CONSTRUCTOR UNIVERSITY

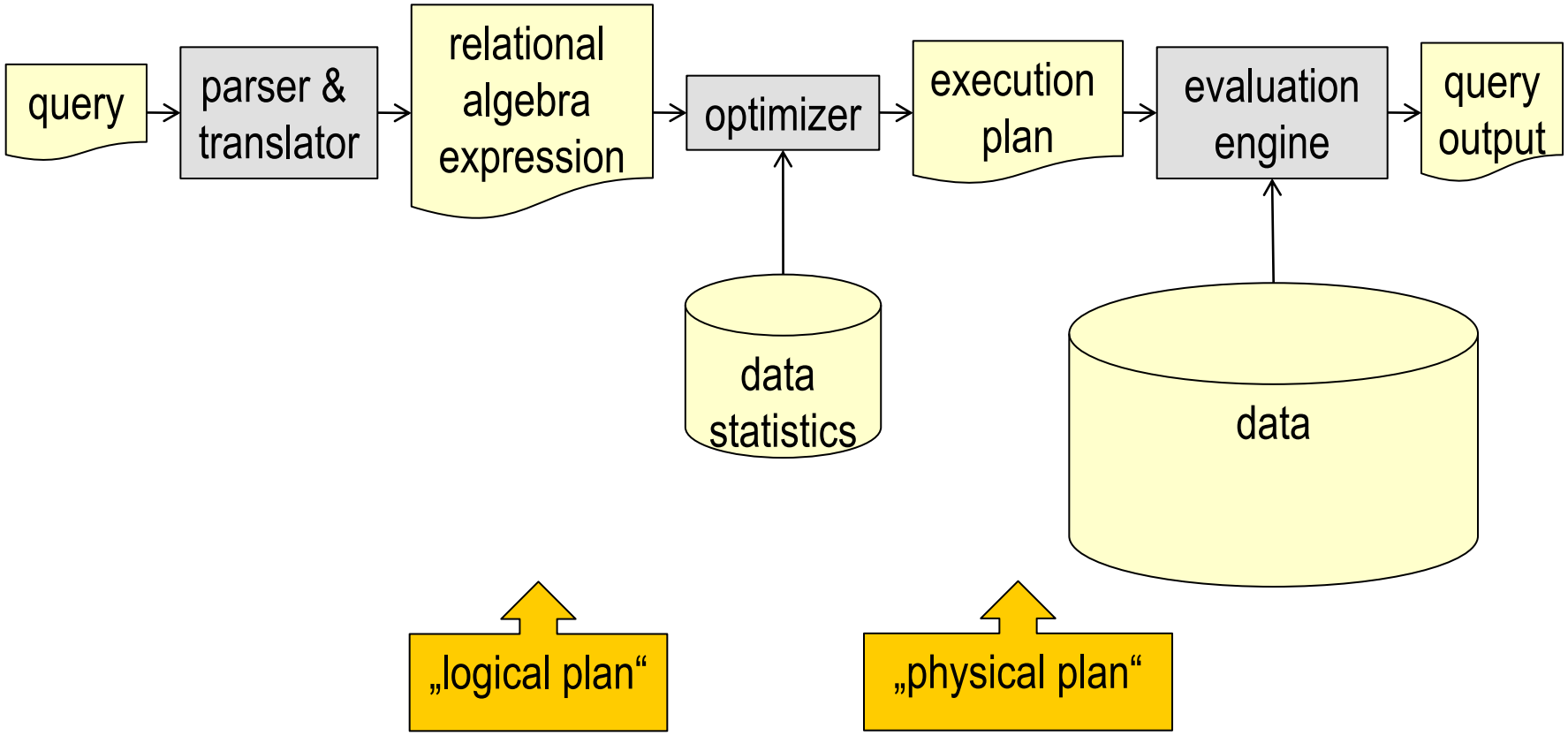
Query Processing and Optimization

Jennifer Widom

Ramakrishnan/Gehrke Chapters 10, 12

Steps in Database Query Processing

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution



Running Example

- Tables (what are the keys?):

```
Student(ID, Name, Major)
Course(Num, Dept)
Taking(ID, Num)
```

- Query to find all EE students taking at least one CS course:

