

Where We Are

Three components:

- Cost/cardinality estimation
- Search space
 - Algebraic laws
 - Restricting the query plans ← ...finishing this
- Search algorithm ← then we'll discuss this

Two Types of Plan Enumeration Algorithms

- Dynamic programming (in class)
 - Based on System R (aka Selinger) style optimizer[1979]
 - Limited to joins: *join reordering algorithm*
 - Bottom-up
- Rule-based algorithm (will not discuss)
 - Database of rules (=algebraic laws)
 - Usually: dynamic programming
 - Usually: top-down

Two Types of Optimizers

- Rule-based (heuristic) optimizers:
 - Apply greedily rules that always improve plan
 - Typically: push selections down
 - Very limited: no longer used today
- Cost-based optimizers:
 - Use a cost model to estimate the cost of each plan
 - Select the “cheapest” plan
 - We focus on cost-based optimizers

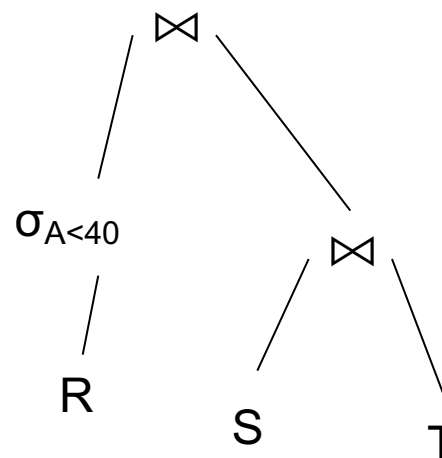
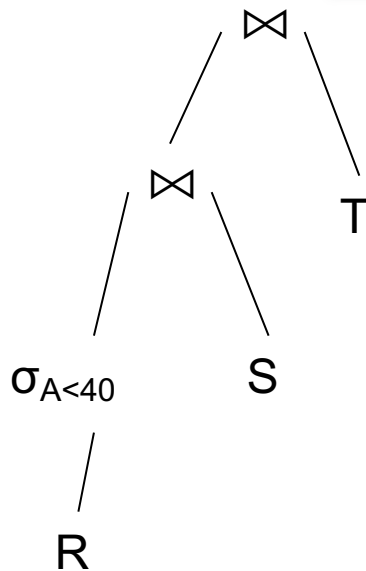
Three Approaches to Search Space

- Complete plans
- Bottom-up plans
- Top-down plans

Complete Plans

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<40
```



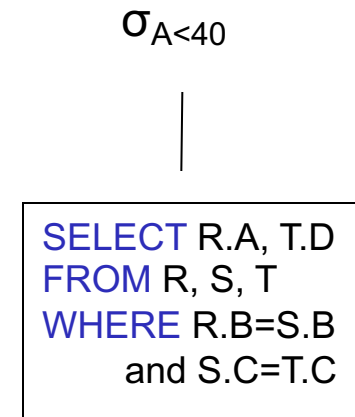
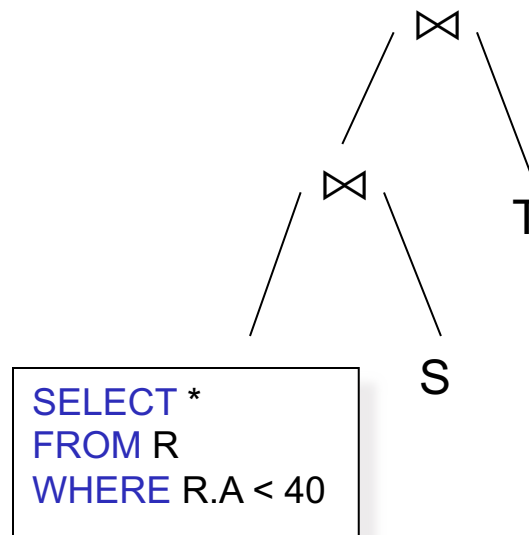
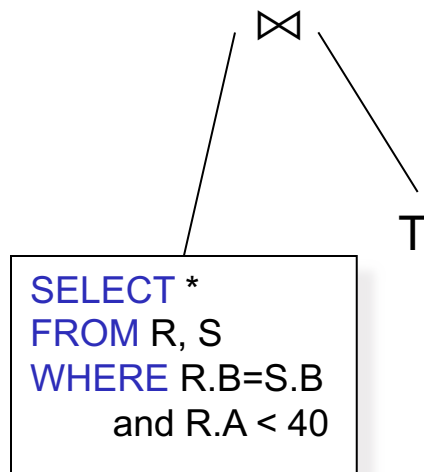
Why is this
search space
inefficient ?

Answer: No way to do early pruning

Top-down Partial Plans

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<40
```



