CSE544
Data Management

Lecture15
Query Optimization – Part 4

Announcements

- HW3 due on Friday
- Review 6 due on Monday:
  - Read the entire paper
  - Will discuss in class
- Project: MAJOR milestone next Friday!

Query Optimization

Three major components:

1. Search space — last week
2. Cardinality and cost estimation — last lecture
3. Plan enumeration algorithms — today

Recap: Selectivity Factors

Uniformity+Containment of values

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

Equality:

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

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

Range:

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

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

Independence assumption

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

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

Recap: Selectivity Factors

Join

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

Why???

Alternate CE Methods: ML, Sampling

Upper bounds: next week

ML-Based CE

- A trend in research: use some ML model to do cardinality estimation
- Main hope: remove independence and uniformity assumptions
- Data-driven or Query-driven

Data-Driven Estimators

- Train a generative ML model that represents p(A,B,C,…) from the database instance
- Map query Q to hyperrectangle
- Problem: space is over all attributes
  - "Full outer join"
  - Lots of complications to make this work

Query-Driven Estimators

- Training set is a set of pairs (Q, |Q|)
- All predicates in the WHERE condition need to be featurized, embedded
- Train a discriminative model for Est(Q)

Discussion

Problems with ML-based CE:

- Large model size: 1MB – 1GB
- Inference time varies
- Updates, change in skewness, correlations
- Not explainable

Consensus: not ready for production

Sampling-Based CE

- Main idea: use a sample of the database to estimate the output size
- Warning: different probability space!
  - Sampler v.s. data distribution
  - Sampler is a better space: can repeat

Offline Sampling or Online Sampling

Offline Sampling

- Compute uniform sample $R_{sample} \subseteq R$
- To estimate |Q(DB)|:

Offline Sampling

- Compute uniform sample $R_{sample} \subseteq R$
- To estimate |Q(DB)|, use Horwitz-Thompson:

$$Est\big(Q(DB)\big) = \frac{|R|}{|R_{sample}|} \cdot \frac{|S|}{|S_{sample}|} \cdots |Q\big(DB_{sample}\big)|$$

Offline Sampling

- Compute uniform sample $R_{sample} \subseteq R$
- To estimate |Q(DB)|, use Horwitz-Thompson:

$$Est\big(Q(DB)\big) = \frac{|R|}{|R_{sample}|} \cdot \frac{|S|}{|S_{sample}|} \cdots |Q\big(DB_{sample}\big)|$$

- The missing tuple problem: $Q\big(DB_{sample}\big) = 0$
  → Still unbiased (why??) but high variance

Online Sampling

- To estimate $R \bowtie S$:
  - Sample a tuple from R
  - Sample a matching tuple from S
  - Repeat
- Need to use index on S
- WanderJoin: next slides show how to estimate $R \bowtie S \bowtie T$ using random walk

