CS 4440 A

Emerging Database Technologies

Lecture 16

03/11/26

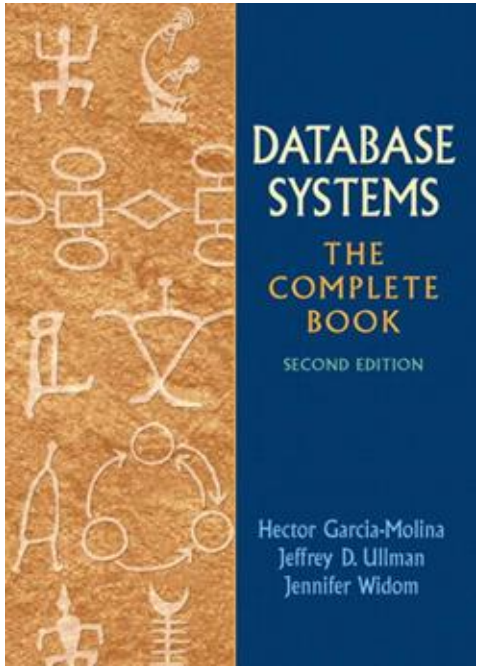
Reading Materials

Query execution (Chapters 15.1 - 15.6)

- Physical operators
- Implementing operators and estimating costs

Query optimization (Chapters 16.1 - 16.5)

- Parsing
- Algebraic laws
- Parse tree -> logical query plan
- Estimating result sizes
- Cost-based optimization



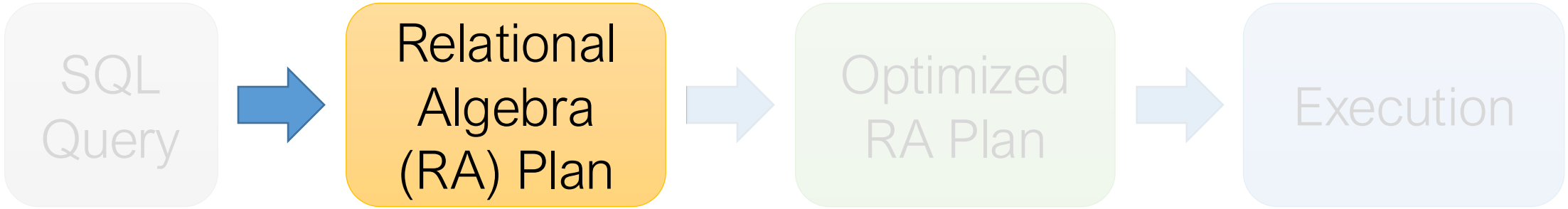
Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang.

Agenda

1. Logical Optimization
2. Physical Optimization
3. Estimating cost of a physical plan

Recap: RDBMS Architecture

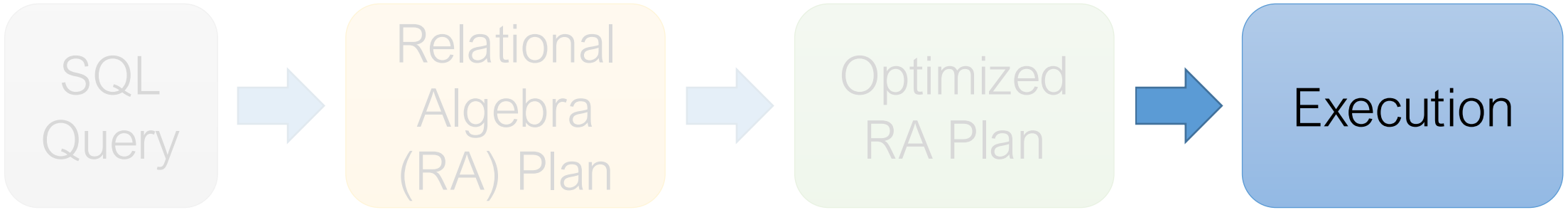
How does a SQL engine work ?



We saw how we can transform declarative SQL queries into precise, compositional RA plans

RDBMS Architecture

How is the RA “plan” executed?



RA Plan Execution

Natural Join / Join:

- Last lecture: how to use **memory & IO cost** considerations to pick the correct algorithm to execute a join with (BNLJ, SMJ, HJ…)!

Selection:

- We saw how to use **indexes to aid selection**
- Can always fall back on scan / binary search as well

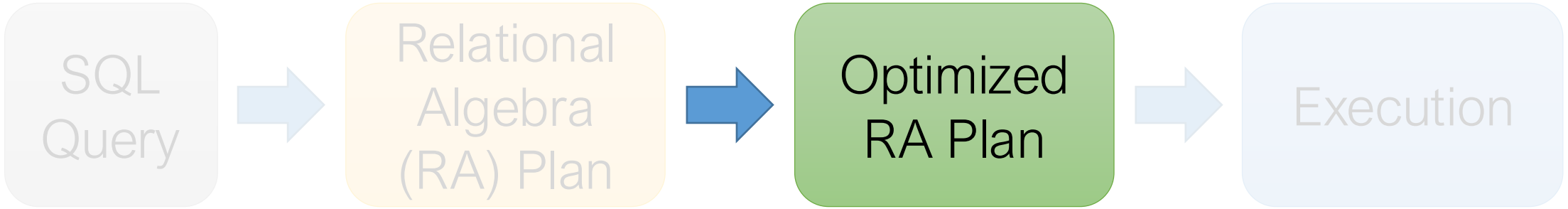
Projection:

- The main operation here is finding *distinct* values of the project tuples; we briefly discussed how to do this with e.g. **hashing** or **sorting**

We already know how to execute all the basic operators!

RDBMS Architecture

How does a SQL engine work ?



