CS 4440 A

Emerging Database Technologies

Lecture 17

03/16/26

Announcements

- Assignment 3 due on Wednesday

- Assignment 4 will be released today
  - Programming assignment in C++.
  - Due Apr 8. Start early!

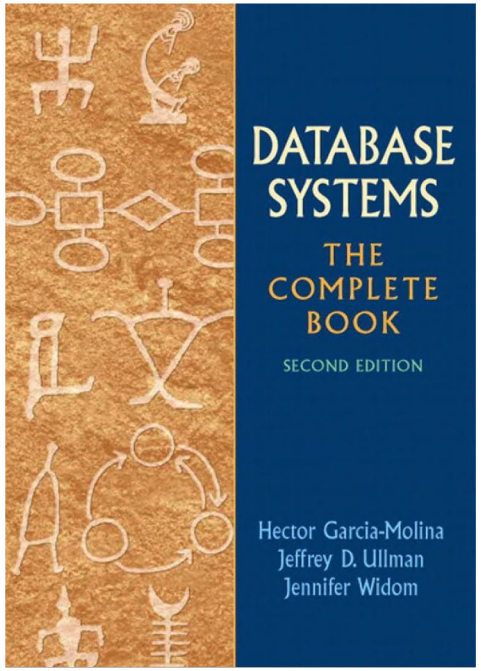
Reading Materials

Query execution (Chapters 15.1 - 15.6)

- Physical operators
- Implementing operators and estimating costs

Query optimization (Chapters 16.1 - 16.5)

- Parsing
- Algebraic laws
- Parse tree -> logical query plan
- Estimating result sizes
- Cost-based optimization

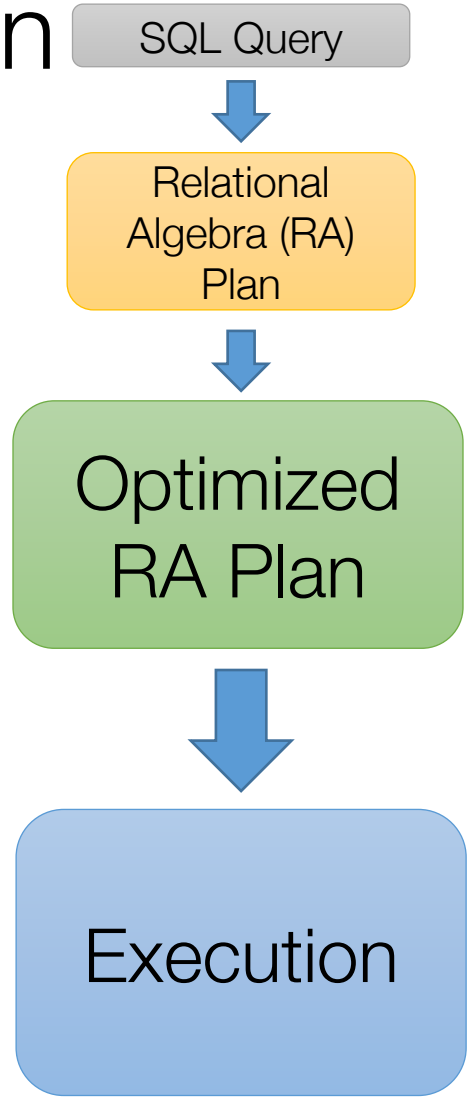


Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang.

Agenda

1. Estimating cost of a physical plan
2. Cost-based Query Optimization
3. *Brief intro to learned query optimizers

Recall: Logical vs. Physical Optimization



Logical optimization:

- Find equivalent plans that are more efficient
- *Intuition: Minimize # of tuples at each step by changing the order of RA operators*

Physical optimization:

- Find algorithm with lowest IO cost to execute our plan
- *Intuition: Calculate based on physical parameters (buffer size, etc.) and estimates of data size (histograms)*

1. Estimating cost of a physical plan

Estimating the cost of a physical query plan

Step 1: Estimate the size of results

- Projection
- Selection
- Joins

Step 2: Estimate the # of disk I/O's

We already know how to do step 2 for joins!

Notation: Size parameters

$P(R)$: # pages to hold tuples in R

$T(R)$: # tuples in R

$V(R, a)$: # distinct values of attribute a in R

Notation: Size parameters

Example:

R

A	B	C
cat	1	2000
cat	1	2001
dog	1	2002

A: 10 byte string
B: 4 byte integer
C: 8 byte date

T(R) = 3
V(R, A) = 2
V(R, B) = 1
V(R, C) = 3

Suppose each page is 100 bytes
Then a page fits 4 tuples
If T(R) = 1000
Then P(R) = 1000 / 4 = 250

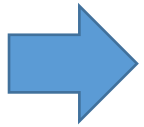
For $\pi_A(R)$, each page fits 10 tuples, so
P(R) = 1000 / 10 = 100

Estimating size of selection

A selection generally reduces the number of tuples

Estimated result size
(without any additional information)

$$S = \sigma_{A=c}(R)$$



$$T(S) = \frac{T(R)}{V(R,A)}$$

*Assumption: values in A = c are uniformly distributed over possible V(R, A) values

Estimating size of selection

A selection generally reduces the number of tuples

Estimated result size
(without any additional information)

$$S = \sigma_{A<c}(R) \quad \Rightarrow \quad T(S) = \frac{T(R)}{3}$$

*Assumption: queries involving inequalities tend to retrieve a small fraction of possible tuples

Example: postgres/src/include/utils/selfuncs.h

Estimating size of selection

If selection condition is AND of conditions, multiply all selectivity factors

$$S = \sigma_{A=10 \wedge B<20}(R)$$

$$T(R) = 10{,}000$$
$$V(R, A) = 50$$

Q: What is T(S)?

$$T(S) = \frac{T(R)}{50 \times 3} = 67$$

Estimating size of selection

If selection condition is an OR of conditions, can assume independence of conditions

$$S = \sigma_{A=10 \vee B<20}(R)$$

$$T(R) = 10{,}000$$
$$V(R, A) = 50$$

Q: What is T(S)?

$$T(S) = \frac{T(R)}{1 - (1 - 1/50)(1 - 1/3)} = 3466$$

Estimating size of join

We study $R(X,Y) \bowtie S(Y,Z)$

Two simplifying assumptions

- Containment of value sets: if $V(R,Y) \leq V(S,Y)$, then every Y-value of R is a Y-value of S
- Preservation of value sets: $V(R \bowtie S, X) = V(R, X)$

Example when these assumptions are true:
Y is a key in S and the corresponding foreign key in R

Estimating size of join

$R(X,Y) \bowtie S(Y,Z)$

Two simplifying assumptions

- Containment of value sets: if $V(R,Y) \leq V(S,Y)$, then every Y-value of R is a Y-value of S
- Preservation of value sets: $V(R \bowtie S, X) = V(R, X)$

$\boldsymbol{Case\ 1}: V(R,Y) \geq V(S,Y)$

$\Rightarrow T(R \bowtie S) = T(R)T(S)/V(R,Y)$

$\boldsymbol{Case\ 2}: V(R,Y) < V(S,Y)$

$\Rightarrow T(R \bowtie S) = T(R)T(S)/V(S,Y)$

For each pair (r, s), we know that the Y-value of S is one of the Y-values of R by containment of value sets, so the probability of r having the same Y-value is 1/V(R,Y)

$$T(R \bowtie S) = T(R)T(S)/\max(V(R,Y), V(S,Y))$$

Joins of many relations

Compute intermediate *T*, *V* results
Example: *R* ⋈ *S* ⋈ *T*

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

$T(R) = 1000$
$V(R, B) = 20$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 100$

$T(T) = 5000$
$V(T, C) = 500$
$V(T, D) = 200$

Q: What is T(R ⋈ S) and V(R ⋈ S, C)?

Joins of many relations

Compute intermediate T, V results
Example: $R \bowtie S \bowtie T$

$R(A, B)$

$T(R) = 1000$
$V(R, B) = 20$

$S(B, C)$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 100$

$R \bowtie S(A, B, C)$

$T(R \bowtie S) = T(R)\ T(S)\ /$
$\max(V(R, B), V(S, B)) = \ 40000$

$V(R \bowtie S, C) = 100$

Joins of many relations

Compute intermediate T, V results
Example: $R \bowtie S \bowtie T$

$R \bowtie S(A, B, C)$

$T(R \bowtie S) = 40000$
$V(R \bowtie S, C) = 100$

$T(C, D)$

$T(T) = 5000$
$V(T, C) = 500$
$V(T, D) = 200$

$(R \bowtie S) \bowtie T$

$T((R \bowtie S) \bowtie T)$
$= 40000 \text{ x } 5000 \text{ / } \max\{100, 500\}$
$= 400000$

Joins of many relations

Compute intermediate T, V results
Example: consider $R \bowtie S \bowtie T$

$$R \bowtie (S \bowtie T)$$

$$T(R \bowtie (S \bowtie T)) = 1000 \text{ x } (2000 \text{ x } 5000 \text{ / } \max\{100, 500\}) \text{ / } \max\{20, 50\}$$
$$= 400000$$

Assuming containment and preservation of value sets, the estimated result size is the same regardless of how we group and order the terms in a natural join of relations.

Natural joins with multiple join attributes

Same as R ⋈ S with single join attribute, but divide by max{V(R, A), V(S, A)} for each joining attribute A

$R(A, B, C)$

$T(R) = 1000$
$V(R, B) = 20$
$V(R, C) = 100$

$S(B, C, D)$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 50$

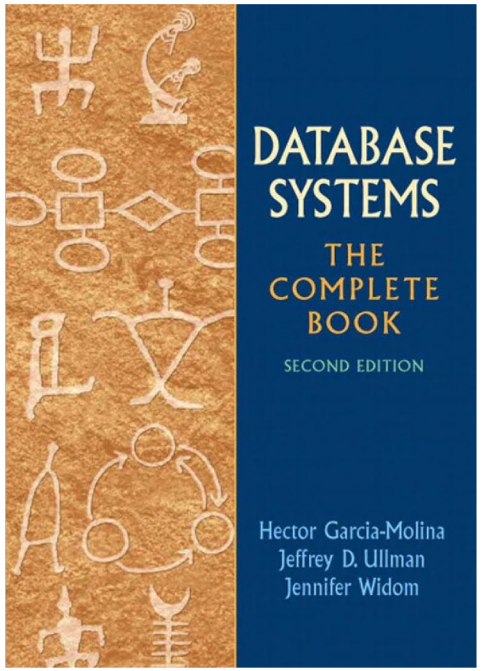
$R \bowtie S$

$T(R \bowtie S) = 1000 \text{ x } 2000$
$/ \max\{20, 50\}$
$/ \max\{100, 50\}$
$= 400$

Further reading

- Using similar ideas, can estimate sizes of other operations like union, intersect, difference, duplicate elimination, grouping

- Chapter 16.4.7

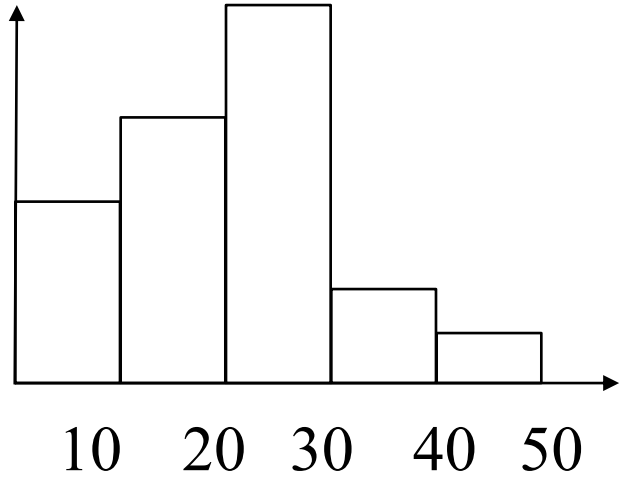


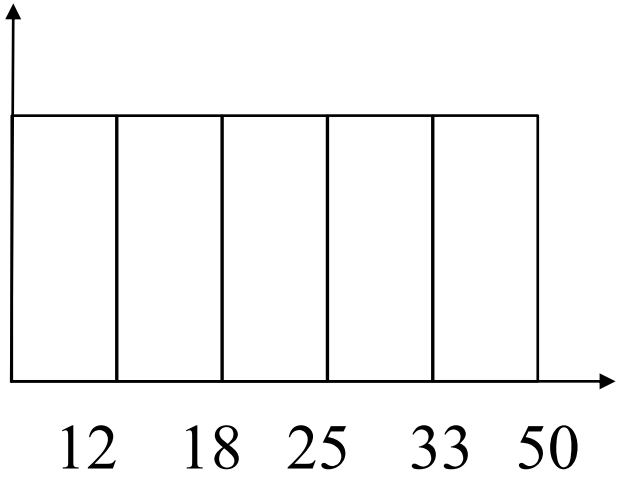
Obtaining estimates for size parameters

Scan entire relation *R* to obtain *T*(*R*), *V*(*R*, *A*), and *P*(*R*)

