Relational Query Optimization

SUN YAT-SEN UNIVERSITY

Review

- 单个关系运算的求值方法
  - 选择运算：cheapest access path（文件扫描、索引）
  - 投影运算: 消除重复（排序）
  - 连接运算：Nested Loops Join、Sort-Merge Join
- 相应的开销估算公式
  - 外排序 $$2N(1+\lceil \log_{B-1}\lceil N/B\rceil\rceil)$$
  - 块嵌套循环连接算法 **Cost = [R] + [R]/N * [S]**
  - 排序归并连接算法 Cost: Sort R + Sort S + ([R]+[S])

Review

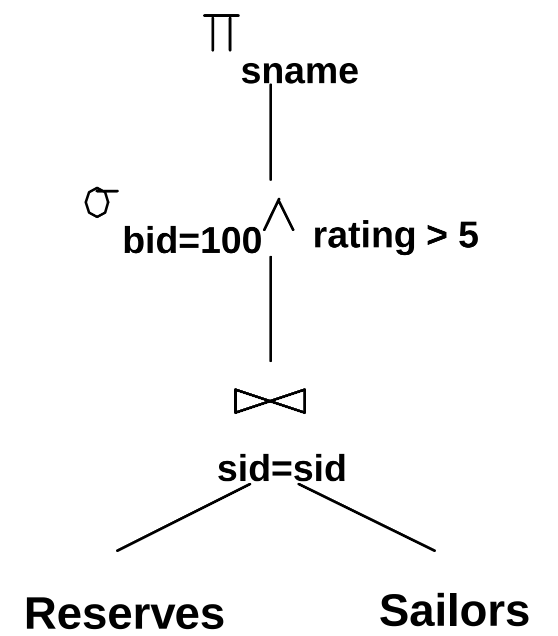
- Choice of single-table operations
  - Depends on indexes, memory, stats,…
- Joins
  - Blocked nested loops:
    - simple, exploits extra memory
  - Indexed nested loops:
    - best if 1 rel small and one indexed
  - Sort/Merge Join
    - good with small amount of memory, bad with duplicates
  - Hash Join
    - fast (enough memory), bad with skewed data

Query Optimization Overview

- Query can be converted to relational algebra
  - Relational Algebra converts to tree, joins form branches

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

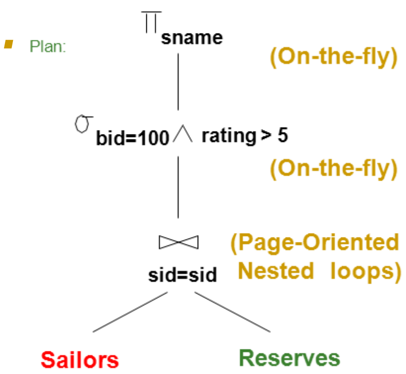
$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)} \text{ (Reserves} \bowtie \text{Sailors)}$$



- Each operator has implementation choices
- Operators can also be applied in different order!

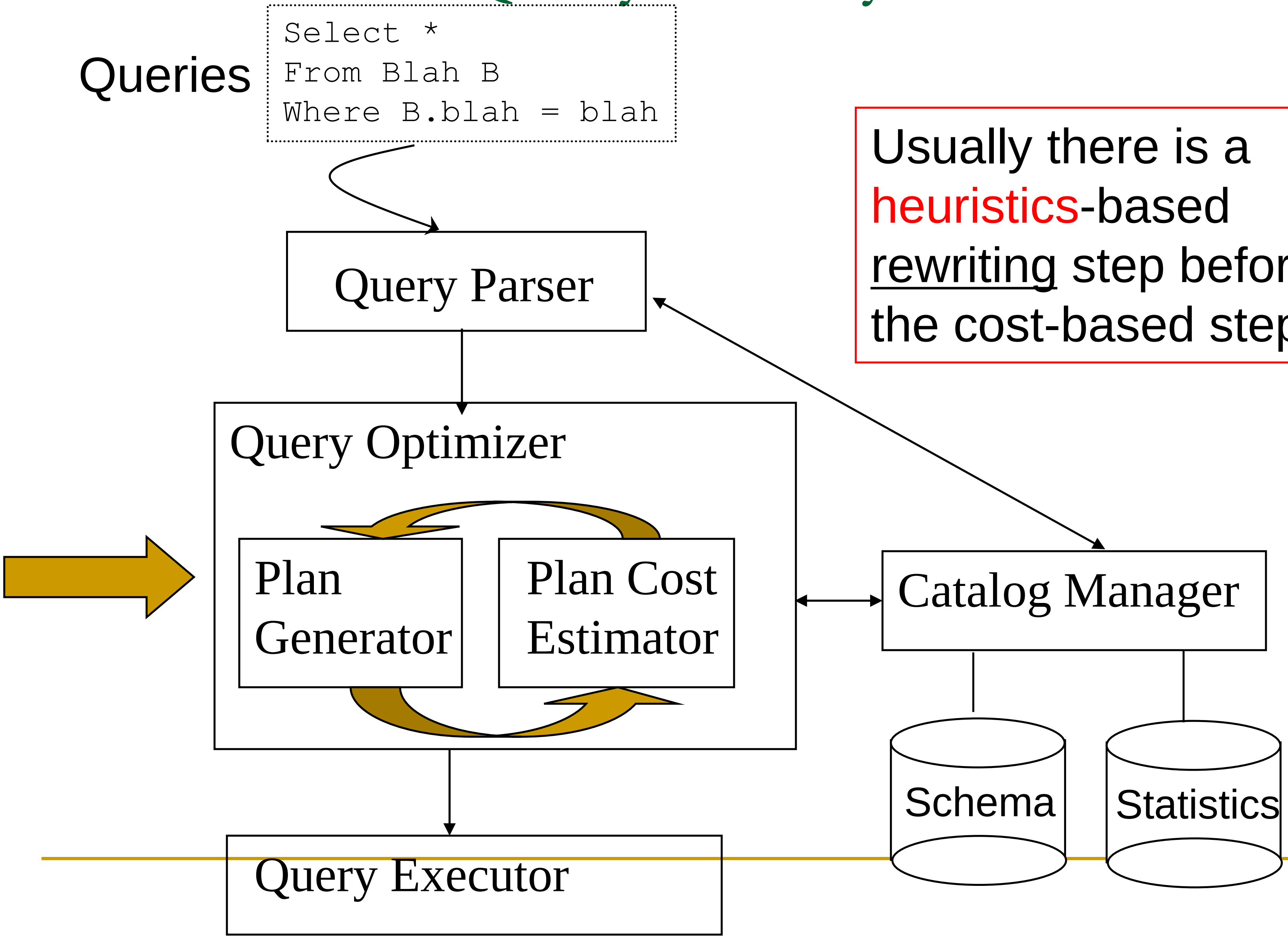
Query Optimization Overview (cont.)

- *Plan(执行计划,查询求解计划):* *Tree of Relation Algebra operations (and some others) with choice of algorithm for each operation.*



- Three main issues:
  - For a given query, what plans are considered?
  - How is the cost of a plan estimated?
  - How do we "search" in the "plan space"?
- Ideally: Want to find best plan.
- Reality: Avoid worst plans!

Cost-based Query Sub-System



Usually there is a heuristics-based rewriting step before the cost-based steps.

