CSE544
Data Management

Lectures 7-8

Query Optimization

Announcements

- HW2 due on Friday
- HW3
  - Main option (mandatory for CSE): SimpleDB
  - Alternate: data cleaning (a bit open ended)

An Intermediate Format: PAX

- PAX = Partition Attributes Across
- Addresses memory access bottleneck (not the disk bottleneck)

From Row to Column Storage (Initial Designs - 1985)

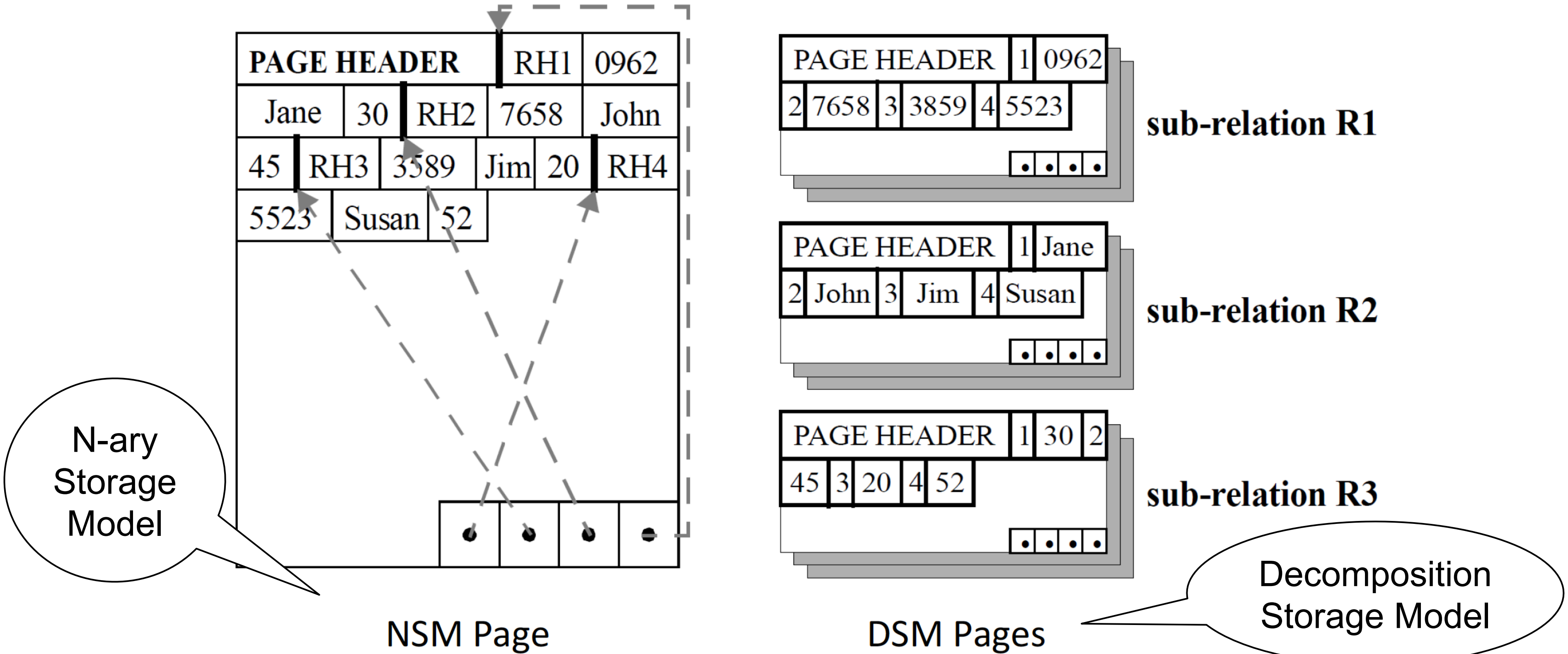


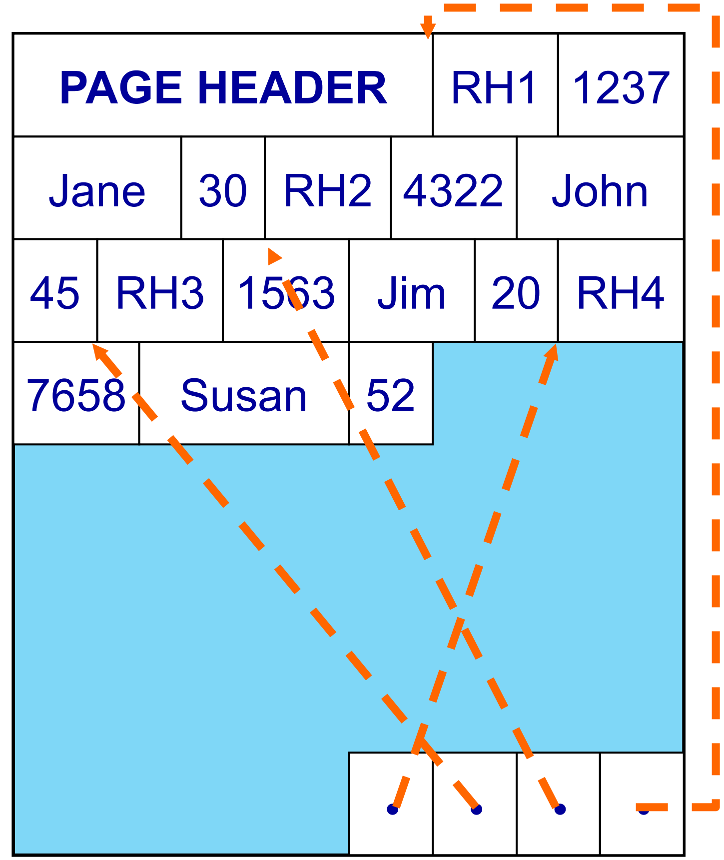
Figure 2.1: Storage models for storing database records inside disk pages: NSM (row-store) and DSM (a predecessor to column-stores). Figure taken from [5].

Current Scheme: Slotted Pages

Formal name: NSM (N-ary Storage Model)

R

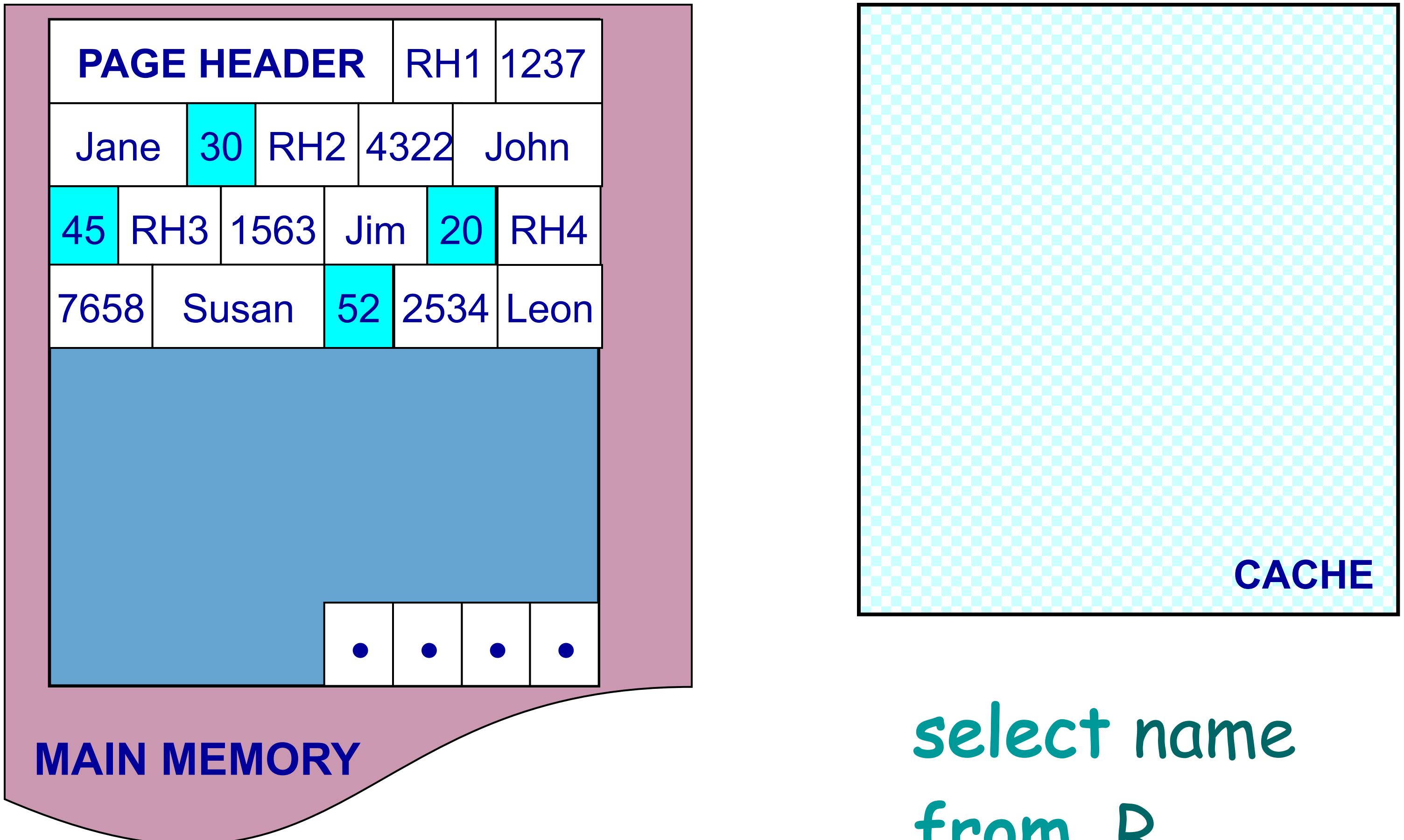
RID	SSN	Name	Age
1	1237	Jane	30
2	4322	John	45
3	1563	Jim	20
4	7658	Susan	52
5	2534	Leon	43
6	8791	Dan	37



- Records are stored sequentially
- Offsets to start of each record at end of page

