Artificial Intelligence Meets Database

Guoliang Li

Tsinghua University

Motivation

Artificial Intelligence Meets Database

AI4DB

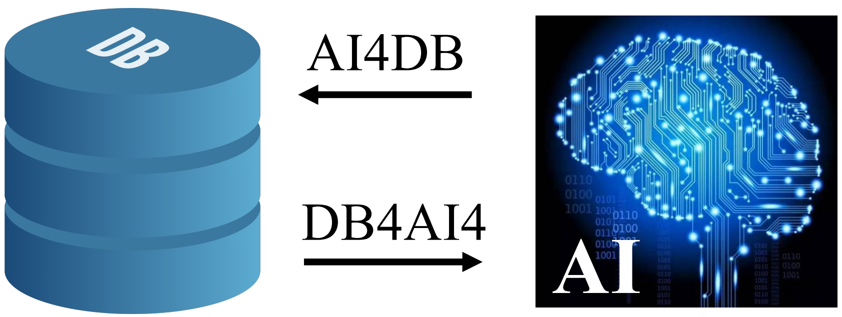
Manual → Automatic

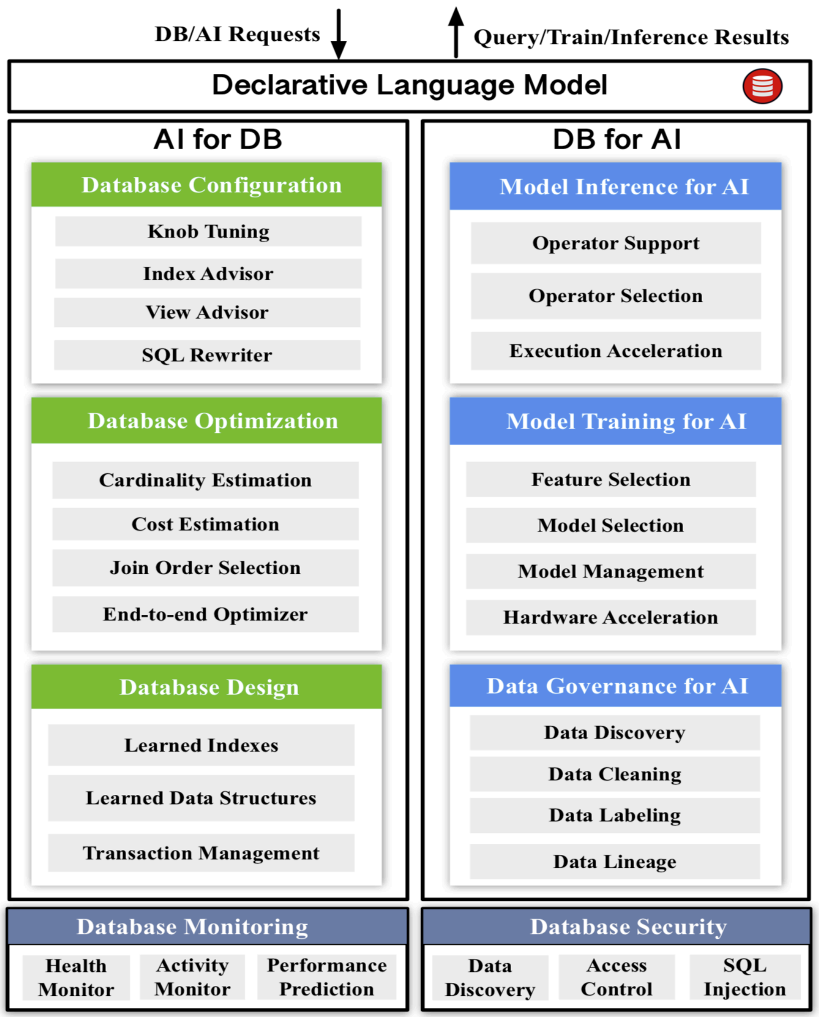
- ☐ Self-optimization
- ☐ Self-configuration
- ☐ Self-monitoring
- ☐ Self-healing
- ☐ Self-security
- ☐ Self-design

DB4AI

AI → as easy as DB

- ☐ Declarative AI
- ☐ AI optimization
- ☐ Data governance
- ☐ Model management
- ☐ AI+DB hybrid model
- ☐ AI+DB hybrid inference





Revisit DB Systems: Driving Factors

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

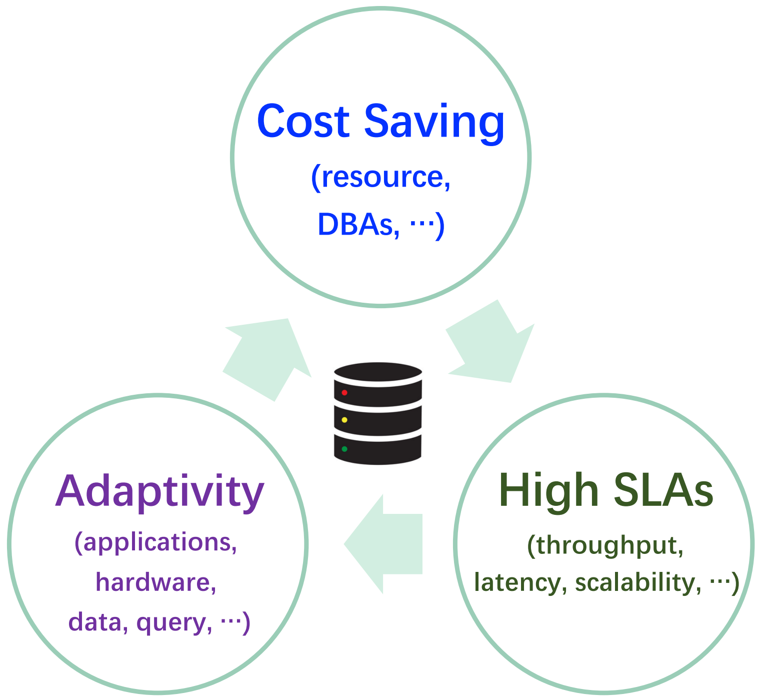
NoSQL: data models, indexing, partitioning, …

transactions +big data ↓ **H**igh Scalability; **ACID**

Distributed OLTP: 2PC, Paxos, Distributed SQL…

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

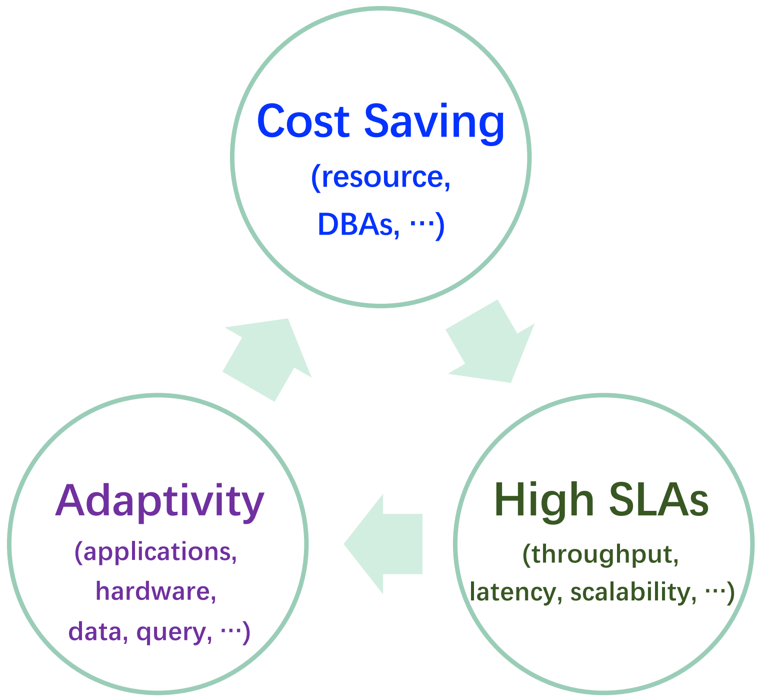
Cloud-Native: compute-storage disaggregation, …



Learned Database

New Opportunities: What Can AI Bring for DB?

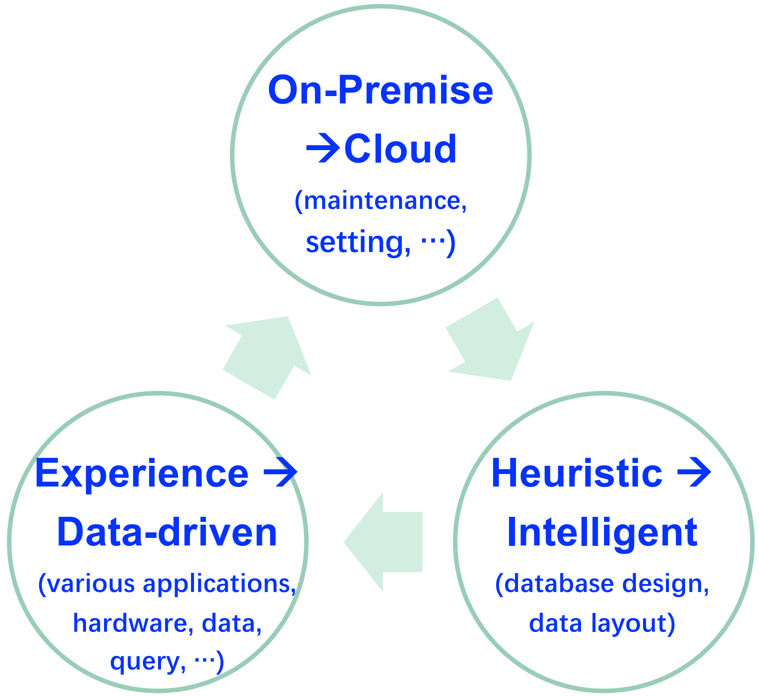
- **Cost Saving: Manual → Autonomous**
  - Auto Knob Tuner: ↓ Maintenance cost
  - Auto Index Advisor: ↓ Optimization latency
- **High SLAs: Heuristic → Intelligent**
  - Intelligent Optimizer: ↓ Query plan costs
  - Intelligent Scheduler: ↑ Workload performance
- **Adaptivity: Empirical → Data-Driven**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

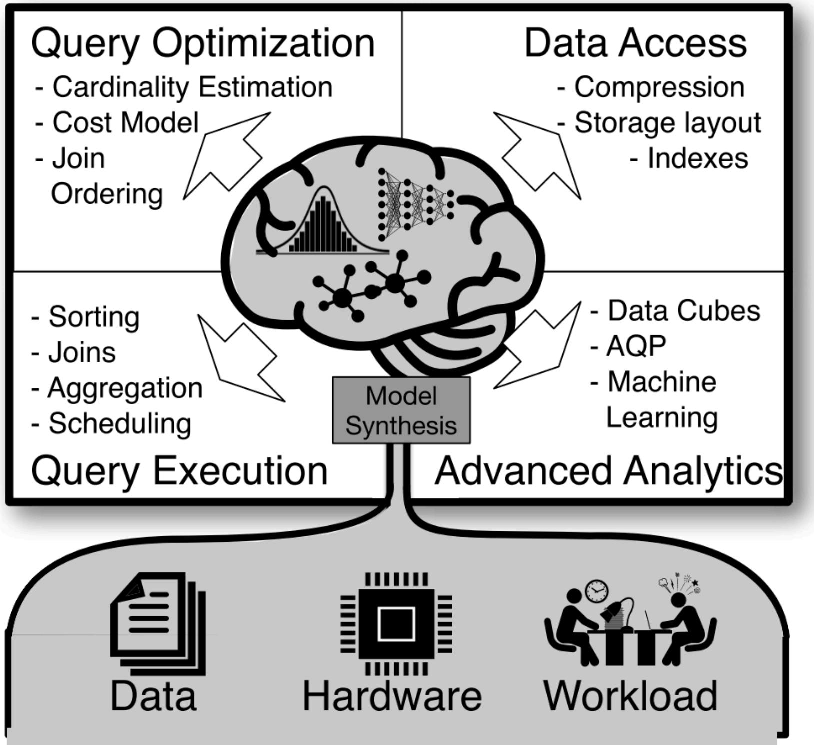
- **Feature Selection:** Pick relevant features from numerous query / database / os metrics ;
- **Model Selection:** Design ML models to solve different database problems;
- **Diverse Targets:** Meet the SLA requirements under different scenarios;
- **Adaptivity**
- **Training Data**

AI4DB Techniques: Motivation

☐ Learned Database Kernel

- Cardinality/Cost Estimation, Query Rewrite, Plan Generation

●Manual → Autonomous



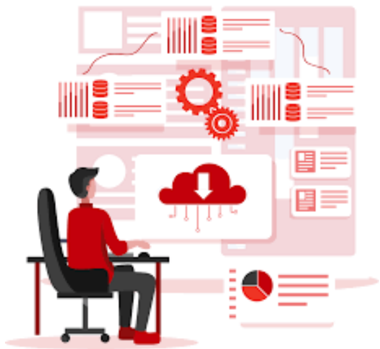
●Heuristic → Intelligent

●Empirical → Data-Driven

AI4DB Techniques: Motivation

□ Learned Database Configuration

- Automate database configurations, e.g., DRL for knob tuning, binary classifier for index selection.



✗ Labor-intensive tuning

✗ Time-consuming tuning

✓ Rich tuning experience

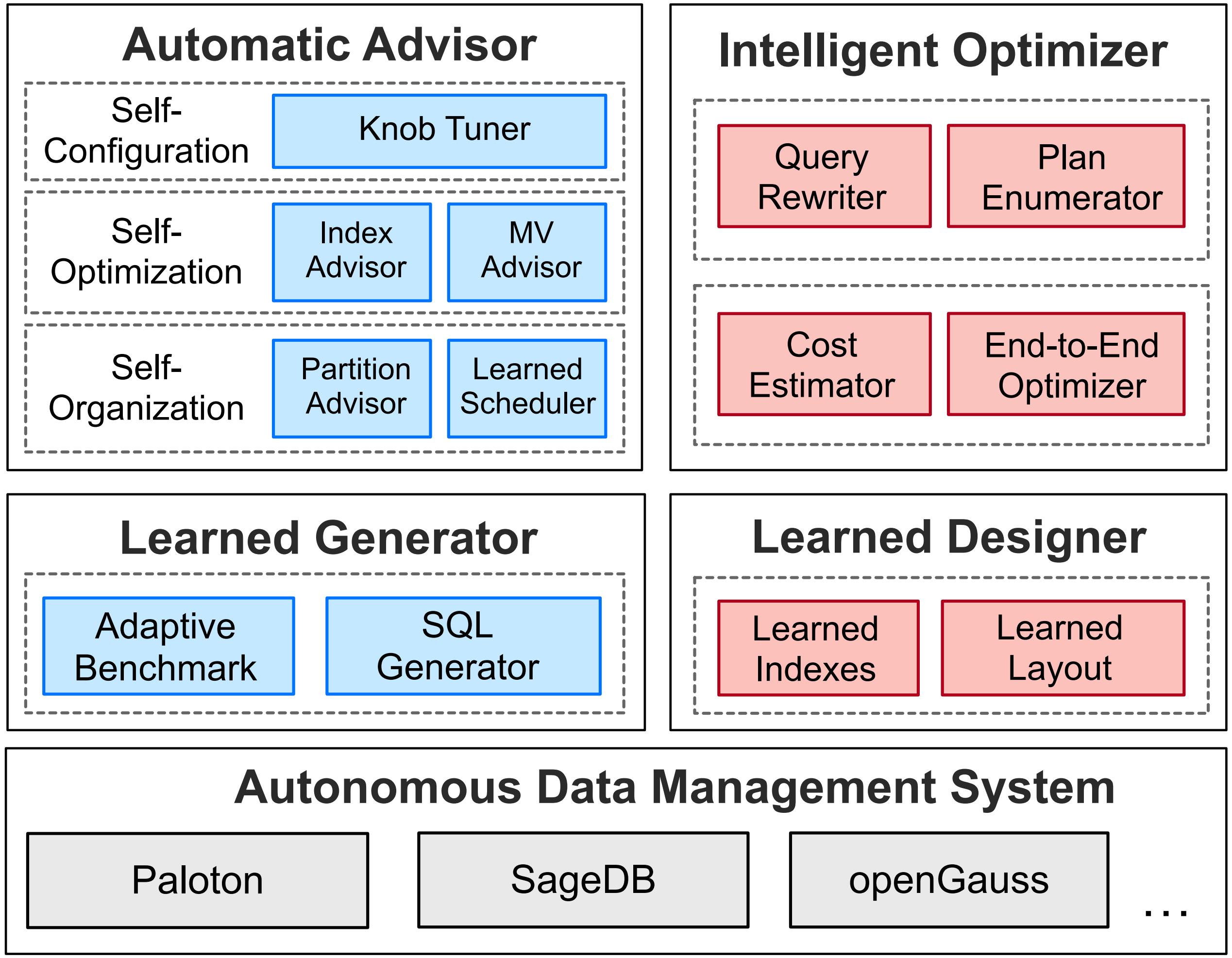
✓ Automatic tuning

✓ Low tuning latency

✗ Adaptivity for different DBs

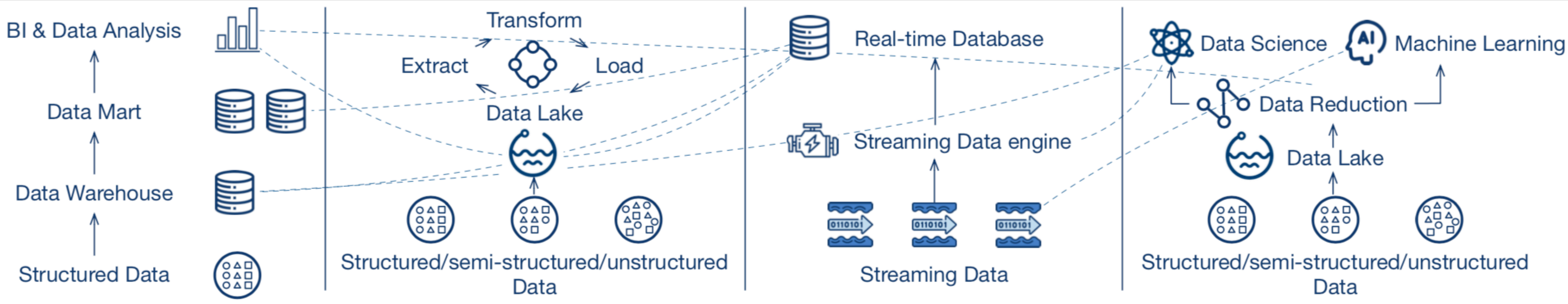
AI4DB Problems

- **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**



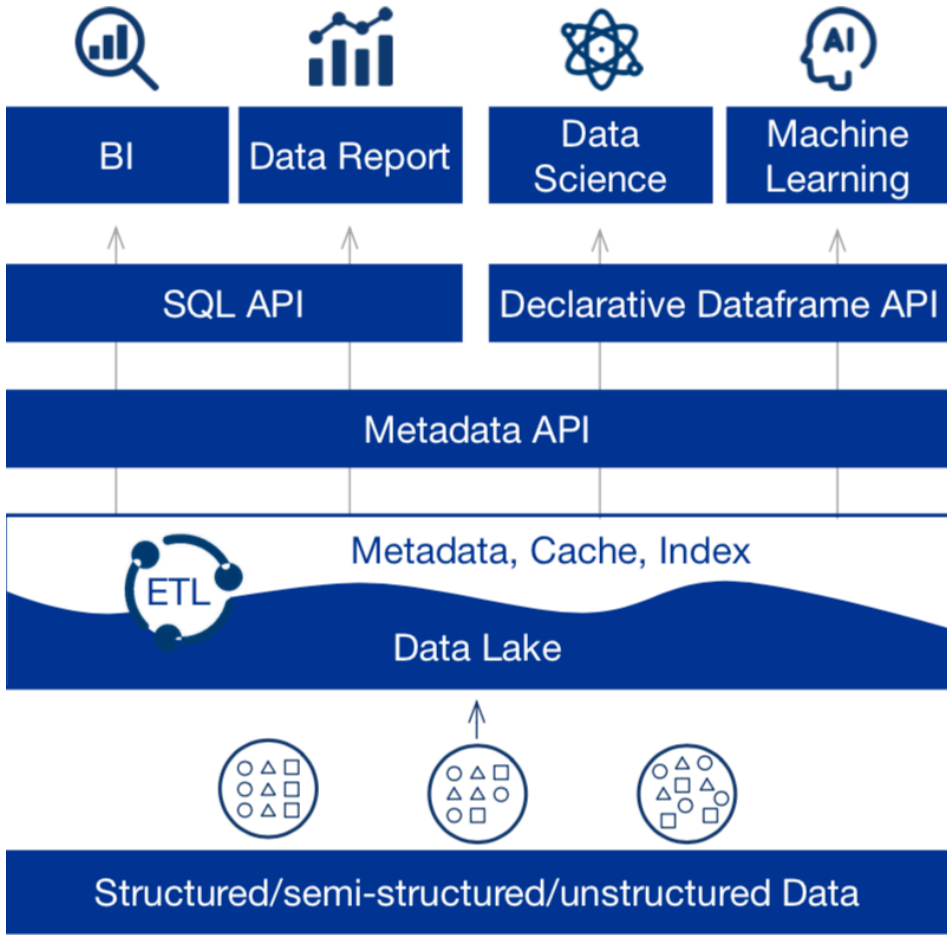
DB4AI Motivation

- Online Inference: T+1 → T+0
- Data Security: ETL → In-DB
- Resource Utilization: data duplicates → one data
- Optimization: DB optimization on learning models
- DB usability: easy to use



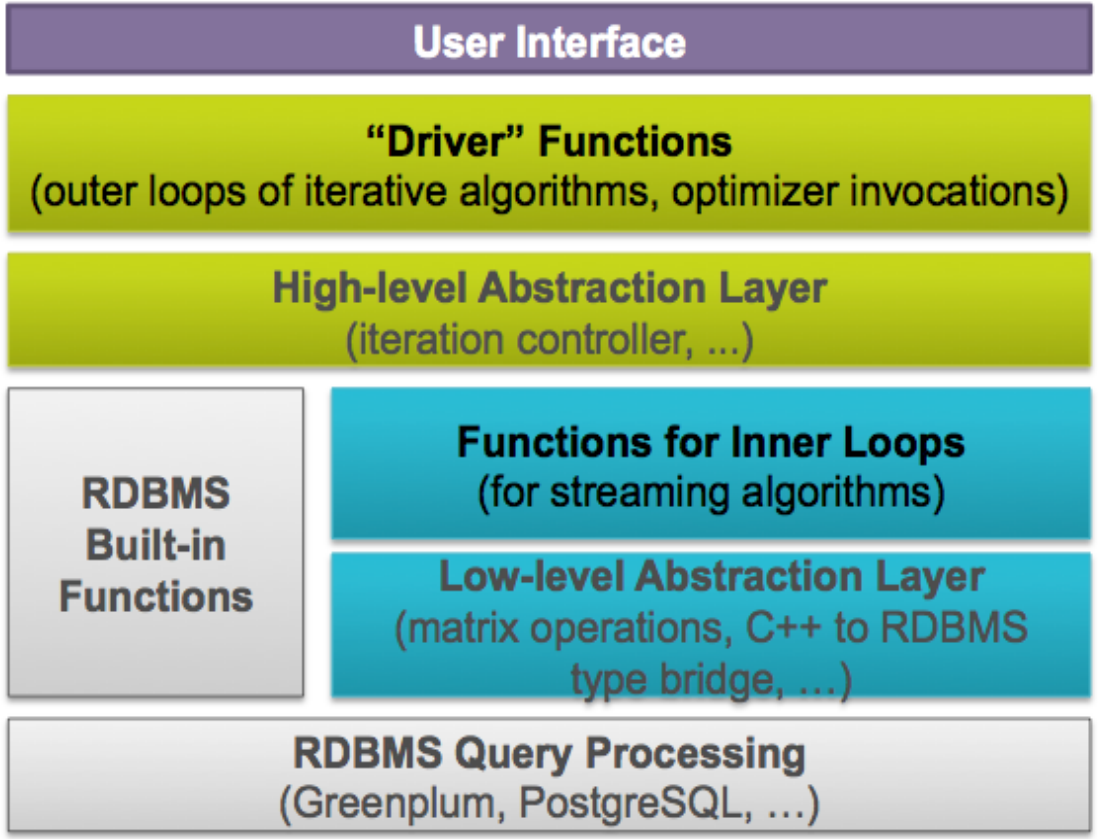
DB4AI Motivation

- One Data
  - Unstructured
  - Structured
  - Semi-structured
- One Analytics
  - SQL
  - Machine Learning
  - Data Science
  - Business Intelligence (BI)



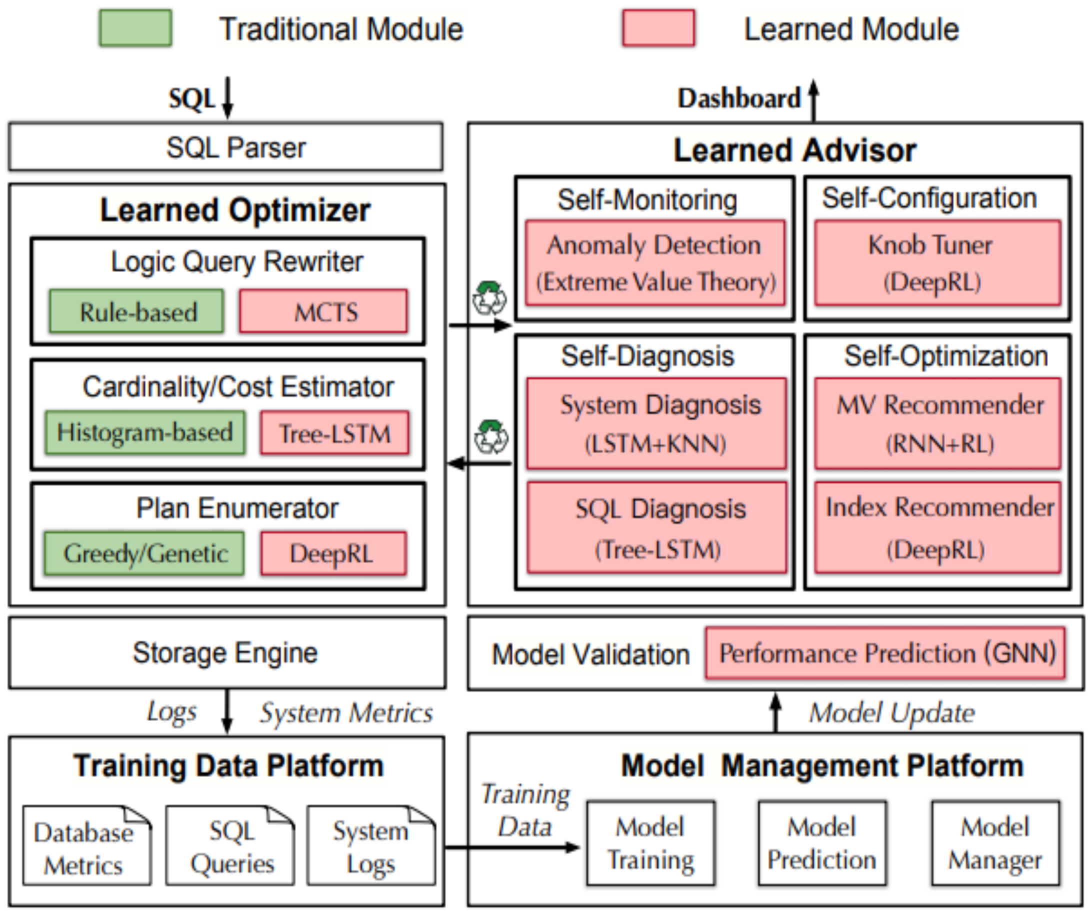
DB4AI

- ☐Declarative AI
  - – AI to SQL
  - – SQL completeness
  - – SQL advisor
- ☐AI optimizations
  - – Cost estimation
  - – Auto parameter
  - – Auto model
  - – Parallel computing
- ☐Lightweight In-DB Model
  - – Training
  - – Inference



Autonomous DB System Architecture

- **Learned Optimizer**
  - Cost Estimation (Tree-LSTM)
  - Logical Optimization (Tree Search)
  - Physical Optimization (RL)
- **Learned Advisor**
  - Monitoring/Diagnosis (LSTM)
  - Configuration (DRL)
  - Optimization (DRL)
- **Model Validator (GNN)**
- **Training-Data/Model Platform**

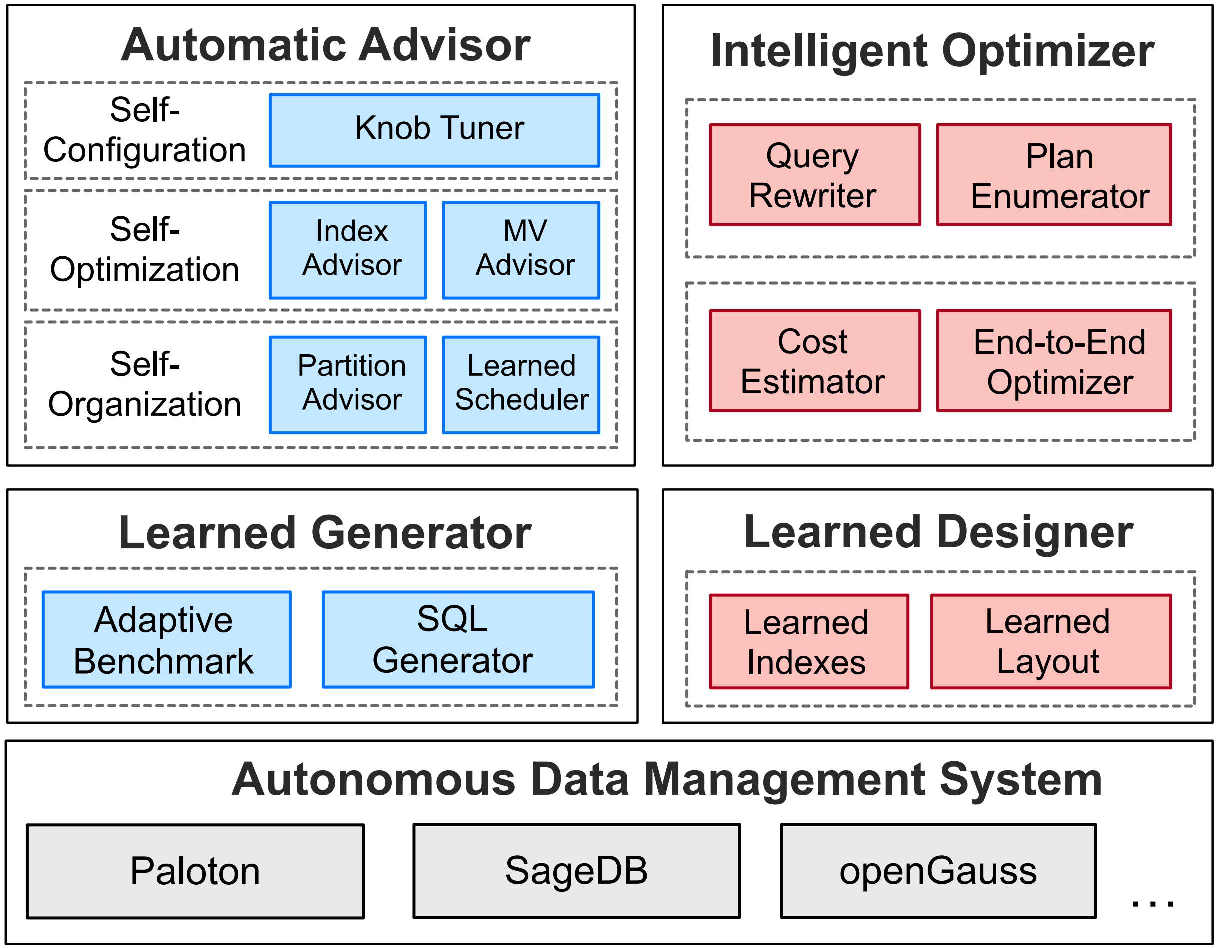


Guoliang Li. openGauss: An Autonomous Database System. VLDB 2021

Category of AI4DB Problems

AI4DB: 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**



Overview of AI4DB

Problem	Description	DB Tasks
Offline NP Optimization	Optimize an NP-hard problem with large search space	**Knob Tuning**
		Index/View Selection
		Partition-key Selection
Online NP Optimization	Optimize an NP-hard problem with large search space (<u>instant feedback</u>)	**Query rewrite**
		Plan Enumeration
Regression	Determine the relationship between one dependent variable and a series of other independent variables	**Cost/Cardinality Estimation**
		Index/View Benefit Estimation
		Latency Estimation
Prediction	Forecast the likelihood of a particular outcome	**Trend Forecast**
		Workload Prediction & Scheduling

Overview of NP-hard Problems

	Method	Strategy	Search Space	Training Data
Offline Optimization (knob tuning, view selection, index selection, partition-key selection)	**Gradient based**	Local search	Small	Huge
	Deep Learning (DL)	Continuous space approximation	Large	Huge
	Meta Learning	Share common model weights	Various spaces	Huge
	Reinforcement Learning (RL)	Multi-step search	Large	--
Online Optimization (query rewrite, plan enumeration)	**MCTS(Monte Carlo Tree Search)+DL**	Multi-step search	Large	Huge
	Multi-armed	Multi-step search	Small	Small

Overview of Regression Problems

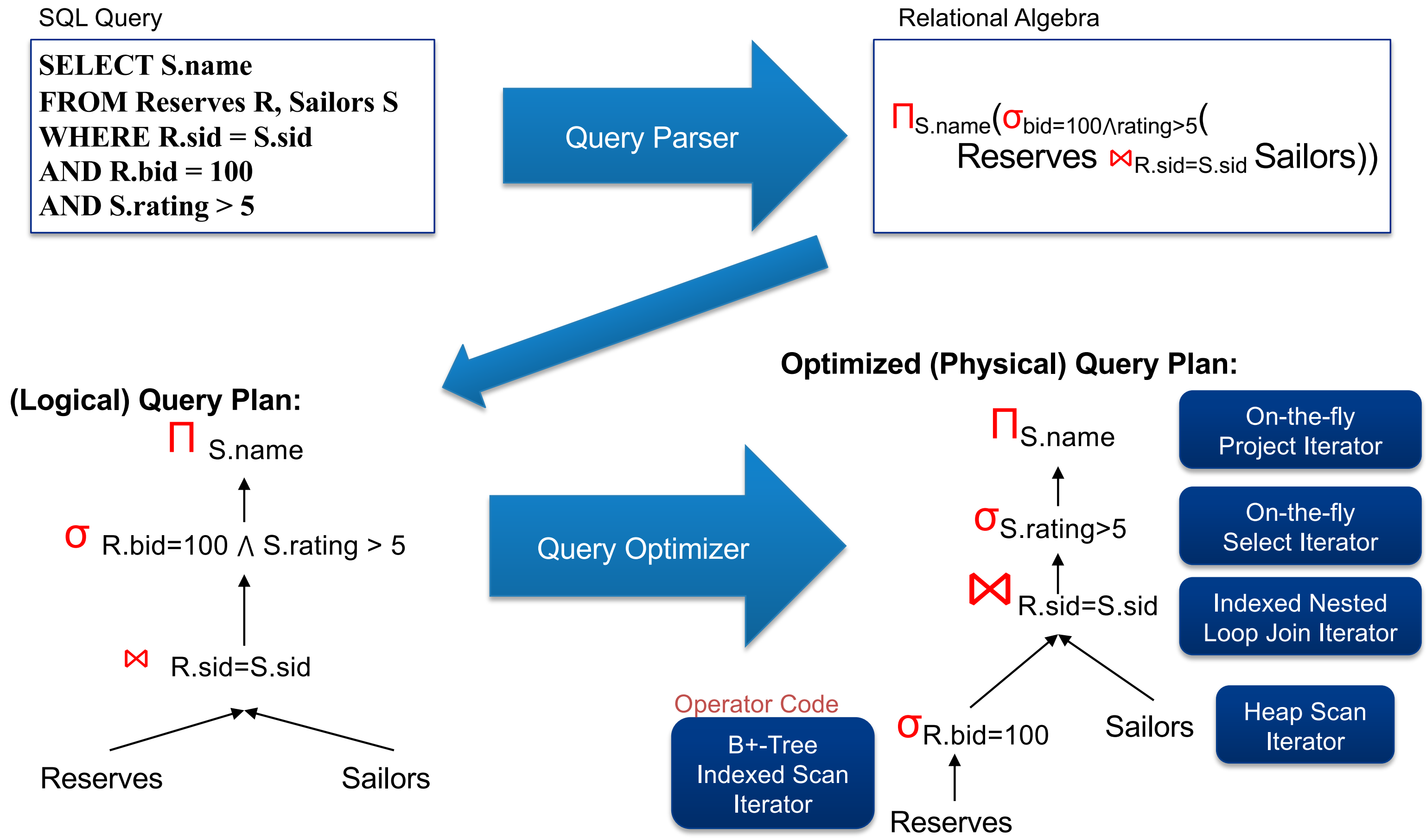
Method	Task	Feature Space	Feature Type	Training Data
Classic ML (e.g., tree-ensemble, gaussian, autoregressive)	cost estimation, view/index benefit estimation	Small	Continuous	Huge
Sum-Product Network	cost estimation	Small	Discrete	Small
Deep Learning	cost estimation, benefit estimation, latency estimation	Large	Continuous	Huge
Graph Embedding	benefit estimation, latency estimation	Large	Continuous	Huge

Overview of Prediction Problems

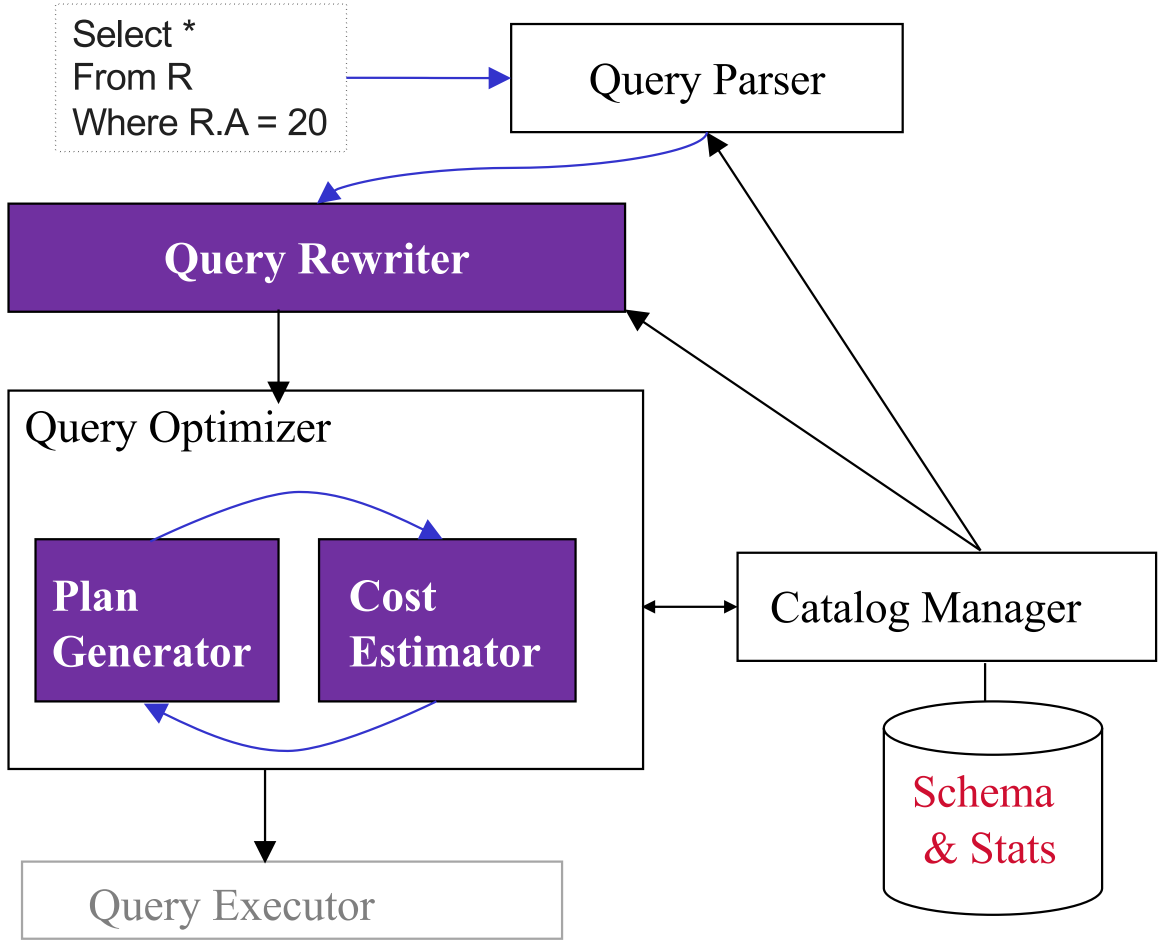
Method	Task	Target	Training Data
Clustering Algorithm	Trend Forecast	High accuracy	Huge
Reinforcement Learning	Workload Scheduling	High performance	--

Learning-based Cardinality/Cost Estimation

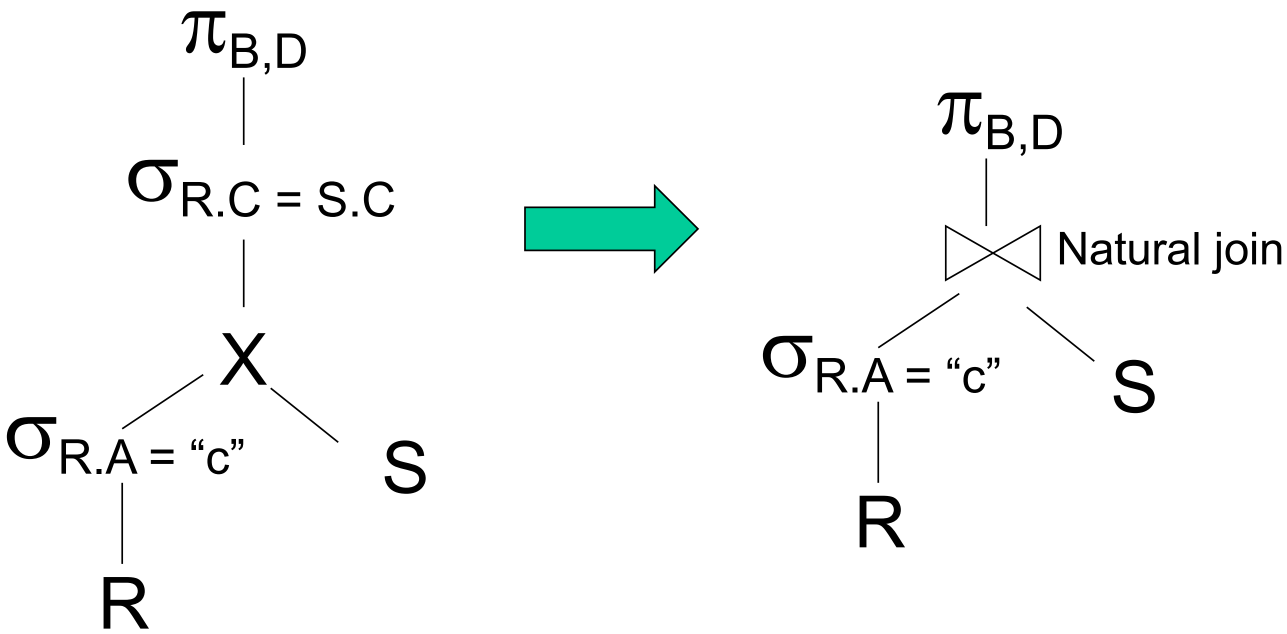
Query Optimizer



Query Optimizer

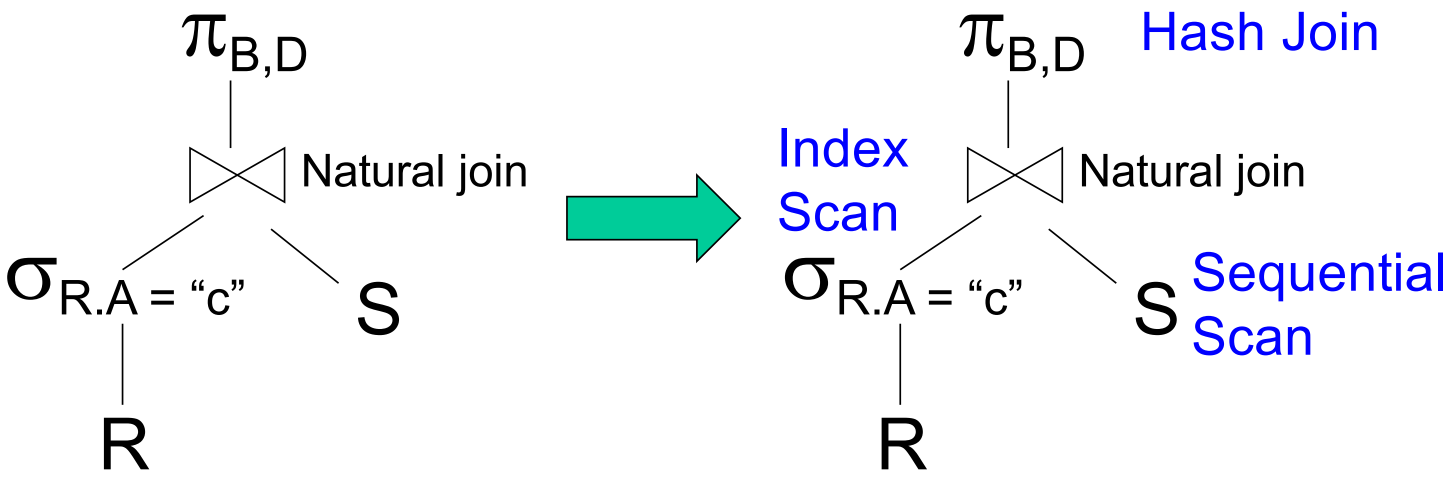


Logical Optimization – Query Rewrite



$$\pi_{B,D}\,[[\sigma_{R.A=\text{“c”}}(R)] \bowtie S]$$

Physical Optimization



$$\pi_{B,D}\, [[\sigma_{R.A=\text{"c"}}(R)] \bowtie S]$$

Cardinality Estimation: Selection

Selectivity Factor (SF) = Cardinality / #tuples

Assumptions:

- Uniformity
  - $\sigma_{A=c}(R) \rightarrow SF = 1/V(R,A)$
  - $\sigma_{A<c}(R) \rightarrow SF = (c - Low(R, A))/(High(R,A) - Low(R,A))$
- Independence
  - Cond1 and Cond2 $\rightarrow$ SF = SF(Cond1) * SF(Cond2)
  - Cond1 or Cond2 $\rightarrow$ SF = SF(Cond1)+SF(Cond2) -SF(Cond1)* SF(Cond2)
  - Not Cond1 $\rightarrow$ SF = 1- SF(Cond1)
- Containment of values
  - $R \bowtie_{A=B} S \rightarrow SF = 1 / \max(V(R,A),V(S,B))$
- Preservation of values
  - $V(R \bowtie_{A=B} S, C) = V(R, C)$

Cardinality Estimation: Selection

Q = SELECT list
FROM R1, ..., Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

- Estimate the number of results of Q: T(Q)
- Obtain number of tuples in each table: T(R1), T(R2), …, T(Rn)
- Also need the selectivity of each condition
  - Selectivity factor (SF) of selection and joins
  - SF(R1.A=v)=T(R1)/V(R1,A)
  - SF(R1.A=R2.B)=T(R1) T(R2)/ max(V(R1,A), V(R2,B))
  - e.g., selectivity(A=3) = 0.01
  - e.g., selective (R1.A=R2.B) = 0.001

T(Q) = T(R1) × ... × T(Rn) × SF(cond1) × ... × SF(condk)
Remark: T(Q) ≤ T(R1) × T(R2) × ... × T(Rn)

Cardinality Estimation: Predicate

- The **selectivity** (**sel**) of a predicate **P** is the fraction/probability of tuples that qualify.
- Formula depends on type of predicate:
  - Equality(=): sel(P(c=x))=count(c=x)/count(all)
  - Range(>=): sel(P(c>=x))=(max–x+1) / (max–min+1)
  - Negation (!=) : sel(P(c!=x))=1-sel(P(c=x))
  - Conjunction (and)
    - Independent assumption
      - sel(P1 & P2) = sel(P1) * sel(P2)
  - Disjunction (or)
      - sel(P1 or P2) = sel(P1) + sel(P2) - sel(P1) * sel(P2)

Cardinality Estimation: Join

$R \bowtie_{A=B} S$: Selectivity = 1 / max(V(R,A),V(S,A))

Q= SELECT * FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A=40

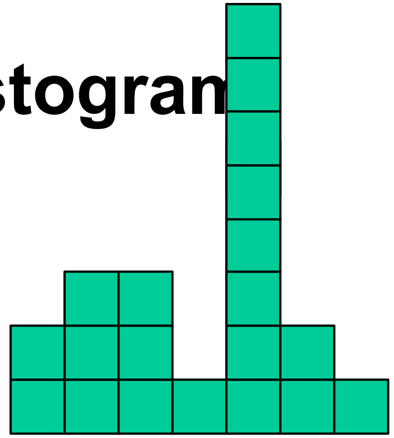
T(R) = 30k, T(S) = 200k, T(T) = 10k
Selectivity of R.B = S.B is 1/3
Selectivity of S.C = T.C is 1/10
Selectivity of R.A = 40 is 1/200

R(A,B)
S(B,C)
T(C,D

$T(Q)=30k*200k*10k*1/3*1/10*1/200 = 10^{10}$

Histograms

- For better estimation, use a histogram

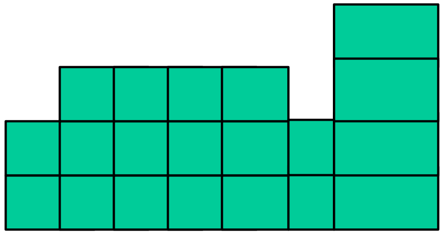


equiwidth

No. of Values	2	3	3	1	8	2	1
Value	0-.99	1-1.99	2-2.99	3-3.99	4-4.99	5-5.99	6-6.99

equidepth

No. of Values	2	3	3	3	3	2	4
Value	0-.99	1-1.99	2-2.99	3-4.05	4.06-4.67	4.68-4.99	5-6.99



Note: 10-bucket equidepth histogram divides the data into *deciles*
- akin to quantiles, median, etc.

Common trick: "end-biased" histogram
- very frequent values in their own buckets

Sketch

- How to count the number of values in a column?
  - E.g., Age = 20?
- Sketch: a cost-model can replace histograms with sketches to improve its selectivity estimate accuracy.
- Probabilistic data structures that generate approximate statistics about a data set.
- Most common examples:
  - Count-Min Sketch (1988): Approximate frequency count of elements in a set.
  - HyperLogLog (2007): Approximate the number of distinct elements in a set.

Sketch

- Given a column with a set of values, build a hash function H, and a sketch M with w entries.
  - M[i] is initialized as 0 for $0 \leq i \leq w-1$
  - For each value **v** in the set,
    - M[H[v]%w] ++;
- Given a value x, use M[H[x]%w] to estimate its account.

Overestimate! Because of collisions!

Count-min Sketch

- How about use *d* hash functions to reduce collisions?
- A matrix M with *d* rows and *w* columns, initialized with 0 for each cell value; *d* hash functions
- For each value v, each hash function h_i:
  - $M[i][h_i(v)]=M[i][h_i(v)]+1$; $h_i(v)$ in [0, w)
- Given x, the frequency f(x) can be estimated as
  - $f(x)=\min_{i \text{ in } [0,d-1]} M[i][h_i(v)]$

	0	1	2	3	4	…	w-1
$h_0(v)$	1	0	0	2	0	0	1
$h_1(v)$	1	0	0	1	0	0	0
…	0	0	0	0	2	0	0
$H_{d-1}(v)$	3	0	0	1	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2,

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	0	0	0	0
$h_1(v)$	0	0	0	0	1	0	0
$h_2(v)$	0	0	0	0	0	1	0
$h_3(v)$	0	0	0	0	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;
- Add 2, 3

	0	1	2	3	4	5	6
$h_0(v)$	0	0	1	1	0	0	0
$h_1(v)$	0	0	1	0	1	0	0
$h_2(v)$	1	0	0	0	0	1	0
$h_3(v)$	1	0	0	0	0	0	1

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3, <u>2</u>

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	0	0	0
$h_1(v)$	0	0	1	0	2	0	0
$h_2(v)$	1	0	0	0	0	2	0
$h_3(v)$	1	0	0	0	0	0	2

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 3, 2,5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3, 2, <u>4</u>

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	0	0
$h_1(v)$	0	0	1+1	0	2	0	0
$h_2(v)$	1	0	1	0	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2

Count-min Sketch

- *d=4* hash functions, w=7 columns
- Given values {2, 3, 2, 4, 5}
  - $h_0(v)=v \% w$; $h_1(v)=v^2 \% w$; $h_2(v)=(2v+1) \% w$; $h_2(v)=(3v^2+1) \% w$;

- Add 2, 3, 2, 4, 5

	0	1	2	3	4	5	6
$h_0(v)$	0	0	2	1	1	1	0
$h_1(v)$	0	0	1+1	0	2+1	0	0
$h_2(v)$	1	0	1	1	0	2	0
$h_3(v)$	1+1	0	0	0	0	0	2+1

count(2)=2; count(3)=1; count(4)=1; count(5)=1

Loglog

Crucial insight: suppose we have a perfect hash function h taking an integer from [1,r] and reporting an integer [0, n)

h(x) in [0, n) for x in [1,r]

The probability that the hash contains:

0 leading zeroes: 1/2

1 leading zero: 1/4

2 leading zeroes: 1/8

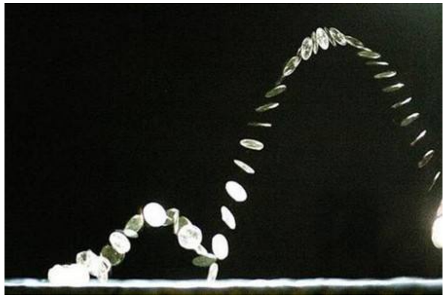
3 leading zeroes: 1/16

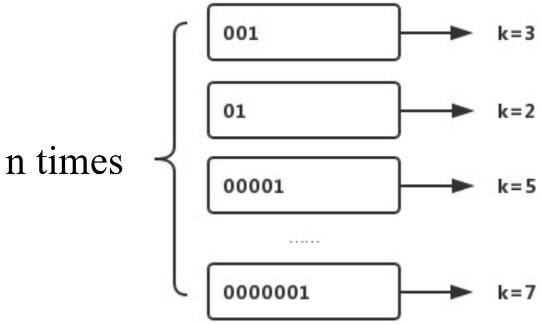
…

w<log n leading zeroes $1/2^w$

LogLog

- Estimate the number of distinct values in a column
- Linear Counting: inefficient
- Hash: large space
- Log Counting: $n=2^{\max(\text{leading 0s})}$
- dv=0: initial distinct number
- For each element v in the column
  - Hash(v) to 0/1 values
  - C0=count leading 0s
  - $dv=\max(dv, 2^{C0})$





Loglog

- Crucial insight: maintaining the largest number of leading zeroes across all hashes allows us to get a (very) rough estimate of the number of distinct elements.
- take multiple independent hashes for each element, average them out?

