Advanced Topics in Data Management

Lecture 2

Announcements

- Review 1 was due today
- Review 2 due next Thursday

- Today: continue query processing
- Next Thursday:
  - Rebecca Taft, Cockroach Labs optimizer
  - Max Willsey, EGG equality saturation

Outline for Today

- Recap
- Dynamic Programming
- Subqueries
- Operator Interface
- Cardinality Estimation

Recap

- Relational data model:

- Logical Data Model = relations
- Physical Data Model = NONE!

- Physical data independence

Recap

- SELECT-FROM-WHERE
  - Nested loop semantics
- Complications:
  - NULLs
  - Group-by / Aggregates
  - Outer Joins
  - Subqueries

Recap: Relational Algebra

- Selection: σ
- Projection: Π
- Join: $\bowtie$
- Union: $\cup$
- Group-by, aggregate: γ
- Outer-join: ⟗, ⟕, ⟖
- Semi-join: $\ltimes$, Anti-semi-join: $\rhd$

Recap: Physical Plans

- Joins:
  - Nested loop join
  - Hash-join
  - Merge-join
  - Extensions: disk-based, distributed
- Group-by:
  - Hash, merge
- Selection: on-the-fly

Dynamic Programming

Dynamic Programming

- Let n = number of relations to join
- For s = 1, n do:
  - For each subset S of of size s do:
    - Split S into [relation R] + [set of s-1 relations S']
    - Lookup Cost(S')
    - Cost(S) := $\min_{splits}$ (Cost(S') + cost-of($R \bowtie S'$))
    - Memorize (S, Cost(S))
- Return Cost(All-relations)

Paper Discussion

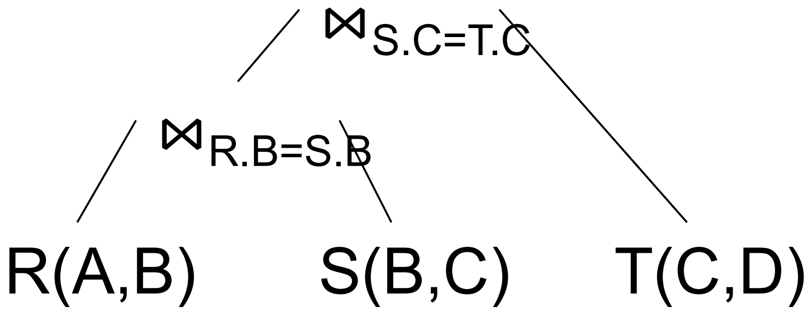
- Analysis of Two Existing and One New Dynamic Programming Algorithm, VLDB'2006
- Assumes only plans without cartesian products (why?)

Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$

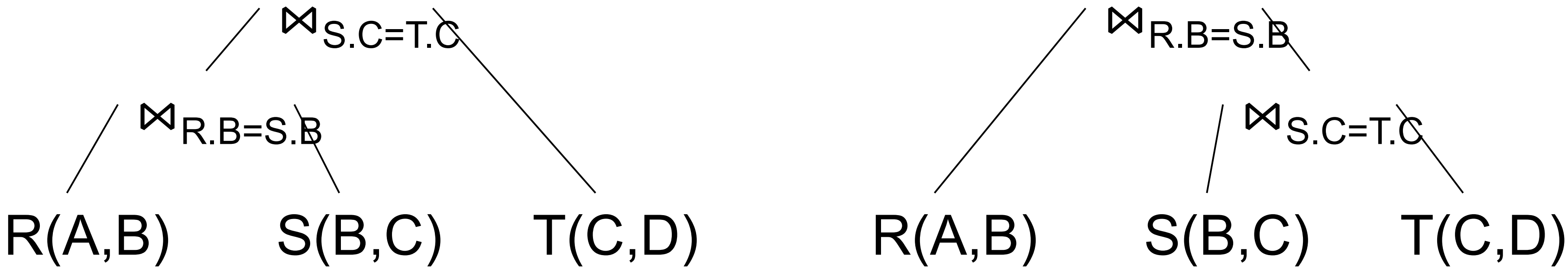
Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$



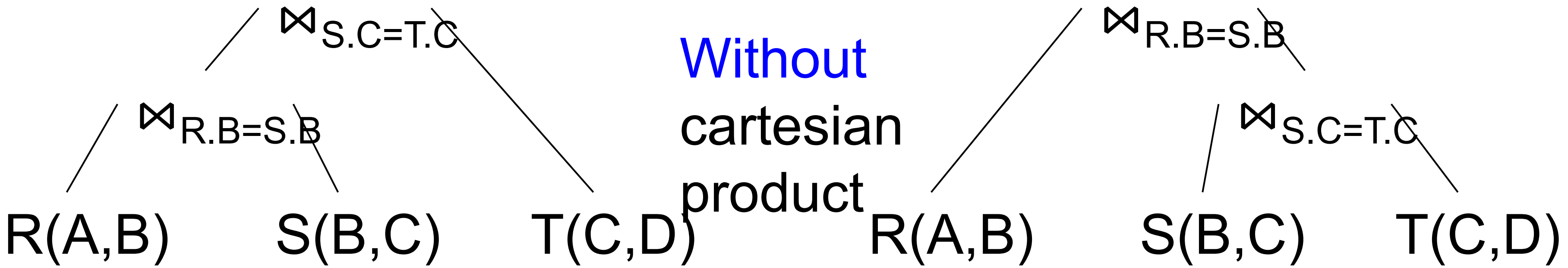
Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$



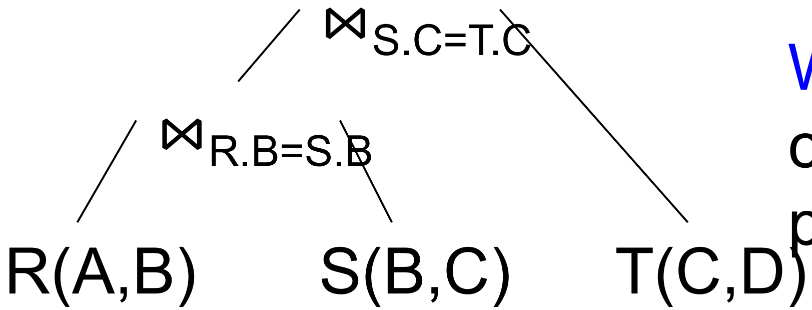
Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$

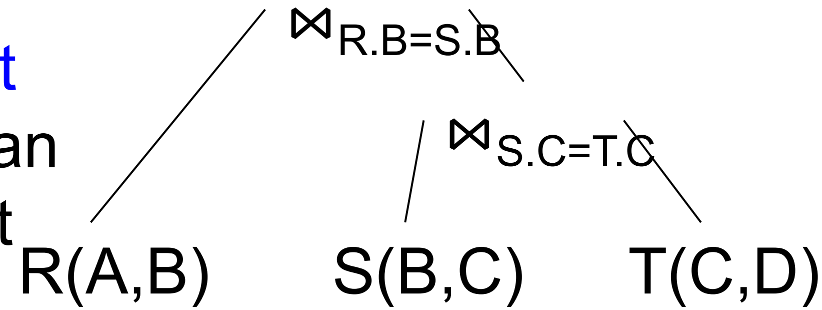


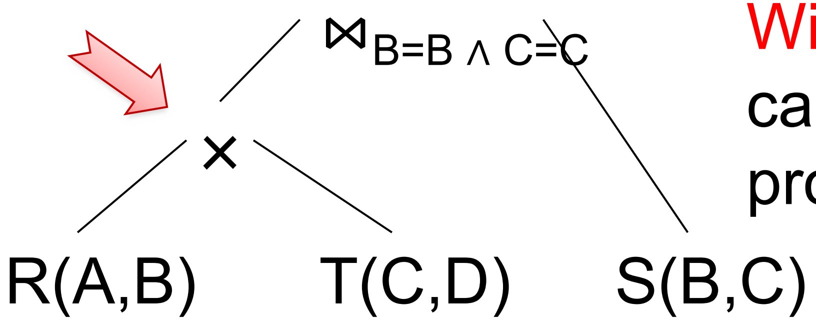
Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$



Without cartesian product

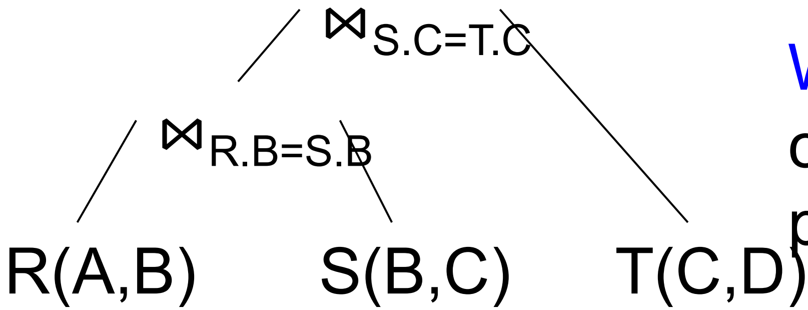




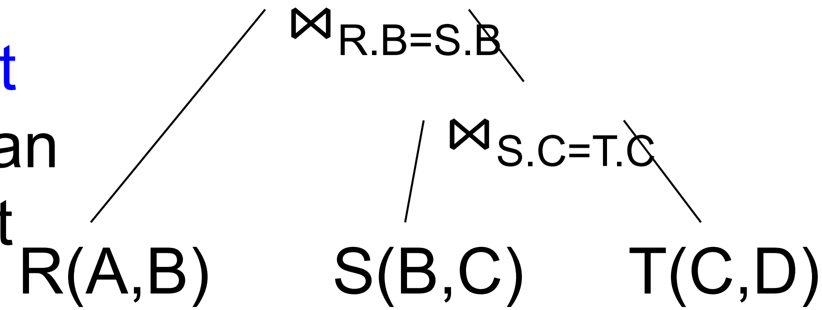
With cartesian product

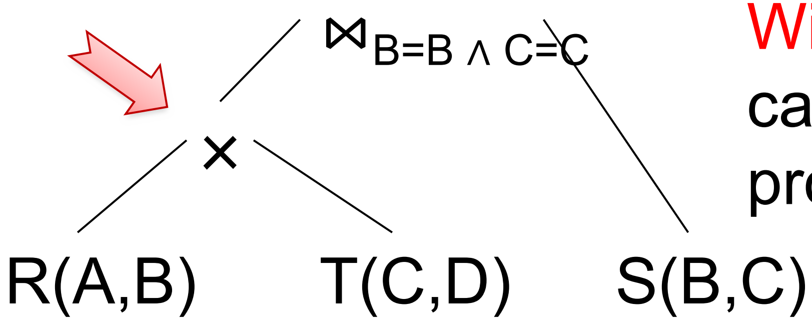
Cartesian Products

$R(A,B) \bowtie_{R.B=S.B} S(B,C) \bowtie_{S.C=T.C} T(C,D)$



Without cartesian product

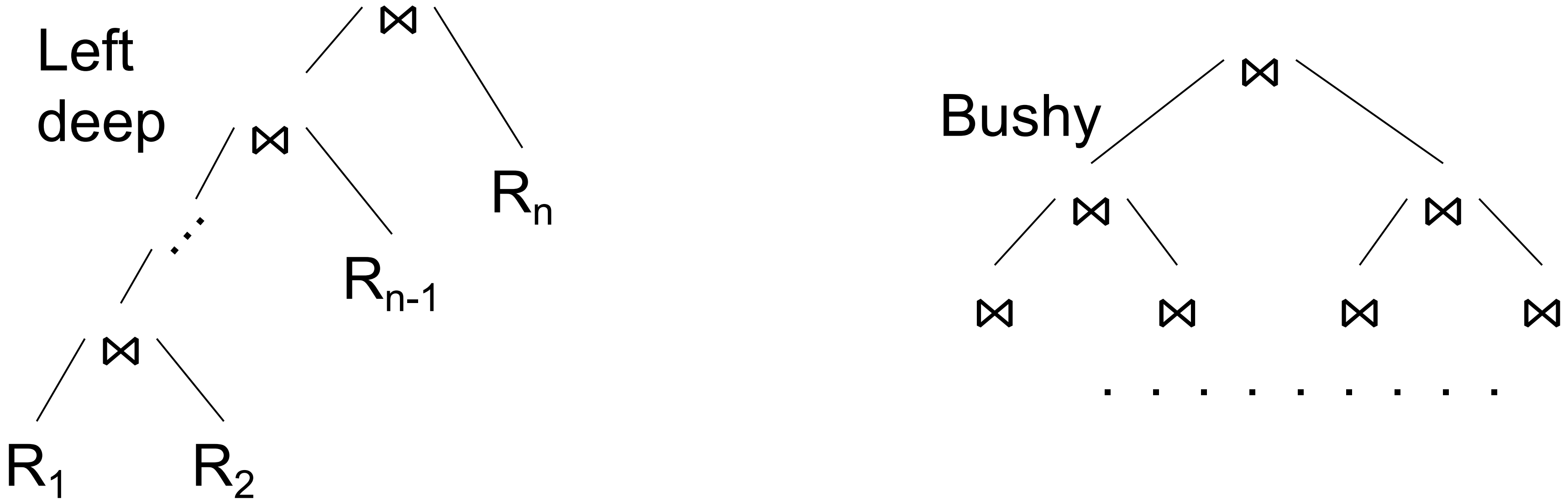




With cartesian product

When could this plan be better?

Left-Deep v.s. Bush Trees



Some Popular Query Shapes

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

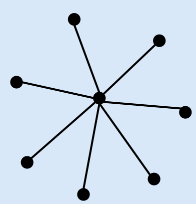
Some Popular Query Shapes

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

Star query: $S(X_1, X_2, \ldots, X_n) \bowtie R_1(X_1) \bowtie R_2(X_2) \bowtie \cdots \bowtie R_n(X_n)$

Query graph



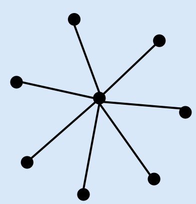
Some Popular Query Shapes

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 ——— R_2 ——— R_3 ——— . . . ——— R_n

Star query: $S(X_1, X_2, \ldots, X_n) \bowtie R_1(X_1) \bowtie R_2(X_2) \bowtie \cdots \bowtie R_n(X_n)$

Query graph



Clique query: very rare in practice

Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

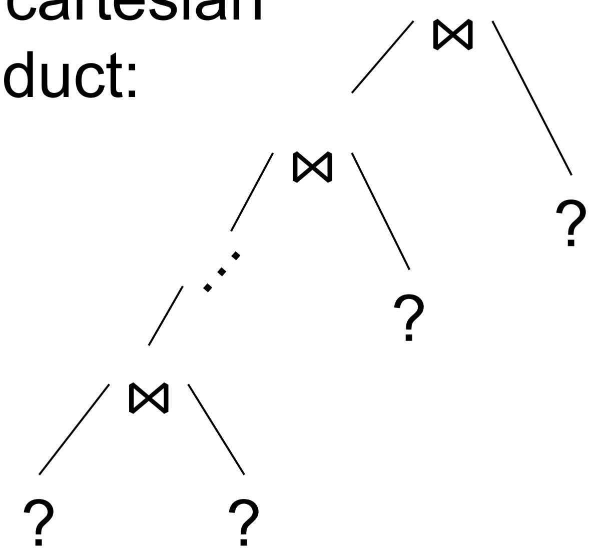
Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:

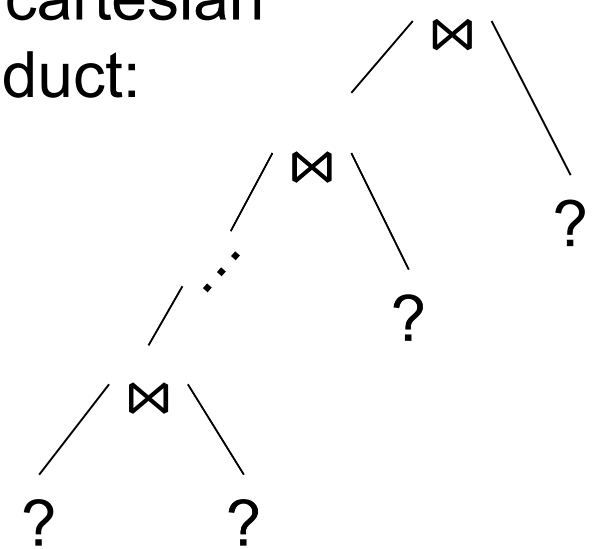


Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



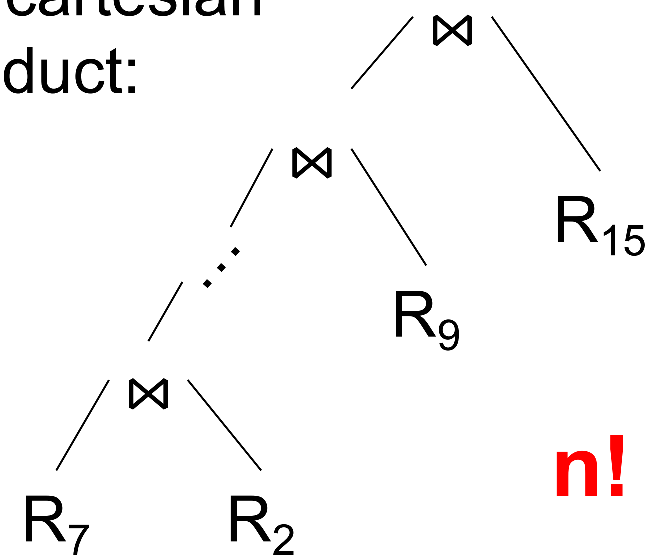
Can place relations in any order

Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



Can place relations in any order

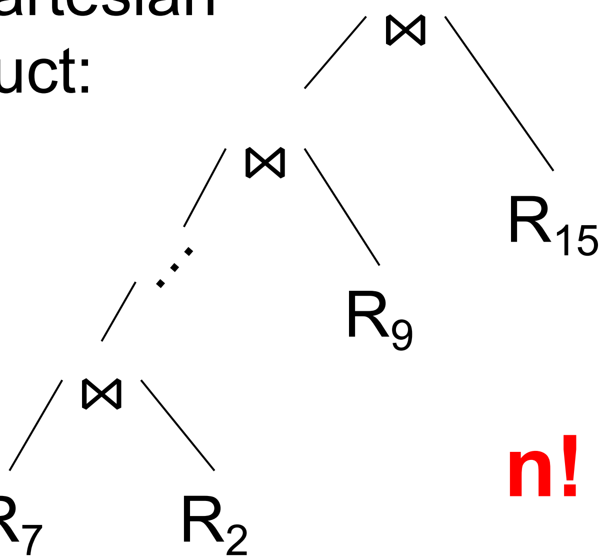
n!

Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

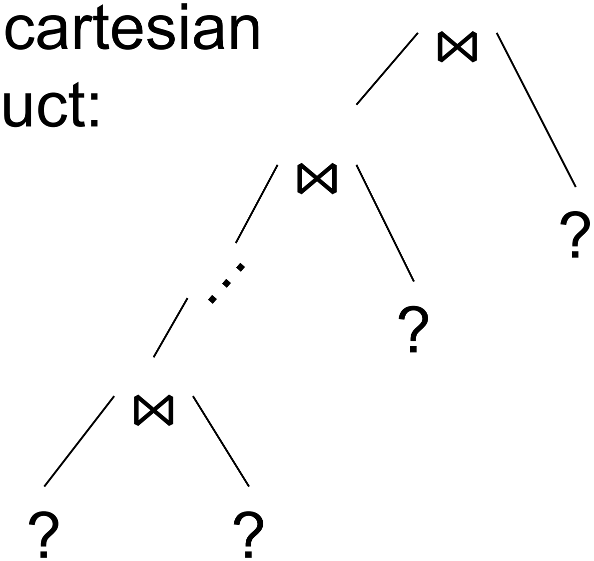
Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



n!

W/o cartesian product:

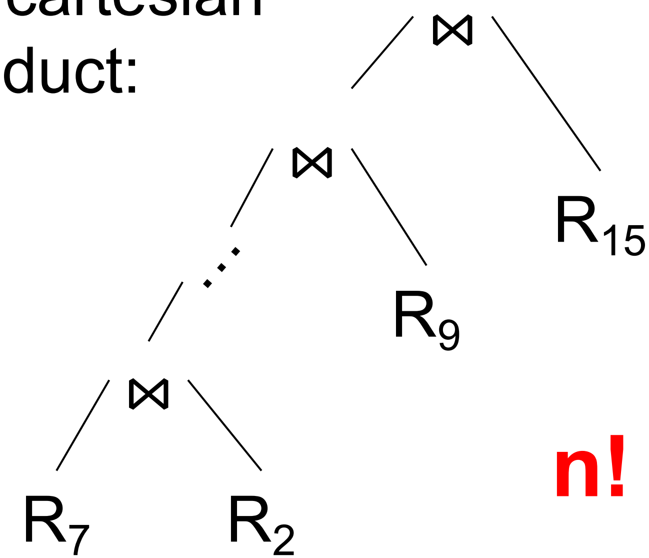


Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

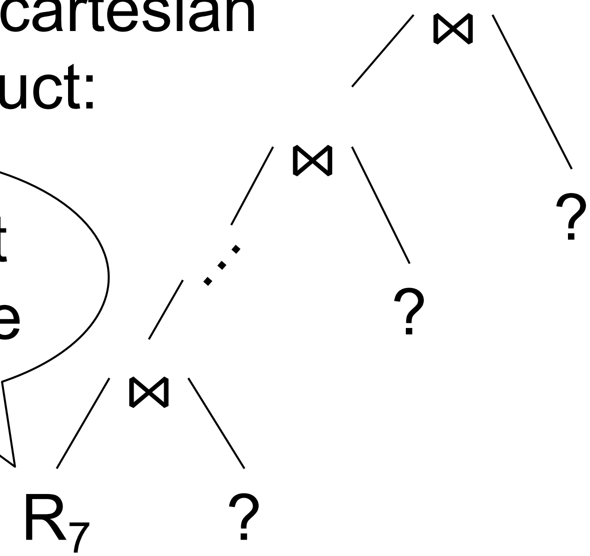
W/ cartesian product:



n!

W/o cartesian product:

May start anywhere

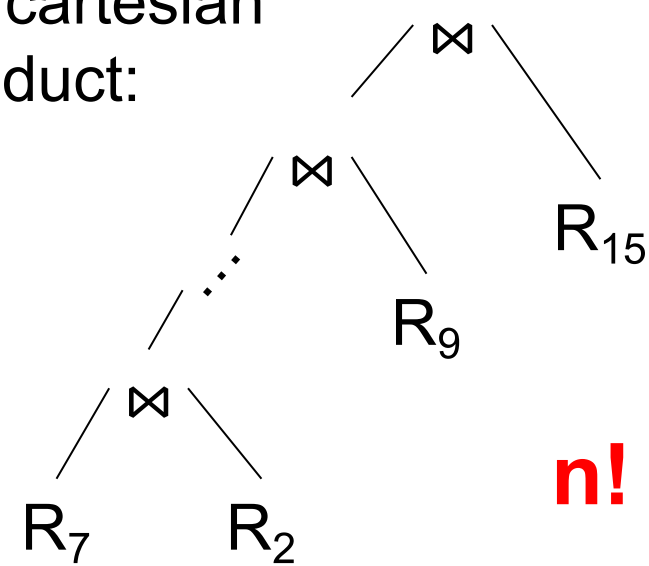


Number of Left-Deep Plans

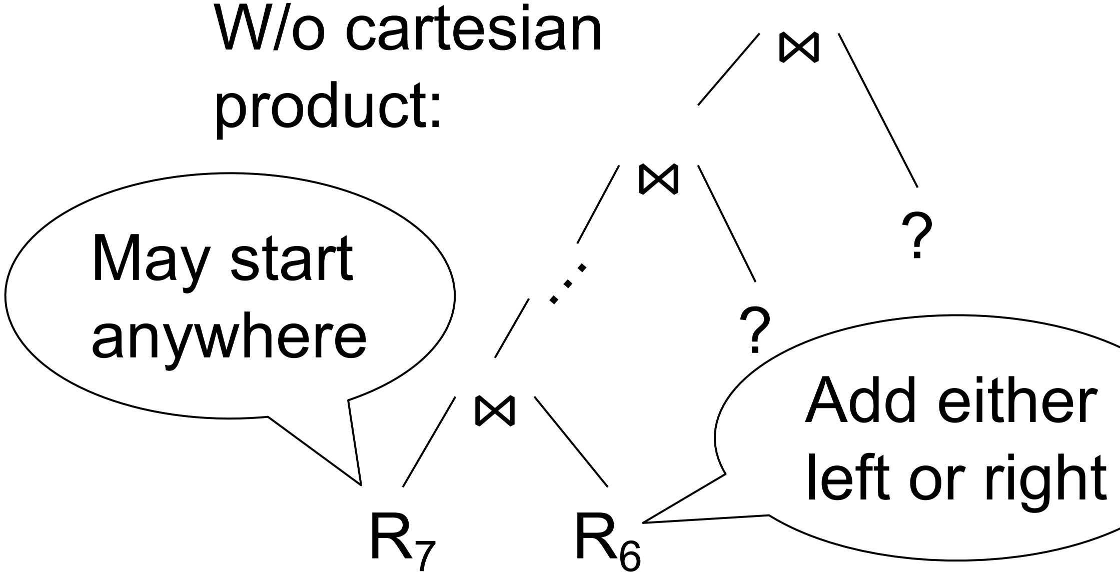
Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



W/o cartesian product:

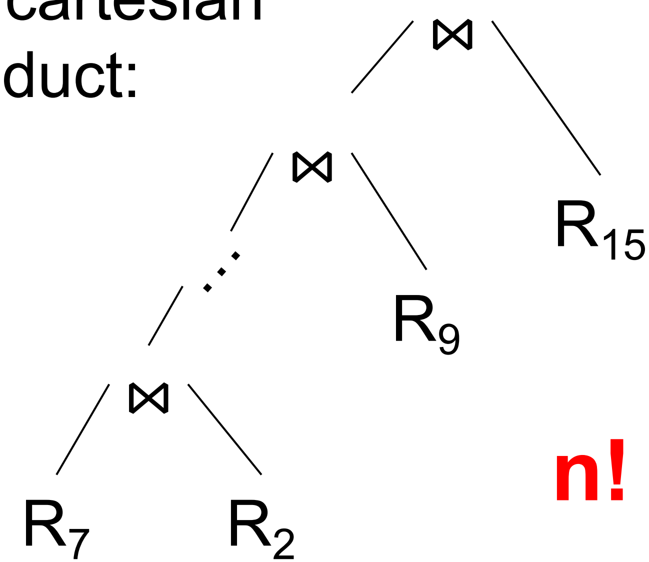


Number of Left-Deep Plans

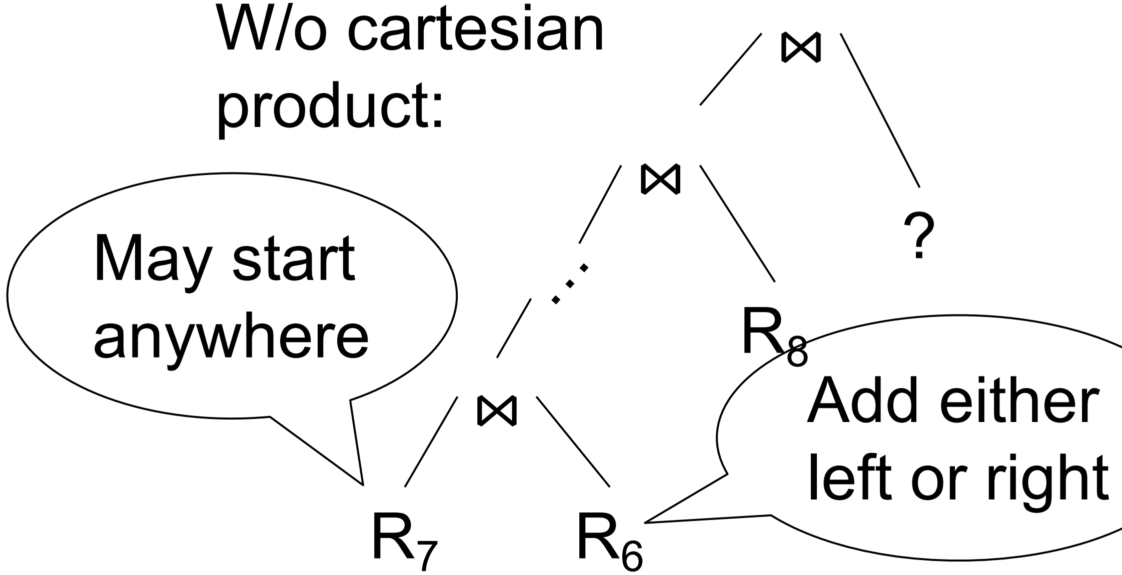
Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



W/o cartesian product:

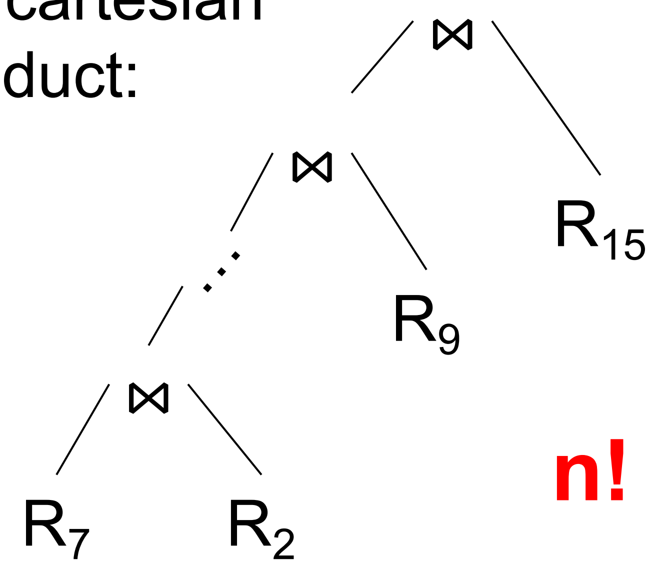


Number of Left-Deep Plans

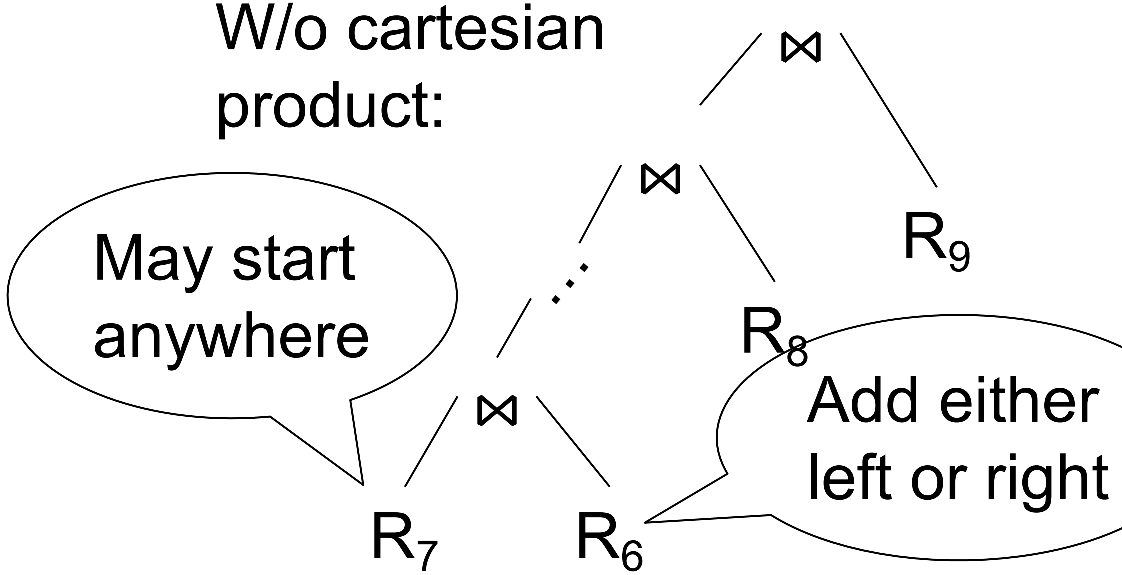
Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



W/o cartesian product:

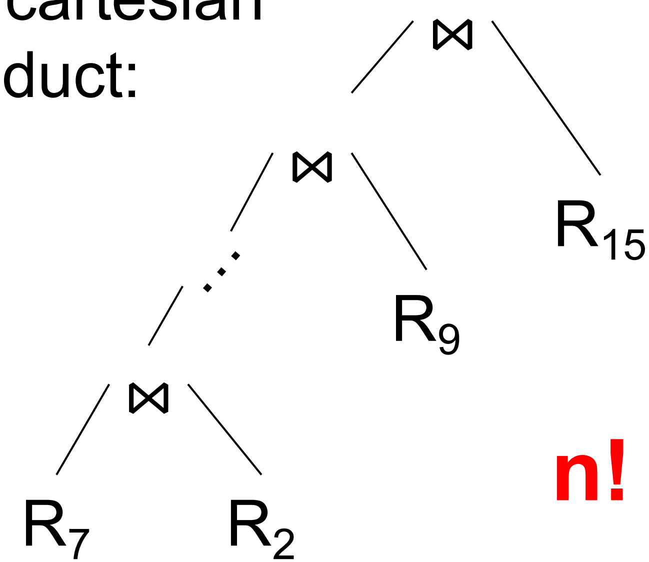


Number of Left-Deep Plans

Chain query: $R_1(X_0, X_1) \bowtie R_2(X_1, X_2) \bowtie \cdots \bowtie R_n(X_{n-1}, X_n)$

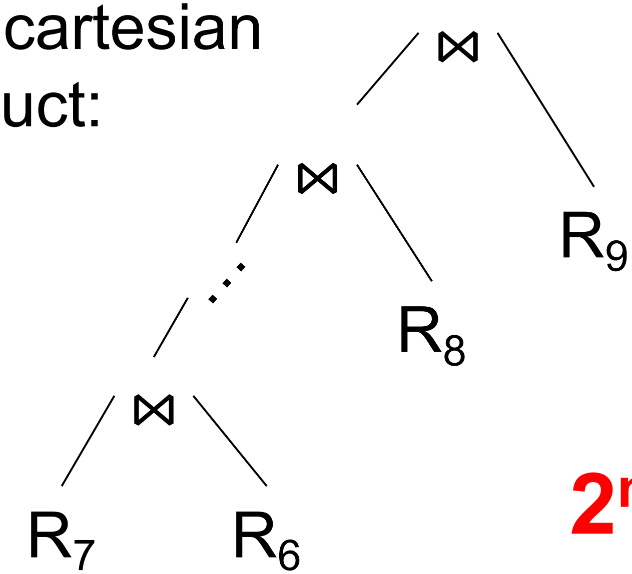
Query graph R_1 —— R_2 —— R_3 —— . . . —— R_n

W/ cartesian product:



n!

