Lesson 9: Indexing Structures

CSC430/530 – DATABASE MANAGEMENT SYSTEMS

DR. ANDREY TIMOFEYEV

OUTLINE

- Introduction.
- Data storage principles.
  - File records.
  - Spanned vs. unspanned records.
  - File organizations.
  - Hashing techniques.
- Indexing structures.
  - Primary indexes.
  - Clustering indexes.
  - Secondary indexes.
  - Multi-level indexes.

INTRODUCTION

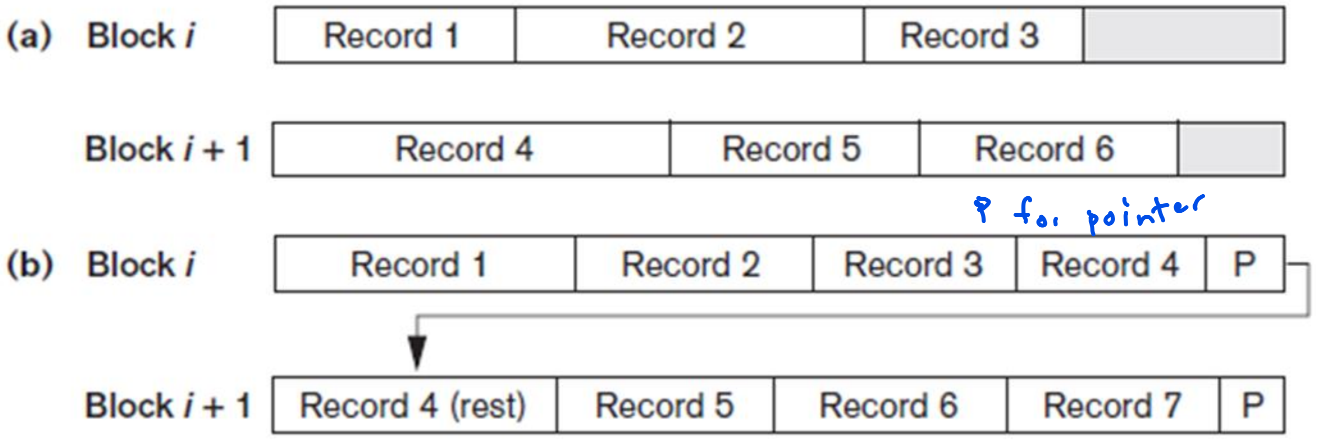
- **Storage hierarchy**:
  - **Primary** storage.
    - CPU memory, cache memory, RAM.
  - **Secondary** storage (mass storage).
    - Magnetic disks, flash memory, solid-state drives.
  - **Tertiary** storage.
    - Removable media.
- **Data** in **databases** is typically stored on **magnetic disks** or **solid-state drives**.
  - **Accessed** using **physical** database **file structures**.

FILE RECORDS

- **Record** is a unit of **measure** with respect to the **storage structures**.
- **Tuples** (rows) in a **relation** (table) are represented as **records**.
  - **Records** comprise of a **sequence** of **fields** (*column*, *attribute*)
  - **Files** on a disk comprise of a **sequences** of **records**.
- Two general **types** of records:
  - **Fixed-length records**.
    - Every record has exact same size.
  - **Variable-length records**.
    - Different records have different sizes.

SPANNED VS UNSPANNED RECORDS

- **File records** are allocated to **disk blocks**.
  - **Blocks** refer to **physical units** of storage in storage devices.
    - Examples: **sectors** in hard disks or **pages** in virtual memory.
- **Records** are classified based on the **block occupation**:
  - **Unspanned records**.
    - Records not allowed to cross block boundaries.
  - **Spanned records**.
    - Larger than a single block.
    - Pointer at end of first block points to block containing remainder of record.



Spanned & unspanned records.

FILE ORGANIZATIONS

- **Relations** (tables) are stored in **files** as **logical** “**records**” & read in terms of **physical** “**blocks**”.
  - **File organization** refers to the way **records** are **stored** in terms of **blocks** and the way **blocks** are **placed** on the **storage** medium and **interlinked**.

- **Types** of file **organizations**:
  - **Unsorted files**.
    - Records placed in file in order of insertion
  - **Sorted files**.
    - Records sorted by ordering field (key field).
  - **Hash files**.
    - Maps data of arbitrary size to data of fixed size.
      - Key-value mapping.

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
⋮						
Block ***n*****−1**	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
Block ***n***	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

