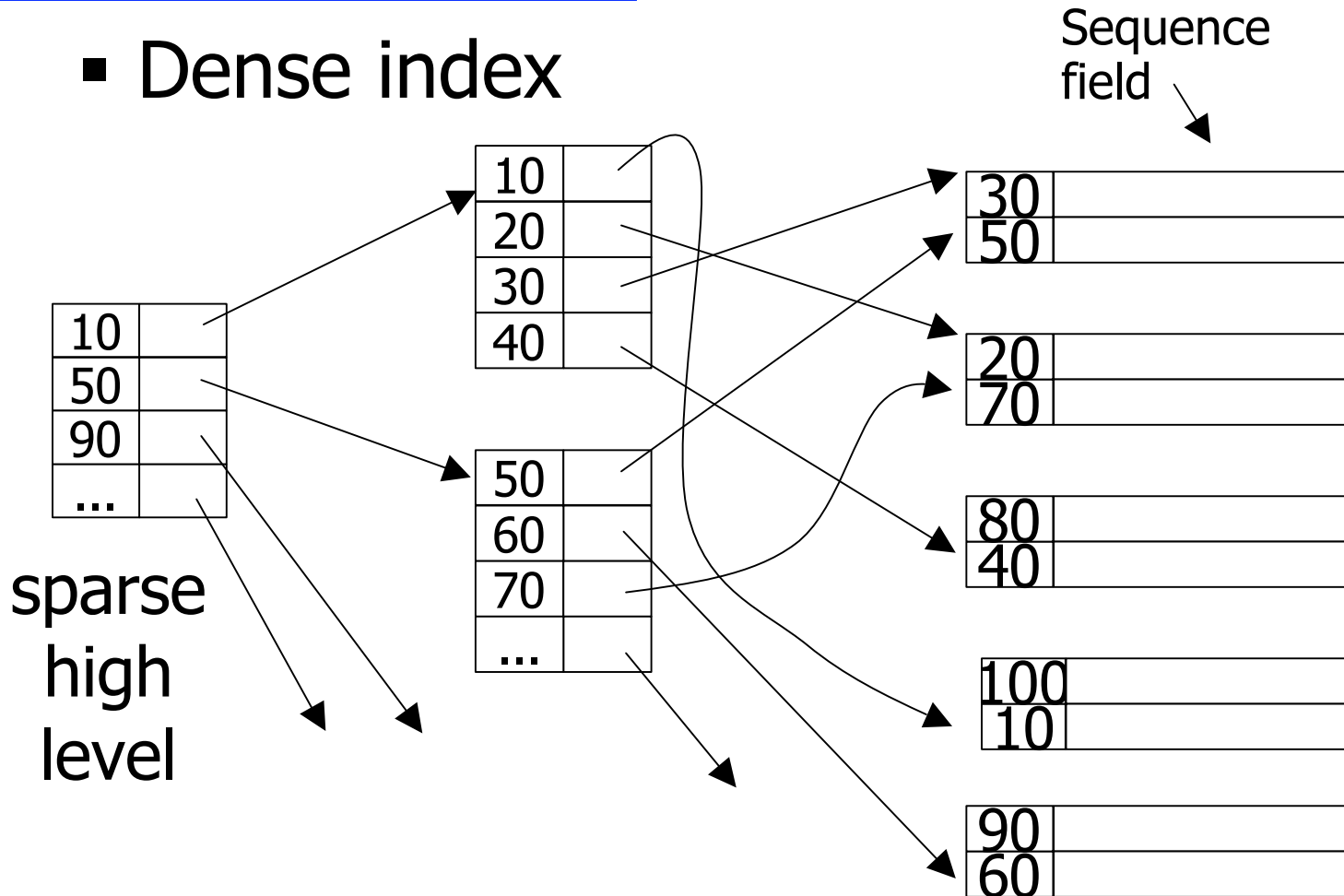
Secondary indexes

- Sparse index



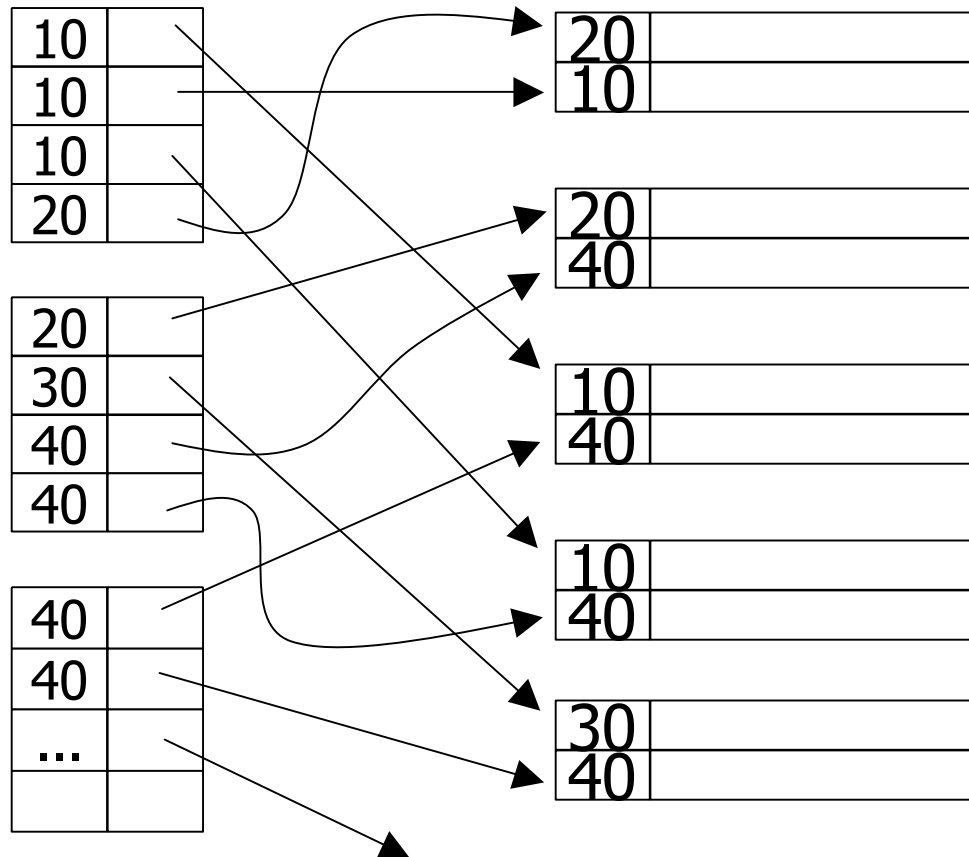
Secondary indexes

- Dense index



Duplicate values & secondary indexes

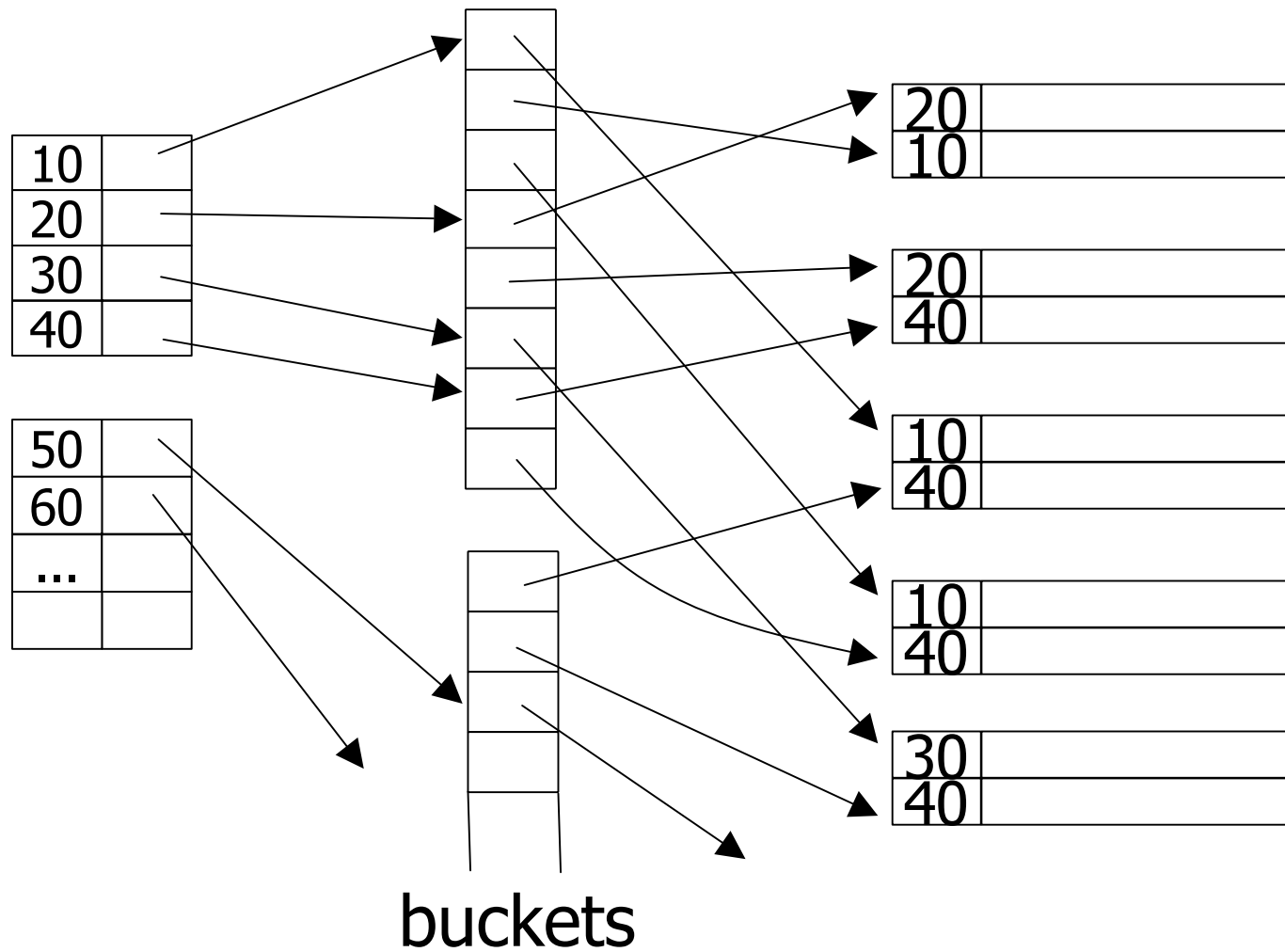
one option...



Problem:

Uses more space than necessary

Duplicate values & secondary indexes



Summary

- Indexes allow finding a particular attribute value in a few I/Os.
- Unresolved problems regarding insertions and deletions.
- **Next time:** Also obtaining efficient updates (using B-trees). Hash indexes - sometimes more efficient.