CompSci 516
Data Intensive Computing Systems

Lecture 10
Cost-based Query Optimization

Instructor: Sudeepa Roy

Announcements

- Solution of Homework-1 has been posted on sakai
  - Many equivalent solutions of the queries are possible
- Homework-2 has been posted
  - Due on February 29, Monday, 11:55 pm
  - Goal: review all key concepts covered so far, and practice for exams
  - Start early
  - Ask questions on piazza
- Xiaodan's office hour canceled this week
  - Will be rescheduled
- Lecture Pdfs will be (mostly) posted right before the class
  - Don' t forget to see the updated version after the class

What will we learn?

- Last lecture:
  - Estimating cost of all operators and join algorithms
- Next:
  - Combine cost in a plan
  - Query Optimization

Reading Material

- [GUW]
  - Chapter 16.2-16.7

- Original paper by Selinger et al. :
  - P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. *Access Path Selection in a Relational Database Management System*
    Proceedings of ACM SIGMOD, 1979. Pages 22-34
  - No need to understand the whole paper, but take a look at the example (link on the course webpage)

Acknowledgement:
Some of the following slides have been created by adapting slides by Profs. Shivnath Babu and Magda Balazinska

Notation

- T(R) : Number of tuples in R
- B(R) : Number of blocks in R
- V(R, A) : Number of distinct values of attribute A in R

Query Optimization Problem

Pick the best plan from the space of physical plans

Cost-based Query Optimization

Pick the plan with least cost

Challenge:

- Do not want to execute more than one plans
- Need to estimate the cost without executing the plan

"heuristic-based" optimizer (e.g. push selections down) have limited power and not used much

Cost-based Query Optimization

Pick the plan with least cost

Tasks:

1. Estimate the cost of individual operators

   done

2. Estimate the size of output of individual operators

   today

3. Combine costs of different operators in a plan

   today

4. Efficiently search the space of plans today

Task 1 and 2
Estimating cost and size of different operators

- Size = #tuples, NOT #pages
- Cost = #page I/O
  - but, need to consider whether the intermediate relation fits in memory, is written back to/read from disk (or on-the-fly goes to the next operator), etc.

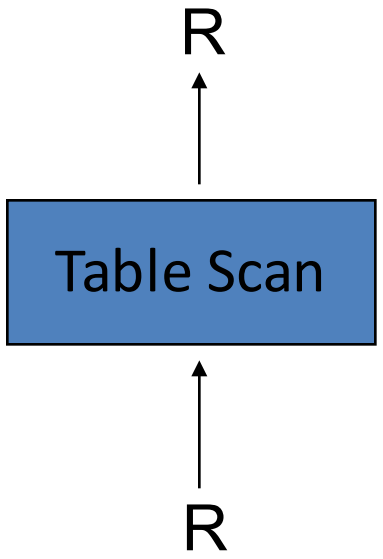
Desired Properties of Estimating Sizes of Intermediate Relations

Ideally,

- should give accurate estimates (as much as possible)
- should be easy to compute
- should be logically consistent
  - size estimate should be independent of how the relation is computed
  - e.g. which join algorithm/join order is used

- But, no "universally agreed upon" ways to meet these goals

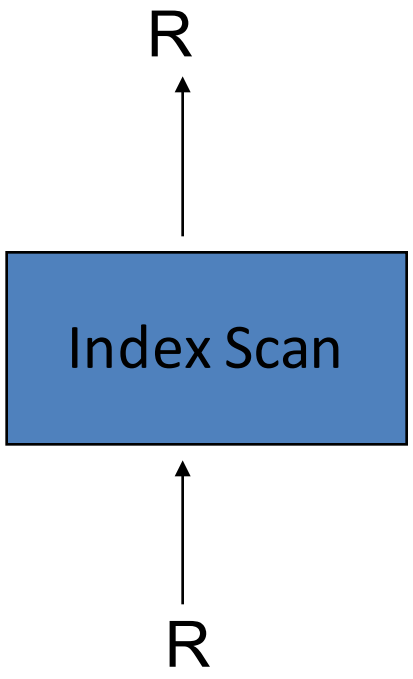
Cost of Table Scan



Cost: B(R)
Size: T(R)

T (R) : Number of tuples in R
B (R) : Number of blocks in R

Cost of Index Scan



Cost: B (R) – if clustered
T(R) – if unclustered

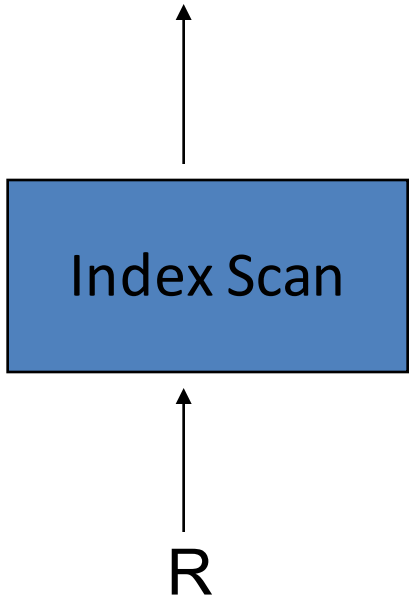
Size: T(R)

T (R) : Number of tuples in R
B (R) : Number of blocks in R

Note: size is independent of the implementation of the scan/index

Cost of Index Scan with Selection

$X = \sigma_{R.A > 50} R$



Cost: B(R) * f – if clustered
T(R) * f – if unclustered

Size: T(R) * f

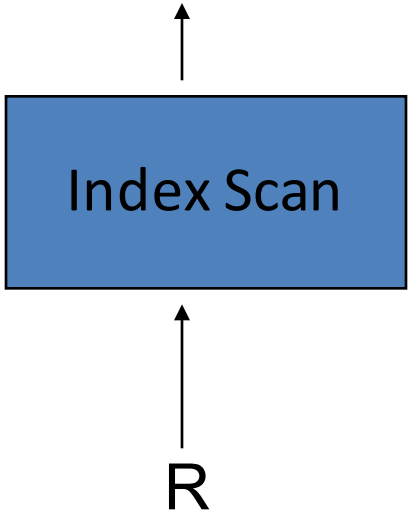
T (R) : Number of tuples in R
B (R) : Number of blocks in R

Reduction factor
f = Max(R.A) - 50) / (Max(R.A) - Min(R.A))
(assumes uniform distribution)

Cost of Index Scan with Selection (and multiple conditions)

$X = \sigma_{R.A > 50 \text{ and } R.B = C} R$

assume index on (A, B)



What is f1 if the first condition is 100 > R.1 > 50?

Cost: B(R) * f – if clustered
T(R) * f – if unclustered

Size: T(R) * f

Reduction factors

range selection

f1 = Max(R.A) - 50) / (Max(R.A) - Min(R.A))

f2 = T(R) / V(R, B)

value selection

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

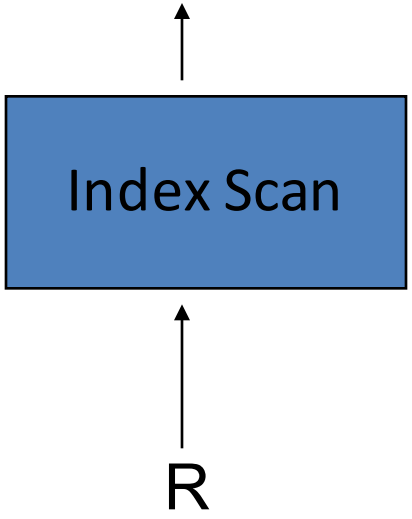
f = f1 * f2 (assumes independence and uniform distribution)

Cost of Index Scan with Selection (and multiple conditions)

$X = \sigma_{R.A > 50 \text{ and } R.B = C} R$

assume index on (A, B)

What is f if



Cost: B(R) * f – if clustered
T(R) * f – if unclustered

Size: T(R) * f

Reduction factors

range selection

f1 = Max(R.A) - 50) / (Max(R.A) - Min(R.A))

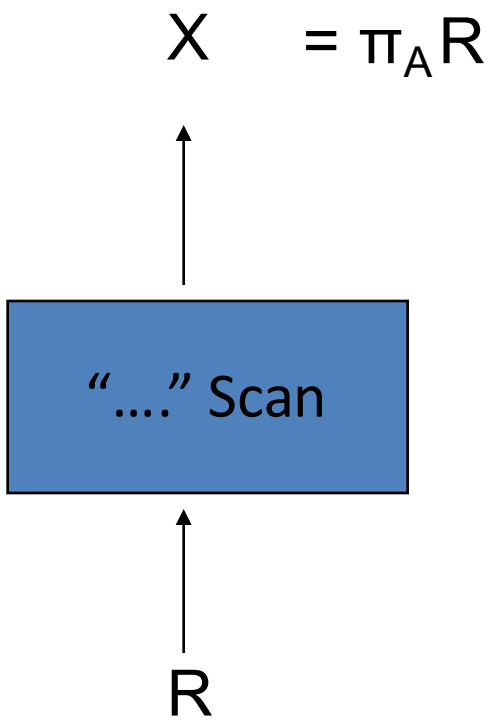
f2 = T(R) / V(R, B)

value selection

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

f = f1 * f2 (assumes independence and uniform distribution)

Cost of Projection



Cost: depends on the method of scanning R

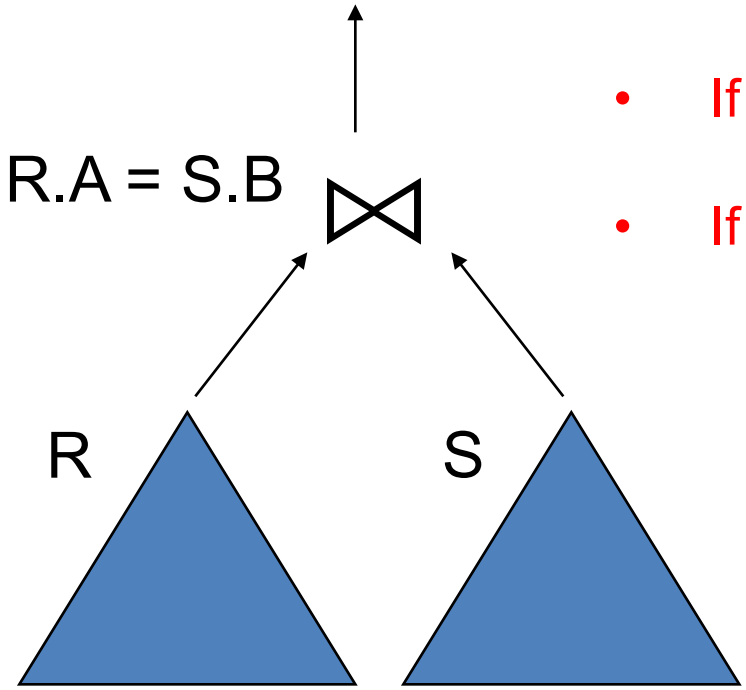
B(R) for table scan or clustered index scan

Size: T(R)

But tuples are smaller

If you have more information on the size of the smaller tuples, can estimate #I/O better

Size of Join



Quite tricky

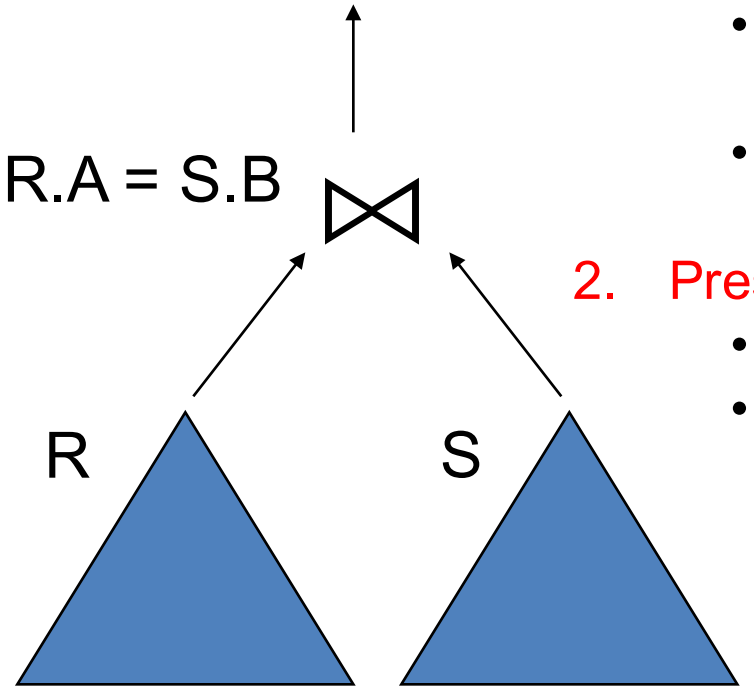
- If disjoint A and B values
  - then 0
- If A is key of R and B is foreign key of S
  - then T(S)
- If all tuples have the same value of R.A= S.B = x
  - then T(R) * T(S)

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

Size of Join

Two assumptions

1. Containment of value sets:
   - if V(R, A) <= V(S, B), then all A-values of R are included in B-values of S
   - e.g. satisfied when A is foreign key, B is key
2. Preservation of value sets:
   - V(R ⋈ S, A or B) = V(R, A) = V(S, B)
   - No value is lost in join



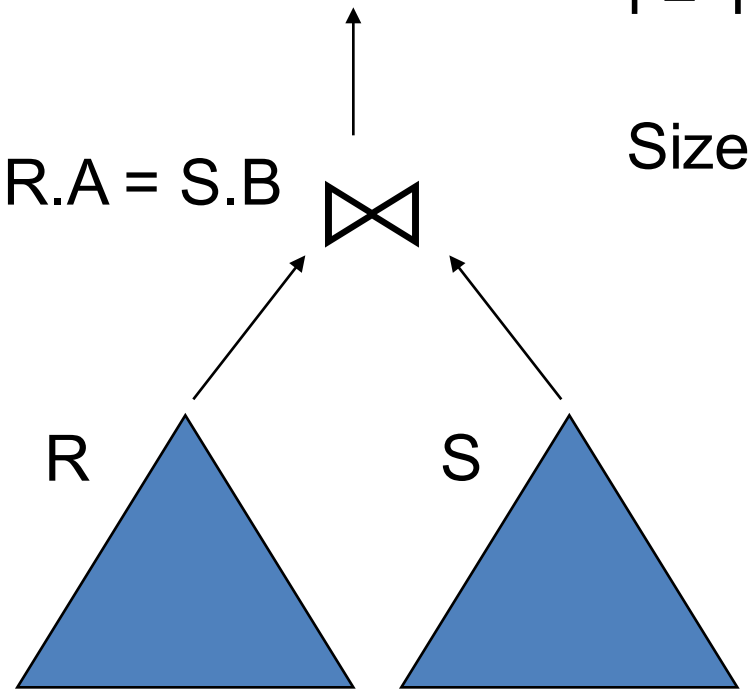
T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

Size of Join

Reduction factor

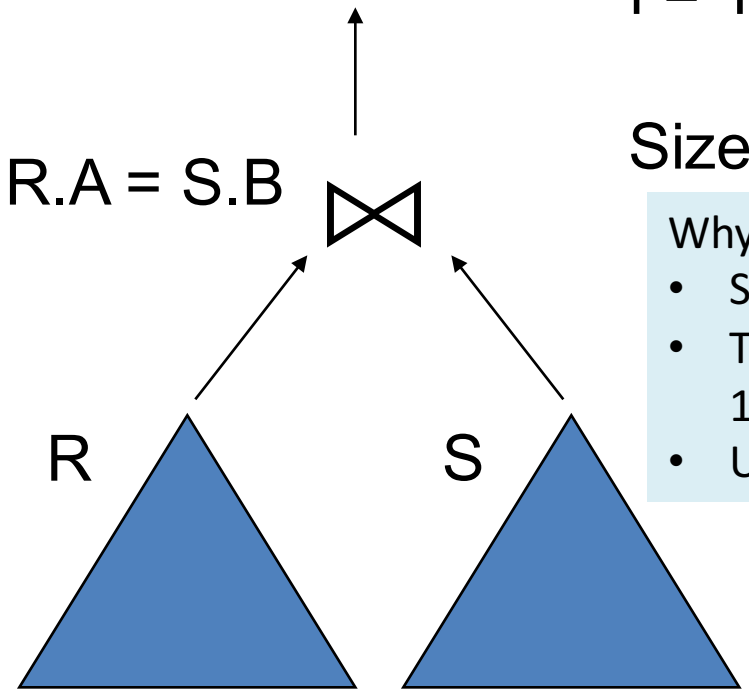
f = 1/max(V(R, A), V(S, B))

Size = T(R) * T(S) * f



T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

Size of Join



Reduction factor

f = 1/max(V(R, A), V(S, B))

Size = T(R) * T(S) * f

Why max?