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

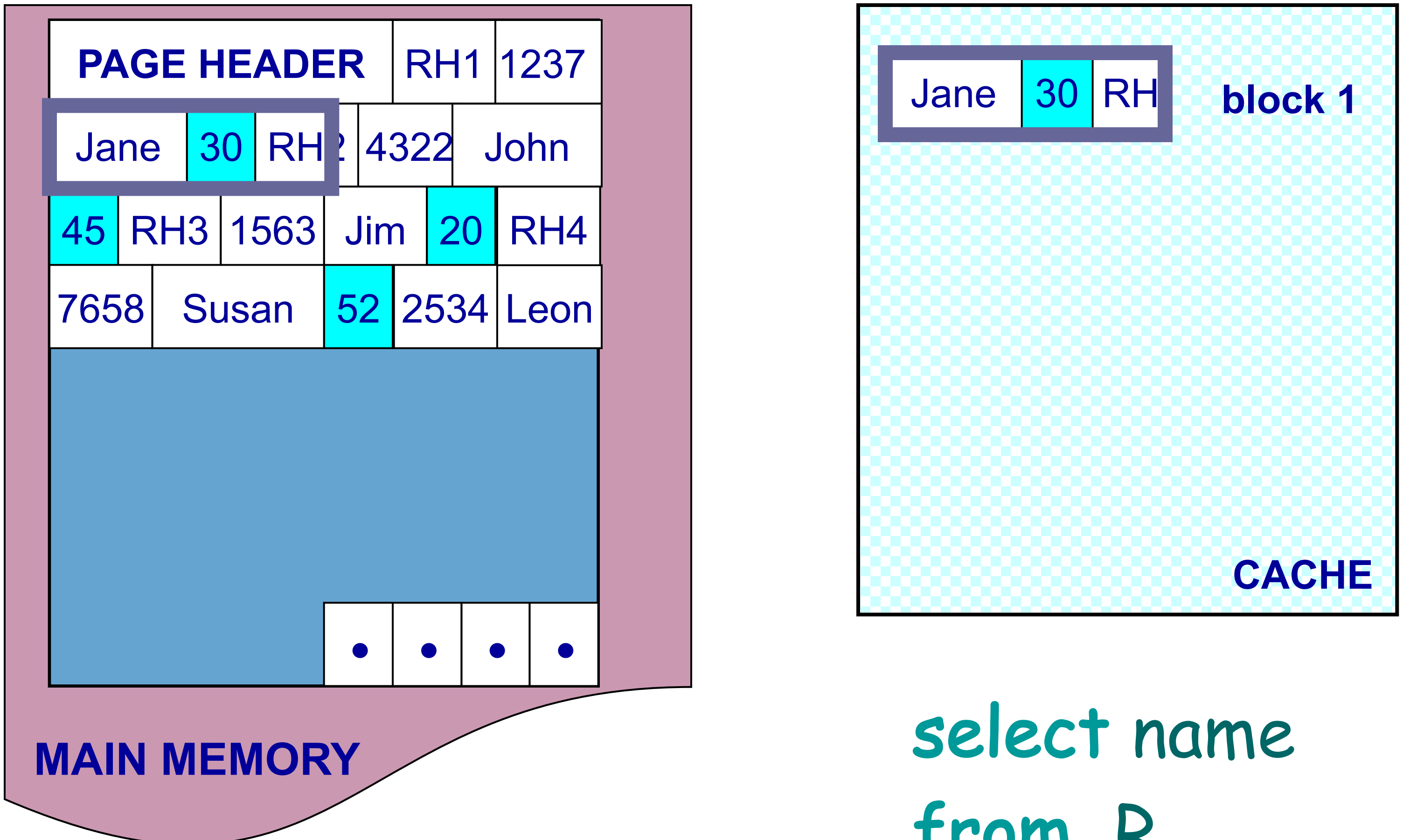


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

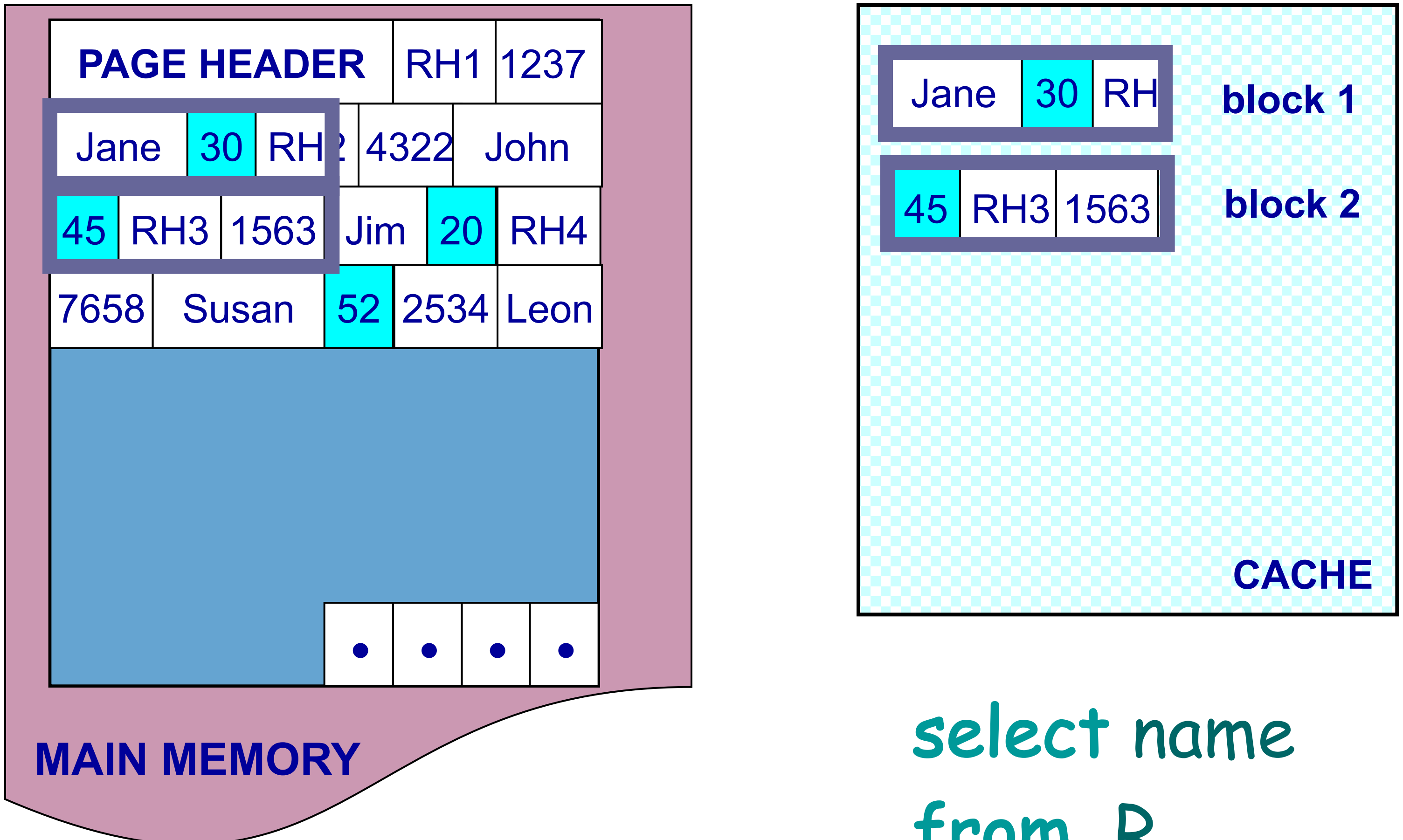


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

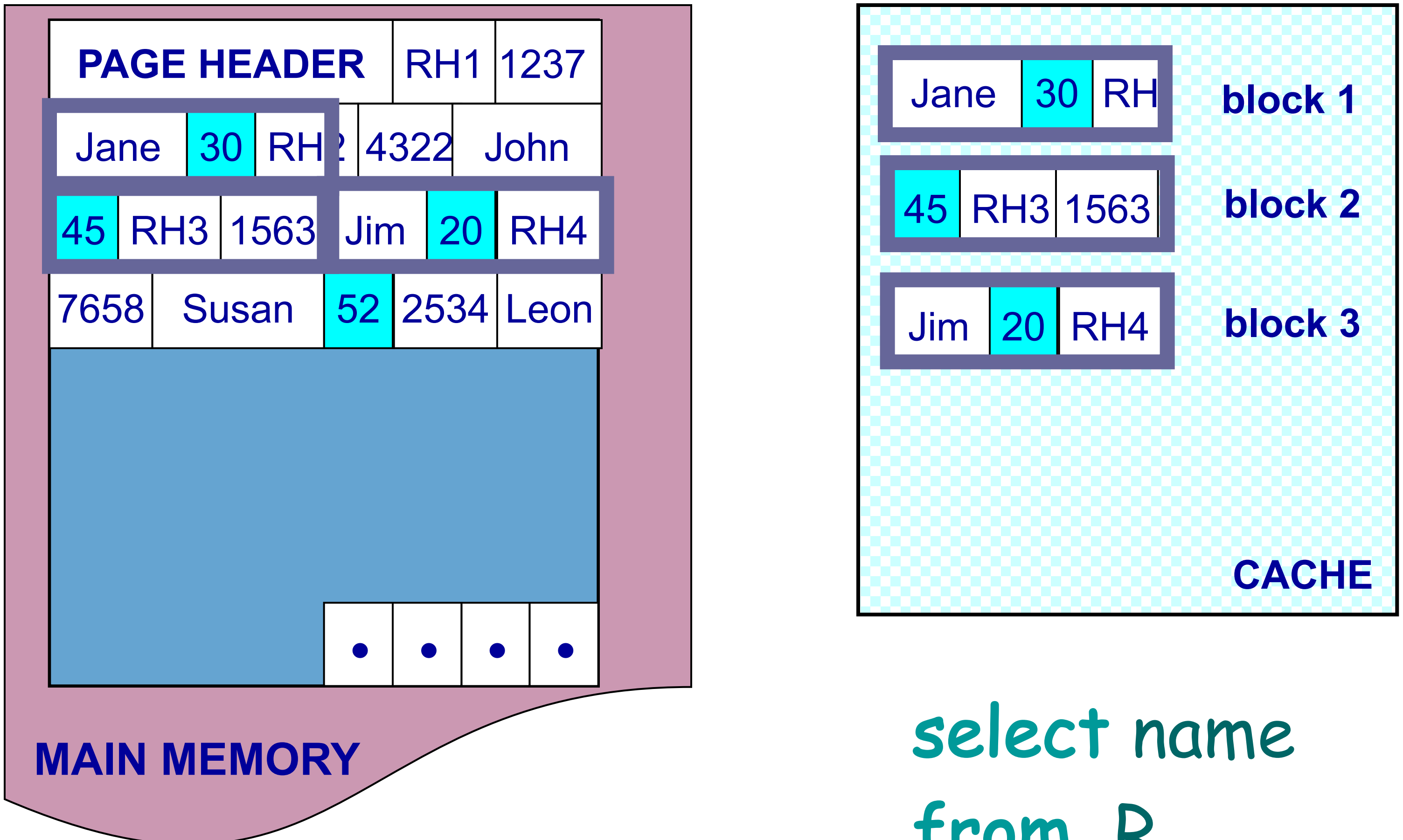


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM

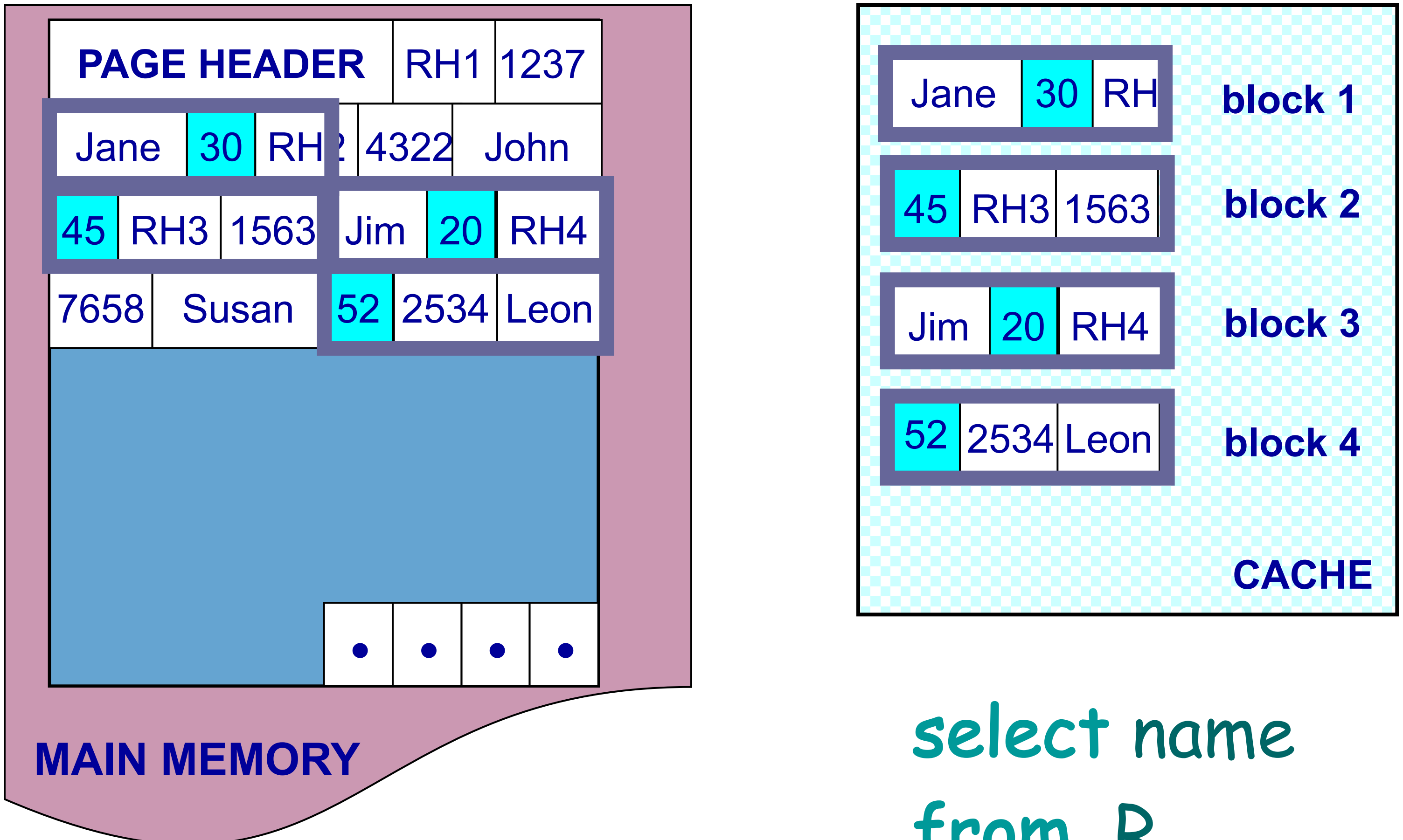


select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using NSM



select name
from R
where age > 50

NSM pushes non-referenced data to the cache

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Need New Data Page Layout

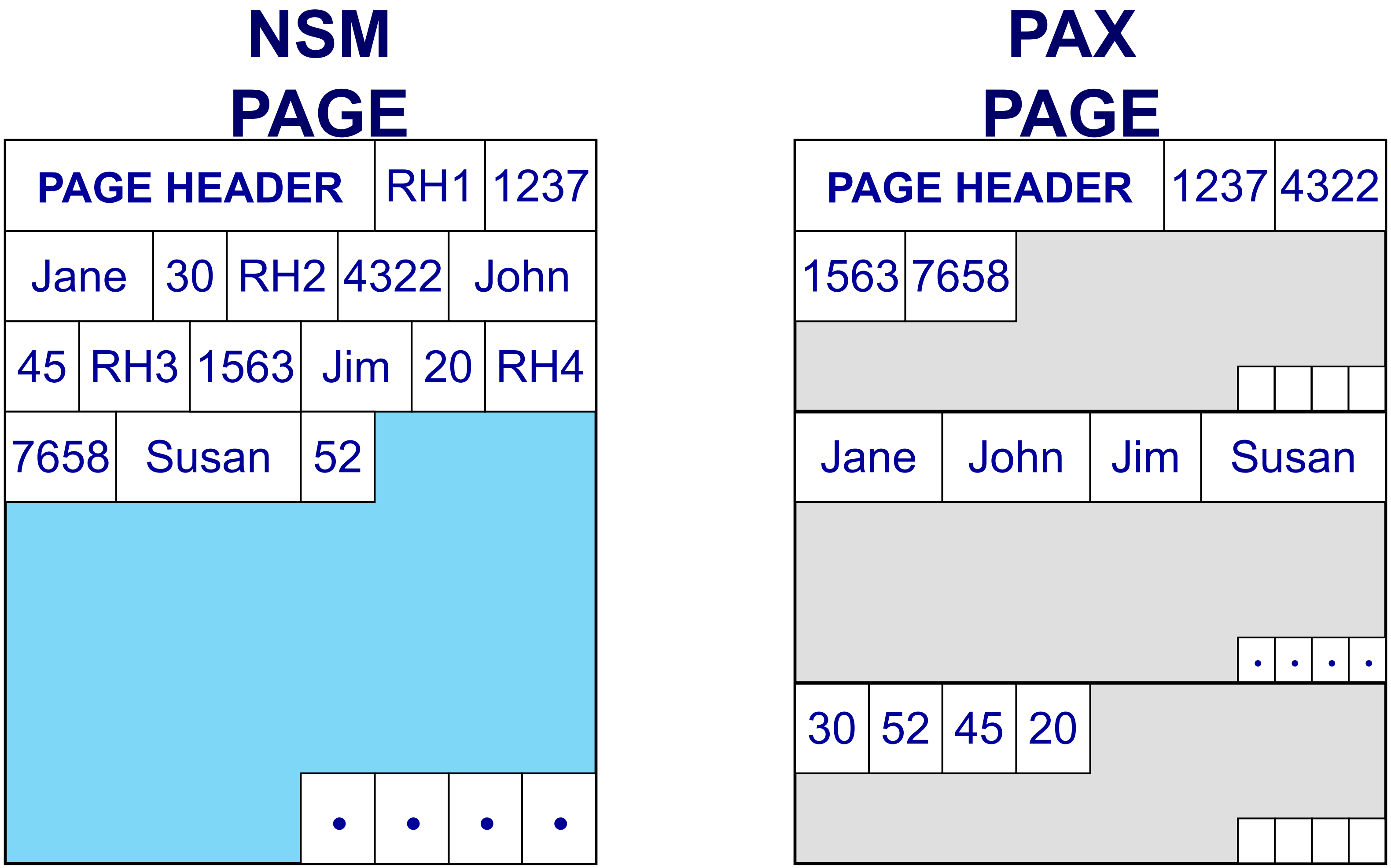
- Eliminates unnecessary memory accesses
- Improves inter-record locality
- Keeps a record's fields together
- Does not affect I/O performance

and, most importantly, is…

low-implementation-cost, high-impact

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

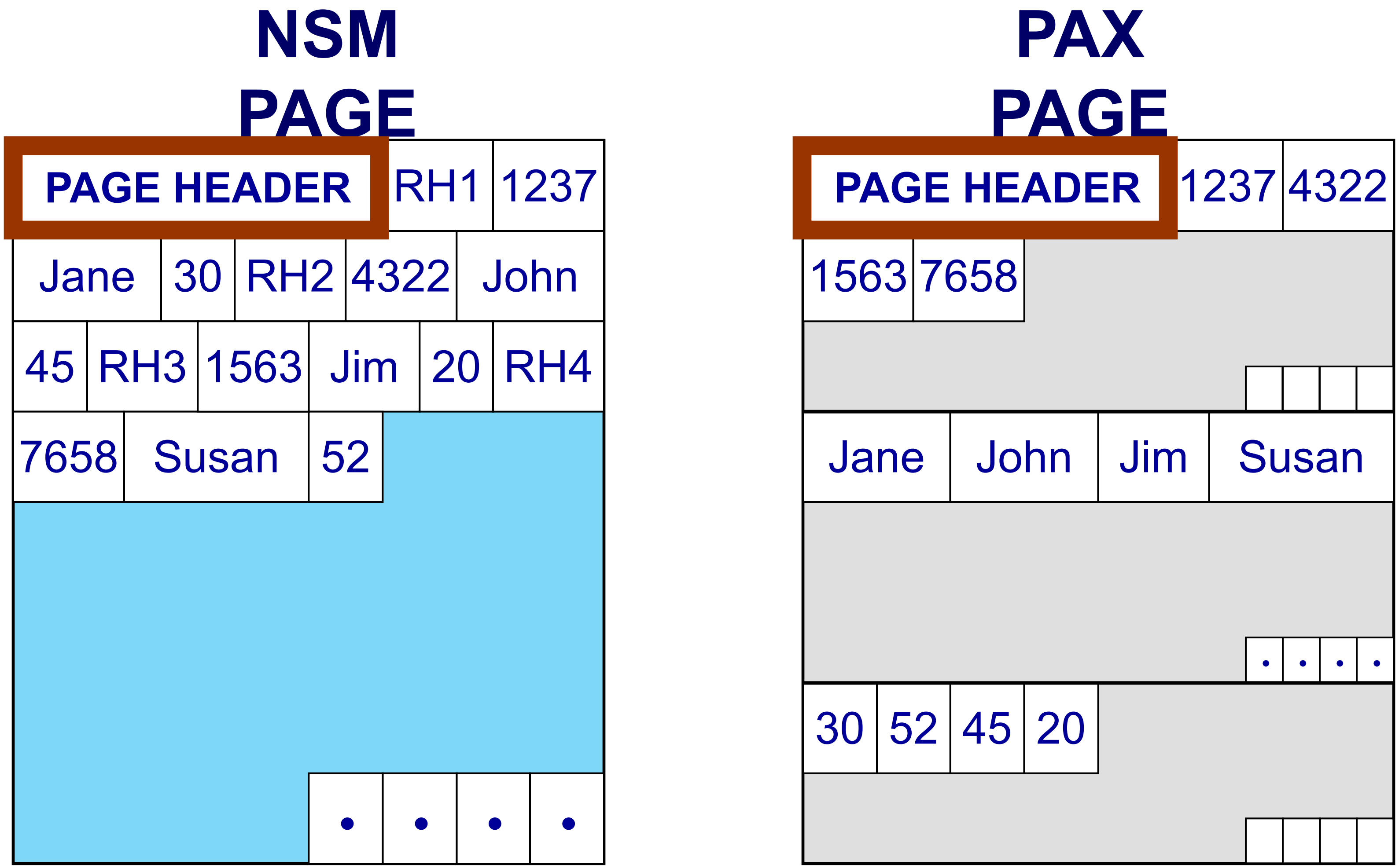
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

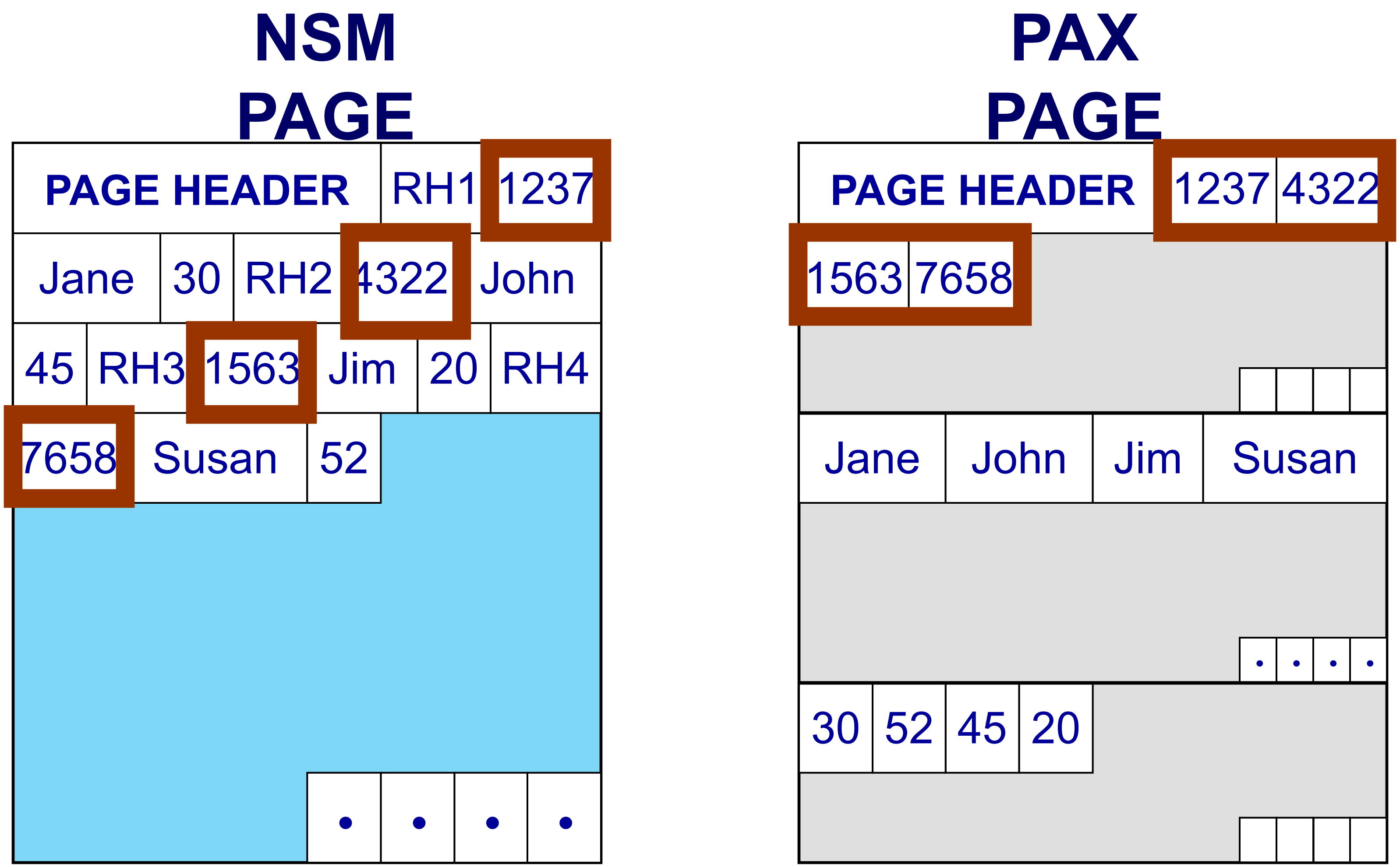
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

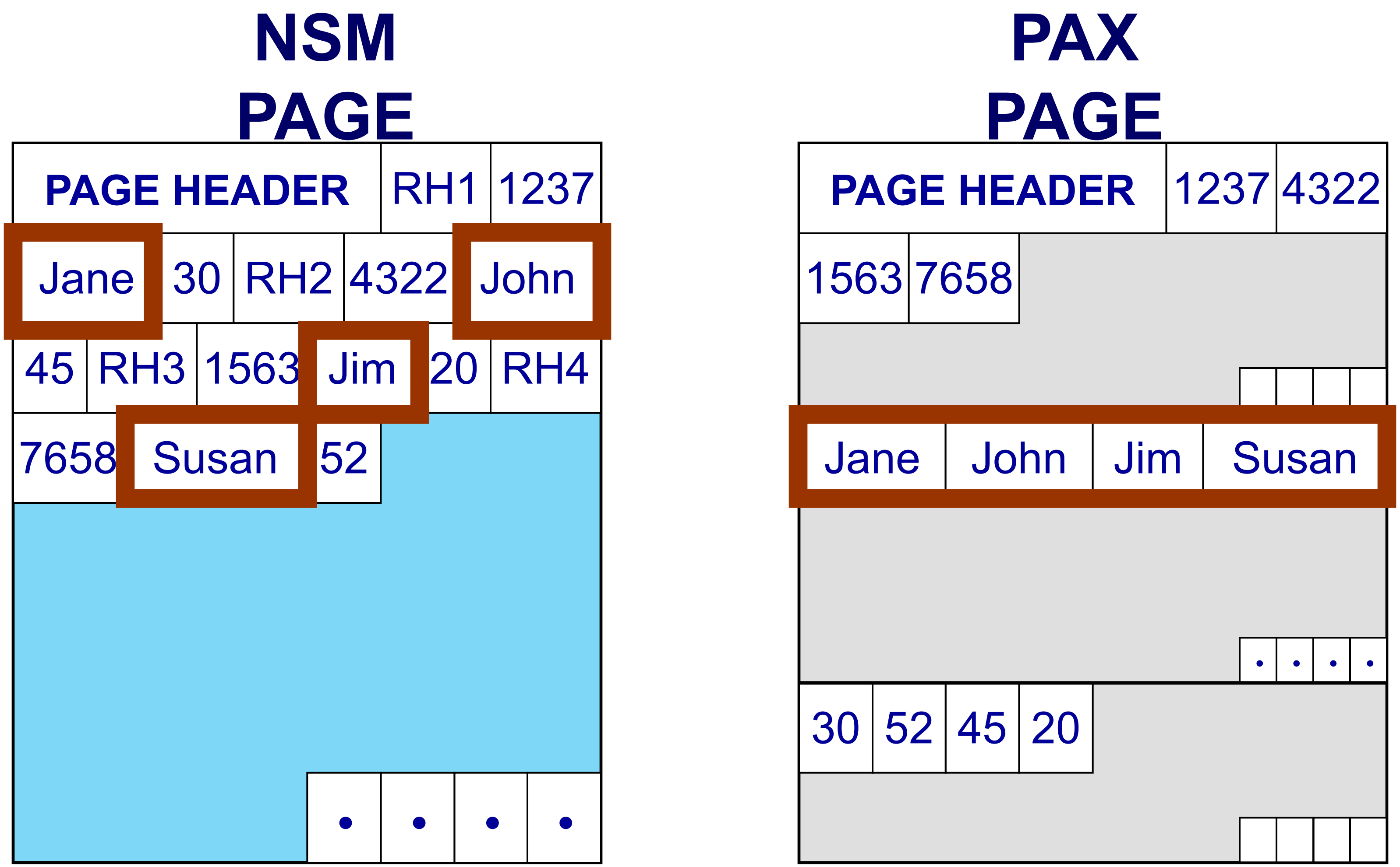
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

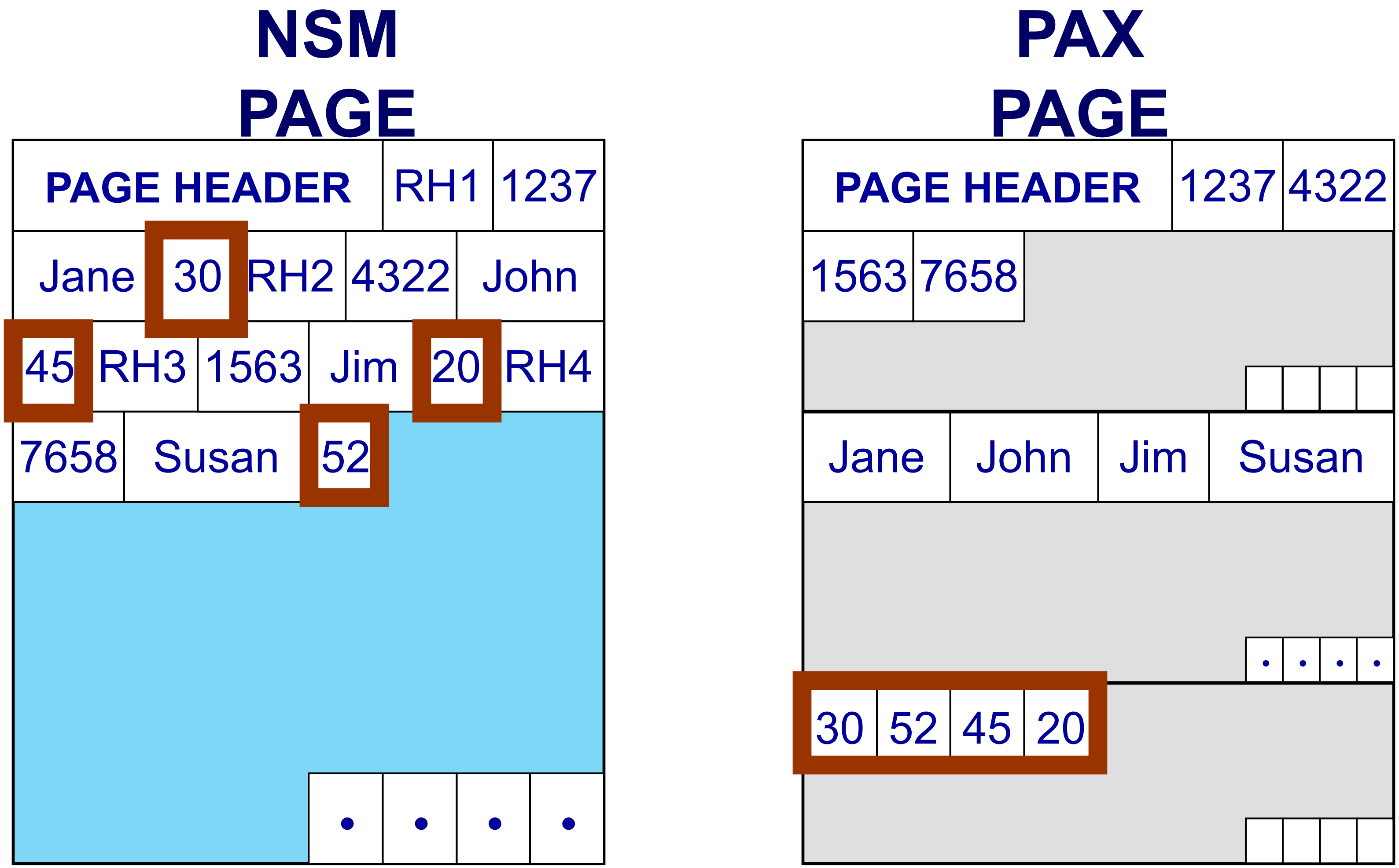
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

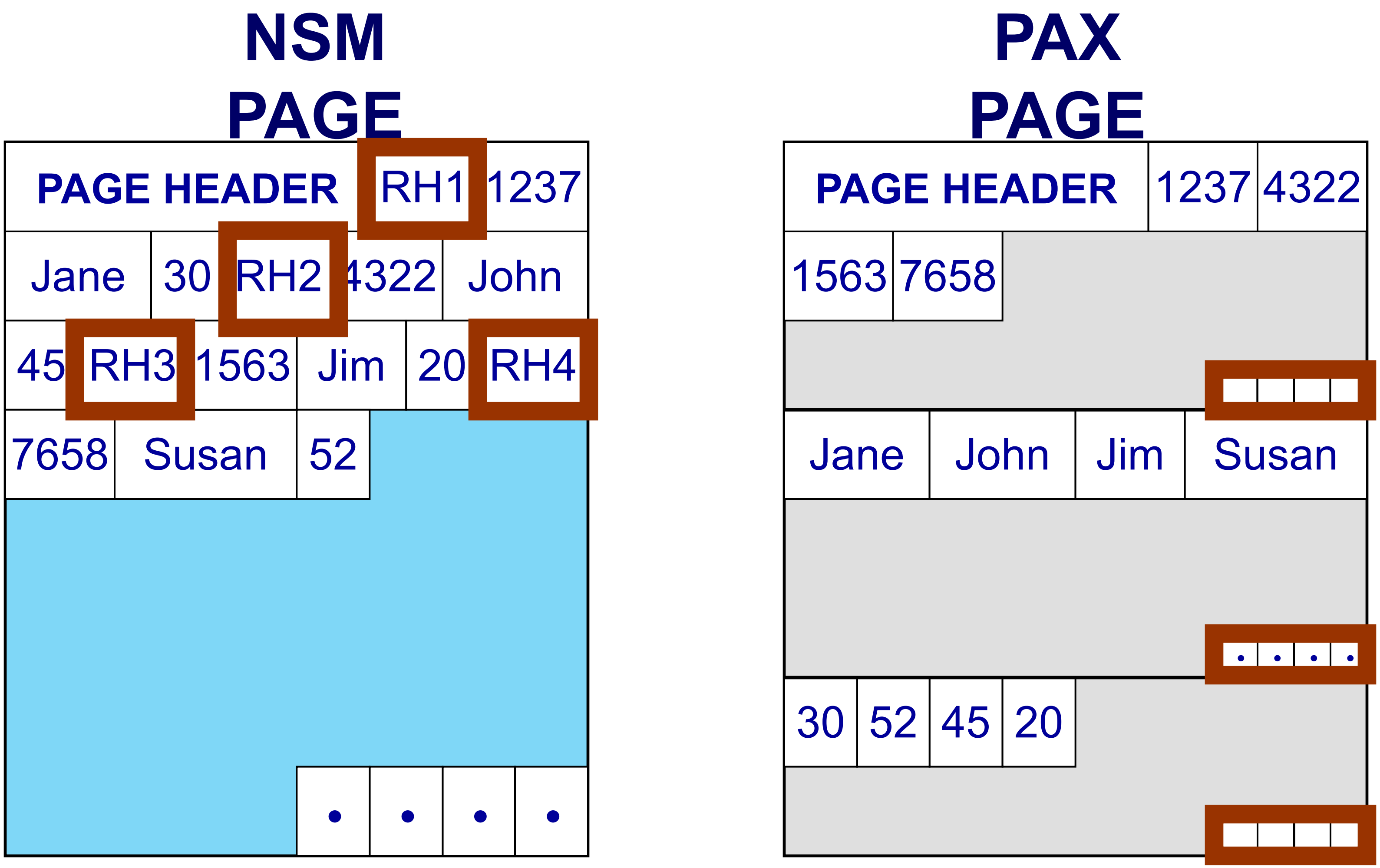
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

