Research and Implementation of Database Concepts

Kickoff

Martin Boissier, Thomas Bodner, Markus Dreseler, Jan Koßmann, Dr. Michael Perscheid
Enterprise Platform and Integration Concepts

Prologue

- Feel free to use the chat function for questions
- In the unlikely case of any disruptions, we will update the webpage in real-time:
- hpi.de/plattner - Teaching - Winter Term - Research and Implementation of DB Concepts

EPIC Teaching Activities

	Winter Term	Summer Term
Bachelor	**Scalable Software Engineering** (Lecture, 5th term, revised SWT II)	**Foundation of Business Software** (Lecture, 4th term)
	Bachelor Project (5th and 6th term) Example: SAP Graph – Reinvent the Development Experience for the Intelligent Enterprise	
Master	**Trends and Concepts in the Software Industry II** (Seminar with Prof. Plattner and target-actual comparison of enterprise software at customers)	**Trends and Concepts in the Software Industry I** (Lecture with Prof. Plattner and industry partners) **Trends and Concepts in the Software Industry III** (Optional Project Seminar)
	Data-driven Decision Support (Lecture and Project)	**Causal Inference** (Lecture and Project)
	Research on Database Concepts (Research Seminar)	**Build Your Own Database** (Lecture and Project)
	Master Projects Data-driven Decision Support	 In-Memory Data Management

What to expect?

- Better understand how database systems work
- Learn how to familiarize yourself with a larger code base
- Work in small teams on a larger project

Same as in the Develop your own Database (DYOD) seminar

- Gain experience in systems development
- Improve your C++(2a) skills

Less of a focus than in DYOD

- Research experience
- Conduct experiments, visualize results, communicate findings

New in this seminar

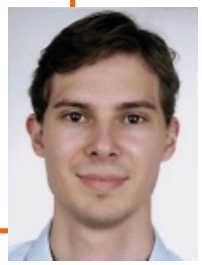
How does this relate to Develop your own Database?

- We found that thesis students often have little experience in communicating their results
- This seminar is supposed to be a „thesis light“, including literature research, implementation, designing and executing experiments, and presenting the results in speech and writing
- It is both suitable for those students who have taken DYOD and for those who have not
- BUT: No weekly meetings with the entire group, thus no DBMS/C++ introduction
  - Previous experience, e.g, from Trends and Concepts or the DBS lectures is helpful
  - DYOD slides and sprint documents are available if you want to read up on details
- More research-oriented, i.e., the projects are proposals, not full specifications

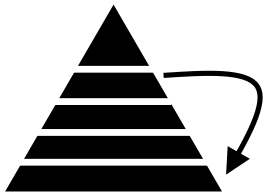
Who are we?

Cloud Data Management

Thomas Bodner



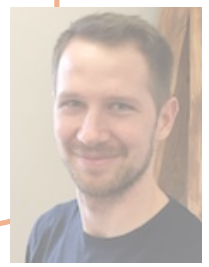
NVM & Tiering



Markus Dreseler

Database Replication

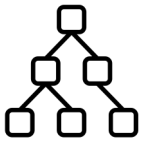
Stefan Halfpap

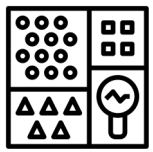


Compression

Martin Boissier

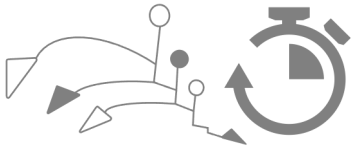
Physical DB Design





Jan Kossmann

Spatio-Temporal Data



Keven Richly

Not in this Seminar

Hyrise

- An In-Memory Storage Engine for Hybrid Transactional and Analytical Processing
- HYRISE is a research database for the systematic evaluation of new concepts for hybrid transactional and analytical data processing on modern hardware
- Developed with and by HPI students
- Open Source (https://git.io/hyrise)
- System paper published at EDBT'19

