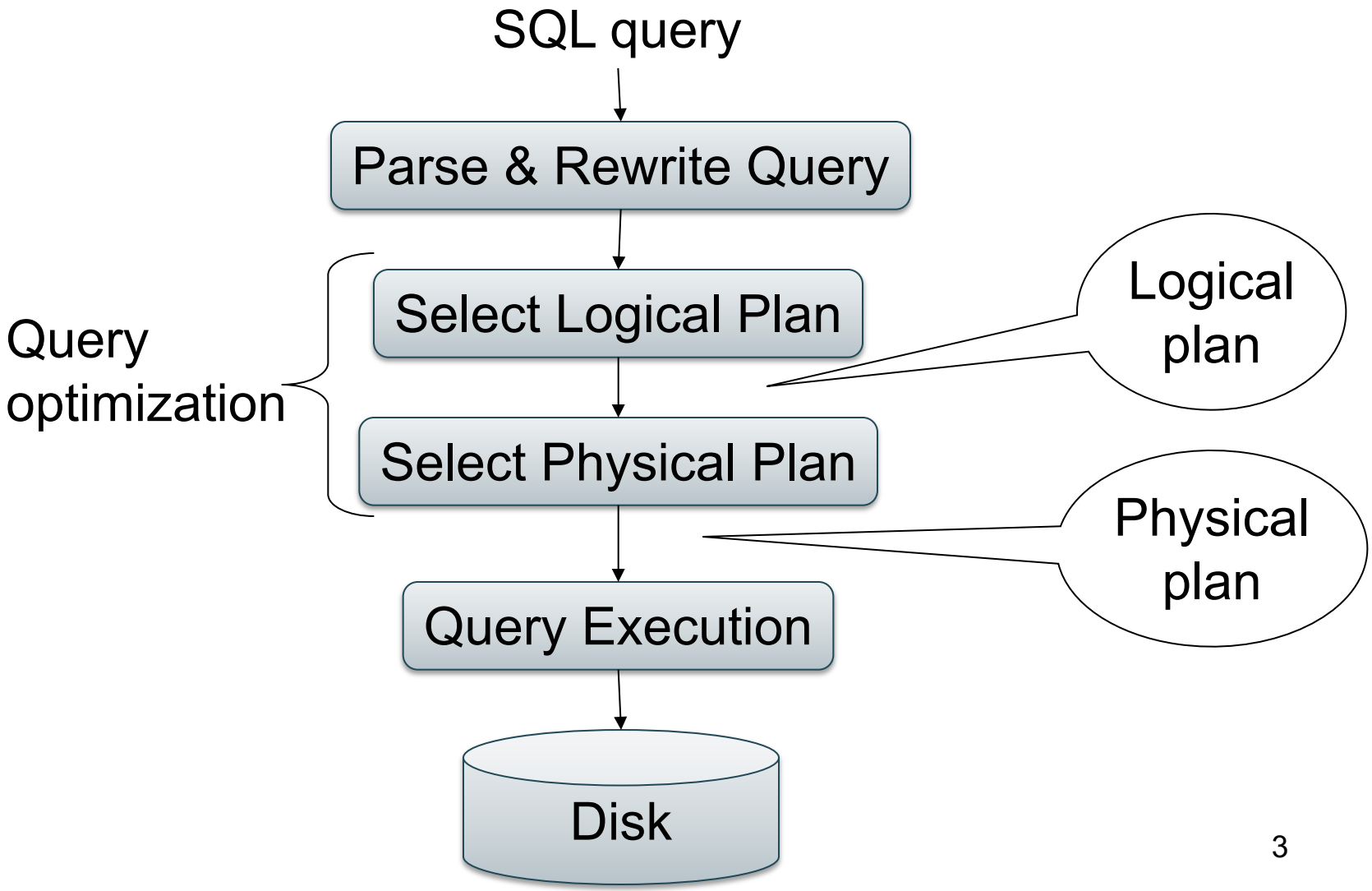
CSE544
Data Management

Lectures 7
Query Execution – Part I

Announcements

- Project proposals due next Monday, 2/5
  - pdf file
  - Latex/bibtex STRONGLY recommended
  - Submit via gilab, in the project directory
- HW2 due next Wednesday, 2/7
- Review 3 due next Wednesday, 2/7

Lifecycle of a Query



Query Rewriting

SQL Parsing and Rewriting

- Parsing -- CSE501
  - Read file w/ SQL commands
  - Check syntax
  - Checks the schema, permissions, etc
  - Build abstract syntax tree
- SQL rewriting
  - Inline view definitions
  - Unnest queries

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Query Rewriting

View: Suppliers in Seattle

```
CREATE VIEW SeattleSupp AS
        SELECT x.sno, x.sname
        FROM Supplier x
        WHERE x.scity='Seattle' AND x.sstate='WA'
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Added to the schema

SeattleSupp(sno, sname)

Query Rewriting

View: Suppliers in Seattle

```
CREATE VIEW SeattleSupp AS
        SELECT x.sno, x.sname
        FROM Supplier x
        WHERE x.scity='Seattle' AND x.sstate='WA'
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting

Find the names of all suppliers in Seattle who supply part number 2

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting

Find the names of all suppliers in Seattle who supply part number 2

```
SELECT x.sno, x.sname
FROM SeattleSupp x
WHERE x.sno IN (SELECT y.sno
                FROM Supply y
                WHERE y.pno = 2 )
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Unnesting

```
SELECT x.sno, x.sname
FROM SeattleSupp x
WHERE x.sno IN (SELECT y.sno
                FROM Supply y
                WHERE y.pno = 2 )
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

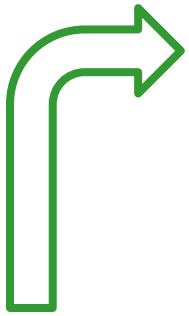
Query Rewriting: Unnesting

```
SELECT x.sno, x.sname
FROM SeattleSupp x
WHERE x.sno IN (SELECT y.sno
                FROM Supply y
                WHERE y.pno = 2 )
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Unnesting



```
SELECT DISTINCT x.sno, x.sname
FROM SeattleSupp x, Supply y
WHERE x.sno = y.sno and y.pno = 2
```

```
SELECT x.sno, x.sname
FROM SeattleSupp x
WHERE x.sno IN (SELECT y.sno
                FROM Supply y
                WHERE y.pno = 2 )
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Unnesting

```
SELECT DISTINCT x.sno, x.sname
FROM SeattleSupp x, Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: View Inlining

```
SELECT DISTINCT x.sno, x.sname
FROM SeattleSupp x, Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: View Inlining

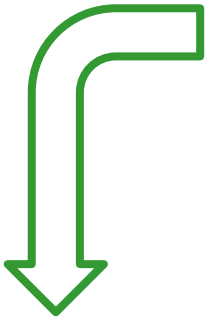
```
SELECT DISTINCT x.sno, x.sname
FROM SeattleSupp x, Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: View Inlining

```
SELECT DISTINCT x.sno, x.sname
FROM SeattleSupp x, Supply y
WHERE x.sno = y.sno and y.pno = 2
```



```
SELECT DISTINCT x.sno, x.sname
FROM (SELECT x.sno, x.sname
      FROM Supplier x
      WHERE x.scity='Seattle' AND x.sstate='WA') x,
     Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: View Inlining

```
SELECT DISTINCT x.sno, x.sname
FROM (SELECT x.sno, x.sname
        FROM Supplier x
        WHERE x.scity='Seattle' AND x.sstate='WA') x,
        Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Unnesting

```
SELECT DISTINCT x.sno, x.sname
FROM (SELECT x.sno, x.sname
        FROM Supplier x
        WHERE x.scity='Seattle' AND x.sstate='WA') x,
        Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

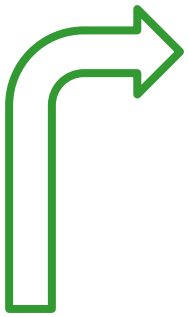
Query Rewriting: Unnesting

```
SELECT DISTINCT x.sno, x.sname
FROM (SELECT x.sno, x.sname
      FROM Supplier x
      WHERE x.scity='Seattle' AND x.sstate='WA') x,
     Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Unnesting



```
SELECT DISTINCT x.sno, x.sname
FROM Supplier x, Supply y
WHERE x.sno = y.sno and y.pno = 2
      and x.scity='Seattle' AND x.sstate='WA'
```

```
SELECT DISTINCT x.sno, x.sname
FROM (SELECT x.sno, x.sname
      FROM Supplier x
      WHERE x.scity='Seattle' AND x.sstate='WA') x,
     Supply y
WHERE x.sno = y.sno and y.pno = 2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

SeattleSupp(sno, sname)