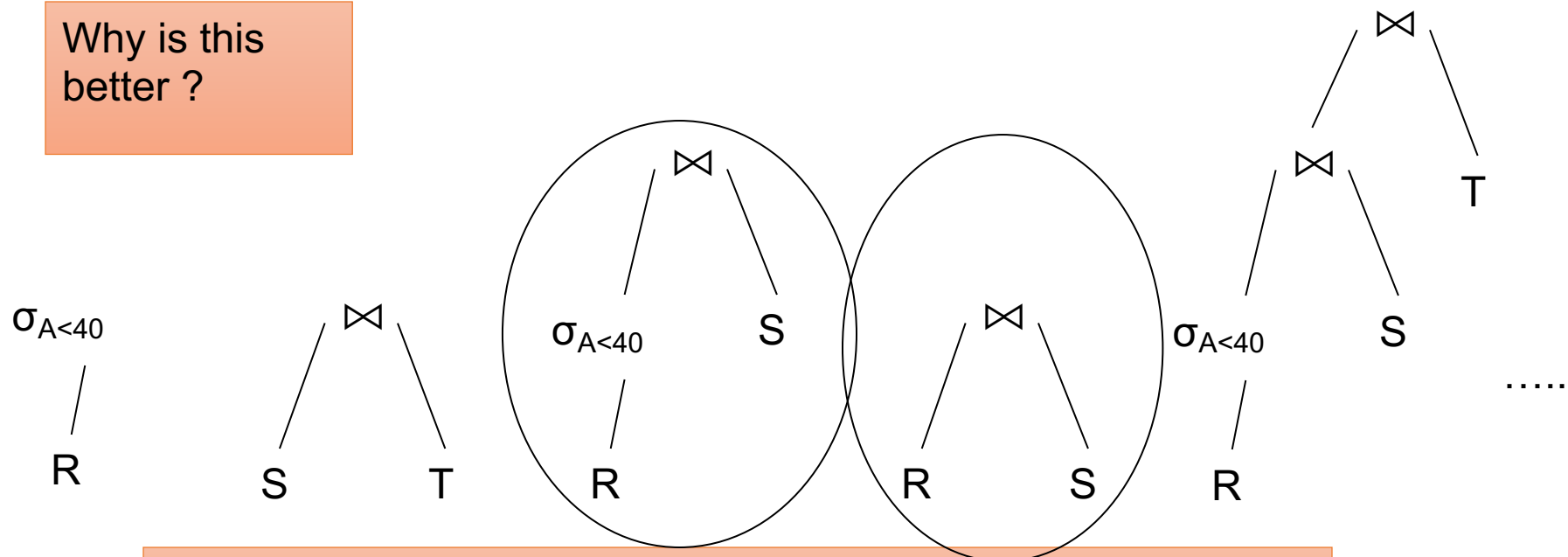
.....

Bottom-up Partial Plans

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<40
```

Why is this
better ?



We will prune bad plans for sub-expressions

Query Optimizer Overview

- **Input:** A logical query plan
- **Output:** A good physical query plan
- **Basic query optimization algorithm**
 - Enumerate alternative plans (logical and physical)
 - Compute estimated cost of each plan
 - Compute number of I/Os
 - Optionally take into account other resources
 - Choose plan with lowest cost
 - This is called cost-based optimization

Search Algorithm

- Dynamic programming (in class)
 - Based on System R (aka Selinger) style optimizer[1979]
 - Limited to joins: *join reordering algorithm*
 - Bottom-up
- Rule-based algorithm (will not discuss)
 - Database of rules (=algebraic laws)
 - Usually: dynamic programming
 - Usually: top-down

Dynamic Programming

Originally proposed in System R [1979]

- Only handles single block queries:

