Data-Intensive Computing Systems

Introduction to Query Processing

Shivnath Babu

Query Processing

Declarative SQL Query $\rightarrow$ Query Plan

NOTE: You will not be tested on how well you know SQL. Understanding the SQL introduced in class will be sufficient (a primer follows). SQL is described in Chapter 6, GMUW.

Focus: Relational System (i.e., data is organized as tables, or relations)

SQL Primer

We will focus on SPJ, or Select-Project-Join Queries

Select <attribute list>

From <relation list>

Where <condition list>

Example Filter Query over R(A,B,C):

Select B

From R

Where R.A = "c" $\wedge$ R.C > 10

SQL Primer (contd.)

We will focus on SPJ, or Select-Project-Join-Queries

Select <attribute list>

From <relation list>

Where <condition list>

Example Join Query over R(A,B,C) and S(C,D,E):

Select B, D

From R, S

Where R.A = “c” $\wedge$ S.E = 2 $\wedge$ R.C = S.C

R	A	B	C
	a	1	10
	b	1	20
	c	2	10
	d	2	35
	e	3	45

S	C	D	E
	10	x	2
	20	y	2
	30	z	2
	40	x	1
	50	y	3

Select B,D

From R,S

Where R.A = “c” ∧

S.E = 2 ∧ R.C=S.C

Answer

B	D
2	x

- How do we execute this query?

Select B,D

From R,S

Where R.A = “c” $\wedge$ S.E = 2 $\wedge$ R.C=S.C

One idea

- Do Cartesian product
- Select tuples
- Do projection

R X S

Select B,D

From R,S

Where R.A = “c”

∧ S.E = 2 ∧

R.C=S.C

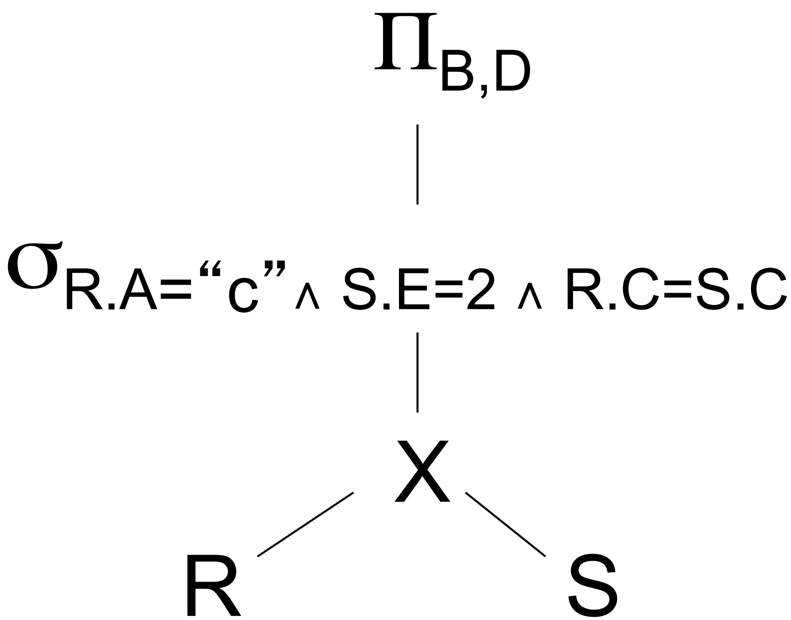
R.A	R.B	R.C	S.C	S.D	S.E
a	1	10	10	x	2
a	1	10	20	y	2
.					
.					
c	2	10	10	x	2
.					
.					

Bingo!

Got one...

Relational Algebra - can be used to describe plans

Ex: Plan I



Relational Algebra Primer (Chapter 5, GMUW)

Select: $\sigma_{R.A=\text{"c"} \wedge R.C=10}$

Project: $\Pi_{B,D}$

Cartesian Product: R X S

Natural Join: $R \bowtie S$

Relational Algebra - can be used to describe plans

Ex: Plan I

$$\Pi_{B,D}$$

$$|$$

$$\sigma_{R.A=\text{“c”} \wedge S.E=2 \wedge R.C=S.C}$$

$$|$$

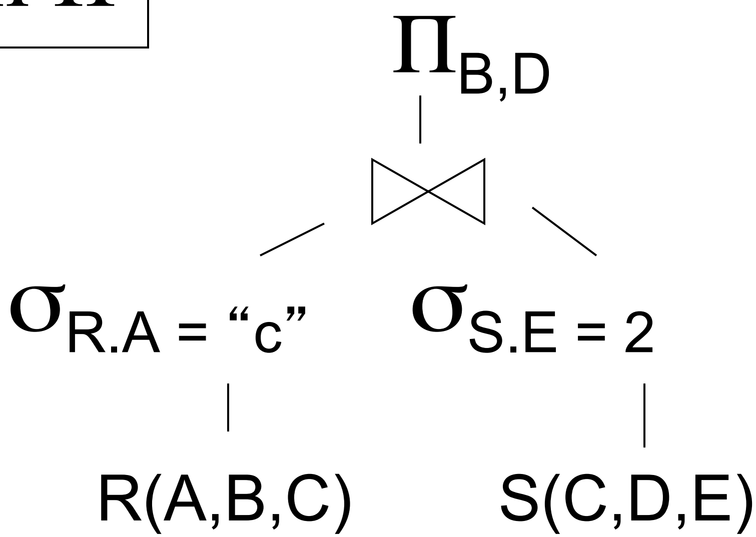
$$X$$

$$R \qquad S$$

OR: $\Pi_{B,D} [\, \sigma_{R.A=\text{“c”} \wedge S.E=2 \wedge R.C = S.C} \,(RXS)]$

Another idea:

Plan II



$\bowtie$ natural join

Select B,D

From R,S

Where R.A = “c” $\wedge$

S.E = 2 $\wedge$ R.C=S.C

R

A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

σ (R)

A	B	C
c	2	10

σ(S)

C	D	E
10	x	2
20	y	2
30	z	2

S

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

$\bowtie$

Select B,D

From R,S

Where R.A = “c” $\wedge$

S.E = 2 $\wedge$ R.C=S.C

Plan III

Use R.A and S.C Indexes

(1) Use R.A index to select R tuples with R.A = "c"

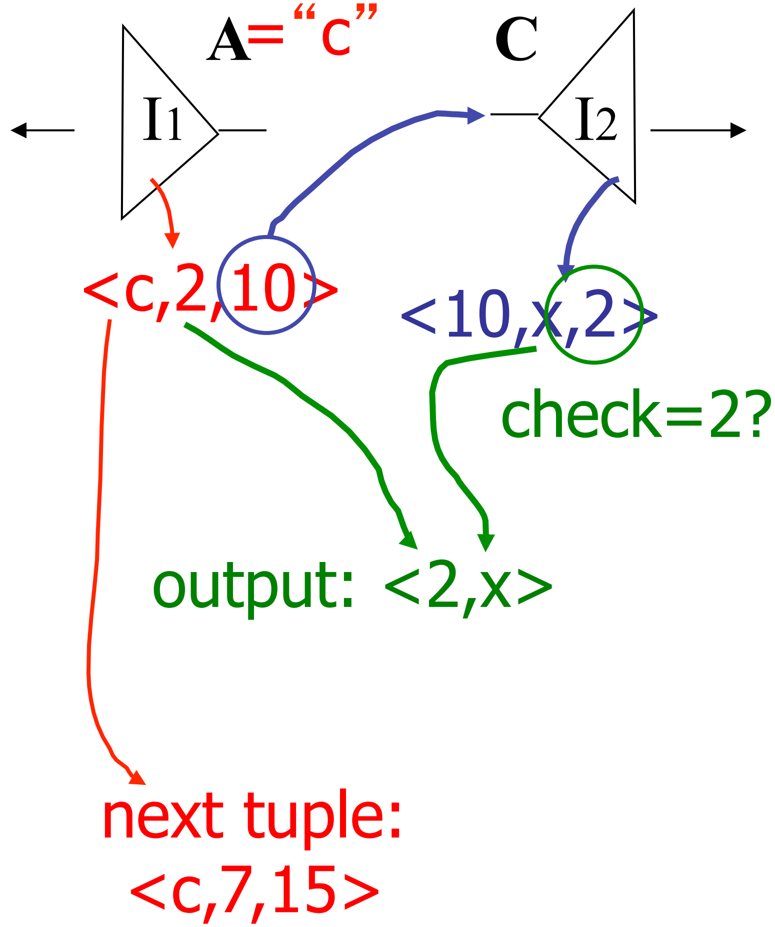
(2) For each R.C value found, use S.C index to find matching tuples