A DBMS may also compute histograms per attribute for more accurate estimations

- Equal-width and equal-depth histograms





$$\sigma_{A=22}(R) = \ ?$$

Computation of statistics

Computed periodically or by request

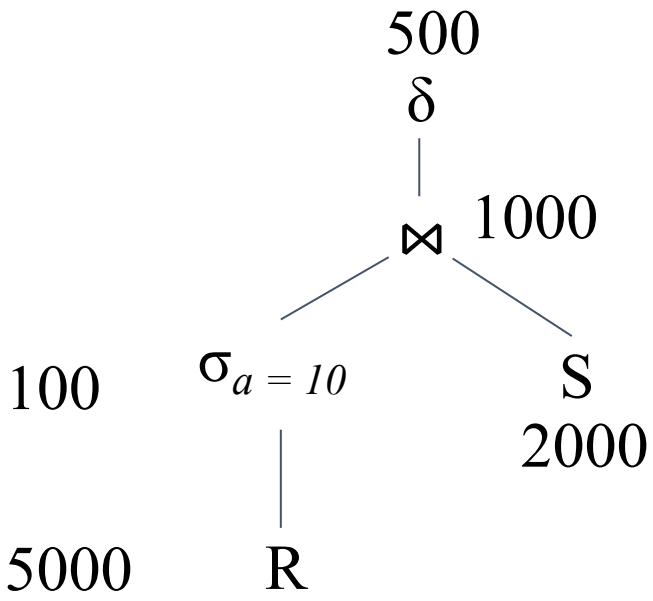
Sampling used to compute approximate statistics quickly

Example:

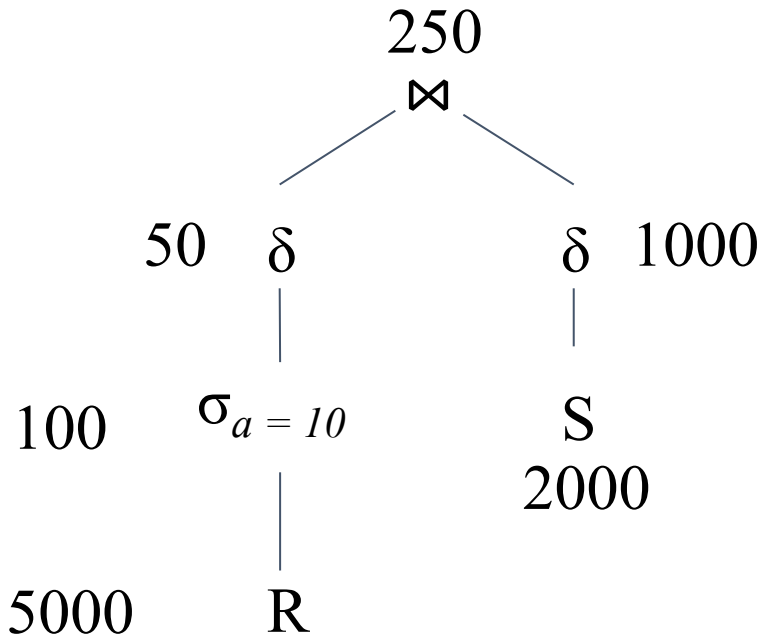
- **`ANALYZE`** command in Postgres
- See also: https://www.postgresql.org/docs/current/planner-stats.html

Comparing logical query plan cost

- Cost estimates (sum of intermediate results) can be used to compare costs before and after transformations



vs.



2. Cost-based Query Optimization

Example: The Index Selection Problem

Input:

- Schema of the database
- **Workload description:** set of (query template, frequency) pairs

Goal: Select a set of indexes that minimize execution time of the workload.

- Cost / benefit balance: Each additional index may help with some queries, but requires updating

This is an optimization problem!

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category = ?
```

Frequency
10,000,000

```
SELECT pname
FROM Product
WHERE year = ? AND Category = ?
AND manufacturer = ?
```

Frequency
10,000,000

Which indexes might we choose?

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category = ?
```

Frequency 10,000,000

```
SELECT pname
FROM Product
WHERE year = ? AND Category = ?
AND manufacturer = ?
```

Frequency 100

Now which indexes might we choose? Worth keeping an index with manufacturer in its search key around?

Simple Heuristic

- Can be framed as standard optimization problem: Estimate how cost changes when we add index.
    - We can ask the optimizer!
- Search over all possible space is too expensive, optimization surface is really nasty.
    - Real DBs may have 1000s of tables!
- Techniques to exploit *structure* of the space.
    - In SQLServer Autoadmin.

NP-hard problem, but can be solved!

Estimating index cost?

- Note that to frame as optimization problem, we first need an estimate of the ***cost*** of an index lookup

- Need to be able to estimate the costs of different indexes / index types…

This mainly depends on getting estimates of result set size!

Ex: Clustered vs. Unclustered

Cost to do a range query for M entries over N-page file (P per page):

- Clustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - To scan: 1 random IO + $\left\lceil \frac{M-1}{P} \right\rceil$ sequential IO
- Unclustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - To scan: ~ M random IO

Suppose we are using a B+ Tree index with:

- Fanout f
- Fill factor 2/3

Plugging in some numbers

To simplify:
- Random IO = ~10ms
- Sequential IO = free

- Clustered:
  - To traverse: $\text{Log}_F(1.5N)$
  - **To scan: 1 random IO + $\left\lceil \frac{M-1}{P} \right\rceil$ sequential IO**

~ 1 random IO = 10ms

- Unclustered:
  - To traverse: $\text{Log}_F(1.5N)$
  - **To scan: ~ M random IO**

~ *M* random IO = M*10ms

- If M = 1, then there is no difference!
- If M = 100,000 records, then difference is ~10min. Vs. 10ms!

We need good estimates of M…

Query Optimization Overview

Output: A good physical query plan

Basic cost-based query optimization algorithm

- Enumerate candidate query plans (logical and physical)
- Compute estimated cost of each plan (e.g., number of I/Os)
  - Without executing the plan!
- Choose plan with lowest cost

The Three Parts of an Optimizer

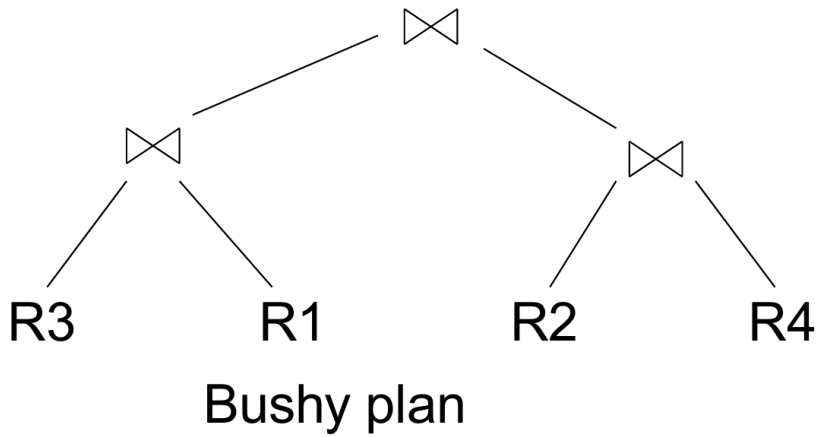
1. Cost estimation
   - Estimate size of results
   - Also consider whether output is sorted/intermediate results written to disk etc.

2. Search space
   - Algebraic laws, restricted types of join trees

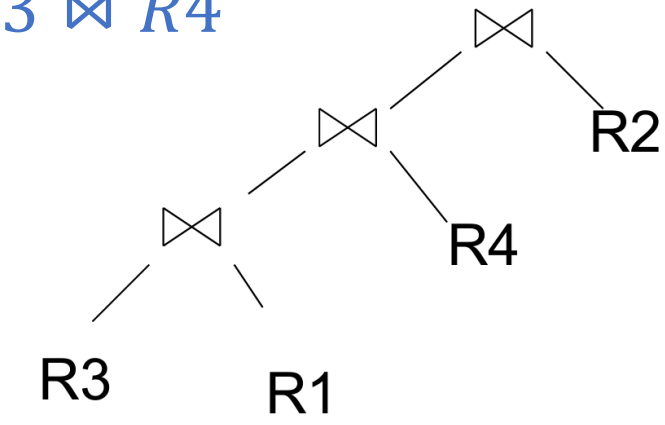
3. Search algorithm
   - Example: Selinger algorithm

2. Search Space

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



Bushy plan



Left-deep plan

Logical plan space:

- Several possible structures of the trees
- Each tree can have n! permutations of relations on leaves

Physical plan space:

- Different implementation (e.g., join algorithm) and scanning of intermediate operators for each logical plan

Heuristic for pruning plan space

Apply predicates as early as possible

Avoid plans with cartesian products

- $(R(A,B) \bowtie T(C,D)) \bowtie \mathrm{S}(B,C)$

Consider only left-deep join trees

- Studied extensively in traditional query optimization literature
- Works well with existing join algorithms such as nested-loop and hash join
  - Left deep: $\big((\mathrm{A} \bowtie B) \bowtie C\big) \bowtie D$
  - Bushy: $(\mathrm{A} \bowtie B) \bowtie (C \bowtie D)$
  - e.g., might not need to write tuples to disk if enough memory

3. Search Algorithm

Selinger Algorithm: dynamic programming based

- Based on System R (aka Selinger) style optimizer [1979]
- Consider different logical and physical plans at the same time
- Limited to joins: join reordering algorithm
- Cost of a plan is I/O + CPU

Exploits ”principle of optimality”

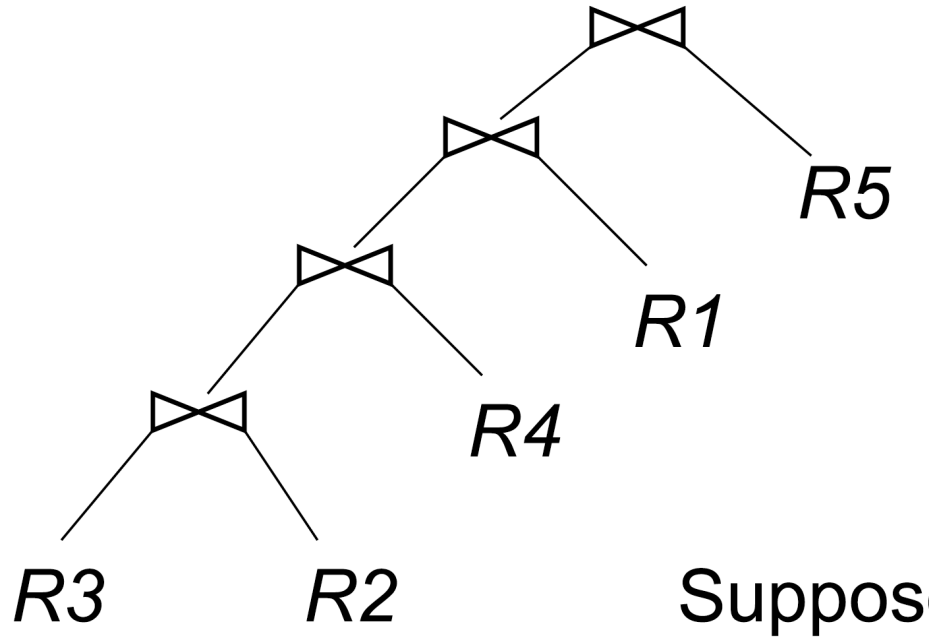
- Optimal for “whole” made up from optimal for ”parts”