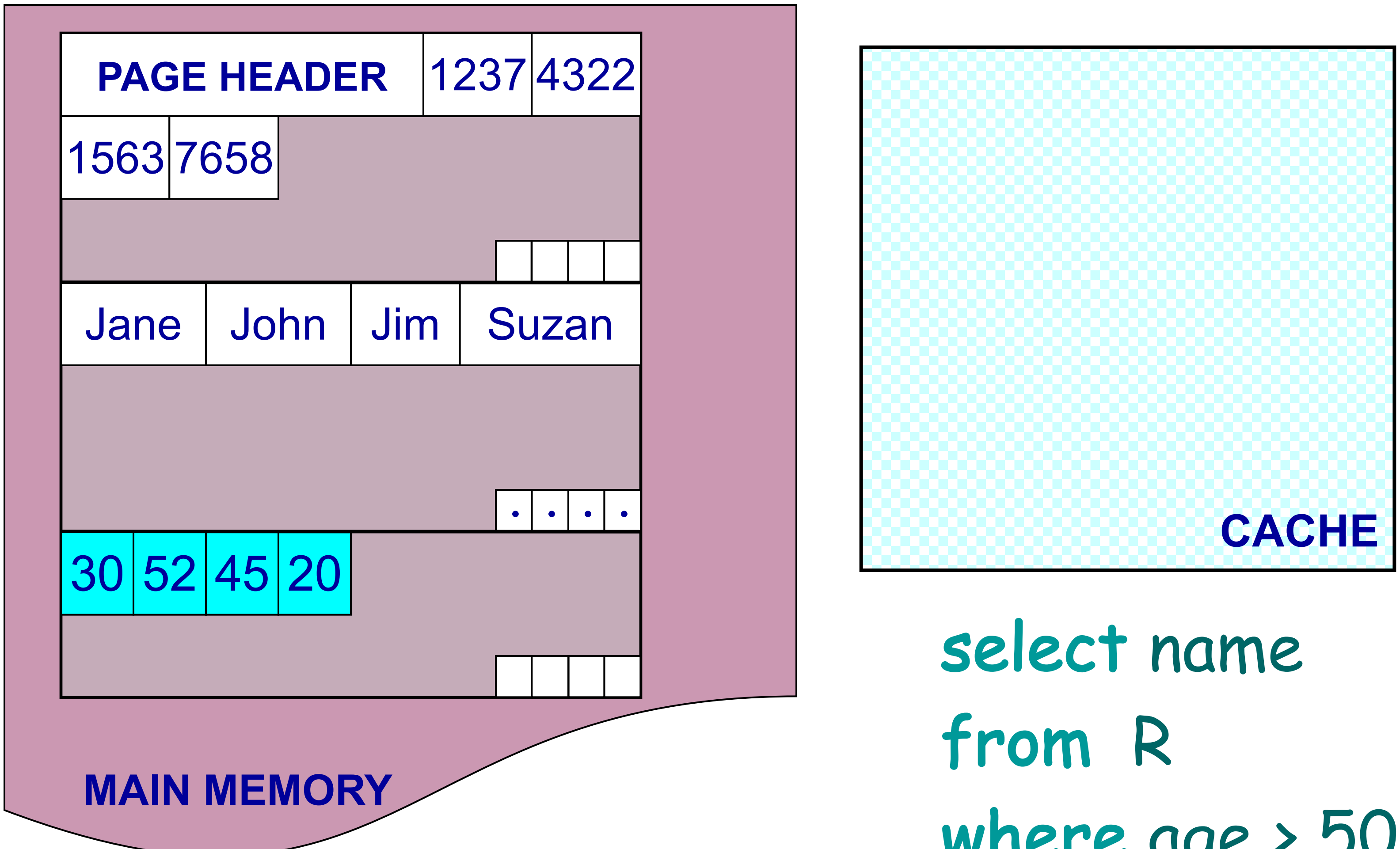
Partition Attributes Across (PAX)



Partition data *within* the page for spatial locality

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using PAX

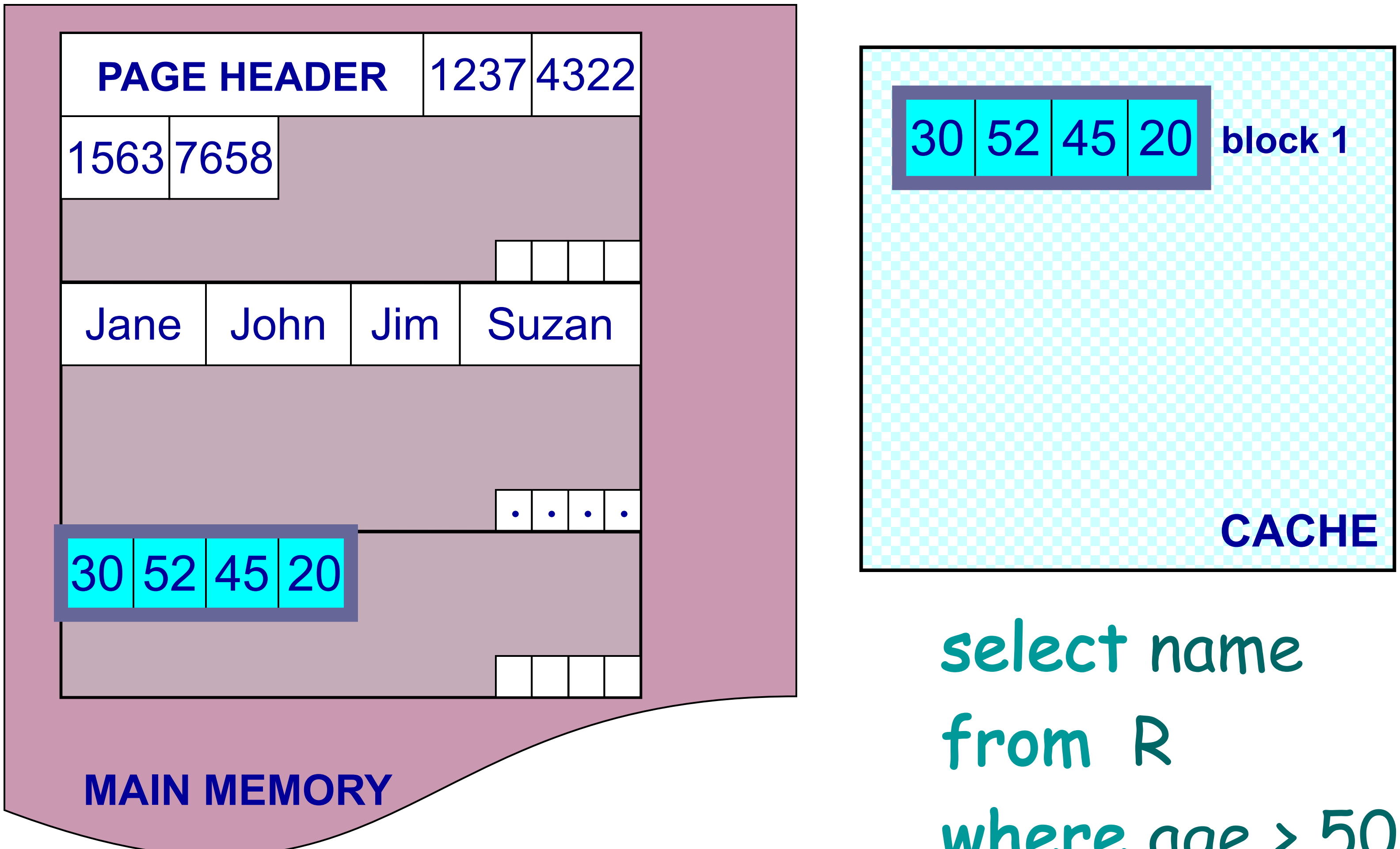


select name
from R
where age > 50

Fewer cache misses, low reconstruction cost

Ailamaki VLDB’01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

Predicate Evaluation using PAX

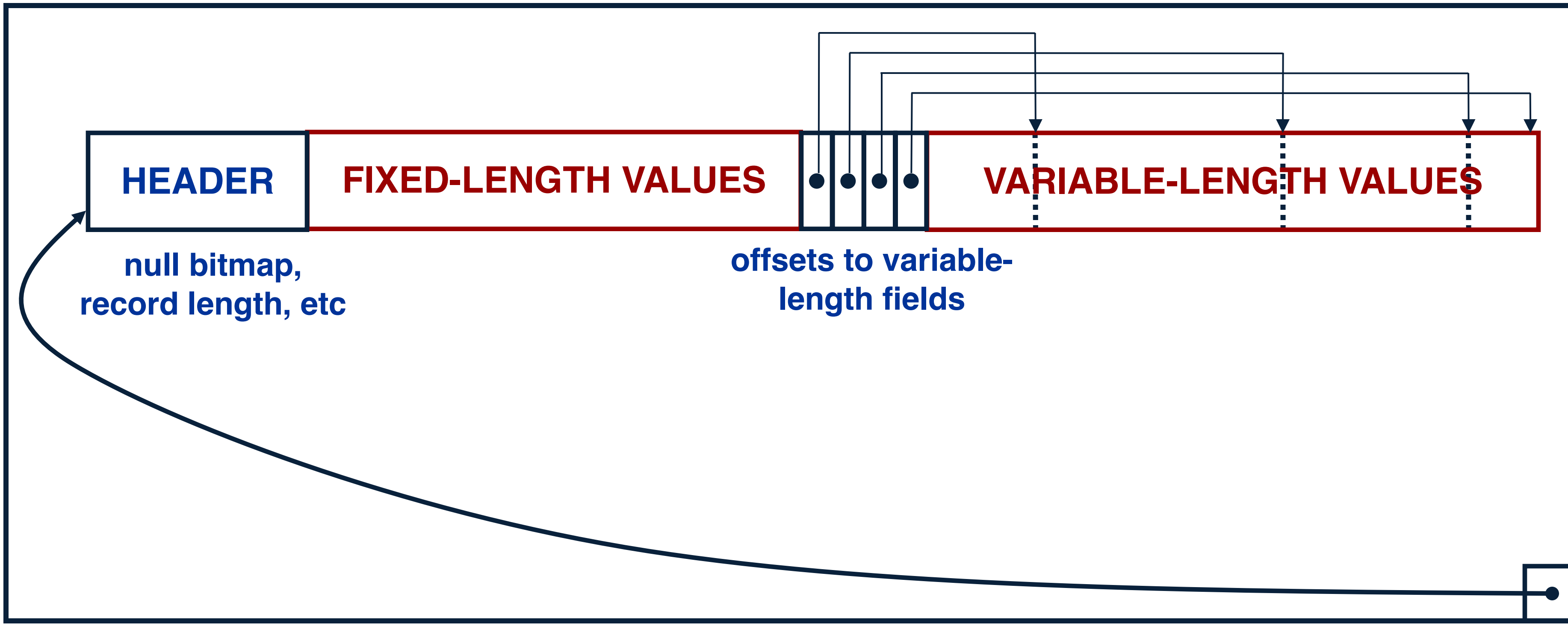


select name
from R
where age > 50

Fewer cache misses, low reconstruction cost

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

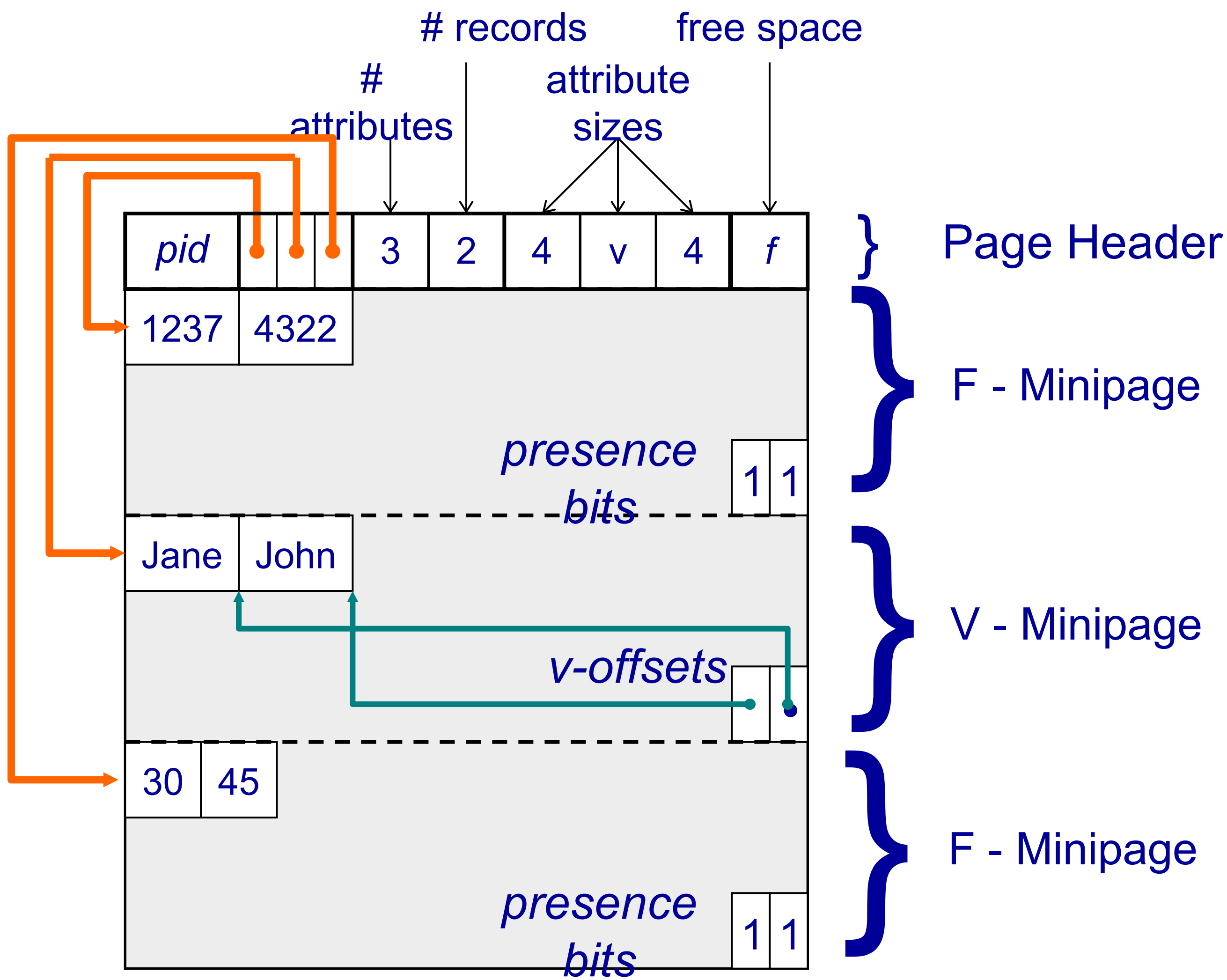
A Real NSM Record



NSM: All fields of record stored together + slots

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

PAX: Detailed Design



PAX: Group fields + amortizes record headers

Ailamaki VLDB'01 http://research.cs.wisc.edu/multifacet/papers/vldb01_pax_talk.ppt

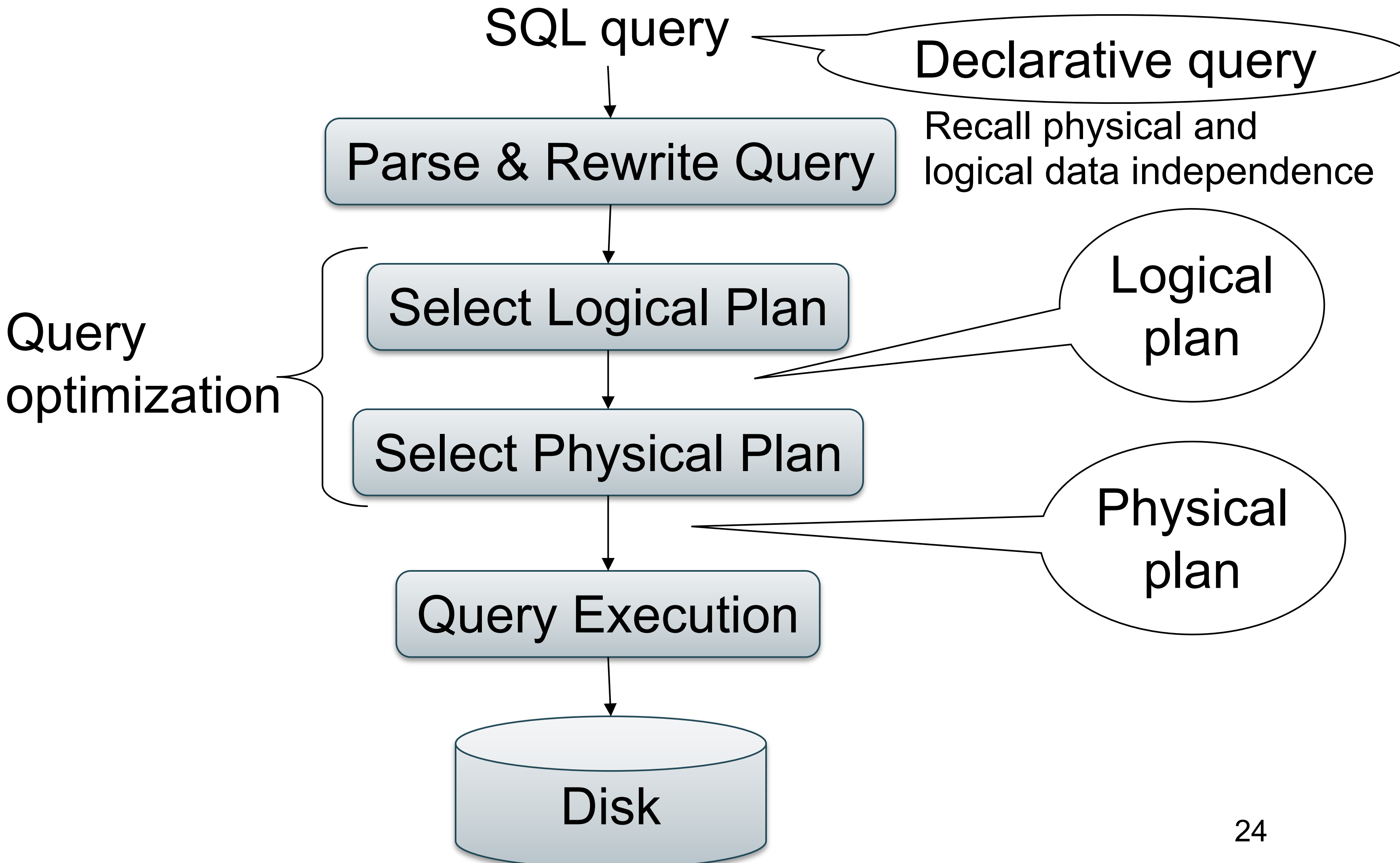
PAX - Summary

- Improves processor cache locality
- Does not affect I/O behavior
  - Same disk accesses for NSM or PAX storage
  - No need to change the buffer manager

- Today:
  - Most engines use a PAX layout of the disk
  - Beyond disk: Snowflake partitions tables horizontally into files, then uses column-store inside each file (hence, PAX)

Query Optimization

Query Optimization Motivation



Query Optimization

Three major components:

1. Search space
   – Access path selection
   – Rewrite rules

2. Cardinality and cost estimation

3. Plan enumeration algorithms

Access Path

Access path: implements a selection $\sigma_P(R)$,
Note: P is called *search argument* SARG

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

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

- B+-tree on sid; clustered
- B+-tree on scity; unclustered

Which access path should we use?

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use?

1. Sequential scan: cost = 100

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use?

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70

Supplier(sid,sname,scity,sstate)

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use?

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70
3. Index scan on scity: cost = 1000/20 = 50

Rewrite Rules

Search space is defined by the set of rewrite rules that the optimizer implements

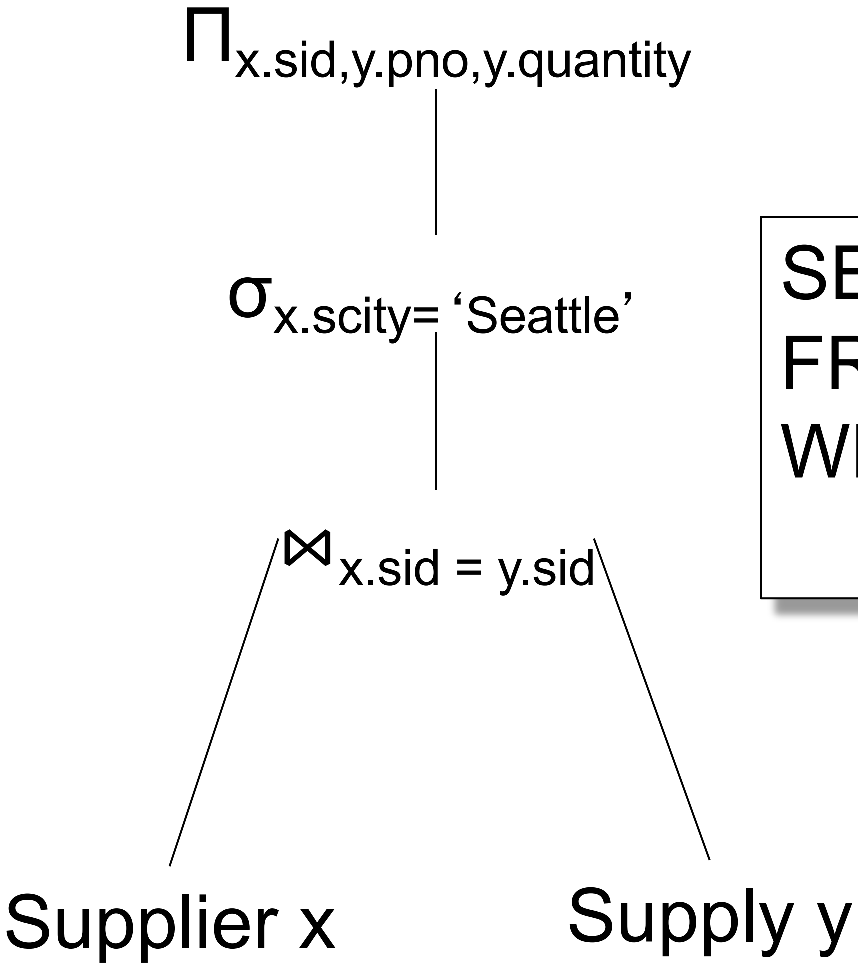
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Example Optimization

```
SELECT x.sid, y.pno, y.quantity
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
      and x.scity = 'Seattle'
```

