CS-300: Data-Intensive Systems

Query Processing with Relational Operations

(Chapters 16)

Prof. Anastasia Ailamaki, Prof. Sanidhya Kashyap

EPFL

Today's focus

- Overview
- Query transformation
- Cost estimation
- Plan enumeration and costing
- System R strategy

What we already know …

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

For each SQL query….

```
SELECT S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle'
  AND S.sstate='WA'
  AND S.sno=U.sno
  AND U.pno=2
```

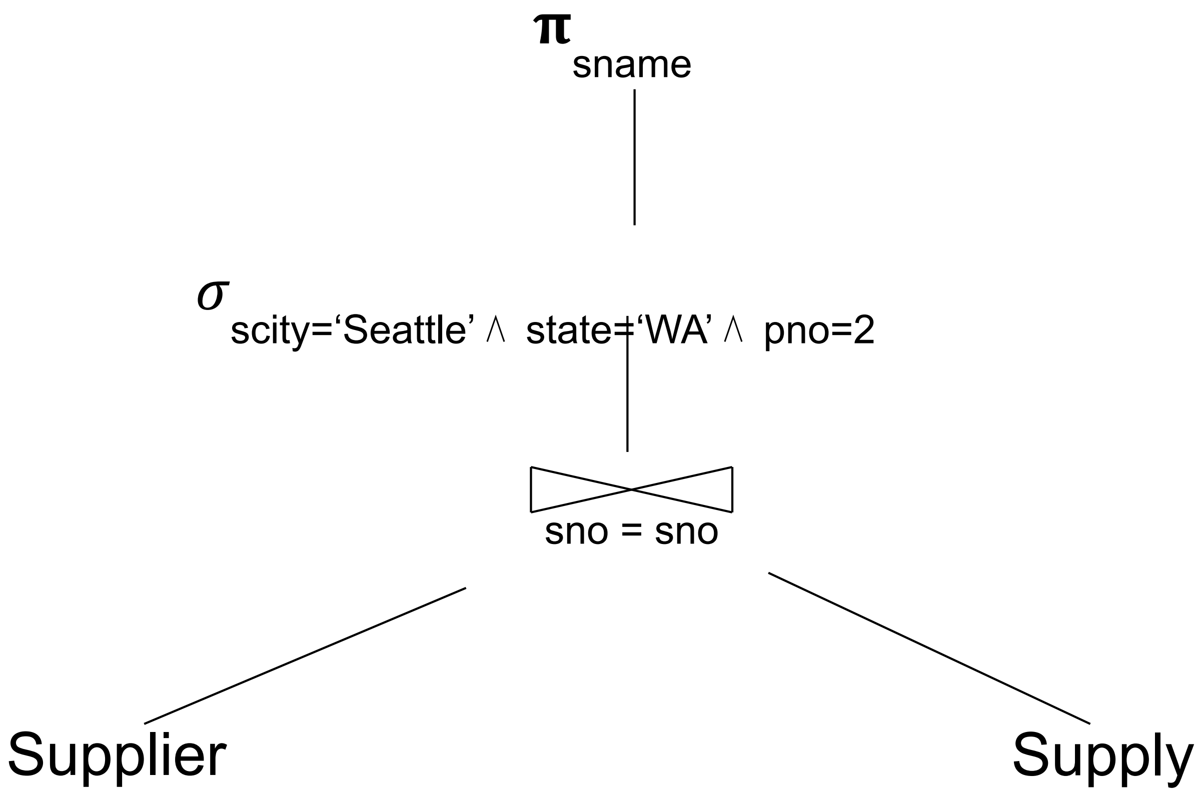
There exist many logical query plans…

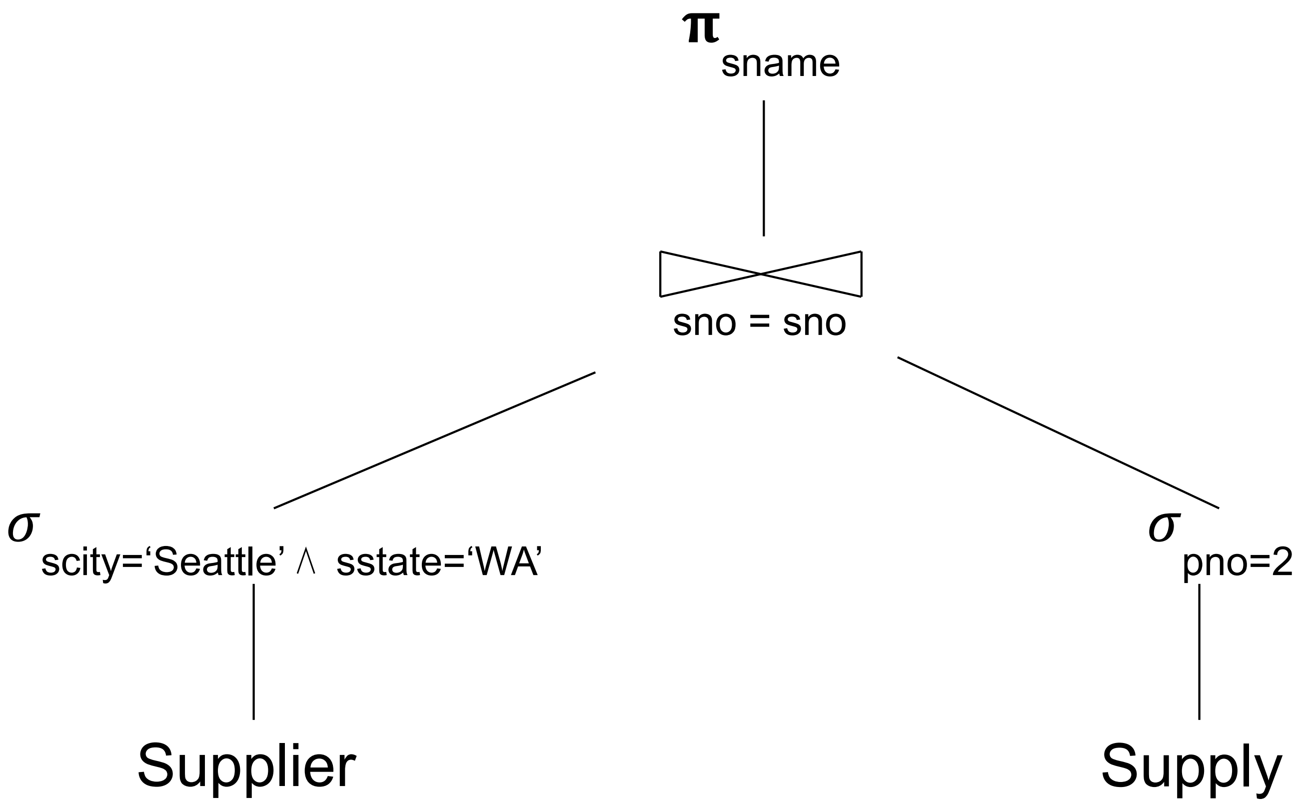
Question: Logical plan equivalence

Why can two **logically equivalent queries** can have **varying runtimes**?

Example query: Logical plans

```
SELECT S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle'
  AND S.sstate='WA'
  AND S.sno=U.sno
  AND U.pno=2
```



