Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 3

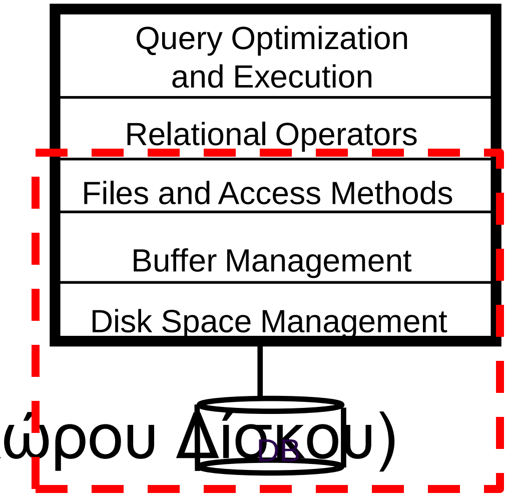
Storage II: Disks and Files

Chap. 9.1-9.7: Ramakrishnan & Gehrke

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

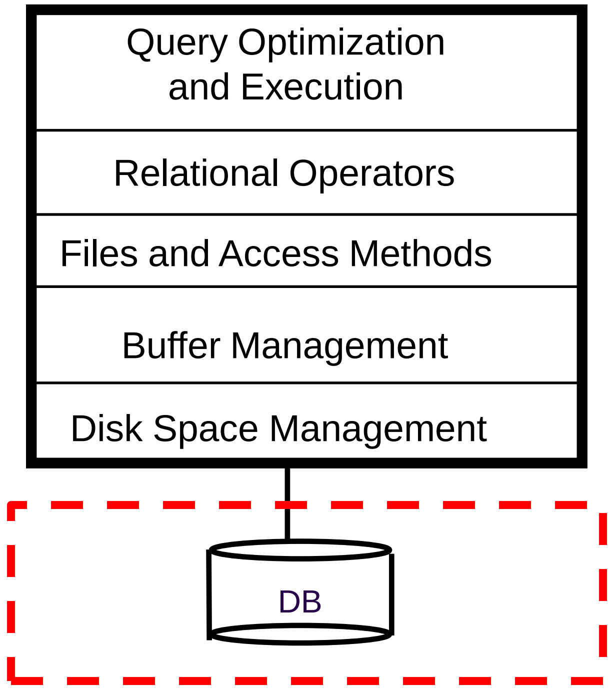
Lecture Outline
Overview of Storage and Indexing

- **Note:** In lecture 2 we gave an overview of **Storage and Indexing**. In this lecture we will explore **Storage** (**Disks & Files**) in more detail.



- 9.1-9.2) **Disks & RAID**
  - Components (Συστατικά) of a Disk
  - Accessing (Προσπέλαση) a Disk Block.
  - Arranging (Διάταξη) Pages on Disk
  - RAID Basic Concepts, Levels: 0 to 5 and 0+1
- 9.3) **Disk Space Manager** (Διαχειριστής Χώρου Δίσκου)
- 9.4) **Buffer Manager** (Διαχειριστής Κρυφής Μνήμης)
  - Definitions (Pin/Unpin, Dirty-bit), Replacement Policies (LRU, MRU, clock), Sequential Flooding, Buffer in OS
- 9.5-9.7) **File, Page and Record Formats**
  - **File Structure** (Linked-List/Directory-based), **Page Structure** with Fixed/Variable-length records, **Record Structure** (Fixed-length/Variable-length), **System Catalog**