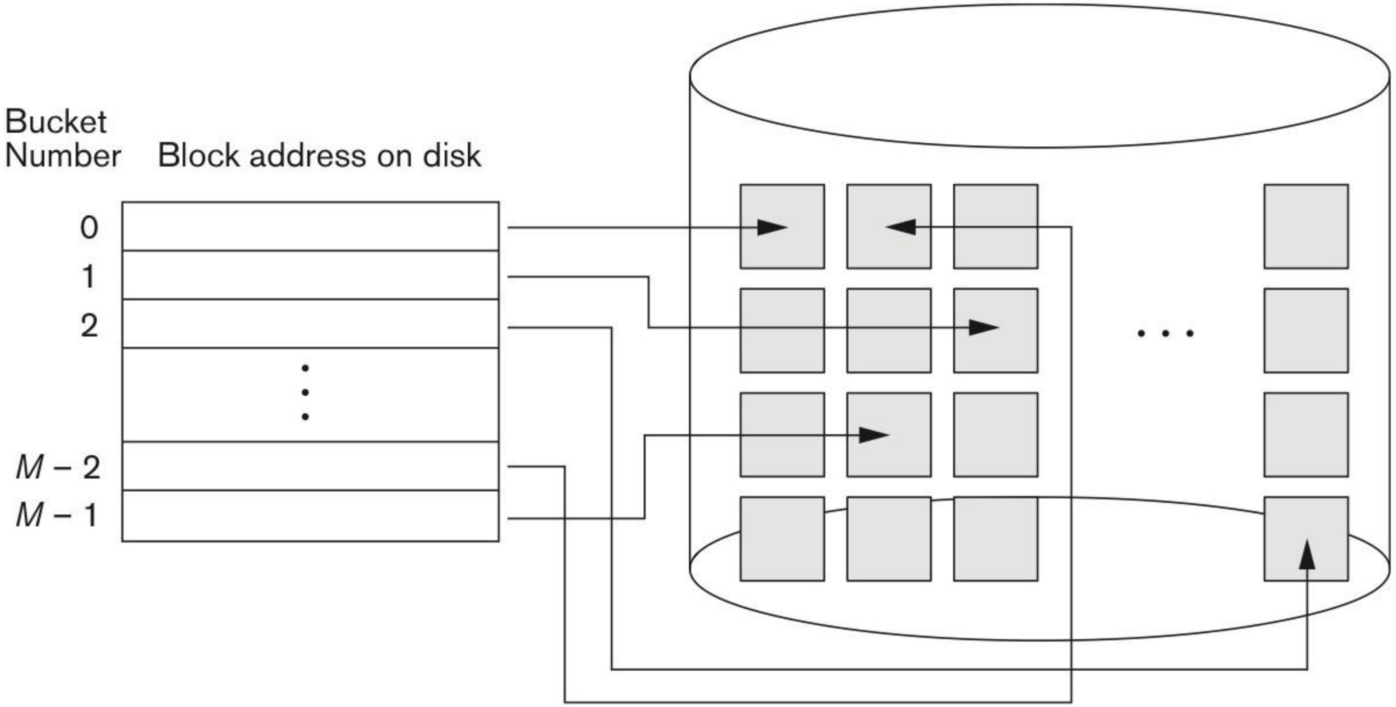
Sorted file organization.

HASHING TECHNIQUES (1)

- Each **record** in a hash file contains a special **hash field** (*hash key*).
  - **Key field** of a hash file.
- **Hash function** (*randomization function*) is **applied** on the **hash field value** of record.
  - Yields the **address** of the disk **block** where the record is **stored**.
- Two **types** of **hashing techniques**:
  - **External hashing**.
    - Operates on **disk files**.
  - **Internal hashing**.
    - Operates on **records** in **files**.
      - Group of records is accessed by using value of one field.
      - Examples: dictionaries or hash tables.

HASHING TECHNIQUES (2)

- In **external hashing** target address space is made of “**buckets**”.
  - **Bucket** – **single** disk **block** or **contiguous blocks**.
- **Hashing function** maps **key** into relative **bucket**.
  - Table in file header converts **bucket number** into disk block **address**.
- Two **types** of **external hashing**:
  - **Static hashing**
    - Allocates fixed number of buckets.
  - **Dynamic hashing**.
    - Allows number of buckets to grow and shrink dynamically.



Matching bucket numbers to disk block addresses.

HASHING TECHNIQUES: COLLISIONS

- **Hashing techniques** are prone to **collision** issue.
  - **Collision** - **hash field value** for inserted record hashes to address **already containing** a different **record**.
- **Collision resolving** techniques:
  - **Open addressing**.
    - Algorithm checks subsequent addresses to find an empty one.
  - **Chaining**.
    - New empty address is referred to by the pointer of occupied hash address.
  - **Multiple hashing**.
    - Second hash function is applied if the first resulted in collision.