Schema for Examples

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

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats

[R]=1000, p_R=100
100 boats

- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings

[S]=500, p_S=80.
10 ratings

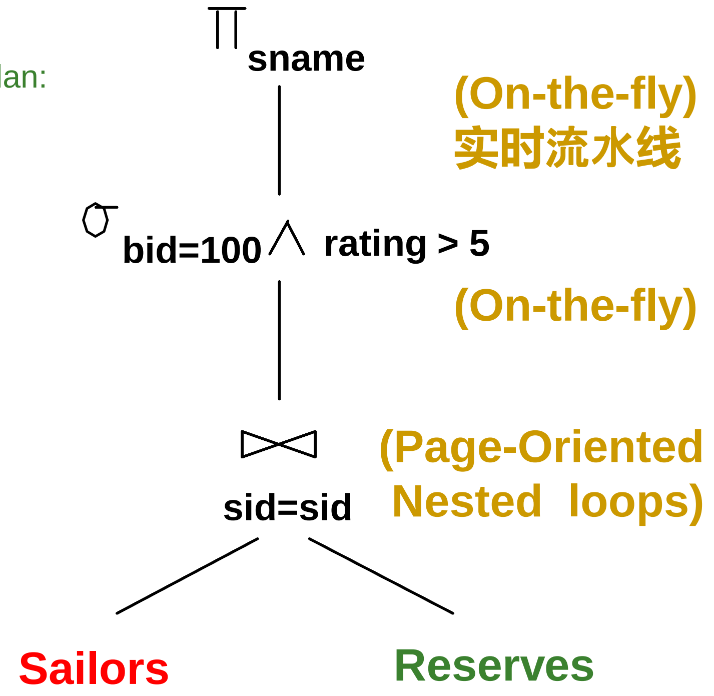
- Assume we have 5 pages in our buffer pool!

Motivating Example

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

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages

Plan:



- Cost: 500+500*1000 I/Os=500,500 IOs

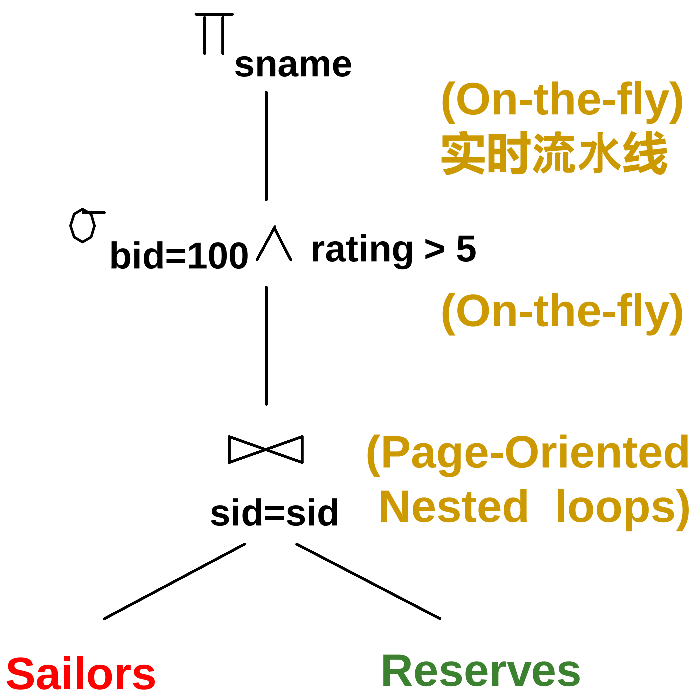
  Cost = [O] + [O] * [I]

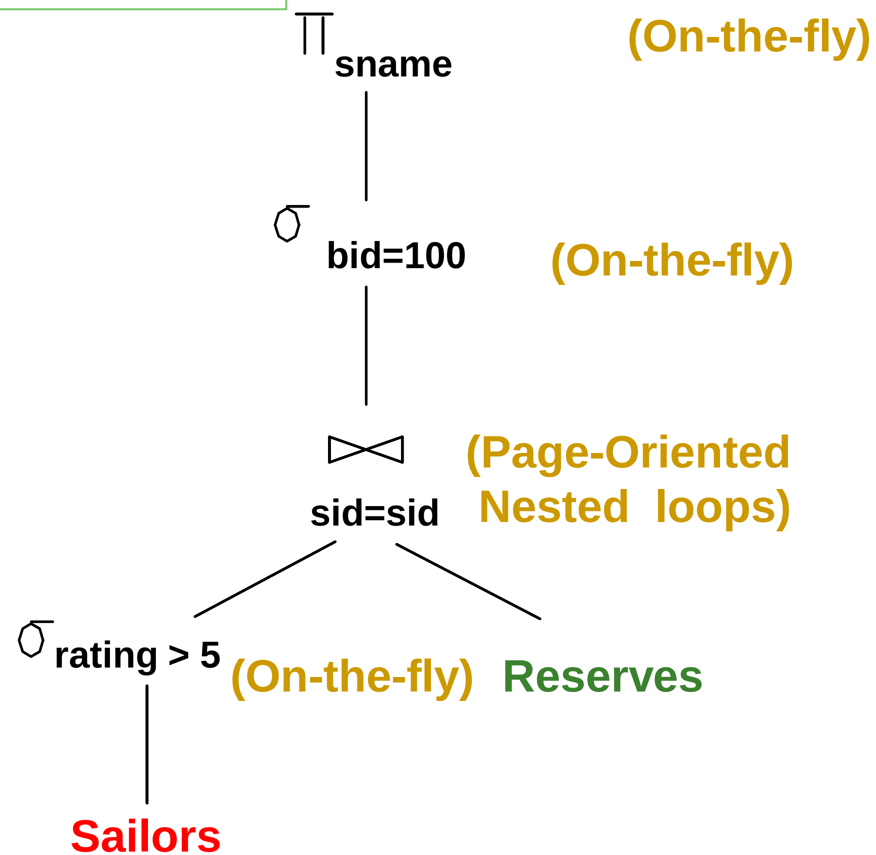
- By no means the worst plan!
- Misses several opportunities:
  - selections could be`pushed’ down(下推)
  - no use made of indexes
- *Goal of optimization:* Find faster plans that compute the same answer.

Alternative Plans – Push Selects (No Indexes)　下推选择

[R]=1000, p_R=100
100 boats
[S]=500,　p_S=80.
10 ratings
5 buffer pages

Cost = [O] + [O] * [I]





