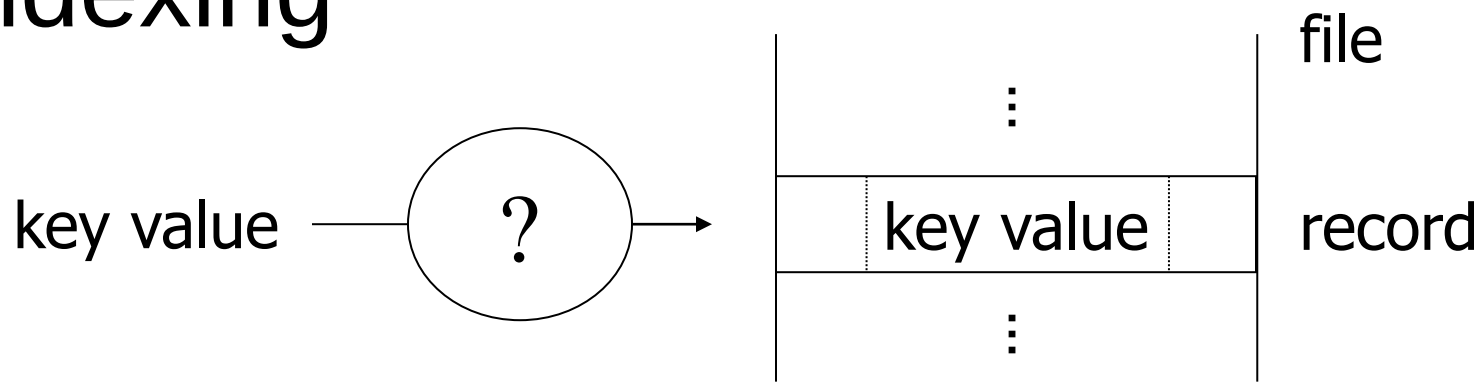
PA152: Efficient Use of DB

4. Indexing

Vlastislav Dohnal

Indexing

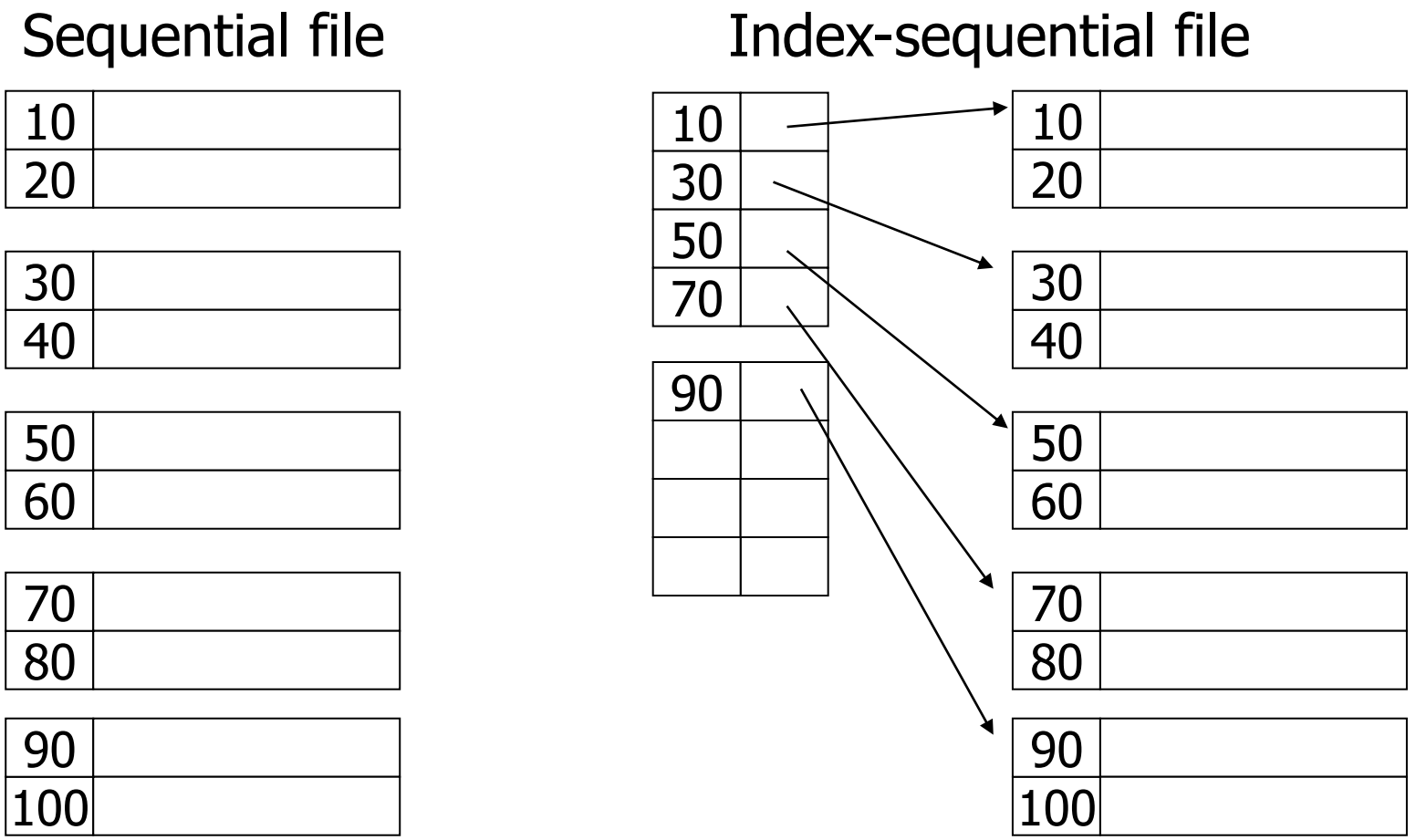


- Reason: faster access to records
  - than sequential (table) scan
- Variants:
  - Conventional indexes
