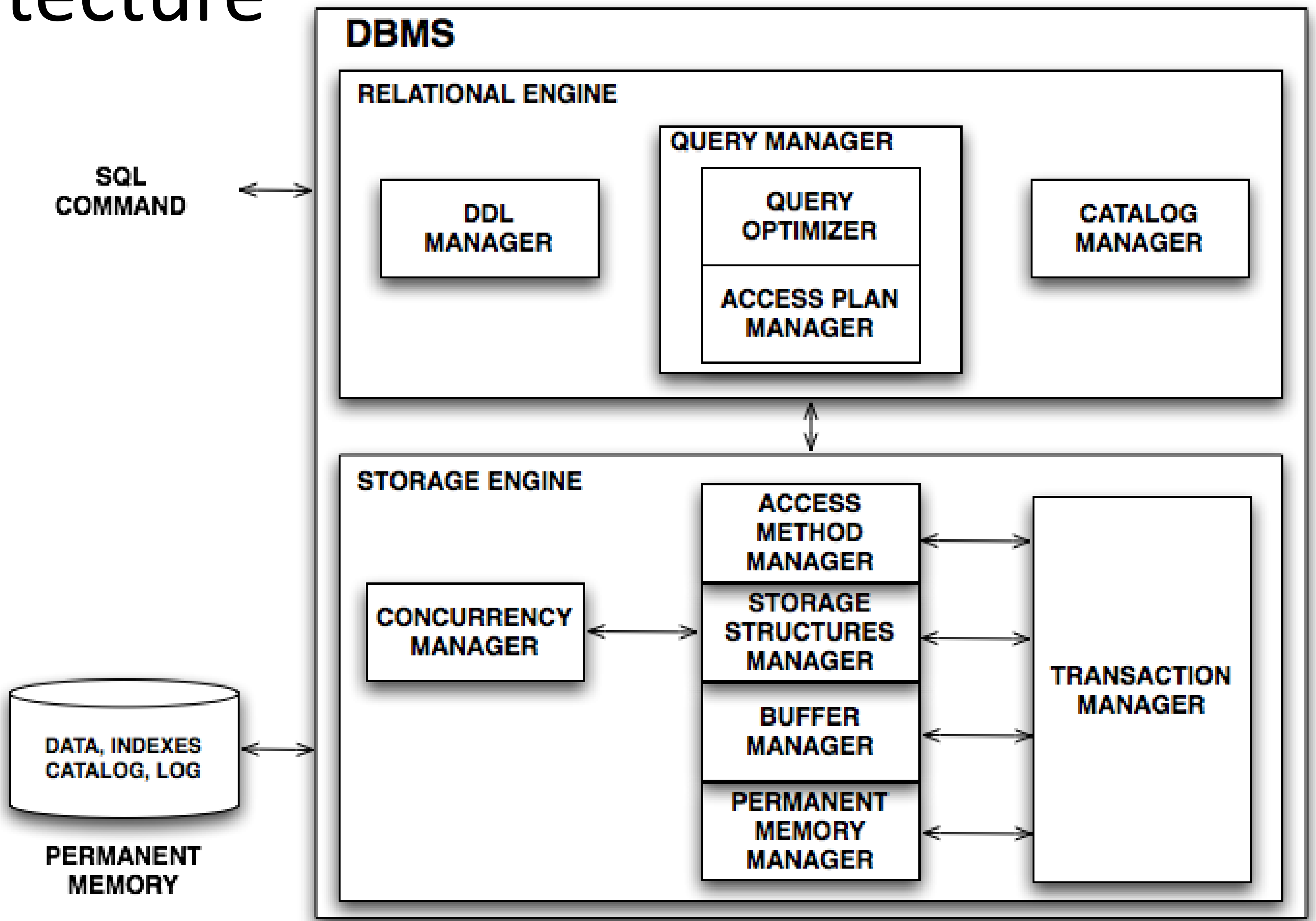
Architecture



Relational operations

- We will consider how to implement and define physical operators for:
  - Projection
  - Selection
  - Grouping
  - Set operators
  - Join
- Then we will discuss how the optimizer use them to generate physical query plans

Java Relational System (JRS):

- http://www.di.unipi.it/~albano/JRS/toStart.html

Selectivity of selection $\sigma_\psi(E)$

- The DBMS Catalog stores information about database tables and indexes.

$$s_f(A = v) = \frac{1}{N_{\mathsf{key}}(A)} \qquad (1/10)$$

$$s_f(A > v) = \frac{\mathsf{max}(A) - v}{\mathsf{max}(A) - \mathsf{min}(A)} \qquad (1/3)$$

$$s_f(A < v) = \frac{v - \mathsf{min}(A)}{\mathsf{max}(A) - \mathsf{min}(A)} \qquad (1/3)$$

$$s_f(v_1 < A < v_2) = \frac{v_2 - v_1}{\mathsf{max}(A) - \mathsf{min}(A)} \qquad (1/4)$$

Selectivity of selection $\sigma_\psi(E)$

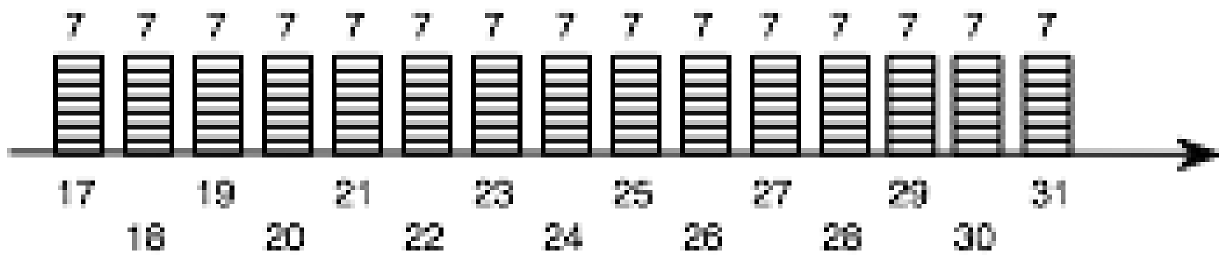
$$s_f(A = B) = \frac{1}{\mathsf{max}(N_{\mathsf{key}}(A), N_{\mathsf{key}}(B))}$$ (1/10)

