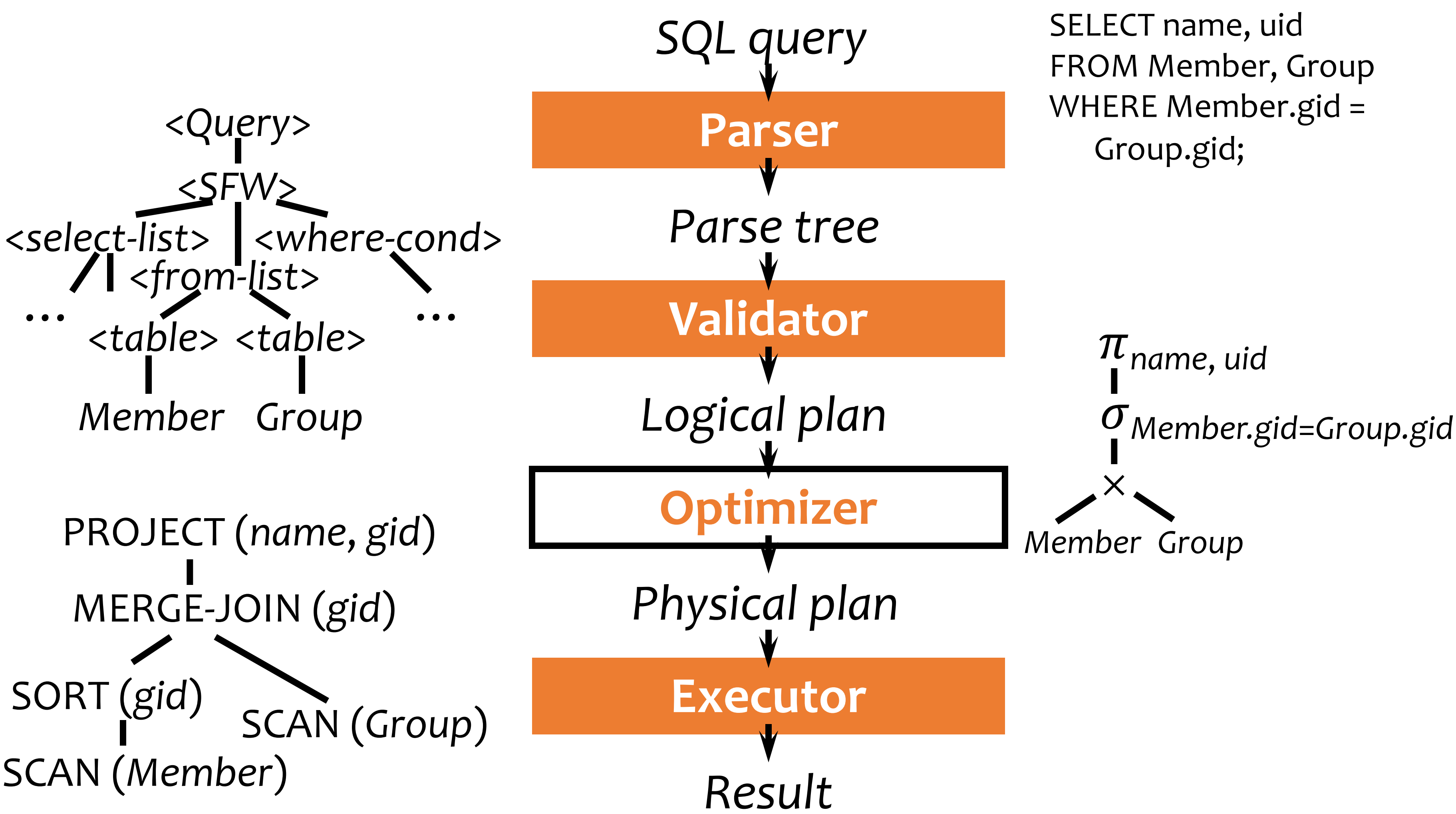
Lecture 17: Query Processing & Optimization

