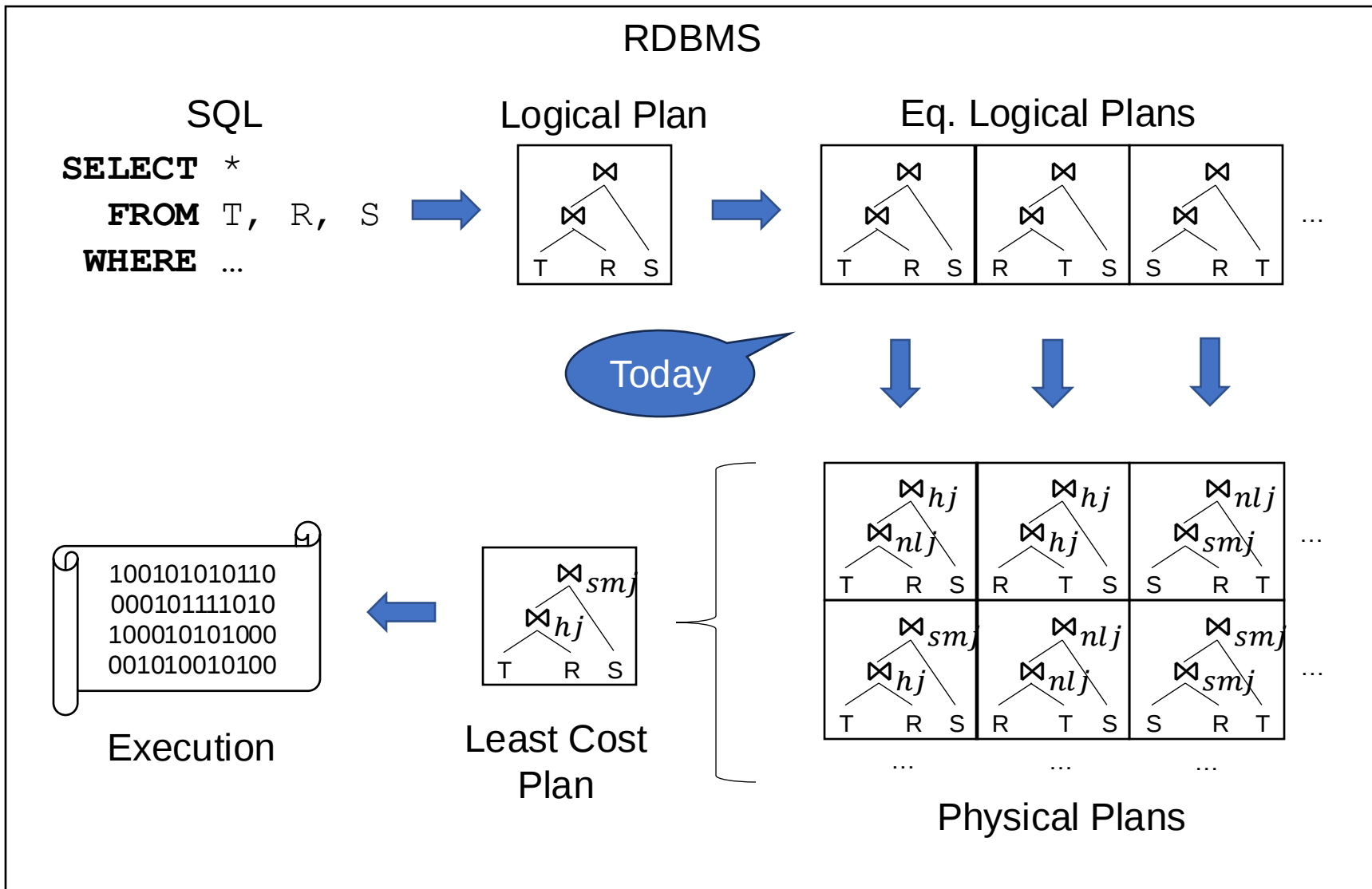


Announcements

- HW6: Part 2 due 11/27. More work than part 1
- Lecture on Wednesday, Nov. 27:
 - No in-person lecture
 - Recording only
- Monday, Dec. 9, Final Review session
 - CSE2 G10
 - 10:30 – 12:20 (we may finish earlier)

Query Optimization

Plan Enumeration



Query Optimization

- Space space



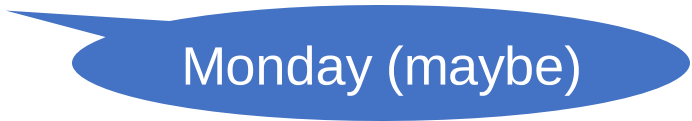
Today

- Cardinality and cost estimation



Monday

- Search algorithm



Monday (maybe)

Search Space

Search Space

- Optimizer explores alternative plans $P_1, P_2, \dots$
- The set of all plans is called the **search space**
- Two parts:
 - Rewrite rules
 - Access path selection

Rewrite Rules

Rewrite Rules

- A **rewrite rule** is an identity in relational algebra
- Similar to identities in linear algebra, e.g.

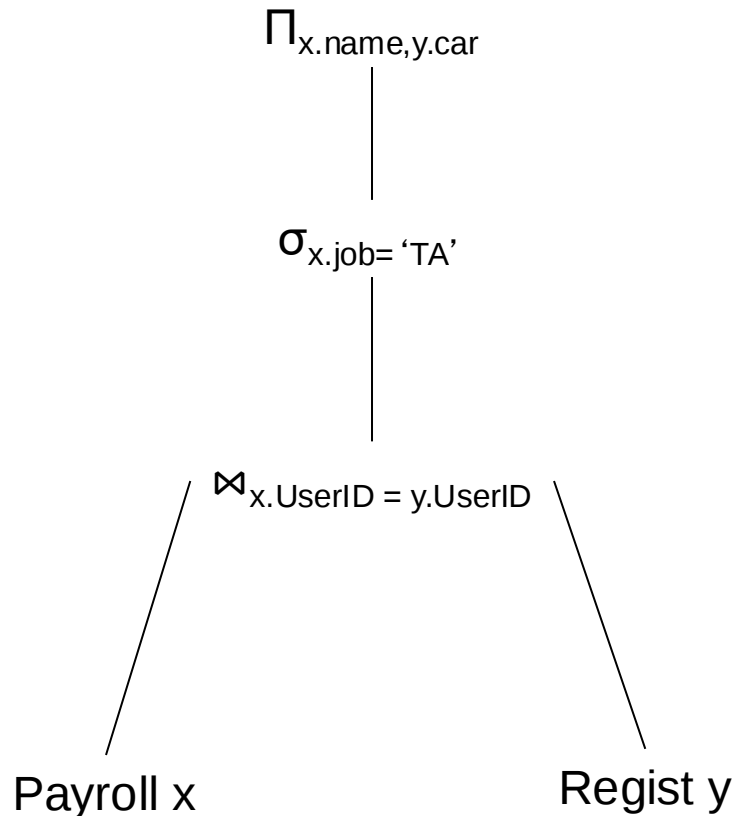
$a+b=b+a$	// commutativity
$a*(b+c)=a*b+a*c$	// distributivity
...	

- An optimizer applies a set of rewrite rules

Rewrite Rules: Predicate Pushdown

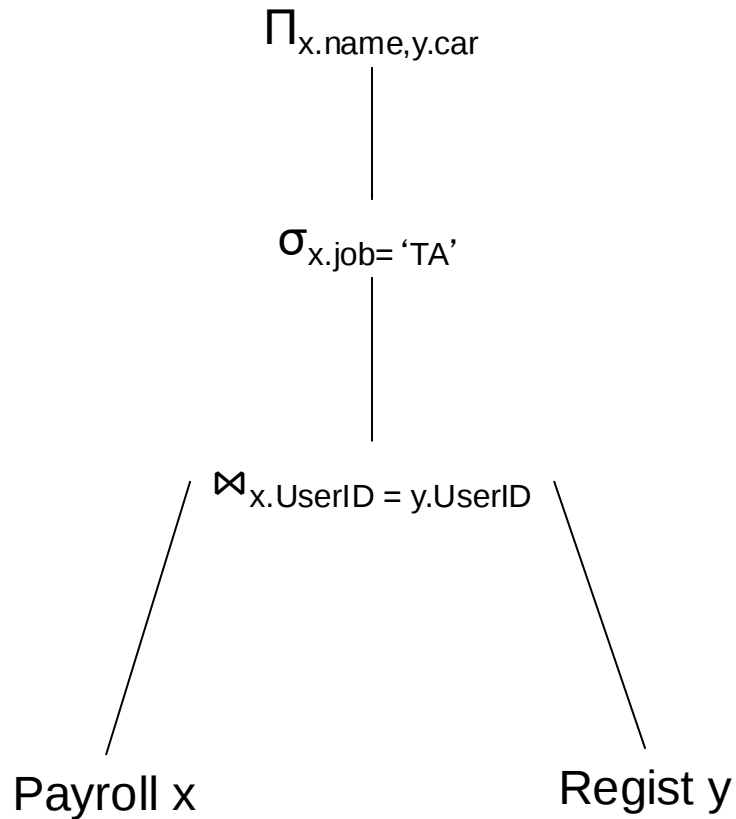
```
SELECT x.name, y.car  
FROM Payroll x, Regist y  
WHERE x.UserID = y.UserID  
      and x.job = 'TA' ;
```

Rewrite Rules: Predicate Pushdown

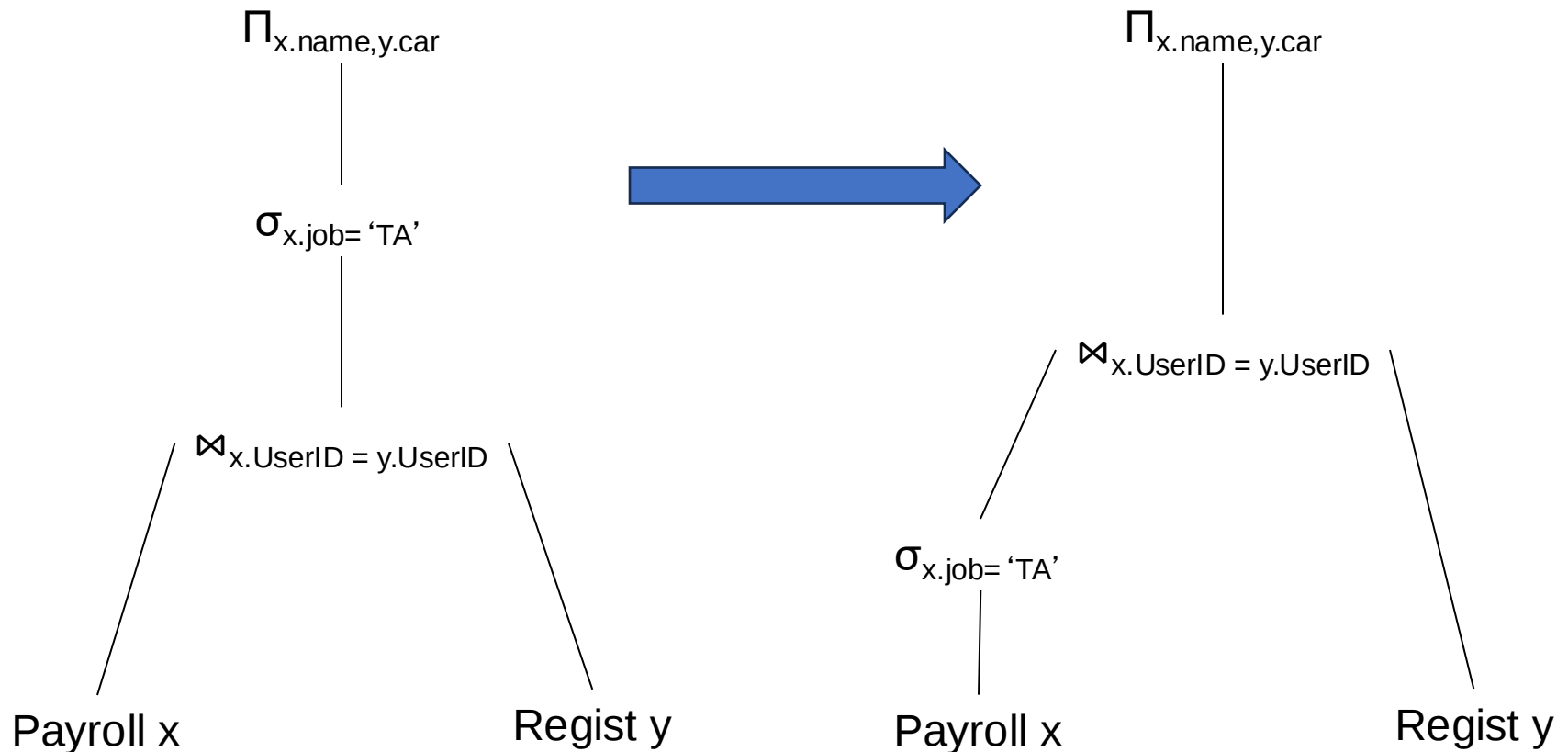


```
SELECT x.name, y.car  
FROM Payroll x, Regist y  
WHERE x.UserID = y.UserID  
      and x.job = 'TA';
```

