Machine Learning for Data Management: A System View

Guoliang Li, Xuanhe Zhou

Department of Computer Science,

Tsinghua University, Beijing, China

Revisit database systems: What are the critical *requirements*

database development

File System

Relations ↓ **E**asy to manipulate data

Relational Model, SQL

transactions ↓ **A**tomicity, **C**onsistency, **I**solation, **D**urability

OLTP: rollbacks, triggers, locking, logging, …

data analysis ↓ **H**igh Quality

OLAP: JIT compilation, vectorized execution, …

Scale out ↓ **U**nstructured Data; **H**igh Scalability

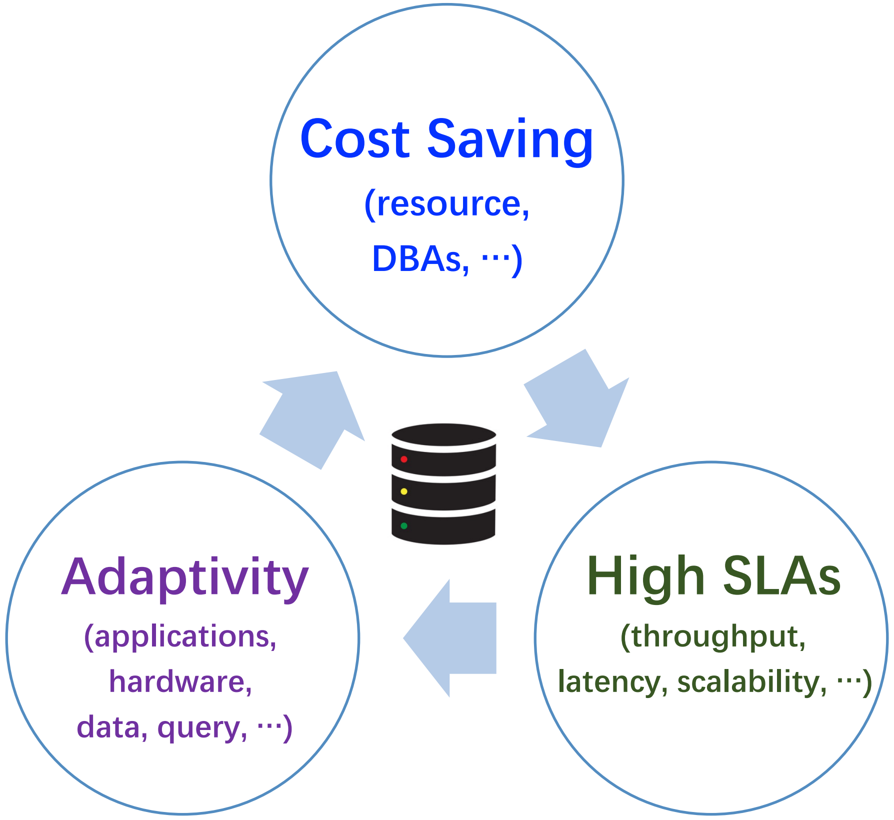
NoSQL: data models, indexing, partitioning, …

transactions +big data ↓ High Scalability; **ACID**

Distributed OLTP: 2PC, Paxos, Distributed SQL…

DBaaS ↓ **F**lexibility, **C**ost-Saving

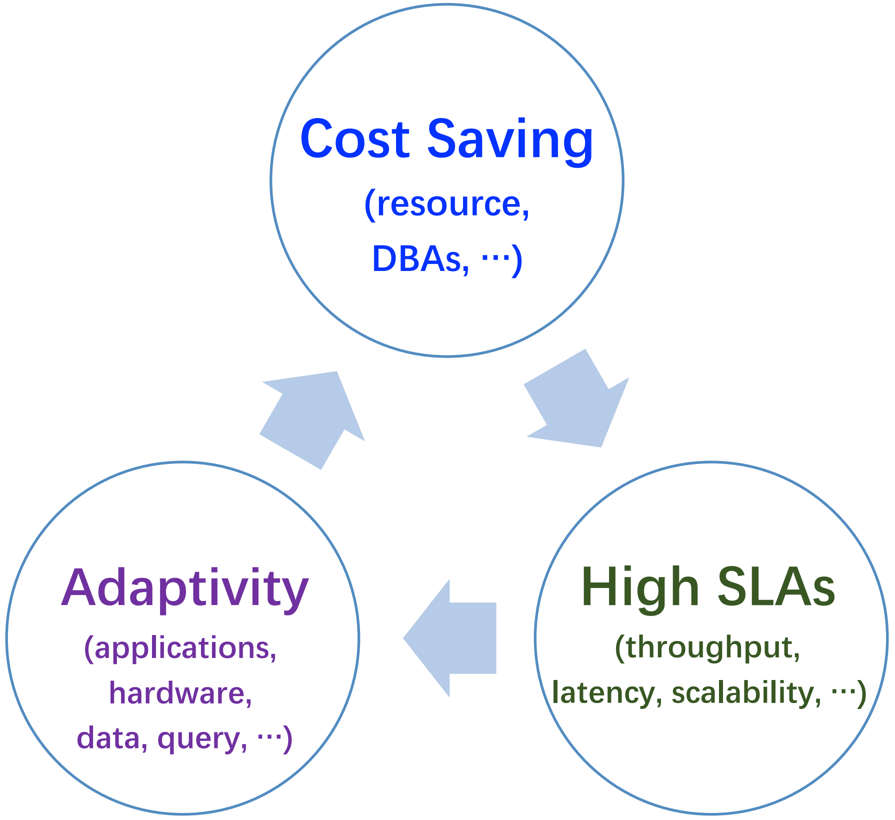
Cloud-Native: compute-storage disaggregation, …



Learned Database

New Opportunities: What benefits can *ML bring for databases*?

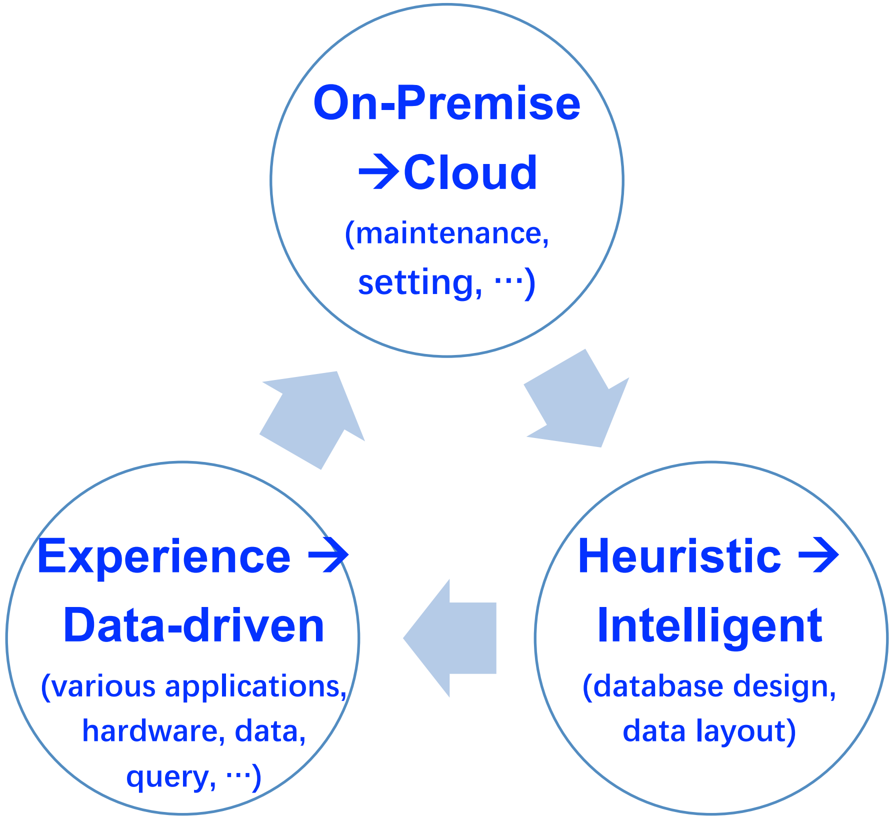
- **Cost Saving: Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency
- **High SLAs: Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance
- **Adaptivity: Empirical → Data-Driven**
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency



Learned Database

New Opportunities: Why Now?

- **Cost Saving: Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency
- **High SLAs: Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance
- **Adaptivity: Empirical → Data-Driven**
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency



Learned Database

Double-Edged Sword: What are the *challenges*?

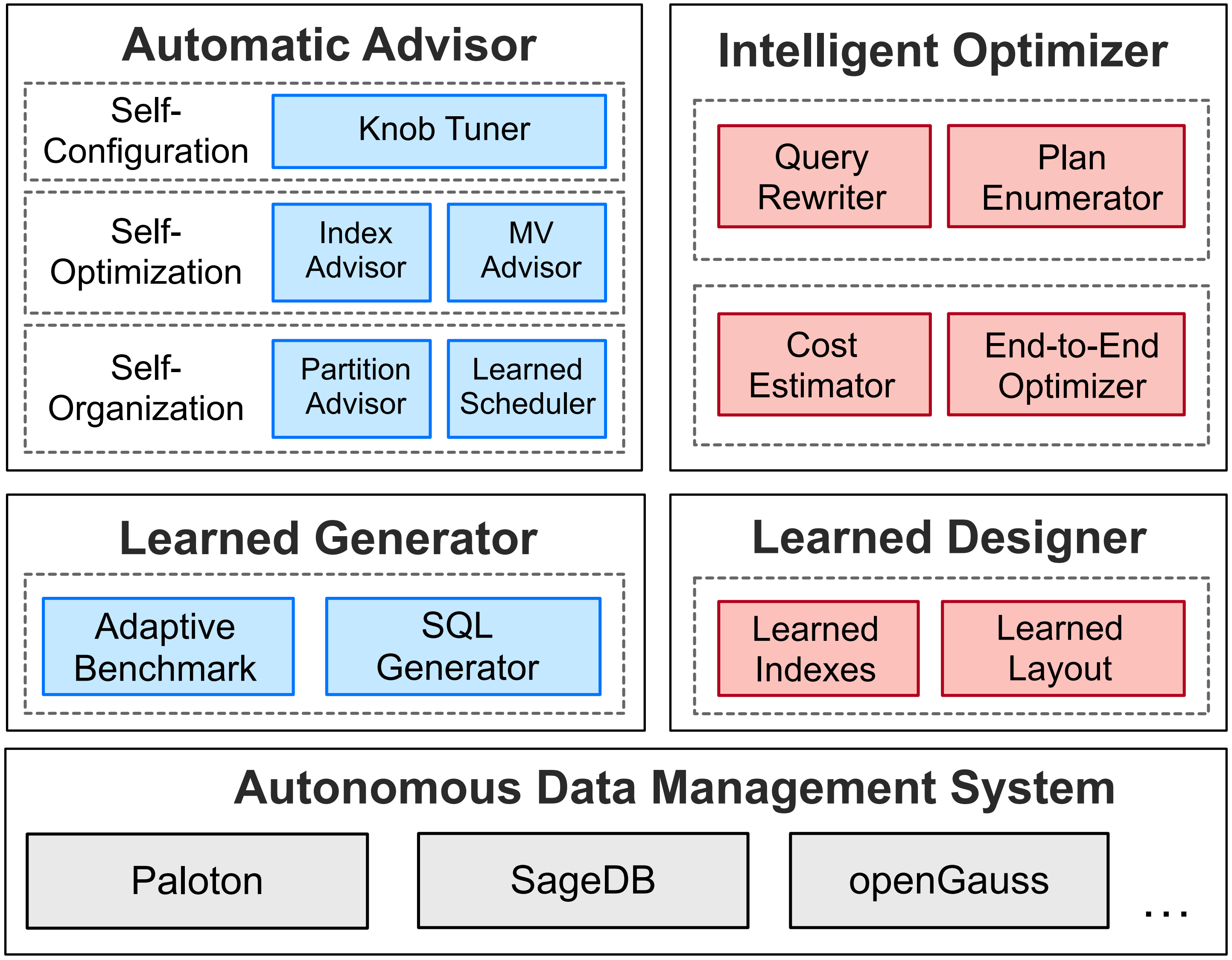
- **Cost Saving: Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency
- **High SLAs: Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance
- **Adaptivity: Empirical → Data-Driven**
  - Learned Index: ↑ Data access efficiency
  - Learned Layout: ↑ Data manipulation efficiency

Challenges

- **Feature Selection:** Pick relevant features from numerous query / database / os metrics ;
- **Model Selection:** Design ML models to solve different database problems;
- **Diverse Targets:** Meet the SLA requirements under different scenarios;
- **Training Data**
- **Adaptivity**

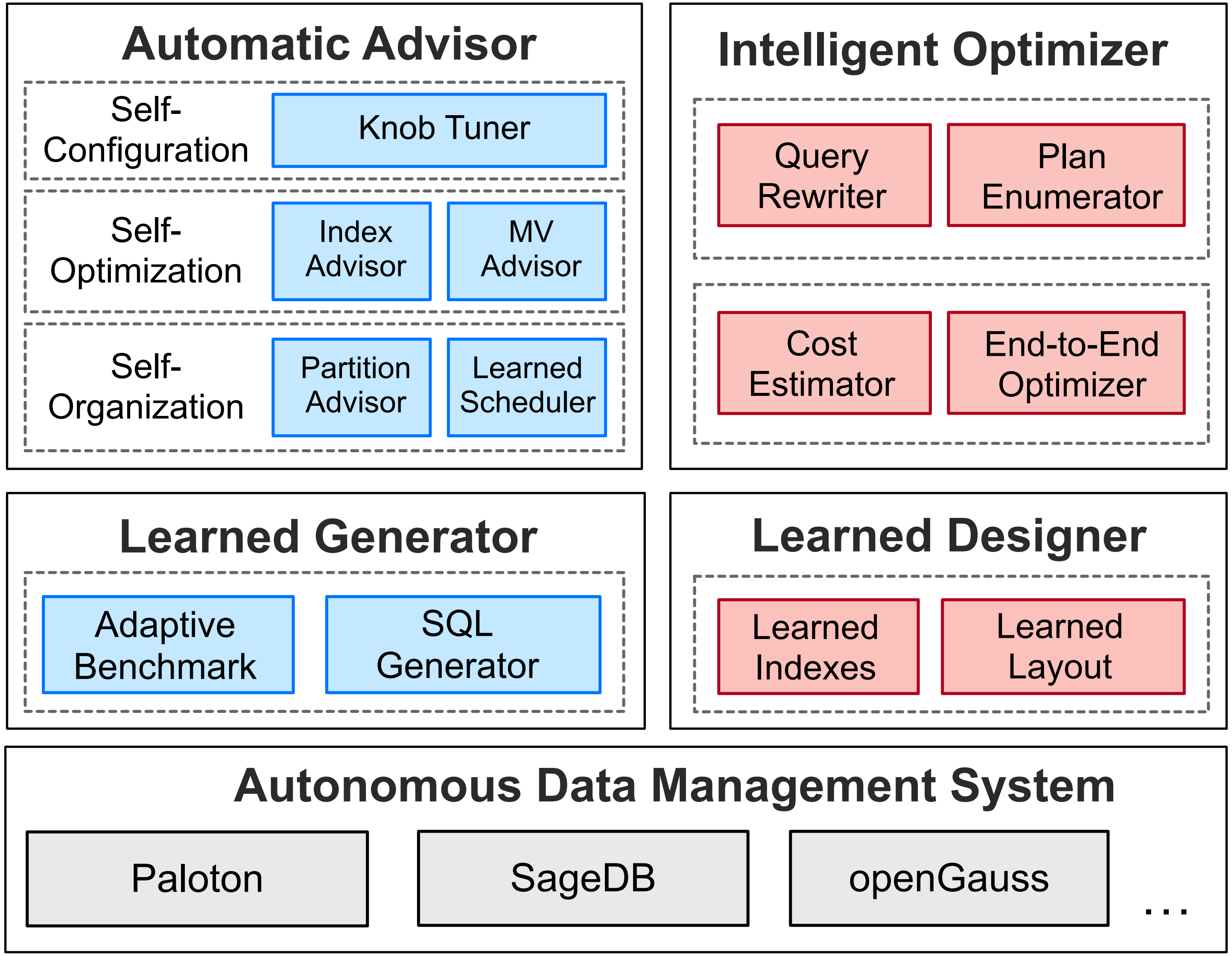
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Advisor: Technique Development

- **Performance (tuning quality, overhead, benefit estimation)**
- **Adaptivity (queries/codes, datasets, instances)**

Knob Tuner

Utilize Historical Data
- Gaussian Process [sigmod, 2017]

Tuning Effect Estimation
- Deep Learning [vldb, 2019]

Optimize Tuning Quality
- Reinforcement Learning [sigmod, 2019] [vldb, 2019]

Reduce Tuning Overhead
- Delayed RL [vldb, 2021]
- DB Manual , [sigmod, 2022]

Improve Adaptivity
- Meta Learning [sigmod, 2021]

Spark Tuning
- Contextual GP [sigmod, 2020]
- Adaptive Learning [icde, 2022]

Index Advisor

Optimize Index Estimation
- Deep Neural Networks [sigmod, 2019]

Improve Index Quality
- Reinforcement Learning [cikm, 2020] [icde, 2021]

Support Index Update
- Monte Carlo Tree Search [icde, 2022]

View Advisor

Optimize MV Selection
- Reinforcement Learning [icde, 2020]

Optimize MV Estimation
- Encoder-Decoder + RL [icde, 2021]

Automatic Knob Tuning

Motivation

Large number of configuration knobs

- Total > 400
- 10-15 are most vital for any workloads

Knobs control nearly every aspect and have complex correlations

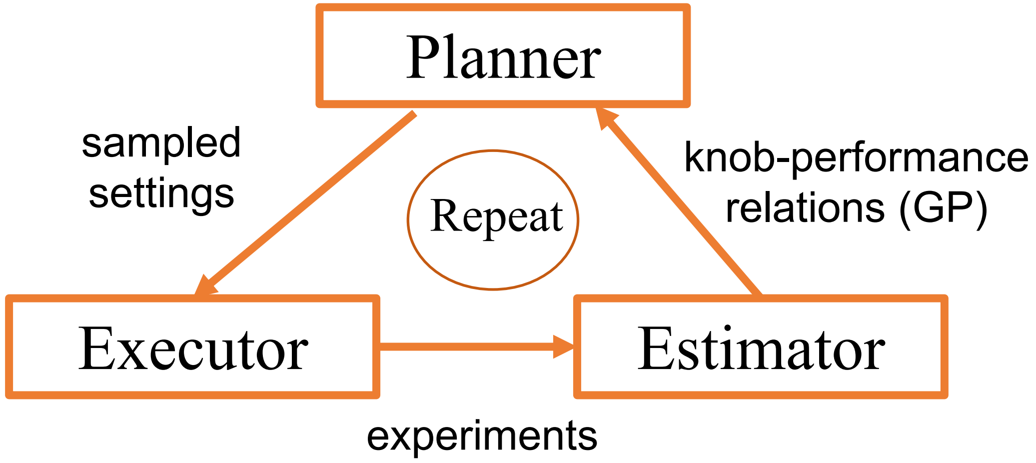
- The relations are non-linear
- One-knob-at-a-time is inefficient

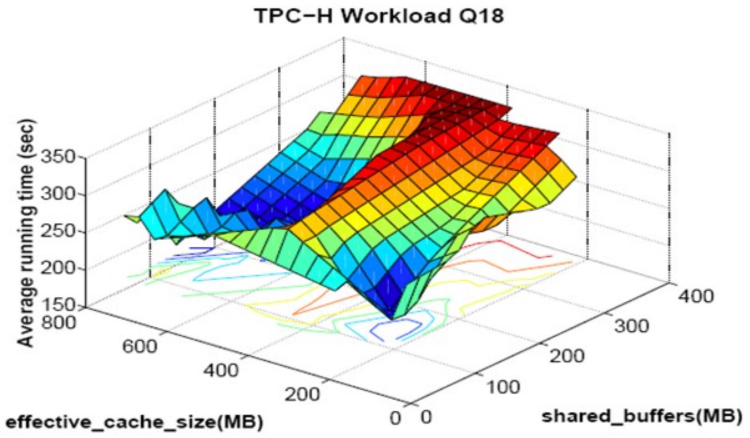
Hi, list. I've just upgraded pgsql from 8.3 to 8.4. I've used pgtune before and everything worked fine for me. And now i have ~93% cpu load. Here's changed values of config:

default_statistics_target = 50
maintenance_work_mem = 1GB
constraint_exclusion = on
checkpoint_completion_target = 0.9
effective_cache_size = 22GB
work_mem = 192MB
wal_buffers = 8MB
checkpoint_segments = 16
shared_buffers = 7680MB
max_connections = 80

Traditional Knob Tuning Method

- **Sampling-based: Explore knob-performance relations**
  - **Planner:** <u>Adaptively sample</u> some knob settings
  - **Executor:** <u>Get the performance</u> of sampled settings by running workloads
  - **Estimator:** <u>Predict knob-performance relations</u> with Gaussian Process;
  - **Termination:** Terminate if arriving time limit; otherwise repeat above steps





knob-performance relations

Songyun Duan, Vamsidhar Thummala, Shivnath Babu. Tuning Database Configuration Parameters with iTuned. VLDB, 2009.

Problems in Traditional Knob Tuning

- **Challenges**
  - **Sampling configurations from scratch is inefficient**
    - Utilize historical configuration data
  - **Knob-performance relations are extremely complex**
    - Advanced ML techniques (depending on scenarios)
  - **Important configuration features are not utilized**
    - Inner metrics; query features; data features

Knob Tuner: Technique Development

- **Performance (tuning quality, overhead, benefit estimation)**
- **Adaptivity (queries/codes, datasets, instances)**

Knob Tuner

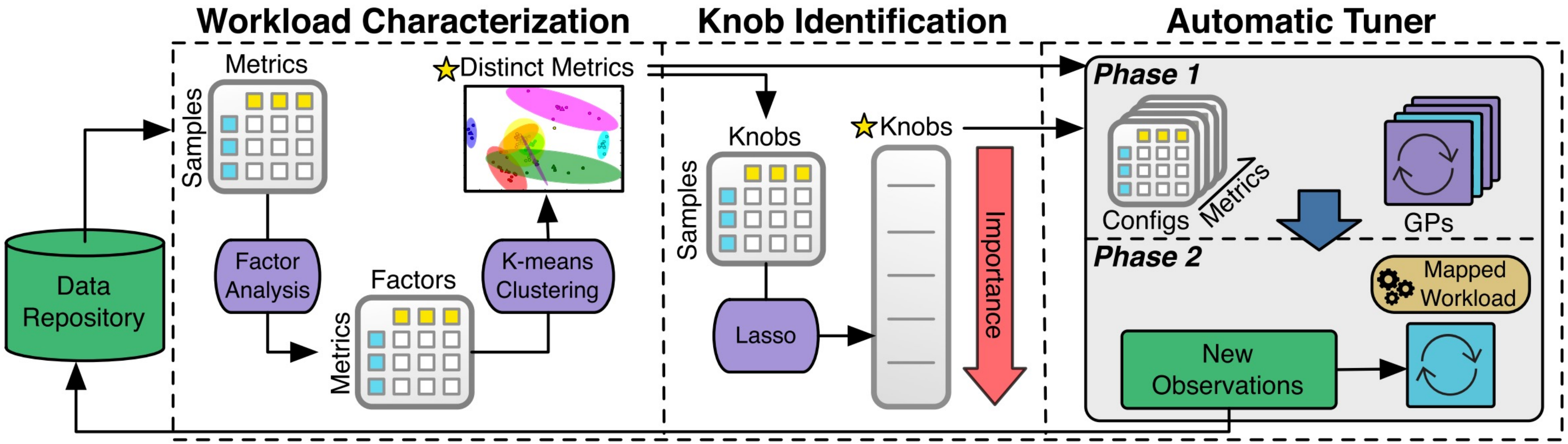
Utilize Historical Data → **Tuning Effect Estimation** → **Optimize Tuning Quality** → **Reduce Tuning Overhead** → **Improve Adaptivity** → **Spark Tuning**

- **Utilize Historical Data**
  - Gaussian Process [sigmod, 2017]
- **Tuning Effect Estimation**
  - Deep Learning [vldb, 2019]
- **Optimize Tuning Quality**
  - Reinforcement Learning [sigmod, 2019] [vldb, 2019]
- **Reduce Tuning Overhead**
  - Delayed RL [vldb, 2021]
  - DB Manual , [sigmod, 2022]
- **Improve Adaptivity**
  - Meta Learning [sigmod, 2021]
- **Spark Tuning**
  - Contextual GP [sigmod, 2020]
  - Adaptive Learning [icde, 2022]

Knob Tuner: Utilize Historical Data

☐ Automatically tune knobs with numerous historical data

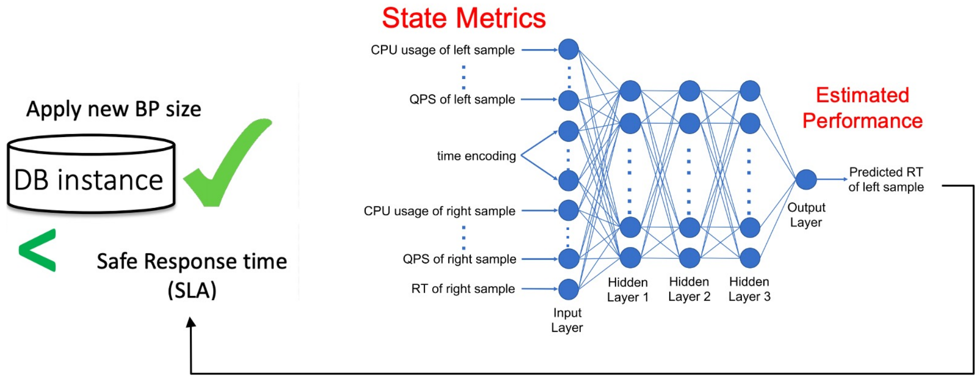
- Characterize workloads with runtime metrics (e.g., #-read-page, #-write-page)
- Identify important knobs (rank knobs through knob-performance sampling)
- Generate workload-to-identified-knob-settings correlations (data repository)
- Given a workload, compute a mapped workload via metric similarity, use corresponding knob settings to initialize GP, explore more settings to get better performance



Dana Van Aken, Andrew Pavlo, et al. Automatic Database Management System Tuning Trough Large-scale Machine Learning. SIGMOD, 2017.

Knob Tuner: Tuning Effect Estimation

- **Motivation: Expensive to run workloads to evaluate tuning effects**
- **Basic Idea: Estimate tuning effects without running workloads**
- **Challenge: Many metrics affect the performance**
- **Solution:**
  - Collect DB metrics: logical-read, QPS, CPU usage, response time;
  - Initialize a buffer size using historical workloads with similar metrics;
  - Design a neural network to estimate the response time as tuning feedback;
  - Greedily reduce the initialized buffer size until arriving safe response time.

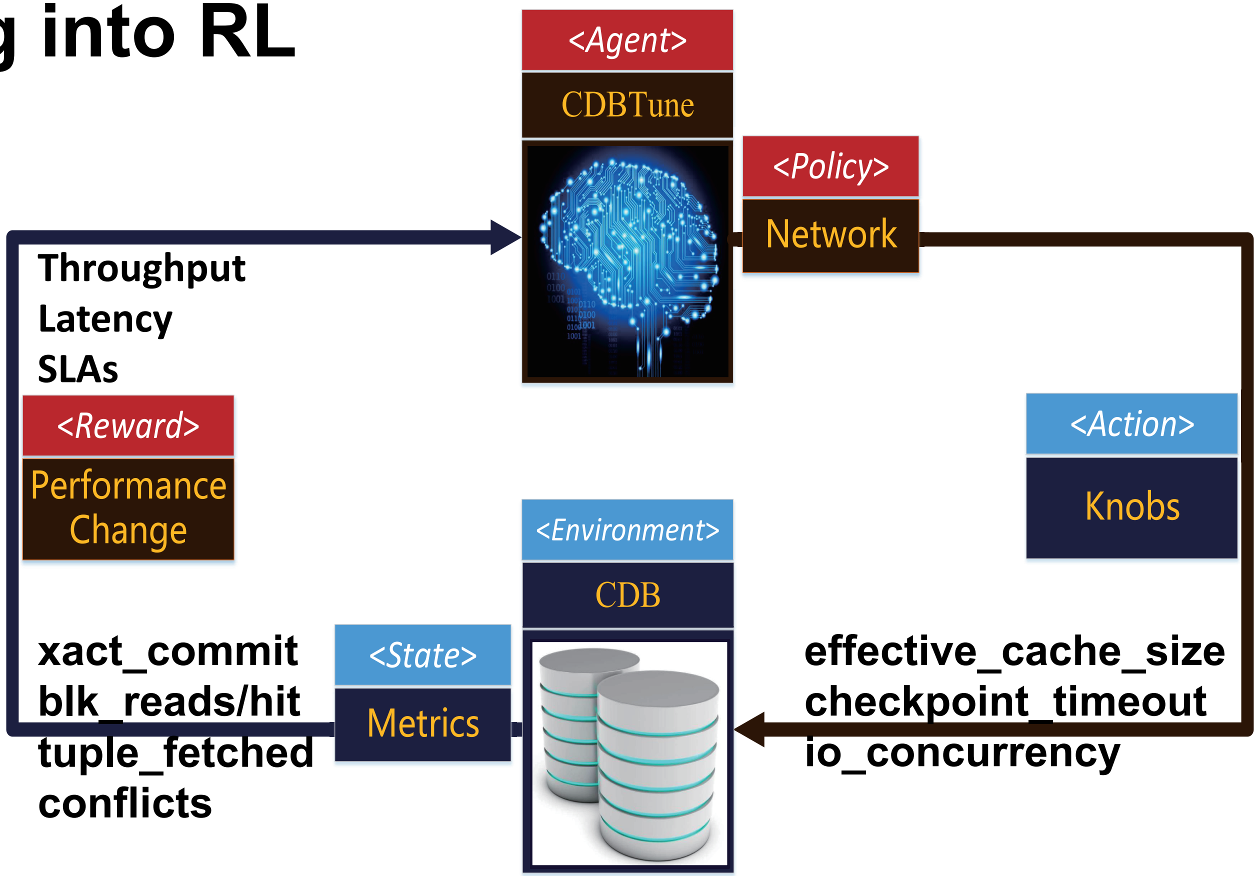


J. Tan, T. Zhang, F. Li, et al. iBTune: Individualized Buffer Tuning for Large-Scale Cloud Databases. VLDB 2019.

Knob Tuner: Optimize Tuning Quality

- Motivation: Traditional methods fall into local optimum
- Basic Idea: Use reinforcement learning (exploration-exploitation)
- Challenge: Map knob tuning into RL
- Solution

RL	CDBTune
Agent	The tuning system
Environment	DB instance
State	Internal metrics
Reward	Performance change
Action	Knob configuration
Policy	Deep neural network



Knob Tuner: Optimize Tuning Quality

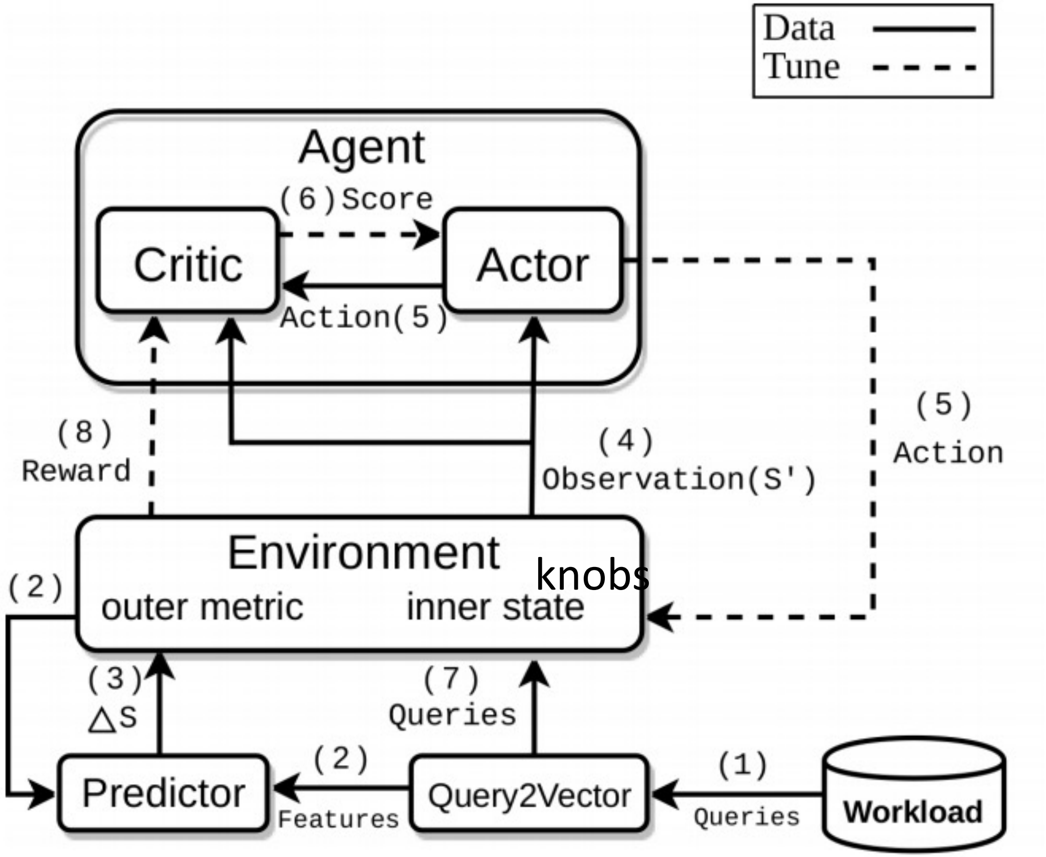
☐ Select proper RL models

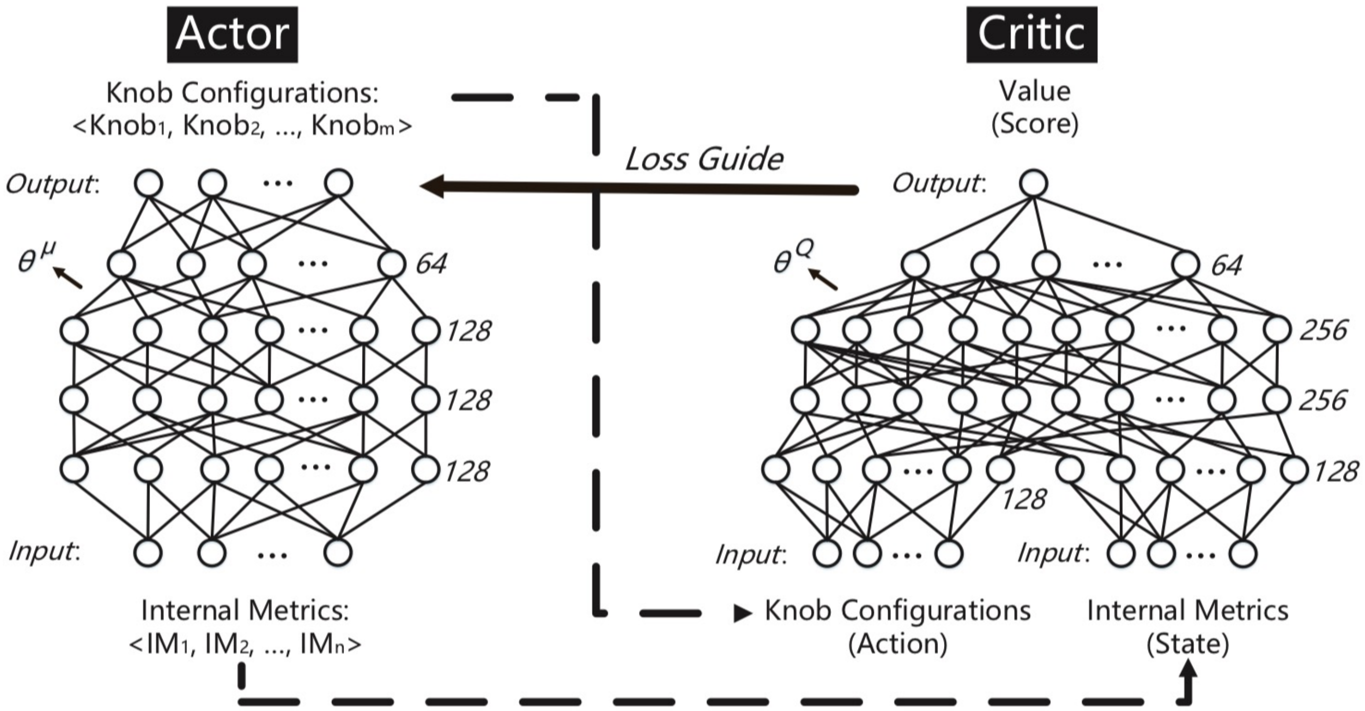
- **Many continuous system metrics and knobs →**
  - **Value-based method (DQN)** **Discrete Action ✕**
    - Replace the Q-table with a neural network
    - **Input:** state metrics; **Output:** Q-values for all the actions
  - **Policy-based method (DDPG)** **Continuous State/Action ✓**
    - (**actor**) Parameterized policy function: $a_t = \mu(s_t|\theta^{\mu})$
    - (**critic**) Score specific action and state: $Q(s_t, a_t|\theta^{Q})$

Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li et al. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. SIGMOD, 2019.

