Intelligent techniques for processing large and structured data

Lecture 3

Faculty of Mathematics and Computer Science
Babeș-Bolyai University

Sergiu Limboi, PhD Teaching Assistant

Motto:"When data becomes large, infrastructure becomes the algorithm."

Processing Large Structured Data: Architectures, Formats, and Scalable Analytics

AGENDA

- Warm-Up
- Industry reality
- When Does Data Become “Large”?
- The Three Bottlenecks of Large Data Systems
- Data Storage Formats
- Data Partitioning
- Distributed Processing
- Modern Data Architectures
- Teamwork time
- Case study: Uber
- Case study: Netflix
- Key takeaways

Warm-Up

Faculty of Mathematics and Computer Science

Warm-Up

Go to www.menti.com and enter the code **2244 0285**

or use the QR code

Industry reality

Faculty of Mathematics and Computer Science

Industry reality

- What happens if you load a 200GB dataset with pandas?
  - Works normally
  - Slower but works
  - Crashes
  - Depends on GPU

- Pandas is an in-memory system.
  - Data must fit entirely in RAM.

RAM (16GB)

200 GB

Industry reality

- Most ML tutorials assume: dataset → memory → model

- A real pipeline look like

When Does Data Become “Large”?

Faculty of Mathematics and Computer Science

When Does Data Become “Large”?

- Data becomes large when it cannot be processed efficiently on a single machine.

- Important criteria:
  - Data does not fit in RAM
  - Data cannot be processed in acceptable time
  - Data requires distributed storage

- Memory required = rows x columns x bytes

When Does Data Become “Large”?

DATASET

Rows: 1 billion
Features: 20
Data type: float64 (8 bytes)

MEMORY

1B X 20 X 8 = 1,000,000,000 × 160 bytes= 160 GB

- Laptop RAM: 16 GB
- The dataset is 10x larger than MEMORY → system becomes unusable

The Three Bottlenecks of Large Data Systems

Faculty of Mathematics and Computer Science

The Three Bottlenecks of Large Data Systems

- Large-scale systems are limited by **three physical resources**.
  - Memory
  - CPU computation
  - Disk I/O

Memory limit

- Most Python tools assume in-memory processing.

```
df = pd.read_csv("data.csv")
```

→ Pandas loads the **entire dataset into RAM**.

- If the dataset > RAM → Memory error → Kernel died

Memory limit

- Memory hierarchy

Level	Speed
CPU cache	Extremely fast
RAM	Fast
SSD	Slower
HDD	Very slow

- Disk access can be millions of times slower than cache.

CPU Complexity

- Some operations grow very quickly in computational cost.
- Example: Large joins become expensive.

CUSTOMERS
100M rows

JOIN

TRANSACTIONS
100M rows

- Worst case complexity: $O(n \times m)$
- Processing may take **hours or days.**

CPU Complexity

- Optimized joins: hash join, merge join.
- But even optimized joins require sorting, shuffling, hashing
- In distributed systems, this requires network communication.

Disk I/O Bottleneck

- Reading data from disk is often the **slowest stage**.

Dataset size
200GB

+

SSD speed
500MB/s

Time to read 200GB / 500MB/s ≈ 400 seconds ~7 minutes

Before processing even starts !!!!

Disk I/O Bottleneck

- If ingredients are far away → cooking slows down.

Data Storage Formats

Data storage formats

- Storage format drastically changes performance !!!

- Row storage (CSV)
  - Data stored row-by-row.