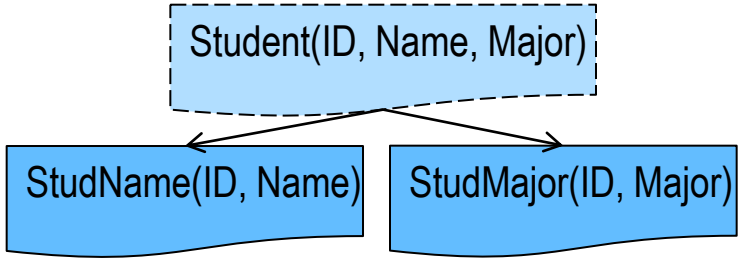
```
SELECT  Name                          π
FROM    Student, Course, Taking       ×
WHERE   Taking.ID = Student.ID        ▷◁
    AND Taking.Num = Course.Num       ▷◁
    AND Major = 'EE'                  σ
    AND Dept = 'CS'                   σ
```

View Expander

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

- Suppose Student is view:

```
CREATE VIEW Student AS
SELECT StudName.ID, Name, Major
FROM   StudName, StudMajor
WHERE  StudName.ID = StudMajor.ID
```



- Via view expander original query becomes:

```
SELECT Name
FROM   Course, Taking, Student AS ( SELECT StudName.ID, Name, Major
FROM StudName, StudMajor WHERE  StudName.ID = StudMajor.ID )
WHERE  Taking.ID = Student.ID AND Taking.Num = Course.Num AND
          Student.Major = 'EE' AND Course.Dept = 'CS' AND StudName.ID = StudMajor.ID
```

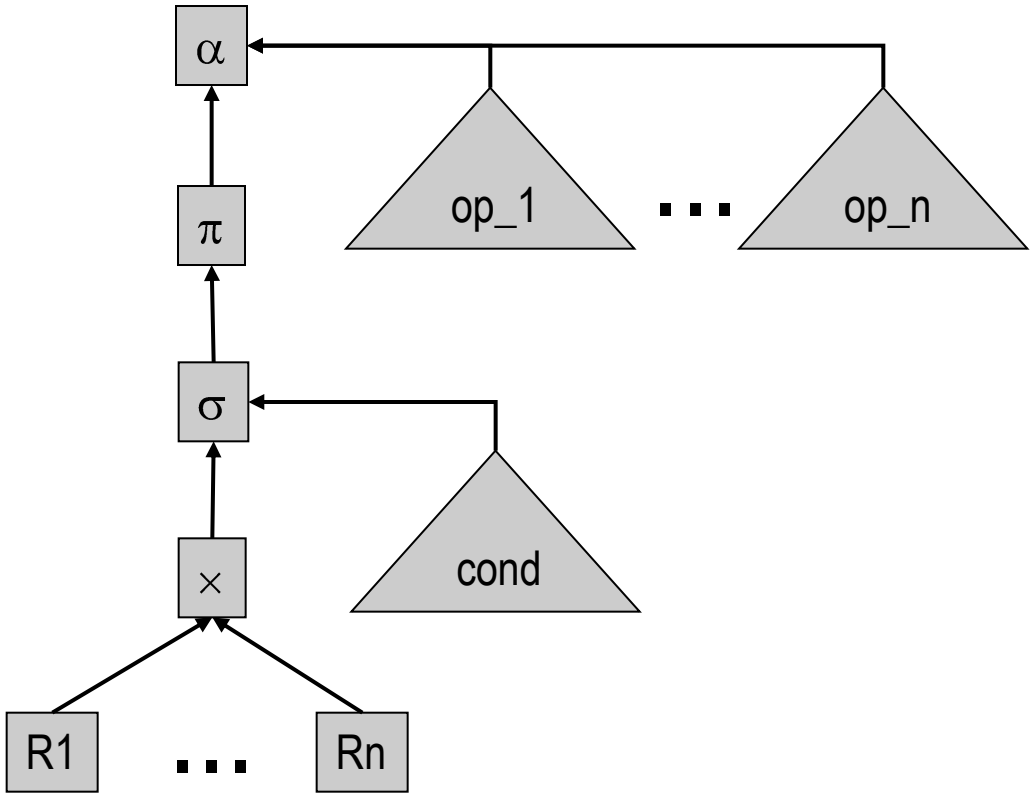
```
SELECT Name
FROM    Student, Course, Taking
WHERE  Taking.ID = Student.ID
   AND  Taking.Num = Course.Num
   AND  Major = 'EE'
   AND  Dept = 'CS'
```

- "flattened":

```
SELECT Name
FROM   Course, Taking, StudName, StudMajor
WHERE  Taking.ID = StudName.ID AND Taking.Num = Course.Num AND
StudMajor.Major = 'EE' AND Course.Dept = 'CS' AND StudName.ID = StudMajor.ID
```