Knob Tuner: Optimize Tuning Quality

- **Select high performance settings with RL** **(QTune as an Example)**

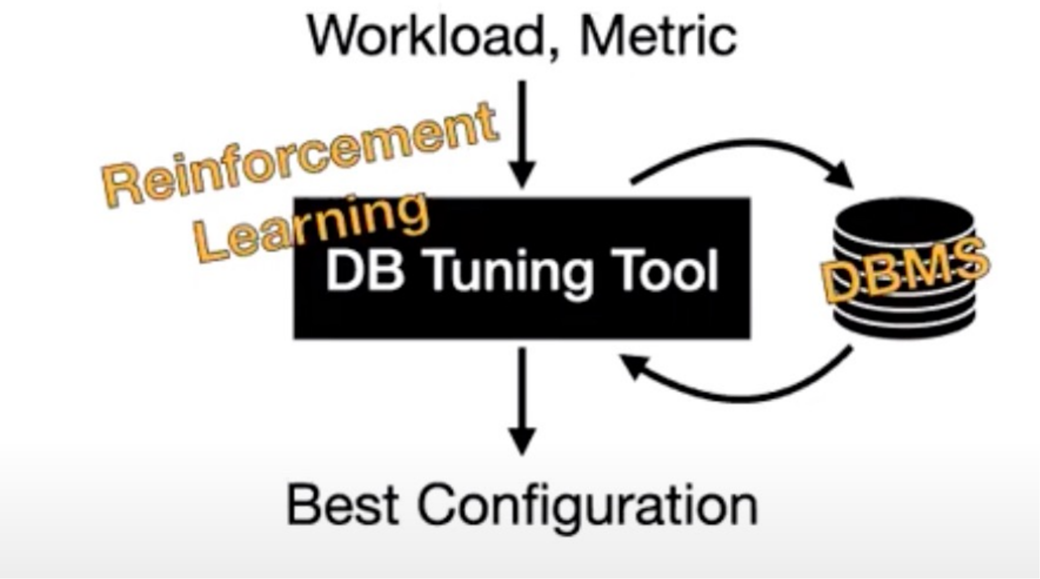




Guoliang Li, Xuanhe Zhou, Shifu Li, Bo Gao. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. VLDB 2019.

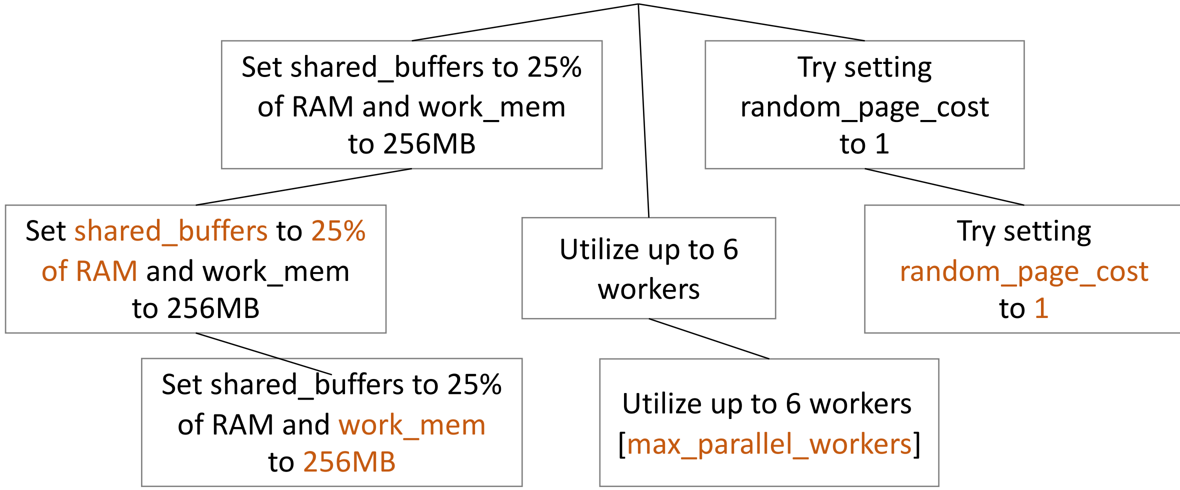
Knob Tuner: Reduce Tuning Overhead

- Problems in RL
  - Significant Tuning Overhead
  - Require DBAs (select knobs, select knob ranges)



- Tuning hints from manual

dba.stackexchange.com



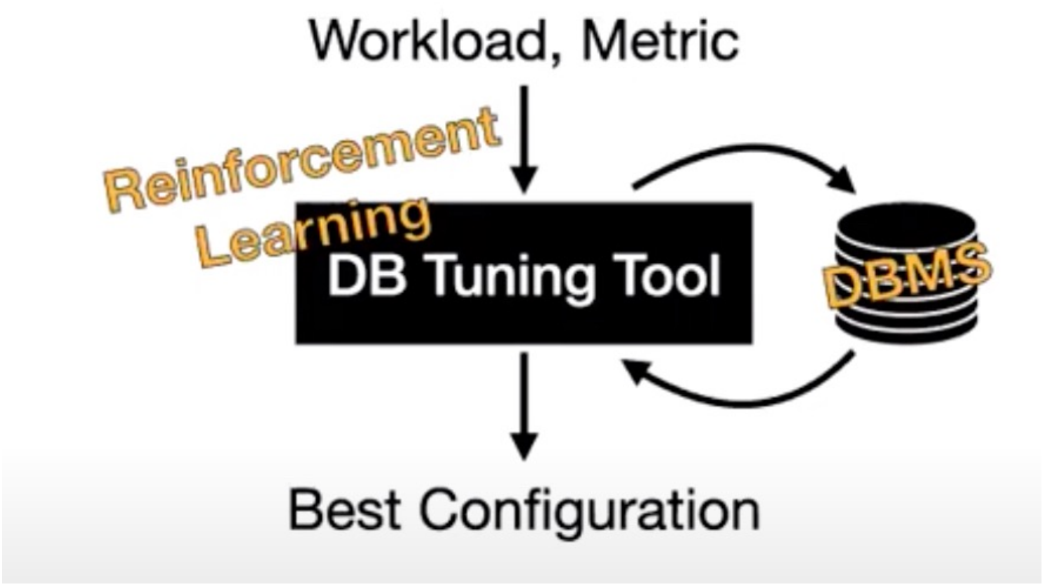
Extract hints from manual

$$Parameter = Value\ [*\ System\ Property][*\ Constant]$$

Given in Text RAM/Disk/Cores

Knob Tuner: Reduce Tuning Overhead

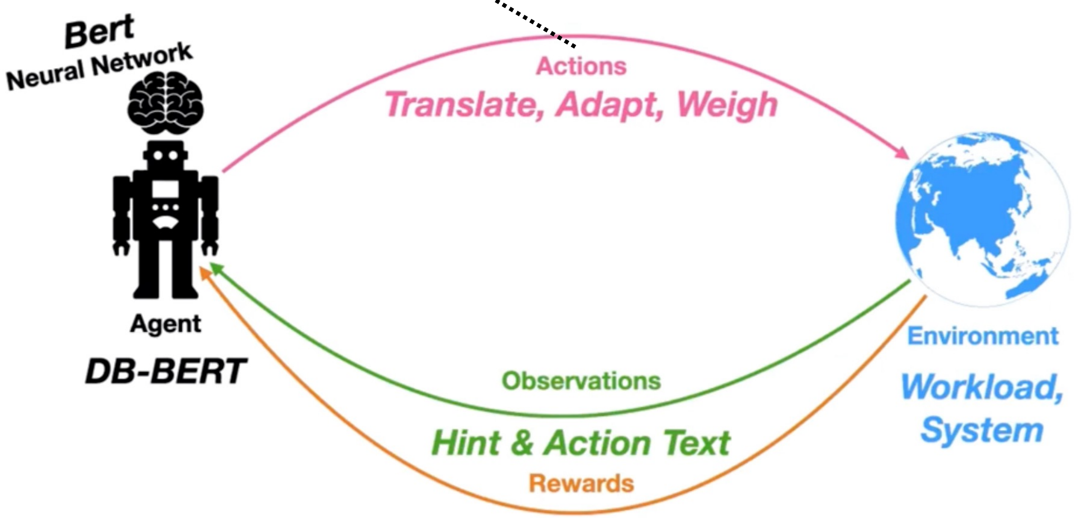
- **Problems in RL**
  - Significant Tuning Overhead
  - Require DBAs (select knobs, select knob ranges)



- **Tuning hints from texts**

Apply collected hints with reinforcement learning

$$Parameter = Value\ [*\ System\ Property][*\ Constant]$$

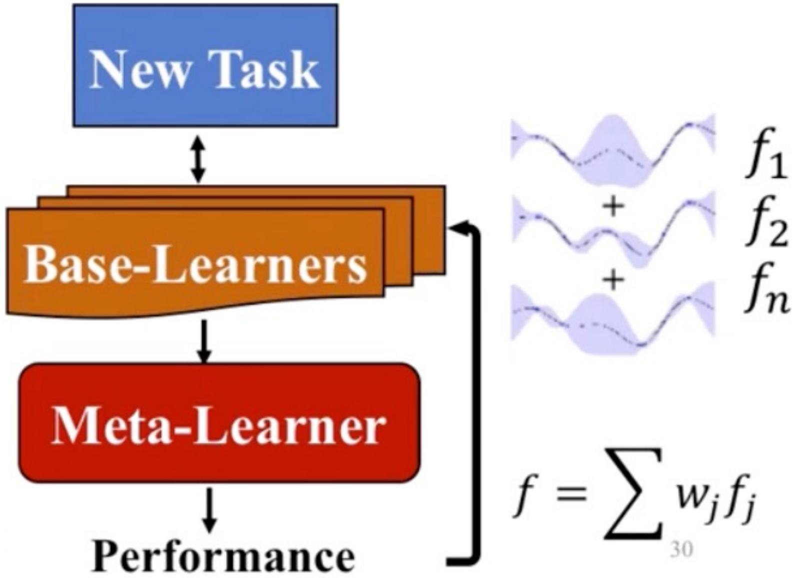


Immanuel Trummer. DB-BERT: a Database Tuning Tool that" Reads the Manual". SIGMOD, 2022.

Knob Tuner: Improve Tuning Adaptivity

□ Historical learned models are hard to migrate to new scenarios

- **Characterize the common features of workloads**
  - Reserved words in the SQLs
- **Cluster similar historical workloads**
  - Cluster with random forest and learn a base learner for each workload cluster
- **Migrate to new workloads with meta learning**
  - Given a workload, generate meta-learner based on the weighted sum of the base learners;
  - Fine-tune the meta-learner by running workload;
  - Recommend promising knobs with meta-learner.



Xinyi Zhang, Hong Wu, and et al. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. SIGMOD, 2021.

Knob Tuner: Spark Tuning

- **Spark tuning needs to consider knobs at different levels**
  - **Empirically initialize knob values at resource/APP/VM levels**

E.g., Memory Efficiency: $q_2^{\mathbf{x}} = \dfrac{M_i + m_c}{\min(m_o^{\mathbf{x}}, m_c^{\mathbf{x}})}$

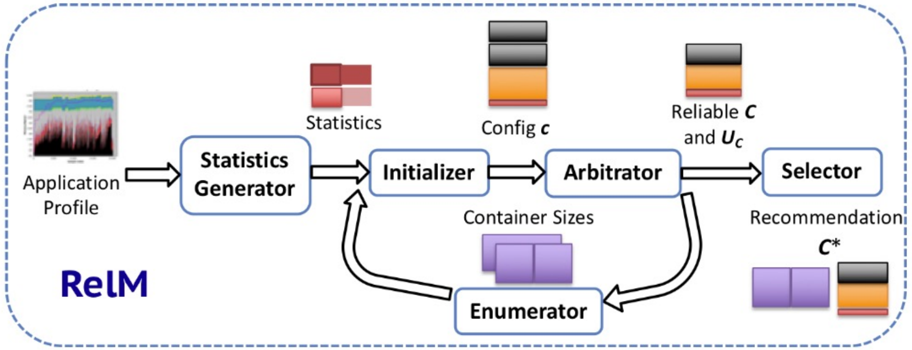
x: Tested knob setting
M_i: Code overhead value
m_c: Required cache storage
m_o: GC settings

- **Interpretable**
- **Easy to migrate**

  - **Guided Gaussian Process**

(1) Input both the execution statistics and <u>initialized knob values</u>;

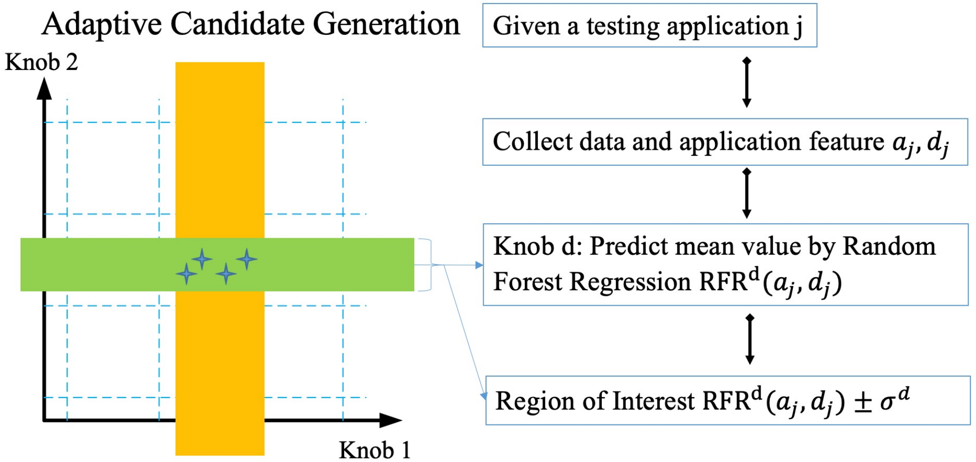
(2) Use GP to fit existing tuning data.

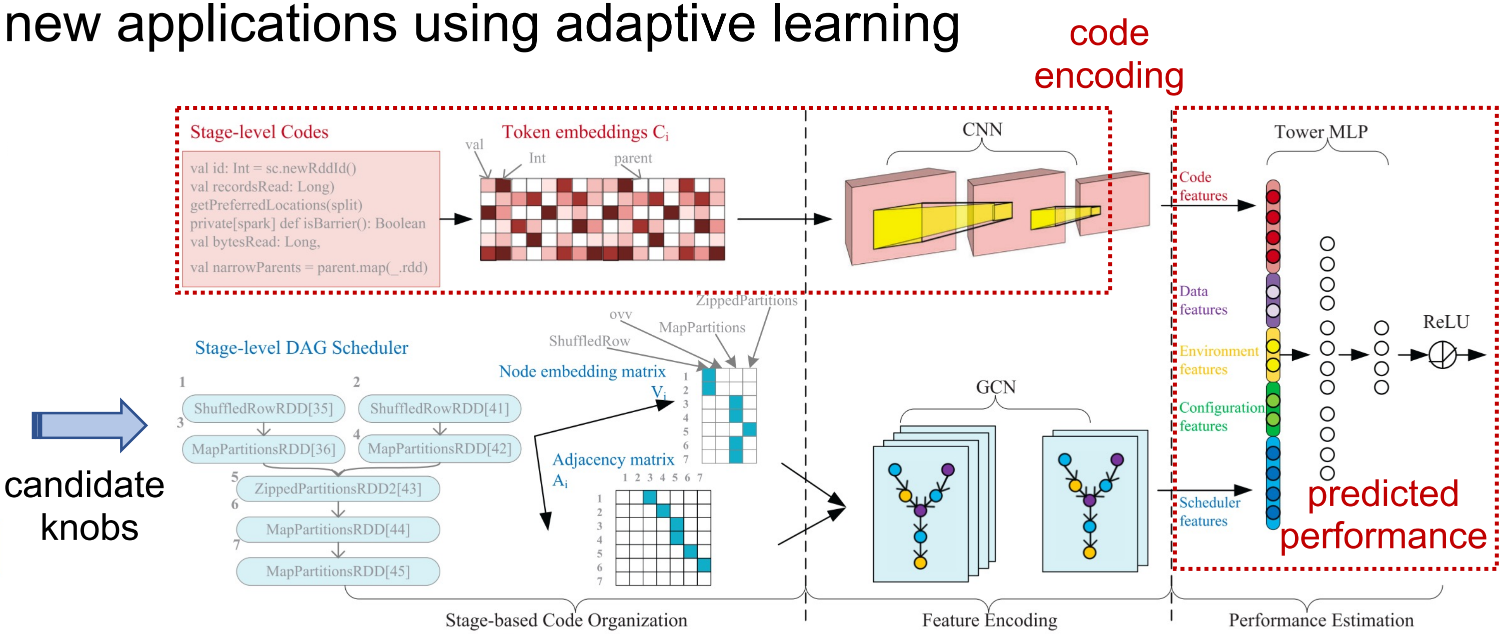


Mayuresh Kunjir, Shivnath Babu. Black or White? How to Develop an AutoTuner for Memory-based Analytics. SIGMOD 2020.

Knob Tuner: Spark Tuning

- **Spark code involves complex semantics, and it is expensive to migrate tuning models across applications**
  - Sample candidate knob settings based on the data and code features;
  - Conduct code instrumentation to enrich code tokens; then encode the code with CNN;
  - Predict the tuning performance (NECS model) with *encoded code*, *data*, *knob*, DAG features;
  - Generalize the NECS model to new applications using adaptive learning





Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, Guoliang Li. Adaptive Code Learning for Spark Configuration Tuning. ICDE 2022.

Summarization of Learned Knob Tuning

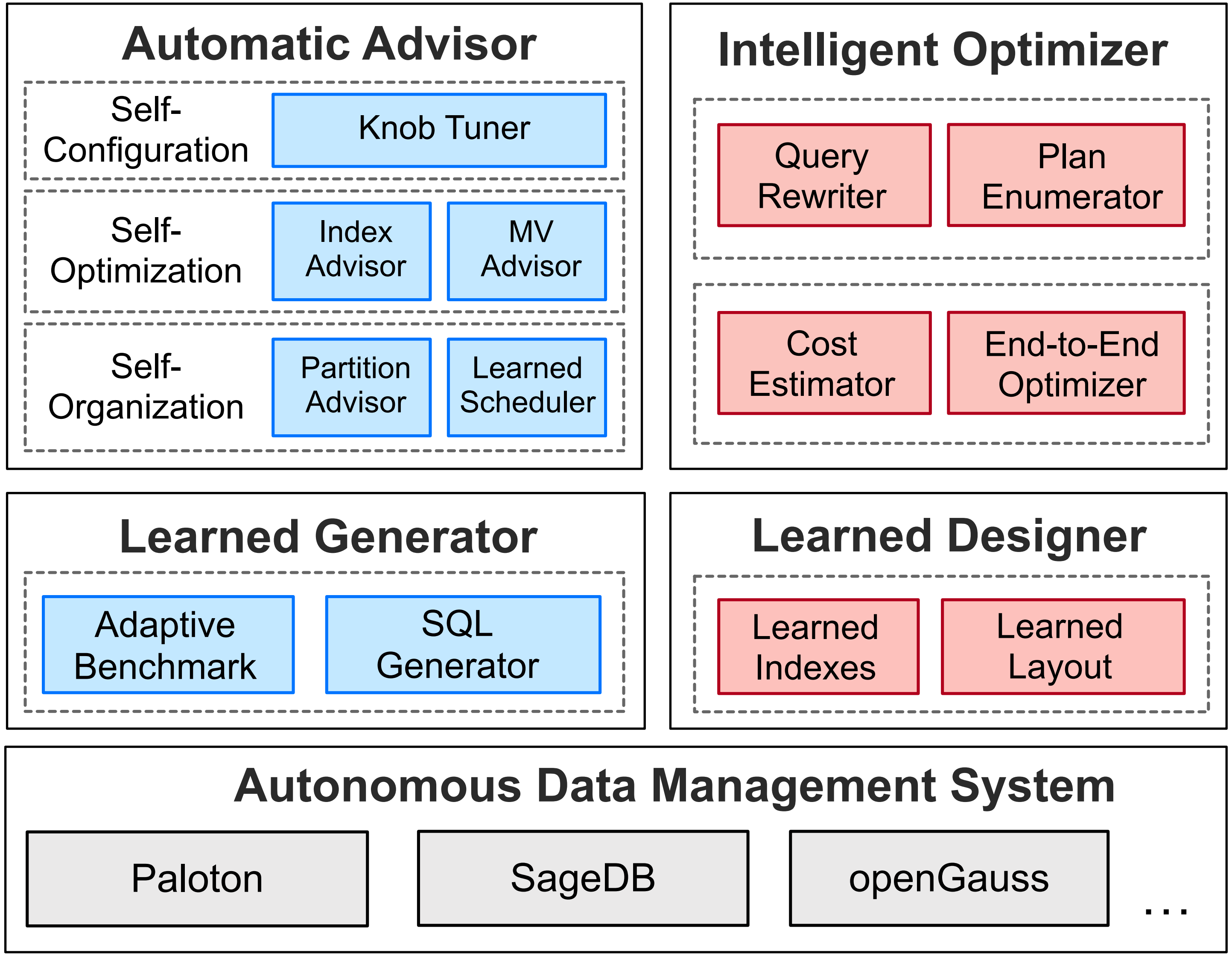
	Quality	Overhead	Training Data	Adaptive
Gaussian Process	✓	--	✓✓	✓
Deep Learning	✓	✓	✓✓	✓
Reinforcement Learning	✓✓	--	No Prepared Data	✓
Manual Learning	✓	--	✓✓✓	✓✓
Meta Learning	✓	✓	✓✓	✓✓
Spark Tuning (Contextual GP)	✓	✓	✓	✓✓
Spark Tuning (MLP+ Adaptive Learning)	✓✓	✓	✓✓	✓✓

Take-aways of Knob Tuning

- **Gradient-based method reduces the tuning complexity by filtering out unimporant features.** However, it heavily relies on training data, and requires other migration techniques to adapt to new scenarios
- **Deep learning method considers both query performance and resource utilization.** And they can significantly reduce the tuning overhead.
- **Reinforcement learning methods take long training time, e.g., hours, from scratch.** However, it only takes minutes to tune the database after well trained and gains relatively good performance.
- **Learning based methods may recommend bad settings when migrated to a new workload.** Hence, it is vital to validate the tuning performance.
- **Open problems:**
  - One tuning model fits multiple databases
  - Natively integrate empirical knowledge

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Index Selection

- **Motivation:**
  - **Indexes are essential for efficient execution**
    - SELECT c_discount from bmsql_customer where c_w_id = 10;
    - CREATE INDEX on bmsql_customer(c_w_id);
  - **Select from numerous indexable columns**
    - Columns have different access frequencies, data distribution
  - **Indexes may cause negative effects**
    - Increase maintenance costs for update/delete operations
    - Performance Degradation ($T_{(hash\text{-}index)} > T_{(full\text{-}table\text{-}scan)}$ $T_{(btree\text{-}index)} > T_{(full\text{-}table\text{-}scan)}$)

Automatic Index Selection

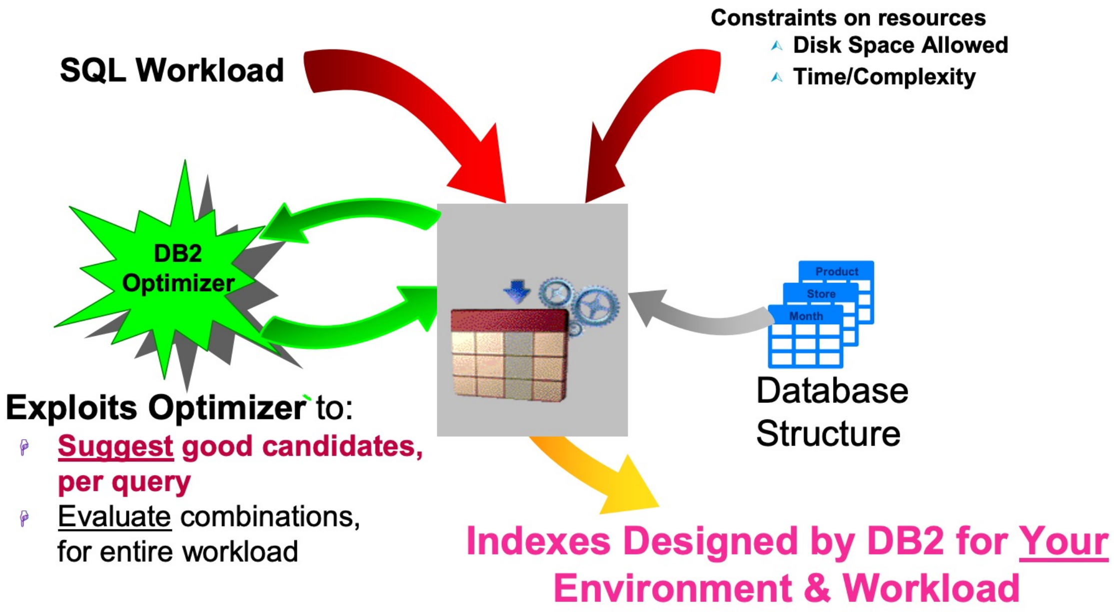
- **Challenge**
  - **The index benefit is hard to evaluate**
    - Multiple evaluation metrics (e.g., index benefit, space cost)
    - Cost estimation by the optimizer is inaccurate
  - **Index selection is an NP-hard problem**
    - The set of candidate index combinations is huge
  - **Index Update**

Automatic Index Selection

- Two sub-problems
  - Index selection
    - Select indexes from a large number of possible combinations to maximize the benefit within a budget
  - Benefit estimation
    - Estimate the benefit of creating an index
      - Cost(q) - Cost(q, index), q is a query

Traditional Index Advisor (Dynamic Programming)

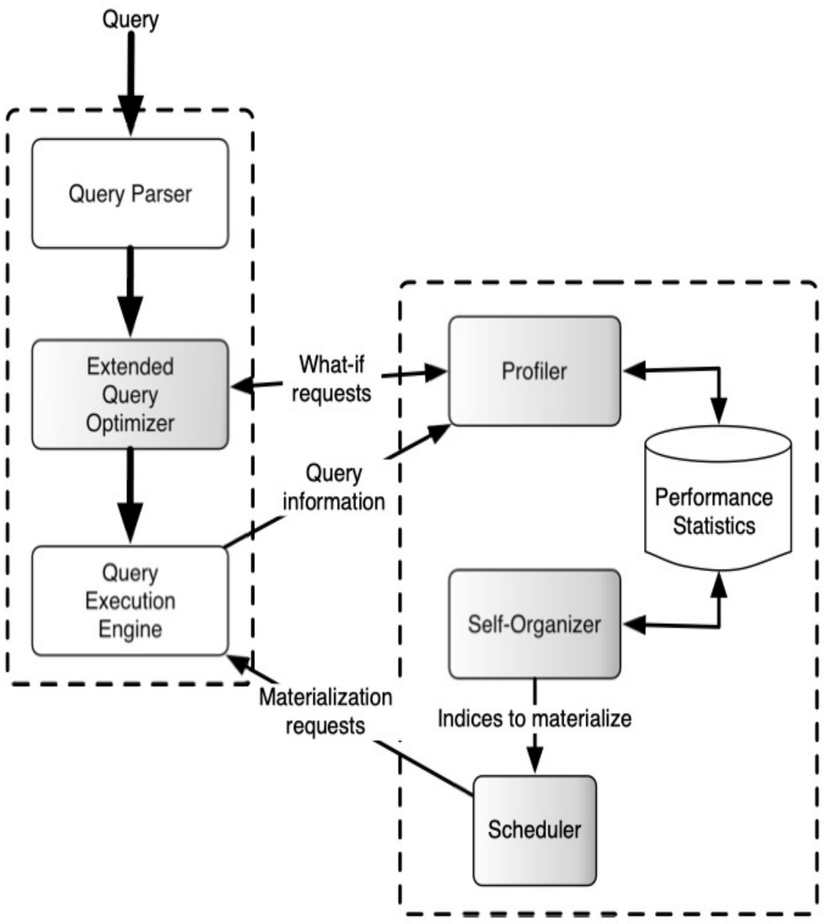
- **Model index selection as a knapsack problem**
  - Candidate index scheme as item
  - Index size as item weight
  - Benefit of the item (optimizer) as value
- **Use DP to select the highest-benefit indexes**



G. Valentin, M. Zuliani, D. C. Zilio, et al. *DB2 advisor: An optimizer smart enough to recommend its own indexes*. In ICDE 2000.