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

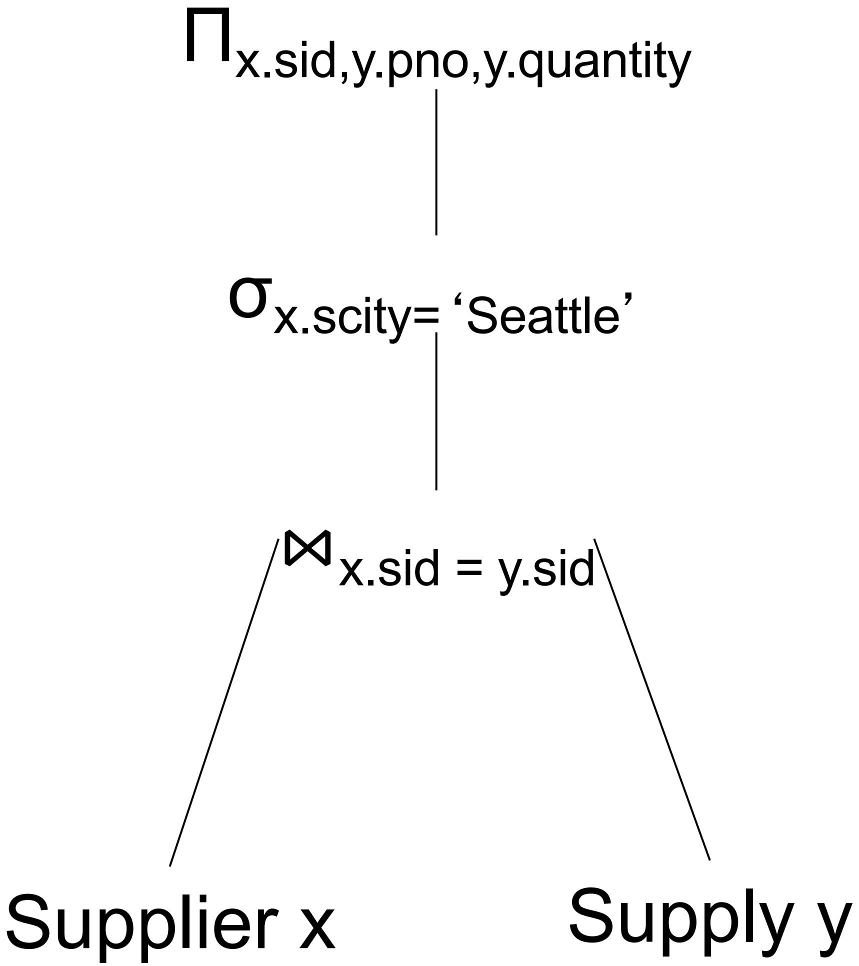
Example Optimization




```
SELECT x.sid, y.pno, y.quantity
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
      and x.scity = 'Seattle'
```

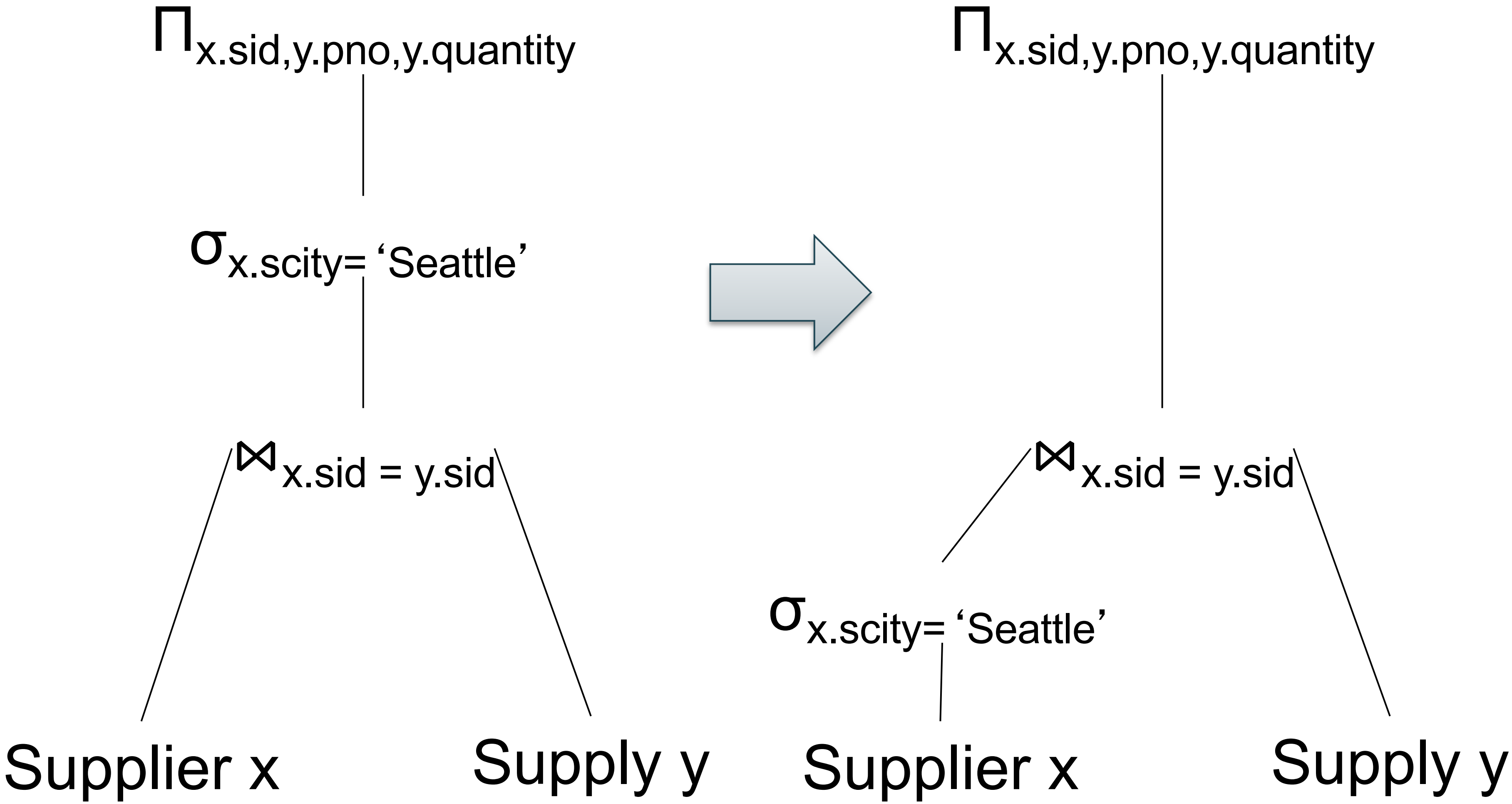
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



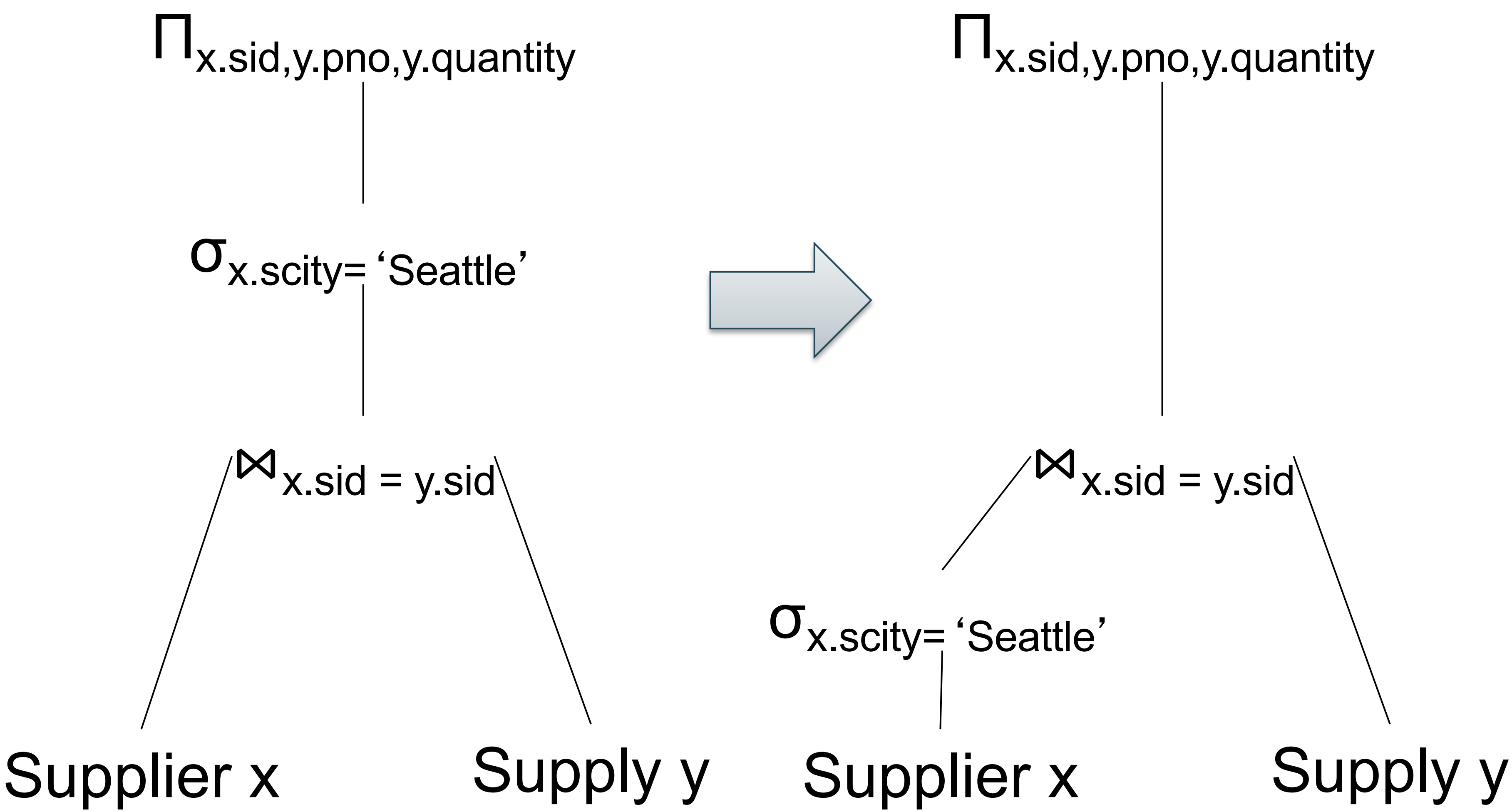
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

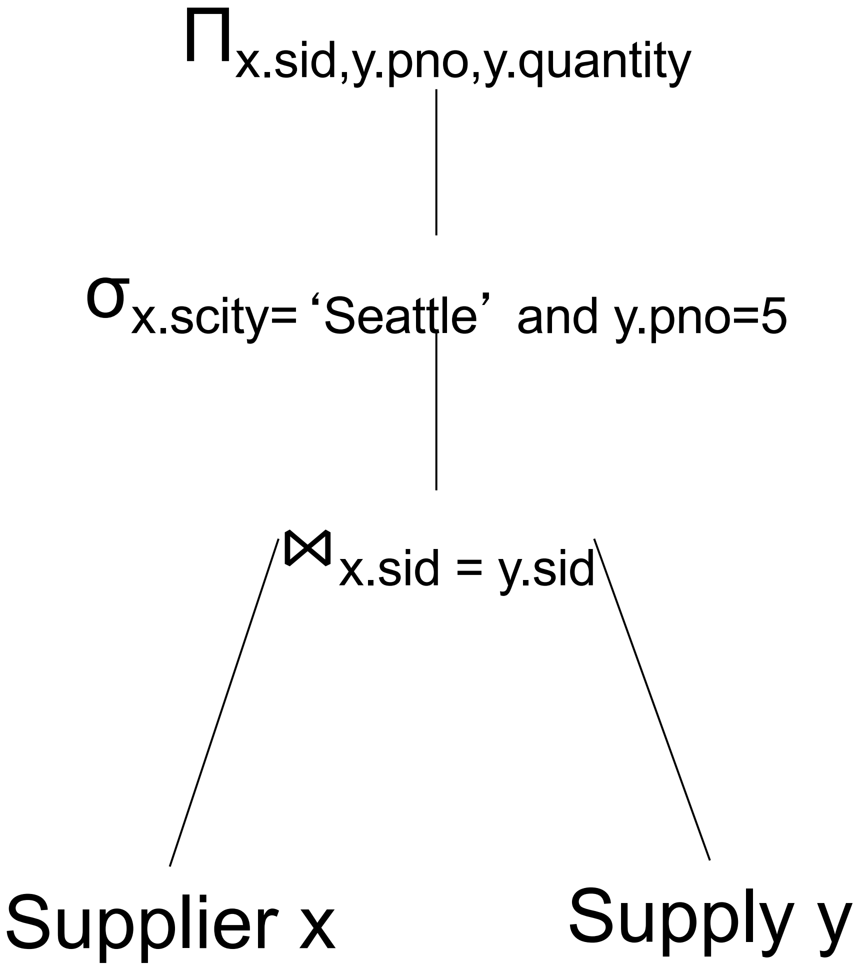
Push Selections Down



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

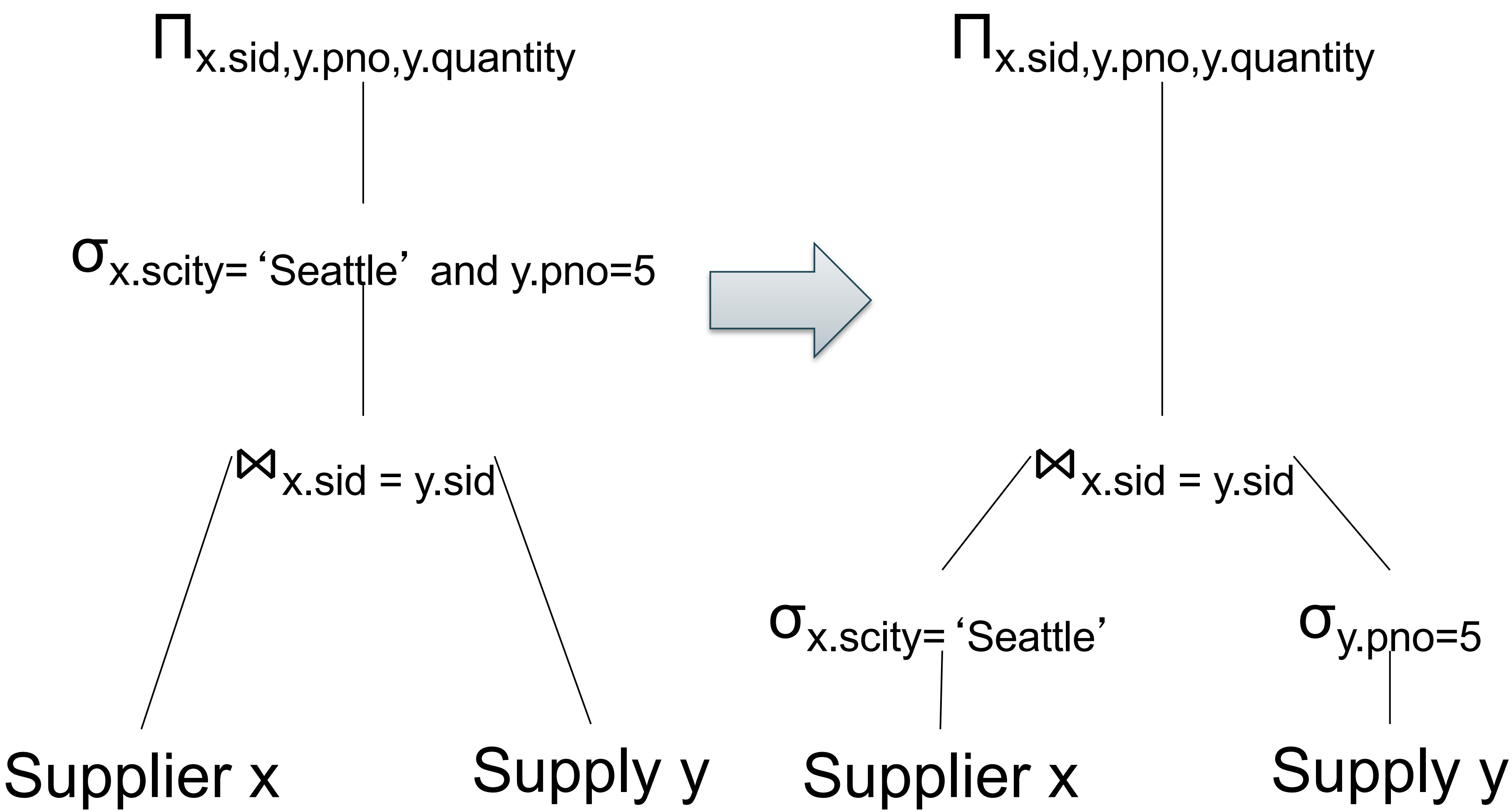
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

Push Selections Down



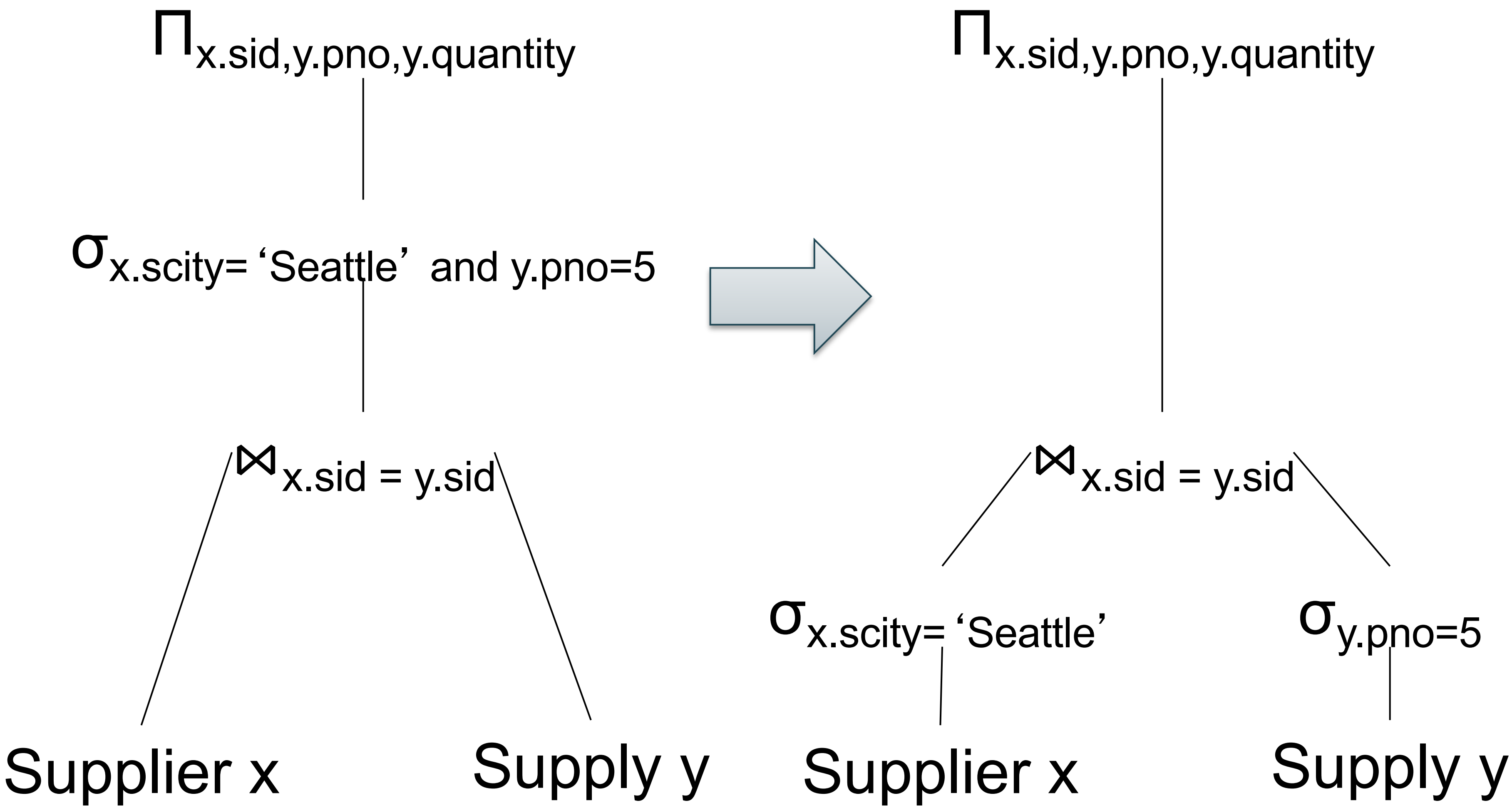
Push Selections Down

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



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

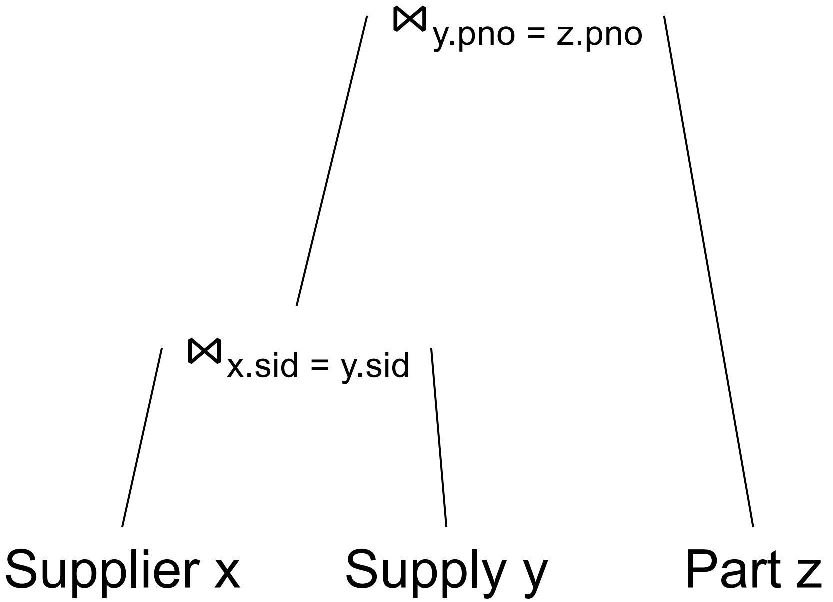
Push Selections Down



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

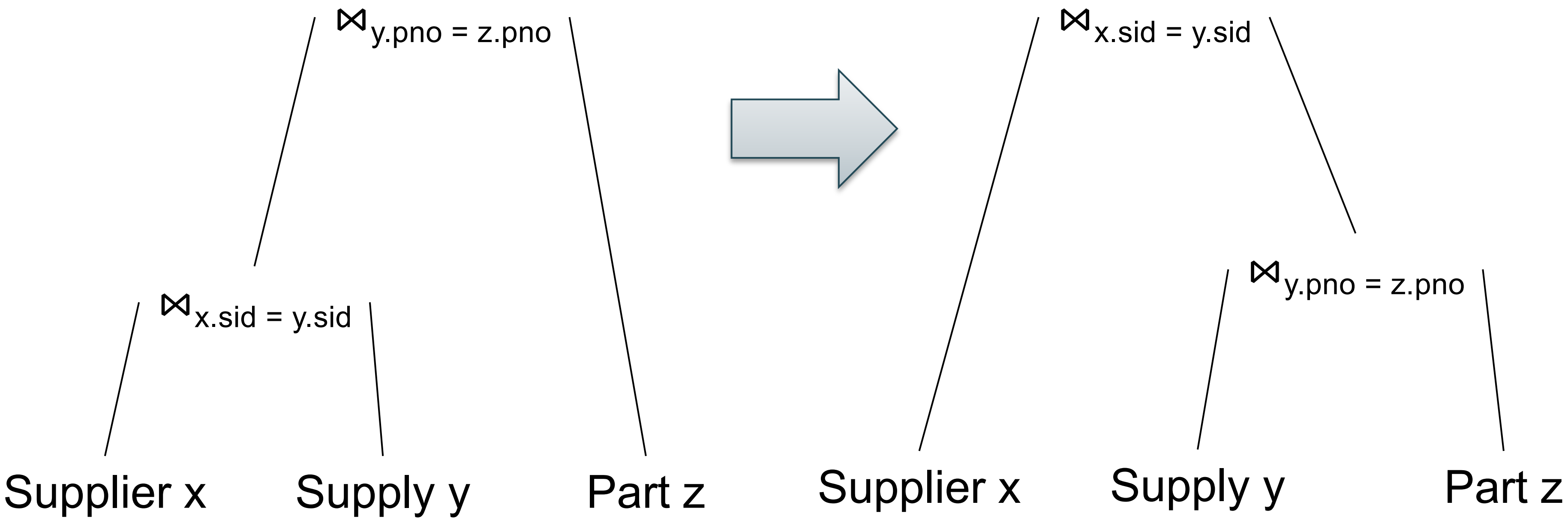
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Join Reorder



