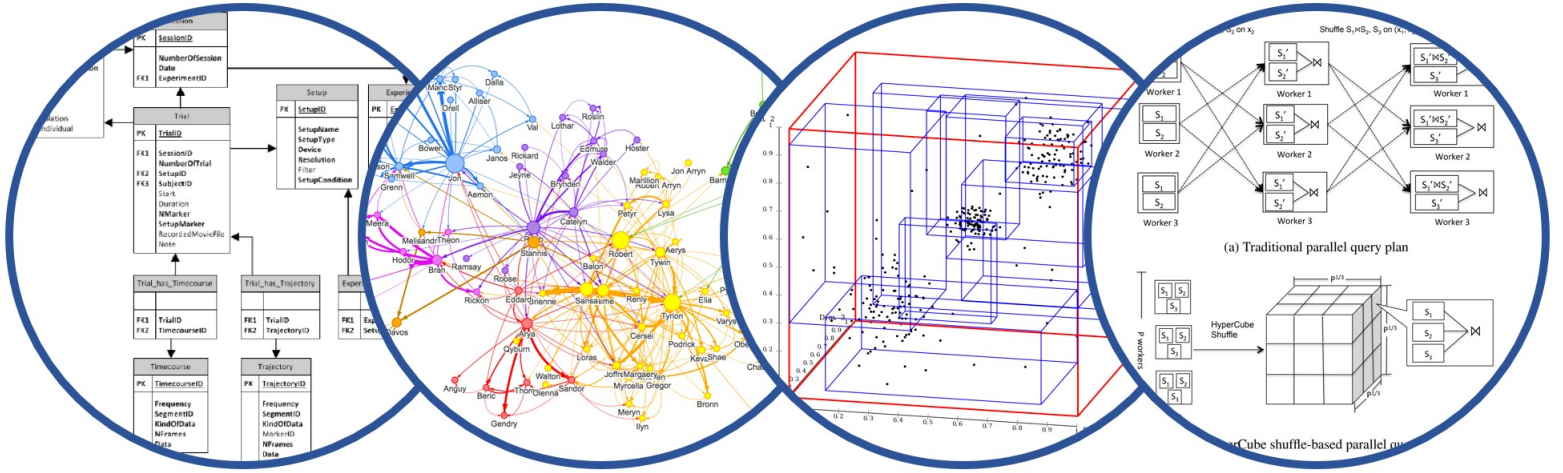


Introduction to Data Management

Query Optimization

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW6 Part 1 due tonight
- HW6 Part 2 due next Friday
- **No in-person Lecture on Friday, May 24**:
  Please watch recorded video instead (on canvas)

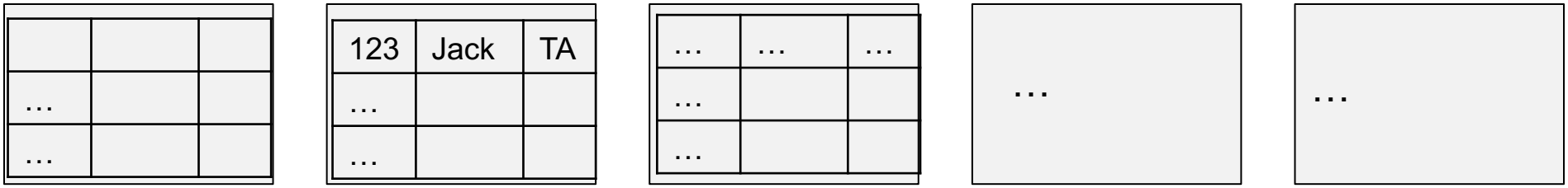
Indexes in SQL

Index in SQL

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

Index in SQL

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

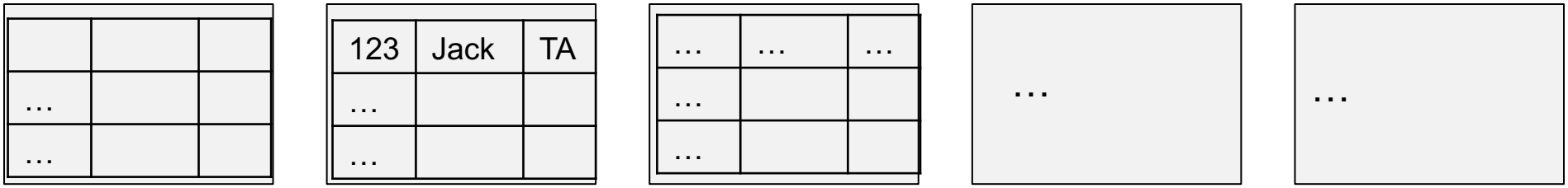


Payroll data file

Index in SQL

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

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

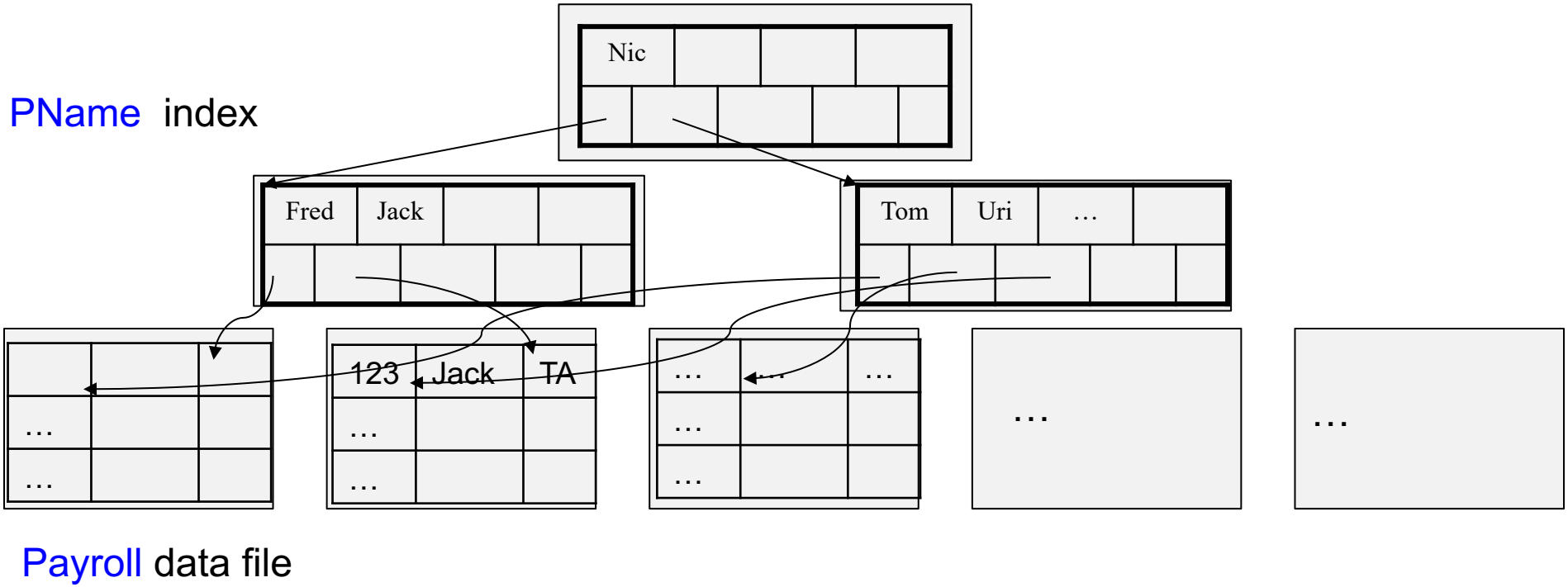


Payroll data file

Index in SQL

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

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



Index in SQL

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

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

Index in SQL

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

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

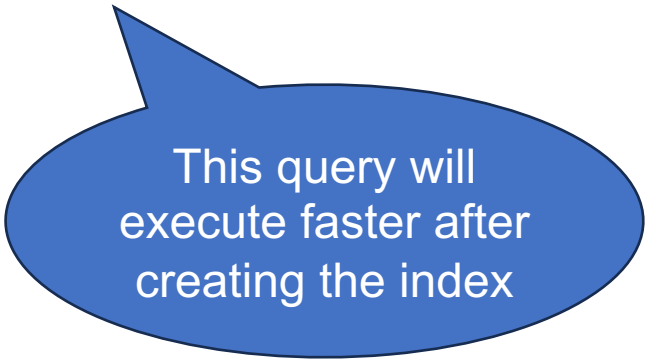
```
SELECT *
FROM Payroll
WHERE name = 'Allison';
```

Index in SQL

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

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

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



Index in SQL

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

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

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

This query will execute faster after creating the index

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

The index won't help this query

Discussion

- We can create many indexes for a table (next)
- But one selection predicate can only use one index
- Each insert/update/delete updates ALL indices

Choosing which indices to create is a difficult database administration task

Create Index

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

Payroll:

Create Index

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

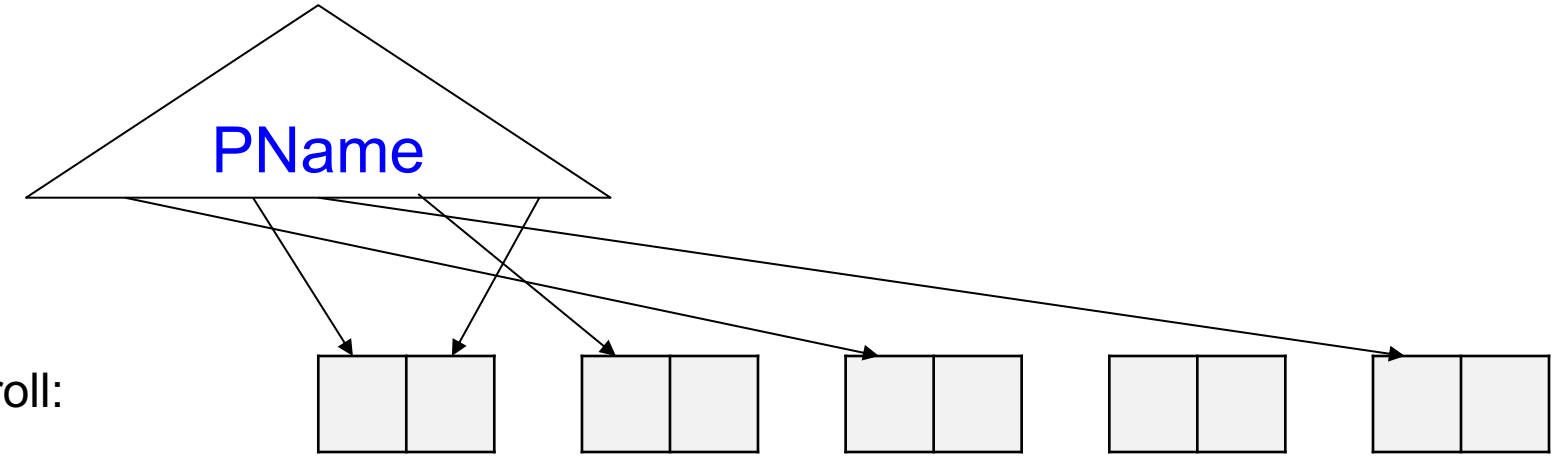
```
CREATE INDEX Pname on Payroll(name);
```

Payroll:

Create Index

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

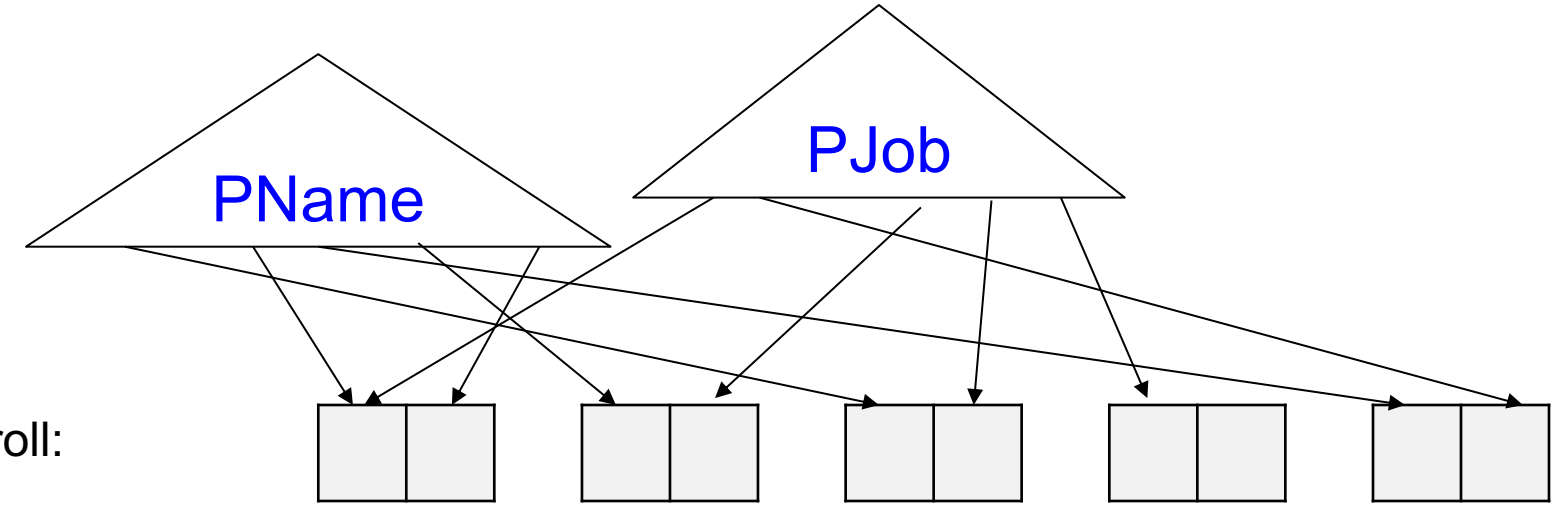
```
CREATE INDEX Pname on Payroll(name);
```



Create Index

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

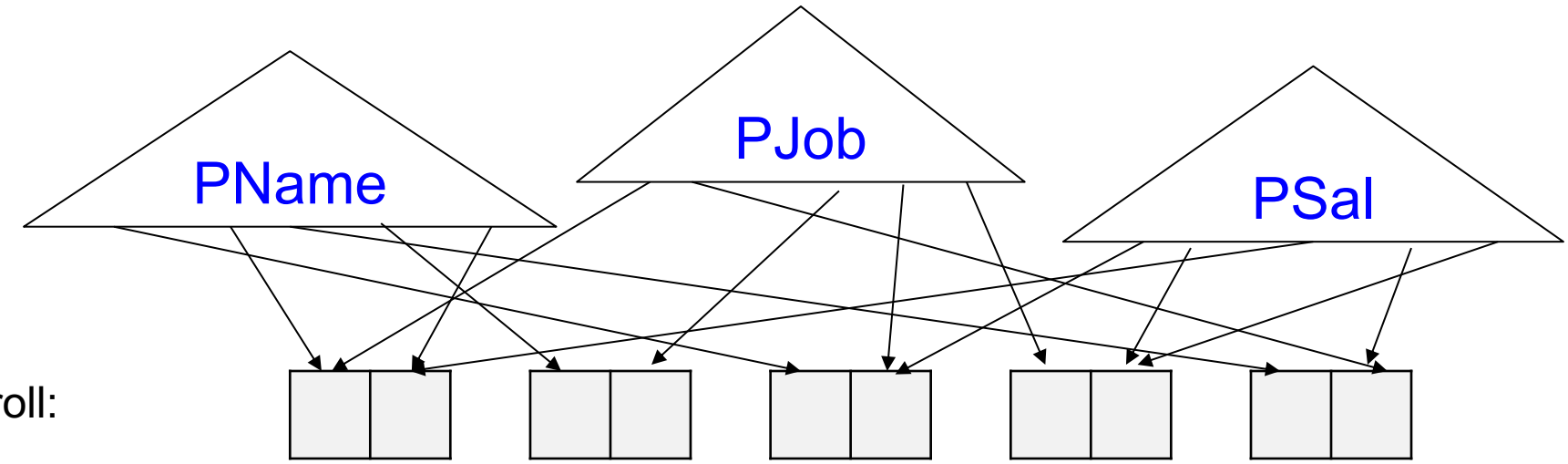
```
CREATE INDEX Pname on Payroll(name);
CREATE INDEX Pjob  on Payroll(job);
```



Create Index

```
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);
```

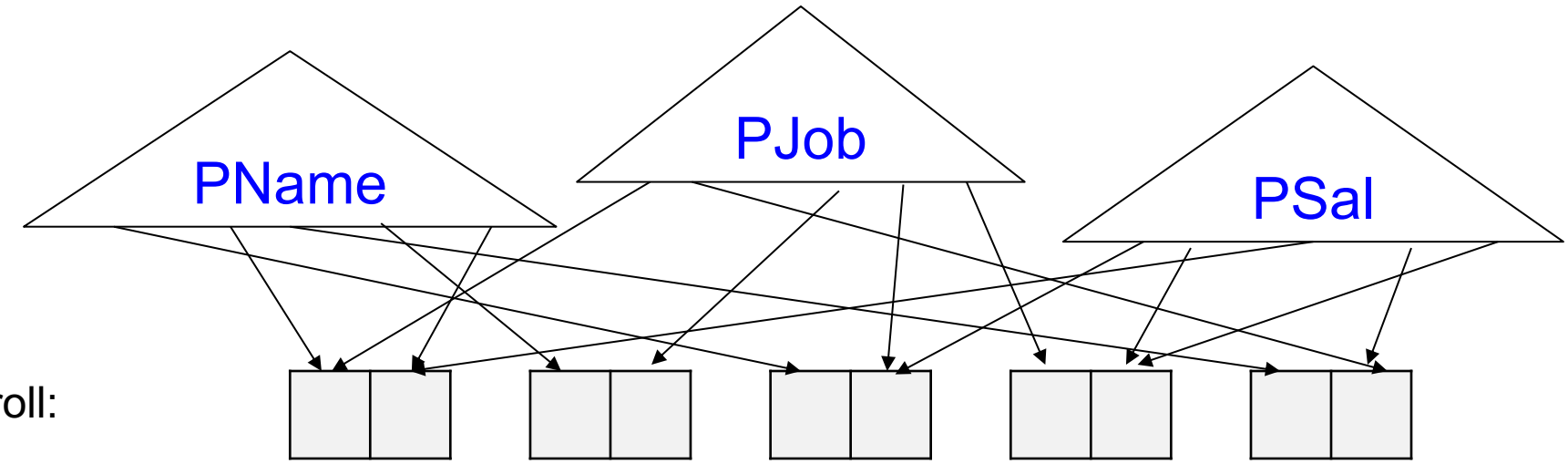


Create Index

```
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;
```



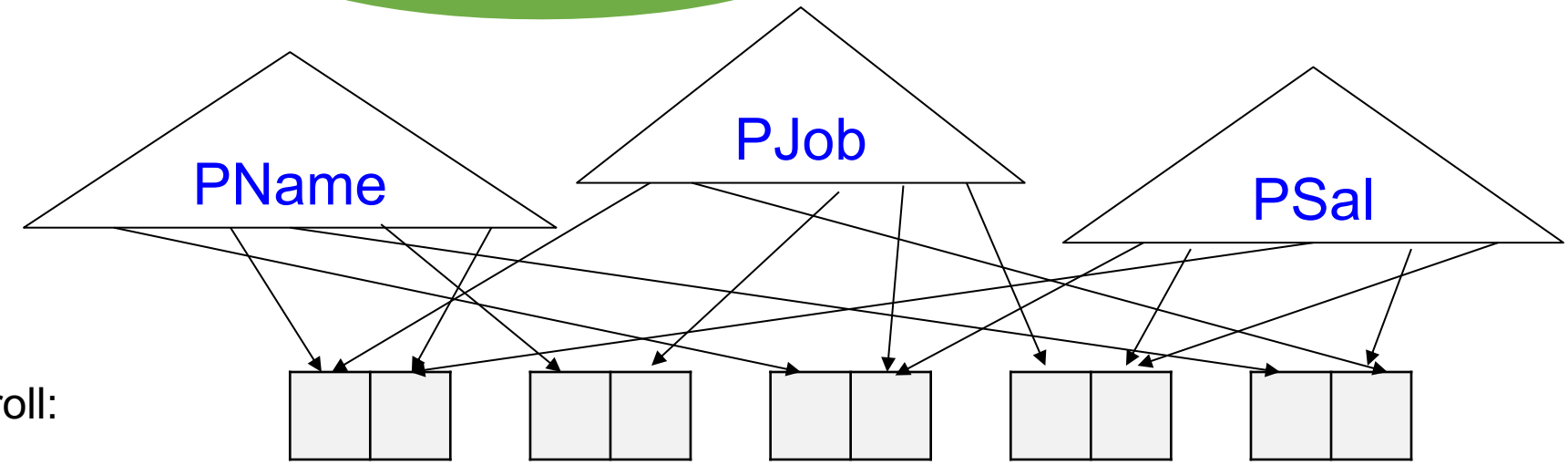
Create Index

```
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;
```

Optimizer may choose index on Job



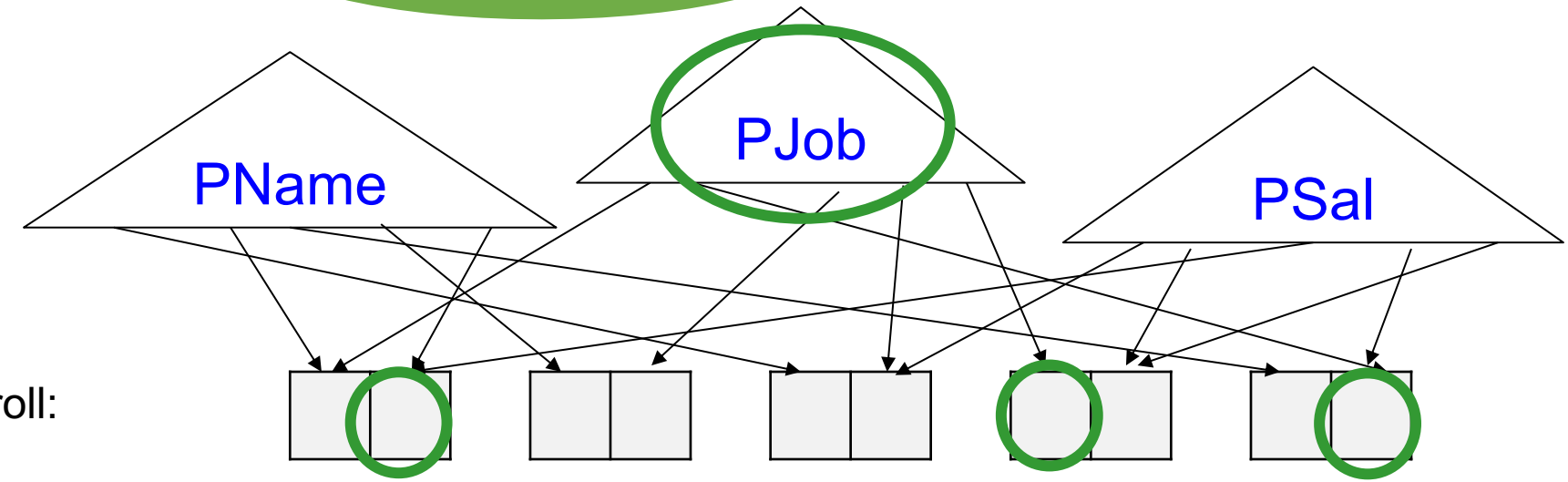
Create Index

```
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;
```

