CS 6400 A

Database Systems Concepts and Design

Lecture 15

10/21/26

Announcements

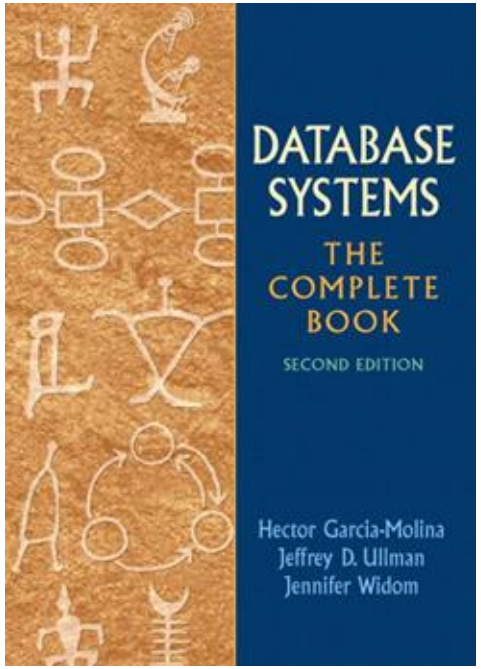
Project Milestone has been released

- Due: Nov 4
- What we are expecting:
  - Progress check in
  - Preliminary results
  - Methodology update
  - Risk and mitigation

Reading Materials

Query optimization (Chapters 16.1 - 16.3, 16.7)

- Parsing
- Algebraic laws
- Parse tree -> logical query plan
- Completing the Physical-Query-Plan



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

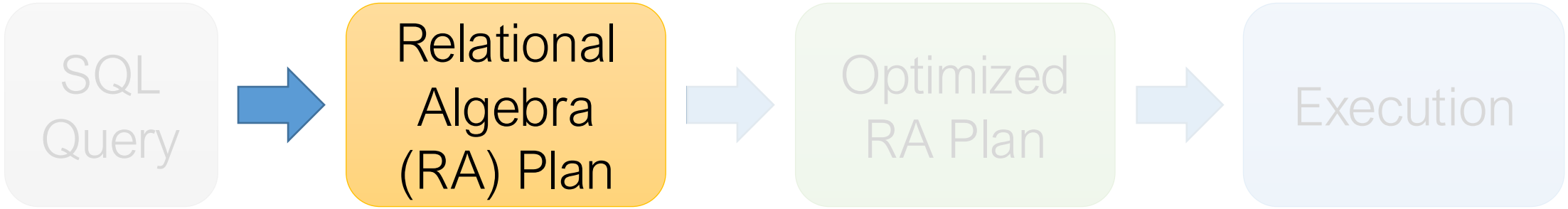
Agenda

1. Logical Optimization

2. Physical Optimization

Recap: RDBMS Architecture

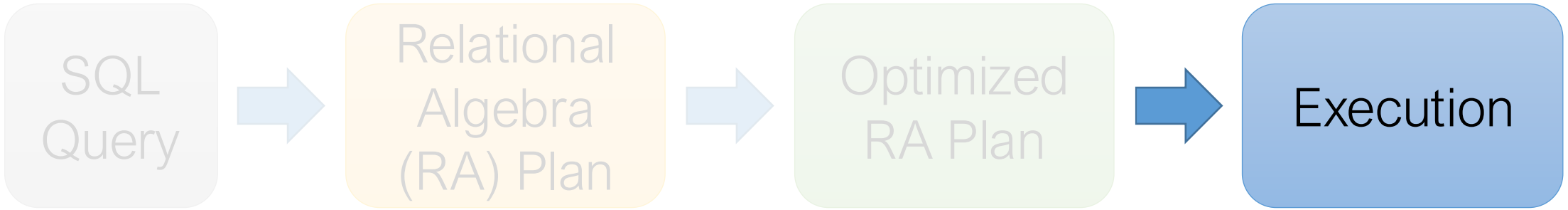
How does a SQL engine work ?



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

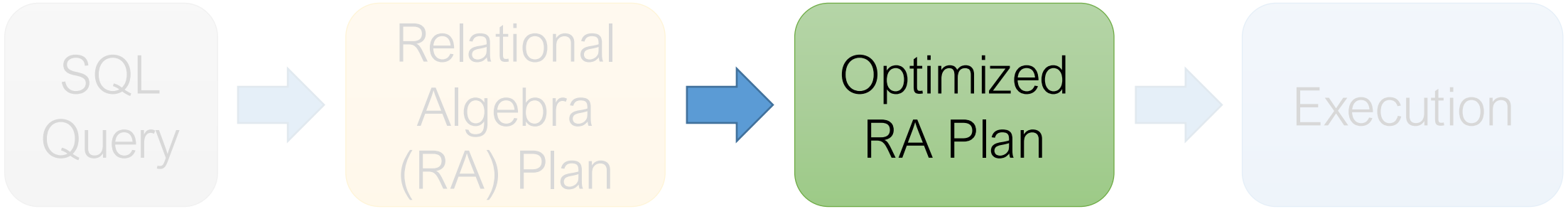
Projection:

- 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

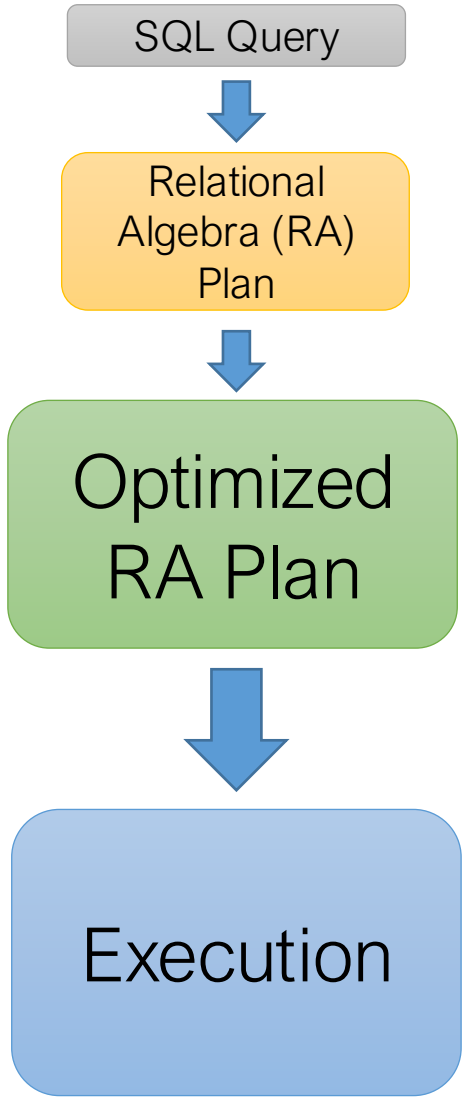
Logical vs. Physical Optimization

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)*



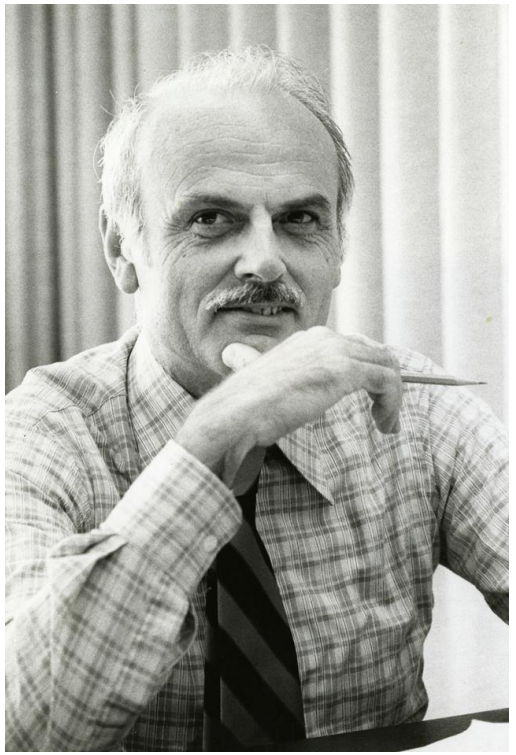
IBM System R

- 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

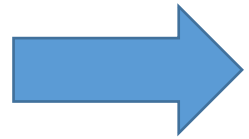
1. Logical Optimization