R

A	B
…	1
…	2
…	1
…	3
…	4
…	2
…	4

R

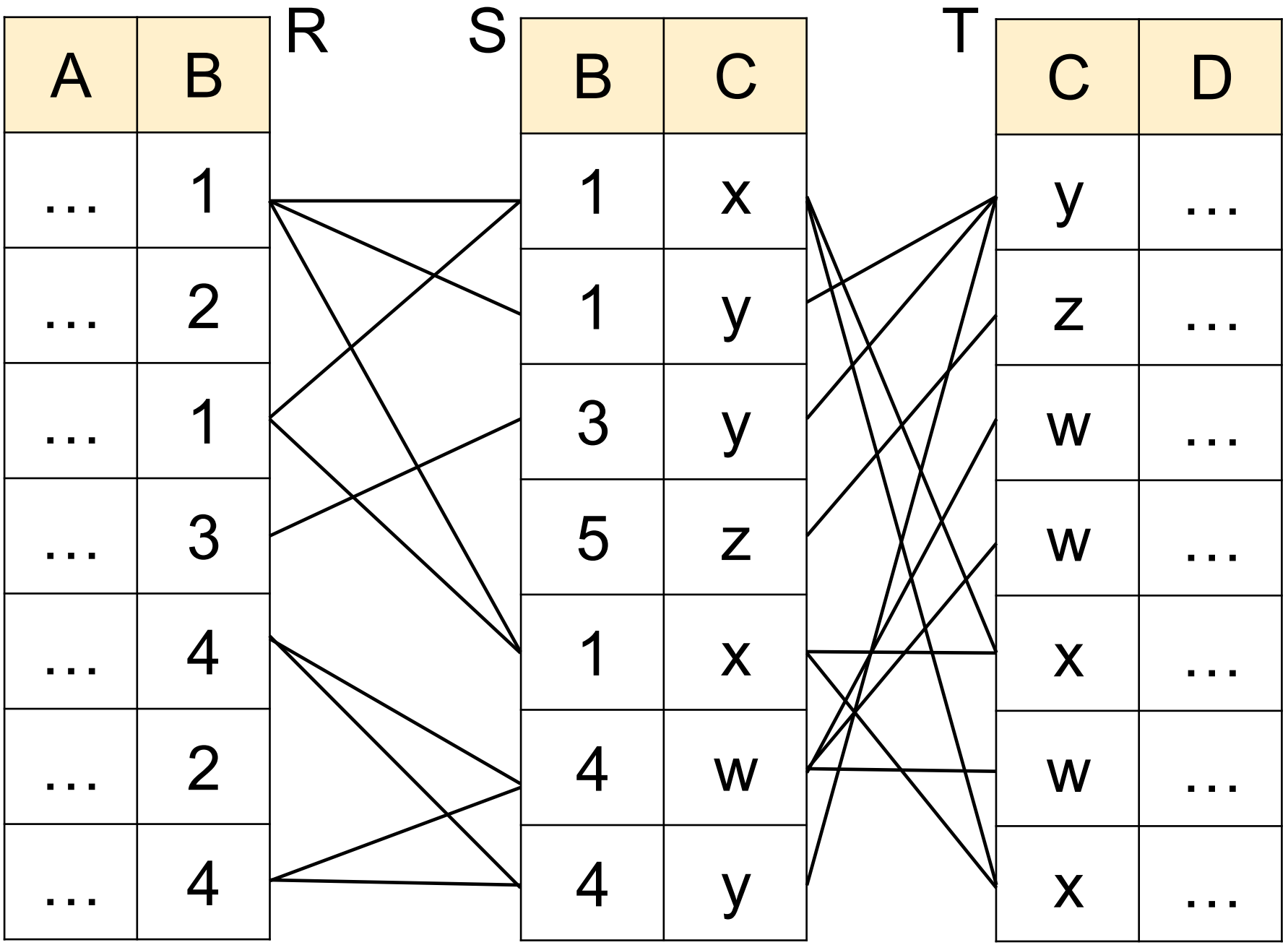
A	B
…	1
…	2
…	1
…	3
…	4
…	2
…	4

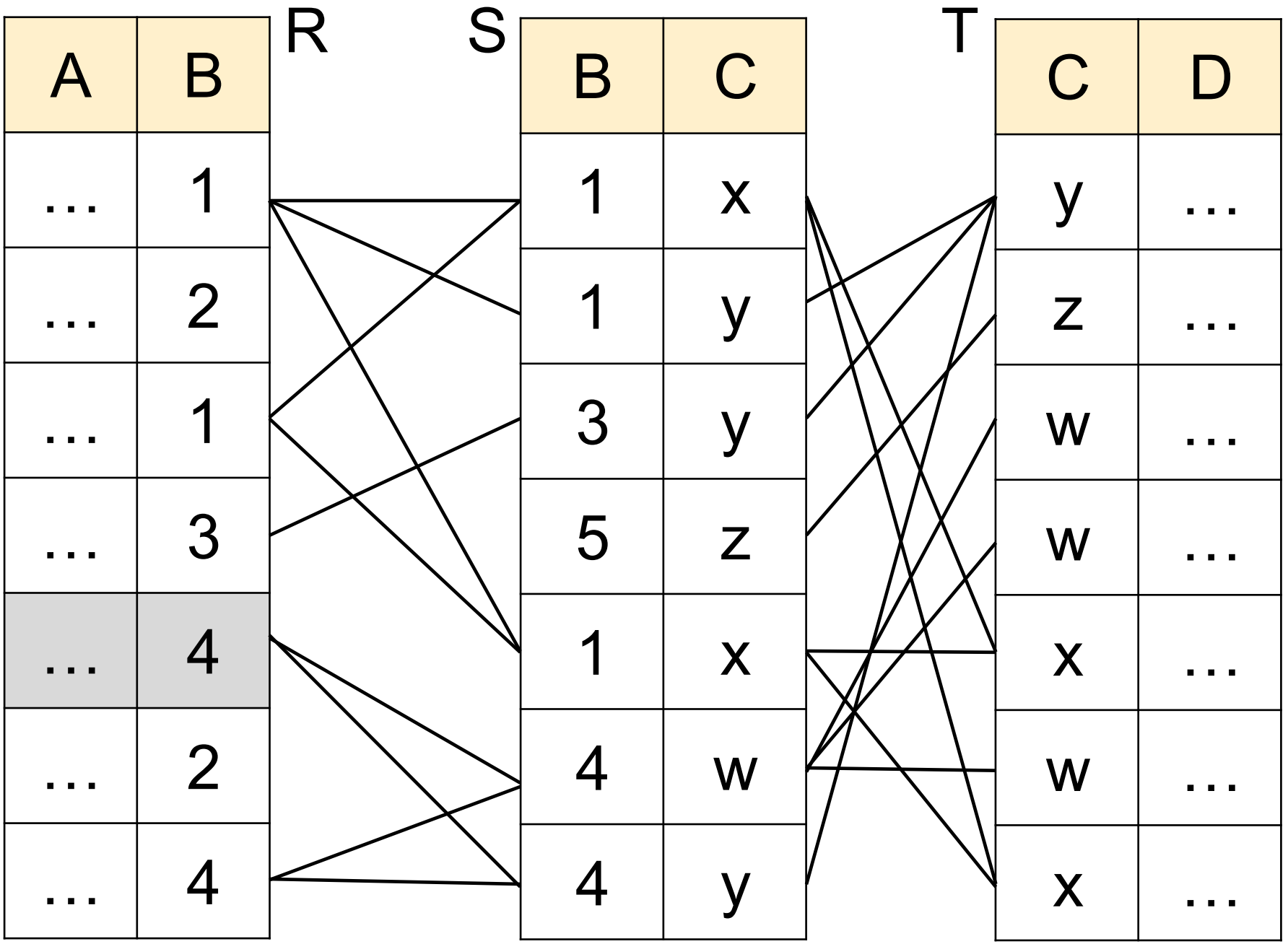
S

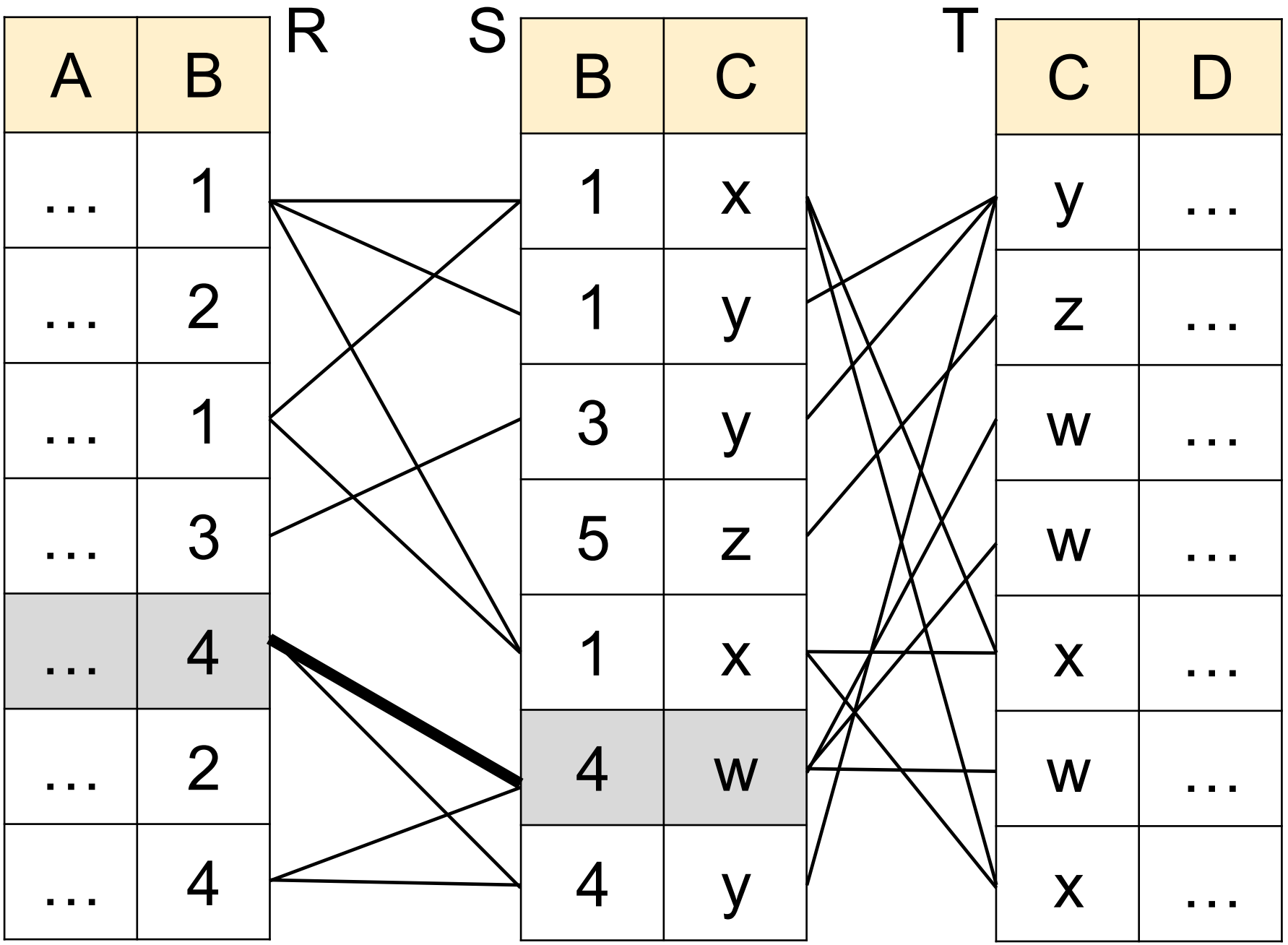
B	C
1	x
1	y
3	y
5	z
1	x
4	w
4	y

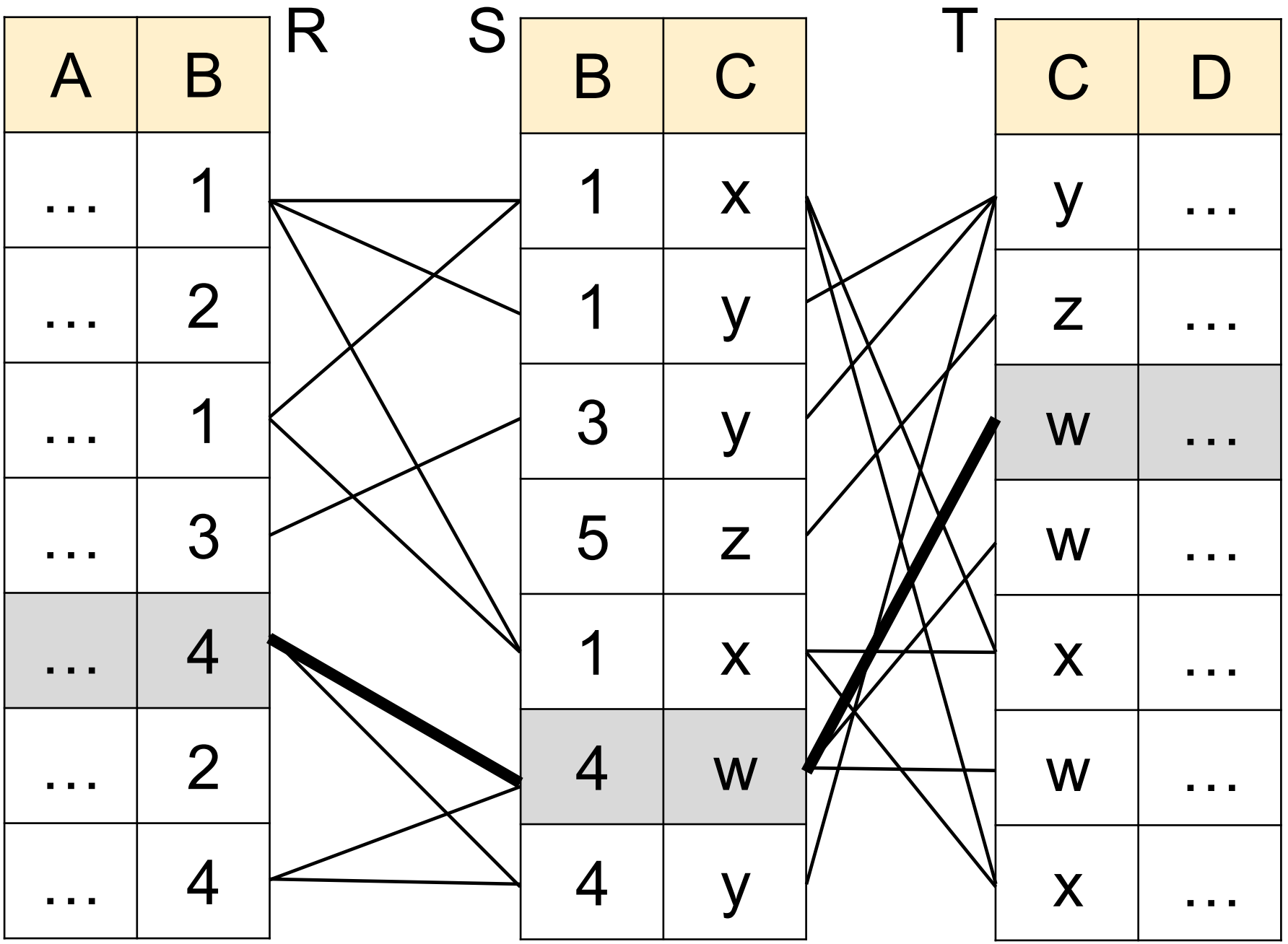
T

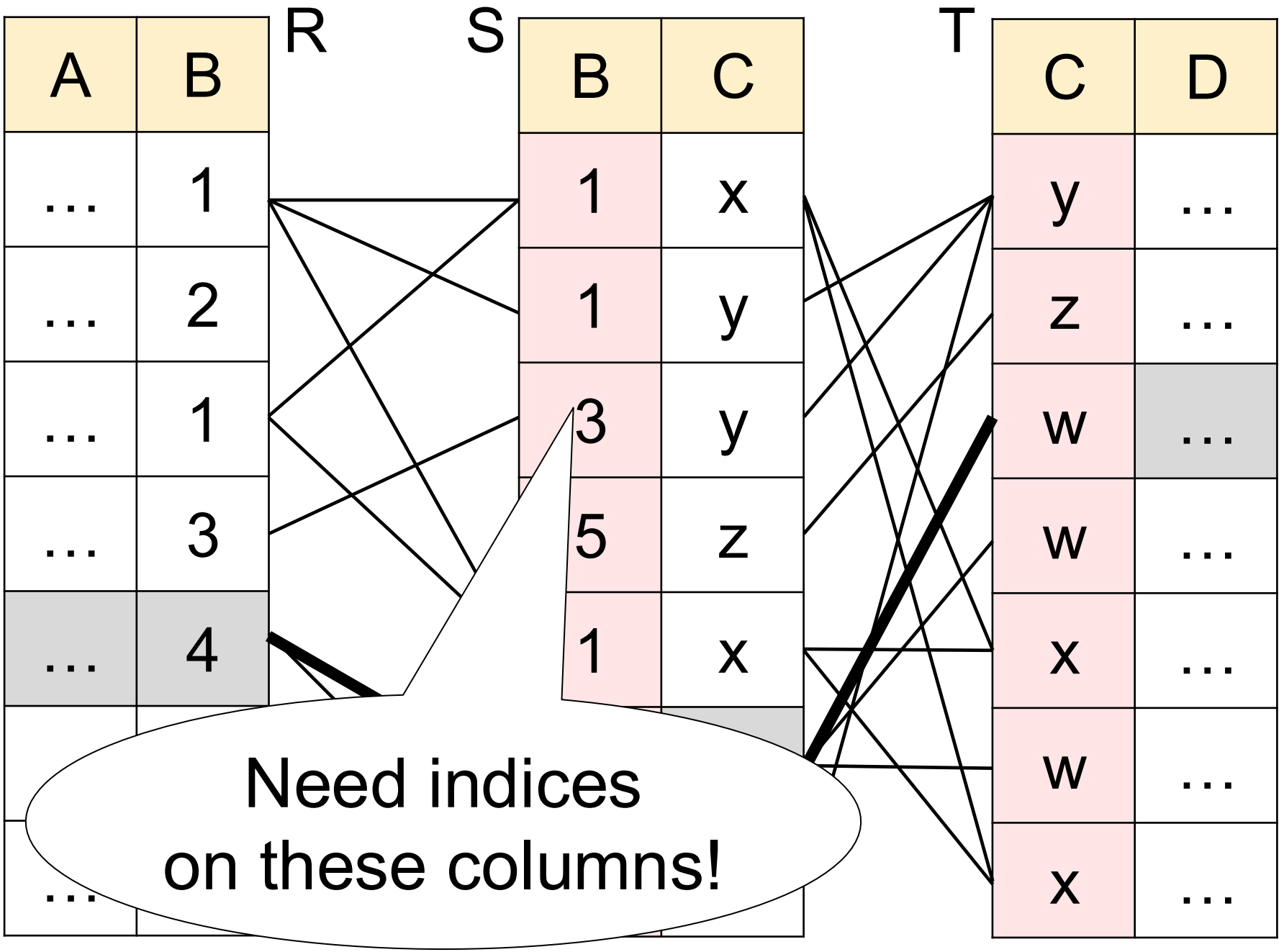
C	D
y	…
z	…
w	…
w	…
x	…
w	…
x	…

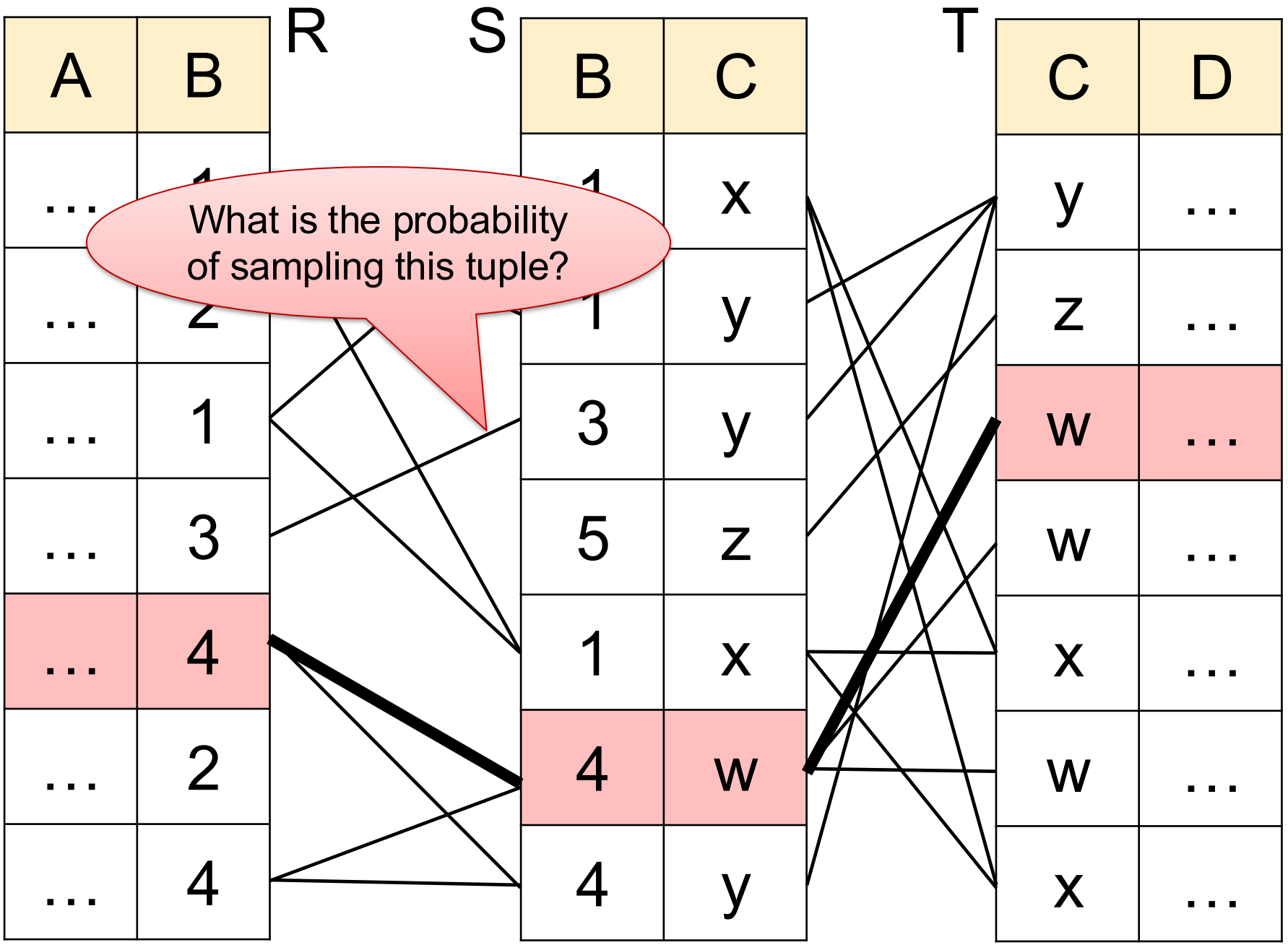

R
S
T
A
B
1
2
1
3
4
2
4
B
C
1
x
1
y
3
y
5
z
1
x
4
w
4
y
C
D
y
…
z
…
w
…
w
…
x
…
w
…
x
…