- Suppose V(R, A) <= V(S, B)
- The probability of a A-value joining with a B-value is 1/V(S.B) = reduction factor
- Under the two assumptions stated earlier + uniformity

T (R) : Number of tuples in R
B (R) : Number of blocks in R
V(R, A) : Number of distinct values of attribute A in R

Task 3: Combine cost of different operators in a plan

With Examples

"Given" the physical plan

- Size = #tuples, NOT #pages
- Cost = #page I/O
  - but, need to consider whether the intermediate relation fits in memory, is written back to disk (or on-the-fly goes to the next operator) etc.

Example Query

Student (sid, name, age, address)
Book(bid, title, author)
Checkout(sid, bid, date)

Query:

```
SELECT S.name
FROM Student S, Book B, Checkout C
WHERE S.sid = C.sid
AND B.bid = C.bid
AND B.author = 'Olden Fames'
AND S.age > 12
AND S.age < 20
```

S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

Assumptions

- Student: S, Book: B, Checkout: C

- Sid, bid foreign key in C referencing S and B resp.
- There are 10,000 Student records stored on 1,000 pages.
- There are 50,000 Book records stored on 5,000 pages.
- There are 300,000 Checkout records stored on 15,000 pages.
- There are 500 different authors.
- Student ages range from 7 to 24.

Warning: a few dense slides next ☺

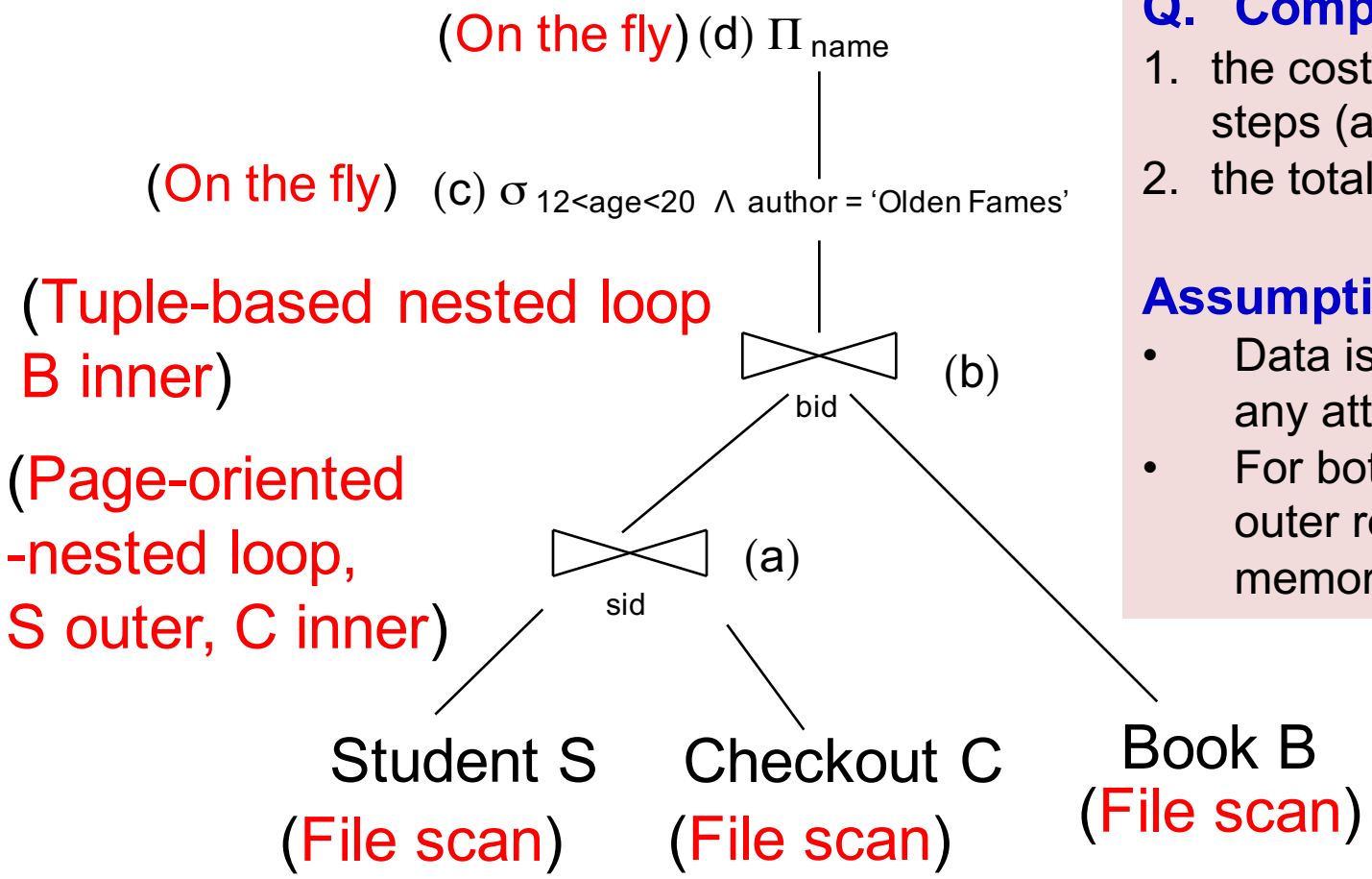
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

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

Physical Query Plan – 1



(Tuple-based nested loop B inner)

(Page-oriented -nested loop, S outer, C inner)

Q. Compute

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

Assumptions:

- Data is not sorted on any attributes
- For both in (a) and (b), outer relations fit in memory

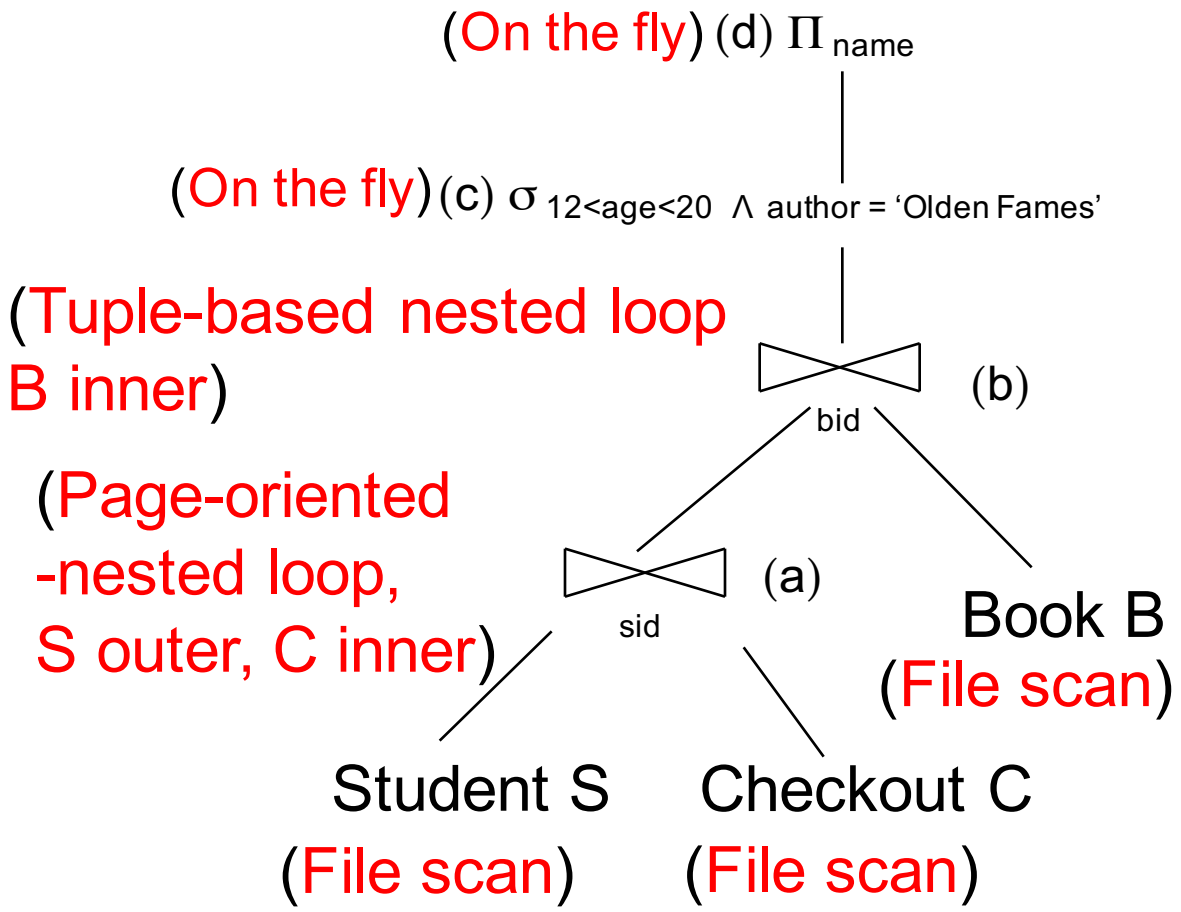
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

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

(a)



(Tuple-based nested loop B inner)

(Page-oriented -nested loop, S outer, C inner)

Cost =

B(S) + B(S) * B(C)
= 1000 + 1000 * 15000
= 15,001,000

Cardinality =

T(C) = 300,000

- foreign key join, output pipelined to next join
- Can apply the formula as well

T(S) * T(C)/max (V(S, sid), V(C, sid))
= T(S)
since V(S, sid) > = V(C, sid)
and
T(S) = V(S, sid)

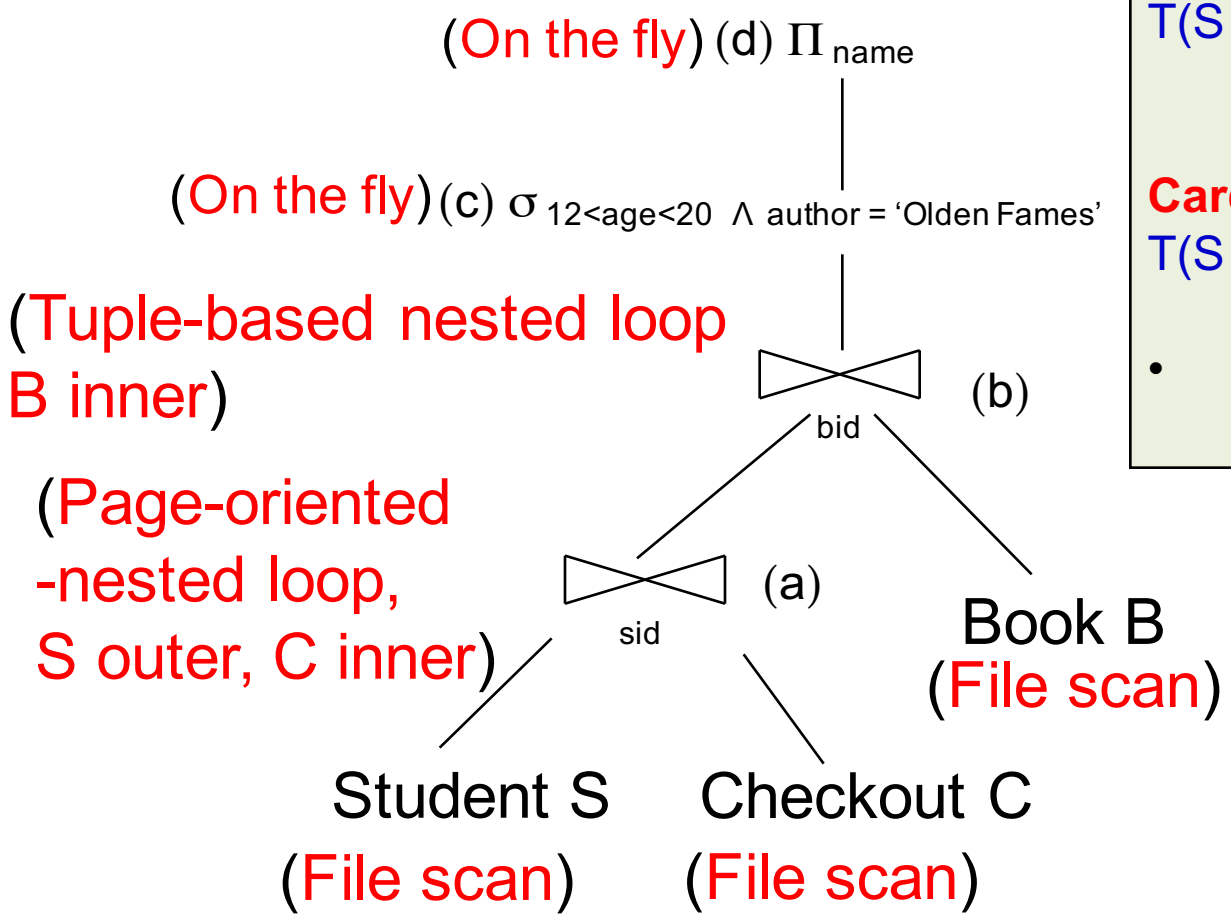
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

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

(b)



Cost =
T(S ⋈ C) * B(B)
= 300,000 * 5,000 = $15 * 10^8$

Cardinality =
T(S ⋈ C) = 300,000

- foreign key join, don't need scanning for outer relation

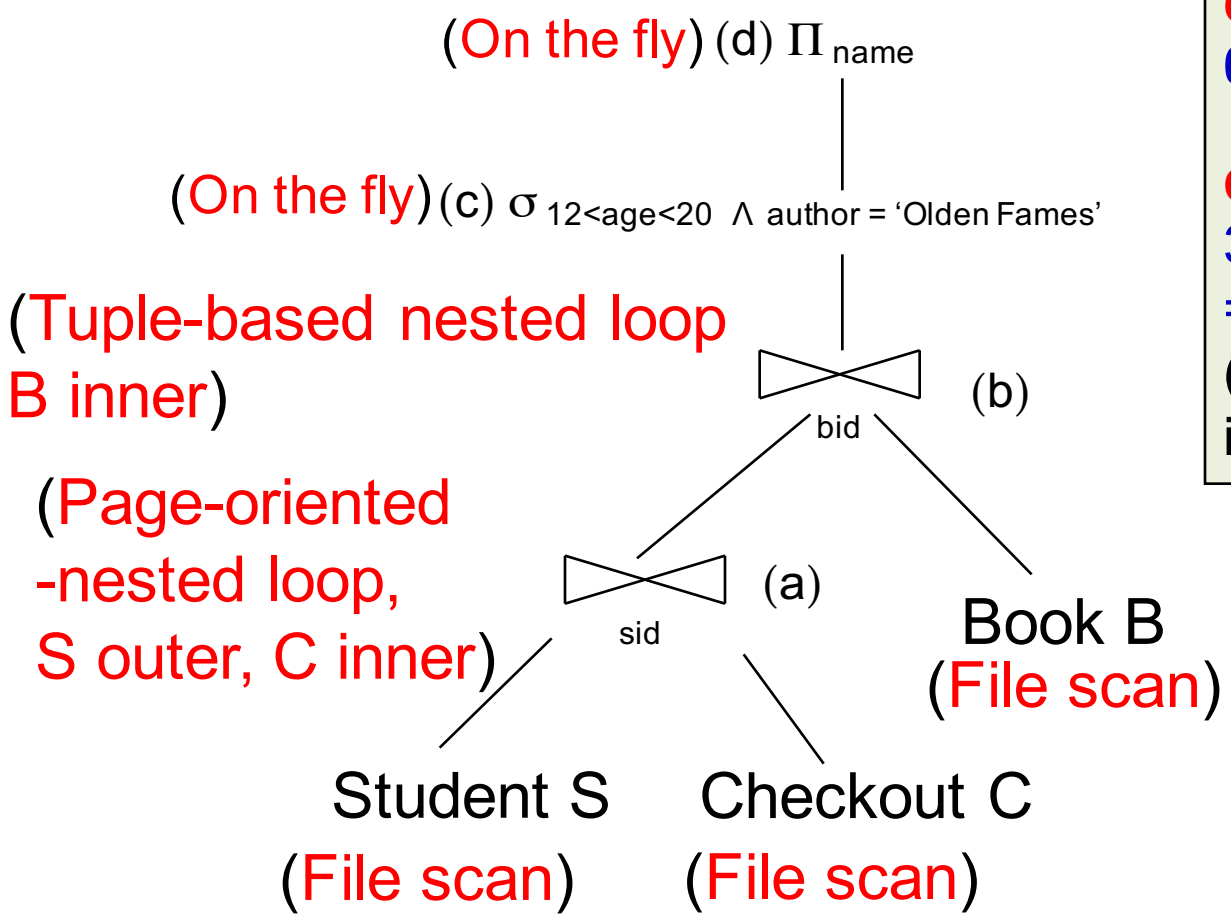
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

