CS 6400 A

Database Systems Concepts and Design

Lecture 13

10/07/24

Announcements

- Assignment 1 grades will be released today
- No class or OH this Wednesday (Oct 9)
- No class next Monday (Oct 14)

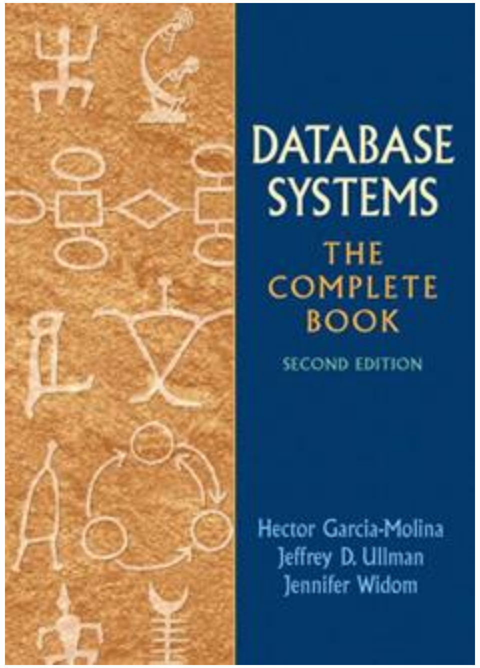
Reading Materials

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.

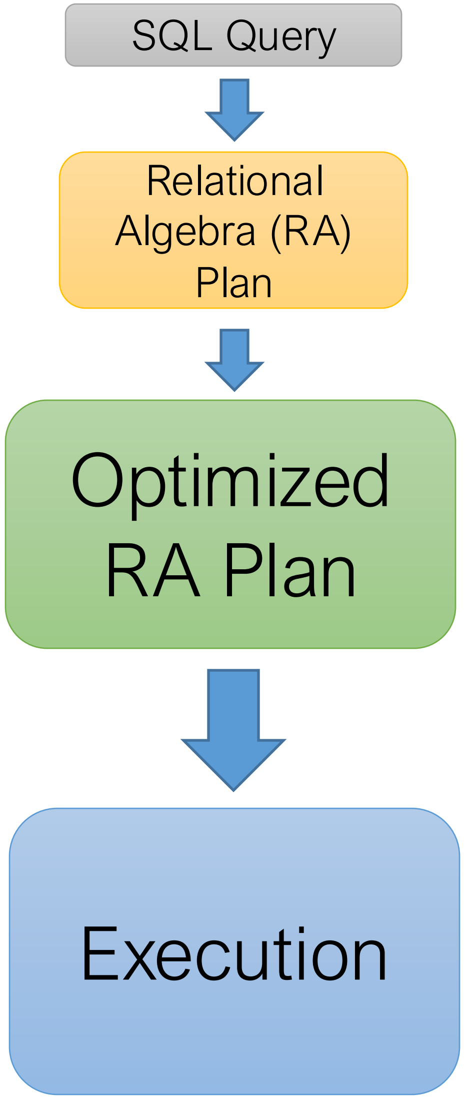
Agenda

1. Physical Optimization

2. Estimating cost of a physical plan

3. Cost-based Query Optimization

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. Physical Optimization

Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

$$\pi_{starName}$$

$$\sigma_{year\ =\ 2008\ AND\ studioName\ =\ \text{‘Ghibli’}}$$

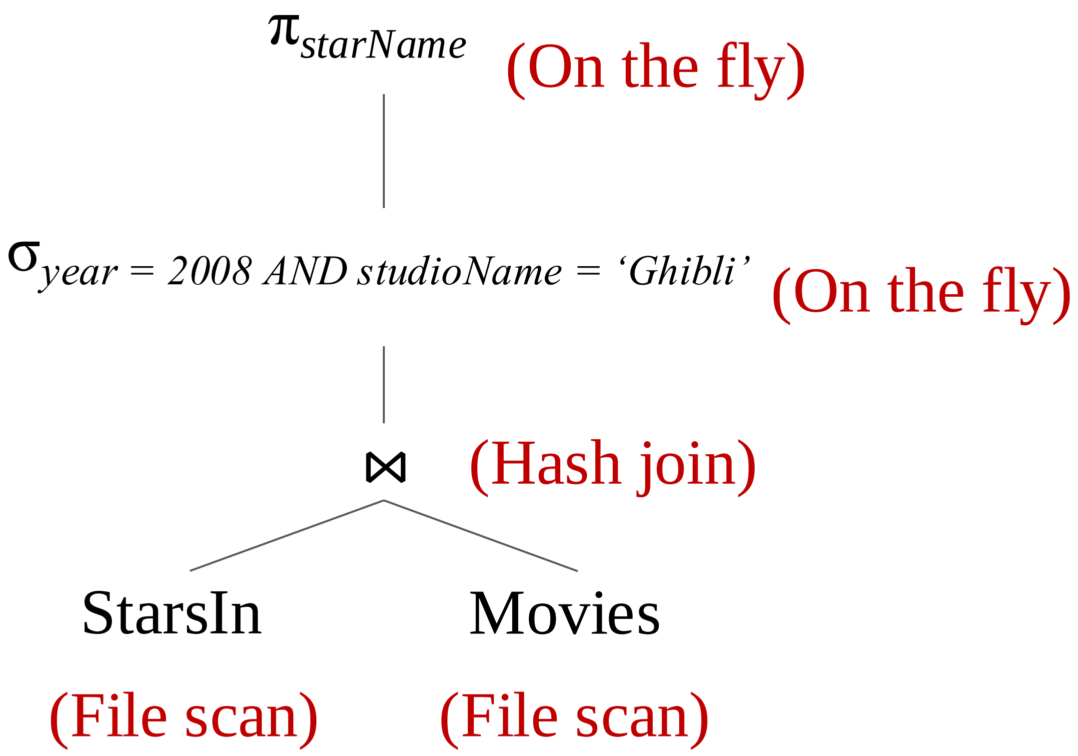
$$\bowtie$$

StarsIn Movies

Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations



Physical query plan 1

Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

Physical query plan 2

$\pi_{starName}$ (On the fly)

$\sigma_{year\ =\ 2008\ AND\ studioName\ =\ \text{‘Ghibli’}}$ (On the fly)

$\bowtie$ (Nested loop join)