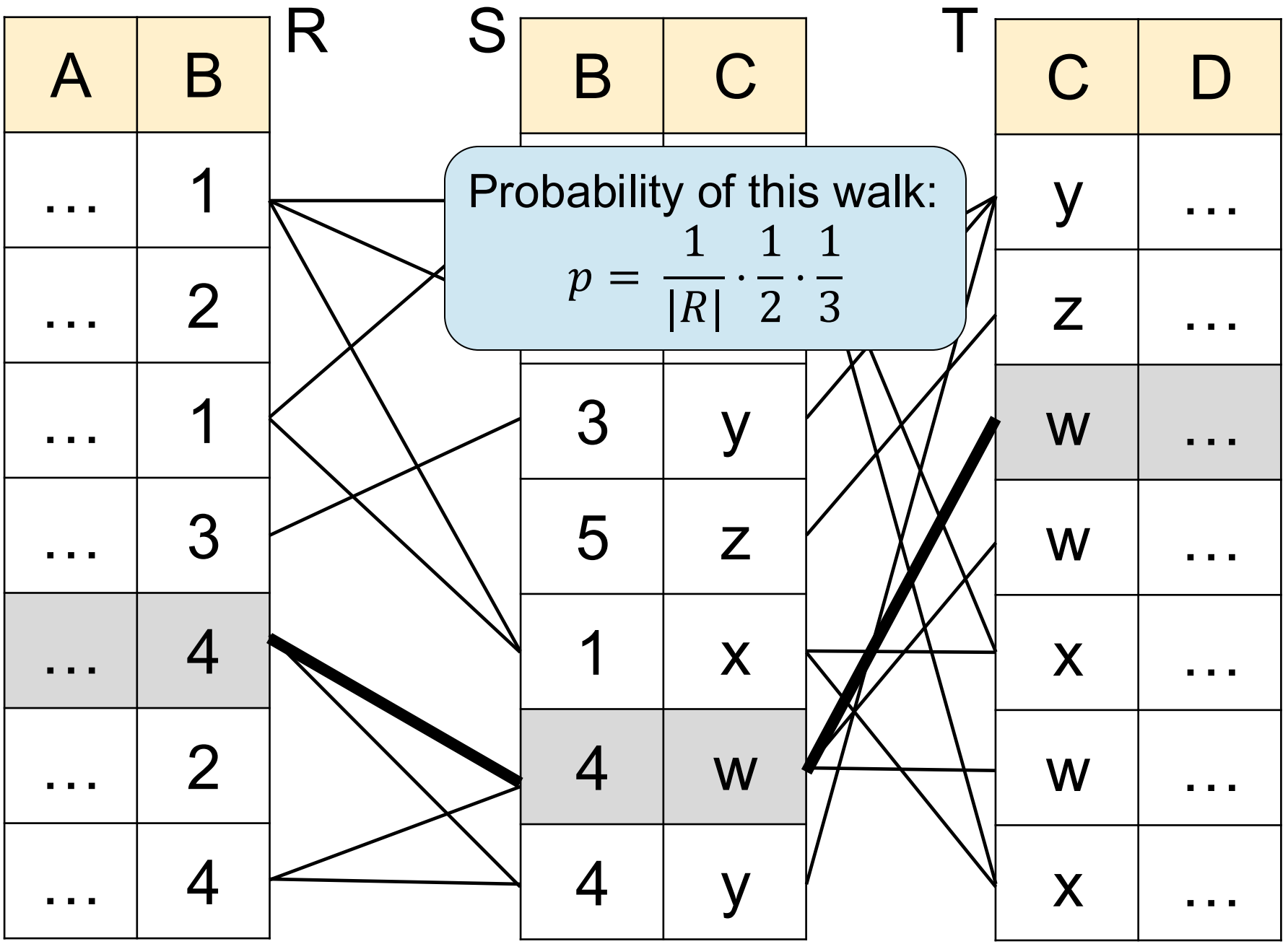

R
S
T
A
B
...
1
...
2
...
1
...
3
...
4
...
2
...
4
B
C
1
x
1
y
3
y
5
z
1
x
4
w
4
y
C
D
y
...
z
...
w
...
w
...
x
...
w
...
x
...

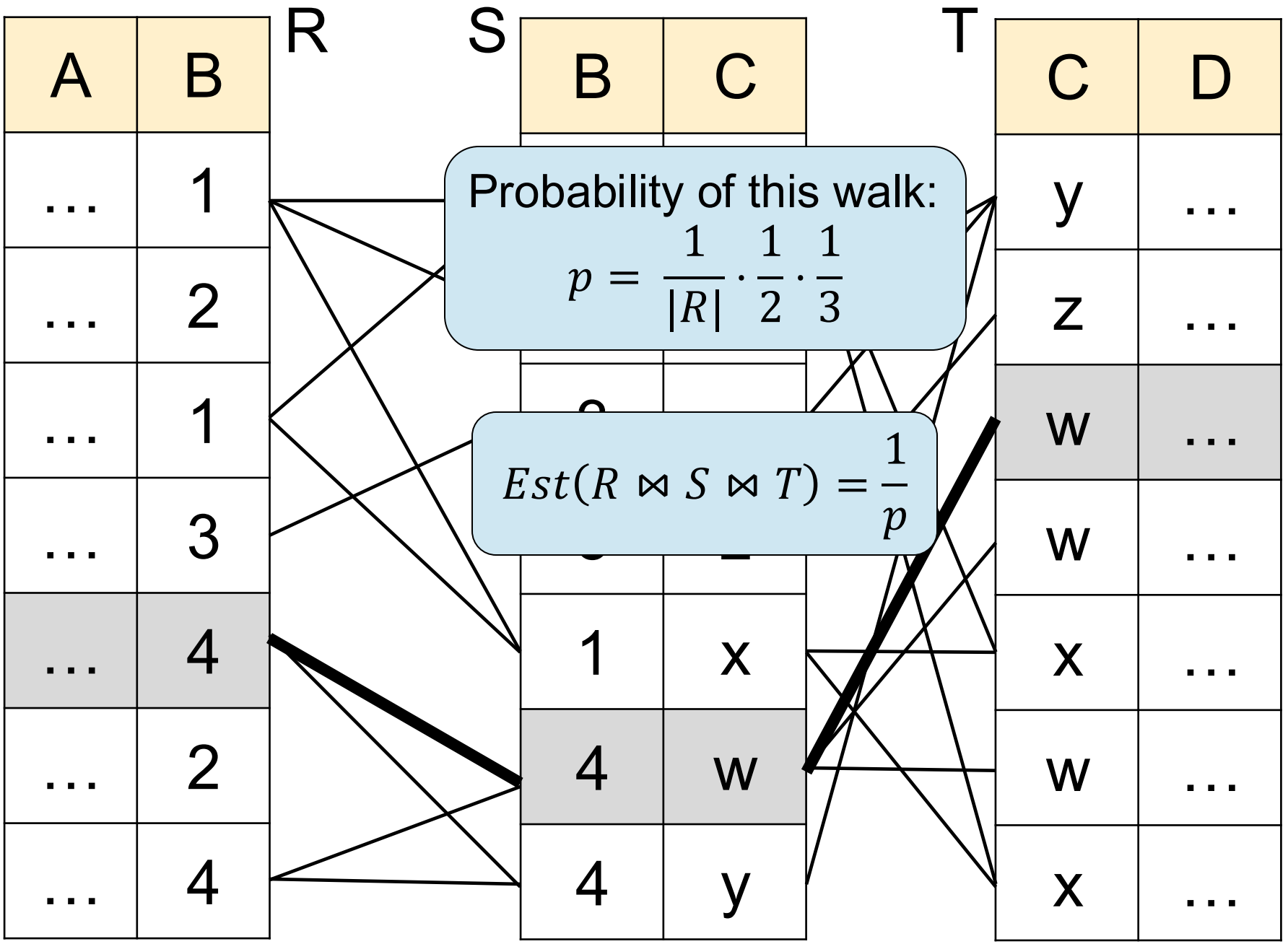

R
S
T
A
B
...
1
...
2
...
1
...
3
...
4
...
2
...
4
B
C
1
x
1
y
3
y
5
z
1
x
4
w
4
y
C
D
y
...
z
...
w
...
w
...
x
...
w
...
x
...

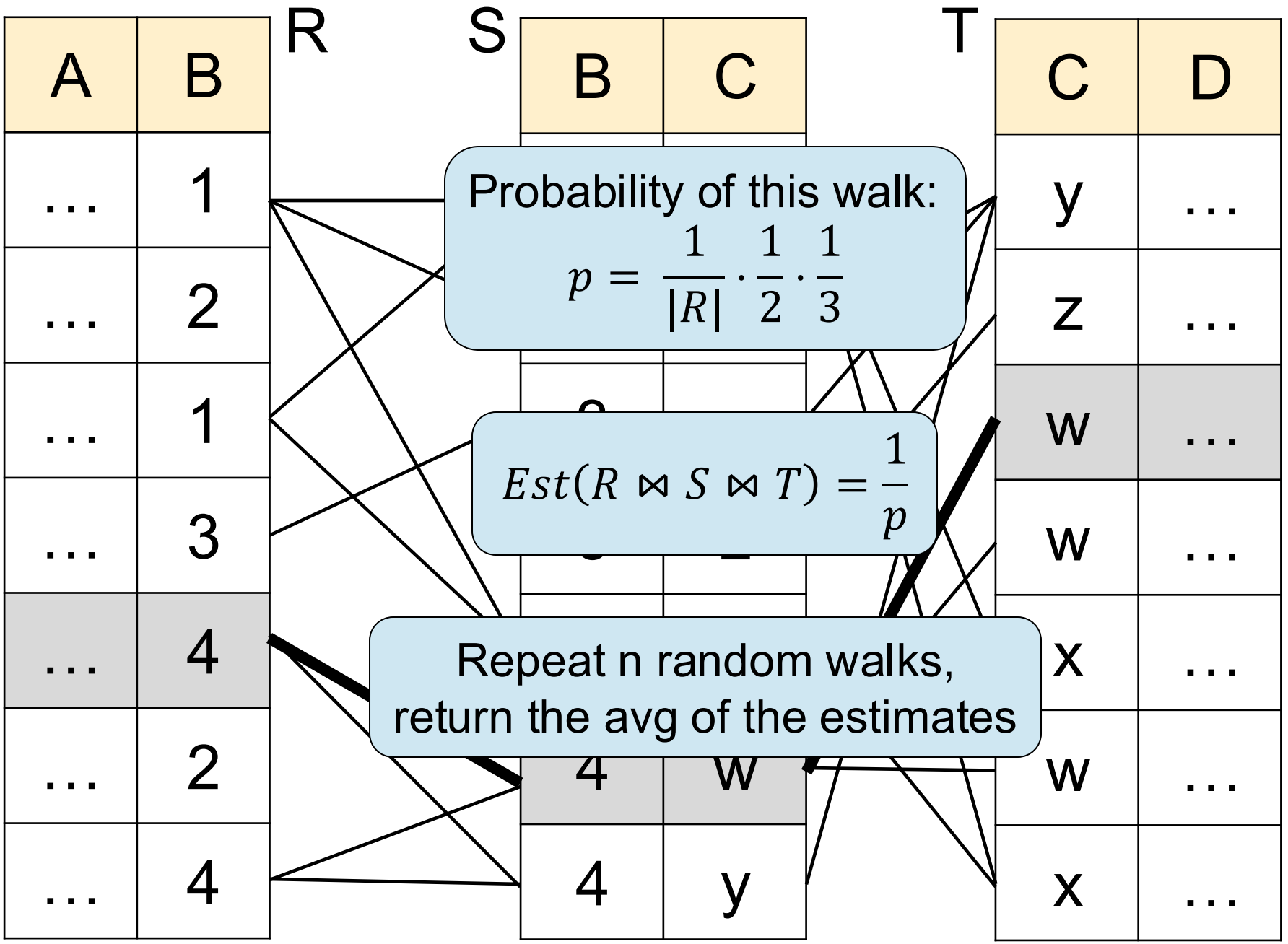

R
S
T
A
B
1
2
1
3
4
2
4
B
C
1
x
1
y
3
y
5
z
1
x
4
w
4
y
C
D
y
…
z
…
w
…
w
…
x
…
w
…
x
…


R
S
T
A
B
B
C
C
D
...
1
2
1
3
4
1
1
3
5
1
x
y
y
z
x
y
z
w
w
x
w
x
...
...
...
...
...
...
...
...
...
...
...
...
Need indices
on these columns!


R
S
T
A
B
B
C
C
D
What is the probability of sampling this tuple?


