Introduction to Data Structure (Data Management) Lecture 12

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  – University ID Num Name (no “(   )“)
  – Ex: 202054321 Juan Dela Cruz
  –
  – Not changing your name to this format
    - you might be marked Absent

  * → absent?

- Relational Algebra (Part 2)
- Query Evaluation
- From Logical Plans to Physical Plans

INTRO TO DATA STRUCTURE

Relational Algebra (Part 2)

So which join is it?

- Theta join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross product followed by a selection θ

So which join is it?

- Theta join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross product followed by a selection θ
- Equijoin: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join condition θ consist only of equalities

So which join is it?

- Theta join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross product followed by a selection θ
- Equijoin: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join condition θ consist only of equalities
- Natural join1: $R \bowtie S = \pi_A(\sigma_{\theta}(R \times S))$
  - Equijoin
  - Equality on all fields with same name in R and in S
  - Projection π_A drops all redundant attributes

So which join is it?

- When we use $R \bowtie S$,
  - we usually mean an equijoin
  - but often omit the equality predicate when it is clear from the context

More Joins

- Outer Join
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
  - Does not eliminate/remove duplicate columns

- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example (Review)

Lecture

Id	Name	Room	Class
1	S1	406	Discrete Math
2	S2	408	Data Struc
3	S3	409	Programming

Research

Project	Focus	Room
Brain	Deep Learning	406
Ships	Sensors	408

L ⟕ R

Outer Join Example (Review)

Lecture 4

Id	Name	Room	Class
1	S1	406	Discrete Math
2	S2	408	Data Struc
3	S3	409	Programming

Research

Project	Focus	Room
Brain	Deep Learning	406
Ships	Sensors	408

$L ⟕ R$

L.Id	L.Name	L.Room	L.Class	R.Project	R.Focus	R.Room
1	S1	406	Discrete Math	Brain	Deep Learning	406
2	S2	408	Data Struc	Ships	Sensors	408
3	S3	409	Programming	Null	Null	Null

More Examples

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

a. Name of supplier of parts with size greater than 10

a. Name of supplier of red parts or parts with size greater than 10

More Examples

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

a. Name of supplier of parts with size greater than 10

$\Pi_{sname}(Supplier \bowtie Supply \bowtie (\sigma_{psize>10})(Part))$

a. Name of supplier of red parts or parts with size greater than 10

$\Pi_{sname}(Supplier \bowtie Supply \bowtie (\sigma_{pcolor='red' \text{ OR } psize>10}(Part)))$

More Examples

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

a. Name of supplier of parts with size greater than 10

$\Pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10})(\text{Part}))$

a. Name of supplier of red parts or parts with size greater than 10

∪

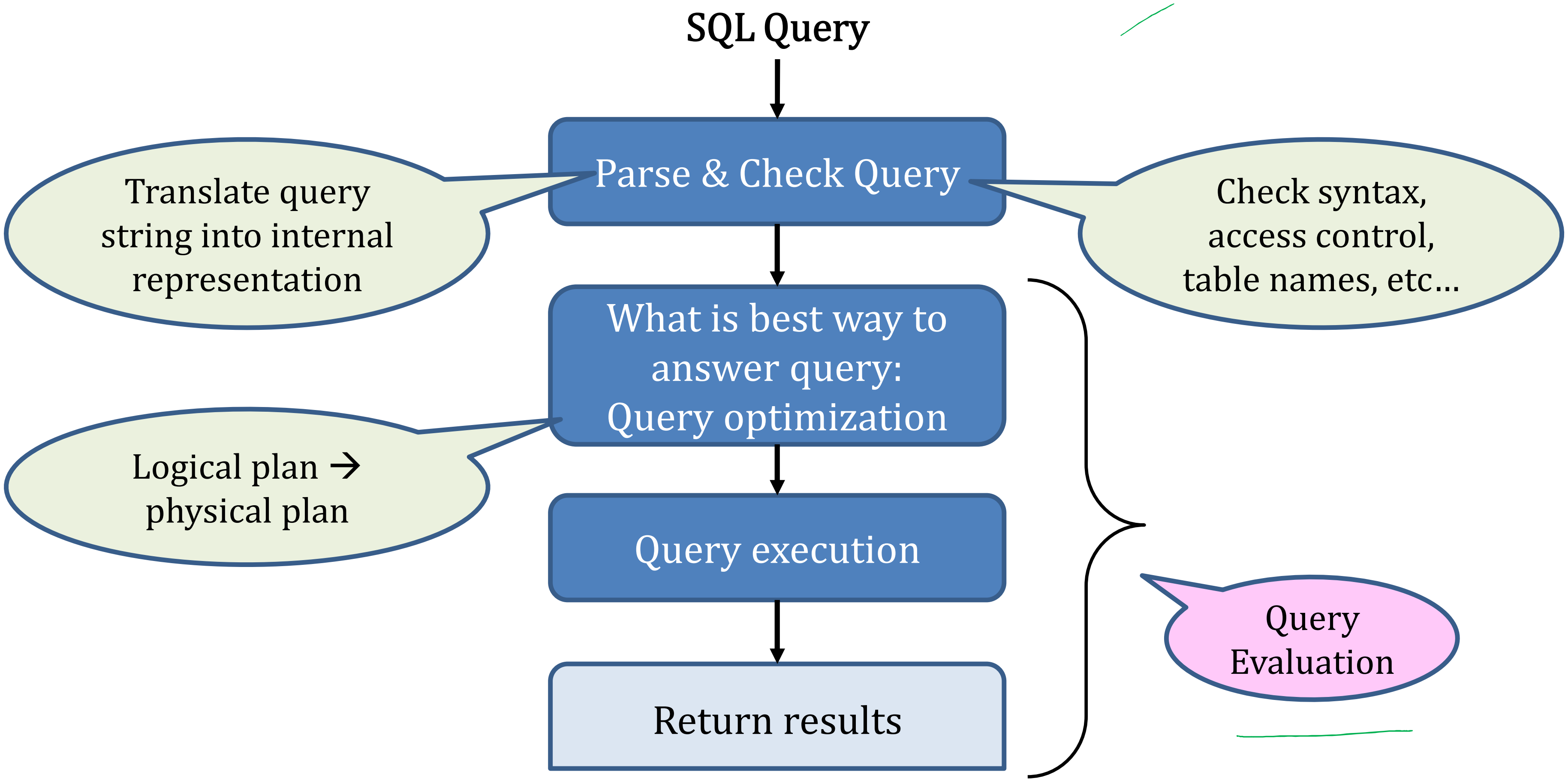
$\Pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10})(\text{Part}) \cup \sigma_{'color='red'} (\text{Part}))$

