Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 3

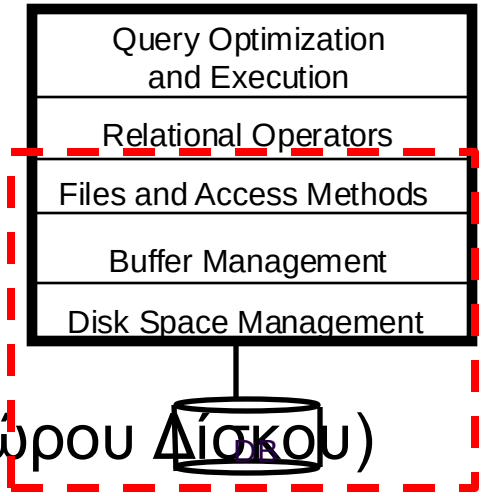
Storage II: Disks and Files

Chap. 9.1-9.7: Ramakrishnan & Gehrke

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

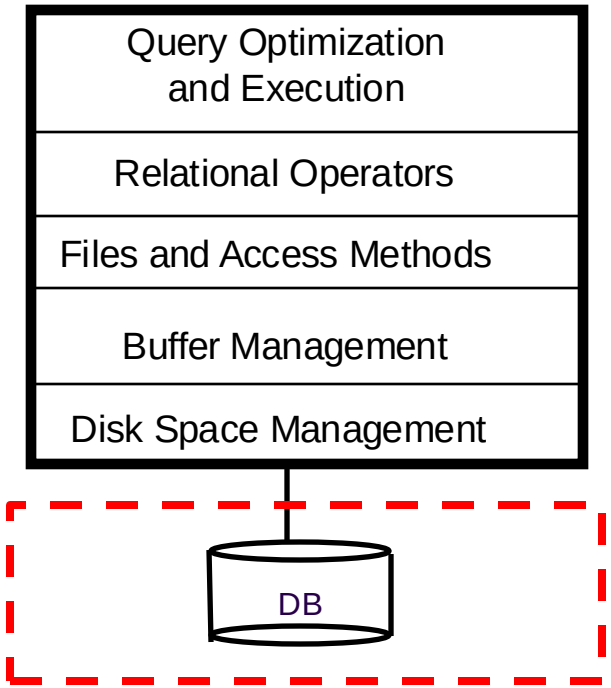
Lecture Outline
Overview of Storage and Indexing

- **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**

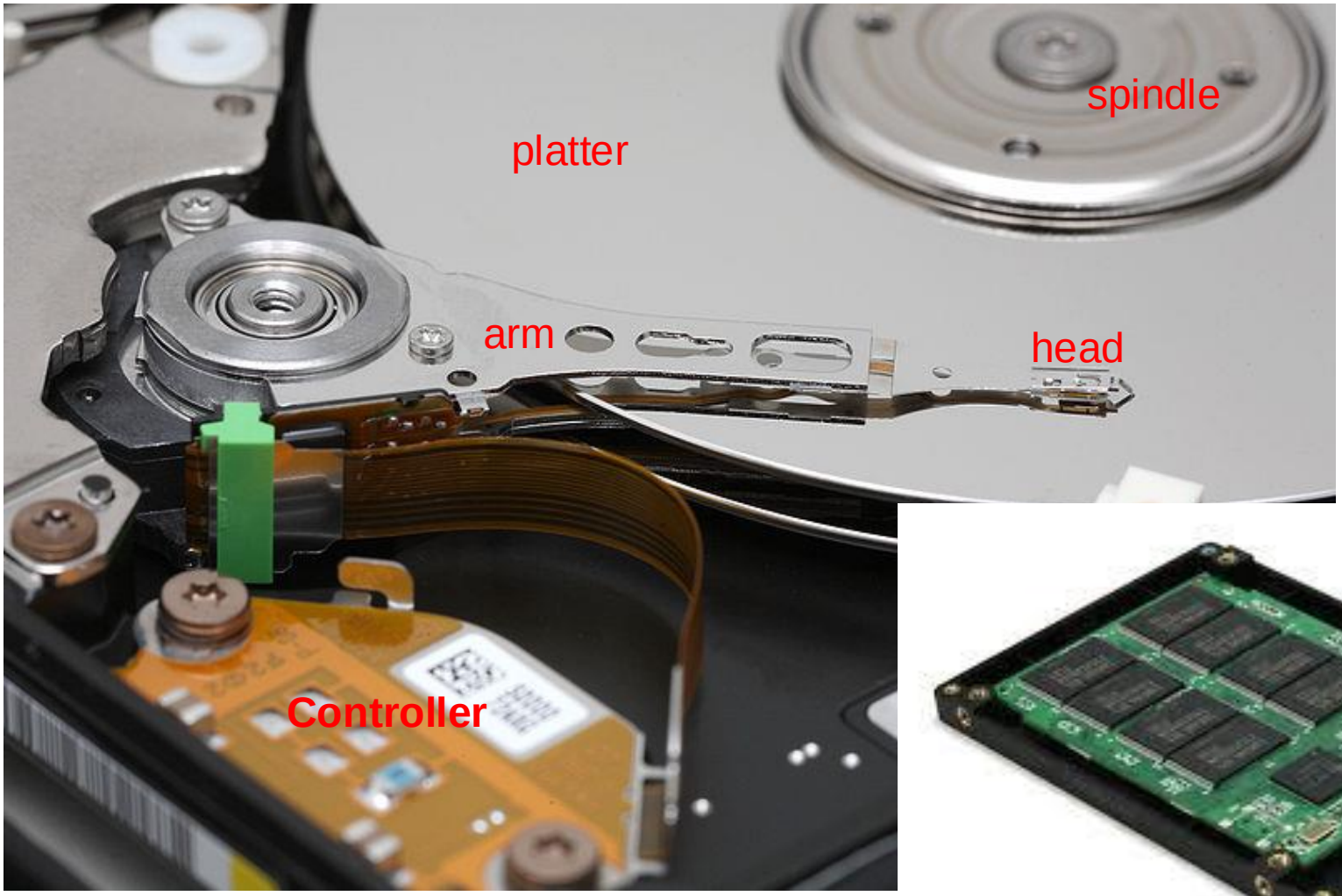
Context of next slides



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)



HDD (Hard Disk Drive)

SSD (Solid State Disk)

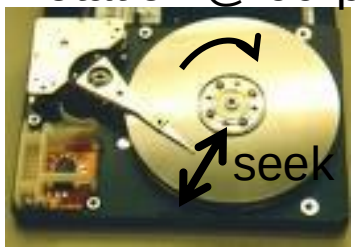
Accessing a Disk Block (Προσπέλαση Μπλοκ Δίσκου)

- **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

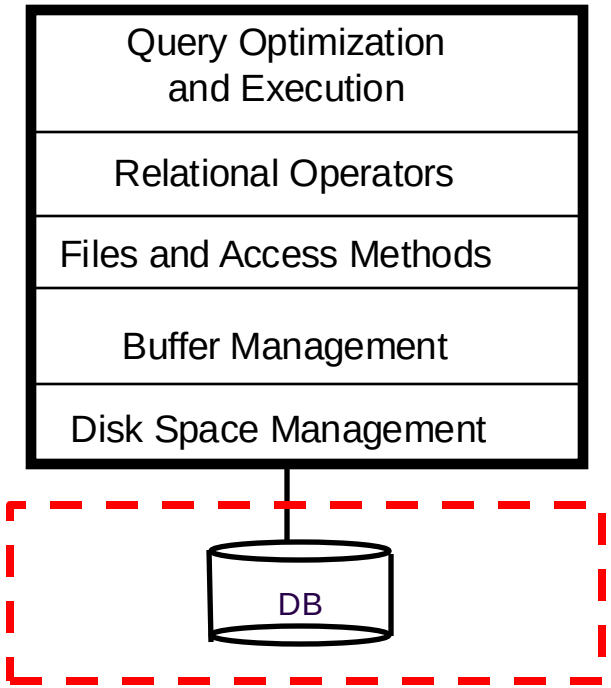
faster ↓

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*)

