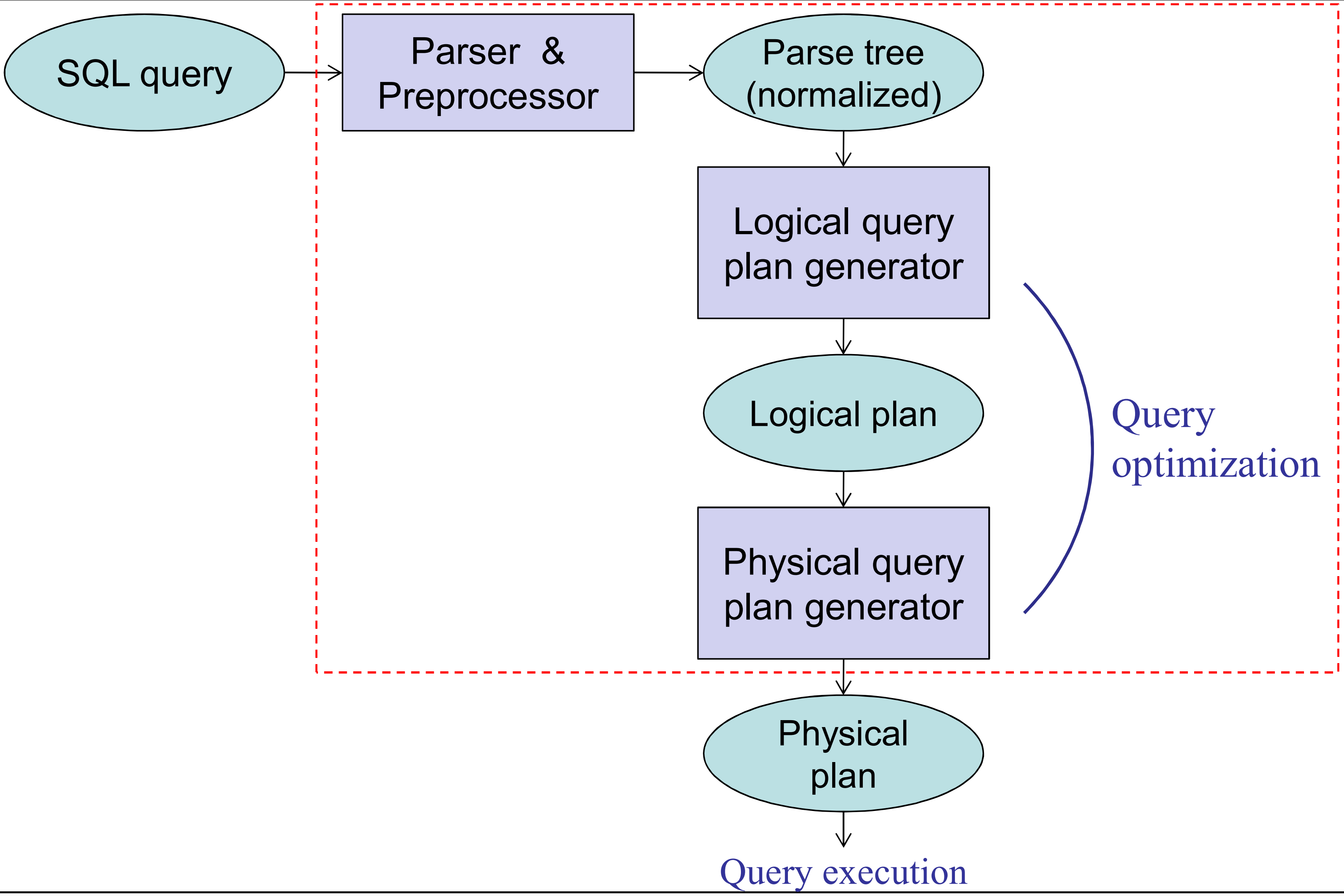
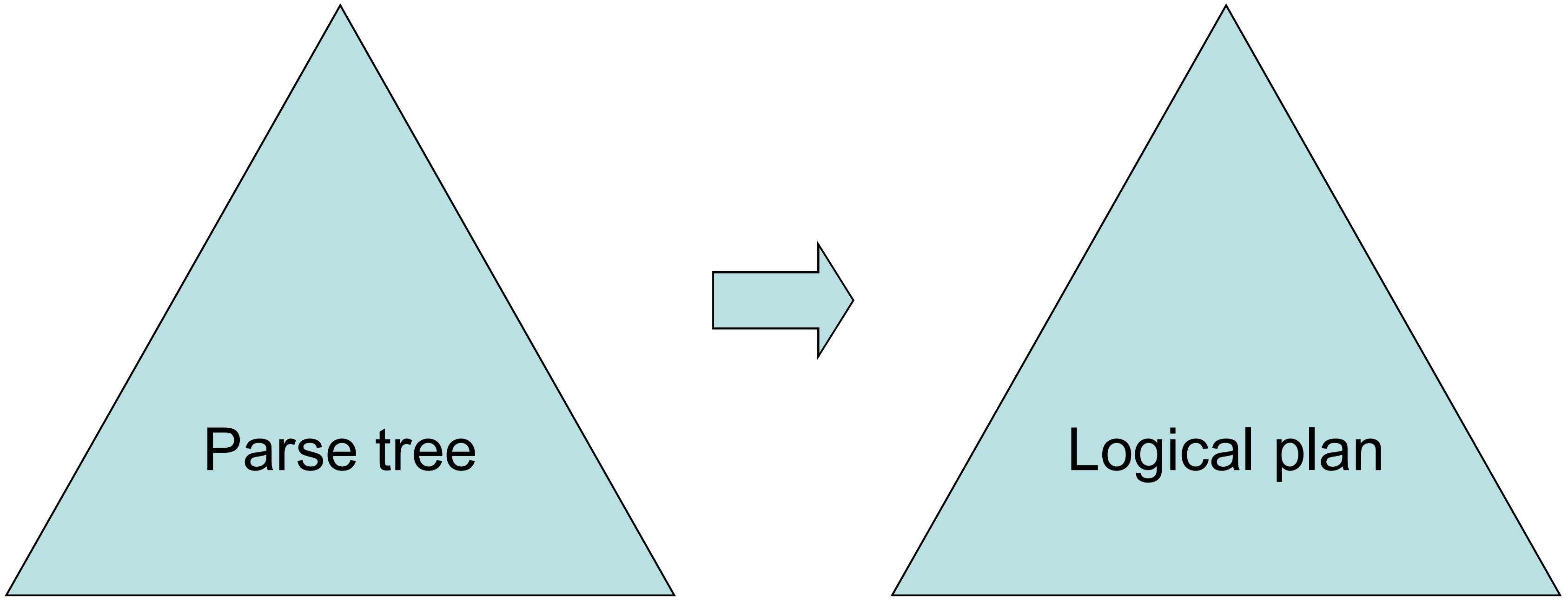
Database system implementation