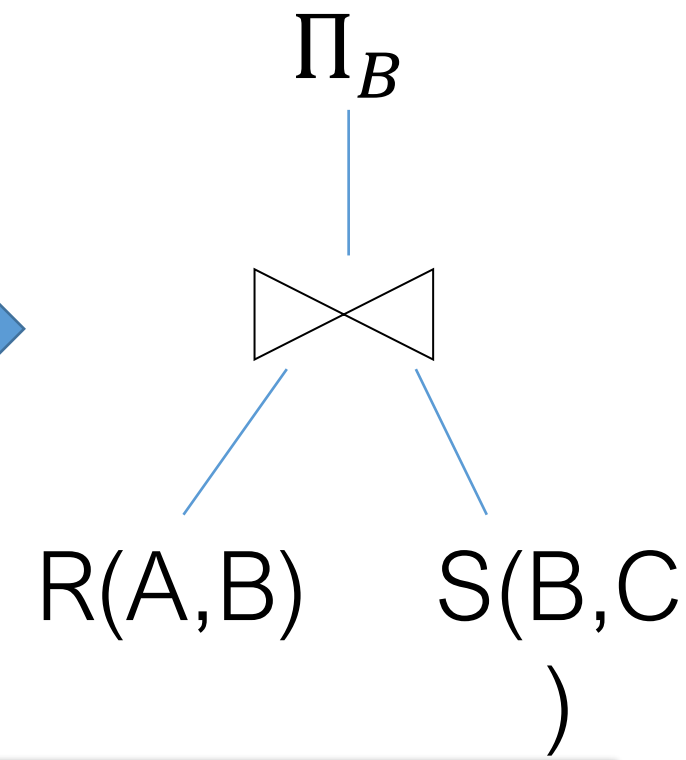
Relational Algebra Equivalences

- 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

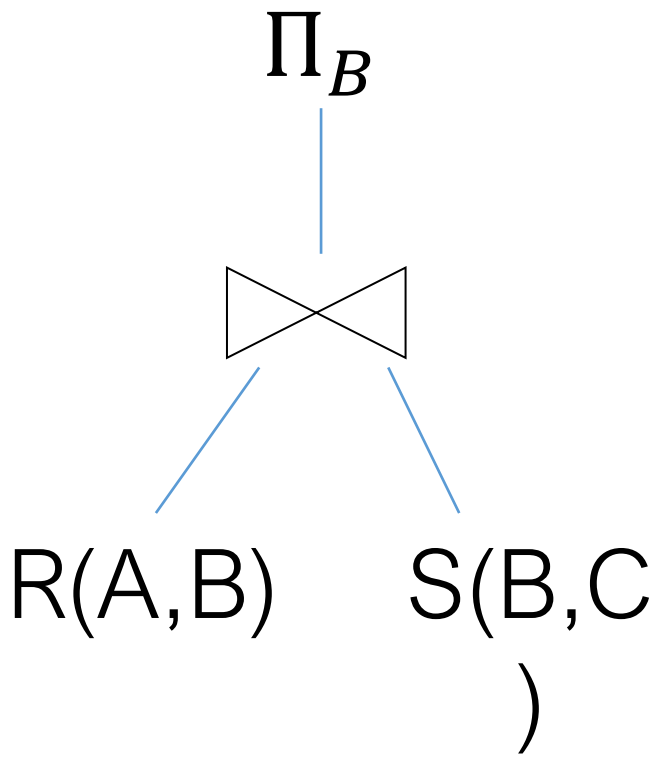
$$\Pi_B(R(A,B) \bowtie S(B,C))$$





Bottom-up tree traversal = order of operation execution!

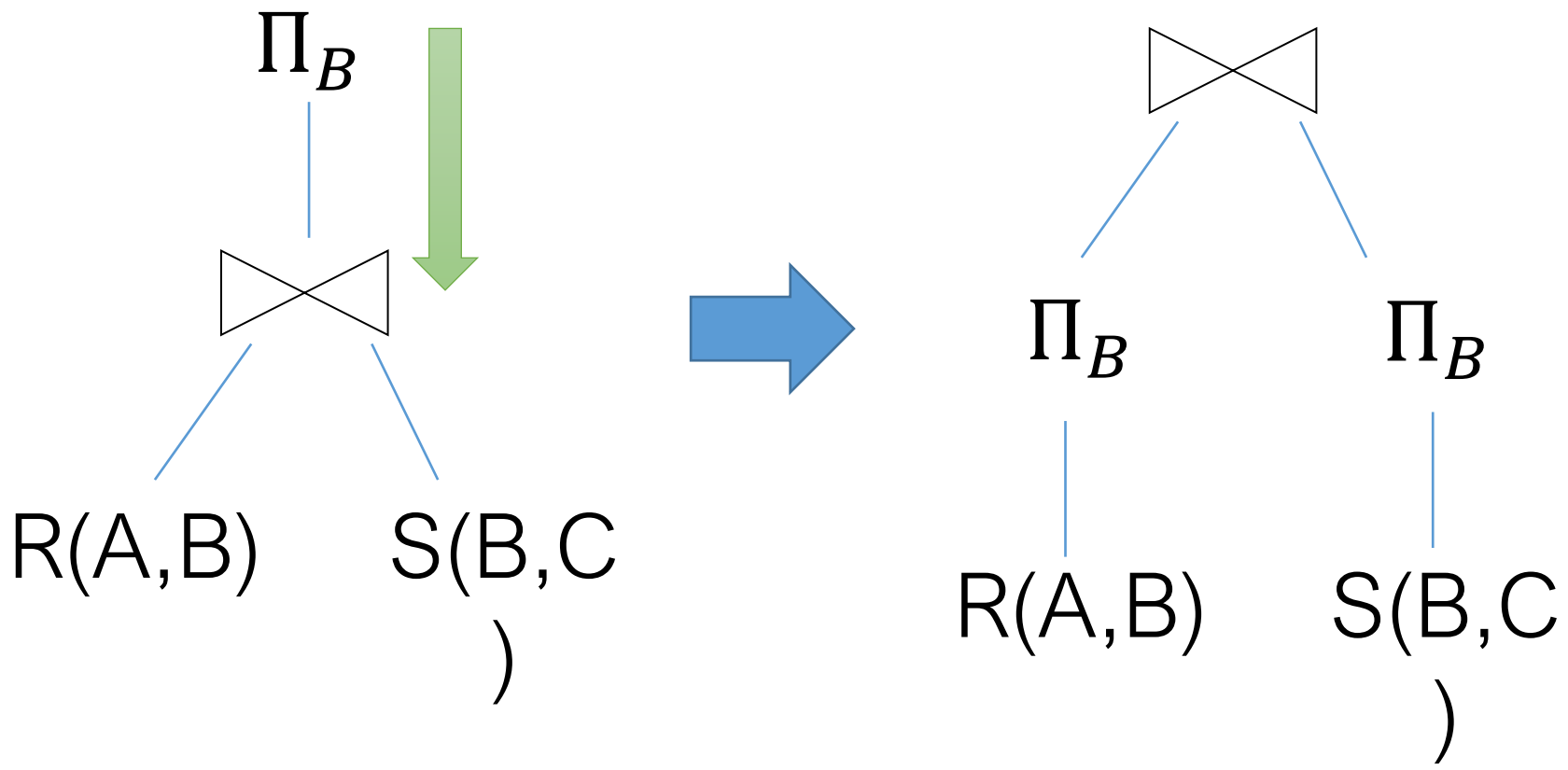
A simple plan



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!

