CAS CS 460/660
Introduction to Database Systems

Query Optimization

Review

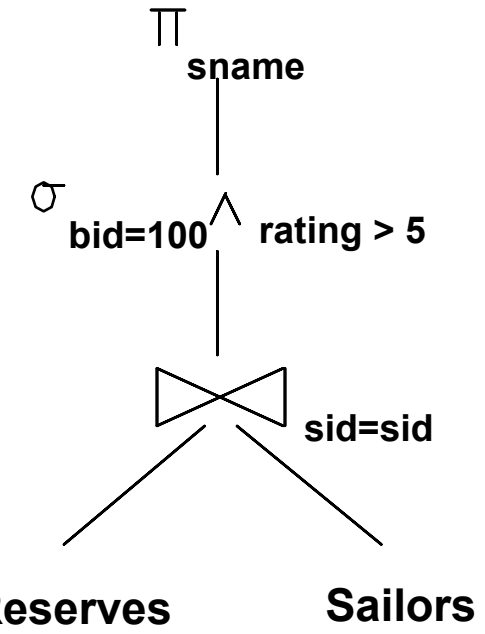
- Implementation of Relational Operations as Iterators
  - Focus largely on External algorithms (sorting/hashing)
- Choices depend on indexes, memory, stats,…
- Joins
  - Blocked nested loops:
    - simple, exploits extra memory
  - Indexed nested loops:
    - best if 1 rel small and one indexed
  - Sort/Merge Join
    - good with small amount of memory, bad with duplicates
  - Hash Join
    - fast (enough memory), bad with skewed data
    - Relatively easy to parallelize
- Sort and Hash-Based Aggs and DupElim

Query Optimization Overview

- Query can be converted to relational algebra
- Rel. Algebra converted to tree, joins as branches
- Each operator has implementation choices
- Operators can also be applied in different order!

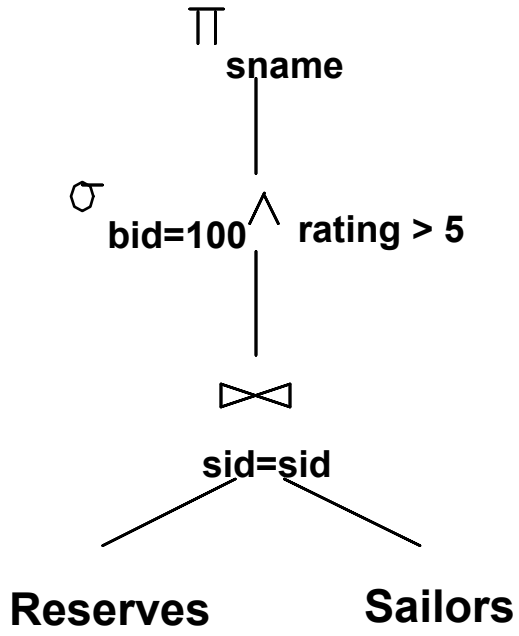
```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

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



Iterator Interface (pull from the top)

- Recall:



•Relational operators at nodes support uniform *iterator* interface:

Open( ), get_next( ), close( )

•Unary Ops – On Open() call Open() on child.

•Binary Ops – call Open() on left child then on right.

•By convention, outer is on left.

Alternative is pipelining (i.e. a “push”-based approach).

Can combine push & pull using special operators.

Query Optimization Overview (cont)

- *Logical Plan:* *Tree of R.A. ops*
- *Physical Plan:* *Tree of R.A. ops, with choice of algorithm for each operator.*

- Two main issues:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?

- Ideally: Want to find best plan.