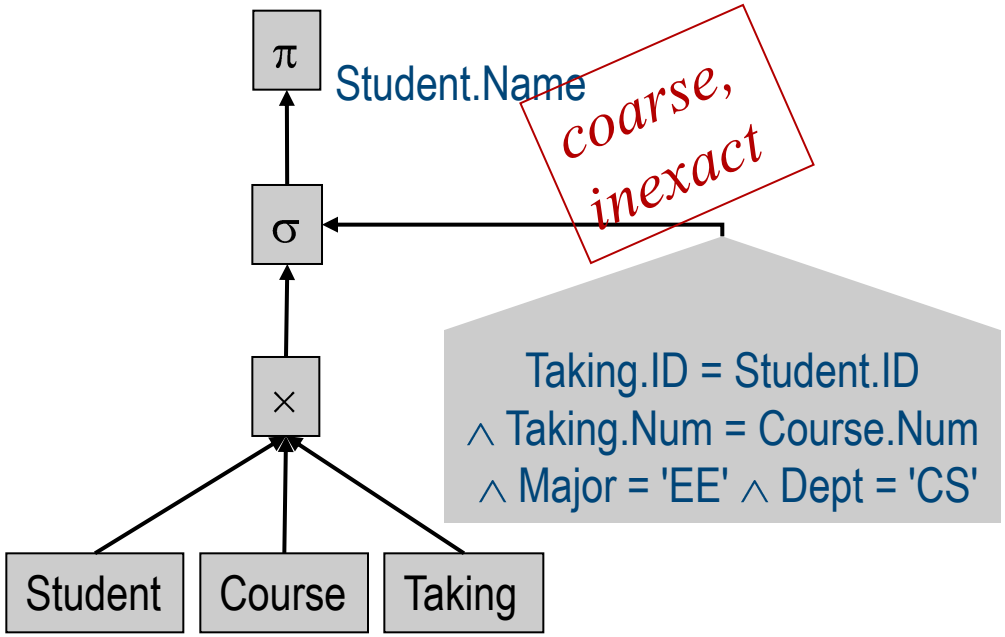
Logical Query Tree: Notation Overview

- Logical query tree
  = Logical plan = parsed query, translated into relational algebra
- Equivalent to relational algebra expression (why not calculus?) using:
  - $\times$ cross product
  - σ selection from set, based on condition *cond*
  - π projection to attributes
  - α application of an expression to arguments
  - $\triangleright\triangleleft$ joins...