(3) Eliminate S tuples S.E $\neq$ 2

(4) Join matching R,S tuples, project B,D attributes, and place in result

R

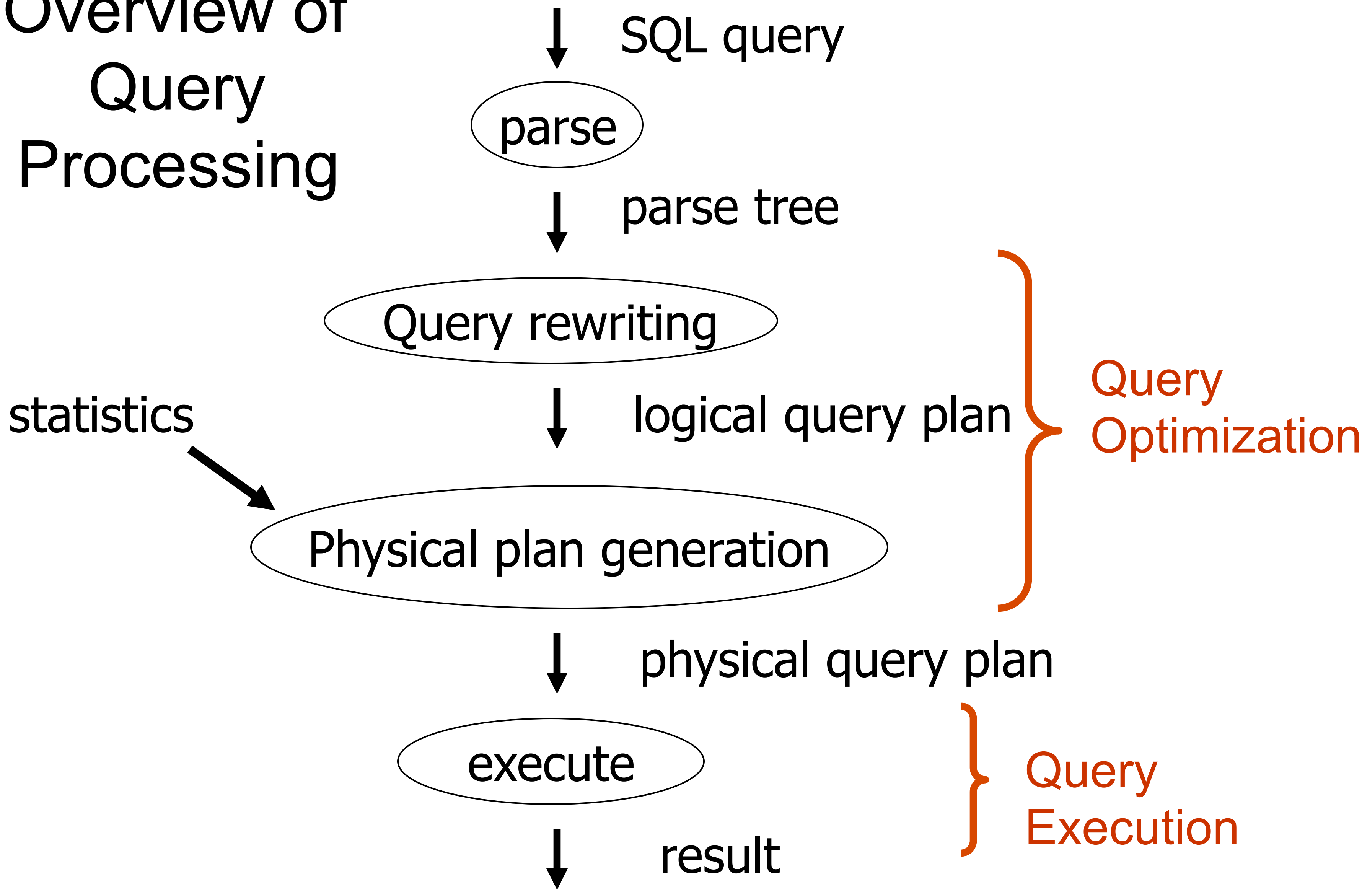
A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45
c	7	15



