Query Optimization

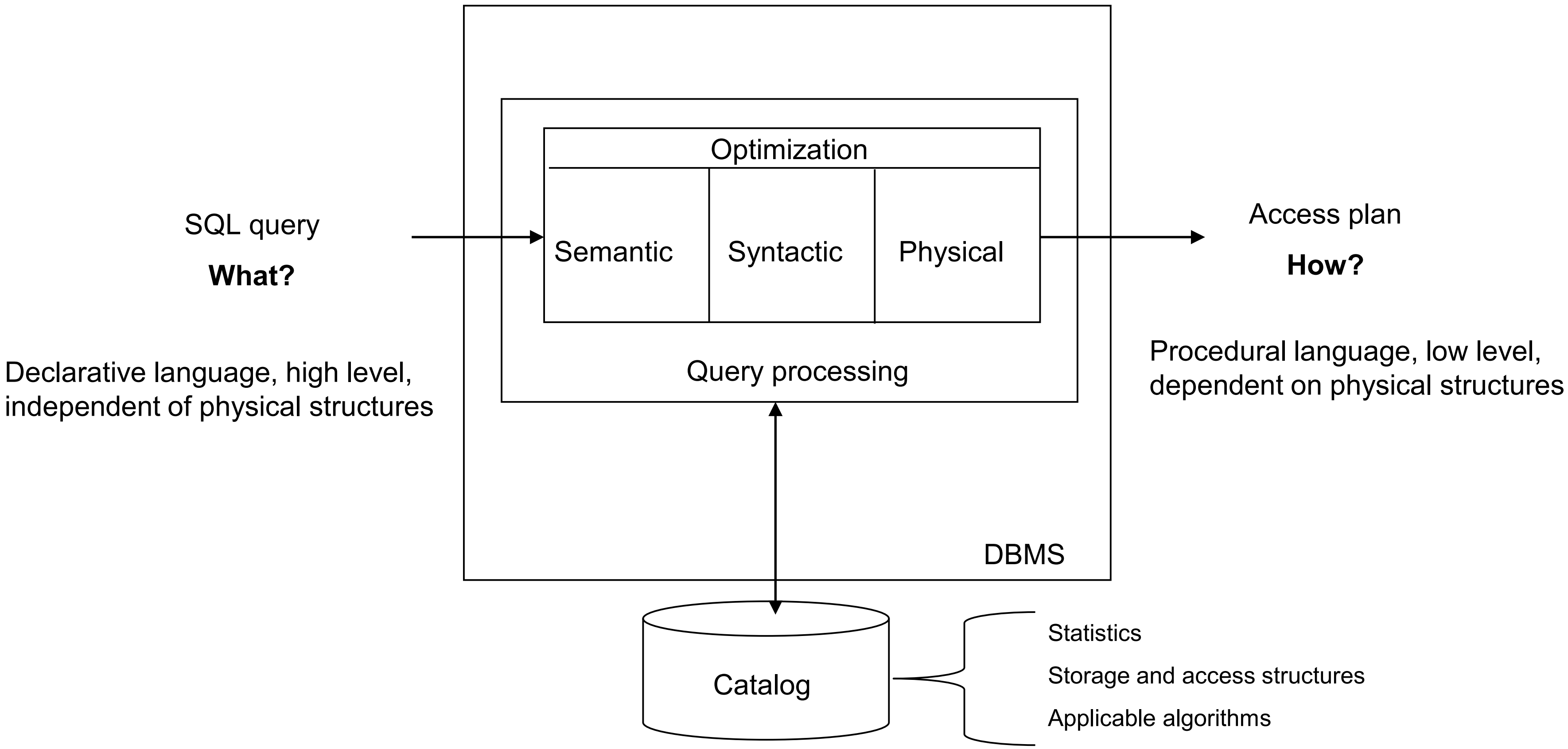
Semantic and Syntactic Optimization

About Query Optimization

- It is the last step in query processing, which also includes:
  - Query (SQL) parsing
  - Check privileges
  - Unfold view definitions

- **Input**: An SQL query over tables, syntactically correct and authorized
- **Output**: An algorithm (access plan) that must be executed by the DBMS in order to get the result
- **Goal**: Minimize the use of resources
  - In general, a DBMS does not find the optimal access plan, but it obtains an approximation (in a reasonable time)

Main Modules



Semantic Optimization

Semantic optimization

- Consists of transforming the SQL sentence into an equivalent one with a lower cost, by considering:
  - Integrity constraints (e.g., contradictions in the query with the CHECKs defined)
  - Logics-based rules

Semantic Optimization: Examples

```
CREATE TABLE students (
    id CHAR(8) PRIMARY KEY,
    mark FLOAT CHECK (mark>3)
    );
```

Constraint defined at the attribute level

```
SELECT *
FROM students
WHERE mark<2;
```

The predicate can be removed when executing the query since it is guaranteed by the attribute constraint (**constraint-based semantic optimization**)

```
SELECT *
FROM students
WHERE mark<6 AND mark>8;
```

The predicate yields an empty result (a value cannot meet both subpredicates at the same time). The query is not executed (**logics-based semantic optimization**)

```
SELECT *
FROM students
WHERE mark<6
        AND mark<7;
```

The predicate mark < 7 can be removed before executin the query since it is subsumed by mark < 6. (**logics-based semantic optimization**)

Example of Semantic Optimization (ORACLE)

- Not all the DBMS execute the same semantic optimization. For example, Oracle considers the following one:

```
SELECT *
FROM employees e, departments d
WHERE e.dpt=d.code AND d.code>5;
```

e.dpt is defined as a FK to d.code (PK)

```
SELECT *
FROM employees e, departments d
WHERE e.dpt=d.code AND d.code>5
        AND e.dpt>5;
```

Oracle propagates the selections on the PK to the FK so that the join can be avoided

Example of semantic optimization (IBM DB2)

- Some semantic optimizations are very DBMS-specific and depend on their internal physical optimizations:

```
SELECT *
FROM students
WHERE mark=5 OR mark=6;
```

⇓

DB2 deals better with IN at the physical level. Therefore, rewrites all queries with disjunctions (OR) in terms of IN cluases

```
SELECT *
FROM students
WHERE mark IN [5, 6];
```

Semantic Optimization: Exercise

Given the following tables and query, can you think of any semantic optimization that would result into an equivalent query likely to be less costly to execute?

```
CREATE TABLE students (
        id CHAR(8) PRIMARY KEY,
        mark FLOAT CHECK (mark>3),
        school CHAR(8) REFERENCES schools(id)
        );

CREATE TABLE schools (
        id CHAR(8) PRIMARY KEY,
        name CHAR(50)
        );
```

```
SELECT t.id
FROM students t, schools c
WHERE t.school=c.id;
```

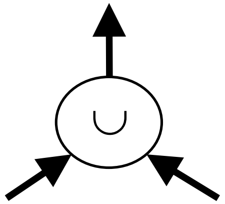
Syntactic Optimization

The Syntactic Tree

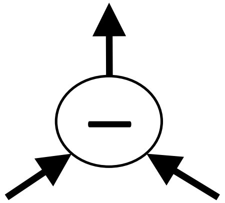
- Consists of translating the sentence from SQL into a sequence of algebraic operations in the form of syntactic tree, with minimum cost, by means of heuristics (there is more than one solution)
  - Therefore, it is responsible for translating SQL into a pipe of relational algebra operators
  - It is part of the syntactic optimization to apply heuristics in order to find an optimal pipe order
- The syntactic tree looks as follows:
  - Nodes
    - Leaves: Tables
    - Internal: Operations
    - Root: Result
  - Edges
    - Denote direct usage

Internal Nodes of the Syntactic Tree

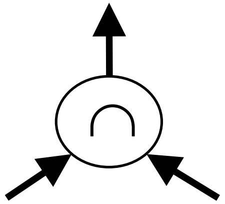
Union



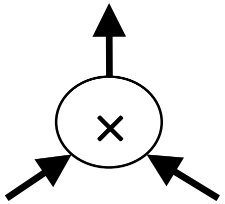
Difference



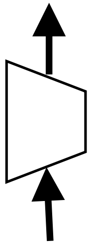
Intersection



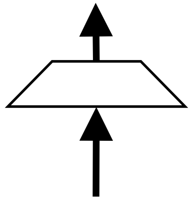
Cross product



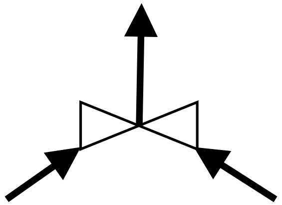
Selection



Projection



Join



Why Syntactic Optimization?

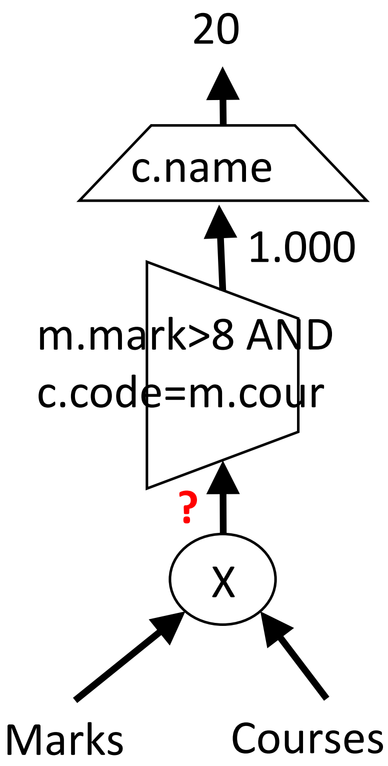
- A SQL query could be translated to different syntactic trees. The syntactic optimization aims at generating the most optimal version out of an arbitrary input syntactic tree
- Check the following slide and the three options out there. With the cardinality information provided (squared box at the top), the schema information (arrows denote PK-FK relationships), and knowing the operation applied, could you predict the missing cardinality (i.e., number of instances) reaching the internal nodes of the syntactic tree?