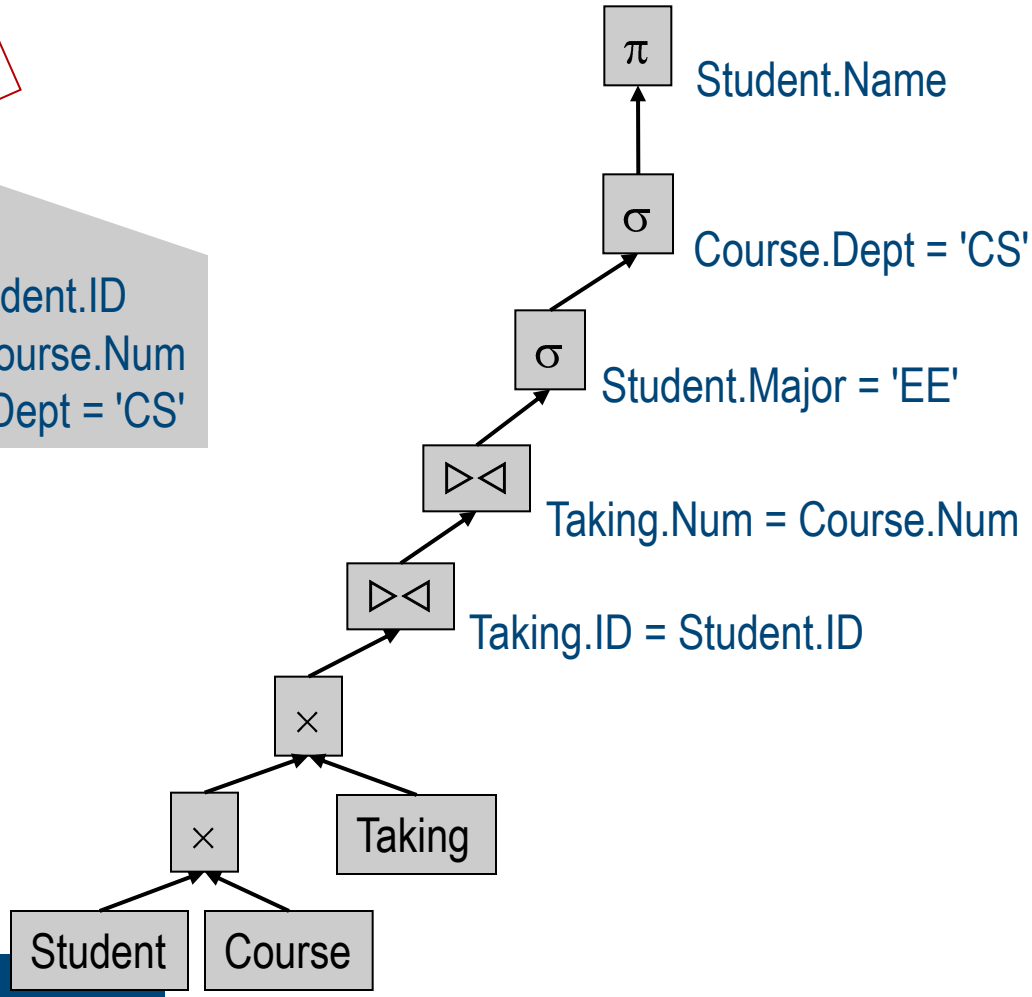


SELECT α(op_1(R1,R2,…)),op_2(R1,R2,…), …)
FROM R1, R2, …
WHERE σ(R1,R2,…)

Logical Query Plan




```
SELECT  Name
FROM    Student, Course, Taking
WHERE   Taking.ID = Student.ID
  AND   Taking.Num = Course.Num
  AND   Major = 'EE'
  AND   Dept = 'CS'
```



Logical vs Physical Query Plan

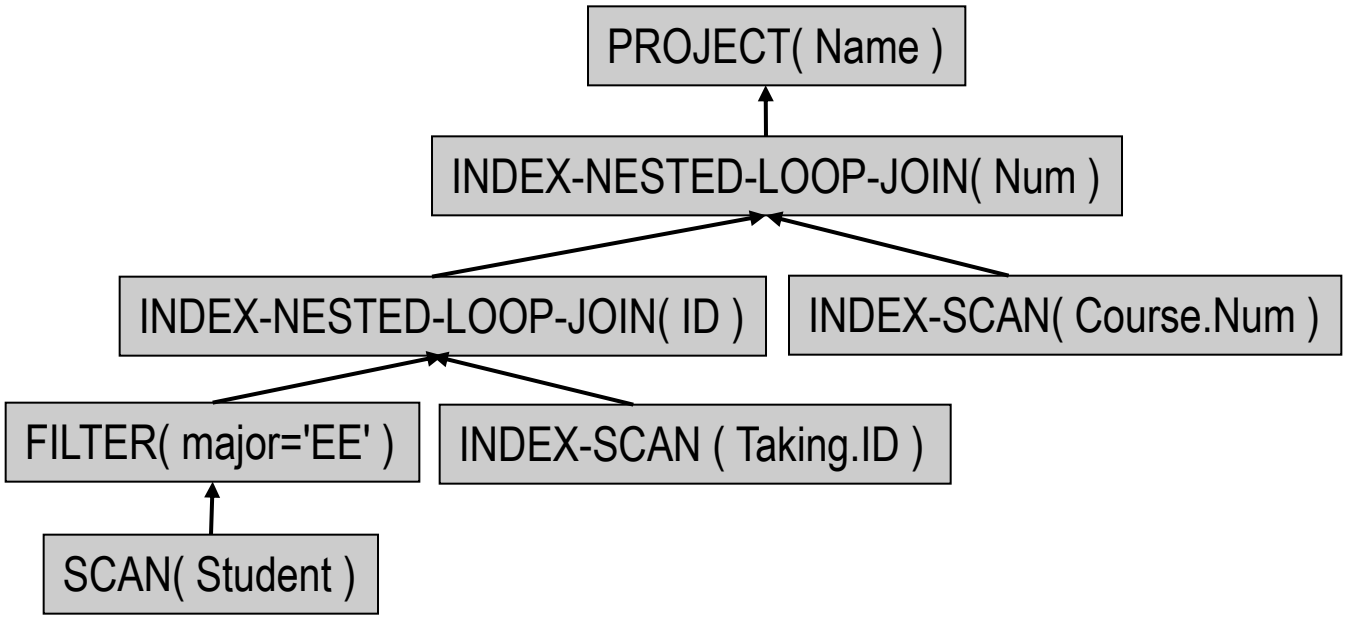
Parser – Checker - Views - Logical plan - Rewriter - Physical plan - Code gen. - Execution

- Commonalities:
  - Trees representing query evaluation
  - Leaves = data (table vs table/index)
  - Internal nodes = "operators" over data
- Differences:

	Level	**Operators**
Logical plan	higher-level, algebraic	query language constructs
Physical plan	lower-level, operational	"access methods"

Physical Query Plan

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution




```
SELECT  Name
FROM    Student, Course, Taking
WHERE   Taking.ID = Student.ID
   AND  Taking.Num = Course.Num
   AND  Major = 'EE'
   AND  Dept = 'CS'
```

one of manyManyMany possible plans, assumes particular index situation.

