QUERY OPTIMIZATION

CS 564- Fall 2021

ACKs: *Jeff Naughton, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT?

- What is a query optimizer?
- Generating query plans
- Cost estimation of query plans

ARCHITECTURE OF AN OPTIMIZER

query (**SQL**)

Query Parser

parsed query (**Relational Algebra**)

Query Optimizer
- Plan generator
- Plan cost estimator

System Catalog

Relational Algebra is the glue!

query plan (**annotated RA tree**)

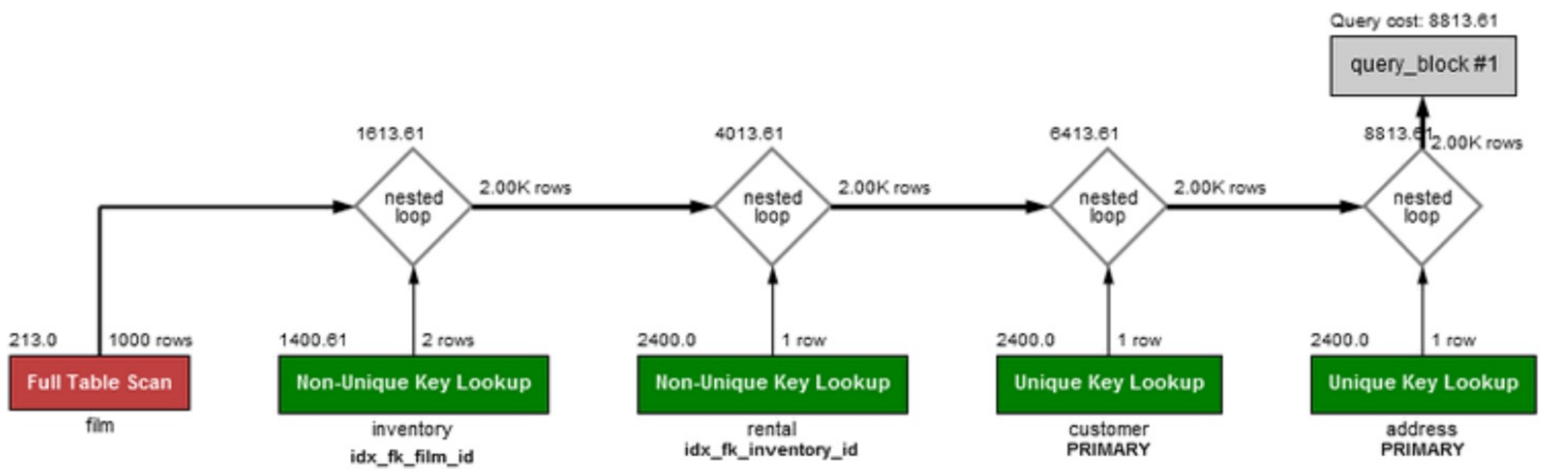
QUERY PLAN

A **query plan** specifies:

- What algorithm to choose for each operator
- How the execution of the operators is coordinated

REAL-WORLD EXAMPLE

```
SELECT CONCAT(customer.last_name, ', ', customer.first_name) AS customer,
        address.phone, film.title
FROM rental
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
WHERE rental.return_date IS NULL
AND rental_date + INTERVAL film.rental_duration DAY < CURRENT_DATE() LIMIT 5;
```



QUERY OPTIMIZATION: BASICS

The query optimizer

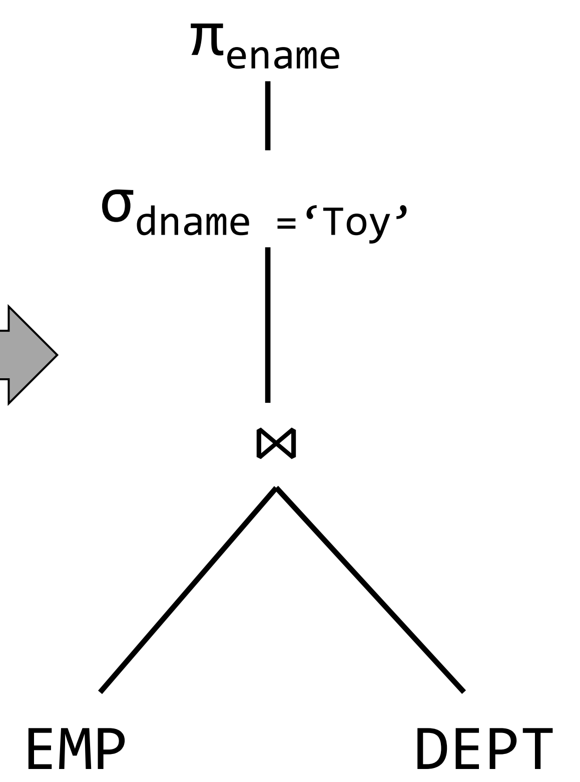
1. searches through the space of equivalent query plans
2. chooses the best overall plan by estimating the I/O cost of each plan

