Ceng 302

Database Management Systems

Fundamental File Structure and Indexing Concepts

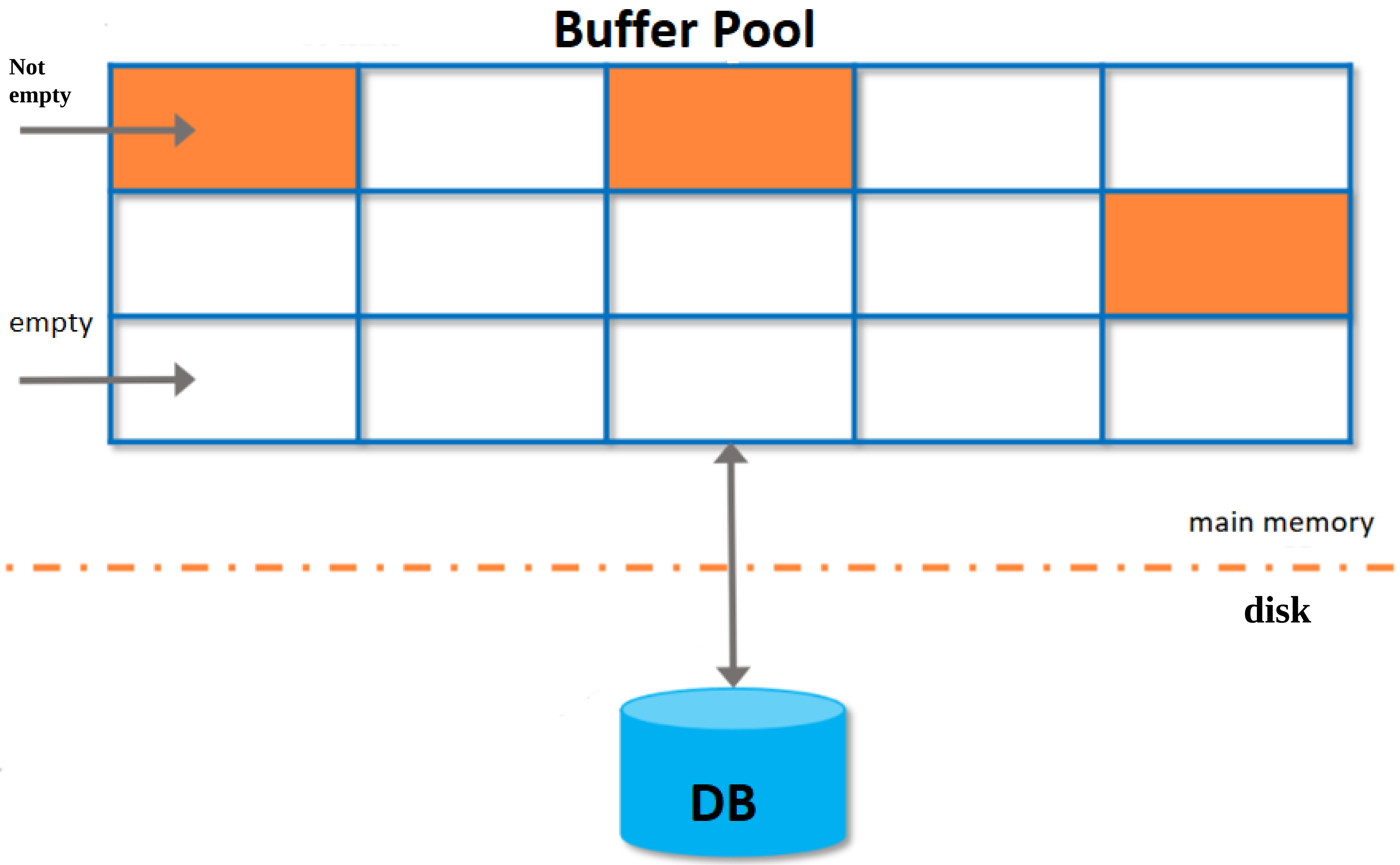
Prof. Dr. Adnan YAZICI

Department of Computer Engineering,

Middle East technical University

(Fall 2021)

Files



Files

A file can be seen as

1. A stream of bytes (no structure), or
   - File is viewed as a sequence of bytes:

```
87359CarrollAlice in wonderland38180FolkFile Structures ...
```

   - Data semantics is lost: there is no way to get it apart again.

2. A collection of records with fields

Field and Record Organization

Record: a collection of related fields.

Field: the smallest logically meaningful unit of information in a file.

Key: a subset of the fields in a record used to identify (uniquely) the record.

e.g. In the example file of books:

- Each line corresponds to a record.
- Fields in each record: ISBN, Author, Title

Record Keys

- **Primary key**: a key that uniquely identifies a record.
- **Secondary key**: other keys that may be used for search
  - Author name
  - Book title
  - Author name + book title
- Note that in general not every field is a key (keys correspond to fields, or a combination of fields, that may be used in a search).

Field Structures

- Fixed-length fields

```
87359Carroll   Alice in wonderland
38180Folk      File Structures
```

- Begin each field with a length indicator

```
058735907Carroll19Alice in wonderland
053818004Folk15File Structures
```

- Place a delimiter at the end of each field

```
87359|Carroll|Alice in wonderland|
38180|Folk|File Structures|
```

- Store field as `keyword = value`

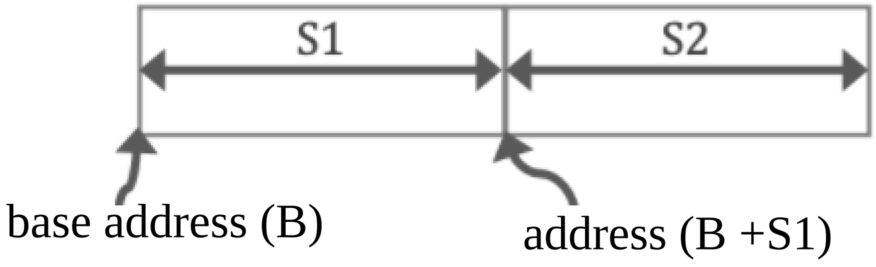
```
ISBN=87359|AU=Carroll|TI=Alice in wonderland|
ISBN=38180|AU=Folk|TI=File Structures
```

