Big Data Analytics

9: Efficient In-Memory Analytics with Polars

Instructor: Oleh Tymchuk

#11: Agenda

- Introduction to Polars
- Polars vs. Other Tools
- Use cases
- Practical cases
- Useful Links

Introduction to Polars

What is Polars?

Definition:

- High-performance DataFrame library for Python and Rust
- Built on Apache Arrow for memory efficiency
- Written in Rust, supports parallel execution

Key Comparisons:

- Faster than Pandas for large datasets
- Similar speed to Spark/Dask, but simpler—no cluster setup

Why Use Polars?

Speed

5x–50x faster than Pandas in many workloads

Memory Efficiency

Uses Apache Arrow's efficient columnar memory format

Lazy Execution

Defers computation for optimization

Clean API

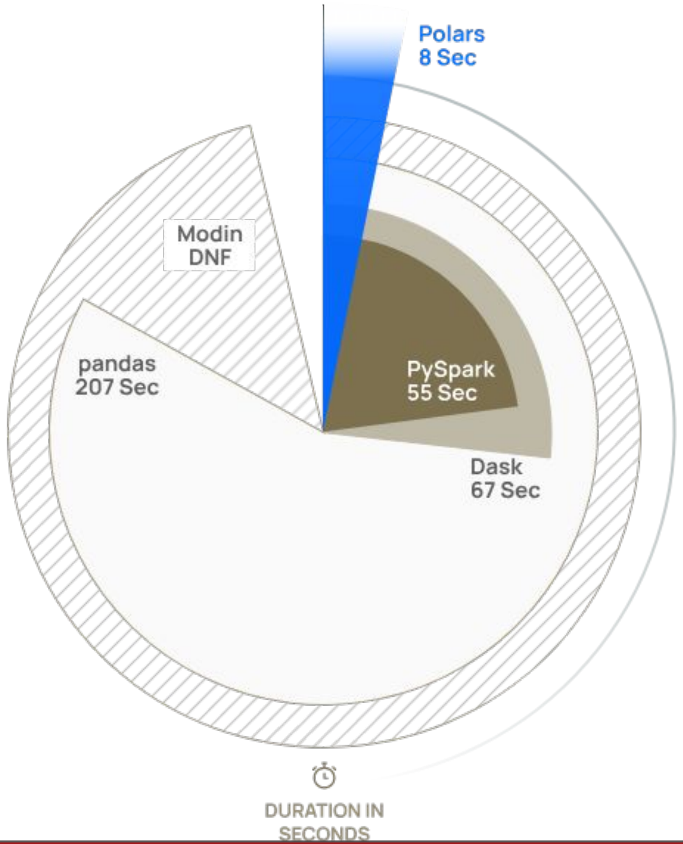
Chainable methods for readable, maintainable code