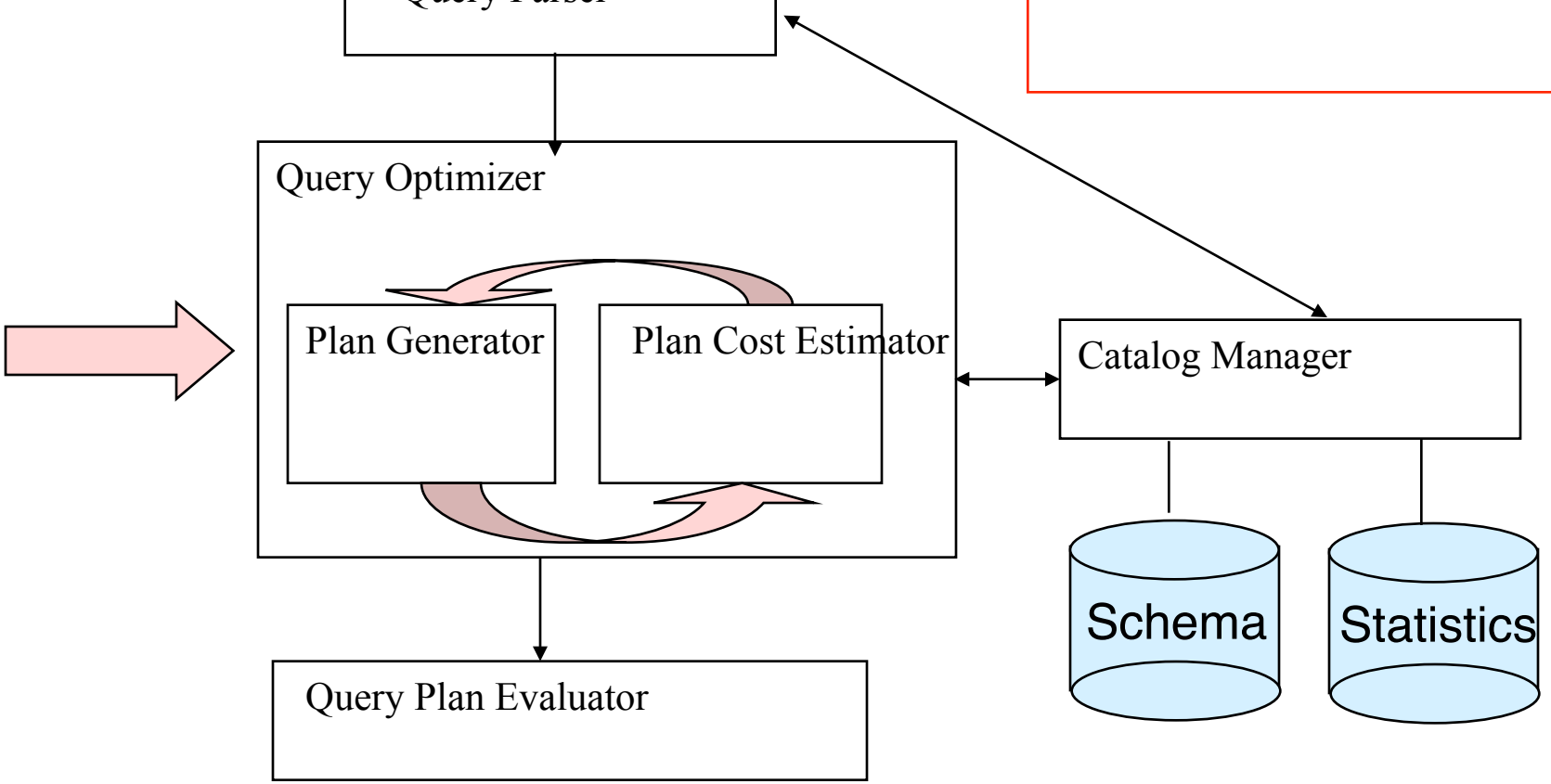
- Reality: Avoid worst plans!

Cost-based Query Sub-System

Queries

```
Select *
From Blah B
Where B.blah = blah
```

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



Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)

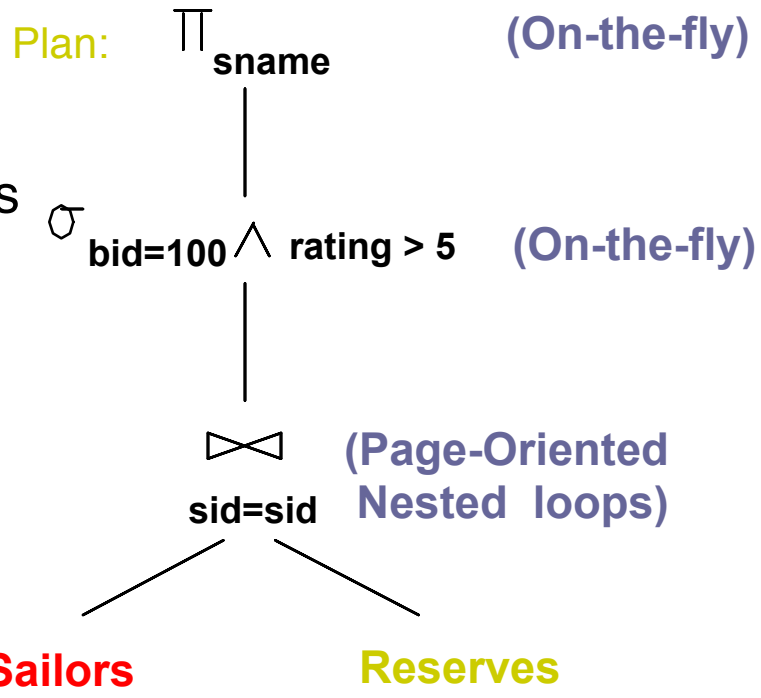
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- As seen in previous lectures…
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Let's say there are 100 boats.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Let's say there are 10 different ratings.
- Assume we have 5 pages in our buffer pool.

