DuckDB:

An embedded database for data science

Pedro Holanda
pedro@duckdblabs.com

CWI

About Us

- CWI in Amsterdam;
  - Database Architectures Group
- Why a new system?
  - Most Projects at CWI are sponsored by private companies;
  - Most of our projects are related to **data** science (Particularly, ML pipelines);
  - Main goal is to optimize/facilitate the management of data in data science projects.

Outline

Outline (Today)

- Why should I use a Database System?
- Combining Database Systems with Data Science.
- DuckDB: An embedded database system for data science.
- **Hands-on ~ 45 min. [Using DuckDB]**

Why should I use a database system?

Database Example

- Database that models a digital music store to keep track of artists and albums.

- Things we need to store:
  - Information about <u>artists</u>.
  - What <u>albums</u> those artists released.

This example is originally from:
https://15445.courses.cs.cmu.edu/fall2019/slides/01-introduction.pdf

Flat File

- Store database as comma-separated value (CSV) files that we manage in our own code
  - Use separate file per entity
  - The application has to parse files each time they want to read/update records

Flat File Example

- Database that models a digital music store

Artist (name, year, country)

“Backstreet Boys”,1994,"USA"
“Ice Cube”, 1992,"USA
“Notorious BIG”,1989,USA

Album (name,artist,year)

“Millenium", "Backstreet Boys", 1999
“DNA”, “Backstreet Boys”, 2019
“AmeriKKKa's Most Wanted”, “Ice Cube”, 1990

Flat File Example

- Get the year that Ice Cube went solo

Artist (name, year, country)

“Backstreet Boys”,1994,"USA"

“Ice Cube”, 1992,"USA

“Notorious BIG”,1989,USA

```
for line in file
    record = parse(line)
    if 'Ice Cube' == record[0]
        print int(record[1])
```

Data Integrity

- How do we ensure that the artist is the same for each album entry?
- What if someone overwrites the album year with an invalid string?
- How do we store that there are multiple artists on an album?

Implementation

- How do we find a particular record?
- What if we now want to create a new application that uses the same database?
- What if two threads try to write to the same file at the same time?

CWI

Database Management System

- Software that allows application to store and analyse information in a database.

- A general-purpose DBMS is designed to allow the definition, creation, querying, update and administration of databases.

Motivation

- Many data scientists do not use database systems
  - Despite requiring many of the things they offer!
  - Data management, data wrangling...
- Instead: Engineer their own solutions
  - **Flat files** to store data (CSV, Binary, HDF5, etc).
  - **dplyr/pandas** as query execution engines.

CWI

What is the problem?

- Manually managing files is cumbersome.
- Loading and parsing e.g. CSV files is inefficient.
- File writers typically do not offer resiliency.
  - Files can be corrupted;
  - Difficult to change/update.
- It does not scale!
- Why people use it?
  - A pip install and you ready to go.

```
$ pip install pandas, numpy
```

Dplyr; Pandas; DataFrames - What is the problem?

- The problem is that they are *very poor query engines*!
- Materialize huge intermediates
- **No** query optimizer
  - Not even for basics like filter pushdown
- No support for out of memory computation
- No support for parallelization
- Unoptimized implementations for joins/aggregations

Data Science

- Data scientists **need** the functionality database systems offer
- But they opt not to use database systems
- Often this leads to problems down the road
  - When the data gets bigger..
  - When a power outage corrupts their data...

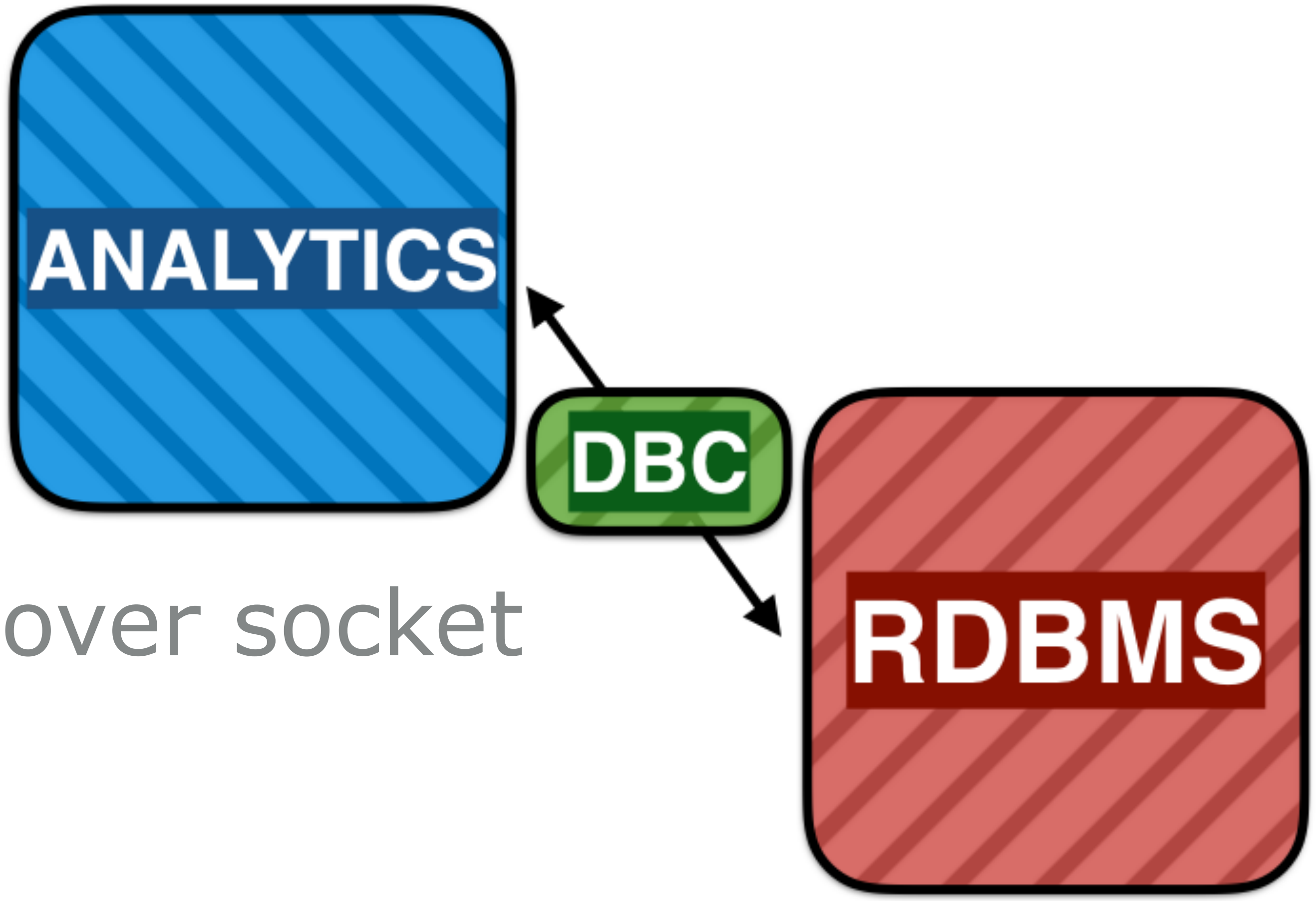
Combining Database Systems with Data Science

- How can we combine analytical tools (R/Python) with database systems?

How?

- Database Client Connections
- User Defined Functions
- Embedded Database Systems

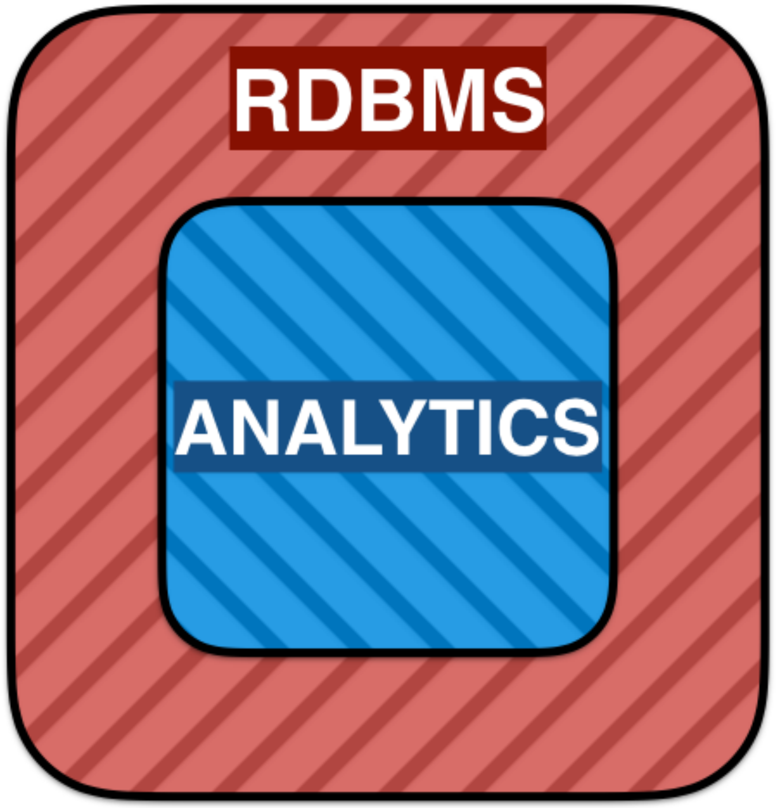
Database Client Connections



- **1: DB Connection**
- DBMS is separate process
- Queries & Data transferred over socket
- Problems:
  - Data transfer is very slow (both directions)
  - Requires setup & management of DBMS server

2. User Defined Functions

- Most cases a data scientist exports data from the RDBMS to the analytical tool
- **User-Defined Functions (UDFs)**
- Analytics is run inside DBMS server
- No separate analytics program!
- Problems:
  - Difficult to implement and debug
  - DBMS-specific, requires knowledge of DBMS internals
  - Also requires setup & management of DBMS server



CWI

3. Embedded Databases

- It runs inside the analytics applications;
- Has same low-transfer cost advantages as UDFs;
- Are easy to install/use;
- Binds to almost every language.
- What the most famous embeddable DBMS?
  - Runs on every phone, browser and OS
  - It even runs inside airplanes!
  - **Great for transactions, not so good for analytics.**

SQLite

CWI

DuckDB

CWI

DuckDB’s Core Features

- DuckDB: The SQLite for Analytics

- **Simple installation**

```
$ pip install duckdb
```

- **Embedded: no server management**
- **Fast analytical processing**
- **Fast transfer between R/Python and RDBMS**

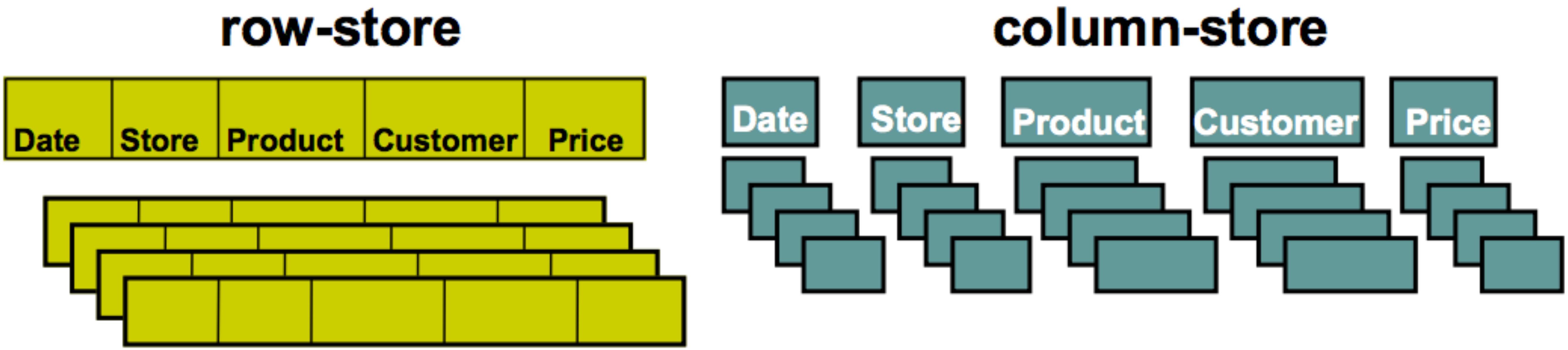
- **Duckdb is currently in pre-release**
  - **Check duckdb.org for more details.**

What makes duckdb special?

- Data Science is equal to Analytical Processing!
- Storage Model
  - Row-Store vs Column-Store
- Compression
- Query Execution
  - Row-wise vs vector-wise
- Relational API

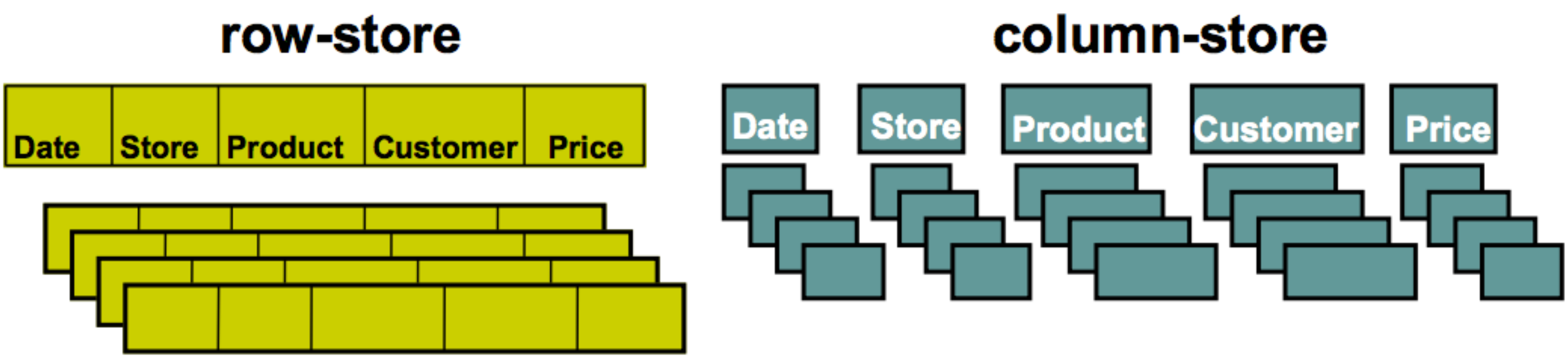
Storage Model

- SQLite use a row-storage model
- DuckDB uses a columnar storage model



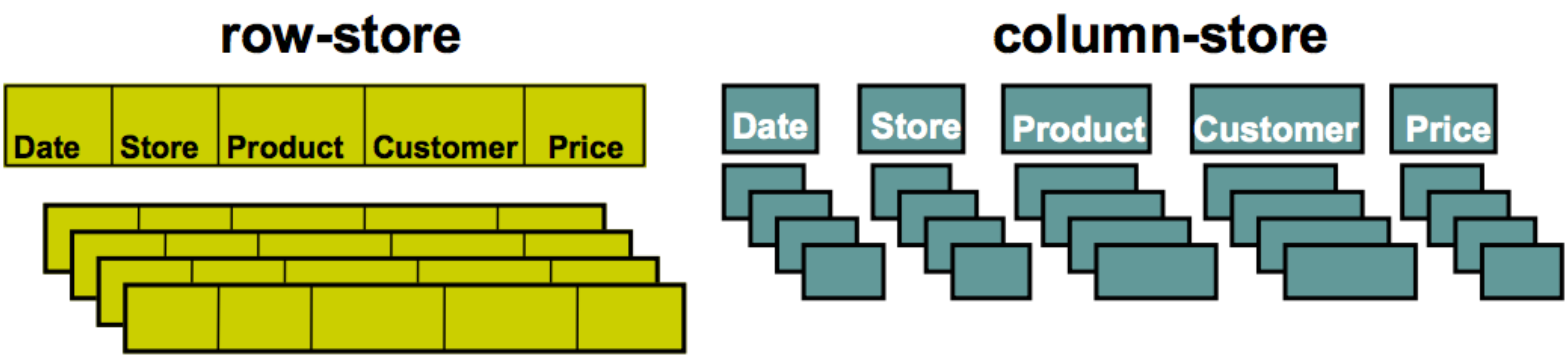
Storage Model

- **Row-Storage**:
  - Individual rows can be fetched cheaply
  - However, **all columns must always be fetched!**
- What if we only use a few columns?
- **e.g.**: What if we are only interested in the price of a product, not the stores in which it is sold?



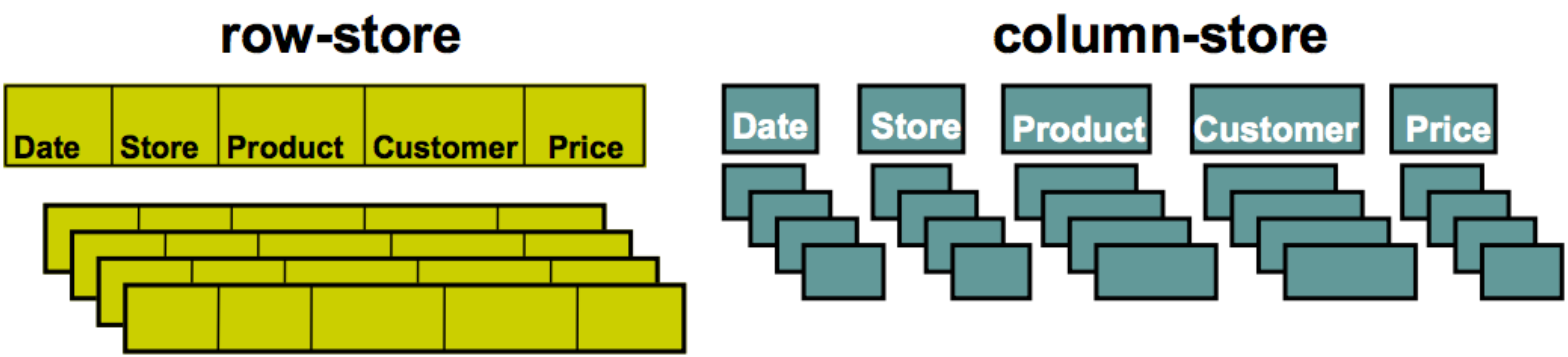
Storage Model

- **Column-Storage**:
  - We can fetch individual columns
  - Immense savings on disk IO/memory bandwidth when only using few columns



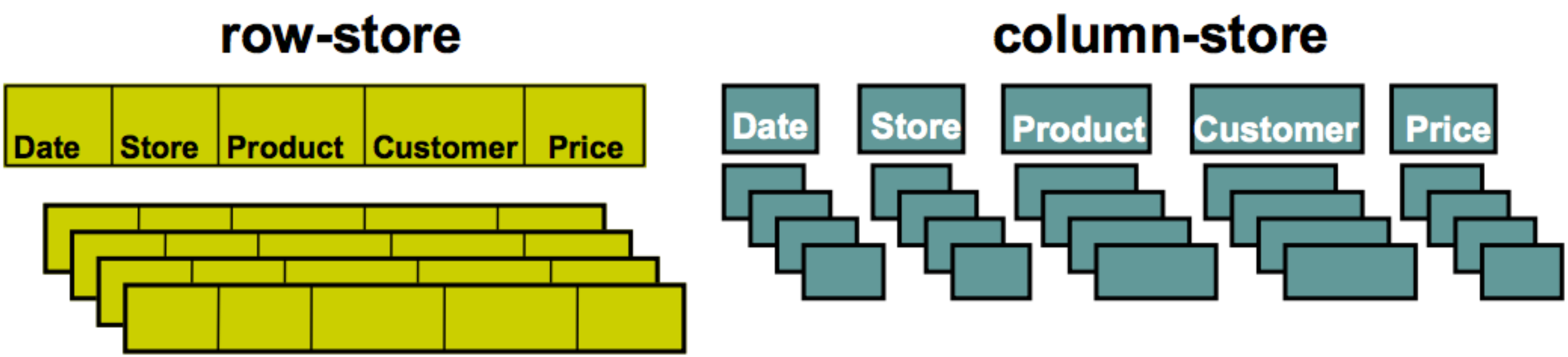
Storage Model

- **Example:** Suppose we have a 1TB table with 100 columns. We have a query that requires 5 columns of the table.
  - **Row-store:** Read entire 1TB of data from disk at 100MB/s ≅ 3 hours
  - **Column-store:** Read 5 columns (50GB) from disk ≅ 8 minutes



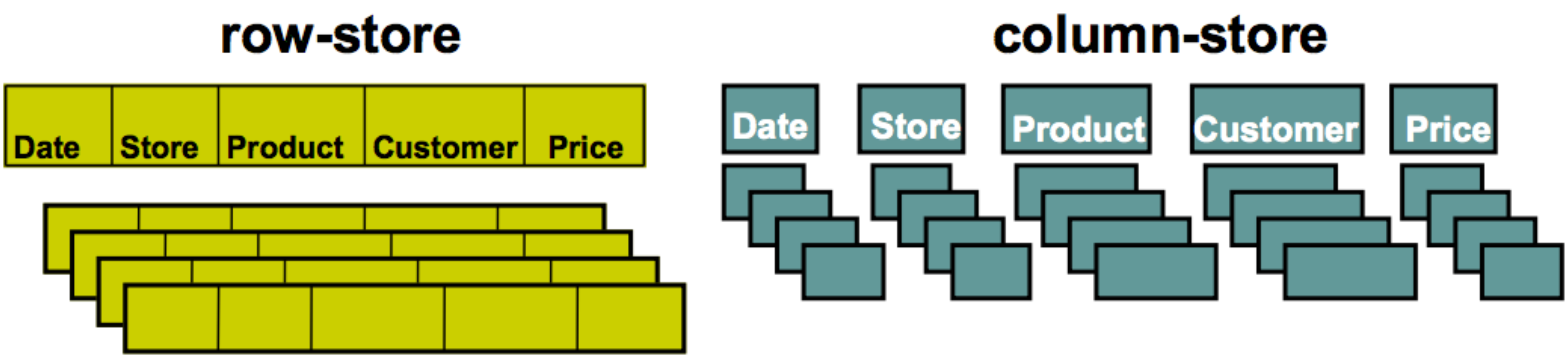
Compression

- **Compressibility** is another advantage of column-storage
- Individual columns often have similar values, e.g. dates are usually increasing
- Save ~**2-10X** on storage (depending on compression algorithms used and data)



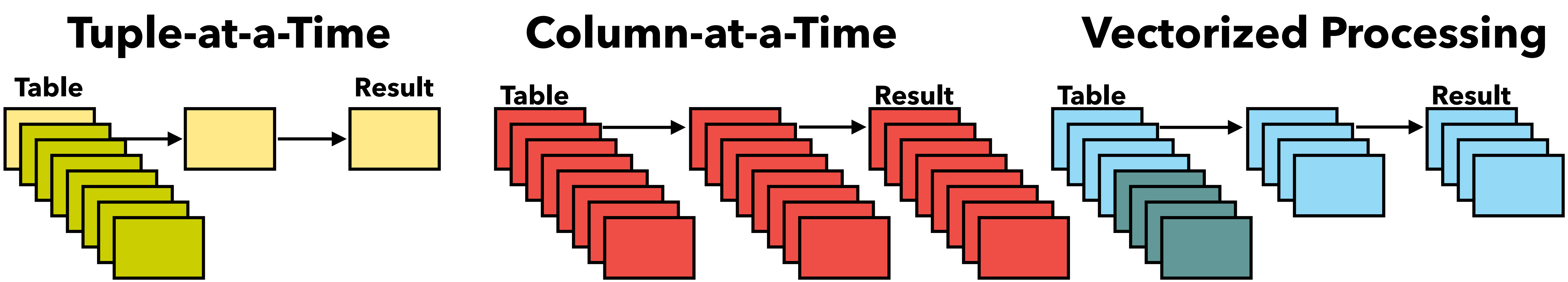
Compression

- **Example:** Suppose we have a 1TB table with 100 columns. We have a query that requires 5 columns of the table.
  - **No compression:** Read 5 columns (50GB) from disk ≅ 8 minutes
  - **Compression:** Read 5 compressed columns (5GB) from disk ≅ 50 seconds



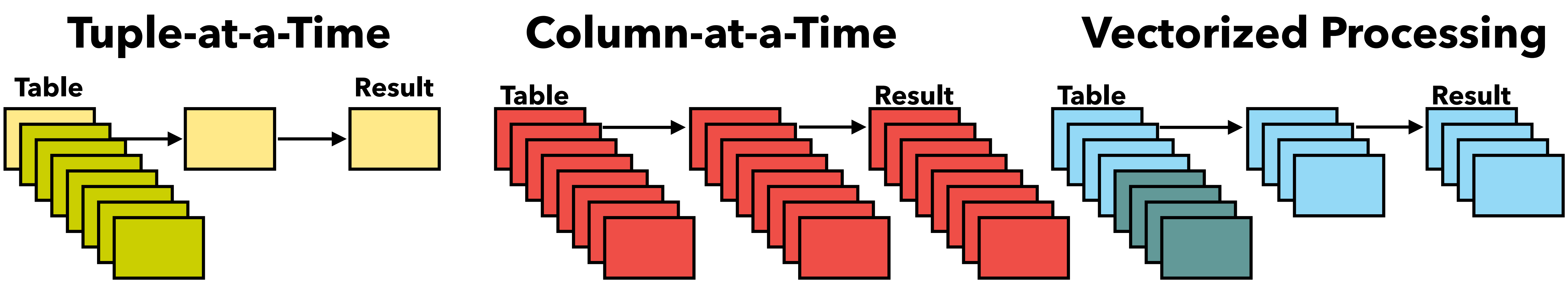
Query Execution

- **SQLite** use tuple-at-a-time processing
  - Process **one row** at a time
- **NumPy/R** use column-at-a-time processing
  - Process entire columns at once
- **DuckDB** uses **vectorized** processing
  - Process **batches** of columns at a time



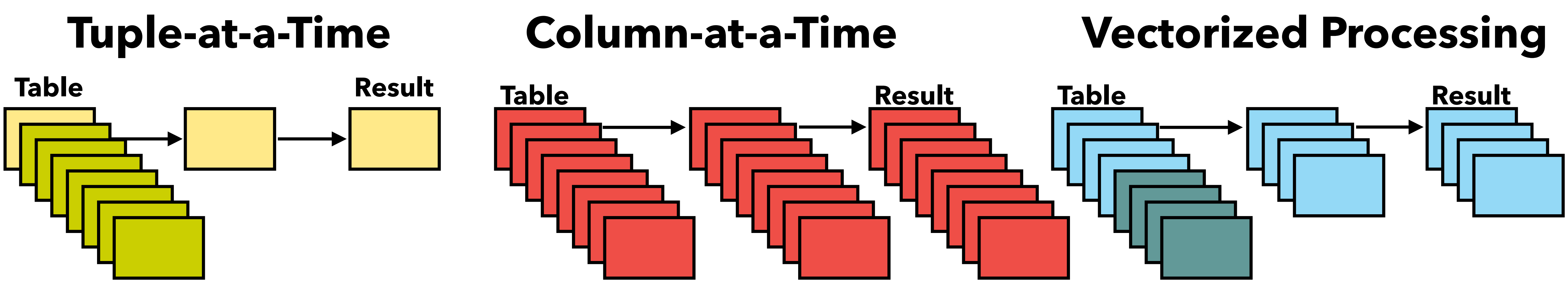
Query Execution

- **Tuple-at-a-Time (SQLite)**
  - Optimize for low memory footprint
  - Only need to keep **single row** in memory
- Comes from a time when **memory was expensive**
- **High CPU overhead per tuple!**



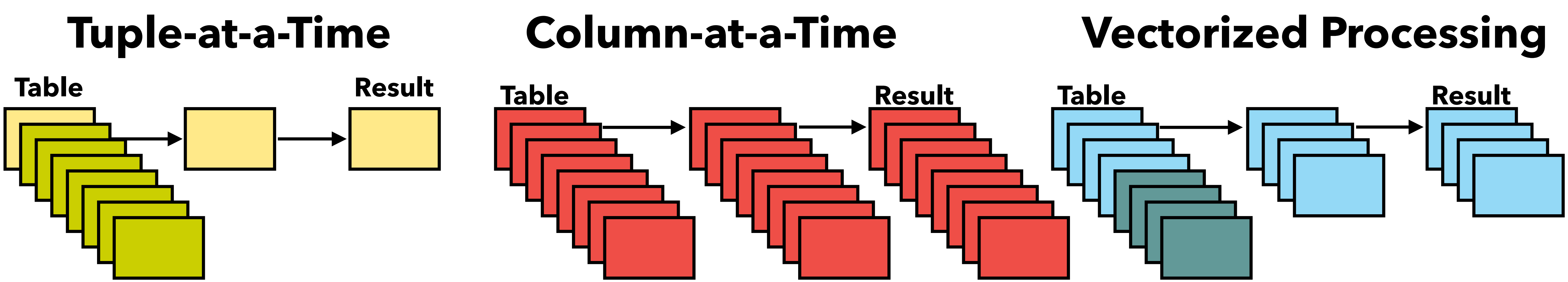
Query Execution

- **Column-at-a-Time (NumPy/R)**
  - Better CPU utilization, allows for SIMD
  - Materialize **large intermediates** in memory!
- Intermediates can be gigabytes each…
- **Problematic** when data sizes are large



Query Execution

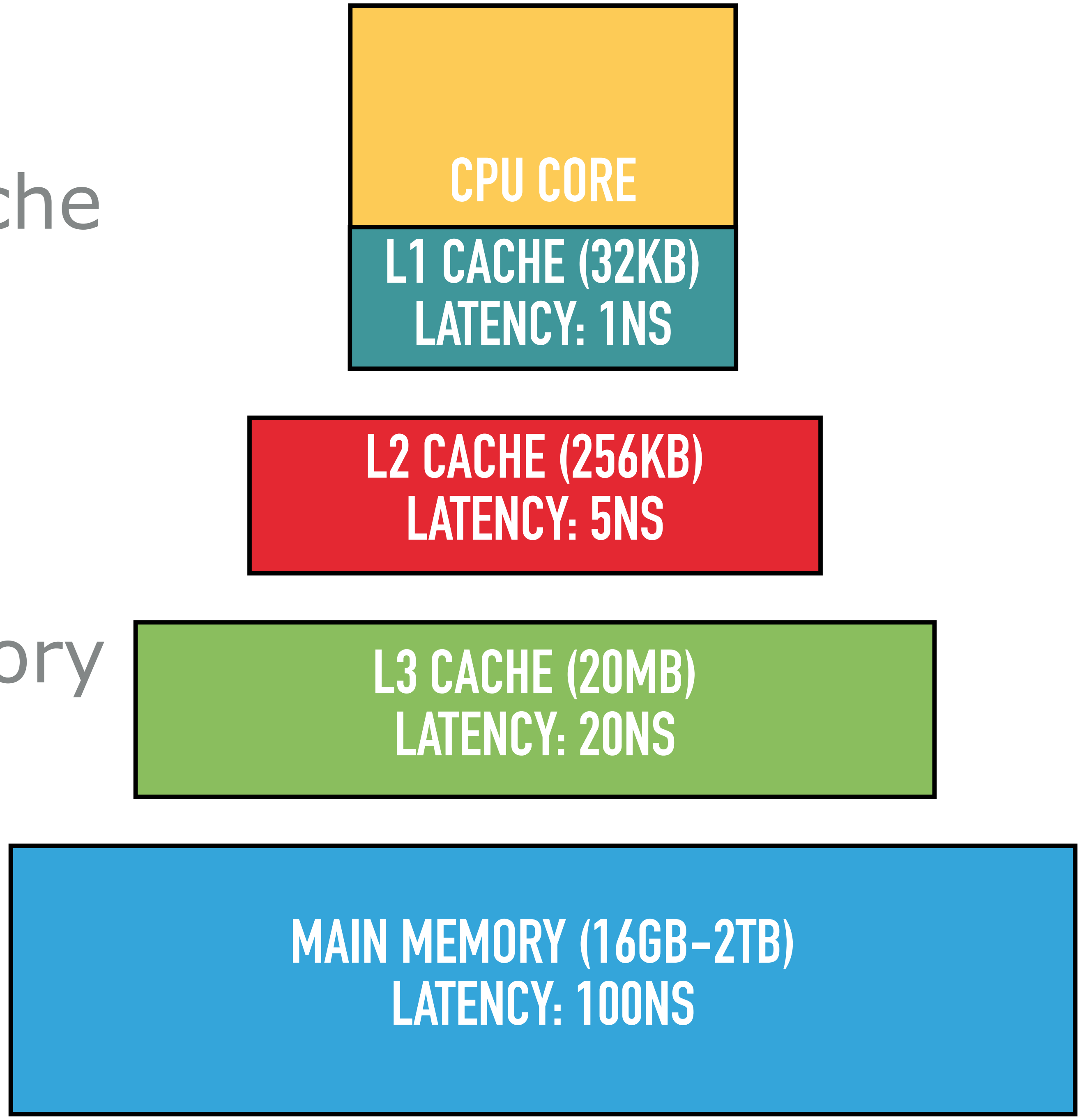
- **Vectorized Processing (DuckDB)**
  - Optimized for CPU Cache locality
  - SIMD instructions, Pipelining
  - **Small** intermediates **(ideally fit in L1 cache)**



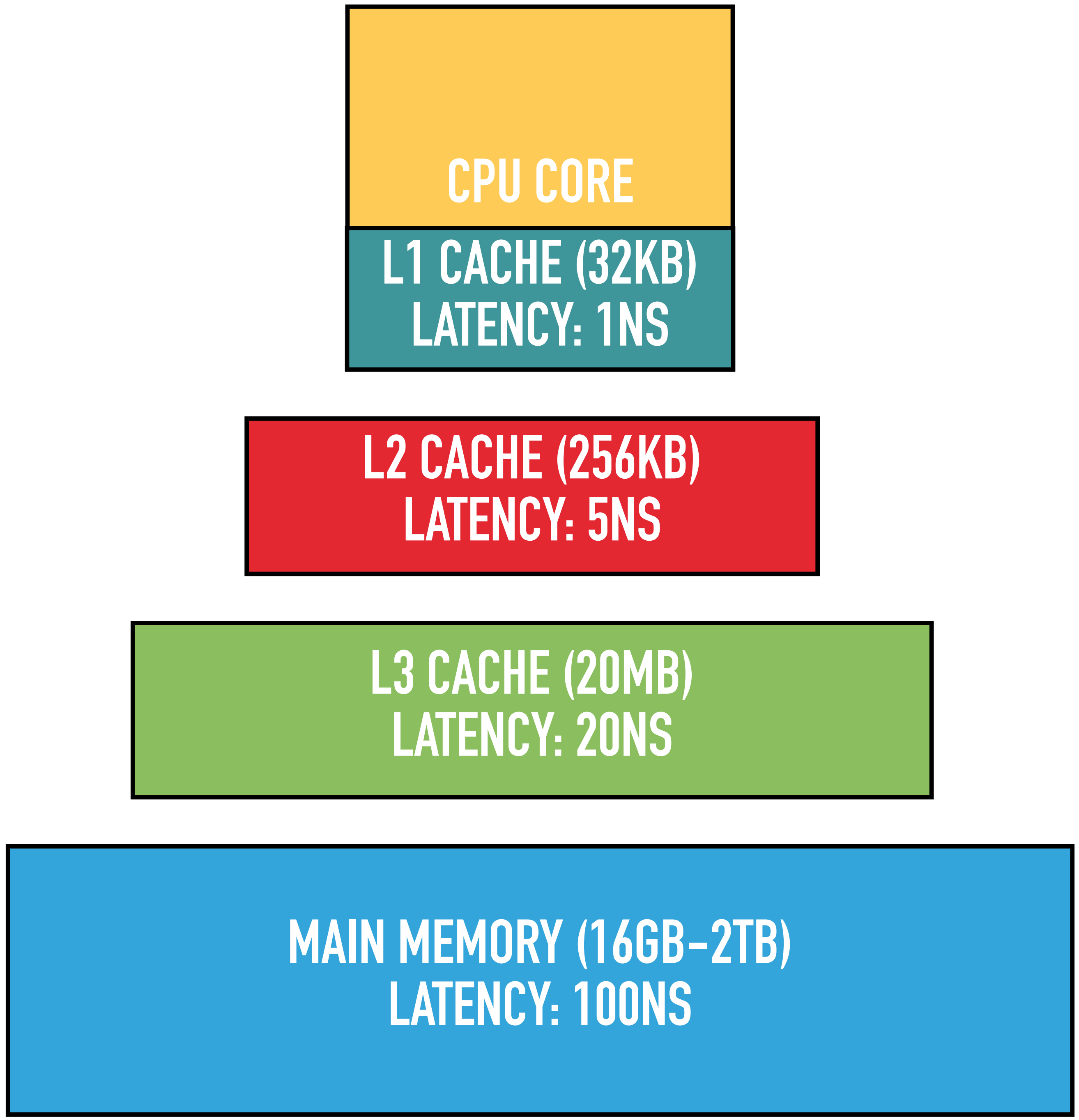
Query Execution

- **Vectorized Processing**
- Intermediates fit in L3 cache

- **Column-at-a-Time**
- Intermediates go to memory



Query Execution



CWI

Relational API

- One of the reasons data scientists love libraries like pandas is their easy API.
- Database Systems forces developers to use SQL
- In DuckDB we have been developing a new API, called relational API that resembles the API of libraries like Pandas.

DB API

```
import duckdb
con = duckdb.connect('database.db')
cursor = db.cursor()
cursor.execute('Select j+1 from integers where i = 2')
```

RELATIONAL API

```
import duckdb
con = duckdb.connect('database.db')

# table operator returns a table scan
tbl = duckdb.table('integers')
# we can inspect intermediates
tbl.show()
# we can chain multiple operators
tbl.filter('i=2').project('j+1').show()
```

Hands-on

Hands-on

- Goal: See in practice the differences of:
  - Pandas;
  - DuckDB.
- Tasks (Benchmark):
  - Queries (e.g., aggregations, filters and joins);
  - Transactions.

Hands-on

- Right Now:
  - Clone Repo: https://github.com/pdet/duckdb-tutorial
  - Upload file: Part 1/Exercise/Exercise.ipynb to https://colab.research.google.com/ as a Python 3 Notebook.