R
S
T
A
B
B
C
C
D
…
1
…
2
…
1
…
3
…
4
…
2
…
4
3
y
5
z
1
x
4
w
4
y
y
…
z
…
w
…
w
…
x
…
w
…
x
…
Probability of this walk:
$p = \frac{1}{|R|} \cdot \frac{1}{2} \cdot \frac{1}{3}$


R
S
T
A
B
B
C
C
D
…
1
…
2
…
1
…
3
…
4
…
2
…
4
1
x
4
w
4
y
y
…
z
…
w
…
w
…
x
…
w
…
x
…
Probability of this walk:
$p = \frac{1}{|R|} \cdot \frac{1}{2} \cdot \frac{1}{3}$
$Est(R \bowtie S \bowtie T) = \frac{1}{p}$


R
A
B
…
1
…
2
…
1
…
3
…
4
…
2
…
4
S
B
C
4
w
4
y
T
C
D
y
…
z
…
w
…
w
…
x
…
w
…
x
…
Probability of this walk:
$p = \frac{1}{|R|} \cdot \frac{1}{2} \cdot \frac{1}{3}$
$Est(R \bowtie S \bowtie T) = \frac{1}{p}$
Repeat n random walks,
return the avg of the estimates

Discussion

- Sampling based CE:
  - Unbiased, guaranteed error bounds
- However:
  - Offline samples need to be too large
  - Online samples require access to indices
- Some systems use offline sample for single-table predicates only

Plan Enumeration Algorithm

Two Types of Plan Enumeration Algorithms

- Dynamic programming
  - Based on System R [Selinger 1979]
  - *Join reordering algorithm*
- Cascades optimizer

System R Optimizer

- Optimizes single-block SQL query

System R Optimizer

- Optimizes single-block SQL query
- Push selections down

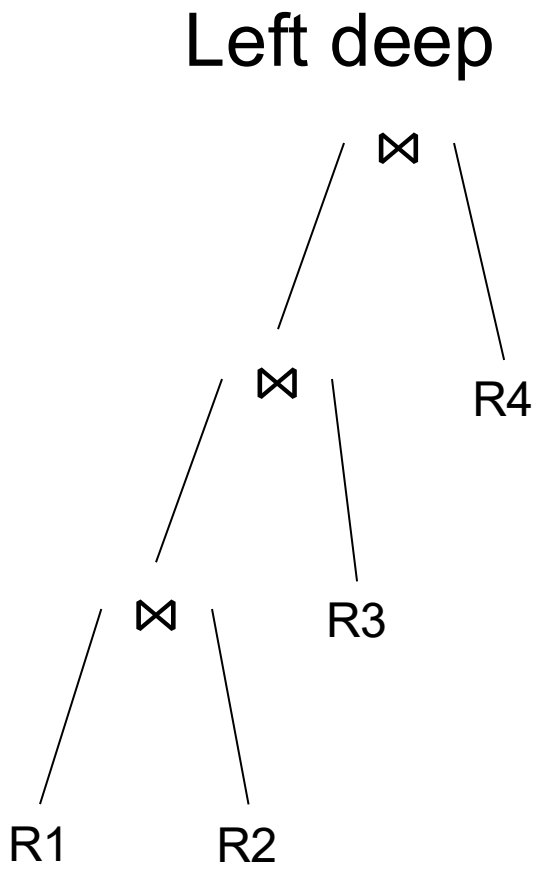
System R Optimizer

- Optimizes single-block SQL query
- Push selections down
- Goal: optimize the join order
  $(R_1 \bowtie R_2) \bowtie R_3$ vs. $R_2 \bowtie (R_1 \bowtie R_3)$

System R Optimizer

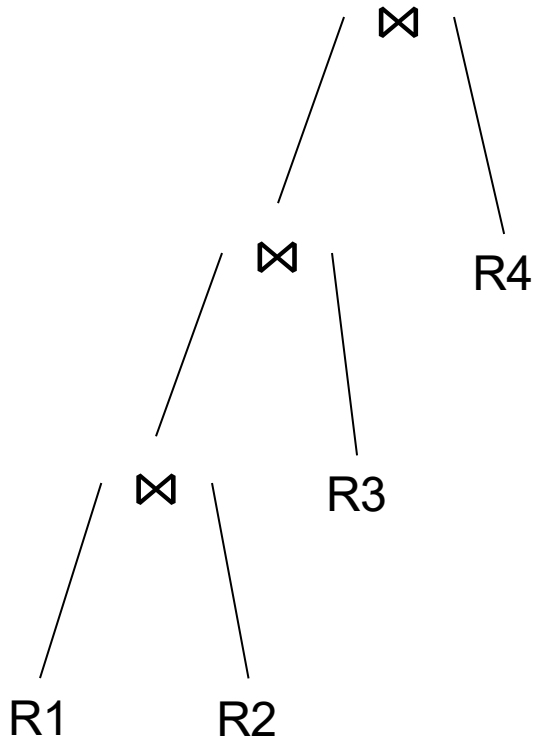
- Optimizes single-block SQL query
- Push selections down
- Goal: optimize the join order
  $(R_1 \bowtie R_2) \bowtie R_3$ vs. $R_2 \bowtie (R_1 \bowtie R_3)$
- Dynamic programming