We'll look at how to then optimize these plans now

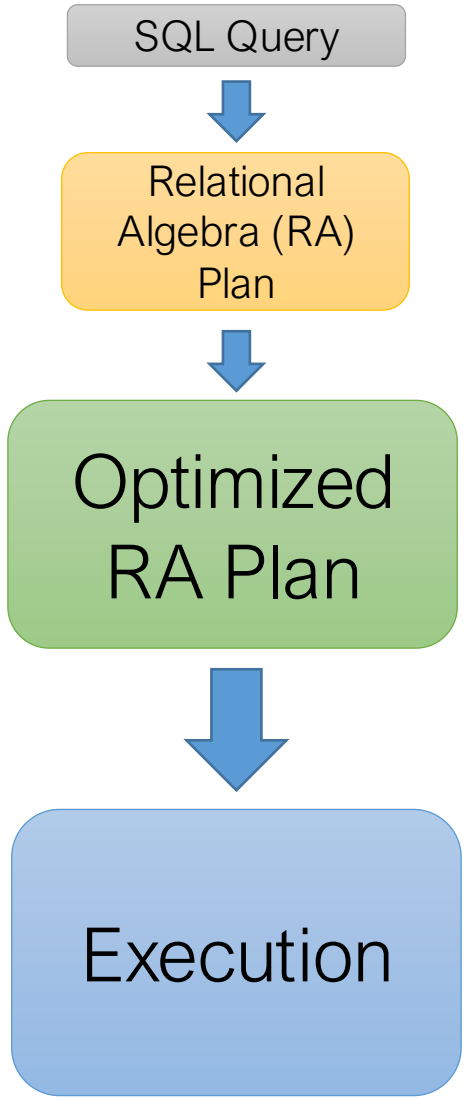
Logical vs. Physical Optimization

Logical optimization:

- Find equivalent plans that are more efficient
- *Intuition: Minimize # of tuples at each step by changing the order of RA operators*

Physical optimization:

- Find algorithm with lowest IO cost to execute our plan
- *Intuition: Calculate based on physical parameters (buffer size, etc.) and estimates of data size (histograms)*

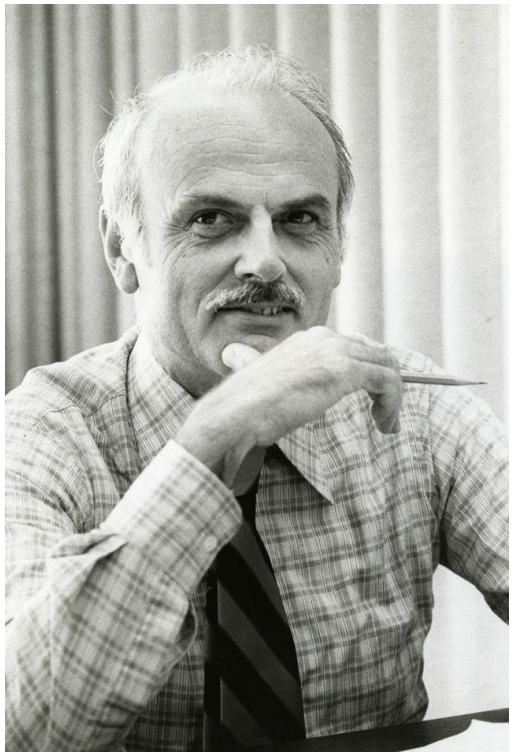


IBM System R

- First implementation of a query optimizer from the 1970s.
  - People argued that the DBMS could never choose a query plan better than what a human could write.

- Many concepts and design decisions from the System R optimizer are still used today.

Turing Award 1981



Codd

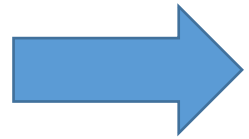
1. Logical Optimization

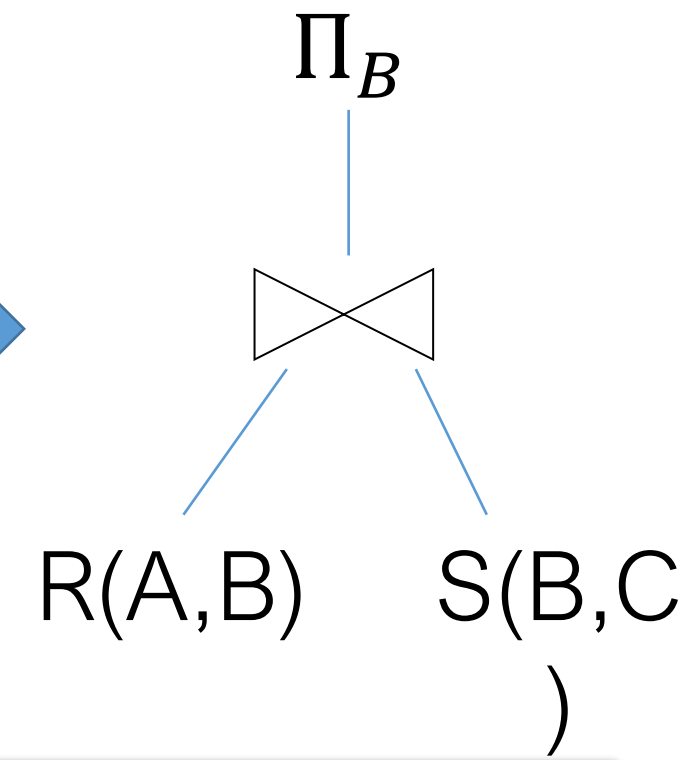
Relational Algebra Equivalences

- Two relational algebra expressions are equivalent if they generate the same set of tuples.
- The DBMS can identify better query plans using rules/heuristics.
- This is often called query rewriting

Recall: We can visualize the plan as a tree

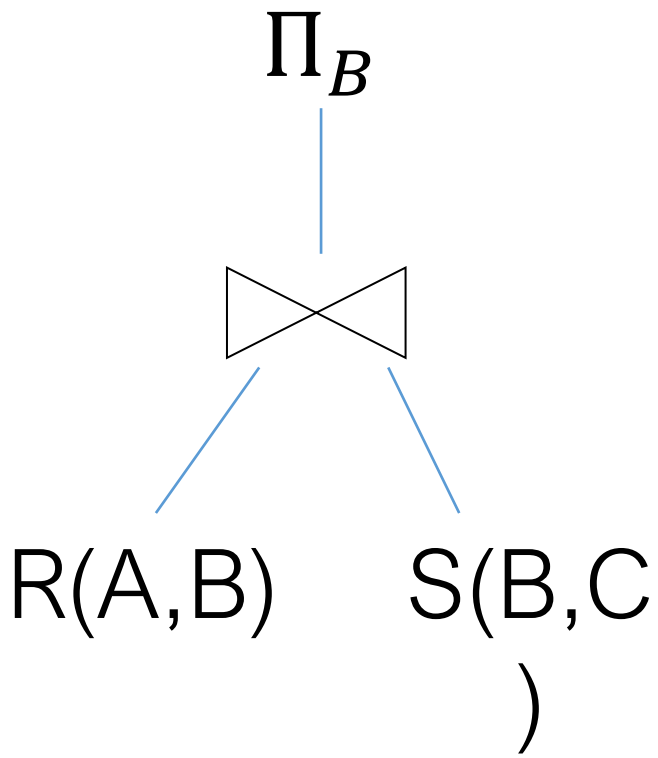
$$\Pi_B(R(A,B) \bowtie S(B,C))$$





Bottom-up tree traversal = order of operation execution!

