THE UNIVERSITY *of* EDINBURGH

Advanced Database Systems

Spring 2026

Lecture #07:

Storage Models & Compression

Database Workloads

On-Line Transactional Processing (OLTP)

- Fast, simple operations that handle small amounts of data per transaction

On-Line Analytical Processing (OLAP)

- Complex queries that read large amounts of data to compute aggregates

Hybrid Transactional and Analytical Processing (HTAP)

- Combines OLTP and OLAP on the same database instance
- Real-time analytics on live operational data w/o moving data between systems (e.g., real-time fraud detection)

OLTP: On-Line Transactional Processing

High volumes of real-time transactions

- Simple queries that read/update a small amount of data related to a single entity

Focused on operational tasks

- E.g., order processing, payments, inventory

Key features

- Short queries
- High concurrency
- Balanced read-write operations

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revision AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
INSERT INTO revisions
VALUES (?,?,?)
```

OLAP: On-Line Analytical Processing

Designed for data analysis and reporting

- Complex queries that read large portions of the database spanning multiple entities

Get business insights from historical data

- E.g., trend analysis, decision-making insights
- OLAP runs on data collected from OLTP apps

Key features

- Long-running queries over many tables
- Read-heavy
- Aggregated data

```
SELECT COUNT(U.lastLogin),
       EXTRACT(MONTH FROM
           U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(
        MONTH FROM U.lastLogin)
```

Observation

The relational model does not require the DMBS to store all tuple attributes in a single page

This may **not** actually be the best layout for some workloads

The DBMS can store records in different ways that are better for either OLTP or OLAP workloads

Storage Models

Storage model specifies how tuples are physically arranged on disk and in memory