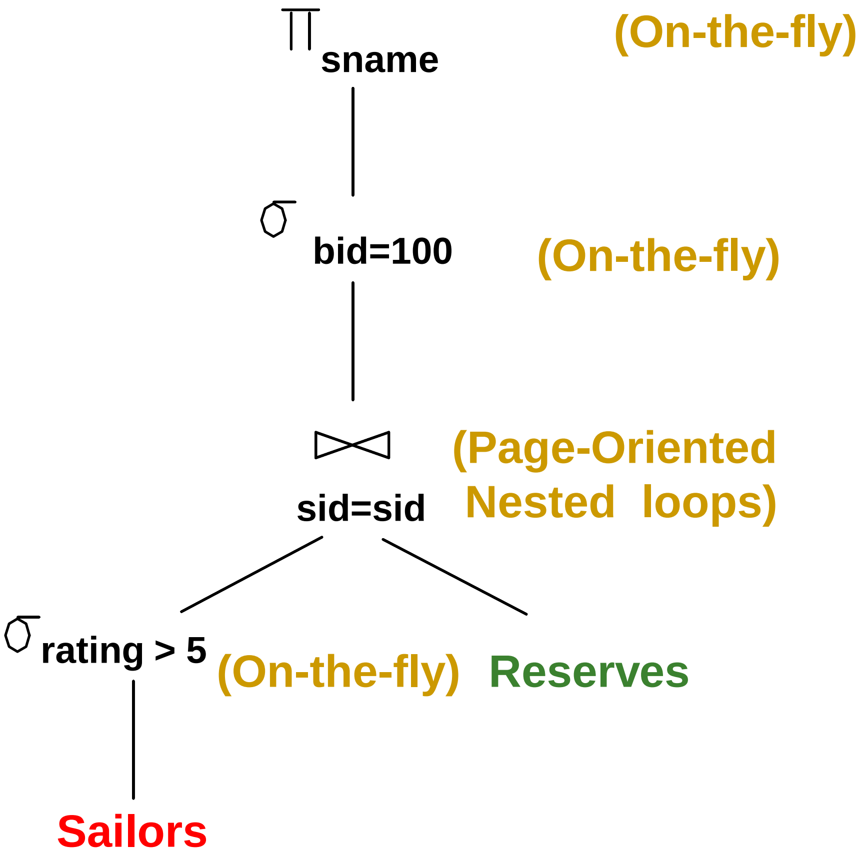
500,500 IOs

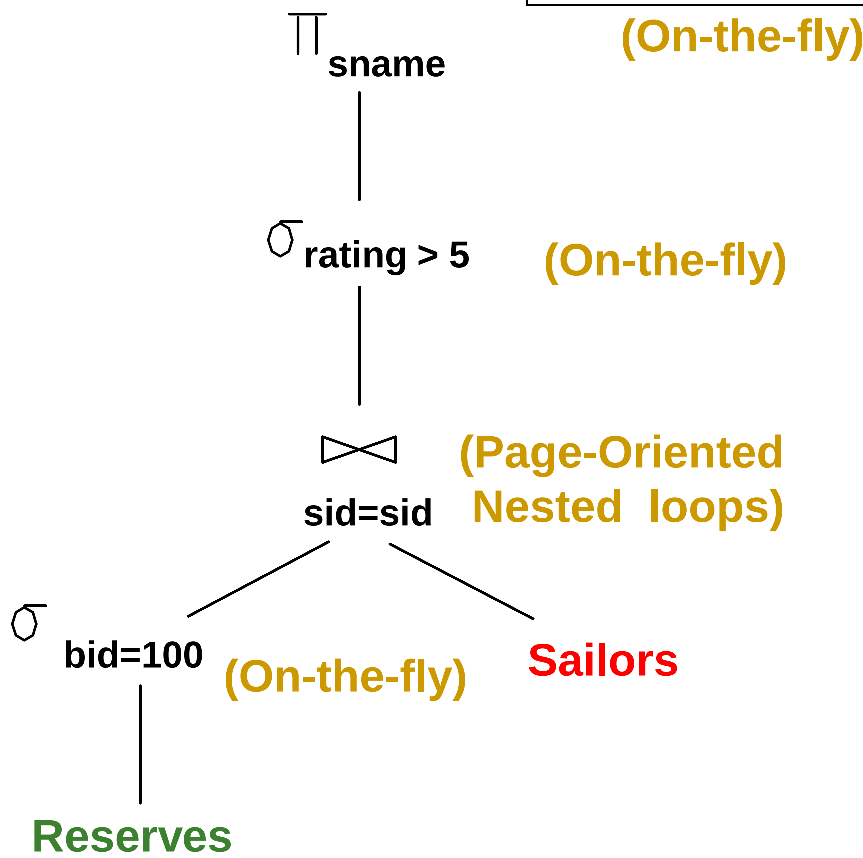
500+~~250~~+250*1000 =250,500 IOs

Alternative Plans – Push Selects (No Indexes)

Cost = [O] + [O] * [I]

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages





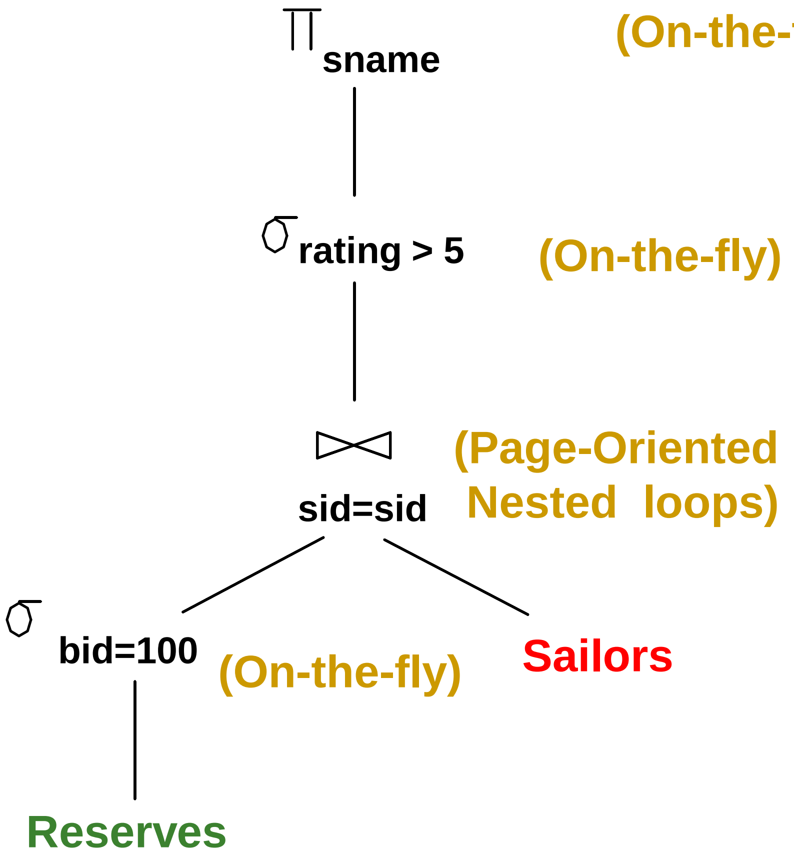
250,500 IOs

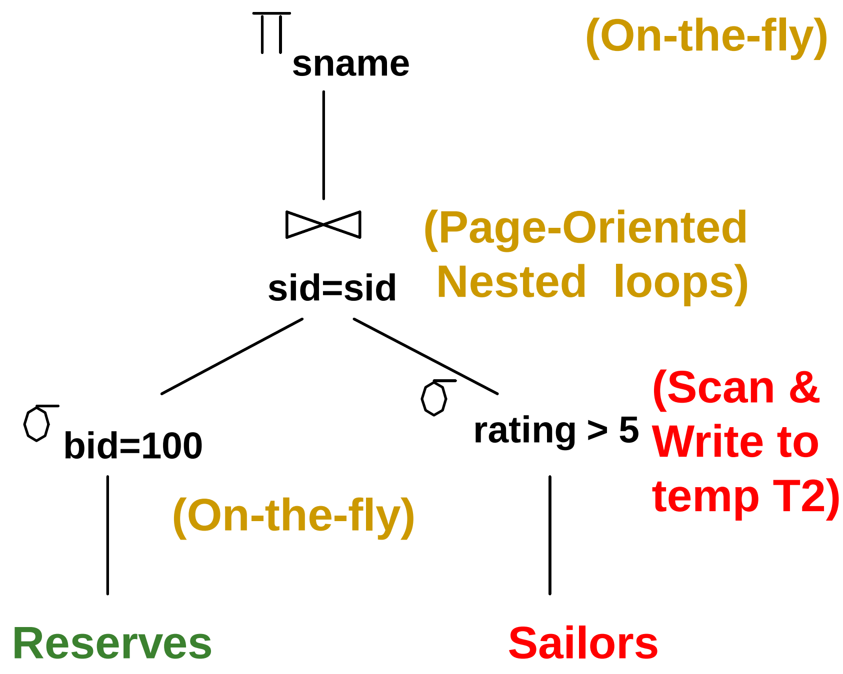
1000 +10*500=6000 IOs

Alternative Plans – Push Selects (No Indexes)