  - B-tree
  - Hashing

Terminology

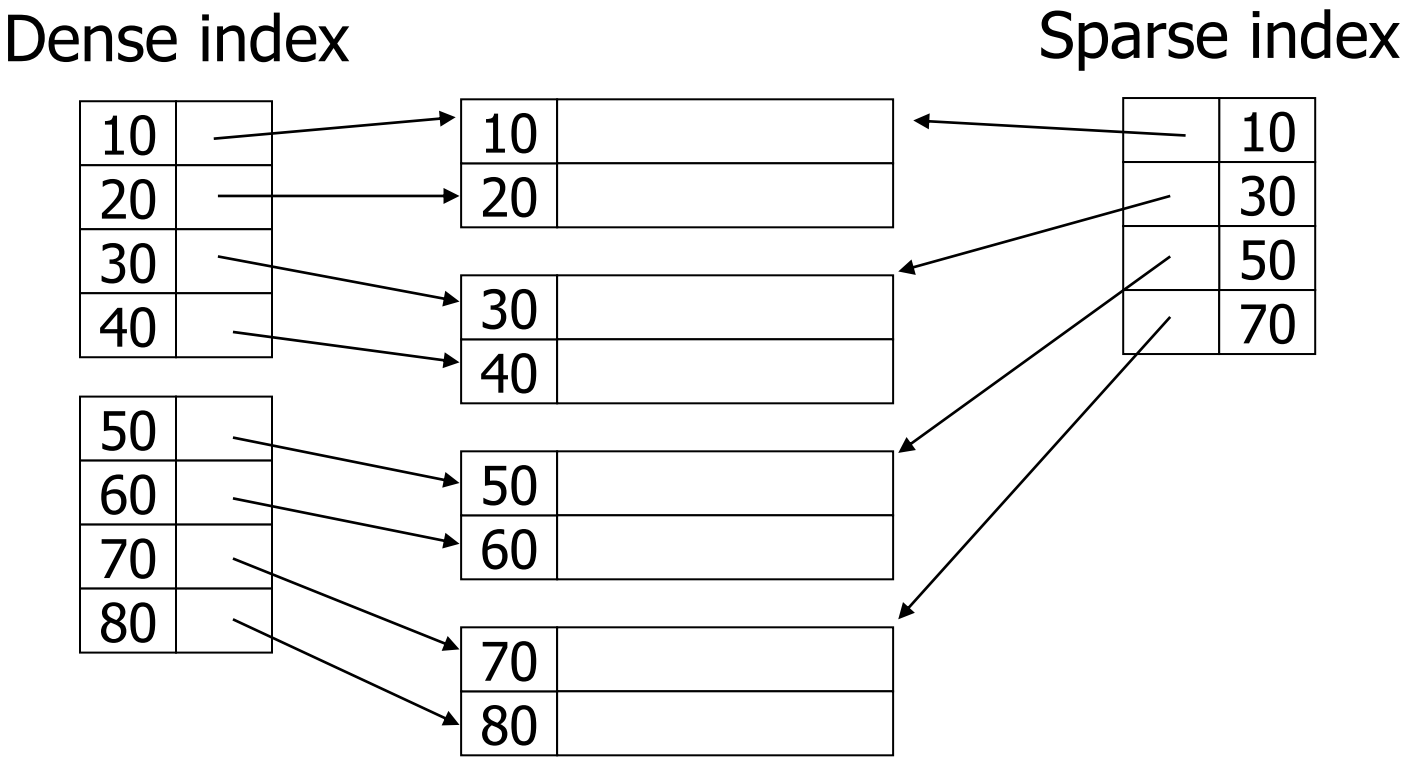
- Sequential file
  - Index-sequential file
- Index
  - Primary index
  - Secondary index
  - Dense index
  - Sparse index
  - Multilevel index
- Search key
  - Primary key