- Can have different performance characteristics based on the target workload (OLTP vs. OLAP)
- Influences the design choices of the rest of the DBMS

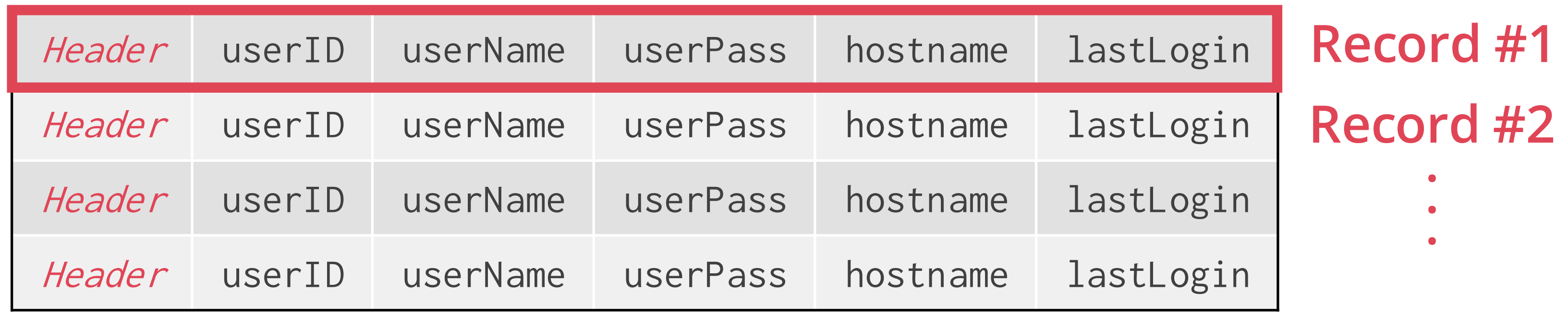
Common models

- Row Storage Model
- Column Storage Model
- Hybrid Storage Model (PAX)

Row Storage Model

Stores all attributes of a tuple (row) contiguously in memory and on disk

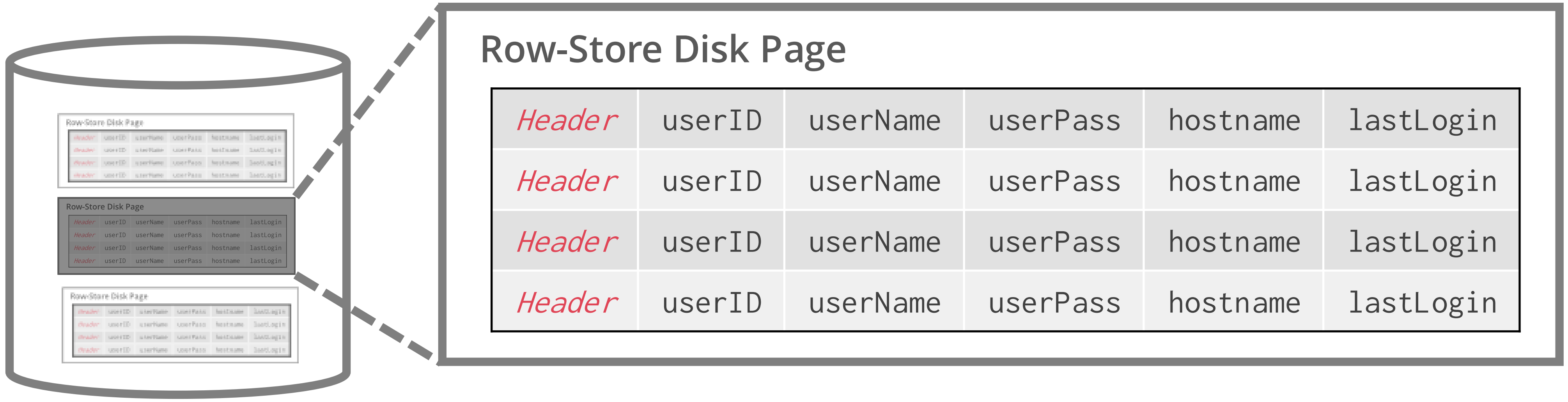
Ideal for OLTP workloads with frequent individual entity access and updates



Row Storage Model

Stores all attributes of a tuple (row) contiguously in memory and on disk

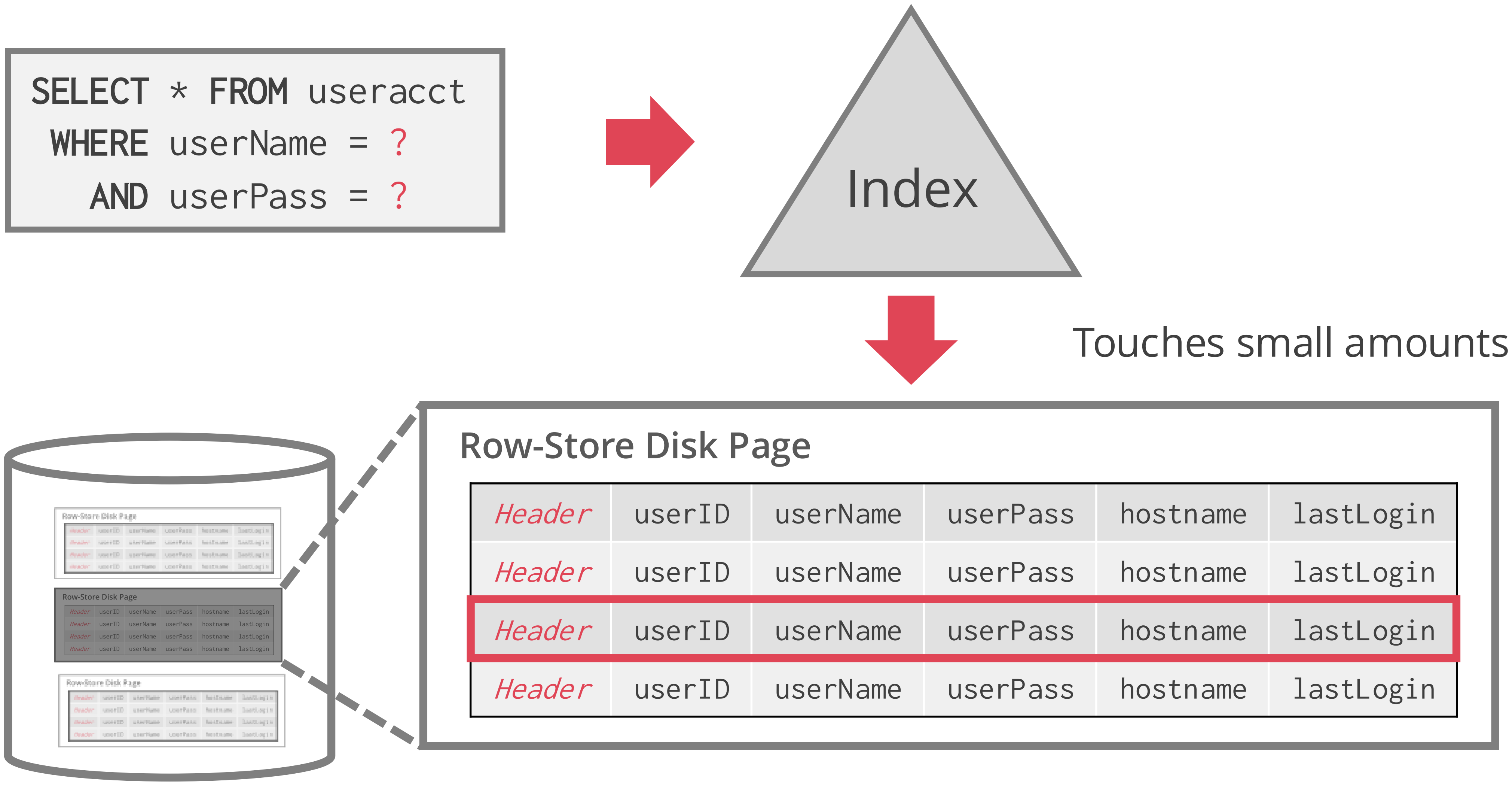
- Fixed-length and variable-length attributes stored contiguously in a single slotted page
- Record ID = (page ID, slot ID) is how the DBMS uniquely identifies a physical tuple



Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

ROW STORAGE MODEL

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```



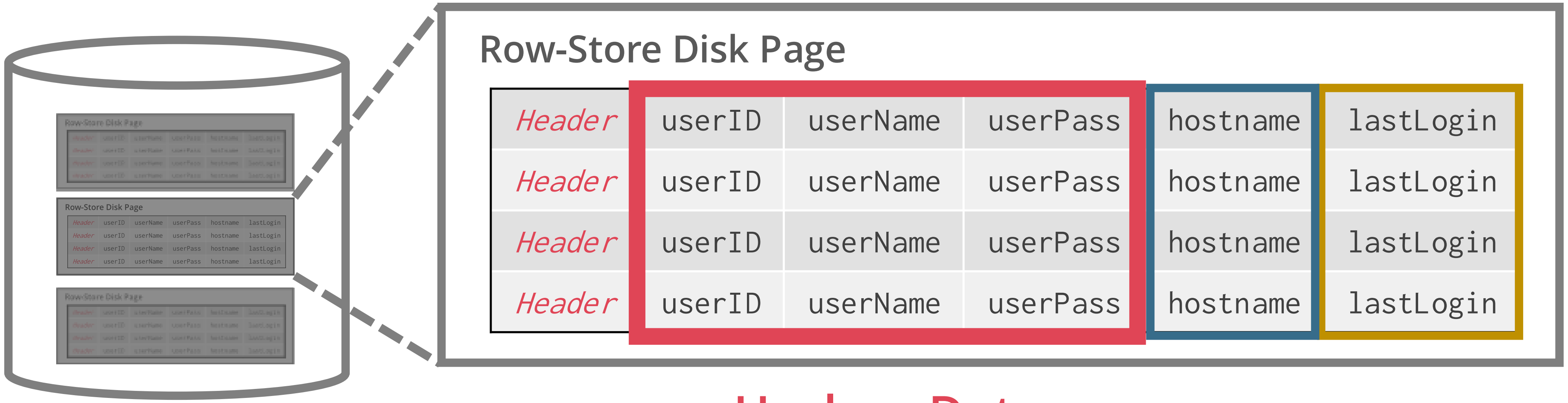
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

Row Storage Model