Record Structures

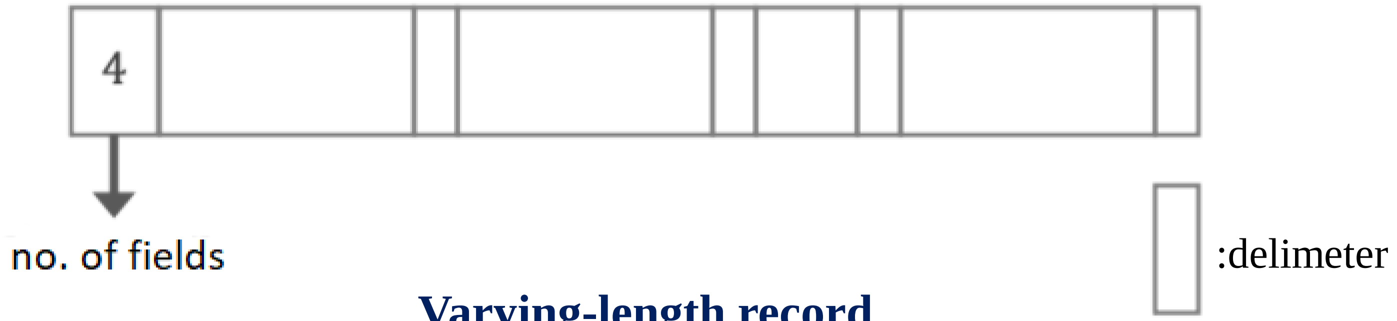
1. Fixed-length records.
2. Fixed number of fields.
3. Begin each record with a length indicator.
4. Use an index to keep track of addresses.
5. Place a delimiter at the end of the record.

Record Structures

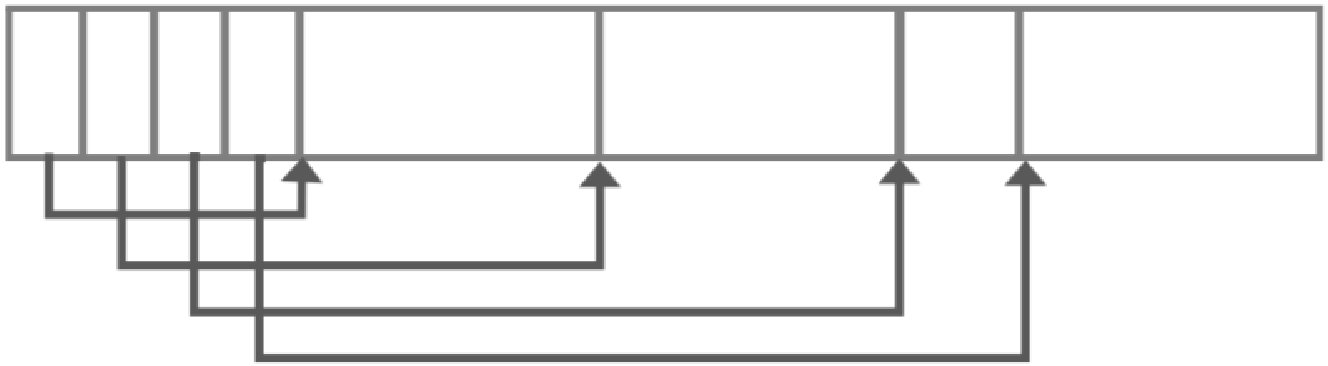
Fixed-sized record



Varying-length record



Varying-length record



Fixed-length records

Two ways of making fixed-length records:

1. Fixed-length records with fixed-length fields.

87359	Carroll	Alice in wonderland
03818	Folk	File Structures

2. Fixed-length records with variable-length fields.

87359|Carroll|Alice in wonderland|	*unused*
38180|Folk|File Structures|	*unused*

Variable-length records

- Fixed number of fields:

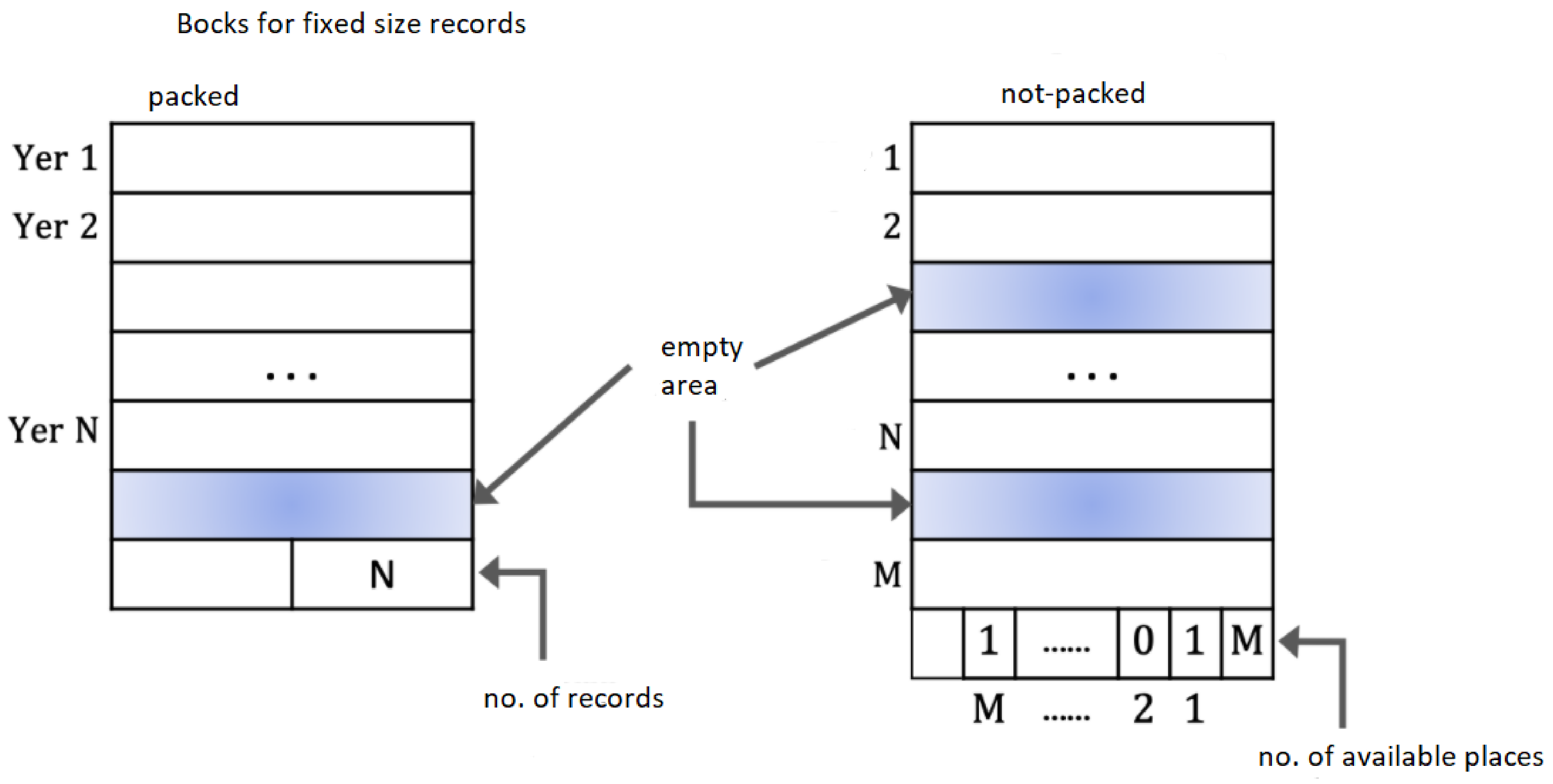
```
87359|Carroll|Alice in wonderland|38180|Folk|File Structures| ...
```