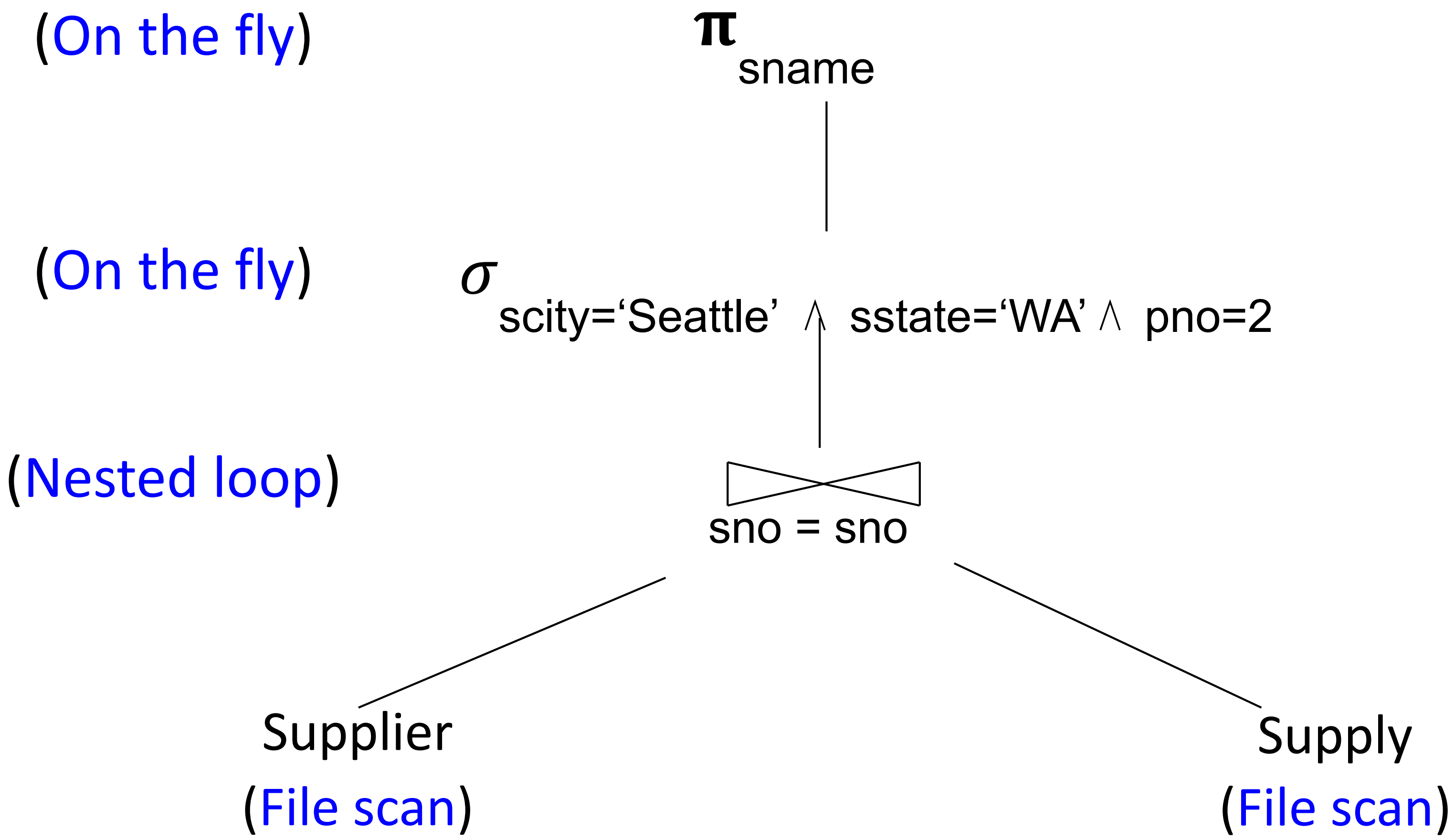


What we also know …

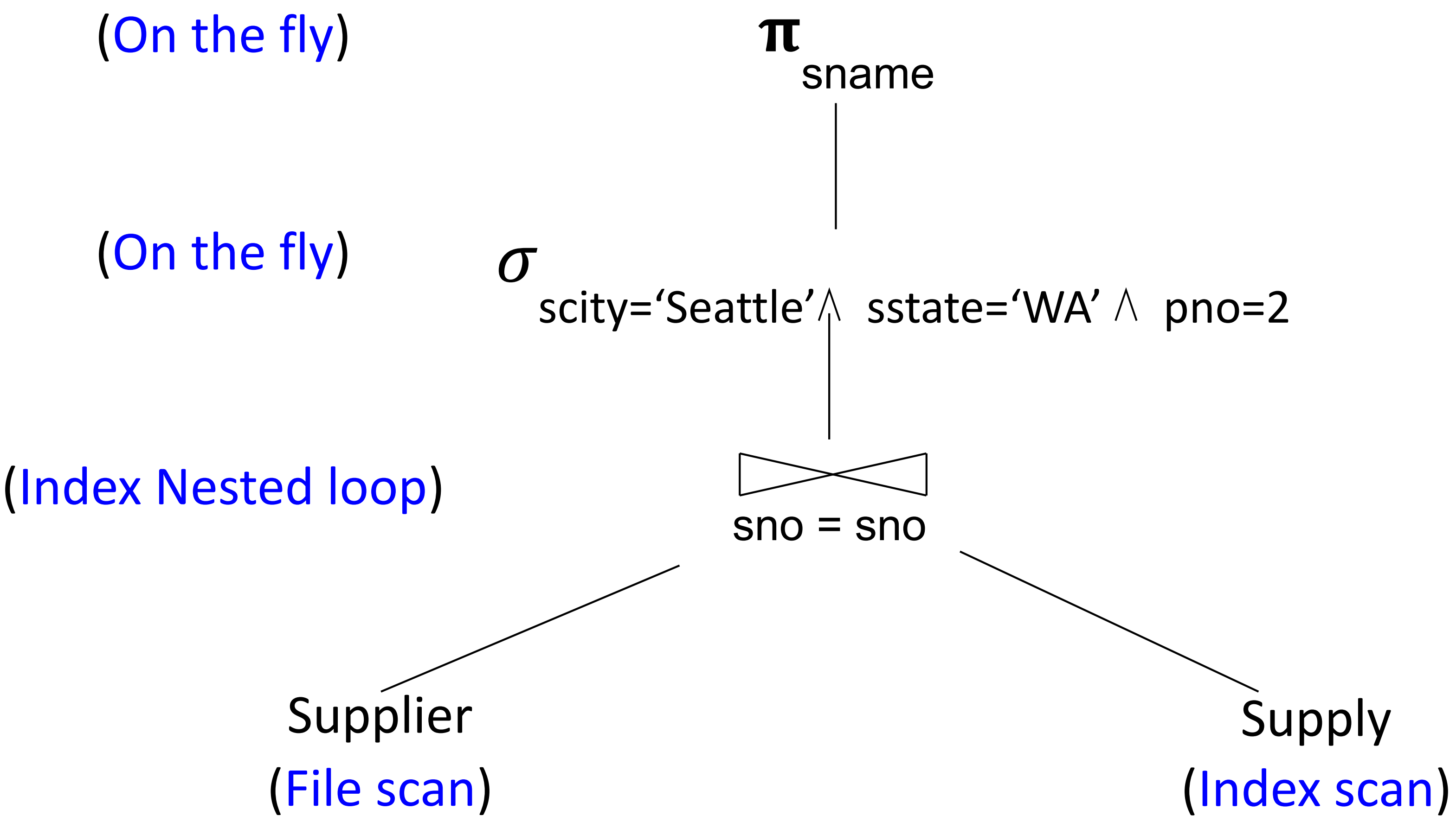
- For each logical plan…

- There exist many physical plans

Example query: Physical plan 1



Example query: Physical plan 2



Query optimizer overview

Input: A logical query plan

Output: A physical query plan

- Optimizes use of resources
- … while minimizing response time

- **Cost-based query optimization algorithm**
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost

Question: Query optimization importance

In your opinion, which stage of the optimizer takes most of the time in practice?

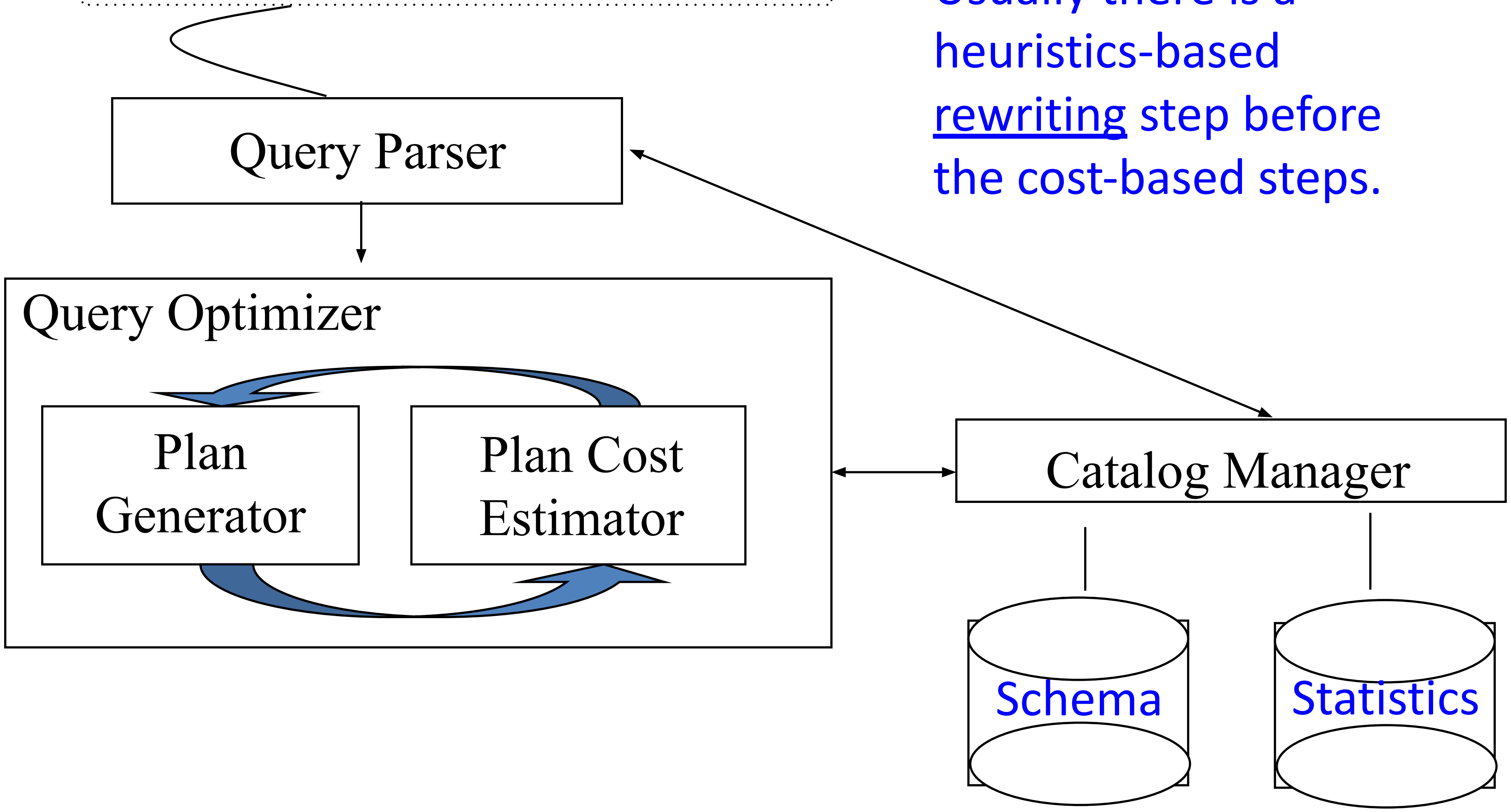
a) **Query rewrite**
b) **Logical plan enumeration**
c) **Physical plan costing**
d) **Execution itself**

Optimizer and query execution

Query

```
SELECT  S.sid
FROM Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103
```

Usually there is a heuristics-based <u>rewriting</u> step before the cost-based steps.



