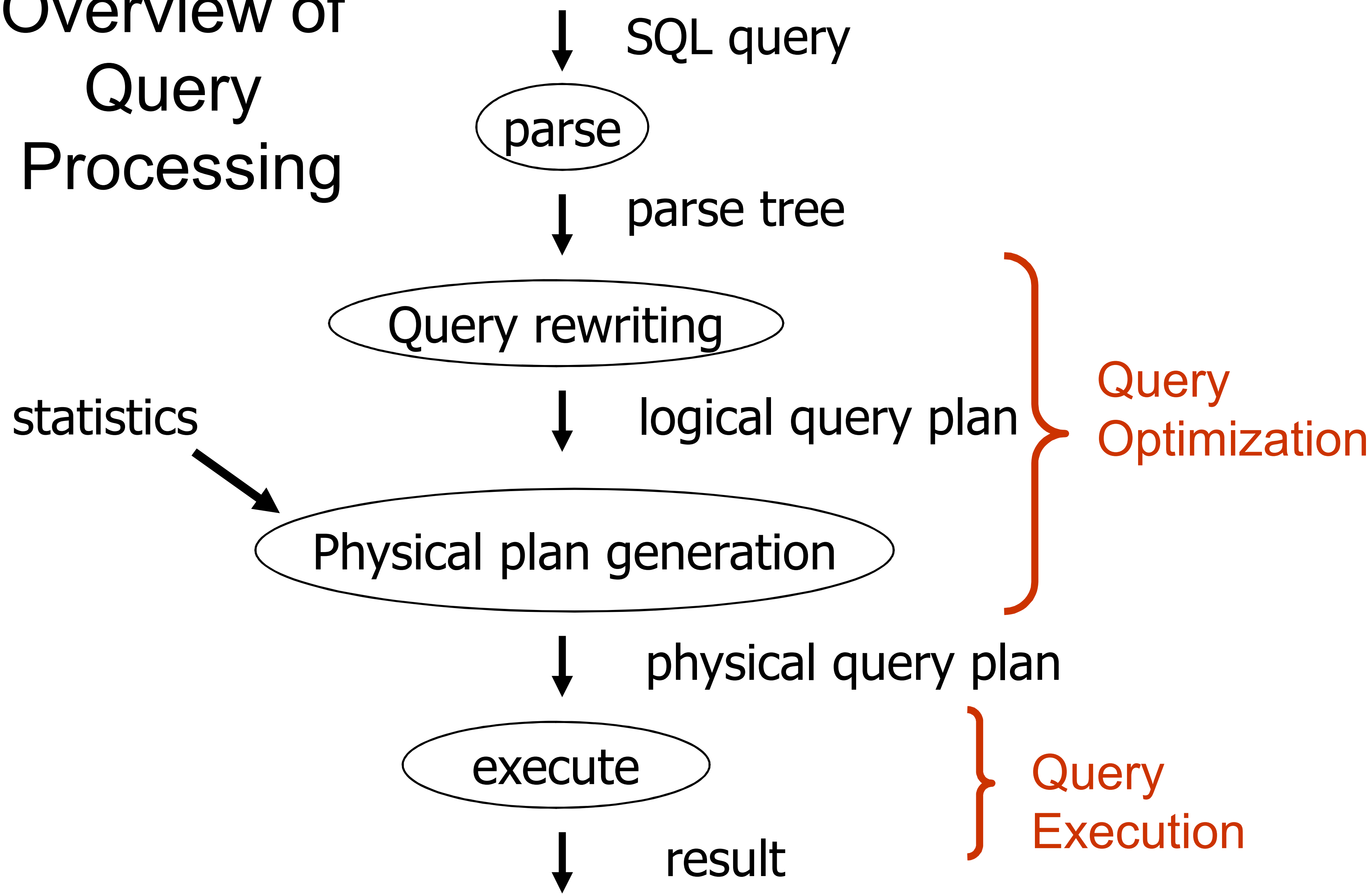
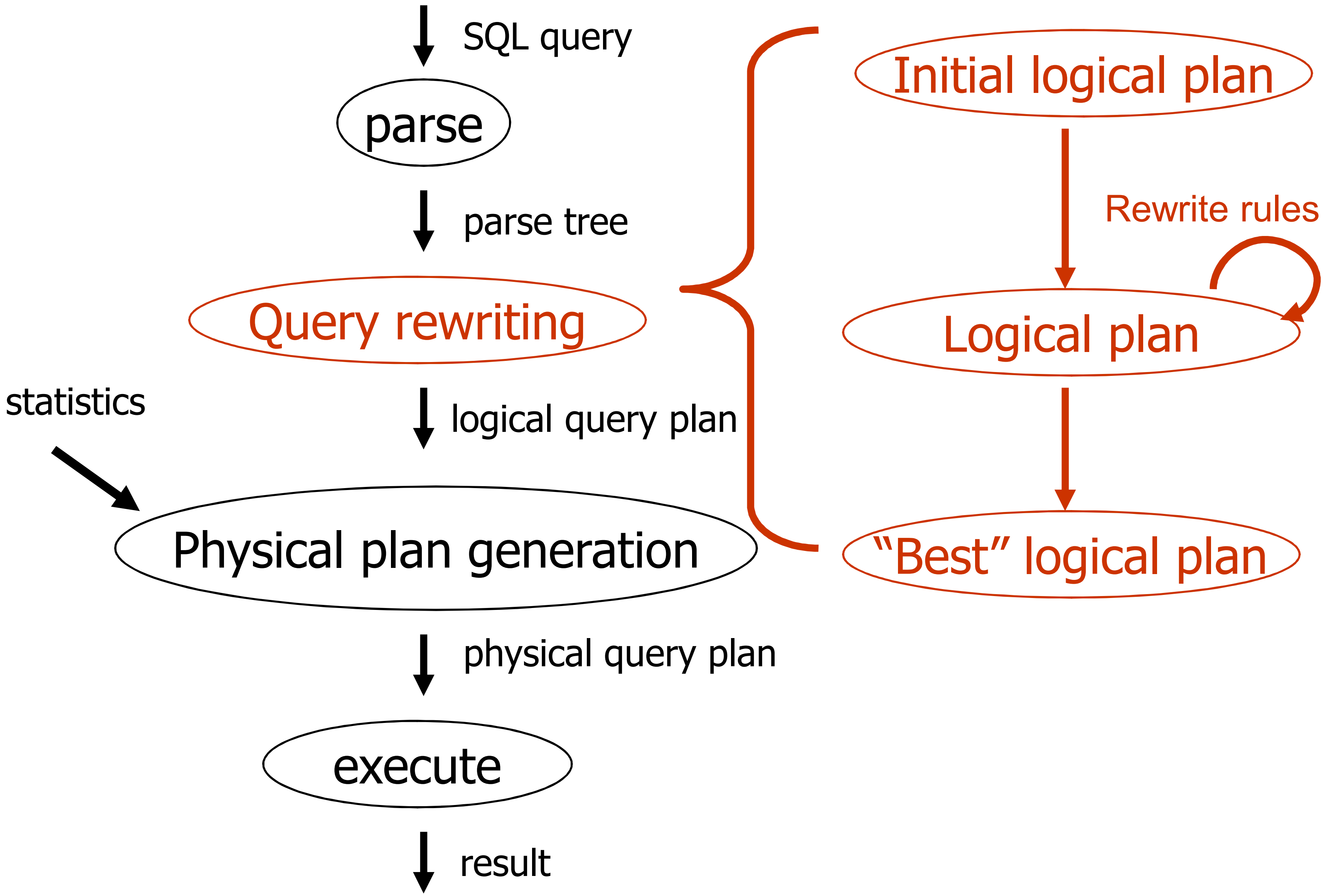
CPS216: Advanced Database Systems

Notes 03:Query Processing (Overview, contd.)

