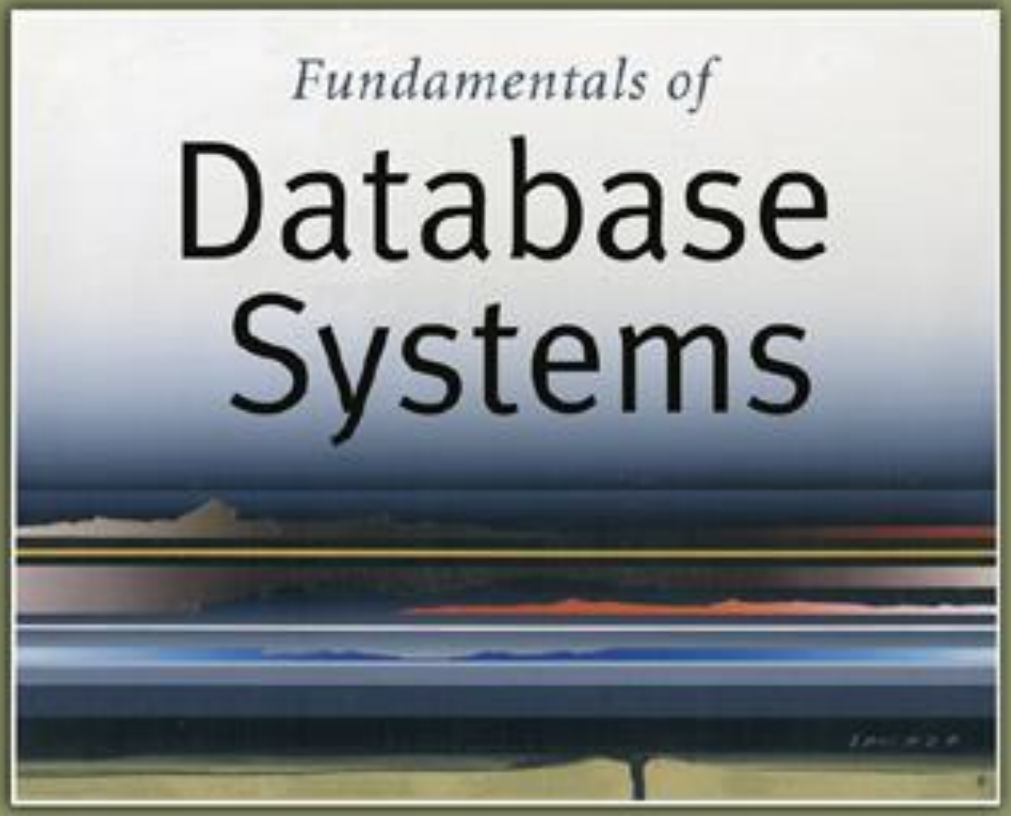
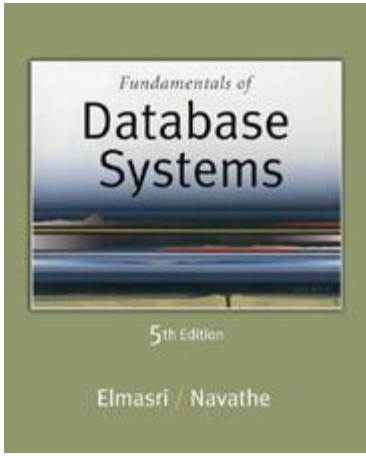


5th Edition

Elmasri / Navathe

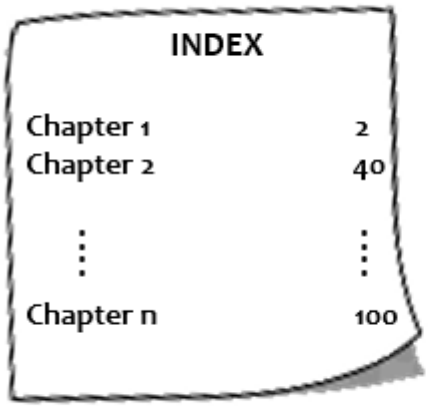
Indexing Structures



Chapter Outline

- Types of Single-level Ordered Indexes
  - Primary Indexes
  - Clustering Indexes
  - Secondary Indexes
- Multilevel Indexes

Indexing



Index page of a book

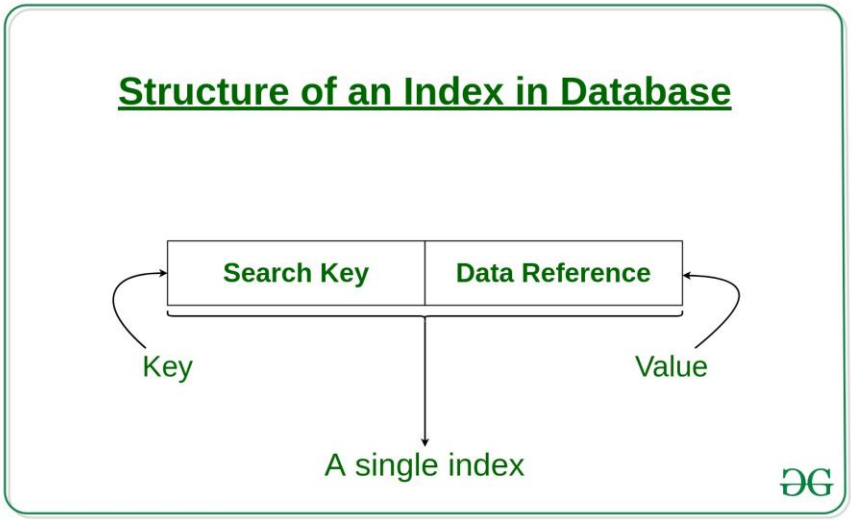
Analogy

Search Key	Block pointer
1	Address of block 1
10	Address of block 2
21	Address of block 3

Index file of a database file

Indexing

- Indexing is a way to optimize the performance of a database by **minimizing the number of disk accesses** required when a query is processed.
- It is **a data structure technique** which is used to quickly locate and access the data in a database.