```
SELECT list  
FROM   R1, ..., Rn  
WHERE  cond1 AND cond2 AND . . . AND condk
```

- Some heuristics for search space enumeration:
 - Selections down
 - Projections up
 - Avoid cartesian products

Dynamic Programming

```
SELECT list  
FROM R1, ..., Rn  
WHERE cond1 AND cond2 AND . . . AND condk
```

- For each subquery $Q \subseteq \{R_1, \dots, R_n\}$ compute the following:
 - $T(Q)$ = the estimated size of Q
 - $\text{Plan}(Q)$ = a best plan for Q
 - $\text{Cost}(Q)$ = the estimated cost of that plan

Dynamic Programming

```
SELECT list  
FROM   R1, ..., Rn  
WHERE  cond1 AND cond2 AND . . . AND condk
```

- **Step 1:** For each $\{R_i\}$ do:
 - $T(\{R_i\}) = T(R_i)$
 - $\text{Plan}(\{R_i\}) = \text{access method for } R_i$
 - $\text{Cost}(\{R_i\}) = \text{cost of access method for } R_i$

- **Step 2:** For each $Q \subseteq \{R_1, \dots, R_n\}$ of size k do:
 - $T(Q)$ = use estimator
 - Consider all partitions $Q = Q' \cup Q''$
compute $\text{cost}(\text{Plan}(Q') \bowtie \text{Plan}(Q''))$
 - $\text{Cost}(Q)$ = the smallest such cost
 - $\text{Plan}(Q)$ = the corresponding plan

- **Note**
 - If we restrict to left-linear trees: Q'' = single relation
 - May want to avoid cartesian products

Dynamic Programming

```
SELECT list  
FROM   R1, ..., Rn  
WHERE  cond1 AND cond2 AND . . . AND condk
```

- **Step 3:** Return Plan($\{R_1, \dots, R_n\}$)

Example

```
SELECT *  
FROM   R, S, T, U  
WHERE  cond1 AND cond2 AND . . .
```

- $R \bowtie S \bowtie T \bowtie U$
- Assumptions:

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

- Every join selectivity is 0.001

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000		
S	5000		
T	3000		
U	1000		
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000		
S	5000		
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000		
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000		
U	1000		
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000		
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000	$R \bowtie S$ nested loop join	...
RT	6000		
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000	$R \bowtie S$ nested loop join	...
RT	6000	$R \bowtie T$ index join	...
RU	2000		
ST	15000		
SU	5000		
TU	3000		
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000	$R \bowtie S$ nested loop join	...
RT	6000	$R \bowtie T$ index join	...
RU	2000	$R \bowtie U$ index join	
ST	15000	$S \bowtie T$ hash join	
SU	5000	...	
TU	3000	...	
RST	30000		
RSU	10000		
RTU	6000		
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000	$R \bowtie S$ nested loop join	...
RT	6000	$R \bowtie T$ index join	...
RU	2000	$R \bowtie U$ index join	
ST	15000	$S \bowtie T$ hash join	
SU	5000	...	
TU	3000	...	
RST	30000	$(RT) \bowtie S$ hash join	...
RSU	10000	$(SU) \bowtie R$ merge join	
RTU	6000	...	
STU	15000		
RSTU	30000		

Example

$T(R) = 2000$
 $T(S) = 5000$
 $T(T) = 3000$
 $T(U) = 1000$

Assume
 $B(..) = T(..)/10$

Join selectivity
 is 0.001

Subquery	T	Plan	Cost
R	2000	Clustered index scan R.A	200
S	5000	Table scan	500
T	3000	Table scan	300
U	1000	Clustered index scan U.F	100
RS	10000	$R \bowtie S$ nested loop join	...
RT	6000	$R \bowtie T$ index join	...
RU	2000	$R \bowtie U$ index join	
ST	15000	$S \bowtie T$ hash join	
SU	5000	...	
TU	3000	...	
RST	30000	$(RT) \bowtie S$ hash join	...
RSU	10000	$(SU) \bowtie R$ merge join	
RTU	6000	...	
STU	15000		
RSTU	30000	$(RT) \bowtie (SU)$ hash join	...

Discussion

- For the subset $\{RS\}$, need to consider both $R \bowtie S$ and $S \bowtie R$
 - Because the cost may be different!
- When computing the cheapest plan for $(Q) \bowtie R$, we may consider new access methods for R , e.g. an index look-up that makes sense only in the context of the join

Discussion