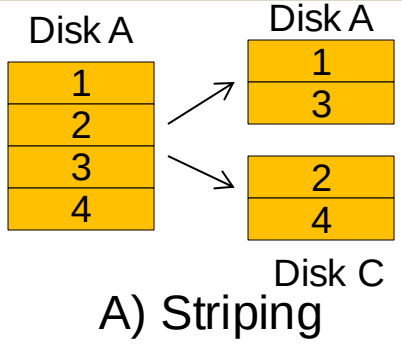
- **Disk Array:** Arrangement of several disks that gives **abstraction** of a **Single**, **Large Disk**!
- **Goals:**
  - Increase **Performance (Επίδοση);**
    - Why? Disk: a mechanical component that is inherently slow!
  - Increase **Reliability (Αξιοπιστία)**.
    - Why? Mechanical and Electronic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

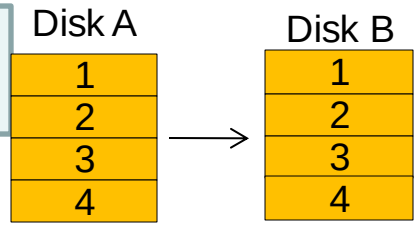
- Improving **Performance (Επίδοση)** and **Load Balancing** (εξισορρόπηση φόρτου)!
- **NOT** improving **Reliability (αξιοπιστία)!** (if one disk fails all data is useless)



A) Striping

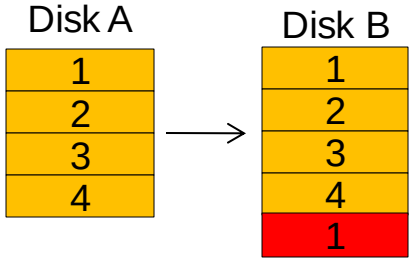
B. **Mirroring (Κατοπτρισμός) or Shadowing (Σκίαση):** the copying of data to more than one disk

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



B) Mirroring

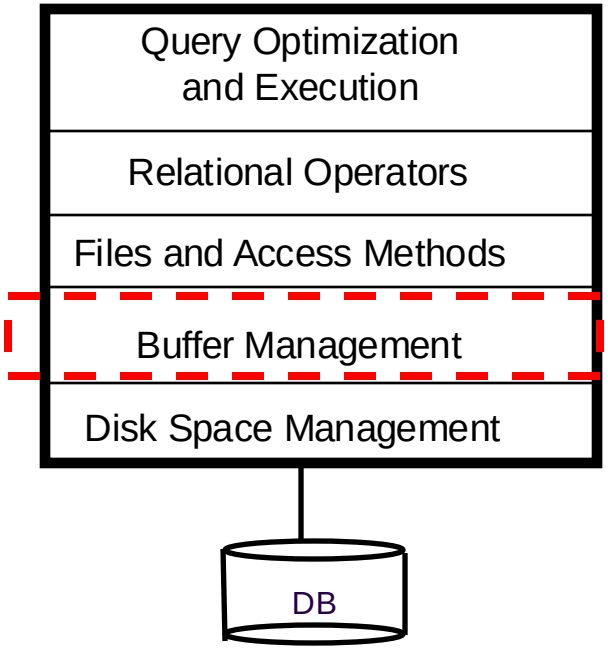
C. **Error Detection/Correction (Εντοπισμός/Διόρθωση Σφαλμάτων):** the storage of additional information, either on same disks or on redundant disk, allowing the **detection (parity, CRC)** and/or **correction** (Hamming/Reed-Solomon) of failures.



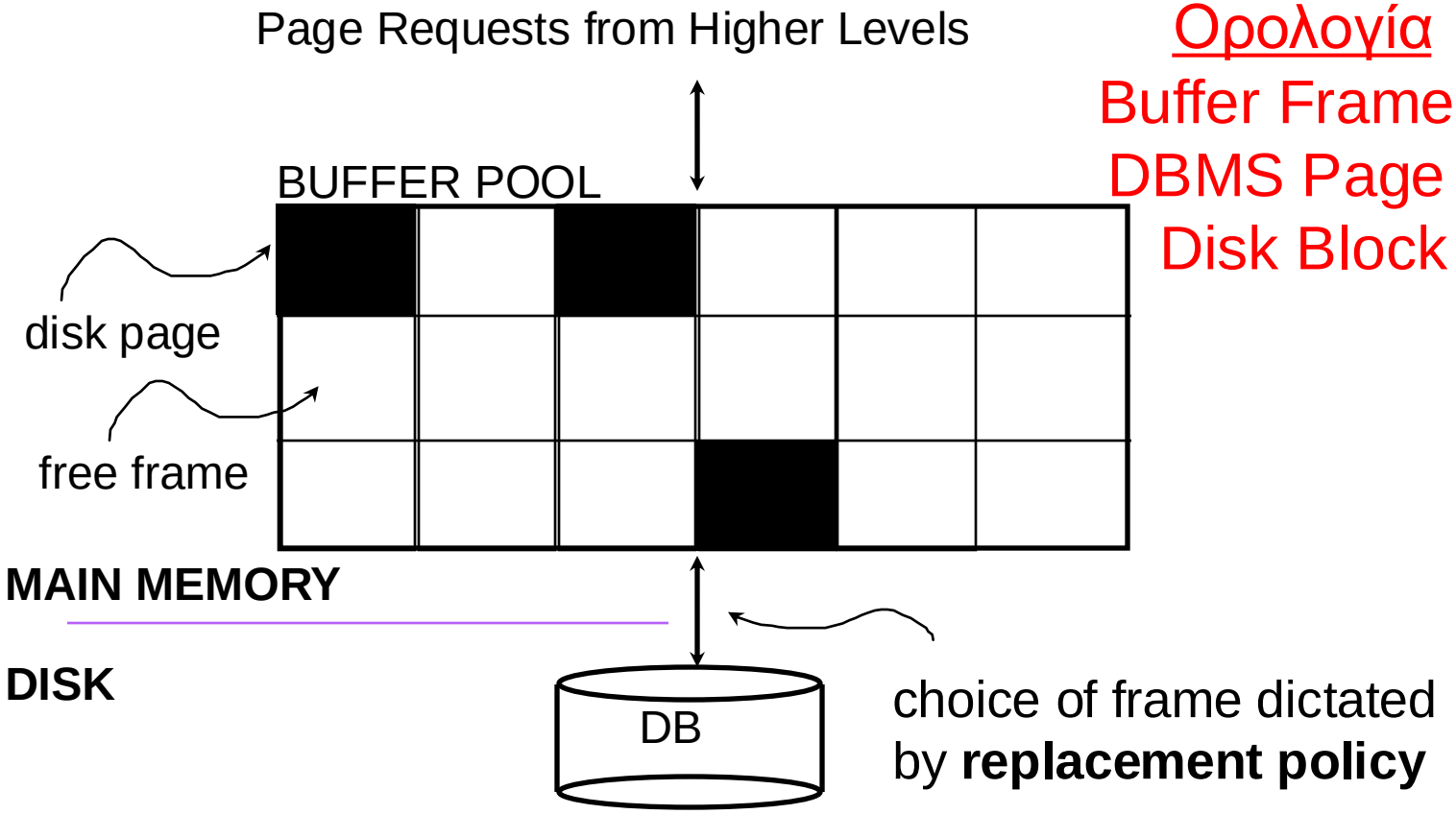
C) Error Detection

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

Context of next slides



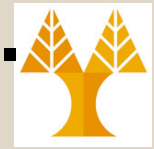
Buffer Management in a DBMS
(Διαχειριστής Κρυφής Μνήμης)



Ορολογία
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.*

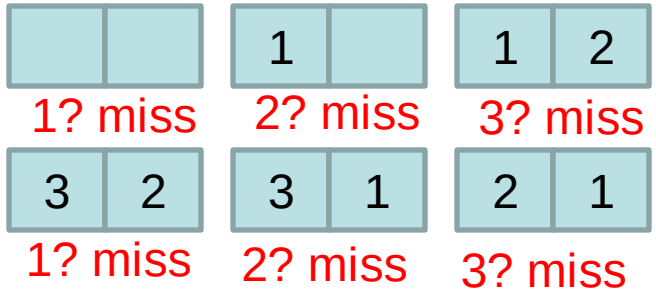
Step 2 (Save): If frame (page) is **dirty** **(has been modified by a write)**, then write it to disk