Query Rewriting: Final Query

```
SELECT DISTINCT x.sno, x.sname
FROM Supplier x, Supply y
WHERE x.sno = y.sno and y.pno = 2
      and x.scity='Seattle' AND x.sstate='WA'
```

Query Rewriting

Complex SQL constructs without similar operator in Relational Algebra

- Nested queries
- Correlated queries

Also possible to do this in flavors of the Relational Algebra ("dependent joins")

Relational Algebra

Relational Algebra

- All operators in RA are ”first order”:
  - They take as input sets and returns sets
  - No variables, no nested expressions
- Consequences:
  - Easy to evaluate RA: operator at a time
  - Easy to optimize RA: algebraic identities
  - Hard(er) to translate complex SQL

Five Basic Relational Operators

- Selection $\sigma_{\text{condition}}(S)$
- Projection $\Pi_{\text{list-of-attributes}}(S)$
- Union $\cup$
- Set difference $-$
- Cartesian product $\times$; Join $R \bowtie_\theta S = \sigma_\theta(R \times S)$

Extended Relational Algebra

- Duplicate elimination δ
- Group-by aggregate $\gamma_{attr1,attr2,\ldots,agg1,\ldots}$
- Sort τ

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{\text{sstate}='\text{WA}'}(\text{Supplier})$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{\text{sstate}='\text{WA}'}(\text{Supplier})$

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s555	Walmart	Seattle	WA

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{\text{sstate}='\text{WA}'}(\text{Supplier})$

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s555	Walmart	Seattle	WA

Projection: $\Pi_{\text{sstate}}(\text{Supplier})$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{sstate='WA'}(Supplier)$

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s555	Walmart	Seattle	WA

Projection: $\Pi_{sstate}(Supplier)$

sstate
WA
OR
WA

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{\text{sstate}='\text{WA}'}(\text{Supplier})$

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s555	Walmart	Seattle	WA

Projection: $\Pi_{\text{sstate}}(\text{Supplier})$

sstate
WA
OR
WA

Union: $\text{Table}_1 \cup \text{Table}_2$

Difference $\text{Table}_1 - \text{Table}_2$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Selection: $\sigma_{sstate='WA'}(Supplier)$

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s555	Walmart	Seattle	WA

Projection: $\Pi_{sstate}(Supplier)$

sstate
WA
OR
WA

Union: $Table_1 \cup Table_2$

Difference $Table_1 - Table_2$

$Table_1$ and $Table_2$ must have the same schema

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Join: $\text{Supplier} \bowtie_{\text{Supplier.sno=Supply.sno}} \text{Supply}$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Relational Operators

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Supply:

sno	pno	price
s444	p001	49
s444	p502	199
s555	P502	189
…		

Join: $\text{Supplier} \bowtie_{\text{Supplier.sno=Supply.sno}} \text{Supply}$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Relational Operators

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Supply:

sno	pno	price
s444	p001	49
s444	p502	199
s555	P502	189
…		

Join: $\text{Supplier} \bowtie_{\text{Supplier.sno=Supply.sno}} \text{Supply}$

sno	sname	scity	sstate	sno	pno	price
s444	Walmart	Portland	OR	s444	p001	49
s444	Walmart	Portland	OR	s444	p502	199
s555	Walmart	Seattle	WA	s555	P502	189
…	…					

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Relational Operators

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Supply:

sno	pno	price
s444	p001	49
s444	p502	199
s555	P502	189
…		

Join: $\text{Supplier} \bowtie_{\text{Supplier.sno=Supply.sno}} \text{Supply}$

sno	sname	scity	sstate	sno	pno	price
s444	Walmart	Portland	OR	s444	p001	49
s444	Walmart	Portland	OR	s444	p502	199
s555	Walmart	Seattle	WA	s555	P502	189
…	…					

Ambiguous schema: two sno's

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Group by: $\gamma_{\text{sstate,count}(*)}(\text{Supplier})$

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Group by: $\gamma_{\text{sstate,count}(*)}(\text{Supplier})$

sstate	count(*)
WA	2
OR	1

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Supplier:

sno	sname	scity	sstate
s333	ACME	Seattle	WA
s444	Walmart	Portland	OR
s555	Walmart	Seattle	WA

Relational Operators

Group by: $\gamma_{\text{sstate,count}(*)}(\text{Supplier})$

sstate	count(*)
WA	2
OR	1

Duplicate elimination: δ

Relational Operators

- Bag semantics!
- Many variants of joins:
  - Natural join: equality on common attributes
  - Left/Right/Full Outer Join: ⟕, ⟖, ⟗
  - Semi-join: ⋉ (in a later lecture)
  - Anti-semi-join: …
  - Can use other than equality condition

Logical and Physical Plans

SQL to Logical Plan

- SQL query is converted to a logical plan
- The initial plan can be naïve; we will discuss later how to optimize it

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

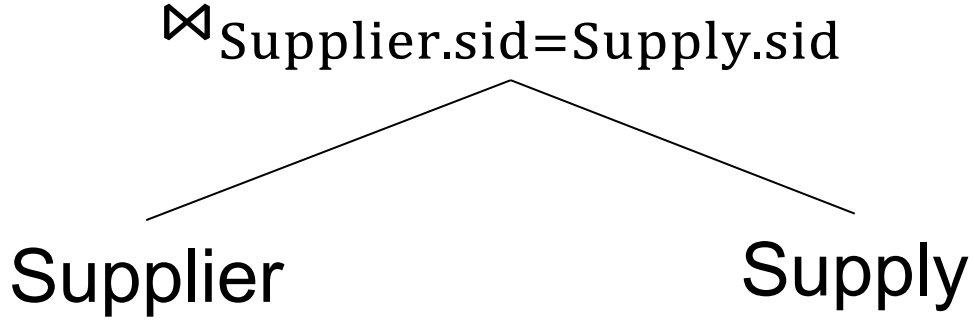
Logical Query Plan

```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Logical Query Plan

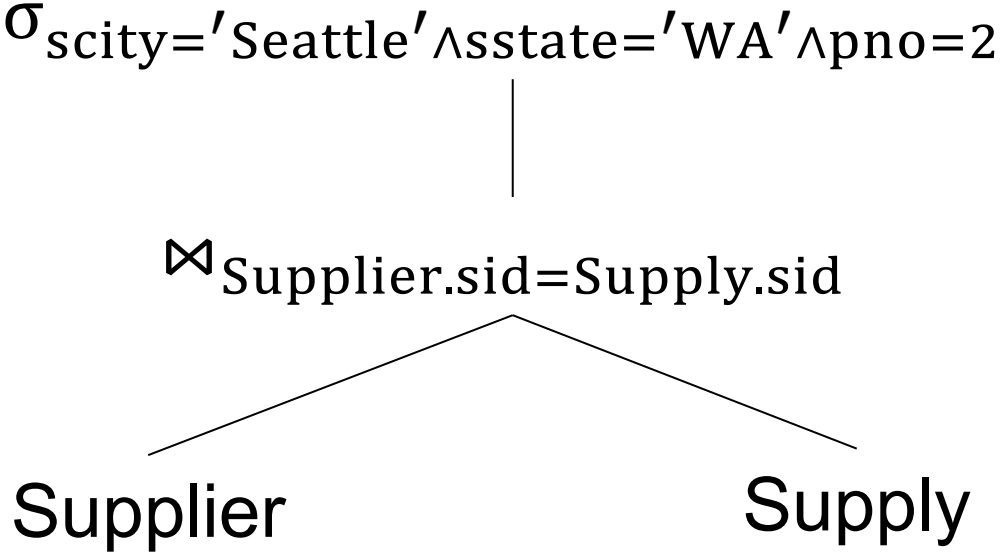
```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Logical Query Plan