Optimizer may choose index on Job



Create Index

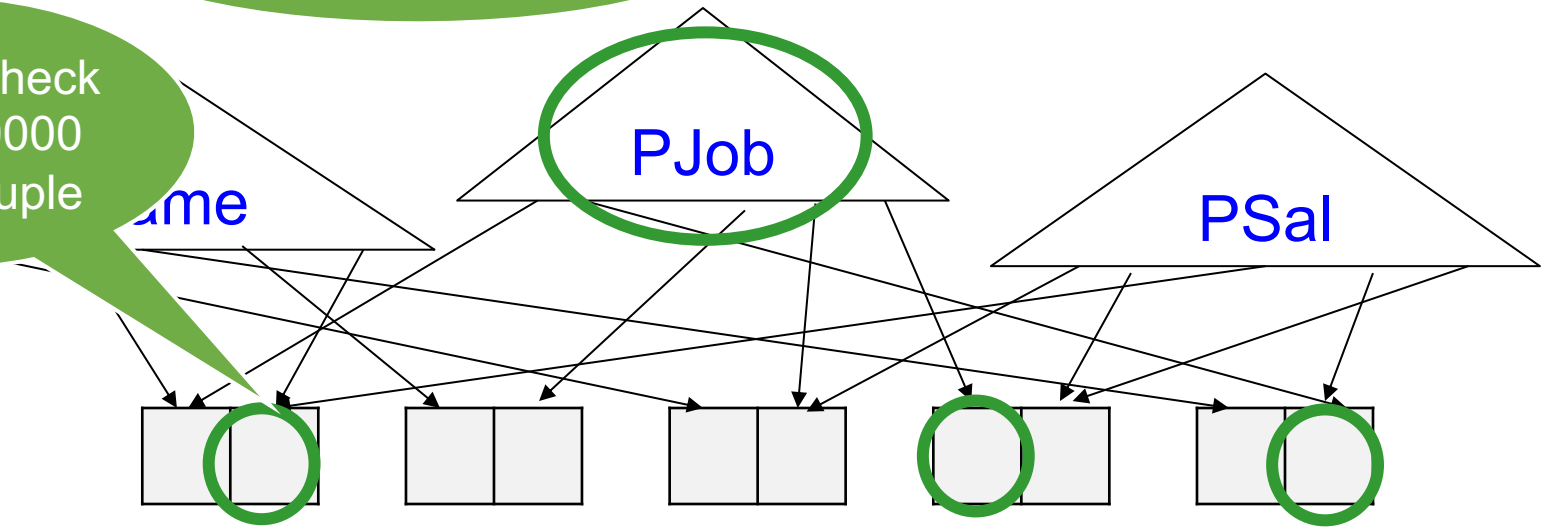
```
CREATE TABLE Payroll(UserID int, name text, job text, salary int)
```

```
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;
```

Optimizer may choose index on Job

Must still check salary=50000 for each tuple



Payroll:

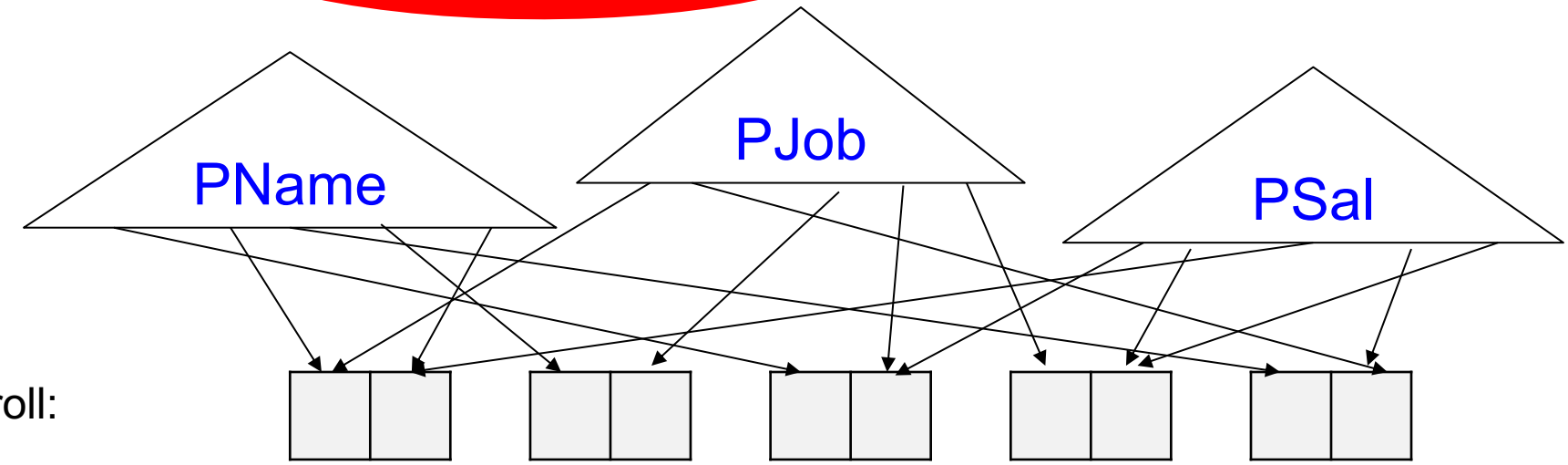
Create Index

```
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;
```

…or optimizer may choose index on Salary



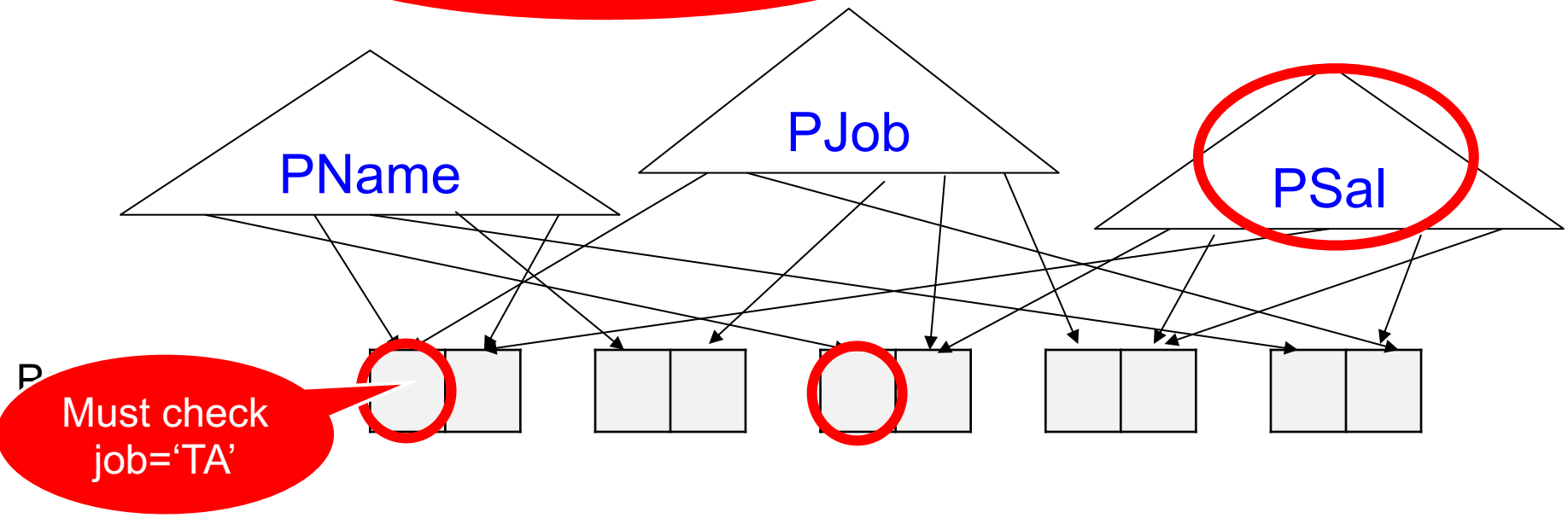
Create Index

```
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;
```

…or optimizer may choose index on Salary



Discussion

- There may be multiple indexes relevant to
  Attr1=val1 and Attr2=val2 and …
- Optimizer needs to choose one, then iterate over the answers and filter out the other conditions
- This problem is called access path selection
- We can also create a multi-attribute index (next)

Multi-attribute Index

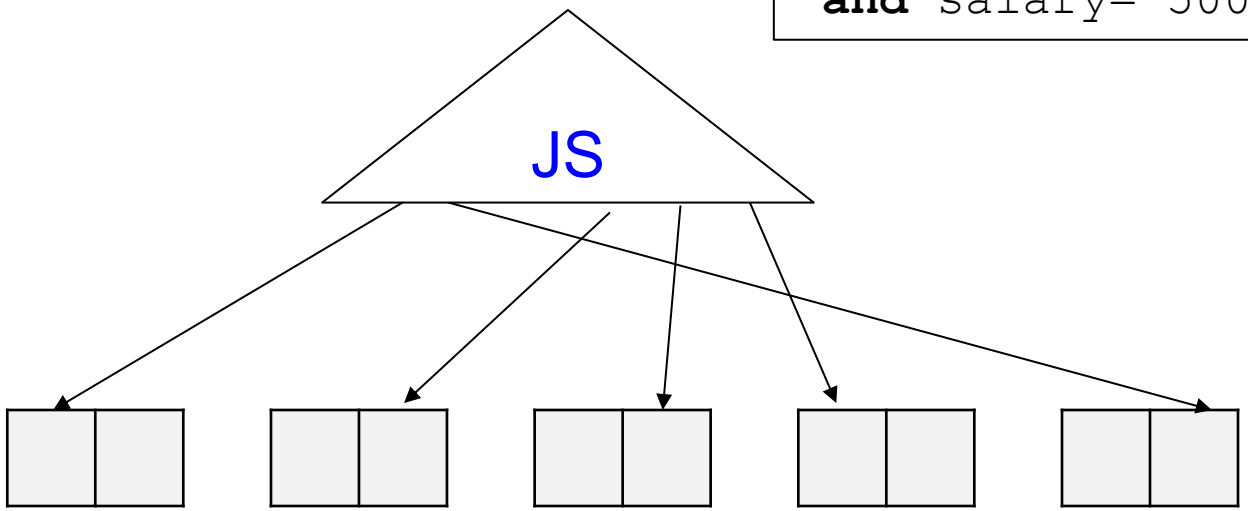
- An index can be on multiple attributes: A, B, C, …
  - Create index on R(A,B,C)
  - Create index on R(C,A,B)
- Values are concatenated, then sorted
- Attribute order matters

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';
```



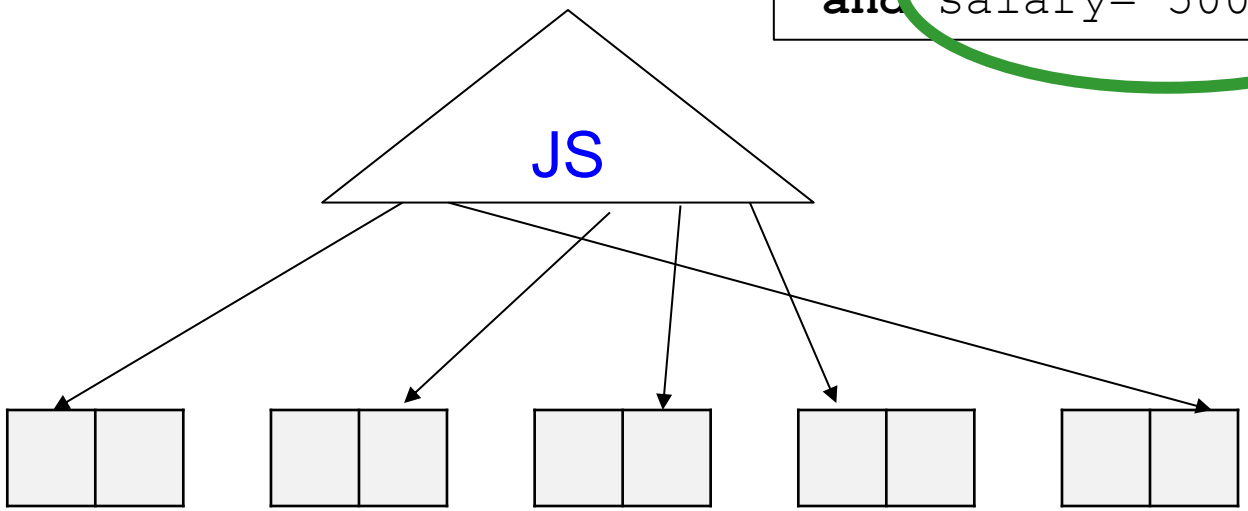
Payroll:

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';
```