CS348 Spring 2025:
Introduction to Database Management

Instructor: **Xiao Hu**

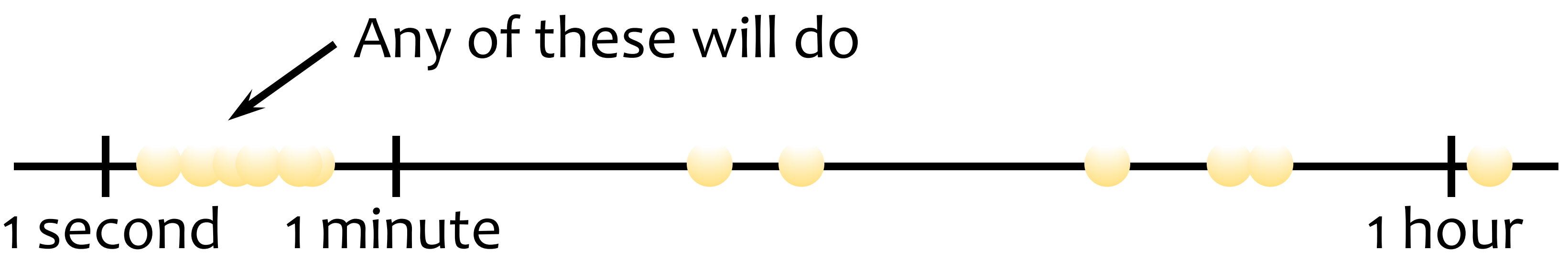
Sections: 001, 002, 003

A query's trip through the DBMS



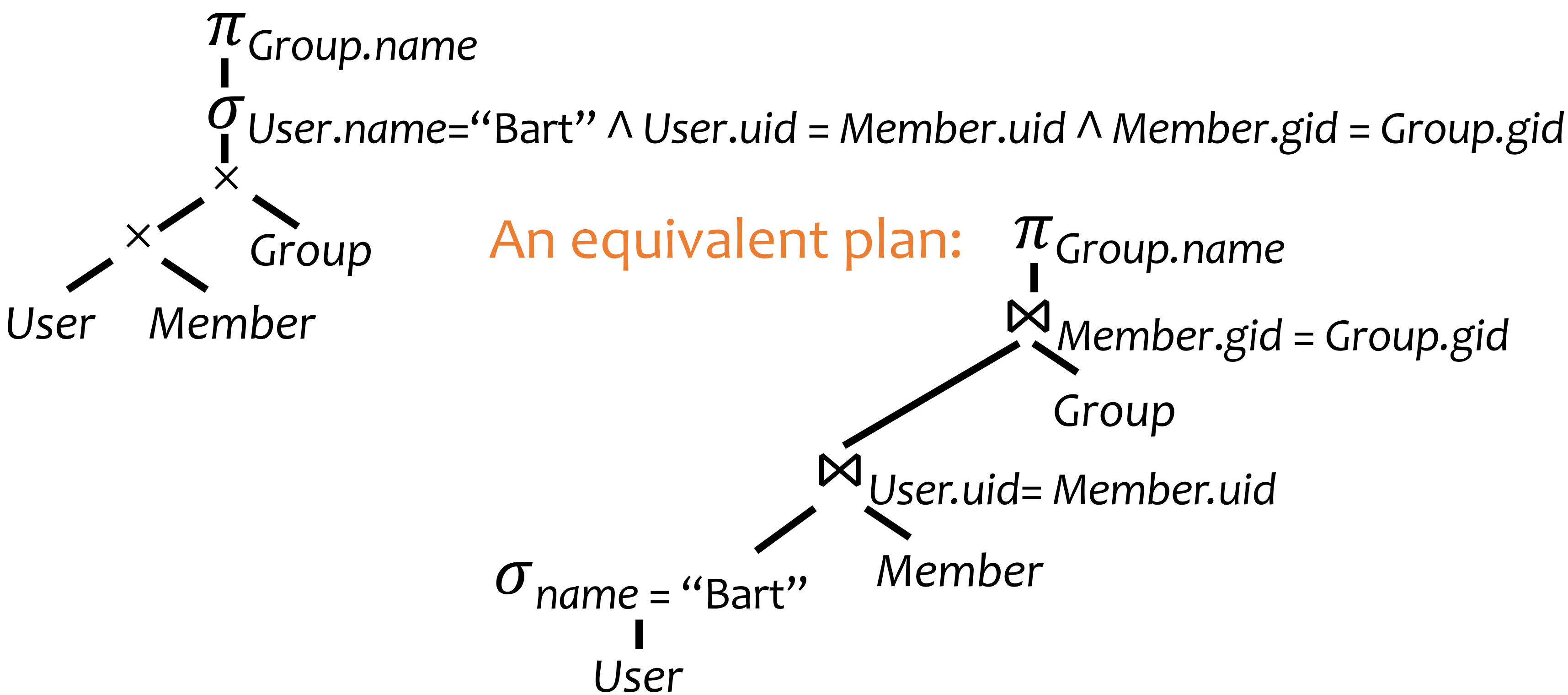
(Recap) Query optimization

- Why query optimization?
- Search space
  - What are the possible equivalent logical plans?
  - What are the possible physical plans? (Lecture 16)
- Search strategy
  - Rule-based strategy
  - Cost-based strategy



(Recap) Logical plan

- Nodes are logical operators (often relational algebra operators)
- Apply algebraic equivalences to systematically transform a plan to new ones