Rewrite Rules: Predicate Pushdown

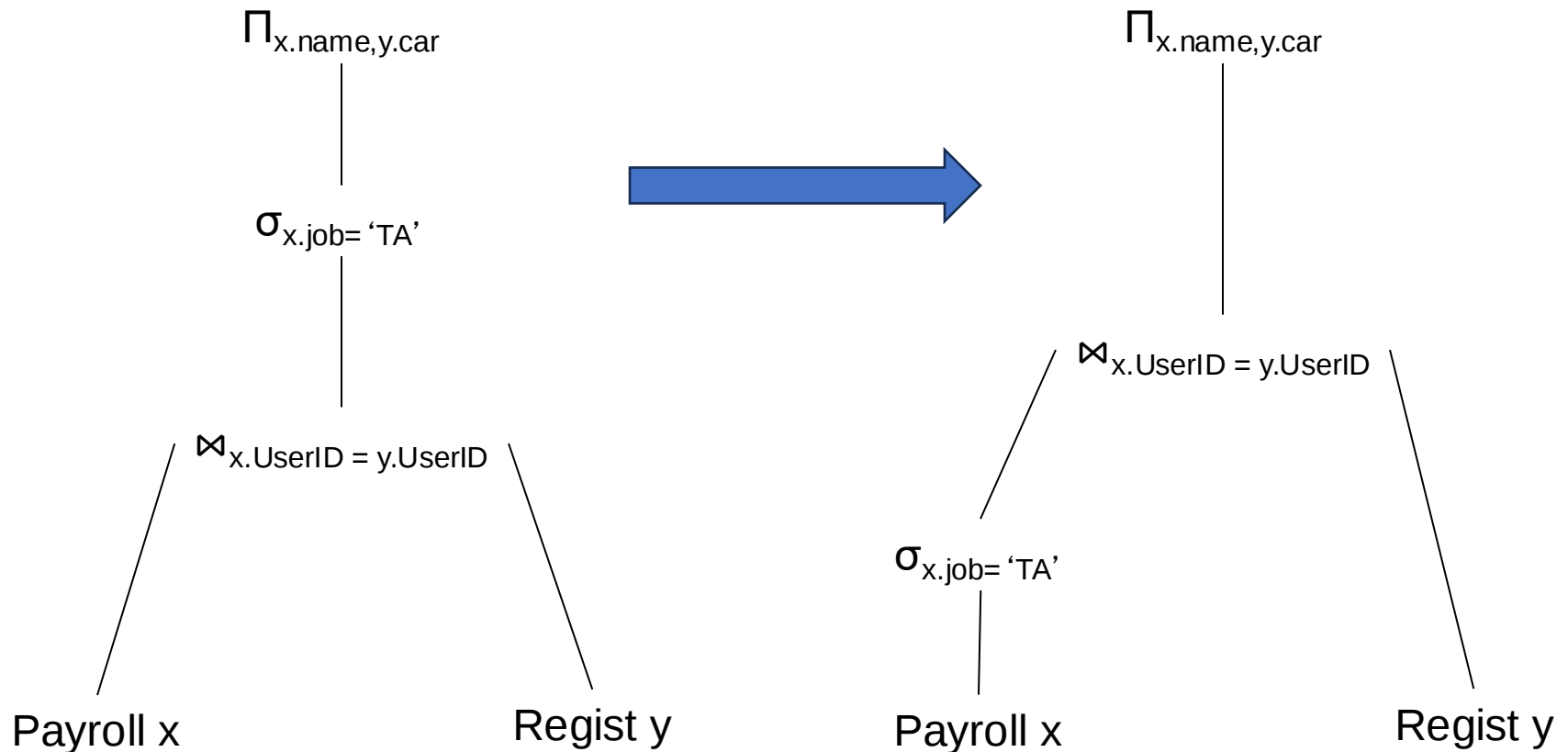


Rewrite Rules: Predicate Pushdown

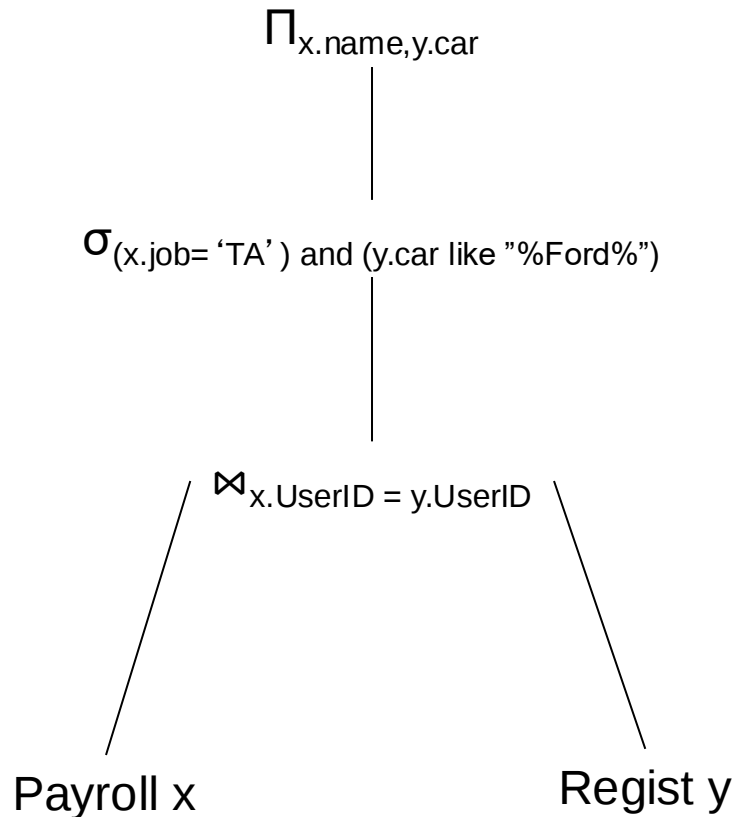


Rewrite Rules: Predicate Pushdown

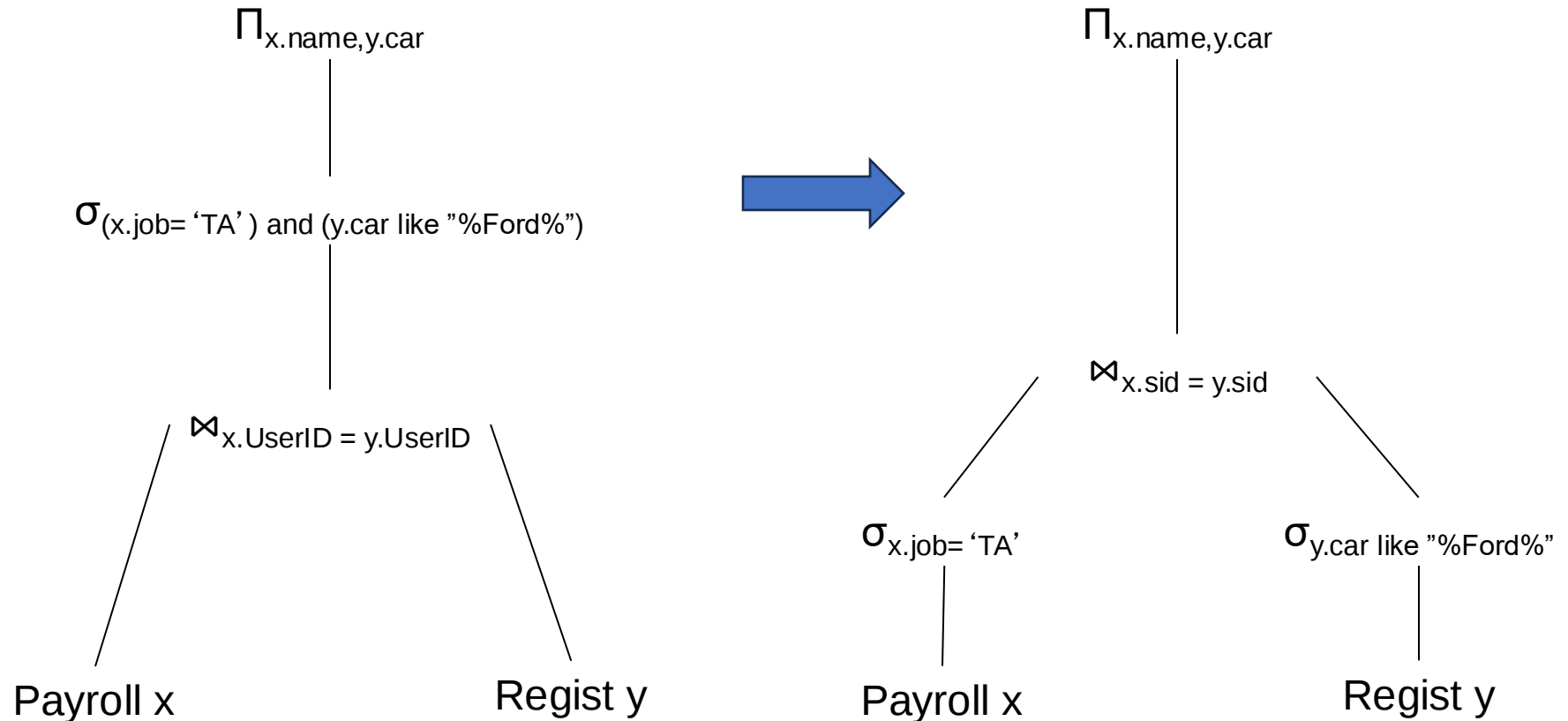
$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S \quad \text{when } C \text{ refers only to } R$$



Rewrite Rules: Predicate Pushdown

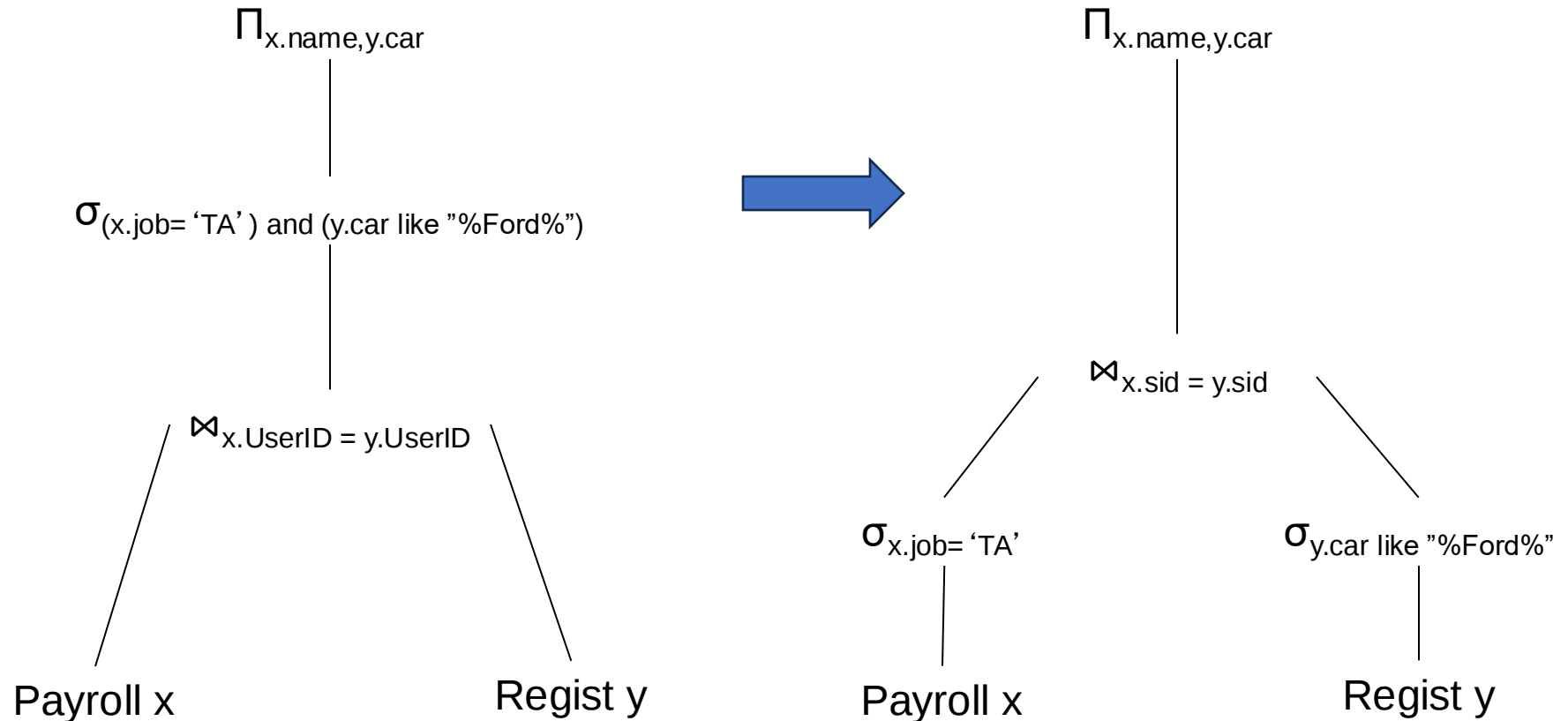


Rewrite Rules: Predicate Pushdown

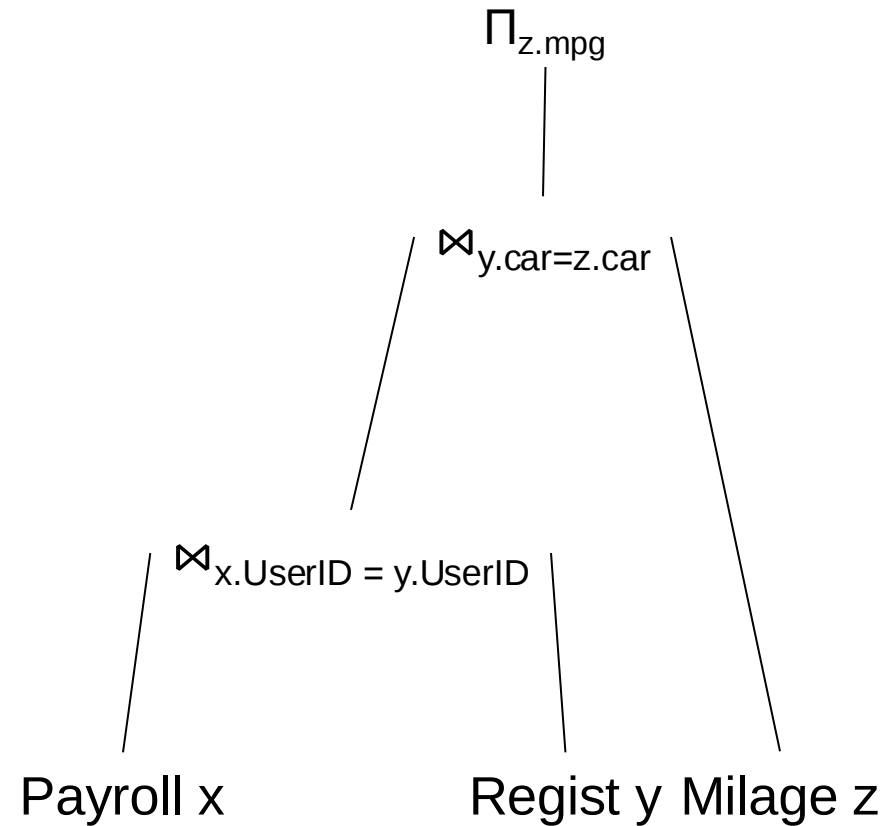


Rewrite Rules: Predicate Pushdown

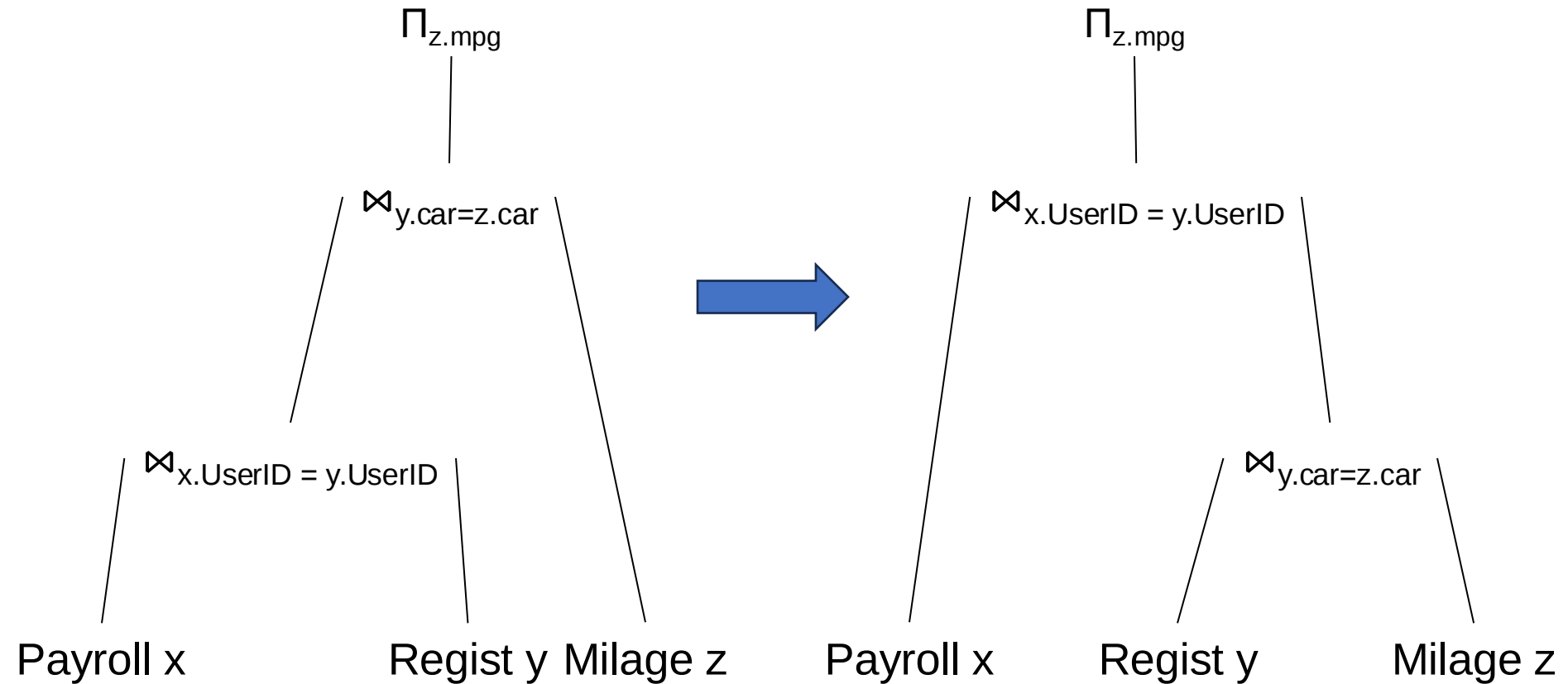
$$\sigma_{C1 \text{ and } C2}(R \bowtie S) = \sigma_{C1}(\sigma_{C2}(R \bowtie S))$$



