

Architecting and Engineering Main Memory Database Systems in Modern C



Projects

Marcus Pinnecke, M.Sc.

Research associate / Working Group Database & Software Engineering
Institute of Technical and Business Information Systems (ITI)



Proceeding

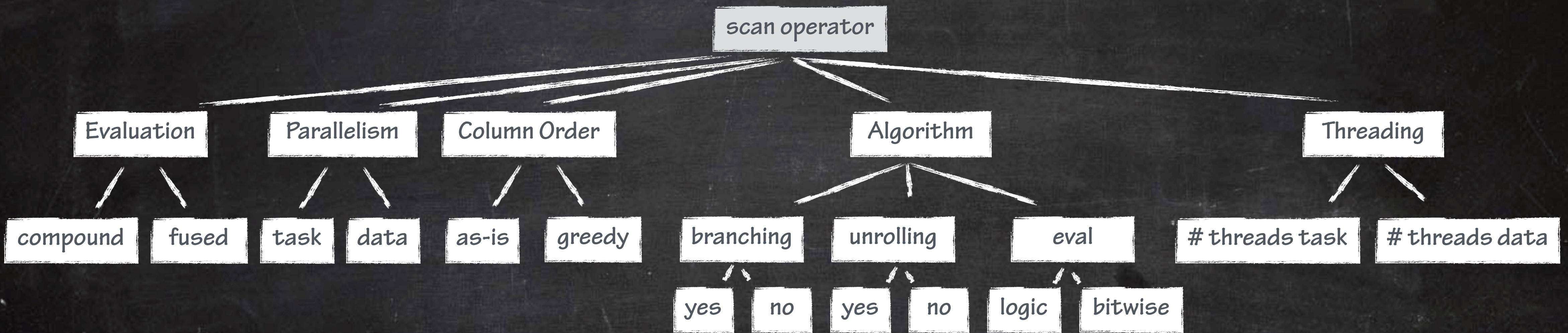
1. Form a team (from 2 to 3 students)
(keep the exam schedule in mind; all team mates are examined at the same day)
2. Mail your team mate names to your tutor
3. Pick an exclusive project (see next slides)
(first comes, first serves)
4. Work on your project (team-based) after discussion w/ project owner
(in case of troubles: 1st level support is your tutor, 2nd level support is the owner)

Dates

- Latest date 21th of December 2017

In case you do not form a team, we perform the team group on that day
In case you do not pick a project, we assign you one on that day

Extend REACTS Physical Operator Pool



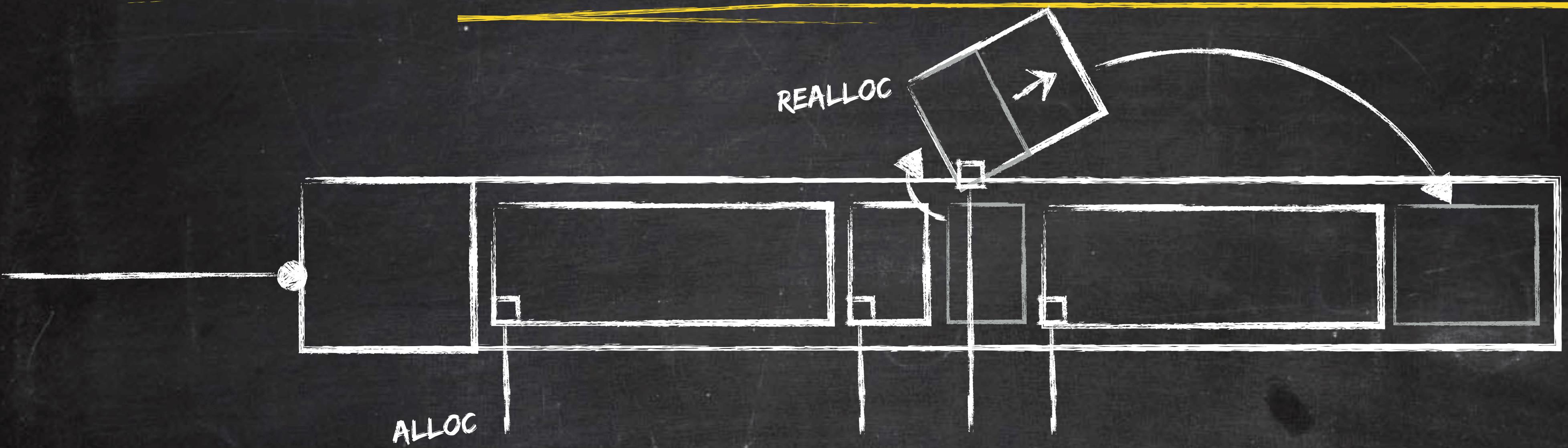
Context: Self-managed tuning and reorganization of long-standing (physical) queries plans