Key Features of Polars

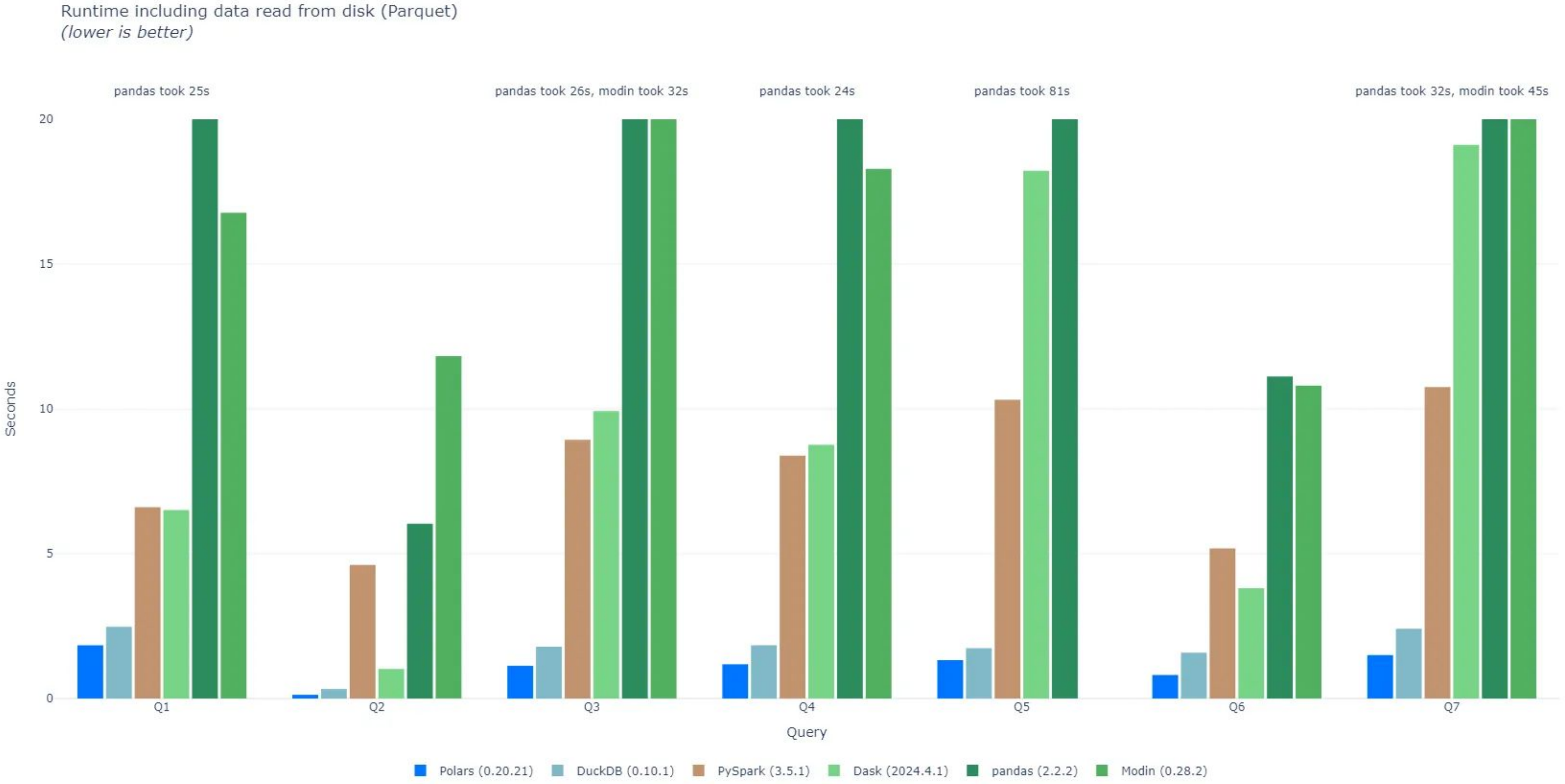
Feature	Description
Lazy Evaluation	Optimize complex queries automatically
Arrow Integration	Zero-copy interoperability with Arrow
Parallel Execution	Utilizes all available CPU cores
Rich Data Types	Supports nested data, nulls
Interoperability	Reads/Writes CSV, Parquet, Pandas, NumPy