Consider the search space of left-deep join trees

- Reduces search space but still n! permutations

Principle of Optimality

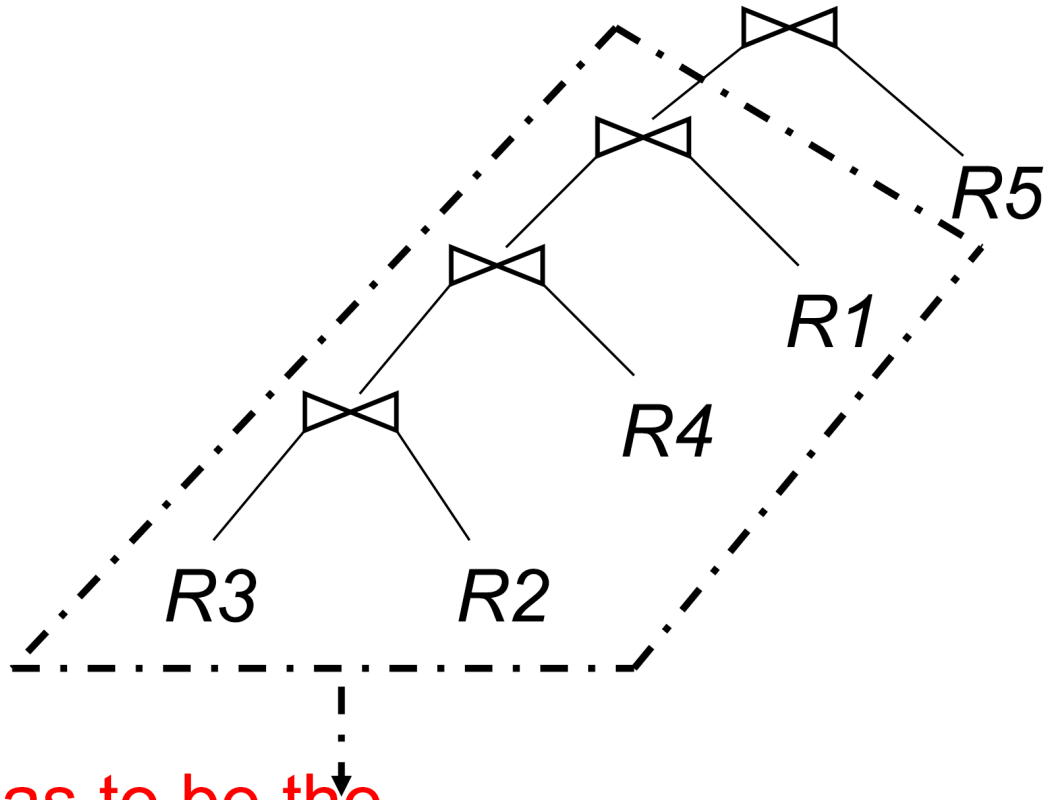
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$



Suppose,
this is an Optimal Plan
for joining R1…R5:

Principle of Optimality

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$



This has to be the
optimal plan for joining *R3, R2, R4, R1*

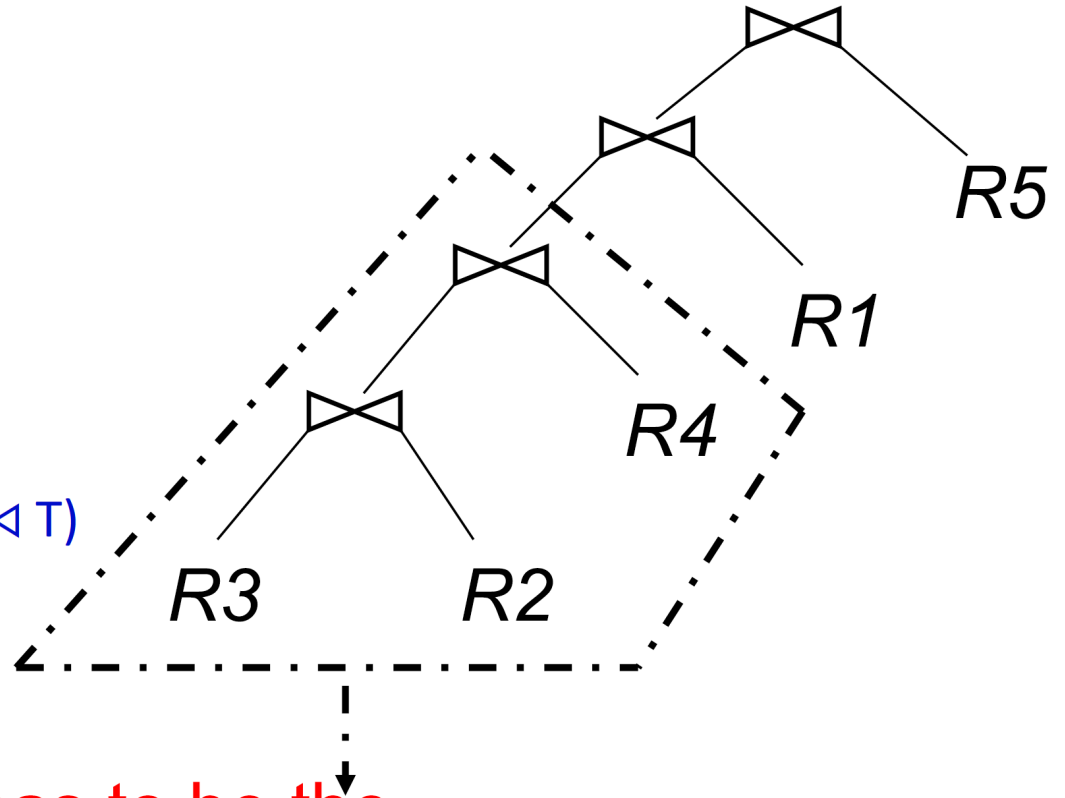
Principle of Optimality

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$

We are using the associativity and commutativity of joins

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \bowtie S = S \bowtie R$



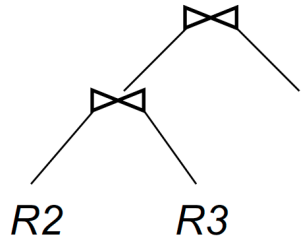
This has to be the optimal plan for joining *R3, R2, R4*

Principle of Optimality

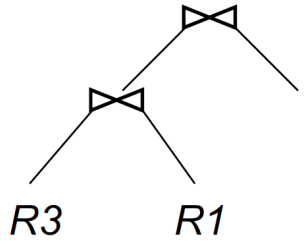
Query: $R1 \bowtie R2 \bowtie \ldots \bowtie Rn$

Both are giving the same result

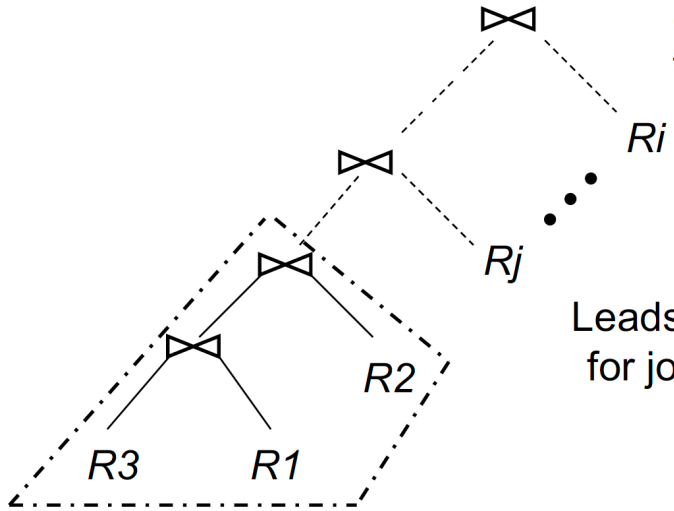
R2 ⋈ R3 ⋈ R1 = R3 ⋈ R1 ⋈ R2



Optimal
for joining *R1, R2, R3*



Sub-Optimal
for joining *R1, R2, R3*



Suppose you chose
the sub-optimal one

Leads to sub-Optimal
for joining R1,…,Rn

Notation and Setup

$\mathrm{OPT}(\{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}\})$:

Cost of optimal plan to join R1, R2, R3

$\mathrm{T}(\{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}\})$:

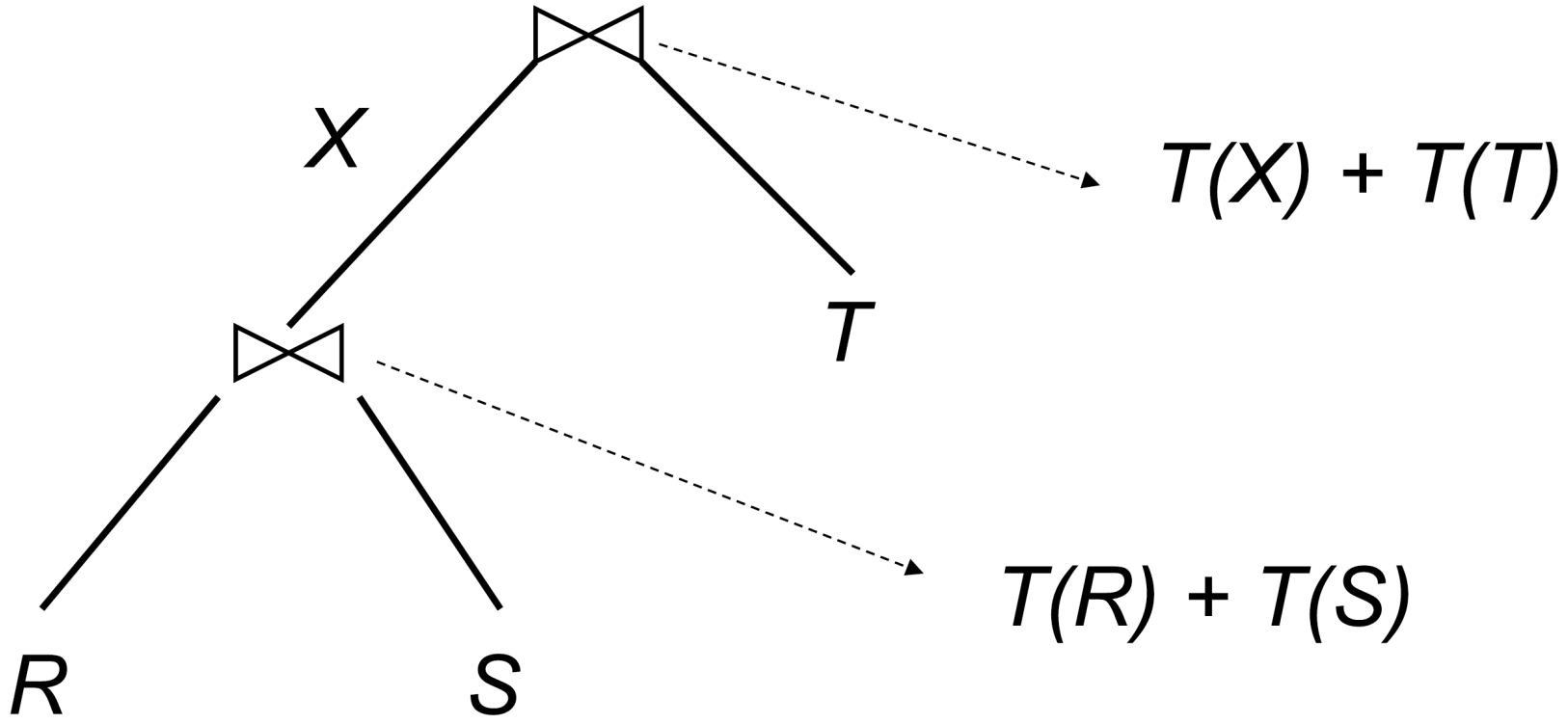
Number of tuples in $R1 \bowtie R2 \bowtie R3$