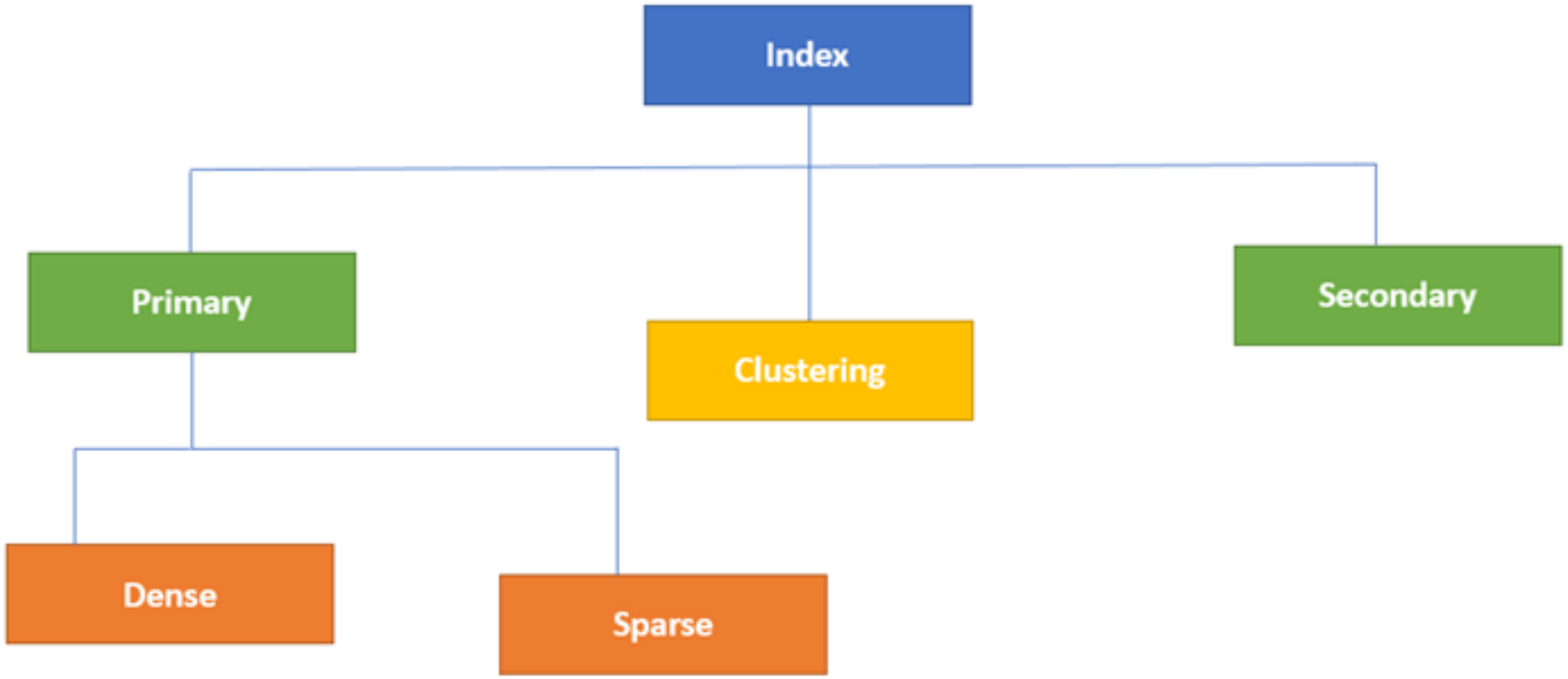
Indexes as Access Paths

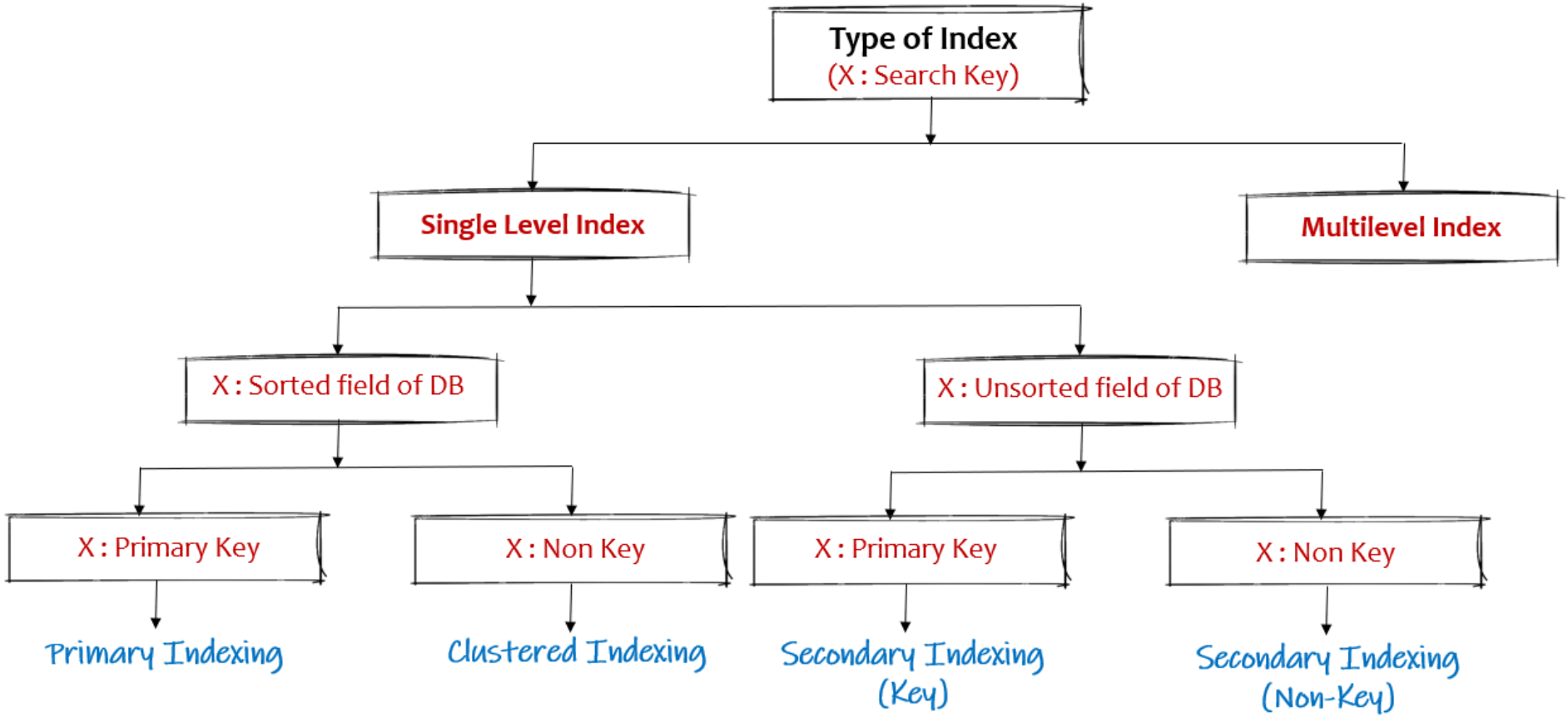
- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on any one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries <**field value, pointer to record>**, which is ordered by field value
- The index is called an access path on the field.

