Index Structures

What is an Index?

An index is a data structure that allows us to directly locate units of data based on certain values

- Not just used for databases: also books can contain an index

Indexes for databases are used to find records that have a particular value for the indexed attribute (the “search key”)

An index has to be created before it can be used

- creation often initiated by the database designer
- cost of maintenance

Different categories exist

- primary / secondary indexes
- dense / sparse indexes

based on the key being indexed.
based on whether an index entry exists for every record or only for certain records.

Sequential Files

Records ordered by search key (may not be "key" in DB sense).

- facilitates queries on the search key

Blocks containing records therefore ordered.

- physically contiguous
- chained

On insert: put record in appropriate block if room.

- Good idea: initialize blocks to be less than full; reorganize periodically if file grows.

If no room in proper block:

- 1. Create new block; insert into proper order if possible (what if blocks are consecutive around a track for efficiency?).
- 2. If not possible, create overflow block, linked from original block.

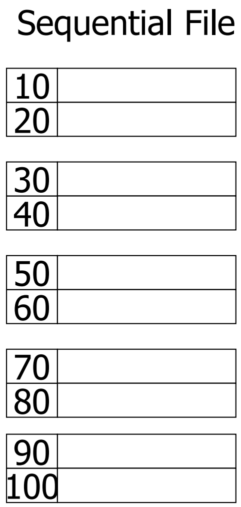
Indexes

Dense Indexes: Pointer to every record of file, ordered by search key.

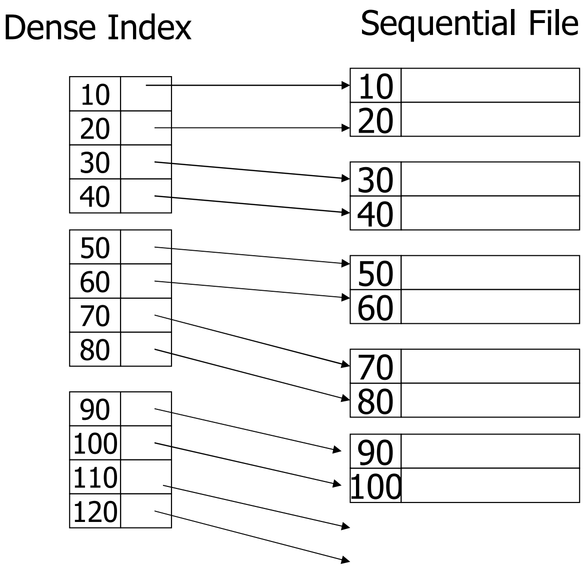
- Can make sense because records may be much bigger than key-pointer pairs.
  - If index requires fewer blocks faster search through index than data file
  - Index might fit in memory, even if data file does not
- Test existence of record without going to data file.

Sparse Indexes: Keypointer pairs for only a subset of records, typically first in each block.