∩

INTRO TO DATA STRUCTURE

Query Evaluation

Query Evaluation Steps



```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```

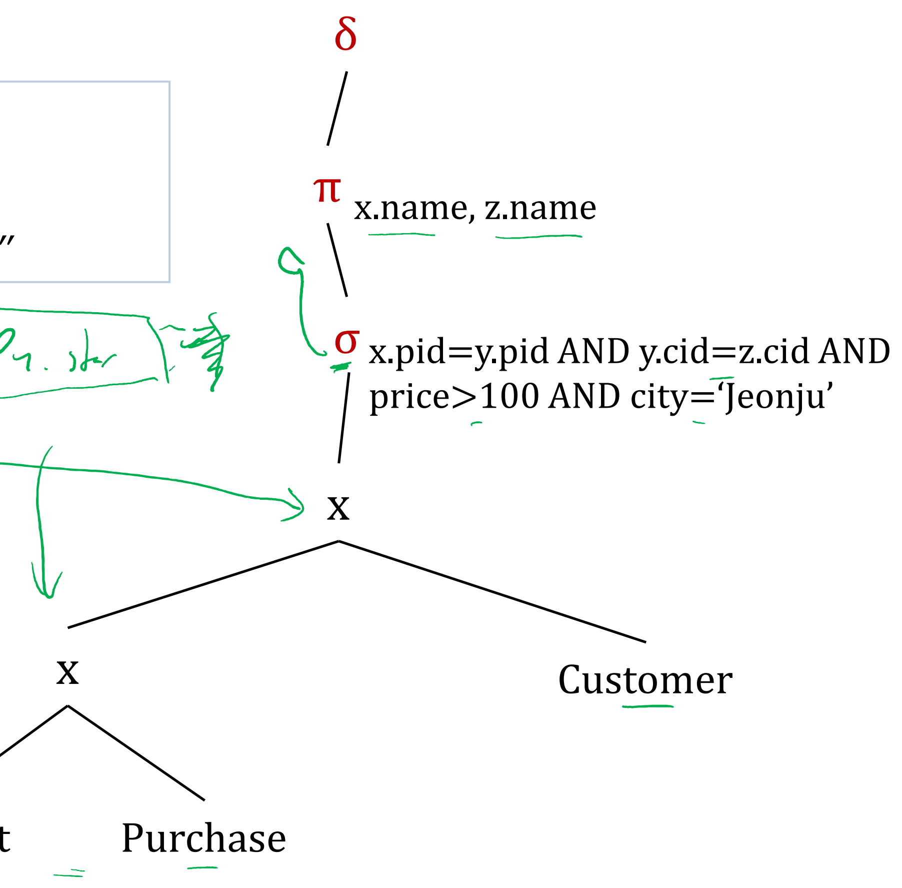
```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```

P.pid | P.name | P.price | Pu.pid | Pu.cid | Pu.store

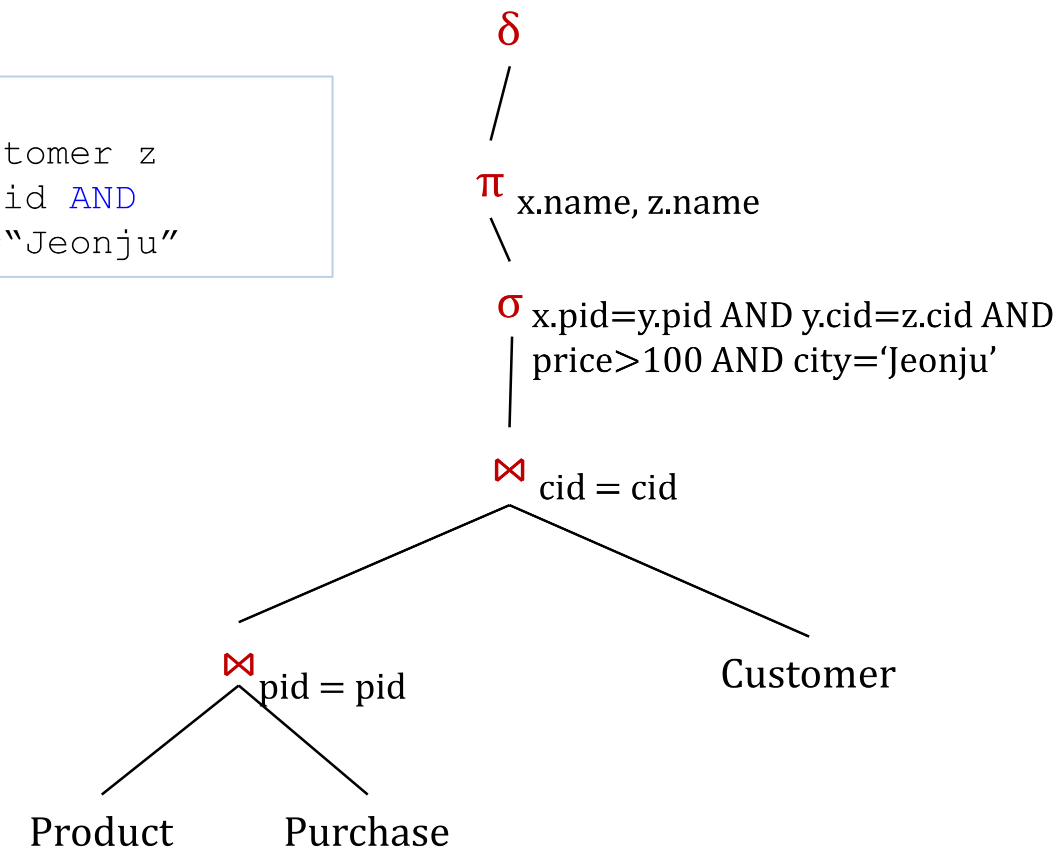
c.cid | c.na | c.cid,



```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

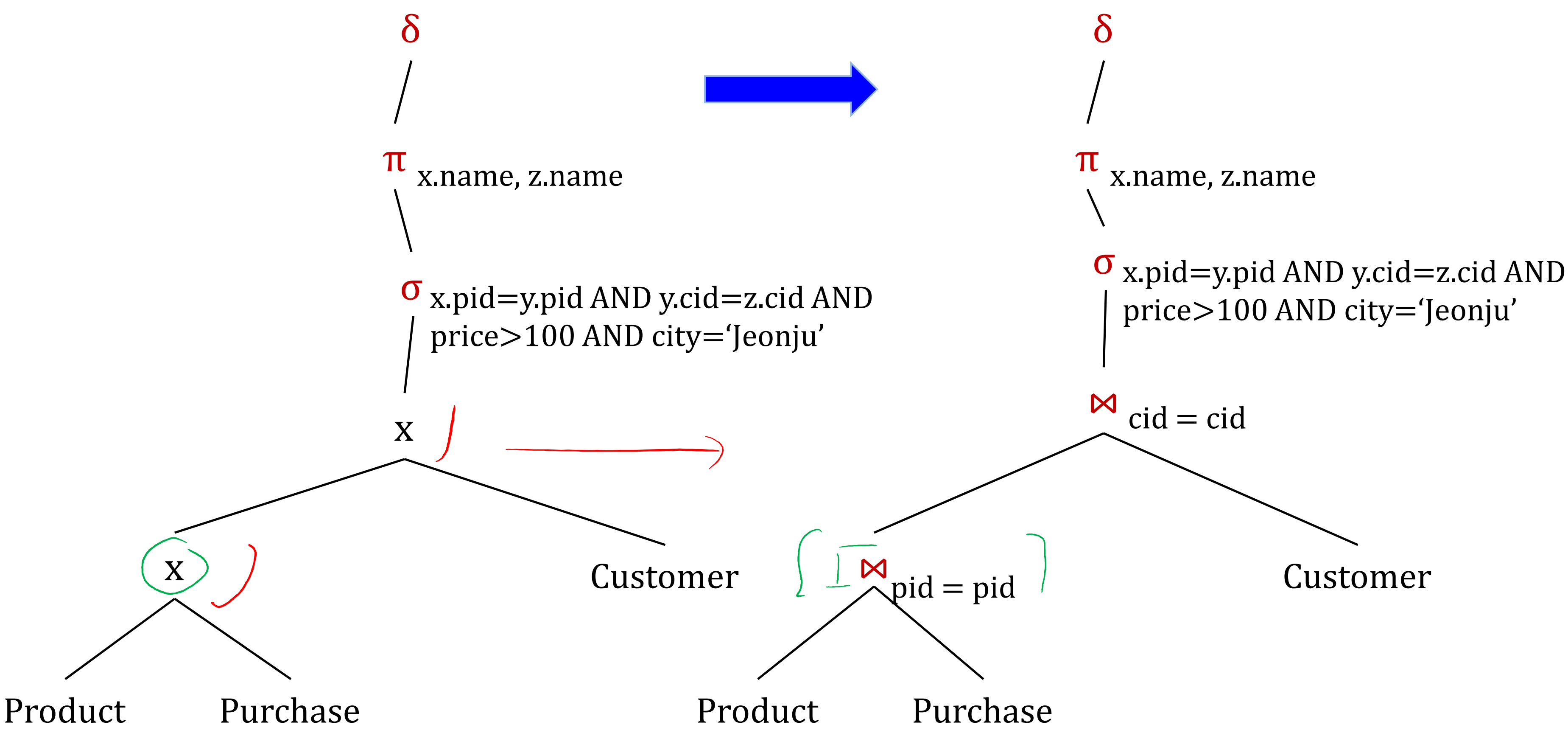
From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```



