# Query Optimization

Introduction to Databases

CompSci 316 Spring 2019

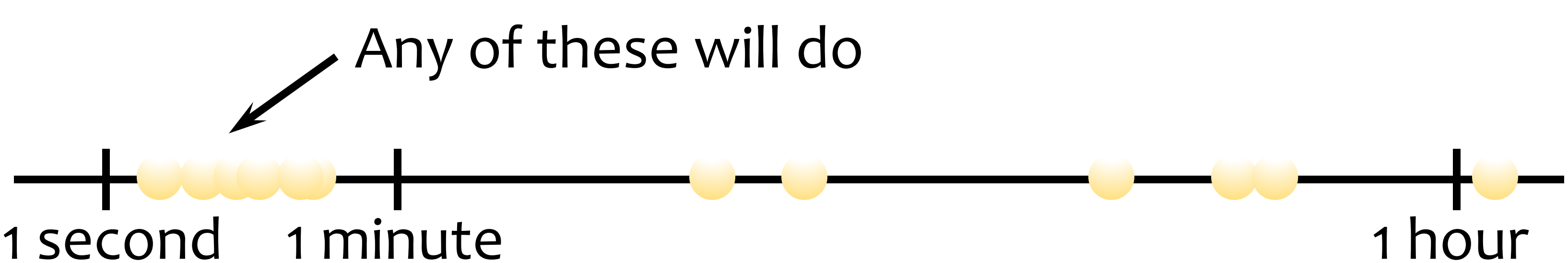
DUKE
COMPUTER SCIENCE

# Announcements (Thu., Apr. 9)

- Friday 04/12: HW4-problem 1 due (gradiance)
- Monday 04/15: Hw4-problem 3 due (gradescope)

# Query optimization

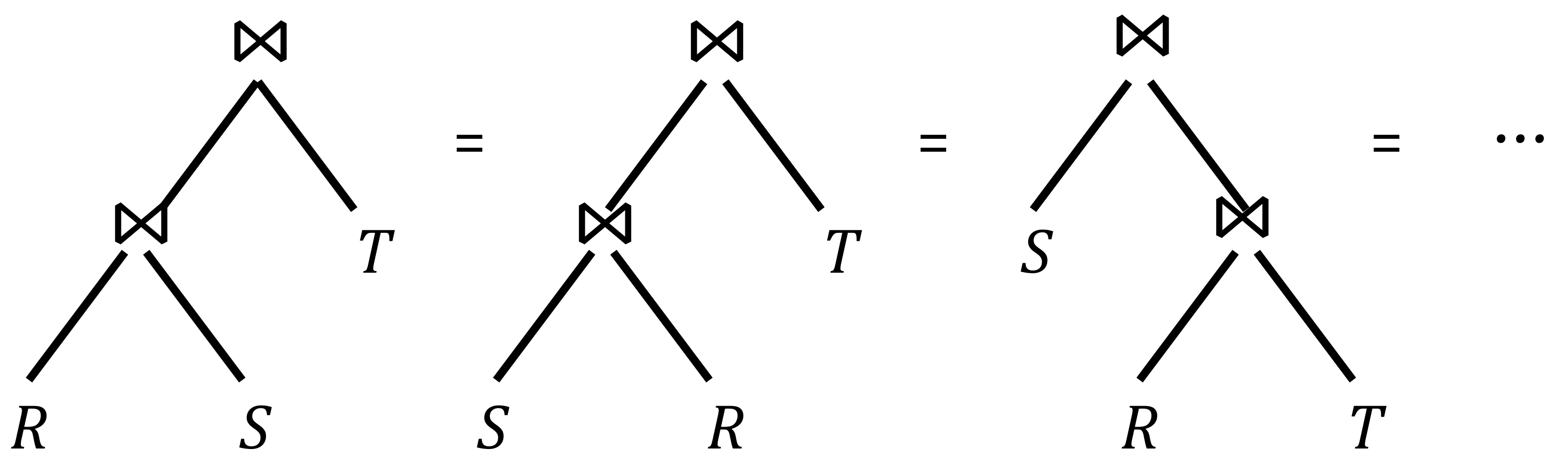
- One logical plan → "best" physical plan
- Questions
  - How to enumerate possible plans
  - How to estimate costs
  - How to pick the "best" one
- Often the goal is not getting the optimum plan, but instead avoiding the horrible ones



# Plan enumeration in relational algebra

- Apply relational algebra equivalences

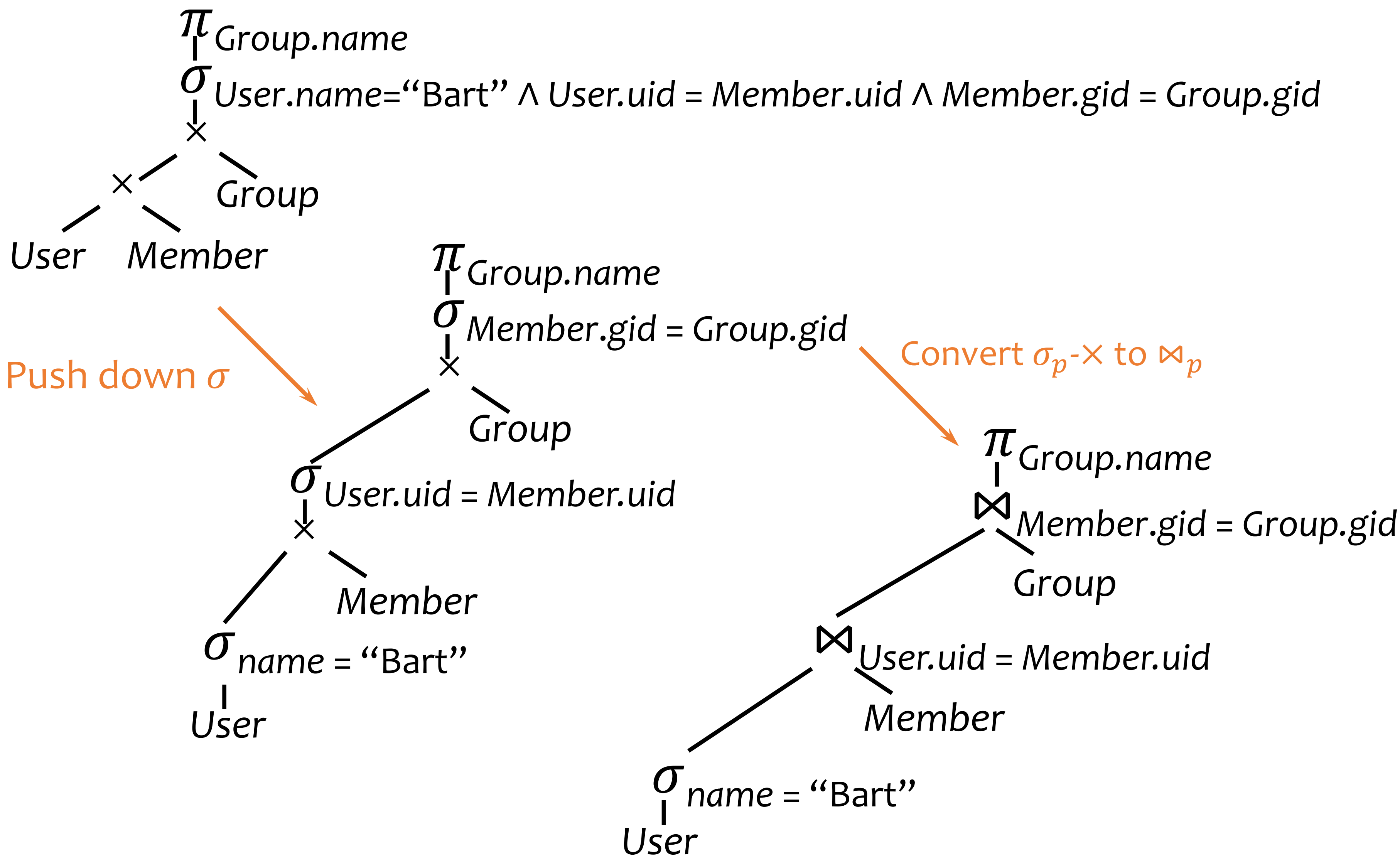
☞ Join reordering: × and ⋈ are associative and commutative (except column ordering, but that is unimportant)



# More relational algebra equivalences

- Convert $\sigma_p$-$\times$ to/from $\bowtie_p$: $\sigma_p(R \times S) = R \bowtie_p S$
- Merge/split $\sigma$'s: $\sigma_{p_1}\left(\sigma_{p_2} R\right) = \sigma_{p_1 \wedge p_2} R$
- Merge/split $\pi$'s: $\pi_{L_1}\left(\pi_{L_2} R\right) = \pi_{L_1} R$, where $L_1 \subseteq L_2$
- Push down/pull up $\sigma$:
  $\sigma_{p \wedge p_r \wedge p_s}\left(R \bowtie_{p'} S\right) = \left(\sigma_{p_r} R\right) \bowtie_{p \wedge p'} \left(\sigma_{p_s} S\right)$, where
  - $p_r$ is a predicate involving only $R$ columns
  - $p_s$ is a predicate involving only $S$ columns
  - $p$ and $p'$ are predicates involving both $R$ and $S$ columns
- Push down $\pi$: $\pi_L\left(\sigma_p R\right) = \pi_L\left(\sigma_p\left(\pi_{LL'} R\right)\right)$, where
  - $L'$ is the set of columns referenced by $p$ that are not in $L$
- Many more (seemingly trivial) equivalences…
  - Can be systematically used to transform a plan to new ones

# Relational query rewrite example



# Heuristics-based query optimization

- Start with a logical plan
- Push selections/projections down as much as possible
  - Why? Reduce the size of intermediate results
  - Why not? May be expensive; maybe joins filter better
- Join smaller relations first, and avoid cross product
  - Why? Reduce the size of intermediate results
  - Why not? Size depends on join selectivity too
- Convert the transformed logical plan to a physical plan (by choosing appropriate physical operators)

# SQL query rewrite

- More complicated—subqueries and views divide a query into nested "blocks"
  - Processing each block separately forces particular join methods and join order
  - Even if the plan is optimal for each block, it may not be optimal for the entire query
- Unnest query: convert subqueries/views to joins

☞We can just deal with select-project-join queries

  - Where the clean rules of relational algebra apply

# SQL query rewrite example

- SELECT name
  FROM User
  WHERE uid = ANY (SELECT uid FROM Member);
- SELECT name
  FROM User, Member
  WHERE User.uid = Member.uid;
  - Wrong—consider two Bart's, each joining two groups
- SELECT name
  FROM (SELECT DISTINCT User.uid, name
  FROM User, Member
  WHERE User.uid = Member.uid);
  - Right—assuming User.uid is a key

# Dealing with correlated subqueries

- SELECT gid FROM Group
  WHERE name LIKE 'Springfield%'
  AND min_size > (SELECT COUNT(*) FROM Member
  WHERE Member.gid = Group.gid);
- SELECT gid
  FROM Group, (SELECT gid, COUNT(*) AS cnt
  FROM Member GROUP BY gid) t
  WHERE t.gid = Group.gid AND min_size > t.cnt
  AND name LIKE 'Springfield%';
  - New subquery is inefficient (it computes the size for *every* group)
  - Suppose a group is empty?