EXAMPLE: FROM SQL TO RA

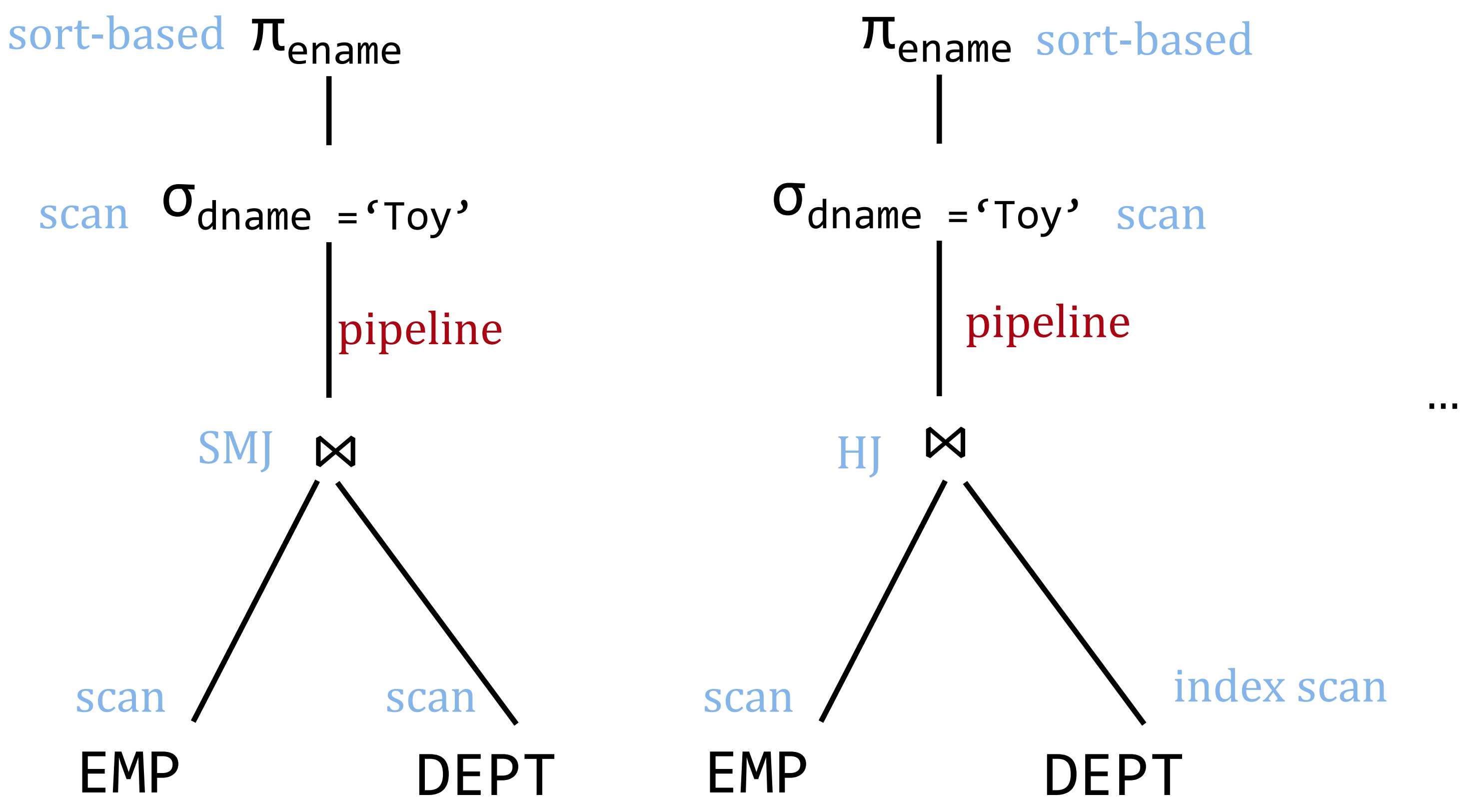
EMP(ssn, ename, addr, sal, did)

DEPT(did, dname, floor, mgr)

```
SELECT DISTINCT ename
FROM   Emp E, Dept D
WHERE  E.did = D.did
AND    D.dname = 'Toy' ;
```



EXAMPLE: QUERY PLANS



GENERATING QUERY PLANS

1 - QUERY BLOCKS

The first step is to split a SQL query into a collection of **query blocks**:

- No nesting inside a block
- One `SELECT` and `FROM` clause
- At most one `WHERE`,`GROUP BY`,`HAVING` clause

The optimizer independently figures out the best query plan for each block

QUERY BLOCK: EXAMPLE

```
SELECT C.Name
FROM Country C
WHERE C.code =
   (SELECT C.CountryCode
    FROM City C
    WHERE C.name = 'Berlin');
```