Today's focus

- Overview
- Query transformation
- Cost estimation
- Plan enumeration and costing
- System R strategy

Query transformation

```
Sailors(sid,sname,rating,age)
Boats(bid,bname,color)
Reserves(sid,bid,day)
```

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
  AND R.bid = 100
  AND S.rating > 5
```

1. Query first broken into “blocks”
2. Each block converted to relational algebra

Step 1: Break query into query blocks

- Query block = unit of optimization

- Nested blocks are usually treated as calls to a subroutine, made once per outer tuple
  (This is an over-simplification, but serves for now)

Outer block

```
SELECT S.sname
FROM Sailors S
WHERE S.age IN
   (WHERE MAX (S2.age)
    FROM Sailors S2)
```

Nested block

Step 2: Query block → relational algebra expr.

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

$$\pi_{S.sid}(\sigma_{B.color = \text{"red"}}(\text{Sailors} \bowtie \text{Reserves} \bowtie \text{Boats}))$$

Select-project-join optimization

- Core of every query is a **select-project-join (SPJ)** expression
- Other aspects, if any, carried out on result of SPJ core:
  - Group By (either sort or hash)
  - Having (apply filter on-the-fly)
  - Aggregation (easy once grouping done)
  - Order By (sorting is the name of the game)

- Not much room to exploit equivalences on non-SPJ parts
- Focus on optimizing SPJ core

Relational algebra equivalences

Selections: $\sigma_{c_1 \wedge \cdots \wedge c_n}(R) \equiv \sigma_{c_1}\left(\ldots\left(\sigma_{c_n}(R)\right)\right)$ *(Cascade)*

$$\sigma_{c_1}\left(\sigma_{c_2}(R)\right) \equiv \sigma_{c_2}\left(\sigma_{c_1}(R)\right) \quad (Commute)$$

Projections: $\pi_{a_1}(R) \equiv \pi_{a_1}\left(\ldots\left(\pi_{a_n}(R)\right)\right)$ *(Cascade)*

a_i is a subset of attributes of R and $a_i \subseteq a_{i+1}$ for i = 1…n-1

- These equivalences allow us to `push’ selections and projections ahead of joins

Examples

$$\sigma_{age<18 \wedge rating>5}(Sailors)$$

$$\leftrightarrow \sigma_{age<18}(\sigma_{rating>5}(Sailors))$$

$$\leftrightarrow \sigma_{rating>5}(\sigma_{age<18}(Sailors))$$

$$\pi_{age,rating}(Sailors) \leftrightarrow \pi_{age}(\pi_{rating}(Sailors))$$ (??)

$$\pi_{age,rating}(Sailors) \leftrightarrow \pi_{age,rating}(\pi_{age,rating,sid}(Sailors))$$

Another equivalence

A projection commutes with a selection that only uses attributes retained by the projection

$$\pi_{\text{age, rating, sid}} (\sigma_{\text{age<18} \wedge \text{rating>5}} (\text{Sailors}))$$

$$\leftrightarrow \sigma_{\text{age<18} \wedge \text{rating>5}} (\pi_{\text{age, rating, sid}} (\text{Sailors}))$$

Q. Can a projection always commute with selection?

→ Not always, projection commutes with selection **only if the selection predicate references no attributes that the projection would discard**

Equivalence involving joins

$$R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \qquad \textit{(Associative)}$$

$$(R \bowtie S) \equiv (S \bowtie R) \qquad \textit{(Commutative)}$$

These equivalences allow us to choose different join orders

Mixing joins with selections and projections

Converting selection + cross-product to join

$$\sigma_{S.sid = R.sid} \text{(Sailors x Reserves)}$$

$$\leftrightarrow \quad \text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves}$$

Selection on just attributes of S commutes with R $\bowtie$ S

$$\sigma_{S.age<18} (\text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves})$$

$$\leftrightarrow (\sigma_{S.age<18} (\text{Sailors})) \bowtie_{S.sid = R.sid} \text{Reserves}$$

We can also "push down" projection (*but be careful*…)

$$\pi_{S.sname} (\text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves})$$

$$\leftrightarrow \pi_{S.sname} (\pi_{sname,sid}(\text{Sailors}) \bowtie_{S.sid = R.sid} \pi_{sid}(\text{Reserves}))$$

Query rewriting

- Modern DBMS may **rewrite** queries before the optimizer sees them
- Main purpose: **de-correlate** and/or **flatten** nested queries

- De-correlation:
  - Convert correlated subquery into un-correlated subquery

- Flattening:
  - Convert query with nesting $\rightarrow$ query without nesting

Example: decorrelating a query

```sql
SELECT S.sid
FROM Sailors S
WHERE EXISTS
   (SELECT *
    FROM Reserves R
    WHERE R.bid = 103
    AND R.sid = S.sid)
```

<u>Equivalent uncorrelated query:</u>

```sql
SELECT S.sid
FROM Sailors S
WHERE S.sid IN
   (SELECT R.sid
    FROM Reserves R
    WHERE R.bid = 103)
```

Advantage: nested block only needs to be executed **once**
(rather than once per S tuple)

Example: flattening a query

```sql
SELECT S.sid
FROM Sailors S
WHERE S.sid IN
   (SELECT R.sid
    FROM Reserves R
    WHERE R.bid = 103)
```

<u>Equivalent non-nested query:</u>

```sql
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
   AND R.bid = 103
```

Advantage: can use a join algorithm

+ optimizer can select among join algorithms and reorder freely

Query transformation: Summary

- Before optimizations, queries are flattened and de-correlated
- Queries are first broken into blocks
- Blocks are then converted into relational algebra expressions
- Equivalence transformations are used to push down selections and projections

Today's focus

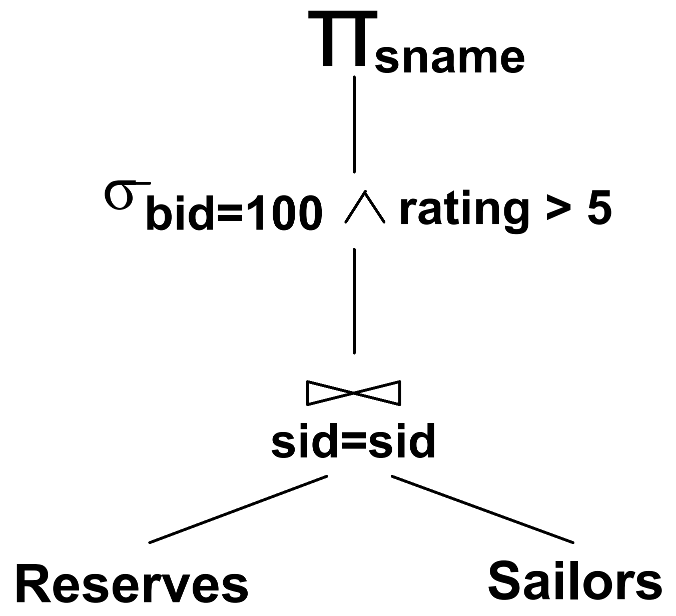
- Overview
- Query transformation
- Cost estimation
- Plan enumeration and costing
- System R strategy

Query optimization phases

1. Transformation produces relational algebra expression per “block”
2. Then, for each block, several alternative **query plans** are considered
3. Plan with lowest **estimated cost** is selected

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
   AND R.bid = 100
   AND S.rating > 5
```

$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)}(Reserves \bowtie Sailors)$$



Two main optimization issues

1. For a given query, **what plans are considered?**
2. How is the **cost of a plan estimated?**

- **Ideally:** Want to find a best plan

- **Reality:** Avoid worst plans

Cost estimation

For each plan considered, must estimate cost as follows:

- Must **estimate *cost*** of each operation in plan tree
  - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Depends on input cardinalities → # rows fed into a query operator

- Must **estimate *size of result*** for each operation in tree!
  - Use information about the input relations
  - Estimate sizes of intermediates

Statistics and catalog

- Need information about the relations and indexes involved
- *Catalogs* typically contain at least:
  - # tuples (**NTuples**) and # pages (**NPages**) per relation
  - # distinct key values (**NKeys**) for each index
  - low/high key values (**Low/High**) for each index
  - Index height (**IHeight**) for each tree index
  - # index pages (**INPages**) for each index
- Statistics in catalogs are updated periodically
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency is OK
- More detailed information (e.g., histograms of the values in some field) often stored

Size estimation and reduction factors

- Consider a query block:

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- Maximum # tuples in result → product of the cardinalities of relations in the FROM clause
- *Reduction factor (RF)* associated with each *term*
  - Reflects the impact of the *term* in reducing result size
- RF is usually called "**selectivity**"

Result size estimations for selections

- *Result cardinality* = Max # tuples * product of all RF's

  (Implicit <u>assumption</u> that values are uniformly distributed and *terms* are independent!)

- For **equality condition:** Term *col=value* (given index I on *col*)

  RF = *1/NKeys(I)*

  (NKeys(I) → # distinct values in that indexed column)

- For **range condition:** Term *col>value*

  RF = *(High(I)-value)/(High(I)-Low(I))*

  (High(I) → highest value of column col; Low(I) → lowest value of column col)

Note: if missing indexes, assume RF = 1/10

Result size estimations for joins

Q: Given a join of R and S, what is the range of possible result sizes (in #of tuples)?

- Hint: what if R_cols ∩ S_cols = ∅?

- R_cols ∩ S_cols is a key for R (and a Foreign Key in S)?

Result size estimations for joins

Q: Given a join of R and S, what is the range of possible result sizes (in #of tuples)?

- Hint: what if R_cols ∩ S_cols = ∅?
  - No common columns; simply a cross product of **|R| x |S|**

- R_cols ∩ S_cols is a key for R (and a Foreign Key in S)?

Result size estimations for joins

Q: Given a join of R and S, what is the range of possible result sizes (in #of tuples)?

