ICS 421 Spring 2010

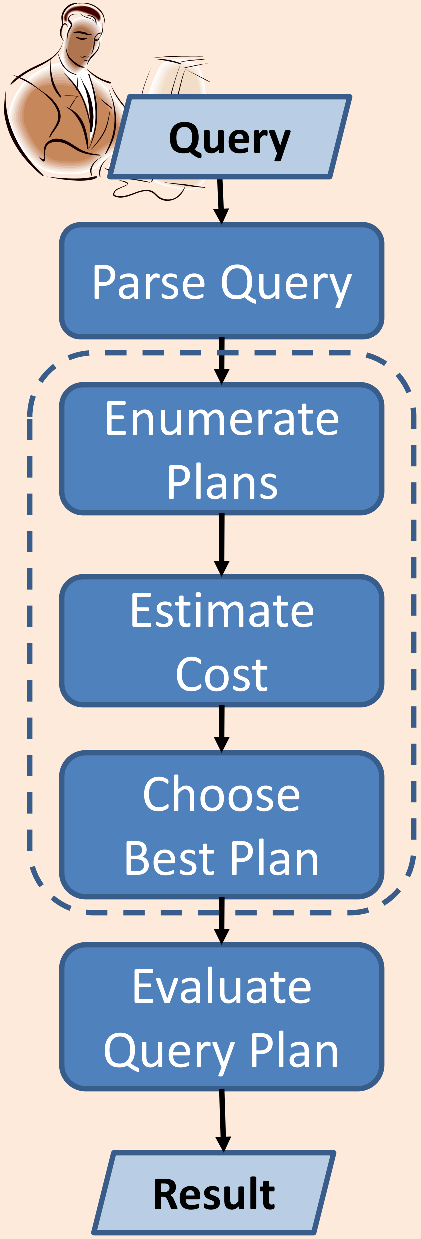
Relational Query Optimization

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Query Optimization

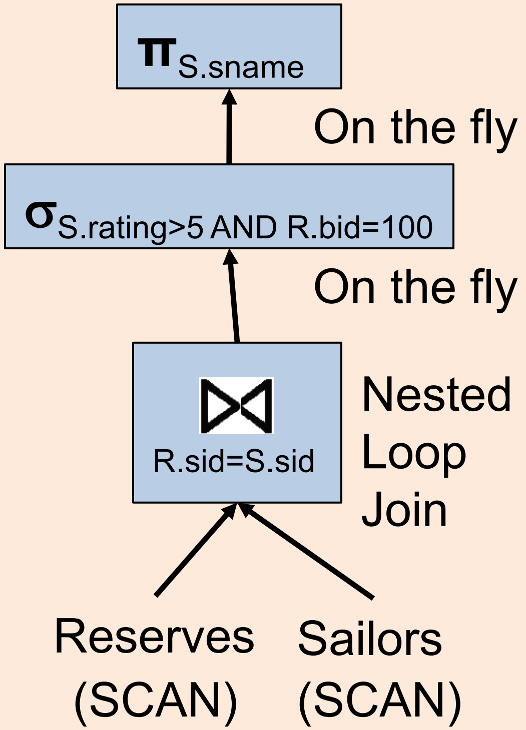


- Two main issues:
  - For a given query, what plans are considered?
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan. Practically: Avoid worst plans!
- System R Optimizer:
  - Most widely used currently; works well for < 10 joins.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Only the space of *left-deep plans* is considered.
  - Cartesian products avoided.

Example

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

Reserves	40 bytes/tuple	100 tuples/page	1000 pages
Sailors	50 bytes/tuple	80 tuples/page	500 pages