query block

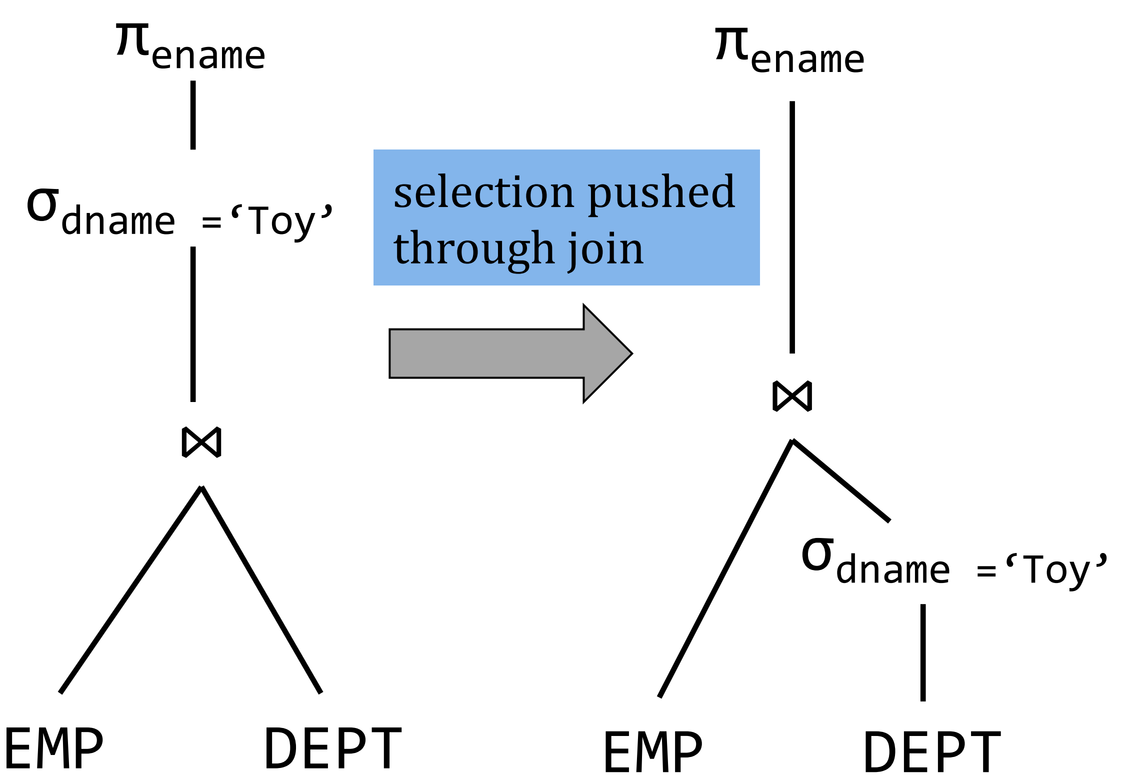
2 – GENERATING RA EXPRESSIONS

- The space of possible query plans is huge and it is hard to navigate
- Relational Algebra provides us with rules that transform one RA expression to an equivalent one
  - push down selections & projections
  - join reordering
- These transformations allow us to construct many alternative RA expressions

PUSHING DOWN SELECTIONS

A selection can be pushed down through

- projections
- joins
- other selections



PUSHING DOWN SELECTIONS

$\sigma_{A=10}$

$\pi_{A,B}$

R(A,B,C)

selection pushed through projection

$\pi_{A,B}$

$\sigma_{A=10}$

R(A,B,C)

SELECTION REORDERING

It is always possible to change the order of selections

$\sigma_{A=10}$

$\sigma_{B>0}$

R(A,B,C)

