Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 2

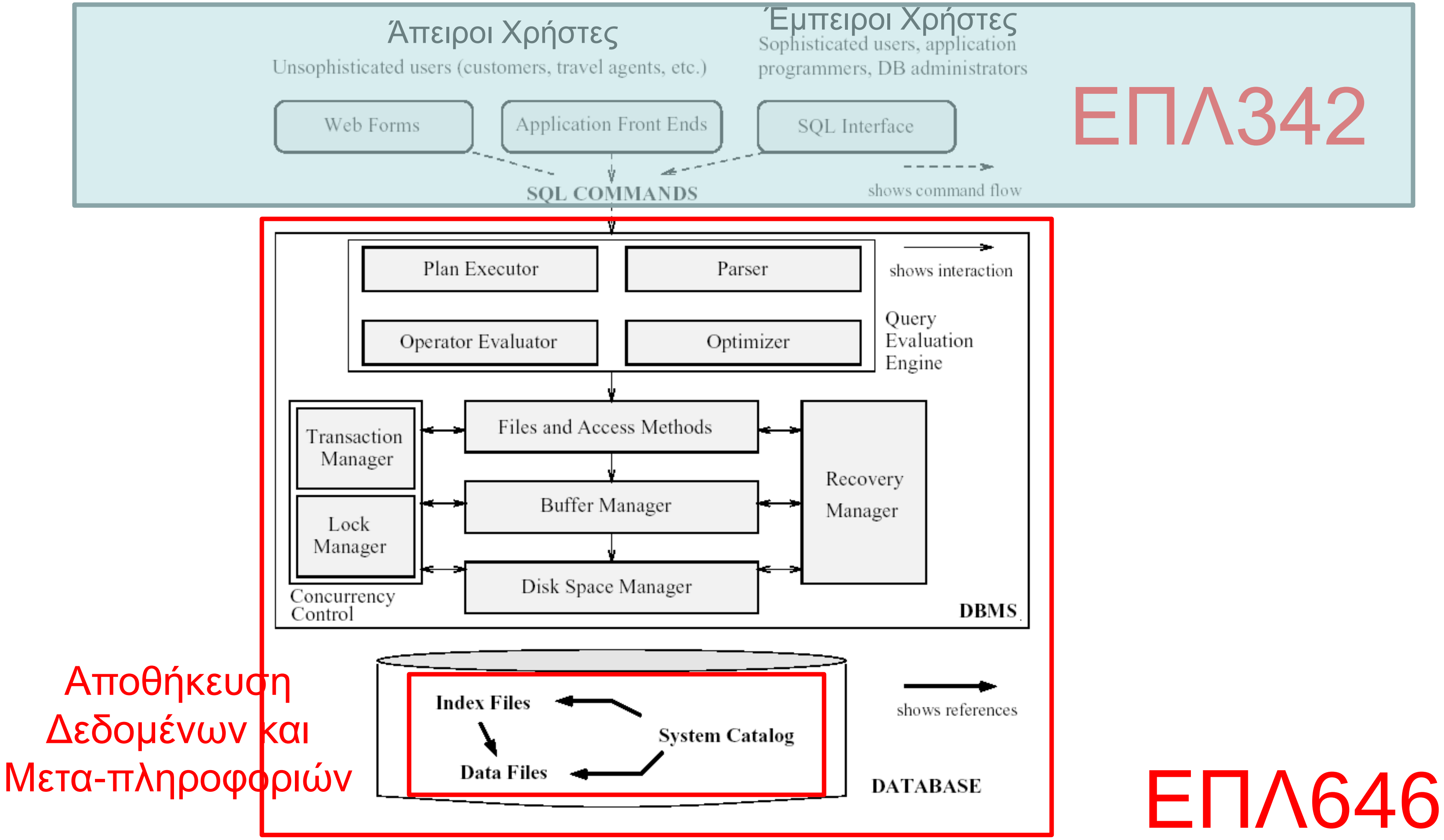
Storage I: Storage and Indexing

Chap. 8.1-8.5: Ramakr. & Gehrke

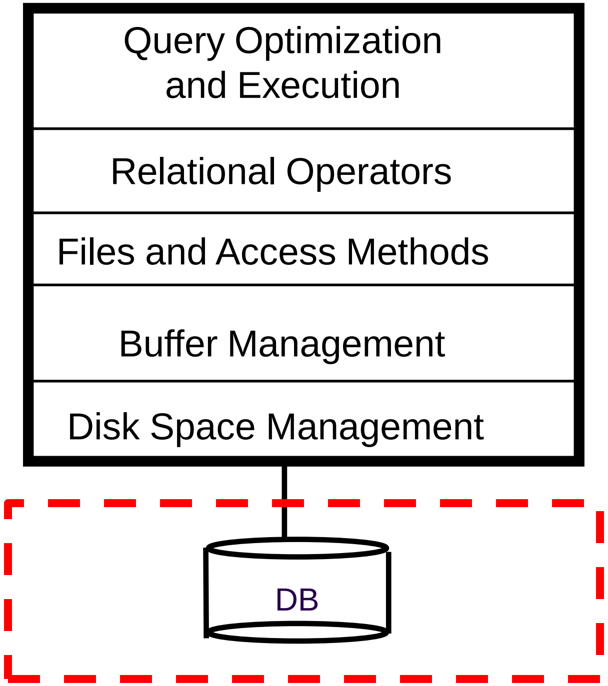
*** exclude 8.4.5-8.4.6**

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl646

ΕΠΛ646: Ενότητα Α
Εσωτερική Λειτουργία ενός RDBMS



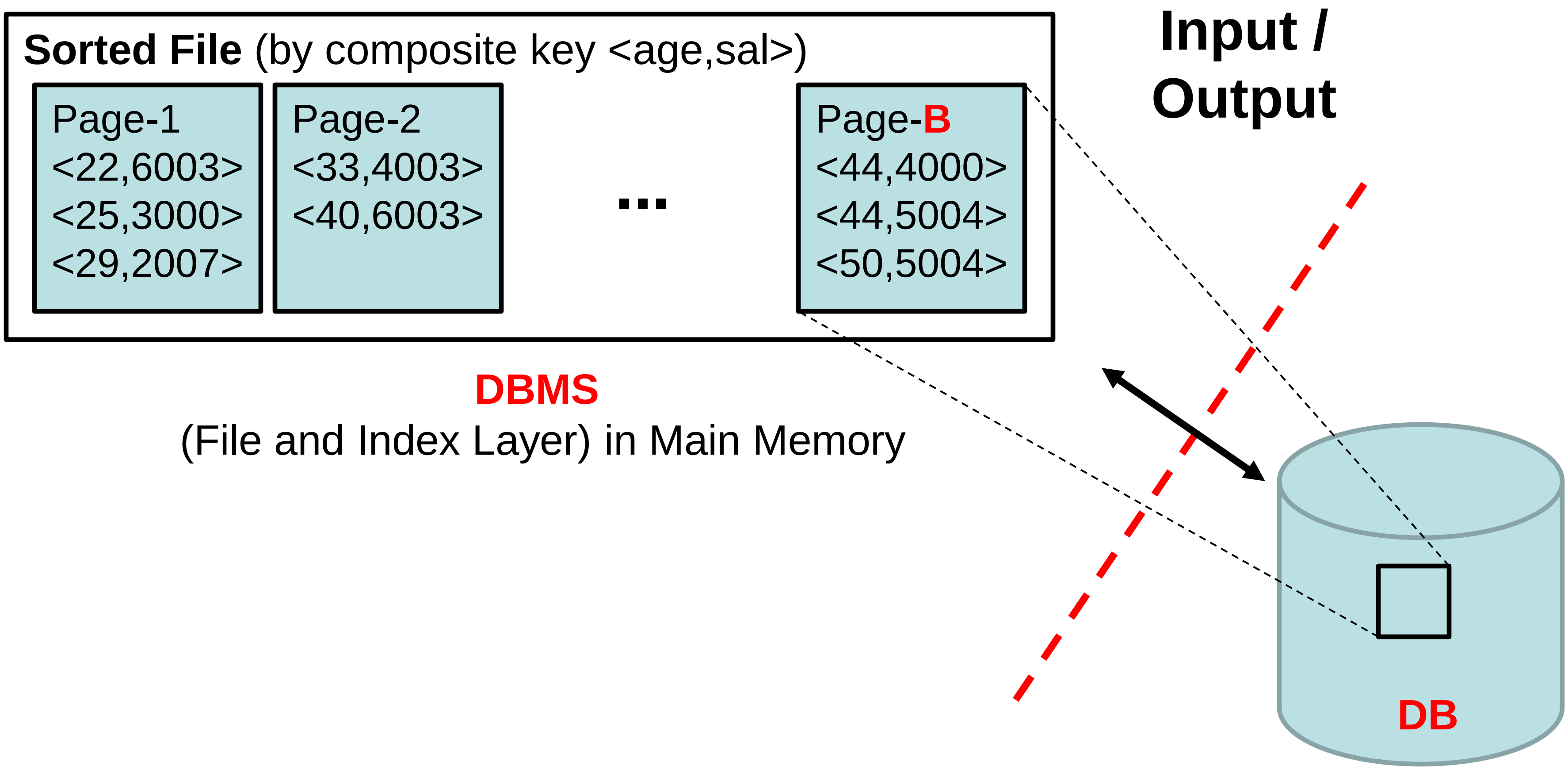
Context of next slides



Data on External Storage
(Δεδομένα στη Δευτερεύουσα Μνήμη)

- A DBMS stores **vast amounts** of data and the data has to **persists** across program executions.
- Therefore, data is stored on **external storage** and **fetched** into **main memory** as needed for processing.
- The unit of information that is read and written to a disk is called **Page (Σελίδα)**, e.g., 4KB ή 8KB