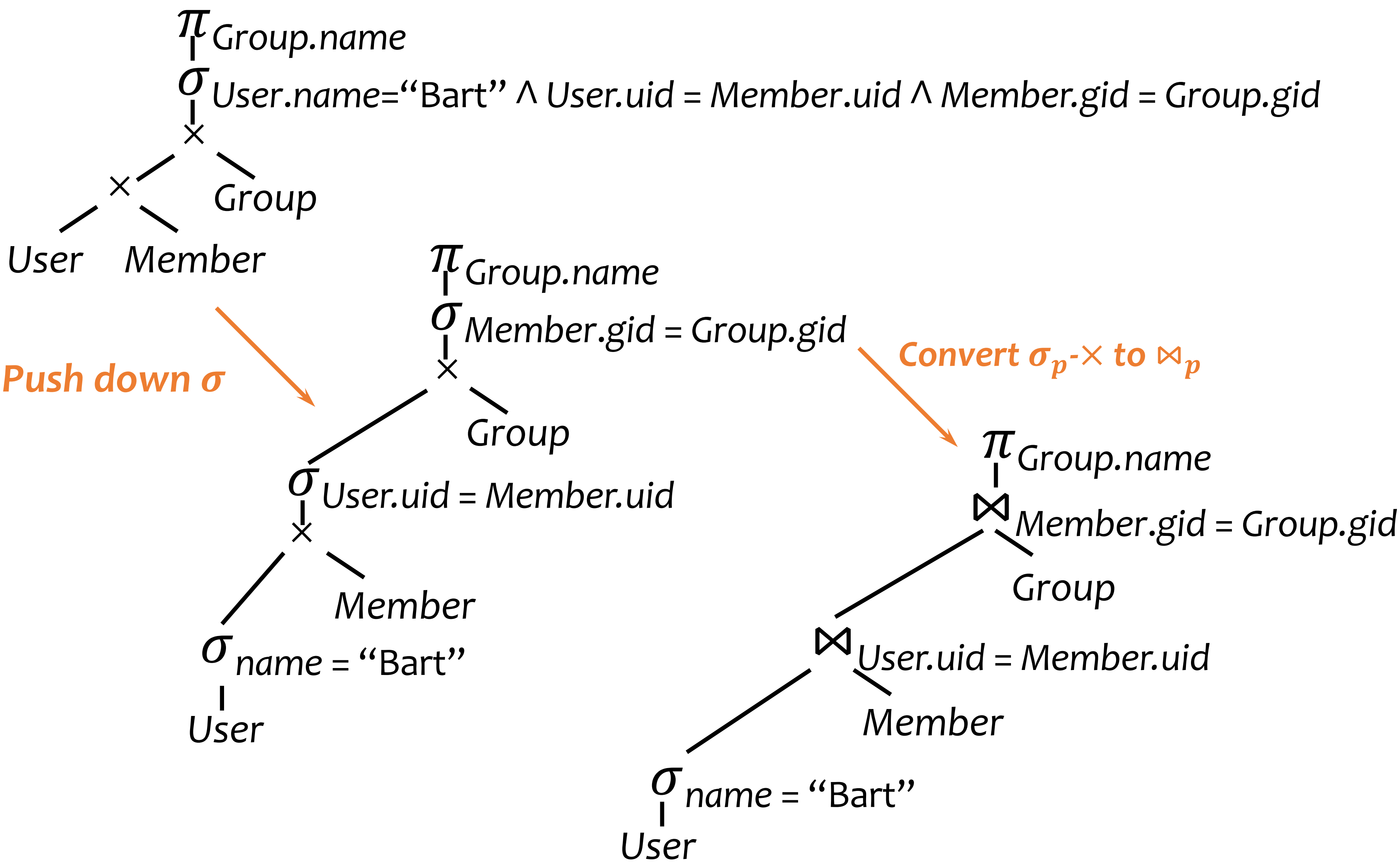
(Recap) Algebraic equivalences

- Join reordering: $\times$ and $\bowtie$ are associative and commutative (except column ordering)
- Convert σ_p-$\times$ to/from $\bowtie_p$: $\sigma_p(R \times S) = R \bowtie_p S$
- Merge/split σ's: $\sigma_{p_1}\left(\sigma_{p_2} R\right) = \sigma_{p_1 \wedge p_2} R$
- Merge/split π's: $\pi_{L_1}\left(\pi_{L_2} R\right) = \pi_{L_1 \cup L_2} R$
- Push down/pull up σ: (p_R involves only R; p_S involves only S; p and p' involve R and S)

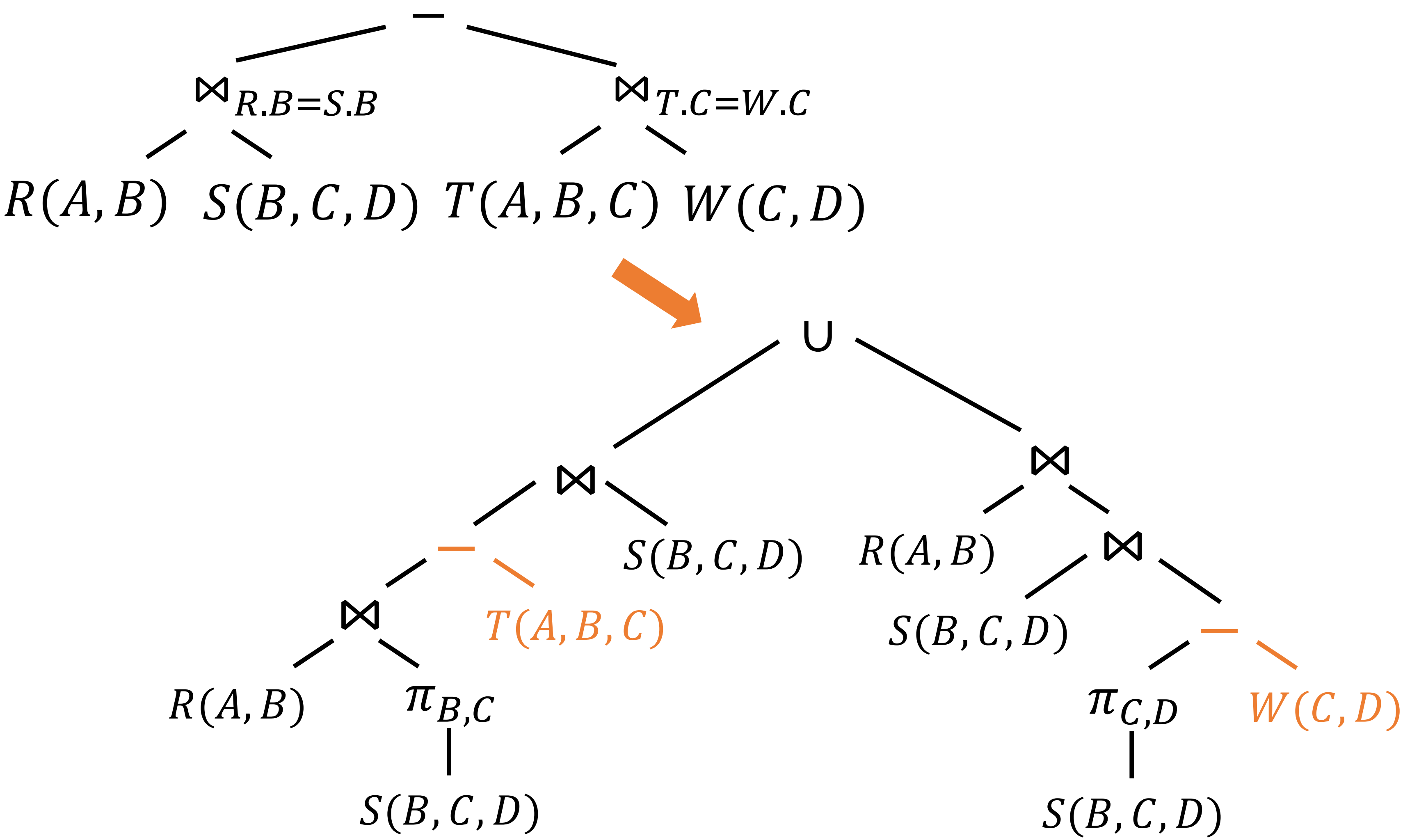
$$\sigma_{p \wedge p_R \wedge p_S}\left(R \bowtie_{p'} S\right) = \left(\sigma_{p_R} R\right) \bowtie_{p \wedge p'} \left(\sigma_{p_S} S\right)$$

- Push down π: $\pi_L\left(\sigma_p R\right) = \pi_L\left(\sigma_p(\pi_{L \cup L'} R)\right)$
  - L' is the set of columns referenced by p

More complicated examples



More complicated examples



Rule-based query optimization

- Do we need to examine all the logical plans?
  - No! We can apply rules to find a “cheaper” logical plan

- Start with a logical plan
- Push selections/projections down as much as possible
  - Why? Reduce the size of intermediate results
- Join smaller relations first and avoid cross product
  - Why? Joins are more optimized and have alternate implementations
- Many other rules to be further exploited…

From rule-based to cost-based opt.

