Query Optimization
Heuristic Optimization

SWEN 304

Trimester 2, 2017

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

slides by: Pavle Morgan & Hui Ma

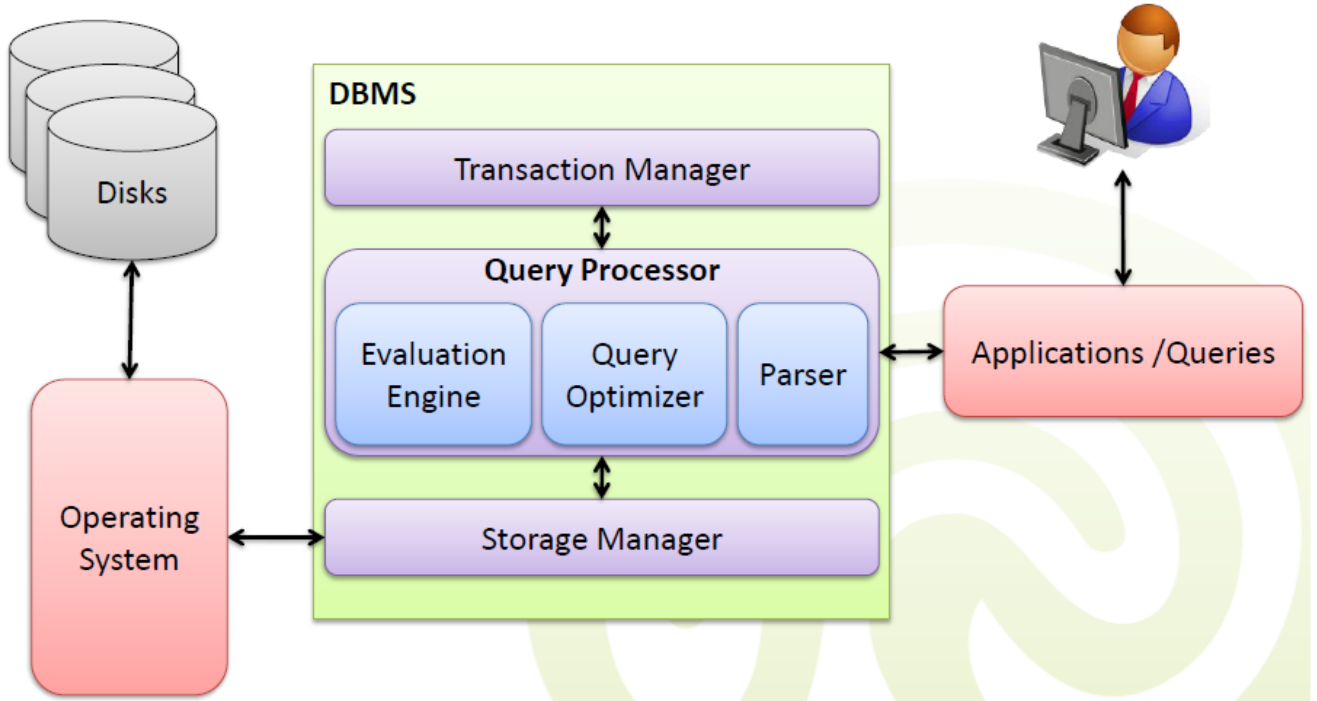
Outline

- Query Processing in DBMS
- Query optimization techniques
- Heuristic query optimization
  - Translating SQL into relational algebra
  - Reordering query operations
  - Transformation rules for relational algebra operations

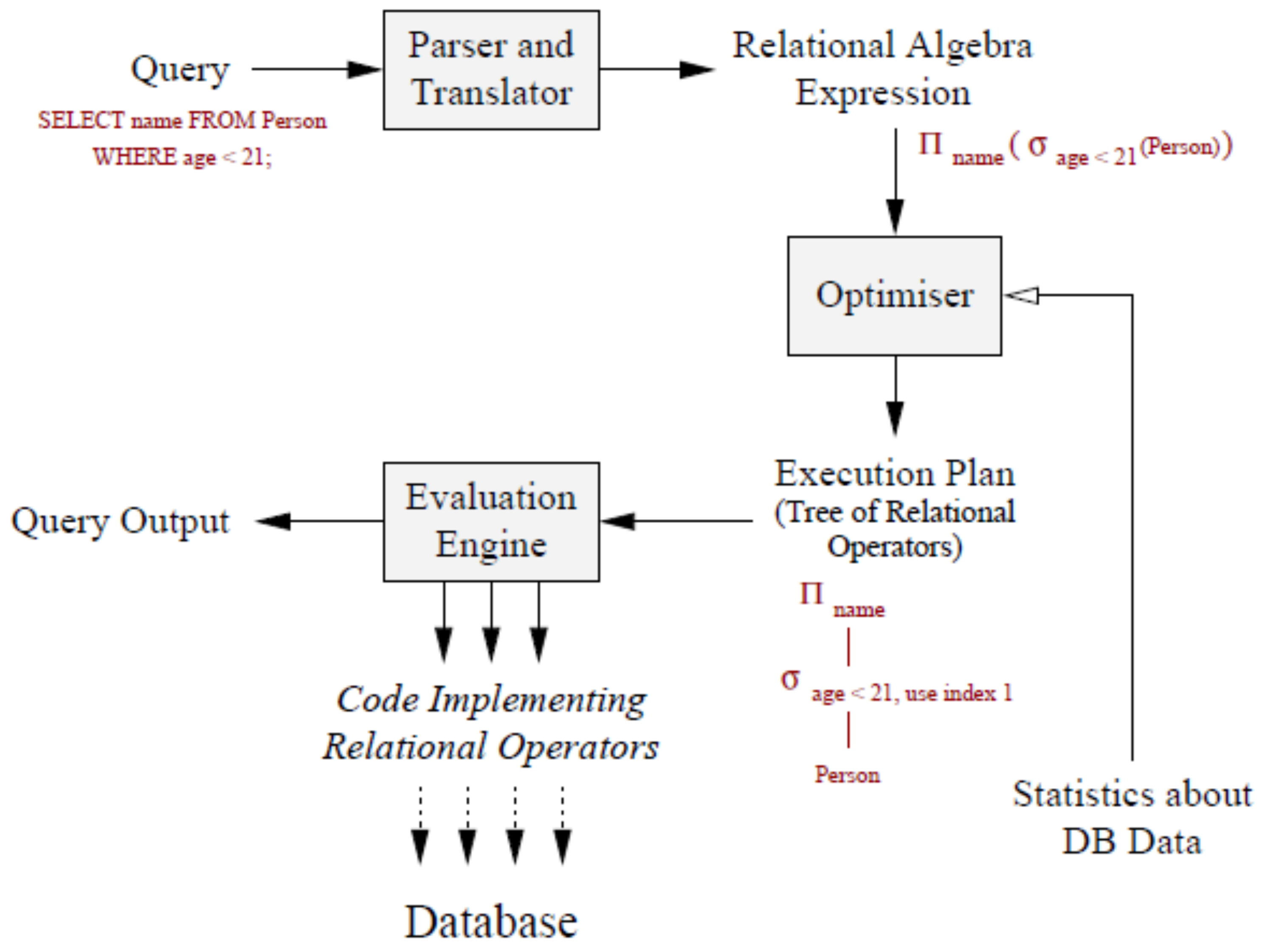
- *Readings from the textbook:*
  - Chapter 19: Algorithms for Query Processing and Optimization
  - Chapters 17: Disk Storage, Basic File Structures, and Hashing (Sections: 17.2, to 17.8)
  - Chapter 18: Indexing Structures for Files (Sections: 18.1 to 18.5)