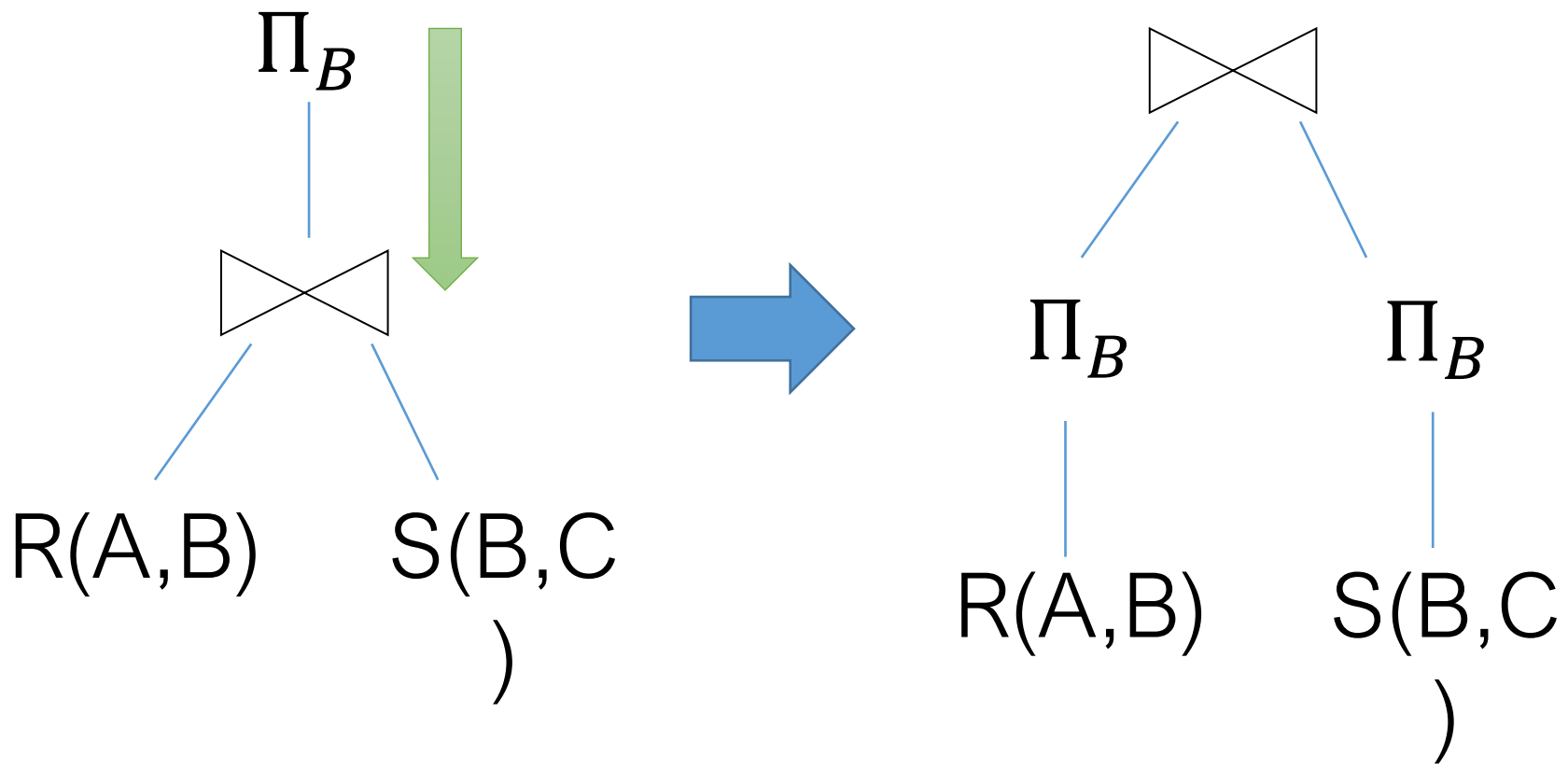
A simple plan



What SQL query does this correspond to?

Are there any logically equivalent RA expressions?

“Pushing down” projection



Why might we prefer this plan?

Logical Optimization

This process is called logical optimization

- Relational algebra is an important abstraction.

Heuristically, we want selections and projections to occur as early as possible in the plan

- Terminology: “push down **selections**” and “pushing down **projections.”**

Intuition: We will have fewer tuples in a plan.

- Could fail if the selection condition is very expensive.

Laws involving selection

Rule of thumb:

- Since selections tend to reduce the size of relations significantly, it usually helps to push the selections down the tree as far as they will go without changing what the expression does
- Break complex predicates and push down

$$\sigma_{p_1 \wedge p_2 \wedge \ldots p_n}(R) = \sigma_{p_1}\left(\sigma_{p_2}(\ldots \sigma_{p_n}(R)\right)$$

Example:

$$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$$

R has all attributes mentioned in C

$$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie \sigma_c(S)$$

R and S both have all attributes mentioned in C

Laws involving projection

Rule of thumb:

- Perform them early to create smaller tuples and reduce intermediate results (if duplicates are eliminated)
- Project out all attributes except the ones requested or required (e.g., joining keys)
- * This is not important for a column store

Example:

$$\pi_L\,(R \bowtie_C S) = \pi_L\,(\pi_M(R) \bowtie_C \pi_N(S))$$

M, N are join attributes mentioned in condition C

Commutative and associative laws

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

- Same holds for $\times$, $\cup$, $\cap$
- Holds for both set and multi-set semantics

Example ($R \bowtie S \bowtie T$):

- $|R| = 1{,}000$ tuples
- $|S| = 10{,}000$ tuples
- $|T| = 100$ tuples
- $|R \bowtie S| \approx 50{,}000$ tuples (many matches)
- $|S \bowtie T| \approx 500$ tuples (few matches)

Which join order is better?

$(R \bowtie S) \bowtie T$

$R \bowtie (S \bowtie T)$

Algebraic Laws for Improving Query Plans

Additional reading: Chapter 16.2

- Laws about joins and product
- Laws involving grouping and aggregations

Nested sub-queries:

- Rewrite to de-correlate and/or flatten them
- Decompose nested query and store result to temporary table

Note that this is not an exhaustive set of operations

- This covers *local re-writes; global re-writes possible but much harder*

This simple set of tools allows us to greatly improve the execution time of queries by optimizing RA plans!