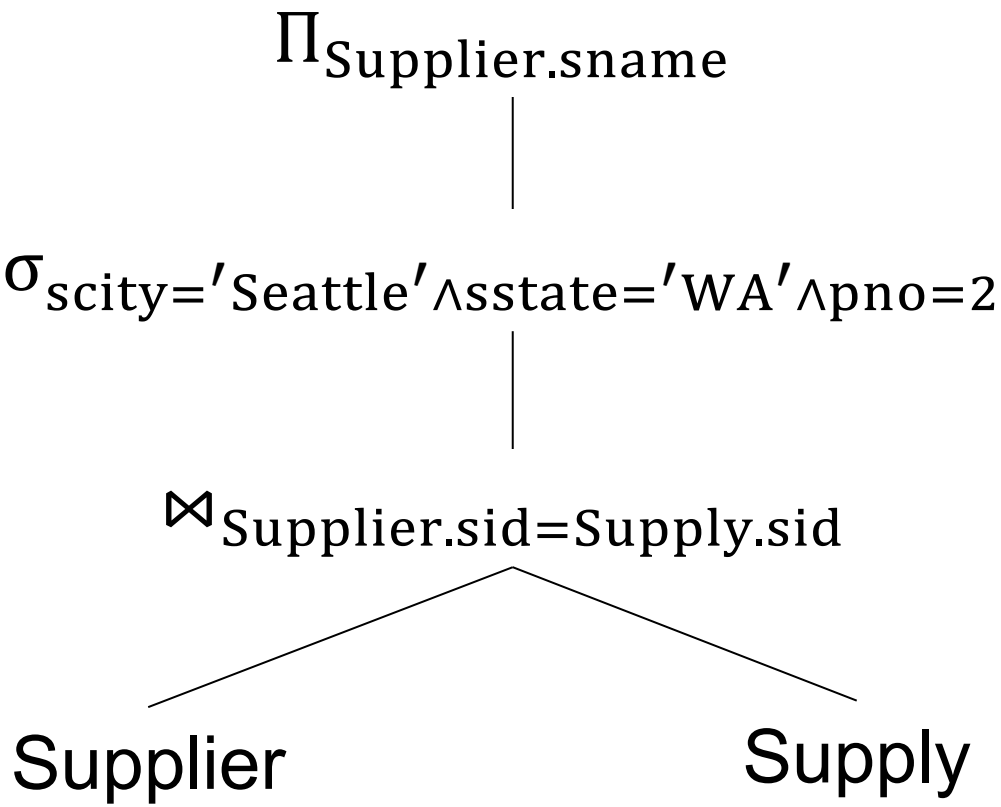
```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Logical Query Plan

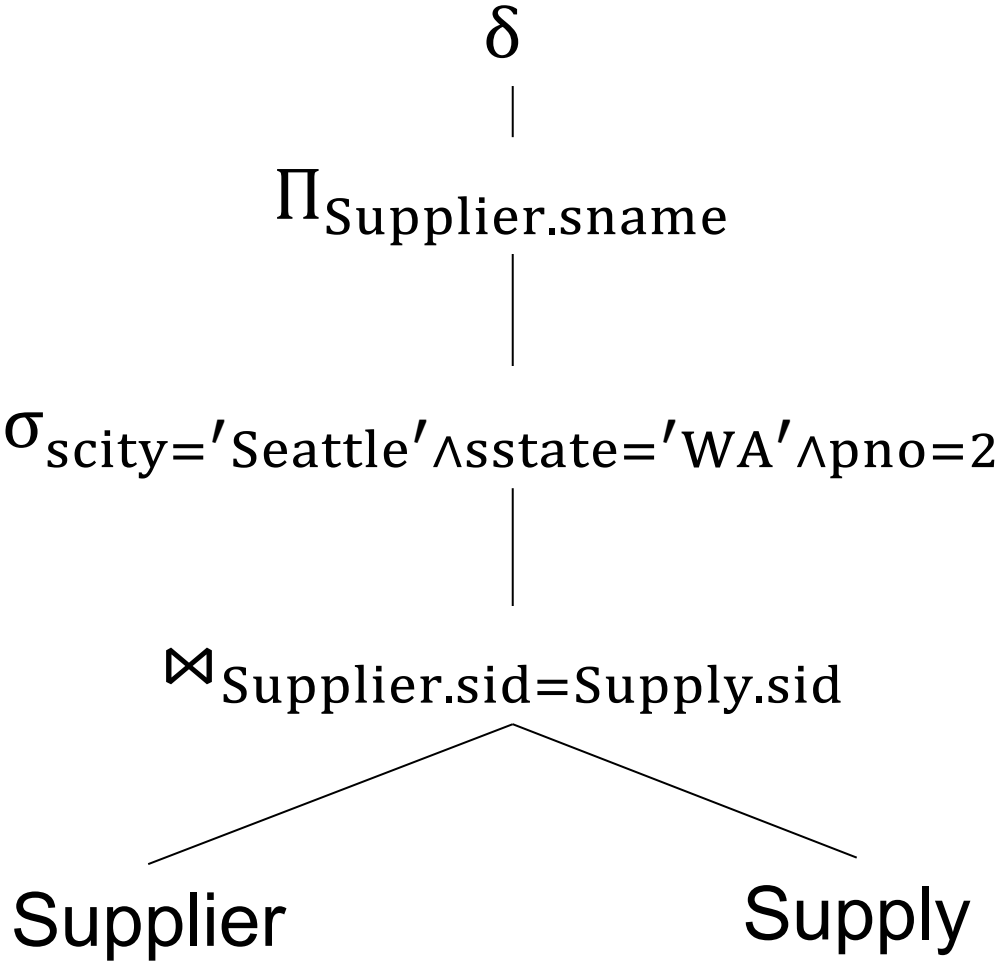
```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Logical Query Plan

```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Notation Convention

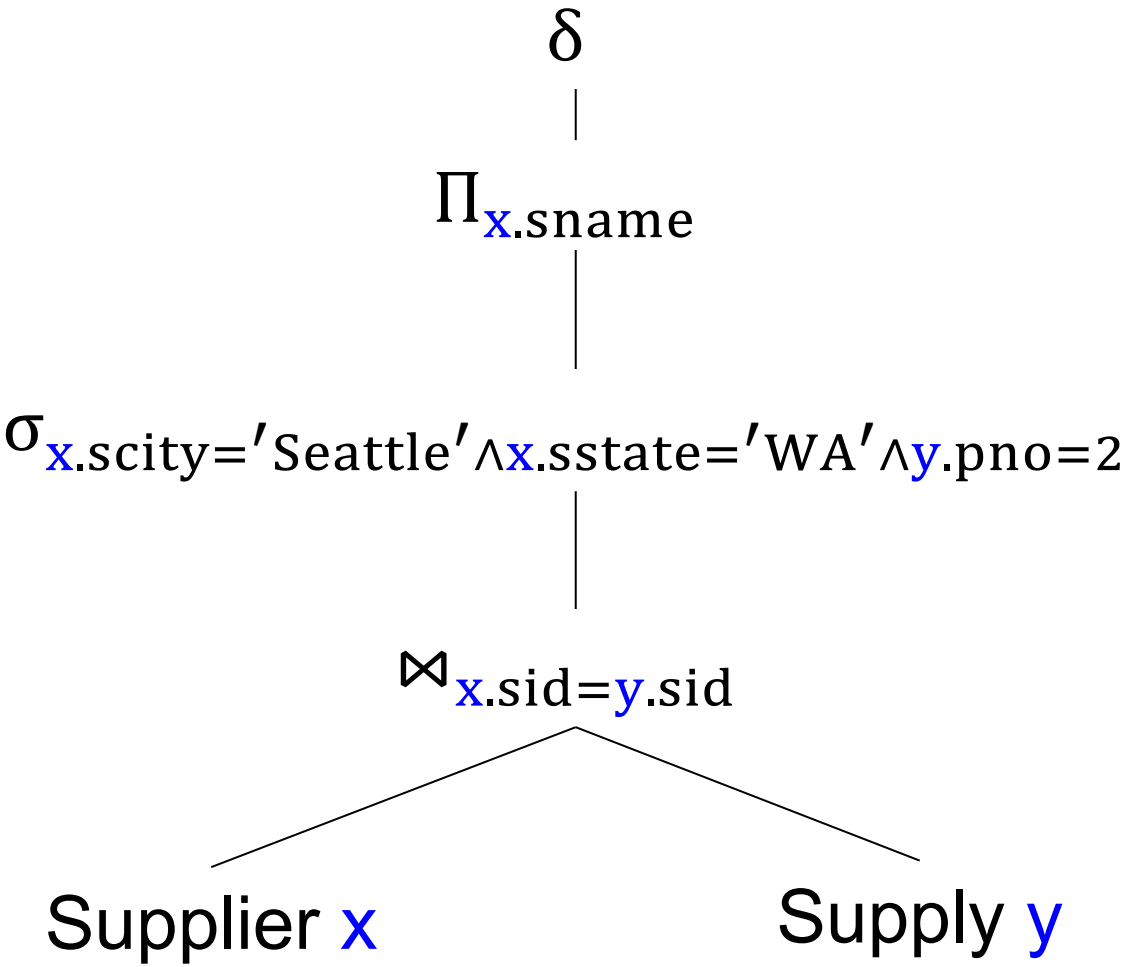
- Database systems have an internal way to represent the schema of the intermediate results, e.g. to deal with duplicate attribute names

- We use a simple convention: add tuple variables to the leaves, refer to them

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Logical Query Plan