Query Processing in DBMS

- Users/applications submit queries to the DBMS
- The DBMS processes queries before evaluating them
  - Recall: DBMS mainly use declarative query languages (such as SQL)
  - Queries can often be evaluated in different ways
  - SQL queries do not determine how to evaluate them



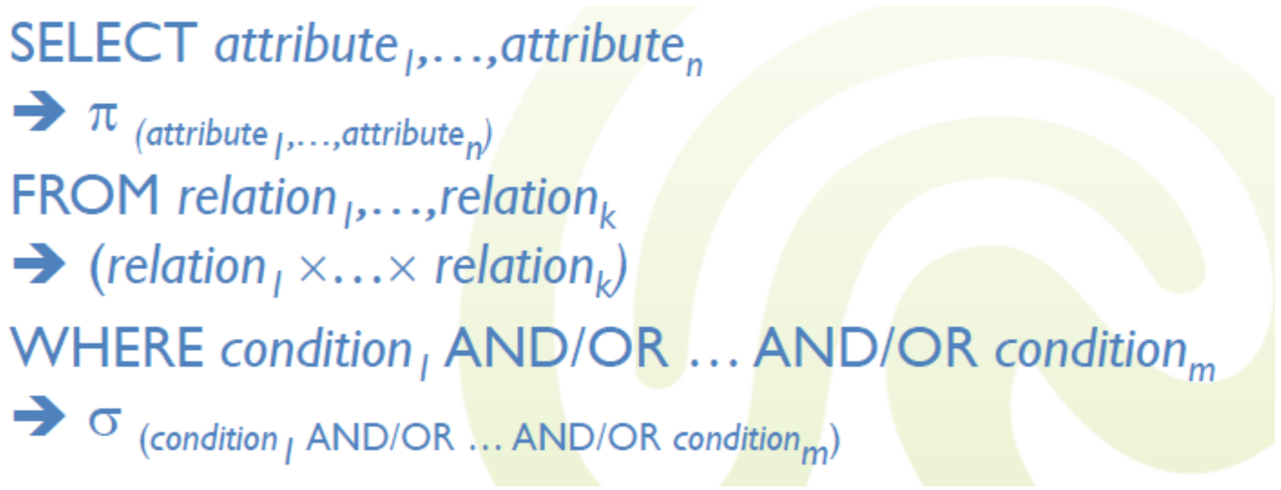
Query Processing in DBMS



Query
SELECT name FROM Person WHERE age < 21;
Parser and Translator
Relational Algebra Expression
$\Pi_{name}(\sigma_{age<21}(Person))$
Optimiser
Statistics about DB Data
Execution Plan (Tree of Relational Operators)
Π_{name}
$\sigma_{age<21, use\ index\ 1}$
Person
Evaluation Engine
Query Output
Code Implementing Relational Operators
Database

Query Optimisation

- Query preparation:
  - Decompose query into query blocks containing
    - Exactly one SELECT and FROM clause
    - At most one WHERE, GROUP BY and HAVING clause



SELECT $attribute_1, \ldots, attribute_n$

$\rightarrow \pi_{(attribute_1, \ldots, attribute_n)}$

FROM $relation_1, \ldots, relation_k$

$\rightarrow (relation_1 \times \ldots \times relation_k)$

WHERE $condition_1$ AND/OR … AND/OR $condition_m$

$\rightarrow \sigma_{(condition_1 \text{ AND/OR } \ldots \text{ AND/OR } condition_m)}$

- Nested queries within a query are identified as separate query blocks
- Aggregate operators in SQL must be included in the extended algebra

Heuristic Query Optimization: An Example

Student({Lname, Fname, <u>StudId</u>, Major})

Grades({<u>StudId</u>, <u>CrsId</u>, Grade})

Course({Cname, <u>CrsId</u>, Points, Dept})

- SQL query:

```
SELECT StudId, Lname
FROM Student s, Grades g, Course c
WHERE s.StudId=g.StudId AND g.CrsId= c.CrsId
AND Grade = 'A+' AND Dept = 'Comp' AND Major = 'Math';
```

- Relational Algebra:

$$\pi_{\text{StudId, LName}} (\sigma_{\text{s.StudId=g.StudId} \wedge \text{ g.CrsId=c.CrsId} \wedge \text{Grade='A+'} \wedge \text{Major = 'Math'} \wedge \text{Dept = 'Comp'}} (\text{Course} \times (\text{Student} \times \text{Grades})))$$