W/o cartesian product:



2^{n-1}

Discussion

- $n!$ can be much worse than 2^{n-1}.

n	2^{n-1}	n!
2	2	2
3	4	6
4	8	24
5	16	120
6	32	720
7	64	5040
8	128	40320
9	256	362880
10	512	3628800
11	1024	39916800
12	2048	479001600
13	4096	6227020800
14	8192	87178291200
15	16384	1307674368000
16	32768	20922789888000
17	65536	355687428096000
18	131072	6402373705728000
19	262144	121645100408832000
20	524288	2432902008176640000

- Other query shapes (star, clique, various) have similar behavior

- Hence: avoid cartesian products

Left-Deep, No C.P.

- Let n = number of relations to join
- For s = 1, n do:
  - For each **connected** S of of size s do:
    - $S = \{R\} \cup S'$, where S' is **connected**
    - Lookup Cost(S')
    - Cost(S) := $\min_{\text{splits}}$ (Cost(S') + cost-of($R \bowtie S'$))
    - Memorize (S, Cost(S))
- Return Cost(All-relations)

Analysis for **Chain Query**

- Let n = number of relations to join
- For s = 1, n do:
  - For each **connected** S of of size s do:

    $= \{R_i, R_{i+1}, \ldots, R_{i+s-1}\}$

    - $S = \{R\} \cup S'$, where S' is **connected**
    - Lookup Cost(S')
    - Cost(S) := $\min_{\text{splits}}$ (Cost(S') + cost-of($R \bowtie S'$))
    - Memorize (S, Cost(S))
- Return Cost(All-relations)

Analysis for **Chain Query**

- Let n = number of relations to join
- For s = 1, n do:
  - For each **connected** S of of size s do:
    - $S = \{R\} \cup S'$, where S' is **connected**
    - Lookup Cost(S')
    - Cost(S) := $\min_{\text{splits}}$ (Cost(S') + cost-of($R \bowtie S'$))
    - Memorize (S, Cost(S))
- Return Cost(All-relations)

$= \{R_i, R_{i+1}, \ldots, R_{i+s-1}\}$

Either $R = R_i$
or $R = R_{i+s-1}$

Analysis for **Chain Query**

- Let n = number of relations to join
- For s = 1, n do:
  - For each **connected** S of of size s do:
    - $S = \{R\} \cup S'$, where S' is **connected**
    - Lookup Cost(S')
    - Cost(S) := $\min_{\text{splits}}$ (Cost(S') + cost-of($R \bowtie S'$))
    - Memorize (S, Cost(S))

$= \{R_i, R_{i+1}, \ldots, R_{i+s-1}\}$

Either $R = R_i$
or $R = R_{i+s-1}$

Runtime:

$$\sum_{s=1}^{n} (n - s + 1) \cdot 2 = n(n+1) = O(n^2)$$

Paper: "Analysis of Two…"

- Bushy plans, no cartesian product
- Challenge: how do we enumerate efficiently the pairs S_1, S_2 where:
  - S_1, S_2 are disjoint
  - Each of S_1, S_2 is connected
  - S_1 and S_2 are connected to each other
- Solution:
  - EnumerateCSG and EnumerateCMP
  - EnumerateCMP had a bug, reported in an Errata

Discussion

- Dynamic programming can be faster than exploring all possible plans
- Still, it runs in exponential time
- Database systems have a limit (configurable) of how many tables they optimize using dynamic programming; beyond that, they use some heuristics
- Outer-, anti- joins add extra complexity

Subqueries in SQL

Subqueries

- Subquery in SELECT:
  - Must return single value
- Subquery in FROM
  - Like a temporary relation
  - Alternative: use the WITH clause
- Subquery in WHERE or in HAVING
  - Can express sophisticated queries

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

Subquery in SELECT

Compute the number of products sold by each supplier

```
SELECT x.sno, x.sname,
      (SELECT count(*)
       FROM Supply y
       WHERE x.sno = y.sno)
FROM  Supplier x
```

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

Subquery in FROM

Better: use the WITH statement!

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

Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

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

Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname)
```

For each supplier, compute how many parts they supply

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

Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
```

Find the maximum

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

Subquery in FROM

Better: use the WITH statement!

Find the supplier who supplies the maximum number of parts

```sql
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

Find the “witness”, i.e. the supplier that supplies the maximum number of parts; **argmax**

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

Subquery in FROM

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

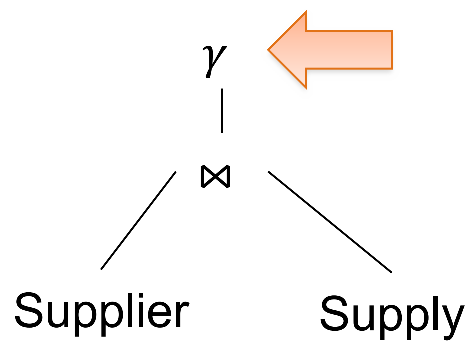
Query Plan:

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

Subquery in FROM

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

Query Plan:

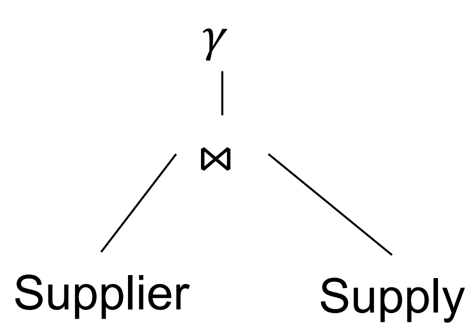


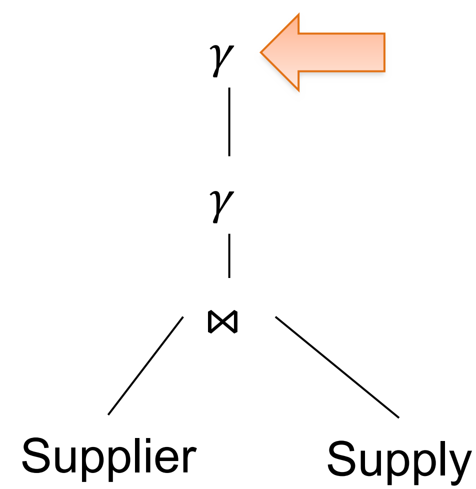
```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subquery in FROM

```sql
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

Query Plan:





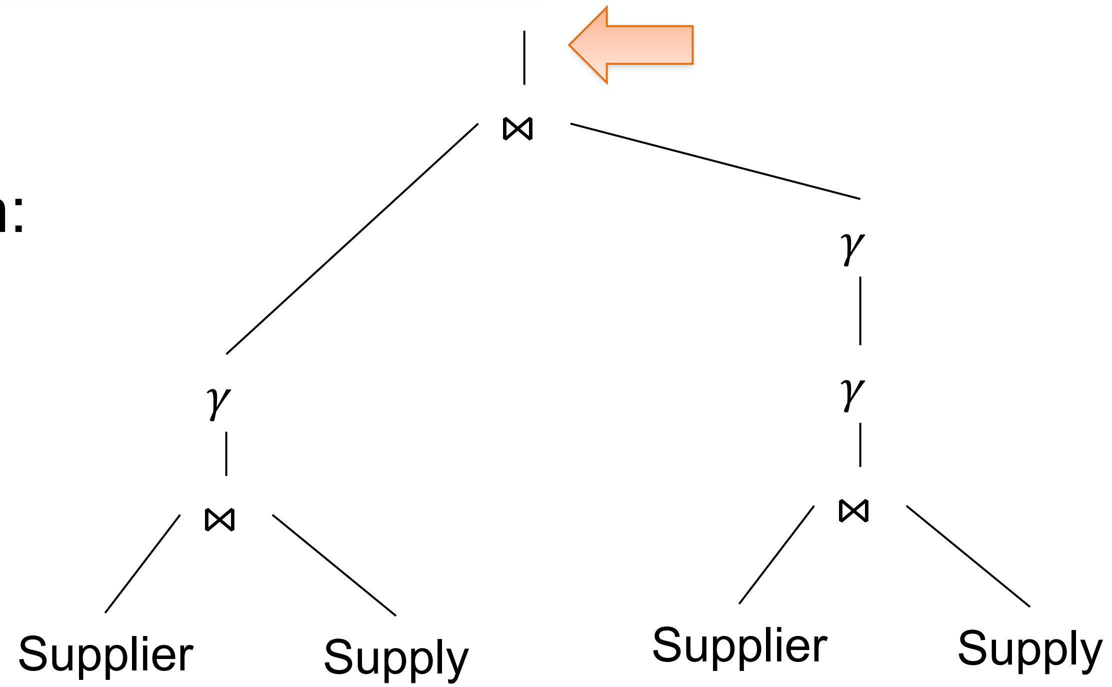
Needs to be a tree!

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

Subquery in FROM

```
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

Query Plan:



Needs to be a tree!

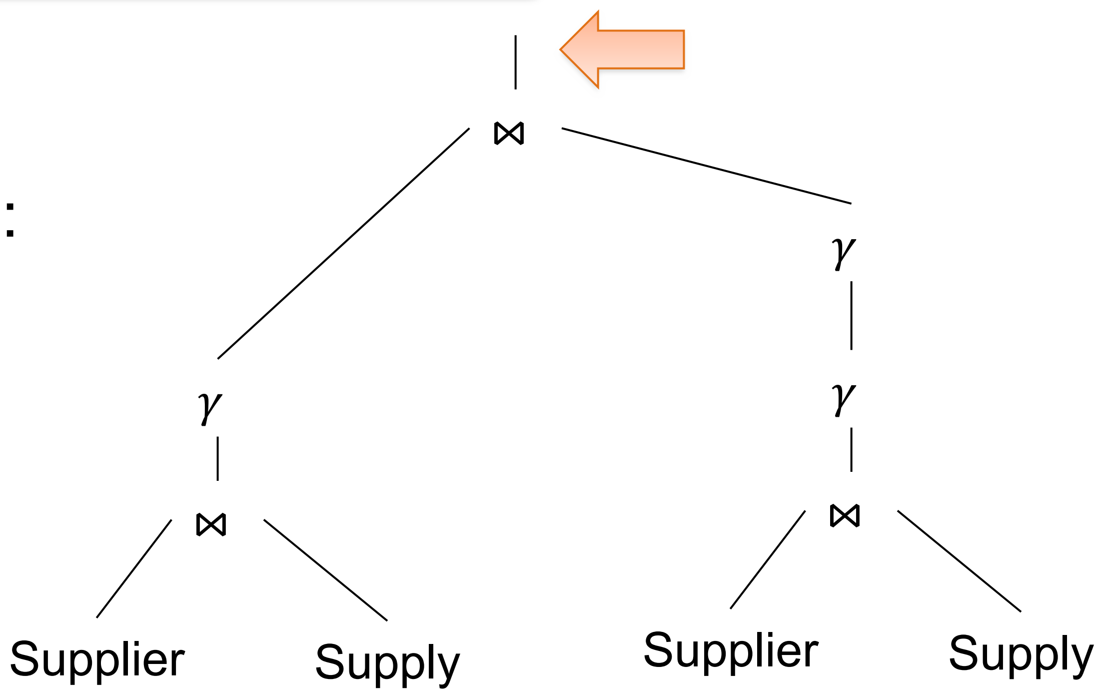
```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subquery in FROM

```sql
WITH Cnt AS (SELECT x.sno, x.sname, count(*) as c
             FROM Supplier x, Supply y
             WHERE x.sno = y.sno
             GROUP BY x.sno, x.sname),
     Mx AS (SELECT max(c) as m
            FROM Cnt)
SELECT z.sno, z.sname, m.m
FROM  Cnt z, Mx m
WHERE z.c = m.m;
```

In Relational Algebra it is easy to compose operations

Query Plan:



Needs to be a tree!

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

Subquery in WHERE

Find suppliers that supply some ‘blue’ parts

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

Subquery in WHERE

Find suppliers that supply some 'blue' parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor = 'blue');
```

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

Subquery in WHERE

Find suppliers that supply some ‘blue’ parts

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor = ‘blue’);
```

A *correlated* subquery:
meaning that it depends on
the variable x defined by
the outer query

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

Subquery in WHERE

Find suppliers that supply *only* ‘red’ parts

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

Subquery in WHERE

Find suppliers that supply *only* ‘red’ parts

Find the *other* suppliers

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

Subquery in WHERE

Find suppliers that supply *only* 'red' parts

Find the *other* suppliers

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor != ‘red’);
```

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

Subquery in WHERE

Find suppliers that supply *only* ‘red’ parts

Find the *other* suppliers

```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor != ‘red’);
```

```
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT * FROM Supply y, Part z
                  WHERE x.sno=y.sno
                    and y.pno=z.pno
                    and z.pcolor != ‘red’);
```

Negate to get the right ones

Relational Algebra

- Semijoin: R ⋉ S
  - Subset of R that joins with S
  - $R \ltimes S = \Pi_{Attrs(R)}(R \bowtie S)$

- Anti-semijoin: R▷S
  - Subset of R that does not join with S
  - $R \rhd S = R - (R \ltimes S)$

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

Semi-Join

Find suppliers that supply some ‘blue’ parts

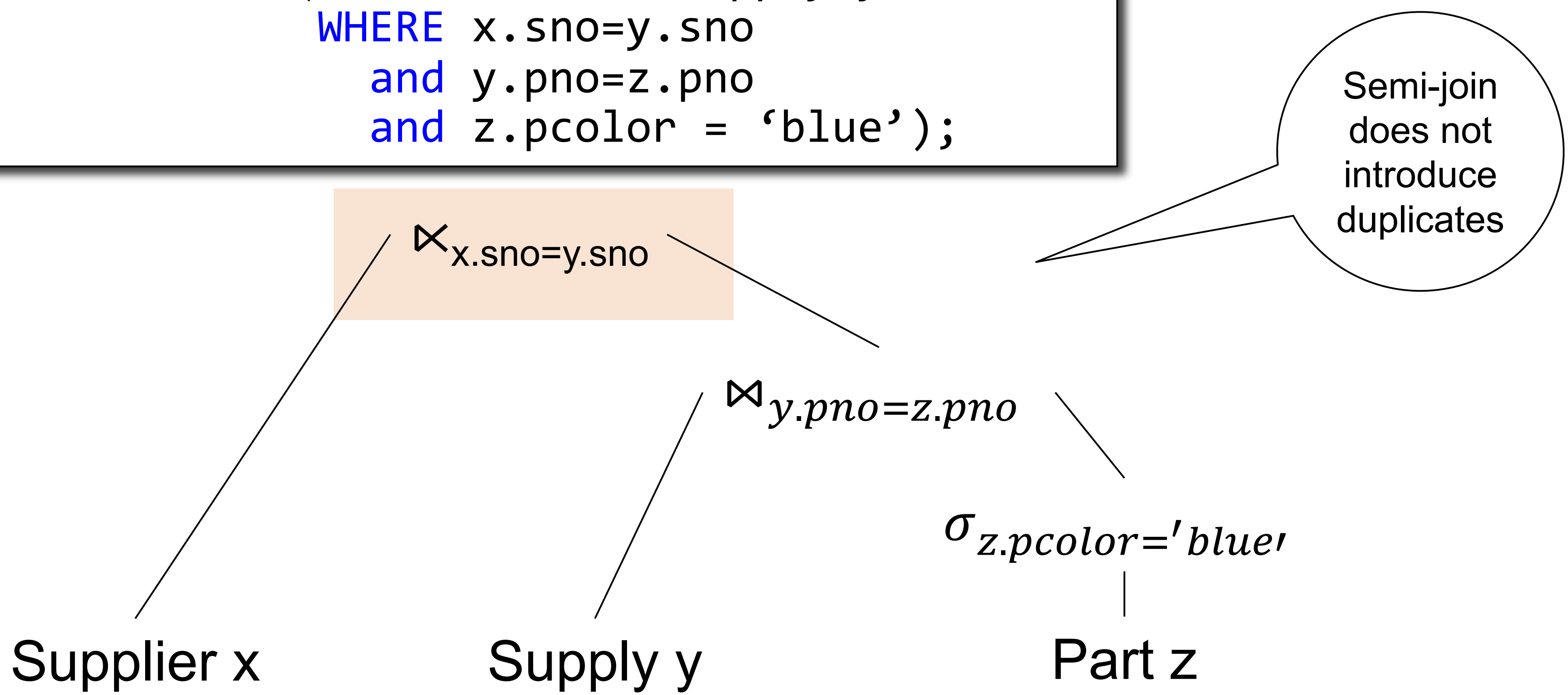
```
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor = ‘blue’);
```

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

Semi-Join

Find suppliers that supply some ‘blue’ parts

```sql
SELECT x.sno
FROM Supplier x
WHERE exists (SELECT * FROM Supply y, Part z
              WHERE x.sno=y.sno
                and y.pno=z.pno
                and z.pcolor = ‘blue’);
```



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

Anti-semi-Join

Find suppliers that supply *only* 'red' parts

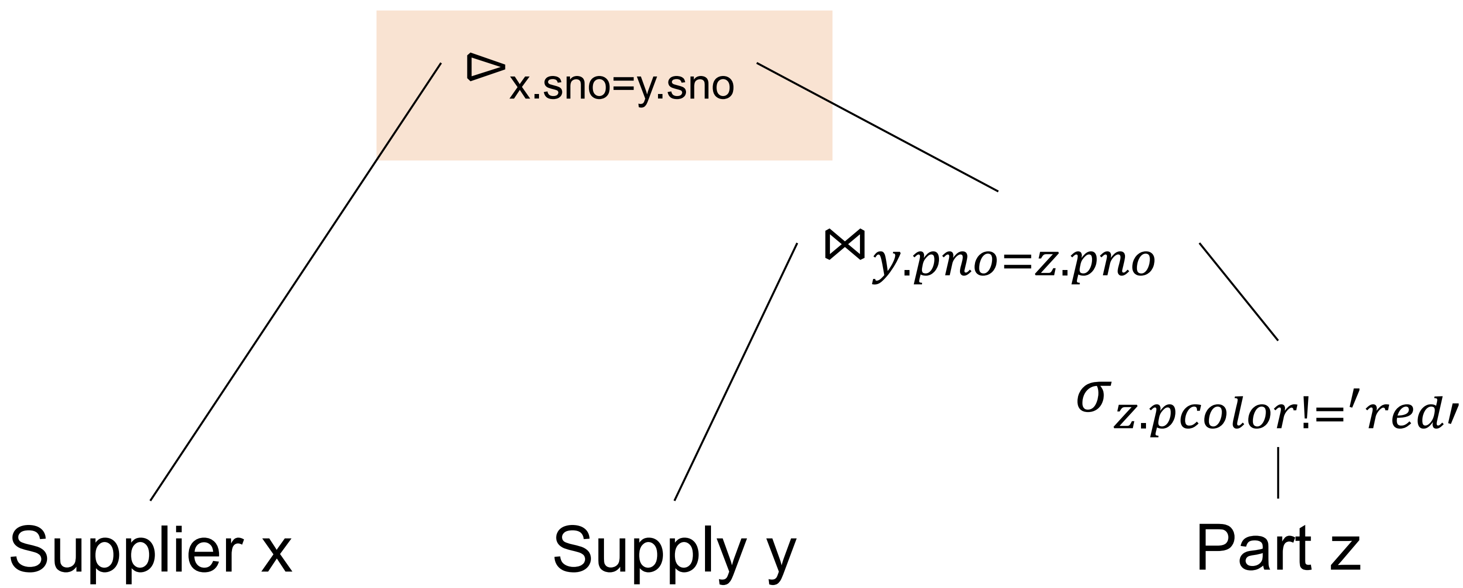
```
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT * FROM Supply y, Part z
                  WHERE x.sno=y.sno
                    and y.pno=z.pno
                    and z.pcolor != 'red');
```

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

Anti-semi-Join

Find suppliers that supply *only* 'red' parts

```
SELECT x.sno
FROM Supplier x
WHERE not exists (SELECT * FROM Supply y, Part z
                  WHERE x.sno=y.sno
                    and y.pno=z.pno
                    and z.pcolor != 'red');
```

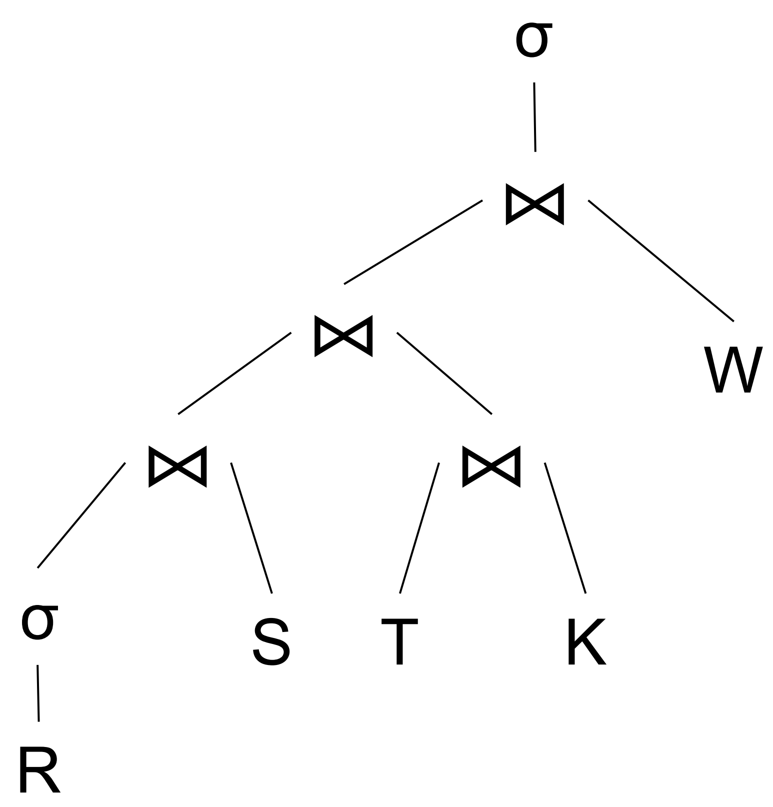


Discussion

- RA does not have variables, no nested expressions
  - Correlated subqueries need to be *decorrelated* first
  - Nested subqueries then need to be *unnested*
- Some systems fail to unnest complicated queries: nested loop join

Operator Interface

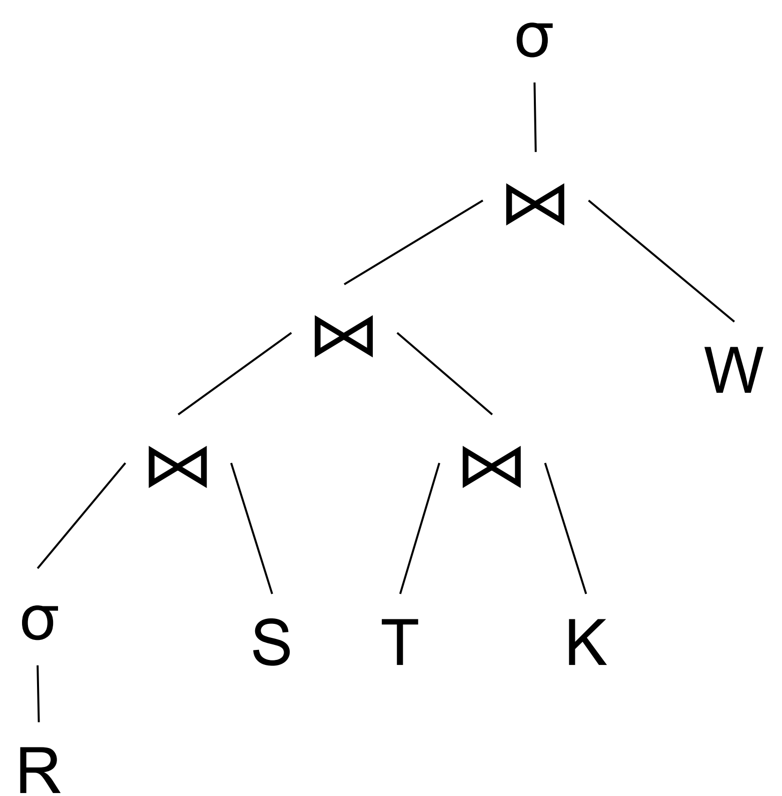
How Do We Combine Them?



How Do We Combine Them?

Option 1:
materialize intermediate results

Option 2:
Pipeline tuples btw. ops

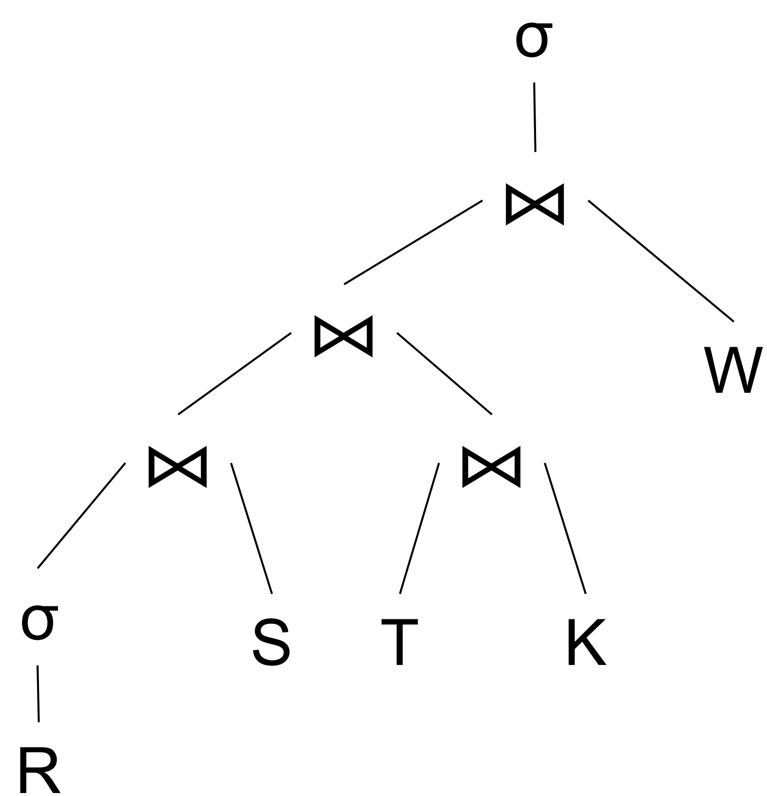


How Do We Combine Them?