# “Magic” decorrelation

- ```
  SELECT gid FROM Group
  WHERE name LIKE 'Springfield%'
  AND min_size > (SELECT COUNT(*) FROM Member
                  WHERE Member.gid = Group.gid);
  ```
- ```
  WITH Supp_Group AS
  (SELECT * FROM Group WHERE name LIKE 'Springfield%'),
  ```
  Process the outer query without the subquery

  ```
  Magic AS
  (SELECT DISTINCT gid FROM Supp_Group),
  ```
  Collect bindings

  ```
  DS AS
  ((SELECT Group.gid, COUNT(*) AS cnt
    FROM Magic, Member WHERE Magic.gid = Member.gid
    GROUP BY Member.gid) UNION
   (SELECT gid, 0 AS cnt
    FROM Magic WHERE gid NOT IN (SELECT gid FROM Member)))
  ```
  Evaluate the subquery with bindings

  ```
  SELECT Supp_Group.gid FROM Supp_Group, DS
  WHERE Supp_Group.gid = DS.gid
  AND min_size > DS.cnt;
  ```
  Finally, refine the outer query

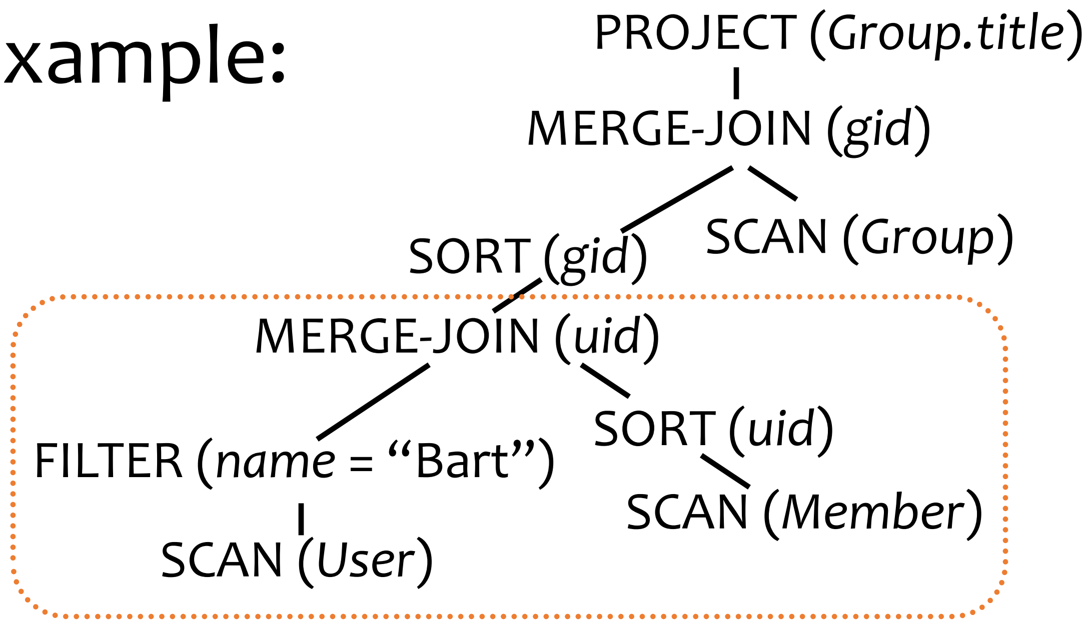
# Heuristics- vs. cost-based optimization

- Heuristics-based optimization
  - Apply heuristics 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

# Cost estimation

Physical plan example:

Input to SORT(*gid*):




- We have: cost estimation for each operator
  - Example: SORT(*gid*) takes $O(B(\text{input}) \times \log_M B(\text{input}))$
    - But what is $B(\text{input})$?
- We need: size of intermediate results

# Cardinality estimation


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

# Selections with equality predicates

- $Q$: $\sigma_{A=v}R$
- Suppose the following information is available
  - Size of $R$: $|R|$
  - Number of distinct $A$ values in $R$: $|\pi_A R|$
- Assumptions
  - Values of $A$ are uniformly distributed in $R$
  - Values of $v$ in $Q$ are uniformly distributed over all $R.A$ values
- $|Q| \approx {}^{|R|}/_{|\pi_A R|}$
  - Selectivity factor of $(A = v)$ is ${}^{1}/_{|\pi_A R|}$

# 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
- $|Q| \approx {}^{|R|}/_{|\pi_A R| \cdot |\pi_B R|}$
  - Reduce total size by all selectivity factors

# Negated and disjunctive predicates

- $Q$: $\sigma_{A \neq v} R$
  - $|Q| \approx |R| \cdot \left(1 - {}^{1}\!/_{|\pi_A R|}\right)$
    - Selectivity factor of $\neg p$ is $(1 -$ selectivity factor of $p)$
- $Q$: $\sigma_{A=u \,\vee\, B=v} R$
  - $|Q| \approx |R| \cdot \left({}^{1}\!/_{|\pi_A R|} + {}^{1}\!/_{|\pi_B R|}\right)$?
    - No! Tuples satisfying $(A = u)$ and $(B = v)$ are counted twice
  - $|Q| \approx |R| \cdot \left({}^{1}\!/_{|\pi_A R|} + {}^{1}\!/_{|\pi_B R|} - {}^{1}\!/_{|\pi_A R||\pi_B R|}\right)$
    - 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: $\mathrm{low}(R.A)$
  - $|Q| \approx |R| \cdot \frac{\mathrm{high}(R.A) - v}{\mathrm{high}(R.A) - \mathrm{low}(R.A)}$
  - In practice: sometimes the second highest and lowest are used instead
    - The highest and the lowest are often used by inexperienced database designer to represent invalid values!

# Two-way equi-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 ${}^{1}/_{\max(|\pi_A R|, |\pi_A S|)}$

# Multiway equi-join

- $Q$: $R(A, B) \bowtie S(B, C) \bowtie T(C, D)$
- What is the number of distinct $C$ values in the join of $R$ and $S$?
- Assumption: preservation of value sets
    - A non-join attribute does not lose values from its set of possible values
    - That is, if $A$ is in $R$ but not $S$, then $\pi_A(R \bowtie S) = \pi_A R$
    - Certainly not true in general
    - But holds in the common case of foreign key joins (for value sets from the referencing table)

# Multiway equi-join (cont'd)

- $Q$: $R(A,B) \bowtie S(B,C) \bowtie T(C,D)$
- Start with the product of relation sizes
  - $|R| \cdot |S| \cdot |T|$
- Reduce the total size by the selectivity factor of each join predicate
  - $R.B = S.B$: ${}^{1}\!/_{\max(|\pi_B R|, |\pi_B S|)}$
  - $S.C = T.C$: ${}^{1}\!/_{\max(|\pi_C S|, |\pi_C T|)}$
  - $|Q| \approx \dfrac{|R| \cdot |S| \cdot |T|}{\max(|\pi_B R|, |\pi_B S|) \cdot \max(|\pi_C S|, |\pi_C T|)}$