- Rule-based optimization
  - Apply algebraic equivalence to rewrite plans into cheaper ones
- Cost-based optimization
  - Rewrite logical plan to combine "blocks" as much as possible
  - Optimize query block by block
    - Enumerate logical plans (already covered)
    - Estimate the cost of plans
    - Pick a plan with acceptable cost
  - Focus: select-project-join blocks

"Selinger"-style query optimization ← *Patricia Selinger*

Cost estimation

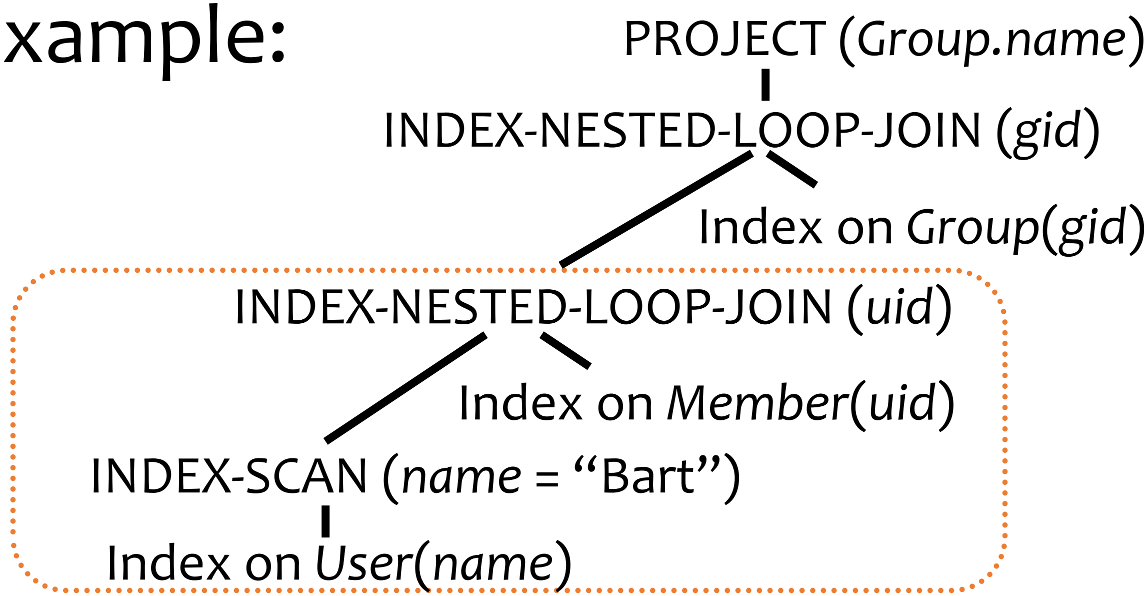
http://www.learningresources.com/product/estimation+station.do

Cost estimation

Physical plan example:

Input to Join(*uid*):

What is its input size? How many tuples with name='Bart'?



- We have: cost estimation for each operator
  - Example: INDEX-NESTED-LOOP-JOIN(*uid*) takes $O(B(R) + |R| \cdot (\text{index lookup} + \text{record fetch}))$
- We need: size of intermediate results

Lecture 16

Selections with equality predicates

- Q: $\sigma_{A=v}R$
- DBMSs typically store the following in the catalog
  - Size of R: $|R|$
  - Number of distinct A values in R: $|\pi_A R|$
- Assumptions
  - Values of A are uniformly distributed in R

- $|Q| \approx {}^{|R|}/_{|\pi_A R|}$
  - Selectivity factor of $(A = v)$ is ${}^{1}/_{|\pi_A R|}$
  - Selectivity: the probability that any row will satisfy a predicate

Conjunctive predicates

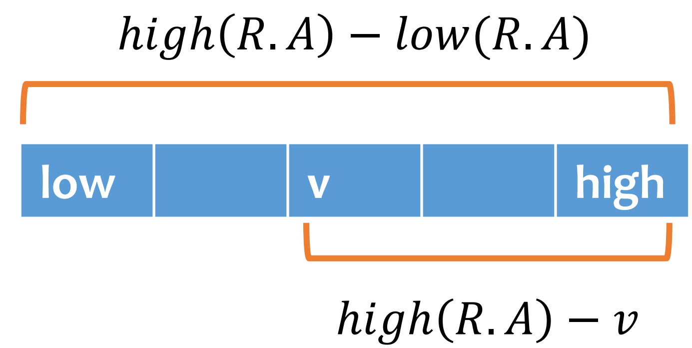
- Q: $\sigma_{A=u \wedge B=v} R$
- Additional assumptions
    - $(A = u)$ and $(B = v)$ are independent
        - Counterexample: major and advisor
    - No "over"-selection
        - Counterexample: A is the key
- $|Q| \approx {}^{|R|}/_{|\delta_A R| \cdot |\delta_B R|}$
    - Reduce total size by all selectivity factors

Negated and disjunctive predicates

- Q: $\sigma_{A \neq v} R$
  - $|Q| \approx |R| \cdot \left(1 - {}^{1}/_{|\delta_A R|}\right)$
    - Selectivity factor of $\neg p$ is $(1 -$ selectivity factor of $p)$
- Q: $\sigma_{A=u \vee B=v} R$
  - $|Q| \approx |R| \cdot \left({}^{1}/_{|\delta_A R|} + {}^{1}/_{|\delta_B R|}\right)$?
    - No! Tuples satisfying $(A = u)$ and $(B = v)$ are counted twice
  - $|Q| \approx |R| \cdot \left({}^{1}/_{|\delta_A R|} + {}^{1}/_{|\delta_B R|} - {}^{1}/_{|\delta_A R||\delta_B R|}\right)$
    - Inclusion-exclusion principle

Range predicates

- Q: $\sigma_{A>v}R$
- Not enough information!
  - Just pick, say, $|Q| \approx |R| \cdot {}^{1}/_{3}$