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

(c, d)



(Tuple-based nested loop B inner)

(Page-oriented -nested loop, S outer, C inner)

Cost =
0 (on the fly)

Cardinality =
300,000 * 1/500 * 7/18
= 234 (approx)
(assuming uniformity and independence)

S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

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

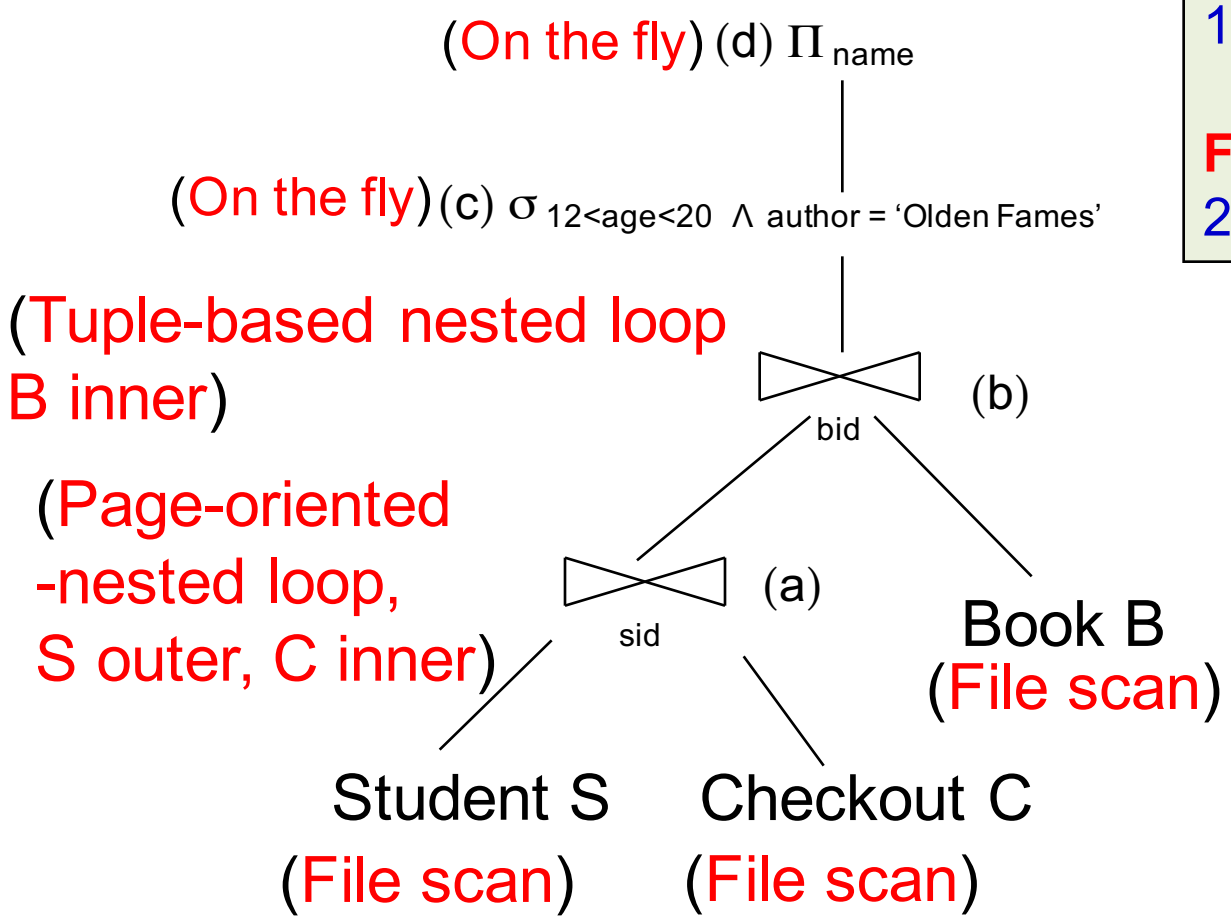
B(S)=1,000
B(B)=5,000
B(C)=15,000

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

(Total)

Total cost =
1,515,001,000

Final cardinality =
234 (approx)



S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

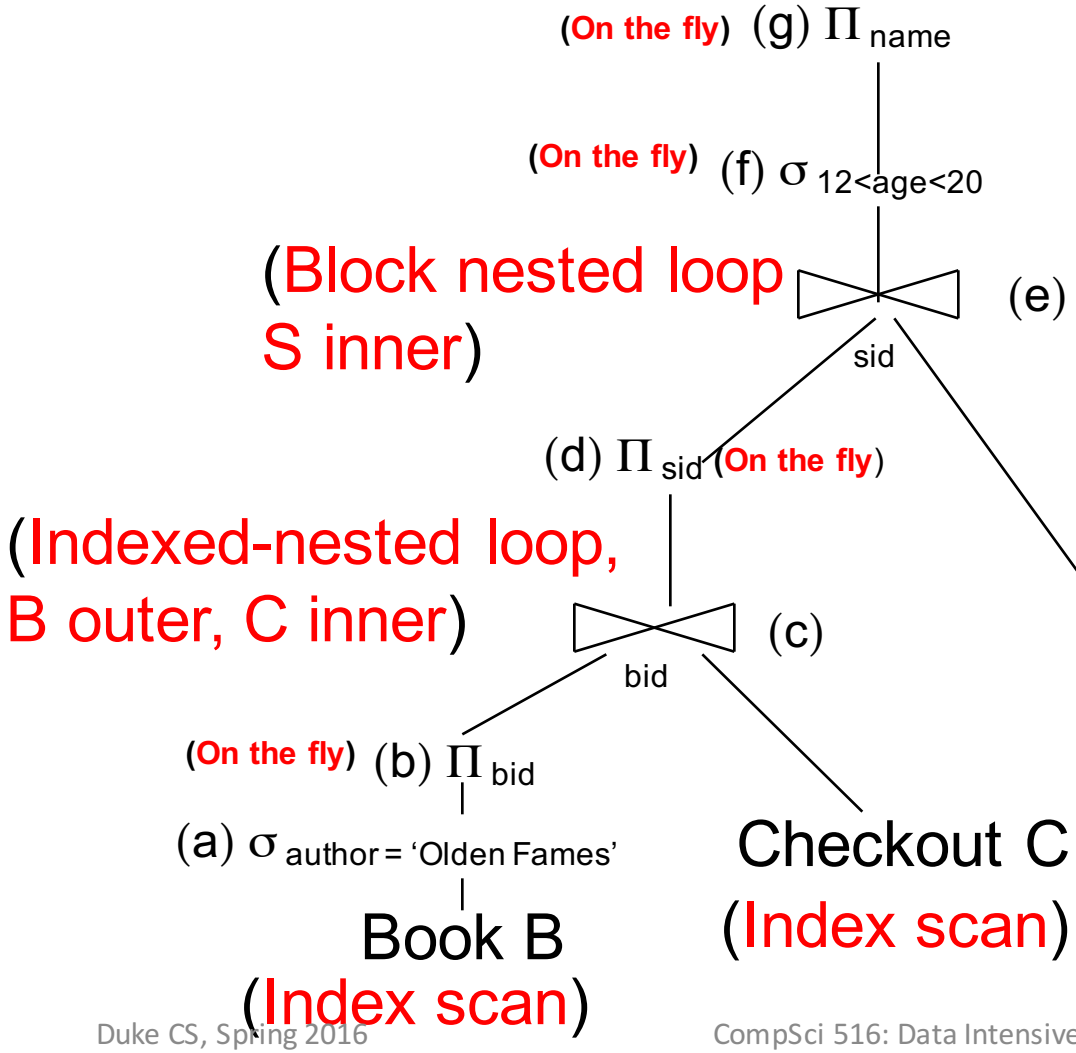
T(S)=10,000
T(B)=50,000
T(C)=300,000

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

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

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

Physical Query Plan – 2



Q. Compute

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

Assumptions:

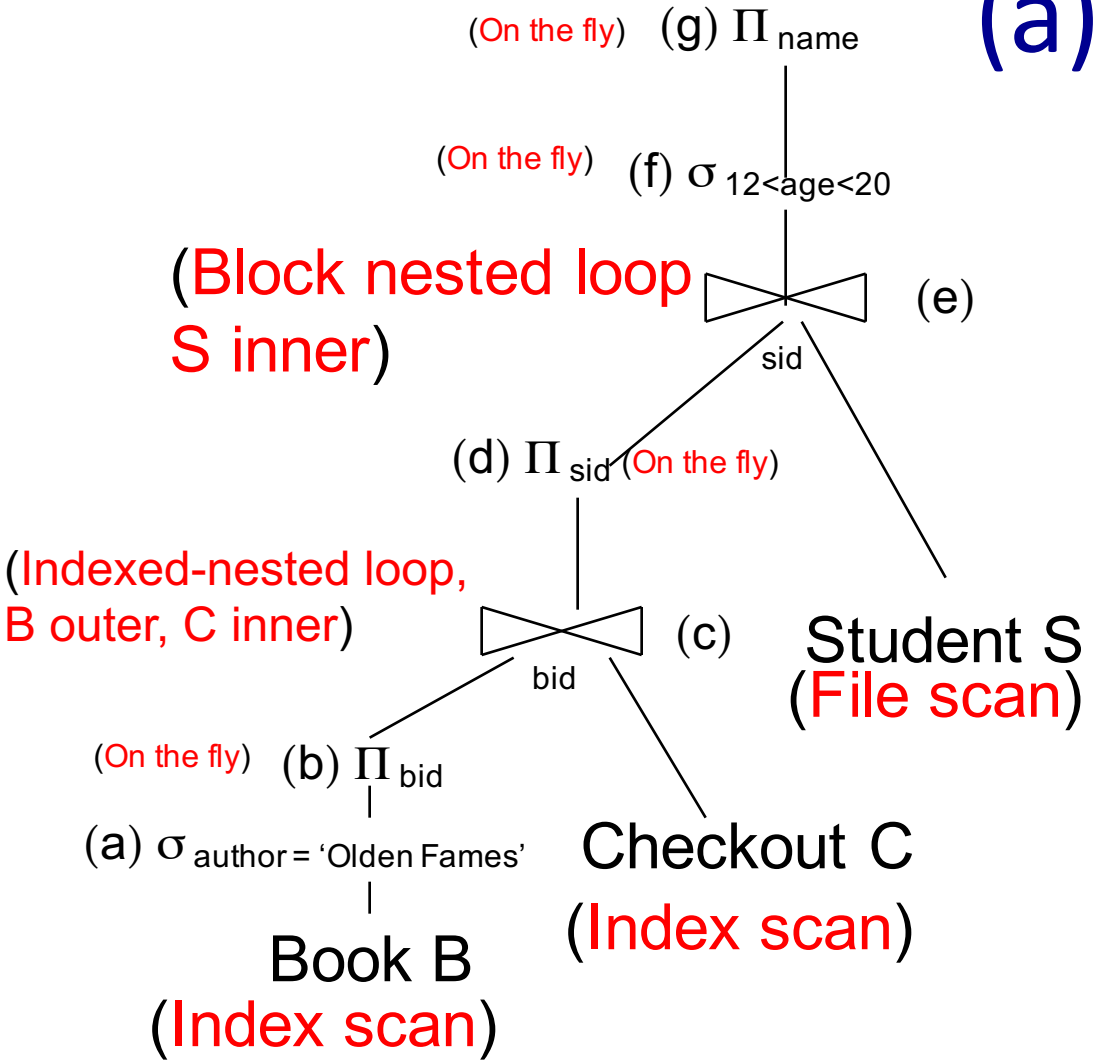
- Unclustered B+tree index on B.author
- Clustered B+tree index on C.bid
- All index pages are in memory
- Unlimited memory

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

(a)



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

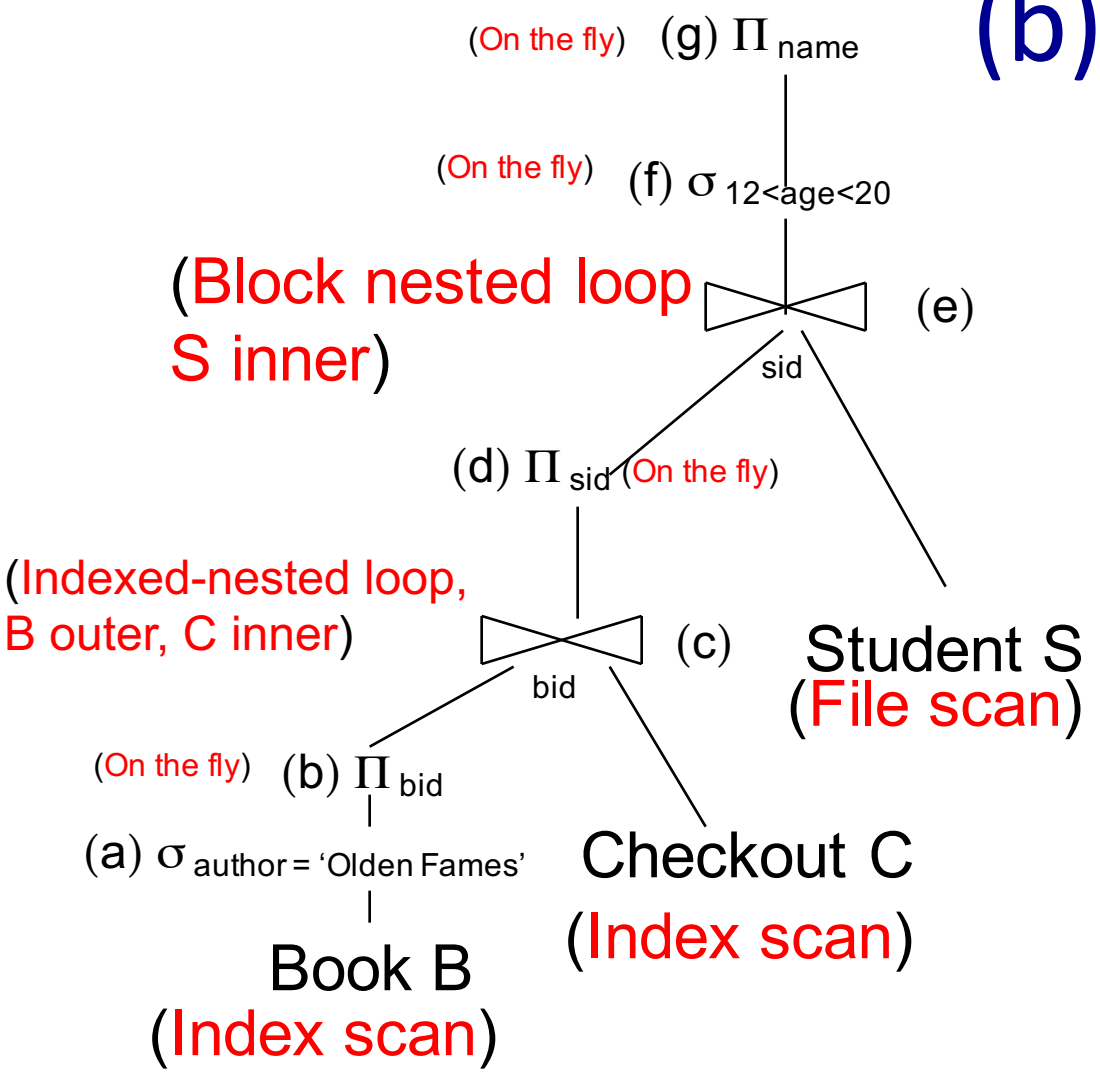
Cardinality =
100

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

(b)