Types of Single-Level Indexes

- Primary/Ordered Index
  - Ordering key field from ordered file record and unique
- Clustering Index
  - Ordering field is not key field, field with non-distinct tuples

  (Atmost one primary index or clustering index but not both)
- Secondary Index
  - Index can be on any non-ordering field of file
  - File can have many secondary index





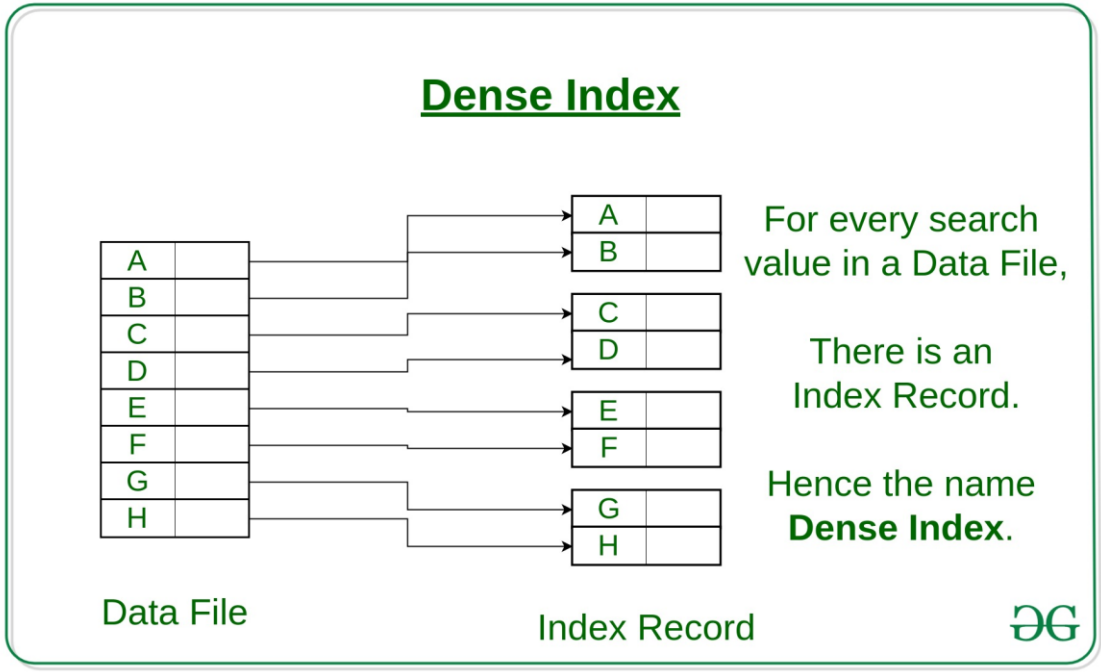
Indexes as Access Paths (contd.)

- **1. Ordered Index File:**
  - In this, the indices are based on a sorted ordering of the values.
  - These are generally fast and a more traditional type of storing mechanism.
  - These Ordered or Sequential file organization might store the data in a **dense or sparse format**:

Primary/Ordered Index

i. Dense Index:

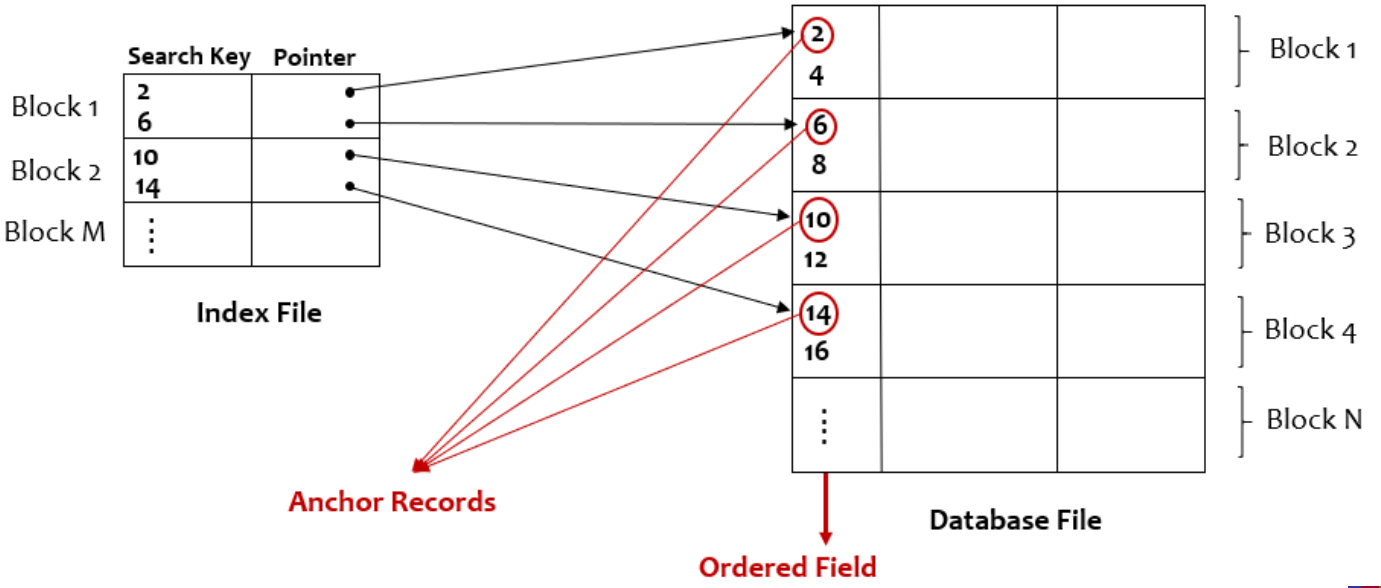
- **For every** search key value in the data file, there is an index record.
- This record contains the search key and also a reference to the first data record with that search key value.