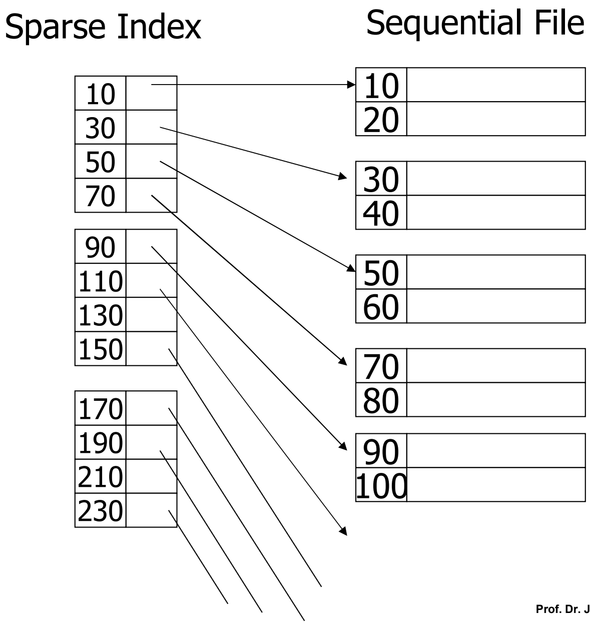
Example: Sequential File



Example: Dense Index



Example: Sparse Index



Sparse vs. Dense Index

Sparse: Less index space per record can keep more of index in memory

Dense: Can tell if any record exists without accessing file

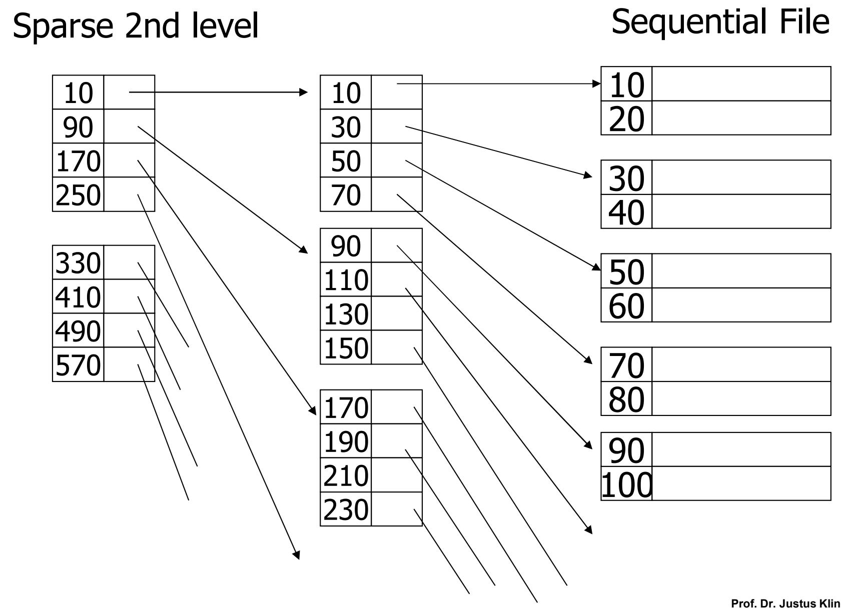
Multiple Levels of Index

A sparse index on a (sparse or dense) index is an option.

Good chance that 2nd or higher level indexes can be housed in main memory, so no additional disk I/O's.

Dense higher level indexes make no sense;

Example: Second Level Index



DB Modifications

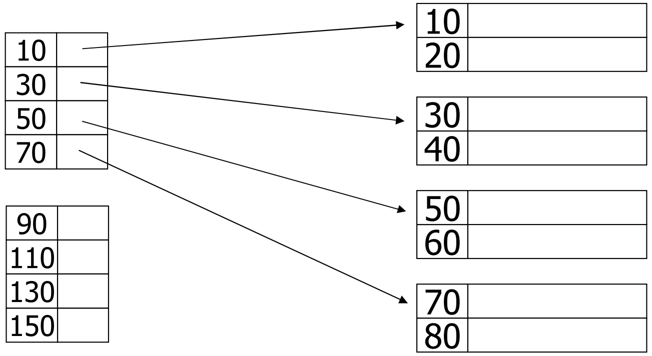
When we insert or delete on the data file, here are the primitive actions we might take:

1. Create or destroy an empty block in the sequence of blocks belonging to the sequential file.
2. Create or destroy an overflow block.
3. Insert a record into a block that has room.
4. Delete a record.
5. Slide a record to an adjacent block.

Effect of Primitive Actions on Index File

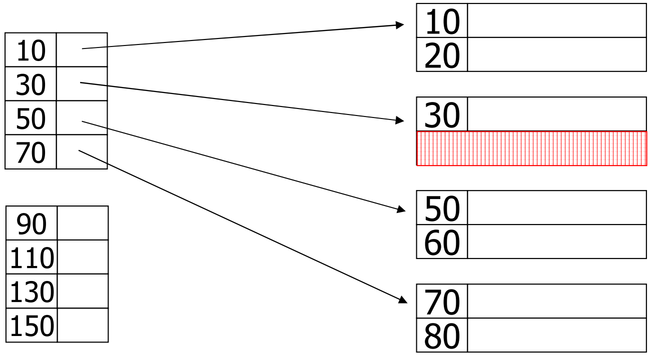
Action	Dense	Sparse
Create/destroy empty overflow block	none	none
Create empty seq. block	none	insert
Destroy empty seq. block	none	delete
Insert record	insert	update(?)
Delete record	delete	update(?)
Slide record	update	update(?)