```
SELECT COUNT(U.lastLogin),
       EXTRACT(MONTH FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(MONTH FROM U.lastLogin)
```

Scans entire relation

Most read data not needed



Useless Data

Row Storage Model

Advantages

- Fast access to all attributes of a single tuple. Fast inserts, updates, and deletes
- Ideal for OLTP workloads involving individual tuple operations
- Can use clustered indices in variant A for storing data *covered later this week*

Disadvantages

- Reading entire rows for queries involving only a few attributes leads to unnecessary I/O
- Not good for reading large portions of the table and/or a subset of the attributes (OLAP)
- Terrible memory locality in access patterns
- Not ideal for compression because of multiple value domains within a single page

Columnar Storage Model

Store a single attribute for all tuples contiguously in memory and on disk

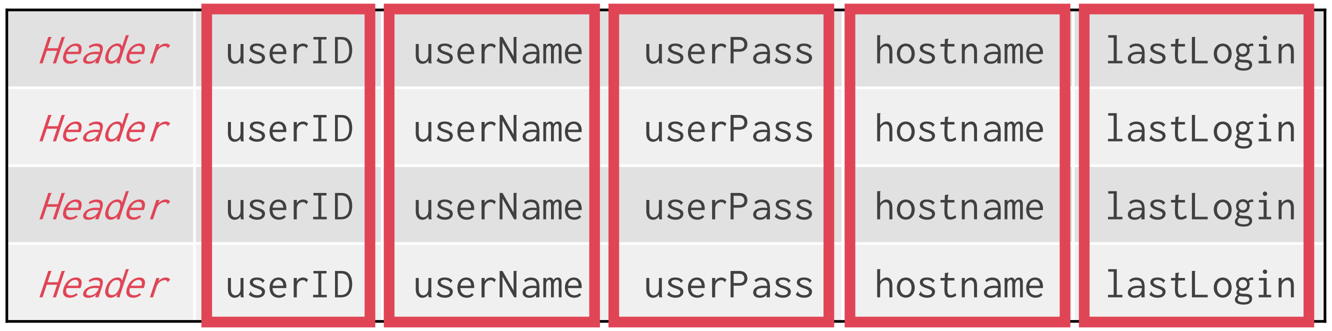
Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table’s attributes

DMBS is responsible for combining/splitting a tuple’s attributes when reading/writing

Columnar Storage Model

Store a single attribute for all tuples contiguously in memory and on disk

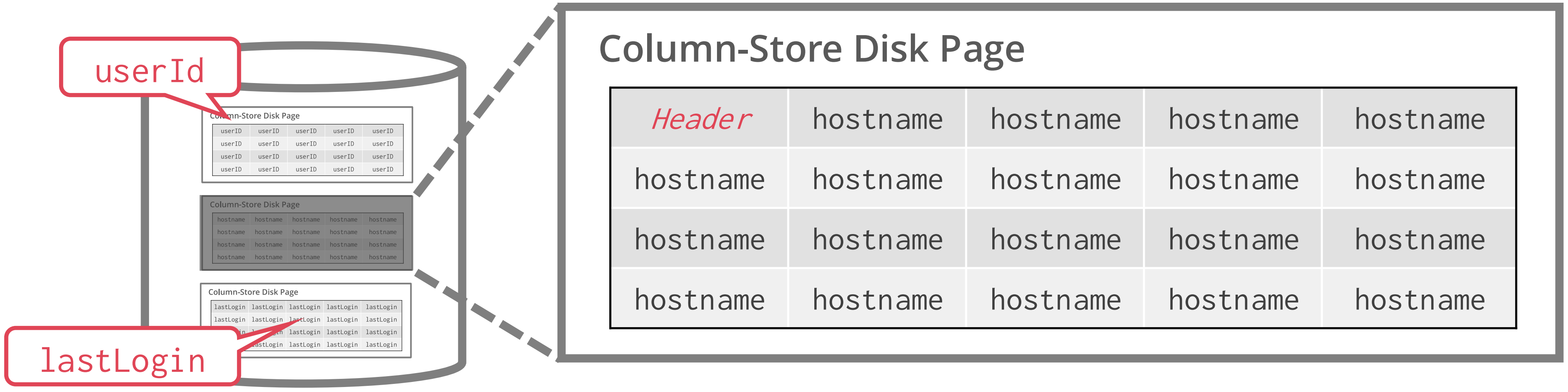
- Store attribute and metadata (e.g., nulls) in separate arrays of **fixed-length** values
- Identify physical tuples using **offsets** into these arrays