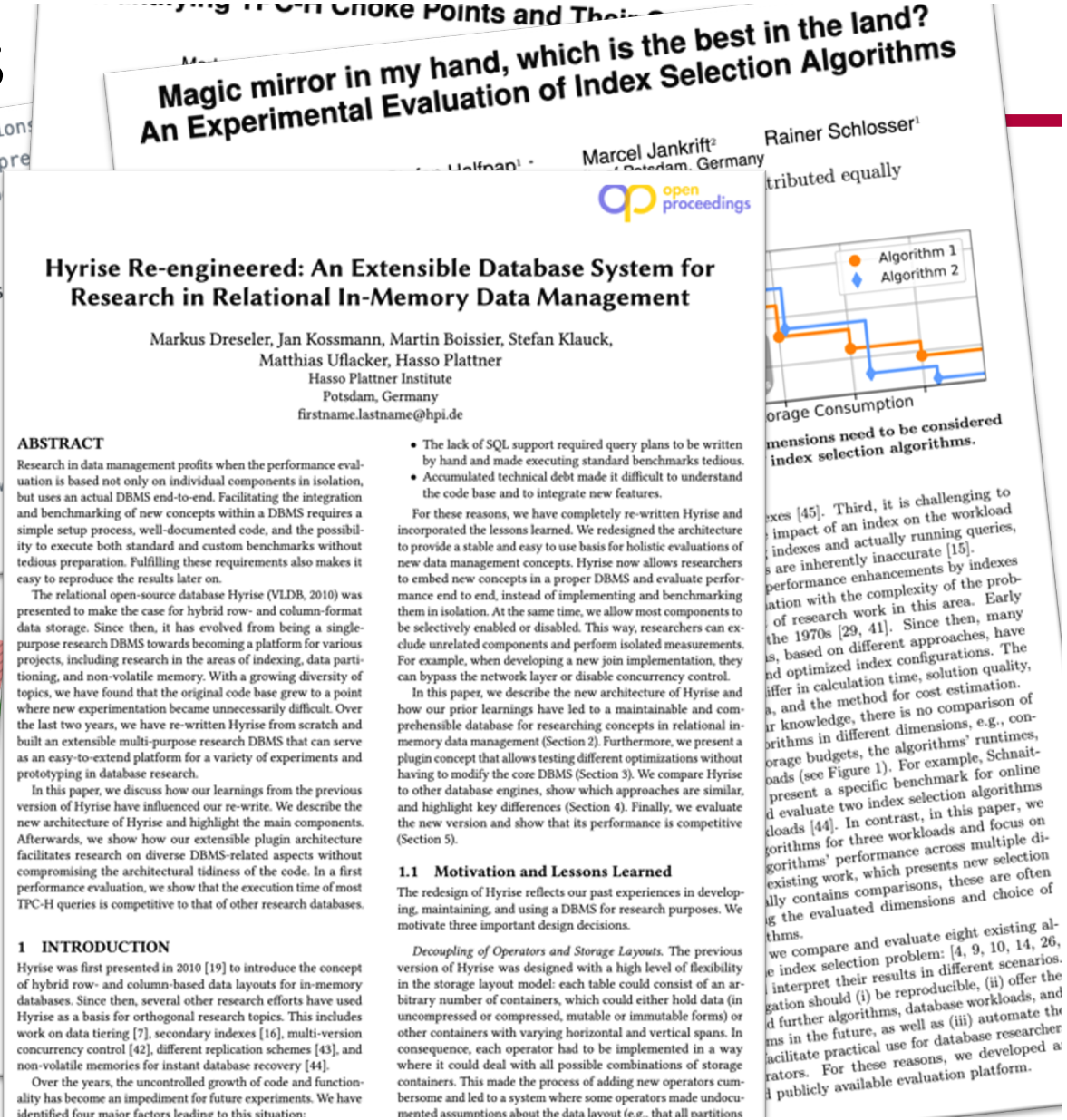
- Modern, documented C++20 code base, 93% test coverage
- SQL interface, PostgreSQL network protocol
- Easy to extend via plug-in interface
- Supported benchmarks: TPC-(C|H|DS), JCC-H, Join-Order
- Runs on Intel, IBM Mainframe, ARM, Raspberry PI

Hyrise in three* pictures

```
s multi-predicate joins are expensive, we do not want to create semi join reductions
Look at each predicate of the join independently. We can do this as a JoinNode's pre
Disjunctive predicates are currently not supported and if they were, they would b
join_predicates entry.
r (const auto& join_predicate : join_node->join_predicates()) {
const auto predicate_expression = std::dynamic_pointer_cast<BinaryPredicateExpres
DebugAssert(predicate_expression, "Expected BinaryPredicateExpression");
if (predicate_expression->predicate_condition != PredicateCondition::Equals) {
  continue;
}

// Since semi join reductions might be beneficial for both sides of the join,
// which can deal with both sides.
const auto reduce_if_beneficial = [&](const auto side_of_join) {
```