# 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;
```

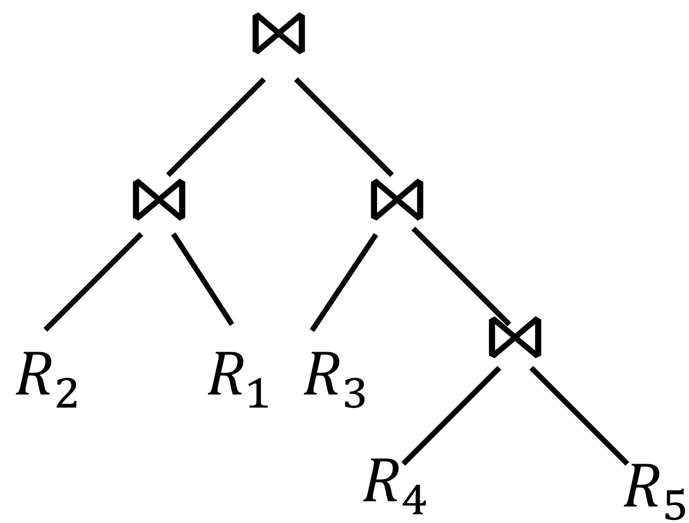
- Not covered: better estimation using histograms

# Search strategy


http://1.bp.blogspot.com/-Motdu8reRKs/TgyAi4ki5QI/AAAAAAAAAKE/mi8ejfZ8S7U/s1600/cornMaze.jpg

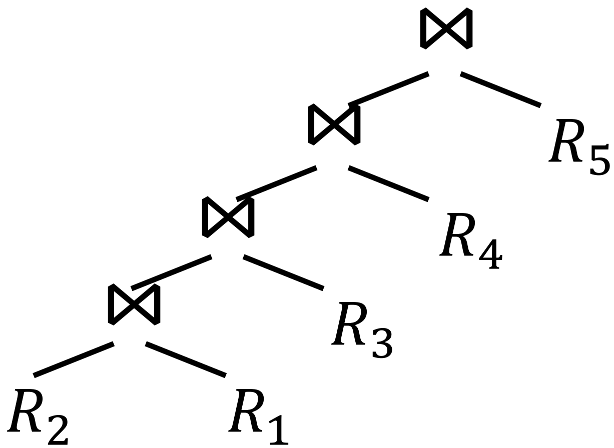
# Search space

- 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:
    - Multiway joins
    - Different join methods
    - Placement of selection and projection operators

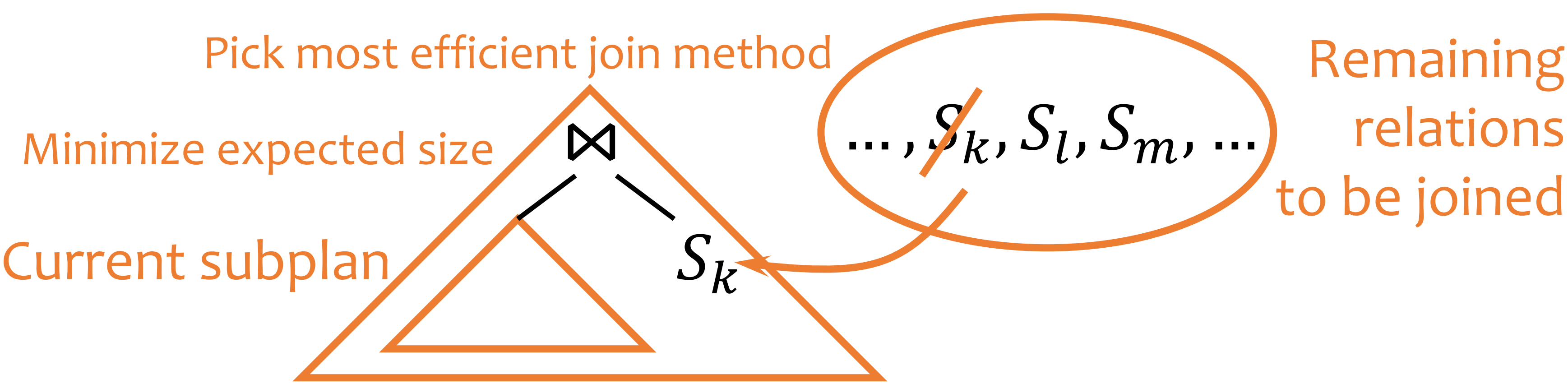
# Left-deep plans




- 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 $n = 6$)

# 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

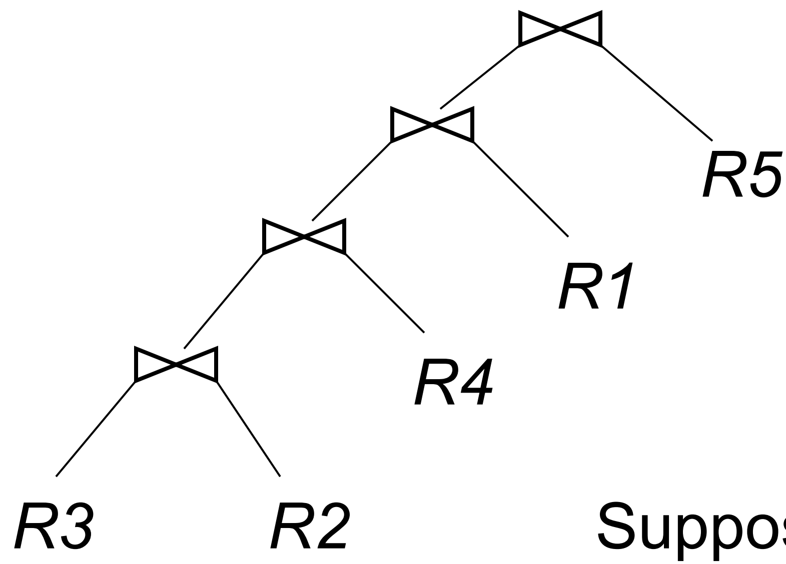


# Selinger's algorithm: A dynamic programming approach

Optimal for "whole" made up from optimal for "parts"

# Principle of Optimality

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

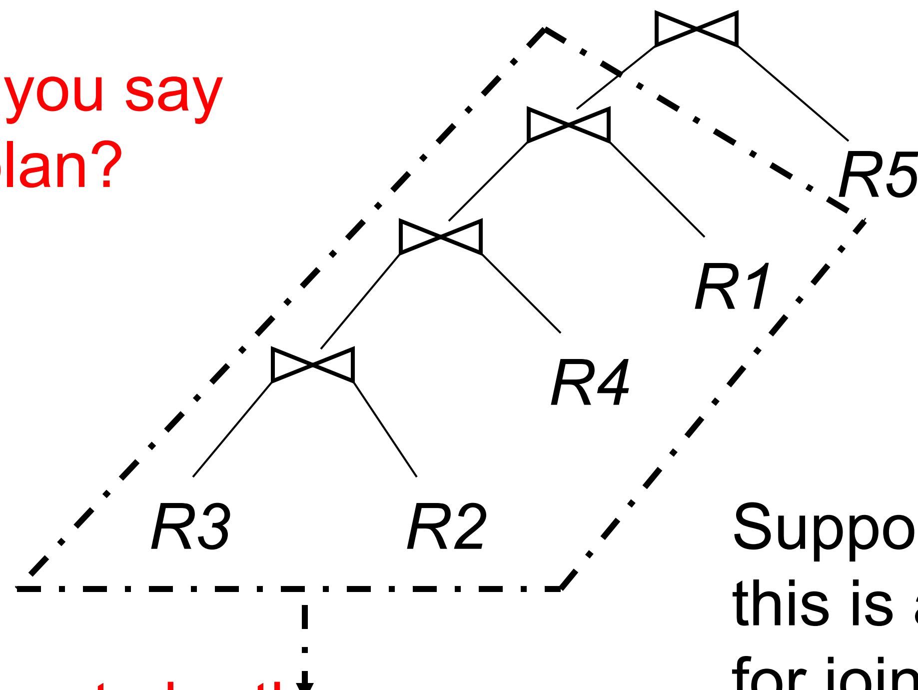



Suppose,
this is an Optimal Plan
for joining R1…R5:

# Principle of Optimality

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

Then, what can you say about this sub-plan?




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

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