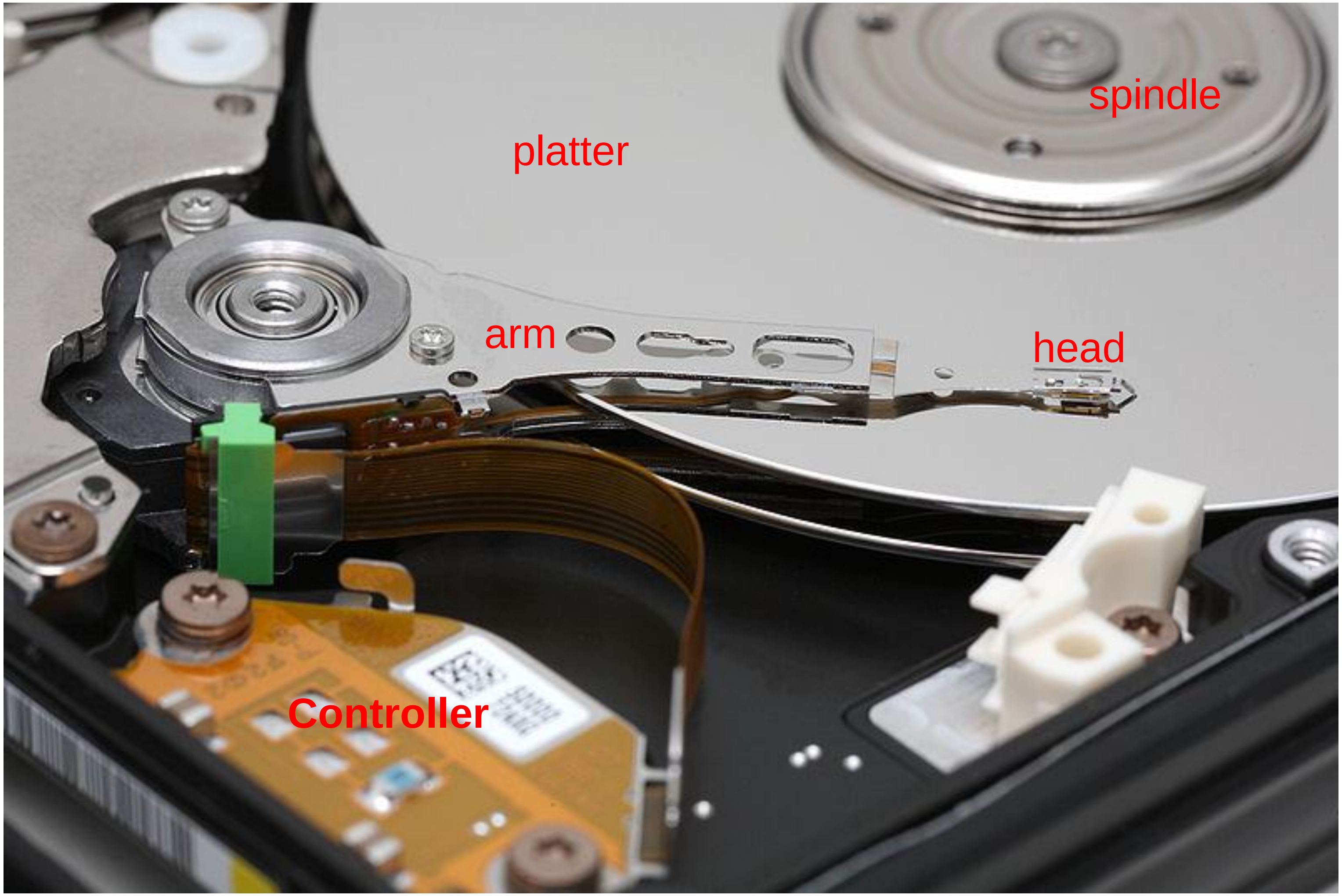
Context of next slides



Magnetic Disks (Μαγνητικοί Δίσκοι)

- DBMS **stores** information on ("hard") disks.
- This has major **implications (επιπτώσεις)** for DBMS design!
  - READ: transfer data from **disk** => **main memory (RAM).**
  - WRITE: transfer data from **RAM => disk**.
- Both are **high-cost operations**, relative to in-memory (RAM) operations, so must be **planned carefully**!
- We already mentioned that Data is stored and retrieved in **units** called ***pages (or disk blocks)*.**
- Unlike RAM, **time to retrieve a disk page** varies depending upon location on disk.
  - Therefore, **relative placement** (τοποθέτηση σε **εγγυήτητα**) of pages (utilized together) on disk has major **impact** on **DBMS performance**!

Magnetic Disks (Μαγνητικοί Δίσκοι)



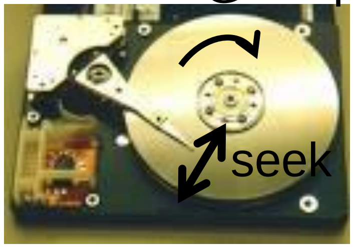
Accessing a Disk Block
(Προσπέλαση Μπλοκ Δίσκου)

- **Access Time (Χρόνος Πρόσβασης)** of a **Disk Block (Page)** =
  - ***+ Seek time (Χρόνος Αναζήτησης):*** *Time to **move arms** to position **disk head on track**.*
  - ***+ Rotational Delay (Καθυστέρηση Περιστροφής):*** Waiting for head to **rotate** to **expected block** (upto 15K rpm)
  - ***+ Transfer Time (Χρόνος Μεταφοράς):*** Time to **move data** to/from disk surface).

- **Seek time** and **Rotational Delay** dominate.
  - **Seek time** varies from about 1 to 20msec
  - **Rotational delay** varies from 0 to 10msec
  - **Transfer rate** is about 1msec per 4KB page

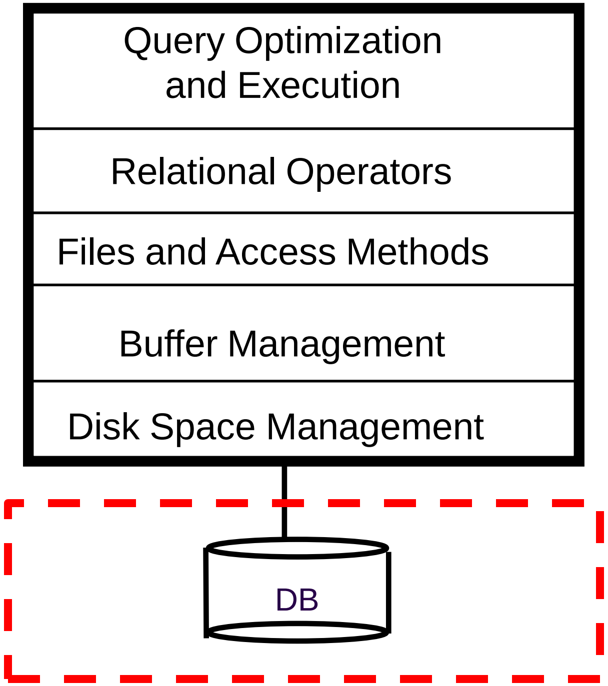
faster

Rotation @ 90rps



- Key to lower I/O cost: reduce seek/rotation delays!

Context of next slides



RAID: Redundant Array of Independent* Disks (*Εφεδρικές Συστοιχίες Ανεξαρτήτων Δίσκων*)

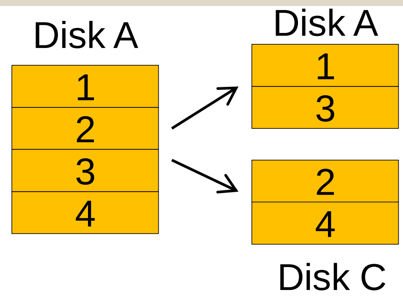
- **Disk Array:** Arrangement of several disks that gives **abstraction** of a **Single**, **Large Disk**!
- **Goals:**
  - Increase **Performance (Επίδοση);**
    - Why? Disk: a mechanical component that is inherently slow!
  - Increase **Reliability (Αξιοπιστία)**.
    - Why? Mechanical and Electronic Components tend to fail!