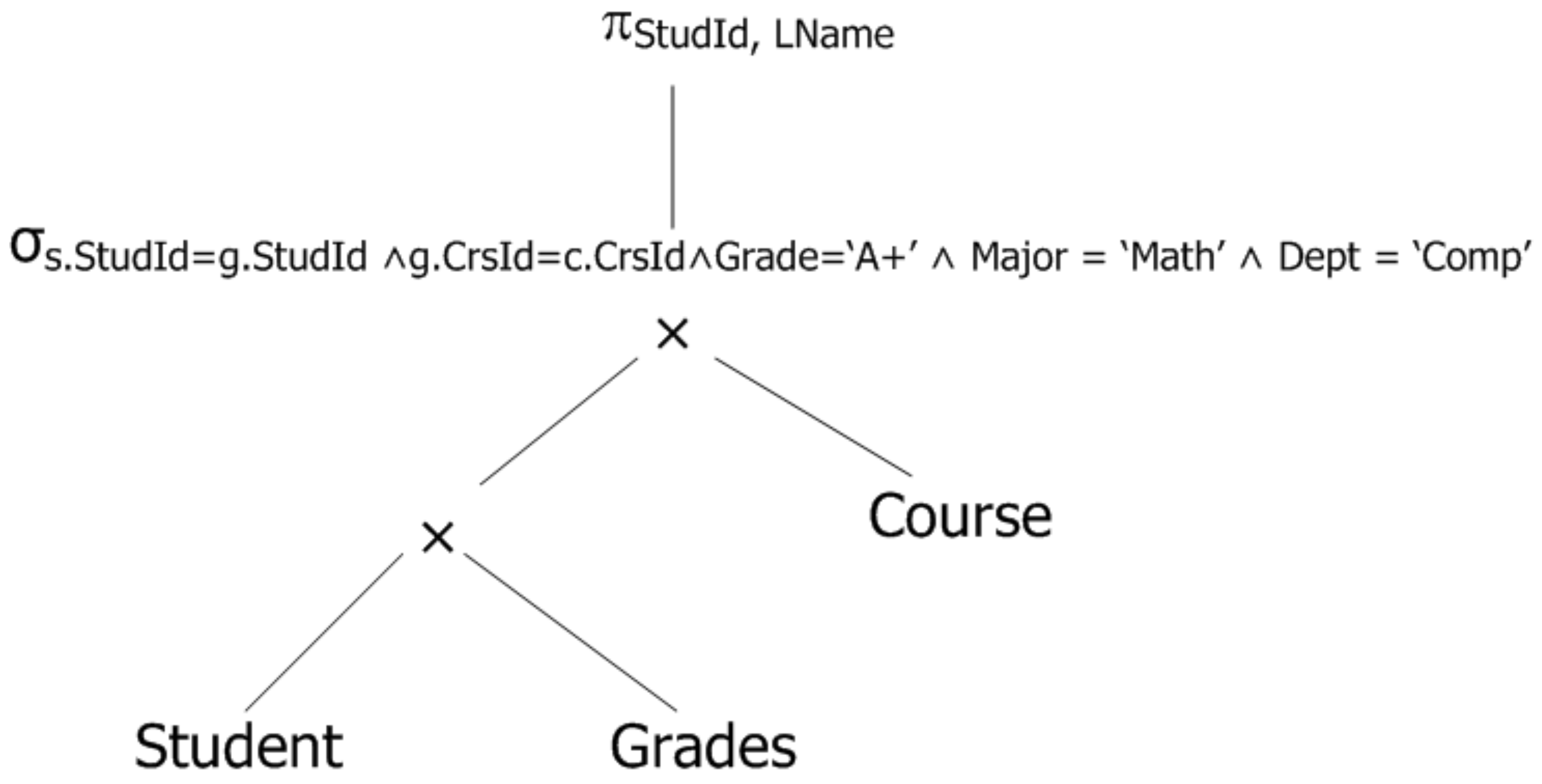
Query Tree

- Query tree: A tree data structure that corresponds to a relational algebra expression
  - It represents the input relations of the query as leaf nodes of the tree, and
  - Represents the relational algebra operations as internal nodes
- The initial query tree for the example SQL query:
  - Lower level nodes, starting from leaves, contain Cartesian product operators
  - These are applied onto relations from the SQL FROM clause
  - After that are (relational) select and join conditions from SQL WHERE clause to upper tree nodes applied
  - Finally is project operator of the SELECT clause attribute list to tree root applied

Query Tree Example

- Relational Algebra:

$\pi_{\text{StudId, LName}} (\sigma_{\text{s.StudId=g.StudId} \wedge \text{g.CrsId=c.CrsId} \wedge \text{Grade='A+'} \wedge \text{Major= 'Math'} \wedge \text{Dept = 'Comp'}} (\text{Course} \times (\text{Student} \times \text{Grades})))$



$\pi_{\text{StudId, LName}}$
$\sigma_{\text{s.StudId=g.StudId} \wedge \text{g.CrsId=c.CrsId} \wedge \text{Grade='A+'} \wedge \text{Major = 'Math'} \wedge \text{Dept = 'Comp'}}$
×
×
Course
Student
Grades

Query Tree Exercise

- Draw query trees for the following relational algebra expressions

$\pi_{\text{StudId}}\ (\sigma_{\text{Grade='A+'}}\ (\text{Grades}))$

$\pi_{\text{StudId, LName}}\ (\sigma_{\text{Grade='A+'}}\ (\text{Student * Grades}))$

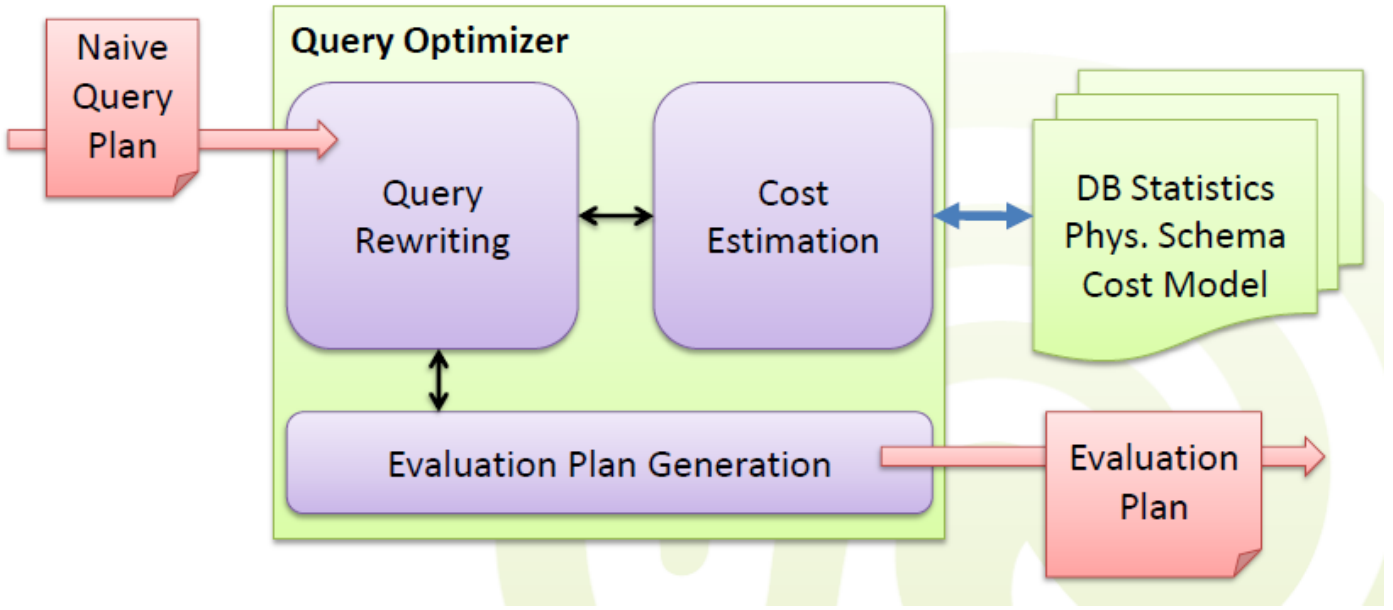
$(\pi_{\text{StudId,}}(\sigma_{\text{CrsId = 'M214'}}\ (\text{Grades}))) - (\pi_{\text{StudId,}}(\sigma_{\text{CrsId = 'C201'}}\ (\text{Grades})))$

Query Execution Plan

- For each query tree, computation proceeds bottom-up:
  - Child nodes must be executed before their parent nodes
  - But there can exist multiple methods of executing sibling nodes, e.g.
    - Process sequentially
    - Process in parallel
- A query execution plan consists of a query tree with additional annotation at each node indicating the implementation method for each RA operator
- The query optimizer determines which query execution plan is optimal, using a variety of algorithms
- Realistically, we cannot expect to always find the best plan but we expect to consistently find a plan that is good (near-optimal)

Query Optimisation

- Query optimizer rewrites the naïve (canonical) query plan into a more efficient evaluation plan



- Choosing a suitable query execution plan is called Query Optimization and it mainly means making decisions
  - On the order of the execution of basic relational algebra operations and
  - On data access methods