Simple Cost Model: $\mathbf{Cost}(R \bowtie S) = T(R) + T(S)$

All other operations have 0 cost

* The simple cost model used for illustration only, it is not used in practice

Cost Model Example



Total Cost: $T(R) + T(S) + T(T) + T(X)$

Selinger Algorithm

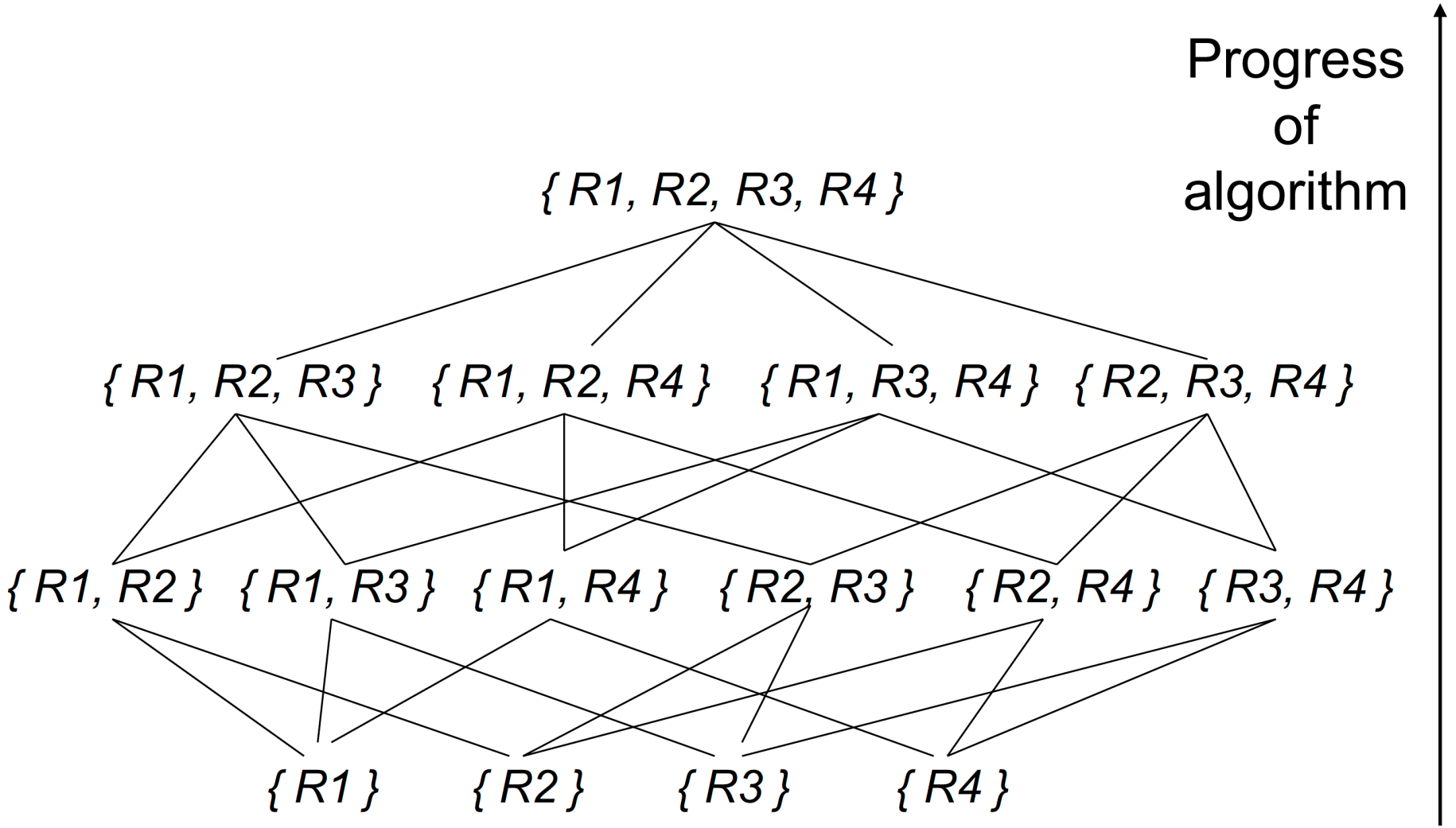
Bellman Equation:

* Valid only for the simple cost model

$$\mathrm{OPT}(\{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}\}) = min \begin{cases} \mathrm{OPT}(\{\mathrm{R1}, \mathrm{R2}\}) + \mathrm{T}(\{\mathrm{R1}, \mathrm{R2}\}) + \mathrm{T}(\mathrm{R3}) \\ \mathrm{OPT}(\{\mathrm{R2}, \mathrm{R3}\}) + \mathrm{T}(\{\mathrm{R2}, \mathrm{R3}\}) + \mathrm{T}(\mathrm{R1}) \\ \mathrm{OPT}(\{\mathrm{R1}, \mathrm{R3}\}) + \mathrm{T}(\{\mathrm{R1}, \mathrm{R3}\}) + \mathrm{T}(\mathrm{R2}) \end{cases}$$

Selinger Algorithm

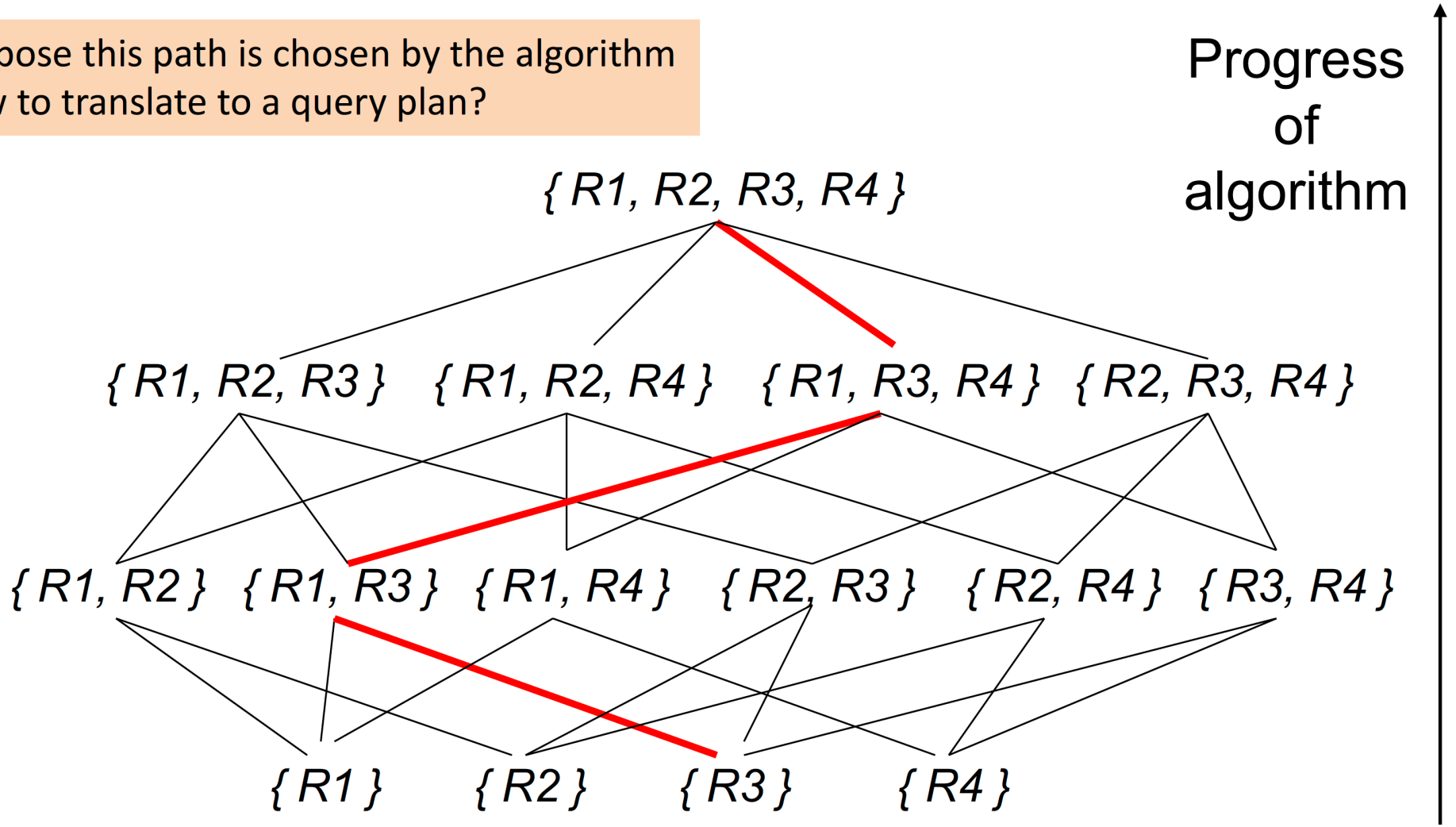
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



Selinger Algorithm

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

Suppose this path is chosen by the algorithm
How to translate to a query plan?

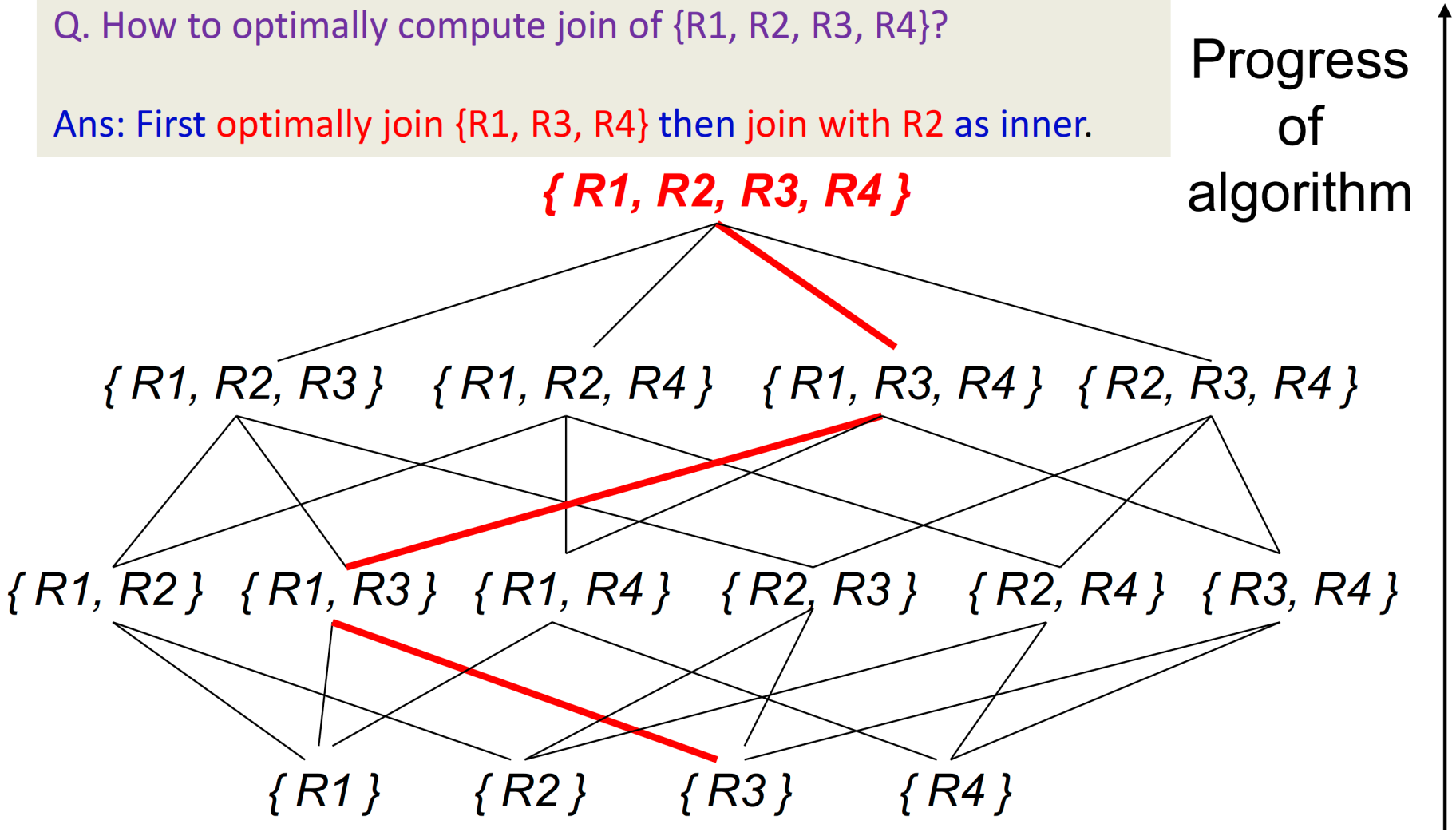


Selinger Algorithm

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

Q. How to optimally compute join of {R1, R2, R3, R4}?

Ans: First optimally join {R1, R3, R4} then join with R2 as inner.

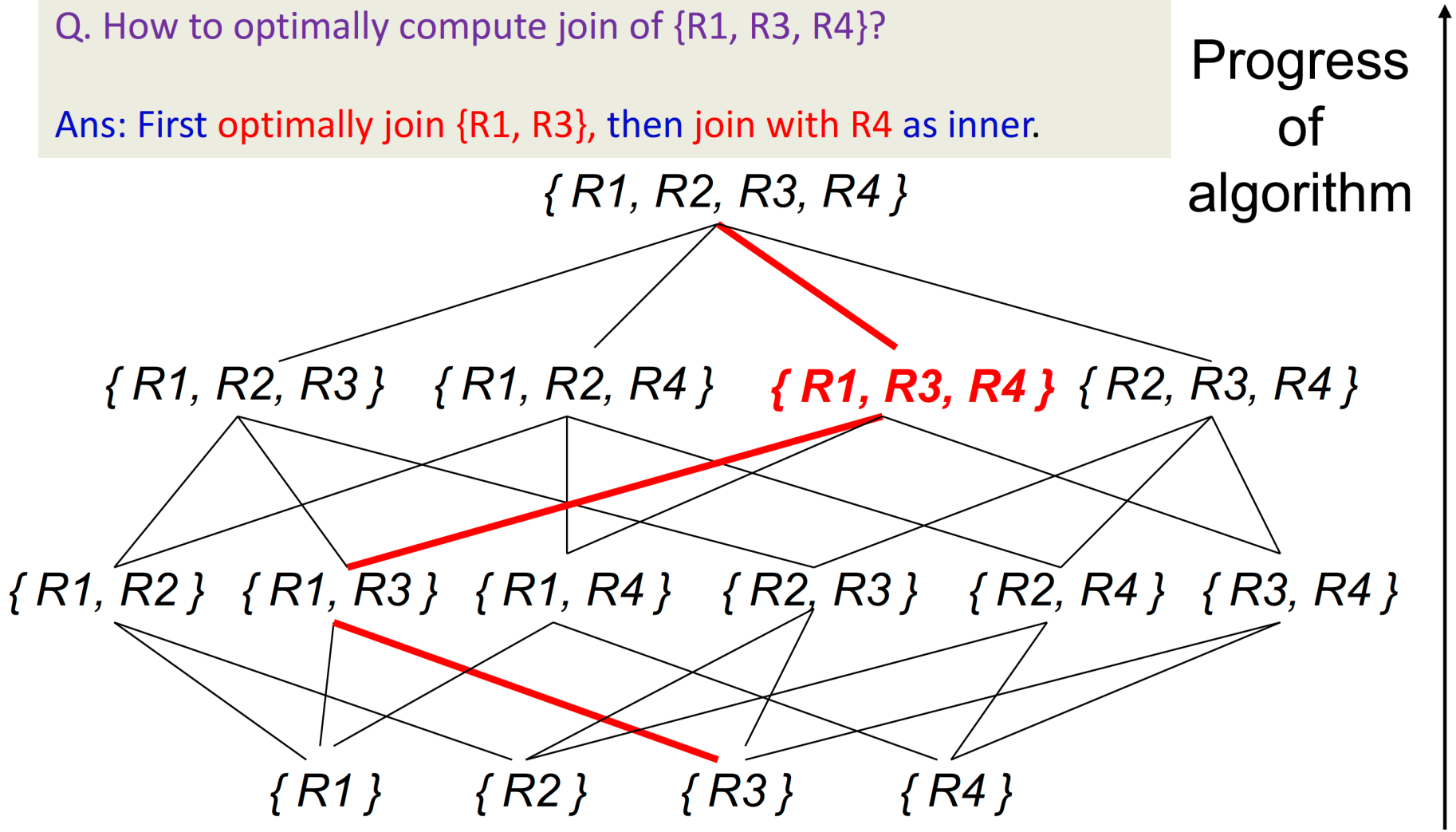


Selinger Algorithm

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

Q. How to optimally compute join of {R1, R3, R4}?

Ans: First optimally join {R1, R3}, then join with R4 as inner.