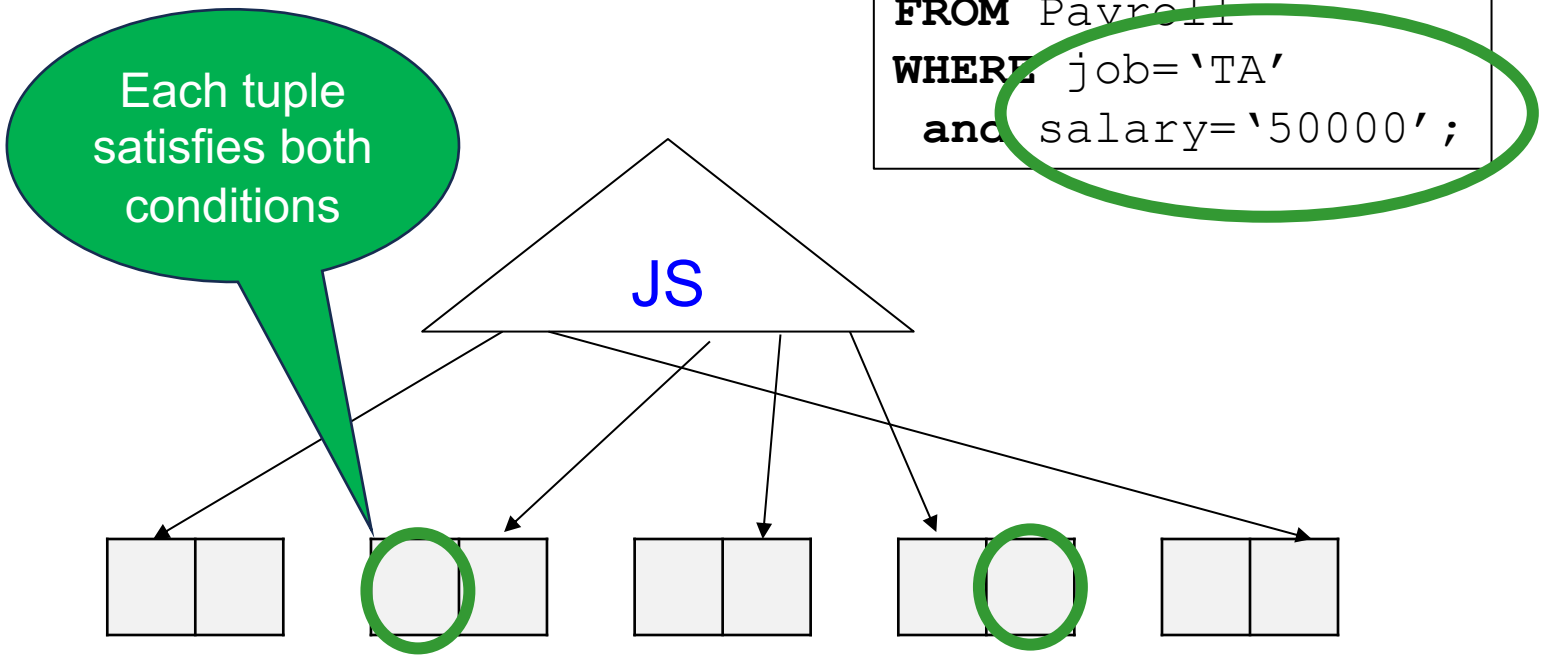
Payroll:

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';
```

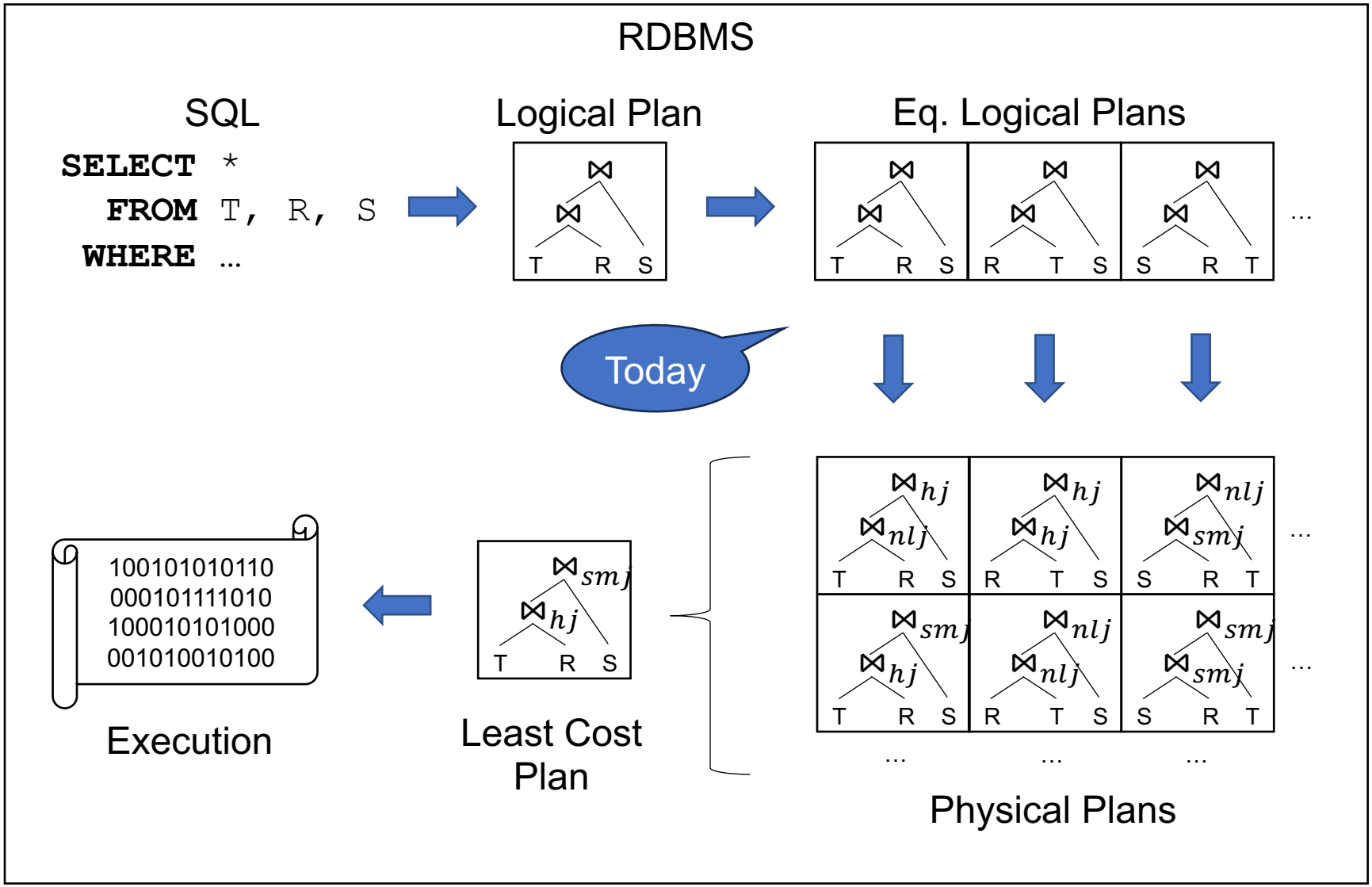


Multi-attribute Index

- A pair ordered lexicographically:
  - (‘Director’,200000) < (‘Prof’,50000)<(‘Prof’,200000)<…
- If only 1st attribute is known: range seach
  - Find (‘Prof’, *)
- If only 2nd attribute is known: impossible
  - Find (*, 200000)

Query Optimization

Plan Enumeration



Query Optimization

- Access path selection
- Rewrite rules
- Cardinality and cost estimation

Access Path

Access path: implements a selection $\sigma_P(R)$. Can be:

- A file scan, or
- An index *plus* a matching selection condition

Access Path Selection

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

Option 1: sequential scan of Payroll

Option 2: use the index on Payroll(job), filter salary=50000

Option 3: use the index on Payroll(salary), filter job='TA'

Rewrite Rules

- A rewrite rule is an identity in relational algebra

- It is similar to identities in linear algebra, e.g.

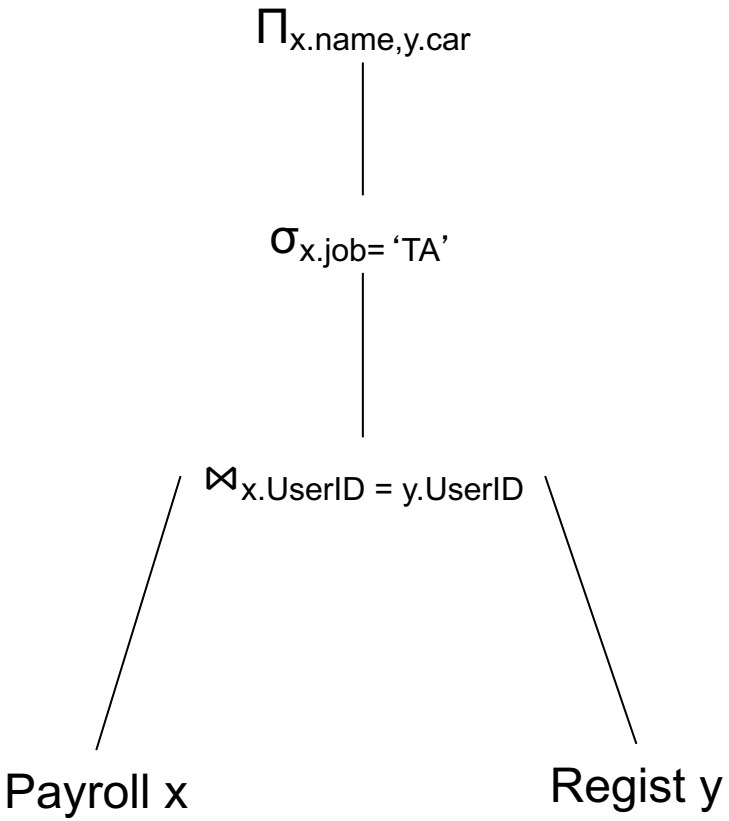
```
a+b=b+a                 // commutativity
a*(b+c)=a*b+a*c         // distributivity
etc, etc
```

- An optimize applies a set of rewrite rules in order to find the cheapest plan

Rewrite Rules: Predicate Pushdown

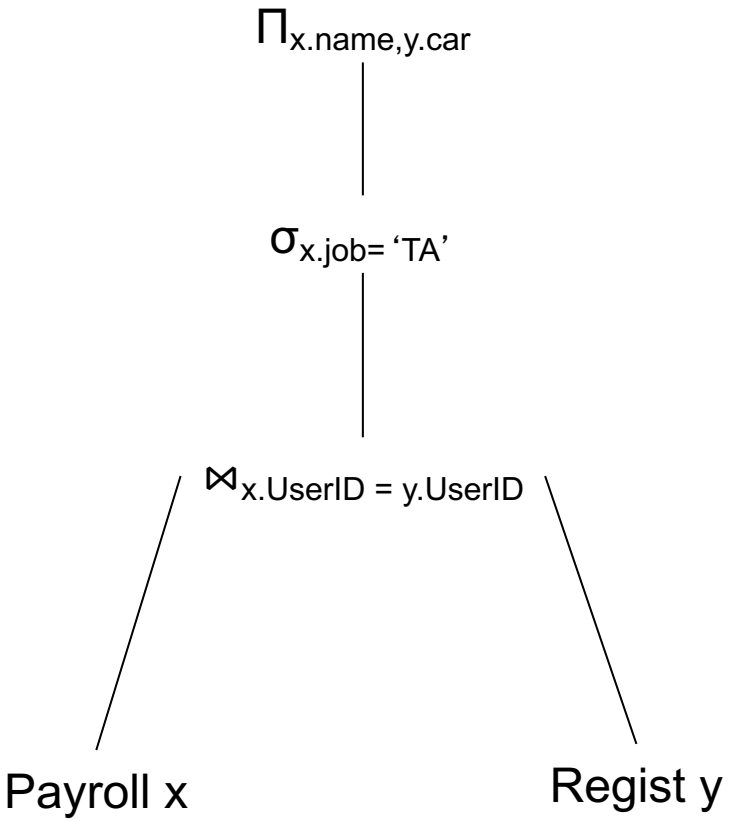
```
SELECT x.name,y.car
FROM Payroll x, Regist y
WHERE x.UserID = y.UserID
 and x.job='TA';
```

Rewrite Rules: Predicate Pushdown

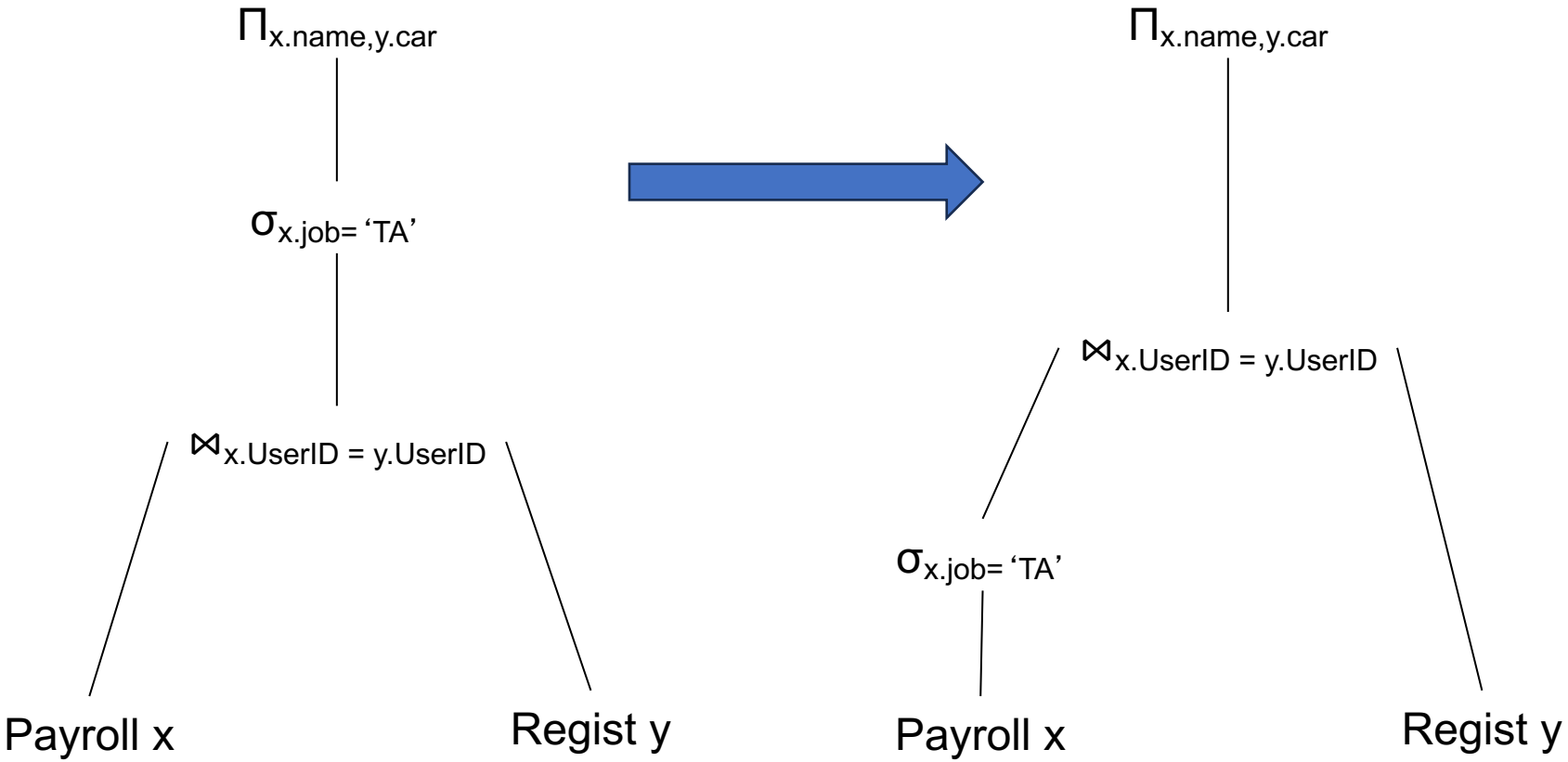