S

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

Overview of Query Processing



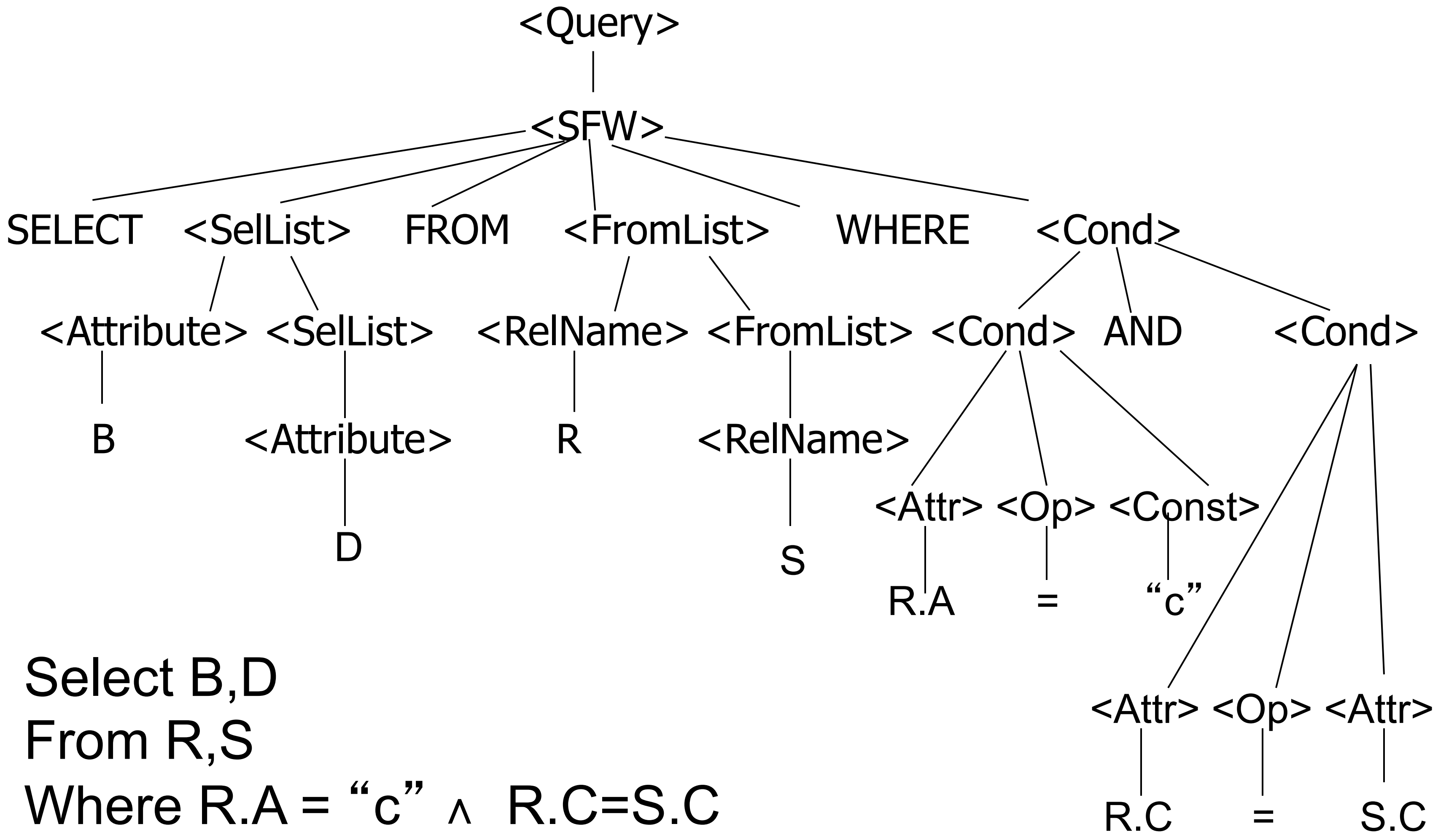
Example Query

Select B,D

From R,S

Where R.A = “c” $\wedge$ R.C=S.C

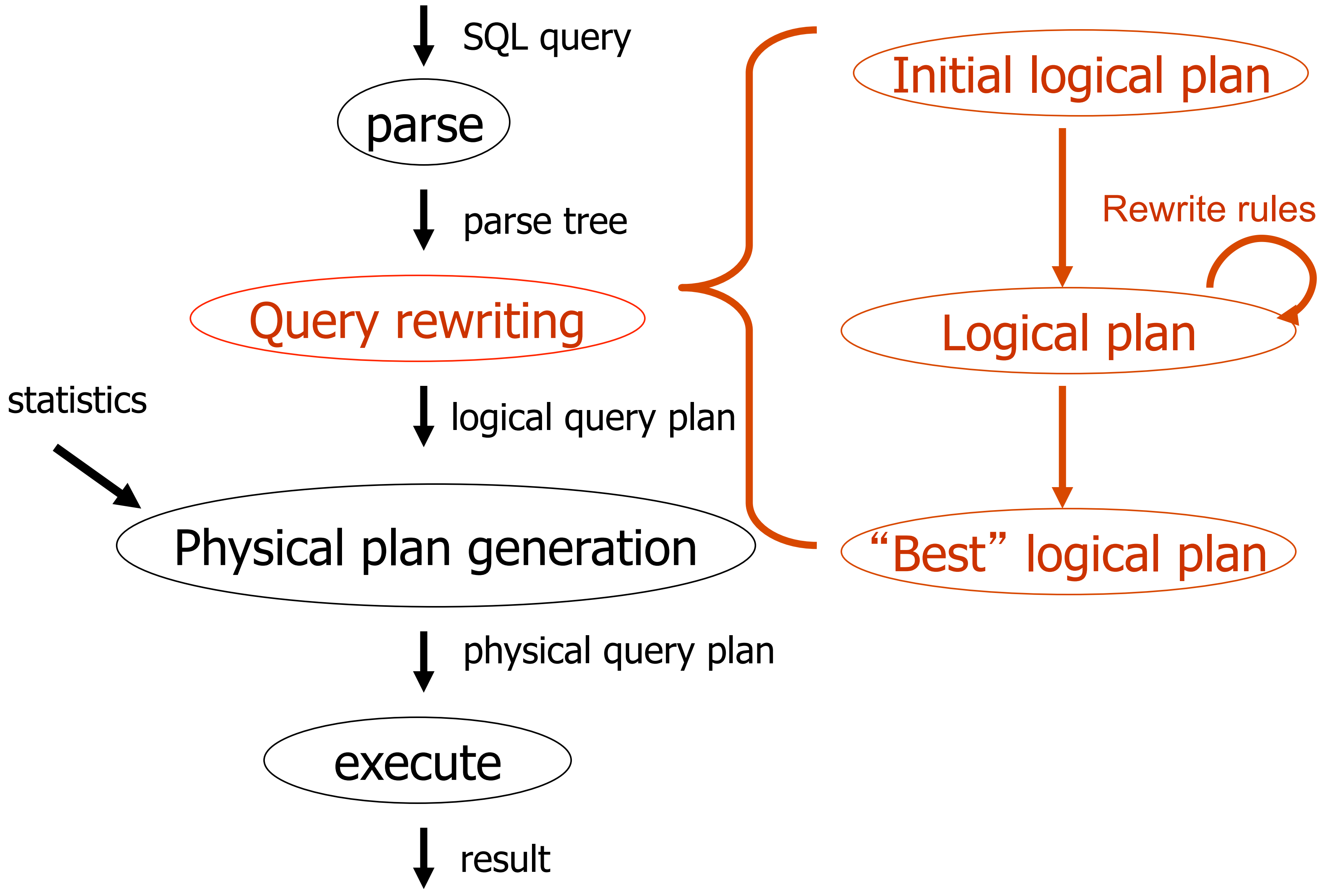
Example: Parse Tree



Select B,D
From R,S
Where R.A = "c" $\wedge$ R.C=S.C

Along with Parsing …

- Semantic checks
  - Do the projected attributes exist in the relations in the From clause?
  - Ambiguous attributes?
  - Type checking, ex: R.A > 17.5
- Expand views


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

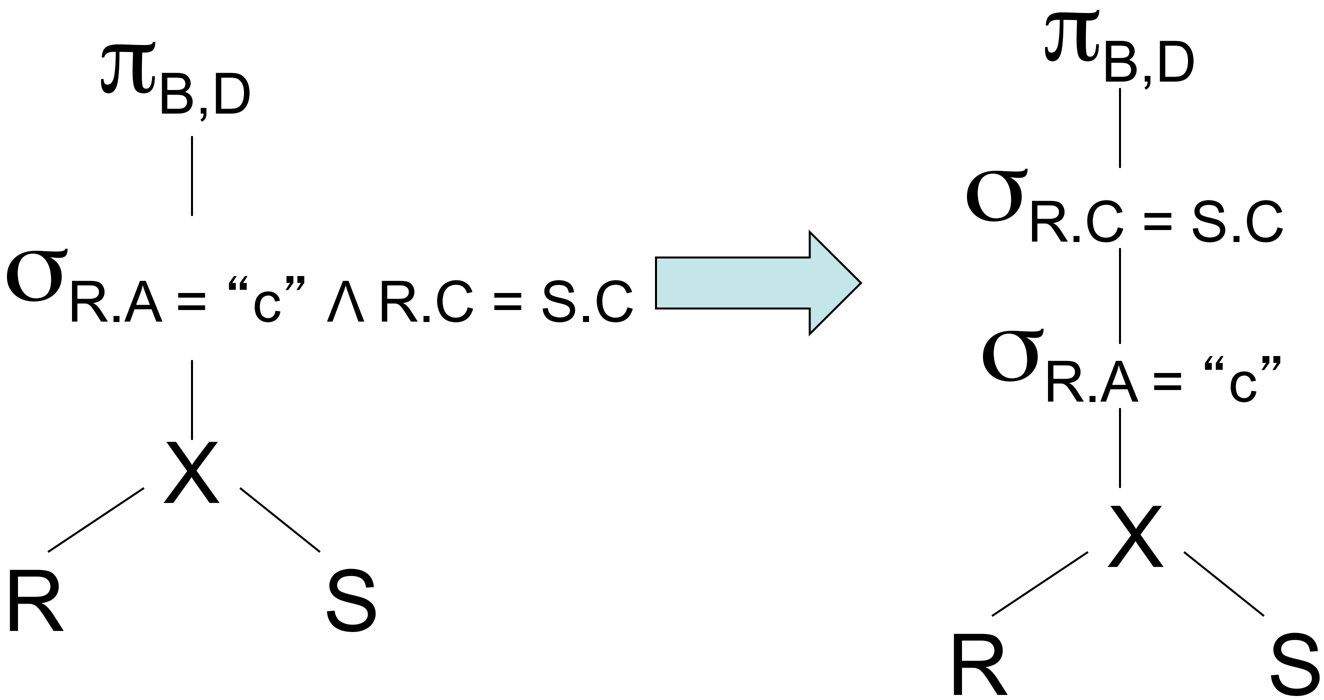
Initial Logical Plan

Select B,D
From R,S
Where R.A = "c" $\wedge$
R.C=S.C

$$\pi_{B,D}$$
$$\sigma_{R.A = \text{"c"} \wedge R.C = S.C}$$
$$X$$
$$R \qquad S$$