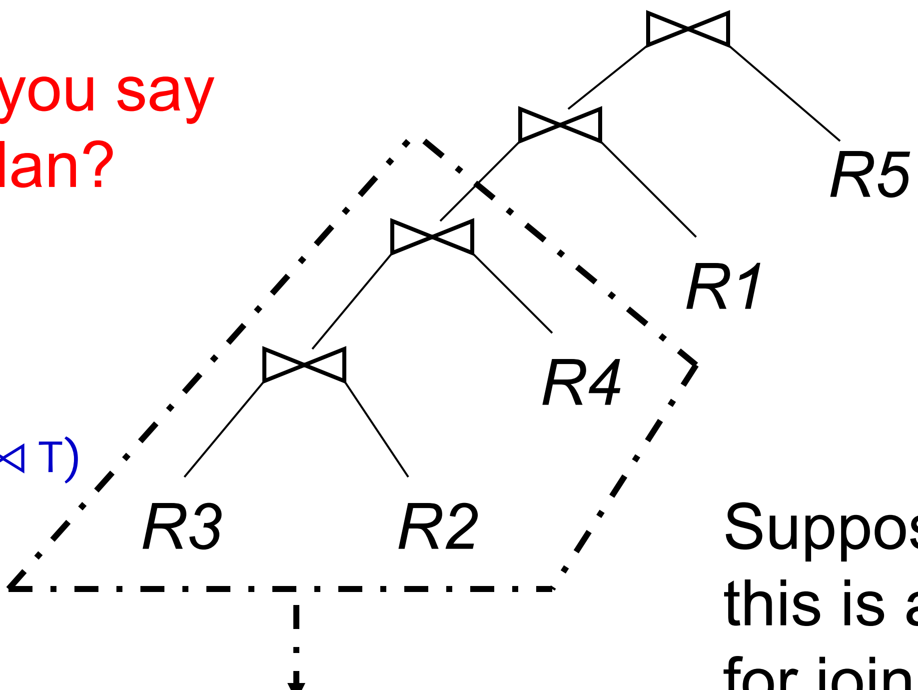
# Principle of Optimality

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

---

Then, what can you say about this sub-plan?

We are using the associativity and commutativity of joins
$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
$R \bowtie S = S \bowtie R$




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

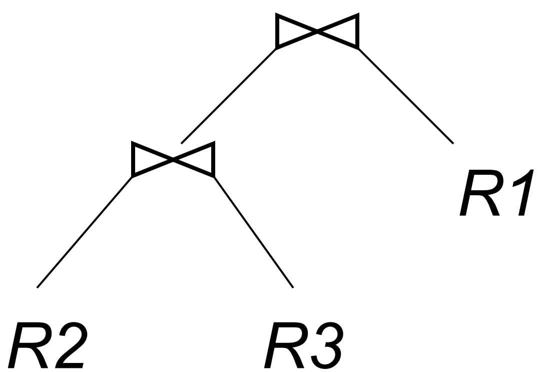
This has to be the optimal plan for joining *R3, R2, R4*

# Exploiting Principle of Optimality

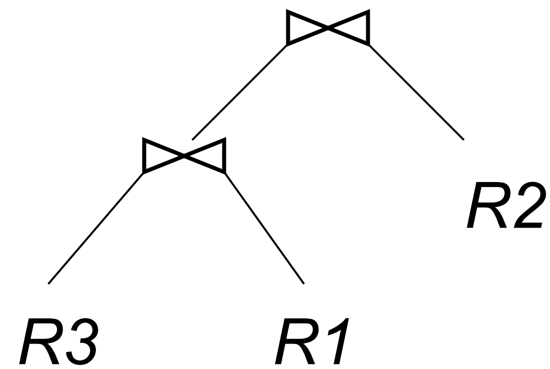
Query: $R1 \bowtie R2 \bowtie \quad \ldots \quad \bowtie Rn$

Both are giving the same result

$R2 \bowtie R3 \bowtie R1 = R3 \bowtie R1 \bowtie R2$




Optimal
for joining *R1, R2, R3*




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

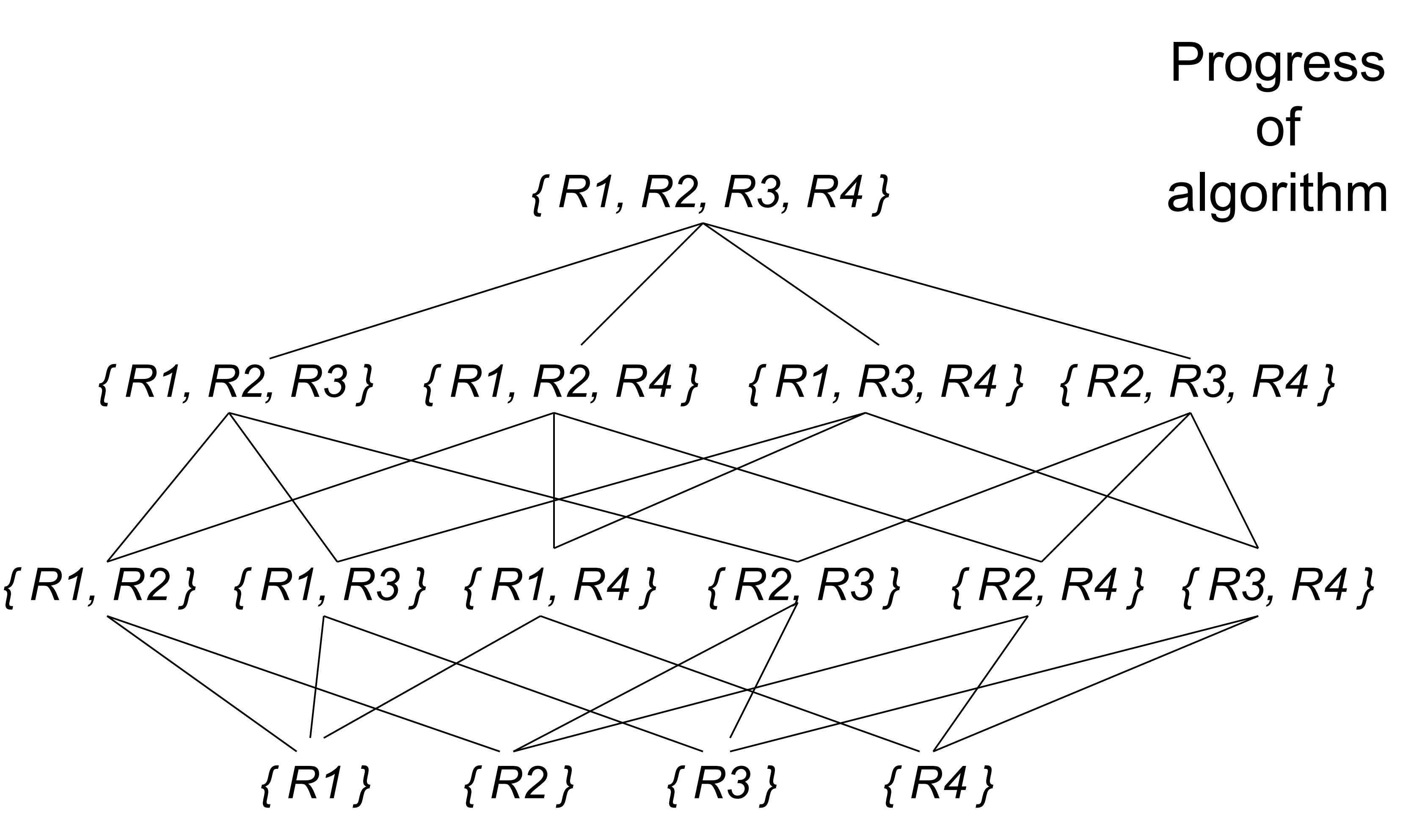
# Selinger Algorithm:

OPT ( { R1, R2, R3 } ):

$$
\text{Min} \begin{cases} \text{OPT ( \{ R1, R2 \} ) + cost-to-join (\{R1, R2 \}, \{R3\})} \\ \text{OPT ( \{ R2, R3 \} ) + cost-to-join (\{R2, R3 \}, \{R1\})} \\ \text{OPT ( \{ R1, R3 \} ) + cost-to-join (\{R1, R3 \}, \{R2\})} \end{cases}
$$

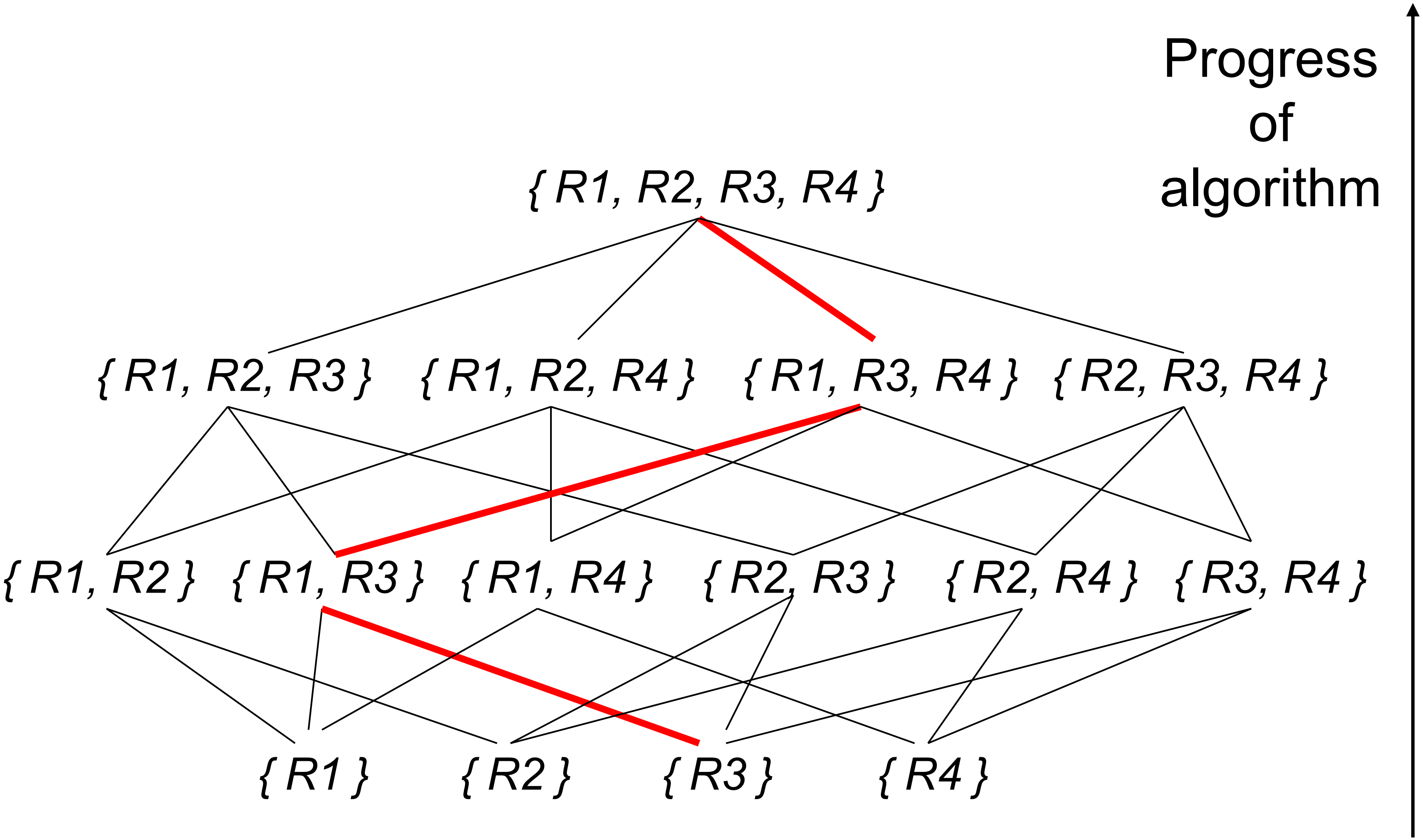
# Selinger Algorithm:

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