```
SELECT x.name,y.car
FROM Payroll x, Regist y
WHERE x.UserID = y.UserID
 and x.job='TA';
```

Rewrite Rules: Predicate Pushdown

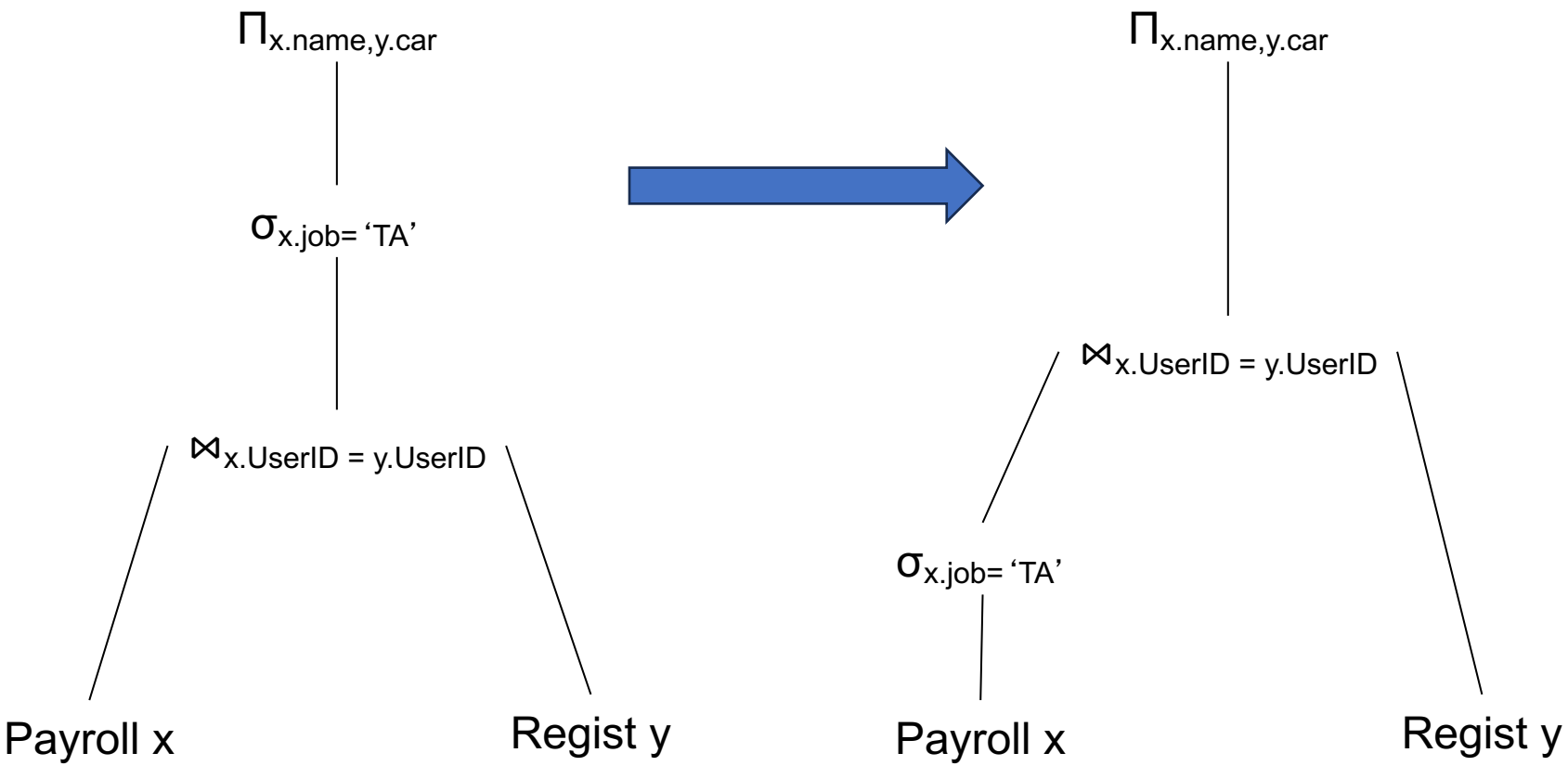


Rewrite Rules: Predicate Pushdown

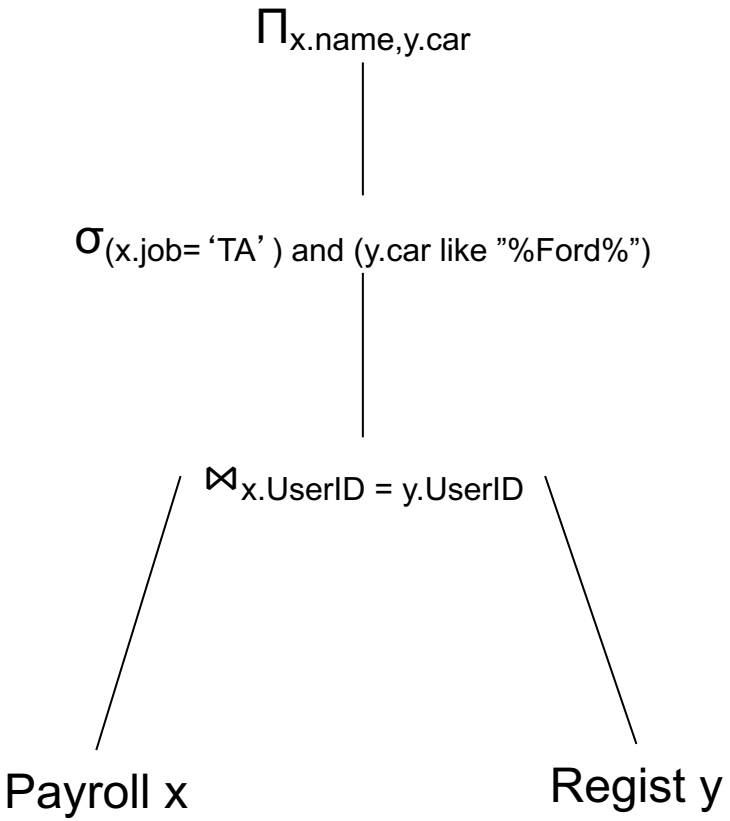


Rewrite Rules: Predicate Pushdown

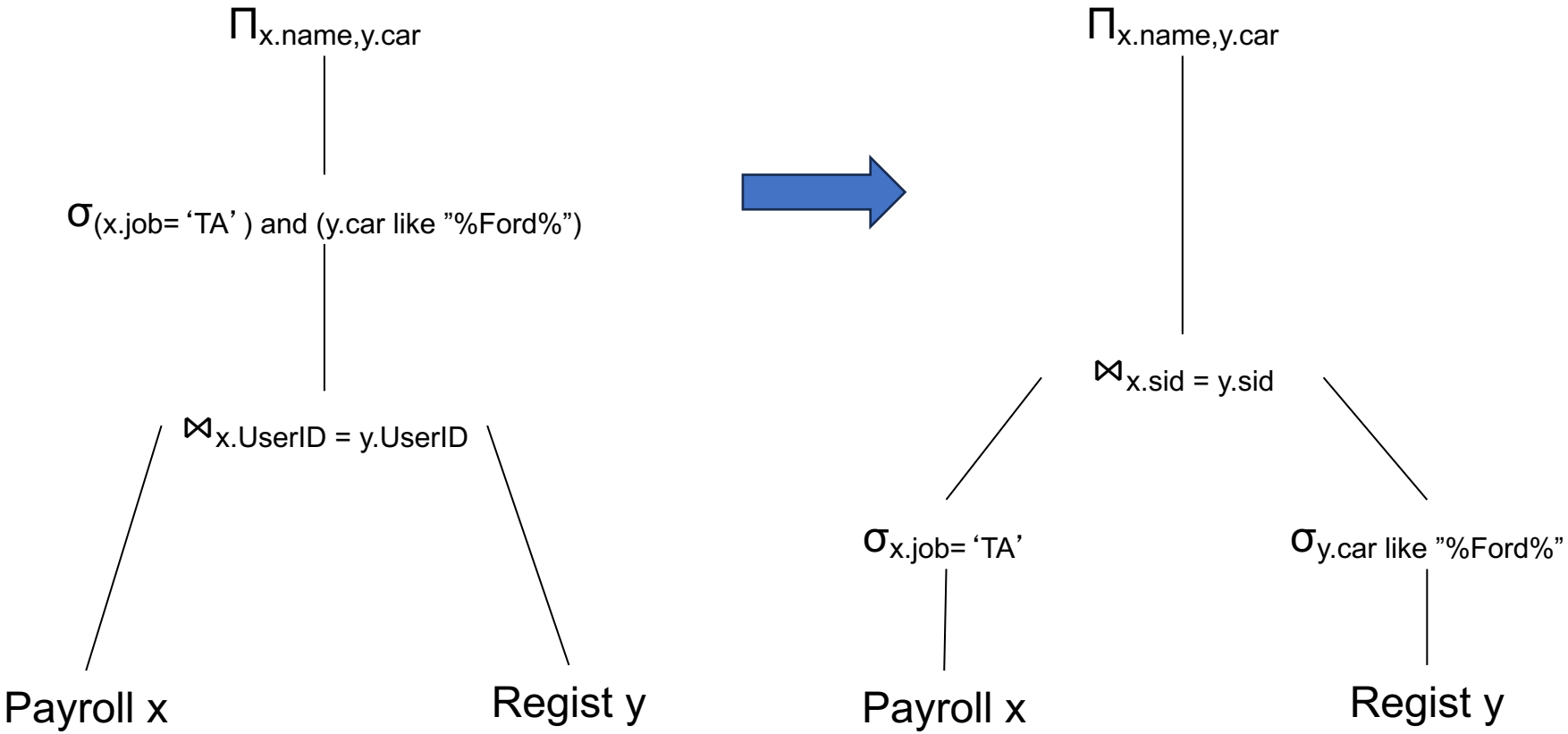
$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$ when C refers only to R



Rewrite Rules: Predicate Pushdown

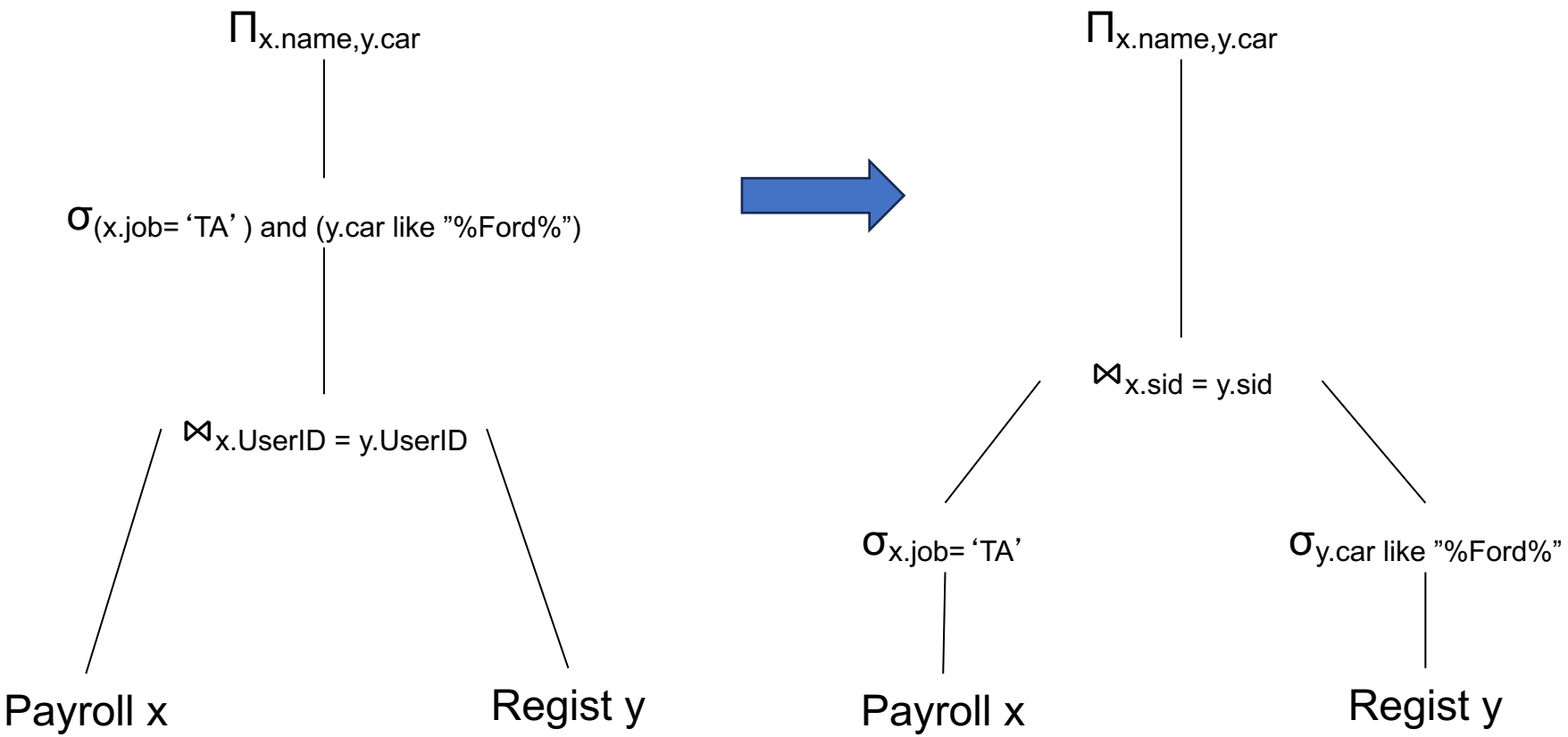


Rewrite Rules: Predicate Pushdown

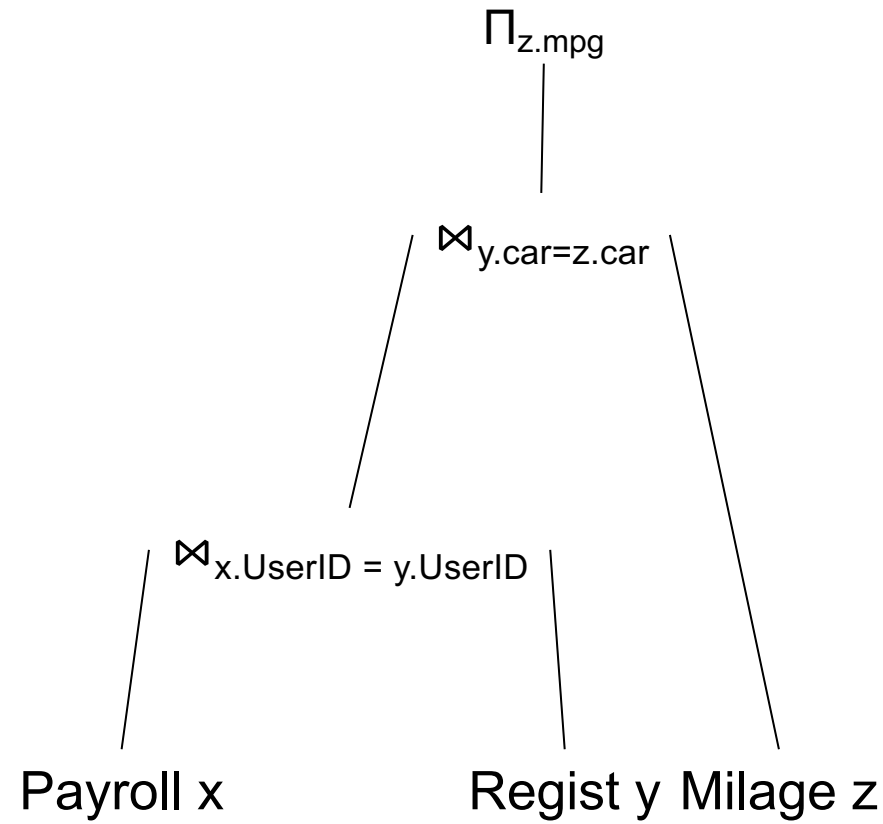


Rewrite Rules: Predicate Pushdown

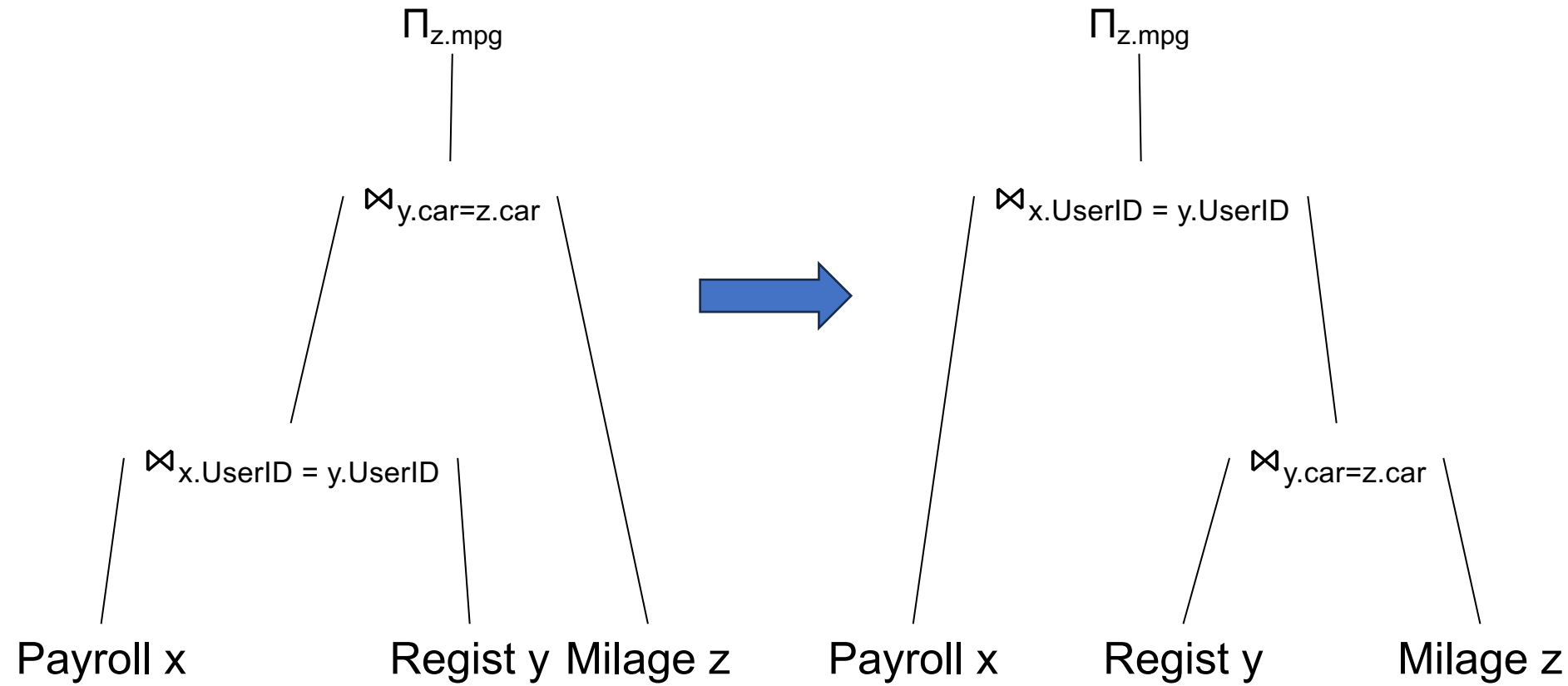
$$\sigma_{C1 \text{ and } C2}(R \bowtie S) = \sigma_{C1}(\sigma_{C2}(R \bowtie S))$$



Rewrite Rules: Join Associativity

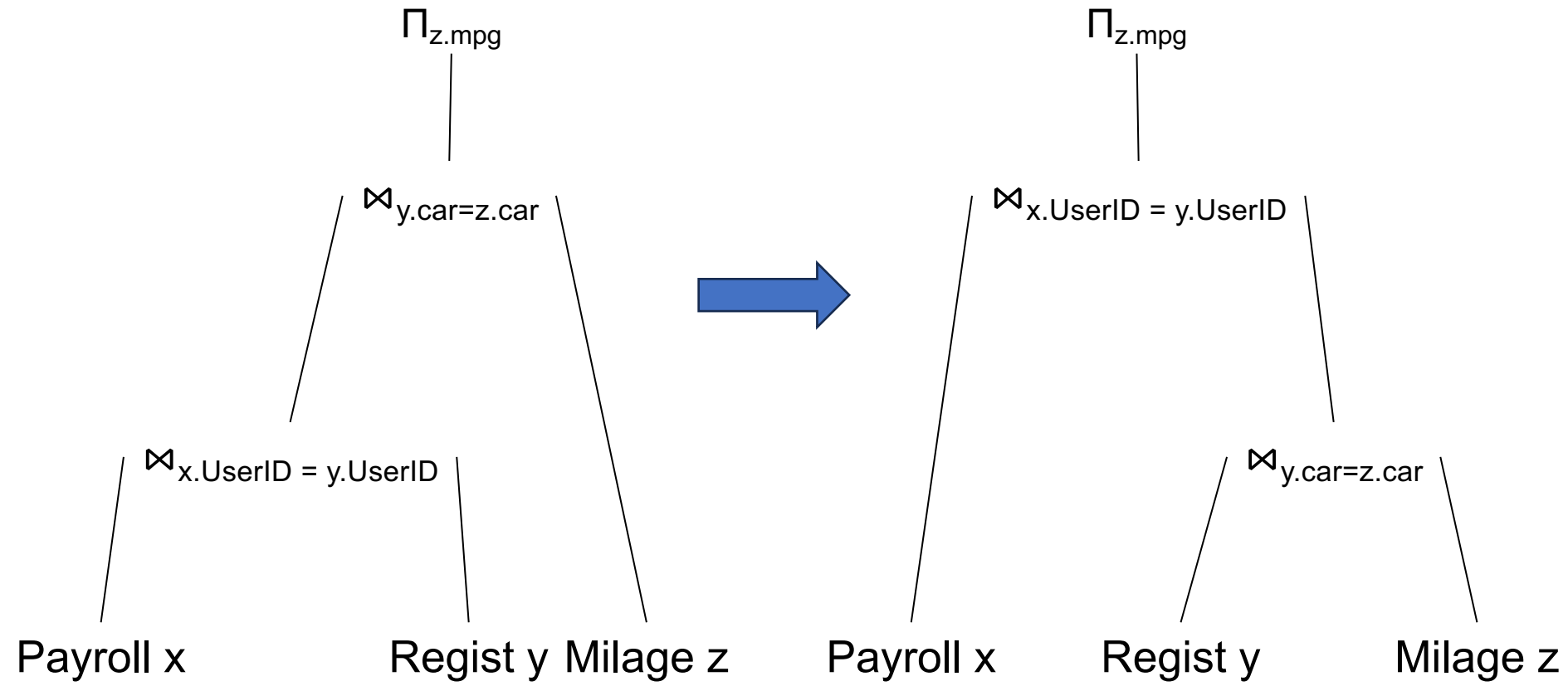


Rewrite Rules: Join Associativity



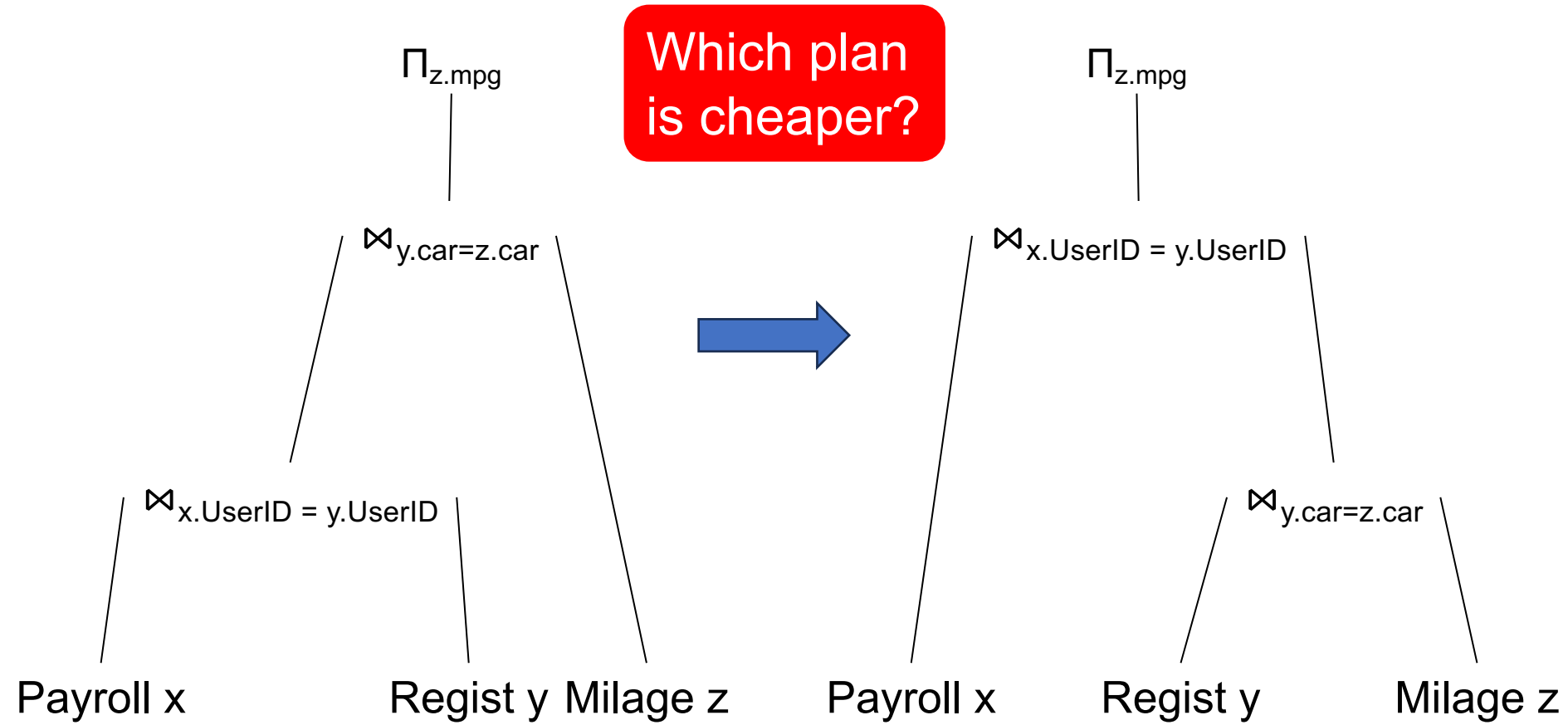
Rewrite Rules: Join Associativity

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



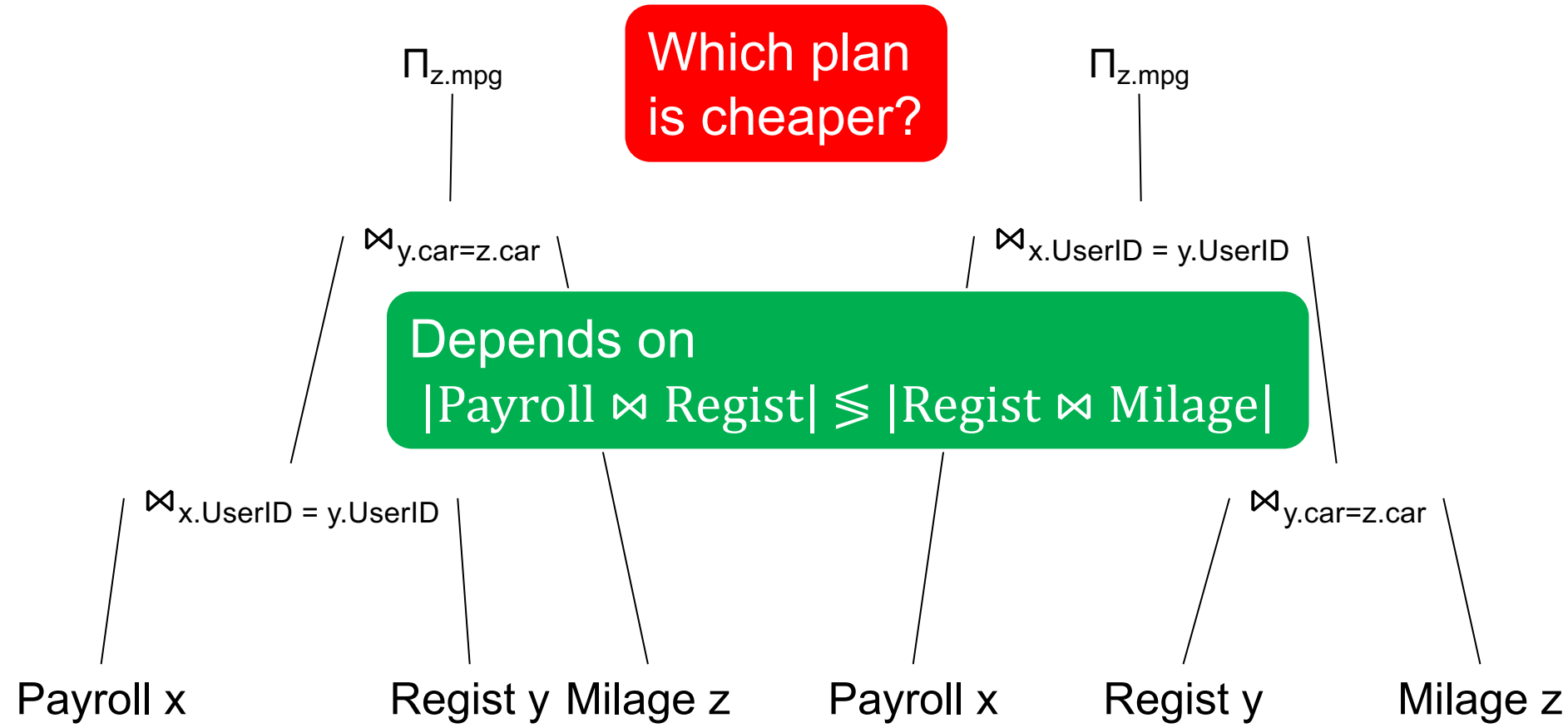
Rewrite Rules: Join Associativity

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

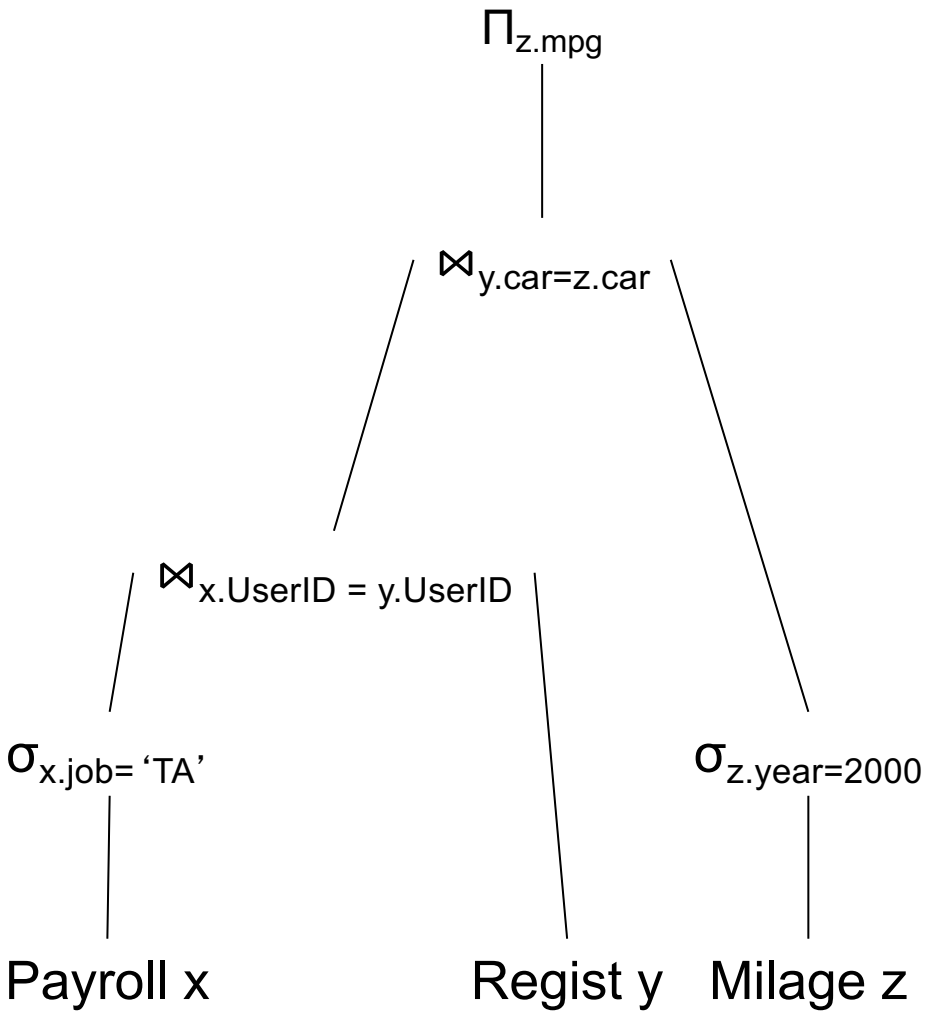


Rewrite Rules: Join Associativity

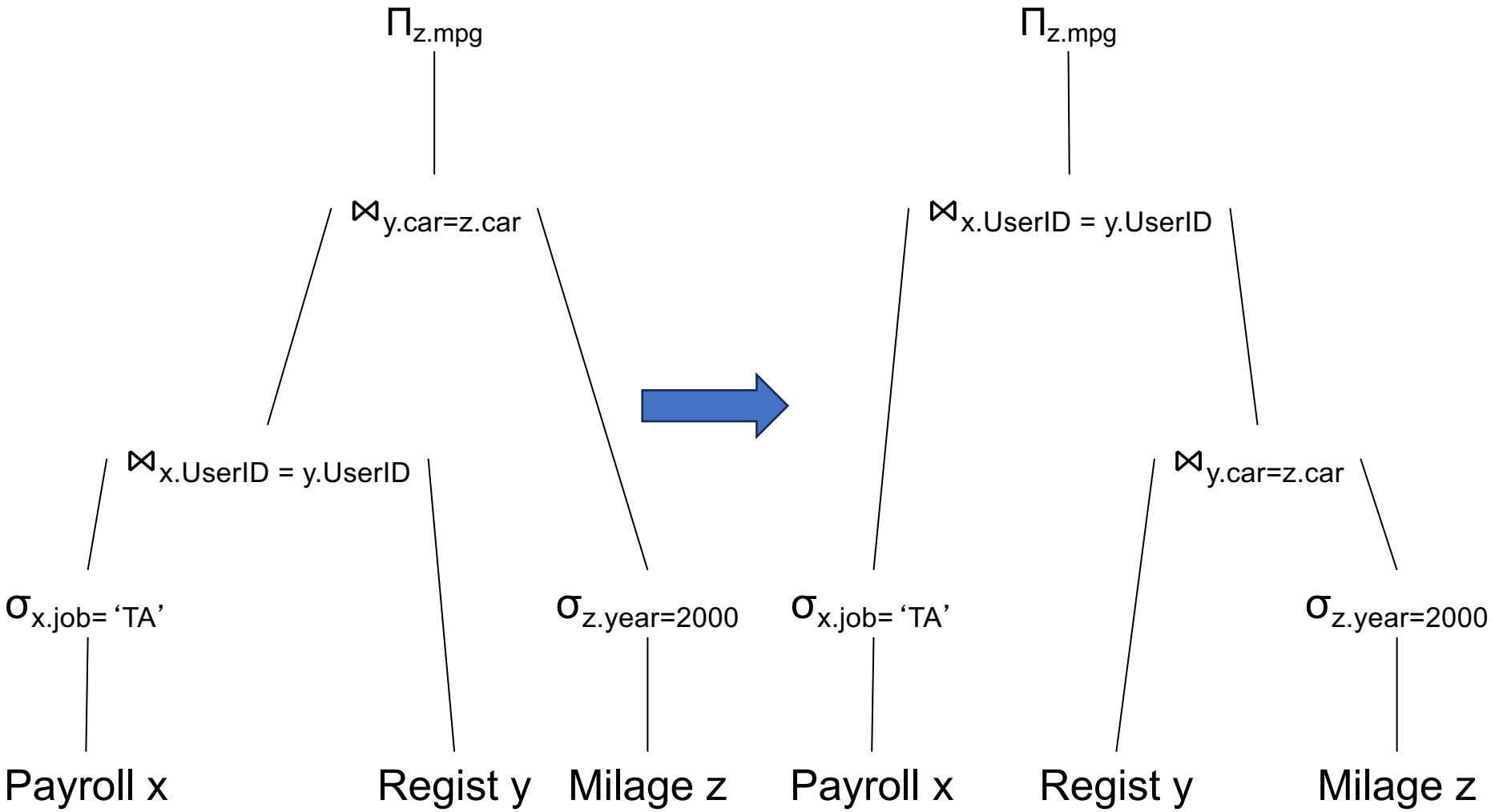