$$s_f(\psi_1 \wedge \psi_2) = s_f(\psi_1) \times s_f(\psi_2)$$

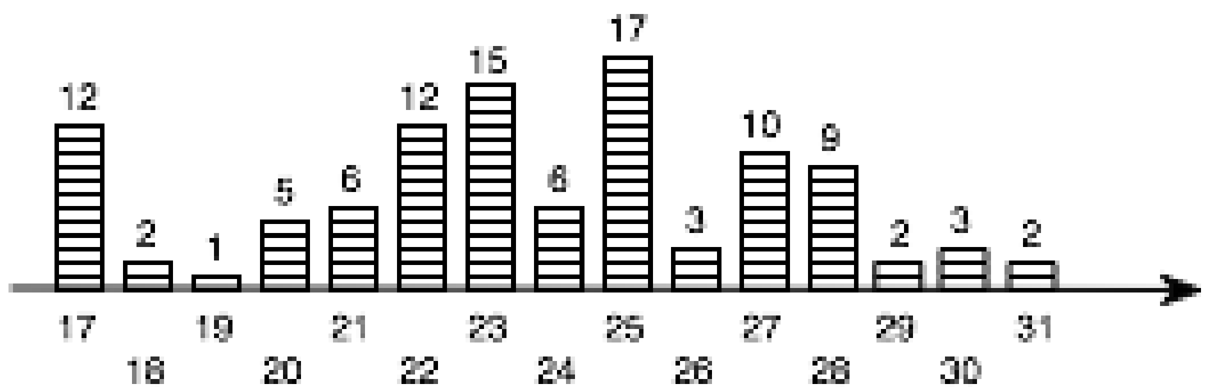
$$s_f(\psi_1 \vee \psi_2) = s_f(\psi_1) + s_f(\psi_2) - s_f(\psi_1) \times s_f(\psi_2)$$

Hystograms

105 records and A with 15 possible values in the range 17 to 31



Uniform distribution

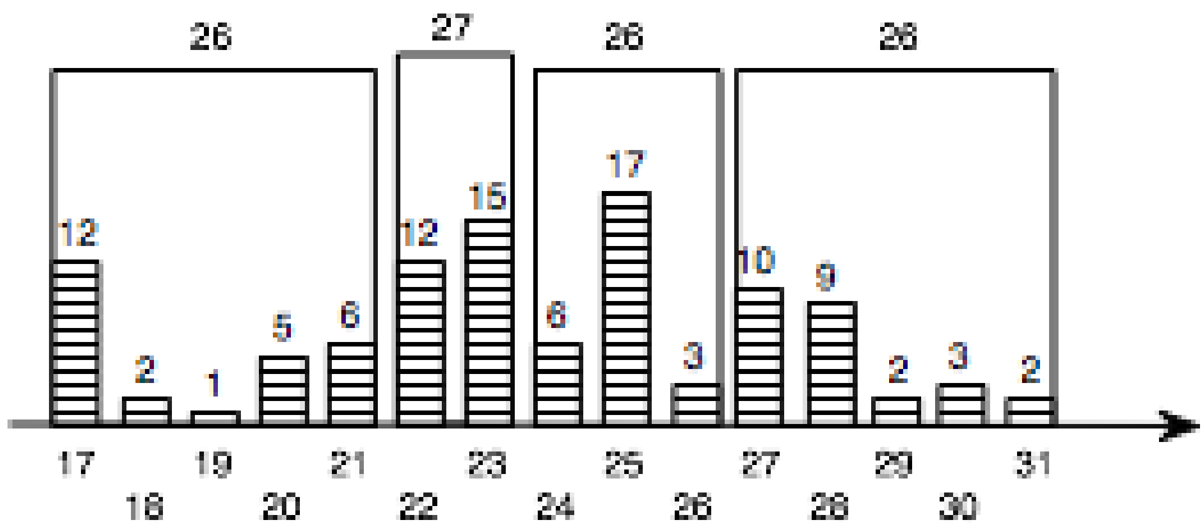


NonUniform distribution

Selectivity of selection with hystograms

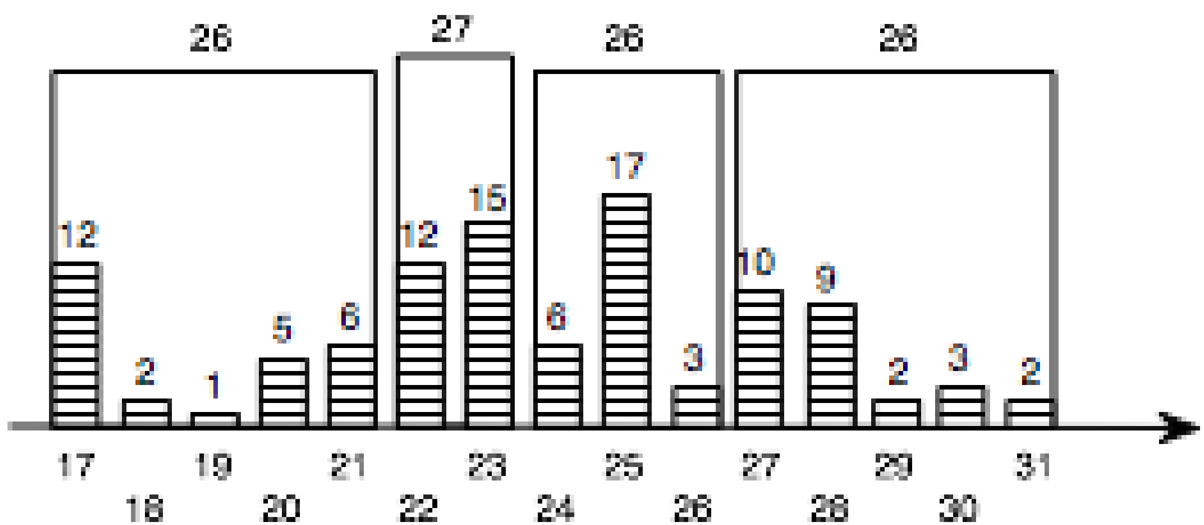
- An Equi-Height histogram is used as an approximation of the actual distribution
- The active domain of A is divided into k intervals, containing a number of records of about N_{rec}/k
- For each interval h_i are known:
  - $min(h_i)$, $max(h_i)$, $N_{key}(h_i)$, $N_{rec}(h_i)$

Equi-Height histograms



Equi-Height histograms

- $s_f(V = 24)$ = $N_{rec}(V{=}24)/N_{rec}$
- s_f real = 6/105 = 0,057
- s_f uniform = 1/15 = 7/105= 0,066
- s_f histEqH = (26/3)/105 = 0,082