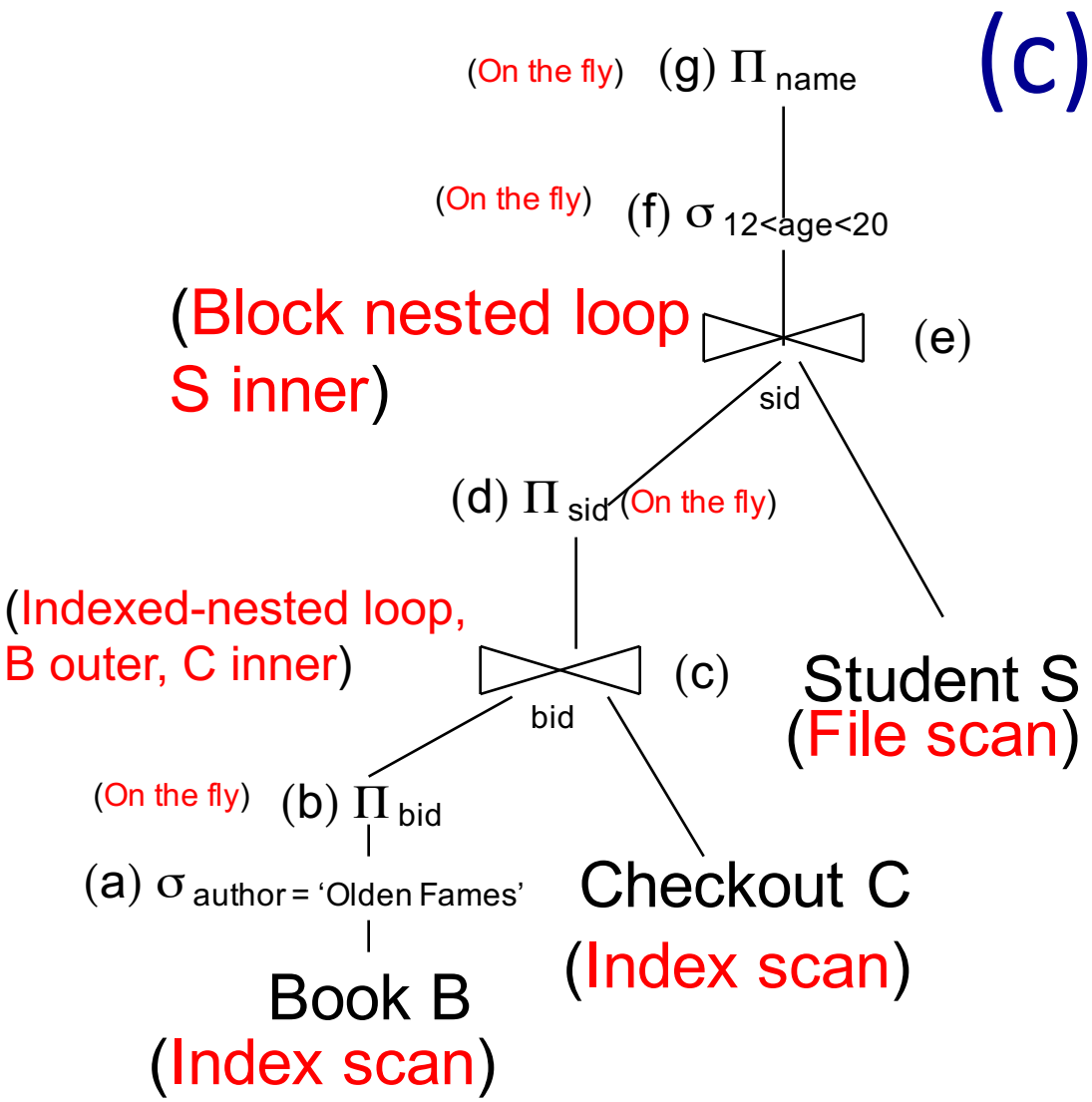
Cost =
0 (on the fly)

Cardinality =
100

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



(c)

- 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

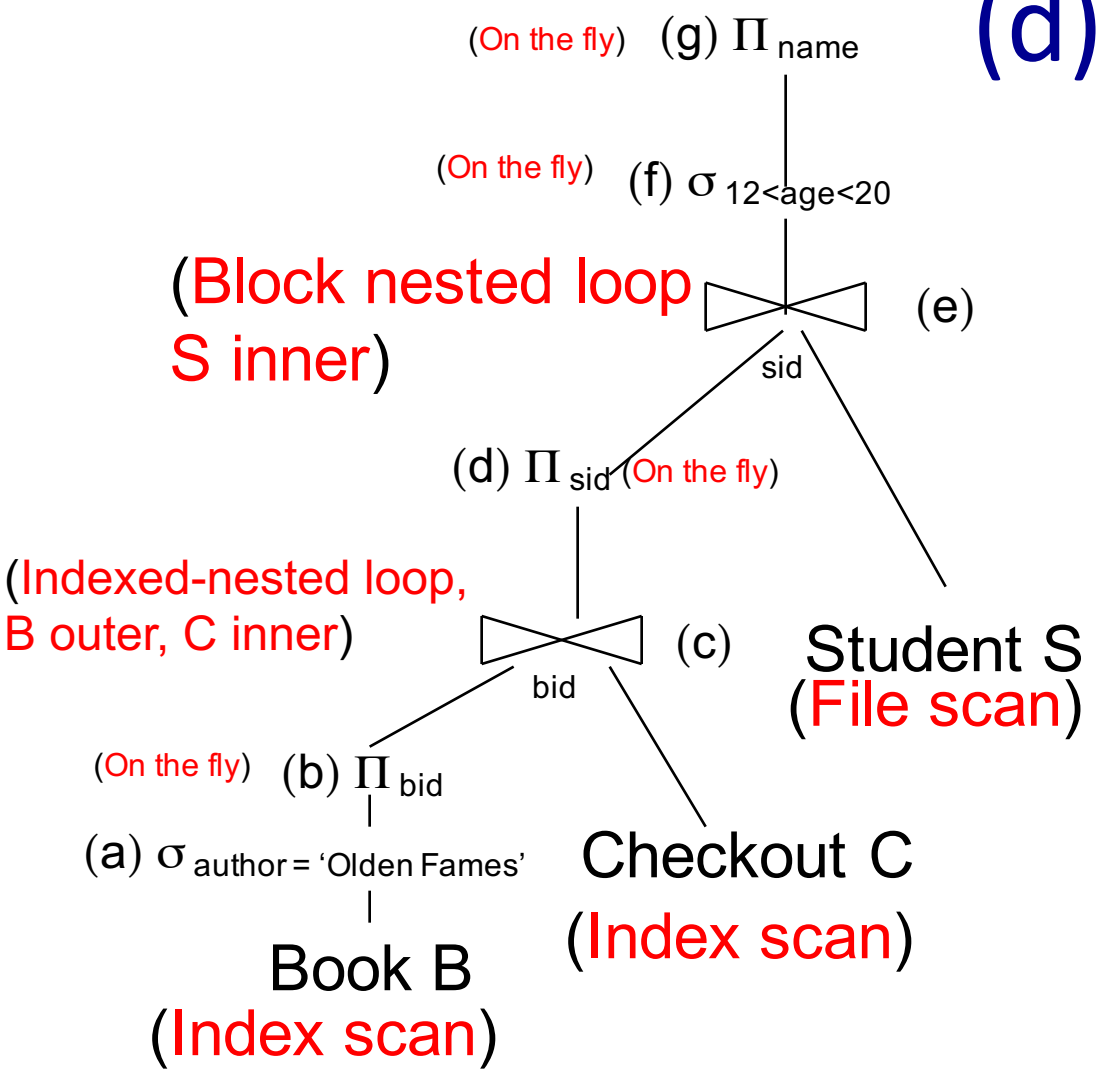
= 100 * T(C)/ MAX(100, V(C, bid))
assuming
V(C, bid) = V(B, bid) = T(B) = 50,000

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 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

(d)



Cost =
0 (on the fly)

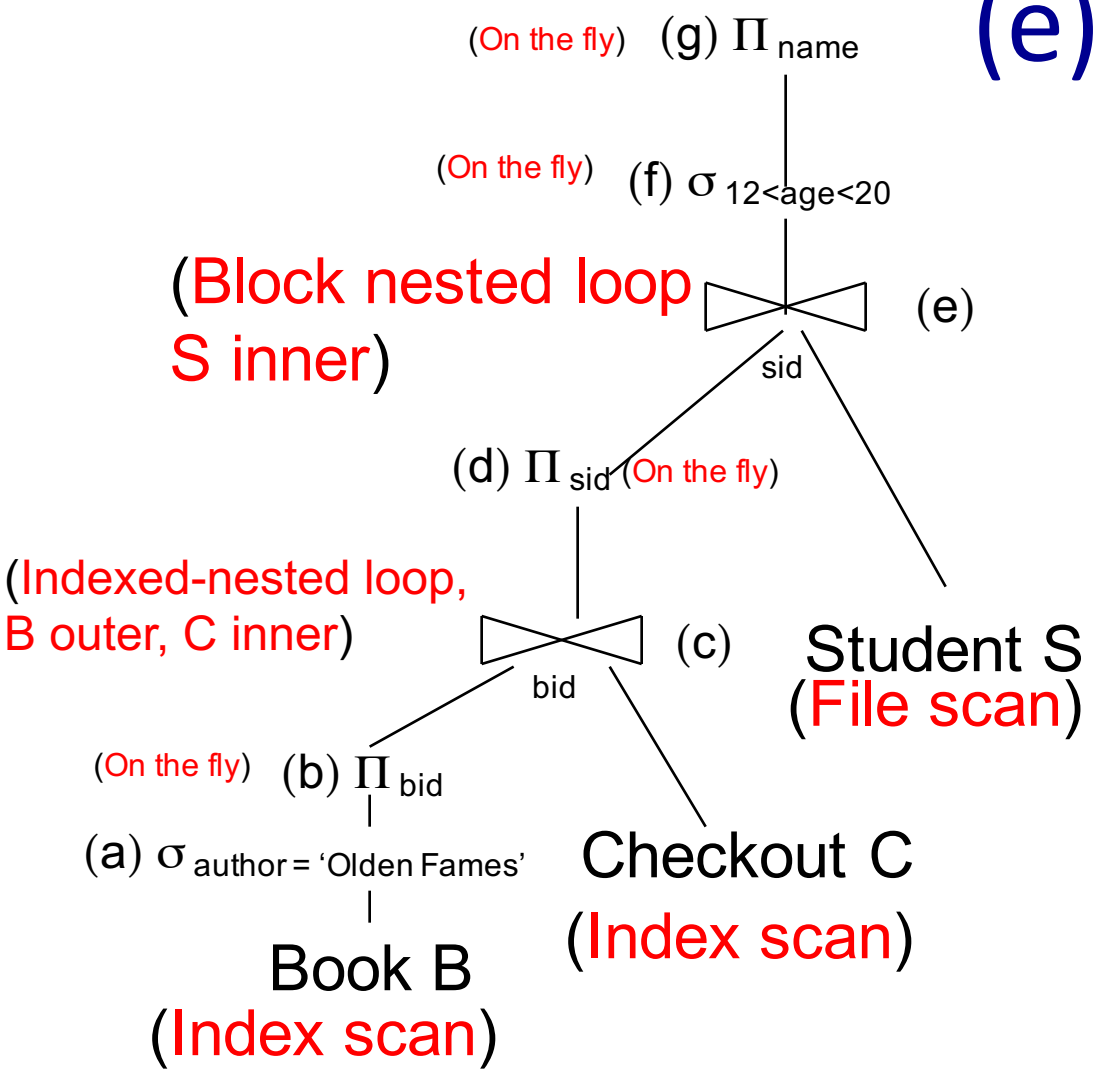
Cardinality =
600

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

(e)



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

Cost =
B(S) = 1000

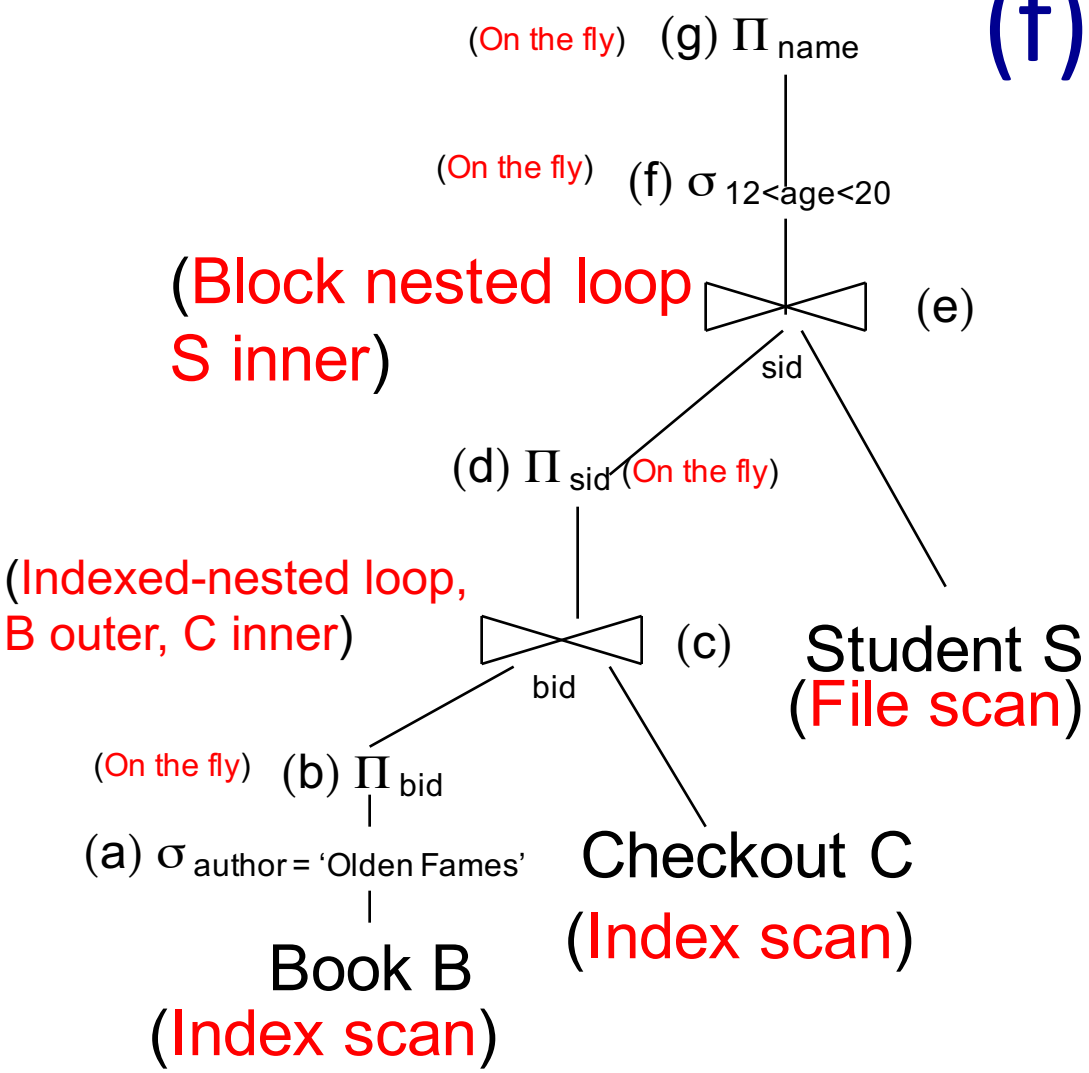
Cardinality =
600

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

(f)