```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Physical Query Plan

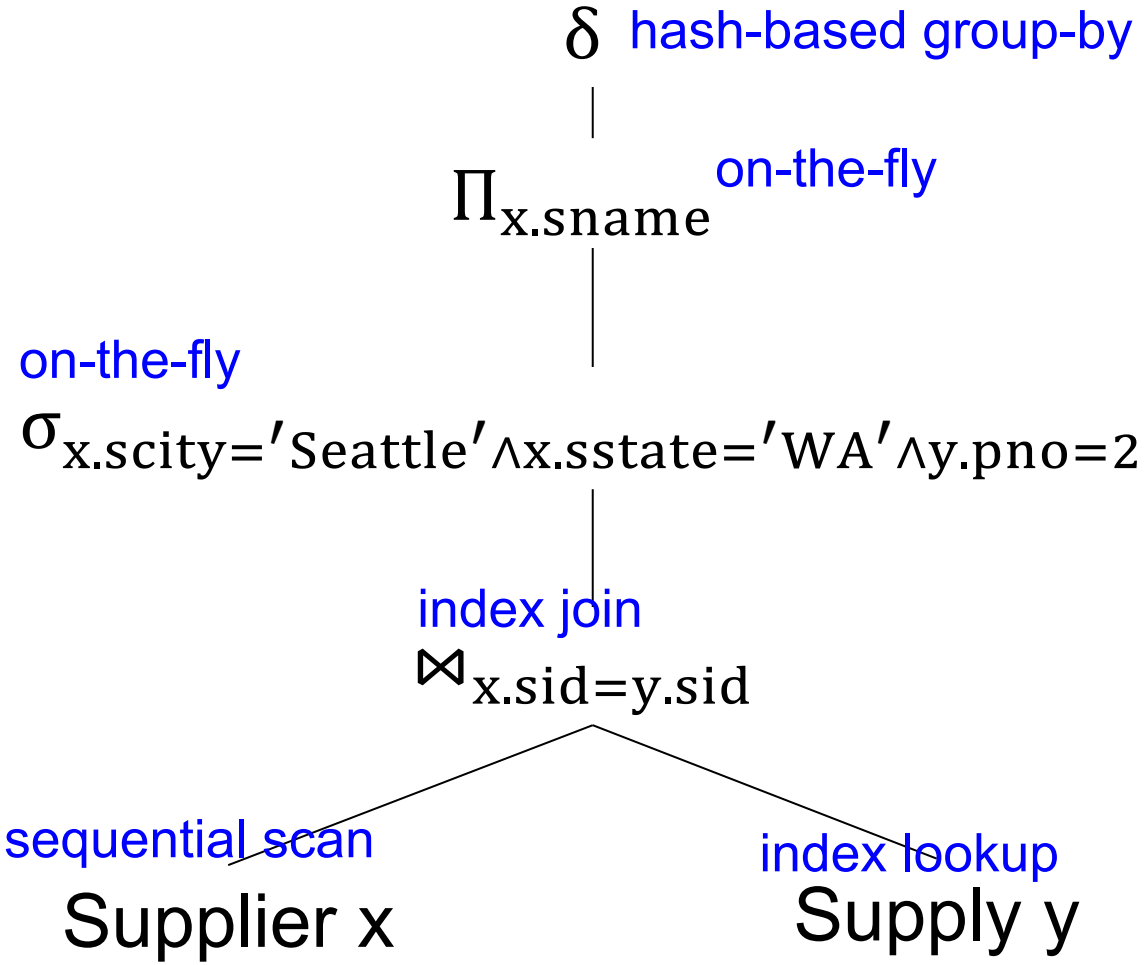
Annotate each operator with an algorithm that implements it

- Operator+algorithm = physical operator
- Annotated plan = physical plan

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Physical Query Plan

```
SELECT DISTINCT x.sname
FROM Supplier x, Supply y
WHERE x.sno=y.sno
      and x.scity='Seattle'
      and x.sstate='WA'
      and y.pno=2
```



Discussion

- Relational algebra does not have subqueries:
  - You can't put a subquery in σ
- Instead need to unnest
- Complications of unnesting:
  - Correlated subqueries -- next
  - May need outer joins

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

Find all suppliers in ‘WA’ that supply *only* parts under $100

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Find all suppliers in 'WA' that supply *only* parts under $100

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Find all suppliers in 'WA' that supply *only* parts under $100

Translate to RA

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

$$\sigma_{\text{x.sstate}='\text{WA}'\wedge\neg\exists(\ldots)}$$

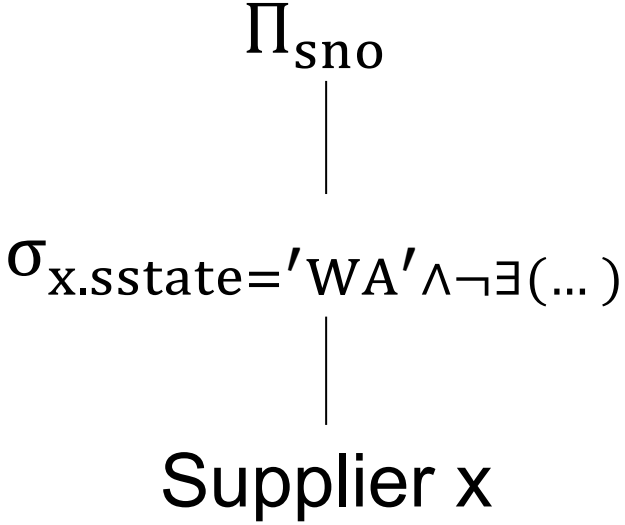
|

Supplier x

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

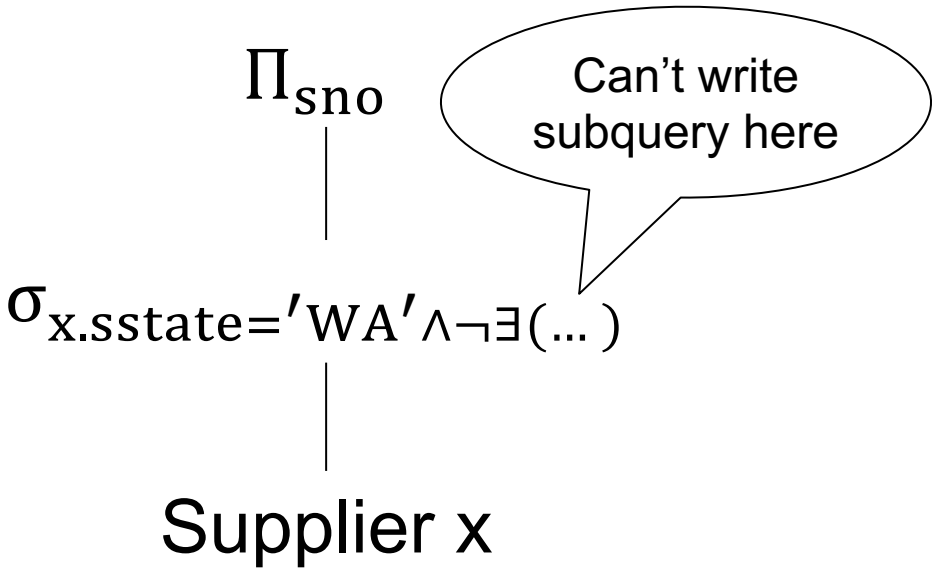
```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

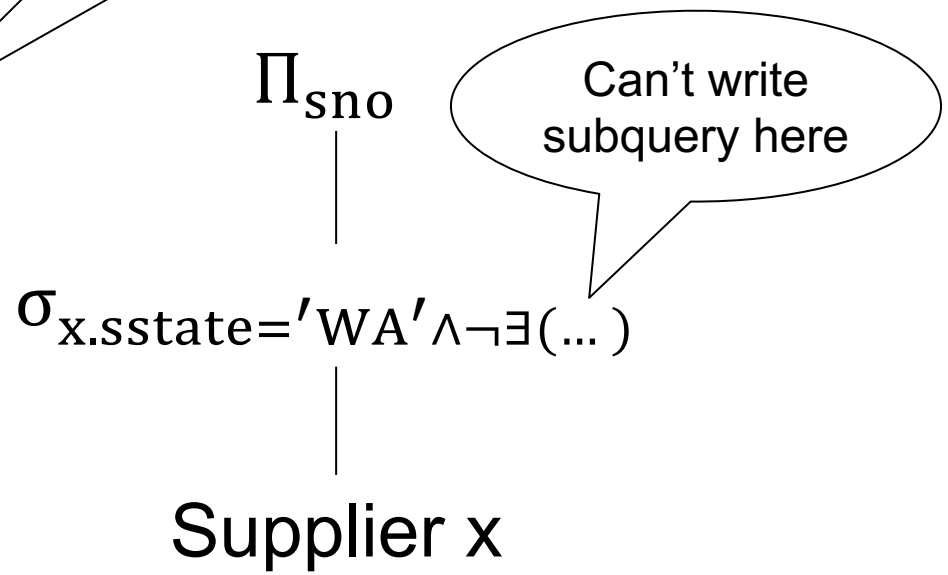


Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Need to unnest



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Need to unnest

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

Need to unnest

Correlation !

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```

De-Correlation

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and x.sno not in
       (SELECT y.sno
        FROM Supply y
        WHERE y.price > 100)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

Un-nesting