# Selinger Algorithm:

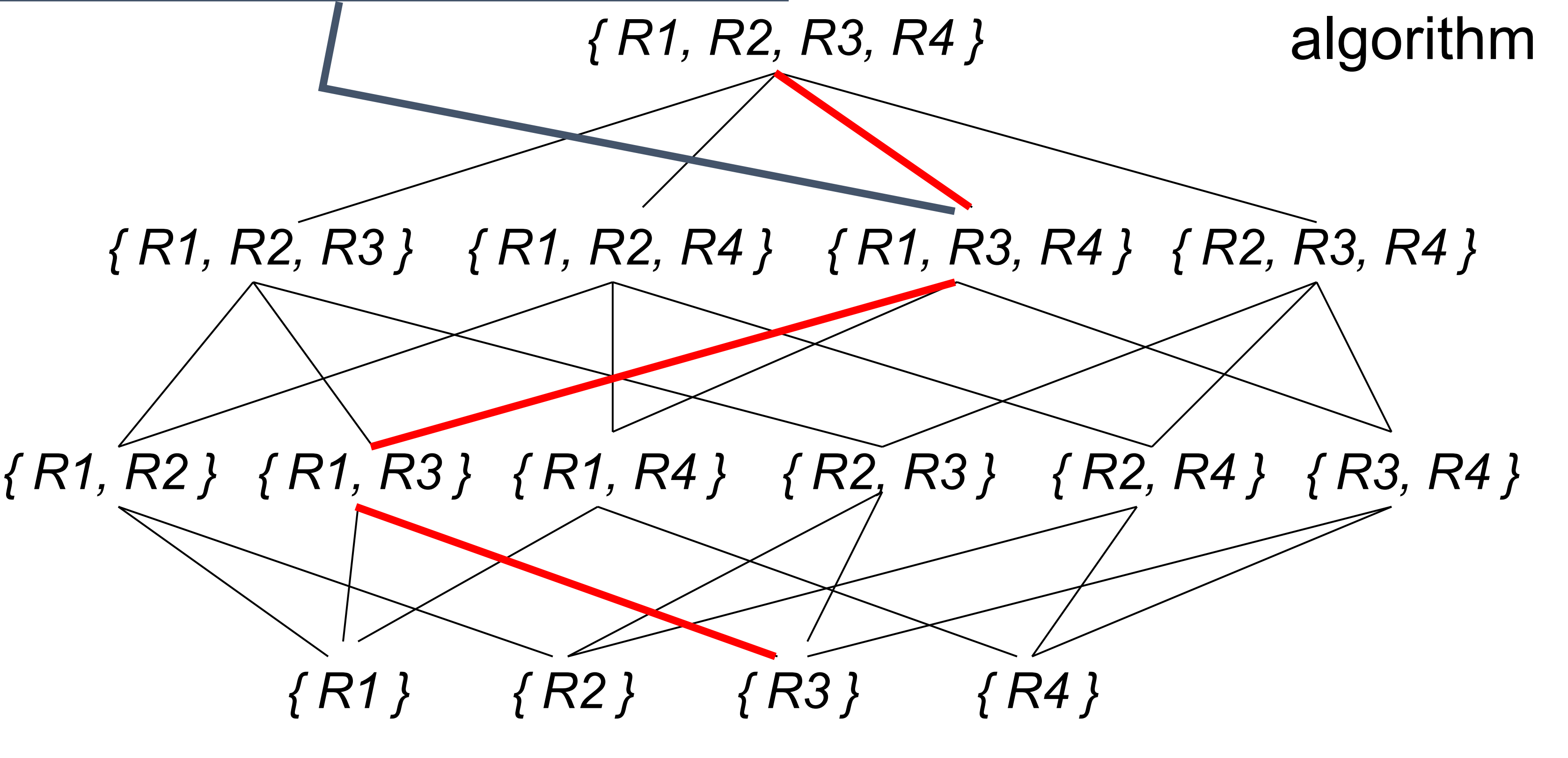
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



# Selinger Algorithm:

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

e.g. All possible permutations of R1, R3, R4 have been considered after OPT({R1, R3, R4}) has been computed

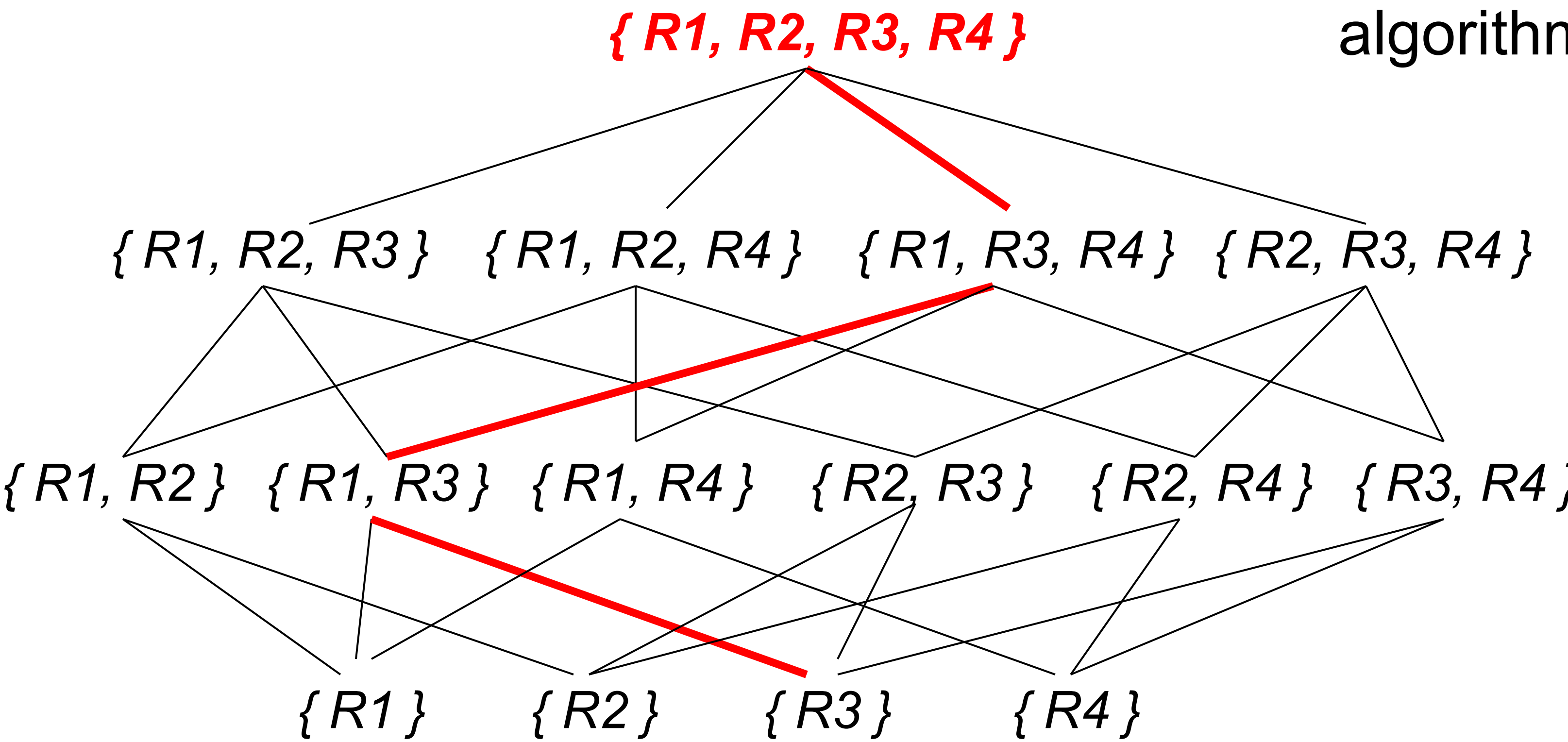


# Selinger Algorithm:

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.



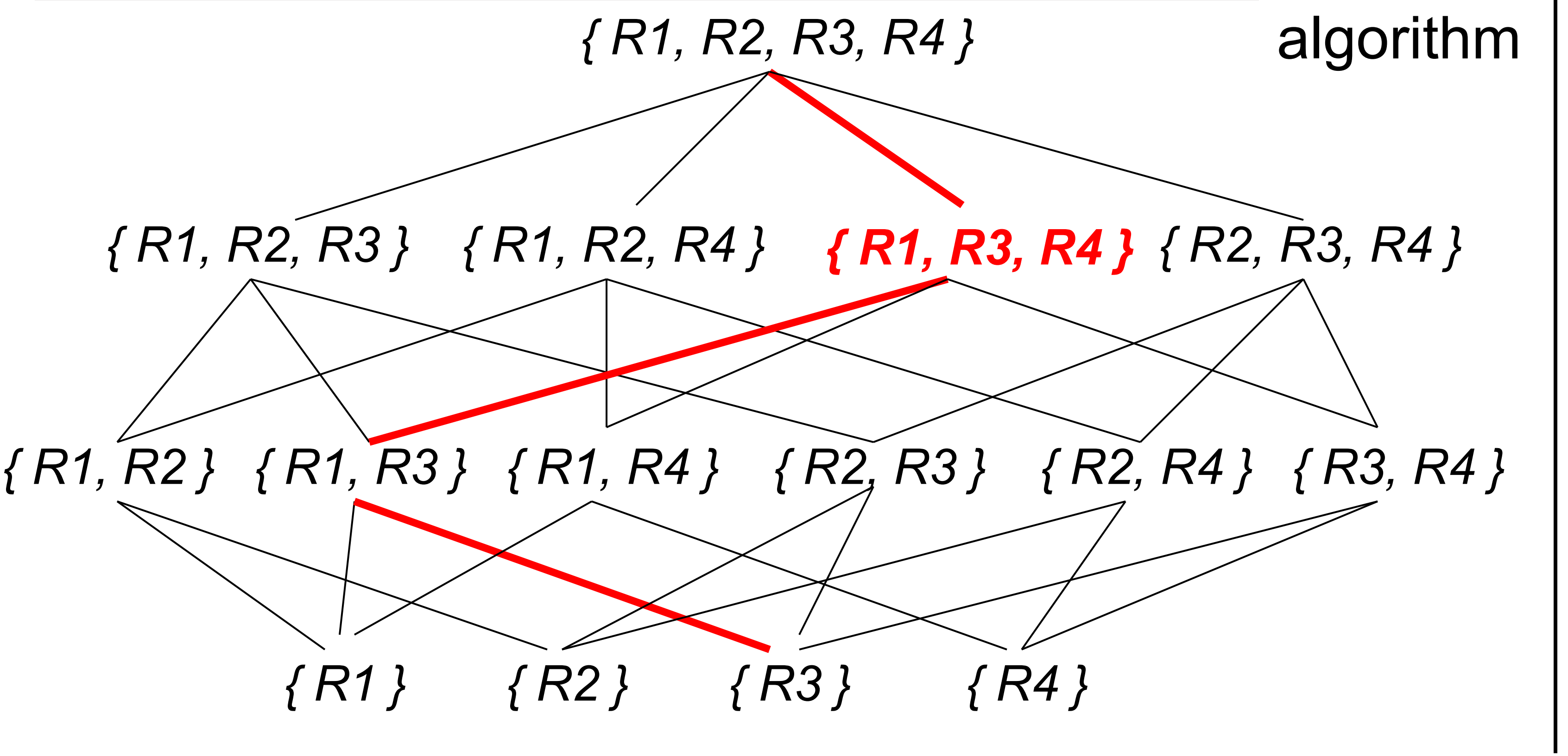

Progress of algorithm

# 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.

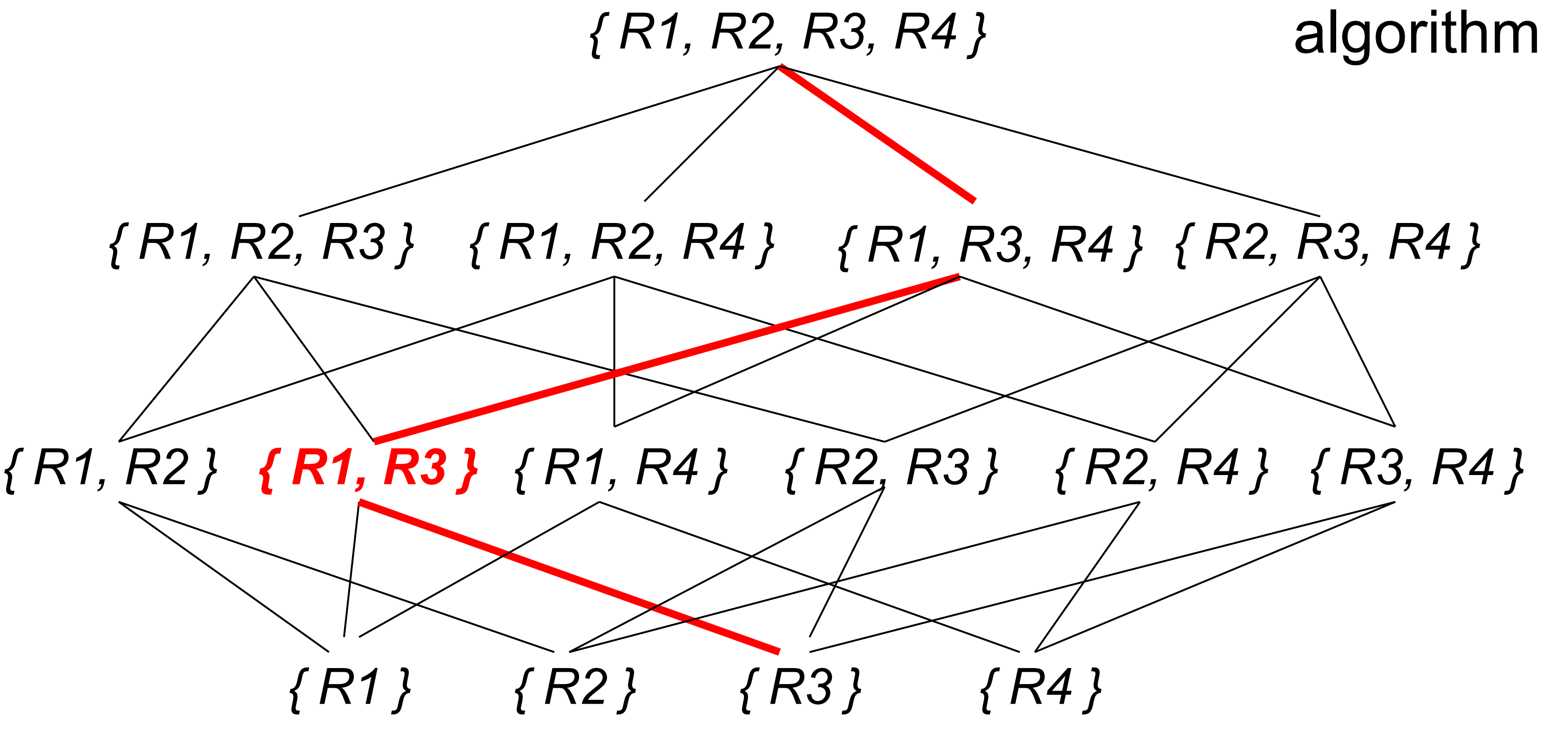


# Selinger Algorithm:

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.

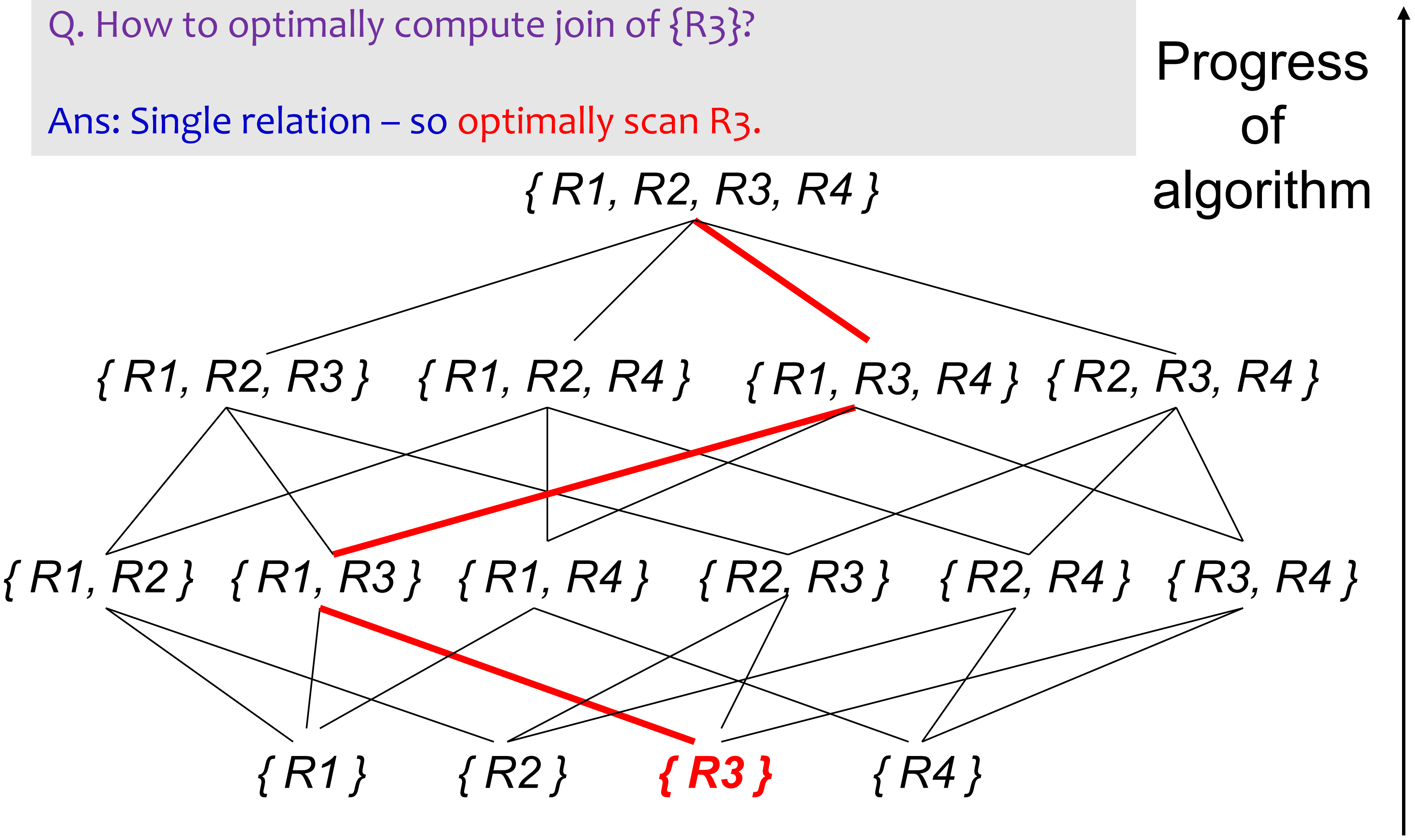


# Selinger Algorithm:

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

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

Ans: Single relation – so optimally scan R3.

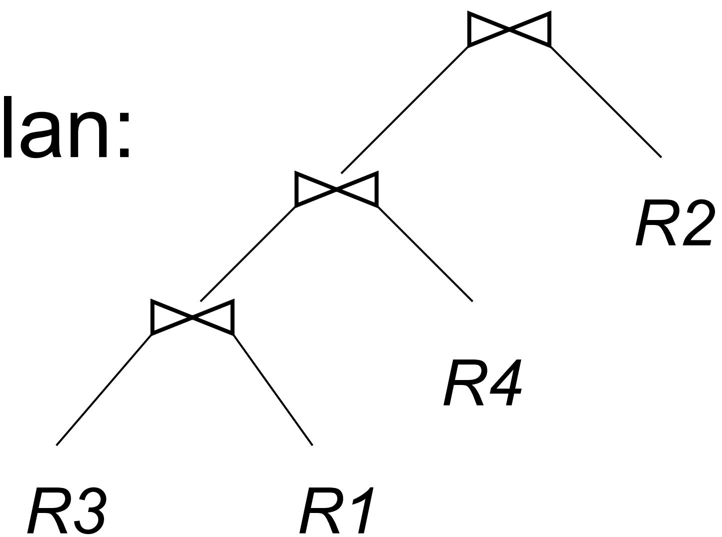


# Selinger Algorithm:

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

# The need for "interesting order"

- Optimal plan may not have an optimal sub-plan in practice!
- Example: $R(A, B) \bowtie S(A, C) \bowtie T(A, D)$
- Best plan for $R \bowtie S$: hash join (beats sort-merge join)