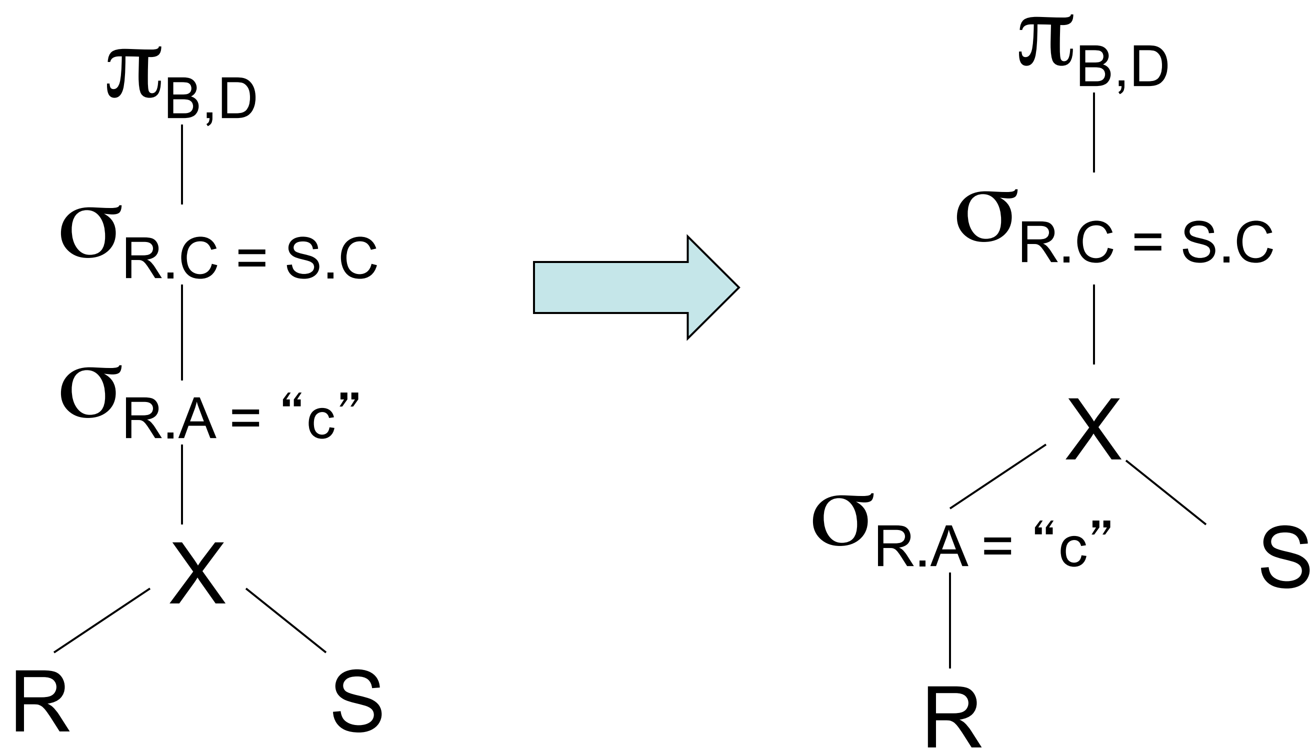
Relational Algebra: $\Pi_{B,D} [\sigma_{R.A=\text{"c"} \wedge R.C = S.C} (RXS)]$

Apply Rewrite Rule (1)



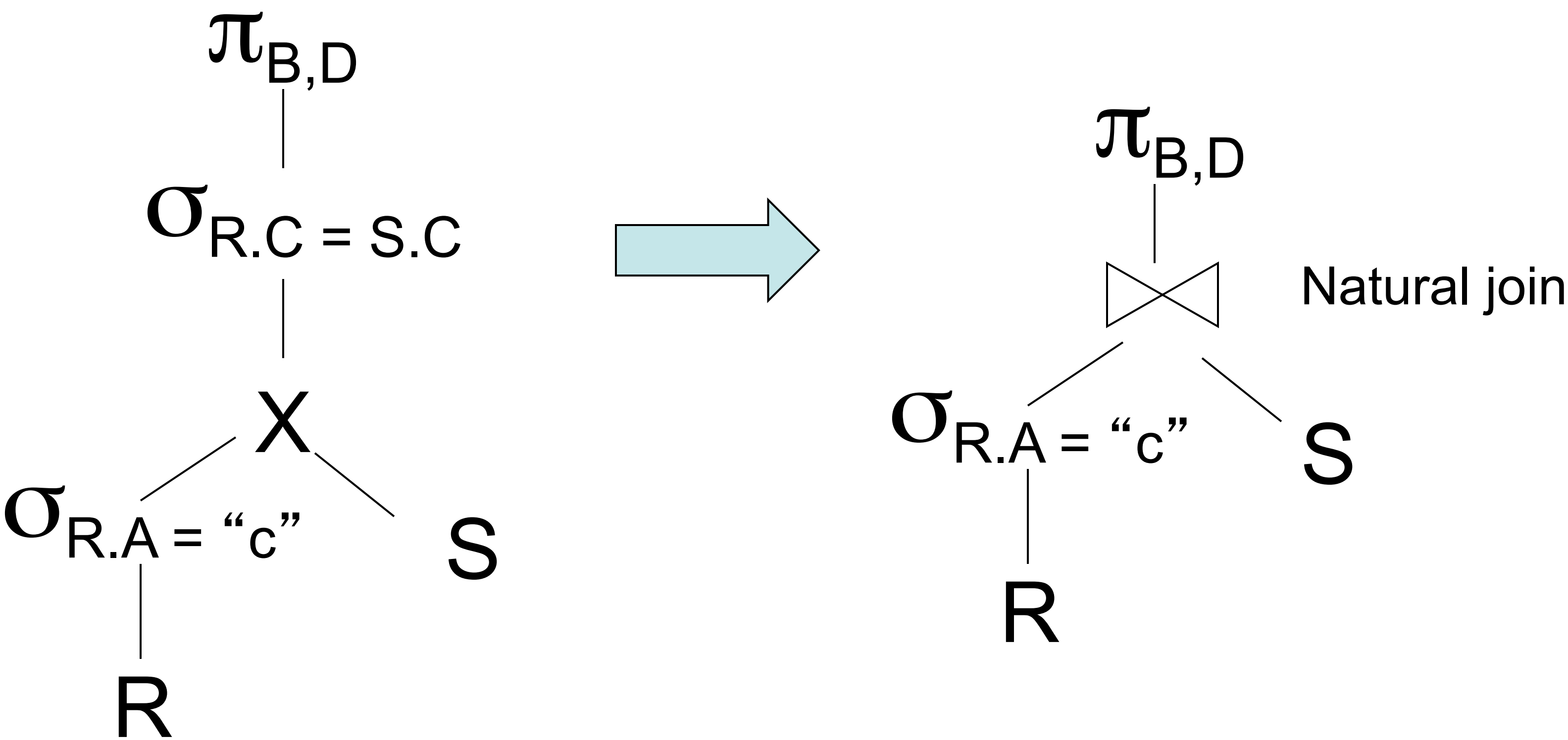
$$\Pi_{B,D}\,[\,\sigma_{R.C=S.C}\,[\sigma_{R.A=\text{“c”}}(R \text{ X } S)]]$$

Apply Rewrite Rule (2)



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

Apply Rewrite Rule (3)