selection pushed through another selection

$\sigma_{B>0}$

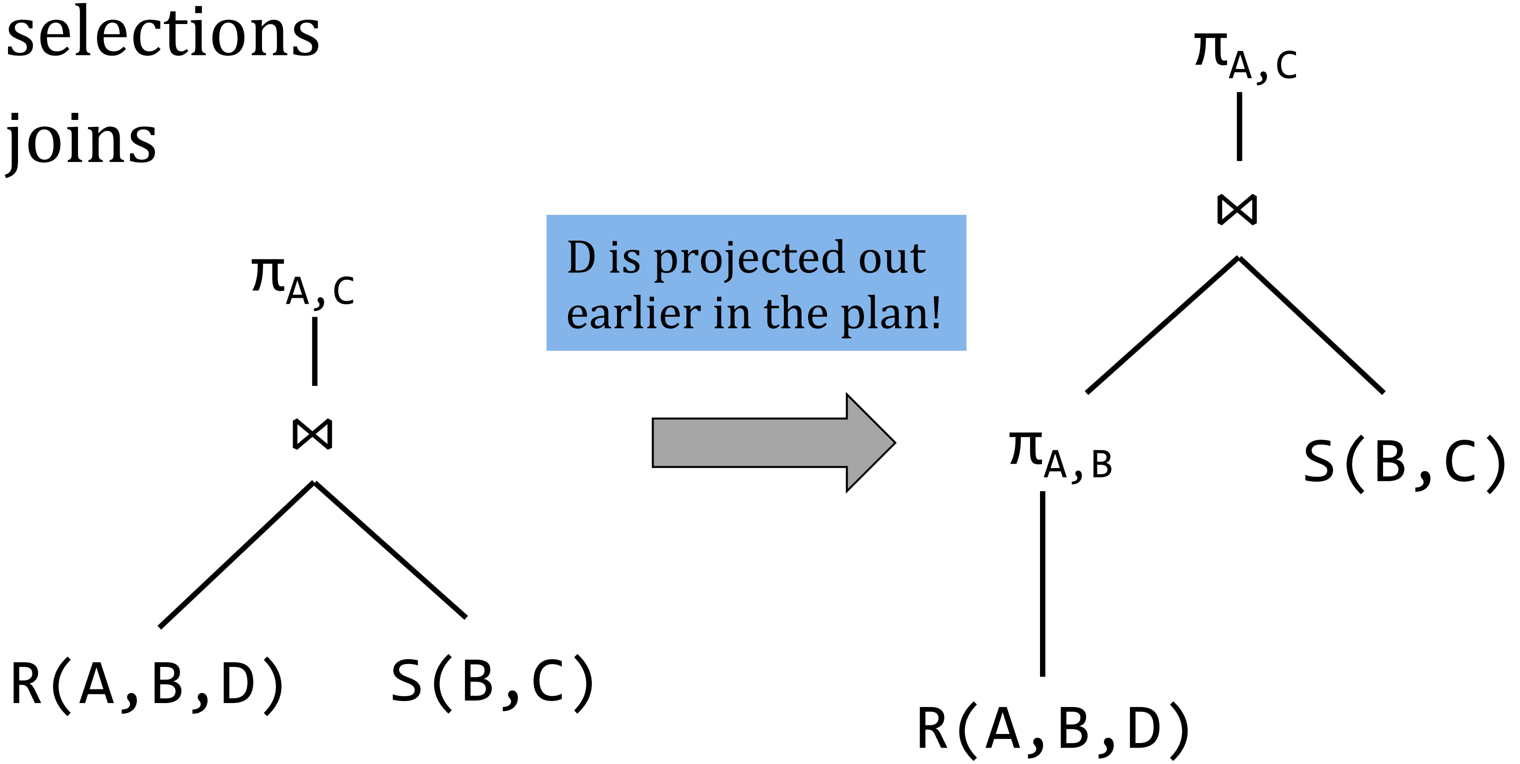
$\sigma_{A=10}$

R(A,B,C)

PUSHING DOWN PROJECTIONS

A projection can be pushed down through

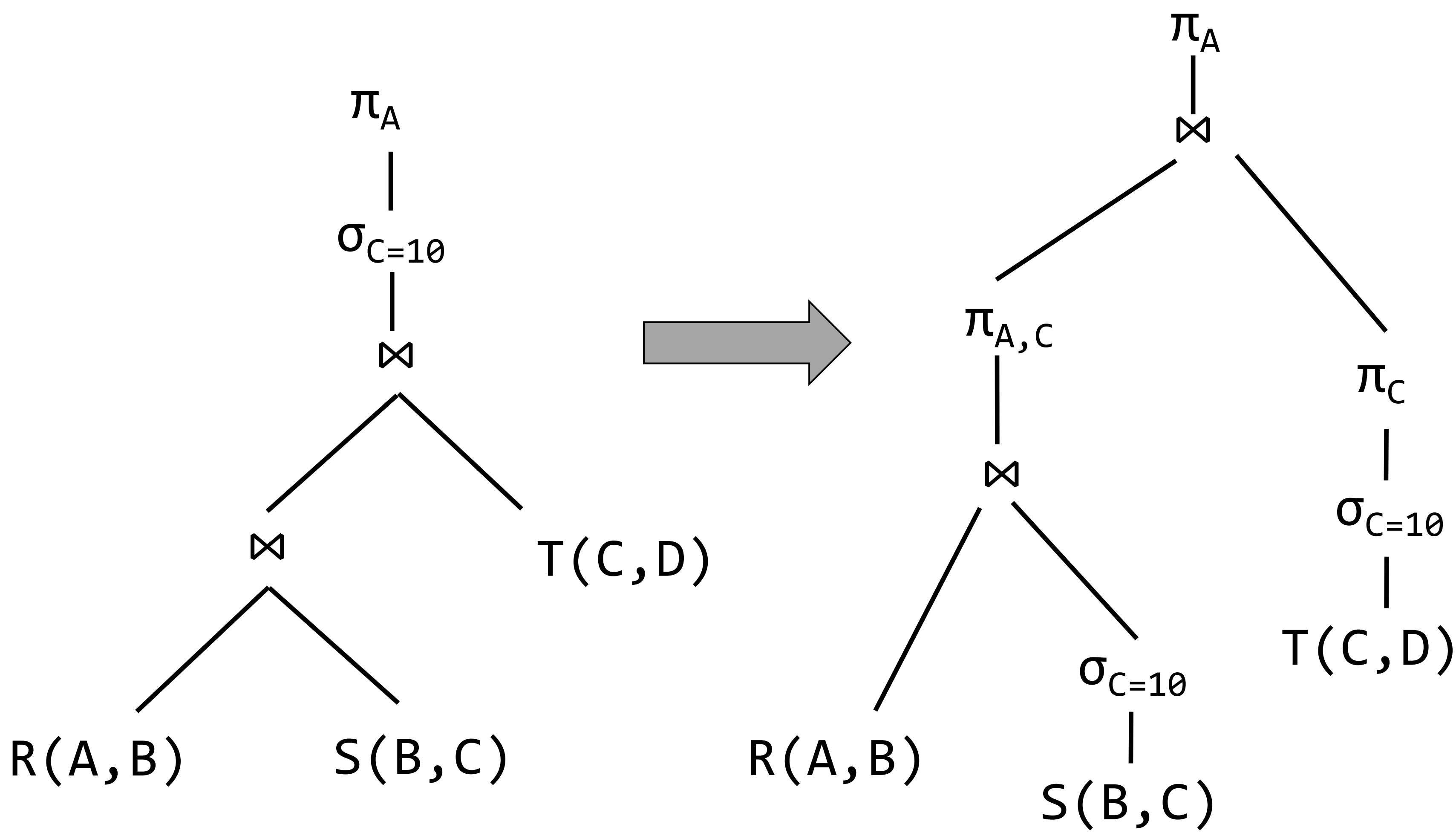
- selections
- joins



SELECTIONS & PROJECTIONS

- Heuristically, we want selections and projections to occur as early as possible in the query plan
- **The reason**: we will have fewer tuples in the intermediate steps of the plan
  - this could fail if the selection condition is very very expensive
  - projection could be a waste of effort, but more rarely