* Historically used to be **Inexpensive**

RAID: Key Concepts (RAID: Βασικές Αρχές)

A. **Striping (Διαχωρισμός):** the splitting of data across more than one disk using a round-robin (i mod disks);

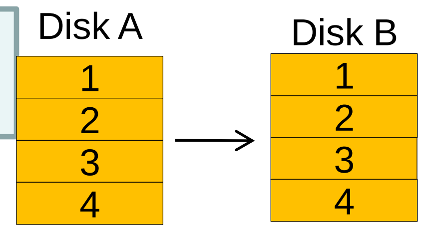
- Improving **Performance (Επίδοση)** and **Load Balancing** (εξισορρόπηση φόρτου)!
- **NOT** improving **Reliability (αξιοπιστία)!** (if one disk fails all data is useless)



A) Striping

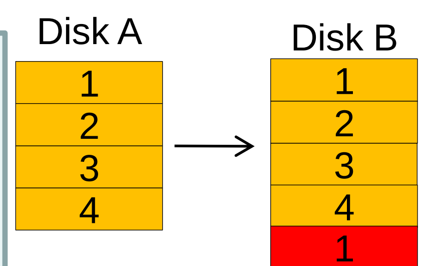
B. **Mirroring (Κατοπτρισμός) or Shadowing (Σκίαση):** the copying of data to more than one disk

- Improving **Reliability (Αξιοπιστία)!**
- Improving **Read Performance** but NOT **Write Performance** (same as 1 disk!) / **Wasting space**



B) Mirroring

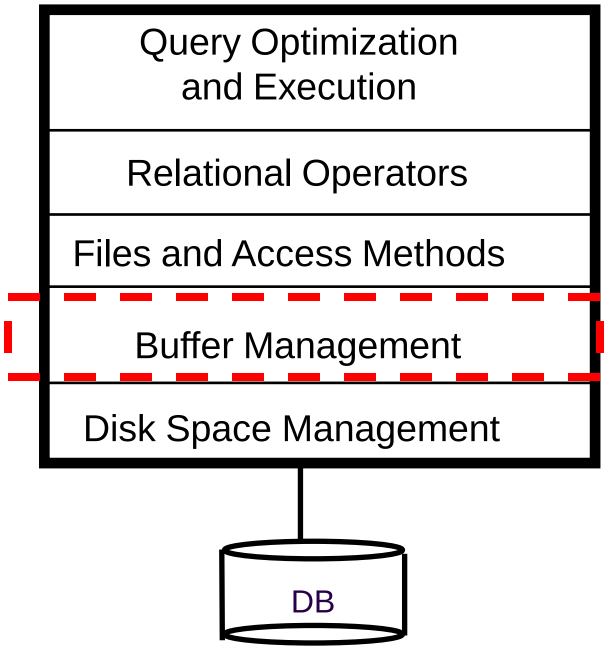
C. **Error Detection/Correction (Εντοπισμός/Διόρθωση Σφαλμάτων):** the storage of additional information, either on same disks or on redundant disk, allowing the **detection (parity, CRC)** and/or **correction** (Hamming/Reed-Solomon) of failures.