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

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

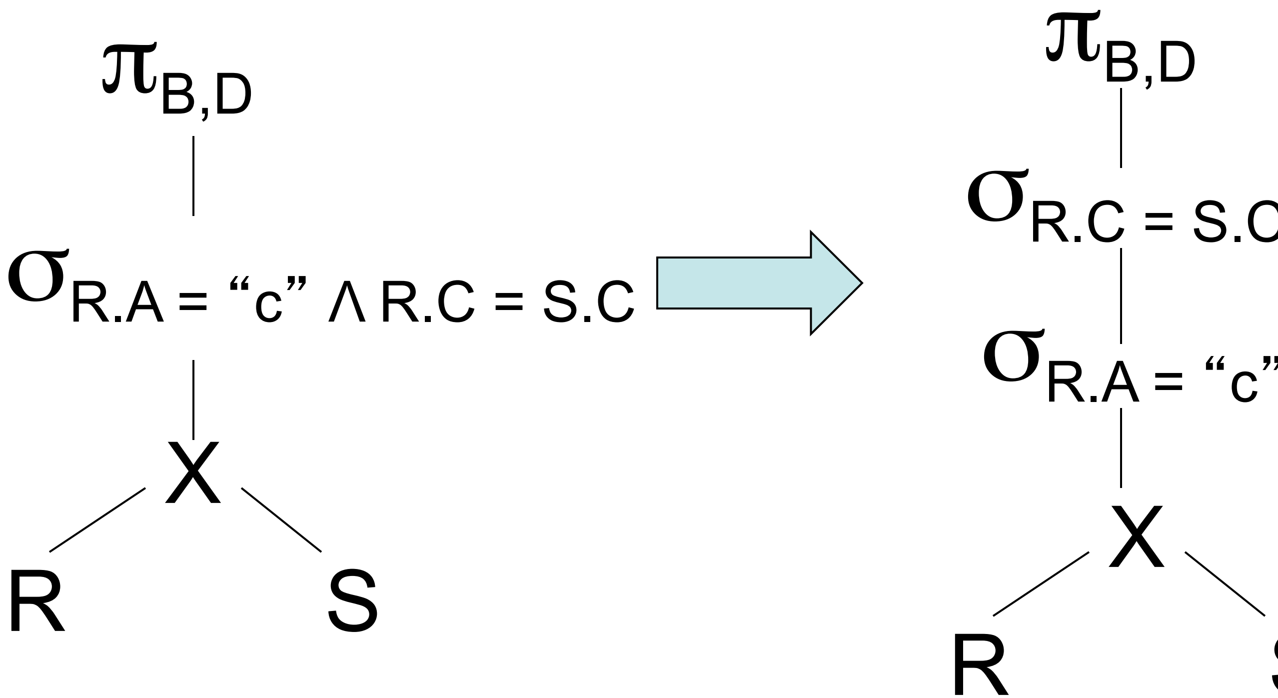
$R \times S = S \times R$

$(R \times S) \times T = R \times (S \times T)$

$R \cup S = S \cup R$

$R \cup (S \cup T) = (R \cup S) \cup T$

Apply Rewrite Rule (1)



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

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

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

Rules: $\sigma + \bowtie$ combined

Let p = predicate with only R attribs

q = predicate with only S attribs

m = predicate with only R,S attribs

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

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

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

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

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

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

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

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

Which are "good" transformations?

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

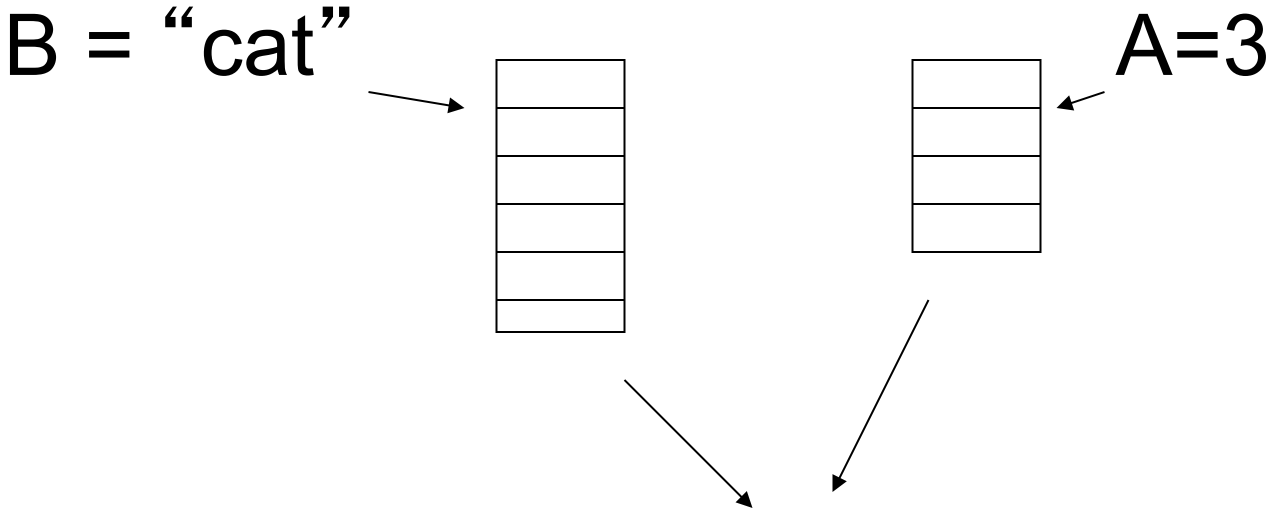
Conventional wisdom: do projects early

Example: R(A,B,C,D,E)

P: (A=3) ∧ (B="cat")

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

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

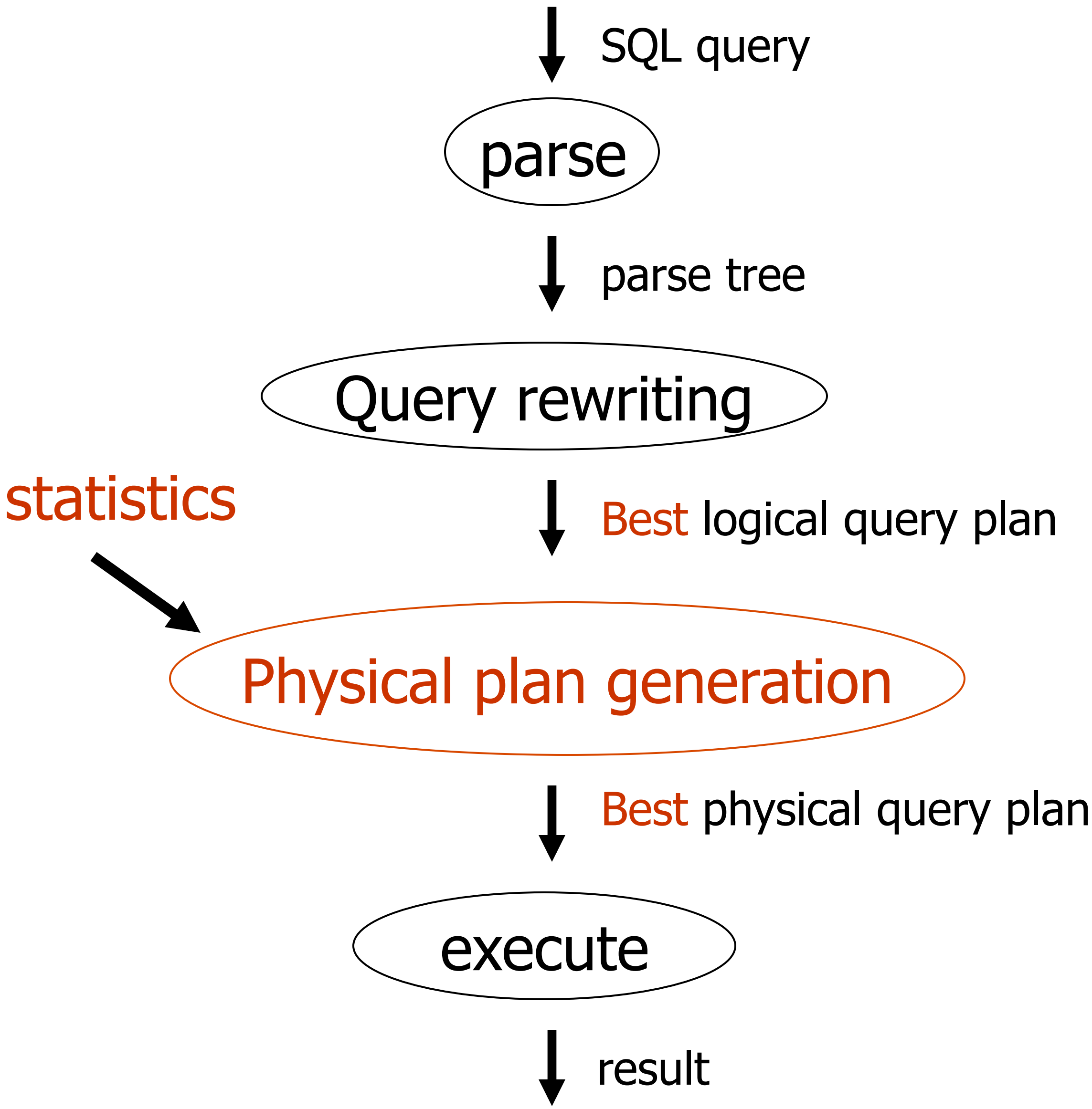
Avoid Cross Products (if possible)