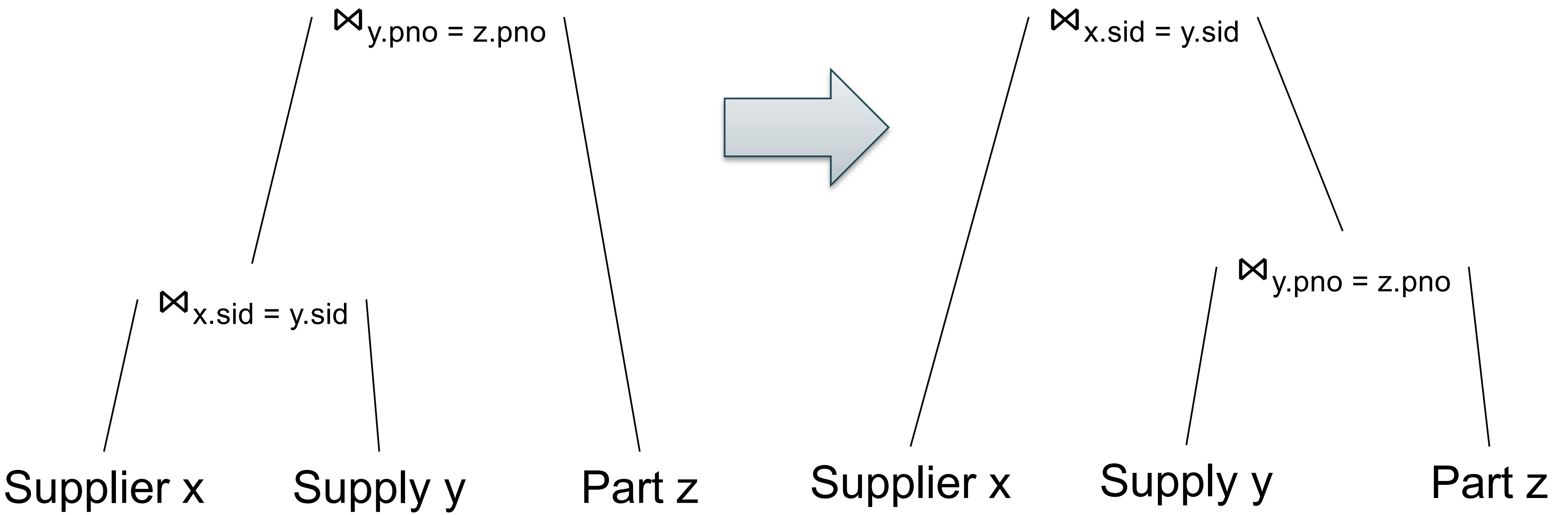
Join Reorder

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Join Reorder



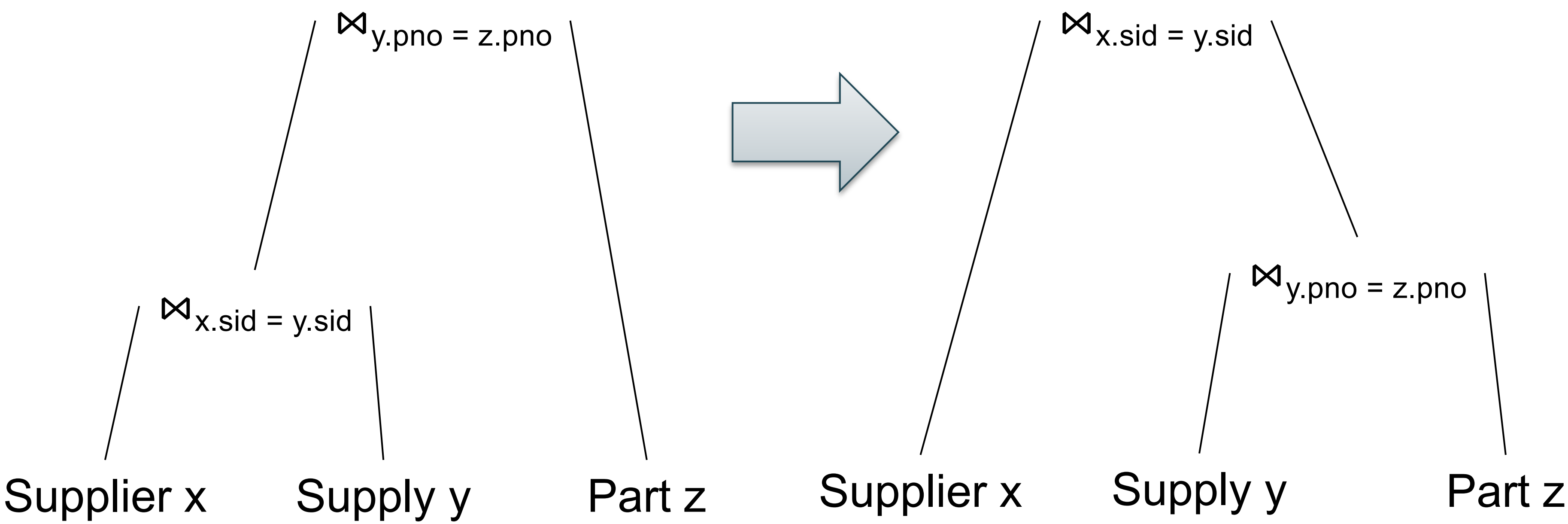
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \bowtie S = S \bowtie R$$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

When is one plan better than the other?



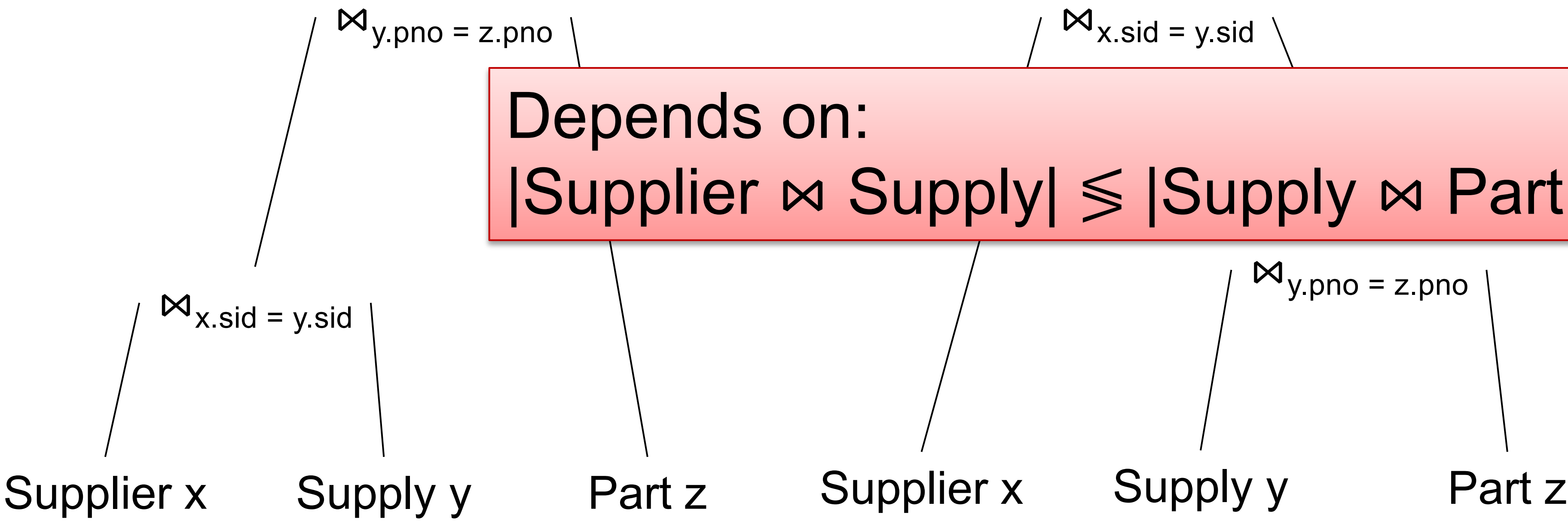
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \bowtie S = S \bowtie R$$

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Join Reorder

When is one plan better than the other?



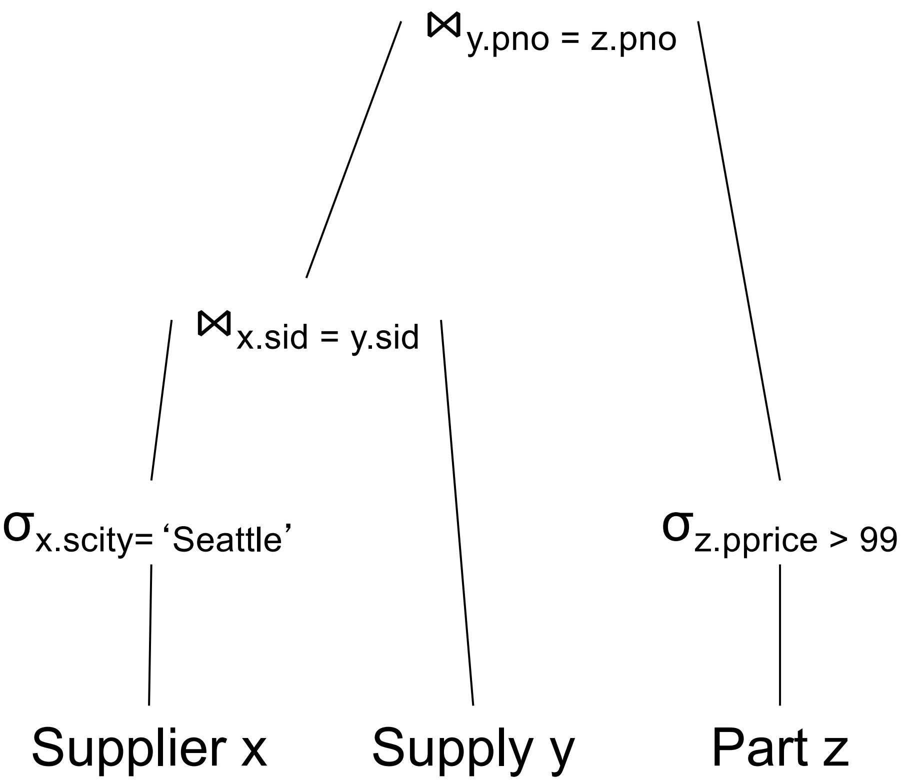
Depends on:
$|\text{Supplier} \bowtie \text{Supply}| \lesseqgtr |\text{Supply} \bowtie \text{Part}|$

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

$R \bowtie S = S \bowtie R$

Join Reorder

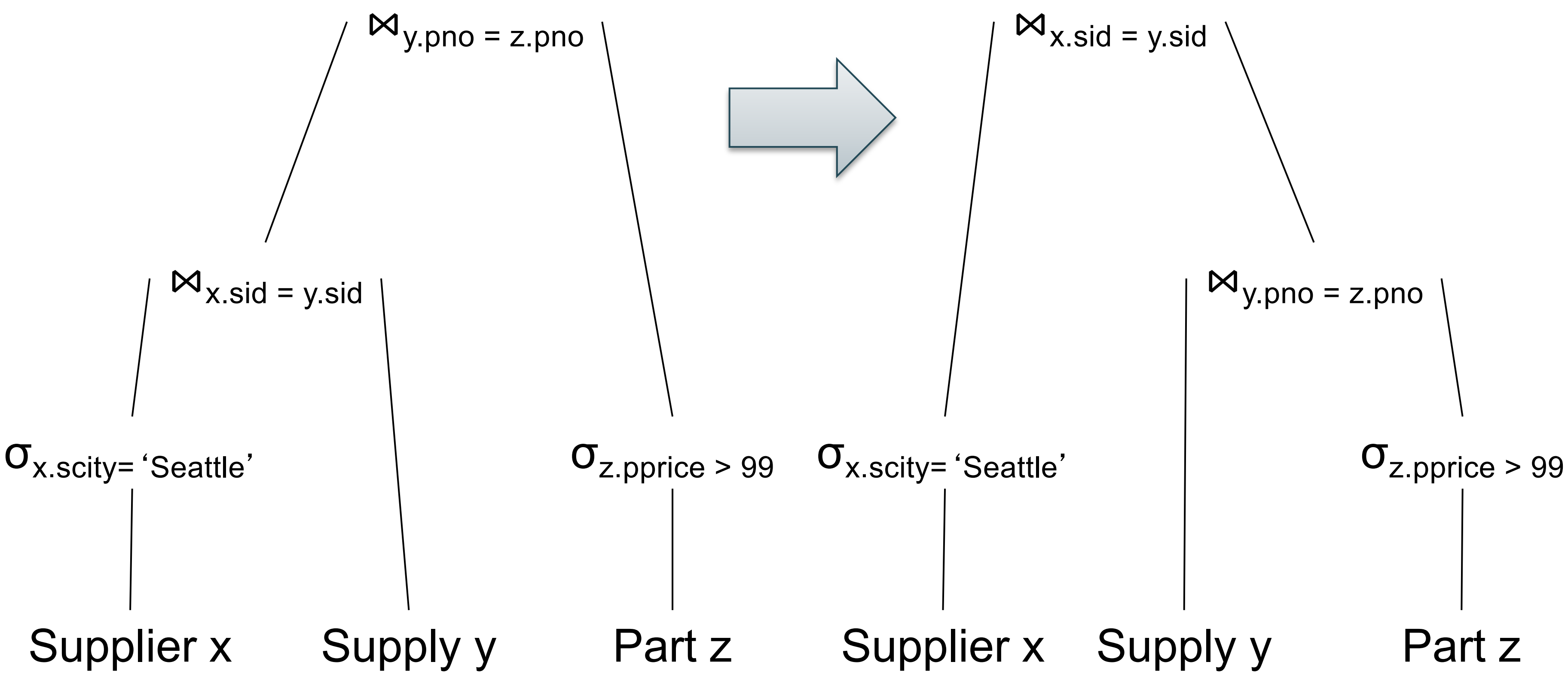
`Supplier(sid, sname, scity, sstate)`
`Supply(sid, pno, quantity)`
`Part(pno, pname, pprice)`



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Join Reorder

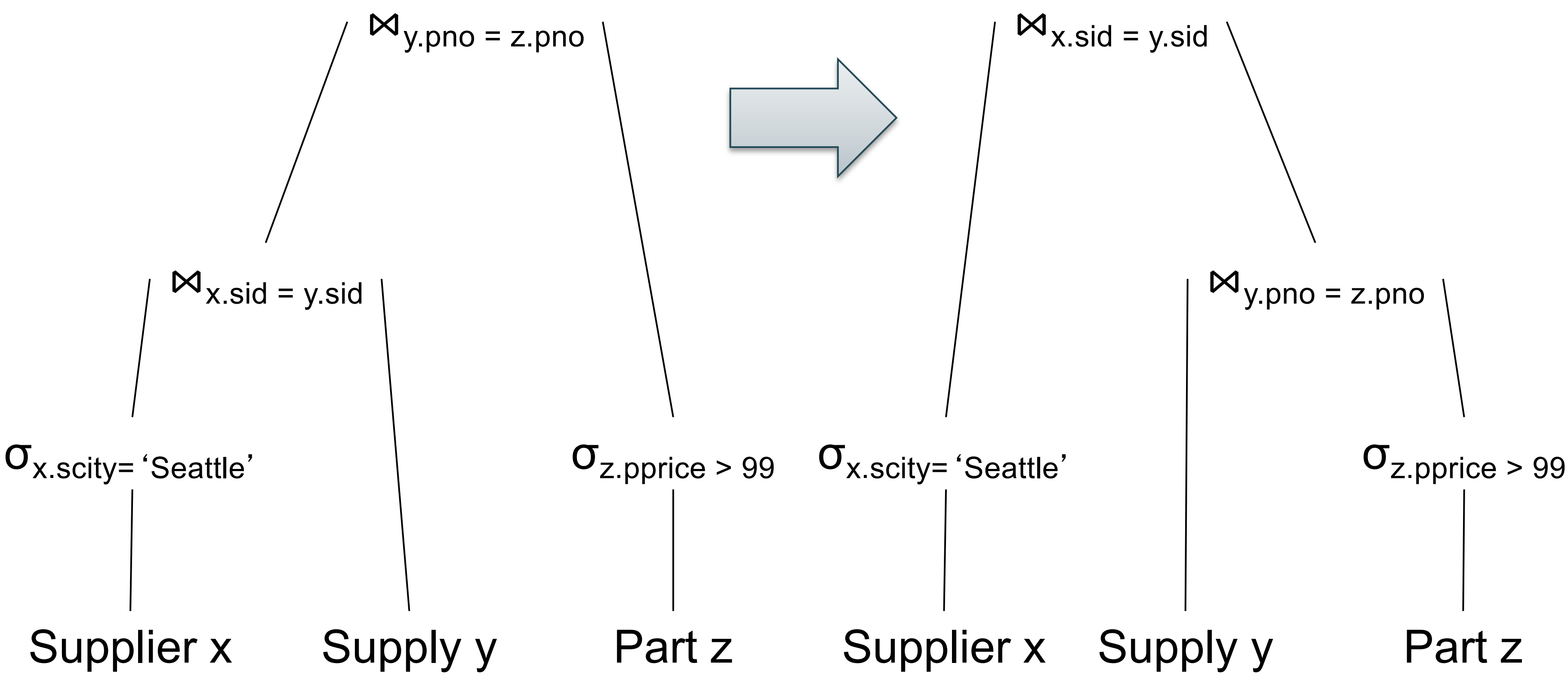
When is one plan better than the other?



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Join Reorder

When is one plan better than the other?



Lesson: need sizes of $\sigma_{x.scity=\text{'Seattle'}}$(Supplier), $\sigma_{z.pprice > 99}$(Part)