- Record beginning with length indicator:

```
3387359|Carroll|Alice in wonderland|2638180|Folk|File Structures| ..
```

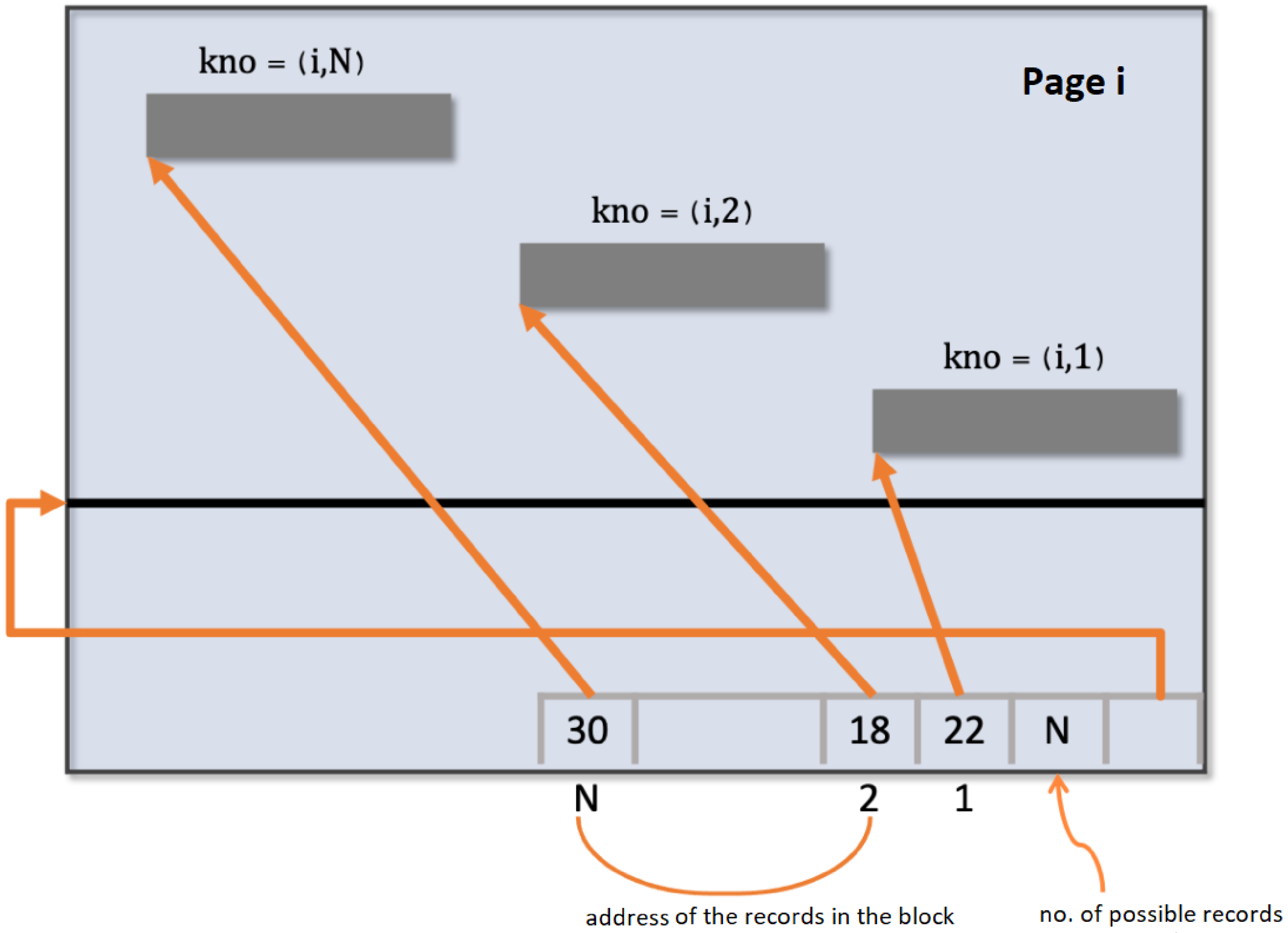
- Use an index file to keep track of record addresses:
  - The index file keeps the byte offset for each record; this allows us to search the index (which have fixed length records) in order to discover the beginning of the record.
- Placing a delimiter: e.g. end-of-line char

Block Structures



Block Structures

blocks for variable length of records



File Organization

- Four basic types of organization:
  1. Sequential
  2. Indexed
  3. Indexed Sequential
  4. Hashed
- In all cases we view a **file** as a sequence of **records.**
- A **record** is a list of **fields**.
- Each **field** has a **data type**.