On the Need of Syntactic Optimization

Courses (code, name, …)
Marks (cour, stu, mark)
Students (id, …)

|Courses| = 200
|Marks| = 15000
|Students| = 3000

```
SELECT DISTINCT c.name
 FROM courses c, marks m
 WHERE c.code=m.cour
 AND m.mark>8;
```

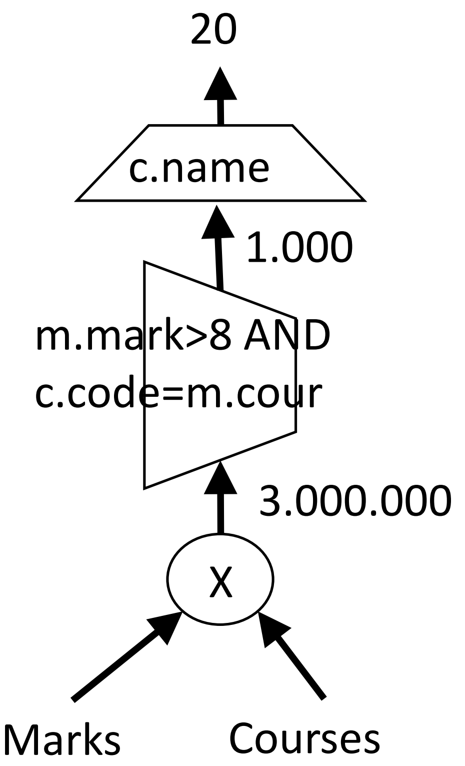


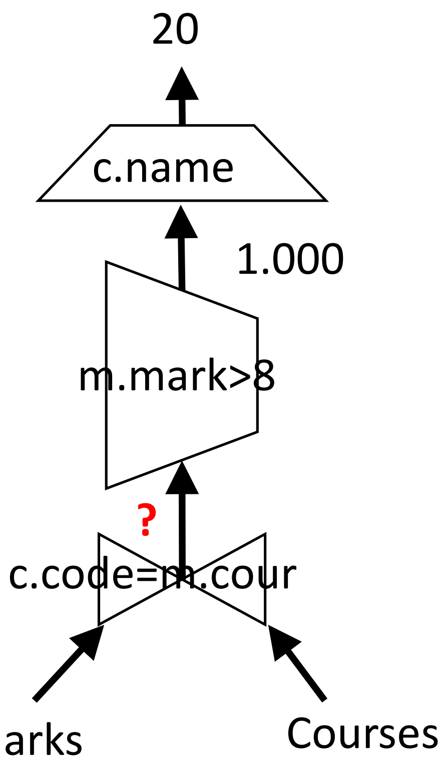
On the Need of Syntactic Optimization

Courses (<u>code</u>, name, …)
Marks (<u>cour, stu</u>, mark)
Students (<u>id</u>, …)

|Courses| = 200
|Marks| = 15000
|Students| = 3000

```
SELECT DISTINCT c.name
 FROM courses c, marks m
 WHERE c.code=m.cour
 AND m.mark>8;
```



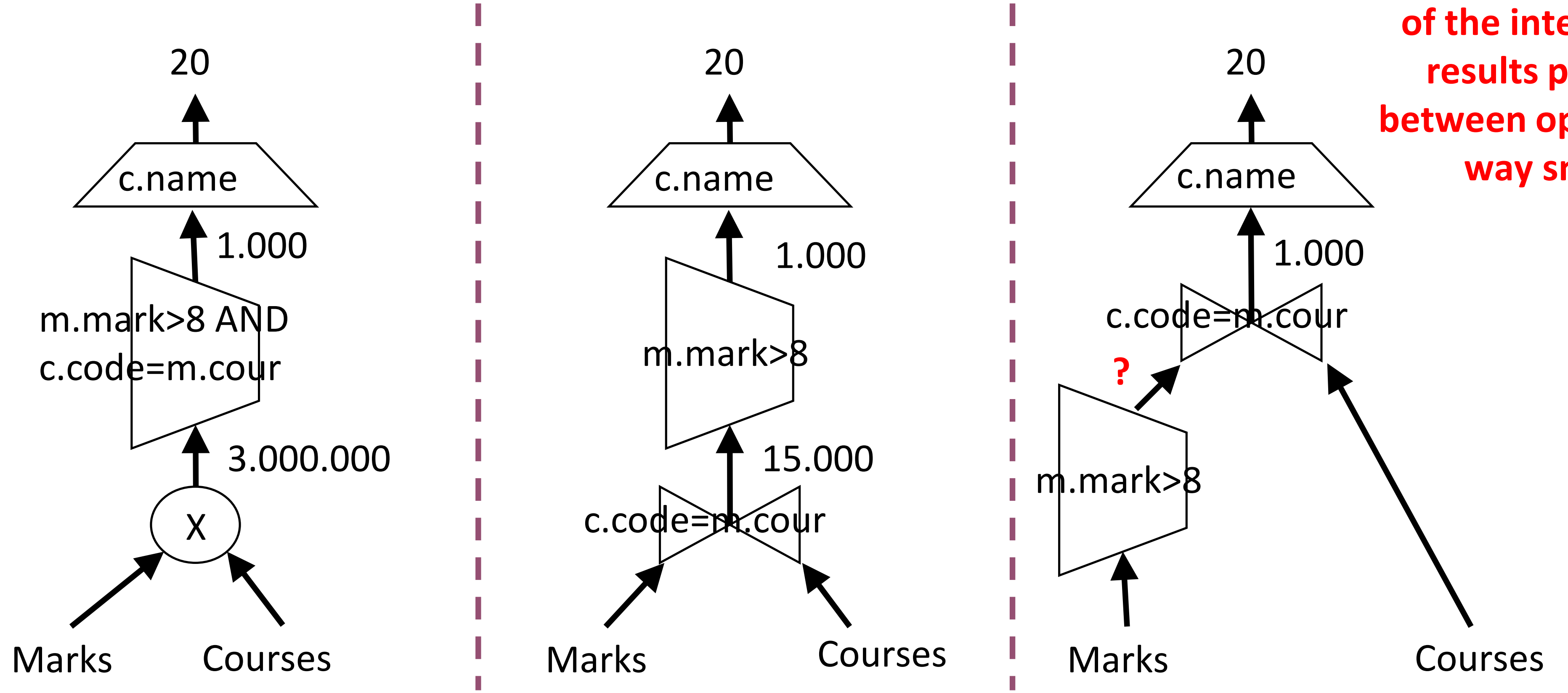


On the Need of Syntactic Optimization

Courses (code, name, …)
Marks (cour, stu, mark)
Students (id, …)

|Courses| = 200
|Marks| = 15000
|Students| = 3000

SELECT DISTINCT c.name
FROM courses c, marks m
WHERE c.code=m.cour
AND m.mark>8;



The third option is clearly better as the size of the intermediate results pipelined between operations is way smaller

Syntactic Optimization: Overview

The syntactic optimitzation follows these steps:

[**INPUT**: A Syntactic Tree]

- Apply heuristics to obtain a near optimal syntactic tree
  - To guarantee that the output optimised tree is equivalent to the input tree, the input tree can only be manipulated by means of given **equivalence rules**
- The resulting tree is passed to the physical optimizer

[**OUTPUT**: An Optimized Syntactic Tree]

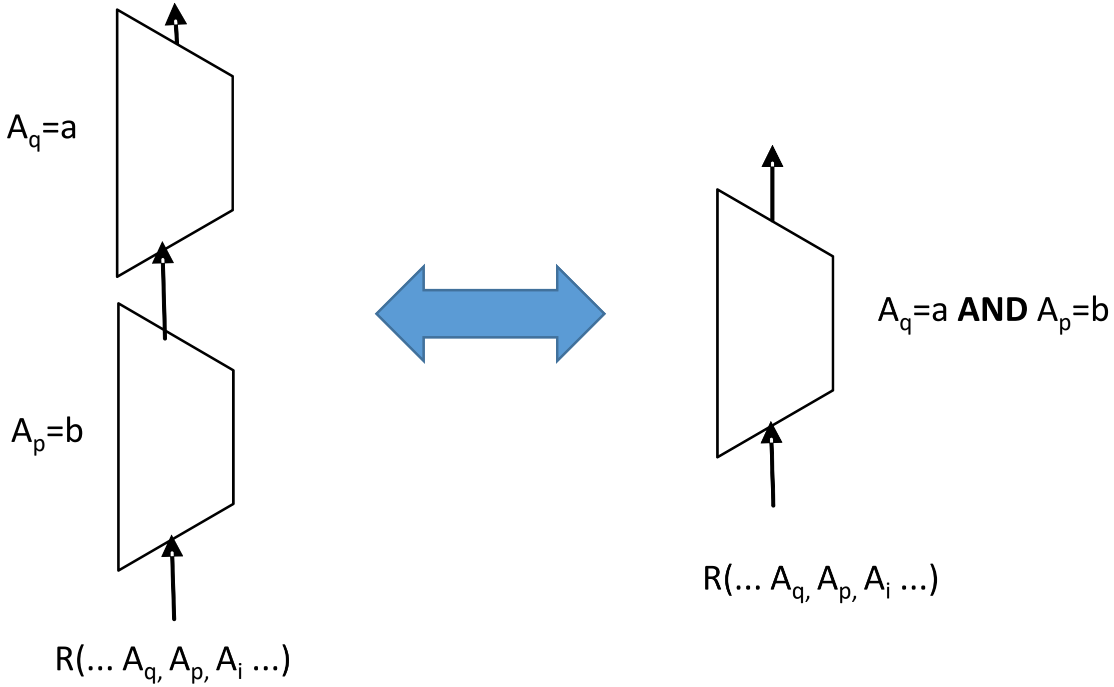
Optimizing the Syntactic Tree

- Objective:
  - **Reduce the size of data reaching the intermediate nodes**
- Steps:
  1. Split the selection predicates into simple clauses
  2. Lower selections as much as possible
  3. Group consecutive selections
     - Simplify them if possible
  4. Lower projections as much as posible
     - Do not leave them just on a table
  5. Group consecutive projections
     - Simplify them if possible