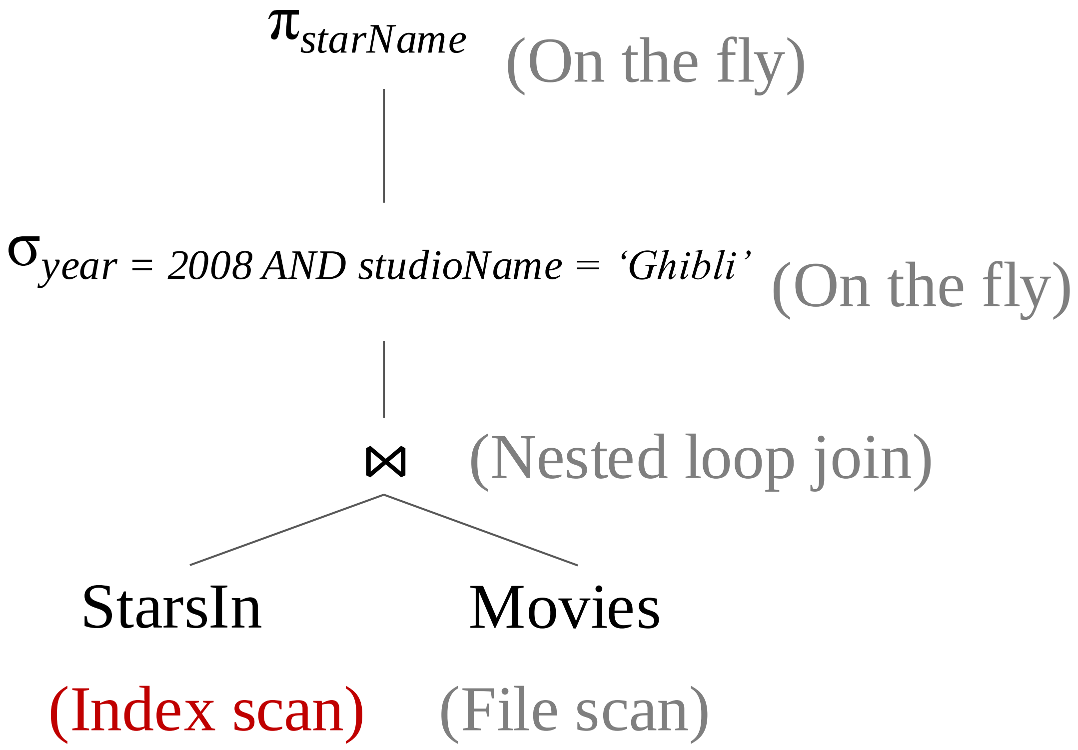
StarsIn (File scan) | Movies (File scan)

Select physical query plan

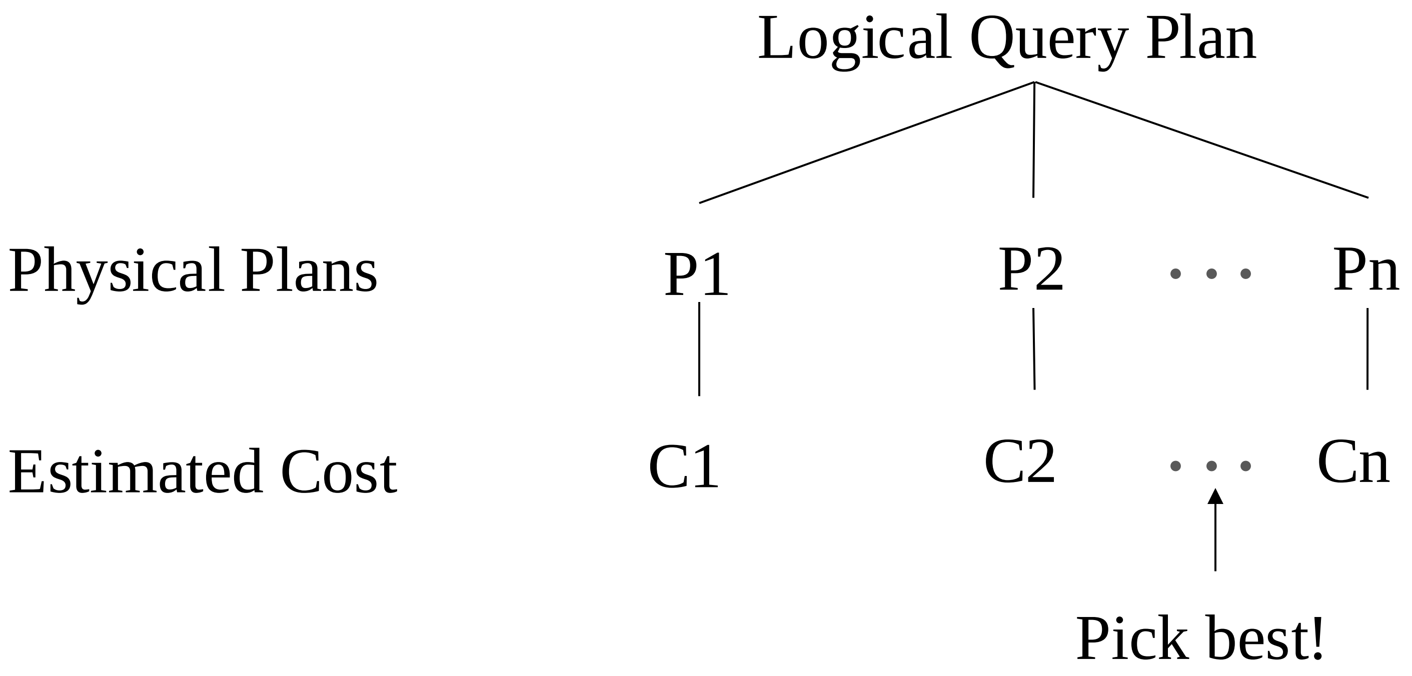
A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

Physical query plan 3



Select physical query plan



In general, there can be many possible physical plans

Query execution

$\pi_{starName}$ (On the fly)

$\sigma_{year\ =\ 2008\ AND\ studioName\ =\ \text{‘Ghibli’}}$ (On the fly)

$\bowtie$ (Nested loop join)

StarsIn (File scan)

Movies (File scan)

Machine Code (e.g., C)

The best physical plan is translated to actual machine code

2. 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

$B(R)$: # blocks 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 block is 100 bytes
Then a block fits 4 tuples
If $T(R) = 1000$
Then $B(R) = 1000 / 4 = 250$

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

Estimating size of selection

A selection generally reduces the number of tuples

Estimated result size
(without any additional information)

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

*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 $\mathrm{T}(S)$?

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

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 $\mathrm{T}(S)$?

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

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: $R \bowtie S \bowtie T$

$R(A, B)$	$S(B, C)$	$T(C, D)$
$T(R) = 1000$	$T(S) = 2000$	$T(T) = 5000$
$V(R, B) = 20$	$V(S, B) = 50$	$V(T, C) = 500$
	$V(S, C) = 100$	$V(T, D) = 200$

Q: What is $T(R \bowtie S)$ and $V(R \bowtie S, C)$?

Joins of many relations

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

$R(A, B)$	$S(B, C)$	$R \bowtie S(A, B, C)$
$T(R) = 1000$ $V(R, B) = 20$	$T(S) = 2000$ $V(S, B) = 50$ $V(S, C) = 100$	$T(R \bowtie S) = 40000$ $V(R \bowtie S, C) = 100$

Joins of many relations

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

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

$T(C, D)$

$(R \bowtie S) \bowtie T$

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

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

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

Joins of many relations

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

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

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

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

$R(A, B, C)$

$S(B, C, D)$

$R \bowtie S$

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

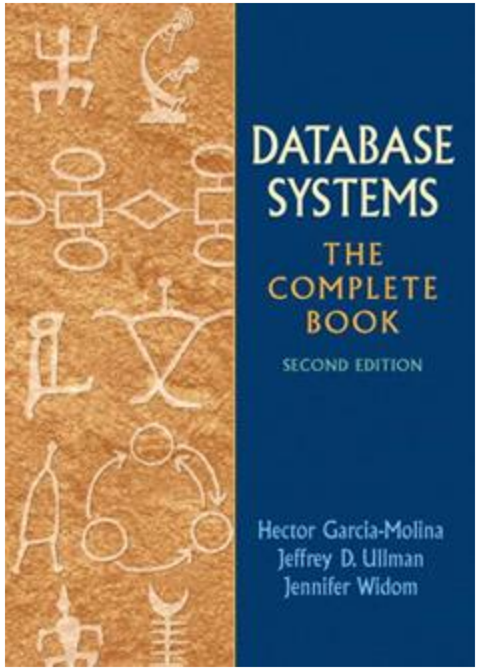
$$T(S) = 2000$$
$$V(S, B) = 50$$
$$V(S, C) = 50$$