File Operations

- **Typical Operations:**
  - Retrieve a record
  - Insert a record
  - Delete a record
  - Modify a field of a record
- **In direct files:**
  - Get a record with a given field value
- **In sequential files:**
  - Get the next record

Sequential files

- Records are stored contiguously on the storage device.
- Sequential files are read from beginning to end.
- Some operations are very efficient on sequential files (e.g., finding averages)
- Organization of records:

1. Unordered sequential files (**pile files**)
2. **Sorted** sequential files (records are ordered by some field)

Pile Files

- A **pile file** is a succession of records, simply placed one after another with no additional structure.
- Records may vary in length.
- Typical Request:
  - Print all records with a given field value
    - e.g., print all books by *Folk*.
  - We must examine each record in the file, in order, starting from the first record.

Searching Sequential Files

- To look-up a record, given the value of one or more of its fields, we must search the whole file.
- In general, (***b*** is the total **number of blocks** in file):
  - At least 1 block is accessed
  - At most *b* blocks are accessed.
  - On average $1/b * b (b + 1) / 2 \Rightarrow b/2$
- Thus, time to find and read a record in a pile file is approximately :

$$\mathbf{T_F = (\boldsymbol{b}/2) * btt}$$

$$\uparrow$$

Time to fetch one record

Exhaustive Reading of the File

- Read and process all records (reading order is not important)

$$\mathbf{T_X = \boldsymbol{b} * btt}$$

(approximately twice the time to fetch one record)

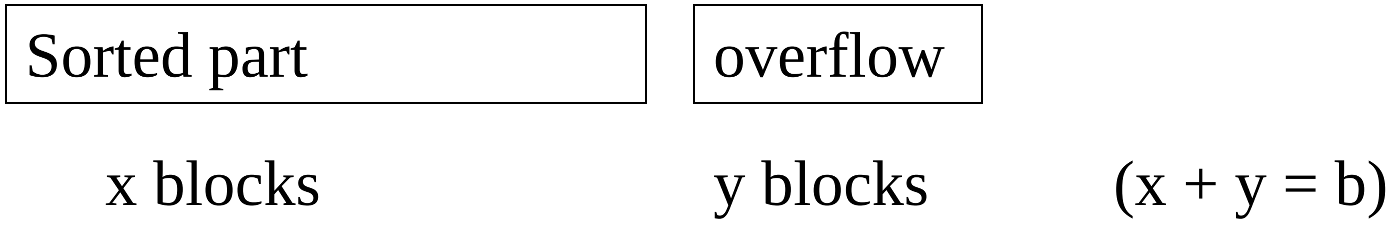
- e.g., Finding averages, min or max, or sum.
  - Pile file is the best organization for this kind of operations.
  - They can be calculated using double buffering as we read though the file once. (Two buffers are used to speed up program execution. Data are processed in one buffer while data are written into or read out of the other.)

Sorted Sequential Files

- Sorted files are usually read sequentially to produce lists, such as mailing lists, invoices, etc.
- A sorted file cannot stay in order after additions (usually it is used as a temporary file).
- A sorted file will have an **overflow area** of added records. Overflow area is not sorted.
- To find a record:
  - First look at sorted area
  - Then search overflow area
  - If there are too many overflows, the access time degenerates to that of a sequential file.

Searching for a record

- We can do binary search (assuming fixed-length records) in the sorted part.



- Worst case to fetch a record :

$$\mathbf{T_F = \log_2 x \;*\; (s + r + btt).}$$

- If the record is not found, search the overflow area too. Thus total time is:

$$\mathbf{T_F = \log_2 x \;*\; (s + r + btt) + s + r + (y/2) * btt}$$

Problem

- Given the following:
  - Block size = 2400
  - File size = 40M
  - Block transfer time (btt) = 0.84ms
  - s = 16ms
  - r = 8.3 ms

Q1) Calculate T_F for a certain record

a) in a pile file

b) in a sorted file (no overflow area)

Q2) Calculate the time to look up 10000 names.

Indexing

- Indexes used to speed up record retrieval in response to certain search conditions
- Index structures provide secondary access paths
- Any field can be used to create an index
  - Multiple indexes can be constructed
- Most indexes based on ordered files
  - Tree data structures organize the index

Types of Single-Level Ordered Indexes

- Ordered index similar to index in a textbook
- Indexing field (attribute)
  - Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value
- Values in index are ordered