- Best overall plan: sort-merge join $R$ and $S$, and then sort-merge join with $T$
  - Subplan of the optimal plan is not optimal!
- Why?
  - 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 $Y$
- Need to keep a set of optimal plans for joining every combination of $k$ tables
  - At most one for each interesting order

# Summary

- Relational algebra equivalence
- SQL rewrite tricks
- Heuristics-based optimization
- Cost-based optimization
  - Need statistics to estimate sizes of intermediate results
  - Greedy approach
  - Dynamic programming approach

# Practice problem:
# Estimating the cost of the entire plan

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author)
C(<u>sid,bid</u>,date)

no. of tuples

T(S)=10,000
T(B)=50,000
T(C)=300,000

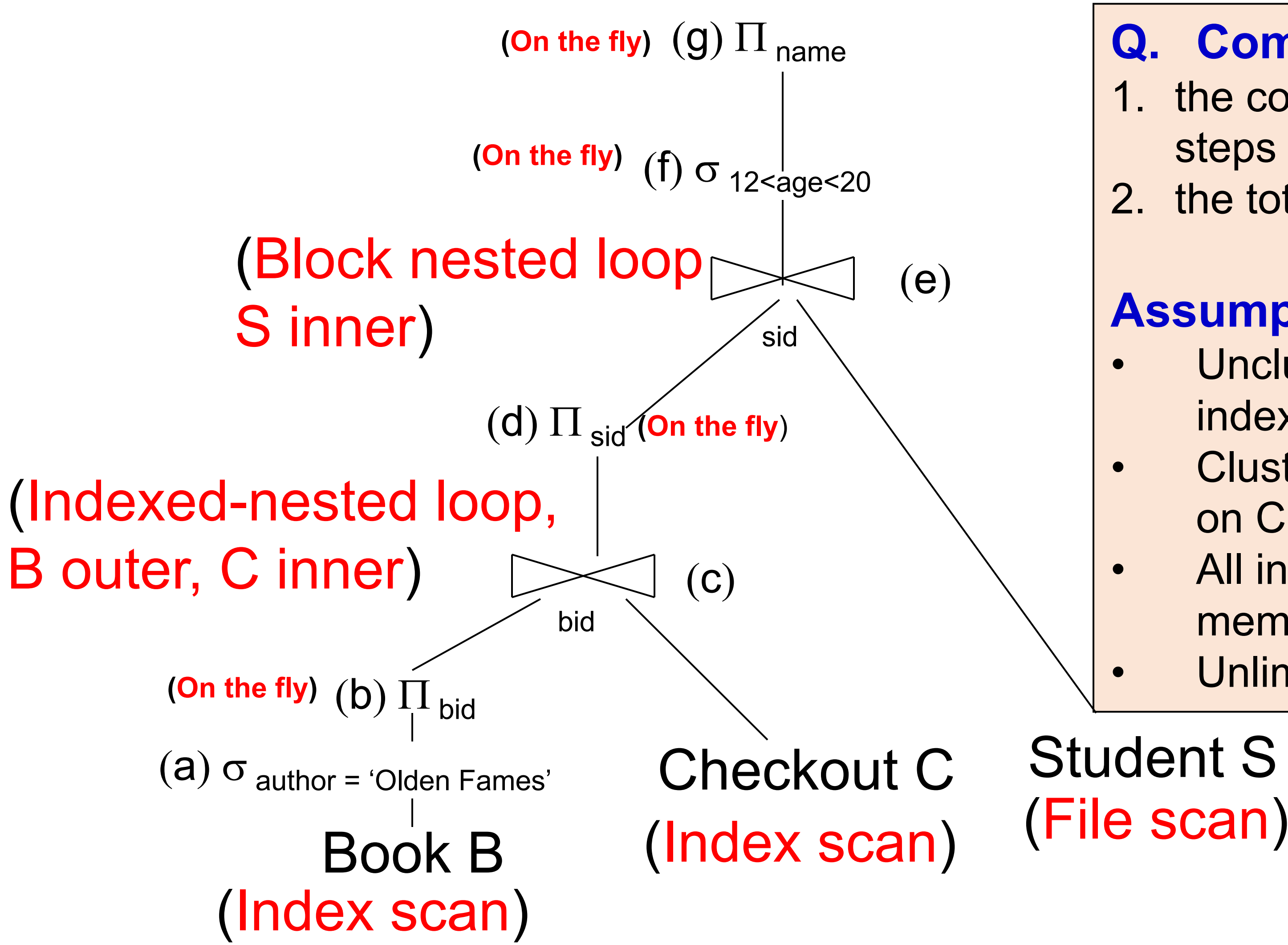
no. of pages

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

V(B,author) = 500
7 <= age <= 24

# Physical Query Plan




**Q. Compute**

1. the cost and cardinality in steps (a) to (g)
2. the total cost

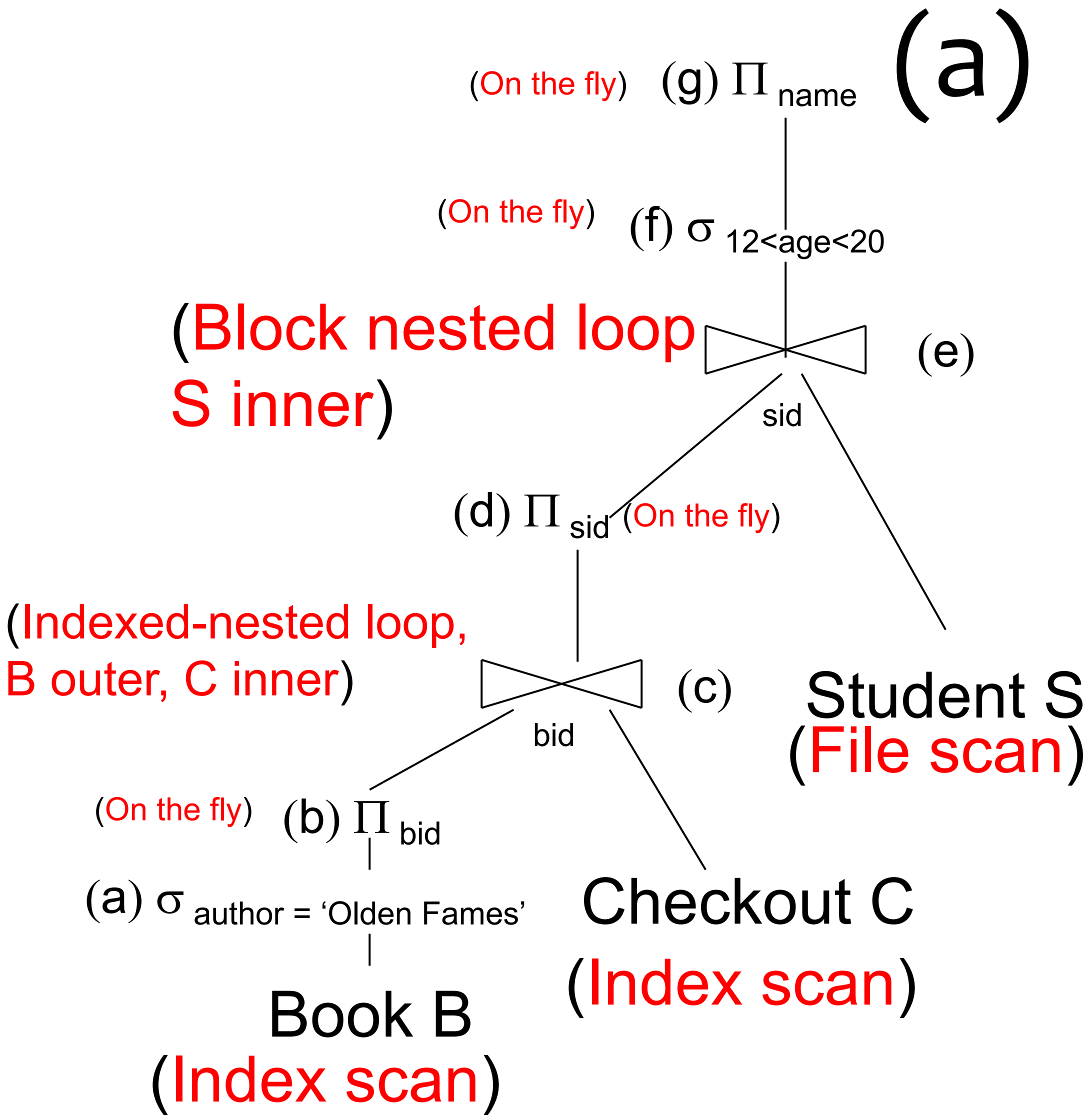
**Assumptions (given):**

- Unclustered B+tree index on B.author
- Clustered B+tree index on C.bid
- All index pages are in memory
- Unlimited memory