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

More on Buffer Management

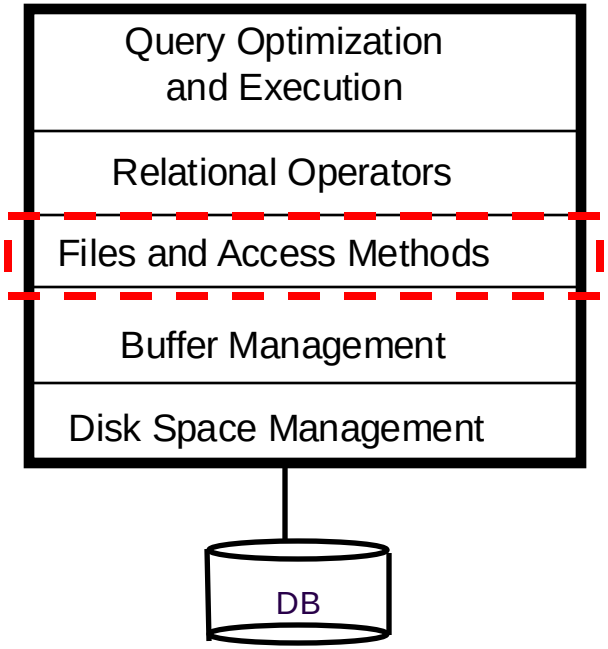
- **Unpinning a page:** Higher levels (requestors of page) **i) unpin** a page (when not needed anymore) and ii) set the **dirty-bit** to indicate the case a page has been modified.
- **Replacement Policy:** Policy that **defines** the **buffer frame** than needs to be **removed from the pool**:
  - **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



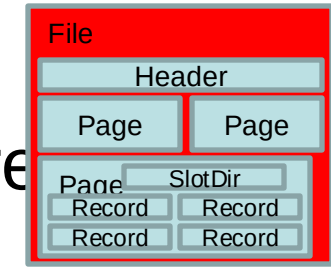
Files of Records
(Αρχείο από Εγγραφές)

- **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)

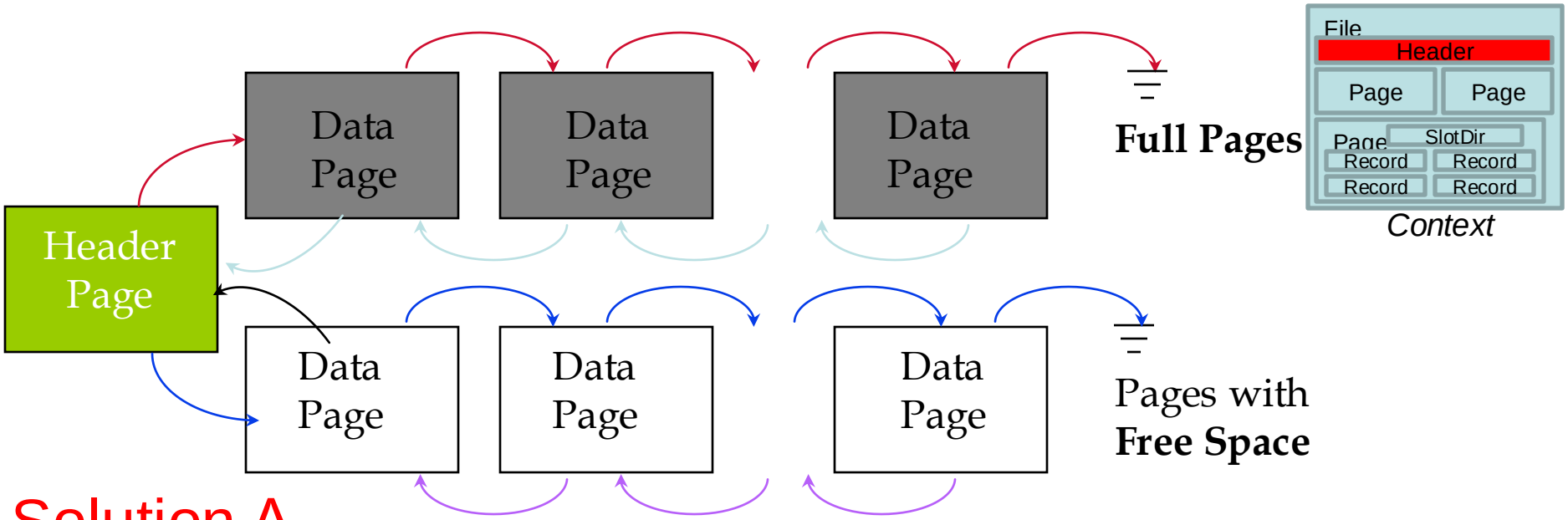
Unordered (Heap) Files
(Μη-διατεταγμένα Αρχεία Σωρού)

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



Context

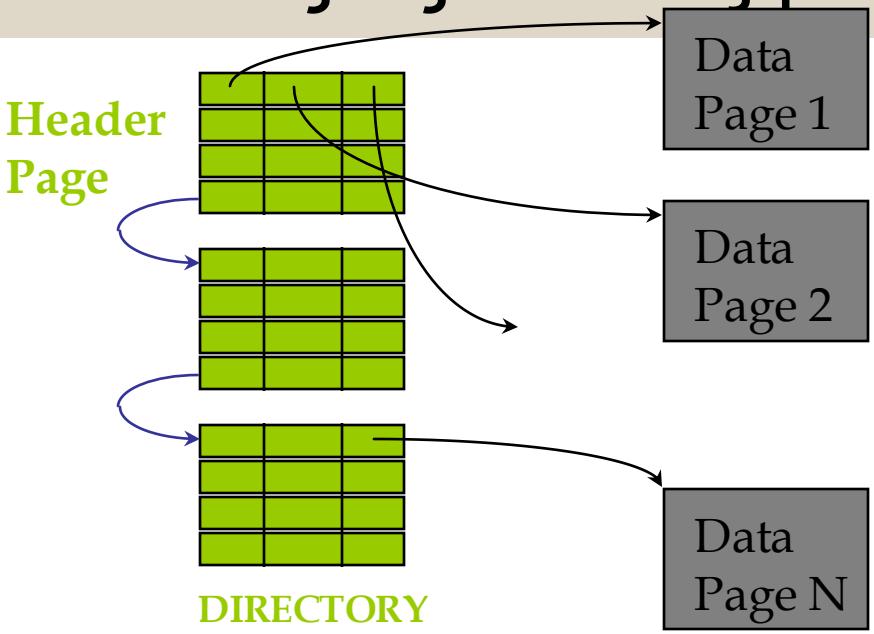
Keeping Track of Empty Pages (Βρίσκοντας τις Σελίδες με Χώρο)

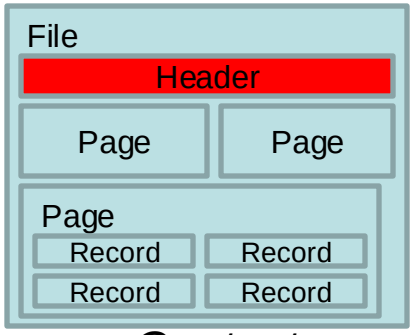


Solution A

- **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)