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

Further reading

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

- Chapter 16.4.7

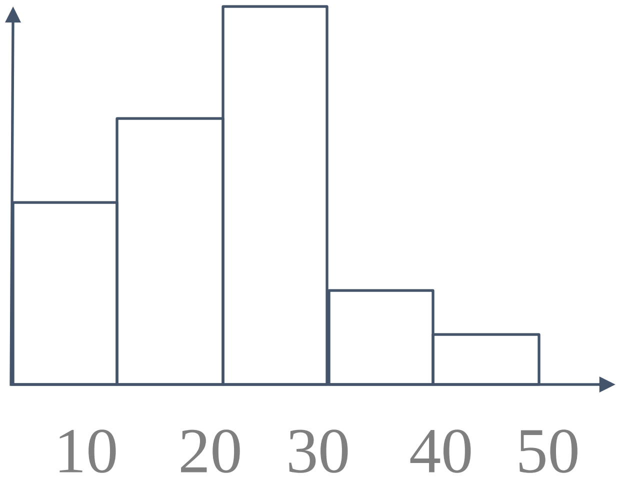


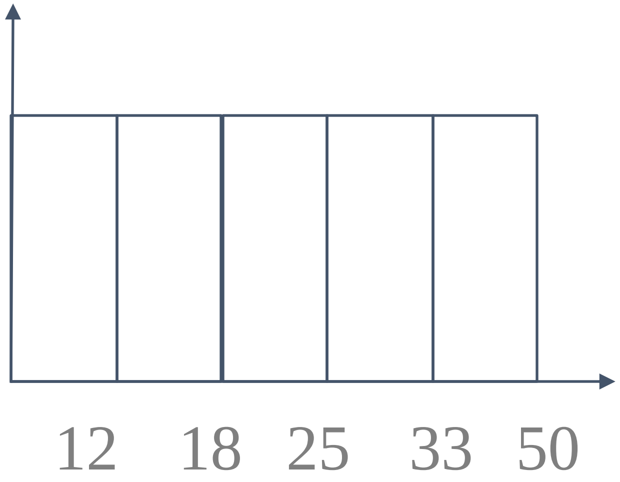
Obtaining estimates for size parameters

Scan entire relation R to obtain $T(R)$, $V(R, A)$, and $B(R)$

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

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

Ex: Clustered vs. Unclustered Index

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

Suppose we are using a B+ Tree index with:
- Fanout f
- Fill factor 2/3

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

Unclustered:
- To traverse: $Log_f(1.5N)$
- To scan: ~ M random IO

Ex: Nested-loop Join

Suppose (from estimates):

- T(R) = 10,000, T(S) = 5,000

Suppose 10 records fit in one block:

- B(R)=1000, B(S)=500

Compute $\mathrm{R} \bowtie S \text{ on } A$:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

$$\mathrm{B(R) + T(R)*B(S) + OUT}$$

For each tuple in R, read all S blocks and join:

Cost(R ⋈ S): 1000 + 10000 x 500 = 5,001,000 I/O's

Memory usage: 2 blocks

Ex: Block Nested-loop Join

Suppose (from estimates):

- T(R) = 10,000, T(S) = 5,000

Suppose 10 records fit in one block:

- B(R)=1000, B(S)=500

Extra memory M=101:

- read 100 blocks of S at a time

Compute R $\bowtie S\ on\ A$:

```
for each M-1 pages pr of R:
 for page ps of S:
  for each tuple r in pr:
   for each tuple s in ps:
    if r[A] == s[A]:
     yield (r,s)
```

$$\mathrm{B}(R) + \frac{B(R)}{M-1} B(S) + \mathrm{OUT}$$

Total cost of S $\bowtie$ R: 500 + 500/100 x 1000) = 5500 I/O's

Memory Usage: M blocks

3. Cost-based Query Optimization

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

Cost estimation

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

Search space

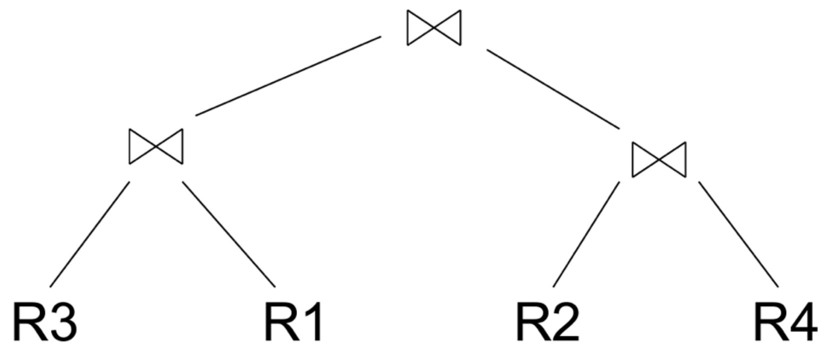
- Algebraic laws, restricted types of join trees

Search algorithm

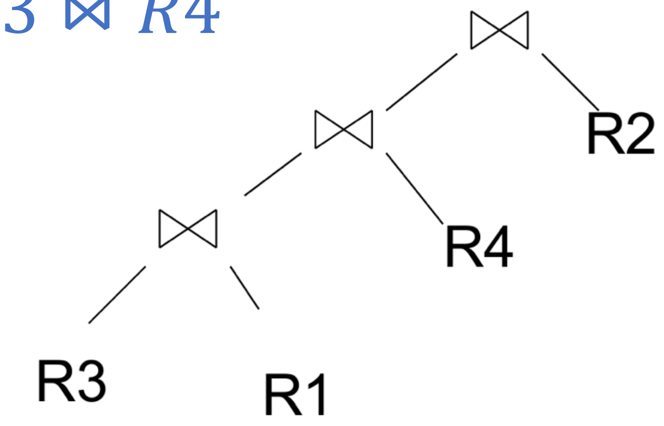
- Example: Selinger algorithm

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
  - e.g., might not need to write tuples to disk if enough memory

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”

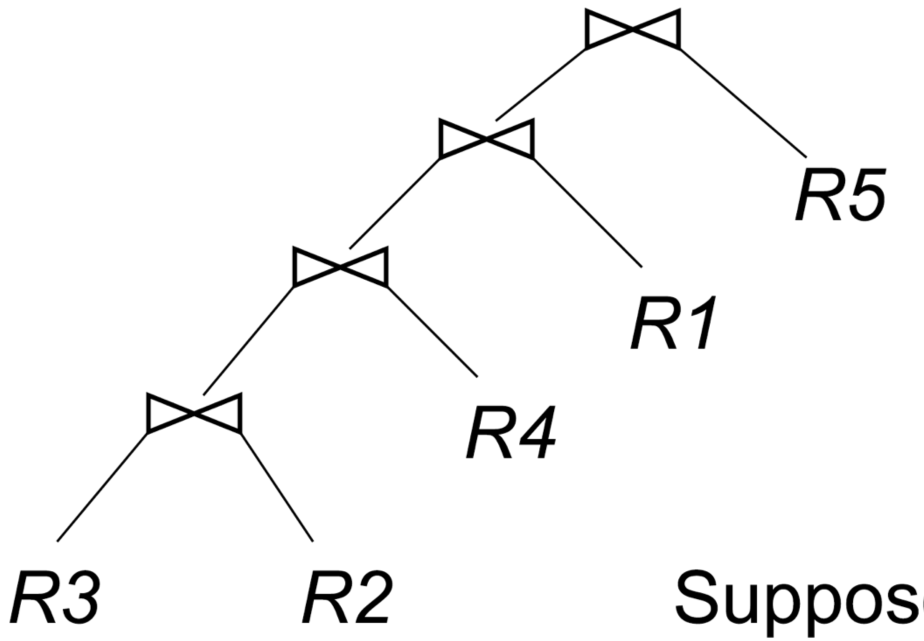
- Optimal for “whole” made up from optimal for ”parts”