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q1
Hyper	**498**	47	969	725	821	207	804	467	1,782	8
Umbra	1,258	119	**676**	**629**	**536**	196	665	435	1,938	**57**
MonetDB	7,488	**31**	816	1,185	1,505	372	1,965	1,136	**1,252**	70
DuckDB	4,874	2,083	2,080	2,283	2,533	723	4,832	2,217	32,681	3,37
Hyrise Paper	13,559	57	1,719	1,796	3,579	50	1,404	1,078	9,740	4,49
Hyrise WIP	6,369	50	976	643	2,854	**36**	**564**	**422**	6,712	2,282



Magic mirror in my hand, which is the best in the land?
An Experimental Evaluation of Index Selection Algorithms

Rainer Schlosser[1]

tributed equally

orage Consumption

mensions need to be considered
index selection algorithms.

xes [45]. Third, it is challenging to
impact of an index on the workload
indexes and actually running queries,
s are inherently inaccurate [15].
performance enhancements by indexes
ation with the complexity of the prob-
of research work in this area. Early
the 1970s [29, 41]. Since then, many
s, based on different approaches, have
nd optimized index configurations. The
iffer in calculation time, solution quality,
, and the method for cost estimation.
r knowledge, there is no comparison of
orithms in different dimensions, e.g., con-
orage budgets, the algorithms' runtimes,
oads (see Figure 1). For example, Schnait-
present a specific benchmark for online
d evaluate two index selection algorithms
loads [44]. In contrast, in this paper, we
gorithms for three workloads and focus on
gorithms' performance across multiple di-
existing work, which presents new selection
lly contains comparisons, these are often
g the evaluated dimensions and choice of
thms.
we compare and evaluate eight existing al-
e index selection problem: [4, 9, 10, 14, 26,
interpret their results in different scenarios.
gation should (i) be reproducible, (ii) offer the
d further algorithms, database workloads, and
ms in the future, as well as (iii) automate the
acilitate practical use for database researcher
rators. For these reasons, we developed a
d publicly available evaluation platform.

open proceedings

Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management

Markus Dreseler, Jan Kossmann, Martin Boissier, Stefan Klauck,
Matthias Uflacker, Hasso Plattner
Hasso Plattner Institute
Potsdam, Germany
firstname.lastname@hpi.de

ABSTRACT

Research in data management profits when the performance evaluation is based not only on individual components in isolation, but uses an actual DBMS end-to-end. Facilitating the integration and benchmarking of new concepts within a DBMS requires a simple setup process, well-documented code, and the possibility to execute both standard and custom benchmarks without tedious preparation. Fulfilling these requirements also makes it easy to reproduce the results later on.

The relational open-source database Hyrise (VLDB, 2010) was presented to make the case for hybrid row- and column-format data storage. Since then, it has evolved from being a single-purpose research DBMS towards becoming a platform for various projects, including research in the areas of indexing, data partitioning, and non-volatile memory. With a growing diversity of topics, we have found that the original code base grew to a point where new experimentation became unnecessarily difficult. Over the last two years, we have re-written Hyrise from scratch and built an extensible multi-purpose research DBMS that can serve as an easy-to-extend platform for a variety of experiments and prototyping in database research.

In this paper, we discuss how our learnings from the previous version of Hyrise have influenced our re-write. We describe the new architecture of Hyrise and highlight the main components. Afterwards, we show how our extensible plugin architecture facilitates research on diverse DBMS-related aspects without compromising the architectural tidiness of the code. In a first performance evaluation, we show that the execution time of most TPC-H queries is competitive to that of other research databases.

1 INTRODUCTION

Hyrise was first presented in 2010 [19] to introduce the concept of hybrid row- and column-based data layouts for in-memory databases. Since then, several other research efforts have used Hyrise as a basis for orthogonal research topics. This includes work on data tiering [7], secondary indexes [16], multi-version concurrency control [42], different replication schemes [43], and non-volatile memories for instant database recovery [44].

Over the years, the uncontrolled growth of code and functionality has become an impediment for future experiments. We have identified four major factors leading to this situation:

- The lack of SQL support required query plans to be written by hand and made executing standard benchmarks tedious.
- Accumulated technical debt made it difficult to understand the code base and to integrate new features.

For these reasons, we have completely re-written Hyrise and incorporated the lessons learned. We redesigned the architecture to provide a stable and easy to use basis for holistic evaluations of new data management concepts. Hyrise now allows researchers to embed new concepts in a proper DBMS and evaluate performance end to end, instead of implementing and benchmarking them in isolation. At the same time, we allow most components to be selectively enabled or disabled. This way, researchers can exclude unrelated components and perform isolated measurements. For example, when developing a new join implementation, they can bypass the network layer or disable concurrency control.