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

T(S)=10,000 B(S)=1,000
T(B)=50,000 B(B)=5,000
T(C)=300,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24




**Cost =**
T(B) / V(B, author)
= 50,000/500
= 100 (unclustered)

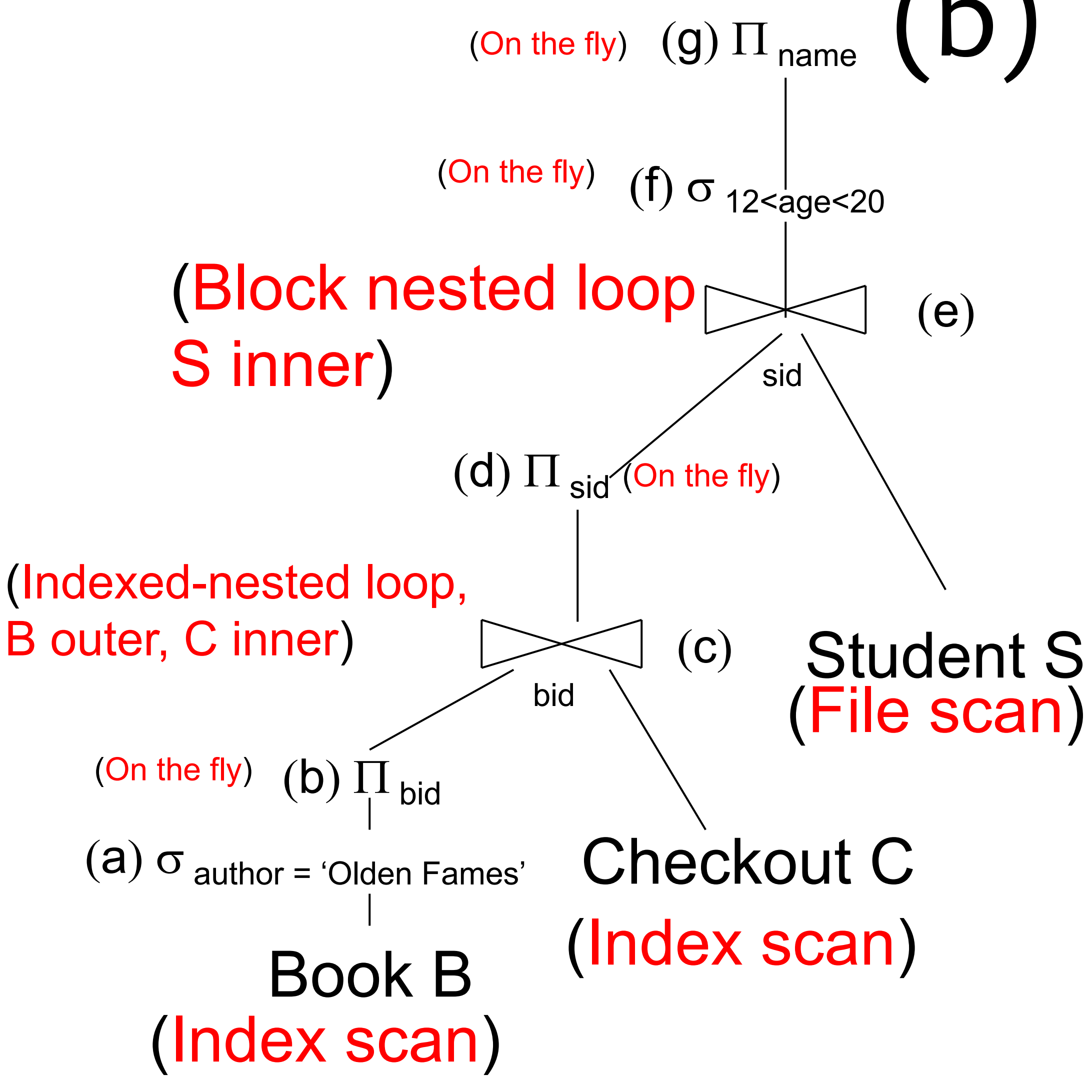
**Cardinality =**
100

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

T(S)=10,000 B(S)=1,000
T(B)=50,000 B(B)=5,000
T(C)=300,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24

# (b)




**Cost =**
0 (on the fly)

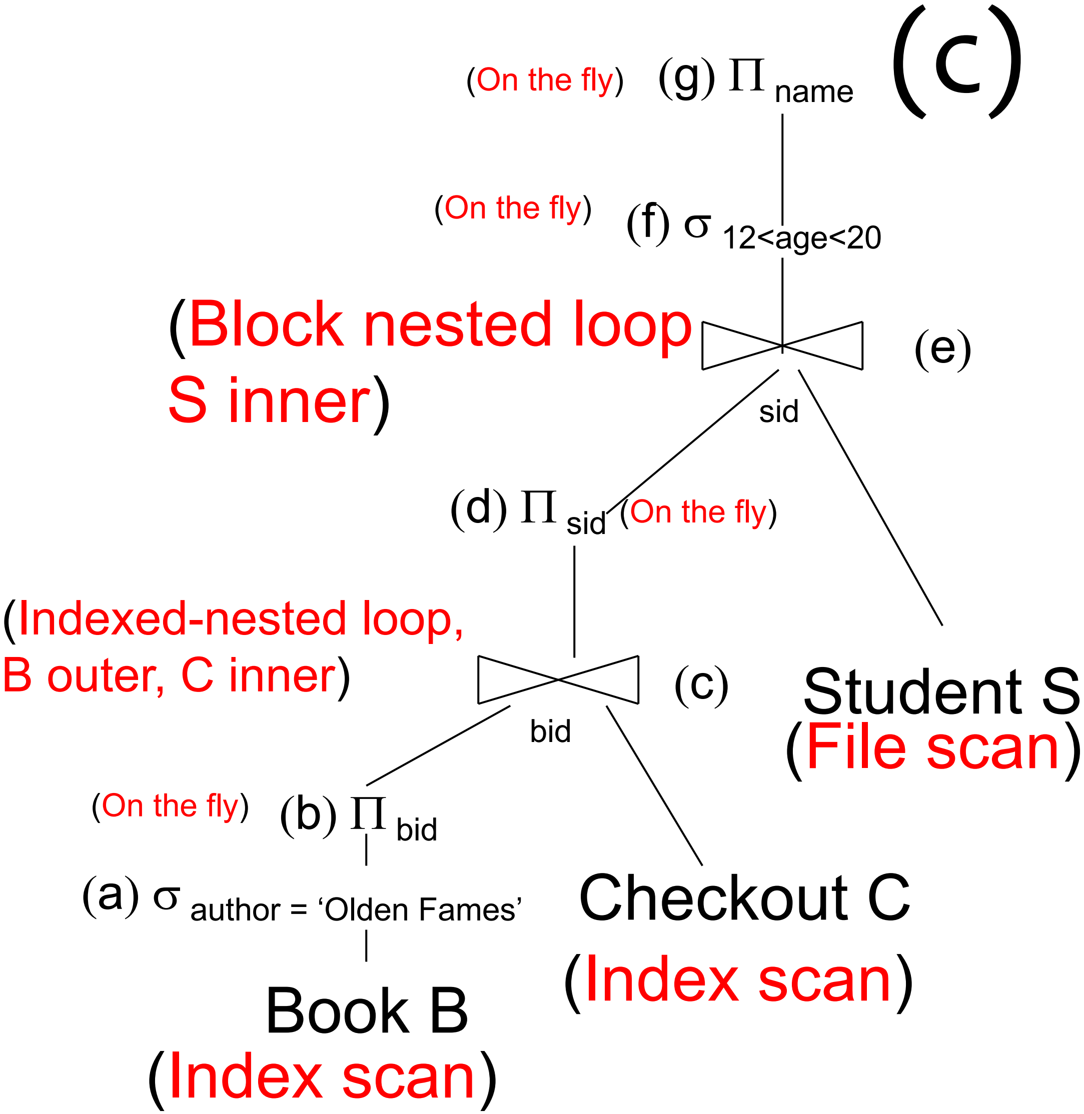
**Cardinality =**
100

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24




- one index lookup per outer B tuple
- 1 book has T(C)/ T(B) = 6 checkouts (uniformity)
- # C tuples per page = T(C)/B(C) = 20
- 6 tuples fit in at most 2 consecutive pages (clustered) could assume 1 page as well

**Cost <=**
100 * 2= 200

**Cardinality =**
100 * 6 = 600

= 100 * T(C)/ MAX(100, V(C, bid)) assuming
V(C, bid) = V(B, bid) = T(B) = 50,000

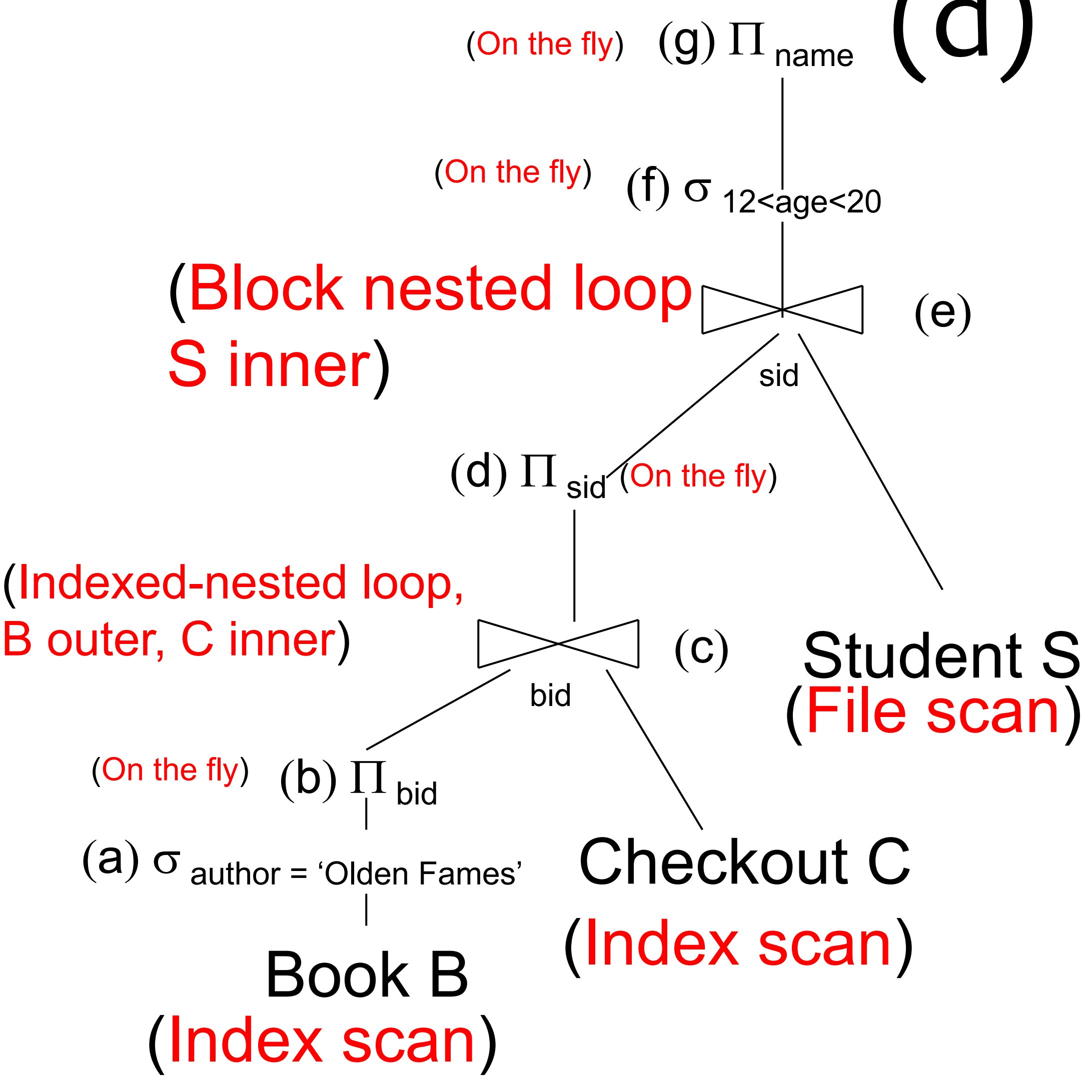
S(sid,name,age,addr)
B(bid,title,author): Un. B+ on author
C(sid,bid,date): Cl. B+ on bid

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

(d)



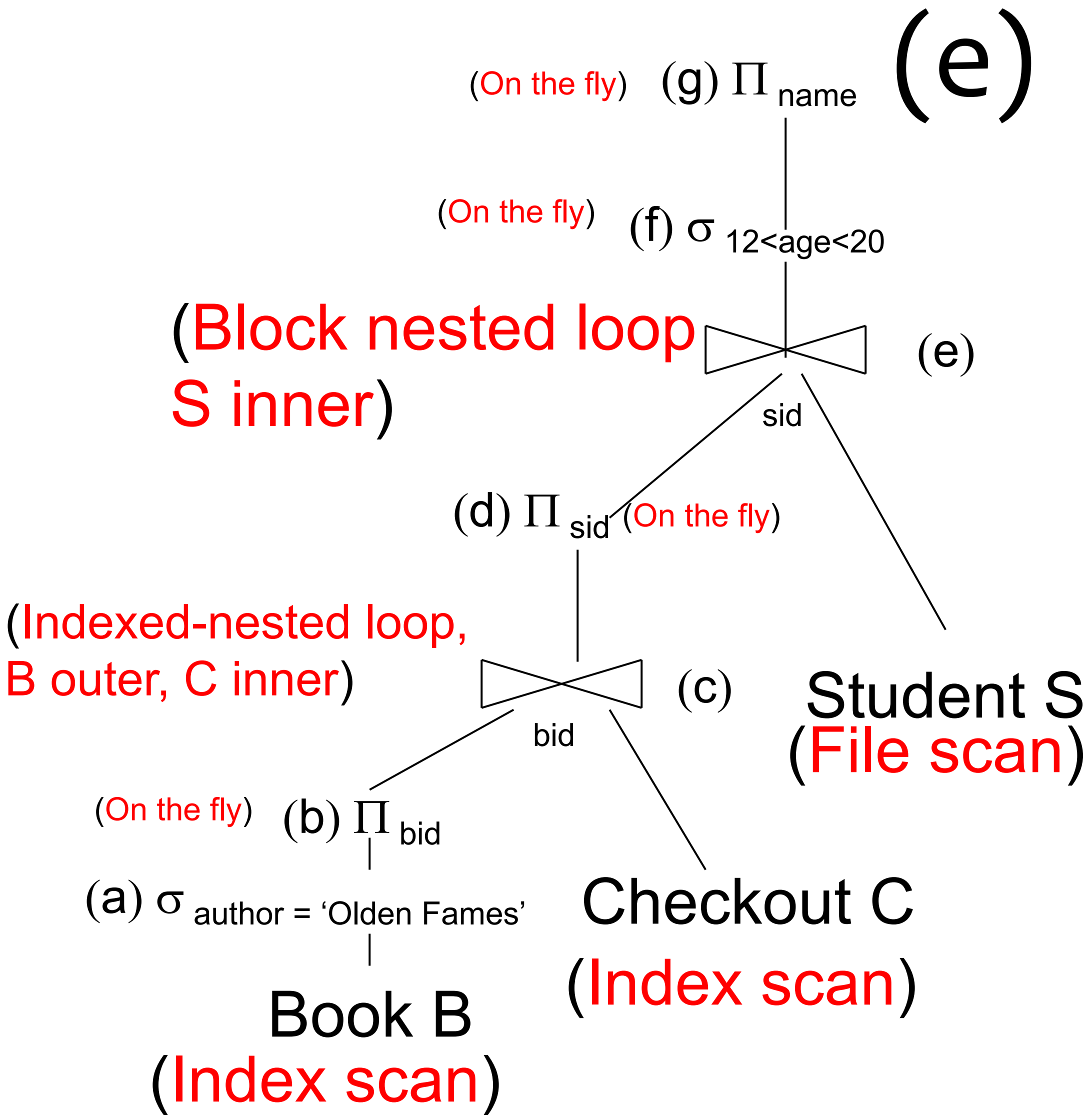

**Cost =**
**0** (on the fly)

**Cardinality =**
600