<u>All</u> steps must be executed via equivalence rules

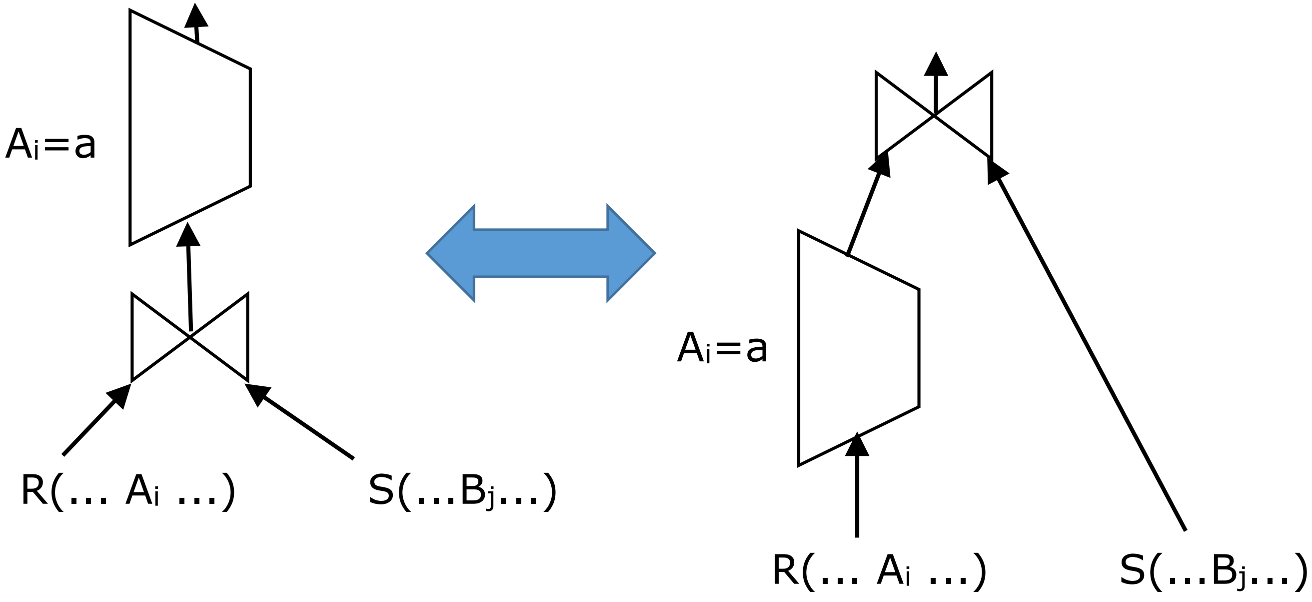
Equivalence rules (I)

Splitting/grouping selections:



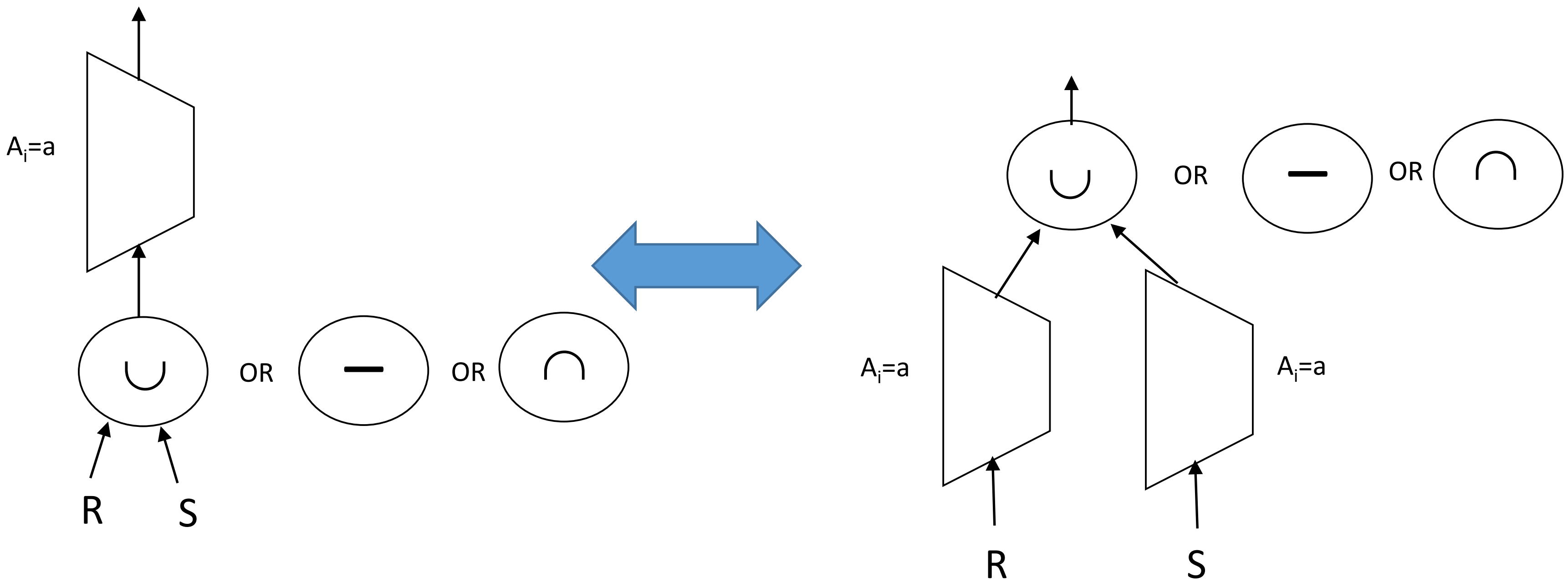
Equivalence rules (II)

Commuting the precedence of selection and join:



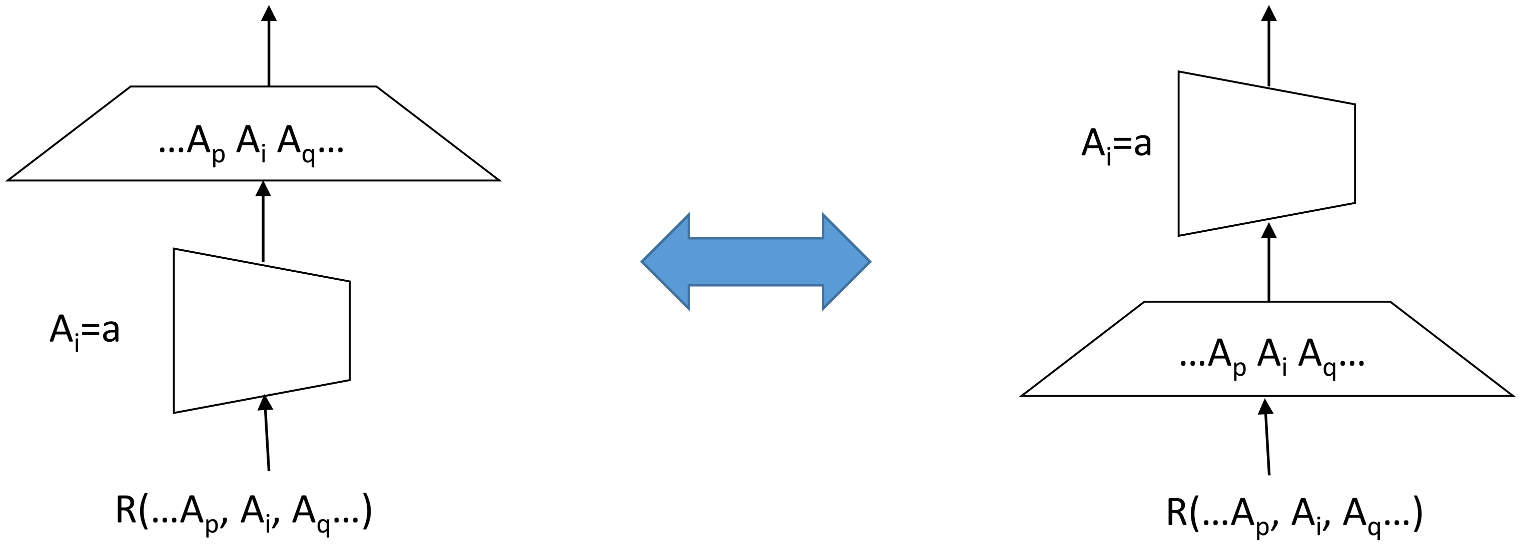
Equivalence rules (III)