In this paper, we describe the new architecture of Hyrise and how our prior learnings have led to a maintainable and comprehensible database for researching concepts in relational in-memory data management (Section 2). Furthermore, we present a plugin concept that allows testing different optimizations without having to modify the core DBMS (Section 3). We compare Hyrise to other database engines, show which approaches are similar, and highlight key differences (Section 4). Finally, we evaluate the new version and show that its performance is competitive (Section 5).

1.1 Motivation and Lessons Learned

The redesign of Hyrise reflects our past experiences in developing, maintaining, and using a DBMS for research purposes. We motivate three important design decisions.

Decoupling of Operators and Storage Layouts. The previous version of Hyrise was designed with a high level of flexibility in the storage layout model: each table could consist of an arbitrary number of containers, which could either hold data (in uncompressed or compressed, mutable or immutable forms) or other containers with varying horizontal and vertical spans. In consequence, each operator had to be implemented in a way where it could deal with all possible combinations of storage containers. This made the process of adding new operators cumbersome and led to a system where some operators made undocumented assumptions about the data layout (e.g., that all partitions

HPI Hasso Plattner Institut

Skyrise

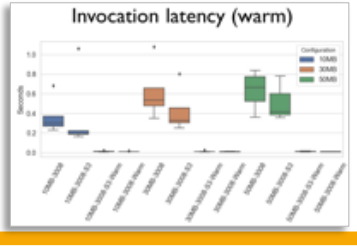
- A cloud-native query processor for interactive ad-hoc analytics on cold data
  - Cold data: Large amounts of (mostly) infrequently accessed data
  - Ad-hoc analytics: Dynamic OLAP workloads
  - Interactivity: Query latencies in seconds
  - SQL query processing: Relational query execution and optimization
  - Modern cloud infrastructure: Function as a service platforms and shared object stores
- Initiated in late 2019 as sister project to our established Hyrise project
  (v1: Grund et al. VLDB ’10, v2: Dreseler et al. EDBT ’19)
  - Different use cases, research problems and approaches
  - Development model inspired by Hyrise (student contributions, code setup and quality)
  - Just starting out!
- Vision paper published at VLDB ’20

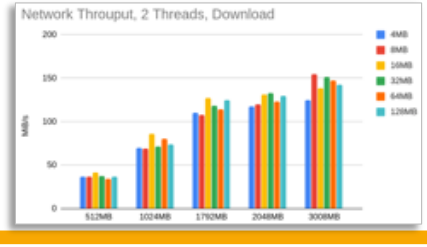

Elastic Query Processing
on Function as a Service Platforms

Thomas Bodner
supervised by Hasso Plattner
Enterprise Platform and Integration Concepts Group
Hasso Plattner Institute
Potsdam, Germany
thomas.bodner@hpi.de

ABSTRACT

1. INTRODUCTION





Proposed Research Topics

- Skew in Databases: JCCH benchmark & Histograms in Hyrise
- Column Compression
- Cost-Performance Tradeoffs in Query Execution on Cloud Function
- Object Metadata Management for Cloud Storage
- Quantifying the Overhead of Iterator Abstractions
- Understanding Memory Consumption with Tracking Allocators
- diPs: data-induced Predicates
- What if?-Optimization for Autonomous Clustering

Skew in Databases: JCCH benchmark & Histograms

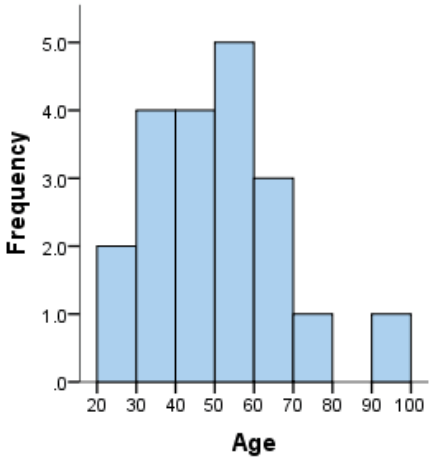
- Data distribution in real-world systems is often highly skewed
  - Example: Amazon does not sell the same volume every month, peaks such as shopping days (e.g., Black Friday) or christmas introduce skew
  - in contrast, benchmarks such as TPC-H often generate data uniformly distributed
- JCCH is an adaption of TPC-H
  - Introduces skew to both workload and data set
- Skew in Hyrise
  - Optimizer and execution engine often make uniformity assumptions
  - Histograms per column to recognize certain skew within the data