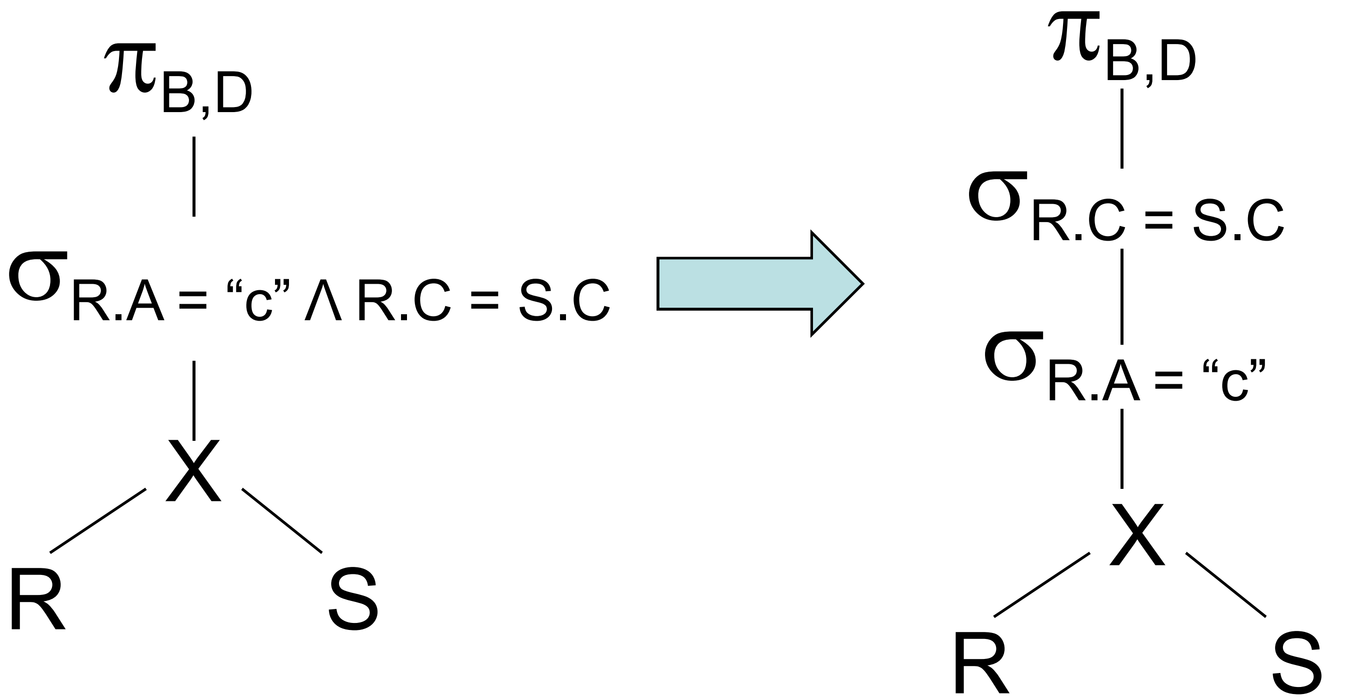
Shivnath Babu

Overview of Query Processing

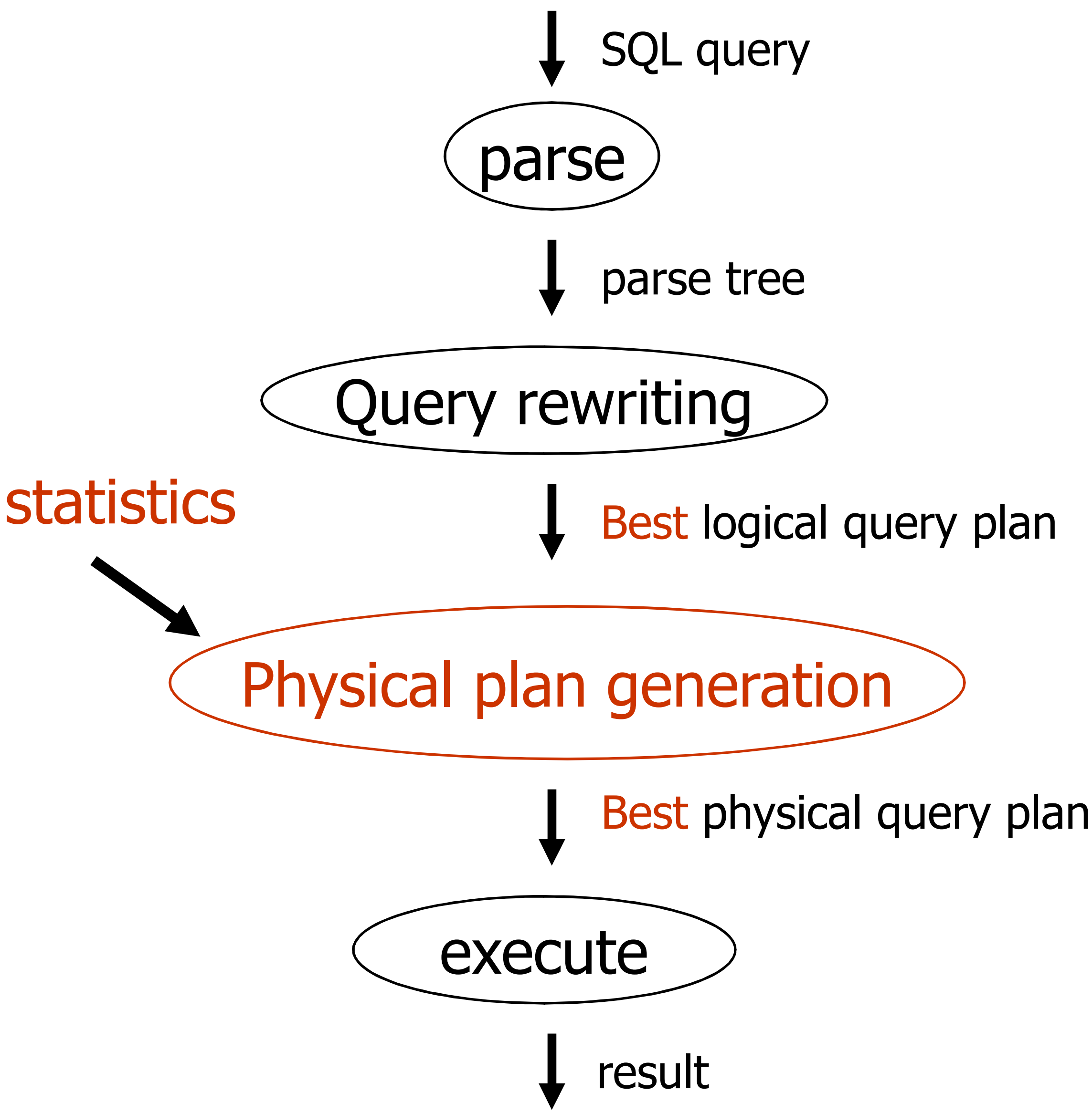



SQL query
parse
parse tree
Query rewriting
statistics
logical query plan
Physical plan generation
physical query plan
execute
result
Initial logical plan
Rewrite rules
Logical plan
"Best" logical plan