- Higher layers of the DBMS view these pages as unified **Files (Αρχείο)** and can read/write **Records (Εγγραφές, Πλειάδες)** to these files.
  - Consider a data record (id:4B, name:28B) and a 4096B (4KB) page size. That would yield ~128 records / page (some bytes go to headers and other auxiliary structures).
- **What is the basic performance cost in a DBMS?**
  - **I/O (Input/Output):** # **pages read/write** for a given operation.
  - **Complexity** of algorithms in DBMSs is expressed in I/Os

Data on External Storage
(Δεδομένα στη Δευτερεύουσα Μνήμη)



Storage Mediums (Μέσα Αποθήκευσης)

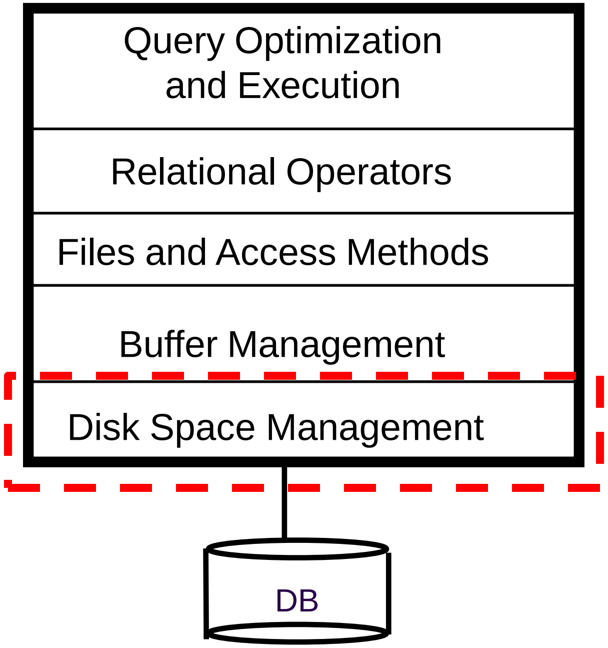
- Disks: Can retrieve a **random** page at a **fixed cost**
  - But reading several **consecutive pages** is much cheaper than reading them in **random order**