Cost = [O] + [O] * [I]

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages





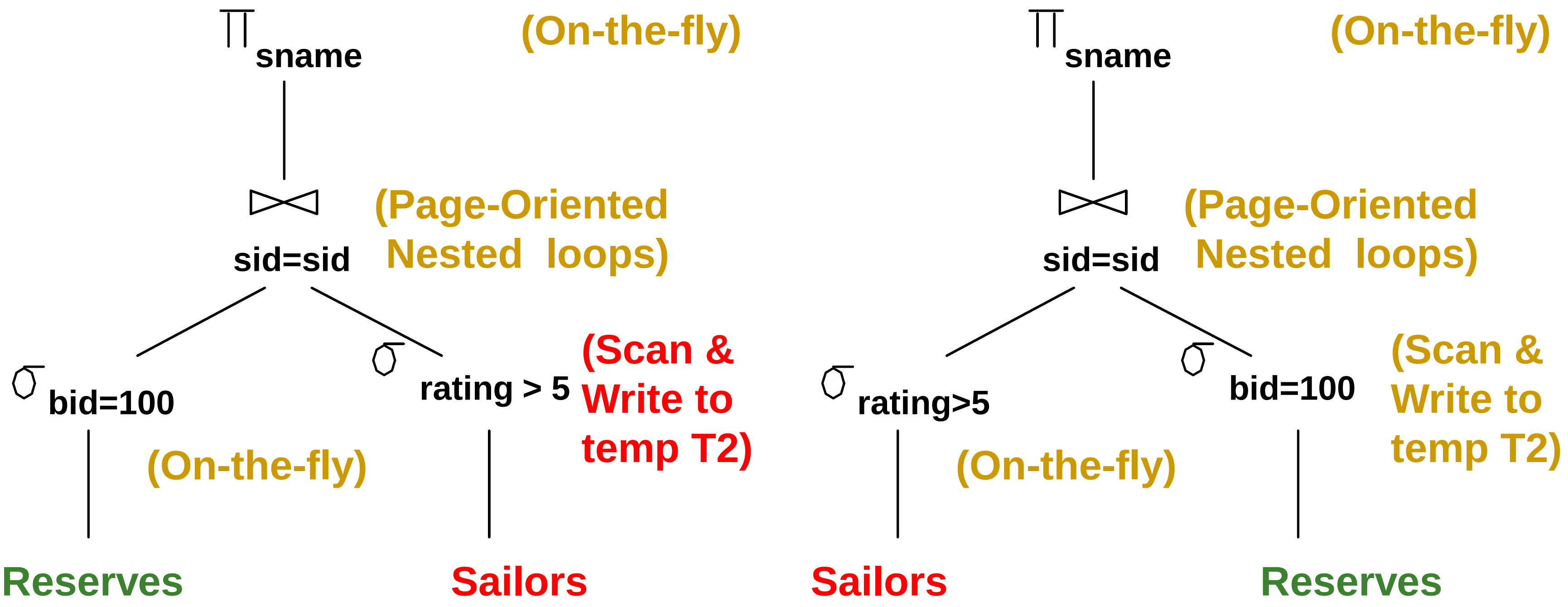
6000 IOs

1000 + 500+ 250 + (10 * 250)=4250 IOs

Alternative Plans – Push Selects (No Indexes)

Cost = [O] + [O] * [I]

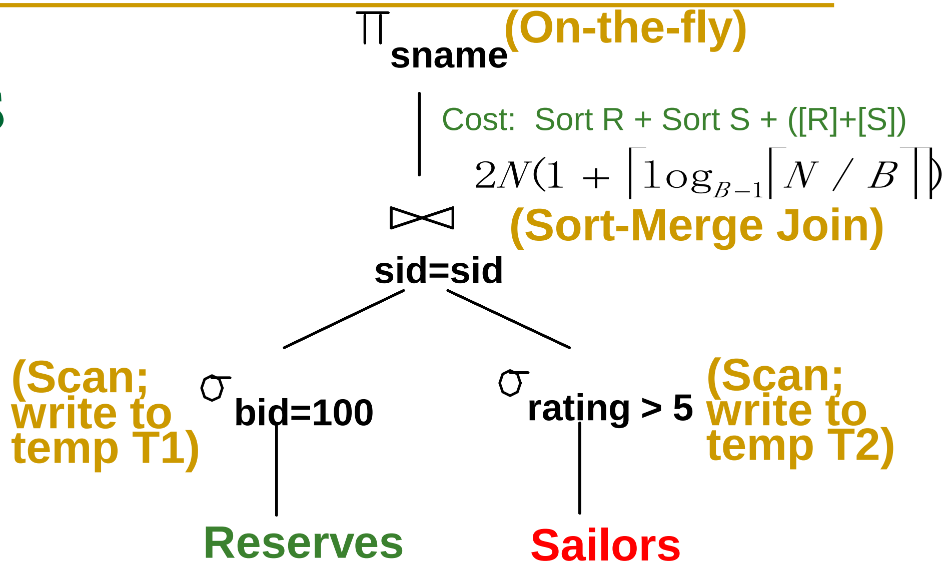
[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages



4250 IOs

500 + 1000 +10 +(250 *10)=4010 IOs

More Alternative Plans (No Indexes)



- ***Sort Merge Join***
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages) = 1010.
  - Scan Sailors (500) + write temp T2 (250 pages) = 750.
  - Sort T1 (2*2*10) + sort T2 (2*4*250) + merge (10+250) = 2300
  - Total: 4060 page I/Os.

Cost = [O] + [O]/N * [I]

- If use BNL join, join = 10+4*250, total cost = 2770.
- Can also `push' projections, but must be careful!
  - T1 has only *sid*, T2 only *sid*, *sname*:
  - T1 fits in 3 pgs, cost of BNL under 250 pgs, total < 2000.

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages

Summing up

- There are *lots* of plans
  - Even for a relatively simple query
- People tend to think they can pick good ones by hand
  - MapReduce is based on that assumption
- Not so clear that's true!
  - Machines are better at enumerating options than people
  - But we will see soon how optimizers make simplifying assumptions

What is Needed for Optimization?