```
(SELECT  x.sno
 FROM Supplier x
 WHERE  x.sstate = 'WA')
   EXCEPT
 (SELECT y.sno
  FROM Supply y
  WHERE y.price > 100)
```

```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
   and x.sno not in
      (SELECT y.sno
       FROM Supply y
       WHERE y.price > 100)
```

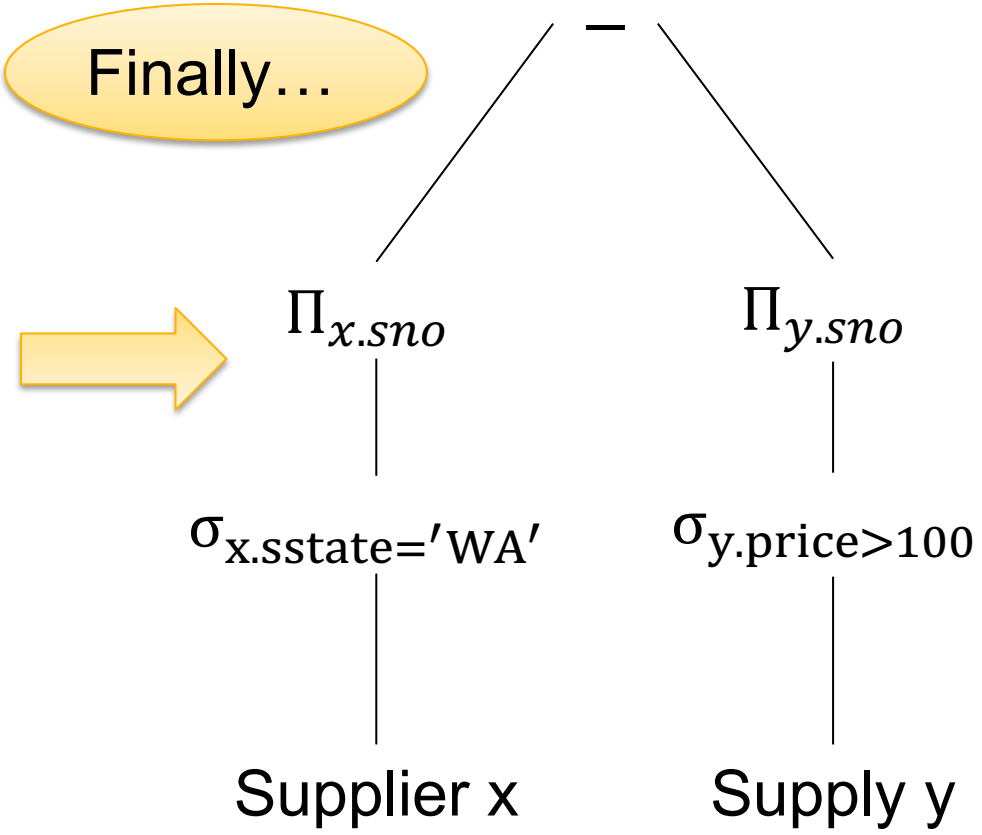
EXCEPT = set difference

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Decorrelation

```
(SELECT  x.sno
 FROM Supplier x
 WHERE  x.sstate = 'WA')
   EXCEPT
 (SELECT y.sno
   FROM Supply y
   WHERE y.price > 100)
```

Finally…



Discussion

- Every SQL query is translated to RA
- RA includes some exotic ops:
  - anti-semi-join (in class)
- Unnesting/decorrelation: difficult task
  - Not all DBMS do this well
  - May introduce left outer joins
  - A comprehensive paper on this:

    https://dl.gi.de/items/137d6917-d8fe-43aa-940b-e27da7c01625

Physical Operators

Physical Operators

- For each operator, several algorithms
- Main memory or external memory

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Main Memory Algorithms

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

How would you compute the join?

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Main Memory Algorithms

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

How would you compute the join?

Three physical operators:

1. Nested Loops
2. Hash-join
3. Merge-join

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Main Memory Algorithms

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Three physical operators:

1. Nested Loops
2. Hash-join
3. Merge-join

Terminology:

$$R \bowtie S$$

R is the outer table

S is the inner table

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Nested Loop Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in Supply do
      if x.sno = y.sno
         then output(x,y)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Nested Loop Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in Supply do
      if x.sno = y.sno
         then output(x,y)
```

If |R|=|S|=n,
what is the runtime?

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Nested Loop Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in Supply do
      if x.sno = y.sno
         then output(x,y)
```

If |R|=|S|=n,
what is the runtime?

$O(n^2)$

1. Nested Loop Join

- Simplest implementation of join

- Variations:
  - Index join: we have an index on inner table
  - Block nested loop join (next time)

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Index Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in SupplyIndex(x.sno) do
      output(x,y)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Index Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in SupplyIndex(x.sno) do
      output(x,y)
```

If |R|=|S|=n, what is the runtime?

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Index Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in SupplyIndex(x.sno) do
      output(x,y)
```

If |R|=|S|=n, what is the runtime?

O(n)

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

1. Index Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
for x in Supplier do
   for y in SupplyIndex(x.sno) do
      output(x,y)
```

If |R|=|S|=n, what is the runtime?

O(n)

When data is on disk
then big difference between
clustered and unclustred

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Clustered v.s. Unclustered

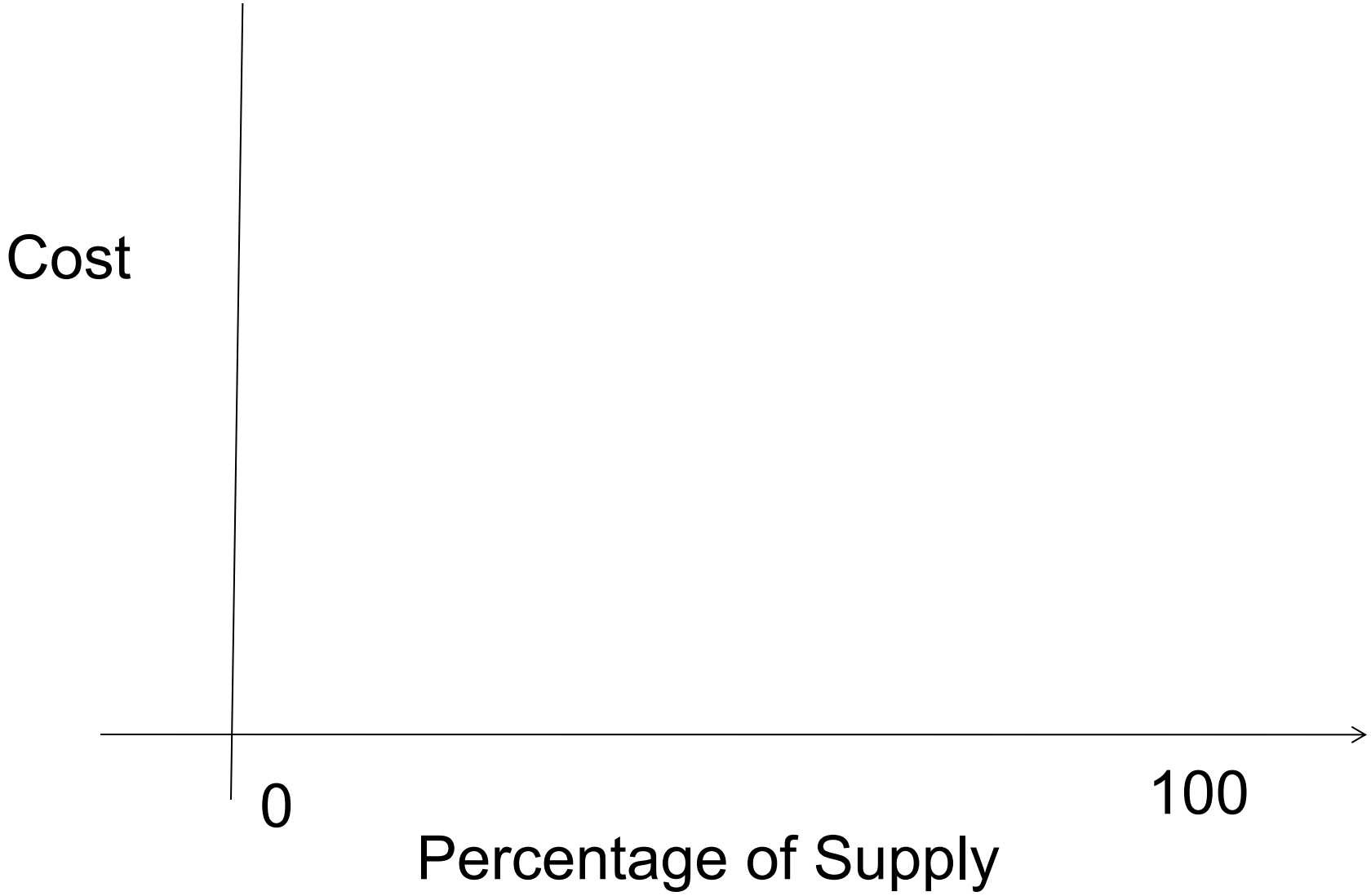
Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