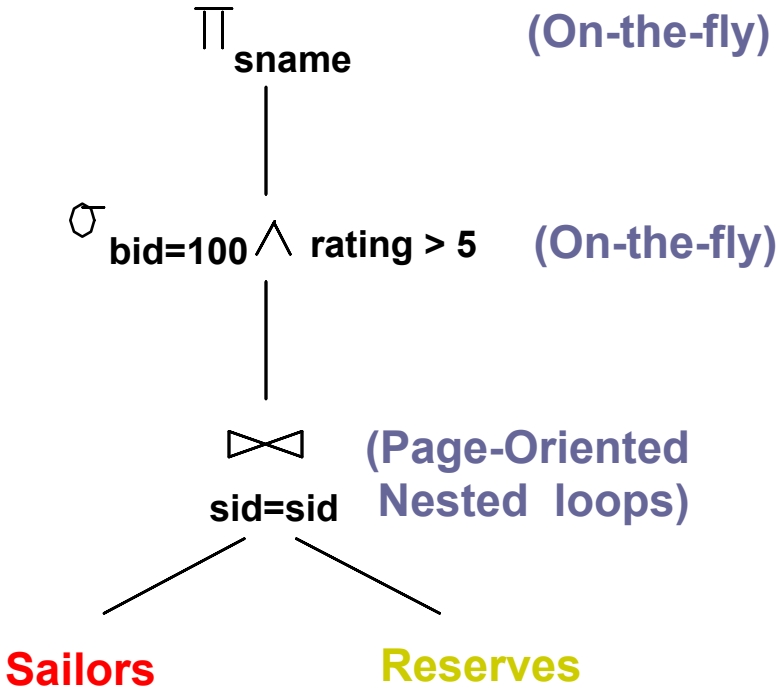
Motivating Example

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

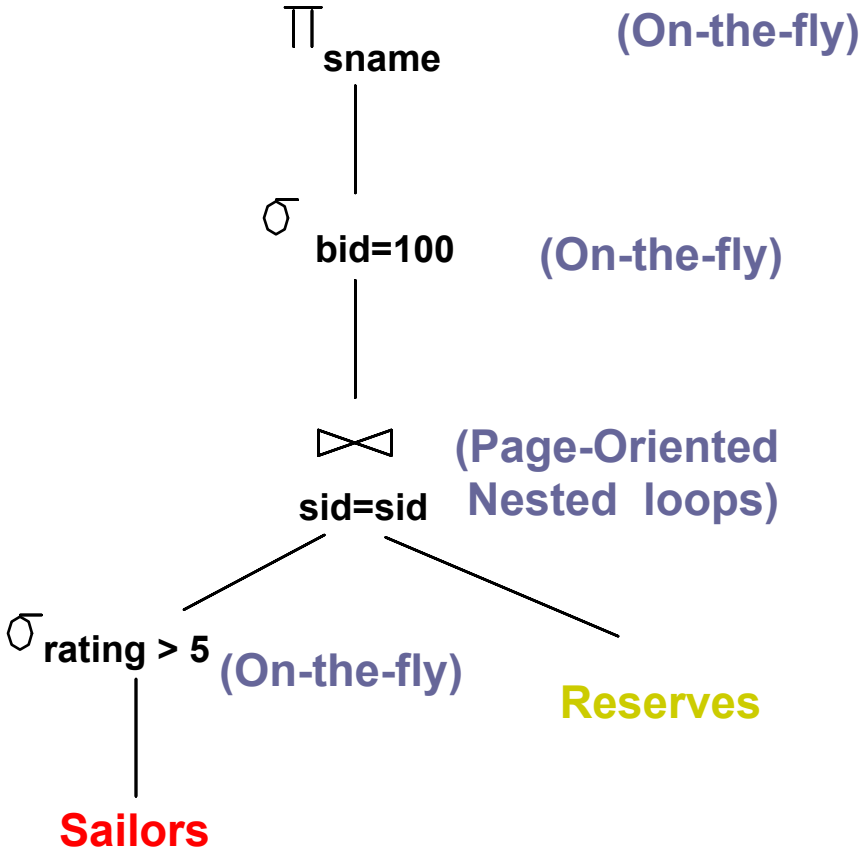
- Cost: 500+500*1000 I/Os
- By no means the worst plan!
- Misses several opportunities: selections could have been `pushed' earlier, no use is made of any available indexes, etc.
- *Goal of optimization:* To find more efficient plans that compute the same answer.