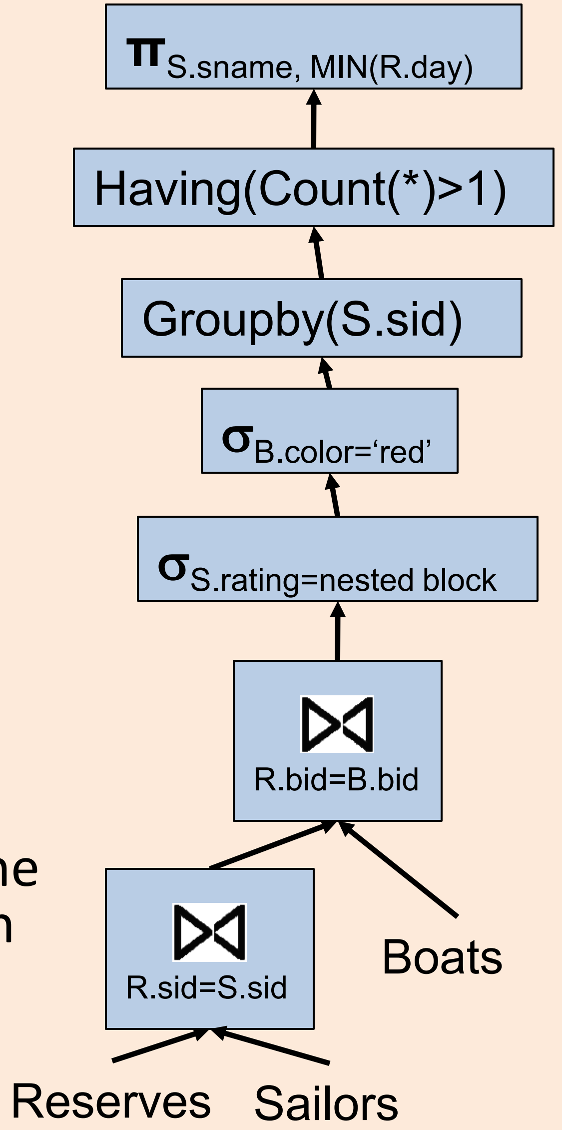
- Nested Loop Join cost 1K+ 100K*500
- On the fly selection and project does not incur any disk access.
- Total disk access = 500001K (worst case)

What about complex queries ?

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(*) > 1

Outer Block

Nested Block



- For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All left-deep join trees (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods.)