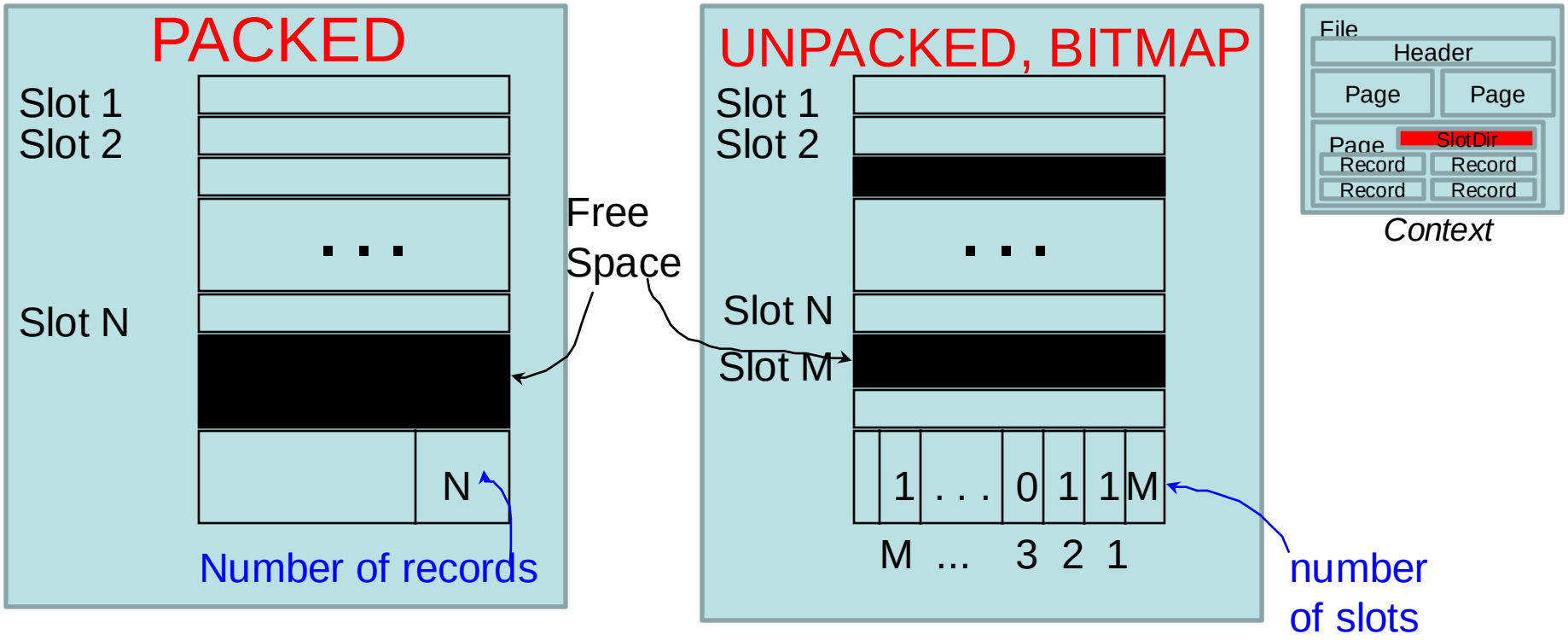


Context

Solution B

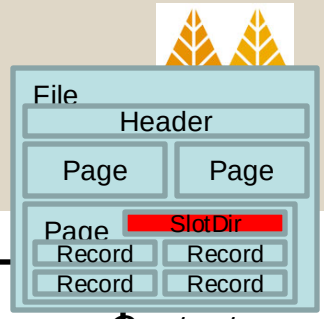
- **Directory-based Organization** (Οργάνωση με Ευρετήριο)
  - The entry for a page can include the **number of free bytes** on the page. That is useful to find if a page has enough space.
- The directory itself is a linked-list of directory pages;
  - *Much smaller than linked list of all File pages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

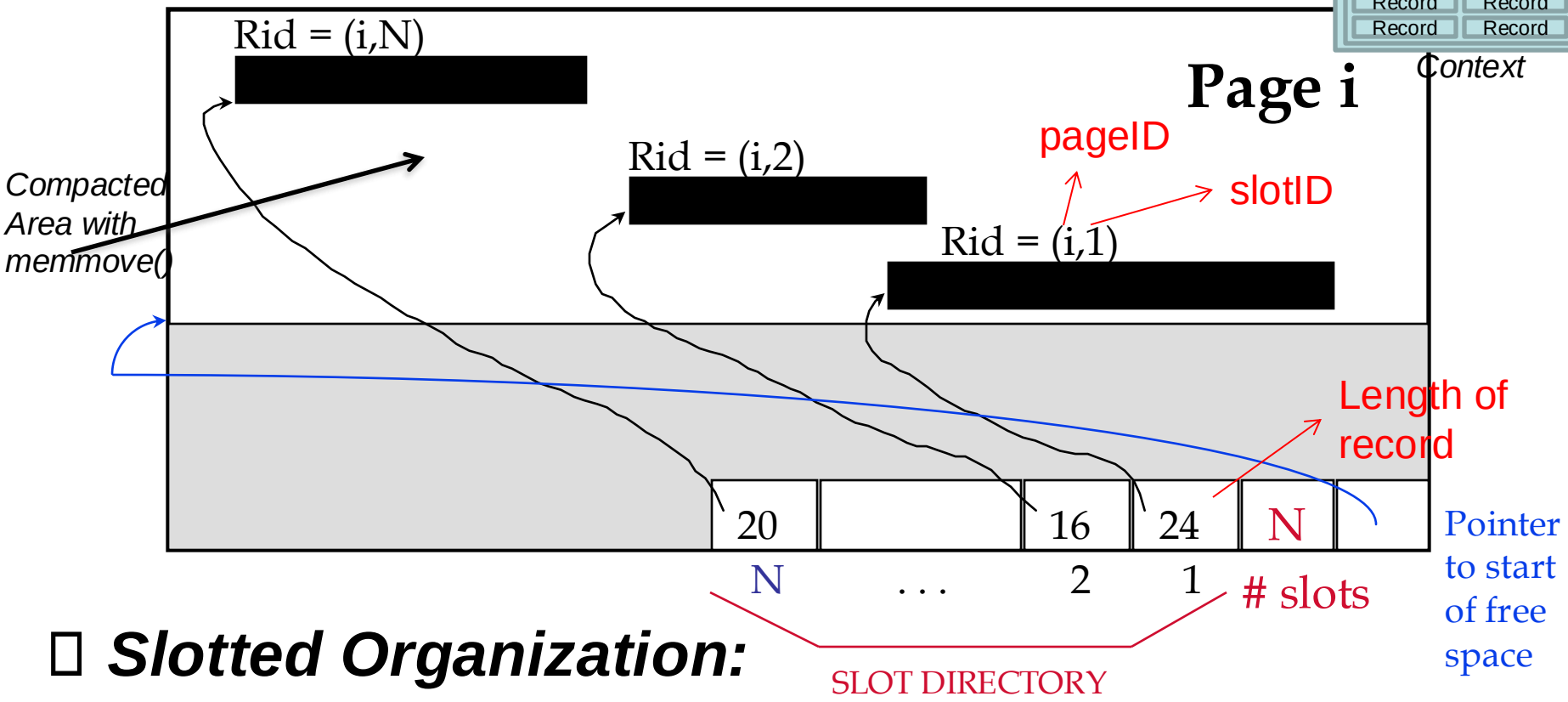
Managing Slots on a Page with **Fixed-Length** Records



* ***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

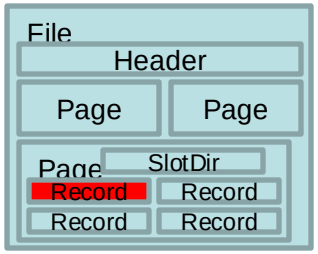
Managing Slots on a Page with **Variable** Records