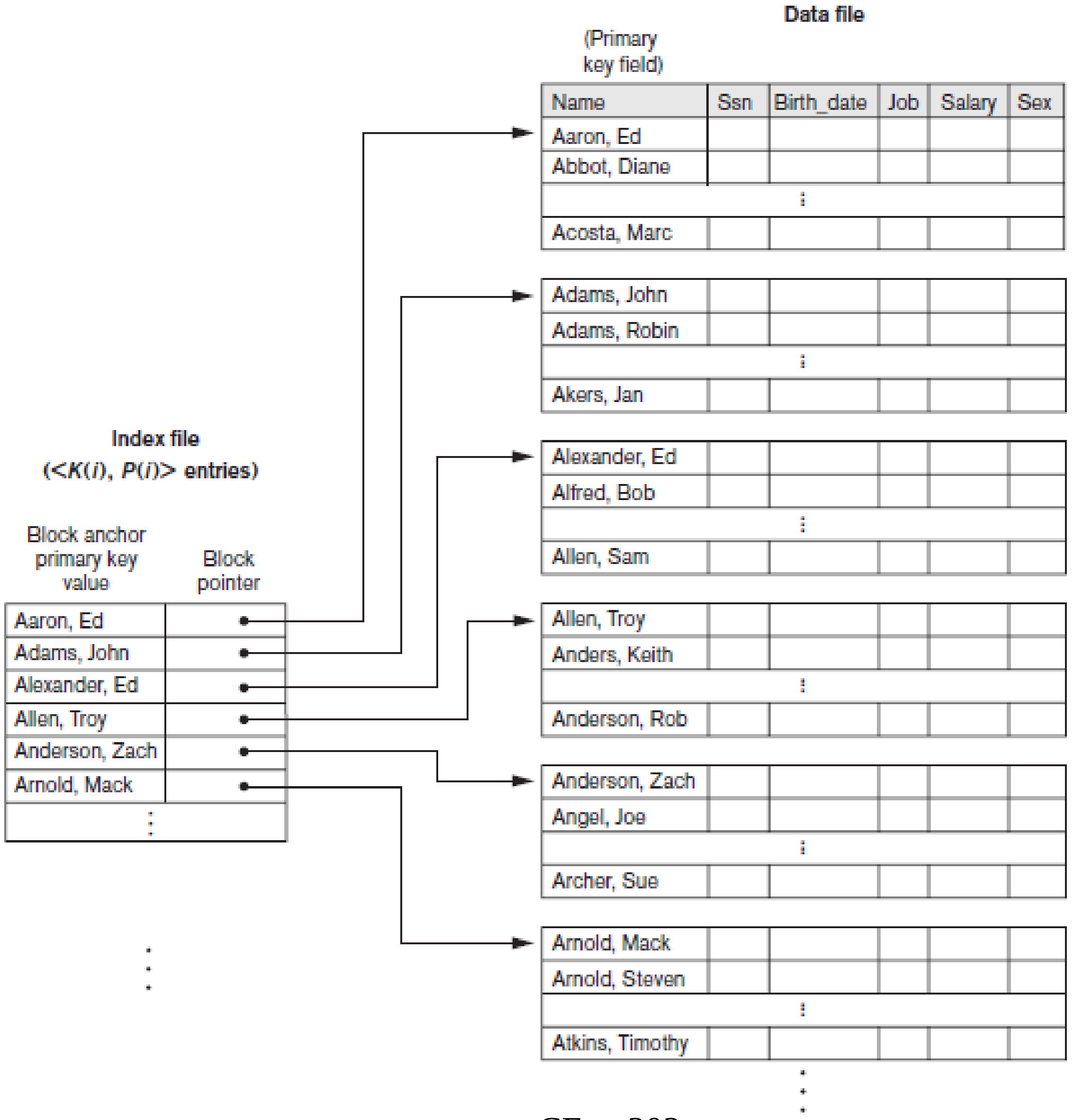
Types of Single-Level Ordered Indexes (cont'd.)

- **Primary index**
  - Specified on the ordering key field of ordered file of records
- **Clustering index**
  - Used if numerous records can have the same value for the ordering field, used for nonkey fields.
- **Secondary index**
  - Can be specified on any nonordering field
  - Data file can have several secondary indexes

Primary Indexes

- Ordered file with two fields
  - Primary key, *K(i)*
  - Pointer to a disk block, *P(i)*
- One index entry in the index file for each block in the data file
- Indexes may be dense or sparse
  - Dense index has an index entry for every search key value in the data file
  - Sparse index has entries for only some search values

Primary Indexes (cont'd.)



Primary index on the ordering key field

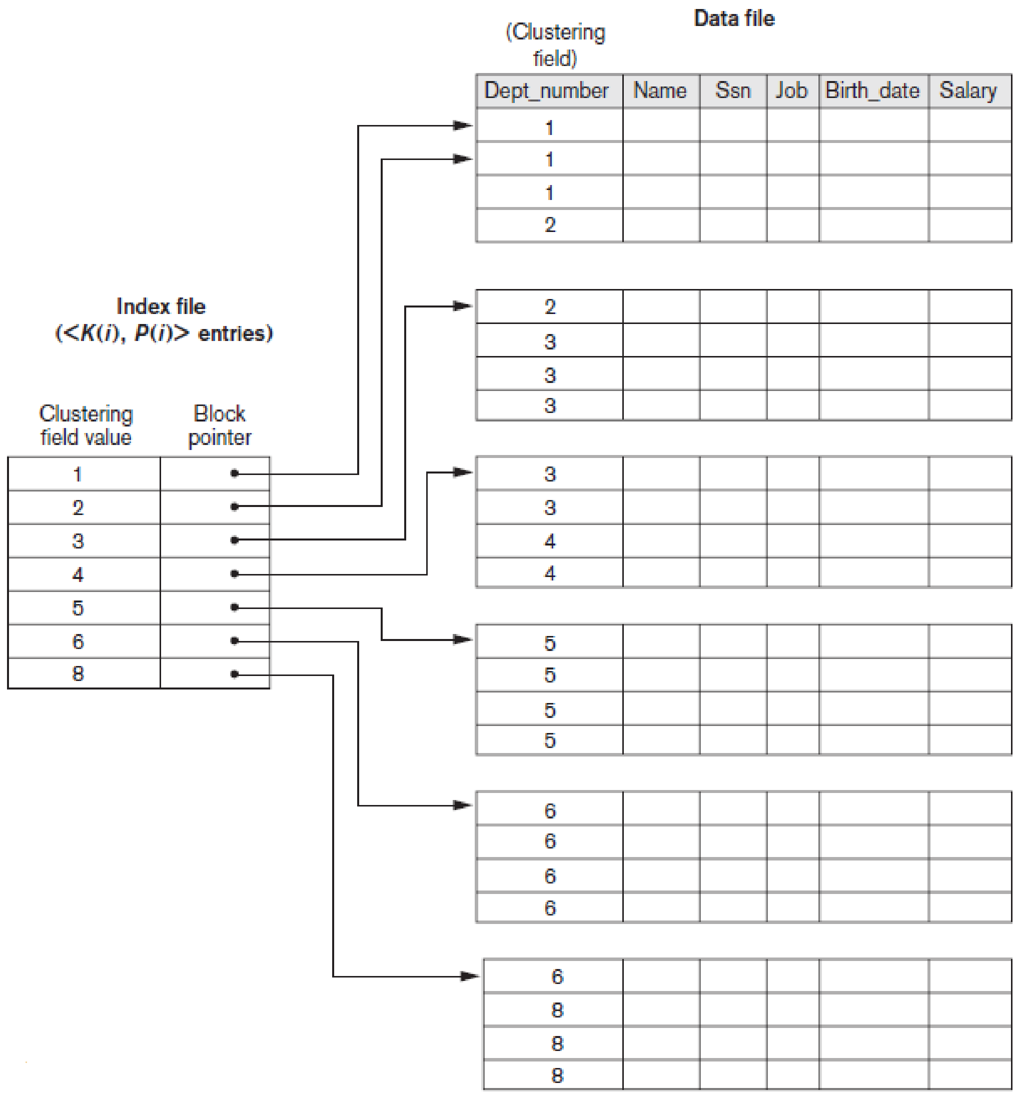
Primary Indexes (cont'd.)