- Hint: what if R_cols ∩ S_cols = ∅?
  - No common columns; simply a cross product of **|R| x |S|**

- R_cols ∩ S_cols is a key for R (and a Foreign Key in S)?
  - Multiple rows in S can match exactly one row in R

→ # result rows = # rows in S (every row in S has exactly one match)

→ **|S|**

Result size estimations for joins

- General case: R_cols ∩ S_cols = {A} (A is *not a key in either tables*)
  - Scenario 1: If NKeys(A,S) > NKeys(A,R)
    - Assume S values are a superset of R values, so each R value finds a matching value in S
    - Each tuple of R matchs **NTuples(S)/NKeys(A,S)** tuples in S (avg), so…

      est_size = NTuples(R) * NTuples(S)/NKeys(A,S)

  - Scenario 2: If NKeys(A,R) > NKeys(A,S) … symmetric argument, yielding:

    est_size = NTuples(R) * NTuples(S)/NKeys(A,R)

  - Overall:

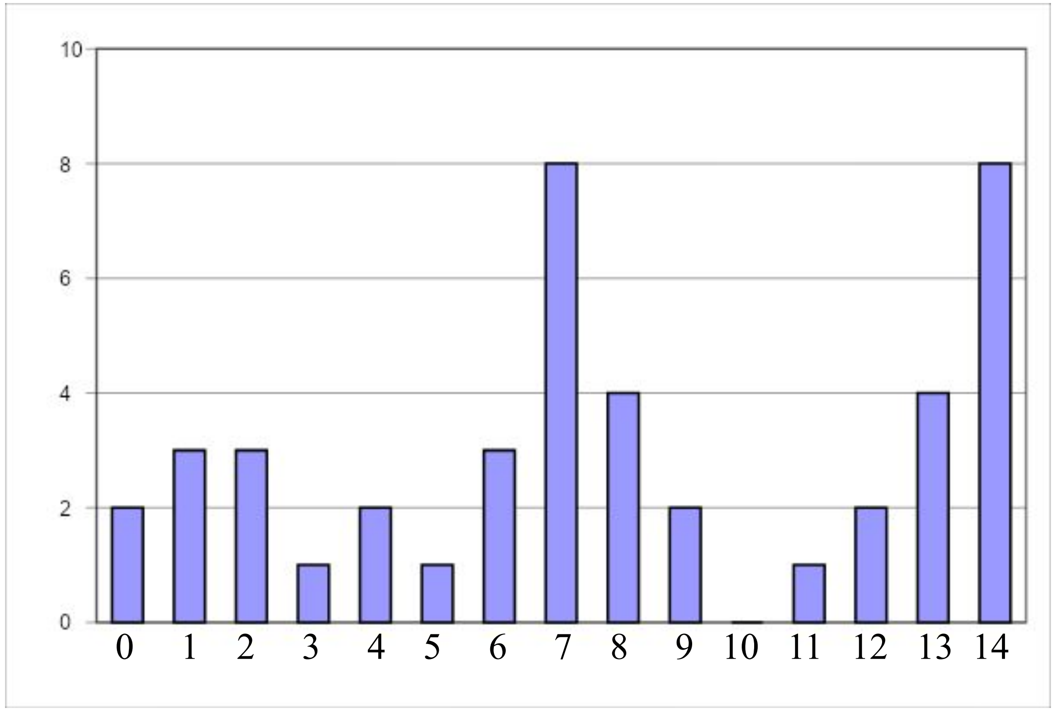
est_size = NTuples(R)*NTuples(S)/MAX{NKeys(A,S), NKeys(A,R)}