EXAMPLE



π_A
$\sigma_{C=10}$
⋈
⋈
T(C,D)
R(A,B)
S(B,C)
π_A
⋈
$\pi_{A,C}$
π_C
⋈
$\sigma_{C=10}$
$\sigma_{C=10}$
T(C,D)
R(A,B)
S(B,C)

JOIN REORDERING

- Commutativity of join

$$R \bowtie S \equiv S \bowtie R$$

- Associativity of join

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

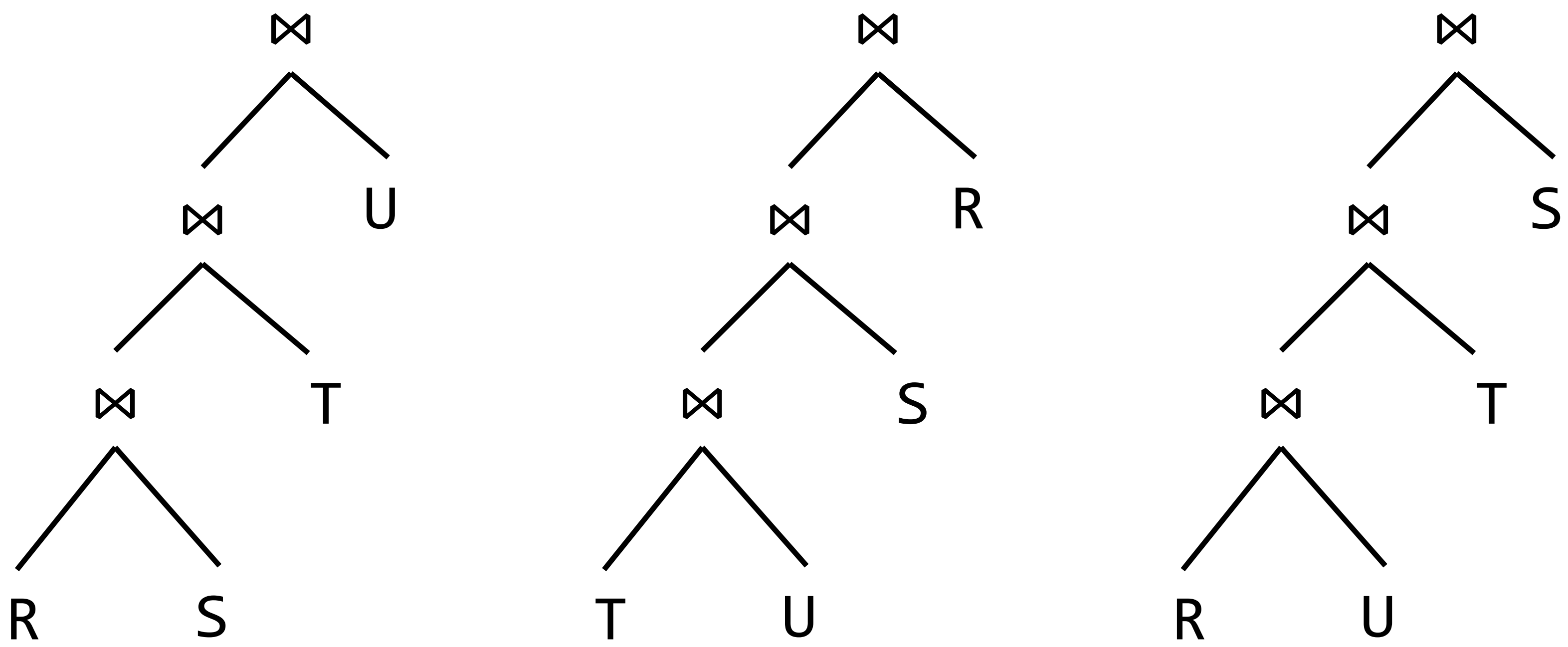
We can reorder the computation of joins in any way (exponentially many orders)!

JOIN REORDERING

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(D,E)$$

left-deep join plans

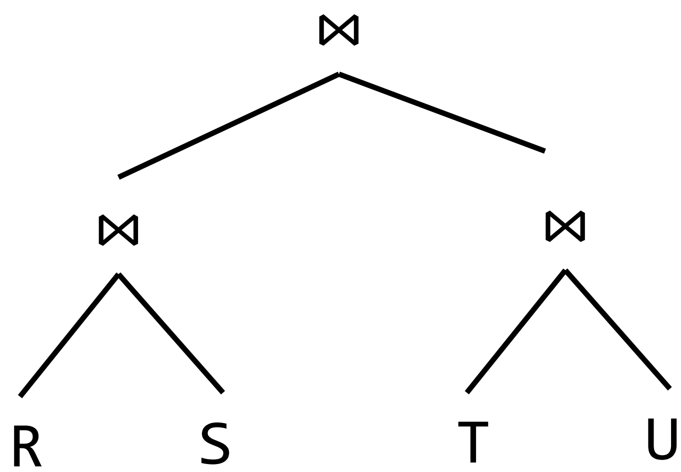
correct, but not a good plan!