Estimate: $2^3 = \mathbf{8}$
Actual: 10

7 → 11101100
6 → 11010101
15 → 10110100
8 → 11100110
15 → 10110100
2 → 00100010
2 → 00100010
9 → 01100100
11 → 00011000
8 → 11100110
14 → 10110111
16 → 01001101
12 → 00110110
6 → 11010101
2 → 00100010

Loglog

$\sum 2^w$

Estimate: $2^4 + 2^2 + 2^0 + 2^1 = \mathbf{23}$

Actual: 10

00 stream

2 → 00100010
2 → 00100010
11 → 00000010
12 → 00110110
2 → 00100010

4

01 stream

9 → 01100100
16 → 01001101

2

10 stream

15 → 10110100
15 → 10110100
14 → 10110111

0

11 stream

7 → 11101100
6 → 11010101
8 → 11100110
8 → 11100110
6 → 11010101

1

Loglog

$$m \times 2^{(\sum w)/m}$$

Estimate: $4 \cdot 2^{(4+2+0+1)/4} \approx$ **13.45**

Actual: 10

00 stream

2 → 00100010
2 → 00100010
11 → 00000010
12 → 00110110
2 → 00100010

4

01 stream

9 → 01100100
16 → 01001101

2

10 stream

15 → 10110100
15 → 10110100
14 → 10110111

0

11 stream

7 → 11101100
6 → 11010101
8 → 11100110
8 → 11100110
6 → 11010101

1

HyperLoglog

LogLog uses the arithmetic mean

$$\boldsymbol{\alpha}_m \times m \times 2^{(\sum w)/m}$$

HyperLogLog uses an alternative, the harmonic mean

$$\boldsymbol{\alpha}_m \times m \times m/(\sum 2^{-w})$$

where $\boldsymbol{\alpha}_m$ is a constant.

$$\alpha_m = \begin{cases} 0.673, & m = 16 \\ 0.697, & m = 32 \\ 0.709, & m = 64 \\ \dfrac{0.7213}{1 + \dfrac{1.079}{m}}, & m \geq 128 \end{cases}$$

Loglog

Thus the LogLog algorithm can be succinctly described. We need a hash function h from [1,r] to [0, n)

1. Initialize an array **w** with size $m=2^b$. Let $w_i = 0$ for all i
2. For each element x, compute its hash **h(x)**
   – Split **h(x)** to its first **b** bits and remaining **m-b** bits.
   – Let **c** be the number represented by the first **b** bits and w_0 be the number of leading zeroes in the **m-b** bits.
   – Set $\mathbf{w_c = \max(w_c, w_0)}$
3. Output $\alpha_m \times m \times 2^{(\sum w_c)/m}$

HyperLogLog

- Estimate the number of distinct values in a column
- m buckets, α_m : a constant; M_j: # of leading 0s
- LogLog: $\alpha_m \cdot m \cdot 2^{\sum_{j=1}^{m} M_j / m}$
- HyperLoglog: $\alpha_m \cdot m^2 \cdot \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$

Value: 10,492,800 Hash Value: 1,475,498,572

1	1	1	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0

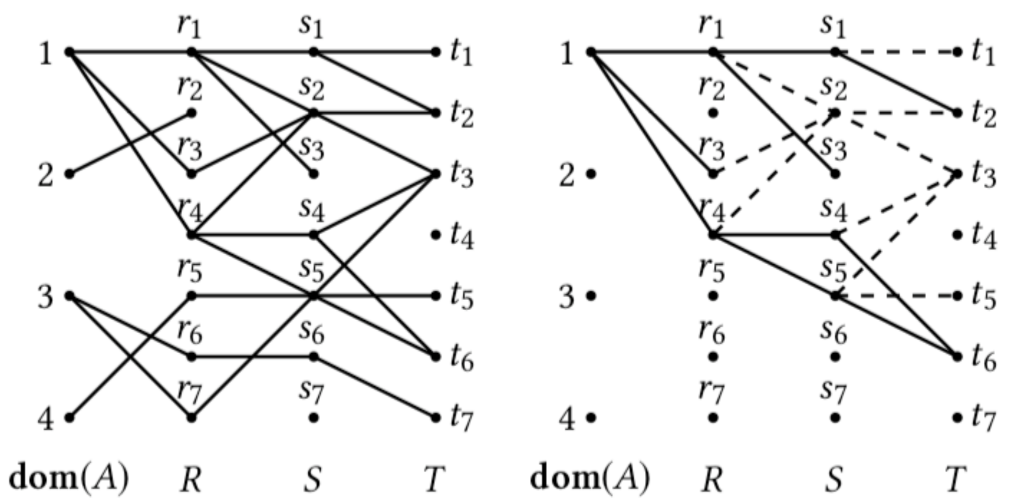
Register Value: 1 Register Index: 12

Register Values: $m = 64$

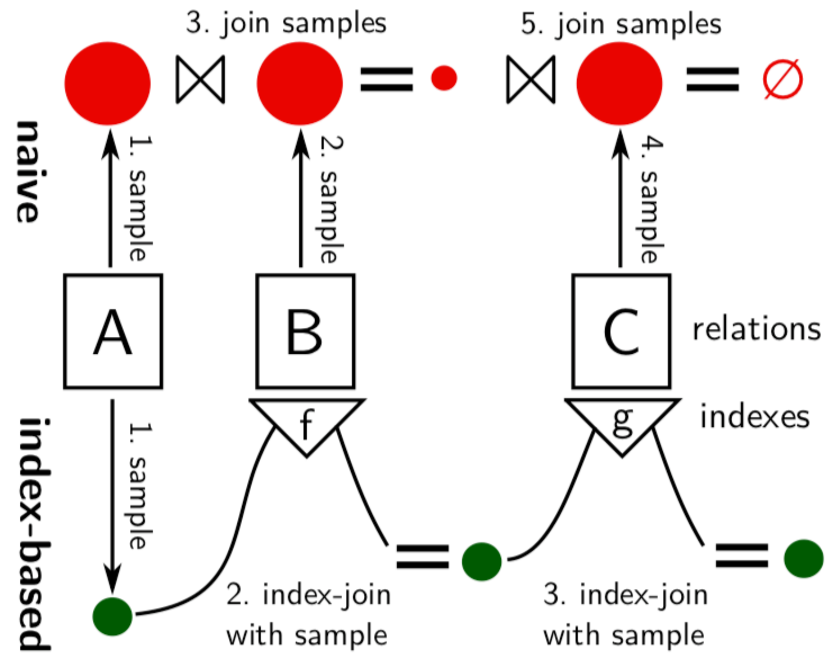
0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
2	0	0	0	0	0	0	2	0	0	3	0	2	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0

Sampling

- Modern DBMSs also collect samples from tables to estimate selectivity.
- Update samples when the underlying tables changes significantly.



(a) Adding conceptual column. (b) Distinct sampling from join.



Traditional Cost Model Summary

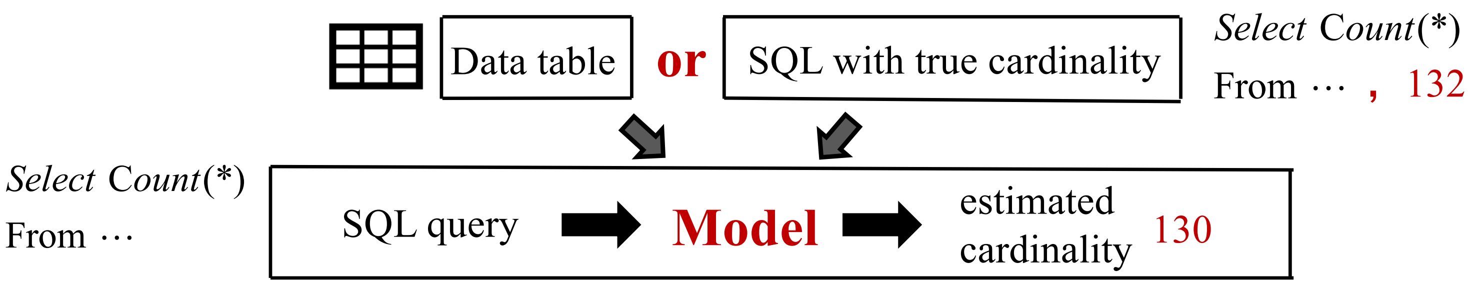
- Sampling-based
- Histogram
- Sketch

Algorithm	**Ref.**	**Year**	**Observables** [14]	**Intuition** [27]	**Core method** (Sec. 3)
FM	[15]	1985	Bit-pattern	Logarithmic hashing	Count trailing 1s
PCSA	[15]	1985	Bit-pattern	Logarithmic hashing	Count trailing 1s
AMS	[3]	1996	Bit-pattern	Logarithmic hashing	Count leading 0s
BJKST	[4]	2002	Order statistics	Bucket-based	Count leading 0s
LogLog	[11]	2003	Bit-pattern	Logarithmic hashing	Count leading 0s
SuperLogLog	[11]	2003	Bit-pattern	Logarithmic hashing	Count leading 0s
HyperLogLog	[14]	2008	Bit-pattern (order statistics)	Logarithmic hashing	Count leading 0s
HyperLogLog++	[24]	2013	Bit-pattern	Logarithmic hashing	Count leading 0s
MinCount	[21]	2005	Order statistics	Interval-based	k-th minimum value
AKMV	[7]	2007	Order statistics	Interval-based	k-th minimum value
LC	[32]	1990	No observable	Bucket-based	Linear synopses
BF	[28]	2010	No observable	Bucket-based	Linear synopses

Cardinality Estimation

☐ Problem Formulation:

- ☐ **Cardinality:** The result size of a query.
- ☐ **Input:** A SQL Query.
- ☐ **Output:** An Estimated Cardinality.



Cardinality Estimation Example

Traditional Cardinality Estimation

- **Sampling**
  - **Core idea:** Estimating selectivity of target query by sampling.
  - **Limitation:** Inference is slow and inaccurate when the amount of data is large.
- **Histogram**
  - **Core idea:** Store the value distribution of each attribute, and calculate the selectivity according to the independence assumption.
  - **Limitation:** Strong independence assumption makes it inaccurate when the data distribution is complex.
- **Sketch:** Estimate the number of distinct elements of a set.

Regression Problems

Database estimation problems can be modeled as regression problems, which fit the high-dimension input variables into target features (e.g., cost, utility) and estimate the value of another variable.

- **Cardinality/Cost Estimation** aims to estimate the cardinality of a query and a regression model (e.g., deep learning model) can be used.

- **Index/View Benefit Estimation** aims to estimate the benefit of creating an index (or a view), and a regression model can be used to estimate the benefit.

- **Query Latency Estimation** aims to estimate the execution time of a query and a regression model can be used to estimate the performance based on query and concurrency features.

Learned Cardinality Estimation

- **Motivation**
  - Due to the attribute value independence assumption as well as the assumption of uniform distribution, traditional cardinality estimation methods tend to fail when the data distribution is complex.
- **Challenge**
  - Correlation between data columns and columns.
  - Multi-table join increases data volume and query types.
- **Optimization Goal**
  - Accuracy, Inference Latency, Model Size, Training Cost

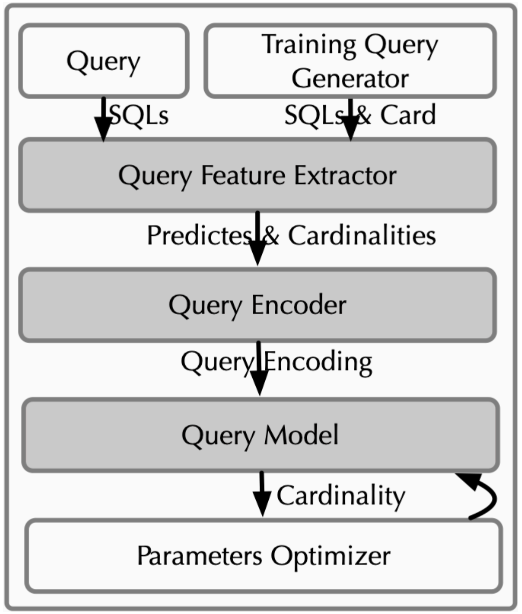
Automatic Cardinality/Cost Estimation

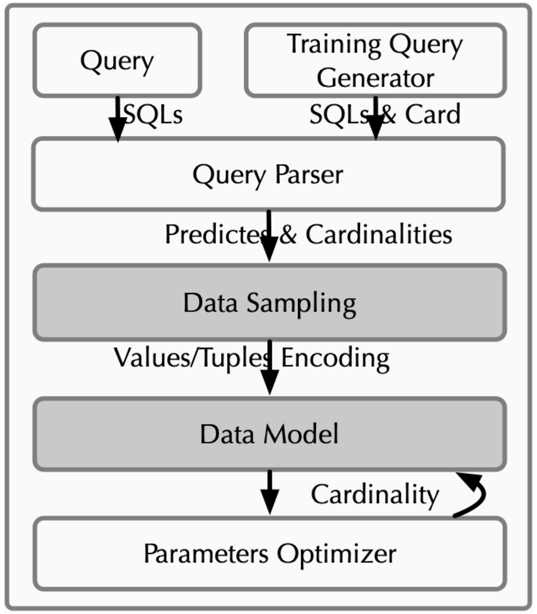
☐ Motivation:

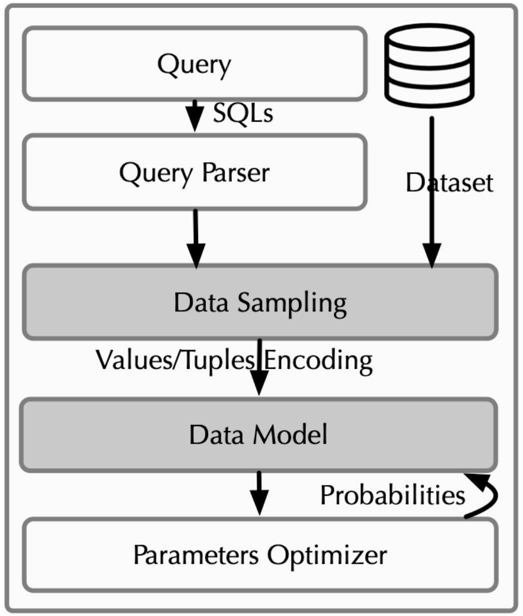
- ☐ **One of the most challenging problems in databases**
  - ➢ Achilles Heel of modern query optimizers
- ☐ **Traditional methods for cardinality estimation**
  - ➢ Sampling (on base tables or joins)
  - ➢ Kernel-based Methods (Gaussian Model on Samples)
  - ➢ Histogram (on single column or multiple columns)
- ☐ **Traditional cost models**
  - ➢ Data sketching/data histogram based methods
  - ➢ Sampling based methods

Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? In VLDB, 2015.

Categories of Learned Cardinality Estimation







(1) Supervised Query Methods

- Neural Network
- XGBoost
- Multi-set Convolutional network

(2) Supervised Data Methods

- Kernel-density model
- Uniform mixture model
- Pre-training summarization model

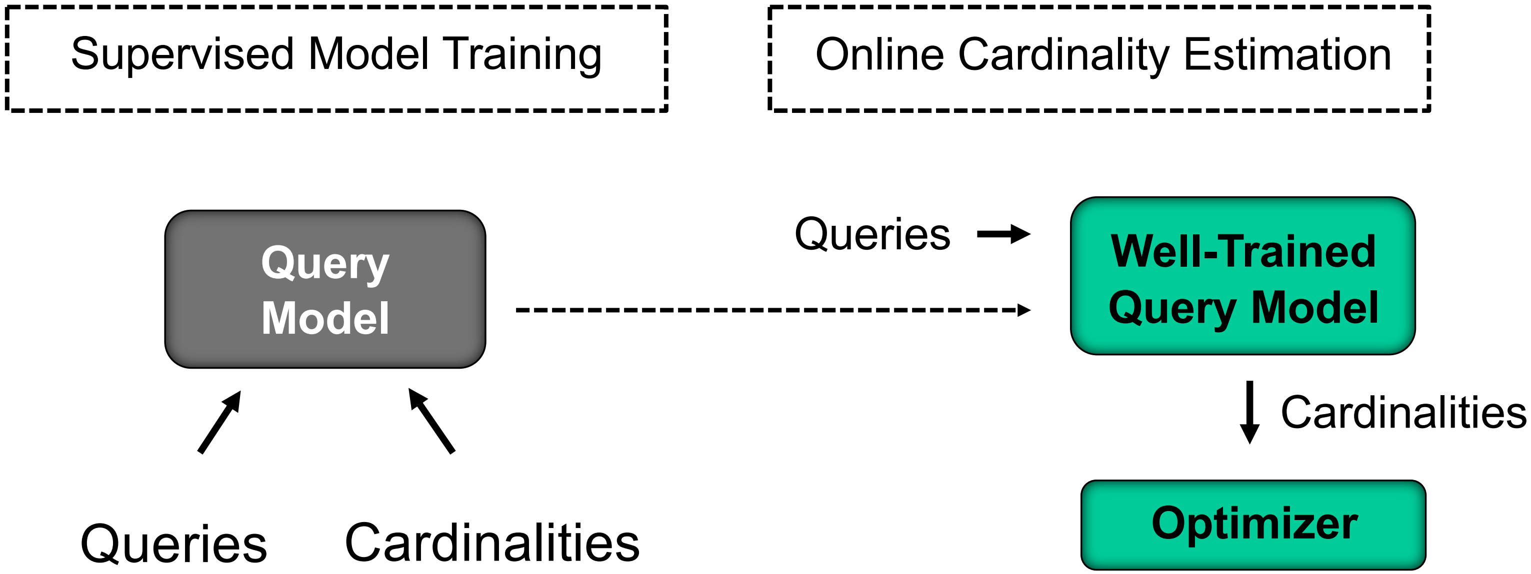
(3) Unsupervised Data Methods

- Sum product network
- Autoregressive (AR) model
- Normalizing Flow (NF) model

1 Supervised Query Methods for Cardinality Estimation

□ Problem Definition

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

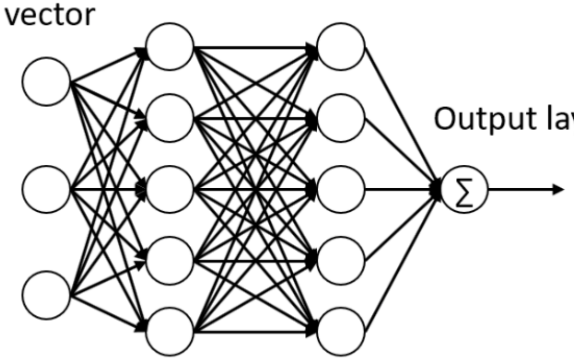


1.1 Query-Driven: Neural Network on Single Tables

- **Motivation:** Traditional estimation methods assume column independence.

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

- **Solution:**
  - **Training**
    - Represent a query $(lb_1 < A_1 < ub_1, \ldots, lb\_d < A_d < ub\_d)$ on a table T with d attributes $A_1, \ldots A\_d$ as $< lb_1, ub_1, \ldots, lb_d, ub_d >$.
    - A neural network with two hidden layers is used to fit the mapping between the representation of the query and its cardinality.
  - **Inference**
    - Answer a query by the trained network.

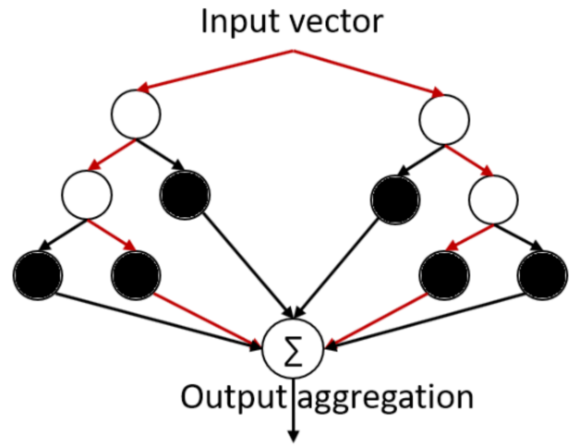


Neural network with 2 hidden layers

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

1.2 Query-Driven: XGBoost for Cardinality Estimation

- **Solution:**
  - The query is represented in the same way as the neural network based approach.
  - Use XGBoost, a decision tree-based ensemble model to fit a mapping between a query's representation and its cardinality.
- **Comparison with Neural Network:**
  - Neural Network-based method are better when training data is sufficient.
  - XGBoost is better when the training data is insufficient.

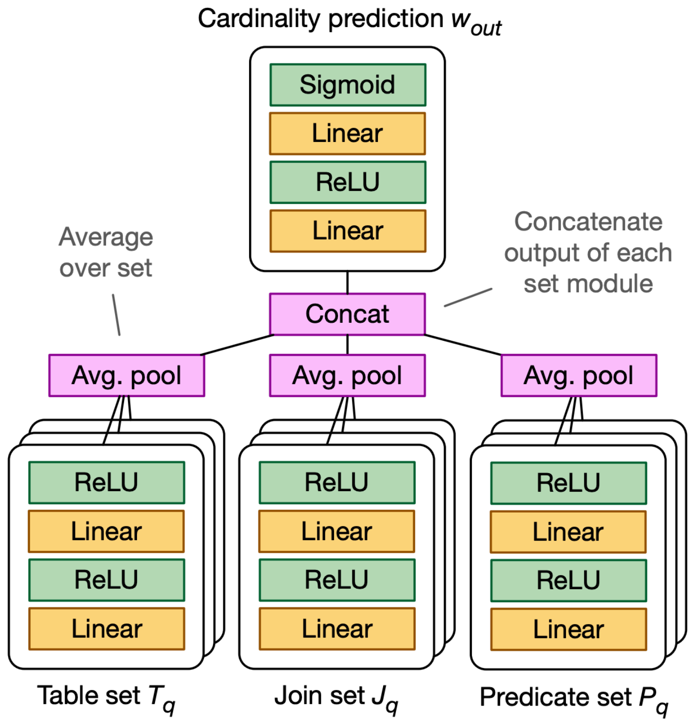


Tree-based ensembles with 2 trees

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

1.3 Query-Driven: Deep Learning for Multi-tables

- **Motivation:** It's difficult for traditional methods to capture join-crossing correlations.
- **Solution:**
  - For table set and join set in the input, encode each table, join with one-hot encoding.
  - For predicates of the form (col, op, val), encode columns col and operators op with one-hot encoding, and represent val as a normalized value in [0, 1].
  - Use some samples to address 0-tuple problem

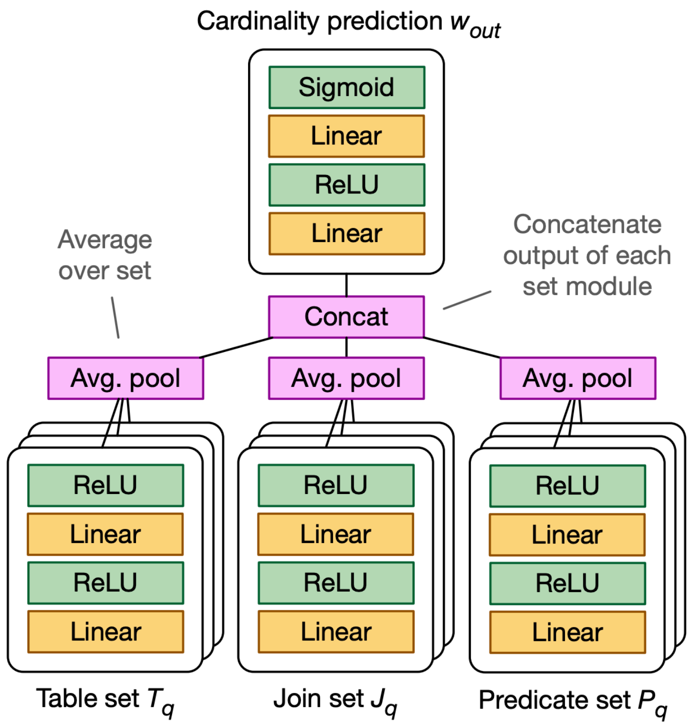


A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In CIDR, 2019.

1.3 Query-Driven: Deep Learning for Multi-tables

◻ Training:

- The three parts of the input are spliced together after going through the linear layer, activation layer, and the dimensionality reduction layer.
- Then go through the linear and activation layer again to get the estimated cardinality.
- Tuning model parameters by backpropagating gradients.



◻ Inference

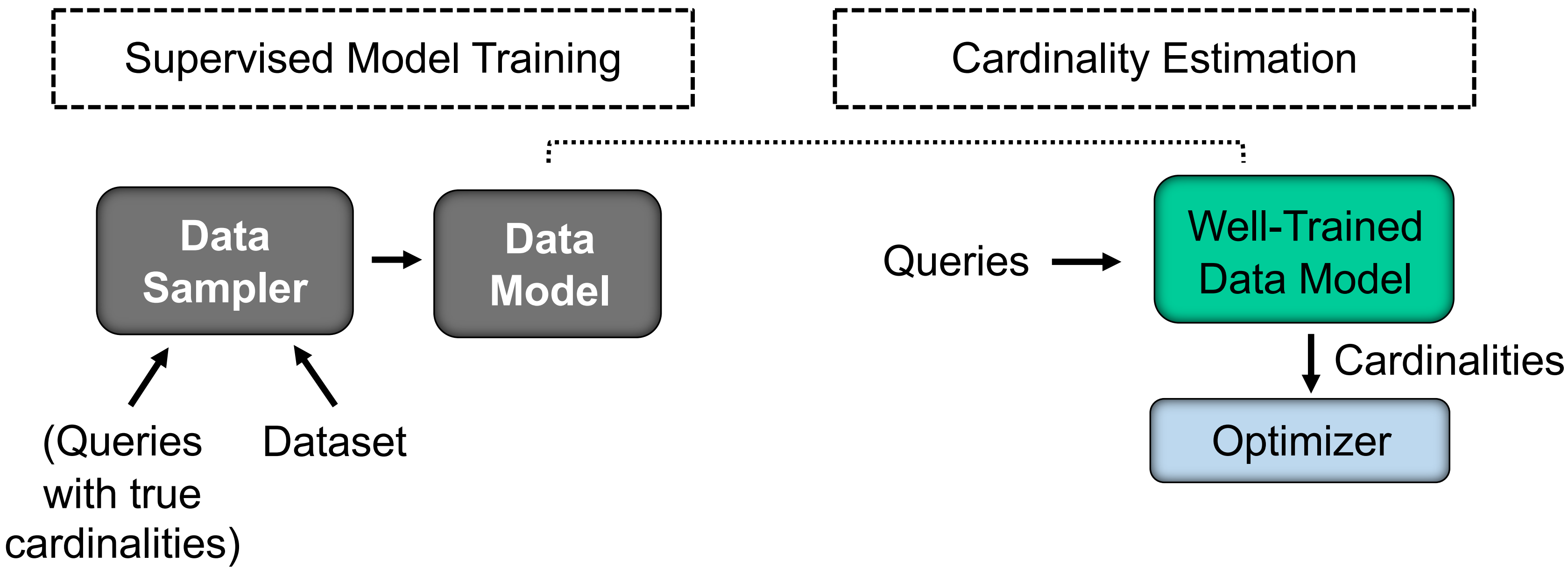
- Answer a query by the trained network.

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.

2 Supervised Data Methods for Cardinality Estimation

□ Problem Definition

A density estimation problem: learn a joint data distribution of each data point. (except for the pre-training summarization model)



2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

- **Motivation:** Multi-dimensional histograms are complex to construct and hard to maintain.
- **Key-idea:** Fit the probability density distribution of a data table by kernel density model.

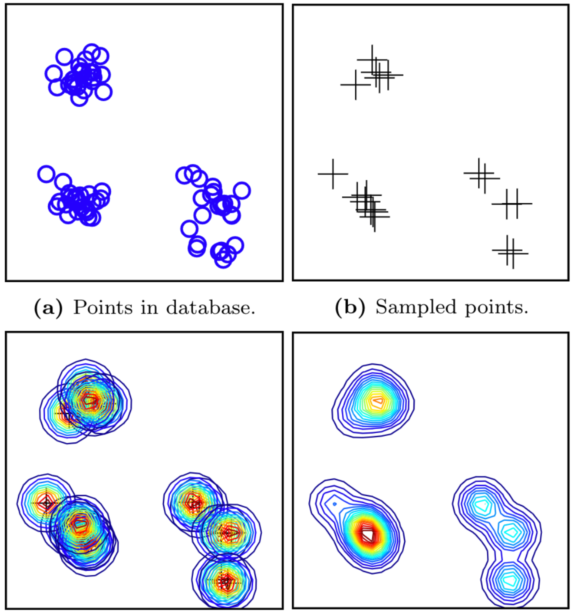
➢ Kernel-Density Model

Model: $$\hat{p}_H(\vec{x}) = \frac{1}{s \cdot |H|} \sum_{i=1}^{s} K\left(H^{-1}\left[\vec{t}^{(i)} - \vec{x}\right]\right)$$

- s is sample size; K is Gaussian function;
- $\vec{t}$ is sampled point; H is a parameter that needs to be learned.,

➢ Inference: $\hat{p}_H(\Omega) = \int_{\Omega} \hat{p}(\vec{x})\, d\vec{x}$

- Ω is the space represented by a query.


(a) Points in database. (b) Sampled points.
(c) Kernels. (d) Estimated distribution.

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

2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

Kernel-Density Estimation

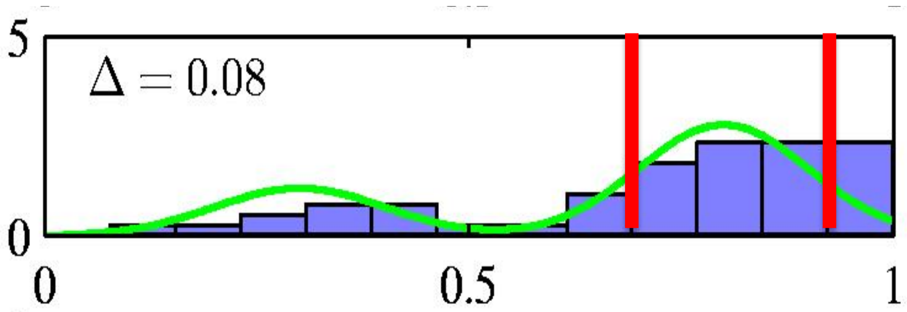
$$p_i = \frac{n_i}{N\,\Delta_i}$$

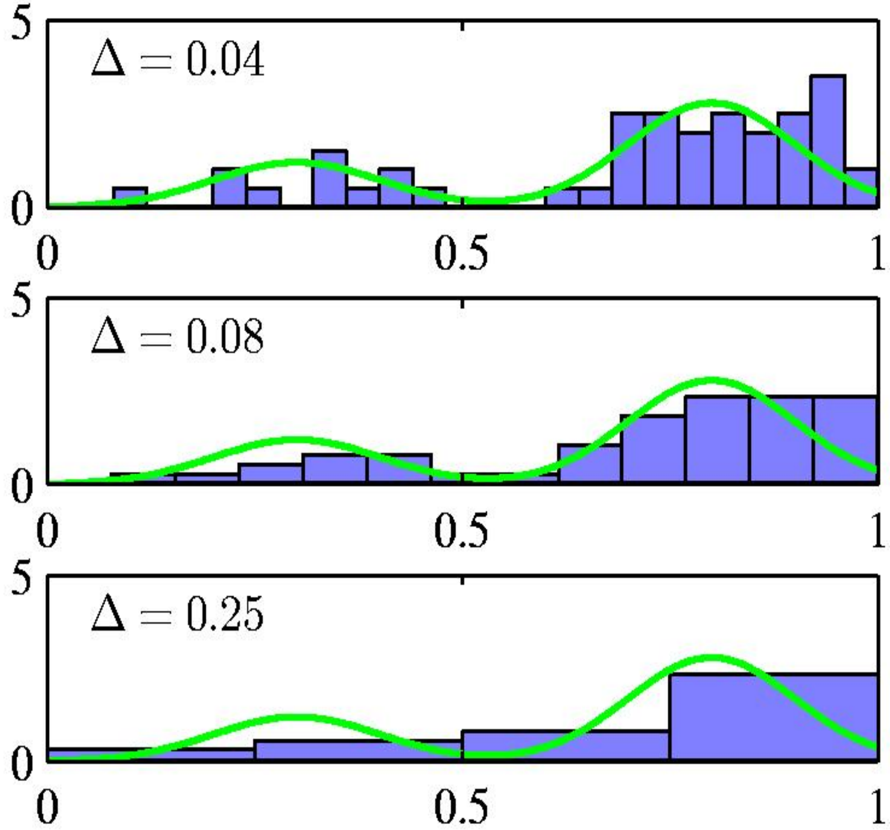
p_i: probability

n_i: number of points

N: total number of points

Δ: bandwidth





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

2.1 Supervised Data-Driven: Kernel-Density Model on Single Table

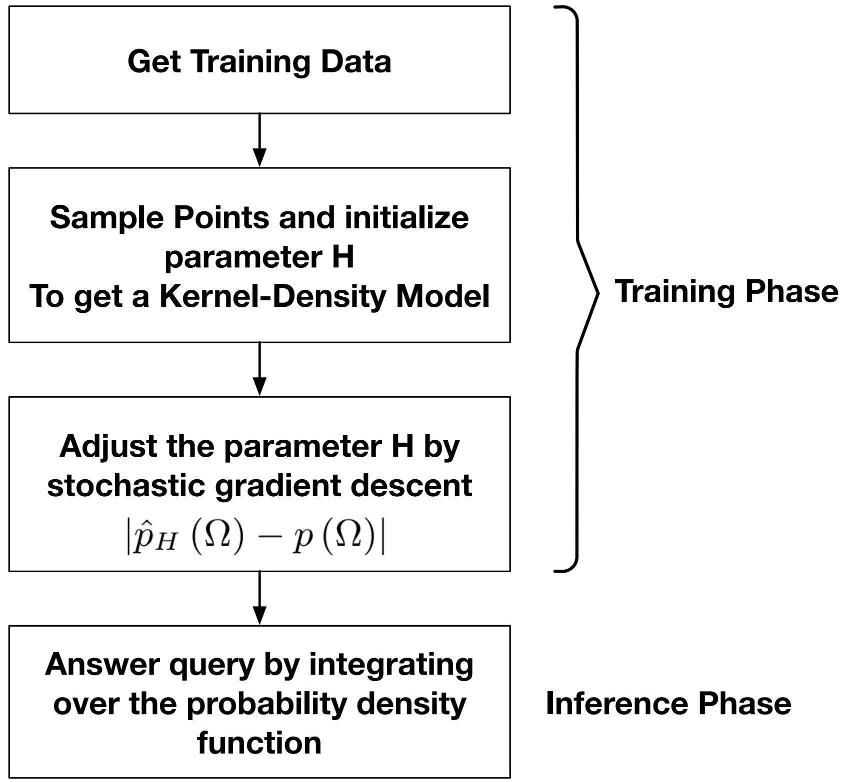
☐ Solution

☐ Training

- Get a lot of queries with true cardinalities.
- Sample points (rows) from the table and initialize the parameter H.
- Adjust the parameter H by stochastic gradient descent according to estimated cardinalities by Kernel-Density Model and the true cardinalities.

☐ Inference

- Answer queries by accumulating kernel density 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.

2.2 Supervised Data-Driven: Uniform Mixture Model on Single Table

- **Motivation:** Traditional methods need to be populated in advance by performing costly table scans.
- **Key-idea:** Fit the probability density distribution of a data table by uniform mixture model.
- **Uniform Mixture Model:**

Model: $f(x) = \sum_{z=1}^{m} h(z)\, g_z(x) = \sum_{z=1}^{m} w_z\, g_z(x)$

- w_z is the weight for a subpopulation z
- $g_z(x) = 1/|\mathrm{G_z}|$ is uniform distribution function
- $|G_z|$ is area of subpopulation z

Inference: $\int_{B_i} f(x)\, dx = \int_{B_i} \sum_{z=1}^{m} w_z\, g_z(x)\, dx$

- B_i is the space represented by a query.

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

2.2 Supervised Data-Driven: Uniform Mixture Model on Single Table

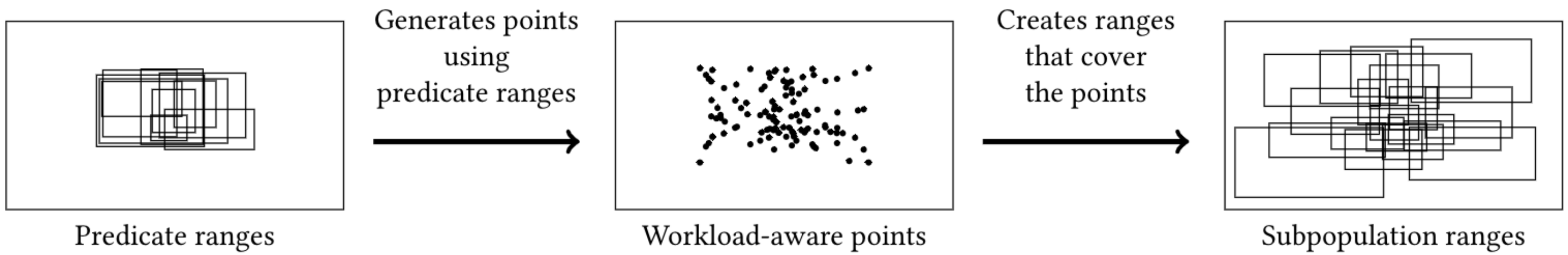
☐ Solution:

☐ Training

- Sample some points within queries with true cardinalities.
- Generate subgroups for the points.
- Learn the weights w_z of the uniformity mixture model.

☐ Inference

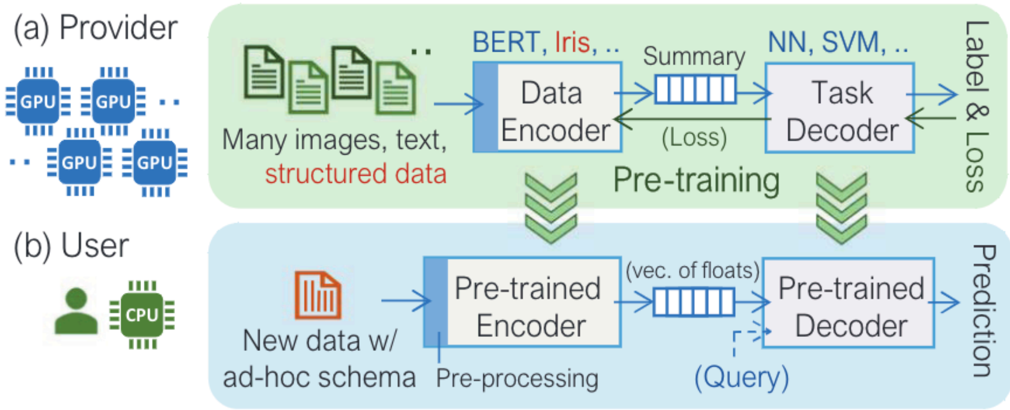
- Answer a query by calculating the cumulative probability density (i.e., selectivity) according to the mixture density function.
  - overlap area between query rectangle and data rectangle



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

2.3 Supervised Data-Driven: Pre-training Summarization Model on Single Table

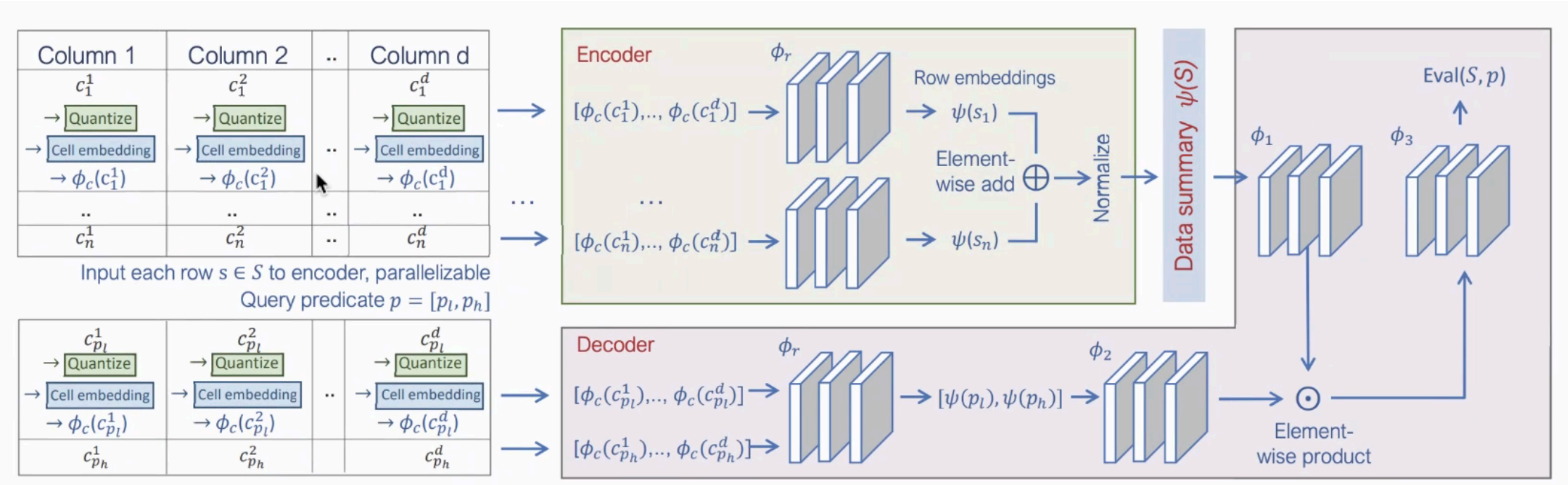
- **Motivation:** Pre-training models avoid per-dataset training.
- **Solution:**
  - Pre-train encoder and decoder with large data tables via gradient descent (Loss function is $\left|log\frac{true\ card}{est.card}\right|$).
  - Encode a table with the pretrained encoder.
  - Input a query and the encoded result of the table into the pre-trained decoder to get the cardinality.



Lu Y, Kandula S, König A C, et al. Pre-training summarization models of structured datasets for cardinality estimation. PVLDB, 15(3): 414-426, 2021

2.3 Supervised Data-Driven: Pre-training Summarization Model on Single Table

- Data Encoder
- Query Decoder

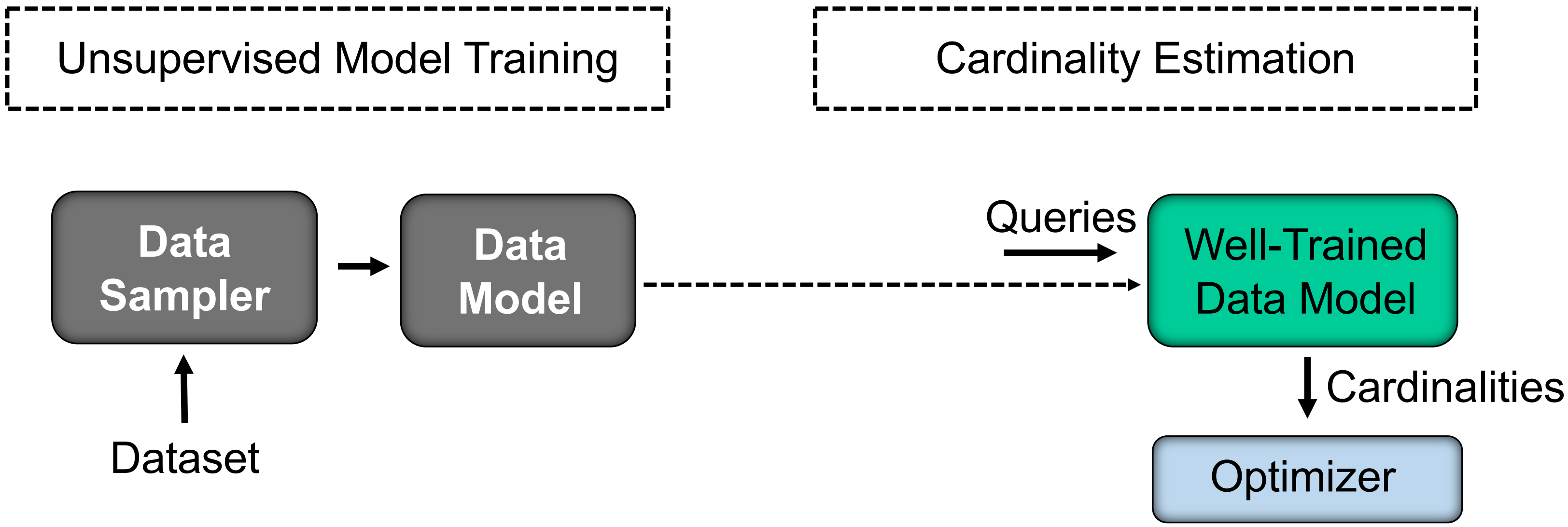


Lu Y, Kandula S, König A C, et al. Pre-training summarization models of structured datasets for cardinality estimation. PVLDB, 15(3): 414-426, 2021

3 Unsupervised Data Methods for Cardinality Estimation

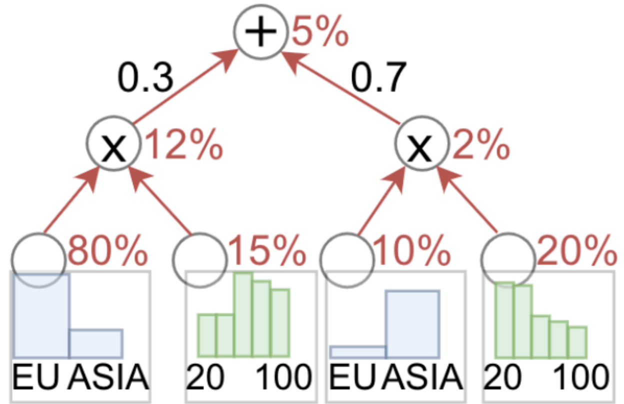
☐ Problem Definition

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



3.1 Unsupervised Data-Driven: Sum-Product Network for Multi-tables

- **Motivation:** Most of the existing estimators require SQL queries.
- **Base-idea:** Learn the joint probability distribution by Sum-Product Network .
- Relational Sum Product Network (RSPN)
  - RSPN consists of three types of node:
    - product node: split the columns of a table.
    - sum node: split the rows of a table.
    - leaf node: represent probability distributions for individual variables.



(d) Probability of European Customers younger than 30

Benjamin Hilprecht, Andreas Schmidt, et al . DeepDB: Learn from Data, not from Queries! PVLDB 13, 13(7): 992-1005, 2020

3.1 Unsupervised Data-Driven: Sum-Product Network for Multi-tables

◻ Solution:

◻ Training

- Generate some queries and their cardinalities.
- Build a RSPN by recursively partitioning
  - Row: K-means clustering
  - Column: randomized dependency coefficient.

◻ Inference

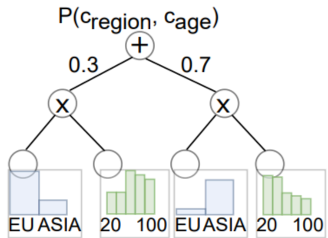
- Estimate cardinality in bottom-up.
- Supports multi-table queries via join sampling, and supports queries on sub-schemas based on fanout scaling.

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

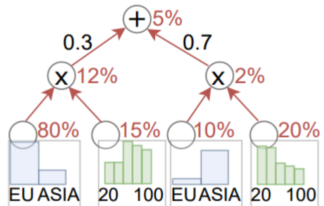
(a) Example Table

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

Fanout scaling for multi-tables

Fanout scaling is to support sub-schema queries on a full outer join table. (full outer join table contains duplicate tuples)

- **Solution:** for each $foreign\ key \rightarrow primary\ key$ relationship, add a column denoting how many corresponding join partners a tuple has.

- Example:

```
SELECT COUNT(*)
  FROM CUSTOMER C
 WHERE c_region='EUROPE';
```

Customer

c_id	c_age	c_region
1	20	EUROPE
2	50	EUROPE
3	80	ASIA

Order

o_id	c_id	o_channel
1	1	ONLINE
2	1	STORE
3	3	ONLINE
4	3	STORE

Customer⟗Order

$\mathcal{N}_C$	c_id	c_age	c_region	$\mathcal{F}'_{C\leftarrow O}$	$\mathcal{N}_O$	o_id	o_channel
1	1	20	EUROPE	2	1	1	ONLINE
1	1	20	EUROPE	2	1	2	STORE
1	2	50	EUROPE	1	0	NULL	NULL
1	3	80	ASIA	2	1	3	ONLINE
1	3	80	ASIA	2	1	4	STORE

- True answer is 2, but there are 3 tuples in the outer join table.

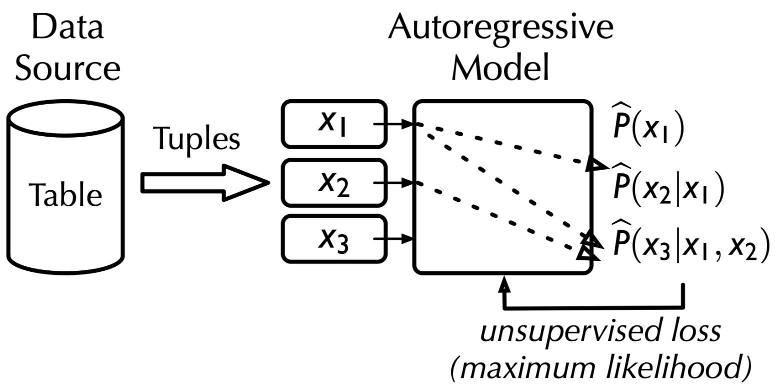
- Fanout scaling result: $|\mathtt{C} ⟗ \mathtt{O}| \cdot \mathbb{E}(1/\mathcal{F}'_{C\leftarrow O} \cdot \mathbf{1}_{\mathtt{c\_region='EU'}} \cdot \mathcal{N}_C)$

$$= 5 \cdot \frac{1/2+1/2+1}{5} = 2$$

3.2 Unsupervised Data-Driven: Autoregressive Model on Single Table

- **Motivation:** Existing estimators struggle to capture the rich multivariate distributions of relational tables.
- **Solution:**
  - **Training:** Use Autoregressive (AR) Model to fit the joint probability of different columns.
  - **Inference:** Estimate the cardinality of the equivalent query based on the AR model.
    - Monte Carlo sampling
  - Range queries are supported by progressive sampling (sampling by learned probability distribution).

$$\begin{aligned}\widehat{P}(\boldsymbol{x}) &= \widehat{P}(x_1, x_2, \cdots, x_n) \\ &= \widehat{P}(x_1)\widehat{P}(x_2|x_1)\cdots\widehat{P}(x_n|x_1, \ldots, x_{n-1})\end{aligned}$$

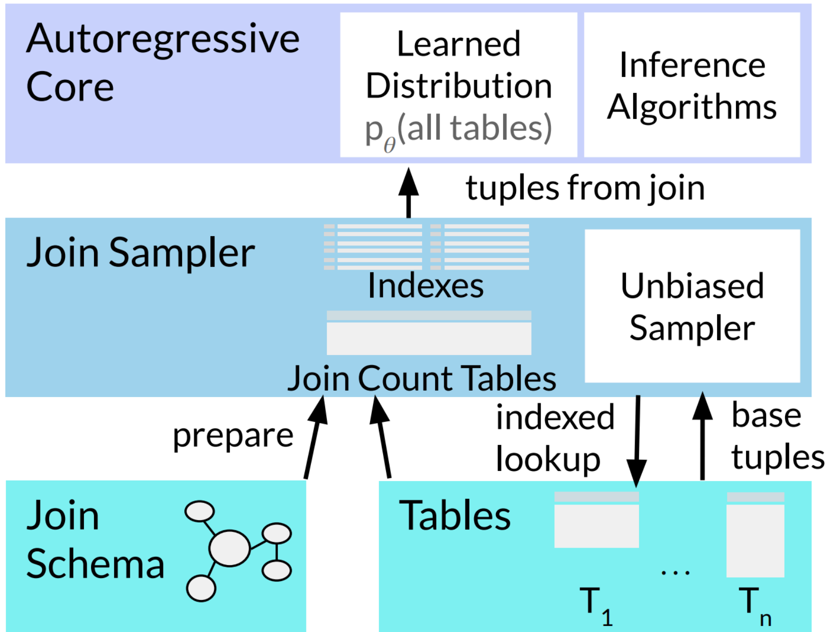


Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep Unsupervised Cardinality Estimation. PVLDB, 13(3): 279-292, 2019.