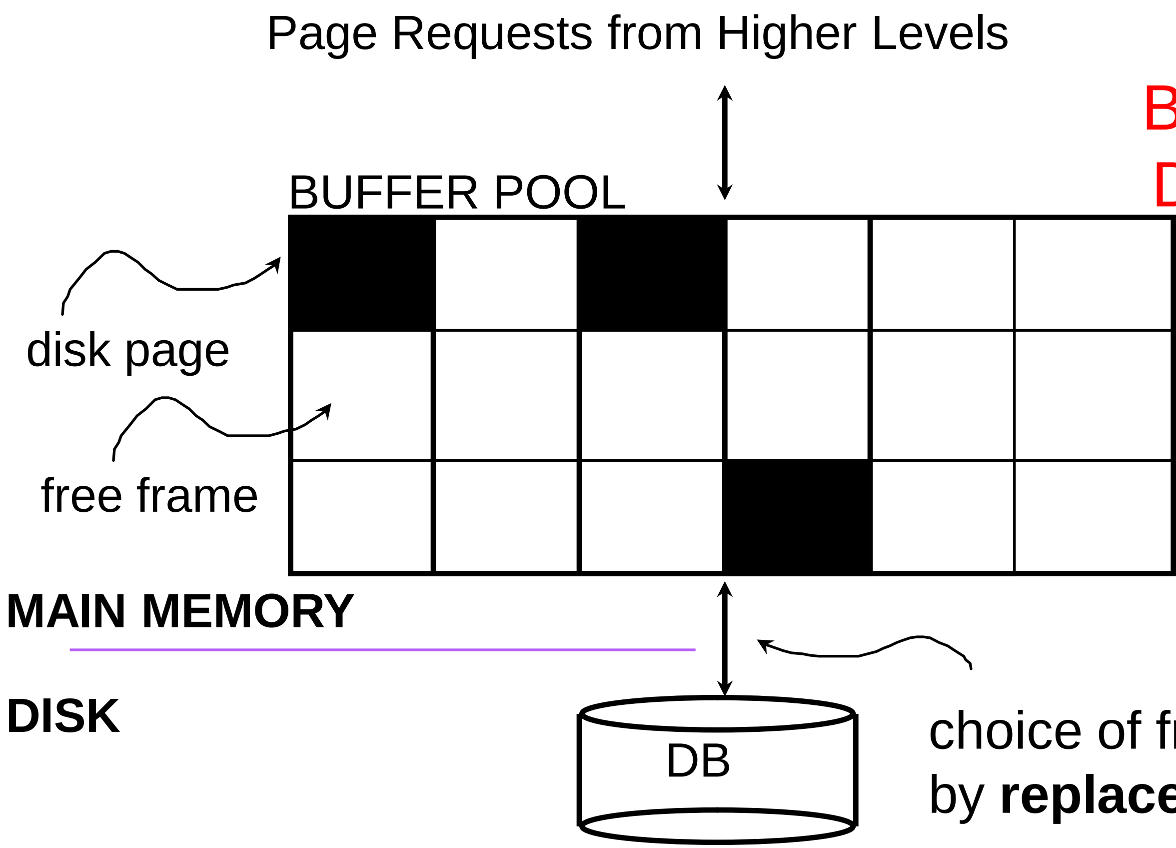
C) Error Detection

RAID levels combine the above basic concepts: 0 (striping), 1 (mirroring), 4,5 (parity)

Context of next slides



Buffer Management in a DBMS
(Διαχειριστής Κρυφής Μνήμης)



choice of frame dictated by **replacement policy**

<u>Ορολογία</u>
Buffer Frame =
DBMS Page =
Disk Block

- *Data must be in RAM for DBMS to operate on it!*
- *A **<pageid,dirty,pin>** is maintained for each **frame#***

When a Page is Requested ...
(Όταν αιτείται μια σελίδα…)

Case 1: Page is in Pool

– ***Pin (επικόλληση, αύξηση μετρητή)*** the page and return its address to the higher layer (file layer).

Case 2: Page NOT in Pool

Step 1 (Find): Choose a frame (page) for ***replacement*** *(*A page is a candidate for replacement iff *pin_count* = 0*). If no such page exist then page cannot be loaded into BM.*

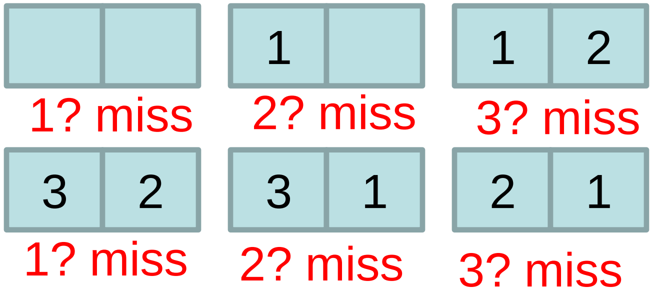
Step 2 (Save): If frame (page) is **dirty** **(has been modified by a write)**, then write it to disk

Step 3 (Load): Read requested page into chosen frame, **pin page** and return its address.

More on Buffer Management

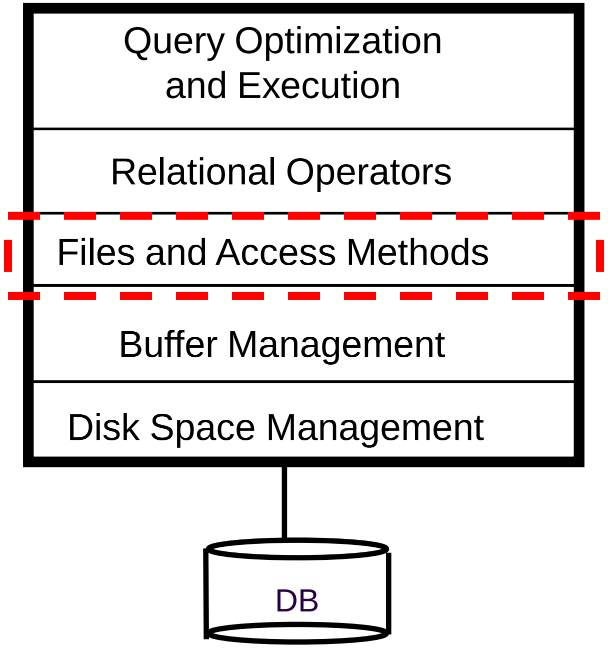
- **Unpinning a page:** Higher levels (requestors of page) **i) unpin** a page (when not needed anymore) and ii) set the **dirty-bit** to indicate the case a page has been modified.
- **Replacement Policy:** Policy that **defines** the **buffer frame** than needs to be **removed from the pool**:
  - **LRU** (using queue, remove the oldest from pool),
  - **MRU** (using stack, remove newest from pool),
  - **RANDOM** (randomly)
- ***Sequential flooding (Γραμμική Υπερχείλιση)*:** Situation caused by LRU + repeated sequential scans (σάρωση).

buffer frames < # pages in file means each page request causes an I/O.



Sequential flooding

Context of next slides



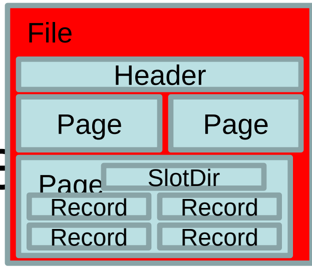
Files of Records
(Αρχείο από Εγγραφές)

- **Page or block** is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records* .