INDEXING STRUCTURES

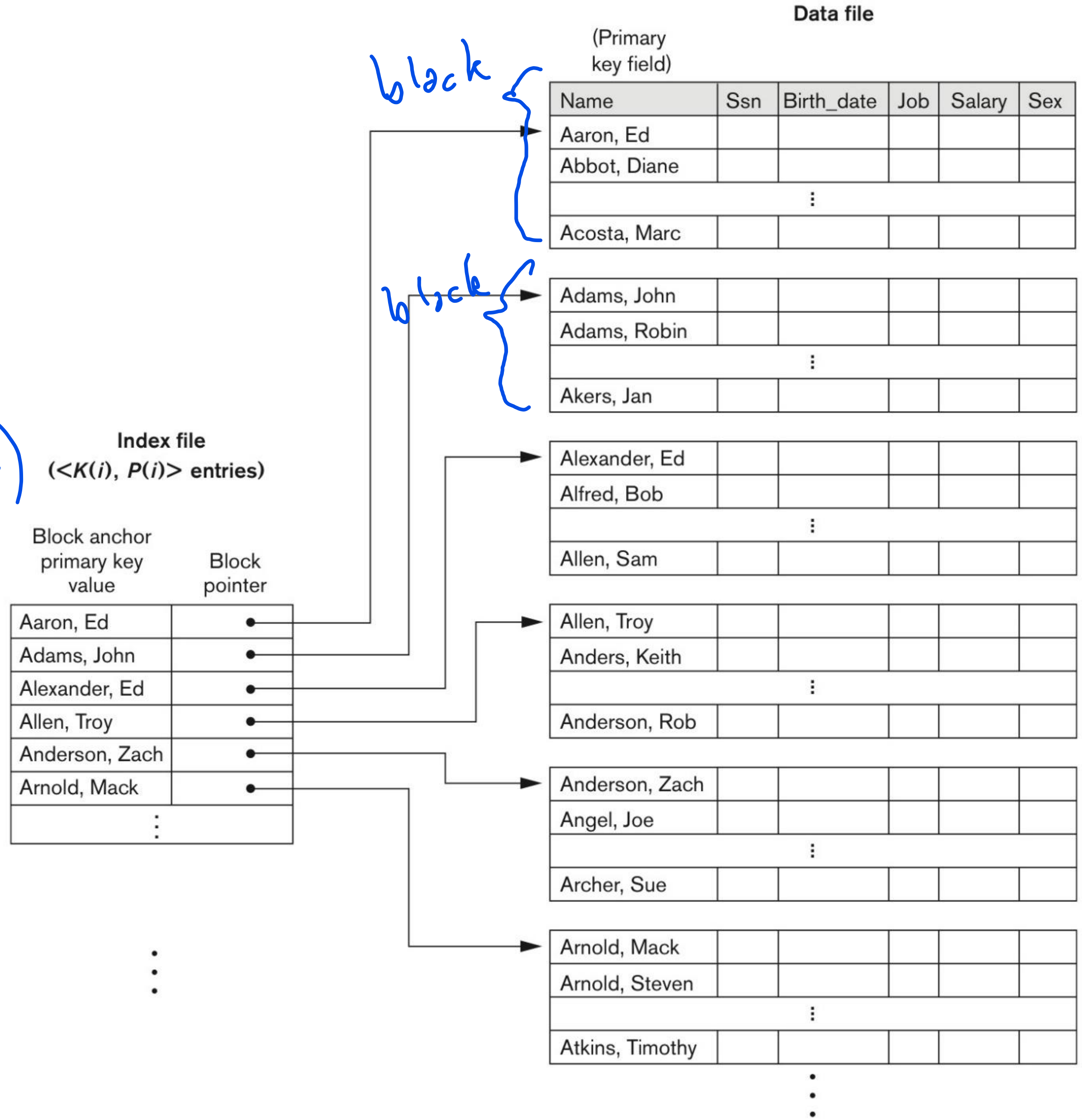
- **Index – auxiliary access structure** used to **speed up** the **retrieval** of the **records**.
  - Index structures are **additional files** on disk that provide **secondary access** path to data files.
- **Indexes** are **comprised** of two fields: **indexing attribute** & **storage block pointer**.
  - Any attribute can be indexed.
  - Multiple indexes on different attributes.
  - Indexes on multiple attributes.
- **Indexing structures** are similar to **indexing** in **books**:
  - Important terms = indexing attributes.
  - Page numbers = pointers to file blocks.
- **Types** of indexes:
  - **Single-level** indexes.
    - **Primary** indexes, **clustering** indexes & **secondary** indexes.
  - **Multi-level** indexes.

PRIMARY INDEXES (1)

- **Primary indexes** are used when **data file** is **ordered** by **primary key** attributes.
- **Primary index** is an **ordered file** with two fields:
  - **Primary key** *K(i).*
    - Indexing attribute.
  - **Pointer** to **disk block** *P(i).*
  - **Examples:**
    - <K(1) = (Aaron, Ed), P(1) = address of block 1>
    - <K(2) = (Adams, John), P(2) = address of block 2>
- **Primary index file** contains **one index** entry for **each block** in the **data file**.
  - **First record** of each block is an **anchor record**.
- Two **types** of **primary indexes**:
  - **Dense indexes**.
    - Index entry for every search key value in the data file.
  - **Sparse indexes**.
    - Entries for only some search values.

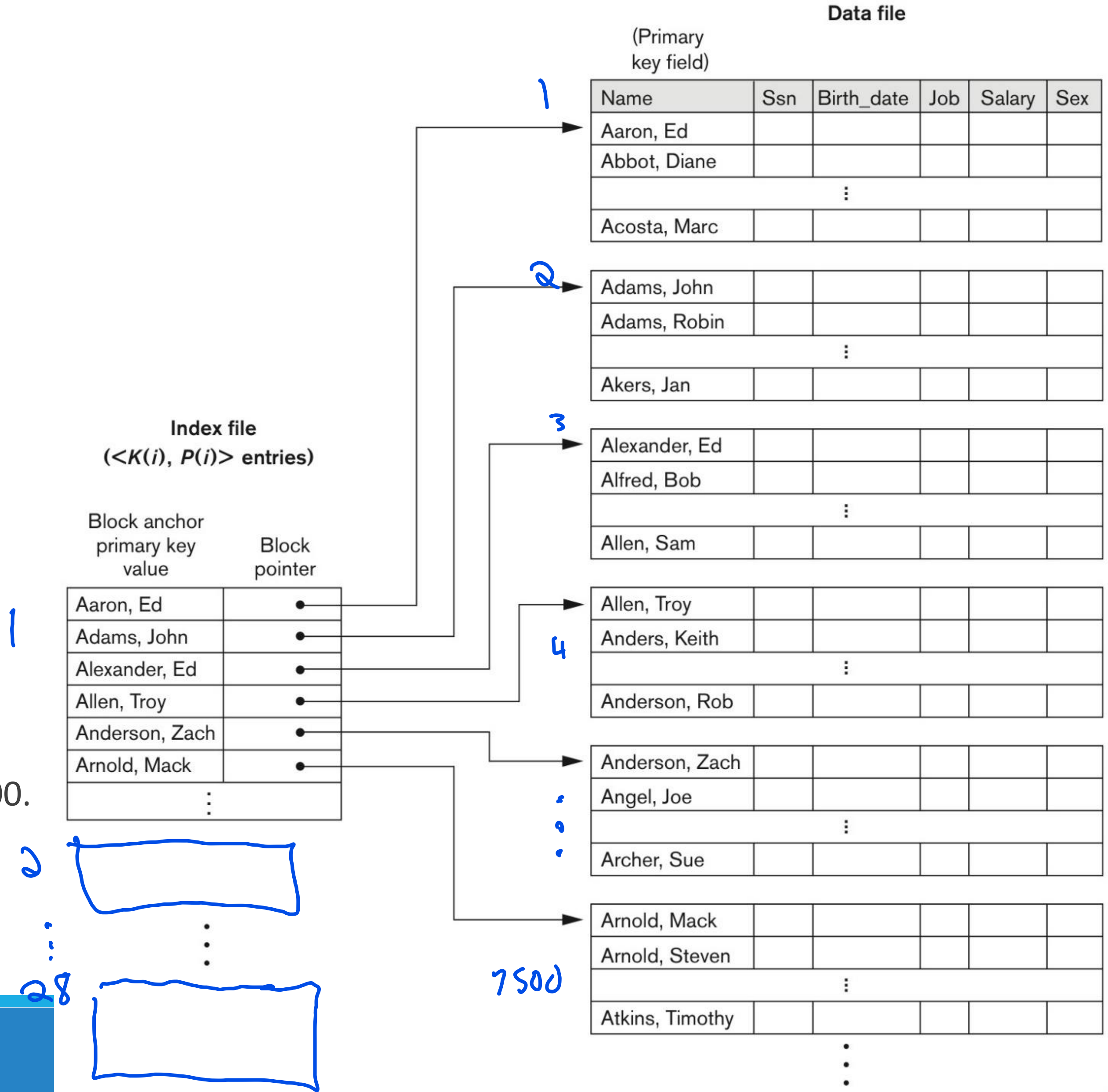
PRIMARY INDEXES (2)