```
SELECT list  
FROM R1, ..., Rn  
WHERE cond1 AND cond2 AND . . . AND condk
```

Given a query with n relations $R_1, \dots, R_n$

- How many entries do we have in the dynamic programming table?
- For each entry, how many alternative plans do we need to inspect?

Discussion

```
SELECT list  
FROM R1, ..., Rn  
WHERE cond1 AND cond2 AND . . . AND condk
```

Given a query with n relations $R_1, \dots, R_n$

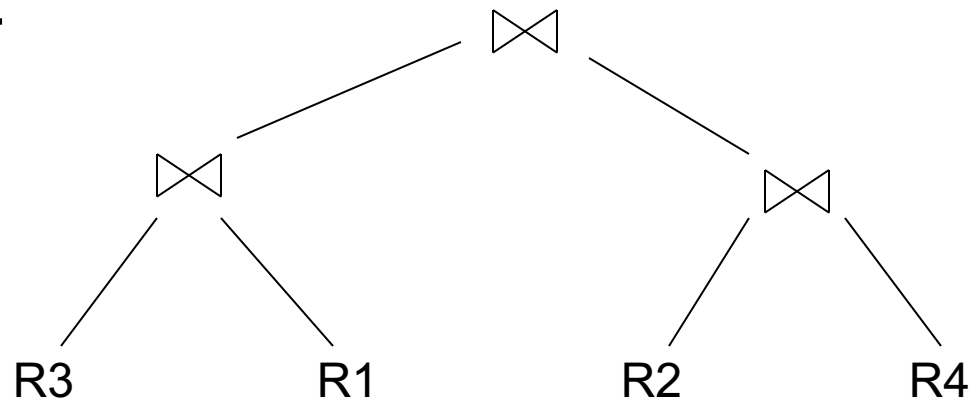
- How many entries do we have in the dynamic programming table?
 - A: $2^n - 1$