Option 1:
materialize intermediate results

Option 2:
Pipeline tuples btw. ops

Implementation:
Iterator Interface



Operator Interface

Volcano model:

- open(), next(), close()
- Pull model
- Volcano optimizer: G. Graefe's (Wisconsin) → SQL Server
- Supported by most DBMS today
- Will discuss next

Operator Interface

Volcano model:

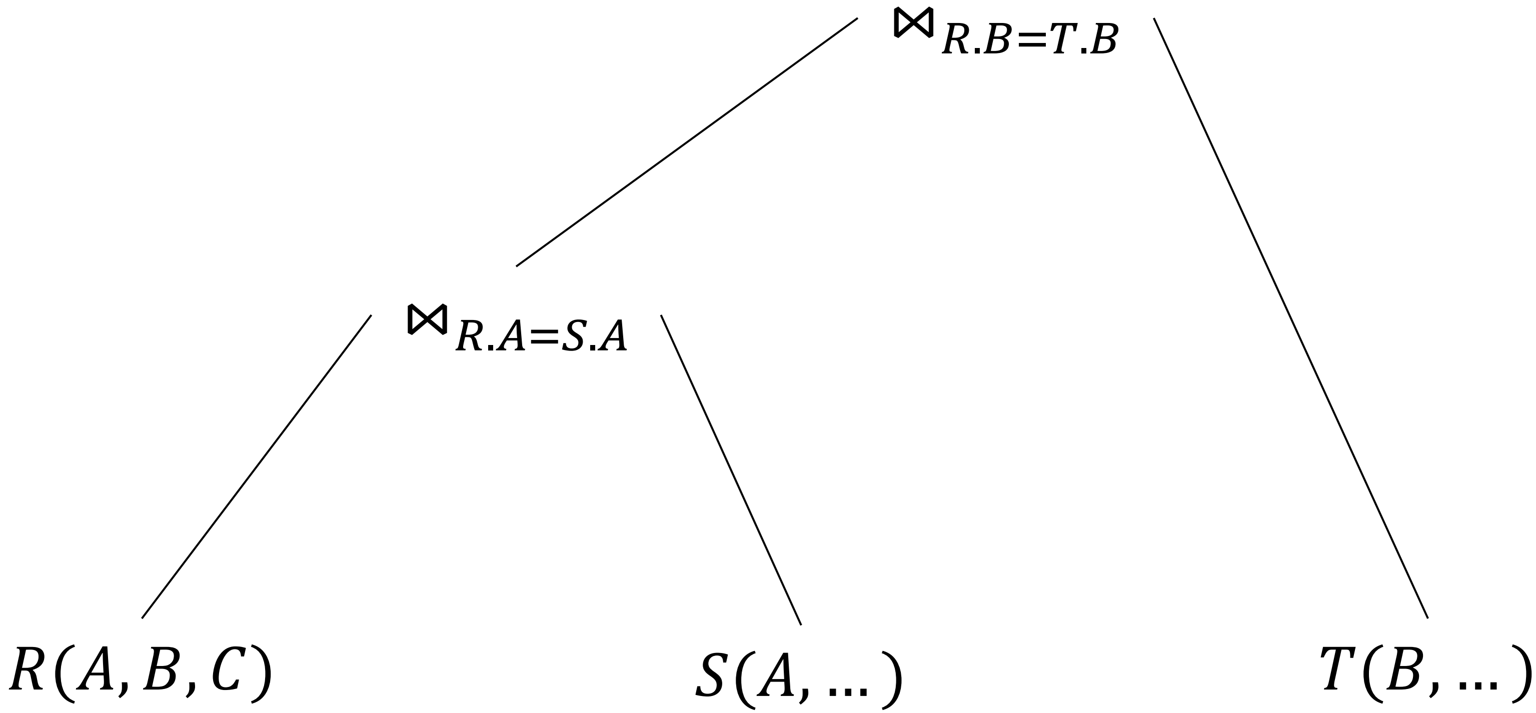
- open(), next(), close()
- Pull model
- Volcano optimizer: G. Graefe's (Wisconsin) → SQL Server
- Supported by most DBMS today

Data-driven model:

- open(),produce(), consume(),close()
- Push model
- Introduced by Thomas Neumann in Hyper (at TU Munich), later acquired by Tableau

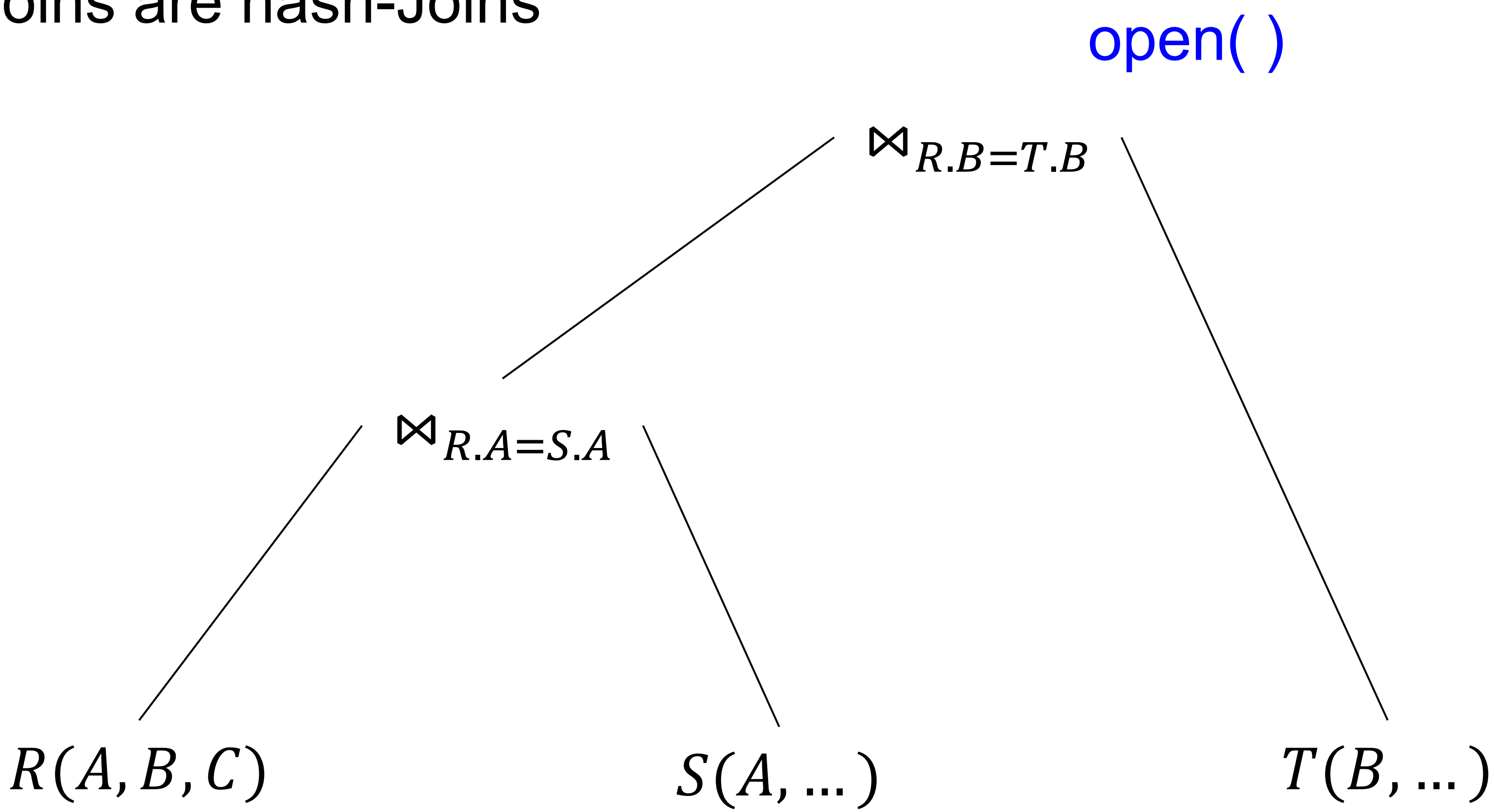
Operator Interface

Both joins are hash-Joins



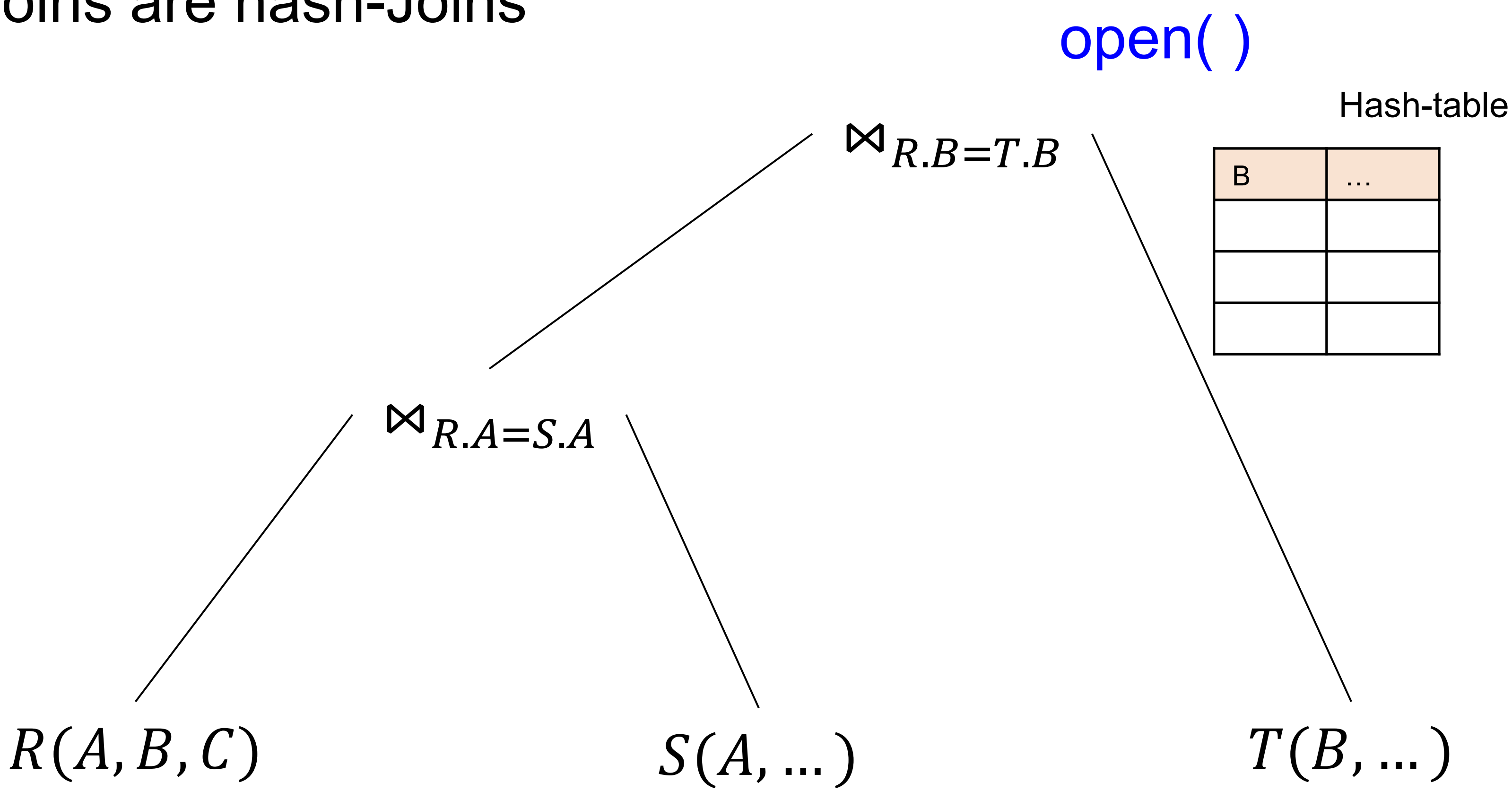
Operator Interface

Both joins are hash-Joins



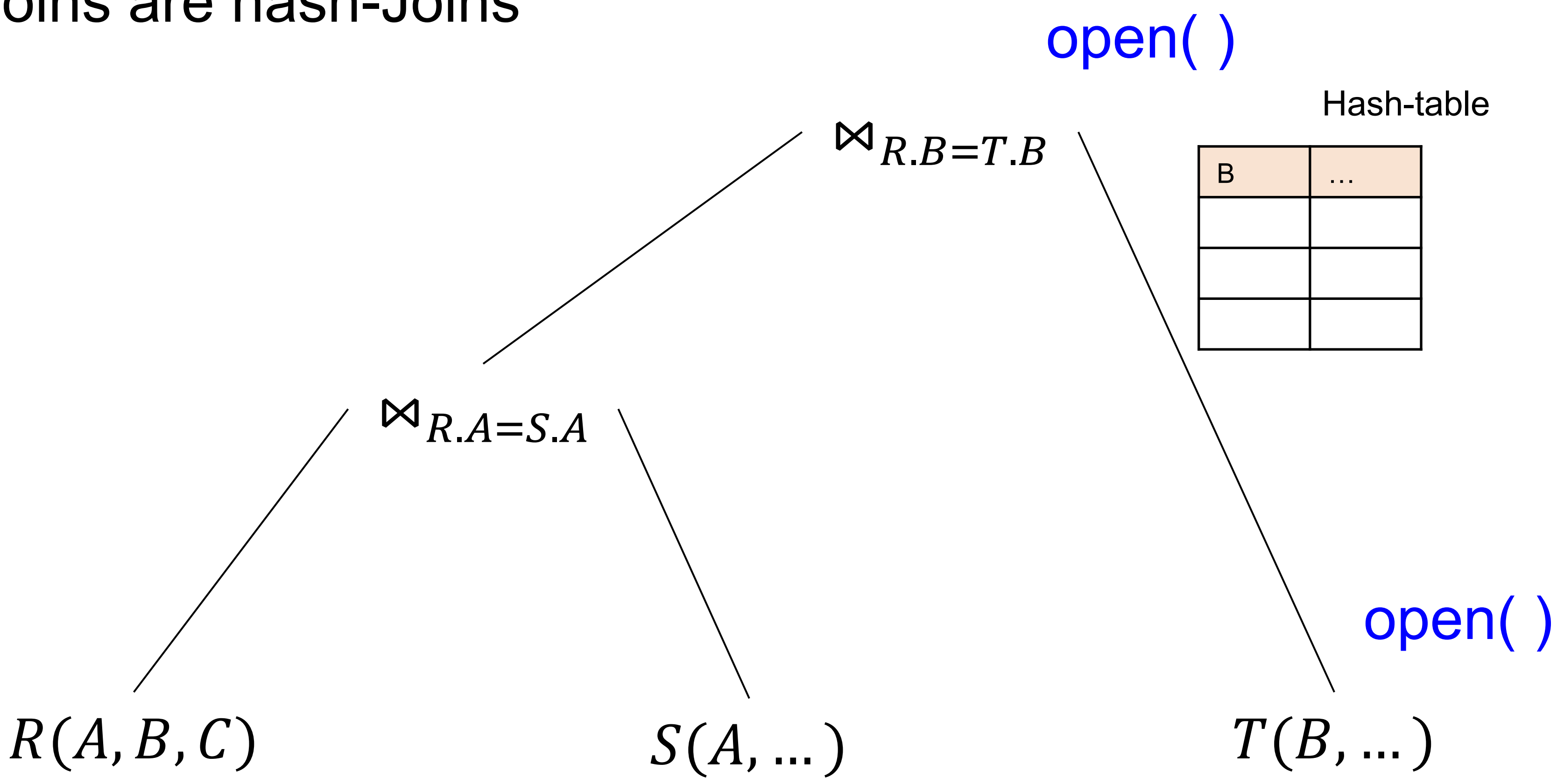
Operator Interface

Both joins are hash-Joins



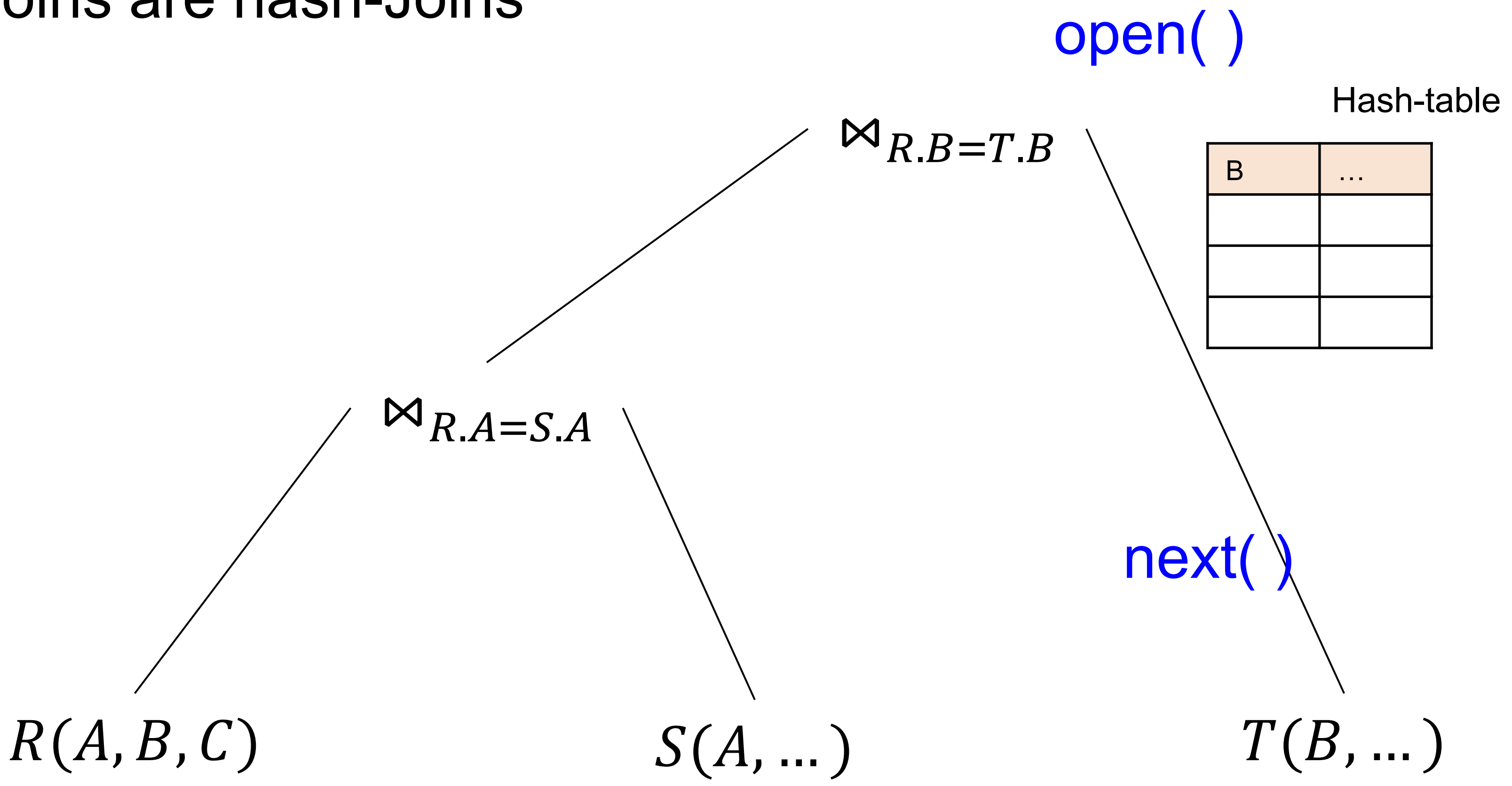
Operator Interface

Both joins are hash-Joins



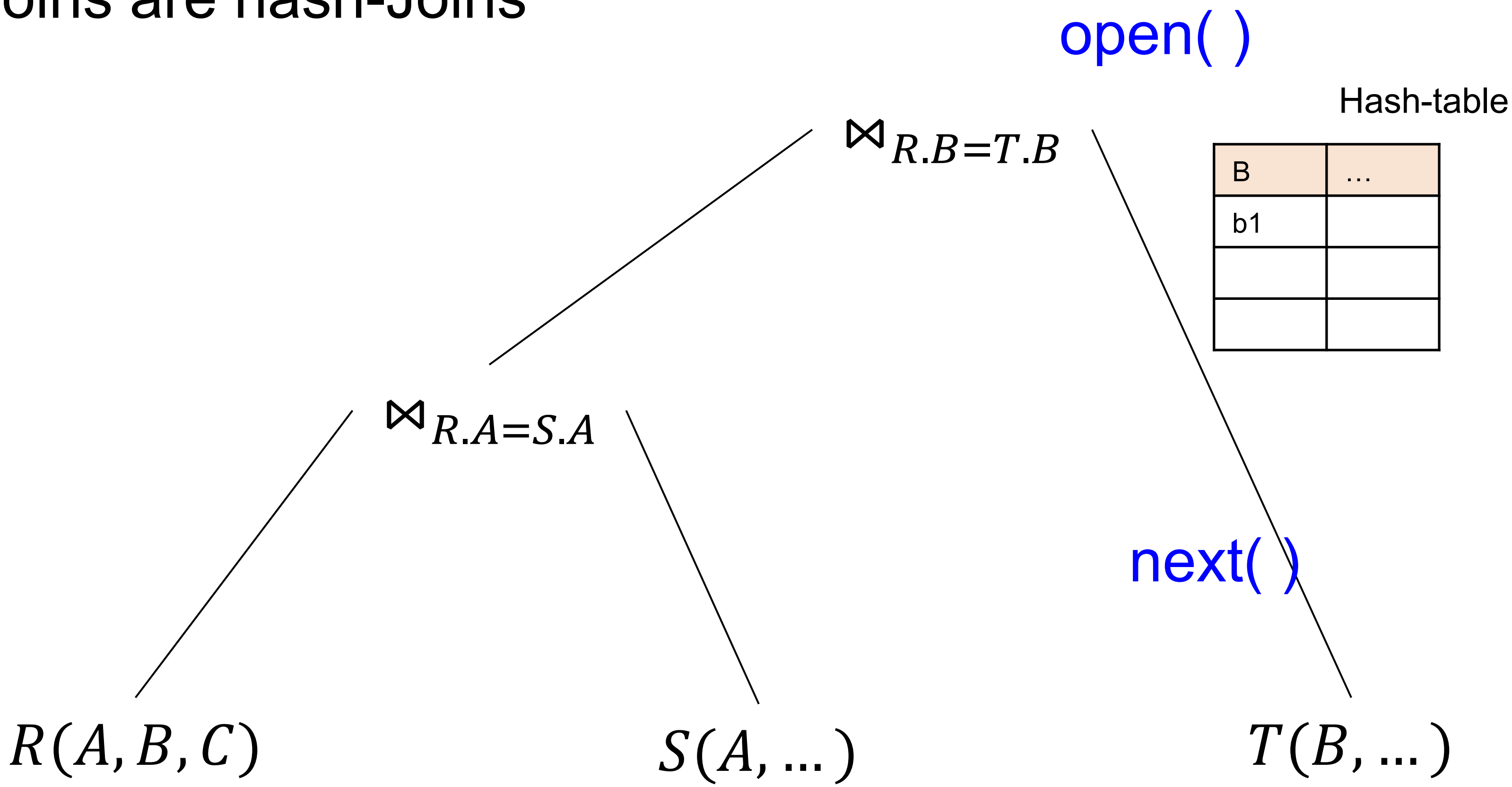
Operator Interface

Both joins are hash-Joins



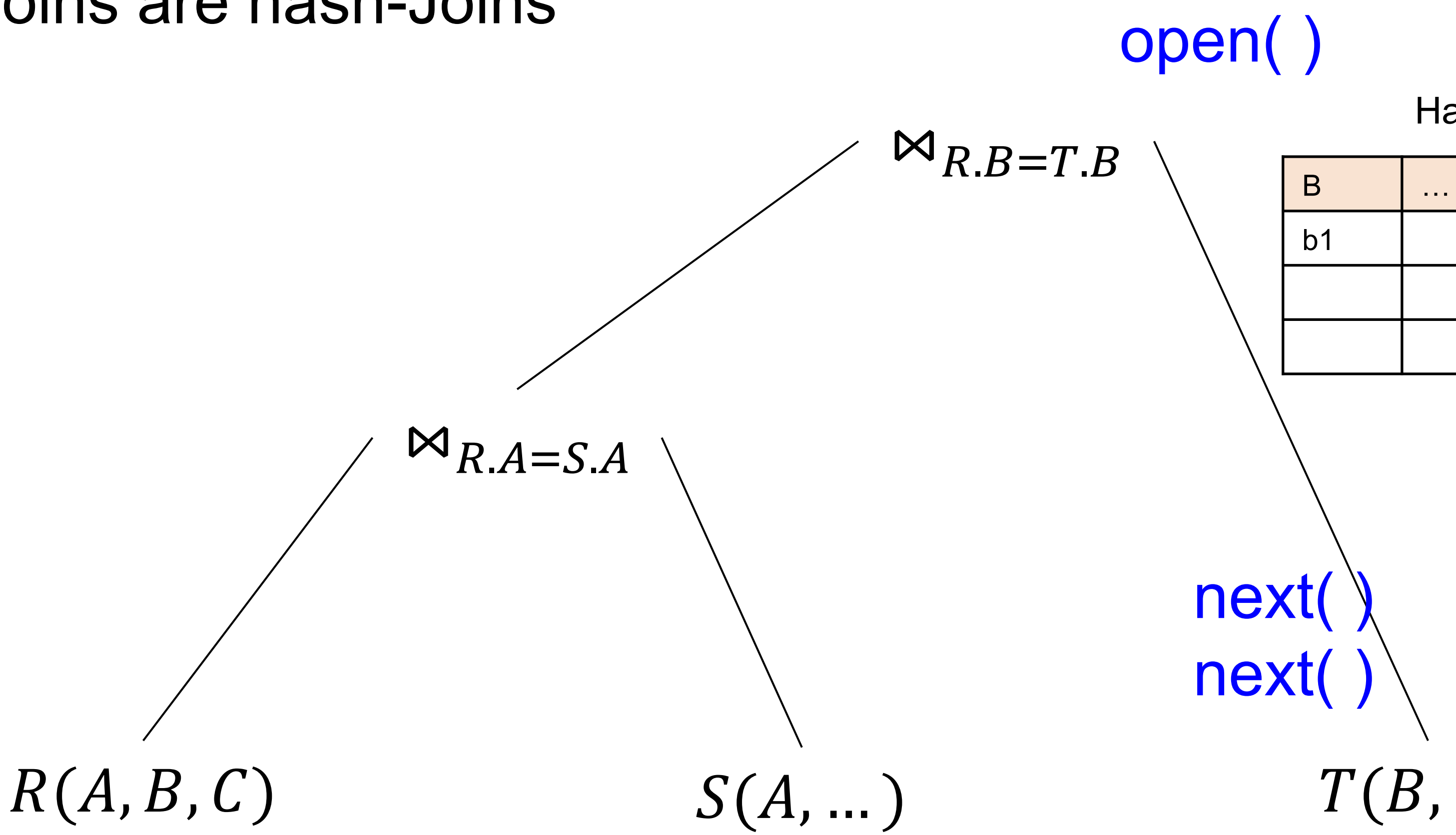
Operator Interface

Both joins are hash-Joins



Operator Interface

Both joins are hash-Joins

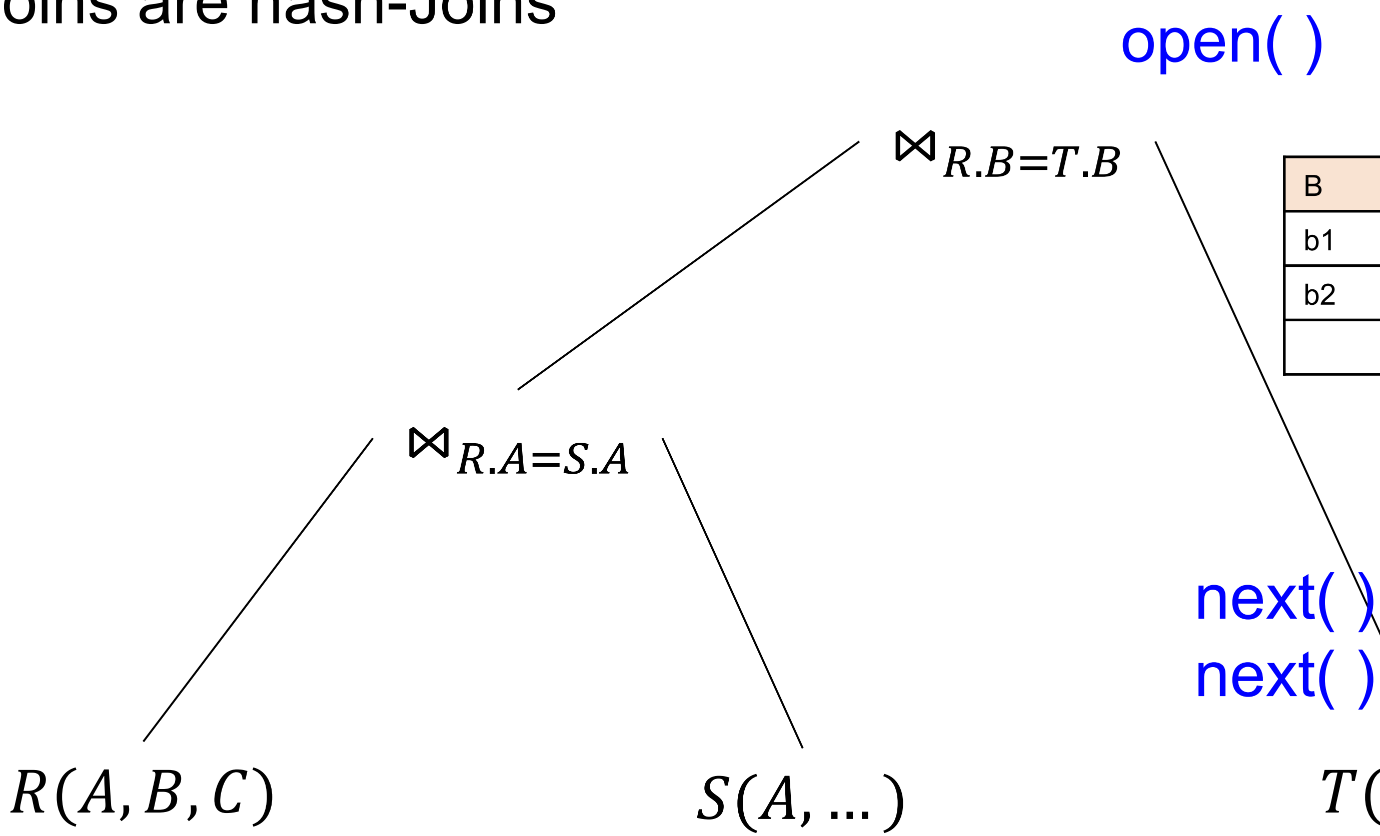


Hash-table

B	…
b1	

Operator Interface

Both joins are hash-Joins



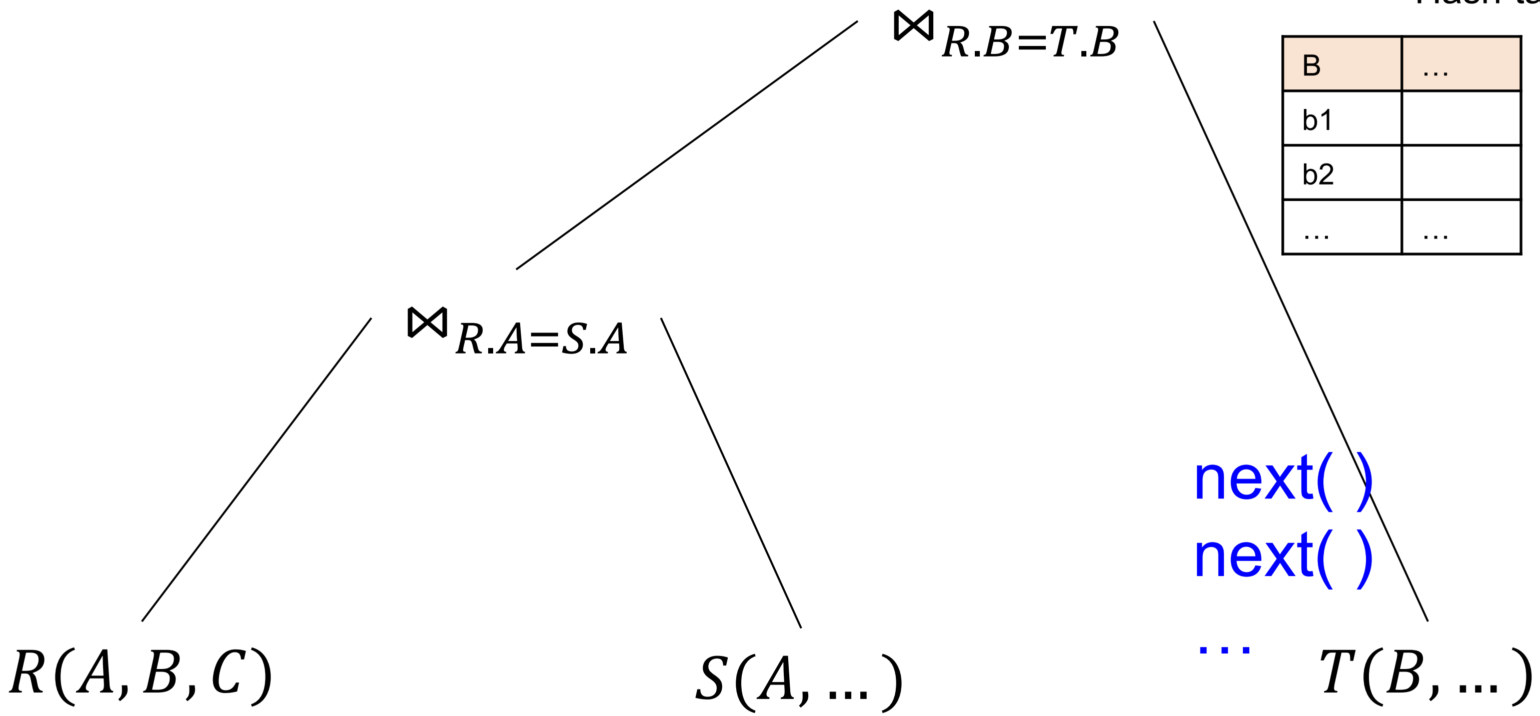
Hash-table

B	…
b1	
b2	

Operator Interface

Both joins are hash-Joins

open( )