- A closed set of operators
  - Relational ops (table in, table out)
  - Encapsulation (e.g. based on iterators)
- Plan space
  - Based on relational equivalences, different implementations
- Cost Estimation, based on
  - Cost formulas
  - Size estimation, in turn based on
    - Catalog information on base tables
    - Selectivity (Reduction Factor) estimation
- A search algorithm: To sift through the plan space and find lowest cost option!

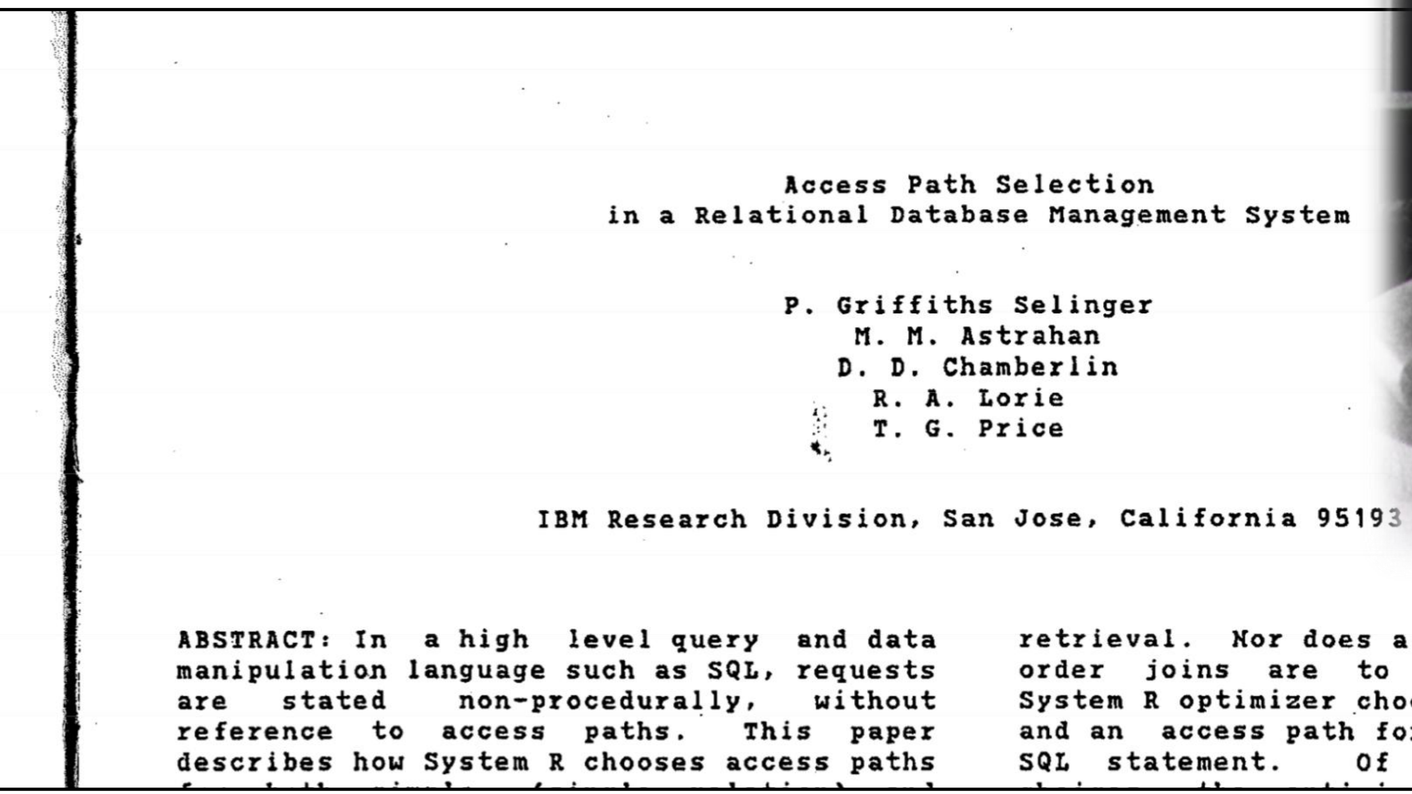
Cost = [R] + [R]/N *[S]

$$2N(1+\lceil \log_{B-1} \lceil N/B \rceil \rceil)$$

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages

Query Optimization

- Will focus on “System R” (Selinger) style optimizers



Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths

retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible

Highlights of System R Optimizer

- Impact:
  - Most widely used currently; works well for 10-15 joins.
- Cost estimation:
  - Very inexact, but works OK in practice.
  - Statistics in system catalogs used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.

Highlights of System R Optimizer (Contd)

- Plan Space: Too large, must be pruned.
  - Many plans share common, “overpriced” subtrees
    - ignore them all!
  - In some implementations, only the space of *left-deep plans (左深计划)* is considered.
  - Cartesian products avoided in some implementations.

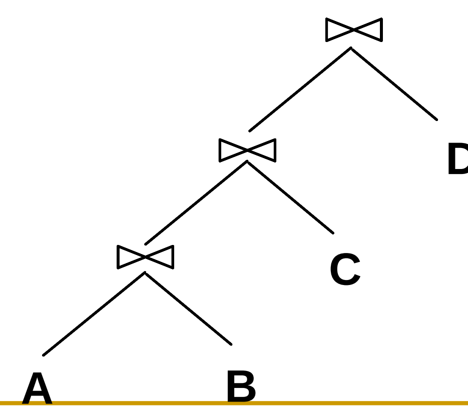
Query Blocks: Units of Optimization 查询块

- Break query into *query blocks*
- Optimized one block at a time
- Uncorrelated nested blocks computed once
- Correlated nested blocks like function calls
  - But sometimes can be “decorrelated”
  - Beyond the scope of introductory course!

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

Outer block　　*Nested block*

- For each block, the plans considered are:
  - All available access methods, for each relation in FROM clause.
  - All *left-deep join trees*
    - right branch always a base table
    - consider all join orders and join methods



Schema for Examples

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

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages. 100 distinct bids.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages. 10 ratings, 40,000 sids.

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
40,000 sids

Translating SQL to Relational Algebra

```
SELECT  S.sid, MIN (R.day)
FROM  Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = “red”
GROUP BY S.sid
HAVING COUNT (*) >= 2
```

For each sailor with at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.

Translating SQL to Relational Algebra

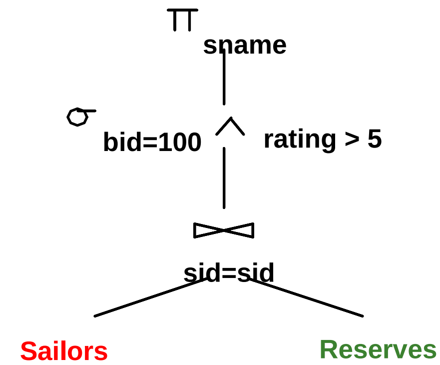
```
SELECT  S.sid, MIN (R.day)
FROM  Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = "red"
GROUP BY S.sid
HAVING COUNT (*) >= 2
```

$$\pi_{\text{S.sid, MIN(R.day)}}\Big(\text{HAVING}_{\text{COUNT(*)>2}}\big(\text{GROUP BY}_{\text{S.Sid}}\big(\sigma_{\text{B.color = "red"}}\big(\text{Sailors} \bowtie \text{Reserves} \bowtie \text{Boats}\big)\big)\big)\Big)$$