Types of Query Plans

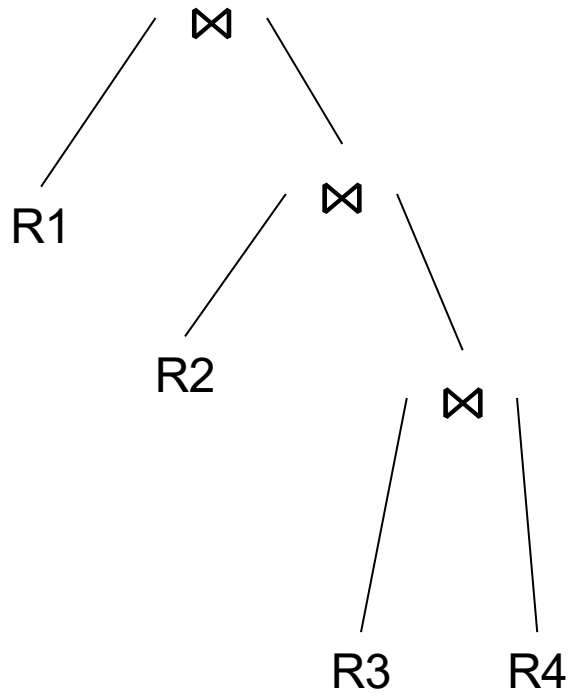


Types of Query Plans

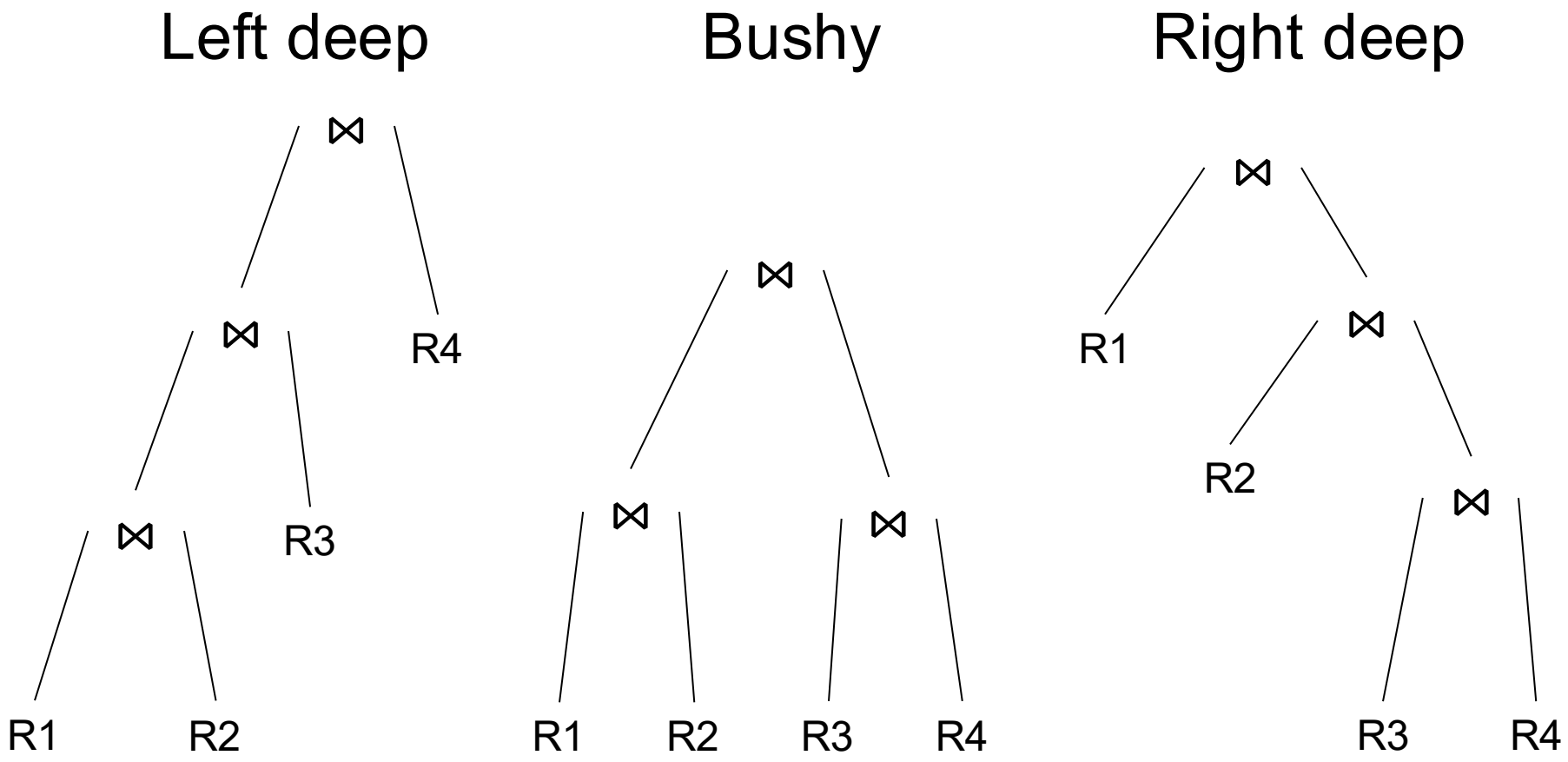
Left deep



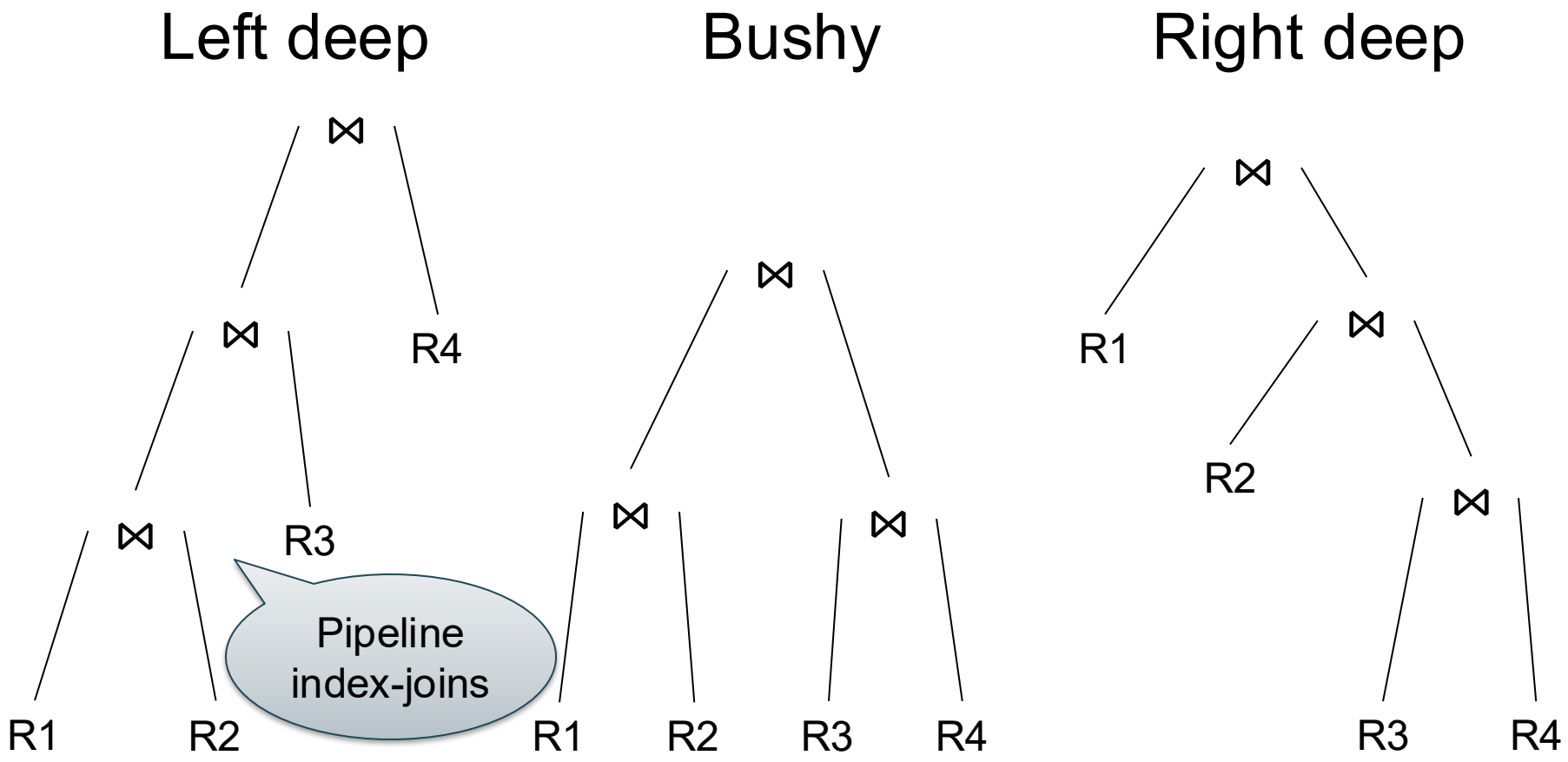
Right deep



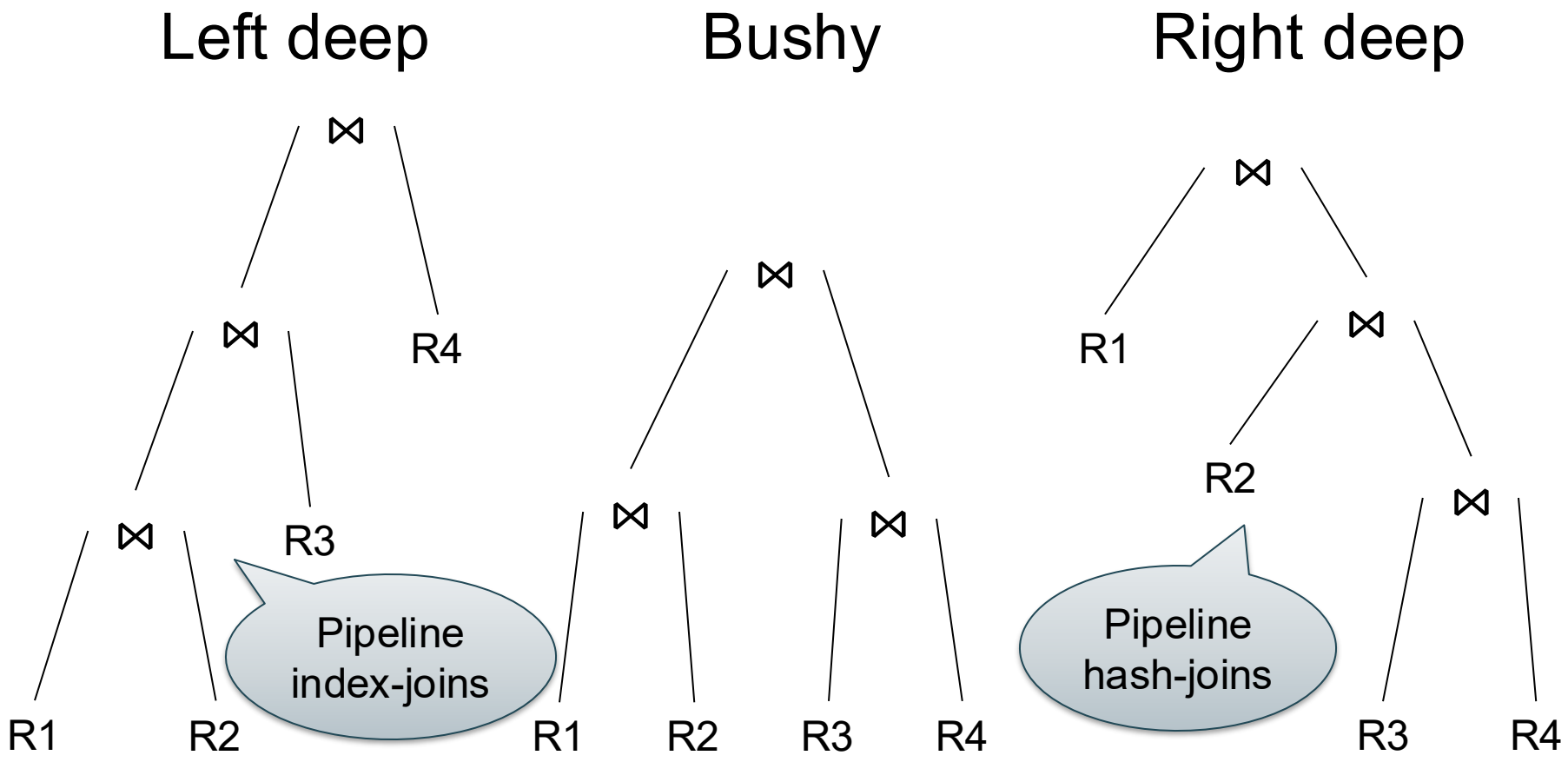
Types of Query Plans



Types of Query Plans



Types of Query Plans



System R Optimizer

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$, compute best plan:

- Step 1: $Q = \{R_1\}, \{R_2\}, \ldots, \{R_n\}$
- Step 2: $Q = \{R_1,R_2\}, \{R_1,R_3\}, \ldots, \{R_{n-1}, R_n\}$
- …
- Step n: $Q = \{R_1, \ldots, R_n\}$

Example

Best plan for {R1, R2, R3, R4, R5}:

Example

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …

Example

Best plan for {R1, R2, R3, R4, R5}:

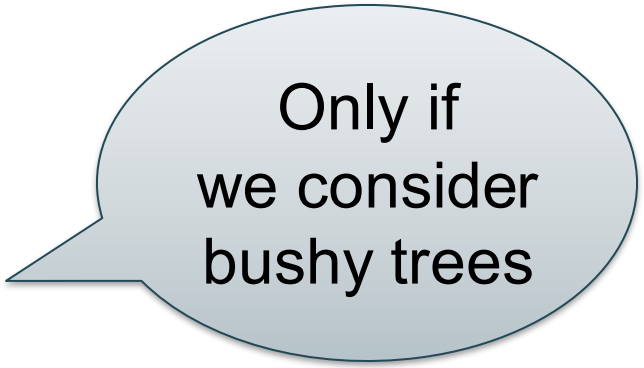
- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …

Optimal plan for R2,R3,R4,R5 is already in the table

Example

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …
- {R1,R2,R3} ⋈ {R4,R5}
- …



Example

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …
- {R1,R2,R3} ⋈ {R4,R5}
- …

Estimate their cost, keep the minimum

Example

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …
- {R1,R2,R3} ⋈ {R4,R5}
- …

Estimate their cost, keep the minimum

Store in the entry for {R1, R2, R3, R4, R5}

System R Optimizer

For each subquery Q $\subseteq \{R_1, \ldots, R_n\}$ store:

- Estimated Size(Q)
- A best plan for Q: Plan(Q)
- The cost of that plan: Cost(Q)

One plan for each "interesting order"

System R Optimizer

Step 1: single relations $\{R_1\}$, $\{R_2\}$, …, $\{R_n\}$

- Consider all possible access paths:
  - Sequential scan, or
  - Index 1, or
  - Index 2, or
  - …
- Keep optimal plan for each "interesting order"

System R Optimizer

Step k = 2…n:

For each $Q = \{R_{i_1}, \ldots, R_{i_k}\}$

- For all plans of the form $P = P_1 \bowtie P_2$:
  - $Cost(P) = Cost(\bowtie) + Cost(P_1) + Cost(P_2)$
  - Keep the cheapest plan, or
  - Keep multiple plans, for “interesting orders”

Runtime: exponential in n.
Mitigated by: no cartesian products, restricted tree shapes
Let’s analyze for some special query shapes…

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

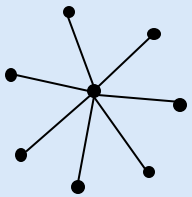
Some Popular Query Shapes

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

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

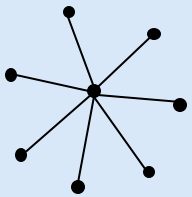


Some Popular Query Shapes

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

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

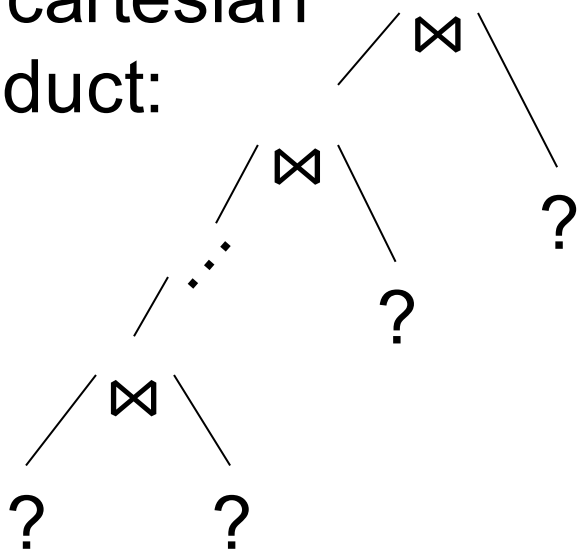
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

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:

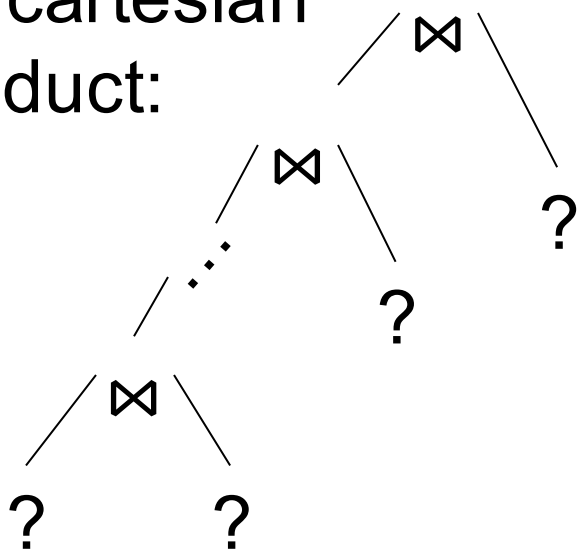


Number of Left-Deep Plans

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

W/ cartesian product:



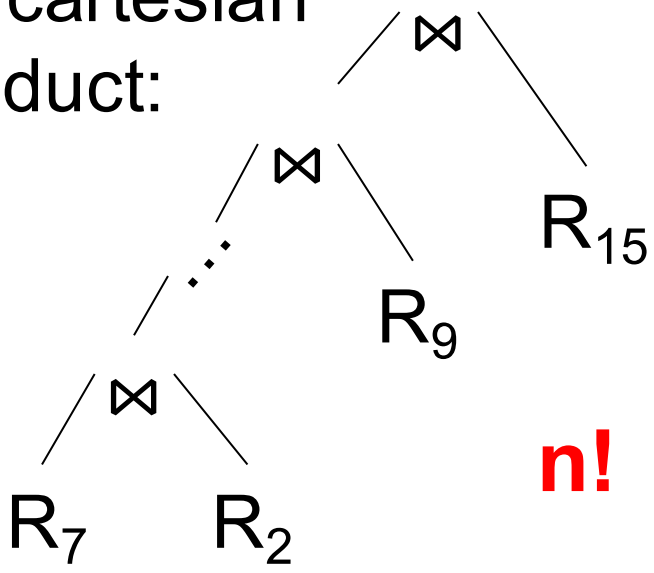
Can place relations in any order

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

W/ cartesian product:



Can place relations in any order

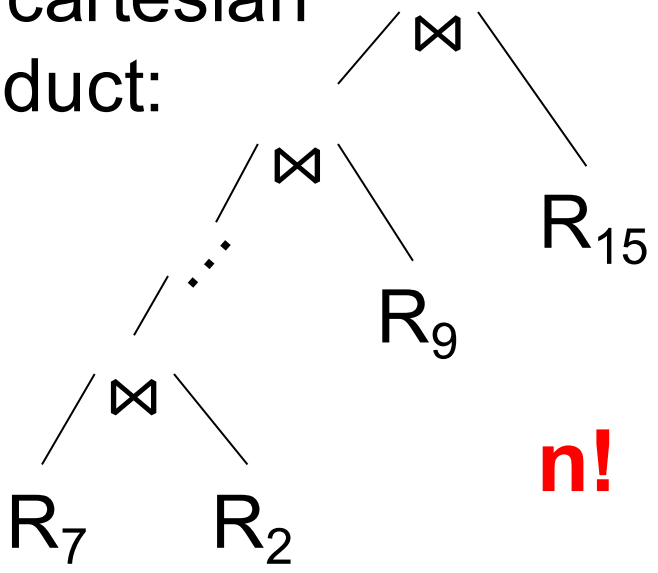
n!