- FILE: A collection of pages, each **containing** a **collection of records**. Must support:
  - **insert/delete/modify** record
  - **read** a particular **record** (specified using *record id*)
  - **scan all records** (possibly with some conditions on the records to be retrieved)

Unordered (Heap) Files
(Μη-διατεταγμένα Αρχεία Σωρού)

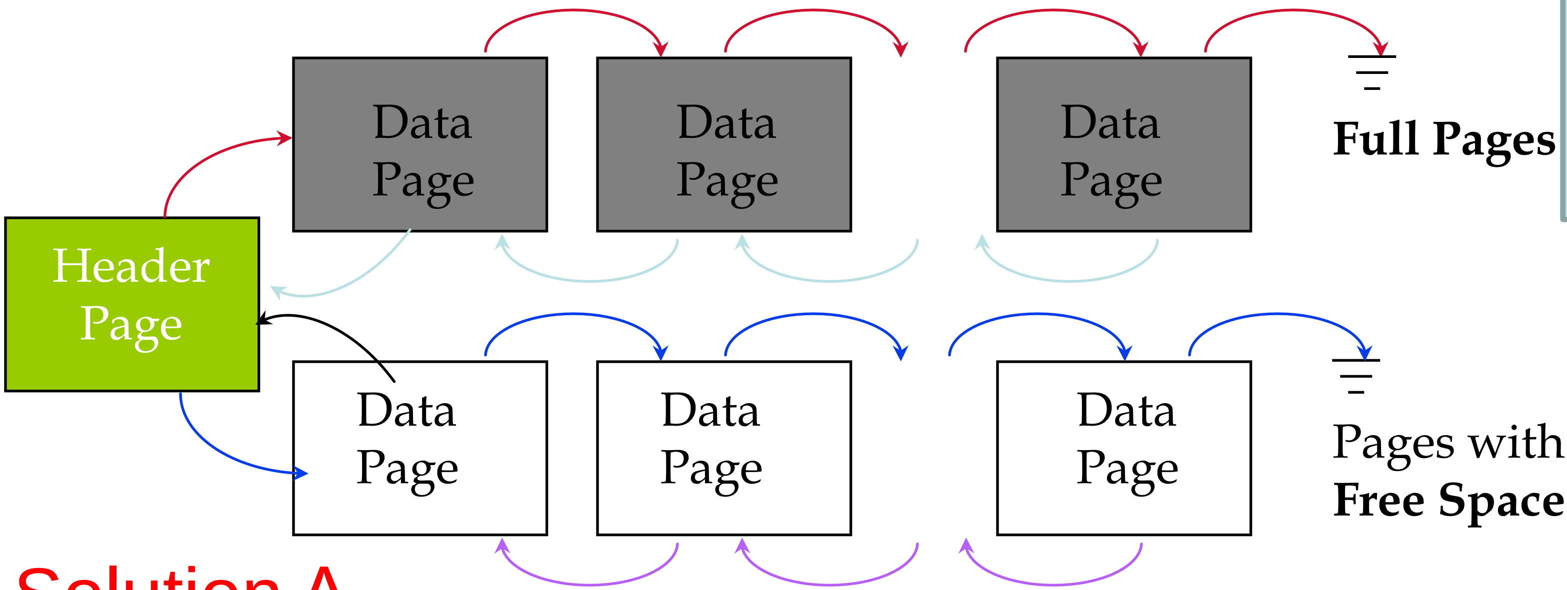
- Simplest **file structure** contains records in no particular order.
- As **file** grows and shrinks, disk **pages** are allocated and de-allocated.

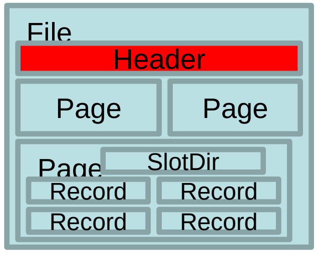


Context

- To support record level operations, we must:
  - keep track of the ***pages* in a file**
  - keep track of ***free space* on pages**
  - keep track of the ***records* on a page**
- There are **many alternatives** for keeping track of this. The following discussion presents these alternatives.

Keeping Track of Empty Pages (Βρίσκοντας τις Σελίδες με Χώρο)



