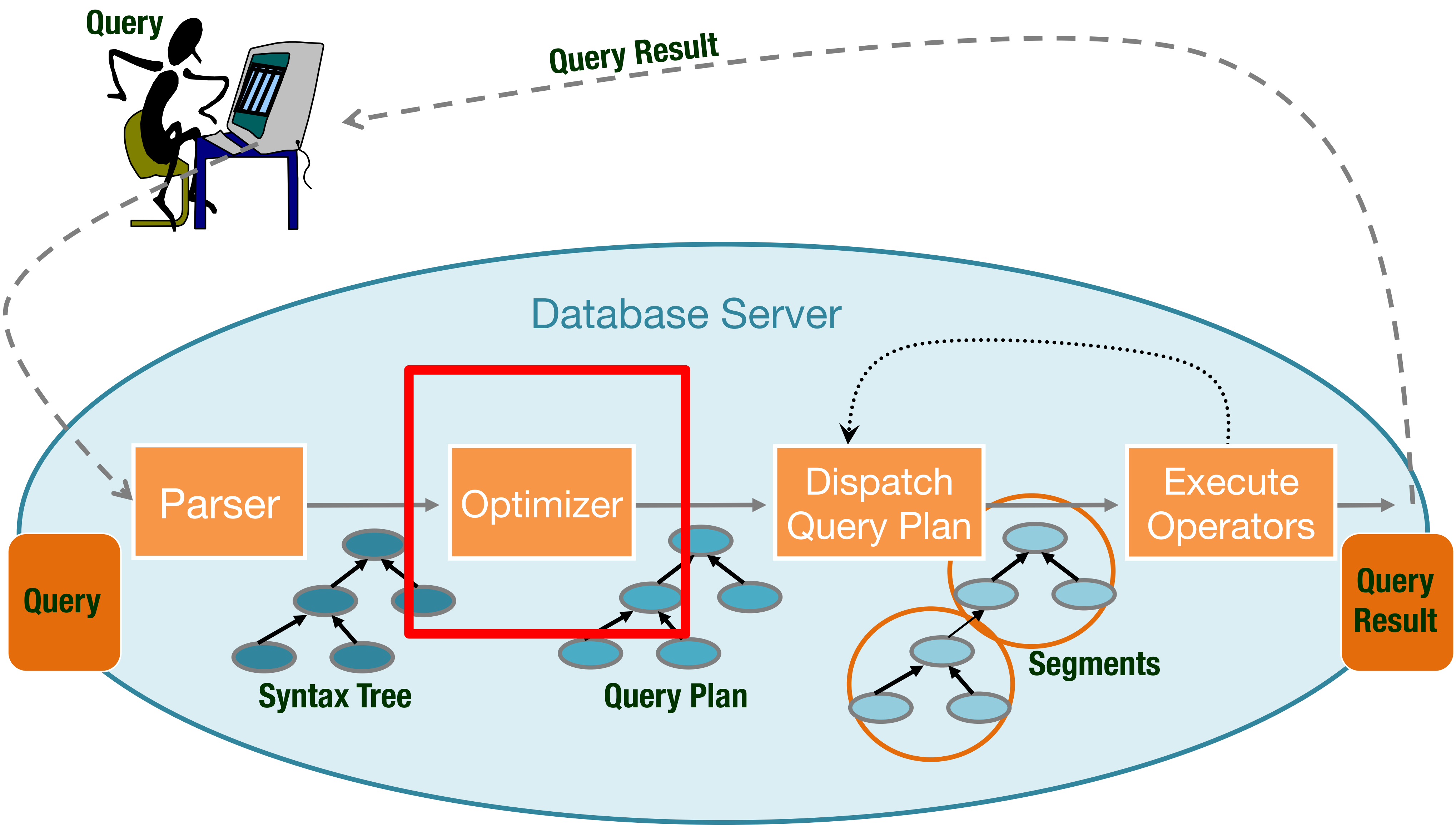
EECS 484- Database Management Systems

Query Optimization

Chapter 15

Query Execution Life-Cycle



Query Optimization

Query optimizer selects an evaluation plan with the lease cost in two steps:

1. **Plan Enumeration**

   Today
   - Different query plans
   - Different implementations (i.e., evaluation algorithms) for each operator