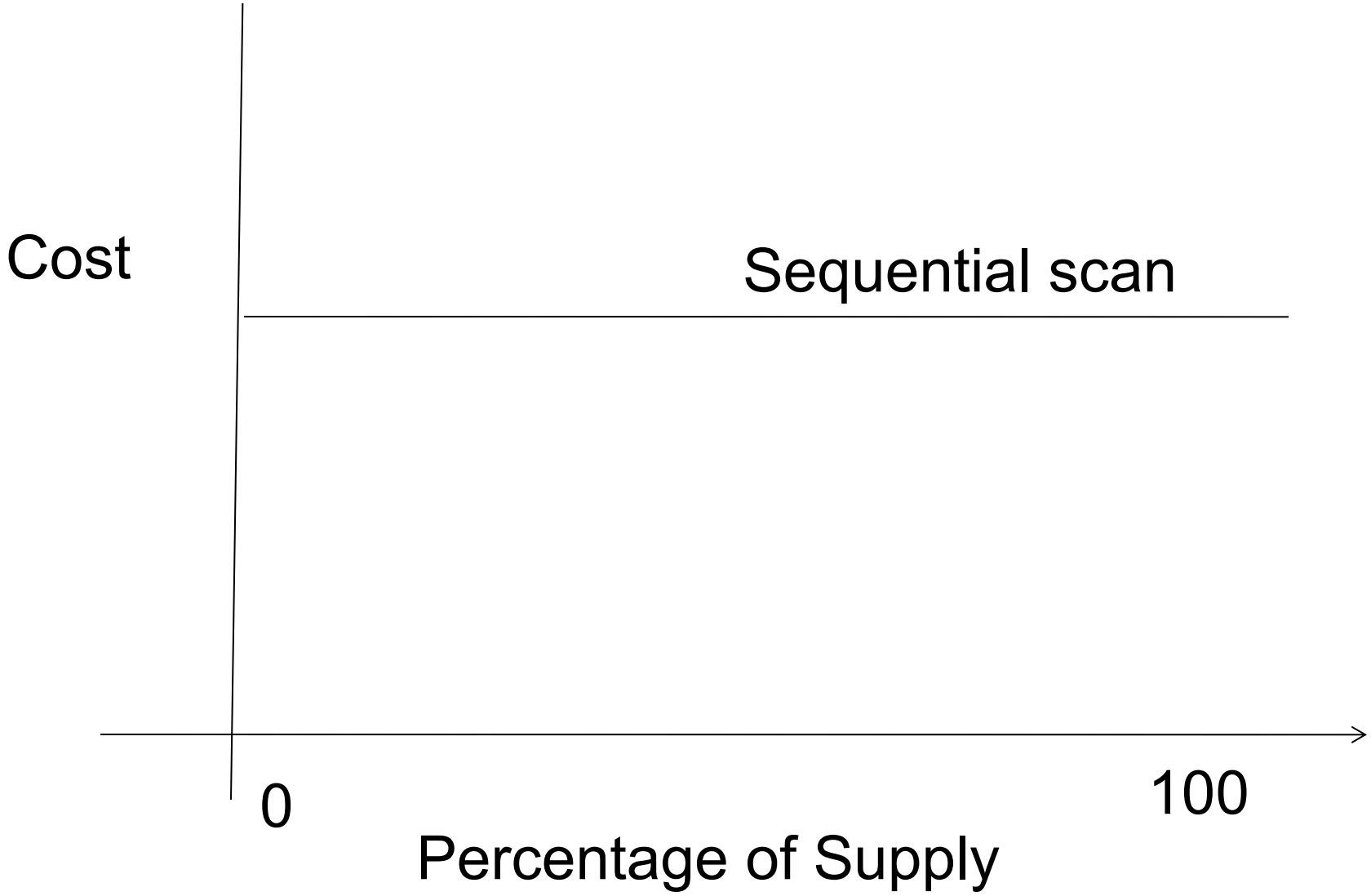
```
for x in Supplier do
   for y in SupplyIndex(x.sno) do
      output(x,y)
```

Rule of thumb:
Random reading
1-2% of Supply ≈
sequential scan
entire file

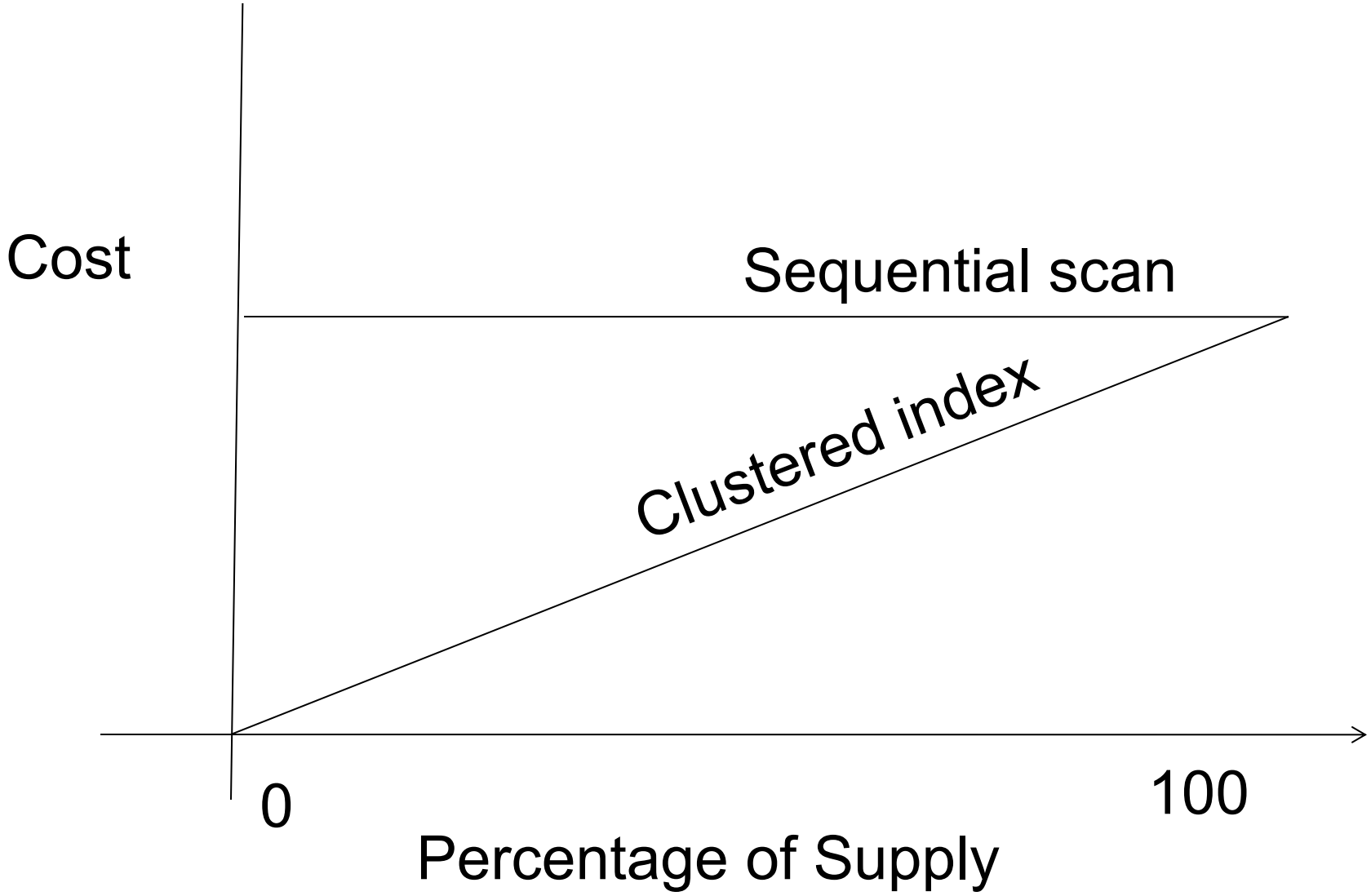
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)