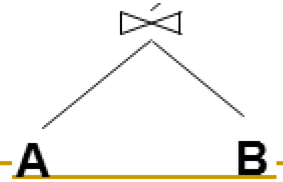
Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- Selections:
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (*cascade-级联*)
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (*commute-交换律*)
- Projections:
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an}(R))\ldots)$ (*cascade*)
- Cartesian Product
  - $(R \times S) \times T \equiv R \times (S \times T)$ (associative-结合律)
  - $R \times S \equiv S \times R$ (commutative)
  - *This means we can do joins in any order.*

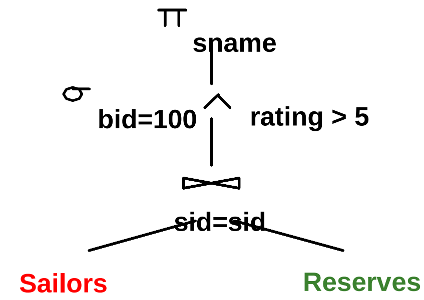
More Equivalences



- Eager projection
  - Can cascade and “push” some projections thru selection
  - Can cascade and “push” some projections below one side of a join
  - Rule of thumb: can project anything not needed “downstream”
- Selection on a cross-product is equivalent to a join.
  - If selection is comparing attributes from each side
- A selection on attributes of R commutes with R $\bowtie$ S.
  - i.e., $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$
  - but only if the selection doesn’t refer to S!



Cost Estimation



- For each plan considered, must estimate total cost:
  - Must estimate *cost* of each operation in plan tree.
    - We've already discussed this for various operators
      - sequential scan, index scan, joins, etc.
    - Depends on input cardinalities(基数).

Cost = [R] + [R]/N *[S]

$$2N(1+\lceil \log_{B-1}\lceil N/B \rceil \rceil)$$

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

- Q: Is "cost" the same as estimated "run time"?

Statistics and Catalogs

- Need infomation on relations and indexes involved.
- *Catalogs* typically contain at least:

Statistic	Meaning
NTuples	# of tuples in a table (cardinality)
NPages	# of disk pages in a table
Low/High	min/max value in a column
Nkeys	# of distinct values in a column
IHeight	the height of an index
INPages	# of disk pages in an index

- Catalogs updated periodically.
- Modern systems do more
  - keep more detailed information on data values, e.g., histograms

Size Estimation and Selectivity

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

- Max output cardinality = product of input cardinalities
- *Selectivity (sel)* associated with each *term*
  - *Book calls selectivity "Reduction Factor" (RF)*
  - |output| / |input|
  - reflects the impact of the *term* in reducing result size.

Result cardinality = Max # tuples * $\prod \text{sel}_i$

Reduction Factor Estimation

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

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

- Term *col=value* (given Nkeys(I) on *col*)

  RF = *1/NKeys(I)*

- Term *col1=col2* (handy for joins too…)

  RF = *1/MAX(NKeys(I1), NKeys(I2))*

- Term *col>value*

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

Implicit assumptions: values are uniformly distributed and *terms* are independent!

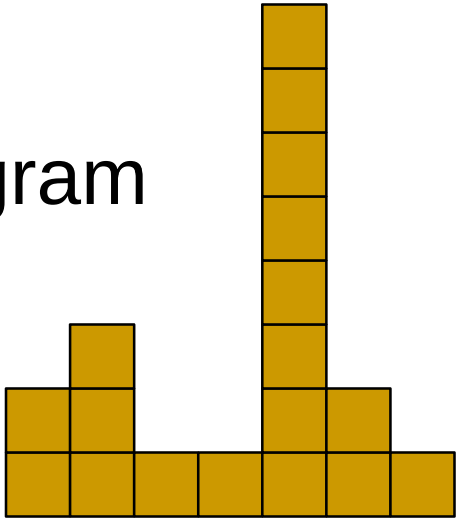
- *Note, if missing the needed stats, assume 1/10!!!*

Reduction Factors & Histograms

- For better estimation, use a histogram

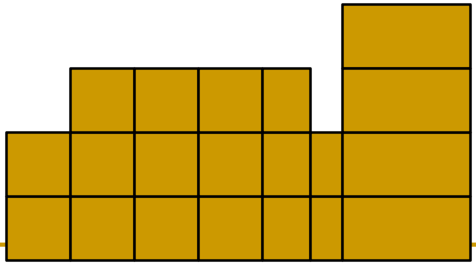
equiwidth

No. of Values	2	3	3	1	8	2	1
Value	0-.99	1-1.99	2-2.99	3-3.99	4-4.99	5-5.99	6-6.99



equidepth

No. of Values	2	3	3	3	3	2	4
Value	0-.99	1-1.99	2-2.99	3-4.05	4.06-4.67	4.68-4.99	5-6.99

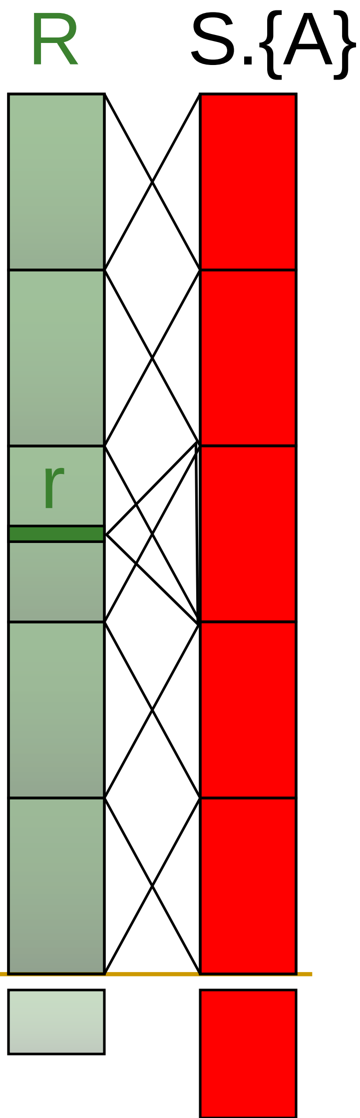


Think Through Estimation for Joins

- Term *col1=col2*
  - RF = *1/MAX(NKeys(I1), NKeys(I2))*

- Q: Given R *join* S, what is result cardinality?
  Two cases:
  1. Key for R, Foreign Key for S
     - A common case, treat it specially! RF = 1/|R|
  2. join on non-key {A}
     - For each r ∈ R, NTuples(S)/NKeys(A,S) result tuples
       so…(NTuples(**R**) * NTuples(**S**)) / NKeys(A,**S**)
     - For each s ∈ S, NTuples(**R**)/NKeys(A,**R**)
       so… (NTuples(**S**) * NTuples(**R**)) / NKeys(A,**R**)

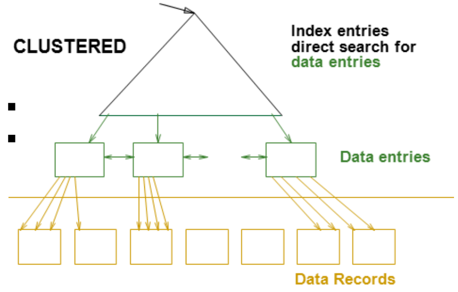
  - If these two estimates differ, take the lower one!
    - Q: Why?