Main focus of DBMSs

- Tapes: Can only read pages **in sequence**
  - **Cheaper** than disks; used for **archiving** (αρχειοθέτηση)
- Flash Memory (Solid State Disks): Reading data at the speed of main memory, writing is slower.
  - More **expensive** than disks; used for applications with read workloads that require **fast random** accesses.



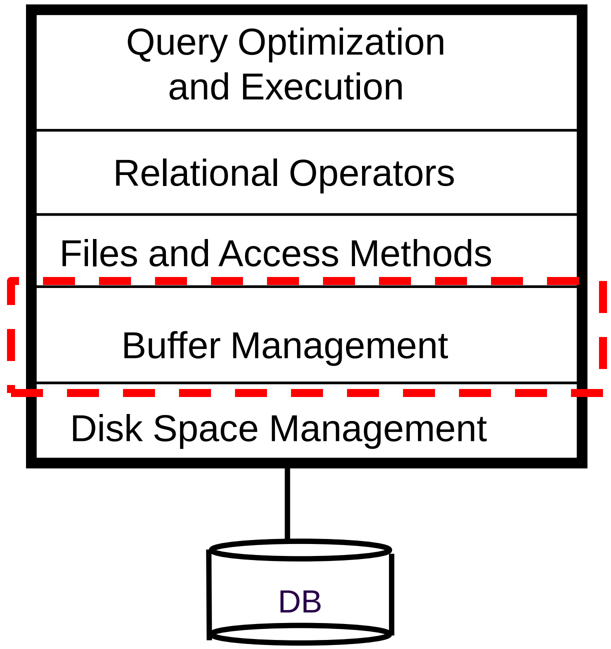
Context of next slides



Disk Space Manager (DSM)
(Διαχειριστής Χώρου Δίσκου)

- **DSM:** Supports the concept of a **page** as a unit of data and provides commands to **allocate/deallocate**, **read/write** a page to external storage.
  - Size of Page == Size of Disk Block, in order to support read/write operations in one I/O operation.
  - The higher layers in the DB architecture (i.e., the **Buffer Manager**) interact directly with the DSM.
- Other Duties: **Keep track of Free Blocks**.
  - Initially a DB is stored on consecutive **disk blocks** (when it acts in its own partition) or inside a **file** (when it is stored inside an Operation System file).
  - Subsequent **deletions** might easily create **“holes”** in that sequence (either file or disk), thus the DSM needs to track the free pages.

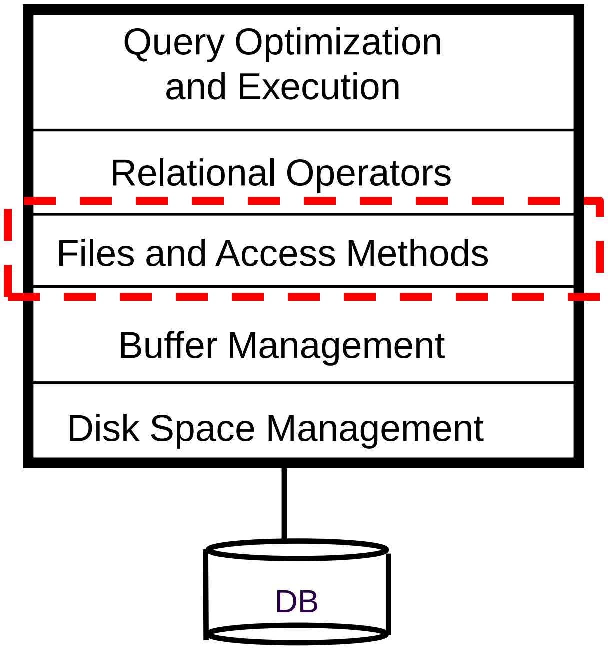
Context of next slides



Buffer Manager (BM)
(Διαχειριστής Κρυφής Μνήμης)

- **BM:** Subsystem that is responsible for **loading** pages from **external storage** to the main memory **buffer pool**
  - File & Index layers make calls to the buffer manager.
  - **Idea:** Keep as many blocks (pages) in memory as possible to reduce disk accesses.
- **Replacement Policies**
  - **e.g., LRU** (Least-Recently-Used pages … are discarded => the oldest are discarded first), **MRU** (Most-Recently-Used, the newest are discarded first), **LFU** (Least-Frequently-Used)
- **Prefetching or Double Buffering**
  - **Idea:** speed-up access by pre-loading "future"-needed data
  - **Cons:** requires extra main memory; no help if requests are random

Context of next slides



Alternative File Organizations (Εναλλακτική Οργάνωση Αρχείων)

- **File organization (Οργάνωση Αρχείου):** Method for **arranging** a collection of records and supporting the concept of a **file.**
- In all file organizations the records are accessed by their respective **RecordID**
  - Note that a **Record ID (RID)** usually has the following structure **(PageID, SlotID)**, where **SlotID** defines the offset : i) inside **PageID** at which RID begins; or ii) inside the Slot Directory that resides within page PageID (explained in next lectures)*Page i*

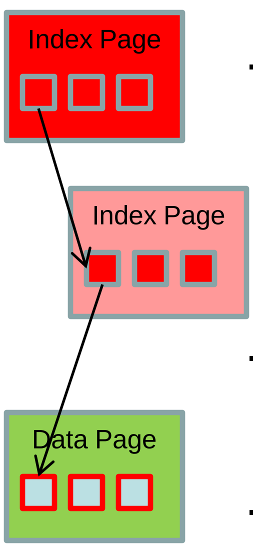
- **Basic Questions**
  - How to store data inside **data records** (fixed-length records vs. variable-length records)?
  - How to store data-records **inside a file** (heap file, sorted file, indexed file)?
  - How to make a certain File Organization more powerful by complementing them with an Index?