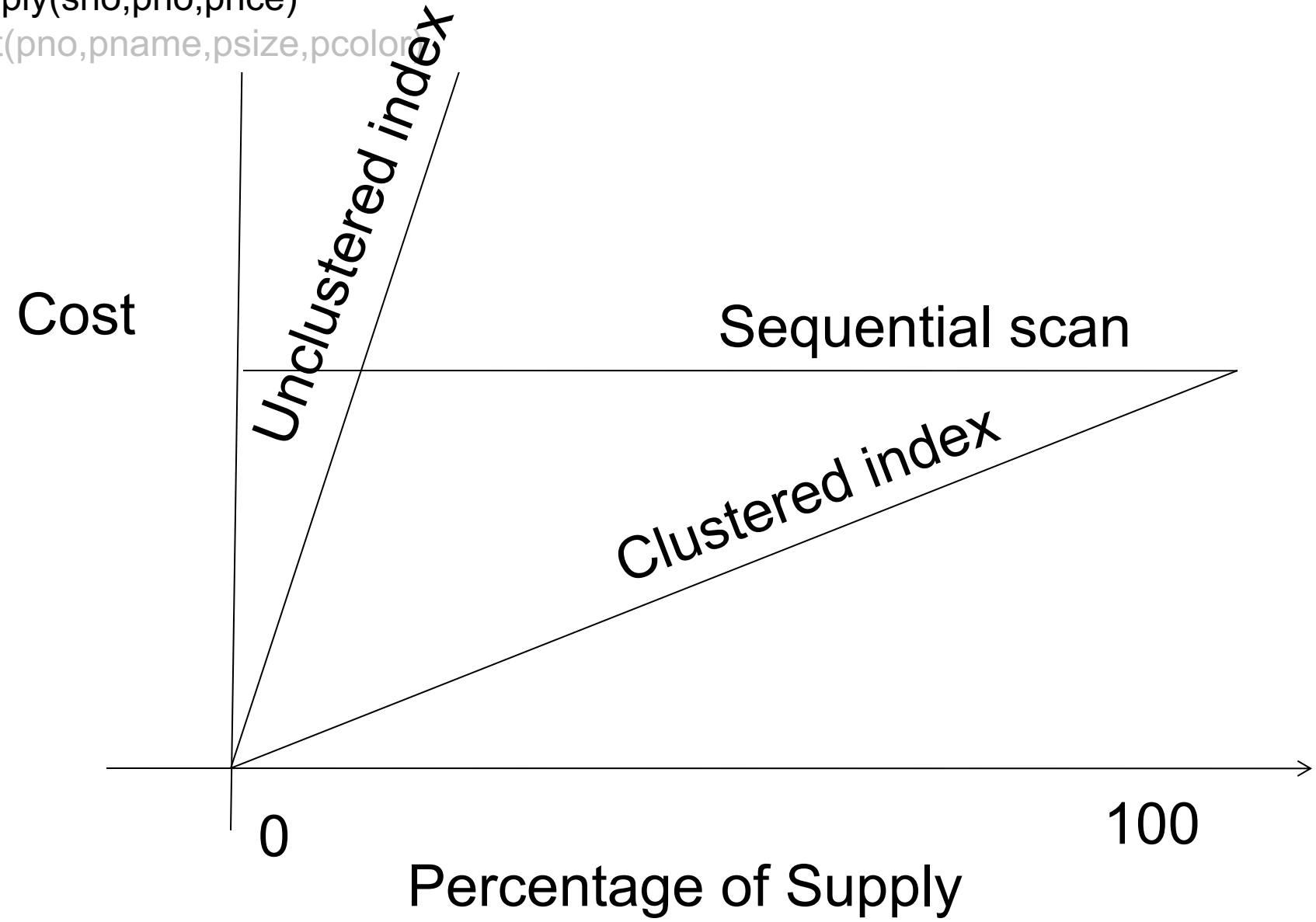
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)



Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Describe it!

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Build phase

```
for x in Supplier do
    insert(x.sno, x)
```

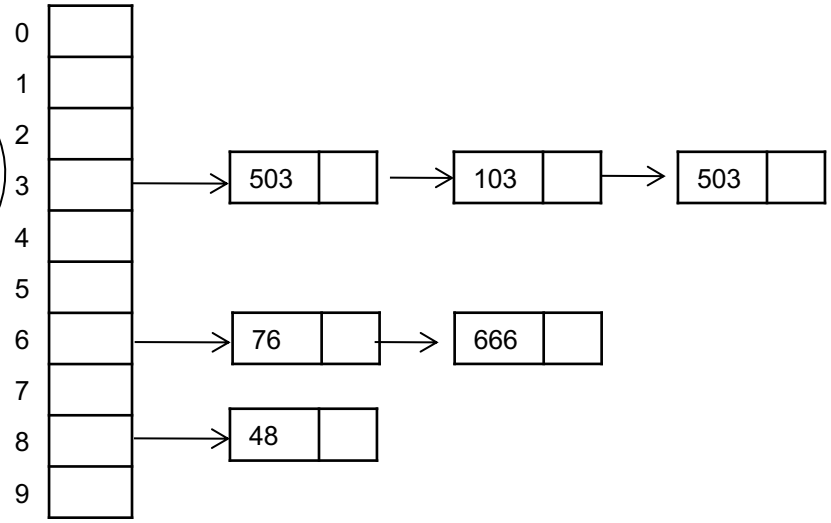
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Build phase




```
for x in Supplier do
    insert(x.sno, x)
```

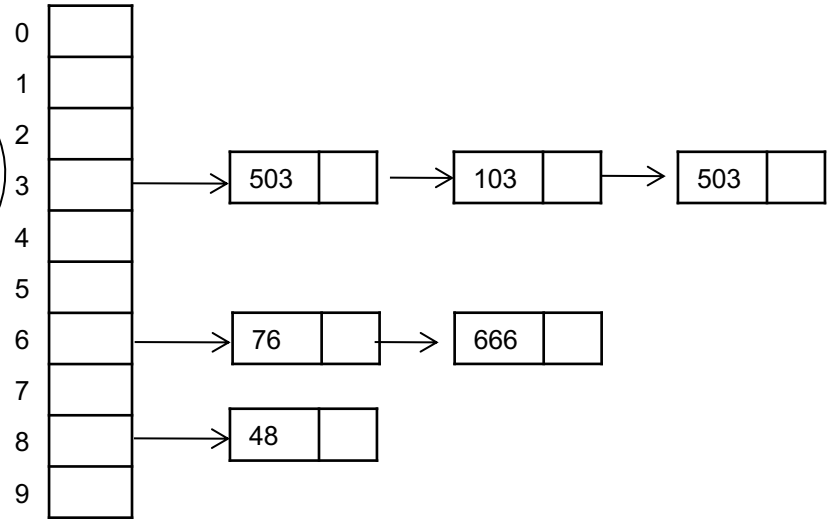
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Build phase




```
for x in Supplier do
      insert(x.sno, x)

for y in Supply do
      x = find(y.sno);
      output(x,y);
```

Probe phase

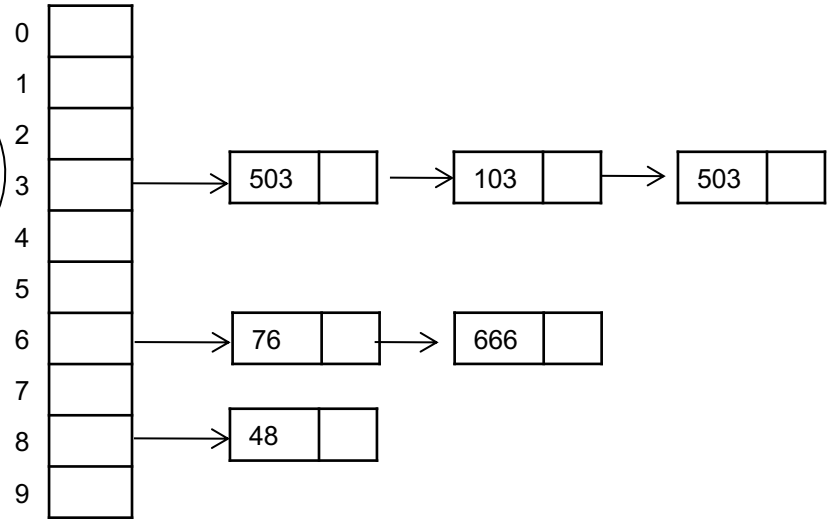
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Build phase




```
for x in Supplier do
      insert(x.sno, x)

for y in Supply do
      x = find(y.sno);
      output(x,y);
```

Probe phase

If |R|=|S|=n,
what is the runtime?