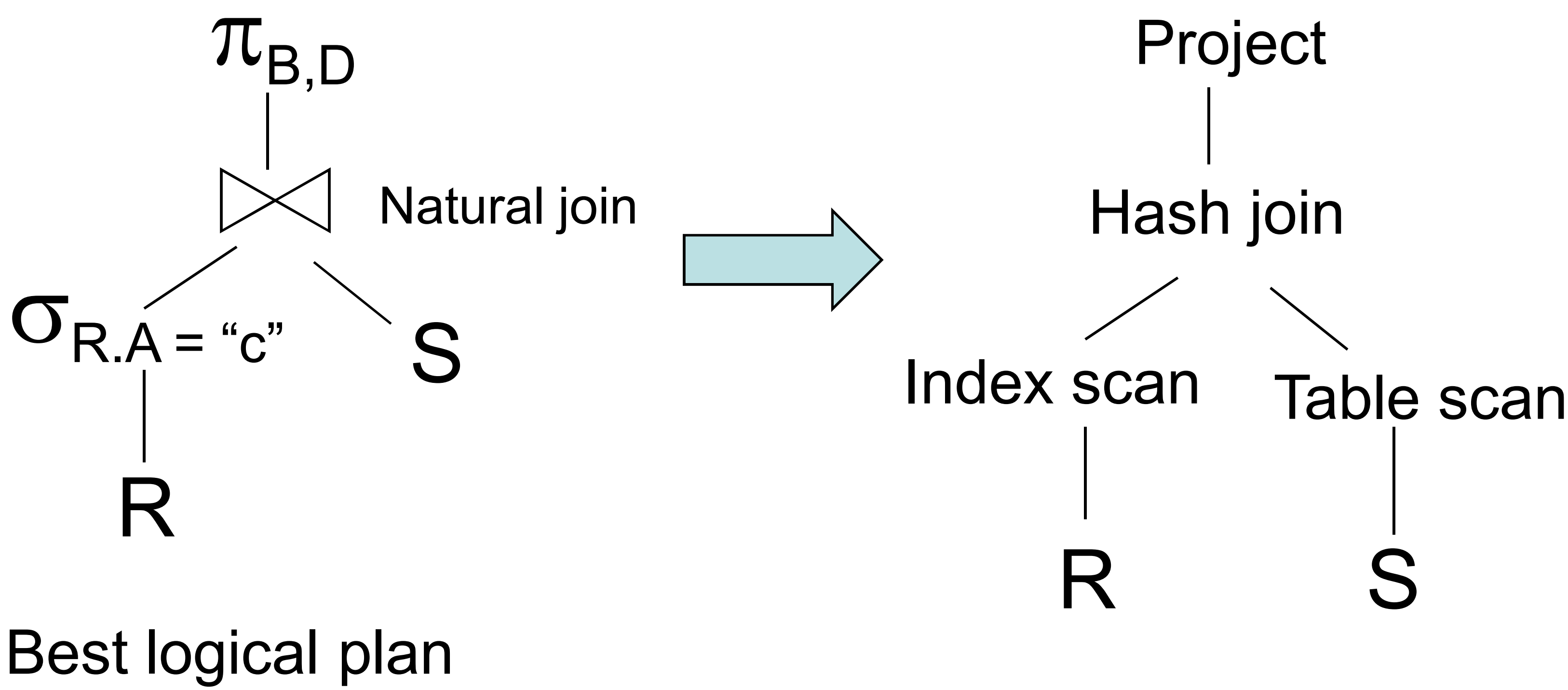
Query Rewriting

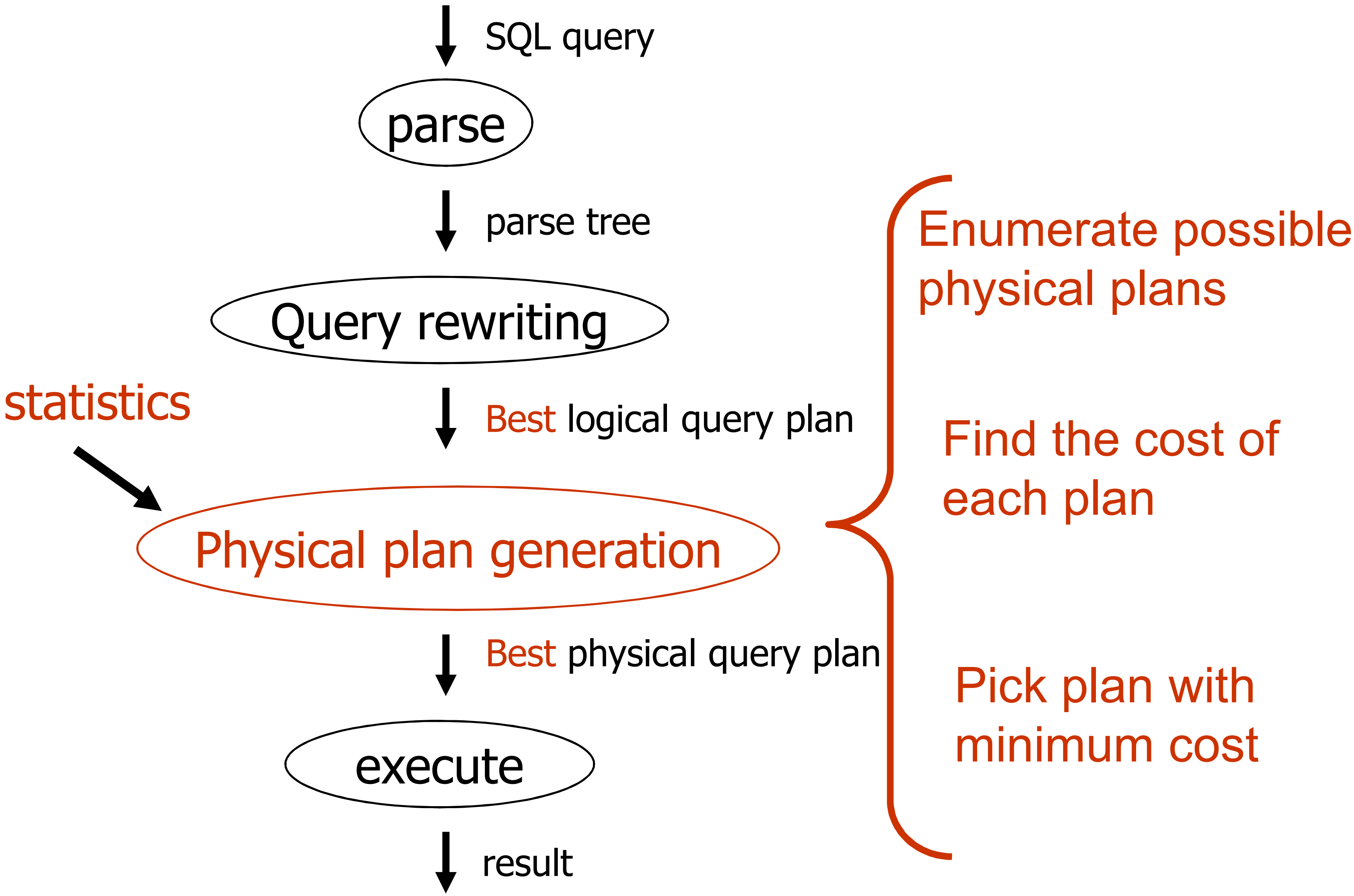


We will revisit it towards the end of this lecture

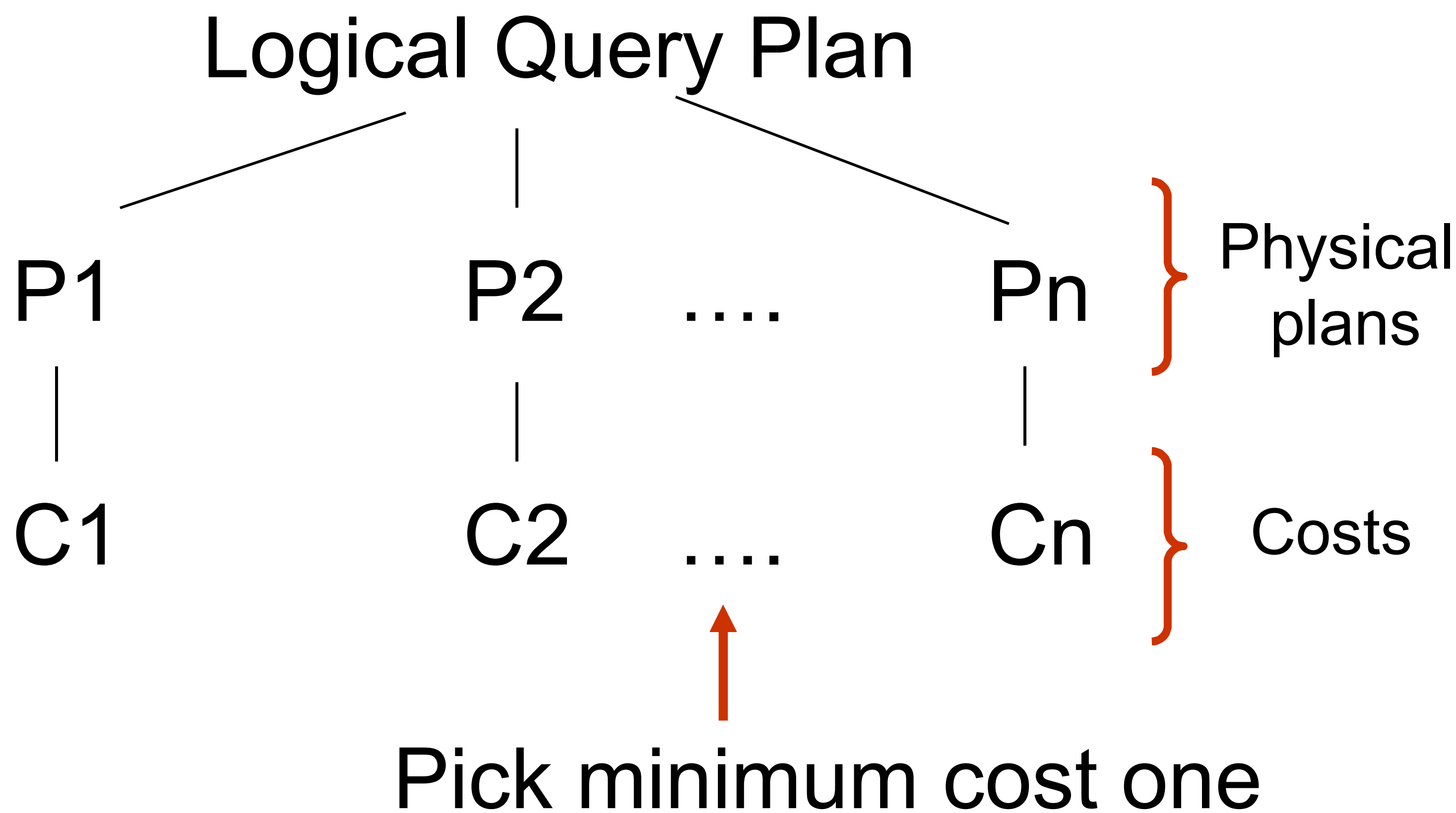

SQL query
parse
parse tree
Query rewriting
statistics
Best logical query plan
Physical plan generation
Best physical query plan
execute
result