Deletion from sparse index



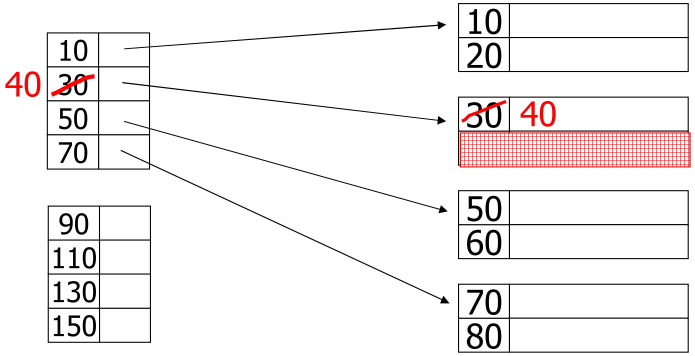
Deletion from sparse index

– delete record 40



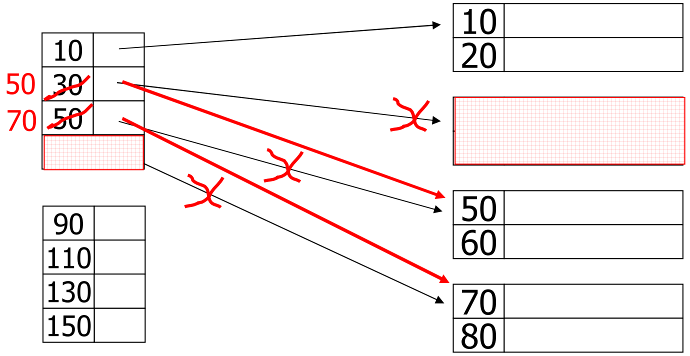
Deletion from sparse index

– delete record 30

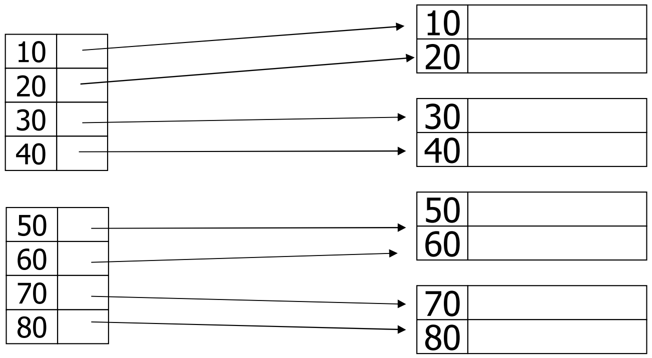


Deletion from sparse index

– delete records 30 & 40

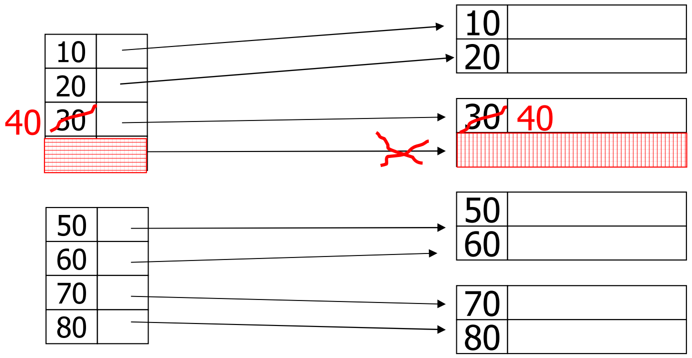


Deletion from dense index

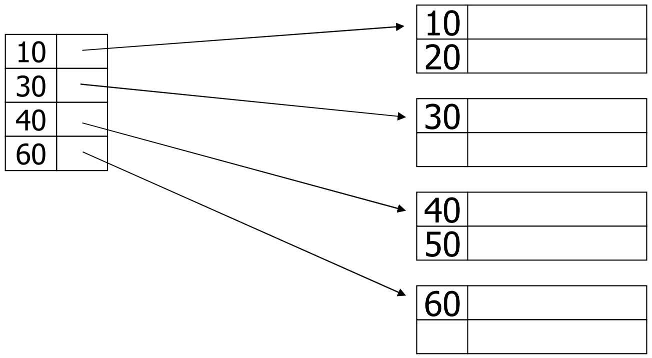


Deletion from dense index

– delete record 30

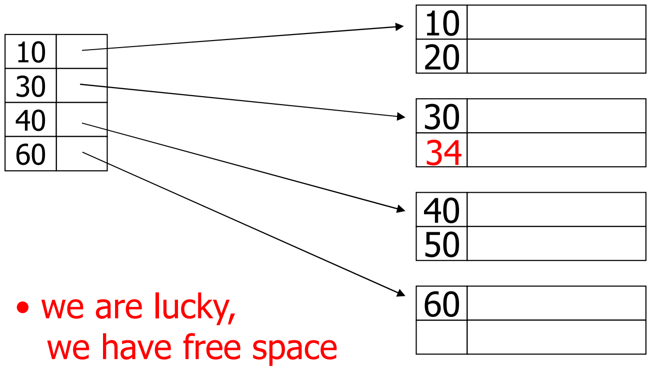


Insertion, sparse index case



Insertion, sparse index case

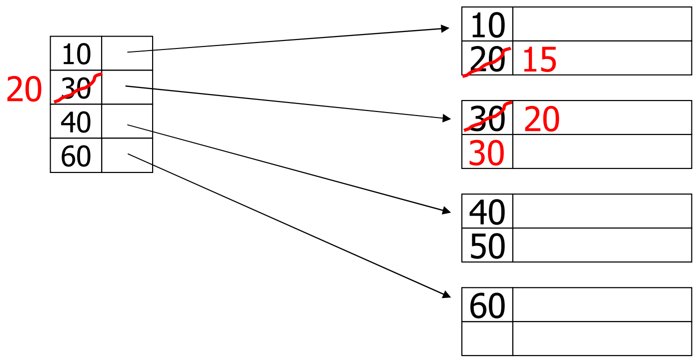
– insert record 34



- we are lucky, we have free space where we need it!

Insertion, sparse index case

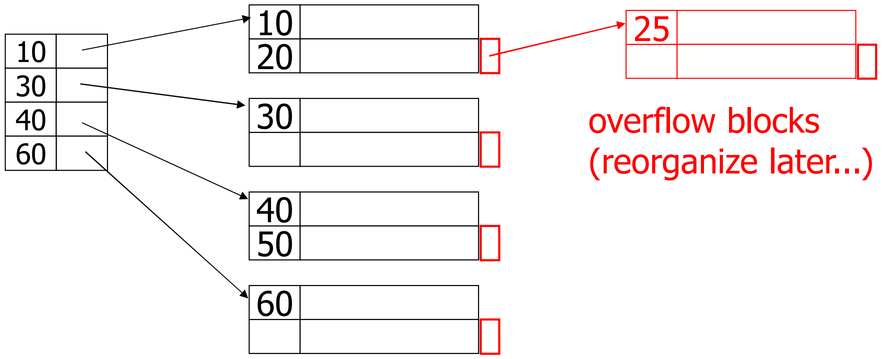
– insert record 15



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

Insertion, sparse index case

– insert record 25



overflow blocks
(reorganize later...)

Insertion, dense index case

- Similar
- Often more expensive . . .