- **Advantage**:
  - **Faster** records **search** & **retrieval**.
    - Only ($\log_2 b$ + 1) accesses.
- **Disadvantage**:
  - **Insertions** & **deletions** of records are **complex**.
    - Move records around & change index values.
- **Example:** (no index file, just searching Data file)
  - Ordered file with **r** = 300 000 records.
  - Disk block size **B** = 4096 bytes.
  - Record length **R** = 100 bytes.
  - Blocking factor **bfr** = $\lfloor(B/R)\rfloor = \lfloor(4096/100)\rfloor$ = 40 records per block.
  - Number of blocks needed **b** = $\lceil(r/bfr)\rceil = \lceil(300000/40)\rceil$ = 7500 blocks.
  - Binary search takes $\lceil\log_2 b\rceil = \lceil\log_2 7500\rceil$ = 13 block accesses.



PRIMARY INDEXES (2)

- **Advantage**:
  - **Faster** records **search** & **retrieval**.
    - Only ($\log_2 b + 1$) accesses.
- **Disadvantage**:
  - **Insertions** & **deletions** of records are **complex**.
    - Move records around & change index values.
- **Example (cont.)**: (index file used)
  - Ordering key of file is **V** = 9 bytes long.
  - Block pointer is **P** = 6 bytes long.
  - Size of each index entry is $\mathbf{R_i}$ = (9 + 6) = 15 bytes.
  - Blocking factor for index $\mathbf{bfr_i} = \lfloor (B/R_i) \rfloor = \lfloor (4096/15) \rfloor$ = 273 entries per block.
  - Total number of indexes $\mathbf{r_i}$ = total number of blocks in data file = 7500.
  - Number of index blocks $\mathbf{b_i} = \lceil (r_i / bfr_i) \rceil = \lceil (7500/273) \rceil$ = 28 blocks.
  - Binary search takes $\lceil \log_2 b_i \rceil = \lceil \log_2 28 \rceil$ = 5 block accesses.
  - Total number of accesses to search for record = 6 accesses (bin +1).