3.2 Unsupervised Data-Driven: Autoregressive Model on Single Table

- **Motivation:** Previous AR models do not support multi-table queries.

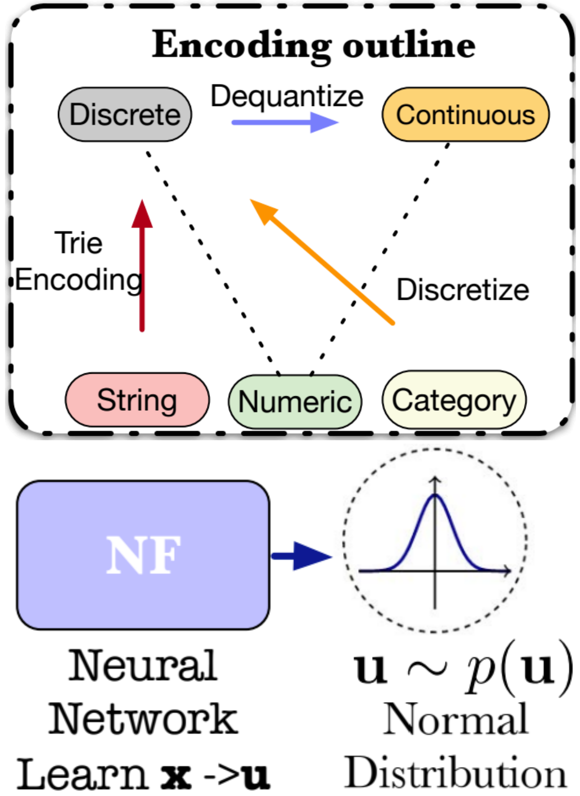
- **Solution:**
  - Learn an autoregressive model for the outer join of all tables.
  - Supports multi-table queries via join sampling, and supports queries on sub-schemas through fanout scaling.



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

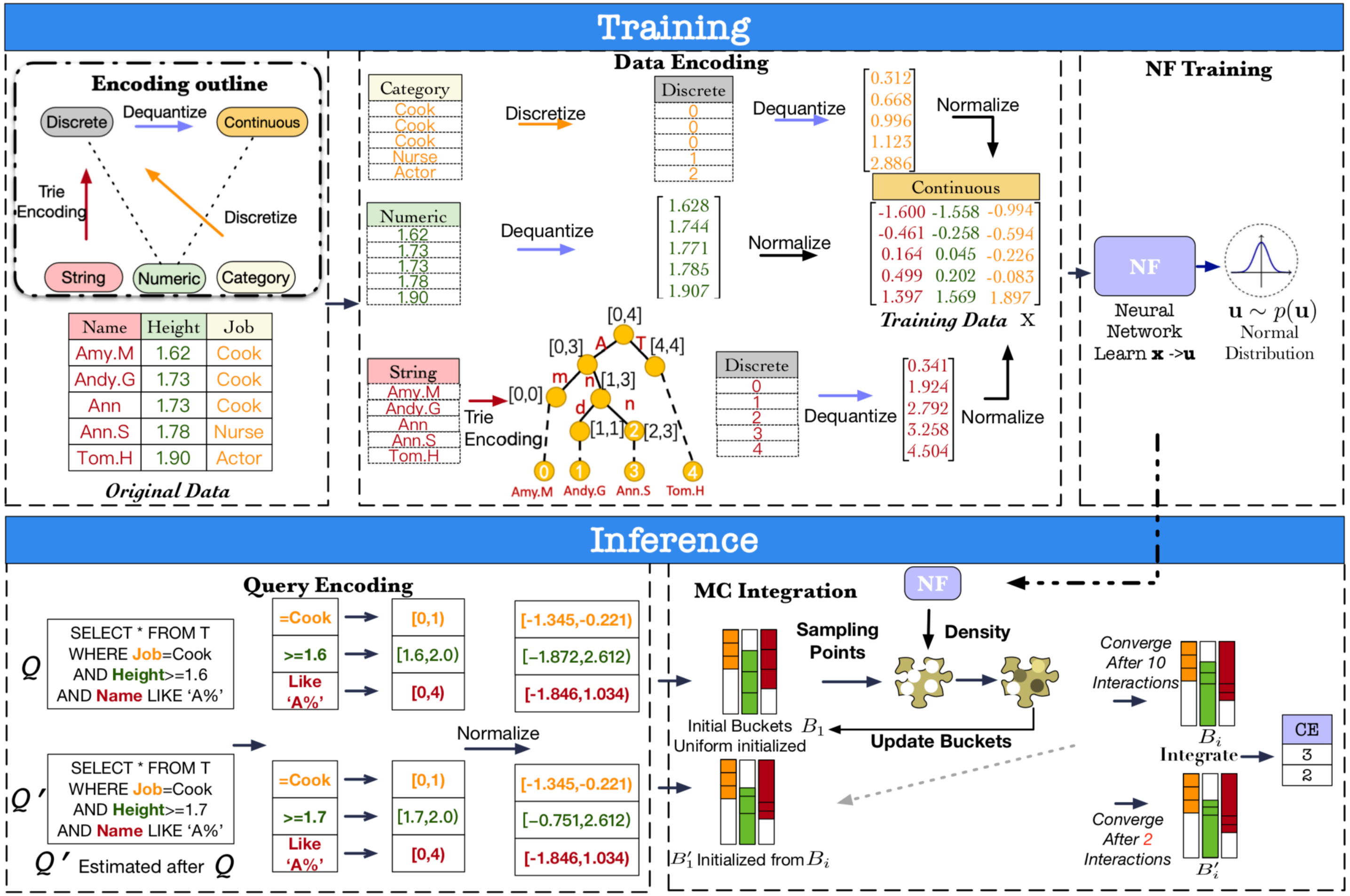
3.3 Unsupervised Data-Driven: Normalizing Flow (NF) model for Multi-tables

- **Motivation:** Previous data-driven approaches do not handle tables with large domain sizes well.
- **Solution:**
  - Dequantize and normalize discrete variables to continuous variables.
  - Use Normalizing Flow (NF) model to learn the joint probability distribution of data points.
  - Accumulating continuous normalized flow distribution function by adaptive importance sampling to answer a query.
  - Support multi-table query through fanout scaling.



Wang J, Chai C, Liu J, et al. FACE: a normalizing flow based cardinality estimator[J]. Proceedings of the VLDB Endowment, 15(1): 72-84, 2021

3.3 Unsupervised Data-Driven: Normalizing Flow (NF) model for Multi-tables



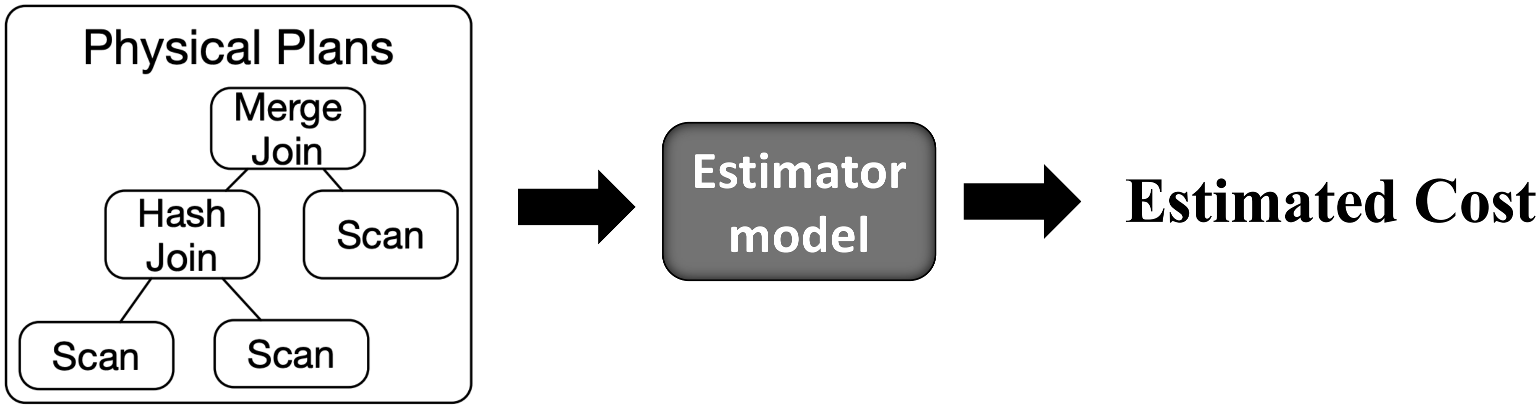
Wang J, Chai C, Liu J, et al. FACE: a normalizing flow based cardinality estimator. Proceedings of the VLDB Endowment, 15(1): 72-84, 2021

Summarization of Learned Cardinality Estimation

Method	Quality	Training overhead	Training Data	Adaptive	Model Size	Inference Latency
Lightweight Neural Network & XGBoost	✓	low	many queries	✓	small	fast
Convolutional Neural Network	✓✓	low	many queries	✓✓	small	fast
Kernel-Density Model	✓	medium	Data samples	✓	small	medium
Uniform Mixture Model	✓	medium	Data samples	✓	small	medium
Autoregressive (AR) Model	✓✓	high	Data samples	✓✓✓	high	slow
Sum-Product Network	✓✓	high	Data samples	✓✓✓	medium	medium
Normalizing Flow (NF) model	✓✓	high	Data samples	✓✓✓	medium	medium
Pre-training summarization model	✓	high	Lots of tables	✓✓	very small	fast

Cost Estimation

- **Problem Formulation:**
  - **Cost:** Execution cost of a query plan.
  - **Input:** A SQL Query Plan.
  - **Output:** An Estimated Cost.

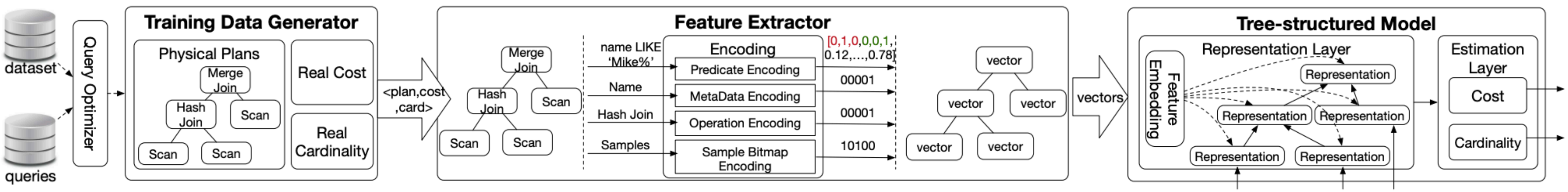


Relations of Cardinality/Cost Estimation

- **Task Target**
  - Cost estimation is to approximate the execution-time/ resource-consumption.
- **Correlations**
  - Cost estimation is based on cardinality.
- **Estimation Difficulity**
  - Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, CPU usage).

Tree-LSTM for Cost Estimation

- **Motivation:** Traditional cost estimation is inaccurate without learned plan representation.
- **Solution:**
  - Generate many query plans and true costs as training data.
  - Encode the query plan via one-hot encoding.
  - Representation layer learns an embedding of each query plan by Tree-LSTM.
  - Estimation layer outputs estimated cost based on the representation layer's output.

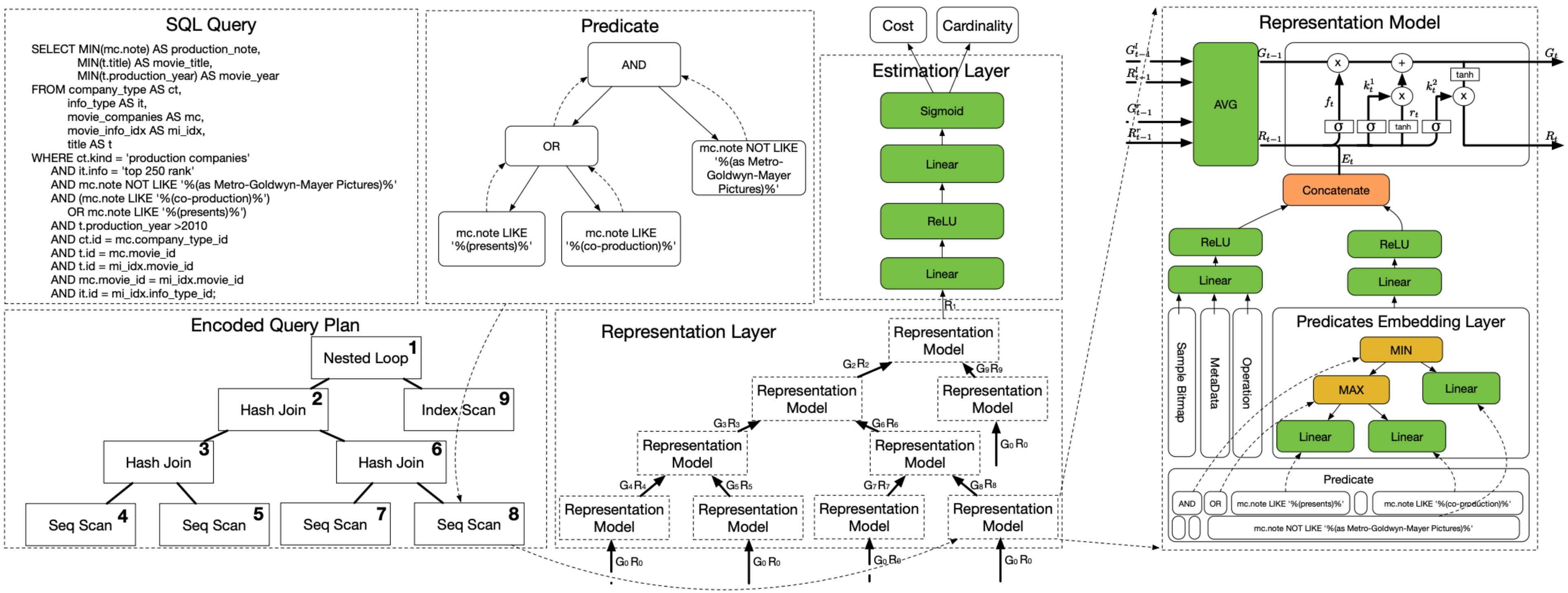


J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

Tree-LSTM for Cost Estimation

◻ Model Construction

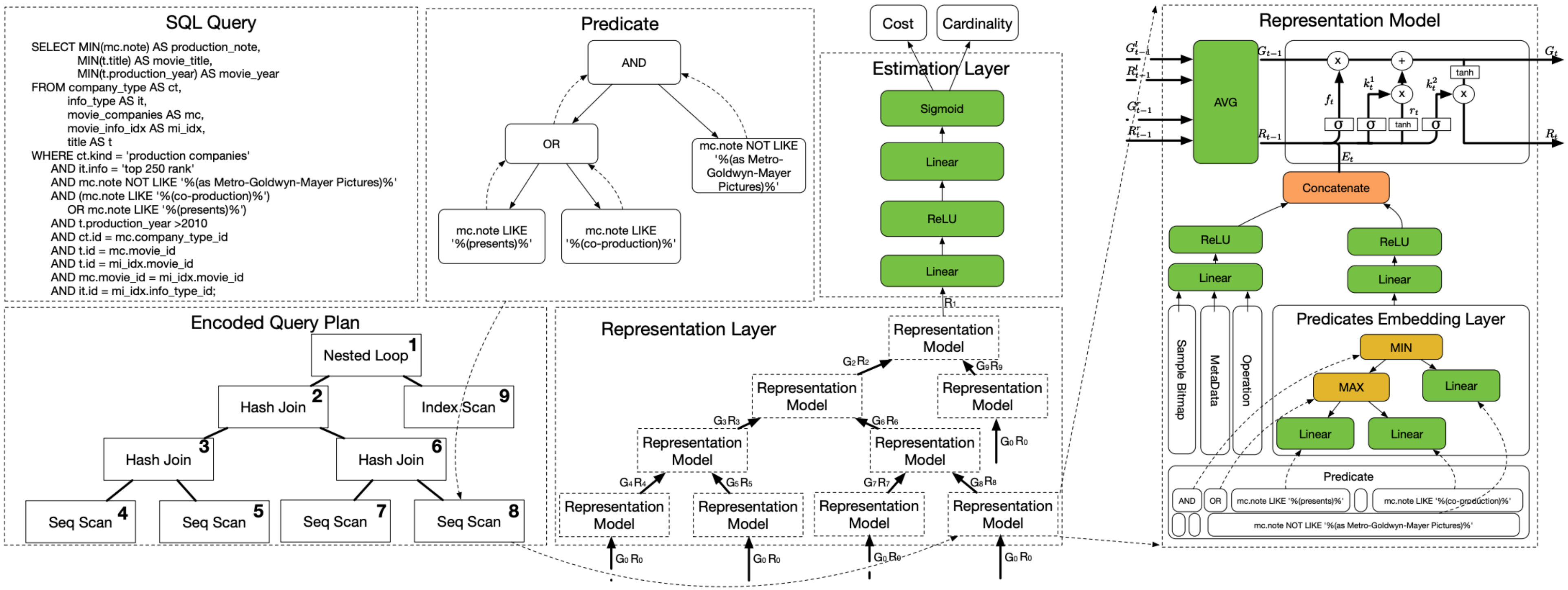
- **Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding →**



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

Tree-LSTM for Cost Estimation

- **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 of Cardinality Estimation

Cardinality Estimation

- Data-driven methods are more effective for single tables.
- Query-driven methods are more efficient than Data-driven methods.
- Data-driven methods are more robust than Query-driven methods.
- Samples are crucial to most Data-driven methods.

Cost Estimation

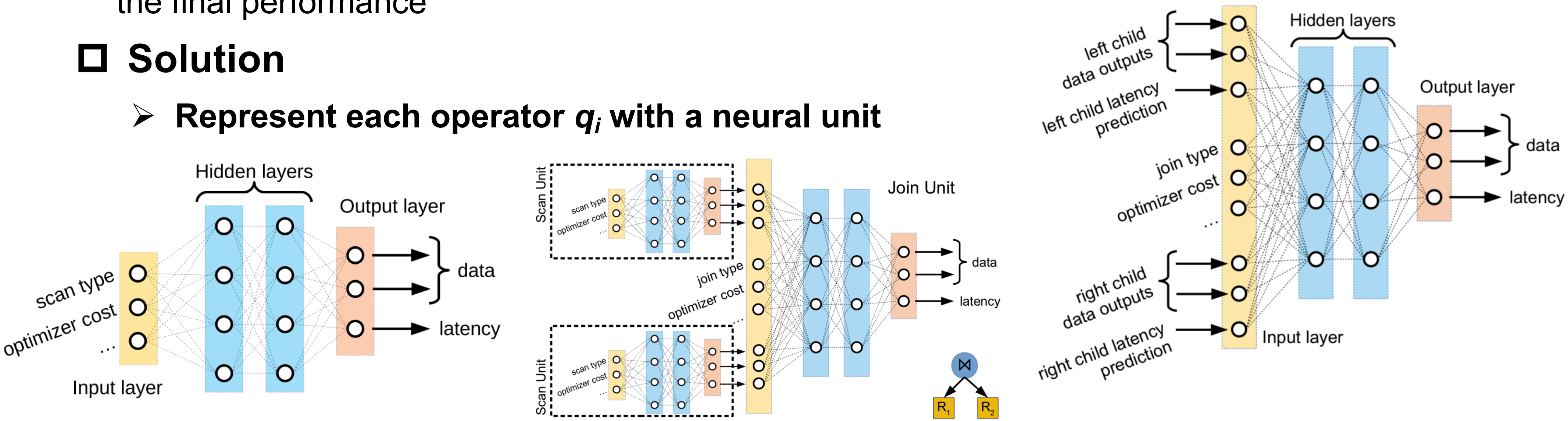
- Accurate cost estimation requires better plan embedding.

Open Problems

- **High Accuracy with small model size and inference latency**
- Adaptivity

Deep Learning for Query Latency Estimation

- **Motivation**
  - Statistical methods fail to estimate based on query structures, and cause great errors;
  - Compared with cost estimation, latency estimation is more complex because (1) it relies on system resources and (2) Cost is one important factor of latency estimation.
- **Core Idea:** Utilize deep learning to capture the relations between input tables, operators, and the final performance
- **Solution**
  - **Represent each operator q_i with a neural unit**



R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. VLDB, 2019.

Deep Learning for Query Latency Estimation

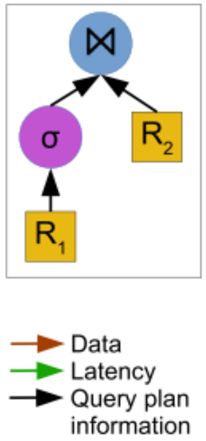
◻ Motivation

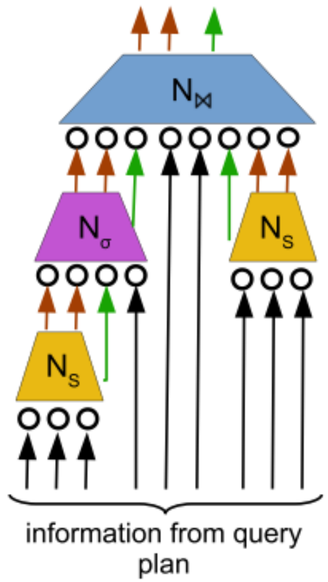
- ◻ Statistical methods fail to estimate based on query structures, and cause great errors;
- ◻ Compared with cost estimation, latency estimation is more complex because (1) it relies on system resources and (2) Cost is one important factor of latency estimation.

◻ Core Idea: Utilize deep learning to capture the relations between input tables, operators, and the final performance

◻ Solution

- ➢ **Represent each operator q_i with a neural unit**
- ➢ **Concatenate neural units by following the query structures**
  - **Example Query *Q* (2 Scans, 1 Join)**
  - **Tree-structured Network for *Q*:**
    - – The outputs of the scan units ($N\sigma$ + Ns)
    - – Input of the join operator ($N_{\bowtie}$)
    - – The final predicted query latency.
  - **Matches the query structure to predict the query latency**

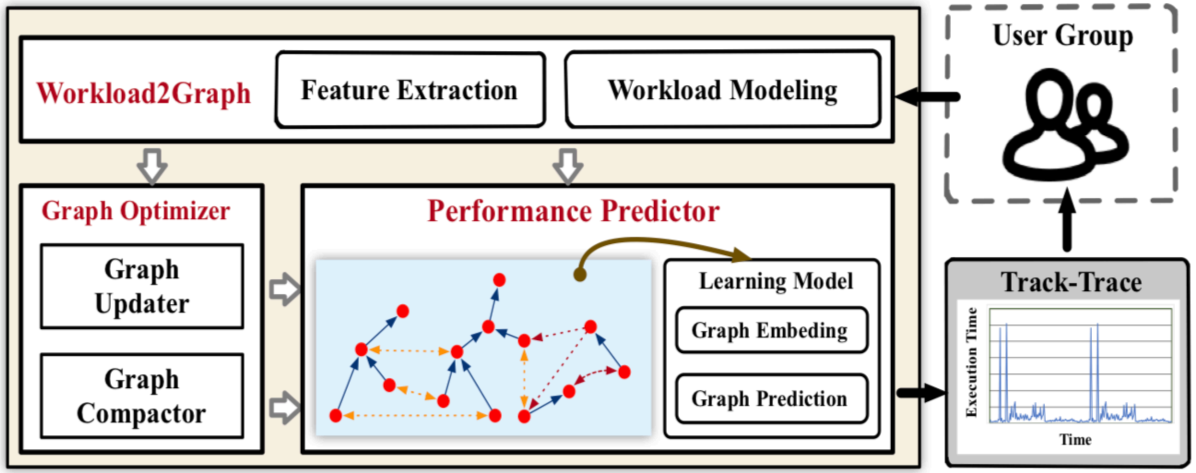




R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. VLDB, 2019.

Graph Embedding for Query Latency Estimation

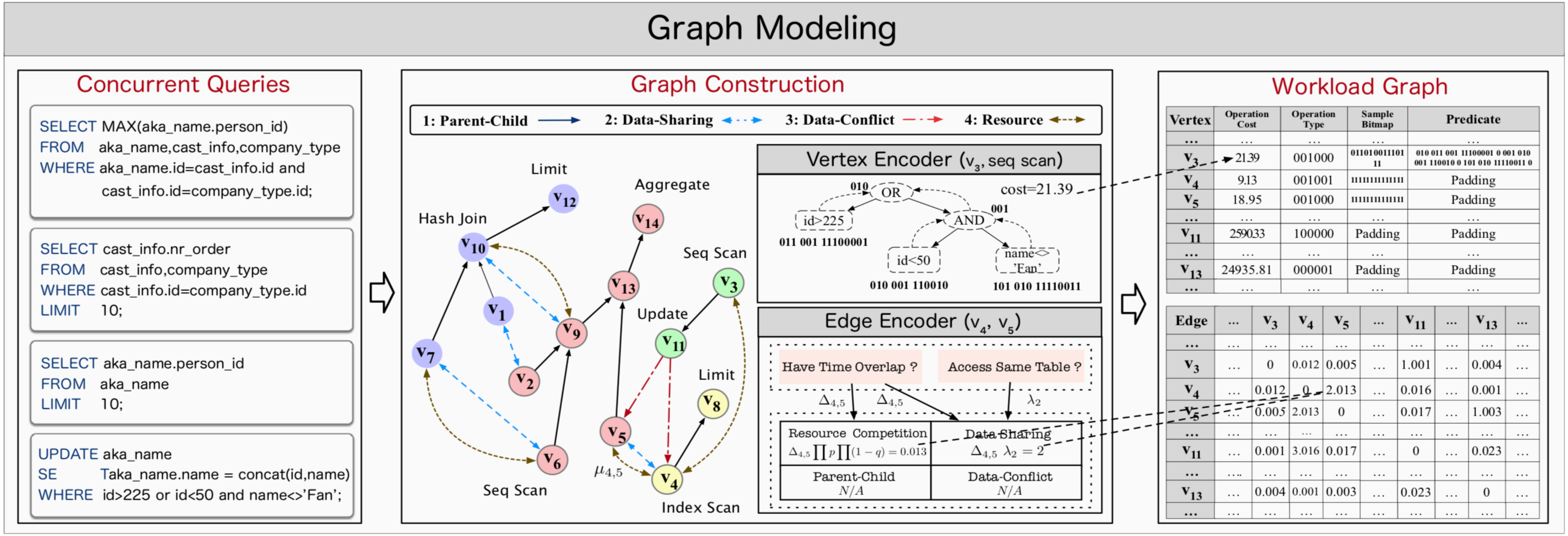
- ☐ Latency Estimation for Concurrent Queries
  - ☐ data-sharing
  - ☐ data-conflict
  - ☐ resource-competition
  - ☐ parent-child relationship



- ☐ Graph-based method
  - Workload2Graph: graph modeling
  - Graph prediction: GNN to predict the latency
  - Graph update: on-the-fly update the model

X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Graph Embedding for Query Latency Estimation

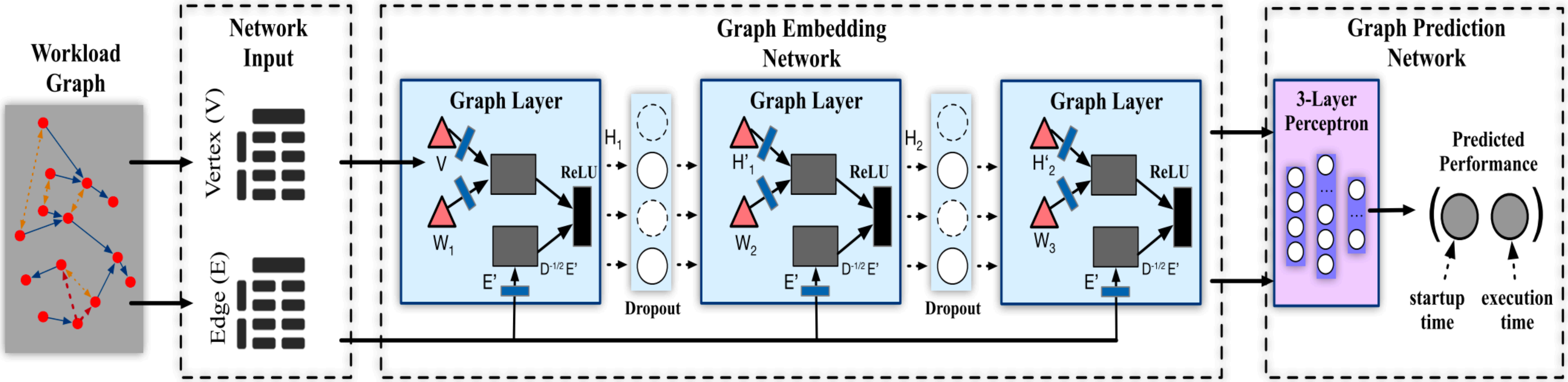


X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Graph Embedding for Query Latency Estimation

☐ Model Construction

- **Performance prediction of concurrent queries**
  - ➢ Represent concurrent queries with a graph model
  - ➢ Embed the graph with graph convolution network and predict the latency of all the operators with a simple dense network



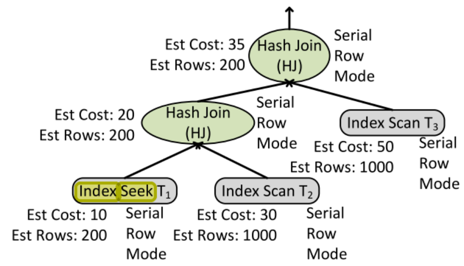
X. Zhou, J. Sun, G. Li, et al. Query Performance Prediction for Concurrent Queries using Graph Embedding. VLDB, 2020.

Deep Learning for Index Benefit Estimation

- **Challenge**
  - **The index/view benefit is hard to evaluate**
    - Multiple evaluation metrics (e.g., index benefit, space cost)
    - Cost estimation by the optimizer is inaccurate
  - **Interactions between existing data structures**
    - Multiple column access, Data refresh
    - Conflicts between MVs

Deep Learning for Index Benefit Estimation

- **Motivation: Critical to estimate index benefits by comparing execution costs of plans with/without created indexes**
- **Core Idea: Take benefit estimation as an ML classification task.**
- **Challenge: It is hard to accurately estimate the index benefits**
- **Solution:**
  - **Prepare training data**
    - **Query Plans + Costs under different indexes**
  - **Train the classification model**
    - **Input: Two query plans with/without indexes**
    - **Output: 1 denotes performance gains; 0 denotes no gains**
  - **Solve the index selection 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.

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

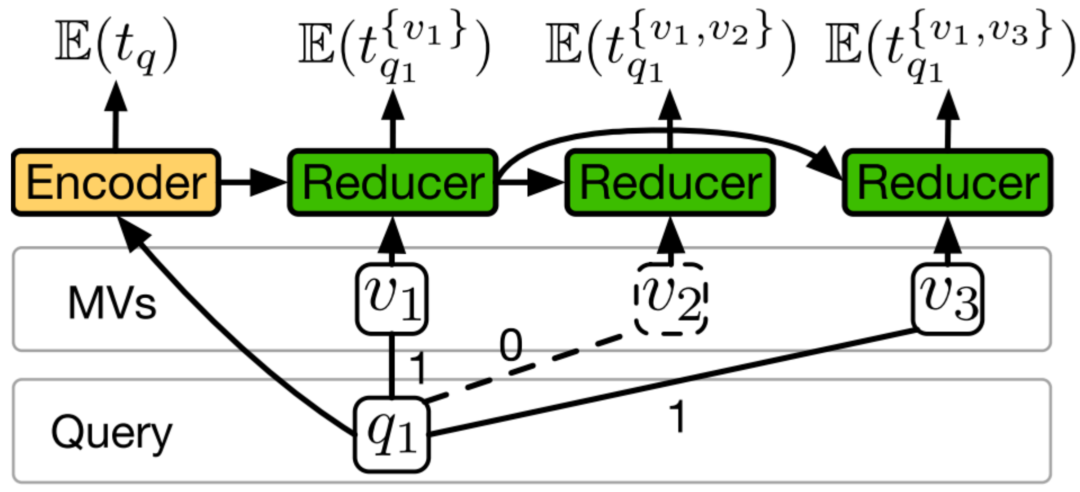
Encoder-Decoder for View Benefit Estimation

☐ Feature Extraction

- **Previous work take candidate views as fixed length →**
- **Encode various number and length of queries and views with an *encoder-reducer model*, which captures correlations with *attention***

☐ Model Construction

- **It is hard to jointly consider MVs thatmay have conflicts →**
- **(1) Split the problem into sub-steps that select one MV;**
- **(2) Use attention-based model to estimate the MV benefit**



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

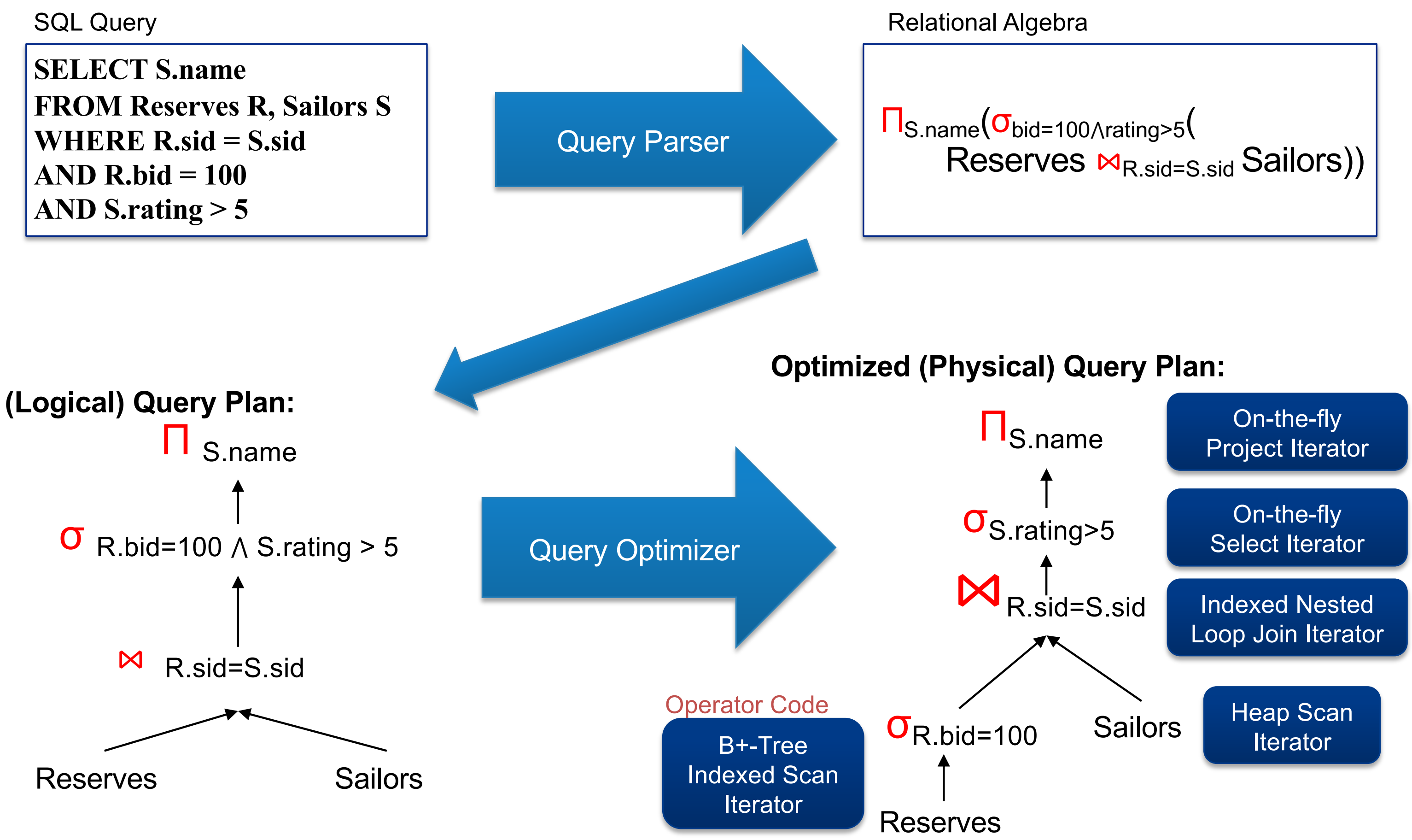
Take-aways of Benefit Estimation

- **Learned utility estimation is more accurate than traditional empirical methods**
- **Learned utility estimation is also accurate for multiple-MV optimization**
- **Query encoding models need to be trained periodically when data update**
- **Open problems:**
  - Benefit prediction for future workload
  - Cost of initialization and future updates

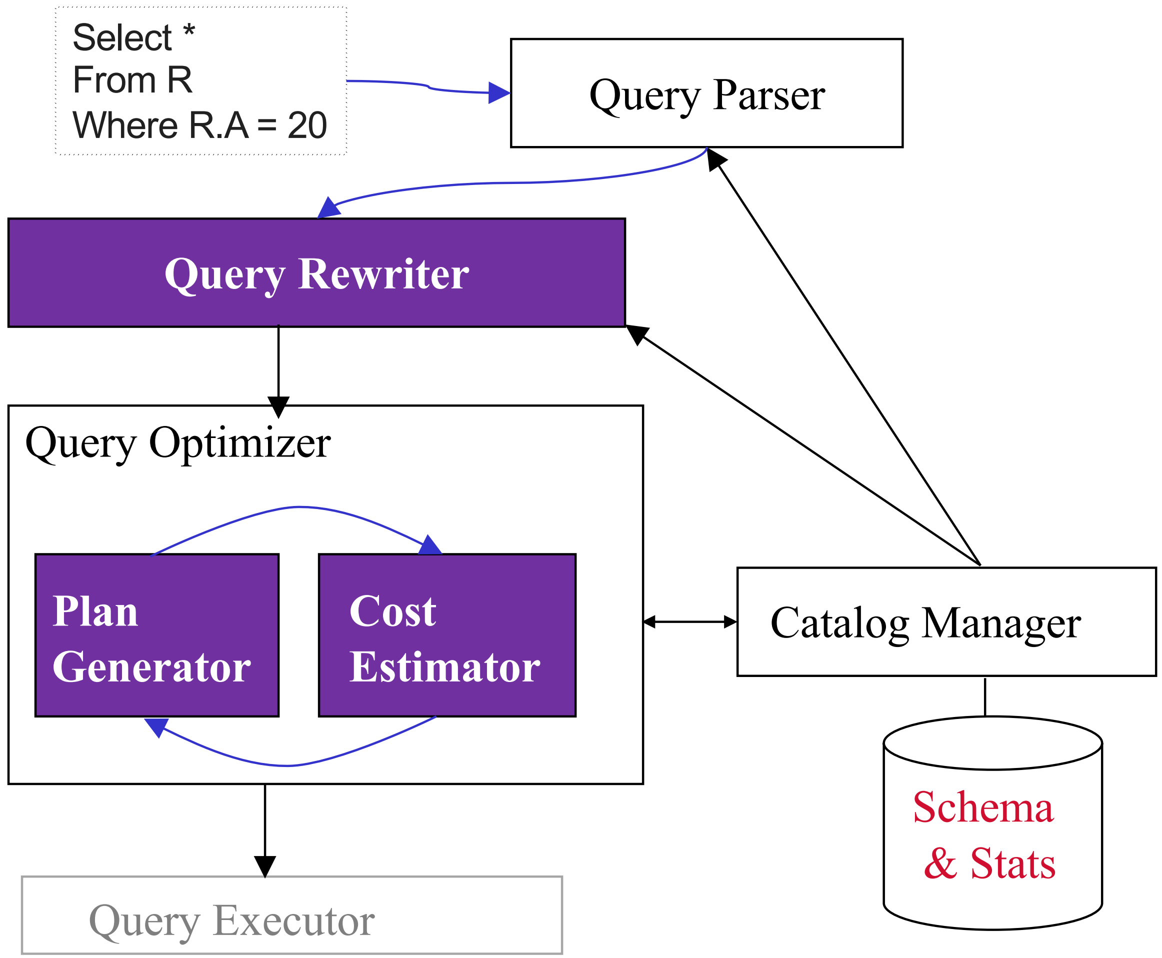
Thanks

Learned Optimizer

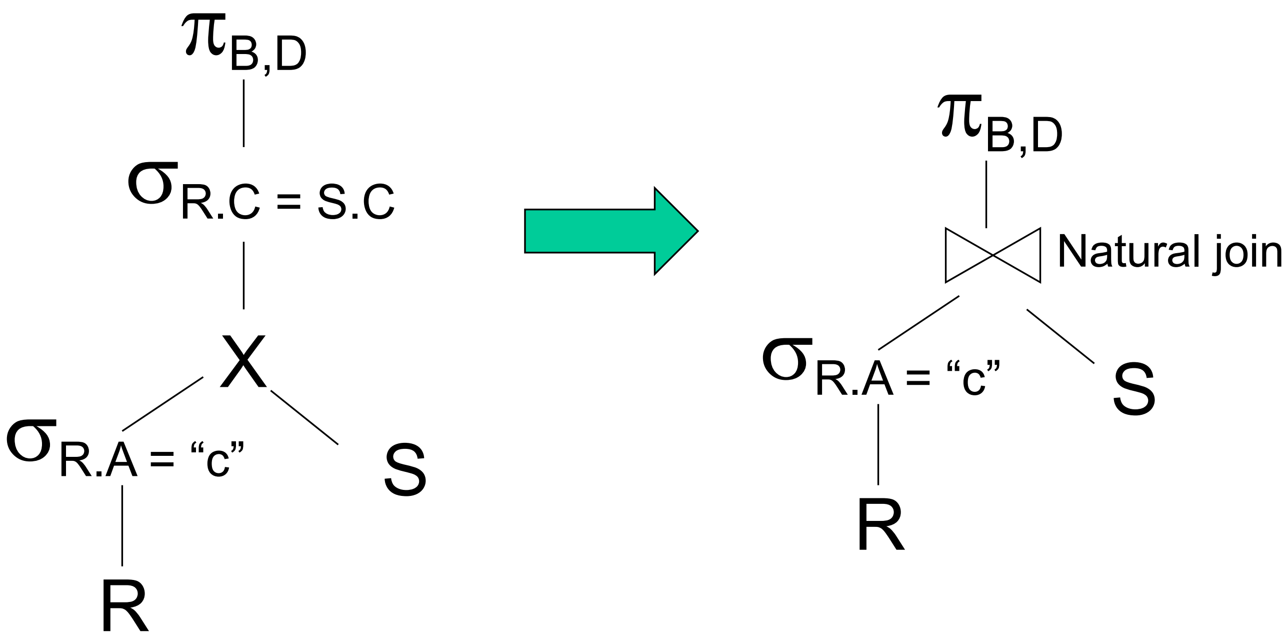
Query Optimizer



Query Optimizer



Logical Optimization – Query Rewrite



$$\pi_{B,D}\,[[\sigma_{R.A=\text{"c"}}(R)] \bowtie S]$$

Query Rewrite

- Transform one logical plan into another equivalent plan (usually with lower cost)
- Theory Guarantee: Equivalences in relational algebra
- Rule-based: Applying rewrite rules
  - Push-down predicates
  - Do projects early
  - Avoid cross-products if possible
  - Use left-deep trees
  - Use of constraints, e.g., uniqueness
  - Subqueries → Joins (we will study this rewrite rule after we do physical plan selection)

Query Rewrite is important to achieve high performance!

Query Rewrite Rules

$$
\begin{aligned}
\sigma_{p_1 \wedge p_2}(e) &\equiv \sigma_{p_1}(\sigma_{p_2}(e)) && (1)\\
\sigma_{p_1}(\sigma_{p_2}(e)) &\equiv \sigma_{p_2}(\sigma_{p_1}(e)) && (2)\\
\Pi_{A_1}(\Pi_{A_2}(e)) &\equiv \Pi_{A_1}(e) && (3)\\
&\quad \text{if } A_1 \subseteq A_2 && \\
\sigma_p(\Pi_A(e)) &\equiv \Pi_A(\sigma_p(e)) && (4)\\
&\quad \text{if } \mathcal{F}(p) \subseteq A && \\
\sigma_p(e_1 \cup e_2) &\equiv \sigma_p(e_1) \cup \sigma_p(e_2) && (5)\\
\sigma_p(e_1 \cap e_2) &\equiv \sigma_p(e_1) \cap \sigma_p(e_2) && (6)\\
\sigma_p(e_1 \setminus e_2) &\equiv \sigma_p(e_1) \setminus \sigma_p(e_2) && (7)\\
\Pi_A(e_1 \cup e_2) &\equiv \Pi_A(e_1) \cup \Pi_A(e_2) && (8)
\end{aligned}
$$

Query Rewrite Rules

$$
\begin{aligned}
e_1 \times e_2 &\equiv e_2 \times e_1 && (9)\\
e_1 \bowtie_p e_2 &\equiv e_2 \bowtie_p e_1 && (10)\\
(e_1 \times e_2) \times e_3 &\equiv e_1 \times (e_2 \times e_3) && (11)\\
(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 &\equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3) && (12)\\
\sigma_p(e_1 \times e_2) &\equiv e_1 \bowtie_p e_2 && (13)\\
\sigma_p(e_1 \times e_2) &\equiv \sigma_p(e_1) \times e_2 && (14)\\
& \quad \text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1) &&\\
\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) &\equiv \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 && (15)\\
& \quad \text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1) &&\\
\Pi_A(e_1 \times e_2) &\equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) && (16)\\
& \quad \text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2) &&
\end{aligned}
$$

Phases of Logical Query Optimization

1. break up conjunctive selection predicates (equivalence (1) →)
2. push selections down (equivalence (2) →, (14) →)
3. introduce joins (equivalence (13) →)
4. determine join order (equivalence (9), (10), (11), (12))
5. push down projections (equivalence (3) ←, (4) ←, (16) →)

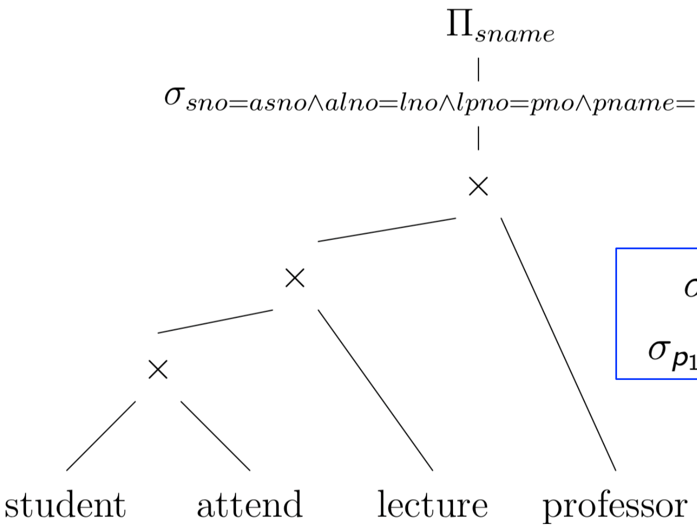
$$
\begin{aligned}
\sigma_{p_1 \wedge p_2}(e) &\equiv \sigma_{p_1}(\sigma_{p_2}(e)) && (1)\\
\sigma_{p_1}(\sigma_{p_2}(e)) &\equiv \sigma_{p_2}(\sigma_{p_1}(e)) && (2)\\
\Pi_{A_1}(\Pi_{A_2}(e)) &\equiv \Pi_{A_1}(e) && (3)\\
&\quad \text{if } A_1 \subseteq A_2 &&\\
\sigma_p(\Pi_A(e)) &\equiv \Pi_A(\sigma_p(e)) && (4)\\
&\quad \text{if } \mathcal{F}(p) \subseteq A &&\\
\sigma_p(e_1 \cup e_2) &\equiv \sigma_p(e_1) \cup \sigma_p(e_2) && (5)\\
\sigma_p(e_1 \cap e_2) &\equiv \sigma_p(e_1) \cap \sigma_p(e_2) && (6)\\
\sigma_p(e_1 \setminus e_2) &\equiv \sigma_p(e_1) \setminus \sigma_p(e_2) && (7)\\
\Pi_A(e_1 \cup e_2) &\equiv \Pi_A(e_1) \cup \Pi_A(e_2) && (8)
\end{aligned}
$$

$$
\begin{aligned}
e_1 \times e_2 &\equiv e_2 \times e_1 && (9)\\
e_1 \bowtie_p e_2 &\equiv e_2 \bowtie_p e_1 && (10)\\
(e_1 \times e_2) \times e_3 &\equiv e_1 \times (e_2 \times e_3) && (11)\\
(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 &\equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3) && (12)\\
\sigma_p(e_1 \times e_2) &\equiv e_1 \bowtie_p e_2 && (13)\\
\sigma_p(e_1 \times e_2) &\equiv \sigma_p(e_1) \times e_2 && (14)\\
&\quad \text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1) &&\\
\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) &\equiv \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 && (15)\\
&\quad \text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1) &&\\
\Pi_A(e_1 \times e_2) &\equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) && (16)\\
&\quad \text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2) &&
\end{aligned}
$$

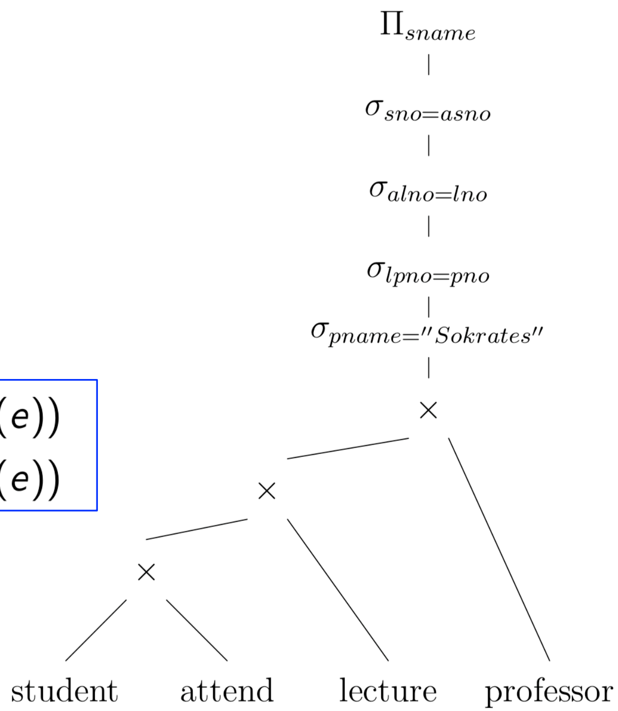
Step 1: Break up conjunctive selection predicates

select distinct s.sname
from student s, attend a, lecture l, professor p
where s.sno = a.asno and a.alno = l.lno and l.lpno = p.pno and p.pname =" Sokrates"

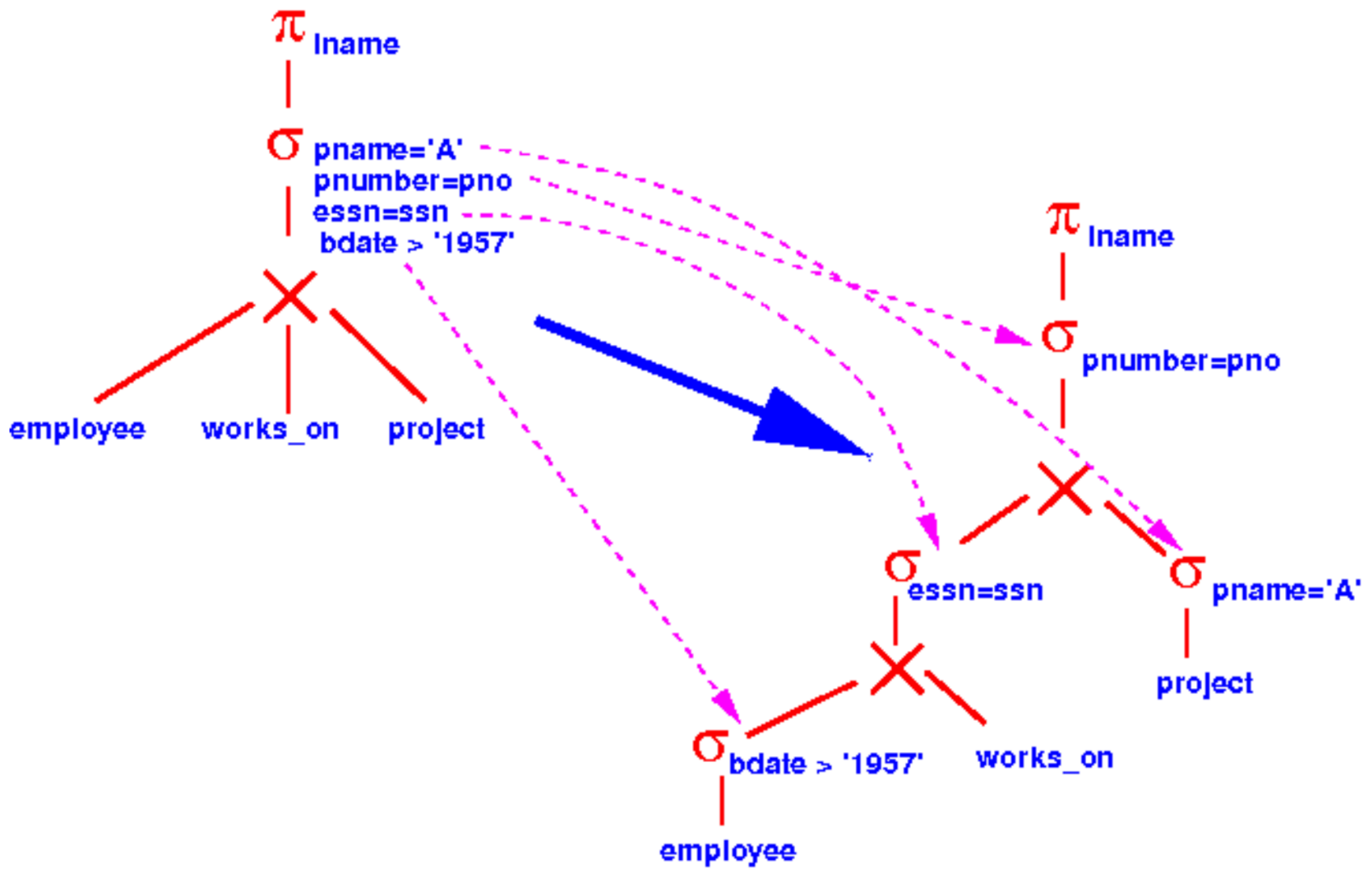
selection with simple predicates can be moved around easier



$$\begin{aligned} \sigma_{p_1 \wedge p_2}(e) &\equiv \sigma_{p_1}(\sigma_{p_2}(e)) \\ \sigma_{p_1}(\sigma_{p_2}(e)) &\equiv \sigma_{p_2}(\sigma_{p_1}(e)) \end{aligned}$$