File Organization Types (Τύποι Οργάνωσης Αρχείων)

- Heap files (Αρχεία Σωρού): Suitable when typical access is a **file scan (σάρωση αρχείου)** retrieving **ALL** records
  - Suitable for queries like "SELECT * FROM Employees;"
- Sorted Files (Ταξινομημένα Αρχεία): Best if records must be retrieved in some order, or only a **`range' (διάστημα)** of records is needed.
  - Suitable for queries like "SELECT * FROM Employees WHERE 20<age and age<30;"
- **Each file organization makes certain operations efficient, but we are interested in supporting more than one operation!**
- To deal with such situations the DBMS builds one or more **indexes**.
- An **index** on a file is designed to speed up operations that are **not efficiently** supported by the basic organization of that file.

Indexes (Access Methods) (Ευρετήρια Δευτερεύουσας Μνήμης)

- An *index* is a data structure that has **index records** which **point to** certain **data records**.
- An index can **optimize** certain kinds of retrieval operations (depending on the index).
- **Definitions**

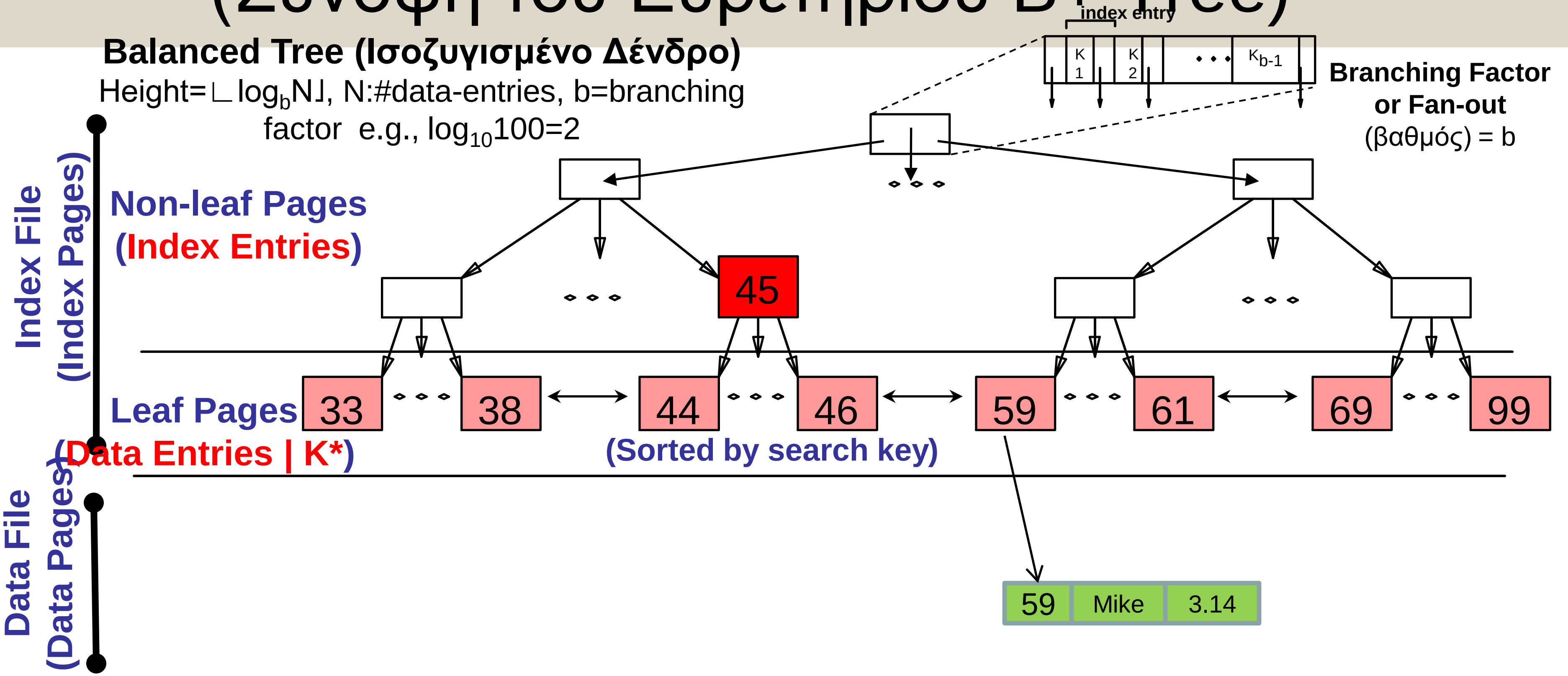


  - **Index Page (Σελίδες Ευρετηρίου) vs. Data Pages (Σελίδες Δεδομένων):** Index Pages store index records to data records. Both reside on disk because we might have many of these pages!
  - **Data Record (Εγγραφή Δεδομένων):** Stores the actual data e.g., (59,Mike,3.14) .
  - **Index Record (Εγγραφή Ευρετηρίου):** Stores the RID of another index record or a data record.