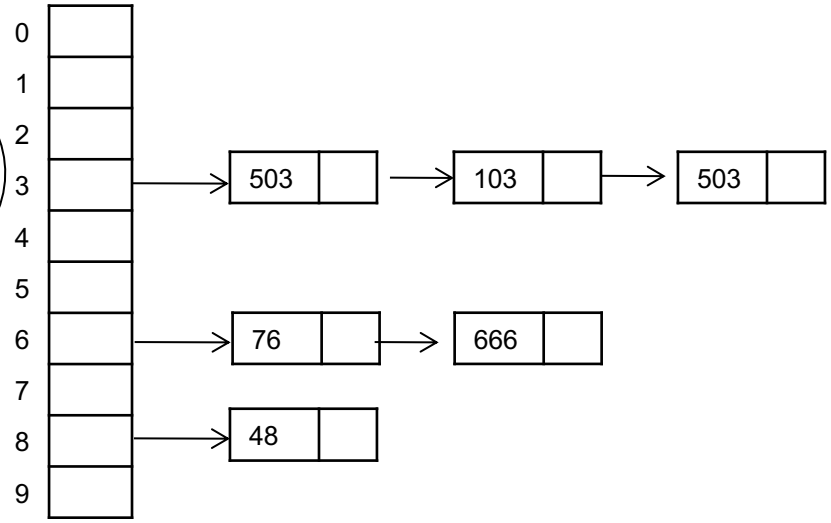
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Build phase




```
for x in Supplier do
      insert(x.sno, x)

for y in Supply do
      x = find(y.sno);
      output(x,y);
```

Probe phase

If |R|=|S|=n,
what is the runtime?

O(n)

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Change join order

Logical operator:

$$\text{Supply} \bowtie_{\text{sno=sno}} \text{Supplier}$$

```
for y in Supply do
        insert(y.sno, y)

for x in Supplier do
        ????
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Change join order

Logical operator:

$$\text{Supply} \bowtie_{\text{sno=sno}} \text{Supplier}$$

```
for y in Supply do
        insert(y.sno, y)

for x in Supplier do
        ????
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

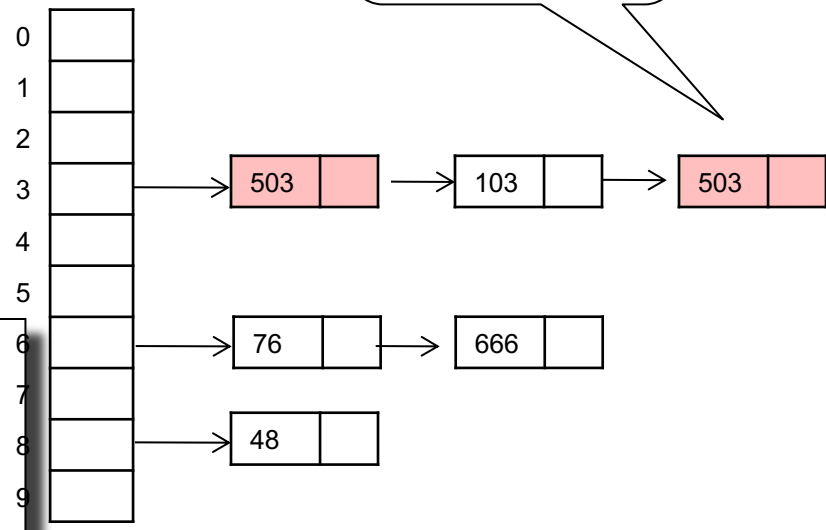
2. Hash Join

Change join order

Duplicate sno's

Logical operator:

$$\text{Supply} \bowtie_{\text{sno=sno}} \text{Supplier}$$




```
for y in Supply do
      insert(y.sno, y)

for x in Supplier do
    for y in find(x.sno) do
          output(x,y);
```

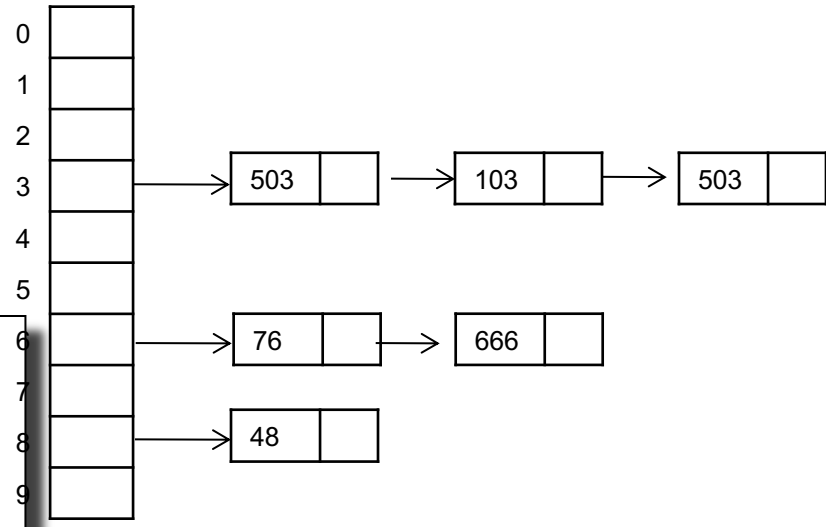
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Change join order

Logical operator:

$$\text{Supply} \bowtie_{\text{sno=sno}} \text{Supplier}$$




```
for y in Supply do
      insert(y.sno, y)

for x in Supplier do
    for y in find(x.sno) do
          output(x,y);
```

If |R|=|S|=n, what is the runtime?

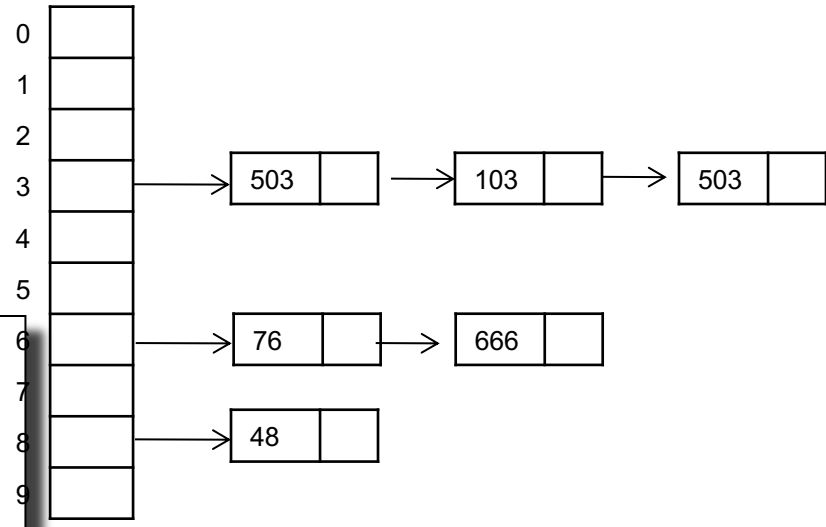
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

2. Hash Join

Change join order

Logical operator:

$$\text{Supply} \bowtie_{\text{sno=sno}} \text{Supplier}$$




```
for y in Supply do
      insert(y.sno, y)

for x in Supplier do
    for y in find(x.sno) do
        output(x,y);
```

If |R|=|S|=n, what is the runtime?

O(n)

But can be $O(n^2)$ why?

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

Describe it!

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: ???
    x.sno = y.sno: ???
    x.sno > y.sno: ???
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: x = x.next()
    x.sno = y.sno: ???
    x.sno > y.sno: ???
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: x = x.next()
    x.sno = y.sno: output(x,y); y = y.next();
    x.sno > y.sno: ???
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: x = x.next()
    x.sno = y.sno: output(x,y); y = y.next();
    x.sno > y.sno: y = y.next();
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: x = x.next()
    x.sno = y.sno: output(x,y); y = y.next();
    x.sno > y.sno: y = y.next();
```

If |R|=|S|=n,
what is the runtime?

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

3. Merge Join

Logical operator:

$$\text{Supplier} \bowtie_{\text{sno=sno}} \text{Supply}$$

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sno < y.sno: x = x.next()
    x.sno = y.sno: output(x,y); y = y.next();
    x.sno > y.sno: y = y.next();
```

If |R|=|S|=n,
what is the runtime?

O(n log(n))

Summary

- Each operator can have several implementations
- Join = the most important and most complex: nested loops, hash, merge
- Other operators (group-by, union, difference) use variations of nested loops, hash, merge
- When data is on disk, specialized operators are needed