Number of Left-Deep Plans

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

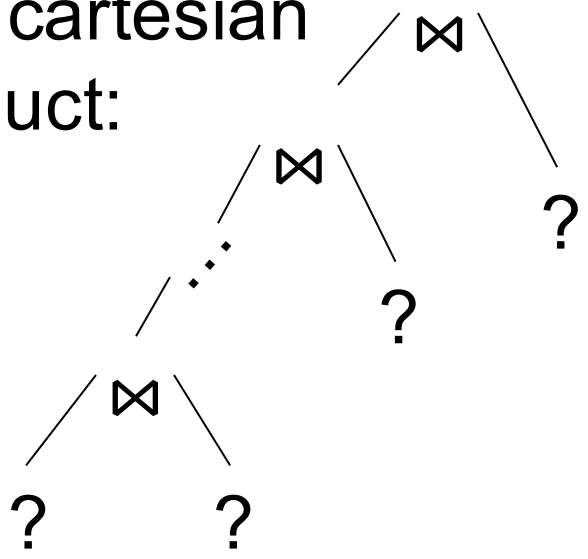
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:



n!

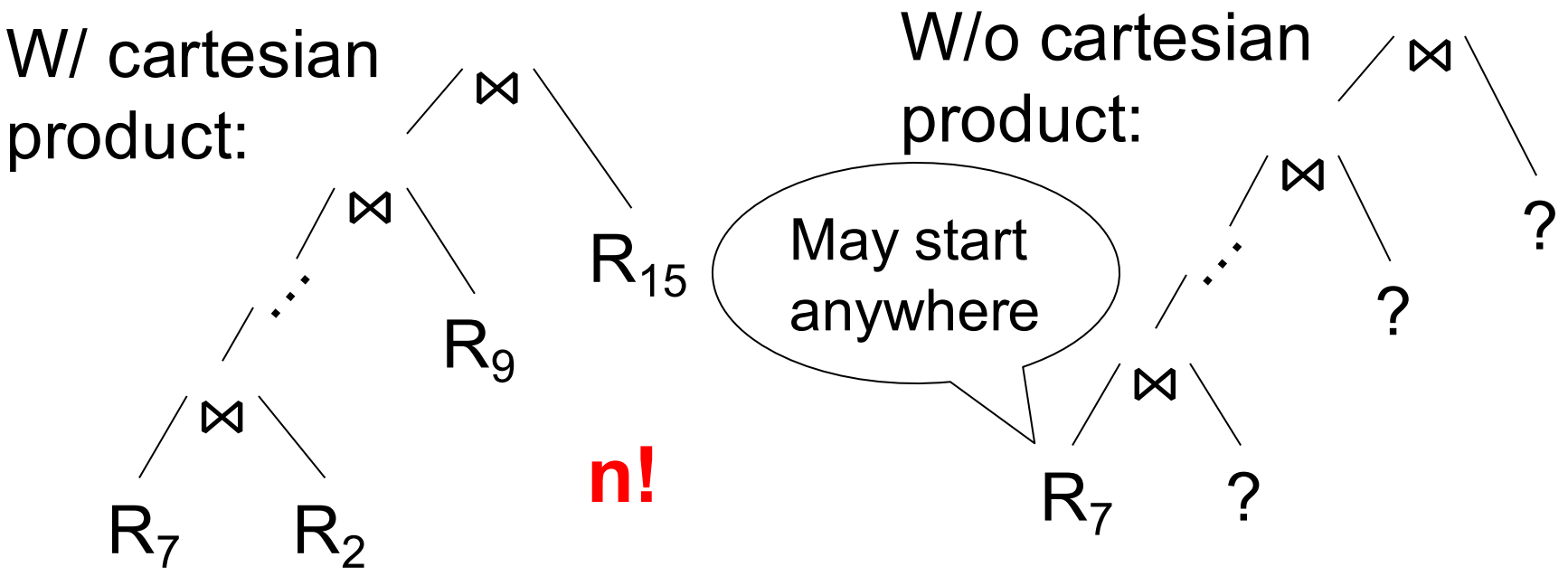
W/o cartesian product:



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

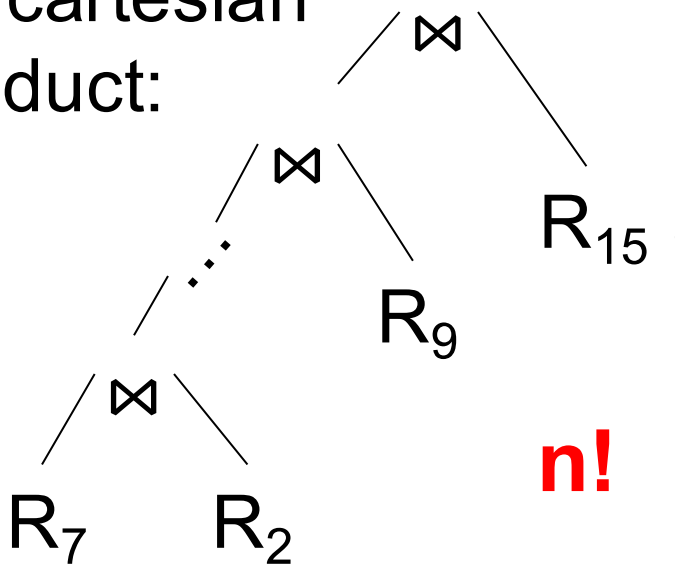


Number of Left-Deep Plans

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

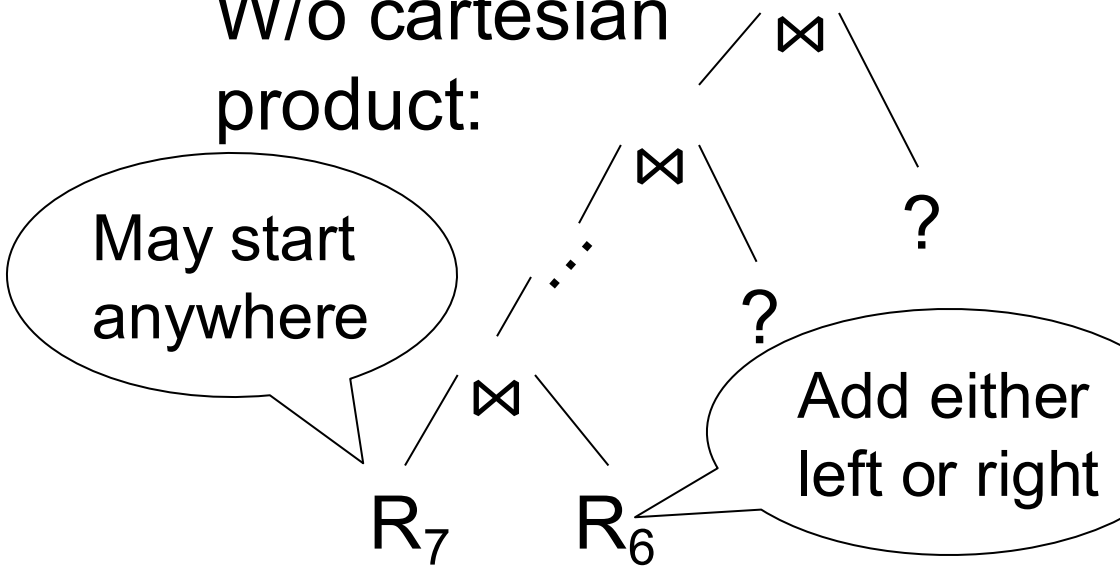
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:



n!

W/o cartesian product:

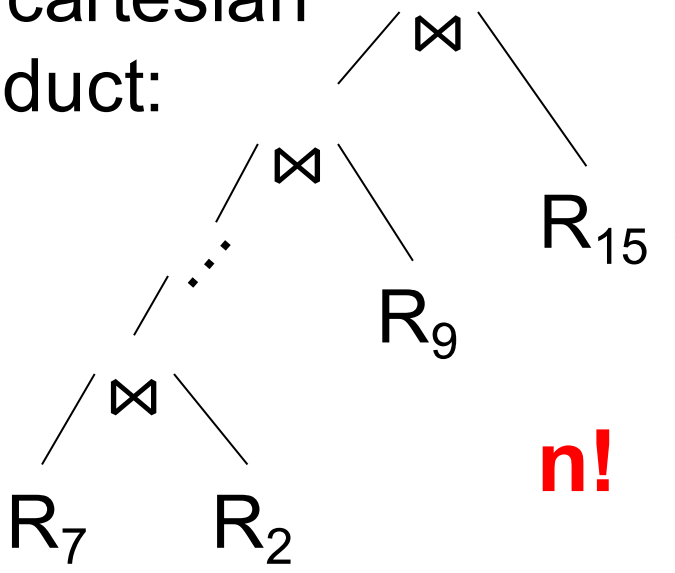


Number of Left-Deep Plans

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

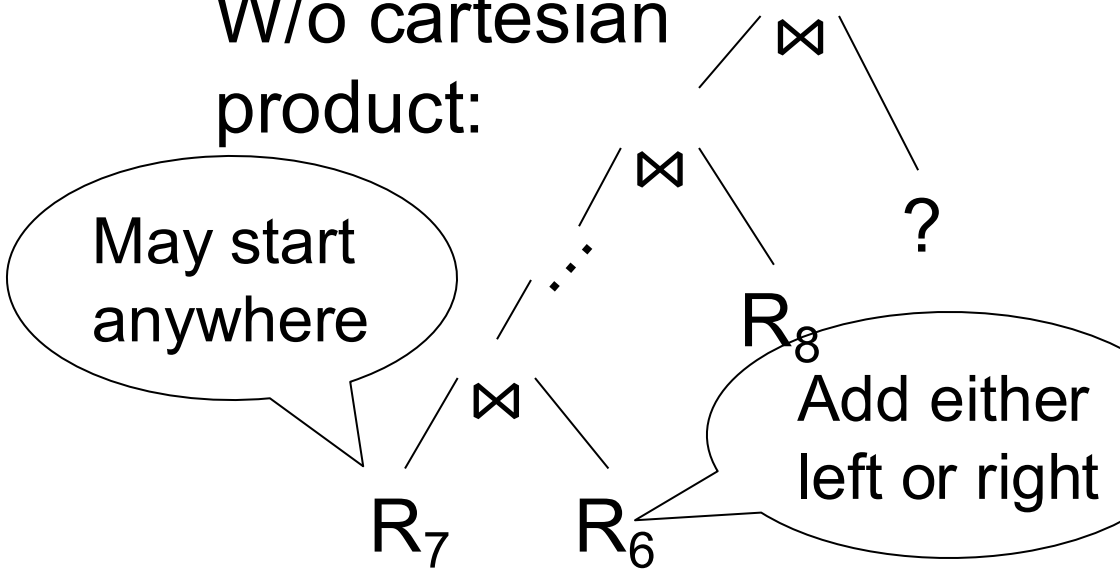
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:



n!

W/o cartesian product:

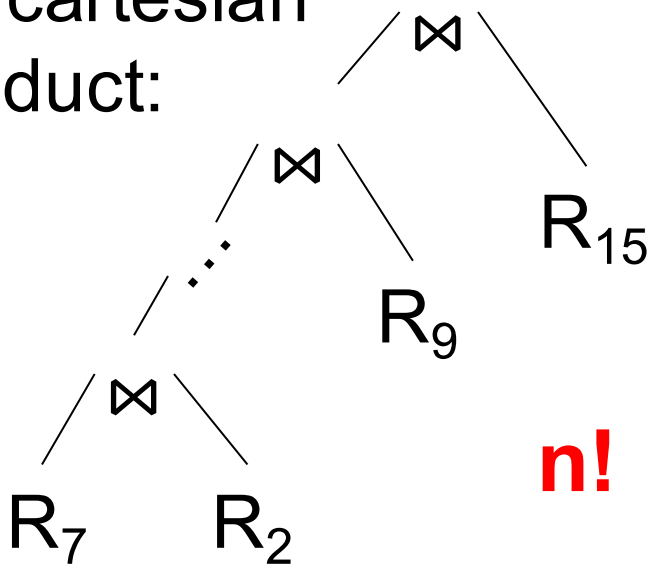


Number of Left-Deep Plans

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

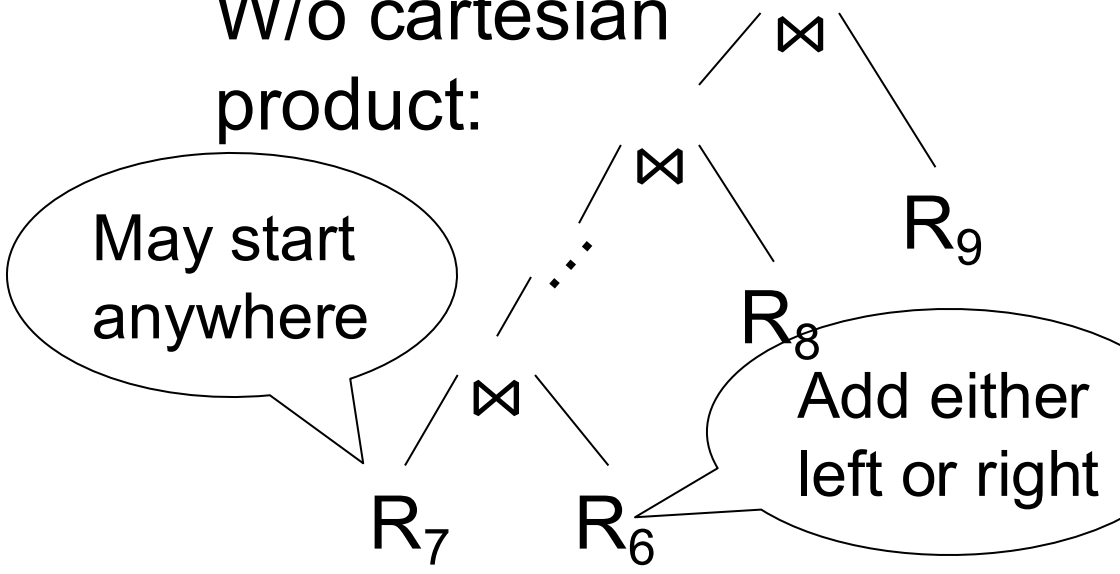
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:



n!