3. Query conversion

Outline of query compilation



Query conversion



Logical plan

- ***Logical plan of query execution*** is a tree which nodes are the relational operations, and leaves are the relations.
- The logical plan unambiguously corresponds to a relational algebra expression.

Query conversion

- Conversion of simple query
- Conversion of complex query

Conversion of simple query

- ***Simple query*** is a <SFW> construct with a <Condition> that has no subqueries.

```
SELECT A, D
FROM R,S
WHERE E>5 AND R.C=S.C;
```

Conversion of simple query to relational algebra expression

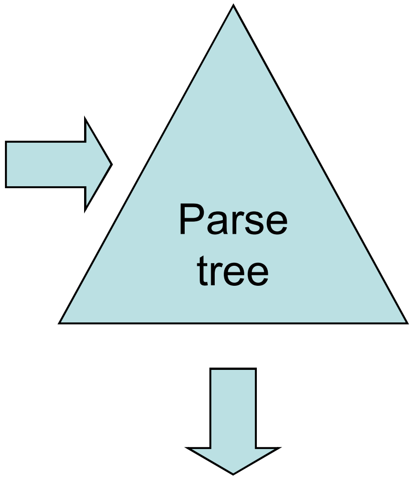
1. The cartesian product of all the relations mentioned in the <FromList>, which is the argument of:
2. A selection σ_C, where C is the <Condition> expression, which is the argument of:
3. A projection π_L, where L is the list of attributes in the <SelList>.

Conversion of simple query

Database schema:
R(A,B,C); S(C,D,E)

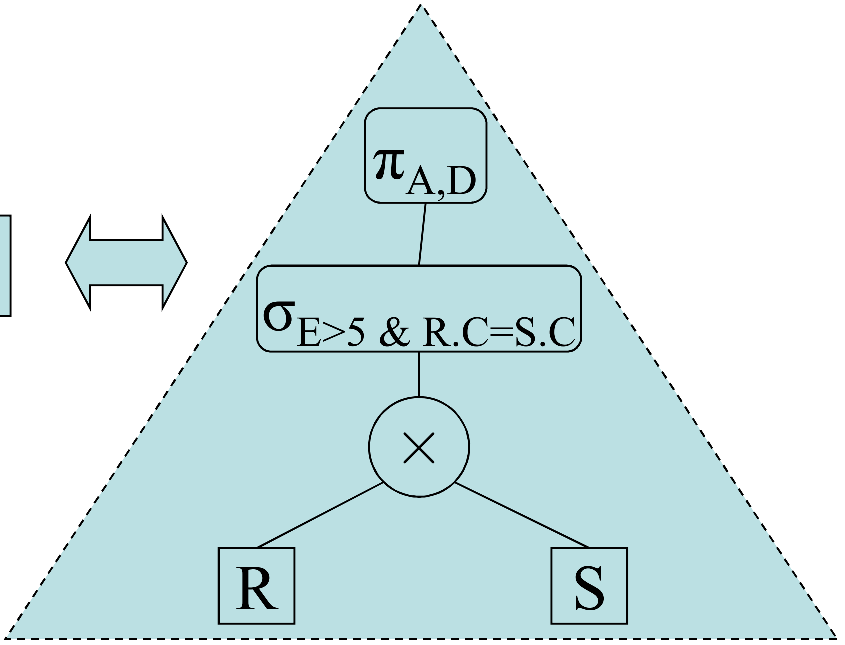
```
SELECT A, D
FROM R,S
WHERE E>5
AND R.C=S.C;
```

SQL query



$\pi_{A,D}(\sigma_{E>5\ \&\ R.C=S.C}(R \times S))$

Relational algebra expression



Logical plan

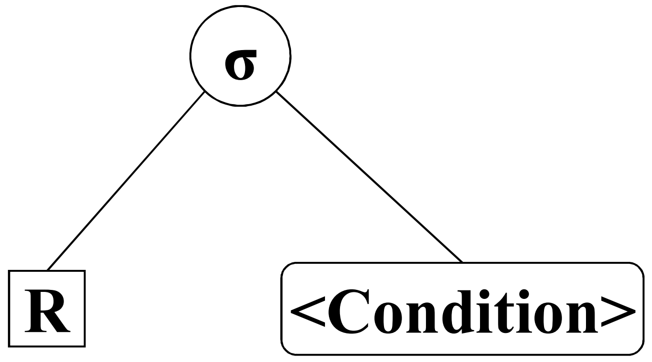
Conversion of complex query

Complex query is a <SFW> construct with a <Condition> that has a subquery.

```
SELECT *
FROM R
WHERE C IN
   ( SELECT C
     FROM S
     WHERE D > 5 );
```