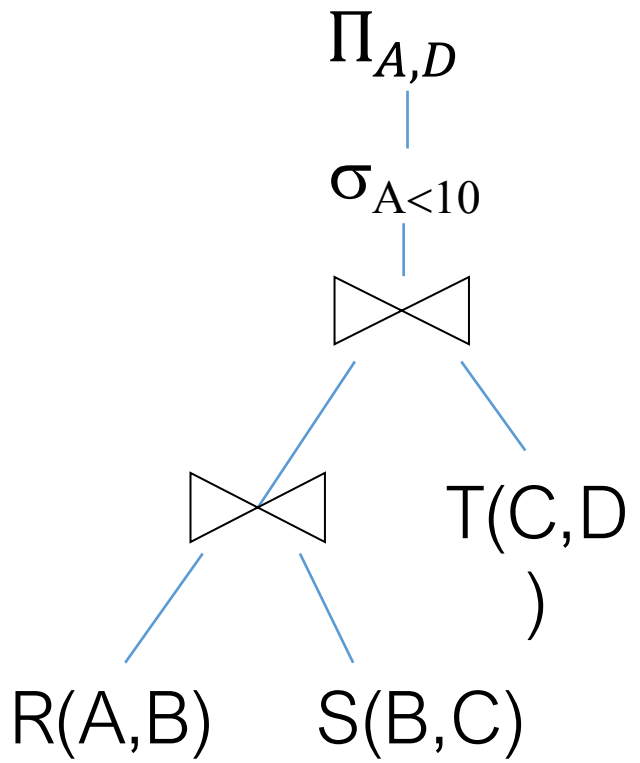
Example: Optimizing the SFW RA Plan

Translating to RA

R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

$$\Pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S)))$$



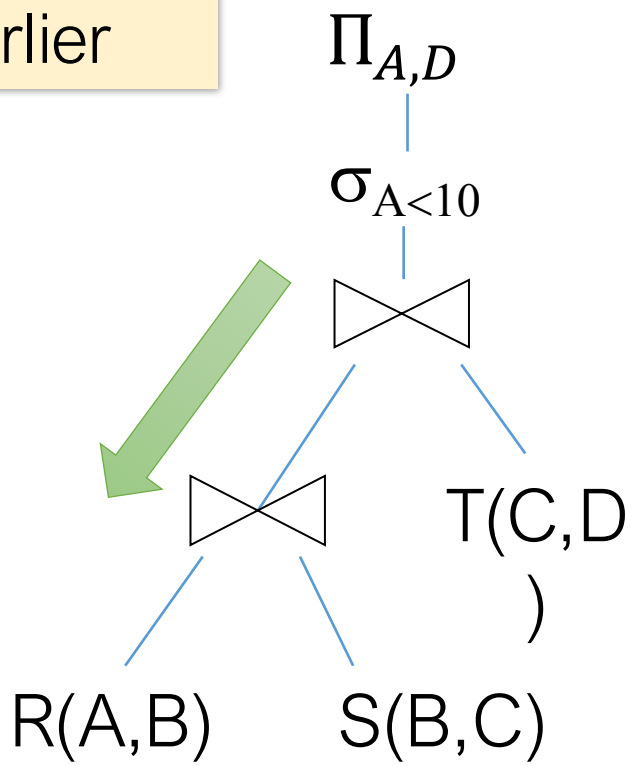
Optimizing RA Plan

R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

Push down selection on A so it occurs earlier

$$\Pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S)))$$



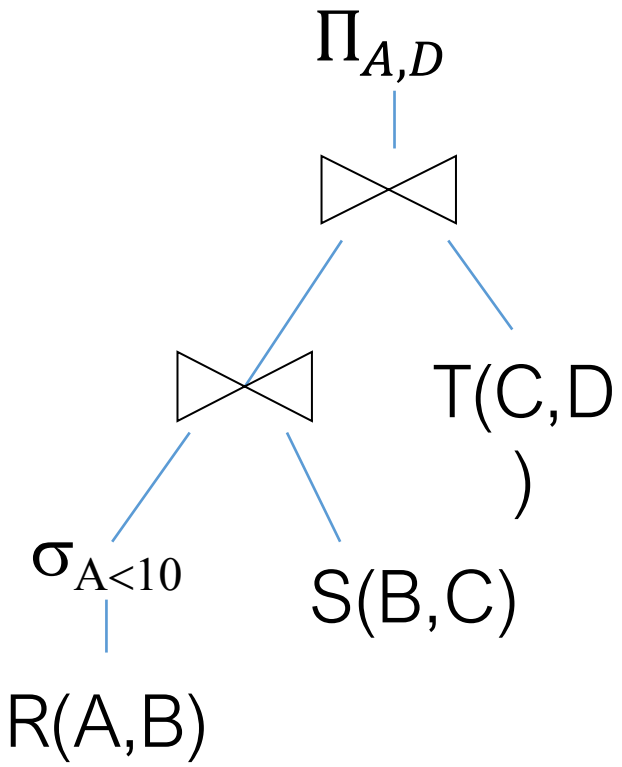
Optimizing RA Plan

R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

Push down selection on A so it occurs earlier

$$\Pi_{A,D}\left(T \bowtie \left(\sigma_{A<10}(R) \bowtie S\right)\right)$$



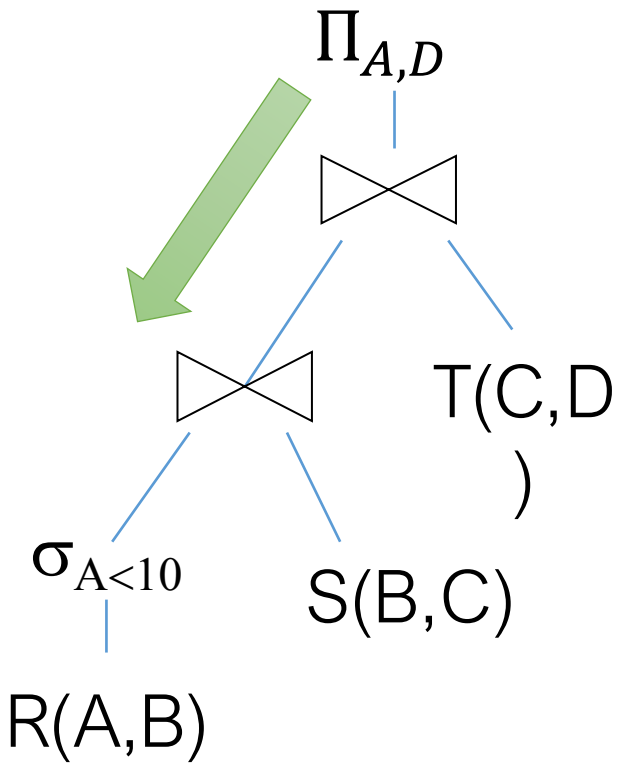
Optimizing RA Plan

R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

Push down projection so it occurs earlier

$$\Pi_{A,D}\left(T \bowtie \left(\sigma_{A<10}(R) \bowtie S\right)\right)$$



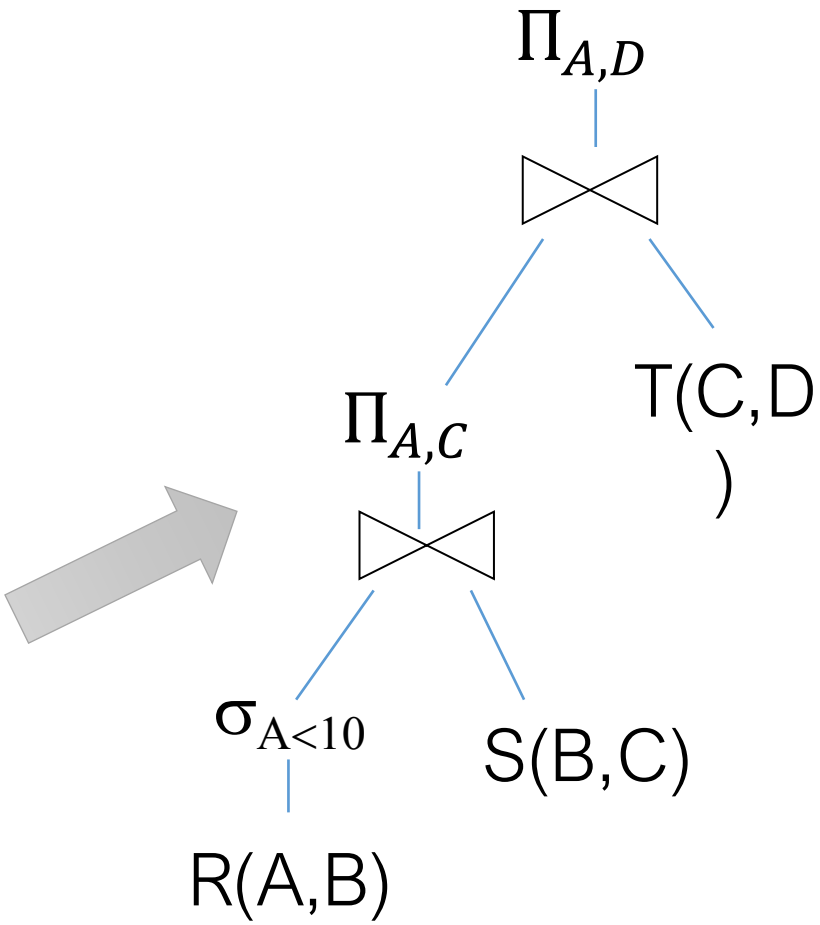
Optimizing RA Plan

We eliminate B earlier!