Commuting the precedence of selection & union / difference / intersection



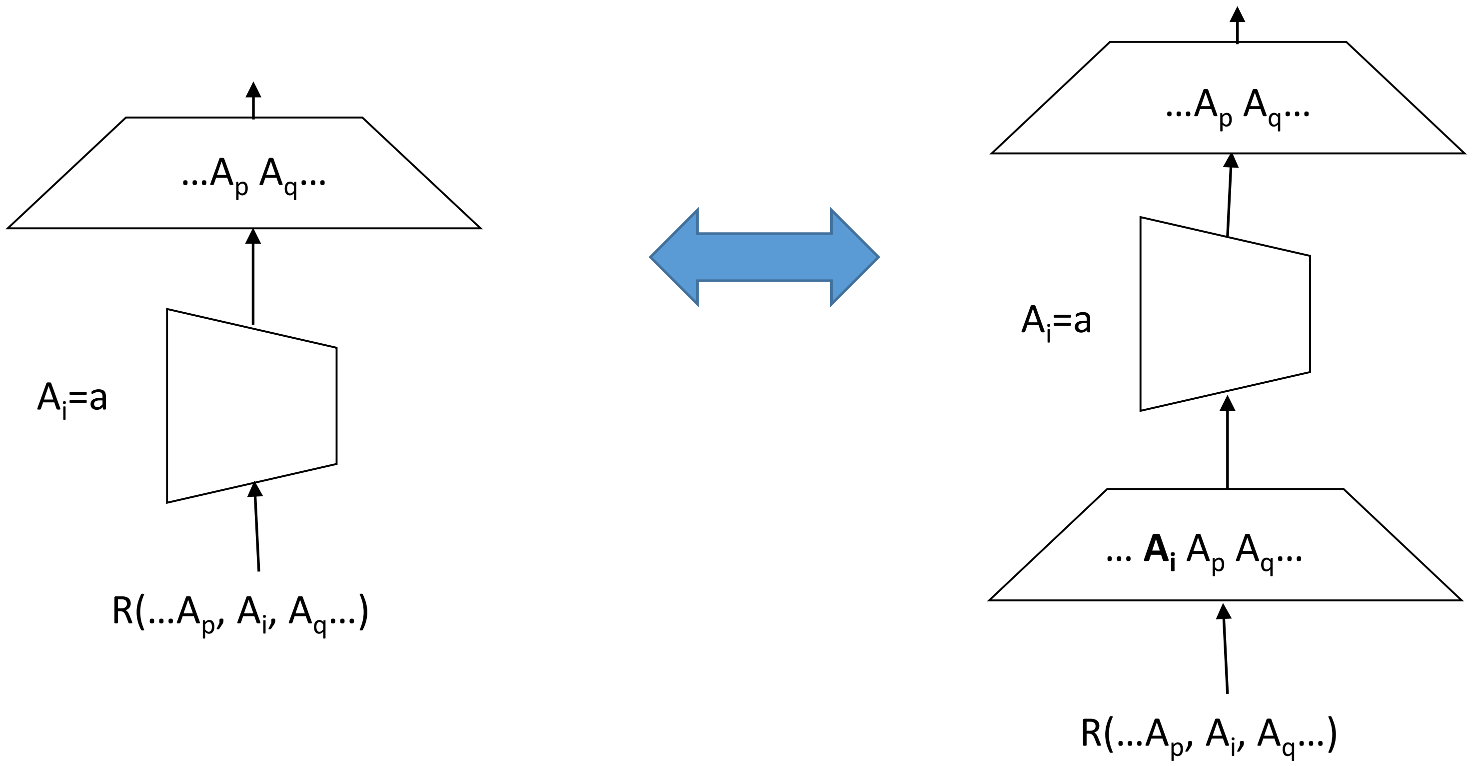
Equivalence rules (IV)

Commuting the precedence of selection & projection when the selection attribute belongs to the projection attributes:



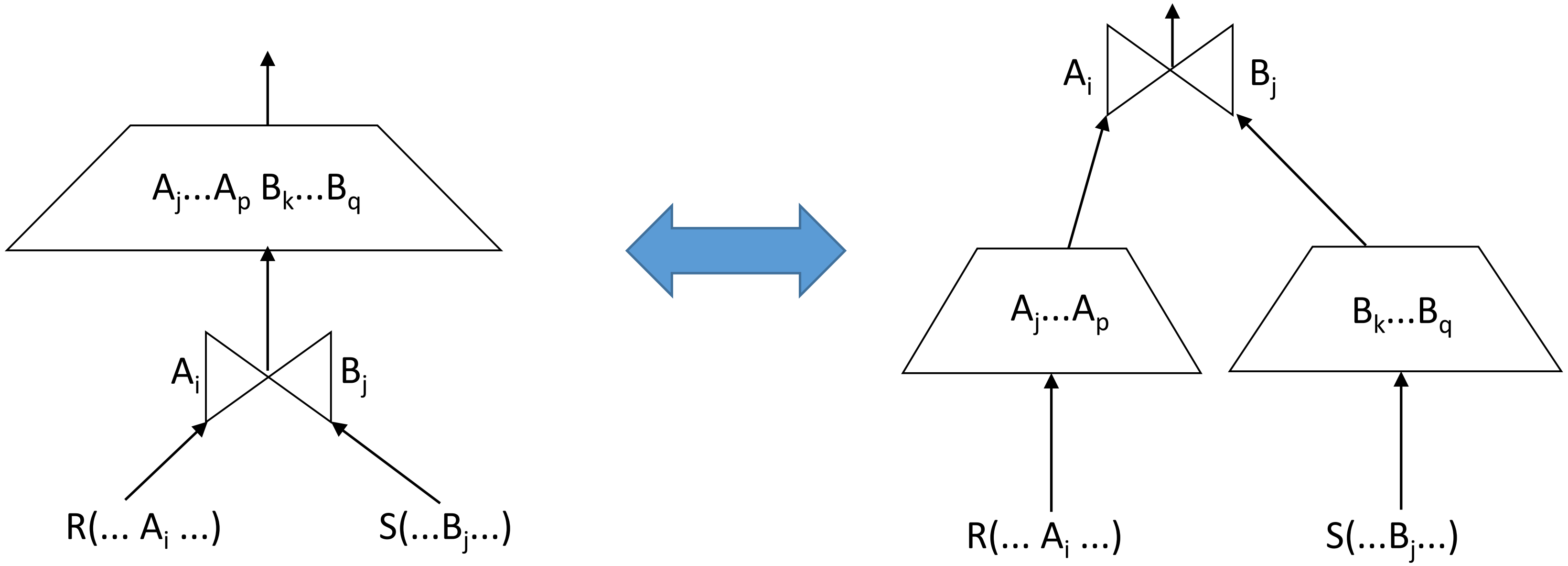
Equivalence rules (V)

Commuting the precedence of selection & projection when the selection attribute does **not** belong to the projection attributes:



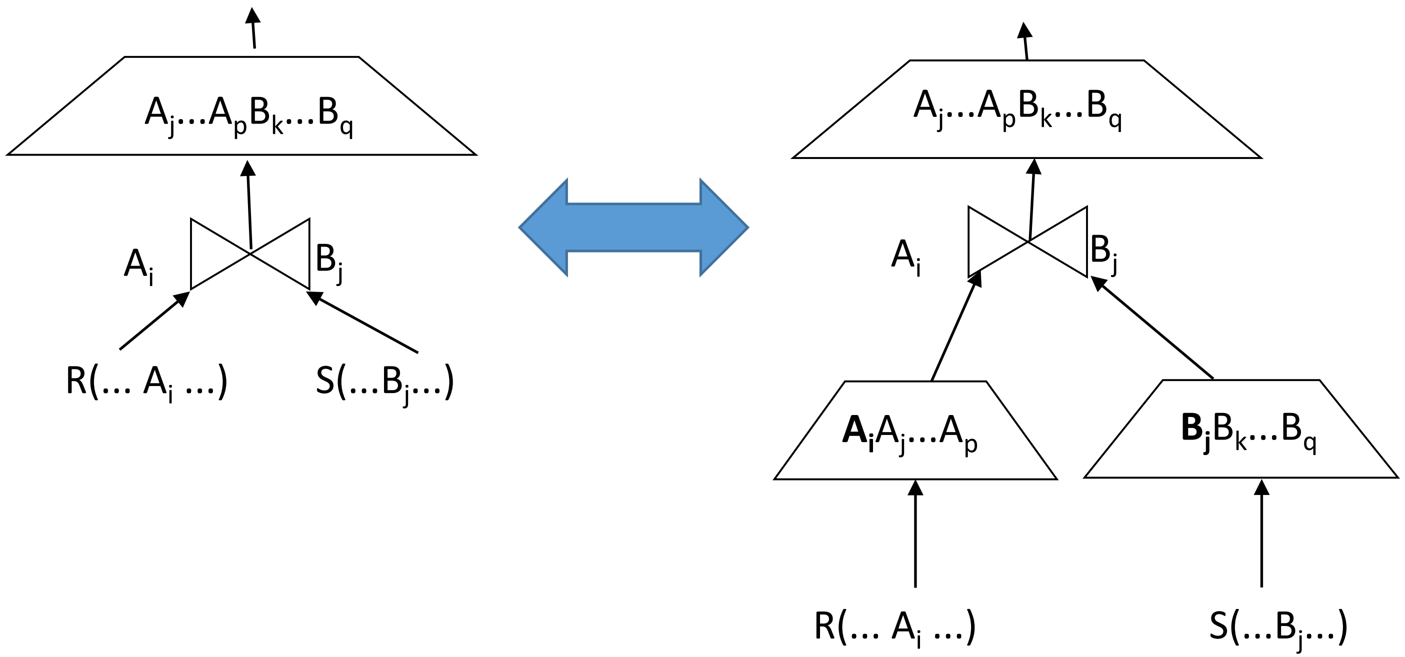
Equivalence rules (VI)

Commuting the precedence of projection & join when the join attributes belong to the projection attributes:



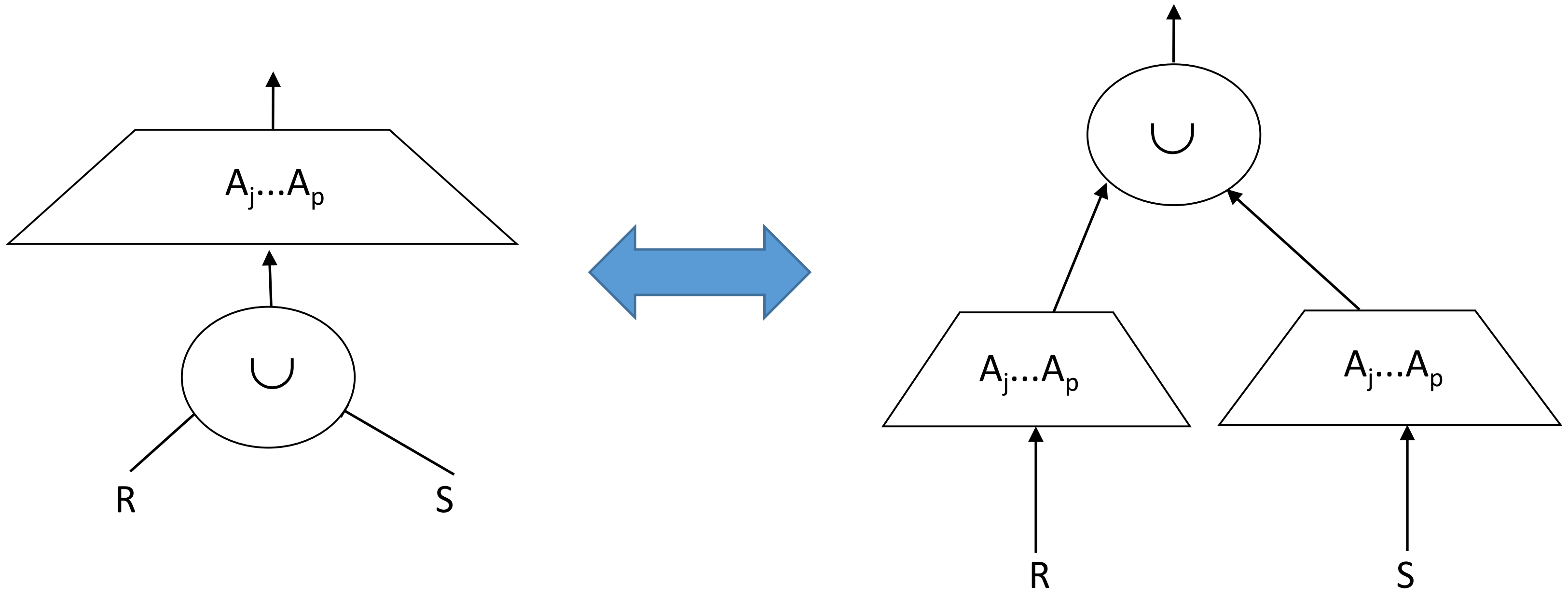
Equivalence rules (VII)