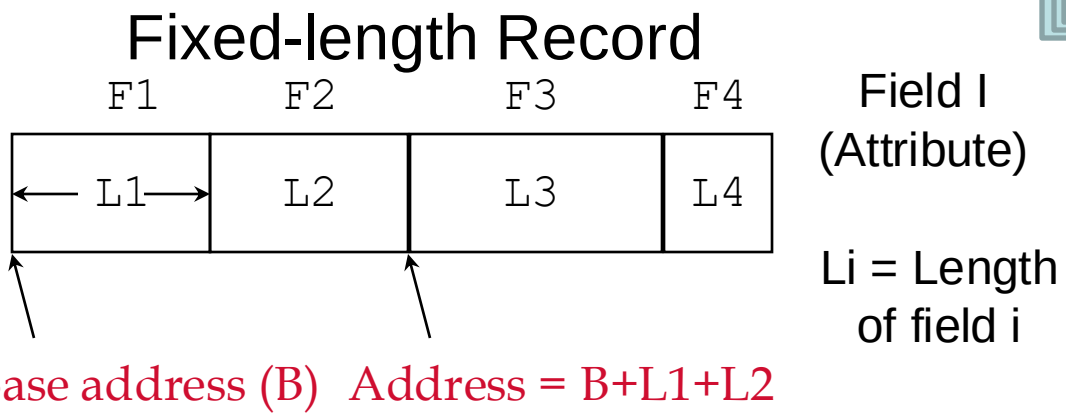


- ***Slotted Organization:***
  - *Suitable for **Variable-size Records** (slots never moved)*
  - *Can move records on page without changing RID so, attractive for **fixed-length** records too.*

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



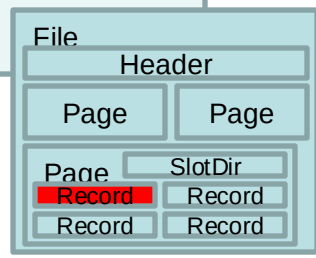
Context

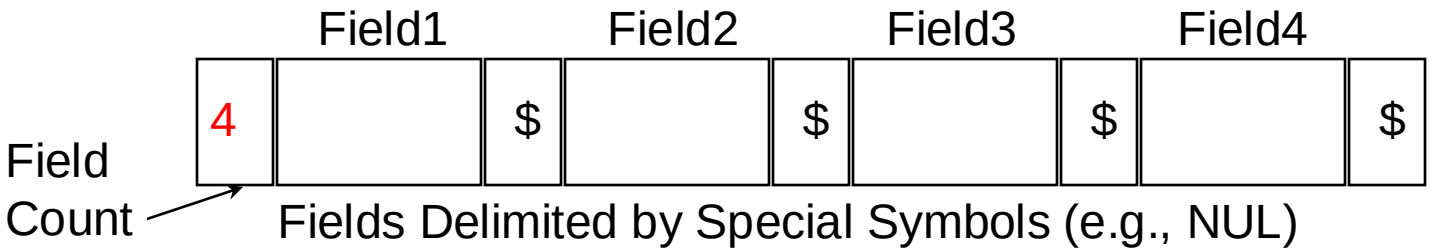


- Information about **field types** same for all records in a file; stored in *system catalogs (κατάλογος συστήματος).*
- Finding ***i'th* field** (or **record**) does not require scan of file, but the position of the file (or record) can be computed using simple **offset arithmetic**.

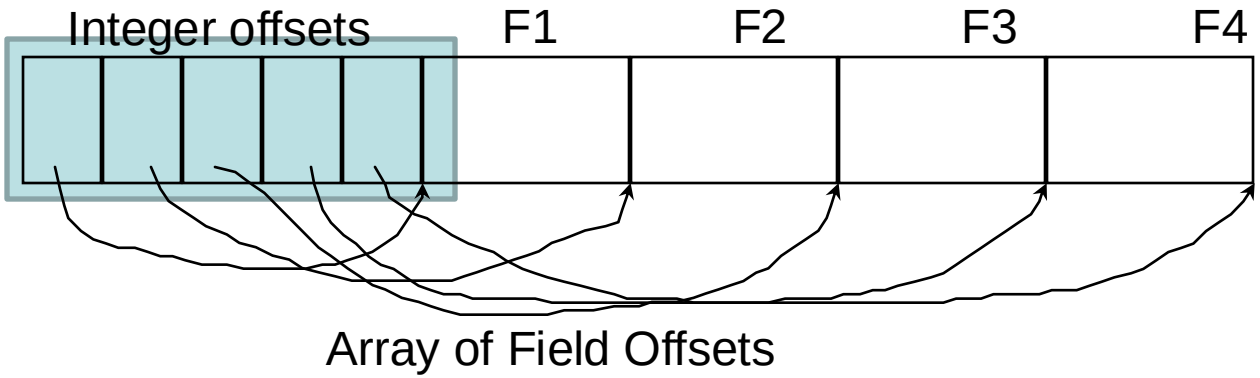
Record Formats: Variable Length (**Δομή Εγγραφής: Μεταβλητού Μήκους**)

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





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



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

SQL Server Data Types Example (Characterization)

bigint	8	Integer from -2^63 (-9 223 372 036 854 775 808) to 2^63-1 (9 223 372 036 854 775 807).
int	4	Integer from -2^31 (-2 147 483 648) to 2^31-1 (2 147 483 647).
smallint	2	Integer from -2^15 (-32 768) to 2^15-1 (32 767).
tinyint	1	Integer from 0 to 255.
bit	1 bit	Integer 0 or 1.
decimal(precision, scale)	5-17	Numeric data type with fixed precision and scale (accuracy 1-38, 18 by default and scale 0-p, 0 by default).
numeric	5-17	Same as data type 'decimal'.
		Financial data type from -

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
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

Column Files
(Apache Parquet & Apache Arrow)

- Apache Parquet is an open-source, standard, column-oriented file format that grew out of the Hadoop era of big-data.
  - typically used with big data processing frameworks like Apache Spark, Apache Hive, and Apache Drill. It was created by Twitter and Cloudera and is part of the Apache Hadoop ecosystem.
    - **Apache Arrow** is a universal columnar format and **multi-language toolbox** for fast data interchange and in-memory analytics. It contains a set of technologies that enable data systems to efficiently store, process, and move data.

	Product	Customer	Country	Date	Sales Amount
Row 1	Ball	John Doe	USA	2023-01-01	100
Row 2	T-Shirt	John Doe	USA	2023-01-02	200
Row 3	Socks	Maria Adams	UK	2023-01-01	300
Row 4	Socks	Antonio Grant	USA	2023-01-03	100
Row 5	T-Shirt	Maria Adams	UK	2023-01-02	500
Row 6	Socks	John Doe	USA	2023-01-05	200

Row

Column 1	Column 2	Column 3	Column 4	Column 5
Product	Customer	Country	Date	Sales Amount
Ball	John Doe	USA	2023-01-01	100
T-Shirt	John Doe	USA	2023-01-02	200
Socks	Maria Adams	UK	2023-01-01	300
Socks	Antonio Grant	USA	2023-01-03	100
T-Shirt	Maria Adams	UK	2023-01-02	500
Socks	John Doe	USA	2023-01-05	200

Column

	Column 1	Column 2	Column 3	Column 4	Column 5
	Product	Customer	Country	Date	Sales Amount
Row Group 1	Ball	John Doe	USA	2023-01-01	100
	T-Shirt	John Doe	USA	2023-01-02	200
Row Group 2	Socks	Maria Adams	UK	2023-01-01	300
	Socks	Antonio Grant	USA	2023-01-03	100
Row Group 3	T-Shirt	Maria Adams	UK	2023-01-02	500
	Socks	John Doe	USA	2023-01-05	200

Column + Groups

More on groups, compression in later sections

Pandas (Python)

- **Pandas or Dask/Polars (distributed and parallel Pandas) provides data structures for in-memory analytics**
- **Pandas** loads the data into memory for processing.
  - it takes advantage of the in-memory data structure (like DataFrame) to perform operations efficiently.

pandas

```
import pandas as pd

# Creating a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles',
'Chicago']
}

df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

dask

Distributed processing,