File


Sequential file
10
20
30
40
50
60
70
80
90
100
Index-sequential file
10
30
50
70
90
10
20
30
40
50
60
70
80
90
100

Index

- Collection of items:
  - <key value, pointer to record/block>



Problem of Duplicate Keys

- Index type
  - dense index?
  - sparse index?

File

10	
10	

10	
20	

20	
30	

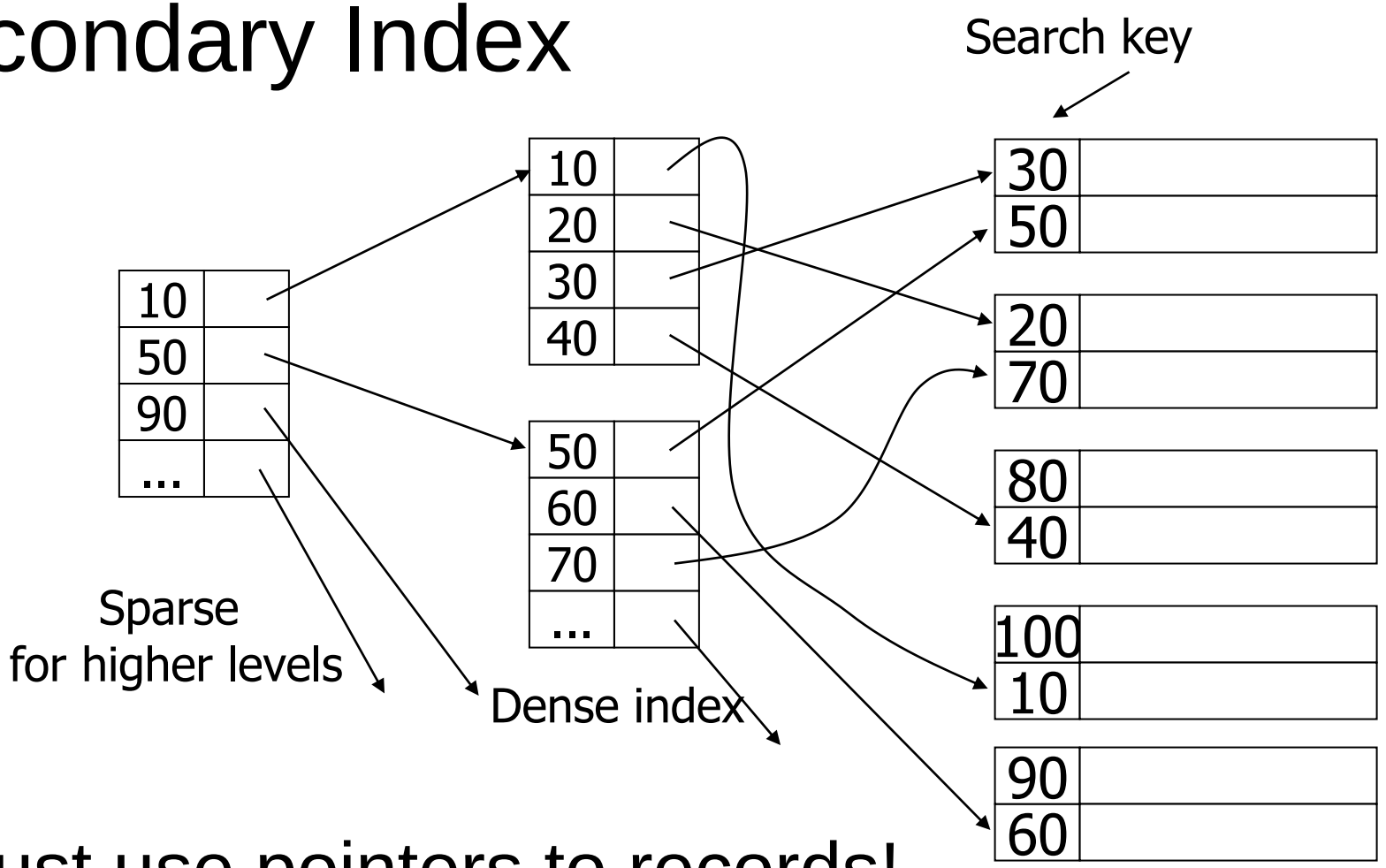
30	
30	

40	
45	

Secondary Index

- File ordered by another key
  - i.e., index created for different key than the primary file
  - Or the file is not ordered at all
- Which type:
  - Dense or sparse?

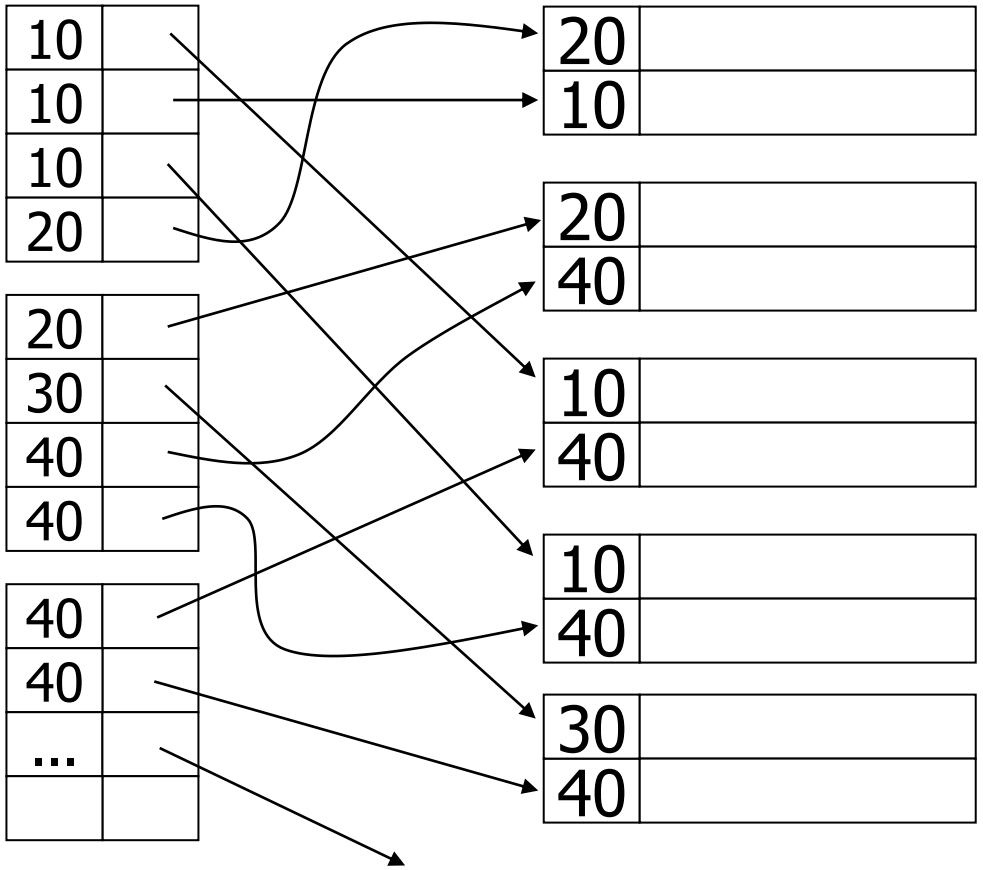
Secondary Index



- Must use pointers to records!