Two-argument selection

- First argument - relation,
- Second argument - condition



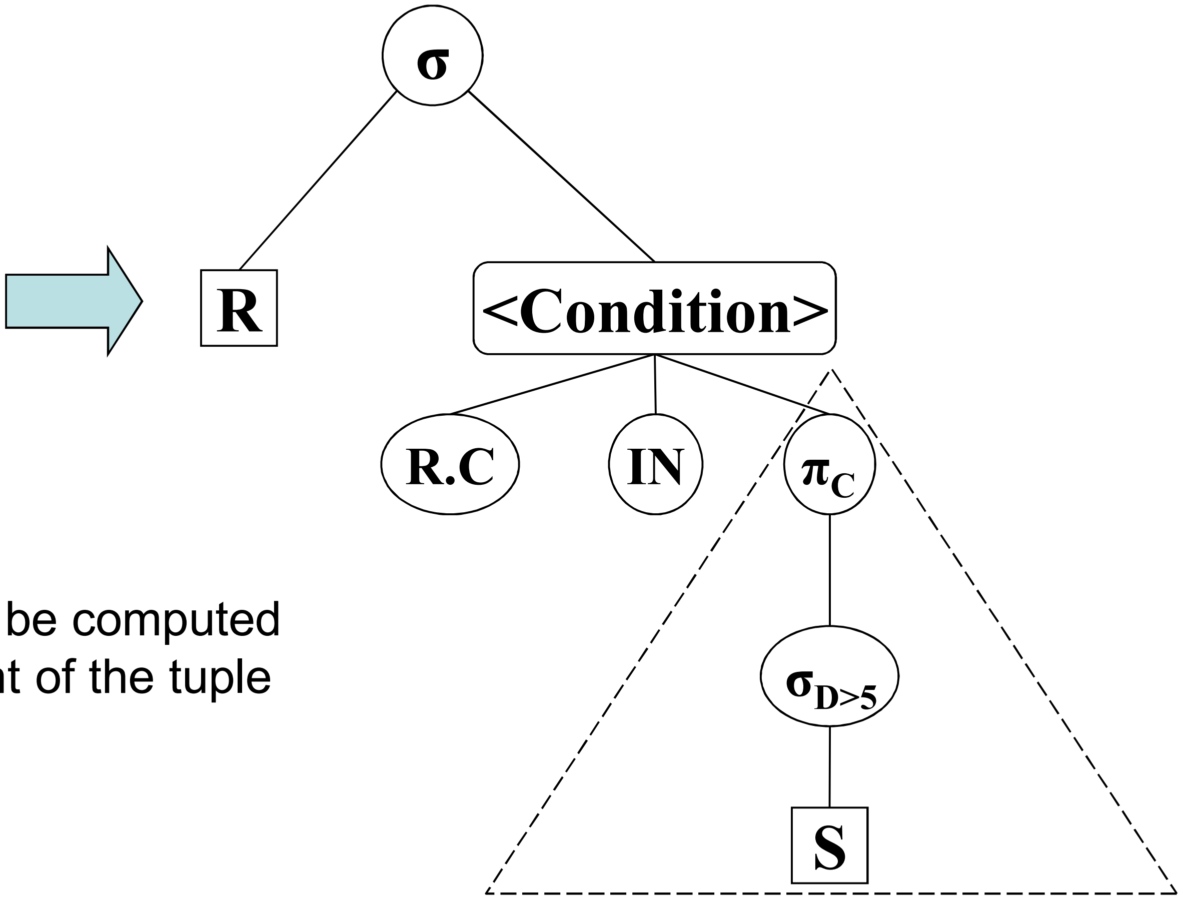
Conversion of complex query

1. To construct a logical plan using the two-argument selection.
2. Replace the two-argument selection by a one-argument selection and other operations of relational algebra.