2. **Cost Estimation**
   - Cost of each operator
   - Overall cost of the plan

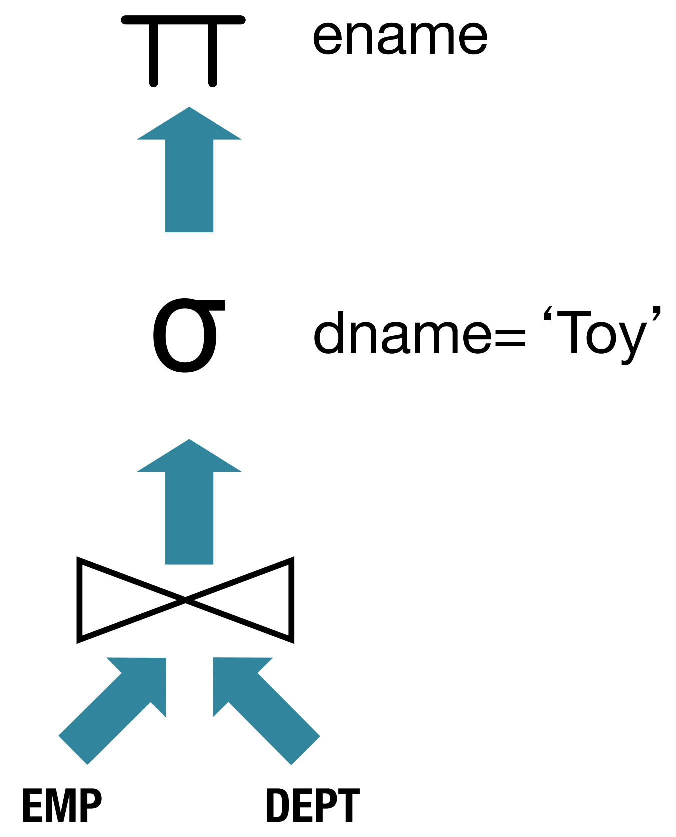
Previous lectures

Example: RA Tree

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

```
DEPT (did, dname, floor, mgr)
```