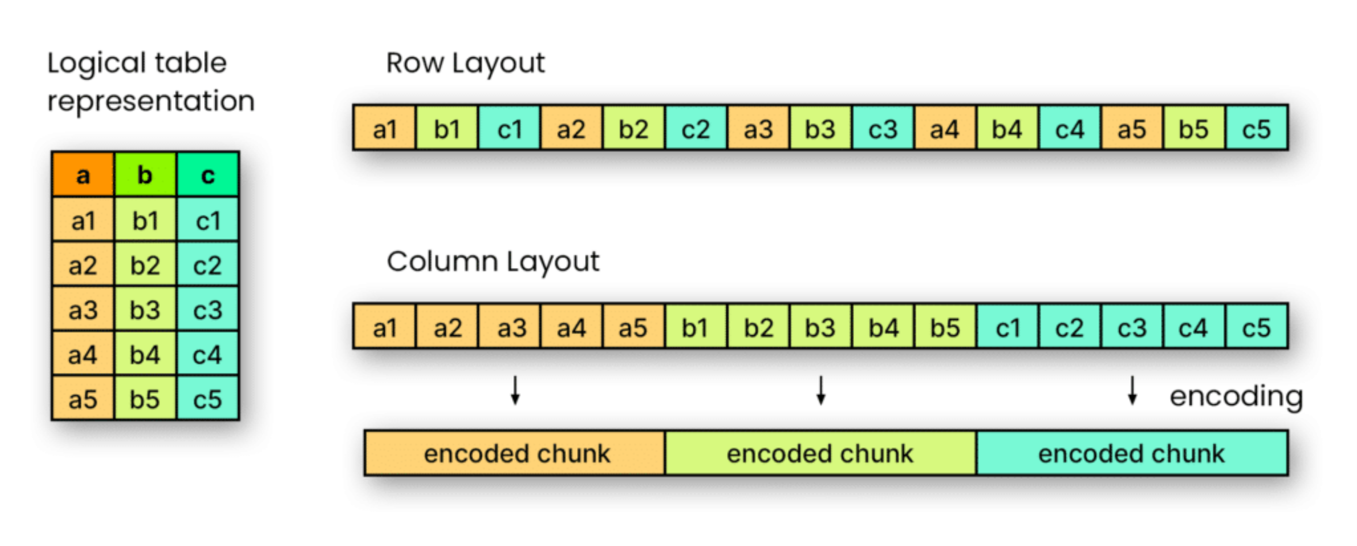
CSV

id	name	price	date
1	Alice	10	2024
2	Bob	20	2024

  - If we query only the price column, the CSV systems must scan every row, and during that scan they must parse every column→ very inefficient
  - No schema - Types are not stored.
  - Large size- CSV cannot compress efficiently.

Data storage formats

- Column Storage (Parquet)
  - Data stored column-by-column.
  - Column data is stored in compressed blocks (encoded chunks).



Data storage formats

- Parquet
  - advantages:
    - Read only needed columns
    - Better compression –column values are similar
      - Compression ratio can be very high.
      - Query performance : minutes → seconds

Country
RO
RO
RO
UK

- ORC (Optimized Row Columnar)
  - It is a columnar storage format designed for big data analytics and distributed systems, like Parquet.