Consider the search space of left-deep join trees

- Reduces search space but still n! permutations

Principle of Optimality

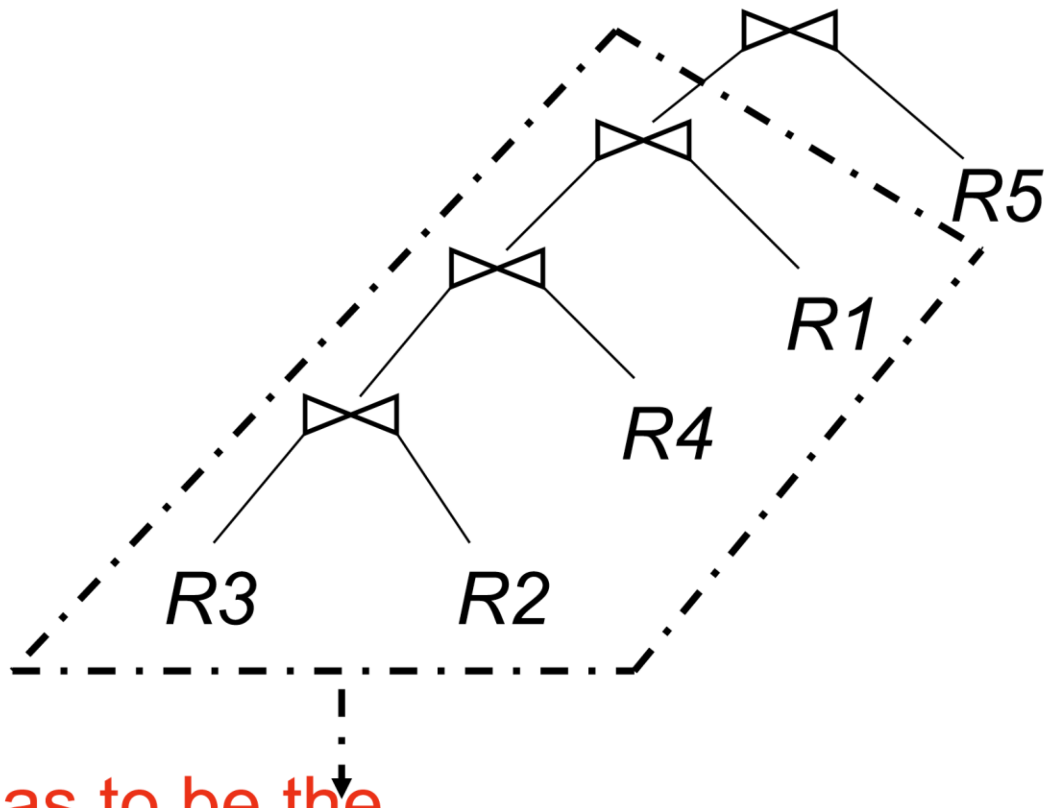
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$



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

Principle of Optimality

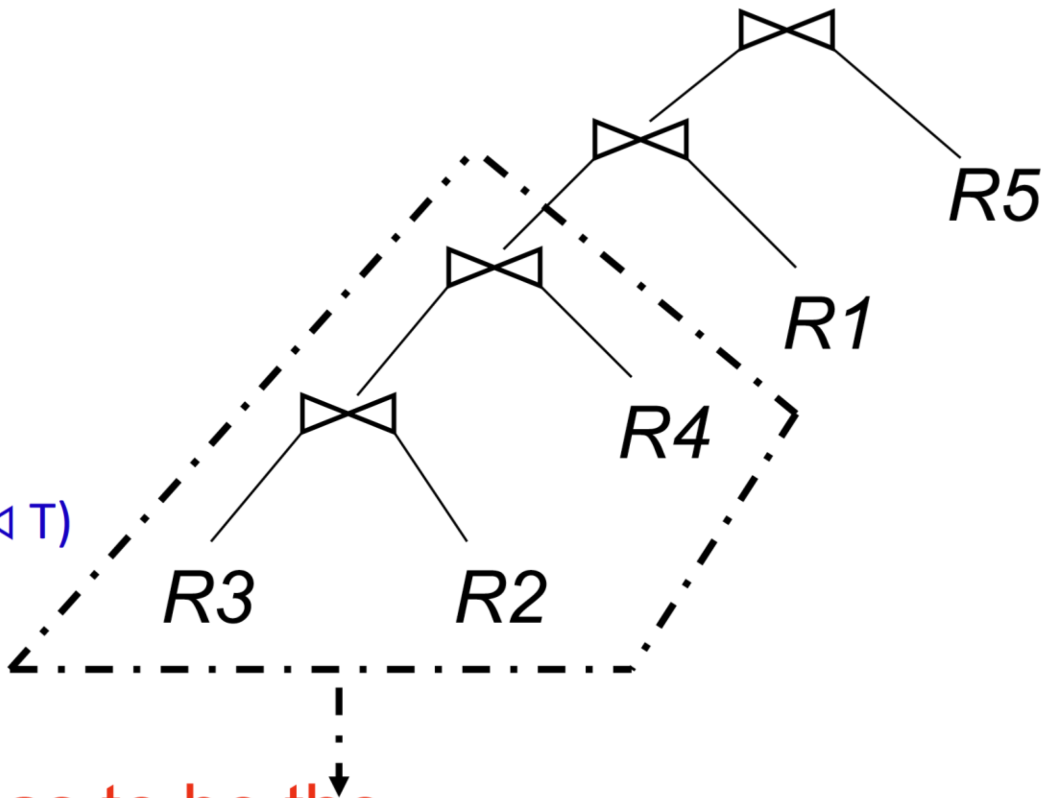
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$



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

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

$R \bowtie S = S \bowtie R$

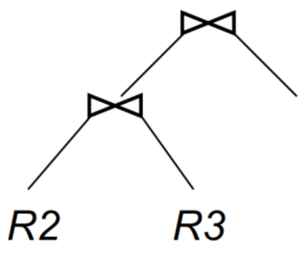
This has to be the optimal plan for joining *R3, R2, R4*

Principle of Optimality

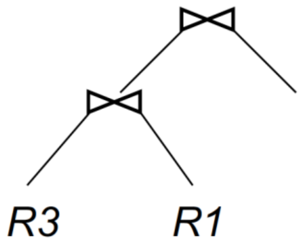
Query: $R1 \bowtie R2 \bowtie \quad \dots \quad \bowtie Rn$