- With more information
  - Largest R.A value: $\mathrm{high}(R.A)$
  - Smallest R.A value: $\mathrm{low}(R.A)$
  - $|Q| \approx |R| \cdot \frac{\mathrm{high}(R.A) - v}{\mathrm{high}(R.A) - \mathrm{low}(R.A)}$
  - Selectivity factor: $\frac{\mathrm{high}(R.A) - v}{\mathrm{high}(R.A) - \mathrm{low}(R.A)}$

Two-way natural join

- $Q = R(A, B) \bowtie S(A, C)$
- Assumption: containment of value sets
  - Every tuple in the "smaller" relation (one with fewer distinct values for the join attribute) joins with some tuple in the other relation
    - That is, if $|\pi_A R| \leq |\pi_A S|$ then $\pi_A R \subseteq \pi_A S$
  - Certainly not true in general
  - But holds in the common case of foreign key joins
- $|Q| \approx \frac{|R| \cdot |S|}{\max(|\pi_A R|, |\pi_A S|)}$
  - Selectivity factor of $R.A = S.A$ is ${}^{1}/_{\max(|\pi_A R|, |\pi_A S|)}$

Cost estimation: summary

- Using similar ideas, we can estimate the size of projection, duplicate elimination, union, difference, aggregation (with grouping)

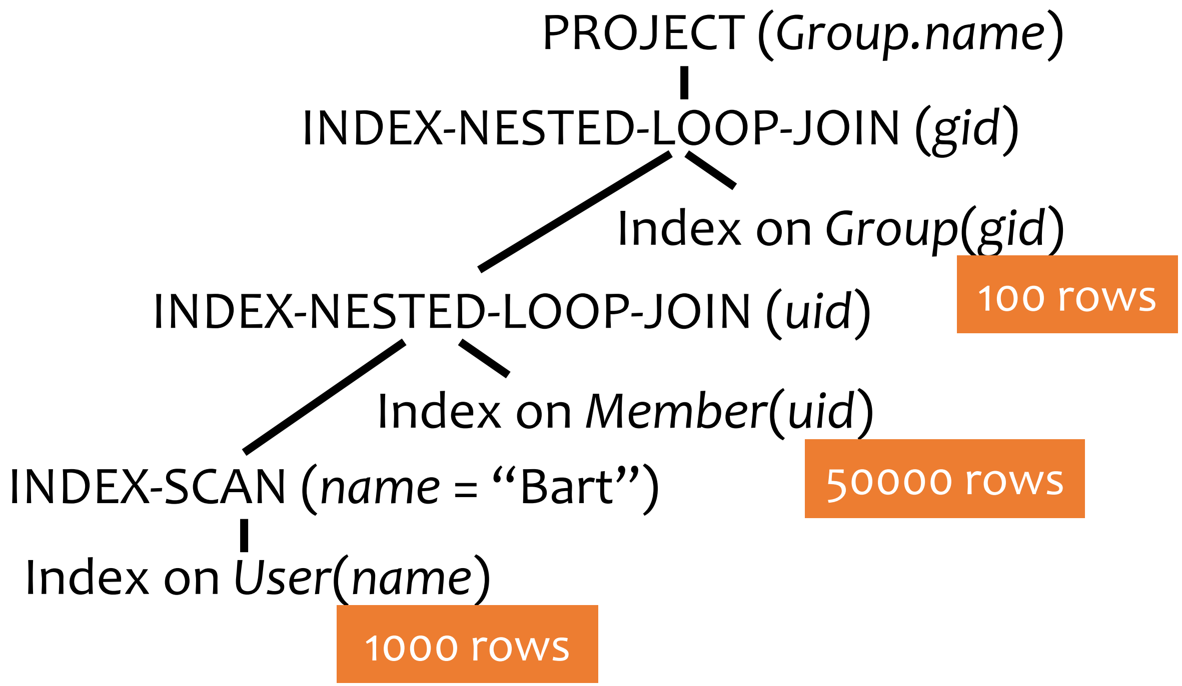
- Lots of assumptions and very rough estimation
  - Accurate estimate is not needed
  - Maybe okay if we overestimate or underestimate consistently
  - May lead to very nasty optimizer "hints"

```
SELECT * FROM User WHERE pop > 0.9;
SELECT * FROM User WHERE pop > 0.9 AND pop > 0.9;
```

Case Study

Data statistics:

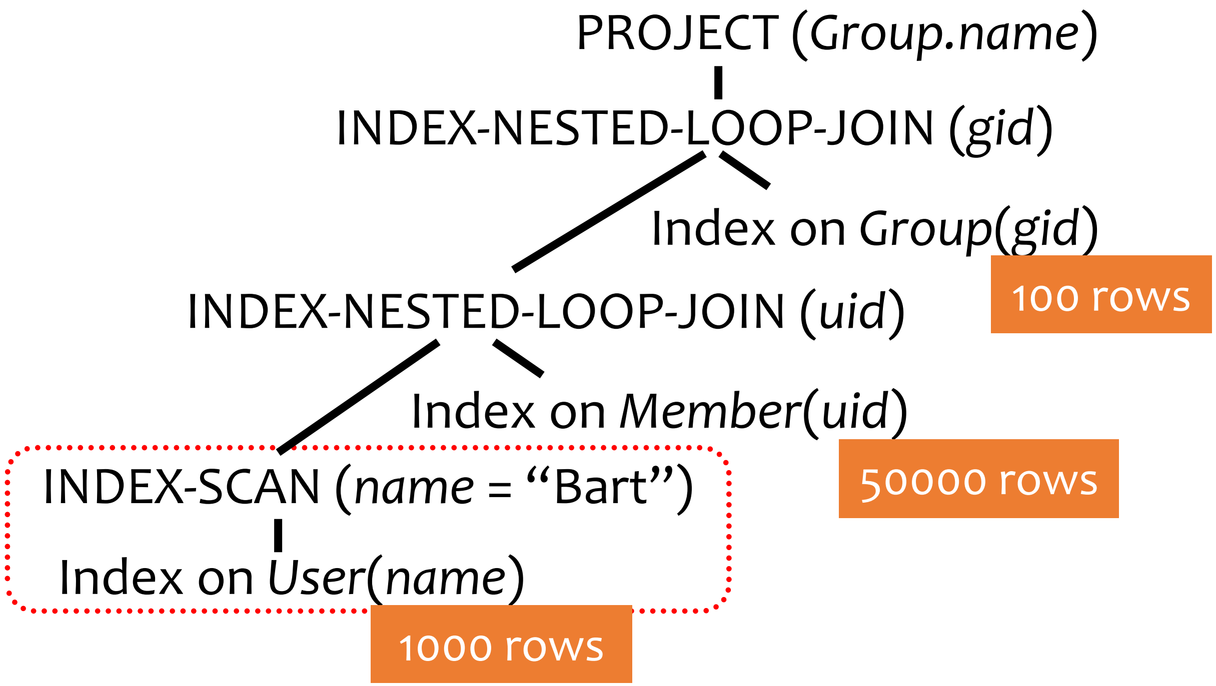
- $|User| = 1000$
- $|Member| = 50000$
- $|Group| = 100$
- $|\pi_{name} User| = 50$
- $|\pi_{uid} Member| = 500$
- $|\pi_{gid} Member| = 100$
- $|\pi_{gid} Group| = 100$