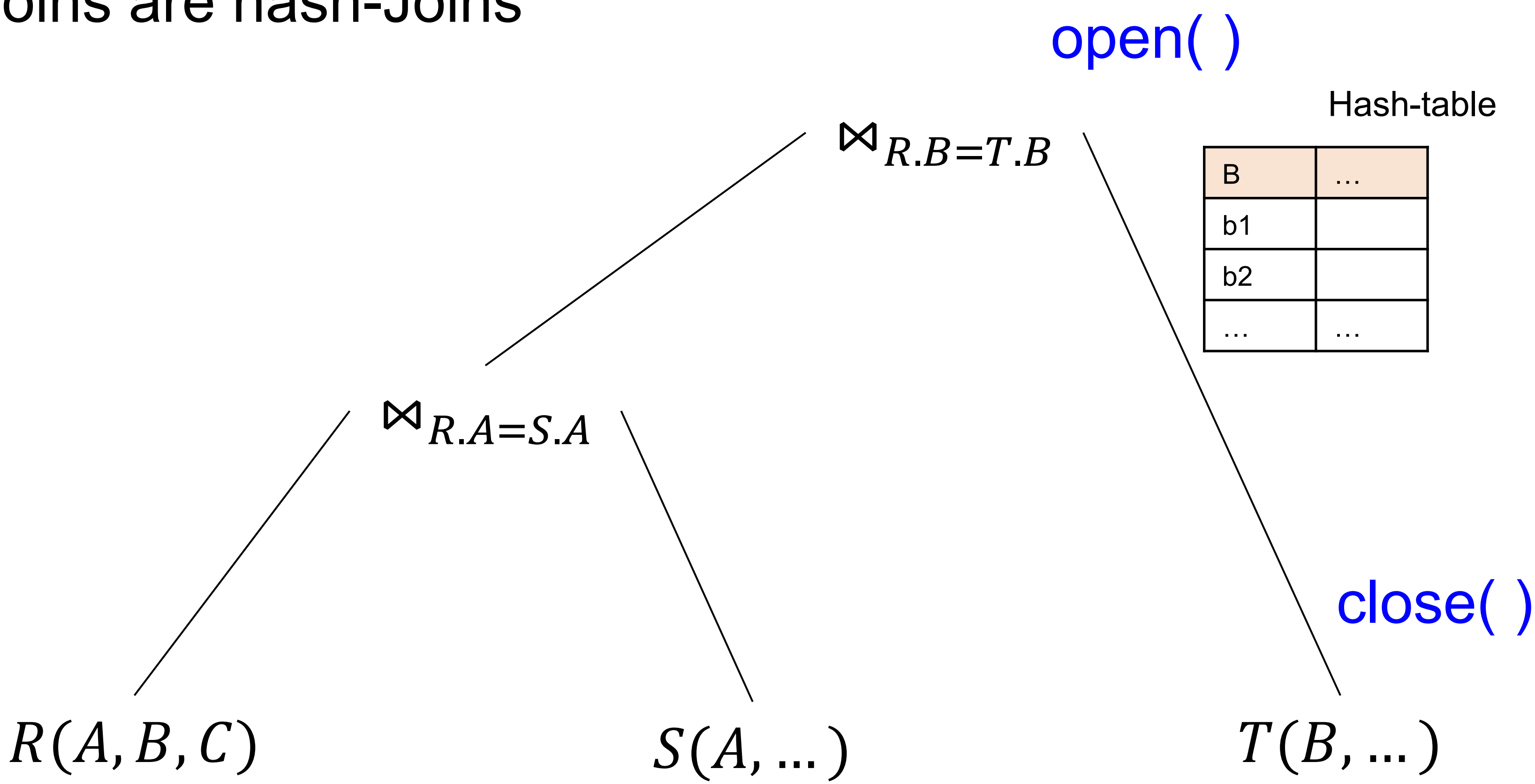


Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins

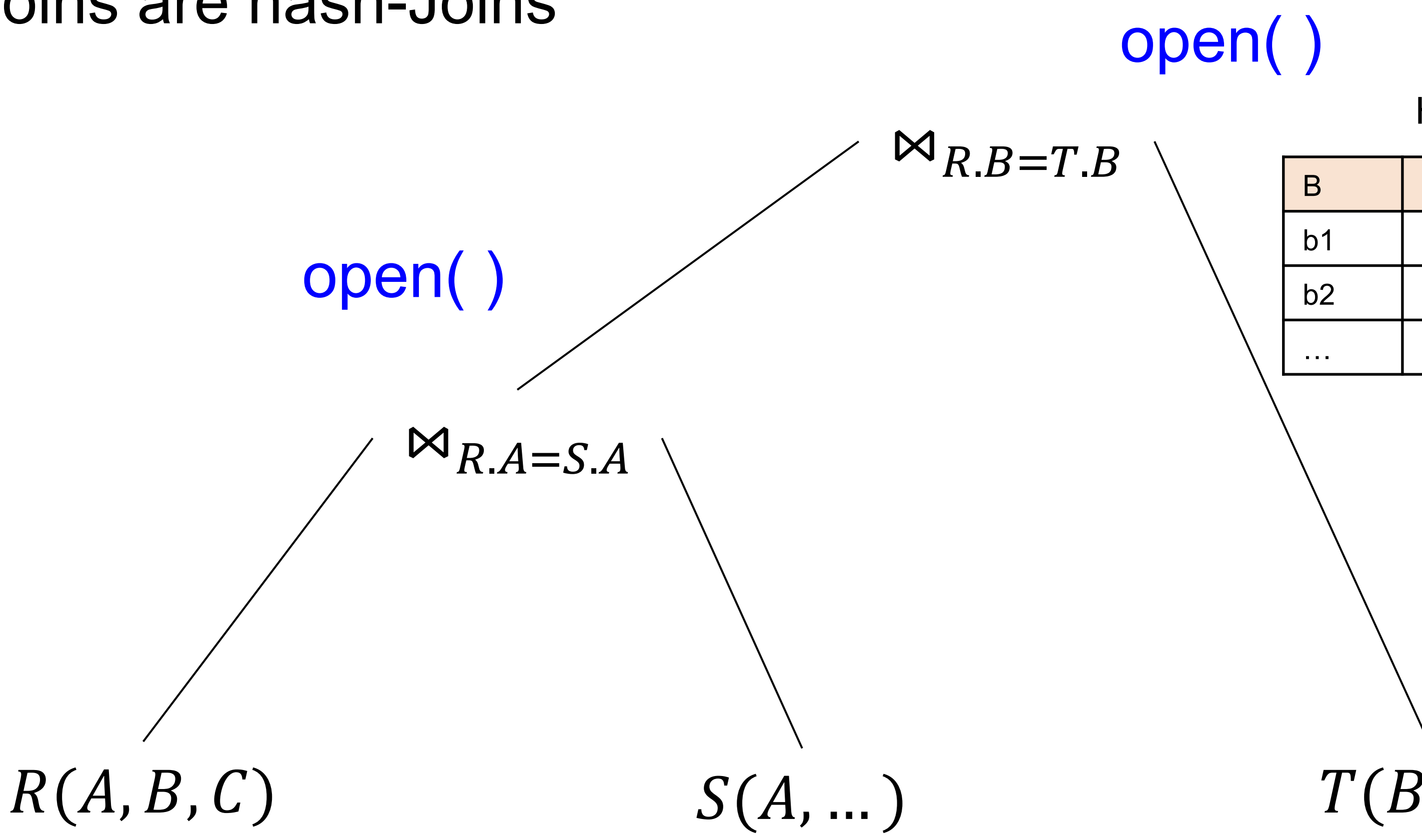


Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins

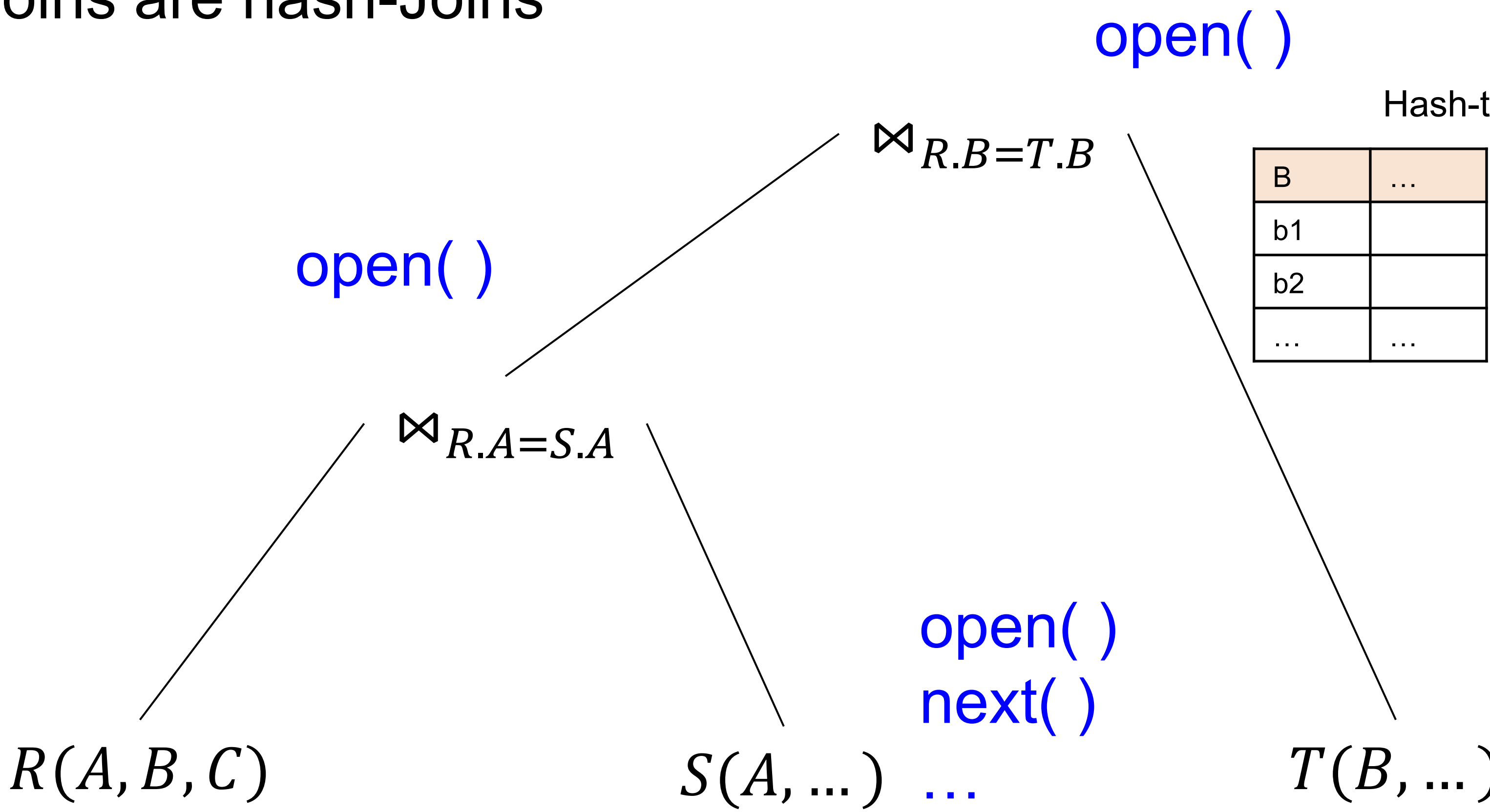


Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins

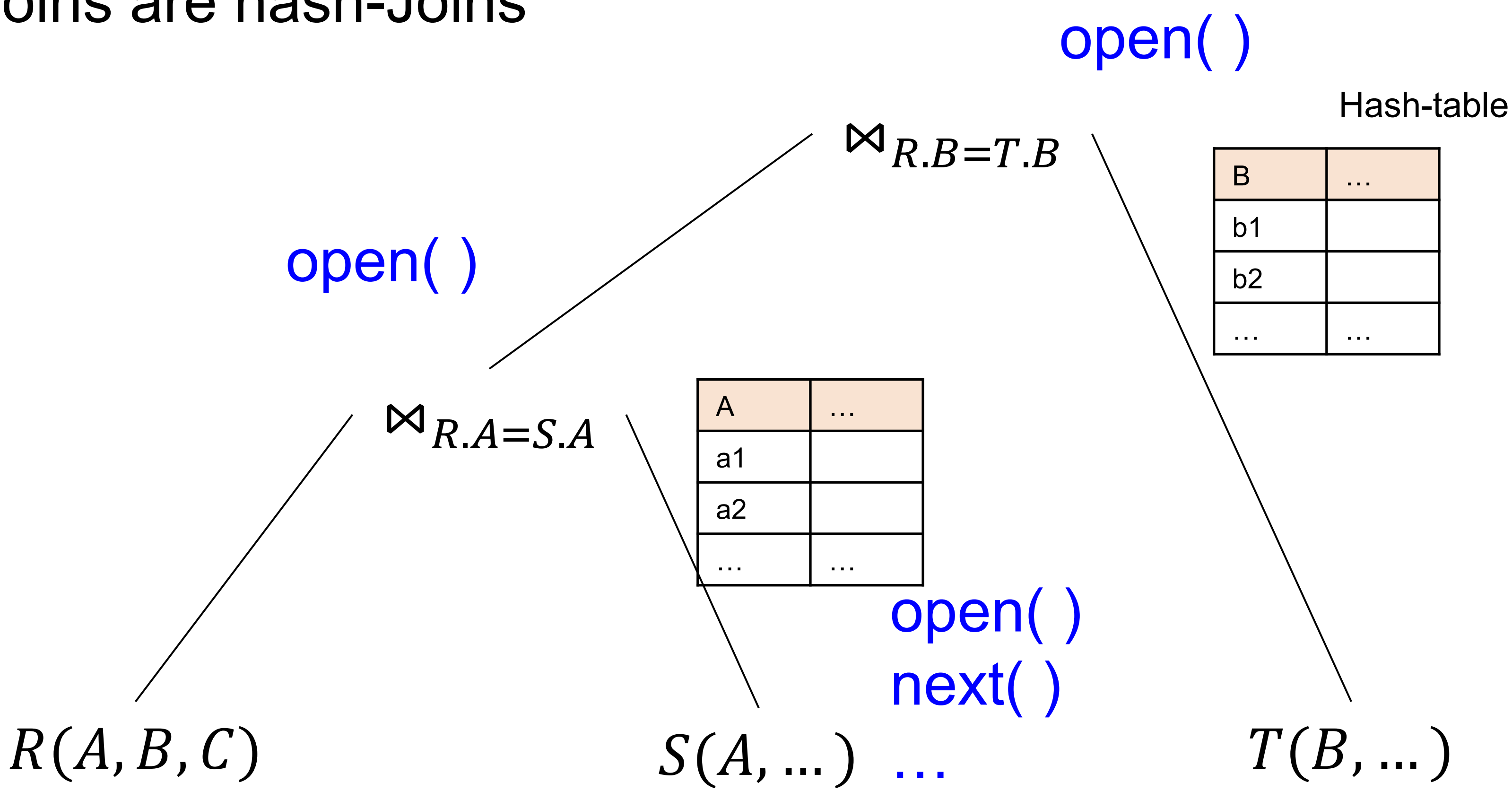


Hash-table

B	…
b1	
b2	
…	…

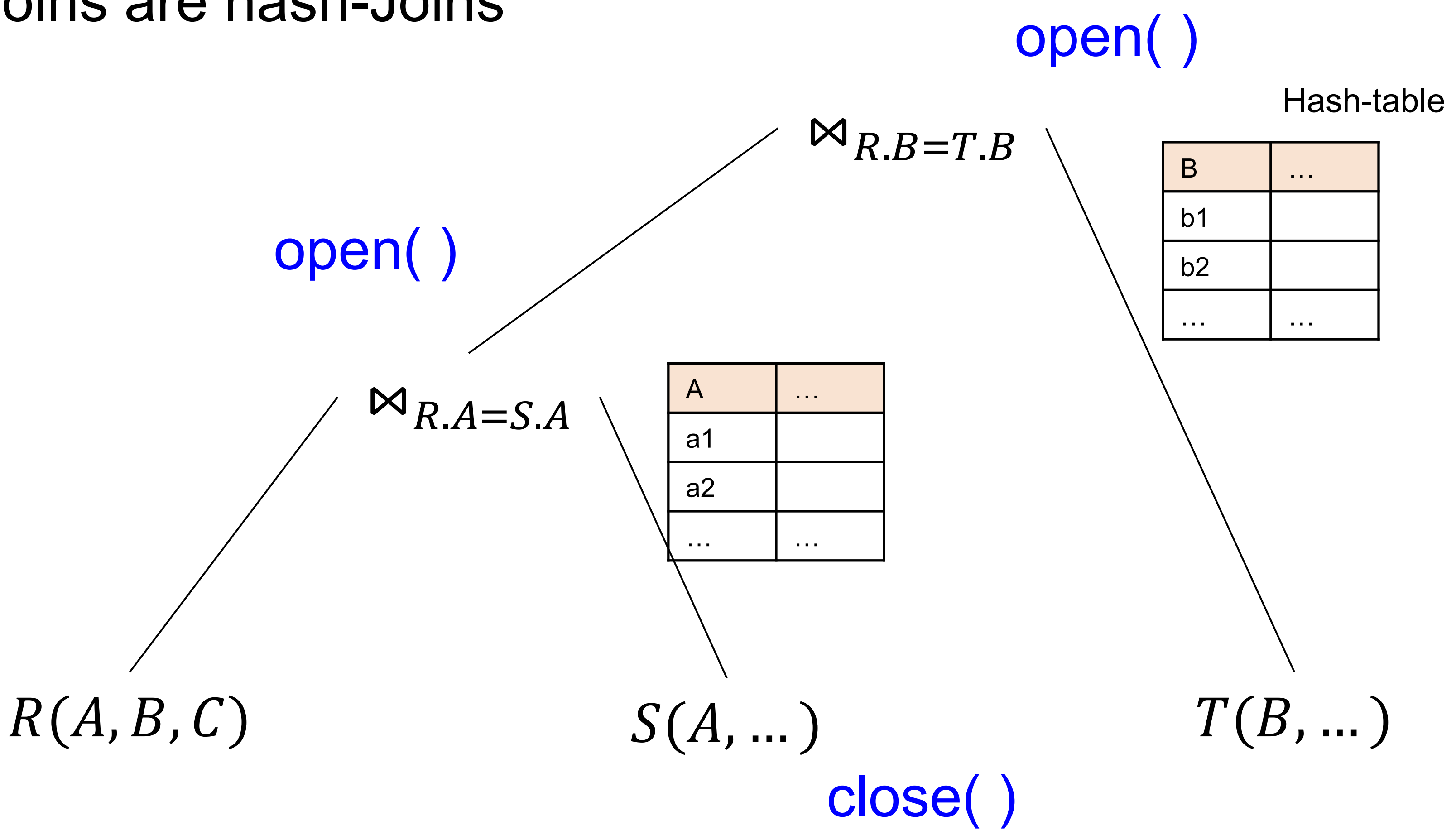
Operator Interface

Both joins are hash-Joins



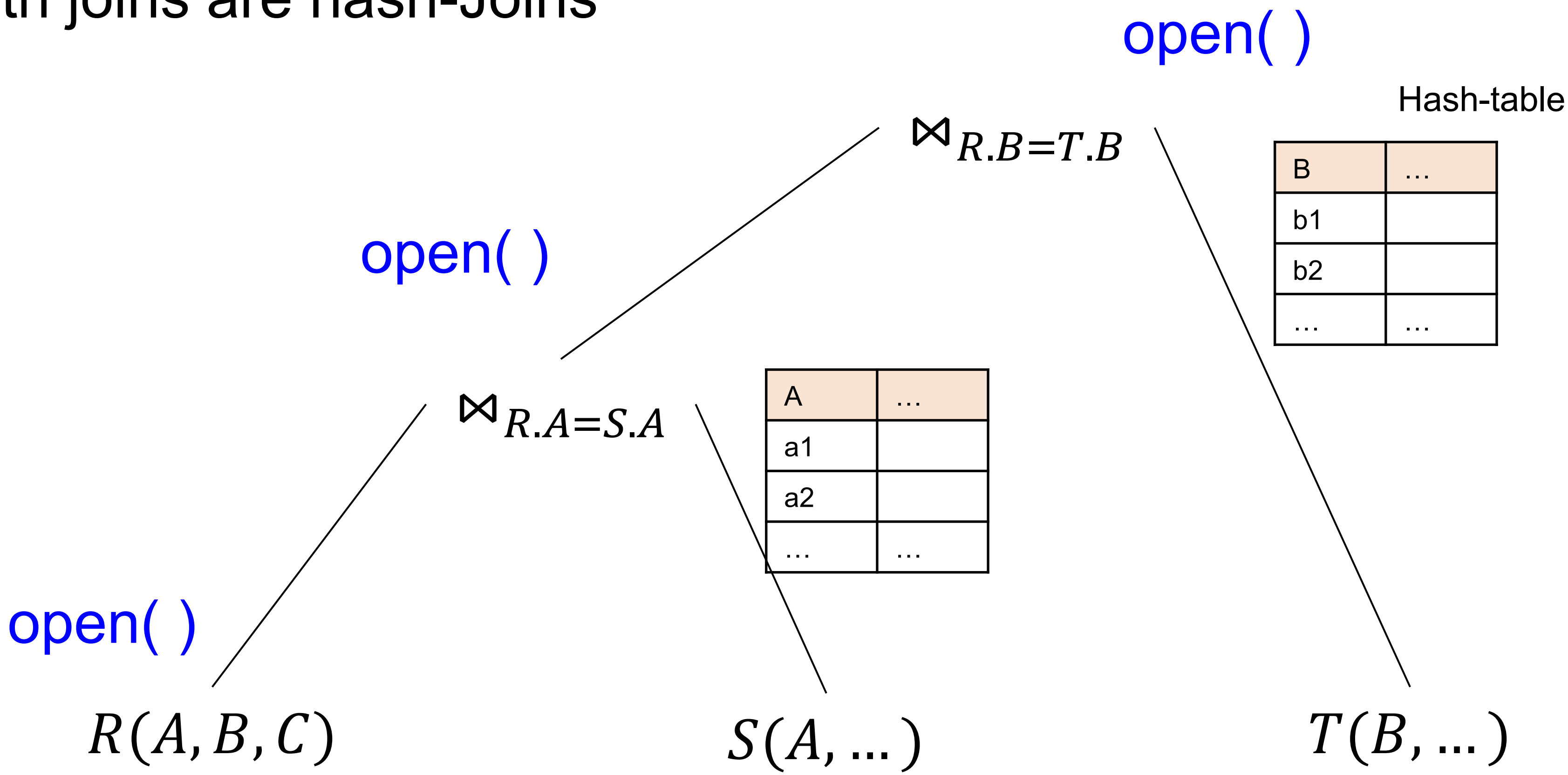
Operator Interface

Both joins are hash-Joins



Operator Interface

Both joins are hash-Joins



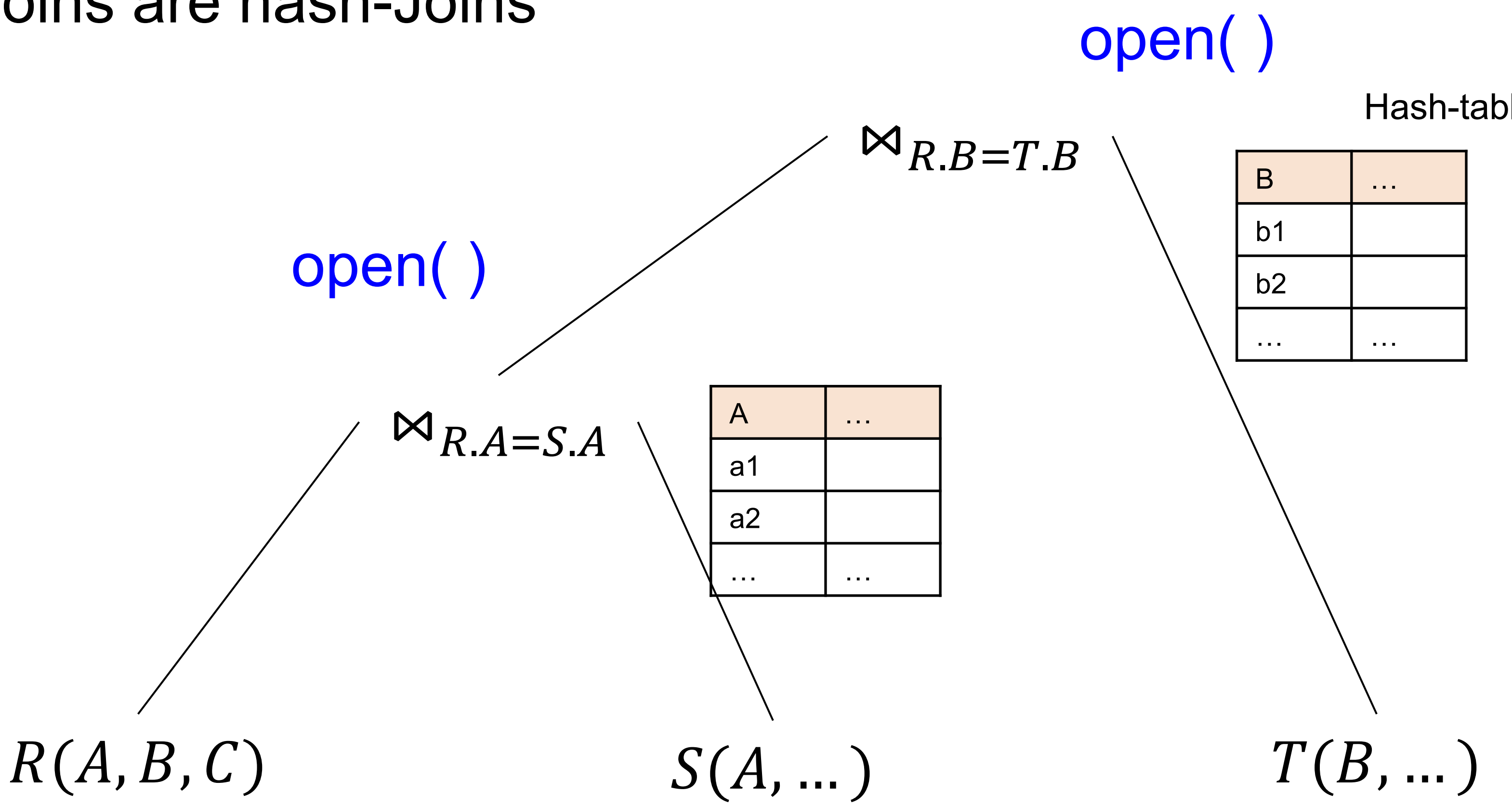
Hash-table

B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

Operator Interface

Both joins are hash-Joins



Hash-table

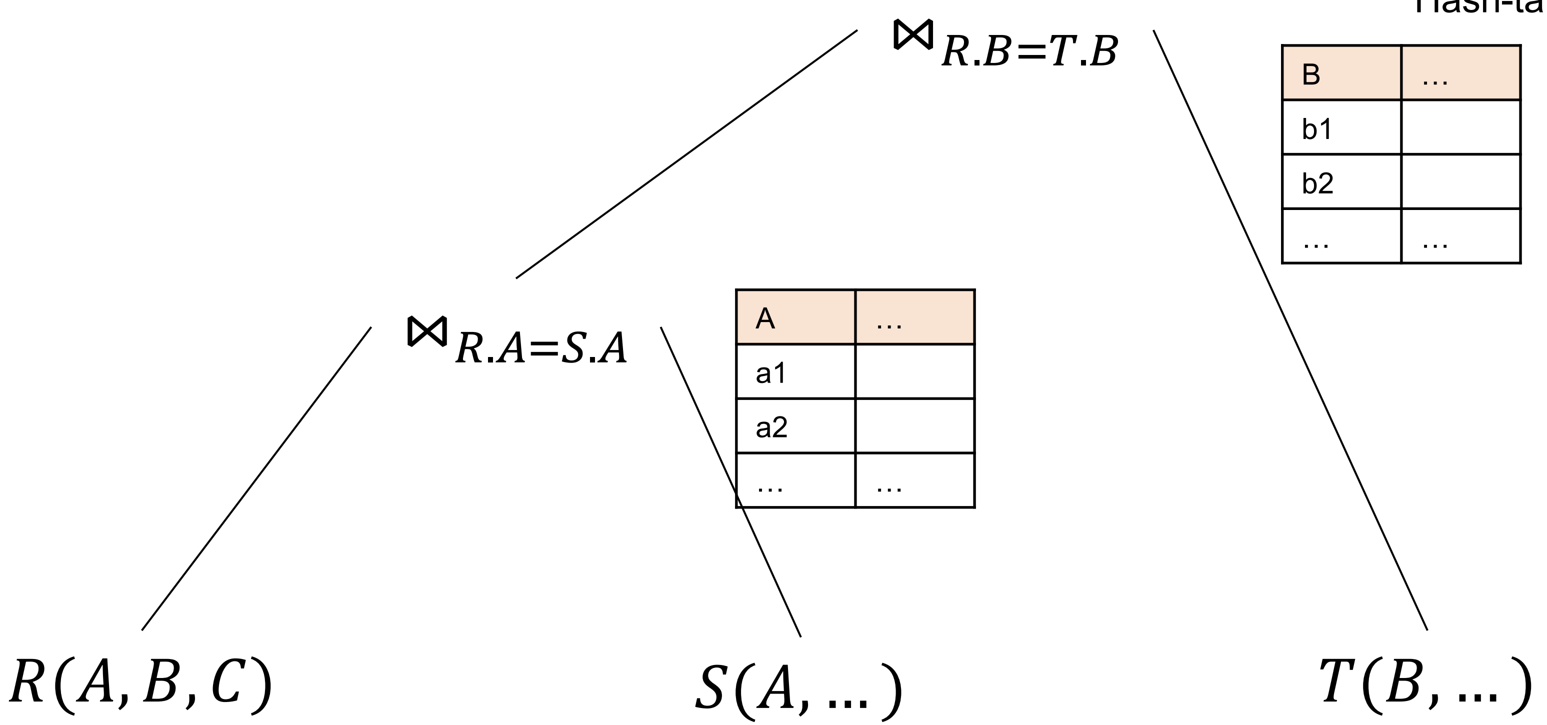
B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

Operator Interface

Both joins are hash-Joins

open( )