JCCH – Analyze and Optimize

- 1st step: analyze JCCH‘s performance and compare it to TPC-H
  - What are the pain points?
  - Where does Hyrise improve, where does it degrade?
- 2nd step: Hypothesize what the reason for the changes are
  - Execute mini experiments to (in-)validate your hypotheses
- 3nd step: Draw ways to mitigate/eliminate these problems
  - Simple (mock) implementations to show how certain pain points can be tackled
  - If time permits, integrate these changes into Hyrise‘s master

Histograms in Hyrise

- Currently, histograms are a relatively static data structure in Hyrise
- Each column of a tables has an histogram, max. 100 bins if sufficient distinct values
- Drawbacks:
  - Coarse-grained: required for efficient estimations within the optimizer, but no knowledge about local differences
  - Not updatable
  - Slow to generate



Histograms in Hyrise

- First part:
  - Quantify short-comings of current approach
  - Run benchmarks with skewed data (e.g. JCCH and JOB) and analyze
- Second part:
  - Create concept how to fix these short comings
    - E.g., small mergable histograms per segment
  - Implement your fixes

Column Compression

- Parts that constitute a column (so called segments) can be encoded in various forms in Hyrise (e.g., dictionary encoding, frame-of-reference, run-length encoding, LZ4)
- Hyrise heavily uses dictionary encoding, but is limited to byte-sized (i.e., elements are either 8, 16, or 32bits wide)
- Light-weight integer compression schemes have seen a lot of research lately and improved significantly
- Tasks:
  - Implement and evaluate modern compression libraries (e.g., bit compression for attributes vectors and PFOR delta encoding for integer columns)
  - Measure impact on standard benchmarks and analyze

Cost-Performance Tradeoffs in Query Execution on Cloud Functions

- Elastic compute resources allow choosing infrastructure per query (operator)
- Enable multi-objective (cost vs. performance) query optimization
- Requirement for degrees of freedom in query execution:
  - Pre-provisioning of cloud functions to avoid coldstart latencies
  - Interleaved materialized execution to reduce the impact of stragglers
  - Staged data exchange operator for reduced parallelism and storage request cost
  - Combine cheap but slow storage with fast but expensive storage
- Prior experience with cloud (function) services benefitial

M. Perron, R. C. Fernandez, D. DeWitt, S. Madden. Starling: A Scalable Query Engine on Cloud Function. SIGMOD 2020.
Q. Pu, S. Venkataraman, I. Stoica. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. NSDI 2019.

Object Metadata Management for Cloud Storage

- Cloud object storage systems store petabytes of data in millions of objects cost-efficiently
- They come with their own set of challenges:
  - Weak data consistency guarantees
  - Simplistic data access APIs
  - Poor request latencies and high per-request cost
- Requirement for table format for effective and efficient query processing on top:
  - Concurrency control
  - Fast file listing
  - Data pruning
- Prior experience with cloud (object storage) services beneficial

M. Armbrust. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. VLDB 2020.
Apache ORC and Iceberg.

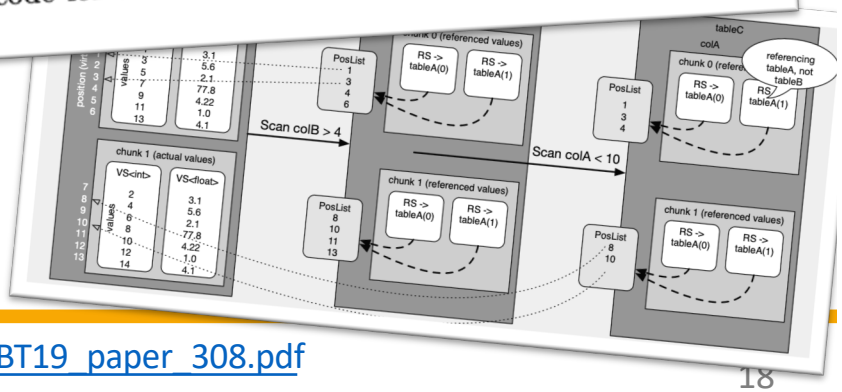
Quantifying the Overhead of Iterator Abstractions

- In Hyrise, we support different types of encoding (e.g., unencoded, dictionary encoding, …)
- Additionally, rows may be accessed indirectly, e.g., after a scan
- When we write an operator, we want to simply iterate over the data without having to consider the encoding
- Hyrise uses iterators that hide all of this

```
auto segment = table->get_chunk(0)->get_segment(0);
segment_with_iterators(segment, [](auto it, auto end) {
  while (it != end) {
    std::cout << it->value << "\n";
  }
});
```

Listing 3.1: Example code for accessing the data stored in a segment through iterators