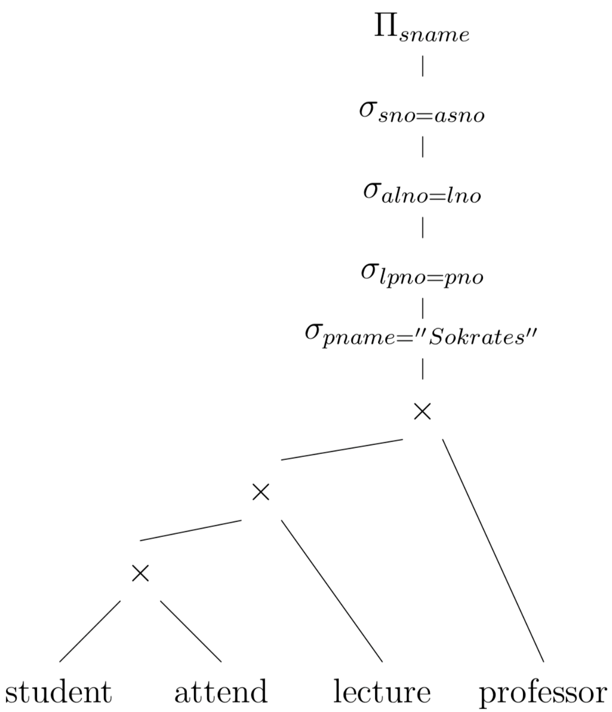


Step 1: Break up conjunctive selection predicates

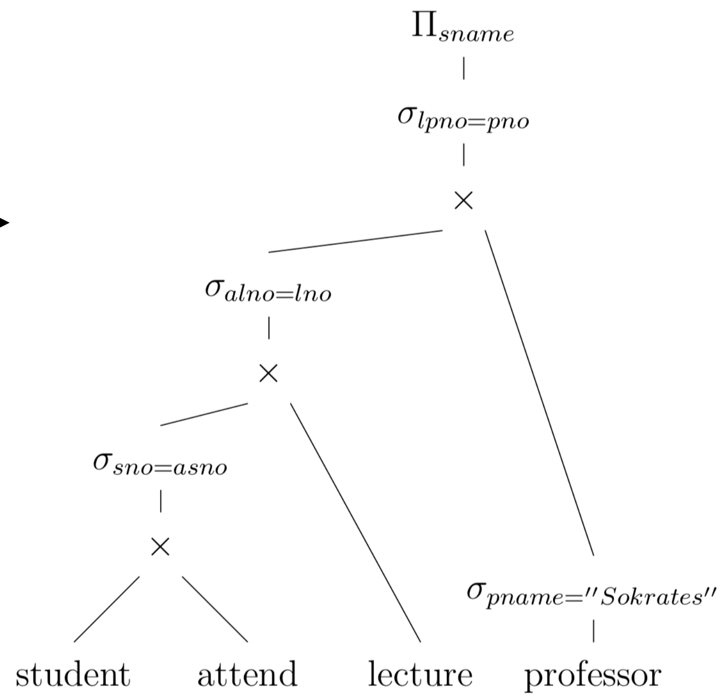


Step 2: Push Selections Down

reduce the number of tuples early, reduces the work for later operators

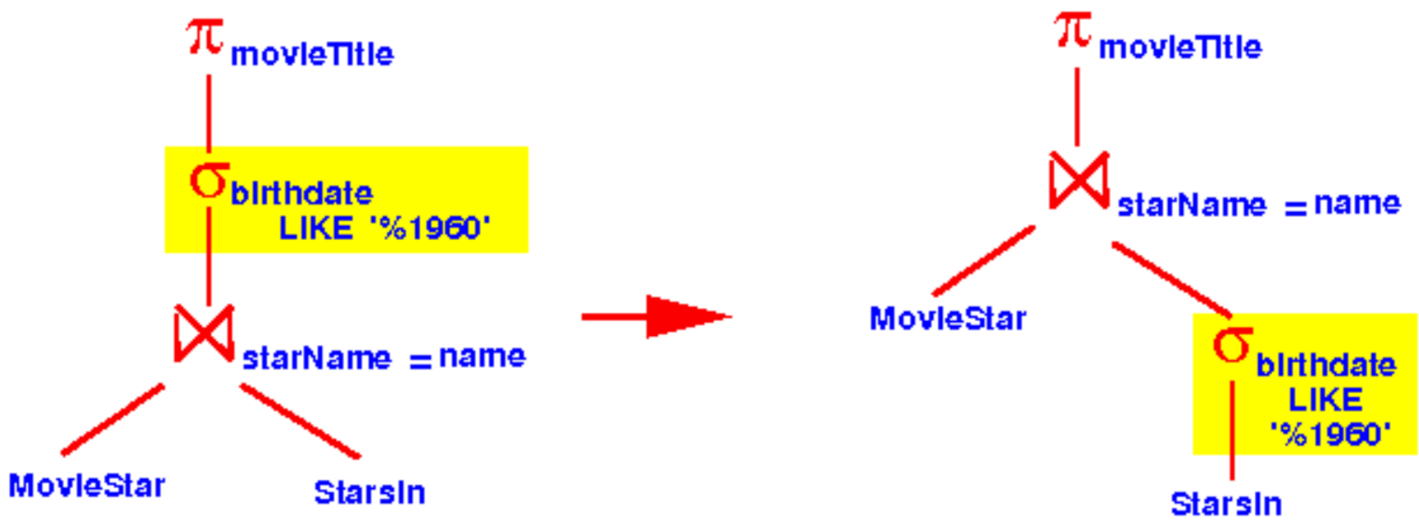


$$
\begin{array}{rcl}
\sigma_p(e_1 \times e_2) & \equiv & \sigma_p(e_1) \times e_2 \\
 & & \text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1) \\
\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) & \equiv & \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 \\
 & & \text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1)
\end{array}
$$



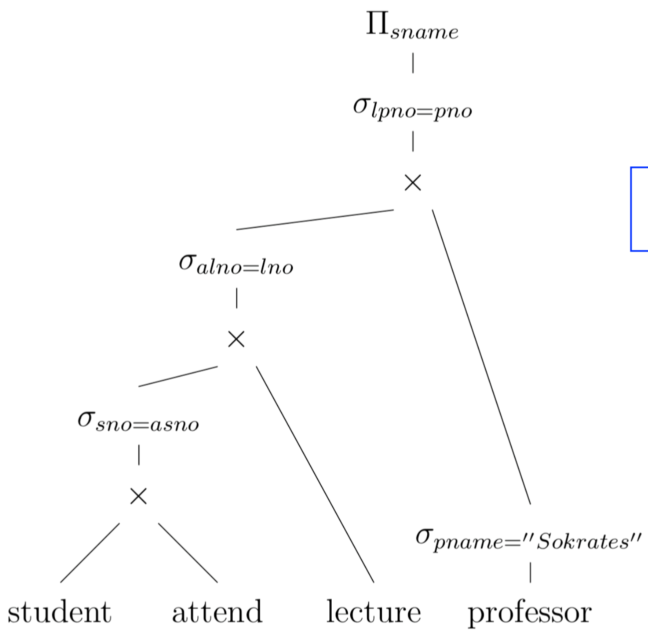
Step 2: Push Selections Down

reduce the number of tuples early, reduces the work for later operators

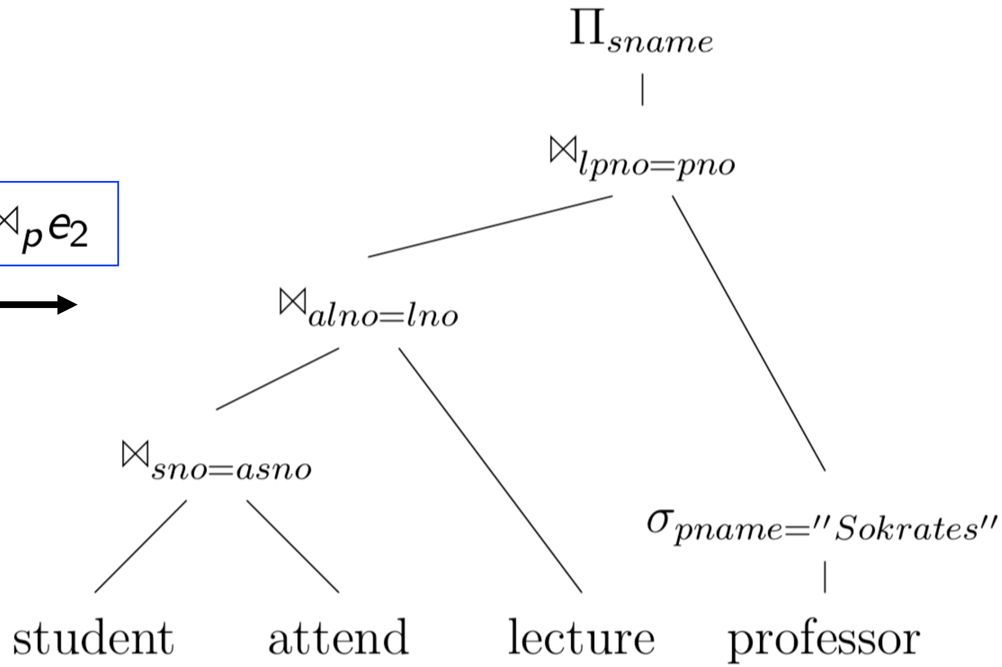


Step 3: Introduce Joins

joins are cheaper than cross products



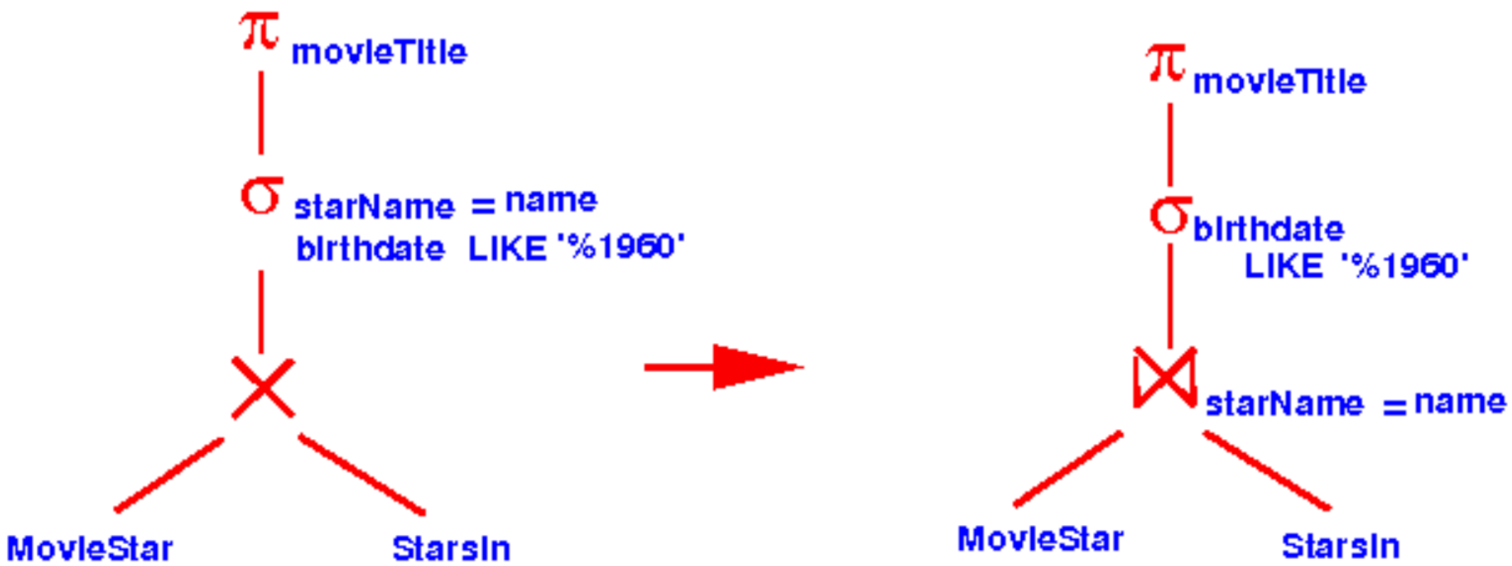
$$\boxed{\sigma_p(e_1 \times e_2) \quad \equiv \quad e_1 \bowtie_p e_2}$$



Step 3: Introduce Joins

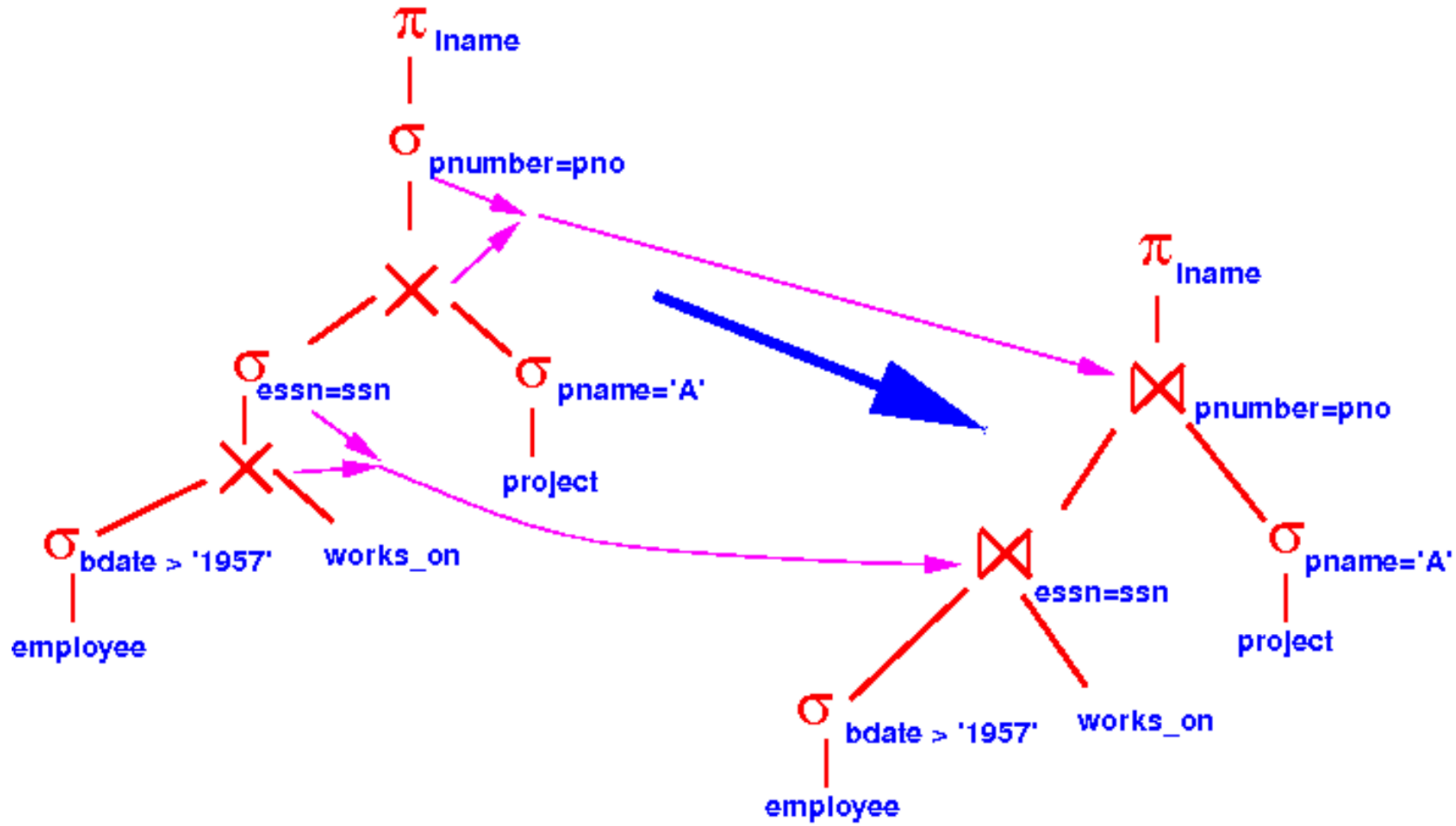
Cartesian Product to Natural Join

$\sigma_{starName=name}$ (MovieStar $\times$ StarsIn) $\equiv$ MovieStar $\bowtie_{starName=name}$ StarsIn



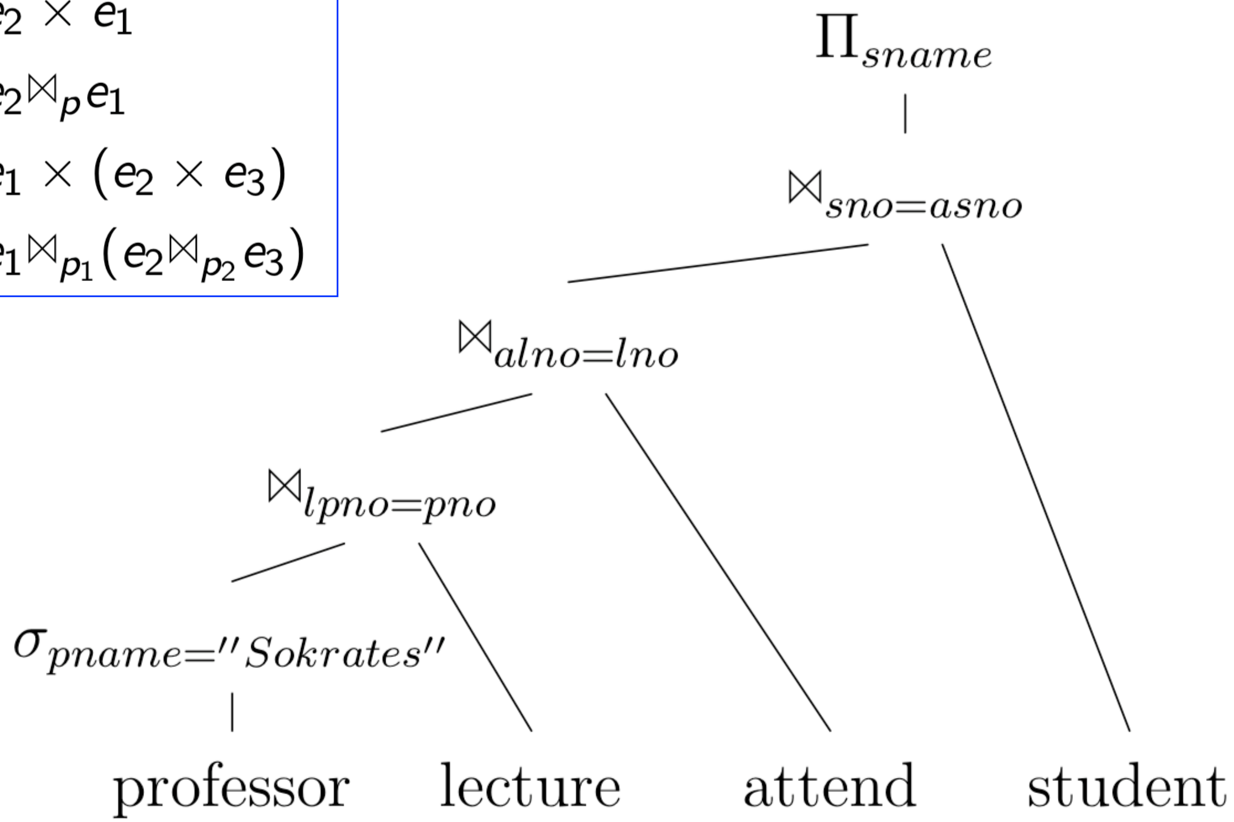
Step 3: Introduce Joins

Replace σ + × with ⋈:



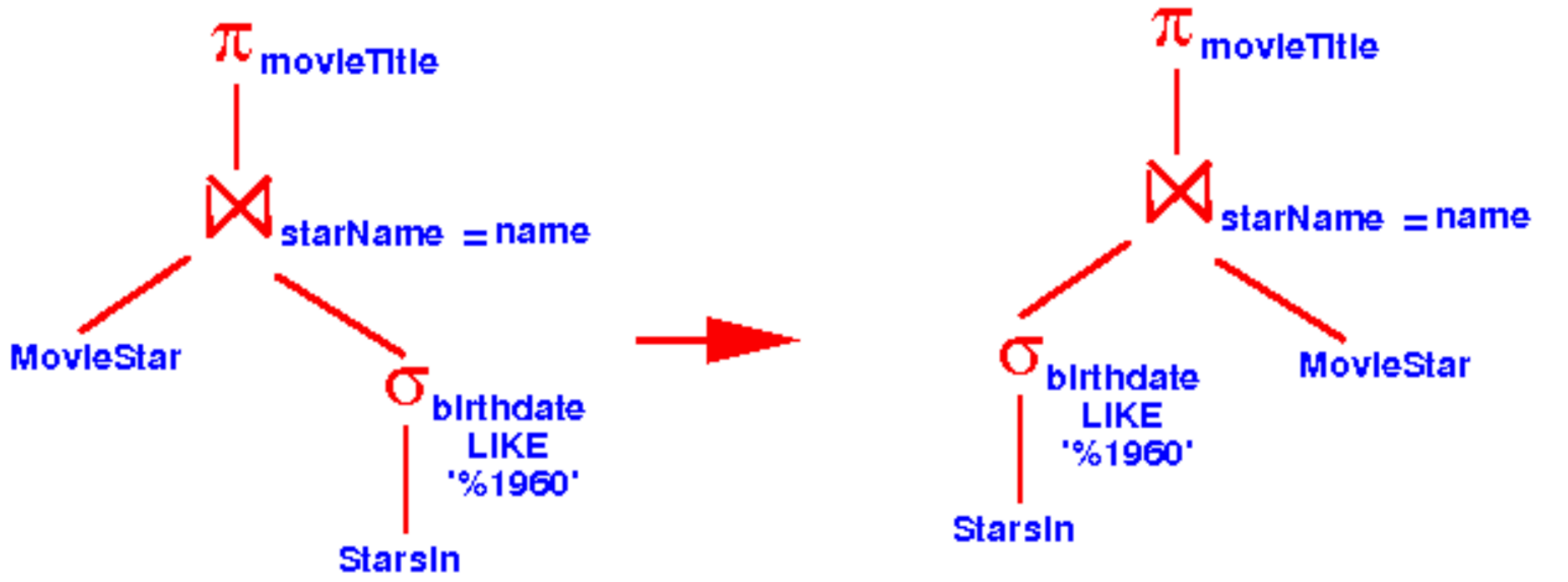
Step 4: Determine Join Order

$$
\begin{aligned}
e_1 \times e_2 &\equiv e_2 \times e_1 \\
e_1 \bowtie_p e_2 &\equiv e_2 \bowtie_p e_1 \\
(e_1 \times e_2) \times e_3 &\equiv e_1 \times (e_2 \times e_3) \\
(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 &\equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3)
\end{aligned}
$$



Step 4: Determine Join Order

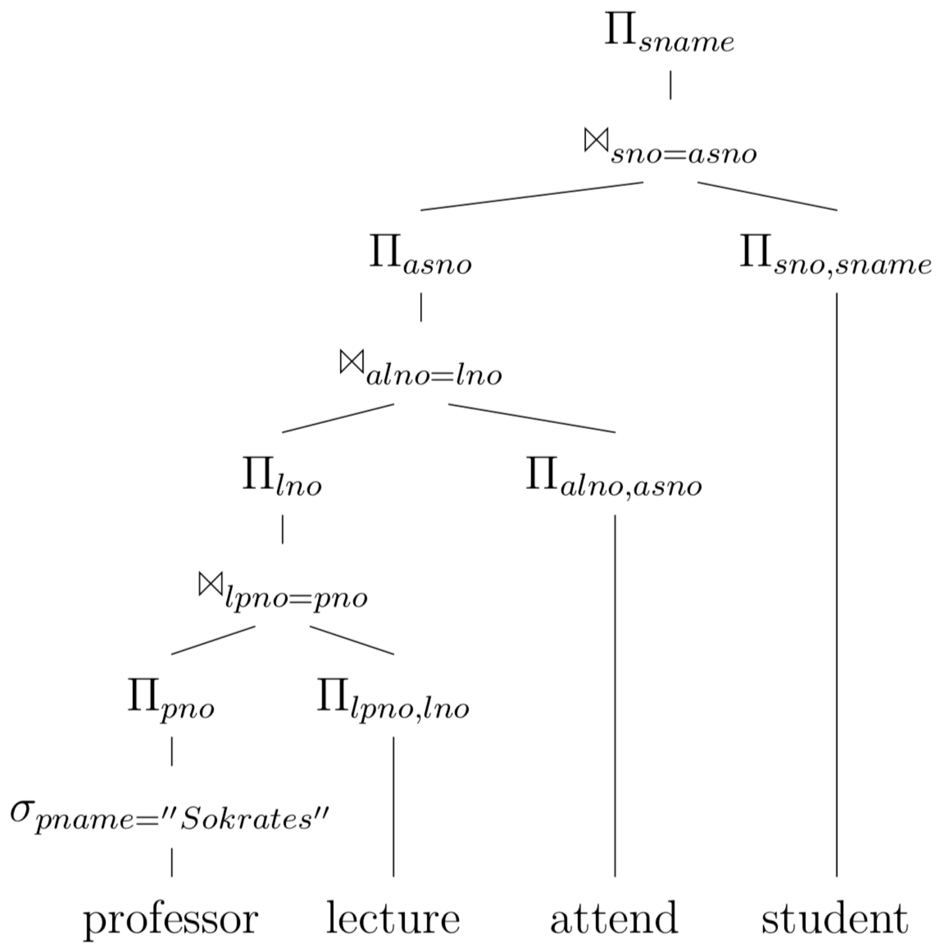
smaller **input relation** as the ***left*** **input relation** in a **join (⋈) operator**



Step 5: Introduce and Push Down Projections

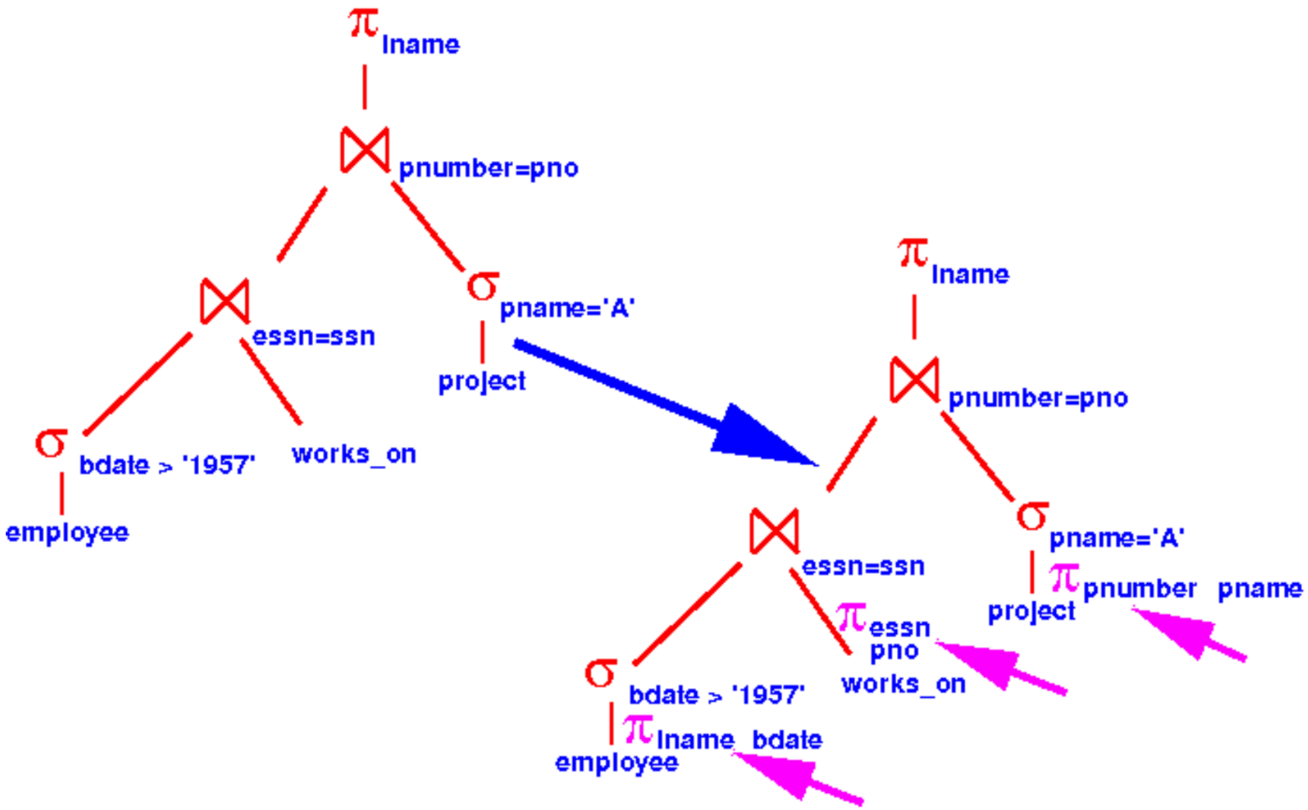
- eliminate redundant attributes
- only before pipeline breakers

$$\Pi_A(e_1 \times e_2) \equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) \quad \text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2)$$

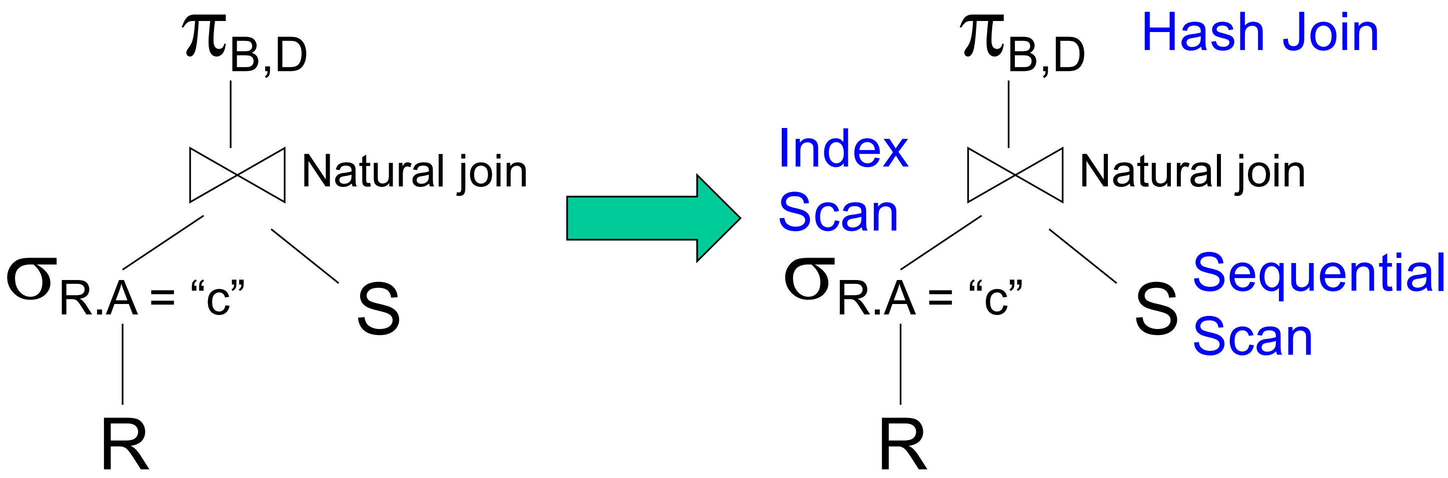


Step 5: Introduce and Push Down Projections

Remove the unused attributes by inserting projection (π):



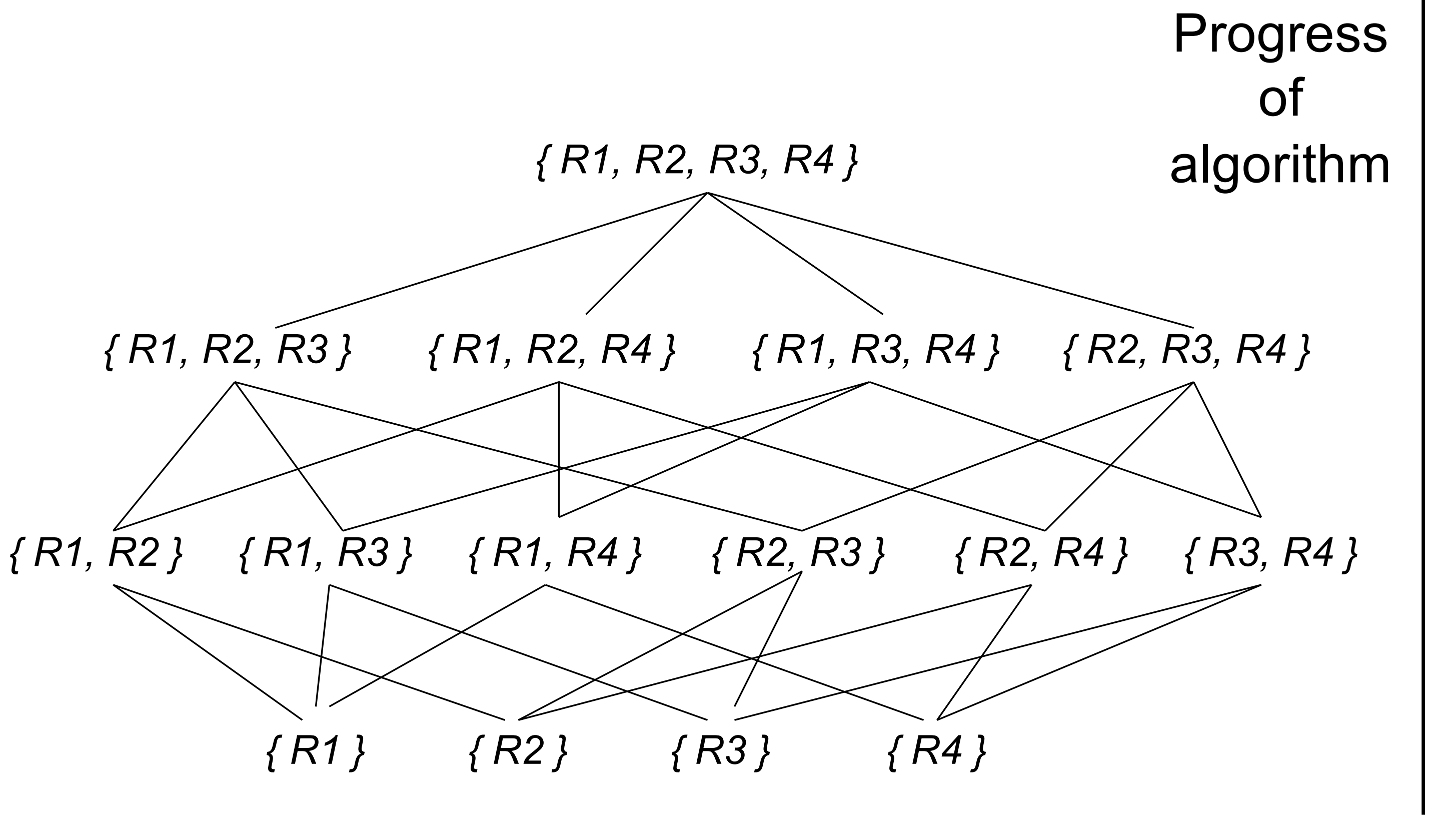
Physical Optimization



$$\pi_{B,D}\,[[\sigma_{R.A=\text{“c”}}(R)] \bowtie S]$$

Join Order Selection

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



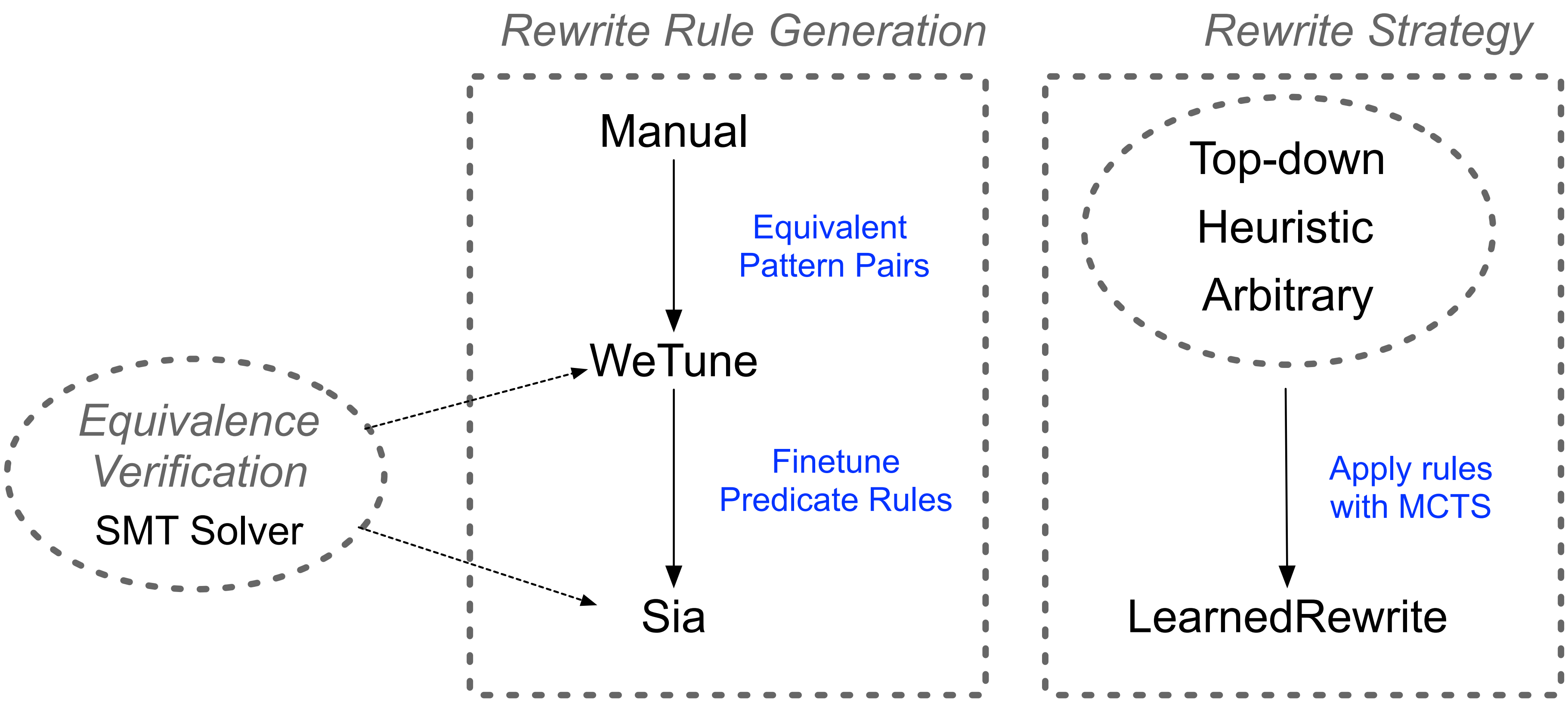
Dynamic Programming

Relations joined	size of join result	Cost of join	Best join ordering (plan)
R	1000	0	R
S	1000	0	S
T	1000	0	T
U	1000	0	U
R, S	5000	0	R ⋈ S
R, T	1000000	0	R ⋈ T
R, U	10000	0	R ⋈ U
S, T	2000	0	S ⋈ T
S, U	1000000	0	S ⋈ U
T, U	1000	0	T ⋈ U
R,S,T	10000	2000	(S ⋈ T) ⋈ R
R,S,U	50000	5000	(R ⋈ S) ⋈ U
R,T,U	10000	1000	(T ⋈ U) ⋈ R
S,T,U	2000	1000	(T ⋈ U) ⋈ S
R,S,T,U	???	???	???

Selinger Algorithm

- Step 1: Enumerate all access paths for a single relation
  - File scan or index scan
  - Keep the cheapest for each interesting order
- Step 2: Consider all ways to join two relations
  - Use result from step 1 as the outer relation
  - Consider every other possible relation as inner relation
  - Estimate cost when using sort-merge or nested-loop join
  - Keep the cheapest for each interesting order
- Steps 3 and later: Repeat for three relations, etc.

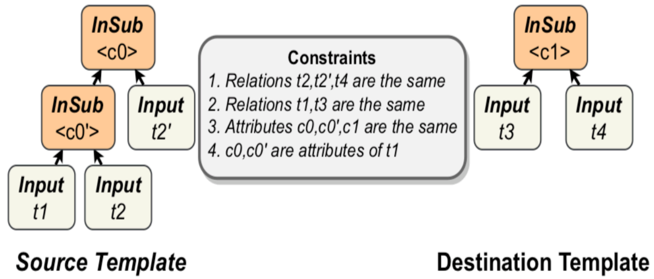
Learning-based Query Rewrite



Learning New Rewrite Rules

- **Motivation: Identify new rules to gain performance improvement**
- **Basic Idea: Extract relatively simple query pattern pairs from the public datasets and synthesize new rewrite rules**
- **Challenge: (1) How to generate new rewrite rules; (2) How to verify the rewrite equivalence**

Original Query	Opt. By Existing DB	Ideal (WeTune)
q0: SELECT * FROM labels WHERE id IN (SELECT id FROM labels WHERE id IN (SELECT id FROM labels WHERE project_id=10) ORDER BY title ASC)	q1: SELECT * FROM labels WHERE id IN (SELECT id FROM labels WHERE project_id=10)	q2: SELECT * FROM labels WHERE project_id=10
q3: SELECT id FROM notes WHERE type='D' AND id IN (SELECT id FROM notes WHERE commit_id=7)	Unchanged	q4: SELECT id FROM notes WHERE type='D' AND commit_id=7

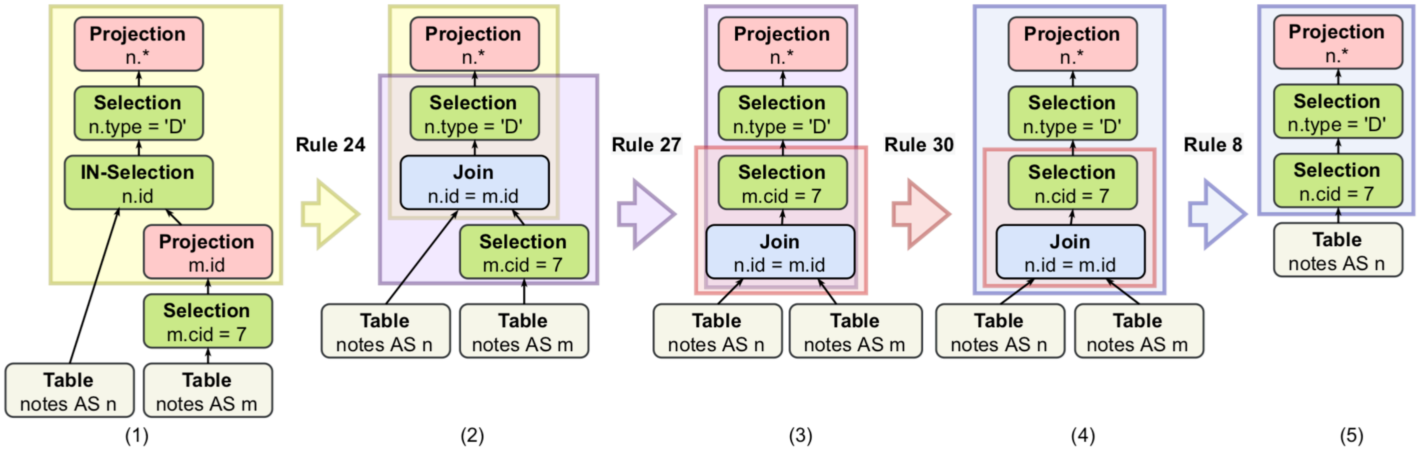



```
q5: ... FROM T WHERE T.x IN (SELECT R.y FROM R)
                AND T.x IN (SELECT R.y FROM R)
q6: ... FROM T WHERE T.x IN (SELECT R.y FROM R)
```

Wang Z, Zhou Z, Yang Y, et al. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. SIGMOD, 2022.

Learning New Rewrite Rules

- Motivation: Identify new rules to gain performance improvement
- Basic Idea: Extract relatively simple query pattern pairs from the public datasets and synthesize new rewrite rules
- Challenge: (1) How to generate new rewrite rules; (2) How to verify the rewrite equivalence

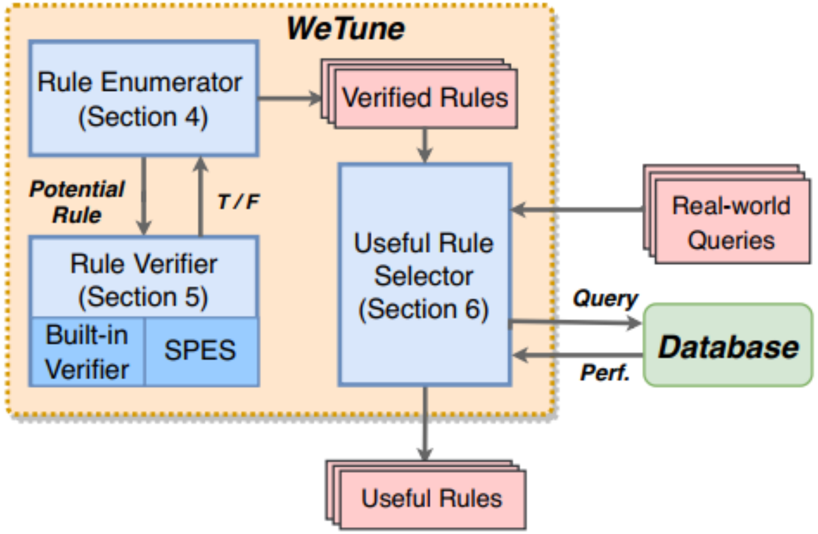



```
Original    SELECT n.* FROM notes AS n WHERE n.type = 'D'
              AND n.id IN (SELECT m.id FROM notes AS m WHERE m.commit_id = 7)
Optimized   SELECT n.* FROM notes AS n WHERE n.type = 'D' AND n.commit_id = 7
```

Wang Z, Zhou Z, Yang Y, et al. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. SIGMOD, 2022.

Learning New Rewrite Rules

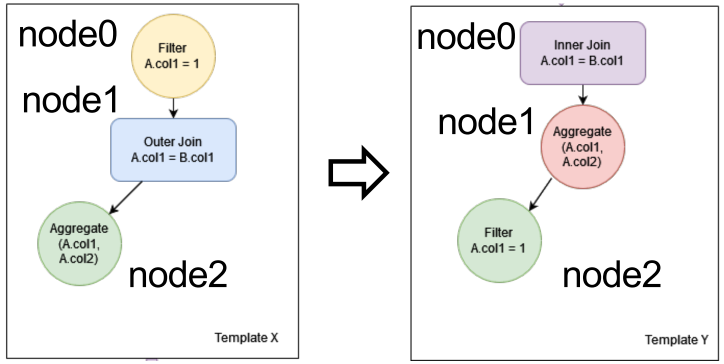
- **Motivation: Identify new rules to gain performance improvement**
- **Basic Idea: Extract relatively simple query pattern pairs from the public datasets and synthesize new rewrite rules**
- **Challenge: (1) How to generate new rewrite rules; (2) How to verify the rewrite equivalence**
- **Solution**
  - **Generate rules via rule enumerator**
    - Rule: *(source pattern, destination pattern, constraints)*
  - **Verify rule equivalence via SMT solver**
    - Only queries with no more than 4 operators
  - **Use verified rules to greedily rewrite queries**



Wang Z, Zhou Z, Yang Y, et al. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. SIGMOD, 2022.

Learning New Rewrite Rules

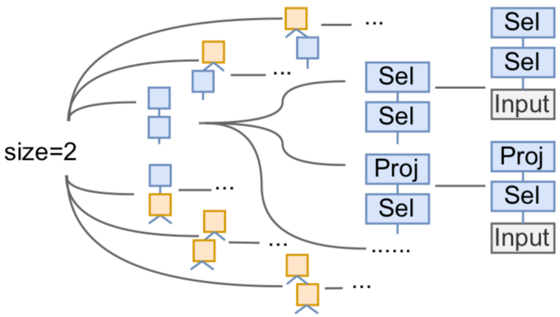
- **Generate rules via rule enumerator**
  - Rule: *(source pattern, destination pattern, constraints)*



- **Verify rule equivalence via SMT solver**
  - Only queries with no more than 4 operators
  - Transform: SQL Query → U-expression → FOL Formula
  - Q(X) and Q(Y) are equivalent *iff,*
    - $query_{FOL}(X) \rightarrow query_{FOL}(Y)$ && $query_{FOL}(Y) \rightarrow query_{FOL}(X)$

U-expression	FOL formula
$E_1 = E_2$	$\mathrm{Tr}(E_1) = \mathrm{Tr}(E_2)$
$E_1 + E_2$	$\mathrm{Tr}(E_1) + \mathrm{Tr}(E_2)$
$E_1 \times E_2$	$\mathrm{Tr}(E_1) \times \mathrm{Tr}(E_2)$
$\|\|E\|\|$	$ite(\mathrm{Tr}(E) > 0, 1, 0)$
$not(E)$	$ite(\mathrm{Tr}(E) > 0, 0, 1)$
$[p]$	$ite(p, 0, 1)$
$\|\|\sum_x E\|\|$	$ite(\exists x.\mathrm{Tr}(E) > 0, 1, 0)$
$not(\sum_x E)$	$ite(\exists x.\mathrm{Tr}(E) > 0, 0, 1)$
$\sum_x f(x) = 1$	$\exists x.(f(x) = 1 \wedge (\forall y. y \neq x \Rightarrow f(y) = 0))$
$\sum_x r(x) \times E = \sum_x r(x) \times E'$	$\forall x. r(x) \times \mathrm{Tr}(E) = r(x) \times \mathrm{Tr}(E')$
$\sum_x r(x) \times E = \sum_{x,y} r(x) \times E' \times D_y$	$\forall x.((r(x) \times \mathrm{Tr}(E) = r(x) \times \mathrm{Tr}(E')) \wedge ((r(x) \times \mathrm{Tr}(E) = 0) \vee \mathrm{Tr}(\sum_y D_y = 1)))$

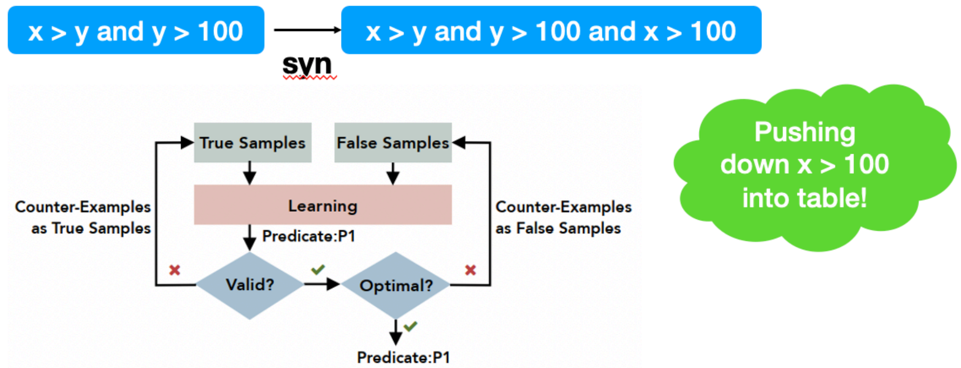
- **Use verified rules to greedily rewrite queries**



Wang Z, Zhou Z, Yang Y, et al. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. SIGMOD, 2022.

Finetune Predicate Rules

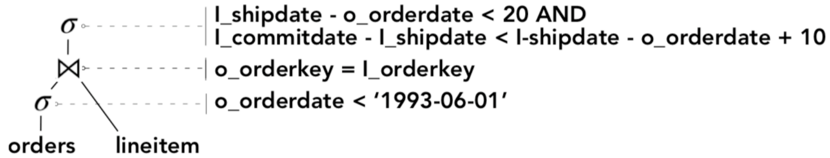
- ☐ **Motivation: Traditional predicate-pushdown is less powerful in many cases**
- ☐ **Core Idea: Synthesize new predicates that are both valid (semantic equivalence) and optimal (performance gain)**
- ☐ **Challenge: (1) How to generate new predicates; (2) How to verify the predicates are valid and optimal.**
- ☐ **Solution**
  - **Build a classification model (SVM)**
    - Classification Model ⇔ 0/1 ⇔ New Predicate
  - **Use true/false samples to finetune the model**
    - **Valid:** if the model filters out samples in origin predicate, it is not valid (true samples);
    - **Optimal:** if the model accepts samples not in origin predicate, it is not optimal (false samples).



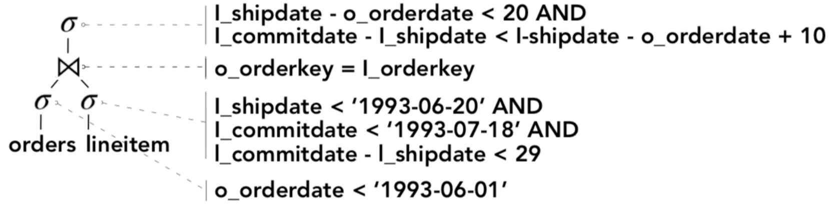
Qi Zhou, Joy Arulraj, Shamkant B, et al. SIA: Optimizing Queries using Learned Predicates. SIGMOD, 2021.