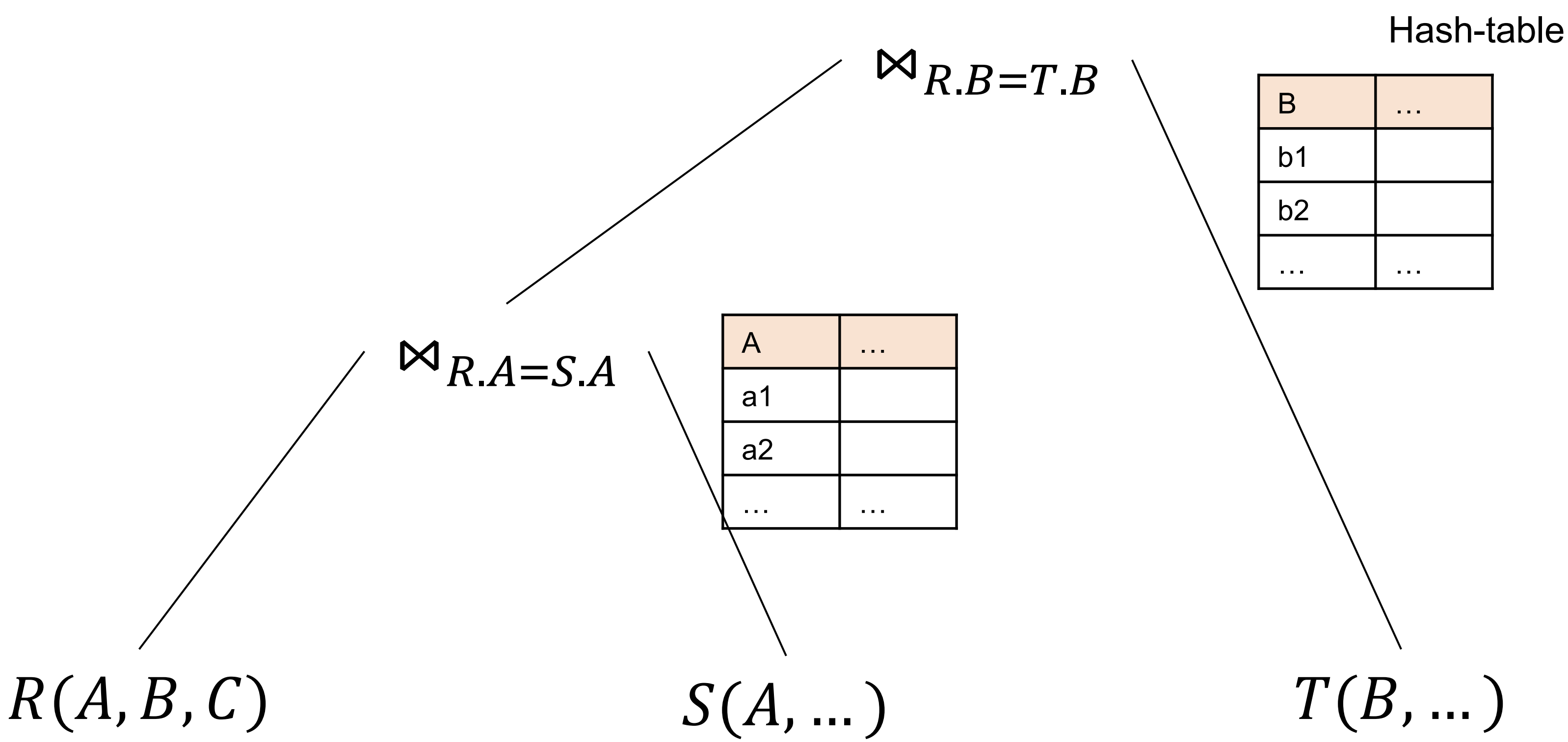
Hash-table

B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

Operator Interface

Both joins are hash-Joins

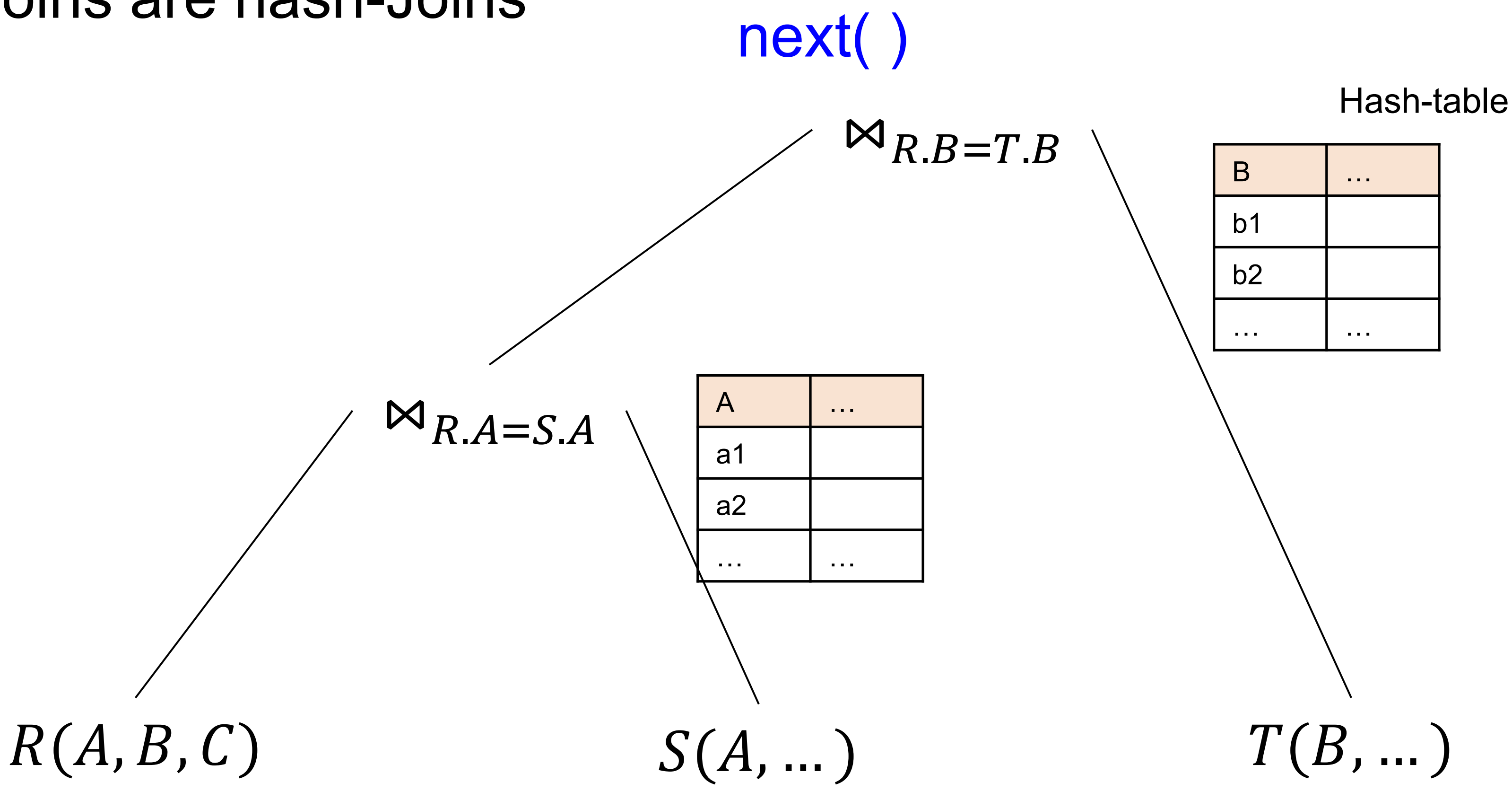


B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

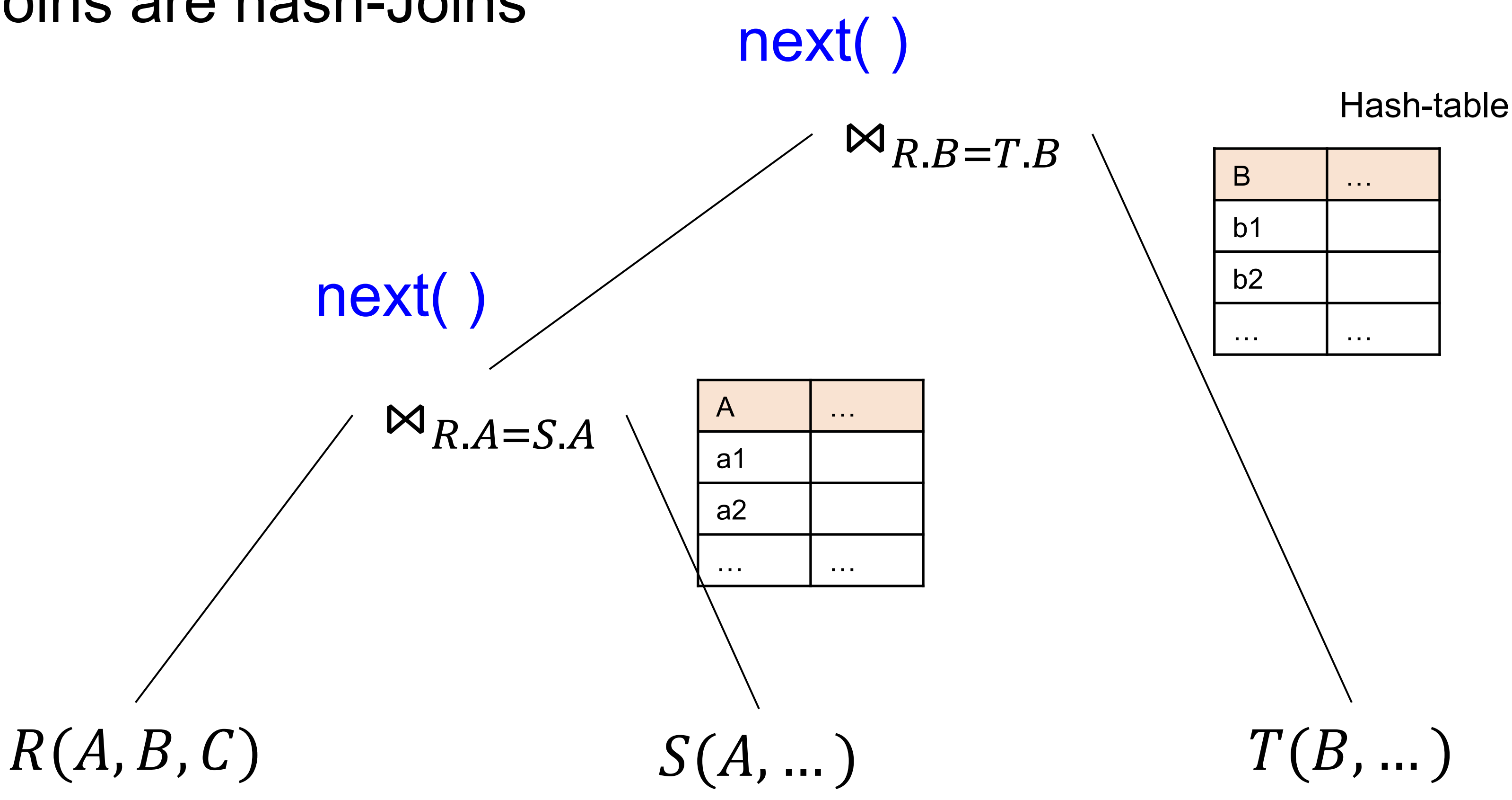
Operator Interface

Both joins are hash-Joins



Operator Interface

Both joins are hash-Joins

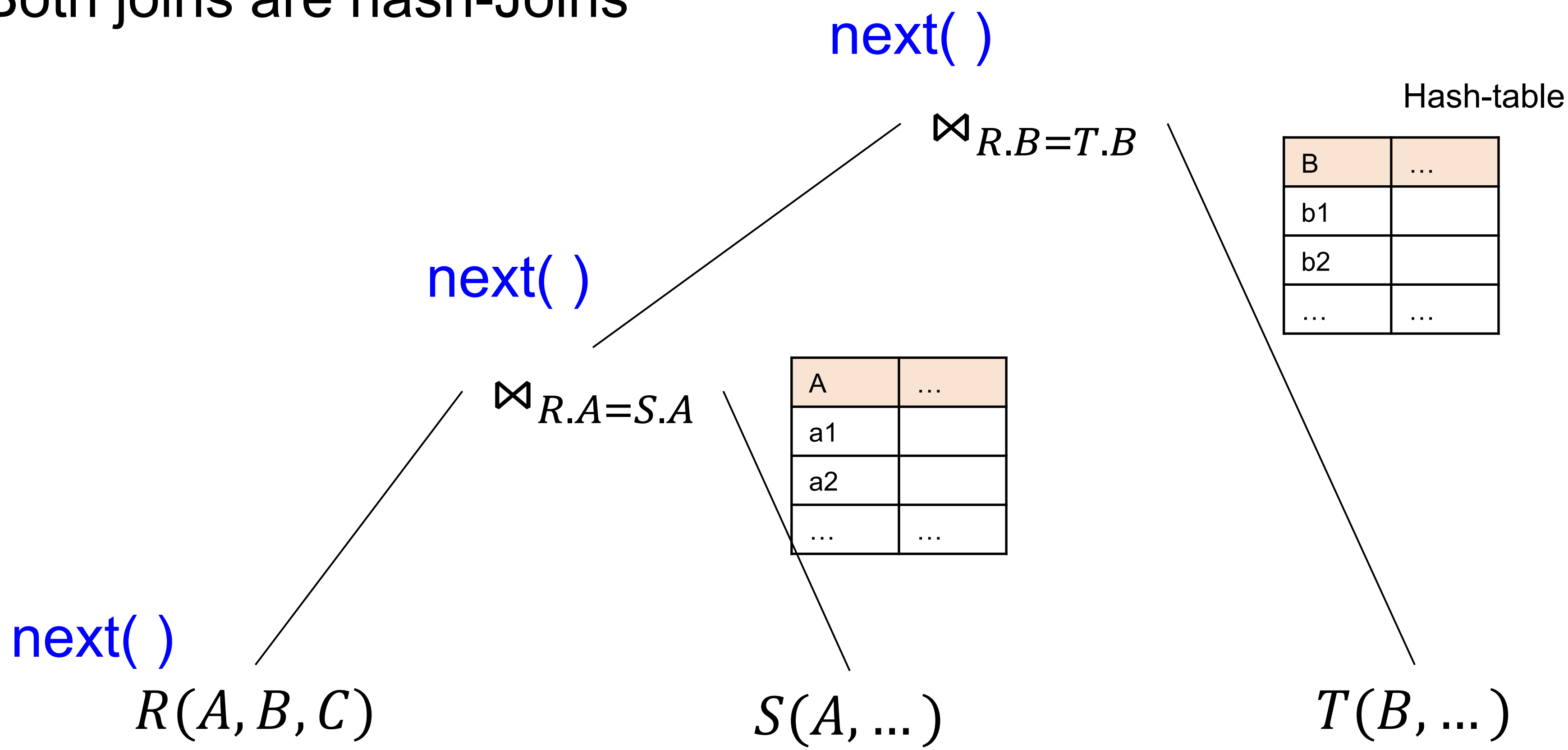


B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

Operator Interface

Both joins are hash-Joins

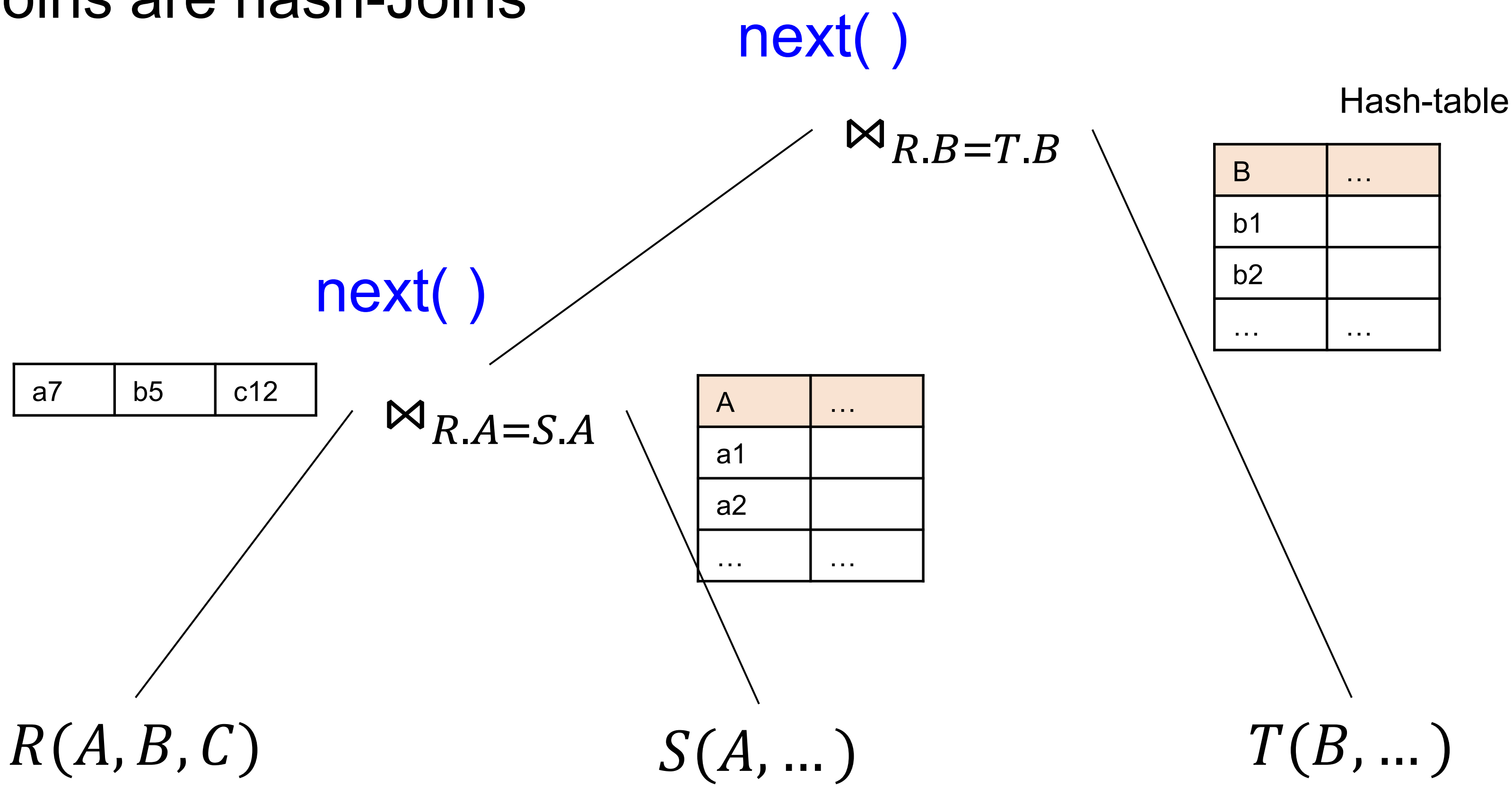


B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

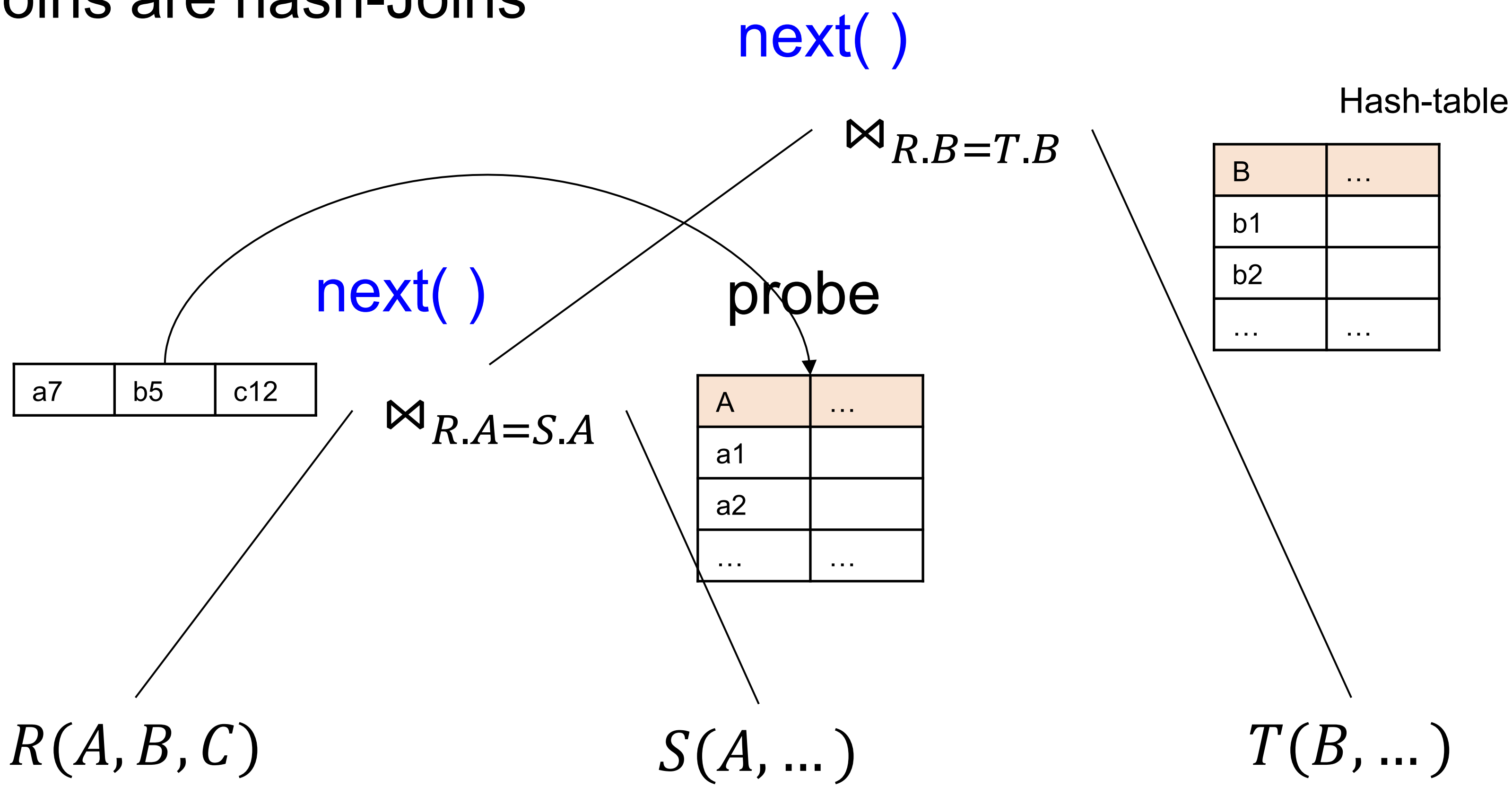
Operator Interface

Both joins are hash-Joins



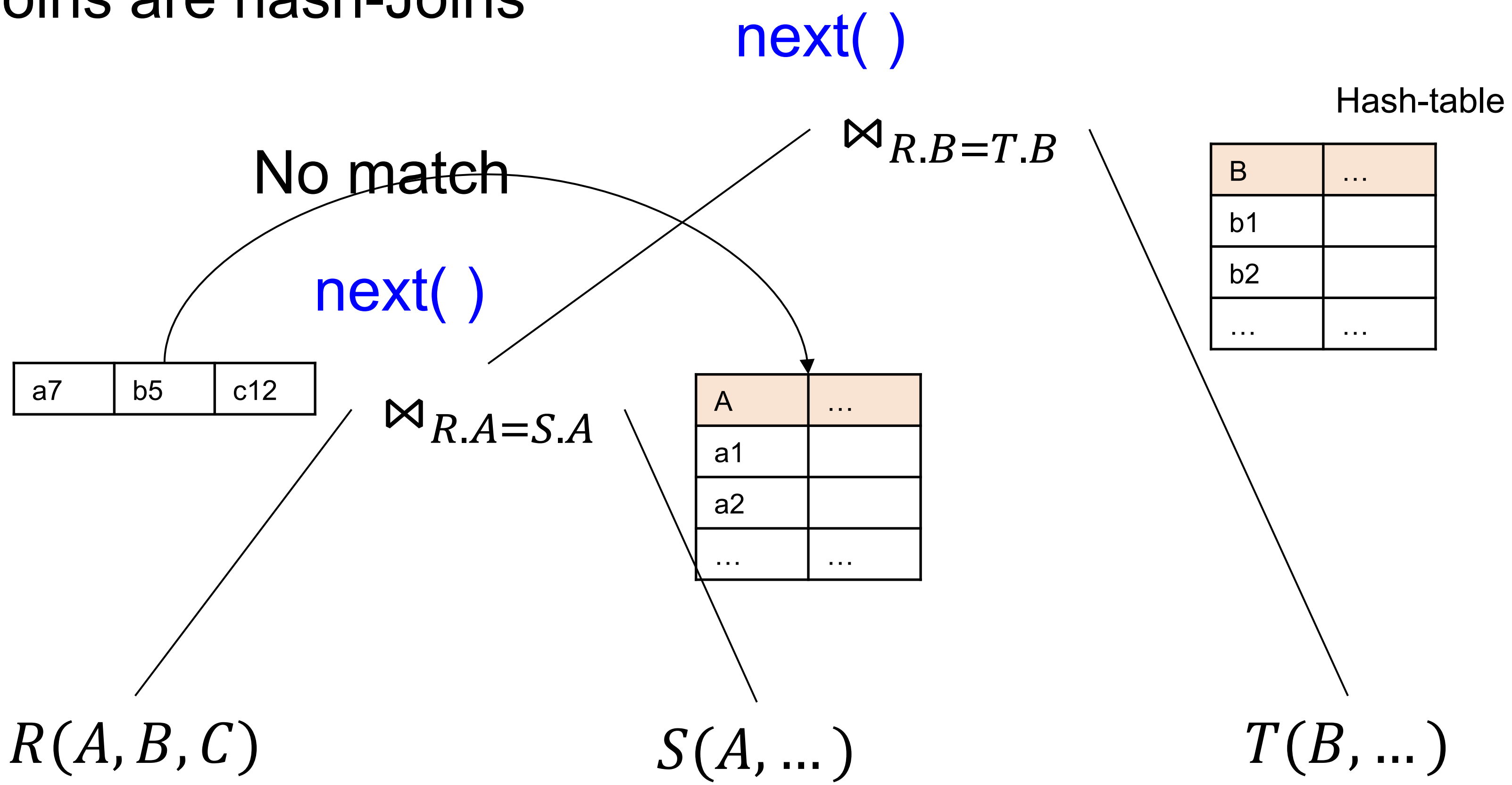
Operator Interface

Both joins are hash-Joins



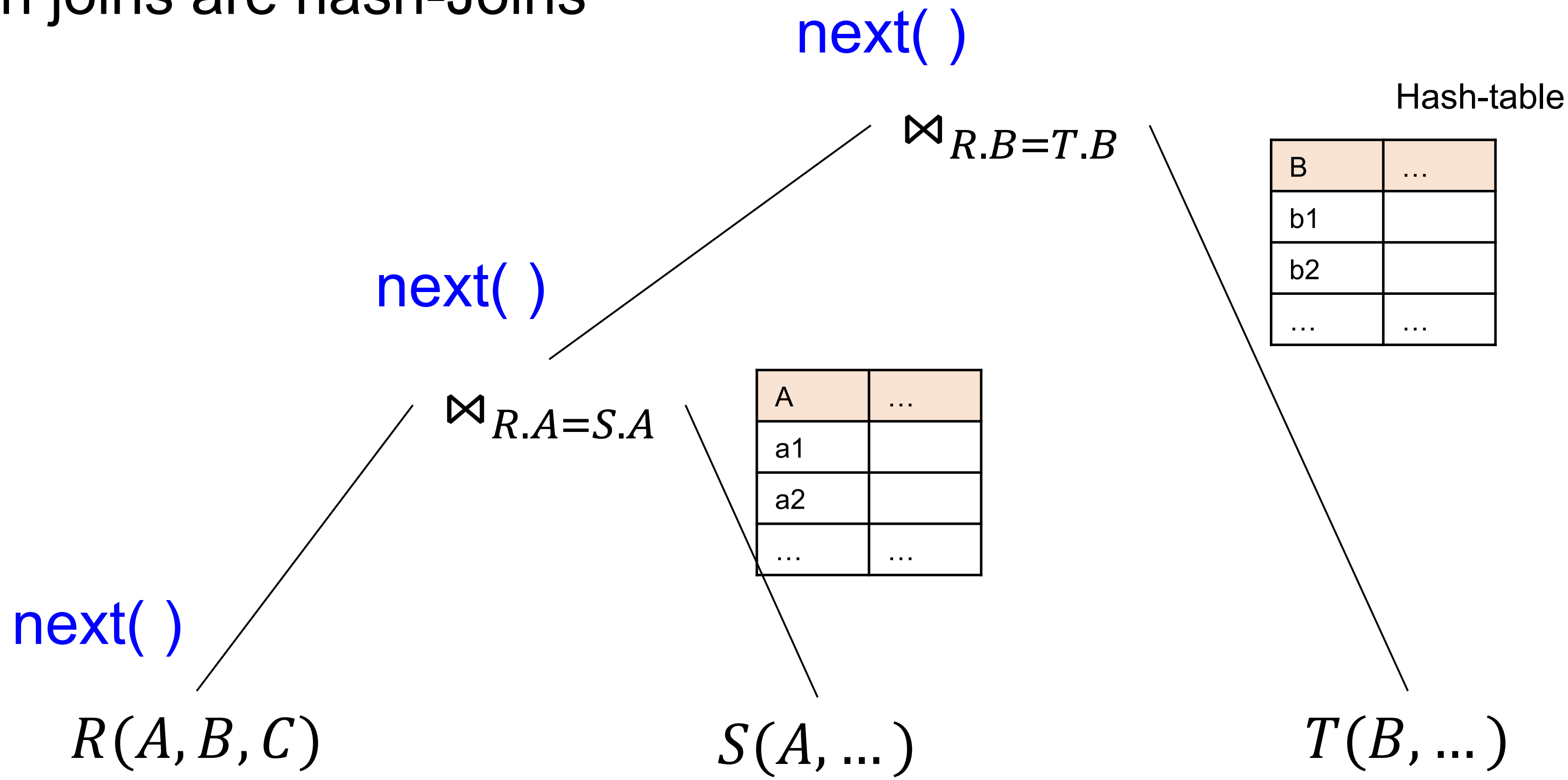
Operator Interface

Both joins are hash-Joins



Operator Interface

Both joins are hash-Joins

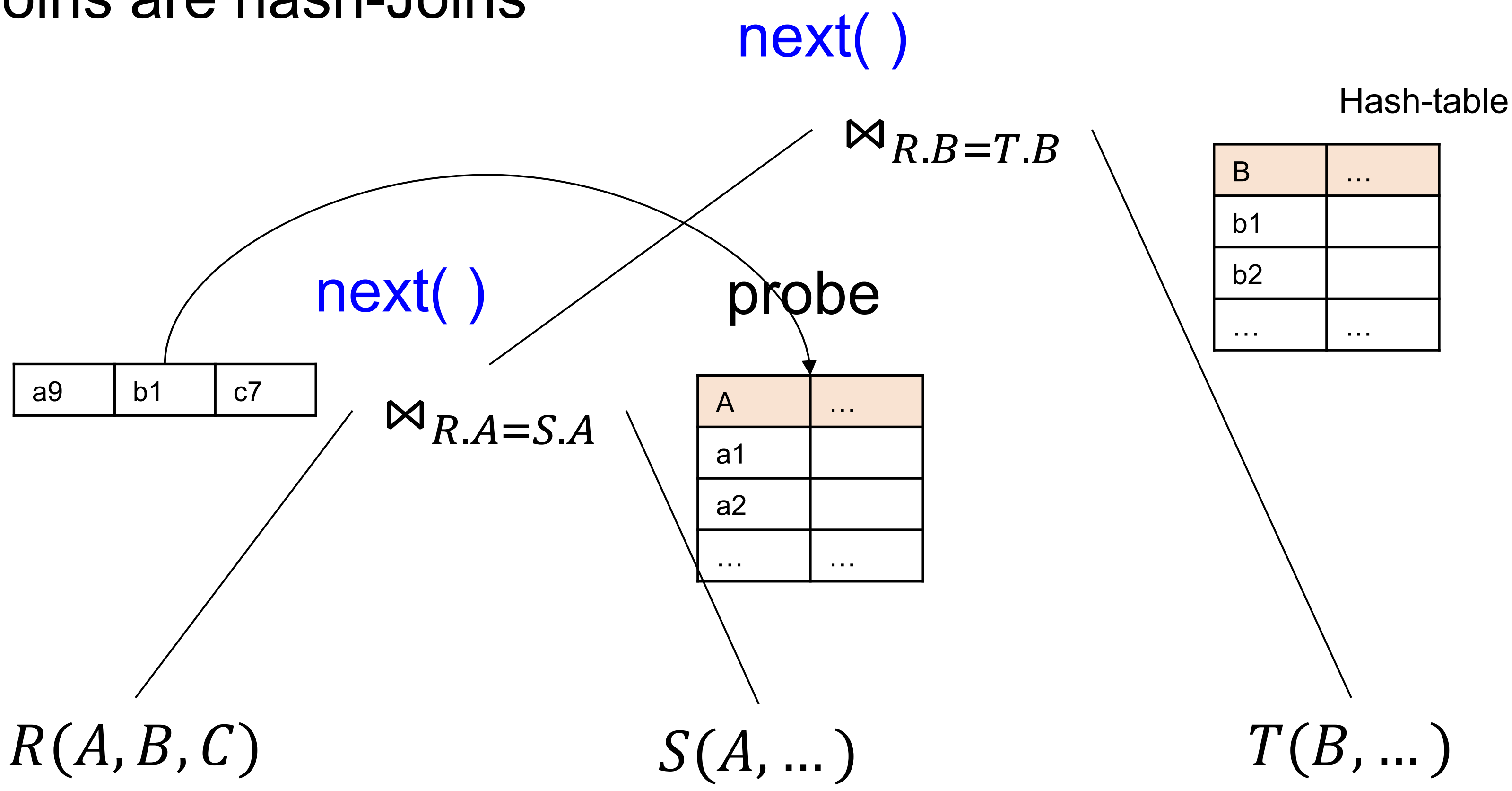


B	…
b1	
b2	
…	…

A	…
a1	
a2	
…	…

Operator Interface

Both joins are hash-Joins

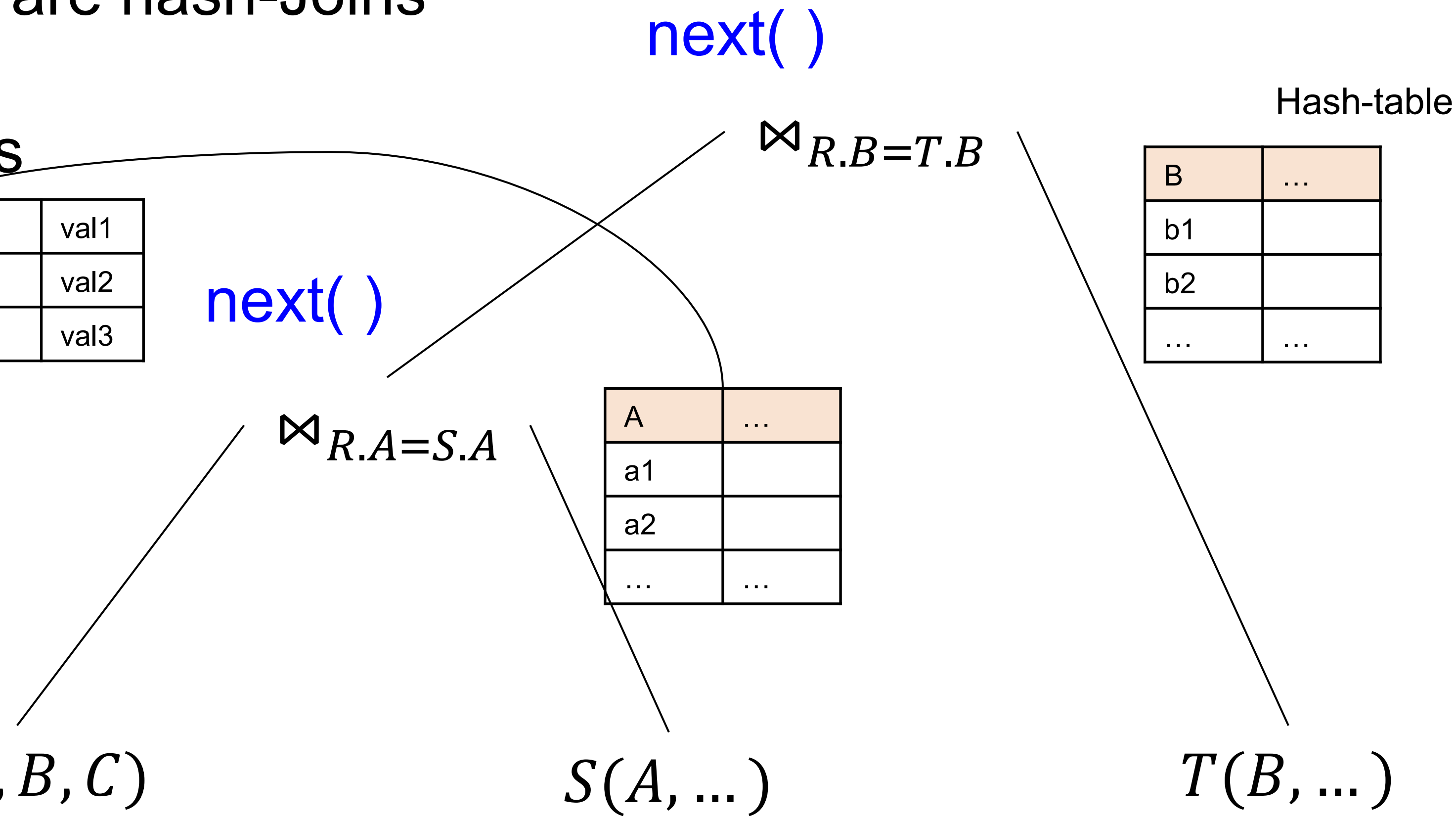


Operator Interface

Both joins are hash-Joins

3 matches

a9	b1	c7	val1
a9	b1	c7	val2
a9	b1	c7	val3



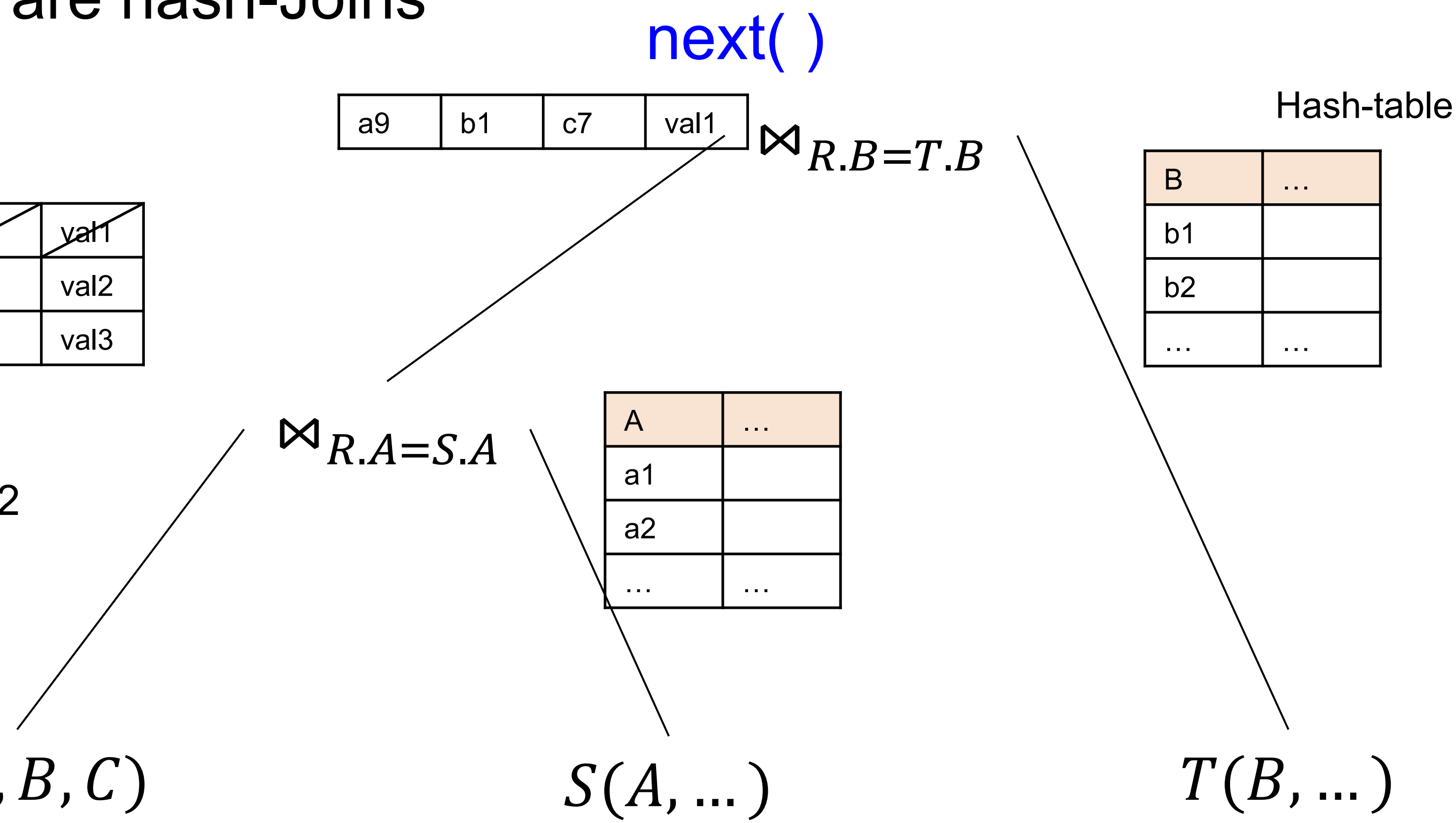
A	…
a1	
a2	
…	…

Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins



a9	b1	c7	val1

~~a9~~	~~b1~~	~~c7~~	~~val1~~
a9	b1	c7	val2
a9	b1	c7	val3

Return 1,
save the other 2

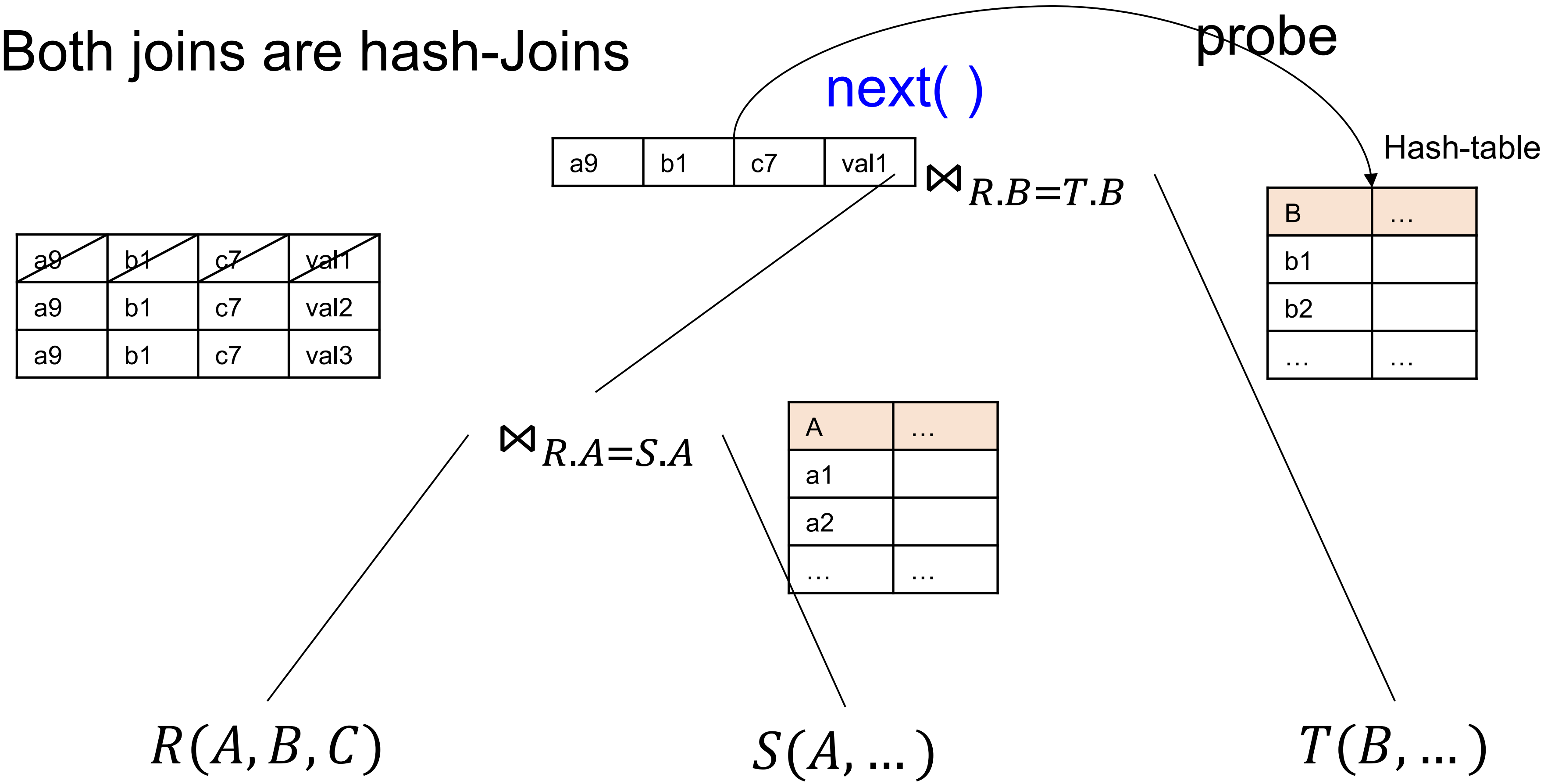
A	…
a1	
a2	
…	…

Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins

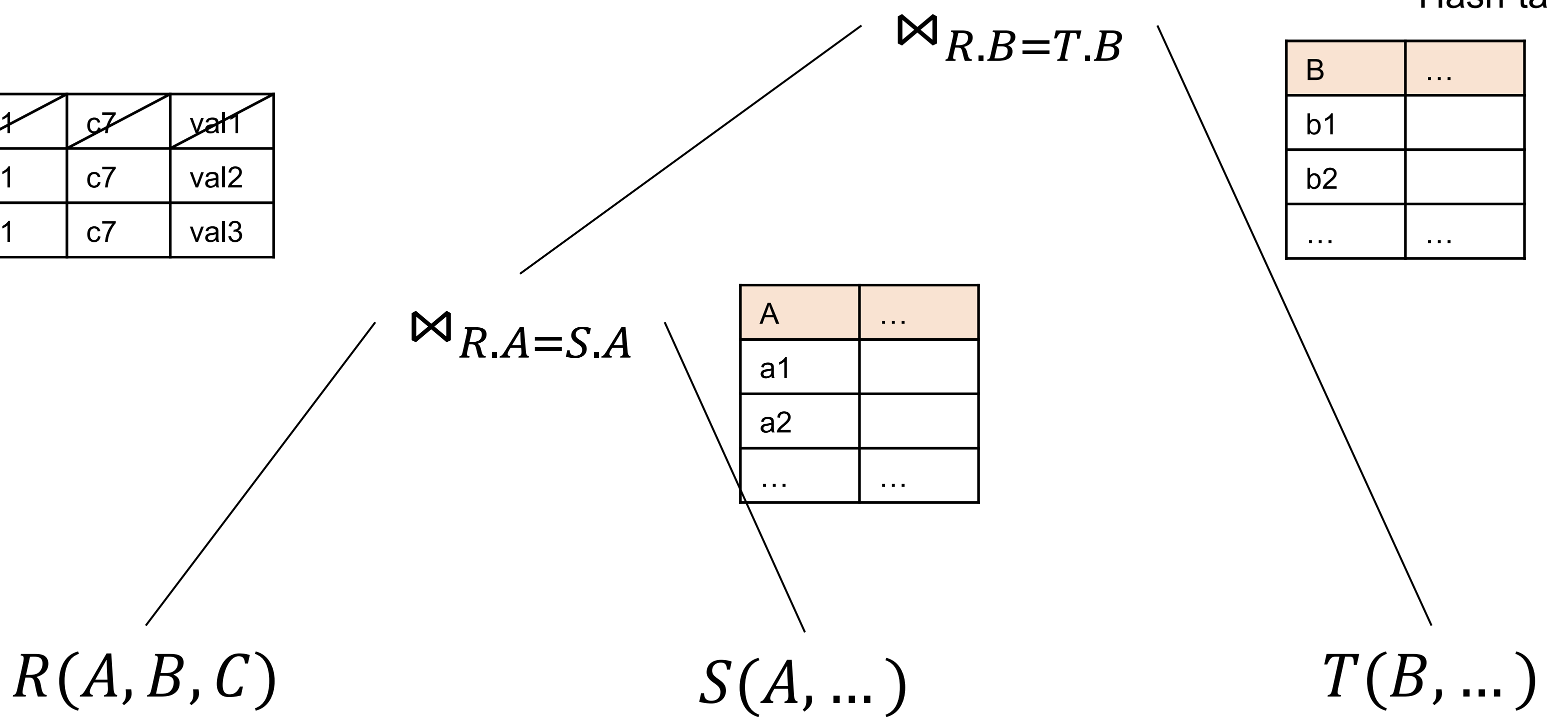


Operator Interface

return:

a9	b1	c7	val1	xyz

Both joins are hash-Joins



~~a9~~	~~b1~~	~~c7~~	~~val1~~
a9	b1	c7	val2
a9	b1	c7	val3

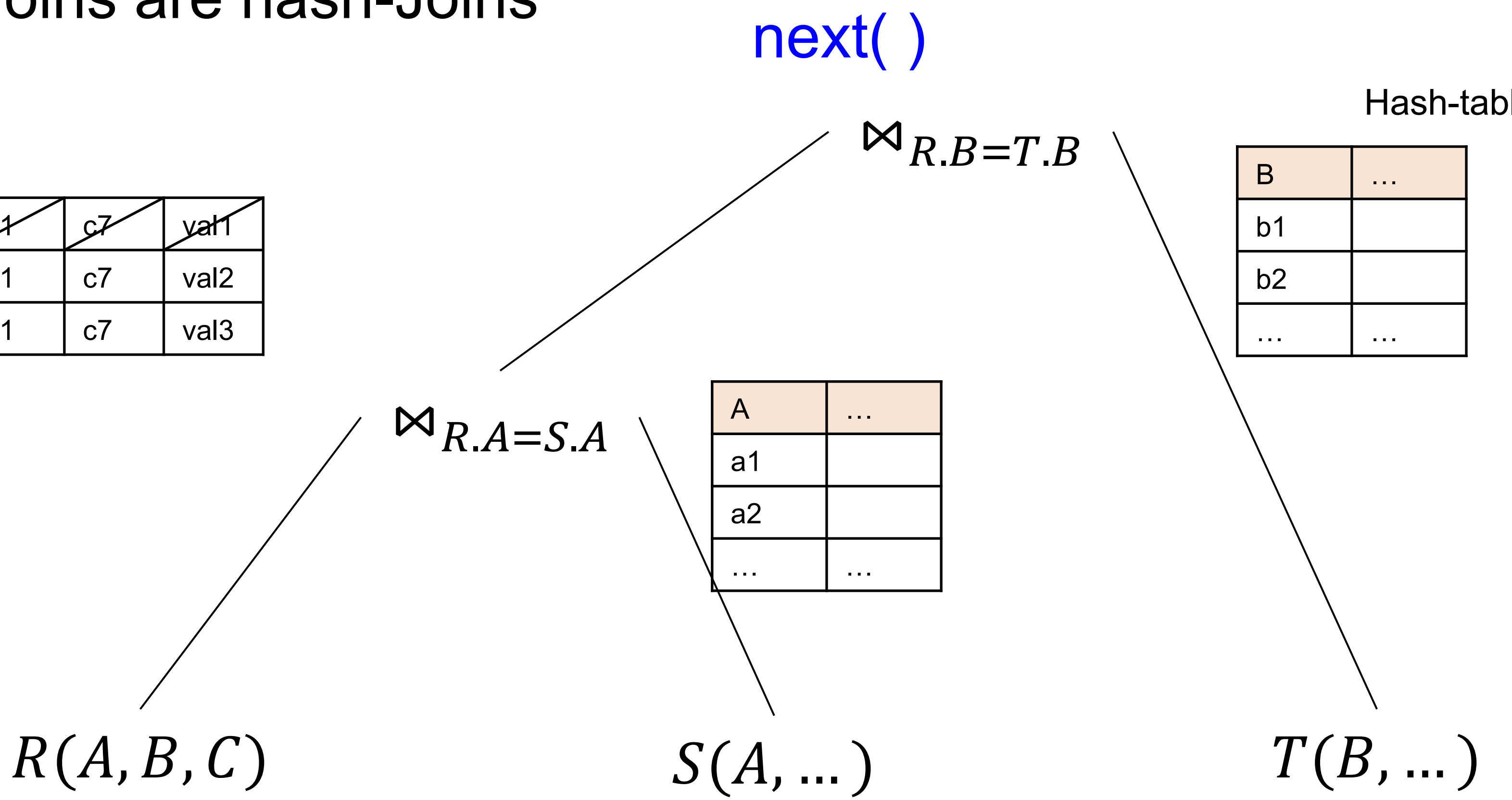
A	…
a1	
a2	
…	…

Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins



~~a9~~	~~b1~~	~~c7~~	~~val1~~
a9	b1	c7	val2
a9	b1	c7	val3

A	…
a1	
a2	
…	…

Hash-table

B	…
b1	
b2	
…	…

Operator Interface

Both joins are hash-Joins

~~a9~~	~~b1~~	~~c7~~	~~val1~~
a9	b1	c7	val2
a9	b1	c7	val3

next( )

$\bowtie_{R.B=T.B}$

Hash-table

B	…
b1	
b2	
…	…

next( )

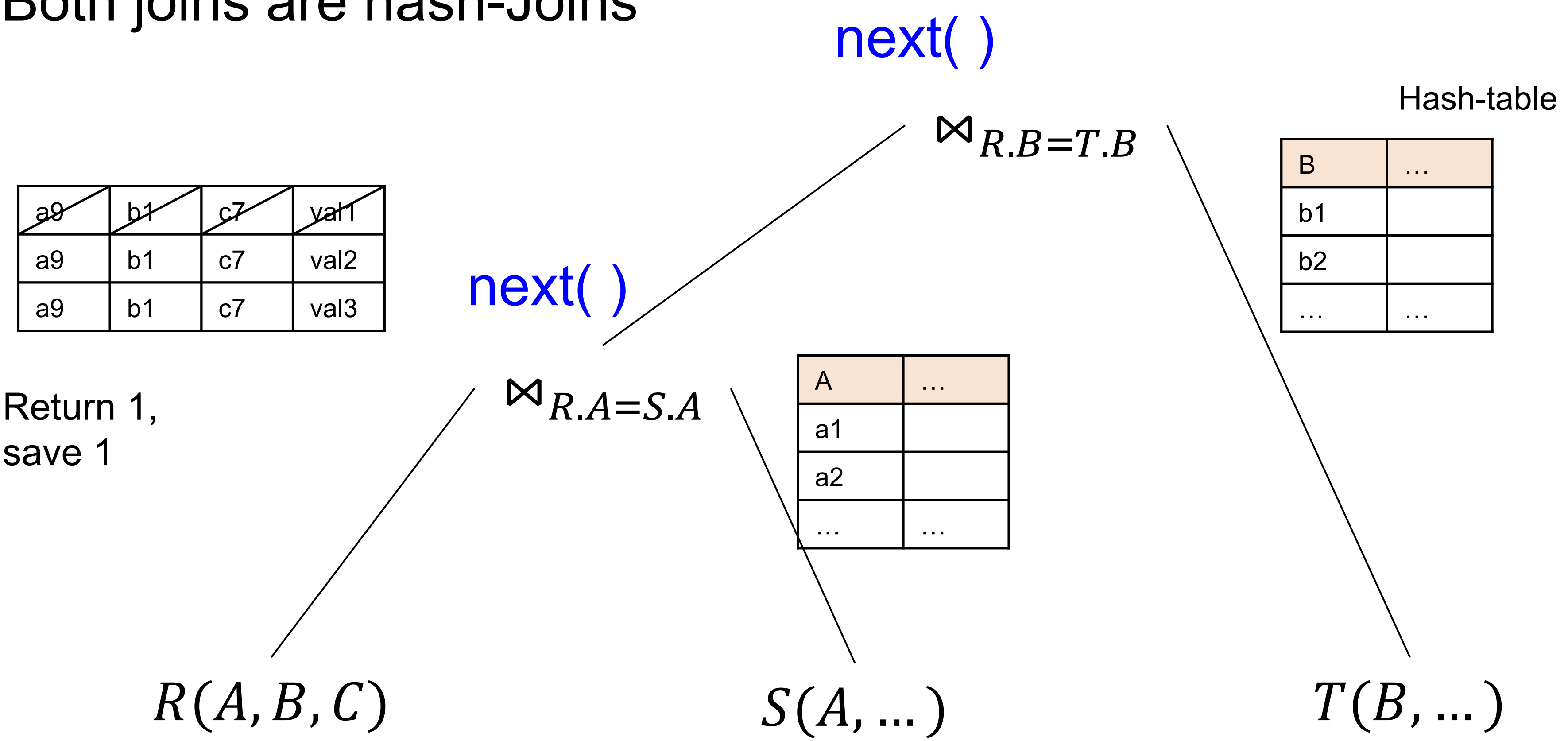
$\bowtie_{R.A=S.A}$

A	…
a1	
a2	
…	…

$R(A, B, C)$ $S(A, \ldots)$ $T(B, \ldots)$

Operator Interface

Both joins are hash-Joins



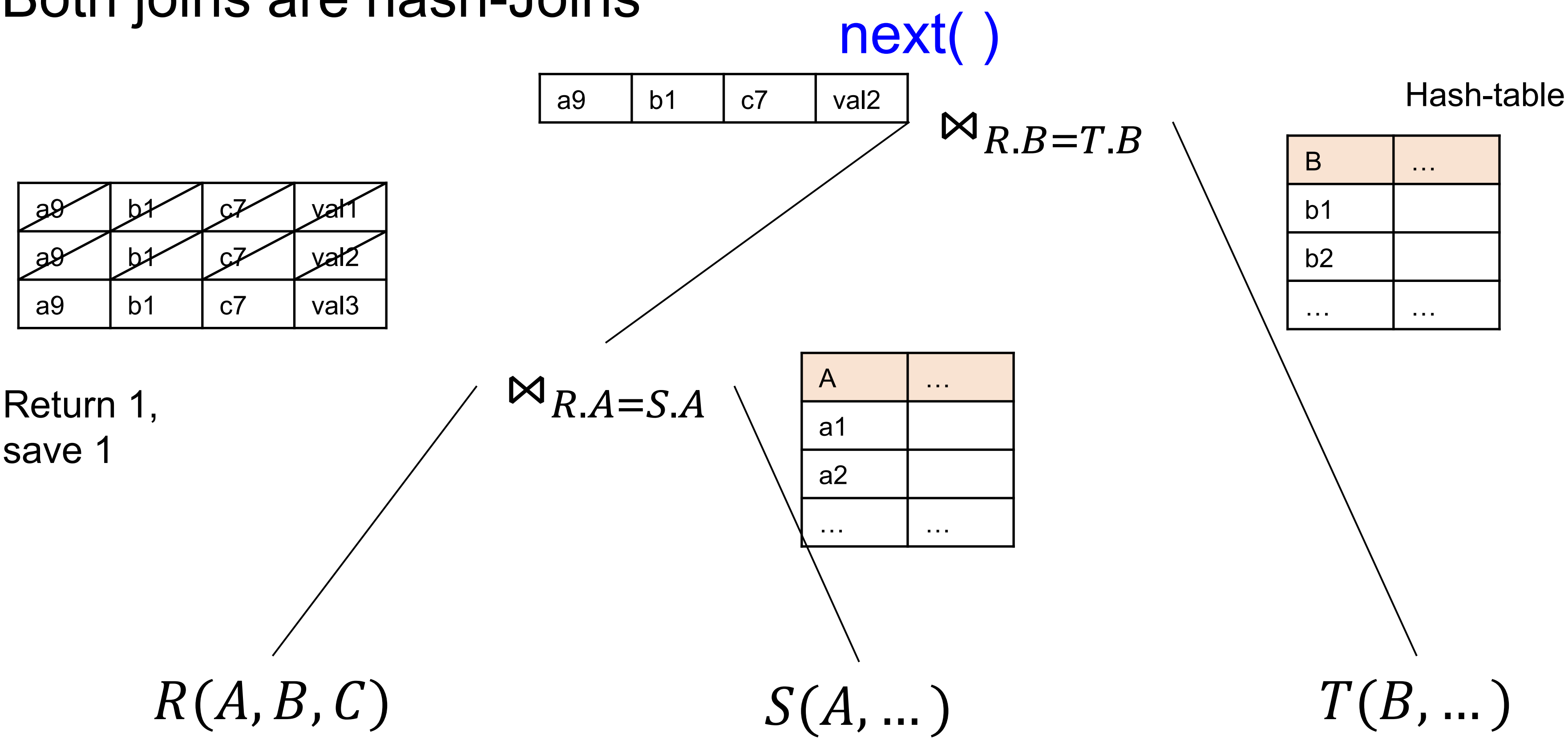
~~a9~~	~~b1~~	~~c7~~	~~val1~~
a9	b1	c7	val2
a9	b1	c7	val3

A	…
a1	
a2	
…	…

B	…
b1	
b2	
…	…

Operator Interface

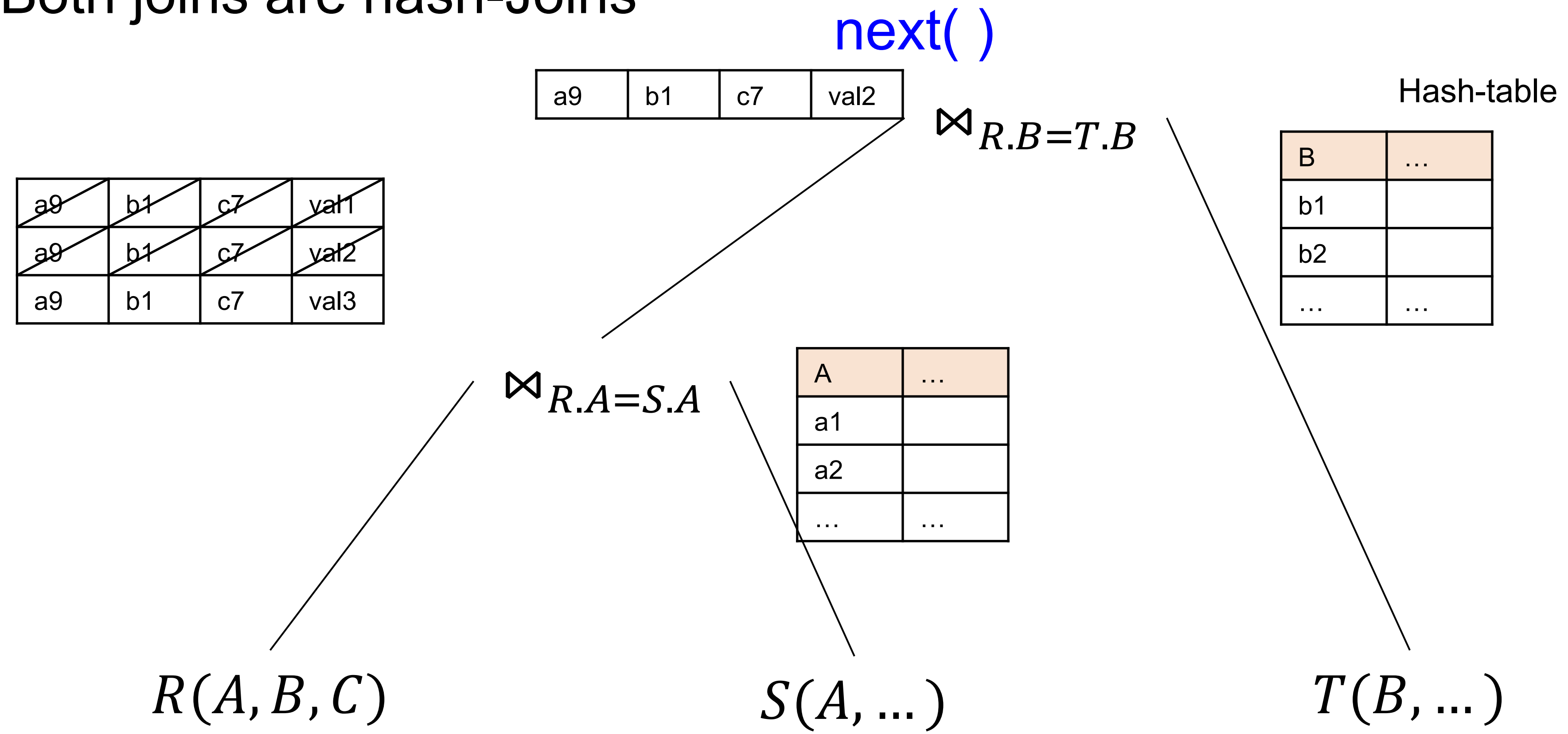
Both joins are hash-Joins



Operator Interface

And so on, until all tuples are computed

Both joins are hash-Joins



Discussion

- Most systems adopt the Volcano-model, a.k.a. the iterator interface
- Vectorized processing = iterator interface that processes a *block* of tuples (vector?) instead of one tuple
- Compiled model = compile to machine code *and* use the push model

Cardinality Estimation

Cardinality Estimation

Problem: given statistics on base tables and a query, estimate size of the answer

Very difficult, because:

- Need to do it very fast
- Need to use very little memory

Statistics on Base Data

- Number of tuples (cardinality) T(R)
- Number of physical pages B(R)
- Indexes, number of keys in the index V(R,a)
- Histogram on single attribute (1d)
- Histogram on two attributes (2d)

Computed periodically, often using sampling