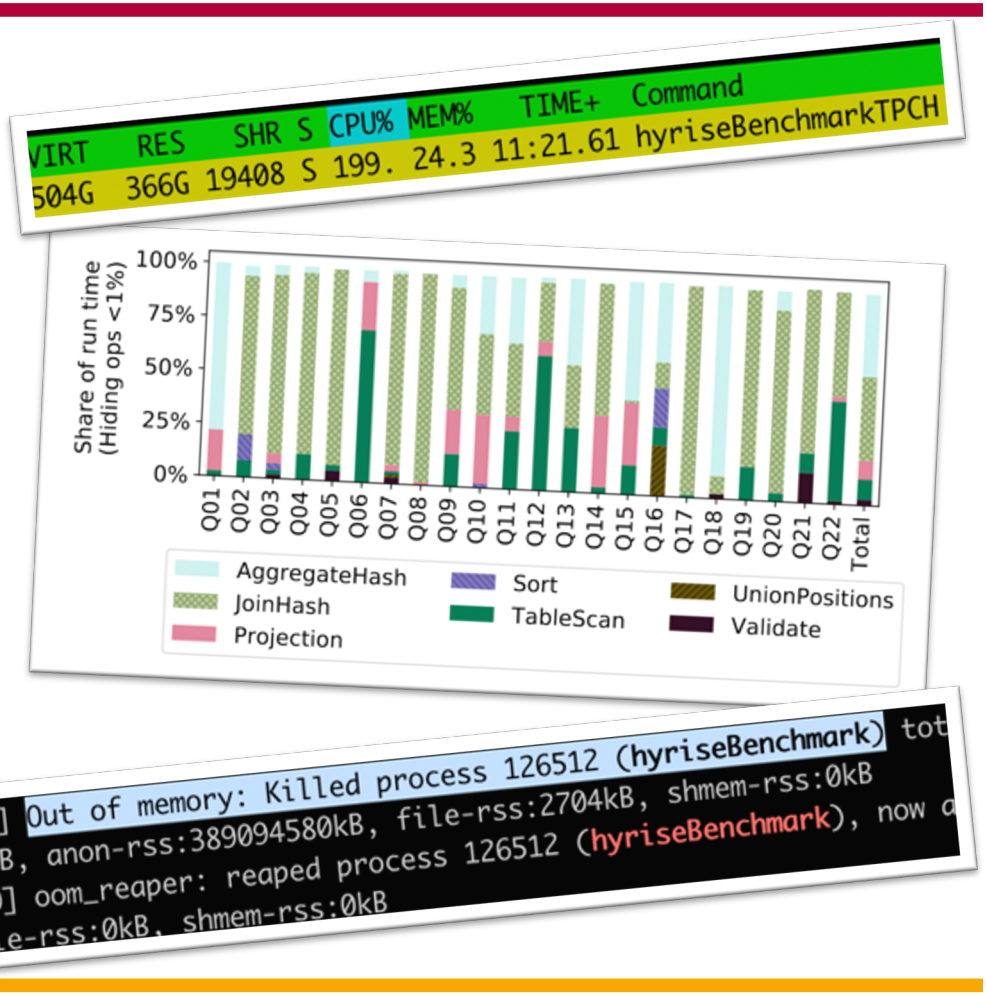
- But what is the cost of this compared to accessing the data directly? Can it be reduced?



Suggested Reading: https://openproceedings.org/2019/conf/edbt/EDBT19_paper_308.pdf

Understanding Memory Consumption with Tracking Allocators

- Where does our memory go?
- Profilers tell us which code allocates memory, but not what it belongs to
- For tables, memory consumption is somewhat easy to estimate
- Joins sometimes allocate vast amounts of memory, but it is hard to tell which join is at fault
- Can we use Polymorphic Memory Resources to track these allocations and how much does it cost?
- How does memory utilization differ in benchmarks?
- Can we use this information to identify a data object by its address?



Suggested Watch: https://www.youtube.com/watch?v=0MdSJsCTRkY

diPs: data-induced Predicates

	Movie		
	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title FROM Movie WHERE ProductionYear = 1989
```

	Movie		
	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title FROM Movie WHERE ProductionYear = 1989
```

	Movie		
	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title, Name FROM Movie, Person
    WHERE Movie.ProductionYear = 1989 AND Movie.ID = Person.MovieId
```

	Person			
	ID	MovieID	Name	Role
Min - Max	1 – 47	1 – 2	Arnol… – Kim…	Doug… – Vicki…
	1	1	Jack Nicholson	The Joker
	47	2	Arnold Schwa…	Douglas Quaid
	15	1	Kim Basinger	Vicki Vale
			• • •	
Min - Max	5 – 47	17 – 18	Arnol… – Lin…	Dete… – Miss…
	5	17	Bruce Willis	John McClane
	47	18	Arnold Schwa…	Detective John…
	34	18	Linda Hunt	Miss Schlowski

	Movie		
	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title, Name FROM Movie, Person
    WHERE Movie.ProductionYear = 1989 AND Movie.ID = Person.MovieId
```