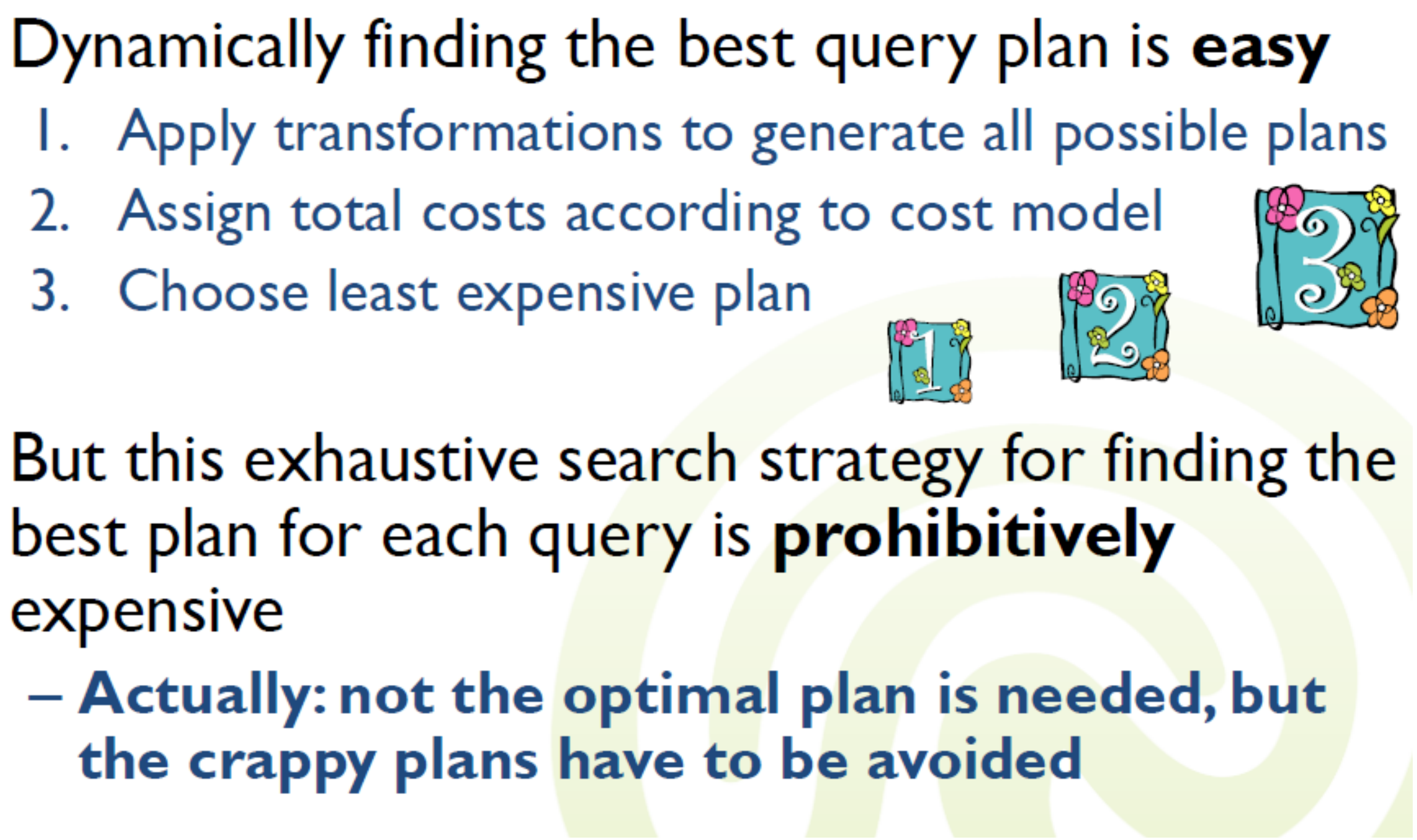
Query Optimization Techniques

- Two main query optimization techniques
  - One relies on the heuristic reordering of the relational algebra operations,
  - The other involves systematic estimating the cost of different execution plans, and choosing one with the lowest cost

- **Heuristics** vs. **cost-based** optimization
  - General heuristics allow to improve performance of most queries
  - Costs estimated from statistics allow for a good optimization of each specific query
  - Most DBMS use a hybrid approach between heuristics and cost estimations

Query Optimization Techniques

- Enumerating potential query plans:



Dynamically finding the best query plan is **easy**

1. Apply transformations to generate all possible plans
2. Assign total costs according to cost model
3. Choose least expensive plan

But this exhaustive search strategy for finding the best plan for each query is **prohibitively** expensive

– **Actually: not the optimal plan is needed, but the crappy plans have to be avoided**

Heuristic Query Optimization

- Process for heuristics optimization
  1. The parser of a high-level query generates an initial internal representation
  2. Apply heuristics rules to optimize the internal representation
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query

- The main heuristic is to apply first the operations that reduce the size of intermediate results
  - E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations

Using Heuristics in Query Optimization (1)

- General Transformation Rules for Relational Algebra Operations:

1. Cascade of σ: A conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations:
   - $\sigma_{c1 \wedge c2 \wedge \ldots \wedge cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$
2. Commutativity of σ: The σ operation is commutative:
   - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
3. Cascade of π: In a cascade (sequence) of π operations, all but the last one can be ignored:
   - $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List1}(R)$
4. Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
   - $\pi_{A1, A2, \ldots, An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, \ldots, An}(R))$

Using Heuristics in Query Optimization (2)

- General Transformation Rules for Relational Algebra Operations (contd.):

5. Commutativity of $\bowtie$ (and x):
   - $R_1 \bowtie_C R_2 = R_2 \bowtie_C R_1$; $R_1 \times R_2 = R_2 \times R_1$

6. Commuting σ with $\bowtie$ (or x): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R_1—the two operations can be commuted as follows:
   - $\sigma_c (R_1 \bowtie R_2) = \sigma_c (R_1) \bowtie R_2$

- Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R_1 and condition c2 involves only the attributes of R_2, the operations commute as follows:
   - $\sigma_c(R_1 \bowtie R_2) = \sigma_{c1}(R_1) \bowtie \sigma_{c2} (R_2)$

Using Heuristics in Query Optimization (3)

- General Transformation Rules for Relational Algebra Operations (contd.):

7. Commuting π with $\bowtie$ (or x): Suppose that the projection list is AL = $\{A_1, ..., A_n, B_1, ..., B_m\}$, where $A_1, ..., A_n$ are attributes of R_1 and $B_1, ..., B_m$ are attributes of R_2. If the join condition c involves only attributes in AL, the two operations can be commuted as follows:

   - $\pi_{AL} (R_1 \bowtie_C R_2) = (\pi_{A1, ..., An} (R_1)) \bowtie_C (\pi_{B1, ..., Bm} (R_2))$

- If the join condition C contains additional attributes not in AL, these must be added to the projection list, and a final π operation is needed.