- Major problem: insertion and deletion of records
  - Move records around and change index values
  - Solutions
    - Use unordered overflow file
    - Use linked list of overflow records

Clustering Indexes

- Clustering field
  - File records are physically ordered on a nonkey field without a distinct value for each record
- Ordered file with two fields
  - Same type as clustering field
  - Disk block pointer

Clustering Indexes (cont'd.)

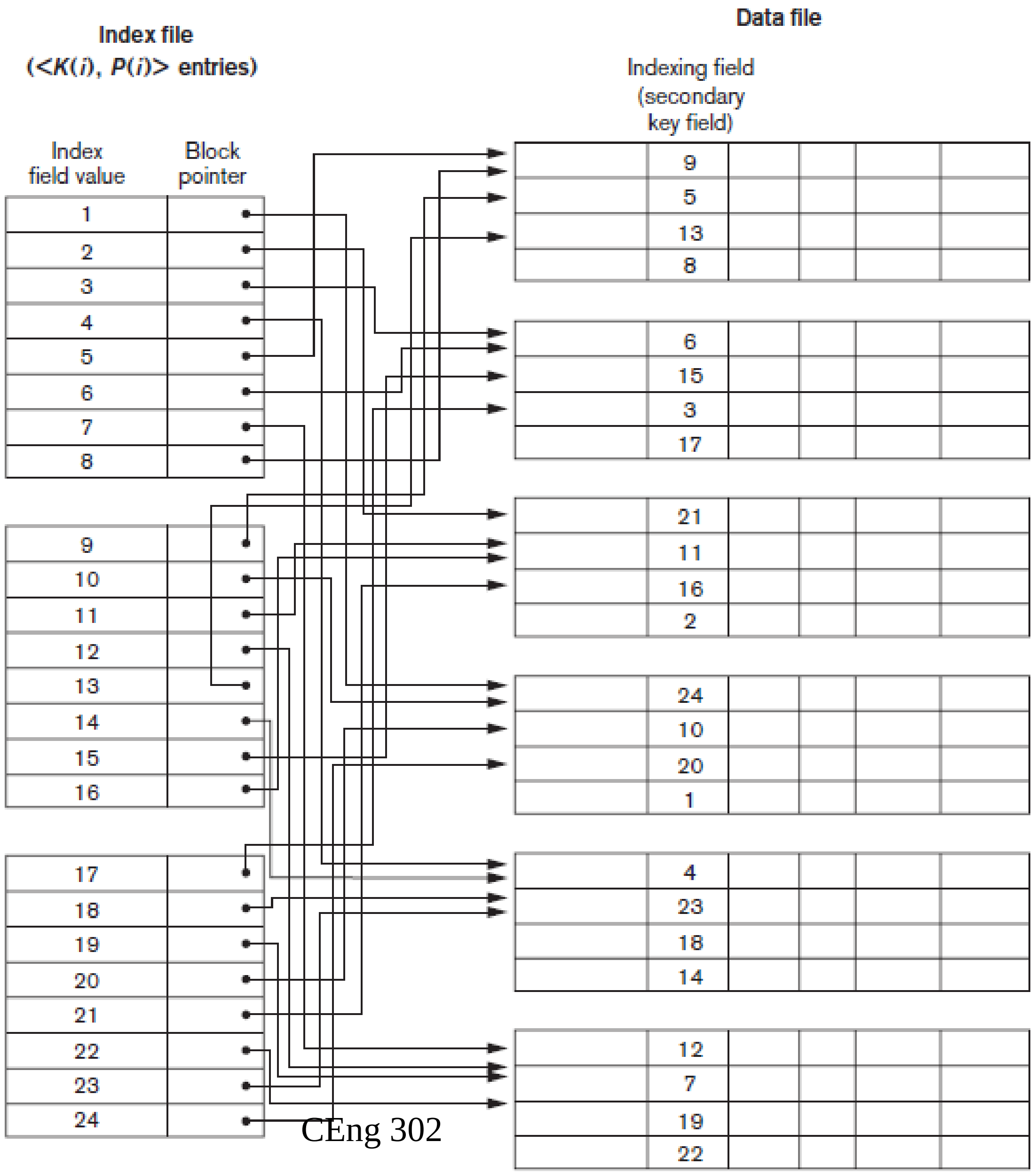


A clustering index on ordering nonkey field of a file

Secondary Indexes

- Provide secondary means of accessing a data file
  - Some primary access exists
- Ordered file with two fields
  - Indexing field, *K(i)*
  - Block pointer or record pointer, *P(i)*
- Usually need more storage space and longer search time than primary index
  - Improved search time for arbitrary record

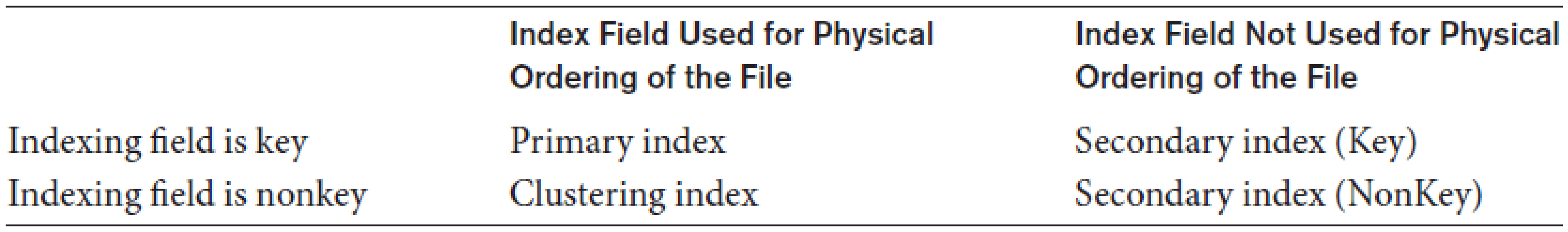
Secondary Indexes (cont'd.)



