QUERY OPTIMIZATION

CS 564- Fall 2016

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

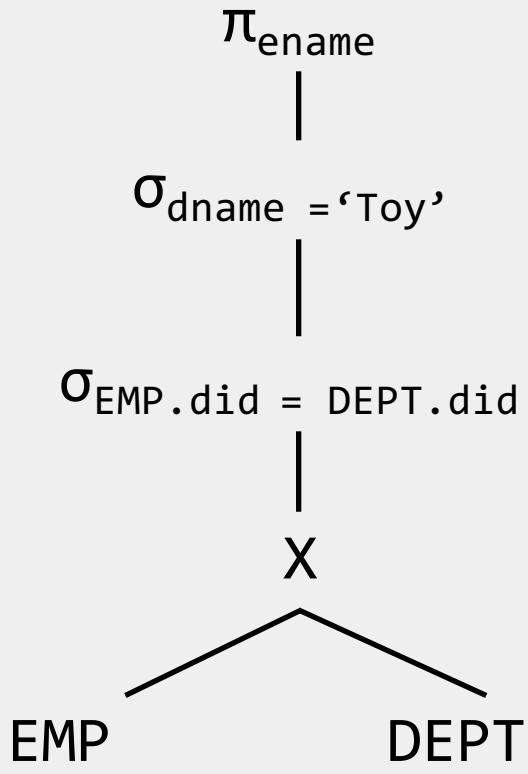
EXAMPLE QUERY

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

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

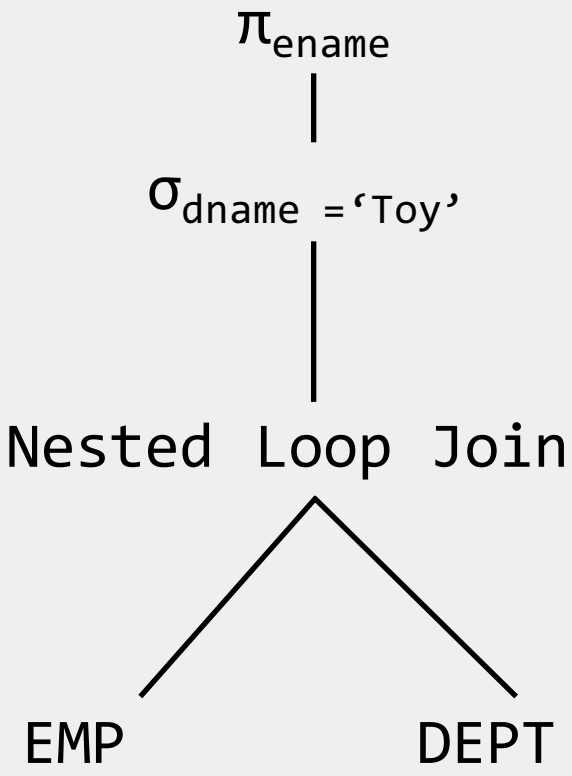
Evaluation Plan (1)

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



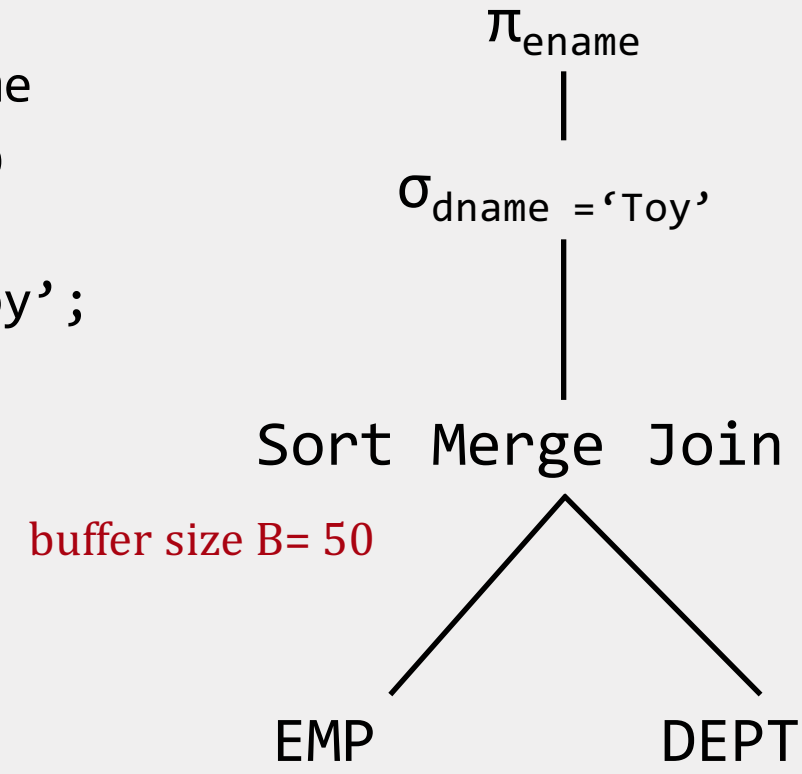
Evaluation Plan (2)