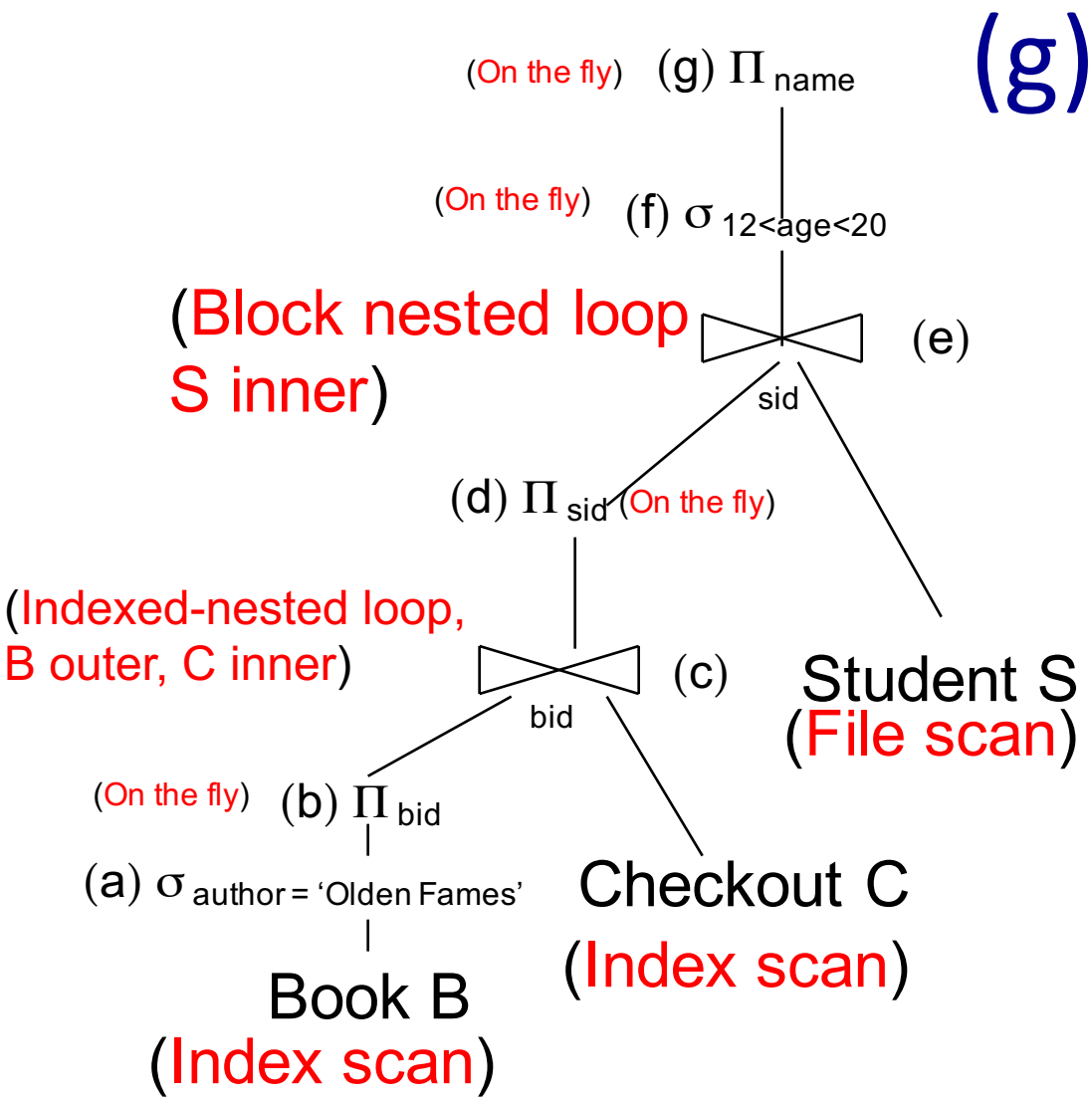
Cost =
0 (on the fly)

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

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



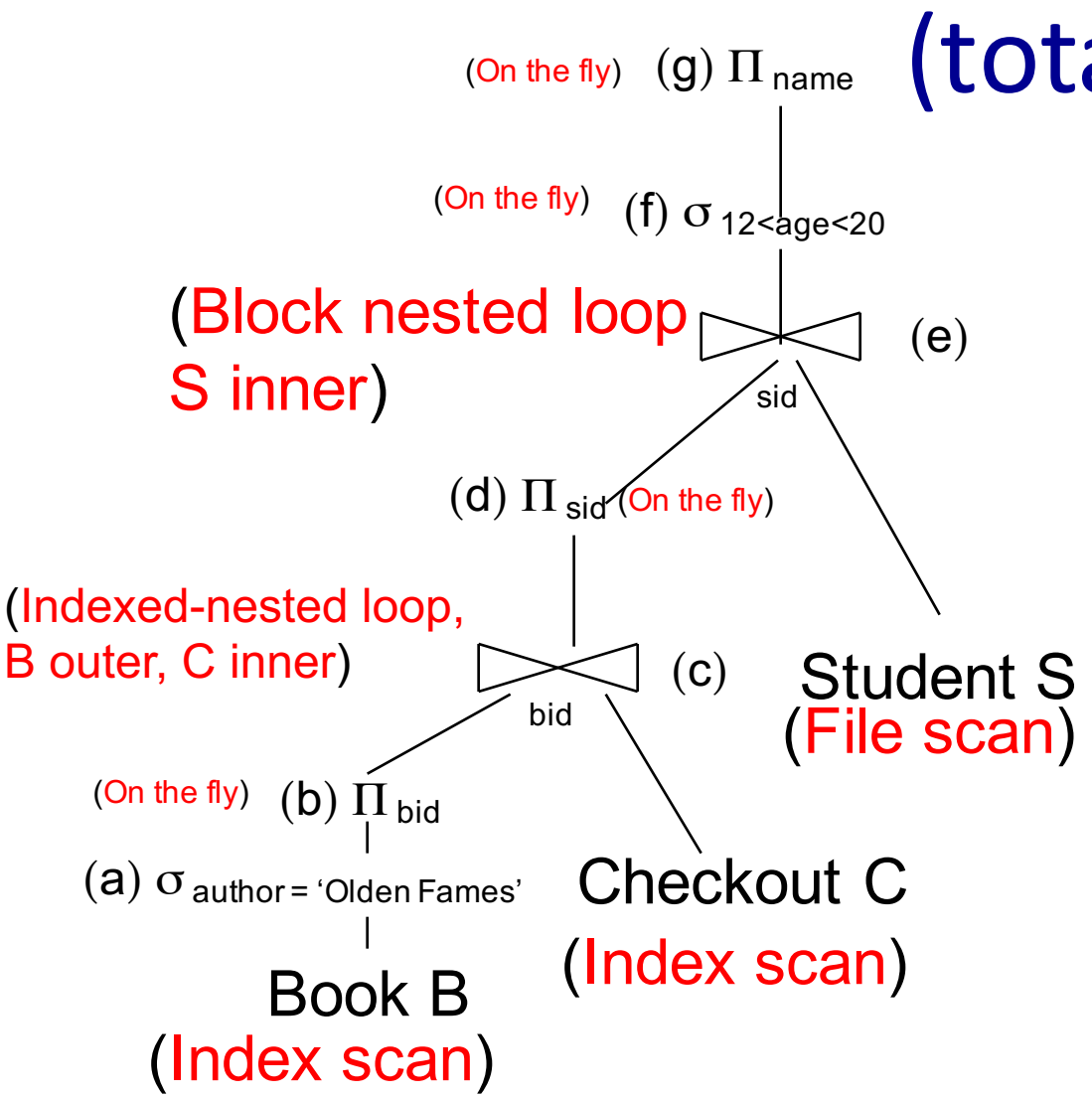
Cost =
0 (on the fly)

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 cost =
1300
(compare with 1,515,001,000 for plan 1!)

Final cardinality =
234 (approx)
(same as plan 1!)

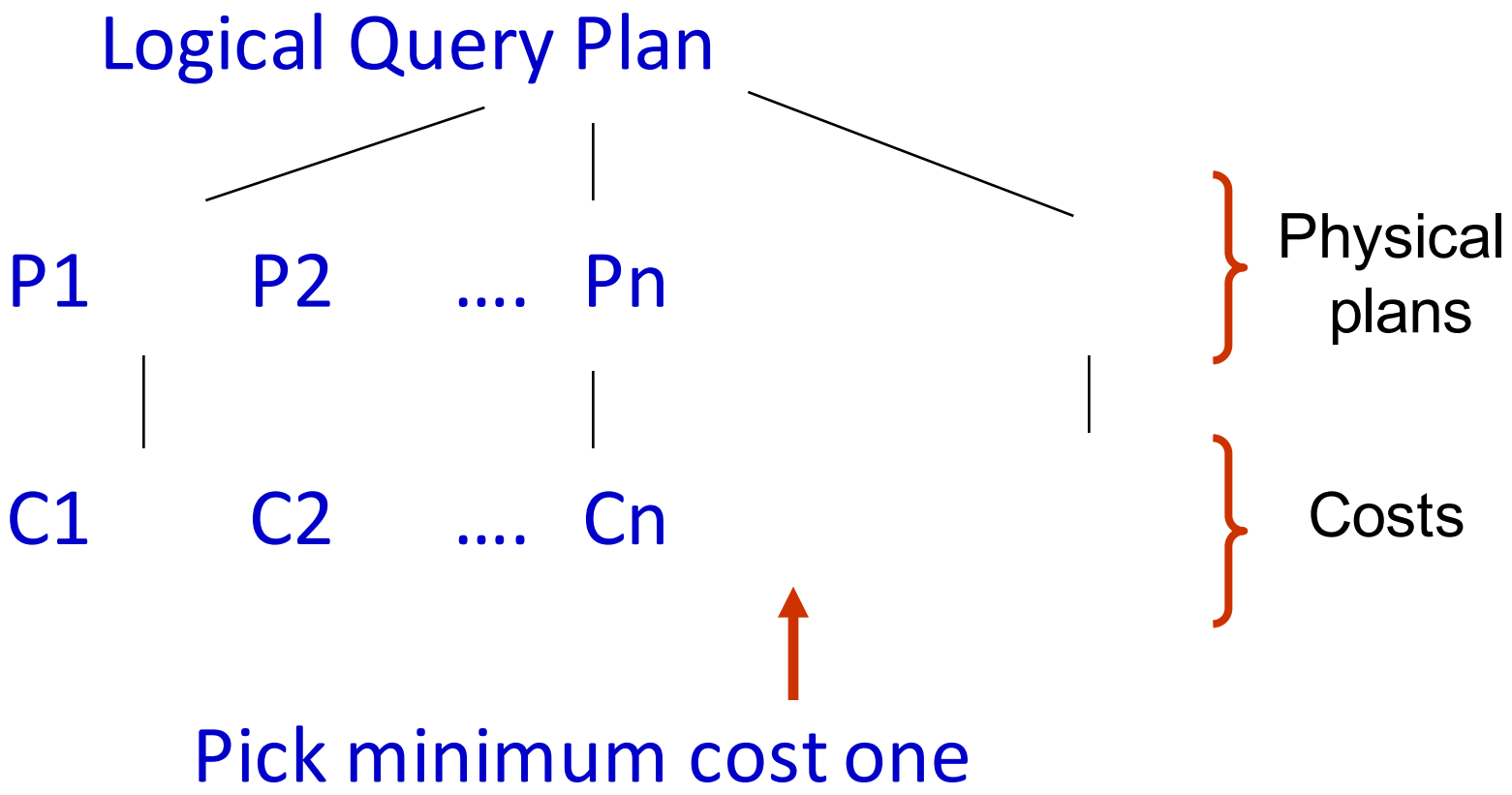
Task 4:
Efficiently searching the plan space

Use dynamic-programming based Selinger's algorithm

Heuristics for pruning plan space

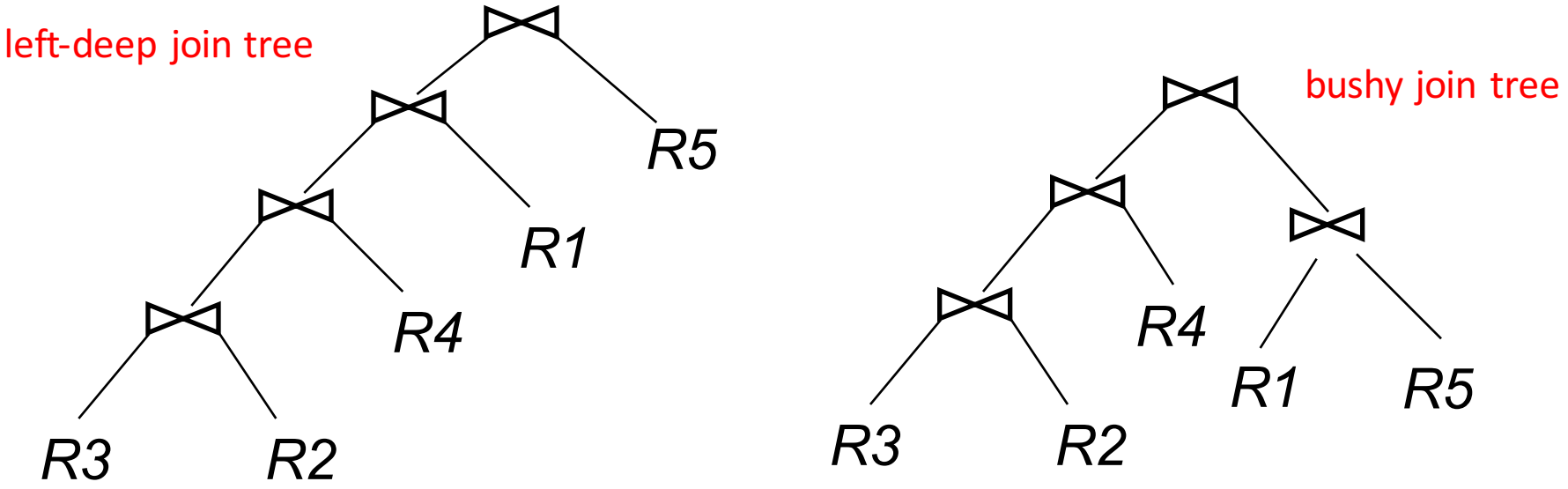
- Predicates as early as possible
- Avoid plans with cross products
- Only left-deep join trees

Physical Plan Selection



Join Trees

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



(logical plan space)

- Several possible structure of the trees
- Each tree can have n! permutations of relations

(physical plan space)

- Different implementation and scanning of intermediate operators for each logical plan

Selinger Algorithm

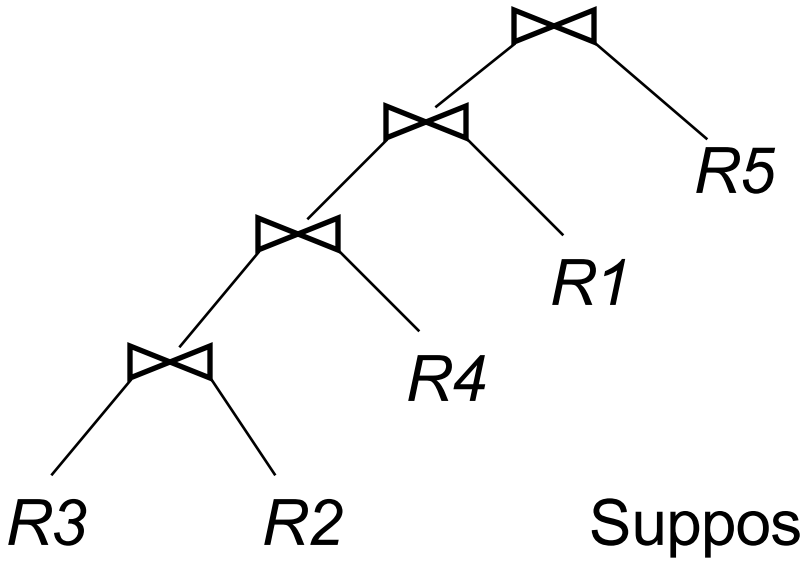
- Dynamic Programming based
- Dynamic Programming:
  - General algorithmic paradigm
  - Exploits "principle of optimality"
  - Useful reading:
  - Chapter 16, Introduction to Algorithms, Cormen, Leiserson, Rivest
- Considers the search space of left-deep join trees
  - reduces search space (only one structure), still n! permutations
  - interacts well with join algos (esp. NLJ)
  - e.g. might not need to write tuples to disk if enough memory

Principle of Optimality

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

Principle of Optimality

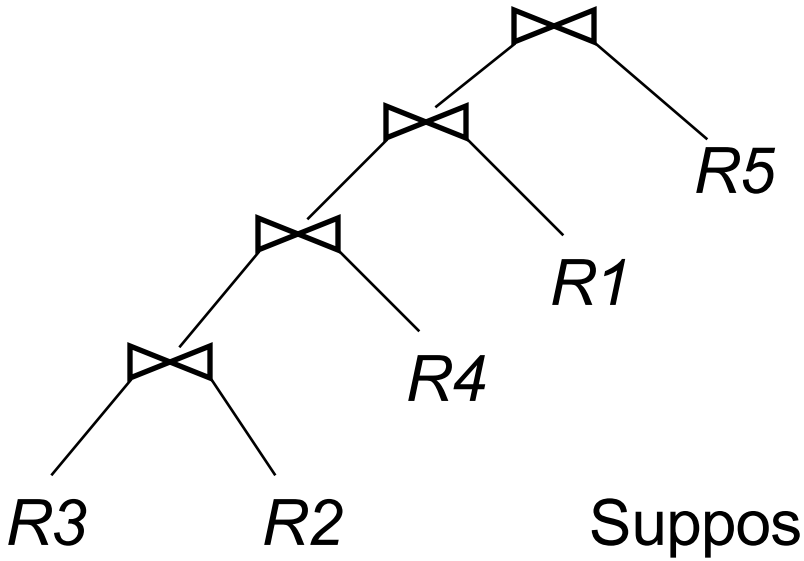
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4 \bowtie R5$