Rewrite Rules: Join Associativity

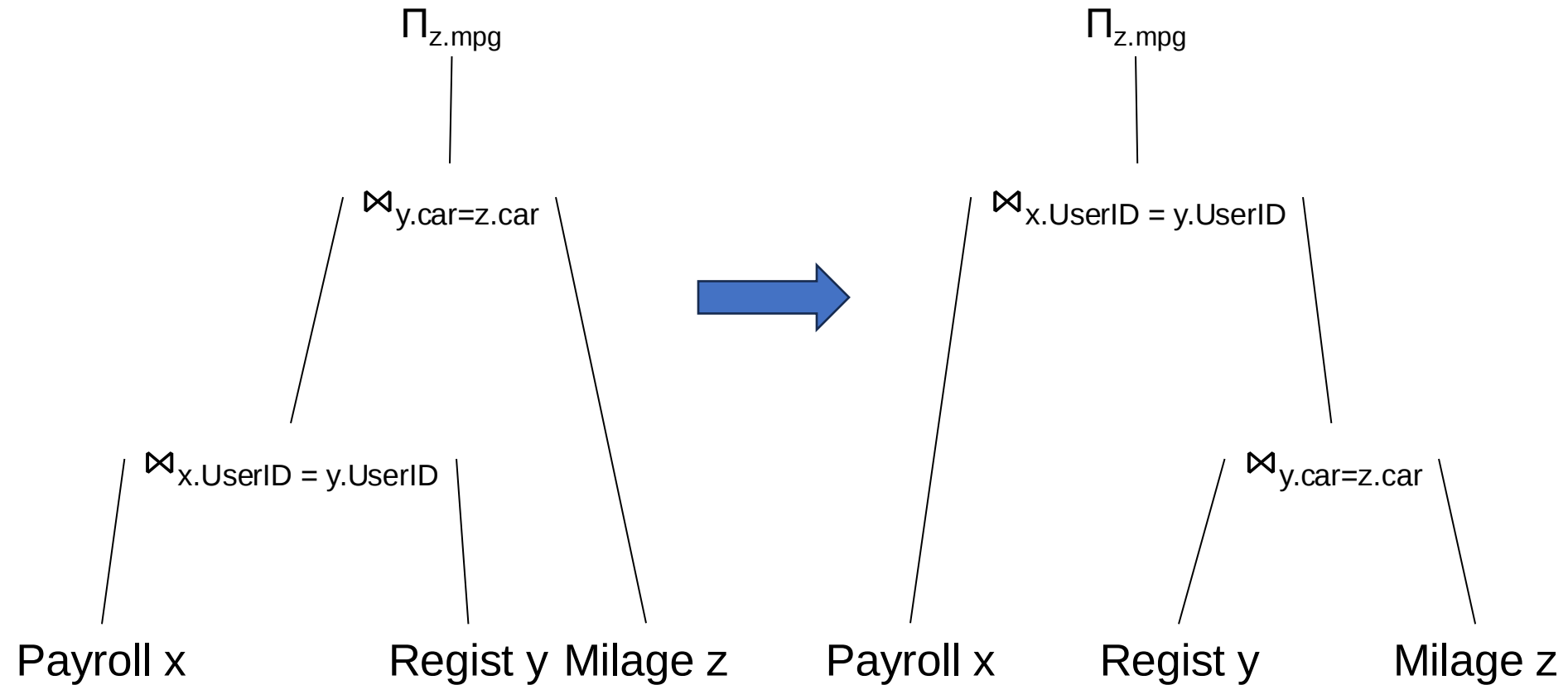


Rewrite Rules: Join Associativity



Rewrite Rules: Join Associativity

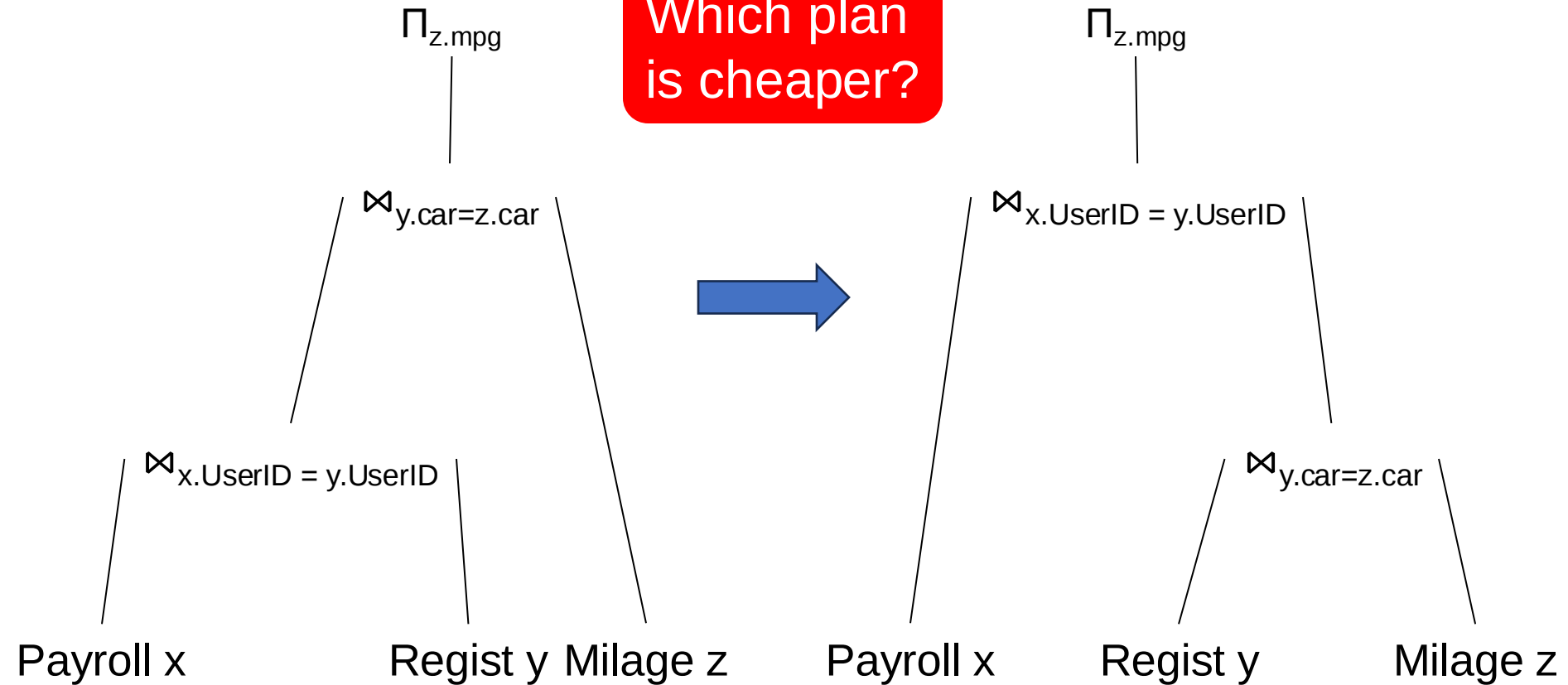
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$



Rewrite Rules: Join Associativity

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Which plan is cheaper?



Rewrite Rules: Join Associativity

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Which plan is cheaper?

$\Pi_{z.mpg}$

$\bowtie_{y.car=z.car}$

$\Pi_{z.mpg}$

$\bowtie_{x.UserID = y.UserID}$

Depends on

$$|\text{Payroll} \bowtie \text{Regist}| \leq |\text{Regist} \bowtie \text{Milage}|$$

$\bowtie_{x.UserID = y.UserID}$

$\bowtie_{y.car=z.car}$

Payroll x

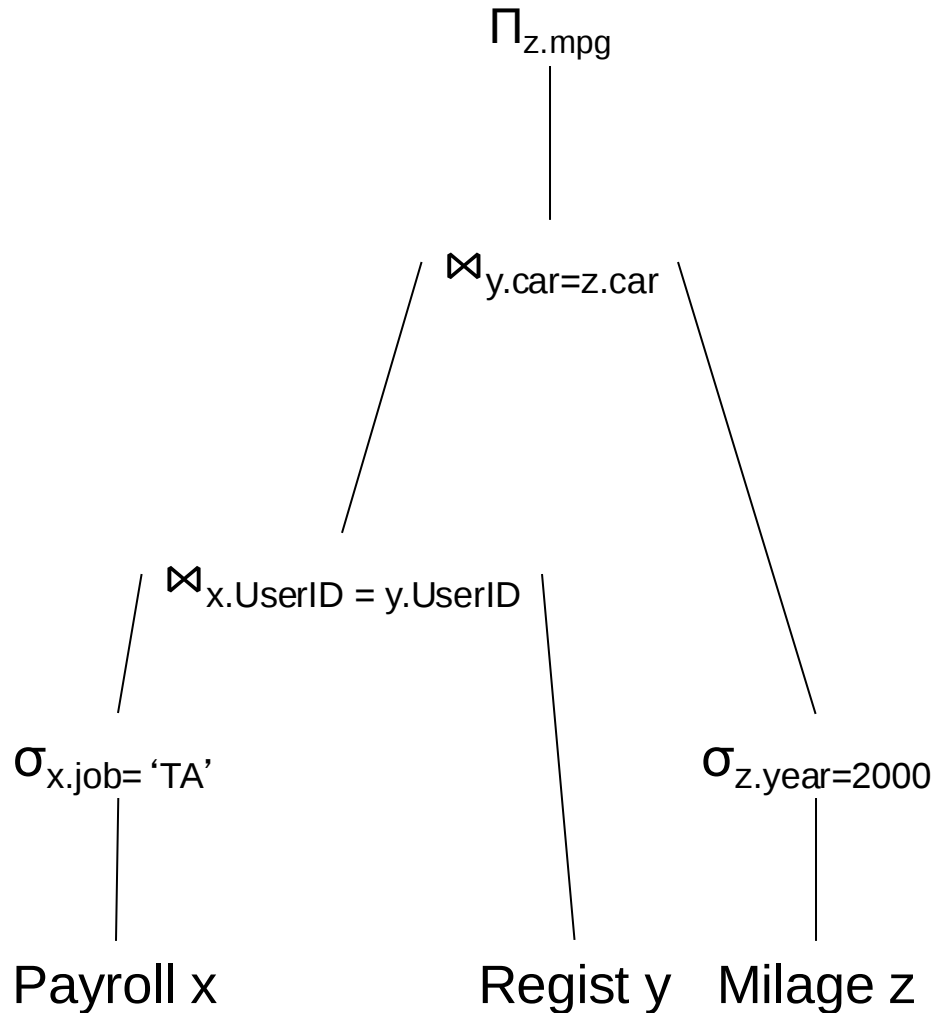
Regist y Milage z

Payroll x

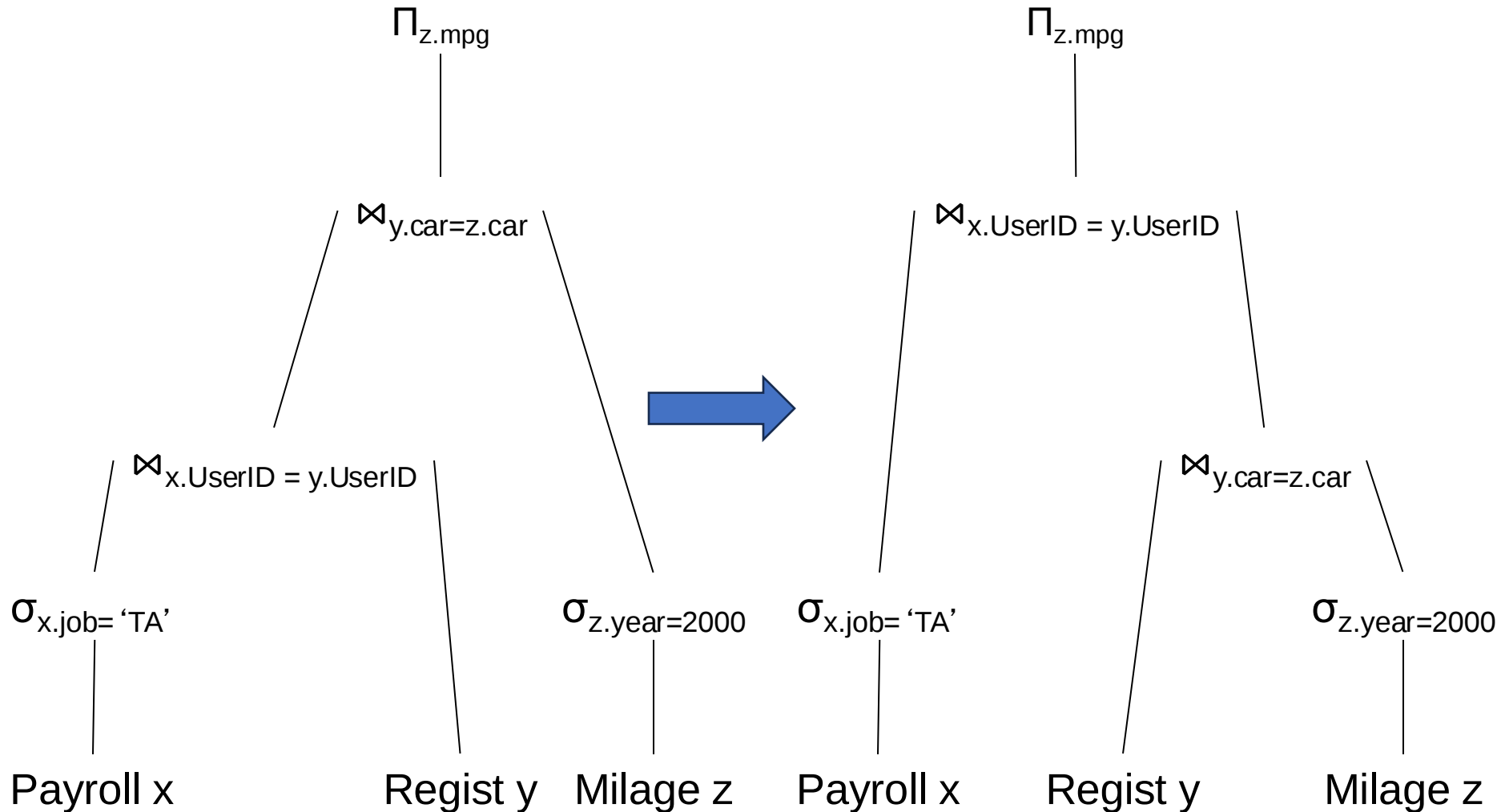
Regist y

Milage z

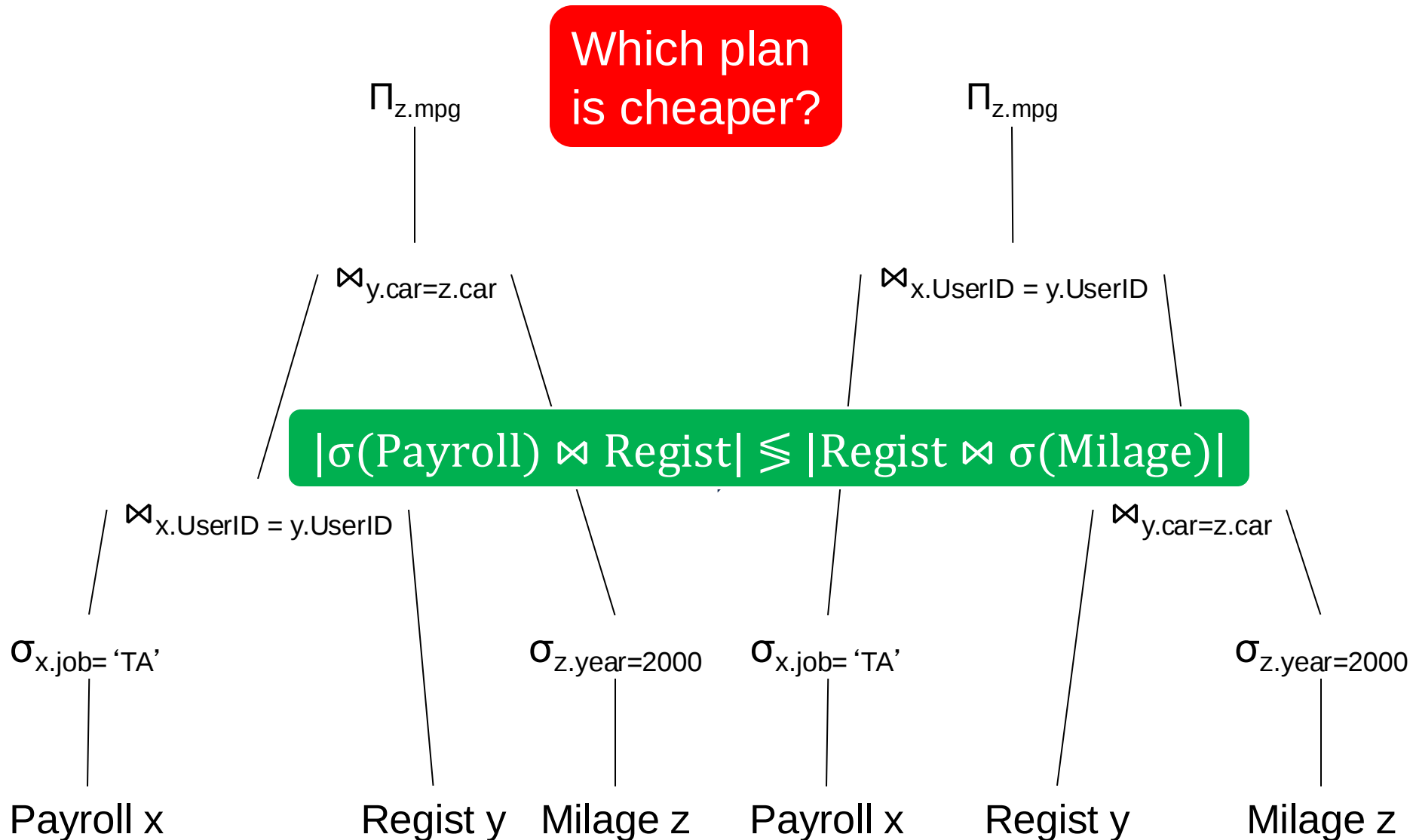
Rewrite Rules: Join Associativity



Rewrite Rules: Join Associativity



Rewrite Rules: Join Associativity



Practice

- Optimizers have a fixed list of rewrite rules
- E.g. SQL Server: about 500 rules
- Rules become complex and specialized

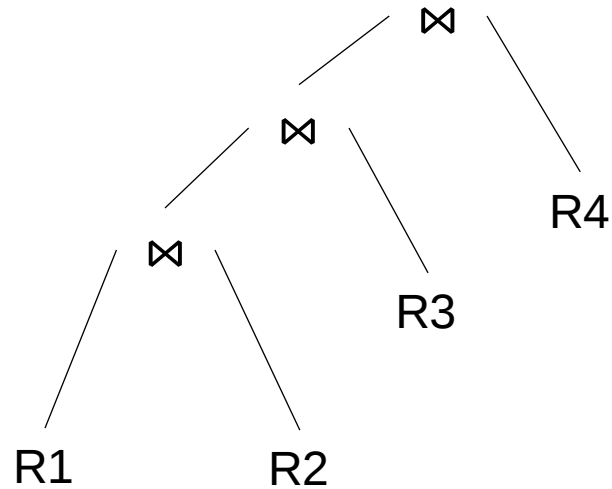
Search Space Challenges

Search space is huge!