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



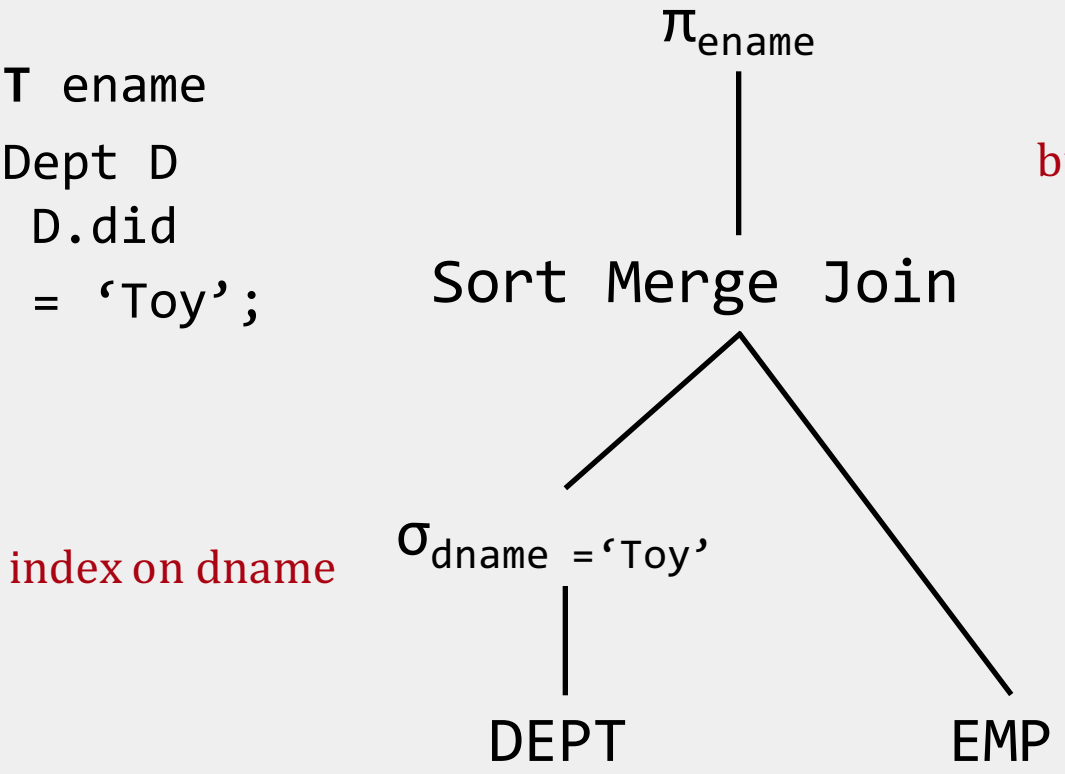
Evaluation Plan (3)

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



Evaluation Plan (4)