Dense secondary index (with block pointers) on a nonordering key field of a file.

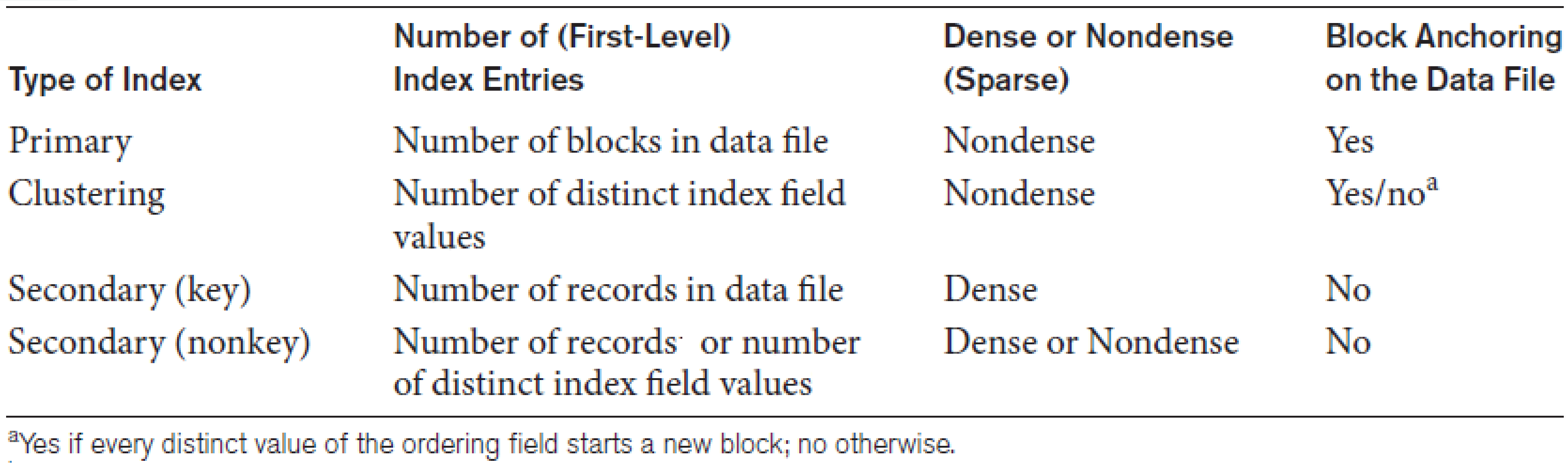
Types of Single-Level Ordered Indexes (cont'd.)

Types of indexes based on the properties of the indexing field



	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Properties of index types



Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records or number of distinct index field values	Dense or Nondense	No

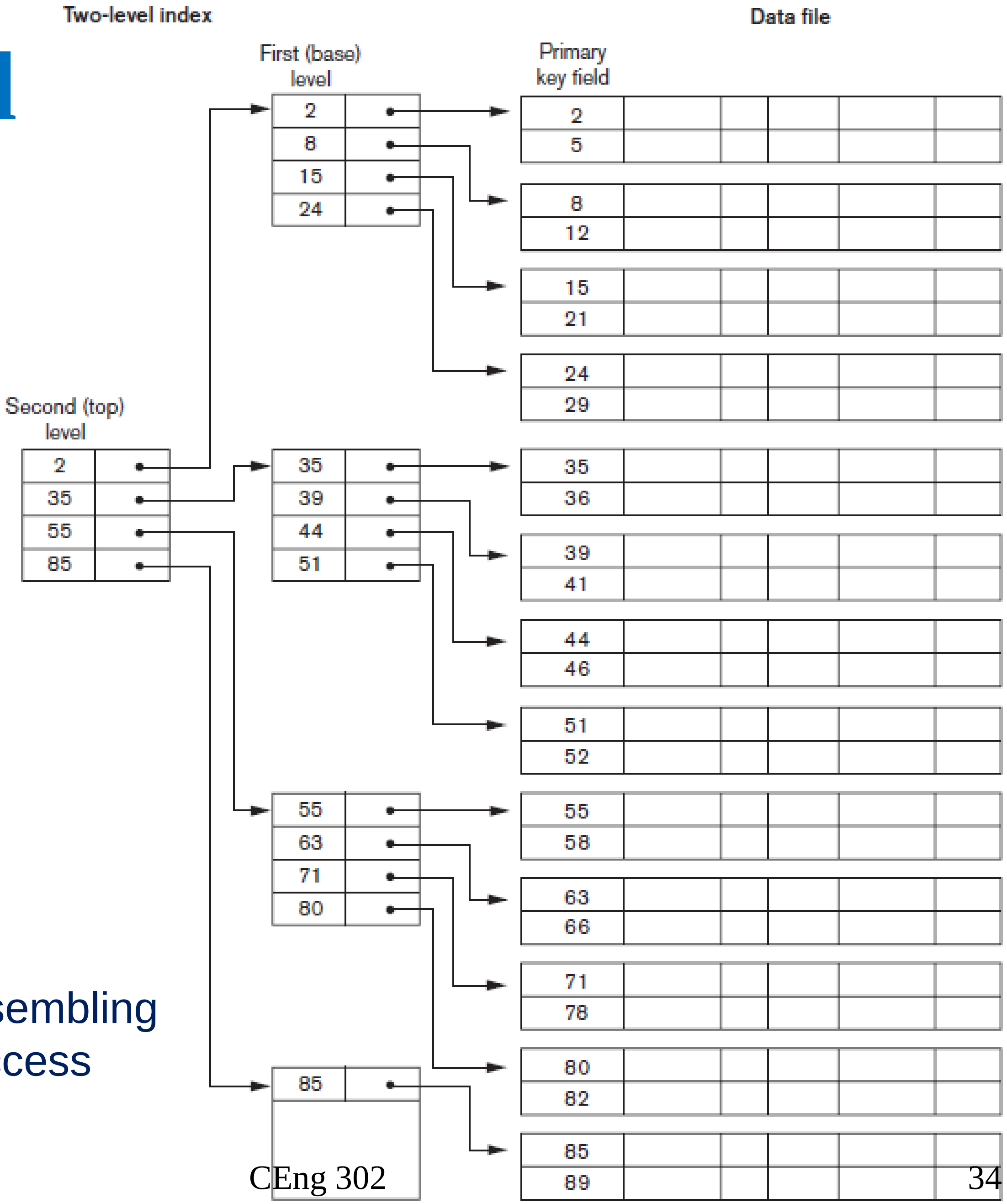
[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.