   - $\pi_{AL}(R_1 \bowtie_C R_2) = \pi_{AL}(\pi_{AL1}(R_1) \bowtie_C \pi_{AL2}(R_2))$,

where AL_i contains the common attributes in Ri and AL and the common attributes of R1 and R2

Using Heuristics in Query Optimization (4)

- General Transformation Rules for Relational Algebra Operations (contd.):

8. Commutativity of set operations: The set operations ∪ and ∩ are commutative but "–" is not.

9. Associativity of ⋈ , x, ∪, and ∩ : These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have
   - $(R_1\ \theta\ R_2)\ \theta\ R_3 = R_1\ \theta\ (R_2\ \theta\ R_3)$

10. Commuting σ with set operations: The σ operation commutes with ∪, ∩, and –. If θ stands for any one of these three operations, we have
   - $\sigma_c\ (R_1\ \theta\ R_2) = (\sigma_c\ (R_1))\ \theta\ (\sigma_c\ (R_2))$

Using Heuristics in Query Optimization (5)

- General Transformation Rules for Relational Algebra Operations (contd.):

11. The π operation commutes with $\cup$.

$$\pi_{AL} (R_1 \cup R_2) = (\pi_{AL} (R_1)) \cup (\pi_{AL} (R_2))$$

12. Converting a (σ, x) sequence into $\bowtie$: If the condition c of a σ that follows a x corresponds to a join condition, convert the (σ, x) sequence into a $\bowtie$ as follows:

$$(\sigma_C (R_1 \times R_2)) = R_1 \bowtie_C R_2$$

Using Heuristics in Query Optimization (6)

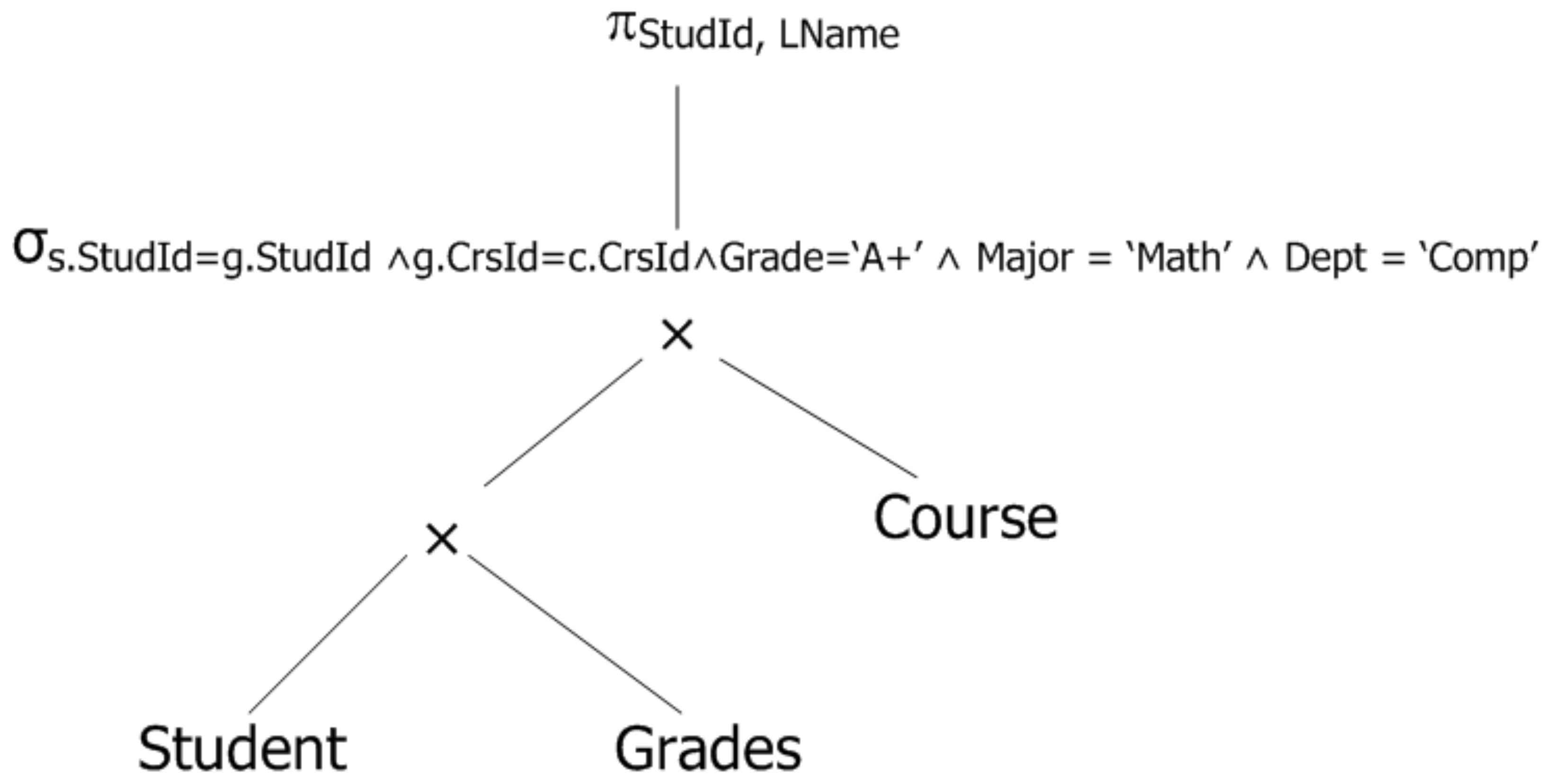
- Summary of Heuristics for Algebraic Optimization:
  1. The main heuristic is to apply first the operations that reduce the size of intermediate results.
  2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible, e.g. rule 6, 7, 10, 11)
  3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)

An Example Initial Query Tree:

- Relational algebra query

$\pi_{\text{StudId, LName}}\ (\sigma_{\text{Grade='A+'} \wedge \text{Major = 'Math'} \wedge \text{Dept = 'Comp'} \wedge \text{s.StudId=g.StudId} \wedge \text{g.CrsId=c.CrsId}}\ (\text{Course} \times (\text{Student} \times \text{Grades})))$

- Query tree



πStudId, LName
σs.StudId=g.StudId ∧g.CrsId=c.CrsId∧Grade='A+' ∧ Major = 'Math' ∧ Dept = 'Comp'
×
×
Course
Student
Grades

Analysis of the Initial Query Tree

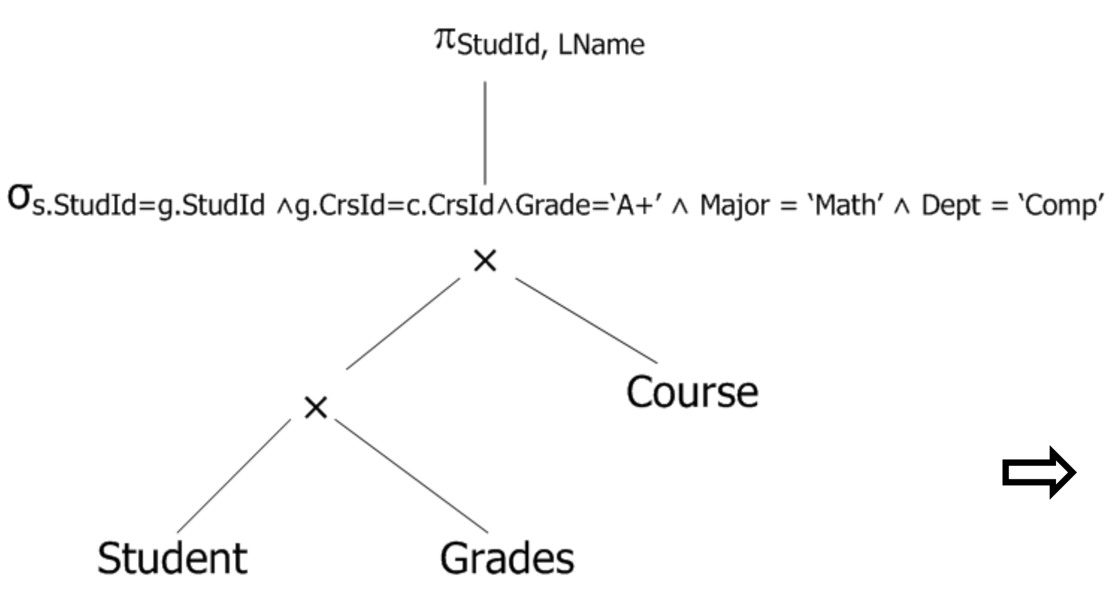
- According to the structure of the initial query tree, two Cartesian products should be executed first