W/o cartesian product:

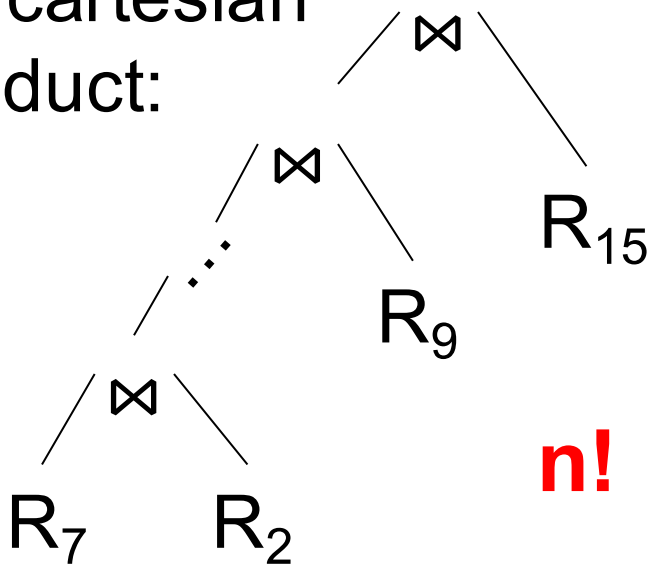


Number of Left-Deep Plans

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

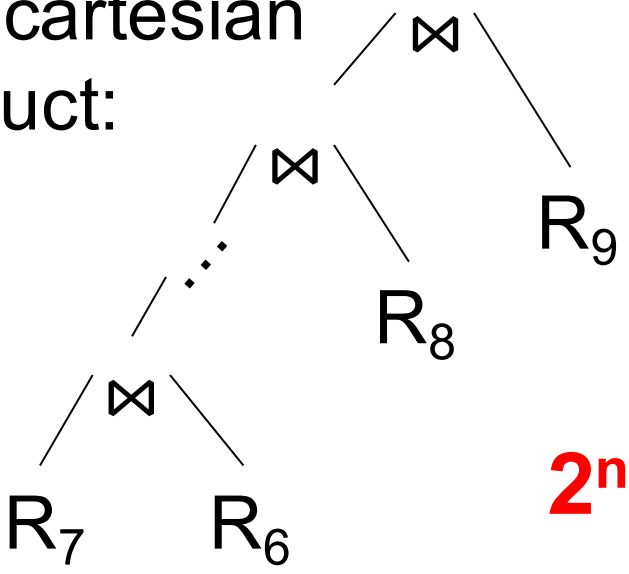
Query graph $R_1 \text{ —— } R_2 \text{ —— } R_3 \text{ —— } \ldots \text{ —— } R_n$

W/ cartesian product:



n!

W/o cartesian product:



2^{n-1}

Discussion

- n! can be much worse than 2^{n-1}.
- Other query shapes (star, clique, various) have similar behavior
- Hence: avoid cartesian products

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

Dynamic Programming: Summary

- All DBMS use this DP for join order
- Outer-, anti- joins add extra complexity
- Exponential time: DBMS have a limit (configurable) of # of tables for DP; beyond that, they use heuristics
- For other operators (group-by, aggregates, difference): rule-based

Two Types of Plan Enumeration Algorithms

- Dynamic programming

- Cascades optimizer

Cascades Optimizer

- Extends join ordering to full rewrite
- Supported by some of the most advanced DBMS today: SQL Server, Cockroach Lab; (unsure about DuckDB)

Cascades Optimizer

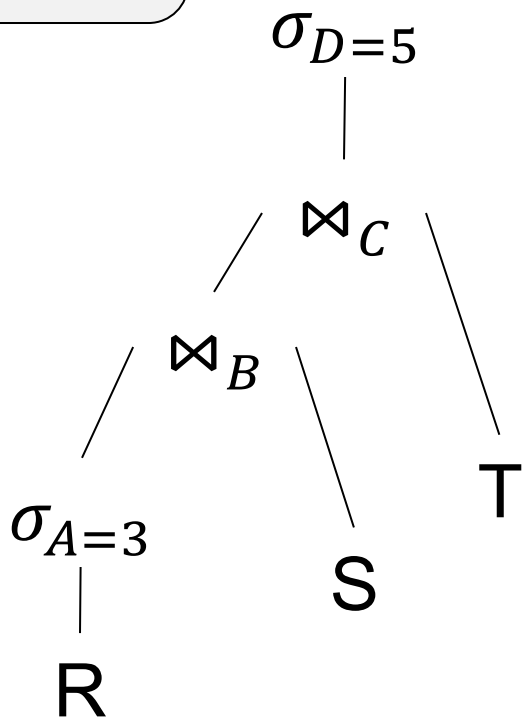
- Main idea: apply optimization rules:
  Q → Q'
- But keep both Q and Q'
- "Memo" data structure: reuses subplans

R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

Initialize Memo w/ one (naïve) plan



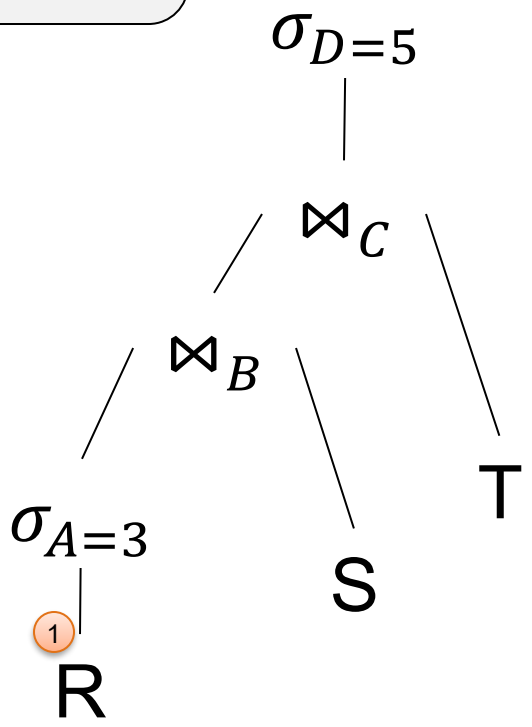
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1 Scan R

Initialize Memo w/ one (naïve) plan



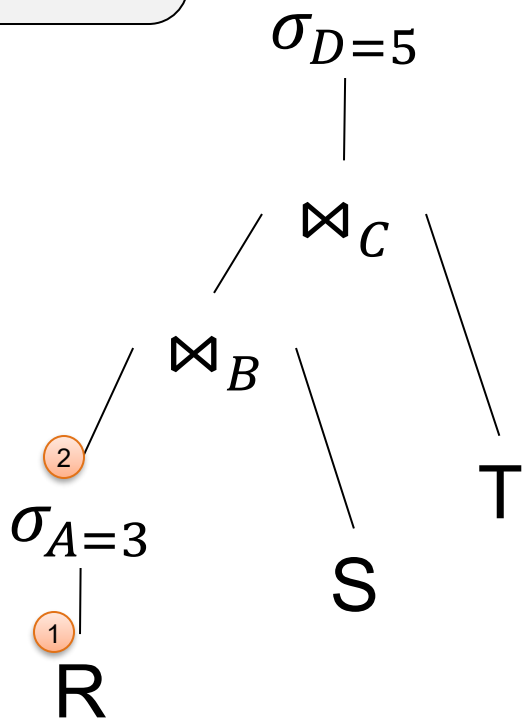
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1

Initialize Memo w/ one (naïve) plan



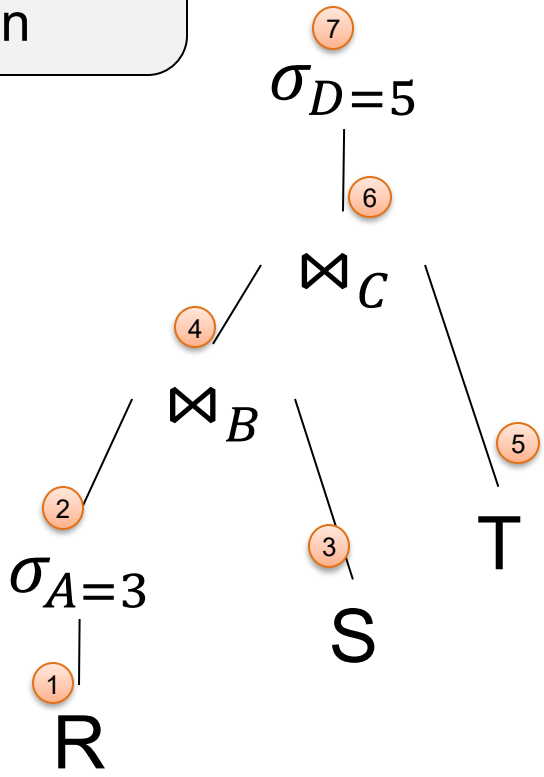
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Initialize Memo w/ one (naïve) plan

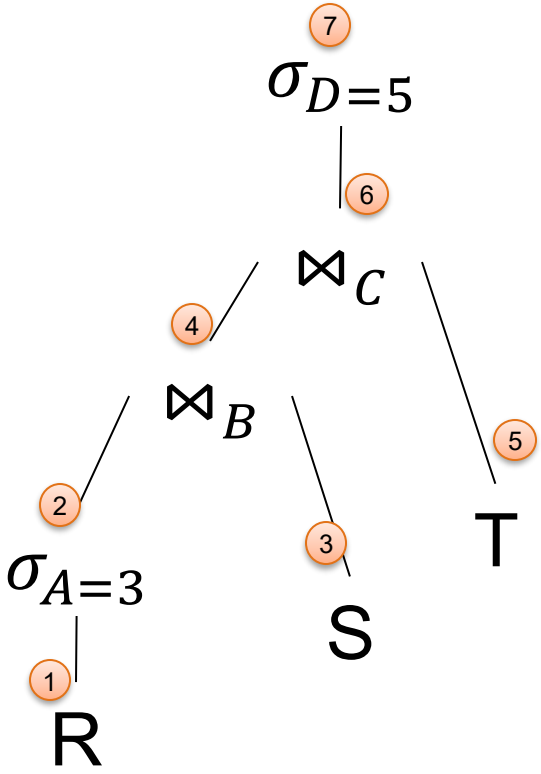


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6



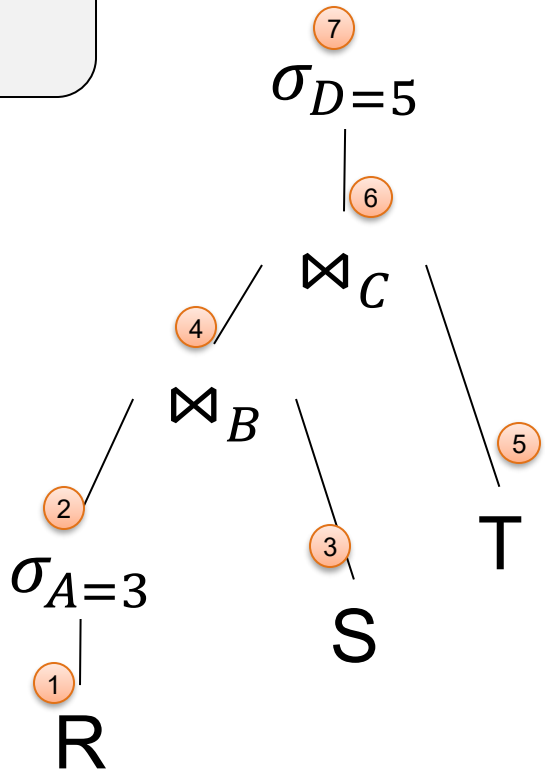
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Apply an optimization rule