Case Study

Data statistics:

- $|User| = 1000$
- $|Member| = 50000$
- $|Group| = 100$
- $|\pi_{name} User| = 50$
- $|\pi_{uid} Member| = 500$
- $|\pi_{gid} Member| = 100$
- $|\pi_{gid} Group| = 100$

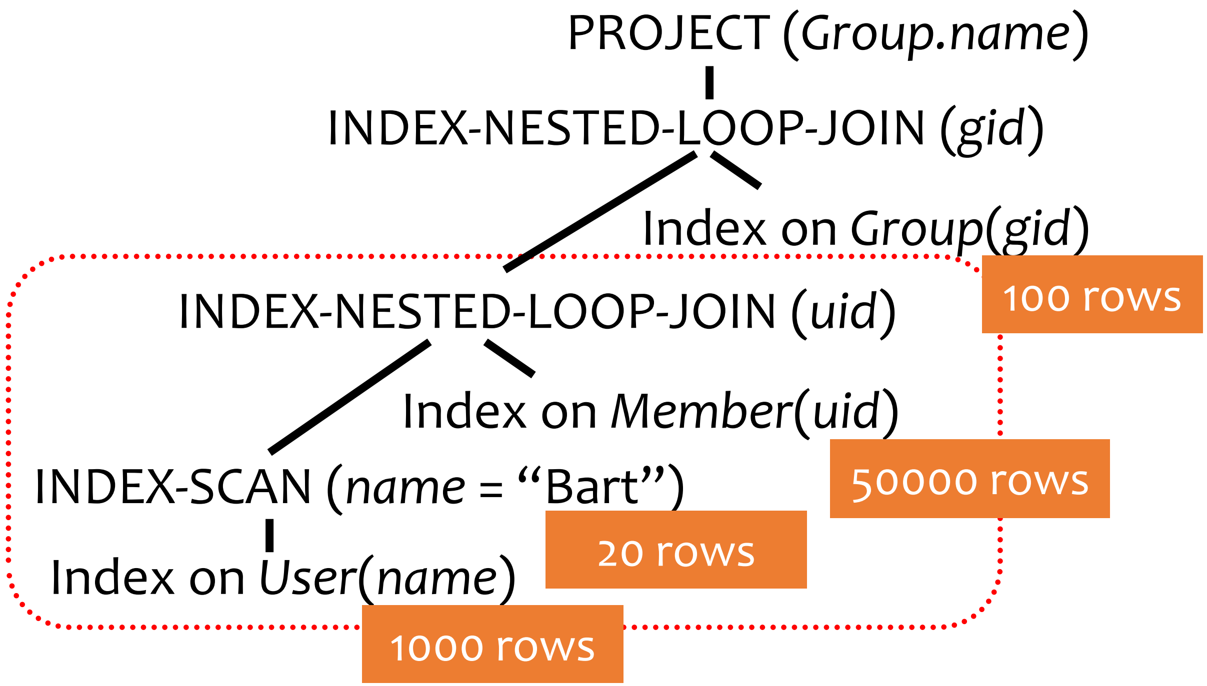


- Assume that the values of $name$ are uniformly distributed in $User$
- What is $|\sigma_{name="Bart"} User| = ?$

Case Study

Data statistics:

- $|User| = 1000$
- $|Member| = 50000$
- $|Group| = 100$
- $|\pi_{name} User| = 50$
- $|\pi_{uid} Member| = 500$
- $|\pi_{gid} Member| = 100$
- $|\pi_{gid} Group| = 100$



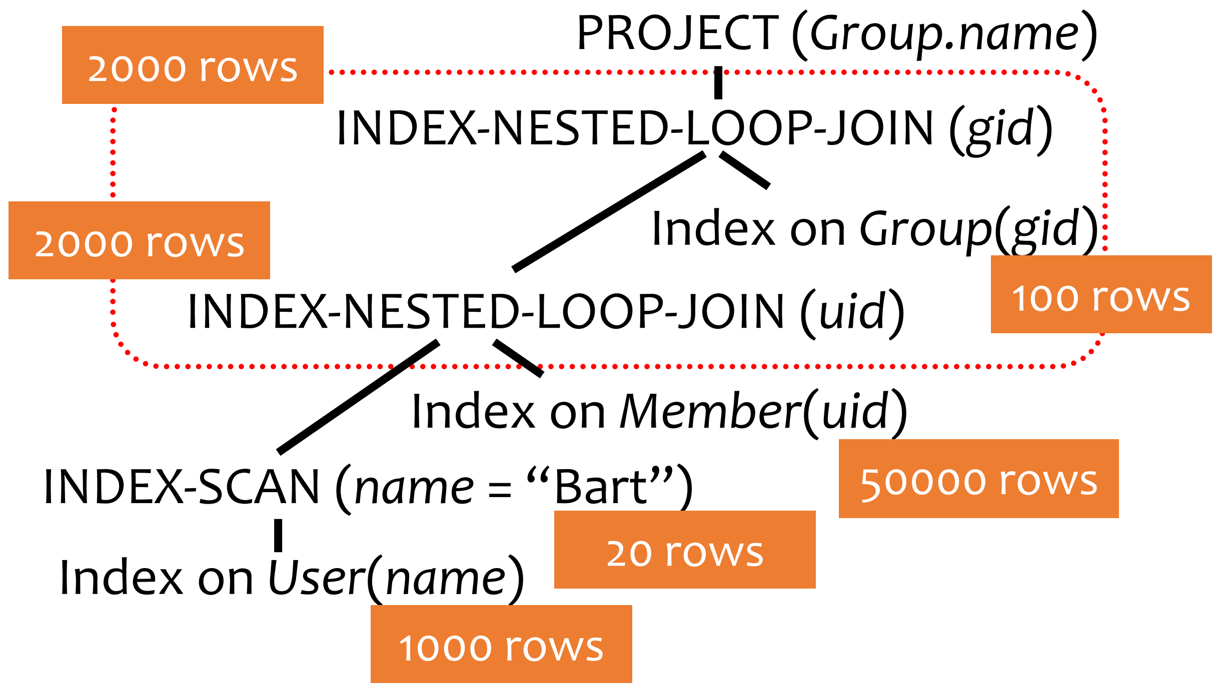
- What is the intermediate join size?

