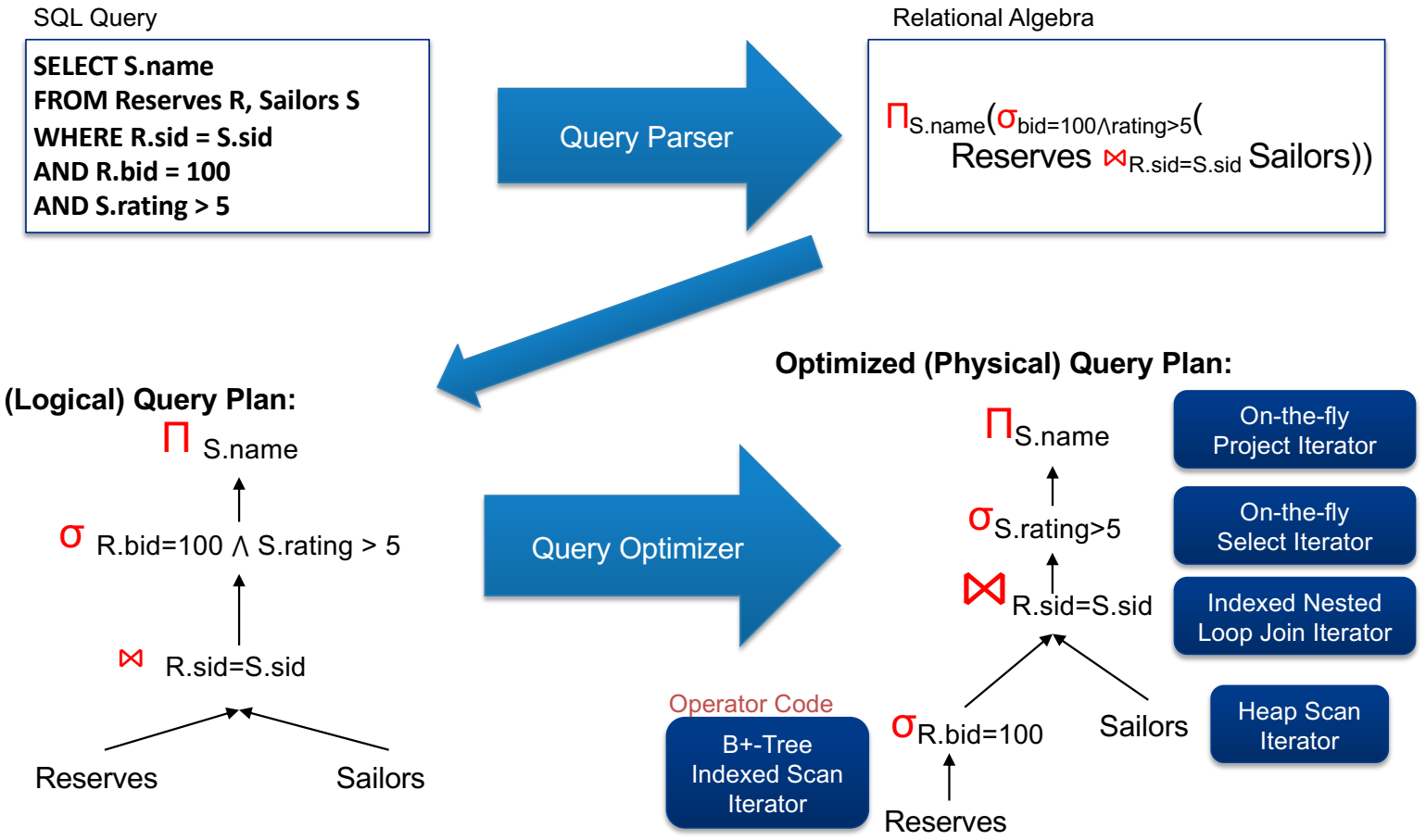
Learning-based Optimizer

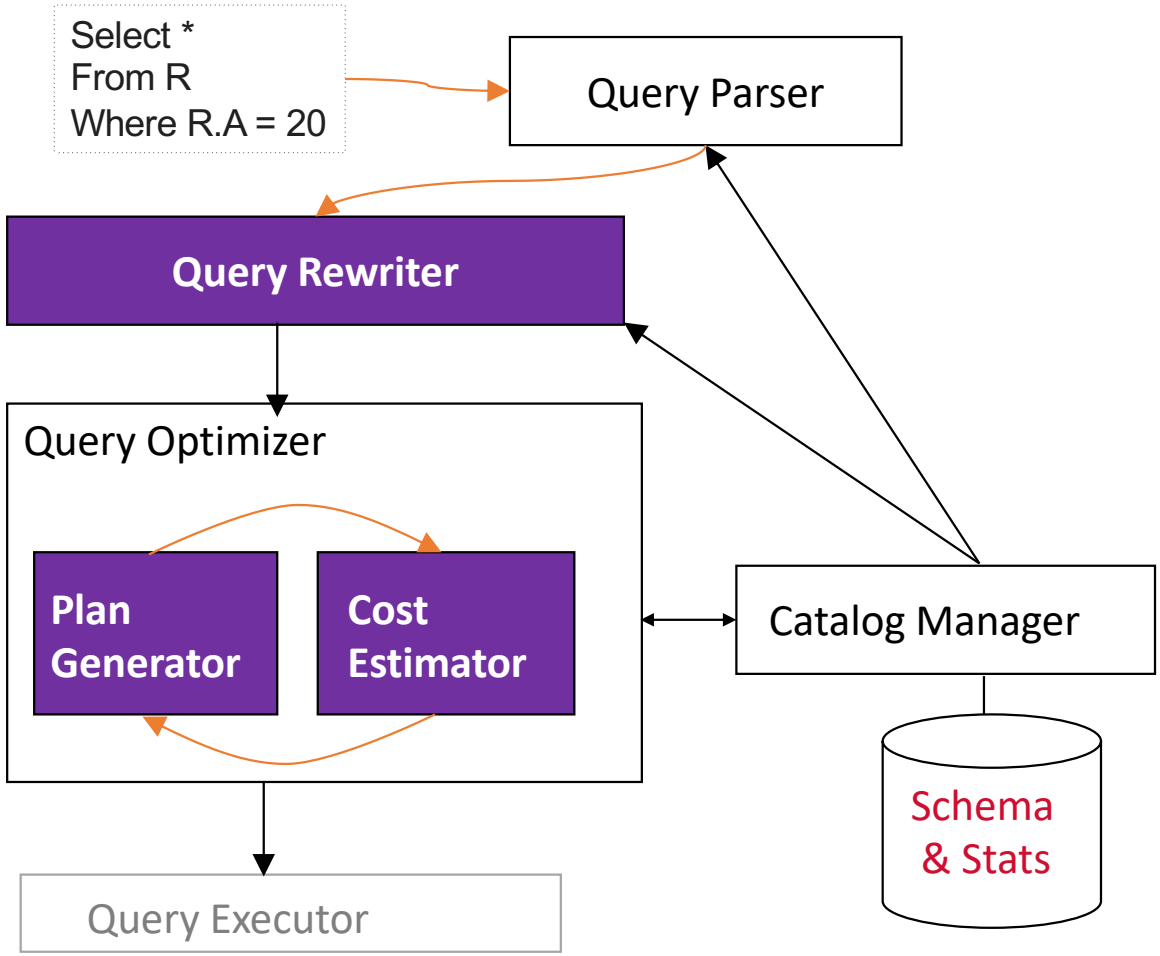
Guoliang Li

Tsinghua University

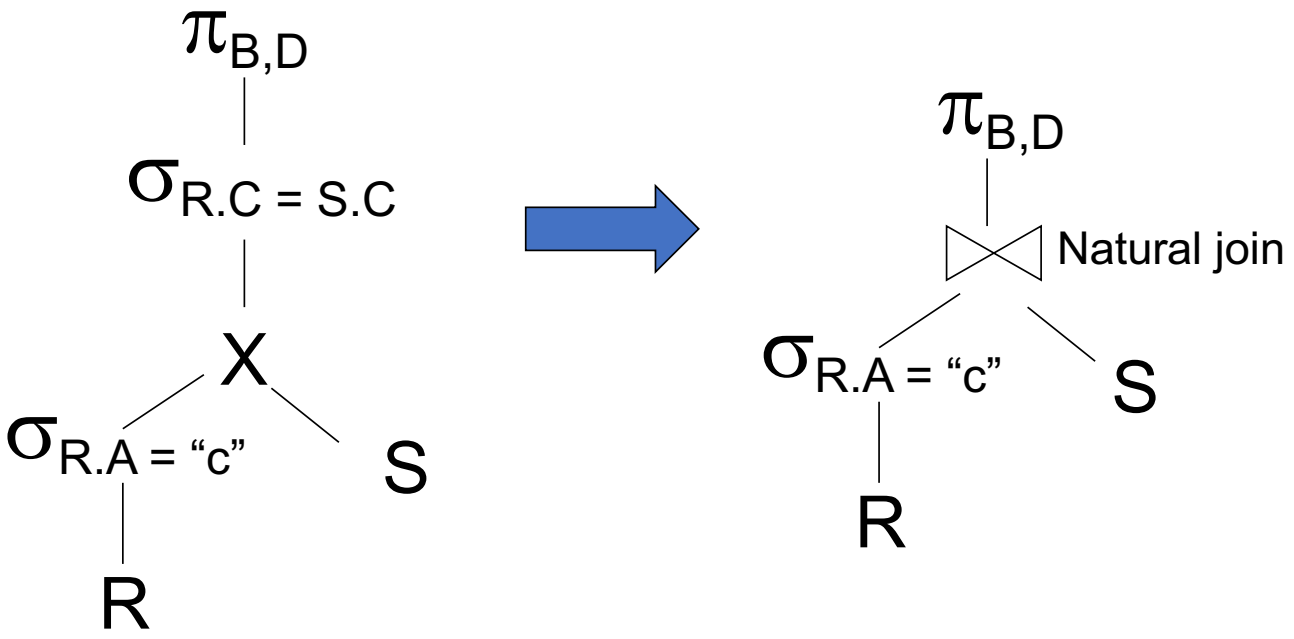
Query Optimizer



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

$$\sigma_{p_1 \wedge p_2}(e) \equiv \sigma_{p_1}(\sigma_{p_2}(e)) \quad (1)$$

$$\sigma_{p_1}(\sigma_{p_2}(e)) \equiv \sigma_{p_2}(\sigma_{p_1}(e)) \quad (2)$$

$$\Pi_{A_1}(\Pi_{A_2}(e)) \equiv \Pi_{A_1}(e) \quad (3)$$

$$\text{if } A_1 \subseteq A_2$$

$$\sigma_p(\Pi_A(e)) \equiv \Pi_A(\sigma_p(e)) \quad (4)$$

$$\text{if } \mathcal{F}(p) \subseteq A$$

$$\sigma_p(e_1 \cup e_2) \equiv \sigma_p(e_1) \cup \sigma_p(e_2) \quad (5)$$

$$\sigma_p(e_1 \cap e_2) \equiv \sigma_p(e_1) \cap \sigma_p(e_2) \quad (6)$$

$$\sigma_p(e_1 \setminus e_2) \equiv \sigma_p(e_1) \setminus \sigma_p(e_2) \quad (7)$$

$$\Pi_A(e_1 \cup e_2) \equiv \Pi_A(e_1) \cup \Pi_A(e_2) \quad (8)$$

Query Rewrite Rules

$$e_1 \times e_2 \equiv e_2 \times e_1 \tag{9}$$

$$e_1 \bowtie_p e_2 \equiv e_2 \bowtie_p e_1 \tag{10}$$

$$(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3) \tag{11}$$

$$(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 \equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3) \tag{12}$$

$$\sigma_p(e_1 \times e_2) \equiv e_1 \bowtie_p e_2 \tag{13}$$

$$\sigma_p(e_1 \times e_2) \equiv \sigma_p(e_1) \times e_2 \quad \text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1) \tag{14}$$

$$\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) \equiv \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 \quad \text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1) \tag{15}$$

$$\Pi_A(e_1 \times e_2) \equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) \quad \text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2) \tag{16}$$

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) →)

$$\sigma_{p_1 \wedge p_2}(e) \equiv \sigma_{p_1}(\sigma_{p_2}(e)) \quad (1)$$

$$\sigma_{p_1}(\sigma_{p_2}(e)) \equiv \sigma_{p_2}(\sigma_{p_1}(e)) \quad (2)$$

$$\Pi_{A_1}(\Pi_{A_2}(e)) \equiv \Pi_{A_1}(e) \quad (3)$$
$$\text{if } A_1 \subseteq A_2$$

$$\sigma_p(\Pi_A(e)) \equiv \Pi_A(\sigma_p(e)) \quad (4)$$
$$\text{if } \mathcal{F}(p) \subseteq A$$

$$\sigma_p(e_1 \cup e_2) \equiv \sigma_p(e_1) \cup \sigma_p(e_2) \quad (5)$$

$$\sigma_p(e_1 \cap e_2) \equiv \sigma_p(e_1) \cap \sigma_p(e_2) \quad (6)$$

$$\sigma_p(e_1 \setminus e_2) \equiv \sigma_p(e_1) \setminus \sigma_p(e_2) \quad (7)$$

$$\Pi_A(e_1 \cup e_2) \equiv \Pi_A(e_1) \cup \Pi_A(e_2) \quad (8)$$

$$e_1 \times e_2 \equiv e_2 \times e_1 \quad (9)$$

$$e_1 \bowtie_p e_2 \equiv e_2 \bowtie_p e_1 \quad (10)$$

$$(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3) \quad (11)$$

$$(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 \equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3) \quad (12)$$

$$\sigma_p(e_1 \times e_2) \equiv e_1 \bowtie_p e_2 \quad (13)$$

$$\sigma_p(e_1 \times e_2) \equiv \sigma_p(e_1) \times e_2 \quad (14)$$
$$\text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1)$$

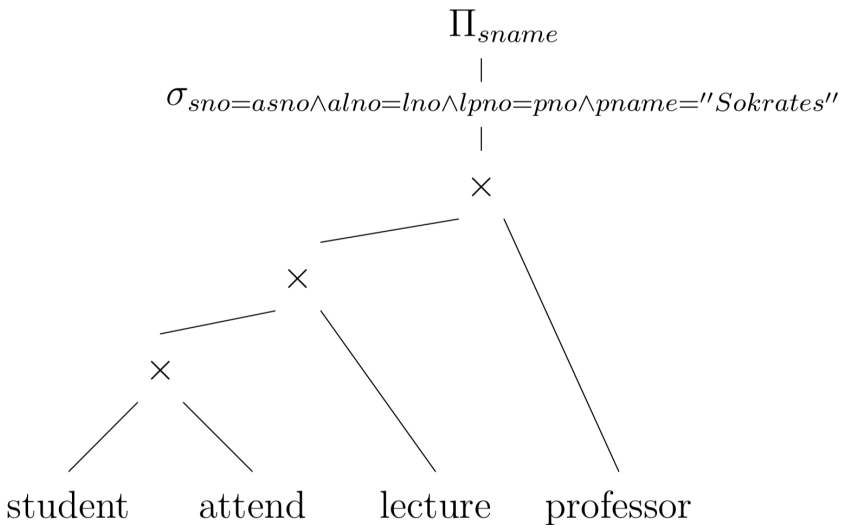
$$\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) \equiv \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 \quad (15)$$
$$\text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1)$$