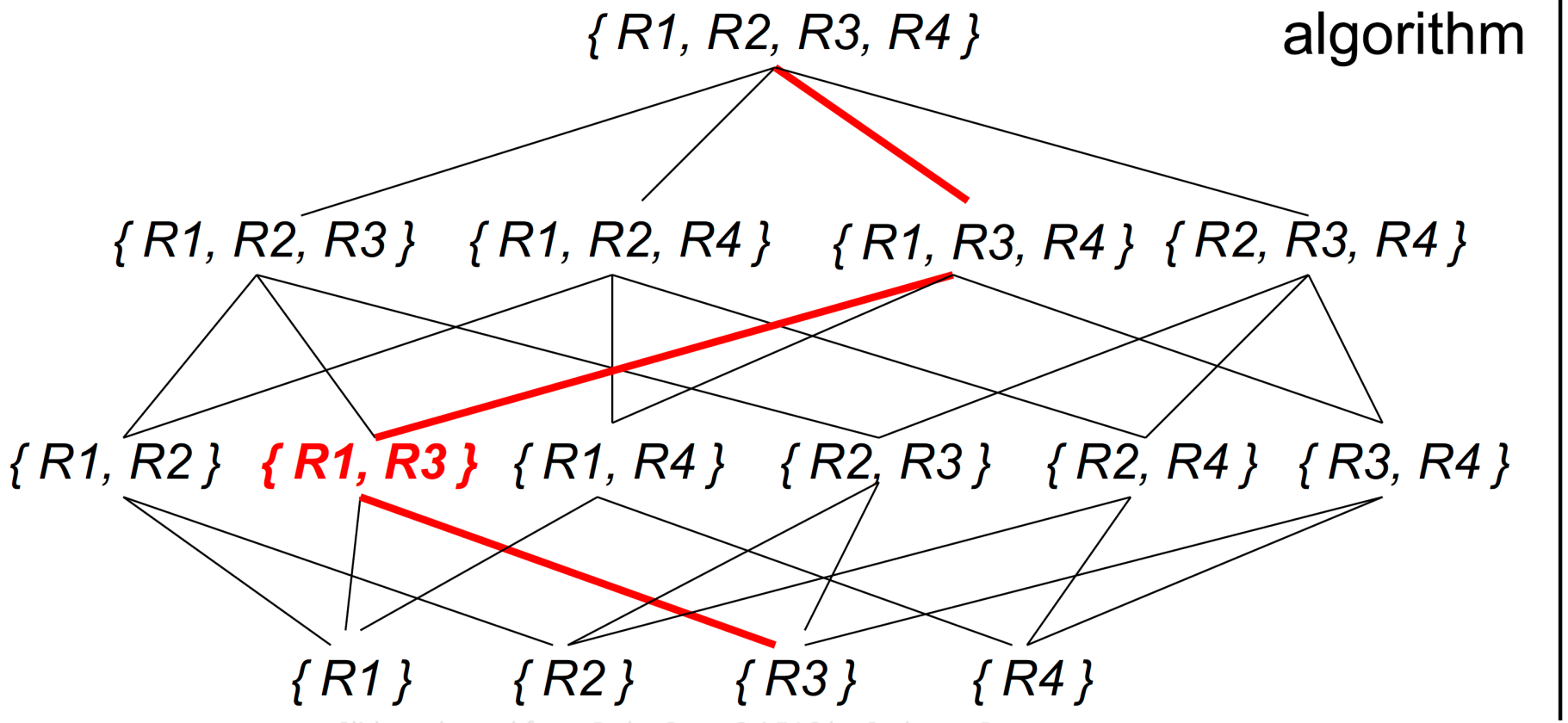


Selinger Algorithm

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

Q. How to optimally compute join of {R1, R3}?

Ans: First optimally join {R3}, then join with R1 as inner.

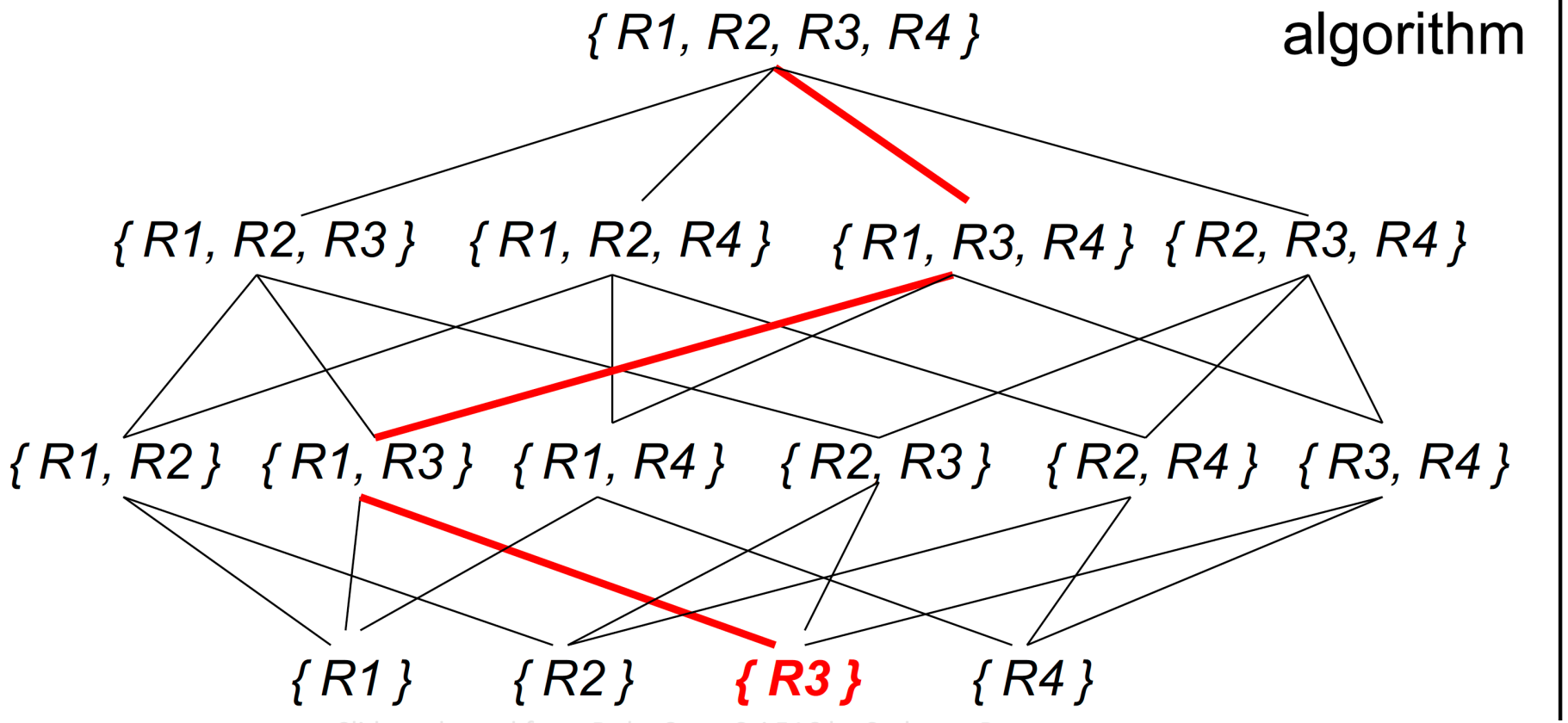


Selinger Algorithm

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

Q. How to optimally compute join of {R3}?

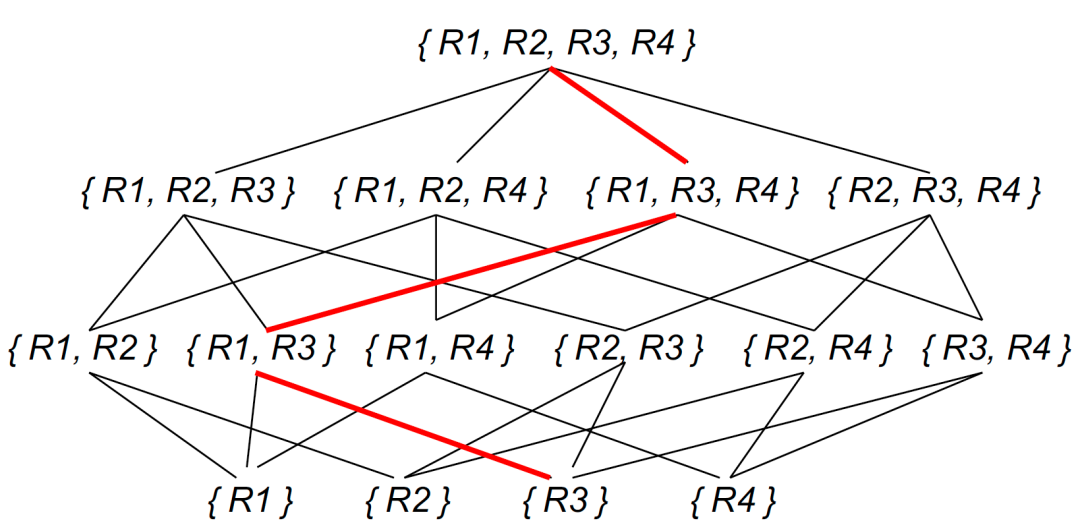
Ans: Single relation – so optimally scan R3.