Physical operators for tables and sort

- Operators for R :
  - TableScan (R)
  - SortScan (R, {A_i})
  - IndexScan (R, Idx)
  - IndexSequentialScan (R, Idx)
- Operator to sort (τ {Ai}):
  - Sort (O, {Ai})

Physical operators for $\pi_b\{A_i\}$

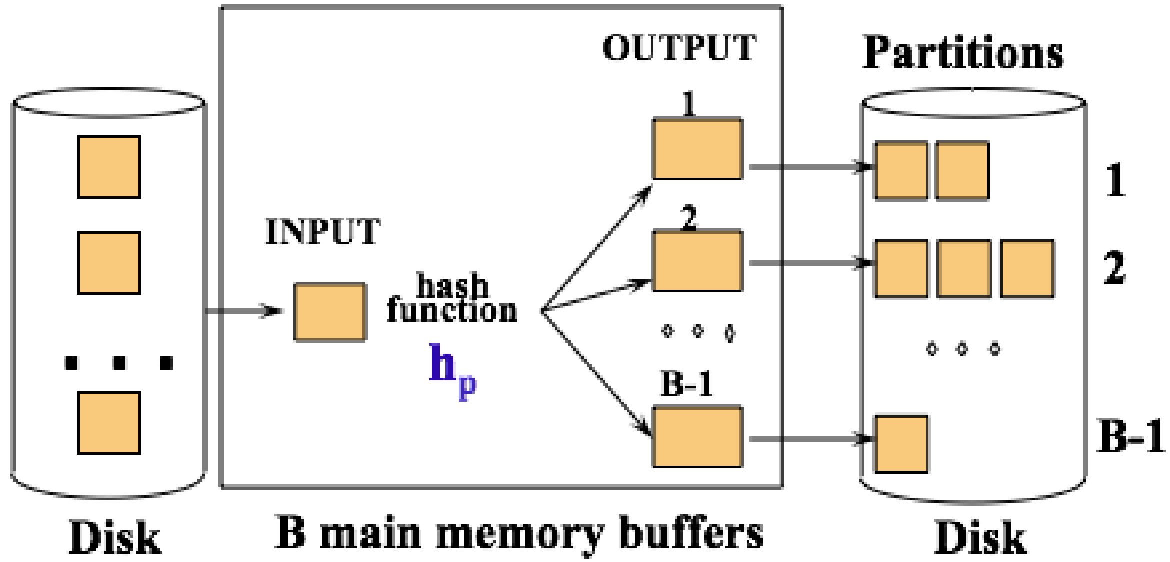
- Project (O, $\{A_i\}$): to project the records of O without duplicates elimination
  - $C = C(O)$
- IndexOnlyScan(R, Idx, $\{A_i\}$):
  - Idx an index on the attributes to project (or that contains them as prefix)
  - $C = N_{leaf}(Idx)$
- Is the result equivalent to a Project or to a Project+Distinct ?

Physical ops for duplicate elimination

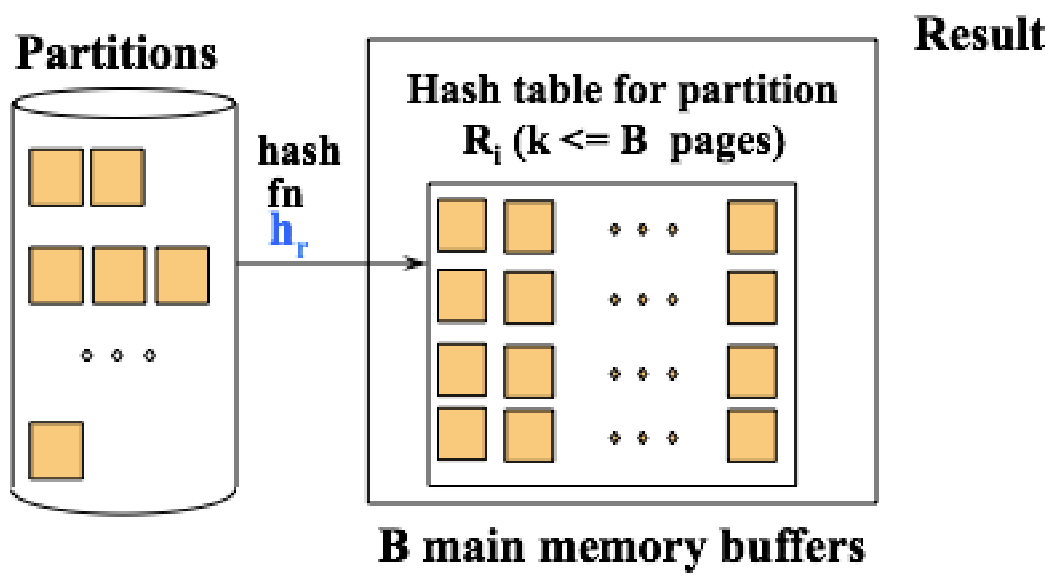
- Distinct (O): to eliminate duplicates from *sorted* records of O
  - C = C(O)
- Actually, we only need them to be grouped:
  - $r_i = r_j$ and $i < l < j$ => $r_i = r_l = r_j$
- HashDistinct(O): to eliminate duplicates from records of O;
  - C = ?

Hashdistinct

- **partitioning** phase:

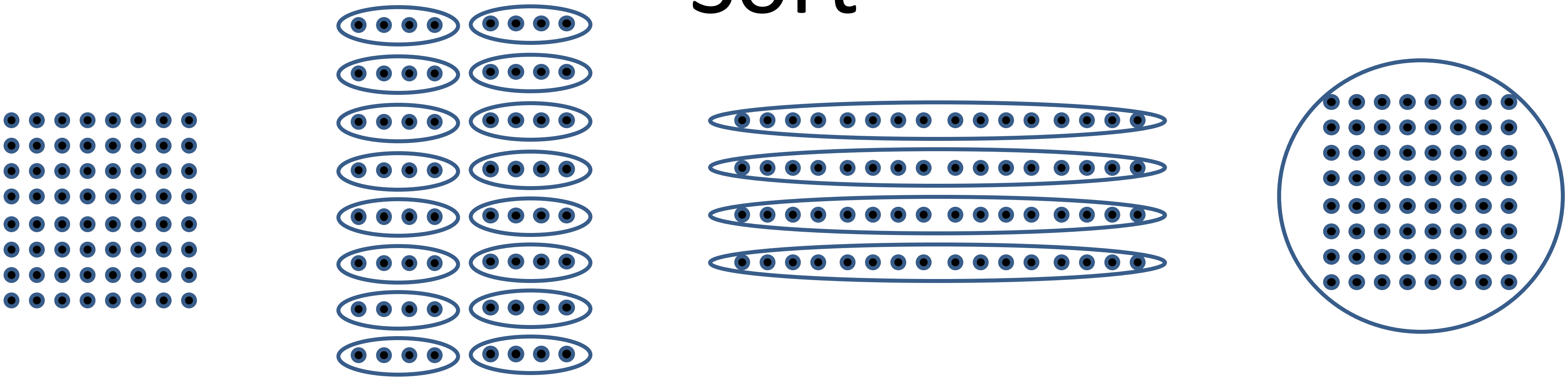


- **duplicate elimination** phase ($h_r \neq h_p$):