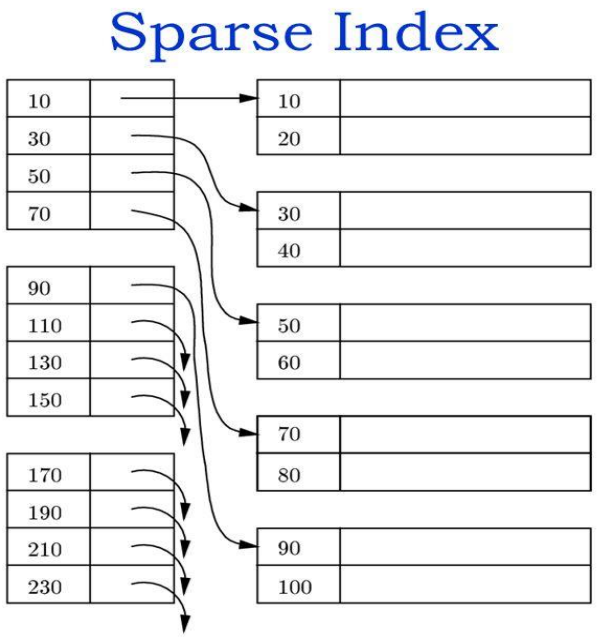
Primary/Ordered Index

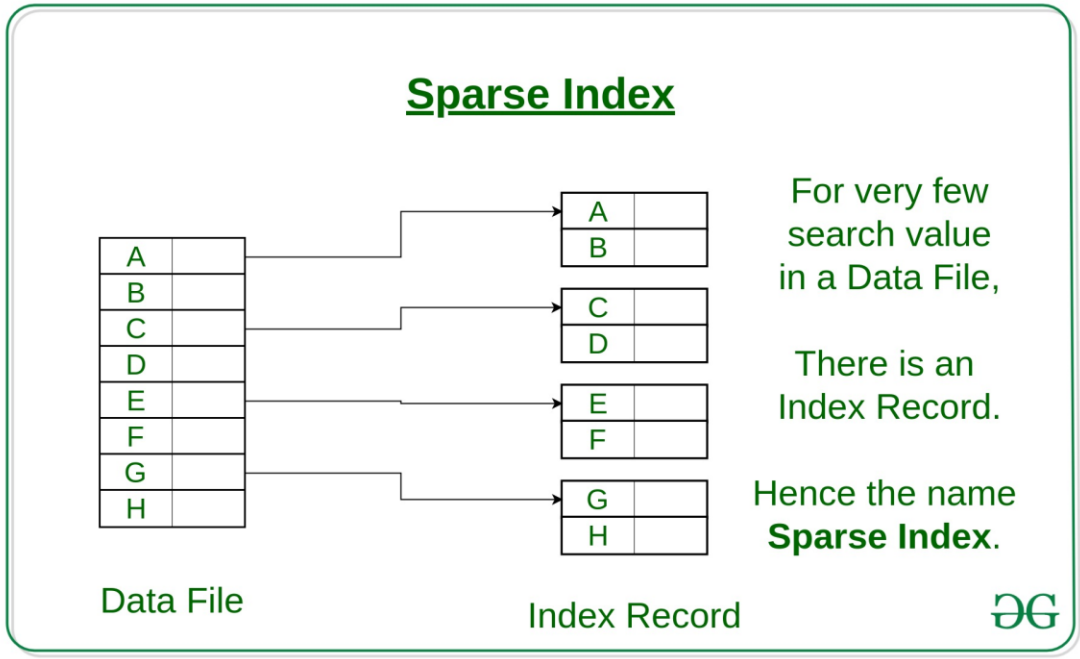
ii. Sparse Index:

- The **index record appears only for a few items** in the data file. Each item points to a block as shown.
- The sparse index can be built only on the ordered field of the database file. **The first record of the block is called the anchor record**.



Primary/Ordered Index



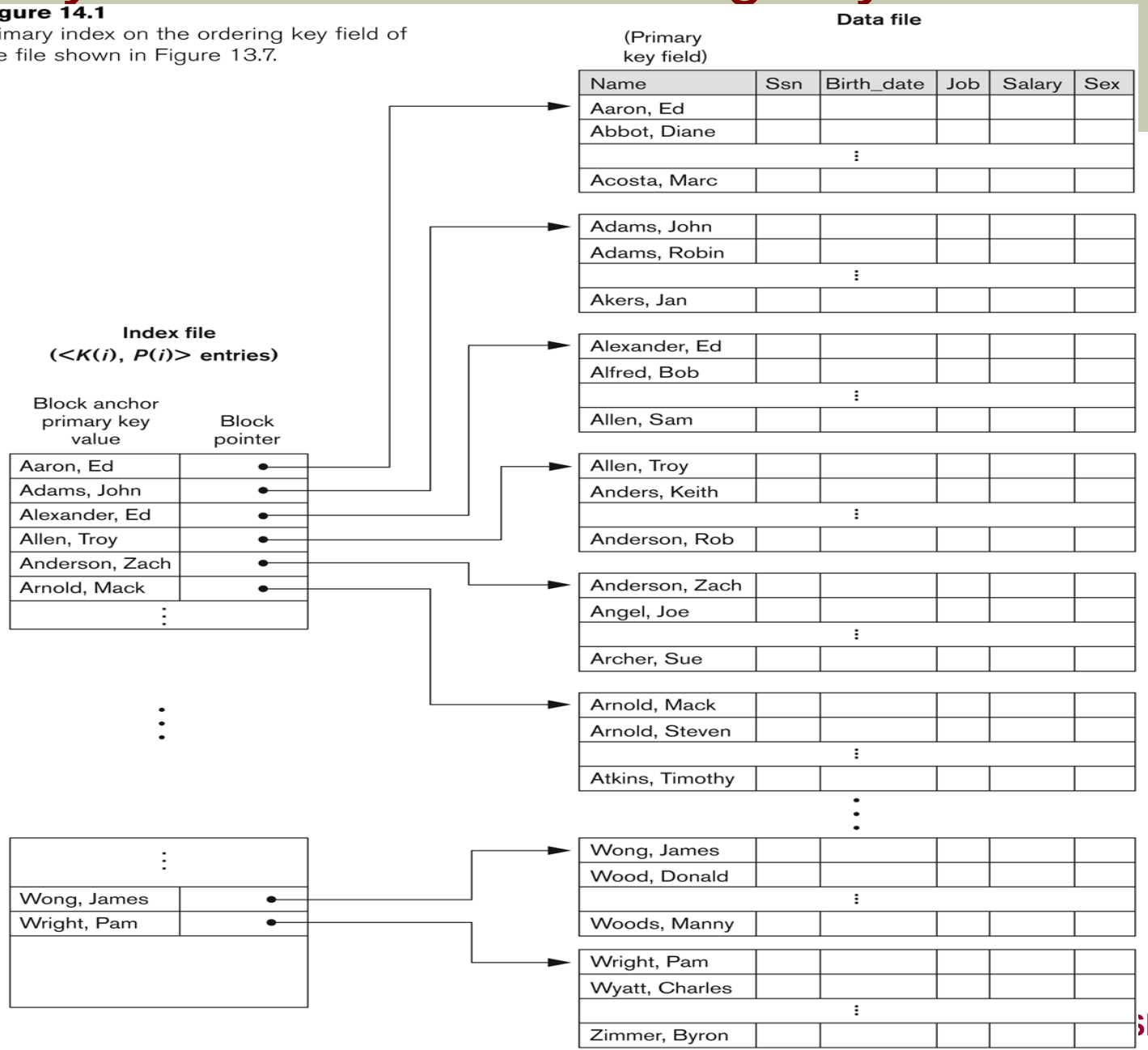


Primary/Ordered Index - Brief

- Defined on an ordered data file
- The data file is ordered on a **key field**
- Includes one index entry *for each block* in the data file; the index entry has the key field value for the *first record* in the block, which is called the *block anchor*
- A primary index is a non-dense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.
- *Note: no. of entries in index file = no. of blocks in data file*
- *Complexity: ($\log_2 n + 1$)*

Primary index on the ordering key field

Figure 14.1
Primary index on the ordering key field of the file shown in Figure 13.7.



Numerical

- The value of Blocking Factor (bfr) called as fan-out (fo)

Example 1. Suppose that we have an ordered file with $r = 30{,}000$ records stored on a disk with block size $B = 1024$ bytes. File records are of fixed size and are unspanned, with record length $R = 100$ bytes. The blocking factor for the file would be $bfr = \lfloor (B/R) \rfloor = \lfloor (1024/100) \rfloor = 10$ records per block. The number of blocks needed for the file is $b = \lceil (r/bfr) \rceil = \lceil (30000/10) \rceil = 3000$ blocks. A binary search on the data file would need approximately $\lceil \log_2 b \rceil = \lceil (\log_2 3000) \rceil = 12$ block accesses.