- For each entry, how many alternative plans do we need to inspect?
 - A: for each entry with k tables, examine $2^k - 2$ plans

Reducing the Search Space

- Left-linear trees
- No cartesian products

Join Trees

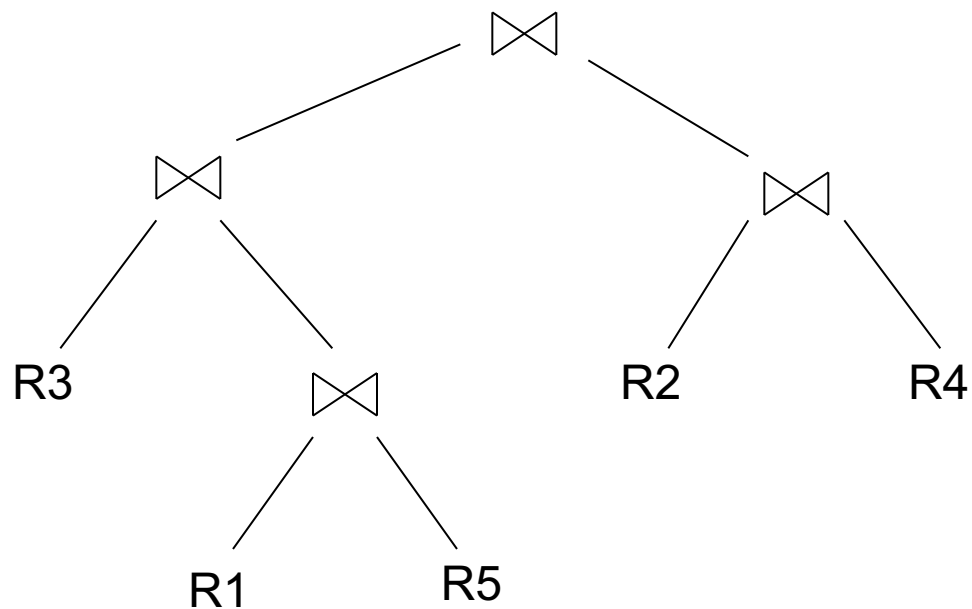
- $R1 \bowtie R2 \bowtie \dots \bowtie Rn$
- Join tree:



- A plan = a join tree
- A partial plan = a subtree of a join tree

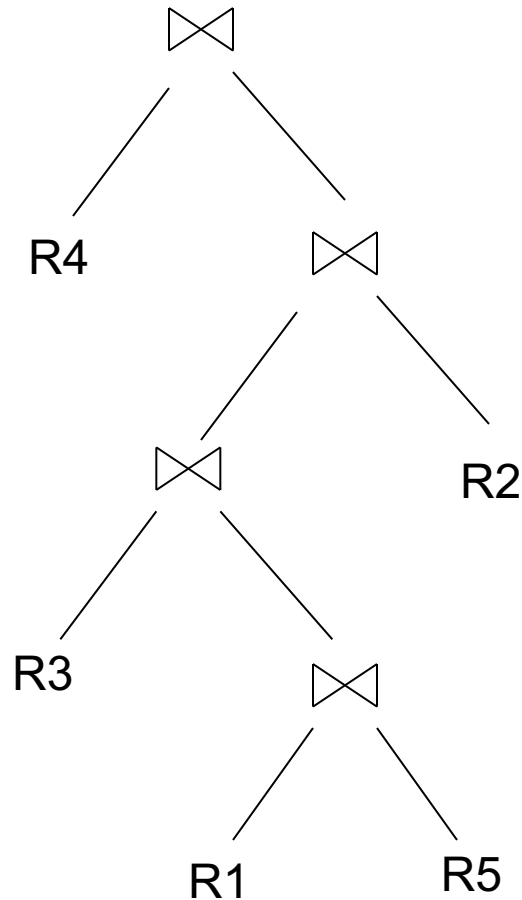
Types of Join Trees

- Bushy:



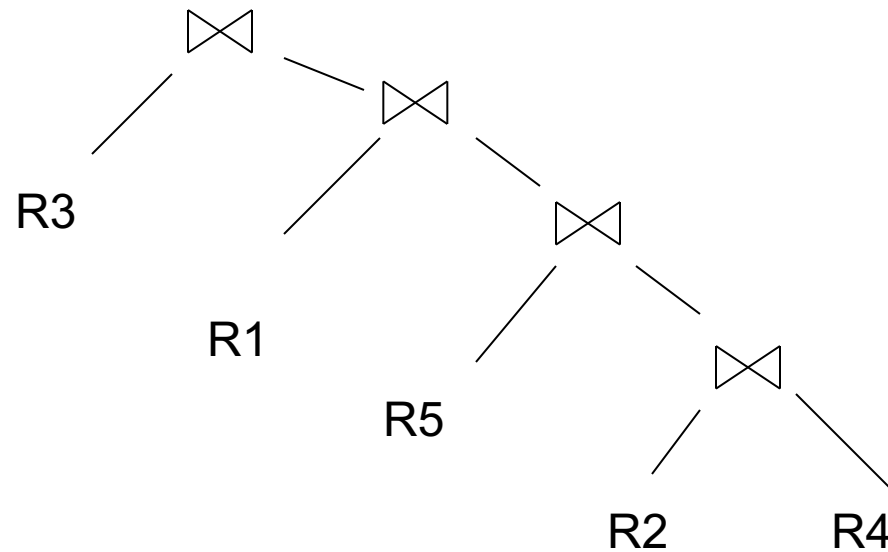
Types of Join Trees

- Linear :



Types of Join Trees

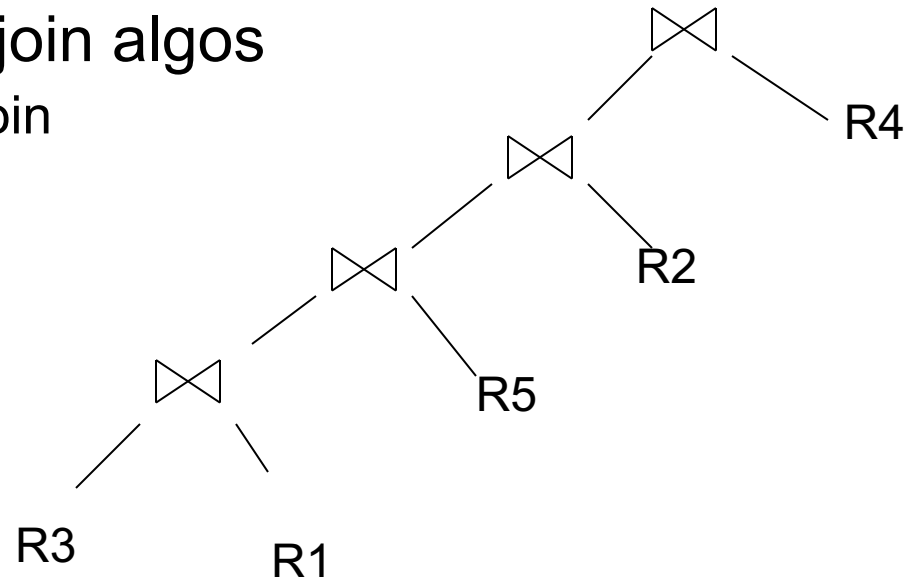