Physical Plan Generation




SQL query
parse
parse tree
Query rewriting
statistics
Best logical query plan
Physical plan generation
Best physical query plan
execute
result
Enumerate possible physical plans
Find the cost of each plan
Pick plan with minimum cost

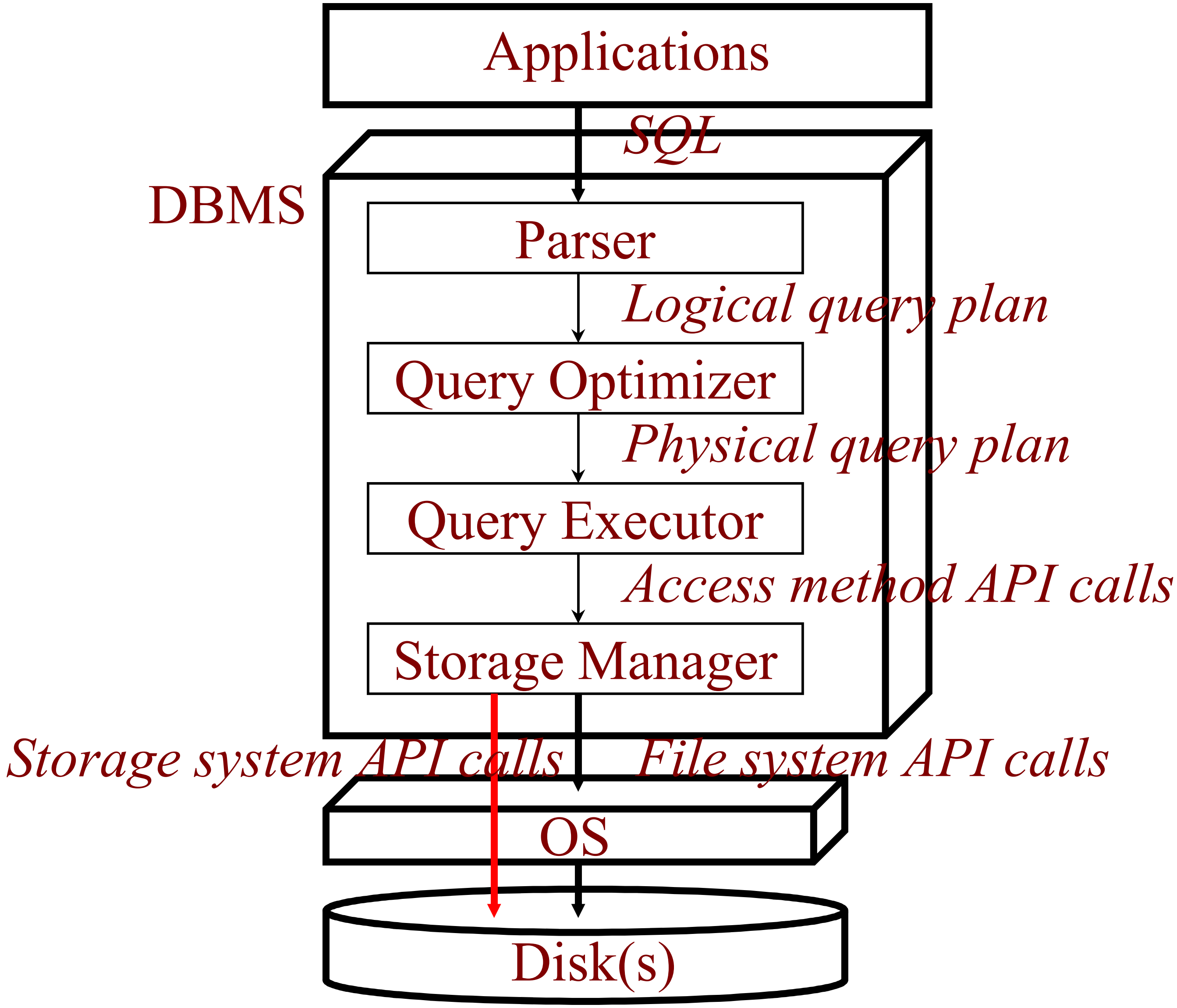
Physical Plan Generation



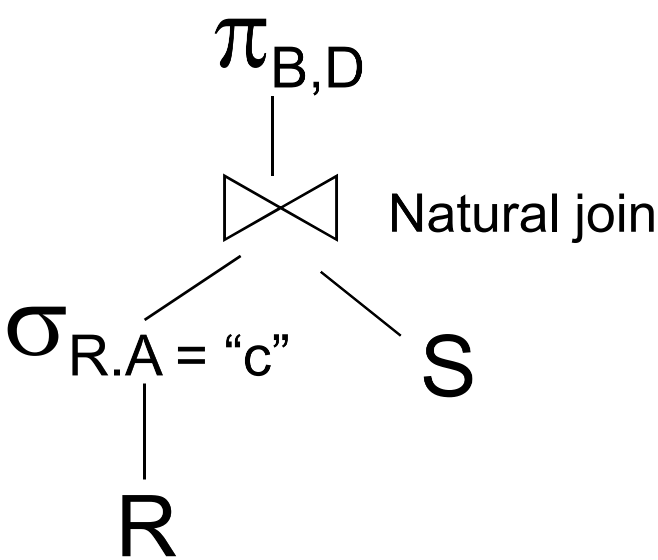
Plans for Query Execution

- Roadmap
  - Path of a SQL query
  - Operator trees
  - Physical Vs Logical plans
  - Plumbing: Materialization Vs pipelining