Subquery in the condition is uncorrelated

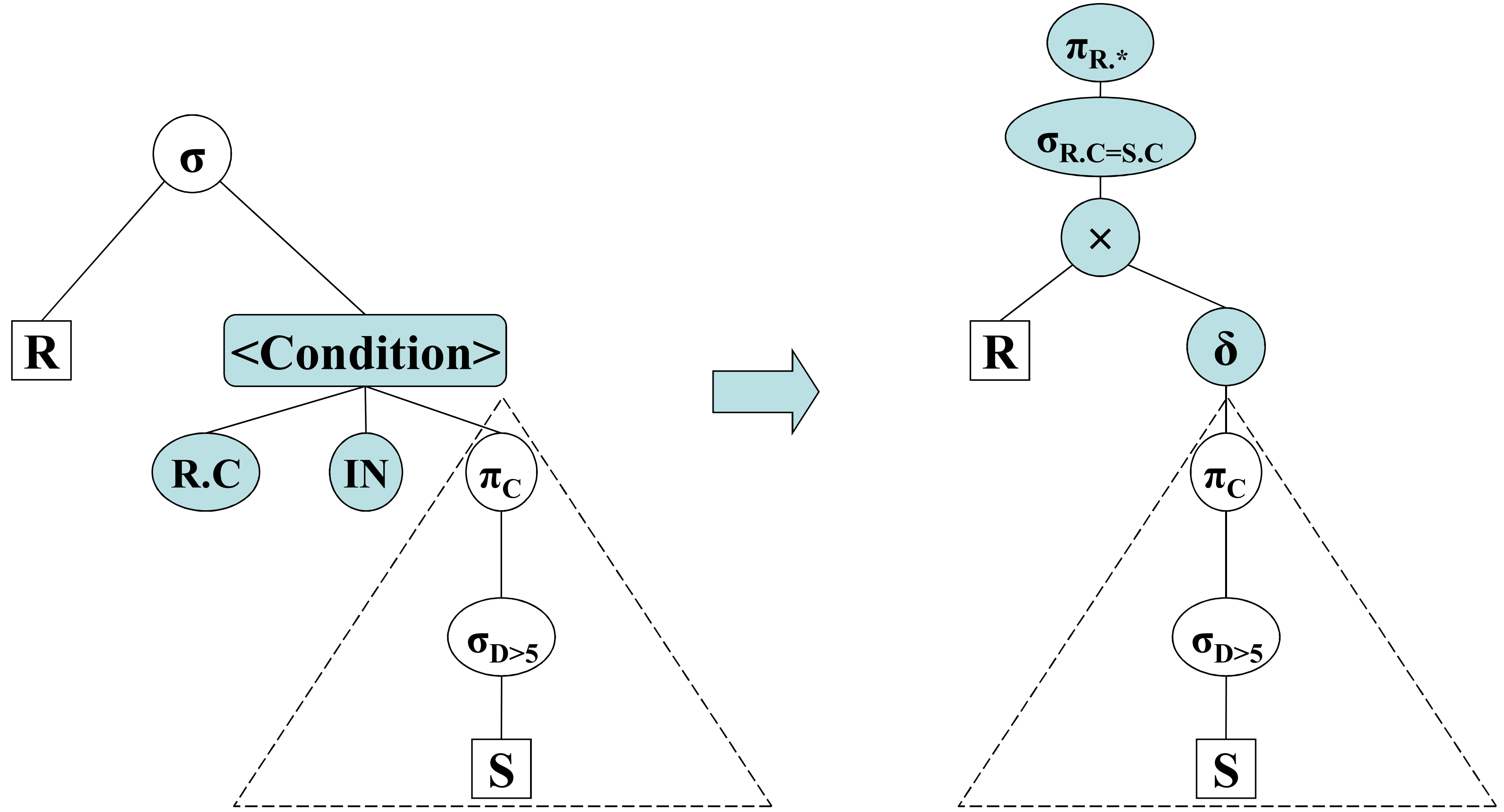
Database schema:
R(A,B,C); S(C,D,E)

```
SELECT *
FROM R
WHERE R.C IN
  (SELECT S.C
  FROM S
  WHERE D > 5);
```



The subquery's relation can be computed once and for all, independent of the tuple of outer query being tested.

Replacement of two-argument selection by one-argument selection



Correlated subquery

To calculate the names of suppliers having deposit which is less then the average price of all theirs supplies.

AVG – calculates the average value.

```
SELECT   Name_S
FROM   S
WHERE   Deposit < (
  SELECT   AVG(Price*Amount)
  FROM  SP
  WHERE   S.ID_S = SP.ID_S
);
```

Algorithm for correlated subquery

1. To introduce the aliases S1 and S2 for S
2. To use two-argument selection
3. To introduce the additional attribute AP (average price) in grouping operation
4. To replace the two-argument selection by the one-argument selection

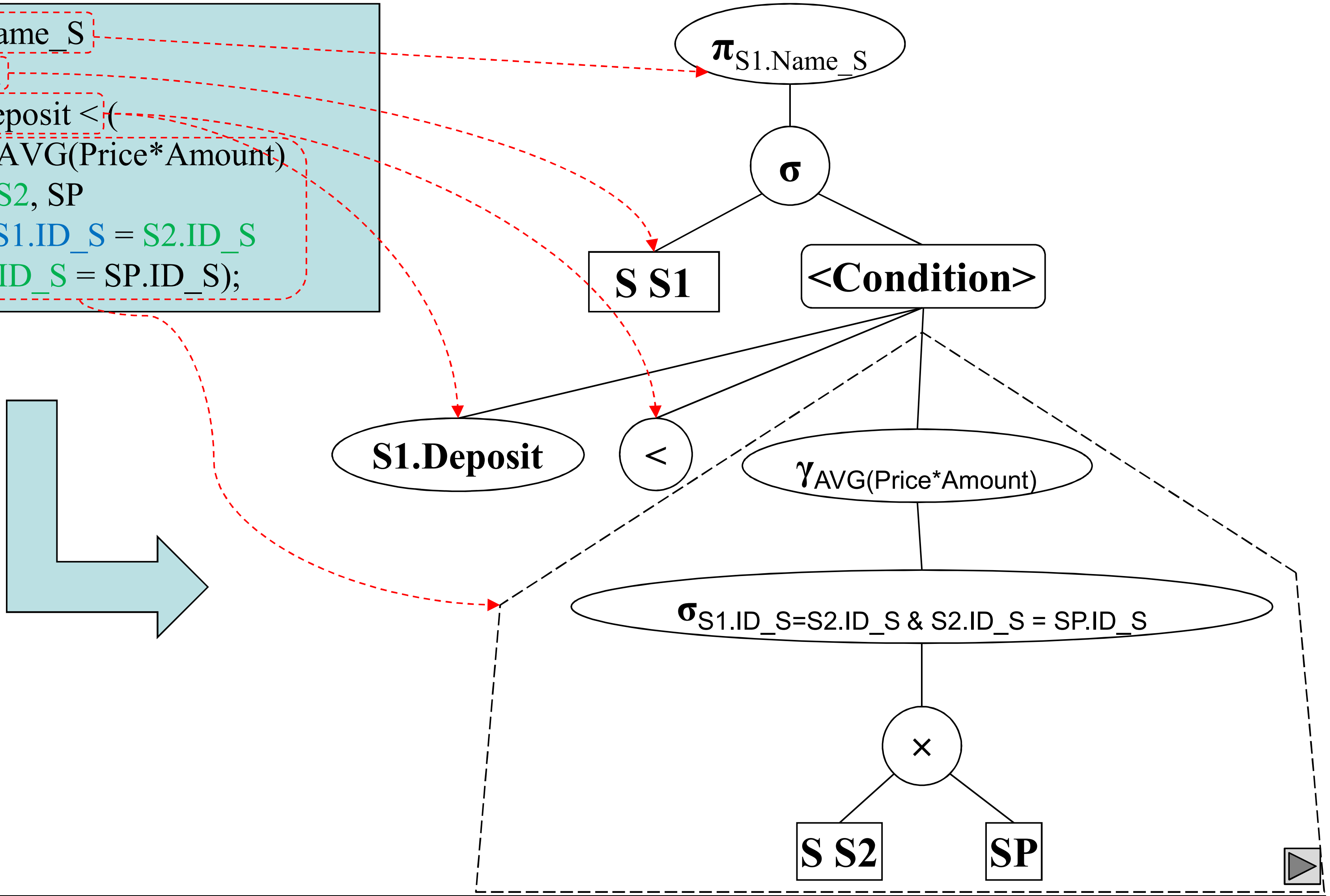
Introduction of aliases S1 and S2 for S

```
SELECT Name_S
FROM S
WHERE Deposit < (
  SELECT AVG(Price*Amount)
  FROM SP
  WHERE S.ID_S = SP.ID_S);
```

```
SELECT Name_S
FROM S S1
WHERE Deposit < (
  SELECT AVG(Price*Amount)
  FROM S S2, SP
  WHERE S2.ID_S = SP.ID_S
  AND S1.ID_S = S2.ID_S);
```