Traditional Index Advisor (what-if estimation)

- **Index selection for dynamic workloads**
  - **Divide a workload into epochs of queries**
  - **Profile candidate indexes for each new query**
    - **Index Benefit:** average latency reduction for the queries within the same epoch (time-series)
    - Estimate the index benefit through a what-if call (assume: *similar queries have similar index benefits*)
    - Update the index set and statistics
  - **Create indexes with highest index benefit at the end of each epoch**



K. Schnaitter, S. Abiteboul, T. Milo and N. Polyzotis. *On-Line Index Selection for Shifting Workloads*. In ICDE 2007.

Index Advisor: Technique Development

- **Performance (tuning quality, overhead, benefit estimation)**
- **Adaptivity (queries/codes, datasets, instances)**

Index Advisor

Optimize Index Estimation → **Improve Index Quality** → **Support Index Update**

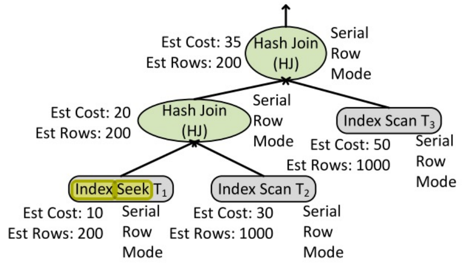
- Deep Neural Networks
[sigmod, 2019]

- Reinforcement Learning
[cikm, 2020] [icde, 2021]

- Monte Carlo Tree Search
[icde, 2022]

Index Advisor: Optimize Index Estimation

- **Critical to estimate index benefits by comparing execution costs of plans with/without created indexes**
  - **Prepare training data:** Workloads + execution feedback from customers
  - **Train the evaluation model:** Predict the index benefits (1: performance gains; 0: no)
  - **Solve Classification Problem:** Use the model to create indexes with performance gains



(a) Example query plan.

EstNodeCost

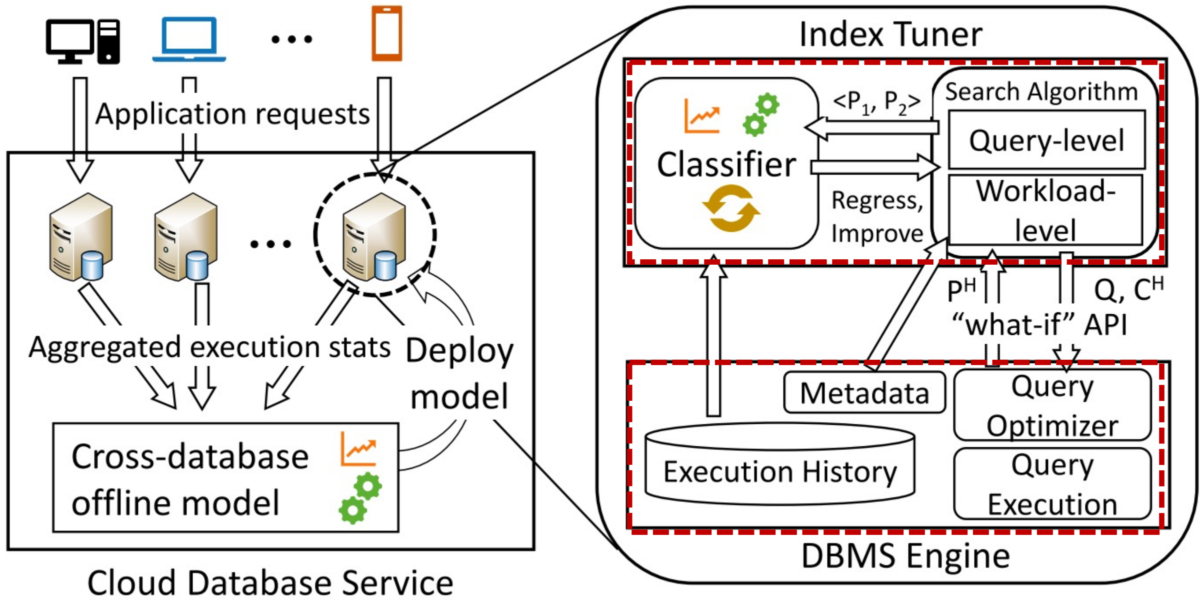
Seek_Row_Serial	10
Scan_Row_Serial	80
HJ_Row_Serial	55
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

...

LeafWeightEstRows WeightedSum

Seek_Row_Serial	200
Scan_Row_Serial	2000
HJ_Row_Serial	4600
NLJ_Row_Serial	0
MJ_Row_Serial	0
...	...

(b) Feature channels for the plan.



Create Indexes via Evaluation Model

Training data Generation

Bailu Ding, Sudipto Das, et al. *AI meets ai: leveraging query executions to improve index recommendations*. In SIGMOD, 2019.

Index Advisor: Optimize Index Selection

- **Motivation: Index selection using reinforcement learning**
- **How to extract candidate indexes?**

  - Extract candidate indexes from query predicates with empirical rules

Rule 1: Construct all single-attribute indexes by using the attributes in J, EQ, RANGE.
Rule 2: When the attributes in O come from the same table, generate the index by using all attributes in O.
Rule 3: If table *a* joins table *b* with multiple attributes, construct indexes by using all join attributes.

- **How to choose from candidate indexes?**

  - Map into *Markov Decision Process* (MDP)

State: Info of current built indexes

Action: Choose an index to build

Reward: Cost reduction ratio after building the index

large state space →

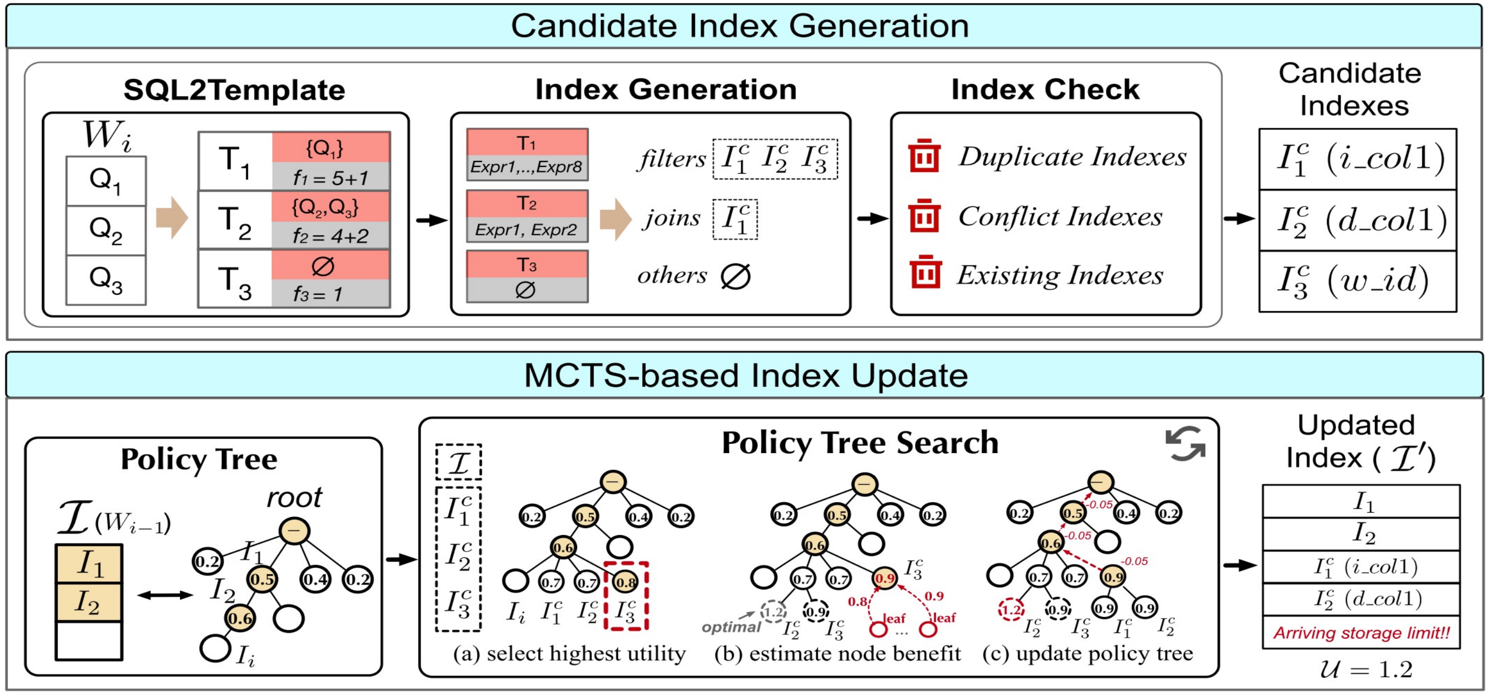
discrete action space →

DQN Model

H. Lan, Z. Bao, Y. Peng. *An Index Advisor Using Deep Reinforcement Learning*. CIKM, 2020.

Index Advisor: Support Index Update

- **Motivation: Indexes need to be updated based on workload changes**
- **Core Idea: Incrementally add/remove indexes with MCTS**
  - Generate candidate indexes based on incoming queries
    - Merge similar queries into templates
    - Extract columns from template predicates
    - Combine columns into candidate indexes
  - Update existing indexes with candidate indexes
    - Initialize a policy tree
    - Explore more beneficial index sets on the tree



Xuanhe Zhou, Luyang Liu, et al. AutoIndex: *An Incremental Index Management System for Dynamic Workloads*. ICDE, 2022.

Summarization of Automatic Index Advisor

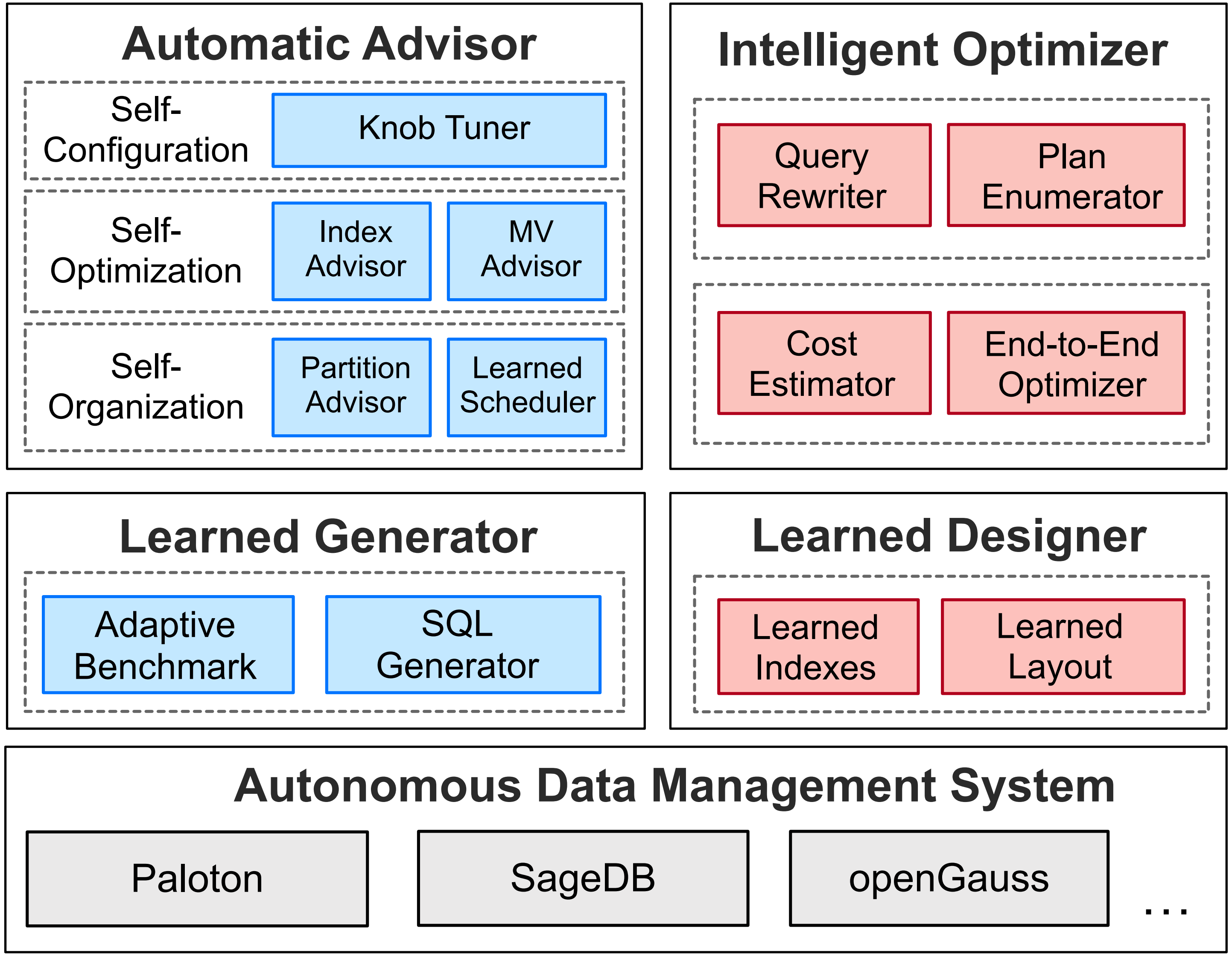
	Optimization Targets	Overhead	Training Data	Adaptive
Deep Learning	Accurate Estimation	numerous data	much	query changes
Reinforcement Learning	High Performance	high computation costs	no prepared Data	query changes
MCTS	High Performance with index update	trade-off (costs, performance)	a few prepared data	query changes

Take-aways of Index Advisor

- **RL-based index selection works takes much time for model training (cold start); while MCTS can gain similar performance and better interpretability (or regret bounds)**
- **Learned estimation models need to be trained periodically for data or workload update**
- **Open problems:**
  - ➢ Benefit prediction for future workload
  - ➢ Cost for future updates

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Challenges in Heuristic MV Selection

- **Estimate** the utility of view candidates:

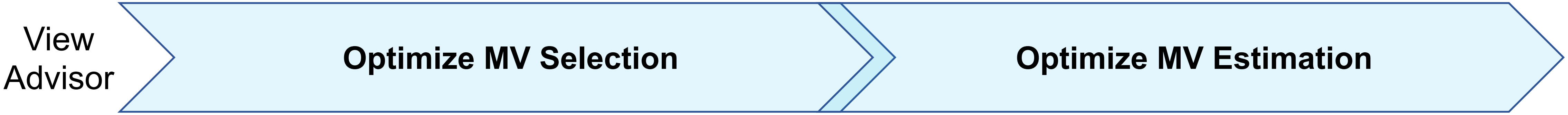
$$B = t_v - t_{v_{scan}}$$

  - Make no sense when MVs change the query plan drastically; Hard to estimate MV update cost
- **Select** views to materialize
  - Greedy/Genetic/other heuristics, Integer Linear Programming
  - Perform poor when the assumption is not satisfied (e.g. MV with higher cost has higher utility)
- **Update** views based on credits
  - A view's credit is the sum of future utility and recreation cost
  - Cause delay to measure and estimate the credit value

D. Zilio, C. Zuzarte, S. Lightstone, W. Ma, et al. Recommending Materialized Views and Indexes with IBM DB2 Design Advisor. ICAC, 2004.

MV Advisor: Technique Development

- **Performance (tuning quality, overhead, benefit estimation)**
- **Adaptivity (queries/codes, datasets, instances)**



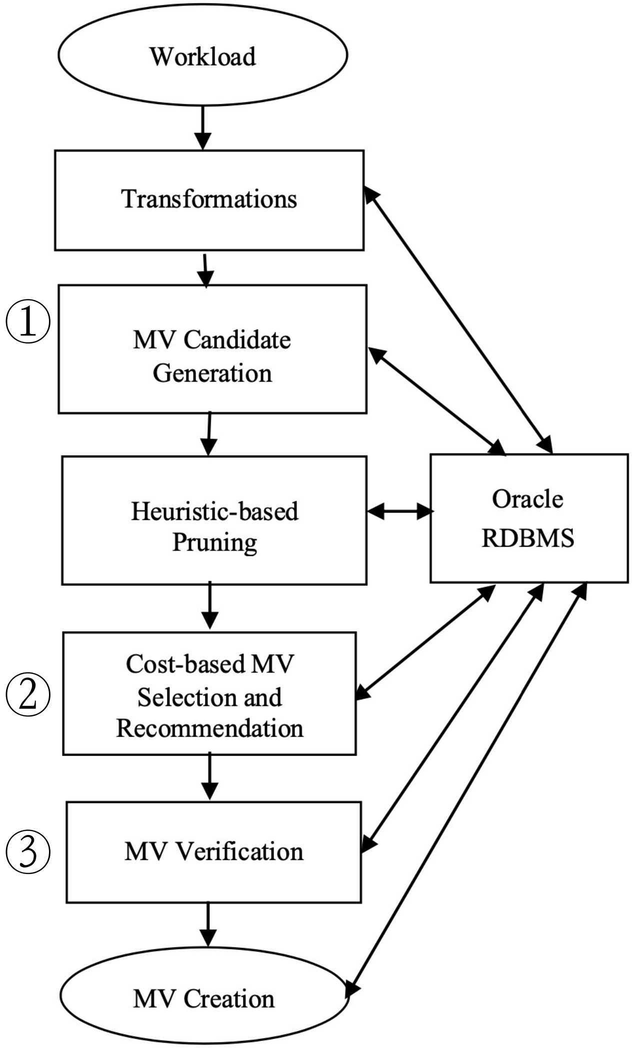
- Reinforcement Learning
[VLDB, 2020]

- Reinforcement Learning
[icde, 2020]

- Encoder-Decoder + RL
[icde, 2021]

MV Advisor: Optimize MV Selection

- **Numerous candidate MVs → Greedily select MVs**
  - ① Generate candidate MVs that balance between conflict queries (merge MVs)
  - ② **Enumerate** queries, MVs, and estimate the query costs with/without the selected MVs
  - ③ Verify the performance of selected MVs
- **MV Update → Predict MV usage frequency with a neural network**

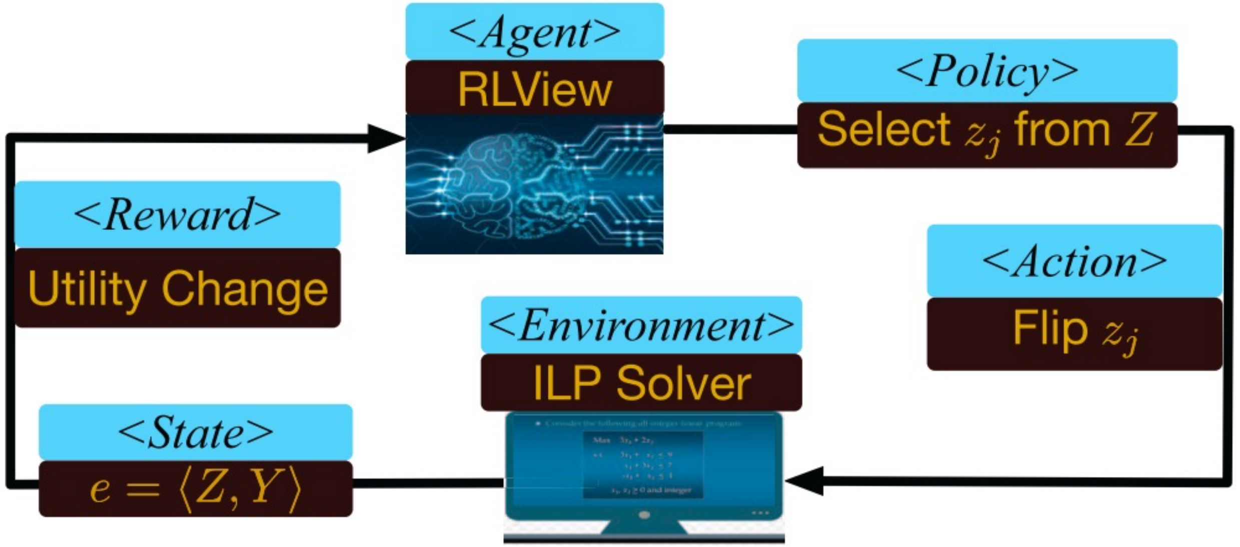


Ahmed, R., Bello, R., Witkowski, A. Kumar. Automated generation of materialized views in Oracle. VLDB 2020.

MV Advisor: Optimize MV Selection

- **Extract candidate MVs from numerous common subqueries**
  - **Cluster equivalent queries and select the least overhead ones as the candidate;**

- **Select optimal candidate MVs with RL (under budget)**
  - **(1) Solve MV Selection with DQN model:**
  - **(2) Estimate the MV benefits with a deep neural network**

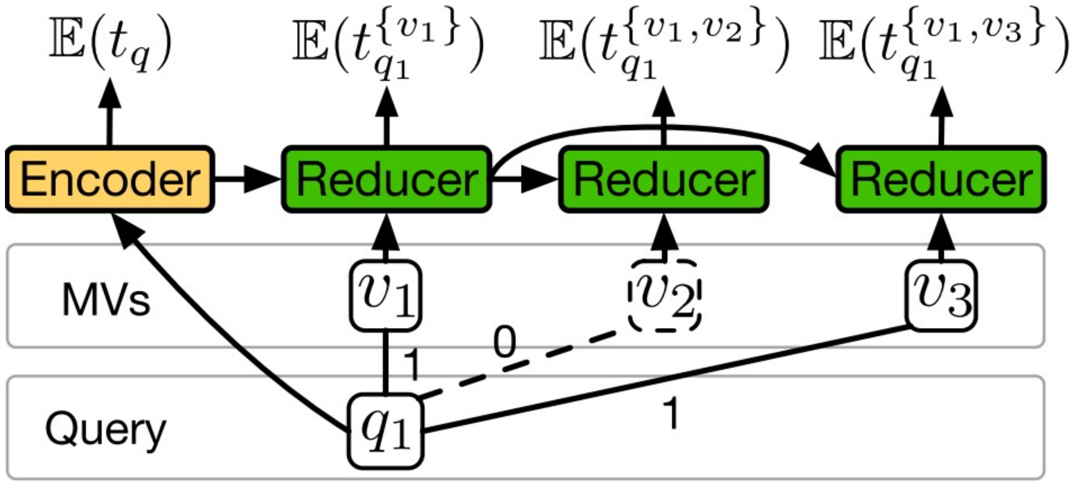


$Z = \{z_j\}$:z_j is a 0/1 variable indicating whether to materialize the subquery s_j
$Y = \{y_{ij}\}$:y_{ij} is a 0/1 variable indicating whether to use the view v_{s_j} for the query q_i

H. Yuan, G. Li, L. Feng, and et al. Automatic view generation with deep learning and reinforcement learning. In ICDE, 2020.

MV Advisor: Optimize MV Estimation

- **Previous MV estimation cannot capture query-MV correlations**
- **Capture query-MV correlations with Encoder-Reducer Model**
  - Generate query-MV pairs (queries can utilize multiple MVs)
  - Estimate the query-MV benefits with encoder-reducer model
    - Encoder-Reducer Model: Encode various number of queries and views with LSTM network, which captures query-MV correlations with *attention*
  - Select optimal MV combinations with reinforcement learning



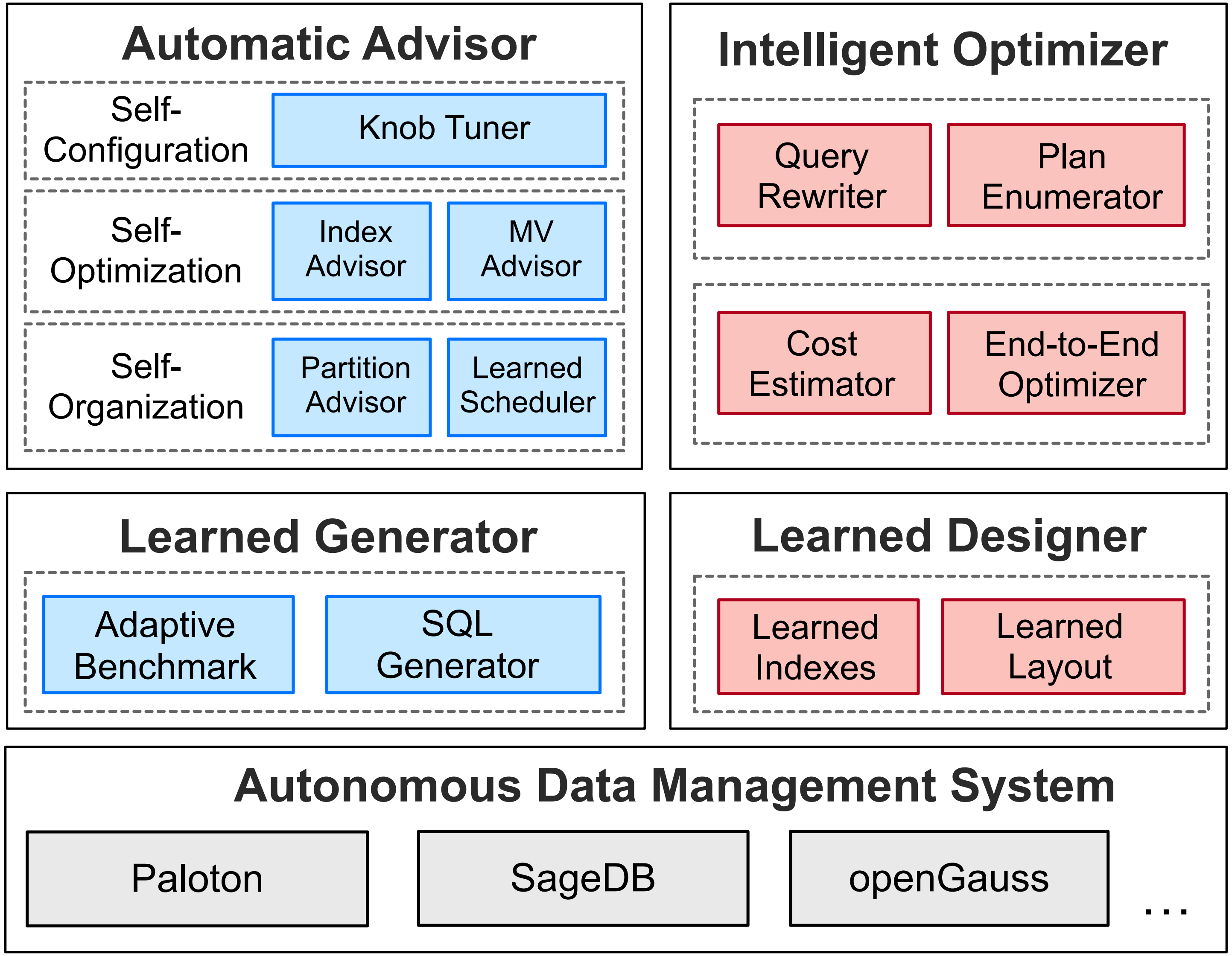
Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In ICDE, 2021.

Take-aways of MV Advisor

- Learned MV selection gains higher performance than heuristic methods
- Learned MV selection works well for read workloads, and cannot efficiently support data update
- Learned MV utility estimation is more accurate than traditional empirical methods
- Learned MV utility estimation is also accurate for multiple-MV optimization

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



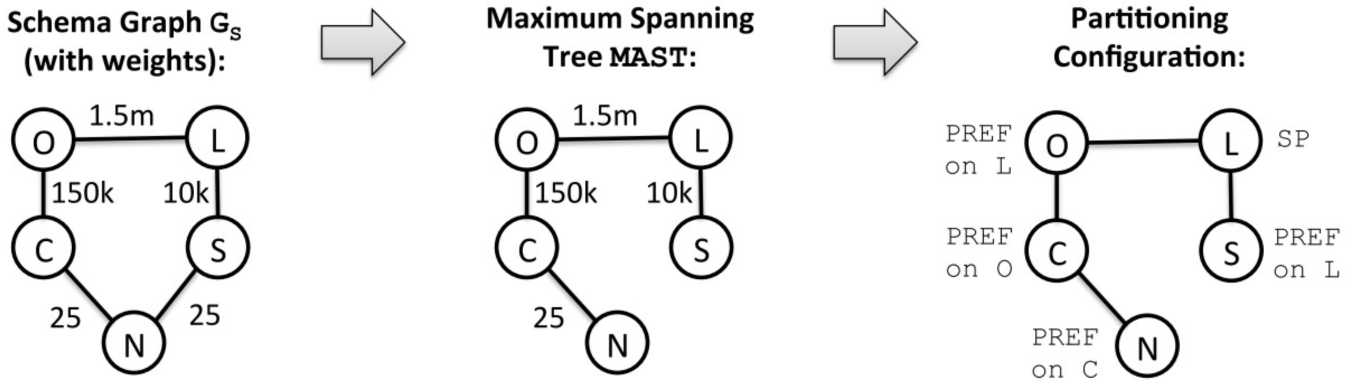
Automatic Database Partition

☐ Motivation:

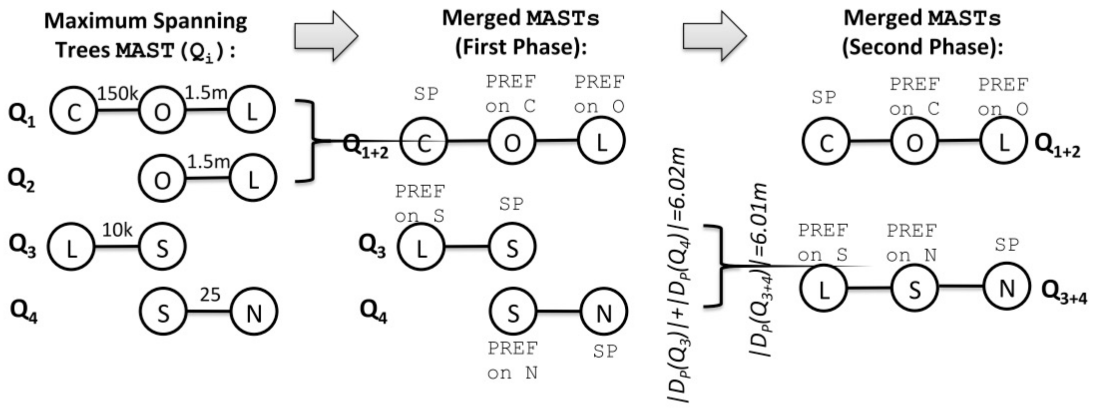
- **A vital component in distributed database**
  - Place partitions on different nodes to speedup queries
  - Trade-off between data balance & access frequency
- **Database partition problem is NP-hard**
  - **Combinatorial problem:** 61 TPC-H columns, 145 query relations, 2.3×10^{18} candidate combinations

Traditional Database Partition Method

- **Select partition keys from foreign-key relations**
  - ↑ **Data-locality: for each query, select partition keys with Maximum spanning tree**



  - ↓ **Data-redundancy: for all the queries, combine selected partition keys and take the optimal combination with DP**



Erfan Zamanian, Carsten Binnig, Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. SIGMOD 2015.

Traditional Database Partition Method

- Combine exact and heuristic algorithms to find good partition strategies
  - The partitioning performance is affected by the join queries →
  - Build a weighted undirected graph, where the nodes are tables and edges are join relations.
  - Key Selection on the graph is a maximum weight matching problem →
  - Provide both exact (i.e., each table uses a column, and turn into a integer programming problem) and heuristic (i.e., select the table columns whose edge weights are maximal) algorithms; and apply the appropriate algorithm under the time budget.

P. Parchas, et al. Fast and Effective Distribution-Key Recommendation for Amazon Redshift. Proc. VLDB Endow, 2020.

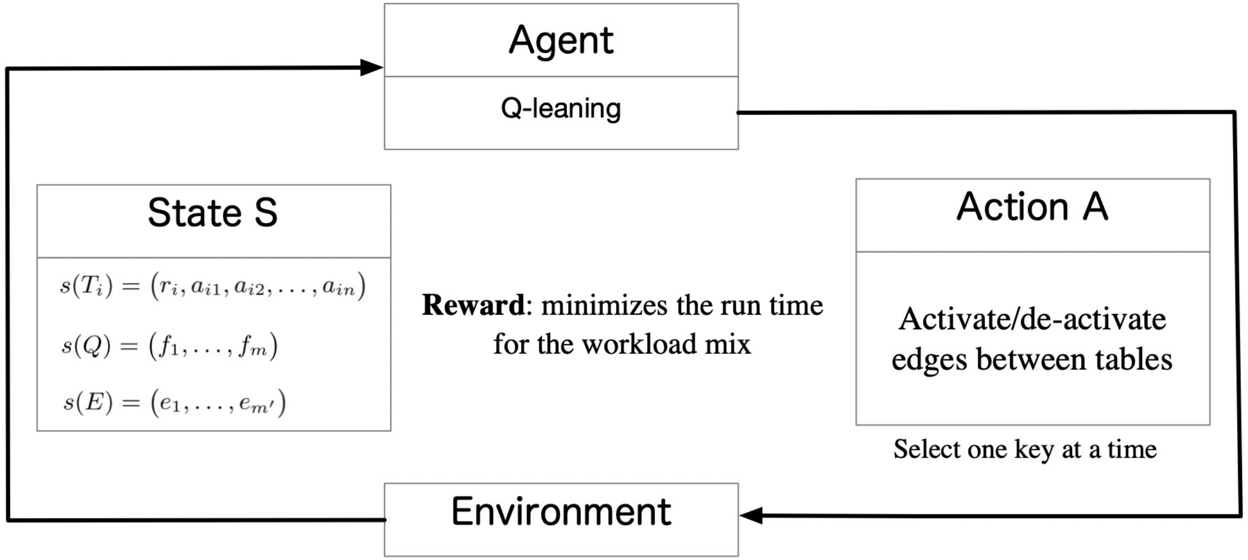
Challenges in Traditional Database Partition

- **Rely on foreign-key relations to select partition keys**
  - Other vital columns may be ignored, and cause sub-optimum
- **Greedily select partition keys without considering the query costs and data distributions**
- **Cannot learn from historical partitioning data**

Erfan Zamanian, Carsten Binnig, Abdallah Salama. Locality-aware Partitioning in Parallel Database Systems. SIGMOD 2015.