```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

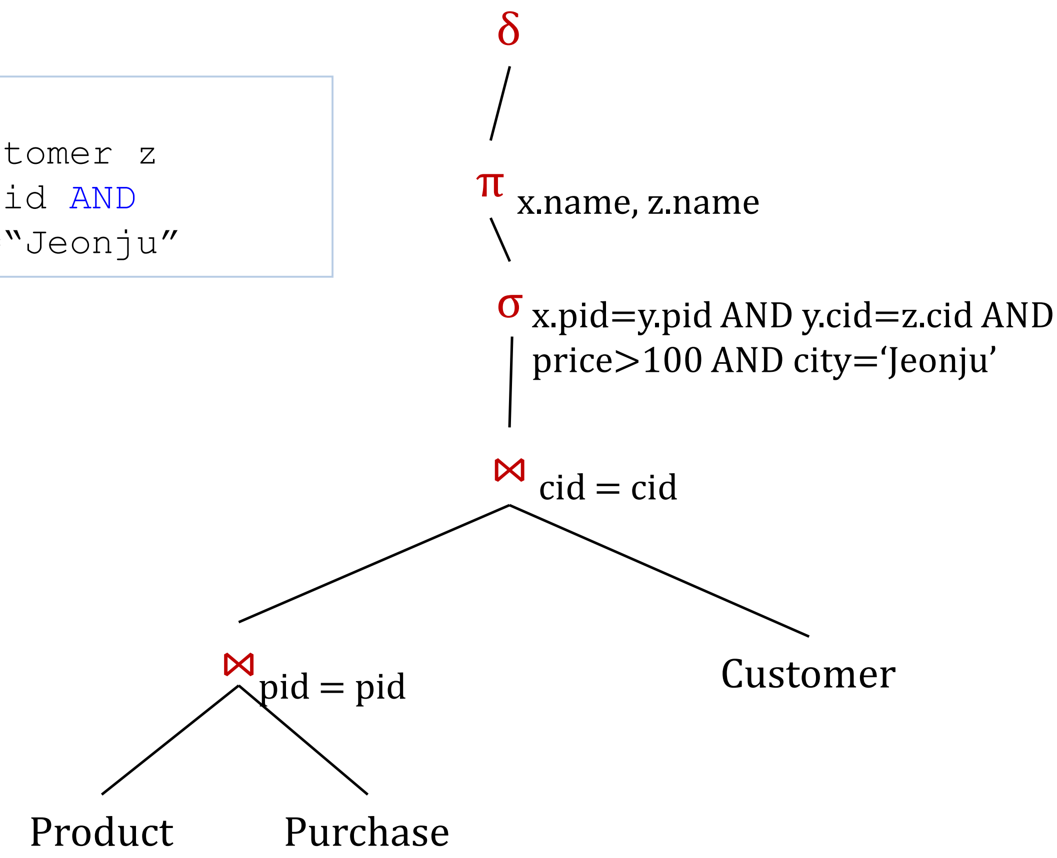
From SQL to RA



```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA

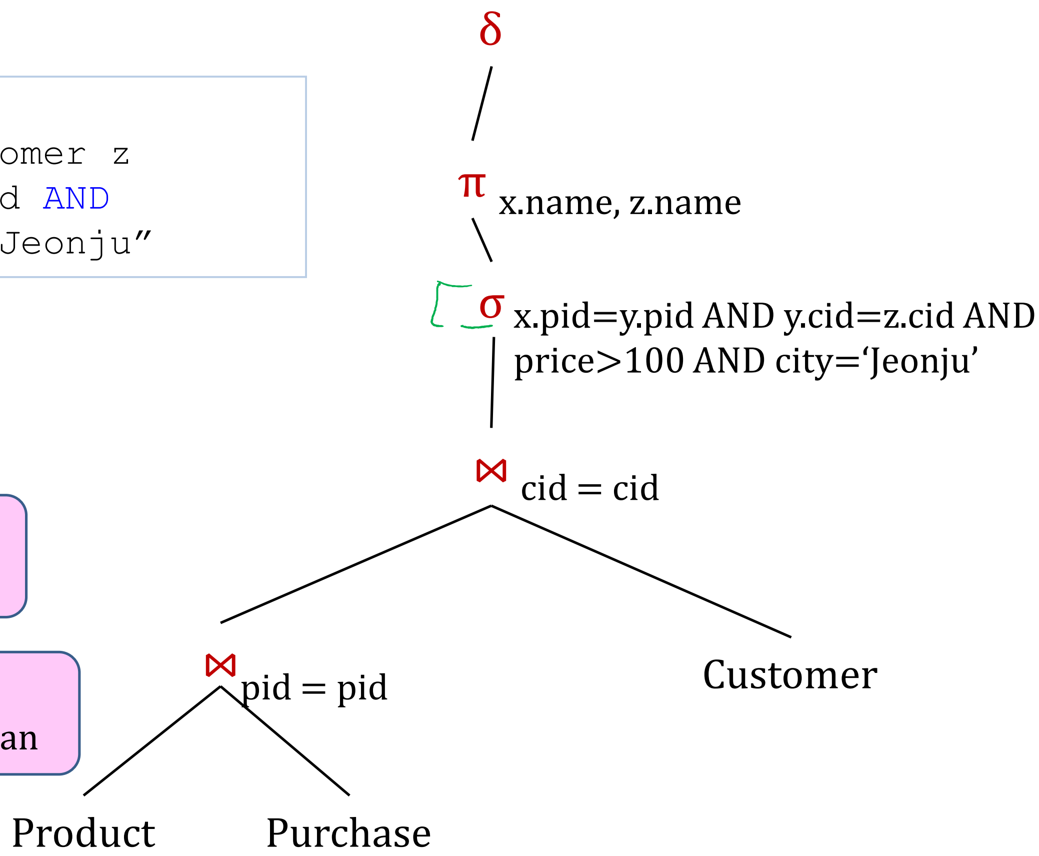
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```



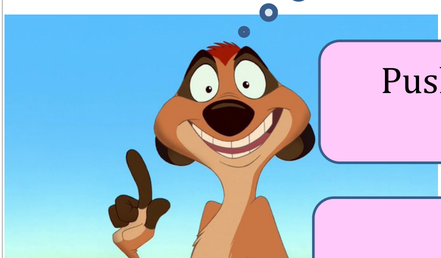
Product(<u>pid</u>, name, price
Purchase(<u>pid</u>, <u>cid</u>, store)
Customer(<u>cid</u>, name, city)

From SQL to RA

```sql
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```



We can do this way!