- The main heuristic: Apply SELECT and PROJECT operations before applying the JOIN or other binary operations

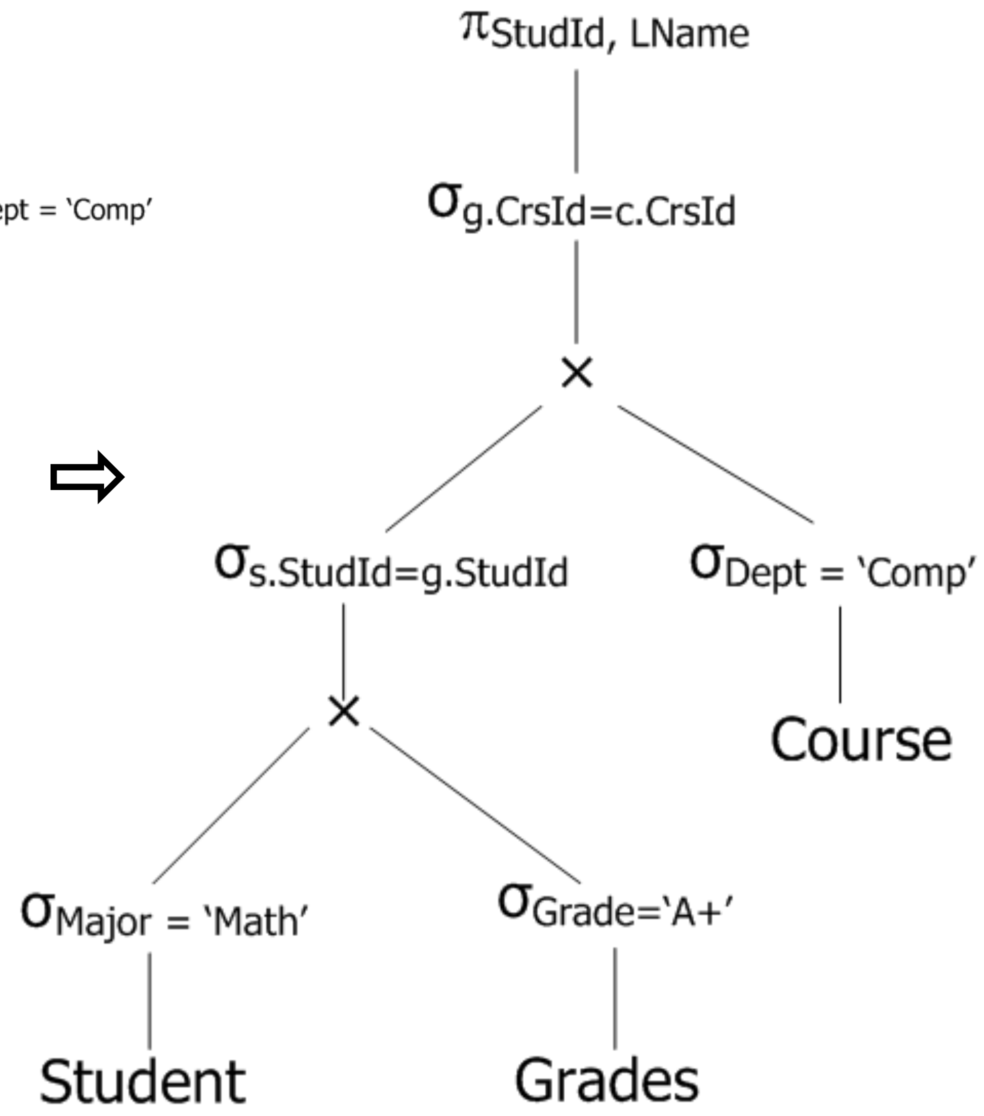
- Hence, move select operations down the tree

Query Tree After Moving Down Selects

- Move select operations down the tree (Rule 6)