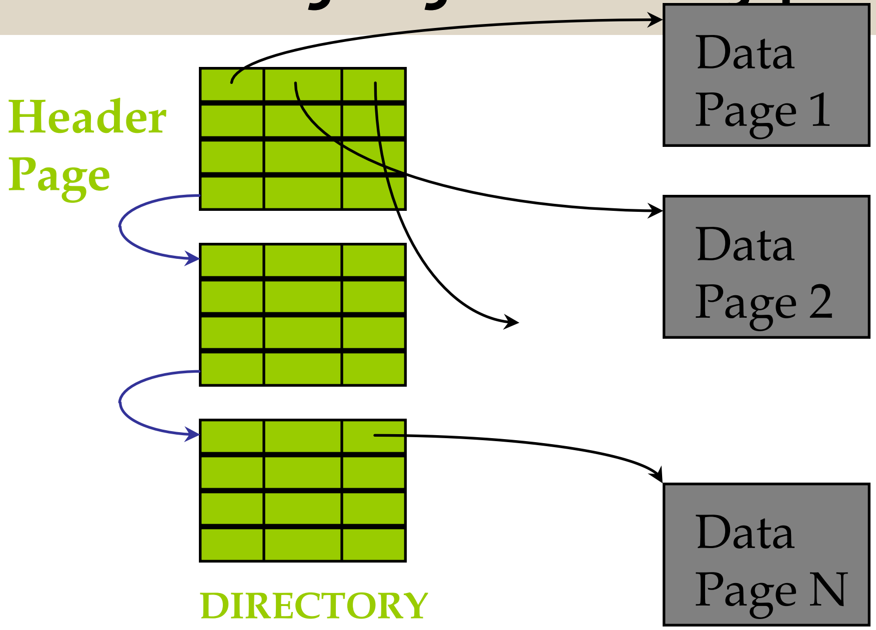


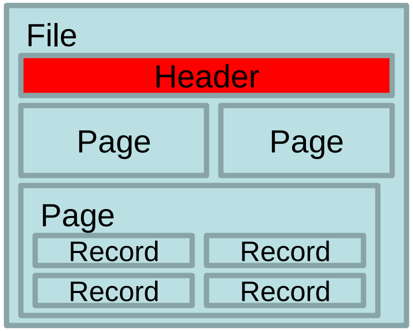
Context

Solution A

- **Linked-List Organization:** Each page contains 2 `pointers' plus data.
- Every time we **delete some data** from a page it is added to the **Free-Space list**
- **Drawbacks:**
  - All pages might end up in the Free-space list (every page might have a few empty bytes)
  - Linked list too big to fit into main memory, the next approach solves this problem!

Keeping Track of Empty Pages (Βρίσκοντας τις Σελίδες με Χώρο)



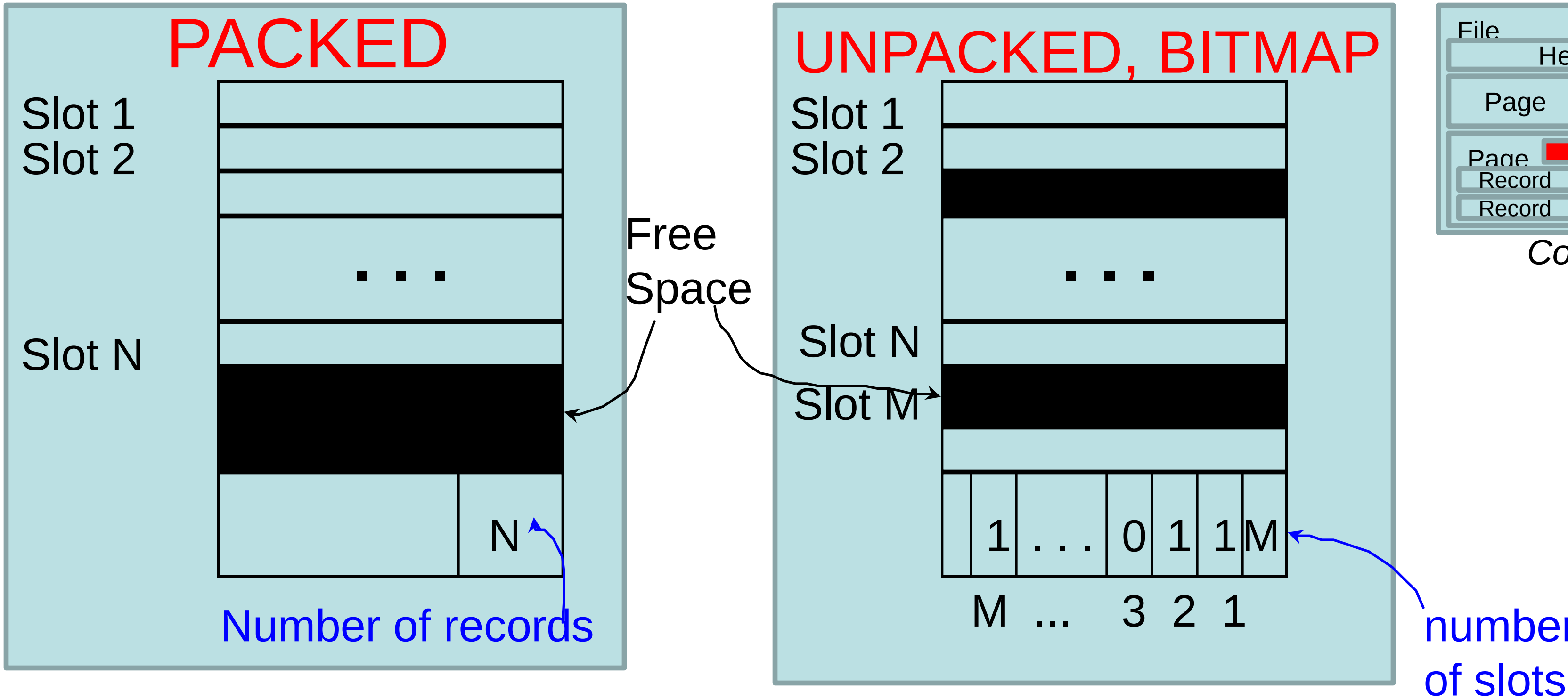


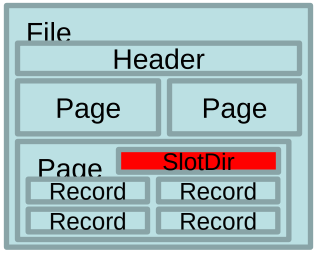
Context

Solution B

- **Directory-based Organization** (Οργάνωση με Ευρετήριο)
  - The entry for a page can include the **number of free bytes** on the page. That is useful to find if a page has enough space.
- The directory itself is a linked-list of directory pages;
  - *Much smaller than linked list of all File pages used in previous solution*!