Common formats used in industry

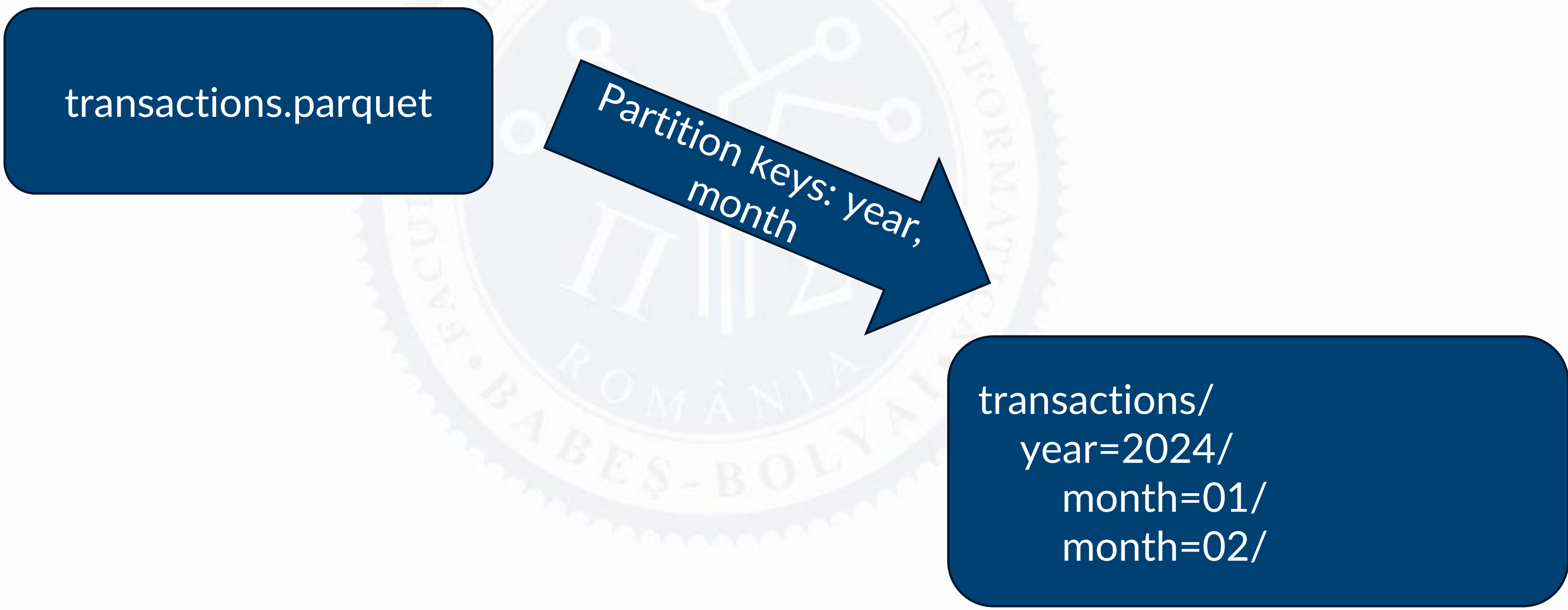
Format	Used in
Parquet	Spark, Big query, Polars
ORC	Hive
Delta	Databricks
Iceberg	Data lakes

Big Data Analysis with Apache Hive

Data partitioning

Data partitioning

- Partitioning divides large datasets into logical subsets.
- Partitioning allows systems to skip irrelevant data.



Data partitioning

```
SELECT * FROM
transactions
WHERE month = 01
```

→

```
transactions/
  year=2024/
    month=01/
    month=02/
```

System reads only January data. Not the entire dataset.

- Performance impact: dataset (1 TB) → monthly partitions (80 GB)
- Query reads 80 GB instead of 1 TB

Partitioning in practice (Spark)

```
df.write.partitionBy("year", "month").parquet("transactions/")
```

```
transactions/
    year=2024/
        month=01/
        month=02/
```

```
df = spark.read.parquet("transactions")

df.filter("month = 1").show()
```

Partitioning in SQL Systems

- CREATE TABLE transactions (id INT, amount DOUBLE) PARTITIONED BY (year INT, month INT)
- INSERT INTO transactions PARTITION (year=2024, month=1)
  SELECT ...
- Bad partitioning can make systems **slower**.
- Good partition keys:
  - date
  - Region
  - Category
- Bad partition keys:
  - user_id
  - transaction_id
- Why?
  - Because they create **too many partitions**.

Reading partitioned data in Polars

```
import polars as pl
..............

df.write_parquet("transactions/", partition_by="year")


df = pl.scan_parquet("transactions/")

df.filter(pl.col("year") == 2024).collect()
```

Distributed Processing

Distributed Processing

- When one machine is insufficient, computation must be distributed across many machines.

- Cluster concept
  - Instead of one machine (laptop), we use a cluster of machines
  - Example: Node1, Node2, Node3, Node4
  - Each node processes data in parallel.

- Example: dataset with 1 billion rows
  - Cluster : 10 machines
  - Each machine processes 100 million rows
  - Parallel computation speeds up analysis.