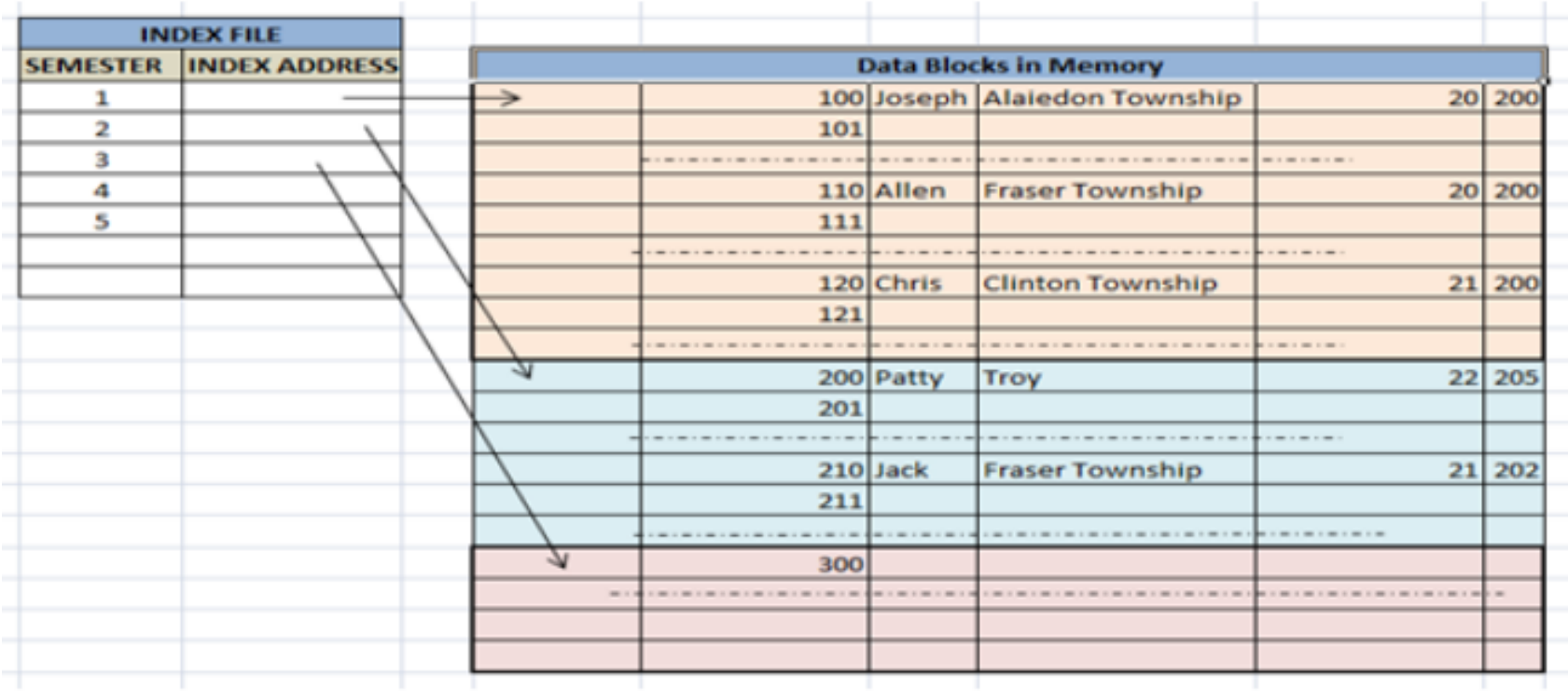
Now suppose that the ordering key field of the file is $V = 9$ bytes long, a block pointer is $P = 6$ bytes long, and we have constructed a primary index for the file. The size of each index entry is $R_i = (9 + 6) = 15$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor = 68$ entries per block. The total number of index entries r_i is equal to the number of blocks in the data file, which is 3000. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (3000/68) \rceil = 45$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 45) \rceil = 6$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of $6 + 1 = 7$ block accesses—an improvement over binary search on the data file, which required 12 disk block accesses.

Clustering Index

- Defined on an ordered data file
- The data file is ordered on a *non-key field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
- Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
- It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.
- *Note: no. of entries in index file = no. of distinct values of non-key field in data file*

A Clustering Index Example

- For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc. are grouped.