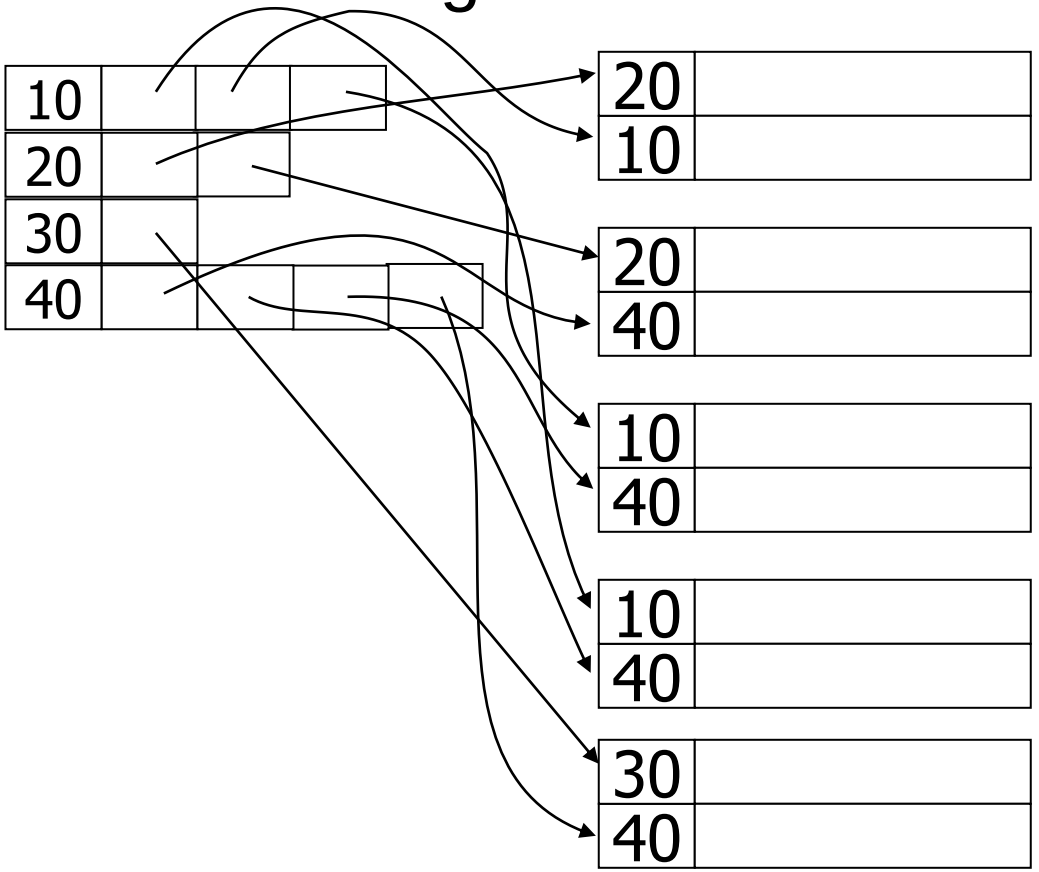
Secondary Index: Duplicate Keys

- Replicated in index
  - Increases
    - space requirements
    - access time



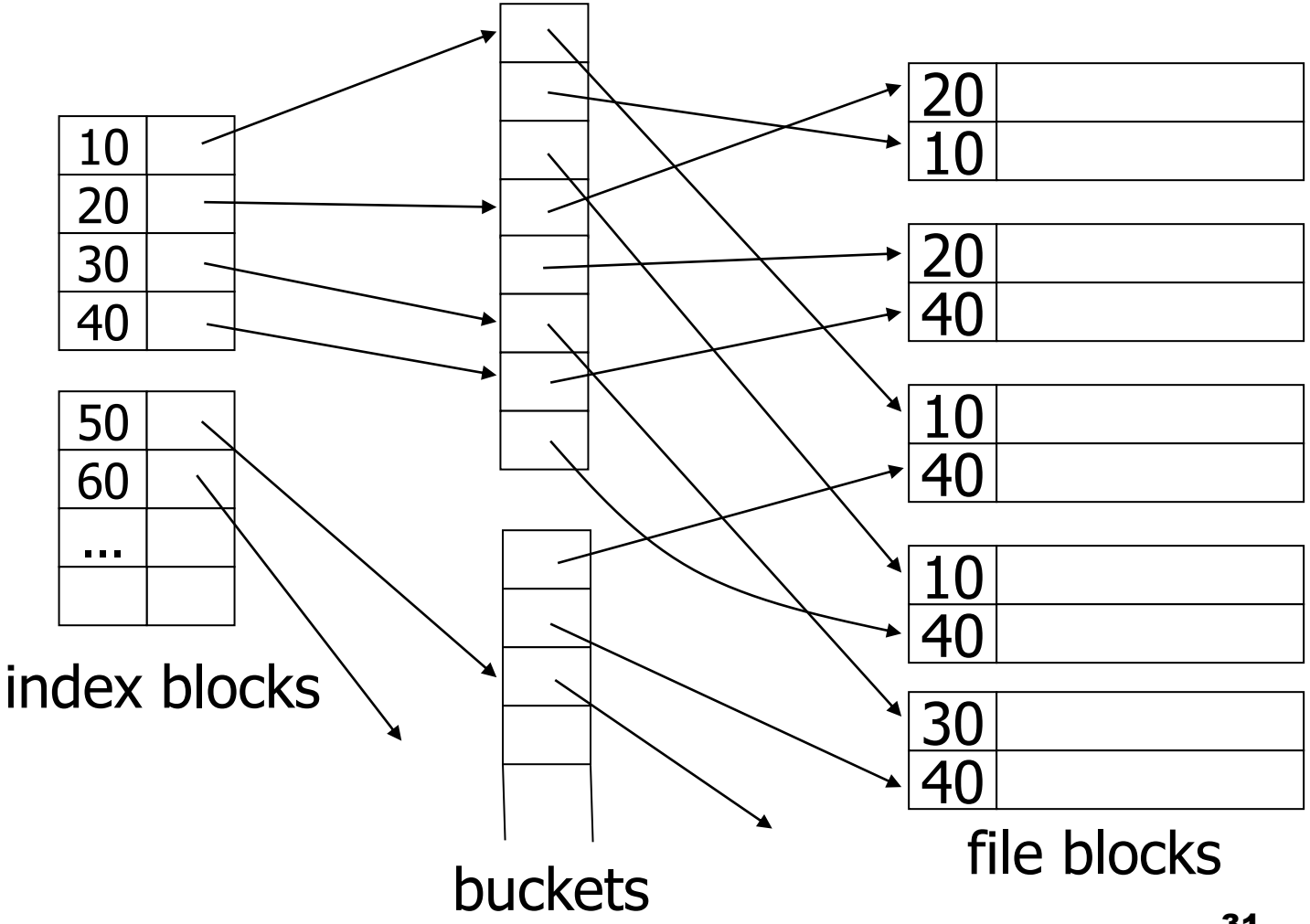
Secondary Index: Duplicate Keys

- Index item contains list of pointers
  - But the item is of variable length



Secondary Index: Duplicate Keys

- Shift the variable-length list to “buckets”