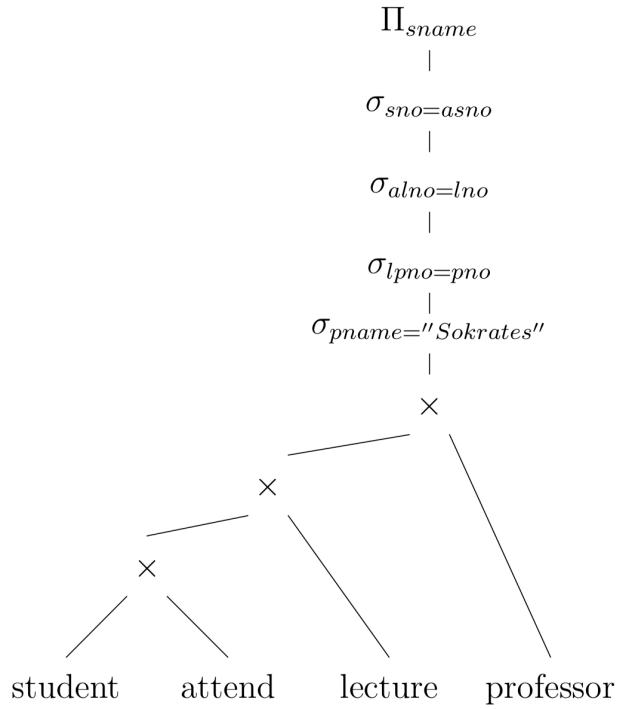
$$\Pi_A(e_1 \times e_2) \equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) \quad (16)$$
$$\text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2)$$

Step 1: Break up conjunctive selection predicates

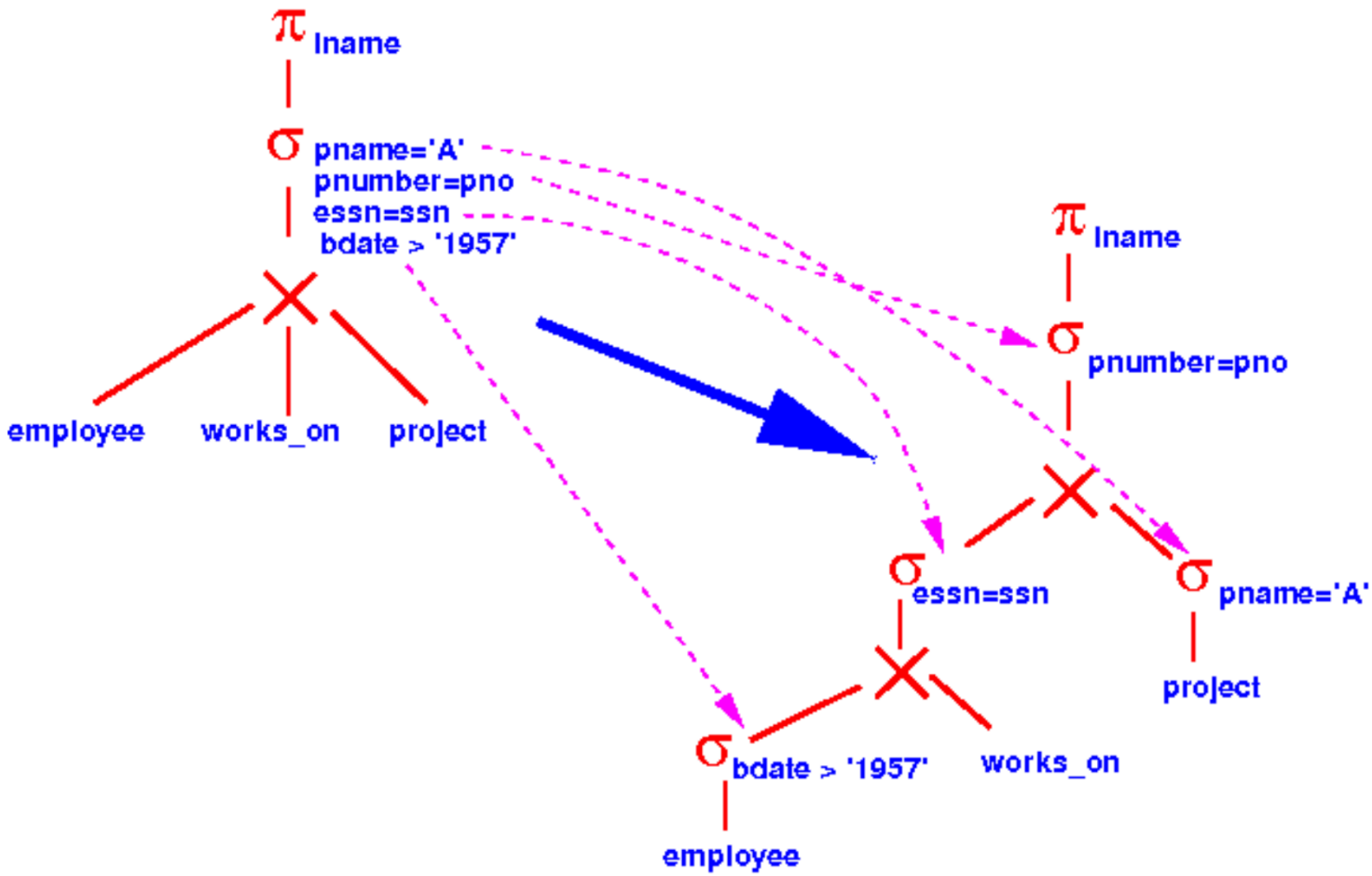
select distinct s.sname
from student s, attend a, lecture l, professor p
where s.sno = a.asno and a.alno = l.lno and l.lpno = p.pno and p.pname = "Sokrates"