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



buffer size B= 50

index on dname

PIPELINED EVALUATION

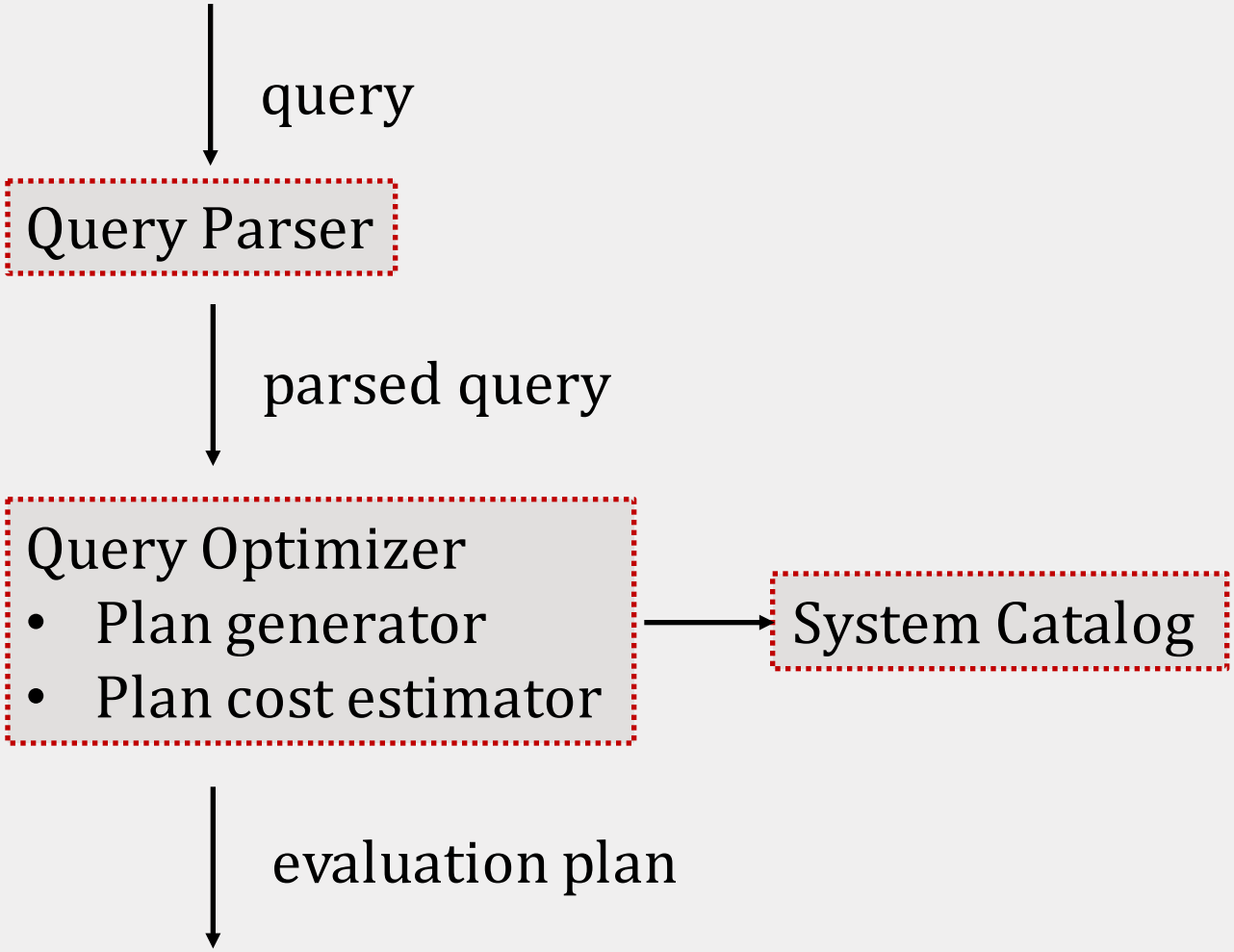
- Instead of materializing the temporary relation to disk, we can instead pipeline to the next operator in memory
- By using pipelining we benefit from:
  - no reading/writing to disk of the temporary relation
  - overlapping execution of operators
- Pipelining is not always possible!

Query Optimization

The query optimizer

1. identifies candidate equivalent trees
2. for each tree it finds the best annotated version (using any available indexes): this is called a plan
3. chooses the best overall plan by estimating the cost of each plan

Architecture Of an Optimizer



QUERY OPTIMIZATION

- **query plan**: annotated Relational Algebra tree
  - iterator interface: open() /getNext() /close()
  - can be pipelined or materialized
- The optimizer must solve two main issues:
  - What is the space of possible query plans?
  - How can we estimate the cost of each plan?
- *Ideally*: best plan!
- *Practically*: avoid worst plans + look at a subset of all plans

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 know 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

Evaluation Plans

- The space of possible query plans is typically huge and it is hard to navigate through
- The RA formalism provides us with mathematical rules that transform one RA expression to an equivalent one: for example
  - push selections down
  - reorder joins
- This way we can construct many equivalent alternative query plans

RA Equivalence (1)

- Commutativity of σ

$$\sigma_{P_1}(\sigma_{P_2}(R)) \equiv \sigma_{P_2}(\sigma_{P_1}(R))$$

- Cascading of σ

$$\sigma_{P_1 \wedge P_2 \wedge \cdots \wedge P_n}(R) \equiv \sigma_{P_1}(\sigma_{P_2}(\ldots \sigma_{P_n}(R)))$$

- Cascading of π

$$\pi_{\alpha_1}(R) \equiv \pi_{\alpha_1}(\pi_{\alpha_2}(\ldots \pi_{\alpha_n}(R) \ldots)) \text{ when } a_i \subseteq a_{i+1}$$

- *We can evaluate selections in any order!*

RA Equivalence (2)

- 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)!*

RA Equivalence (3)

- Selections + Projections

$\sigma_{\mathrm{P}}\ (\pi_a(R))\ \equiv \pi_a(\sigma_P(R))$ (if the selection involves attributes that remain after projection)

- Selections + Joins

$\sigma_{\mathrm{P}}(R \bowtie S) \equiv \sigma_{\mathrm{P}}(R) \bowtie S$ (if the selection involves attributes only in R)

- *We can push selections down the plan tree!*

Evaluation Plans

Single relation plan (no joins):

- file scan
- index scan(s): clustered or non-clustered
  - more than one index may "match" predicates
- The optimizer chooses one with the least estimated cost
- We can also merge or pipeline selection and projection (and aggregate when there is no group by)

EVALUATION PLANS

Multiple relation plan

- joins can be evaluated in any order
- selections can be combined into the join operator
- selections and projections can be pushed down the plan tree using the RA equivalence transformations

JOIN REORDERING

Consider the following join: $R \bowtie S \bowtie T \bowtie U$

- Most DBMSs consider *left-deep* join plans
- These allow for fully pipelined evaluation