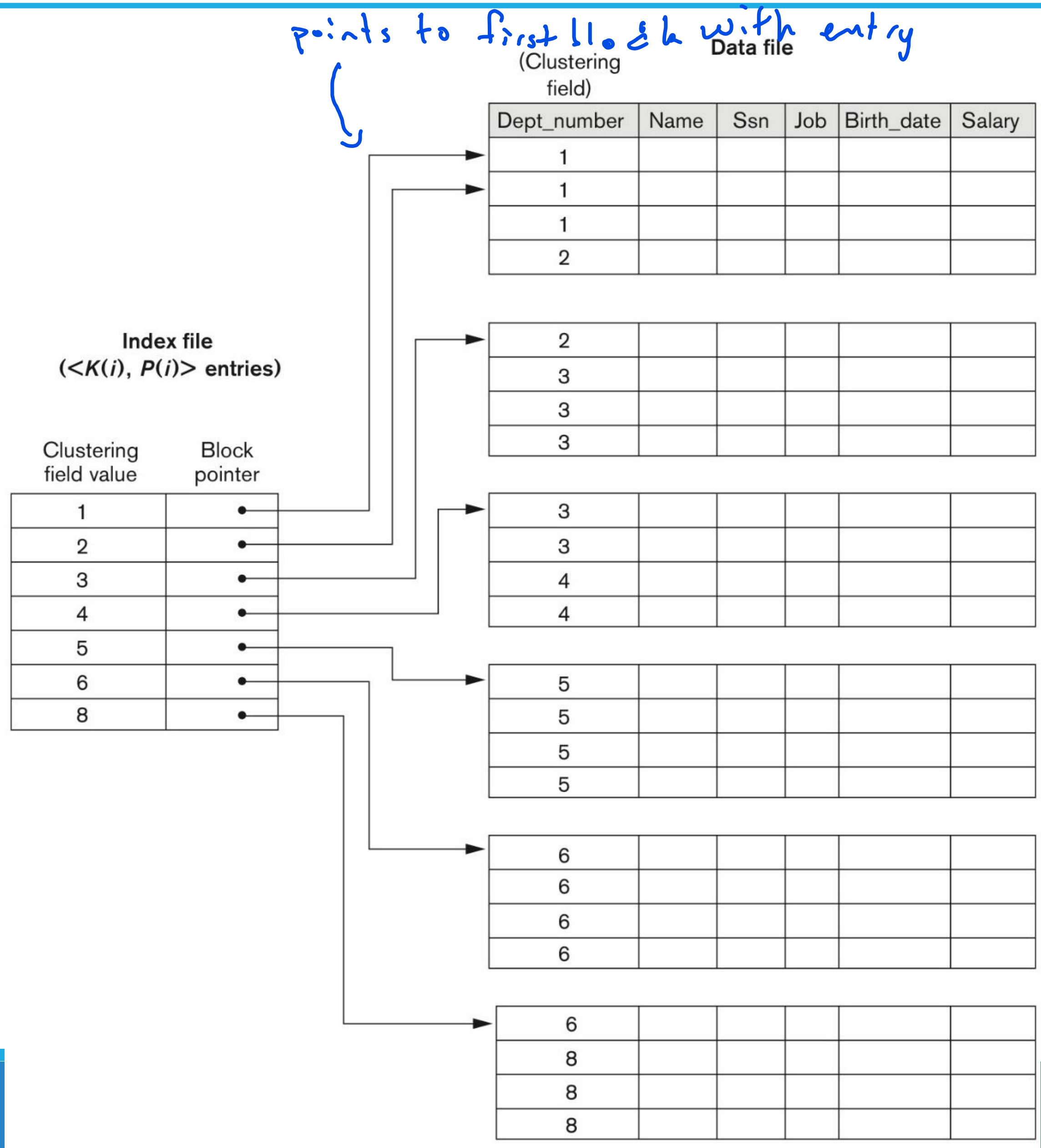


CLUSTERING INDEXES (1)

- **Clustering index** is used when **data file** is **ordered** on a **non-key attribute**.
  - **Ordering** attribute is **not unique**.
- **Clustering index** is an **ordered** file with two fields:
  - **Clustering attribute** *K(i).*
    - Indexing attribute.
  - **Pointer** to **disk block** *P(i).*
    - Points to the **first block** that **has** a record with the value of a clustering field.
- **Clustering index file** contains **one index** entry for **each distinct value** of clustering attribute.
  - **Sparse** index.

CLUSTERING INDEXES (2)

- **Advantage**:
  - **Faster** records **search** & **retrieval**.
    - Only ($\log_2 b$ + 1) accesses.
- **Disadvantage**:
  - **Insertions** & **deletions** of records are **complex**.
    - Move records around & change index values.



SECONDARY INDEXES (1)

- **Secondary index** is defined over a **non-ordering attribute(s)** of a record.
  - Provides **secondary access** when **primary access** already **exists**.
- **Secondary index** can be **defined** over:
  - **Candidate key attribute** (*unique*).
    - Dense index.
  - **Non-key attribute** (*duplicated*).
    - Sparse index.
- **Secondary index** is an **ordered file** with two fields:
  - **Non-ordering attribute** *K(i).*
    - Indexing attribute.
  - **Pointer** to **disk block** or **record** *P(i).*

SECONDARY INDEXES (2)

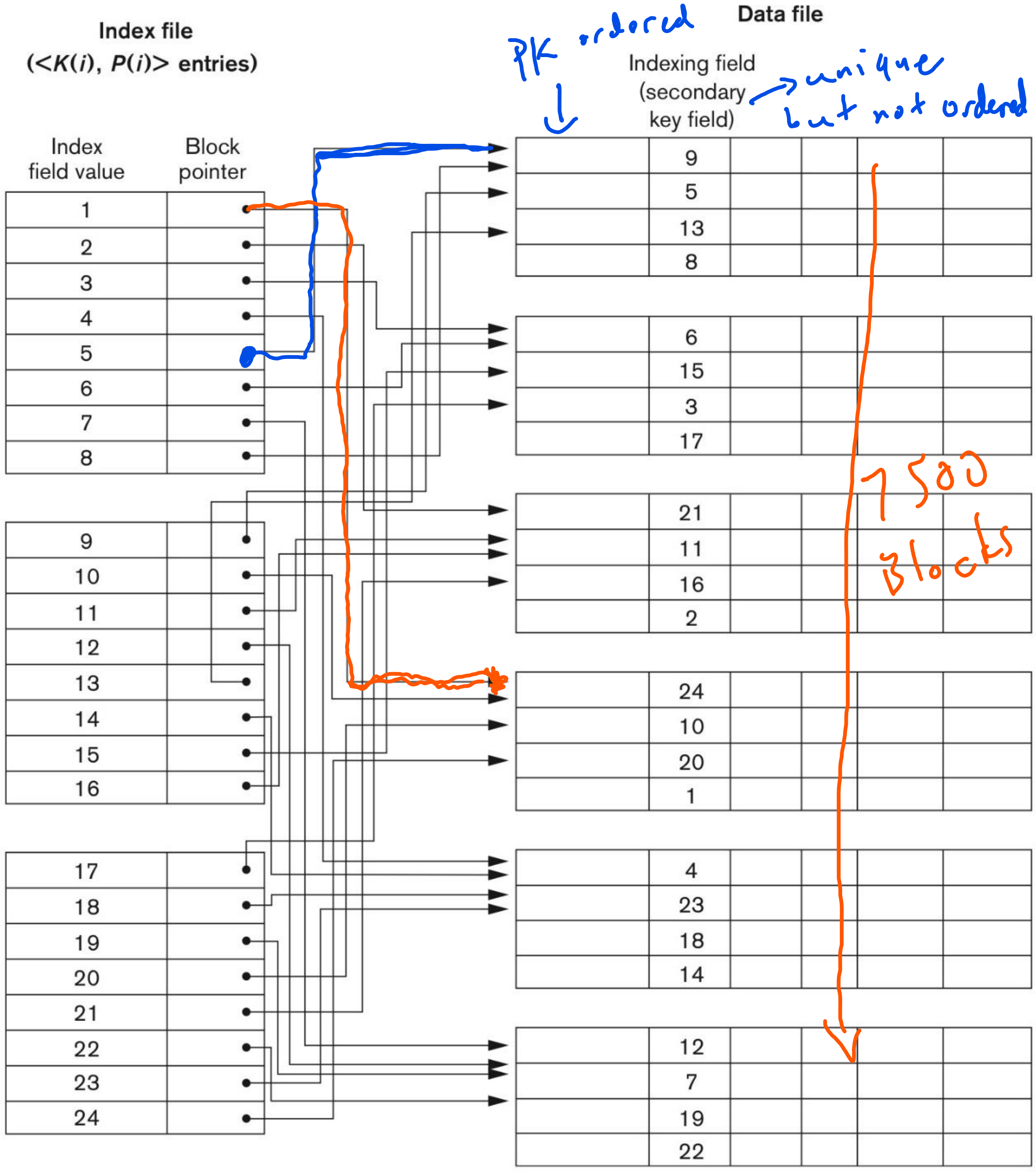
- **Secondary index** provides **logical ordering** of the records by **indexing attribute**.
- **Secondary index** needs **more storage** space and **longer search** time compared to primary index.
  - Contains **larger number** of entries.