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

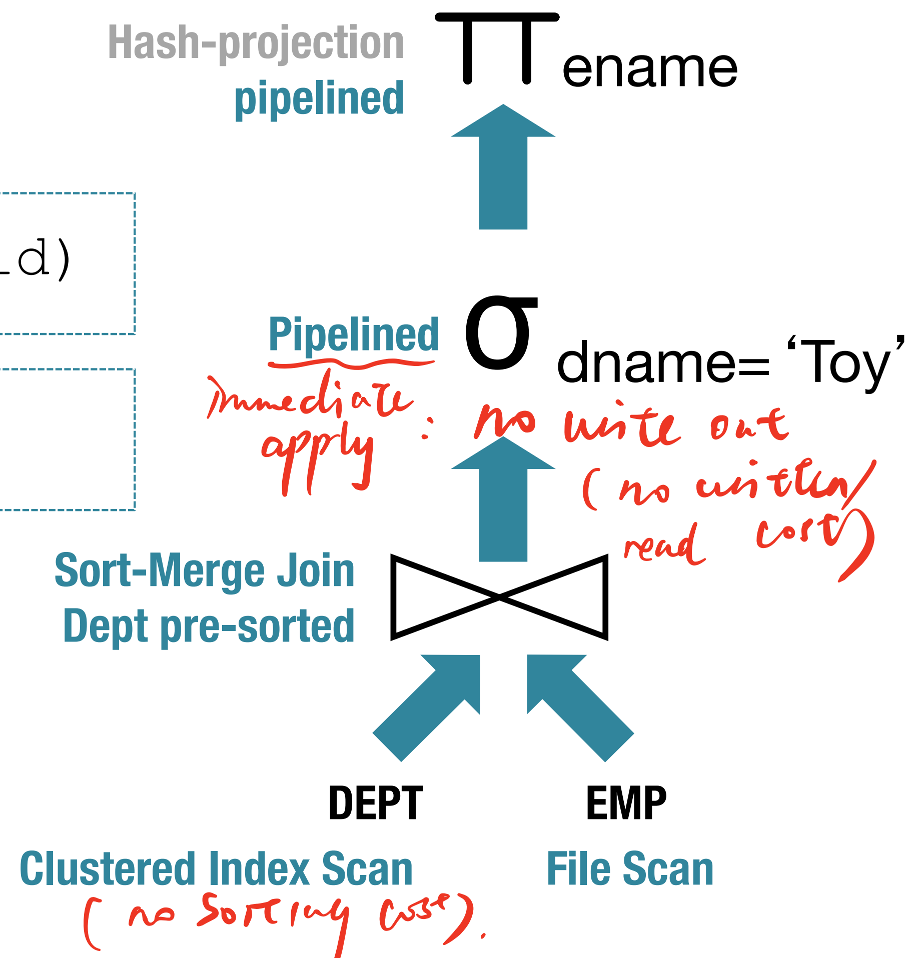


Example: Annotated RA Tree

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

```
DEPT (did, dname, floor, mgr)
```

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



Annotated RA Tree

Annotated RA Expressions

- Which algorithm to use for each operator
- Where Intermediate Results are:
  - Pipelined: Tuples resulting from one operator fed directly into the next
  - Materialized: Create a temporary table to store intermediate results

RA Equivalence - Selections

$\sigma_{P1}(\sigma_{P2}(R)) \equiv \sigma_{P2}(\sigma_{P1}(R))$ (σ commutativity)

$\sigma_{P1 \wedge P2 \ldots \wedge Pn}(R) \equiv \sigma_{P1}(\sigma_{P2}(\ldots \sigma_{Pn}(R)))$ (cascading σ)

Selection operation is commutative and multiple selections on the same relation can be combined into a single selection

RA Equivalence – Projections

$\prod_{a1}(R) \equiv \prod_{a1}(\prod_{a2}(\ldots\prod_{ak}(R)\ldots))$, $a_i \subseteq a_{i+1}$ (cascading $\prod$)

previous proj. must be superset of following projections

- In the above, each a_i is a set of attributes.
- For example: Let's say R has attributes c1, c2, c3, c4

$\prod_{c1,c3}(R) \equiv (\prod_{c1,c3}(\prod_{c1,c2,c3}(R))$

- Basically, we are eliminating one attribute at a time
- The above is useful when optimizing expressions that involve both projections and joins

RA Equivalence: Cross-Products & Joins

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

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

$R \times S \equiv S \times R$ (commutativity)

$R \times (S \times T) \equiv (R \times S) \times T$ (associativity)

- => joins and cross products can be performed in any order (SQL can reorder the columns at final presentation time)

Question?

For particular tables R and S in a database instance, a query optimizer has determined that the join method chosen to compute $R \bowtie S$ should use R as outer and S as inner.

If asked to compute $S \bowtie R$, this query optimizer will likely choose:

A. S as outer and R as inner
B. S as inner and R as outer ✓
C. The choice does not matter.

Doing projections and selections first

$\Pi_A(\sigma_c(R)) \equiv \sigma_c(\Pi_A(R))$ (if c only uses attr in A)

$\sigma_P(R \times S) \equiv (R \bowtie_P S)$

P only applies on R

$\sigma_P(R \times S) \equiv \sigma_P(R) \times S$ (if p is only on R)

$\sigma_P(R \bowtie S) \equiv \sigma_p(R) \bowtie S$ (if p is only on R)

Key ideas: When possible, consider doing:

- Joins rather than cross-products
- Selections before joins
- (Partial) Projections before all operators.

Doing selections before join

$$\sigma_P\ (R \times S) \equiv \sigma_{P4}\ (\sigma_{P1}(R)\ \bowtie_{p3}\ \sigma_{P2}(S))$$

Note: $P = p1 \wedge p2 \wedge p3 \wedge p4$

Idea: Replace P by a cascade of conditions p1, p2, p3, and p4 such that:

P1: conditions only on R (e.g., R.a = 5)
P2: conditions only on S (e.g., S.d > 9)
P3: join conditions with equality on R and S (R.a = S.a)
P4: other conditions involving both R and S columns (e.g., R.a > S.b)

Example

$$\sigma_P \ (R \times S) \equiv \sigma_{P4} \ (\sigma_{P1}(R) \ \bowtie_{p3} \ \sigma_{P2}(S))$$