JOIN REORDERING

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(D,E)$$

bushy plan



RA EXPRESSIONS: RECAP

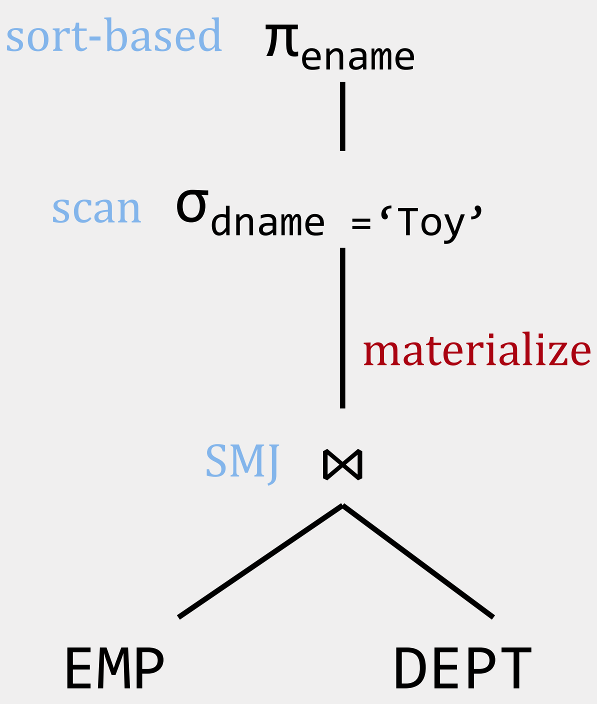
- selections can be evaluated in any order
- joins can be evaluated in any order
- selections and projections can be pushed down the tree using the RA equivalence transformations

3 – ANNOTATING THE PLAN

Finally, we need to choose

- the algorithm for each operator in the plan
- how to coordinate across different operators
  - materialization
  - pipelining

MATERIALIZATION



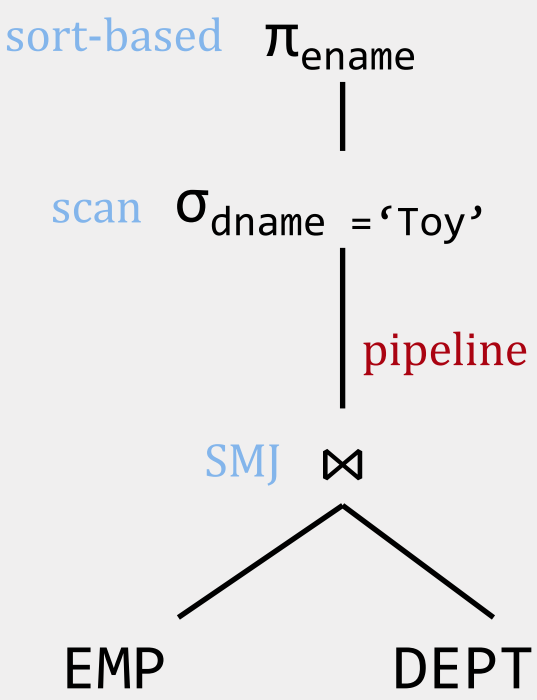
- After the operator is evaluated, write the intermediate result to disk
- The next operator reads its input from the disk

- The join output is written to disk
- The selection operator reads the intermediate result from disk

MATERIALIZATION

- We can always apply materialization!
- The cost can be high, since we need to write/read the intermediate result from the disk

PIPELINING



- Whenever an operator produces a result, pass it to its parent
- Operators can run simultaneously!

We can apply the selection condition as the tuples are generated from the join operator, without writing the result to disk!

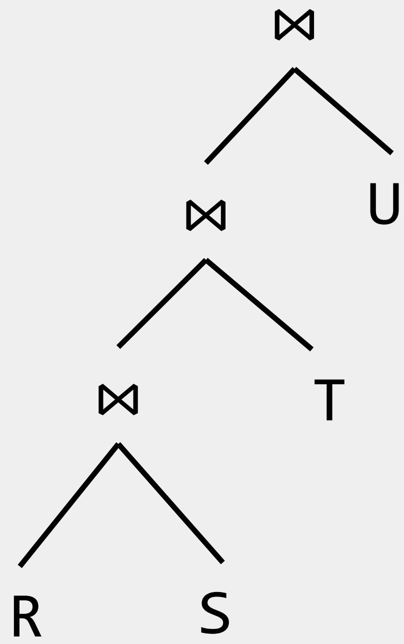
PIPELINING

- By using pipelining we benefit from:
  - no reading/writing to disk of the temporary relation
  - overlapping execution of operators
- Pipelining is not always possible!
- **Blocking operators** cannot output tuples without having read the whole input:
  - Any sort-based algorithm is blocking and cannot be pipelined!
  - Hash join is also blocking

PIPELINING

Left-deep join plans allow for fully pipelined evaluation

for BNLJ, left child = outer relation



QUERY PLAN COST ESTIMATION

COST ESTIMATION

Estimating the cost of a query plan involves:

- estimating the cost of each operation in the plan
  - depends on input cardinalities
  - algorithm cost (we have seen this!)
- estimating the size of intermediate results
  - we need statistics about input relations
  - for selections and joins, we typically assume independence of predicates

COST ESTIMATION

- Statistics are stored in the system catalog:
  - number of tuples (*cardinality*)
  - size in pages
  - # distinct keys (when there is an index on the attribute)
  - range (for numeric values)
- The system catalog is updated periodically
- Commercial systems use additional statistics, which provide more accurate estimates:
  - histograms
  - wavelets

EXAMPLE: COST ESTIMATION

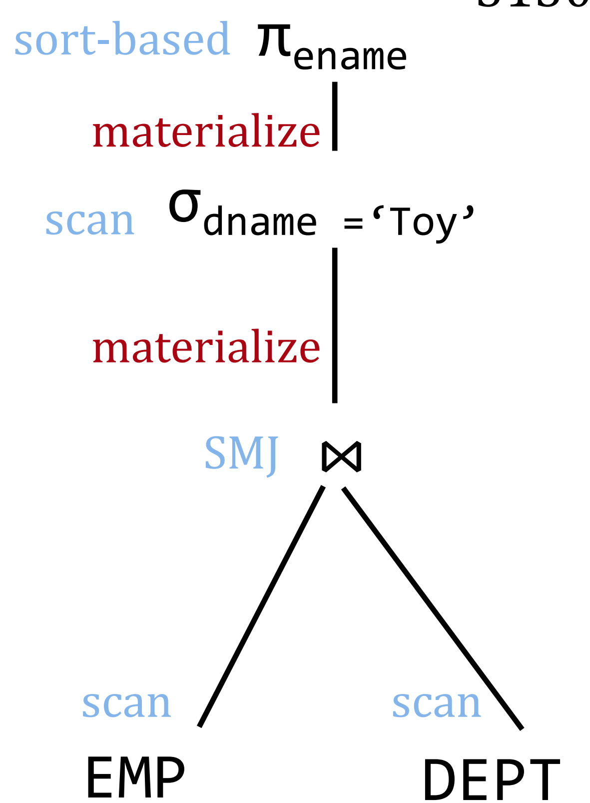
- `EMP(ssn, ename, addr, sal, did)`
  - 10000 tuples, 1000 pages
- `DEPT(did, dname, floor, mgr)`
  - 500 tuples, 50 pages
  - 100 distinct values for `dname`

```
SELECT DISTINCT ename
FROM   Emp E, Dept D
WHERE  E.did = D.did
AND    D.dname = 'Toy' ;
```

EXAMPLE: COST ESTIMATION

buffer size B= 40

total I/O cost = 3150 +



intermediate result ~ 2000 pages

- 10,000 tuples in the result
- The join tuple is double the size
- Thus we need twice as many pages!

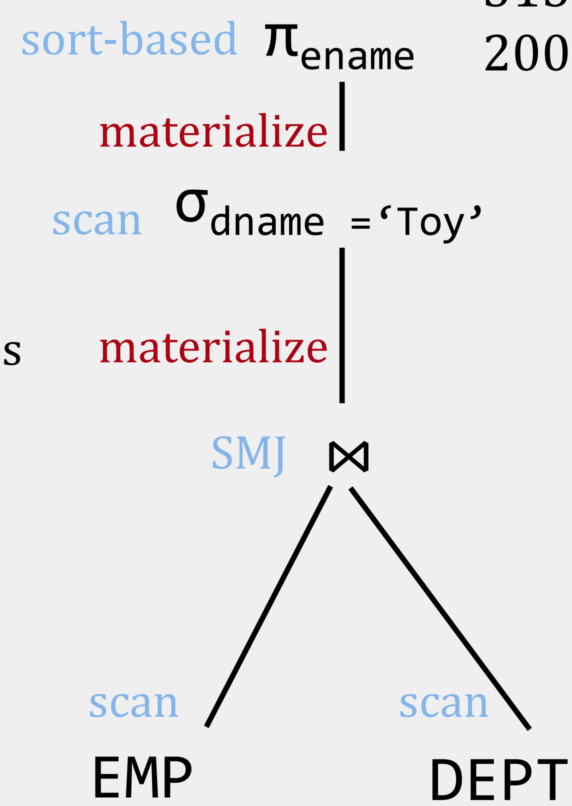
cost of SMJ = $3 * (1000 + 50)$

Since this join involves a primary key, no need for backup!