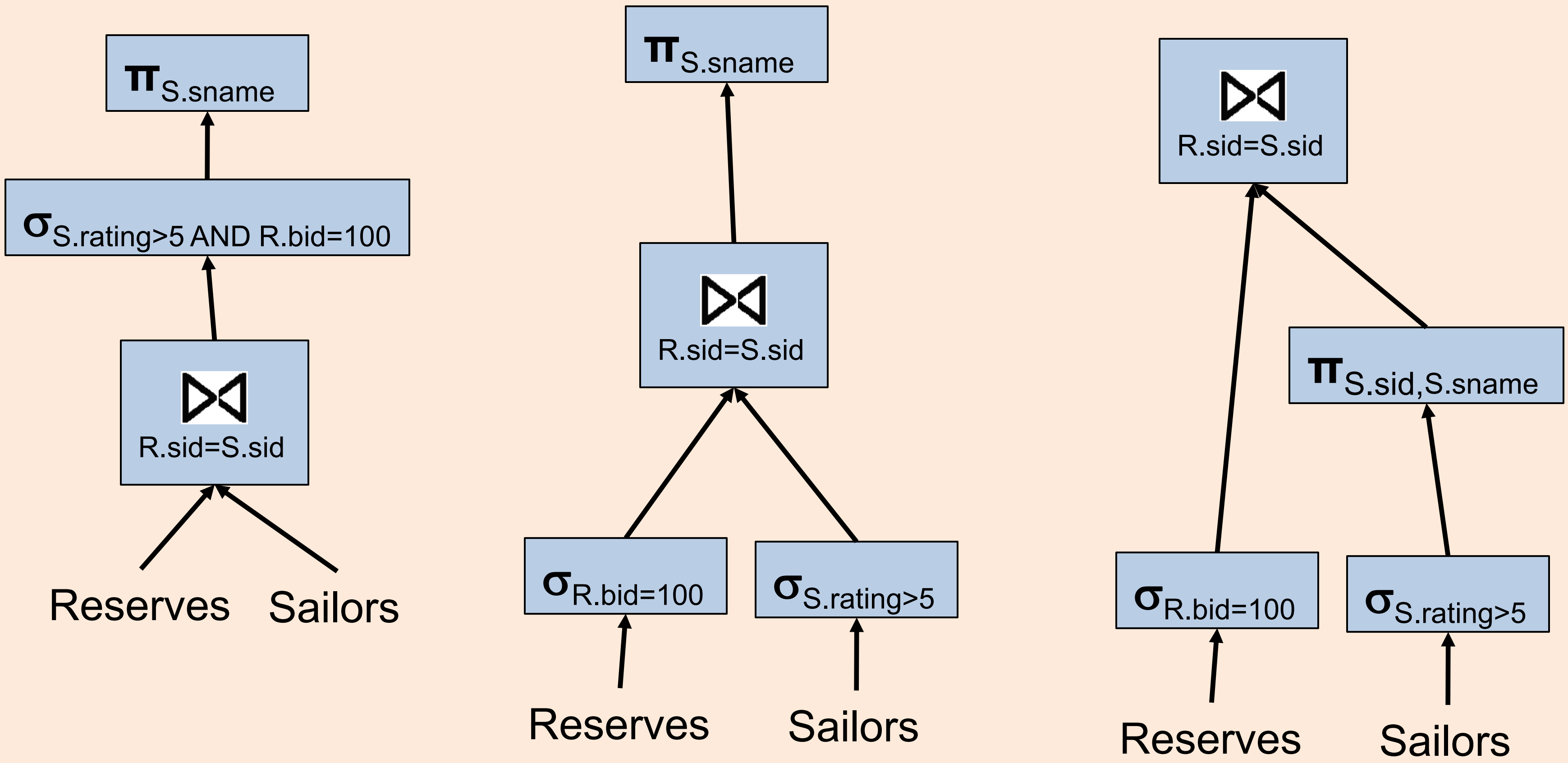
RA Equivalences

- **Selections**
  - Cascade: $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R) \ldots)$
  - Commute: $\sigma_{c1}(\sigma_{cn}(R)) \equiv \sigma_{cn}(\sigma_{c1}(R))$
- **Projections**
  - Cascade: $\pi_{c1}(R) \equiv \pi_{c1}(\ldots \pi_{cn}(R) \ldots)$, c1 subset ci, i>1
- **Joins**
  - Associativity: R join (S join T) $\equiv$ (R join S) join T
  - Commutative: R join S $\equiv$ S join R
  - Definition: R join S $\equiv \sigma_{R.col=S.col}(R \times S)$

More equivalences

- Commutability between projection & selection
  - $\pi_{c1,\ldots cn}(\sigma_{predicate}(S)) \equiv \sigma_{predicate}(\pi_{c1,\ldots cn}(S))$ iff predicate only uses c1,…,cn
- Commutability between selection & join (predicate pushdown)
  - $\sigma_{predicate}(R \text{ join } S) \equiv (\sigma_{predicate}(R)) \text{ join } S$ iff predicate only uses attributes from R
- Commutability between projection & join
  - $\pi_{c1,..,cn}(R \text{ join}_{cr=cs} S) \equiv (\pi_{c1,..,cn,cr}(R)) \text{ join}_{cr=cs} S$

Example: Using Equivalences



Cost Estimation

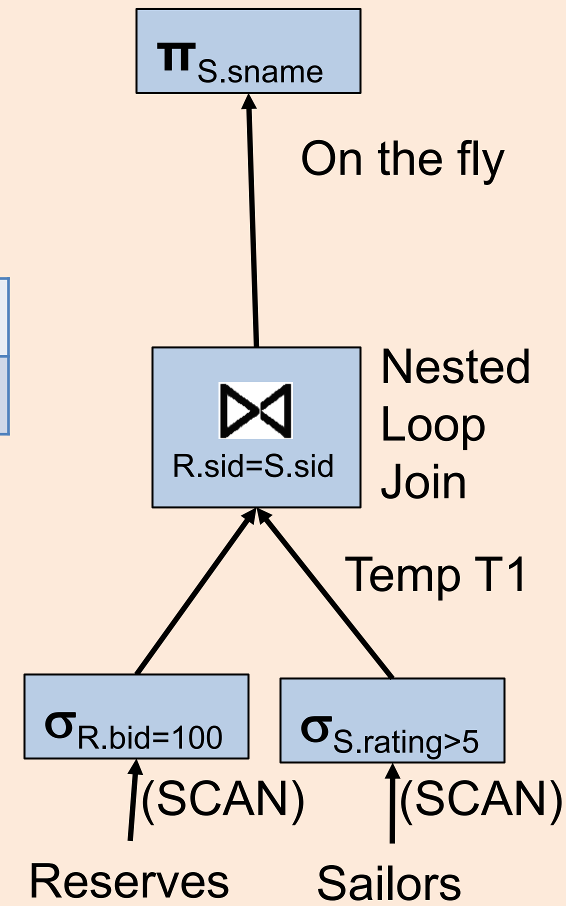
- Obvious inefficient plans are pruned during enumeration. Eg. Predicate pushdown etc.
- For each plan considered,
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Example: Predicate Pushdown