Assumptions

- Uniformity
- Independence
- Containment of values
- Preservation of values

Size Estimation

Selection: size decreases by ***selectivity factor*** θ

$$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$$

Size Estimation

Selection: size decreases by ***selectivity factor*** θ

$$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$$

$$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$$

Selectivity Factors

Uniformity assumption

Equality:

$\boxed{\sigma_{A=c}(R)}$

Selectivity Factors

Uniformity assumption

$\sigma_{A=c}(R)$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

$\boxed{\sigma_{A=c}(R)}$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$\boxed{\sigma_{c1<A<c2}(R)}$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(max(R,A) - min(R,A))$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$\boxed{\sigma_{A=c}(R)}$

$\boxed{\sigma_{c1<A<c2}(R)}$

Range:

- $\theta_{c1<A<c2} = (c2 – c1)/(max(R,A) - min(R,A))$

Conjunction

$\boxed{\sigma_{A=c \text{ and } B=d}(R)}$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

Equality:

$\sigma_{A=c}(R)$

- $\theta_{A=c} = 1/V(R,A)$

$\sigma_{c1<A<c2}(R)$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(max(R,A) - min(R,A))$

Conjunction

$\sigma_{A=c \text{ and } B=d}(R)$

Independence assumption

- $\theta_{\text{pred1 and pred2}} = \theta_{pred1} * \theta_{pred2} = 1/V(R,A) * 1/V(R,B)$

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Join

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Join

- $\theta_{R.A=S.B} = 1/\ (\ MAX(\ V(R,A), V(S,B))$

Why? Will explain next...

Warmup: Key / Foreign-key Join

Supplier(<u>sno</u>,sname,scity,sstate)
Supply(<u>sno,pno</u>,qty,price)

We know T(Supplier) and T(Supply)

Q: How large is Supplier $\bowtie$ Supply?

Warmup: Key / Foreign-key Join

Supplier(<u>sno</u>,sname,scity,sstate)
Supply(<u>sno,pno</u>,qty,price)

We know T(Supplier) and T(Supply)

Q: How large is Supplier $\bowtie$ Supply?

A: T(Supplier $\bowtie$ Supply) = T(Supply)

Make sure you understand why…

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Containment of values: if $V(R,A) \leq V(S,B)$, then the set of A values of R is included in the set of B values of S

- Note: this indeed holds when A is a foreign key in R, and B is a key in S

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume $V(R,A) \leq V(S,B)$

- Tuple t in R joins with $T(S)/V(S,B)$ tuples in S

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume $V(R,A) \leq V(S,B)$

- Tuple t in R joins with $T(S)/V(S,B)$ tuples in S
- Hence $T(R \bowtie_{A=B} S) = T(R)\ T(S)\ /\ V(S,B)$

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume $V(R,A) \leq V(S,B)$

- Tuple t in R joins with $T(S)/V(S,B)$ tuples in S
- Hence $T(R \bowtie_{A=B} S) = T(R)\ T(S) / V(S,B)$

In general:

- $T(R \bowtie_{A=B} S) = T(R)\ T(S) / \max(V(R,A),V(S,B))$
- $\theta_{R.A=S.B} = 1/ (\max(V(R,A), V(S,B))$

Key / Foreign-key Join

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
```

We know T(Supplier) and T(Supply)

Q: How large is Supplier ⋈ Supply?

A: T(Supplier ⋈ Supply) =
= T(Supplier)*T(Supply) /max(V(Supplier,sno), V(Supply,sno))

Key / Foreign-key Join

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
```

We know T(Supplier) and T(Supply)

Q: How large is Supplier ⋈ Supply?

