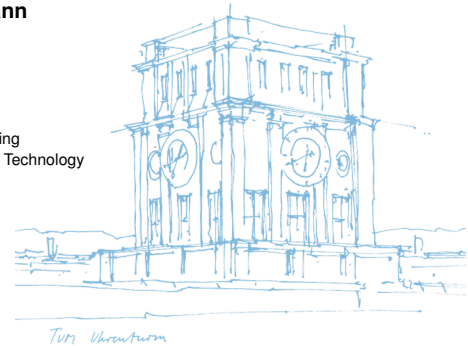
# Practical Course: Database Implementation

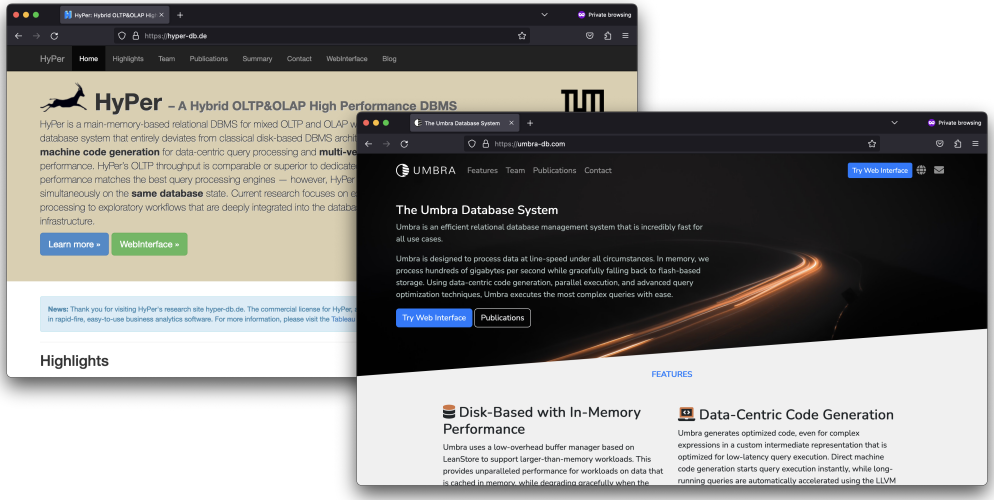
**Adrian Riedl, Simon Ellmann**

Tuesday 8th July, 2025

Chair of Data Science and Engineering
TUM School of Computation, Information and Technology
Technical University of Munich



# Motivation

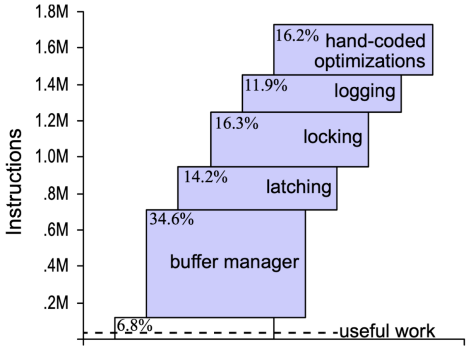

HyPer: Hybrid OLTP&OLAP Hig
Private browsing
https://hyper-db.de
HyPer
Home
Highlights
Team
Publications
Summary
Contact
WebInterface
Blog
HyPer – A Hybrid OLTP&OLAP High Performance DBMS
HyPer is a main-memory-based relational DBMS for mixed OLTP and OLAP w
database system that entirely deviates from classical disk-based DBMS archi
machine code generation for data-centric query processing and multi-ve
performance. HyPer's OLTP throughput is comparable or superior to dedicate
performance matches the best query processing engines — however, HyPer
simultaneously on the same database state. Current research focuses on e
processing to exploratory workflows that are deeply integrated into the databa
infrastructure.
Learn more »
WebInterface »
News: Thank you for visiting HyPer's research site hyper-db.de. The commercial license for HyPer,
in rapid-fire, easy-to-use business analytics software. For more information, please visit the Tableau
Highlights
The Umbra Database System
Private browsing
https://umbra-db.com
UMBRA
Features
Team
Publications
Contact
Try Web Interface
The Umbra Database System
Umbra is an efficient relational database management system that is incredibly fast for all use cases.
Umbra is designed to process data at line-speed under all circumstances. In memory, we process hundreds of gigabytes per second while gracefully falling back to flash-based storage. Using data-centric code generation, parallel execution, and advanced query optimization techniques, Umbra executes the most complex queries with ease.
Try Web Interface
Publications
FEATURES
Disk-Based with In-Memory Performance
Umbra uses a low-overhead buffer manager based on LeanStore to support larger-than-memory workloads. This provides unparalleled performance for workloads on data that is cached in memory, while degrading gracefully when the
Data-Centric Code Generation
Umbra generates optimized code, even for complex expressions in a custom intermediate representation that is optimized for low-latency query execution. Direct machine code generation starts query execution instantly, while long-running queries are automatically accelerated using the LLVM

