Storage Models

Storage Models

Database Workloads

On-Line Transaction Processing (OLTP)

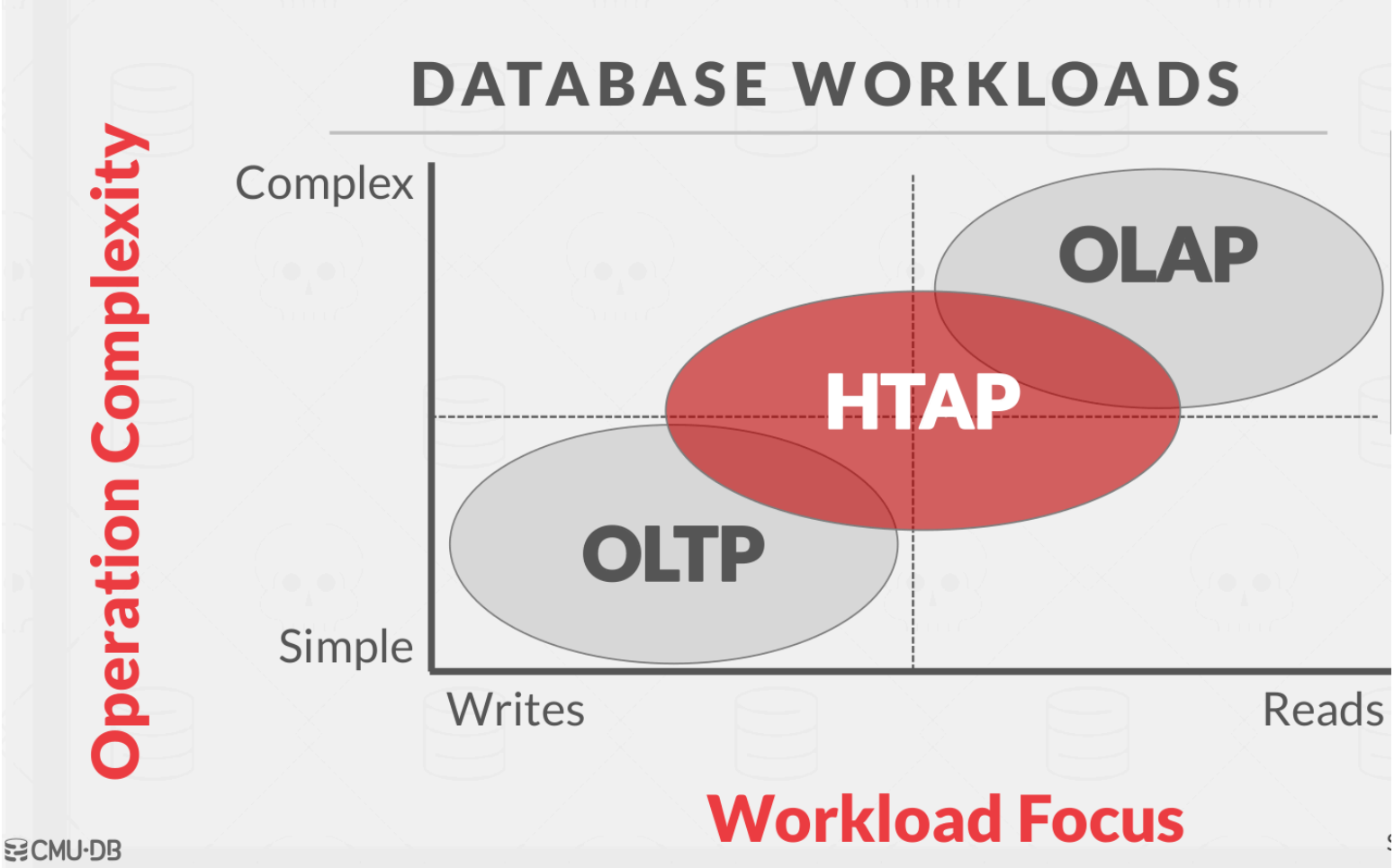
Fast operations that only read/update a small amount of data each time

On-Line Analytical Processing (OLAP)

Complex queries that read a lot of data to compute aggregates

Hybrid Transaction + Analytical Processing (HTAP)