Note: $P = p1 \wedge p2 \wedge p3 \wedge p4$

$$\sigma_{major=CS \ \wedge \ R.umid=S.umid \ \wedge \ grade>C} \ (R \times S) \equiv$$
$$\sigma_{true} \ (\sigma_{major=CS}(R) \ \bowtie_{umid} \ \sigma_{grade>c}(S))$$

umid	sname	major
23	Alice	CS
71	Bob	CE
11	Mary	CS
13	John	CS

umid	course	semes	grade
13	484	W17	C+
23	484	W17	C+
71	484	W17	C+
23	482	W17	C
71	482	W17	D
11	482	W17	D-

RA Equivalence – Multiple Ops

$$\Pi_P (R \times S) \equiv \Pi_{P1}(R) \times \Pi_{P2}(S)$$

Columns p in the cross-product consist of columns $\underbrace{\text{p1 from R}}_{1}$ and columns $\underbrace{\text{p2 from S}}_{2}$

$$\Pi_P (R \bowtie_c S) \equiv \Pi_P (\Pi_{P1}(R) \bowtie_c \Pi_{P2}(S))$$

☆ for join & projection

p1 are attrs of R that appear in p or c

p2 are attrs of S that appear in p or c

Example

$$\Pi_P\ (R \bowtie_C S) \equiv \Pi_P\ (\Pi_{P1}(R) \bowtie_C \Pi_{P2}(S))$$

$$\Pi_{\mathtt{sname,\ course}}\ (R \bowtie_{\mathtt{umid}} S)$$
$$\equiv \Pi_{\mathtt{sname,\ course}}\ (\Pi_{\mathtt{sname,umid}}(R) \bowtie_{\mathtt{umid}} \Pi_{\mathtt{course,umid}}(S))$$

umid	sname	major
23	Alice	CS
71	Bob	CE
11	Mary	CS
13	John	DS

umid	course	semes	grade
13	484	W17	C+
23	484	W17	C+
71	484	W17	C+
23	482	W17	C
71	482	W17	D
11	482	W17	D-

RA Equivalence – More Rules

$$\Pi_{A1,A2,\ldots An}(\sigma_P\ (R)) \equiv \Pi_{A1,A2,\ldots An}(\sigma_P\ (\Pi_{A1,\ldots An,\ B1,\ldots\ BM} R))$$

need for selection & projection

B1 … BM attributes in P

$$\Pi_{umid}(\sigma_{major=CE}(R)) \equiv \Pi_{umid}(\sigma_{major=CE}(\Pi_{umid,major} R))$$

umid	sname	major
23	Alice	CS
71	Bob	CE
11	Carl	CS
13	Debra	DS

Optimization Strategies

- Rule-Based
  - Choose more likely cheaper plan based on RA tree
  - E.g. Push selections in.
  - E.g. Push projections in.
- Cost-Based
  - Estimate cost and choose lower-cost plan
  - E.g. choosing between join methods.

Cost Based Optimization

- Traditional techniques work well for < 10 joins
- Cost estimation: Approximate at best
  - Use statistics from systems catalogs
  - Combination of CPU and I/O costs
    - EECS 484 focuses on I/O costs only
    - We also use **System R** approach
      - very inexact but ok in practice (better techniques are known now)

Estimating the Cost of a Plan

1. Estimate *cost* of each operation in plan tree
   - Depends on input cardinalities (# of rows)
   - Algorithm cost (see previous lectures)
2. Estimate *size of result*
   - Use information about the input relations
   - For selections and joins, assume independence of predicates

Collecting & Maintaining Statistics

- Statistics stored in the catalogs (pg_stats & pg_class in Postgres)
  - Relation
    - Cardinality (# rows)
    - Size in pages
  - Index or attribute
    - 1 Cardinality (# distinct key values)
    - 2 For index: Size in pages and Height (of b-tree)
    - 3 Range of values; Possibly histogram as well e.g. employee age 18-75
- Catalogs update periodically
  - Can be slightly inconsistent/inaccurate

Estimating Output Size

```
SELECT attribute list
FROM relation list
WHERE  term1 AND ... AND termk
```

Question: What is the cardinality of the result set?