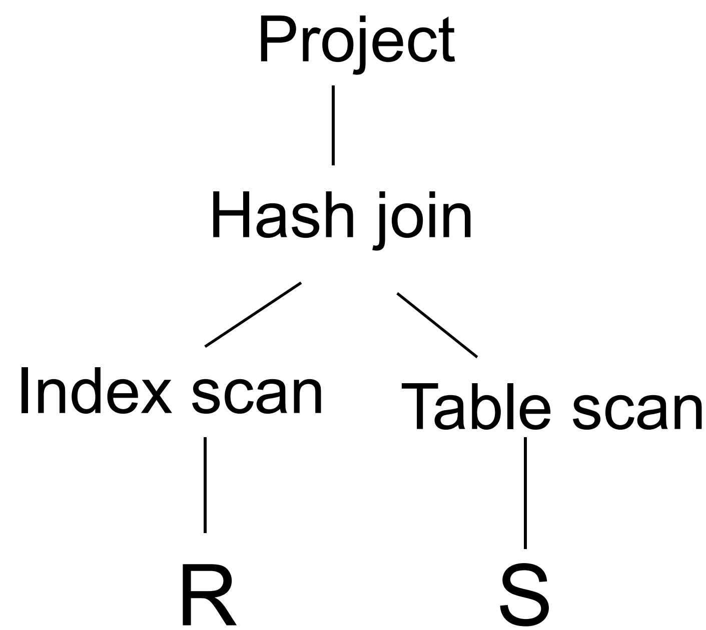
Modern DBMS Architecture



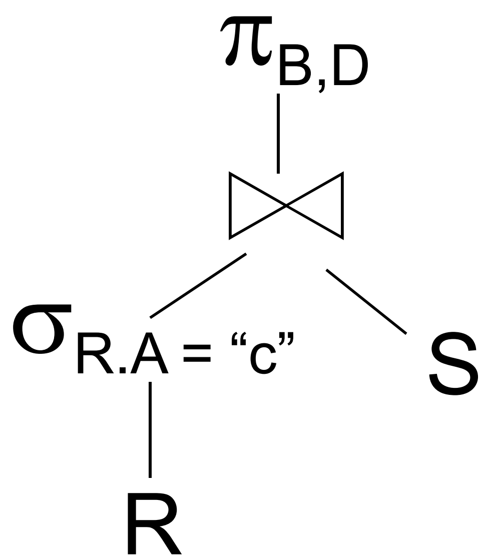
Logical Plans Vs. Physical Plans



Best logical plan

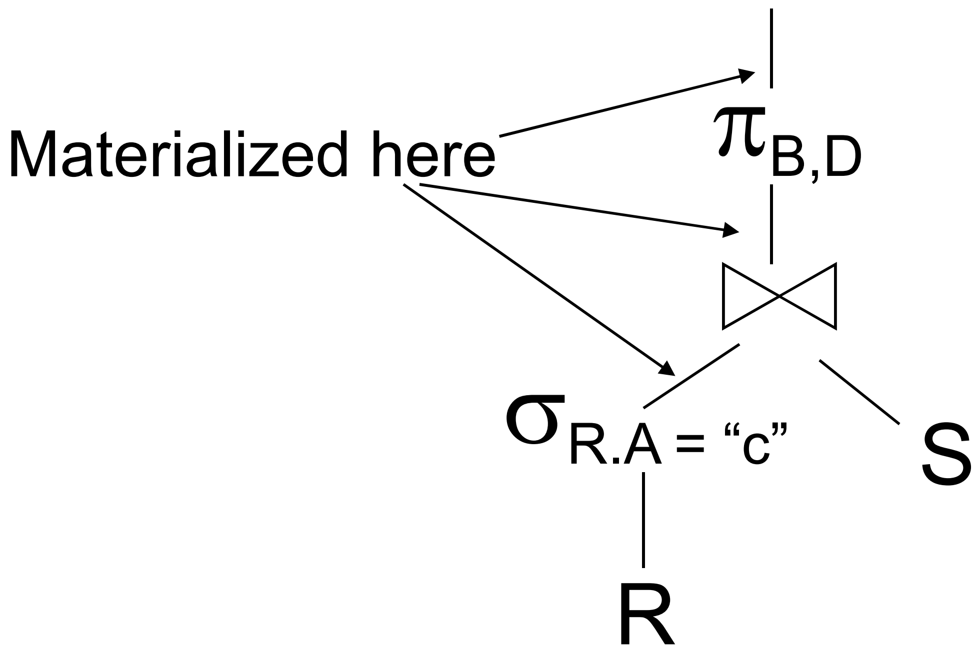


Operator Plumbing

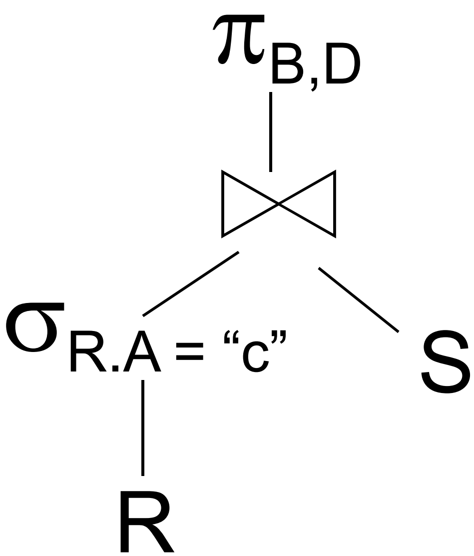


- Materialization: output of one operator written to disk, next operator reads from the disk
- Pipelining: output of one operator directly fed to next operator

Materialization



Iterators: Pipelining



➔ Each operator supports:

- Open()
- GetNext()
- Close()

Iterator for Table Scan (R)

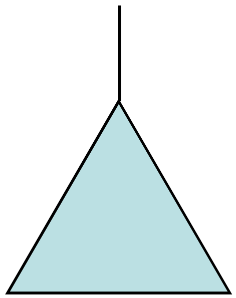
```
Open() {
  /** initialize variables */
  b = first block of R;
  t = first tuple in block b;
}
```

```
GetNext() {
  IF (t is past last tuple in block b) {
      set b to next block;
      IF (there is no next block)
          /** no more tuples */
          RETURN EOT;
      ELSE t = first tuple in b;
  }
  /** return current tuple */
  oldt = t;
  set t to next tuple in block b;
  RETURN oldt;
}
```

```
Close() {
  /** nothing to be done */
}
```

Iterator for Select

$\sigma_{R.A = \text{“c”}}$



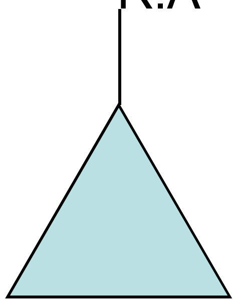
```
Open() {
  /** initialize child */
  Child.Open();
}

Close() {
  /** inform child */
  Child.Close();
}
```

```
GetNext() {
  LOOP:
      t = Child.GetNext();
      IF (t == EOT) {
        /** no more tuples */
        RETURN EOT;
      }
      ELSE IF (t.A == “c”)
        RETURN t;
  ENDLOOP:
}
```

Iterator for Sort

$\tau_{R.A}$

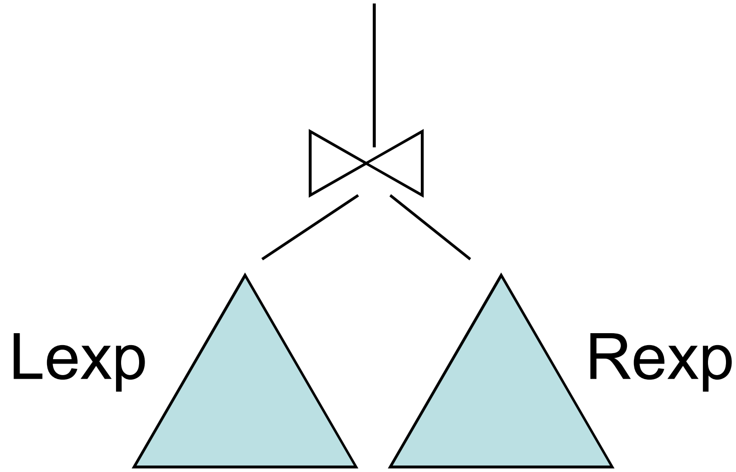


```
GetNext() {
   IF (more tuples)
      RETURN next tuple in order;
   ELSE RETURN EOT;
}
```

```
Open() {
  /** Bulk of the work is here */
  Child.Open();
  Read all tuples from Child
     and sort them
}
```

```
Close() {
  /** inform child */
  Child.Close();
}
```

Iterator for Tuple Nested Loop Join



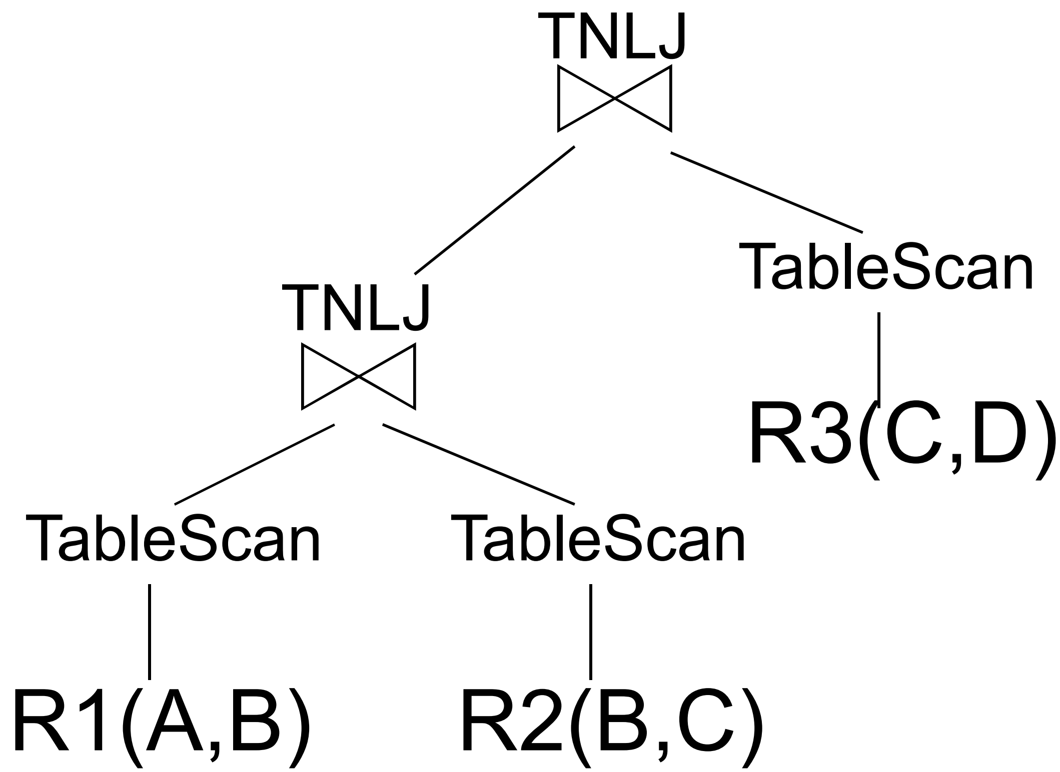
- TNLJ (conceptually)