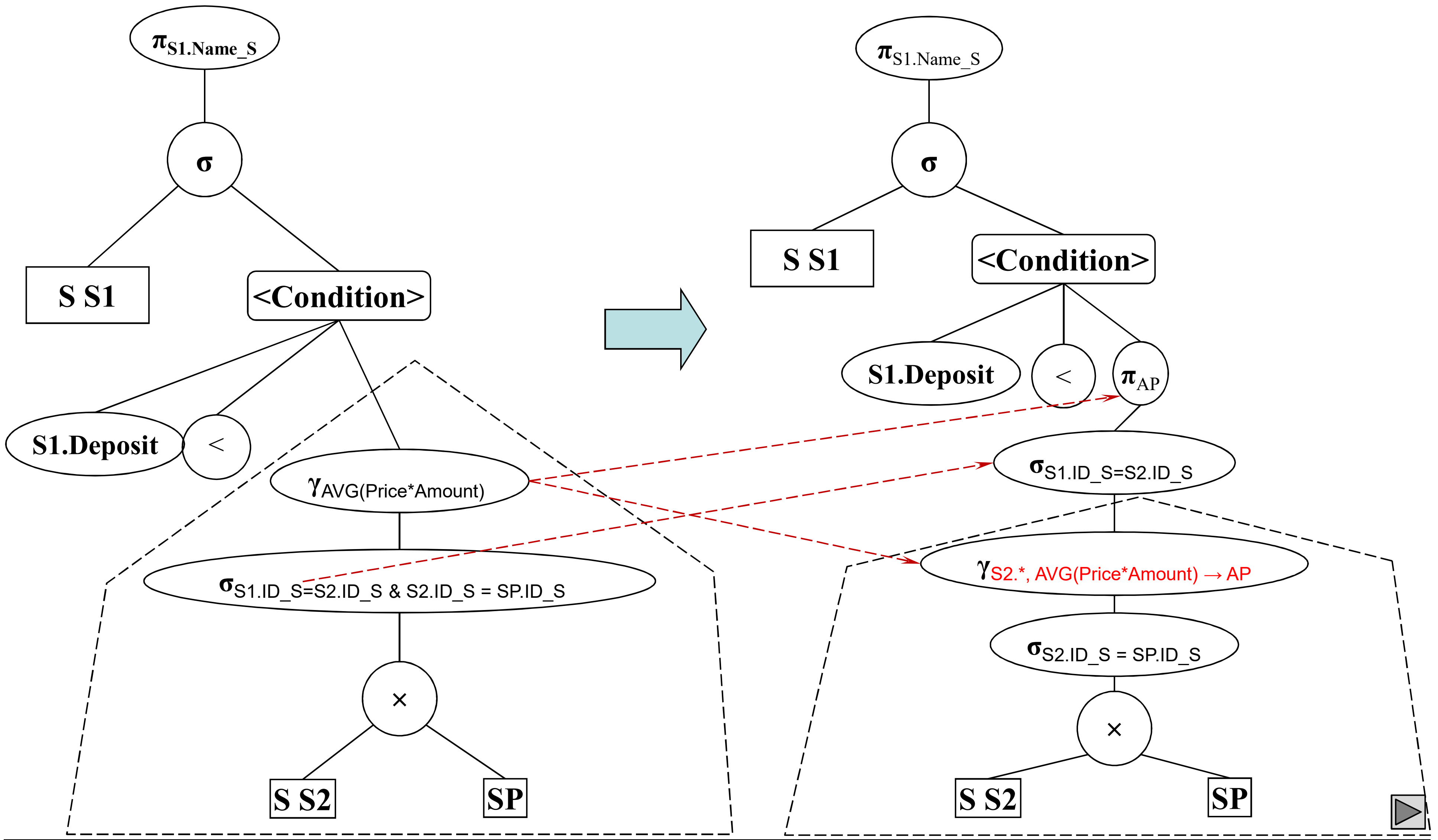
Use of two-argument selection

```
SELECT Name_S
FROM S S1
WHERE Deposit < (
  SELECT AVG(Price*Amount)
  FROM S S2, SP
  WHERE S1.ID_S = S2.ID_S
  AND  S2.ID_S = SP.ID_S);
```