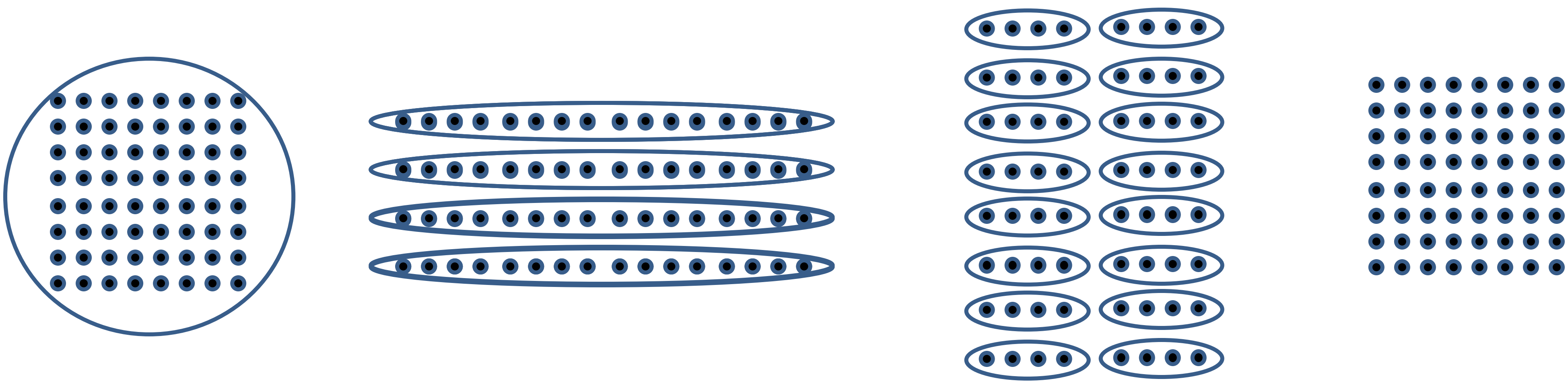


$$C = C(O) + 2\,N_{pag}(O)$$

Sort



Hash



Result size: Distinct(O)

- If A is the only O attribute
  - $|Q| = N_{key}(A)$
- If $\{A_1, A_2, \ldots, A_n\}$ are the attributes
  - $|Q| = \min(\, |O|/2, \prod_i N_{key}(A_i)\,)$
- Why not just $\prod_i N_{key}(A_i)$?

Conclusion

- The sort method often preferred because produces a sorted result and because sort is heavily optimized
- DBMS:
  - Informix uses Hash
  - DB2, Oracle, Sybase ASE use Sort
  - SQL Server e Sybase ASIQ use both methods

Simple selection

```
SELECT *
FROM Students
WHERE City = 'PI'
```

- With no index, and data unsorted:
  - Relation scan with cost $N_{pag}(R)$.
- With an index on selection attribute:
  - Use index to retrieve RIDs, then retrieve data records: $C_I + C_D$.
- Cost depends on qualifying tuples (size of $R \times s_f$), and clustering
- General selection conditions…

Physical operators for selection

- Filter (O, ψ)
  - $C = C(O)$
  - $E_{rec} = s_f(\psi) \times N_{rec}(O)$
- IndexFilter(R, Idx, ψ)
  = RidIndexFilter(Idx, ψ) + TableAccess(O, R)
  - $C = C_I + C_D$
  - $E_{rec} = s_f(\psi) \times N_{rec}(R)$

Physical operators for selection

- IndexSequentialFilter(R, Idx, ψ)
  - $C = s_f(\psi) \times N_{leaf}(Idx)$
- IndexOnlyFilter(R, Idx, {A_i}, ψ)
  - $C = s_f(\psi) \times N_{leaf}(Idx)$
- AndIndexFilter(R, {Idx_i, ψ_i})
  - $C_I = \Sigma_i\, C_I(Idx_i)$
  - $C_D = \Phi(E_{rec}, N_{pag}(R))$
  - $E_{rec} = s_f(\psi) \times N_{rec}(R)$
- OrIndexFilter(R, {Idx_i, ψ_i})

Conjunctive selection in DBMSs

- Informix, DB2 use intersection of RID sets
- Oracle, Sybase ASIQ use bitmaps
- Oracle use also HashJoin with index (later) on the attribute RID
- Sybase ASE uses one index only
- SQL Server use index join (later)

Exercises

SELECT A, B
FROM R
WHERE (A **BETWEEN** 50 **AND** 100) **AND** B > 20;

Idx an index on A

SELECT A, B
FROM R
WHERE (A **BETWEEN** 50 **AND** 100) **AND** B > 20
ORDER BY A;

One index on A, B

SELECT **DISTINCT** A, B
FROM R
WHERE (A **BETWEEN** 50 **AND** 100) **AND** B > 20
ORDER BY B;

Two indexes on A and on B

Physical operators for grouping

- As for duplicate elimination:
  - Sorting
  - Hashing
  - Using an index on the grouping attributes
- (Duplicate elimination is grouping on all attributes with no aggregation)

Physical operators for ($\{A_i\}\ \gamma\ \{f_j\}$)

- GroupBy (O, $\{A_i\}$, $\{f_j\}$): to group the records of O **sorted** on the $\{A_i\}$, computing functions in $\{f_j\}$.
  - $\{f_j\}$ contains the aggregation functions present in the SELECT and HAVING clauses.
  - The operator returns records with attributes $\{A_i\} \cup \{f_j\}$
  - The records of O must be sorted on the $\{A_i\}$
  - Any permutation of $\{A_i\}$ is ok
  - (Actually, they only need to be grouped on the $\{A_i\}$)
- HashGroupBy (O, $\{A_i\}$, $\{f_j\}$)
- C? Cardinality?