EXAMPLE: COST ESTIMATION

buffer size B= 40

total I/O cost =
3150 +
2000 {materialize} +



intermediate result ~ 2000 pages

cost of SMJ = 3 ∗ (1000 + 50)

after the join, we **materialize** the result to disk

EXAMPLE: COST ESTIMATION

buffer size B= 40

total I/O cost =
3150 +
2000 {materialize} +
2000 +

intermediate result ~ 20 pages

- dname has 100 distinct values
- selectivity is 1%

cost of selection = 2000

intermediate result ~ 2000 pages

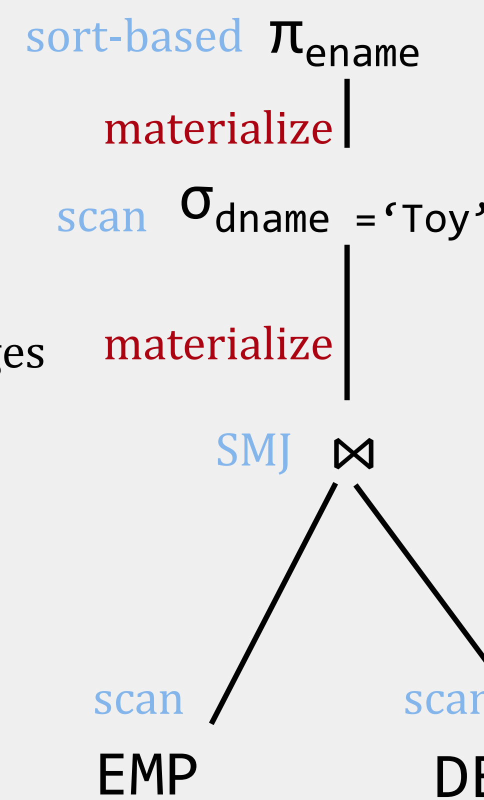
cost of SMJ = $3 * (1000 + 50)$

sort-based $\pi_{\texttt{ename}}$
materialize
scan $\sigma_{\texttt{dname = 'Toy'}}$
materialize
SMJ $\bowtie$
scan EMP
scan DEPT

EXAMPLE: COST ESTIMATION

buffer size B= 40

total I/O cost =
3150 +
2000 {materialize} +
2000 +
20 {materialize} +



intermediate result ~ 20 pages

cost of selection = 2000

intermediate result ~ 2000 pages

cost of SMJ = $3 * (1000 + 50)$

after the selection, we **materialize** the result to disk

EXAMPLE: COST ESTIMATION

buffer size B= 40

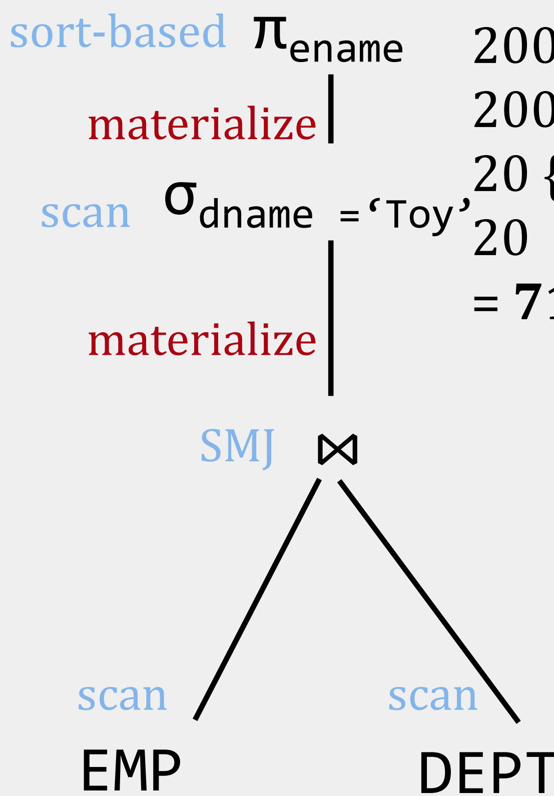
cost of projection = 20

intermediate result ~ 20 pages

cost of selection = 2000

intermediate result ~ 2000 pages

cost of SMJ = 3 ∗ (1000 + 50)



total I/O cost =
3150 +
2000 {materialize} +
2000 +
20 {materialize} +
20
= **7190**

COST ESTIMATION W/ PIPELINING

buffer size B= 40

cost of projection = 20

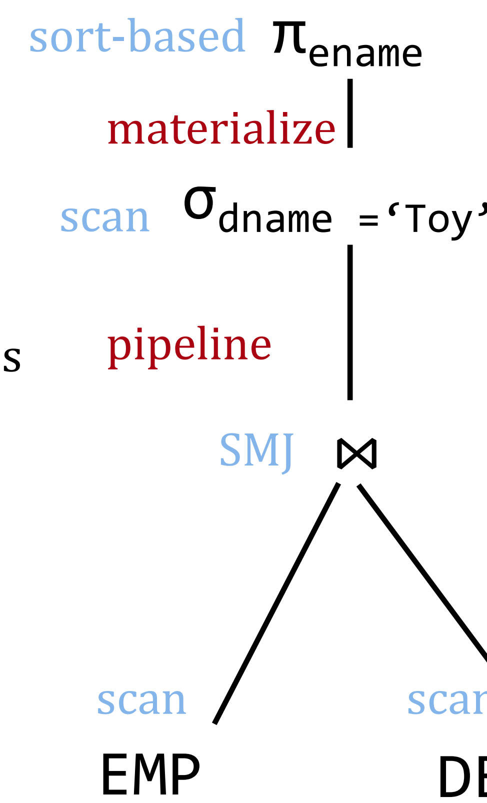
intermediate result ~ 20 pages

cost of selection = 0

intermediate result ~ 2000 pages

cost of SMJ = $3 * (1000 + 50)$

we can pipeline the result after the join operator!



total I/O cost =
3150 +
0 {pipeline} +
0 +
20 {materialize} +
20
= **3190**