A block's first record may be referred to as an anchor record of a block or a block anchor.

Multilevel Indexes

- Designed to greatly reduce remaining search space as search is conducted
- Index file
  - Considered first (or base level) of a multilevel index
- Second level
  - Primary index to the first level
- Third level
  - Primary index to the second level

Multilevel Indexes



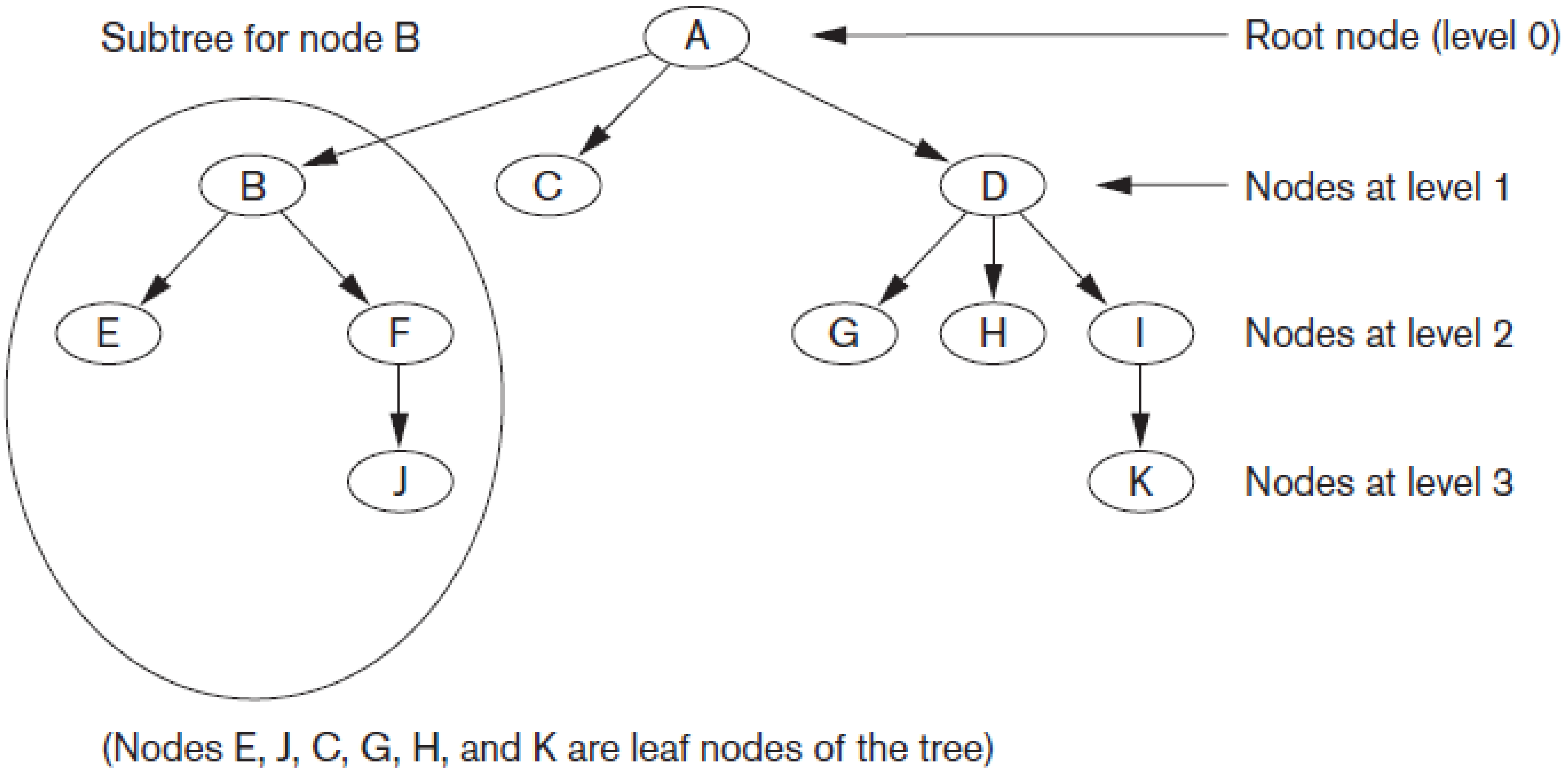
A two-level primary index resembling ISAM (indexed sequential access method) organization

Dynamic Multilevel Indexes Using B-Trees and B+ -Trees

- Tree data structure terminology
  - Tree is formed of nodes
  - Each node (except root) has one parent and zero or more child nodes
  - Leaf node has no child nodes
    - Unbalanced if leaf nodes occur at different levels
  - Nonleaf node called internal node
  - Subtree of node consists of node and all descendant nodes

Tree Data Structure

A **tree** data structure that shows an unbalanced tree


Subtree for node B
A
Root node (level 0)
B
C
D
Nodes at level 1
E
F
G
H
I
Nodes at level 2
J
K
Nodes at level 3
(Nodes E, J, C, G, H, and K are leaf nodes of the tree)