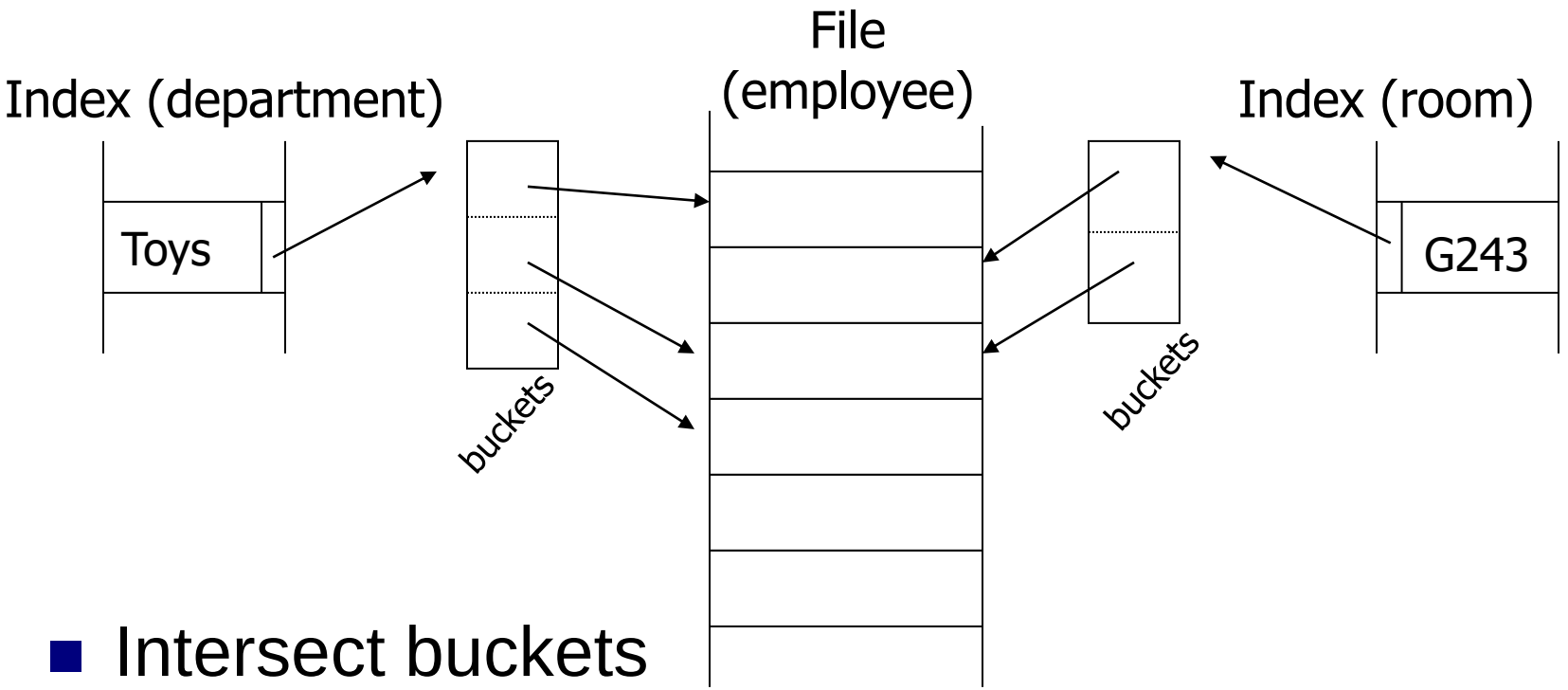
Secondary Index: Buckets with Pointers

- Advantage: a list of records for querying
  - Evaluate more selection constrains without accessing records

- Example:
  - Relation
    - employee(name, department, room)
  - Indexes:
    - name – primary index
    - department – secondary index
    - room – secondary index

Secondary Index: Duplicate Keys

- Query employees of Toys dept. in room G243



- Intersect buckets
  - To get pointers to matching employee records
  - Also used in *text information retrieval*

Example: Text Information Retrieval

- "Full-text" index for documents
- Split documents into words

Document 1

Caesar and Brutus are ambitious.

1. Split to words
2. Stemming
3. Ignore words in stop-list

Word List for Doc1

- *Caesar*
- *Brutus*
- *ambitious*

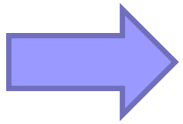
- Build an inverted file
  - over all documents
  - i.e., a file of records <word; [docId, docId2, …]>

Example: Text Information Retrieval

- Inverted file

Relational view

Term	docID
ambitious	1
brutus	1
brutus	3
capitol	2
caesar	1
caesar	2



Inverted file

Term	Posting list of docIDs
ambitious	1
brutus	1, 3
capitol	1
caesar	1, 2

- Retrieve docs containing *Brutus & Caesar*
  - Read *posting lists* for Brutus and Caesar
  - Intersect them