B+ Tree Index Overview (Σύνοψη του Ευρετηρίου B+ Tree)

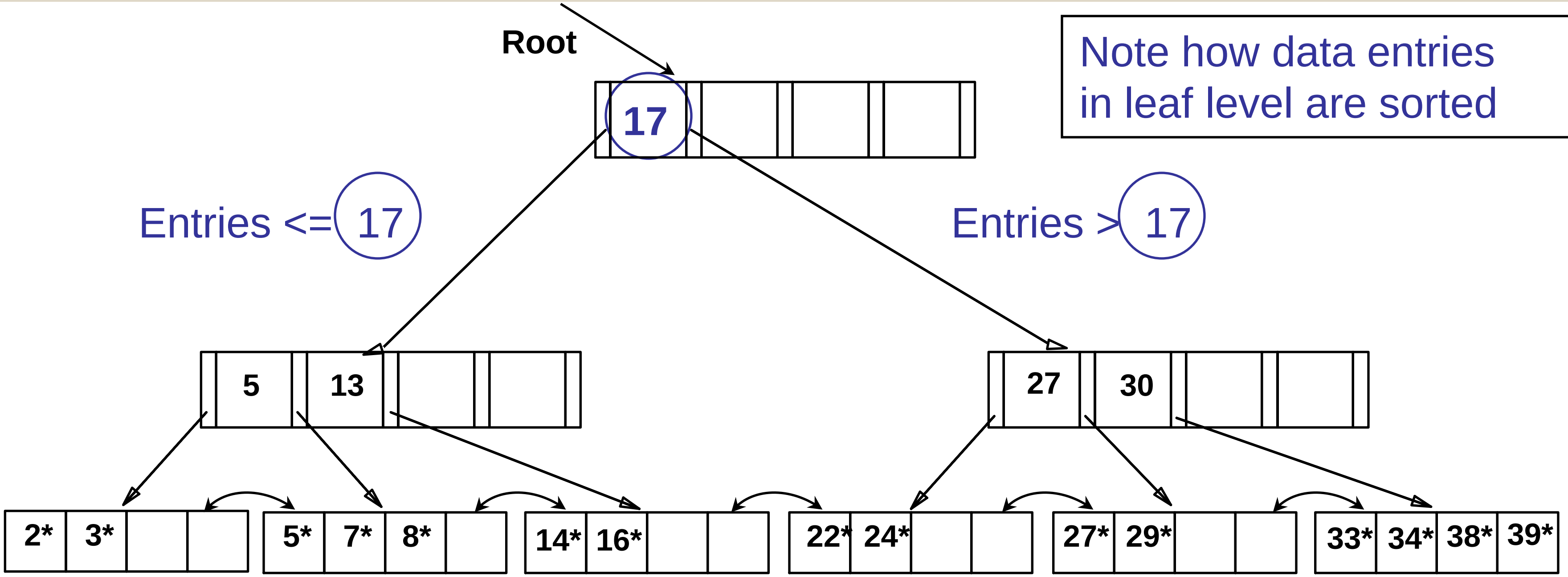
Balanced Tree (Ισοζυγισμένο Δένδρο)

Height=$\lfloor \log_b N \rfloor$, N:#data-entries, b=branching factor e.g., $\log_{10}100=2$



- Non-leaf pages have *index entries;* only used to direct searches.
- Leaf pages contain *data entries K**, and are chained (prev&next)
- The data records e.g., (59,Mike,3.14) could have been stored inside he respective data entry. Then the index file would be the same with the data file. ➔ **Index File Organization.**

Example B+ Tree
(Παράδειγμα Χρήσης B+ Tree)



- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

Structure of Data Entry k*
(Δομή της Καταχώρησης K*)

- In a **data entry k*** we can store:
  - **Alternative 1: <k> (**Key Value), or
  - **Alternative 2: <k, RID>**(Key Value, Data record with key value k)>, or
  - **Alternative 3:** $\mathbf{<k, [RID_1, RID_2, \ldots, RID_n]>}$**,** where $\boldsymbol{RID_i}$ is a data record with key value k.
- *Choice of alternative for data entries* is **orthogonal** to the indexing technique used to locate data entries with a given key value k.
  - **In particular, ANY of the above alternatives might be used with ANY index (hash or tree)**

Data Entry **k*** Examples (Παραδείγματα Καταχώρησης **k***)

- **Alternative 1: <k>**

Results in a Index File Organization!