selection with simple predicates can be moved around easier



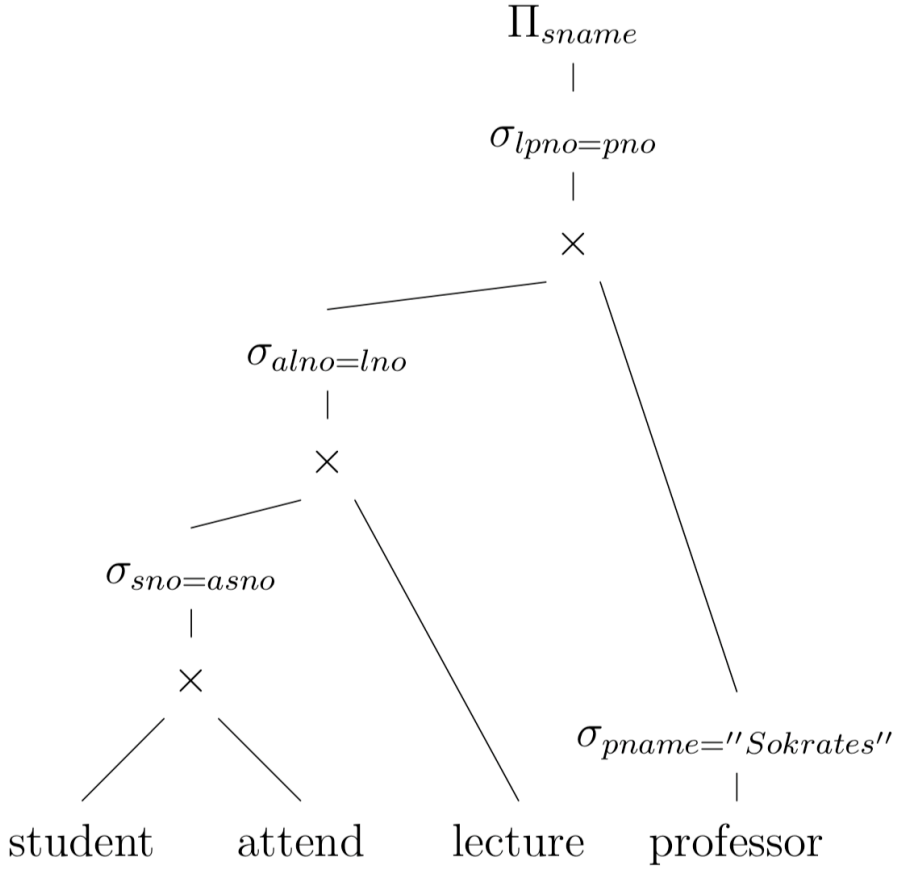


Step 1: Break up conjunctive selection predicates



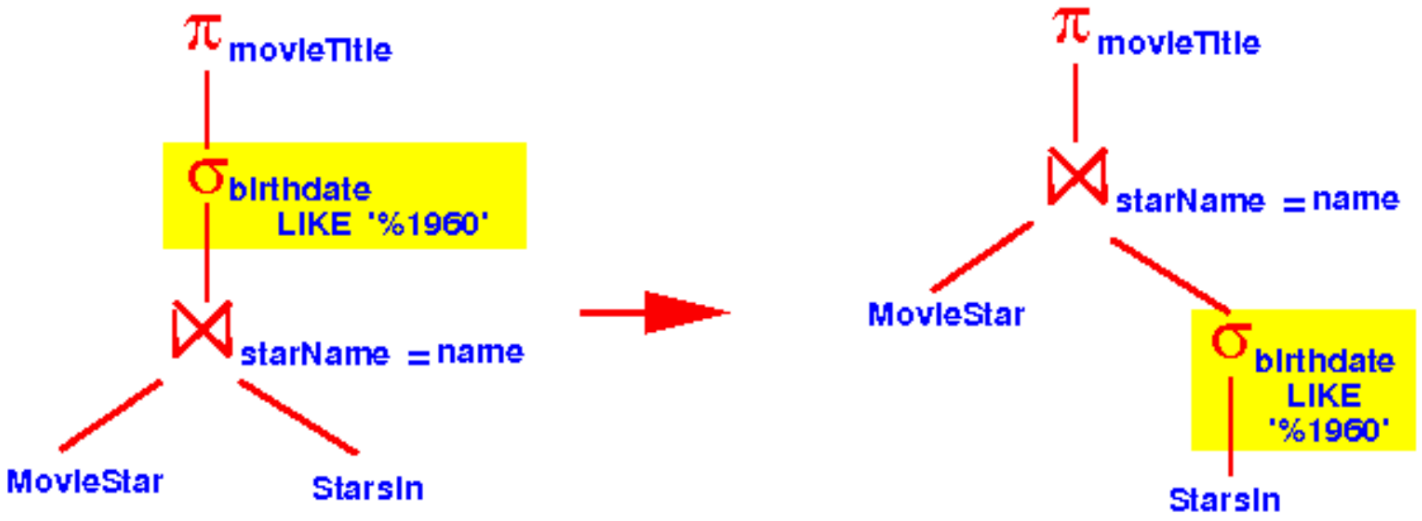
Step 2: Push Selections Down

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



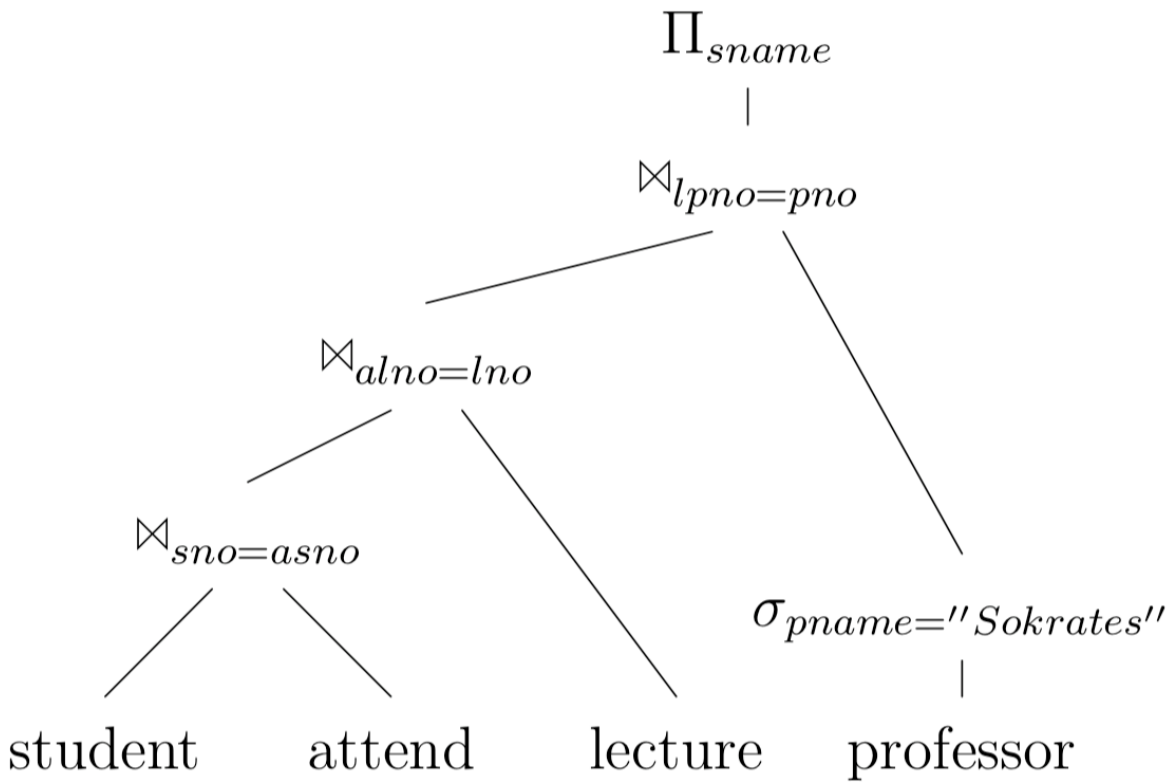
Step 2: Push Selections Down

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



Step 3: Introduce Joins

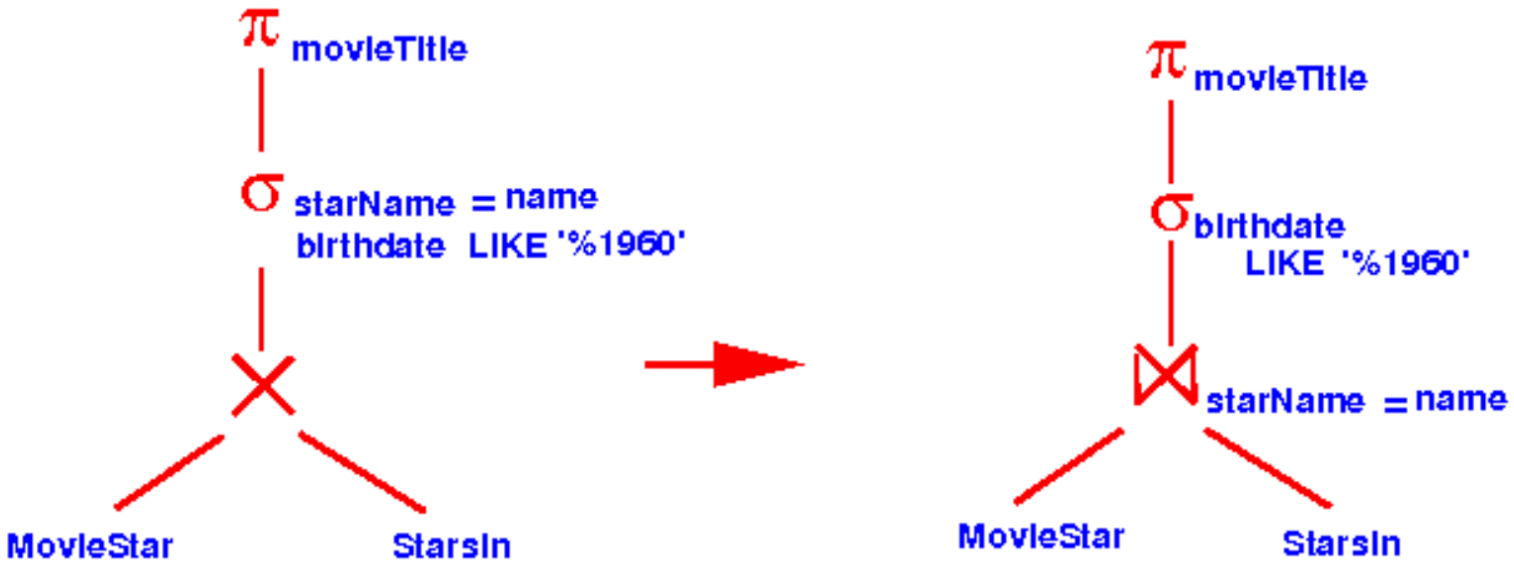
joins are cheaper than cross products



Step 3: Introduce Joins

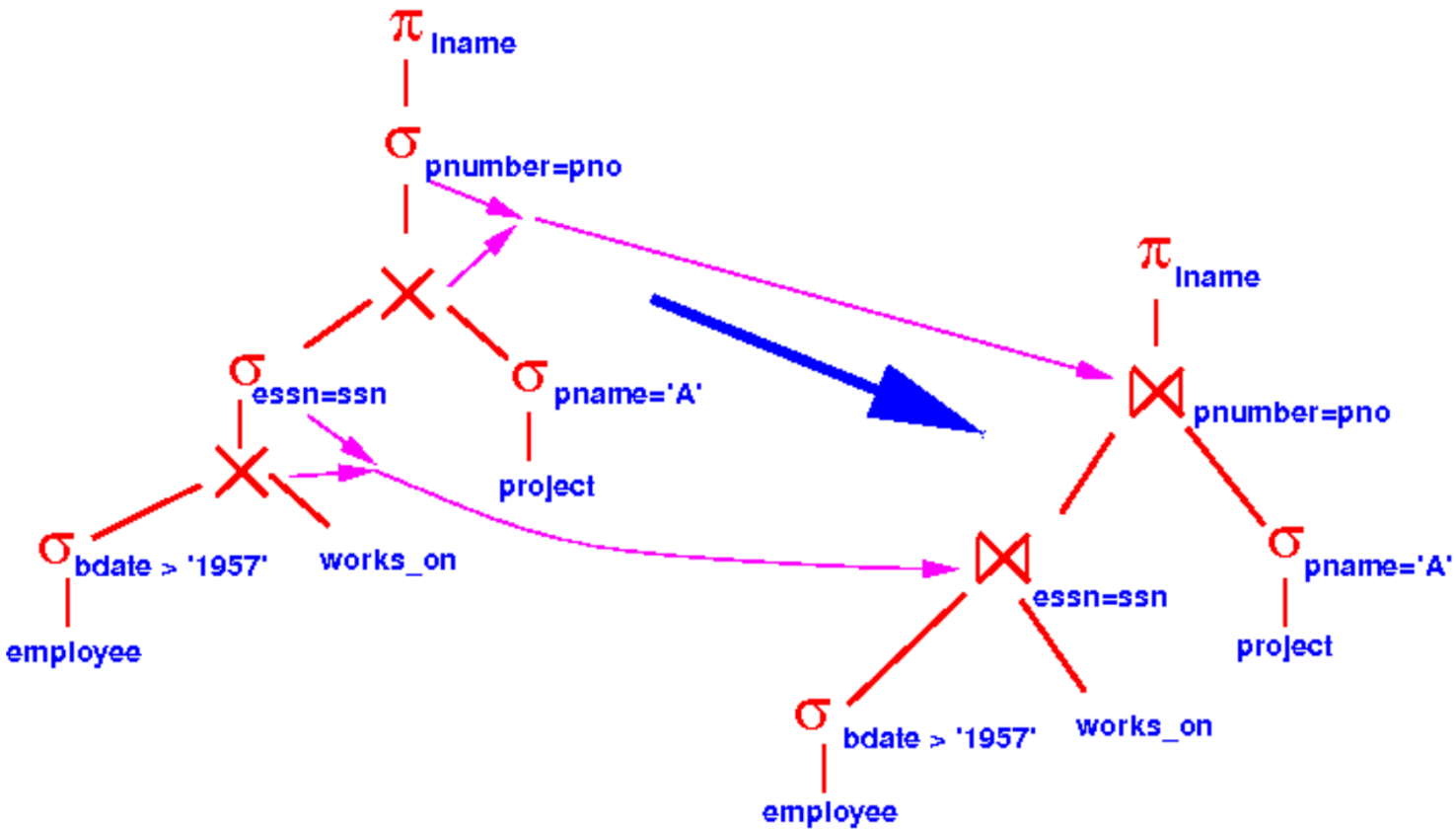
Cartesian Product to Natural Join

$\sigma_{starName=name}$ (MovieStar × StarsIn) ≡ MovieStar $\bowtie_{starName=name}$ StarsIn

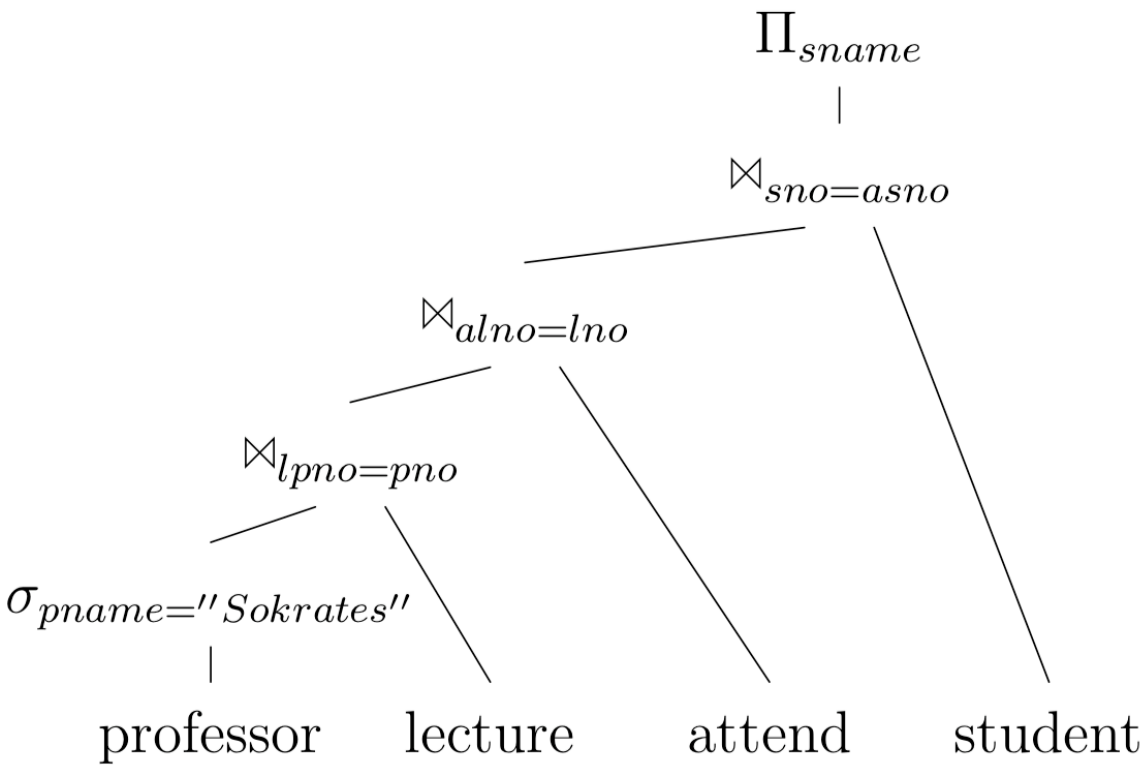


Step 3: Introduce Joins

Replace σ + × with ⋈:

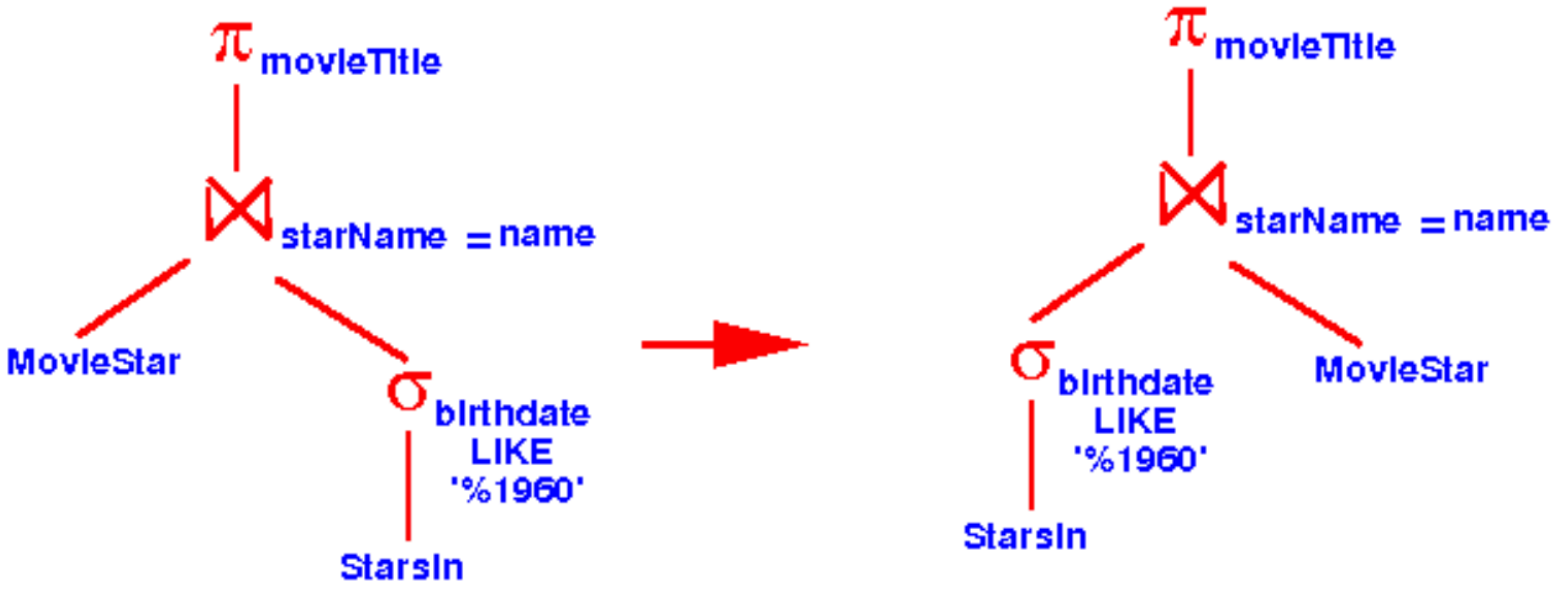


Step 4: Determine Join Order



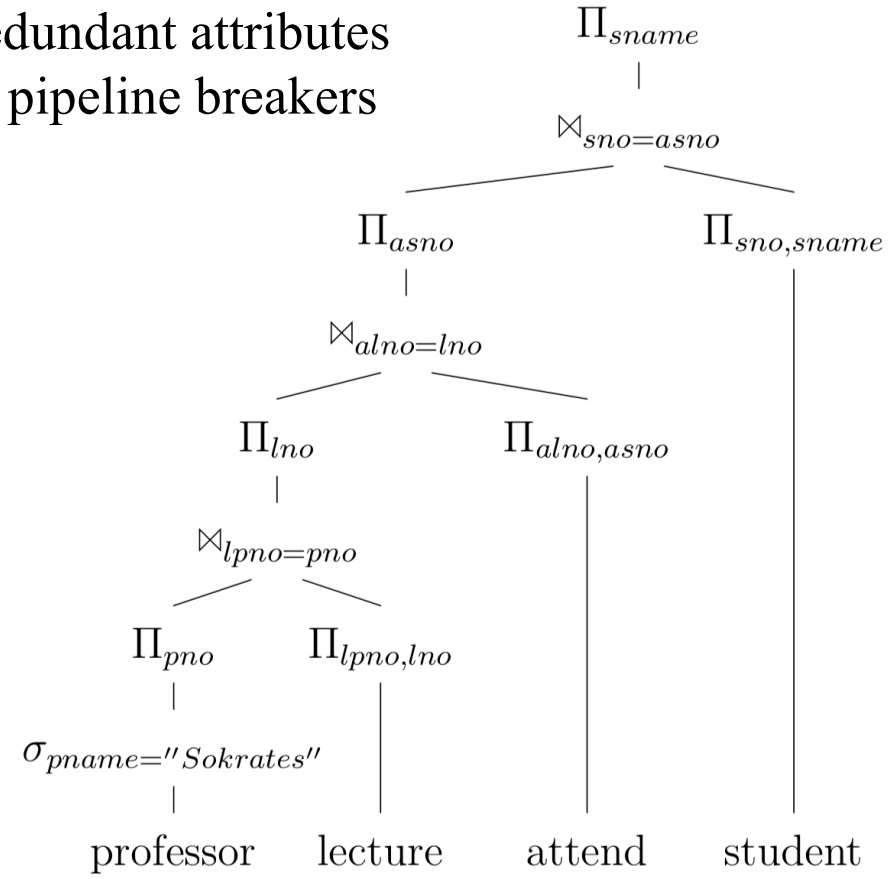
Step 4: Determine Join Order

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



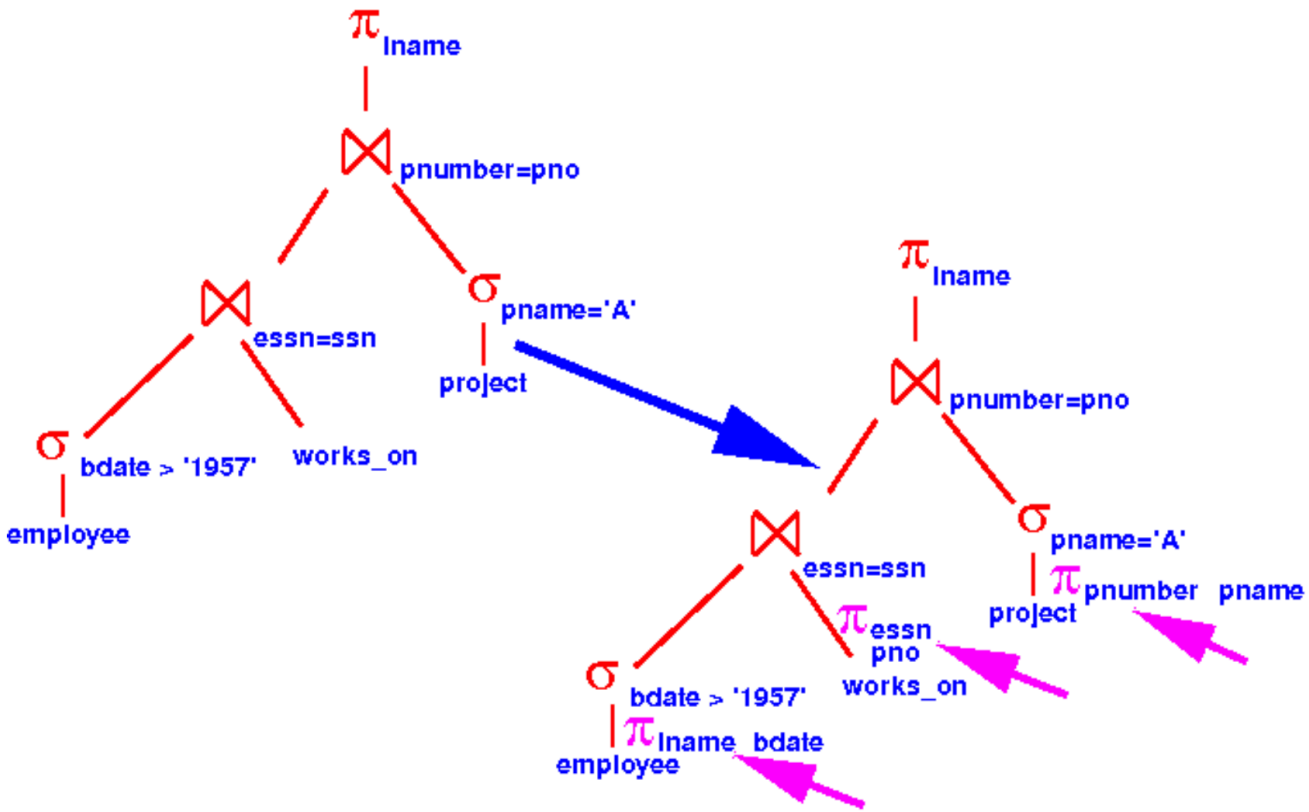
Step 5: Introduce and Push Down Projections

- eliminate redundant attributes
- only before pipeline breakers

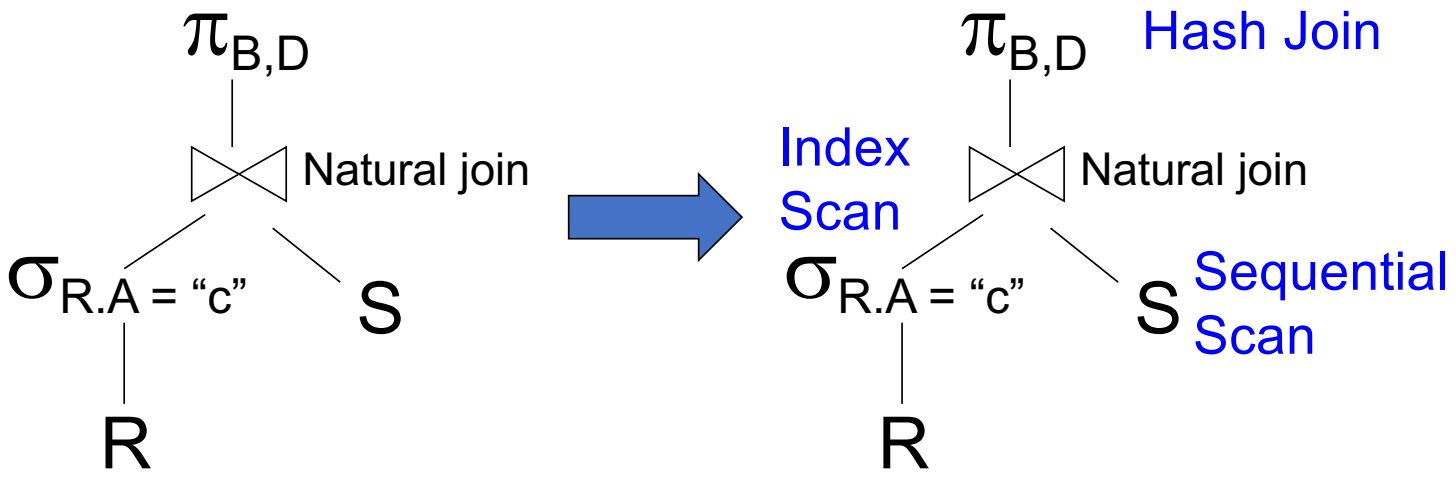


Step 5: Introduce and Push Down Projections

Remove the **unused attributes** by inserting **projection (π)**:



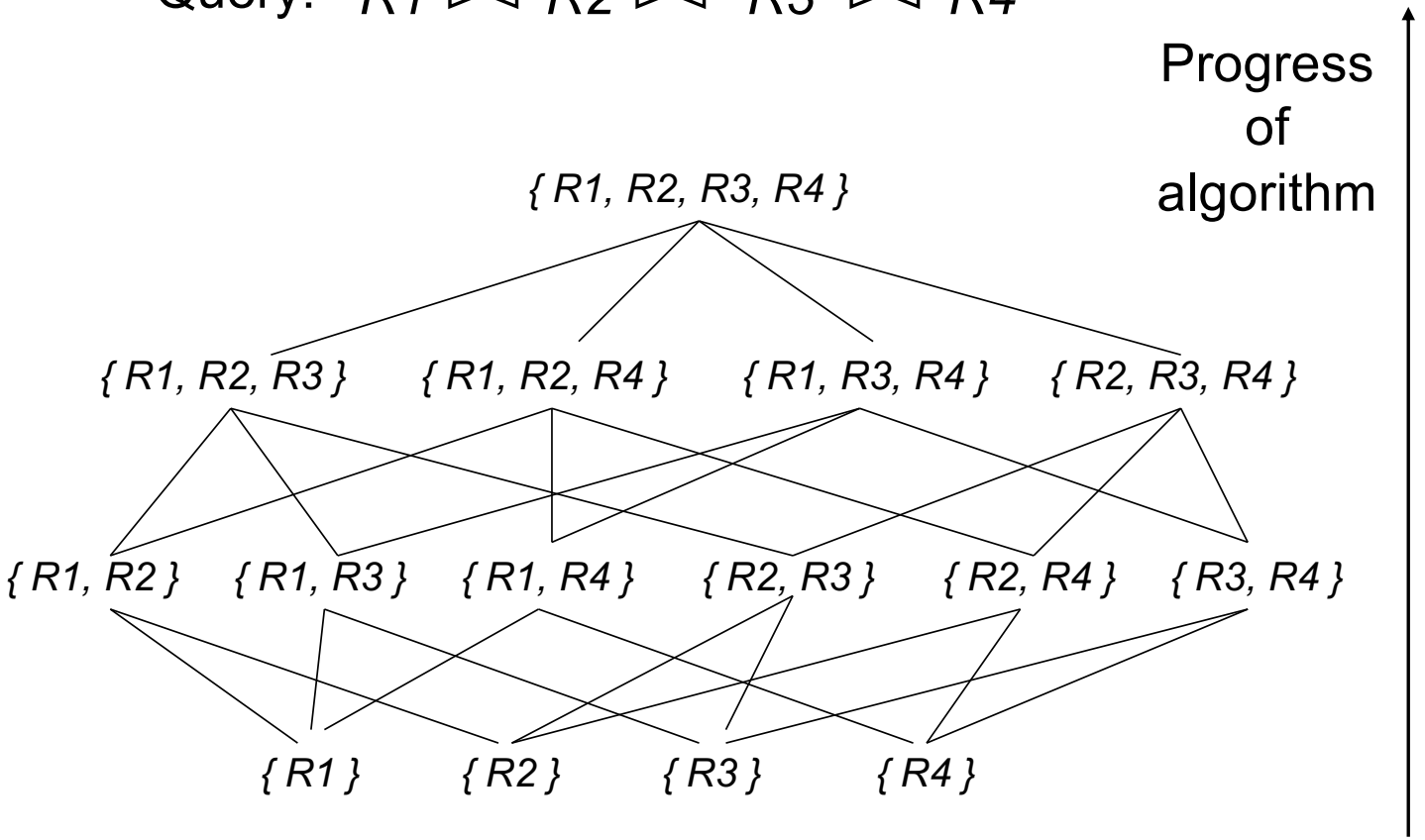
Physical Optimization



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

Join Order Selection

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



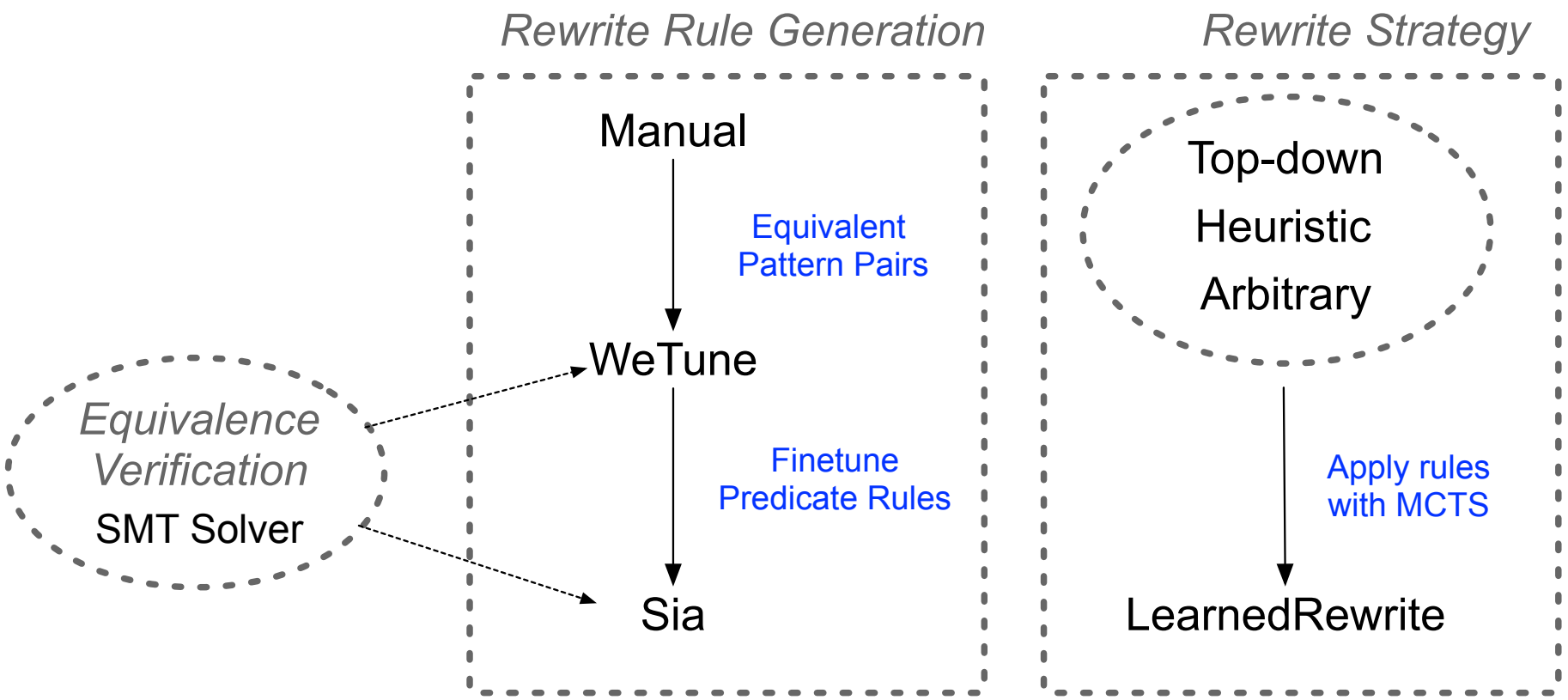
Dynamic Programming

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.