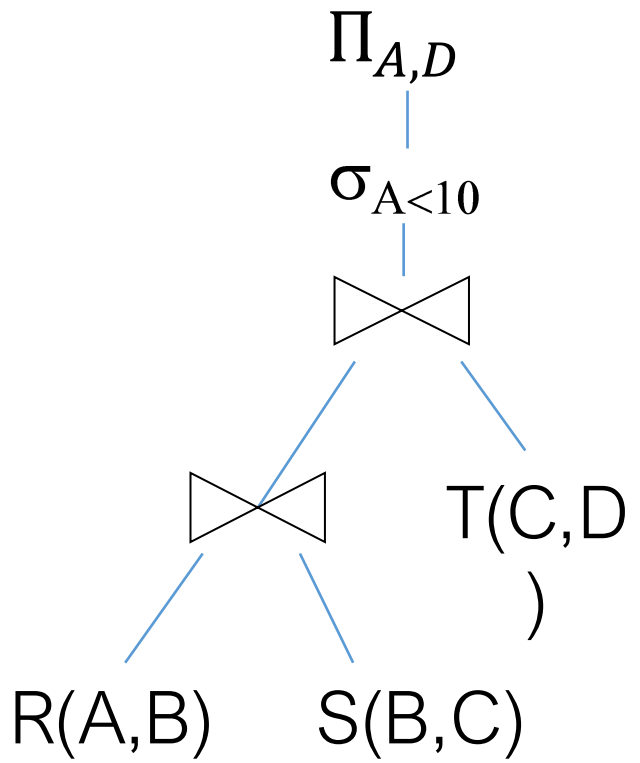
Example: Optimizing the SFW RA Plan

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;
```

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



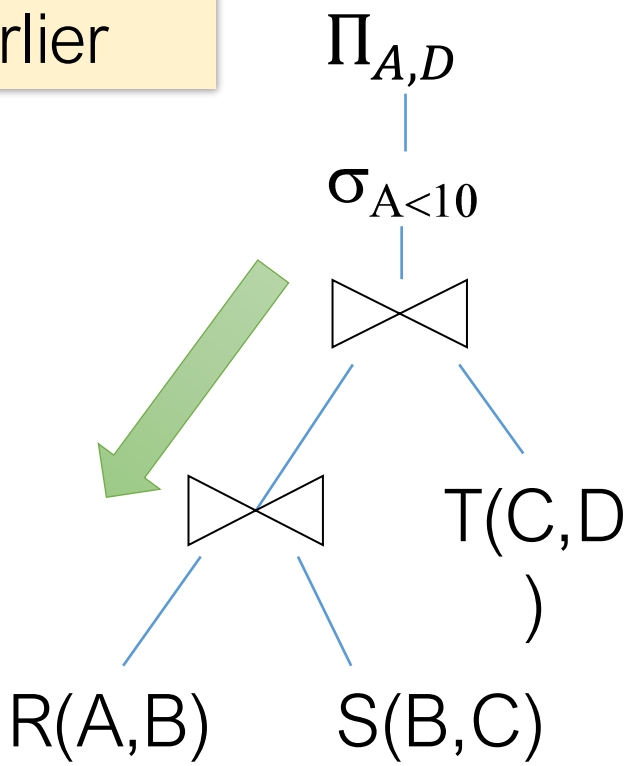
Optimizing RA Plan

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

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



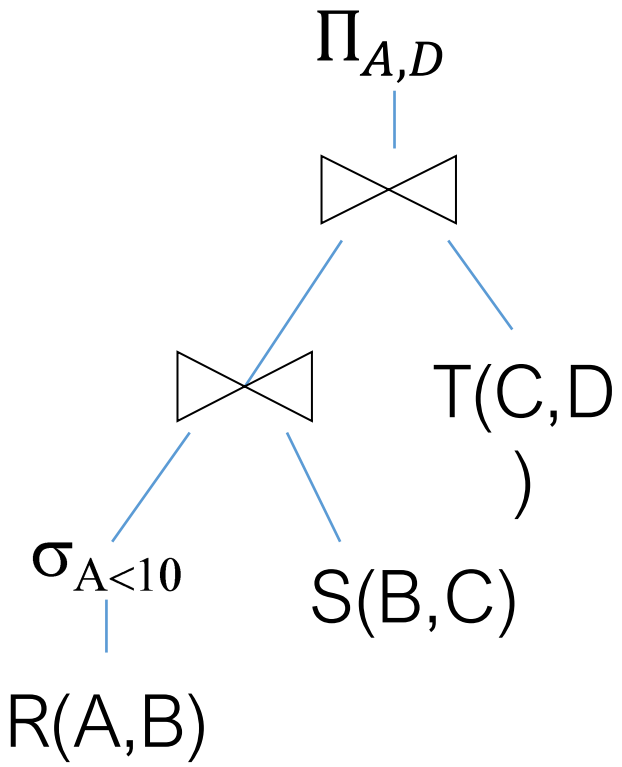
Optimizing RA Plan