Commuting the precedence of projection & join when one (or both) join attribute does (do) not belong to the projection attributes:



Equivalence rules (VIII)

Commuting the precedence of projection & union:

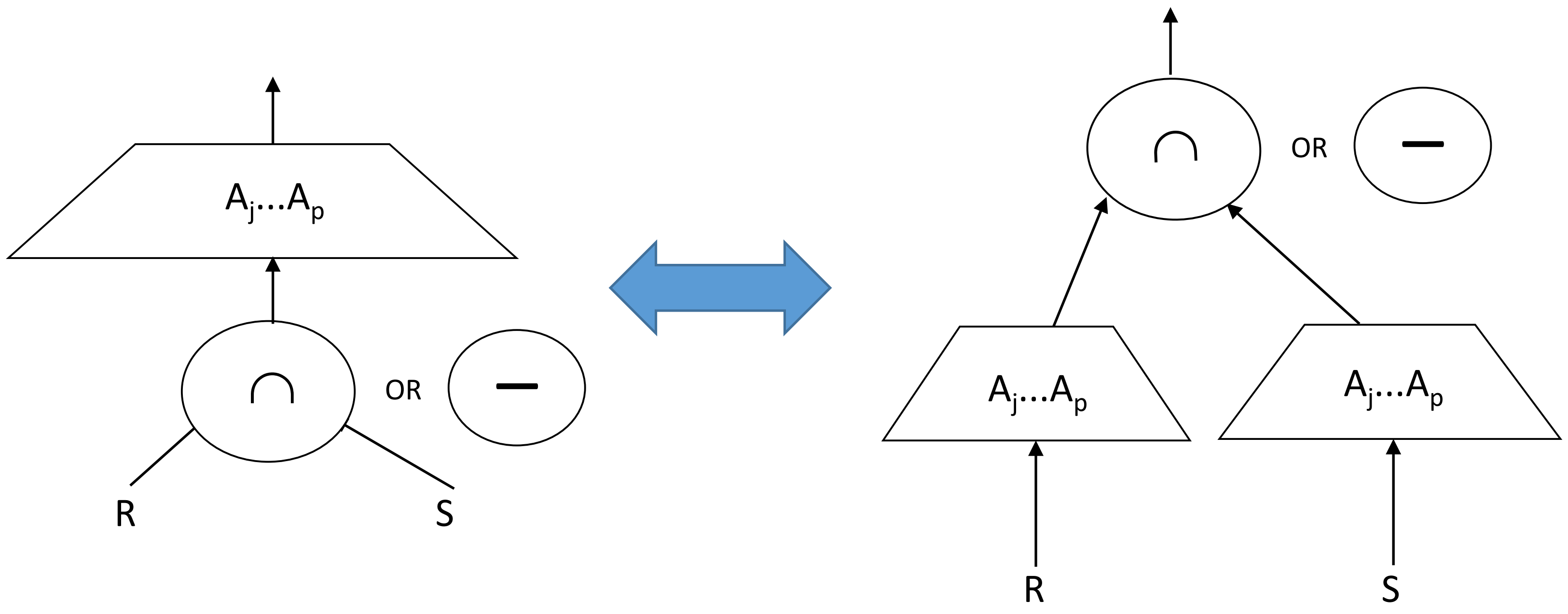


Important:

- Projection & intersection precedence cannot be freely commuted
- Projection & difference precedence cannot be freely commuted

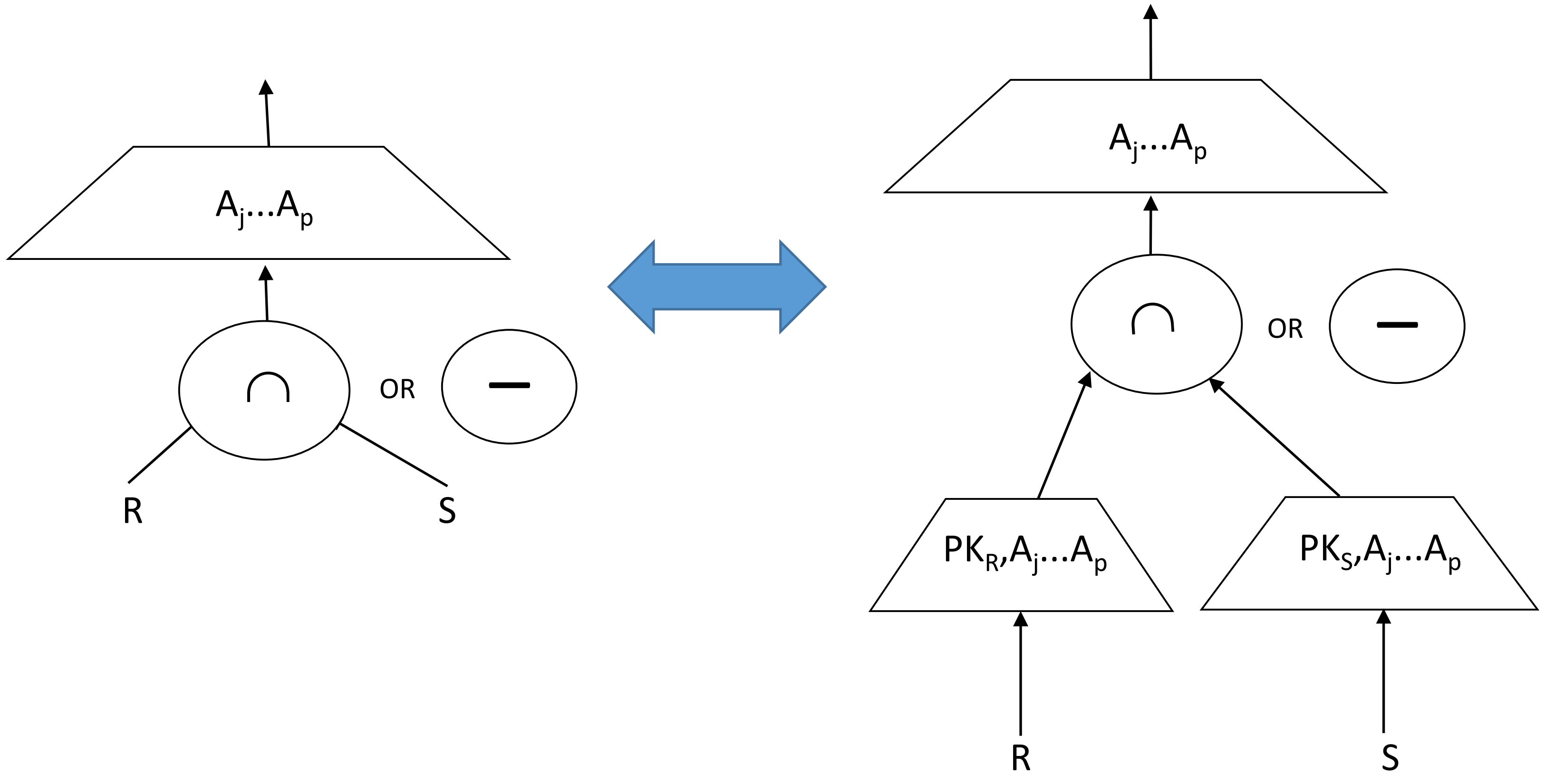
Equivalence rules (IX)

Commuting the precedence of projection & intersection/difference when PK_R, PK_S belong to the projection attributes:



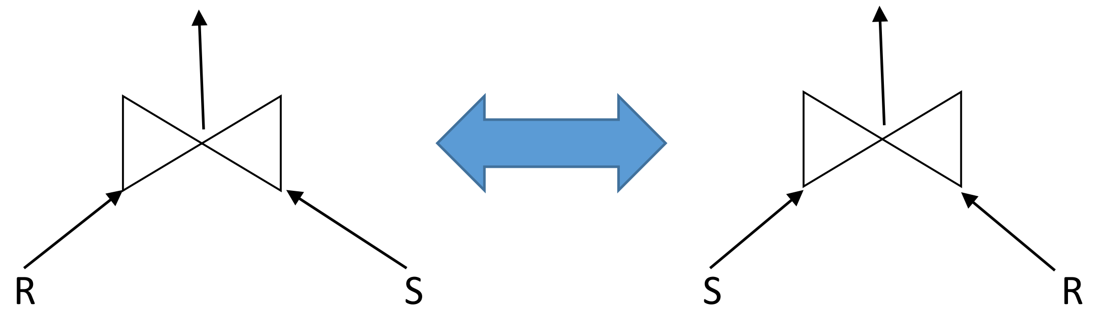
Equivalence rules (X)

Commuting the precedence of projection & intersection/difference when PK_R, PK_S do not belong to the projection attributes:



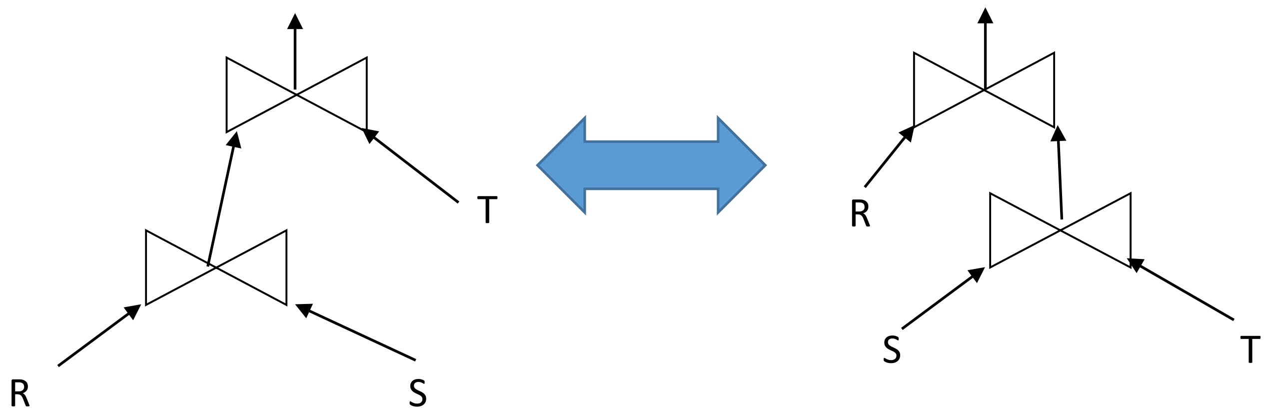
Equivalence rules (XI)

Commuting join branches:



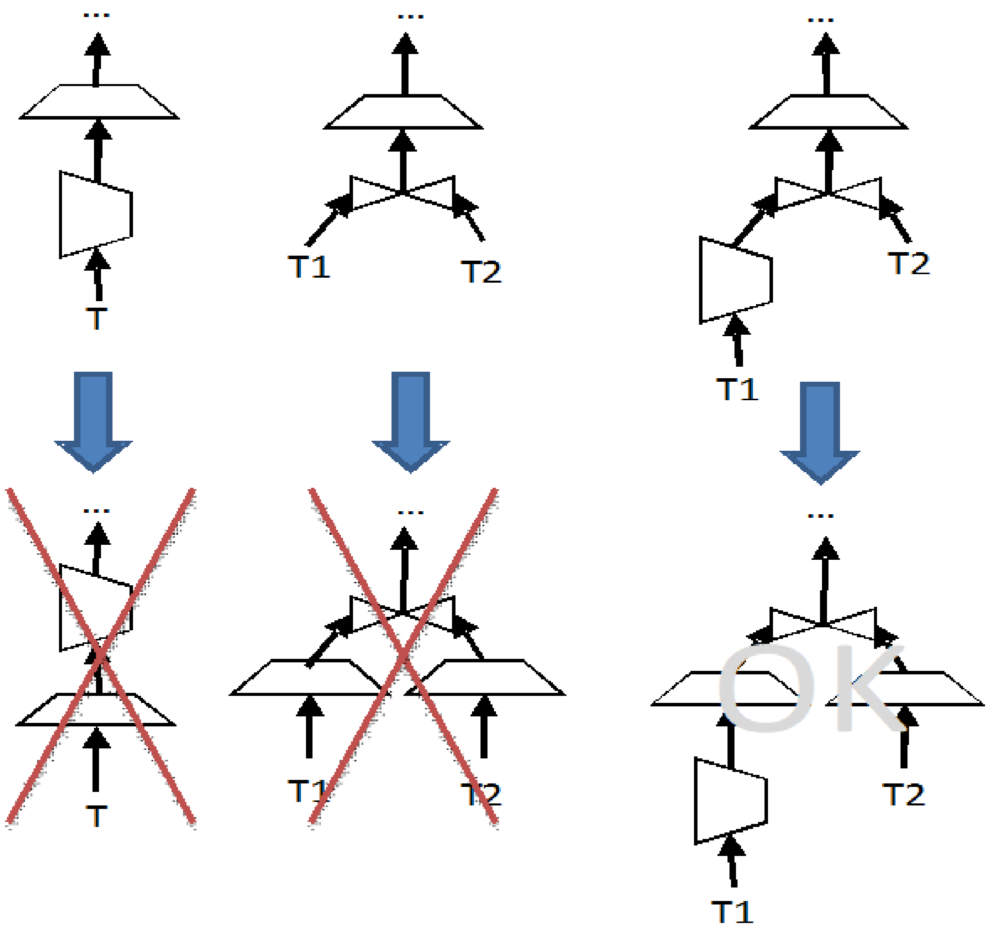
Equivalence rules (XII)

Associating join branches:



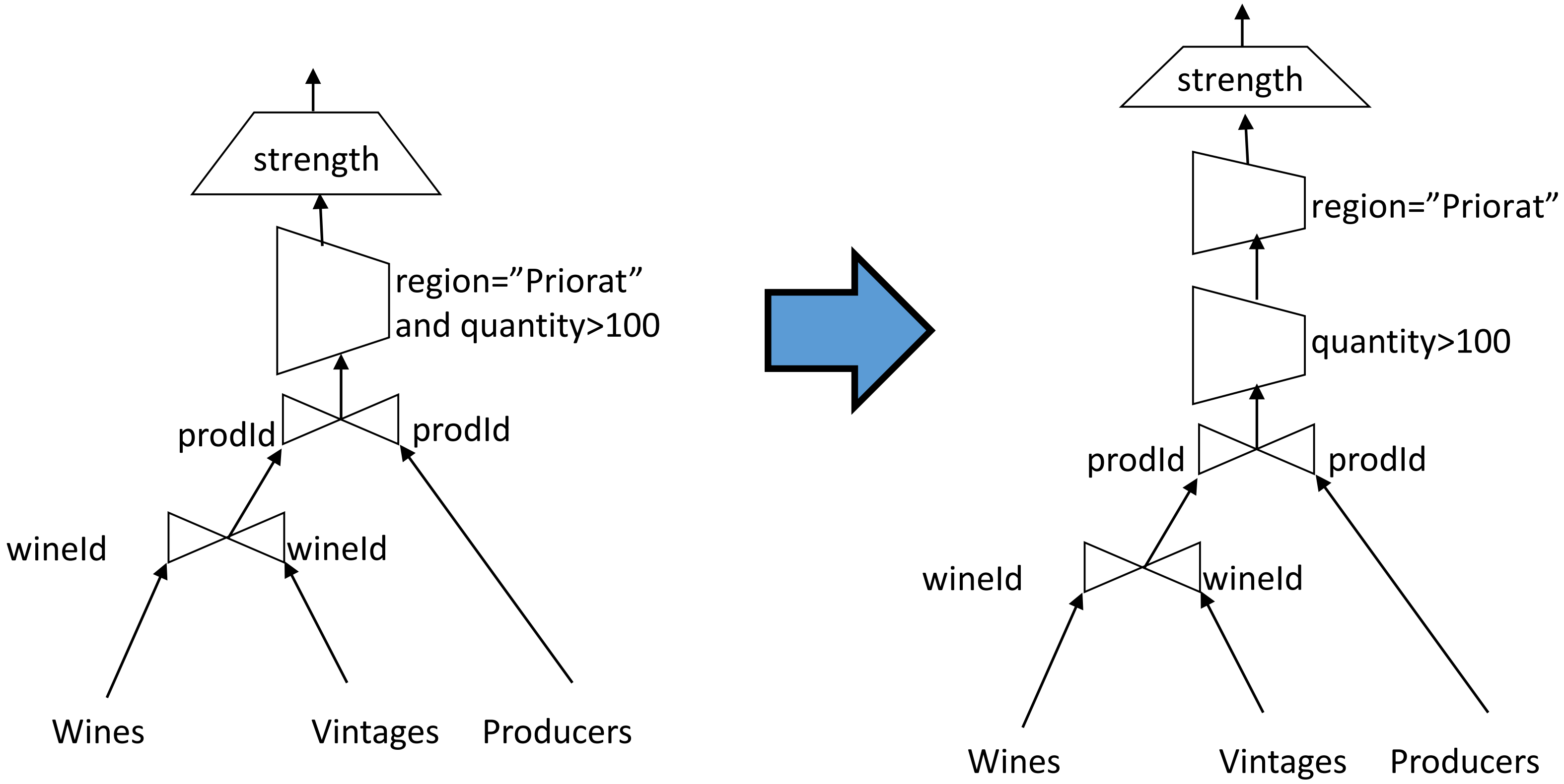
Note About Projections

- It is meaningless to push projections down to tables



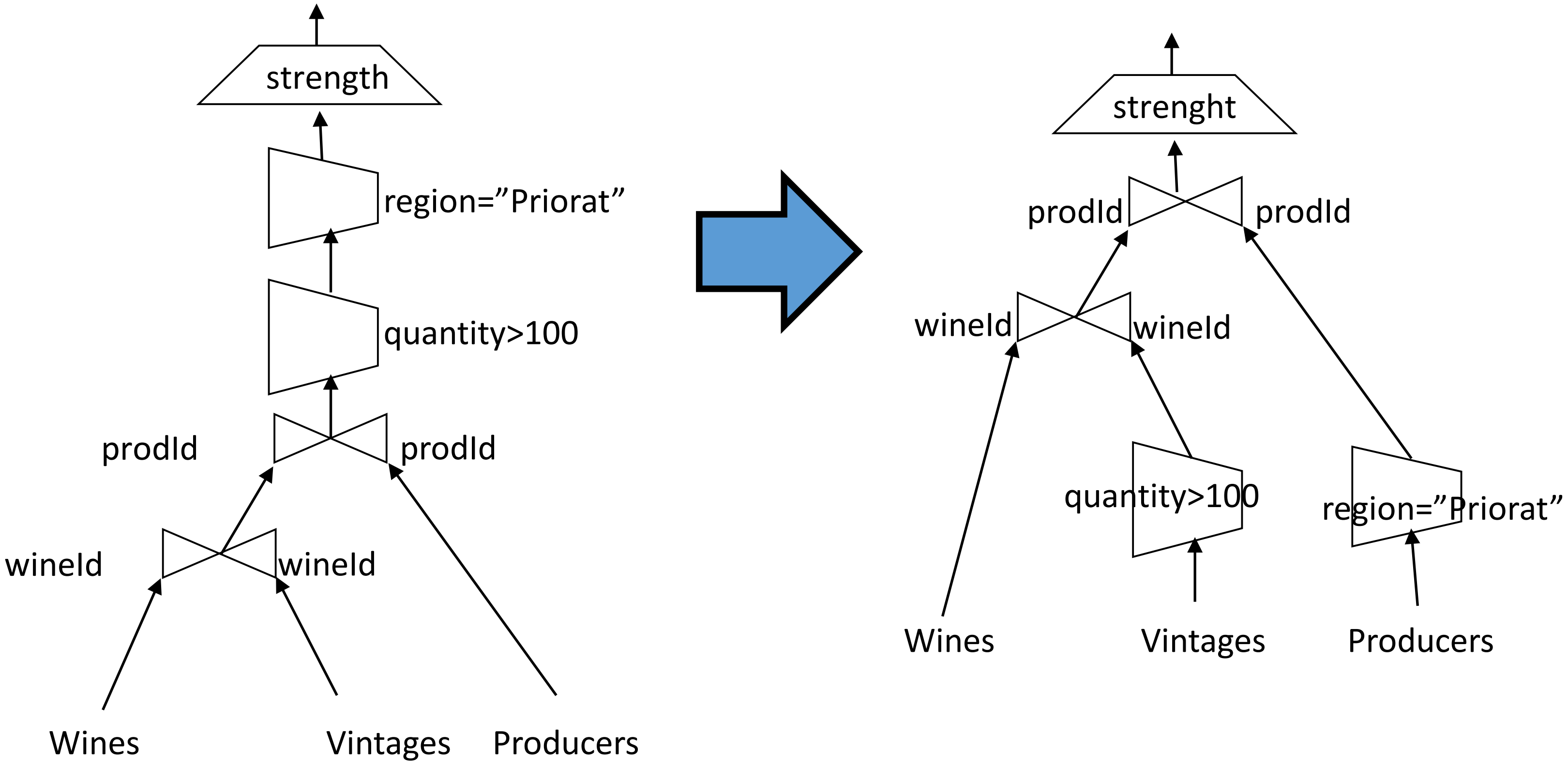
Syntactic Optimization: Example

1. **Split the selection predicates into simple clauses**
2. Lower selections as much as possible
3. Group consecutive selections (simplify them if possible)
4. Lower projections as much as possible (do not leave them just on a table)
5. Group consecutive projections (simplify them if possible)



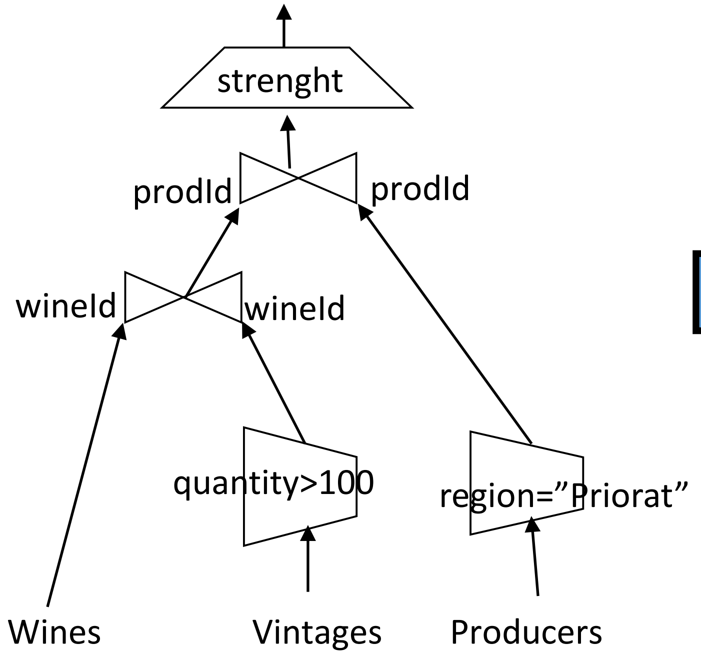
Syntactic Optimization: Example

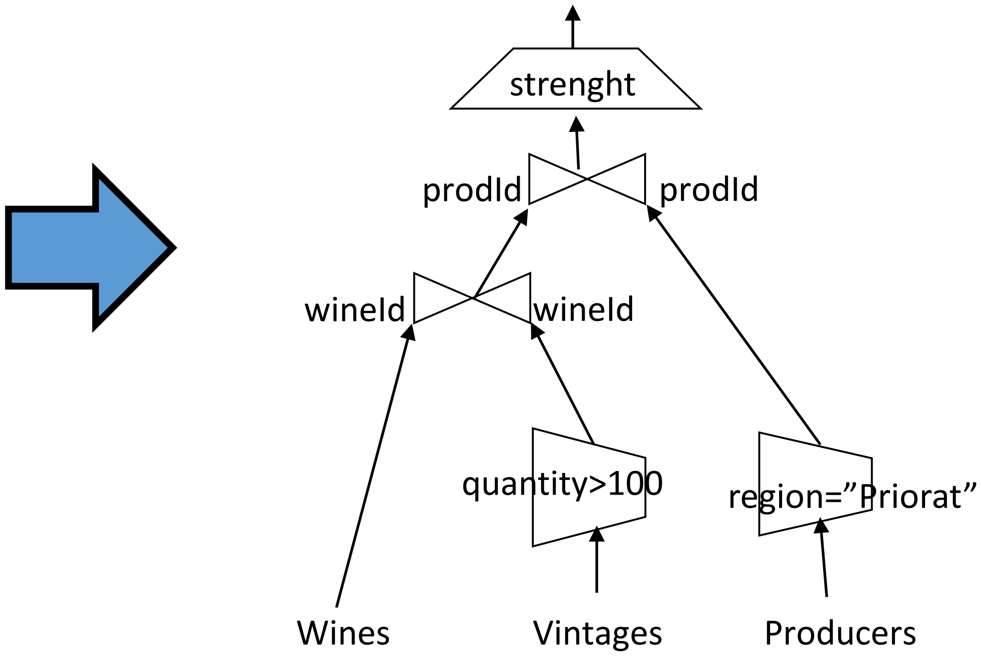
1. Split the selection predicates into simple clauses
2. Lower selections as much as possible
3. Group consecutive selections (simplify them if possible)
4. Lower projections as much as possible (do not leave them just on a table)
5. Group consecutive projections (simplify them if possible)



Syntactic Optimization: Example

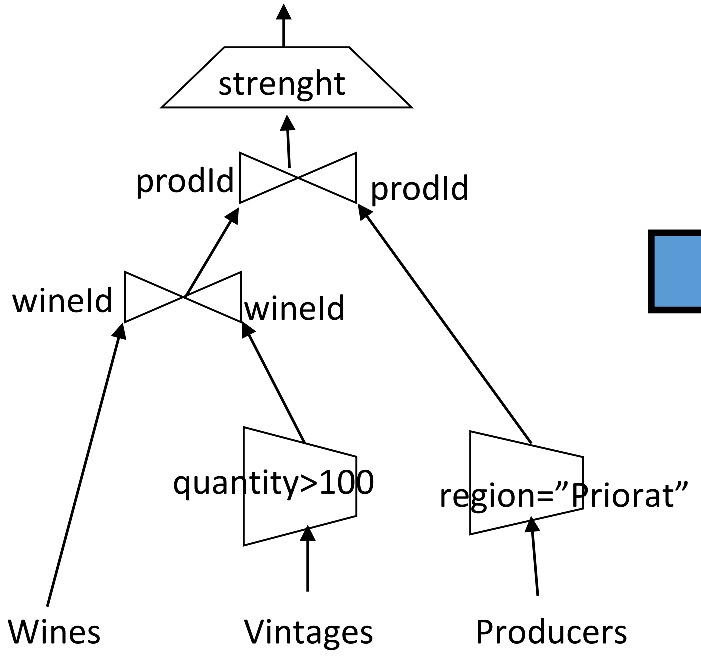
1. Split the selection predicates into simple clauses
2. Lower selections as much as possible
3. **Group consecutive selections (simplify them if possible)**
4. Lower projections as much as possible (do not leave them just on a table)
5. Group consecutive projections (simplify them if possible)