- Convert variable-length data into fixed-length values using **dictionary compression**



Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

Columnar Storage Model

Store a single attribute for all tuples contiguously in memory and on disk

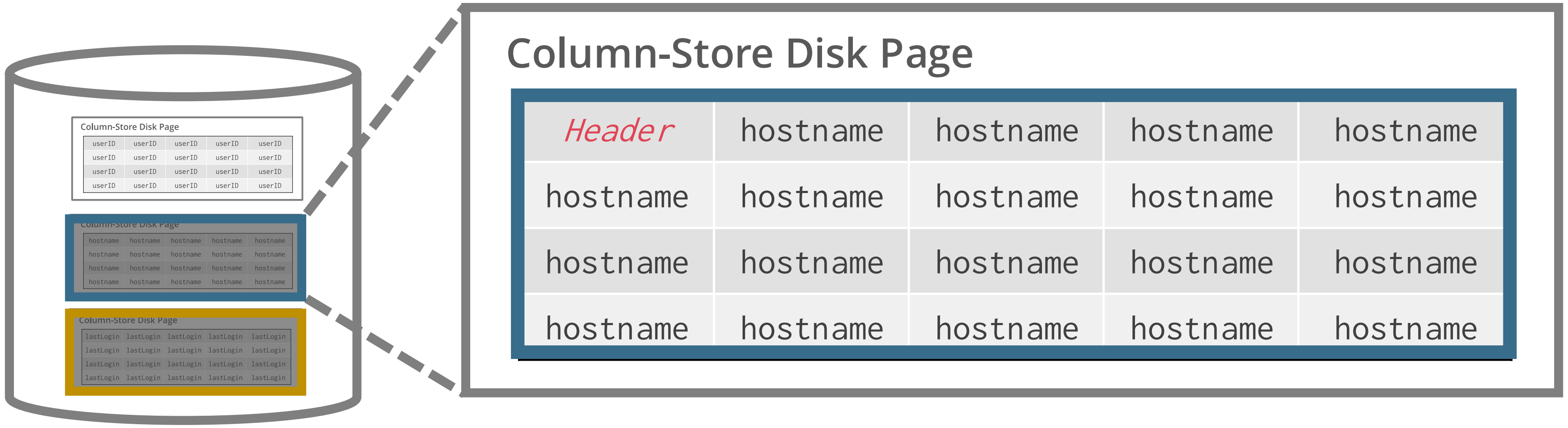


Column-Store Disk Page

Header	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname

Columnar Storage Model

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Header	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname

Columnar Storage Model

Advantages

Reduces the amount of wasted I/O because the DBMS only reads the data that it needs (free projection pushdown)

Faster query processing because of increased cache locality

Better data compression

Disadvantages

Slow for point queries, inserts, updates, and deletes because of tuple splitting / stitching

Hybrid Storage Model (PAX)

OLAP queries rarely access a single column in isolation

- During query execution, the DBMS must get other columns and reconstruct the original tuple

Ideally, we want columnar benefits (compression, efficient processing) without losing the speed of accessing related data together

Partition Attributes Across (PAX) is a hybrid storage model that vertically partitions attributes within a database page

- Examples: Parquet, ORC, and Arrow