# Motivation

- We build a tiny **main-memory compiling database system**!
- We focus on correctness and *speeeeed*[1] ...

[1] Warning: Query compilation will be slow

# Motivation




| OLTP properties and new platforms | DBMS modification |
|---|---|
| logless architectures | remove logging |
| partitioning, commutativity | remove locking (when applicable) |
| one transaction at a time | single thread, remove locking, remove latching |
| main memory resident | remove buffer manager, directory |
| transaction-less databases | avoid transaction bookkeeping |

Figure 1: *OLTP Through the Looking Glass, and What We Found There* by Harizopoulos et al. (SIGMOD '08)

# Demo

# Contents

- **Data representation** in memory and **query execution**
- Building a **shell** for our DBMS
- **Lexing** and **parsing** SQL
- **Algebra trees** and **query optimization**[2]
- **Code generation** and **execution**
- **Multithreading**

[2] Only predicate pushdown

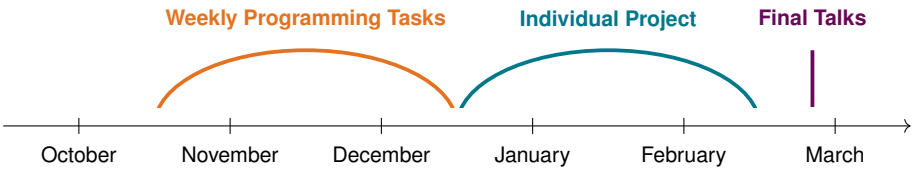
# Course Structure

**Weekly Programming Tasks** ($\sim$ 7 weeks):

- **Weekly meetings** and **assignments**

**Individual Project** ($\sim$ 7 weeks):

- Work on a state-of-the-art topic
- Present **your work** $\rightarrow$ **Final Talks**



# Course Structure

**Weekly Programming Tasks**:

- **1-2 students present their solution** for the previous assignment
- Introduction to a **new topic** + **assignment**
- Code **should** be submitted before next meeting

# Course Structure

**Individual Project**:

- Try to **reproduce a paper**, *or*
- **Implement some fancy feature** in your database, *or*
- Work on a **topic provided by a PhD student** of the chair

→ No more weekly course meetings, instead individual ones with your advisor

# Course Structure

**Final Talk**:

- **Present your project** in 10 to 15 minutes
- **On-site meeting** for all presentations + discussions (usually $\sim$ 3h)
- Date for meeting up to you (e.g., end of lectures vs. end of semester)


GitLab


Mattermost

# Organization

Tasks:

- We provide templates for the programming tasks in C++ and Rust
- The tasks build on each other → Decide early on the programming language!
- You may use helper libraries / crates (e.g., for text formatting, error handling, ...) that do not affect the overall implementation effort

Grading:

- Programming tasks and individual project (including the final talk) are graded roughly 50% each
- We will grade after the final presentations (which is also the deadline for your code)

# Registration

- Solve the qualification challenge: https://db.in.tum.de/teaching/ws2526/imlab/
- **Deadline: 22nd of July**
- Register via the matching system!

Questions?