$$\approx \frac{|R| \cdot |S|}{\max(|\pi_A R|, |\pi_A S|)} = \frac{20 \cdot 50000}{\max(20, 500)} = 2000$$

Case Study

Data statistics:

- $|User| = 1000$
- $|Member| = 50000$
- $|Group| = 100$
- $|\pi_{name}User| = 50$
- $|\pi_{uid}Member| = 500$
- $|\pi_{gid}Member| = 100$
- $|\pi_{gid}Group| = 100$



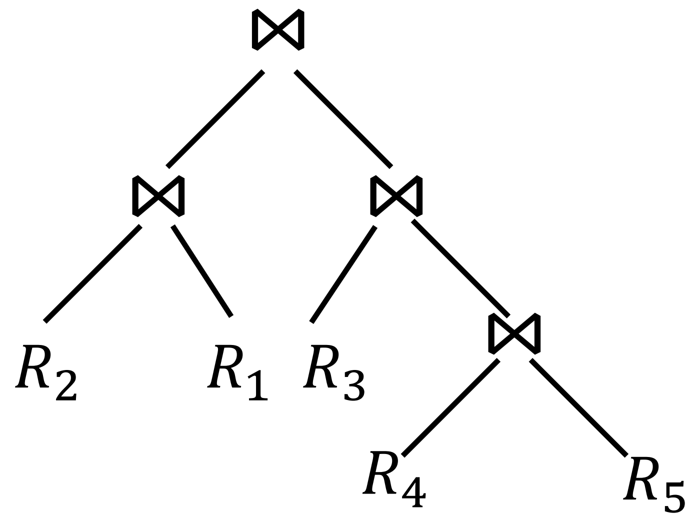
- What is the intermediate join size?

$$\approx \frac{|R| \cdot |S|}{\max(|\pi_A R|, |\pi_A S|)} = \frac{2000 \cdot 100}{\max(100, 100)} = 2000$$

Cost-based optimization for Multi-way equi-joins

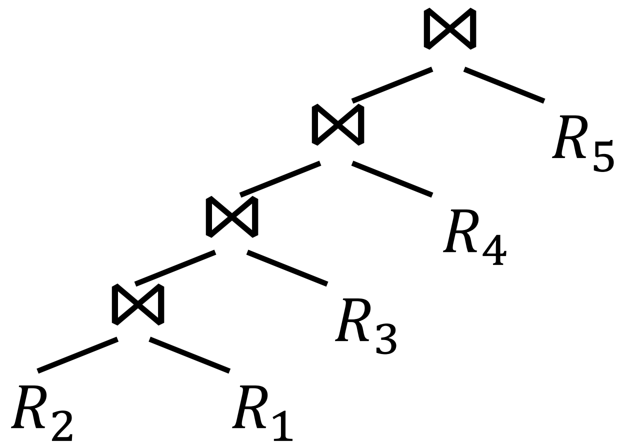
Search space is huge

- Huge!
- "Bushy" plan example:



- Just considering different join orders, there are $\frac{(2n-2)!}{(n-1)!}$ bushy plans for $R_1 \bowtie \cdots \bowtie R_n$
  - 30240 for $n = 6$
- And there are more if we consider:
  - Multi-way joins
  - Different join methods
  - Placement of selection and projection operators

Left-deep plans



- Heuristic: consider only "left-deep" plans, in which only the left child can be a join
  - Tend to be better than plans of other shapes, because many join algorithms scan inner (right) relation multiple times—you will not want it to be a complex subtree
- How many left-deep plans are there for $R_1 \bowtie \cdots \bowtie R_n$?
  - Significantly fewer, but still lots— $n!$ (720 for $n = 6$)

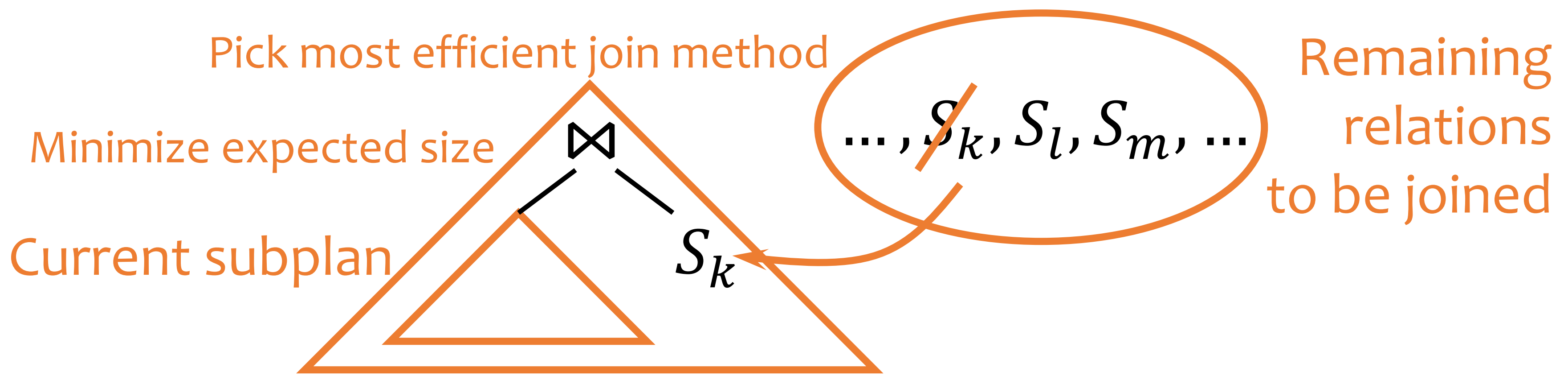
It is a matter of intermediate join size