- Max # tuples: product of input relation cardinalities
- Each term "filters" out some tuples: Reduction factor
- Result cardinality = Max # tuples * product of all RF's
- Assumption: terms are independent!
- Term *col=value* RF: 1/Range(I)
- Term col>value RF: (High(I)-value)/(High(I)-Low(I))
- Term col1=col2 RF: 1/MAX(Range(I1), Range(I2))

Cost of different plans

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

```
SELECT E.ename
FROM   Emp E
WHERE  E.did=8
AND    E.sal > 40K
```

Estimate the cost of different access methods:

- Index on did:
  - Clustered index:
  - Unclustered index:
- Index on sal:
  - Clustered index:
  - Unclustered index:
- File scan: 1,000 pages

1,000 data pages, 10K tuples
100 pages in B+-tree
depts: 10
Salary Range: 0K – 200K

Cost of different plans

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

```
SELECT E.ename
FROM   Emp E
WHERE  E.did=8
AND    E.sal > 40K
```

- Index on did:
  - Tuples Retrieved: (1/10) * 10,000
  - Clustered index:

    leaf pages data pages

    (1/10) * (100+1,000) pages of I/O = 110 I/Os
  - Unclustered index:

    (1/10) * (100+10,000) pages I/O = 1010 I/Os

    leaf

    each tuple cost an IO for the unclustered index

Given:

- 1,000 data pages, 10K tuples
- 100 leaf pages in B+-tree
- All tree levels except leaves cached in memory
- # depts: 10
- Salary Range: 0K – 200K

Question?

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

- Index on did:
- Index on sal:
  - Tuples Retrieved = ??

A. 1000
B. 4000
C. 8000 ✓
D. 10,000

```
SELECT E.ename
FROM   Emp E
WHERE  E.did=8
AND    E.sal > 40K
```

80%

Given:

- 1,000 data pages, 10K tuples
- 100 leaf pages in B+-tree
- All tree levels except leaves cached in memory
- # depts: 10
- Salary Range: 0K – 200K

Cost of different plans

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

```
SELECT  E.ename
FROM    Emp E
WHERE   E.did=8
AND     E.sal > 40K
```

- Index on did:
  - Tuples Retrieved: (1/10) * 10,000
  - Clustered index: (1/10) * (100+1,000) = 110 pages
  - Unclustered index: (1/10) * (100+10,000) = 1,010 pages
- Index on sal:

  80 /

  - Clustered index: (200-40)/(200-0) * (100+1,000) = 880 pages
  - Unclustered index: (200-40)/(200-0) * (100+10,000) = 8,080 pages
- File scan: 1,000 pages

1,000 data pages, 10K tuples
100 pages in B+-tree
depts: 10
Salary Range: 0K – 200K

Estimating Output Size

```
SELECT attribute list
FROM relation list
WHERE  term1 AND ... AND termk
```

Question: What is the cardinality of the result set?

- Max # tuples: product of input relation cardinalities
- Each term "filters" out some tuples: Reduction factor
- **Result cardinality = Max # tuples * product of all RF's**
- Assumption: terms are independent!
- Term *col=value* RF: 1/Range(I)
- Term col>value RF: (High(I)-value)/(High(I)-Low(I))
- **Term col1=col2 RF: 1/MAX(Range(I1), Range(I2))**

Example: Estimate size of join

- Given:

  unique parts in Parts ⇒ 1million

  - Parts Relation containing 1 million distinct values of (the primary key) attribute PartID

    unique parts in Orders ⇒ 1000

  - Orders relation comprising 2000 different orders of 1000 distinct Parts. (Some parts have more than one order).

- Q: What is estimated # of tuples in natural join between Parts and Orders on the attribute PartID?