OLTP + OLAP together on the same database instance

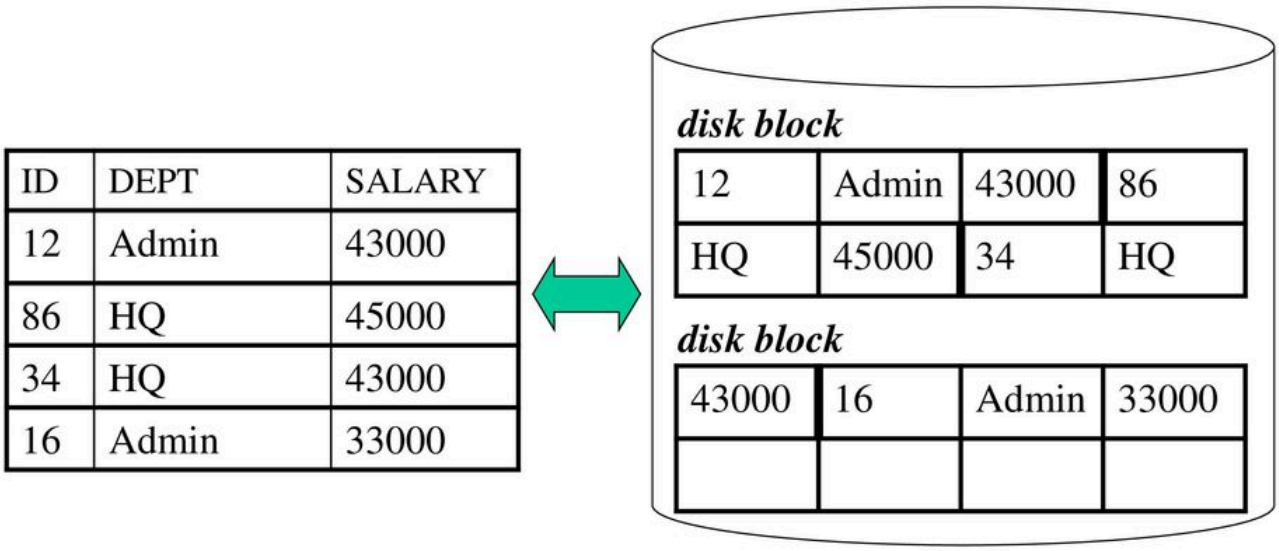


N-ary Storage Model (NSM)

all records in a DB relation are stored together. Assuming the relation is N-ary, the storage is a sequence of N-tuples.. In table parlance: Table data is stored row-by-row (with N being the number of table columns).

N-ary storage model (NSM)

- Records stored on disk in same way they are seen at the logical (conceptual) level