- Suppose the natural join operator is computed by the sort-merge join algorithm
  - So, the cost is proportional to the input size and join size

- How to minimize the intermediate join size?

A greedy algorithm

- $S_1, \ldots, S_n$
  - Say selections have been pushed down; i.e., $S_i = \sigma_p(R_i)$
- Start with the pair S_i, S_j with the smallest estimated size for $S_i \bowtie S_j$
- Repeat until no relation is left:
  Pick S_k from the remaining relations such that the join of S_k and the current result yields an intermediate result of the smallest size



A dynamic programming approach

- Generate optimal plans bottom-up
  - Pass 1: Find the best single-table plans (for each table)
  - Pass 2: Find the best two-table plans (for each pair of tables) by combining best single-table plans
  - …
  - Pass k: Find the best k-table plans (for each combination of k tables) by combining two smaller best plans found in previous passes
  - …
- Rationale: Any subplan of an optimal plan must also be optimal (otherwise, just replace the subplan to get a better overall plan)

  ☞Well, not quite — we will see later

Example of a DP algorithm

- $Q: R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, A)$
  - $|R| = |S| = |T| = |U| = 1000$
  - $|\delta_A R| = 100, |\delta_B R| = 200$
  - $|\delta_B S| = 100, |\delta_C S| = 500$
  - $|\delta_C T| = 20, |\delta_D T| = 50$
  - $|\delta_D U| = 50, |\delta_A U| = 1000$

$Q: R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(D,A)$

- Pass 1:

$\{R\}$	$\{S\}$	$\{T\}$	$\{U\}$
1K	1K	1K	1K

- Pass 2:

$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
5K	1M	10K	2K	1M	1K
$(R \bowtie S)$	$(R \bowtie T)$	$(R \bowtie U)$	$(S \bowtie T)$	$(S \bowtie U)$	$(T \bowtie U)$

For example: $|R \bowtie S| = \frac{|R|\cdot|S|}{\max\{|\delta_B R|,|\delta_B S|\}} = \frac{1000\cdot 1000}{200}$

- Pass 3:

$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
10K	50K	10K	2K
$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(S \bowtie T) \bowtie U$

For example: best for $\{R,S,T\}$ =
best among: {
(best for $\{R,S\}$) $\bowtie T$,
(best for $\{R,T\}$) $\bowtie S$,
(best for $\{S,T\}$) $\bowtie R$}

- Pass 4:

$\{R,S,T,U\}$
??
??

The best among: {
$\{R,S,T\} \bowtie U,\ \{R,S,U\} \bowtie T,\ \{R,T,U\} \bowtie S,\ \{S,T,U\} \bowtie R,$
$\{R,S\} \bowtie \{T,U\},\ \{R,T\} \bowtie \{S,U\},\ \{R,U\} \bowtie \{S,T\}$
}

The need for "interesting order"

Example: $R(A, B) \bowtie S(A, C) \bowtie T(A, D)$

- Let's say the best plan for $R \bowtie S$ is hash join (beats sort-merge join)
- But the best overall plan may be sort-merge join R and S, and then sort-merge join with T
  - Subplan of the optimal plan is not optimal!
- How could this happen?
  - The result of the sort-merge join of R and S is sorted on A
  - This is an interesting order that can be exploited by later processing (e.g., join, dup elimination, GROUP BY, ORDER BY, etc.)!

Dealing with interesting orders

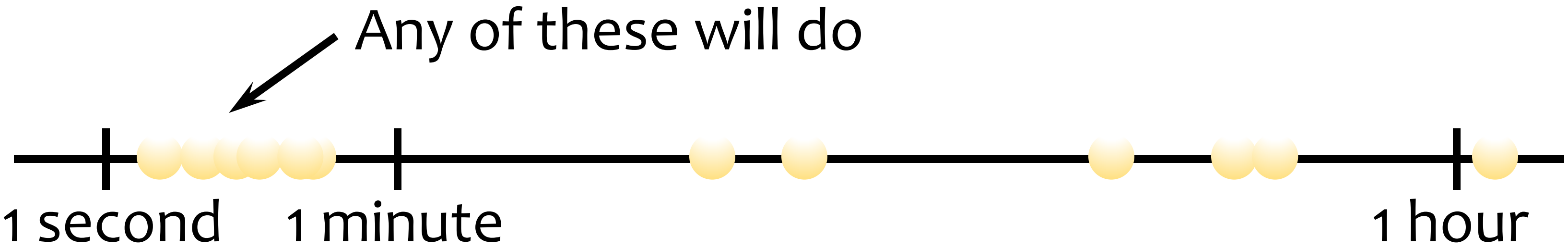
When picking the best plan

- Comparing their costs is not enough
  - Plans are not totally ordered by cost anymore
- Comparing interesting orders is also needed
  - Plans are now partially ordered
  - Plan X is better than plan Y if
    - Cost of X is lower than Y, and
    - Interesting orders produced by X "subsume" those produced by Y
      - E.g., Y sorts output by column A, while X sorts by A or (A, B)
- Need to keep a set of optimal plans for joining every combination of k tables
  - At most one for each interesting order

Summary

- Search space
  - What are the possible equivalent logical plans?
  - What are the possible physical plans? (Lecture 16)

- Search strategy
  - Rule-based strategy
  - Cost-based strategy



A query’s trip through the DBMS