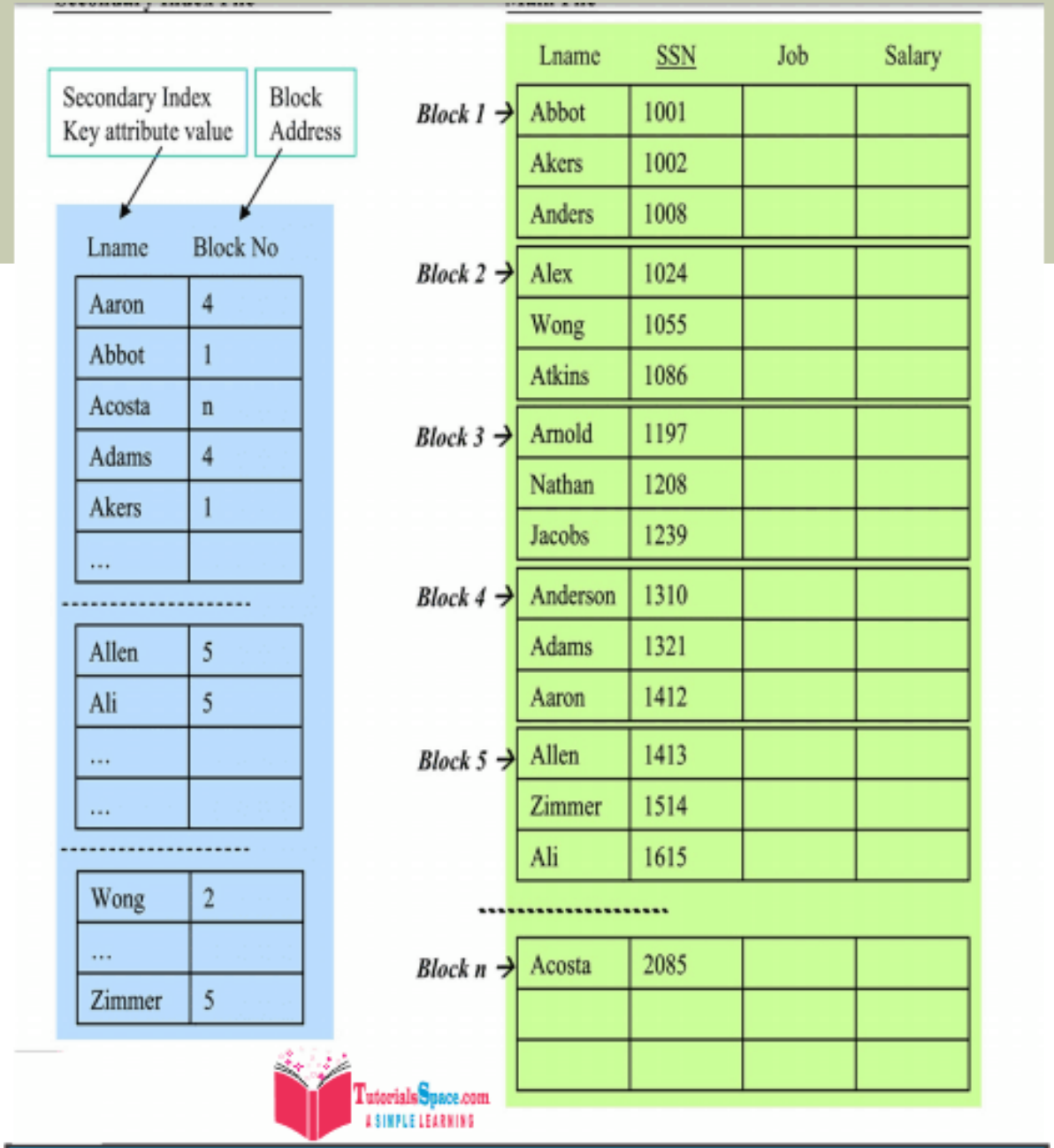


Non-clustered or Secondary Index

- Unordered data file
- The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- The index is an ordered file with two fields.
  - The first field is of the same data type as some **non-ordering field** of the data file that is an indexing field.
  - The second field is either a **block** pointer or a record pointer.
  - There can be *many* secondary indexes (and hence, indexing fields) for the same file.
- Includes one entry *for each record* in the data file; hence, it is a *dense index*
- *Note: no. of entries in index file = no. of records in data file*

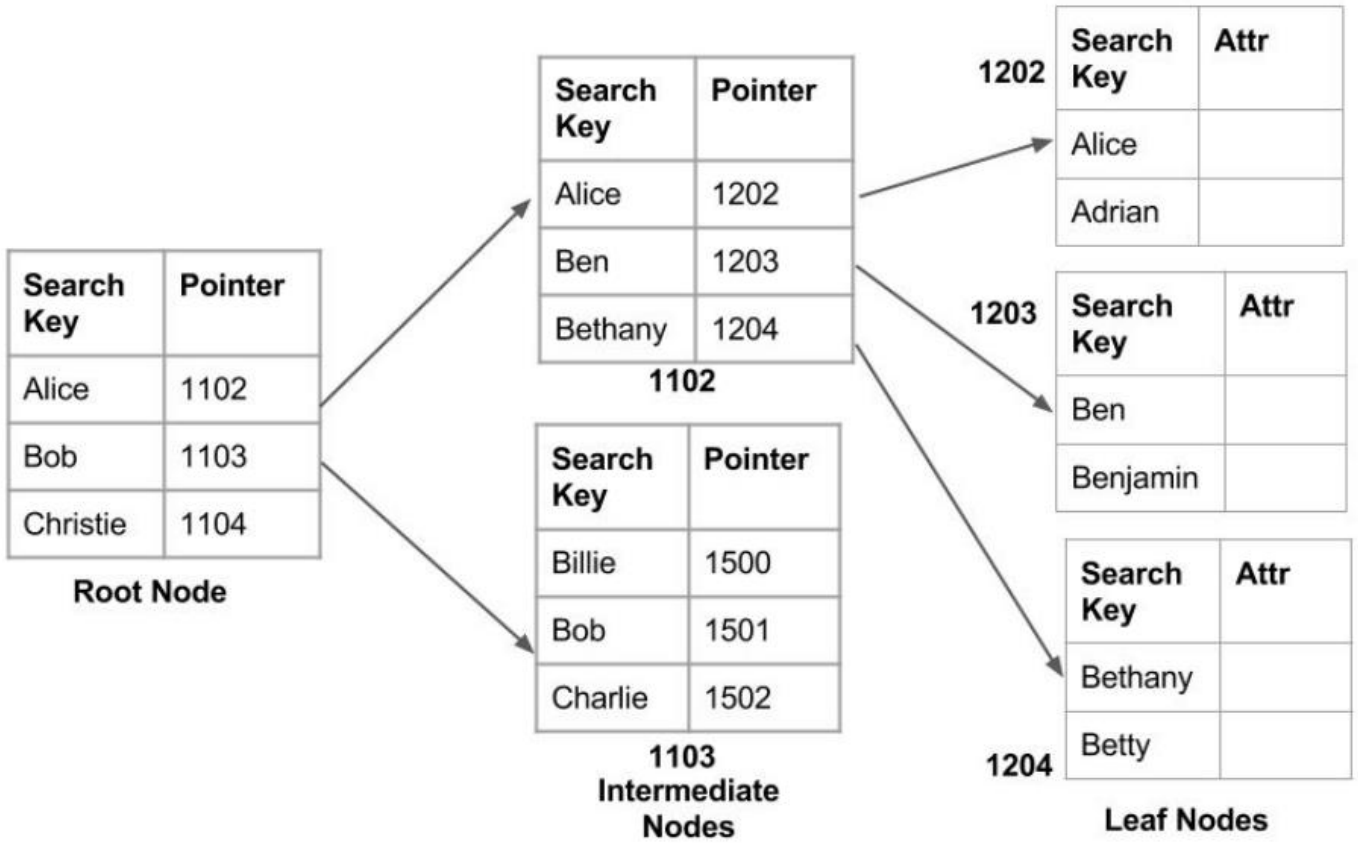
Non-clustered or Secondary Index

- A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored.
  - Data is not physically stored in the order of the index.
  - Instead, **data is present in leaf nodes**.
- It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer.



Secondary Indices

Example of a Dense Secondary Index



Non clustered index

Numerical

Example 2. Consider the file of Example 1 with $r = 30{,}000$ fixed-length records of size $R = 100$ bytes stored on a disk with block size $B = 1024$ bytes. The file has $b = 3000$ blocks, as calculated in Example 1. Suppose we want to search for a record with a specific value for the secondary key—a nonordering key field of the file that is $V = 9$ bytes long. Without the secondary index, to do a linear search on the file would require $b/2 = 3000/2 = 1500$ block accesses on the average. Suppose that we construct a secondary index on that *nonordering key* field of the file. As in Example 1, a block pointer is $P = 6$ bytes long, so each index entry is $R_i = (9 + 6) = 15$ bytes, and the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor = 68$ entries per block. In a dense secondary index such as this, the total number of index entries r_i is equal to the *number of records* in the data file, which is 30,000. The number of blocks needed for the index is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (3000/68) \rceil = 442$ blocks.

A binary search on this secondary index needs $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 442) \rceil = 9$ block accesses. To search for a record using the index, we need an additional block access to the data file for a total of $9 + 1 = 10$ block accesses—a vast improvement over the 1500 block accesses needed on the average for a linear search, but slightly worse than the 7 block accesses required for the primary index. This difference arose because the primary index was nondense and hence shorter, with only 45 blocks in length.

Multi-Level Indexes

- With the growth of the size of the database, indices also grow.
- As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses.
- The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block.
- The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.

Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself.*
  - In this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index*.
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

Multi-Level Indexes

- The value of Blocking Factor (bfr) called as fan-out (fo) of multilevel index
- Searching a multilevel index requires approximately ($\log_{fo} bi$) block accesses.
- First level requires (r1/fo) blocks, which is no. of entries of index at 2nd level
- Increase no. of levels until all the entries at some index level ‘t’ fit in a single block, known as top level index

Multi-Level Indexes

- Each level reduces the no. of entries at the previous level by factor of fo.
- To calculate t, use formula 1<=(r1/(fo)^t)
- Hence, multilevel index with r1 first level entries will have approximately t levels,

Where, $t = \text{ceil}(\log_{fo}(r1))$

(Note**: Refer numerical examples from Navathe to calculate no. of block accesses for primary, secondary and multilevel indexing).

Multi-Level Indexes

- A common file organization used in business data processing is an ordered file with a multilevel primary index on its ordering key field. Such an organization is called an **indexed sequential file – Indexed Sequential Access Method(ISAM) and was used in a large number of early IBM systems.**

A Two-level Primary Index

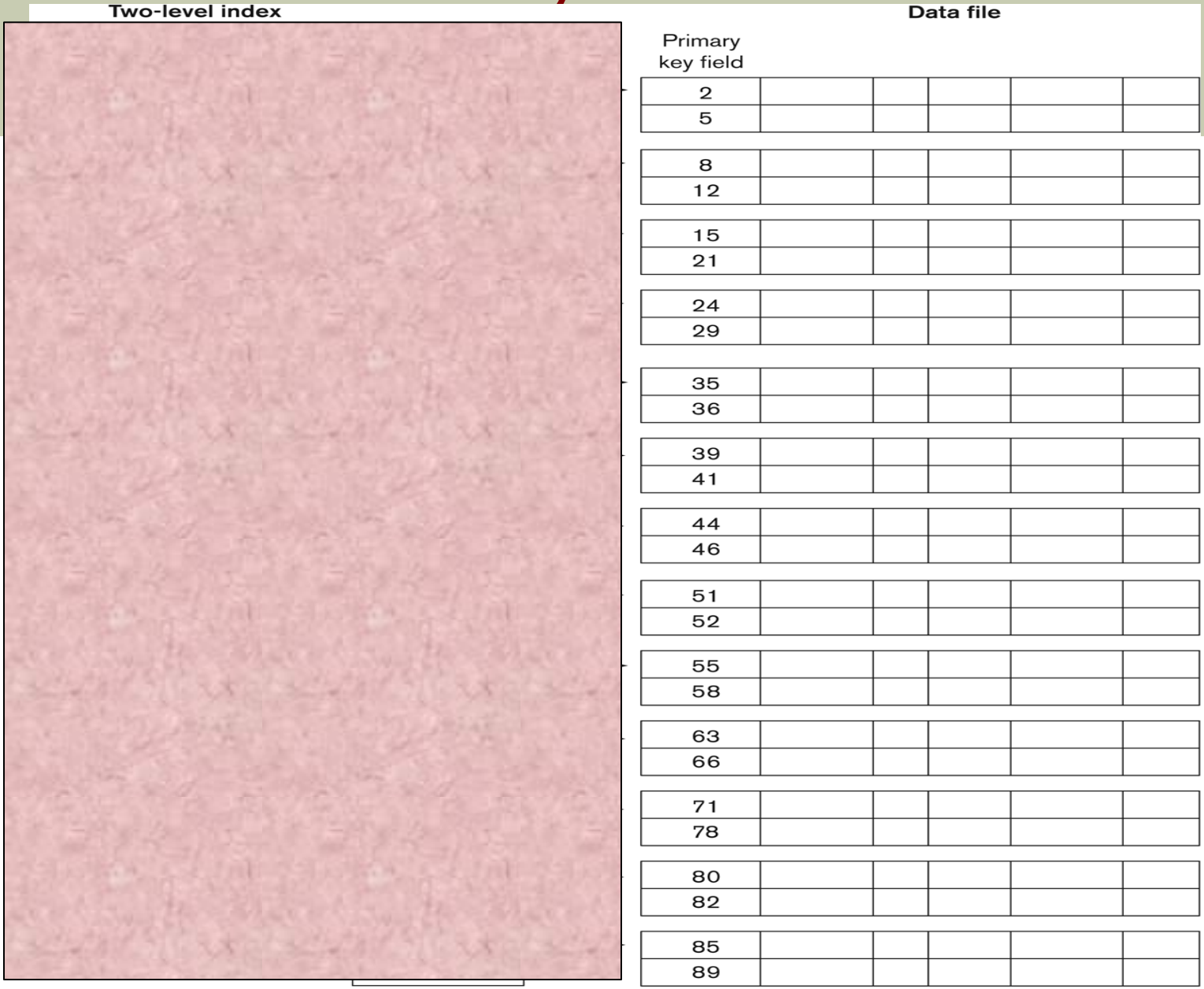


Figure 14.6
A two-level primary index resembling ISAM (Index Sequential Access Method) organization.