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

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



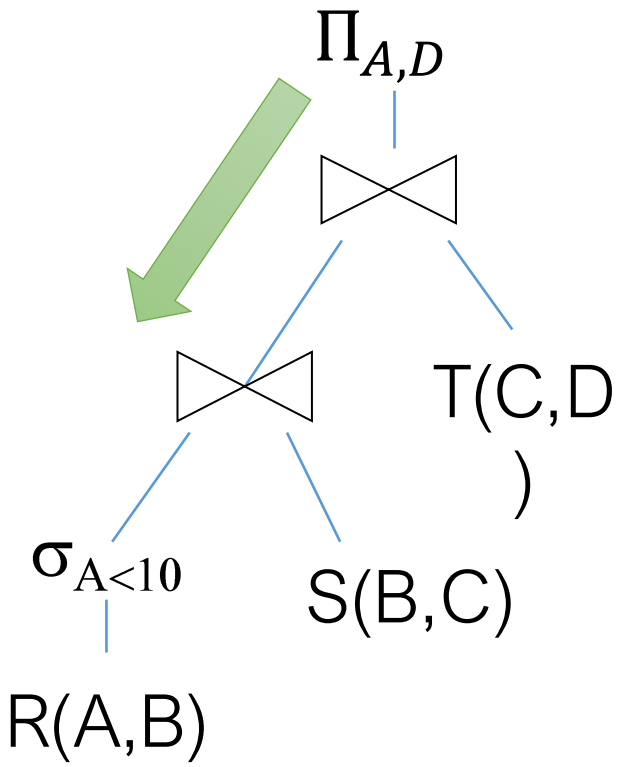
Optimizing RA Plan

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

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



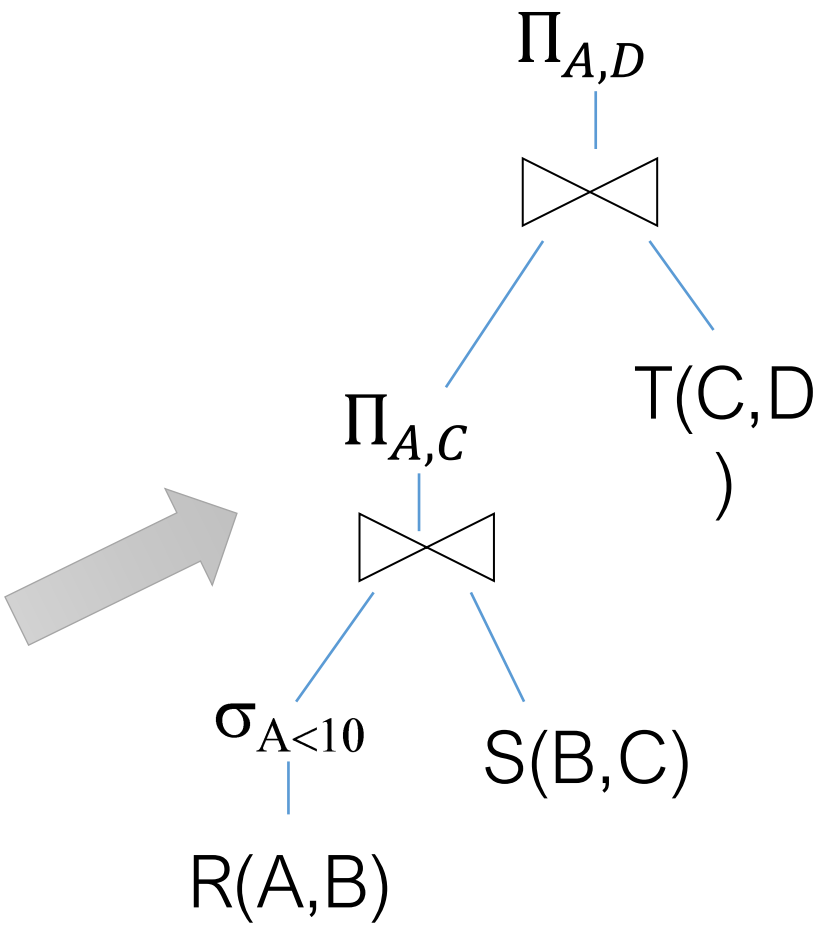
Optimizing RA Plan

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;
```

We eliminate B earlier!