Managing Slots on a Page with **Fixed-Length** Records

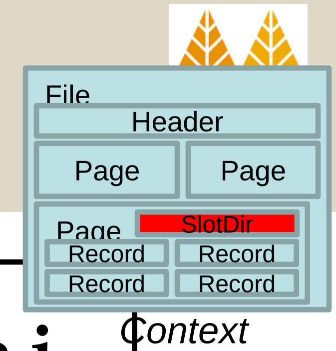


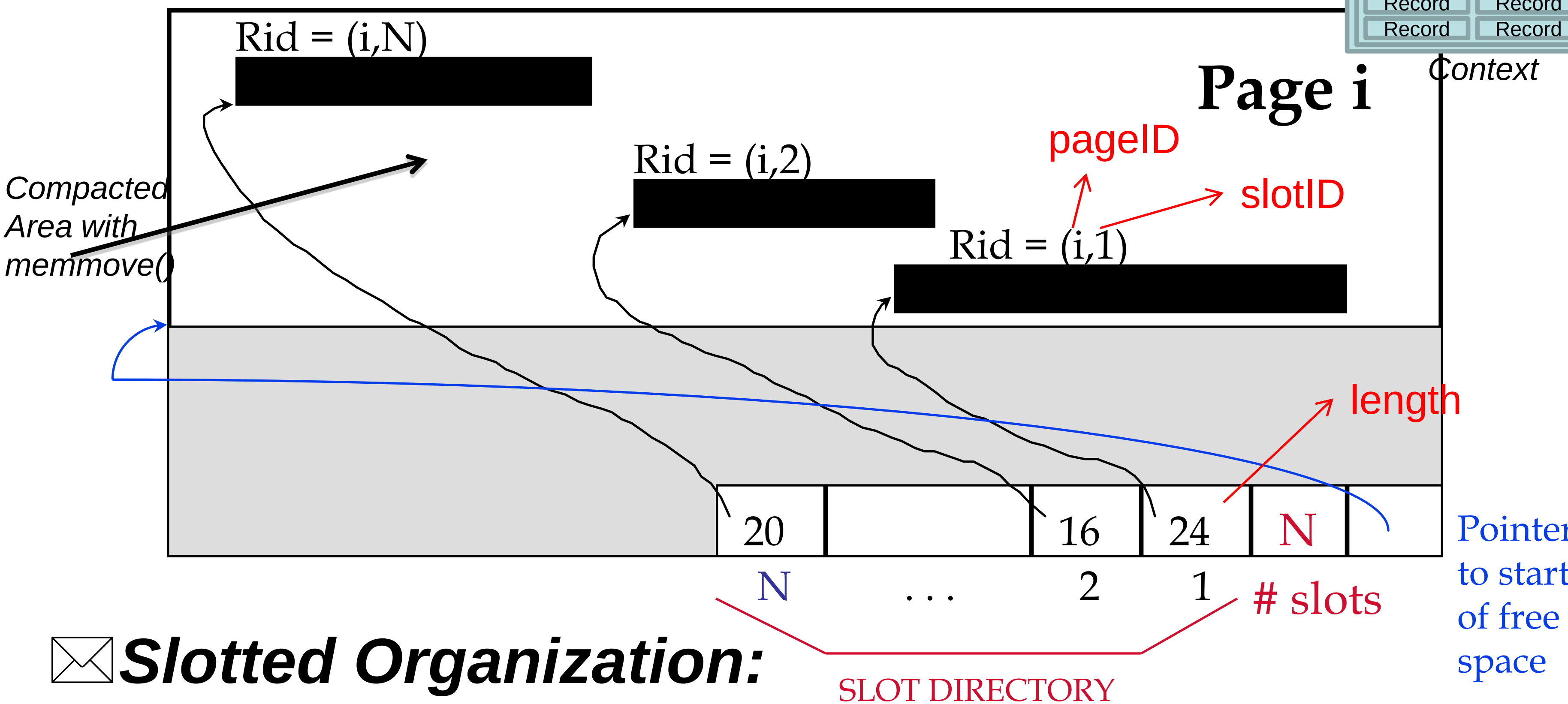


Context

- ***Packed:*** *If record Is deleted move the last record on the page into the vacated slot*
  - That changes RID (PageID, SlotID),which is not acceptable!
- **Unpacked/Bitmap:** Keep M-Bitmap which indicates which slots are vacant

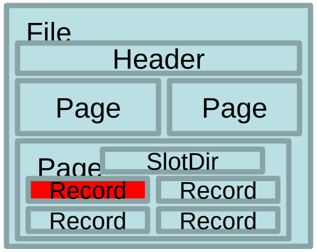
Managing Slots on a Page with **Variable** Records





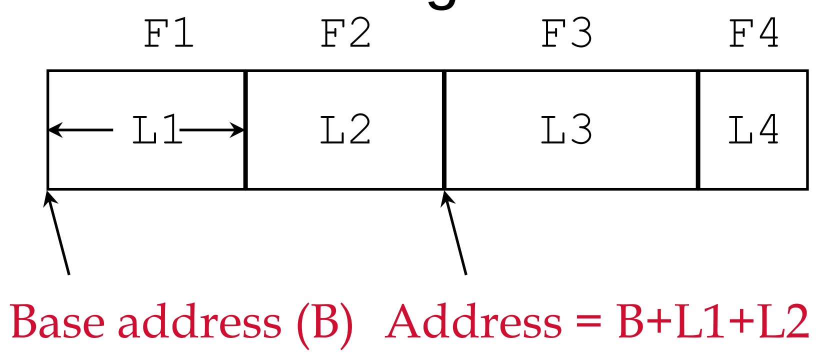
- ***Slotted Organization:***
  - *Suitable for **Variable-size Records** (slots never moved)*
  - *Can move records on page without changing RID so, attractive for **fixed-length** records too.*