  for each $r \in$ Lexp do

  for each $s \in$ Rexp do

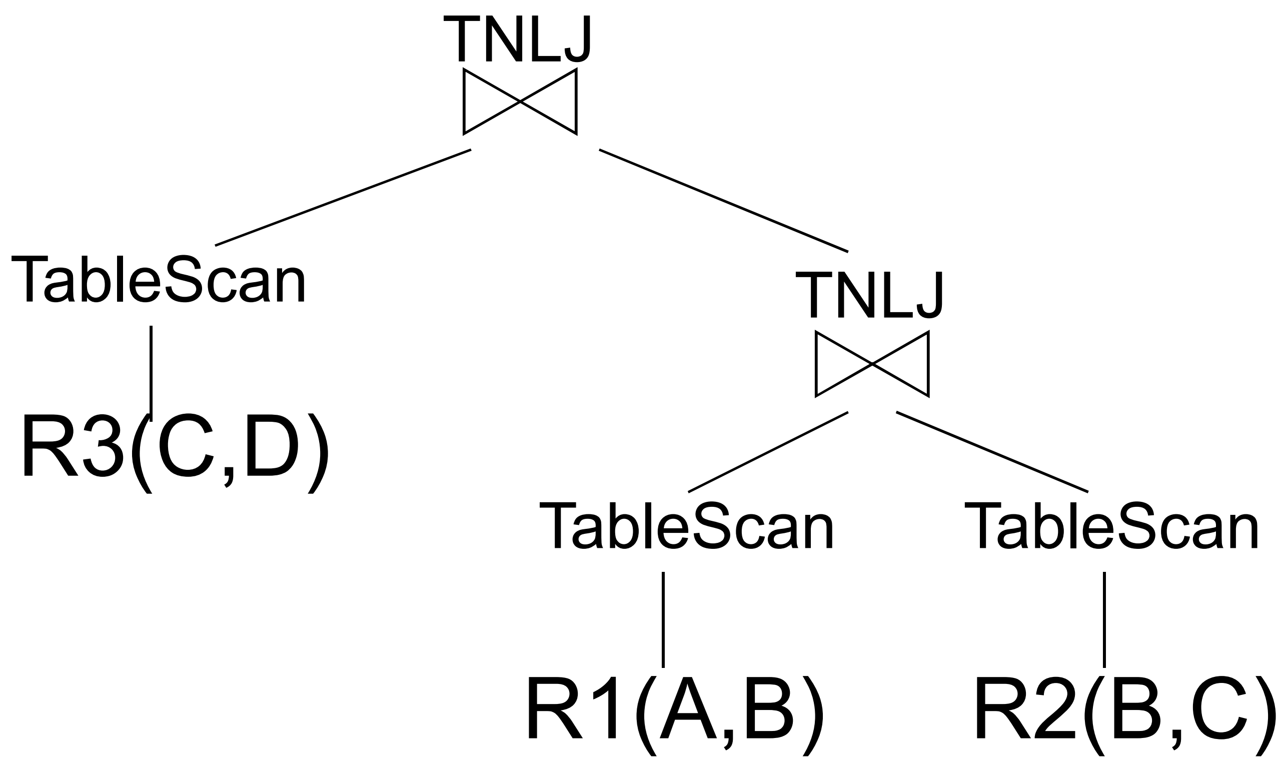
  if Lexp.C = Rexp.C, output r,s

Example 1: Left-Deep Plan



Question: What is the sequence of getNext() calls?

Example 2: Right-Deep Plan



Question: What is the sequence of getNext() calls?

Example

Worked on blackboard

Cost Measure for a Physical Plan

- There are many cost measures
  - Time to completion
  - Number of I/Os (we will see a lot of this)
  - Number of getNext() calls
- Tradeoff: Simplicity of estimation Vs. Accurate estimation of performance as seen by user

Textbook outline

Chapter 15

15.1 Physical operators

- Scan, Sort (Ch. 11.4), Indexes (Ch. 13)

15.2-15.6 Implementing operators + estimating their cost

15.8 Buffer Management

15.9 Parallel Processing

Textbook outline (contd.)

Chapter 16

16.1 Parsing

16.2 Algebraic laws

16.3 Parse tree → logical query plan

16.4 Estimating result sizes

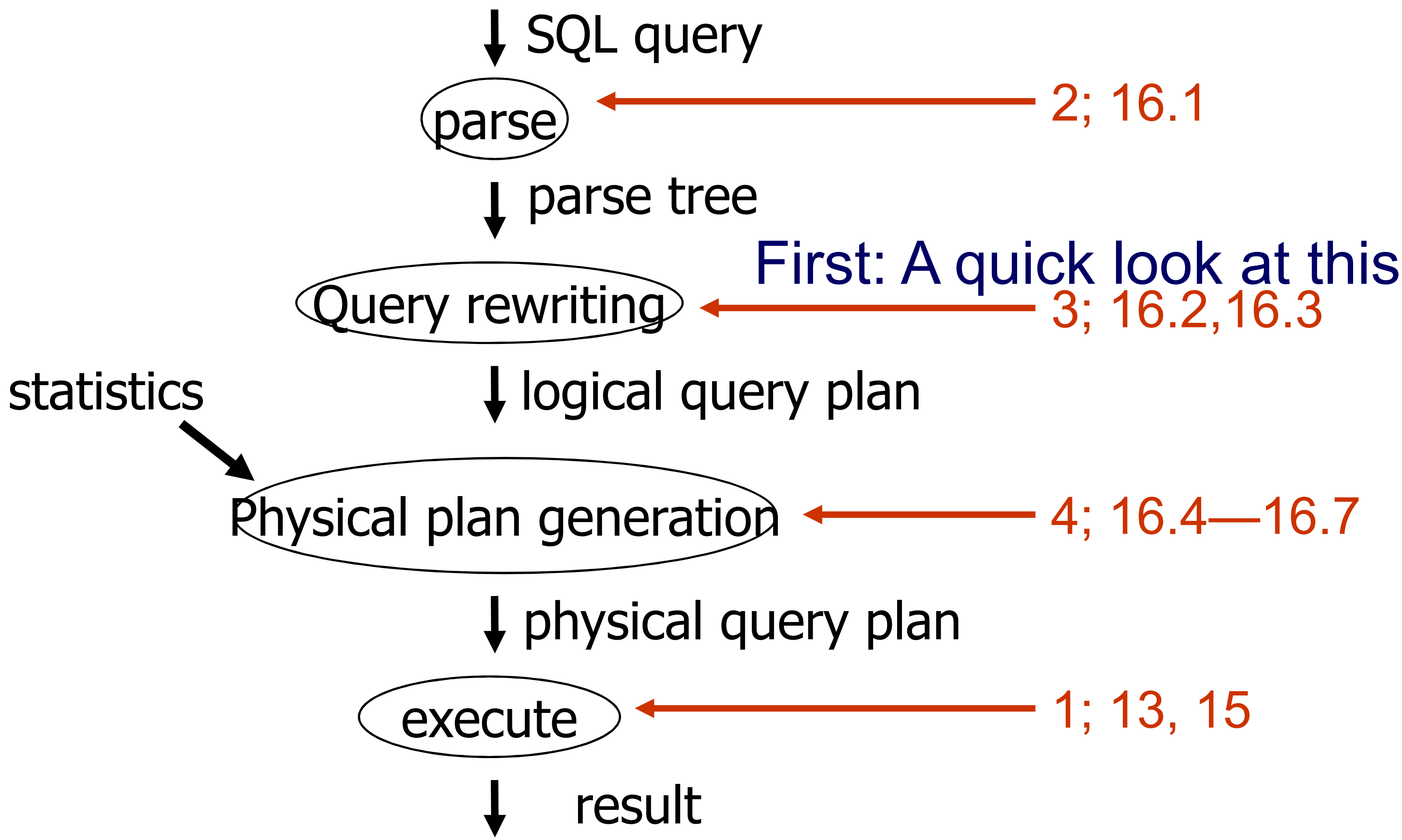
16.5-16.7 Cost based optimization

Background Material

Chapter 5 Relational Algebra

Chapter 6 SQL

Query Processing - In class order



Why do we need Query Rewriting?