DRL for Partition-Key Selection

- Typical OLAP Workloads contain complex and recursive queries
  - State Features: [tables, query frequencies, foreign keys]
- Select from numerous partition-key combinations and support new queries
  - (1) Use DQN to partition or replicate tables;
  - (2) Pretrain a cluster of RL models to support new queries



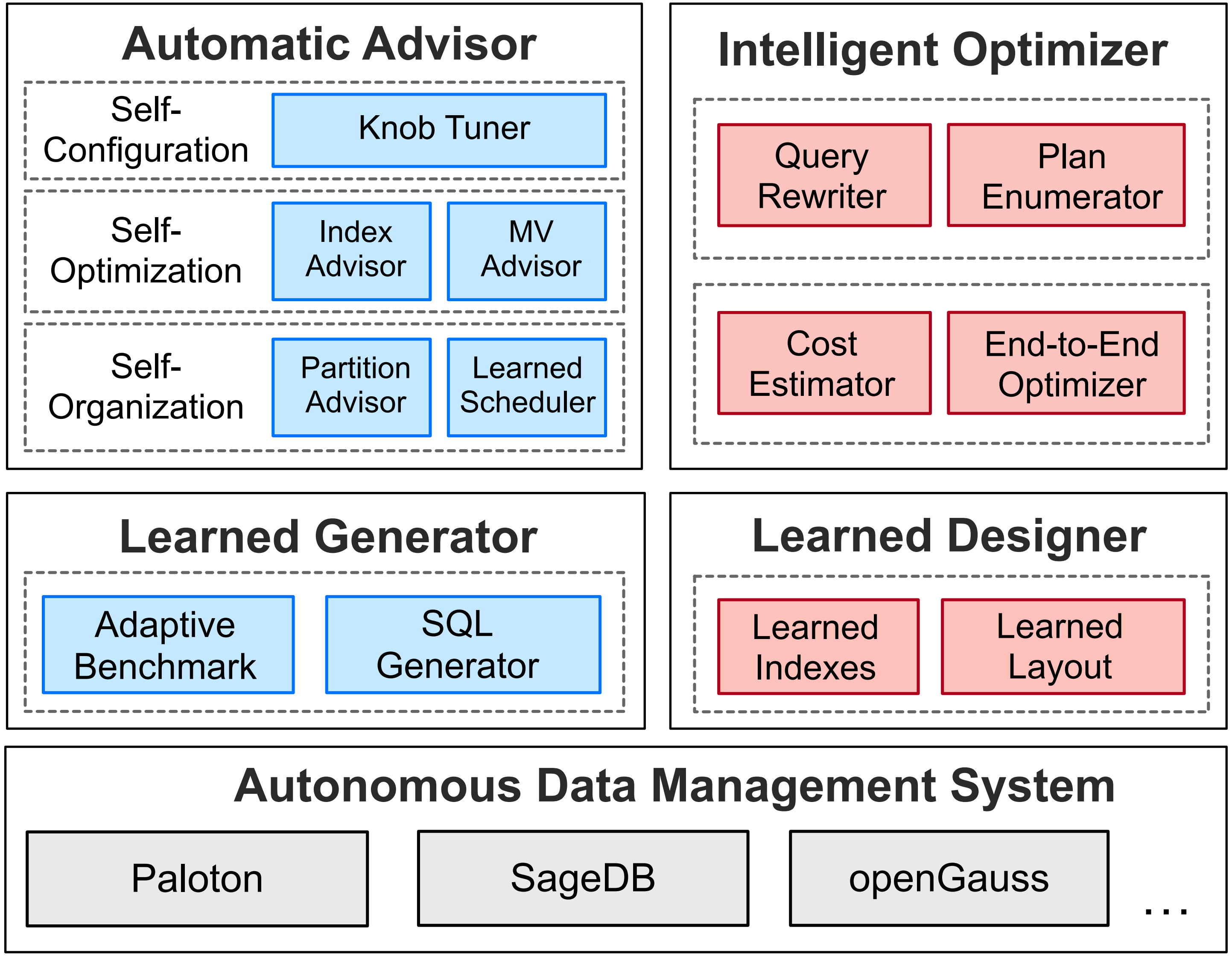
Benjamin Hilprecht, Carsten Binnig, Uwe Röhm. Towards learning a partitioning advisor with deep reinforcement learning. SIGMOD 2019.

Takeaways of Database Partition

- **Learned key-selection partition outperforms heuristic partition**
- **Learned key-selection partition has much higher partition latency for model training**
- **Open Problems:**
  - Adaptive partition for relational databases
  - Partition quality prediction
  - Improve partition availability with replicates

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Query Scheduling

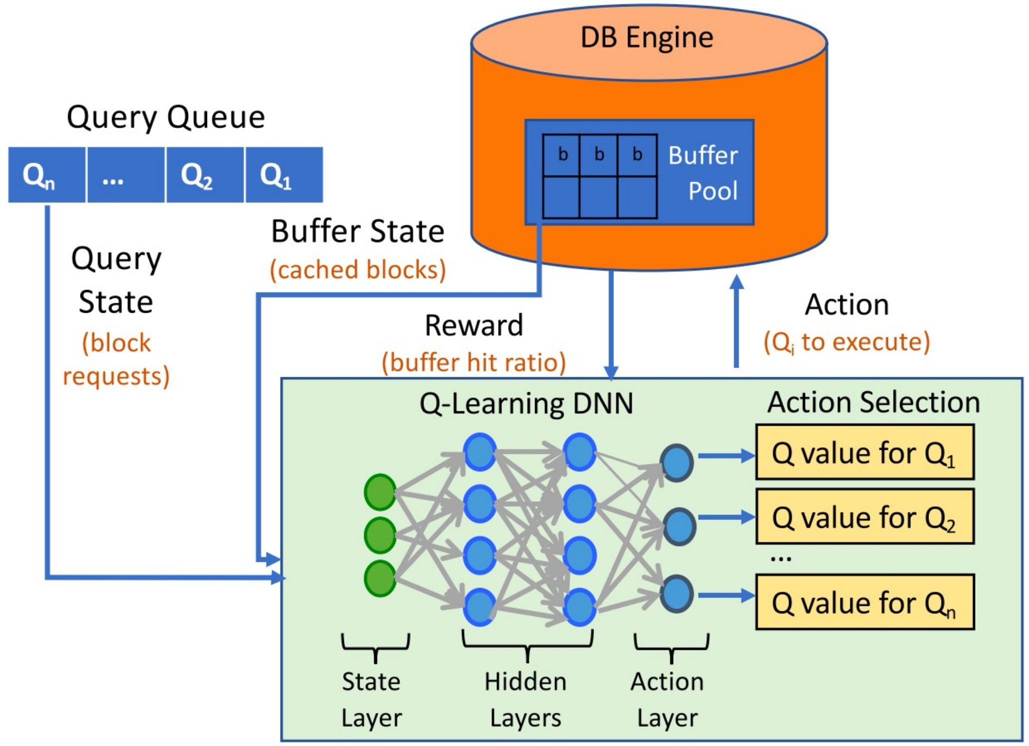
Motivation

- **Effective Scheduling can Improve the Performance**
  - Minimize conflicts between read queries
- **Concurrency Control is Challenging**
  - #-CPU Cores Increase
- **Transaction Management Tasks**
  - Transaction Prediction
  - Transaction Scheduling

Learned Query Scheduling

- **Challenge: Keep the most important blocks cached**
- **Core Idea: Schedule queries to minimize disk access requests with RL**

- Collect requested data blocks (buffer hit) from the buffer pool:
- State Features: buffer pool size, data block requests;
- Schedule Queries to optimize global performance with Q-learning



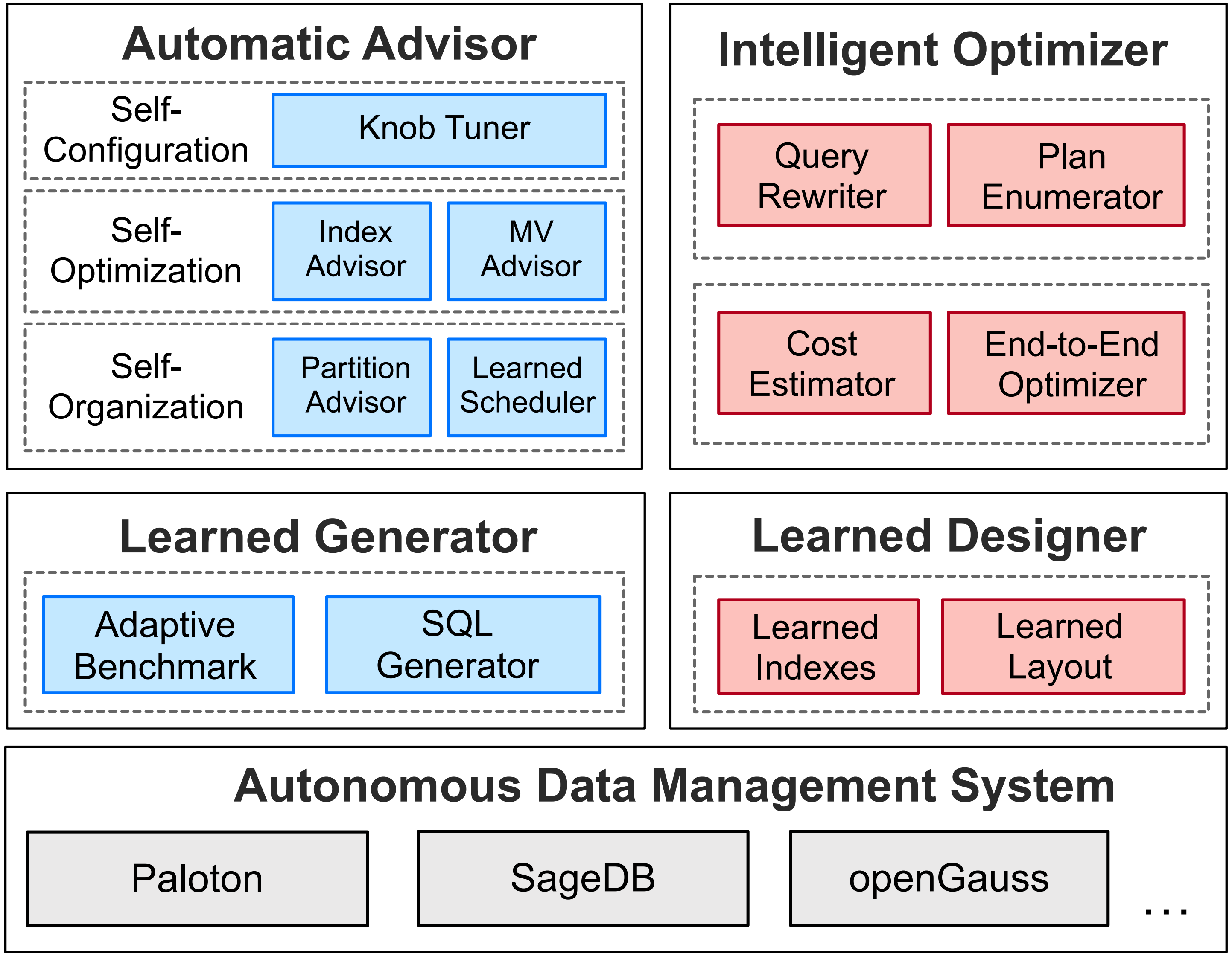
Chi Zhang, Ryan Marcus, and et al. Buffer Pool Aware Query Scheduling via Deep Reinforcement Learning. In VLDB, 2020.

Takeaways of Transaction Scheduling

- **Learned scheduling can achieve higher performance, but takes intolerable long training time**
- **Learned scheduling requires detailed caching block information, which may not be available in some scenarios**
- **Open Problems:**
  - Online workload Scheduling
  - Query Trend Prediction
  - Support transactions

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Query Generation

- **Motivation**
  - Companies generally do not release user queries (out of privacy issues);
  - It is vital to generate synthetical workloads (in replace of real workloads), and release the synthetical workloads to the public to train the ML models

Automatic Query Generation

➢ **How to generate queries that meet legality, diversity, and reprsentative?**

Definiation: Given a scheme and constraints (e.g., cost/ cardinality ranges), we generate k SQL queries which can (i) legally execute in the databse and (ii) meet the constraints.

Example: Generate 1000 TPC-H SQLs whose cardinality equals 1000.

➢ **Challenges & Solutions:**

- It is hard to predict the performance of generated SQLs, i.e., whether they meet the constraints; → (RL) Construct a LSTM-based critic to predict the long-term benefits of any intermediate queries; utilize actor to explore new tokens;
- It is hard to generate diverse SQLs; → Construct a probablistics model to ensure the diversity of generated queries;
- Grammar and syntax constraints need to be considered to generate legal queries; → Construct a FSM to prune illegal tokens for current intermediate queries;

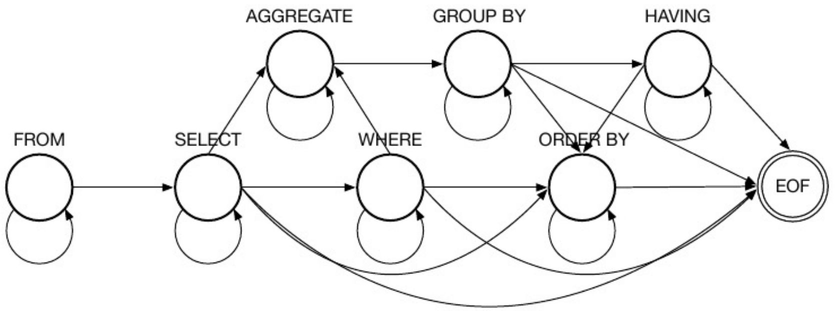
Lixi Zhang, Chengliang Chai, Xuanhe Zhou, Guoliang Li. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. SIGMOD 2022.

Automatic Query Generation

Query Legality

➢ SQL Grammar:

- FSM



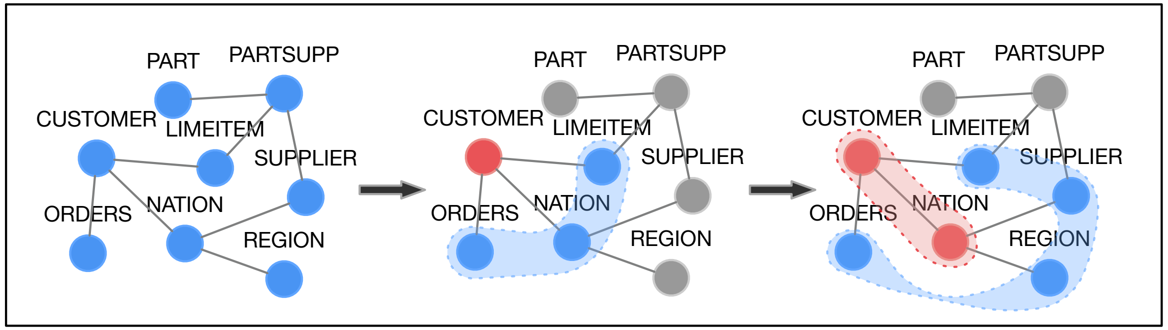
Advantage:

✓ Easy to add new grammar

✓ Customize SQL queries

➢ Semantic Checks:

① **Join Relation**



② **Type Checking**

- **Aggregation**: Aggregate Function
- **Predicate**: WHERE caluse, HAVING clause

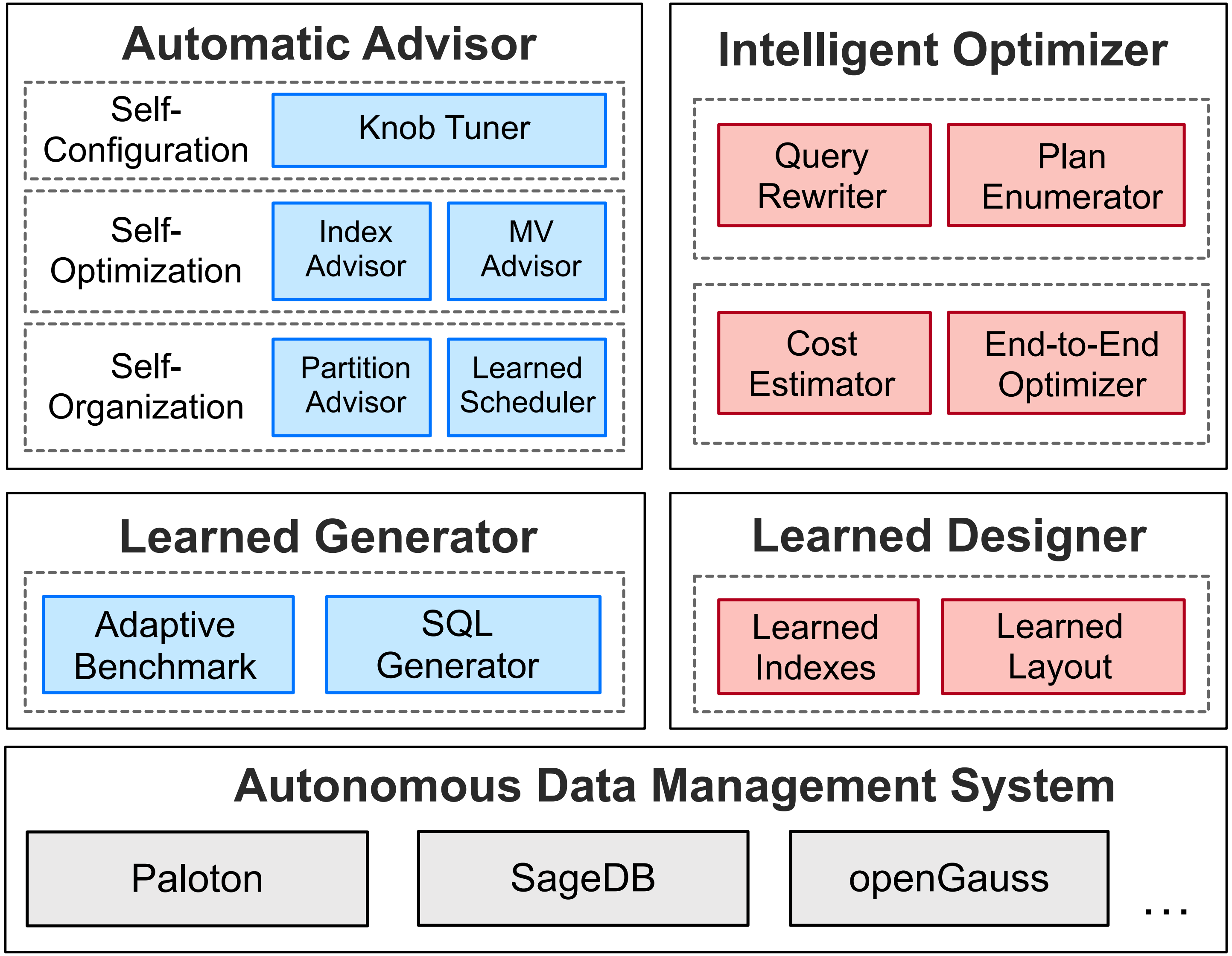
③ **Operand Restriction**

- " people_name = China " X

Lixi Zhang, Chengliang Chai, Xuanhe Zhou, Guoliang Li. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. SIGMOD 2022.

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Training Data Generation

Motivation

- **Machine learning is widely adopted in database components**
- **It is challenging to obtain suitable datasets**
  - Training data is rarely available in public
  - It is time-consuming to manually generate samples (e.g., over 6 months for 10,000 jobs with 1T data)
- **It is hard to measure the dataset quality**
  - The size of training data
  - The quality of extracted features
  - The availability of valuable ground-truth labels

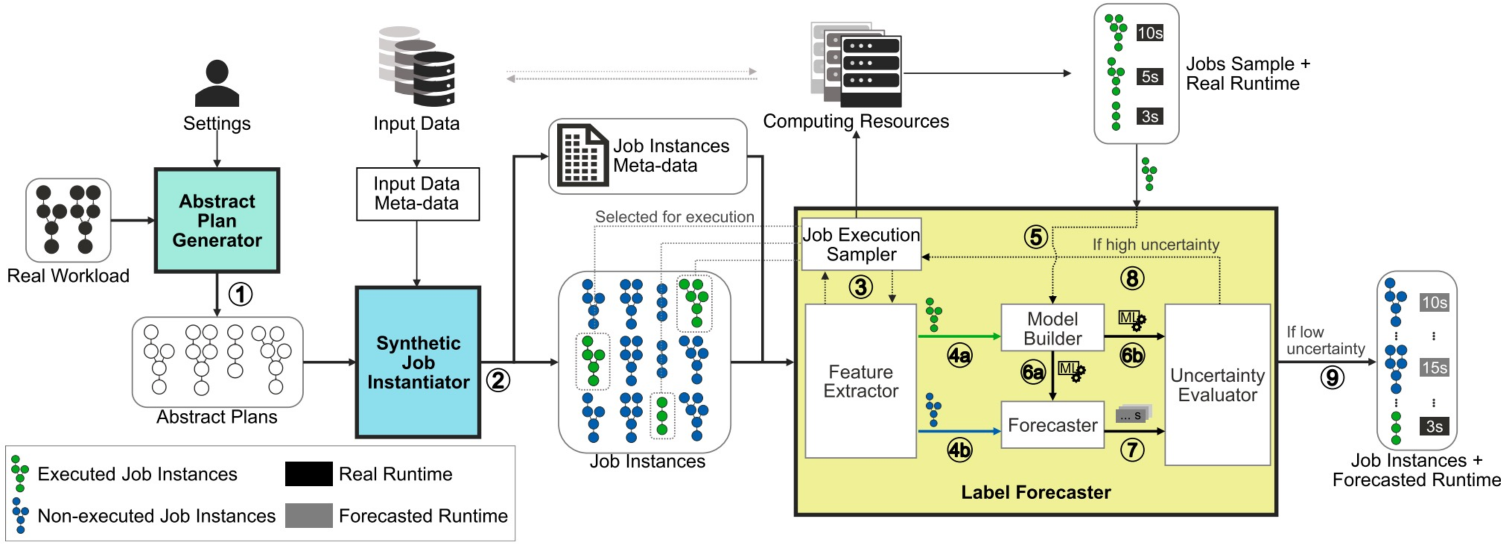
Automatic Training Data Generation

☐ Challenges in existing workload generators (tpch, sqlsmith)

- ➢ The workloads have low variance
- ➢ Fail to label the workloads (e.g, cost, execution time)

☐ Core Idea: Label workloads with adaptive learning

- ➢ Input a small query workload and sample data;
- ➢ Create abstract plans (DAGs without actual physical operators) which follow the patterns in the input workload;
- ➢ Instantiate the abstract plans based on the data distribution in the sample data;
- ➢ Label the generated plans via active learning without executing all the plans



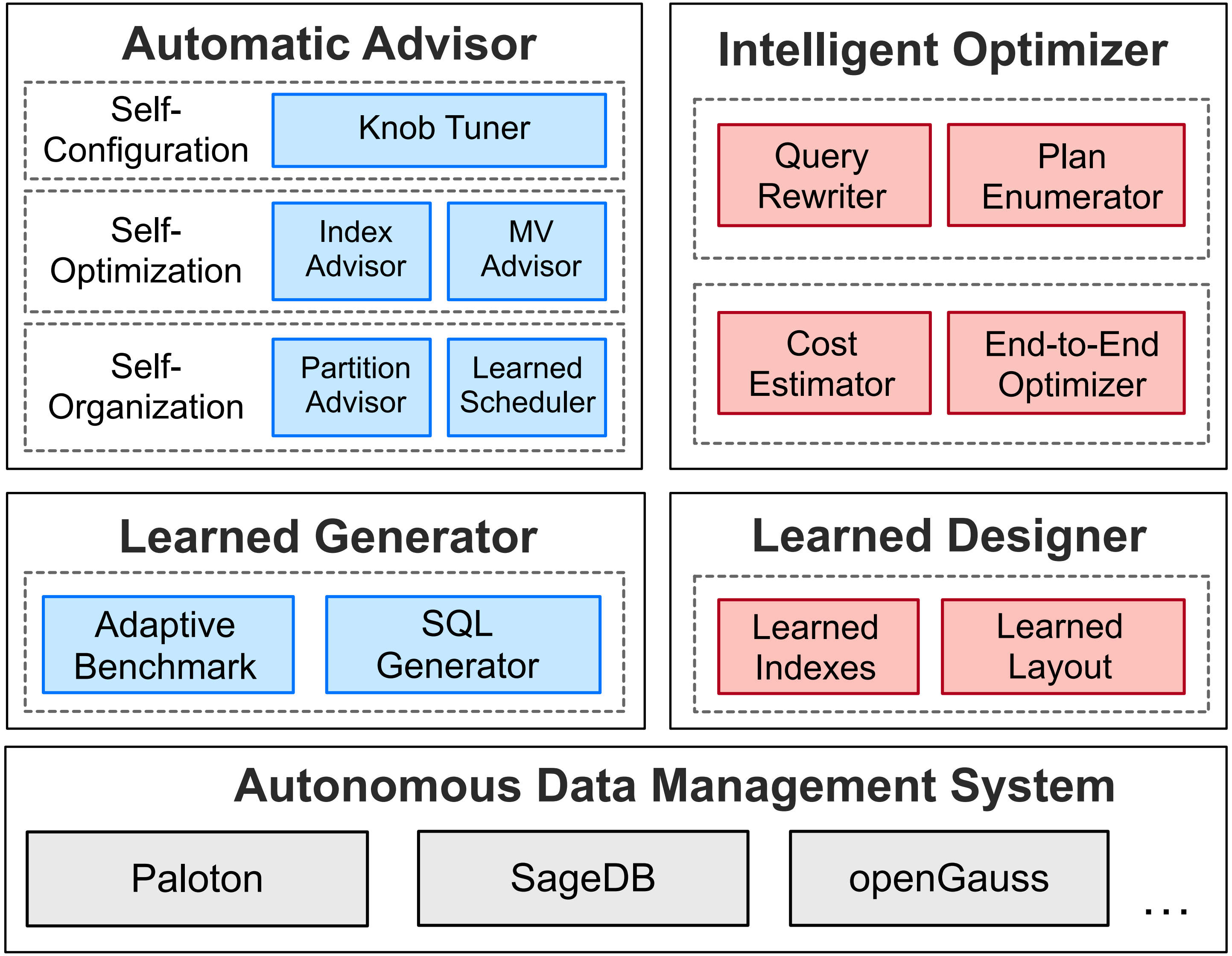
Francesco Ventura. Expand your training limits! generating training data for ML-based data management. SIGMOD, 2021.

Takeaways of Learned Generator

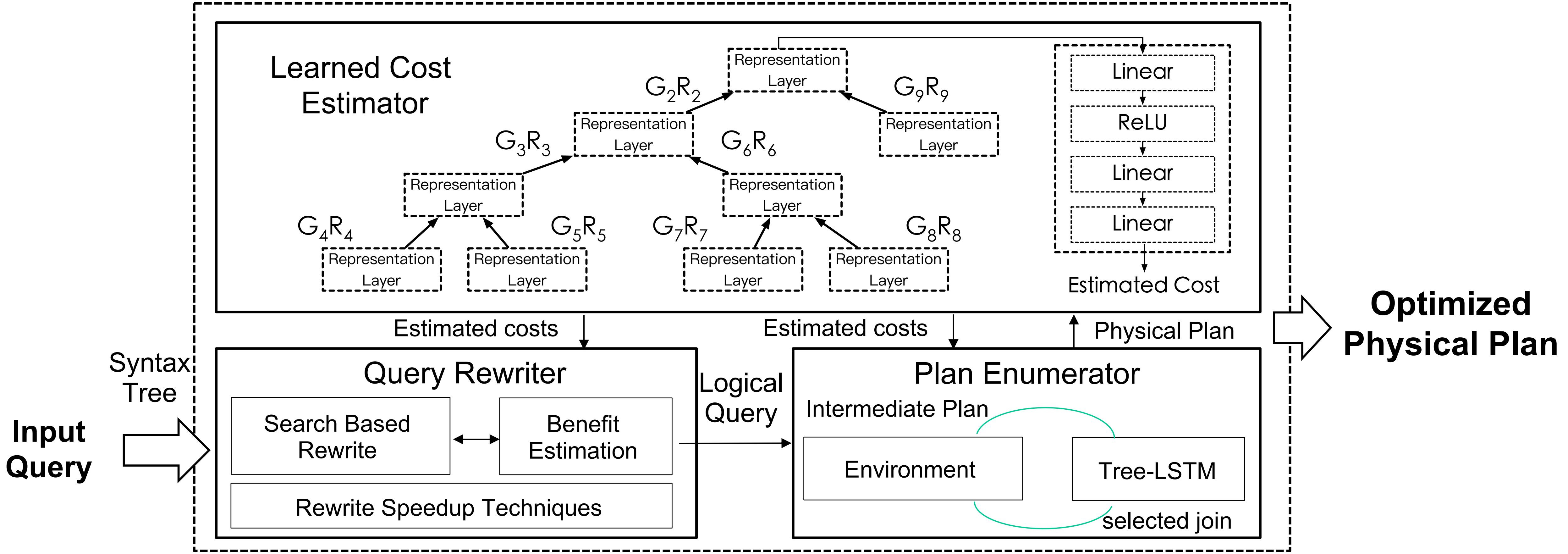
- **Generated queries or performance labels are useful to test database functions**
- **Sometimes most real queries have similar structures and may not be effective as generated queries**
- **Open Problems:**
  - Semantic-aware query generation
  - Low overhead query generation

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- ***Autonomous* Databases**



Learned Optimizer: An Overview



- **Query Rewriter: Efficiently optimize query in logical Level**
- **Plan Enumerator: Powerfully optimize query in physical Level**
- **Cost Estimator: Accurately estimate the plan execution cost**

Intelligent Optimizer: Technique Development

- Performance (latency, quality, cost accuracy)
- Adaptivity (queries, datasets)

Query Rewriter

Optimize Predicate Pushdown
- SMT+Binary Clasifier

Optimize Rewrite Orders
- Monte Carlo Tree Search

Join Enumerator

Optimize Join Orders
- Reinforcement Learning

Dynamic Plan Adjustment
- Monte Carlo Tree Search

Optimize Physical Operators
- Learned Hinter

Cost Estimator

Improve Estimation Quality
- AR; SPNs
- LSTMs;

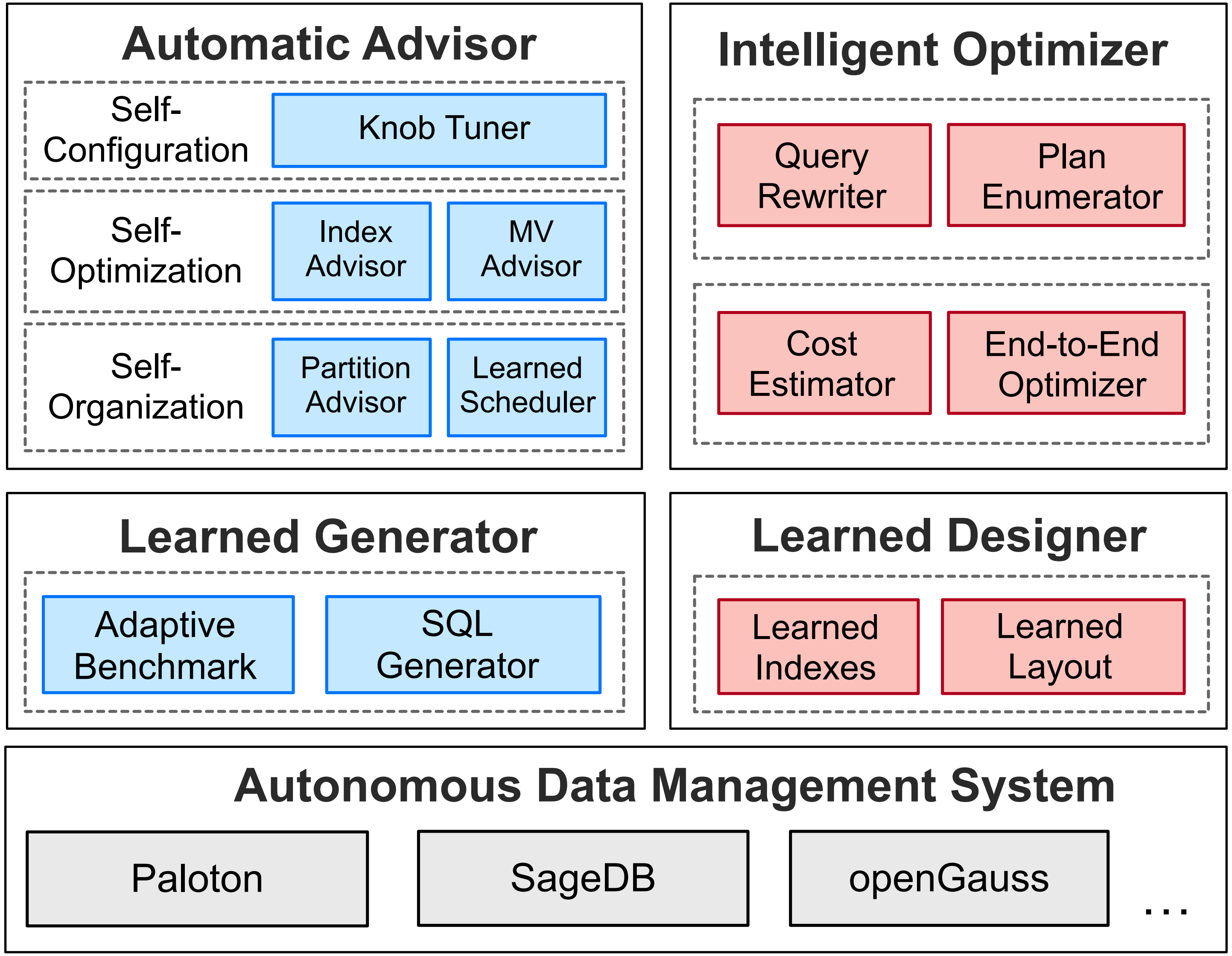
Support Multi-Table
- Tree Ensembles
- Joint AR