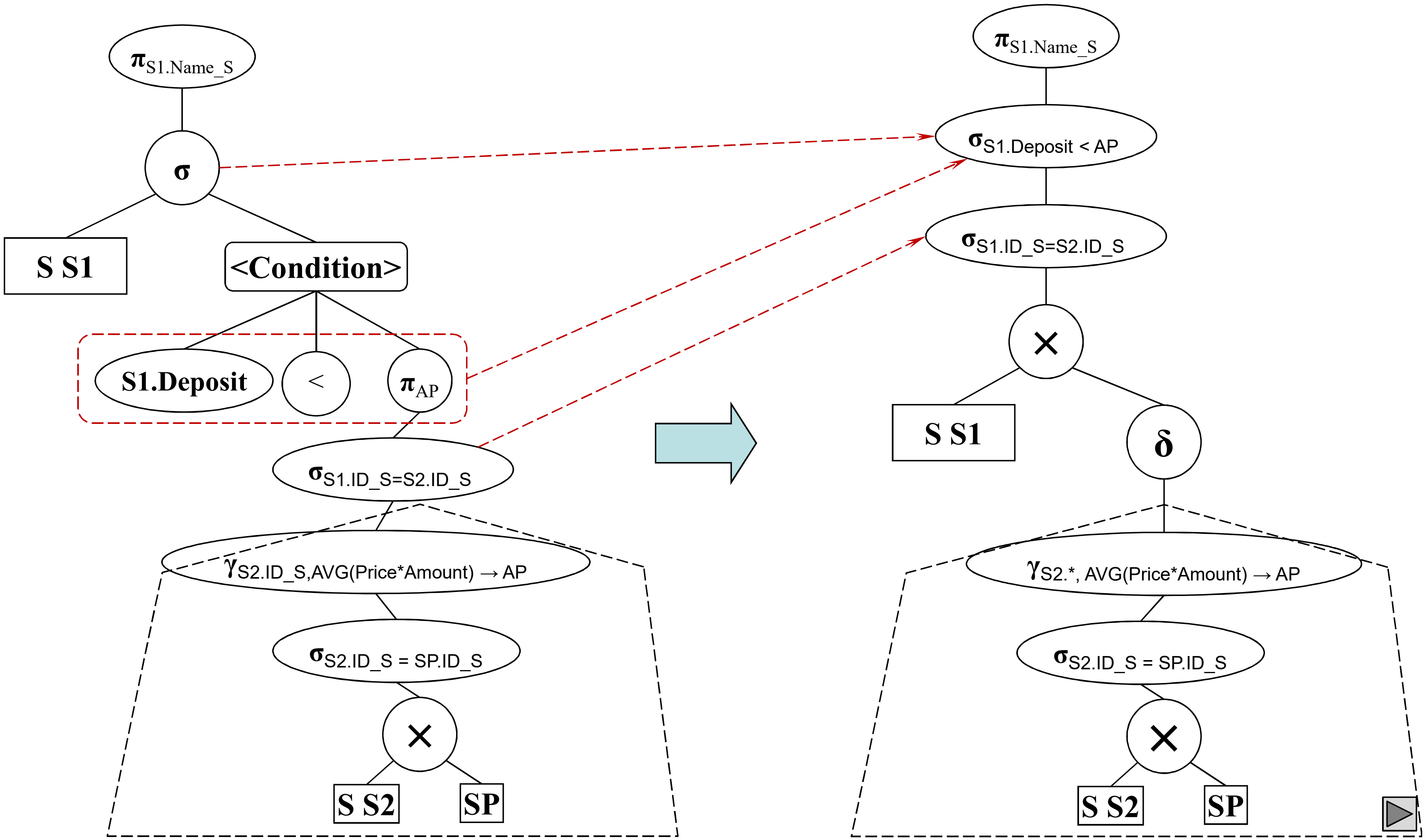


$\pi_{S1.Name\_S}$
σ
S S1
<Condition>
S1.Deposit
<
$\gamma_{AVG(Price*Amount)}$
$\sigma_{S1.ID\_S=S2.ID\_S \ \& \ S2.ID\_S = SP.ID\_S}$
×
S S2
SP