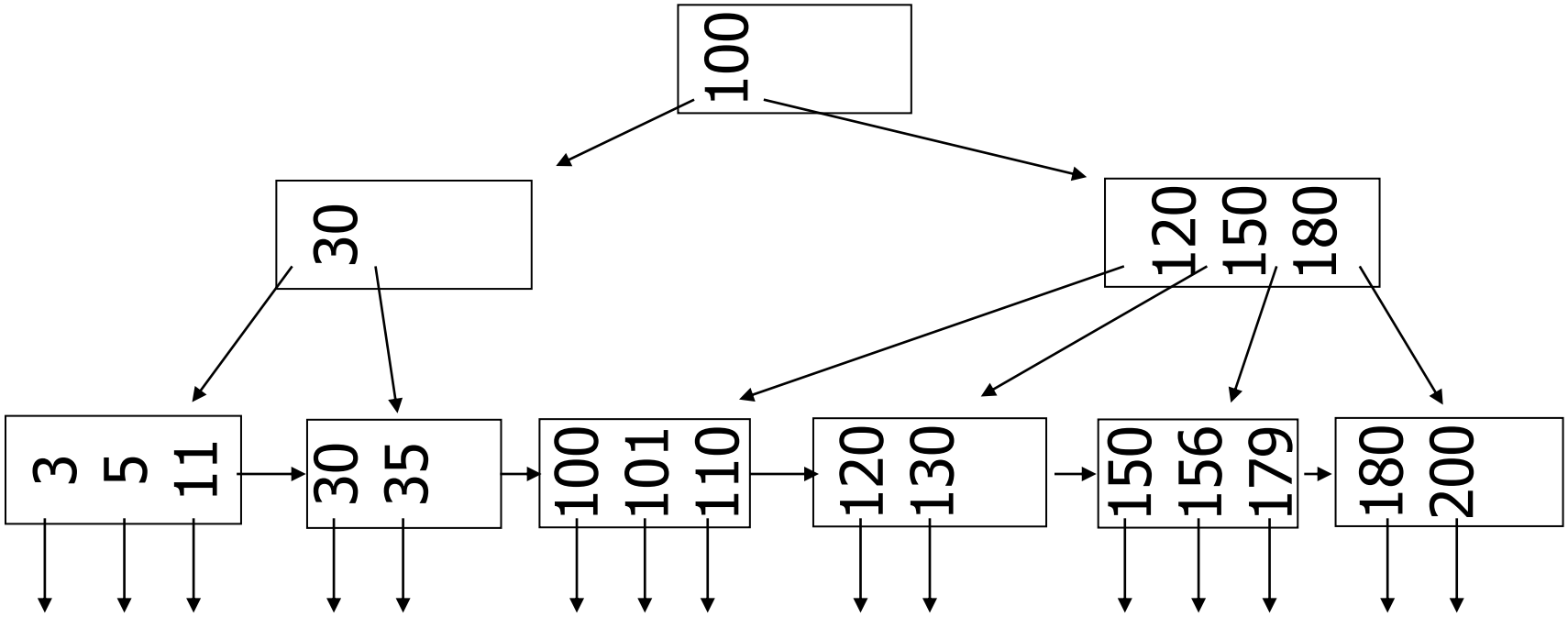
B-trees

- Another index type
  - Sequential order not necessary
  - Balanced – max I/Os guarantee
- More variants
  - B-tree, B^+-tree, B^*-tree, …
    - Typically, by saying "*B-tree*" we mean "*B^+-tree*"!
- Origin
  - Rudolf Bayer and Ed McCreight invented the B-tree while working at Boeing Research Labs in 1971 (Bayer & McCreight 1972)
    - They did not explain what, if anything, the B stands for.
    - Douglas Comer explains:
      - The origin of "B-tree" has never been explained by the authors. As we shall see, "balanced," "broad," or "bushy" might apply. Others suggest that the "B" stands for Boeing. Because of his contributions, however, it seems appropriate to think of B-trees as "Bayer"-trees.

* Source: Wikipedia

B$^+$-tree

- Example *n=4*



B^+-tree

- B^+-tree as file
  - Leaves store the records themselves.
- Duplicate keys
  - Pointers in leaves = pointers to buckets
    - i.e., blocks with a list of record pointers with the same key value
- Variable-length key values (e.g., strings)
  - Store completely $\rightarrow$ low arity, varying arity, …
  - Use prefixes (prefix compression)

Lecture's Takeaways

- Principle of indexing
  - Use of record pointers and their utilization
  - Handling duplicate keys
- Efficiency of B+ trees
  - also, with respect to query types
- Revision of terminology
  - Dense / sparse index
  - Primary / secondary index
    - Clustered / non-clustered index
  - Covering index