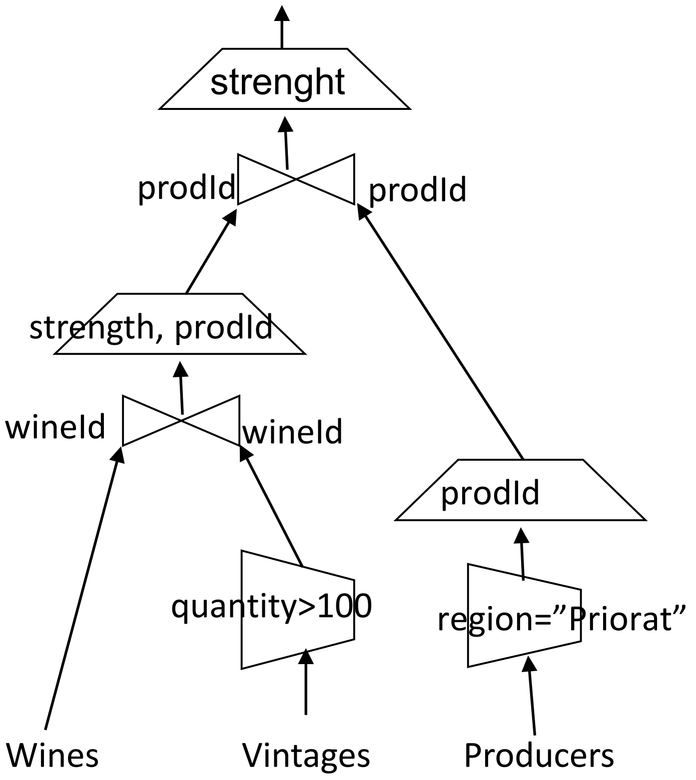




Syntactic Optimization: Example

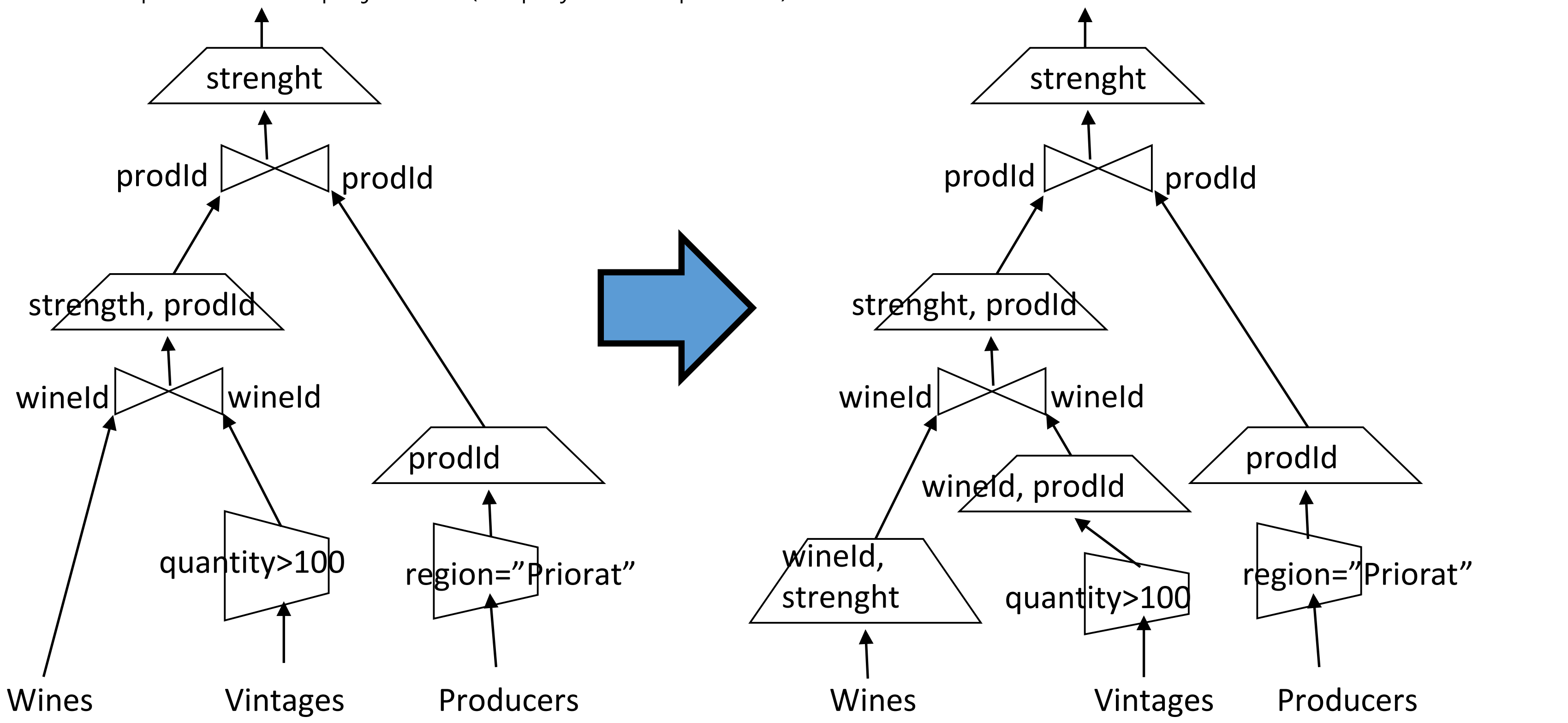
1. Split the selection predicates into simple clauses
2. Lower selections as much as possible
3. Group consecutive selections (simplify them if possible)
4. **Lower projections as much as possible (do not leave them just on a table)**
5. Group consecutive projections (simplify them if possible)





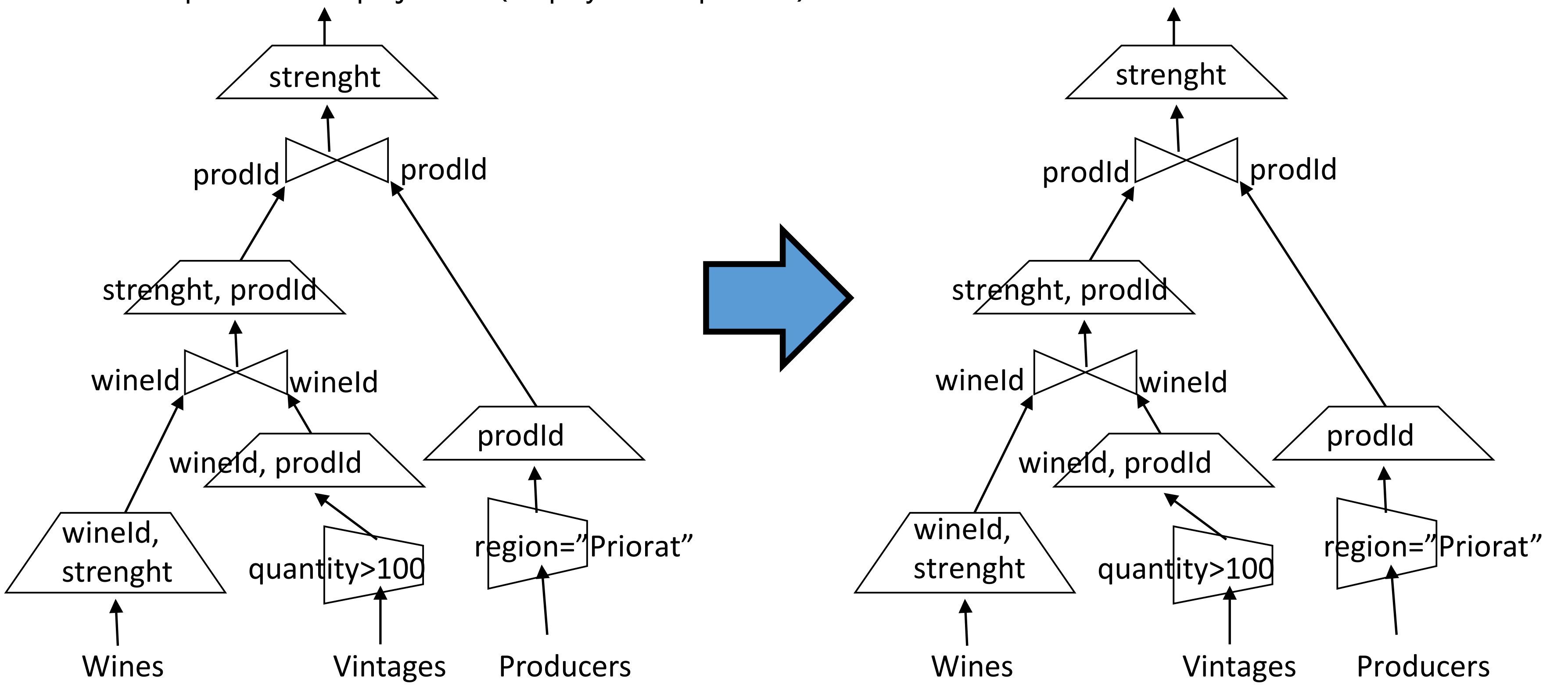
Syntactic Optimization: Example

1. Split the selection predicates into simple clauses
2. Lower selections as much as possible
3. Group consecutive selections (simplify them if possible)
4. **Lower projections as much as possible (do not leave them just on a table)**
5. Group consecutive projections (simplify them if possible)



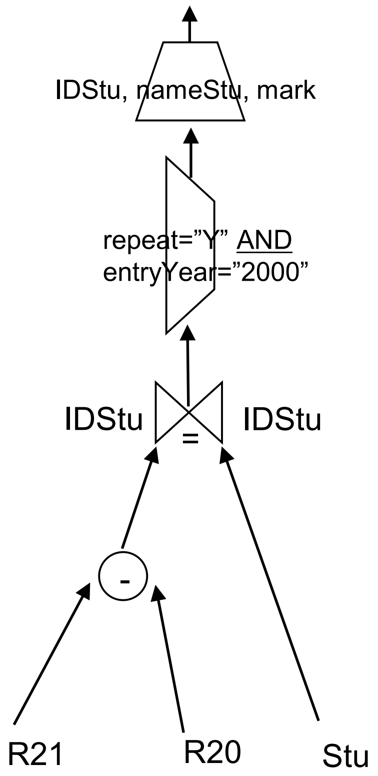
Syntactic Optimization: Example

1. Split the selection predicates into simple clauses
2. Lower selections as much as possible
3. Group consecutive selections (simplify them if possible)
4. Lower projections as much as possible (do not leave them just on a table)
5. Group consecutive projections (simplify them if possible)



Syntactic Tree Optimization: Exercise

Initial Syntactic Tree:



Schema:

Subjects(IDSub, nameSub, credits)
Students(IDStu, nameStu, degree, entryYear)
Registration2020(IDStu, IDSub, mark, repeat)
{IDStu} FK to Students
{IDSub} FK to Subjects
Registration2021(IDStu, IDSub, mark, repeat)
{IDStu} FK to Students
{IDSub} FK to Subjects

Summary

- Query optimization phases
  - Semantic
  - Syntactic
  - Physical
- Syntactic optimization algorithm
  - Relational algebra equivalence rules

Bibliography

- Y. Ioannidis. *Query Optimization*. ACM Computing Surveys, vol. 28, num. 1, March 1996
- R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd Edition, 2003
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007