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

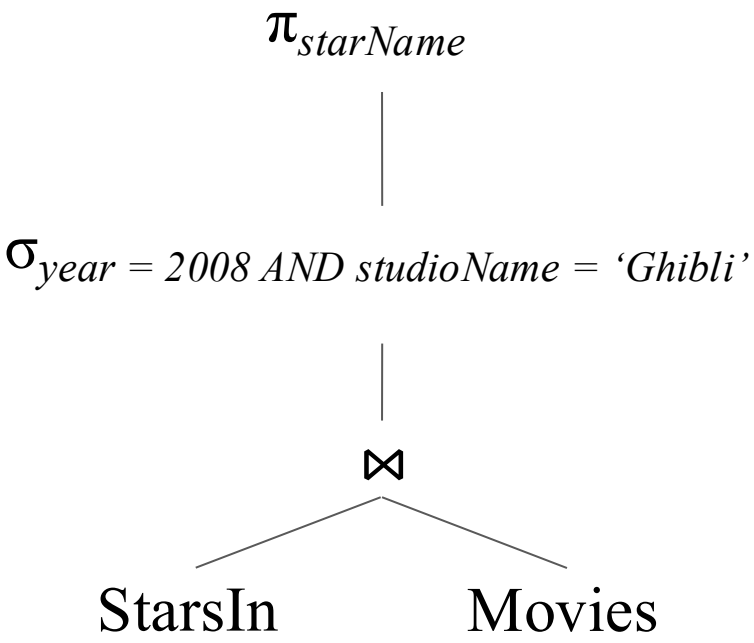


2. Physical Optimization

Select physical query plan

A logical query plan is turned into a physical query plan by choosing

- Algorithm for each operator
- How to access relations

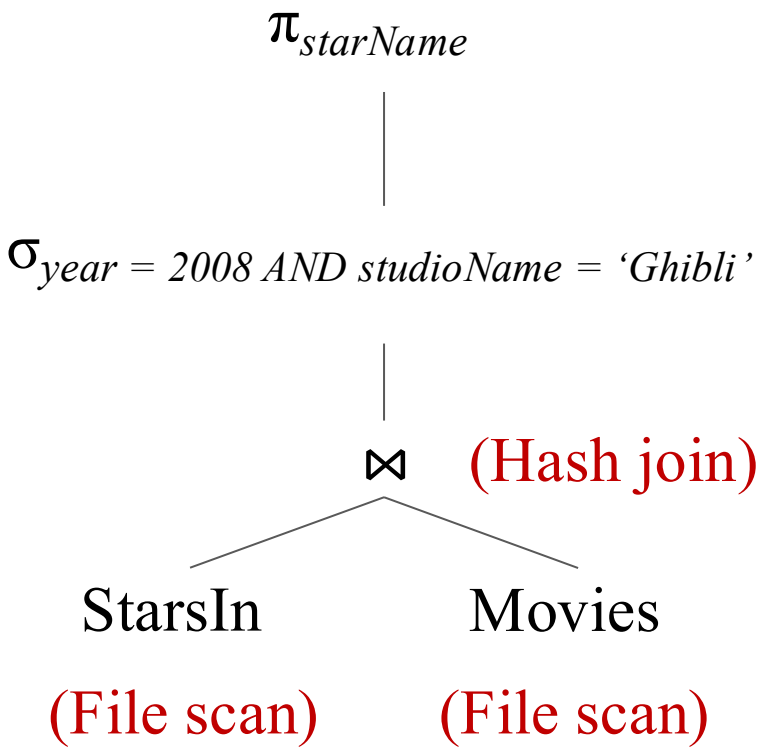


Select physical query plan

A logical query plan is turned into a physical query plan by choosing

- Algorithm for each operator