Index file is ordered

- **Example:** (no index file)
  - Same settings as in primary index.
  - **r** = 300000 records.
  - **R** = 100 bytes.
  - **B** = 4096 bytes.
  - **b** = 7500 blocks.
  - If no primary index → linear search on 7500 blocks = 3750 accesses.

    r　　n/2
  - Secondary indexing on non-key value with **V** = 9 bytes. (index file)
  - **P** = 6 bytes.
  - $\mathbf{R_i}$ = (9 + 6) = 15 bytes.
  - $\mathbf{bfr_i}$ = 273 entries per block.

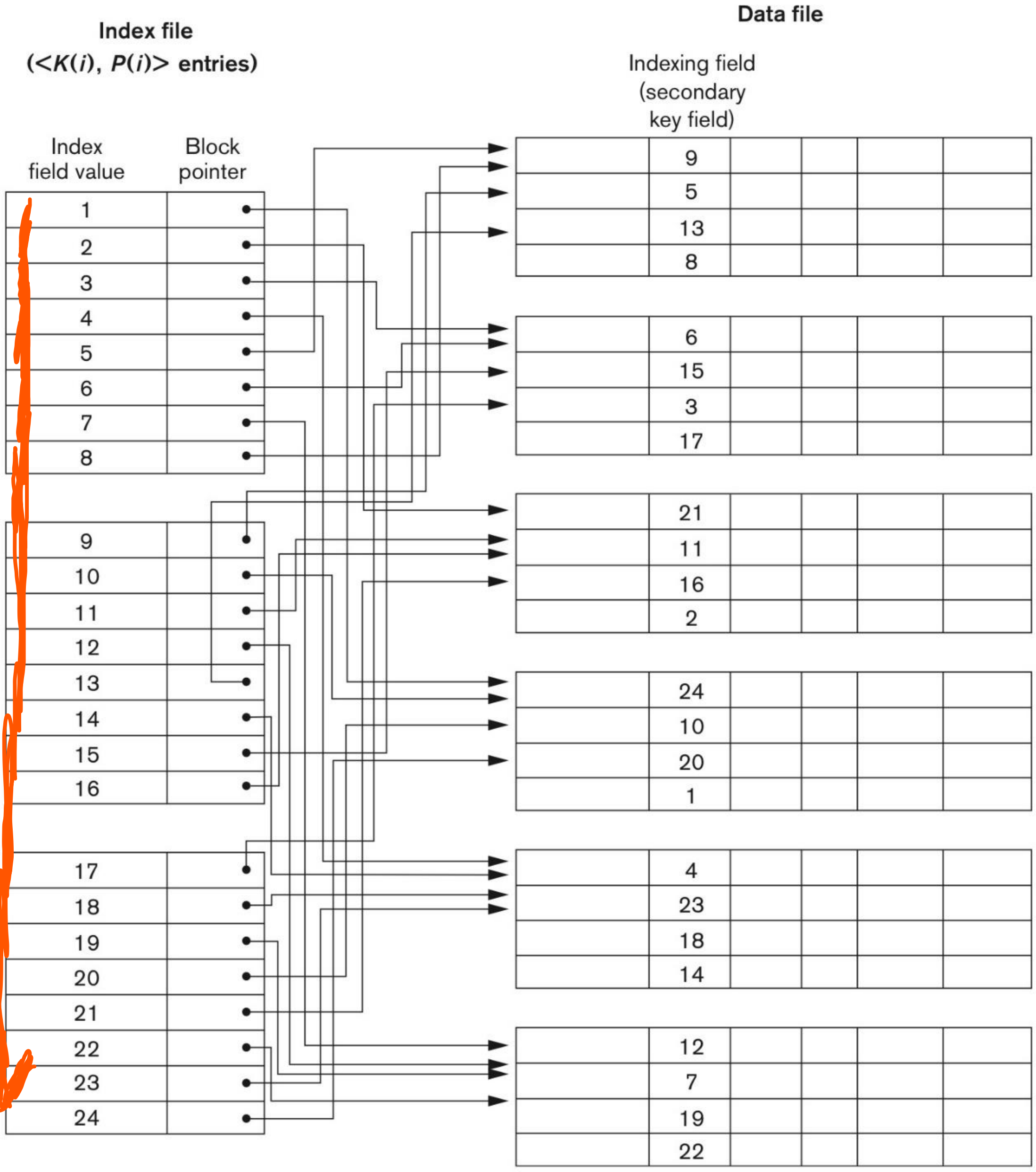
con't on next slide



SECONDARY INDEXES (2)