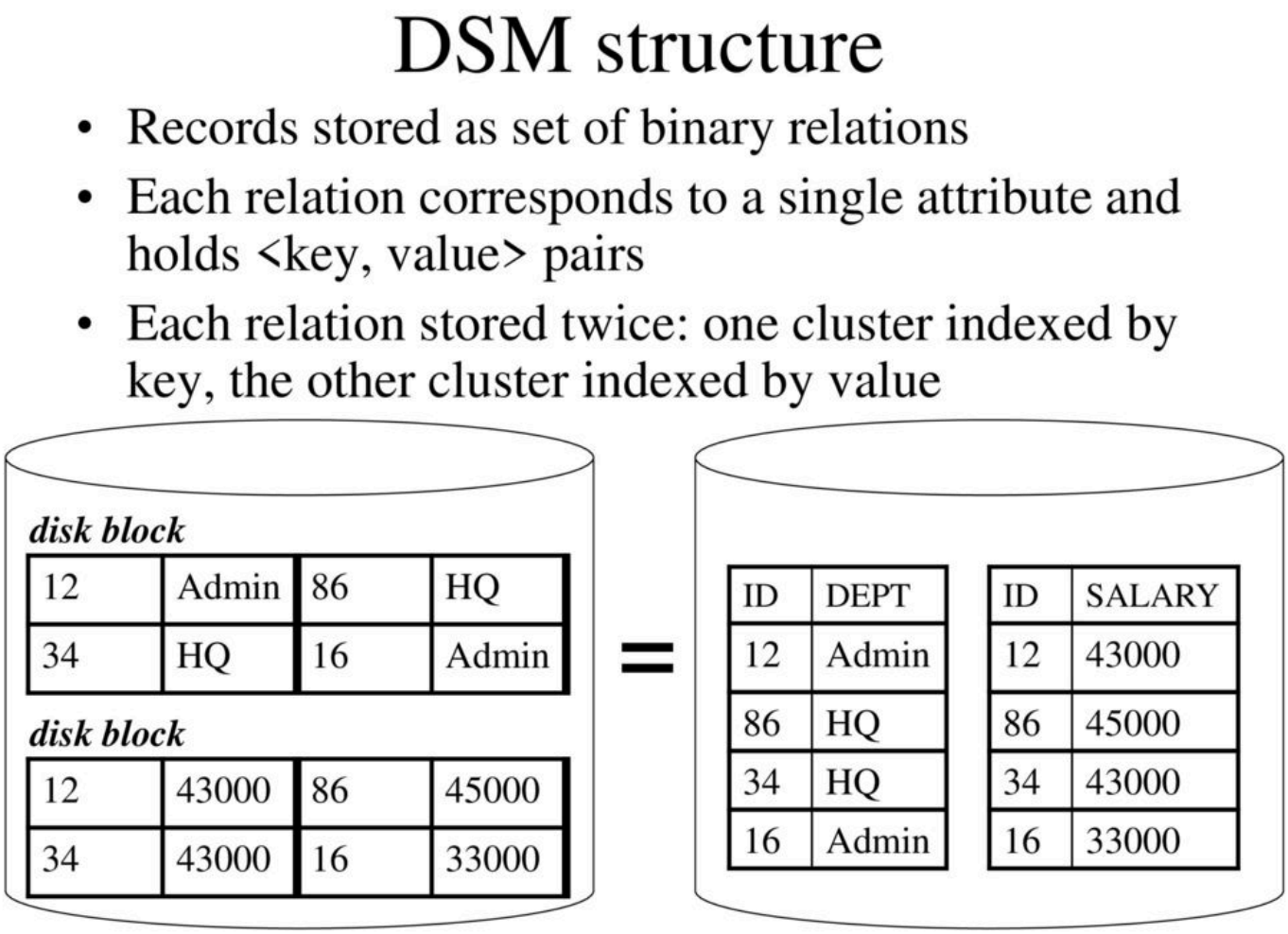


Ideal for OLTP workloads where queries are more likely to access individual entities and execute write-heavy workloads.

NOT Ideal for OLAP because we load unnecessary data that might not be needed.

Advantages
→ Fast inserts, updates, and deletes.
→ Good for queries that need the entire tuple (OLTP).
→ Can use index-oriented physical storage for clustering.
Disadvantages
→ Not good for scanning large portions of the table and/or a subset of the attributes.
→ Terrible memory locality in access patterns.
→ Not ideal for compression because of multiple value domains within a single page.

Decomposition Storage Model (DSM)



Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

Advantages
→ Reduces the amount wasted I/O per query because the DBMS only reads the data that it needs.
→ Faster query processing because of increased locality and cached data reuse. → Better data compression.
Disadvantages
→ Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching/reorganization.

Hybrid Storage Model (PAX)

Column & Row Store

In a Relational database, the data for either type of activity is organized into rows and columns that form a table (or tables) to show the relationship between data points. The data can be stored in two different ways:

- In a columnar database (Column Store), all the data is grouped together by column.
- In a row-oriented database (Row Store), all the data is grouped together by row.

Columnar databases excel at handling analytical workloads (OLAP), while row-oriented databases are better suited for Transactional workloads (OLTP).

Column Store

Data is stored column by column. Each column is stored as a contiguous block of data.
Efficient for Analytical Workloads (OLAP):
- Optimized for operations that aggregate or analyze data over large numbers of rows (e.g., `SUM`, `AVG`, `COUNT`).

- Since only relevant columns are read, fewer I/O operations are needed for such queries.
2. **Compression**: Columns often contain similar data types or repeating values, enabling efficient compression.
3. **Vectorized Operations**: Processing can be faster due to contiguous storage of similar data types.
Examples: Apache Parquet, Google BigQuery, Amazon Redshift, SAP HANA.