Task: Implement 5+ scan operator variants for our (data stream) query engine REACTS & briefly evaluate these implementations

Audience: Practical experienced students

Language: C

Cache-Efficient Memory Allocation



Context: Random access lead to cache trashing. An (custom) generic memory allocator is needed for „in-block“ memory allocation for nested structures (c.f., linearization in ELF)

Task: Design and implement a custom memory allocator that manages allocations in one continuous memory block. Challenges are low-level + memory fragmentation + reorganization.

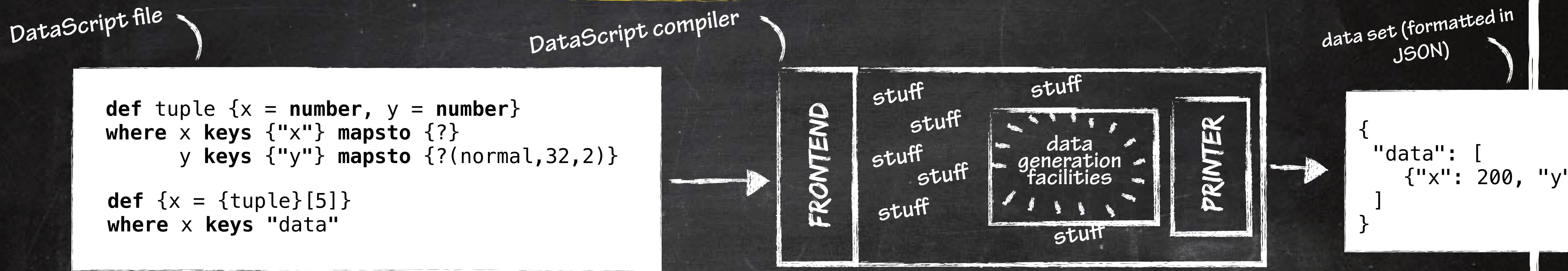
Audience: Practical experienced students (advanced skills required)

Language: C

Project #3

Project owner
Marcus Pinnecke

Data Generation for the DataScript Language



Context: Benchmarking is important for comparison, evaluation, practice and science. We designed a benchmark generator-generator, the DataScript language, that promises to eliminate the need for custom benchmark generator tools and micro-benchmarks.

Task: Implement core facilities in the DataScript compiler to generate datasets according given properties (no parsing required)

Audience: Practical interested students with fundamental C skills, knowledge in statistics

Language: C or Java

Project #4

Project owner
Marcus Pinnecke

A Stock Data Stream Micro-Benchmark Generator



Context: Evaluation of REACTS: Long-standing queries are affected by changes in the working set (e.g., data distribution affects selectivities + lead to suboptimal (running) plans.

Task: Implement a data stream generator that produces a stream of stock data. During runtime, parameters of this data generation are mutable.

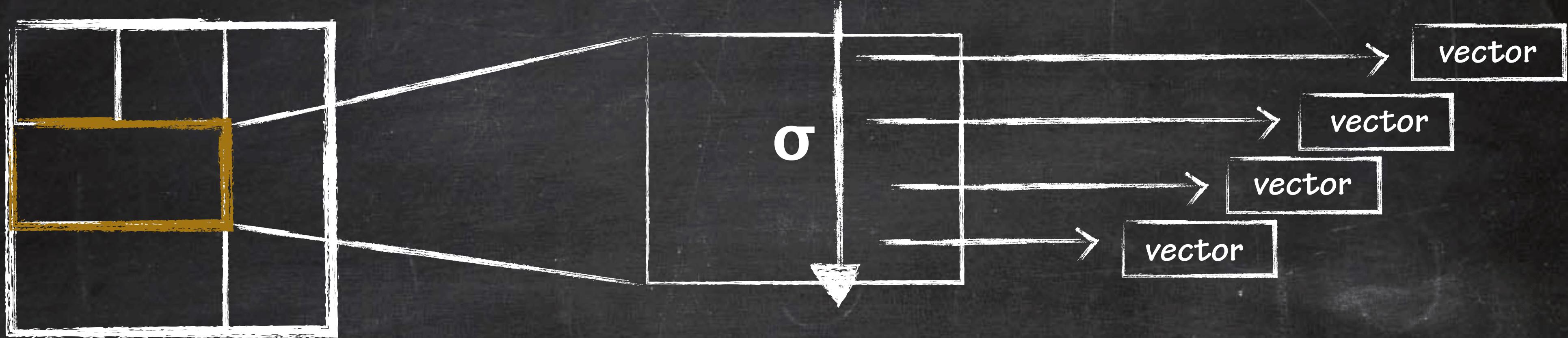
Audience: Practical interested students with fundamental C skills