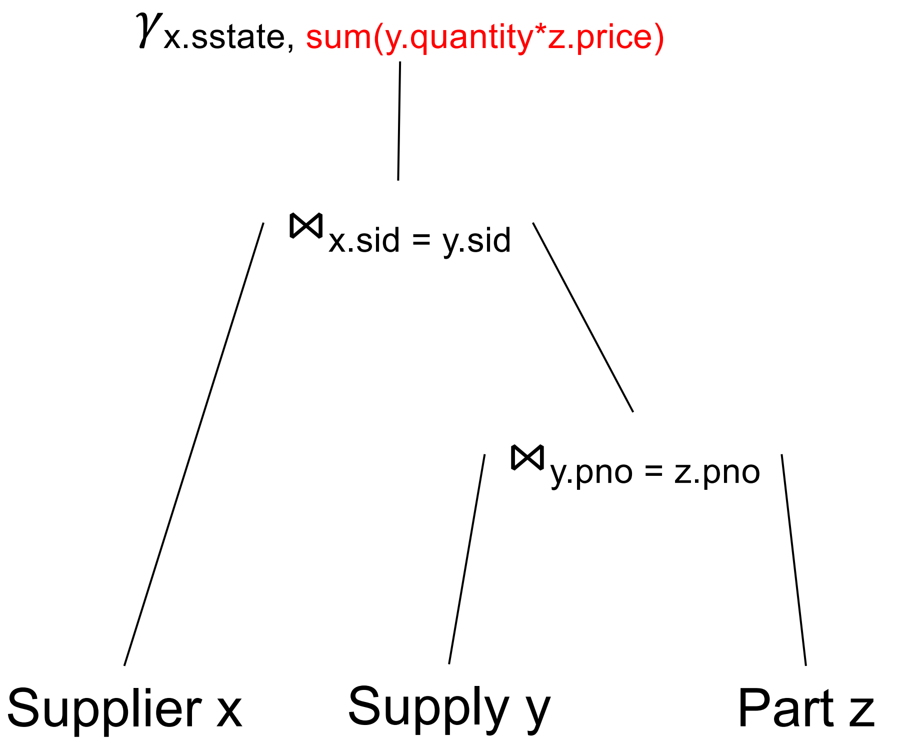
Aggregate Push-down

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

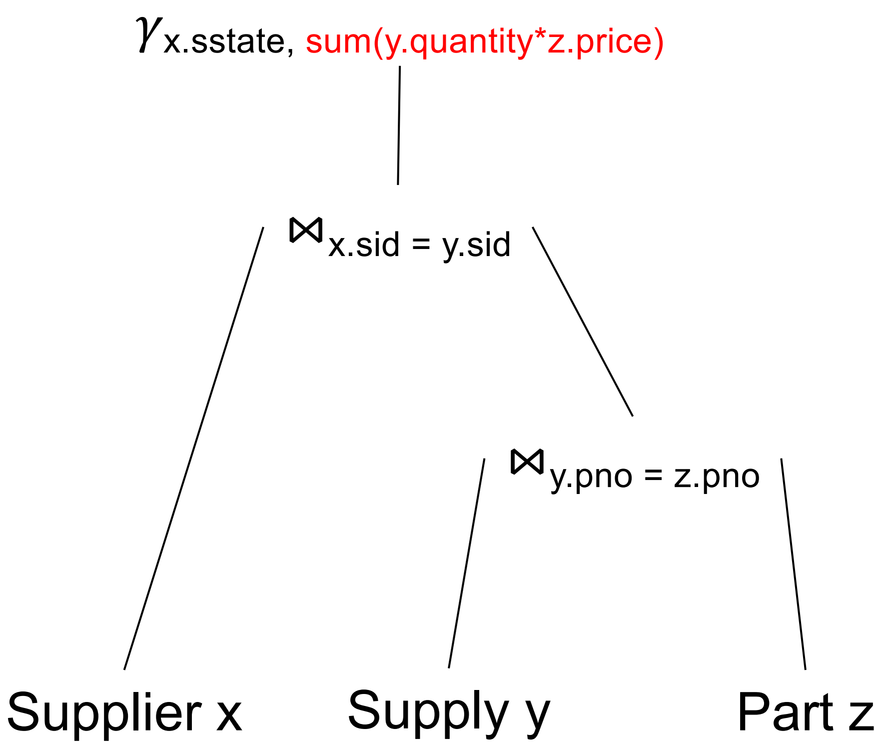
Aggregate Push-down

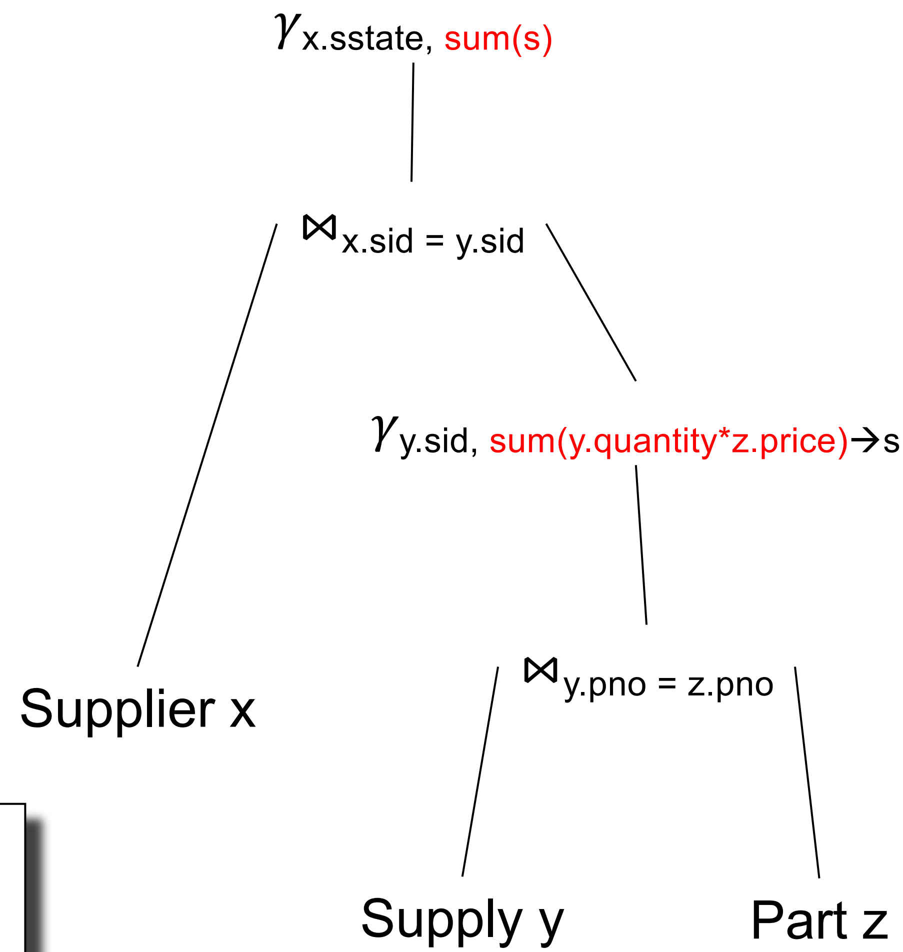


SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)
```

Aggregate Push-down

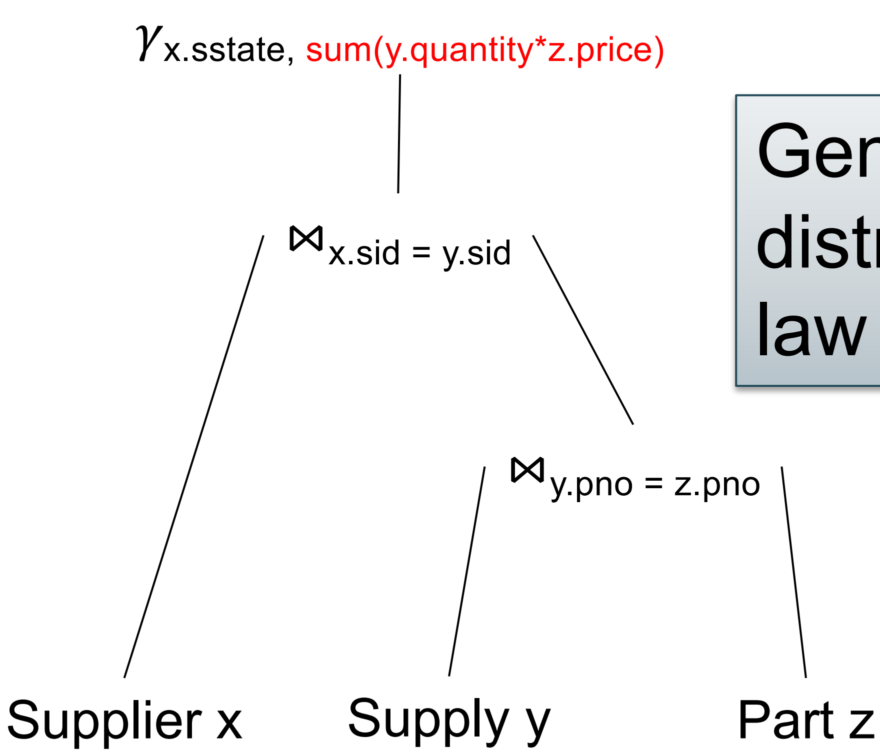




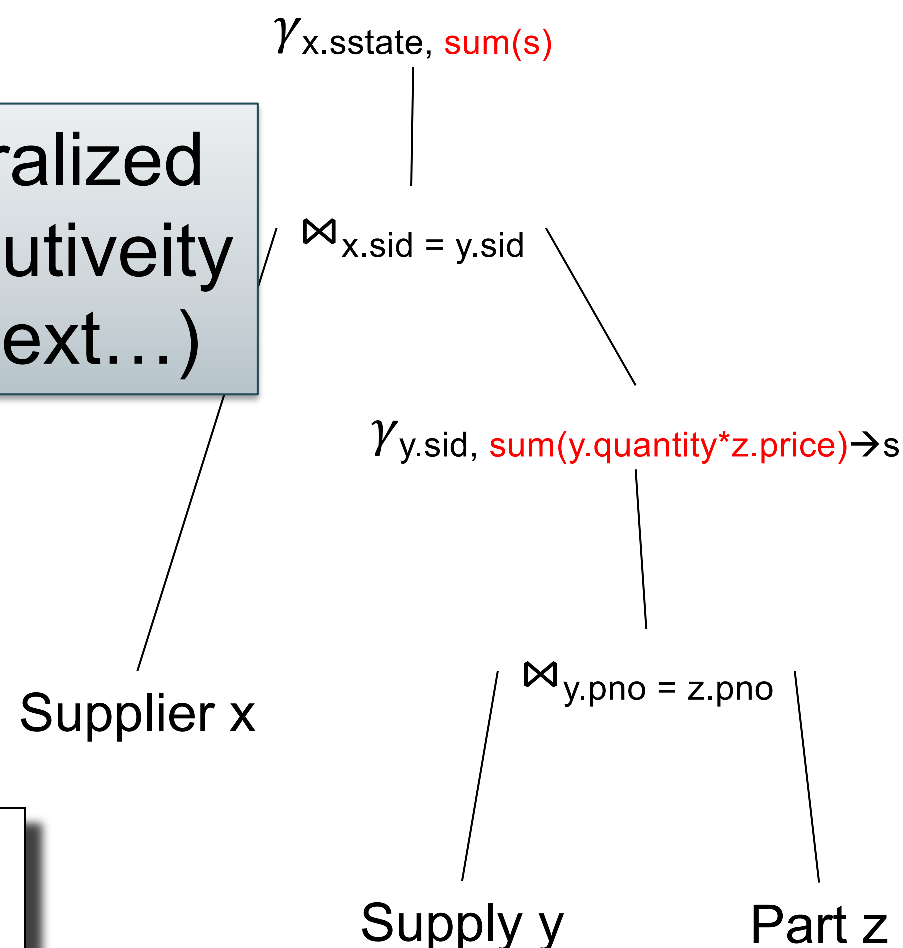

```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Aggregate Push-down



Generalized distributiveity law (next…)




```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

Aggregate Push-Down

- Motivation: try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

Aggregate Push-Down

- Motivation: try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

```
select count(*) from publication;
```

Answer: 5120896
Time: 0.062 s

Aggregate Push-Down

- Motivation: try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

```
select count(*) from publication;
```

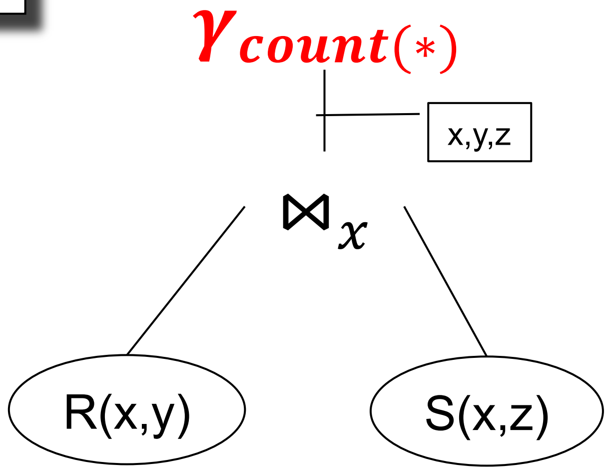
Answer: 5120896
Time: 0.062 s

```
select count(*) from author, publication;
```

Timeout!!!

Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

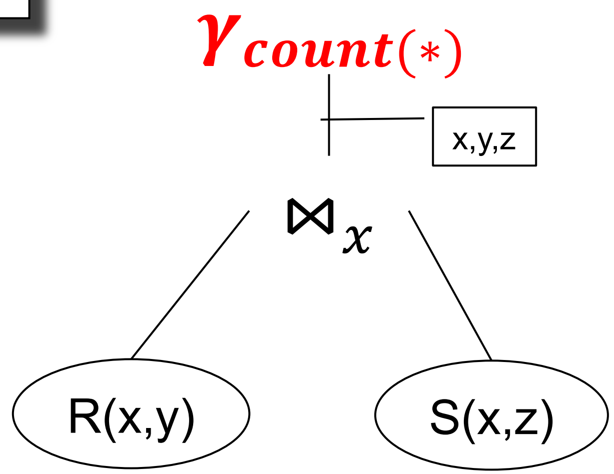
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = O(N^2)



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

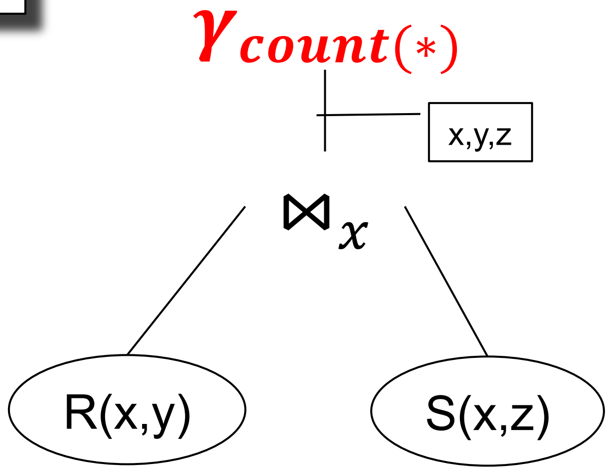
x	y
b	a
b	c
f	d
h	g

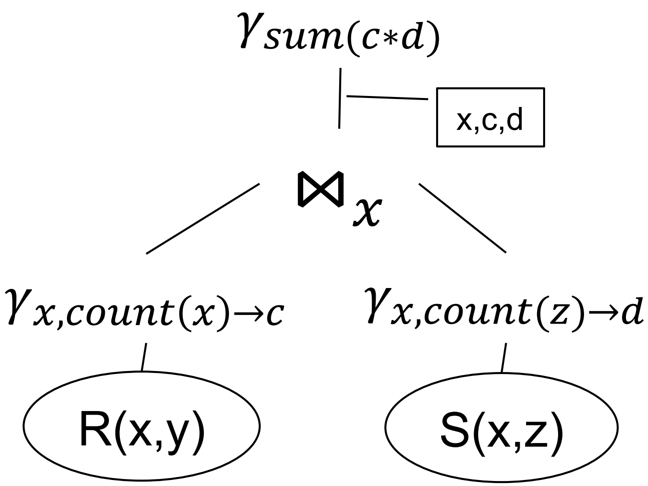
S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$





Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

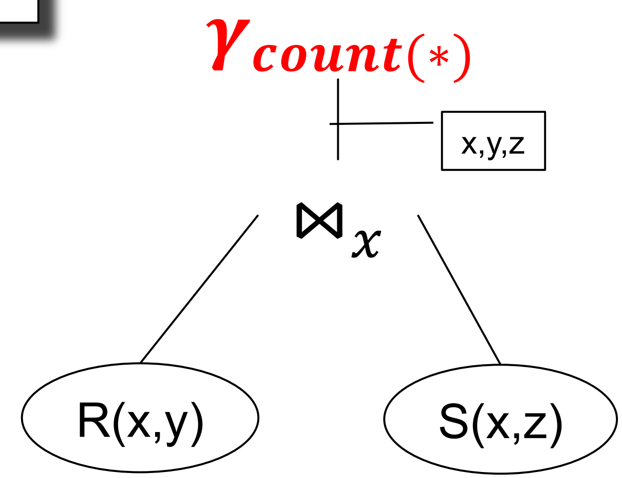
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$



A:

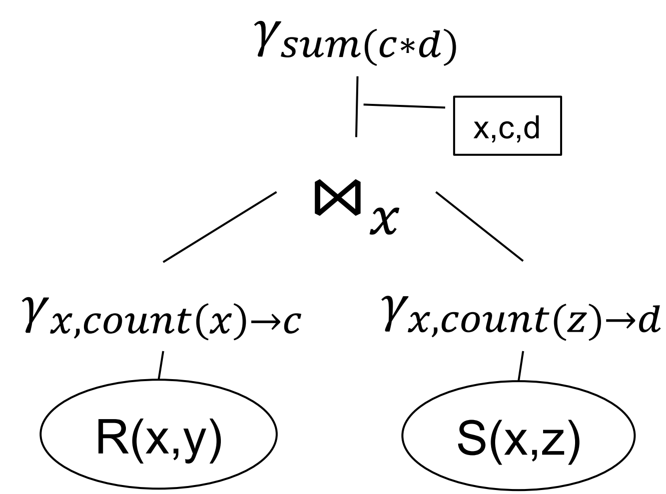
x	c
b	2
f	1
h	1

B:

x	d
b	2
h	1

A⋈B

x	c	d
b	2	2
h	1	1



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

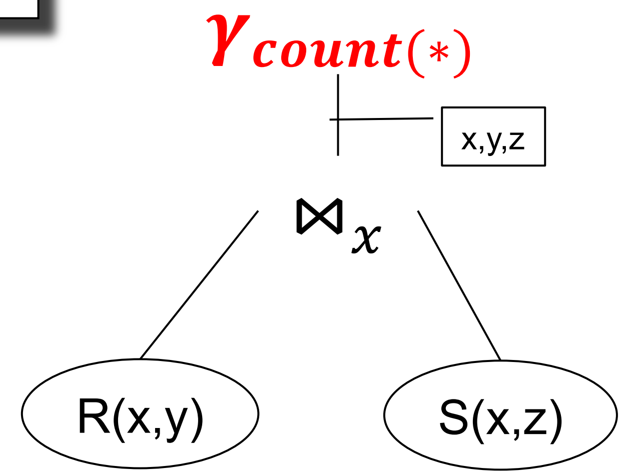
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$



Runtime = $O(N)$

A:

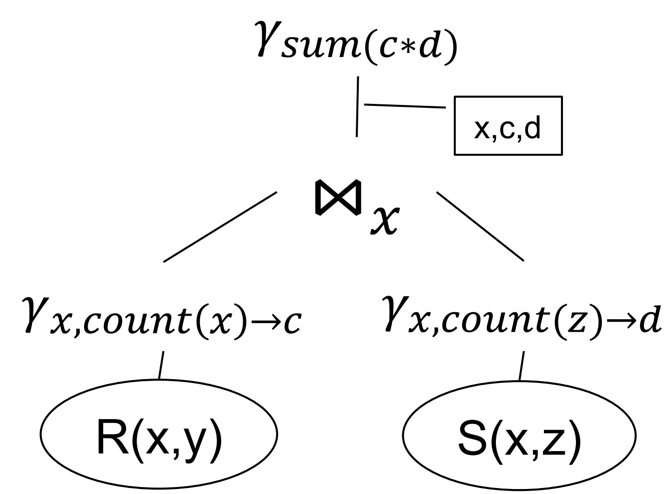
x	c
b	2
f	1
h	1

B:

x	d
b	2
h	1

A⋈B

x	c	d
b	2	2
h	1	1



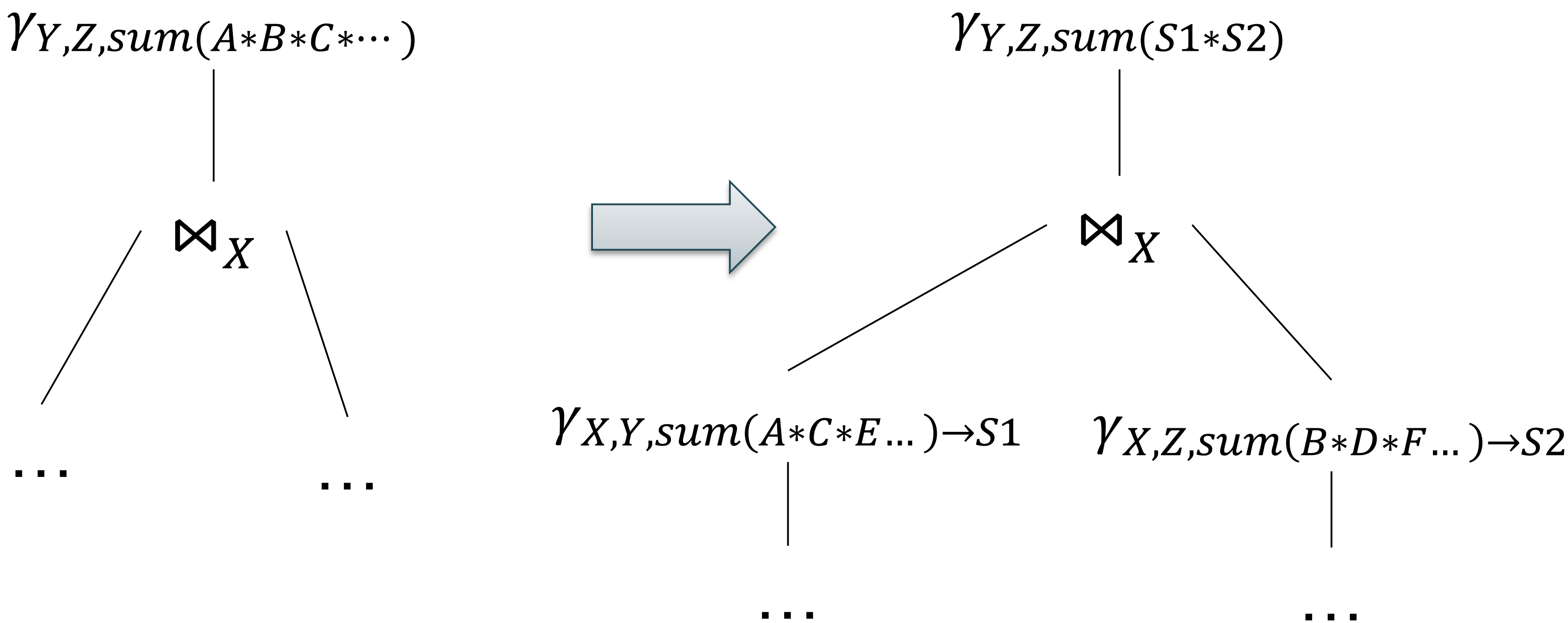
Generalized Distributivity Law

$$\gamma_{Y,Z,sum(A*B*C*\cdots)}$$

$$\bowtie_X$$

$$\cdots \qquad \cdots$$

Generalized Distributivity Law



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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

?

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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

```
Select x.pno, x.quantity
From Supply x
```

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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

```
Select x.pno, x.quantity
From Supply x
```

Only if these constraints hold:

1. Supplier.sid = key
2. Supply.sid = foreign key
3. Supply.sid NOT NULL

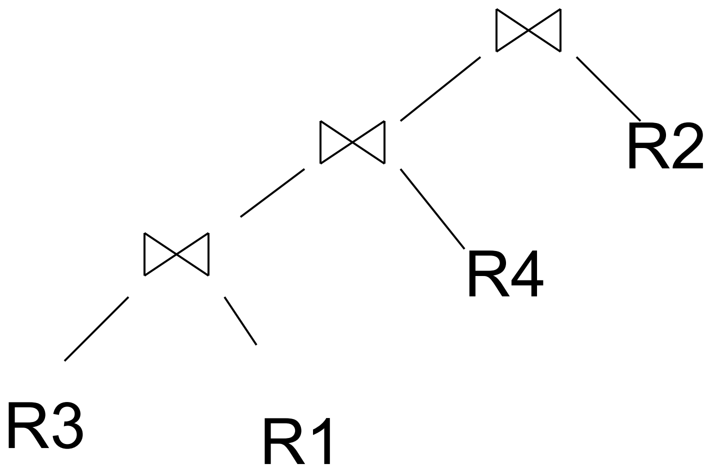
Practice

- Database optimizers typically have a database of rewrite rules
- E.g. SQL Server is rumored to have about 500 rules
- Rules become complex as they need to serve specialized types of queries

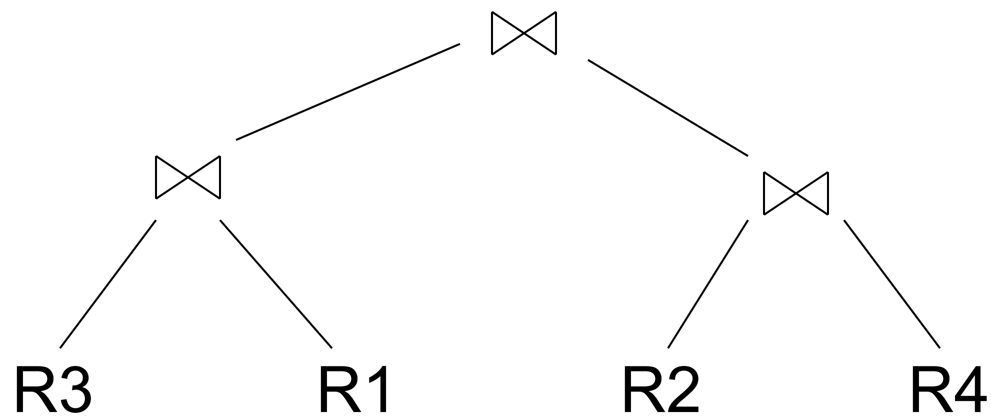
Search Space Challenges

- Search space is huge
- Typical compromises:
  - Left-deep plans
  - Plans without cartesian products

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

Announcements

- HW2 was due last Friday
- Paper review due on Wednesday
- Project proposals due next Friday
- HW3 is posted

Query Optimization

<u>Three major components</u>:

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms

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

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

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

Selectivity Factors

Uniformity assumption

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

Equality:

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

Selectivity Factors

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

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

Uniformity assumption

Equality:

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

Range:

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

Selectivity Factors

Uniformity assumption

Equality:

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

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

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

Range:

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

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

Independence assumption

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

Selectivity Factors

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

Join

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

Why? Will explain next...

Selectivity Factors

Containment of values: if V(R,A) ≤ 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

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

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

Selectivity Factors

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:

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

Final Assumption

Preservation of values:

For any other attribute C:

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

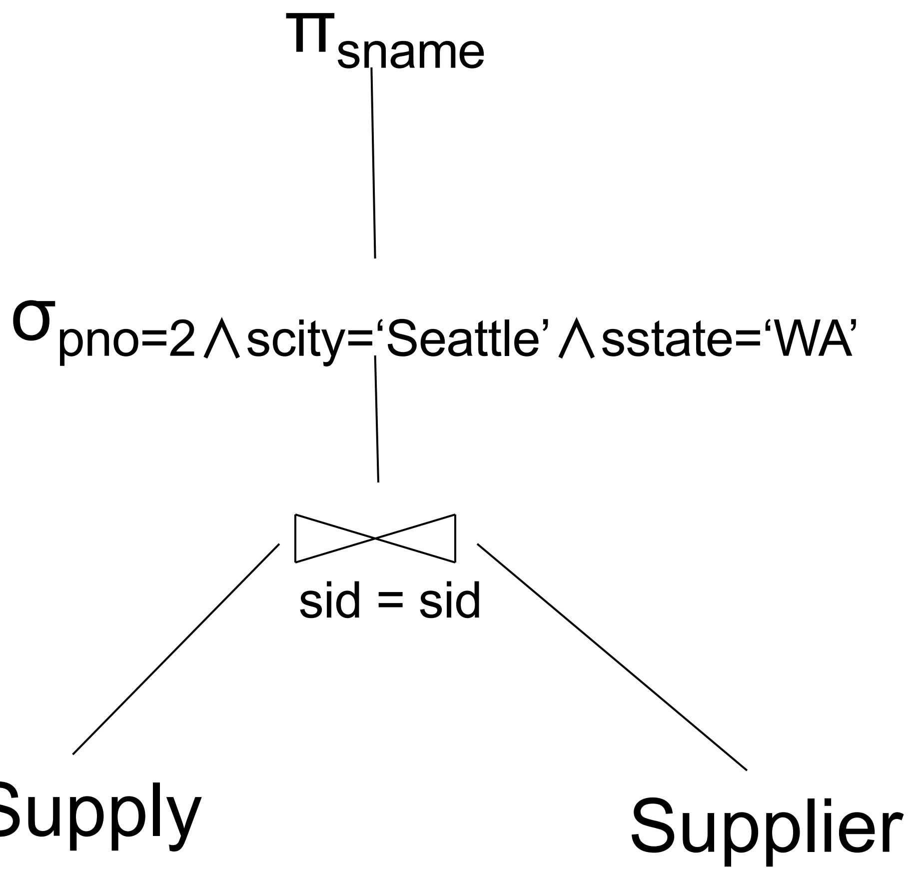
- This is needed higher up in the plan

Computing the Cost of a Plan

- Estimate cardinalities bottom-up
- Estimate cost by using estimated cardinalities
- Extensive example next...

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, 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'
```