Person

	ID	MovieID	Name	Role
Min - Max	1 – 47	1 – 2	Arnol… – Kim…	Doug… – Vicki…
	1	1	Jack Nicholson	The Joker
	47	2	Arnold Schwa…	Douglas Quaid
	15	1	Kim Basinger	Vicki Vale
	• • •			
Min - Max	5 – 47	17 – 18	Arnol… – Lin…	Dete… – Miss…
	5	17	Bruce Willis	John McClane
	47	18	Arnold Schwa…	Detective John…
	34	18	Linda Hunt	Miss Schlowski

Movie

	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title, Name FROM Movie, Person
    WHERE Movie.ProductionYear = 1989 AND Movie.ID = Person.MovieId
    AND Person.MovieID BETWEEN 1 AND 7
```

Person

	ID	MovieID	Name	Role
Min - Max	1 – 47	1 – 2	Arnol… – Kim…	Doug… – Vicki…
	1	1	Jack Nicholson	The Joker
	47	2	Arnold Schwa…	Douglas Quaid
	15	1	Kim Basinger	Vicki Vale
	• • •			
Min - Max	5 – 47	17 – 18	Arnol… – Lin…	Dete… – Miss…
	5	17	Bruce Willis	John McClane
	47	18	Arnold Schwa…	Detective John…
	34	18	Linda Hunt	Miss Schlowski

Movie

	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

diPs: data-induced Predicates