Both are giving the same result

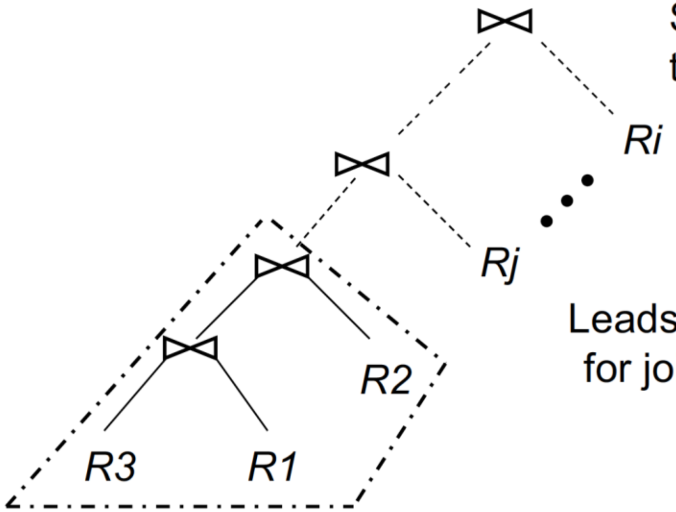
R2 ⋈ R3 ⋈ R1 = R3 ⋈ R1 ⋈ R2



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

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

Cost of optimal plan to join R1, R2, R3

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

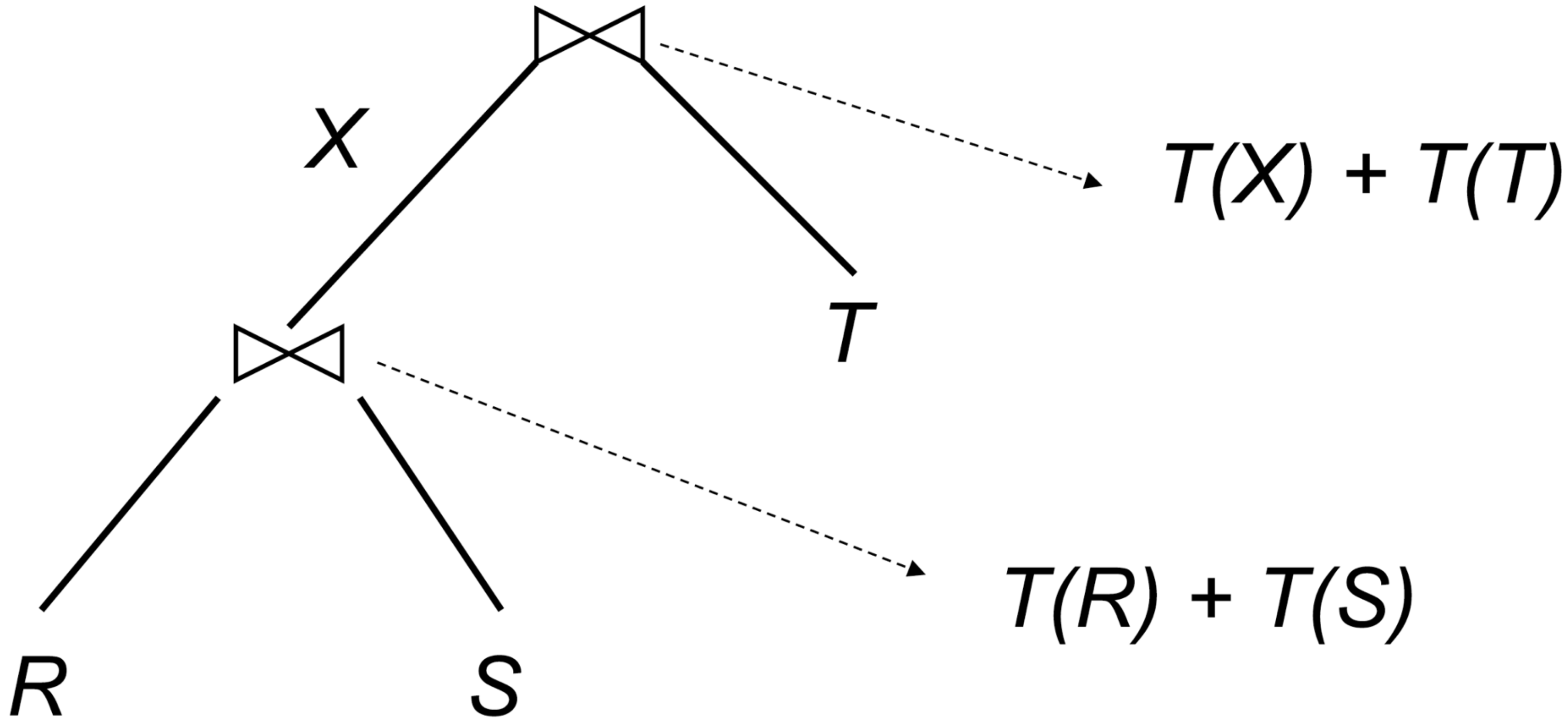
Number of tuples in $R1 \bowtie R2 \bowtie R3$