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

Principle of Optimality

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

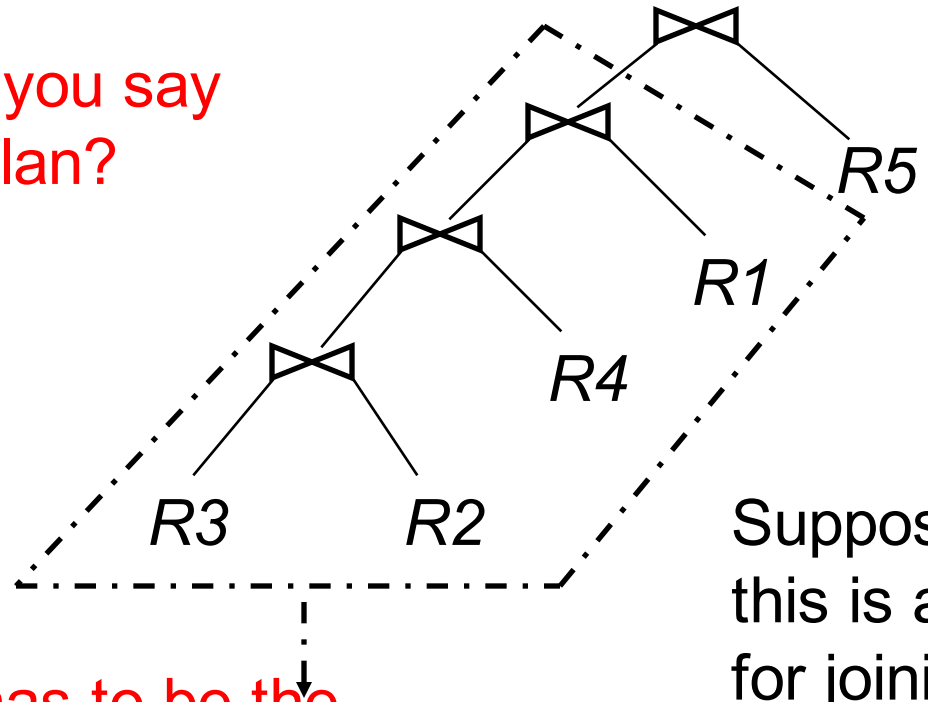


Suppose,
this is an Optimal Plan
for joining R1…R5:

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:

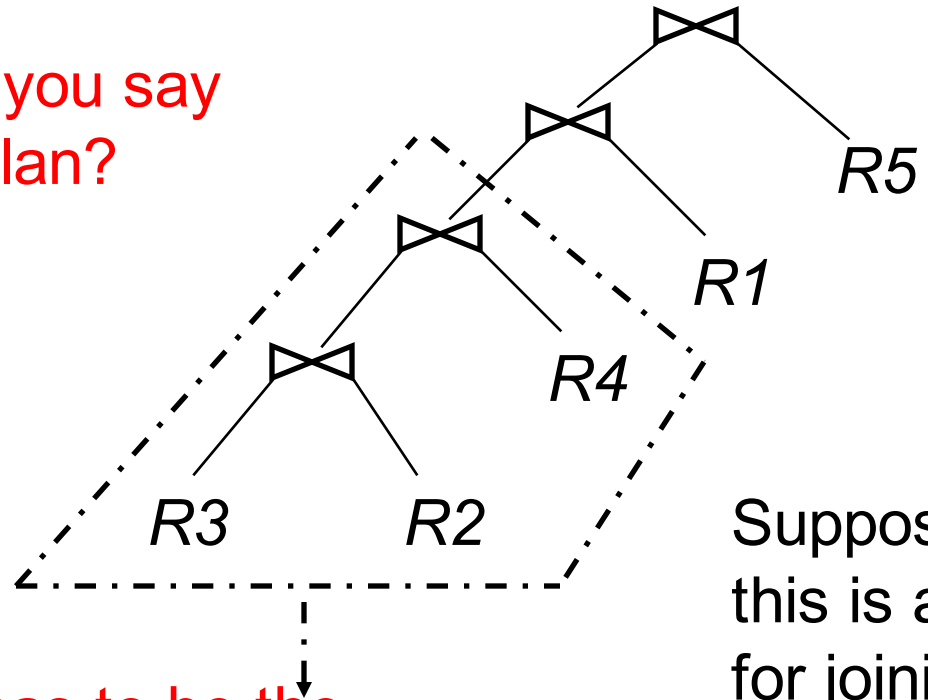
This has to be the
optimal plan for joining *R3, R2, R4, R1*

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



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

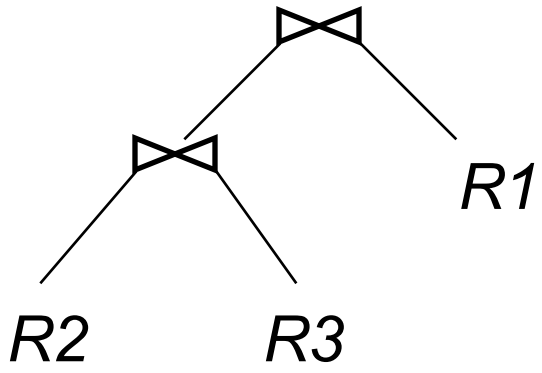
This has to be the optimal plan for joining *R3, R2, R4*

Exploiting Principle of Optimality

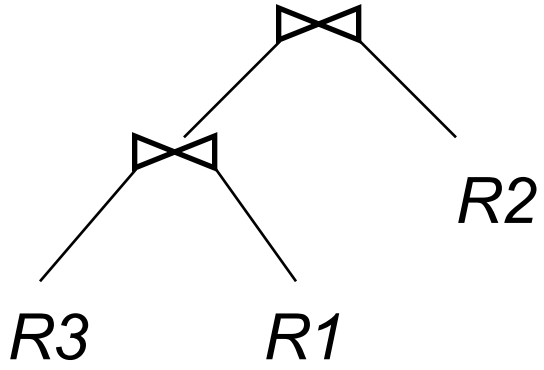
Query: $R1 \bowtie R2 \bowtie \ldots \bowtie Rn$

Both are giving the same result

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



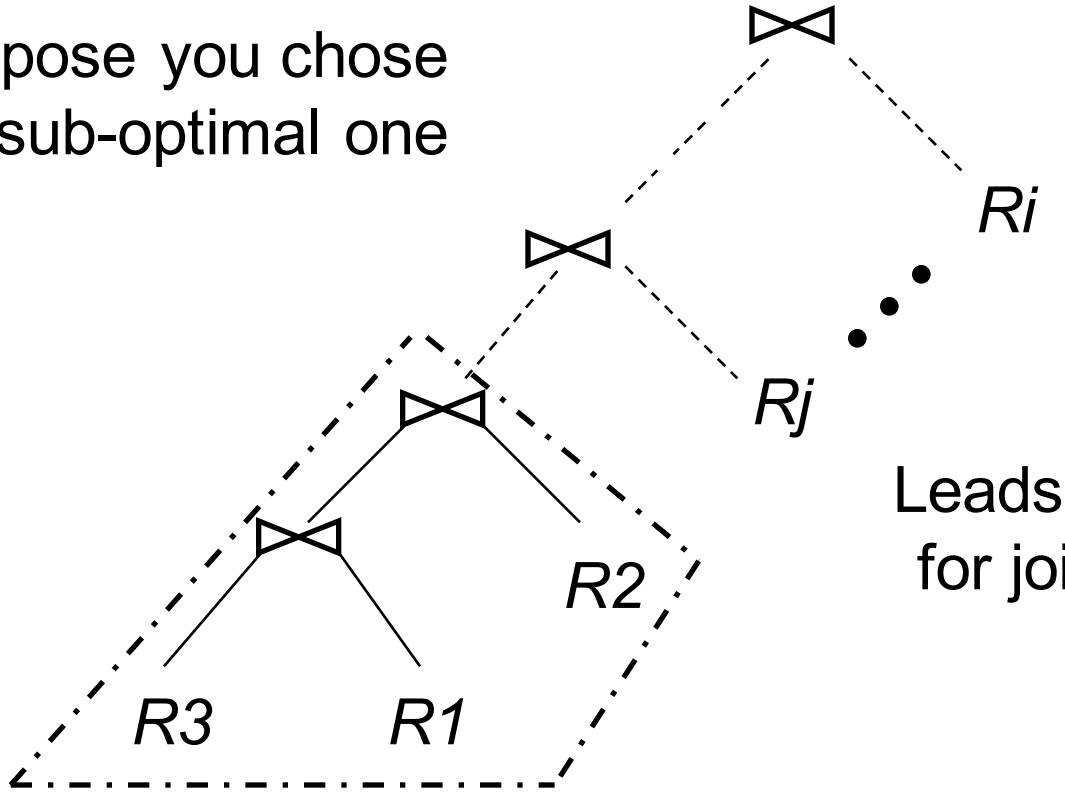
Optimal
for joining *R1, R2, R3*



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

Exploiting Principle of Optimality

Suppose you chose the sub-optimal one



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

A sub-optimal sub-plan cannot lead to an optimal plan

Notation

OPT ({ *R1, R2, R3* }):

Cost of optimal plan to join *R1,R2,R3*

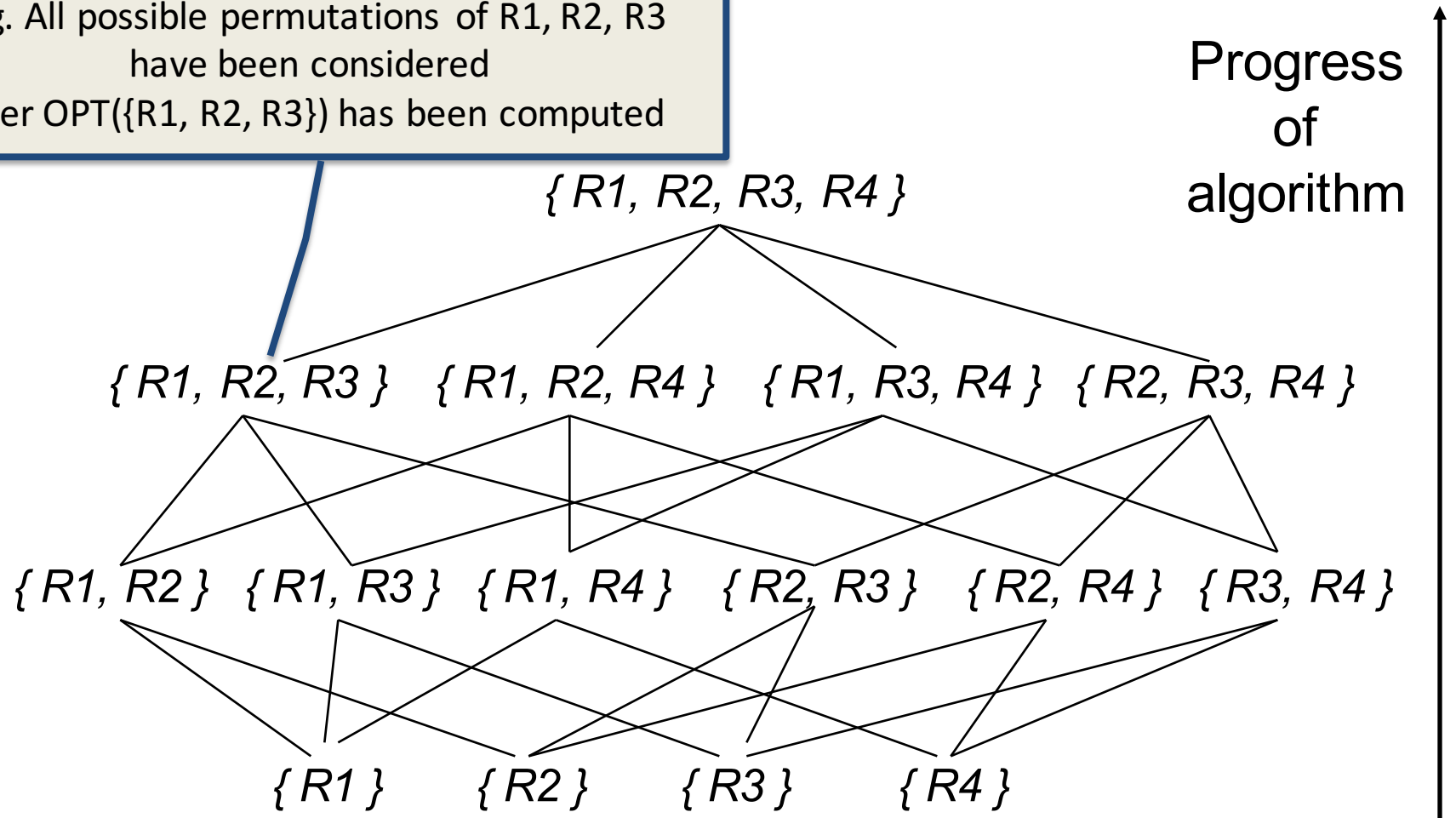
T ({ *R1, R2, R3* }):

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

Selinger Algorithm:

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

e.g. All possible permutations of R1, R2, R3 have been considered after OPT({R1, R2, R3}) has been computed



Progress of algorithm

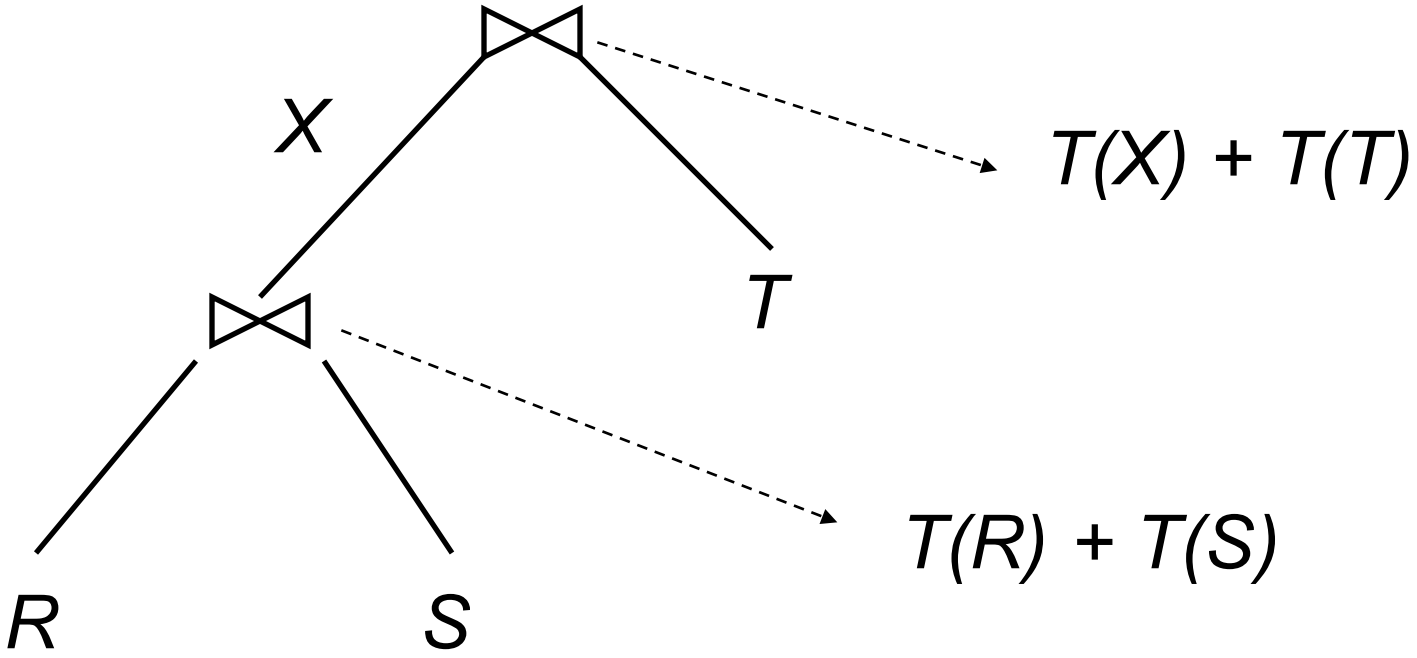
Simple Cost Model

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

All other operators have 0 cost

Note: 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:

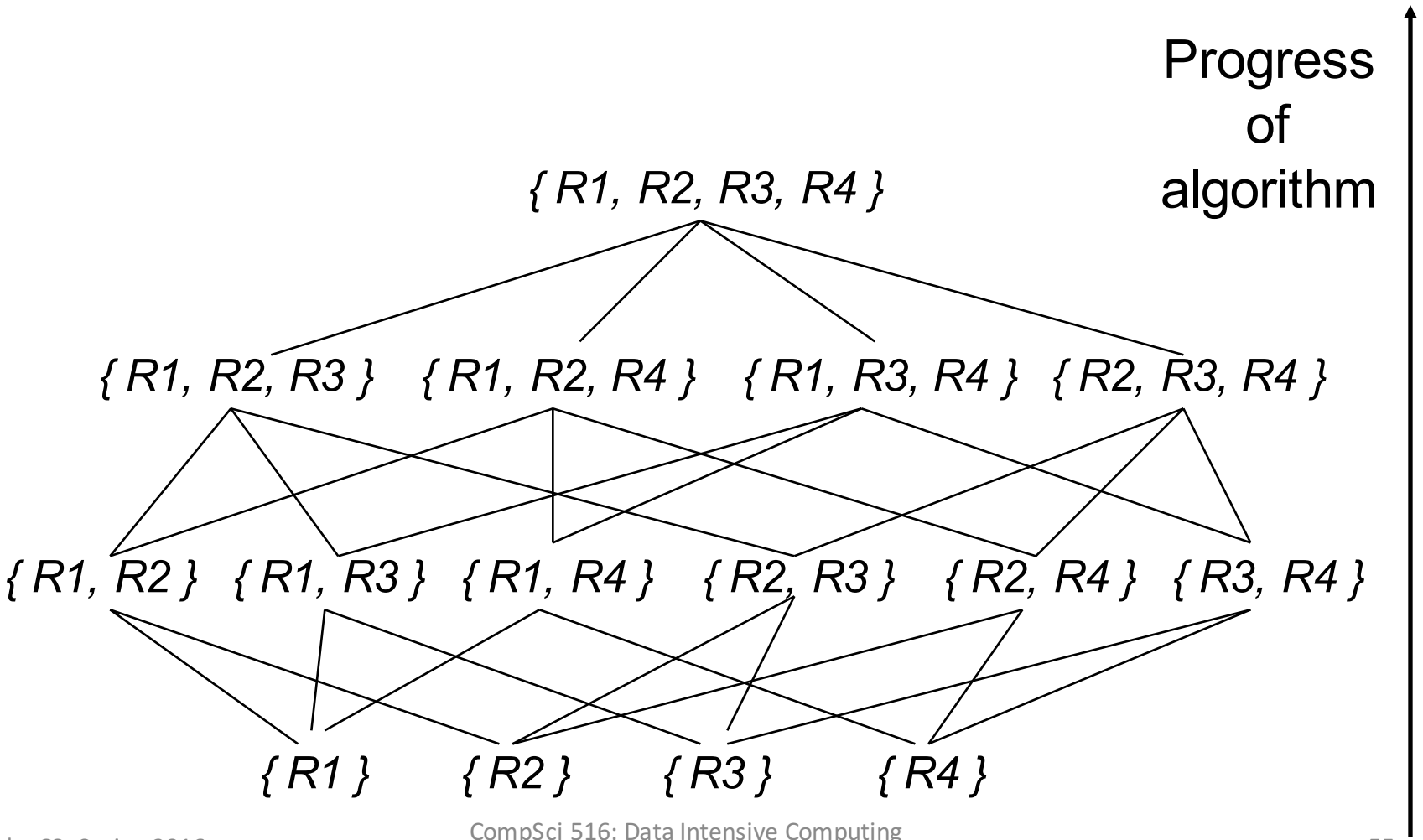
OPT ({ R1, R2, R3 }):

Min
- OPT ({ R1, R2 }) + T ({ R1, R2 }) + T(R3)
- OPT ({ R2, R3 }) + T ({ R2, R3 }) + T(R1)
- OPT ({ R1, R3 }) + T ({ R1, R3 }) + T(R2)

Note: Valid only for the simple cost model

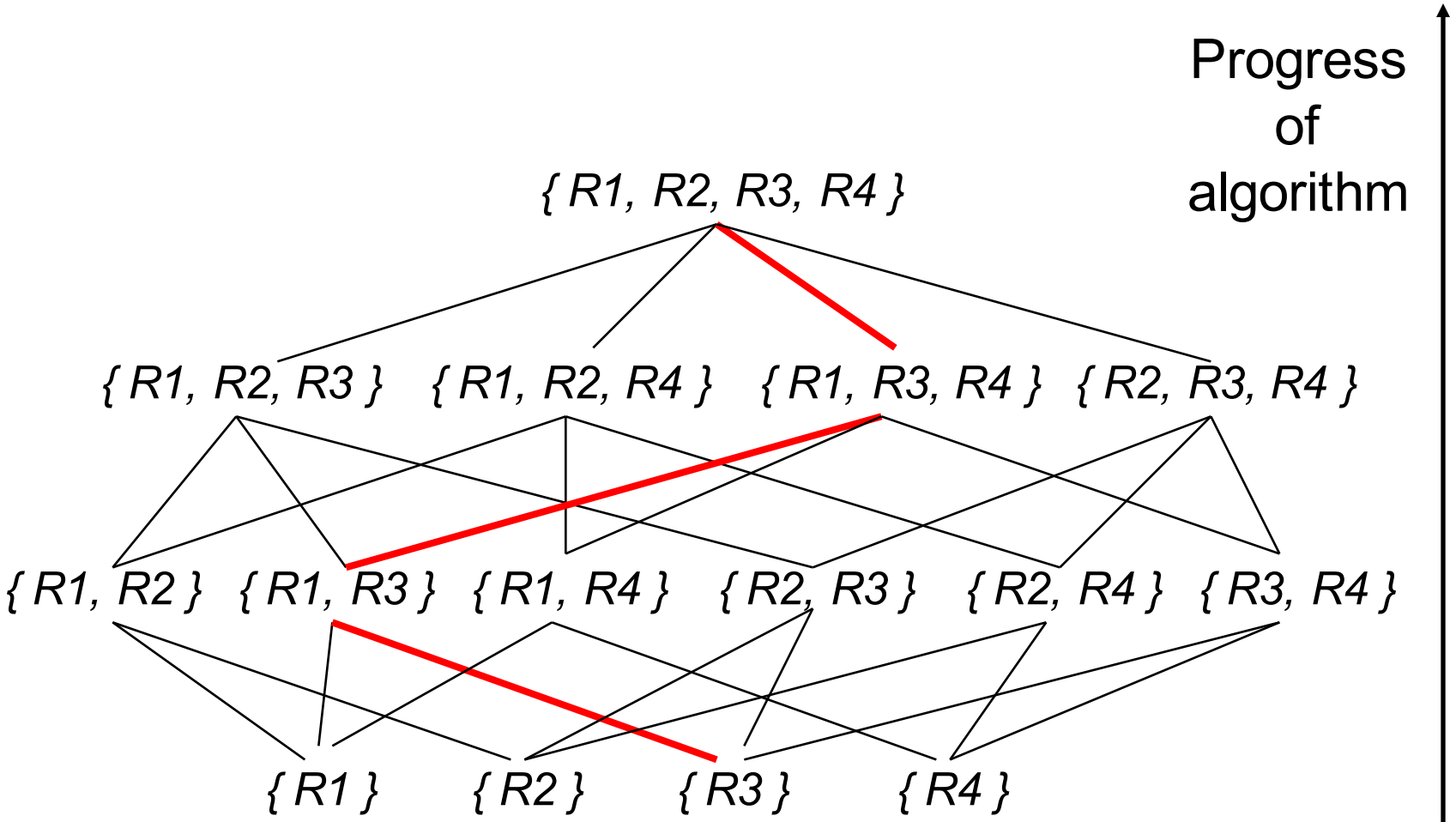
Selinger Algorithm:

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



Selinger Algorithm:

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

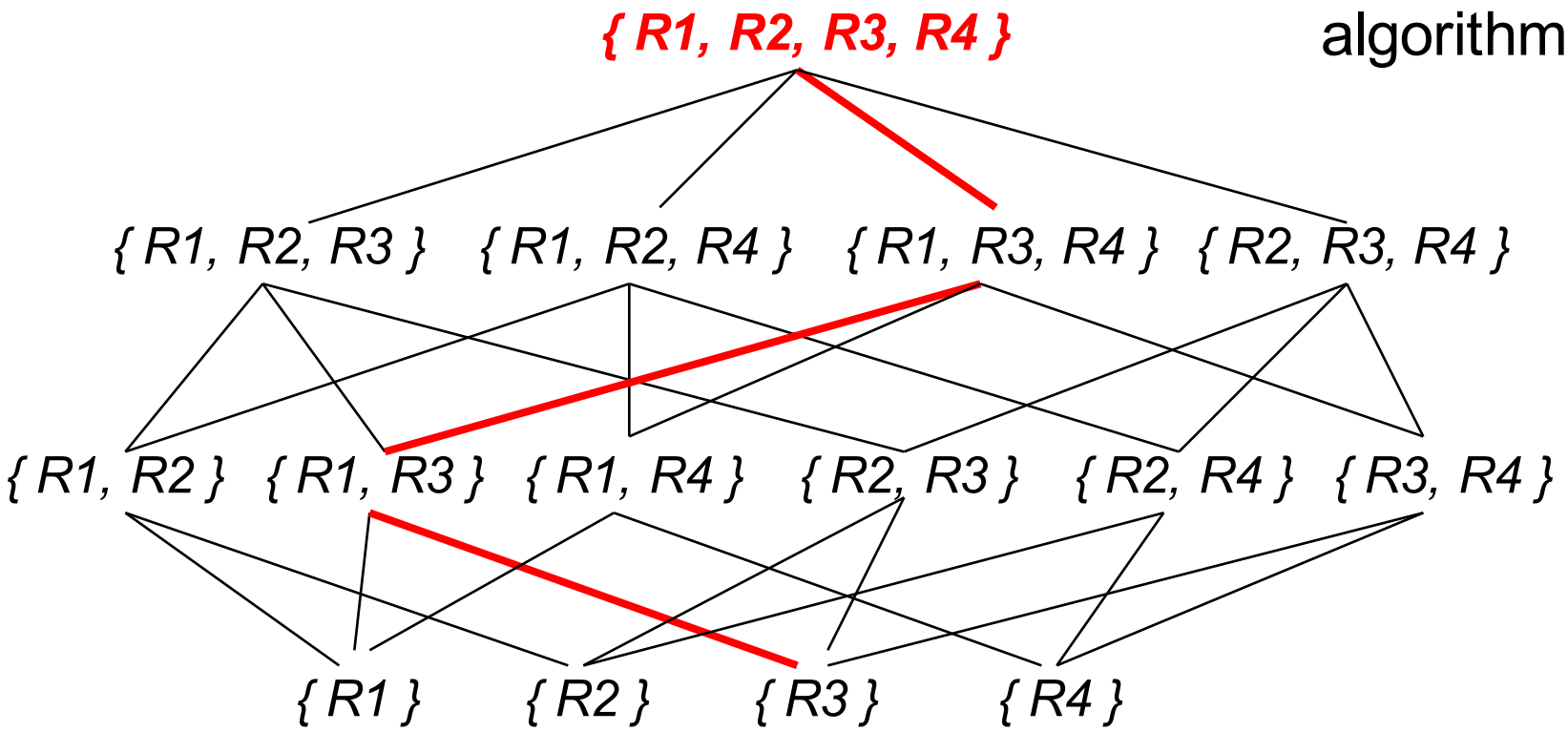


Selinger Algorithm:

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.



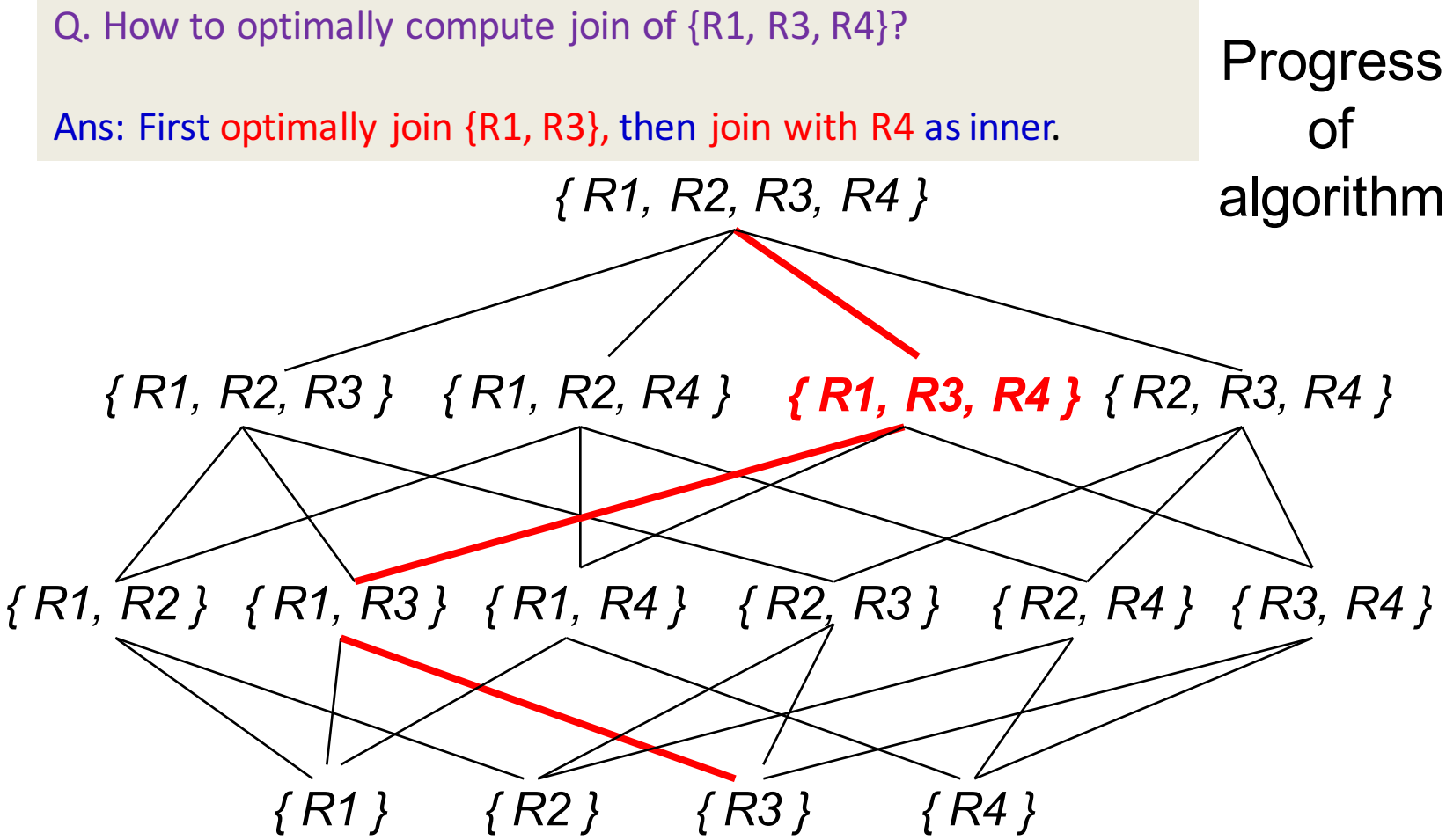
Progress of algorithm

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.