- Right deep:



Types of Join Trees

- Left deep:

- Work well with existing join algos
 - Nested-loop and hash-join
- Facilitate pipelining



- Dynamic programming can be used with all trees

No Cartesian Products

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D)$$

Plan: $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$

has a cartesian product.

Most query optimizers will not consider it

Avoid Cartesian Products

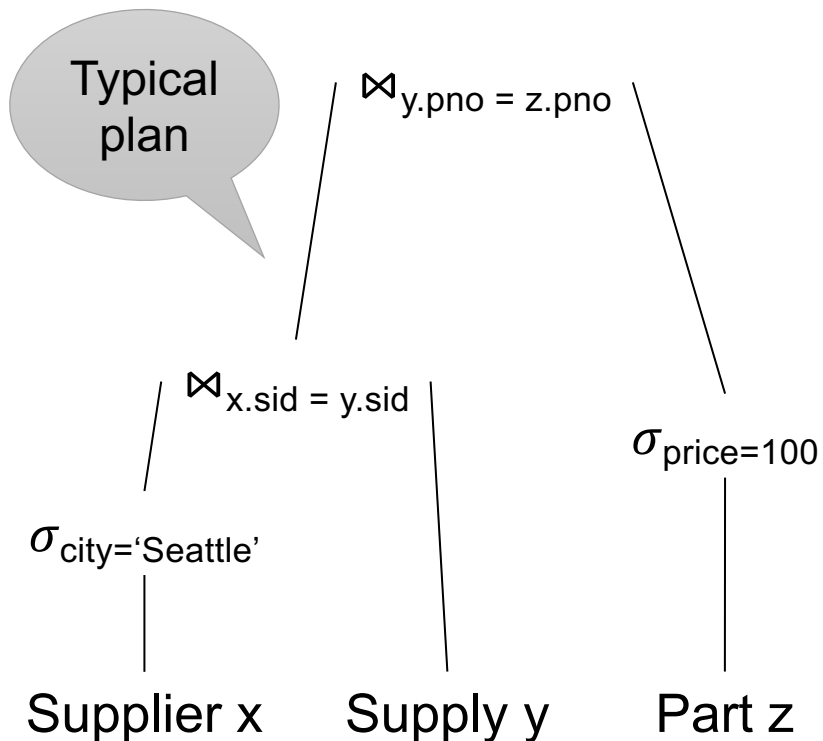
Supplier(sid, name, discount, city)
Supply(sid, pno)
Part(pno, pname, price)