RF = 1/MAX{NKeys(A,S), NKeys(A,R)}

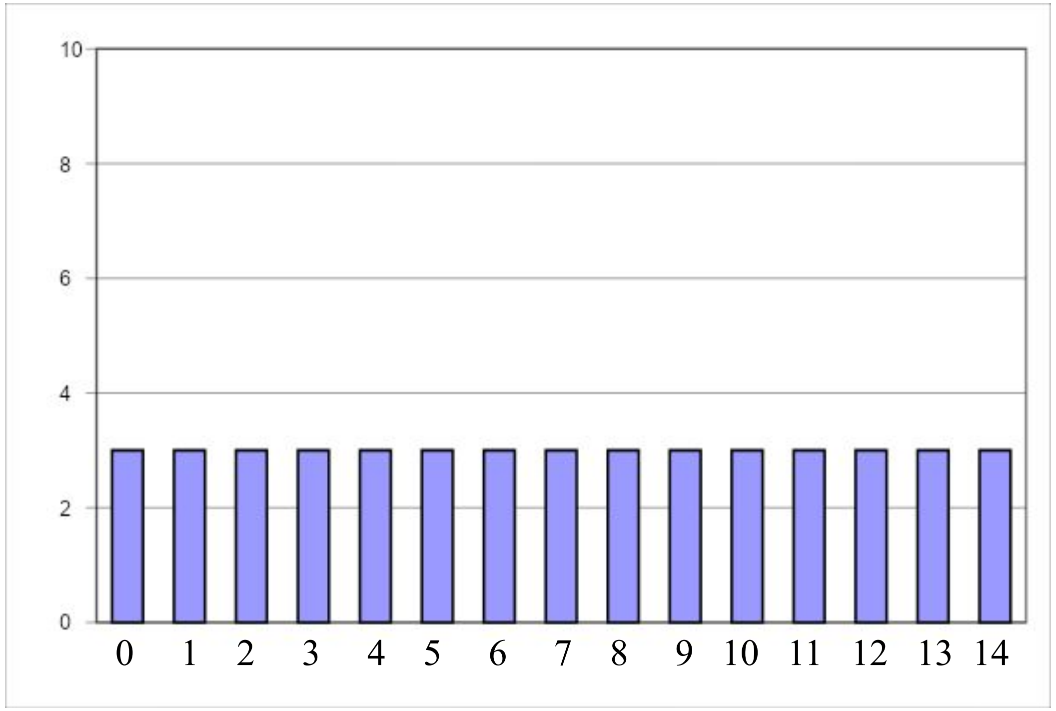
On the uniform distribution assumption

Assuming uniform distribution is rather crude

Distribution D



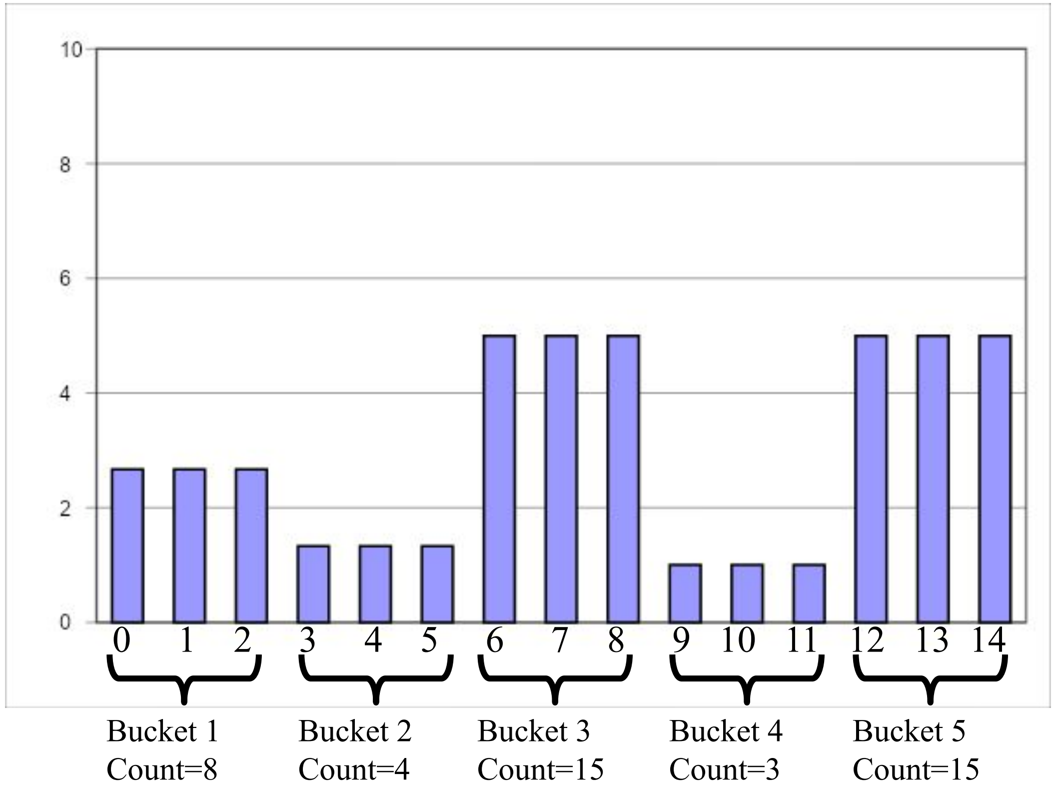
Uniform distribution approximating D



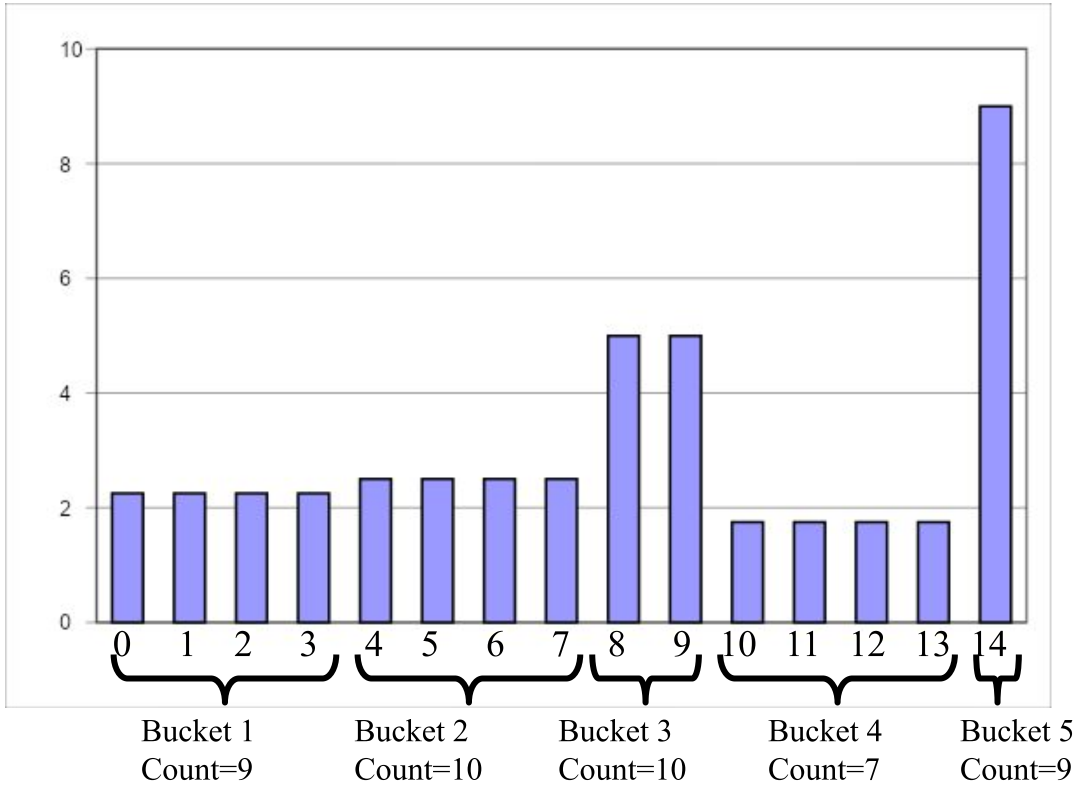
Histograms

For better estimation, use a ***histogram***

Equi-width histogram



Equi-depth histogram



Today's focus

- Overview
- Query transformation
- Cost estimation
- Plan enumeration and costing
- System R strategy

Enumeration of alternative plans

- There are two main cases:
  - **Single-relation plans**
  - **Multiple-relation plans**

- For queries over a single relation:
  - Possible access paths: full scan, index lookup, index-only
  - Consider each access path and choose the one with the least estimated cost

Cost estimates for single-relation plans

- Index on primary key matches selection:
  - *Cost is Height(I)+1 for a B+ tree, about 2.2 for hash index*
- Clustered index matching one or more conjuncts:
  - *(NPages(I)+NPages(R)) * product of RF's of matching selects.*
- Non-clustered index matching one or more conjuncts:
  - *(NPages(I)+NTuples(R)) * product of RF's of matching selects*
- Sequential scan of file:
  - *NPages(R)*

Note: *Must also charge for duplicate elimination if required*

Example

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)
Boats (*bid*: integer, *bname*: string, *color*: string)

```sql
SELECT S.sid
FROM Sailors S
WHERE rating = 8
```

Assume

Sailors has 500 pages, 40000 tuples. Data contains 10 distinct ratings

- If we have a 50-page index on *rating*:
  - Cardinality: ??
  - Clustered index: cost = ??
  - Unclustered index: cost = ??
- Doing a file scan:
  - We retrieve ?? pages

Example

Sailors (<u>*sid*: integer</u>, *sname*: string, *rating*: integer, *age*: real)
Reserves (<u>*sid*: integer, *bid*: integer, *day*: dates</u>, *rname*: string)
Boats (<u>*bid*: integer</u>, *bname*: string, *color*: string)

```
SELECT S.sid
FROM Sailors S
WHERE rating = 8
```

Assume

Sailors has 500 pages, 40000 tuples. Data contains 10 distinct ratings

- If we have a 50-page index on *rating*:
  - Cardinality: (1/NKeys(I)) * NTuples(S) = (1/10)*40000 tuples
  - Clustered index: cost = (1/NKeys(I)) * (NPages(I)+NPages(S))
    = (1/10) * (50+500) = 55 pages retrieved
  - Unclustered index: cost = (1/NKeys(I)) * (NPages(I)+NTuples(S)) =
    (1/10) * (50+40000) = 4005 pages retrieved
- Doing a file scan:
  - We retrieve all file pages (500)

Queries over multiple relations

1. Select the order of relations
    - Maximum possible orderings = N! (but no cross-products)
2. For each join, select join algorithm
3. For each input relation, select access method

Queries over multiple relations

1. Select the order of relations
    - Maximum possible orderings = N! (but no cross-products)
2. For each join, select join algorithm
3. For each input relation, select access method

Q: How many plans for a query over N relations?

Queries over multiple relations

1. Select the order of relations
    - Maximum possible orderings = N! (but no cross-products)
2. For each join, select join algorithm
3. For each input relation, select access method

Q: How many plans for a query over N relations?

Back-of-envelope calculation:

- With 3 join algorithms, I indexes per relation:

  # plans ≈ $[N!] * [3^{(N-1)}] * [(I + 1)^N]$

- Suppose N = 3, I = 2: # plans ≈ $3! * 3^2 * 3^3$ = 1458 plans

- **For each candidate plan, must estimate cost**

Query optimization is NP-complete

Pruning the search space

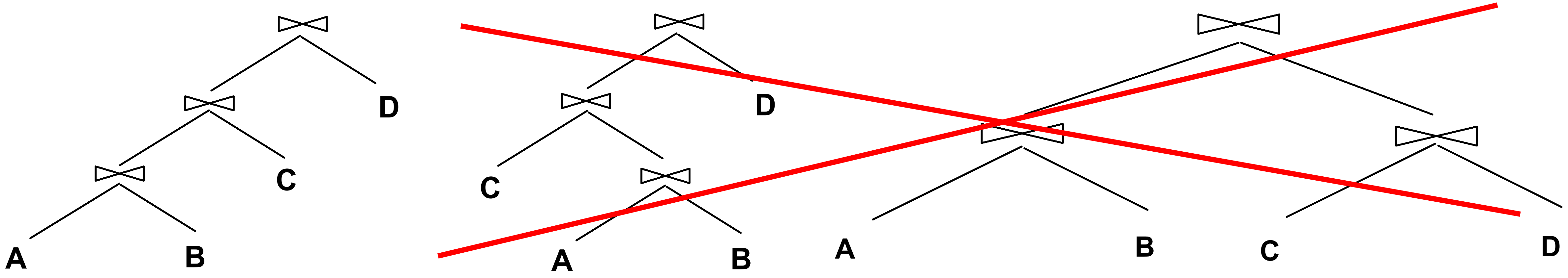
- Number of alternative plans grows rapidly as a function of the (increasing) number of joins

→ ***need to restrict search space***

- Fundamental decision (based on System R):

  Only left-deep join trees are considered

  - Left-deep trees allow us to generate all ***fully pipelined*** plans
    - Intermediate results are not written to temporary files
    - Not all left-deep trees are fully pipelined (e.g., SM join)



Plan enumeration example

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid
   AND R.bid = B.bid
```

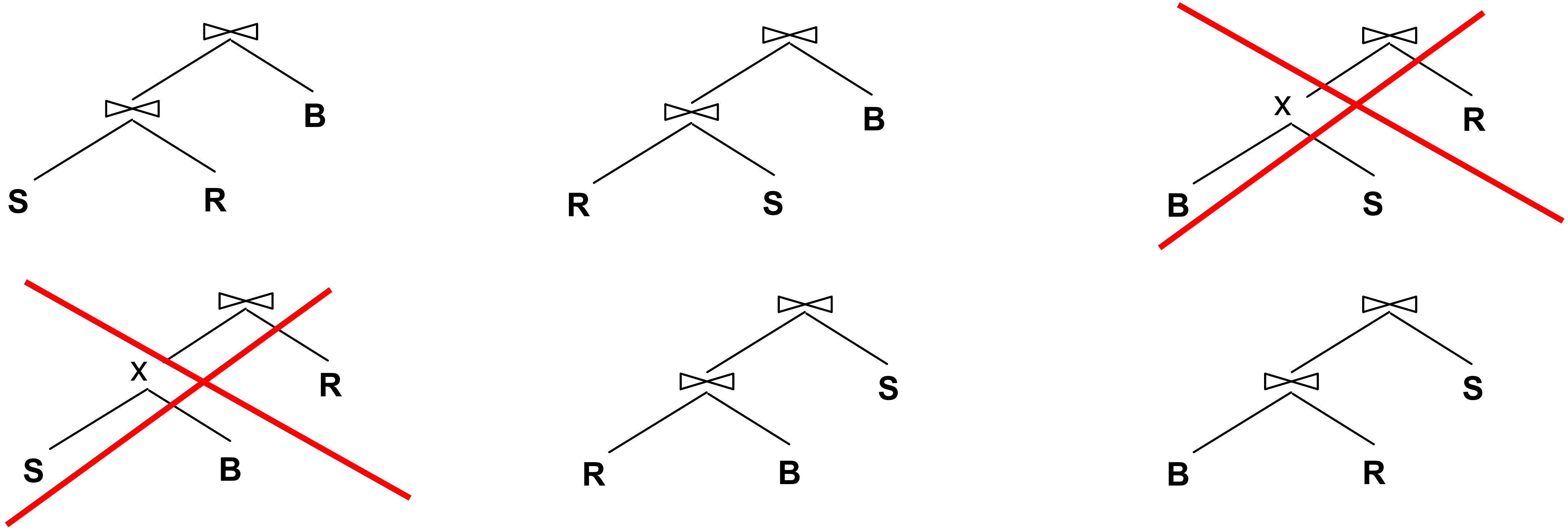
Let’s assume:

- Two join algorithms to choose from:
  - Hash-Join
  - NL-Join (page-oriented or Index-NL-Join)
- Unneeded columns removed at each stage
- Un-clustered B+Tree index on R.sid; no other indexes

Candidate plans

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid
   AND R.bid = B.bid
```