Computing aggregations

- Each aggregation functions is an object o with methods start(), next(v), end()
- With the first record of a group: o.start()
- For each record of the group o.next(v) is invoked
- o.end() computes the final aggregate value.

Example

```
SELECT    A, SUM(C) AS T
FROM      R
WHERE     A BETWEEN 50 AND 100
GROUP BY  A
HAVING    COUNT(*) > 1;
```

$\pi^{b}_{A,\,T}$

|

$\sigma_{N > 1}$

|

$_{A}\gamma_{SUM(C)\ AS\ T,\ COUNT(*)\ AS\ N}$

|

$\sigma_{\dots}$

|

R

Logical Tree

... = A **BETWEEN** 50 **AND** 100

Project
({A, T})

|

Filter
(N > 1)

|

GroupBy
({A}, {SUM(C) AS T , COUNT(*) AS N})

|

Sort
({A})

|

Filter
(...)

|

TableScan
(R)

Physical Tree

Physical ops for join: nested loops

- foreach r in O_E do
  - foreach s in O_i do
    - if r.r1 = s.s1 then add <r, s> to result

- $C = C(O_E) + E_{rec}(O_E) \times C(O_I)$
- $E_{rec} = s_f\,(C_j) \times E_{rec}(O_E) \times E_{rec}(O_I)$
- Inequality join conditions ?

Cost of nested loops

```
foreach  r in R  do
    foreach  s in S do
        if r.r1 = s.s1  then add <r, s> to result
```

With R **external**:

$$\mathbf{C =} \; Npag(R) + Nrec(R) \times Npag(S) \approx Npag(R) \times \frac{Nrec(R)}{Npag(R)} \times Npag(S)$$

With S **external**:

$$\mathbf{C =} \; Npag(S) + Nrec(S) \times Npag(R) \approx Npag(S) \times \frac{Nrec(S)}{Npag(S)} \times Npag(R)$$

Hence…

Page nested loops

R

Pk	B
10	a
20	b
30	c
40	d

Join

S

Fk	C
10	f
20	g
30	g
10	k
40	f
20	k

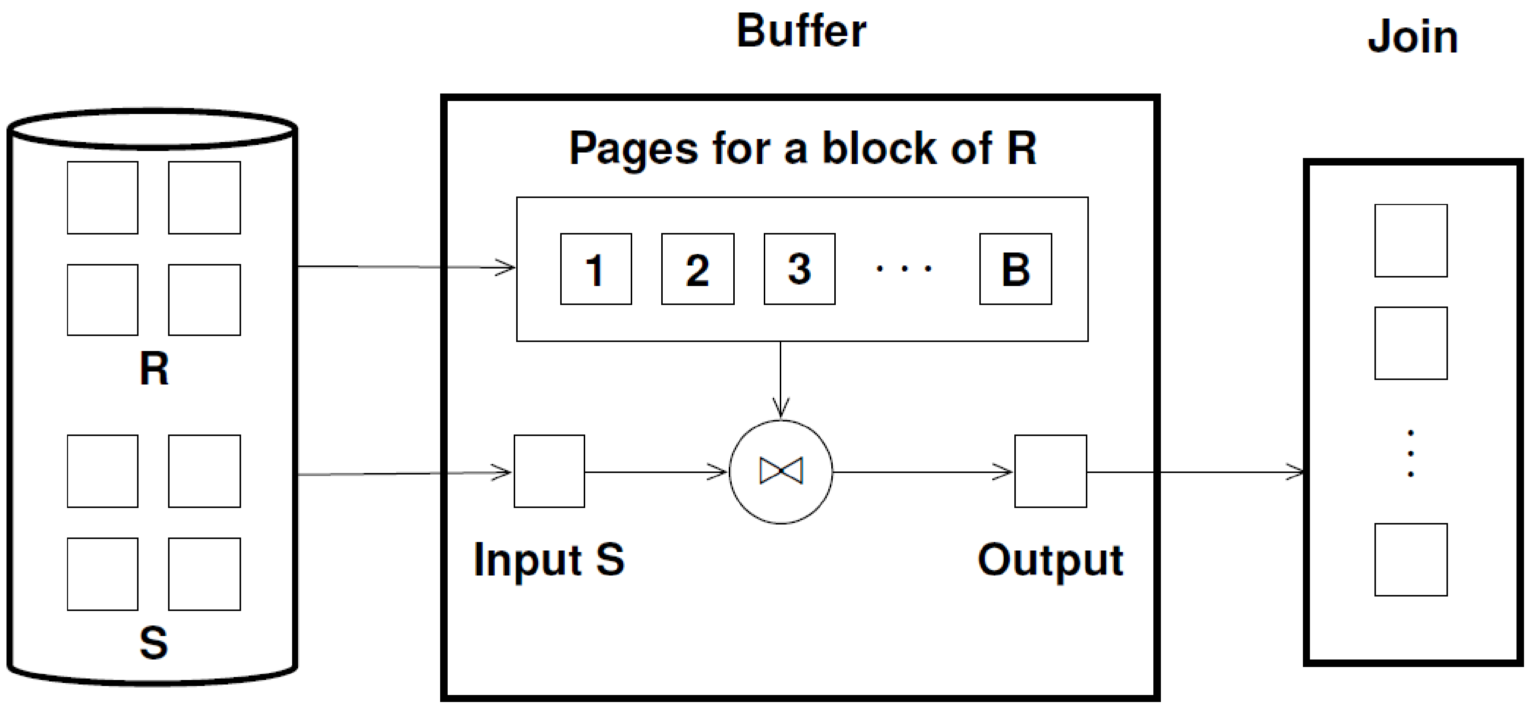
=

Pk	B	Fk	C
10	**a**	**10**	**f**
20	**b**	**20**	**g**
10	**a**	**10**	**k**
20	**b**	**20**	**k**
30	c	30	g
40	d	40	f

C = Npag(R) + Npag(R) * Npag(S)