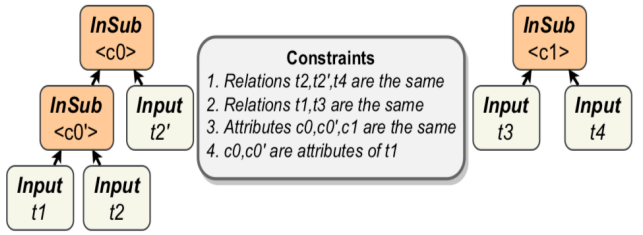
Learning-based Query Rewrite



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

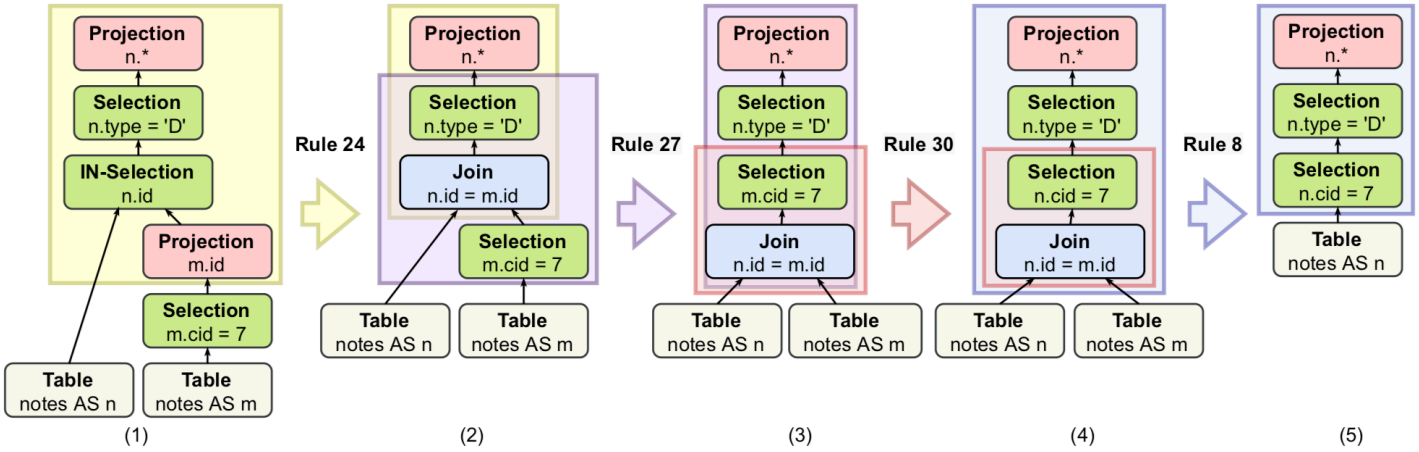



```
q5: ... FROM T WHERE T.x IN (SELECT R.y FROM R)
                  AND T.x IN (SELECT R.y FROM R)
q6: ... FROM T WHERE T.x IN (SELECT R.y FROM R)
```

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

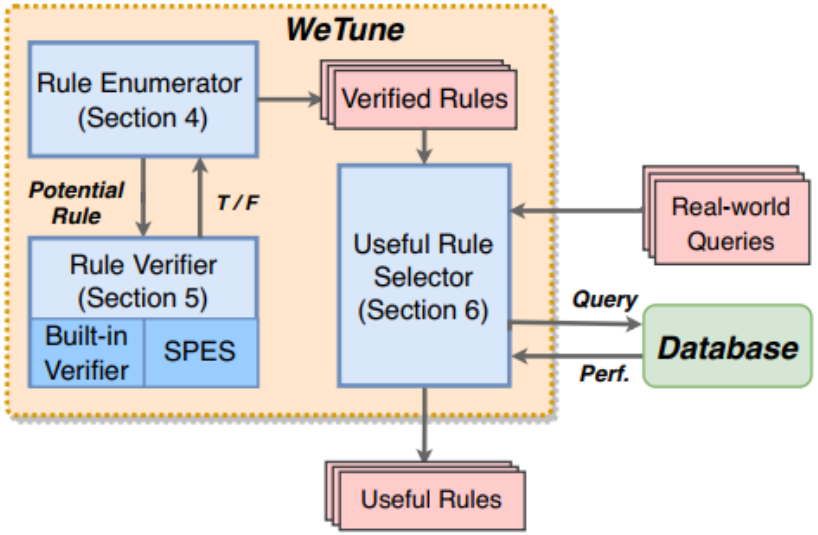



```
Original   SELECT n.* FROM notes AS n WHERE n.type = 'D'
             AND n.id IN (SELECT m.id FROM notes AS m WHERE m.commit_id = 7)
Optimized  SELECT n.* FROM notes AS n WHERE n.type = 'D' AND n.commit_id = 7
```

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

Learning New Rewrite Rules

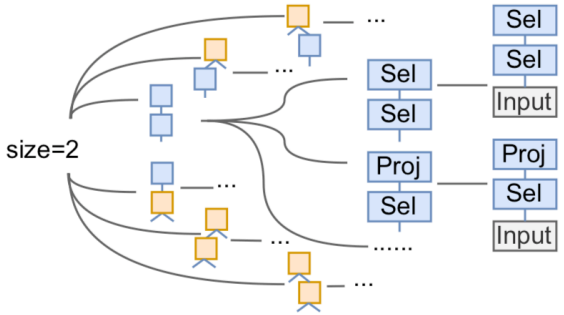
- **Motivation: Identify new rules to gain performance improvement**
- **Basic Idea: Extract relatively simple query pattern pairs from the public datasets and synthesize new rewrite rules**
- **Challenge: (1) How to generate new rewrite rules; (2) How to verify the rewrite equivalence**
- **Solution**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

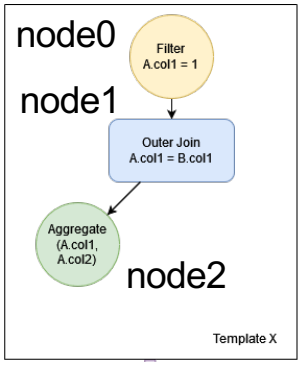


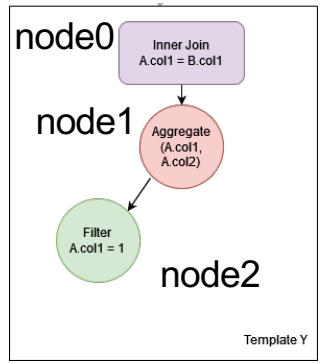
Wang Z, Zhou Z, Yang Y, et al. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. SIGMOD, 2022.

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,*
    - $\boldsymbol{query_{FOL}(X)} \rightarrow \boldsymbol{query_{FOL}(Y)}$ && $\boldsymbol{query_{FOL}(Y)} \rightarrow \boldsymbol{query_{FOL}(X)}$
- **Use verified rules to greedily rewrite queries**





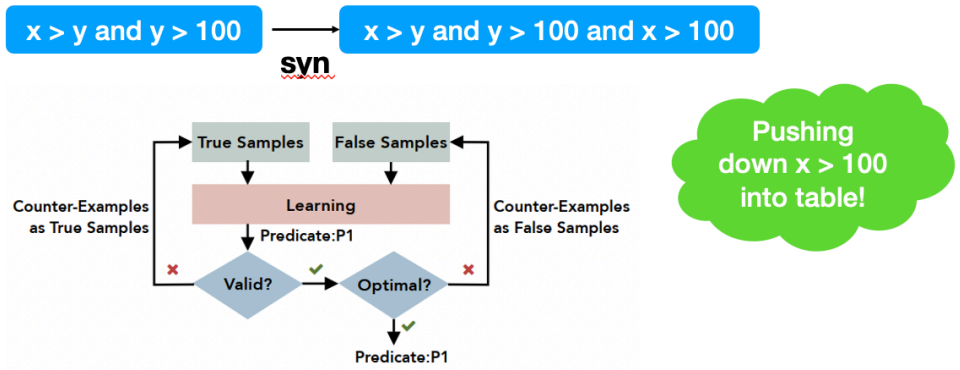


U-expression	FOL formula
$E_1 = E_2$	$\mathrm{Tr}(E_1) = \mathrm{Tr}(E_2)$
$E_1 + E_2$	$\mathrm{Tr}(E_1) + \mathrm{Tr}(E_2)$
$E_1 \times E_2$	$\mathrm{Tr}(E_1) \times \mathrm{Tr}(E_2)$
$\|\|E\|\|$	$ite(\mathrm{Tr}(E) > 0, 1, 0)$
$not(E)$	$ite(\mathrm{Tr}(E) > 0, 0, 1)$
$[p]$	$ite(p, 0, 1)$
$\|\|\sum_x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

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.

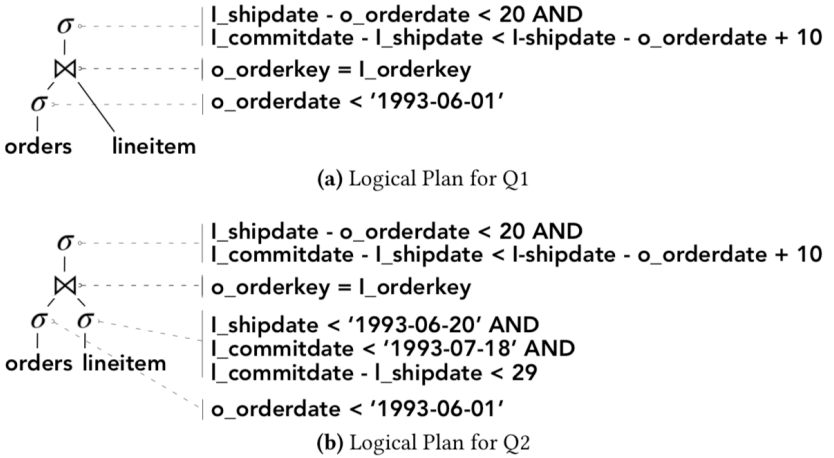
Finetune Predicate Pushdown Rules

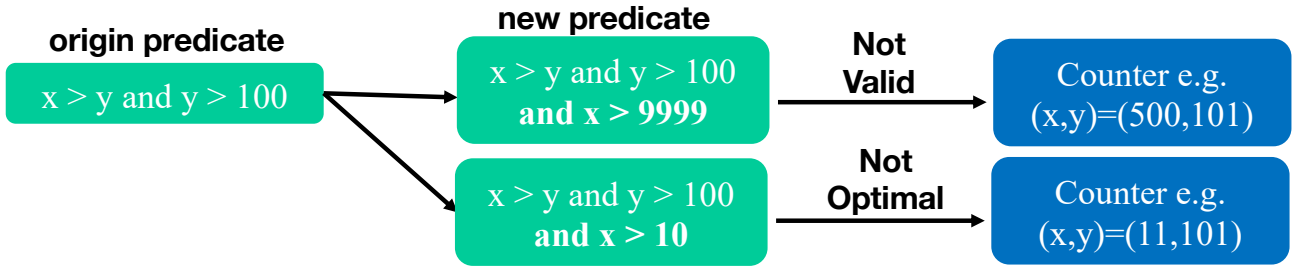
☐ Build a classification model (SVM)

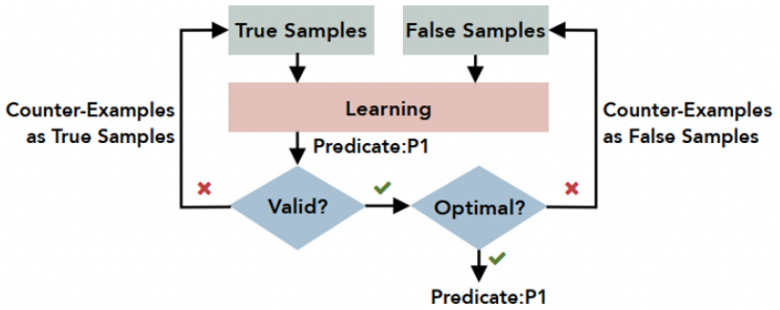
- ➢ Classification Model ⇔ 0/1 ⇔ New Predicate

➢ 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).







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 poorly-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**

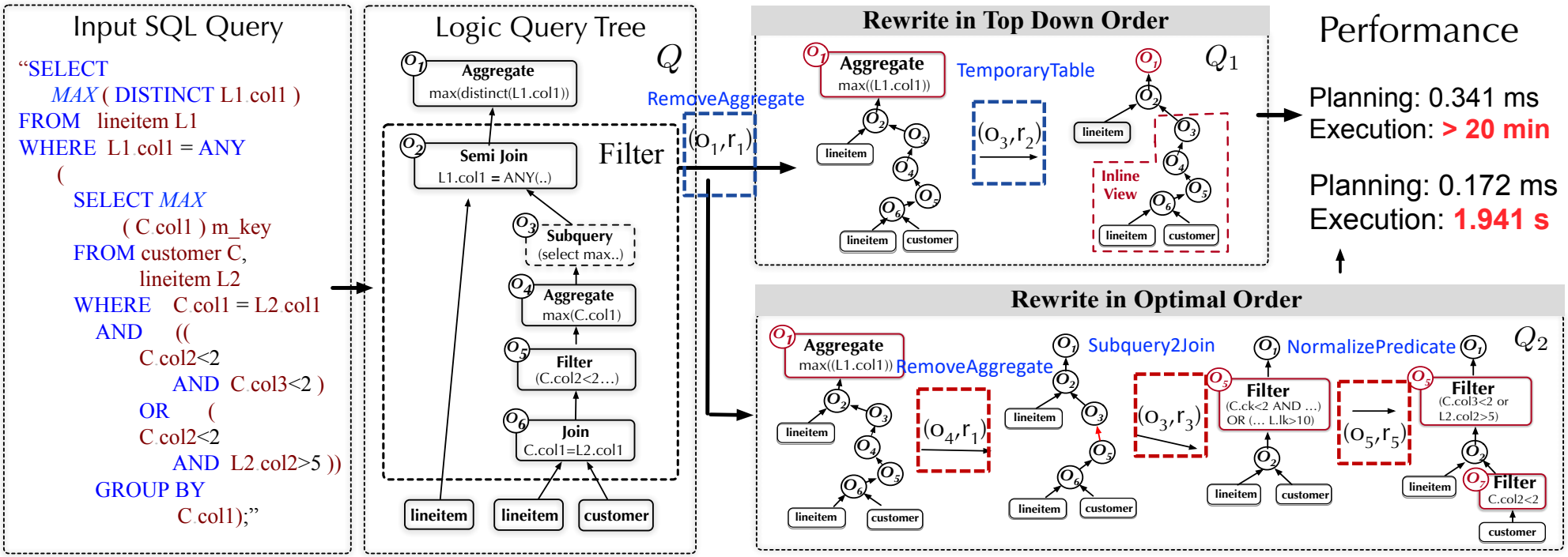
- **Search rewrite space within time constraints**
  - Rewrite within milliseconds;

- **Estimate rewrite benefits by multiple factors**
  - Reduced costs after rewriting
  - Future cost reduction if further rewriting the query

Automatic Query Rewrite

Problem Definition

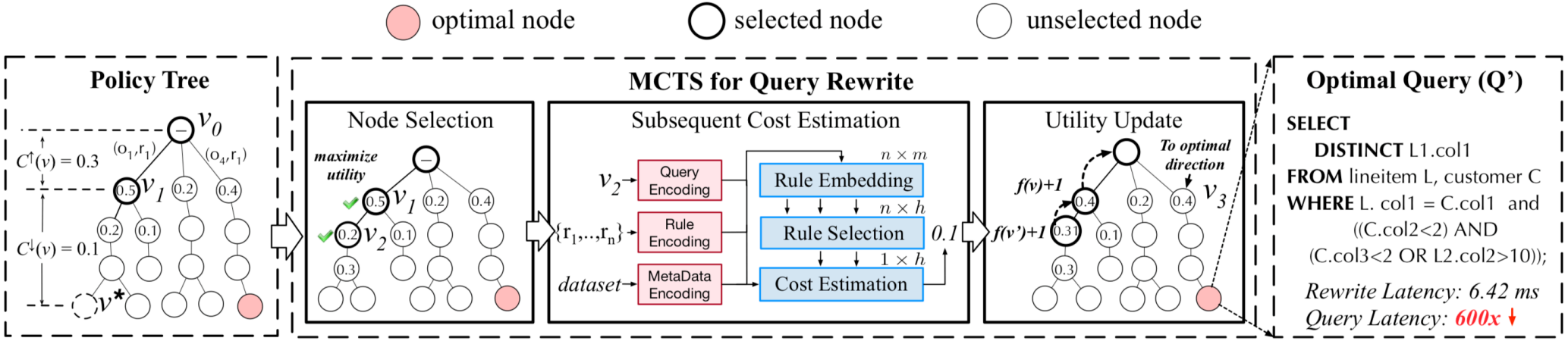
- Given a slow query Q and a set of rewrite rules R, apply the rules R to the query Q so as to gain (a) the equivalent one and (b) the minimal cost.