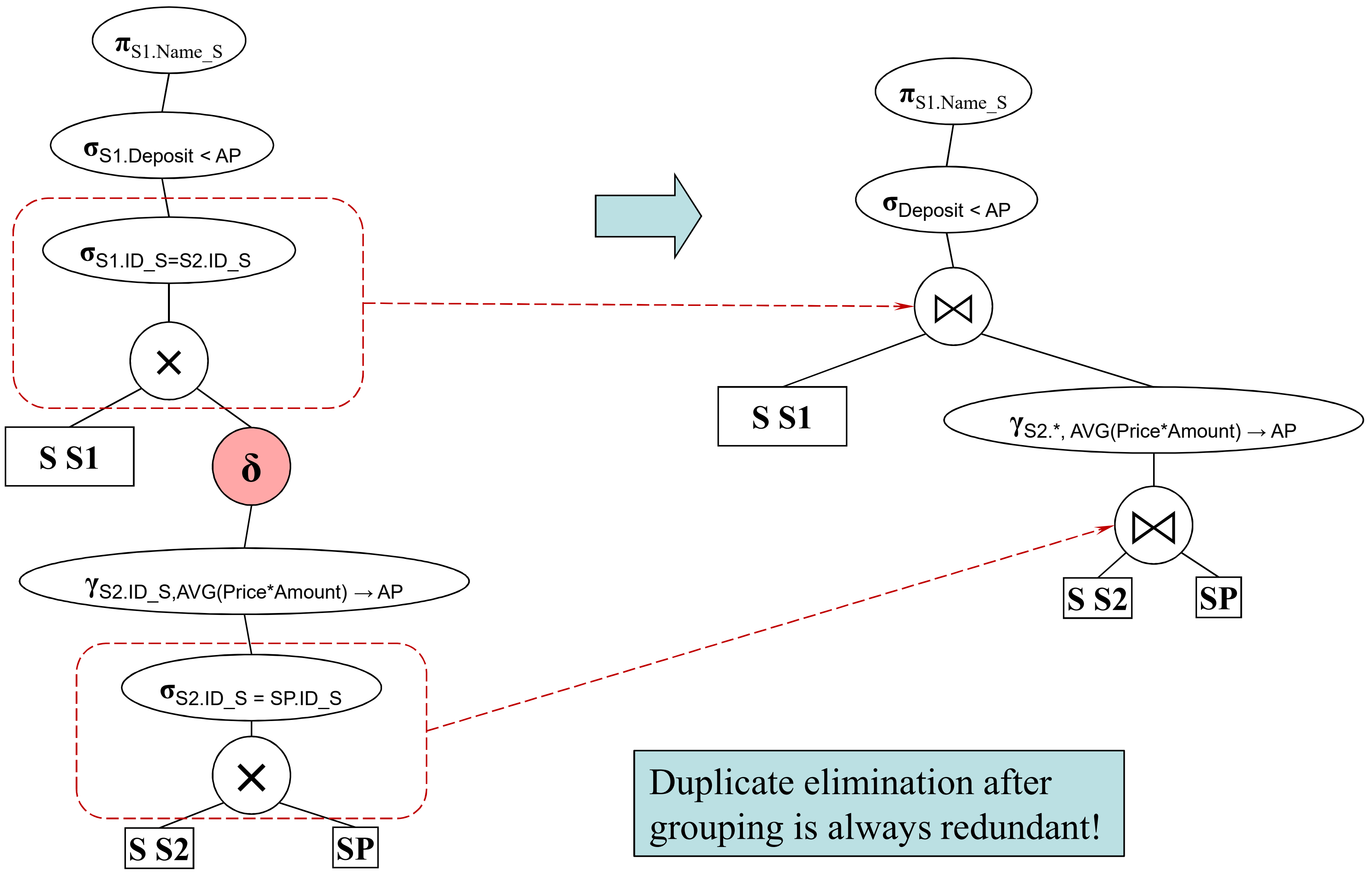
Introduction of S2 attributes and additional attribute AP (average price) in grouping operation


$\pi_{S1.Name\_S}$
σ
S S1
<Condition>
S1.Deposit
<
$\gamma_{AVG(Price*Amount)}$
$\sigma_{S1.ID\_S=S2.ID\_S \& S2.ID\_S = SP.ID\_S}$
×
S S2
SP
$\pi_{S1.Name\_S}$
σ
S S1
<Condition>
S1.Deposit
<
π_{AP}
$\sigma_{S1.ID\_S=S2.ID\_S}$
$\gamma_{S2.*, AVG(Price*Amount) \rightarrow AP}$
$\sigma_{S2.ID\_S = SP.ID\_S}$
×
S S2
SP

