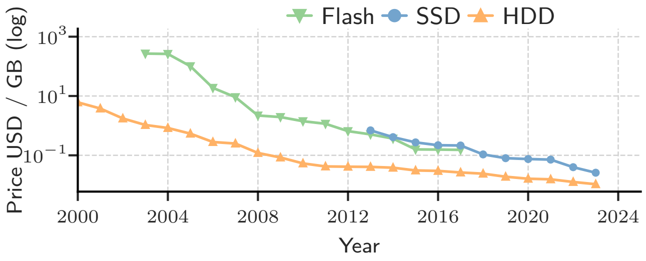
Adaptive Optimizations for Databases

Christoph Anneser

Technical University of Munich

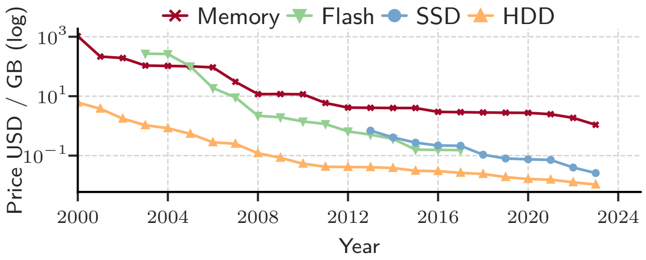
21.05.2024

Introduction



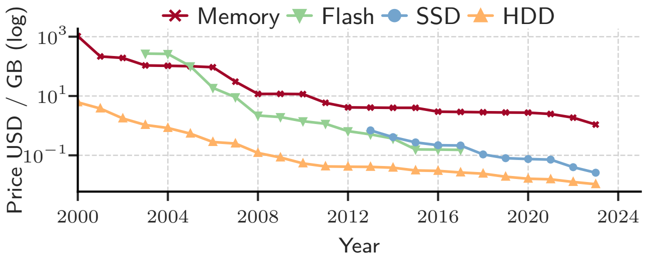
https://ourworldindata.org/

Introduction

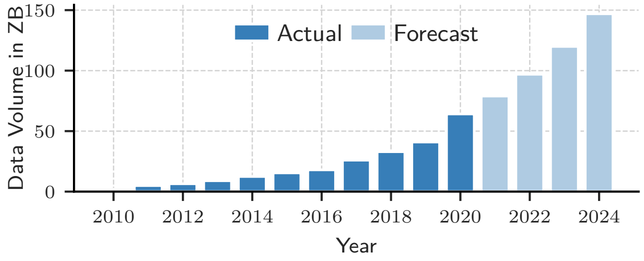


https://ourworldindata.org/

Introduction

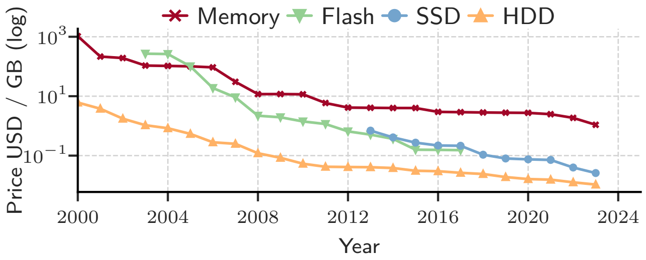


https://ourworldindata.org/

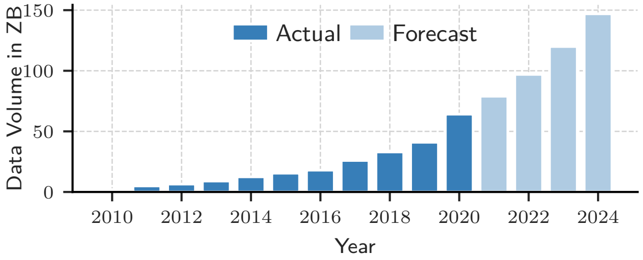


www.statista.com/statistics/871513/worldwide-data-created/

Introduction

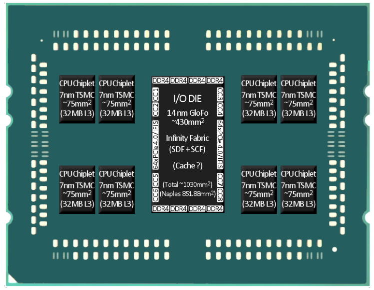


https://ourworldindata.org/

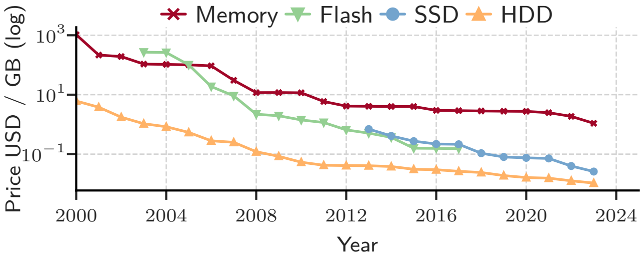


www.statista.com/statistics/871513/worldwide-data-created/

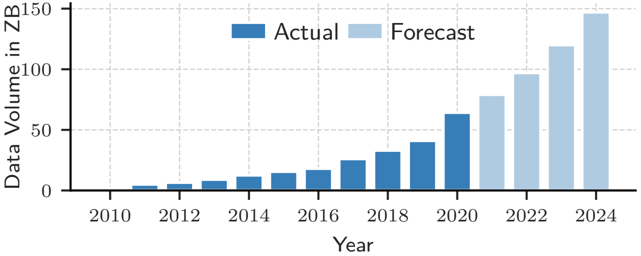
AMD EPYC Rome



Introduction

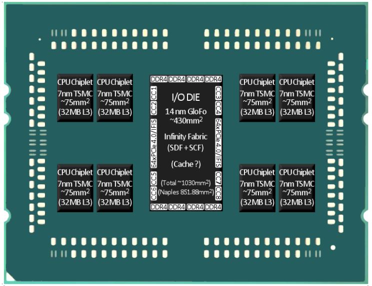


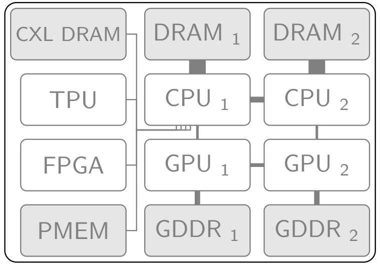
https://ourworldindata.org/



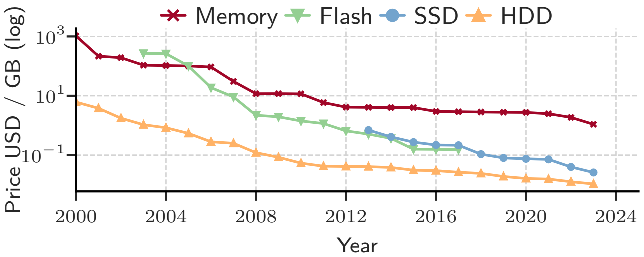
www.statista.com/statistics/871513/worldwide-data-created/

AMD EPYC Rome

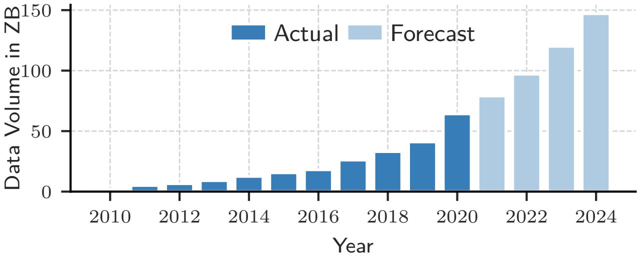




Introduction

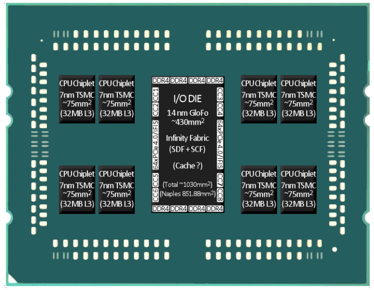


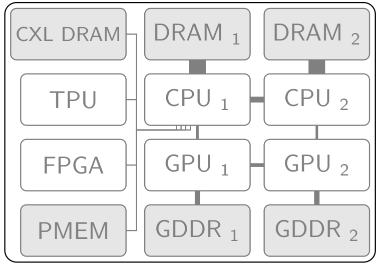
https://ourworldindata.org/



www.statista.com/statistics/871513/worldwide-data-created/

AMD EPYC Rome





Optimize Data Management Systems for Resource Efficiency Sustainably

Static and Adaptive Optimizations

Static Optimizations

Adaptive Optimizations

Static and Adaptive Optimizations

Static Optimizations

$\Rightarrow$ At development time

$\Rightarrow$ Independent of input data

$\Rightarrow$ Theoretical runtime

Adaptive Optimizations

Static and Adaptive Optimizations

Static Optimizations

⇒ At development time

⇒ Independent of input data

⇒ Theoretical runtime

Adaptive Optimizations

⇒ At execution time

⇒ Data distribution & patterns

⇒ Hardware avail. & utilization

Static and Adaptive Optimizations

Static Optimizations

⇒ At development time

⇒ Independent of input data

⇒ Theoretical runtime

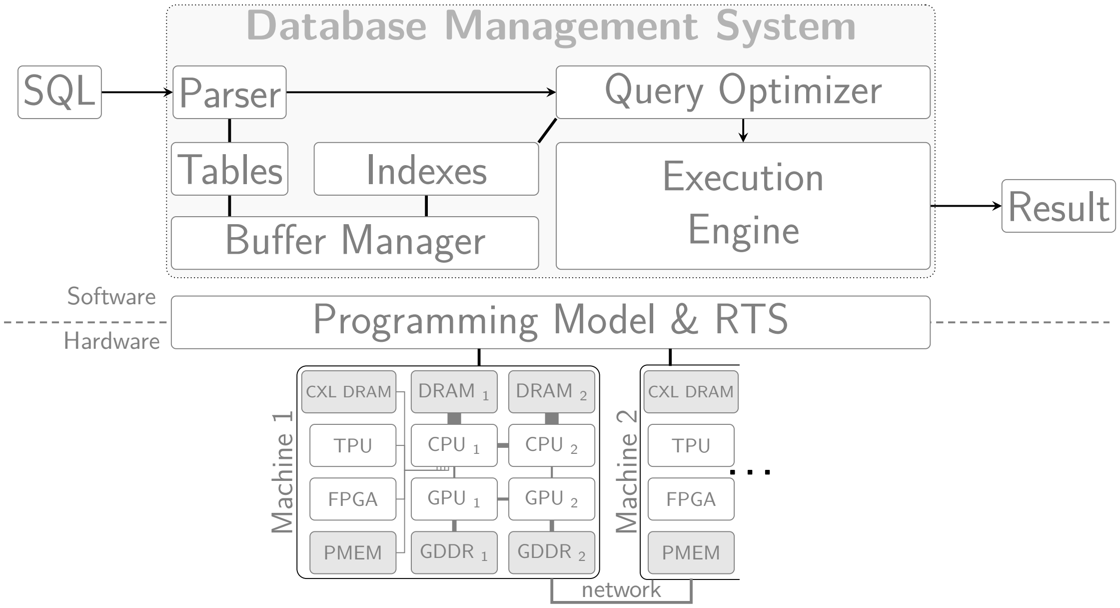
Adaptive Optimizations

⇒ At execution time

⇒ Data distribution & patterns

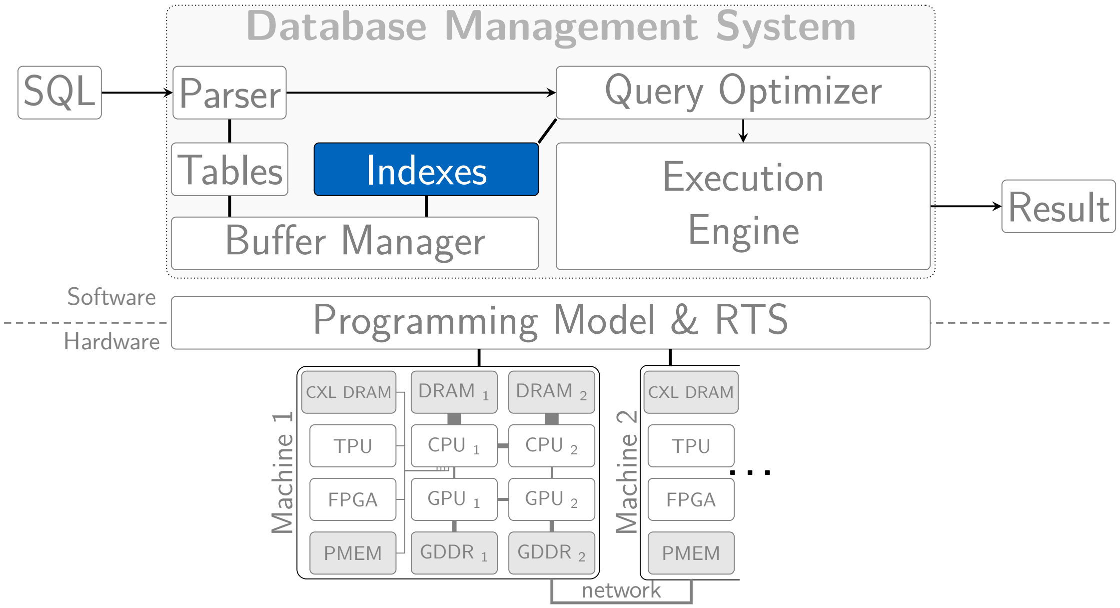
⇒ Hardware avail. & utilization

Overview



Database Management System
SQL
Parser
Query Optimizer
Tables
Indexes
Execution Engine
Buffer Manager
Result
Software
Hardware
Programming Model & RTS
Machine 1
CXL DRAM
DRAM 1
DRAM 2
TPU
CPU 1
CPU 2
FPGA
GPU 1
GPU 2
PMEM
GDDR 1
GDDR 2
Machine 2
CXL DRAM
TPU
FPGA
PMEM
...
network

P1: Adaptive Hybrid Indexes



Adaptive Hybrid Indexes

Christoph Anneser
Technical University of Munich
anneser@in.tum.de

Andreas Kipf
Massachusetts Institute of Technology
kipf@mit.edu

Huanchen Zhang
Tsinghua University
huanchen@tsinghua.edu.cn

Thomas Neumann
Technical University of Munich
neumann@in.tum.de

Alfons Kemper
Technical University of Munich
kemper@in.tum.de

ABSTRACT

While index structures are crucial components in high-performance query processing systems, they occupy a large fraction of the available memory. Recently-proposed compact indexes reduce this space overhead and thus speed up queries by allowing the database to keep larger working sets in memory. These compact indexes, however, are slower than performance-optimized in-memory indexes because they adopt encodings that trade performance for memory efficiency. Applying different encodings within a single index might allow optimizing both dimensions at the same time – however, it is not clear which encodings should be applied to which index parts at *build-time*.

To take advantage of multiple encodings in *one* index structure, we present a new framework forming the basis of *workload-adaptive hybrid indexes* which moves encoding decisions to *run-time* instead. By sampling incoming queries adaptively, it tracks accesses to index parts and keeps fine-grained statistics which are used for space- and performance-optimized encoding migrations. We evaluated our framework using B+-trees and tries, and examine the adaptation process and space/performance trade-off for real-world and synthetic workloads. For skewed workloads, our framework can reduce the space by up to 82% while retaining more than 90% of the original performance.

CCS CONCEPTS

• **Information systems** → **Data access methods**; **Data layout**.

KEYWORDS

Space-efficient Index; Adaptive Index; Hybrid Index

ACM Reference Format:
Christoph Anneser, Andreas Kipf, Huanchen Zhang, Thomas Neumann, and Alfons Kemper. 2022. Adaptive Hybrid Indexes. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3526121

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526121

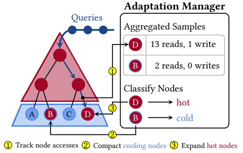


Figure 1: Our sampling-based workload adaptation supports hybrid index structures in choosing the most suitable encoding for each part based on fine-grained access statistics at run-time. It supports user-defined settings such as an upper memory budget and it keeps sampling-related overhead limited by following an adaptive cost-optimized approach.

1 INTRODUCTION

Back in 2006, Jim Gray stated that memory is the new disk and disk is the new tape [5]. This also applies to modern database systems that store the entire data in random access memory (RAM) to allow real-time analyses for trading companies and financial services, for example. They need to process large datasets efficiently to react to new developments and updates within a few milliseconds.

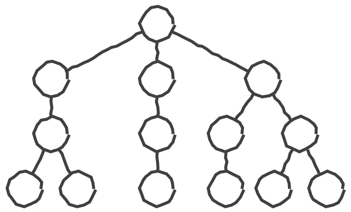
While the DRAM-prices have been stable during the last six to seven years, the data collected by sensors, smartphones, social media platforms, IoT-devices, and digital market-places *increases at a high rate* resulting in data overflows [54], and storing all data in memory becomes infeasible in many cases. However, as in-memory database systems become more and more popular for performance-critical businesses, AWS offers RAM instances that are optimized for in-memory database systems [1]. These instances are equipped with in-memory capacities of up to 24 TB, but the hourly cost of such an instance is more than $120.

To achieve high-performance query-processing for real-time analyses, index structures such as B-trees, tries, and hash tables are widely used by DBMSs. Because there might be multiple indexes per table, especially in OLTP DBMSs, the storage overhead for indexes can be significant. In many cases, more than half of the available memory of a DBMS can be attributed to index structures [54].