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

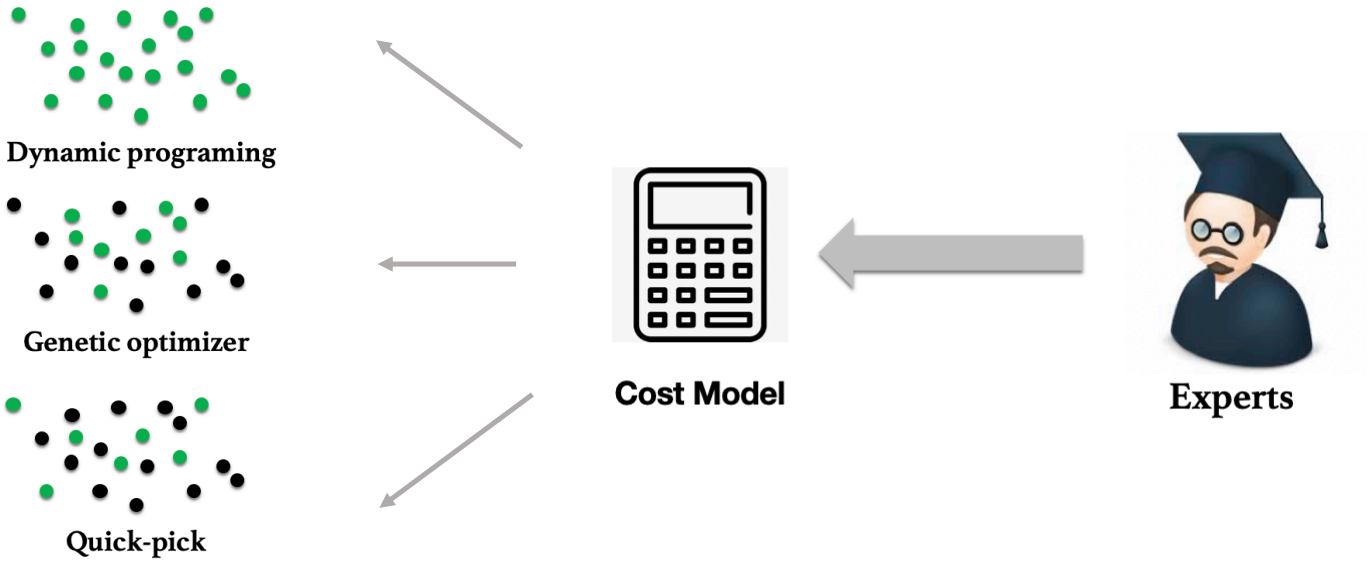
Join Order Selection

- **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.

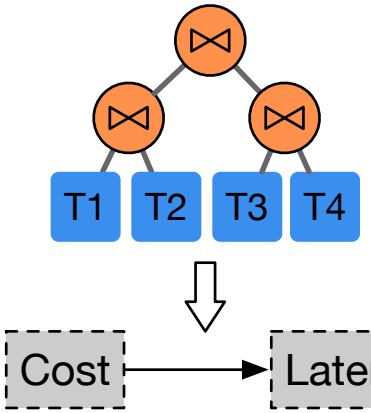
- **Key idea:** Use existing workload to train a learned optimizer, which predicts the plan for future queries.

Online Optimization Methods.

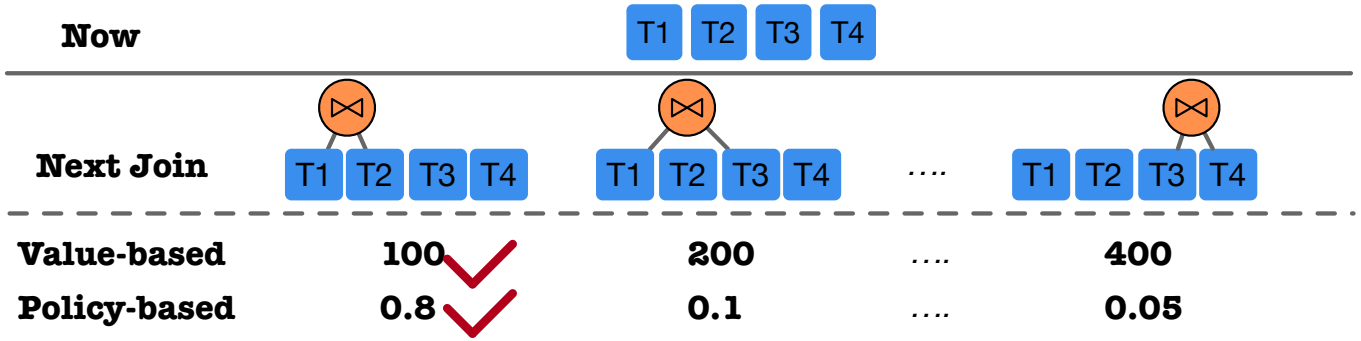
- Characteristic: No workload, but rely on customized Database.

- **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.



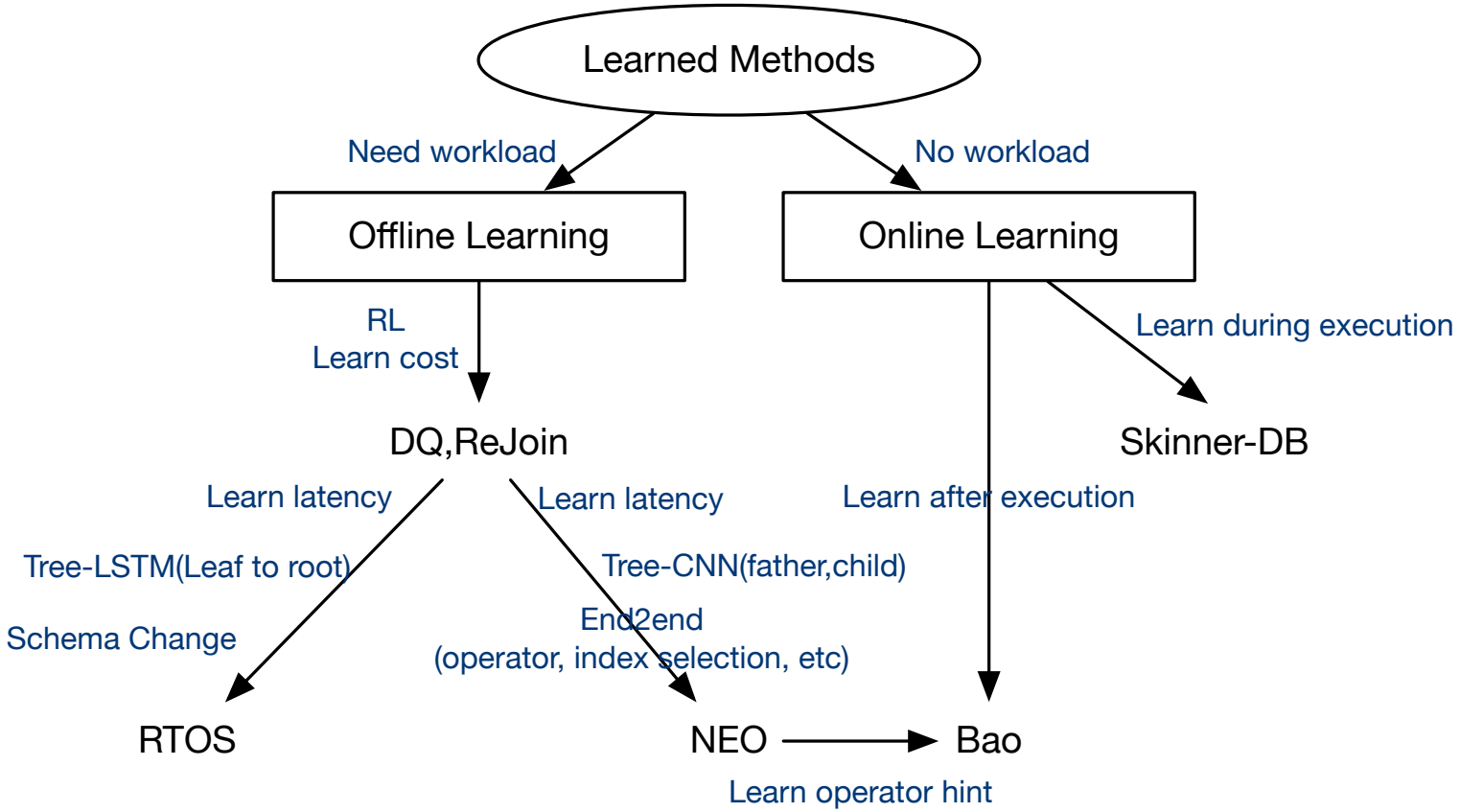
Learned Join Order Selection

- **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

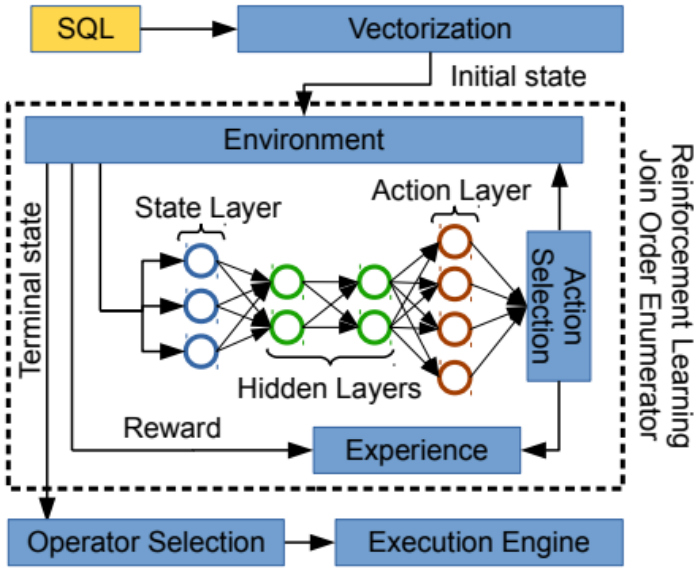
- **Method Classification**
  - **Offline learning methods**
    - **Characteristic : Learn before use - given workload**
    - **Key idea : Use existing workload to train a learned optimizer, which will predict the plan for future workload.**
  - **Online learning methods**
    - **Characteristic: Learn 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

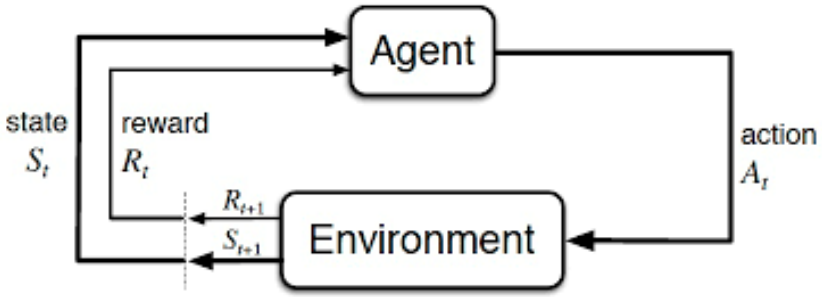
- Motivation
  - The search space for join order is huge.
  - Traditional optimizer did not learn from previous bad or good choice.
- Challenges
  - How to reduce the search space of join order.
  - How to select the best join order.
- Difference：
  - ReJoin uses a policy based method (PPO) to guide 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.



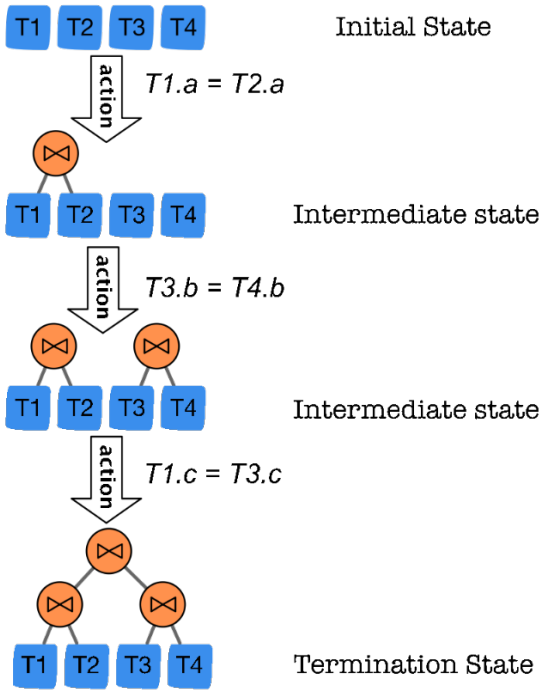
Marcus, Ryan, and Olga Papaemmanouil. "Deep reinforcement learning for join order enumeration." ,aiDM 2018
Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning, arXiv 2018

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
```

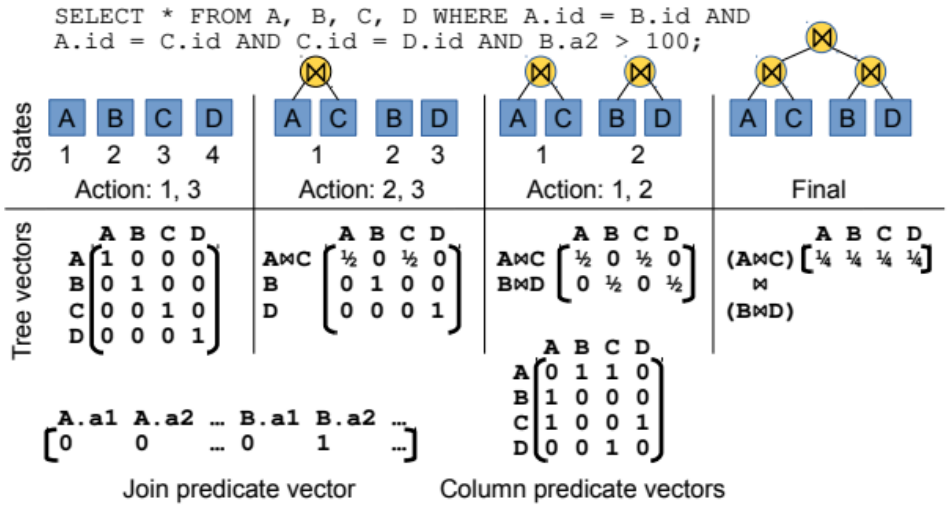


Marcus, Ryan, and Olga Papaemmanouil. “Deep reinforcement learning for join order enumeration.” ,aiDM 2018

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.**

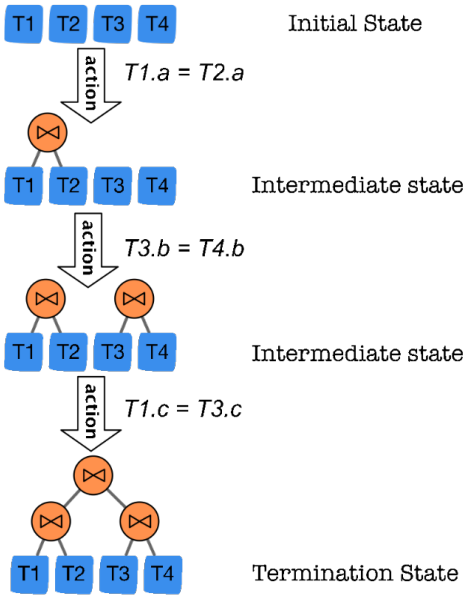


Marcus, Ryan, and Olga Papaemmanouil. “Deep reinforcement learning for join order enumeration.” ,aiDM 2018

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 best terminal state that each state can reach.**
    - **Neural network : A two-layer MLP.**