Compromises:

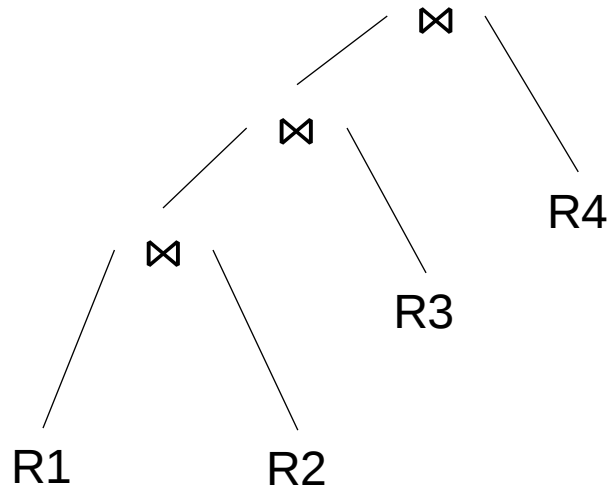
- Left-deep plans or right-deep plans
- Plans without cartesian products

Left-Deep-, Right-Deep-, Bushy-Plans

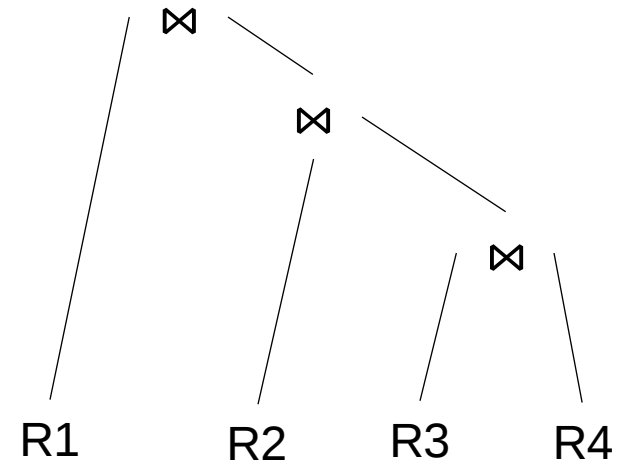


Left Deep

Left-Deep-, Right-Deep-, Bushy-Plans

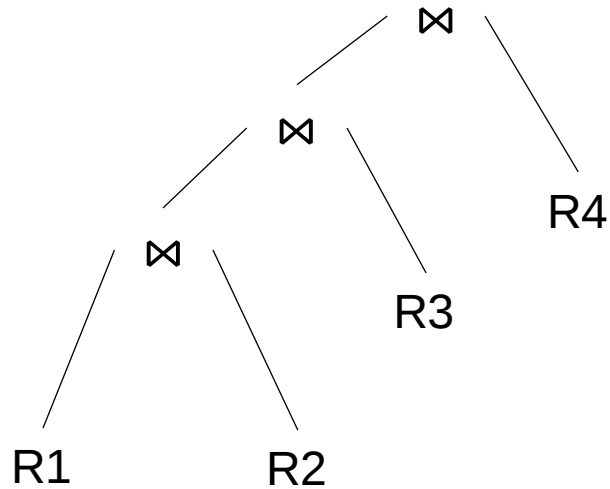


Left Deep

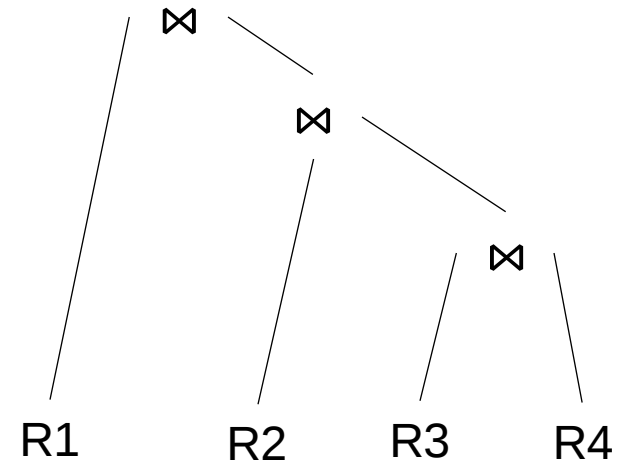


Right Deep

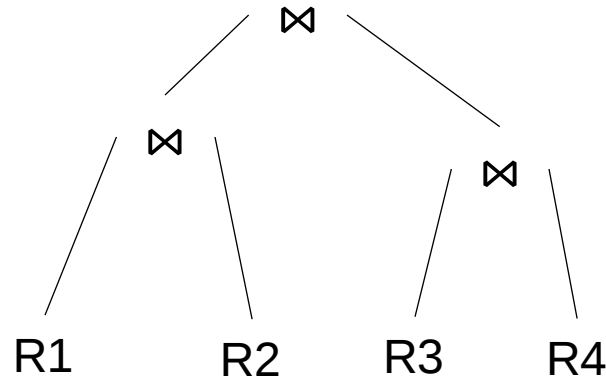
Left-Deep-, Right-Deep-, Bushy-Plans



Left Deep



Right Deep



Bushy

Left-Deep-, Right-Deep-, Bushy-Plans

Join n relations $R_1, R_2, \dots, R_n$

- How many left-deep plans are there?
- How many right-deep plans?
- How many bushy plans?

Left-Deep-, Right-Deep-, Bushy-Plans

Join n relations $R_1, R_2, \dots, R_n$

- How many left-deep plans are there? $n!$
- How many right-deep plans? $n!$
- How many bushy plans?

Left-Deep-, Right-Deep-, Bushy-Plans

Join n relations $R_1, R_2, \dots, R_n$

- How many left-deep plans are there? $n!$
- How many right-deep plans? $n!$
- How many bushy plans? $n! \times C_n$
where $C_{n-1} = \frac{1}{n} \binom{2n}{n-1}$ is the Catalan number

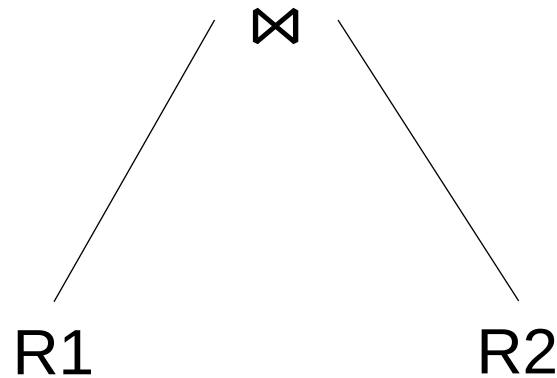
Left-Deep-, Right-Deep-, Bushy-Plans

Join n relations $R_1, R_2, \dots, R_n$

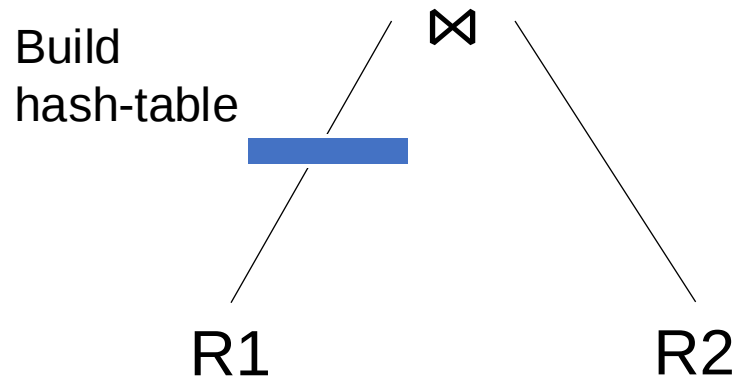
- How many left-deep plans are there? $n!$
- How many right-deep plans? $n!$
- How many bushy plans? $n! \times C_n$
where $C_{n-1} = \frac{1}{n} \binom{2n}{n-1}$ is the Catalan number