A: T(Supplier ⋈ Supply) =

= T(Supplier)*T(Supply) /max(V(Supplier,sno), V(Supply,sno))

= T(Supplier)*T(Supply) / V(Supplier,sno)

Key / Foreign-key Join

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
```

We know T(Supplier) and T(Supply)

Q: How large is Supplier ⋈ Supply?

A: T(Supplier ⋈ Supply) =

= T(Supplier)*T(Supply) /max(V(Supplier,sno), V(Supply,sno))

= T(Supplier)*T(Supply) / V(Supplier,sno)

= T(Supply)

Final Assumption

Preservation of values:

For any other attribute C:

- $V(R \bowtie_{A=B} S, C) = V(R, C)$ or
- $V(R \bowtie_{A=B} S, C) = V(S, C)$

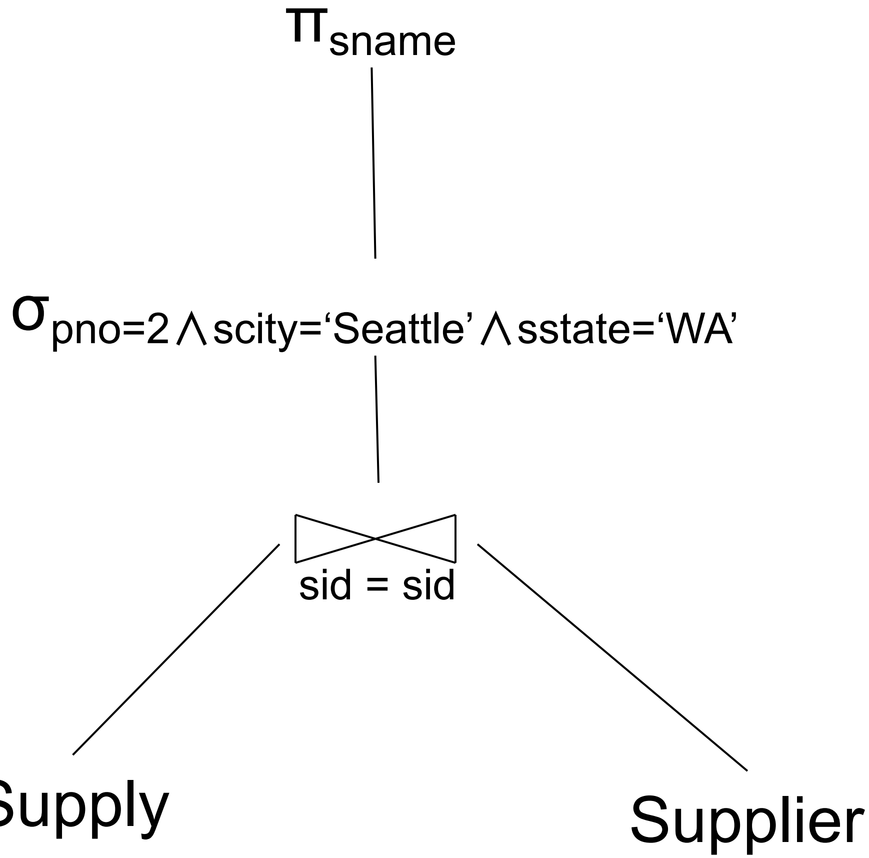
- This is needed higher up in the plan

Computing the Cost of a Plan

- Estimate cardinalities bottom-up
- Estimate cost by using estimated cardinalities
- Examples next...

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

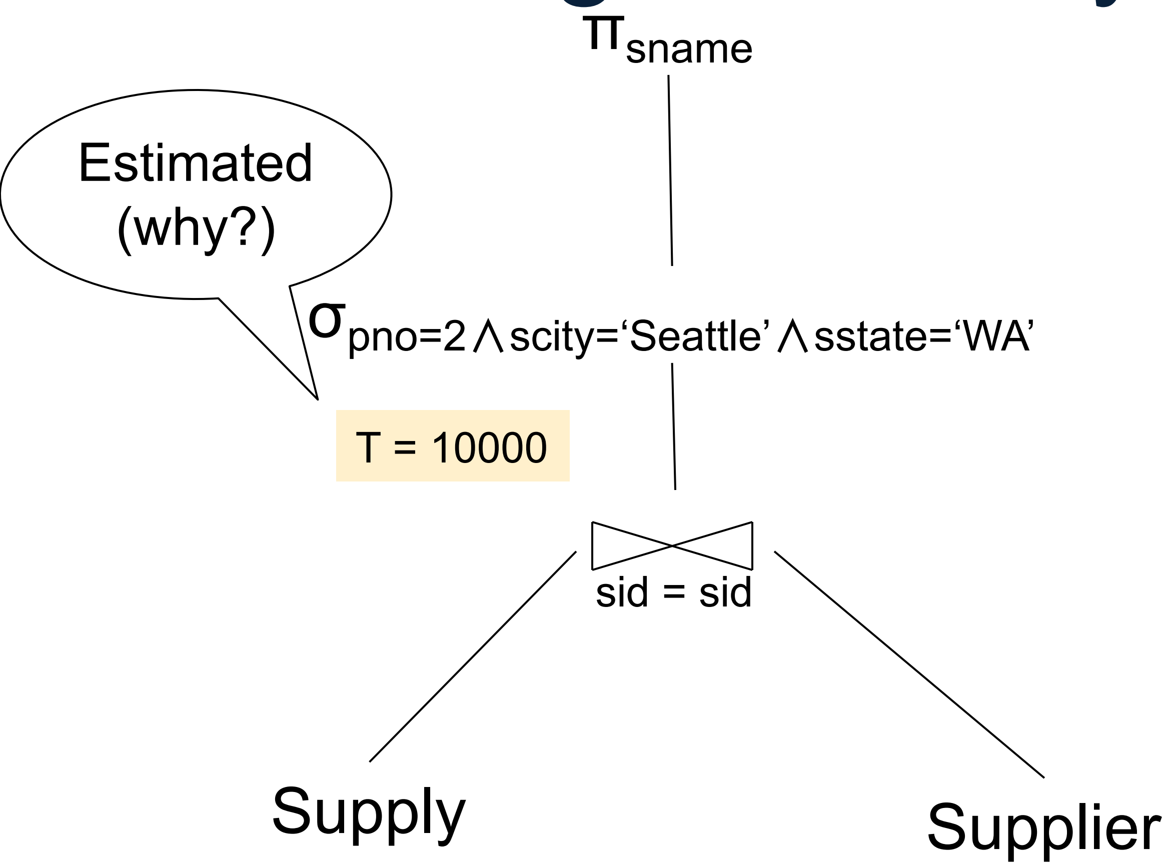
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

π_{sname}

Estimated (why?)

$\sigma_{pno=2 \wedge scity='Seattle' \wedge sstate='WA'}$

T = 10000

Because key / foreign-key

$\bowtie_{sid = sid}$

Supply

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Supplier

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

π_{sname}

Estimated (why?)

$\sigma_{pno=2 \wedge scity=\text{'Seattle'} \wedge sstate=\text{'WA'}}$

T = 10000

$\bowtie_{sid = sid}$

Because key / foreign-key

Also: θ = 1/max(V(Supply,sid)*V(Supplier,sid) = 1/V(Supplier,sid)

Supply

Supplier

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

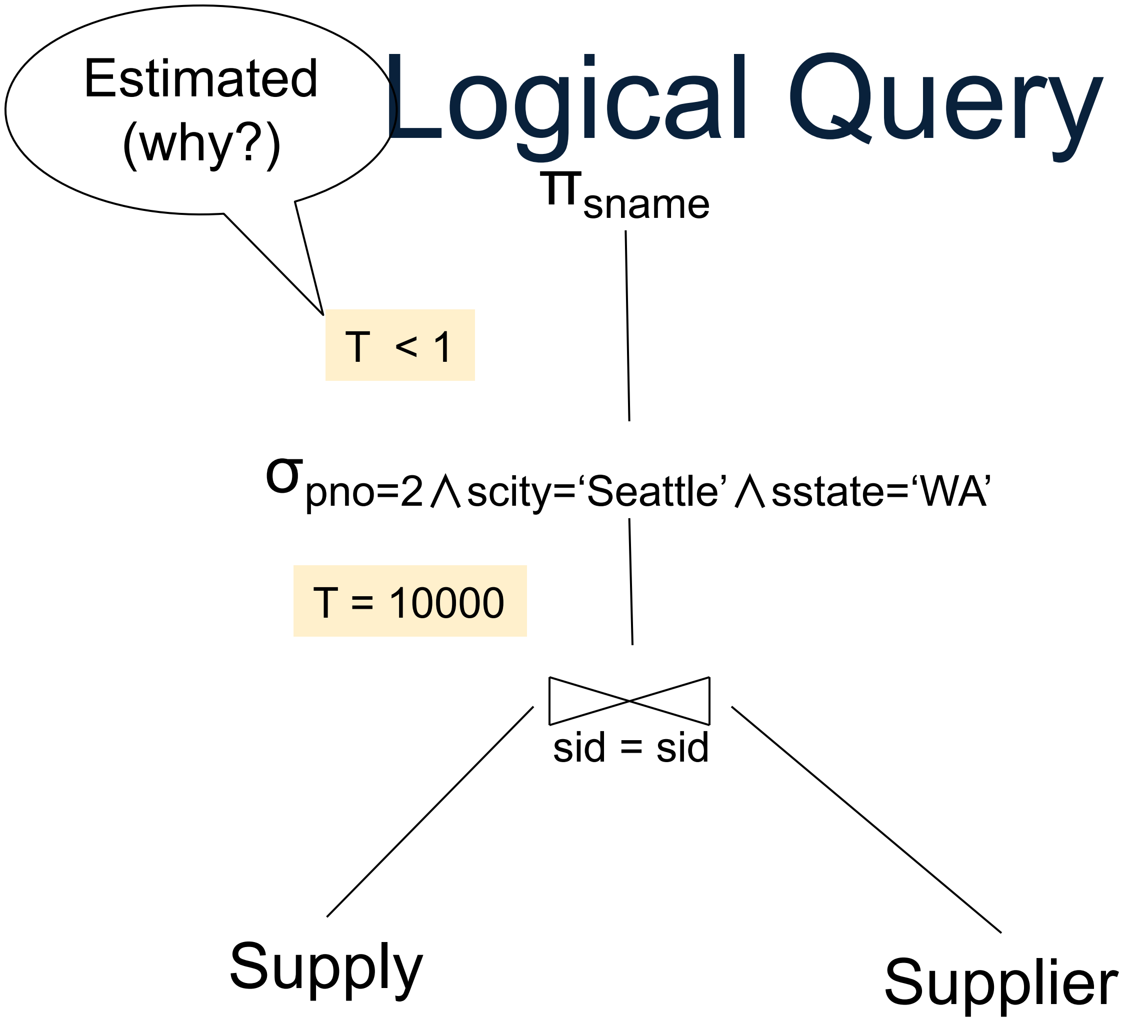
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

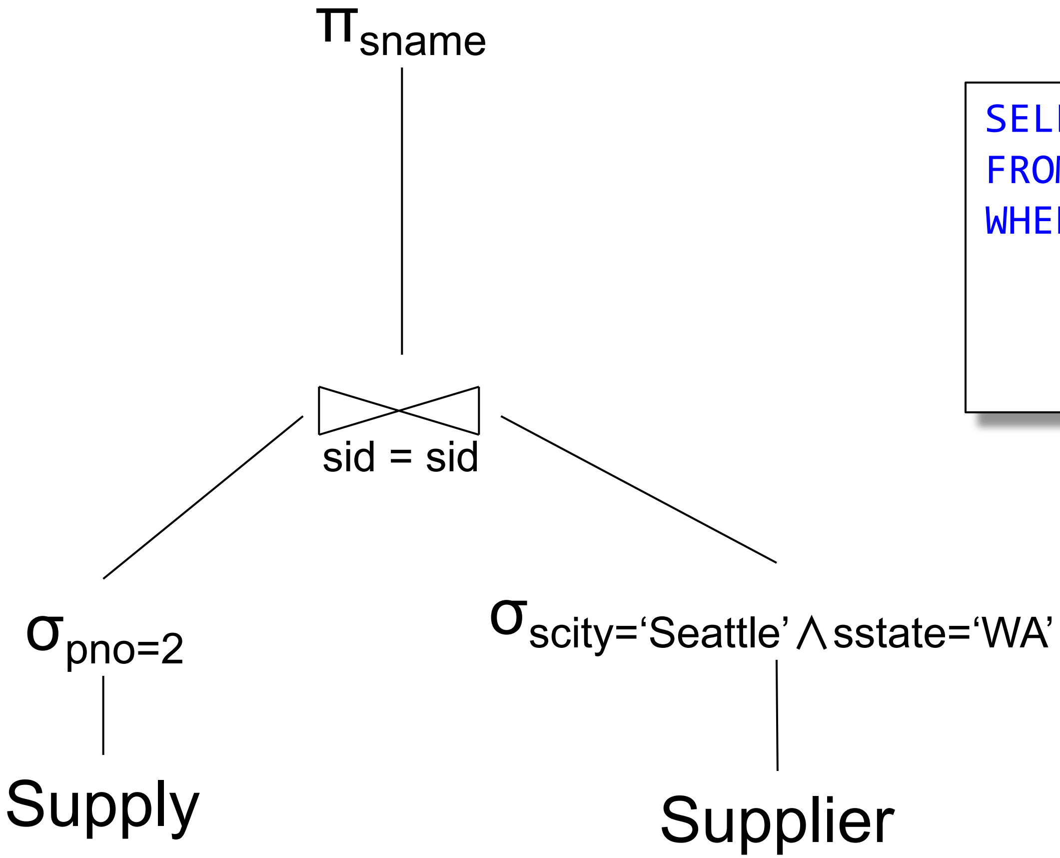
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

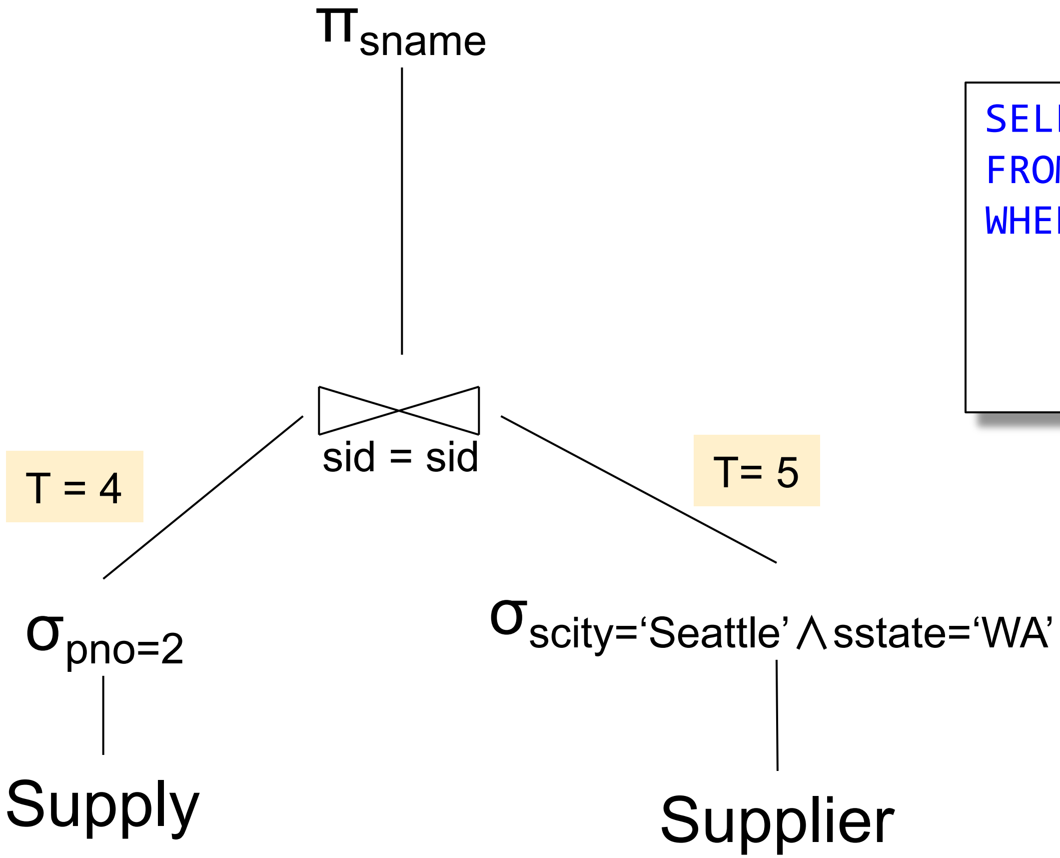
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

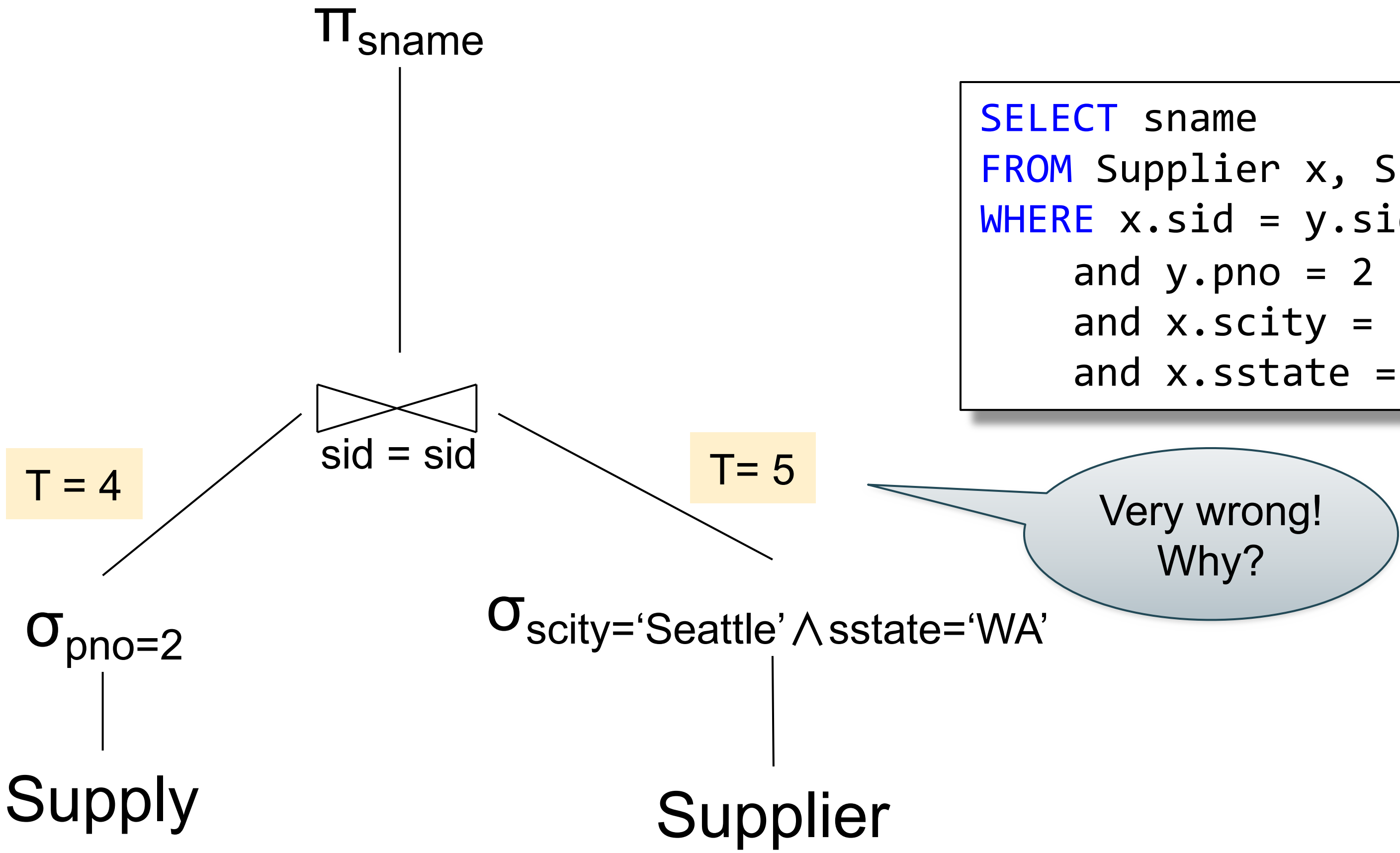
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



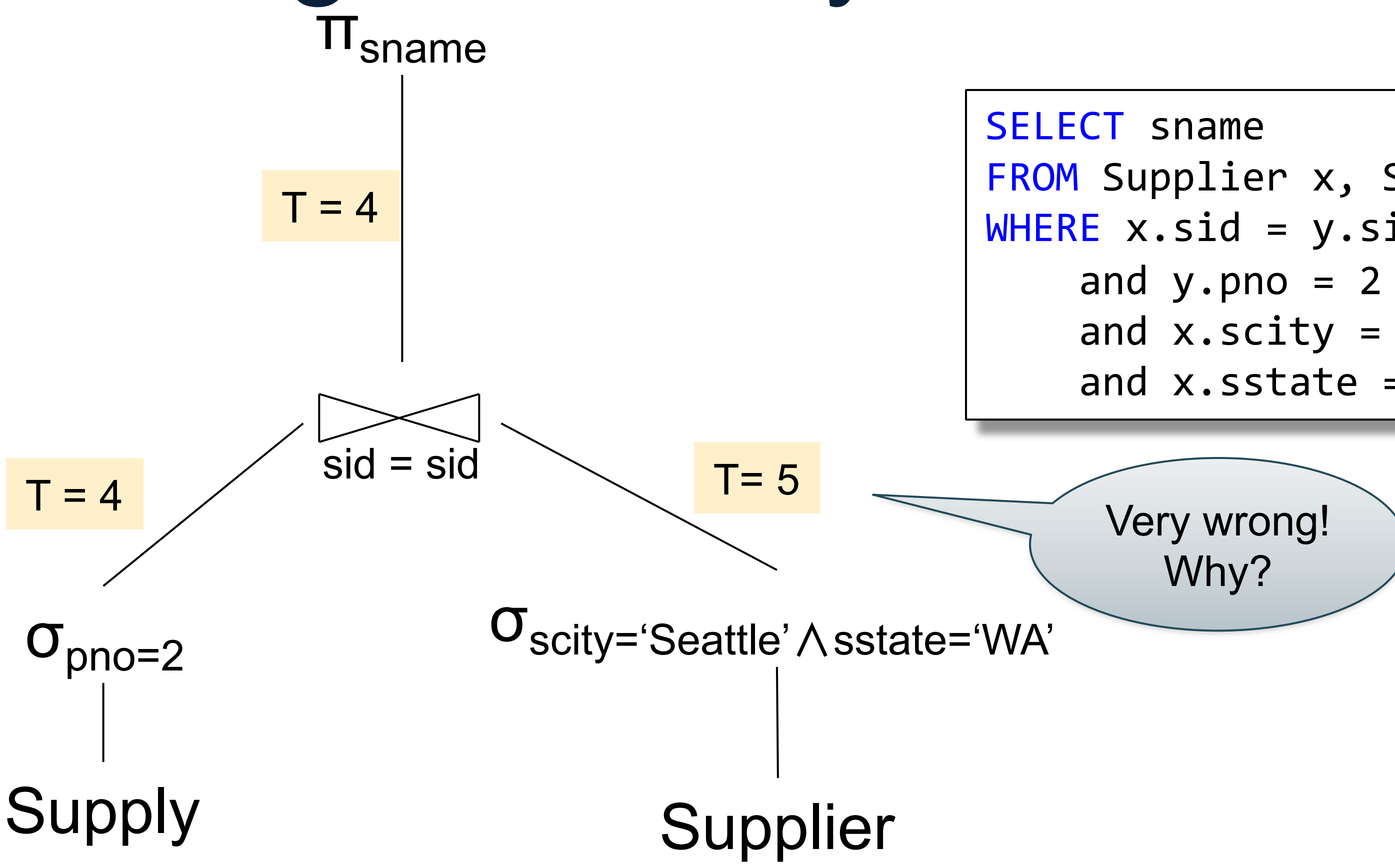
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

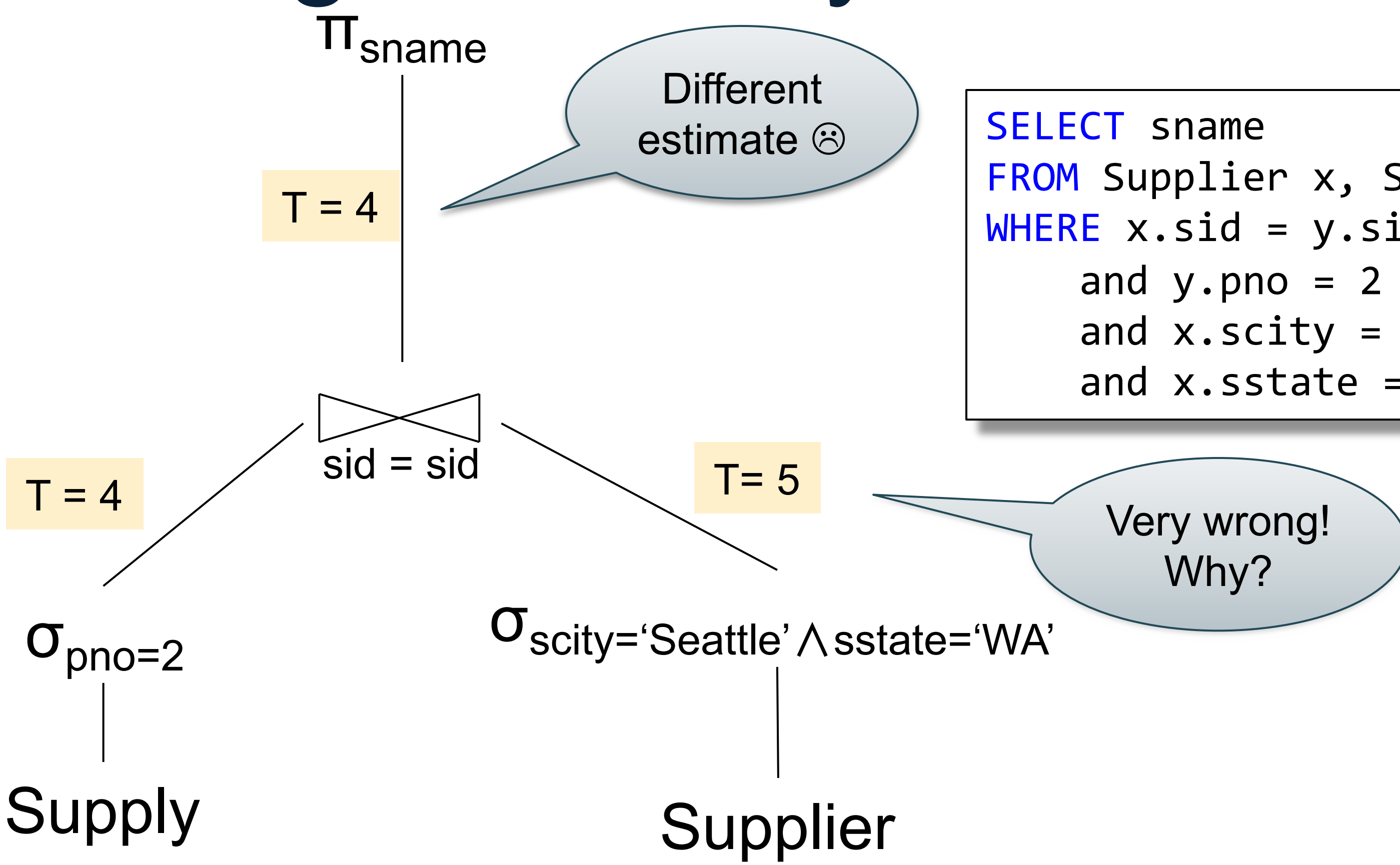
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Recap: External Memory Joins

- Disks are organized into *blocks*
- Typically: 1 block = 4k or 8k or 16k
- Main cost of an external memory algorithm is # blocks read or written
- Notation:
  - B(R) = number of blocks of relation R
  - M = number of blocks (=pages) that fit in main memory

Recap: External Memory Joins

$R \bowtie S$

Block nested loop join:

Recap: External Memory Joins

$R \bowtie S$

Block nested loop join:

Repeat:

    Read M blocks of R in memory

    Scan S (one block at a time):

        Join current records in S

        with current records in R

Until R is exhausted

Recap: External Memory Joins

$R \bowtie S$

Block nested loop join:

Repeat:

Read M blocks of R in memory

Scan S (one block at a time):

Join current records in S

with current records in R

Until R is exhausted

Cost: B(R) + B(R)*B(S)/M

Recap: External Memory Joins

$R \bowtie_{A=B} S$

Index join: assume an index on S.B

Recap: External Memory Joins

$R \bowtie_{A=B} S$

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time)

for each tuple in current block:

look up R.A using index on S.B

Until R is exhausted

Recap: External Memory Joins

$R \bowtie_{A=B} S$

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time)

for each tuple in current block:

look up R.A using index on S.B

Until R is exhausted

Cost: B(R) + T(R)*T(S)/V(S,B)

Recap: External Memory Joins

$R \bowtie_{A=B} S$

Index join: assume an index on S.B

Repeat:

Scan R (one block at a time)

for each tuple in current block:

look up R.A using index on S.B

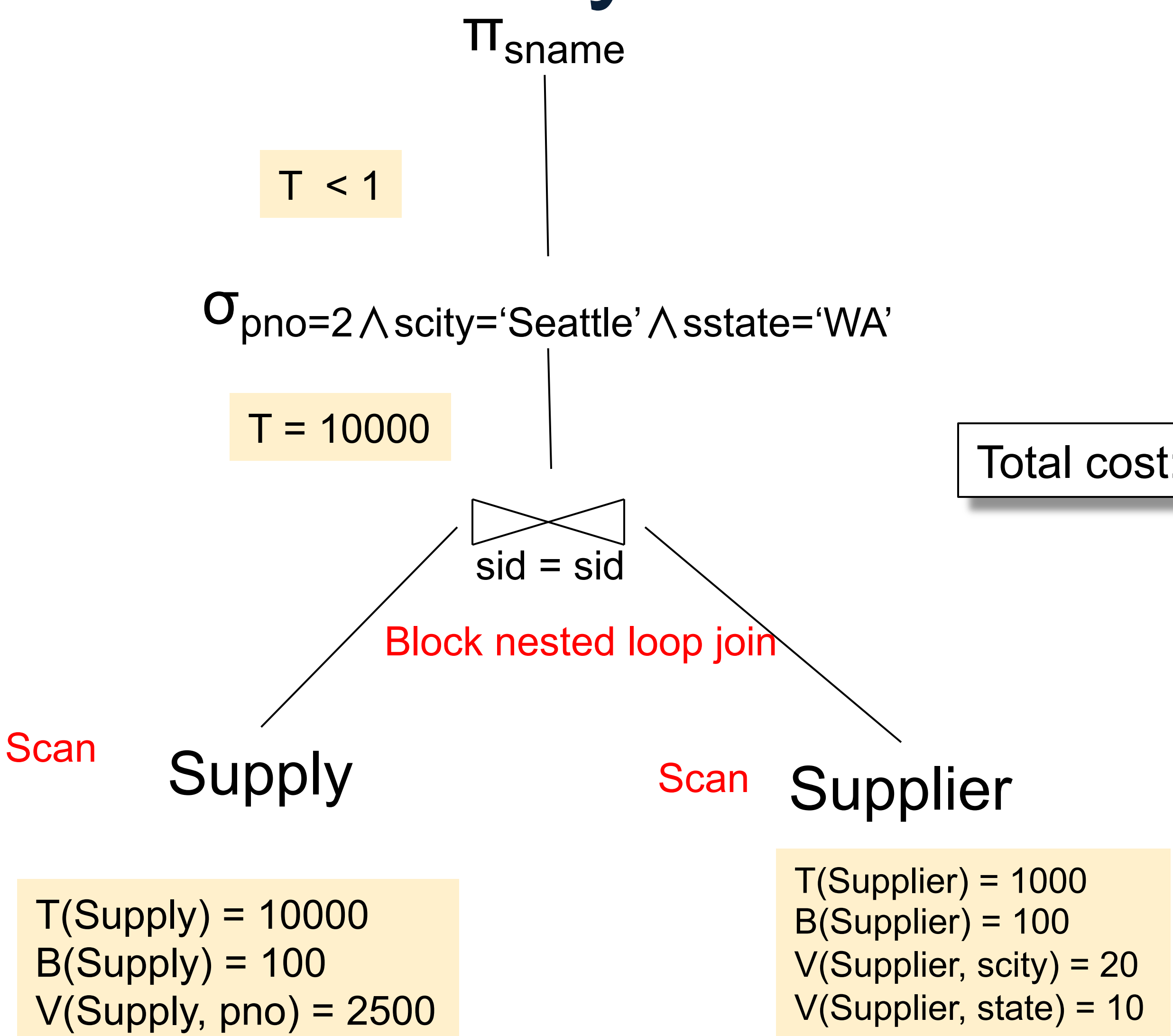
Until R is exhausted

If S.B is
a clustered
index

Cost: B(R) + T(R)*B(S)/V(S,B)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1

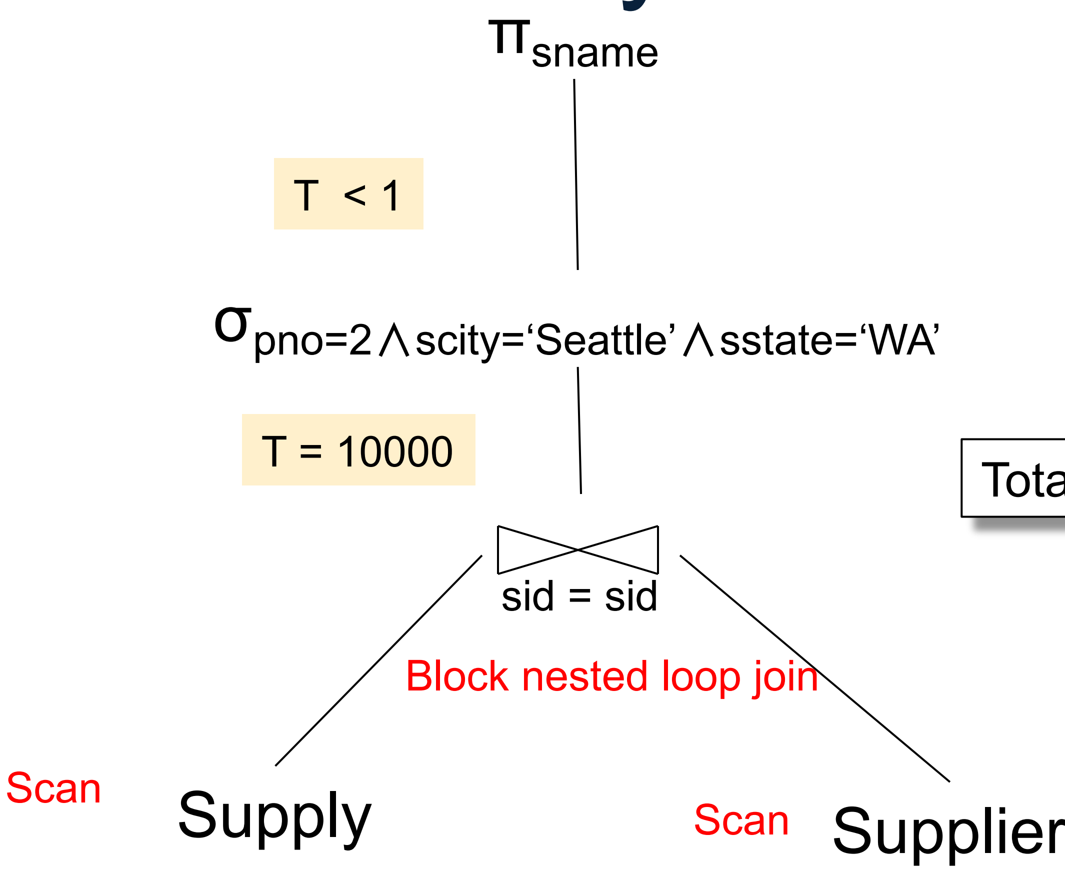


Total cost:

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1



Total cost: 100+100*100/10 = 1100

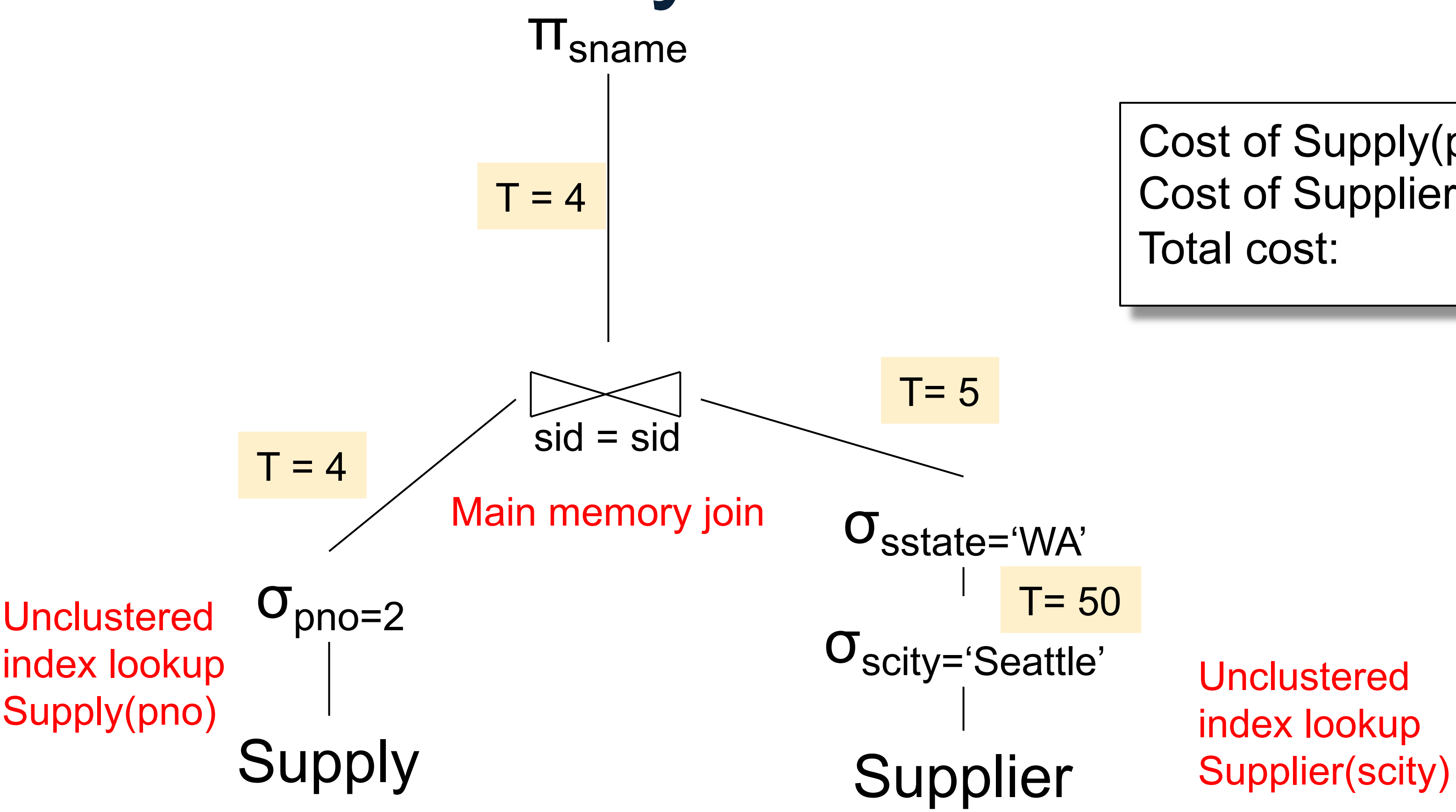
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:

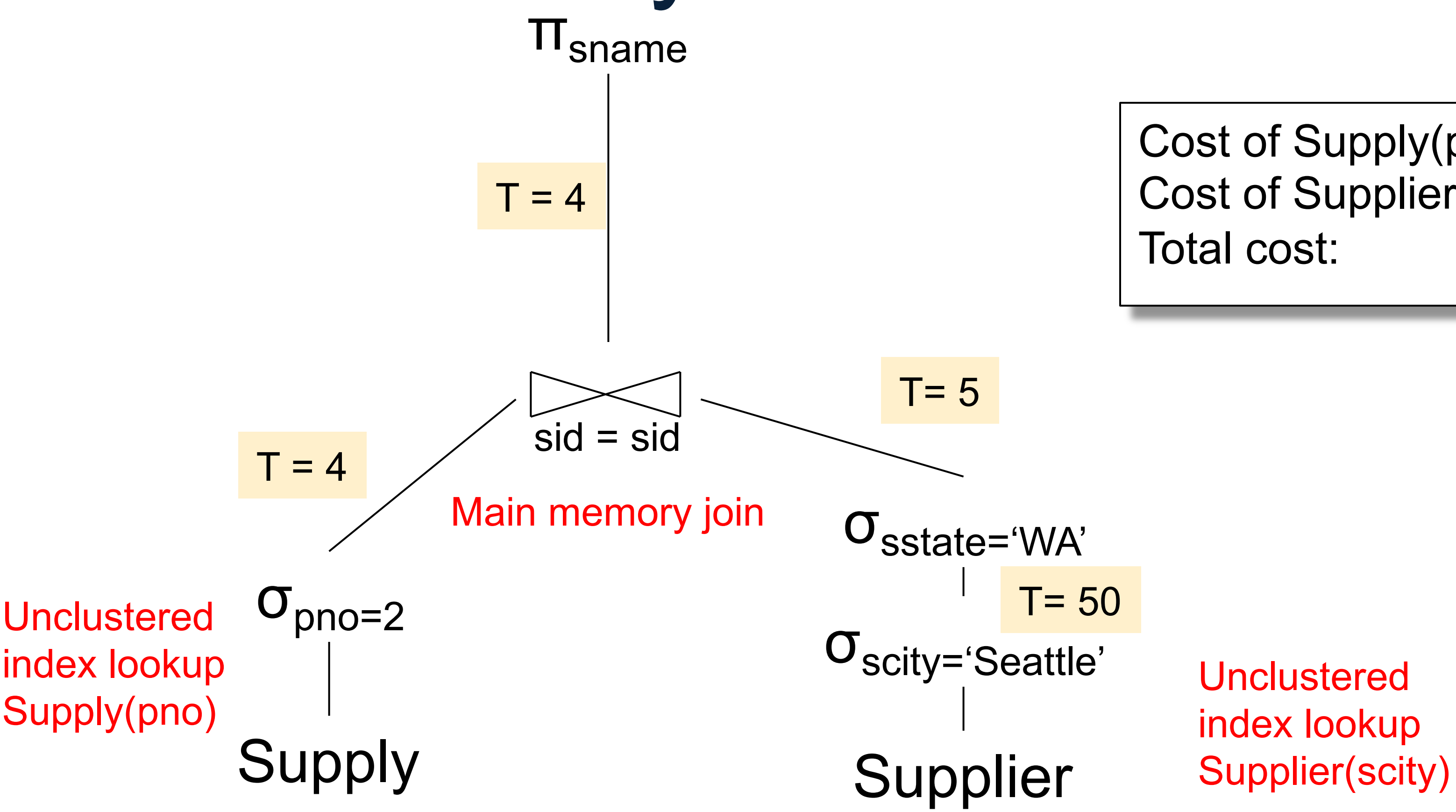
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:

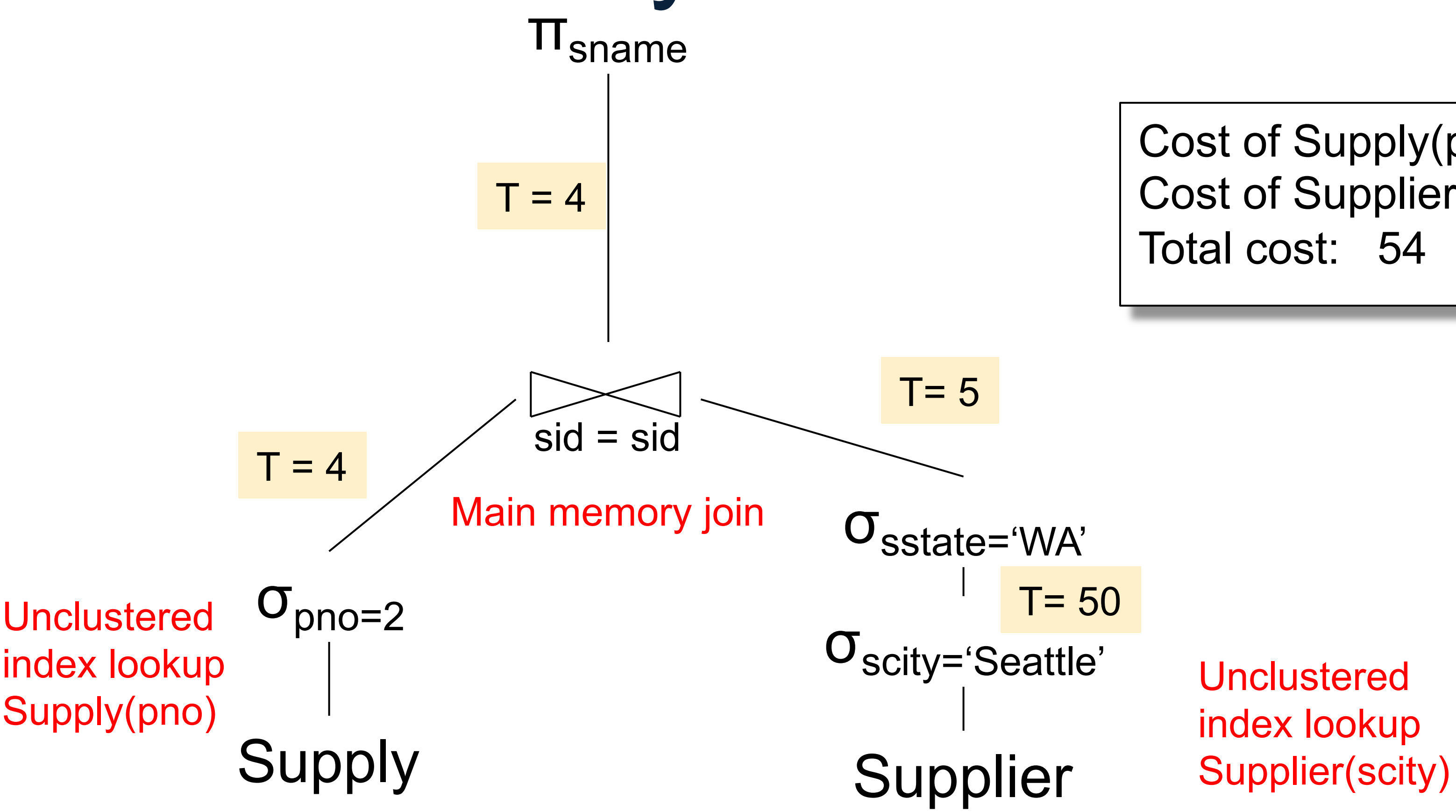
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

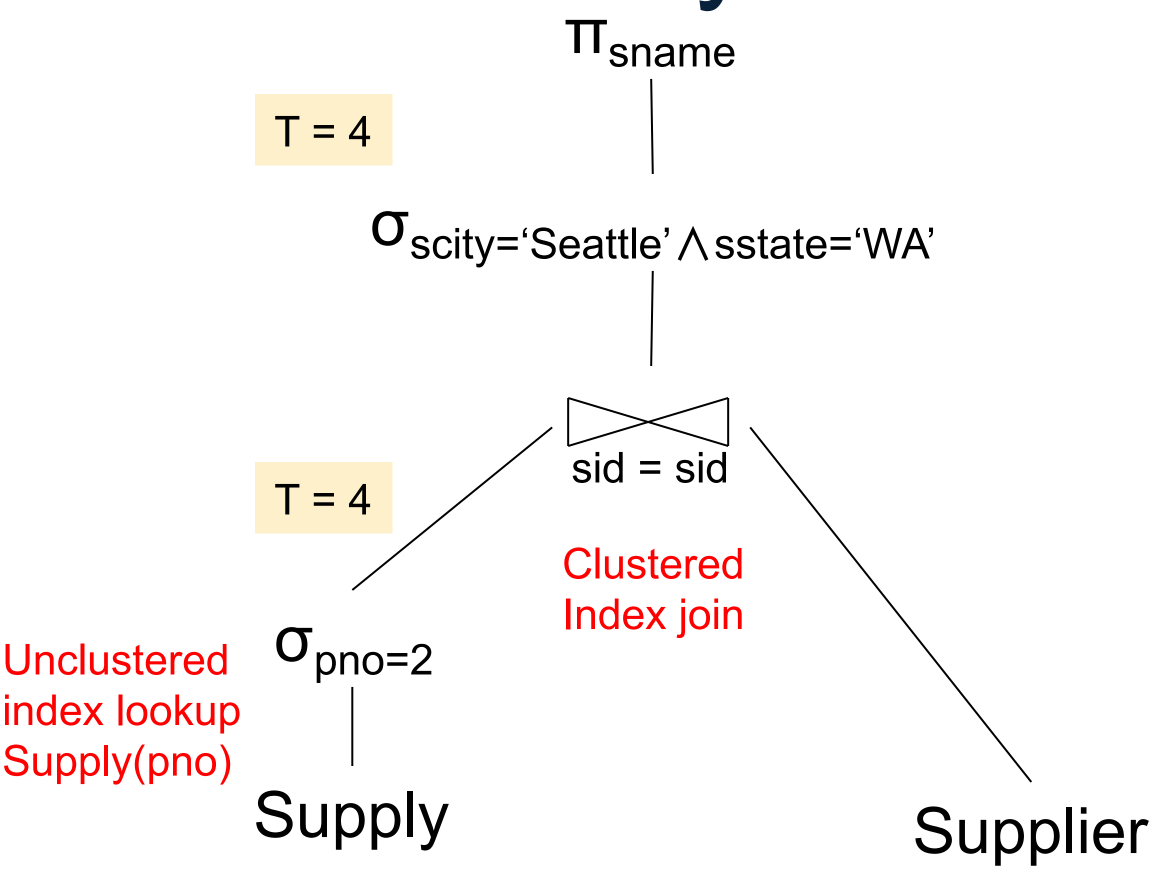
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) =
Cost of Index join =
Total cost:

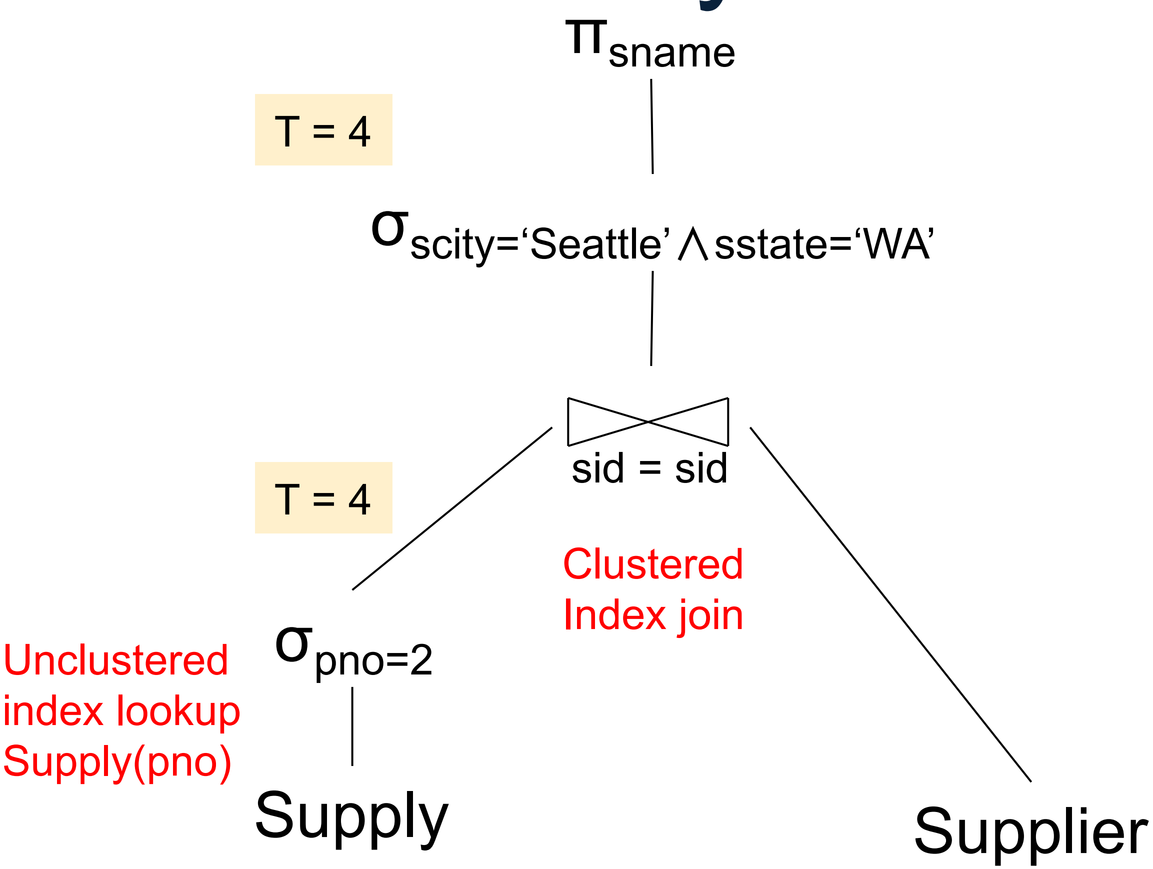
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

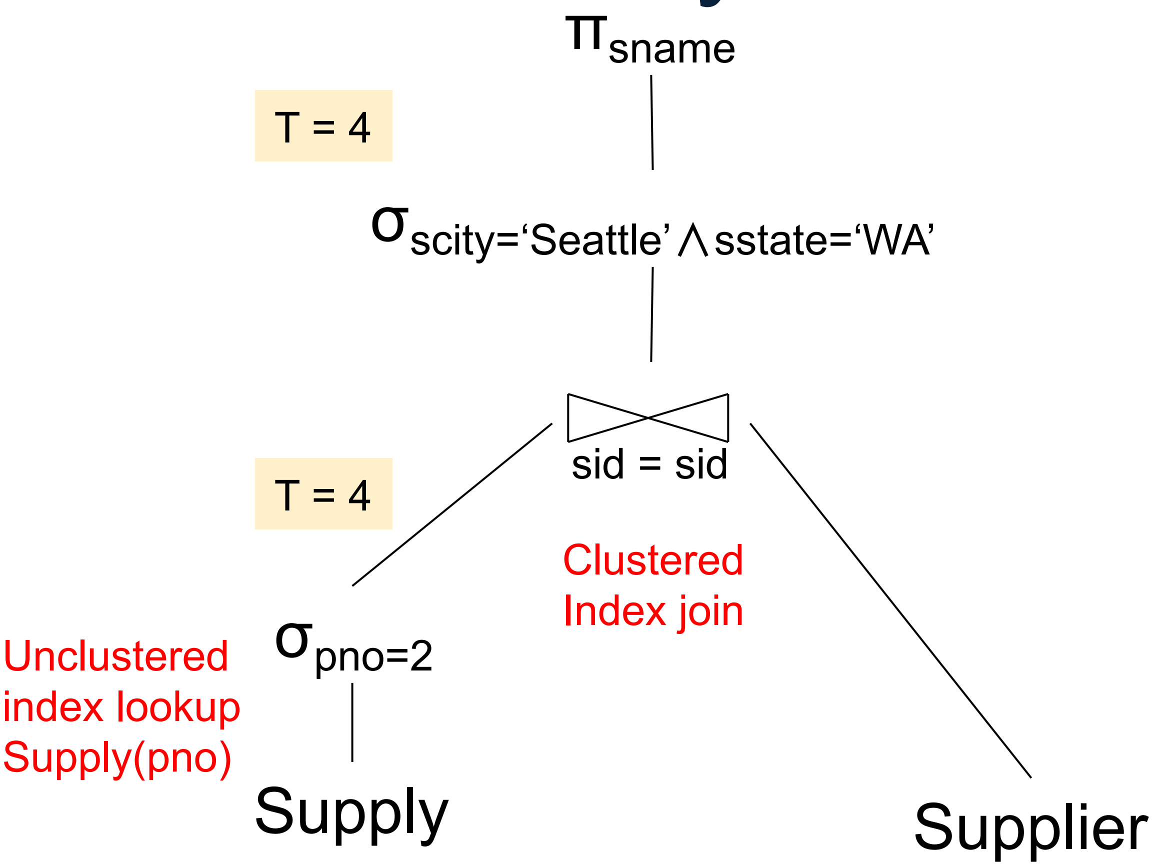
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Discussion

- We considered only IO cost;
  real systems need to consider IO+CPU
- Each system has its own hacks
- We assumed that all index pages were in memory: sometimes we need to add the cost of fetching index pages from disk

Histograms

- T(R), V(R,A) too coarse
- Histogram: separate stats per bucket

- In each bucket store:
  - T(bucket)
  - V(bucket,A) – optional

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

Assume V = 10

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

Assume V = 10

Estimate: 12000/10 = 1200

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500
V =	3	10	7	6	5	4

Estimate: 12000/10 = 1200

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500
V =	3	10	7	6	5	4

Estimate: ~~12000/10 = 1200~~ 12000/6 = 2000

Types of Histograms

- Eq-Width
- Eq-Depth
- Compressed: store outliers separately
- V-Optimal histograms

Employee(ssn, name, age)

Histograms

Eq-width:

Age:	0..20	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5000	12000	6500	500

Eq-depth:

Age:	0..32	33..41	42-46	47-52	53-58	> 60
Tuples	1800	2000	2100	2200	1900	1800

Compressed: store separately highly frequent values: (48,1900)

V-Optimal Histograms

- Error:

$$\sum_{v \in Domain(A)} \left(|\sigma_{A=v}(R)| - est_{Hist}\left(\sigma_{A=v}(R)\right) \right)^2$$

- Bucket boundaries = $\text{argmin}_{\text{Hist}}$(Error)
- Dynamic programming
- Modern databases systems use V-optimal histograms or some variations

Discussion

- Cardinality estimation = still unsolved
- Histograms:
  - Small number of buckets (why?)
  - Updated only periodically (why?)
  - No 2d histograms (except db2) why?
- Samples:
  - Fail for low selectivity estimates, joins
- Cross-join correlation – still unsolved

Yet Another Difficulties

SQL Queries issued from applications:

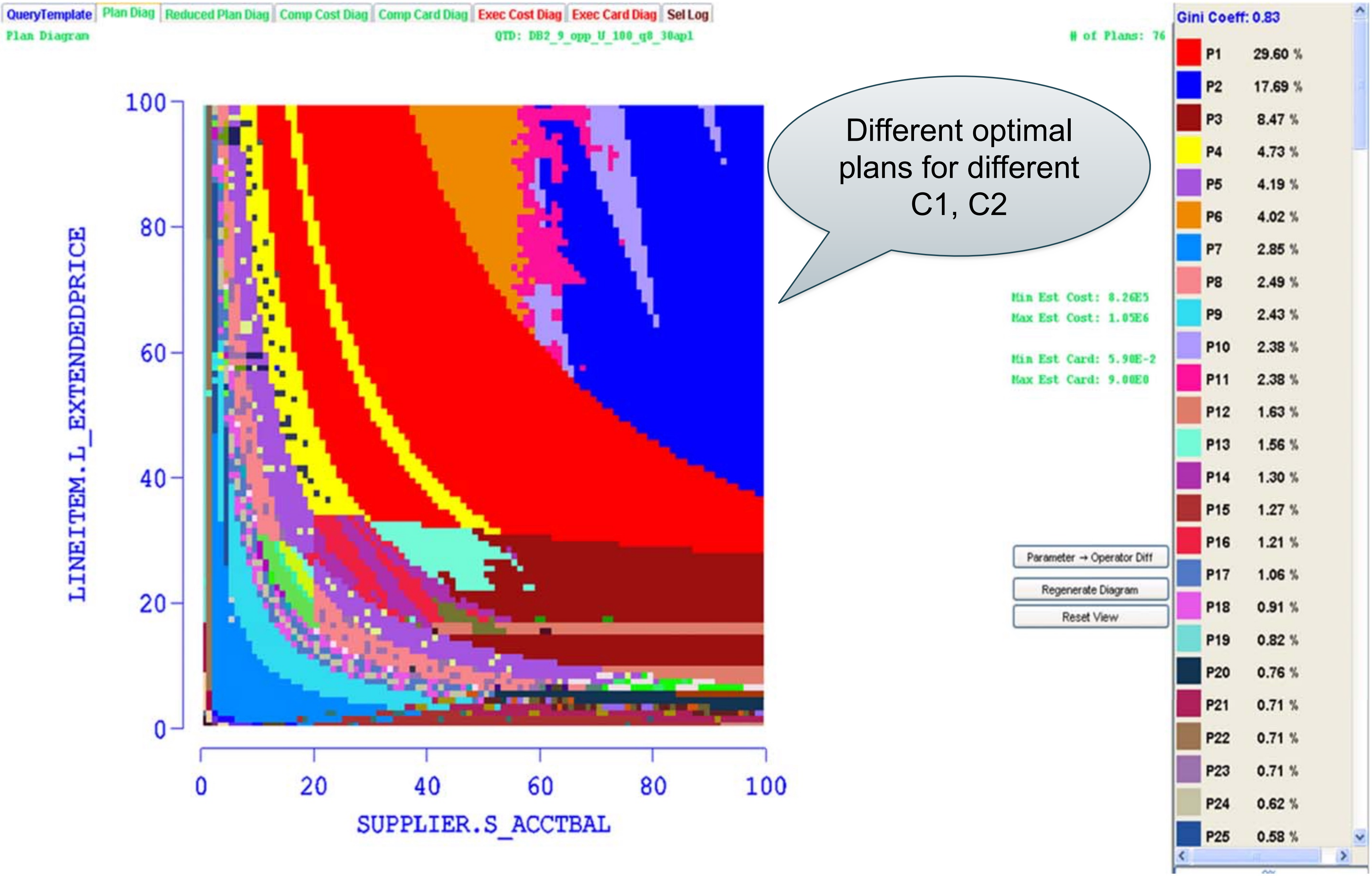
- Query is optimized once: *prepare*
- Then, executed repeatedly

Query constants are unknow until execution: optimized plan is suboptimal