Language: C or C++ (the later with cross compilation)

Project #5

Project owner
Marcus Pinnecke

Vectorized Table Scan Operators in GridStore (Fragment-Level)



Context: The GridStore is our HTAP storage engine for structured data. Its a highly flexible fragmentation scheme supporting diverse storage models. At its core, everything is about data fragments (think of it as row- and column-stores).

Task: Implement a vectorized table scan operator that runs on column-wise and row-wise data (low-level operations)

Audience: Skilled C programming students

Language: C

Project #6
count-based
jumping windows

Project #7
count-based
sliding windows

Project #8
time-based
jumping windows

Project #9
time-based
sliding windows

Project owner
Marcus Pinnecke

Data Stream Windows in REACTS



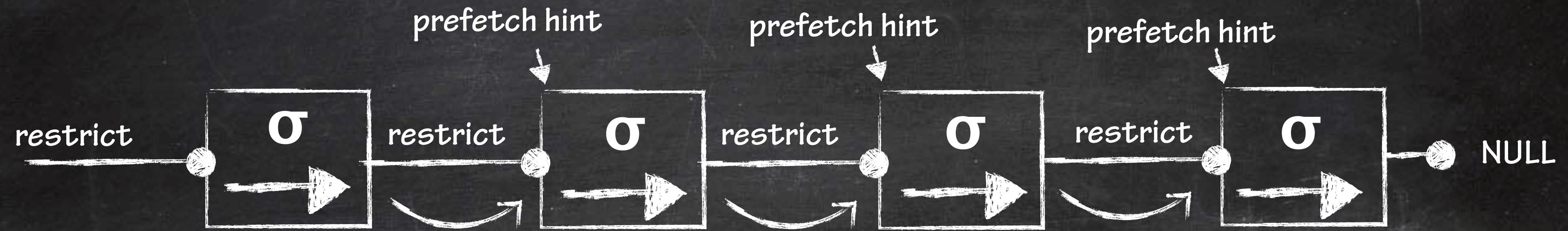
Context: Data streams are potential infinite in size. Hence, streaming data is buffered in „windows“ and continuously processed. Windows are either jumping or sliding and either count-based or time-based. We implemented a count-based tumbling window (jumping window with no overlapping).

Task: Implement either jumping or sliding windows, either count or time-based according the semantics of these windows in stream processing.