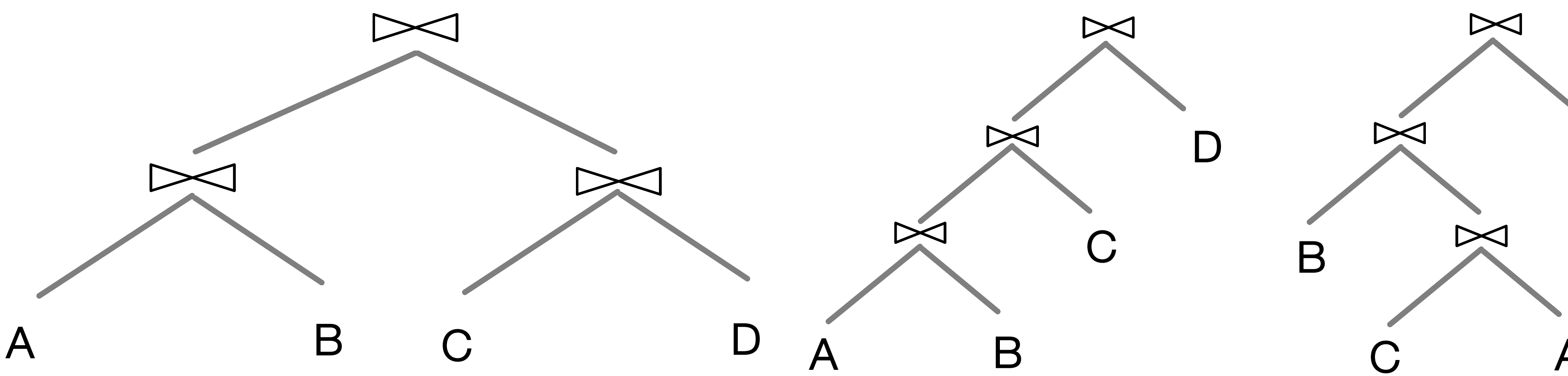
  $\max(1000, 1m) = 1m$

- **Solution:**
  - Size of cross-product**:** 2000 x 1 million tuples. (Call this A)
  - RF of [Orders.PartID = Parts.PartID selection predicate] **=** max(1 million ,1000) = 1 million (call this B)
  - Estimated size of natural join = A/B = **2000 tuples.** (Answer)

$$A \cdot \frac{1}{1m} = 2000 \text{ tuples}$$

Optimizing Join Order

- Consider (A join B join C join D).
- The query planner has many ways it can accomplish the join.
- Given n joins in the expression, way more than n! choices!

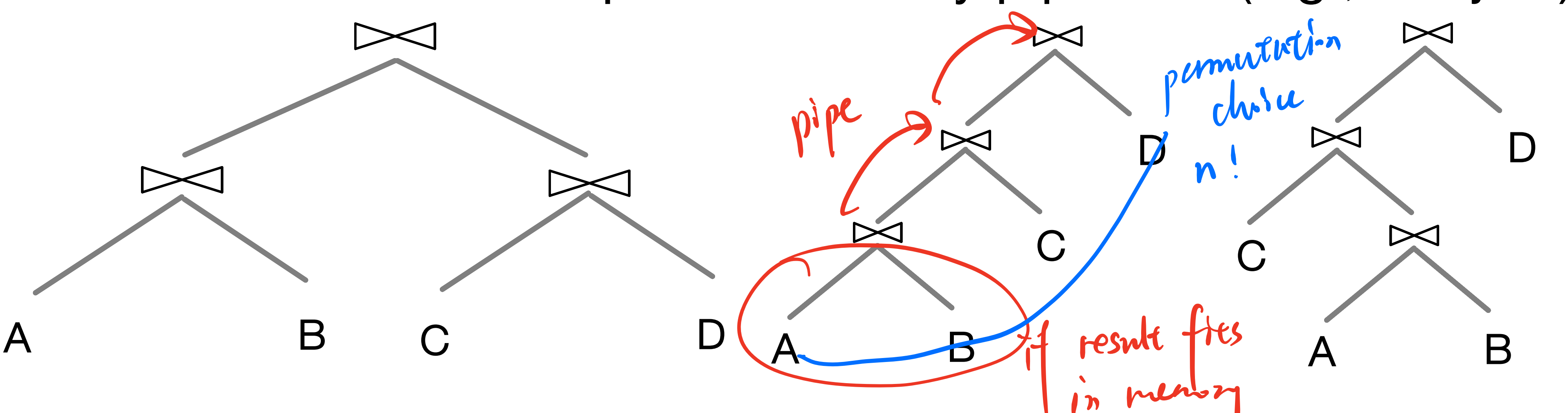


(Here, C and A would be joined if they share an attribute. Else we would do a cross product)

Optimizing Join Order

- System R : Only consider left-deep join trees
  - Used to restrict the search space
  - Left-deep plans can be fully pipelined (usually)
    - Intermediate results not written to temporary files
    - Not all left-deep trees are fully pipelined (e.g., SM join)