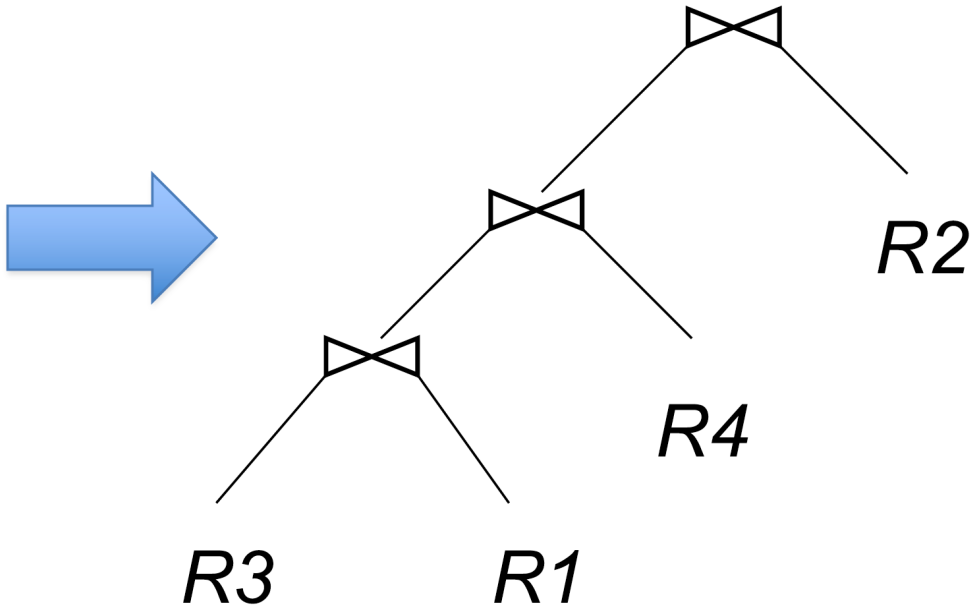


Selinger Algorithm

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

Final optimal plan:





NOTE : There is a one-one correspondence between the permutation (R3, R1, R4, R2) and the above left deep plan

Slides adapted from Duke CompSci 516 by Sudeepa Roy

Selinger Algorithm

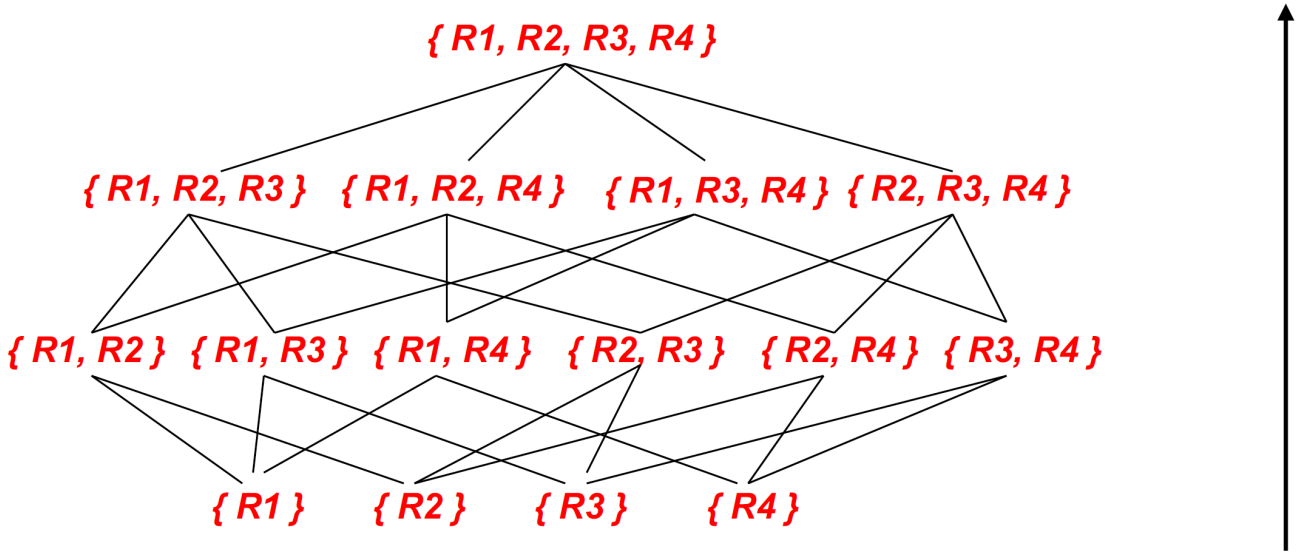
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

NOTE:

This is *NOT* done by top-down recursive calls.

- This is done BOTTOM-UP computing the optimal cost of *all* nodes in this lattice only once (dynamic programming).

Progress of algorithm



Reduces n! to 2^n

Slides adapted from Duke CompSci 516 by Sudeepa Roy

Try it yourself

`EXPLAIN` command: Display the execution plan that the PostgreSQL planner generates for the supplied statement.

```
EXPLAIN SELECT * FROM foo;

                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```

```
EXPLAIN SELECT * FROM foo WHERE i = 4;

                         QUERY PLAN
-------------------------------------------------------------------
 Index Scan using fi on foo  (cost=0.00..5.98 rows=1 width=4)
   Index Cond: (i = 4)
(2 rows)
```

Source: https://www.postgresql.org/docs/current/sql-explain.html

Example: PostgreSQL EXPLAIN

```
postgres=# EXPLAIN SELECT items.* FROM line_items AS items LEFT JOIN orders ON items.order_id = orders.id
LEFT JOIN users ON users.id = orders.user_id WHERE users.name = 'Karisa';
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------
 Gather  (cost=1204.46..1046121.82 rows=259359 width=22)
   Workers Planned: 2
   ->  Hash Join  (cost=204.46..1019185.92 rows=108066 width=22)
         Hash Cond: (items.order_id = orders.id)
         ->  Parallel Seq Scan on line_items items  (cost=0.00..931789.57 rows=22964057 width=22)
         ->  Hash  (cost=204.02..204.02 rows=35 width=4)
               ->  Hash Join  (cost=20.68..204.02 rows=35 width=4)
                     Hash Cond: (orders.user_id = users.id)
                     ->  Seq Scan on orders  (cost=0.00..163.65 rows=7465 width=8)
                     ->  Hash  (cost=20.62..20.62 rows=4 width=4)
                           ->  Seq Scan on users  (cost=0.00..20.62 rows=4 width=4)
                                 Filter: (name = 'Karisa'::text)
(12 rows)
```

Source: https://www.timescale.com/learn/explaining-postgresql-explain

Example: PostgreSQL EXPLAIN

Add indexes for FKs:

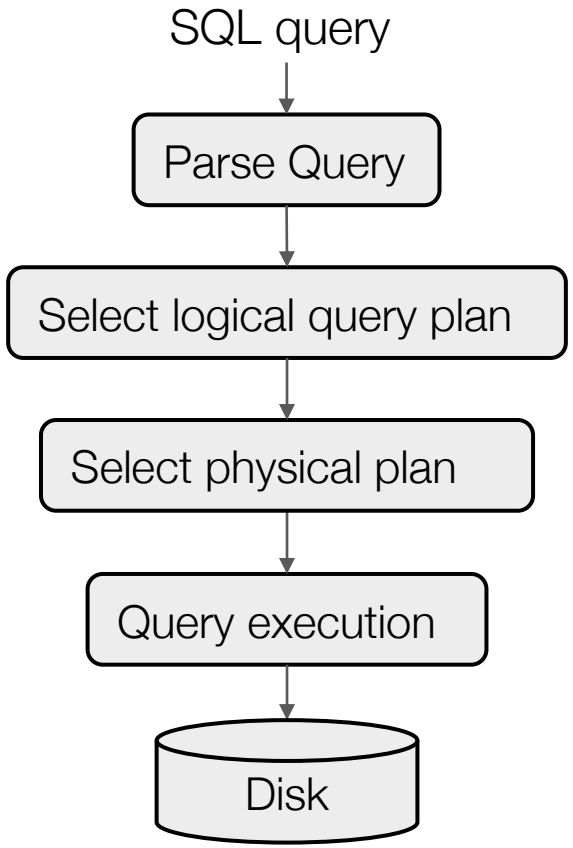
- line_items (order_id)
- orders (user_id)