```
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
 (select YEAR(o_orderdate) as o_year,
             l_extendedprice * (1 - l_discount) as volume,
                  n2.n_name as nation
  from part, supplier, lineitem, orders,
      customer, nation n1, nation n2, region
  where p_partkey = l_partkey and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between '1995-01-01'
    and '1996-12-31'
    and p_type = 'ECONOMY ANODIZED STEEL'
   and s_acctbal ≤ C1 and l_extendedprice ≤ C2 ) as all_nations
group by o_year order by o_year
```

```
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
 (select YEAR(o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
                 n2.n_name as nation
  from part, supplier, lineitem, orders,
      customer, nation n1, nation n2, region
  where p_partkey = l_partkey and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between '1995-01-01'
    and '1996-12-31'
    and p_type = 'ECONOMY ANODIZED STEEL'
   and s_acctbal ≤ C1 and l_extendedprice ≤ C2 ) as all_nations
group by o_year order by o_year
```

Optimize without knowing C1, C2


QueryTemplate
Plan Diag
Reduced Plan Diag
Comp Cost Diag
Comp Card Diag
Exec Cost Diag
Exec Card Diag
Sel Log
Plan Diagram
QTD: DB2_9_opp_U_100_q8_30apl
# of Plans: 76
LINEITEM.L_EXTENDEDPRICE
SUPPLIER.S_ACCTBAL
0
20
40
60
80
100
Different optimal plans for different C1, C2
Min Est Cost: 8.26E5
Max Est Cost: 1.05E6
Min Est Card: 5.90E-2
Max Est Card: 9.00E0
Parameter → Operator Diff
Regenerate Diagram
Reset View
Gini Coeff: 0.83
P1 29.60 %
P2 17.69 %
P3 8.47 %
P4 4.73 %
P5 4.19 %
P6 4.02 %
P7 2.85 %
P8 2.49 %
P9 2.43 %
P10 2.38 %
P11 2.38 %
P12 1.63 %
P13 1.56 %
P14 1.30 %
P15 1.27 %
P16 1.21 %
P17 1.06 %
P18 0.91 %
P19 0.82 %
P20 0.76 %
P21 0.71 %
P22 0.71 %
P23 0.71 %
P24 0.62 %
P25 0.58 %

Discussion

- Cardinality estimation: the weak spot of all query optimizers
- Other approaches:
  - Offline sampling: may lead to 0 estimates
  - Online sampling: sloooooooow
  - ML: large training data, model
  - Theoretical upper bounds: too coarse