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

All other operations have 0 cost

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

Cost Model Example



$$\boxed{\text{Total Cost: } T(R) + T(S) + T(T) + T(X)}$$

Slides adapted from Duke CompSci 516 by Sudeepa Roy

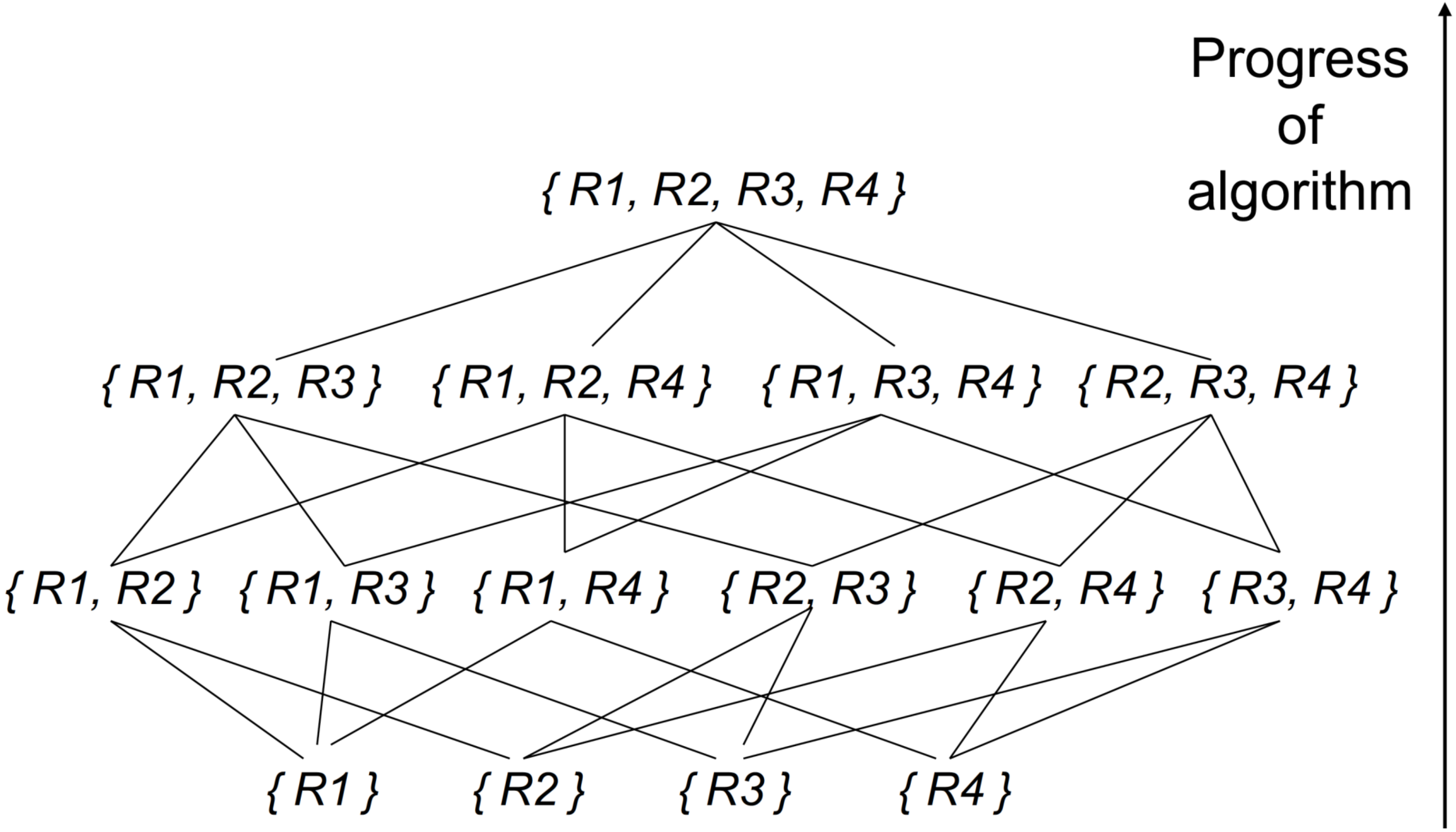
Selinger Algorithm

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

* Valid only for the simple cost model

Selinger Algorithm

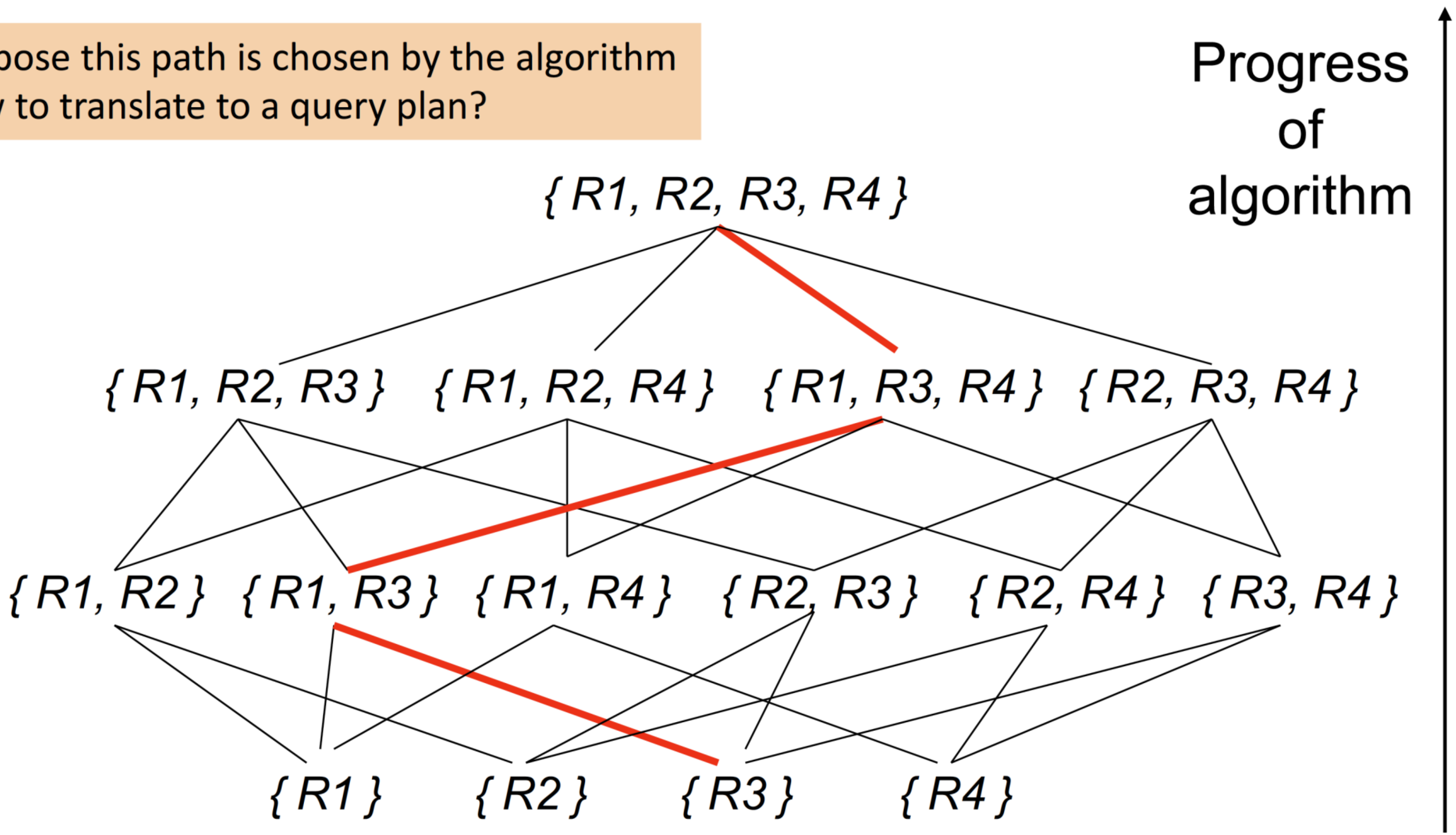
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



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?