```
postgres=# EXPLAIN SELECT items.* FROM line_items AS items LEFT JOIN orders ON items.order_id =
orders.id LEFT JOIN users ON users.id = orders.user_id WHERE users.name = 'Karisa';
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.57..162613.06 rows=26128 width=19)
   ->  Nested Loop  (cost=0.28..409.26 rows=3732 width=4)
         Join Filter: (orders.user_id = users.id)
         ->  Index Scan using orders_pkey on orders  (cost=0.28..296.26 rows=7465 width=8)
         ->  Materialize  (cost=0.00..1.03 rows=1 width=4)
               ->  Seq Scan on users  (cost=0.00..1.02 rows=1 width=4)
                     Filter: (name = 'Karisa'::text)
   ->  Index Scan using idx_line_item_orders on line_items items  (cost=0.29..43.35 rows=11 width=19)
         Index Cond: (order_id = orders.id)
(9 rows)
```

Source: https://www.timescale.com/learn/explaining-postgresql-explain

Putting it all together: RDBMS Architecture

How does a SQL engine work ?



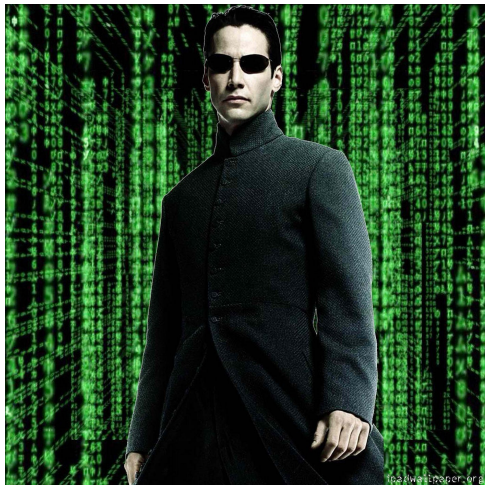
Translate to RA expression and find logically equivalent but more efficient plans

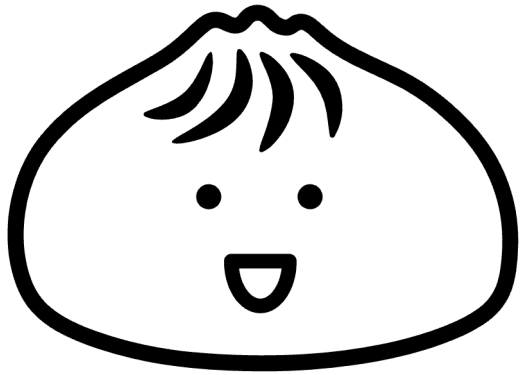
Cost-based query optimization: estimate cost and select physical plan with the smallest cost

Query execution (e.g., run join algorithms against tuples on disk)

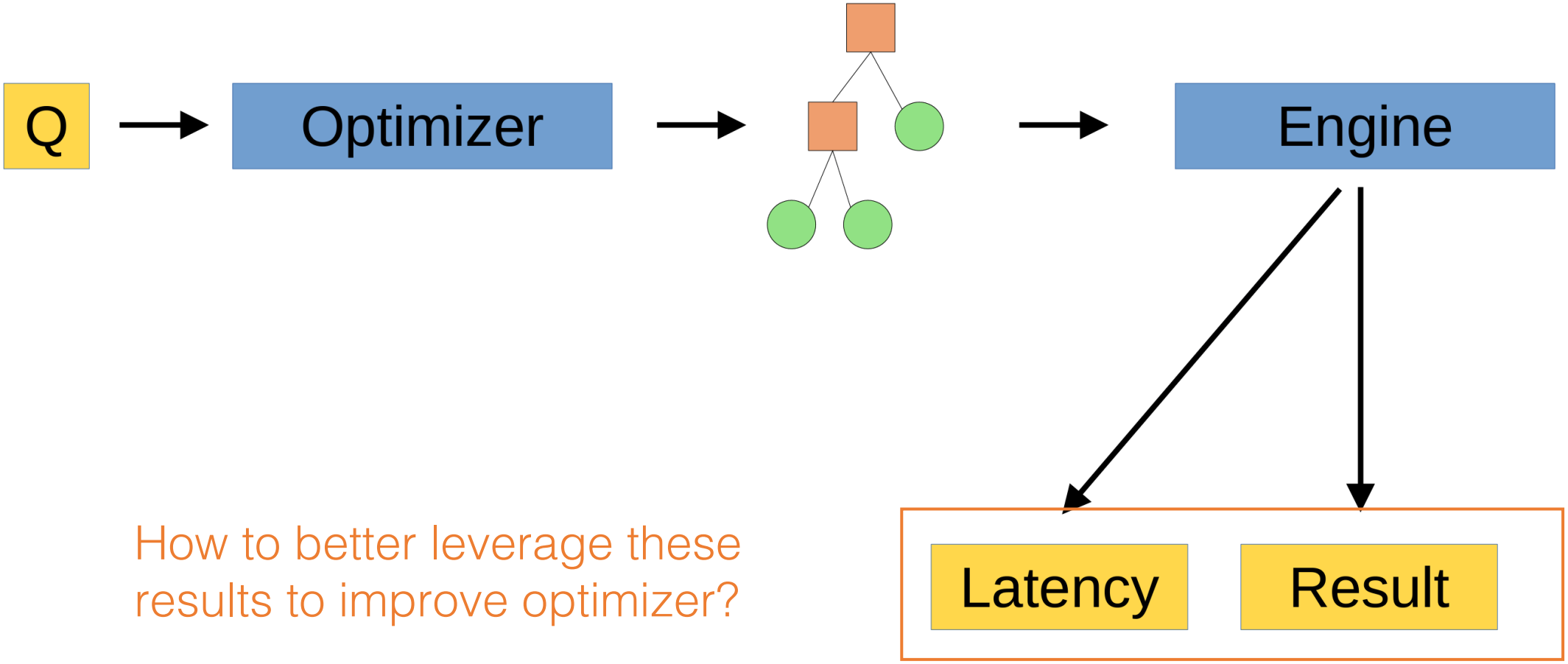
A brief intro to learned query optimizers

Slides adapted from *Machine Learning for Query Optimization … and beyond!* by Ryan Marcus





Query Optimization



How to better leverage these results to improve optimizer?

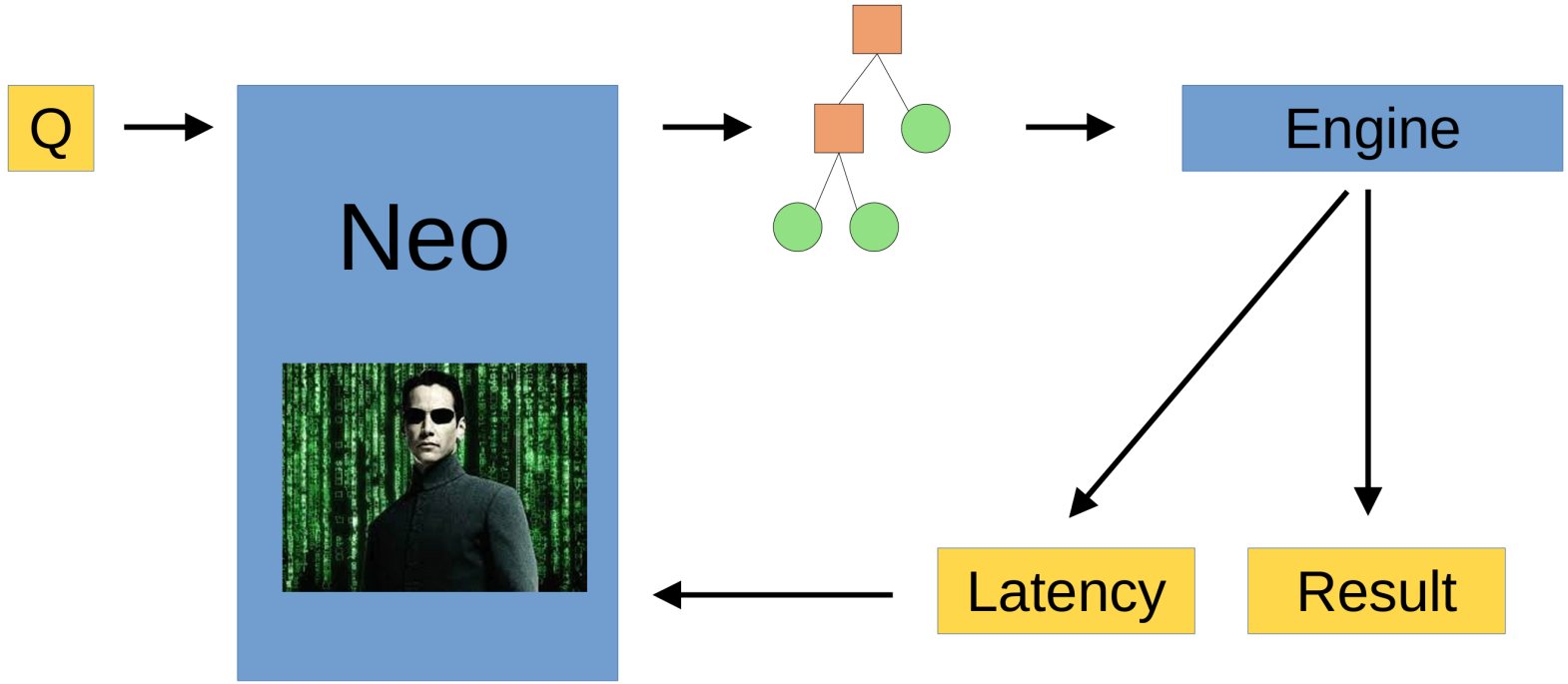
Source: Ryan Marcus

Neo: A Learned Query Optimizer [VLDB'19]

Complete replacement of default query optimizer

First to show we can have *all learned everything*