R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 10;
```

$$\Pi_{A,D}\left(T \bowtie \Pi_{A,C}(\sigma_{A<10}(R) \bowtie S)\right)$$

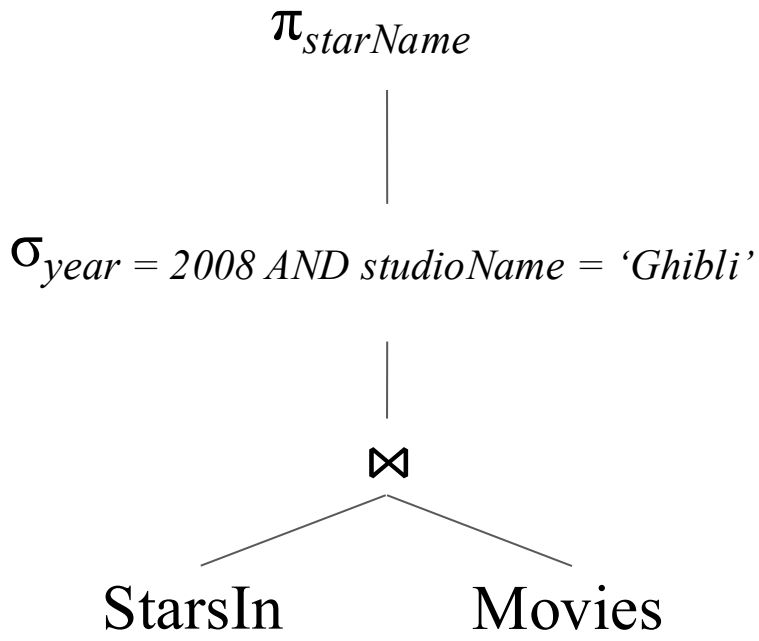


2. Physical Optimization

Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

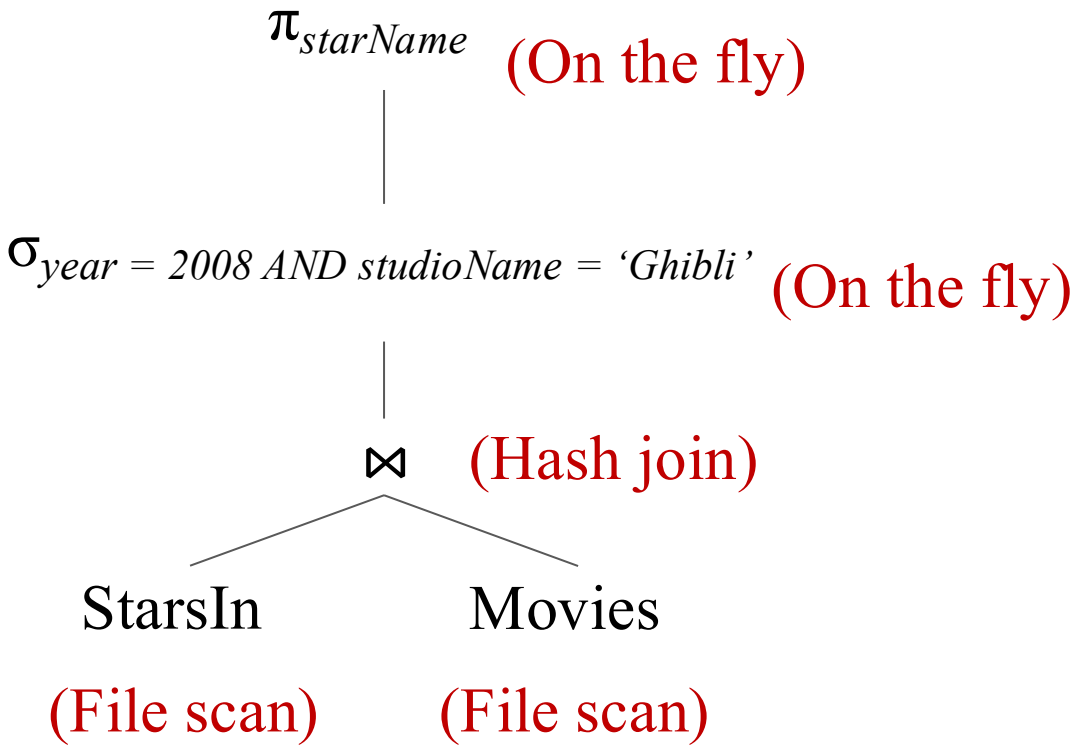


Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

Physical query plan 1

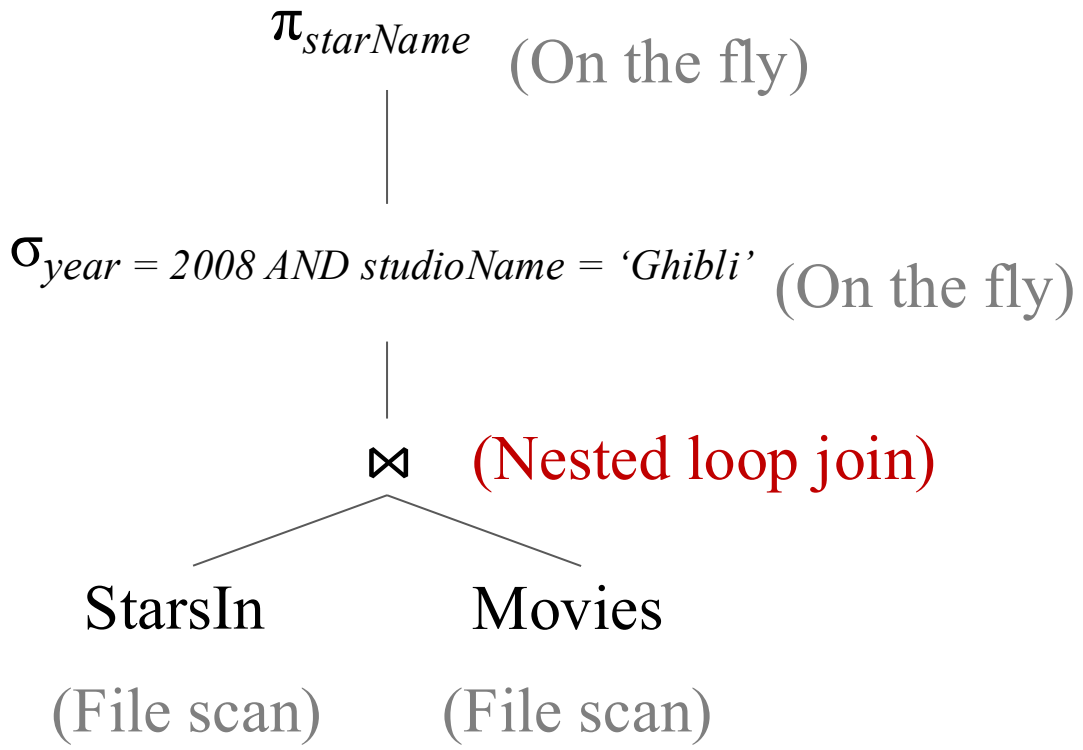


Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

Physical query plan 2

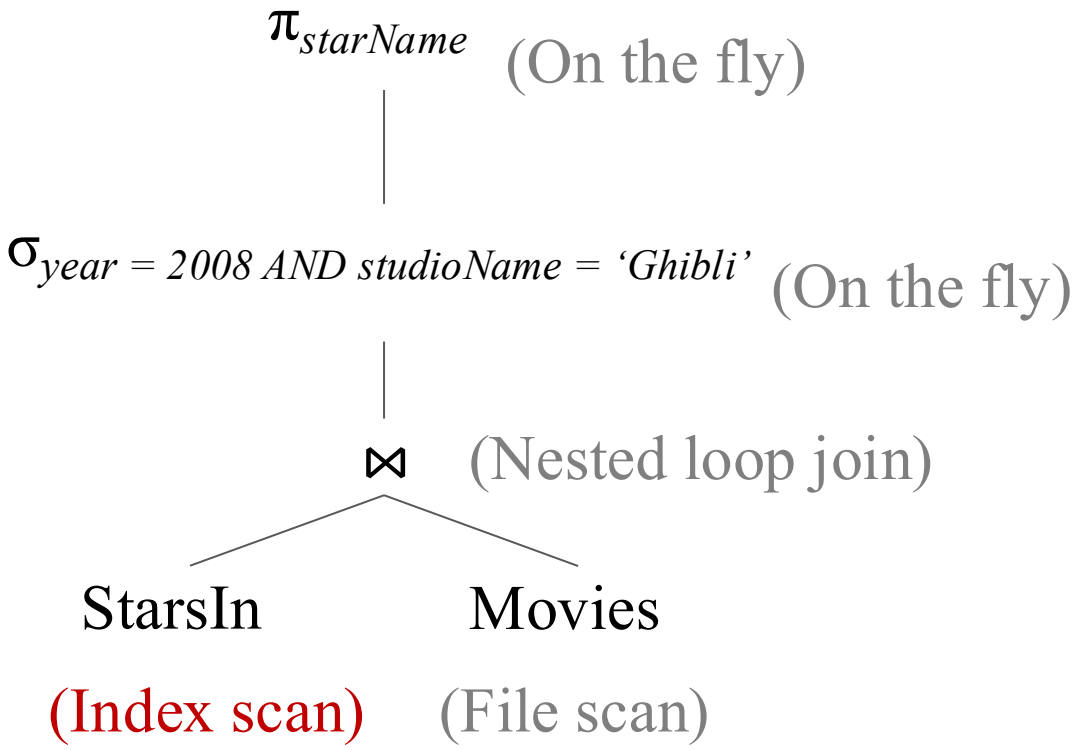


Select physical query plan

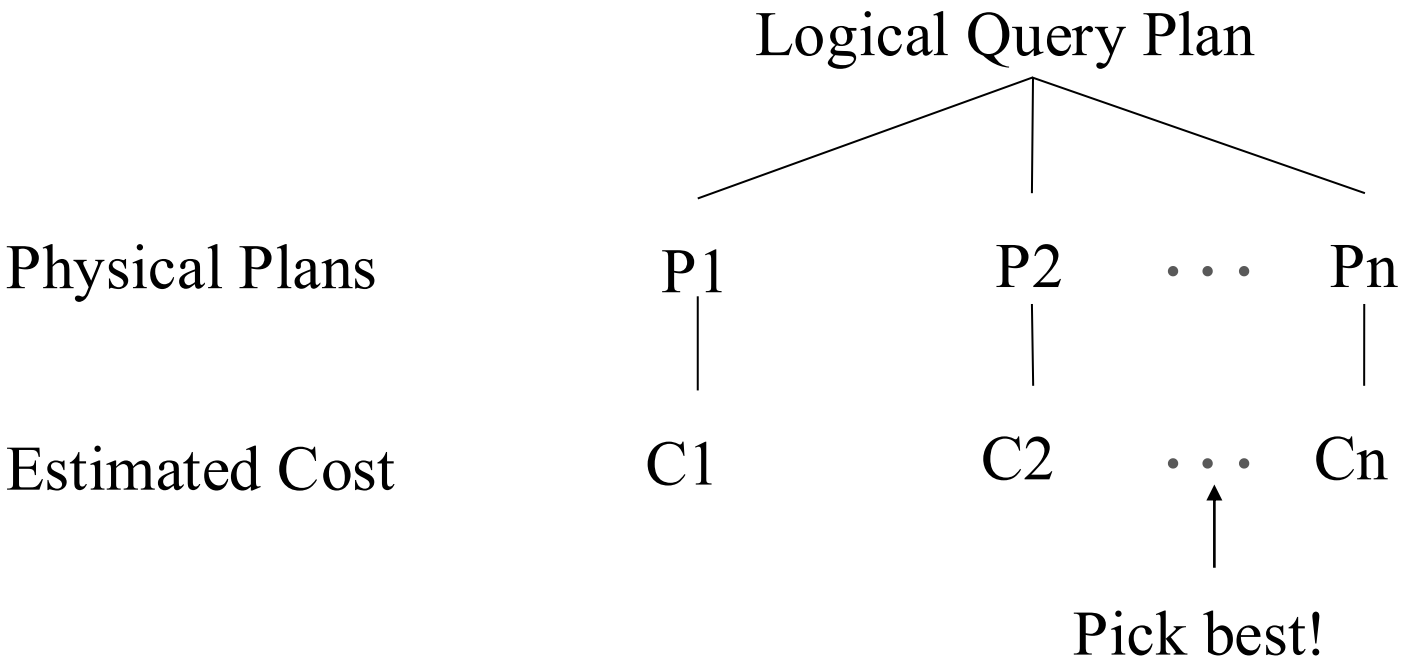
A logical query plan is turned into a physical query plan

- Algorithm for each operator
- Order of execution
- How to access relations

Physical query plan 3

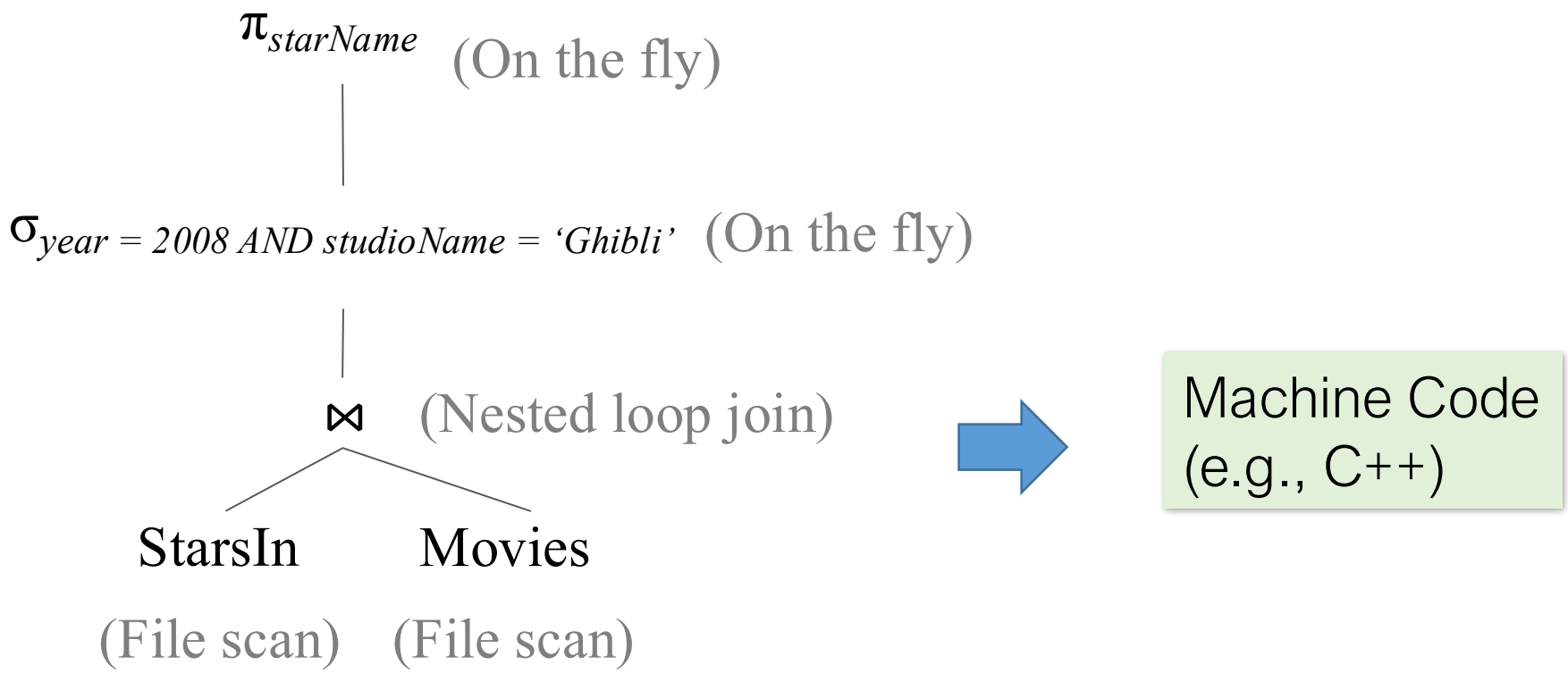


Select physical query plan



In general, there can be many possible physical plans

Query execution



The best physical plan is translated to actual machine code

3. Estimating cost of a physical plan

Estimating the cost of a physical query plan

Step 1: Estimate the size of results

- Projection
- Selection
- Joins

Step 2: Estimate the # of disk I/O's

We already know how to do step 2 for joins!

Notation: Size parameters

$P(R)$: # pages to hold tuples in R

$T(R)$: # tuples in R

$V(R, a)$: # distinct values of attribute a in R

Notation: Size parameters

Example:

R

A	B	C
cat	1	2000
cat	1	2001
dog	1	2002

A: 10 byte string