Push selections down the query plan

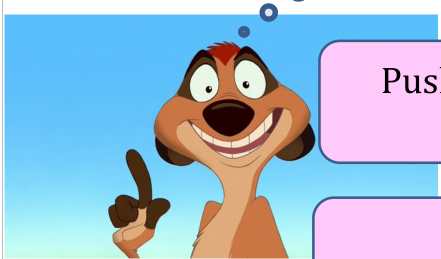
Query Optimization:
Find an equivalent optimal plan

```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA

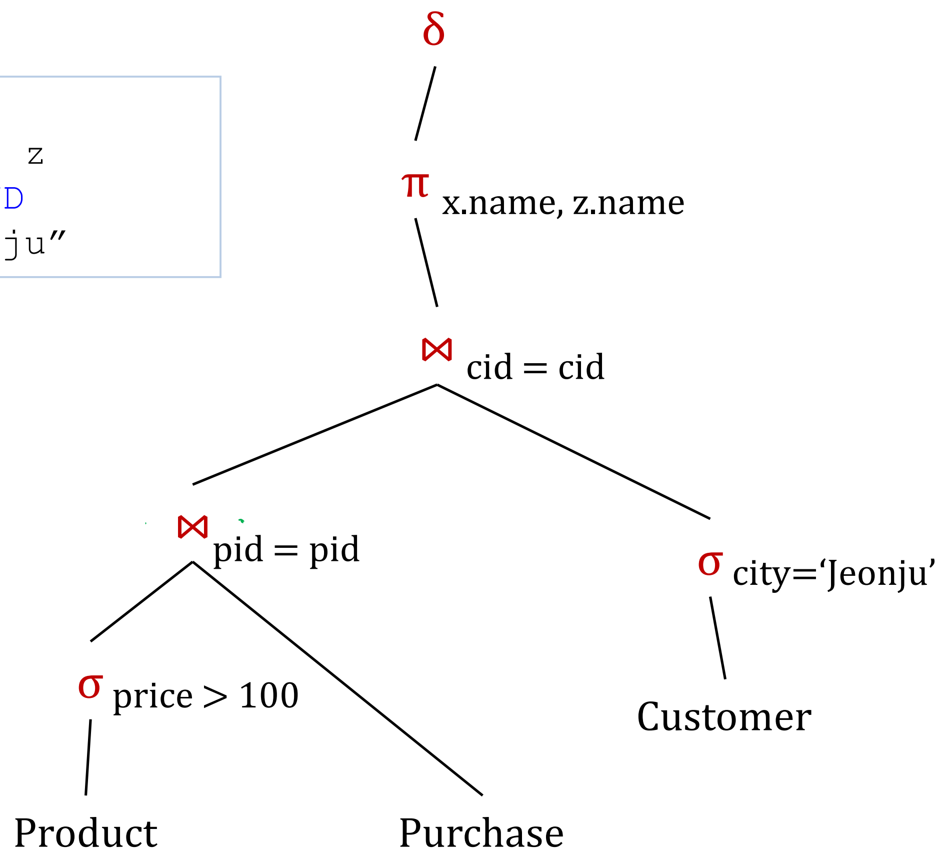
```sql
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid=y.pid AND y.cid=z.cid AND
      x.price > 100 AND z.city="Jeonju"
```

We can do this way!



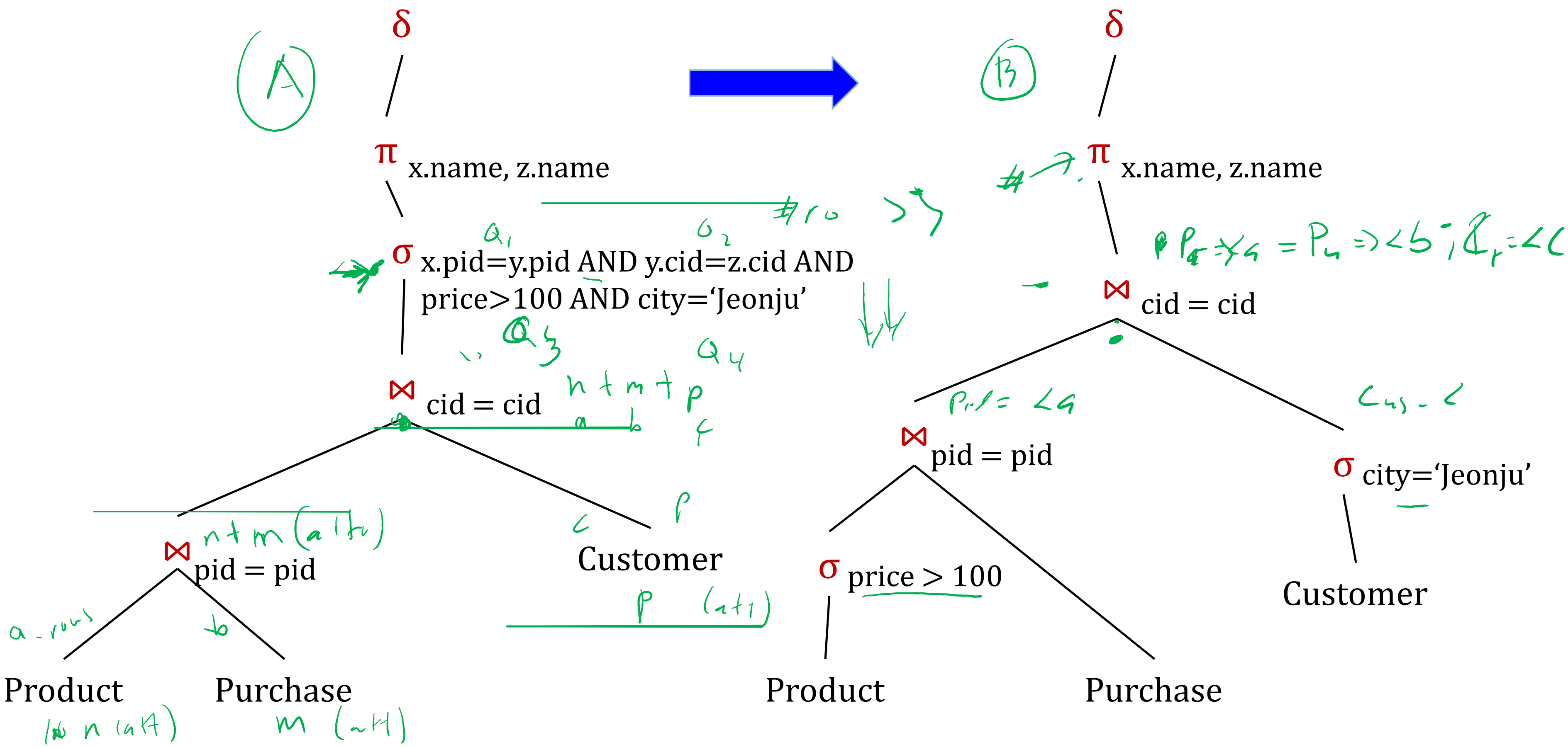
Push selections down the query plan

Query Optimization:
Find an equivalent optimal plan



```
Product(pid, name, price
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA



Extended RA: Operator on Bags

- Duplicate elimination (δ)
- Grouping & Aggregation (γ)
- Sorting (τ)

Logical Query Plan

```
SELECT city, count(*)
FROM sales
GROUP BY city
WHERE sum(price) > 100
```

Logical Query Plan

SQL

```
SELECT city, count(*)
FROM sales
GROUP BY city
WHERE sum(price) > 100
```

$\pi_{\text{city, c}}$ → T3(city, c)

$\sigma_{p > 100}$ → T2(city, p, c)

$\gamma_{\text{city, sum(price)} \rightarrow \text{p, count(*)} \rightarrow \text{c}}$ → T_1(city, p, c)

Sales(product, city, price)