1. Enumerate relation orderings:

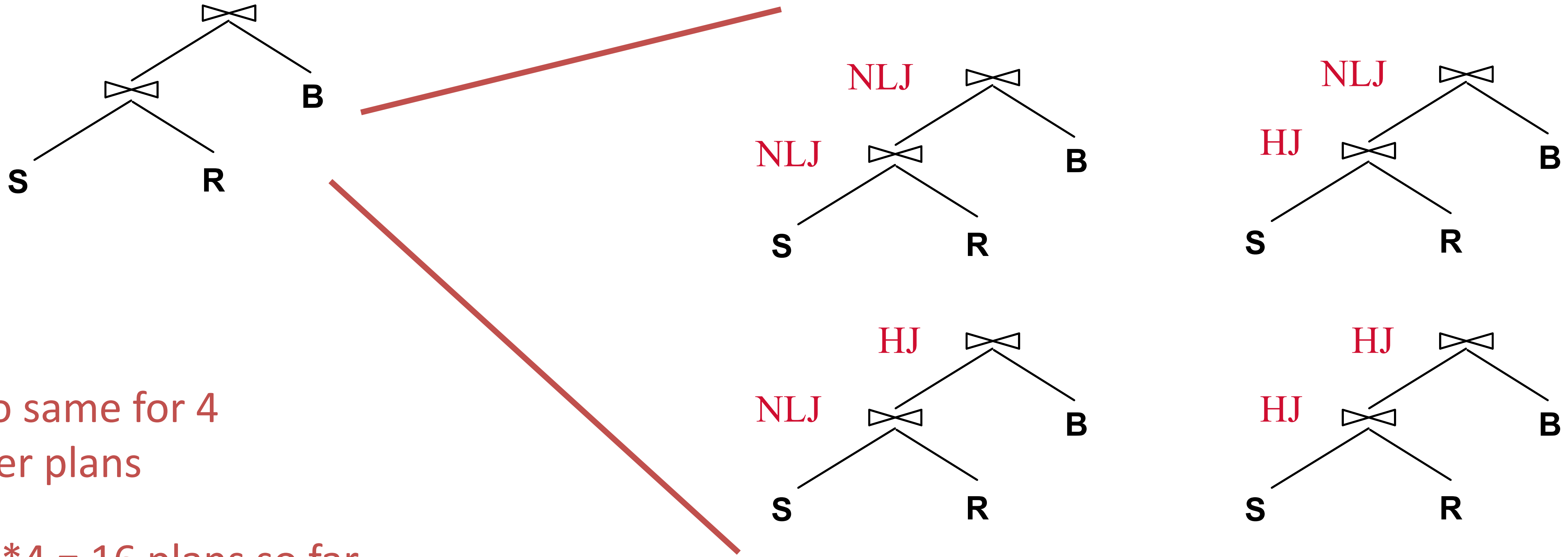


Prune plans with cross-products immediately!

Candidate plans

```sql
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid
   AND R.bid = B.bid
```

2. Enumerate join algorithm choices:



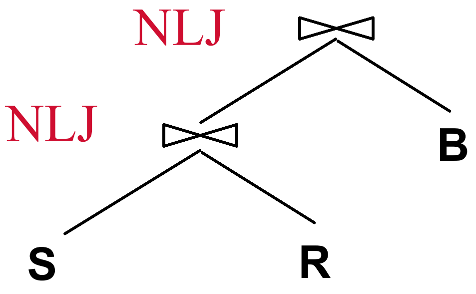
+ do same for 4 other plans

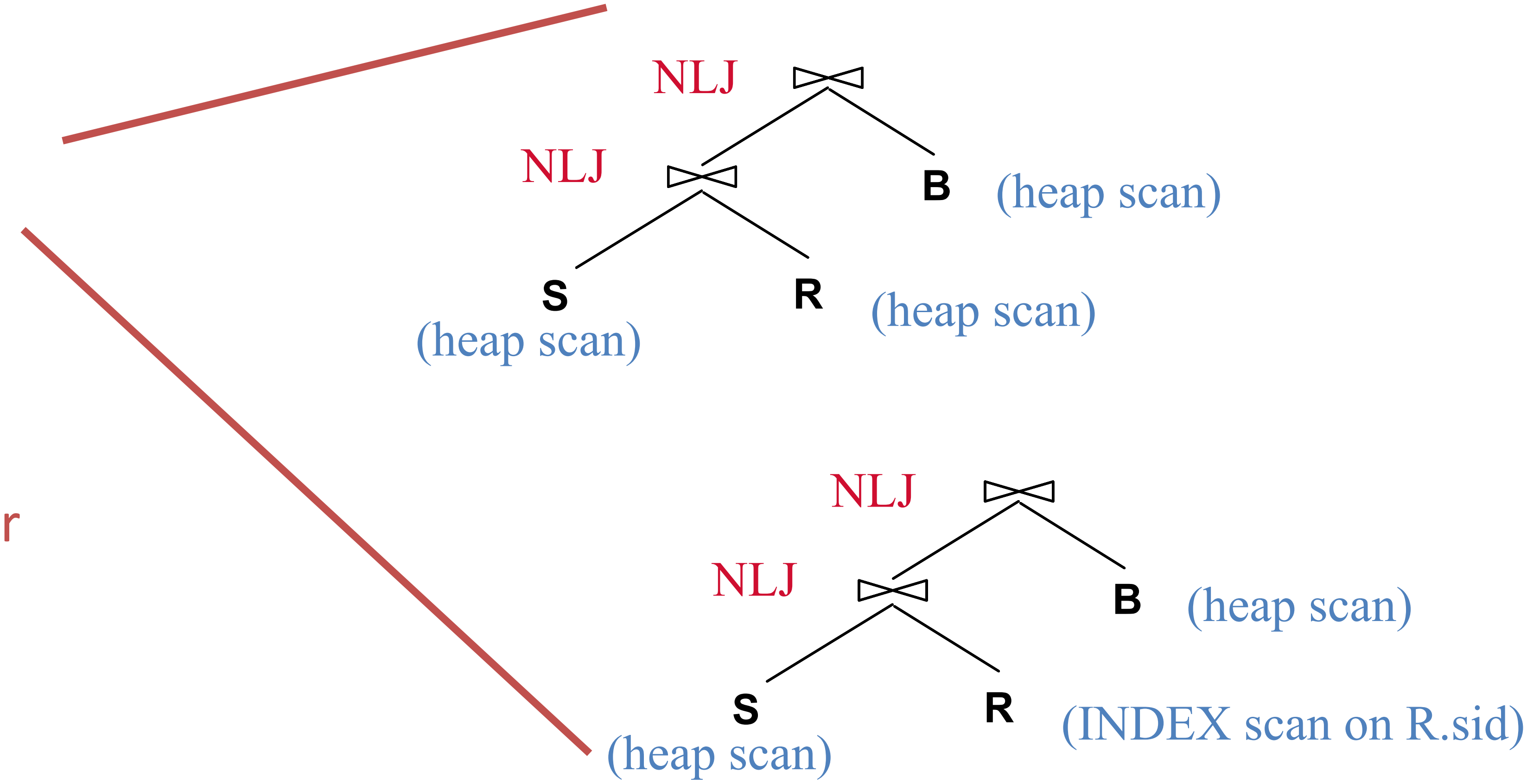
→ 4*4 = 16 plans so far..