S(sid,name,age,addr)
B(bid,title,author): Un. B+ on author
C(sid,bid,date): Cl. B+ on bid

T(S)=10,000 B(S)=1,000
T(B)=50,000 B(B)=5,000
T(C)=300,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24




Outer relation is already in (unlimited) memory need to scan S relation

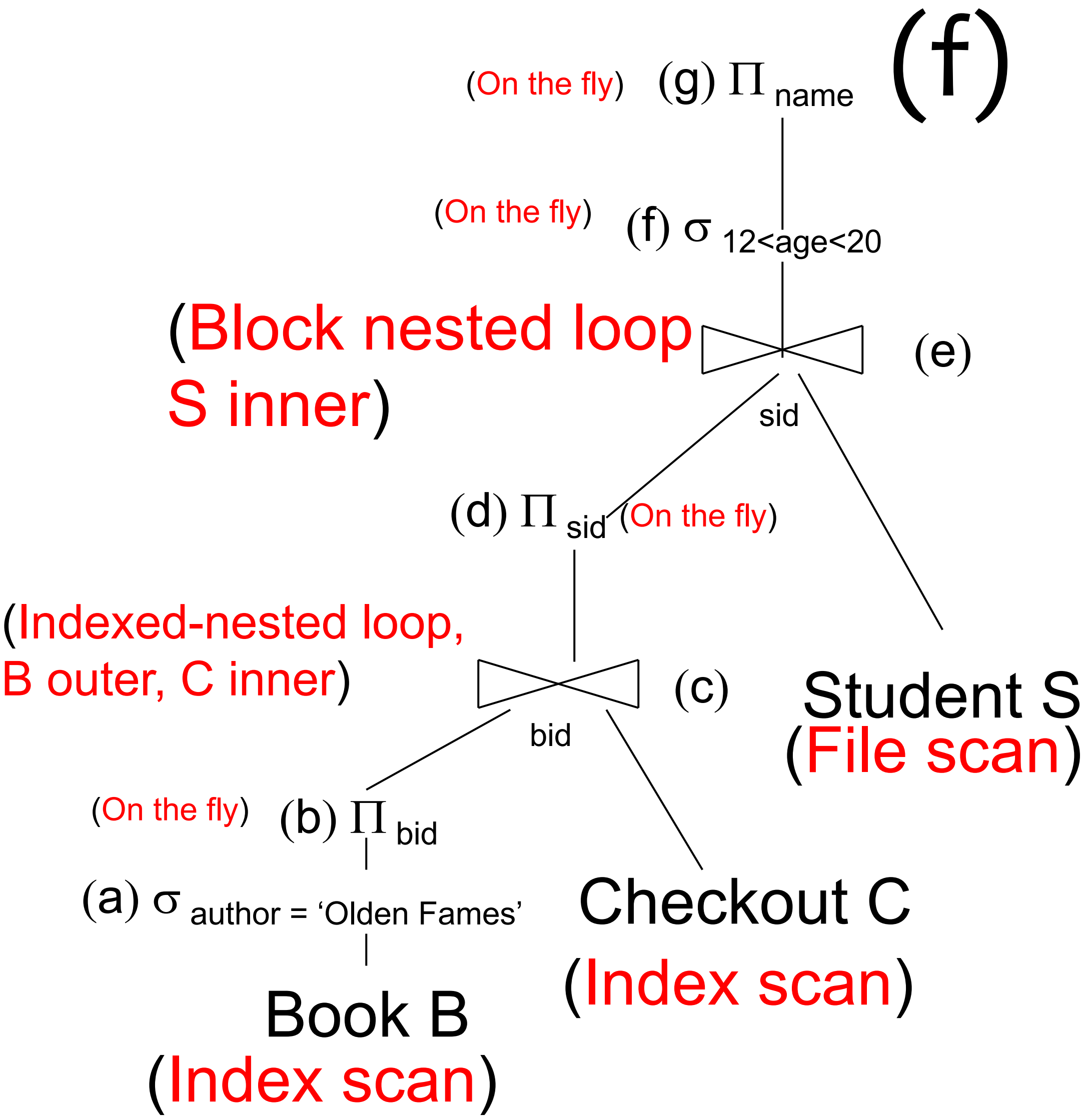
**Cost =**
B(S) = 1000

**Cardinality =**
600
(one student per checkout)

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

T(S)=10,000 B(S)=1,000
T(B)=50,000 B(B)=5,000
T(C)=300,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24




**Cost =**
**0** (on the fly)

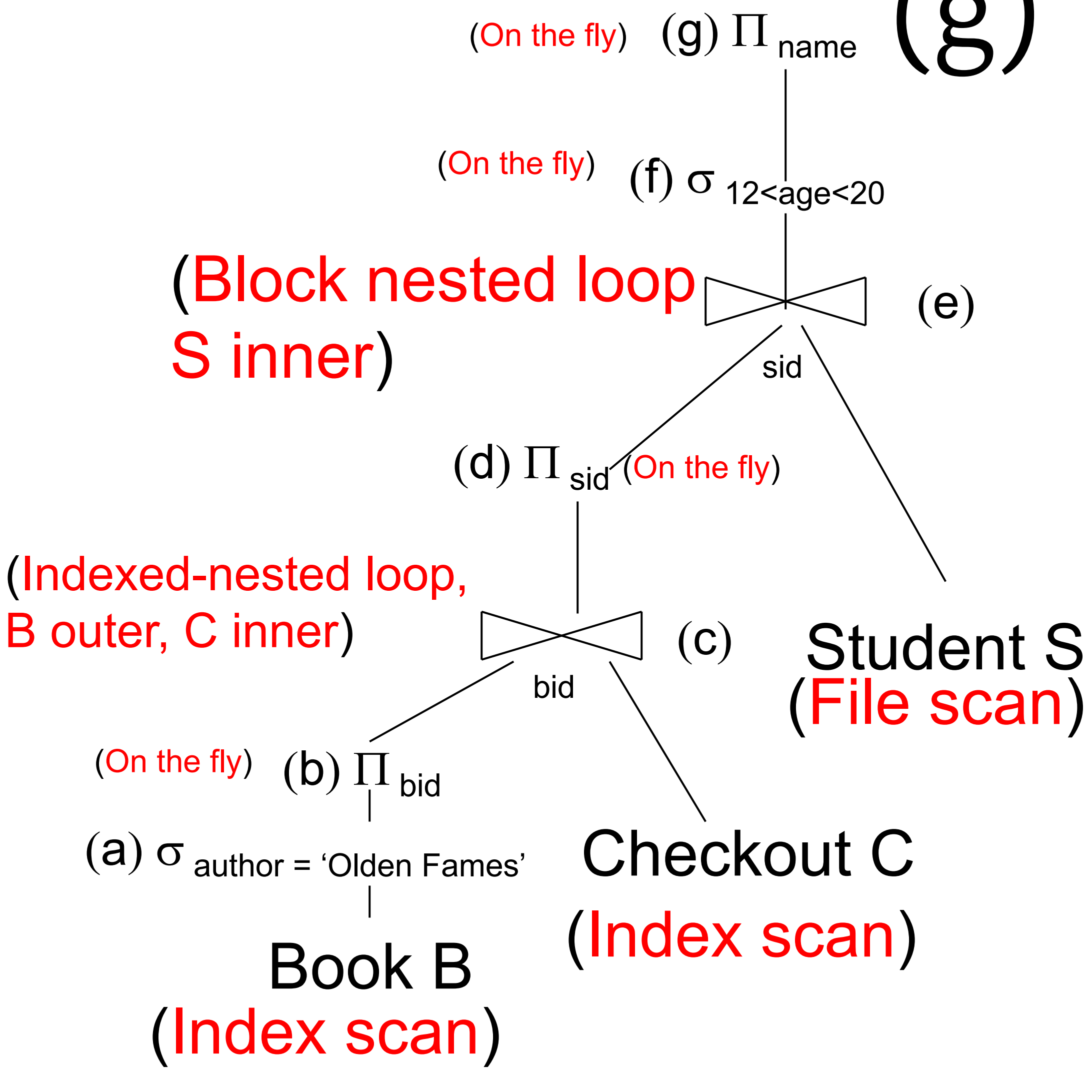
**Cardinality =**
600 * 7/18 = 234 (approx)

S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

$T(S)=10,000$ $B(S)=1,000$
$T(B)=50,000$ $B(B)=5,000$
$T(C)=300,000$ $B(C)=15,000$

$V(B,author) = 500$
$7 <= age <= 24$

# (g)




**Cost =**
**0** (on the fly)

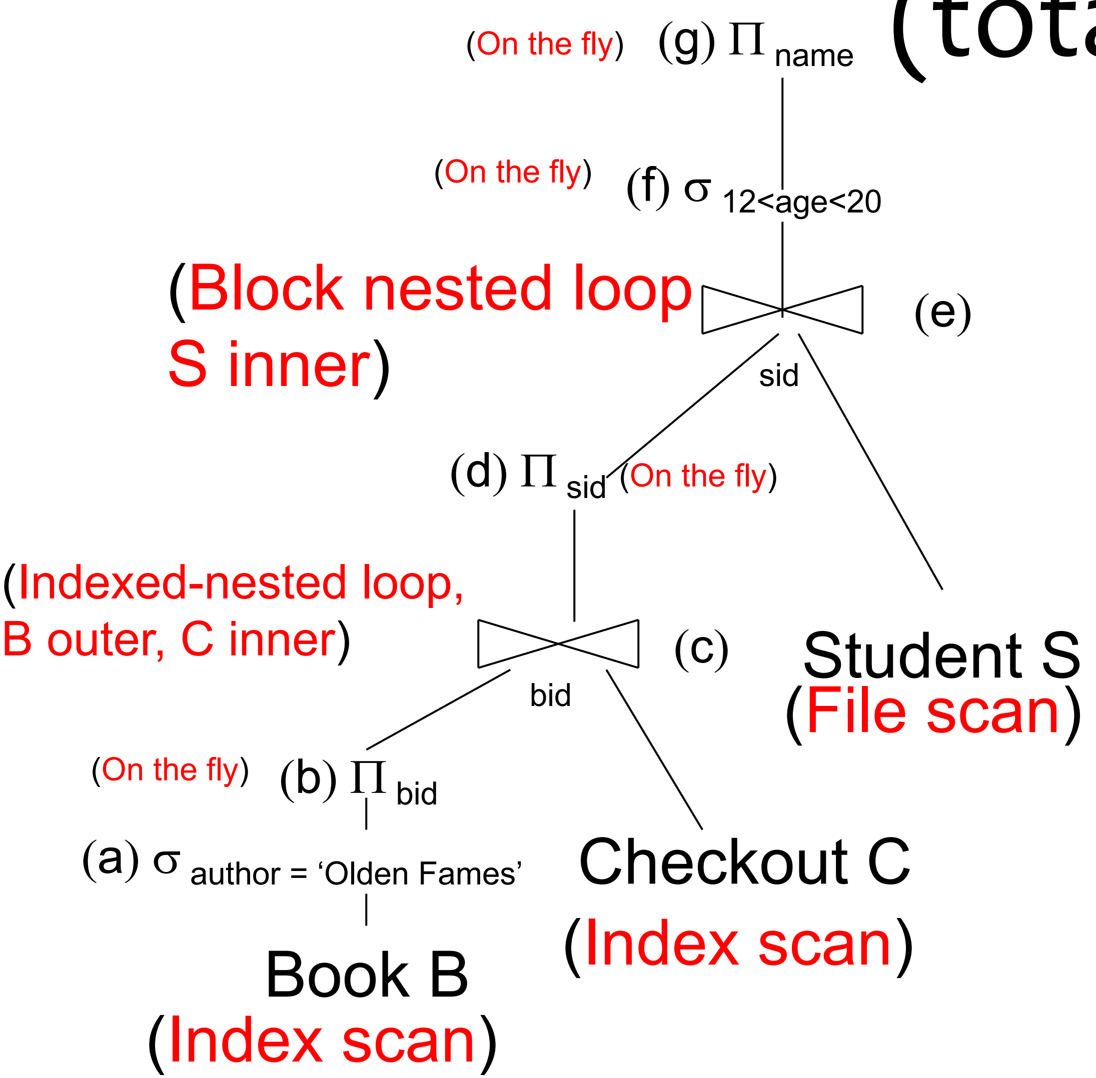
**Cardinality =**
234

S(sid,name,age,addr)
B(bid,title,author): Un. B+ on author
C(sid,bid,date): Cl. B+ on bid

T(S)=10,000 B(S)=1,000
T(B)=50,000 B(B)=5,000
T(C)=300,000 B(C)=15,000

V(B,author) = 500
7 <= age <= 24

# (total)




**Total cost = 1300**

**Final cardinality = 234 (approx)**