Replacement of two-argument selection by one-argument selection

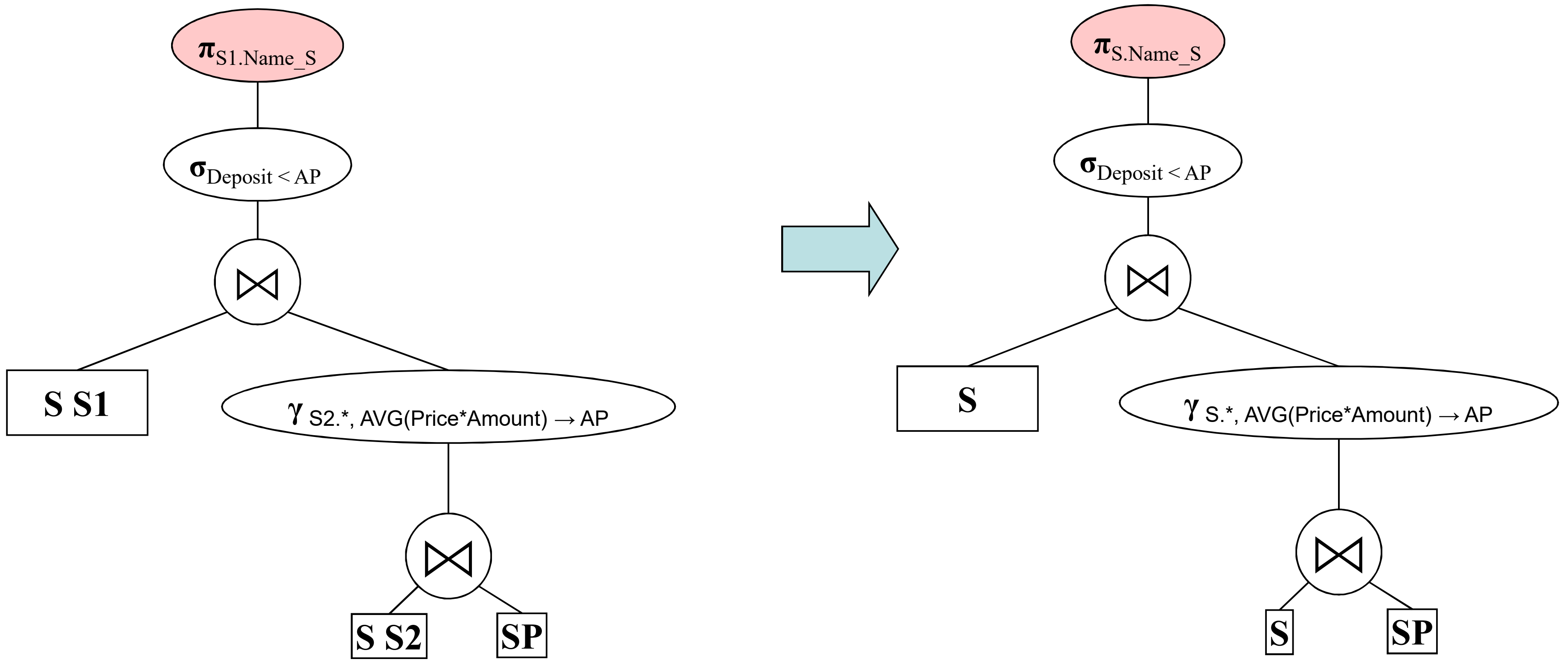

$\pi_{S1.Name\_S}$
σ
S S1
<Condition>
S1.Deposit
<
π_{AP}
$\sigma_{S1.ID\_S=S2.ID\_S}$
$\gamma_{S2.ID\_S,AVG(Price*Amount) \to AP}$
$\sigma_{S2.ID\_S = SP.ID\_S}$
×
S S2
SP
$\pi_{S1.Name\_S}$
$\sigma_{S1.Deposit < AP}$
$\sigma_{S1.ID\_S=S2.ID\_S}$
×
S S1
δ
$\gamma_{S2.*, AVG(Price*Amount) \to AP}$
$\sigma_{S2.ID\_S = SP.ID\_S}$
×
S S2
SP

Logical optimization

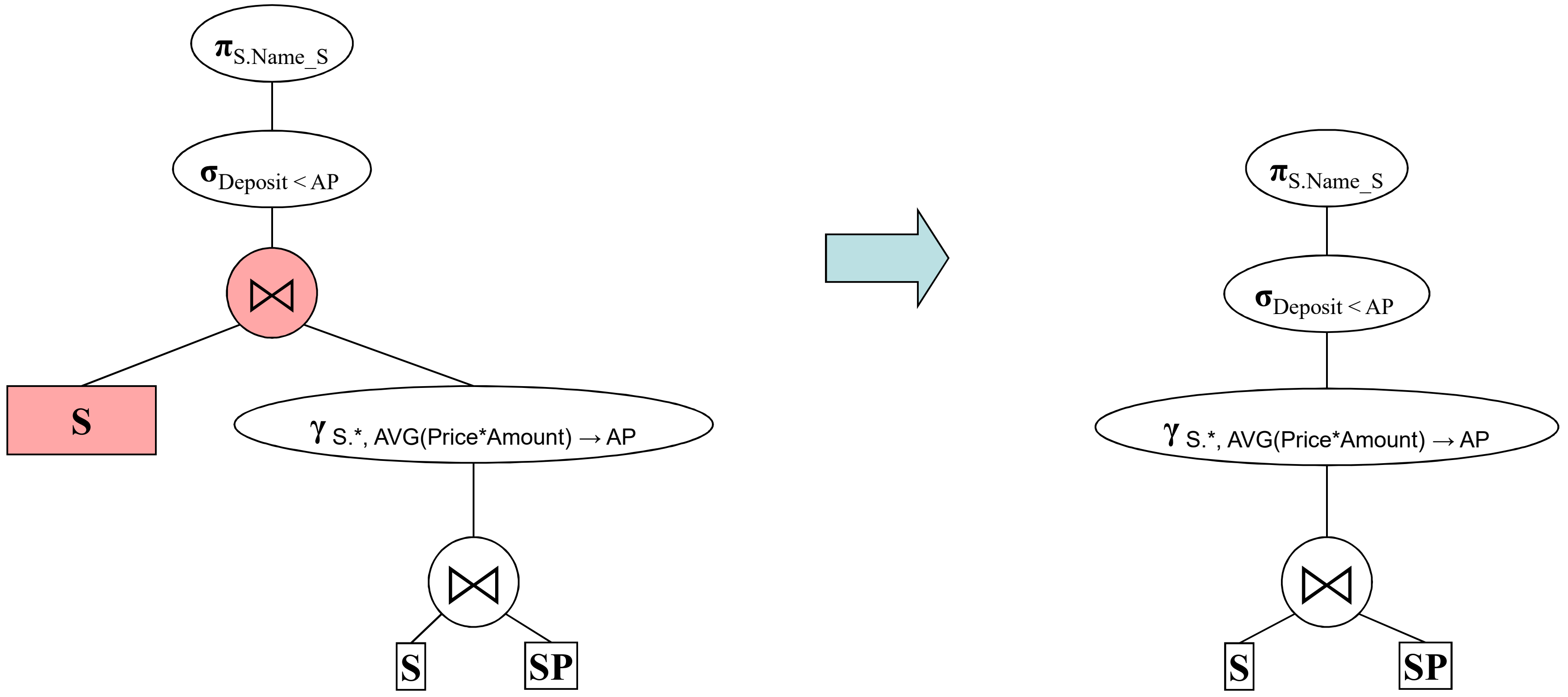

$\pi_{S1.Name\_S}$
$\sigma_{S1.Deposit < AP}$
$\sigma_{S1.ID\_S=S2.ID\_S}$
$\times$
S S1
δ
$\gamma_{S2.ID\_S, AVG(Price*Amount) \rightarrow AP}$
$\sigma_{S2.ID\_S = SP.ID\_S}$
$\times$
S S2
SP
$\pi_{S1.Name\_S}$
$\sigma_{Deposit < AP}$
$\bowtie$
S S1
$\gamma_{S2.*, AVG(Price*Amount) \rightarrow AP}$
$\bowtie$
S S2
SP

Duplicate elimination after grouping is always redundant!

Elimination of aliases



Elimination of redundant natural join


$\pi_{S.Name\_S}$
$\sigma_{Deposit < AP}$
$\bowtie$
S
γ S.*, AVG(Price*Amount) → AP
$\bowtie$
S
SP
$\pi_{S.Name\_S}$
$\sigma_{Deposit < AP}$
γ S.*, AVG(Price*Amount) → AP
$\bowtie$
S
SP