The smallest relation is used as **external**

Block nested loops join



$$C_{BNL} = N_{pag}(R) + \lceil N_{pag}(R)/B \rceil \times N_{pag}(S)$$

Index nested loop

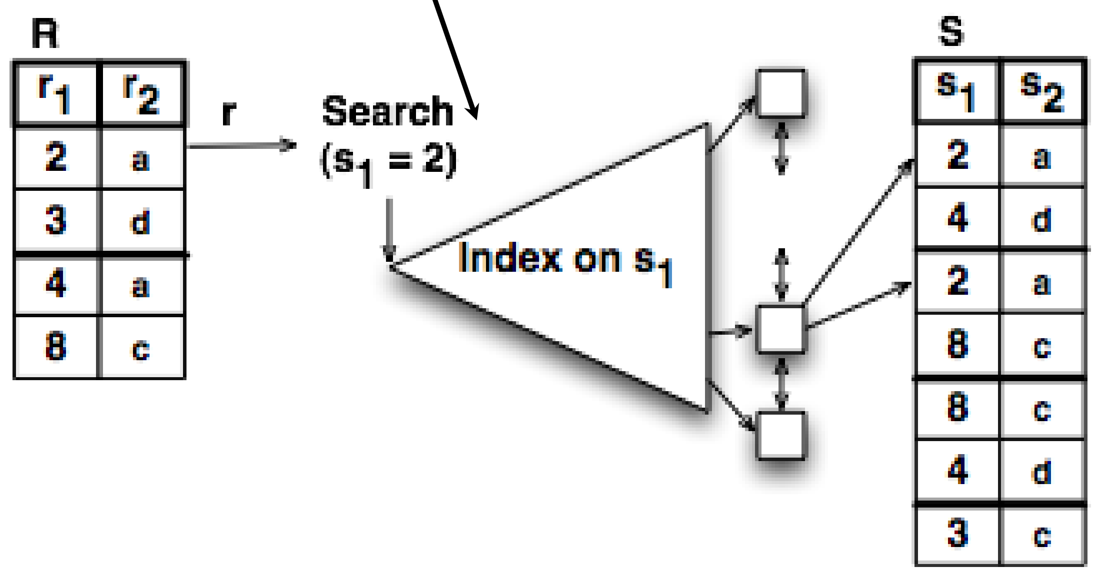
- Hyp: There is an index on the join column of the internal relation (S)

foreach r in R do

  foreach s in ***IndexFilter***(S, I, s.s1=r.r1) do

    add <r, s> to result

$R(r1, r2) \bowtie_{r1=s1} S(s1, s2)$



Index nested loop

```
foreach r in R do
        foreach s in IndexFilter(S,I,s1 =r.r1)  do
                add <r, s> to result
```

- Cost for R join S:
  - $C = N_{pag}(R) + N_{rec}(R) \times CaWithIdx(S)$
- General Case:
  - $C = C(O_E) + E_{rec}(O_E) \times (C_I + C_D)$

Merge join

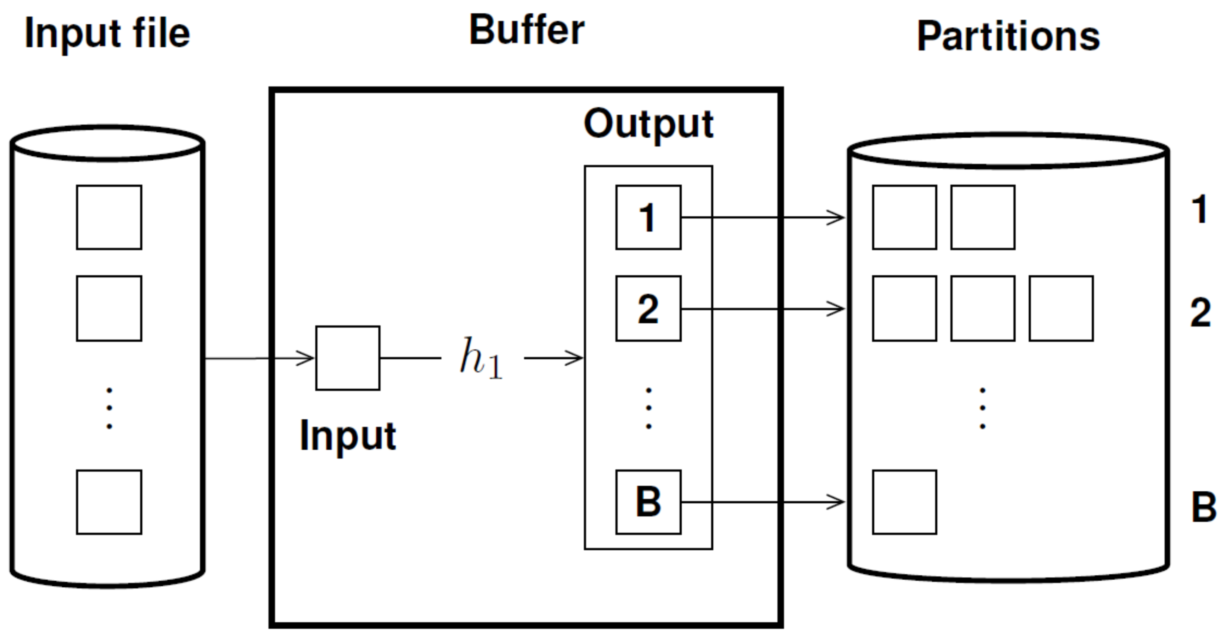
- Hyp: R and S are sorted on the join attribute, a key of the external relation
- $C = C(O_E) + C(O_I)$
- Inequality join conditions ?