Many DBMS use only left- or right-deep plans

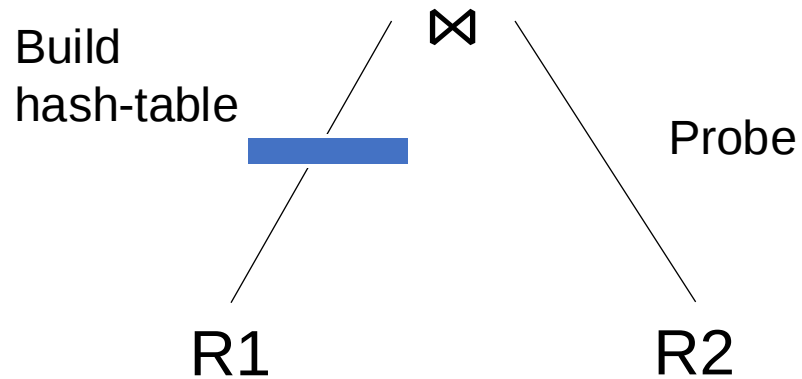
Hash-Join



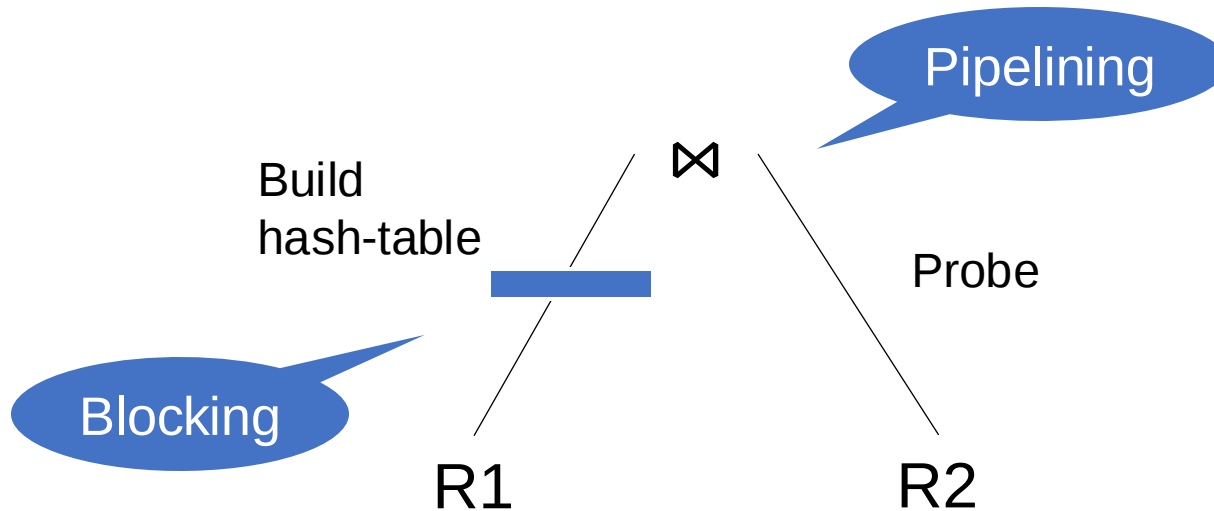
Hash-Join



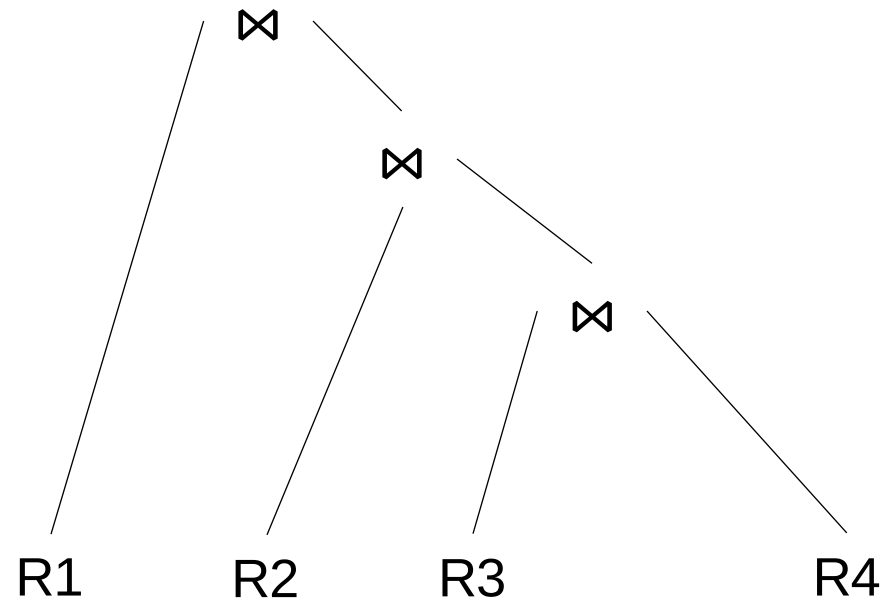
Hash-Join



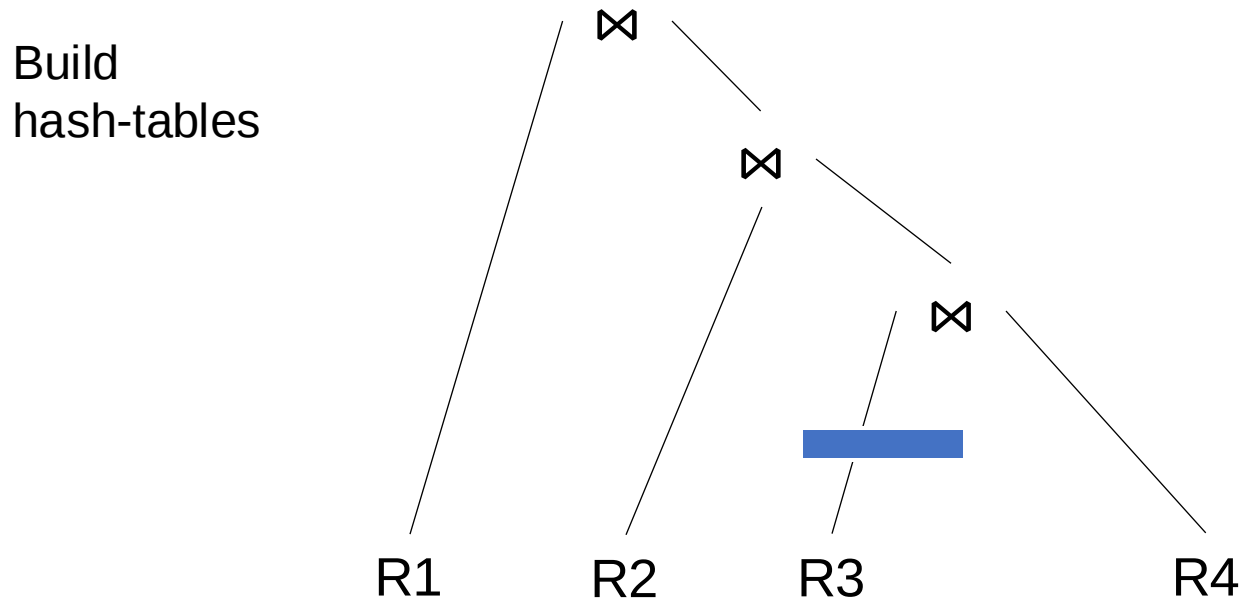
Hash-Join



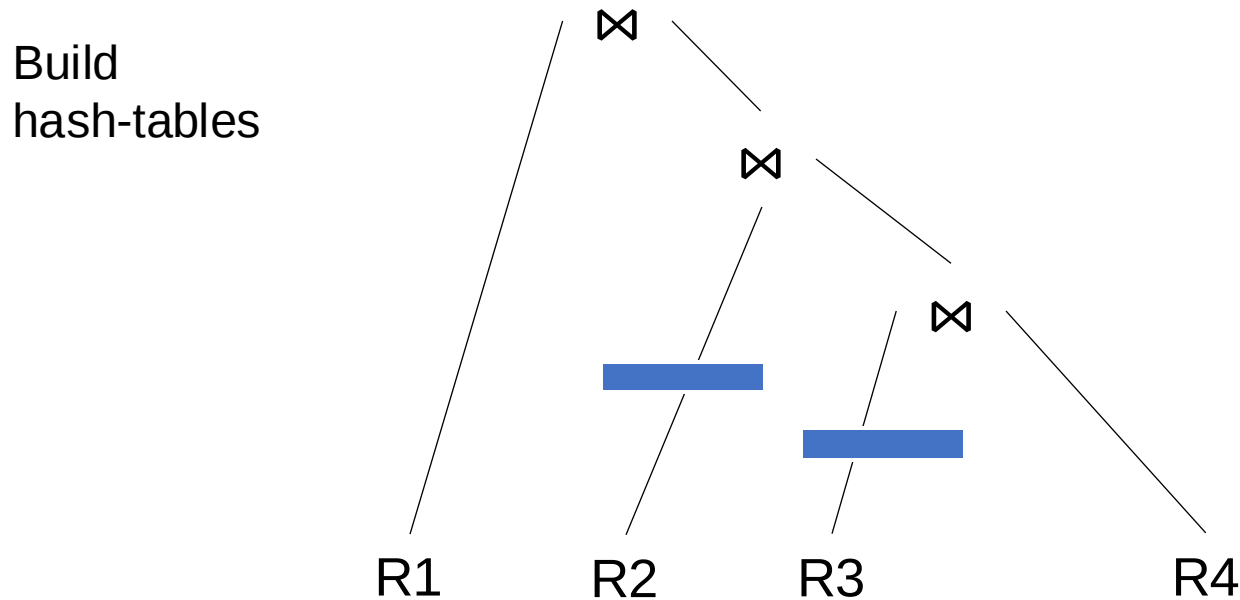
Hash-Join



Hash-Join

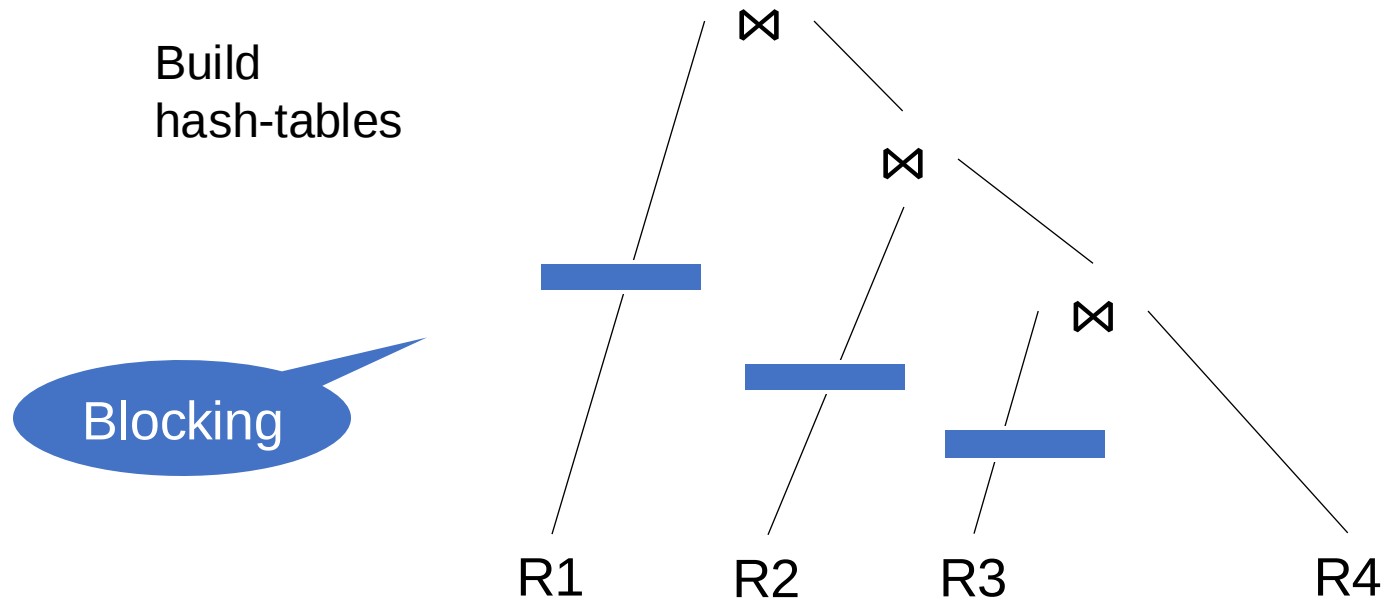


Hash-Join



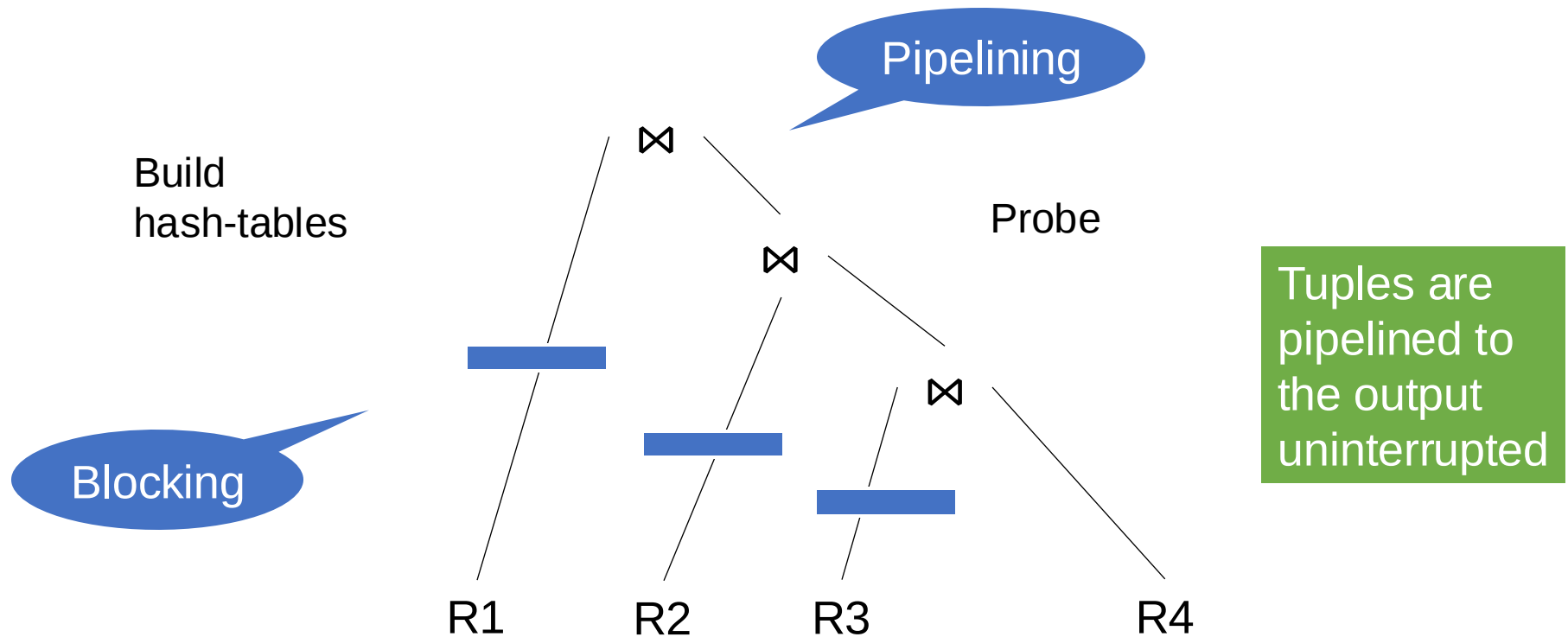
Pipelining v.s. Blocking

Hash-Join

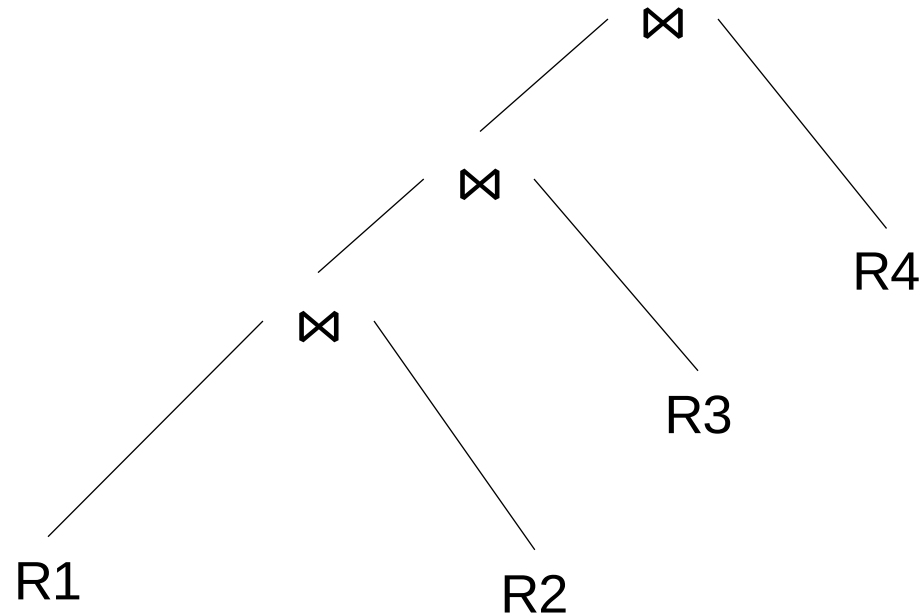


Pipelining v.s. Blocking

Hash-Join

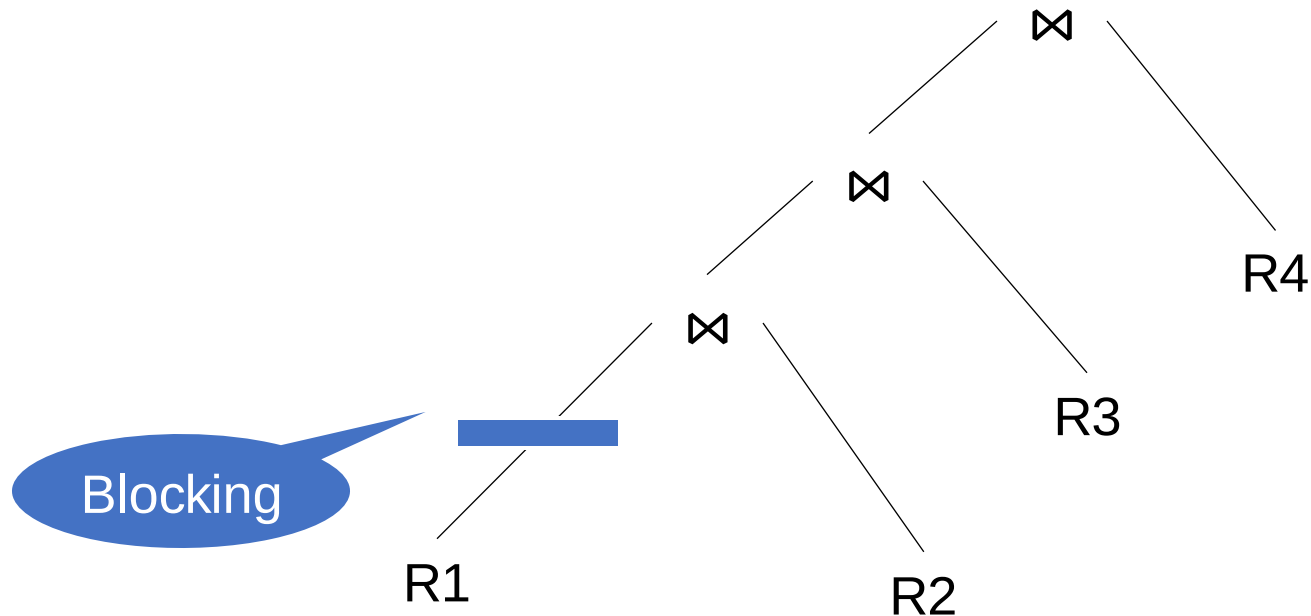


Hash-Join



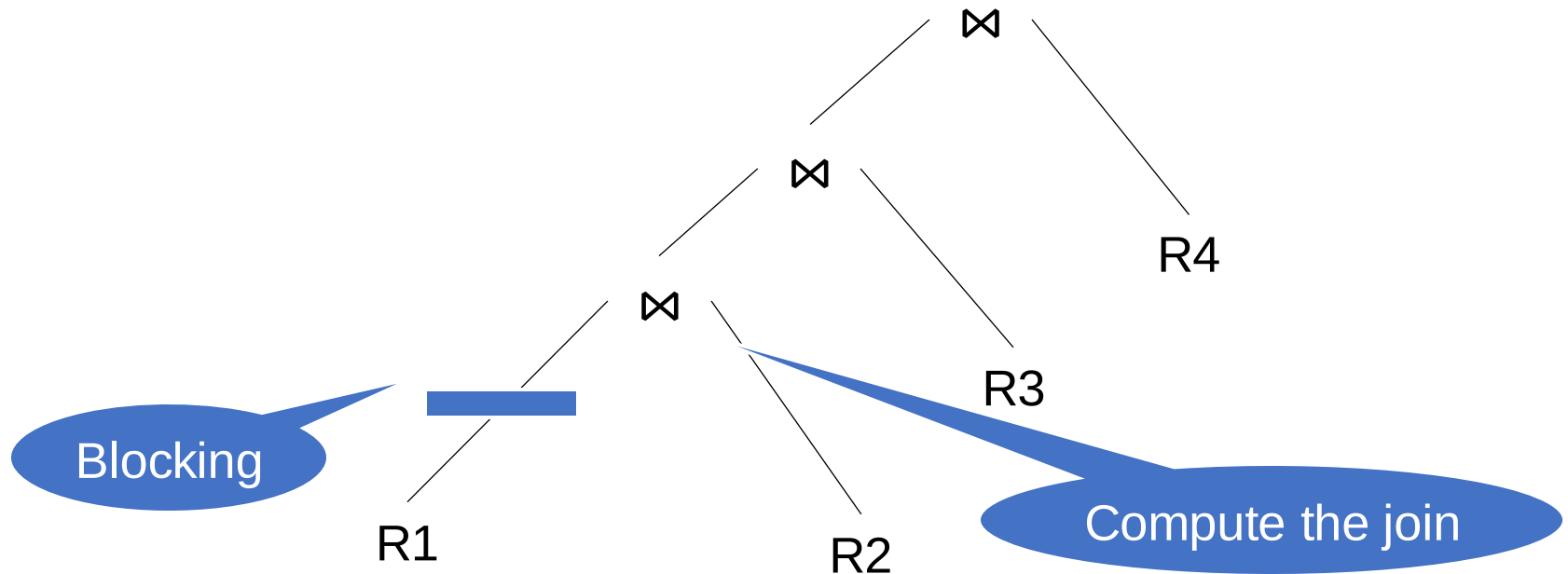
Pipelining v.s. Blocking

Hash-Join



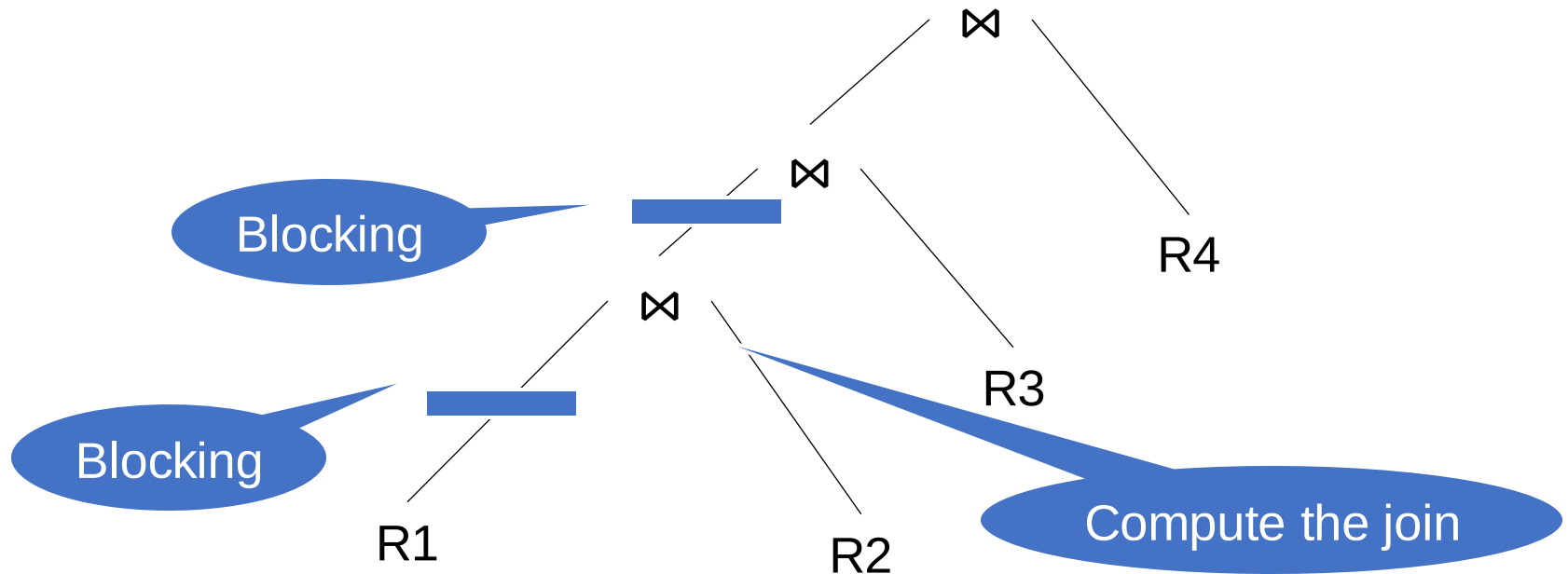
Pipelining v.s. Blocking

Hash-Join



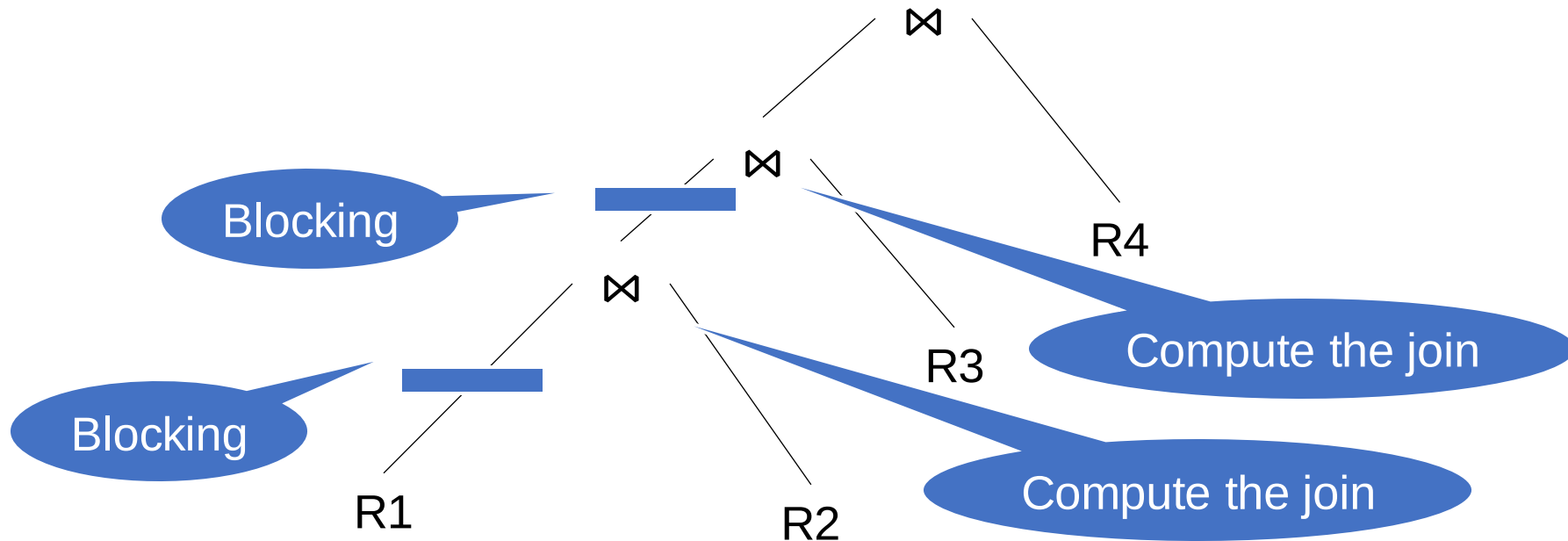
Pipelining v.s. Blocking

Hash-Join



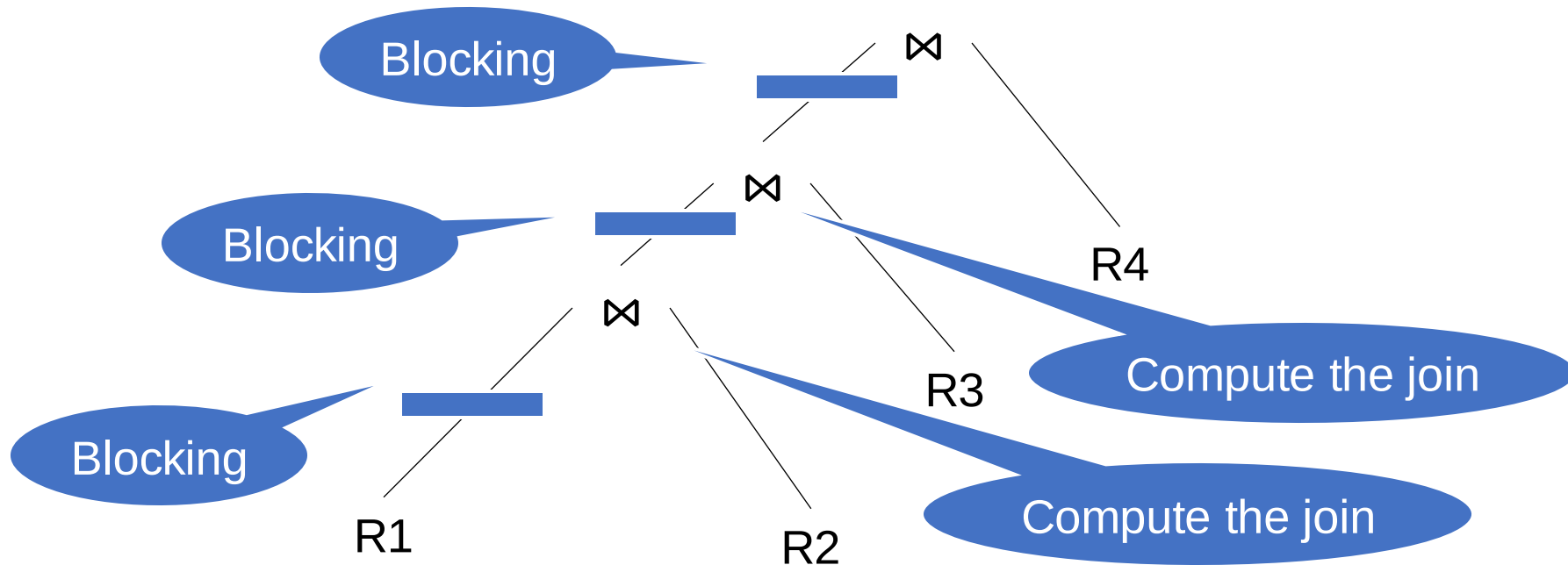
Pipelining v.s. Blocking

Hash-Join



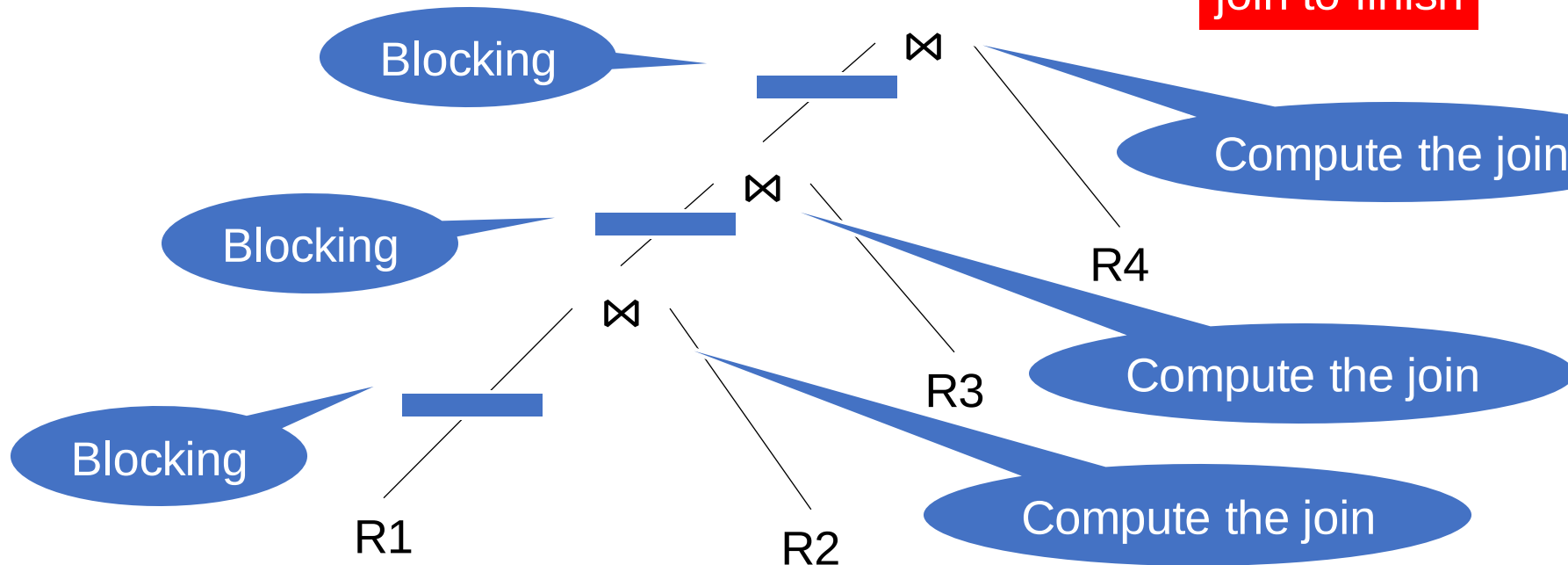
Pipelining v.s. Blocking

Hash-Join