Support String Data
- Normalizing Flows

Improve Adaptivity
- Pre-Trained Models

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Automatic Query Rewrite

Motivation:

- **Many queries are poorly-written**
  - Slow operations (e.g., **subqueries**/joins, **union**/union all) ;
  - Looks pretty to humans, but physically inefficient (e.g., take subqueries as temporary tables);
- **Existing methods are based on heuristic rules**
  - Top-down order may not be optimal (e.g., remove aggregates before pulling up subqueries)
  - Available rules are limited
- **Trade-off in SQL Rewrite**
  - **Best Performance:** Enumerate for the best rewrite order
  - **Minimal Latency:** SQL Rewrite requires low overhead (milliseconds)

Learning-based Query Rewrite

☐ Challenge:

- **Optimize existing rewrite rules, or even generate new rules**
  - Equivalence verification
- **Search rewrite space within time constraints**
  - Rewrite within milliseconds;
- **Estimate rewrite benefits by multiple factors**
  - Case1: Reduced costs by selected rewrites
  - Case2: Future reduced costs by further rewriting the query

Query Rewrite: Technique Development

- **Performance (latency, quality, cost accuracy)**
- **Adaptivity (queries, datasets)**

Query Rewriter

Optimize
Predicate Pushdown

- SMT+Binary Clasifier

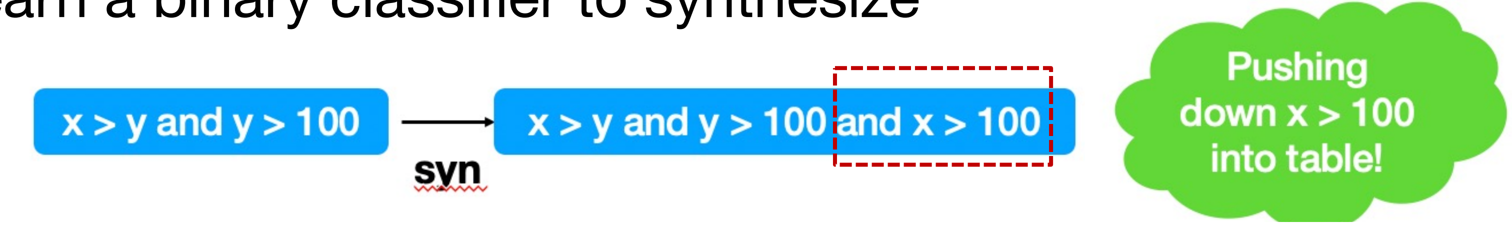
Optimize
Rewrite Order

- Monte Carlo Tree Search

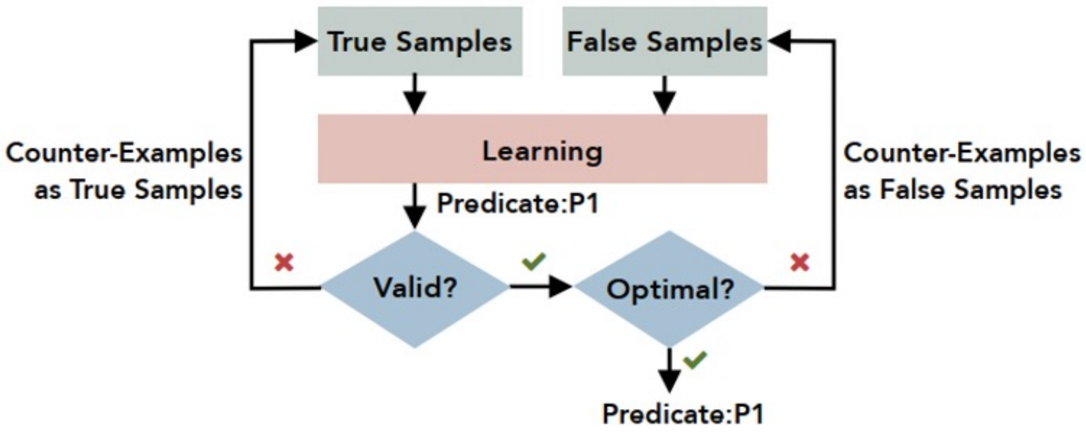
Query Rewrite: Optimize Predicate Pushdown

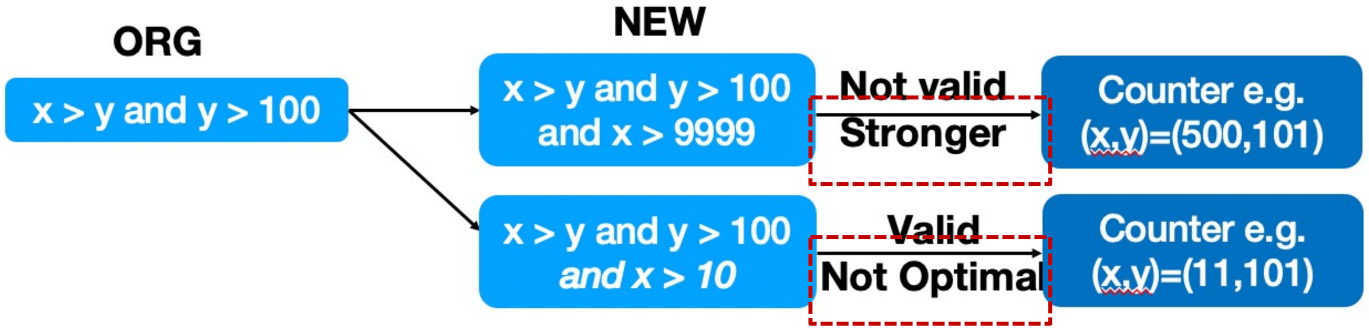
☐ Motivation: Traditional predicate-pushdown is less powerful

- **Core Idea:** Predicate → Learn a binary classifier to synthesize <u>valid</u> and <u>better</u> predicates



- **Approach:** Generate TRUE/FALSE samples to train the binary classifier
  - Classifier: SVM model over the input columns
  - Each TRUE sample should be accepted by a valid predicate
  - Each FALSE sample should by rejected by an optimal predicate



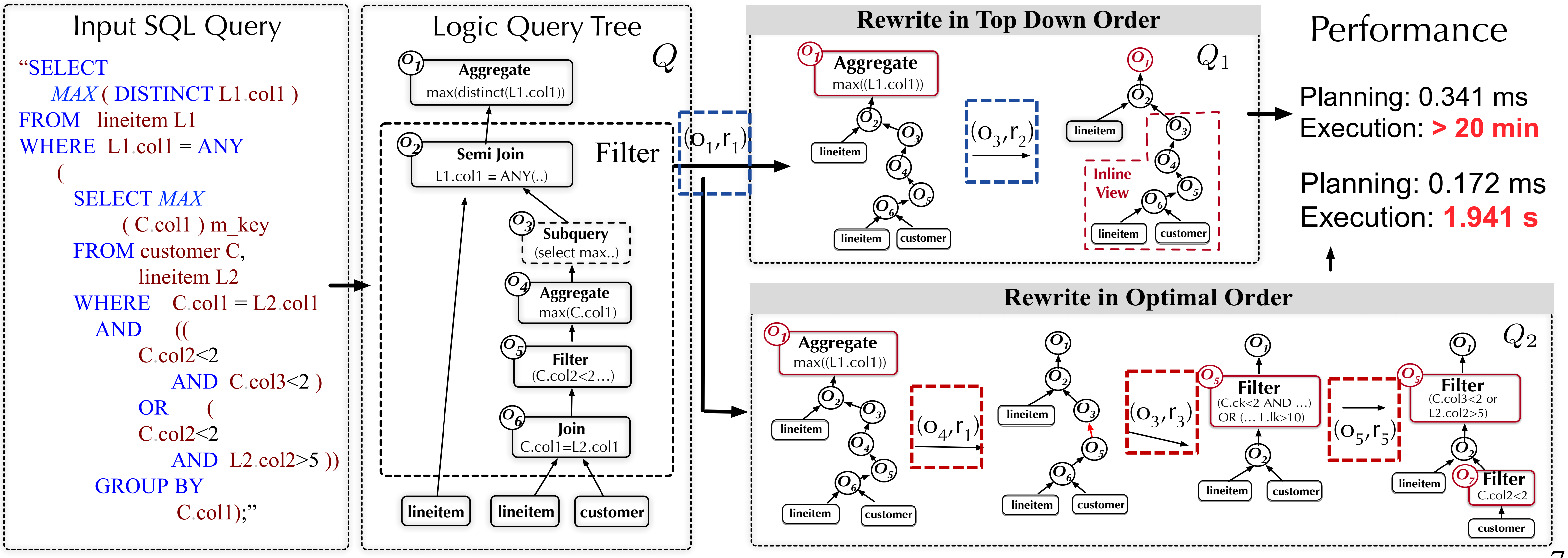


Qi Zhou, Joy Arulraj, Shamkant B, et al. SIA: Optimizing Queries using Learned Predicates. SIGMOD, 2021.

Query Rewrite: Optimize Rewrite Orders

☐ The Strategies of applying rewrite rules

Given a slow query and a set of rewrite rules, apply the rewrite rules to the query so as to gain the equivalent one with the minimal cost.



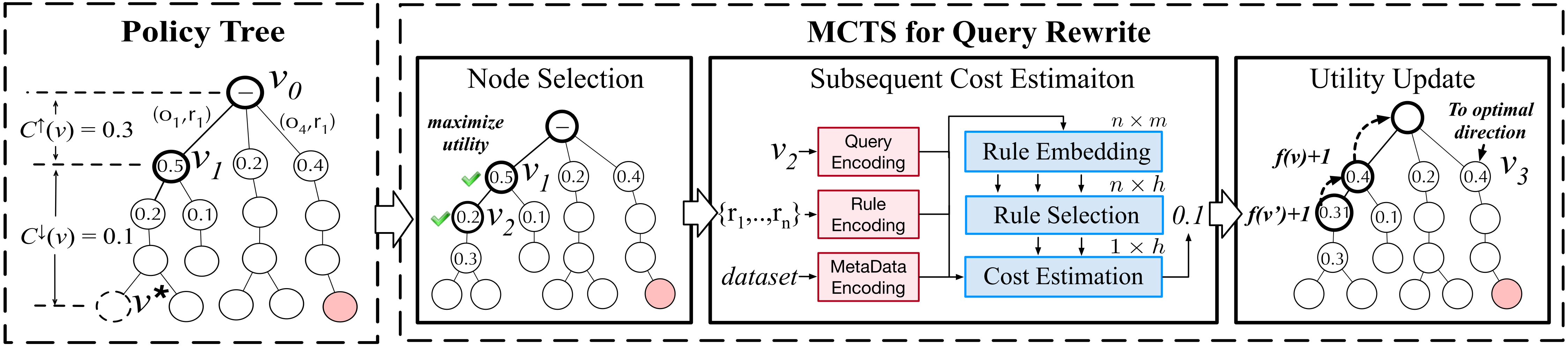
Query Rewrite: Optimize Rewrite Orders

- **A slow query may have various rewrite of different benefits**
  - **(1) Initialize a Policy Tree Model**
    - **Node v_i**: any rewritten query
    - **$C^{\uparrow}(v_i)$**: previous cost reduction
    - **$C^{\downarrow}(v_i)$**: subsequent cost reduction

- **To select from numerous rewrite orders**
  - **(2) Policy Tree Search Algorithm**

$$\mathcal{U}(v_i) = \left(C^{\uparrow}(v_i) + C^{\downarrow}(v_i)\right) + \gamma\sqrt{\frac{ln(\mathcal{F}(v_0))}{\mathcal{F}(v_i)}}$$

  - **(3) Multiple Node Selection**
    - DP Algorithm

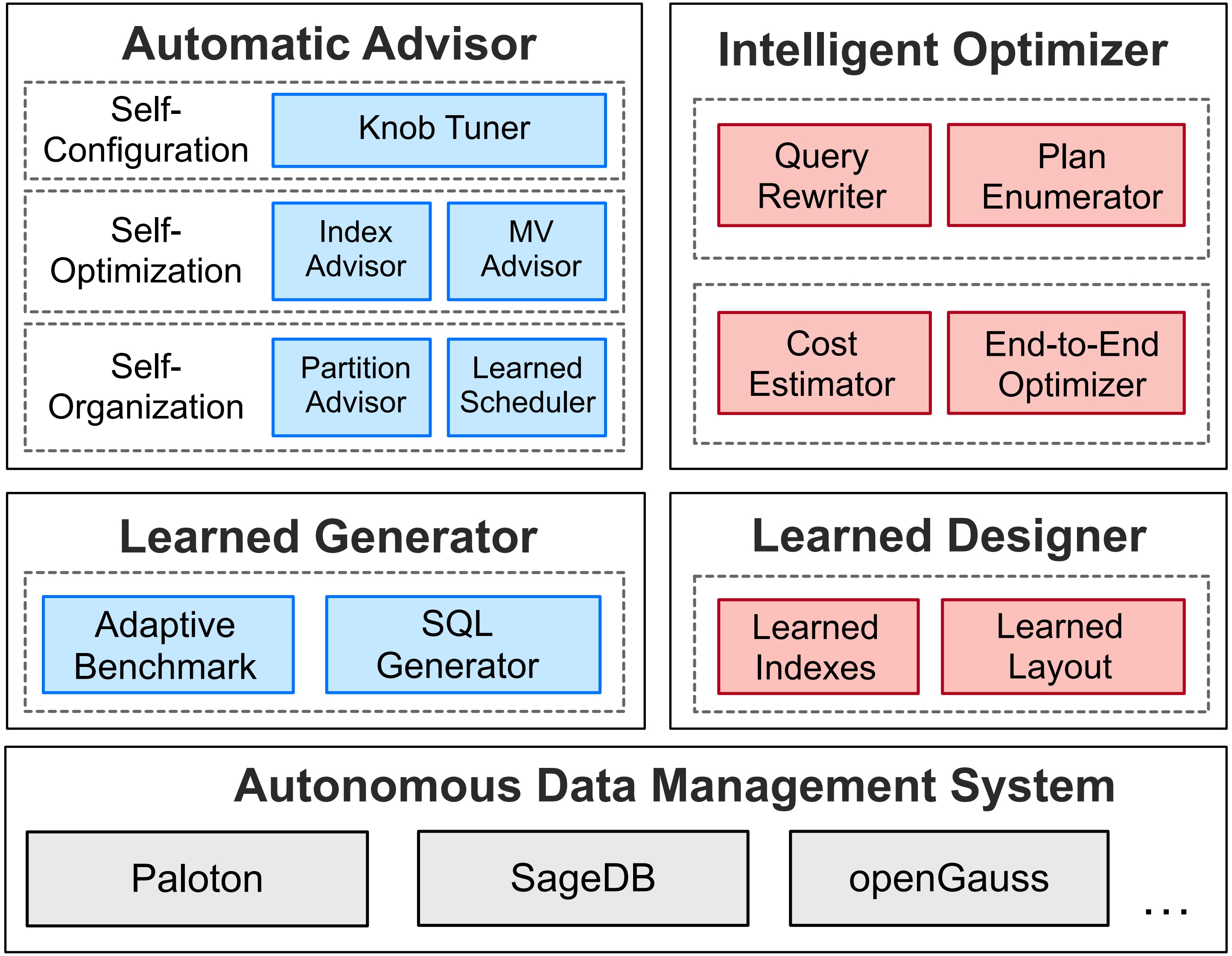


Take-aways of Query Rewrite

- **Traditional query rewrite method is unaware of rewrite benefits, causing redundant or even negative rewrites**
- **Search-based rewrite works better than traditional rewrite for complex queries**
- **Rewrite benefit estimation improves the performance of simple search based rewrite**
- **Open Problems**
  - Balance Rewrite Latency & Performance
  - Adapt to different rule sets/datasets
  - Design new rewrite rules

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Plan Enumerator

◻ Motivation:

- Planning cost is hard to estimate
  - ➢ The plan space is huge
- Traditional optimizers have some limitations
  - ➢ DP gains high optimization performance, but causes great latency;
  - ➢ Random picking has poor optimization ability
- Finetuning existing optimiers can gain higher performance

Plan Enumerator: Technique Development

- **Performance (latency, quality, cost accuracy)**
- **Adaptivity (queries, datasets)**

Plan Enumerator

- Reinforcement Learning

- Monte Carlo Tree Search

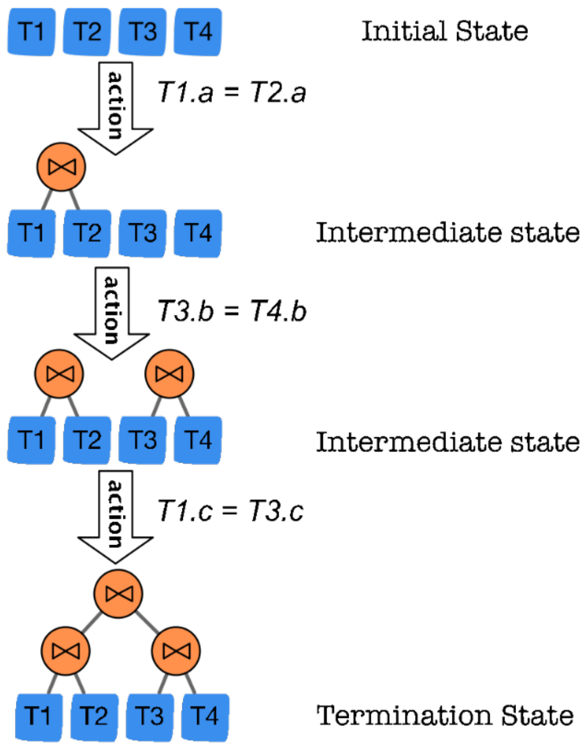
- Learned Hinter

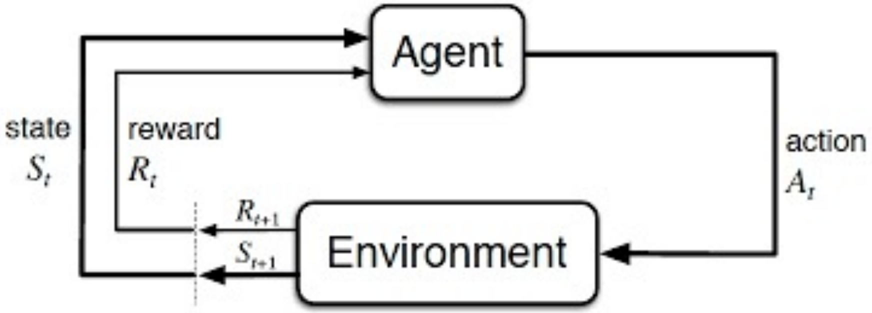
Join Order Selection: Optimize the Performance

☐ Numerous candidate join orders to select before execution → Model it as RL

- ➢ Agent : optimizer
- ➢ Action: join
- ➢ Environment: Cost model, database
- ➢ Reward： Cost ,Latency
- ➢ State : join order (Neo: encode query structures)

```
Select *
From T1,T2,T3,T4
Where T1.a = T2.a
   and T3.b = T4.b
   and T1.c = T3.c
```

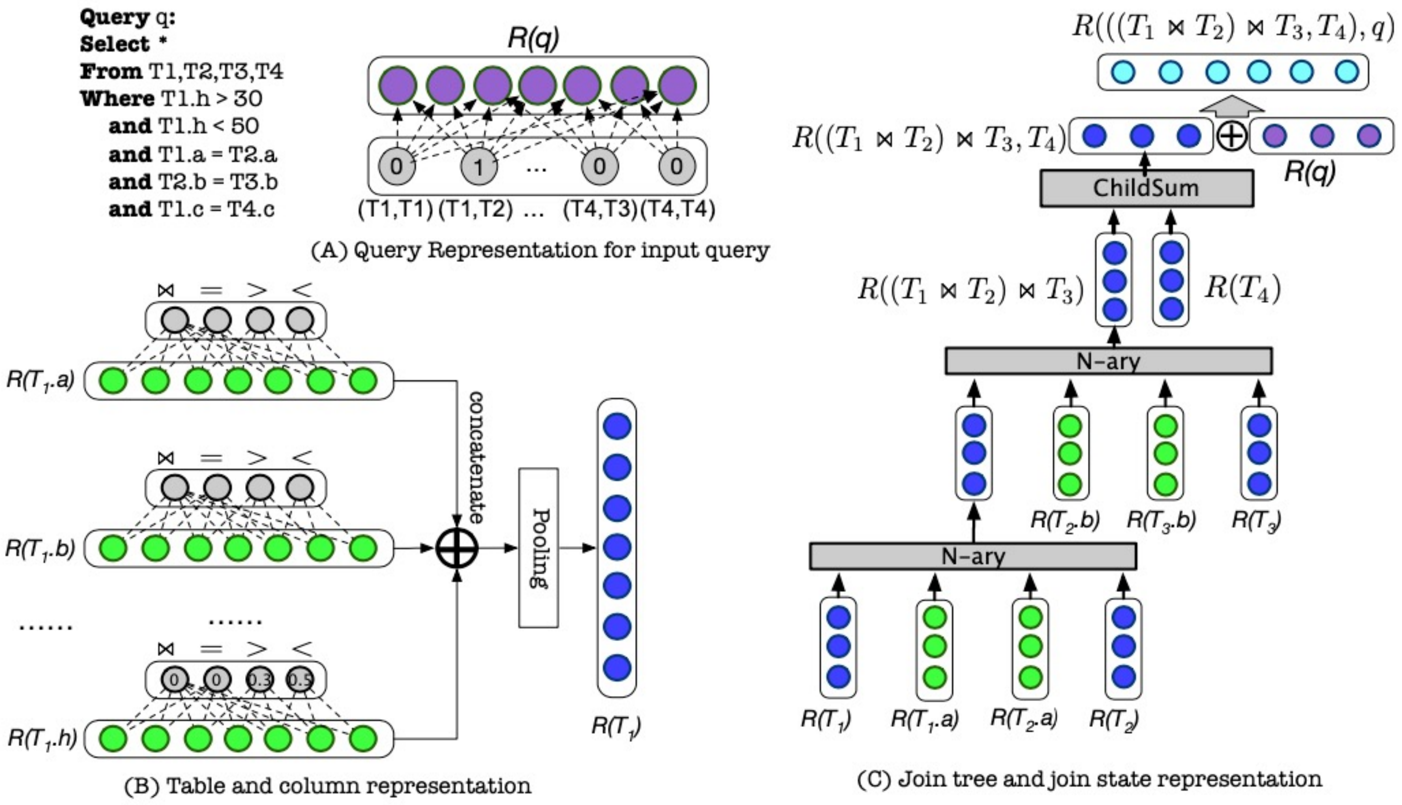




- Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning. CoRR, 2018.
- Marcus, R., Negi, P., et al. Neo: A Learned query optimizer. *VLDB, 2018.*

Join Order Selection: Adapt to Schema Changes

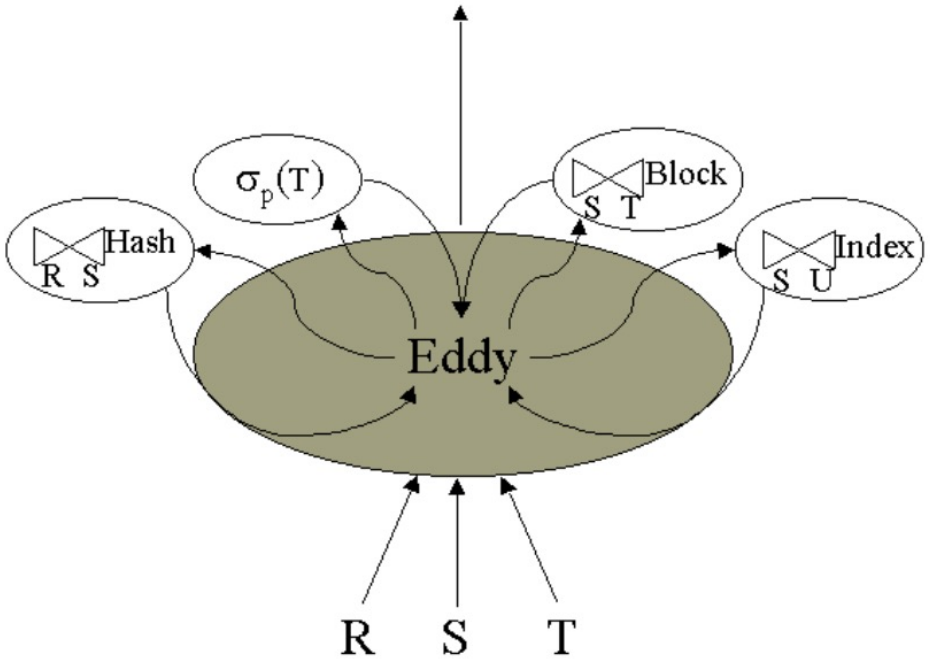
- Adaptively assemble the selection model, and adapt to schema changes (e.g., column, table)
  - Encode the operator relations and metadata features of the query;
  - Embed the query features with Tree-LSTM; (the tree structure can adapt to different tables/columns)
  - Decide join orders with RL model



X. Yu, G. Li, and C.C. et al. Reinforcement learning with tree-lstm for join order selection. In ICDE, 2020.

Join Order Selection On-the-Fly

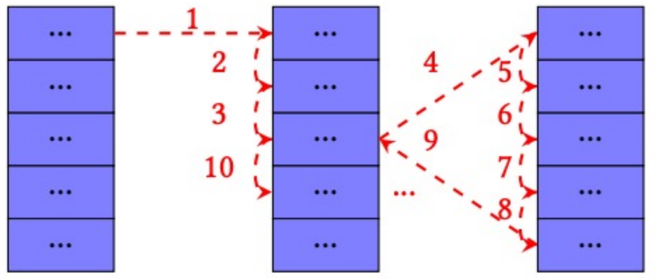
- **Update execution orders of tuples on the fly and preserve the execution state**
  - **Tuples flows into the Eddy from input relations (e.g., R, S, T);**
  - **Eddy routes tuples to corresponding operators (the order is adaptively selected by the operator costs);**
  - **Eddy sends tuples to the output only when <u>the tuples have been handled by all the operators</u>.**



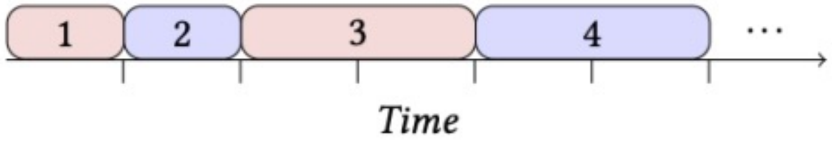
Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. SIGMOD, 2000.

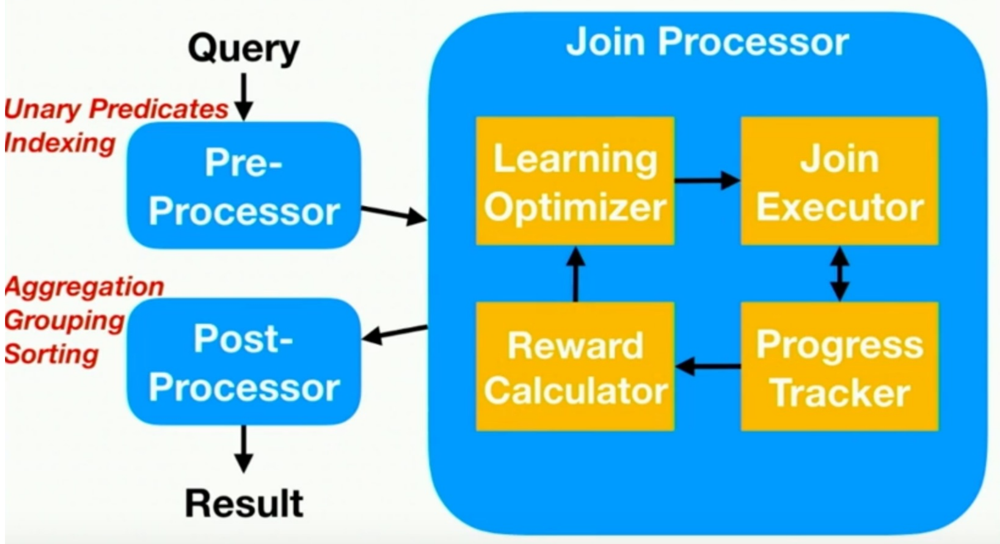
Join Order Selection On-the-Fly

- Improve runtime plan adjustment performance
  → Join reorder with MCTS
  - Assume executing joins in "depth-first";
  - Split time slices: 0.001s;
  - Approach
    - In each slice, reserve complete result tuples, and drop under-join intermediate tuples
    - Evaluate the join order benefits by (1) the table coverage and (2) result tuple ratio
    - Remove duplicate result tuples (based on their position vectors)
    - Judiciously select higher-benefit join orders with MCTS (the UCT function)



N way join

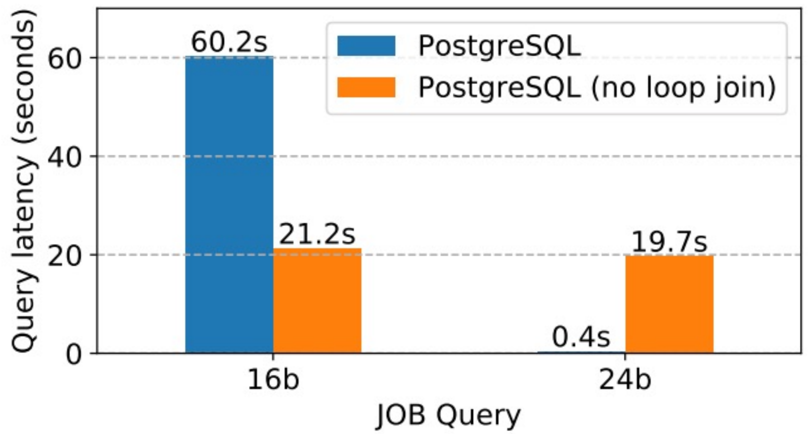




Trummer, et al Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In SIGMOD, 2019.