- Pruning the HUGE space of physical plans
  - Eliminating redundant conditions/operators
  - Rules that will improve performance with very high probability
- Preprocessing
  - Getting queries into a form that we know how to handle best

➔ Reduces optimization time drastically without noticeably affecting quality

Some Query Rewrite Rules

- Transform one logical plan into another
  - Do not use statistics
- Equivalences in relational algebra
- Push-down predicates
- Do projects early
- Avoid cross-products if possible

Equivalences in Relational Algebra

$R \bowtie S = S \bowtie R$ Commutativity

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$ Associativity

Also holds for: Cross Products, Union, Intersection

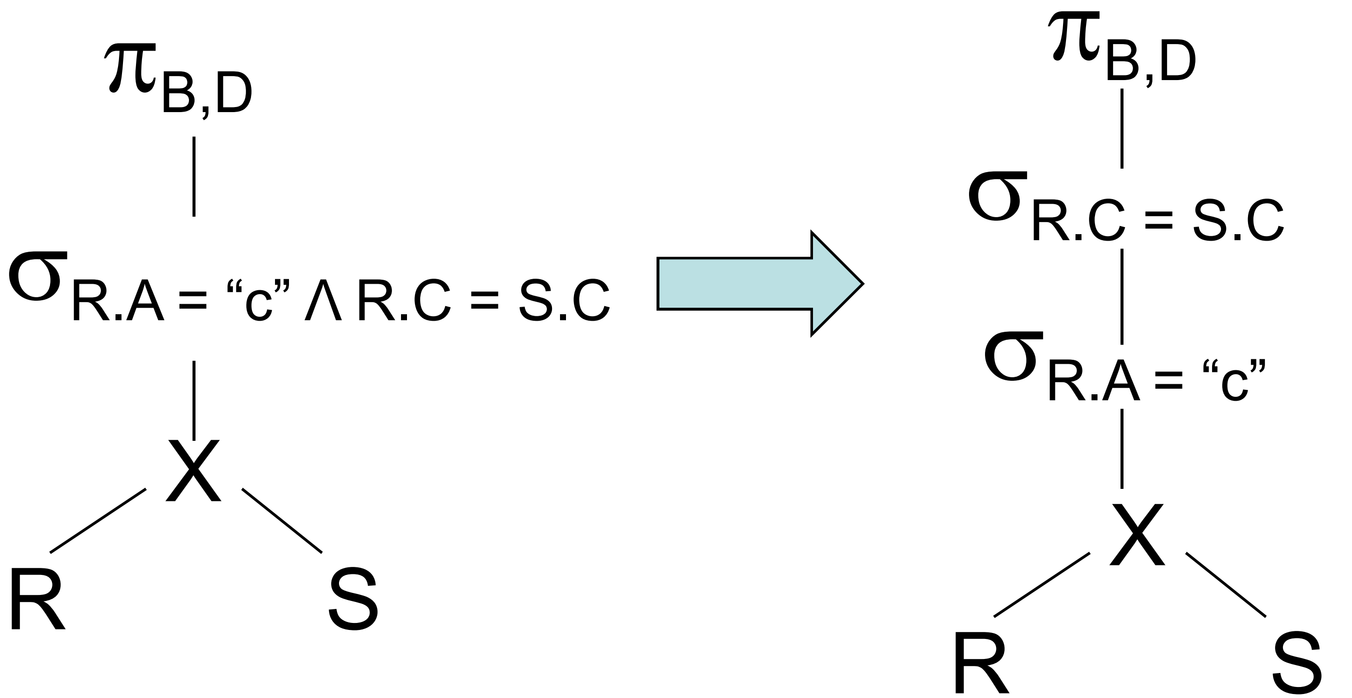
R x S = S x R

(R x S) x T = R x (S x T)

R U S = S U R

R U (S U T) = (R U S) U T

Apply Rewrite Rule (1)