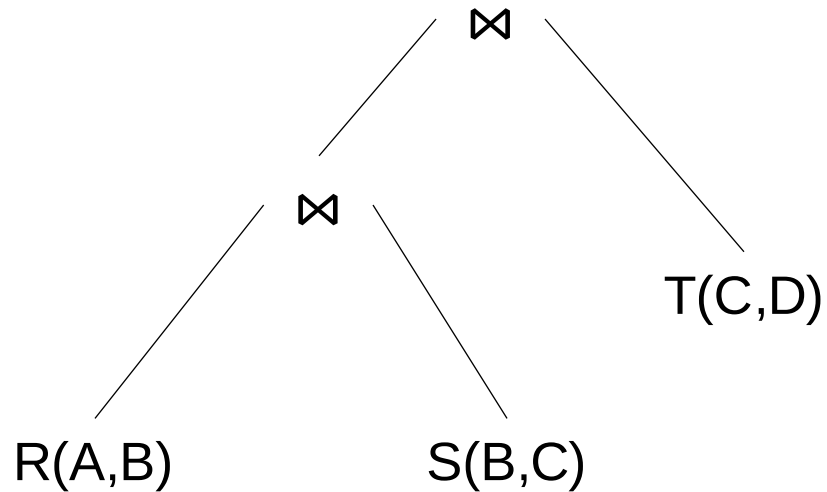
Hash-Join

Each build phase must wait for the previous join to finish



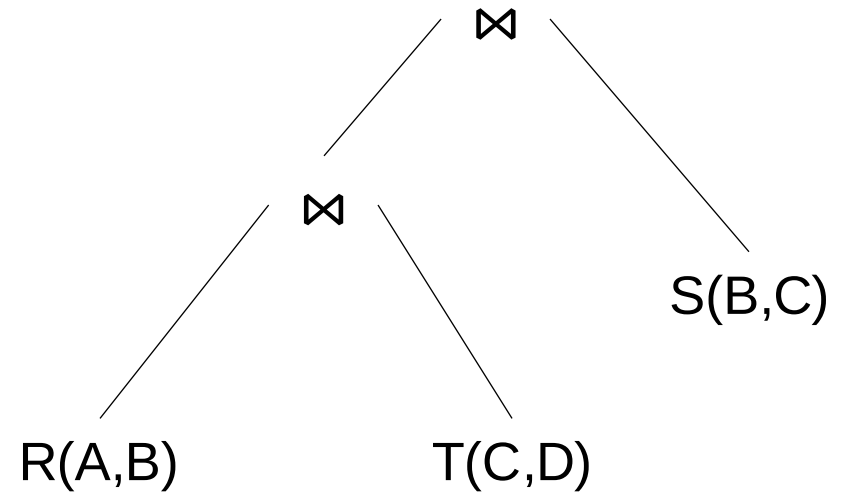
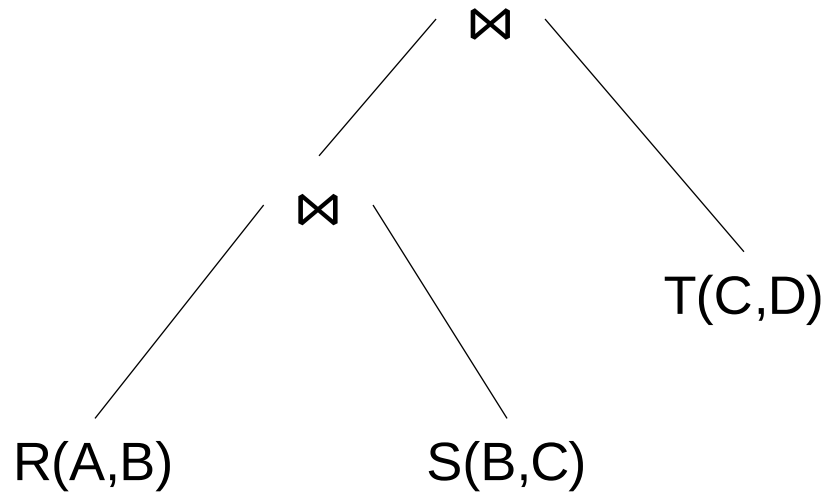
With or Without Cartesian Products

All joins are natural joins:



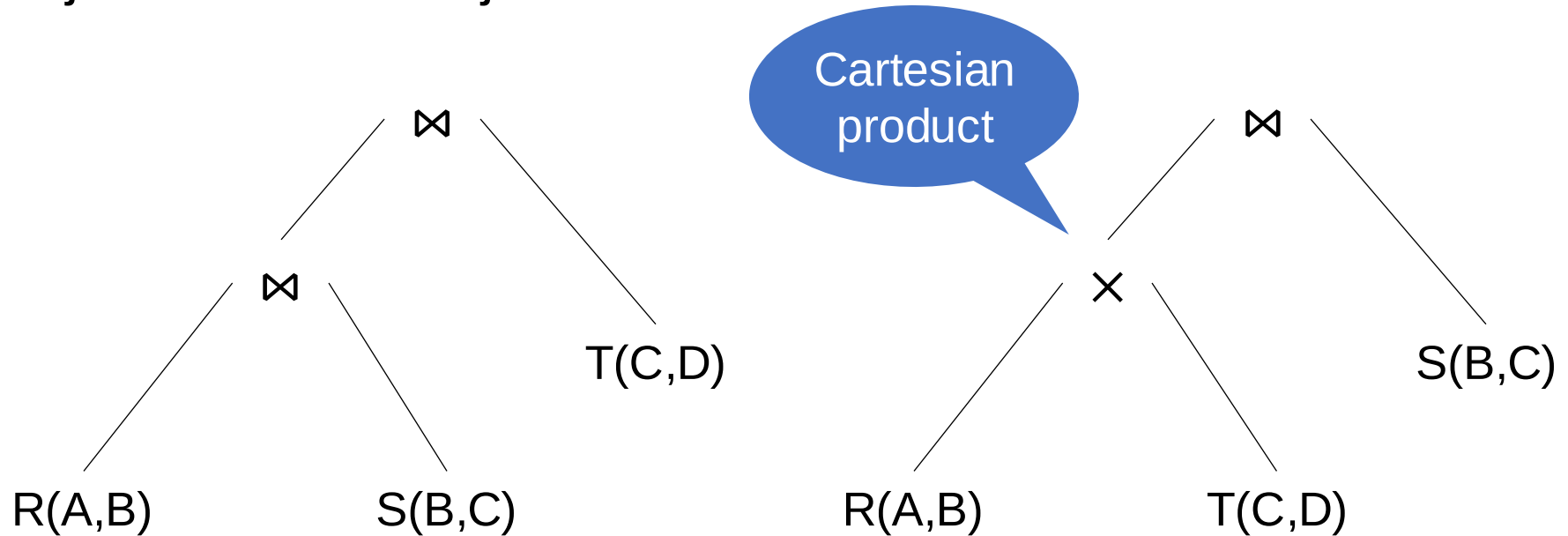
With or Without Cartesian Products

All joins are natural joins:



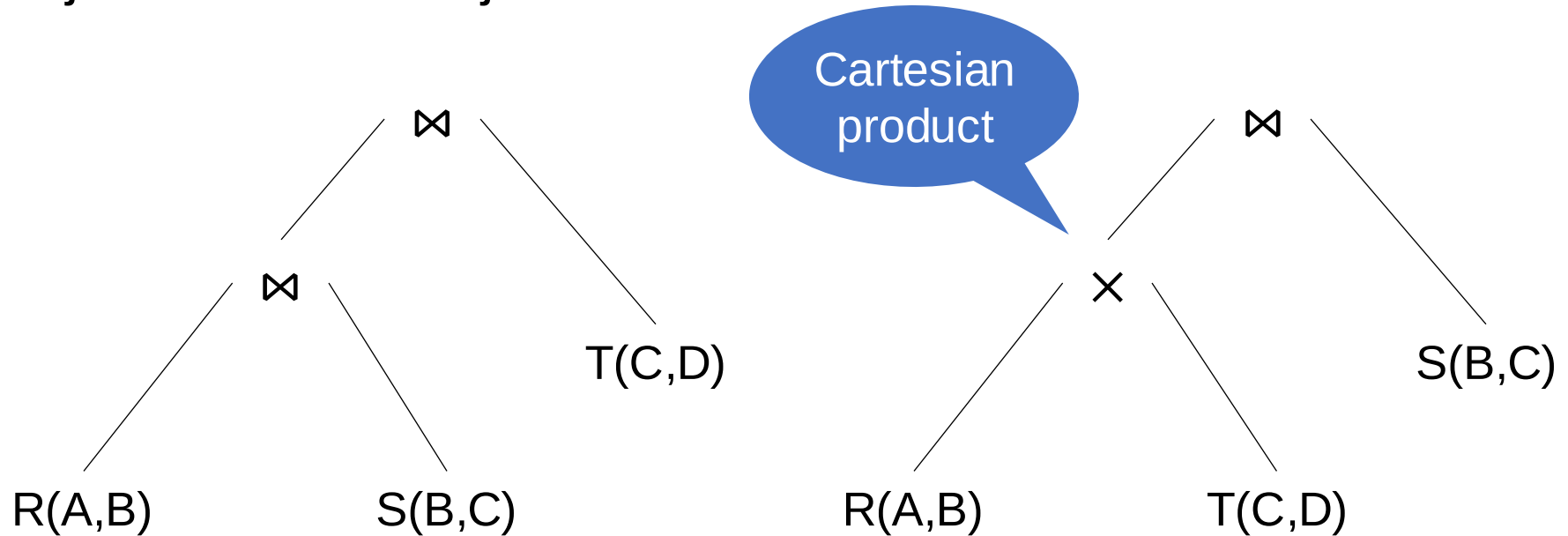
With or Without Cartesian Products

All joins are natural joins:



With or Without Cartesian Products

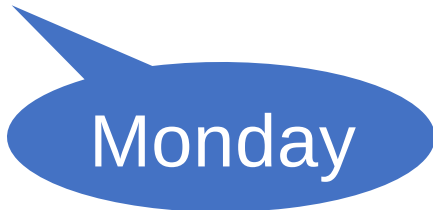
All joins are natural joins:



Most optimizers avoid cartesian products

Discussion

- The optimizer applies rewrite rules to generate alternative query plans
- In order to choose the cheapest plan, it needs to estimate the cardinality of various plans



Monday

Access Path Selection

Access path: implements a selection $\sigma_p(R)$:

- A file scan, or
- An index-based selection

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

...		
...		