Physical Operator Selection (Plan Hinter)

- Physical operators can significantly affect the performance
  - E.g., improving performance by deactivating the loop join operator

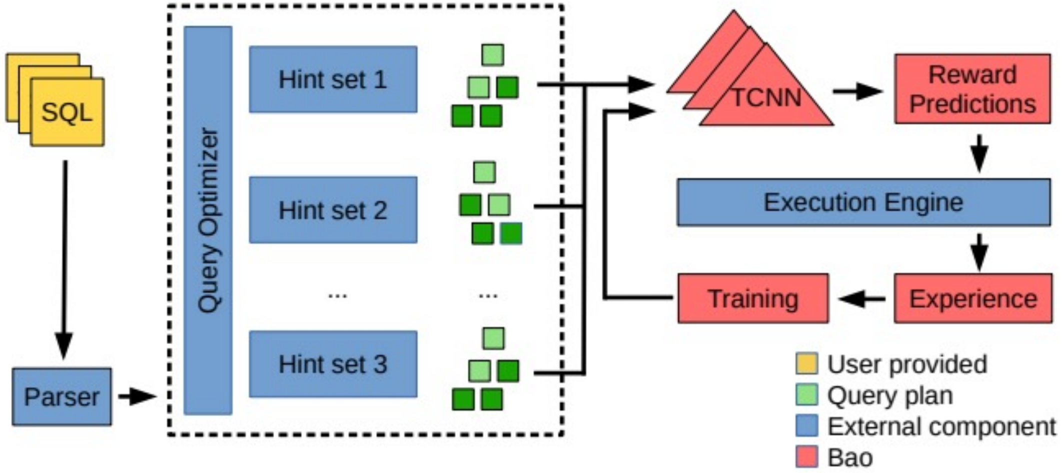


- Model it as a Multi-armed Bandit Problem
  - Model each hint set $HSet_i$ as a query optimizer

$$HSet_i : Q \rightarrow T$$

  - For a query q, it aims to generate optimal plan by selecting proper hint sets, which is dealed as a regret minimization problem:

$$R_q = \left(P(B(q)(q)) - \min_i P(HSet_i(q)) \right)^2$$



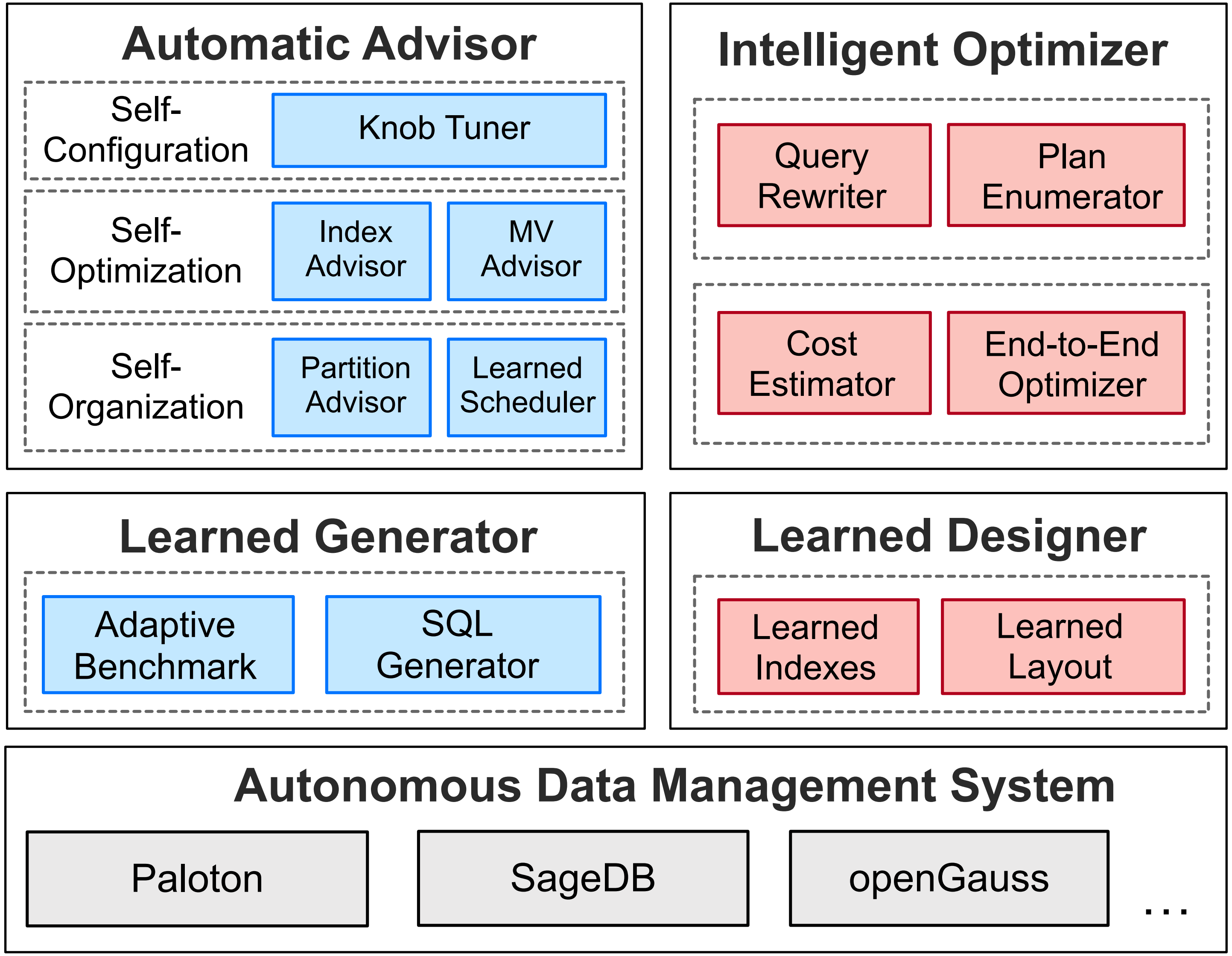
Ryan Marcus et al. Bao: Making Learned Query Optimization Practical. In SIGMOD, 2021.

Take-aways of Plan Enumerator

- Learning based algorithm usually can find high efficient plans, especially for large queries with multiple joins.
- Offline learning methods use the sampled workload to pretrained the model. It will give good plans for the incoming queries.
- Online-learning methods do not need previous workload and can give good plans. But it needs the *customized engine* and is hard to be applied in existing databases.
- **Open Problems**
  - Raise the generalization performance of offline learning methods for unseen queries.
  - Ensure the plan given by learned model is robust (explicable).
  - Speed up the model training time, e.g. transferring previous knowledge.
  - Make the model aware of the data update.

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Cost Estimator: Technique Development

- **Performance (latency, cost accuracy)**
- **Adaptivity (queries, datasets)**

Cost Estimator → **Improve Estimation Quality** → **Support Multi-Table** → **Support String Data** → **Improve Adaptivity**

Improve Estimation Quality	Support Multi-Table	Support String Data	Improve Adaptivity
- AR; SPNs - LSTMs;	- Tree Ensembles - Joint AR	- Normalizing Flows	- Pre-Trained Models

Automatic Cardinality/Cost Estimation

□ Motivation:

□ One of the most challenging problems in databases

- ➢ Achilles Heel of modern query optimizers

□ Traditional methods for cardinality estimation

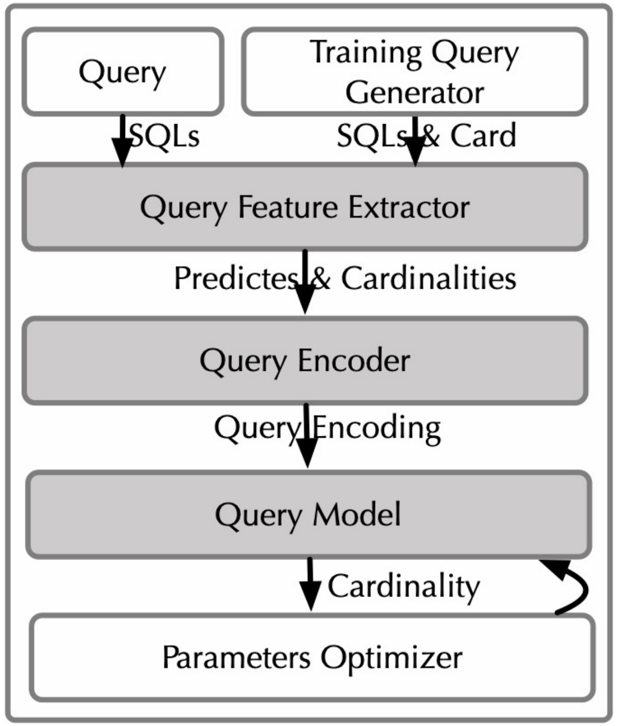
- ➢ Sampling (on base tables or joins)
- ➢ Kernel-based Methods (Gaussian Model on Samples)
- ➢ Histogram (on single column or multiple columns)

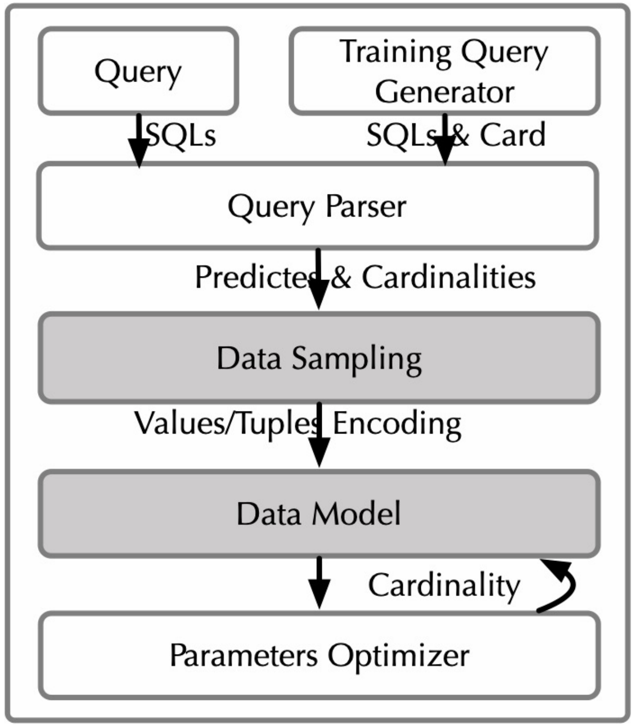
□ Traditional cost models

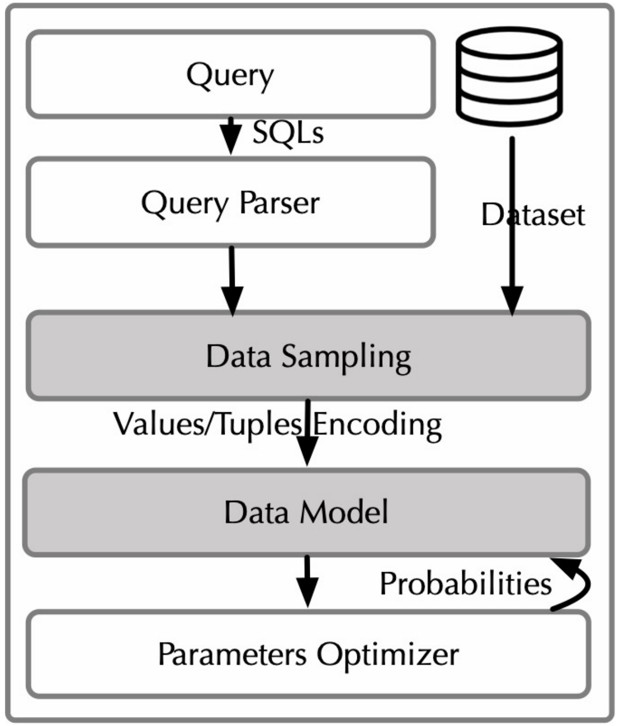
- ➢ Data sketching/data histogram based methods
- ➢ Sampling based methods

Viktor Leis, Andrey Gubichev, et al. How good are query optimizers, really? In VLDB, 2015.

Categories of Cardinality Estimation







(1) Supervised Query Methods

- Multi-set Convolutional network
- Tree-based ensemble

(2) Supervised Data Methods

- Gaussian kernel
- Uniform mixture model

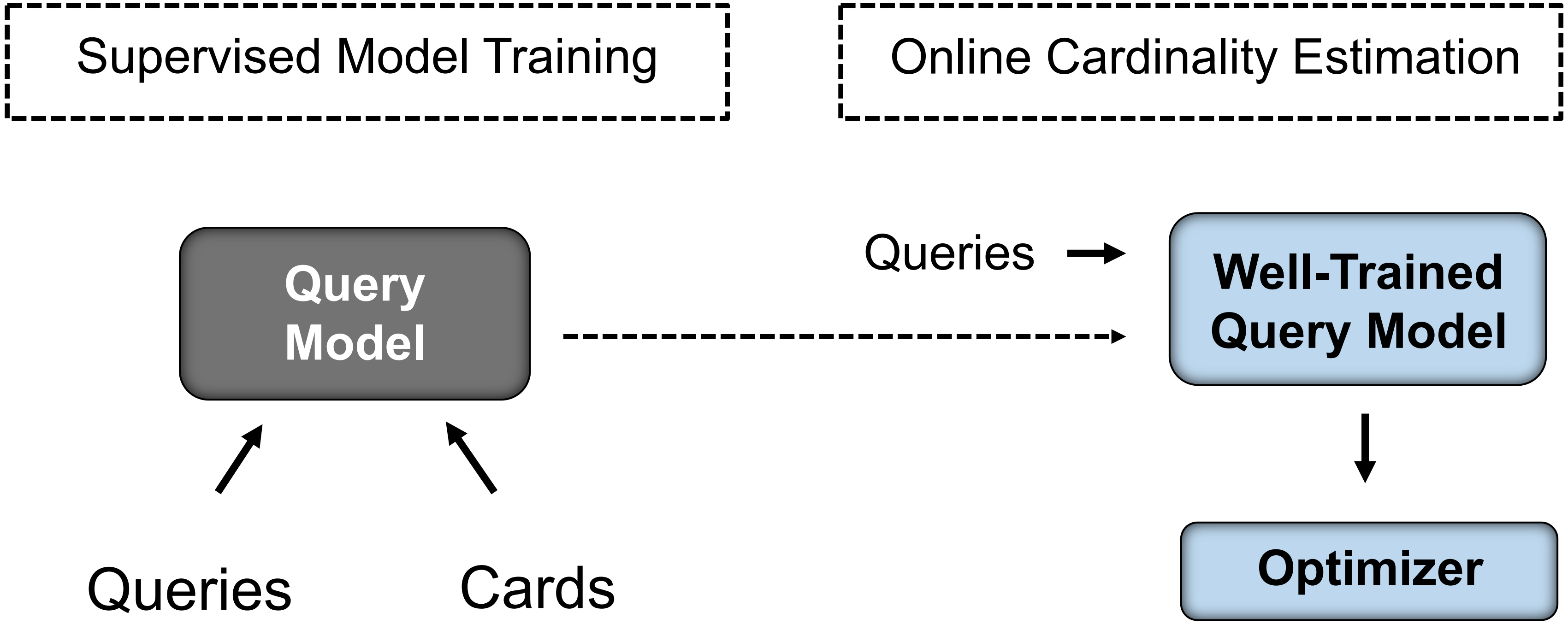
(3) Unsupervised Data Methods

- Autoregressive
- Sum product network

Supervised Query Methods for Cardinality Estimation

□ Problem Definition

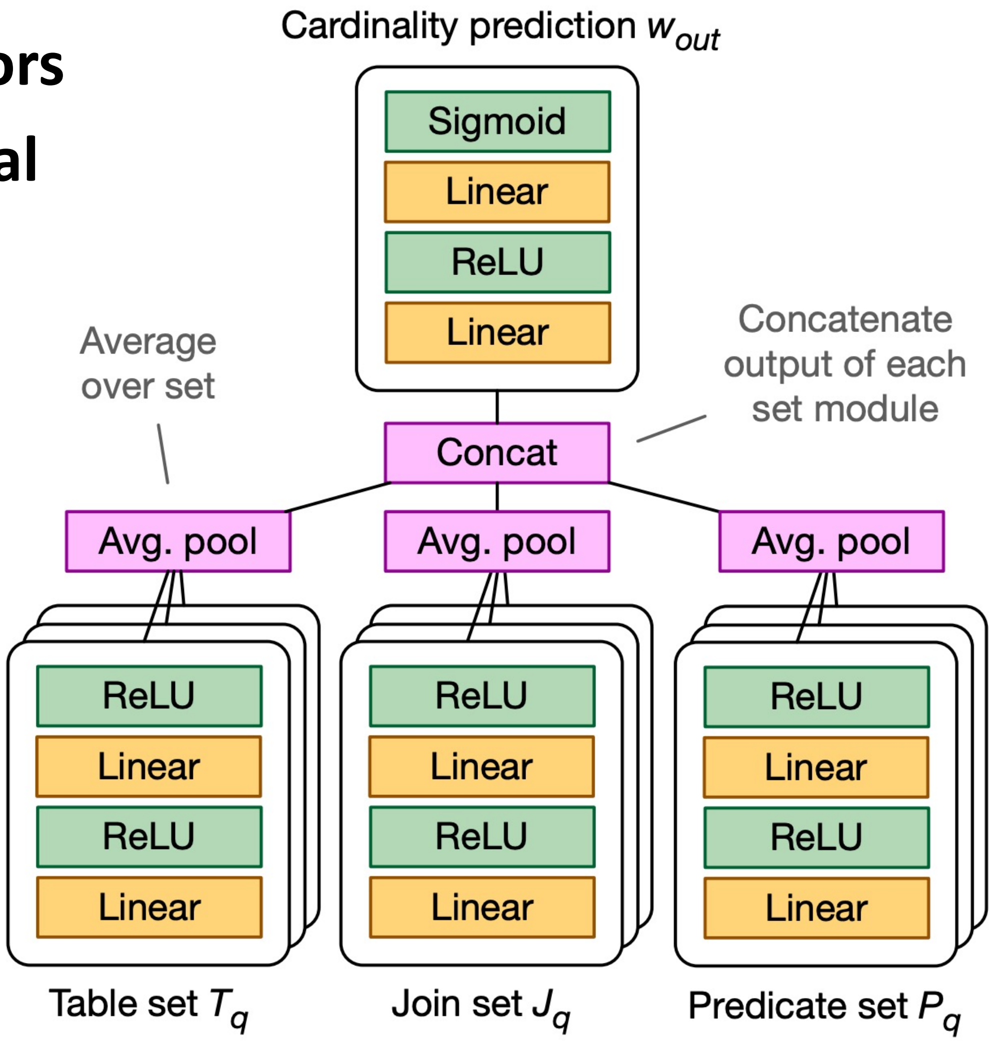
A regression problem: learn the mapping function between query Q and its actual cardinality



Query-Driven: Deep Learning for Cardinality Estimation

- ☐ **Motivation: Traditional estimator makes errors**
- ☐ **Core Idea: Use Multi-set Convolutional Neural Network to support join queries**

  - ➢ Linear Models for different part of SQL (table, joins, predicates)
  - ➢ Pooling Varying-sized representations (avg pooling)
  - ➢ Concatenate different parts



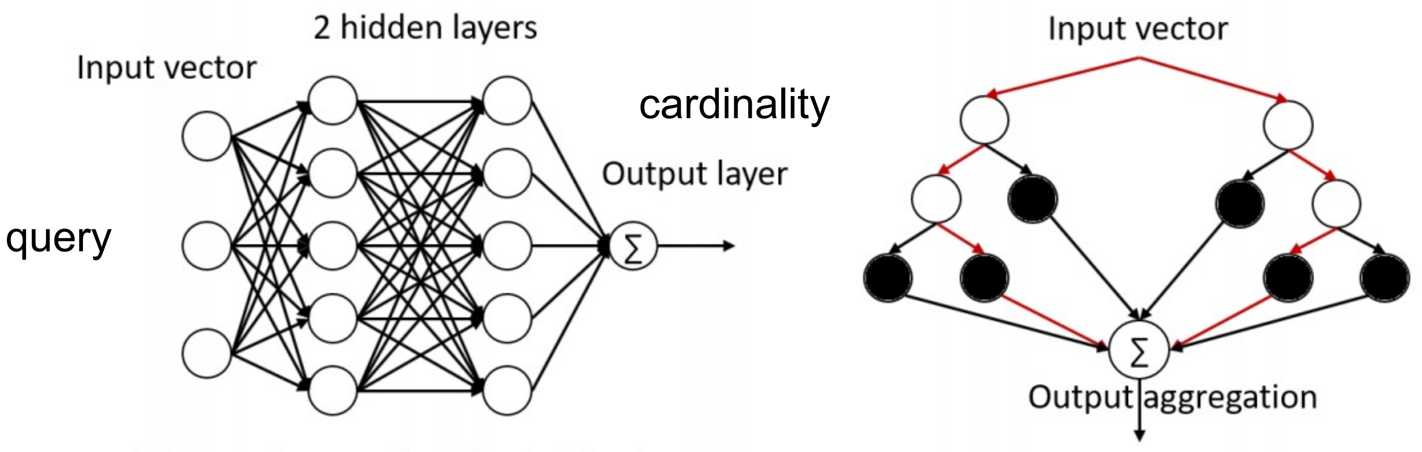
A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR, 2019.

Query-Driven: Tree-Ensembling for Cardinality Estimation

- **Traditional estimation methods assume column independency, and works bad for multi-dimension range queries**
  - **Any conjunctive query on columns *C* can be represented as:**

$$(c_1 \leq lb_1 < c_2) \wedge (c_3 < ub_1 \leq c_4) \wedge (c_5 \leq ub_2 \leq c_6)$$

  - **Tree-based ensembles: pass query encoding vectors (e.g., encode '>', '5' for 'a>5') through the traversal of multiple binary trees**



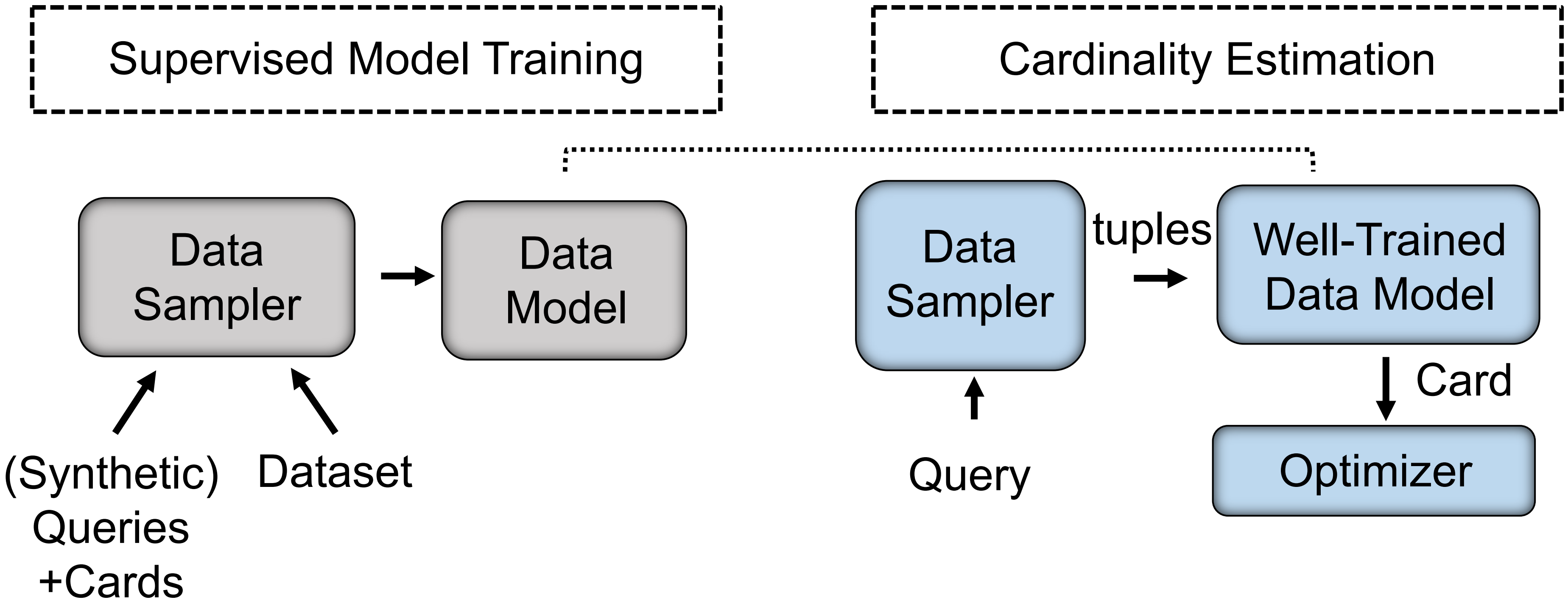
(a) Neural network with 2 hidden layers (b) Tree-based ensembles with 2 trees

A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. PVLDB, 2019.

Supervised Data Methods for Cardinality Estimation

□ Problem Definition

A density estimation problem: learn a joint data distribution of each data point



Supervised Data-Driven: Kernel-Density

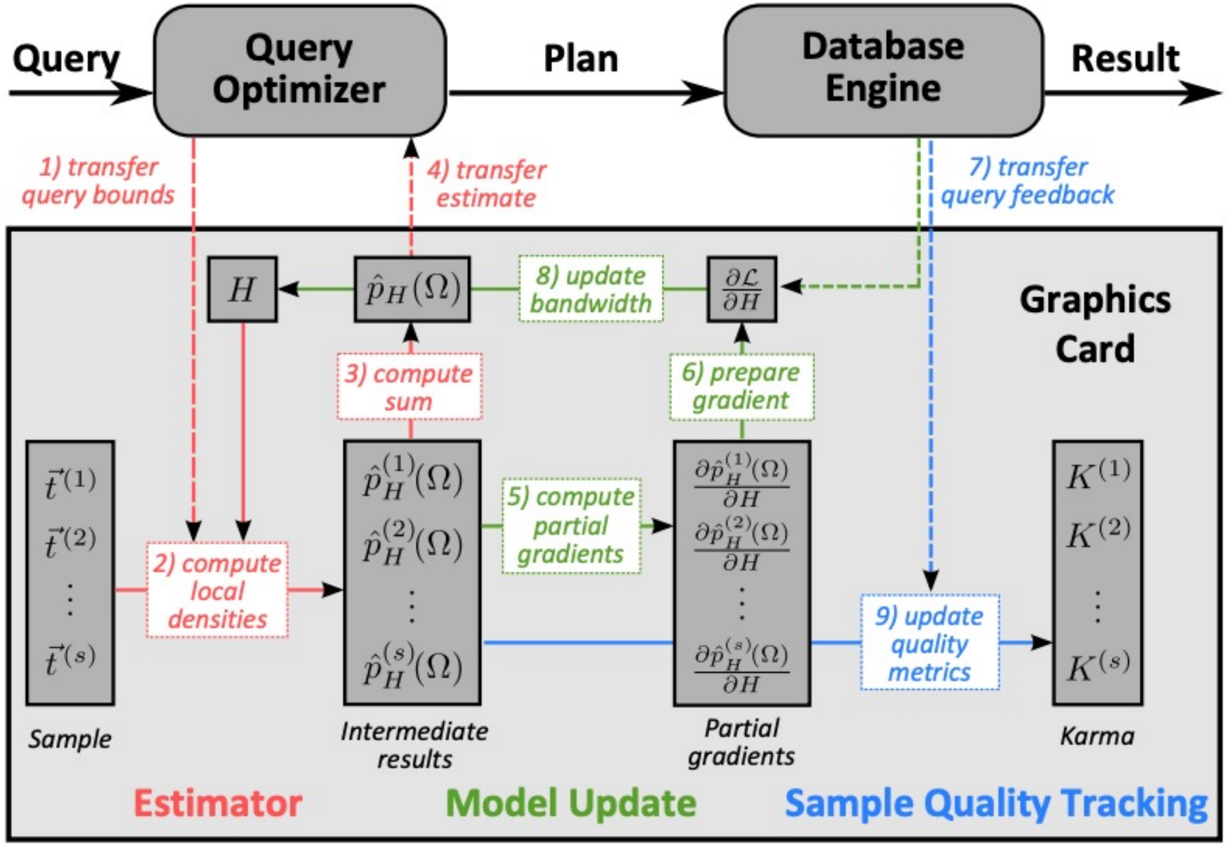
☐ Support point queries on single tables

Training Phase

- ➢ Sample tuples from the table and initialize the **bandwidth** (distance from the true distribution) of the kernel density model.
- ➢ Compute the optimal bandwidth via stochastic gradient descent (with labeled queries).

Inference Phase

- ➢ Sample some new data tuples
- ➢ Estimate the cardinality based on the kernel density model.



M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. SIGMOD, 2015.

Supervised Data-Driven: Mixture Model

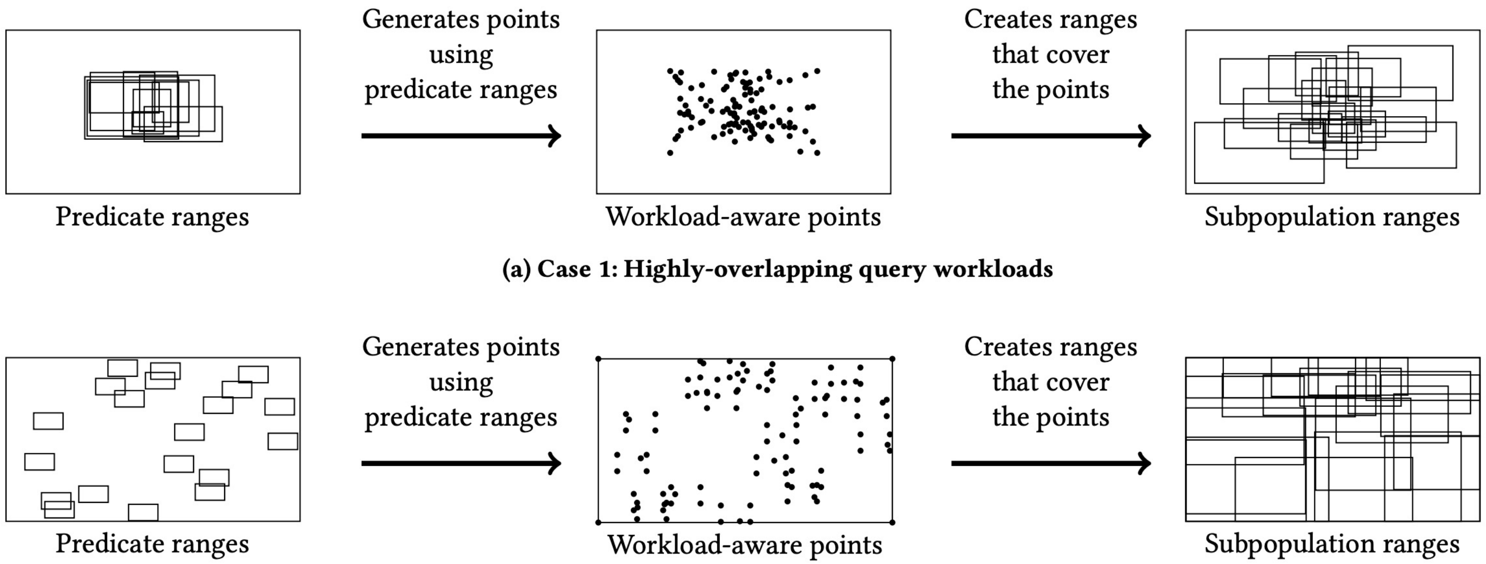
☐ Support Range Queries

Training Phase

- ➢ Sample points within each history queries.
- ➢ Generating **subgroups** for the points.
- ➢ Learn the weights of all the Uniformity Mixture Models for range queries.