```
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 $E \bowtie P$

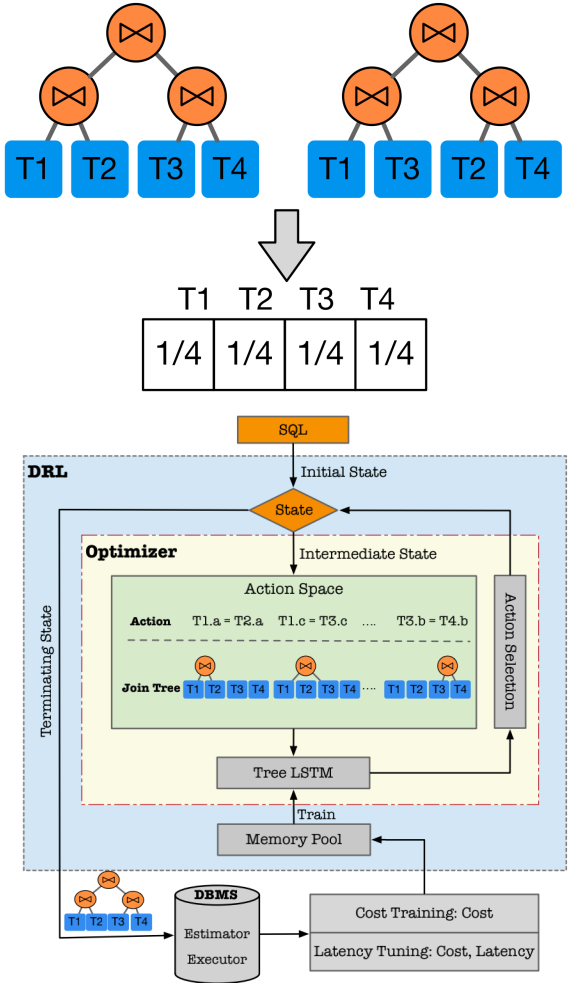
A_L = [E.id, E.name, E.rank, P.rank, P.title, P.code]
= [1 1 1 1 1 1 0 0]
A_R = [S.code, S.amount]
= [0 0 0 0 0 0 1 1]

(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

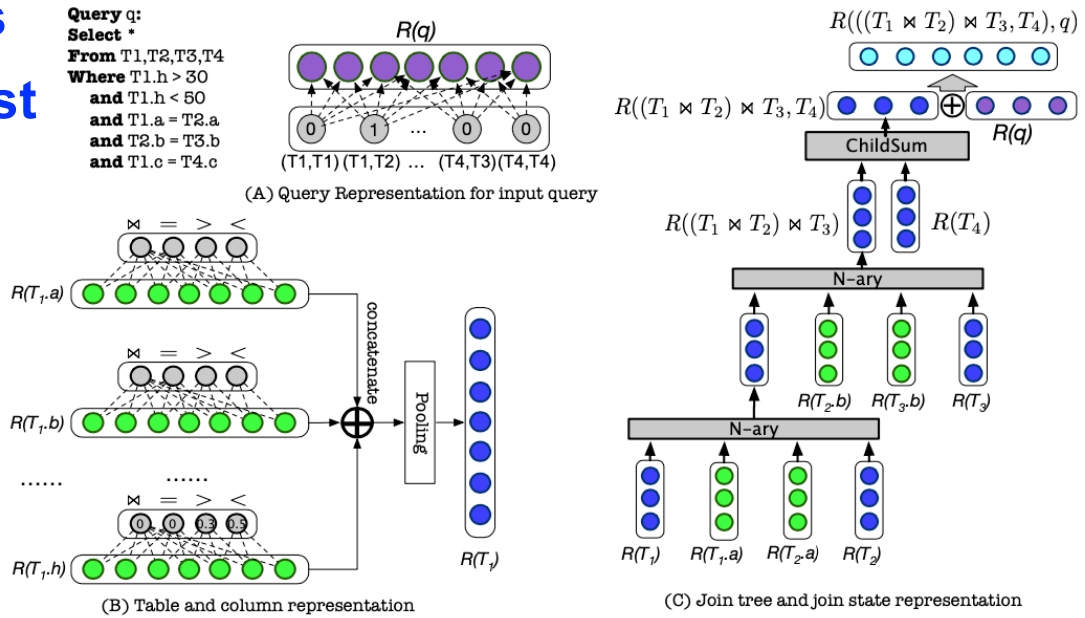
- **Motivation**
  - Previous learning based optimizers give good cost, but they do not give good latency on test queries.
  - Schema often changes in real-world database.
- **Challenges**
  - The intermediate state is a forest, which cannot be represented by a simple feature vector.
  - The training 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

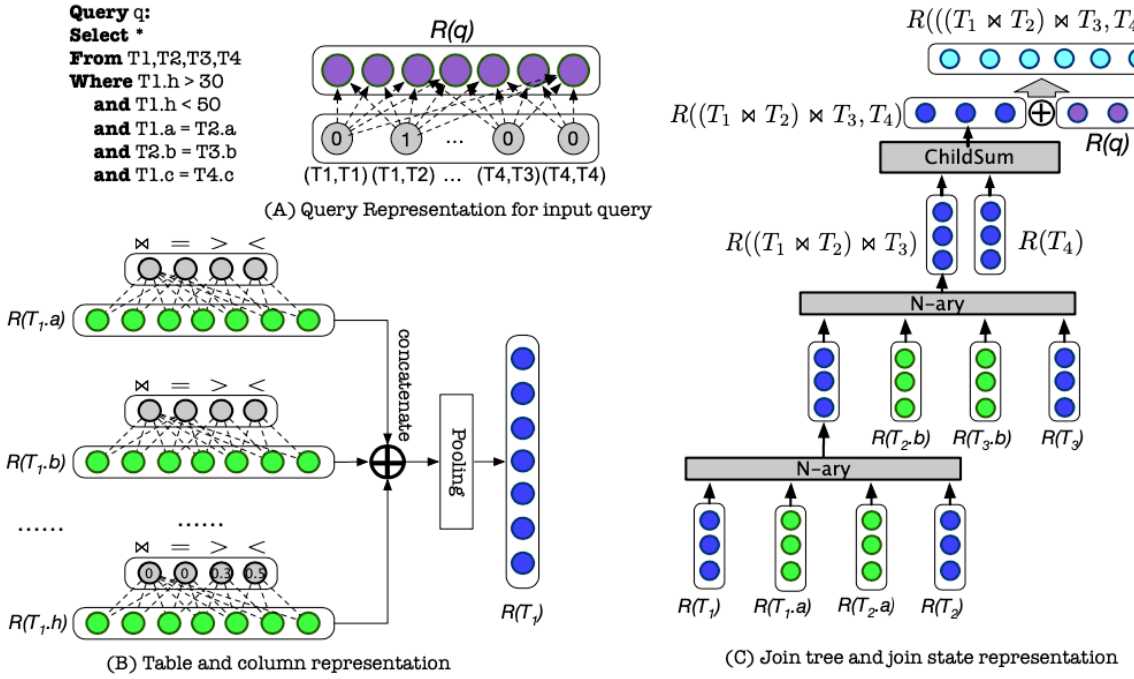


Yu X, Li G, Chai C, et al. Reinforcement learning with tree-lstm for join order selection. ICDE 2020

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.

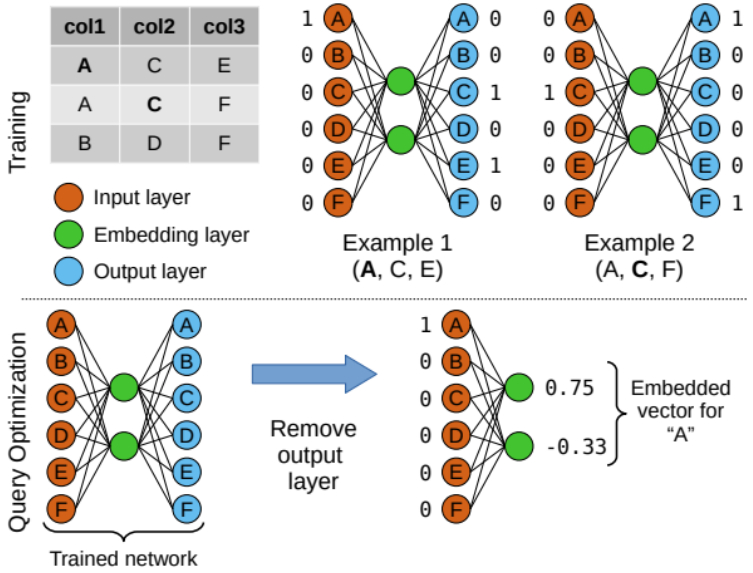
3 Offline Learned Join Order Selection: NEO

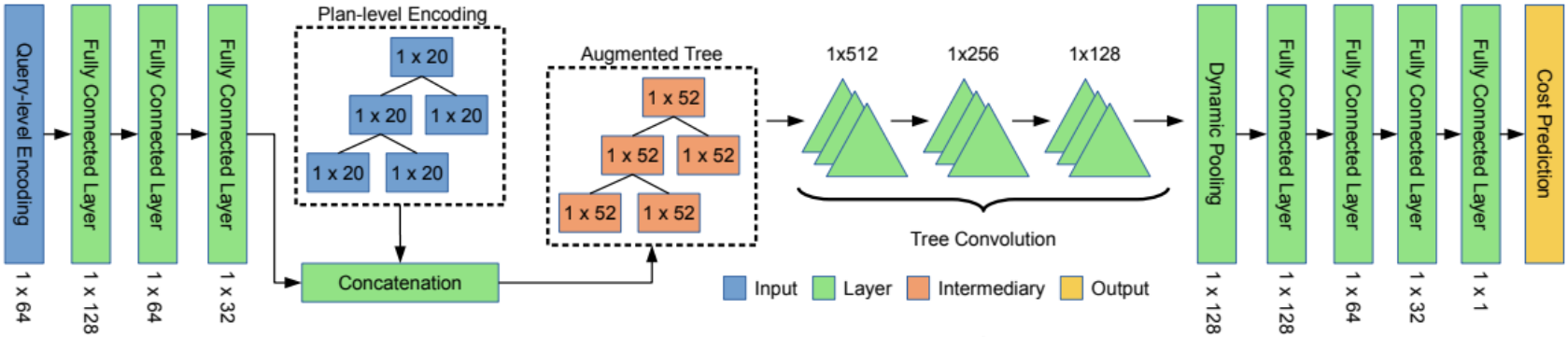
- Motivation
  - Previous traditional optimizer relies on cost models
  - Previous methods solve join ordering only but cannot support physical operator selection.
- Challenges
  - How to build a learn cost model automatically to capture intuitive patterns in tree-structured query plans and predict the 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





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

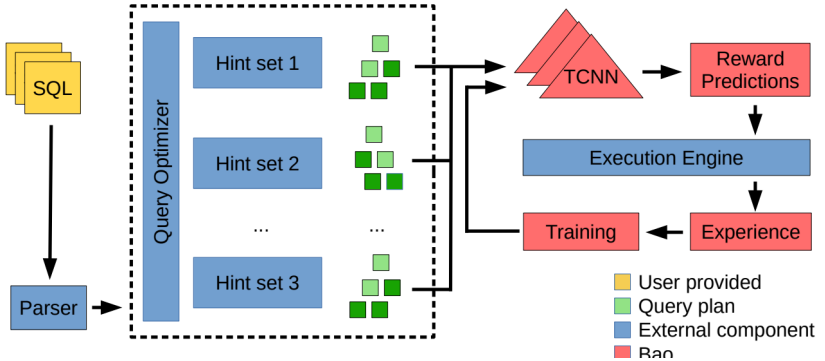
4 Online Learned Join Order Selection: Bao

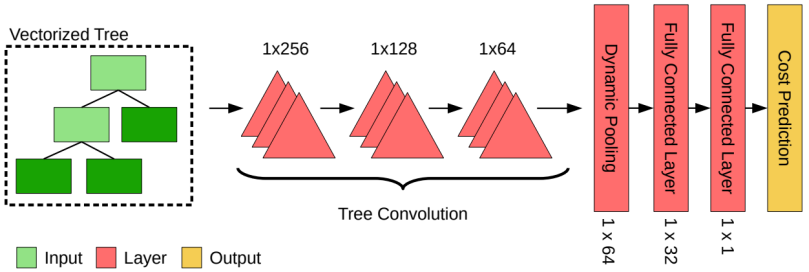
- Motivation
  - Long training time
  - Cannot adjust to data and workload changes
  - Tail latency of worse plans
  - The choice of physical operator affects the quality of the plan
- Challenges
  - How to enumerate the plan?
  - How to study plan latency and choose a high-quality plan?

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.

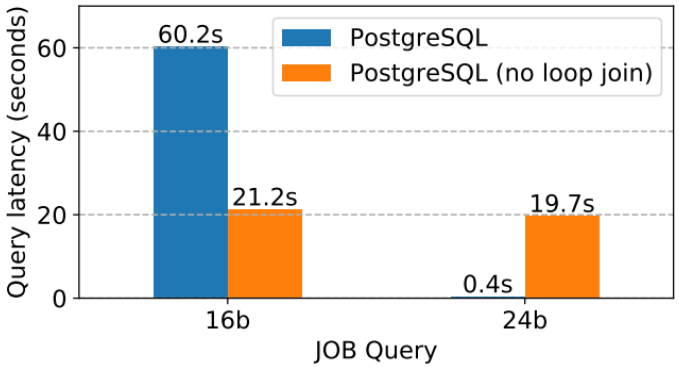




Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

4 Online Learned Join Order Selection: Bao

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

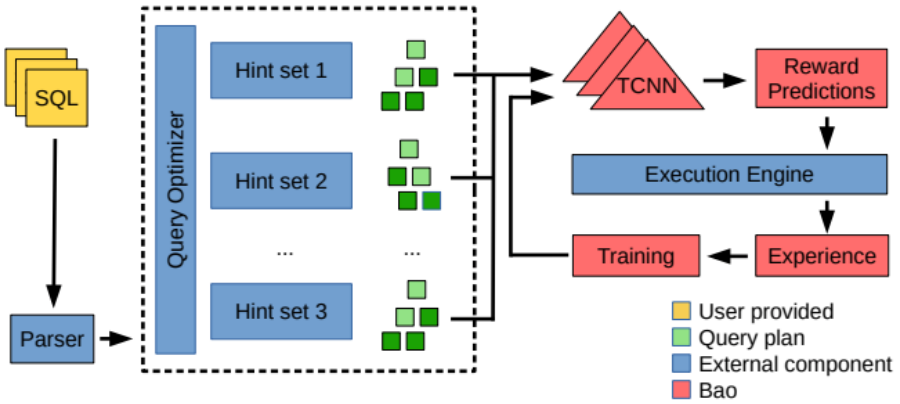


- **Model Plan Hinter as a Multi-armed Bandit Problem**
  - Model each hint set $HSet_i$ as a query optimizer

$$HSet_i : Q \rightarrow T$$

  - 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.