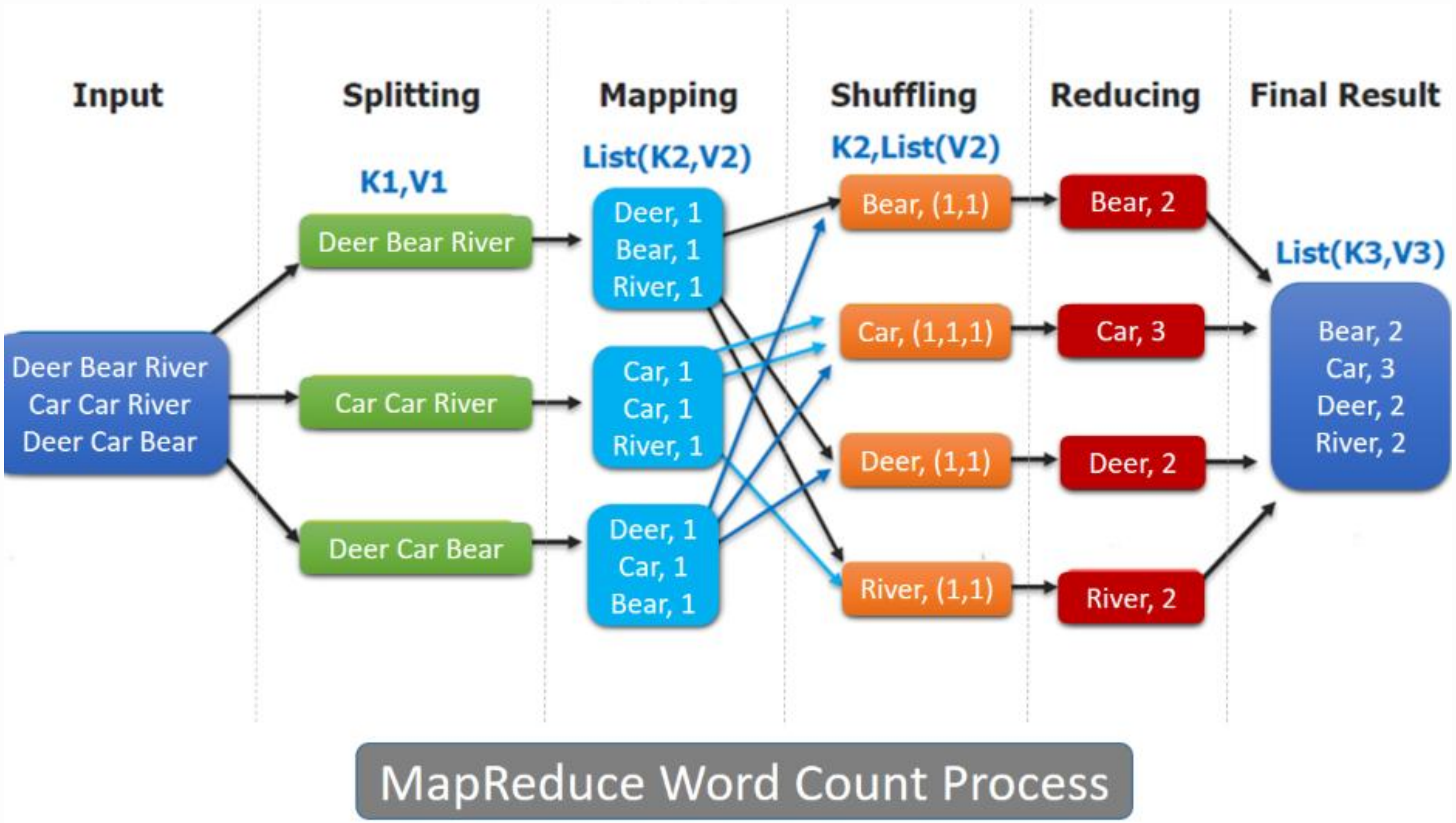
MapReduce Model

- Originally proposed by Google (2004).
- Classic distributed paradigm.
- MAP phase
  - Process partitions independently.
  - Example: Count purchases per country. Each node processes its own data.
    - Node1 → RO=100
    - Node2 → RO=200
    - Node3 → RO=150
- REDUCE phase
  - Combine partial results.
  - Final result: RO=450