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$



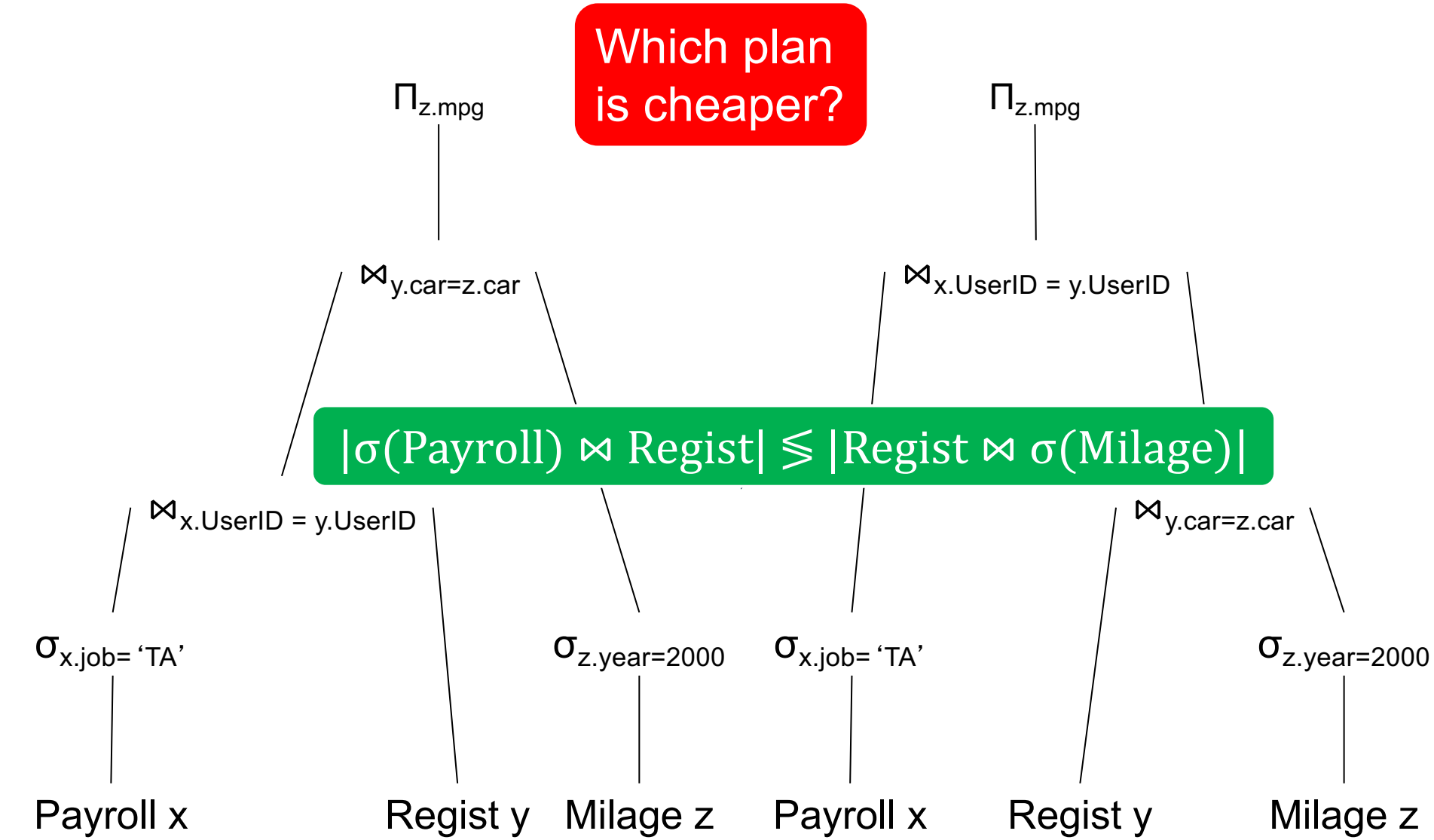
Rewrite Rules: Join Associativity



Rewrite Rules: Join Associativity



Rewrite Rules: Join Associativity



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

Cardinality Estimation

Overview

- DBMS keeps stats for each relation R(A,B,C,…):
  - Cardinality of R: T(R)
  - Number of distinct values in R.A: V(R,A)
  - Similarly: V(R, B), V(R, C), …
  - Histograms (later)

- Cardinality estimation:
  Given only these stats, estimate output size.

Cardinality Estimation

Recursively on the query plan

- For a base table: Est(R) = T(R)

- For an operator:

$$Est(\sigma_{pred}(R)) = \theta_{pred} * Est(R)$$
$$Est(R \bowtie_{A=B} S) = \theta_{A=B} * Est(R) * Est(S)$$

θ is called the *selectivity factor*

Selectivity Factor

For $\sigma_{A=const}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

Selectivity Factor

For $\sigma_{A=const}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

T(Payroll) = 10000
V(Payroll, job) = 20

$EST\left[\sigma_{job='TA'}(Payroll)\right] =$??

Selectivity Factor

For $\sigma_{A=const}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

T(Payroll) = 10000
V(Payroll, job) = 20

$$EST\left[\sigma_{job='TA'}(Payroll)\right] = \frac{1}{20}10000 = 500$$