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

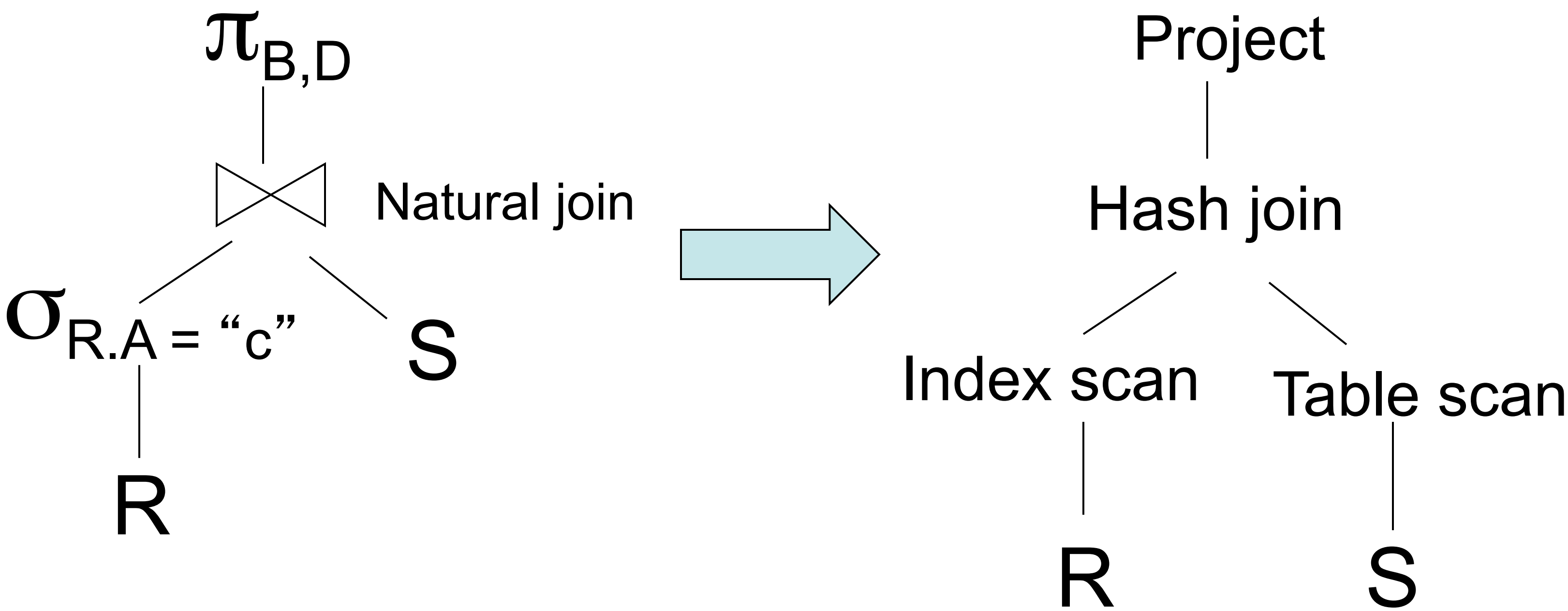
- Which join trees avoid cross-products?
- If you can't avoid cross products, perform them as late as possible

More Query Rewrite Rules

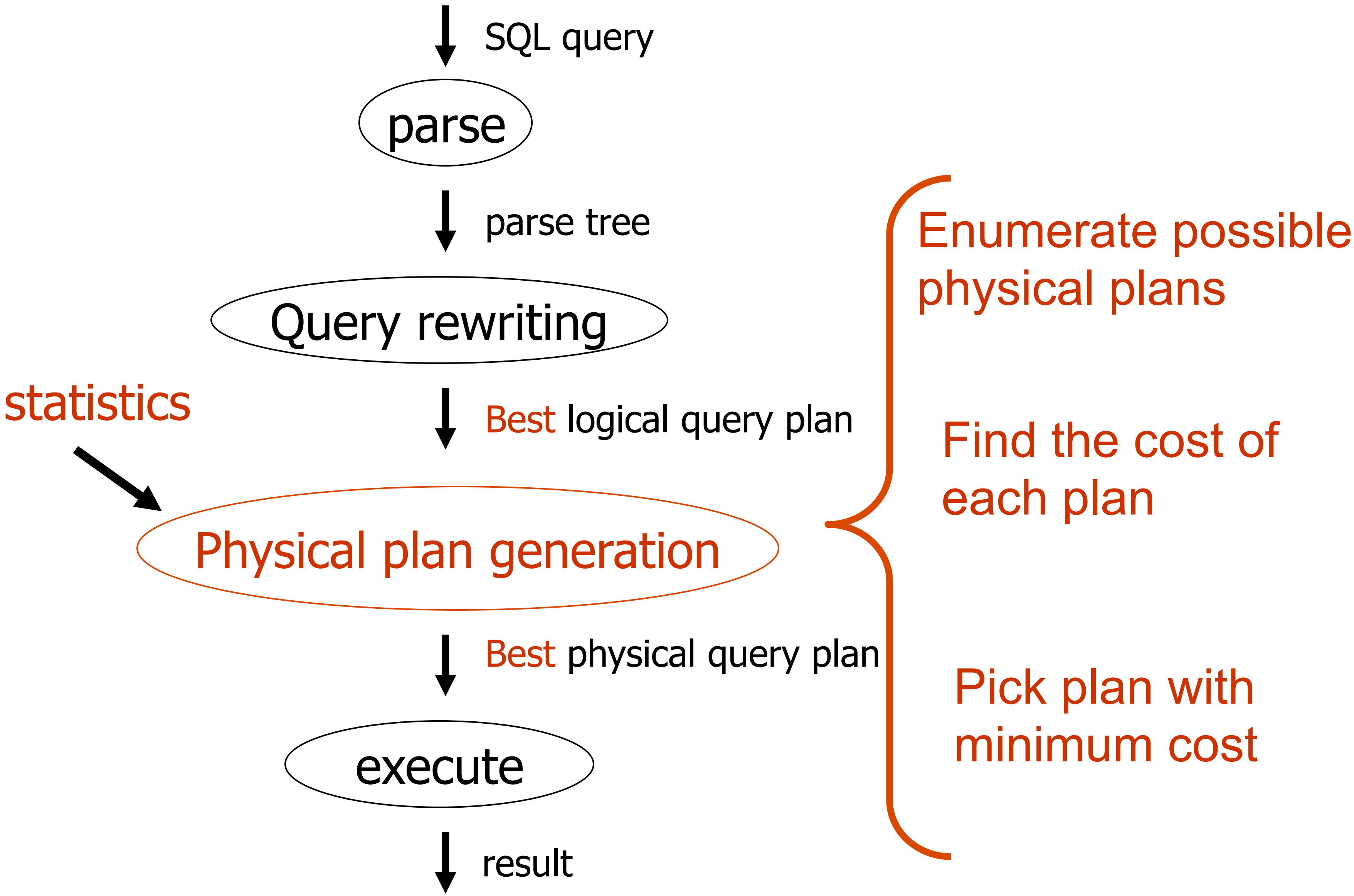
- Transform one logical plan into another
  - Do not use statistics
- Equivalences in relational algebra
- Push-down predicates
- Do projects early
- Avoid cross-products if possible
- Use left-deep trees
- Subqueries → Joins
- Use of constraints, e.g., uniqueness