- The goal is to combine the performance benefits of columnar storage with the spatial locality advantages of row storage

HYBRID STORAGE MODEL

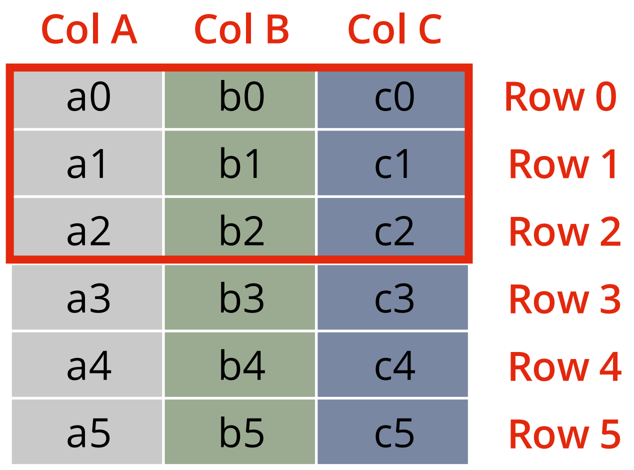
Horizontally partition data into **row groups**

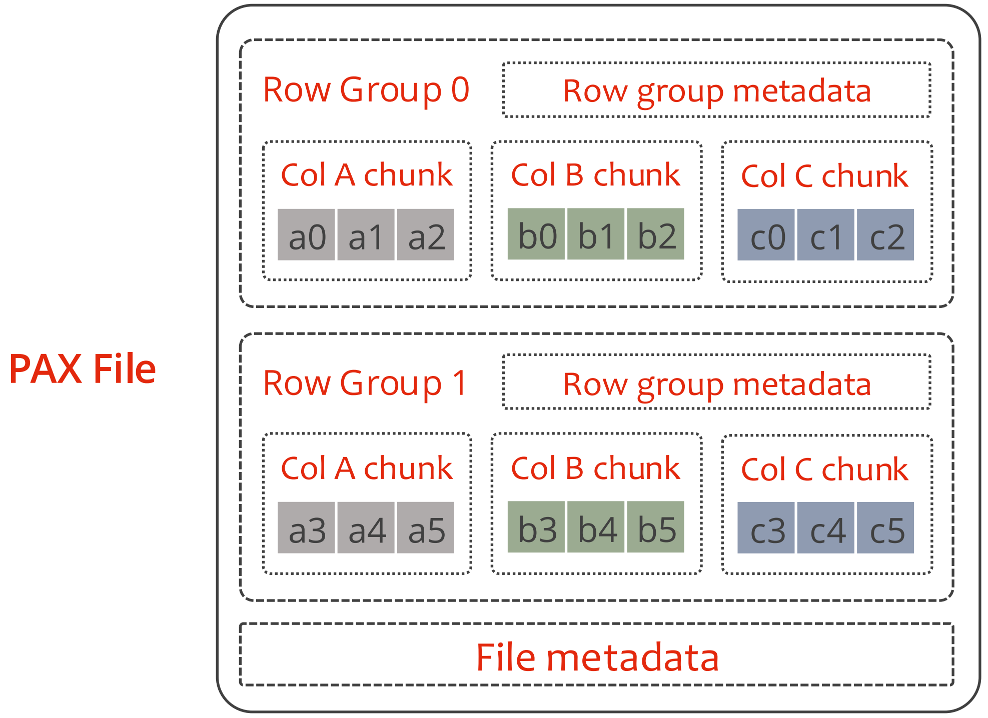
Vertically partition row groups into **column chunks**

Global metadata directory contains offsets to the file's row groups

- This is stored in the footer if the file is immutable (Parquet, Orc). Why?

Each row group contains its own metadata header about its contents

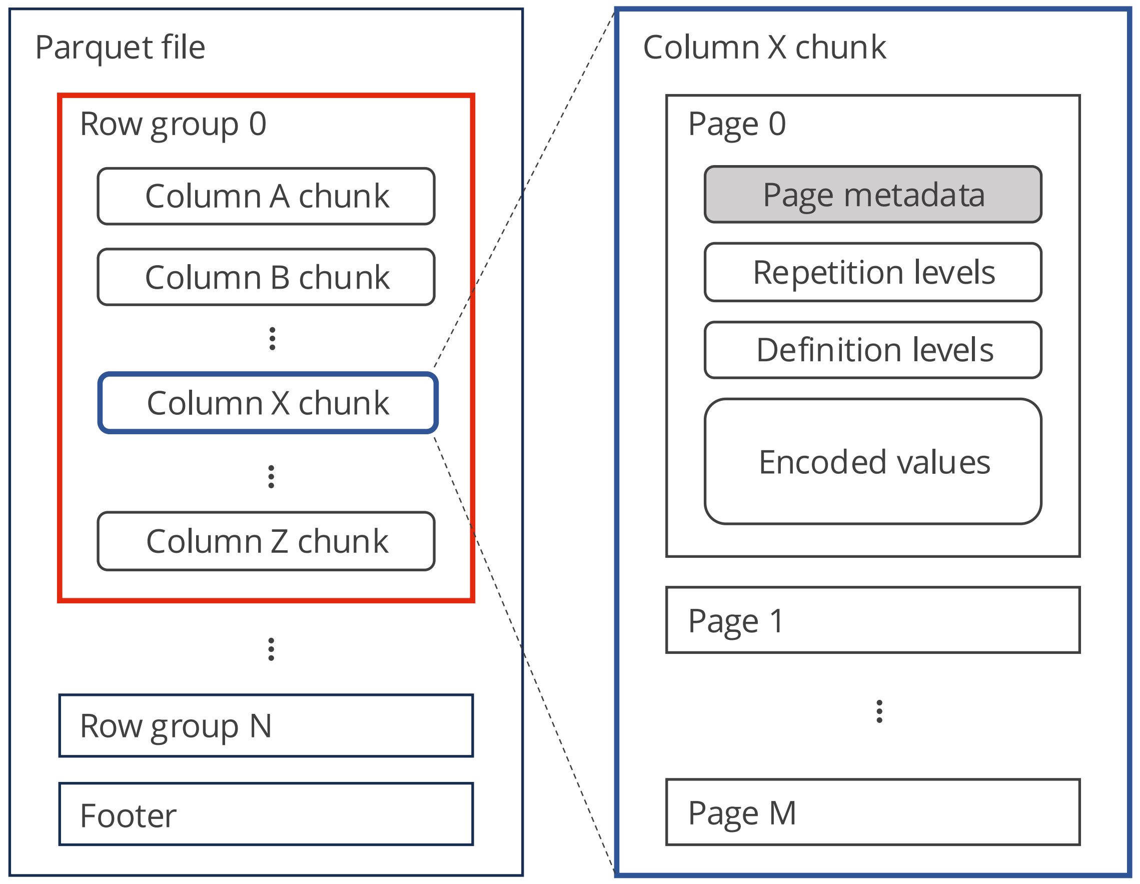




Parquet File Format

Data organisation

- Row groups (default 128MB)
- Column chunks
- Pages (default 1MB)
  - Metadata (min, max, count)
  - Rep/def levels (for nested data)
  - Encoded values
- Footer
  - File, row group, and column metadata
  - (e.g., schema, count, row group offsets)



Parquet File Format

Columnar storage speeds up queries by reading only needed data

High compression reduces file size

Predicate pushdown speeds up queries by skipping irrelevant data based on statistics

Parallel processing: row groups enable distributed/parallel processing

Rich metadata: stores statistics, encoding info, schema (so parsing is fast)

Schema evolution: add/modify columns without rewriting the entire file

Widely used in big data platforms (Spark, Hive, Presto) and storage systems