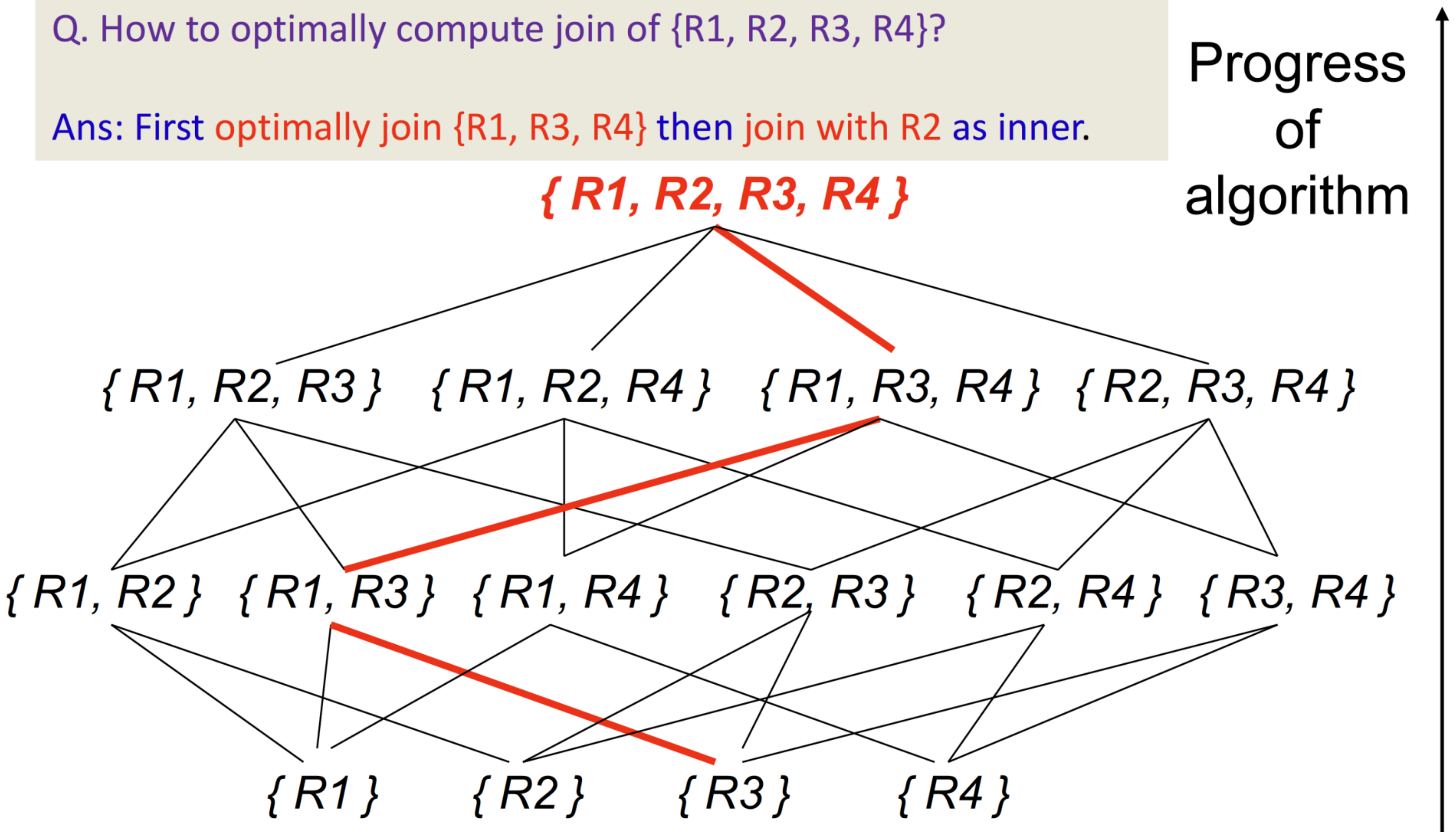


Selinger Algorithm

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.

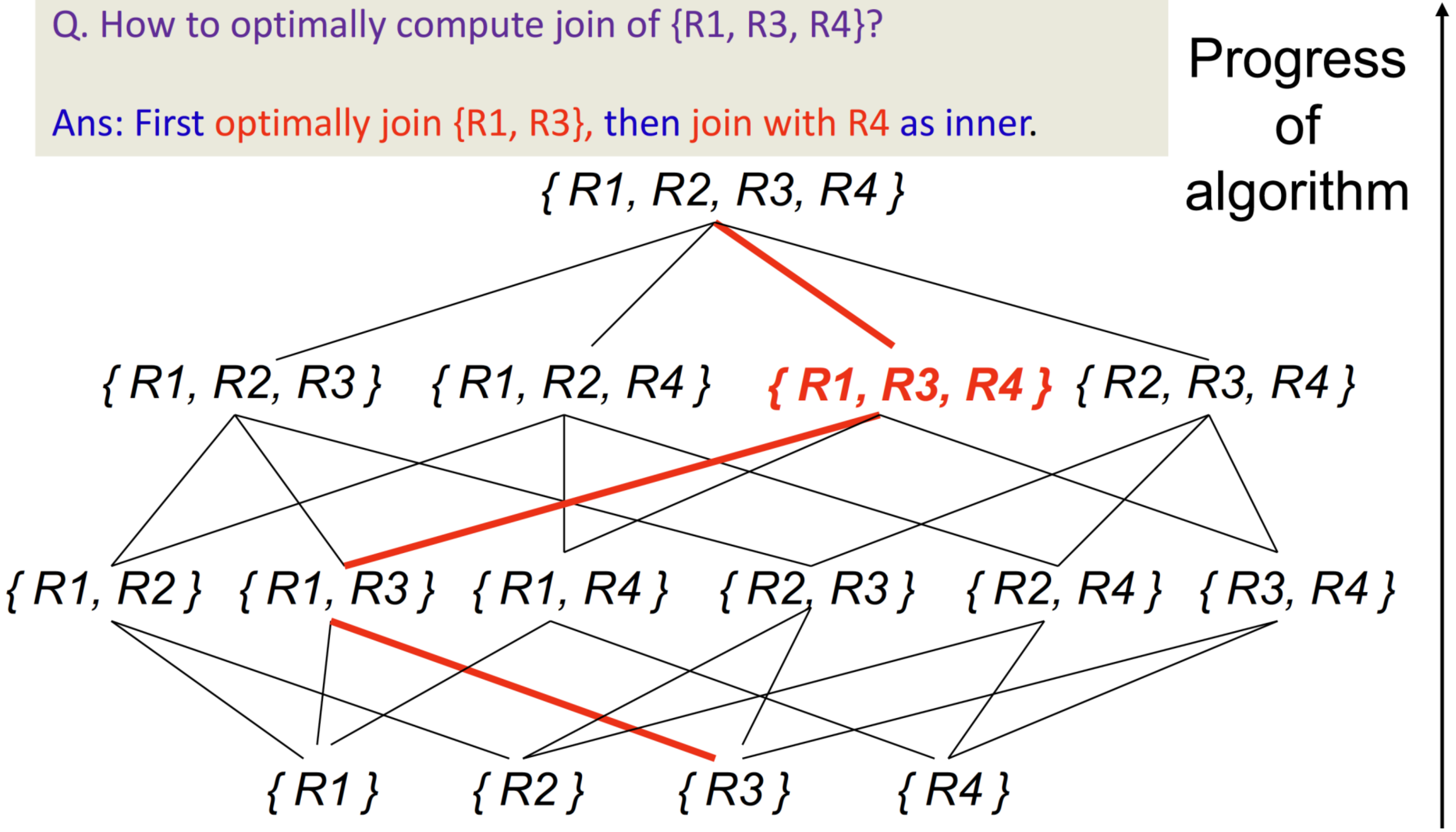


Selinger Algorithm

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.