@SIGMOD'22

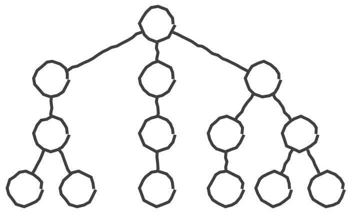
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing



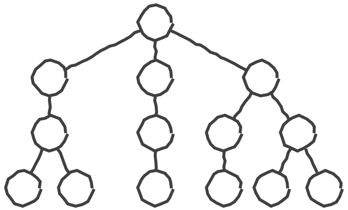
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance
    and not for memory efficiency



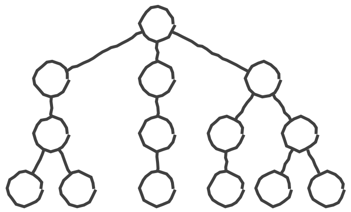
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS



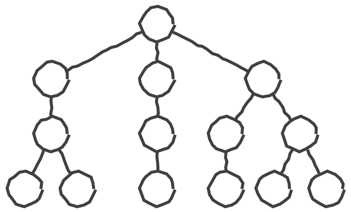
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance
    and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

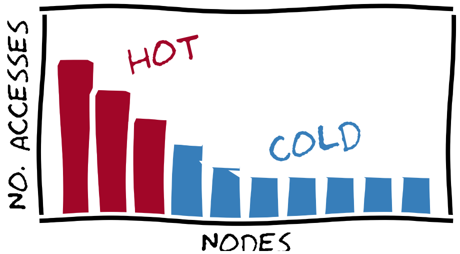


Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

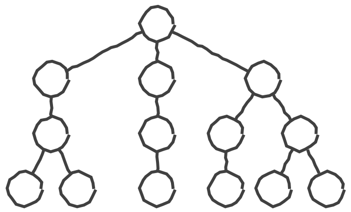


- Real-world workloads are **skewed**

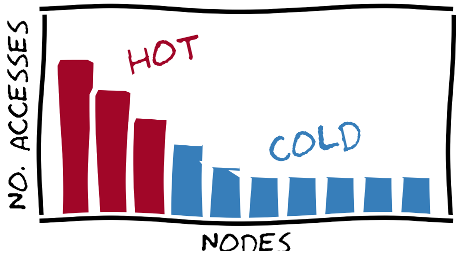


Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

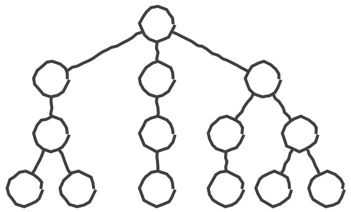


- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the workload

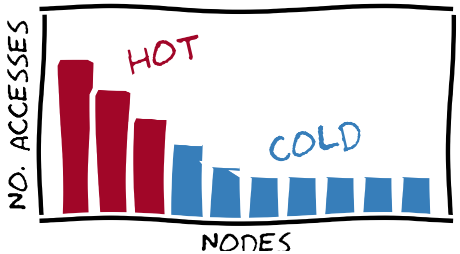


Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!



Optimize at run-time

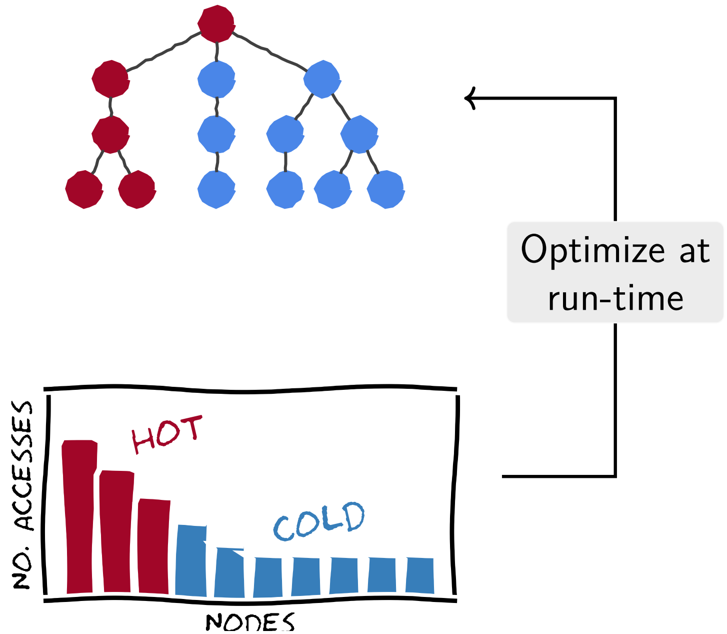


- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the workload

Adaptive Hybrid Indexes – Problem

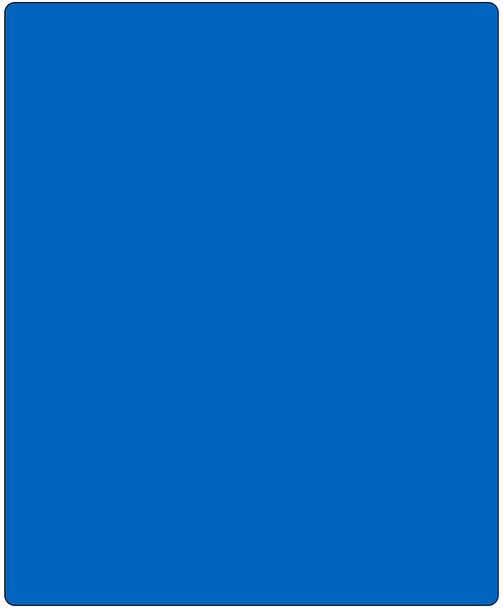
- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the workload

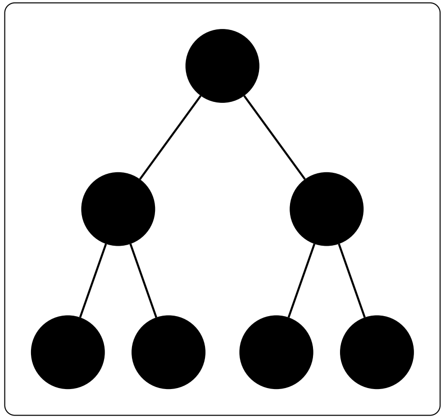


Adaptive Hybrid Indexes – Solution

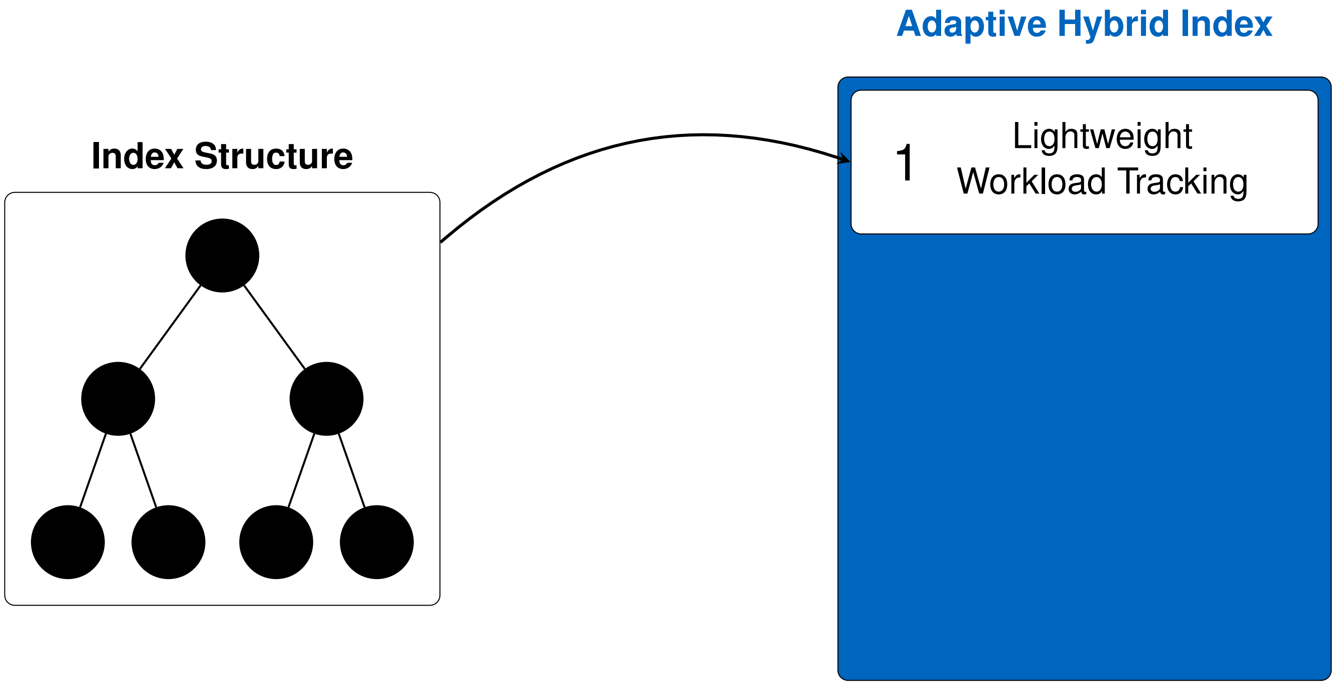
Adaptive Hybrid Index



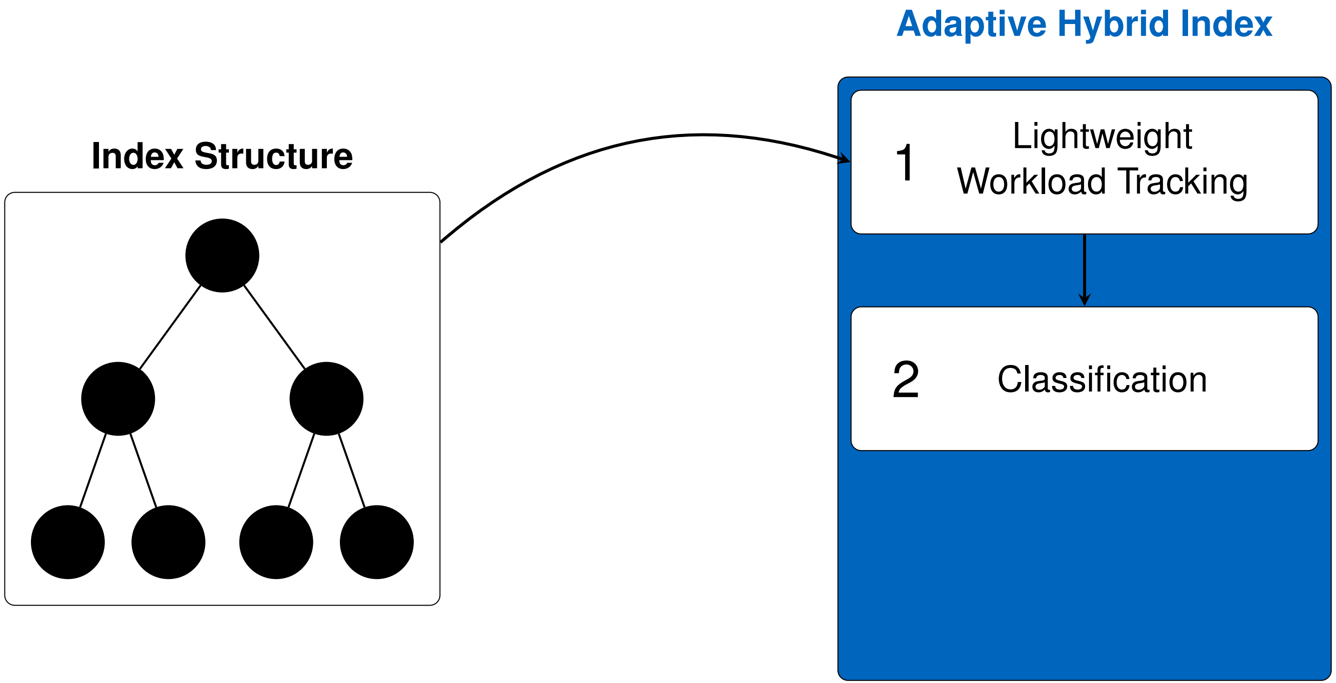
Index Structure



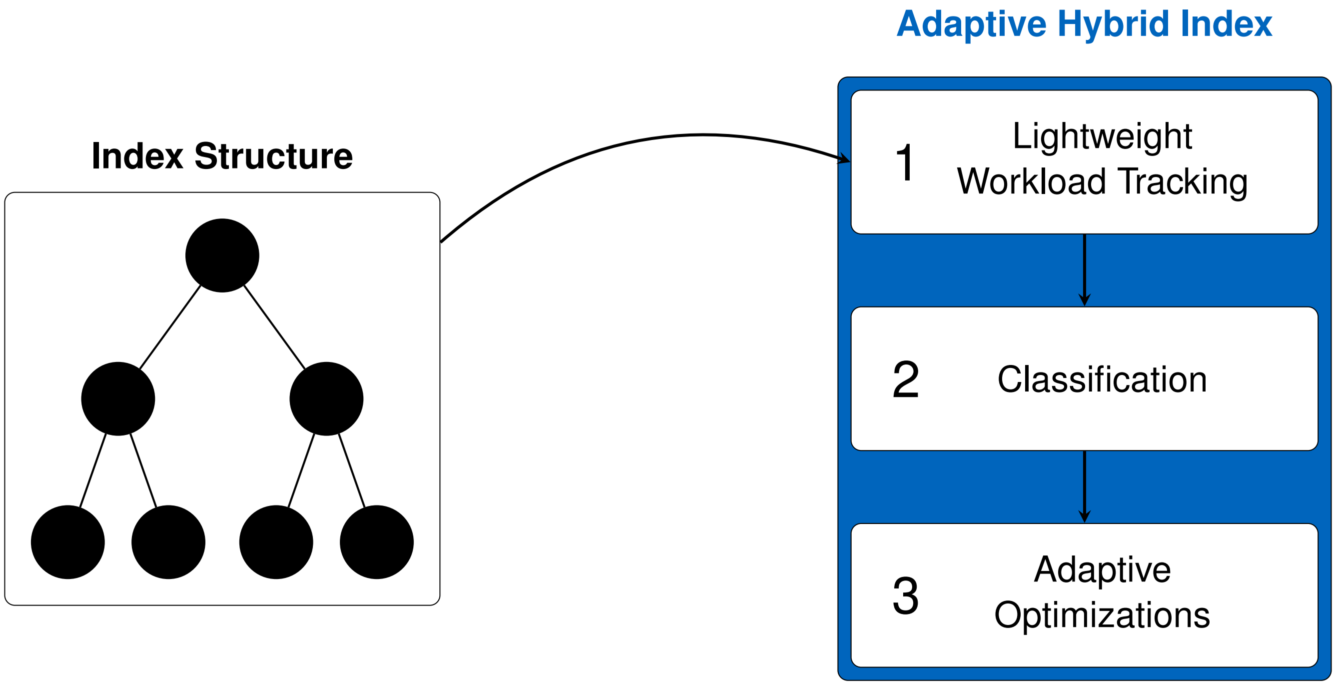
Adaptive Hybrid Indexes – Solution



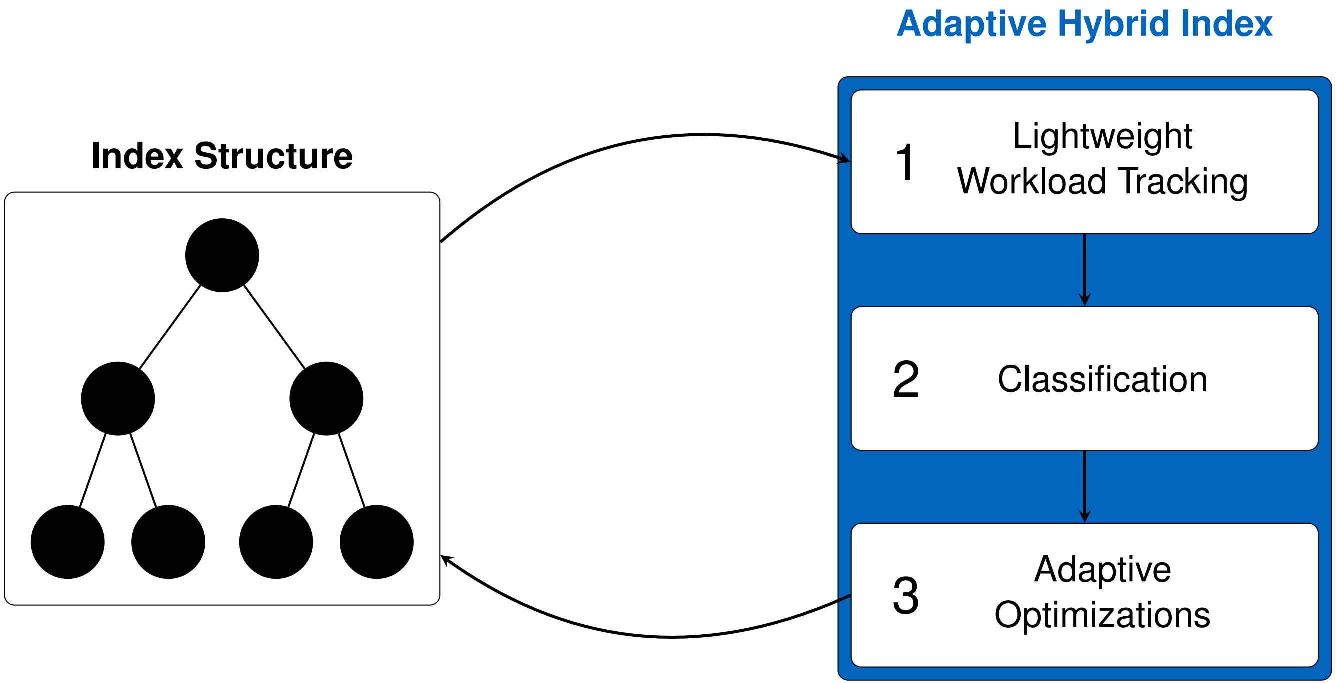
Adaptive Hybrid Indexes – Solution



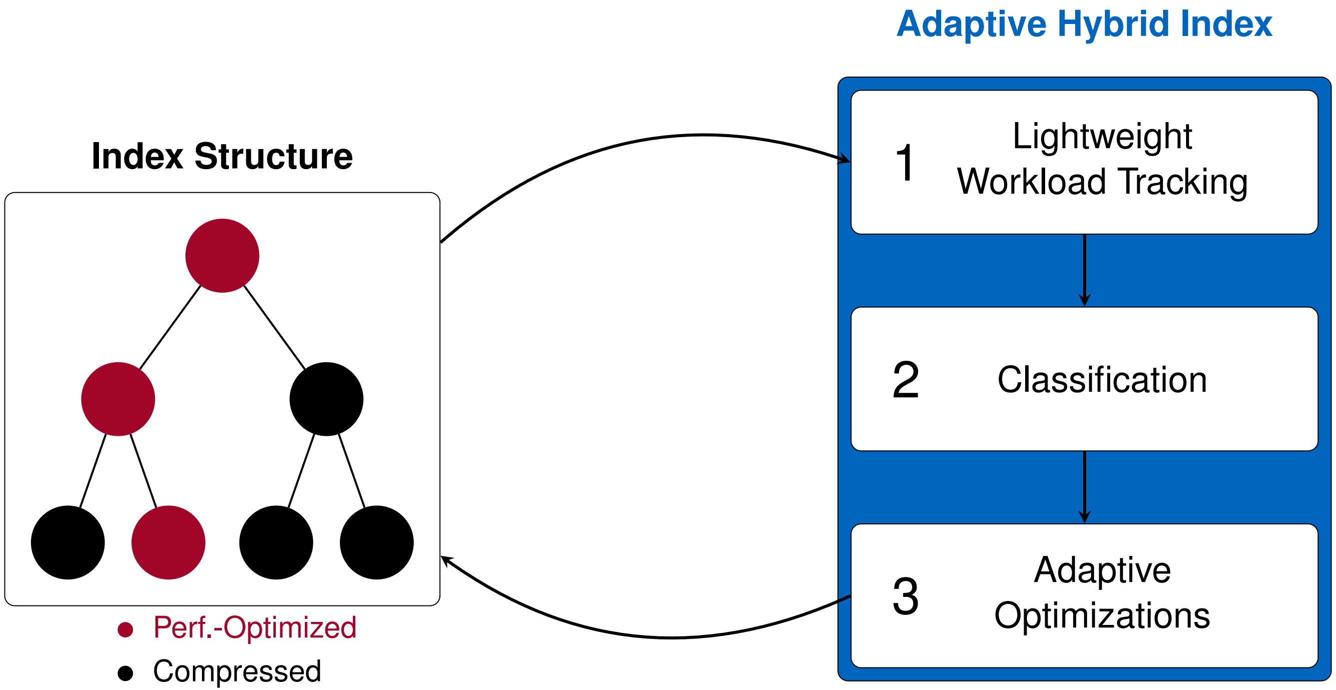
Adaptive Hybrid Indexes – Solution



Adaptive Hybrid Indexes – Solution



Adaptive Hybrid Indexes – Solution



Adaptive Hybrid Trie

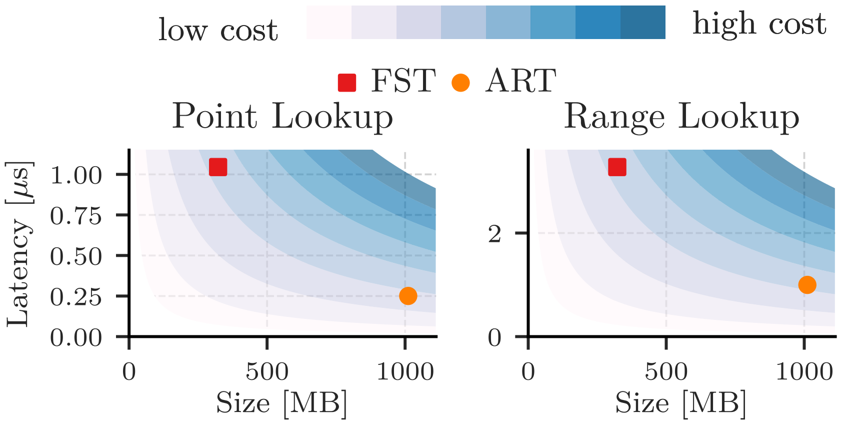


Figure: Query latency and index size of ART and FST

Experiment Setup:

- Dataset: 33M unique email adresses (host-reversed order, e.g. com.foo@<username>)
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- Compiler: GCC 9.3.0 with flags `O3` and `march=native`

Adaptive Hybrid Trie

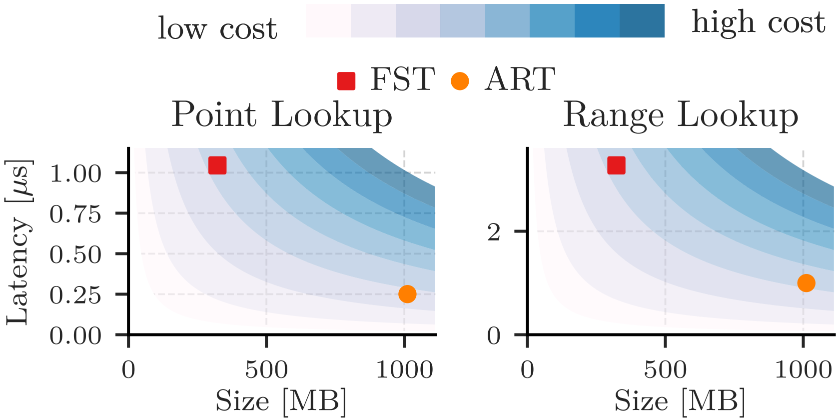


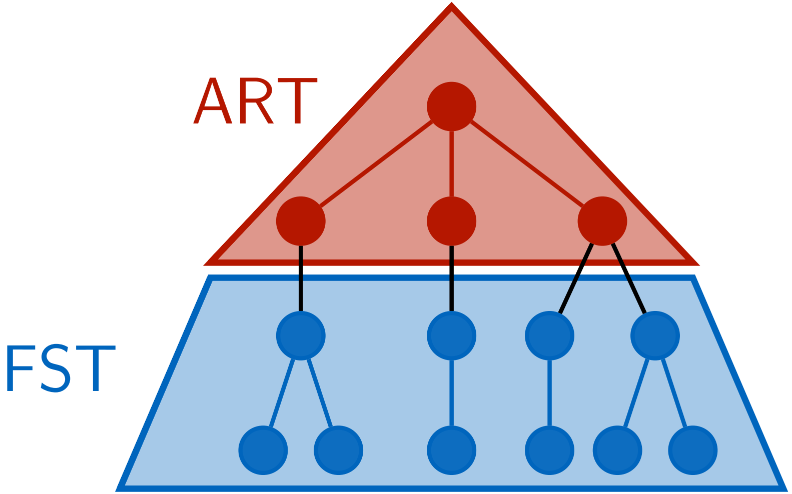
Figure: Query latency and index size of ART and FST

Adaptive Hybrid Trie is a level-wise combination of ART and FST!

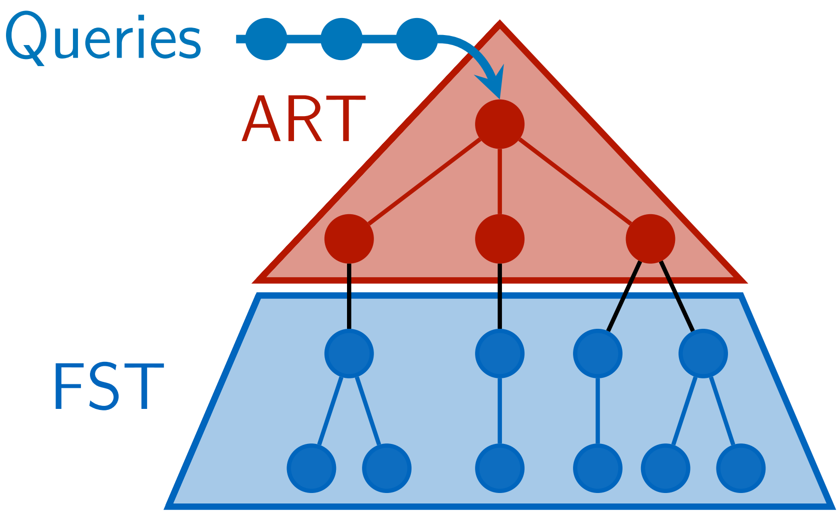
Experiment Setup:

- Dataset: 33M unique email adresses (host-reversed order, e.g. com.foo@<username>)
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- Compiler: GCC 9.3.0 with flags `O3` and `march=native`

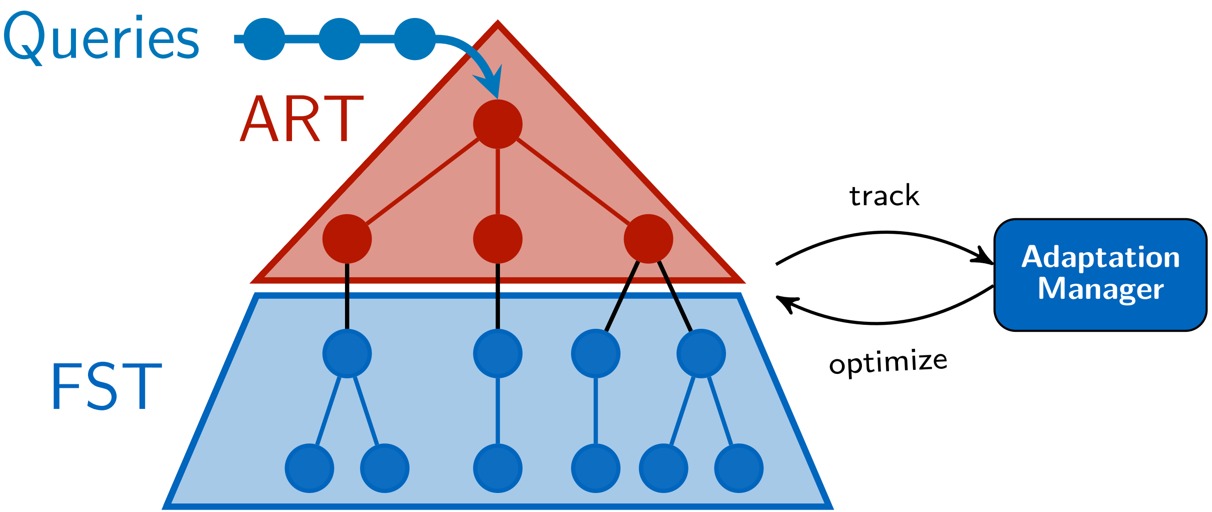
Adaptive Hybrid Trie



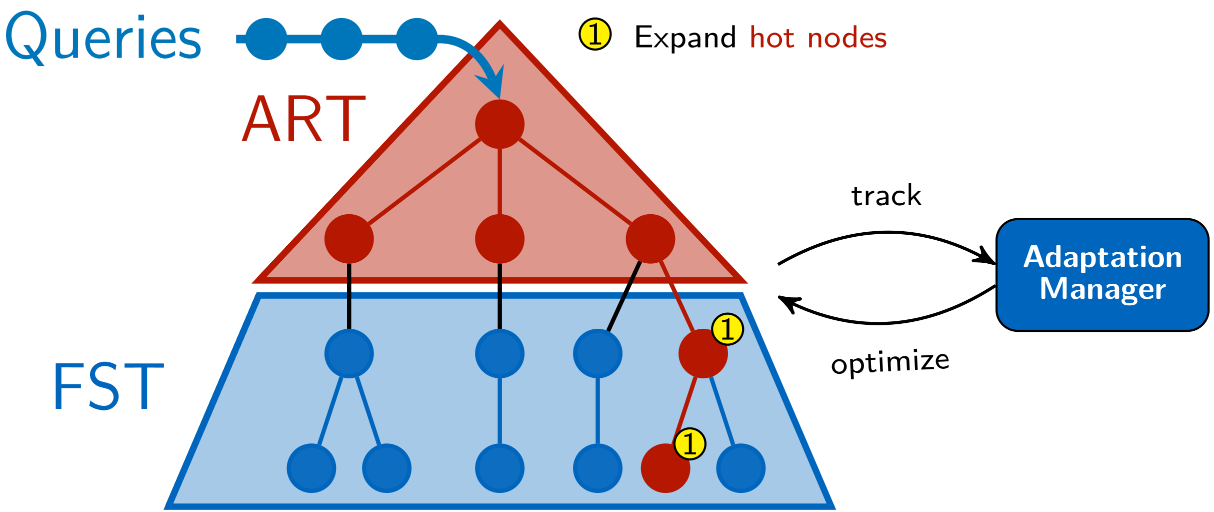
Adaptive Hybrid Trie



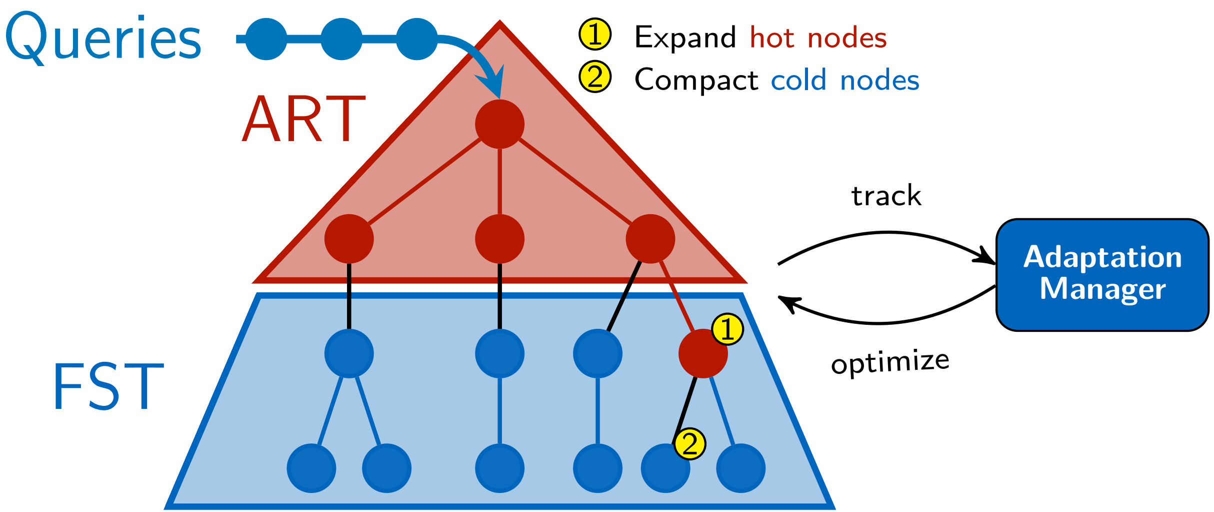
Adaptive Hybrid Trie



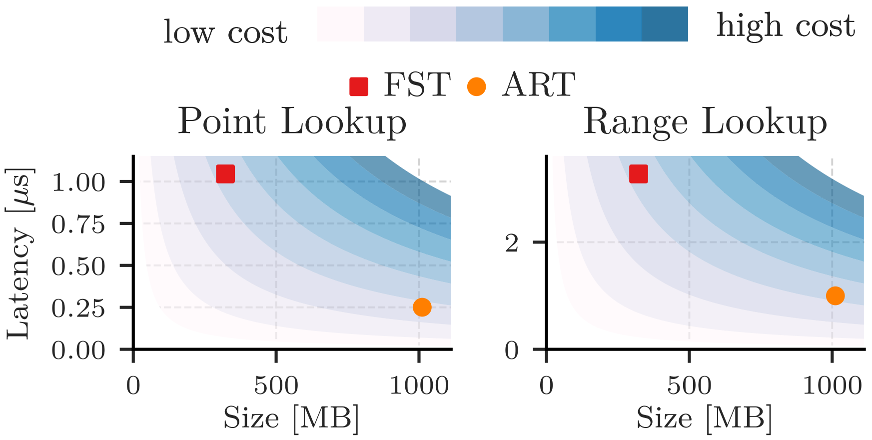
Adaptive Hybrid Trie



Adaptive Hybrid Trie



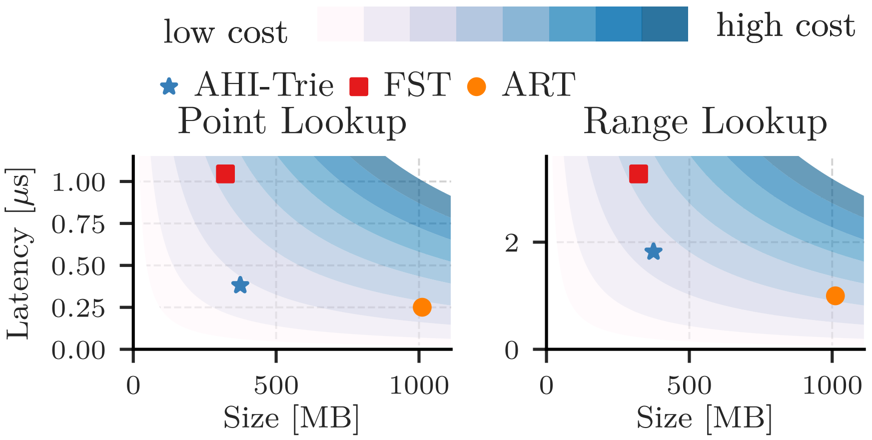
Evaluation: Hybrid Trie – Space & Performance



Experiment Setup:

- Dataset: 33M unique email adresses (host-reversed order, e.g. com.foo@<username>)
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- Compiler: GCC 9.3.0 with flags `O3` and `march=native`

Evaluation: Hybrid Trie – Space & Performance



Conclusions:

For point lookups, Hybrid Trie

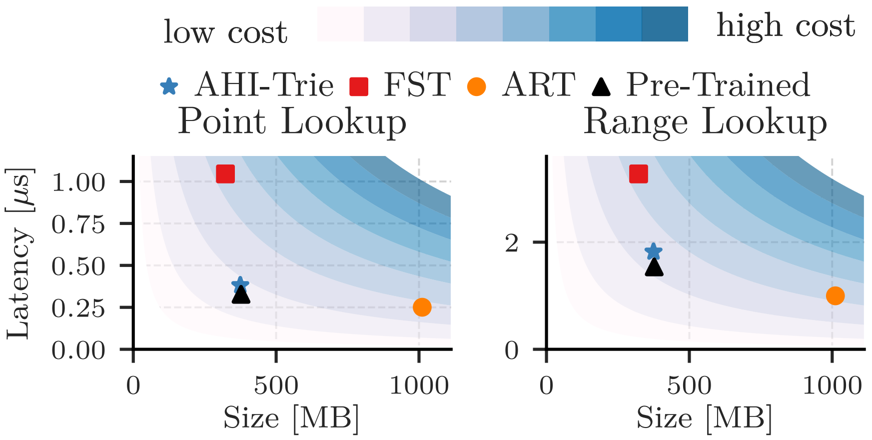
⇒ reduces index size by 63% comp. to ART

⇒ improves performance by 2.7x comp. to FST

Experiment Setup:

- Dataset: 33M unique email adresses (host-reversed order, e.g. com.foo@<username>)
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- Compiler: GCC 9.3.0 with flags `O3` and `march=native`

Evaluation: Hybrid Trie – Space & Performance



Conclusions:

For point lookups, Hybrid Trie

⇒ reduces index size by 63% comp. to ART

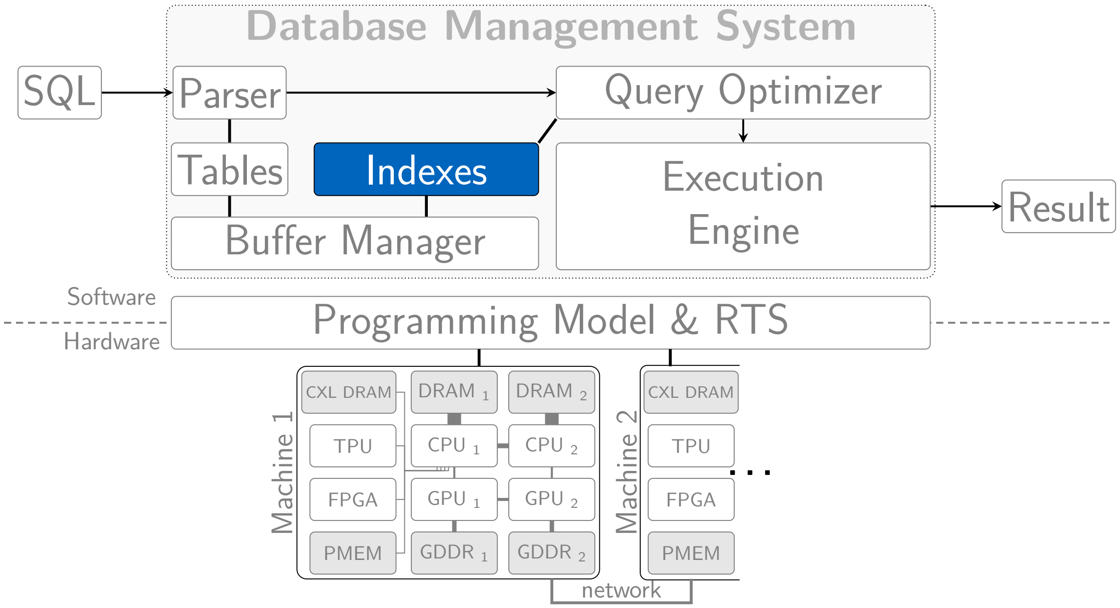
⇒ improves performance by 2.7x comp. to FST

The Pre-Trained Hybrid Trie does not include tracking-related overhead

Experiment Setup:

- Dataset: 33M unique email adresses (host-reversed order, e.g. com.foo@<username>)
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- Compiler: GCC 9.3.0 with flags `O3` and `march=native`

P1: Adaptive Hybrid Indexes



Adaptive Hybrid Indexes

Christoph Anneser, Technical University of Munich, anneser@in.tum.de
Andreas Kipf, Massachusetts Institute of Technology, kipf@mit.edu
Huanchen Zhang, Tsinghua University, huanchen@tsinghua.edu.cn
Thomas Neumann, Technical University of Munich, neumann@in.tum.de
Alfons Kemper, Technical University of Munich, kemper@in.tum.de

ABSTRACT

While index structures are crucial components in high-performance query processing systems, they occupy a large fraction of the available memory. Recently-proposed compact indexes reduce this space overhead and thus speed up queries by allowing the database to keep larger working sets in memory. These compact indexes, however, are slower than performance-optimized in-memory indexes because they adopt encodings that trade performance for memory efficiency. Applying different encodings within a single index might allow optimizing both dimensions at the same time – however, it is not clear which encodings should be applied to which index parts at *build-time*.