B: 4 byte integer
C: 8 byte date

T(R) = 3
V(R, A) = 2
V(R, B) = 1
V(R, C) = 3

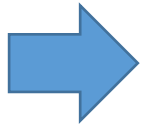
Suppose each page is 100 bytes
Then a page fits 4 tuples
If T(R) = 1000
Then P(R) = 1000 / 4 = 250

For π_A(R), each page fits 10 tuples, so
P(R) = 1000 / 10 = 100

Estimating size of selection

A selection generally reduces the number of tuples

$$S = \sigma_{A=c}(R)$$



Estimated result size
(without any additional information)

$$T(S) = \frac{T(R)}{V(R, A)}$$

*Assumption: values in A = c are uniformly distributed over possible V(R, A) values

Estimating size of selection

A selection generally reduces the number of tuples

Estimated result size
(without any additional information)

$$S = \sigma_{A<c}(R) \quad \Rightarrow \quad T(S) = \frac{T(R)}{3}$$

*Assumption: queries involving inequalities tend to retrieve a small fraction of possible tuples

Example: postgres/src/include/utils/selfuncs.h

Estimating size of selection

If selection condition is AND of conditions, multiply all selectivity factors

$$S = \sigma_{A=10 \wedge B<20}(R)$$

$$T(R) = 10{,}000$$
$$V(R, A) = 50$$

Q: What is T(S)?

$$T(S) = \frac{T(R)}{50 \times 3} = 67$$

Estimating size of selection

If selection condition is an OR of conditions, can assume independence of conditions

$$S = \sigma_{A=10 \vee B<20}(R)$$

$$T(R) = 10{,}000$$
$$V(R, A) = 50$$

Q: What is T(S)?

$$T(S) = \frac{T(R)}{1 - (1 - 1/50)(1 - 1/3)} = 3466$$

Estimating size of join