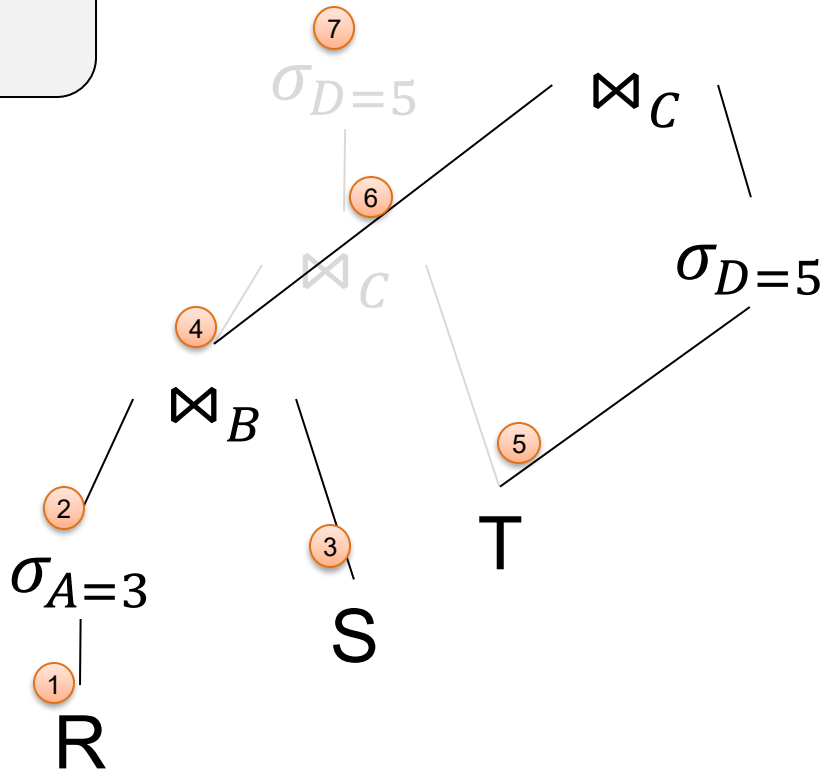
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Apply an optimization rule

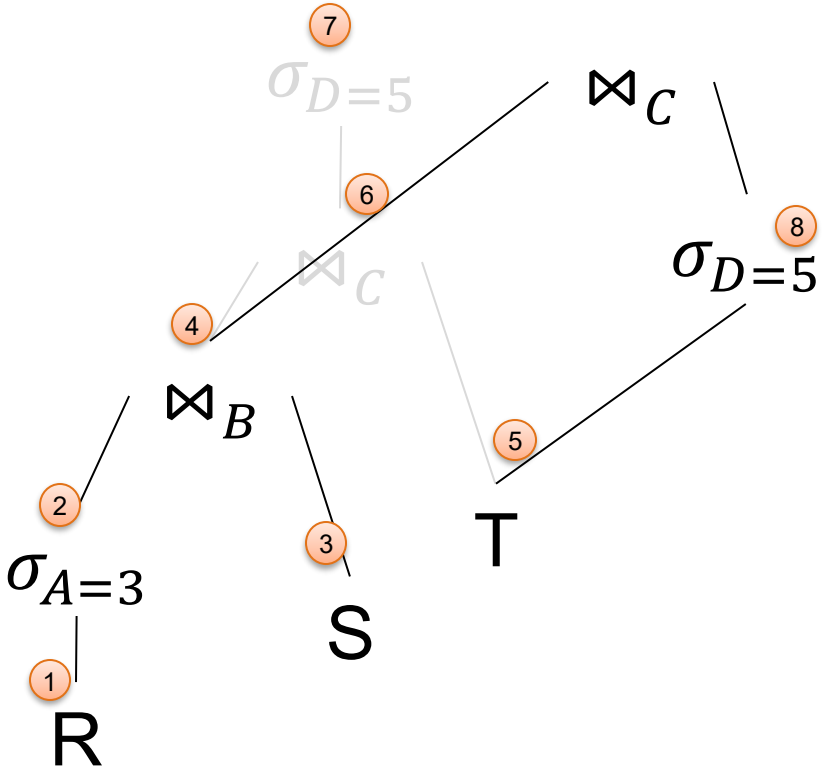


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6
8	Select[D=5] 5

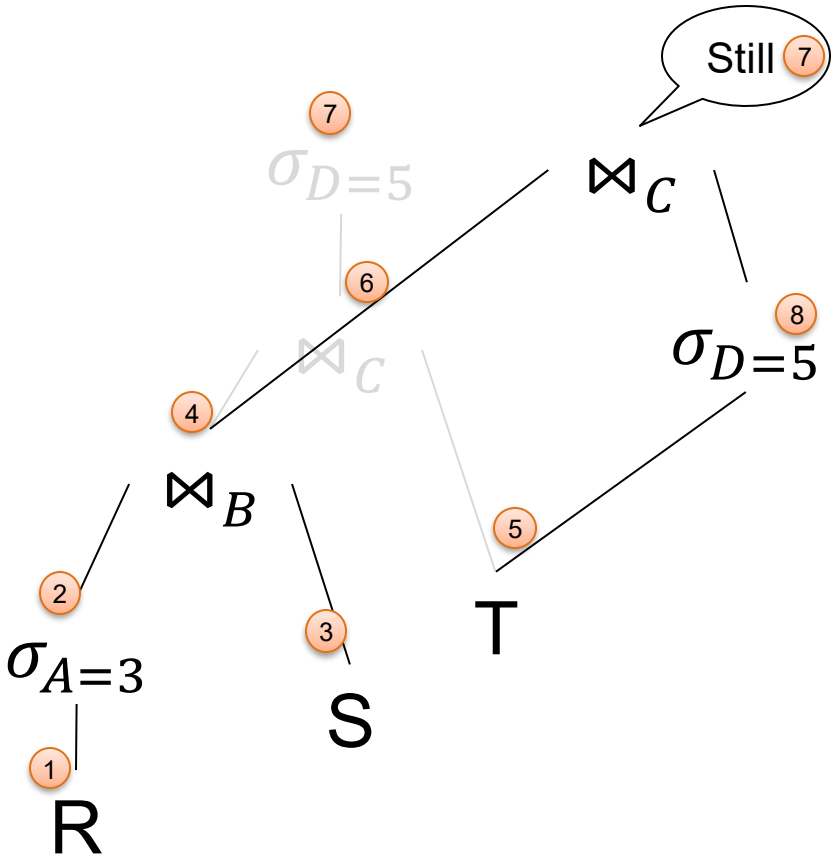


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

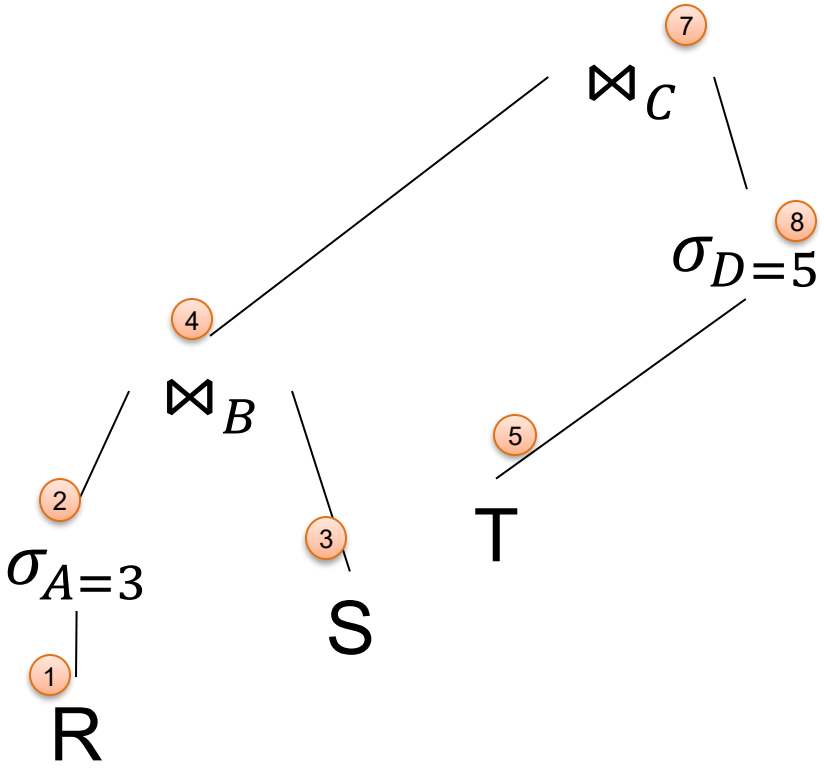


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

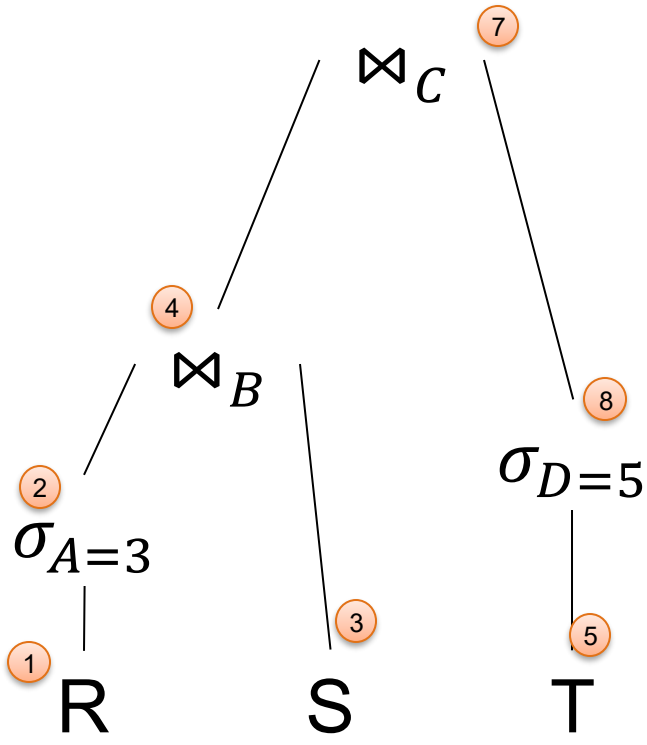


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	



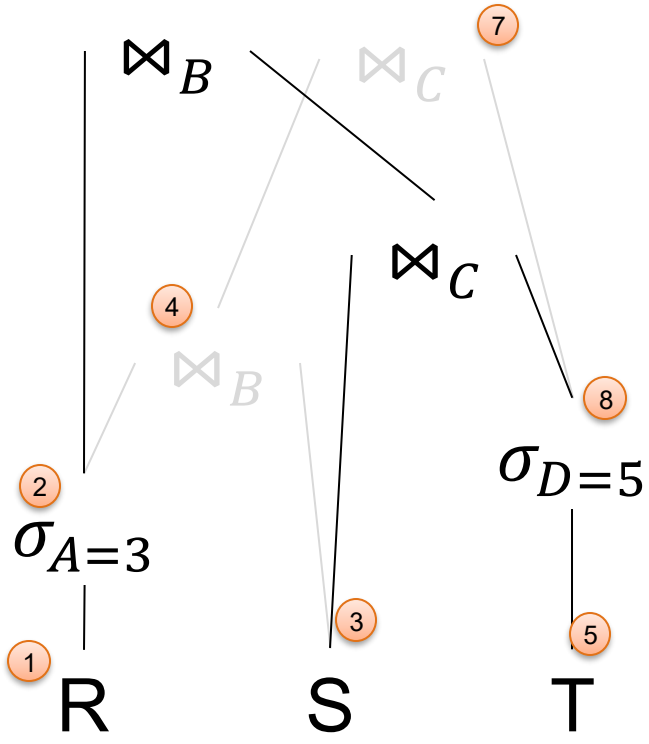
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

Apply another rule

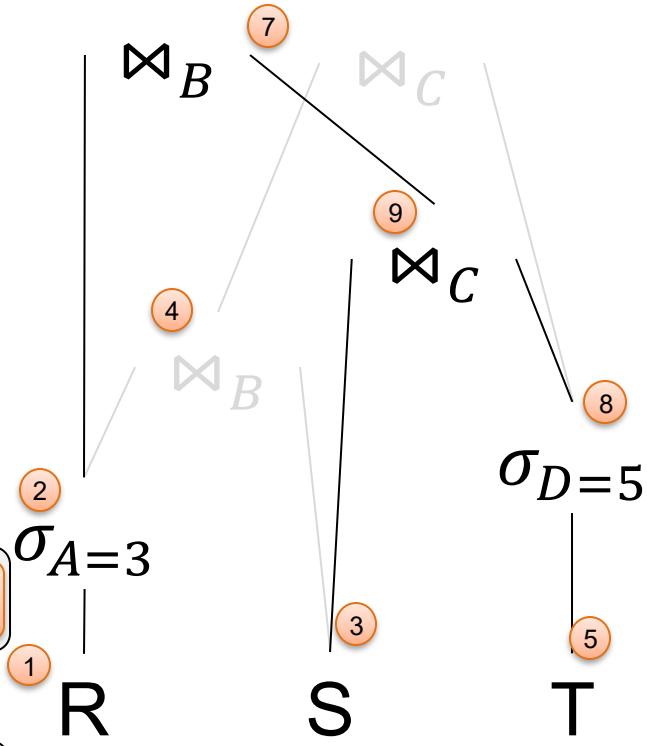


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R		
2	Select[A=3] 1		
3	Scan S		
4	Join[B=B] 2, 3		
5	Scan T		
6	Join[C=C] 4,5		
7	Select[D=5] 6	Join[C=C] 4,8	Join[B=B] 2,9
8	Select[D=5] 5		
9	Join[C=C] 3, 8		



Final Discussion

- Query optimizer = critical part of DBMS
- Search space + Size est + Algorithm
- Ideal: find “optimal” plan
- In practice: avoid “very bad plans”
- Successful because:
  - RA is a set-at-a-time language
  - RA is order-independent