A Two-level Primary Index

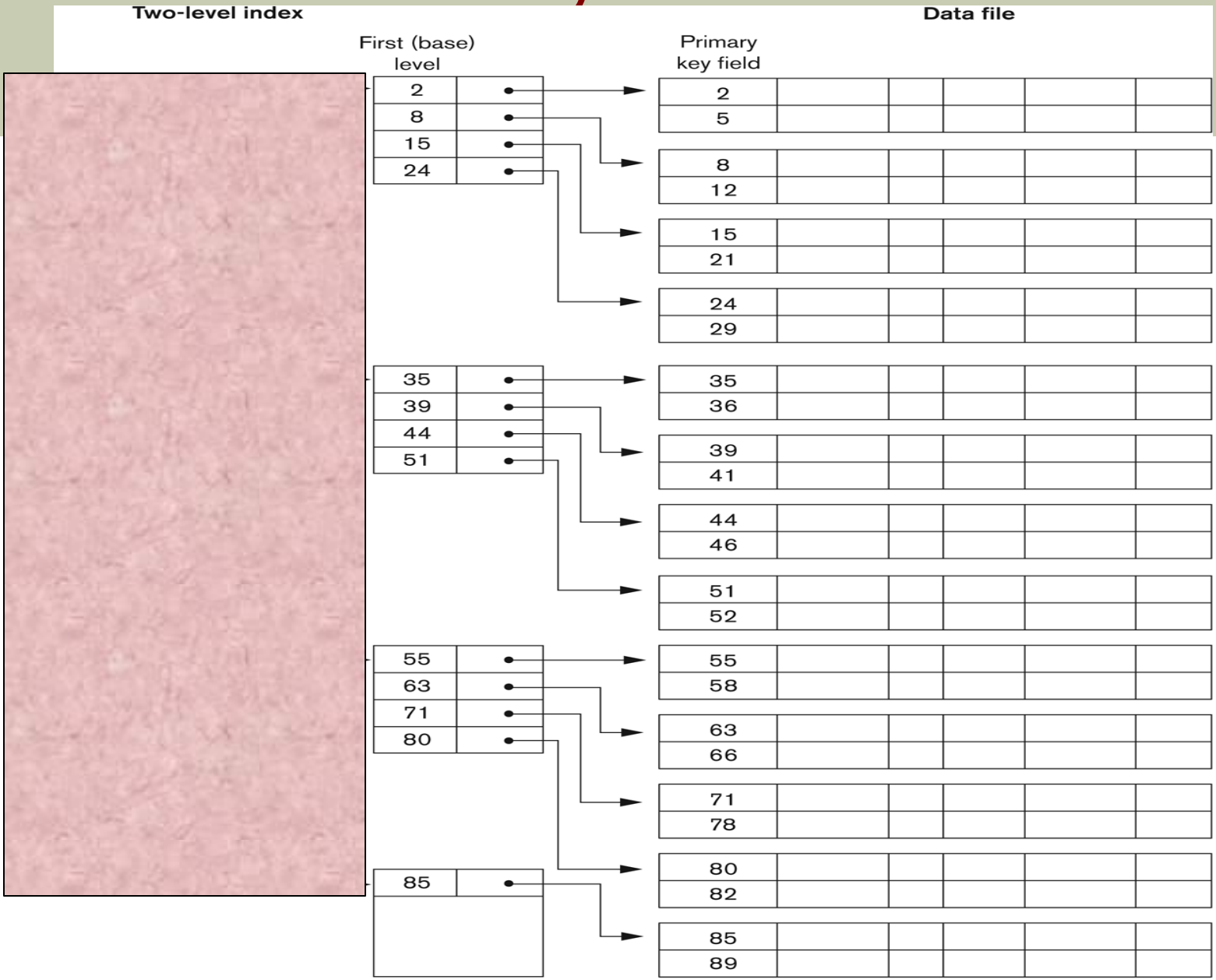


Figure 14.6
A two-level primary index resembling ISAM (Index Sequential Access Method) organization.

A Two-level Primary Index

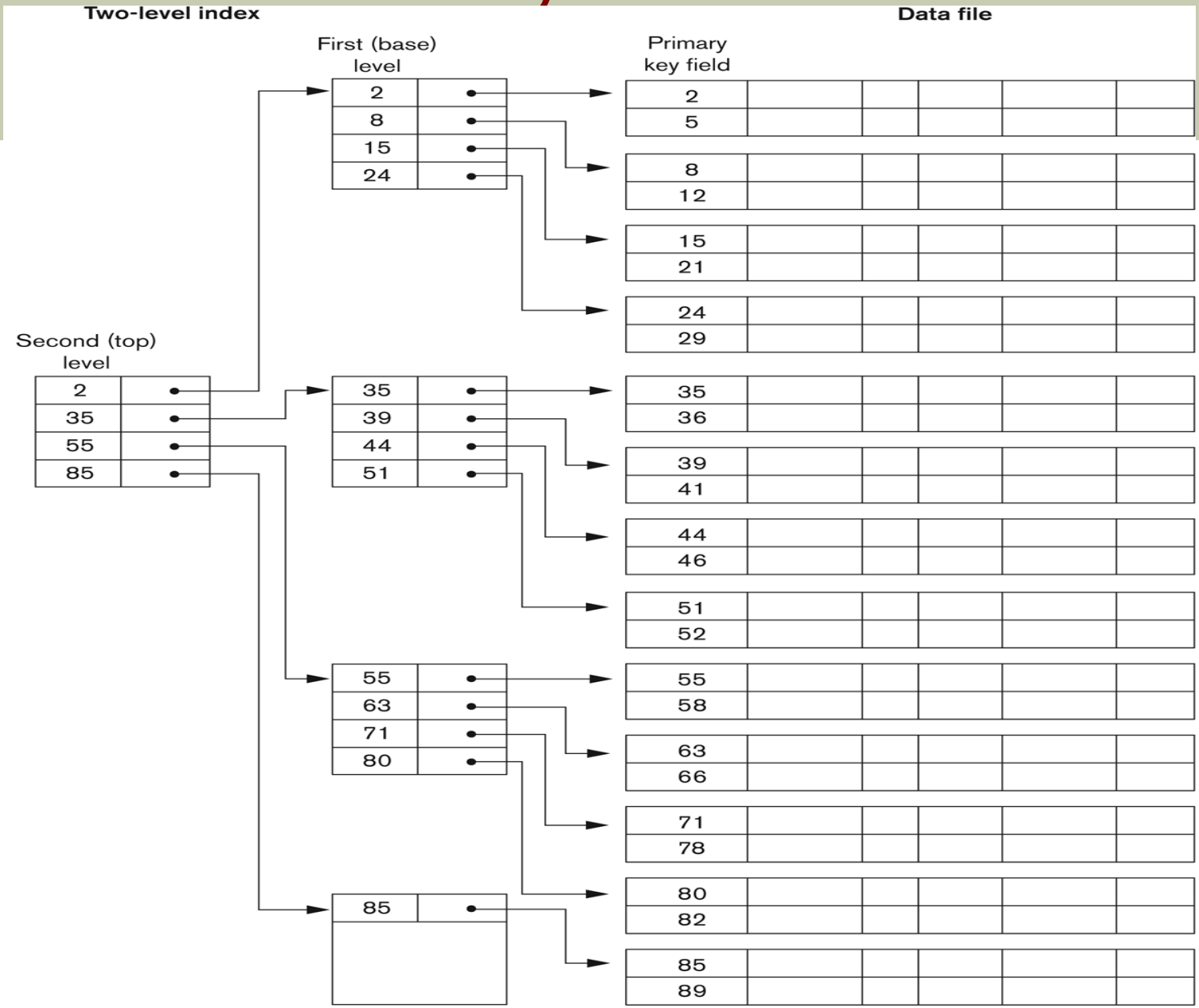


Figure 14.6
A two-level primary index resembling ISAM (Index Sequential Access Method) organization.

Multi-Level Indexes

- Such a multi-level index is a form of *search tree*
  - However, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file*.

Indexes as Access Paths (contd.)

- Example: Given the following data file EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)
- Suppose that:
  - record size R=150 bytes block size B=512 bytes r=30000 records
- Then, we get:
  - blocking factor Bfr= B div R= 512 div 150= 3 records/block
  - number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks
- For an index on the SSN field, assume the field size V_{SSN}=9 bytes, assume the record pointer size P_R=7 bytes. Then:
  - index entry size R_I=(V_{SSN}+ P_R)=(9+7)=16 bytes
  - index blocking factor Bfr_I= B div R_I= 512 div 16= 32 entries/block
  - number of index blocks b= (r/ Bfr_I)= (30000/32)= 938 blocks
  - binary search needs $\log_2 bI = \log_2 938$= 10 block accesses
  - This is compared to an average linear search cost of:
    - (b/2)= 30000/2= 15000 block accesses
  - If the file records are ordered, the binary search cost would be:
    - $\log_2 b = \log_2 30000$= 15 block accesses

Reference

- Navathe