Record Formats: Fixed Length (Δομή Εγγραφής: Σταθερού Μήκους)



Context

Fixed-length Record



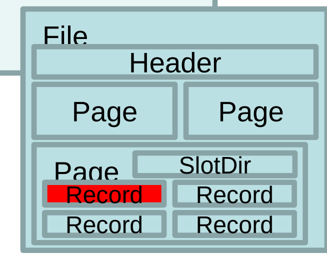
Field I (Attribute)

Li = Length of field i

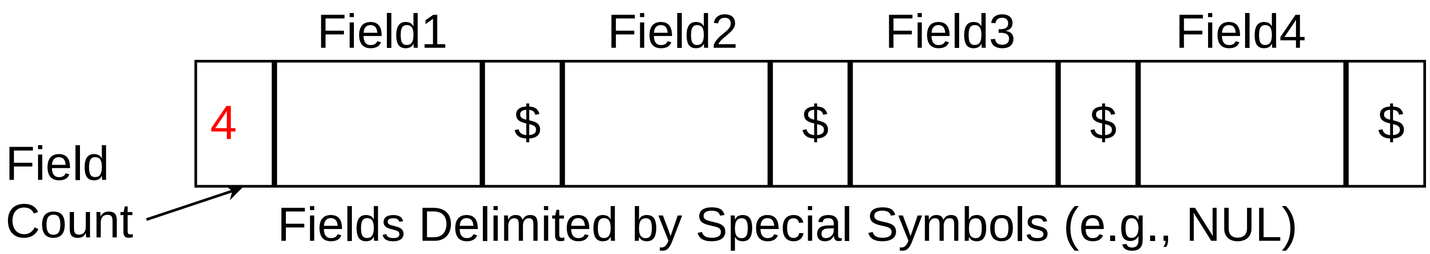
- Information about **field types** same for all records in a file; stored in *system catalogs (κατάλογος συστήματος).*
- Finding ***i'th* field** (or **record**) does not require scan of file, but the position of the file (or record) can be computed using simple **offset arithmetic**.

Record Formats: Variable Length
(Δομή Εγγραφής: Μεταβλητού Μήκους)

- When a record has a **variable length** (occurs with fields of variable size, e.g., strings)
- Two **alternative formats** (# fields is fixed):

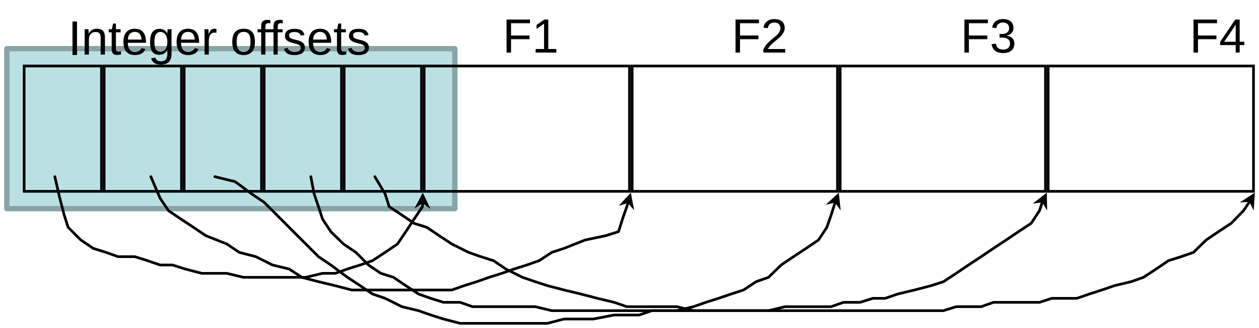


Context



Fields Delimited by Special Symbols (e.g., NUL)

*The **drawback** of the above format is that searching for a field requires to step over all fields. A better approach follows*



Array of Field Offsets

☒ **Second** solution offers **direct access** to i'th field, efficient storage, **fast access**

System Catalogs
(Κατάλογος Συστήματος)

- For each **relation** a DBMS stores the following:
  - name, file name, file structure (e.g., Heap file)
  - for each attribute: attribute name and type
  - for each index: index name
  - integrity constraints
- For each **index**:
  - structure (e.g., B+ tree) and search key fields
- For each **view**:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

⌧ ***Catalogs are themselves stored as relations!***

System Catalog in PostgreSQL

Catalog Name	Purpose
pg_aggregate	**aggregate functions**
pg_am	**index access methods**
pg_amop	access method operators
pg_amproc	access method support procedures
pg_attrdef	column default values
pg_attribute	**table columns ("attributes", "fields")**
pg_cast	casts (data type conversions)
pg_class	tables, indexes, sequences ("relations")
pg_constraint	check constraints, unique / **primary key constraints, foreign key constraints**
pg_conversion	encoding conversion information
pg_database	databases within this database cluster
pg_depend	dependencies between database objects

Catalog Name	Purpose
pg_description	descriptions or comments on database objects
pg_group	groups of database users
pg_index	additional index information
pg_inherits	table inheritance hierarchy
pg_language	languages for writing functions
pg_largeobject	large objects
pg_listener	asynchronous notification
pg_namespace	namespaces (schemas)
pg_opclass	index access method operator classes
pg_operator	operators
pg_proc	functions and procedures
pg_rewrite	query rewriter rules
pg_shadow	database users
pg_statistic	optimizer statistics
pg_trigger	triggers
pg_type	data types

For example, CREATE DATABASE inserts a row into the pg_database catalog -- and creates the database on disk.

Example of Attribute Table in a Typical System Catalog

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Position within relation