59, Mike, 3.14	Index Data Entry

- **Alternative 2: <k, RID>**

59, RID#10	Index Data Entry

59	Mike	3.14	Data Record

RID#10

- **Alternative 3: <k, [RID,…,RID]>**

59, RID#10, RID#61, #RID82	Index Data Entry

59	Mike	3.14

RID#10

59	Chris	33.14

RID#61

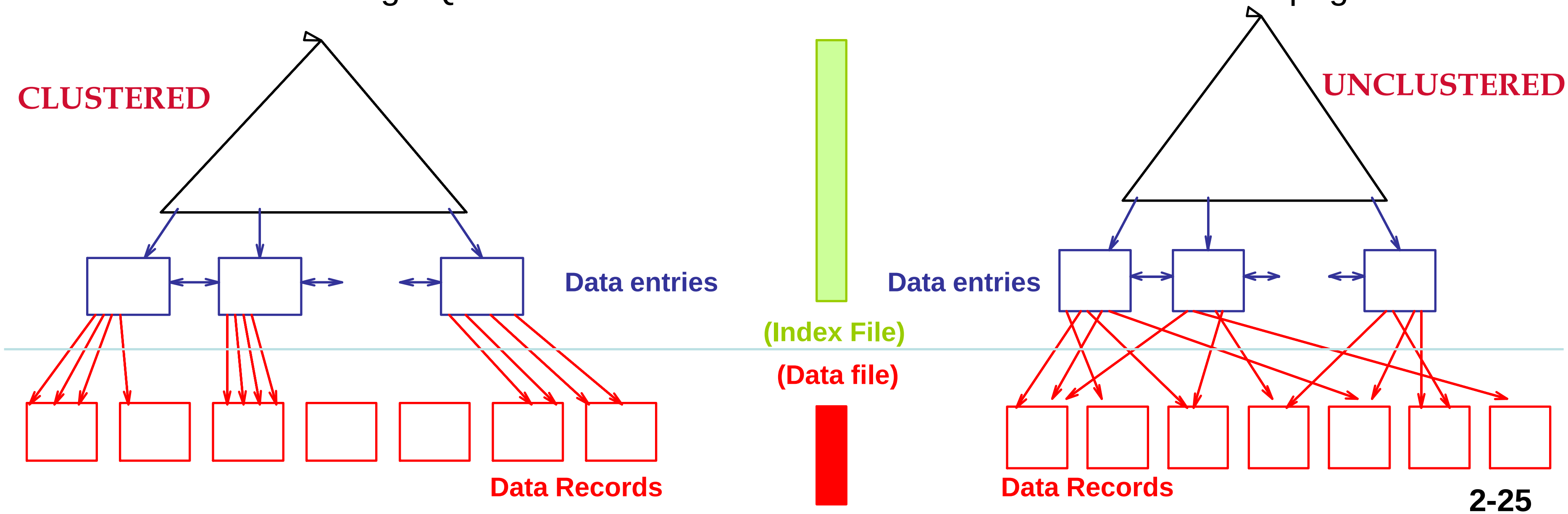
59	Jim	53.14

RID#82

Data Record

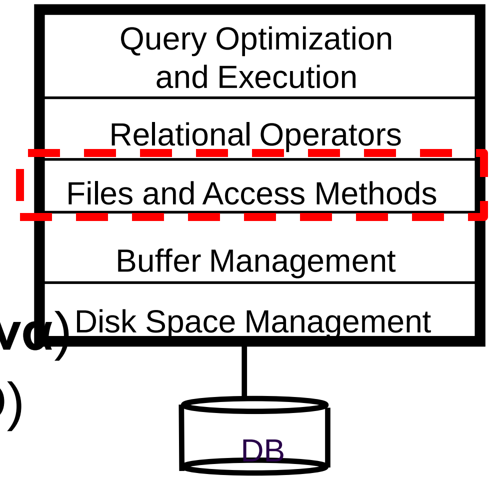
Clustered vs. Unclustered Indexes
(Ομαδοποιημένα vs. Μη-Ομαδοποιημένα Ευρετήρια)

- *Clustered* Index (Ομαδοποιημένο Ευρετήριο): If **order (διάταξη)** of **data records** is the same as, or `close to', order of **data entries,** else called unclustered index.
- **Alternative 1** implies **clustered** (since datarec same as dateentry)
  - Faster Range Queries: Consecutive data records reside on the same page.
- **Alternatives 2,3** are usually **unclustered**.
  - Slower Range Queries: Consecutive data records reside on different pages



Lecture Outline
Overview of Storage and Indexing

- **Note:** The subsequent slides aim to ***qualitatively compare (ποιοτική σύγκριση)*** the file organization and indexes alternatives we introduced previously.
- 8.4) Comparison of File Organization
  – System and Cost Model (Μοντέλο Κόστους)
  – **Heap** Files, **Sorted** Files and **Clustered** Files (Αρχεία: **Σωρού**, **Ταξινομημένα**, **Ομαδοποιημένα**)
  – Comparison on I/O Costs (Σύγκριση Κόστους I/O)



- 8.5) Indexes and **Performance Tuning** (Ρύθμιση Επίδοσης)
  – Understanding the **Workload** (Εκτιμώντας τον Φόρτο Εργασίας)
  – Index **Specification** in SQL (Δήλωση Ευρετηρίων στην SQL)
  – **Index-Only Plans** (Πλάνα με Μόνο το Ευρετήριο)
  – Index Selection **Guidelines** (Οδηγίες Επιλογής Ευρετηρίων)

System Model
(Μοντέλο Συστήματος)

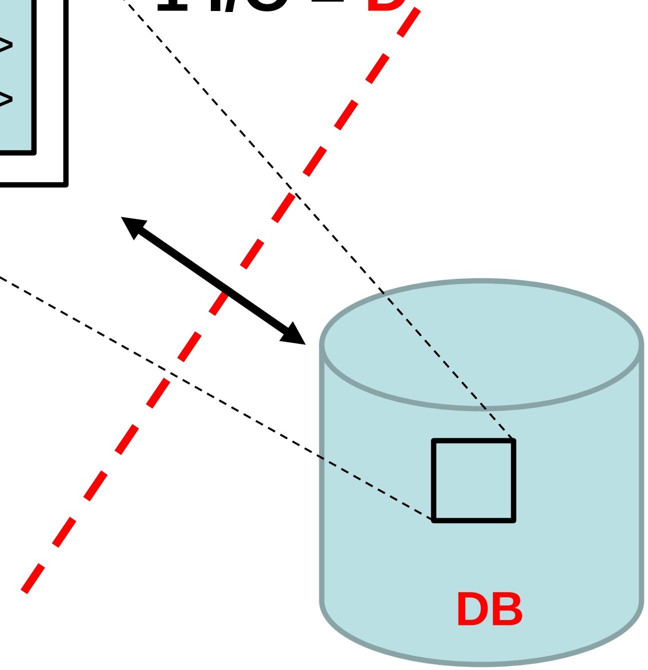
Sorted File (by composite key <age,sal>)