Selectivity Factor

For $\sigma_{p \text{ and } q}$ the sel. factor is $\theta_{p \text{ and } q} = \theta_p \times \theta_q$

Independence assumption

Selectivity Factor

For $\sigma_{p \text{ and } q}$ the sel. factor is $\theta_{p \text{ and } q} = \theta_p \times \theta_q$

Independence assumption

T(Payroll) = 10000
V(Payroll, job) = 20
V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{job}=\text{'TA'} \text{ and } \text{salary}=20000}(\text{Payroll})\right] = ??$$

Selectivity Factor

For $\sigma_{p \text{ and } q}$ the sel. factor is $\theta_{p \text{ and } q} = \theta_p \times \theta_q$

Independence assumption

T(Payroll) = 10000
V(Payroll, job) = 20
V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{job}=\text{'TA'} \text{ and } \text{salary}=20000}(\text{Payroll})\right] = \frac{1}{20}\frac{1}{50}10000 = 10$$

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R)T(S)$

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R)T(S)$

T(Payroll) = 10000
V(Payroll, UserID) = 10000

T(Regist) = 3000
V(Regist, UserID) = 2000

$EST[Payroll \bowtie Regist] =$??

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R)T(S)$

T(Payroll) = 10000
V(Payroll, UserID) = 10000

T(Regist) = 3000
V(Regist, UserID) = 2000

$$EST[Payroll \bowtie Regist] = \frac{1}{\max(10000,2000)} 10000 \cdot 3000 = 3000$$

Summary of Assumptions

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

Computing the Cost of a Plan

- Estimate cardinalities bottom-up
- Estimate cost by using estimated cardinalities

Histograms

- Histogram on R.A refines T(R), V(R,A)
- Each bucket contains: T(bucket), V(bucket, A)

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\mathrm{EST}\left[\sigma_{\mathrm{salary}=25\mathrm{k}}(\mathrm{Payroll})\right] = ???$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50} 10000 = 10$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50}10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50} 10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50}10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = \frac{1}{10}320 = 32$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50}10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = \frac{1}{10}320 = 32$$

A better estimate

Recap: How a Query Engine Works

- Each RA operator has multiple physical operators
- Indexes: speed up some queries slow down others
- Rewrite rules transform one query plan to another
- Cardinality estimators used to estimate the cost; they are often wrong