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

10% 50%

Reserves	40 bytes/tuple	100 tuples/page	1000 pages
Sailors	50 bytes/tuple	80 tuples/page	500 pages



- Nested Loop Join requires materializing the inner table as T1.
- With 50% selectivity, T1 has 250 pages
- With 10% selectivity, outer "table" in join has 10K tuples
- Disk accesses for scans = 1000 + 500
- Writing T1 = 250
- NLJoin = 10K * 250
- Total disk access = 2500.175 K (worst case)

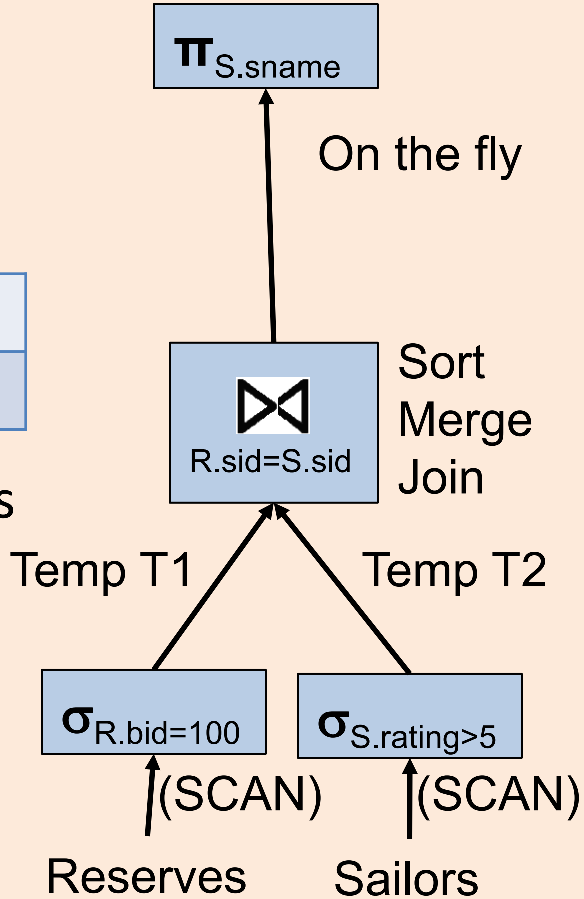
What happens if we make the left leg the inner table of the join ?

Example: Sort Merge Join

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

10% (R.bid=100) **50%** (S.rating>5)

Reserves	40 bytes/tuple	100 tuples/page	1000 pages
Sailors	50 bytes/tuple	80 tuples/page	500 pages



- Sort Merge Join requires materializing both legs for sorting.
- With 10% selectivity, T1 has 100 pages
- With 50% selectivity, T2 has 250 pages
- Disk accesses for scans = 1000 + 500
- Writing T1 & T2 = 100 + 250
- Sort Merge Join = 100 log 100 + 250 log 250 + 100+250 (assume 10 way merge sort)
- Total disk access = 52.8 K