We study $R(X,Y) \bowtie S(Y,Z)$

Two simplifying assumptions

- Containment of value sets: if $V(R,Y) \leq V(S,Y)$, then every Y-value of R is a Y-value of S
- Preservation of value sets: $V(R \bowtie S, X) = V(R, X)$

Example when these assumptions are true:
Y is a key in S and the corresponding foreign key in R

Estimating size of join

$R(X,Y) \bowtie S(Y,Z)$

Two simplifying assumptions

- Containment of value sets: if $V(R,Y) \leq V(S,Y)$, then every Y-value of R is a Y-value of S
- Preservation of value sets: $V(R \bowtie S, X) = V(R, X)$

$\boldsymbol{Case\ 1}: V(R,Y) \geq V(S,Y)$
$\Rightarrow T(R \bowtie S) = T(R)T(S)/V(R,Y)$

$\boldsymbol{Case\ 2}: V(R,Y) < V(S,Y)$
$\Rightarrow T(R \bowtie S) = T(R)T(S)/V(S,Y)$

For each pair (r, s), we know that the Y-value of S is one of the Y-values of R by containment of value sets, so the probability of r having the same Y-value is 1/V(R,Y)

$$T(R \bowtie S) = T(R)T(S)/\max(V(R,Y), V(S,Y))$$

Joins of many relations

Compute intermediate *T*, *V* results
Example: *R* ⋈ *S* ⋈ *T*

$R(A, B)$ $S(B, C)$ $T(C, D)$

$T(R) = 1000$
$V(R, B) = 20$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 100$

$T(T) = 5000$
$V(T, C) = 500$
$V(T, D) = 200$

Q: What is T(R ⋈ S) and V(R ⋈ S, C)?

Joins of many relations

Compute intermediate T, V results
Example: $R \bowtie S \bowtie T$

$R(A, B)$

$T(R) = 1000$
$V(R, B) = 20$

$S(B, C)$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 100$

$R \bowtie S(A, B, C)$