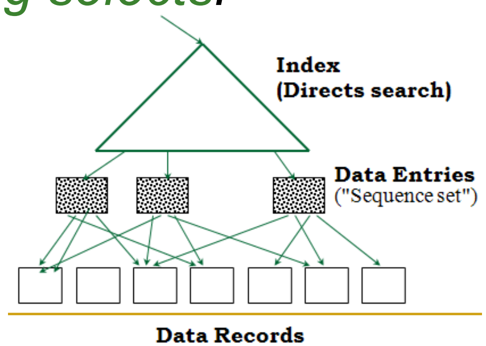
Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans (base case)
  - Multiple-relation plans (induction)
- Single-table queries include selects, projects, and grouping/aggregate operations:
  - Consider each available access path (file scan / index)
    - Choose the one with the least estimated cost
  - Selection/Projection done on the fly
  - Result pipelined into grouping/aggregation

Cost Estimates for Single-Relation Plans



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



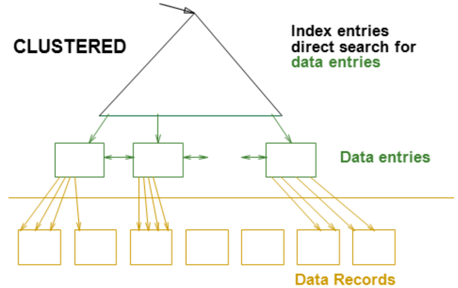
Recall: Must also charge for duplicate elimination if required

Example

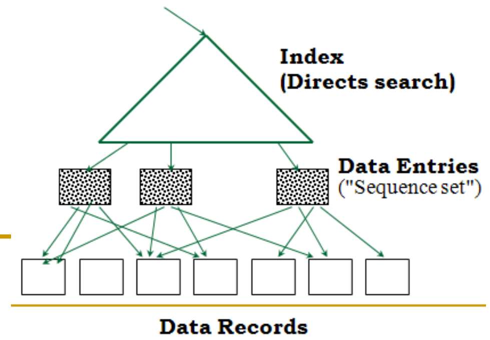
```
SELECT  S.sid
FROM  Sailors S
WHERE  S.rating=8
```

[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages

- If we have an index on *rating*:
  - Cardinality = NTuples(S) * (1/NKeys(I)) = 40000 tuples * (1/10)
  - Clustered index: (NPages(I)+NPages(S))* (1/NKeys(I))
    = (50+500) * (1/10) = 55 pages are retrieved. (This is the ***cost***.)
  - Unclustered index: (NPages(I)+NTuples(S))* (1/NKeys(I))
    = (50+40000) * (1/10) = 4005 pages are retrieved.



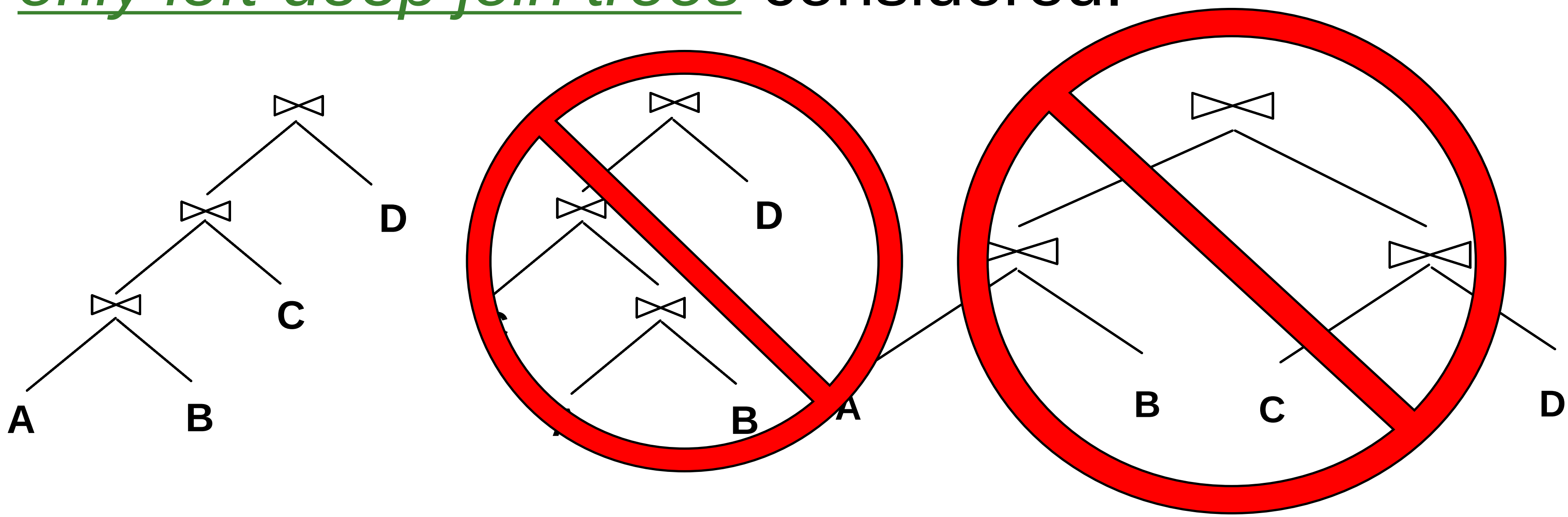
- If we have an index on *sid*:
  - Would have to retrieve all tuples/pages.
    - With a clustered index, the cost is 50+500,
    - with unclustered index, 50+40000.



- Doing a file scan:
  - We retrieve all file pages (500).

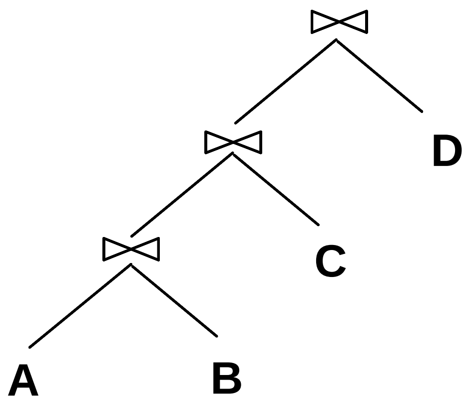
Queries Over Multiple Relations

- A System R heuristic:
  only left-deep join trees considered.



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

Enumeration of Left-Deep Plans



- Left-deep plans differ in
  - the order of relations
  - the access method for each relation
  - the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass i: Find best way to join result of an (i -1)-relation plan (as outer) to the i'th relation. (i between 2 and N.)
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

The Dynamic Programming Table

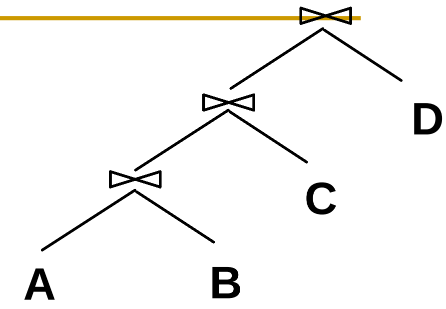
[R]=1000, p_R=100
100 boats
[S]=500, p_S=80.
10 ratings
5 buffer pages

Subset of tables in FROM clause	Interesting-order columns	Best plan	Cost
{R, S}	**<none>**	**hashjoin(R,S)**	**4500**
{R, S}	**<R.a, S.b>**	**sortmerge(R,S)**	**7500**

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 *yet-to-be-added* (downstream) joins