```
import dask.dataframe as dd

# Read the large CSV file using Dask
df = dd.read_csv('large_file.csv')

# Calculate the average salary
average_salary = df['Salary'].mean().compute()

# Print the result
print("Average Salary:", average_salary)
```

parallel data processing,

```
import polars as pl

# Creating a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
df = pl.DataFrame(data)

# Display the DataFrame
print(df)
```

dask Parquet

```
import dask.dataframe as dd

# Read the Parquet file using
Dask
df =
dd.read_parquet('large_file.par
quet')

# Calculate the average salary
using Dask's mean function
average_salary =
df['Salary'].mean().compute()

# Print the result
print("Average Salary:",
average_salary)
```

Parquet File Format

4-byte magic number "PAR1"

<Column 1 Chunk 1>

<Column 2 Chunk 1>

...

<Column N Chunk 1>

<Column 1 Chunk 2>

<Column 2 Chunk 2>

...

<Column N Chunk 2>

...

<Column 1 Chunk M>

<Column 2 Chunk M>

...

<Column N Chunk M>

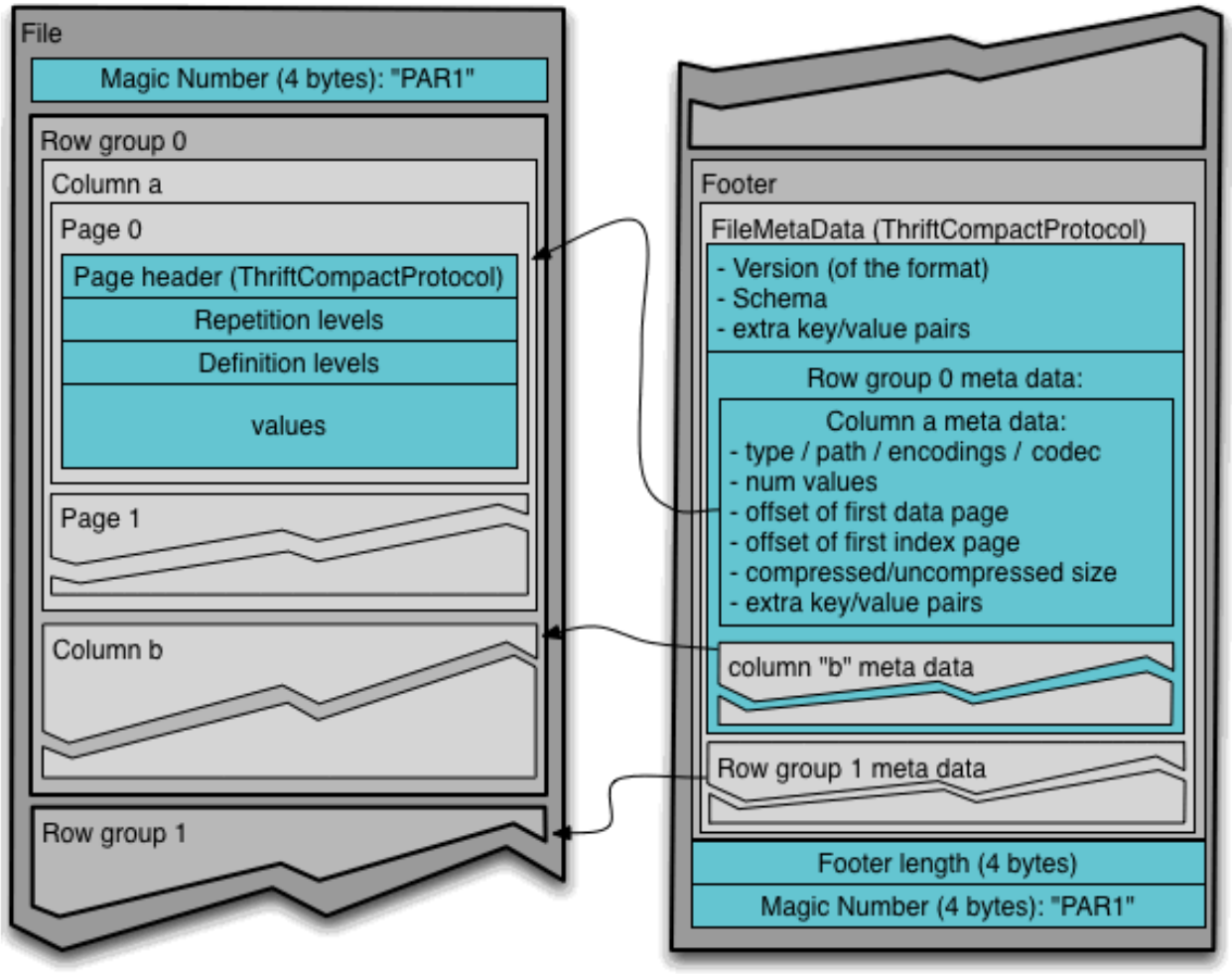
	Column 1	Column 2	Column 3	Column 4	Column 5
	Product	Customer	Country	Date	Sales Amount
Row Group 1	Ball	John Doe	USA	2023-01-01	100
	T-Shirt	John Doe	USA	2023-01-02	200
Row Group 2	Socks	Maria Adams	UK	2023-01-01	300
	Socks	Antonio Grant	USA	2023-01-03	100
Row Group 3	T-Shirt	Maria Adams	UK	2023-01-02	500
	Socks	John Doe	USA	2023-01-05	200

File Metadata

4-byte length in bytes of file metadata (little endian)

4-byte magic number "PAR1"

Parquet File Format



- File Metadata is written after the data to allow for single pass writing.
- Readers are expected to **first read the file metadata** to find all the column chunks they are interested in**.**
- **The columns chunks should then be read sequentially.**

https://github.com/apache/parquet-format?tab=readme-ov-file

Parquet Codecs

UNCOMPRESSED

No-op codec. Data is left uncompressed.

SNAPPY

A codec based on the Snappy compression format. If any ambiguity arises when implementing this format, the implementation provided by Google Snappy library is authoritative.

GZIP

A codec based on the GZIP format (not the closely-related "zlib" or "deflate" formats) defined by RFC 1952. If any ambiguity arises when implementing this format, the implementation provided by the zlib compression library is authoritative.

Readers should support reading pages containing multiple GZIP members, however, as this has historically not been supported by all implementations, it is recommended that writers refrain from creating such pages by default for better interoperability.

LZO

A codec based on or interoperable with the LZO compression library.

BROTLI

A codec based on the Brotli format defined by RFC 7932. If any ambiguity arises when implementing this format, the implementation provided by the Brotli compression library is authoritative.

LZ4

A **deprecated** codec loosely based on the LZ4 compression algorithm, but with an additional undocumented framing scheme. The framing is part of the original Hadoop compression library and was historically copied first in parquet-mr, then emulated with mixed results by parquet-cpp.

It is strongly suggested that implementors of Parquet writers deprecate this compression codec in their user-facing APIs, and advise users to switch to the newer, interoperable LZ4_RAW codec.

ZSTD

A codec based on the Zstandard format defined by RFC 8478. If any ambiguity arises when implementing this format, the implementation provided by the ZStandard compression library is authoritative.

LZ4_RAW

[C65] "Efficient Exploration of Telco Big Data with Compression and Decaying", Constantinos Costa, Georgios Chatzimilioudis, Demetrios Zeinalipour-Yazti, Mohamed F. Mokbel, **Proceedings of the IEEE 33rd International Conference on Data Engineering** (ICDE'17), IEEE Computer Society, pp. 1332-1343, April 19-22, 2017, San Diego, CA, USA, DOI: 10.1109/ICDE.2017.175, ISBN: 978-1-5090-6543-1, **2017**.

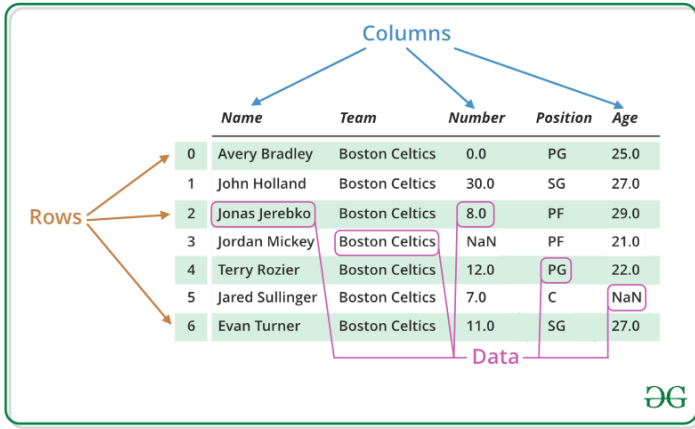
Pandas: Dataframe Only (no data file)

- `sudo pip install pandas`
- `sudo pip install pyarrow`

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