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

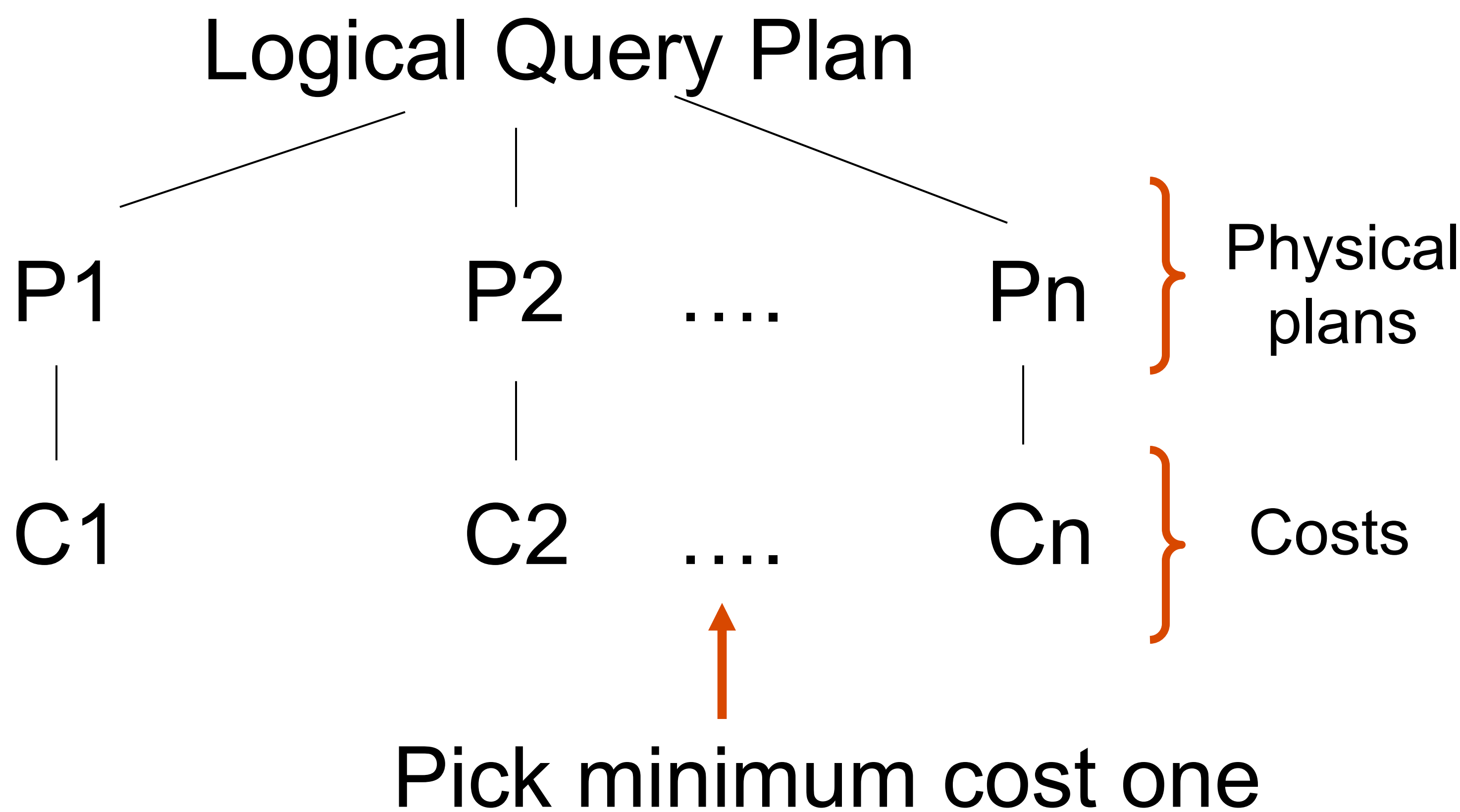
Physical Plan Generation



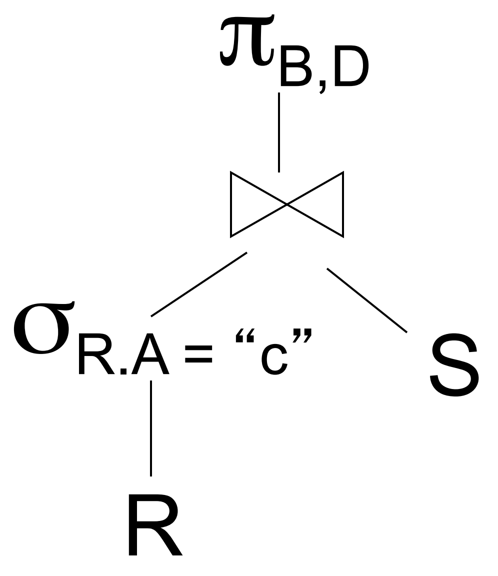
Best logical plan


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

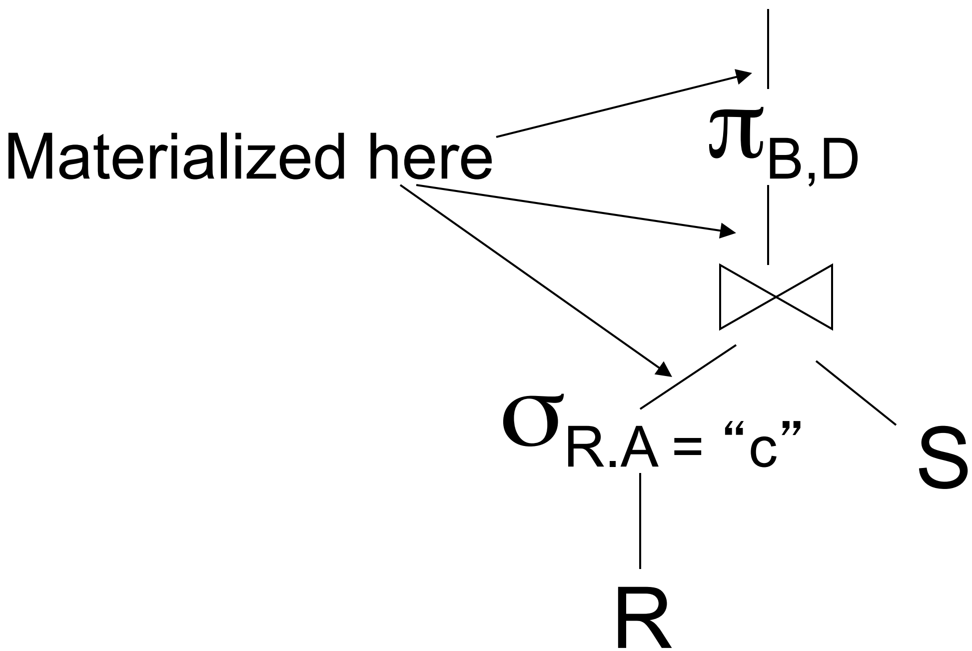


Operator Plumbing

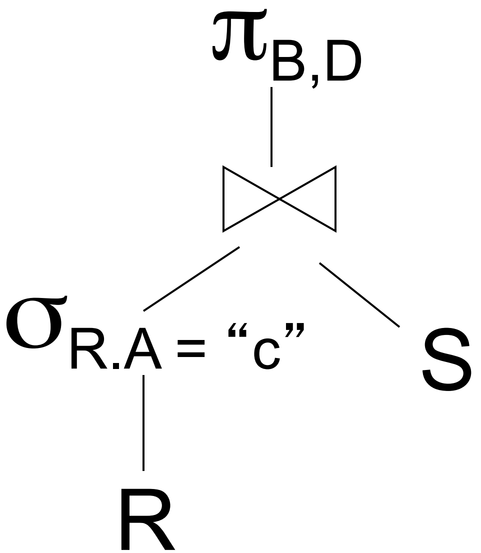


- Materialization: output of one operator written to disk, next operator reads from the disk
- Pipelining: output of one operator directly fed to next operator

Materialization



Iterators: Pipelining



➔ Each operator supports:

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

Iterator for Table Scan (R)