Inference Phase

- ➢ Sample tuples within predicate ranges
- ➢ Compute the cardinality by estimating the density of accessed ranges

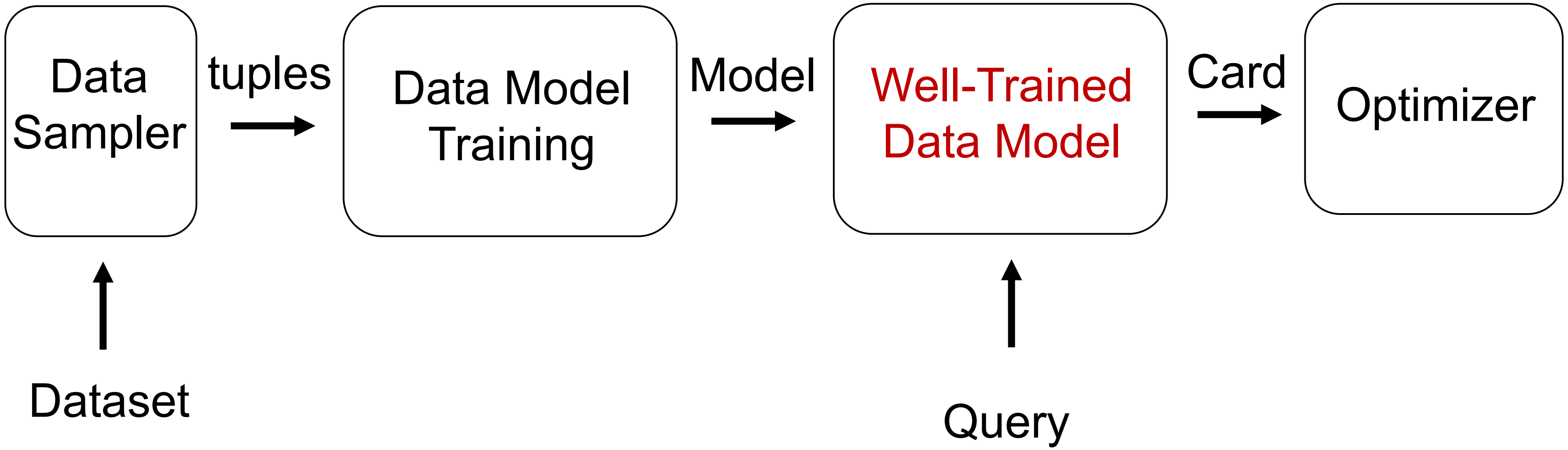


Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020

Unsupervised Data Methods for Cardinality Estimation

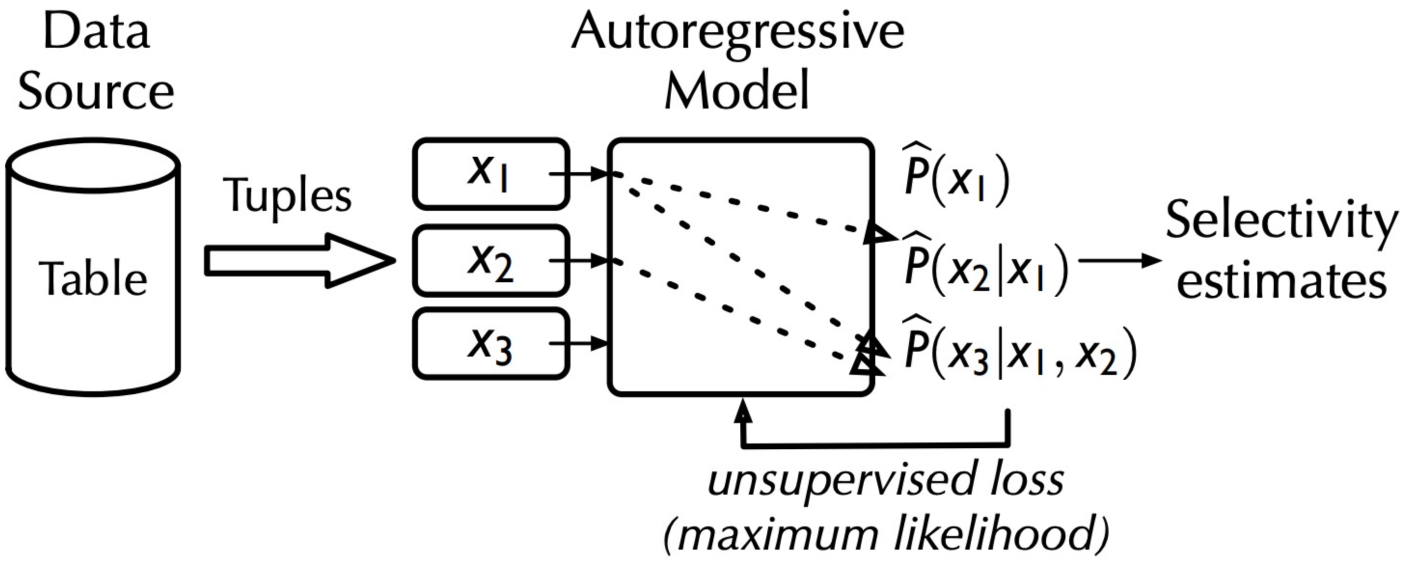
□ Problem Definition

A regression problem: learn a probability function for each data point



Unsupervised Data-Driven: Autoregressive Model

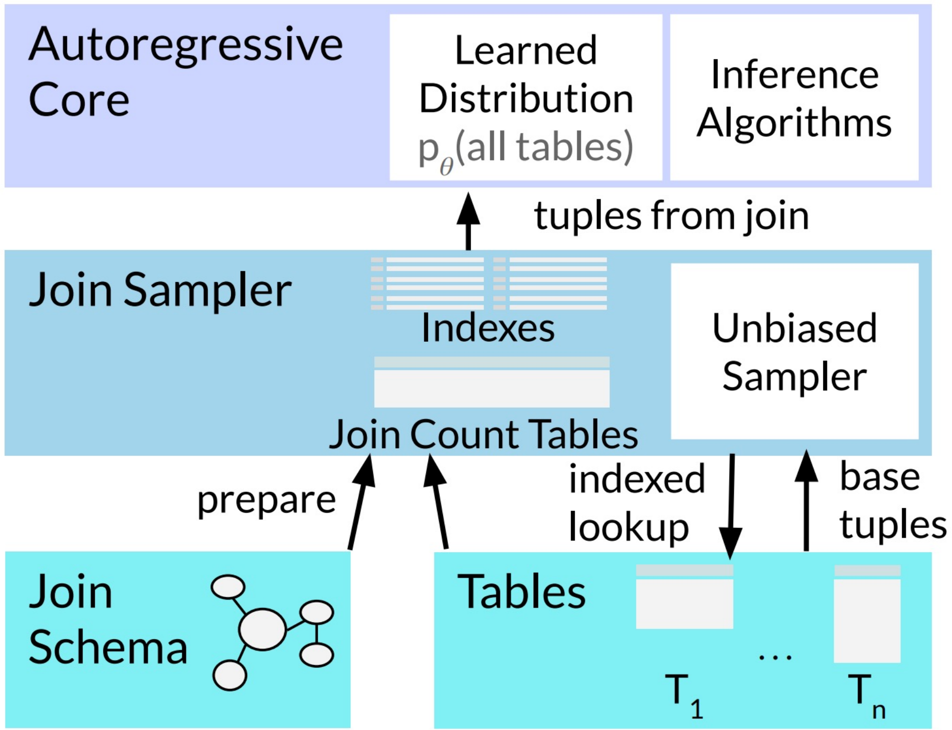
- Learn the joint probability distribution over columns for range queries
  - Use Autoregressive Model to fit the joint probability of different columns
  - Support range query with Progressive Sampling (sample from the estimated data distribution)



1. S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In sigmod, 2020.
2. Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep Unsupervised Cardinality Estimation. PVLDB, 13(3): 279-292, 2019.

Unsupervised Data-Driven: Autoregressive for Multiple Tables

- **Deep AR models can only handle single tables, and we need to learn from join correlations**
  - Learn a single autoregressive model for all the tables (joined)
  - Join Sampler provides correct training data (sampled tuples from join) by using unbiased join counts (reduce sampling time and memory consumption)
  - To estimate for queries with a subset of tables, use fanout scaling to down sample from the joined table



Zongheng Yang, Amog Kamsetty, et al. NeuroCard: One Cardinality Estimator for All Tables. PVLDB, 14(1): 61-73, 2021

Unsupervised Data-Driven: Sum-Product Network

☐ Learn a proper data partitioning strategy to accurately estimate the cardinaltiy

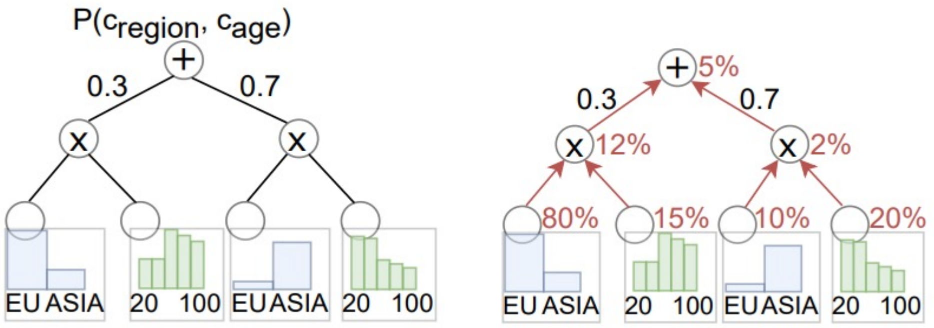
- Split data table into multiple segments and columns in each segment are near independent.
- SPN: Product for different joins and Sum for different filters.
- Adjust the data partitioning (e.g., column splits, column region splits) to learn the accurate SPN models.

c_id	c_age	c_region
1	80	EU
2	70	EU
3	60	ASIA
4	20	EU
...	...	...
998	20	ASIA
998	25	EU
999	30	ASIA
1000	70	ASIA

(a) Example Table

c_age	c_region
80	EU
70	EU
60	ASIA
20	EU
...	...
...	...
20	ASIA
25	EU
30	ASIA
70	ASIA

(b) Learning with Row/Column Clustering



(c) Resulting SPN

(d) Probability of European Customers younger than 30

Summarization of Learned Cardinality Estimation

	Quality	Training Overhead	Training Data	Adaptivity
Deep Neural Network	√√	low	many queries	√√
Tree-Ensemble	√	low	many queries	√
Gaussian Kernel	√	relatively high	Data samples	√√
Mixture Model	√	relatively high	Data samples	√√
Autoregressive	√√	high (join tables)	Data samples	√√√
Sum-Product Network	√√	high	Data samples	√√√

The Relations of Card/Cost Estimation

□ Task Target

- **Cost estimation is to approximate the execution-time/ resource-consumption;**

□ Correlations

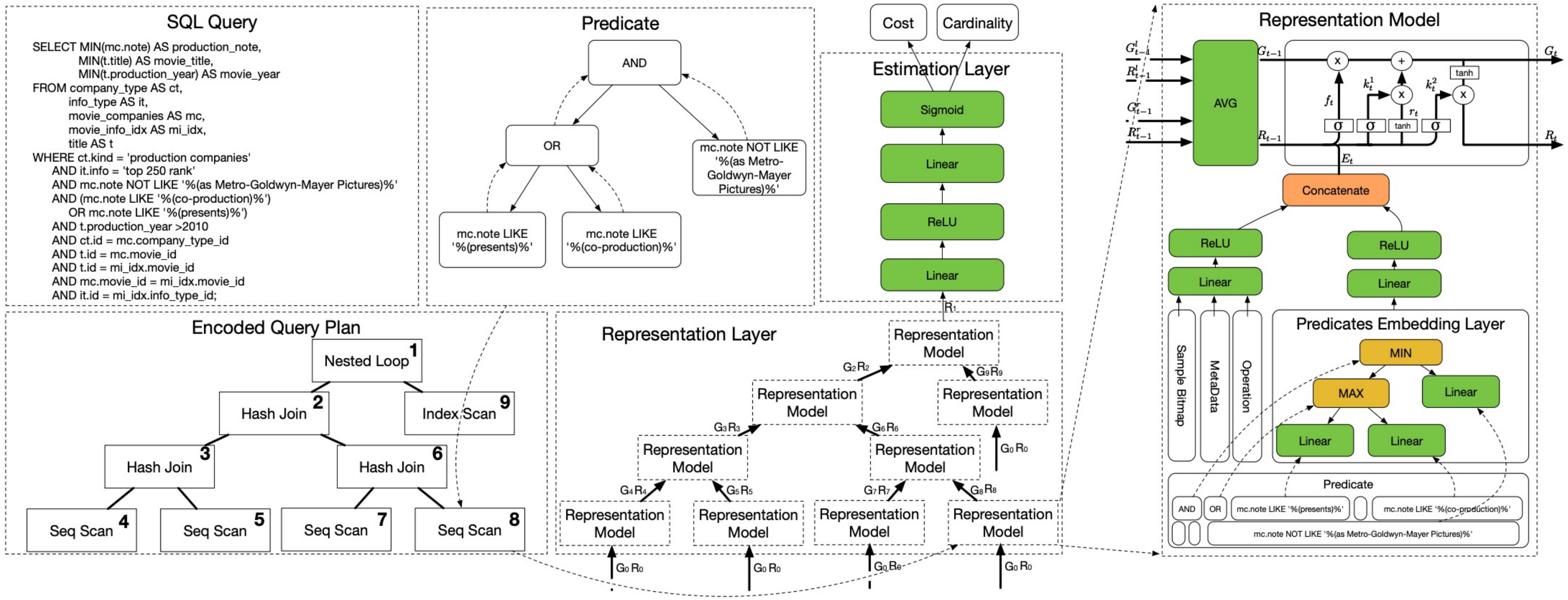
- **Cost estimation is based on cardinality**

□ Estimation Difficulty

- **Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, cpu usage)**

Tree-LSTM for Cost Estimation

- Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding
  - **The representation layer** learns an embedding of each subquery (global vector denotes the subquery, local vector denotes the root operator)
  - **The estimation layer** outputs cardinality & cost simultaneously



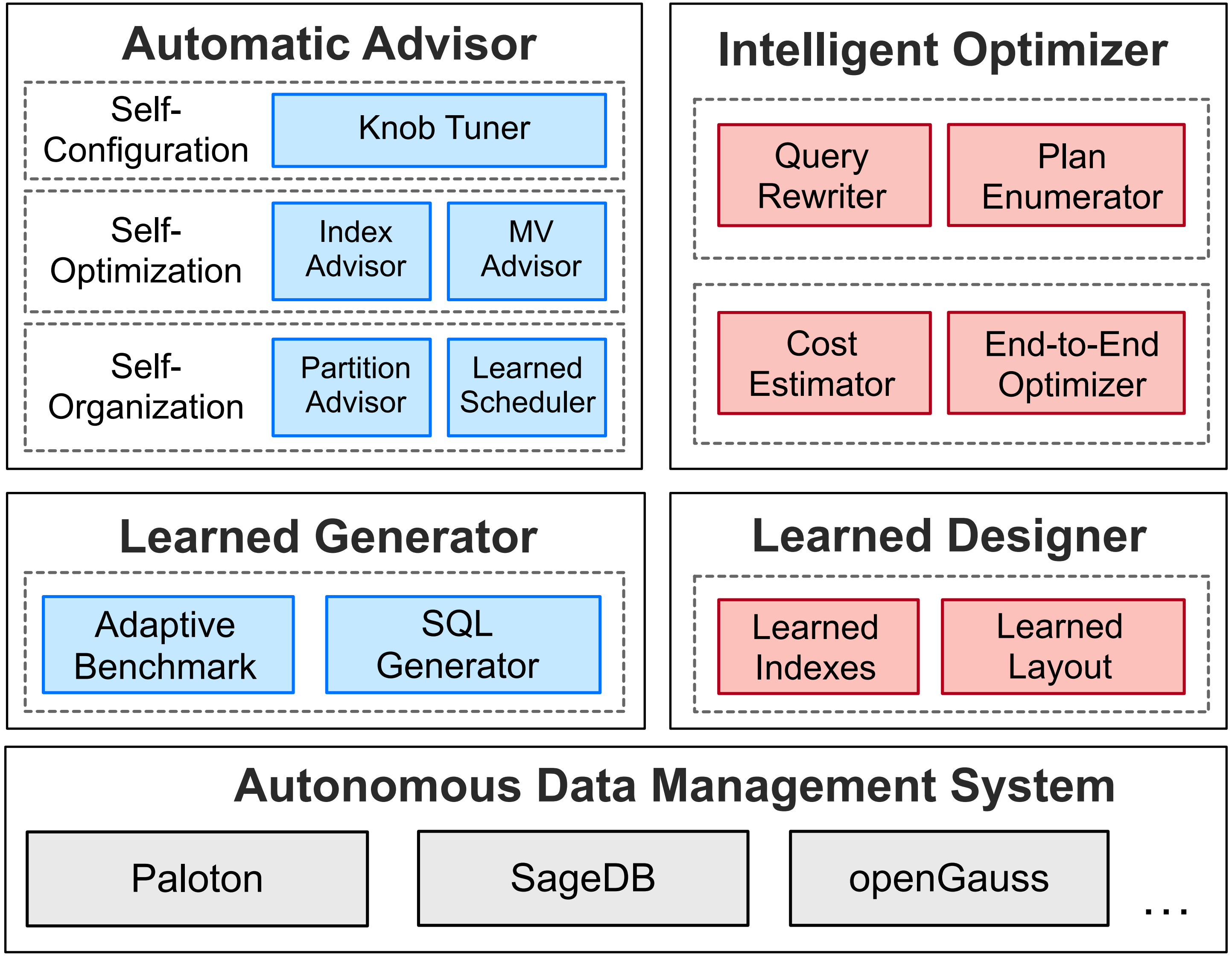
J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Take-aways

- Data-driven methods are more effective for single tables.
- Query-driven methods are more effective for multiple tables.
- Query-driven methods are more efficient than Data-drive methods.
- Data-driven methods are more robust than Query-driven methods.
- Training queries are vital to Query-driven methods.
- Samples are crucial to Data-driven methods.
- Estimators based on neural network are more accurate than statistic-based estimators.
- Statistic-based query model is the most efficient.

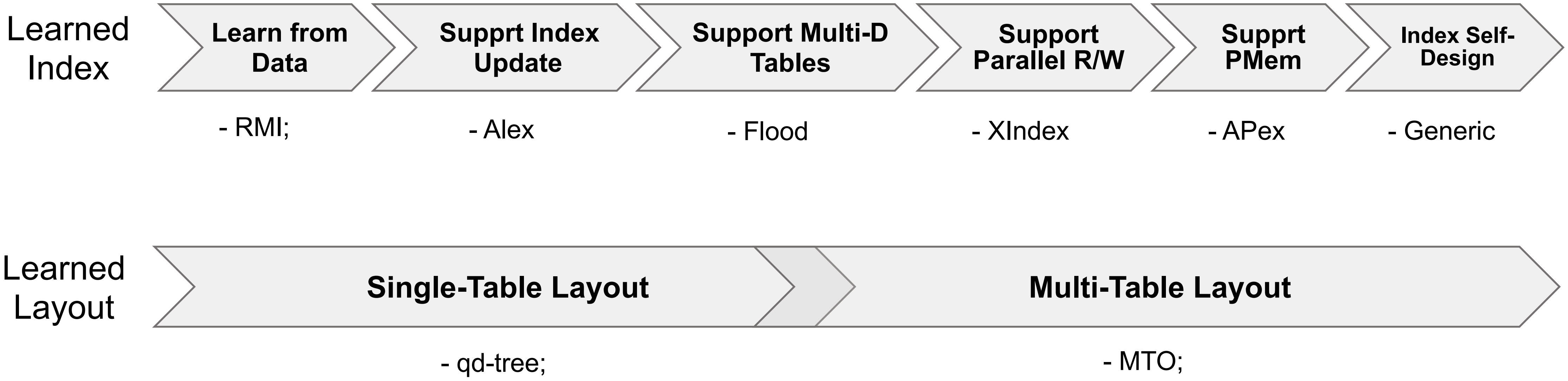
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



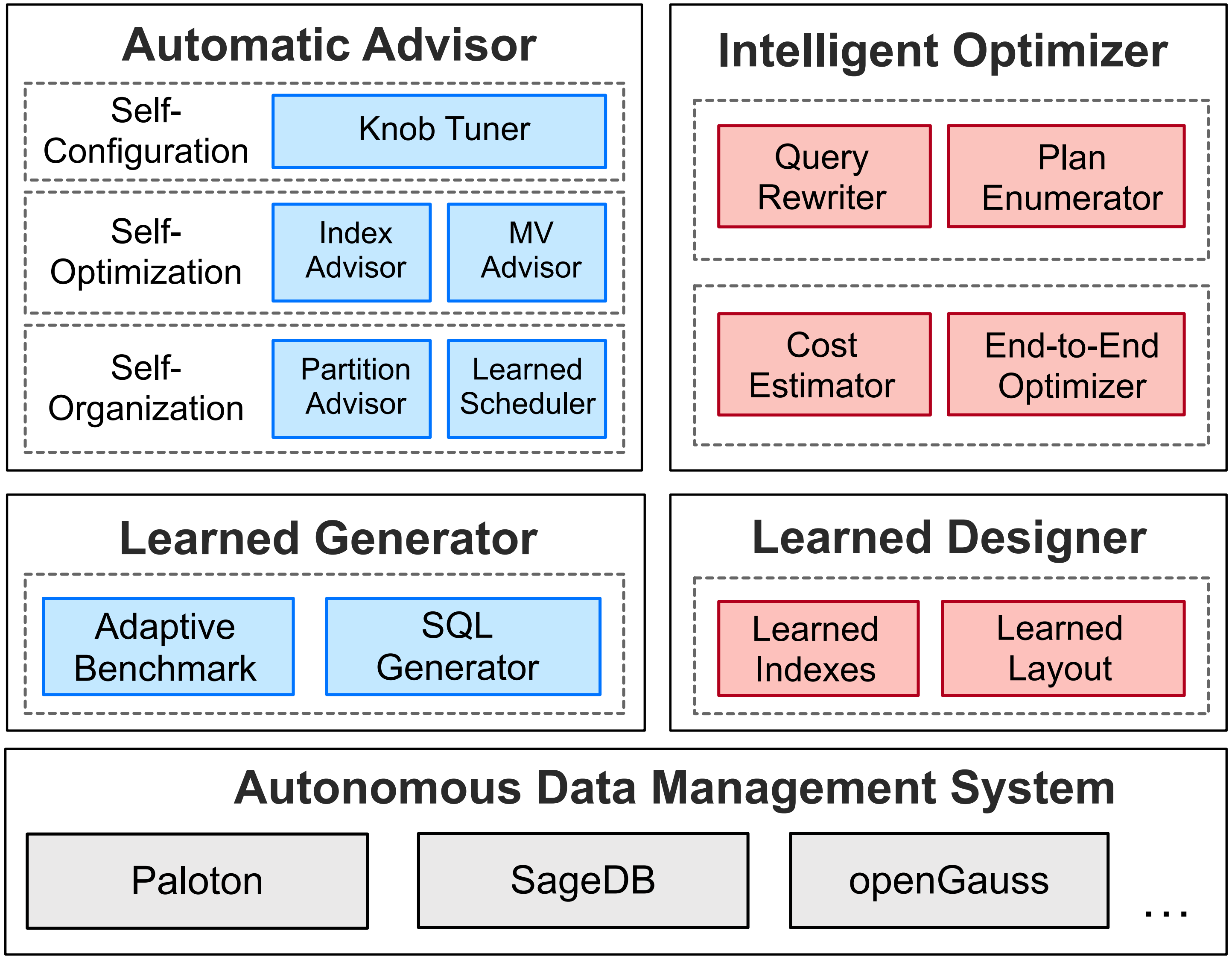
Learned Designer: Technique Development

- **Learned Designer**
  - **Adaptivity (data update)**
  - **Complex Scenarios (e.g., multi-table, concurrency, persistent storage)**



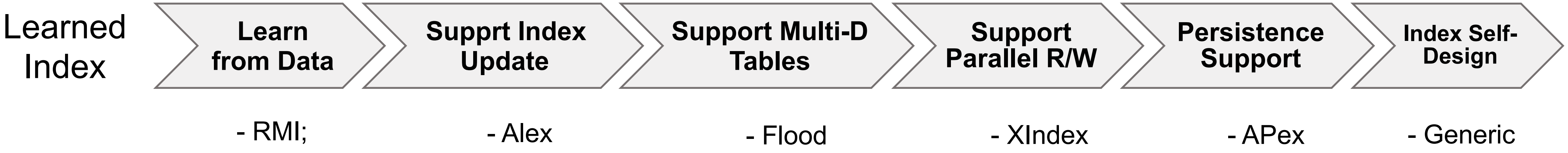
ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Learned Index: Technique Development

- **Learned Designer**
  - **Adaptivity (data update)**
  - **Complex Scenarios (e.g., multi-table, concurrency, persistent storage)**



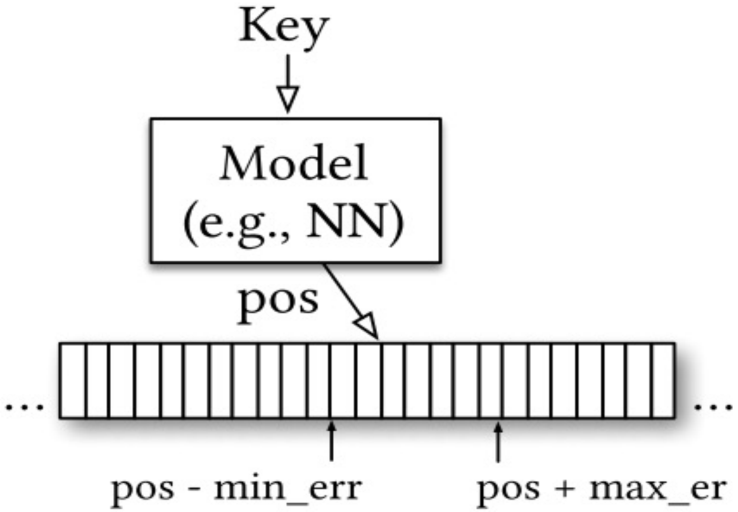
Learned Index

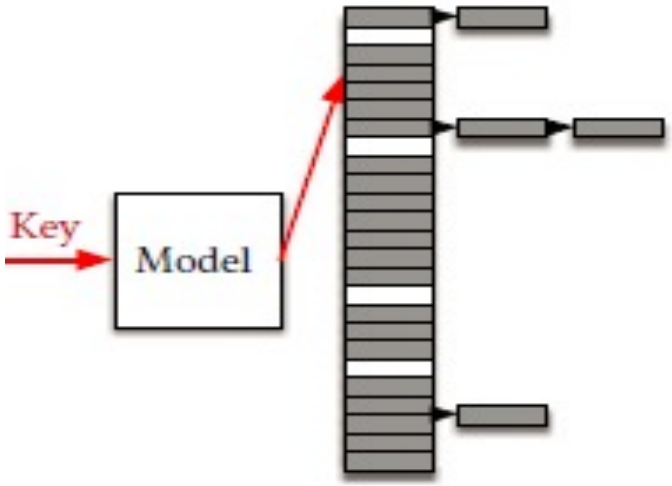
Motivation

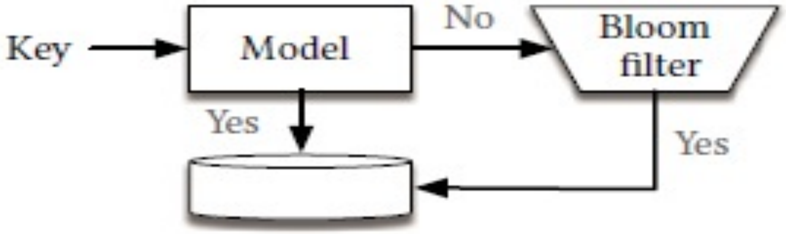
- **Indexes are essential for database system**
  - Indexes significantly speed up query process
  - Take up unignorable memory in huge data-scale situation
- **Limitations in Traditional Index**
  - Unaware of data features
  - Trade-off between Space and Access Efficiency
- **Advantages of Learned Index**
  - Space efficient, only store several parameters
  - Further speed up data access if learning the data distribution well

1-Dimension Immutable Index

- **Model cumulative distribution function (CDF)**
  - **Range index**: predict approximate location as $p = F(Key) * N$, search precise location within error-bounded range
  - **Point index**: CDF as hash function to reduce conflict
  - **Existence index**: add a binary classification model to reduce effective keys of bloom filter



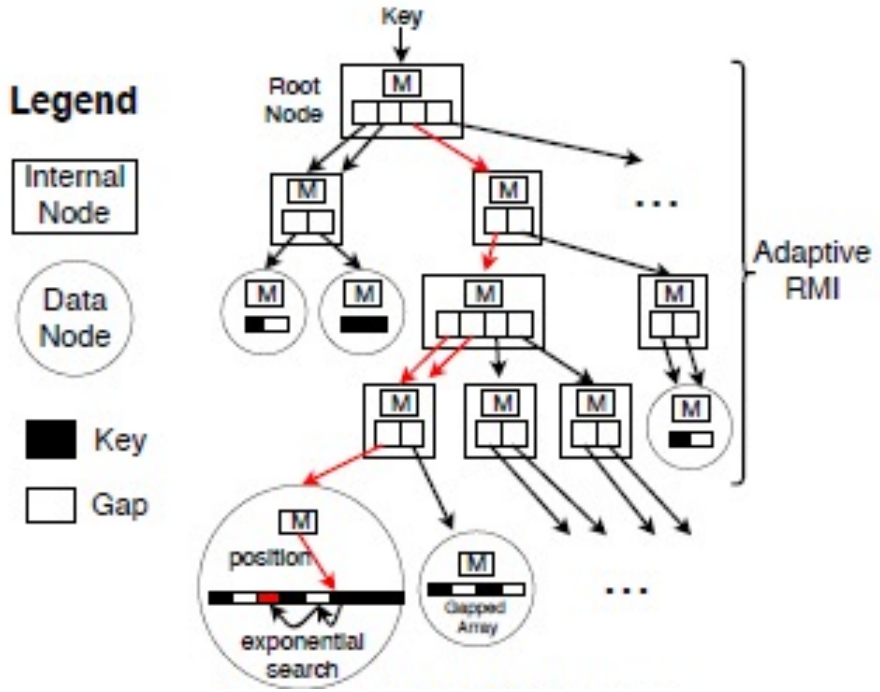




Kraska, T., Beutel, A., Chi, et al. The case for learned index structures. SIGMOD 2018

1-Dimension Mutable Index

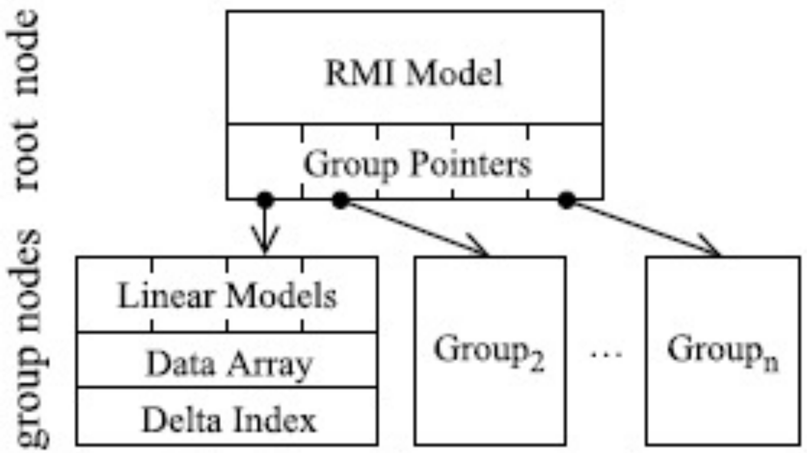
- **Support data inserts and index structure update**
  - Use linear model in each node, exponential search in data node
  - **Gapped array layout**: accelerate inserts (gaps among arrays)
  - **Predict the time to update indexes**: predict costs of read queries and write queries, expand/split data node if cost deviate (given a threshold) to ensure high efficiency

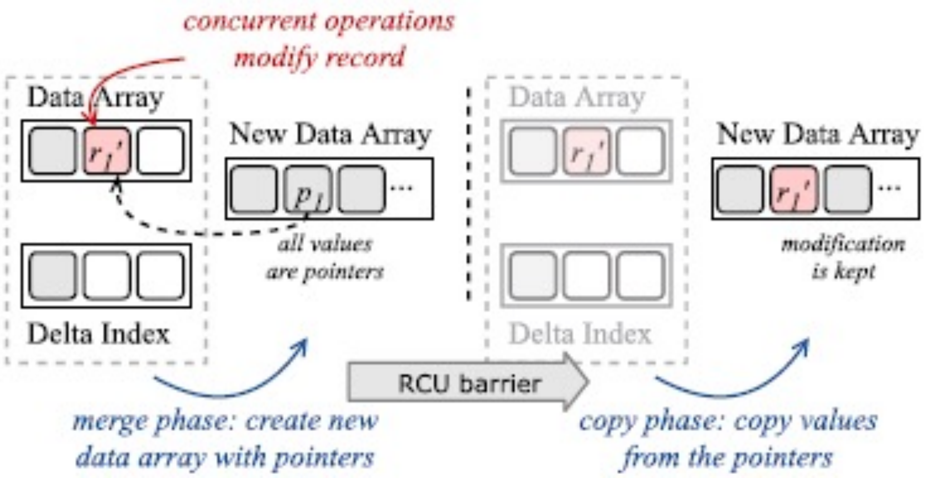


Ding, J., Minhas, U. F., Yu, J., et al. ALEX: An Updatable Adaptive Learned Index. SIGMOD, 2020.

1-Dimension Concurrent Index

- **Handle concurrent updates**
  - **Two update cases**
    - Update in-place,
    - Insert into buffer (delta index)
  - **Approach: 2-phase compaction**
    - Each group has a delta buffer for insertions
    - First merge a group's data and buffer into array of pointers;
    - Then copy the value
    - Similar design for hash index which supports non-blocking resize operations

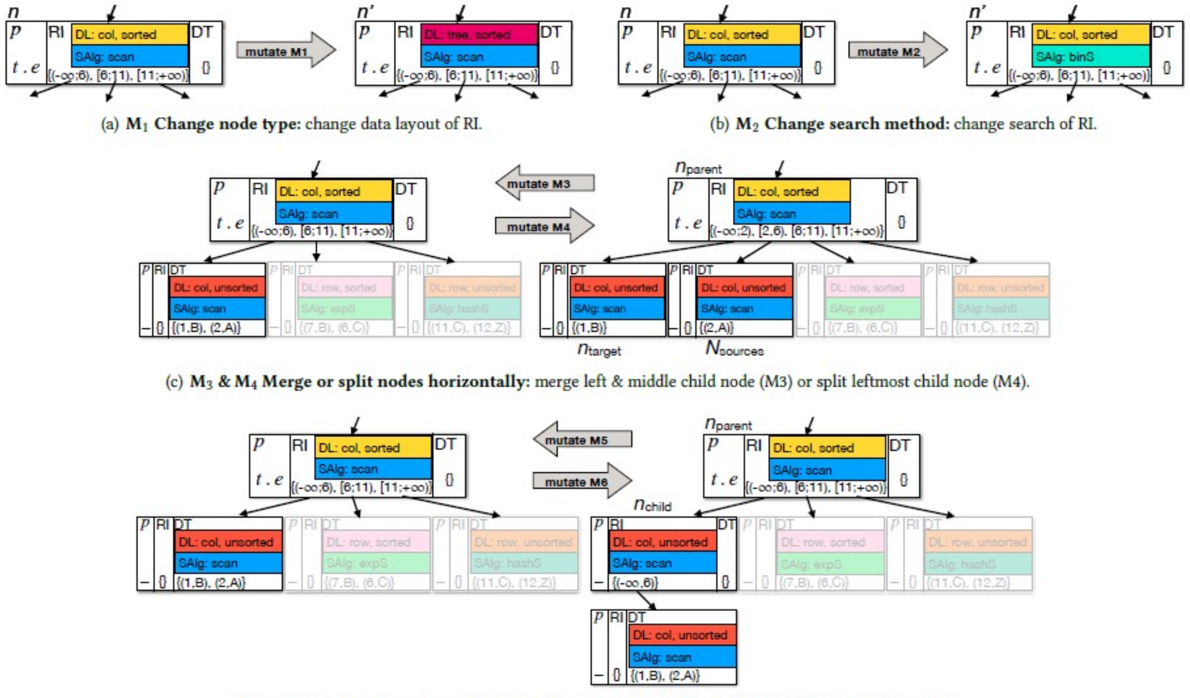




Wang, Z., Chen, H., Wang, Y., et al. The Concurrent Learned Indexes for Multicore Data Storage. ACM Transactions on Storage 2022

1-Dimension Auto-generated Index

- **Use genetic algorithm to optimize indexes from origin data or indexes**
  - **Population**: a set of physical indexes
  - **Mutations**: Adjust the data layout (column/row storage) and searching algorithm (binary/hash …) of a data node; merge/split nodes horizontally and vertically
  - **Fitness function**: optimize indexes for the runtime given a specific workload



(a) M₁ Change node type: change data layout of RI. (b) M₂ Change search method: change search of RI.