Alternative Plans – Push Selects (No Indexes)

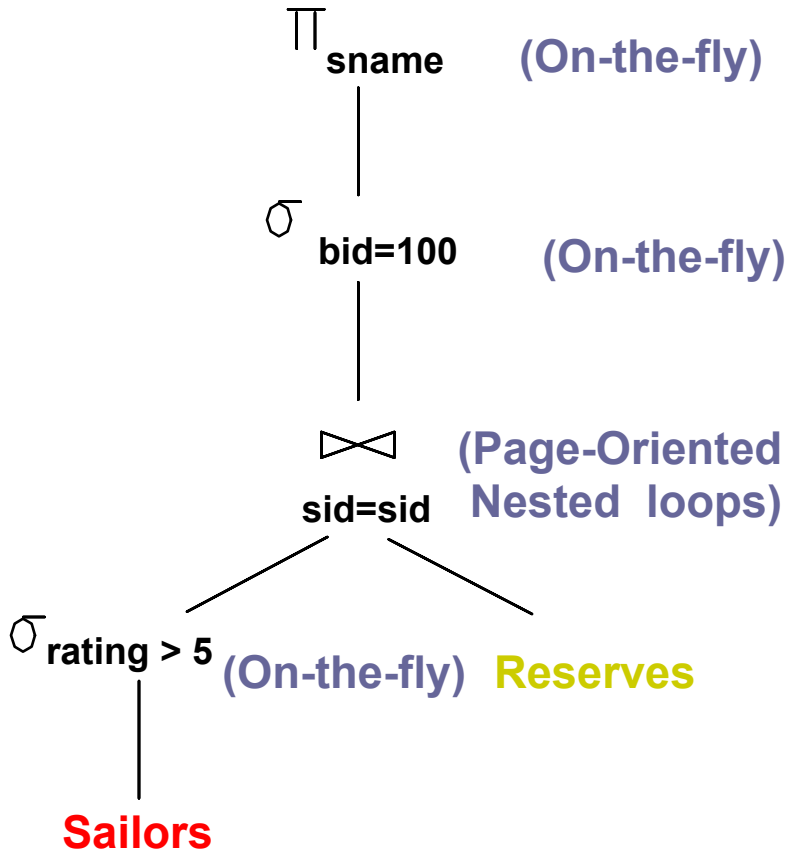


500,500 IOs

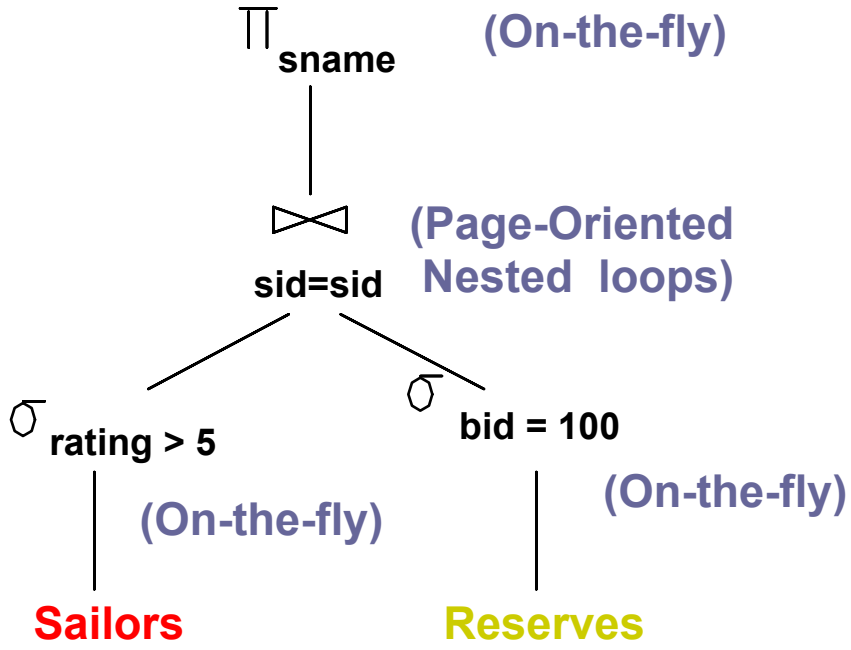


250,500 IOs

Alternative Plans – Push Selects (No Indexes)

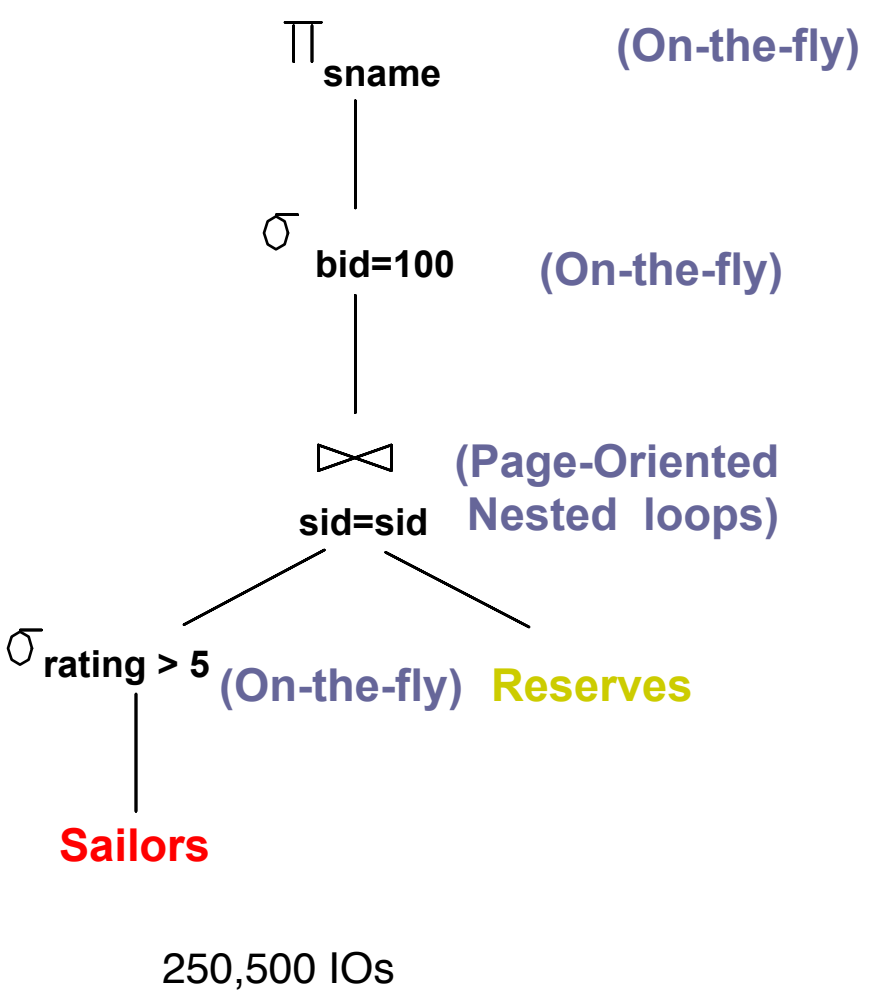


250,500 IOs