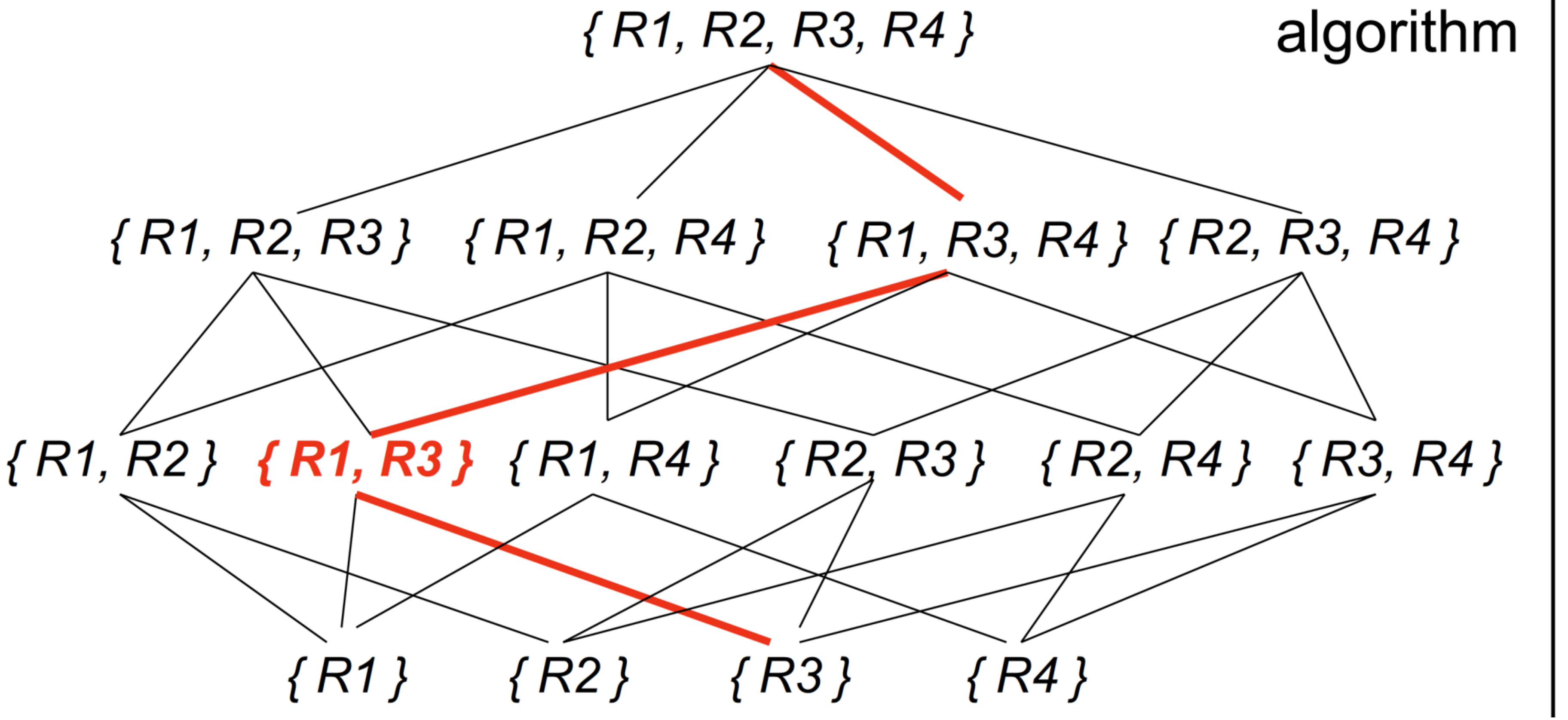


Selinger Algorithm

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.

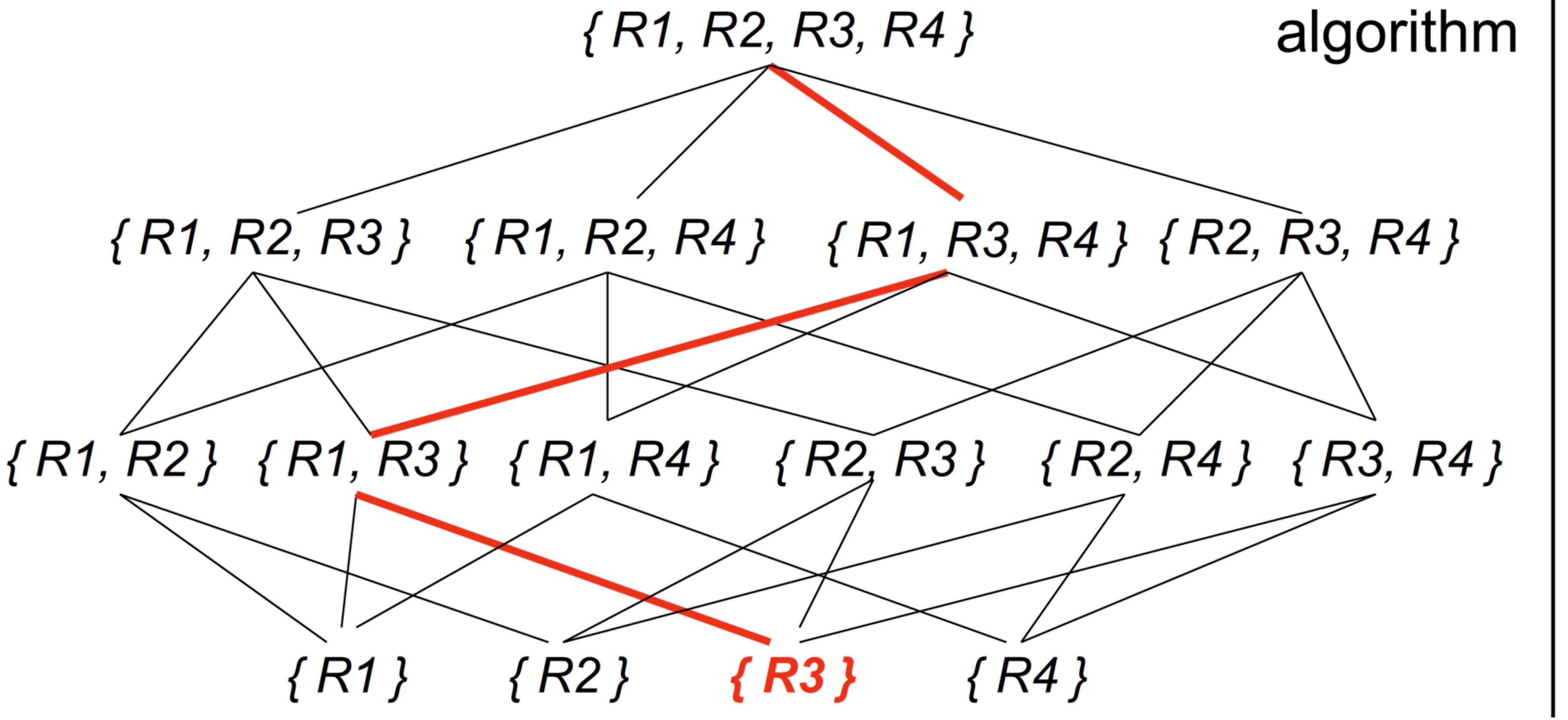


Selinger Algorithm

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

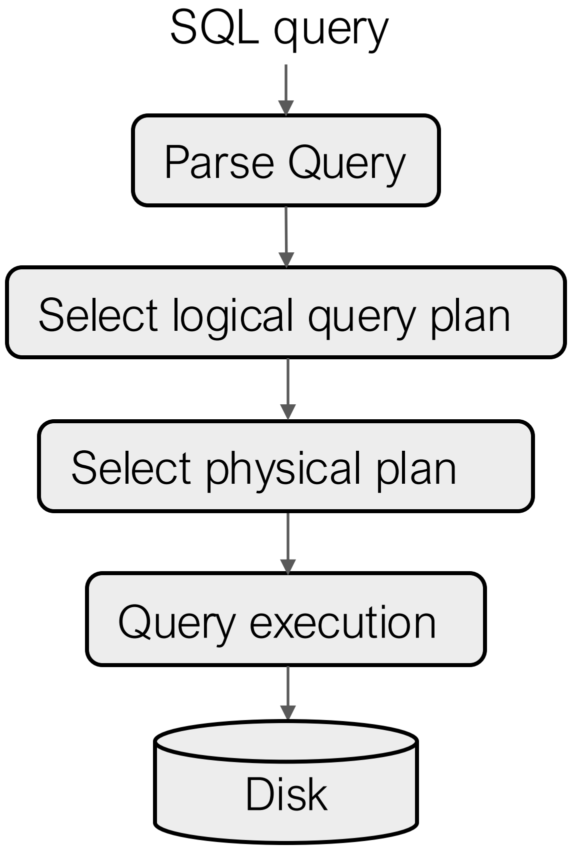
Q. How to optimally compute join of {R3}?

Ans: Single relation – so optimally scan R3.



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)