- How to access relations

Physical query plan 1

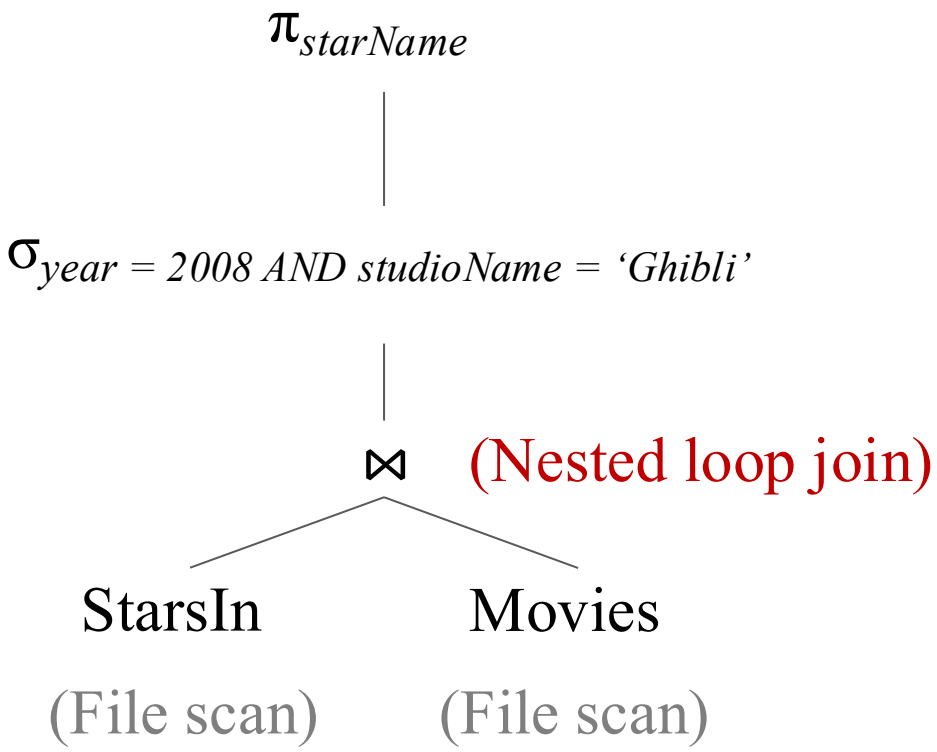


Select physical query plan

A logical query plan is turned into a physical query plan

- Algorithm for each operator
- How to access relations

Physical query plan 2

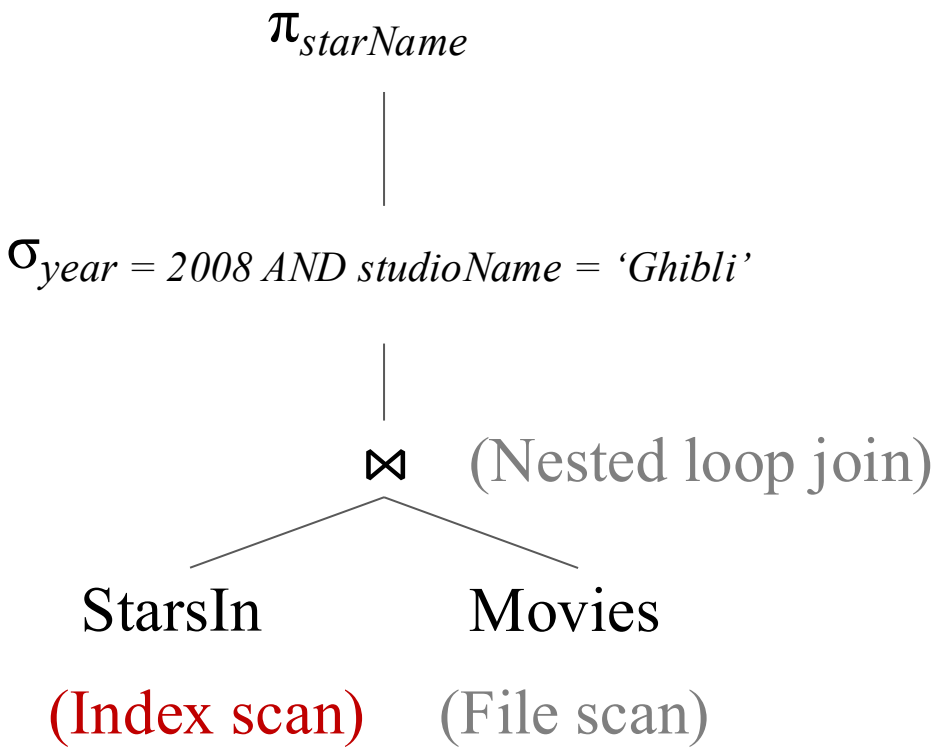


Select physical query plan

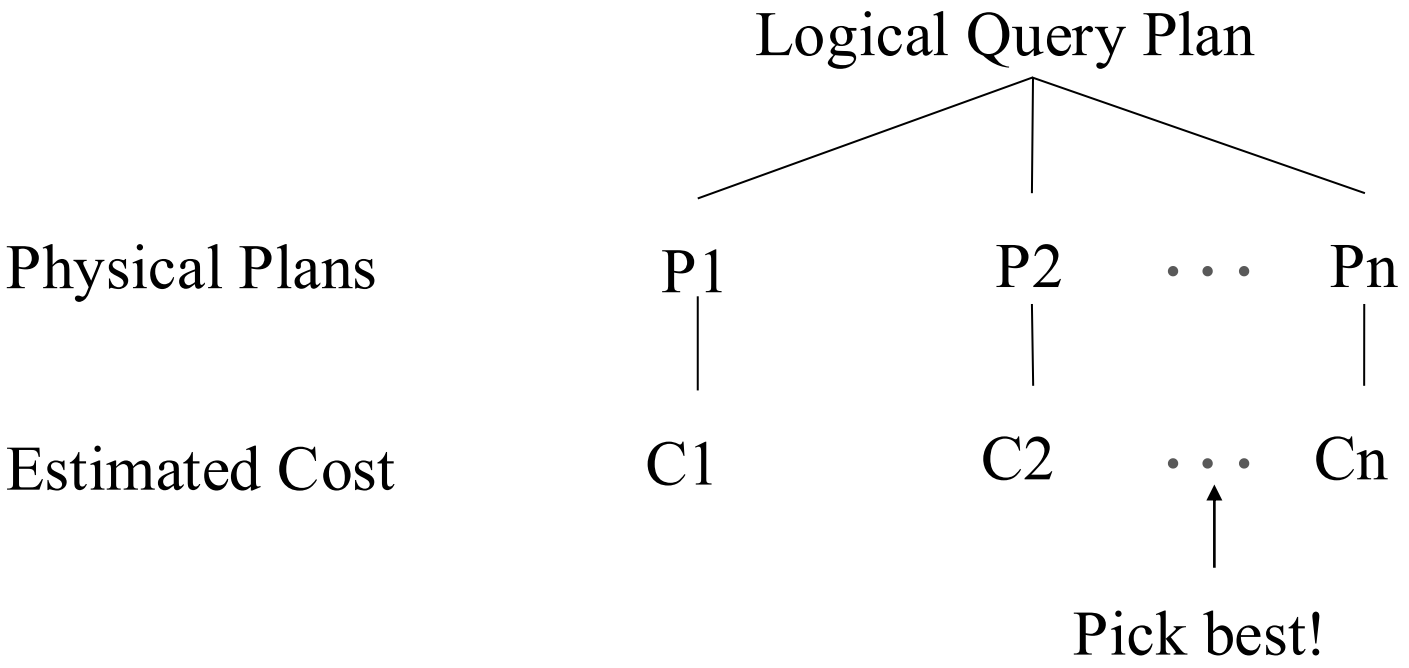
A logical query plan is turned into a physical query plan