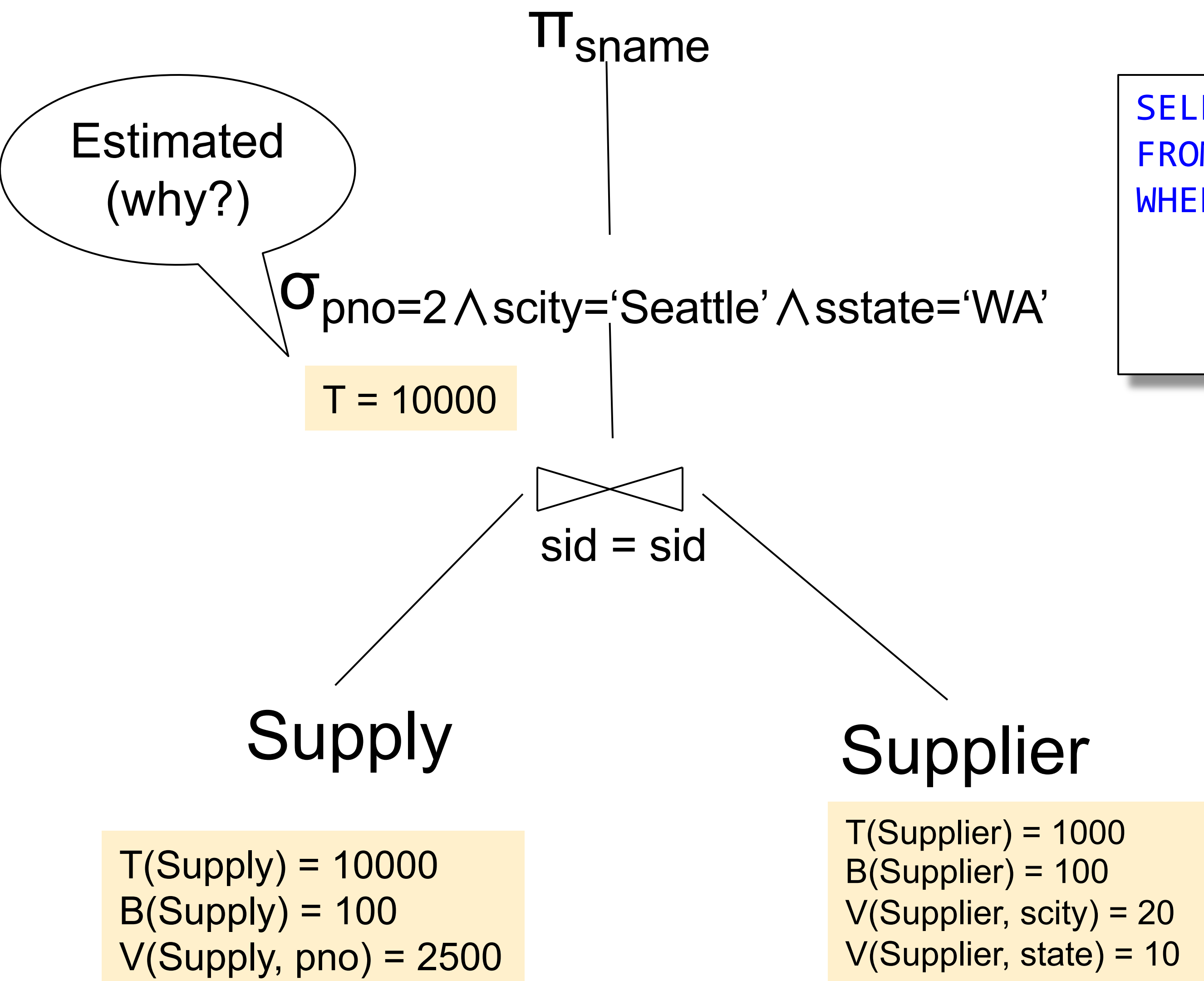
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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'
```

M=11

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

Logical Query Plan 1

Estimated (why?)

π_{sname}

T < 1

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

T = 10000

$\bowtie_{sid = sid}$

Supply

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

Supplier

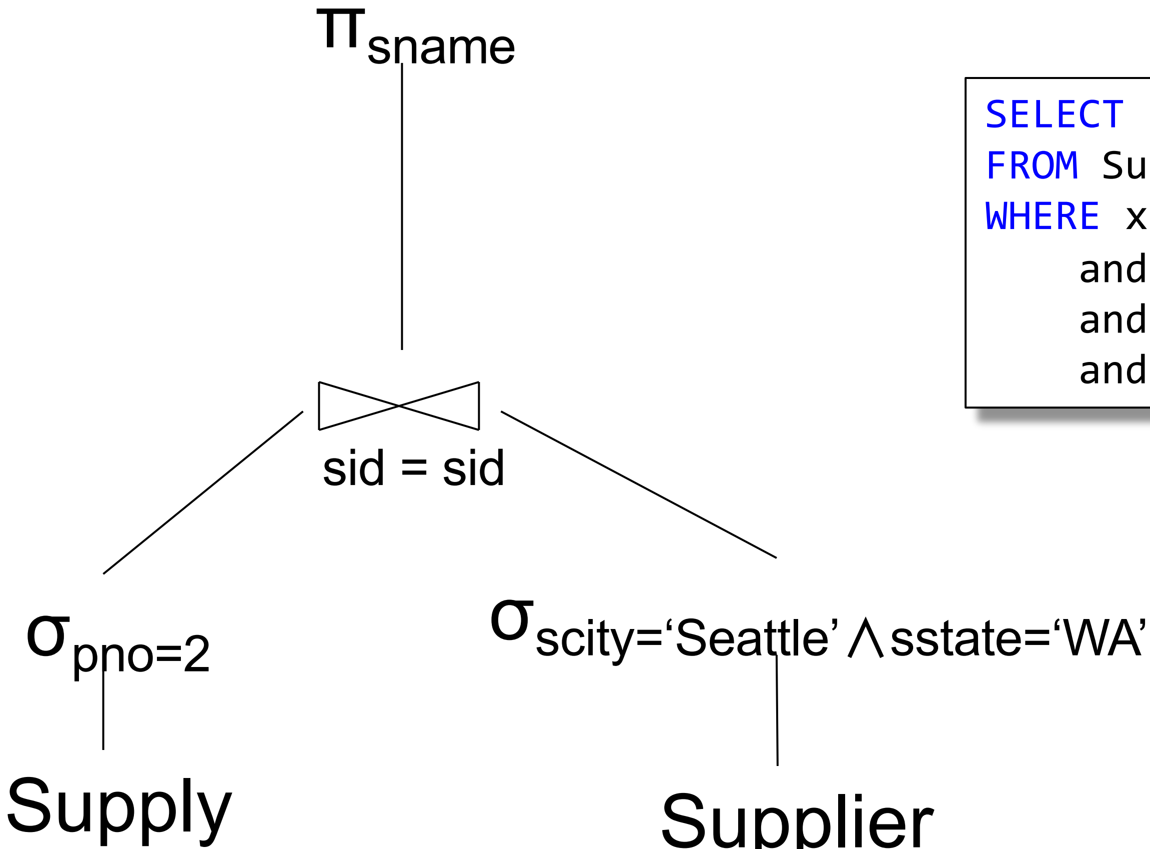
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
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 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'
```

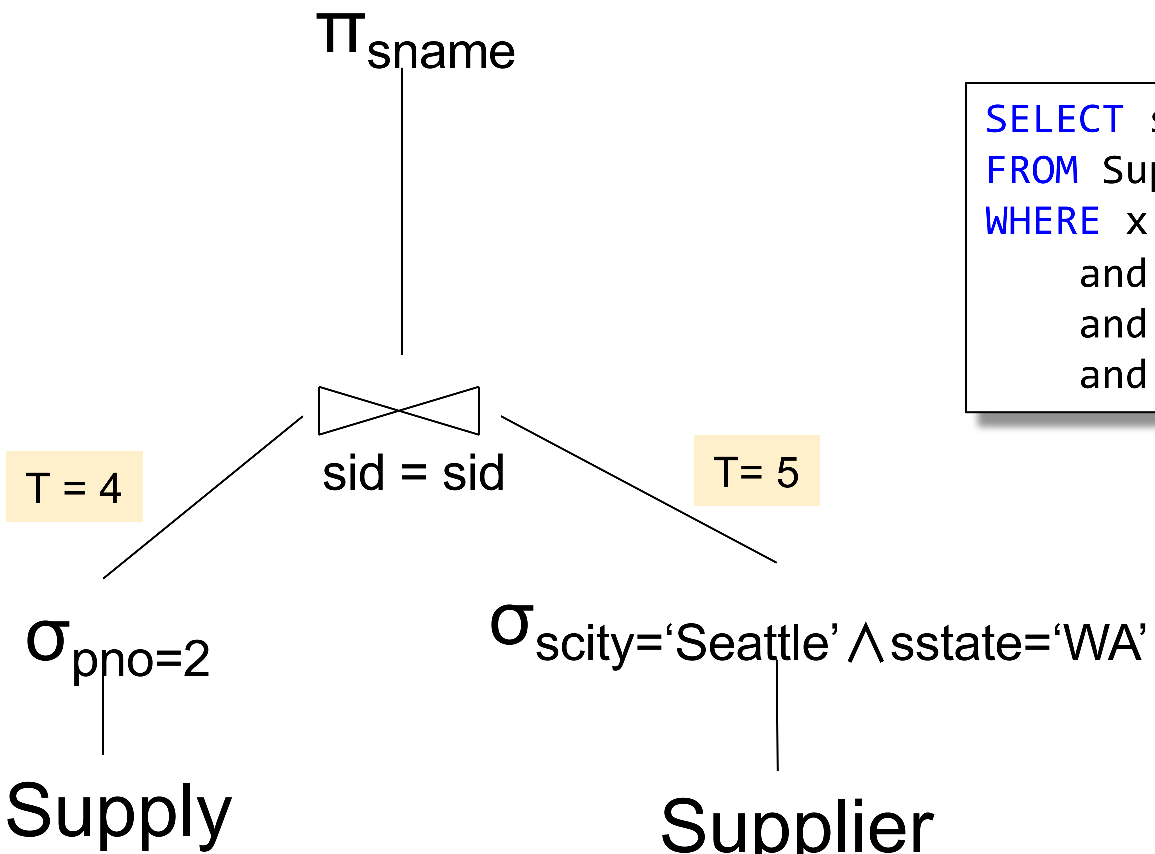
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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'
```

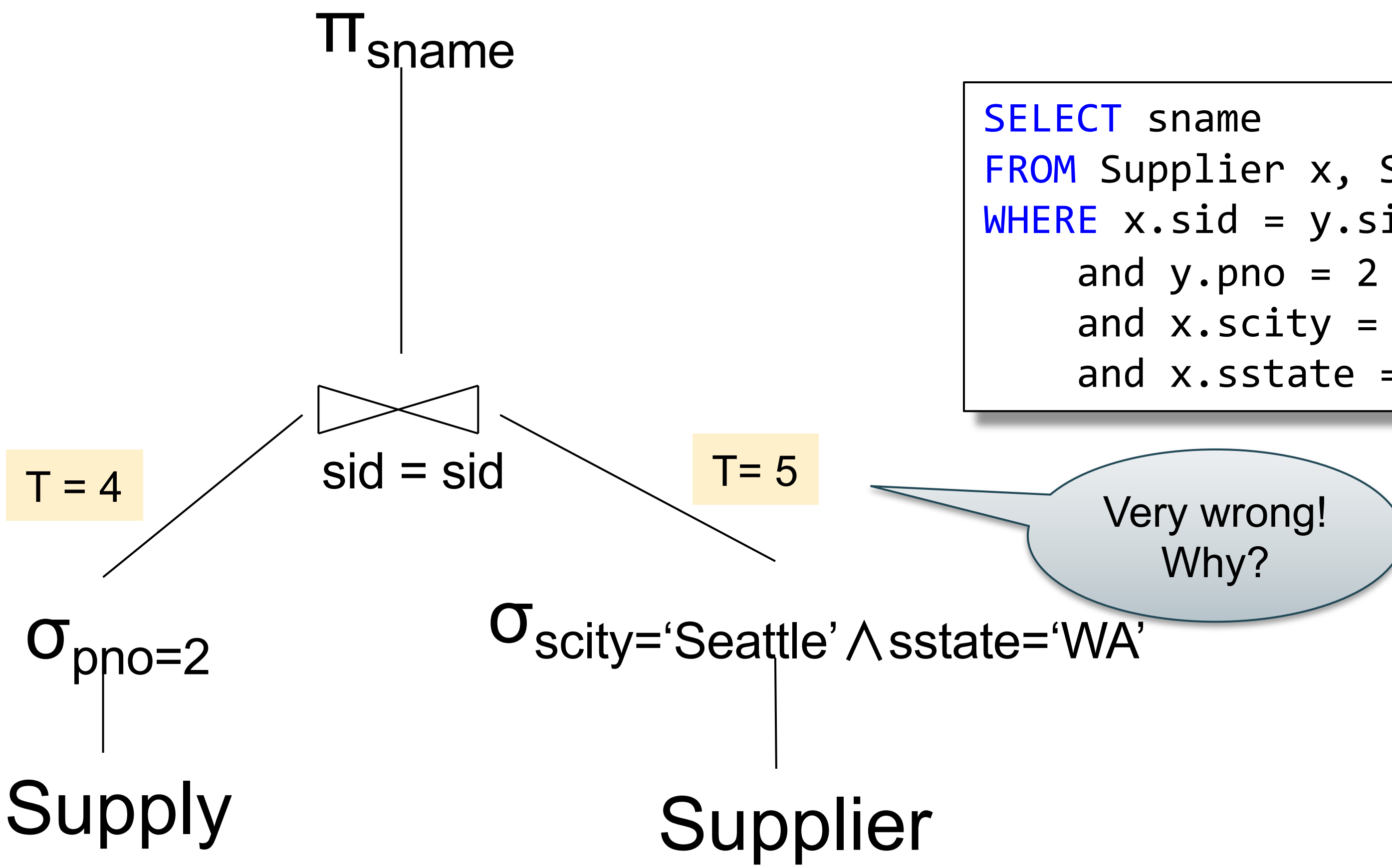
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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'
```

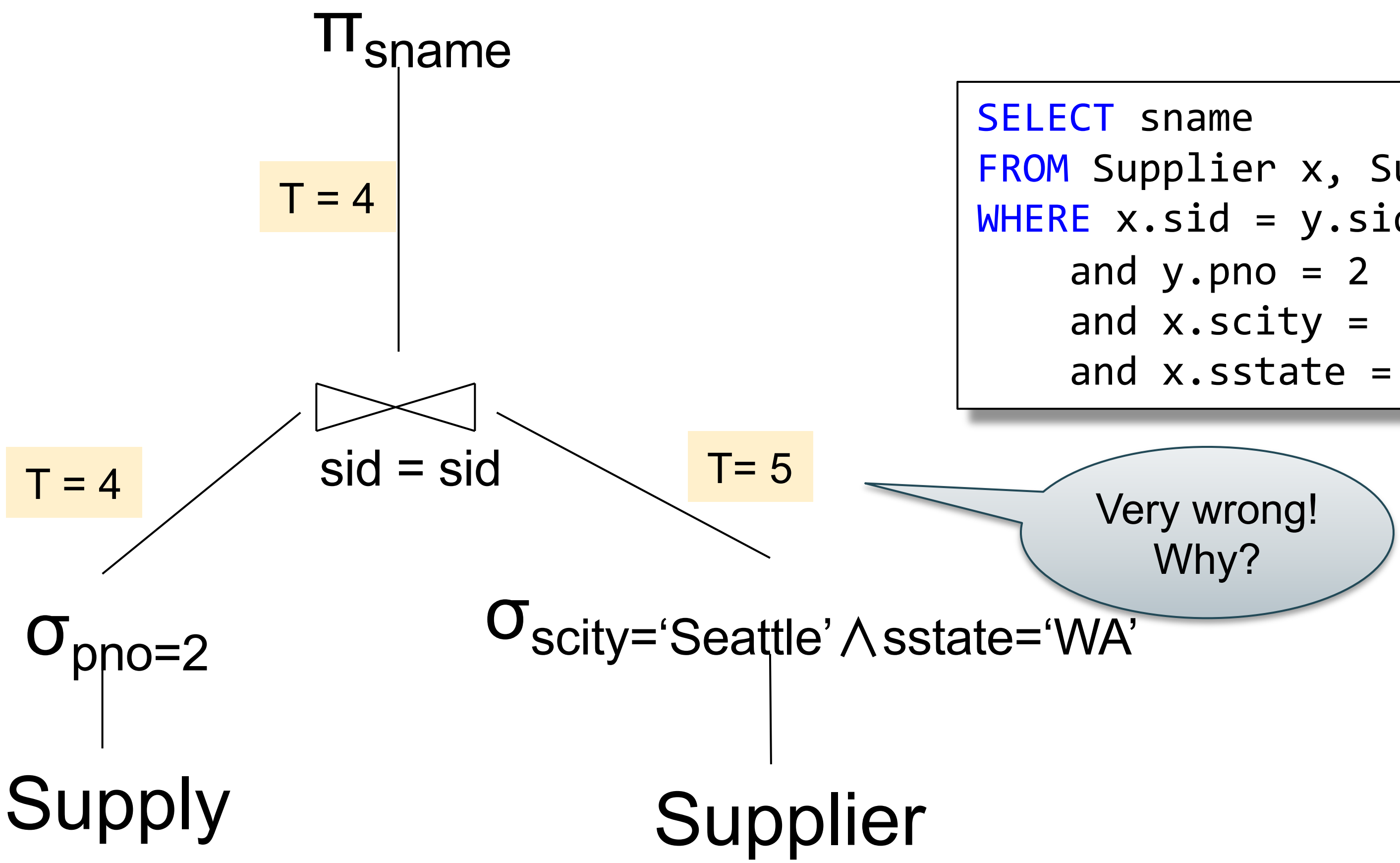
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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




```
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

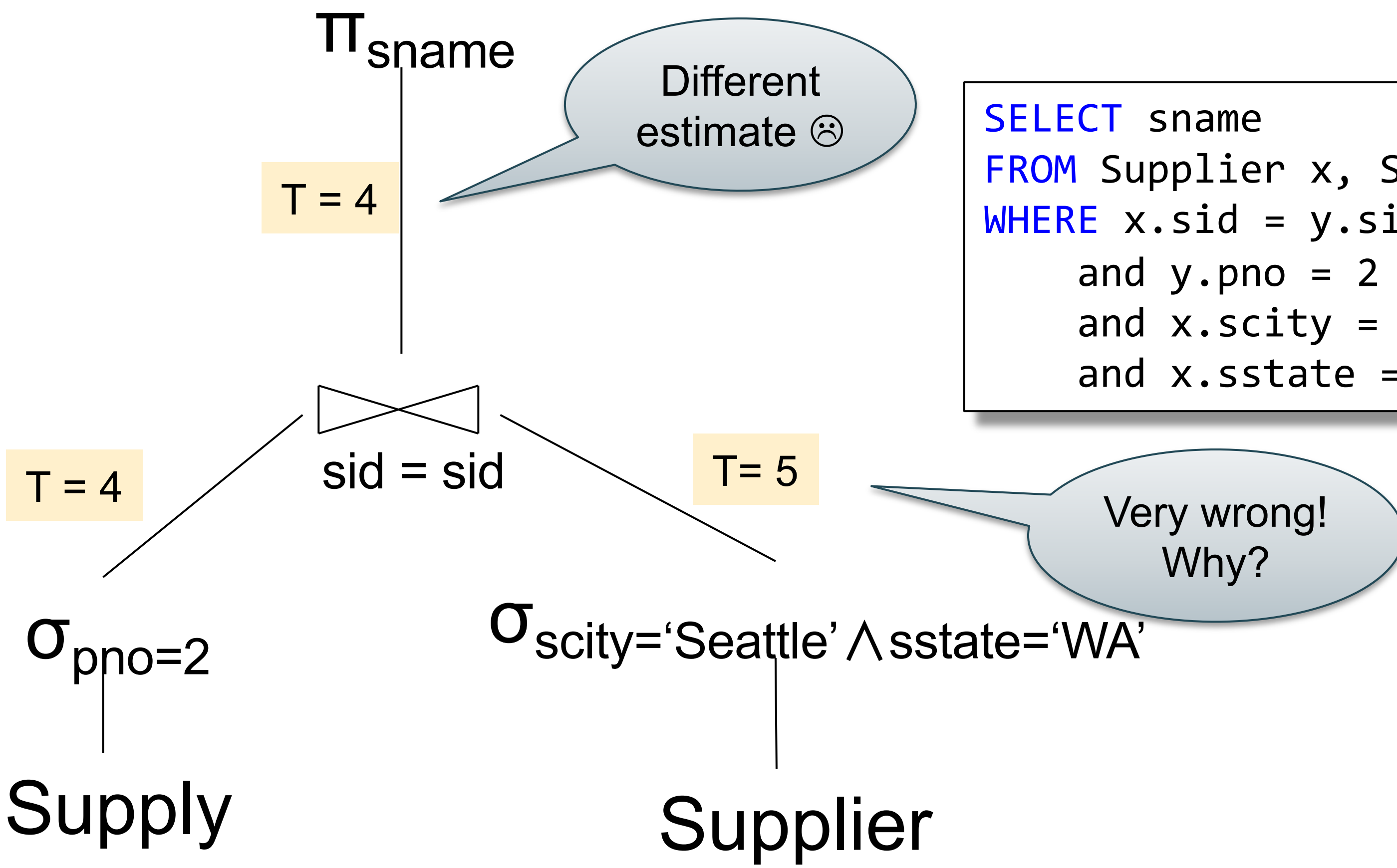
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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’
```



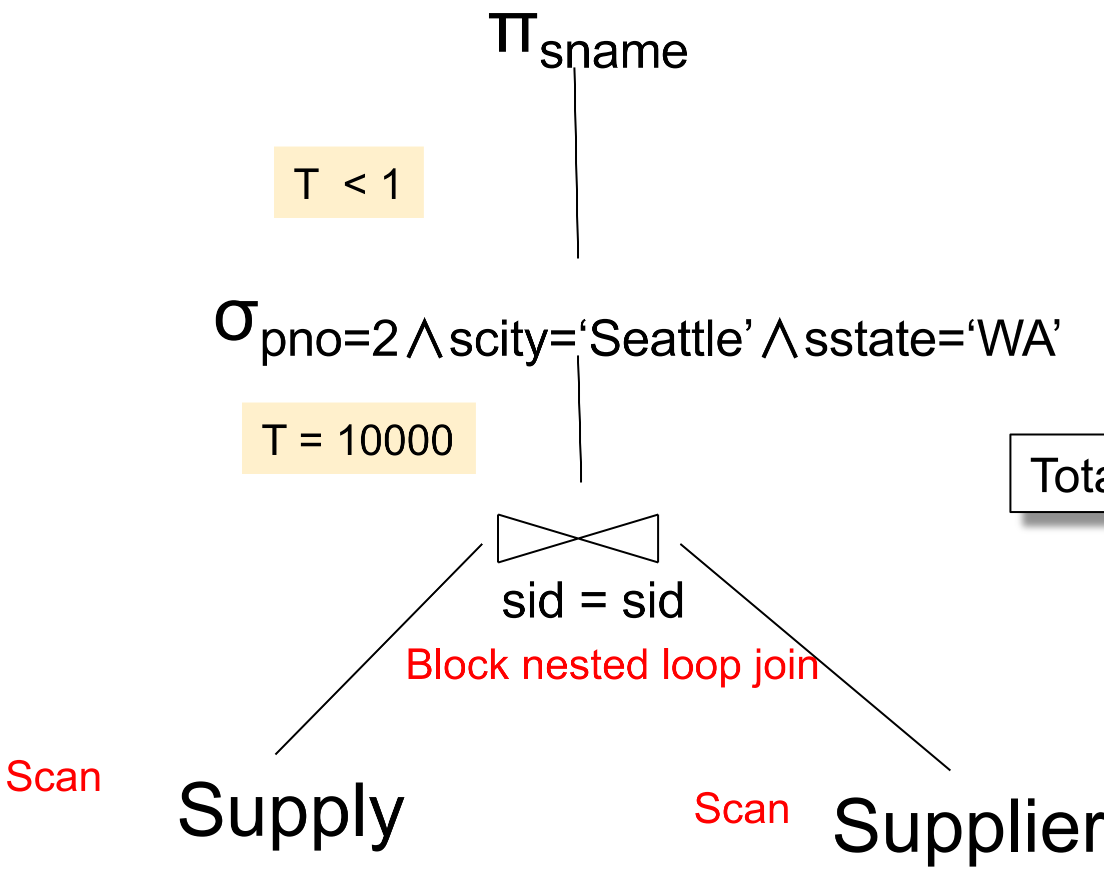
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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)