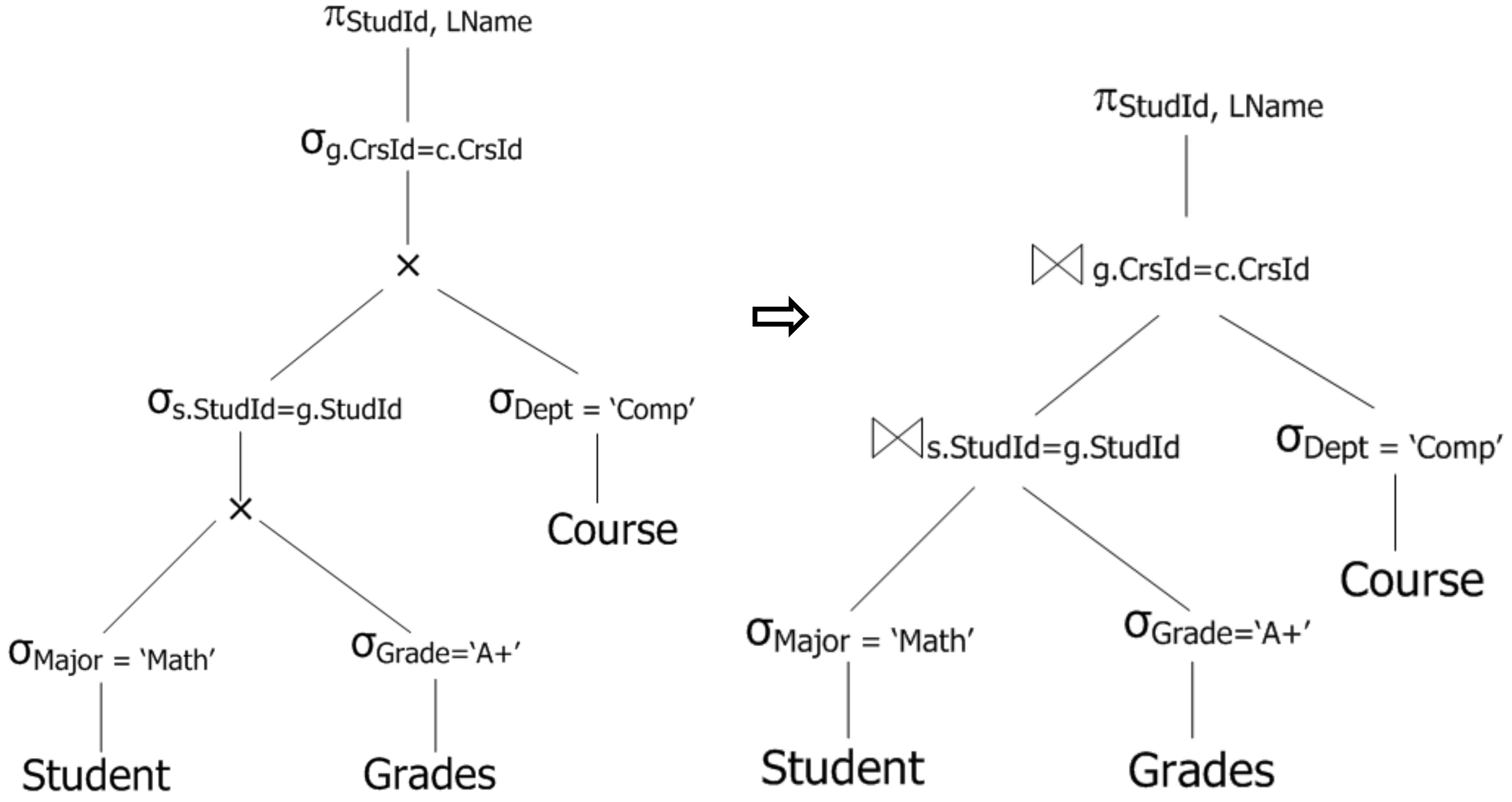

$\pi_{\text{StudId, LName}}$
$\sigma_{\text{s.StudId=g.StudId} \wedge \text{g.CrsId=c.CrsId} \wedge \text{Grade='A+'} \wedge \text{Major = 'Math'} \wedge \text{Dept = 'Comp'}}$
×
×
Course
Student
Grades

⇨


$\pi_{\text{StudId, LName}}$
$\sigma_{\text{g.CrsId=c.CrsId}}$
×
$\sigma_{\text{s.StudId=g.StudId}}$
$\sigma_{\text{Dept = 'Comp'}}$
×
Course
$\sigma_{\text{Major = 'Math'}}$
$\sigma_{\text{Grade='A+'}}$
Student
Grades

Query Tree After Introducing Joins

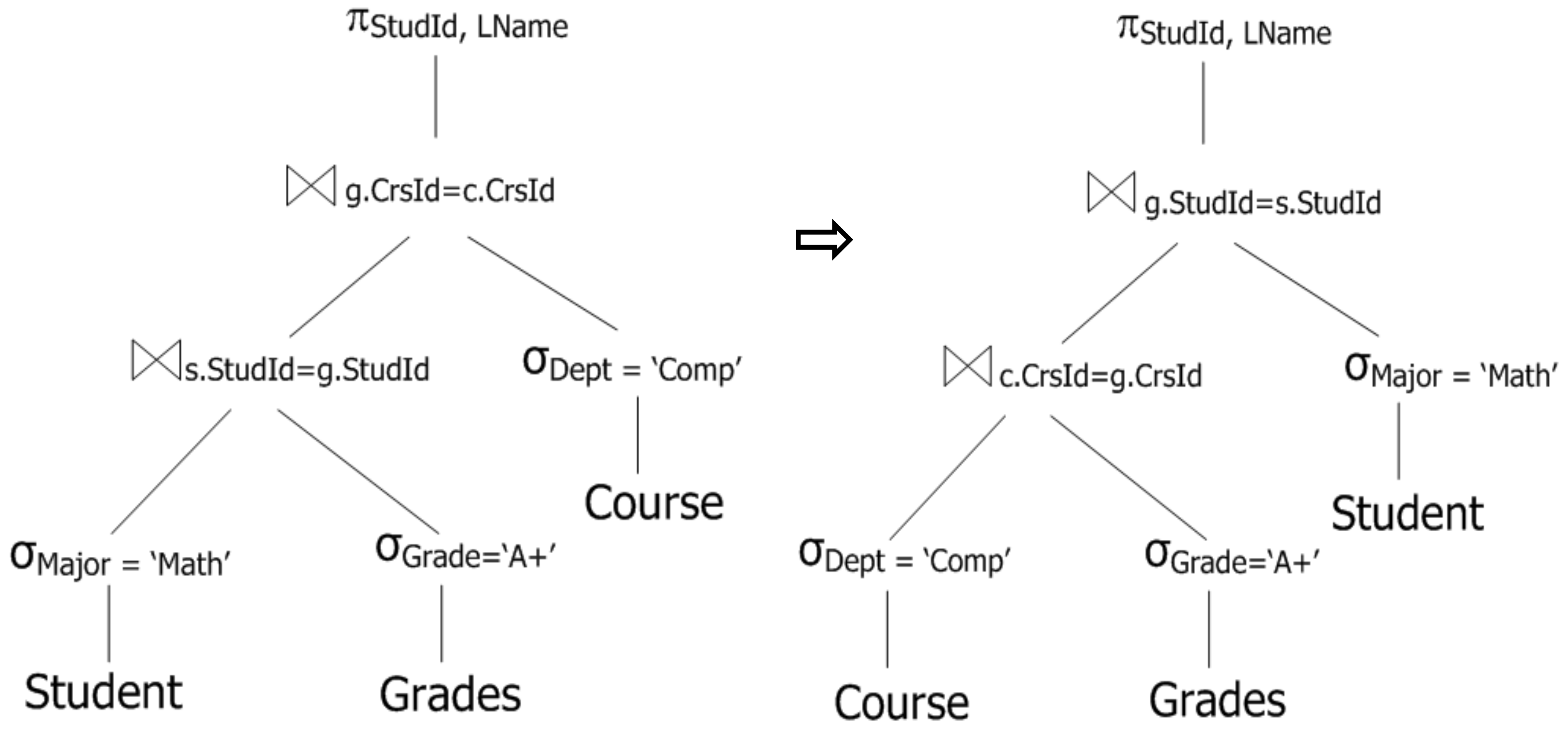
- Replacing each Cartesian product followed by a select according to a join condition with a join operator (Rule 12)



$\pi_{\text{StudId, LName}}$
$\sigma_{\text{g.CrsId=c.CrsId}}$
×
$\sigma_{\text{s.StudId=g.StudId}}$
$\sigma_{\text{Dept = 'Comp'}}$
×
Course
$\sigma_{\text{Major = 'Math'}}$
$\sigma_{\text{Grade='A+'}}$
Student
Grades
⇨
$\pi_{\text{StudId, LName}}$
$\bowtie_{\text{g.CrsId=c.CrsId}}$
$\bowtie_{\text{s.StudId=g.StudId}}$
$\sigma_{\text{Dept = 'Comp'}}$
Course
$\sigma_{\text{Major = 'Math'}}$
$\sigma_{\text{Grade='A+'}}$
Student
Grades