$$\Pi_{B,D}\,[\,\sigma_{R.C=S.C}\,[\sigma_{R.A=\text{“c”}}(R \; X \; S)]]$$

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

$$\pi_{xy}(R) = \pi_x[\pi_y(R)]$$

Rules: $\sigma + \bowtie$ combined

Let p = predicate with only R attribs

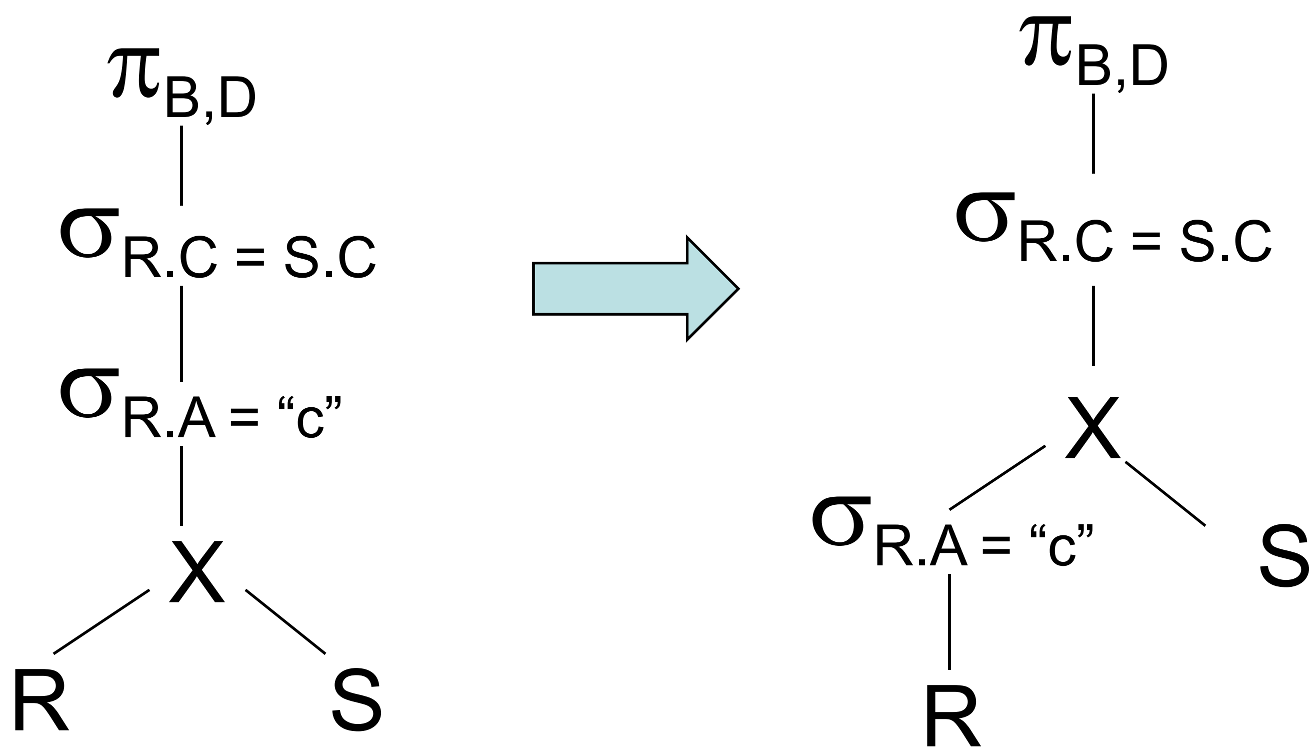
q = predicate with only S attribs

m = predicate with only R,S attribs

$$\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$$

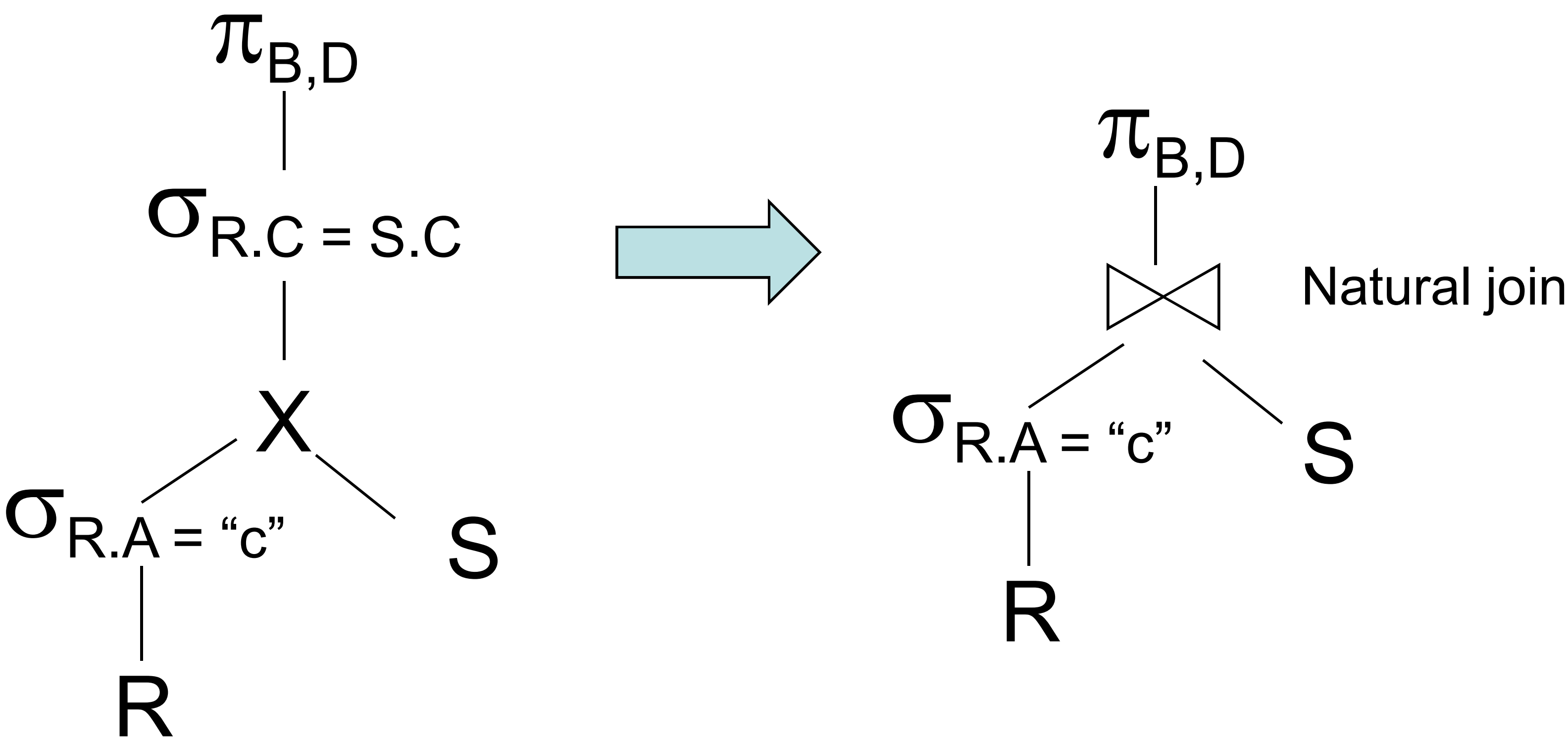
$$\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$$

Apply Rewrite Rule (2)



$$\Pi_{B,D}\,[\,\sigma_{R.C=S.C}\,[\sigma_{R.A=\text{“c”}}(R)]\ X\ S]$$

Apply Rewrite Rule (3)



$$\Pi_{B,D}\,[[\sigma_{R.A=\text{“c”}}(R)] \bowtie S]$$

Rules: σ + $\bowtie$ combined (continued)

$$\sigma_{p \wedge q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)]$$

$$\sigma_{p \wedge q \wedge m} (R \bowtie S) =$$

$$\sigma_m \left[(\sigma_p R) \bowtie (\sigma_q S) \right]$$

$$\sigma_{pvq} (R \bowtie S) =$$

$$\left[(\sigma_p R) \bowtie S \right] \cup \left[R \bowtie (\sigma_q S) \right]$$

Which are "good" transformations?

- □ $\sigma_{p1 \wedge p2}(R) \rightarrow \sigma_{p1}[\sigma_{p2}(R)]$
- □ $\sigma_{p}(R \bowtie S) \rightarrow [\sigma_{p}(R)] \bowtie S$
- □ $R \bowtie S \rightarrow S \bowtie R$
- □ $\pi_{x}[\sigma_{p}(R)] \rightarrow \pi_{x}\{\sigma_{p}[\pi_{xz}(R)]\}$

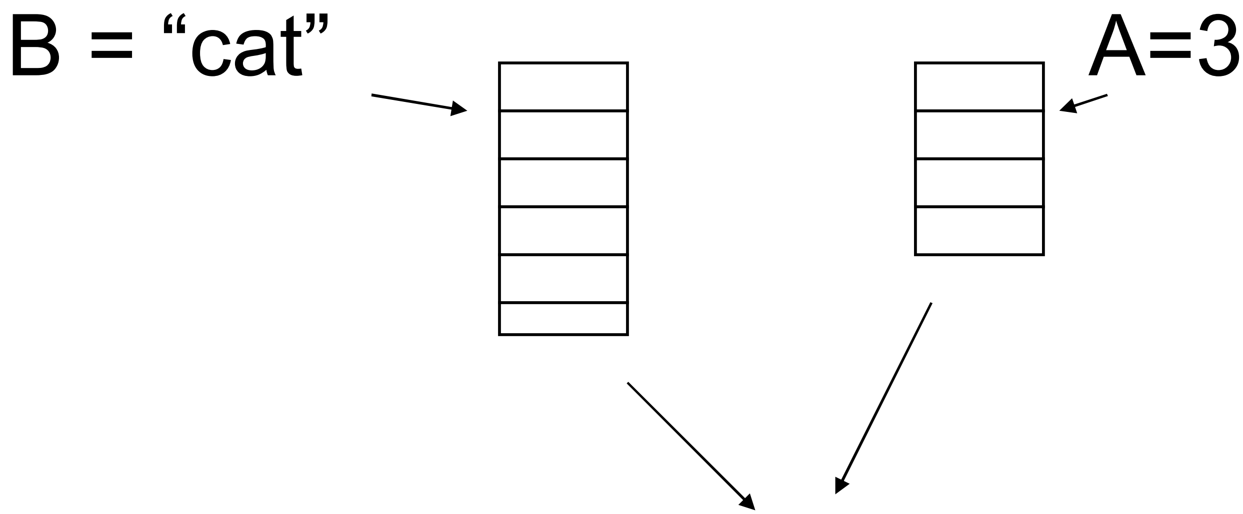
Conventional wisdom: do projects early

<u>Example</u>: R(A,B,C,D,E)

P: (A=3) $\wedge$ (B="cat")

$$\pi_E \{\sigma_p (R)\} \quad \text{vs.} \quad \pi_E \left\{\sigma_p\{\pi_{ABE}(R)\}\right\}$$

But: What if we have A, B indexes?



Intersect pointers to get
pointers to matching tuples

Bottom line:

- No transformation is always good
- Some are usually good:
  - Push selections down
  - Avoid cross-products if possible
  - Subqueries → Joins

Avoid Cross Products (if possible)

Select B,D
From R,S,T,U
Where R.A = S.B $\wedge$
R.C=T.C $\wedge$ R.D = U.D

- Which join trees avoid cross-products?
- If you can't avoid cross products, perform them as late as possible

More Query Rewrite Rules

- Transform one logical plan into another
  - Do not use statistics
- Equivalences in relational algebra
- Push-down predicates
- Do projects early
- Avoid cross-products if possible
- Use left-deep trees
- Subqueries → Joins
- Use of constraints, e.g., uniqueness