Physical Plan 1



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(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Physical Plan 1

π_{sname}

T < 1

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

T = 10000

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

$\bowtie$ sid = sid

Block nested loop join

Scan Supply

Scan Supplier

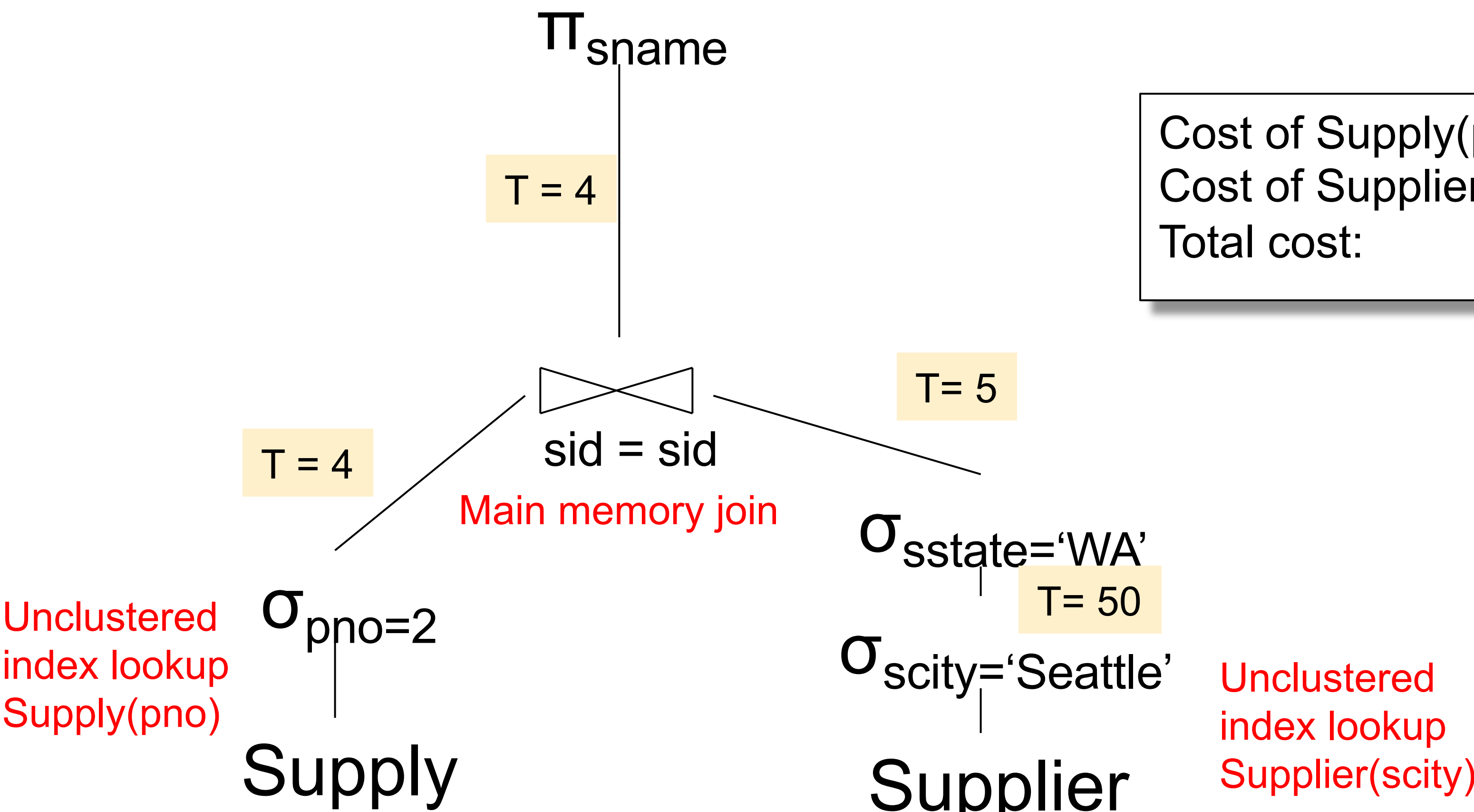
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Physical Plan 2



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

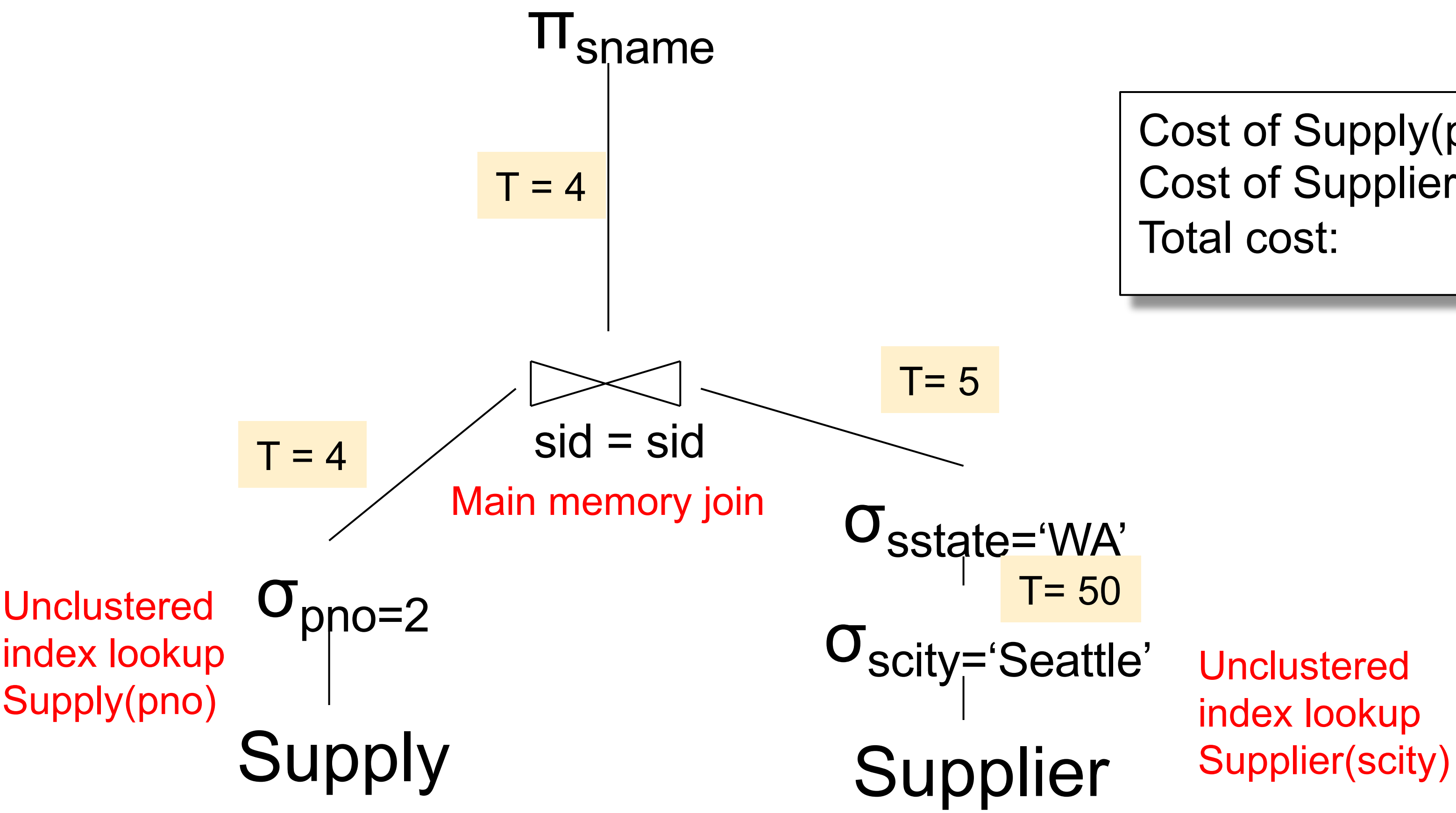
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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:

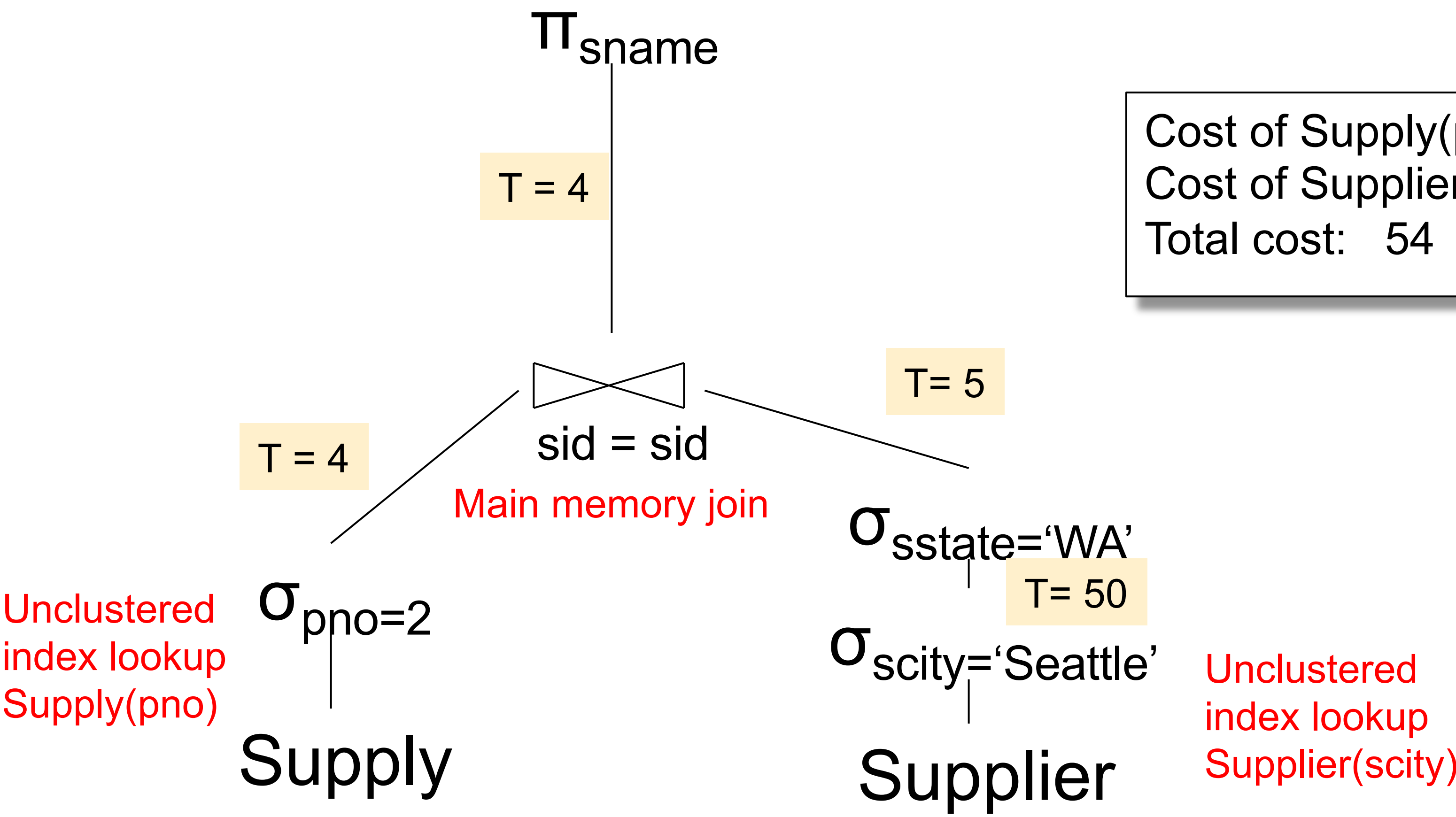
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

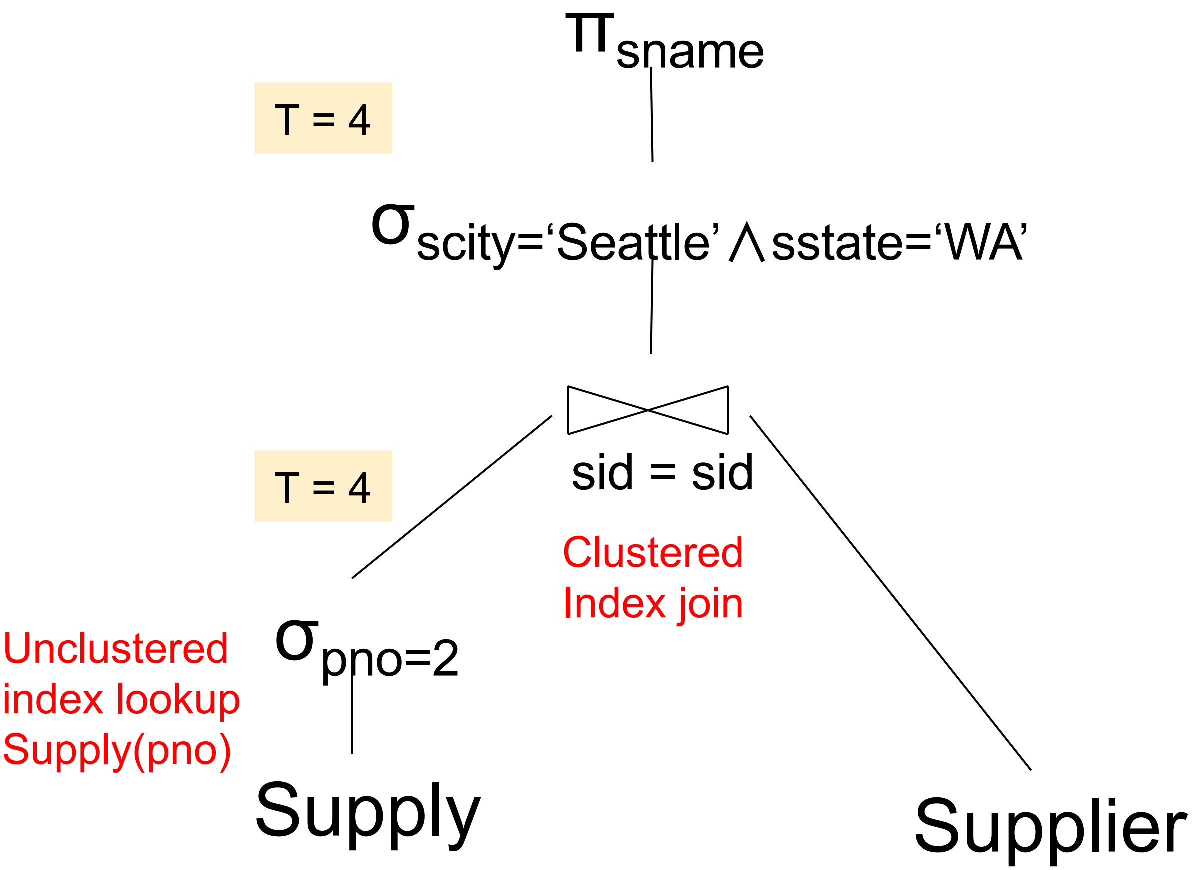
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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:

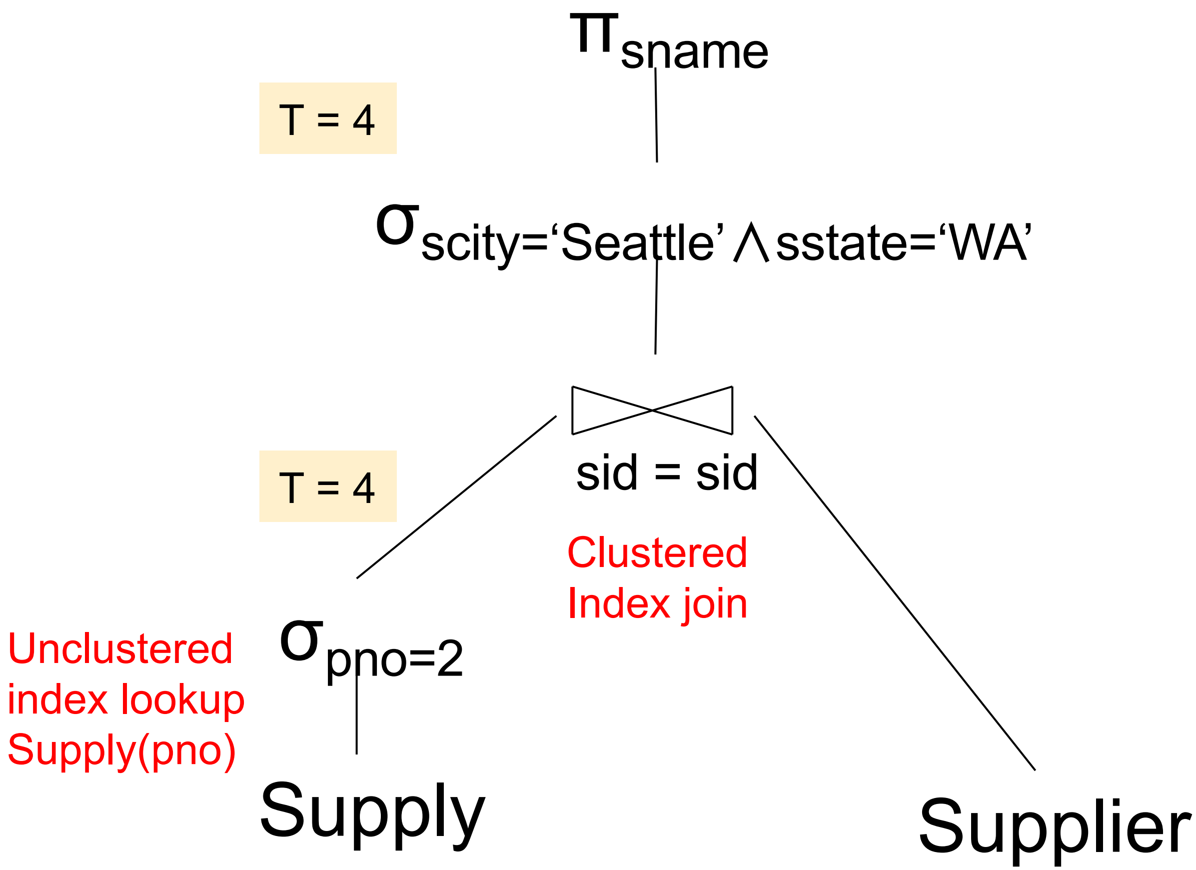
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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:

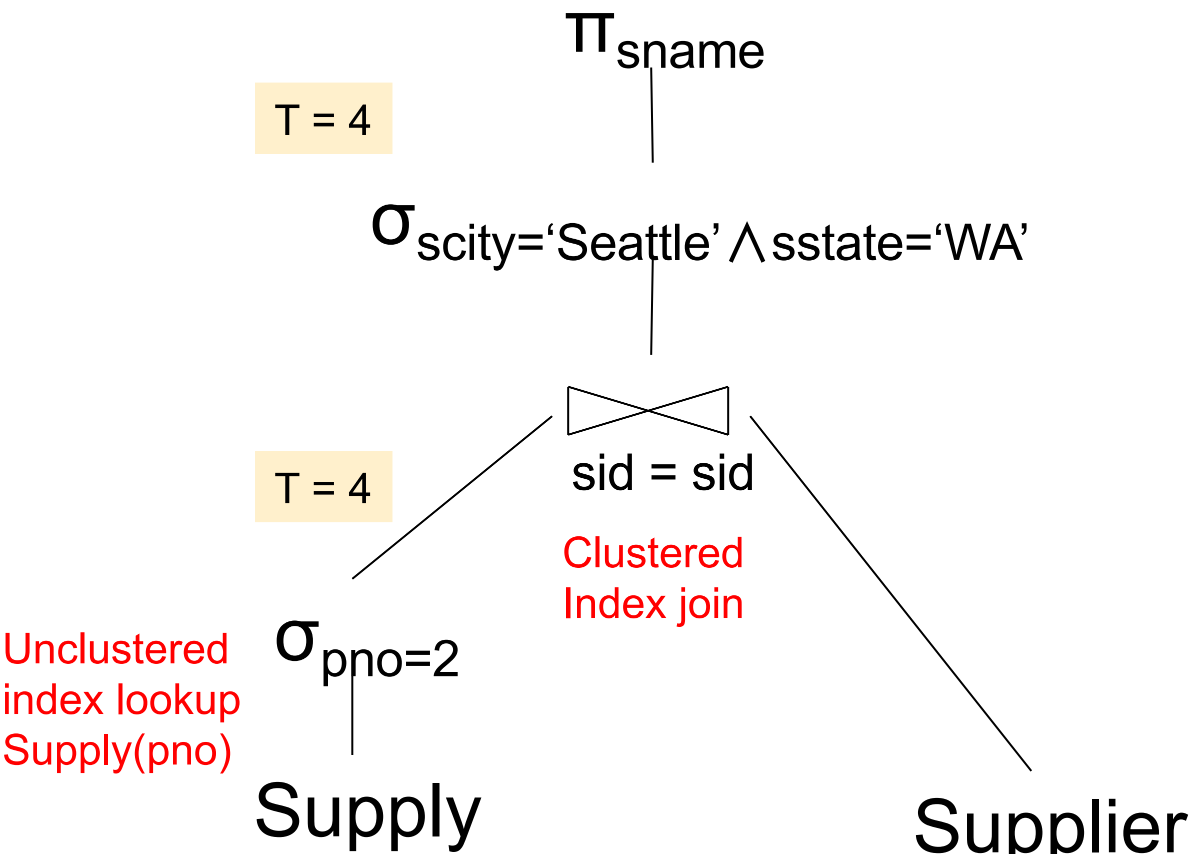
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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; in general we need IO+CPU

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

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

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

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

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(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

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

Discussion

- Cardinality estimation = open problem
- 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 – open problem

Query Optimization

<u>Three major components</u>:

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms

Two Types of Optimizers

- Heuristic-based optimizers
- Cost-based optimizers (next)

Two Types of Plan Enumeration Algorithms

- Dynamic programming (in class)
  - Based on System R [Selinger 1979]
  - *Join reordering algorithm*
- Rule-based algorithm (will not discuss)
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming

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: $Q = \{R_1, \ldots, R_n\}$

Details

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"

Details

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”

Details

Step k = 2…n:

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

- For each j=1,…,k:
  - Let: $Q' = Q - \{R_{i_j}\}$
  - Let: $Plan(Q') \bowtie R_{i_j} \quad Cost(Q') + CostOf(\bowtie)$
- $Plan(Q), Cost(Q) =$ cheapest of the above
  - Keep separate optimal for "interesting orders"

Discussion

- All database systems implement Selinger's algorithm for join reorder
- For other operators (group-by, aggregates, difference): rule-based
- Many search strategies beyond dynamic programming

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
- Next time:
  How good are they?; New approaches