To take advantage of multiple encodings in *one* index structure, we present a new framework forming the basis of *workload-adaptive hybrid indexes* which moves encoding decisions to *run-time* instead. By sampling incoming queries adaptively, it tracks accesses to index parts and keeps fine-grained statistics which are used for space- and performance-optimized encoding migrations. We evaluated our framework using B+-trees and tries, and examine the adaptation process and space/performance trade-off for real-world and synthetic workloads. For skewed workloads, our framework can reduce the space by up to 82% while retaining more than 90% of the original performance.

CCS CONCEPTS

• **Information systems** → **Data access methods**; **Data layout**.

KEYWORDS

Space-efficient Index; Adaptive Index; Hybrid Index

ACM Reference Format:
Christoph Anneser, Andreas Kipf, Huanchen Zhang, Thomas Neumann, and Alfons Kemper. 2022. Adaptive Hybrid Indexes. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3526121

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526121

Adaptation Manager
Queries
Aggregated Samples
D 13 reads, 1 write
B 2 reads, 0 writes
Classify Nodes
D → hot
B → cold
A B C D
① Track node accesses ② Compact cooling nodes ③ Expand hot nodes

Figure 1: Our sampling-based workload adaptation supports hybrid index structures in choosing the most suitable encoding for each part based on fine-grained access statistics at run-time. It supports user-defined settings such as an upper memory budget and it keeps sampling-related overhead limited by following an adaptive cost-optimized approach.

1 INTRODUCTION

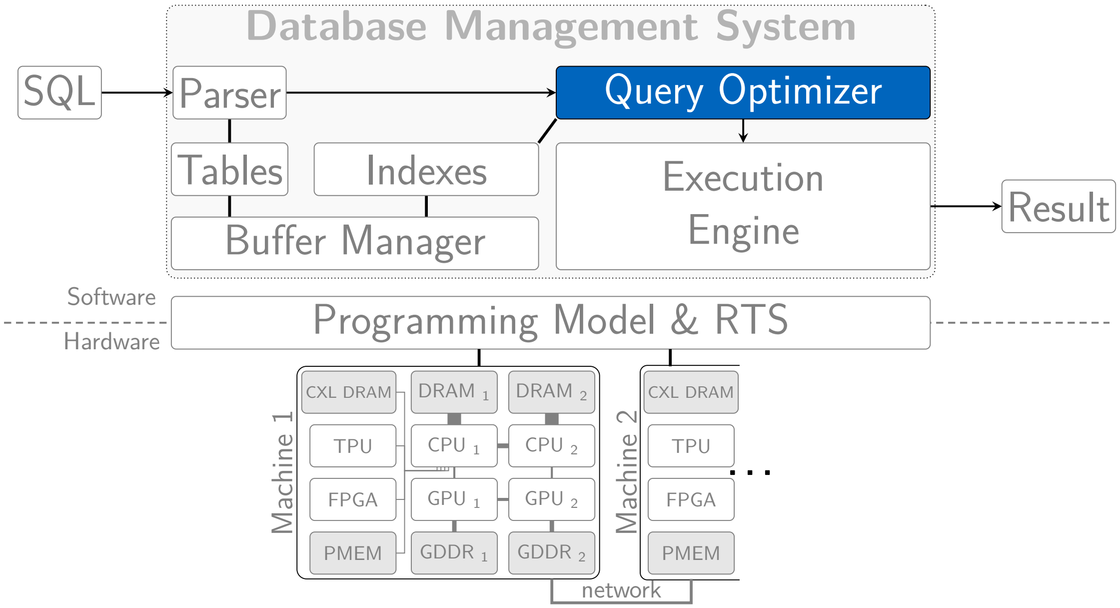
Back in 2006, Jim Gray stated that memory is the new disk and disk is the new tape [5]. This also applies to modern database systems that store the entire data in random access memory (RAM) to allow real-time analyses for trading companies and financial services, for example. They need to process large datasets efficiently to react to new developments and updates within a few milliseconds.

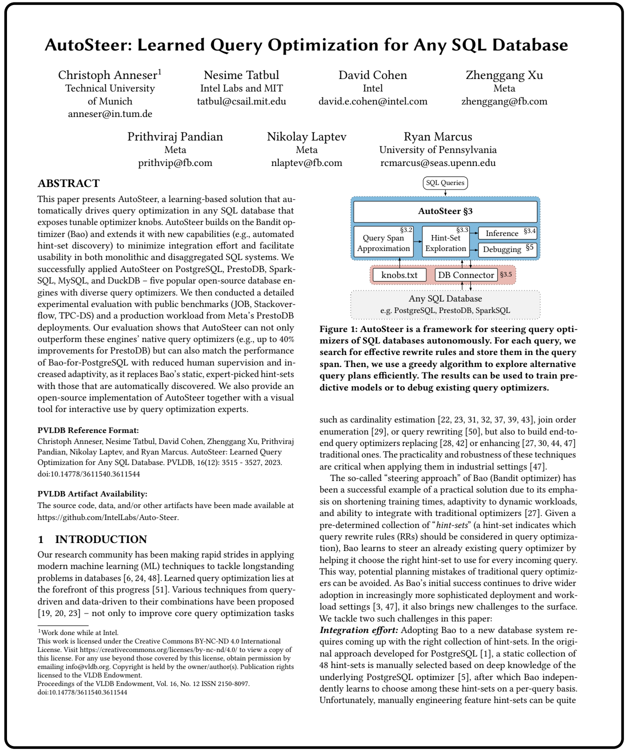
While the DRAM-prices have been stable during the last six to seven years, the data collected by sensors, smartphones, social media platforms, IoT-devices, and digital market-places *increases at a high rate* resulting in data overflows [54], and storing all data in memory becomes infeasible in many cases. However, as in-memory database systems become more and more popular for performance-critical businesses, AWS offers RAM instances that are optimized for in-memory database systems [1]. These instances are equipped with in-memory capacities of up to 24 TB, but the hourly cost of such an instance is more than $120.

To achieve high-performance query-processing for real-time analyses, index structures such as B-trees, tries, and hash tables are widely used by DBMSs. Because there might be multiple indexes per table, especially in OLTP DBMSs, the storage overhead for indexes can be significant. In many cases, more than half of the available memory of a DBMS can be attributed to index structures [54].

@SIGMOD'22

P2: AutoSteer





AutoSteer: Learned Query Optimization for Any SQL Database

Christoph Anneser[1]
Technical University of Munich
anneser@in.tum.de

Nesime Tatbul
Intel Labs and MIT
tatbul@csail.mit.edu

David Cohen
Intel
david.e.cohen@intel.com

Zhenggang Xu
Meta
zhenggang@fb.com

Prithviraj Pandian
Meta
prithvip@fb.com

Nikolay Laptev
Meta
nlaptev@fb.com

Ryan Marcus
University of Pennsylvania
rcmarcus@seas.upenn.edu

ABSTRACT

This paper presents AutoSteer, a learning-based solution that automatically drives query optimization in any SQL database that exposes tunable optimizer knobs. AutoSteer builds on the Bandit optimizer (Bao) and extends it with new capabilities (e.g., automated hint-set discovery) to minimize integration effort and facilitate usability in both monolithic and disaggregated SQL systems. We successfully applied AutoSteer on PostgreSQL, PrestoDB, SparkSQL, MySQL, and DuckDB – five popular open-source database engines with diverse query optimizers. We then conducted a detailed experimental evaluation with public benchmarks (JOB, Stackoverflow, TPC-DS) and a production workload from Meta's PrestoDB deployments. Our evaluation shows that AutoSteer can not only outperform these engines' native query optimizers (e.g., up to 40% improvements for PrestoDB) but can also match the performance of Bao-for-PostgreSQL with reduced human supervision and increased adaptivity, as it replaces Bao's static, expert-picked hint-sets with those that are automatically discovered. We also provide an open-source implementation of AutoSteer together with a visual tool for interactive use by query optimization experts.

PVLDB Reference Format:
Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. AutoSteer: Learned Query Optimization for Any SQL Database. PVLDB, 16(12): 3515 - 3527, 2023.
doi:10.14778/3611540.3611544

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/IntelLabs/Auto-Steer.

1 INTRODUCTION

Our research community has been making rapid strides in applying modern machine learning (ML) techniques to tackle longstanding problems in databases [6, 24, 48]. Learned query optimization lies at the forefront of this progress [51]. Various techniques from query-driven and data-driven to their combinations have been proposed [19, 20, 23] – not only to improve core query optimization tasks such as cardinality estimation [22, 23, 31, 32, 37, 39, 43], join order enumeration [29], or query rewriting [50], but also to build end-to-end query optimizers replacing [28, 42] or enhancing [27, 30, 44, 47] traditional ones. The practicality and robustness of these techniques are critical when applying them in industrial settings [47].

The so-called "steering approach" of Bao (Bandit optimizer) has been a successful example of a practical solution due to its emphasis on shortening training times, adaptivity to dynamic workloads, and ability to integrate with traditional optimizers [27]. Given a pre-determined collection of "*hint-sets*" (a hint-set indicates which query rewrite rules (RRs) should be considered in query optimization), Bao learns to steer an already existing query optimizer by helping it choose the right hint-set to use for every incoming query. This way, potential planning mistakes of traditional query optimizers can be avoided. As Bao's initial success continues to drive wider adoption in increasingly more sophisticated deployment and workload settings [3, 47], it also brings new challenges to the surface. We tackle two such challenges in this paper:

Integration effort: Adopting Bao to a new database system requires coming up with the right collection of hint-sets. In the original approach developed for PostgreSQL [1], a static collection of 48 hint-sets is manually selected based on deep knowledge of the underlying PostgreSQL optimizer [5], after which Bao independently learns to choose among these hint-sets on a per-query basis. Unfortunately, manually engineering feature hint-sets can be quite

Figure 1: AutoSteer is a framework for steering query optimizers of SQL databases autonomously. For each query, we search for effective rewrite rules and store them in the query span. Then, we use a greedy algorithm to explore alternative query plans efficiently. The results can be used to train predictive models or to debug existing query optimizers.

[1]Work done while at Intel.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611544

@VLDB'23

Background – Steering Query Optimizers

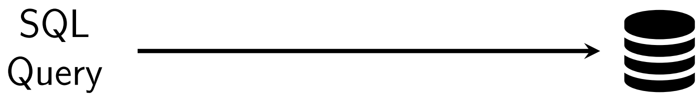
Many Database Management Systems expose tunable optimizer knobs.

- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization

Background – Steering Query Optimizers

Many Database Management Systems expose tunable optimizer knobs.

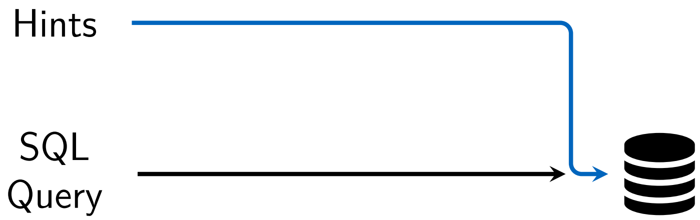
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization



Background – Steering Query Optimizers

Many Database Management Systems expose tunable optimizer knobs.

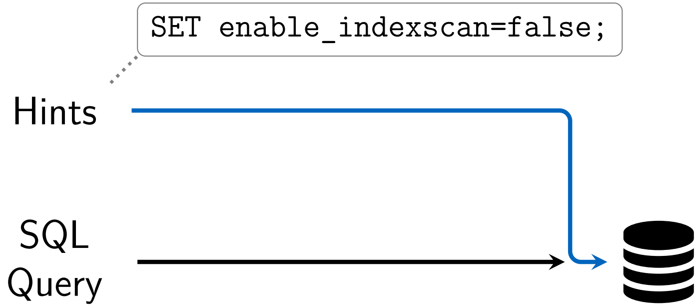
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization



Background – Steering Query Optimizers

Many Database Management Systems expose tunable optimizer knobs.

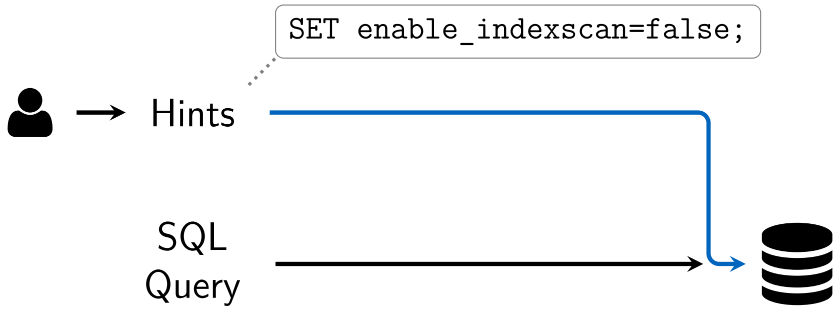
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization



Background – Steering Query Optimizers **Manually**

Many Database Management Systems expose tunable optimizer knobs.

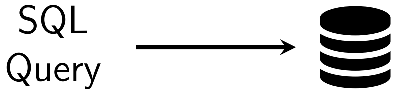
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

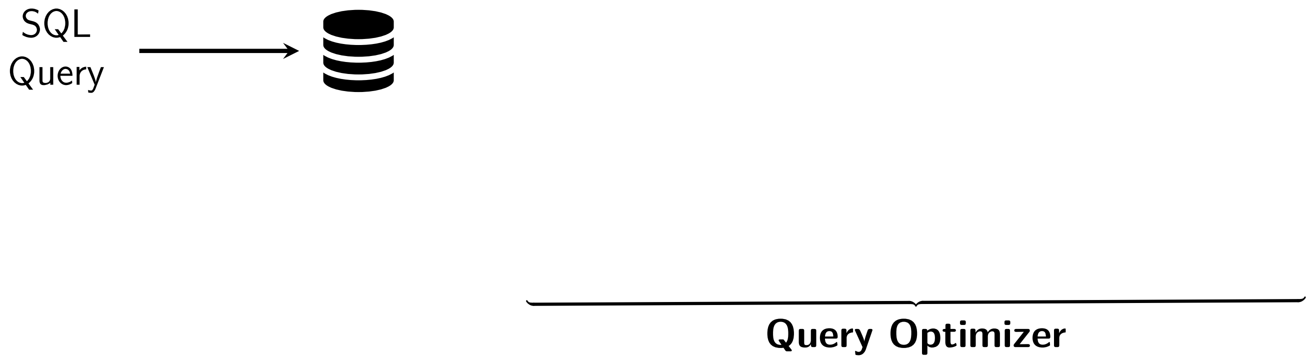
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

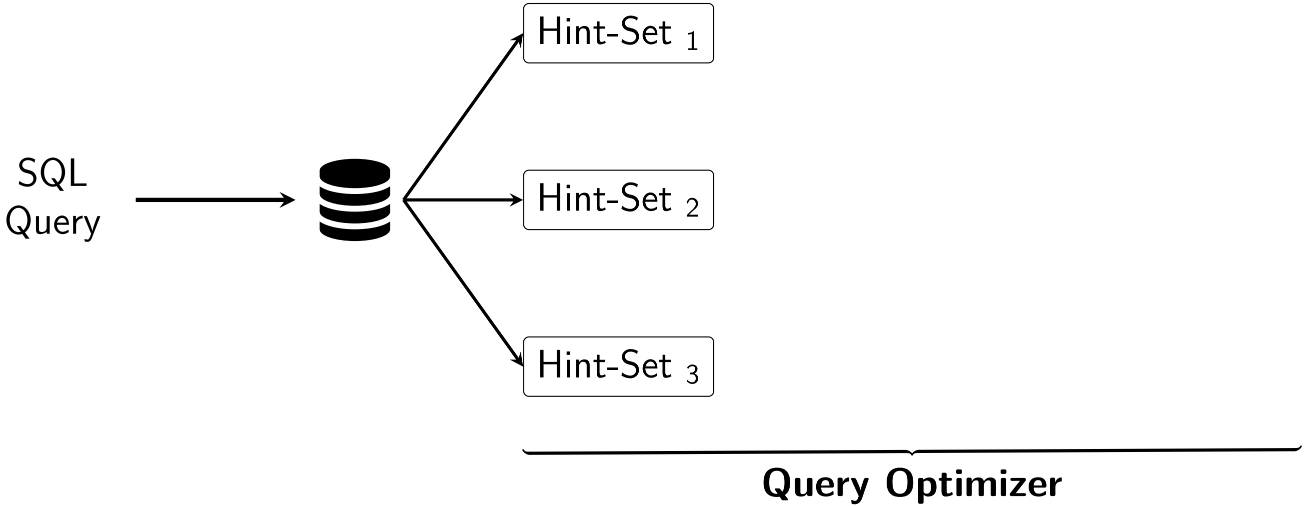
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