```
select *  
from   Supplier x, Supply y, Part z  
where  x.sid = y.sid and y.pno = z.pno  
       and x.city='Seattle' and z.price=100;
```

Avoid Cartesian Products

Supplier(sid, name, discount, city)
Supply(sid, pno)
Part(pno, pname, price)

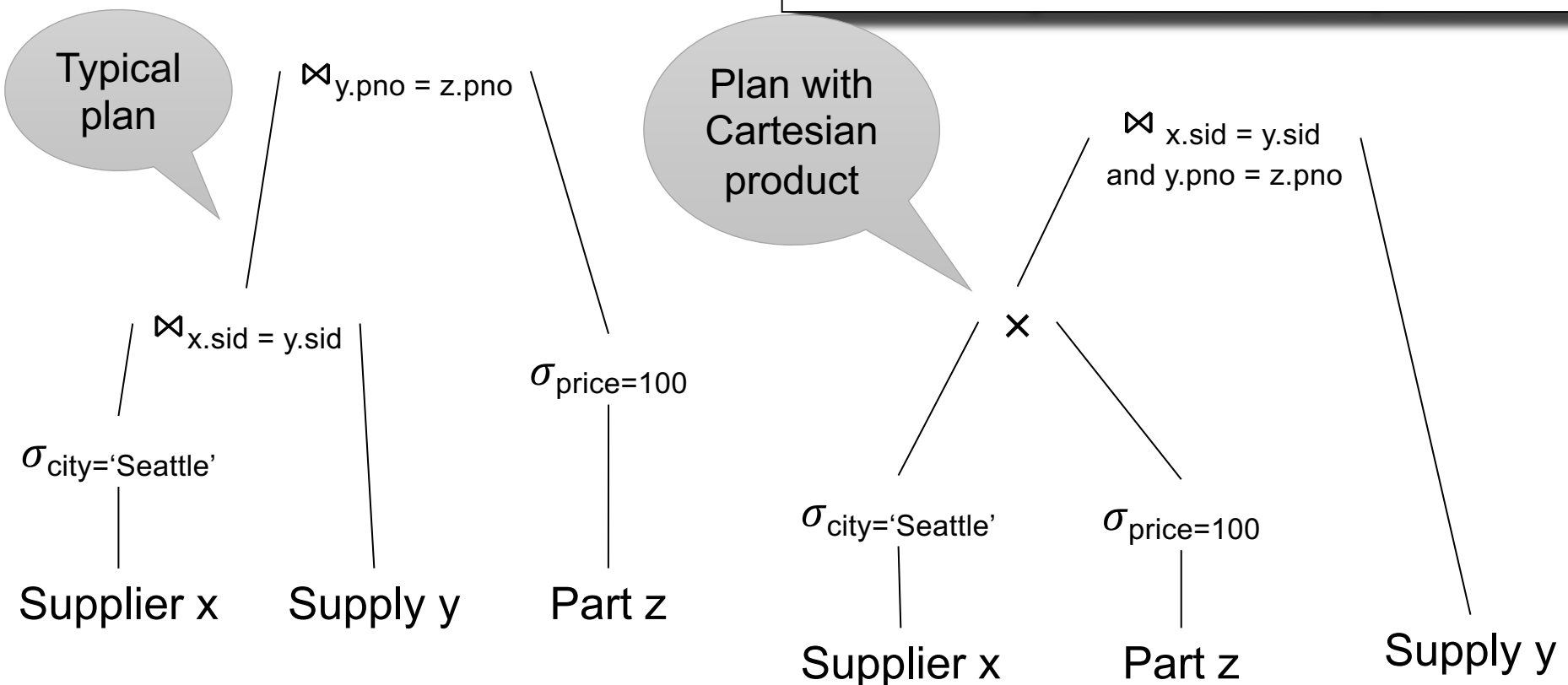
```
select *  
from   Supplier x, Supply y, Part z  
where  x.sid = y.sid and y.pno = z.pno  
       and x.city='Seattle' and z.price=100;
```



Avoid Cartesian Products

Supplier(sid, name, discount, city)
Supply(sid, pno)
Part(pno, pname, price)

```
select *  
from Supplier x, Supply y, Part z  
where x.sid = y.sid and y.pno = z.pno  
and x.city='Seattle' and z.price=100;
```



Most optimizers will not consider this plan