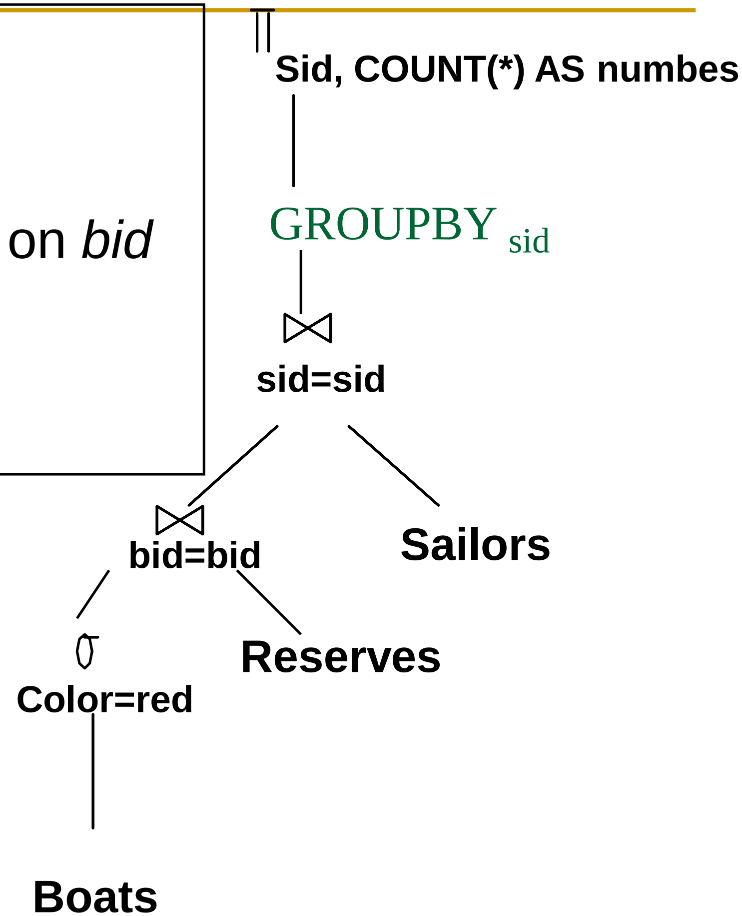
Enumeration of Plans (Contd.)



- Match an i -1 way plan with another table *only if*
  a) there is a join condition between them, *or*
  b) all predicates in WHERE have been used up.
    - i.e., avoid Cartesian products if possible.
- ORDER BY, GROUP BY, aggregates etc. handled as a final step
  - via `interestingly ordered' plan if chosen (free!)
  - or via an additional sort/hash operator
- Despite pruning, this is exponential in #tables.

Example

<u>Sailors:</u>
Hash, B+ on *sid*
<u>Reserves:</u>
Clustered B+ tree on *bid*
B+ on *sid*
<u>Boats</u>
B+ on *color*



Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
  AND B.color = “red”
GROUP BY S.sid

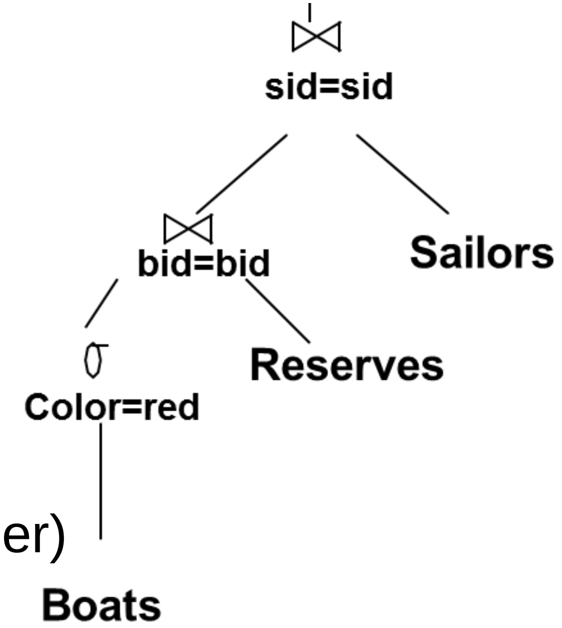
- **Pass1: Best plan(s) for accessing each relation**
  - Reserves, Sailors: File Scan
  - Q: What about Clustered B+ on Reserves.bid???
  - Boats: B+ tree on color

Pass 1

- Find best plan for each relation in isolation:
  - Reserves, Sailors: File Scan
  - Boats: B+ tree on color

Pass 2

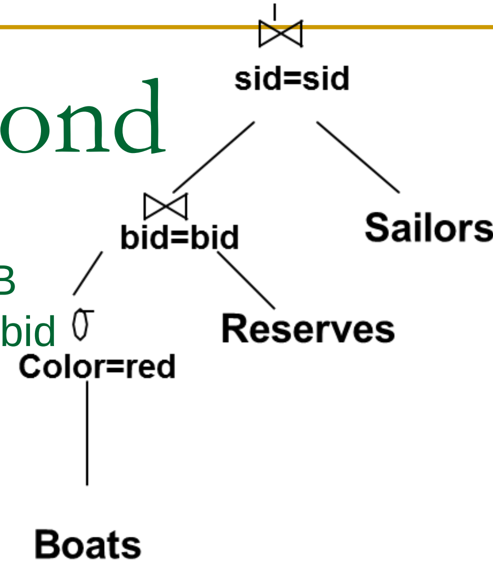
- For each plan in pass 1, generate plans joining another relation as the inner, using all join methods (and matching inner access methods)
  - File Scan Reserves (outer) with Boats (inner)
  - File Scan Reserves (outer) with Sailors (inner)
  - File Scan Sailors (outer) with Boats (inner)
  - File Scan Sailors (outer) with Reserves (inner)
  - Boats Btree on color with Sailors (inner)
  - Boats Btree on color with Reserves (inner)
- Retain cheapest plan for each (pair of relations, order)



FROM Sailors S, Reserves R, Boats B

Pass 3 and beyond

Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
  AND B.color = “red”
GROUP BY S.sid



<u>Sailors:</u>
  Hash, B+ on *sid*
<u>Reserves:</u>
  Clustered B+ tree on *bid*
  B+ on *sid*
<u>Boats</u>
  B+ on *color*

- Using Pass 2 plans as outer relations, generate plans for the next join
  - E.g. Boats B+-tree on color with Reserves (bid) (sortmerge)
    inner Sailors (B-tree sid) sort-merge
- Then, add cost for groupby/aggregate:
  - This is the cost to sort the result by sid, *unless it has already been sorted by a previous operator.*
- Then, choose the cheapest plan

FROM Sailors S, Reserves R, Boats B

Summary

- Optimization is the reason for the lasting power of the relational system
- But it is primitive in some ways
- New areas: many!
  - Smarter summary statistics (fancy histograms and “sketches”)
  - Auto-tuning statistics,
  - Adaptive runtime re-optimization (e.g. *eddies*),
  - Multi-query optimization,
  - And parallel scheduling issues, etc.

Summary

- 要求:
  - 理解查询树、执行计划树、关系代数等价规则、左深计划树和完全流水线计划树等概念
  - 深刻理解选择条件的选择性/缩减因子，进而能够估算结果集大小
  - 能够估算执行计划的开销
  - 理解左深计划的遍历算法
    - 若只有1、2关系，则能够找出最优计划