Dense indexes trade insertion cost for faster search. The extra cost is due to:

Maintaining sorted order

Handling larger index structures

Performing additional I/O when splits happen.

Secondary Indexes

A primary index is an index on a sorted file.

- More general: any index that "controls" the placement of records to be primary, e.g., hash table.

Secondary index = index that does not control placement, surely not on a file sorted by its search key.

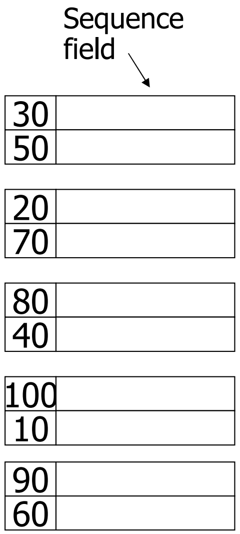
- Sparse, secondary index makes no sense.
- Usually, search key is not a "key"

Multiple Levels:

- Lowest level is dense
- Other levels are sparse

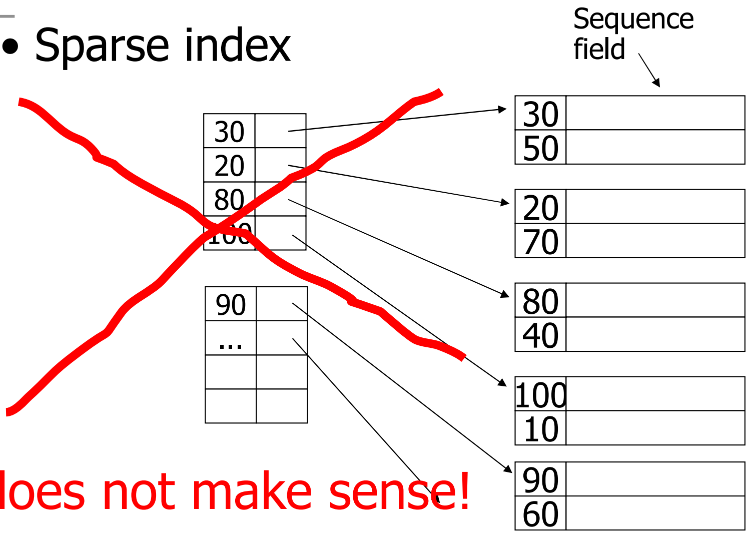
A secondary index must be dense because:
Data is not ordered by the secondary key.
You need a pointer for every record to locate them accurately.
If you keep only one entry per block, you can't guarantee finding all rec
with a given secondary key.
You would miss records because there is no contiguous structure.