Deep reinforcement learning guided search



Source: Ryan Marcus

Neo: A Learned Query Optimizer. VLDB '19

Neo: A Learned Query Optimizer [VLDB'19]

Neo worked great on average but

Sample Inefficiency

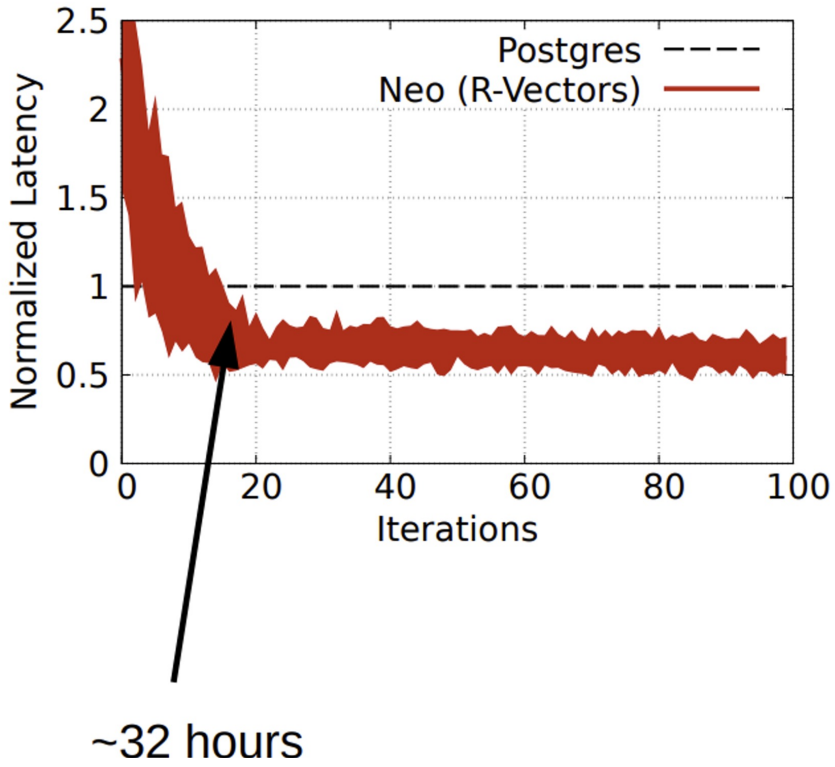
- Typically takes > 1 day for pre-train

Brittleness to workload and schema change

- The encoding of cardinality estimate needs retrain

Tail catastrophe

- Deep RL making wrong estimates due to sample inefficiency



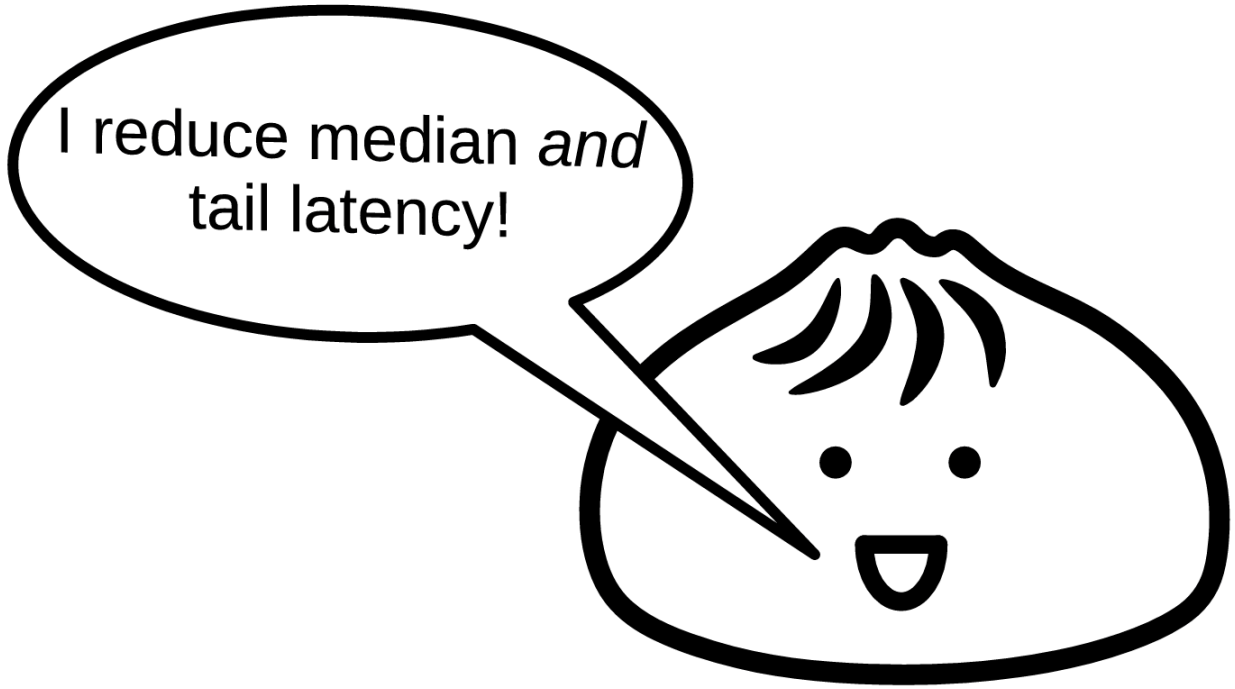
Source: Ryan Marcus

Bao: Making Learned Query Optimization Practical [SIGMOD'21]

Bao: Bandit optimizer

By steering a traditional query optimizer, Bao:

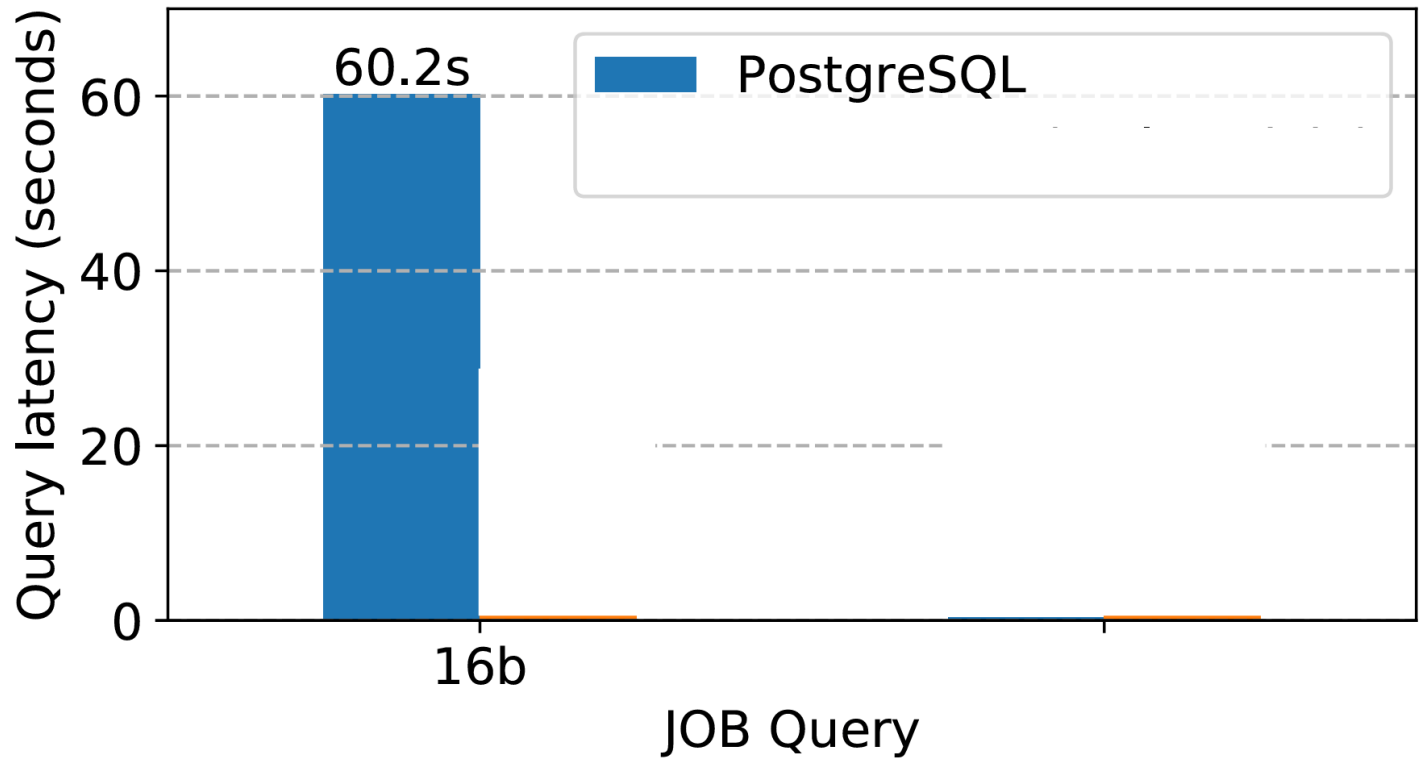
- Outperforms PG after 1 hour of training
- Reduces 99% latency
- Adaptsto changes in workload, schema, and data.



Source: Ryan Marcus

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

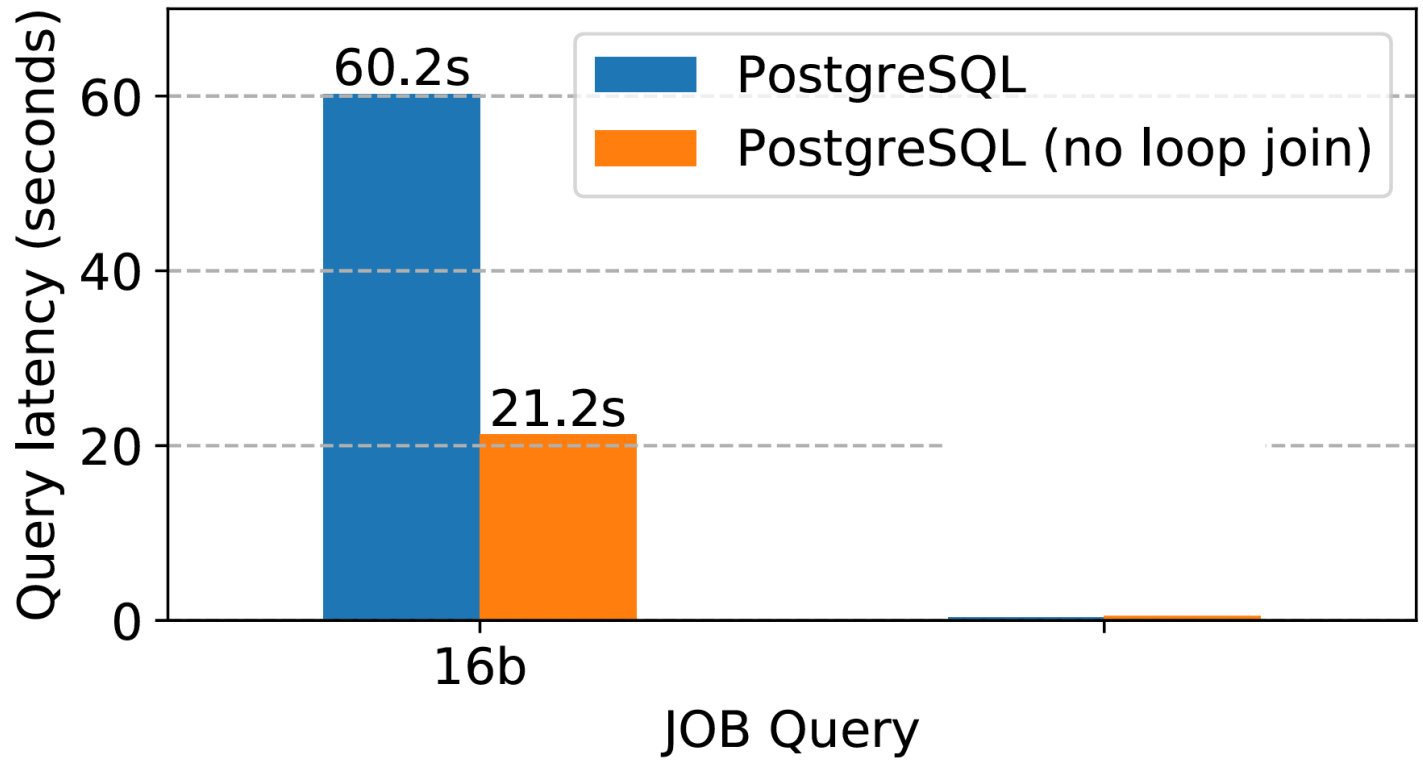


Source: Ryan Marcus

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement



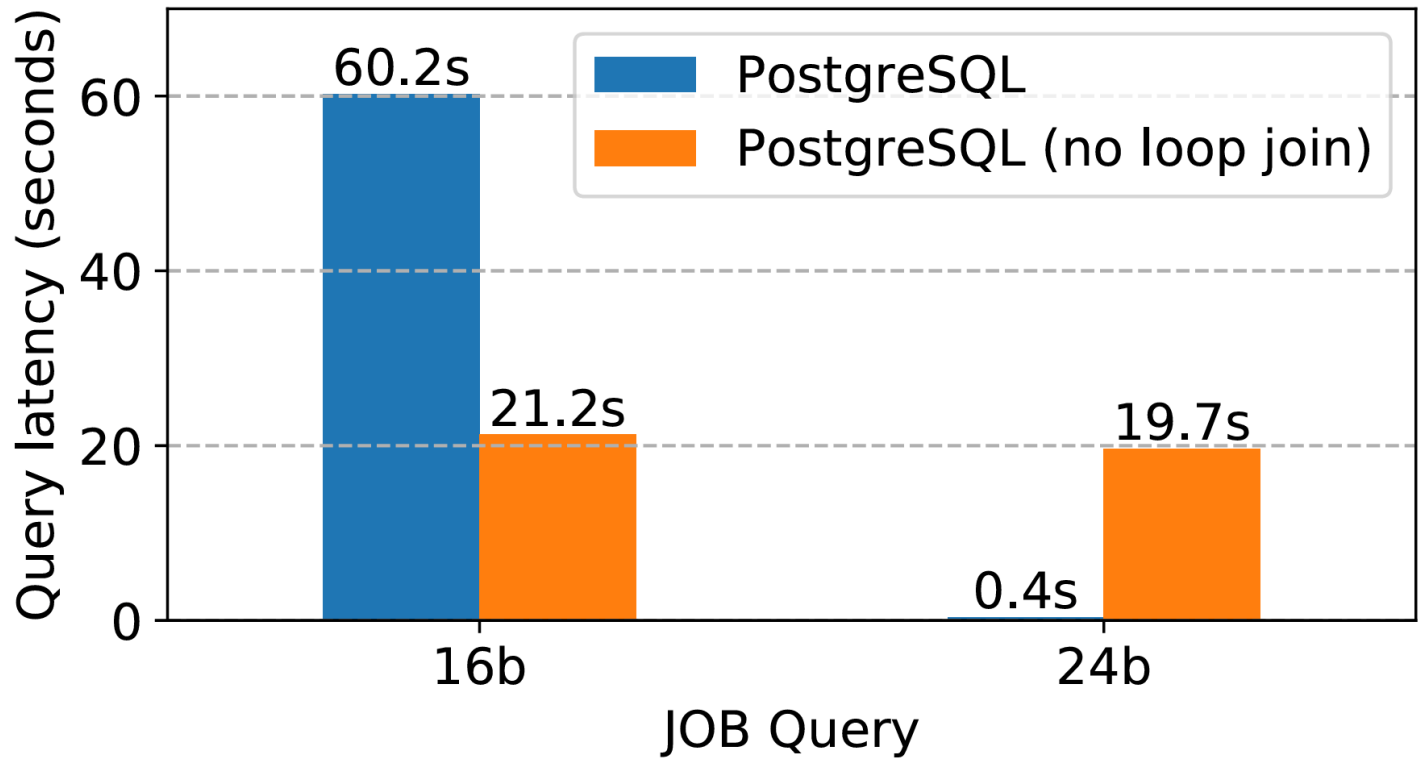
Source: Ryan Marcus

Query Hints

Slow query. Run EXPLAIN.
> Loop join plan,
> Low selectivity

Try disabling loop join
> Huge improvement

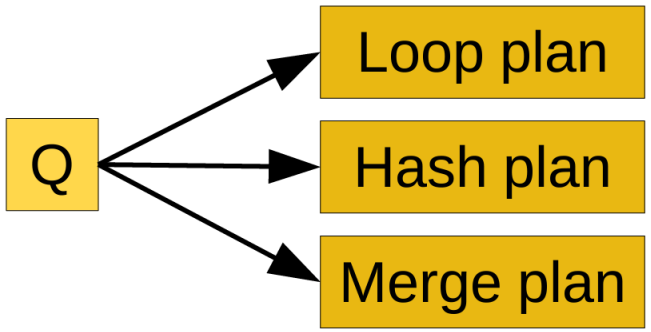
Apply this hint globally
> … other regressions



Source: Ryan Marcus

Bao

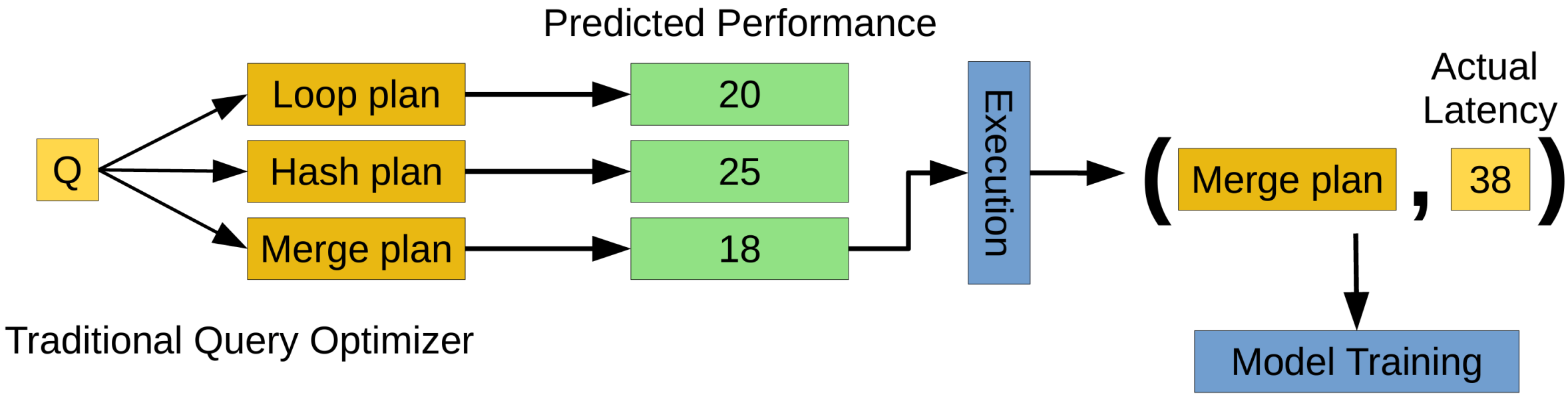
- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



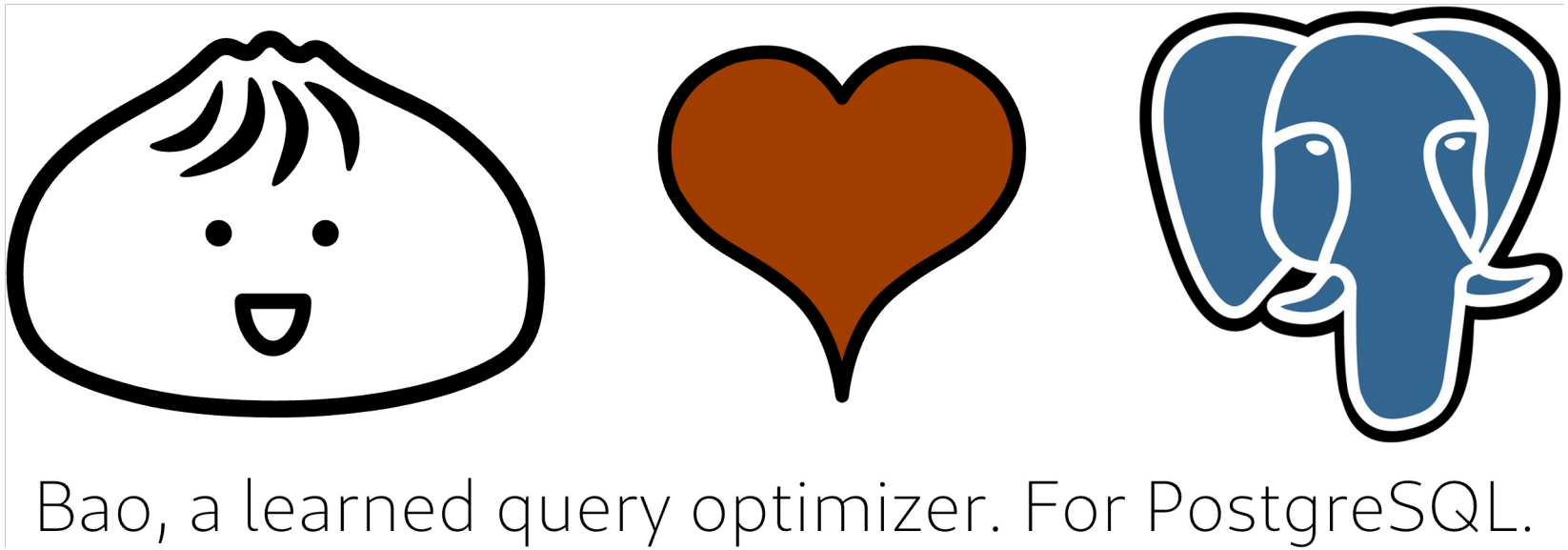
Traditional Query Optimizer

Bao

- Bao automatically determines the right hint to use.
- Consider different hints as *arms* in a *contextual multi-armed bandit*



PostgreSQL Integration



Github: https://github.com/learnedsystems/BaoForPostgreSQL