```
python3 b.py
     Name Qualification
0     Jai           Msc
1  Princi            MA
2  Gaurav           MCA
3    Anuj           Phd
```

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).



Pandas operates in-memory, the entire DataFrame must fit into the system's RAM for processing. ☹ Data Processing Scenario (Not storage)

Pandas: CSV into Dataframe

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
print(first, "\n\n\n")
```

```
Team         Boston Celtics
Number                  0.0
Position                 PG
Age                    25.0
Height                  6-2
Weight                180.0
College               Texas
Salary          7730337.0
Name: Avery Bradley, dtype: object
```

Again Main-Memory only ☹

```
wget https://media.geeksforgeeks.org/wp-content/uploads/nba.csv
```

```
$ head nba.csv
Name,Team,Number,Position,Age,Height,Weight,College,Salary
Avery Bradley,Boston Celtics,0.0,PG,25.0,6-2,180.0,Texas,7730337.0
Jae Crowder,Boston Celtics,99.0,SF,25.0,6-6,235.0,Marquette,6796117.0
John Holland,Boston Celtics,30.0,SG,27.0,6-5,205.0,Boston University,
R.J. Hunter,Boston Celtics,28.0,SG,22.0,6-5,185.0,Georgia State,1148640.0
```

Pandas: Dataframe into Parquet Format

```
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

# Create a sample DataFrame
data = {
    "id": [1, 2, 3],
    "name": ["Alice", "Bob", "Charlie"],
    "age": [25, 30, 35]
}
df = pd.DataFrame(data)

# Convert the DataFrame to a PyArrow Table
table = pa.Table.from_pandas(df)

# Write the table to a Parquet file
pq.write_table(table, 'example.parquet')

print("Parquet file written successfully!")
```

```
vgate@vgate:~/pandas$ hexdump -C example.parquet  | head
00000000  50 41 52 31 15 04 15 30  15 2e 4c 15 06 15 00 12  |PAR1...0..L.....|
00000010  00 00 18 04 01 00 09 01  3c 02 00 00 00 00 00 00  |........<.......|
00000020  00 03 00 00 00 00 00 00  00 15 00 15 14 15 18 2c  |...............,|
00000030  15 06 15 10 15 06 15 06  1c 18 08 03 00 00 00 00  |................|
00000040  00 00 00 18 08 01 00 00  00 00 00 00 00 16 00 28  |...............(|
00000050  08 03 00 00 00 00 00 00  00 18 08 01 00 00 00 00  |................|
00000060  00 00 00 00 00 00 0a 24  02 00 00 00 06 01 02 03  |.......$........|
00000070  24 00 26 e4 01 1c 15 04  19 35 00 06 10 19 18 02  |$.&.....5......|
00000080  69 64 15 02 16 06 16 da  01 16 dc 01 26 52 26 08  |id..........&R&.|
00000090  1c 18 08 03 00 00 00 00  00 00 00 18 08 01 00 00  |................|
```

Data Written in efficient format (i.e., binary or compressed binary) on secondary storage ☺

However, there are no row updates => rewriting the whole file is inefficient vs. traditional DBs ☹

So Parquet should be the input / output from a typical DB (rather than JSON, XML, TSV, CSV) as opposed to the data layer.

Parquet DB Connectivity

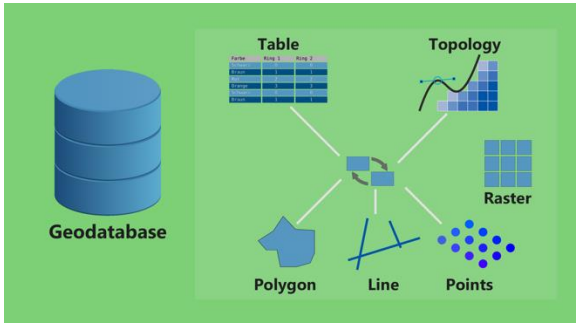
- **pg_parquet: An Extension to Connect Postgres and Parquet**

```
-- copy the table to a parquet file
CREATE TABLE product_example ( … ));
COPY product_example TO '/tmp/product_example.parquet' (format 'parquet', compression 'gzip');
```

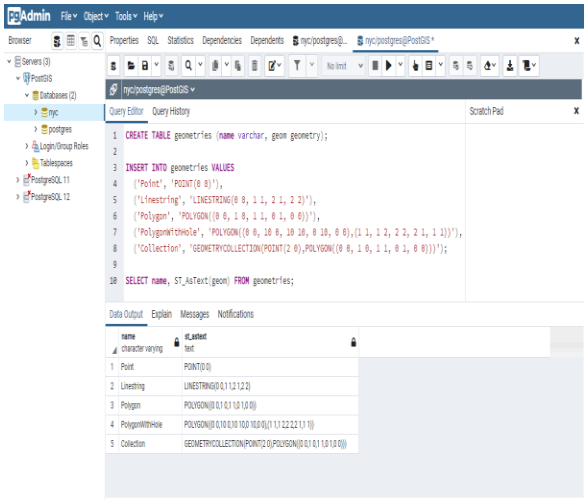
- **SQL Server 2022 (16.x)** can virtualize data from parquet files. This process allows the data to stay in its original location, but can be queried from a SQL Server instance with T-SQL commands, like any other table.
  - **Polybase:** your SQL Server instance to query data with T-SQL directly from SQL Server, Oracle, Teradata, MongoDB, Hadoop clusters, Cosmos DB, and S3-compatible object storage without separately installing client connection software.

Geo (Spatial) Formats

- **GeoParquet:** GeoParquet (Open Geospatial Consortium) adds interoperable geospatial types (e.g. Point, Line, Polygon) to **Parquet**.
- **ESRI:** ESRI is a commercial vendor for GIS data, in particular analysis of geospatial data objects either as online maps or using desktop tooling along with cloud pipelines. Huge amount of formats for **ESRI File Geodatabase (GDB)**
- **PostGIS/Postgres (Relational)**
- **PostGeese/DuckDB (**Embedded OLAP)
- **Apache Sedona:** This is a cluster computing system for processing large-scale spatial data.



GeoParquet



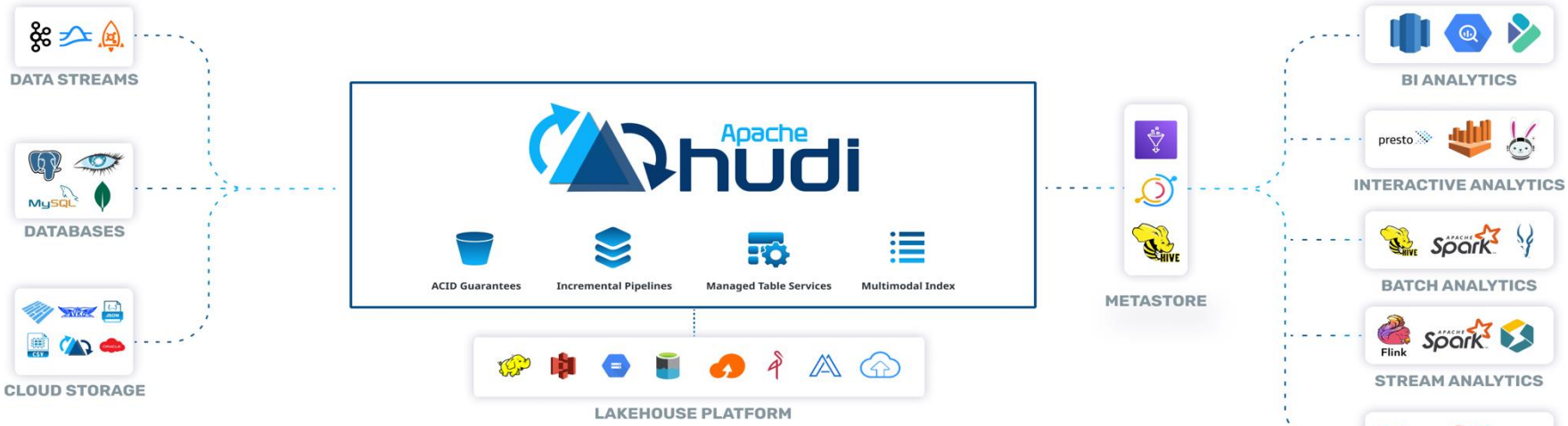
PostGIS/Postgres

Delta Lake Format

- A **data lake** is a system or repository of data stored in its natural/raw format, usually object blobs or files.
  - Structured (rows and columns), Semi-structured (CSV, logs, XML, JSON), and unstructured data (emails, documents, PDFs), and binary data (images, audio, video).
- **Parquet data lakes** are inefficient because they require rewriting entire Parquet files even to update only a single row/
- The **Delta Table format is** built on top of **Parquet**, but adds advanced features like **ACID transactions**, **versioning**, and **schema evolution**.
  - It's part of the **Delta Lake** project (originally developed by Databricks)