123	Jack	TA
...		
...		

...	...	...
...		
...		

...

...

Payroll data file

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

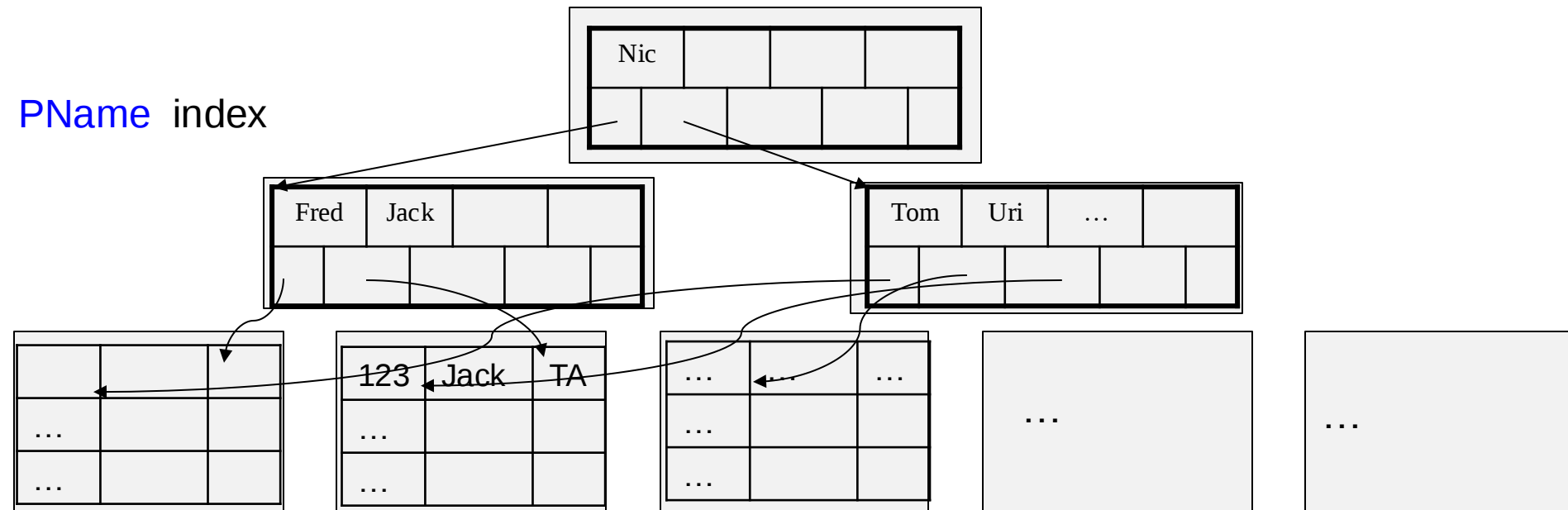
...			...	...
...				

Payroll data file

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```



Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

$\sigma_{\text{name}=\text{Allison}}$

Payroll

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

$\sigma_{\text{name}=\text{Allison}}$

Payroll

Option 1: Index-based selection

Cost = ???

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

Option 1: Index-based selection

$$\text{Cost} = T / V = 50$$

$\sigma_{\text{name}=\text{Allison}}$

Payroll

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

Option 1: Index-based selection

$$\text{Cost} = T / V = 50$$

$\sigma_{\text{name}=\text{Allison}}$

Payroll

Option 2: Table scan

$$\text{Cost} = ???$$

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$\sigma_{\text{name}=\text{Allison}}$

Payroll

Option 1: Index-based selection

Cost = $T / V = 50$

Option 2: Table scan

Cost = $B = 5000$

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

$\sigma_{\text{name}=\text{Allison}}$

Option 1: Index-based selection

Cost = $T / V = 50$

Payroll

Best access path

Option 2: Table scan

Cost = $B = 5000$

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)
```

$T(\text{Payroll}) = 1000000$

$B(\text{Payroll}) = 5000$

$V(\text{Payroll}, \text{name}) = 20000$

```
SELECT *  
FROM Payroll  
WHERE name = 'Allison';
```

```
SELECT *  
FROM Payroll x, Regist y  
WHERE x.UserID = y.UserID;
```



The index
won't help
this query

Indexes are not magic!

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)  
CREATE INDEX Pjob on Payroll(job)
```

```
SELECT *  
FROM Payroll  
WHERE job = 'TA';
```

Option 1: Index-based selection

Option 2: Table scan

$T(\text{Payroll}) = 1000000$
 $B(\text{Payroll}) = 5000$
 $V(\text{Payroll}, \text{name}) = 20000$
 $V(\text{Payroll}, \text{job}) = 25$

$\sigma_{\text{job}=\text{TA}}$
|
Payroll

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)  
CREATE INDEX Pjob on Payroll(job)
```

```
SELECT *  
FROM Payroll  
WHERE job = 'TA';
```

$T(\text{Payroll}) = 1000000$
 $B(\text{Payroll}) = 5000$
 $V(\text{Payroll}, \text{name}) = 20000$
 $V(\text{Payroll}, \text{job}) = 25$

$\sigma_{\text{job}=\text{TA}}$

Payroll

Option 1: Index-based selection

$\text{Cost} = T / V = 40000$

Option 2: Table scan

$\text{Cost} = B = 5000$

Best access path

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)  
CREATE CLUSTERED INDEX Pjob on Payroll(job)
```

```
SELECT *  
FROM Payroll  
WHERE job = 'TA';
```

$T(\text{Payroll}) = 1000000$
 $B(\text{Payroll}) = 5000$
 $V(\text{Payroll}, \text{name}) = 20000$
 $V(\text{Payroll}, \text{job}) = 25$

$\sigma_{\text{job}=\text{TA}}$

Payroll

Option 1: Index-based selection

Cost = ???

Option 2: Table scan

Cost = B = 5000

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name)  
CREATE CLUSTERED INDEX Pjob on Payroll(job)
```

```
SELECT *  
FROM Payroll  
WHERE job = 'TA';
```

$T(\text{Payroll}) = 1000000$
 $B(\text{Payroll}) = 5000$
 $V(\text{Payroll}, \text{name}) = 20000$
 $V(\text{Payroll}, \text{job}) = 25$

Option 1: Index-based selection

Cost = $B / V = 200$

Option 2: Table scan

Cost = $B = 5000$



Best access path

$\sigma_{\text{job}=\text{TA}}$

Payroll

Multiple Indexes

- DB administrator can create multiple indexes
- But only one can be clustered...
- ... and a selection can only use only one index

Access Path Selection

```
CREATE TABLE Payroll(UserID int, name text, job text, salary int)
```

Payroll:

--	--

--	--

--	--

--	--

--	--

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);
```

Payroll:

--	--

--	--

--	--

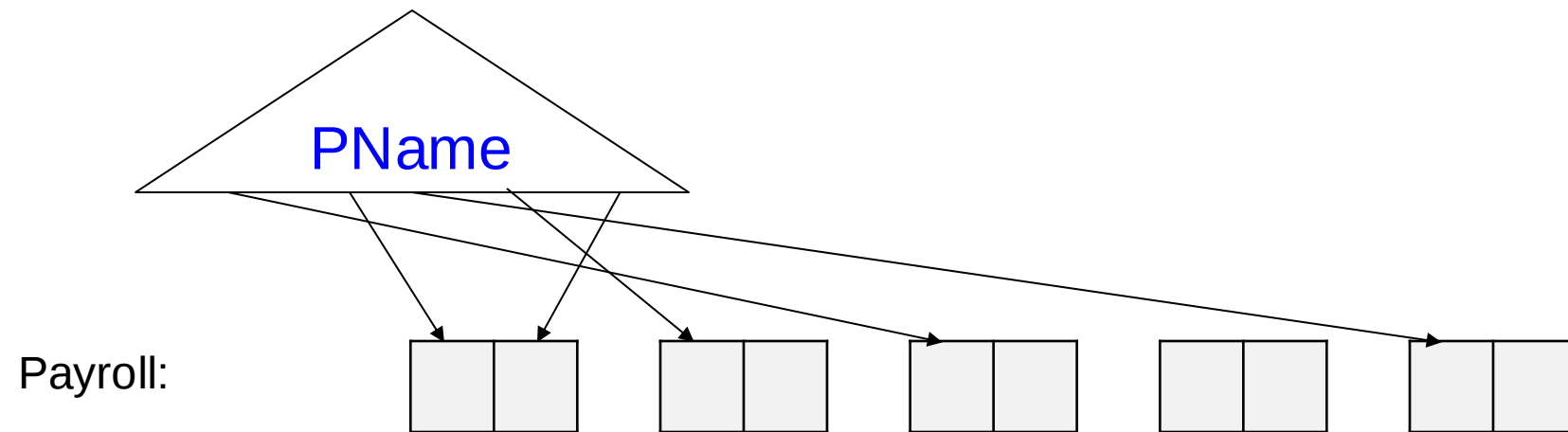
--	--

--	--

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

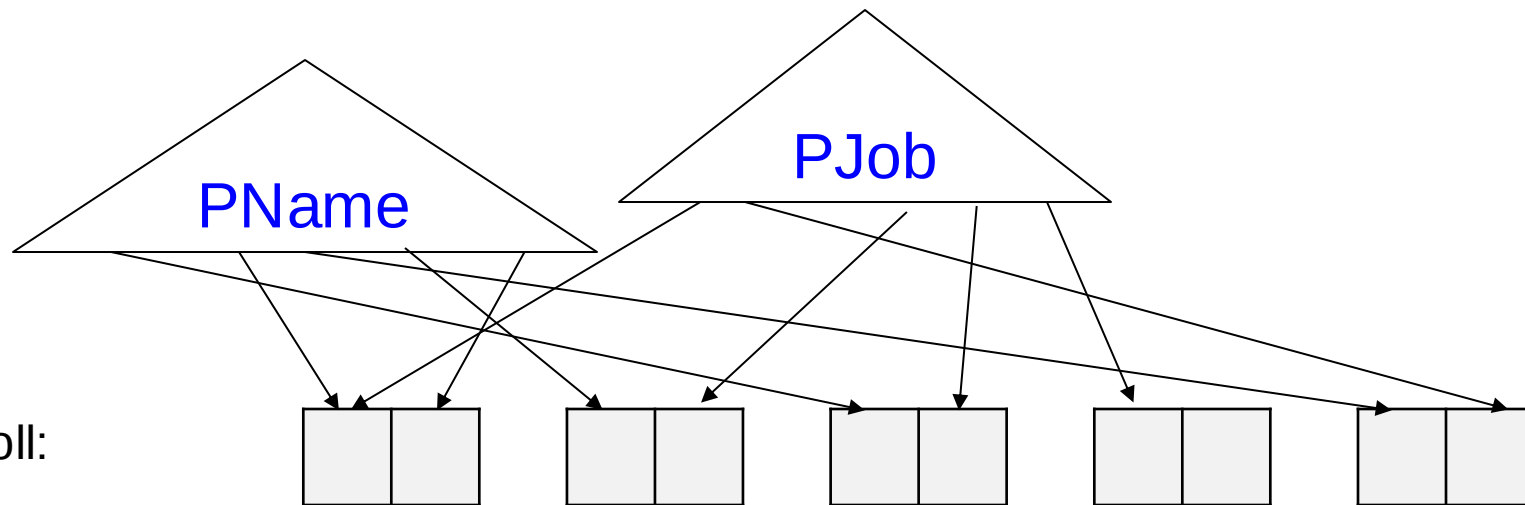
```
CREATE INDEX Pname on Payroll(name);
```



Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

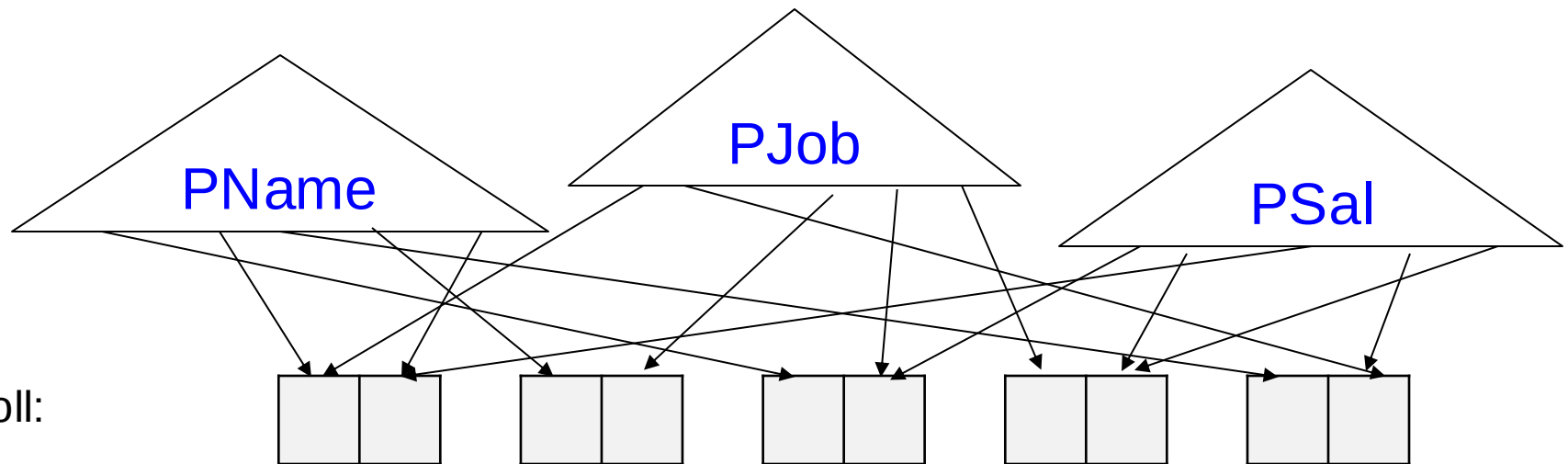
```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);
```



Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```



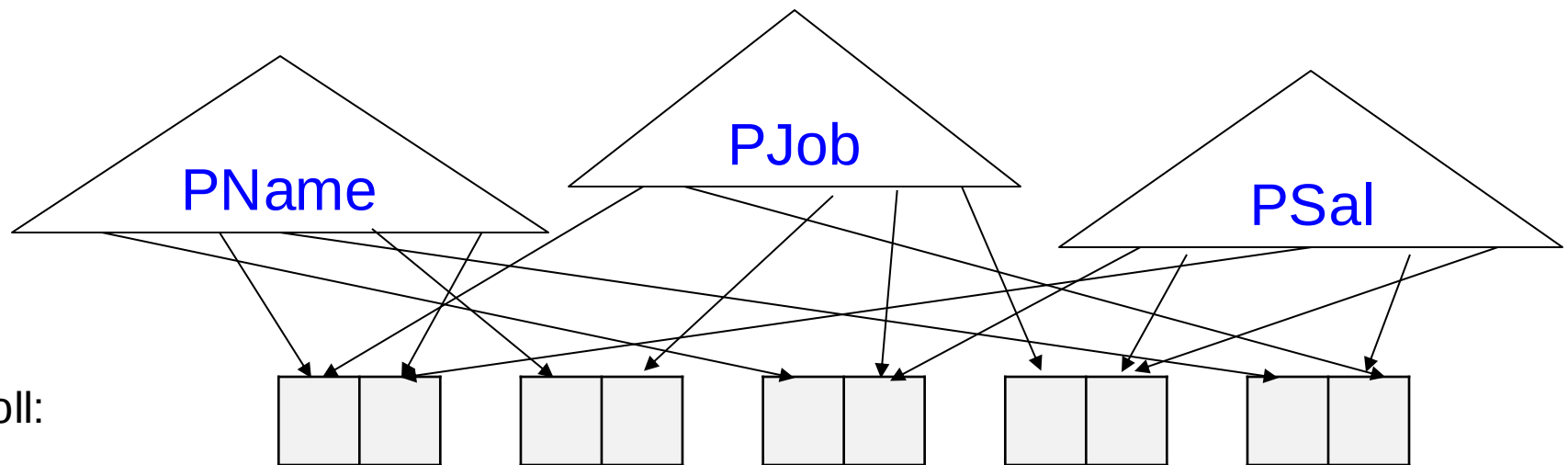
Payroll:

Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```



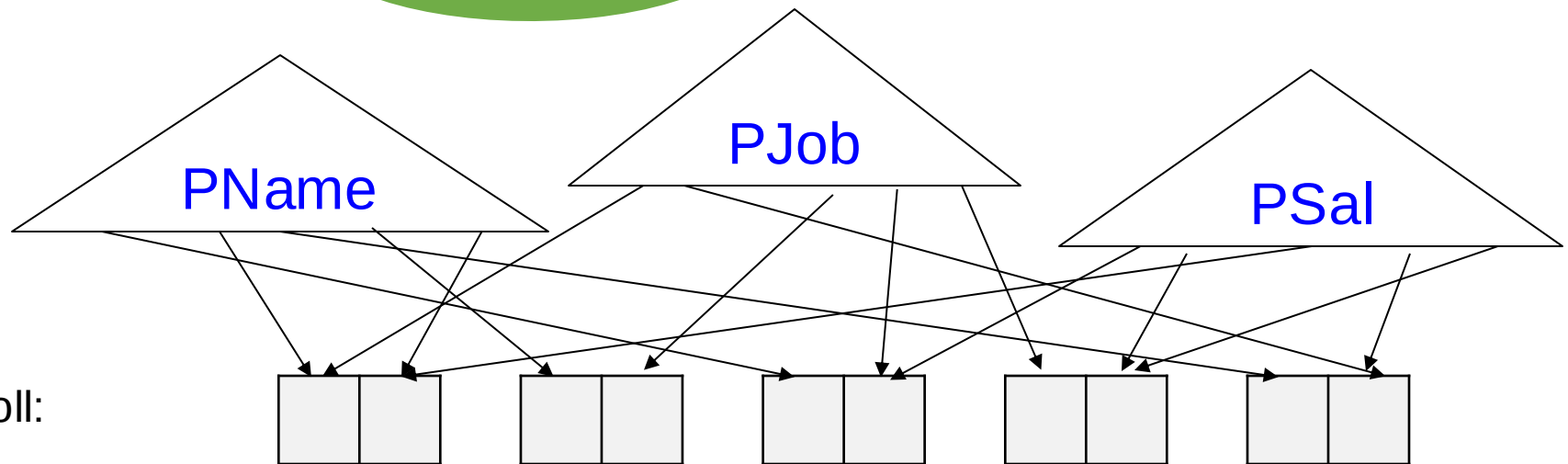
Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```

Access path:
use Pjob index



Payroll:

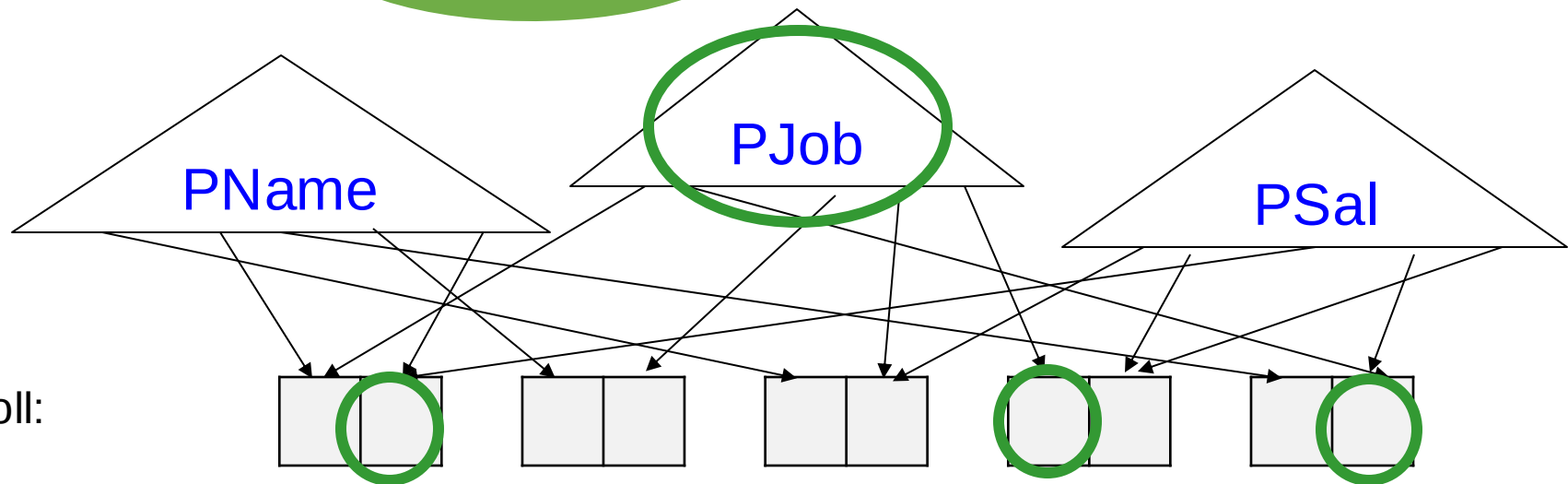
Access Path Selection

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```

Access path:
use Pjob index



Payroll:

Access Path Selection

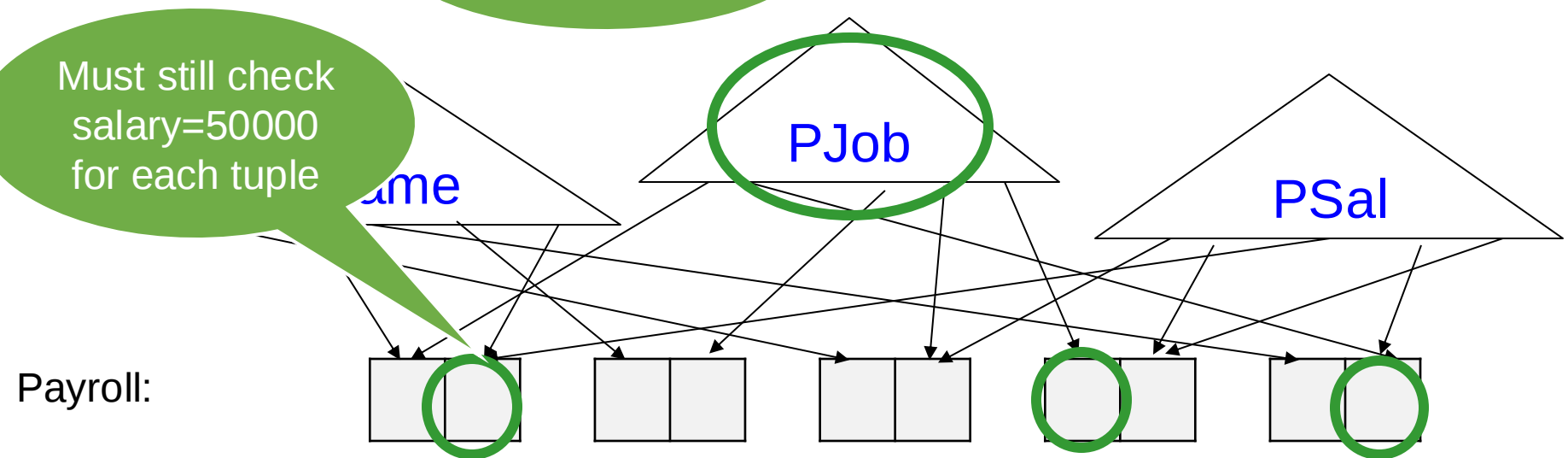
```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```

Access path:
use Pjob index

Must still check
salary=50000
for each tuple



Payroll:

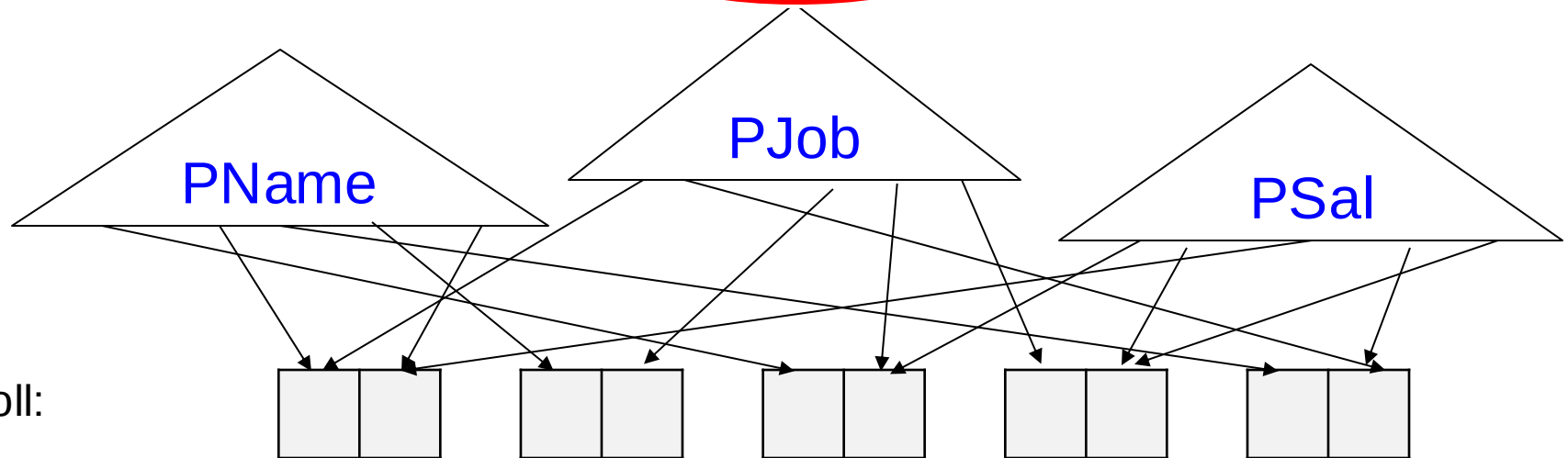
Access Path Selection

```
CREATE TABLE Payroll(UserID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```

Access path:
use Psal index



Payroll:

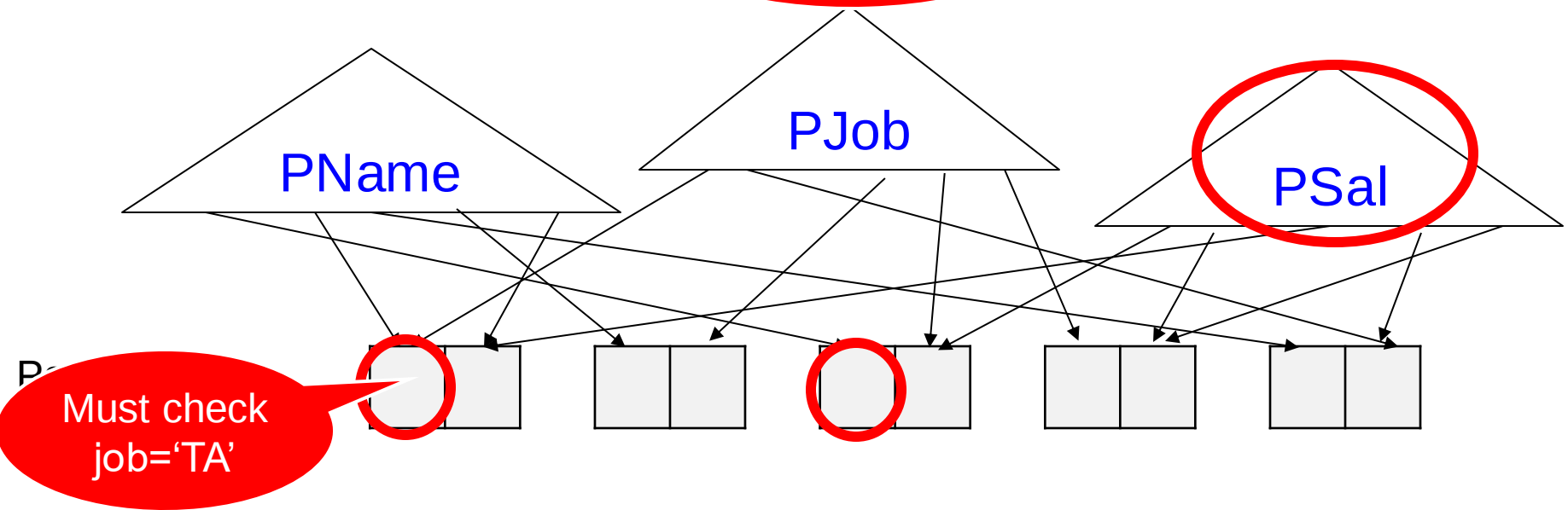
Access Path Selection

```
CREATE TABLE Payroll(UserID int, name text, job text, salary int)
```

```
CREATE INDEX Pname on Payroll(name);  
CREATE INDEX Pjob on Payroll(job);  
CREATE INDEX Psal on Payroll(salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
and salary=50000;
```

Access path:
use Psal index



Multi-attribute Index

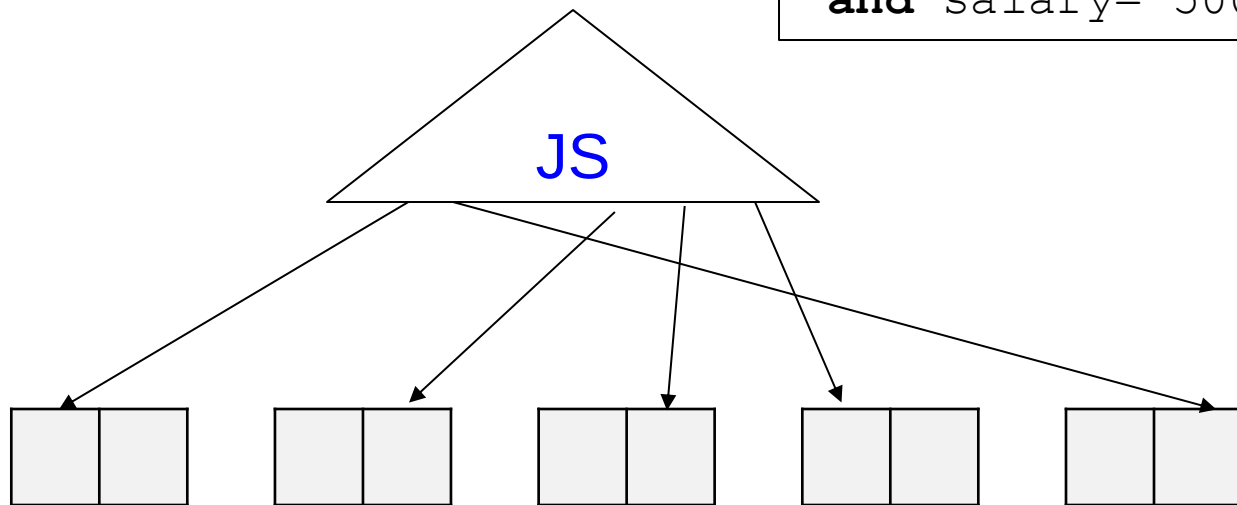
- An index can be on multiple attributes: A, B, C, ...
 - Create index on R(A,B,C)
 - Create index on R(C,A,B)
- Values are concatenated, then sorted
- Attribute order matters

Multi-attribute Index

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX JS on Payroll(job,salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
      and salary='50000';
```

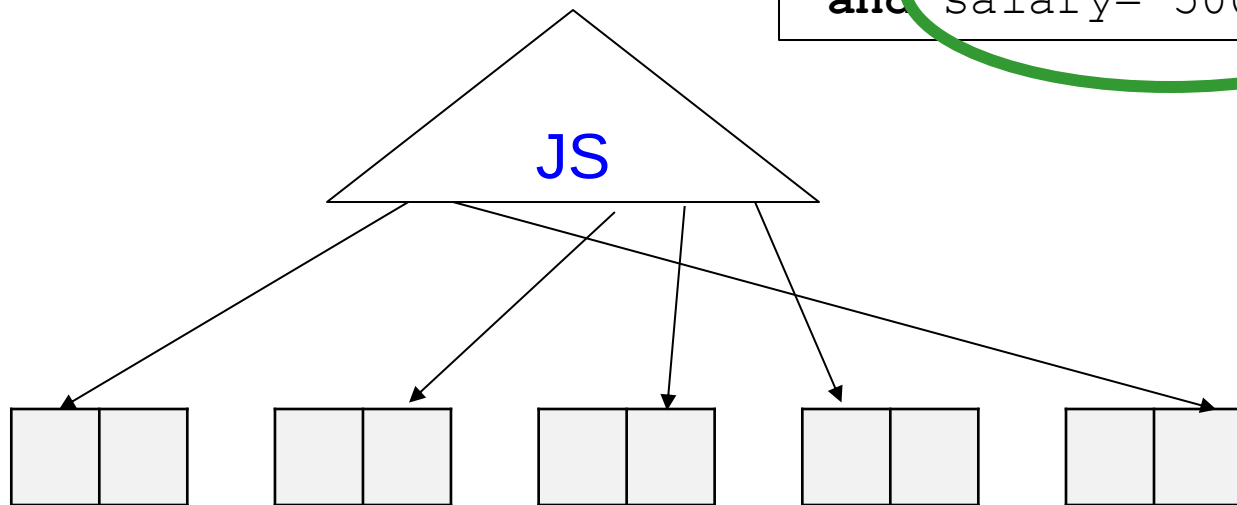


Multi-attribute Index

```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX JS on Payroll(job,salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
      and salary='50000';
```



Payroll:

Multi-attribute Index

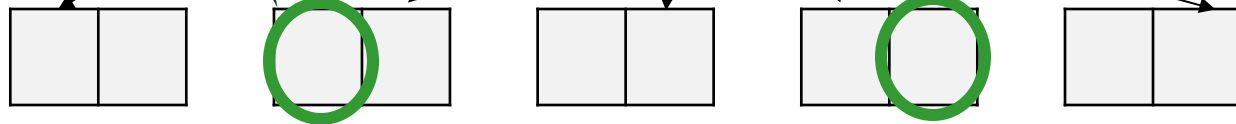
```
CREATE TABLE Payroll(userID int, name text, job text, salary int)
```

```
CREATE INDEX JS on Payroll(job,salary);
```

```
SELECT *  
FROM Payroll  
WHERE job='TA'  
      and salary='50000';
```

Each tuple
satisfies both
conditions

JS



Payroll:

Multi-attribute Index

- A pair ordered lexicographically:
 - ('Director', 200000) < ('Prof', 50000) < ('Prof', 200000) < ...
- Only 1st attribute is known: range search
 - Find ('Prof', *)
- Only 2nd attribute is known: impossible
 - Find (*, 200000)