Compression in DBMS

Why compression?

- Reduces storage and DRAM requirements
- Improves system performance by increasing data per I/O
- Must be **lossless** → any lossy compression must be performed by application

Key trade-off

- Speed vs. compression ratio → lower I/O vs. higher CPU cost

Impact on query execution

- Compressed pages reduce I/O overheads
- May increase CPU cost due to decompression
- Sometimes queries can be run directly on compressed data

Naïve Compression

Uses general-purpose algorithms (e.g., zlib, Snappy, Zstd)

Compresses data block by block without understanding its meaning

Decompression required before reading or modification → limits efficiency

Limited scope: only considers data given as input, not high-level semantics

Lower compression ratio on heterogeneous data

Columnar Compression

Run-length encoding

Supress duplicates, e.g., 2, 2, 2, 3, 4, 4, 4, 4, 4 ➔ 2x3, 3x1, 4x5

Good for mostly sorted integers or categorical data

Delta encoding

Encode differences, e.g., 2, 3, 4, 5 ➔ 2, +1, +1, +1,

Pairs well with run-length encoding, e.g., 2, +1, +1, +1 ➔ 2, +1x3

Good for mostly sorted numeric data (floats)

Bit packing

Use fewer bits for short integers

Pairs well with delta coding

Good for limited precision data

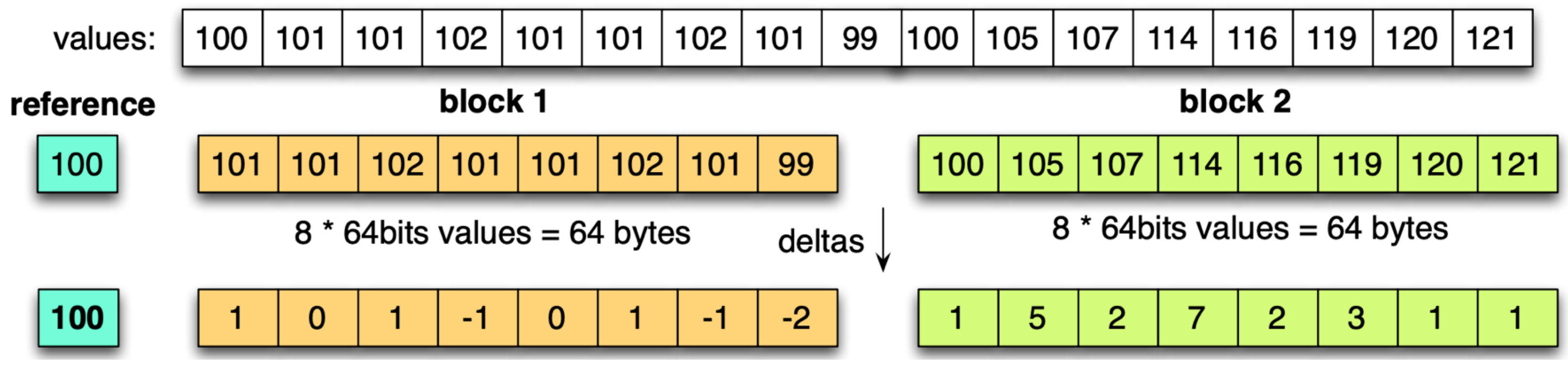
Dictionary encoding