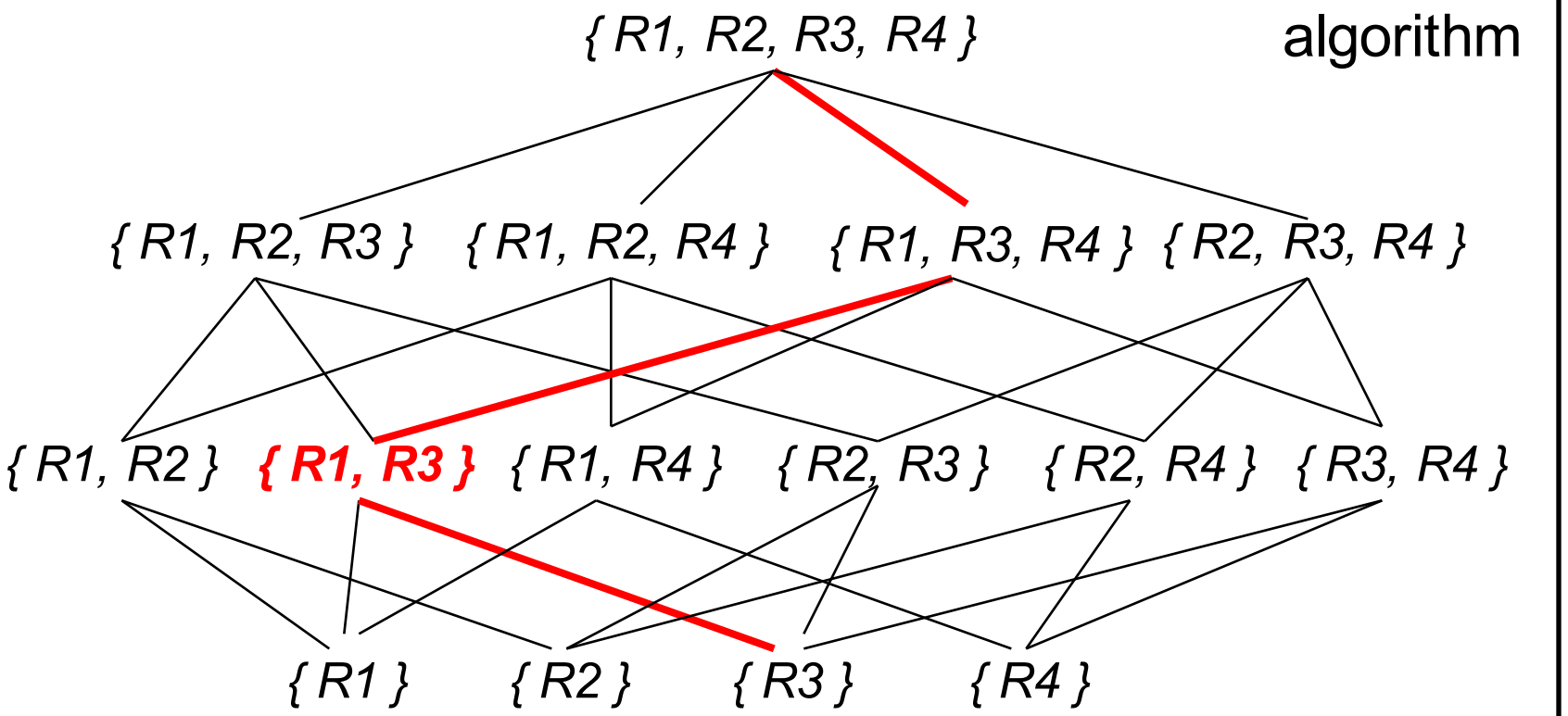
Progress of algorithm

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.



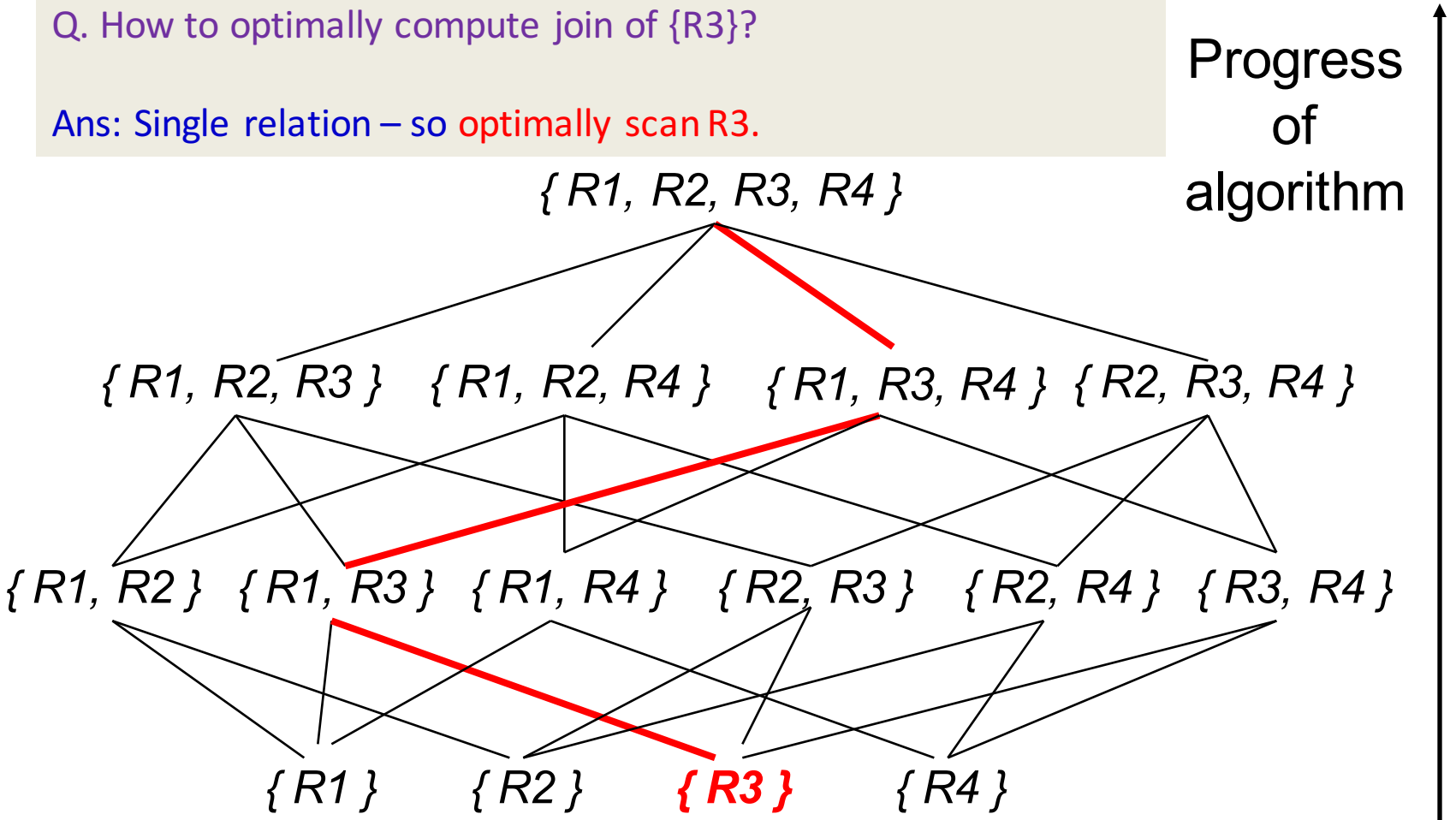
Progress of algorithm

Selinger Algorithm:

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

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

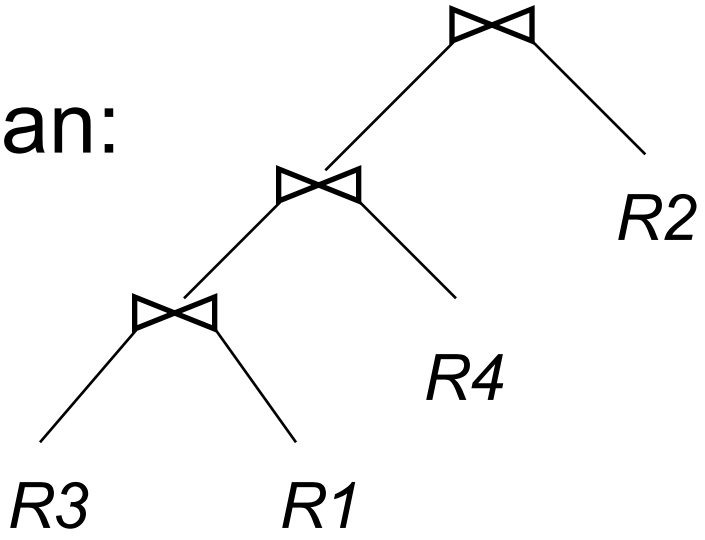
Ans: Single relation – so optimally scan R3.



Selinger Algorithm:

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

Final optimal plan:



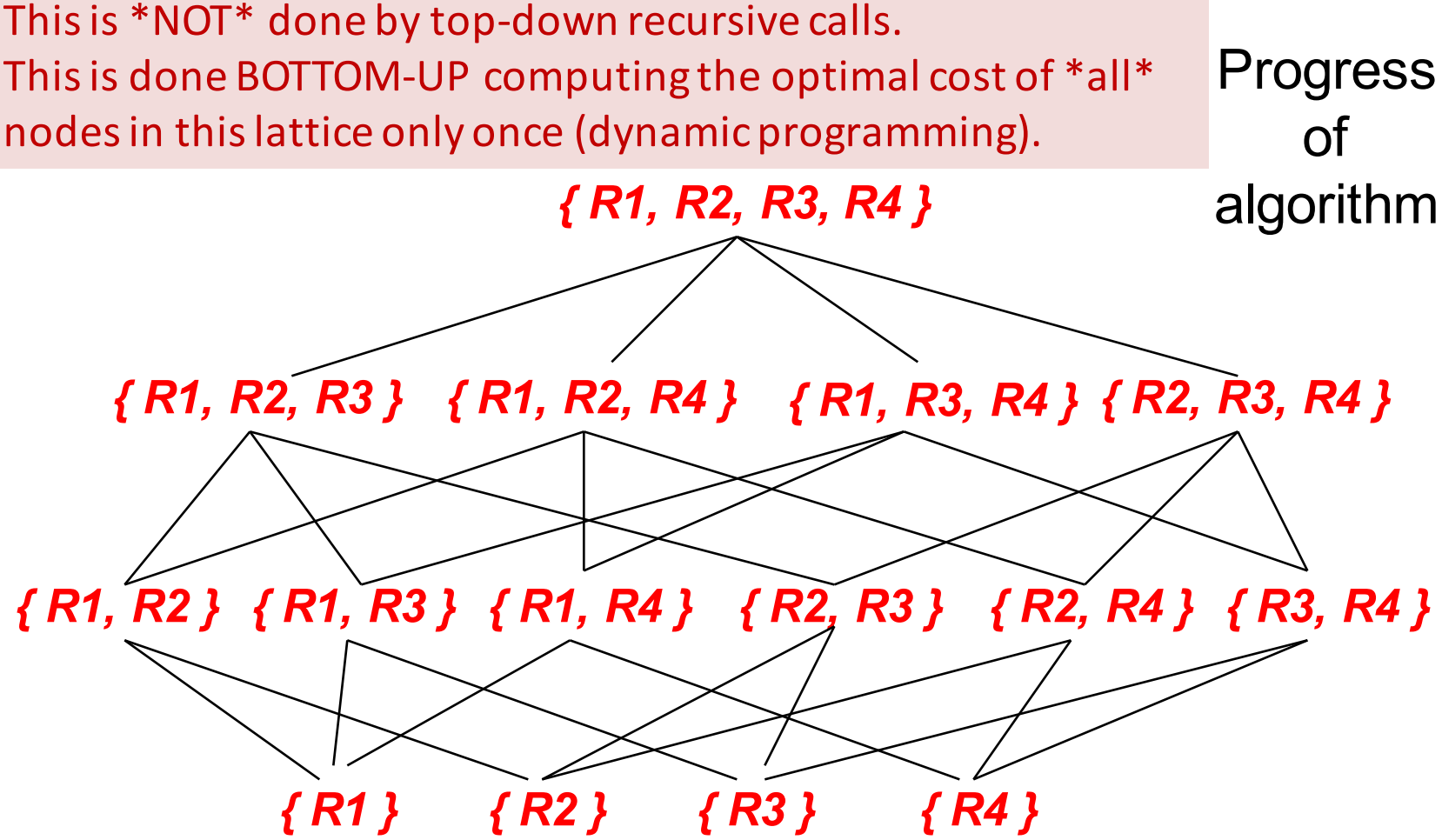
NOTE : There is a one-one correspondence between the permutation (R3, R1, R4, R2) and the above left deep plan

Selinger Algorithm:

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

NOTE: (*VERY IMPORTANT*)

- 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).



Full Example: Optimization with Selinger's

Sailors (sid, sname, srating, age)
Boats(bid, bname, color)
Reserves(sid, bid, date, rname)

Query:

```
SELECT S.sid, R.rname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid
AND B.bid = R.bid
AND B.color = red
```

See yourself how to include actual operator algorithms and scanning methods while running Selinger's

(Simple cost model is not useful in practice)

S (sid, sname, srating, age)
B (bid, bname, color)
R (sid, bid, date, rname)

Available Indexes

- Sailors: S, Boats: B, Reserves: R

- Sid, bid foreign key in R referencing S and B resp.
- Sailors
  - Unclustered B+ tree index on sid
  - Unclustered hash index on sid
- Boats
  - Unclustered B+ tree index on color
  - Unclustered hash index on color
- Reserves
  - Unclustered B+ tree on sid
  - Clustered B+ tree on bid

S (sid, sname, srating, age): B+tree - sid, hash index - sid
B (bid, bname, color) : B+tree - color, hash index - color
R (sid, bid, date, rname) : B+tree - sid, **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

First Pass

- Where to start?
  - How to access each relation, assuming it would be the first relation being read
  - File scan is also available!
- Sailors?
  - No selection matching an index, use File Scan (no overhead)
- Reserves?
  - Same as Sailors
- Boats?
  - Hash index on color, matches B.color = red
  - B+ tree also matches the predicate, but hash index is cheaper
    - B+ tree would be cheaper for range queries

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Second Pass

- What next?
  - For each of the plan in Pass 1 taken as outer, consider joining another relation as inner
- What are the combinations? How many new options?

Outer	Inner	OPTION 1	OPTION 2	OPTION 3
R (file scan)	**B**	(B+-color)	(hash color)	(File scan)
R (file scan)	**S**	(B+-sid)	(hash sid)	,,
S (file scan)	**B**	(B+-color)	(hash color)	,,
S (file scan)	**R**	(B+-sid)	(Cl. B+ bid)	,,
B (hash index)	**R**	(B+-sid)	(Cl. B+ bid	,,
B (hash index)	**S**	(B+-sid)	(hash sid)	,,

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Second Pass

- Which outer-inner combinations can be discarded?
  - B, S and S, B: Cartesian product!

Outer	Inner	OPTION 1	OPTION 2	OPTION 3
R (file scan)	**B**	(B+-color)	(hash color)	(File scan)
R (file scan)	**S**	(B+-sid)	(hash sid)	,,
S (file scan)	**B**	(B+-color)	(hash color)	,,
S (file scan)	**R**	(B+-sid)	(Cl. B+ bid)	,,
B (hash index)	**S**	(B+-sid)	(hash sid)	,,
B (hash index)	**R**	(B+-sid)	(Cl. B+ bid):	,,

OPTION 3 is not shown on next slide, expected to be more expensive

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Outer	Inner	OPTION 1	OPTION 2
R (file scan)	**S**	**(B+-sid)** Slower than hash-index (need Sailor tuples matching S.sid = value, where value comes from an outer R tuple)	**(hash sid):** likely to be faster **2A. Index nested loop join** **2B Sort Merge based join:** (no index is sorted on sid, need to sort, output sorted by sid, retained if cheaper)
R (file scan)	**B**	**(B+-color)** Not useful	**(hash color)** Consider all methods, select those tuples where B.color = red using the color index (note: no index on bid)
S (file scan)	**R**	**(B+-sid)** Consider all methods	**(Cl. B+ bid)** Not useful
B (hash index)	**R**	**(B+-sid)** Not useful	**(Cl. B+ bid)** **2A. Index nested loop join** (no H. I. on bid) **2B. Sort-merge join** (clustered, index sorted on bid, produces outputs in sorted order by bid, retained if cheaper)

Keep the least cost plan between
- (R, S) and (S, R)
- (R, B) and (B, R)

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Third Pass

- Join with the third relation
- For each option retained in Pass 2, join with the third relation
- E.g.
  - Boats (B+tree on color) – sort-merged-join – Reserves (B+tree on bid)
  - Join the result with Sailors (B+ tree on sid) using sort-merge-join
    - Need to sort (B join R) by sid, was sorted on bid before
    - Outputs tuples sorted by sid
    - Not useful here, but will be useful if we had GROUP BY on sid
    - In general, a higher cost "**interesting**" plans may be retained (e.g. sort operator at root, grouping attribute in group by query later, join attriute in a later join)