Candidate plans

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid
   AND R.bid = B.bid
```

3. Enumerate access method choices:

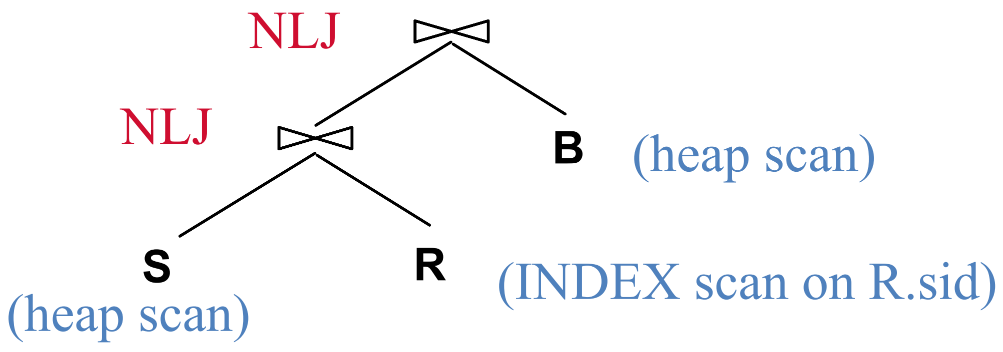




+ do same for other plans

Estimating the cost of each plan

Example:



Assume

R.sid index = 50 pages

S = 500 pages,

= 80 tuples/page

R = 1000 pages,

= 100 tuples/page

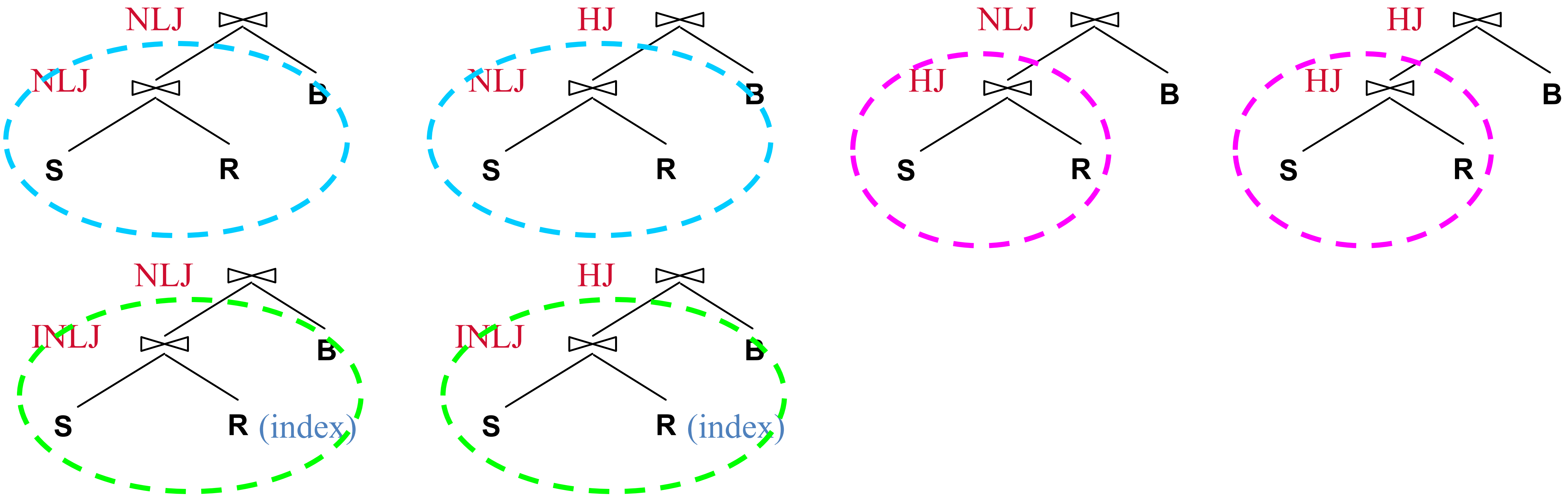
B = 10 pages

100 R $\bowtie$ S tuples/page

- Cost to scan S = 500
- Cost to join w/R = 40000 * (1/40000)(50+100,000) = 100,050
- Size of S $\bowtie$ R = (100,000 * 40,000)/40,000; 100,000/100 = 1000 pages
- Cost to join with B = 1000 * 10 = 10000

Total estimated cost = 500 + 100,050 + 10000 = 110,550

Enumerated plans (just the S-R-B ones)



Observe that many plans share common sub-plans (i.e., only upper part differs)

Today's focus

- Overview
- Query transformation
- Cost estimation
- Plan enumeration and costing
- System R strategy

Improved strategy (used in System R)

- Shared sub-plan observation suggests a better strategy:
- Enumerate plans using N passes (N = # relations joined):
  - Pass 1: Find best 1-relation plans for each relation
  - Pass 2: Find best ways to join result of each 1-relation plan as outer to another relation
    (All 2-relation plans.)
  - Pass N: Find best ways to join result of a (N-1)-relation plan as outer to the Nth relation
    (All N-relation plans.)

- For each subset of relations, retain only:
  - Cheapest subplan overall (possibly unordered), plus
  - Cheapest subplan for each *interesting order* of the tuples
- For each subplan retained, remember cost and result size estimates

A note on “interesting orders”

An intermediate result has an “interesting order” if it is sorted by any of:

- ORDER BY attributes
- GROUP BY attributes
- Join attributes of other joins

System R's plan enumeration

- A N-1 way plan is not combined with an additional relation unless there is a join condition between them (unless all predicates in WHERE have been used up) i.e., avoid Cartesian products if possible

- Always push all selections & projections as far down in the plans as possible

$\longrightarrow$ A good strategy, as long as these operations are cheap

System R's plan enumeration example