- **Secondary index** provides **logical ordering** of the records by **indexing attribute**.
- **Secondary index** needs **more storage** space and **longer search** time compared to primary index.
  - Contains **larger number** of entries.
- **Example (cont.):**
  - Total number of index entries $\mathbf{r_i}$ = total number of records = 300 000.
  - Number of blocks needed for index $\mathbf{b_i} = \lceil (r_i / bfr_i) \rceil = \lceil (300000/273) \rceil$ = 1099 blocks.
  - Binary search takes $\lceil \log_2 b_i \rceil = \lceil \log_2 1099 \rceil$ = 11 block accesses.
  - Total number of accesses to search for record = 12 accesses (bin +1).
    - Better than 3750 accesses with no secondary index.

1099 Blocks



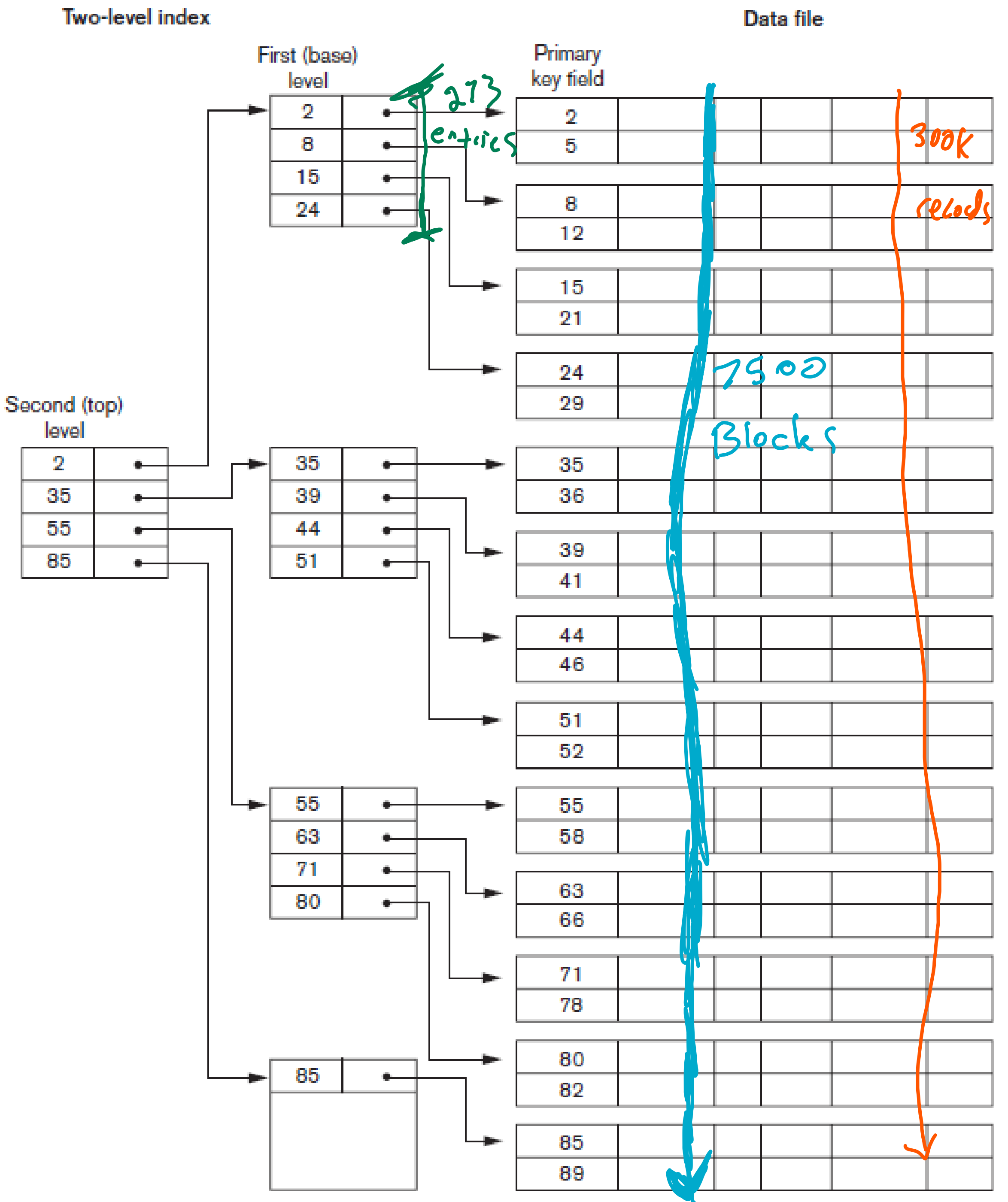
MULTILEVEL INDEXES (1)

- **Multilevel indexes** are aimed to **reduce search space** when **searching** through **index files**.
  - Index files are **sorted** $\rightarrow$ **binary search** is used to search though indexes $\rightarrow$ $\mathbf{log_2 b}$ accesses.
  - Applying **indexing** on **index files** $\rightarrow$ $\mathbf{log_n b}$ accesses, where $n >> 2$.
- **Multilevel indexing** forms a **hierarchy** structure (tree-like):
  - **Index file.**
    - First (base) level of a multilevel index.

      } closest to data file
  - **Second level.**
    - Primary index to the first level.
  - **Third level.**
    - Primary index to the second level.
    - If no more levels – considered **top index level**.
- **Multilevel indexing** is referred by **Indexed Sequential Access Method** (**ISAM**) organization.