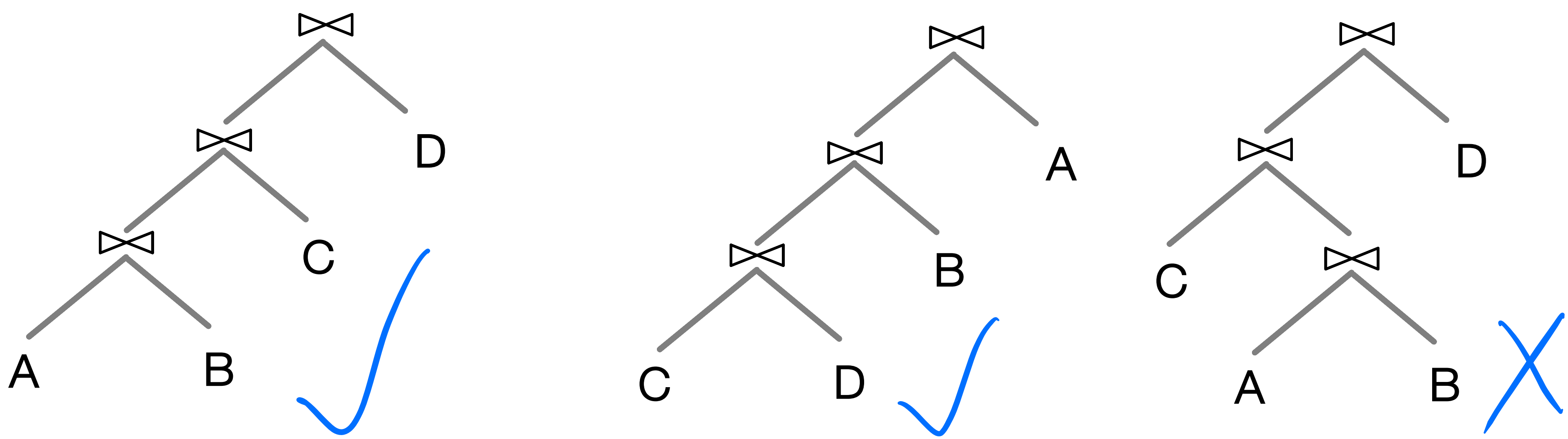
only left child is allowed to have children



Not left-deep join tree | Left-deep join tree | Not left-deep join tree

Example

Which one (if any) is a potential plan that System R will consider for joining A, B, C, and D?



Finding Best Left-Deep Plan

- Decide:
  - Join order
  - Join method for each join
- Enumerated using N passes (if N relations joined):
- Pass 1: Find best 1-relation plan for each relation (apply selections and projections first and consider using indexing)
- For each relation, retain only:
  - Cheapest plan overall (e.g. File scan), plus
  - Cheapest plans that produce ordered tuples. Order may be useful for later stages (e.g., sort-merge-join)

Enumeration of Left-Deep Plans

Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation (All 2-relation plans)

- For each pair (subset) of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for sorted order of tuples, if order is possibly useful for later operations.

- Also, apply selections and projections aggressively, when possible, to reduce result size
- Assume pipelining of results to avoid additional I/O

Enumeration of Left-Deep Plans

Pass N: Find best way to join result of retained (N-1)-relation plans (as outer) to the N^{th} relation

e.g. Sailor Reserves Boats

aim:

$S \bowtie R \bowtie B$

For each subset of relations, retain only:

- Cheapest plan overall, plus
- Cheapest plan for each interesting order of the tuples

find best to do

① S, R, B

② $S \bowtie R$, $R \bowtie B$ (no connection) no $S \bowtie B$

- Only consider joining relations if there is a connecting join condition, i.e., avoid Cartesian products if possible

- ORDER BY, GROUP BY handled as a final step

Question?

Given N relations, the number of possible left-deep plans is close to:

A. N

B. N^2

C. N!

D. $\binom{N}{2}$ = N choose 2

Summary

- Query optimization critical to the DBMS performance
  - Helps understand performance impact of database design
- Two parts to optimizing a query:
  - Enumerate alternative plans. (Typically only consider left-deep plans)
  - Estimate cost of each plan: size of result and cost of algorithm

Optional Exercises

12.1 (all parts), 15.1, 15.5, 15.7, 15.9