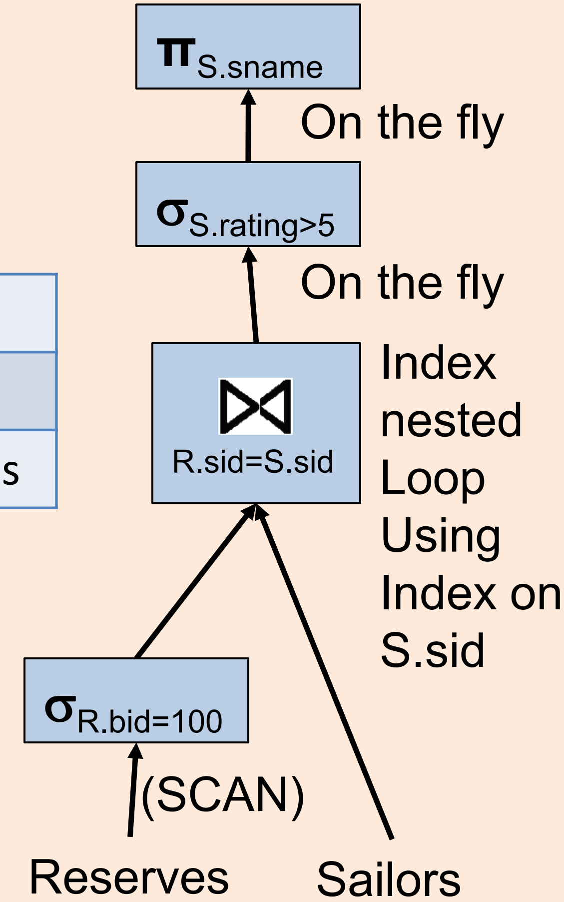
What happens if we make the left leg the inner table of the join ?

Example: Index Nested Loop Join

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

10% **50%**

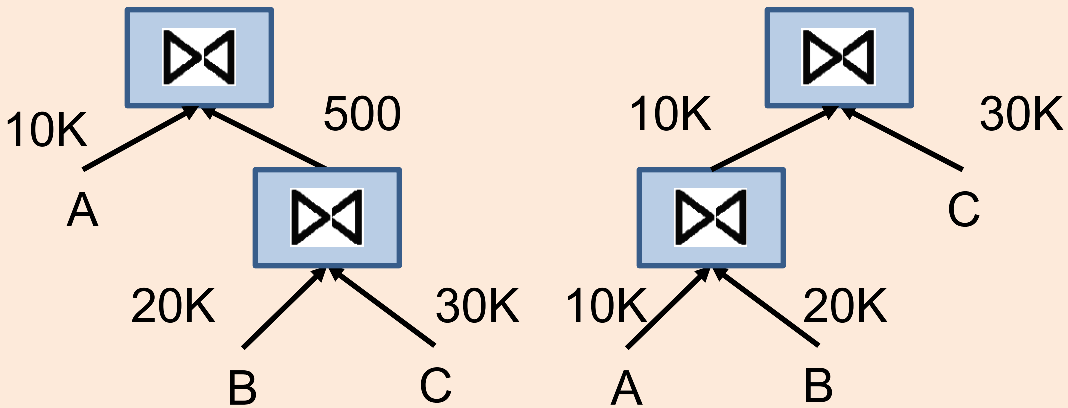
Reserves	40 bytes/tuple	100 tuples/page	1000 pages
Sailors	50 bytes/tuple	80 tuples/page	500 pages
Index(S.sid)			200 leaf pages



- With 10% selectivity, selection on R has 10K tuples
- Disk accesses for scan = 1000
- Index Nested Loop Join = 10K*(1 + $\log_{10} 200$) = 33K
- Total disk access = 34 K

What happens if we make the left leg the inner table of the join ?

Join Ordering



Relations	Tuples	Pages
A	10K	1000
B	20K	2000
C	30K	3000
A join B	10K	1000
B join C	1K	100

- Independent of what join algorithm is chosen, the order in which joins are perform affects the performance.
- Rule of thumb: do the most "selective" join first
- In practice, left deep trees (eg. the right one above) are preferred --- why ?

How to estimate the selectivity & cardinality ?

$\sigma_{col=value}$

- Arbitrary constant 10%
- 1 / Number of distinct values in the column
- 1 / Number of keys in Index(col)

$\sigma_{col>value}$

- Arbitrary constant of 50% if non numeric
- (High Key – value)/(High Key – Low Key)