$T(R \bowtie S) = T(R)\ T(S)\ /$
$\max(V(R, B), V(S, B)) = \ 40000$

$V(R \bowtie S, C) = 100$

Joins of many relations

Compute intermediate T, V results
Example: $R \bowtie S \bowtie T$

$R \bowtie S(A, B, C)$

$T(R \bowtie S) = 40000$
$V(R \bowtie S, C) = 100$

$T(C, D)$

$T(T) = 5000$
$V(T, C) = 500$
$V(T, D) = 200$

$(R \bowtie S) \bowtie T$

$T((R \bowtie S) \bowtie T)$
$= 40000 \text{ x } 5000 \text{ / } \max\{100, 500\}$
$= 400000$

Joins of many relations

Compute intermediate *T*, *V* results
Example: consider $R \bowtie S \bowtie T$

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

$$T(R \bowtie (S \bowtie T)) = 1000 \text{ x } (2000 \text{ x } 5000 \text{ / } \max\{100, 500\}) \text{ / } \max\{20, 50\}$$
$$= 400000$$

Assuming containment and preservation of value sets, the estimated result size is the same regardless of how we group and order the terms in a natural join of relations.

Natural joins with multiple join attributes

Same as R ⋈ S with single join attribute, but divide by max{V(R, A), V(S, A)} for each joining attribute A

$R(A, B, C)$

$T(R) = 1000$
$V(R, B) = 20$
$V(R, C) = 100$

$S(B, C, D)$

$T(S) = 2000$
$V(S, B) = 50$
$V(S, C) = 50$

$R \bowtie S$

$T(R \bowtie S) = 1000 \text{ x } 2000$
$/ \max\{20, 50\}$
$/ \max\{100, 50\}$
$= 400$

Further reading

- Using similar ideas, can estimate sizes of other operations like union, intersect, difference, duplicate elimination, grouping

- Chapter 16.4.7