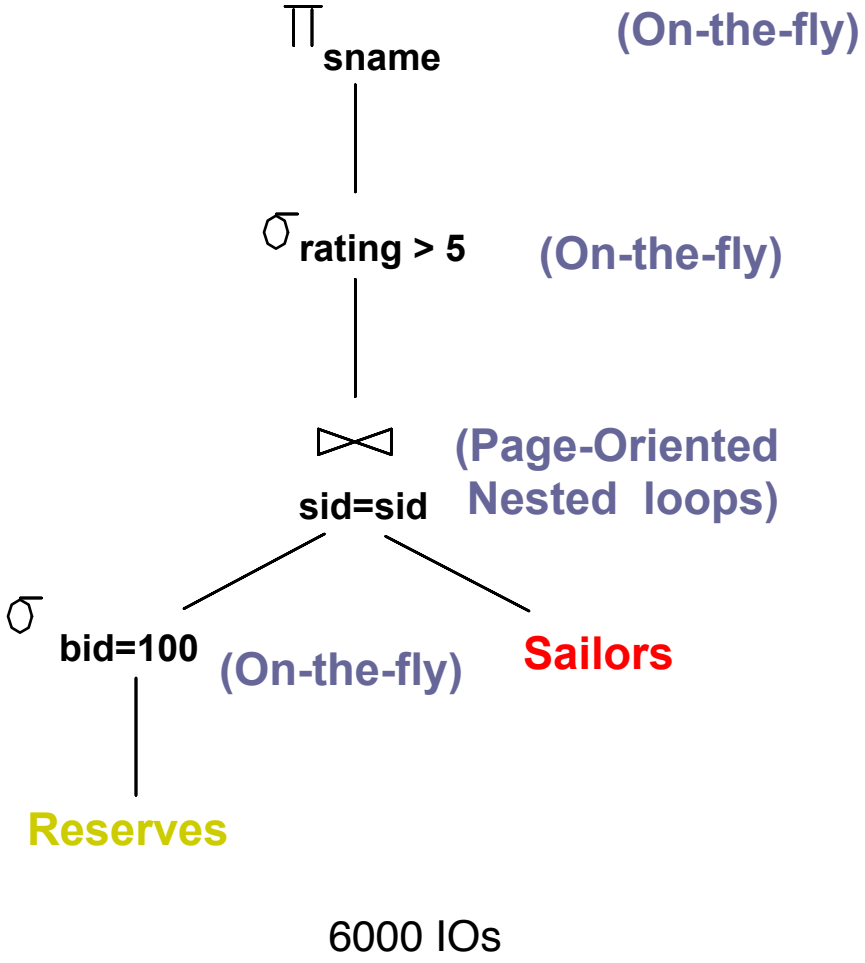


250,500 IOs

Alternative Plans – Push Selects (No Indexes)

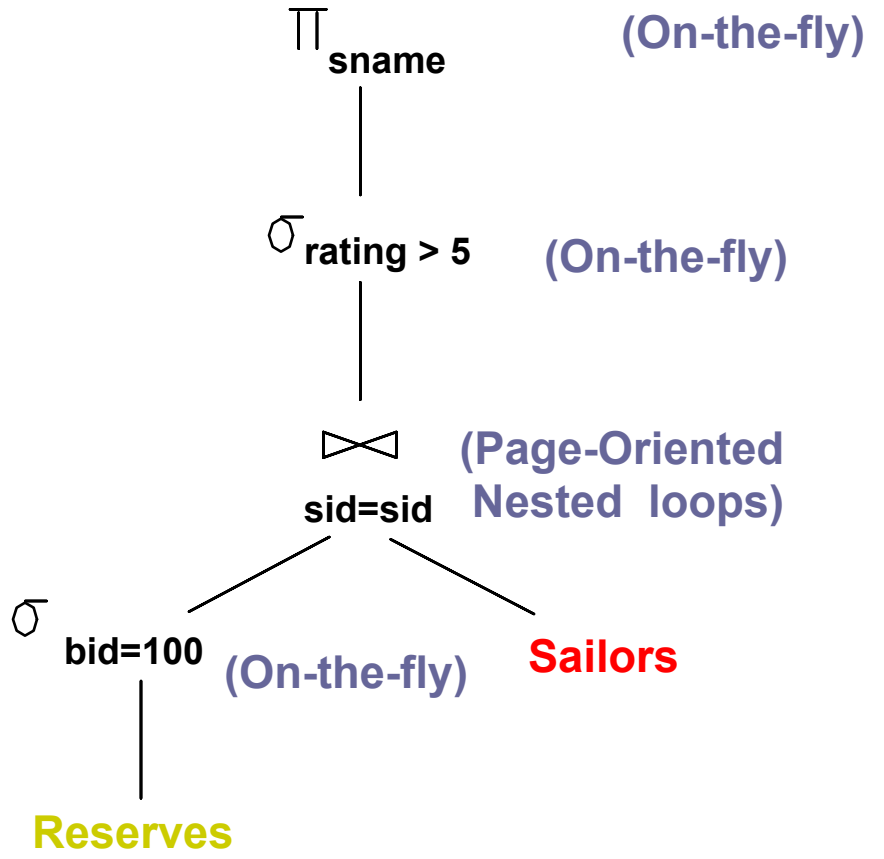


250,500 IOs

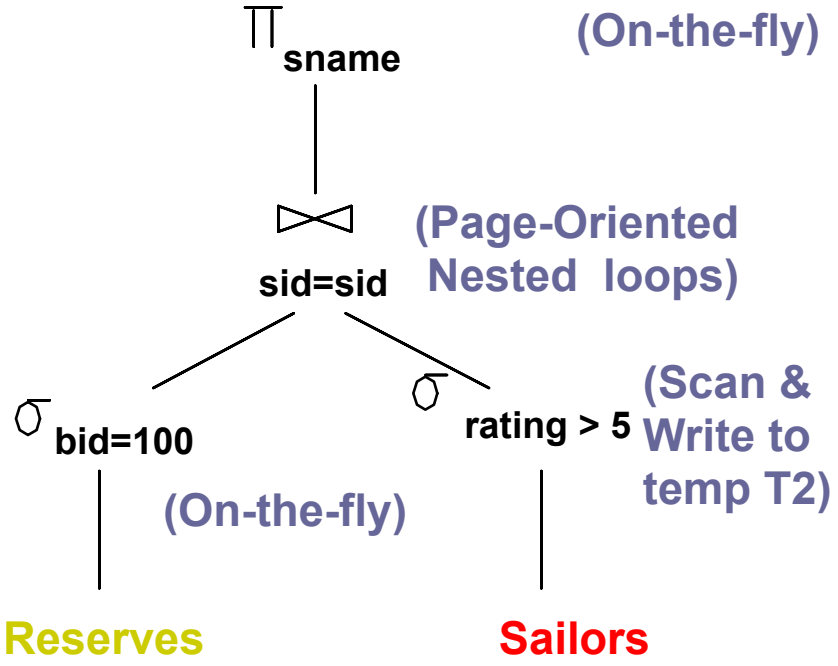


6000 IOs

Alternative Plans – Push Selects (No Indexes)

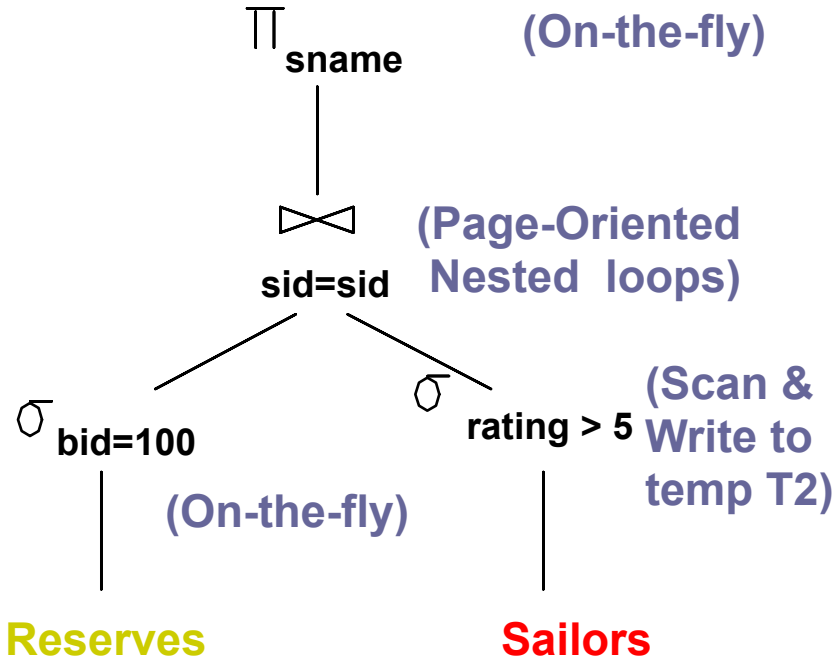