```
SELECT S.sname, B.bname, R.day
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid
   AND R.bid = B.bid
```

Table	tuples/ Page	Pages
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

- This time let's assume:
  - Two join algorithms
    - Sort-Merge-Join
    - Page-oriented NL-Join
  - Clustered B+Tree on S.sid (height=3; 500 leaf pages)
  - S has 10,000 pages, 5 tuples/page
  - R has 10 pages, 10 tuples/page
  - B has 10 pages, 20 tuples/page
  - 10 R $\bowtie$ S tuples fit on a page
  - 10 R $\bowtie$ B tuples fit on a page

Pass 1 (single-relation subplans)

- S: (a) heap scan or (b) scan index on S.sid
  - a) heap scan cost = 10,000
  - b) index scan cost = 500 + 10,000 = 10,500

Retain both, since (b) has "interesting order" by sid

Table	tuples/ Page	Pages
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

- R: heap scan only option

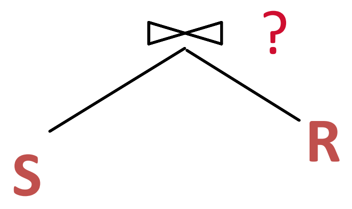
$\longrightarrow$ Cost = 10

- B: heap scan only option

$\longrightarrow$ Cost = 10

Pass 2 (2-relation subplans)

Starting with S as outer



- Heap scan-S as outer:

  a) NL-Join with R

  → cost = 10,000 + 10,000(10) = 110,000

  b) SM-Join with R

  → cost = 3*(10,000+10) = 30,030

- Index scan-S as outer (gives S in sorted order):

  c) NL-Join with R

  → cost = 10,500 + 10,000(10) = 110,500

  d) SM-Join with R

  → cost = 10,500 + 3*10 = 10,530

Table	**tuples/ Page**	**Pages**
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

Pass 2 (contd ...)

Starting with R as outer

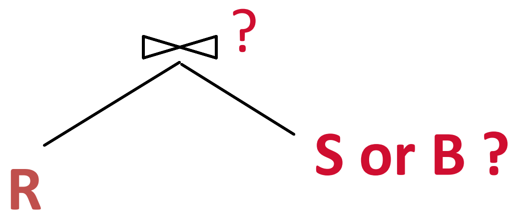


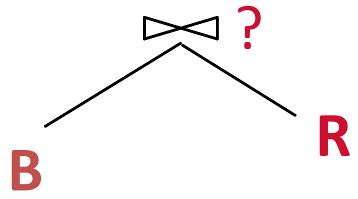
Table	tuples/ Page	Pages
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

- Join with S:
  - a) NL-Join with S, cost = 10 + 10(10,000) = 100,010
  - **b) Index-NL-Join with Index-S, cost = 10 + 100*4 = 410**
  - c) SM-Join with S, cost = 3*(10,000 + 10) = 30,030
  - d) SM-Join with Index-S, cost = 3 * 10 + 10,500 = 10,530

- Join with B:
  - a) NL-Join with B, cost = 10 + 10(10) = 110
  - **b) SM-Join with B, cost = 3*(10+10) = 60**

Pass 2 (contd …)

Starting with B as outer



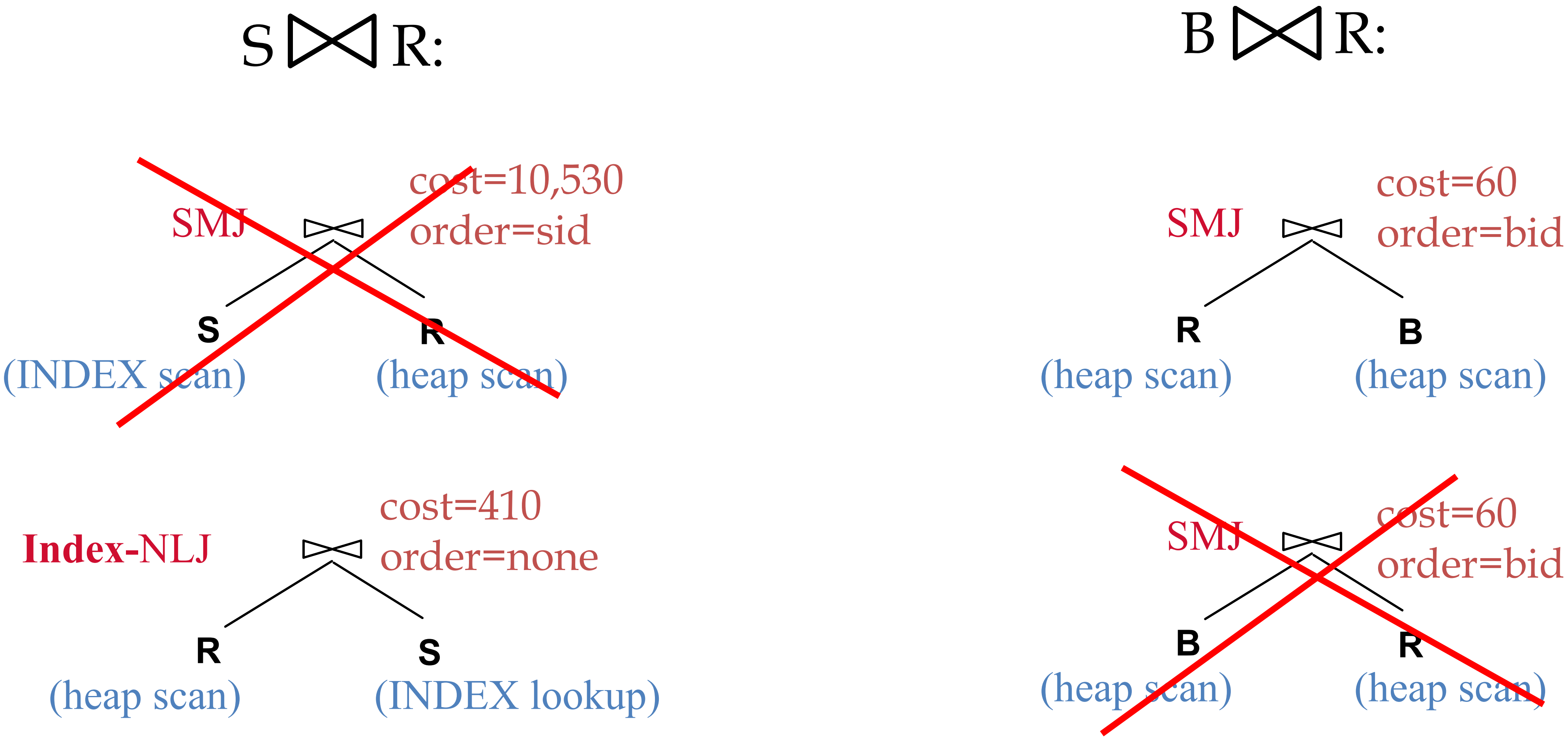
- Join with R:

  a) NL-Join with R, cost = 10 + 10(10) = 110

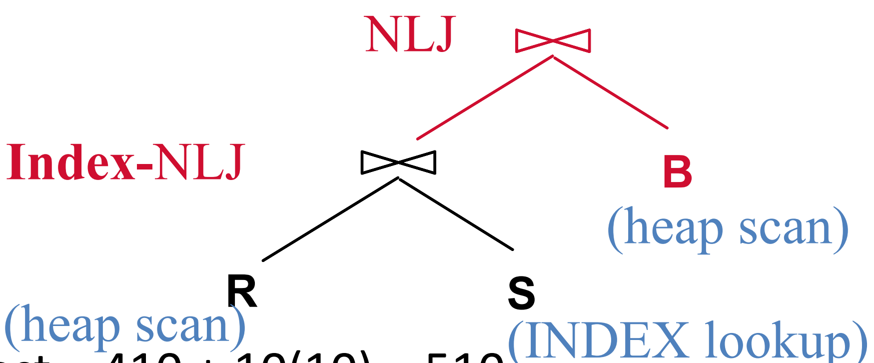
  b) SM-Join with R, cost = 3*(10+10) = 60

Table	**tuples/ Page**	**Pages**
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

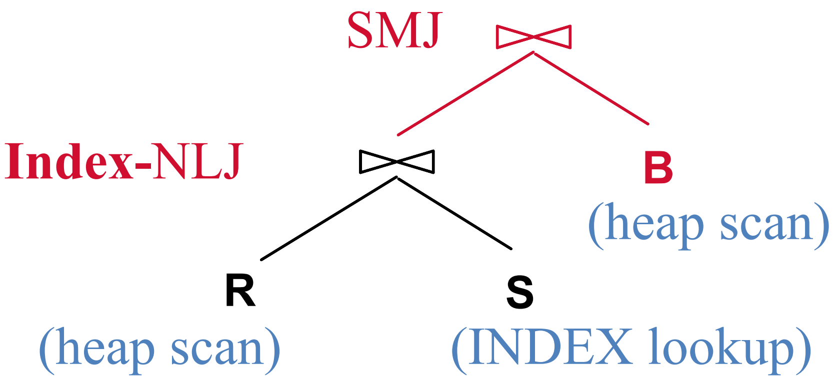
Further pruning of 2-relation subplans



Pass 3 (3-relation subplans)



cost = 410 + 10(10) = 510



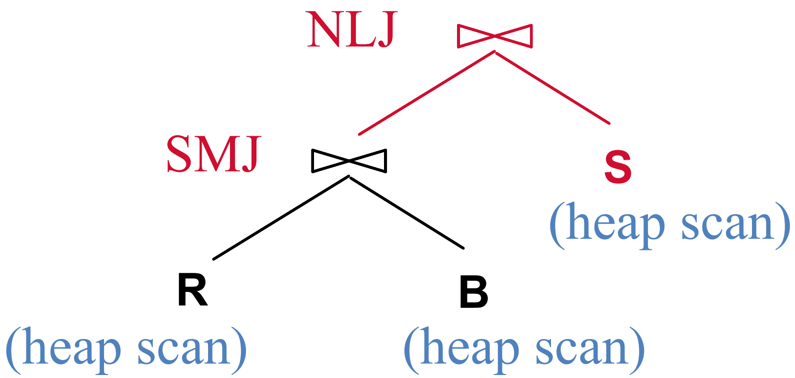
cost = 410 + 2*10 + 3*10 = 460

Table	tuples/ Page	Pages
S	5	10000
B+tree (S)		3i/500l
R	10	10
B	20	10

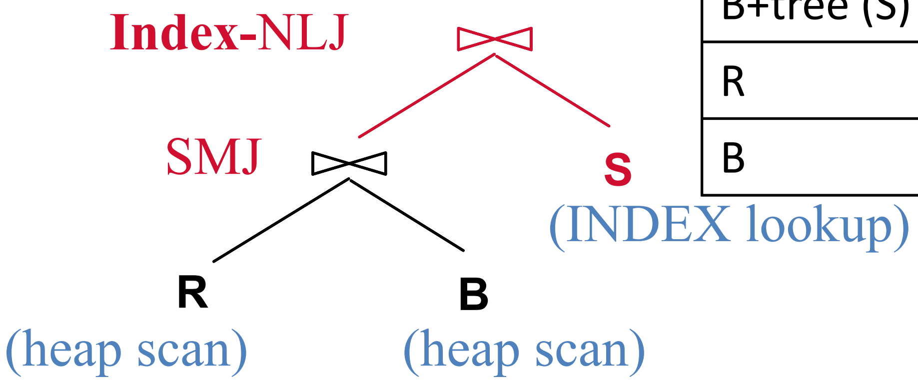
S ⋈ R subplan:
cost=410
order=none
result size = 10 pages

Pass 3 (contd …)

Table	Nrecs/ Page	Pages
S	5	10000
B+tree (S)		500
R	10	10
B	20	10

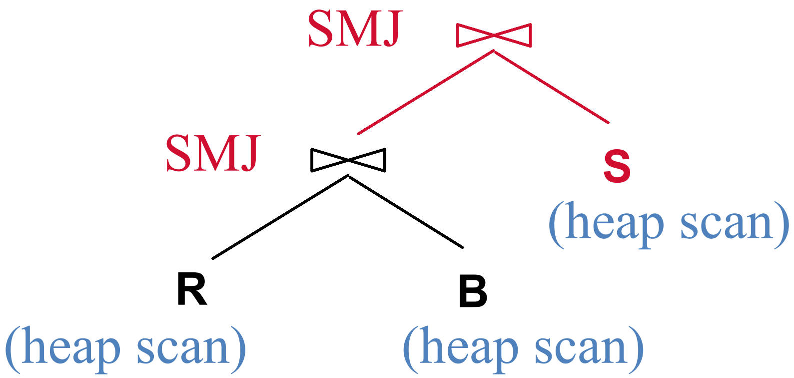


Cost = 60 + 10(10,000)
= 100,060

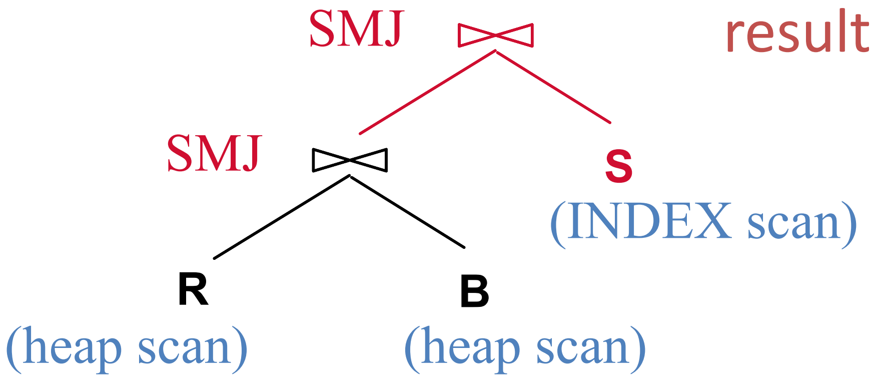


cost = 60 + 100*4 = 460

B $\bowtie$ R subplan:
cost=60, order=bid
result size = 10 pages

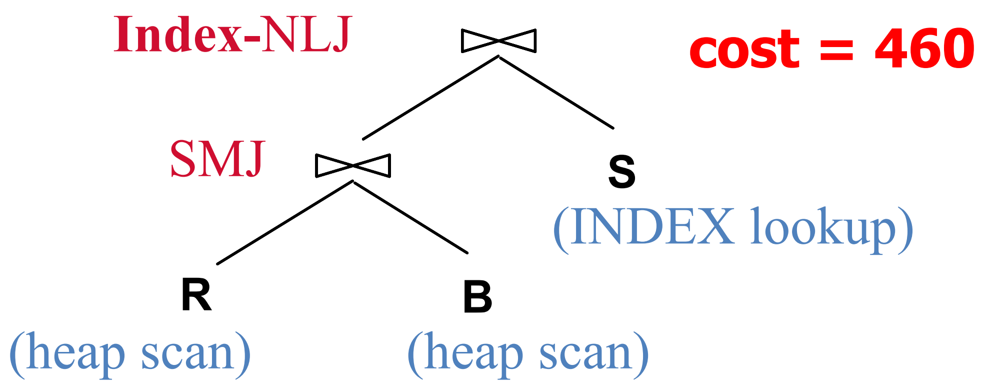


Cost = 60 + 10*2 + 3*10,000
= 30,080



Cost = 60 + 10*2 + 10,500
= 10,580

And the winner is …



Observations:

- Best plan mixes join algorithms
- Worst plan had cost > 100,000

  (exact cost unknown due to pruning)

  - Optimization yielded ~ **1000-fold improvement** over worst plan!

Some notes wrt reality …

- In spite of pruning plan space, this approach is still exponential in the # of tables
  Rule of thumb: works well for < 10 joins

- In real systems, COST considered is:

#IOs + *factor* * #CPU Instructions

System R strategy: summary

- Enumerate plans using N passes (N = # relations joined):

- For each subset of relations, retain only:
  - Cheapest subplan overall (possibly unordered), plus
  - Cheapest subplan for each ***interesting order*** of the tuples

- For each subplan retained, remember cost and result size estimates