Row Store

Data is stored row by row. Each row contains all the data for a single record, grouped together.
Efficient for Transactional Workloads (OLTP)
- Optimized for operations that access complete rows (e.g., `INSERT`, `UPDATE`, and `DELETE`).
- Ideal for scenarios where the application needs to handle individual records frequently.
2. **Good for Mixed Queries**: Supports operations across rows with moderate efficiency.
3. **Simple Indexing**: Indexing can be straightforward and optimized for row lookups.
Examples: MySQL (InnoDB), PostgreSQL, Oracle DB.

Column & Row Store - Comparison

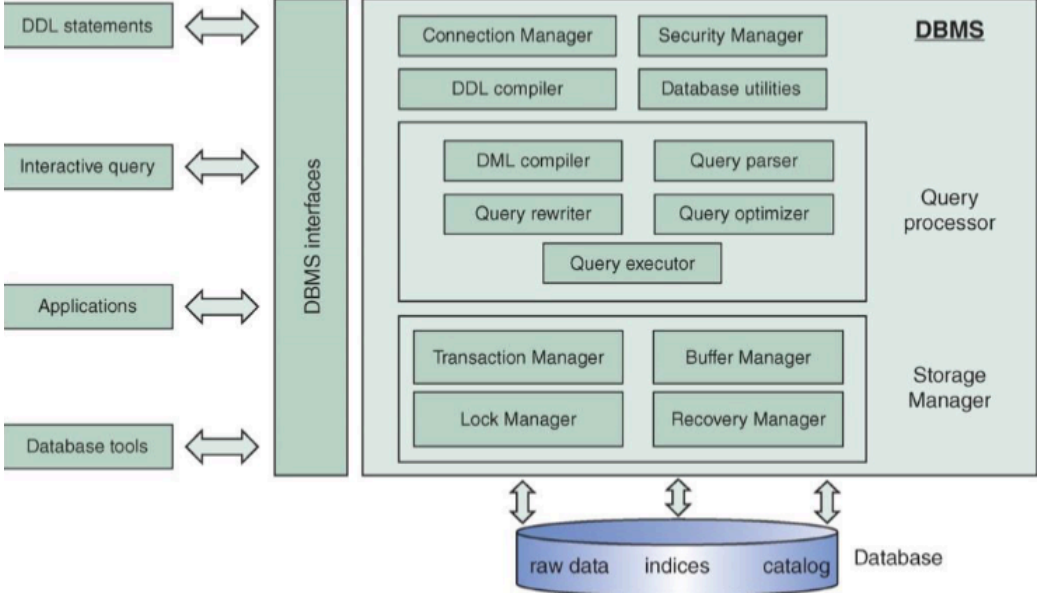
Columnar database	Row-oriented database
✓ High performance for analytical queries	✗ Inefficient for analytical queries
✓ Efficient storage and compression	✗ Suboptimal storage and compression
✓ Ease of scalability	✗ Complex and expensive to scale
✗ Slower for transactional workloads	✓ Efficient for transactional workloads
✗ Complexity in schema design	✓ Easy to understand and implement
✗ Higher overhead for updates and inserts	✓ Easier modification of data

Main-Memory DBMSs

What if all data is in RAM?
• No disk while executing tasks → never need to wait for disk
• Note: There is still disk for persistent storage!
Don’t have to optimize data accesses for disk
→ no buffer manager
→ no need for tuples in slotted-pages in-memory, have direct access to tuples
→ indexes aren’t kept in pages either & usually aren’t persisted
→ aim better cache utilization/accesses
→ row-store for OLTP, column-store for OLAP (*also valid for disk*)
→ cache-conscious index structures *(also valid for disk*)
→ query compilation that generates cache-conscious code (*also valid for disk*)

Downsides
Startups/restarts are expensive
• need to load everything in-memory
• indexes has to be rebuilt
• recovery takes longer if not done carefully
There comes a day where your data doesn’t fit in memory anymore
• most main-memory systems added support for efficiently accessing cold/old data from disk later
because customers ask for it

DBMS Architecture



Query Processor

parsing an SQL query into a plan optimizing that plan executing the optimized plan using relational operators (e.g., join, group by, etc.)

Storage Manager

manages & keeps track of the data read from persistent storage into main memory:
• better optimizations because of application knowledge, e.g., replacement policy, prefetching
• you may want to force-flush some data to disk (e.g., log)
• you may not want to ever flush some data (e.g., aborted requests)