Audience: Skilled C programming students

Language: C

Evaluation of Low-Level Micro-Optimization



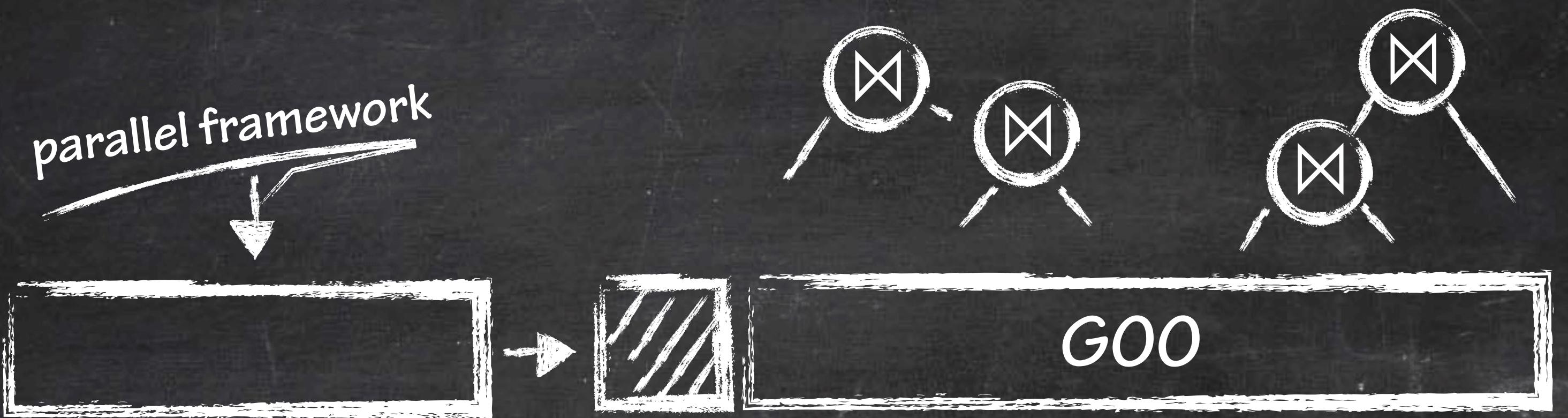
Context: Low-level (compiler) optimization and hints promise performance benefits for main memory system. In context of data-intense systems, we are interested in the impact of particular optimization techniques.

Task: Evaluate the impact of restrict specifier for query processing, CPU prefetching hints for random access, and branching hints to the compiler.

Audience: Interested C programming students with scientific background

Language: C, and statistic tools + plotting (e.g., R)

Integration of Parallel Framework into GOO



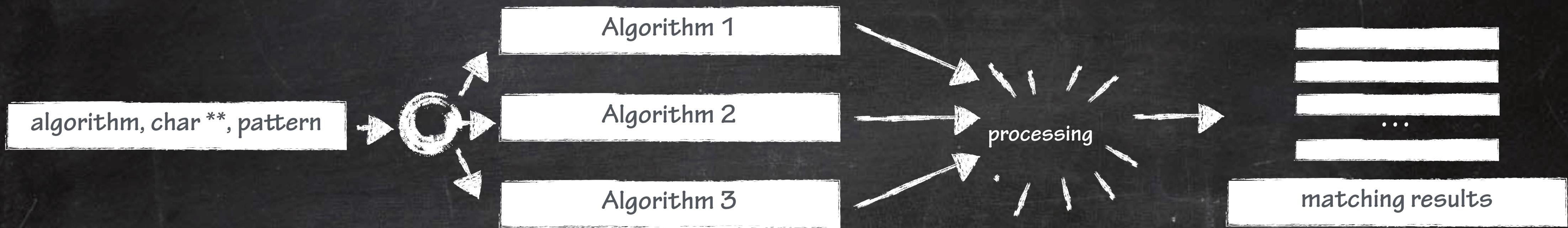
Context: The task of this project is the integration of the functionality of a parallel computation framework, e.g `boost::compute`, into our GOO framework (GPU-optimized optimizer) for join ordering.

Task: Your task is to implement the interface by using the chosen parallel computation framework.

Audience: Students with skills C++ programming, and parallel programming.

Language: C++, CMake, and API of chosen parallel framework

Selective String Pattern Matching



Context: Selection with typical WHERE LIKE x clause: input is *algorithm*, *string array*, *pattern* and output is *selected strings*. Fast selection depends on underlying algorithm, such as Boyer-Moore, Knuth-Morris-Pratt, or Aho-Corasick.

Task: Implement a selection with string pattern matching based on 3+ promising string pattern matching algorithm. (more details by Bala)

Audience: Students with fundamental experience in C/C++ programming.

Language: C or C++

Efficient IN-Predicate Evaluation

$$\text{IN}(\boxed{A_1} \boxed{A_2} \dots \boxed{A_n}, \boxed{B_1} \boxed{B_2} \dots \boxed{B_m}) = \boxed{b_1 b_2 b_3, \dots, b_m}$$

Context: *IN-predicates* are an important filter in DBMSs: it is a function that determines whether a tuple's value is contained in a list of values. Input is a column (integer array), an array of constants, and output: a bit mask of matching tuples.

Task: Implement a function to evaluate IN-Predicates and compare its performance to a baseline. We can provide a data parallel (SIMD) function as baseline.

Audience: Students experienced in C/C++ programming.

Language: C or C++