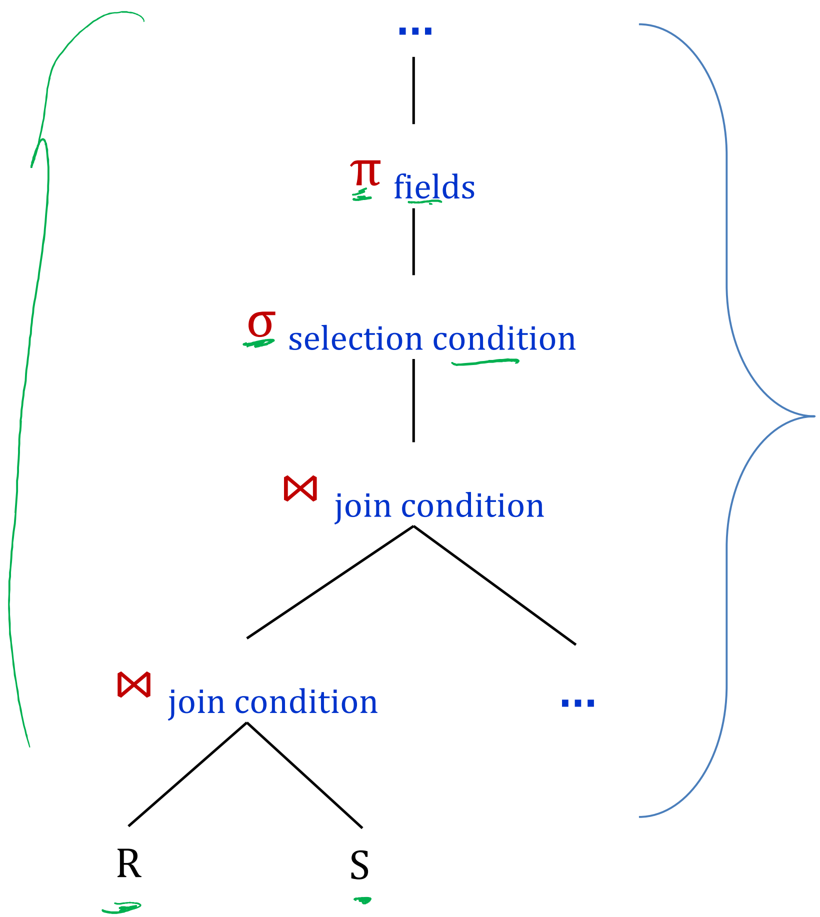
T_1, T_2, T_3 = temporary tables

Typical for Block (1/2)

```
SELECT fields
FROM R, S
WHERE condition
```

SELECT-PROJECT-JOIN Query

Typical for Block (1/2)



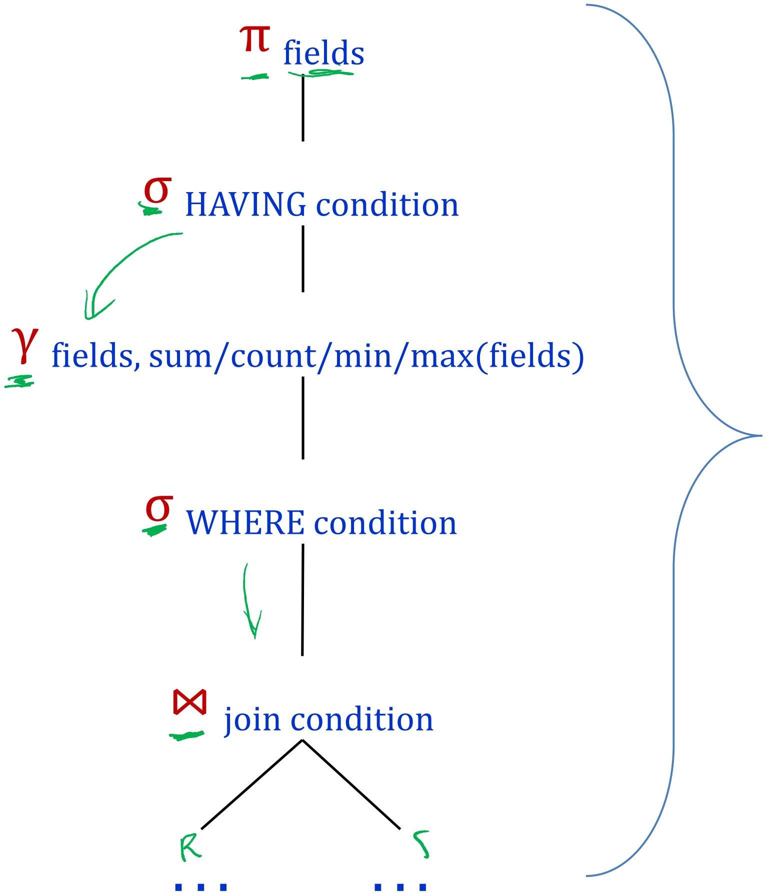

```
SELECT fields
FROM R, S
WHERE condition
```

SELECT-PROJECT-JOIN Query

Typical for Block (2/2)

```
SELECT fields
FROM R, S
WHERE condition
GROUP BY fields
HAVING condition
```

Typical for Block (2/2)




```
SELECT fields
FROM R, S
WHERE condition
GROUP BY fields
HAVING condition
```

How About Subqueries?

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   AND NOT EXISTS
   (SELECT *
    FROM Supply P
    WHERE P.sno = Q.sno
       AND P.price > 100;
```

Correlation!

How About Subqueries?

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   AND NOT EXISTS
   (SELECT *
    FROM Supply P
    WHERE P.sno = Q.sno
       AND P.price > 100;
```

De-Correlation

How About Subqueries?

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   AND NOT EXISTS
   (SELECT *
    FROM Supply P
    WHERE P.sno = Q.sno
       AND P.price > 100;
```

De-Correlation

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   AND Q.sno NOT IN
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

How About Subqueries?

Supplier(**sno**, sname, scity, sprov)
Part(**pno**, pname, psize, pcolor)
Supply(**sno**, **pno**, qty, price)

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   EXCEPT
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

Un-nesting

EXCEPT = set difference

How About Subqueries?

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   EXCEPT
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

n = 1, 0

Un-nesting

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   AND Q.sno NOT IN
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

EXCEPT = set difference

How About Subqueries?

Supplier(**sno**, sname, scity, sprov)
Part(**pno**, pname, psize, pcolor)
Supply(**sno**, **pno**, qty, price)

```
(SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   EXCEPT
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

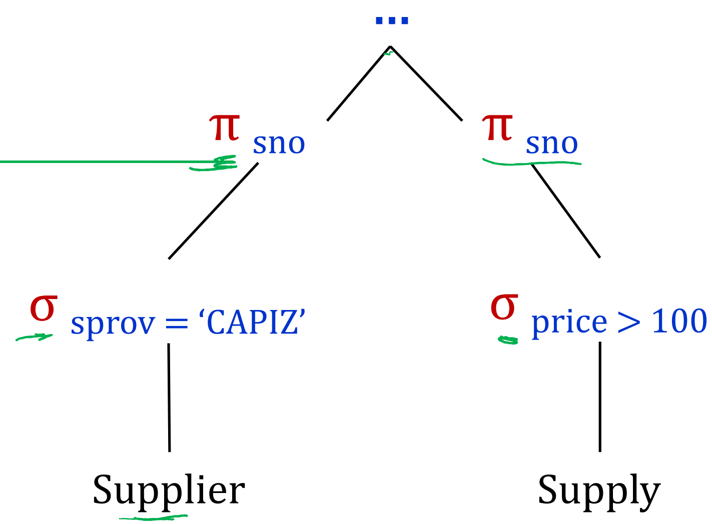
Finally...