Sample Operator: Nested Loop Join

- Consider this equi-join query:

```
SELECT  *
FROM    Sailor S, Reserves R
WHERE   S.sid = R.sid
```

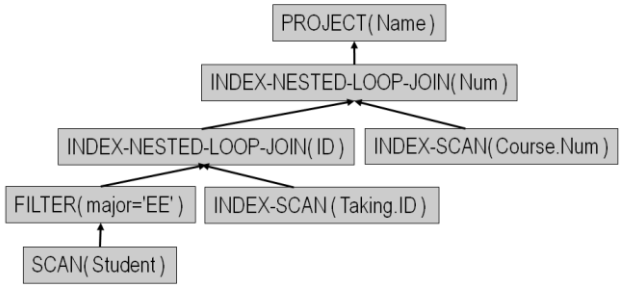
- Naïve, straightforward approach: combine all tuples, pick good ones

```
foreach tuple r in R do
    foreach tuple s in S do
        if r_i == s_j then add <r,s> to result
```

  - Assume there is no index, R small, S big: better R inner or S?
  - What if hash index on S?
  - *…this is what cost-based optimization considers!*

Physical Plan Generation

- ManyManyMany possible physical query plans for a given logical plan
- physical plan generator tries to select "optimal" one
  - Optimal wrt. response time, throughput
- How are intermediate results passed from children to parents?
  - Temporary files
    - *Evaluate tree bottom-up*
    - *Children write intermediate results to temporary files*
    - *Parents read temporary files*
  - Iterator interface (next)



Sample Query Plan

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

```
SET EXPLAIN ON AVOID_EXECUTE;
SELECT   C.customer_num, O.order_num
FROM     customer C, orders O, items I
WHERE    C.customer_num = O.customer_num
         AND O.order_num = I.order_num
```