- Algorithm for each operator
- How to access relations

Physical query plan 3

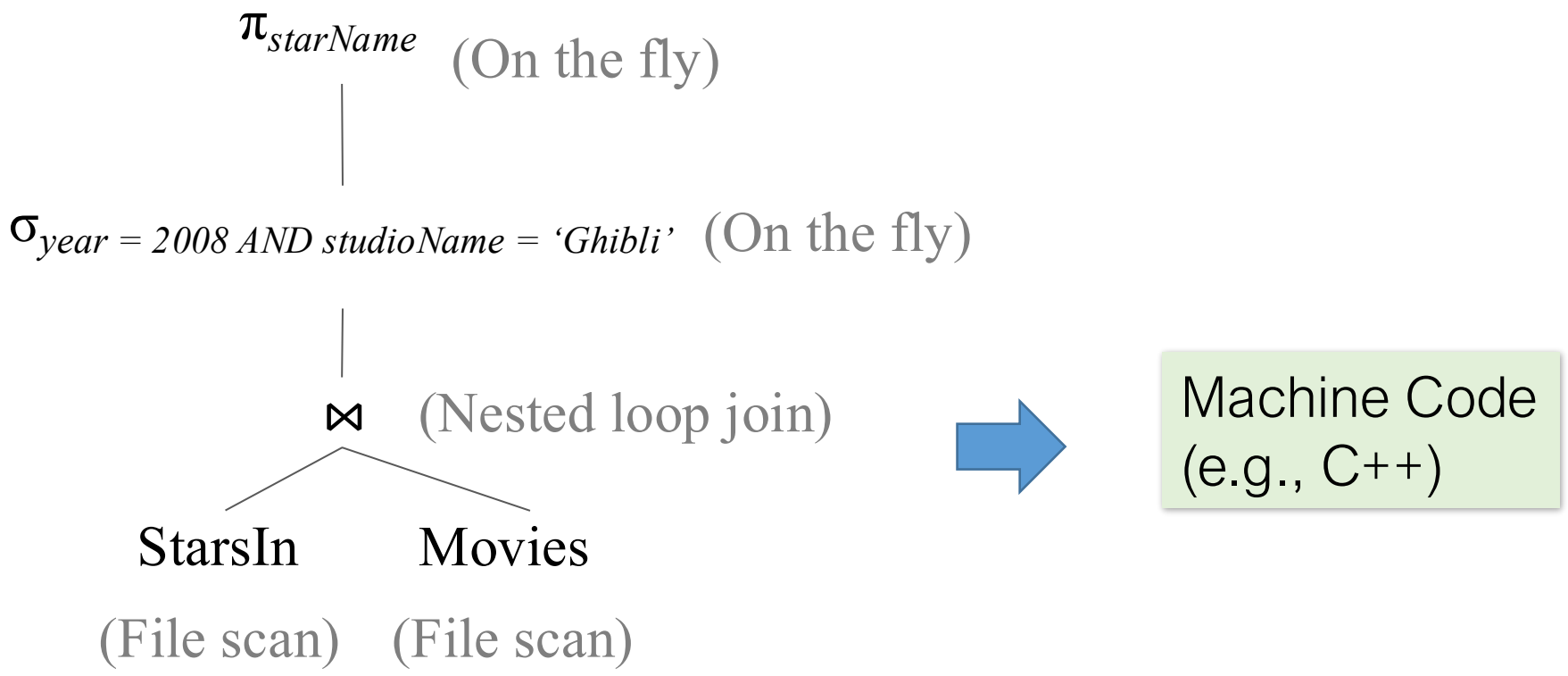


Select physical query plan



In general, there can be many possible physical plans

Query Execution: Running the Physical Plan



The execution engine runs the selected physical query plan.

Query Execution Model

Determines how operators execute and exchange data.

Iterator Model (pull-based)

- Parent operators request tuples from their children, one at a time.
- Intermediate results can be pipelined without being fully stored.

Materialization Model (intermediate results stored)

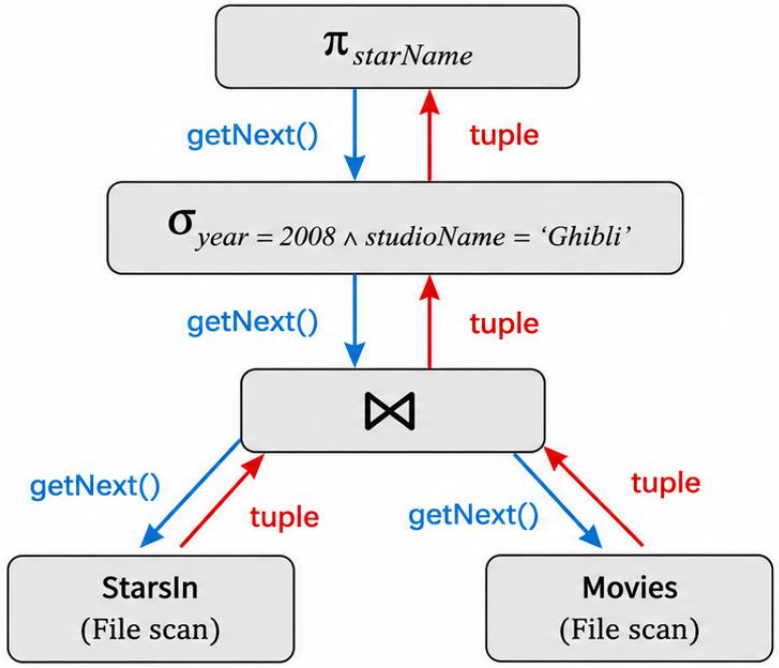
- Each operator produces a complete intermediate result.
- The next operator processes that result after it is available.

The physical plan is chosen by the optimizer; the execution model is largely determined by the DBMS architecture.

Query Execution Model

More details later

Iterator Model



Materialization Model