How About Subqueries?

```
Supplier(sno, sname, scity, sprov)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)
```

```
(SELECT Q.sno
FROM Supplier Q
WHERE Q.sprov = 'CAPIZ'
   EXCEPT
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100;
```

Finally...
Logical Plan!



Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name (no “(   )“)
  - Ex: 202054321 Juan Dela Cruz
  -
  - Not changing your name to this format
    - you might be marked Absent

    * → absent?

INTRO TO DATA STRUCTURE

From Logical Plans to Physical Plans

Physical Operators

- Each of the logical operators may have one or more implementations = physical operations

- Discuss some basic physical operators, paying special attention to join

Main Memory Algorithms

- Logical operator

`Product(pid, name, price)`
`Purchase(pid, cid, store)`

Product(<u>pid</u>, name, price) $\bowtie_{pid=pid}$ Purchase(<u>pid</u>, <u>cid</u>, store)

Propose three physical operators for the join,
assuming the tables are in main memory:

1. Nested Loop Join — O(??)
2. Merge Join — O(??)
3. Hash Join — O(??)

Take note that **<u>pid</u>** is a key.

* time complexity : the computational complexity that describes the amount of time it takes to run an algorithm
** '*n*' – is the input size

Main Memory Algorithms

- Logical operator

```
Product(pid, name, price)
Purchase(pid, cid, store)
```

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge Join O(??)
3. Hash Join O(??)

Two nested loops

* time complexity : the computational complexity that describes the amount of time it takes to run an algorithm
** '*n*' – is the input size

Main Memory Algorithms

- Logical operator

```
Product(pid, name, price)
Purchase(pid, cid, store)
```

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join,
assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge Join $O(n \log n)$
3. Hash Join O(??)

Sort both: $O(n \log n)$
Merge: $O(n)$

* time complexity : the computational complexity that describes the amount of time it takes to run an algorithm
** '*n*' – is the input size

Main Memory Algorithms

- Logical operator

`Product(pid, name, price)`
`Purchase(pid, cid, store)`

Product(**pid**, name, price) $\bowtie_{pid=pid}$ Purchase(**pid**, **cid**, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join — $O(n^2)$
2. Merge Join — $O(n \log n)$
3. Hash Join — $O(n) \ldots O(n^2)$

Add n to hash: O(n)?
Lookup n in hash: O(n)

* time complexity : the computational complexity that describes the amount of time it takes to run an algorithm
** '*n*' – is the input size

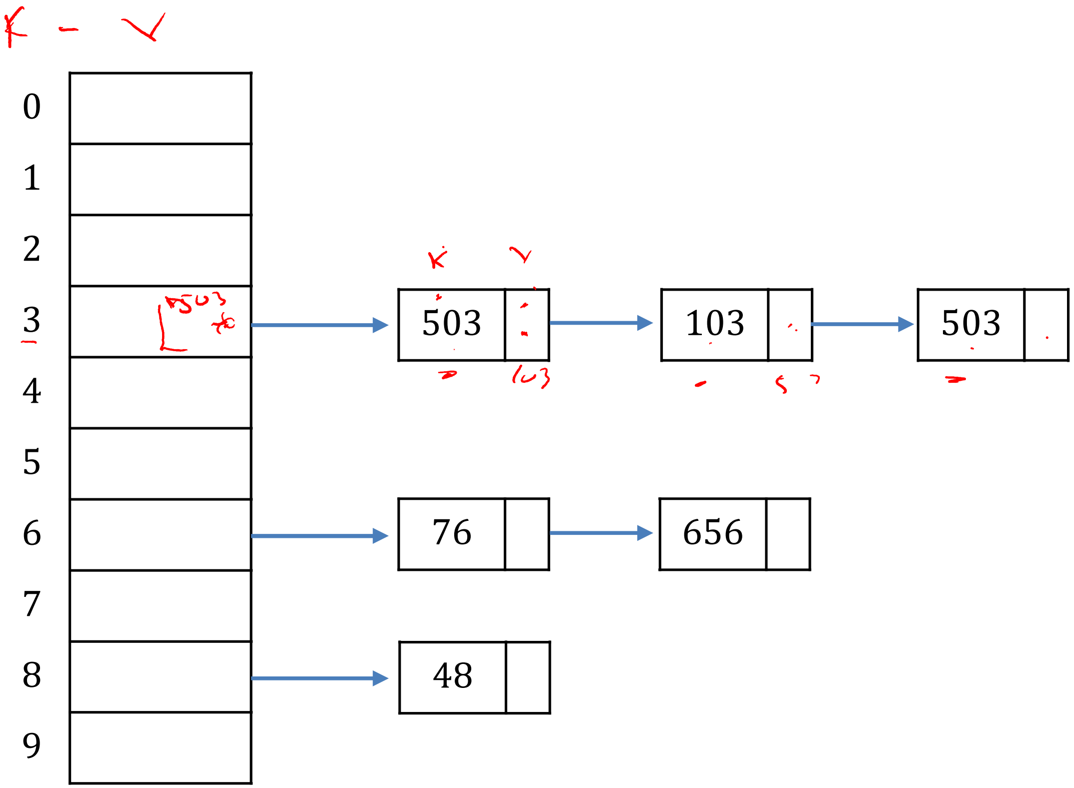
Brief Review of Hash Tables

A naive hash function:

$$h(x) = x \bmod 10$$

Operations:

find(103) = ??
insert(488) = ??



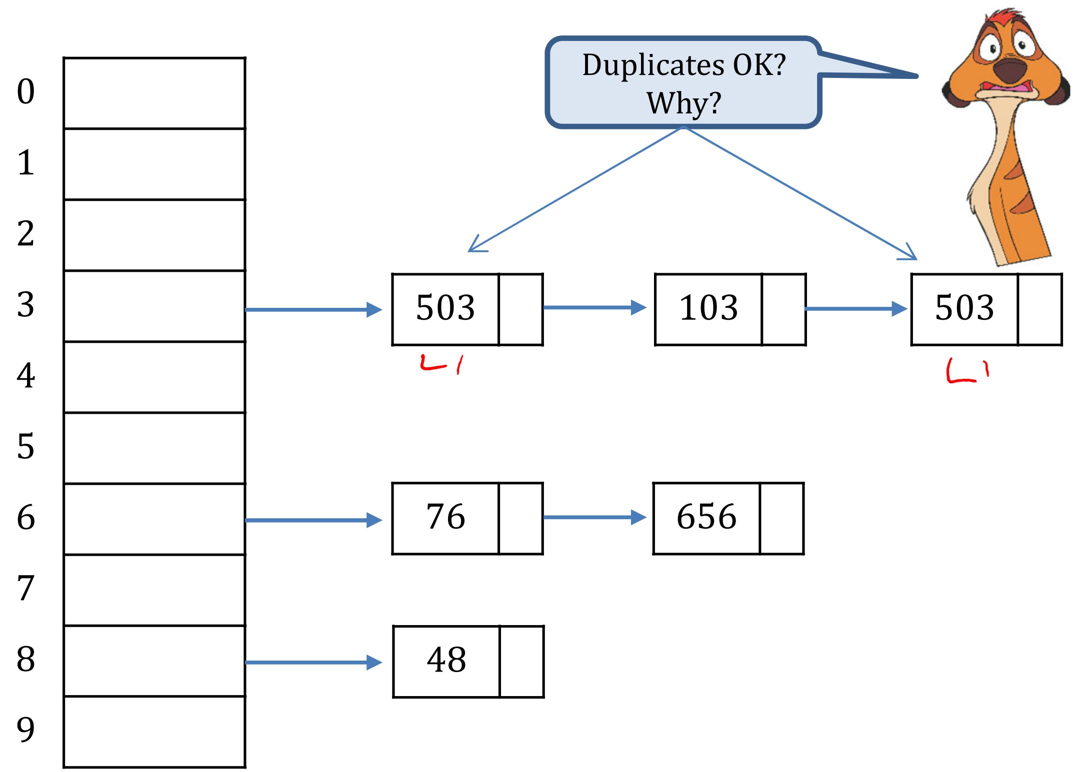
Brief Review of Hash Tables

A naive hash function:

$h(x) = x \bmod 10$

Operations:

find(103) = ??
insert(488) = ??