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept the row and send to user
            end if
         end for
      end if
   end for
end for
```

Iterator Interface

- "ONC protocol":
  Every operator maintains own execution state, implements the following methods:
  - open( ):
    Initialize state, get ready for processing
  - getNext( ):
    Return next tuple in result (or null if no more tuples); adjust state for delivering subsequent tuples
  - close( ):
    Clean up

Ex: Iterator for Table Scan

Sailors: 22|Dustin|7|45.0|31|Lubber|8|55.5|58|Rusty|10|35.0…

- open( )
  - Allocate buffer space
- getNext( )
  - If no block of R has been read yet:
    read first block from disk
    return (R==empty ? null : first tuple in block)
  - If no more tuple left in current block:
    read next block of R from disk
    return (R exhausted ? null : first tuple in block)
  - Return next tuple in block
- close( )
  - Deallocate buffer space

Ex: Iterator for Nested-Loop Join

- open()
  - R.open(); S.open();
  - r = R.getNext();
- getNext()
  - repeat until r and s join:

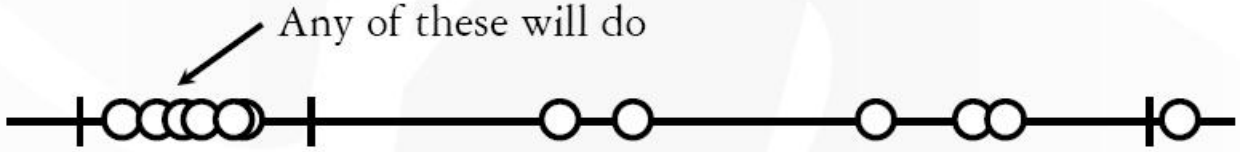
```
s = S.getNext();
if (s = = null)
{       S.close(); S.open(); s = S.getNext();
        if (s = = null) return null;
        r = R.getNext();
        if (r = = null) return null;
}
```

  - return <r,s>;
- close()
  - R.close(); S.close();


Query
Optimization

Query Optimization

- Optimization = find better, equivalent plan
  - Equivalent = produces same result
  - Logical level optimization = aka heuristic optimization
  - Physical level optimization = aka cost-based optimization