Finetune Predicate Pushdown Rules

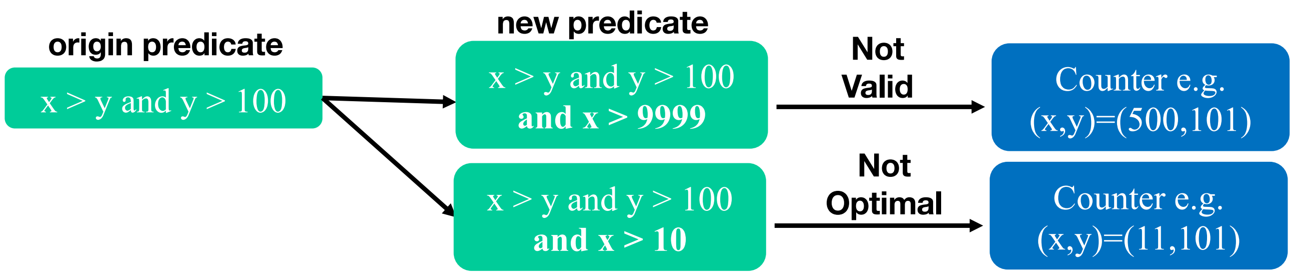
- ☐ **Build a classification model (SVM)**
  - ➢ Classification Model ⇔ 0/1 ⇔ New Predicate
- ➢ **Use true/false samples to finetune the model**
  - ➢ **Valid:** if the model filters out samples in origin predicate, it is not valid (true samples);
  - ➢ **Optimal:** if the model accepts samples not in origin predicate, it is not optimal (false samples).

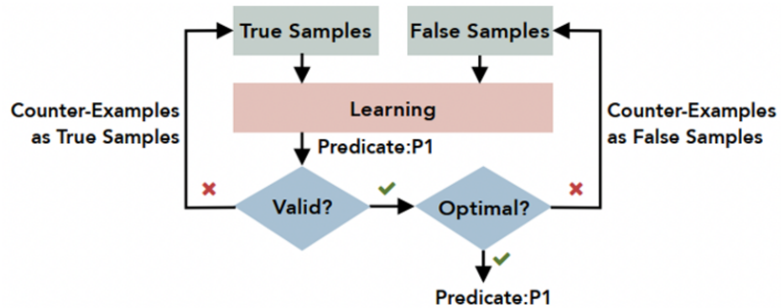


(a) Logical Plan for Q1



(b) Logical Plan for Q2





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

Learning-based Query Rewrite

- **Why Heuristics → Learning-based?**
  - **Many real-world queries are not well-written**
    - Terrible operations (e.g., subqueries/joins, union/union all) ;
    - Look pretty to humans, but physically inefficient (e.g., take subqueries as temporary tables);
  - **Existing methods are based on heuristic rules**
    - Top-down rewrite order may not lead to optimal rewrites (e.g., remove aggregates before pulling up subqueries)
    - Some cases may not be covered by existing rules
  - **Trade-off in SQL Rewrite**
    - **Best Performance:** Enumerate for the best rewrite order
    - **Minimal Latency:** SQL Rewrite requires low overhead (milliseconds)

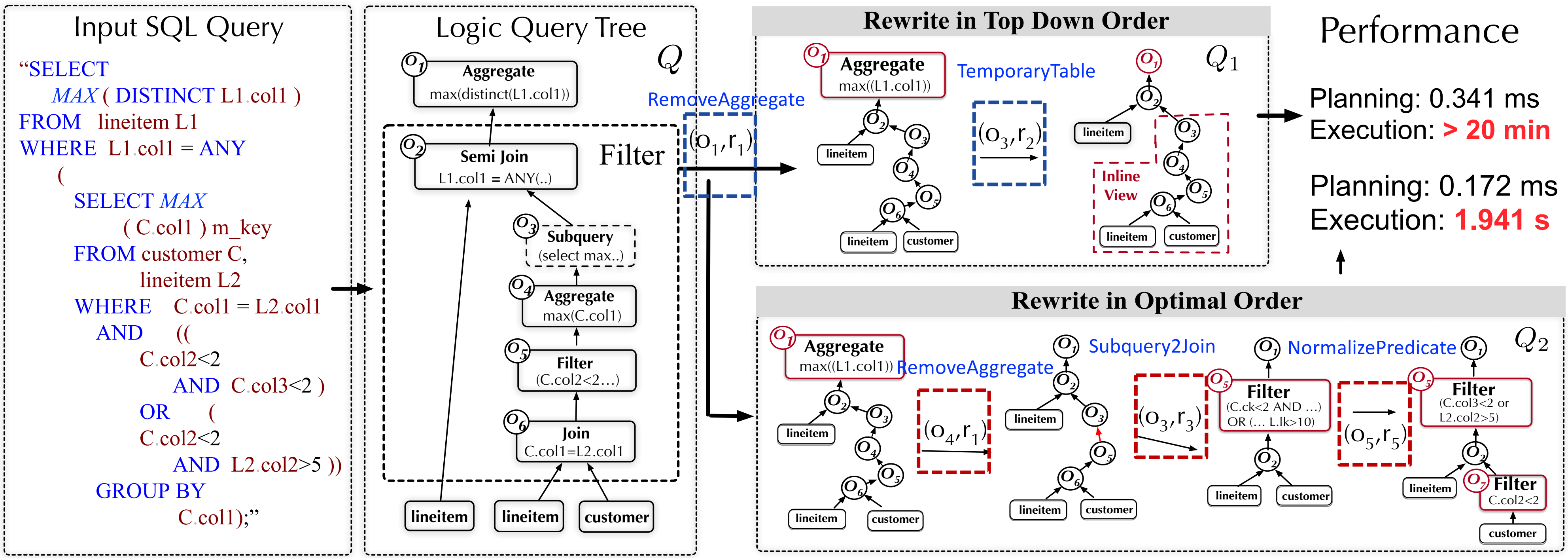
Learning-based Query Rewrite

- **Challenge:**
  - **Equivalence verification for new rules**
  - **Search rewrite space within time constraints**
    - Rewrite within milliseconds;
  - **Estimate rewrite benefits by multiple factors**
    - Reduced costs after rewriting
    - Future cost reduction if further rewriting the query

Automatic Query Rewrite

◻ Problem Definition

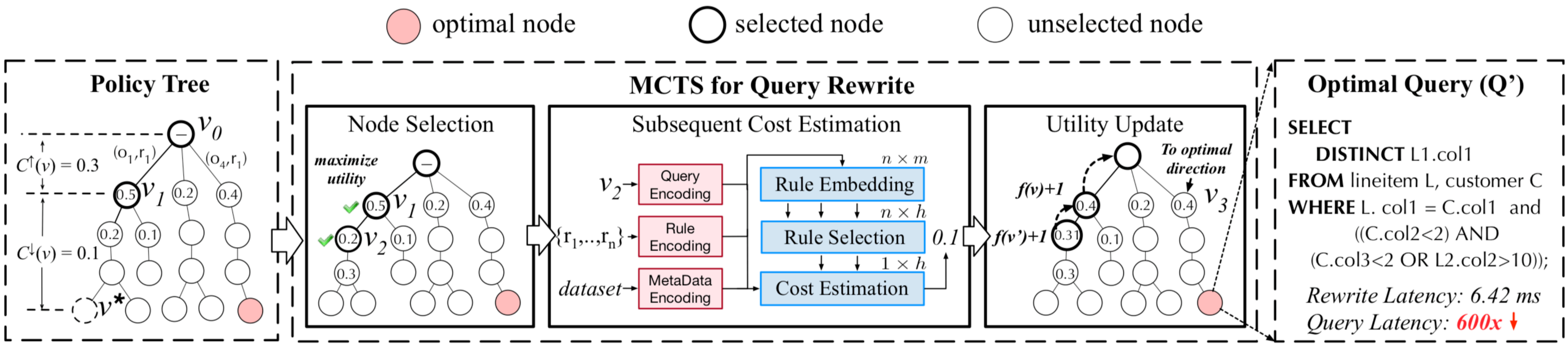
- ◻ Given a slow query Q and a set of rewrite rules R, apply the rules R to the query Q so as to gain (a) the equivalent one and (b) the minimal cost.



Adaptively Apply Rewrite Rules

- **Motivation: A slow query may have various rewrite sequences (different benefits)**
- **Core Idea: Explore optimal rewrite sequences with tree search algorithm**
- **Challenge: (1) How to represent candidate rewrite sequences; (2) How to efficiently find optimal rewrite sequence.**
- **Solution**
  - **Initialize policy tree for a new query**
    - **Node v_i**: any rewritten query; **$C^{\uparrow}(v_i)$**: previous cost reduction; **$C^{\downarrow}(v_i)$**: subsequent cost reduction
  - **Explore rewrite sequences on the policy tree (MCTS)**
    - **Node Value Computation (Node Selection):**

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



Xuanhe Zhou, Guoliang Li, Chengliang Chai. A Learned Query Rewrite System using Monte Carlo Tree Search. VLDB, 2022.

Summarization of Query Rewrite

Methods	Granularity	Equivalence	Supported Rules	Rule Strategy	Rewrite Overhead	Rewrite Performance
WeTune	Logical Plan	√ (within 4 operators)	Generated	heuristic	High for Verify (383*50 ms).	More than 30%~90%
Sia	Predicate	√ (simple queries)	Predicate Rules only	heuristic	High (3s)	More than 2x
Learned Rewrite	Logical Plan	√	Rules from Calcite	MCTS	Medium (6.1-69.8 ms)	More than 2x

Take-aways of Query Rewrite

- Traditional query rewrite method is unaware of cost, 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
  - Further reduce the rewrite overhead
  - Adapt to different rule sets/datasets
  - Design new rewrite rules

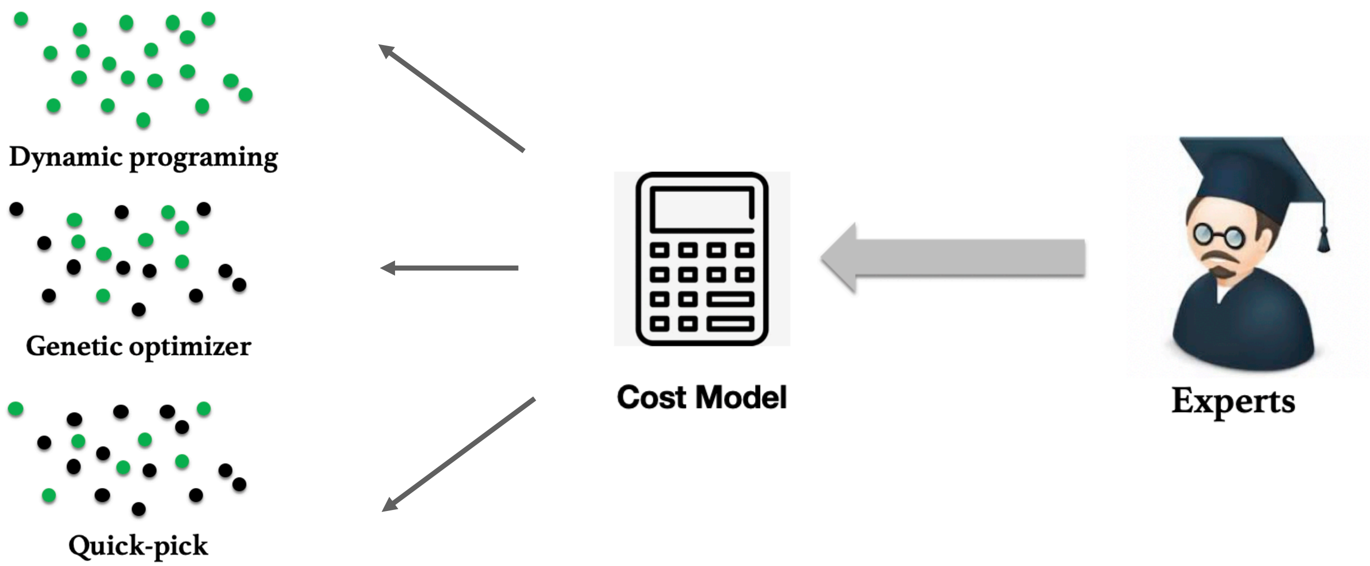
Join Order Selection

- ☐ **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**
  - ☐ **Steer existing optimiers can gain higher performance**
    - ➢ **Hint join orders; Hint operator types**

Join Order Selection

Problem Definition: Given an SQL query, select the "cheapest" join ordering (according to the cost model).

- **Cost, Latency**



Join Order Selection

☐ Method Classification

☐ Offline Optimization Methods.

- Characteristic: given Workload，RL based.

- Key idea: Use existing workload to train a learned optimizer, which predicts the plan for future queries.

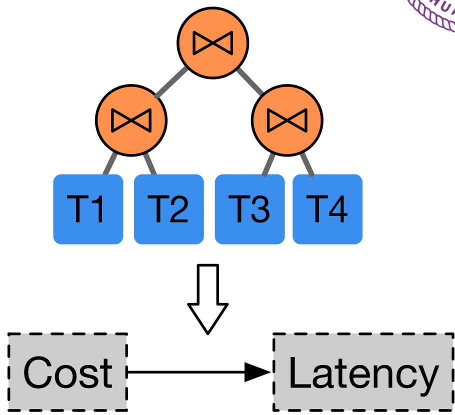
☐ Online Optimization Methods.

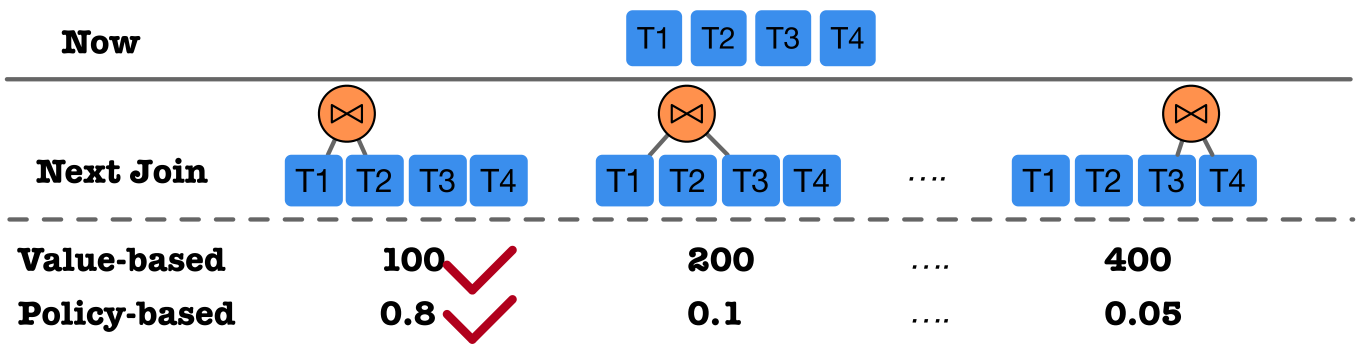
- Characteristic: No workload, but rely on customized Database.

- Key idea: The plan of a query can be changed during execution. The query can switch to another better plan. It learns when the database executes the query.

Why Learned Join Order

- Why learned join order selection?
  - Learned Cost Model
    - Learned from latency when cost estimation is inaccurate.
  - Learned Plan Enumeration
    - not only to estimate the execution time of the complete plan, but also to estimate the generation direction of a good plan
    - guide the direction of plan generation, and reduce the number of enumerated plans.





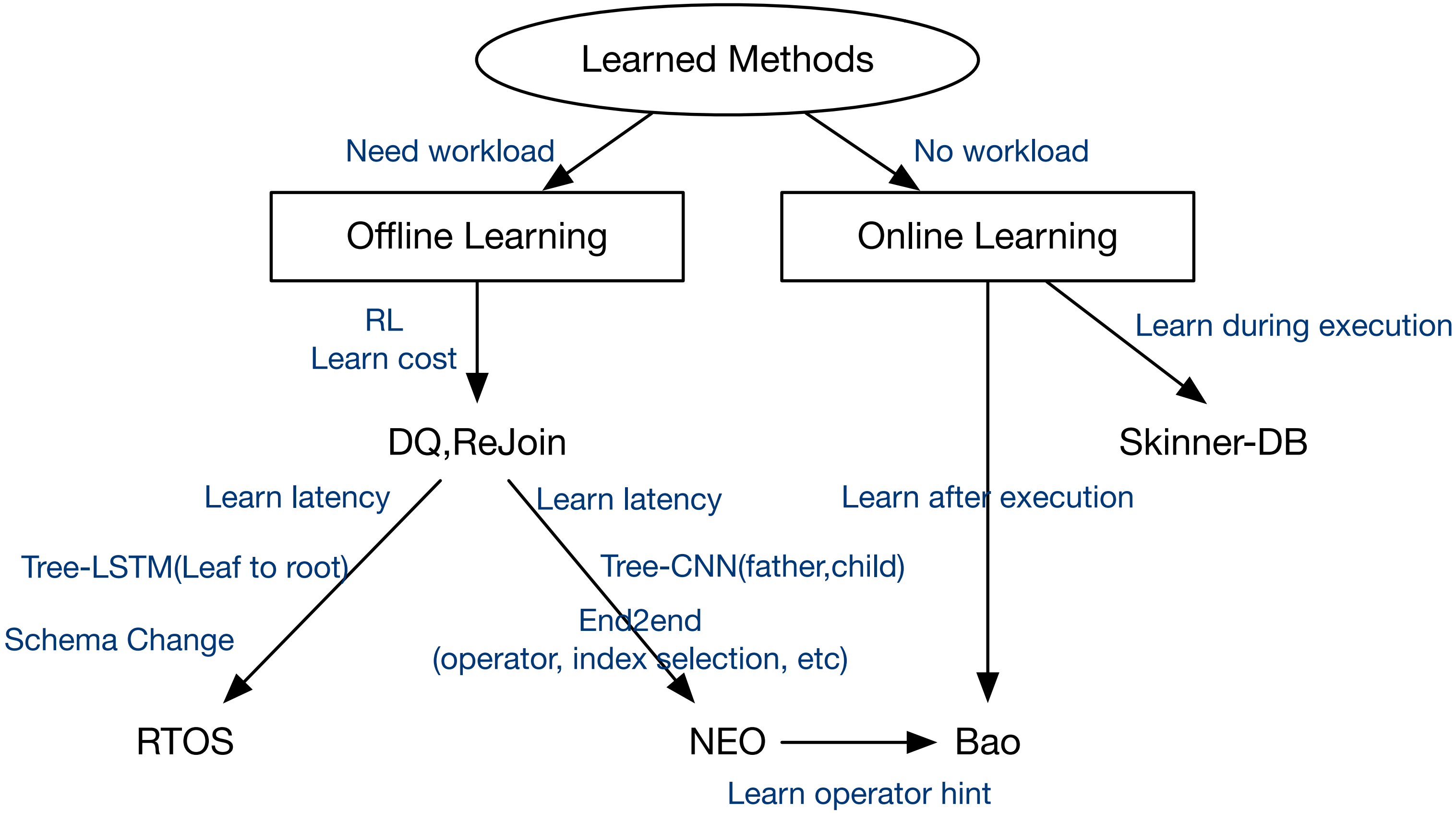
Learned Join Order Selection

- Challenges
  - Learning models need to be able to accurately predict execution times.
  - The latency of plan generation should be low enough.
- Optimization Goals
  - Quality: Latency
  - Adaptivity: Adapt to different DB instances, workloads
  - Update: Join graph, Schema, Data
  - Training Cost

Learned Join Order Selection

- Method Classification
  - Offline learning methods
    - Characteristic : Learn before use - given workload
    - Key idea : Use existing workload to train a learned optimizer, which will predict the plan for future workload.
  - Online learning methods
    - Characteristic: Learn runtime - no workload
    - Key idea : The model can quickly learn from the execution feedback during or after query execution to improve the next plan generation.
  - Key difference: Online learning methods can handle update easily and the performance will not be limited by the given training data.

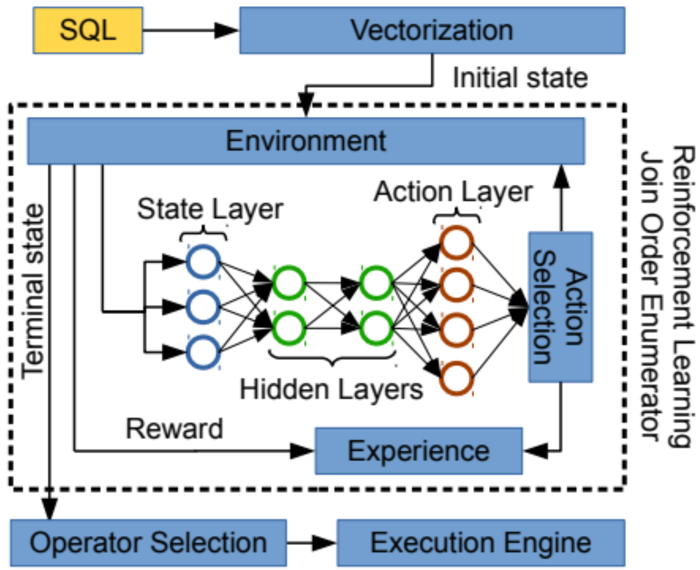
Learned Join Order Selection



1 Offline Join Order Selection: ReJoin & DQ

Motivation

- The search space for join order is huge.
- Traditional optimizer did not learn from pre bad or good choice.



Challenges

- How to reduce the search space of join orc
- How to select the best join order.

Difference：

- ReJoin uses a policy based method (PPO) to guide the plan search.
- DQ uses a value based method (DQN) to guide the plan search. It uses the DP's plan to pretrain the value neural network.

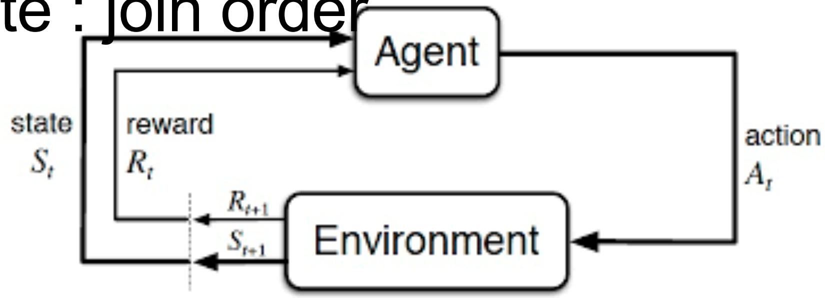
Marcus, Ryan, and Olga Papaemmanouil. "Deep reinforcement learning for join order enumeration." aiDM 2018

Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning, arXiv 2018

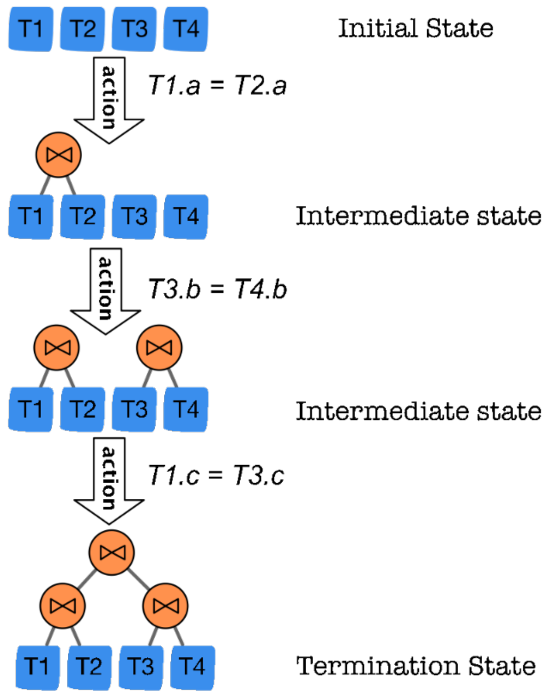
1 Offline Join Order Selection: ReJoin & DQ

☐Map into RL Models (DQ, ReJOIN) [1,2]

- ➢ Agent : optimizer
- ➢ Action: join
- ➢ Environment: Cost model, database
- ➢ Reward：Cost, Latency
- ➢ State : join order



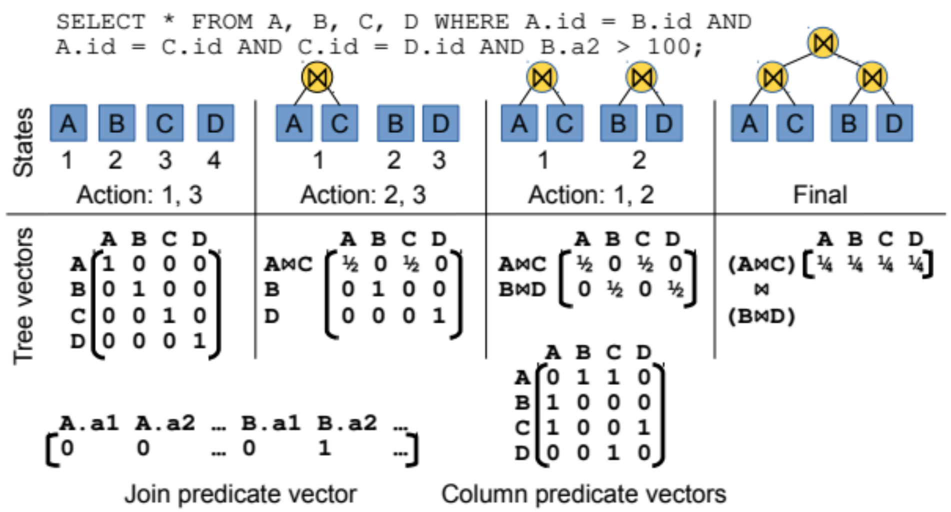
Select *
From T1,T2,T3,T4
Where T1.a = T2.a
and T3.b = T4.b
and T1.c = T3.c



Marcus, Ryan, and Olga Papaemmanouil. “Deep reinforcement learning for join order enumeration.” ,aiDM 2018
Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning, arXiv 2018

1 Offline Learned Join Order Selection: ReJoin

- **RL model**
  - **Agent : optimizer;**
  - **Action: join;**
  - **Environment: Cost model, database**
  - **Reward：Cost ;**
  - **State : join order**
  - **Long-term reward:**
    - **Policy-based : Output all-join probability**
    - **Neural network : A three-layer MLP.**

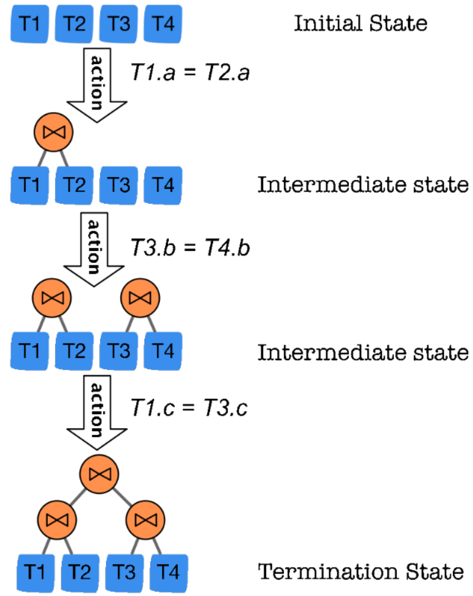


Marcus, Ryan, and Olga Papaemmanouil. "Deep reinforcement learning for join order enumeration." ,aiDM 2018

1 Offline Learned Join Order Selection: DQ

- RL model
  - Agent : optimizer
  - Action: join
  - Environment: Cost model, database
  - Reward：Cost
  - State : join order
  - Long term reward:
    - Value based : Predict the possible cost of the

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




```
SELECT *
  FROM Emp, Pos, Sal
 WHERE Emp.rank
       = Pos.rank
   AND Pos.code
       = Sal.code
```

(a) Example query

A_G = [E.id, E.name, E.rank, P.rank, P.title, P.code, S.code, S.amount]
= [1 1 1 1 1 1 1 1]

(b) Query graph featurization

A_L = [E.id, E.name, E.rank]
= [1 1 1 0 0 0 0 0]
A_R = [P.rank, P.title, P.code]
= [0 0 0 1 1 1 0 0]

(c) Features of $E \bowtie P$

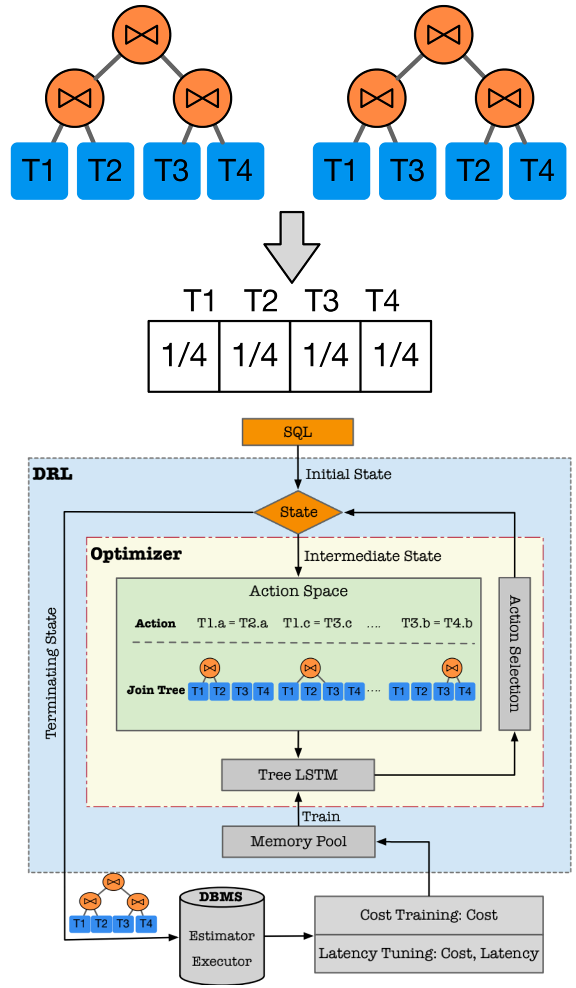
A_L = [E.id, E.name, E.rank, P.rank, P.title, P.code]
= [1 1 1 1 1 1 0 0]
A_R = [S.code, S.amount]
= [0 0 0 0 0 0 1 1]

(d) Features of $(E \bowtie P) \bowtie S$

Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning, arXiv 2018

2 Offline Learned Join Order Selection: RTOS

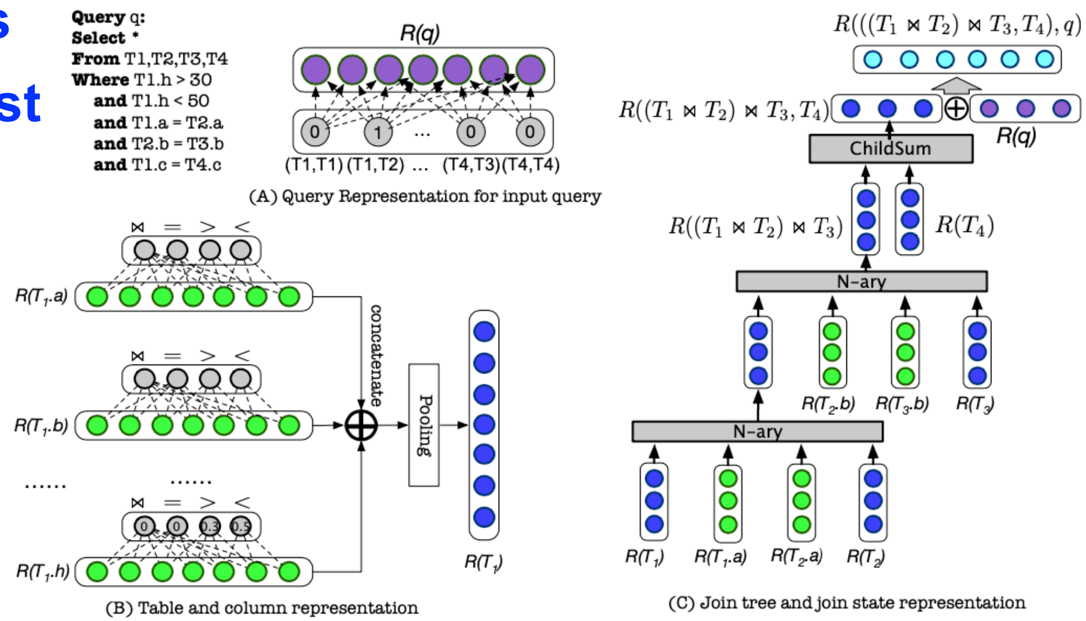
- **Motivation**
  - **Previous learning based optimizers give good cost, but they do not give good latency on test queries.**
  - **Schema often changes in real-world database.**
- **Challenges**
  - **The intermediate state is a forest, which cannot be represented by a simple feature vector.**
  - **The training time is huge when collecting latency as feedback.**
  - **The schema change leads to the retraining.**



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

2 Offline Learned Join Order Selection: RTOS

- TreeLSTM based Q network
  - Use n-ary to represent the sub-trees
  - Use child-sum to represent the forest
- Two step training
  - Cost pretrain
  - Latency fine-tuning
- Dynamic neural network
  - DFS to build neural network
  - Multi-Alias: Parameter sharing
  - Schema change: Local fine-tuning



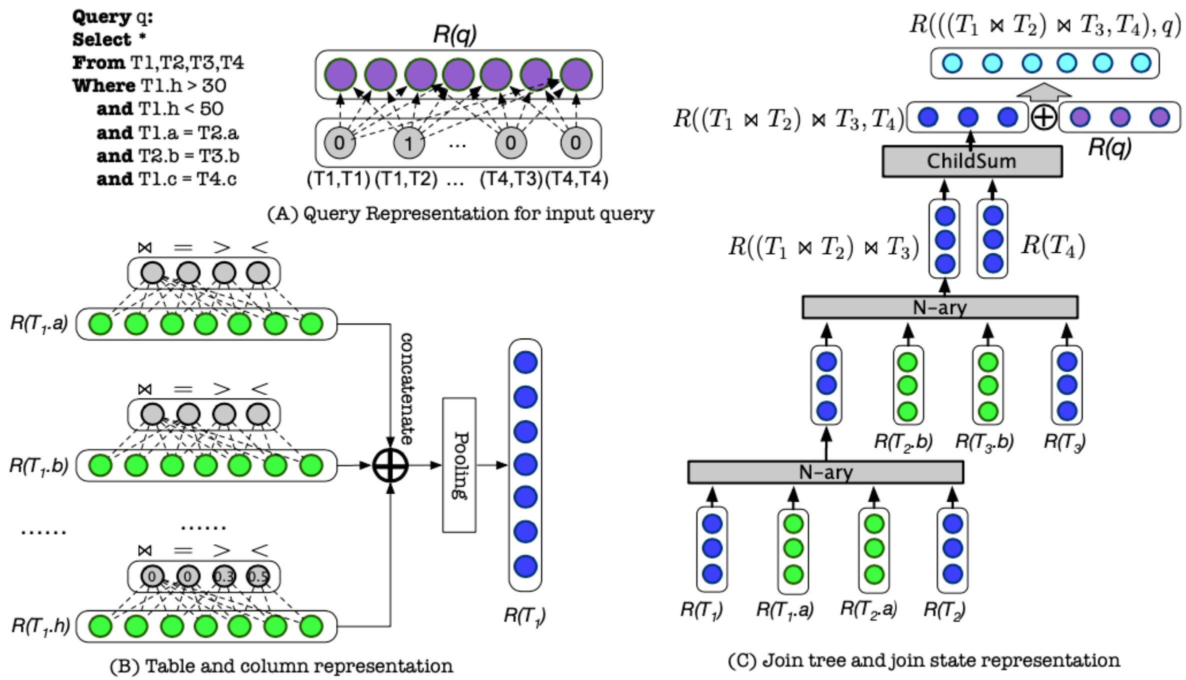
Yu X, Li G, Chai C, et al. Reinforcement learning with tree-lstm for join order selection. ICDE 2020

2 Offline Learned Join Order Selection: RTOS

☐ Feature Extraction

- **The structural information of the execution plan is vital to join order selection →**

- **Encode the operator relations and metadata features of the query**
- **Embed the query features with Tree-LSTM;**
- **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.

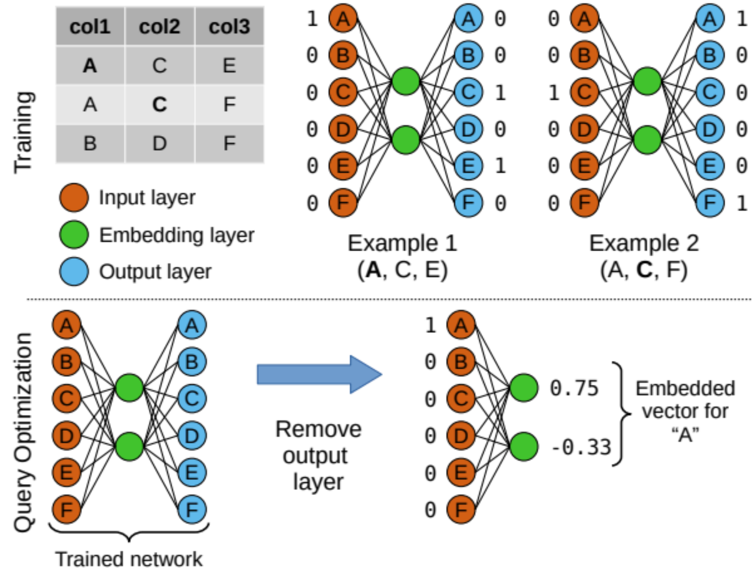
3 Offline Learned Join Order Selection: NEO

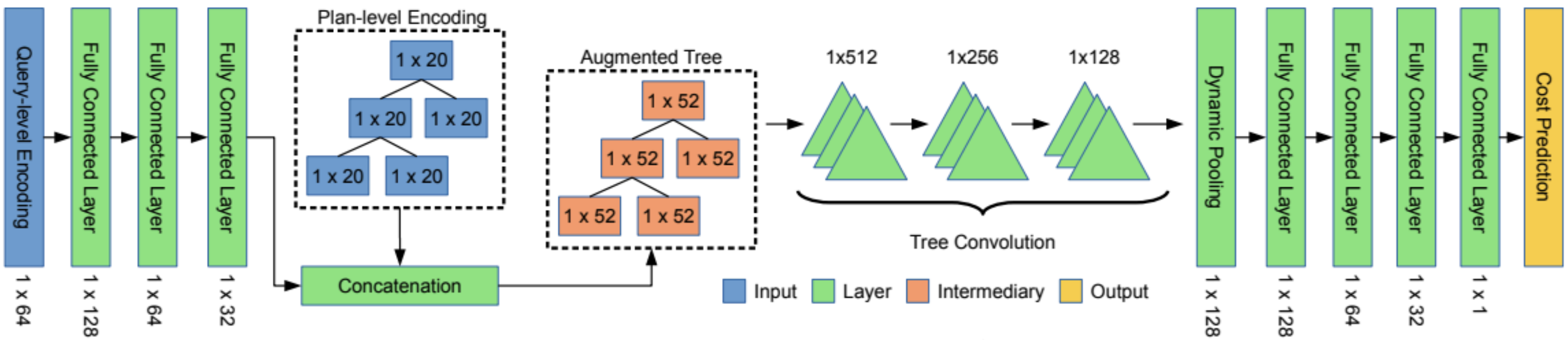
- **Motivation**
  - **Previous traditional optimizer relies on cost models**
  - **Previous methods solve join ordering only but cannot support physical operator selection.**
- **Challenges**
  - **How to build a learn cost model automatically to capture intuitive patterns in tree-structured query plans and predict the latency.**
  - **How to represent query predicate semantics (supporting strings – word2vector) automatically.**
  - **How to overcome reinforcement learning's sample inefficiency (with optimizer guide)**

Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

3 Offline Learned Join Order Selection: NEO

- It uses Tree-CNN to design a value network to represent the query plan (join order, operator).
- It uses row vectors to represent predicates. Each row is a sentence.
- It learns from the expert optimizer learning from demonstration.
- Normalize plan's cost by cost of optimizer's plan





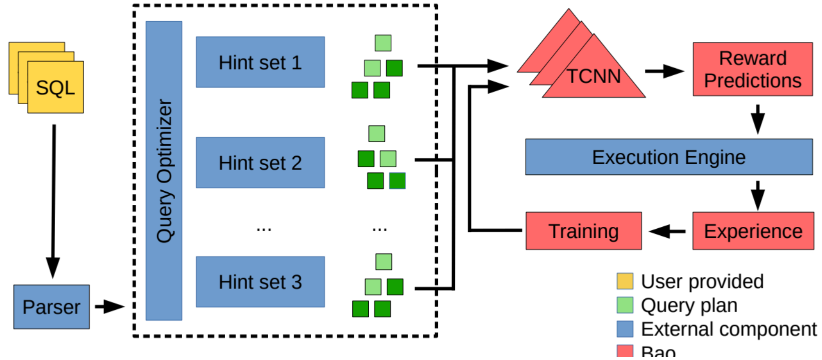
4 Online Learned Join Order Selection: Bao

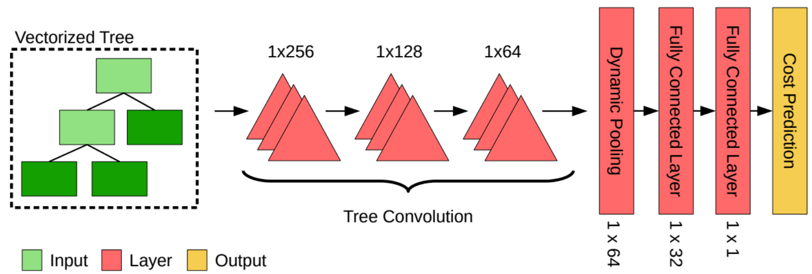
- Motivation
  - Long training time
  - Cannot adjust to data and workload changes
  - Tail latency of worse plans
  - The choice of physical operator affects the quality of the plan
- Challenges
  - How to enumerate the plan?
  - How to study plan latency and choose a high-quality plan?

Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

4 Online Learned Join Order Selection: Bao

- Use operator hint to generate candidate plans.
  - Enable/disable hash join,…
- Use Tree-CNN to predict the latency and guide the plan selection.
  - Latency prediction
    - Encode each plan into a vectorized tree.
  - Contextual multi-armed bandits.
    - Each hint set is an arm
    - Use Thompson sampling to update the model parameter.

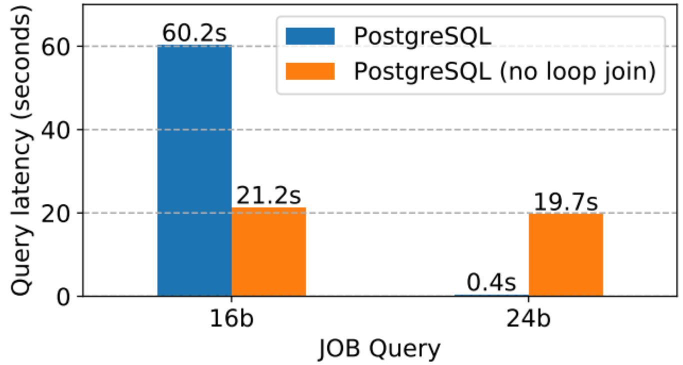




Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

4 Online Learned Join Order Selection: Bao

- **Enhance query optimization with minor changes**
  - E.g., Activate/Deactivate loop join for different queries

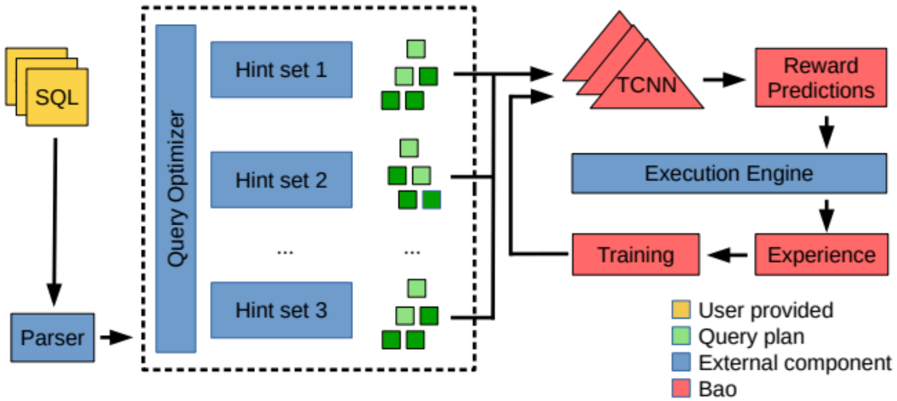


- **Model Plan Hinter as a Multi-armed Bandit Problem**
  - Model each hint set $HSet_i$ as a query optimizer

$$HSet_i : Q \rightarrow T$$

  - 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.

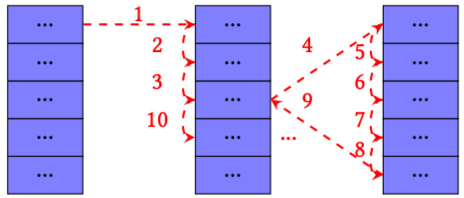
5 Online Learned Join Order Selection: SkinnerDB

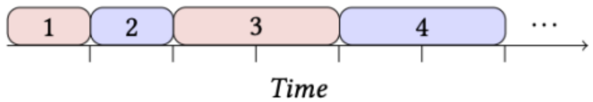
- Motivation
  - Previous works relied on learning from cost models or expert optimizers.
  - Previous learning based optimizers need to give training queries and are hard to provide good plans to different workload.
  - The executor can detect estimation errors during query execution.
- Challenges
  - How to design a new working mechanism that allows the optimizer to learn and switch between different join orders online.
  - How to evaluate and choose different join orders online.

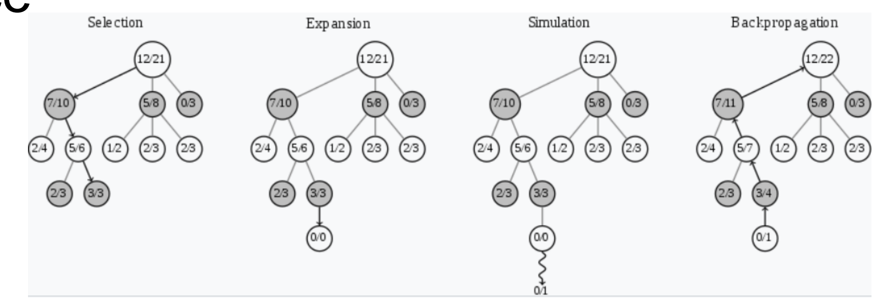
Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

5 Online Learned Join Order Selection: SkinnerDB

- Eddies-style
  - Divide the execution process into several time slices.
  - N way join can support the plan switch.
  - Select the plan for the next time slice based on the previous time slice
- MCTS For JOS
  - Learn and generate a plan in each time slice
- Rely on Customize Database
  - Switch plan in low latency



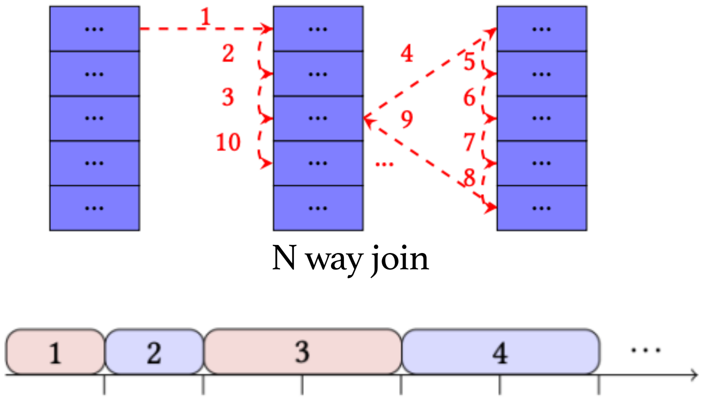


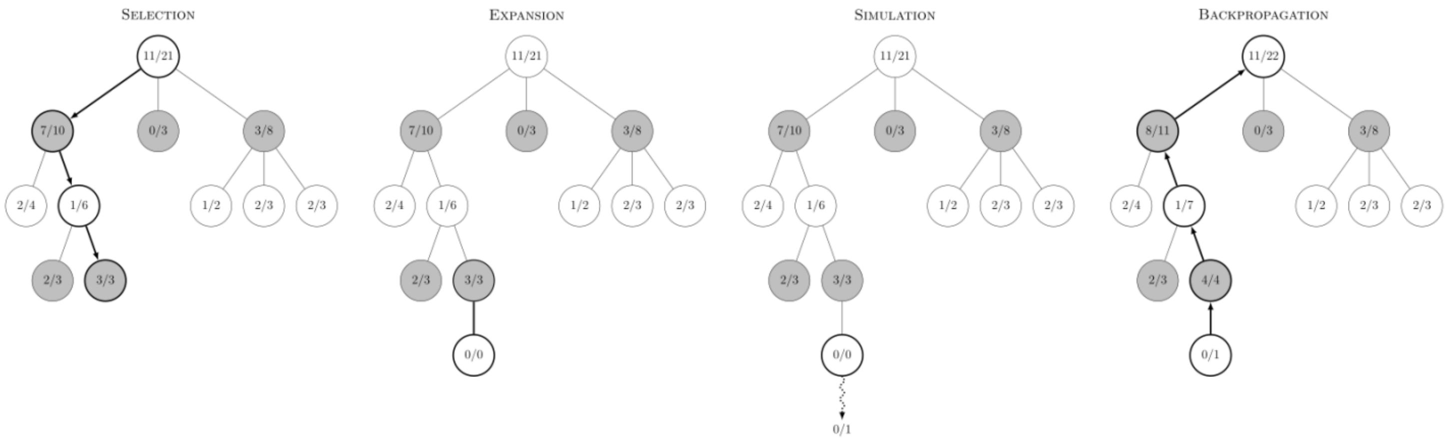


Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

5 Online Learned Join Order Selection: SkinnerDB

- **Support online reorder with MCTS →**
  - Do not require pre-training
  - Time Slides: 0.001s
    - Learn during runtime
  - Customize Database
    - Switch Plan in Low Latency





Monte Carlo tree search (MCTS).

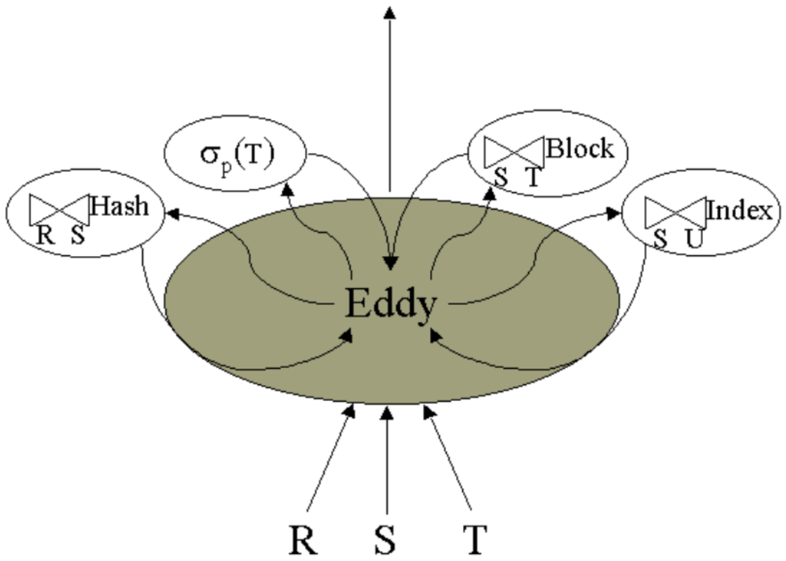
Trummer, et al Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In SIGMOD, 2019.

5 Online Learned Join Order Selection: SkinnerDB

Update execution orders of tuples on the fly

– **Update the plan 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 the tuples have been handled by all the operators.**



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

Learned Join Order Selection

	Quality	Training Cost	Adaptive (workload)	Adaptive (DB Instance)	Learned Operator	Methods
Traditional [Genetic algorithms] [Dynamic Programming]	Low	Low	High	High	✓	Cost model
DQ	Medium	High	Low	High	×	Value-based DRL
ReJoin	Medium	High	Low	High	×	Policy-based DRL
RTOS	High	High	Medium	High	×	Value-based DRL, Tree-LSTM
NEO	High	High	Low	High	✓	Value-based DRL, Tree-CNN
Bao	High-	Medium	High	High	✓	CMAB, Thompson sampling, Value-based, Tree-CNN
Skinner-DB	High	Low	High	Low	×	Eddies-style, Value-based, MCTS

Learned Join Order Selection: Take-away

- Not easy to be applied in real DBMS
- Open problems
  - Low latency plan generation
    - Neural networks bring delays that cannot be ignored. How to apply learning algorithms to low-latency OLTP services.
  - Support complex queries
    - Nested queries.
  - Learning metrics
    - The planned latency will vary with the system state and network delay.
    - Some faster plans may consume more resources. For example, use two-core CPU in parallel to reduce the execution time by 20%.