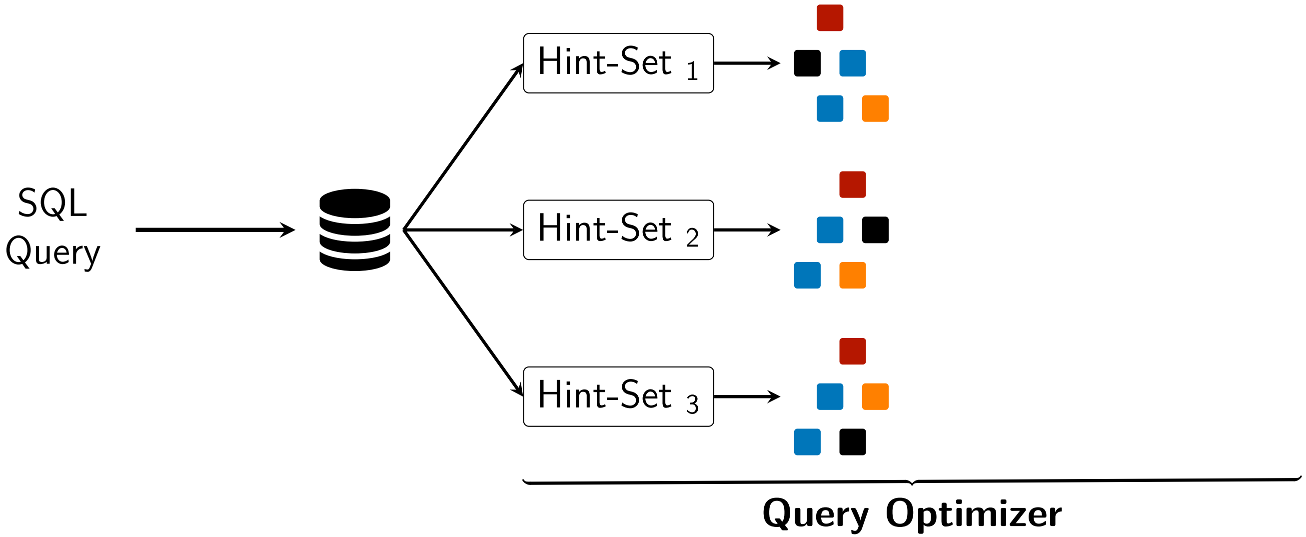
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

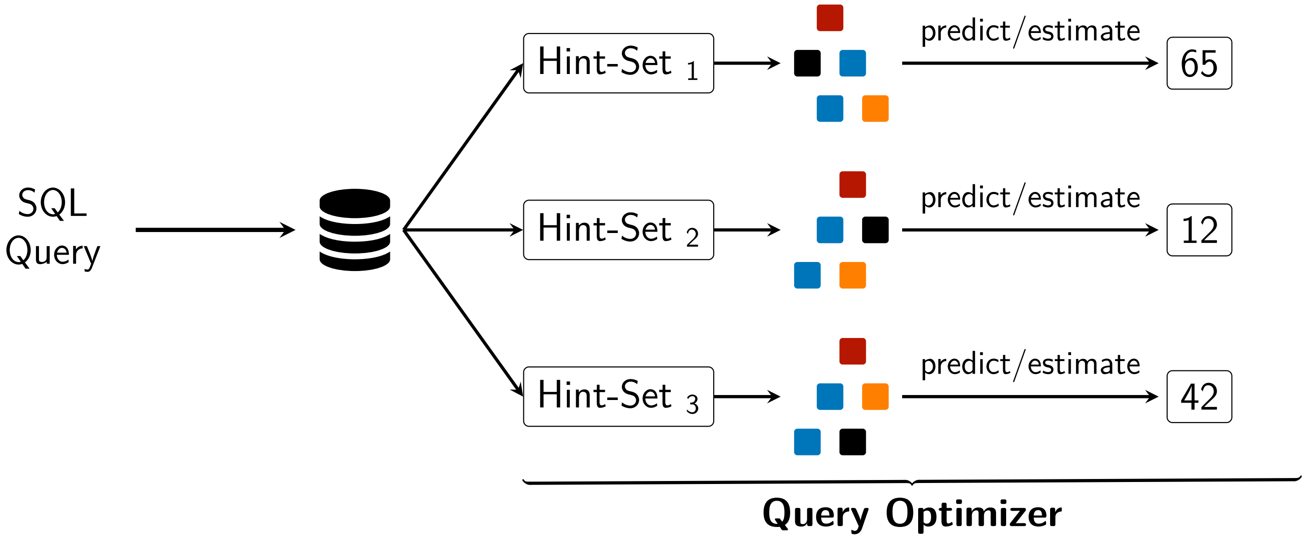
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

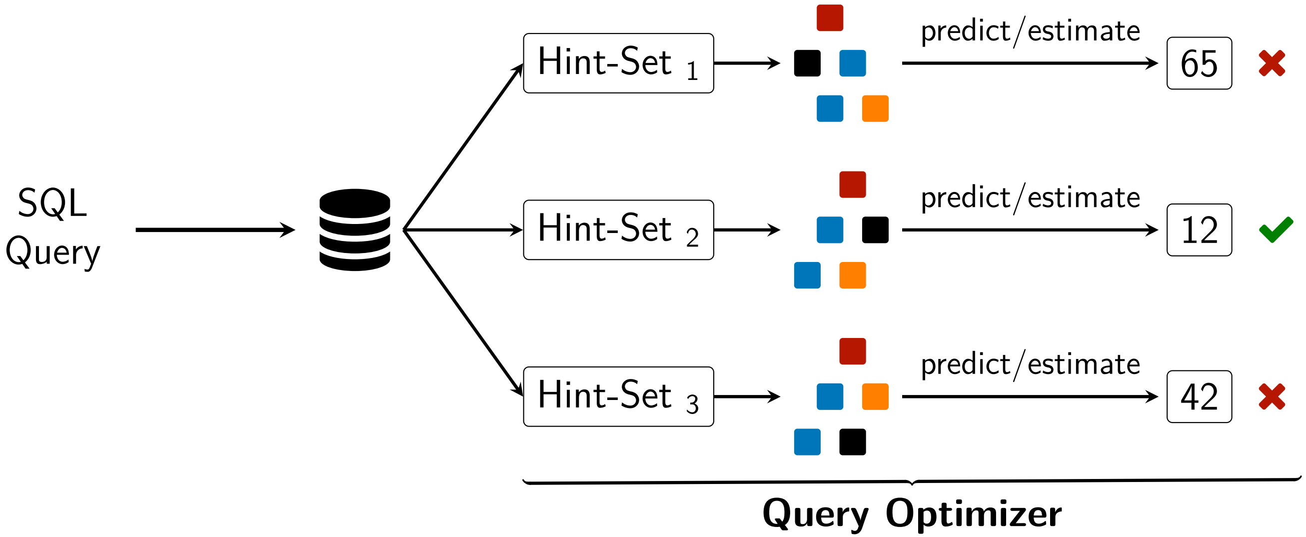
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

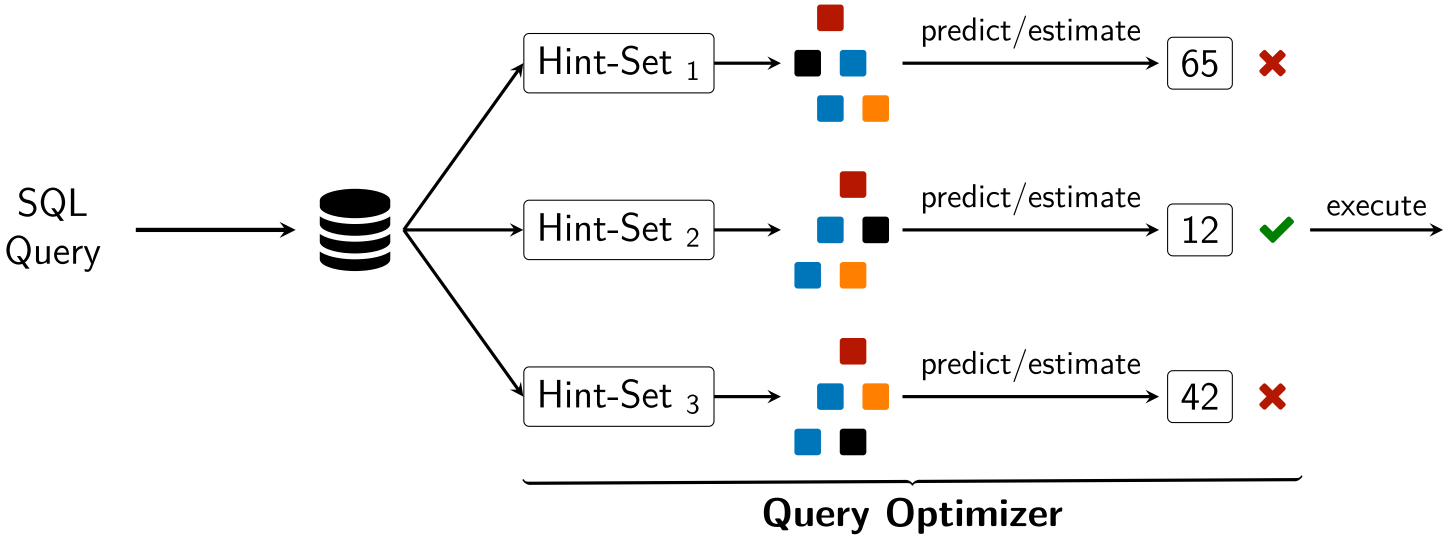
- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.



Background – Steering Query Optimizers **Automatically**

Lot of recent work on steered query optimizers:

- SIGMOD'21: "Bao: Learning to steer query optimizers", Marcus et al.
- SIGMOD'21: "Steering Query Optimizers: A Practical Take on Big Data Workloads", Negi et al.
- SIGMOD'22: "Deploying a Steered Query Optimizer in Production at Microsoft", Zhang et al.

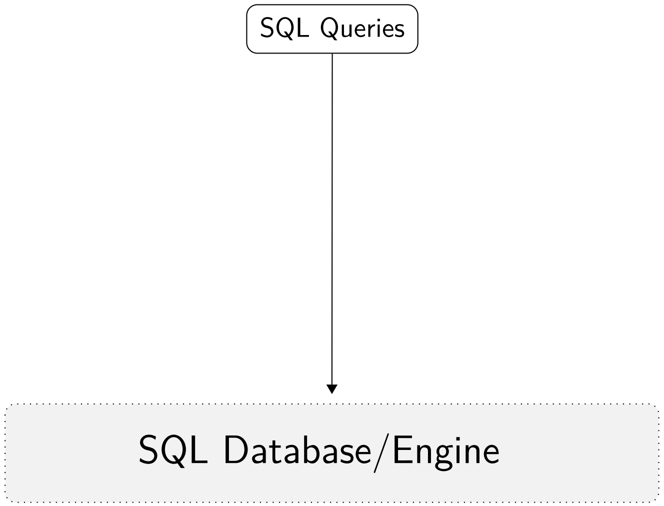


Limitations of Previous Approaches

- ✖ Databases usually expose up to hundreds of knobs
- ✖ Requires good knowledge of the query optimizer
- ✖ Tight integration into the DBMS

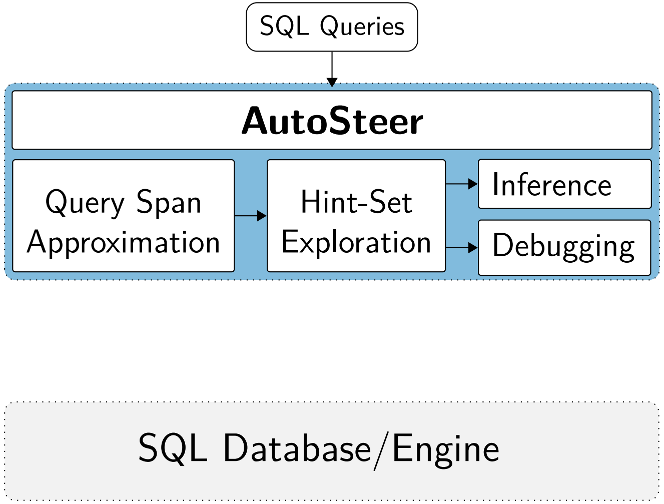
Limitations of Previous Approaches

- ✖ Databases usually expose up to hundreds of knobs
- ✖ Requires good knowledge of the query optimizer
- ✖ Tight integration into the DBMS



Limitations of Previous Approaches

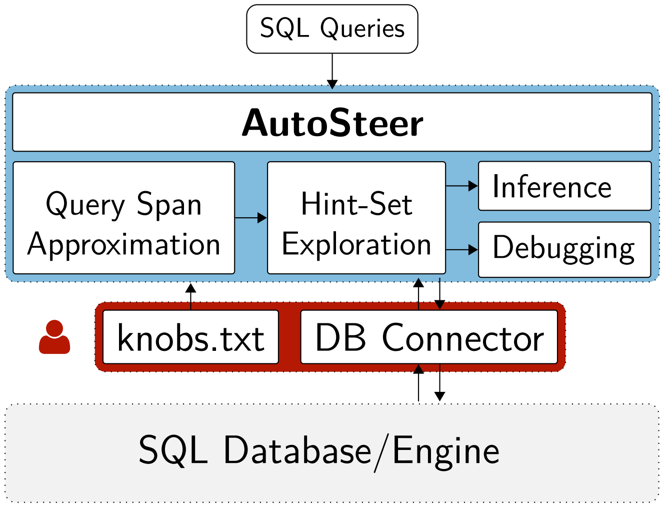
- ✖ Databases usually expose up to hundreds of knobs
- ✖ Requires good knowledge of the query optimizer
- ✖ Tight integration into the DBMS



AutoSteer is a generic framework to steer query optimizers outside the DBMS!

Limitations of Previous Approaches

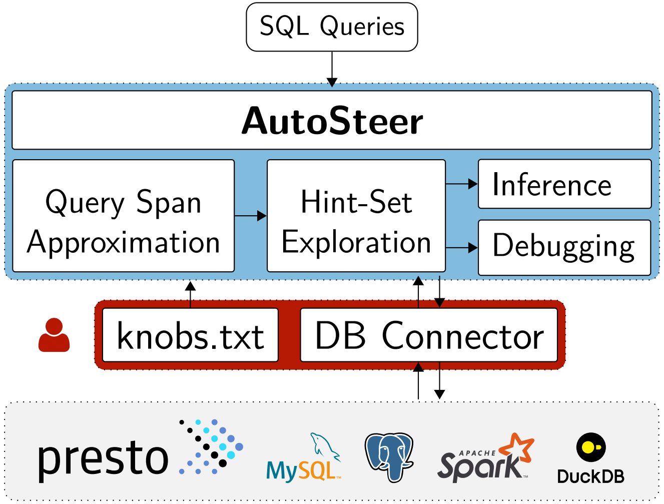
- ✖ Databases usually expose up to hundreds of knobs
- ✖ Requires good knowledge of the query optimizer
- ✖ Tight integration into the DBMS



AutoSteer is a generic framework to steer query optimizers outside the DBMS!

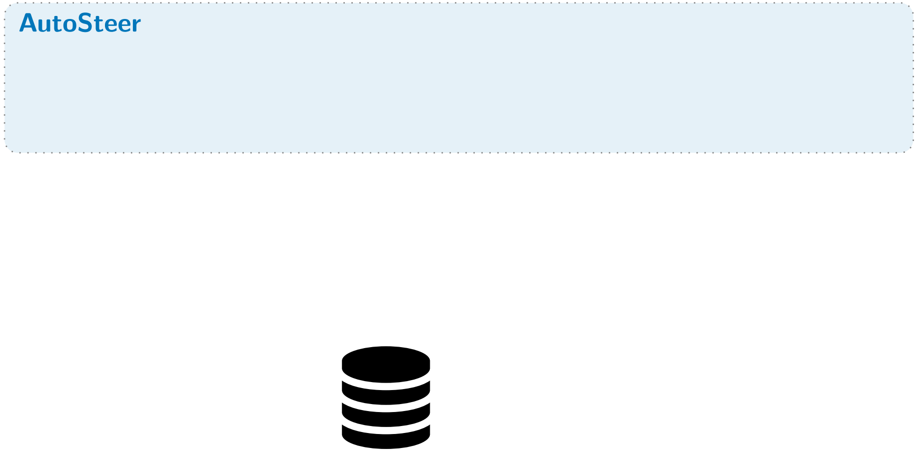
Limitations of Previous Approaches

- ✖ Databases usually expose up to hundreds of knobs
- ✖ Requires good knowledge of the query optimizer
- ✖ Tight integration into the DBMS

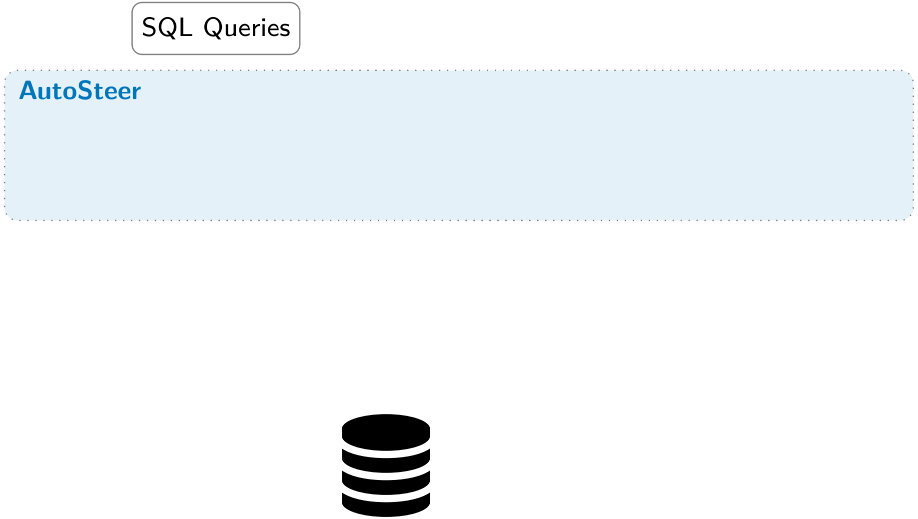


AutoSteer is a generic framework to steer query optimizers outside the DBMS!