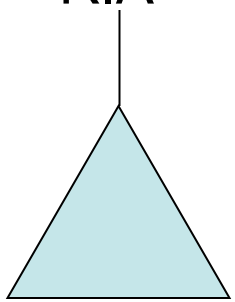
```
Open() {
  /** initialize variables */
  b = first block of R;
  t = first tuple in block b;
}
```

```
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

$\sigma_{R.A = \text{“c”}}$



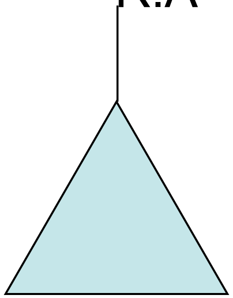
```
Open() {
  /** initialize child */
  Child.Open();
}

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

$\tau_{R.A}$

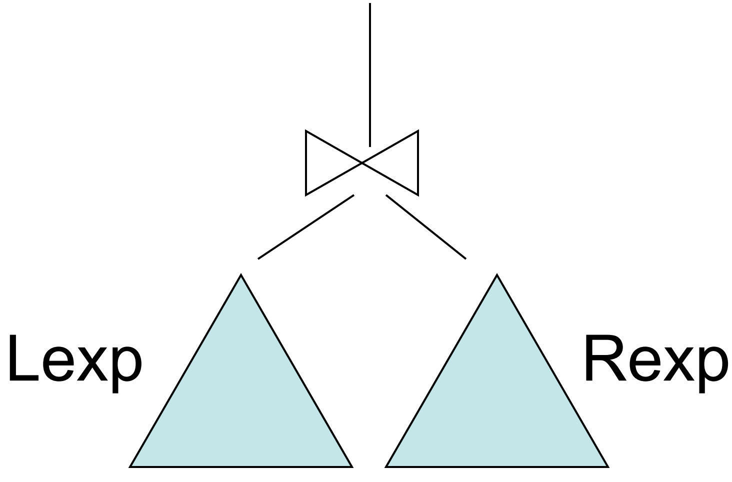


```
GetNext() {
   IF (more tuples)
      RETURN next tuple in order;
   ELSE RETURN EOT;
}
```

```
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



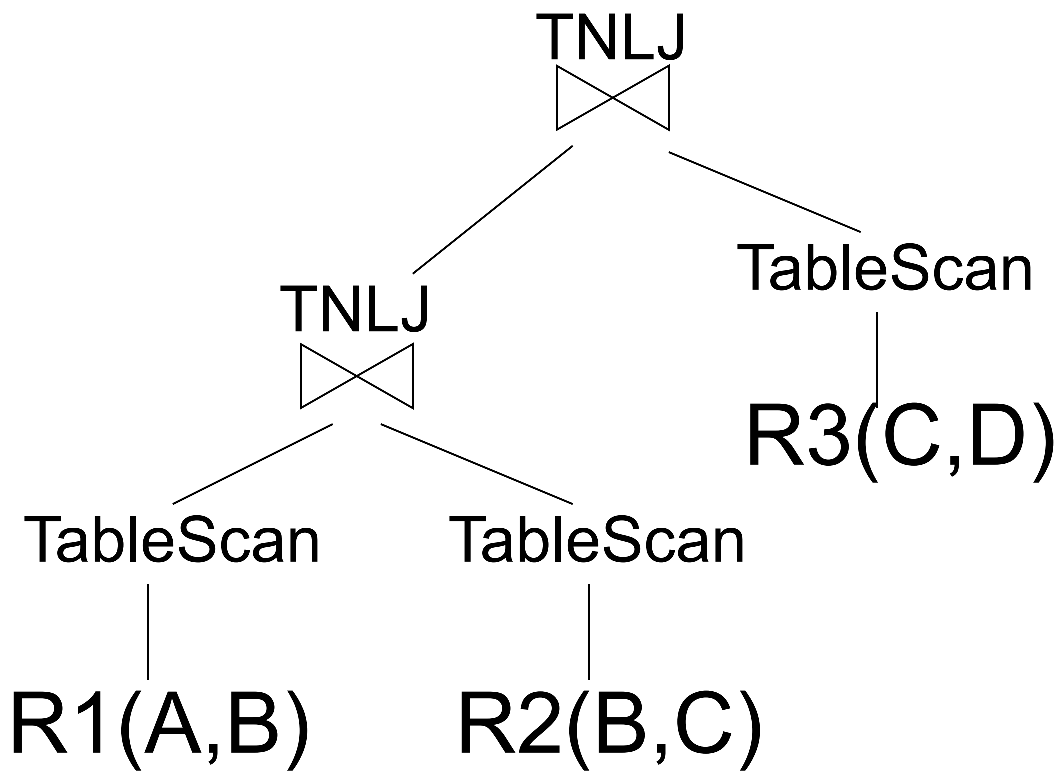
- TNLJ (conceptually)

  for each $r \in$ Lexp do

  for each $s \in$ Rexp do

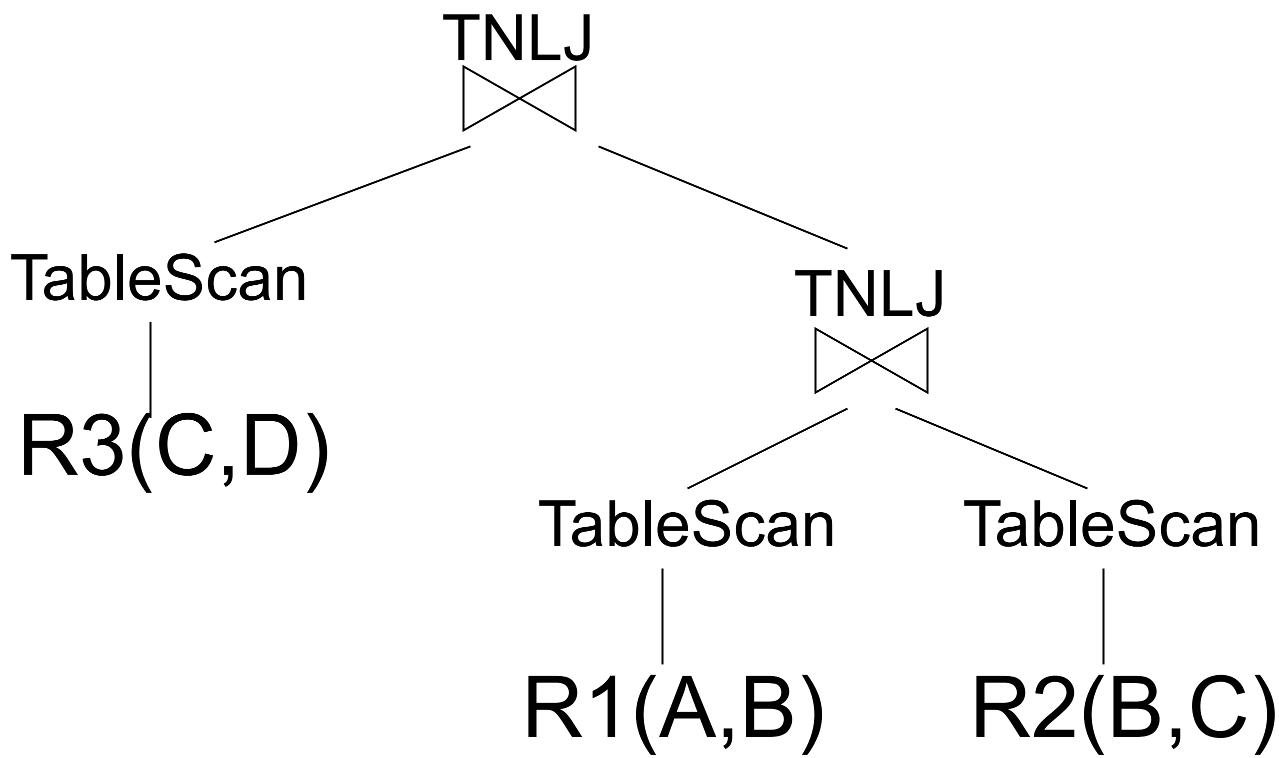
  if Lexp.C = Rexp.C, output r,s

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?

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