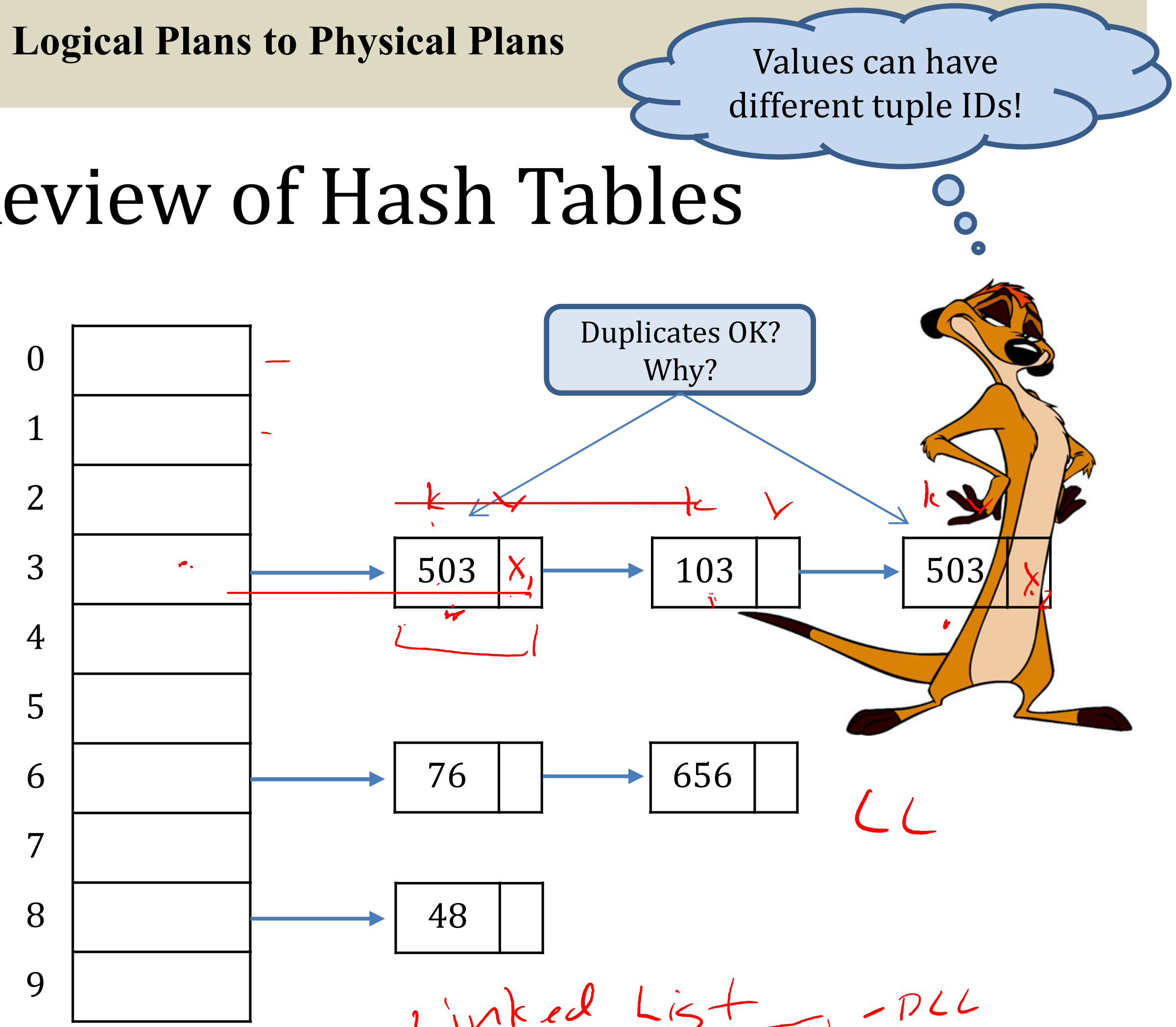
Brief Review of Hash Tables

A naive hash function:

$h(x) = x \bmod 10$

Operations:

find(103) = ??
insert(488) = ??

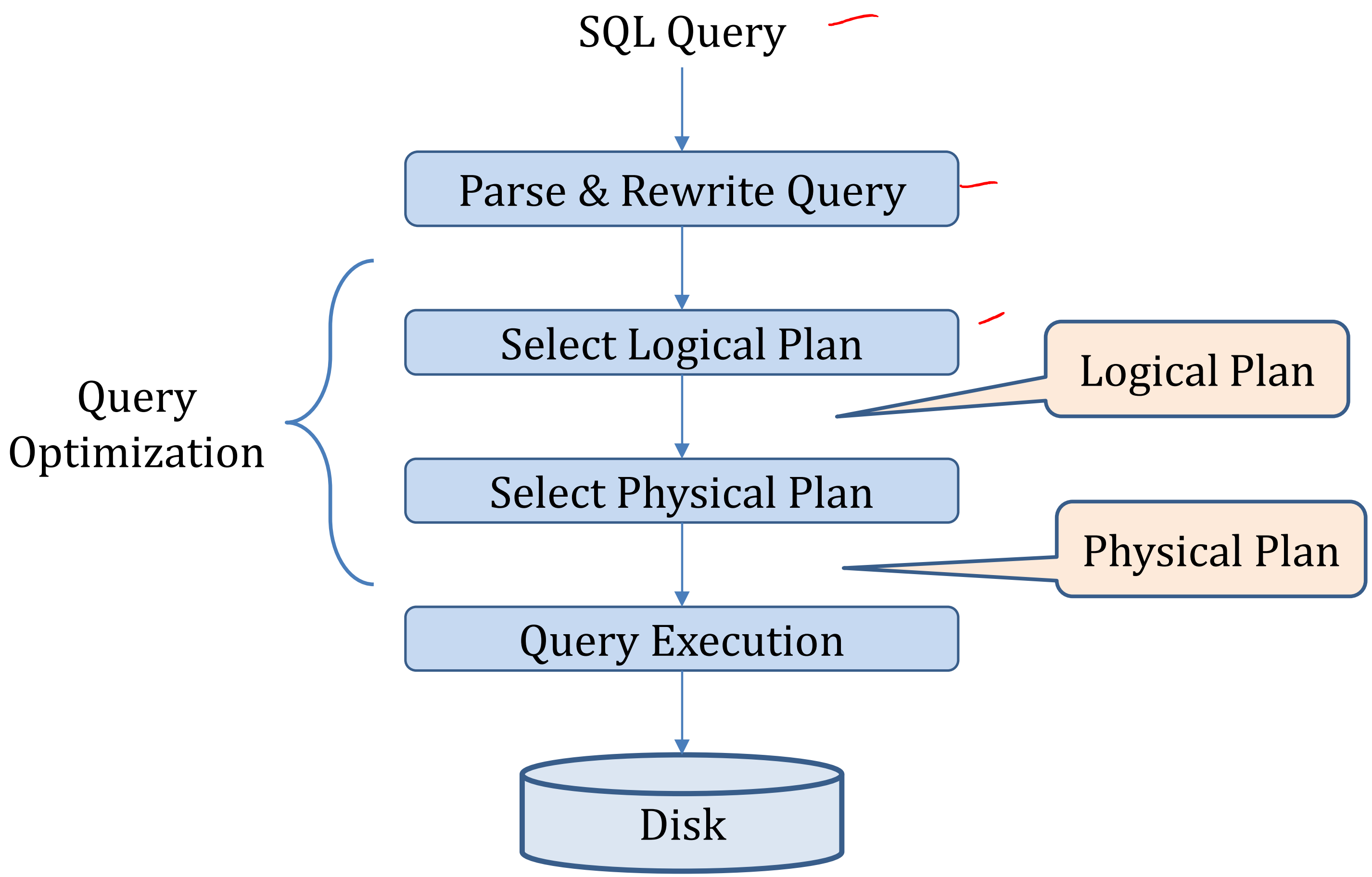


Linked List - DLL

HEAD DATA TAIL HEAD DATA TAIL

LL

Query Evaluation Steps Review



Relational Algebra

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

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

"∧" = AND

Give a relational algebra expression for this query.