- Two main issues:
  - For a given query, how to find cheapest plans?
  - How is cost of a plan estimated?



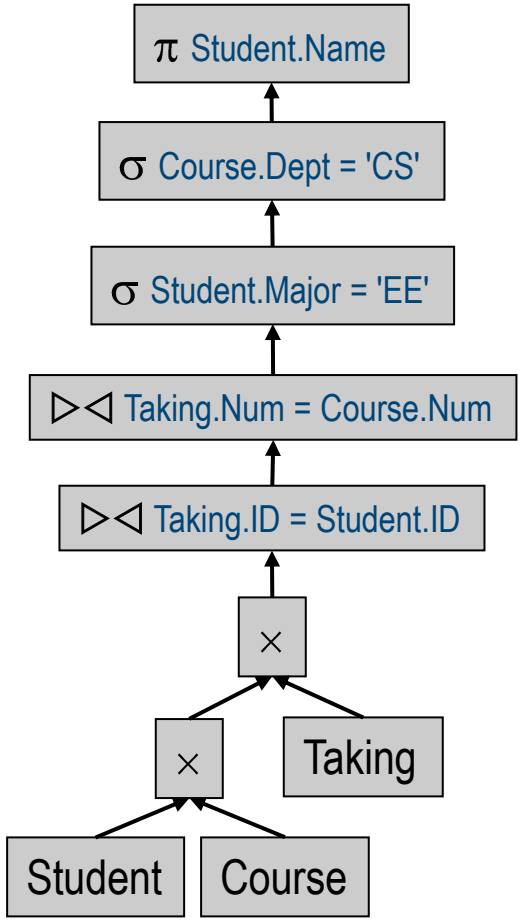
(I) Heuristic Optimization

- logical tree → (more efficient) logical tree
  - heuristically apply algebraic equivalences
    - *heuristics = "looks good, let's try it!"*
- Ex: "push down predicates"

$$\sigma_{major='EE'}(\triangleright\!\triangleleft_{Taking.ID=Student.ID}(Taking,Student))$$

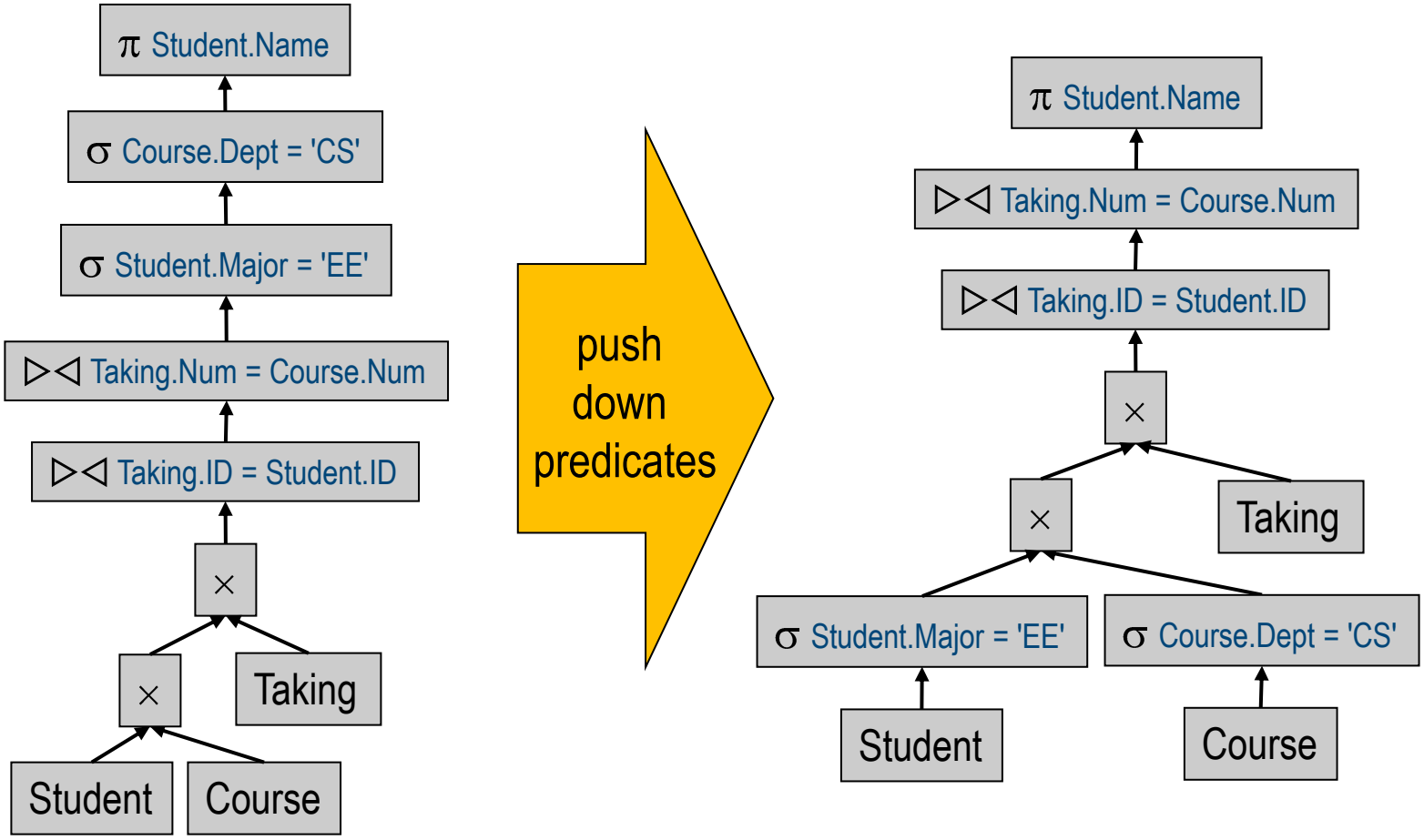
$$\equiv$$

$$\triangleright\!\triangleleft_{Taking.ID=Student.ID}(\sigma_{major='EE'}(Taking),Student)$$



(I) Heuristic Optimization

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution



Relational Algebra Equivalences

R, S, T relational expressions

- Allow to choose different join orders
  and to `push' selections and projections ahead of (="below") joins

- Selections: $\sigma_{c1\wedge\ldots\wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R))$ (Cascade)

  $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (Commute)

- Projections: $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ (Cascade)

- Joins: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (Associative)

  $(R \bowtie S) \equiv (S \bowtie R)$ (Commute)

Show that: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

More Equivalences

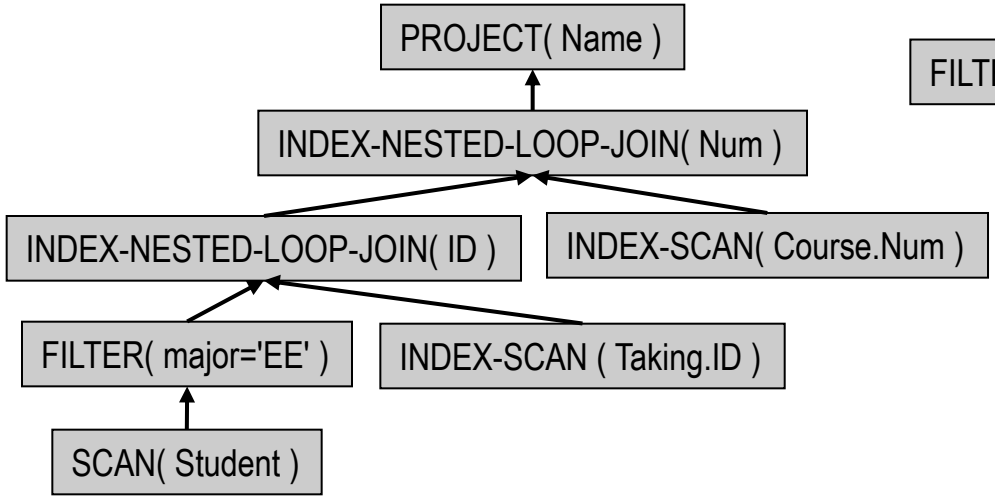
- A projection commutes with a selection that only uses attributes retained by projection
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join

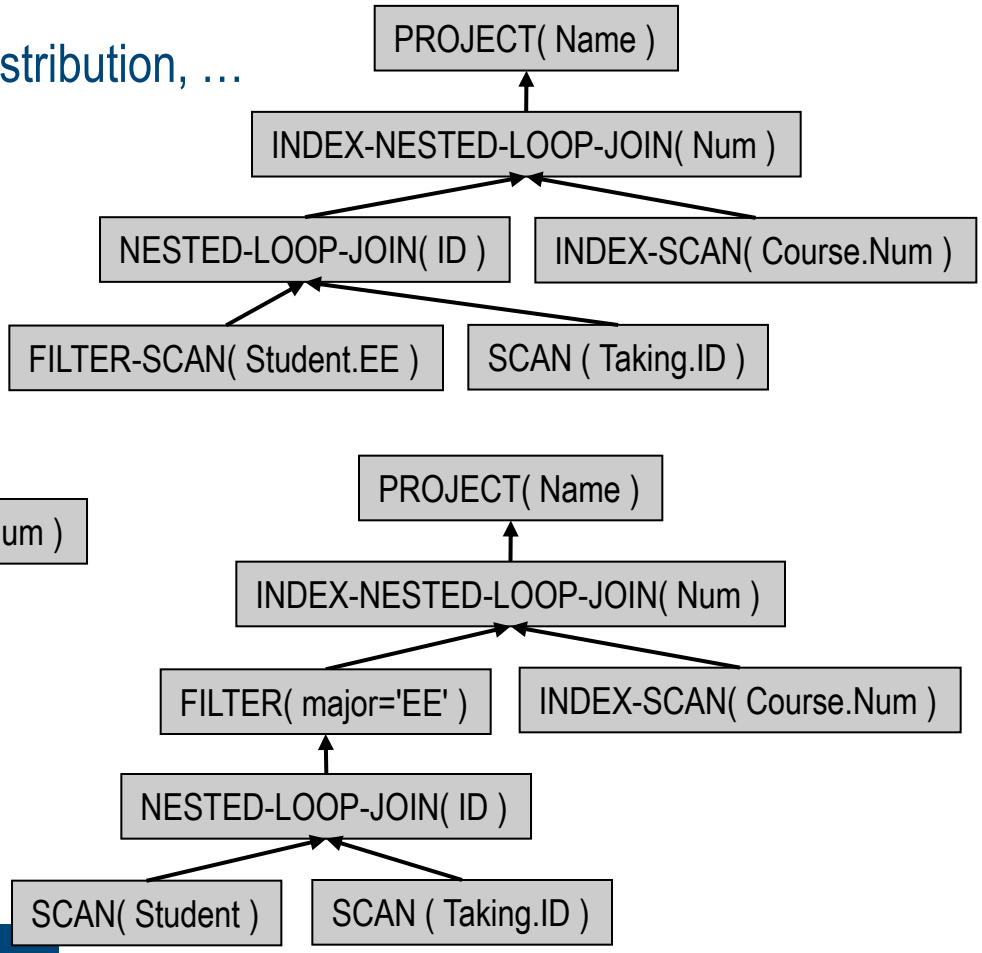
$$\sigma_{p(R.a,S.b)}(R \times S) \equiv \sigma_{R.a,R.b}(R \bowtie S)$$