Secondary Indexes



Secondary Indexes

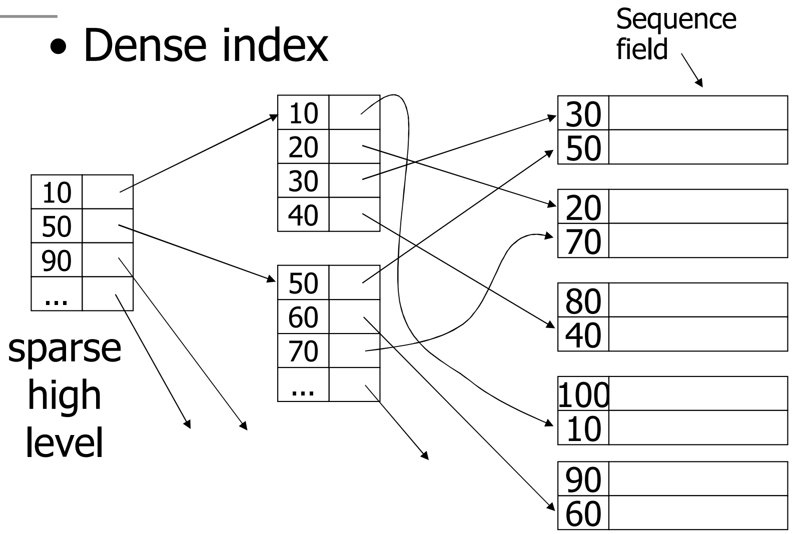
- Sparse index



does not make sense!

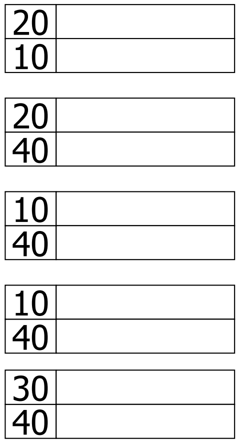
Secondary Indexes

- Dense index



sparse
high
level

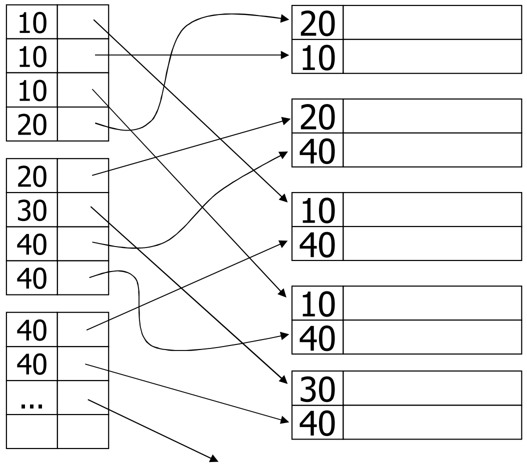
Duplicate values & secondary indexes



Duplicate values & secondary indexes

one option...

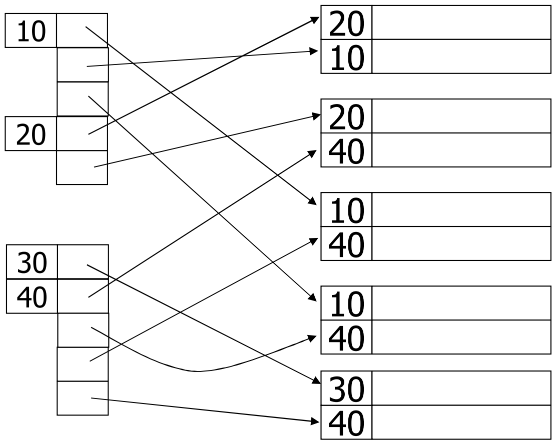
Problem:
excess overhead!
- disk space
- search time



Duplicate values & secondary indexes

another option...

Problem:
variable size
records in
index!