Autonomous Database Systems

Motivation

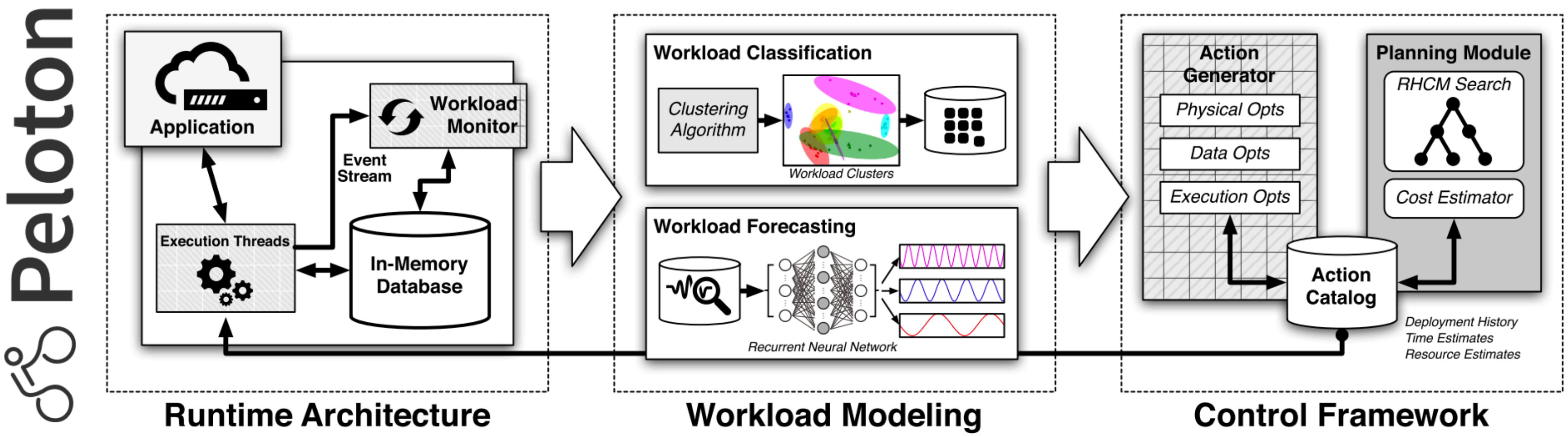
- **Traditional Database Design is laborious**
  - Develop databases based on workload/data features
  - Some general modules may not work well in all the cases

- **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

- **Schedule optimization actions via workload forecasting**
  - **Embedded Monitor:** Detect the event stream
  - **Workload Forecast Model:** Future workload type
  - **Optimization Actions:** Tuning, Planning



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

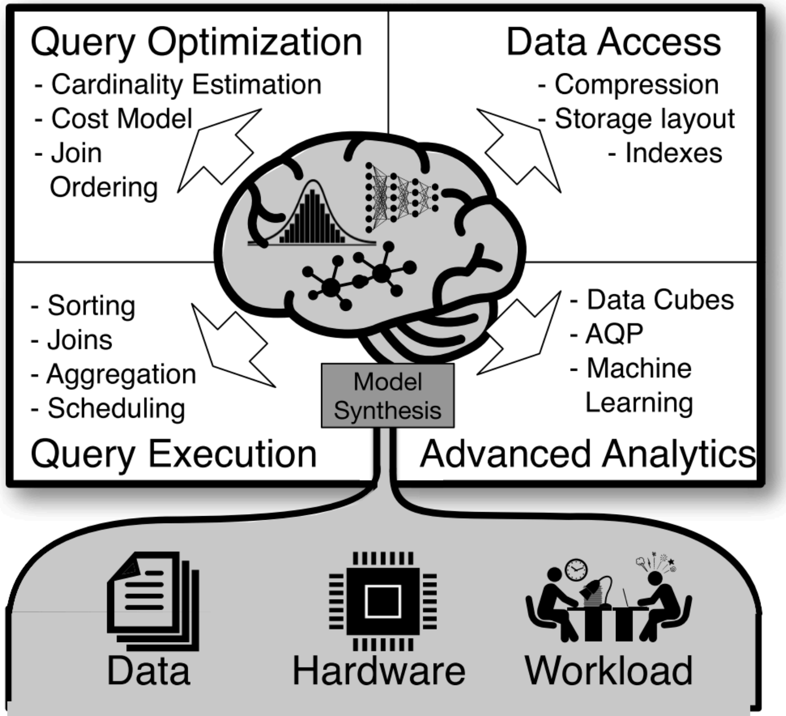
SageDB

□ Customize DB design via learning the Data Distribution

- ➢ Learn Data Distribution by Learned CDF

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

- ➢ Design Components based on the learned CDFs
  - ➢ Query optimization and execution
  - ➢ Data layout design
  - ➢ Advanced analytics

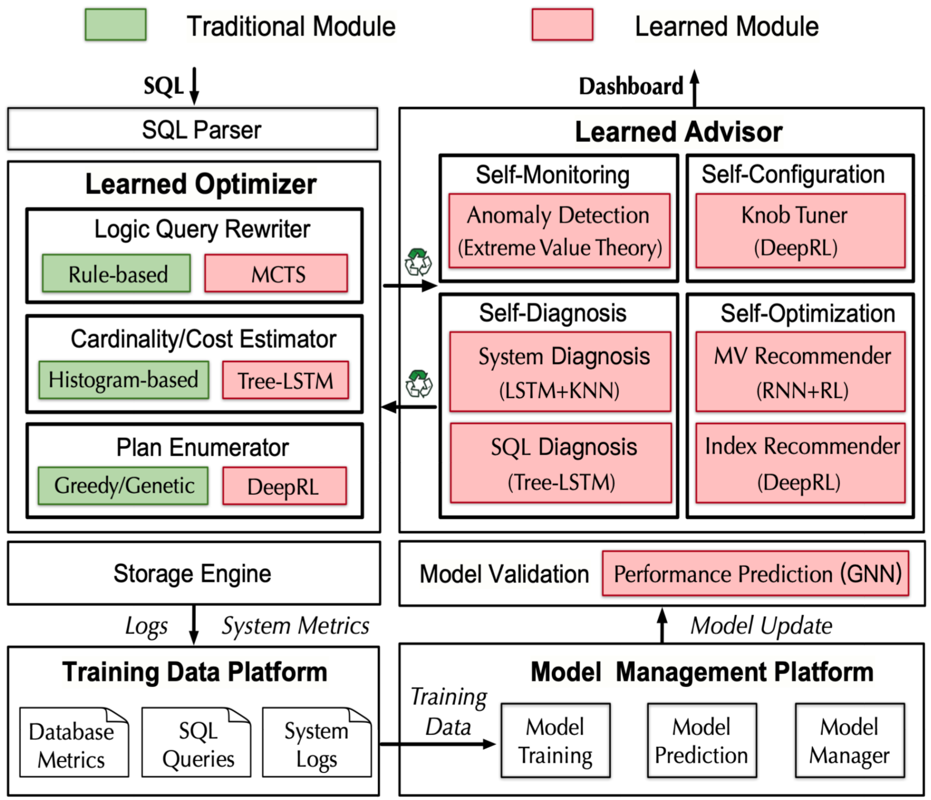


Tim Kraska, et al. SageDB: A Learned Database System. In CIDR, 2019.

openGauss

□ Implement, validate, and manage learning-based modules

- ➢ **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.

Learned Advisor

Learned Advisors

- Learned Knob Tuning
- Learned Index Advisor
- Learned View Advisor
- Learned Partition Advisor
- Learned Data Generation

Knob Tuning

□ A Constrained Optimization Problem

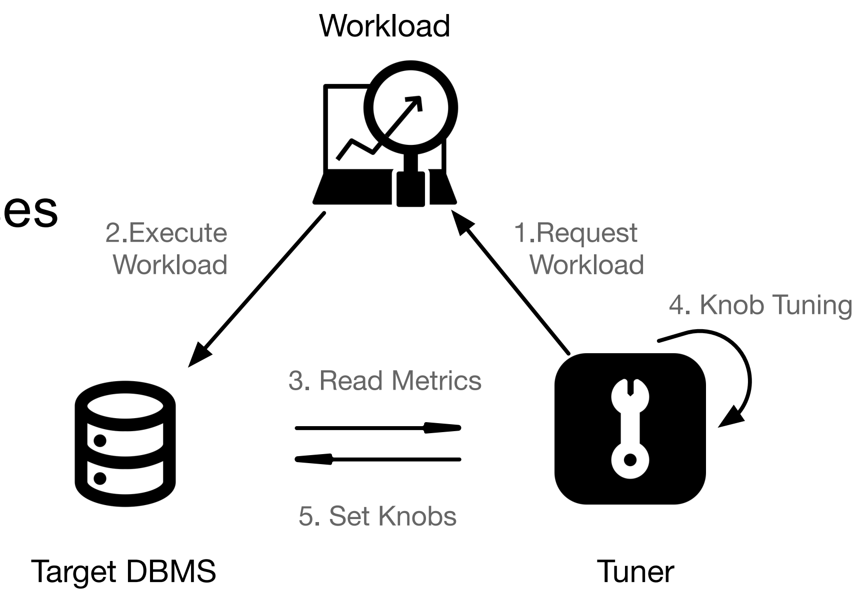
- Given a suite of knobs B and a target T, knob tuning aims to find the optimal values of B, so as to meet T for the incoming workload.

□ Knobs

- concurrency control, optimizer settings
- memory management, background processes

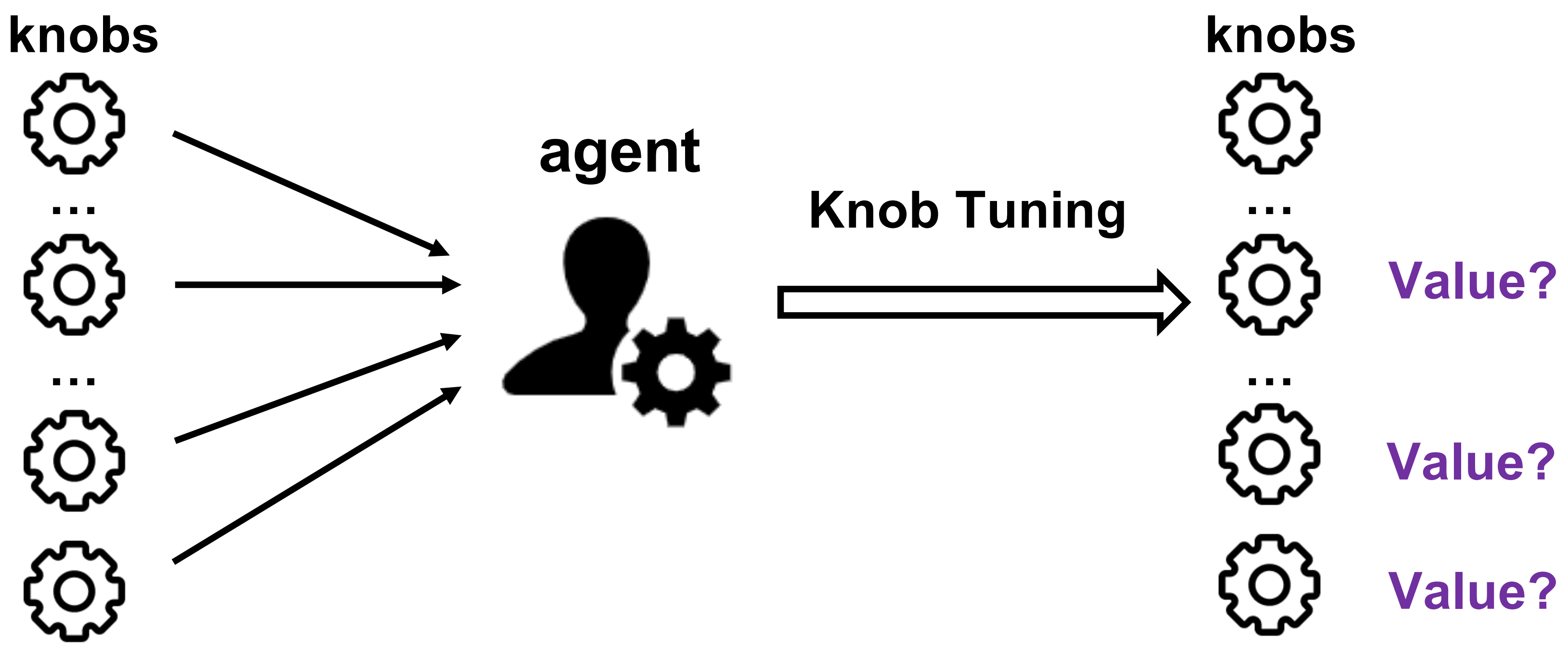
□ Targets

- Performance (throughput, latency)
- Resource Usage (e.g., CPU utilization)



Offline Optimization for Knob Tuning

Problem Definition: Consider a database with different workloads, the target is to find the optimal knob settings to meet required SLA (service-level agreement).

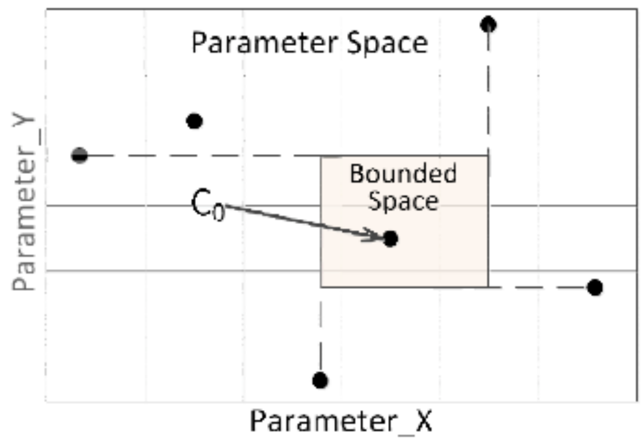


Offline Optimization for Knob Tuning

- **Motivation:**
  - DBMSs have different optimal knob settings, which significantly affect the query performance and resource utilization.
  - DBMSs have numerous runtime metrics. Classic ML models cannot efficiently select knobs based on the metrics.
  - DBMSs have numerous system knobs with continuous values, which makes it harder to find optimal knobs.

Traditional Knob Tuning Methods

- **Motivation: Most users only utilize default knob settings and cause performance regression**
- **Basic Idea: Greedily select local-optimal knob settings with bound-and-search algorithm**
- **Challenge: Optimal settings change with tuning goals and workloads**
- **Solutions:**
  - **Sample Phase:** Divide each knob range into k intervals and sample k settings that cover all the value ranges
  - **Search Phase:** Select the best sampled setting and build search space around the best setting



Random Sampling: Some important settings may not be sampled

Yuqing Zhu et al. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In SoCC, 2017.

Learning-based Knob Tuning

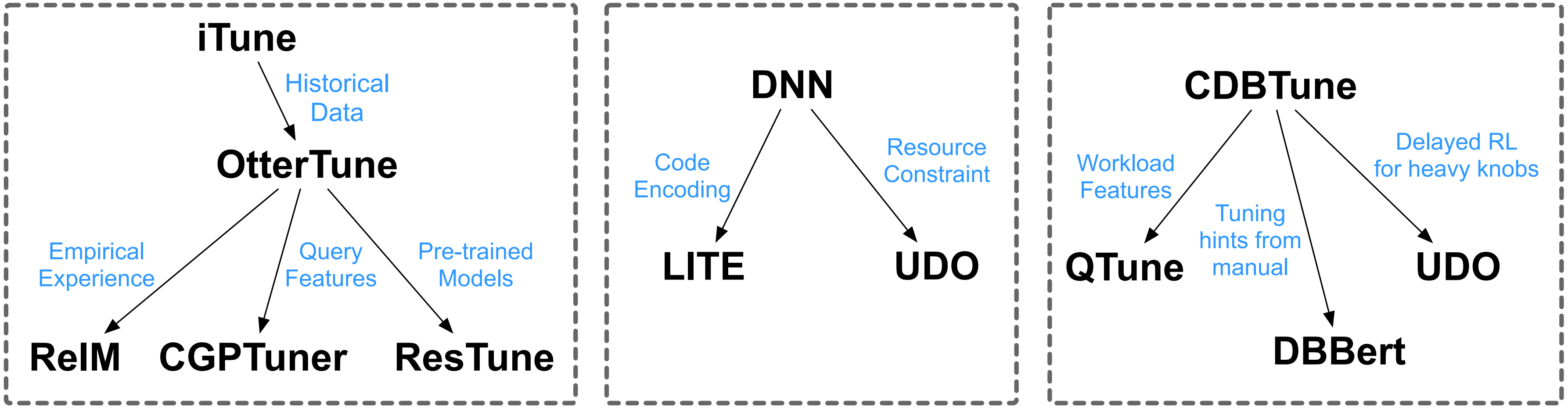
- **Why heuristics → Machine Learning ?**
  - **A large number of configuration knobs**
    - Total > 400
    - *Heuristic Method:* waste much time in search from huge knob space
  - **Knobs control nearly every aspect and have complex correlations**
    - One-knob-at-a-time is inefficient
    - *Heuristic:* The relations are non-monotonic
  - **Learn from the historical tuning**
    - *Heuristic:* Restart tuning from scratch each time

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:

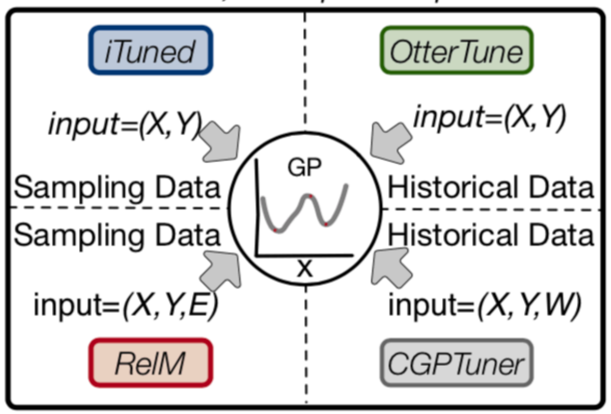
default_statistics_target = 50
maintenance_work_mem = 1GB
constraint_exclusion = on
checkpoint_completion_target = 0.9
effective_cache_size = 22GB
work_mem = 192MB
wal_buffers = 8MB
checkpoint_segments = 16
shared_buffers = 7680MB
max_connections = 80

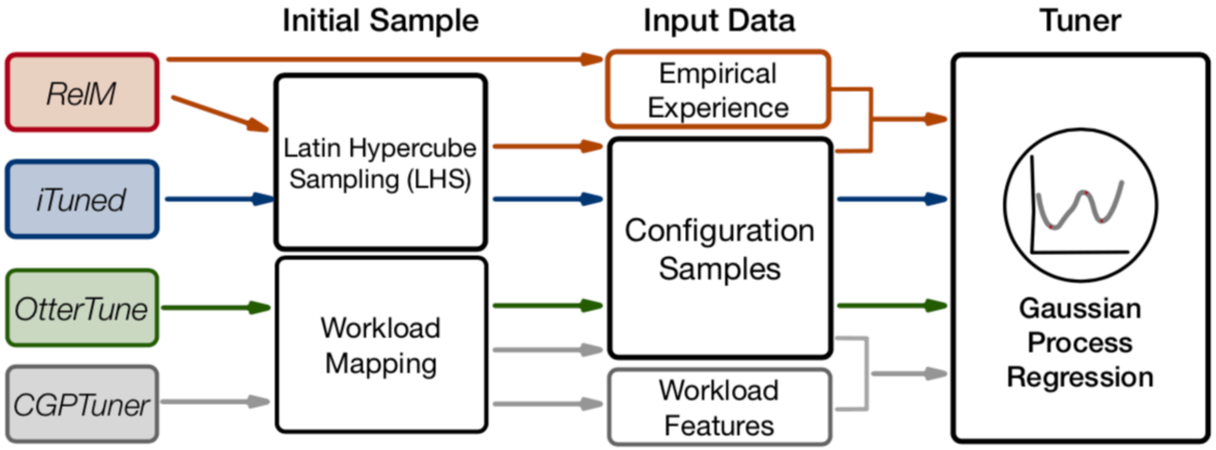
Learning-based Knob Tuning

Bayesian Optimization *Deep Learning* *Reinforcement Learning*



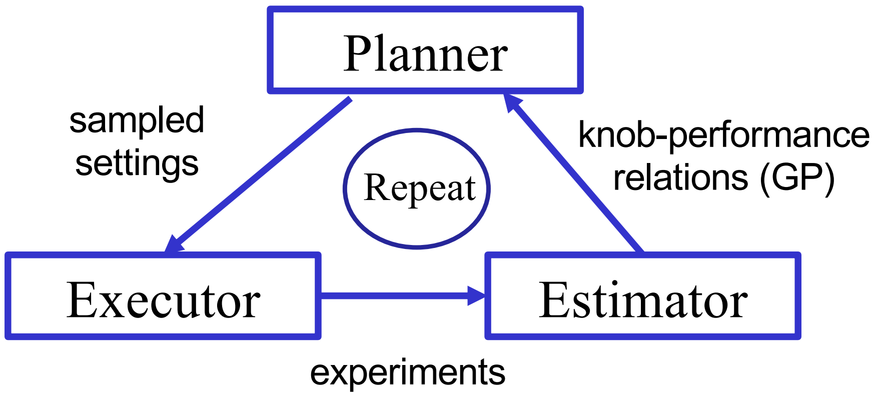
X: Configuration, Y: Performance;
W: Workload; E: Empirical Experience

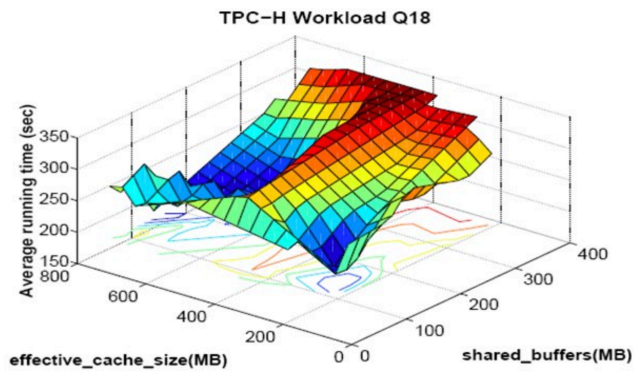




(1.1) Bayesian Optimization for Knob Tuning

- **Motivation: Only a few knobs have significant effects to the performance**
- **Basic Idea: Explore the knob-performance relations by experiments**
- **Challenge: Identify important knobs and their values efficiently**
- **Solution:**
  - **Planner:** Adaptively sample some knob settings
  - **Executor:** Get the performance of sampled settings by running workloads
  - **Estimator:** Predict knob-performance relations with Gaussian Process
  - **Termination:** Terminate if arriving time limit; otherwise repeat above steps



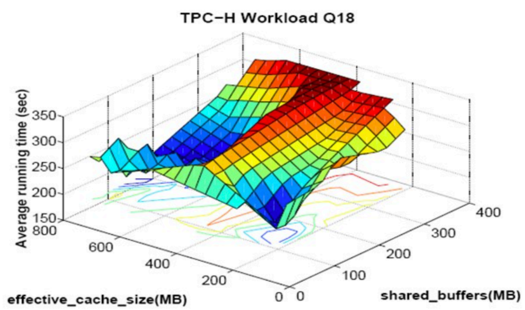


knob-performance relations

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

(1.1) Bayesian Optimization for Knob Tuning

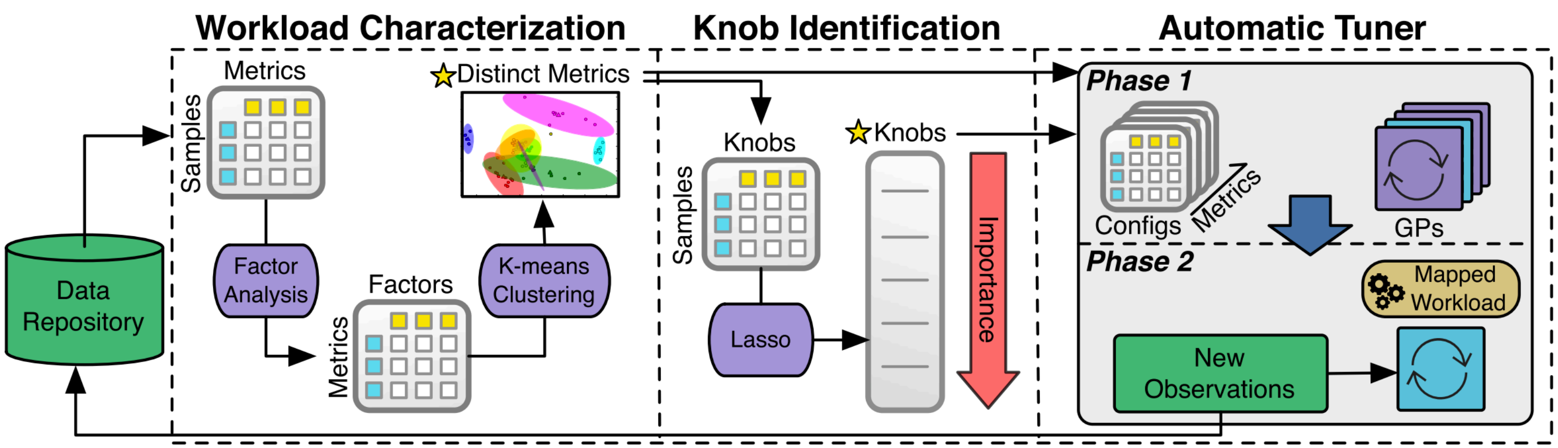
- **Motivation: Only a few knobs have significant effects to the performance**
- **Basic Idea: Explore the knob-performance relations by experiments**
- **Challenge: Identify important knobs and their values within hours**
- **Solution:**
  - **Planner:** Adaptively sample some knob settings
  - **Executor:** Get the performance of sampled settings by running workloads
  - **Estimator:** Predict knob-performance relations with Gaussian Process
  - **Termination:** Terminate if arriving time limit; otherwise repeat above steps
- **Limitations**
  - **Sampling configurations from scratch is inefficient**
  - **Knob-performance relations are extremely complex**
  - **Important workload features are not utilized**



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

(1.2) Bayesian Optimization + Historical Data

- **Data-driven: Optimize tuning performance 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.

(1.3) Bayesian Optimization + Empirical Experience

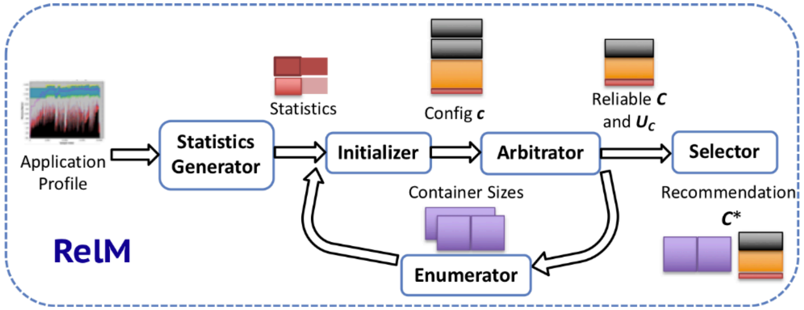
- **Motivation: Expert experience can make learned tuning more robust**
  - e.g., limit the minimal shard buffer size
- **Basic Idea: Utilize expert experience to optimize tuning**
- **Solution**
  - **Empirically compute input features at resource/APP/VM levels**

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

    $\boldsymbol{x}$: Tested knob setting
    $\boldsymbol{M_i}$: Code overhead value
    $\boldsymbol{m_c}$: Required cache storage
    $\boldsymbol{m_o}$: GC settings

  - **Rely on empirical features to estimate tuning performance**

    (1) Input: Empirical features,
    Initialized knob values;
    (2) Model: Gaussian Process;
    (3) Target: Tuning Performance.



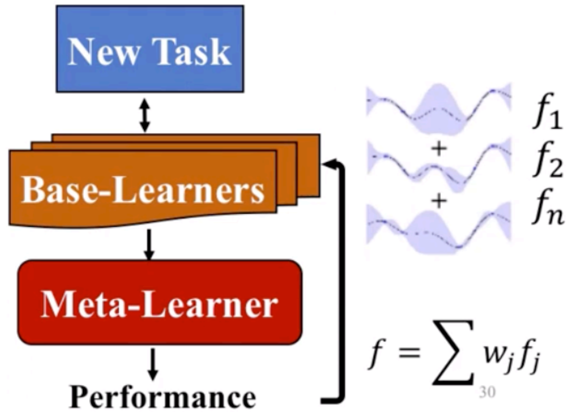
Mayuresh Kunjir, Shivnath Babu. Black or White? How to Develop an AutoTuner for Memory-based Analytics. SIGMOD 2020.

(1.4) Bayesian Optimization + Pretrained Models

- **Motivation: Learning-based tuning is hard to migrate to new scenarios**
- **Basic Idea: Improve migration capability with pre-trained tuning models**
- **Solution:**
  - Characterize the common workload features
    - Reserved SQL words (e.g., SELECT, DISTINCT)
  - Cluster tuning models on historical workloads to generate Base Leaners;
  - For a New Task, generate *Meta Learner* based on the Base Leaners (similarity weight: g_i);
    - The Meta Learner *M* is a gaussian process model:

mean value $\mu_M(\theta) = \frac{\sum_{i=1}^{T+1} g_i \mu_i(\theta)}{\sum_{i=1}^{T+1} g_i}$ *variance* $\sigma_M^2(\theta) = \sum_{i=1}^{T+1} v_i \sigma_i^2(\theta),$

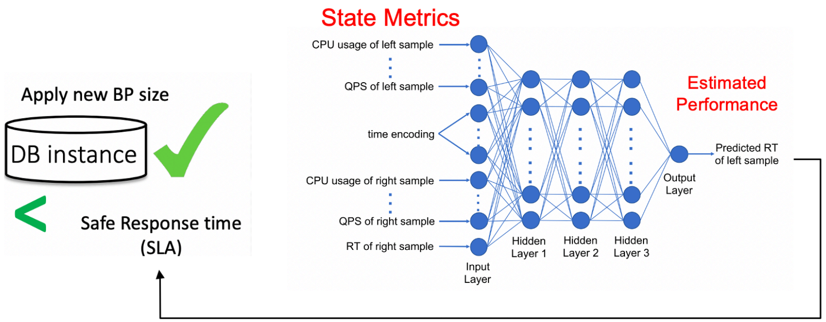
  - Fine-tune the *Meta Learner* by running the new workload;
  - Recommend promising knobs with Meta Learner.



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

(2.1) Deep Learning for Knob Tuning

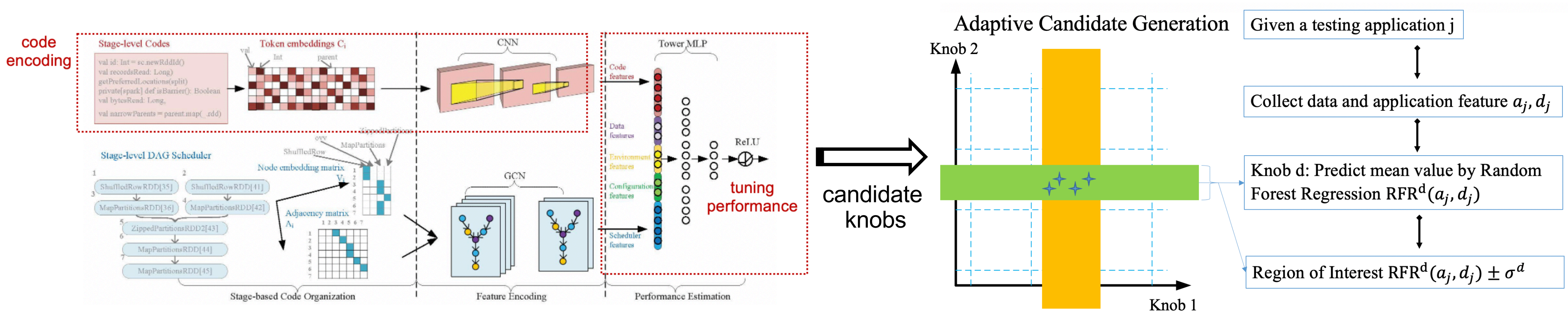
- **Motivation: Expensive to run workloads for evaluating tuning effects**
- **Basic Idea: Estimate tuning effects without running workloads**
- **Challenge: Many metrics affect the performance**
- **Solution:**
  - Collect DB metrics: [logical-read, QPS, CPU usage, response time];
  - Initialize a buffer 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.

(2.2) Deep Learning + Code Encoding

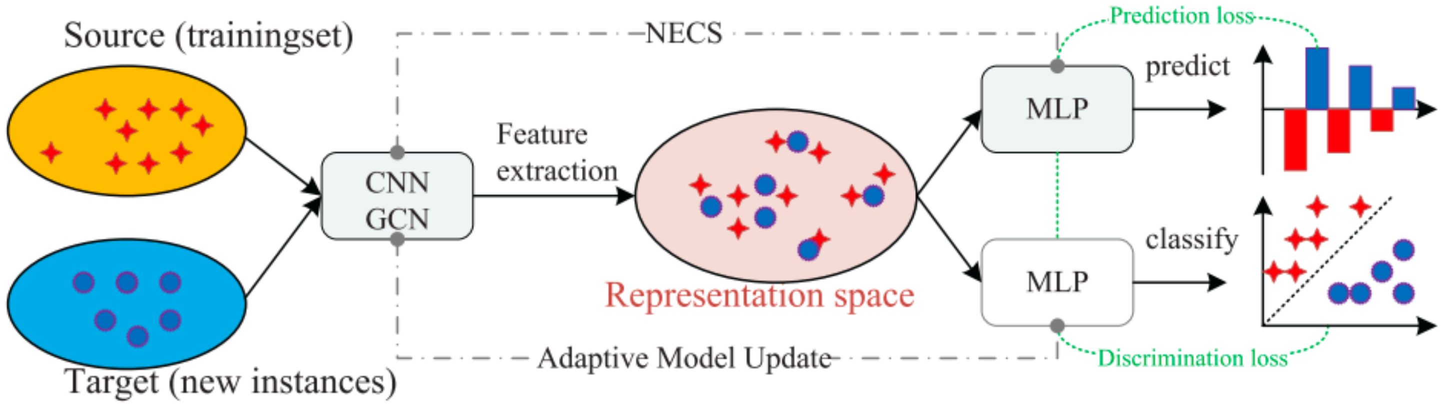
- **Motivation: Spark code involves complex semantics, and it is costly to migrate tuning models from small datasets to large datasets**
- **Basic Idea: Restrict the tuning region by predicting the performance**
  - **Knob Sampling:** Sample candidate knob settings based on the data and code features;
  - **Code Instrumentation:** Enrich semantic features by adding the Spark API;
  - **Performance Prediction:** Predict the performance with encoded code, data, knob, DAG.



Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, Guoliang Li. Adaptive code learning for Spark configuration tuning. ICDE, 2022.

(2.2) Deep Learning + Code Encoding

- **Motivation: Spark code involves complex semantics, and it is costly to migrate tuning models from small datasets to large datasets**
- **Basic Idea: Restrict the tuning region by predicting the performance**
  - **Knob Sampling:** Sample candidate knob settings based on the data and code features;
  - **Code Instrumentation:** Enrich the code features by adding the Spark API tokens;
  - **Performance Prediction:** Predict the performance with *encoded code*, *data*, *knob*, DAG features;
  - **Generalization to Big Datasets:** When dataset changes, utilize adversarial learning to capture the domain-invariant features and update the performance model with newly collected samples.

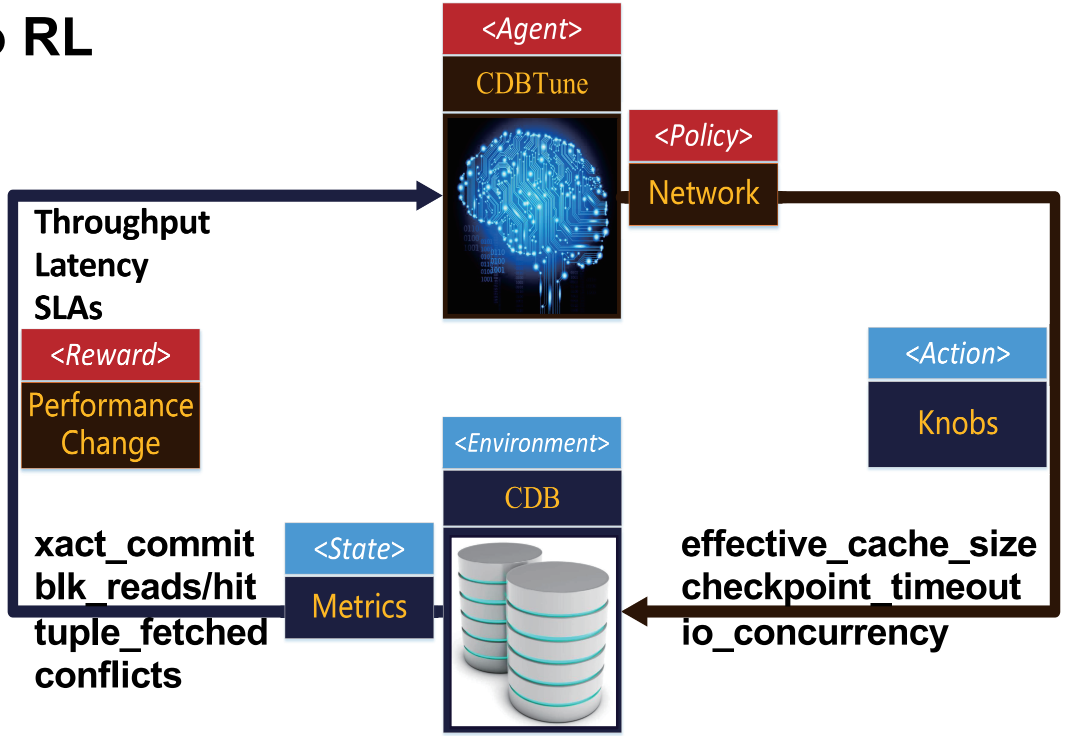


Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, Guoliang Li. Adaptive code learning for Spark configuration tuning. ICDE, 2022.

(3.1) Reinforcement Learning for Knob Tuning

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

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



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.

(3.1) Reinforcement Learning for Knob Tuning

- **Issue1: How to choose an appropriate RL approach**
  - **Challenge: Many continuous runtime 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: $Q(s_t, a_t|\theta^{Q})$

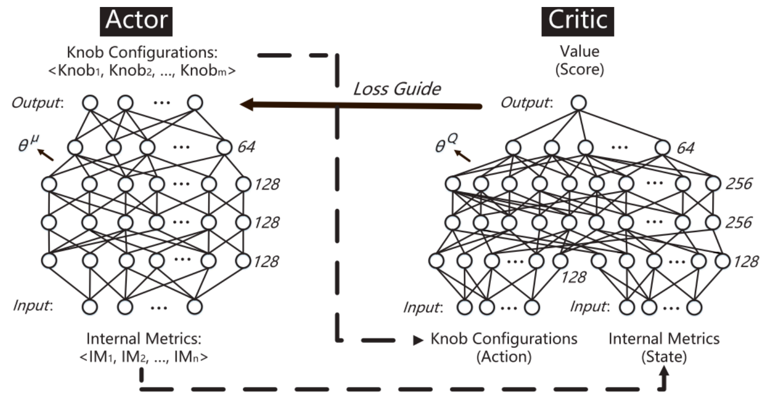
Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li et al. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. SIGMOD, 2019.

(3.1) Reinforcement Learning for Knob Tuning

Issue2: How to train an RL-based Model (e.g., DDPG)

Challenge: Optimize the tuning strategy with execution rewards

- **Design effective reward function *r* (*current benefit*):**



$$r = \begin{cases} \underbrace{((1+\Delta_{t\to 0})^2-1)}_{\text{Improvement over default setting}}\underbrace{|1+\Delta_{t\to t-1}|}_{\text{Improvement over (t-1) setting}}, & \Delta_{t\to 0} > 0 \\ -((1-\Delta_{t\to 0})^2-1)|1-\Delta_{t\to t-1}|, & \Delta_{t\to 0} \leqslant 0 \end{cases}$$

- **Actor Network Training:** Update with the score estimated by the Critic

$$\nabla_{\theta^{\pi_A}}\pi_A = \nabla_{A_i}Q(S_i', A_i|\pi_C)\cdot\nabla_{\theta^{\pi_A}}\pi_A(S_i'|\theta^{\pi_A})$$

$Q(S_i', A_i|\pi_C)$ → The output of Critic

- **Critic Network Training:** Update with accumulated *long-term benefit*:

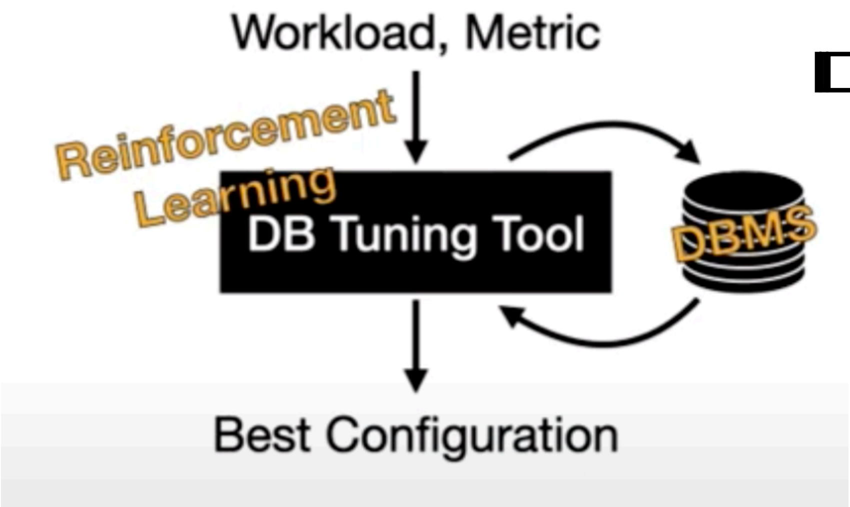
$$L = (Q(S_i', A_i|\pi_C) - Y_i)^2$$
$$Y_i = R_i + \tau\cdot Q(S_{i+1}', \pi_A(S_{i+1}'|\theta^{\pi_A})|\pi_C)$$

Y_i → Long-term benefit based on the reward

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

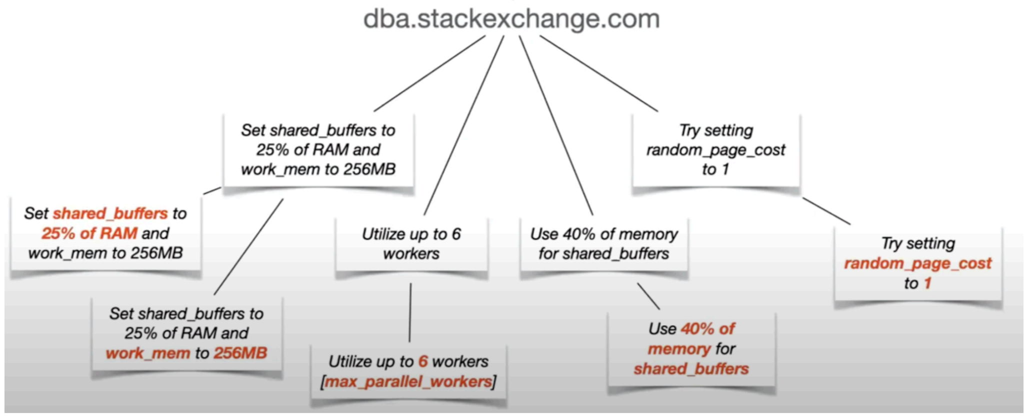
(3.2) Reinforcement Learning + Tuning Hints

- Limitations in RL-based tuning
  - High tuning overhead
  - Require DBAs (e.g., decide the knob ranges)



- Basic Idea: Tuning hints from manual

  (1) Collect tuning hints from website



Extract hints from manual

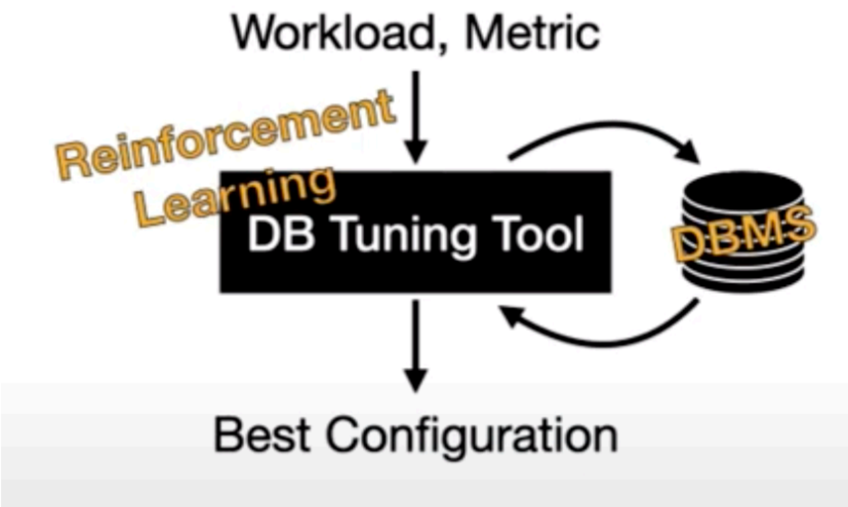
$$Parameter = Value\ [*\ System\ Property][*\ Constant]$$

Given in Text   RAM/Disk/Cores

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

(3.2) Reinforcement Learning + Tuning Hints

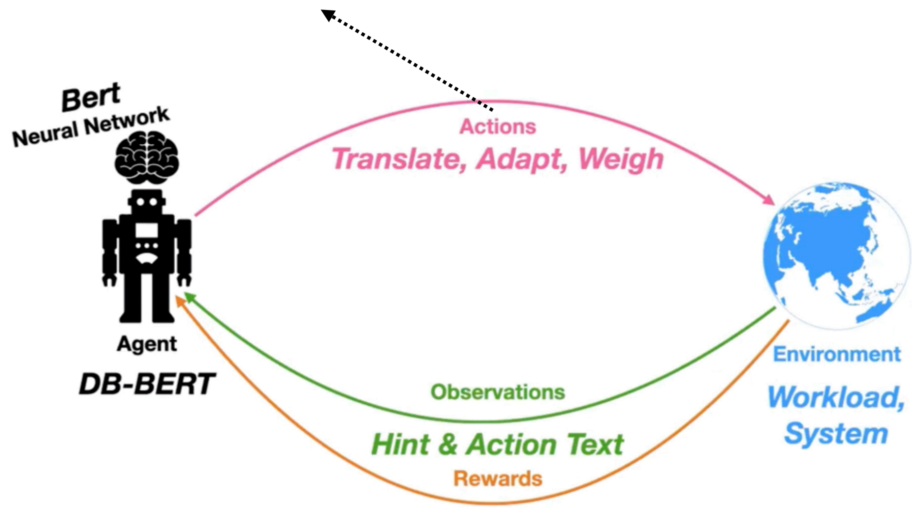
- Limitations in RL-based tuning
  - High tuning overhead
  - Require DBAs (e.g., decide the knob ranges)



- Basic Idea: Tuning hints from manual

  (2) Apply the tuning hints with the reinforcement learning model

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



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

Summarization of Learned Knob Tuning

	Quality	Training Efficiency	Training Data	Adaptivity
Gaussian Process (historical data)	✓	--	✓✓	✓
Gaussian Process (+ empirical features)	✓	✓	✓	✓✓
Gaussian Process (pre-trained models)	✓	✓	✓✓	✓✓
Deep Learning (resource issues)	✓	✓	✓✓	✓
Deep Learning (+ code encoding)	✓✓	✓	✓✓	✓✓
Reinforcement Learning (from scratch)	✓✓	--	No Prepared Data	✓
Reinforcement Learning (+ tuning hints)	✓	--	✓✓✓	✓✓

Take-aways of Knob Tuning

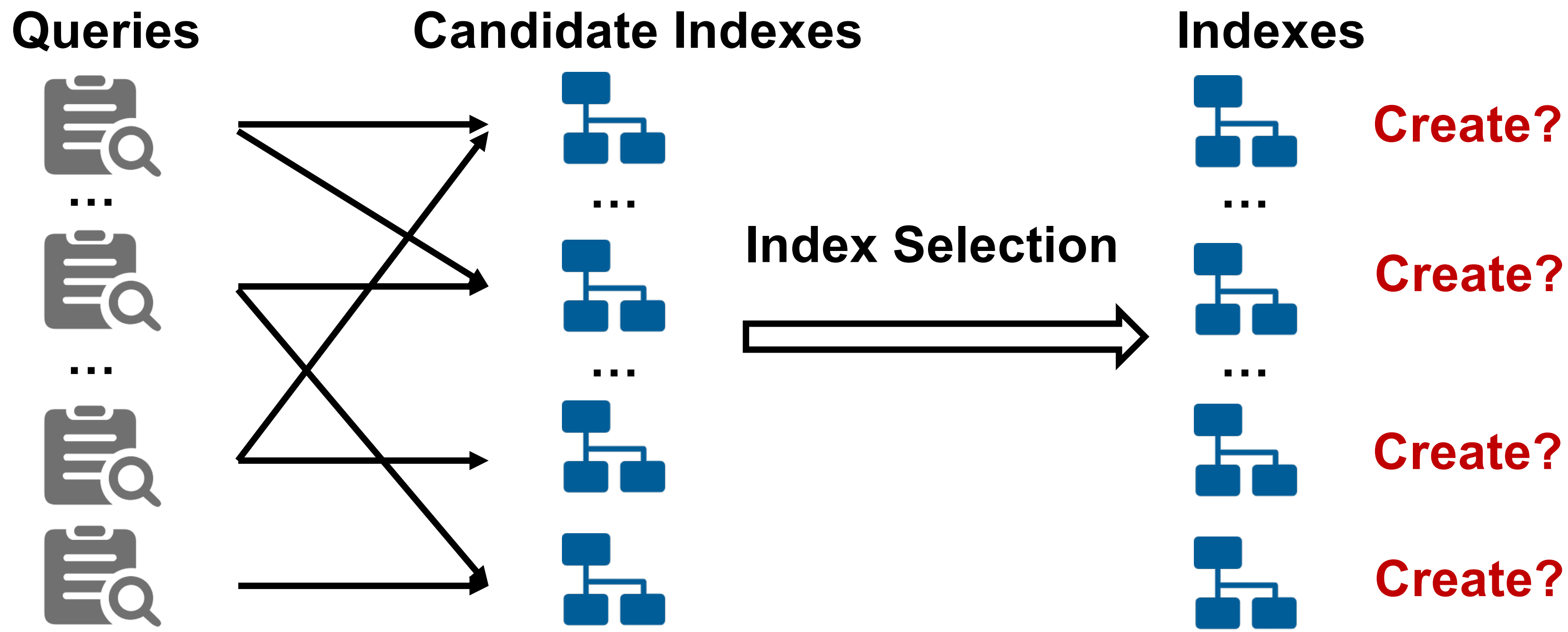
- **Gradient-based GP methods reduce the tuning complexity by filtering out unimporant features.** However, it heavily relies on training data, and requires other migration techniques to adapt to new scenarios
- **Deep learning method considers both query performance and resource utilization.** And they can significantly reduce the tuning overhead.
- **Reinforcement learning methods take longest training time, e.g., hours, from scratch.** It 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

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Index Management

Problem Definition: Given a set of queries W and resource constraint D (e.g., disk limit), create a collection of indexes so as to optimize the execution of these queries under the constraint D. → NP-hard

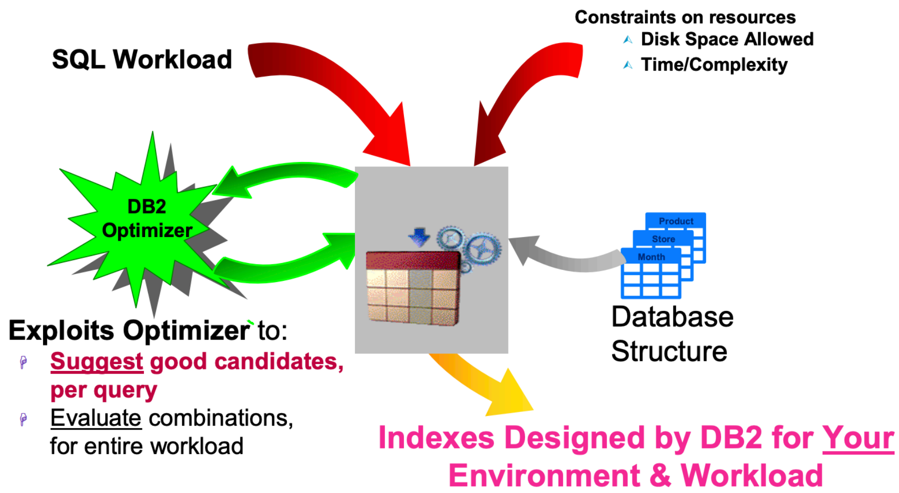


Index Management

- **Index Benefit Estimation**
  - The benefit of building an index on a column
- **Index Selection**
  - Column selection
  - Index-type selection, e.g., B-tree, Hash, bitmap
- **Index Update**
  - Adding or removing an index

Heuristic Index Selection

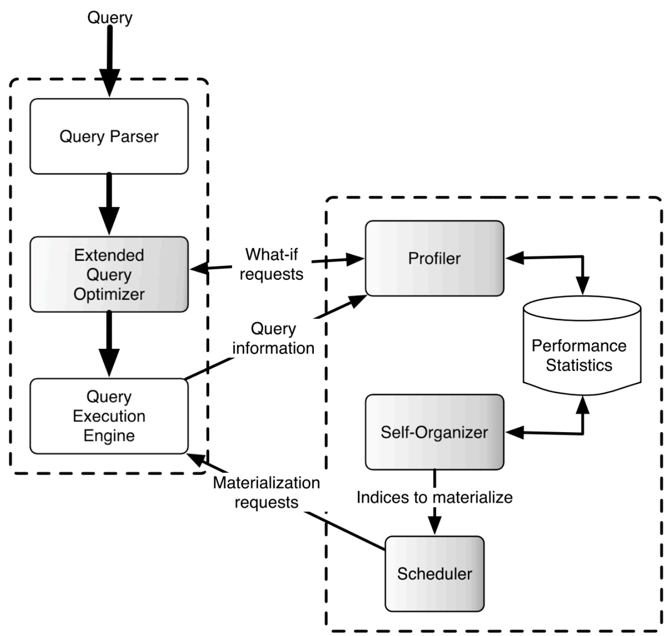
- **Motivation: Proper indexes can significantly improve the performance**
- **Basic Idea: Model index selection as a knapsack problem and heuristically find the best indexes under disk limit**
- **Challenge: There are correlations between indexes (e.g., index sizes)**
- **Solution:**
  - **Model index selection as a knapsack problem**
    - **Item: Candidate index**
    - **Item weight: Index size**
    - **Value: Cost reduced by the index**
  - **Heuristically select the highest-benefit indexes**
    - **Benefit: Cost Reduction / Index Size by optimizer**



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

Heuristic Index Selection for Dynamic Workloads

- Motivation: Performance gets unstable for dynamic workloads
- Basic Idea: Split workloads into epochs and finetune indexes for each epoch
- Challenge: Online index update for new queries
- Solution:
  - Divide a workload into epochs of queries
  - Generate candidate indexes for each new query
    - Index Benefit: average latency reduction for the queries within the same epoch
    - Benefit Estimation: Estimate the index benefit through what-if call (*similar queries have similar index benefits*)
    - Update the index set and statistics
  - Create indexes with highest index benefit at each epoch



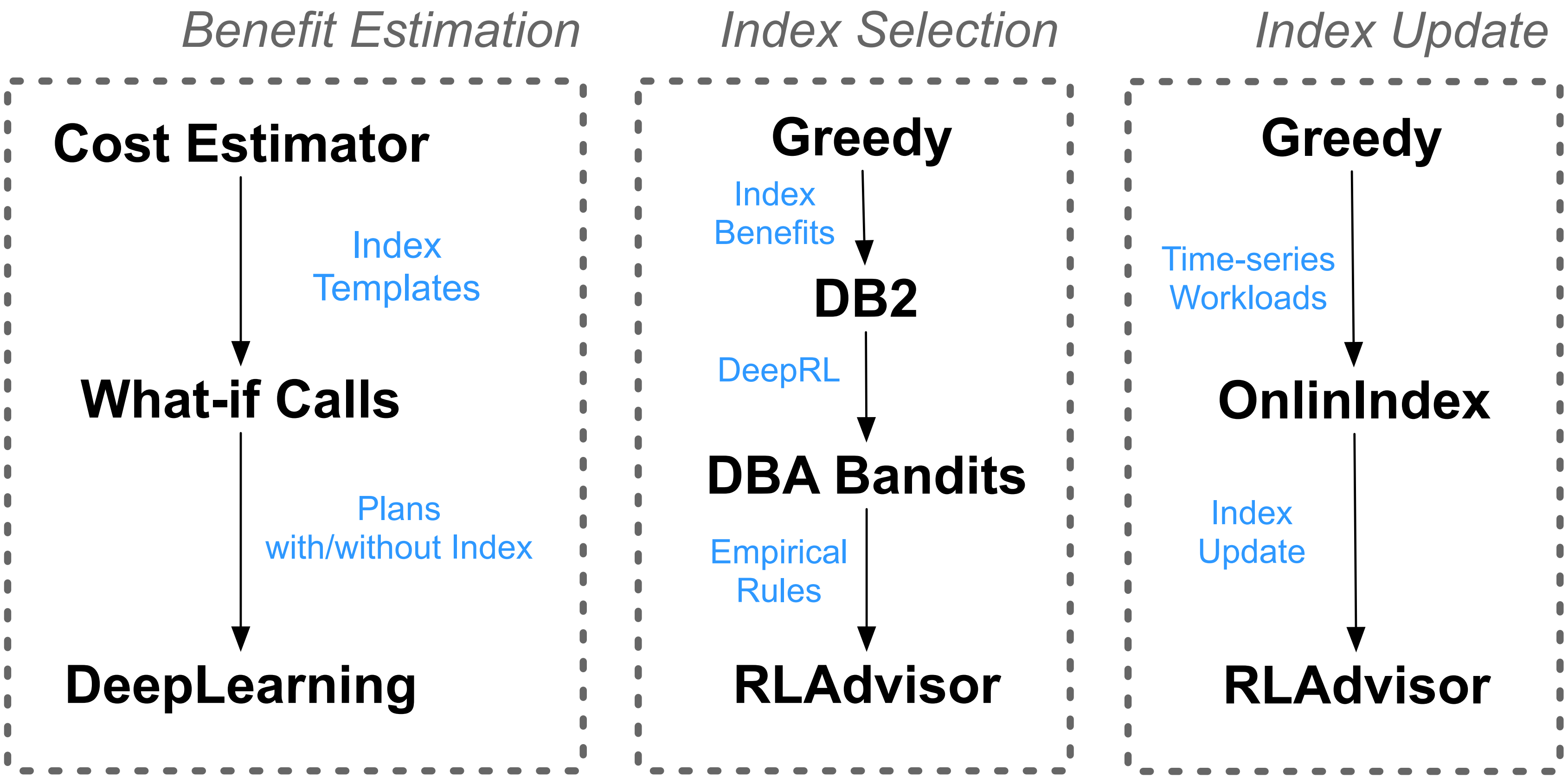
K. Schnaitter, S. Abiteboul, T. Milo and N. Polyzotis. On-Line Index Selection for Shifting Workloads. In ICDE 2007.