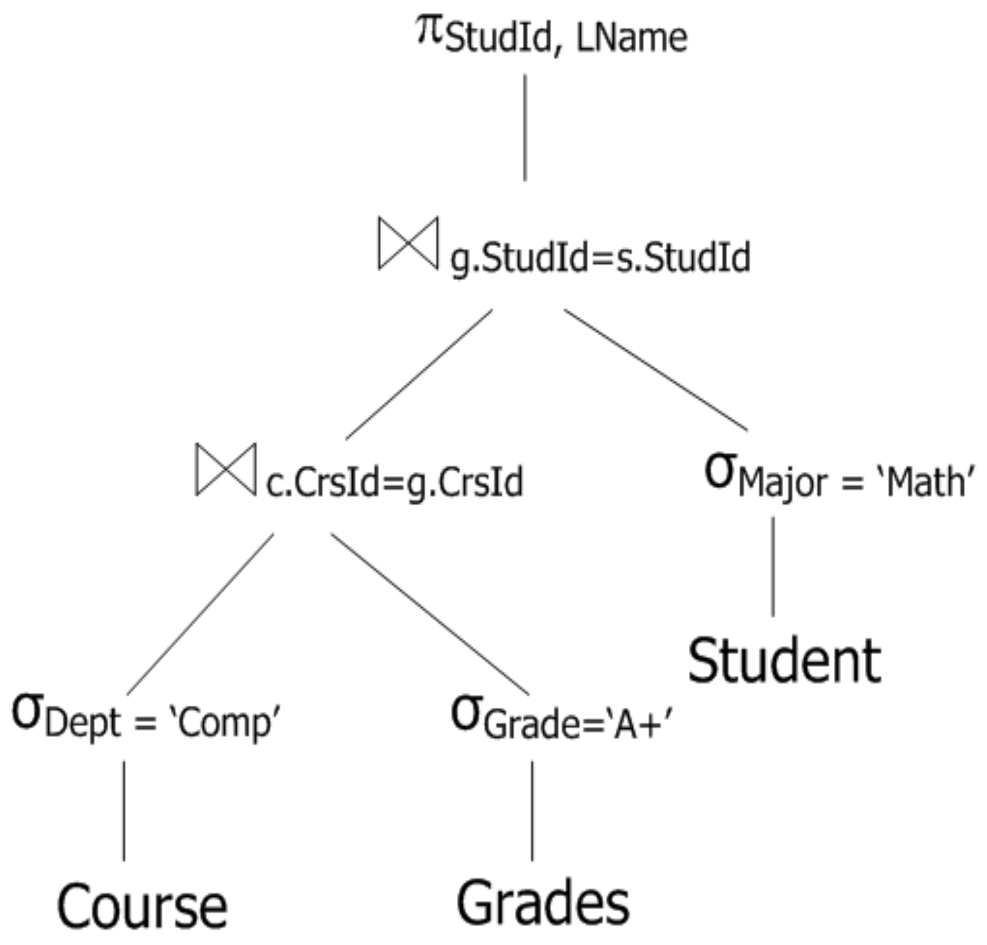
Query Tree After Switching Join Nodes

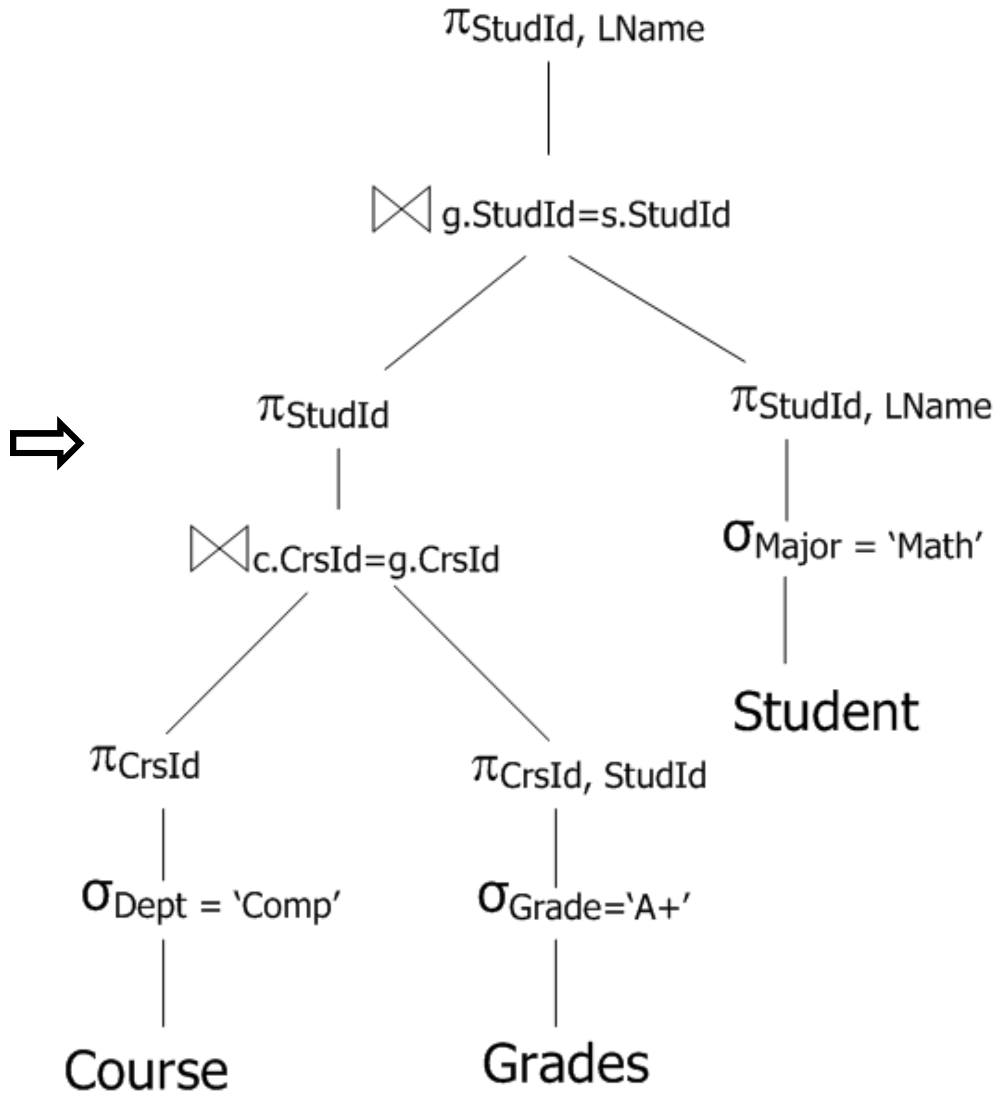
- Assume there are less Comp courses than Math major students
- switching Course and Student so that the very restrictive select operation could be applied as early as possible (Rule 9)



Query Tree after Pushing down Project

- keeping in intermediate relations only the attributes needed by subsequent operations by applying project (π) operations as early as possible (Rule 7)





Effect of Project Operations

- Performing project operations as early as possible brings an improvement in the efficiency of a query execution
  - tuples get shorter after projection, more of them will fit into a file block of the same size
  - the same number of tuples will be contained in a smaller number of blocks
  - As there will be less blocks to be processed by subsequent operations, the query execution will be faster

Query Optimisation

- All transformations can be applied to the canonical evaluation plan
  - However, there is no best operator sequence that is always optimal
  - Efficiency depends on the current data instance, the actual implementation of base operations, access paths and indexes, etc.
- Idea: assign average costs to operators (nodes) and aggregate costs for each query plan

Summary

- Heuristic optimization converts a declarative query to a canonical algebraic query tree that is then gradually transformed using transformation rules
- The main heuristics is to perform unary relational operations (selection and projection) before binary operations (joins, set theoretic), and aggregate functions with (or without) grouping

Next

- Cost-based query optimization
- Readings:
    - Chapter 19: Algorithms for Query Processing and Optimization
    - Chapters 17: Disk Storage, Basic File Structures, and Hashing (Sections: 17.2, to 17.8)
    - Chapter 18: Indexing Structures for Files (Sections: 17.1 to 17.5)
- Supposed knowledge: File Organization – COMP261