AutoSteer – Overview

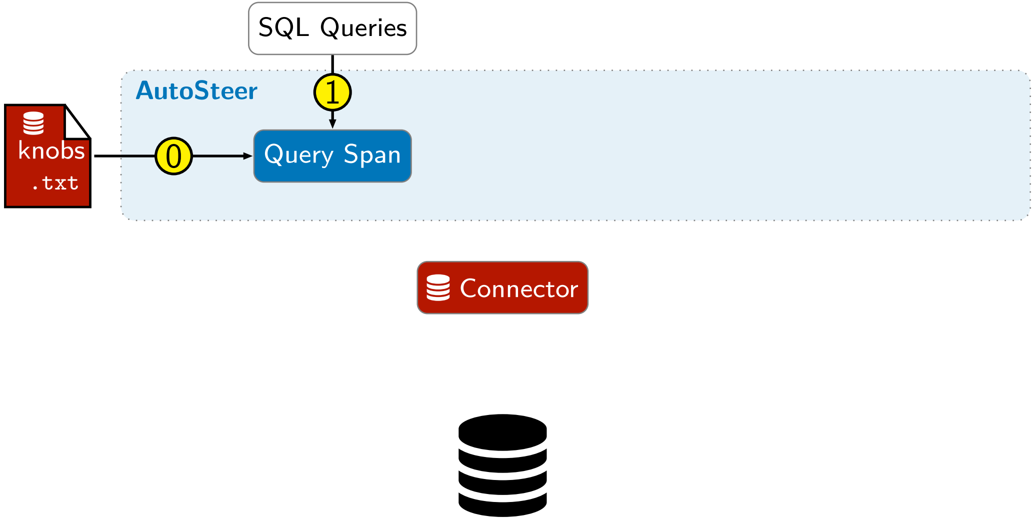


AutoSteer – Overview



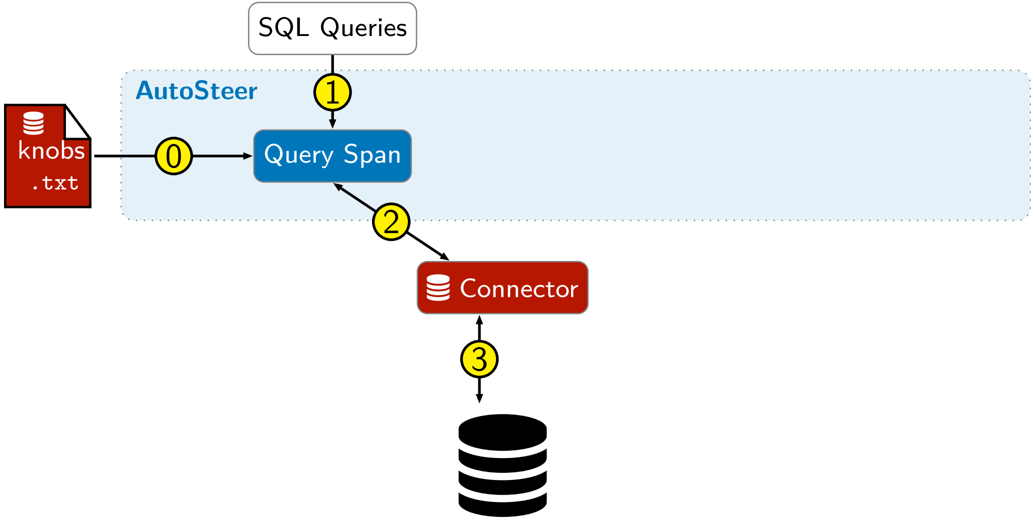
AutoSteer – Overview

AutoSteer – Overview



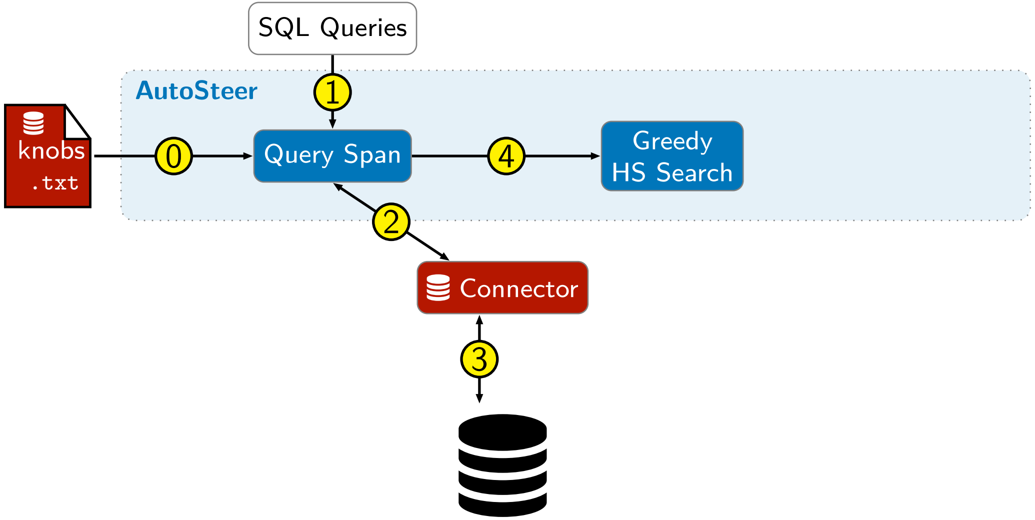
A **Query Span** contains all knobs that affect the query's optimization!

AutoSteer – Overview



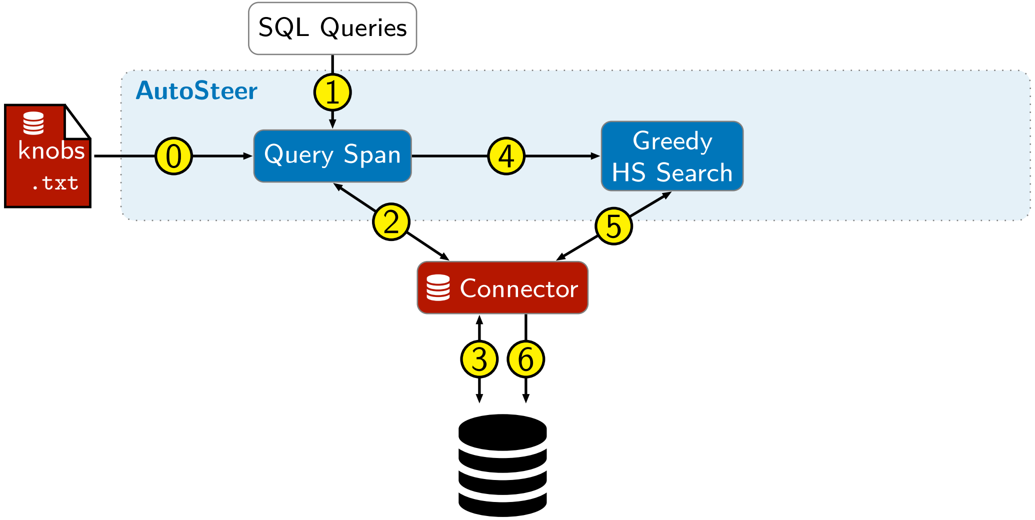
A **Query Span** contains all knobs that affect the query's optimization!

AutoSteer – Overview

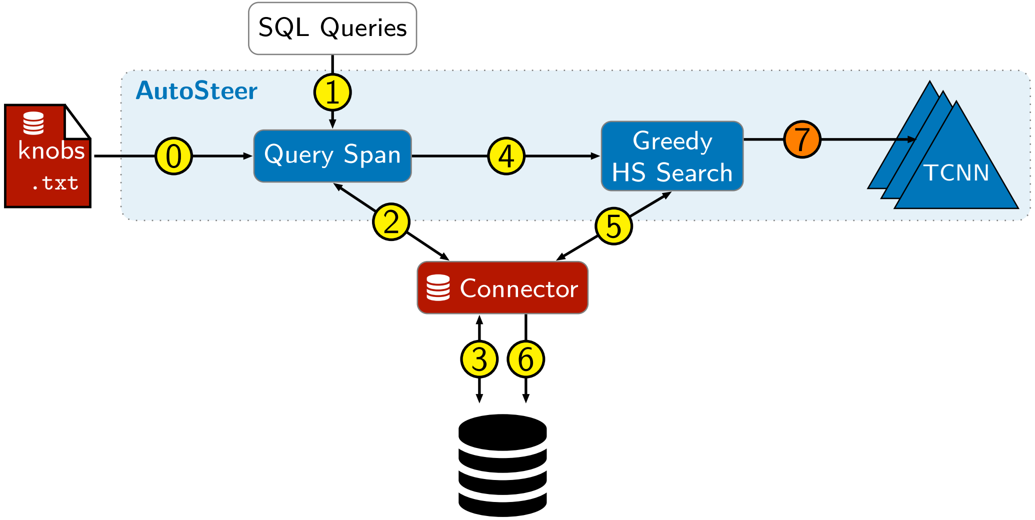


The **greedy search** aims at finding the top hint-sets for a query.

AutoSteer – Overview



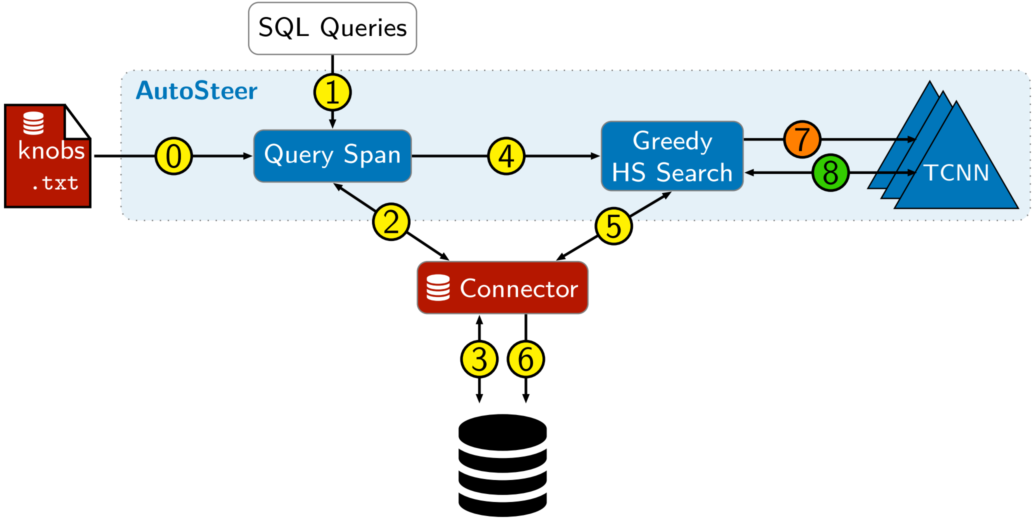
AutoSteer – Overview



Training Mode

AutoSteer **generates training data** by exploring and **executing** alternative plans.

AutoSteer – Overview



Training Mode

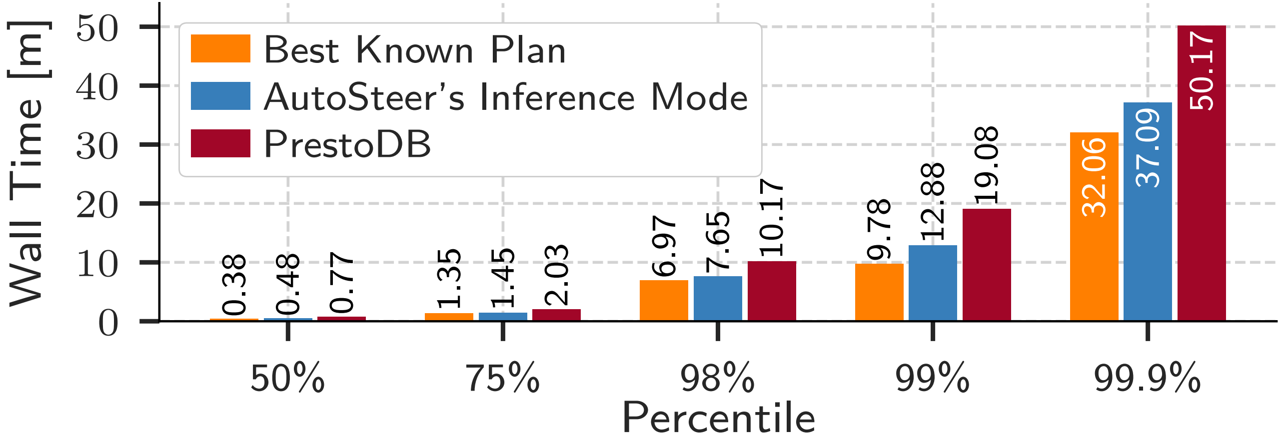
AutoSteer **generates training data** by exploring and **executing** alternative plans.

Inference Mode

AutoSteer **steers queries at runtime** and uses the TCNN to **infer execution times**.

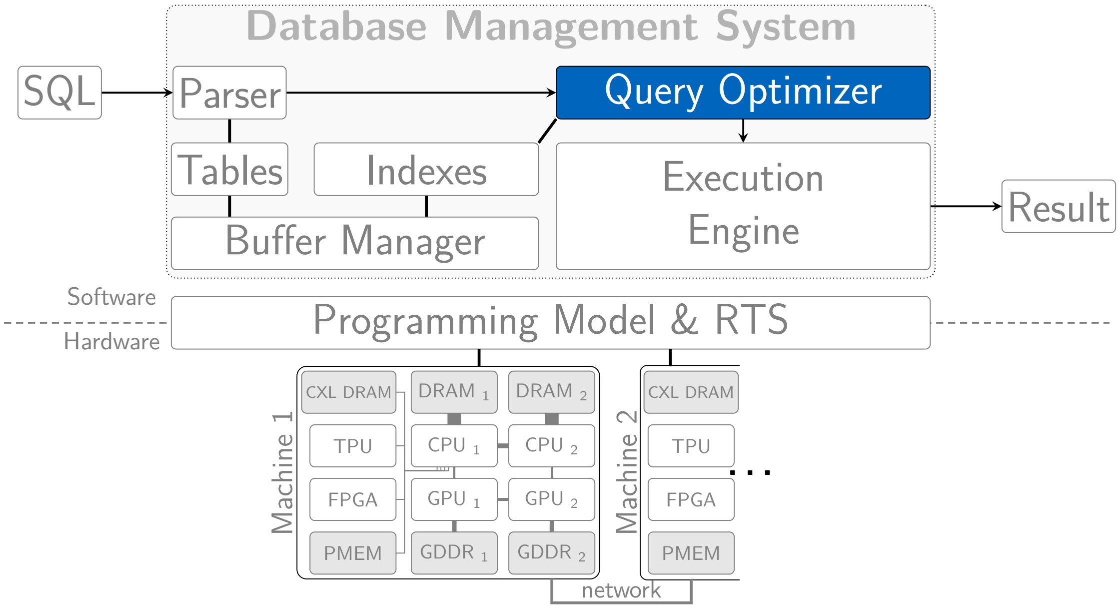
Dashboard Application at Meta – PrestoDB

- Focus on **tail latencies**
- >3000 Queries, scanning PBs of data, hundreds of worker nodes
- Workload runs multiple times per day



AutoSteer significantly reduces tail latencies of production workloads at Meta

P2: AutoSteer



AutoSteer: Learned Query Optimization for Any SQL Database

Christoph Anneser[1]
Technical University of Munich
anneser@in.tum.de

Nesime Tatbul
Intel Labs and MIT
tatbul@csail.mit.edu

David Cohen
Intel
david.e.cohen@intel.com

Zhenggang Xu
Meta
zhenggang@fb.com

Prithviraj Pandian
Meta
prithvip@fb.com

Nikolay Laptev
Meta
nlaptev@fb.com

Ryan Marcus
University of Pennsylvania
rcmarcus@seas.upenn.edu

ABSTRACT

This paper presents AutoSteer, a learning-based solution that automatically drives query optimization in any SQL database that exposes tunable optimizer knobs. AutoSteer builds on the Bandit optimizer (Bao) and extends it with new capabilities (e.g., automated hint-set discovery) to minimize integration effort and facilitate usability in both monolithic and disaggregated SQL systems. We successfully applied AutoSteer on PostgreSQL, PrestoDB, SparkSQL, MySQL, and DuckDB – five popular open-source database engines with diverse query optimizers. We then conducted a detailed experimental evaluation with public benchmarks (JOB, Stackoverflow, TPC-DS) and a production workload from Meta's PrestoDB deployments. Our evaluation shows that AutoSteer can not only outperform these engines' native query optimizers (e.g., up to 40% improvements for PrestoDB) but can also match the performance of Bao-for-PostgreSQL with reduced human supervision and increased adaptivity, as it replaces Bao's static, expert-picked hint-sets with those that are automatically discovered. We also provide an open-source implementation of AutoSteer together with a visual tool for interactive use by query optimization experts.

PVLDB Reference Format:
Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. AutoSteer: Learned Query Optimization for Any SQL Database. PVLDB, 16(12): 3515 - 3527, 2023.
doi:10.14778/3611540.3611544

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/IntelLabs/Auto-Steer.

1 INTRODUCTION

Our research community has been making rapid strides in applying modern machine learning (ML) techniques to tackle longstanding problems in databases [6, 24, 48]. Learned query optimization lies at the forefront of this progress [51]. Various techniques from query-driven and data-driven to their combinations have been proposed [19, 20, 23] – not only to improve core query optimization tasks

[1]Work done while at Intel.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611544

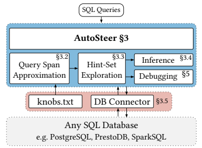


Figure 1: AutoSteer is a framework for steering query optimizers of SQL databases autonomously. For each query, we search for effective rewrite rules and store them in the query span. Then, we use a greedy algorithm to explore alternative query plans efficiently. The results can be used to train predictive models or to debug existing query optimizers.