Learning-based Index Selection

☐ Why heuristics → learned index selection?

- **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);
- **Find better solutions from numerous candidate indexes**
  - ➢ Columns have different access frequencies, data distribution
- **Redundant indexes may cause negative effects**
  - ➢ Increase maintenance costs for update/delete operations

Learning-based Index Recommendation



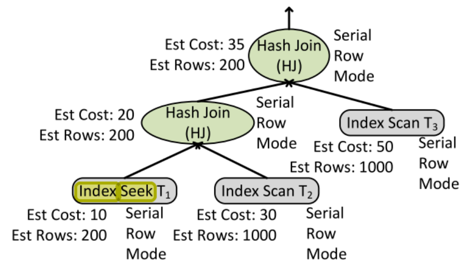
Index Benefit Estimation

☐ Challenge

- **The index benefit is hard to evaluate**
  - ➢ Multiple evaluation metrics (e.g., index benefit, space cost)
  - ➢ Cost estimation by the optimizer is inaccurate
- **Correlations with other components**
  - ➢ Multiple column access, data refresh
  - ➢ Conflicts between Indexes

Deep learning for Index Benefit Estimation

- **Motivation: Critical to estimate index benefits by comparing execution costs of plans with/without created indexes**
- **Core Idea: Model benefit estimation as an ML classification task**
- **Challenge: Hard to accurately estimate the index benefits**
- **Solution:**
  - **Prepare training data**
    - **Query Plans + Costs under different indexes**
  - **Train the classification model**
    - **Input: Two query plans with/without indexes**
    - **Output: 1 denotes performance gains; 0 denotes no gains**
  - **Solve the index selection 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.

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

Learning-based Index Selection

- **Challenges**
  - **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 is expensive**
    - Hard to estimate the number of involved pages

Reinforcement Learning for Index Selection

- **Motivation: Index selection using reinforcement learning**
- **Challenge 1: 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.

- **Challenge 2: How to choose from candidate indexes?**
  - Map into *Markov Decision Process* (MDP)

State: Info of current built indexes $\xrightarrow{\text{large state space}}$ **DQN Model**

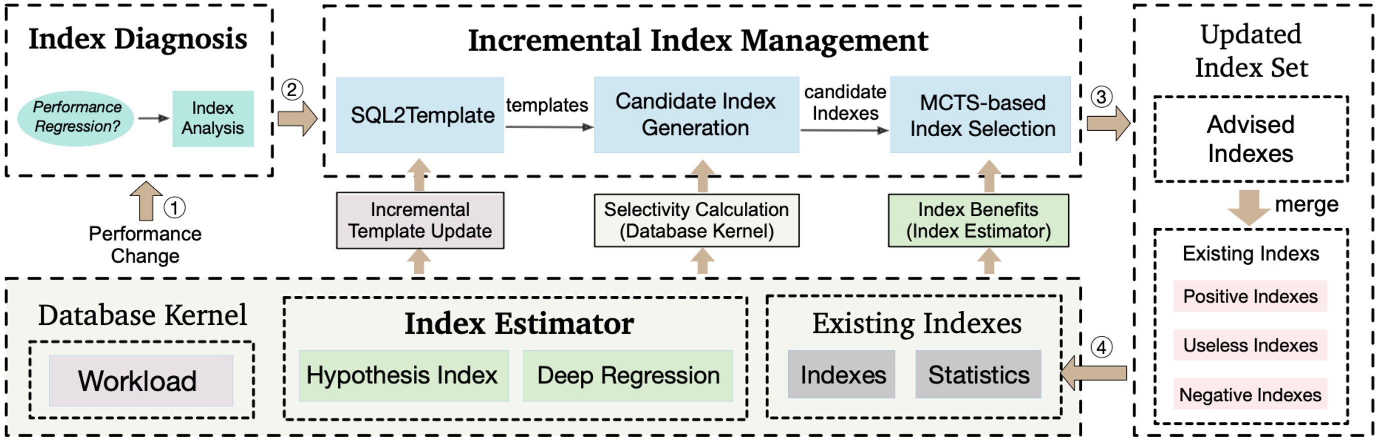
Action: Choose an index to build $\xrightarrow{\text{discrete action space}}$ **DQN Model**

Reward: Cost reduction ratio after building the index

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

MCTS for Index Update

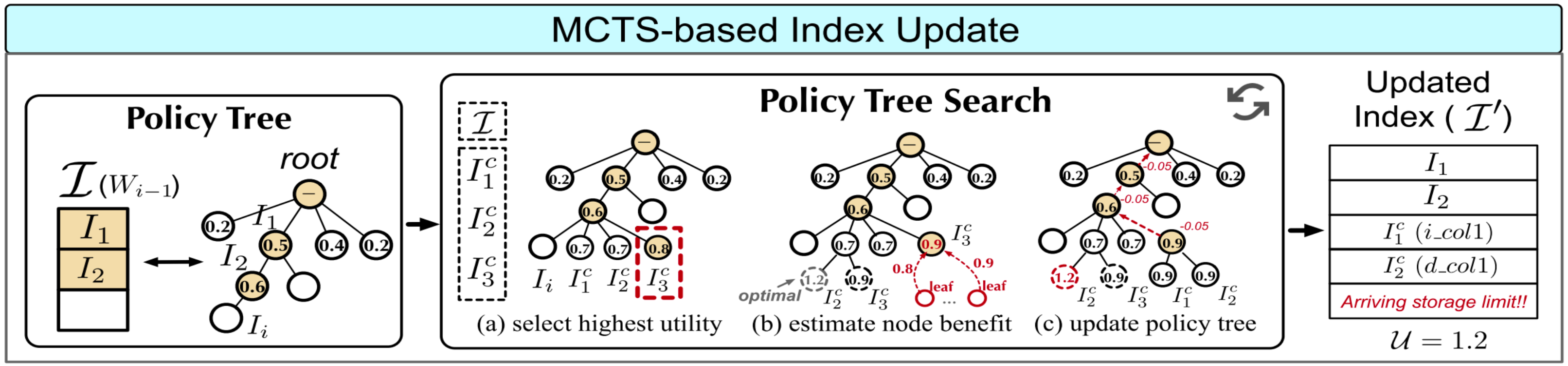
- **Motivation: Existing methods cannot incrementally update indexes**
- **Basic Idea: Incrementally add/remove indexes with MCTS**
- **Challenge: Consider both the read and write queries**
- **Solution:**
  - **Index Diagnosis (anomaly detection)**
  - **Incremental Index Update (policy tree search)**
  - **Index Benefit Estimation (deep regression)**



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

MCTS for Index Update

- **Motivation: Existing methods cannot incrementally update indexes**
- **Basic Idea: Incrementally add/remove indexes with MCTS**
- **Challenge: Consider both the read and write queries**
- **Solution:**
  - **Index Problem Diagnosis: Detect whether the performance regression is caused by index issues;**
  - **Candidate index extraction: Cluster queries → Map to query templates → Extract candidate indexes;**
  - **Incremental Index Update: Initialize a policy tree with existing indexes → Add new candidate indexes;**
  - **Index Benefit Estimation: Index Update Costs = seek_tuples * cpu_cost + insert_tuples * cpu_index_tuple_cost**



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

Summarization of Index Management

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

Take-aways of Index Advisor

- **Learned index estimation is more robust than cost models**
- **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

Learned Advisors

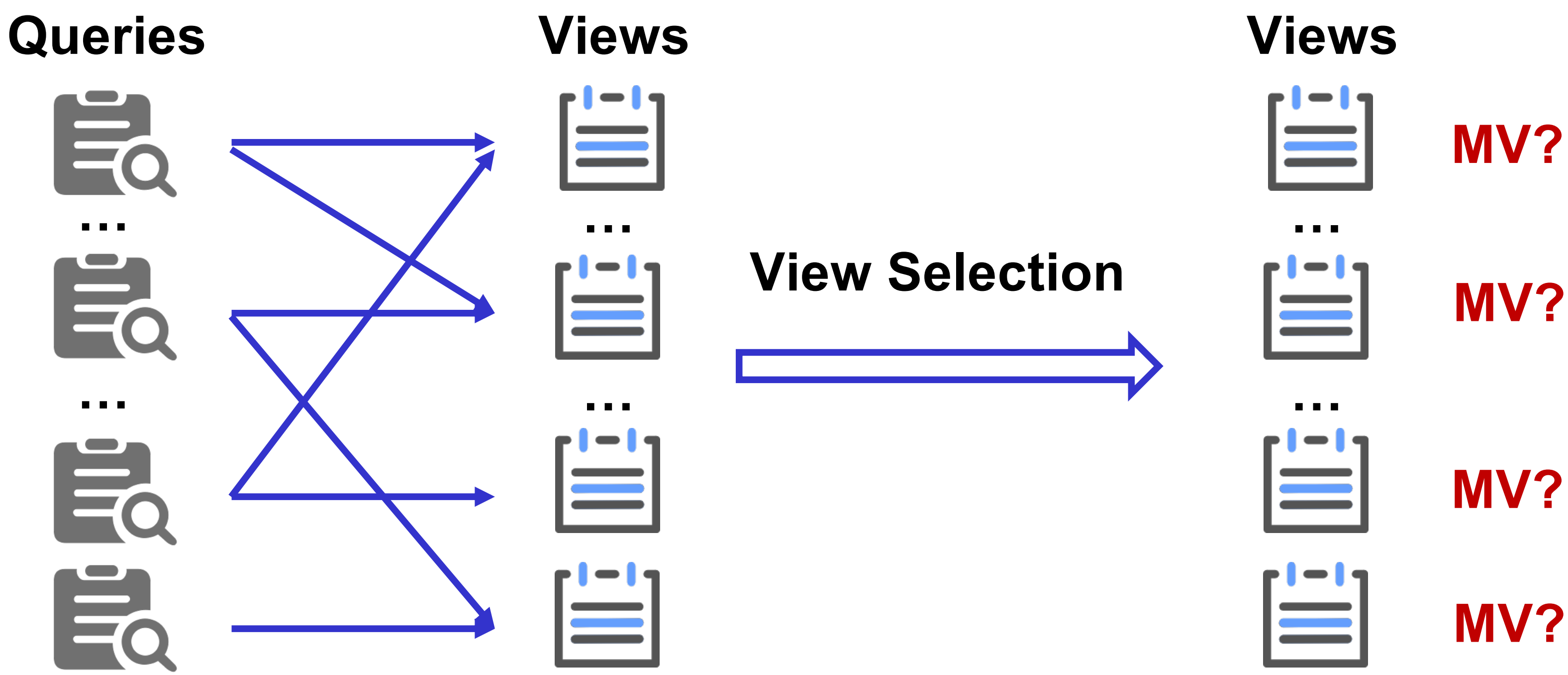
- Learned Knob Tuning
- Learned Index Advisor
- Learned View Advisor
- Learned Partition Advisor
- Learned Data Generation

View Management

- View Benefit Estimation
  - The benefit of building a materialized view (MV) for a subquery
- View Selection
  - Which subquery to create an MV
- View Update/Refresh
  - Adding or removing an MV

View Selection

Problem Definition: Given a workload *Q* and a space budget, select optimal subqueries to materialize (MVs), including (i) MV benefit estimation; (ii) MV Selection; (iii) MV update; (iv) MV rewrite.



View Selection

- **Materialized Views (MVs) can optimize queries**
  - Share common subqueries
- **Space-for-time trade-off principle**
  - Materialize hot data (MVs) within limited space
  - How to estimate the MV utilities
- **The number of potential MVs grows exponentially**
  - Greedy/Genetic/other-heuristics work bad

Traditional View Selection Methods

- **Given a workload, select and maintain materialized views that minimize the total latency within a limited materialized view storage space (NP-hard).**
- **Traditional Methods**
  - **Greedy:** WATCHMAN, DynaMat, CloudViews
  - **Genetic:** EA, Hybrid-GHCA
  - **Coral Reefs Optimization Algorithm:** CROMVS
  - **Backtracking Search Optimization Algorithm:** BSAMVS-penalty
  - **Integer Linear Programming:** BIGSUBS, HAWC

Learned View Management

- **Limitations of Traditional Methods**
  - **View's benefit estimation. Not accurate.**
    - **Traditional models is not accurate for view benefit/multiple view benefit estimation.**
    - **Hard to estimate materialized view update cost.**
  - **View selection. Not generalizable.**
    - **Designed and work well for specific scenarios or workloads.**
    - **Rely on assumptions that are not always right**
  - **View update. Long Delay.**
    - **Based on accumulated benefits and creation cost of views.**
    - **Hard to estimate the a view's future benefit and recreation cost.**

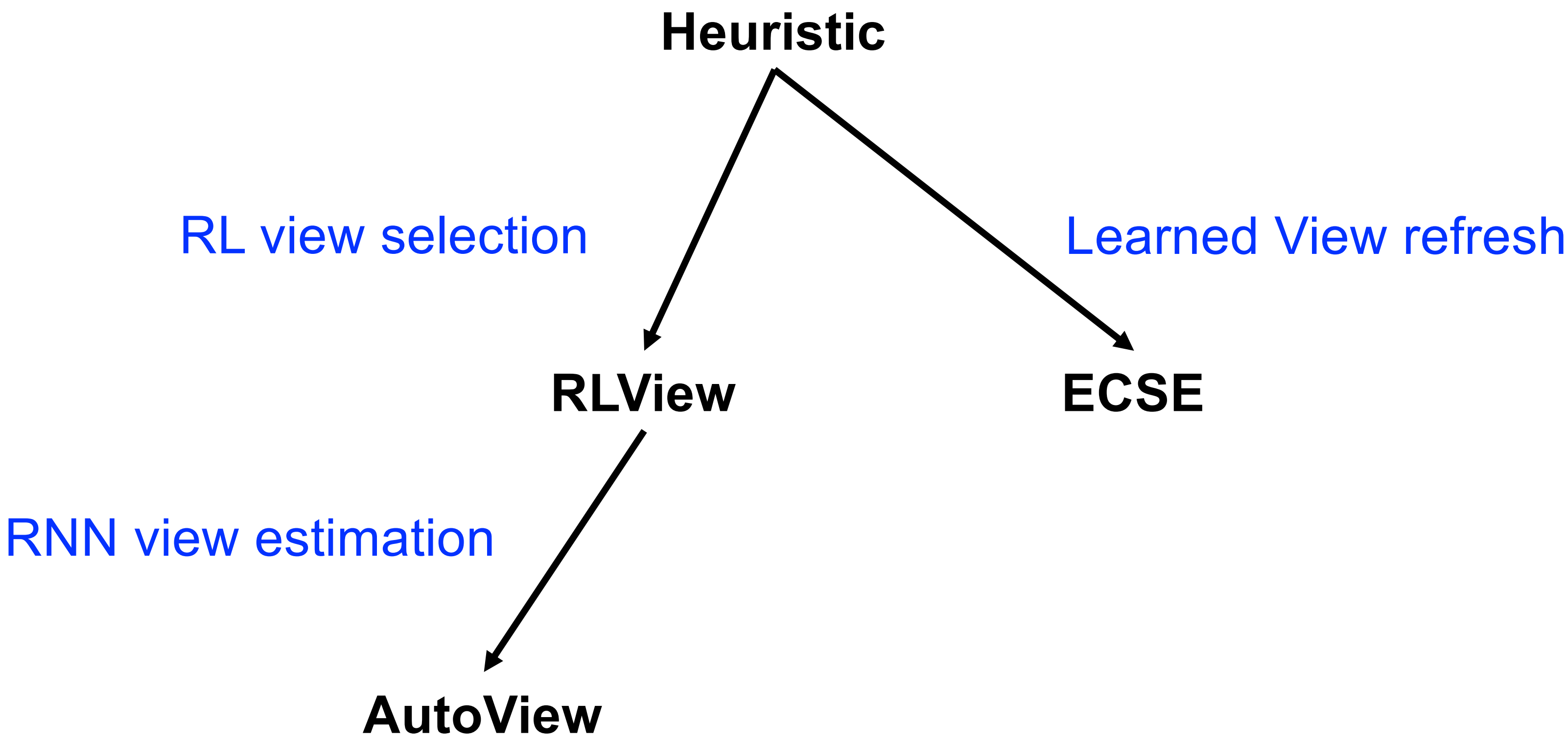
Learned View Management

- **Motivation**
  - **Estimate view benefit accurately.**
    - **Learned based methods from real runtime statistics.**
      (Also verified in learned cardinality and learned join order selection)
  - **Generalizable on different workloads.**
    - **Learns from historical workloads and learns directly from the view selection performance without human experience.**
  - **Predict views' future benefit.**
    - **Learns from historical MV utilization and predict future benefit and update cost.**

Learned View Management

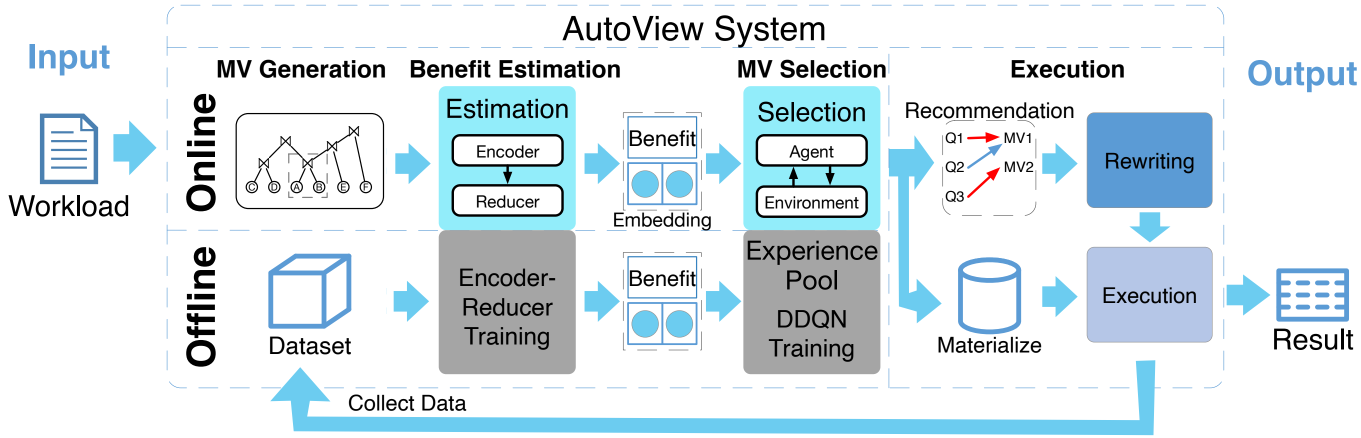
- Challenges
  - View and query need to be encoded for neural networks.
  - New models need to be designed for view benefit estimation.
  - View selection models should be efficient and flexible.
- Optimization Goals
  - View Quality
  - Model Adaptivity
  - Support view update

Learned View Management



Learned View Estimation: AutoView

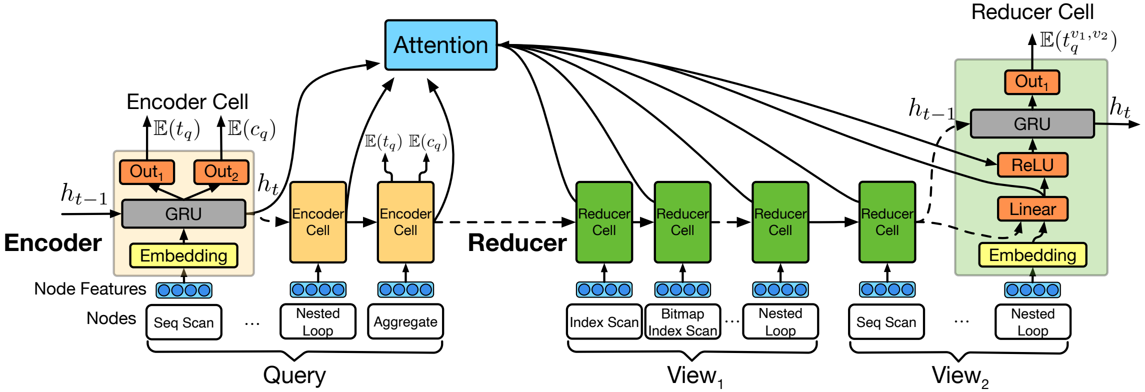
- **Motivation**
  - Estimate views' benefit more accurately.
  - Support variable number of views in RL for view selection.
- **Challenges**
  - Views have different benefits on queries in workload.
  - Hard to extend state representation after model training.

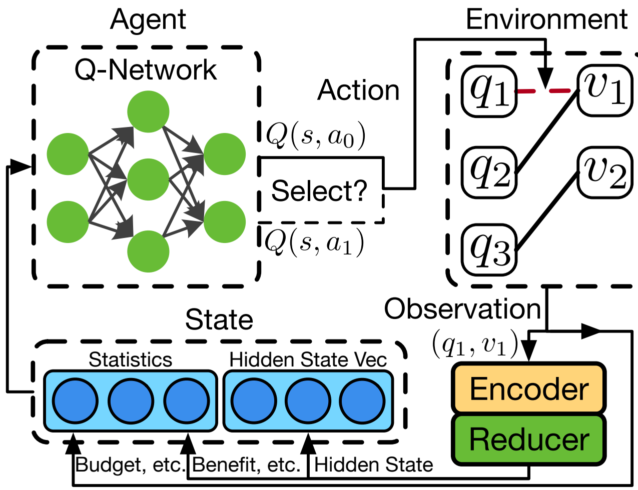


Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In ICDE, 2021.

Learned View Estimation: AutoView

- **Estimate the query-view benefits with encoder-reducer model:**
    - Two LSTM network for query and views, which captures query-MV correlations with attention.
- **Select optimal query-view combinations with reinforcement learning iteratively.**





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

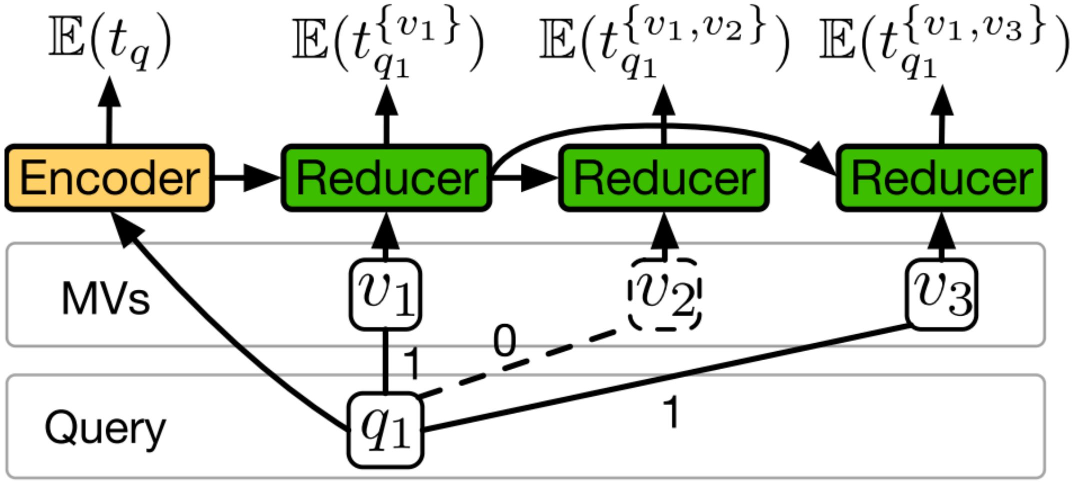
Learned View Estimation: AutoView

☐ Feature Extraction

- **Previous work take candidate views as fixed length →**
- **Encode various number and length of queries and views with an *encoder-reducer model*, which captures correlations with *attention***

☐ Model Construction

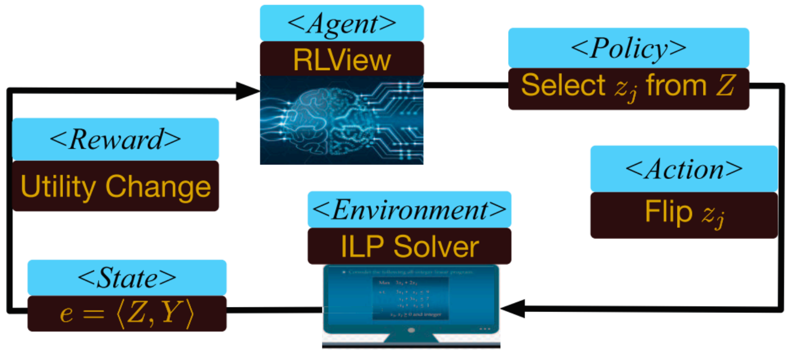
- **It is hard to jointly consider MVs with conflicts →**
- **(1) Split the problem into sub-steps that select one MV;**
- **(2) Use attention-based model to estimate the MV benefit**



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

Learned View Selection: RLView

- **Motivation**
  - RL performs well on combinatorial optimization problem.
- **Challenges**
  - How to solve view selection problem in RL framework.
- **Solutions**
  - Cluster equivalent queries and select the least overhead ones as the candidate;
  - Represent MVs as a fixed-length state vector and solve with DQN model;
  - Estimate the MV benefits with DNN.

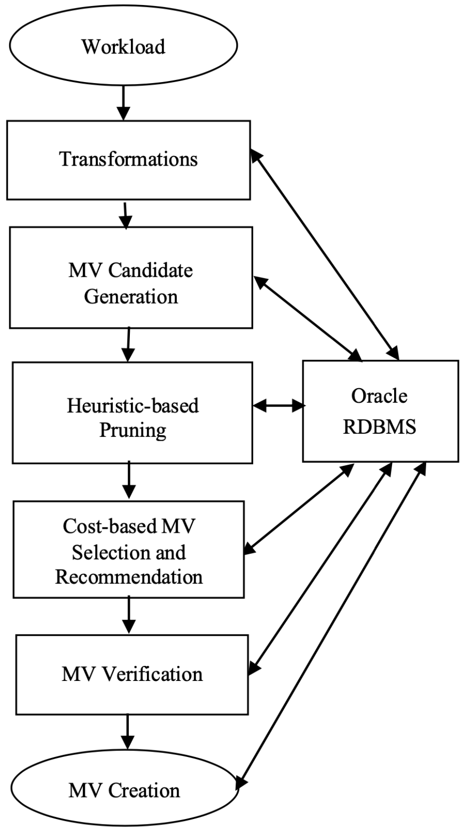


$Z = \{z_j\}$:z_j is a 0/1 variable indicating whether to materialize the subquery s_j
$Y = \{y_{ij}\}$:y_{ij} is a 0/1 variable indicating whether to use the view v_{s_j} for the query q_i

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

Learned View Update: ECSE

- **Motivation**
  - Support MV refresh.
- **Challenges**
  - Hard to estimate refresh benefit/cost from historical workload.
- **Solutions**
  - Traditional view generation, estimation, and selection；
  - Use a neural network model to predict future DML operations and MV usage for scheduling the refresh.
  - Use linear regression to estimate MV refresh time with
    - MV size, refresh method, affected number of rows,
    - previous refreshes time.



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

Learned View Management: Comparison

Method	View Quality	Adaptability	View Update	View Estimation	View Selection	View Update
RLView	Medium	Low	No	Learned	Learned	-
AutoView	High	High	No	Learned	Learned	-
ECSE	Medium	Medium	Yes	Heuristic	Heuristic	Learned

Learned View Advisor: Take-away

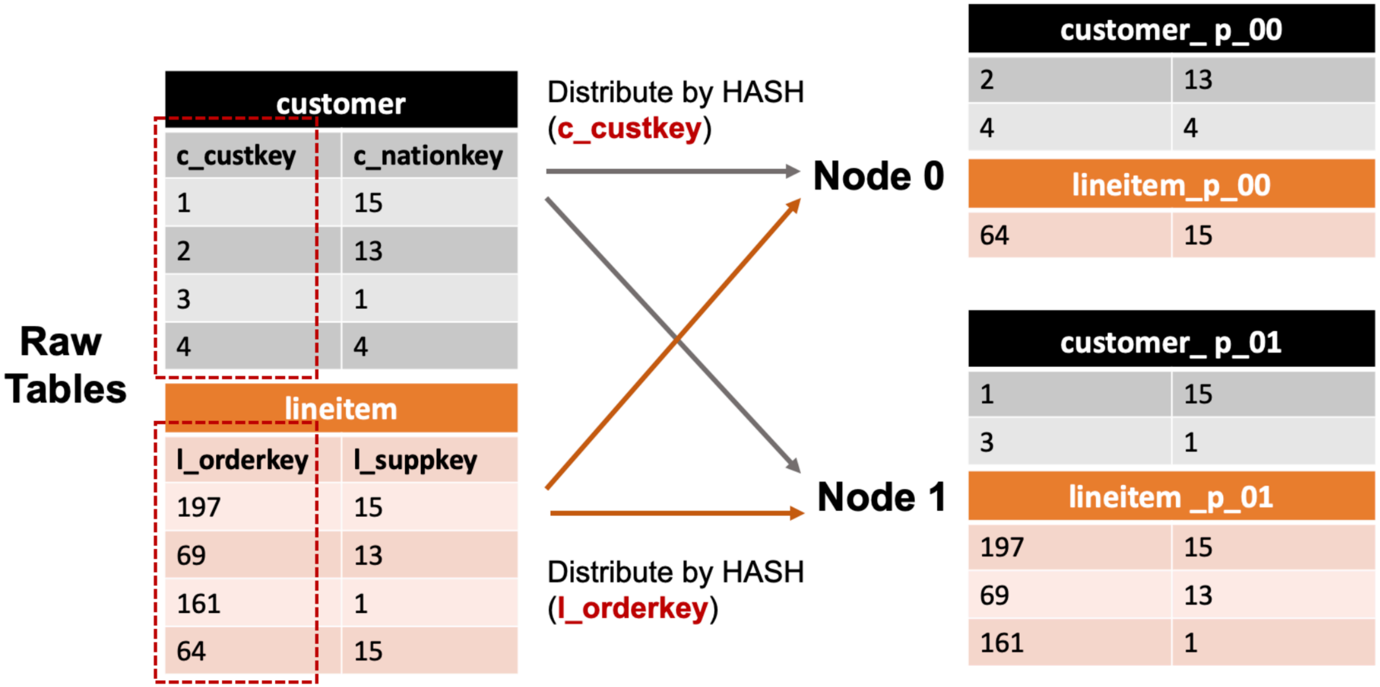
- **Learned view selection gains higher performance than heuristics**
- **Learned view selection works well for read workloads**
- **Learned view benefit estimation is more accurate than traditional empirical methods**
- **Learned view benefit estimation is accurate for multiple-view optimization**
- **Open Problems:**
  - Learned MV update/refresh
  - Learned MV rewrite

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Database Partition

Problem Definition: Given tables $\{T_1, T_2, \ldots, T_m\}$ and a partition function F, database partition selects columns for each table T_i as the partition key, and allocate the tuples in T_i into partitions using F, such that the workload performance is optimal.



SELECT * FROM customer,lineitem WHERE **c_nationkey** = **l_suppkey** and **c_nationkey** < 4;

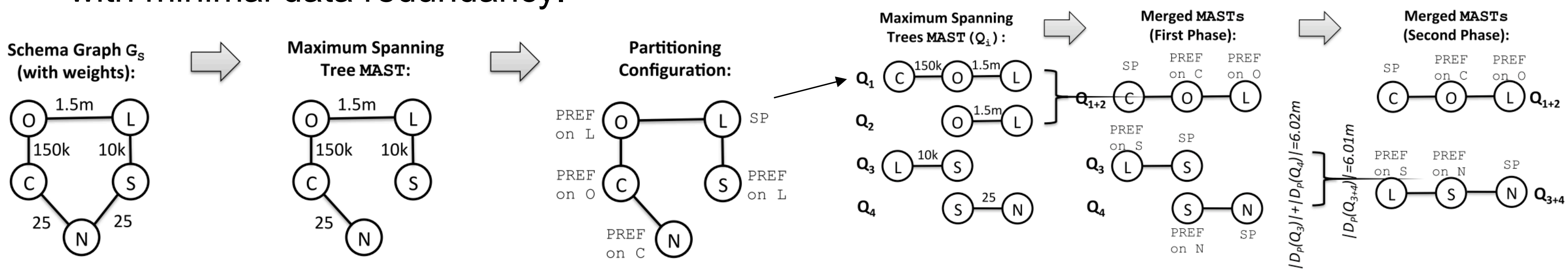
Heuristic Database Partition for OLAP Workloads

- **Motivation**
  - ➢ Reduce the network costs by judiciously partitioning tables
- **Core Idea**
  - ➢ Heuristically co-partition (the tuples of the referenced table are on the same node of referencing table) tables by foreign-key relations
- **Challenge**
  - ➢ It is hard to find a suitable partitioning scheme (for many tables with join correlations) that maximizes data locality.
  - ➢ There can be different partition schemes. How to merge them so as to reduce the data redundancy caused by co-partitioning.

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

Heuristic Database Partition for OLAP Workloads

- **Represent the specific dataset schema → Build a graph mode**
  - Initialize a graph model G,
    - *Nodes*: tables, *Edges:* foreign keys, *Edge weight*: the size of smaller table connected to the edge
- **Improve data locality (reduce network costs) → Partition by join predicates**
  - REF partitioning: a table is co-partitioned by the join predicate that refers to another table;
  - Utilize maximum spanning tree to extract subsets of edges (a partition strategy) that (1) partition all the tables and (2) maximize the data locality.
- **Full data locality may introduce duplicate tuples → Merge duplicated partitions**
  - Utilize dynamic programming to merge candidate partition strategies so as to find the one with minimal data redundancy.

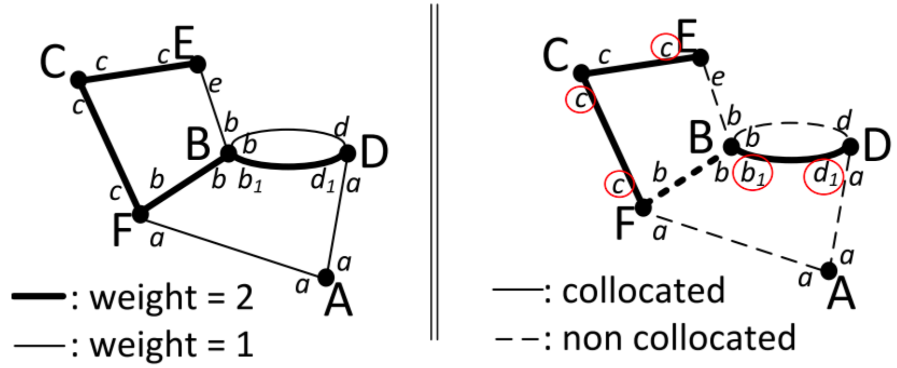


Improve Data Locality **Reduce Data Redundancy**

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

Traditional Database Partition

- **Motivation: Partition on join columns can significantly reduce the network communication and reduce execution costs**
- **Core Idea: Combine exact and heuristic algorithms to find good partition strategies for different workloads**
- **Challenge: Picking join columns as partition keys is NP-complete**
- **Solution**
  - **Build a Join Multi-Graph**
    - *Vertices* are tables, *Edges* denote join relations
  - **Partition with hybrid partitioning algorithms**
    - *Exact algorithm:* Assume each table only uses a column; and turn into an integer programming problem;
    - *Heuristic algorithm:* Select the table columns with largest edge weights



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

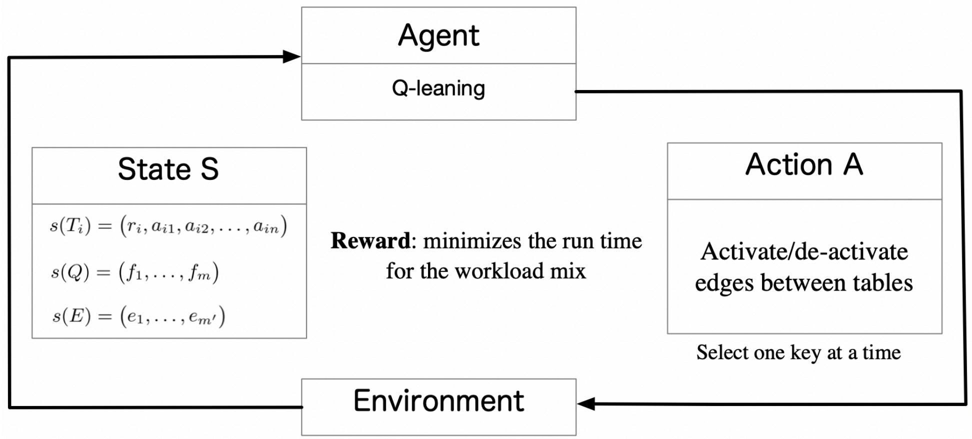
Learning-based Database Partition

☐ Motivation:

- **Consider both the data balance & access efficiency**
  - Place partitions on different nodes to speedup queries
  - Trade-off based on workload and data features
- **Combine ML to optimize the NP optimization problem**
  - **Combinatorial problem:** 61 TPC-H columns, 145 query relations, 2.3×10^{18} candidate combinations

Reinforcement Learning for Database Partition

- Motivation: OLAP Workloads contain complex and recursive queries
- Core Idea: Explore column combinations as partition keys with RL
- Challenge: Characterize partition features; Migrate to new workloads
- Solution
  - Extract partition features as a vector
    - [tables, query frequencies, foreign keys]
  - Use DQN to partition the tables for a workload
    - Iteratively partition tables by long-term reward
  - Support new workloads with trained models
    - Train a cluster of DQN models on typical workloads;
    - Pick models whose workloads are similar to the new workload to partition tables.



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 under complex workloads (e.g., with multiple joins)
- Learned key-selection partition has much higher partitioning latency (e.g., data collection, model training)
- Open Problems:
  - Adaptive partition for relational databases
  - Partition quality prediction
  - Improve partition availability with replicates

Learned Advisors

- ☐ Learned Knob Tuning
- ☐ Learned Index Advisor
- ☐ Learned View Advisor
- ☐ Learned Partition Advisor
- ☐ Learned Data Generation

Automatic Query Generation

☐ Motivation

- Companies generally will not release their data and 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 representative?**

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：**

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；

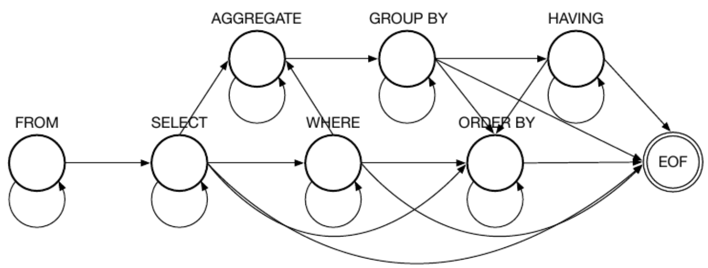
Lixi Zhang, Chengliang Chai, Xuanhe Zhou, Guoliang Li. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. SIGMOD 2022.

Automatic Query Generation

Query Legality

➢ SQL Grammar:

- FSM



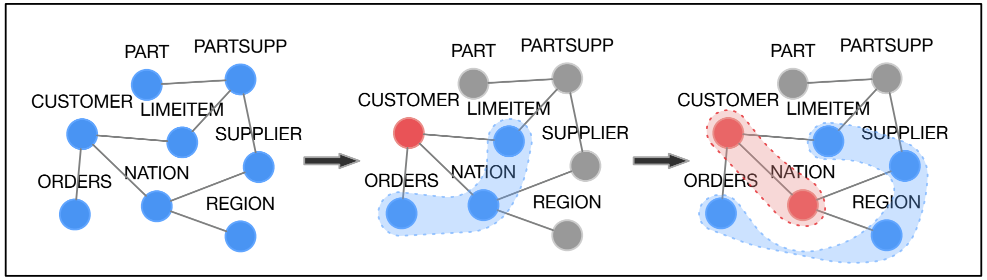
Advantage:

✓ 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.

Automatic Training Data Generation

Motivation

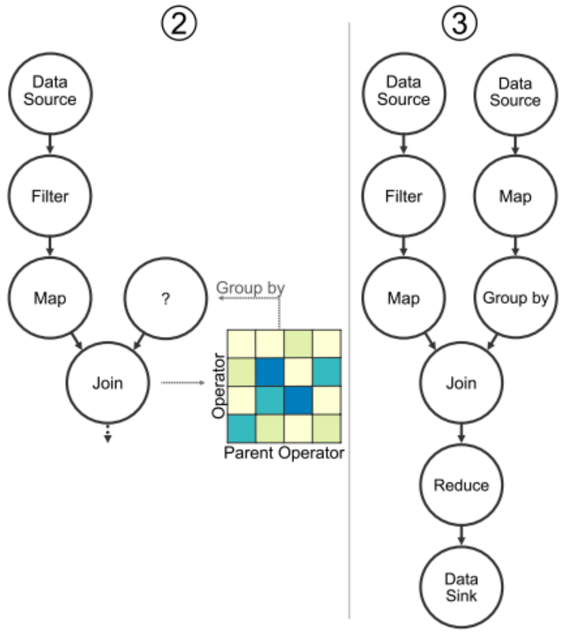
- **Machine learning is widely adopted in database components**

- **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

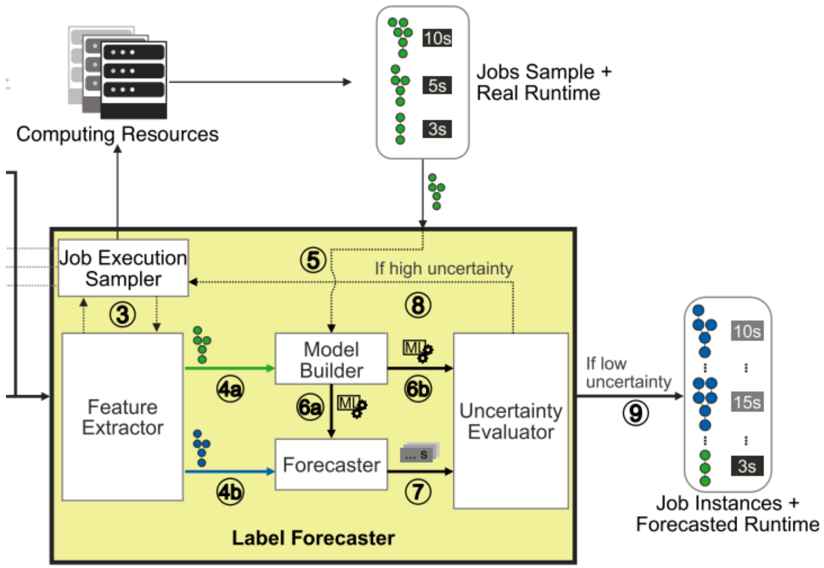
- **Challenges in existing workload generators (TPC-H, sqlsmith)**
  - Limited SQL templates; while real queries have various structures;
  - Fail to label the SQL queries (e.g, cost, execution time)
- **Core Idea: Reduce the labeling time by generating many query jobs and estimating the job latency**
  - SQL Sampling
    - A few real SQL queries + sample data;
  - Plan Synthesis
    - Generate abstract plans from the real SQLs;
    - Collect statistics, e.g., distribution of the longest plan paths;
    - Generate job by imitating the structures/patterns of the plans,
      - E.g., for join operator, they select the operator (Group by) as the child node with the max possibility (the transition matrix)



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

Automatic Training Data Generation

- **Challenges in existing workload generators (TPC-H, sqlsmith)**
  - Limited SQL templates; while real queries have various structures;
  - Fail to label the SQL queries (e.g, cost, execution time)
- **Core Idea: Reduce the labeling time by generating many query jobs and estimating the job latency**
  - SQL Sampling
    - A few real SQL queries + sample data;
  - Plan Synthesis
  - Label Forecasting
    - **Sample and execute jobs** → Get the real latency (labels)
    - **Build an estimator** → Evaluate the latency and uncertainty of the unexecuted jobs
    - **Incrementally sample jobs** → Reduce the uncertainty



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

Learned Prediction

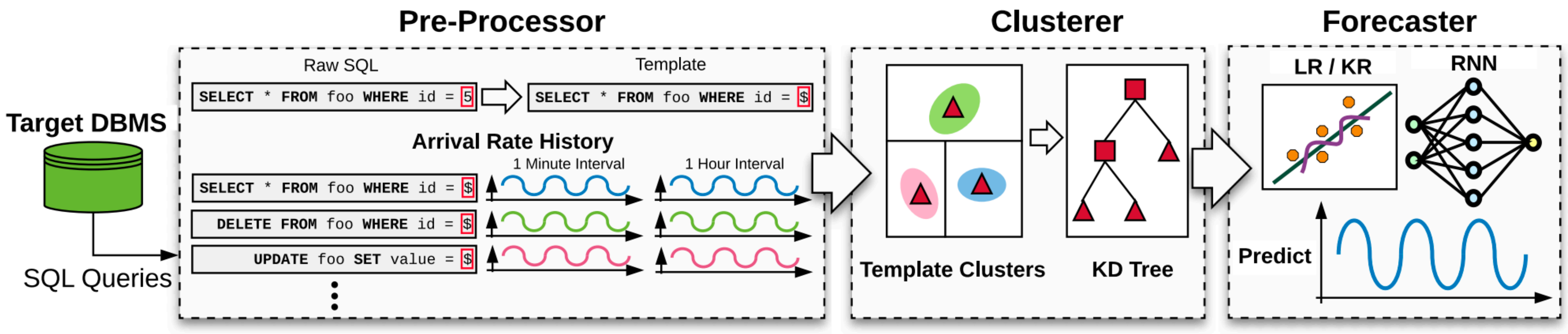
Prediction Problems

☐ Motivation

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

Learned Workload Prediction

- **Predict the future trend of different workloads**
  - **Pre-Processor** identifies query templates and the arrival-rate from the workload;
  - **Clusterer** combines templates with similar arrival rate patterns
  - **Forecaster** utilizes ML models to predict arrival rate in each cluster



Lin Ma, Dana Van Aken, and et al. Query-based Workload Forecasting for Self-Driving Database Systems. In SIGMOD, 2018.

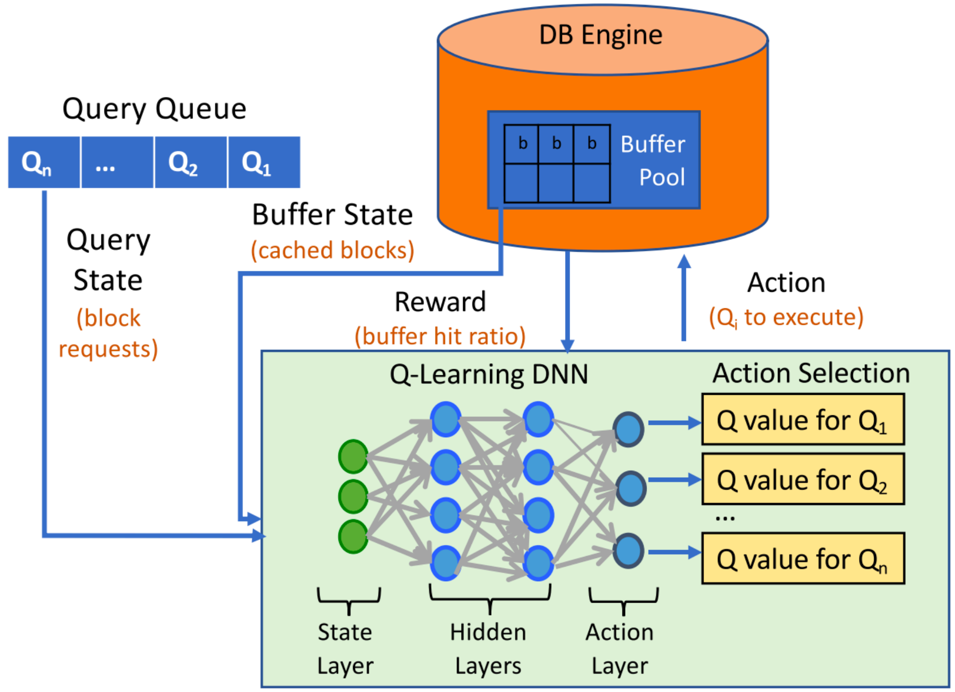
Learned Workload Scheduling

☐ **Learn to schedule queries to minimize disk access requests**

➢ 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



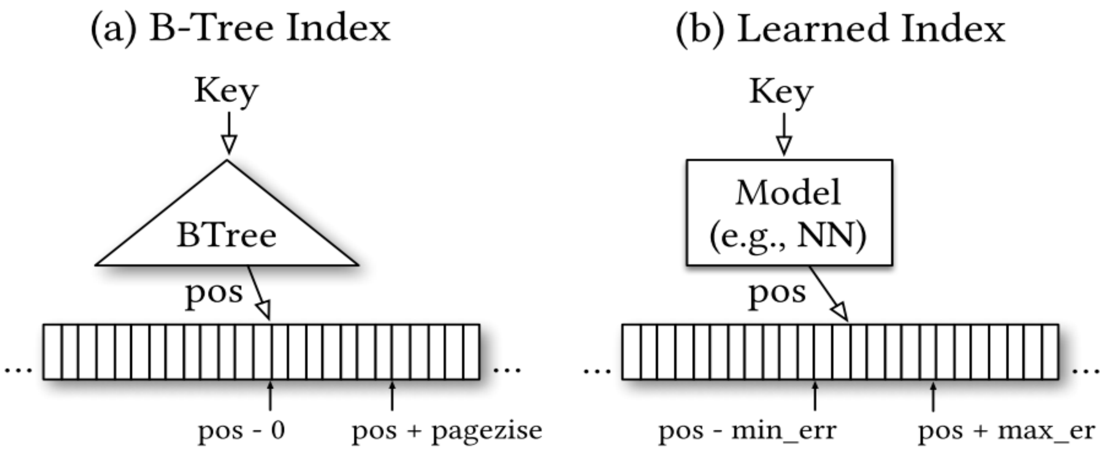
Chi Zhang, Ryan Marcus, and et al. Buffer Pool Aware Query Scheduling via Deep Reinforcement Learning. In VLDB, 2020.