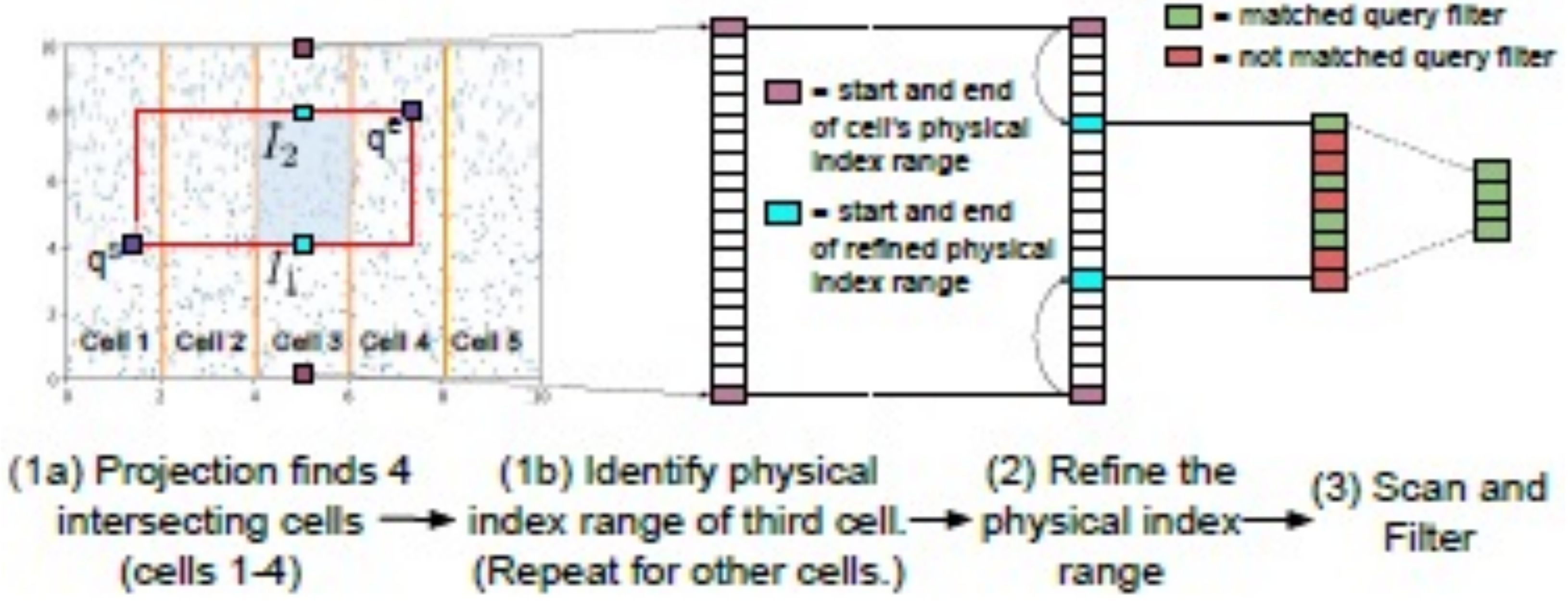
(c) M₃ & M₄ Merge or split nodes horizontally: merge left & middle child node (M3) or split leftmost child node (M4).

(d) M₅ & M₆ Merge or split nodes vertically: merge top-level node's left child (M5) or split it (M6).

Dittrich, J., Nix, J., & Schön, C. The next 50 Years in Database Indexing or: The Case for Automatically Generated Index Structures.. VLDB 2022

Multi-d Immutable Index

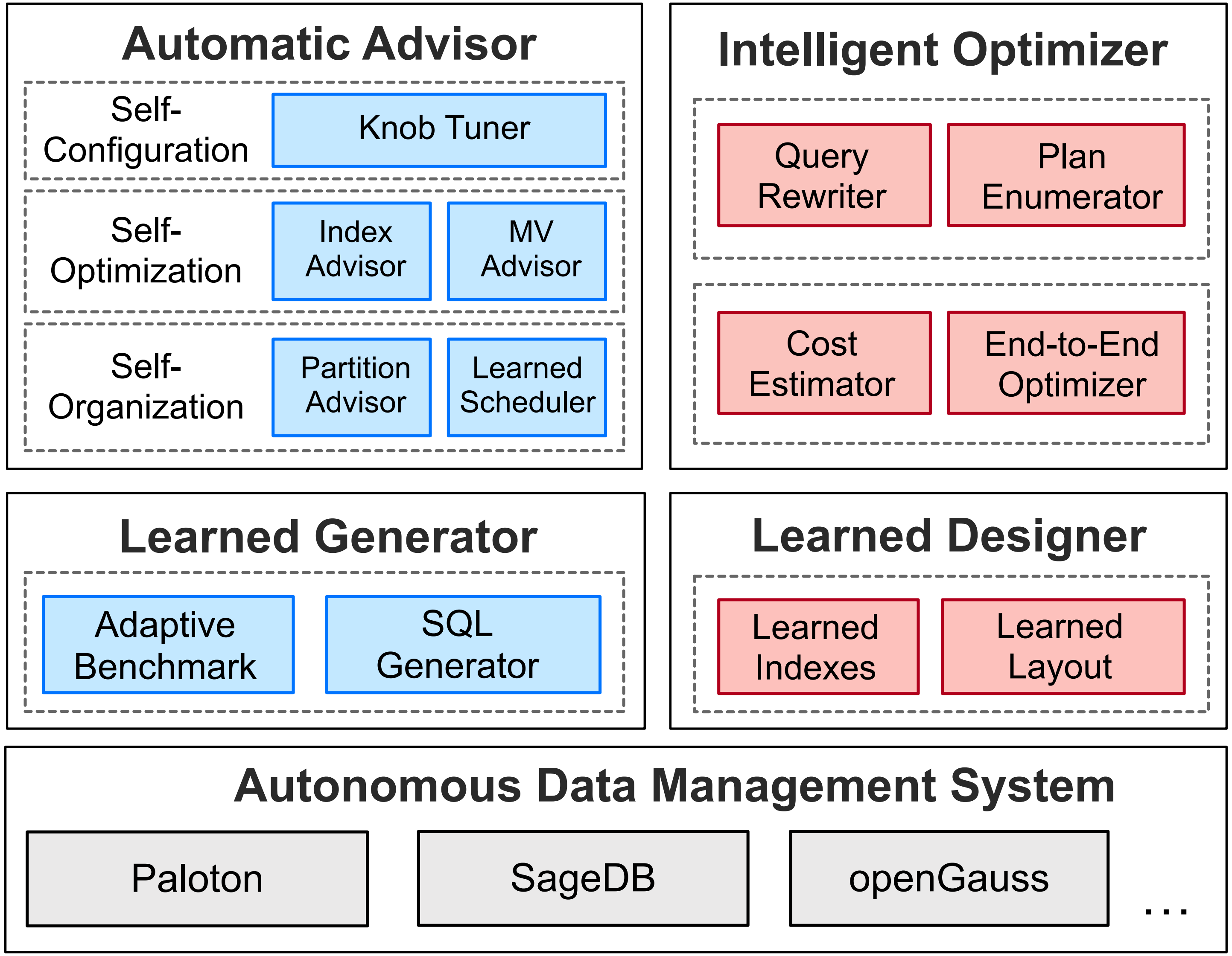
- **Support multi-D index with learned grid index**
  - Sorted cells by 1st, 2nd, … column; within cell, points sorted by the last columns
  - Gradient descent to find the optimal number of regions for each column using sample of dataset and workload
  - Learn CDF of each column to predict region and location within cell



Nathan, V., Ding, J., Alizadeh, M., et al. Learning multi-dimensional indexes. SIGMOD 2020

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Learned Data Layout

Motivation

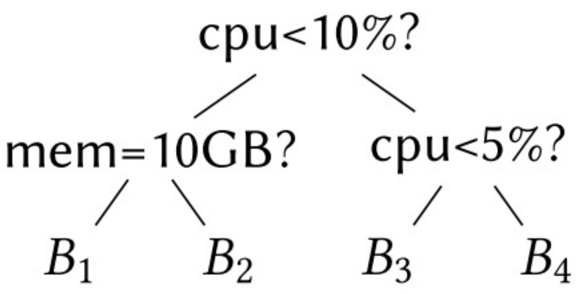
- **To reduce the #-data read from disk**
  - Split data into data blocks (main-memory, secondary storage)
  - in-memory min-max index for each block

- **It is challenging to partition data into data blocks**
  - Numerous ways to assign records into blocks

    Traditional: assign by arrival time; hash/range parititon

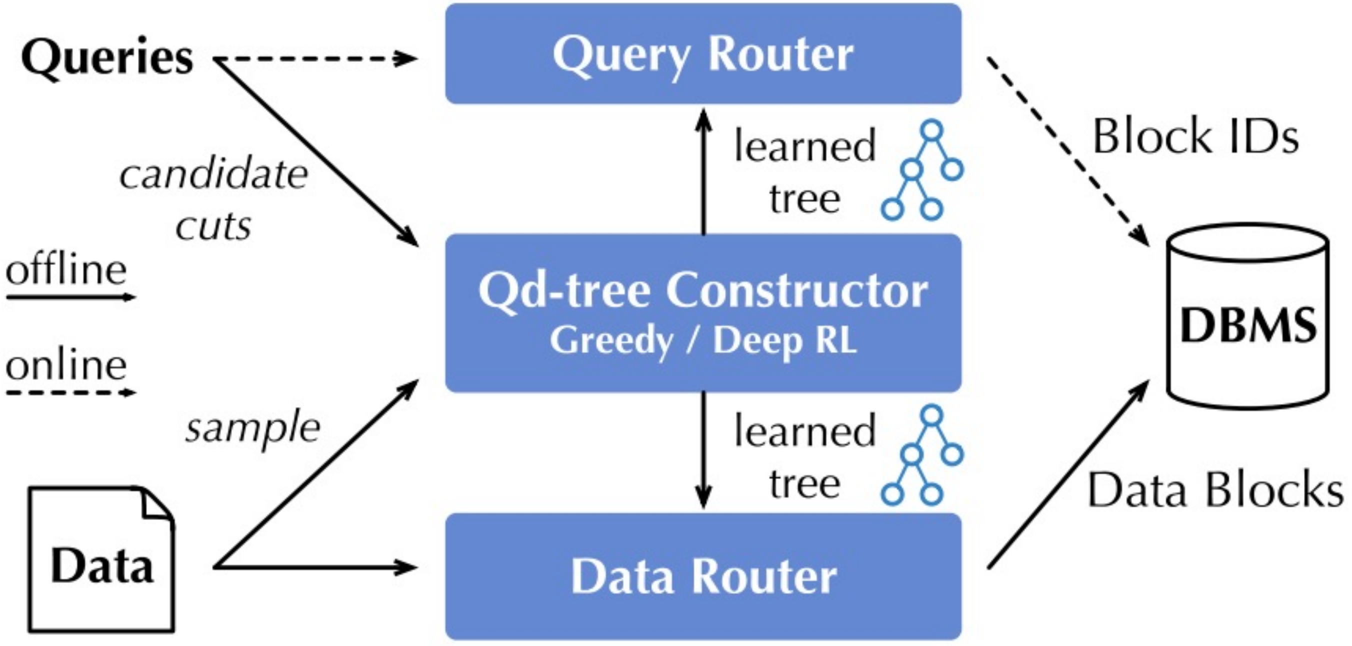
Learned Data Layout (Qd-tree)

- **Qd-tree: Learn the branch predicates**
  - Root Node: The whole data space
  - Other Nodes: A part of the whole space



Example Qd-tree

- **Approach**
  - **Constructor:** Construct a qd-tree based on the workload and dataset (greedy/RL)
  - **Query Router:** Route access requests based on the constructed qd-tree



Zongheng Yang, et al. Qd-tree: Learning Data Layouts for Big Data Analytics. SIGMOD, 2020.

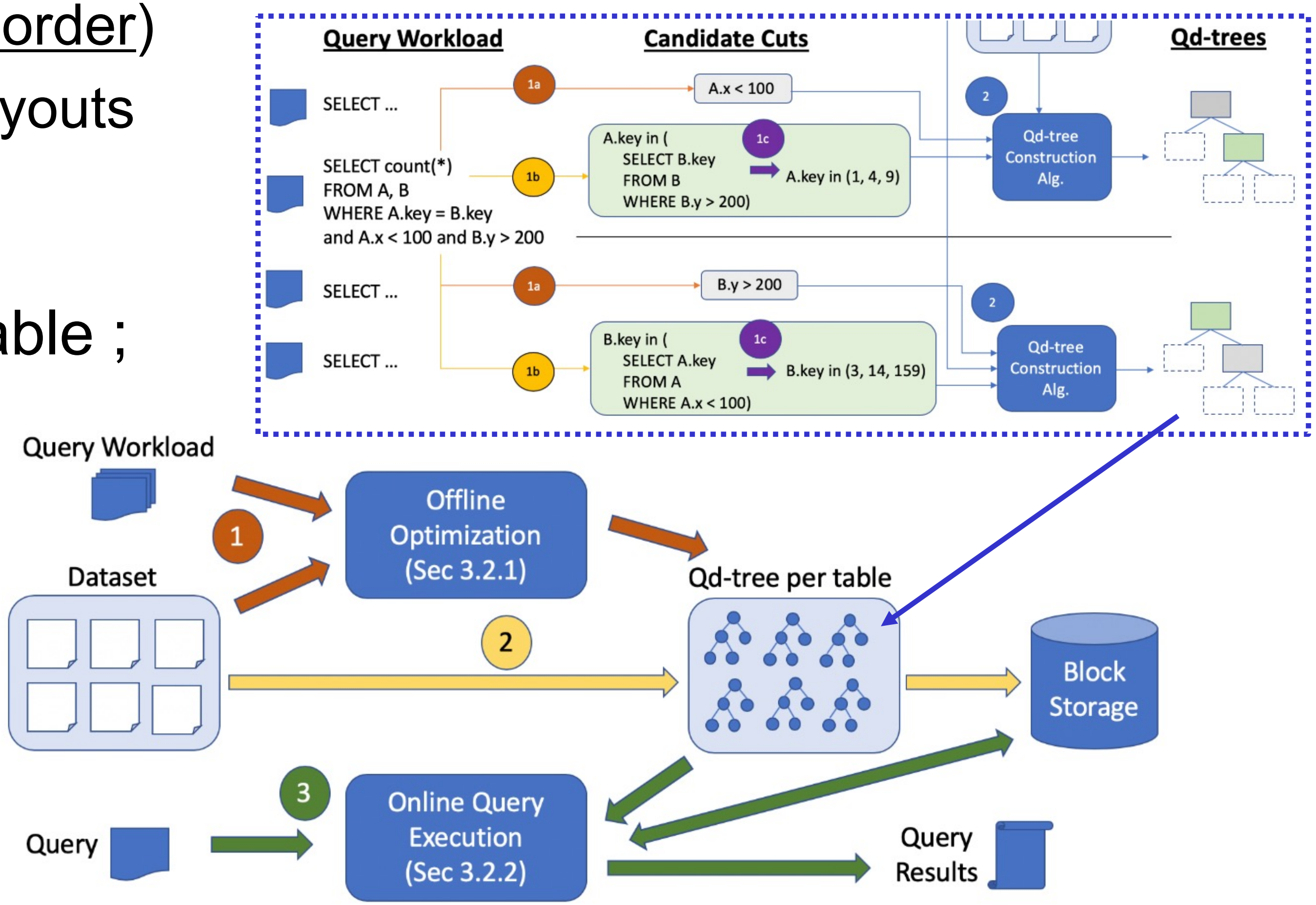
Learned Data Layout: Consider Join Predicates

Motivaiton

- **Traditonal:** either provide rare data skipping (zone maps), or require careful manual designs (Z-order)
- **qd-tree:** only optimize singe-table layouts

Qd-Trees for the whole datasets

- Step#1: Learn qd-tree for each table ;
  - Extract simple predicates;
  - Create join-induced predicates;
  - Induce relevant tuples based on the simple&join-induced predicates
- Step#2: Skip useless blocks based on the qd-trees



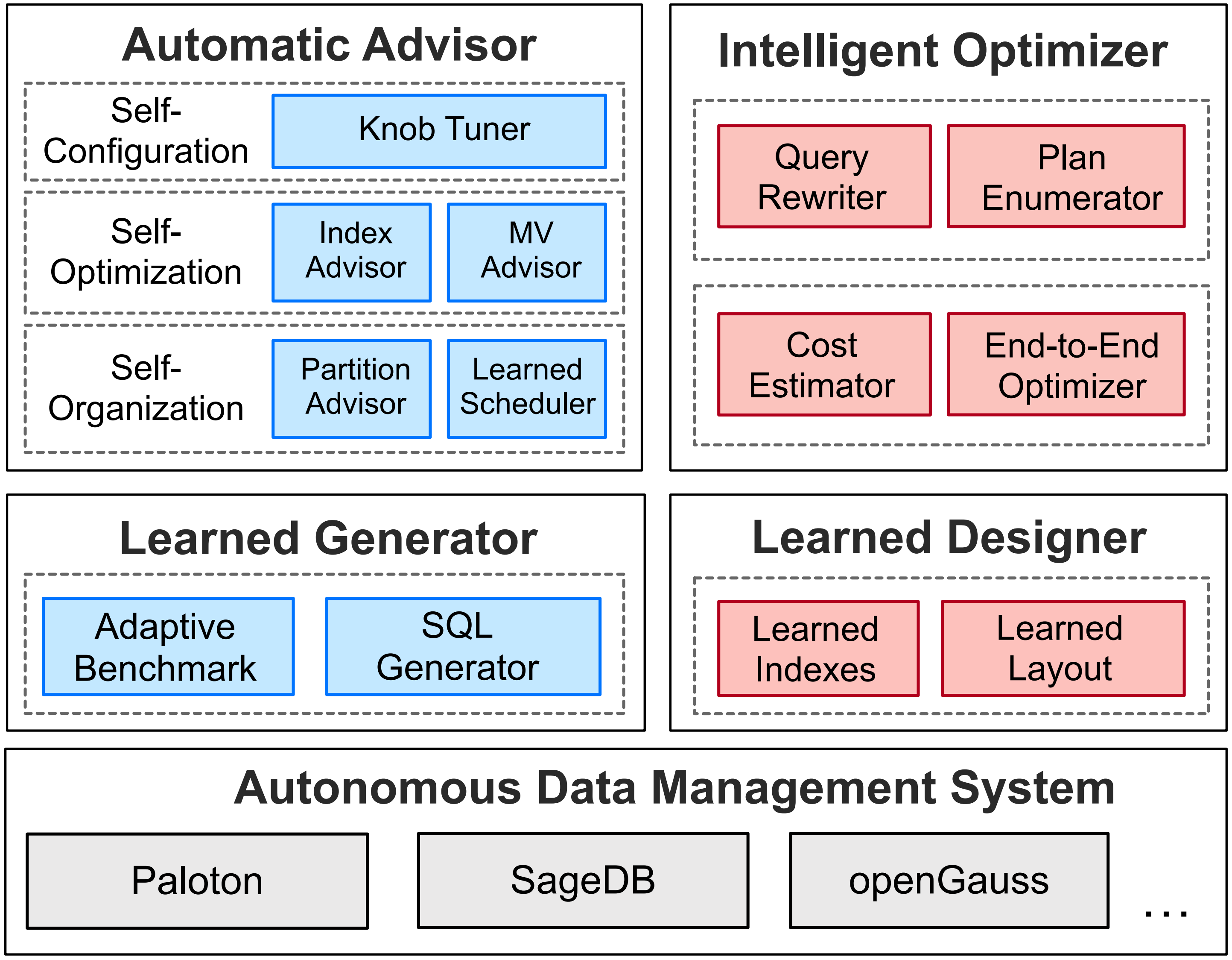
Jialin Ding, et al. Instance-Optimized Data Layouts for Cloud Analytics Workloads. SIGMOD, 2021.

Take-aways of Learned Data Designer

- Learned index opens up a novel idea to replace traditional index, and show good performance in small datasets.
- Learned index uses machine learning technology, which provides probability of combining new hardware like NVM with database system in future.
- Though some research has already verified the benefit of learned index, performance in ***industrial level data scale*** still needs to be studied, especially in ***updatable*** and ***multi-dimension*** situation.
- Open problems
  - Distributed System: **Concurrency Control** algorithms for Learned Index
  - Data scale and stability: Make Learned Index applicable to industrial database systems.

ML4DB: An Overview

- **Automatic Advisor**
  - Knob Tuner
  - Index/View Advisor
  - Partitioner/Scheduler
- **Learned Generator**
  - SQL Generator
  - Adaptive Benchmark
- **Intelligent Optimizer**
  - Query Rewriter
  - Plan Enumerator
  - Cost Estimator
- **Learned Designer**
  - Learned Index
  - Learned Data Layout
- **Autonomous Databases**



Autonomous Database Systems

Motivation

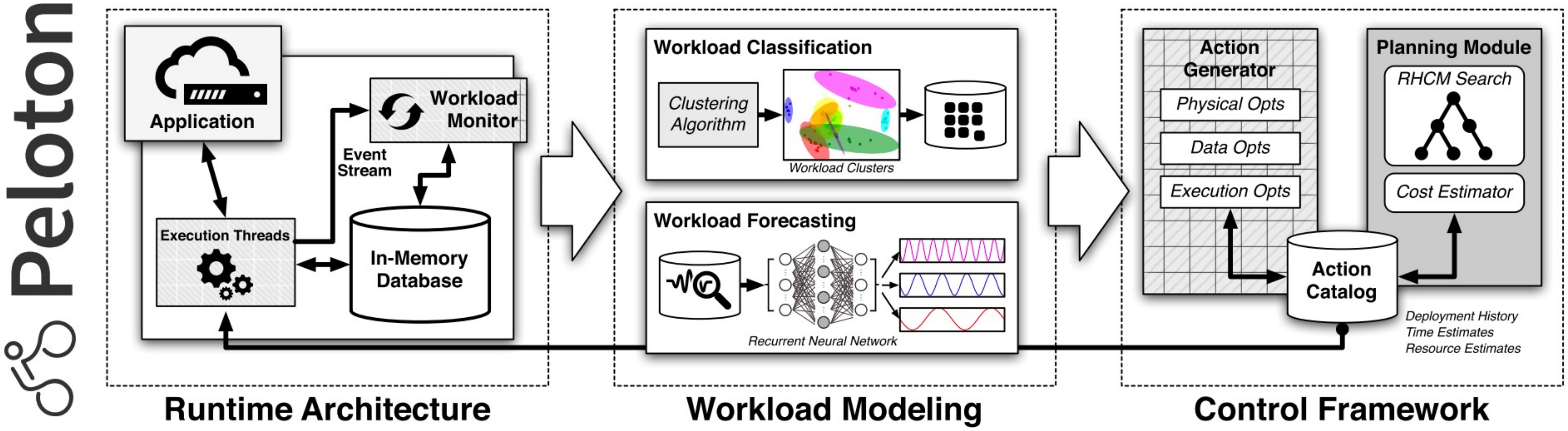
- **Traditional Database Design is laborious**
  - Develop databases based on workload/data features
  - Some general modules may not work well in all the cases

- **Most AI4DB Works Focus on Single Modules**
  - Local optimum with high training overhead

- **Commercial Practices of AI4DB Works**
  - Heavy ML models are hard to implement inside kernel
  - A uniform training platform is required

Peloton

- **Adapt optimization actions based on forecasted workloads**
  - **Embedded Monitor:** Detect the event stream and extract incoming queries
  - **Workload Forecast:** Cluster queries and forecast for each cluster with RNN
  - **Optimization Actions:** Physical design, data layouts, and config tuning



Andy Pavlo, et al. Self-Driving Database Management Systems. In CIDR, 2017.

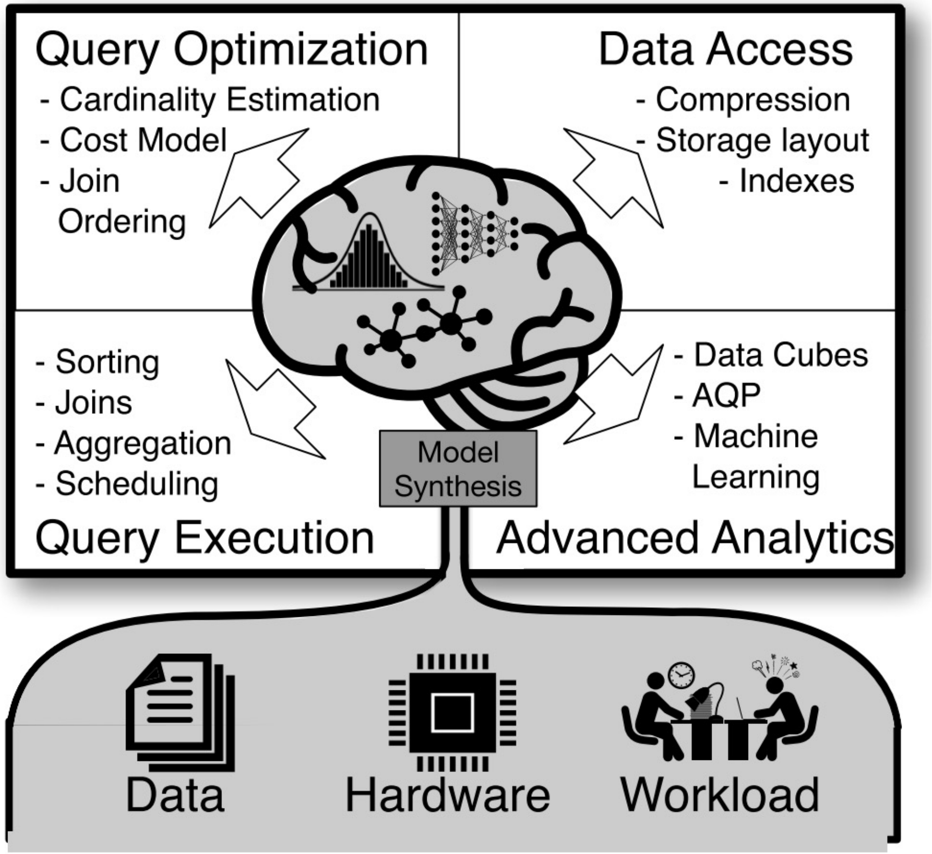
SageDB

☐ Customize DB design via learning the data distribution

- ➢ Learn Data Distribution by Learned CDF

$$M_{CDF} = F_{X_1,...X_m}(x_1, ...x_m) = P(X_1 \leq x_1, ..., X_m \leq x_m)$$

- ➢ Design components via learned CDFs, and utilize cost-saving ML to replace traditional operators (e.g., seq scans)
  - Query optimization and execution
  - Data layout design
  - Advanced analytics

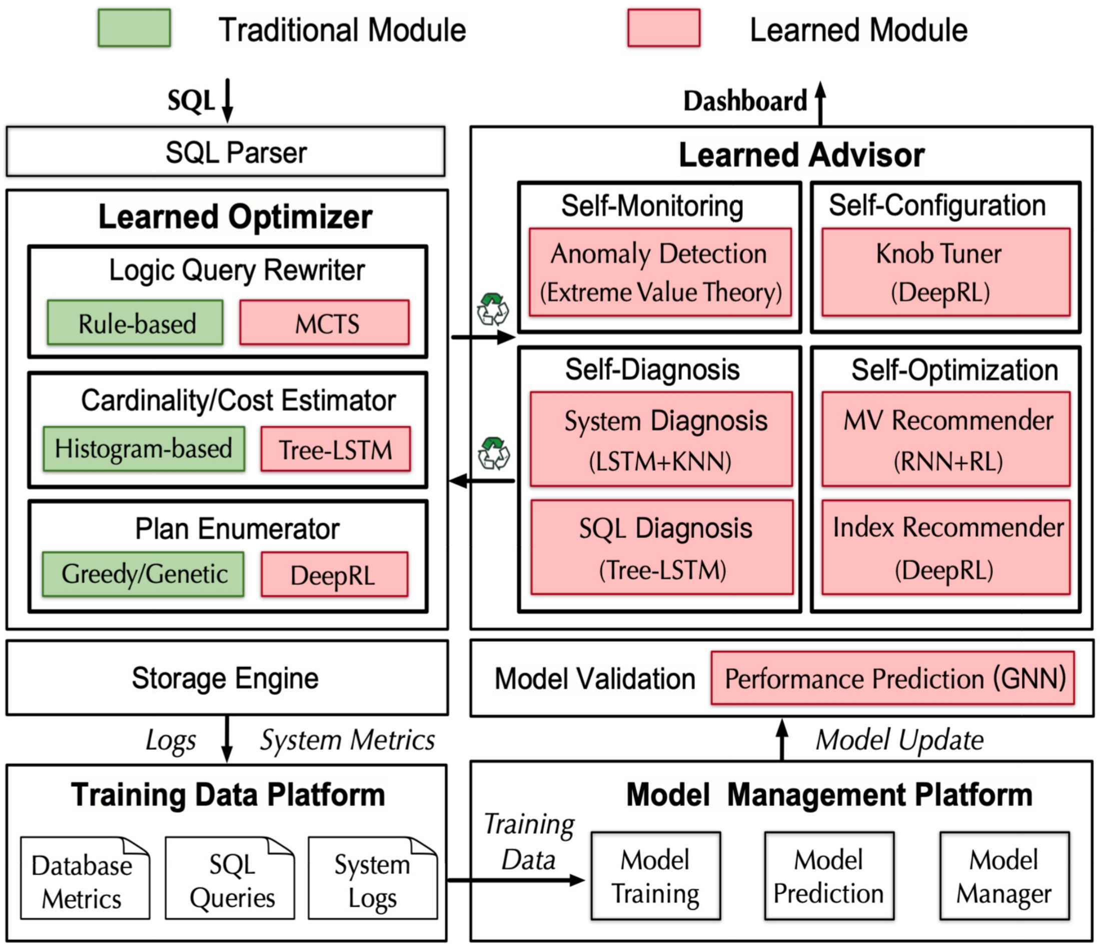


Tim Kraska, et al. SageDB: A Learned Database System. In CIDR, 2019.

openGauss

☐ Implement learned components with model validation

- **Learned Optimizer**
  - Query Rewriter
  - Cost/Card Estimator
  - Plan Enumerator
- **Learned Advisor**
  - Self-Monitoring
  - Self-Diagnosis
  - Self-Configuration
  - Self-Optimization
- **Model Validation**
- **Data/Model Management**



Guoliang Li, et al. openGauss: An Autonomous Database System. In VLDB, 2021.

Research Challenges

Future Works: Optimization Overhead

- **Cold-Start Problems**
  - Across datasets / instances / hardware / database types
- **Lightweight in-kernel components**
  - Efficient ML models; rare-data/compute-dependency;
- **Online Optimization**
- **Workload execution overhead**
- **Model training overhead**

Future Works: Adaptivity

- **Significant data changes**
  - Migration from small datasets to large datasets
- **Completely new instances**
  - New dataset, workload, and SLA requirements;
- **Incremental DB module update**
  - Learned knob tuner for hardware upgrade, learned optimizer for dynamic workloads.

Future Works: Complex Scenarios

- **Hybrid Workloads**
  - HTAP, dynamic streaming tasks
- **Distributed Databases**
  - Distributed plan optimization
- **Cloud Databases**
  - Dynamic environment, serverless optimization

Future Works: Small Training Data

- **Few Training Samples**
  - Few-shot learning
- **Knowledge + Data-driven**
  - Summarize (interpretable) experience from data
- **Pre-Trained Model**
  - Train a model for multiple scenarios

Future Works: SLA Improvement

- **Optimize databases under noisy scenarios**
  - Training Data Cleaning, Model Robust
- **Optimize for extremely complex queries (e.g., nested queries)**
  - Adaptive cardinality estimation → efficient query plan
- **Optimize for OLTP queries**
  - Multiple query optimization

Thanks