```
SELECT Title, Name FROM Movie, Person
    WHERE Movie.ProductionYear = 1989 AND Movie.ID = Person.MovieId
    AND Person.MovieID BETWEEN 1 AND 7
```

Person

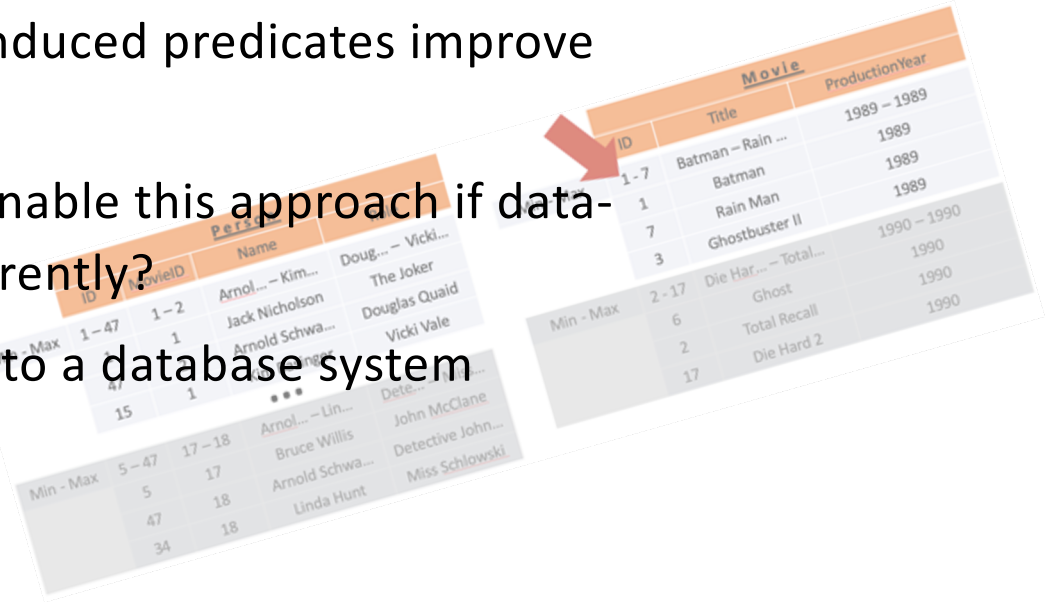
	ID	MovieID	Name	Role
Min - Max	1 – 47	1 – 2	Arnol… – Kim…	Doug… – Vicki…
	1	1	Jack Nicholson	The Joker
	47	2	Arnold Schwa…	Douglas Quaid
	1	1	Kim Basinger	Vicki Vale
			…	
Min - Max	5 – 47	17 – 18	Arnol… – Lin…	Dete… – Miss…
	5	17	Bruce Willis	John McClane
	47	18	Arnold Schwa…	Detective John…
	34	18	Linda Hunt	Miss Schlowski

Movie

	ID	Title	ProductionYear
Min - Max	1 - 7	Batman – Rain …	1989 – 1989
	1	Batman	1989
	7	Rain Man	1989
	3	Ghostbuster II	1989
Min - Max	2 - 17	Die Har… – Total…	1990 – 1990
	6	Ghost	1990
	2	Total Recall	1990
	17	Die Hard 2	1990

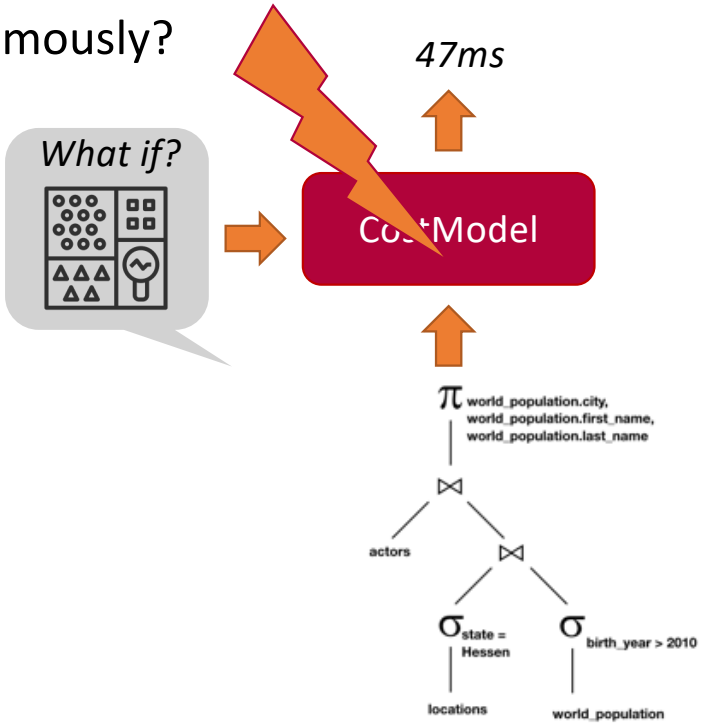
diPs: data-induced Predicates

- diPs' main idea: use data properties of one table to avoid unnecessary work when joining other tables
- Research questions:
  1. **Performance**: To which extent can data-induced predicates improve query performance in practice?
  2. **Transactional correctness:** How can we enable this approach if data-modifying operations are handled concurrently?
  3. **Integration:** How can we integrate diPs into a database system without adding too much complexity?



What if?-Optimization for Autonomous Clustering

- Vision: Database systems configure their physical design autonomously
  - Examples: clustering dimensions, indexes, compression schemes autonomously
- How do database systems approach the physical design autonomously?
  1. Identify sensible candidates
  2. Estimate performance impact of candidates
  3. Pick the best candidate(s)



Suggested Readings: https://dl.acm.org/doi/pdf/10.1145/276304.276337 (What-If Optimization)
http://www.vldb.org/conf/2004/IND5P5.PDF (Clustering)

Timeline


Weekly meetings with advisors
02 Nov 2020
Kickoff
08 Nov 2020 EOD
Submit your group and topic preferences
09 Nov 2020 15:15 (or individual agreement)
First meeting with advisor
20 Nov 2020
Formal deadline to register – Please confirm your participation earlier
08 Feb 2021
Final presentations – 15 min + 5 min Q&A
March 2021
Submission of written report

Corona

- With the current developments, we plan for the course to be largely online
- You are free to make individual arrangements with your advisor

- Should something unexpected come up on your side, please let us know ASAP so that we can find a solution

Administration

- Specialization areas:
  - ITSE: BPET, OSIS, SAMT, ITSE-Analyse, ITSE-Maintenance
  - DATA: Scalable Data Systems
- Maximum of 16 students (FCFS)
- Official deadline to register is 20 Nov, but we would like to fix groups and topic asap
- Grading
  - 50% project result and presentation
  - 40% scientific report (4-8 pages IEEE, depending on group size)
  - 10% personal engagement

Bringing groups and topics together

- You are welcome to hang out in this BBB room after our talk to figure out groups
- If you have found a group and a topic, please mail markus.dreseler
  - Include three (or more) topic preferences
  - The assignment algorithm is strategy-proof ;)
- If you have any questions or are still looking for a group partner, please mail us, too

Background reading: https://en.wikipedia.org/wiki/Strategyproofness
https://en.wikipedia.org/wiki/Gale%E2%80%93Shapley_algorithm