Learned Index

Basic Idea of Learned Index

- **Model the cumulative distribution function(CDF) of the data to predict the location as:**

$$p = F(Key) * N$$



- **Data sampling → Training CDF → Predict approximate location → Search precise location**

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

Why 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
  - Highly parallel, adapt to modern hardware like GPU and TPU

Learned Index: Formulation

Problem Formulation

– Given a set of key-value pairs, index is a data structure that improves the speed of data retrieval operations such as: lookup the value of the key, range query, nearest neighbor query, etc.

Traditional Methods

– B-Tree, ART, R-Tree,…

Limitations

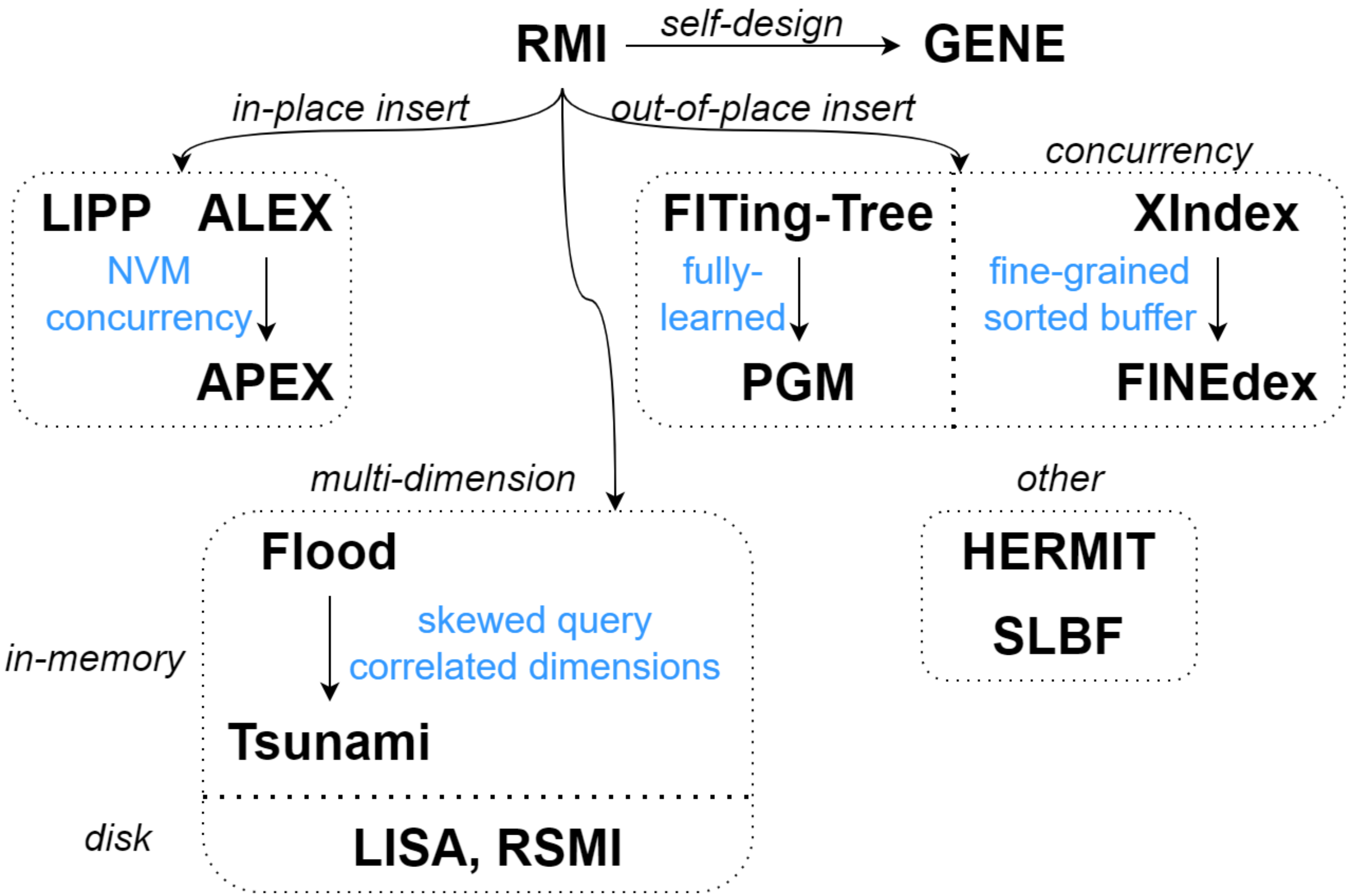
– Unaware of data and workload distribution

– Trade-off between space and access efficiency

Learned Index: Challanges

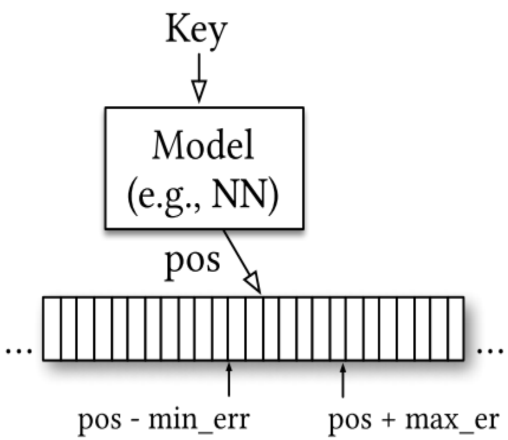
- **Advantages**
  - – Space efficient, only store several parameters
  - – Faster access if the model fit well, predict the position
- **Challenges**
  - – Support update, concurrency, and persistency
- **Optimization Goals**
  - – Higher throughput
  - – Less space
  - – Robustness

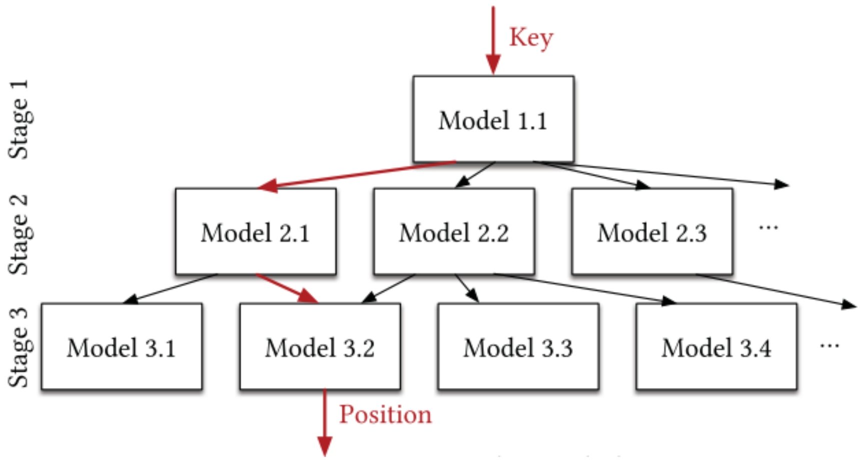
Learned Index: Lineage

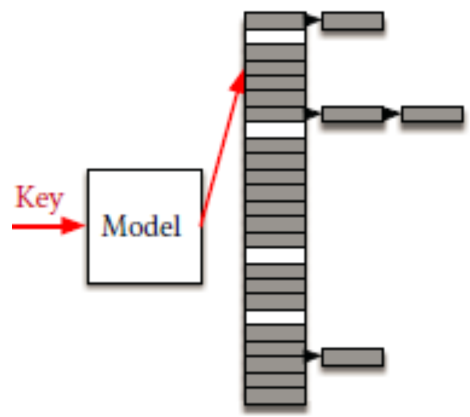


Learned Index: RMI

- **Motivation: indexes are models**
- **Challenge: difficult for the "last mile" to reduce error**
  - **Range index**: approximate location as $p = CDF(Key) * N$, model by hierarchy of simple neural networks, search precise location within error-bounded range
  - **Hash index**: CDF as hash function to reduce conflict



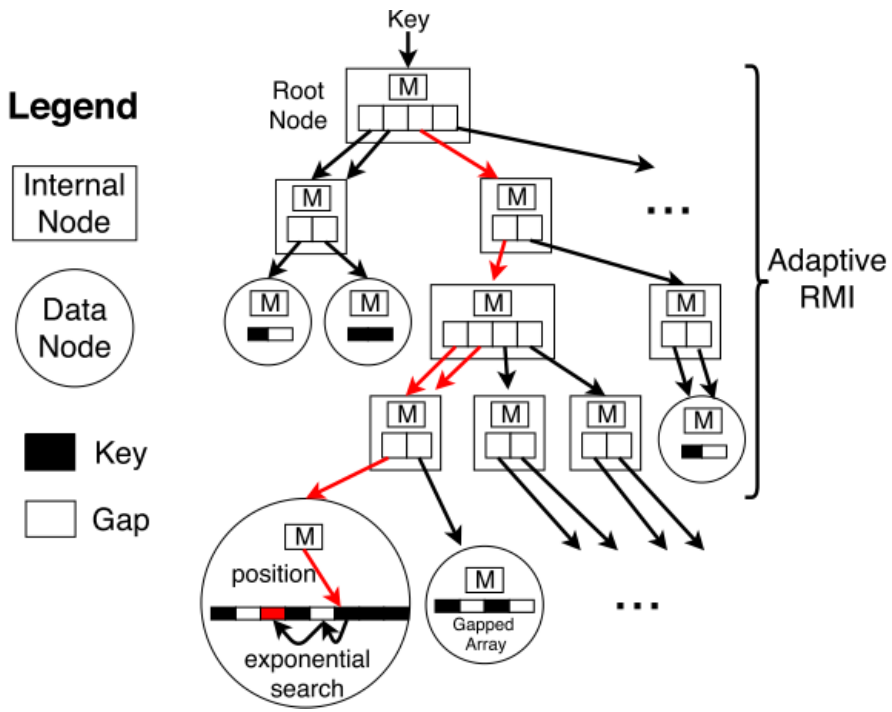




Kraska, T., Beutel, A., Chi, et al. The case for learned index structures. SIGMOD 2018

Updatable Learned Index: ALEX

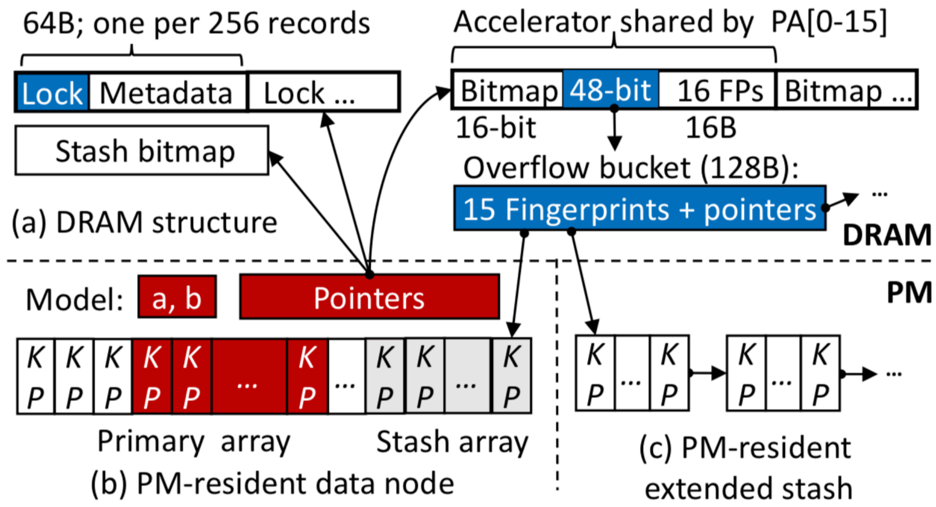
- **Motivation: support update**
- **Challenge: adaptive to dynamic data distribution**
  - Linear model, only exponential search in data nodes
  - Use **gapped array layout** in data nodes to accelerate insert
  - **Cost model**: predict latency of lookup and insert, expand/split data node if slower than a threshold (e.g. 1.5× that at creation)

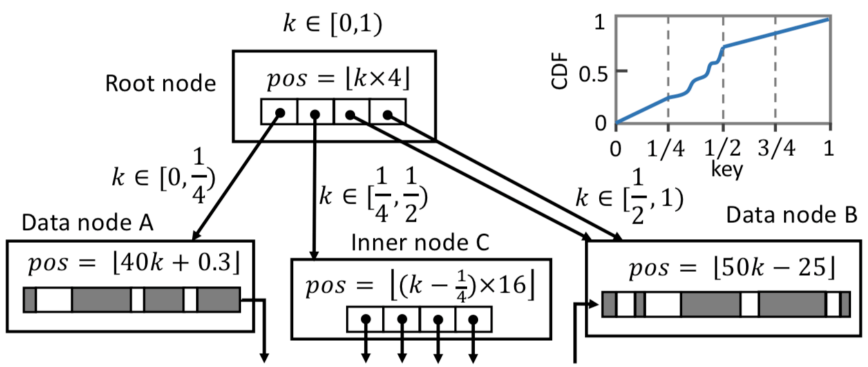


Ding, J., Minhas, U. F., Yu, J., et al. ALEX: An Updatable Adaptive Learned Index. SIGMOD 2020

Persistent Learned Index: APEX

- **Motivation: NVM-optimized ALEX**
- **Challenge: lower write bandwidth, crash consistency**
  - **Reduce write**: linear model in data node as hash function, collision solved by sequential scan and chaining
  - **Concurrency**: reader-writer lock for inner node, fine-grained optimistic lock for data nodes' non-structural update
  - **Crash recovery**: nodes out-of-place expand/split, undo-log before new node prepared, redo-log after

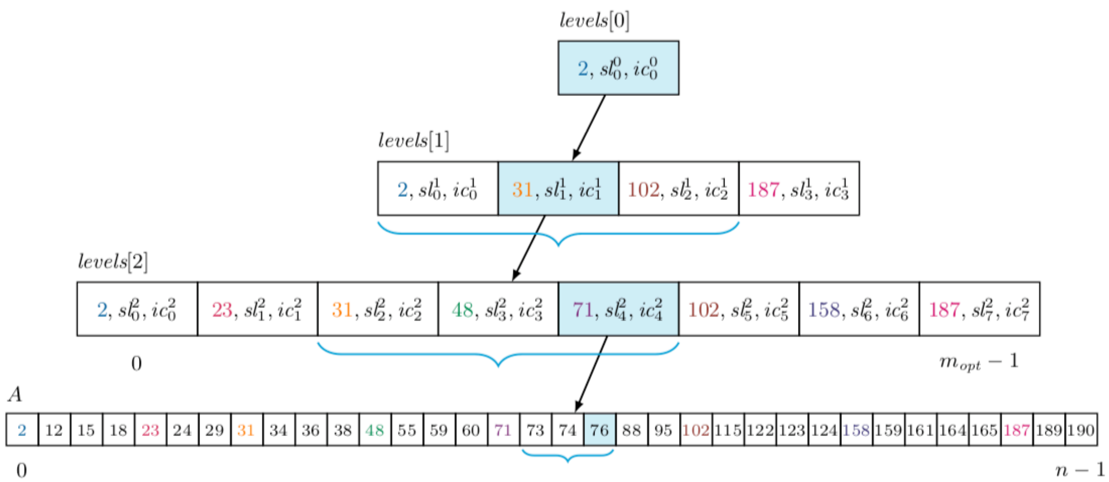




Lu, B., Ding, J., Lo, E., et al. APEX: A High-Performance Learned Index on Persistent Memory. VLDB 2022

Updatable Learned Index: PGM

- **Motivation: support update, fully-dynamic**
- **Challenge: adaptive to dynamic data distribution**
  - **Piecewise Geometric Model index (PGM-index)**
  - **I/O-optimally the predecessor search problem while taking succinct space**
  - **adaptive not only to the key distribution but also to the query distribution**



BUILD-PGM-INDEX(A, n, ε)

```
levels = an empty dynamic array
i = 0;  keys = A
repeat
    M = BUILD-PLA-MODEL(keys, ε)
    levels[i] = M;  i = i + 1
    m = SIZE(M)
    keys = [M[0].key, ..., M[m − 1].key]
until m = 1
return levels in reverse order
```

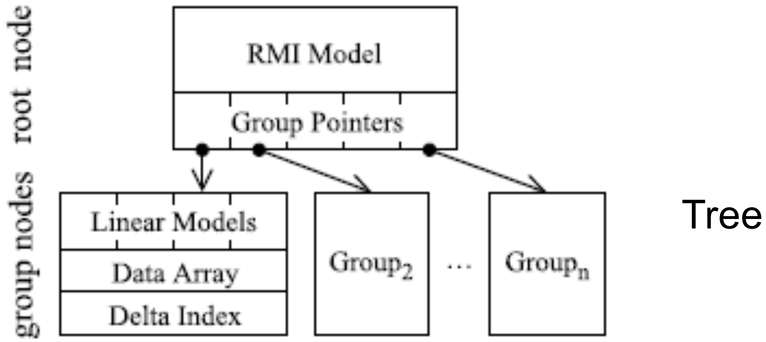
QUERY$(A, n, \varepsilon, levels, k)$

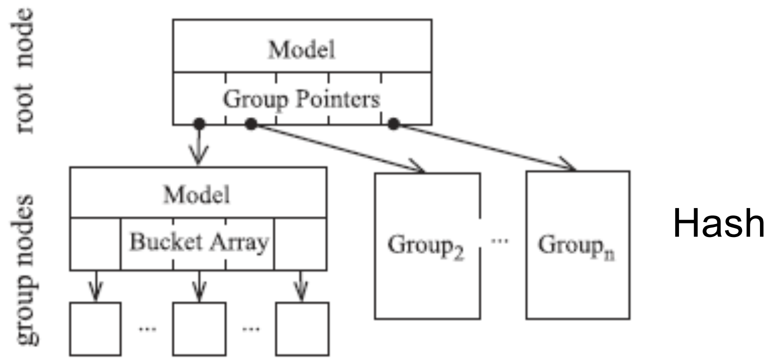
```
pos = f_r(k), where r = levels[0][0]
for i = 1 to SIZE(levels) − 1
    lo = max{pos − ε, 0}
    hi = min{pos + ε, SIZE(levels[i]) − 1}
    s = the rightmost segment s′ in
        levels[i][lo, hi] such that s′.key ≤ k
    t = the segment at the right of s
    pos = ⌊min{f_s(k), f_t(t.key)}⌋
lo = max{pos − ε, 0}
hi = min{pos + ε, n − 1}
return search for k in A[lo, hi]
```

FERRAGINA P, VINCIGUERRA G. The PGM-index: A fully-dynamic compressed learned index with provable worst-case bounds. VLDB 2020

Concurrent Learned Index: XIndex

- ☐ **Motivation: handle concurrent write**
- ☐ **Challenge: update in-place with a non-blocking scheme**
  - Two write types: in-place update, insert into buffer
  - **Two-phase compaction** to preserve effect of update:
    - first merge pointers to group's data and buffer
    - then copy the value
  - Similar design for the hash index, similar two-phase resize

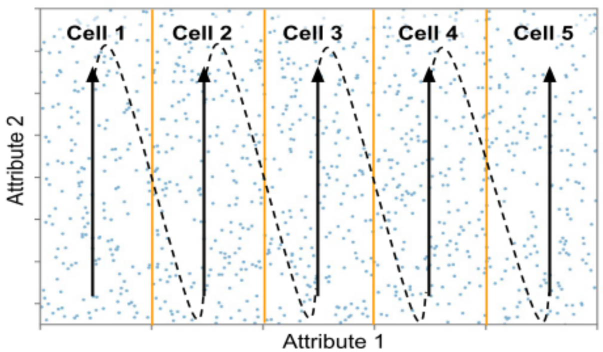


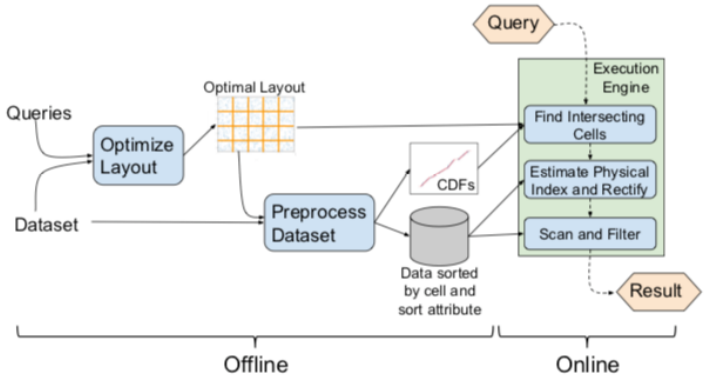


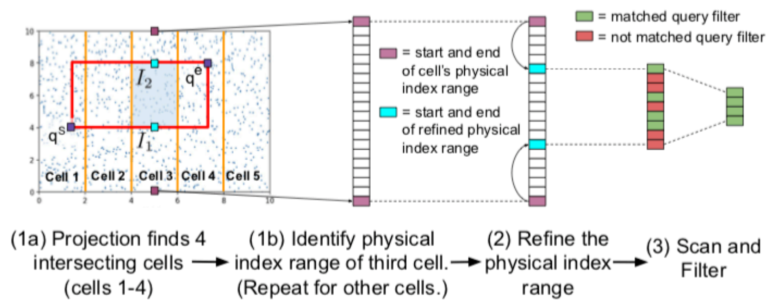
Wang, Z., Chen, H., Wang, Y., et al. The Concurrent Learned Indexes for Multicore Data Storage. ACM Transactions on Storage 2022

Multi-D Learned Index: Flood

- **Motivation: multi-dimensional in-memory read-optimized**
- **Challenge: optimize for data and query distribution**
  - Variant of grid index, cells sorted by 1st, 2nd, … column; within cell, points sorted by the last column
  - Gradient descent to find the optimal number of segments for each column using sample of dataset and workload
  - Use RMI to learn CDF of each column to even out segments and predict position



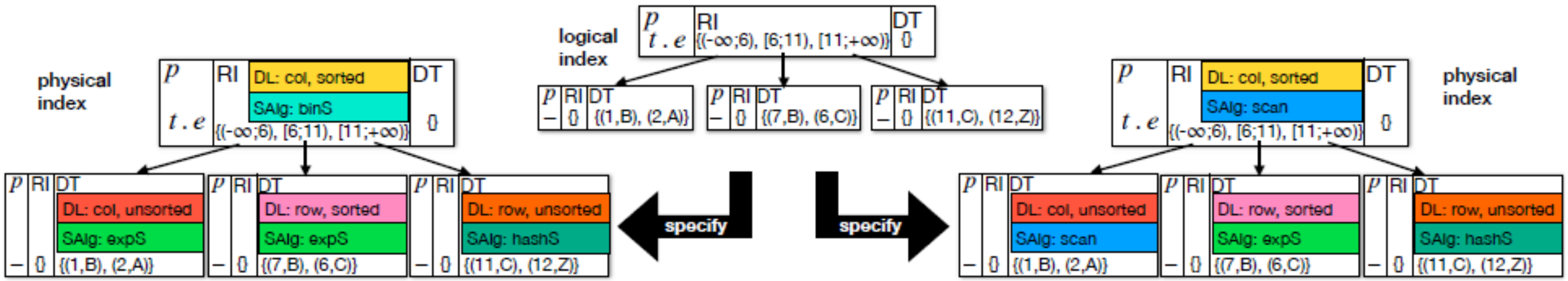




Nathan, V., Ding, J., Alizadeh, M., et al. Learning multi-dimensional indexes. SIGMOD 2020

Learned Index Generation: GENE

- **Motivation: self-design indexes**
- **Challenge: generalize to a genetic index framework**
- **Genetic Algorithm**
  - **Node framework**: child mapping, data, data layout and search method
  - **Population**: a set of indexes (e.g. initially a single node)
  - **Mutations**: change particular node's implementation, or merge/split nodes horizontally and vertically
  - **Fitness function**: optimize indexes for the runtime given workload



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

Learned Index: Comparison

Learned index	Model	Update	Concurrency	Persistency
RMI	simple NN	no	no	no
ALEX	linear	yes	no	no
Flood	simple NN	no	no	no
XIndex	simple NN, linear	yes	yes	no
APEX	linear	yes	yes	yes
GENE	any function	no	no	no

Learned Index: Take-away

- Though some research has already verified the benefit of learned index, performance in **industrial workloads** still needs to be studied, especially in **update distribution-drift** and **multi-dimension** situation.
- Open problems
  - Types of ML models to use
  - More efficiently support update, concurrency, persistency
  - Robustness: more adaptive to update distribution drift
  - Self-design: learn faster, or amortize learning cost
  - Make learned index applicable to industrial database systems

Learned Data Layout

Motivation

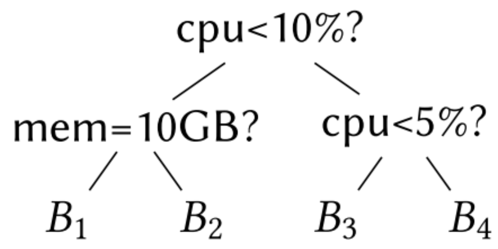
- **To reduce the #-data read from disk**
  - Split data into data blocks (main-memory, secondary storage)
  - in-memory min-max index for each block

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

    Traditional: assign by arrival time; hash/range parititon

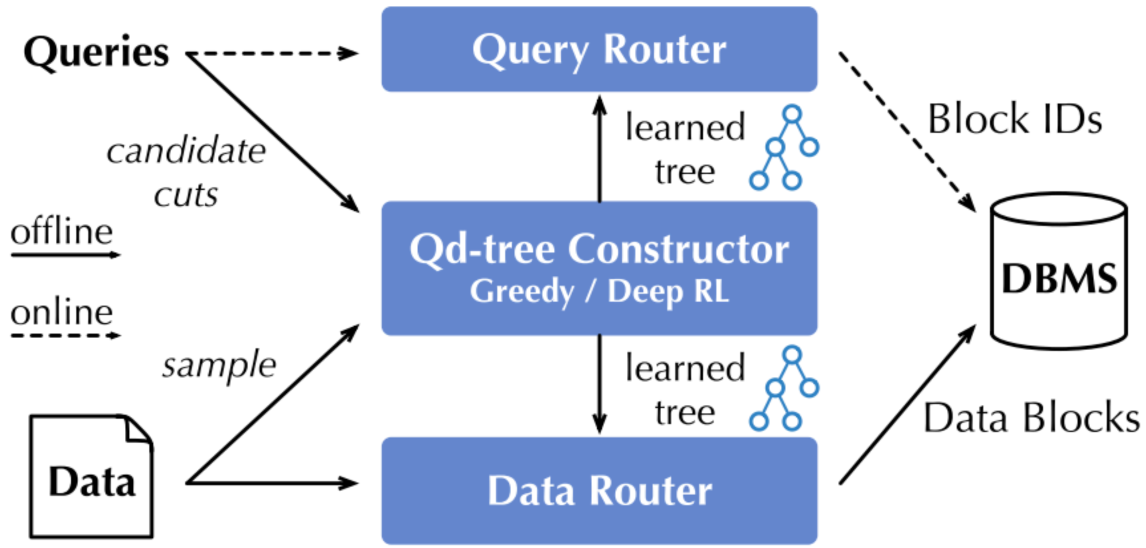
Learned Data Layout (Qd-tree)

- **Qd-tree: Learning 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.

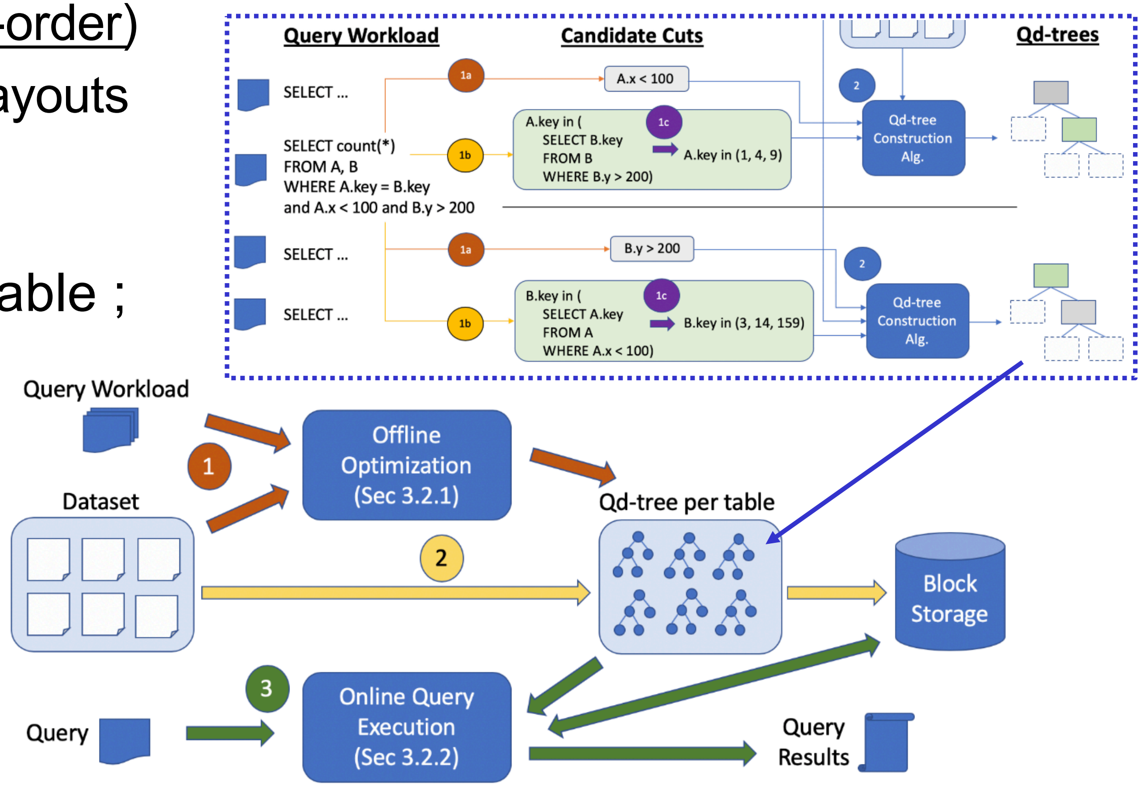
Learned Data Layout: Join Predicates

☐ Motivaiton

- ➢ **Traditonal:** either provide rare data skipping (<u>zone maps</u>), or require careful manual designs (<u>Z-order</u>)
- ➢ **Qd-tree:** only optimize singe-table layouts

☐ Qd-Trees for the whole datasets

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



Jialin Ding, et al. Iinstance-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
  - **Persistent**, **Update, Concurrency Control, Recovery**

Learned E2E System

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

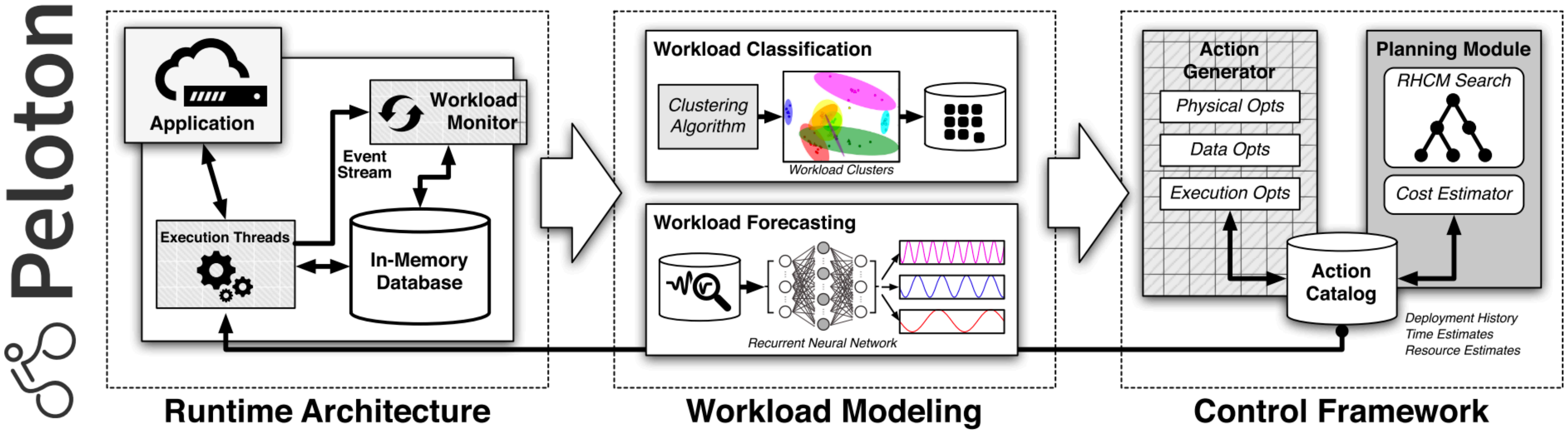
- ➢ Local optimum with high training overhead

□Commercial Practices of AI4DB Works

- ➢ Heavy ML models are hard to implement inside kernel
- ➢ A uniform training platform is required

Peloton

- **Schedule optimization actions via workload forecasting**
  - **Embedded Monitor:** Detect the event stream
  - **Workload Forecast Model:** Future workload type
  - **Optimization Actions:** Tuning, Planning



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

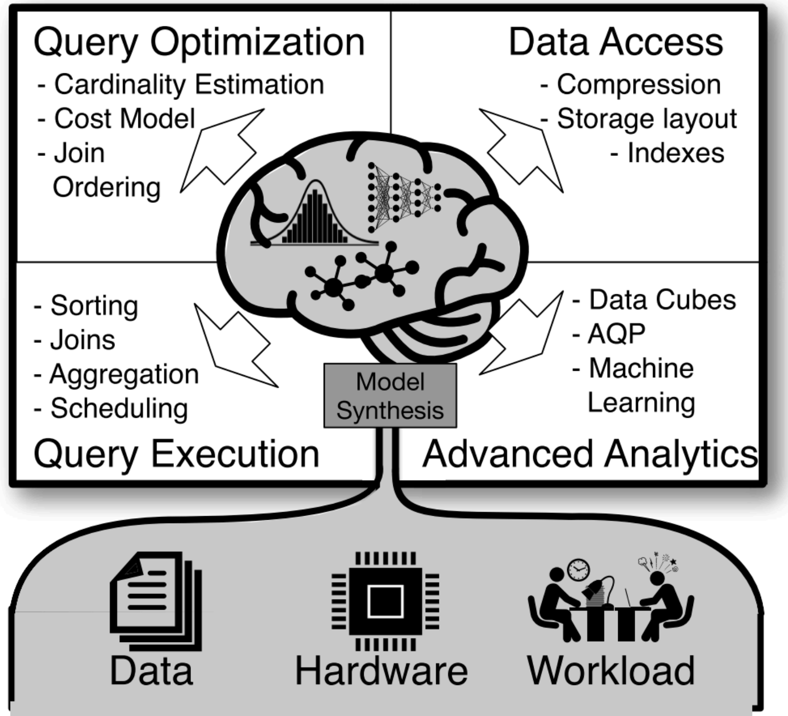
SageDB

□ 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 based on the learned CDFs
  - ➢ Query optimization and execution
  - ➢ Data layout design
  - ➢ Advanced analytics

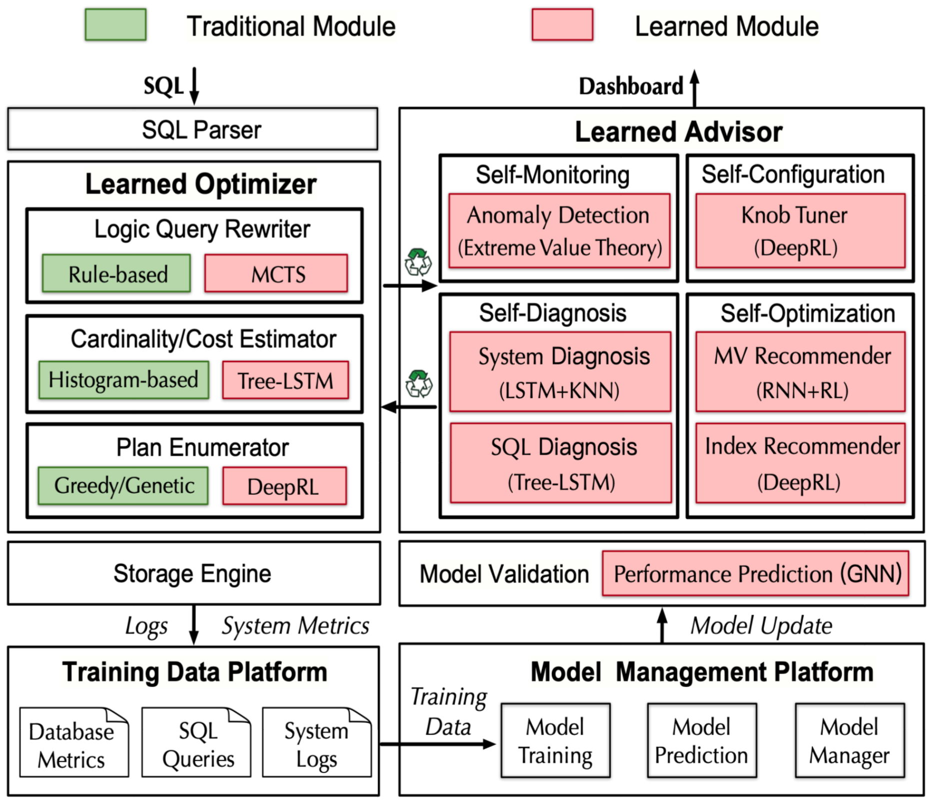


Tim Kraska, et al. SageDB: A Learned Database System. In CIDR, 2019.

openGauss

☐ Implement, validate, and manage learning-based modules

- **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.

Open Problems

Future Works: Adaptability

- **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: 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: 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: Validate Learning-based Models

Model Validation

- Whether a model is effective?
- Whether a model outperforms existing ones?
- Whether a model can adapt to new scenarios?

Future Works: Complex Scenarios

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

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

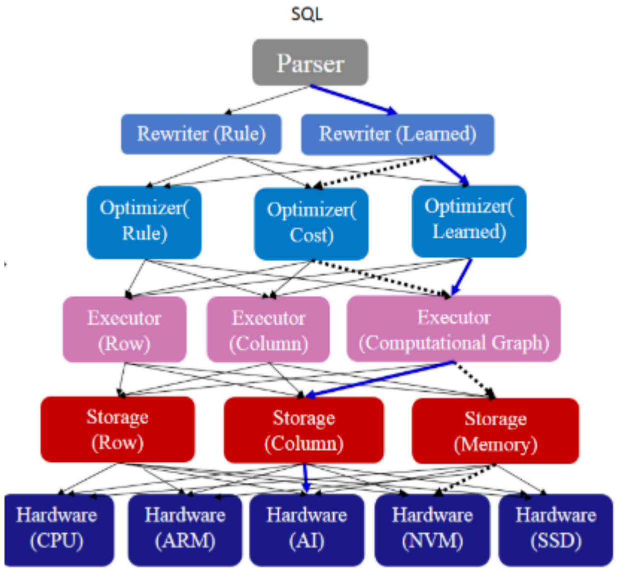
Future Works: One Model Fits Various Scenarios

- **High Adaptability**
  - **Workloads:** query operators; plan structures; underlying data access
  - **Datasets:** tables; columns; data distribution; indexes / views; data updates
  - **DB Instances:** state metrics (DB, resource utilization): hardware configurations
  - **DBMSs:** MySQL; PostgreSQL; MongoDB; Spark
- **Possible Solutions: common knowledge extraction; meta learning**

Future Works: Automatic Learned Model Selection

Automatic Database Assembling

- Automatically select ML models/algorithms for different tasks
- Evaluate the overall performance



Database Assembling

Category	Method
Supervised Learning	Linear Regression Logistic Regression Decision Tree Deep Learning
Unsupervised Learning	K-Means Clustering Association Rules Reinforcement Learning
Descriptive Statistics	Count-Min Sketch Data Profiling

The Stack of ML Algorithms

Future Works: Unified Database Optimization

◻ Arrange Multiple Database Optimization Tasks

- ◻ **Multiple Requirements:** (1) Optimizer can produce good plans with not very accurate estimator; (2) Creating indexes may incur the change of optimal knobs
- ◻ **Hybrid Scheduling:** Arrange different optimization tasks based on the database configuration and workload characters
- ◻ **Optimization Overhead:** Achieve maximum optimization without competing resources with user processes

✓ **Challenges:** various task features; correlations between tasks; trend changes

Thanks

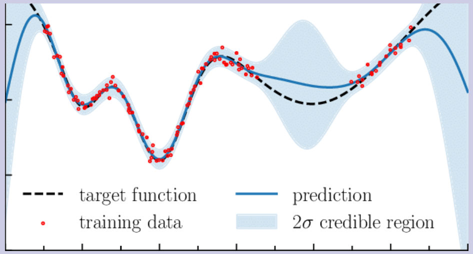
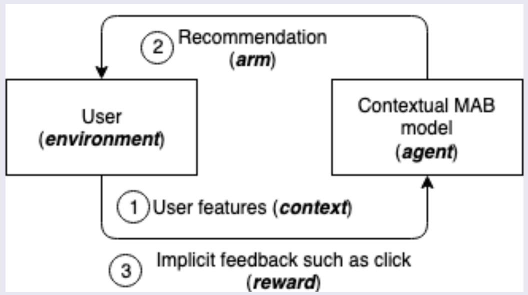
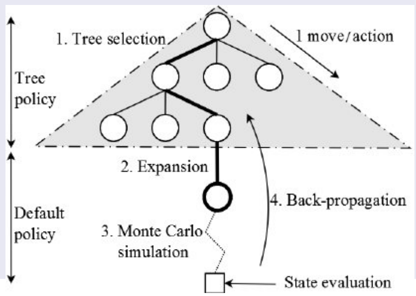
Machine Learing for Databases

Empirical Methods	Heuristic Search/Rules	e.g., knob tuning
	Dynamic Programming	e.g., Index Selection
	Maximum Spanning Tree	e.g., Database Partition
Supervised ML	Gaussian Process	e.g., knob tuning
	Bayesian Optimization	e.g., knob tuning
	CNN	e.g., card Estimation
	Tree-based Ensemble	e.g., card Estimation
	Kernel Density Estimation	e.g., card Estimation
	Uniform Mixture Model	e.g., card Estimation
	Causal Model	e.g., System Diagnosis
	Clustering Algorithns	e.g., System Diagnosis
	Annotated Plan Graph	e.g., System Diagnosis
	Dense Neural Network	e.g., knob tuning, view selection, index selection, lesarned Index, transactions, query latency prediction
	Encoder-Decoder	e.g., view selection
	Tree-LSTM	e.g., Cost estimation, plan enumerator
	Graph Neural Network	e.g., workload performance prediction
Unsupervised ML	AutoRegressive	e.g., card Estimation
	Sum-Product Network	e.g., card estimation
Semi-supervised ML	Meta Learning	e.g., knob tuning
	Pre-Training Network	e.g., query encoding
(Deep) Reinforcement Learning	DDPG	e.g., knob tuning
	DQN	e.g., view selection, index selection, plan enumerator
	Q-learning	e.g., view Selection , database partition, transactions
	MCTS	e.g., plan enumerator

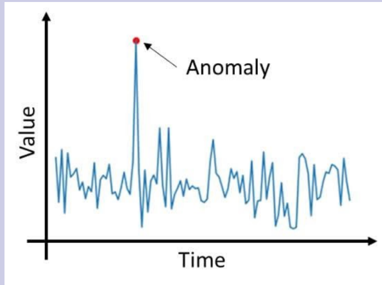
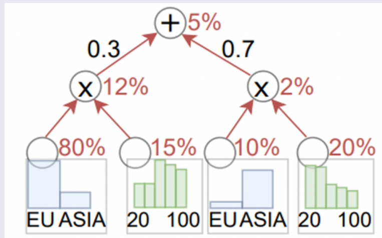
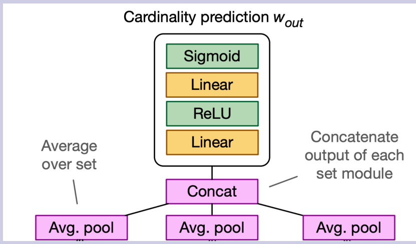
Summarization of AI4DB Techniques

	Database Problem	Method	Performance	Overhead	Training Data	Adaptivity
Offline NP Problem	knob space exploration	gradient-based [1, 18, 47]	High	High	High	–
		dense network [37]	Medium	High/Medium	High	– / instance
		DDPG [23, 46]	High	High	Low/Medium	query
	index selection	q-learning [19]	–	High	Low	–
	view selection	q-learning [43]	Medium	High	Low	–
		DDQN [9]	High	High	Low	query
	partition-key selection	q-learning [11]	–	High	Low	–
Online NP Problem	join order selection	q-learning [27]	High	High	Low	–
		DQN [26, 42]	High	High	Low	query
		MCTS [38]	Medium	Low	Low	instance
	query rewrite	MCTS [21, 49]	–	Low	Low	query
Regression Problem	cost estimation	tree-LSTM [35]	High	High	High	query
	cardinality estimation	tree-ensemble [7]	Medium	Medium	High	query
		autoregressive [41]	High	High/Medium	Low	data
		dense network [16]	High	High	High	query
		sum-product [12]	Medium	High	Low	data
	index benefit estimation	dense network [5]	–	High	High	query
	view benefit estimation	dense network [9]	–	High	High	query
	latency prediction	dense network [28]	Medium	High	High	query
		graph embedding [50]	High	High	High	instance
	learned index	dense network [3]	–	High	High	query
Prediction Problem	trend prediction	clustering-based [24]	–	Medium	Medium	instance
	transaction scheduling	q-learning [44]	–	High	Low	query

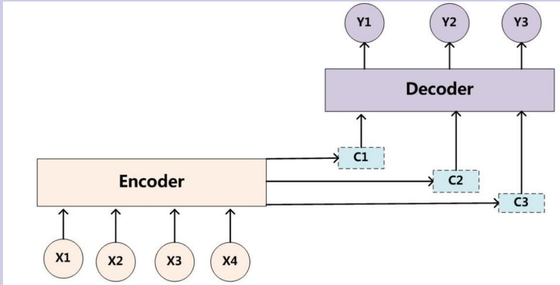
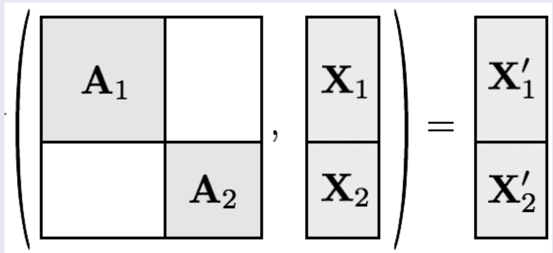
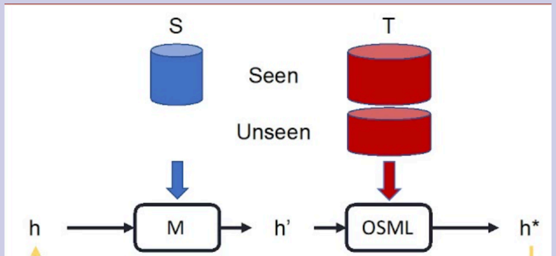
ML Models for Optimization Problems

ML Method	Description	Example	DB Tasks
Gradient-based Methods	Approximate the data distribution with gaussian functions, and select the optimal point by the guidance of gradients		Knob Tuning; Cardinality Estimation
Contextual Multi-armed Bandit	Maximize the reward by repeatedly selecting from a fixed number of arms		Plan Hint; Knob Tuning; MV Selection; Index Selection; Database Partition; Join Order Selection; Workload Schedule
Deep Reinforcement Learning	Learn the selection (actor) or estimation (critic) policy with neural networks		
Monte Carlo Tree Search	Repeated iterations of four steps (selection, expansion, simulation, back-propagation) until termination		Query Rewrite; Online Join Order Selection

ML Models for Regression Problems

ML Method	Description	Example	DB Tasks
Statistical ML	Build a regression model to approximate real distribution based on sampled data		Cardinality Estimation; Trend Prediction
Sum-Product Network	Learn distributions with Sum for different filters and Product for different joins		Cardinality Estimation
Deep Learning (e.g., DNN, CNN, RNN)	Learn the mapping relations from the input features to the targets by graident descent		Knob Tuning; Cardinality Estimation; Cost Estimation

ML Models for Others

ML Method	Description	Example	DB Tasks
Generative Model (e.g., Encoder-Decoder)	Encode varied-length input features into fixed-length vector with mechanisms like multi-head attention		MV Selection
Graph Convolutional Network	Encode graph-structured input features with convolutions on the vertex features and their K-hop neighbor vertices		Query Latency Prediction
Meta Learning	Use the base models to form the target model based on the task similarity and the prediction accuracy during usage		Knob Tuning

Classical ML Methods

- **Techniques**
  - Gradient methods (e.g., GP); Regression methods (e.g., tree-ensembling, kernel-density estimation)
- **Advantages**
  - Lightweight; Easier to interpret than DL
- **Disadvantages**
  - Hard to extend to large data; Complex feature engineering
- **ML4DB Applications**
  - Knob Tuning; Cardinality Estimation

Classical ML Methods: Challenges

- How to apply to a new problem?
  - Problem Modelling: As a regression or gradient-based optimization problems
  - Feature Engineering: Determine the input with feature engineering techniques
  - Model Construction: Select proper classic ML models, collect sample data, and learn the mapping relations
  - Additional Requirements: Reuse classic ML models in limited scenarios (e.g., similar workloads)

Classical ML Methods

	Feature Engineering	Model Selection
Knob Tuning	• Reduce the knob space with linear regression like Lasso; • Reduce redundant metrics with factor analysis and clustering like k-means;	• Gaussian Process: Search local-optimal settings within the selected knob space • Reuse the historical data by matching workloads by their metric values
Cardinality Estimation	• Assumptions like column independency or linear relations between columns • Determine supported queries like range queries	• Query-based: Define input space as conjunction of the query ranges on data columns (Tree-Ensemble) • Data-based: Partition data into indpendent regions (Sum-Product) or learn column correlations (AR)

Reinforcement Learning Methods

- **Techniques**
  - Model-based (e.g.,, MCTS+DL);
  - Model-free (e.g., value-based like Q-learning, policy-based like DDPG)
- **Advantages**
  - High performance on large search space; No prepared data
- **Disadvantages**
  - Long exploration time; Hard to migration to new scenarios
- **ML4DB Applications**
  - Knob Tuning, View/Index/Partition-key Selection, Optimizer, Workload Scheduling

Reinforcement Learning Methods: Challenges

- ☐ **How to apply to a new problem?**
  - ☐ **Problem Modelling: Map to the 6 factors in a RL model (state, action, reward, policy, agent, environment)**
  - ☐ **Feature Characterization: Select target-related features as the state of the RL problem**
  - ☐ **Model Construction: Select proper RL models (e.g., MCTS, DQN, DDPG), design the networks and the reward function**
  - ☐ **Additional Requirements: E.g., encode the query costs with Deep Learning; encode the join relations with GNN**

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
Knob Tuning	• Knobs Values • Innter Metrics • Workloads	• DDPG for both continuous state and continuous actions	• Performance improvements over last tuning action • Performance improvements over first tuning action	• Design a dense network as the estimation (critic) model

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
View Selection	• Candidate Views • Built Views • Workload	• DQN for continuous state and discrete actions	• Utility increase on creating the views	• Encoder-decoder for inputs; Nonlinear layers for utility estimation
Index Selection	• Candidate Indexes • Built indexes • Workload		• Utility increase on creating the indexes	• Design a dense network as the estimation model
Partiton-key Selection	• Columns • Tables • Query templates		• Estimated costs beofore/after partitioning	• Design a dense network as the estimation model

Reinforcement Learning Methods

	Input Features	RL Method	Reward Design	Estimation Model
Query Rewrite	• Logical Query • Rewrite Rules • Table Schema	• MCTS for tree search	• Utility increase for future optimal queries	• Multi-head attention for rules, query, data
Join Order Selection	• Physical Plan • Candidate Joins • Table Schema	• DQN for continuous state and discrete actions	• Saved costs	• Design a dense network as the estimation model
Plan Hinter	• Physical Plan • Hint Sets	• Contextual Multi-armed for limited actions	• Saved costs	• Traditional Optimizer

Deep Learning Methods

- **Techniques**
  - Dense Layer ((non)-linear); Convolutional Layer; Graph Embedding Layer; Recurrent Layer
- **Advantages**
  - Approximate the high-dimension relations
- **Disadvantages**
  - Data-consuming
- **ML4DB Applications**
  - Cost Estimation; Benefit Estimation; Latency Estimation

Deep Learning Methods: Challenges

- How to apply to a new problem?
  - Input Features: Select features that affect the estimation targets (e.g., latency, utility)
  - Encoding Strategy: Encode based on the feature structures (e.g., Graph embedding for query relations)
  - Model Design: Design the network structures (e.g., layers, activation functions, loss functions) based on the input embedding (e.g., fixed-length or varied-length)

Deep Learning Methods

	Input Features	Feature Encoding	Model Design
Cost Estimation	• Physical Plan	• Encode operators with LSTM	• Plan-structured Neural Network
Benefit Estimation	• Physical Plan • Optimization Actions (e.g., views. indexes)	• Encode actions like Encoder-Decoder for Views and linear layer for Indexes	• Design a dense network as the estimation model
Latency Estimation	• Physical Plan • Query Relations • DB State	• Encoder query correlations with graph covolutions	• Design a K-layer graph embedding network for K-hop neighbors