Replace frequent values with smaller fixed-length codes

Maintain a mapping from the codes to the original values

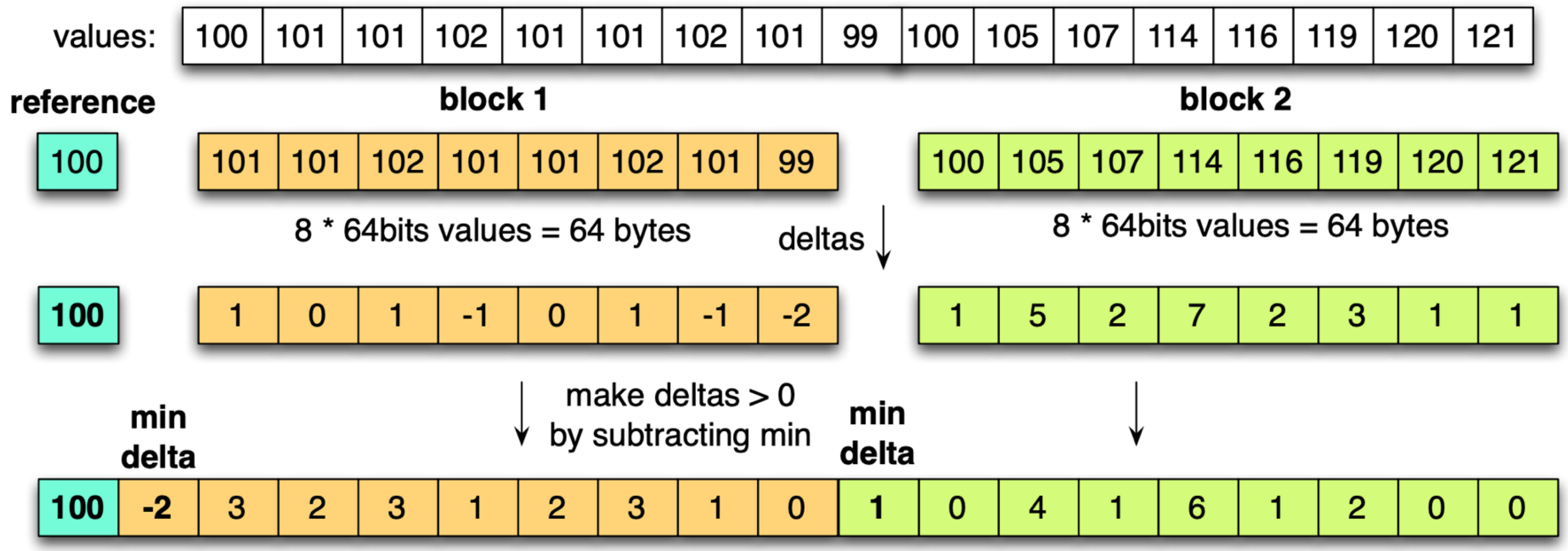
Good for long, frequent strings

Delta Encoding in Parquet

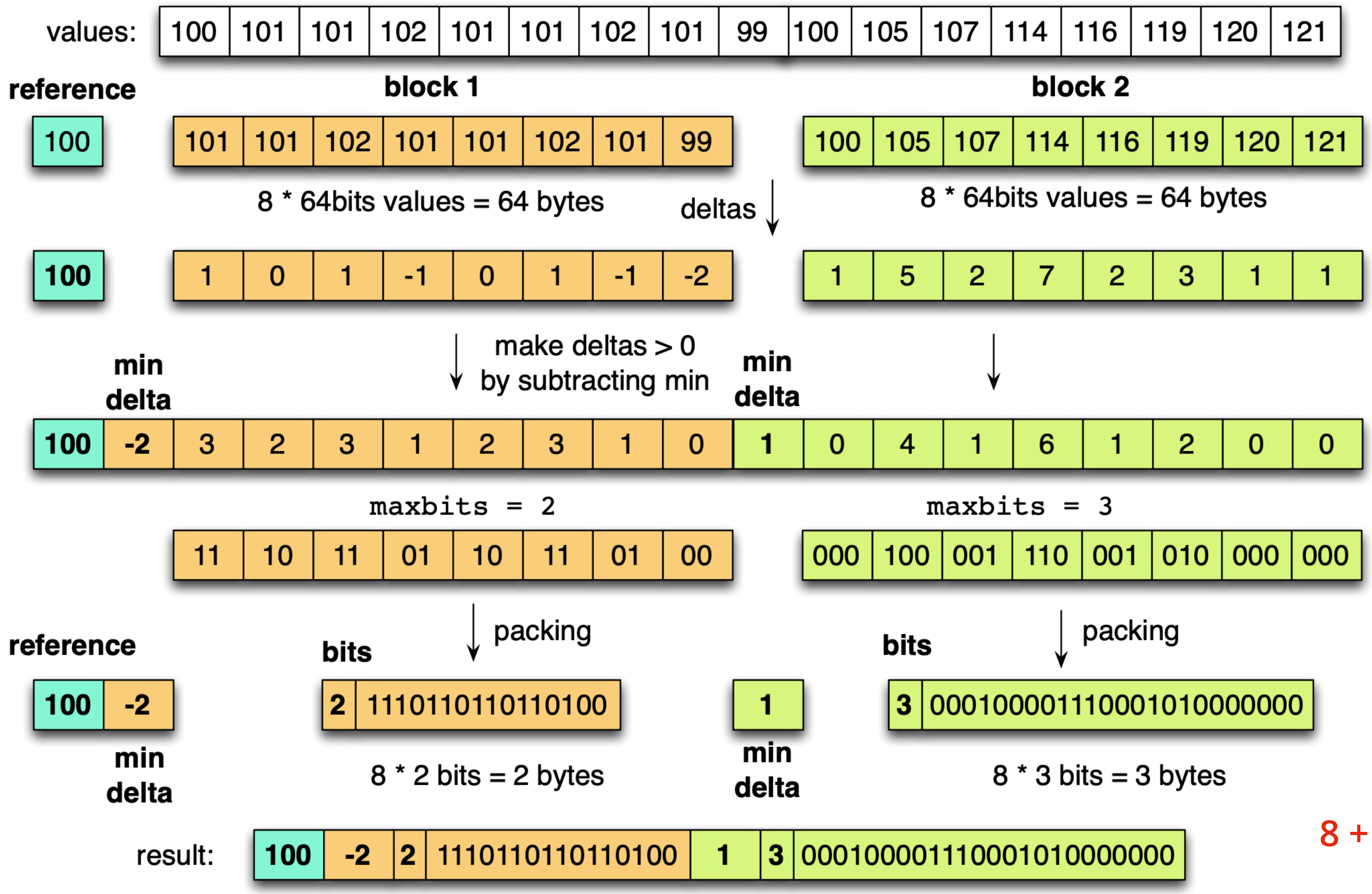


Source: "Efficient Data Storage for Analytics with Apache Parquet 2.0", Julian Le Dem

Delta Encoding in Parquet



Delta Encoding in Parquet



17 x 8 = 136 bytes

8 + 8 + 1 + 2 + 8 + 1 + 3 = 31 bytes

DICTIONARY ENCODING

Concept

- Replaces frequent, long values (e.g., strings) with smaller fixed-length integers
- Uses a dictionary from the integers to the original values
- Most widely used compression technique in DBMSs

Benefits

- Reduces data size
- Eliminates variable-length data
- Does not require pre-sorting
- Improves storage & access efficiency

Original Data

City
New York
London
Paris
New York
Tokyo
London

→

Compressed Data

City
1
2
3
1
4
2

Code	Value
1	New York
2	London
3	Paris
4	Tokyo

Dictionary

Conclusion

Important to choose the right storage model for the target workload

- OLTP = Row store
- OLAP = Column store

Modern column stores use the hybrid storage model and data compression

- Some compressions can be directly operated on, e.g., RLE and dictionary encoding

Apache Parquet

- Columnar storage format optimised for efficient data compression and fast analytical queries on large datasets