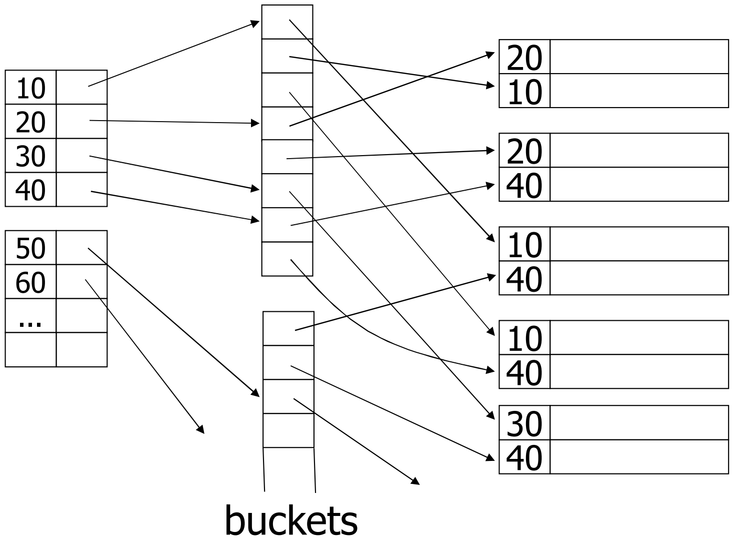


Indirect Buckets

To avoid repeating keys in index, use a level of indirection, called buckets.

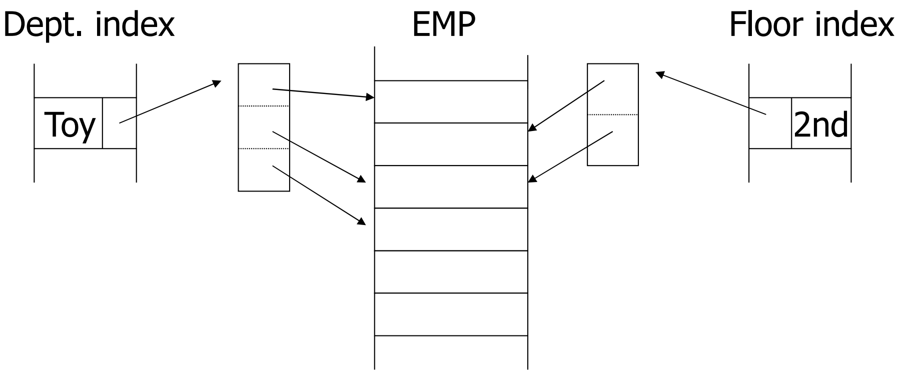
- Additional advantage: allows intersection of sets of records without looking at records themselves.

Duplicate values & secondary indexes



Indirect Buckets

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



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

Assessment of Conventional Indexes

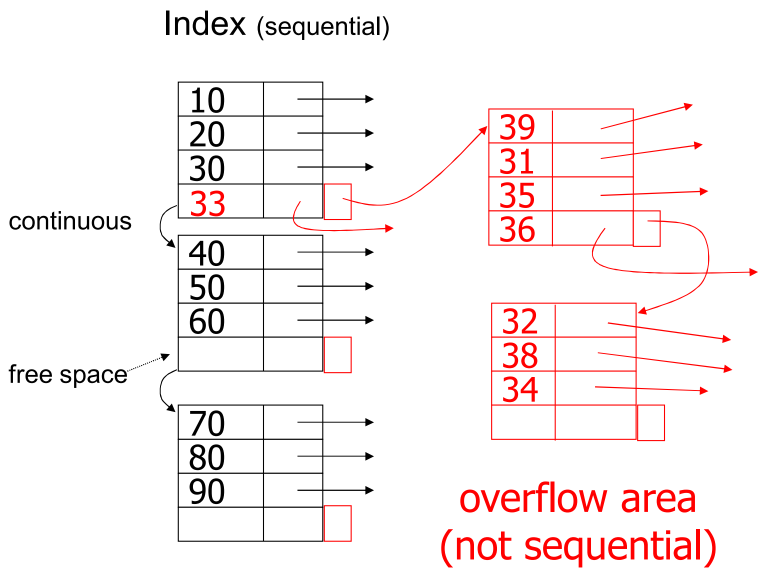
<u>Advantage:</u>

- Simple
- Index is sequential file good for scans

<u>Disadvantage:</u>

- Inserts expensive, and/or
- Lose sequentiality & balance

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)