...	ri
...	1
...	2
...	3
...	4

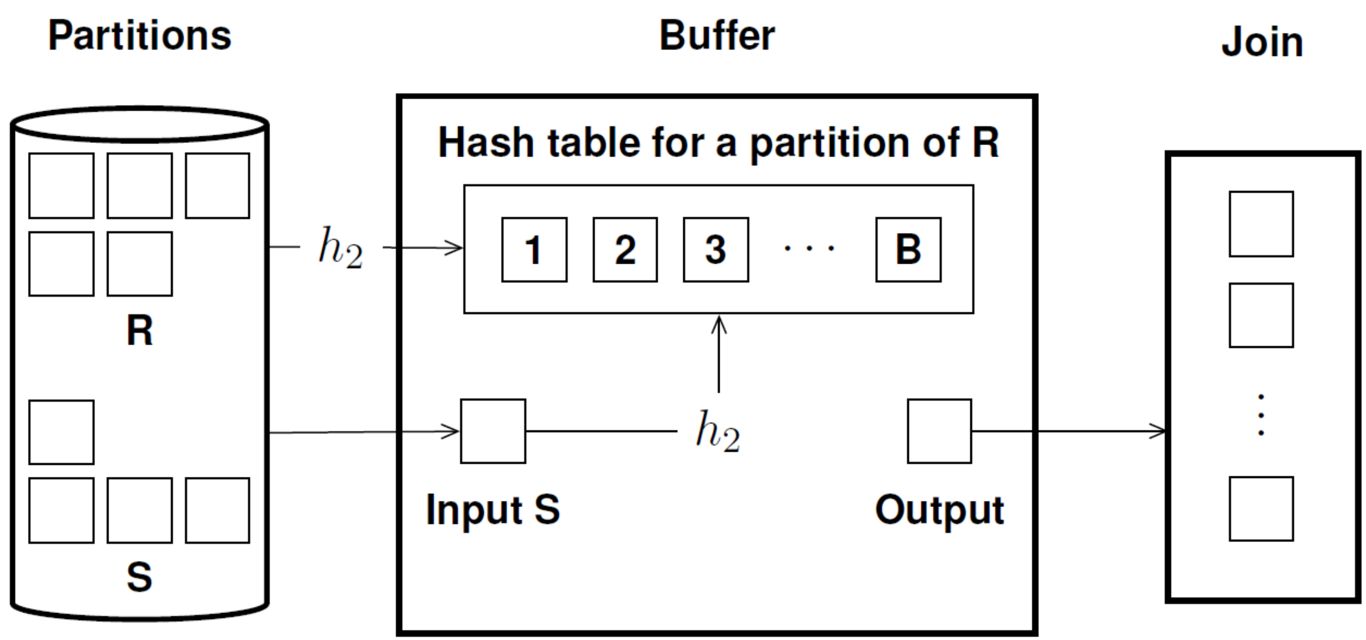
sj	...
1	...
1	...
1	...
3	...
3	...
3	...
4	...

Hash join

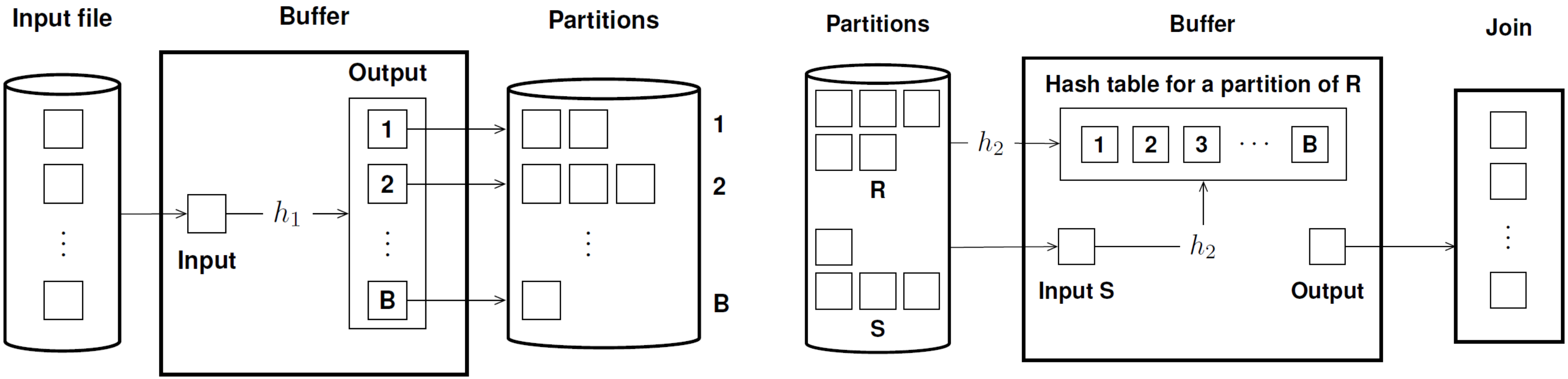
- Partitioning (R and S)



- Matching



Hash join: cost



- Assume $N_{pag}(R)/B < B$ and uniformity
- $C = C(O_E) + C(O_I) + 2(\ N_{pag}(O_E) + N_{pag}(O_I)\ )$
- $C = (\log_B(N_{pag}(O_E)) \times 2\text{-}2)*(\ N_{pag}(O_E) + N_{pag}(O_I)\ )$
- C(R join S) = 3×(Npag(R) + Npag(S))
- What if $N_{pag}(R) < B$?

Complex joins

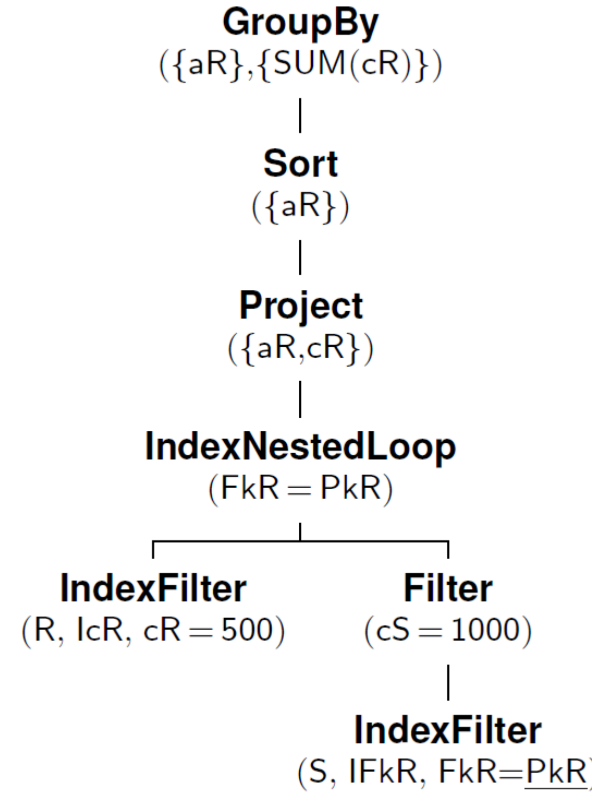
- Join with a conjunctive condition
- Join with a disjunctive condition

Physical operators for join

- NestedLoop (O_E,O_I, ψ_J)
- PageNestedLoop (O_E,O_I, ψ_J)
- IndexNestedLoop (O_E,O_I, ψ_J)
- MergeJoin (O_E,O_I, ψ_J)
- HashJoin (O_E,O_I, ψ_J)