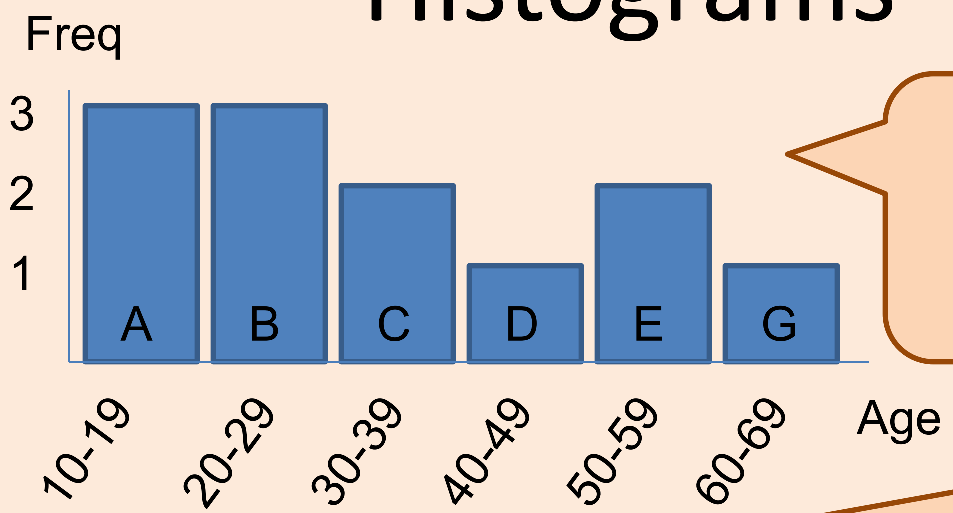
$\sigma_{R.col=S.col}$

- Join result size
- Arbitrary constant 10%
- 1/MAX(Nkeys(Index(R.col)), Nkeys(Index(S.col))

Can we do better ?

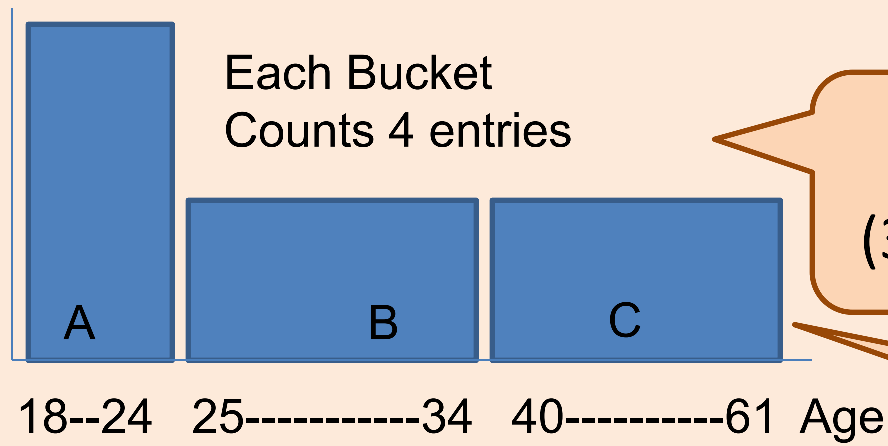
Histograms

Age
18
18
19
24
25
29
30
34
40
50
58
61



$\sigma_{age>45}$:
0.5*f(D) + f(E) + f(G)

Equi-width buckets



$\sigma_{age>30}$:
(34-30)/(34-25)*B + C

Equi-depth buckets

Statistics Collection in DBMS

- Page size
- Data Statistics:
  - Record size -> number of records per data page
  - Cardinality of relations (including temporary tables)
  - Selectivity of selection operator on different columns of a relation
- (Tree) Index Statistics
  - number of leaf pages, index entries
  - Height
- Statistics collection is user triggered
  - DB2: RUNSTATS ON TABLE mytable AND INDEXES ALL

What about the parallel/distributed case?

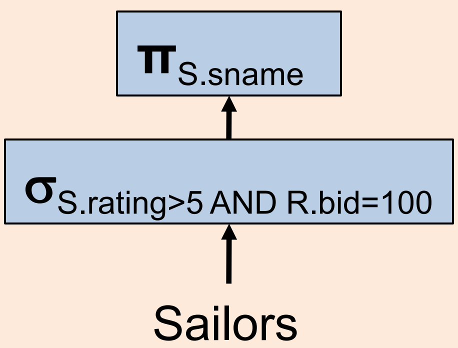
- QEP enumeration/rewrite
  - Main “trick” is expressing a horizontally fragmented table as a union of fragments in RA
  - Push the union up. Conversely push the $\sigma,\pi,\times$ down.
  - Eliminate sub-trees that return empty results.
- Cost estimation takes into account communication costs.