MULTILEVEL INDEXES (2)

- **Multilevel indexes** considerably **reduce** number of **accesses** needed to **retrieve** a record.
- **Insertion** & **deletion** of records still involves **complex** operations.
  - Solution – **dynamic multilevel indexing**.
- **Example:**
  - Same settings as in secondary index.
  - **r** = 300000 records.
  - **R** = 100 bytes.
  - **B** = 4096 bytes.
  - **b** = 7500 blocks.
  - **P** = 6 bytes. **V** = 9 bytes.
  - $\mathbf{R_i}$ = (9 + 6) = 15 bytes.
  - $\mathbf{bfr_i}$ = 273 entries per block. Factor **n** in $\log_n b$.

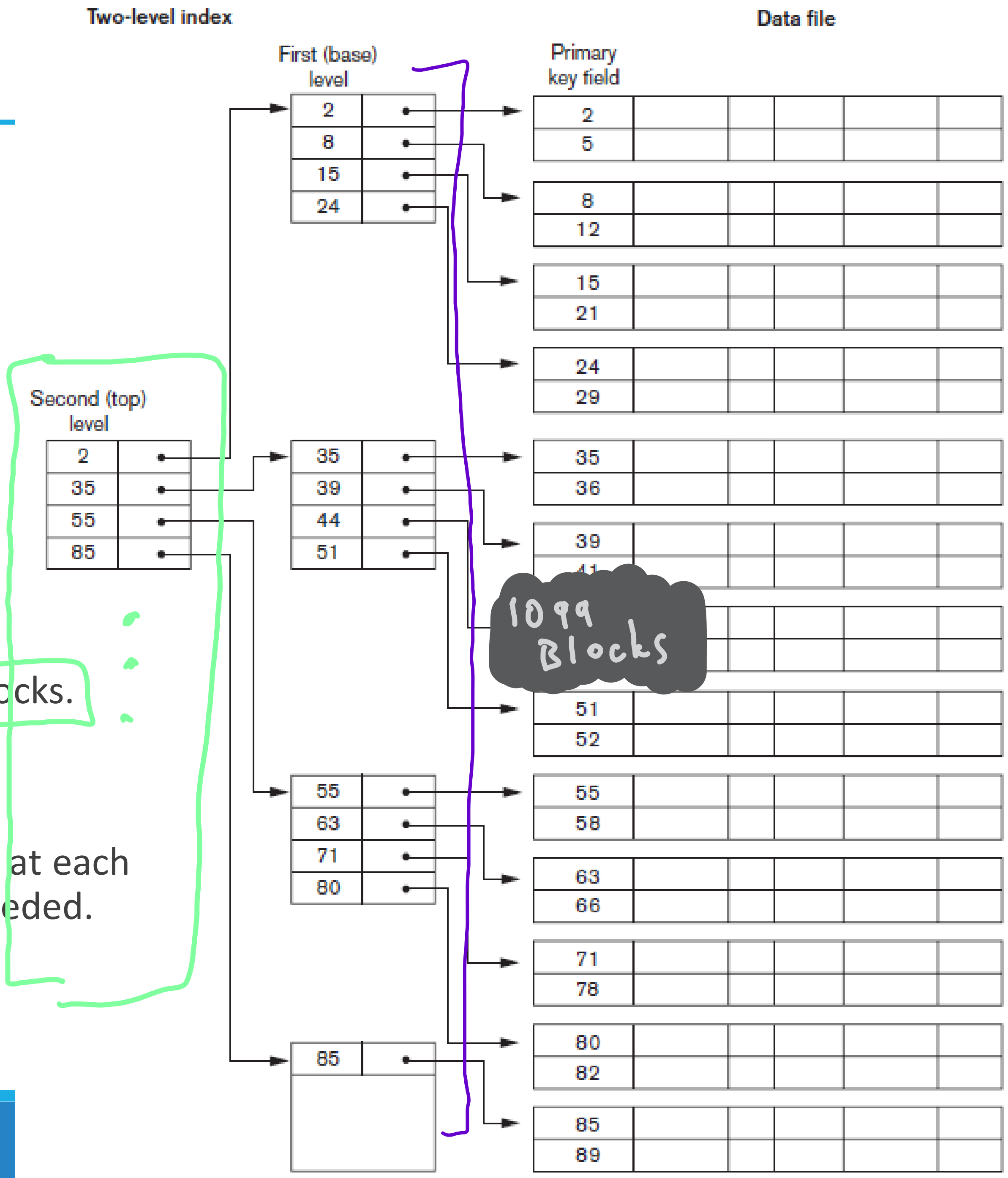


MULTILEVEL INDEXES (2)

- **Multilevel indexes** considerably **reduce** number of **accesses** needed to **retrieve** a record.
- **Insertion** & **deletion** of records still involves **complex** operations.
  - Solution – **dynamic multilevel indexing**.
- **Example (cont.):**
  - Total number of index entries $\mathbf{r_i}$ = 300 000.
  - Number of blocks needed for first-level index $\mathbf{b_1}$ = 1099 blocks.
  - Number of second-level blocks $\mathbf{b_2} = \lceil b_1/n \rceil = \lceil 1099/273 \rceil$ = 5 blocks.
  - Number of third-level blocks $\mathbf{b_3} = \lceil b_2/n \rceil = \lceil 5/273 \rceil$ = 1 block.
  - Hence, third level is the **top level** and number of levels **t** = 3.
  - Accessing record by multilevel index requires accessing a block at each level plus one block from data file → 3 + 1 = 4 total accesses needed.

not pictured

5 blocks



SUMMARY

- Fixed-length vs variable-length records.
- Unspanned vs spanned records.
- Sorted, unsorted & hashed files.
- Hashing techniques.
- Primary indexes.
- Clustering indexes.
- Secondary indexes.
- Multilevel indexes.