MapReduce Model

- This pattern appears in:
  - Hadoop
  - Spark
  - Flink
  - Presto

MapReduce example



Modern Data Architectures

Modern Data Architectures

- When organizations collect massive amounts of data, they face a key question:

Where should the data be stored so that it can be analysed efficiently?

Data Lakes

- A **Data Lake** stores **raw data in its original format.**

- Think of it as a **large storage repository.**

- Features:
  - stores raw data
  - schema applied later
  - supports structured + unstructured data
  - very large scale

- Examples of stored data:
  - Logs
  - JSON files
  - sensor data
  - Images
  - CSV exports
  - API responses

Data Lakes

- Data lake = **a big library without cataloguing**

- All books exist, but finding information is difficult.

- Data is usually stored as Parquet files, ORC or JSON .

- Technologies used for Data lakes:
  - Amazon S3
  - Azure Data Lake
  - Google Cloud Storage
  - HDFS (Hadoop Distributed File System)

Data Warehouse

- A Data Warehouse stores cleaned and structured data optimized for analytics.

- Unlike data lakes, warehouses are designed for fast queries.

- Features:
  - structured tables
  - optimized for SQL queries
  - cleaned and transformed data
  - designed for analytics

Data Warehouse

- Data warehouse = **organized supermarket**

- **Everything is:**
  - Categorized
  - Clean
  - easy to find

- Technologies used for Data Warehouse:
  - Snowflake
  - BigQuery
  - Redshift
  - ClickHouse
  - Azure Synapse

- These systems allow **fast SQL queries on massive datasets.**

Data Lake vs Data Warehouse

Feature	Data Lake	Data Warehouse
Data type	Raw data	Structured data
Schema	Schema-on-read	Schema-on-write
Users	Data engineers / scientists	Analysts
Purpose	Storage	Analytics

Teamwork time

Teamwork time

- Scenario: as a data scientist you receive the following dataset
  - 2 TB
  - rows: 5 billion
  - columns: 30
- Goal: compute the average revenue per country per month
- Laptop: 16 GB RAM
- You need to propose the followings:
  - Storage format
  - Processing system
  - Architecture
  - Do you need partitioning? If yes, how do you proceed?

Case study: Uber

Case study: Uber

- Uber collects massive event streams.

- Data generated by:
  - drivers
  - riders
  - GPS
  - Payments

- Example of features:
  - driver_id
  - trip_start
  - trip_end
  - Location
  - Price
  - Timestamp

- Scale: billions of events daily

Case study: Uber

- **Problem:** Demand fluctuates rapidly.
- Example: concert ends → 10,000 ride requests appear instantly
- System must:
  - detect demand spike
  - update prices
  - notify drivers

Uber pipeline

Case study: Netflix

Case study: Netflix

- Netflix tracks user behaviour:
    - movie id
    - watch duration
    - pause events
    - Device
    - Time
    - Location

- Scale: **petabytes**

Case study: Netflix recommendation pipeline

Case study: Netflix

- Feature: avg_watch_time_per_genre

- Needed data:
  - User123
  - comedy = 120 min
  - drama = 15 min

- Used for ranking recommendations.

- Real Challenge — Data Drift
  - user behavior changes.
  - new movie genres
  - new devices

- Models must be retrained regularly.

Key takeaways

Key takeaways

- Large data rarely fits in memory
- Storage format determines performance
- Distributed systems enable large-scale analytics
- Partitioning is critical for query efficiency
- Modern AI systems are data systems first.

Thank you for your attention — questions, thoughts, or challenges?

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BABEȘ-BOLYAI UNIVERSITY

1 Mihail Kogălniceanu Street,
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro