Bespoke-OLAP

or

Do we still need (general-purpose) Databases in an AI World?

The Bespoke Team:


Carsten Binnig


Johannes Wehrstein


Timo Eckmann


Matthias Jasny

SYSTEMS

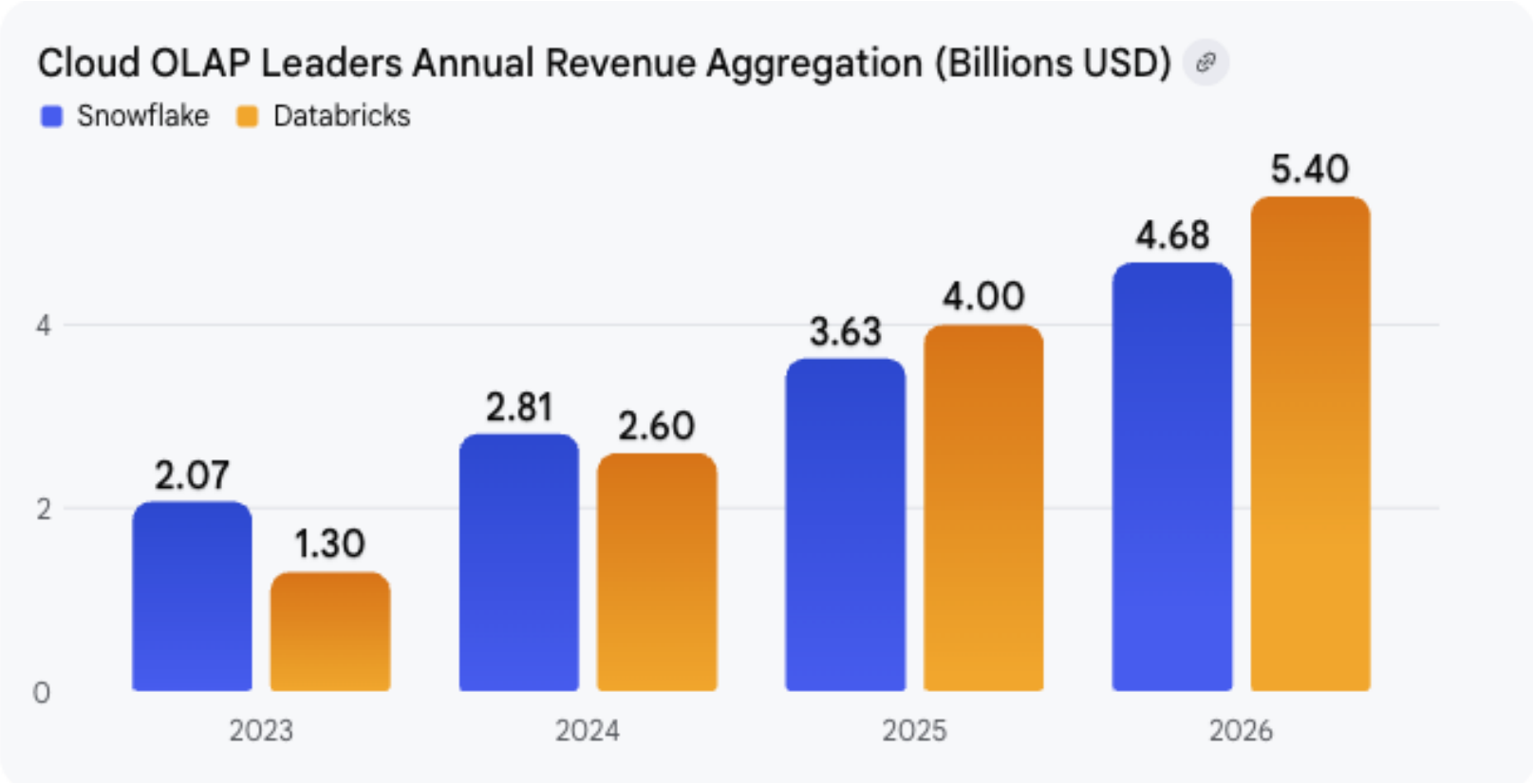
TECHNISCHE UNIVERSITÄT DARMSTADT

dfki ai

hessian.AI

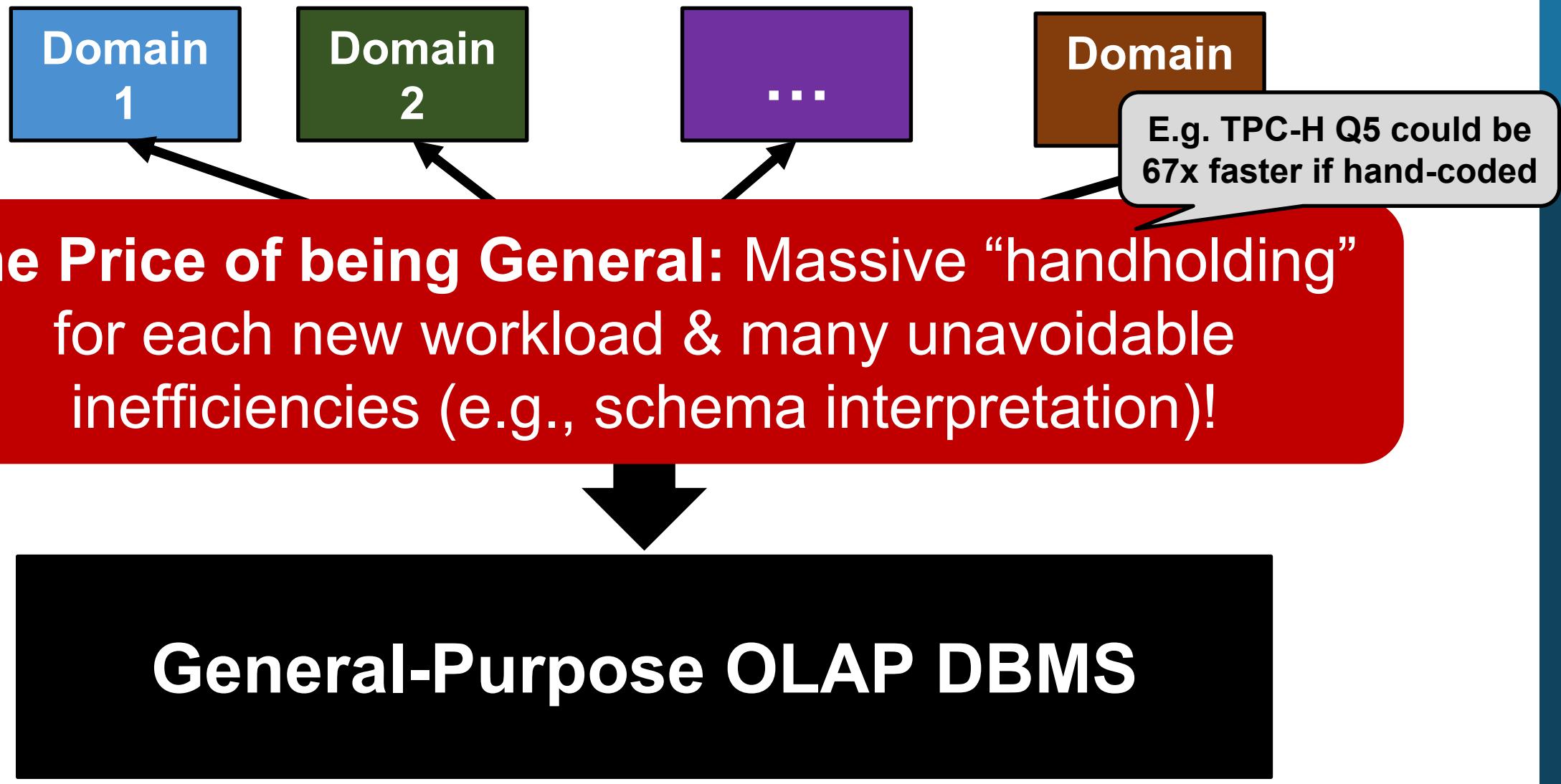
RAI REASONABLE AI

Success of OLAP Comes From General-Purpose

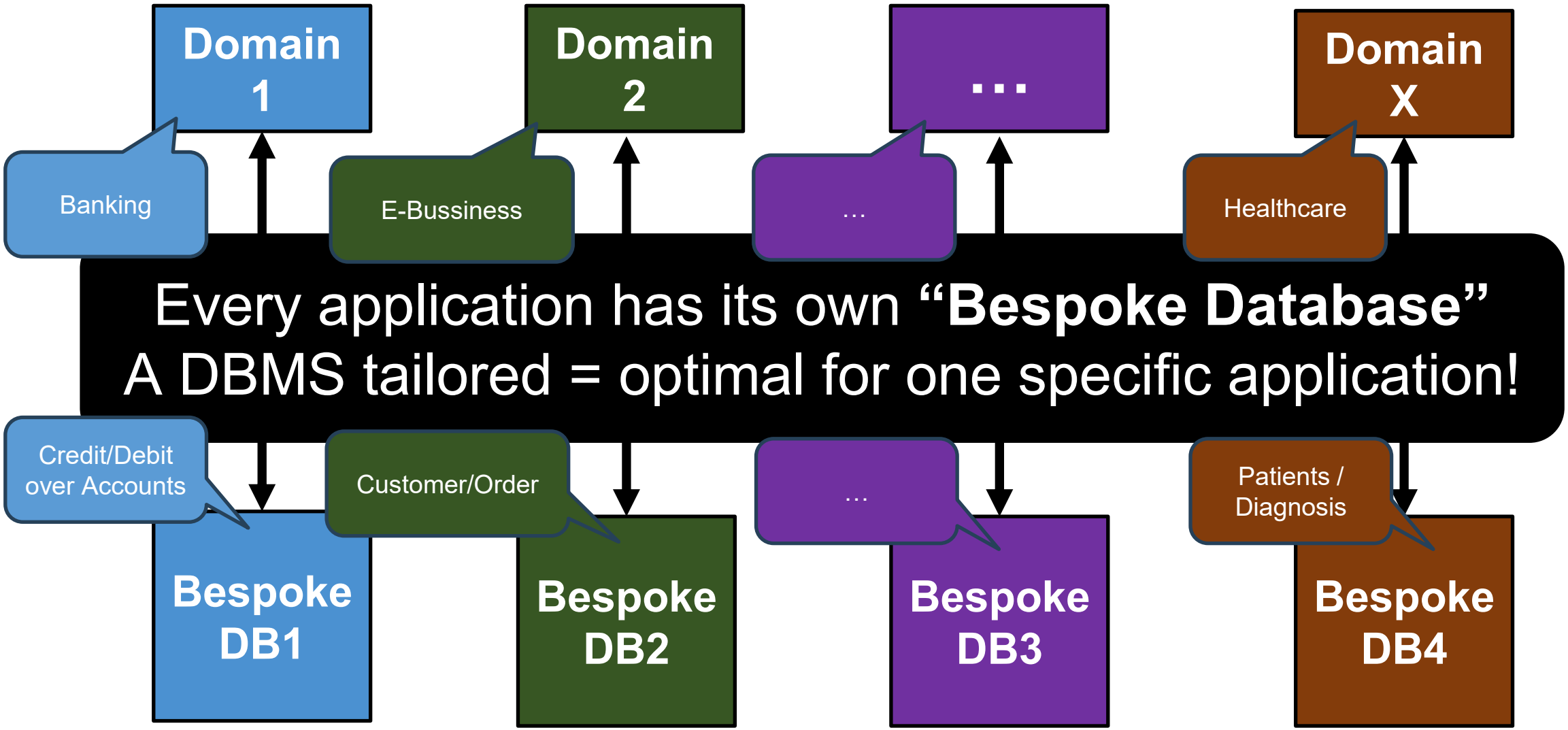


Almost every domain *from banking, government, healthcare, supply chains, e-commerce, …* ***uses OLAP***

BUT: There is a High Price of Generality!



Our Vision: Bespoke Databases!



How to Build a Bespoke DBMS for every Workload?

Bespoke is a Luxury! (Today)

Only Credit/Debit over Accounts

TigerBeetle

Bespoke DB for Finance

TigerBeetle is targeting a Big Market $$$

Clone Thomas Neumann?

BUT Bespoke for All? Too high Costs & ~~Missing Experts!~~

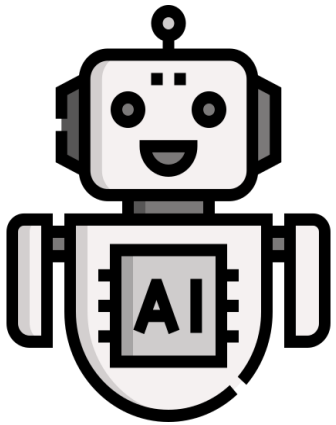
We can Synthesize Bespoke DBMSs!

Schema + Data

Workload (Query Templates)

Hints (optional)

Bespoke Synthesizer (DBMS Coding Agent)



Performance & Correctness

Reference System

e.g. DuckDB

DBMS will be fixed for this schema & query-templates

Non-trivial to synthesize a Full DBMS

Full DBMS in C++ optimal for workload

Magic "Sauce": Complex Synthesis Pipeline = System for System Synthesis

Opportunities of Bespoke DBMSs

General Purpose Storage:
Tables + Indexes (Any Data) -> Overhead: Schema Interpretation & Non-optimal Physical

Bespoke Storage:
`struct Customer {std::string name; ..}` +
Keep data in App-optimal Data Structures

General Purpose Execution:
Relational Engines (Any SQL) -> Overhead: Query Parsing, Optimization, …

Bespoke Execution:
Non-relational Execution in C++ (e.g., Think of late Materialization), No flat query results

General Purpose Optimization:
Runtime cost-based planning for unknown queries

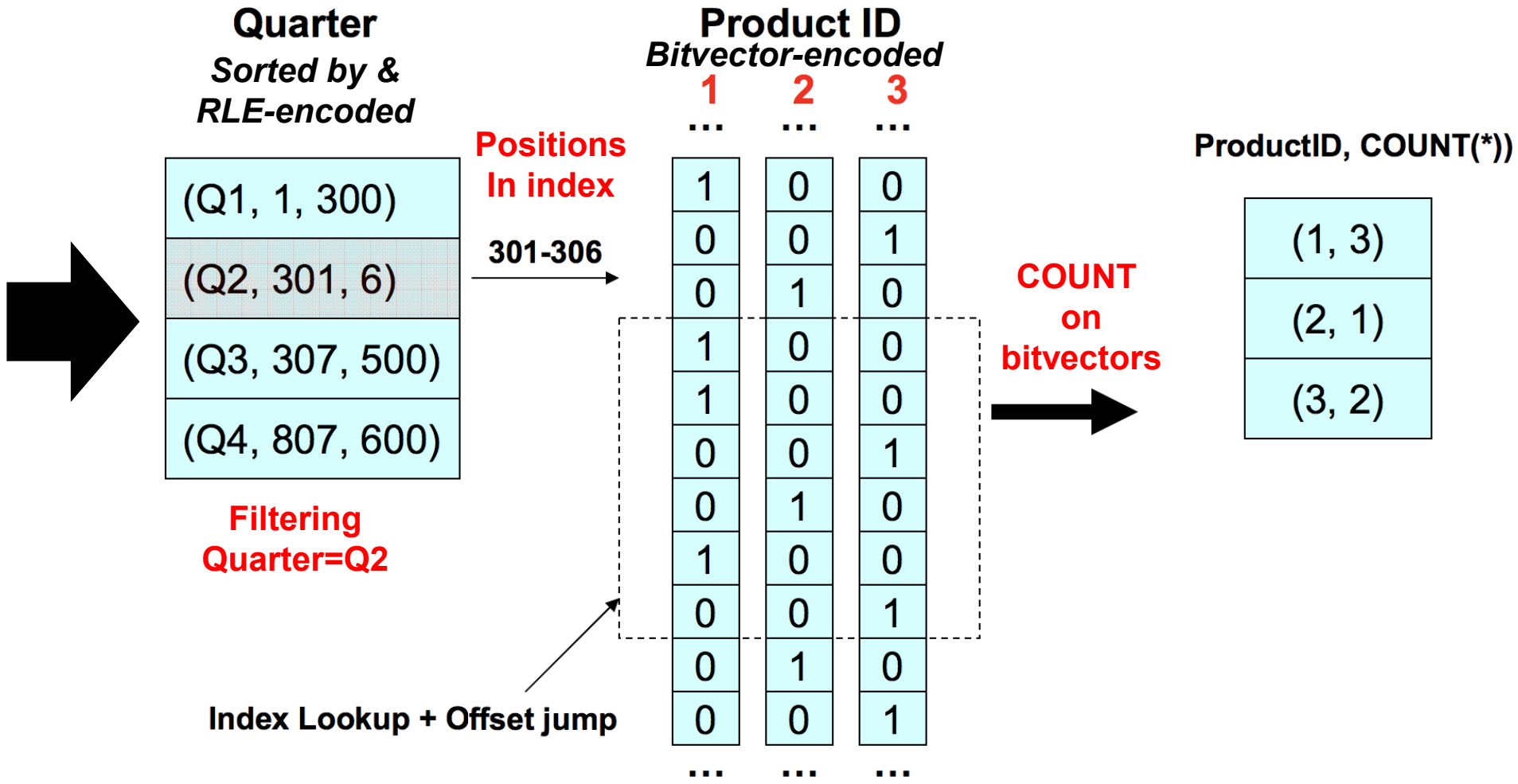
Bespoke Optimization:
Offline empirical synthesis for known queries, data and hardware or bespoke cost/cardinality estimators

see our AIDB@VLDB'26 paper Bespoke-Card

...

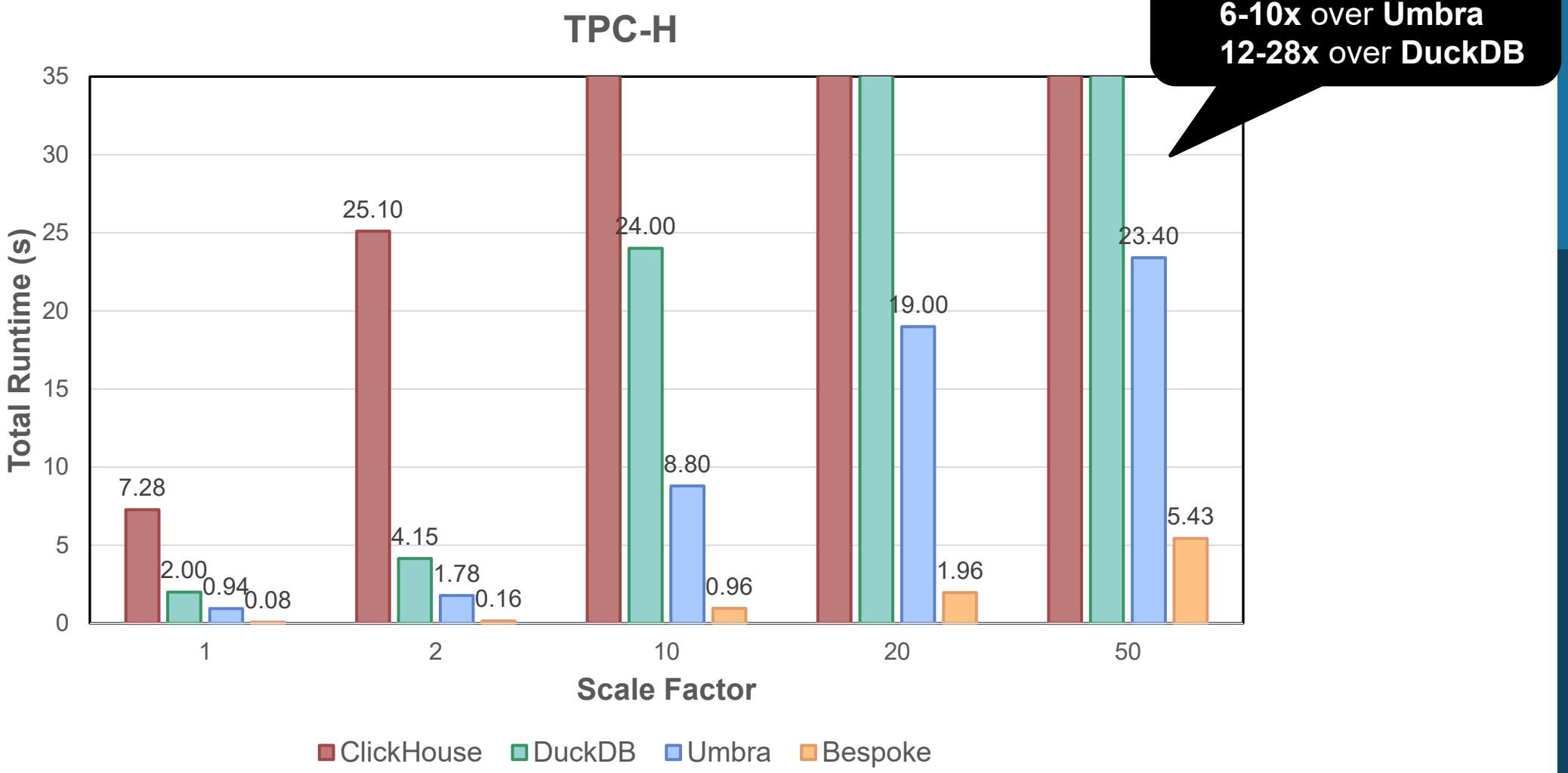
Example of Bespoke Storage + Execution

```
SELECT
   ProductID,
   COUNT(*)
FROM table
WHERE
   Quarter=Q2
GROUP BY
   ProductID
```



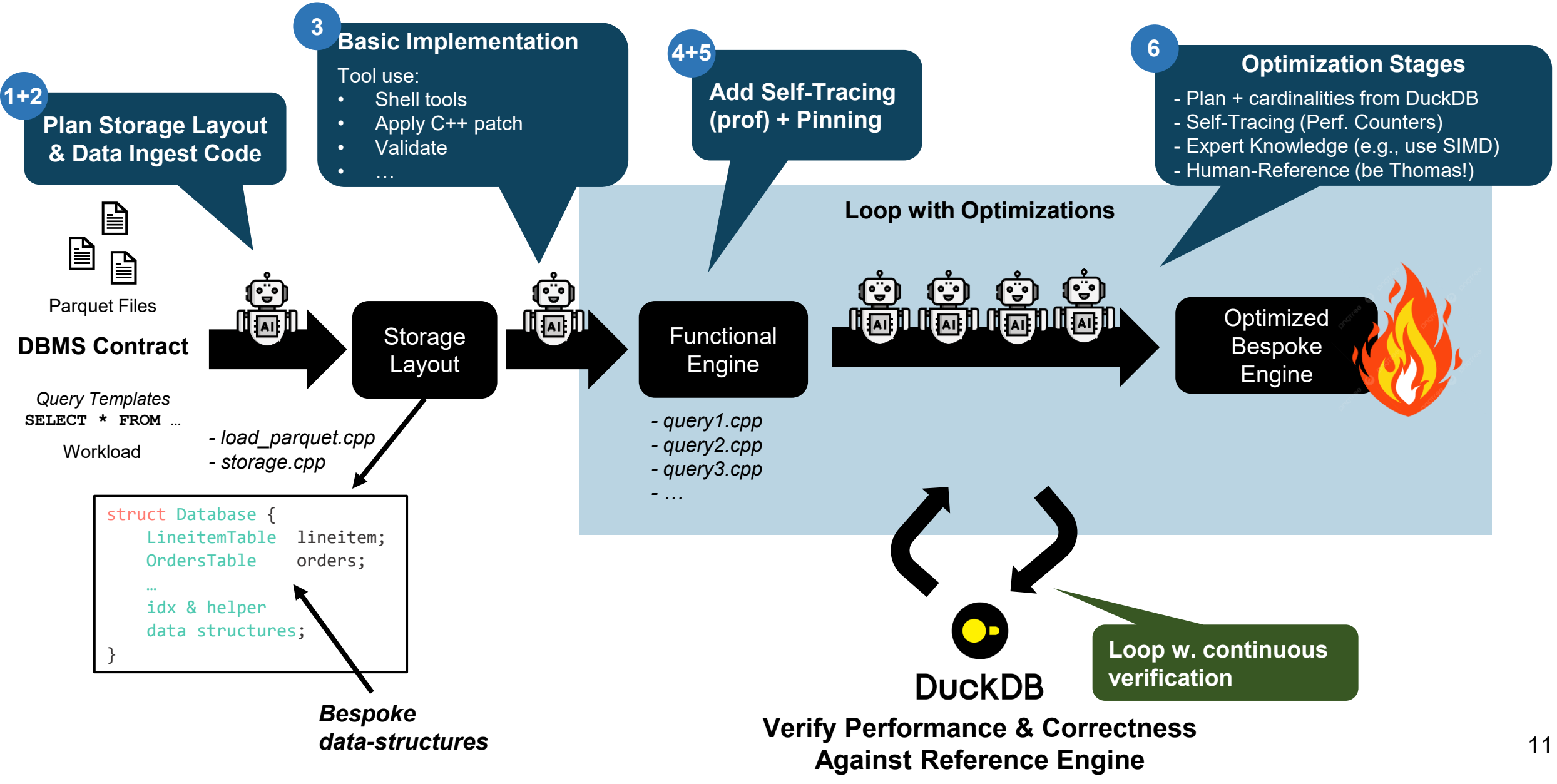
*in-memory, 1 thread

Bespoke DBMS: Does it work?



Order of Magnitude Speedups over Umbra & DuckDB

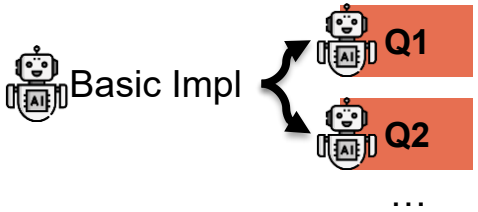
Our System for System Synthesis: The Pipeline



A System for System Synthesis: Four Main Ingredients

① Conversation Branching

Parallel per-query optimization



② Live Hotpatching

Seconds per edit, not minutes

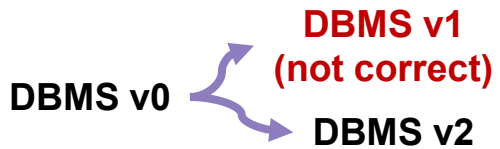
Avoids restart & data reingestion

③ Continuous Testing

Correctness and speedups at every step

④ Regression Monitor

Always recoverable

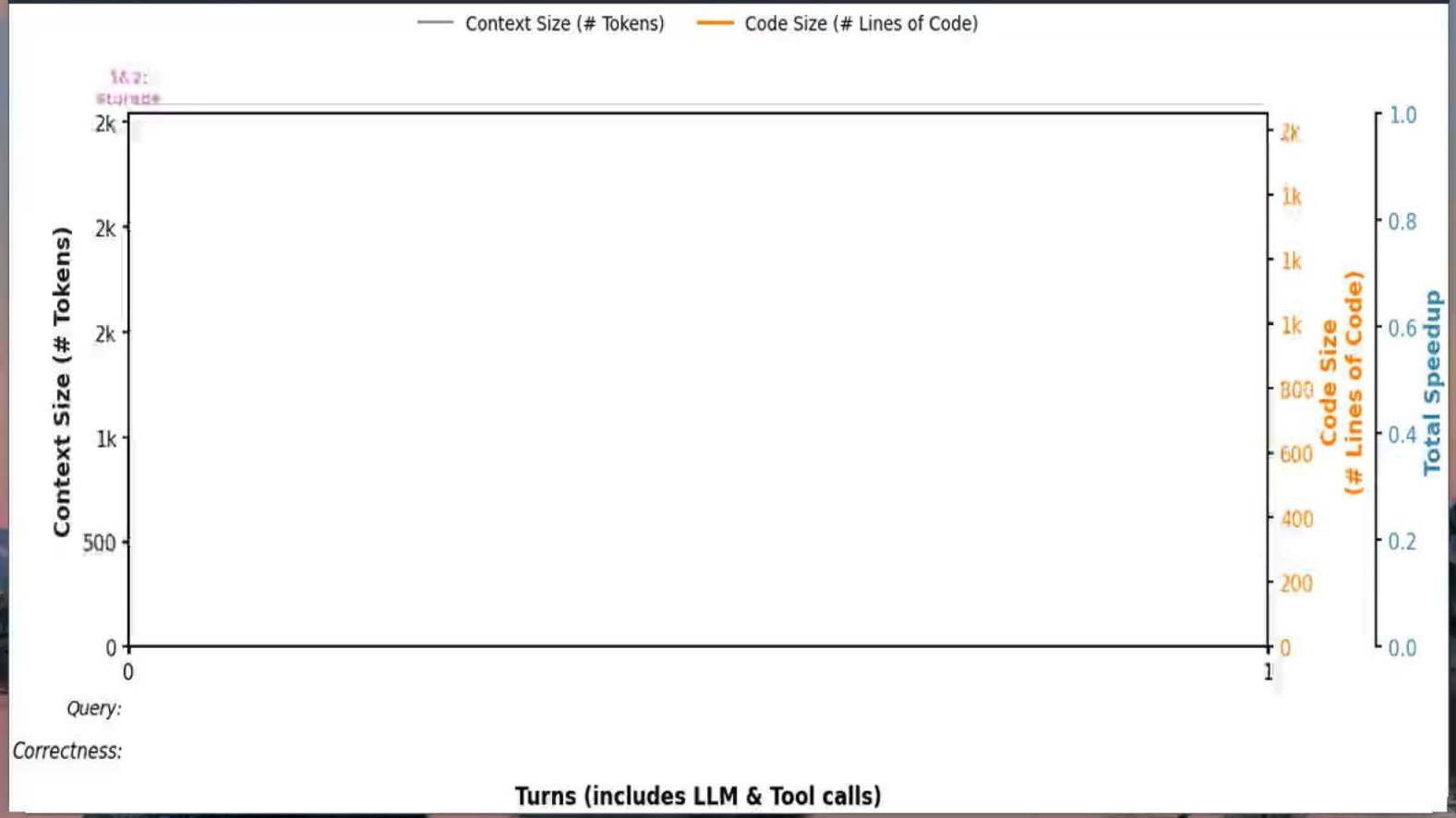


Together these four components form a fast, correct, and recoverable synthesis loop

bespoke_agent.py - Terminal

```
mjasny@fn01:~/bespoke_agent$
```

metrics_live.py



Cost

$0.0084

turn 0 elapsed 0:03

current_prompt.txt

```
Your task is to analyze the workload and produce a creative in-memory
storage-layout summary for the tables accessed by the query. You have
the flexibility to return detailed, free-form text that explores not
only conventional storage-layout recommendations but also
unconventional, novel, and even 'crazy' storage designs.
You are encouraged to include additional ideas, new partitioning
strategies, speculative encoding techniques, or experimental ways of
grouping and organizing columns or data.
For each accessed table, feel free to be inventive and elaborate on
possibilities such as hybrid layouts, speculative SoA/AoS (Array of
Structures/Structure of Arrays) approaches, novel column encodings, or
```

Some Implementation Insights

TPC-H Q5: revenue by *nation* for *one region* and *one year*

🔗 **Stores `lineitems` grouped by `orderkey`**
→ jump directly from each order to its matching lineitems

🧠 **Denormalizes hot attributes**
→ e.g. `lineitem.supp_nationkey` stored next to `orders.cust_nation_key`

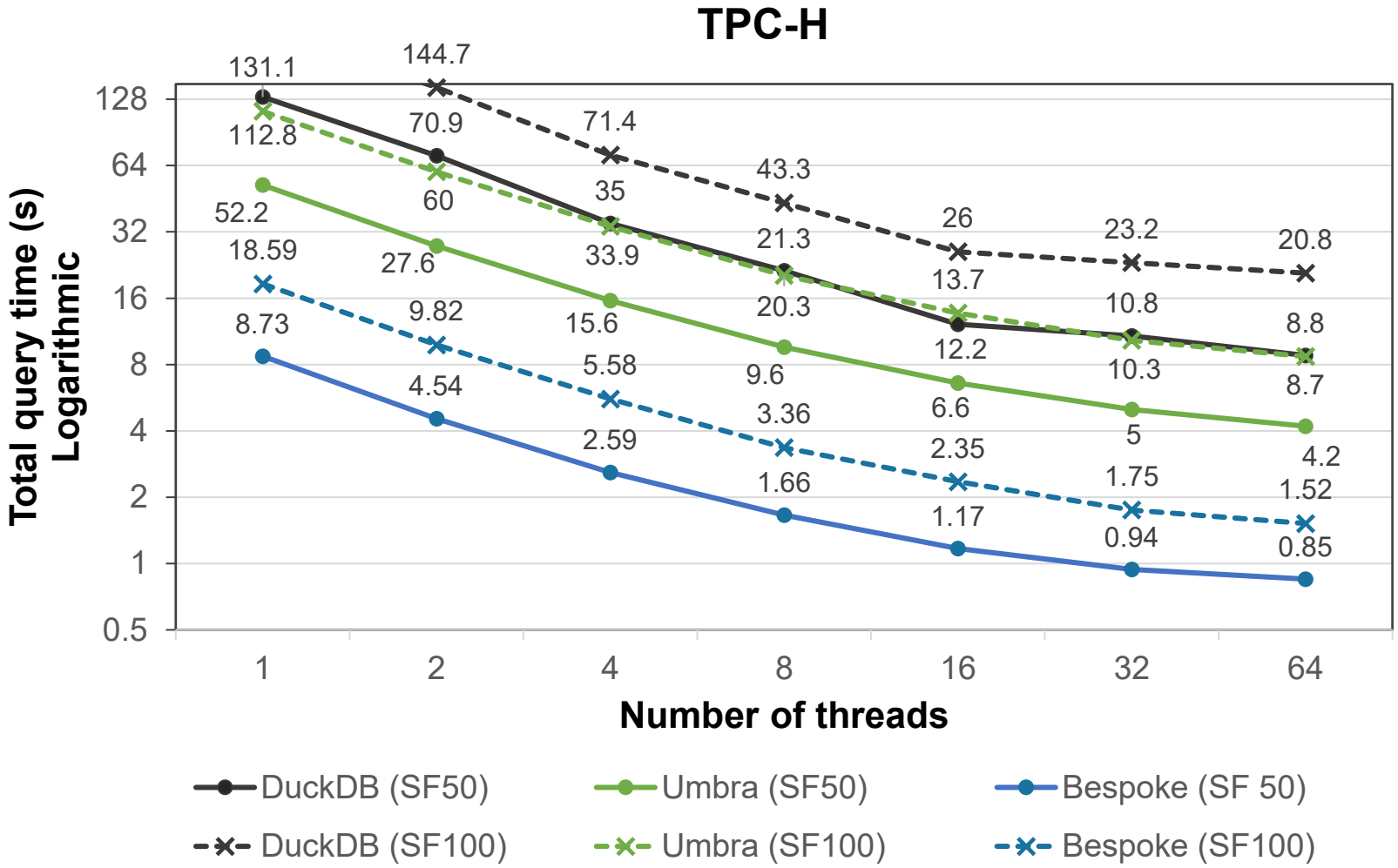
🔎 **Uses data structures for direct lookups to avoid query processing**
→ lookup positions in one data structure and aggregated by using other data structure

Q5 = **direct lookups** and **direct aggregation**

🚀 **67x Speedup over DuckDB**

Bespoke-OLAP effectively codesigns storage & execution

Results: Multi-Threading

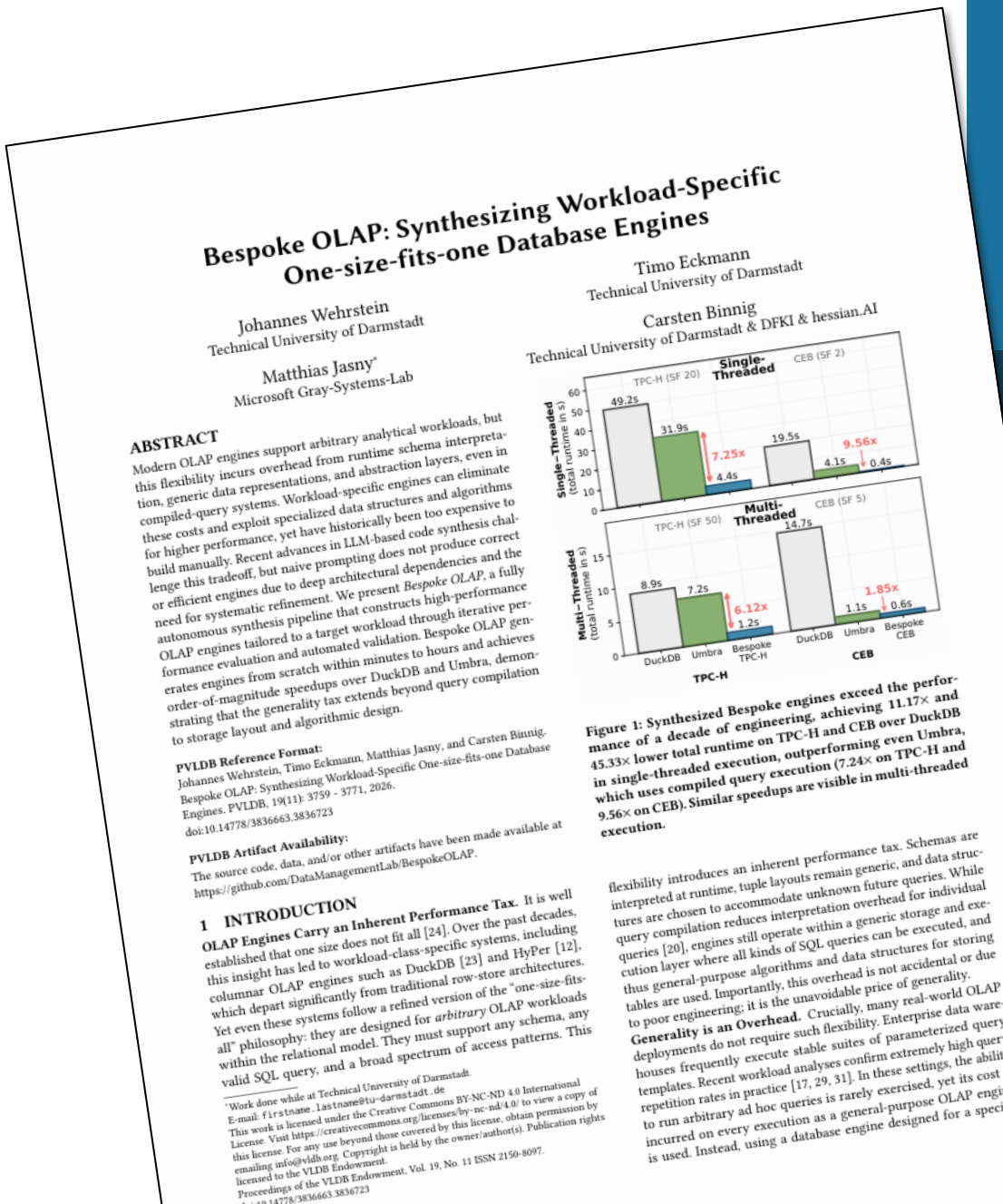


Order of magnitude faster + Scales with number of threads

More details in the paper

More in the paper:

- Aggregate analysis of implementation concepts
- Single vs. Multi-threaded analysis
- Scaling with larger scale-factors
- Impact of the different optimization stages
- Speedups with and without bespoke-storage



Bespoke OLAP: Synthesizing Workload-Specific One-size-fits-one Database Engines

Johannes Wehrstein
Technical University of Darmstadt

Timo Eckmann
Technical University of Darmstadt

Matthias Jasny*
Microsoft Gray-Systems-Lab

Carsten Binnig
Technical University of Darmstadt & DFKI & hessian.AI

ABSTRACT

Modern OLAP engines support arbitrary analytical workloads, but this flexibility incurs overhead from runtime schema interpretation, generic data representations, and abstraction layers, even in compiled-query systems. Workload-specific engines can eliminate these costs and exploit specialized data structures and algorithms for higher performance, yet have historically been too expensive to build manually. Recent advances in LLM-based code synthesis challenge this tradeoff, but naive prompting does not produce correct or efficient engines due to deep architectural dependencies and the need for systematic refinement. We present *Bespoke OLAP*, a fully autonomous synthesis pipeline that constructs high-performance OLAP engines tailored to a target workload through iterative performance evaluation and automated validation. Bespoke OLAP generates engines from scratch within minutes to hours and achieves order-of-magnitude speedups over DuckDB and Umbra, demonstrating that the generality tax extends beyond query compilation to storage layout and algorithmic design.

PVLDB Reference Format:
Johannes Wehrstein, Timo Eckmann, Matthias Jasny, and Carsten Binnig. Bespoke OLAP: Synthesizing Workload-Specific One-size-fits-one Database Engines. PVLDB, 19(11): 3759 - 3771, 2026.
doi:10.14778/3836663.3836723

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DataManagementLab/BespokeOLAP.

Figure 1: Synthesized Bespoke engines exceed the performance of a decade of engineering, achieving 11.17× and 45.33× lower total runtime on TPC-H and CEB over DuckDB in single-threaded execution, outperforming even Umbra, which uses compiled query execution (7.24× on TPC-H and 9.56× on CEB). Similar speedups are visible in multi-threaded execution.

1 INTRODUCTION

OLAP Engines Carry an Inherent Performance Tax. It is well established that one size does not fit all [24]. Over the past decades, this insight has led to workload-class-specific systems, including columnar OLAP engines such as DuckDB [23] and HyPer [12], which depart significantly from traditional row-store architectures. Yet even these systems follow a refined version of the "one-size-fits-all" philosophy: they are designed for *arbitrary* OLAP workloads within the relational model. They must support any schema, any valid SQL query, and a broad spectrum of access patterns. This flexibility introduces an inherent performance tax. Schemas are interpreted at runtime, tuple layouts remain generic, and data structures are chosen to accommodate unknown future queries. While query compilation reduces interpretation overhead for individual queries [20], engines still operate within a generic storage and execution layer where all kinds of SQL queries can be executed, and thus general-purpose algorithms and data structures for storing tables are used. Importantly, this overhead is not accidental or due to poor engineering; it is the unavoidable price of generality.

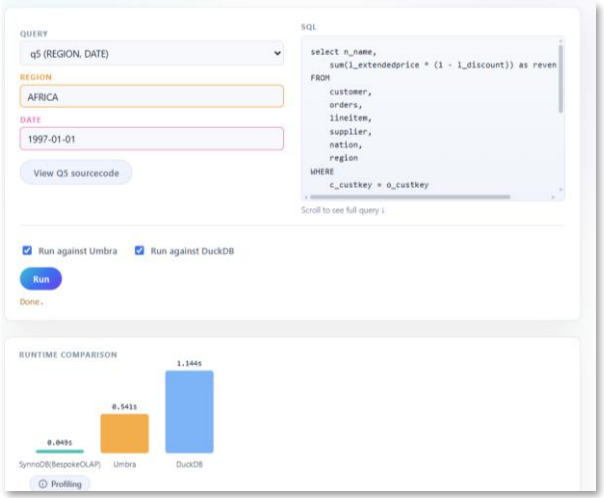
Generality is an Overhead. Crucially, many real-world OLAP deployments do not require such flexibility. Enterprise data warehouses frequently execute stable suites of parameterized query templates. Recent workload analyses confirm extremely high query repetition rates in practice [17, 29, 31]. In these settings, the ability to run arbitrary ad hoc queries is rarely exercised, yet its cost is incurred on every execution as a general-purpose OLAP engine is used. Instead, using a database engine designed for a specific

*Work done while at Technical University of Darmstadt.
E-mail: firstname.lastname@tu-darmstadt.de

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836723

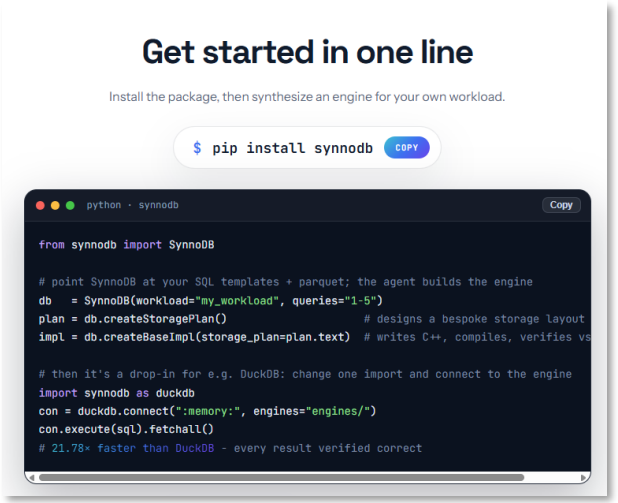
We are starting SynnoDB

Run queries against our
TPC-H engine



synnodb.com/playground

Synthesize your own
DuckDB-Accelerator



synnodb.com/try-out

Learn more: synnodb.com/research/#olap

- Paper
- Source code
- Synthesized Engines

SynnoDB

Synthesize your OLAP &
ETL engines:

SynnoDB synthesizes workload-specific database systems that deliver orders-of-magnitude performance improvements.

synnodb.com