6000 IOs

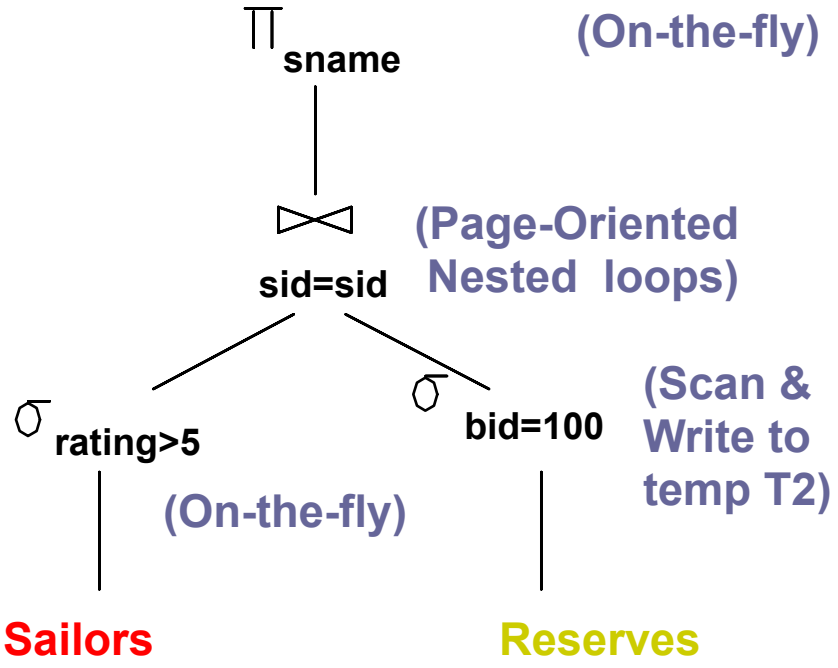


4250 IOs
1000 + 500+ 250 + (10 * 250)

Alternative Plans – Push Selects (No Indexes)

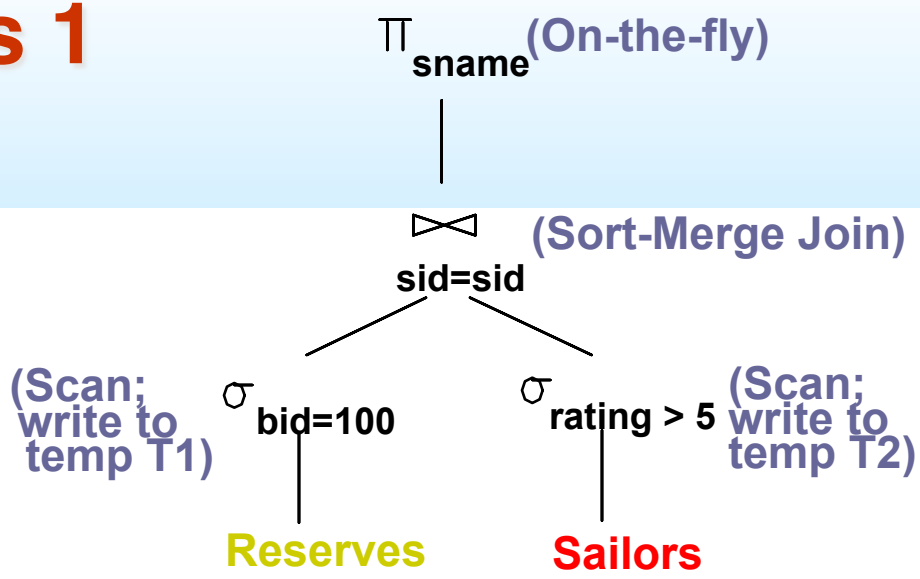


4250 IOs



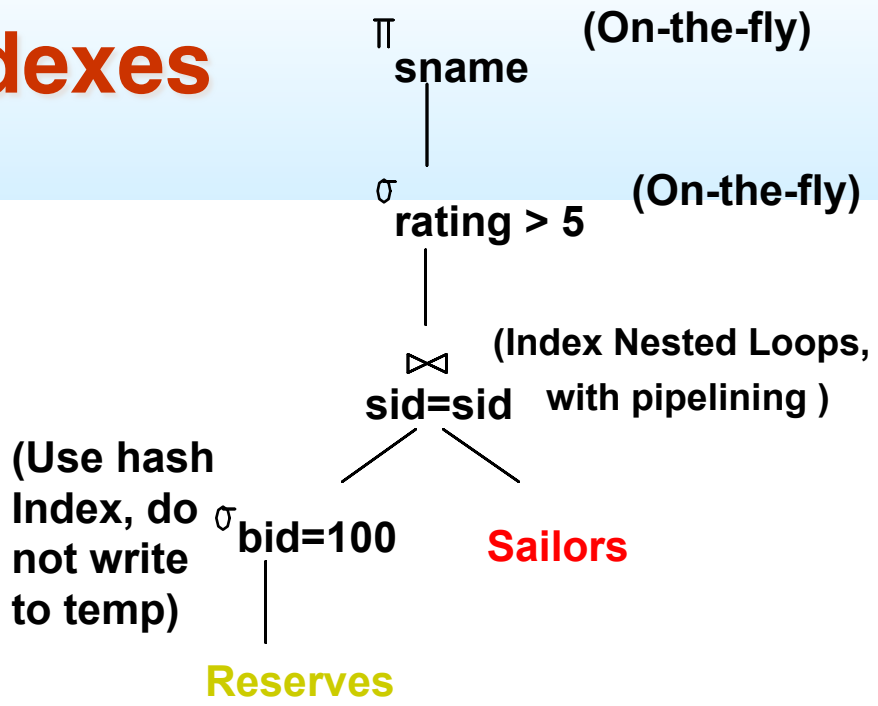
4010 IOs
500 + 1000 +10 +(250 *10)

Alternative Plans 1 (No Indexes)



- *Main difference: Sort Merge Join*
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250)
  - Total: 4060 page I/Os. (note: T2 sort takes 4 passes with B=5)
- 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.