Polars vs. Other Tools

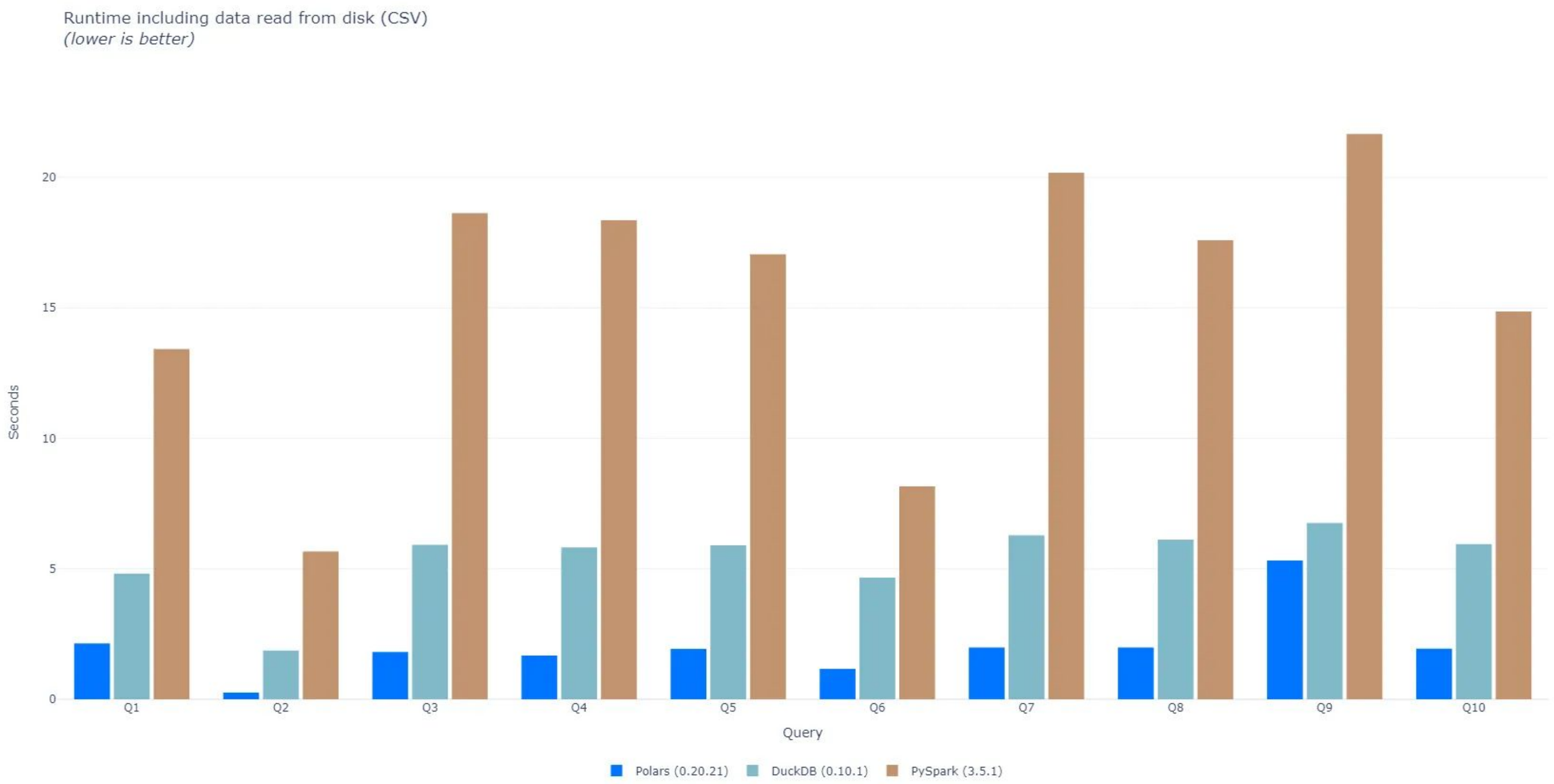
Duration in Seconds



Data read from disk (Parquet)



Data read from disk (CSV)



Use Cases

Use Cases

- Large datasets: Process GBs of data on a single machine.
- ETL pipelines: Fast transformations and aggregations.
- Data analysis: Interactive workflows with lazy evaluation.
- Arrow ecosystems: Integrate with tools like DuckDB, PySpark.

Limitations

- Smaller community vs. Pandas
- Fewer third-party integrations
- Learning curve for expression-based syntax

Practical cases

Useful Links

[Python API reference]

[Polars Academy]

Q&A