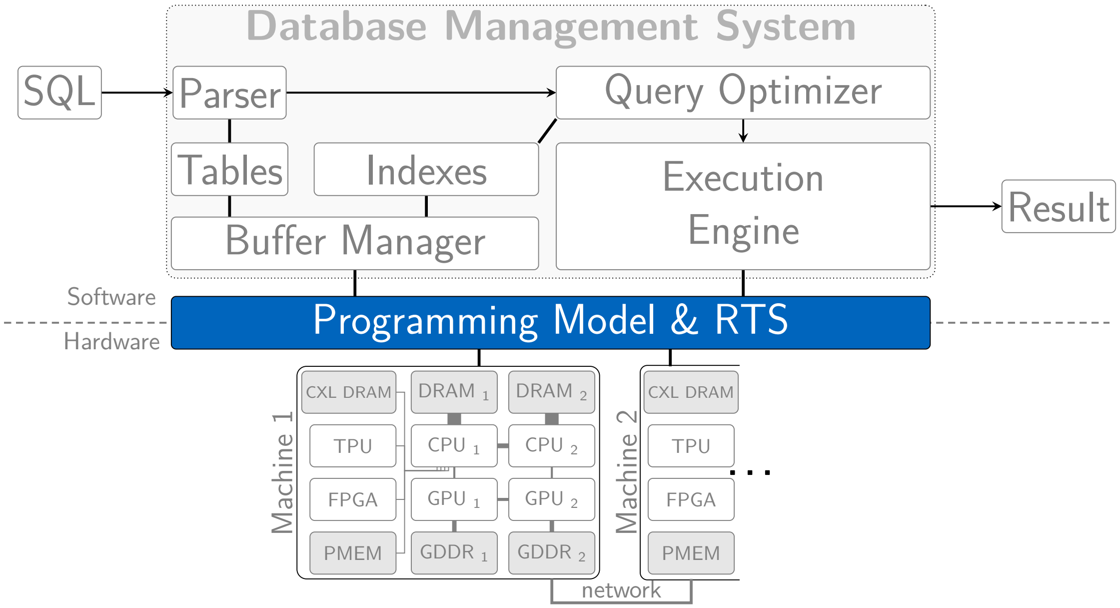
such as cardinality estimation [22, 23, 31, 32, 37, 39, 43], join order enumeration [29], or query rewriting [50], but also to build end-to-end query optimizers replacing [28, 42] or enhancing [27, 30, 44, 47] traditional ones. The practicality and robustness of these techniques are critical when applying them in industrial settings [47].

The so-called "steering approach" of Bao (Bandit optimizer) has been a successful example of a practical solution due to its emphasis on shortening training times, adaptivity to dynamic workloads, and ability to integrate with traditional optimizers [27]. Given a pre-determined collection of "*hint-sets*" (a hint-set indicates which query rewrite rules (RRs) should be considered in query optimization), Bao learns to steer an already existing query optimizer by helping it choose the right hint-set to use for every incoming query. This way, potential planning mistakes of traditional query optimizers can be avoided. As Bao's initial success continues to drive wider adoption in increasingly more sophisticated deployment and workload settings [3, 47], it also brings new challenges to the surface. We tackle two such challenges in this paper:

Integration effort: Adopting Bao to a new database system requires coming up with the right collection of hint-sets. In the original approach developed for PostgreSQL [1], a static collection of 48 hint-sets is manually selected based on deep knowledge of the underlying PostgreSQL optimizer [5], after which Bao independently learns to choose among these hint-sets on a per-query basis. Unfortunately, manually engineering feature hint-sets can be quite

@VLDB'23

P3: Programming Fully Disaggregated Systems



Programming Fully Disaggregated Systems

Christoph Anneser Lukas Vogel Ferdinand Gruber
Maximilian Bandle Jana Giceva
Technical University of Munich
firstname.lastname@in.tum.de

Abstract

With full resource disaggregation on the horizon, it is unclear what the most suitable *programming model* is that enables dataflow developers to fully harvest the potential that recent hardware developments offer. In our vision, we propose to raise the abstraction level to allow developers to primarily reason about their dataflow and the requirements that need to be met by the underlying system in a declarative fashion. Underneath, the system works with typed memory regions and uses the notion of ownership that allows for more flexible memory management across the different compute devices and the tasks mapped onto them. This requires a holistic approach that crosses multiple layers of the system stack, opening exciting systems research questions.

ACM Reference Format:
Christoph Anneser, Lukas Vogel, Ferdinand Gruber, Maximilian Bandle, and Jana Giceva. 2023. Programming Fully Disaggregated Systems. In *Workshop on Hot Topics in Operating Systems (HOTOS '23), June 22–24, 2023, Providence, RI, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3593856.3595889

1 Introduction

With the ever-increasing demand for data, where the datasphere volume is expected to reach 175ZB by 2025 [50], we have reached the point where moving data is the dominating cost factor in data centers [34, 45]. Cloud providers race to serve the different requirements of modern workloads better but with pressure to achieve it in a more sustainable fashion [51]. To improve efficiency, data centers have evolved to more loosely coupled software-defined racks, where they disaggregate resources over fast network interconnects [52].

However, until recently, coherent memory remained tightly coupled, and servers had to be equipped with large memory capacities to serve peak workloads reliably. This

HOTOS '23, June 22–24, 2023, Providence, RI, USA

ACM ISBN 979-8-4007-0195-5/23/06.
https://doi.org/10.1145/3593856.3595889

(a) Current Architectures (b) Our Vision

Figure 1: Moving from a compute-centric to a memory-centric architecture.

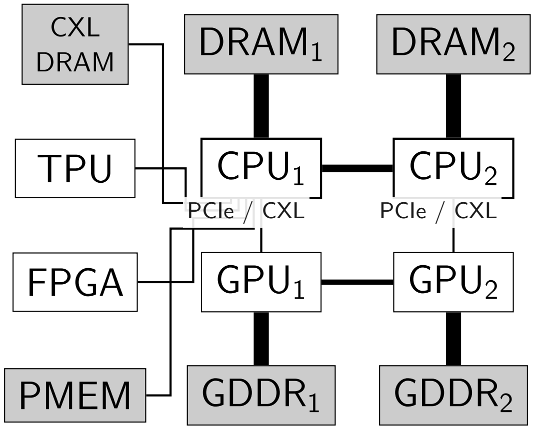
overprovisioning is a considerable cost (50% of Azure's servers [5] and 40% of Meta's rack costs come from memory [40]) for a resource that could not be properly pooled. The average memory utilization reported by many cloud vendors remains low, typically in the range of 50-65% [38, 56]. Therefore, data centers could reduce costs by pooling different types of memory [9, 11, 21, 57] and compute devices [6, 13, 17–19, 30, 33, 47] by connecting them with fast networks [14, 45].

However, data and compute placement within these pools significantly impacts the overall system performance. For example, non-uniform memory accesses (NUMA) can slow down algorithms by up to 3× [39]. Similarly, a naïve data placement in a heterogeneous storage landscape can reduce a database system's performance by up to 3× [59].

Moreover, today, optimal placement has become an issue even *within* single processors. For example, take the recently introduced Intel's 4th Generation Intel® Xeon® Scalable Processors – codenamed *Sapphire Rapids* [7]. They have built-in encryption, compression, streaming, and high-bandwidth memory accelerators. Its most promising feature, however, is the adoption of Compute Express Links™ (CXL™) – an industry standard for cache-coherent interconnects for processors, memory expansion, and accelerators based on PCIe 5.0, which has been adopted by companies like Intel, AMD, ARM, Samsung, and NVIDIA, amongst others [9]. CXL enables us to first *scale-up* nodes by extending their compute and memory pools with 'pluggable' compute devices and DRAM/PMem expansion cards before we have to rely on more expensive 'scale-outs' to other compute nodes that

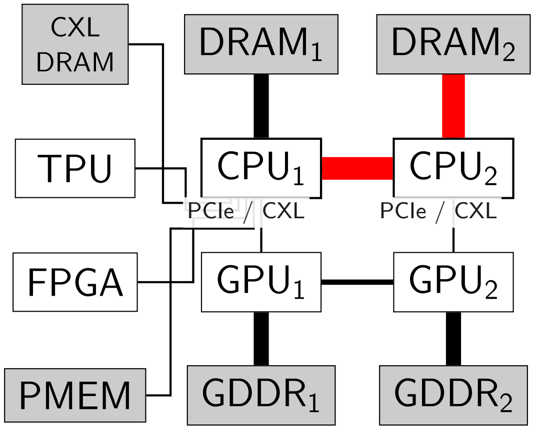
@HotOS'23

Motivation

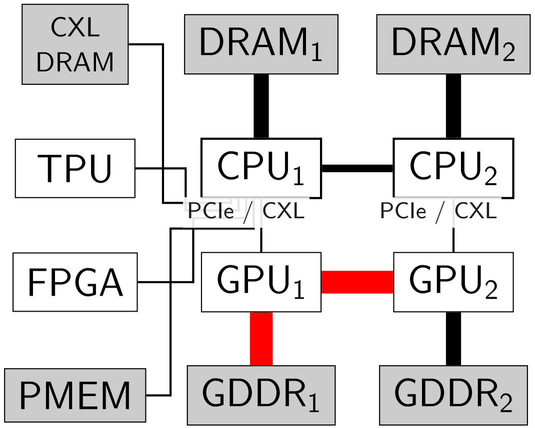


CXL DRAM
DRAM1
DRAM2
TPU
CPU1
CPU2
PCIe / CXL
PCIe / CXL
FPGA
GPU1
GPU2
PMEM
GDDR1
GDDR2

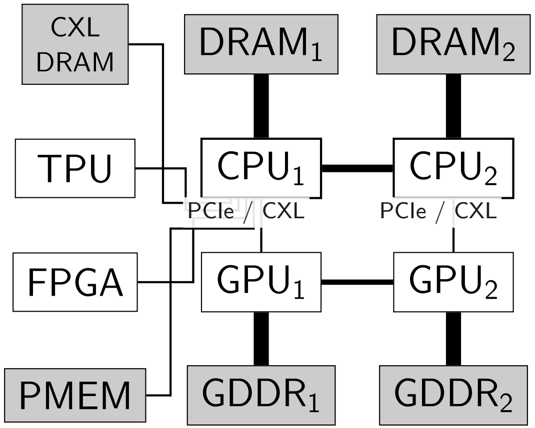
Motivation


CXL DRAM
DRAM1
DRAM2
TPU
CPU1
CPU2
PCIe / CXL
PCIe / CXL
FPGA
GPU1
GPU2
PMEM
GDDR1
GDDR2

Motivation

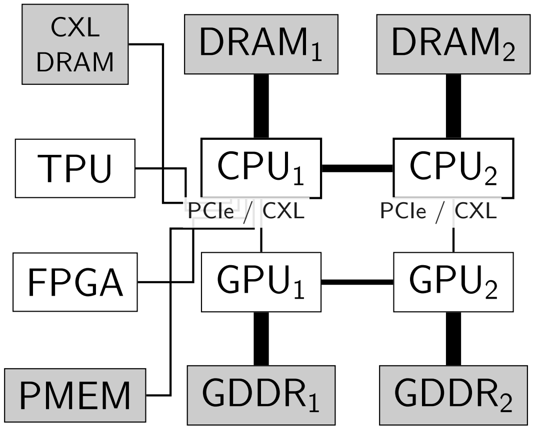

CXL DRAM
DRAM$_1$
DRAM$_2$
TPU
CPU$_1$
CPU$_2$
PCIe / CXL
PCIe / CXL
FPGA
GPU$_1$
GPU$_2$
PMEM
GDDR$_1$
GDDR$_2$

Motivation



Name	Bw.	Lat.	Gran.	Attached	Sync	Persist.
Cache	++	++	1 B	CPU	✓	✗
HBM	++	+	64 B	CPU	✓	✗
DRAM	+	+	64 B	CPU	✓	✗
PMem	○	○	256 B	CPU	✓	✓
CXL-DRAM	○	○	64 B	PCIe	✓/ ✗	✓/ ✗
Disagg. Mem.	○	–	?	NIC	✗	✓/ ✗
SSD	–	–	4 KiB	PCIe	✗	✓
HDD	– –	– –	4 KiB	SATA	✗	✓

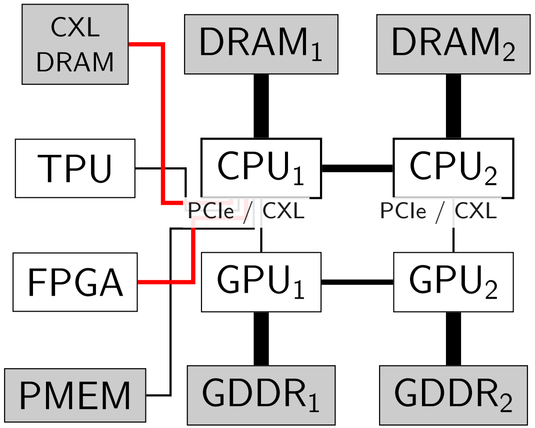
Motivation



Name	Bw.	Lat.	Gran.	Attached	Sync	Persist.
Cache	++	++	1 B	CPU	✓	✗
HBM	++	+	64 B	CPU	✓	✗
DRAM	+	+	64 B	CPU	✓	✗
PMem	○	○	256 B	CPU	✓	✓
CXL-DRAM	○	○	64 B	PCIe	✓/ ✗	✓/ ✗
Disagg. Mem.	○	–	?	NIC	✗	✓/ ✗
SSD	–	–	4 KiB	PCIe	✗	✓
HDD	– –	– –	4 KiB	SATA	✗	✓

How to develop & optimize software for heterogeneous, disaggregated hardware?

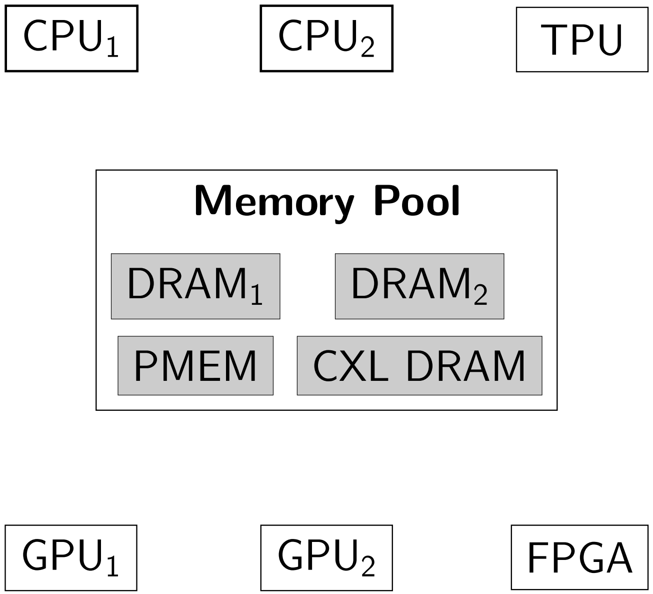
Motivation



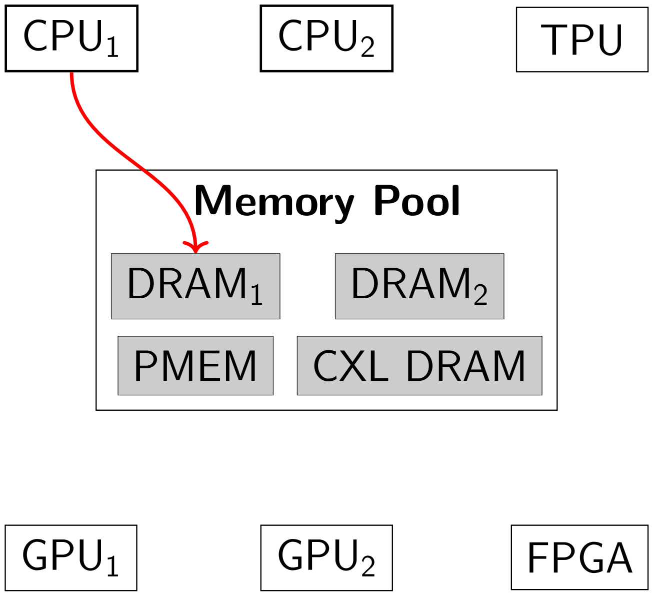
Name	Bw.	Lat.	Gran.	Attached	Sync	Persist.
Cache	++	++	1 B	CPU	✓	✗
HBM	++	+	64 B	CPU	✓	✗
DRAM	+	+	64 B	CPU	✓	✗
PMem	○	○	256 B	CPU	✓	✓
CXL-DRAM	○	○	64 B	PCIe	✓/ ✗	✓/ ✗
Disagg. Mem.	○	–	?	NIC	✗	✓/ ✗
SSD	–	–	4 KiB	PCIe	✗	✓
HDD	– –	– –	4 KiB	SATA	✗	✓

How to develop & optimize software for heterogeneous, disaggregated hardware?