Alt Plan 2: Indexes



- With clustered hash index on *bid* of Reserves, we get 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- INL with outer not materialized.

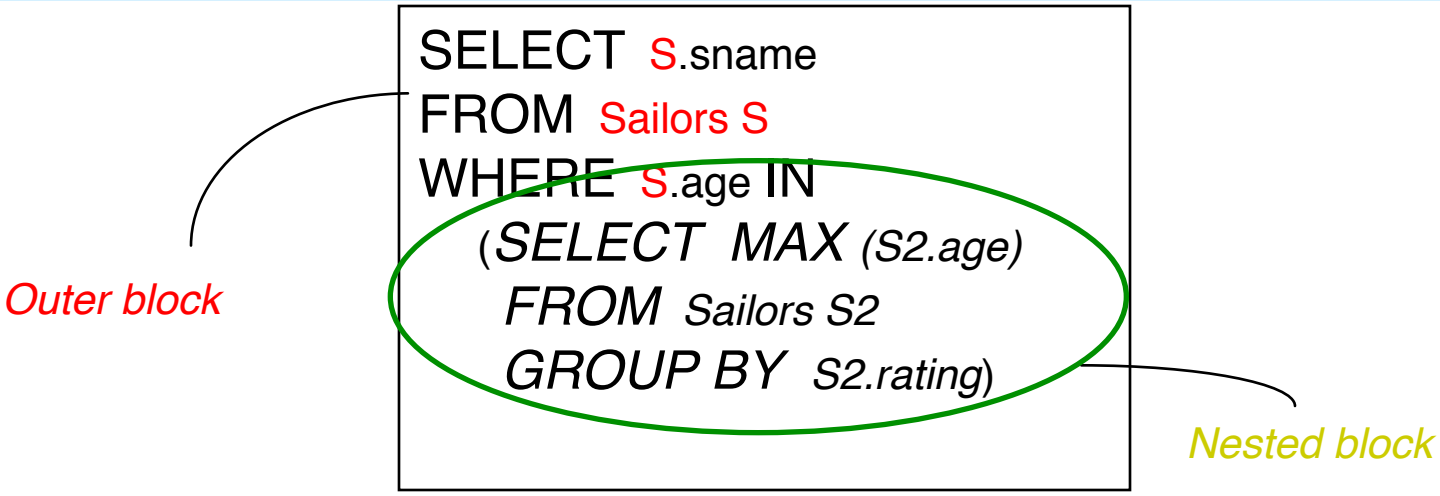
– Projecting out unnecessary fields from outer doesn't help.

- Join column *sid* is a key for Sailors.
  At most one matching tuple, unclustered index on *sid* OK.
- Decision not to push *rating>5* before the join is based on availability of *sid* index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); then, for each, must get matching Sailors tuple (1000*1.2); total 1210 I/Os.

What is needed for optimization?

- Iterator Interface
- Cost Estimation
- Statistics and Catalogs
- Size Estimation and Reduction Factors

Query Blocks: Units of Optimization



- An SQL query is parsed into a collection of *query blocks*, and these are optimized one block at a time.

- Inner blocks are usually treated as subroutines
- Computed:
  - once per query (for uncorrelated sub-queries)
  - or once per outer tuple (for correlated sub-queries)

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”
AND S.rating = ( SELECT MAX (S2.rating) FROM Sailors S2)
GROUP BY S.sid
HAVING COUNT (*) >= 2
```

For each sailor with the highest rating (over all sailors), and 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"
AND S.rating = ( SELECT MAX (S2.rating) FROM Sailors S2)
GROUP BY S.sid
HAVING COUNT (*) >= 2
```

Inner Block

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

Relational Algebra Equivalences

- Allow us to choose different operator orders and to `push' selections and projections ahead of joins.
- *Selections*:

$$\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R)) \qquad \text{(Cascade)}$$

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R)) \qquad \text{(Commute)}$$

- *Projections:*

$$\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R))) \qquad \text{(Cascade)}$$

(if an includes an-1 includes… a1)

- *Joins:*

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

$$(R \bowtie S) \equiv (S \bowtie R) \qquad \text{(Commute)}$$

These two mean we can do joins in any order.