Lakehouse Table Formats

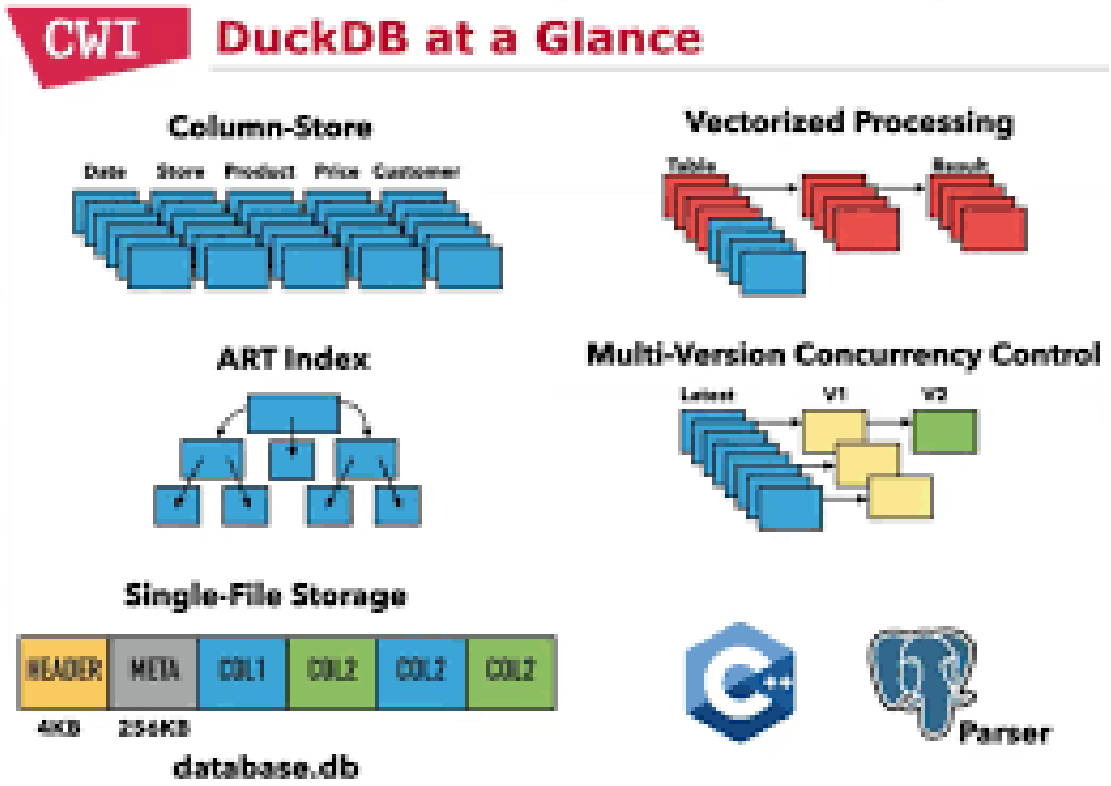
- Lakehouse table formats:
  - Delta Lake, Apache Iceberg and Apache Hudi
  - (can be thought like Parquet files for data lakes allowing row updates with ACID properties)



[Deletion Vectors: No-Regrets Row-Level Updates in Delta Lake](#) by Bart Samwel (Databricks).

DuckDB – Embedded, Columnar, OLAP

- **DuckDB: A Columnar OLAP Database (SIGMOD'19)**

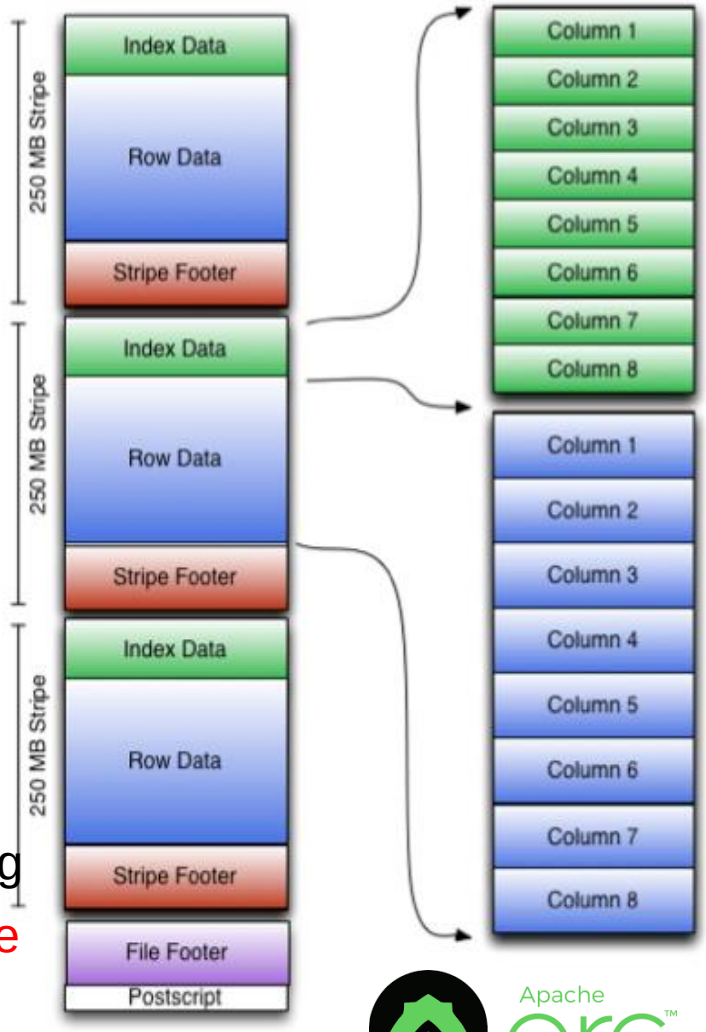


Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1981–1984. https://doi.org/10.1145/3299869.3320212

Optimized Row Columnar (ORC Files for Big Data)

Similar to Parquet but largely forgotten due to Parquet

- **ORC (Optimized Row Columnar) is a popular columnar storage format used in SQL-based big data systems like**
  - Apache Hive, Apache Spark, and Presto. However, traditional relational databases (e.g., MySQL, PostgreSQL, SQL Server) do not natively support ORC tables. ORC is mainly used in distributed computing environments like Apache Hadoop.
- An **ORC file contains** groups of row data called **stripes**, along with auxiliary information in a file footer.
  - At the end of the file a postscript **holds compression parameters** and the **size of the compressed footer.**



https://cwiki.apache.org/confluence/display/hive/languagemanual+orc

CREATE TABLE my_table (id INT, name STRING) STORED AS ORC;

PAX (Parallel Adaptive eXchange) (Hybrid Row/Column File Format)

- **PAX** is a **storage format** designed to optimize the performance of columnar data processing, but it's somewhat less well-known compared to other columnar formats like **ORC** and **Parquet**.
  - PAX is often considered a **hybrid** between **row-based** and **column-based** formats.
  - It organizes data in **blocks**, and within each block, it stores data for multiple columns, but each block has multiple **tuples (rows)** of data.
  - https://www.the-paper-trail.org/post/2013-01-30-columnar-storage/