R

Page-1	Page-2	...	Page-**B**
<22,6003>	<33,4003>		<44,4000>
<25,3000>	<40,6003>		<44,5004>
<29,2007>			<50,5004>

Average Time for 1 I/O = D

DBMS
(File and Index Layer) in Main Memory



Assumptions: Average case analysis,
no processing costs, no prefetching

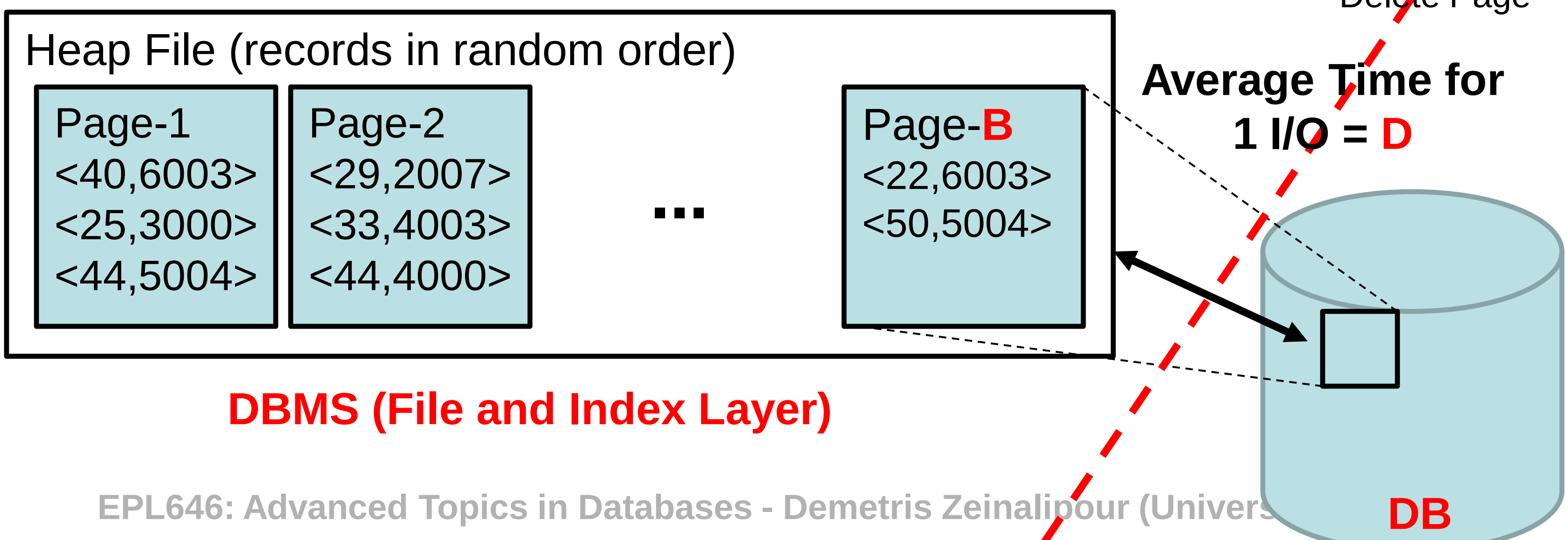
Operations to Compare
(Πράξεις που θα Συγκριθούν)

- **Scan (Σάρωση):** Fetch all records from disk
  - e.g., SELECT * FROM Employees;
- **Equality Selection (Επιλογή Ισότητας)**
  - e.g., SELECT * FROM Employees WHERE **age=33 AND sal=4003**;
- **Range selection (Επιλογή Διαστήματος)**
  - e.g., SELECT * FROM Employees WHERE **age BETWEEN 35 AND 45;**
  - e.g., SELECT * FROM Employees WHERE **35<age AND sal<=4000;**
  - ***But NOT:*** *SELECT * FROM Employees WHERE* ***sal>40;*** *(tree index is on age ☹)*
- **Insert a record (Εισαγωγή Εγγραφής)**
  - e.g., INSERT INTO Employees (age, sal) VALUES (45, 3000);
- **Delete a record (Διαγραφή Εγγραφής)**
  - e.g., DELETE FROM Employees WHERE age=45;

Heap File Analysis
(Ανάλυση Αρχείου Σωρού)

- **Heap File Assumptions**
  - Equality Selection on key <age,sal>
  - Equality Selection produces exactly 1 match.