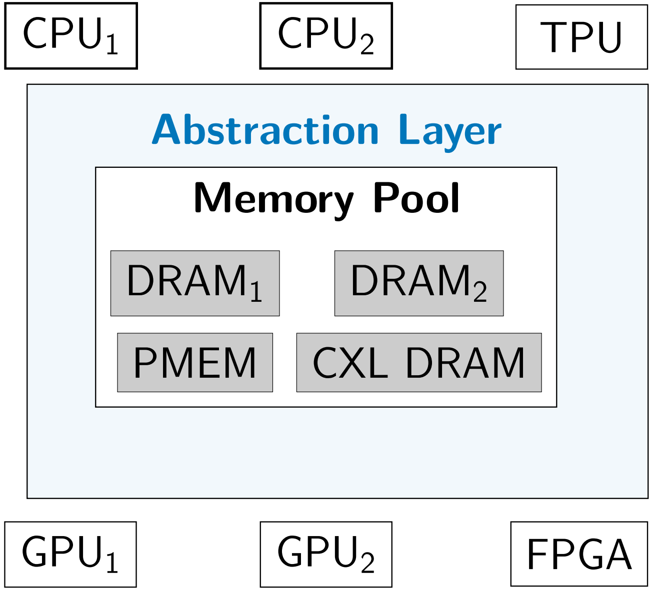
CXL Enables a Memory-Centric View



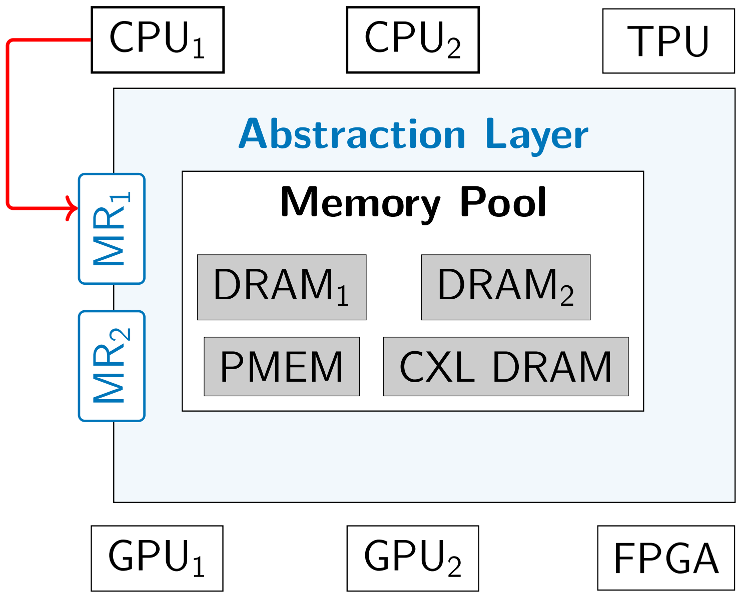
CXL Enables a Memory-Centric View



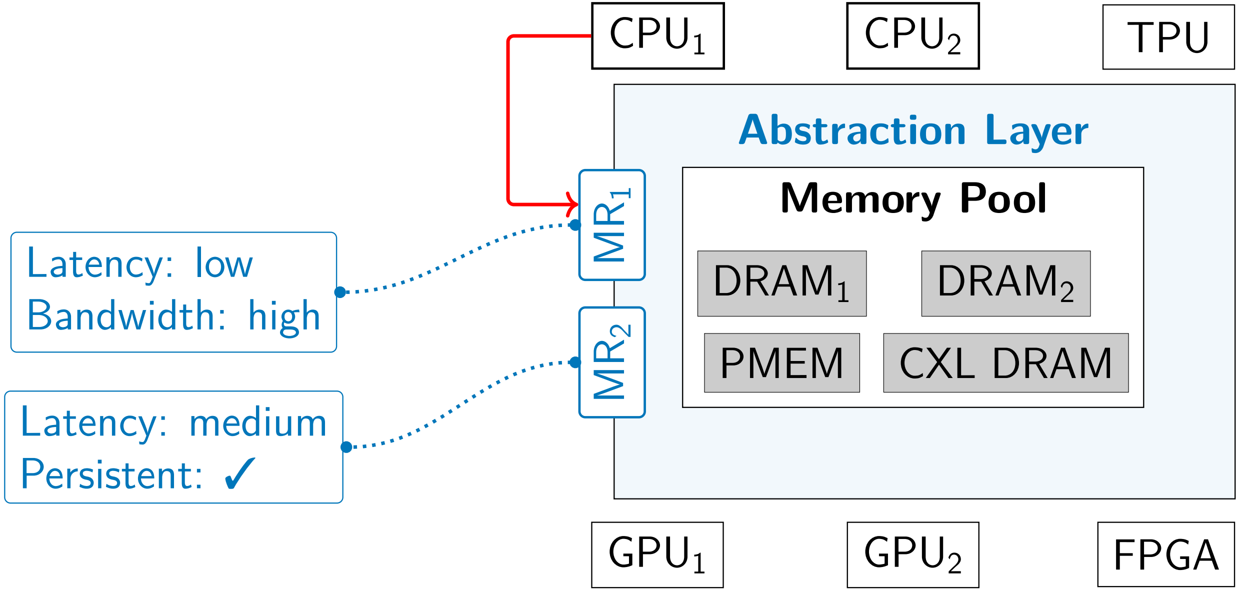
CXL Enables a Memory-Centric View



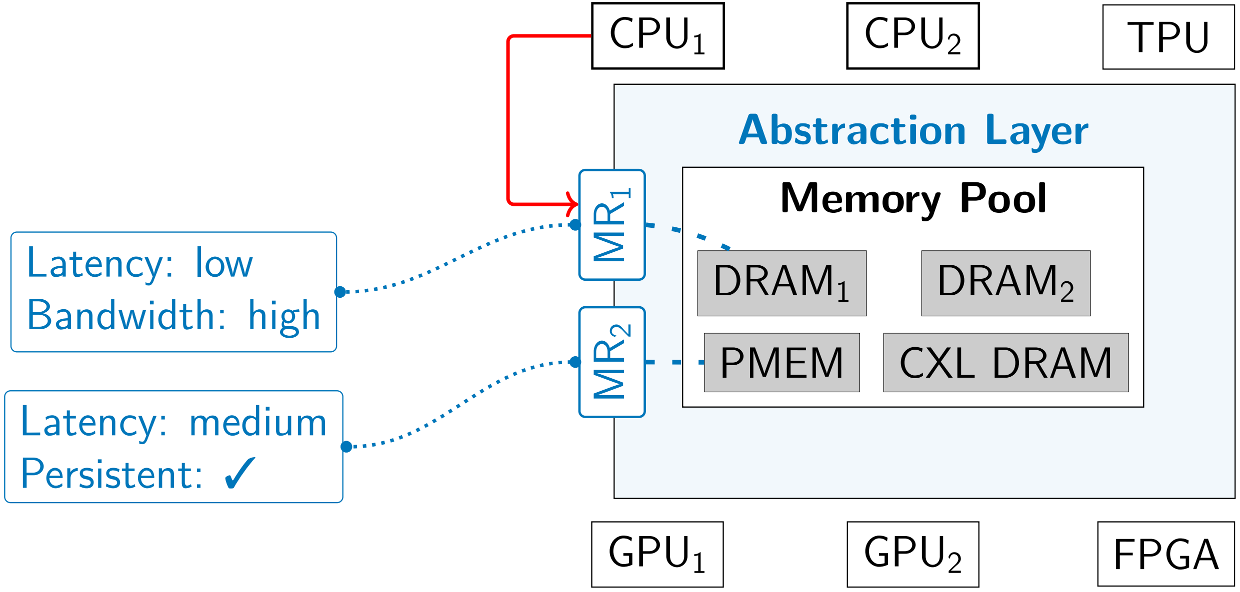
CXL Enables a Memory-Centric View



CXL Enables a Memory-Centric View



CXL Enables a Memory-Centric View



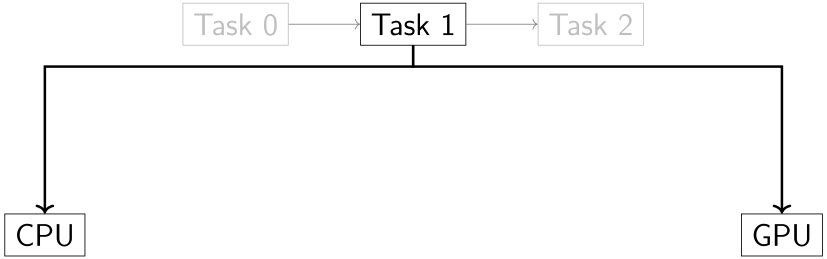
Mapping Memory Regions to Devices

⇒ **Task Placement**

Task 0 → Task 1 → Task 2

Mapping Memory Regions to Devices

⇒ **Task Placement**

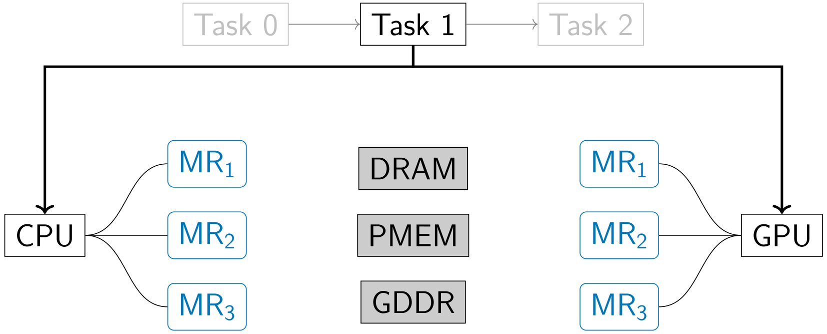


Mapping Memory Regions to Devices

⇒ **Task Placement**

⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync

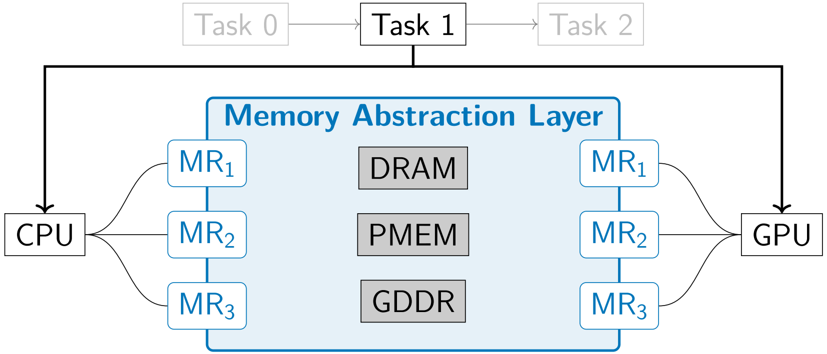


Mapping Memory Regions to Devices

⇒ **Task Placement**

⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync

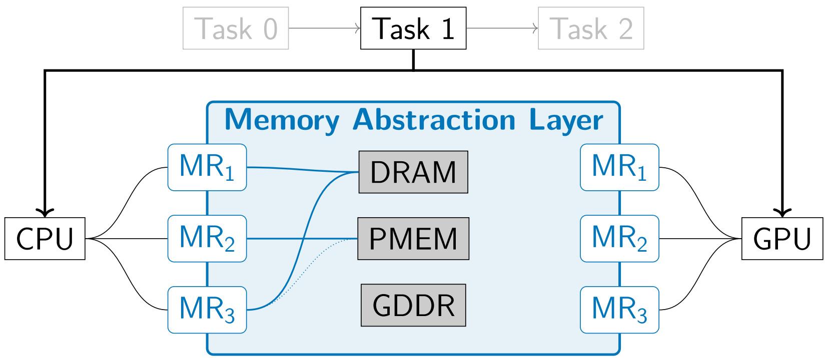


Mapping Memory Regions to Devices

⇒ **Task Placement**

⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync

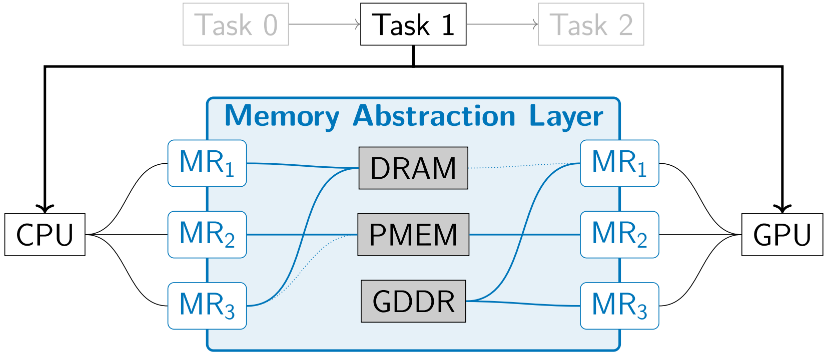


Mapping Memory Regions to Devices

⇒ **Task Placement**

⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync



Mapping Memory Regions to Devices

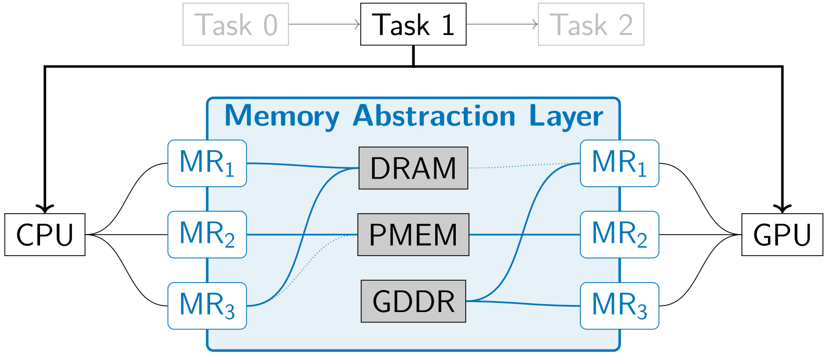
⇒ **Task Placement**

⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync

⇒ **Handovers:**

MR_1: T_0 Output, T_1 Input
MR_2: T_1 Output, T_2 Input



Mapping Memory Regions to Devices

⇒ **Task Placement**

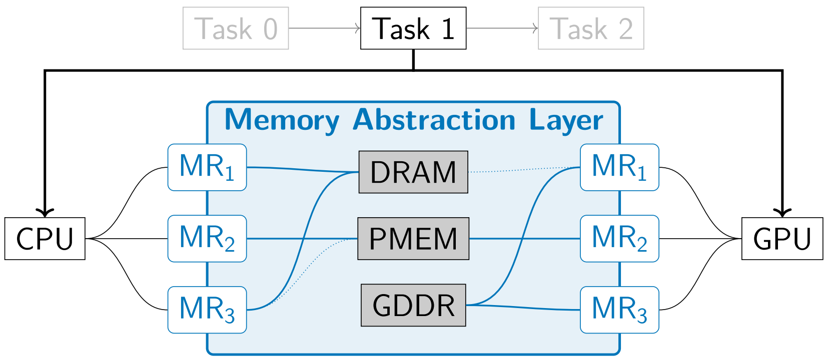
⇒ **Memory Region Properties:**

MR_1: low lat., sync
MR_2: low lat., persistent, async
MR_3: low lat., high bandwith, sync

⇒ **Handovers:**

MR_1: T_0 Output, T_1 Input
MR_2: T_1 Output, T_2 Input

⇒ **Device Utilization**



Mapping Memory Regions to Devices

⇒ **Task Placement**

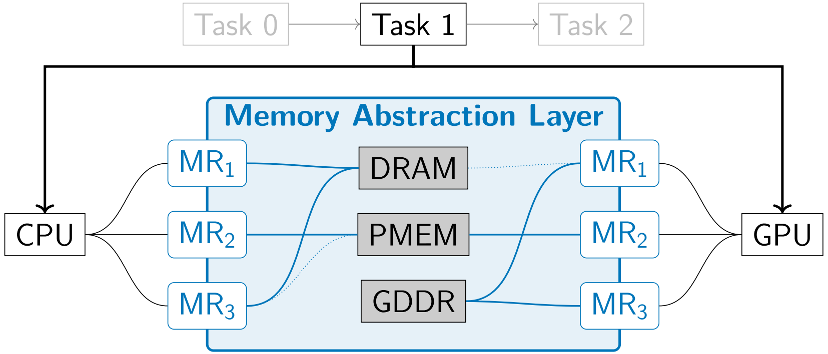
⇒ **Memory Region Properties:**

- MR_1: low lat., sync
- MR_2: low lat., persistent, async
- MR_3: low lat., high bandwith, sync

⇒ **Handovers:**

- MR_1: T_0 Output, T_1 Input
- MR_2: T_1 Output, T_2 Input

⇒ **Device Utilization**



$ RTS needs a comprehensive cost model and late binding

Typed Memory Regions

Typed Memory Regions

Communication

- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Typed Memory Regions

Communication

- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data

- Properties: coherent, async

Typed Memory Regions

Communication

- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data

- Properties: coherent, async

Thread-local State

- Properties: non-coherent, sync, fast

Typed Memory Regions

Communication
- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data
- Properties: coherent, async

Thread-local State
- Properties: non-coherent, sync, fast

Global State

Typed Memory Regions

Communication

- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data

- Properties: coherent, async

Thread-local State

- Properties: non-coherent, sync, fast

Global State

Global Scratch

Typed Memory Regions

Communication
- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data
- Properties: coherent, async

Thread-local State
- Properties: non-coherent, sync, fast

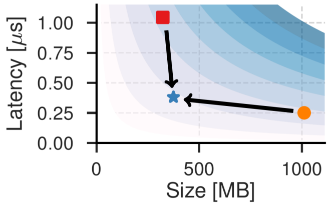
Global State

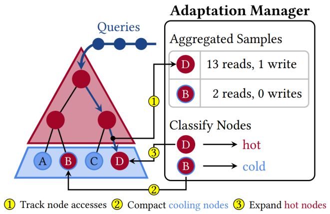
Global Scratch

Private Scratch

Conclusions

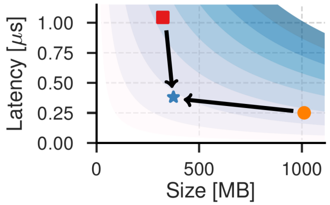
Adaptive Hybrid Indexes reduce storage overheads & retain high-performance

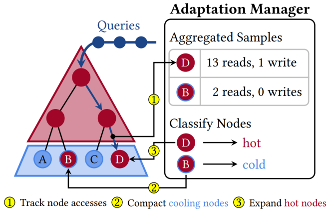


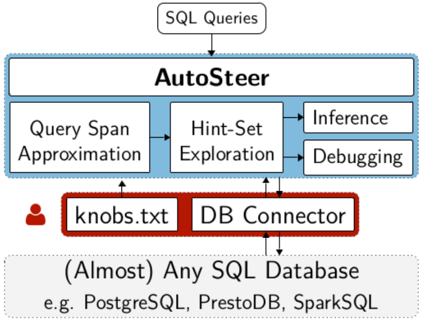


Conclusions

Adaptive Hybrid Indexes reduce storage overheads & retain high-performance

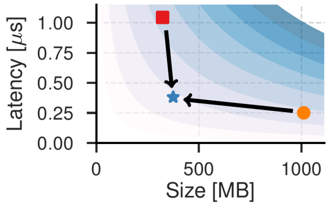


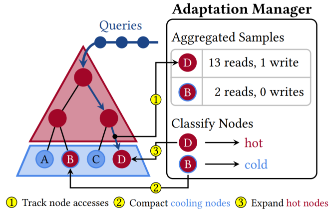




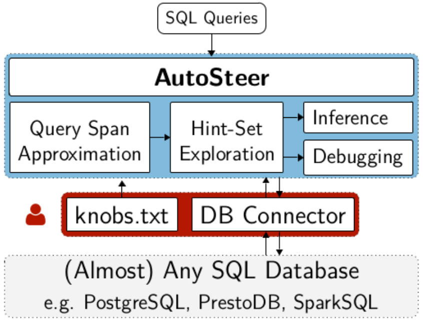
AutoSteer is a framework to steer rule-based query optimizers

Conclusions



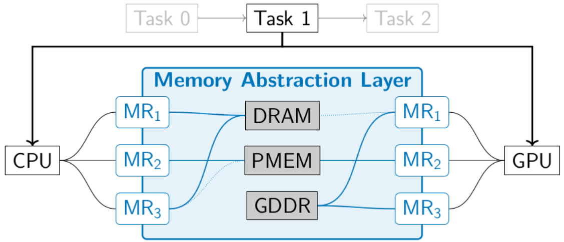


Adaptive Hybrid Indexes reduce storage overheads & retain high-performance



AutoSteer is a framework to steer rule-based query optimizers

Memory-centric programming model for disaggregated memory



Questions?

Thank you for your attention!