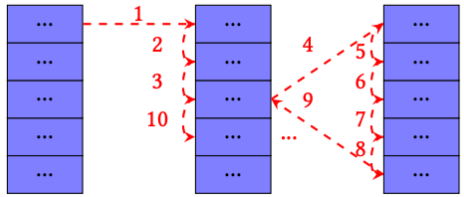
5 Online Learned Join Order Selection: SkinnerDB

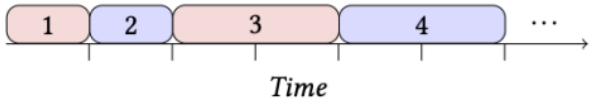
- Motivation
  - Previous work relied on learning from cost models or expert optimizers.
  - Previous learning based optimizer need to give training queries and hard to give good plans to different workload.
  - The executor can detect estimation errors during query execution.
- Challenges
  - How to 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.

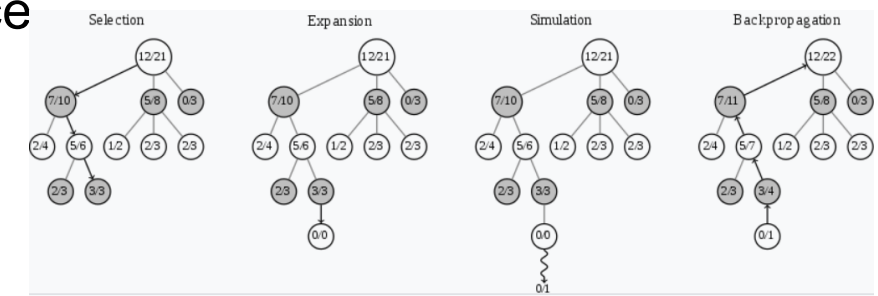
Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

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



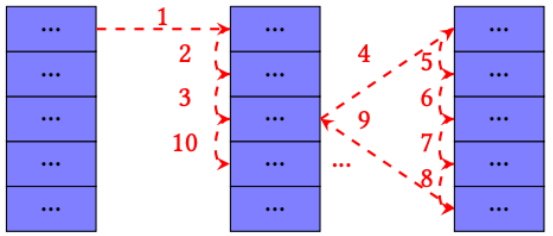




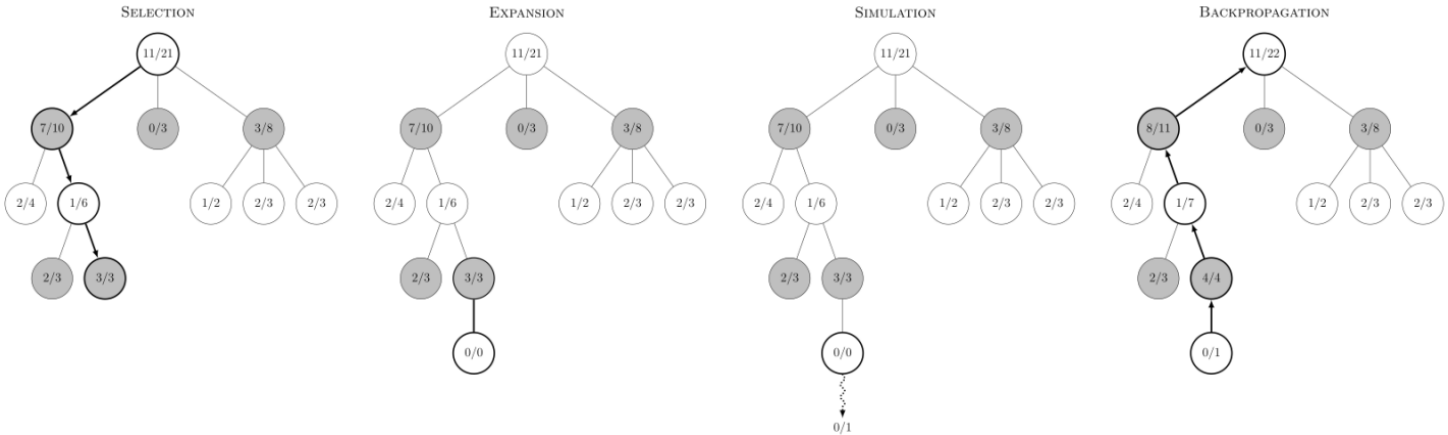
Marcus R, Negi P, Mao H, et al. Neo: a learned query optimizer. VLDB 2019

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**



N way join



Monte Carlo tree search (MCTS).

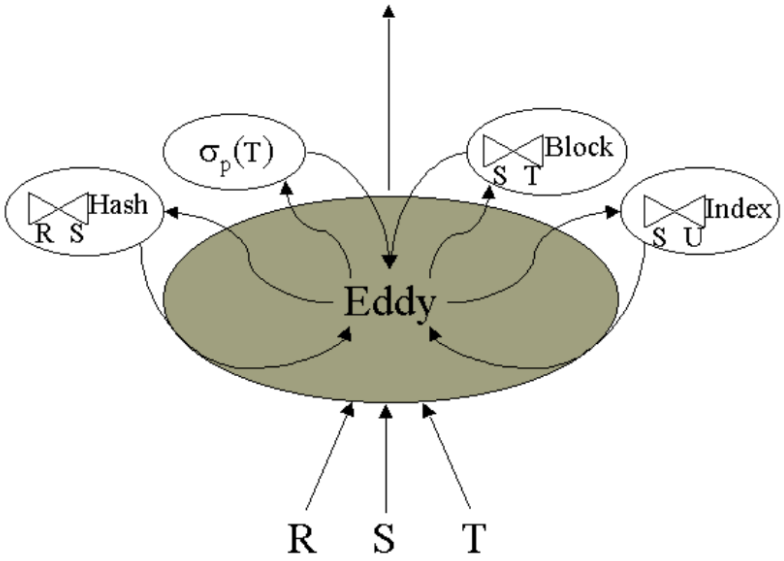
Trummer, et al Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In SIGMOD, 2019.

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

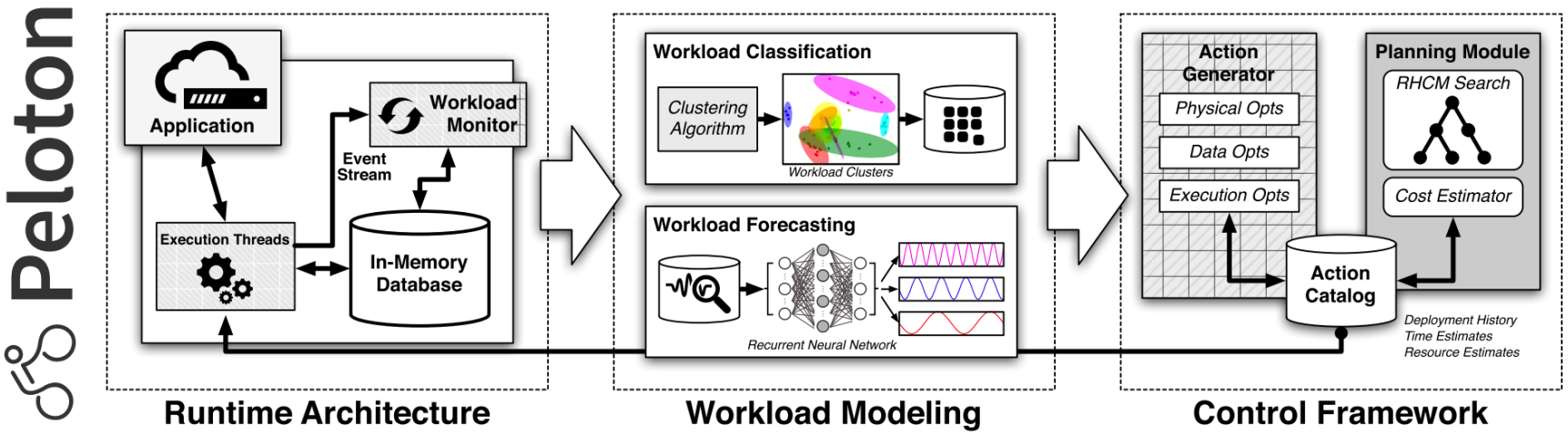
- Local optimum with high training overhead

☐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.

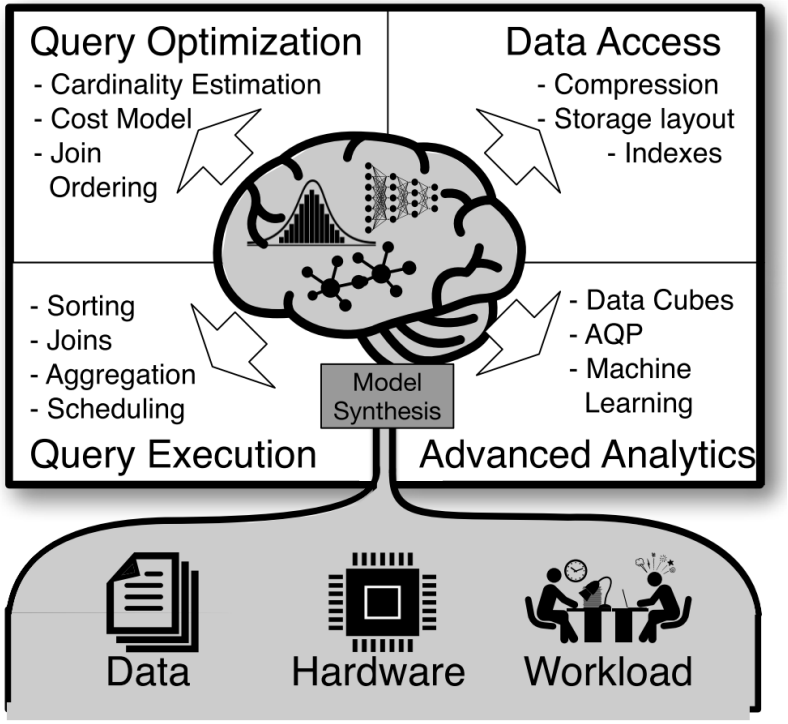
SageDB

- **Customize DB design via learning the Data Distribution**

  - Learn Data Distribution by Learned CDF

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

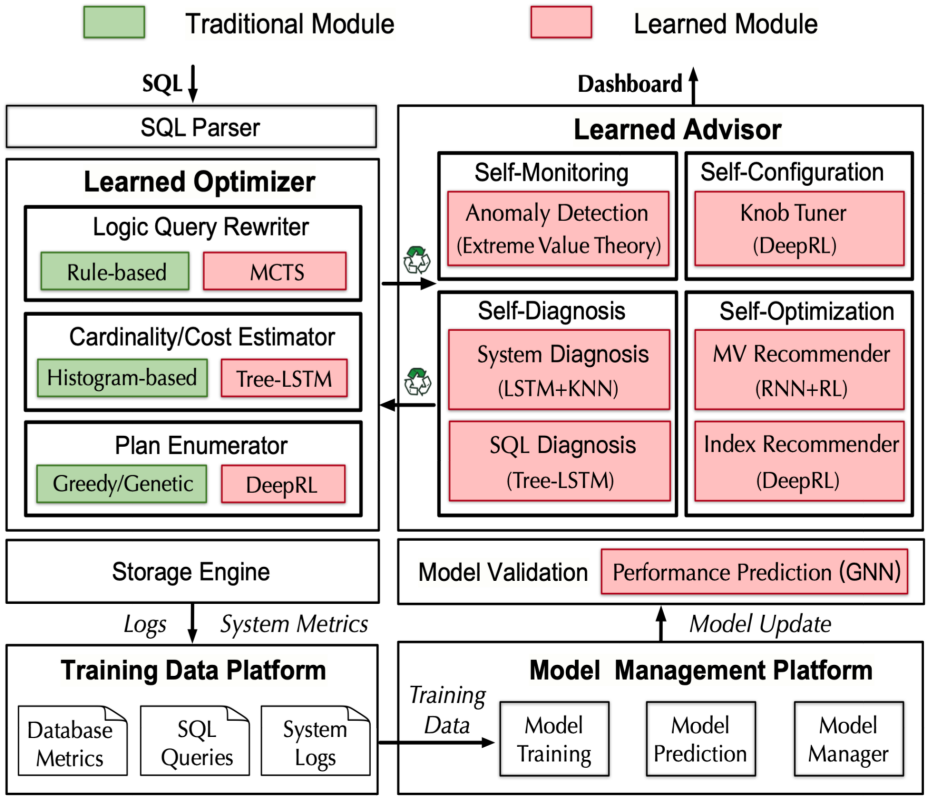
  - Design Components based on the learned CDFs
    - Query optimization and execution
    - Data layout design
    - Advanced analytics



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 of ML4DB

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: Optimization Overhead

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

Future Works: Adaptivity

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

Future Works: Complex Scenarios

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

Future Works: Small Training Data

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

Future Works: 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?

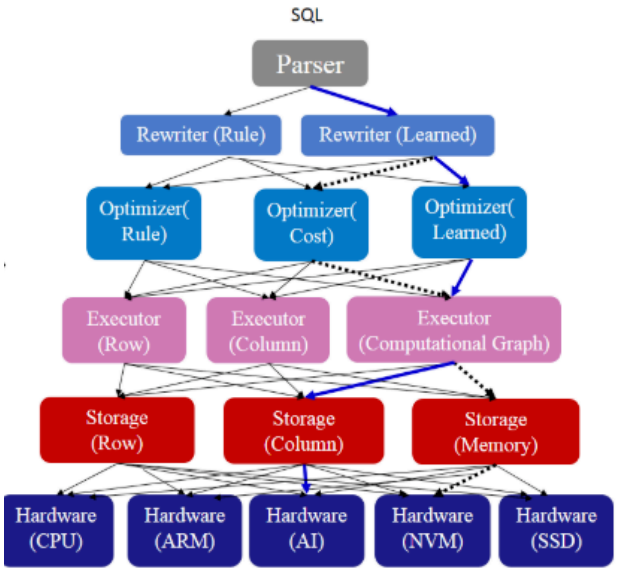
Future Works: One Model Fits Various Scenarios

- **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