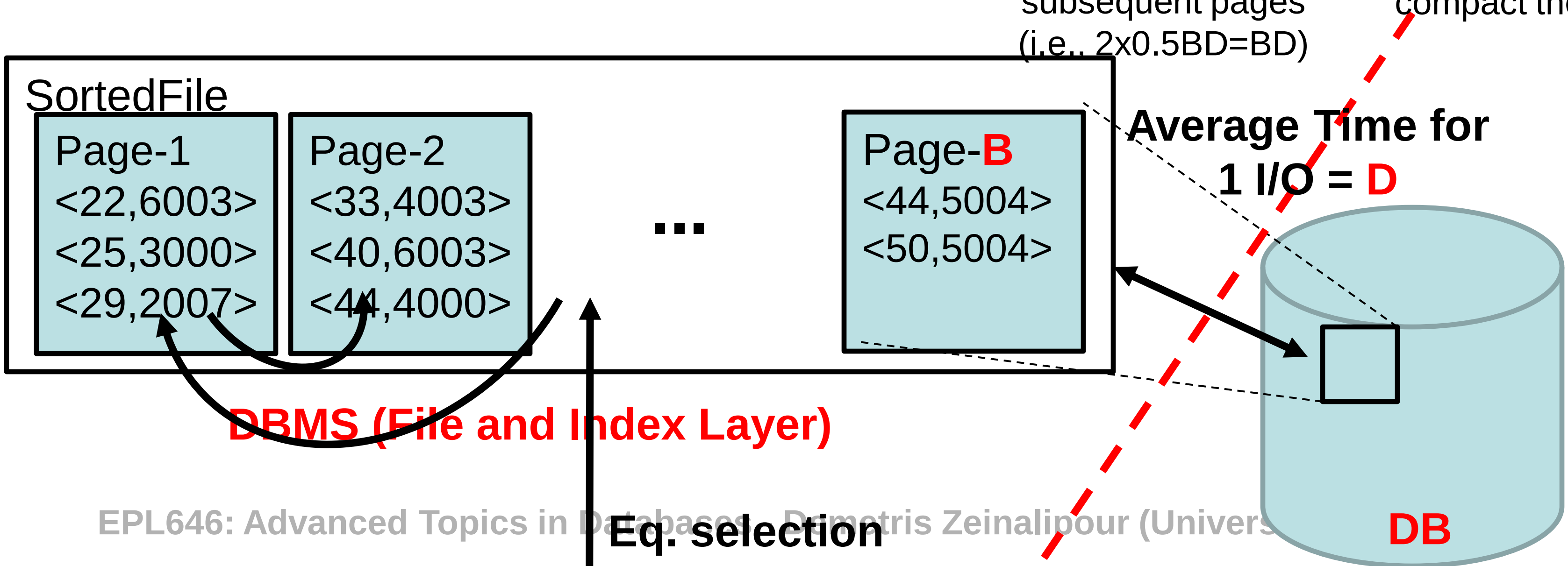
ScanAll	Eq. Selection	Range Selection	Insert	Delete
BD	0.5BD	BD	2D	0.5BD+ D
All records	On average we traverse ½ records	Traverse all to find pages in range	Read (last) PageB + Write PageB	Find Page + Delete Page



Sorted File Analysis (Ανάλυση Ταξινομημένου Αρχείου)

- **Sorted File Assumptions**
  - Files compacted after deletions (no holes in pages)

ScanAll	Eq. Selection	Range Selection	Insert	Delete
BD	$D\log_2 B$	$D(\log_2 B$ + #matches)	$D\log_2 B + BD$	$D\log_2 B + BD$
All records	Binary Search over B pages. Each I/O costs D	Binary Search for 1 tuple, then transfer rest qualifying pages	Binary Search for Correct Position + Shift (Read/Write) ½ subsequent pages (i.e., 2x0.5BD=BD)	Same as Insert but ½ pages are shifted back in order to compact the file



Understanding the Workload (Εκτιμώντας τον Φόρτο Εργασίας)

- **Workload (Φόρτος Εργασίας):** The typical mix of **i) Query (Select)** and ii) **Update (Insert/Delete/Update)** operations in a DBMS system.
- i) For each **query/update** in the **workload** :
  - Which types are involved (Select,Insert,Delete,Update)
  - Which **relations**/attributes**(σχέσεις, χαρακτηριστικά)** does it **access**?
  - Which **attributes** are involved in **selection/join** (επιλογή/ συνένωση) conditions? How **selective** are these conditions likely to be?
- **Selectivity (Επιλεκτικότητα της Συνθήκης):** The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

E.g., $\boldsymbol{\sigma_{age>40}}$**(EMPLOYEE)** returns 10 out of 1000 tuples. **Selectivity=1%**

Index Specification in SQL
Δήλωση Ευρετηρίου στη SQL

- The **SQL standard (up until SQL 2008)** does not include any statement for **creating/dropping** indexes.
- **However**, in practice **every major DBMS supports** such **indexes (access methods)** such as **Btrees, Hash, Rtrees, GIST.**

Example from the PostgreSQL DBMS

```
CREATE INDEX AgeSalIndex
    ON Employees (age, sal)
    USING BTREE
    WHERE sal > 3000
```

Choice of Indexes (Επιλογή των Ευρετηρίων)

- The DBA is usually confronted with several questions in regards to indexes:
  - **Which relations** should have indexes?
  - What **type** of index should we use? Clustered? Hash? Btree?
  - What **attribute(s)** should be the search key?
  - Should we build **several indexes**?

Index Selection Guidelines
(Οδηγίες Επιλογής των Ευρετηρίων)

- **Tip 1:** Consider the **queries executed most of the time** (most important ones), e.g., for Oracle :
  - SELECT executions, sql_text FROM v$sqlarea ORDER BY executions desc;
  - V$ => Oracle's Dynamic Performance Views
- **Tip 2:** Try to choose indexes that **benefit as many queries as possible**
- **Tip 3:** Attributes in **WHERE** clause are candidates for index keys.
- **Tip 4:** **Hash** vs. **Tree**
  - **Exact match** condition suggests **Hash index**.
  - **Range query** suggests **tree index**.

Index Selection Guidelines
(Οδηγίες Επιλογής των Ευρετηρίων)

- **Tip 5:** Consider the best plan using the **current indexes**, and see if a better plan is possible with an additional index. If so, create it!
- **Tip 6:** Since only one index can be **clustered** per relation, choose it based on important queries that would benefit the most from clustering.
- **Tip 7:** **Multi-attribute <a,b,c> search keys** should be considered when a WHERE clause contains several conditions (e.g., a=3 and b>3 and c>3).
- **Tip 8:** Indexes can make **queries go** faster but **updates** become **slower**. Indexes also require additional disk space, choose them wisely!