- A selection on just attributes of R commutes with R $\bowtie$ S
  - $\sigma_{R.x\,\theta\,C}(R \bowtie S) \equiv \sigma_{R.x\,\theta\,C}(R) \bowtie S$
- Similarly, a projection following a join R $\bowtie$ S can be `pushed'
  - by retaining only attributes of R (and S) needed for the join or kept by projection

(II) Cost-Based Optimization

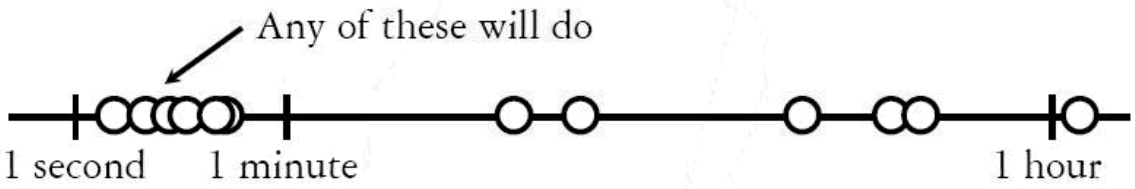
- Estimate costs, based on physical situation
  - concrete table sizes, indexes, data distribution, …
  - Find cheapest plan





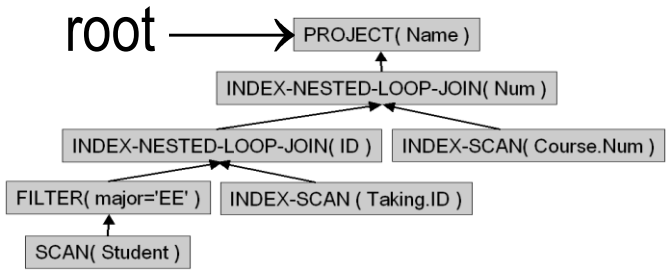
(II) Cost-Based Optimization

- Approach:
  - enumerate all (?) possible physical plans that can be derived from given logical plan
  - estimate cost for each plan
  - pick best (i.e., least cost) alternative
- Ideally: Want to find best plan; practically: Avoid worst plans!



Finale: Execution of Tree

Parser – Checker - Views - Logical plan - Rewriter - Physical plan - Optim. - Execution




```
result = {};
root.open();
do
{
        tmp = root.getNext();
        result += tmp;
} while (tmp != NULL);
root.close();
return result;
```

- Recursive evaluation of tree
  - Requests go down
  - Intermediate result tuples go up
- Often instead: compile into "database machine code" program
  - CPU, GPU, FPGA, ...

Summary

- Query tree = internal representation of query
  - Logical tree: based on relational algebra
  - Physical tree: concrete algorithms („access plans“)
- Optimization = modify tree to perform better
  - Logical optimization = heuristic optimization = query rewriting
  - Physical optimization = cost-based optimization = black magic