$$\pi_{xxxx}(\sigma_{xxxx='xxxx' \wedge xxxx='xxxx' \wedge xxxx=xxxx}(\text{Supplier} \bowtie_{xxxx=xxxx} \text{Supply}))$$

sname y.pno = 2 x.scity = 'Jeonju' x.prov = 'CAPIZ' x.sid = y.sid

Relational Algebra

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

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

Give a relational algebra expression for this query.

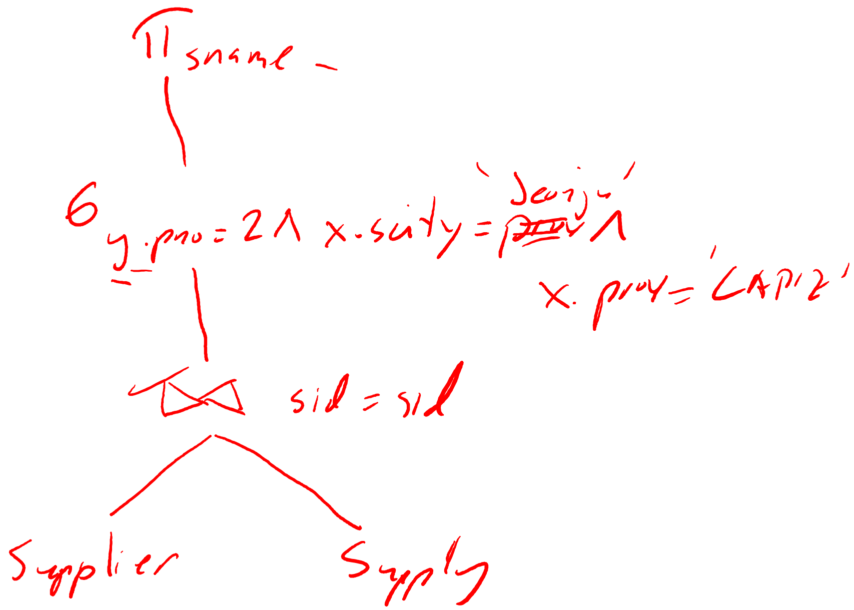
AND AND

$$\pi_{sname}(\sigma_{scity=\text{'Jeonju'} \wedge sprov=\text{'CAPIZ'} \wedge pno = 2}(\text{Supplier} \bowtie_{sid=sid} \text{Supply}))$$

Relational Algebra

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

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```



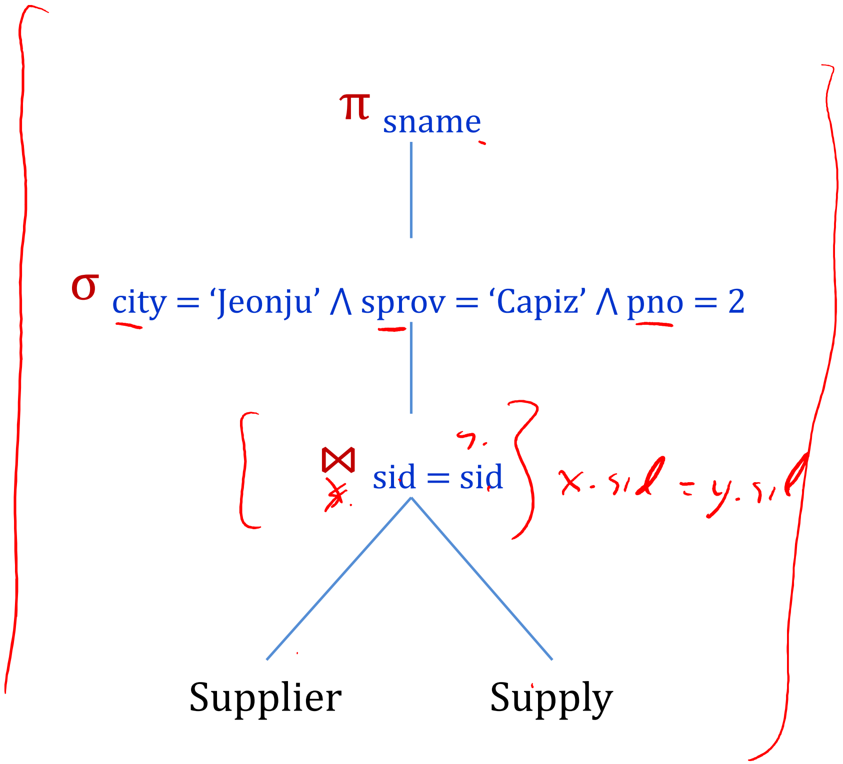
Relational algebra expression is also called the "logical query plan"

Relational Algebra

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

```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

Relational algebra expression is also called the "logical query plan"



1) Hashed Join ~~Loop~~ $O(n^2)$
2) ~~Loop~~ PP.. $O(n \log n)$
3) Hash Loop.. $O(n)$

Physical Query Plan 1

$n^2 >> n$

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

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{city = 'Jeonju'} \wedge \text{sprov = 'Capiz'} \wedge \text{pno = 2}}$

(Nested loop) → $\bowtie_{\text{sid = sid}}$

PP,

Supplier (File scan) Supply (File scan)

A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

Physical Query Plan 2

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

(On the fly) π sname

(On the fly) σ city = 'Jeonju' $\wedge$ sprov = 'Capiz' $\wedge$ pno = 2

PP2

(Hash Join) $\bowtie$ sid = sid

$O(n) \ll\ll O(n^2)$

Supplier (File scan) | Supply (File scan)

Same logical query plan
Different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

SQL

Nested Loop $O(n^2)$
Sort-merge $O(n \log(n))$
Hash Join $O(n)$

Physical Query Plan 3

$n^2 = ?$
$n \log n = ?$
$n = 25{,}000$

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

(On the fly) π sname

(Sort-merge join) $\bowtie$ sid = sid

LP_L ⟷ SQL

~~(Scan & write to T_1)~~
(Scan & write to T_1) σ city = 'Jeonju' $\wedge$ sprov = 'Capiz'

(Scan & write to T_2) σ pno = 2

T_1 T_2

Supplier (File scan)

Supply (File scan)

Same logical query plan
Different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  AND y.pno = 2
  AND x.scity = 'Jeonju'
  AND x.prov = 'CAPIZ';
```

Query Optimization Problem

- For each SQL query … many logical plans

Ex: 1 SQL query ⇒ 2 Logical Plans → LP_A ⇒ 2 PP
LP_A ⇒ 1 PP

- For each logical plan … many physical plans

- How to find a fast physical plan?
  - Will discuss in a few lectures

Thank you.