SELECT S.sname
FROM Sailors S
WHERE S.rating>5

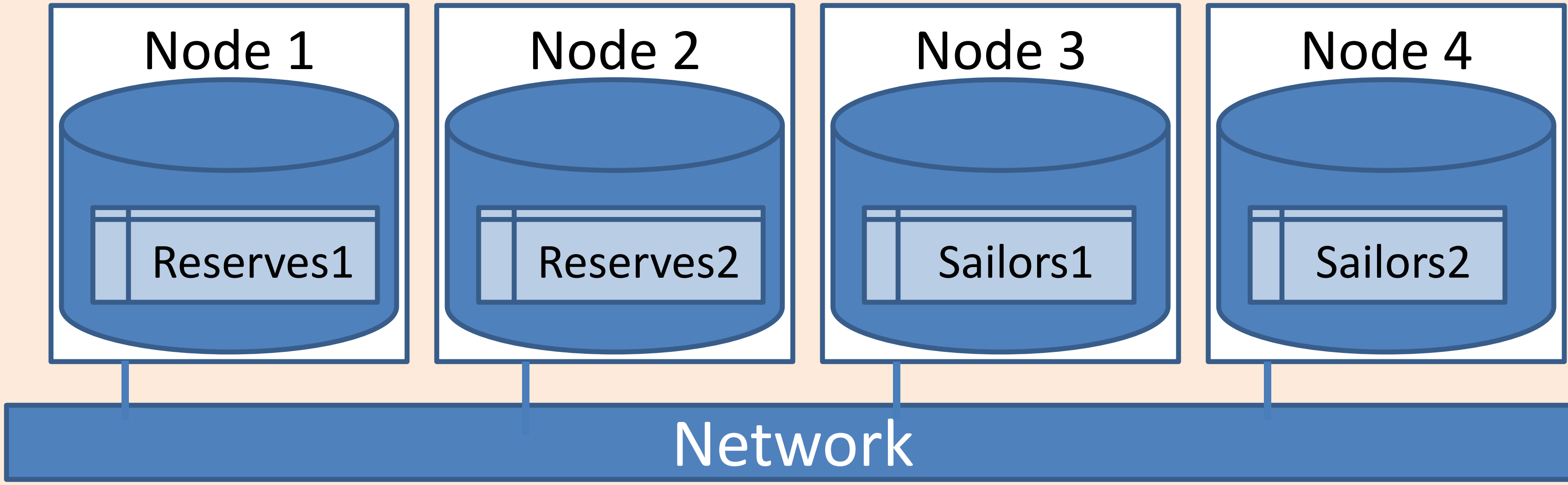
Sailors partitioned by ranges [0,5),[5,10] on rating

$$\pi_{S.sname}(\sigma_{S.rating>5}\ S)$$
$$= \pi_{S.sname}(\sigma_{S.rating>5}\ (S1 \cup S2))$$
$$= \pi_{S.sname}((\sigma_{S.rating>5}\ S1) \cup (\sigma_{S.rating>5}\ S2))$$
$$= (\pi_{S.sname}\sigma_{S.rating>5}\ S1) \cup (\pi_{S.sname}\sigma_{S.rating>5}\ S2)$$

S1’s range is [0,5), so selection is empty!



Distributed Multi-table Query



$R \text{ join } S = \sigma_{R.sid=S.sid} (R \times S)$

$= \sigma_{R.sid=S.sid} ((R1 \cup R2) \times (S1 \cup S2))$

$= \sigma_{R.sid=S.sid} ((R1 \times S1) \cup (R1 \times S2) \cup (R2 \times S1) \cup (R2 \times S2))$

$= \sigma_{R.sid=S.sid} (R1 \times S1) \cup \sigma_{R.sid=S.sid} (R1 \times S2)$
$\quad \cup\, \sigma_{R.sid=S.sid} (R2 \times S1) \cup \sigma_{R.sid=S.sid} (R2 \times S2)$

$= (R1 \text{ join } S1) \cup (R1 \text{ join } S2) \cup (R2 \text{ join } S1) \cup (R2 \text{ join } S2)$

Equivalent to a union of joins over each pair of fragments