Example

SELECT aR, Sum(cR)
FROM R, S
WHERE R.PkR=S.FkR **AND** cS=1000
GROUP BY aR
ORDER BY aR



$$C_{GroupBy} = C_{Sort} = C_{IndexNestedLoop} + 2 \times N_{pag}(Project)$$

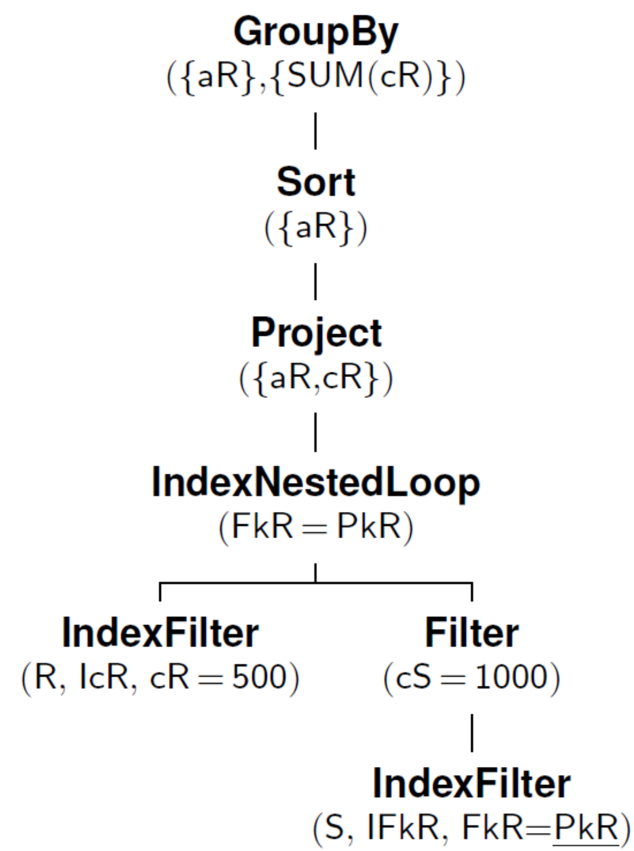
$$N_{Pag}(Project) = (E_{Rec}(IndNestedLoop) \times (L(aR) + L(cR)))/D_{pag}$$

$$C_{INL} = C(IndexFilterOnR) + E_{rec}(IndexFilterOnR) \times C(Filter)$$

$$E_{rec}(INL) = s_f(PkR = FkR) \times E_{rec}(IndexFilterOnR) \times E_{rec}\ (Filter)$$

Example

- $C(\text{IndFilOnR}) = C_I + C_D$
  - $C_I = \lceil N_{leaf}(\text{IcR})/N_{key}(\text{IcR}) \rceil$
  - $C_D = \Phi(N_{rec}(R)/\ N_{key}(\text{IcR}),\ N_{pag}(R))$
- $E_{rec}(\text{IndFilOnR}) = N_{rec}(R)/\ N_{key}(\text{IcR})$
- $C(\text{IndFilOnS}) = C_I + C_D$
  - $C_I = \lceil N_{leaf}(\text{IFkR})/N_{key}(\text{IFkR}) \rceil$
  - $C_D = \Phi(N_{rec}(S)/\ N_{key}(\text{IFkR}),\ N_{pag}(S))$
- $E_{rec}(\text{Filter}) = s_f(cS=1000) \times N_{rec}(S)$

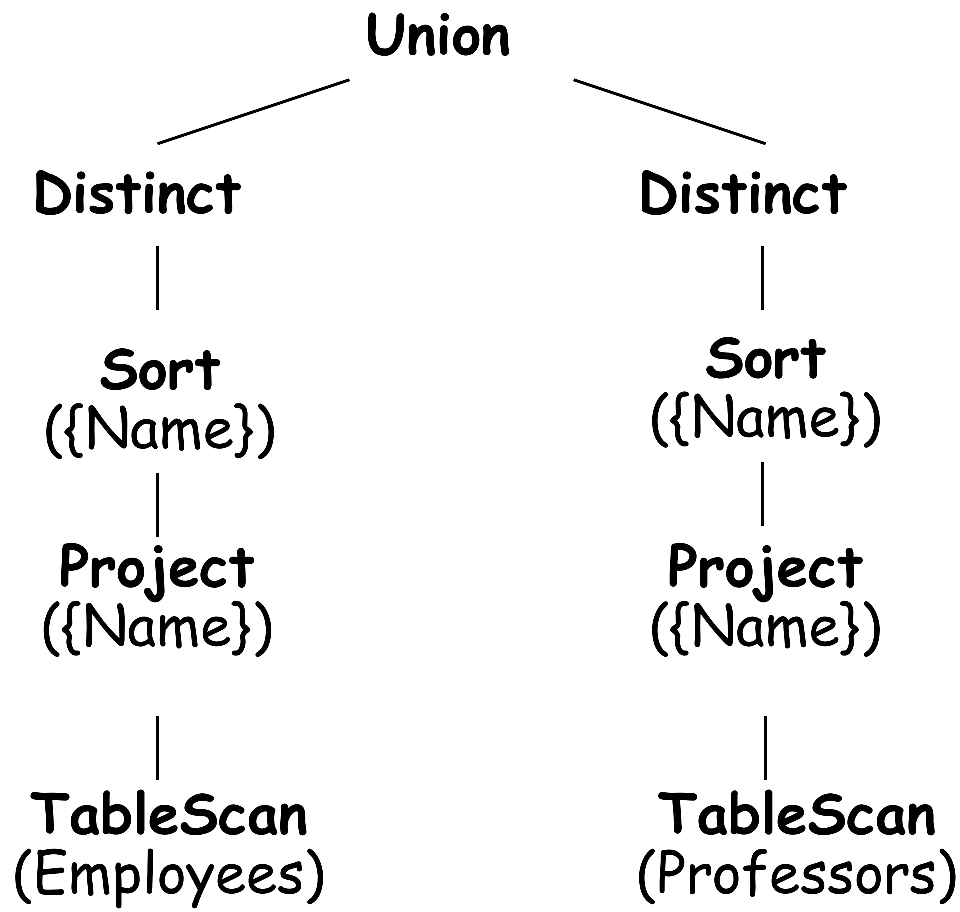


Set operators

- Union(O_E,O_I), Except(O_E,O_I), Intersect(O_E,O_I)
  - Operand sorted and without duplicates.
- HashUnion(O_E,O_I), HashExcept(O_E,O_I), HasIntersect(O_E,O_I)
  - Using hash.
- UnionAll(O_E,O_I)
  - trivial
- ExceptAll(O_E,O_I), IntersectAll(O_E,O_I)
  - Not always supported

Example

```
SELECT   Name
FROM     Employees
 UNION
SELECT   Name
FROM     Professors ;
```



Summary

- Very few basic operators, hence the implementation of these operators can be carefully tuned.
- No universally superior technique for operators with many implementations
- We must consider available alternatives and select the best one: “Query optimization”
- ... the next topic