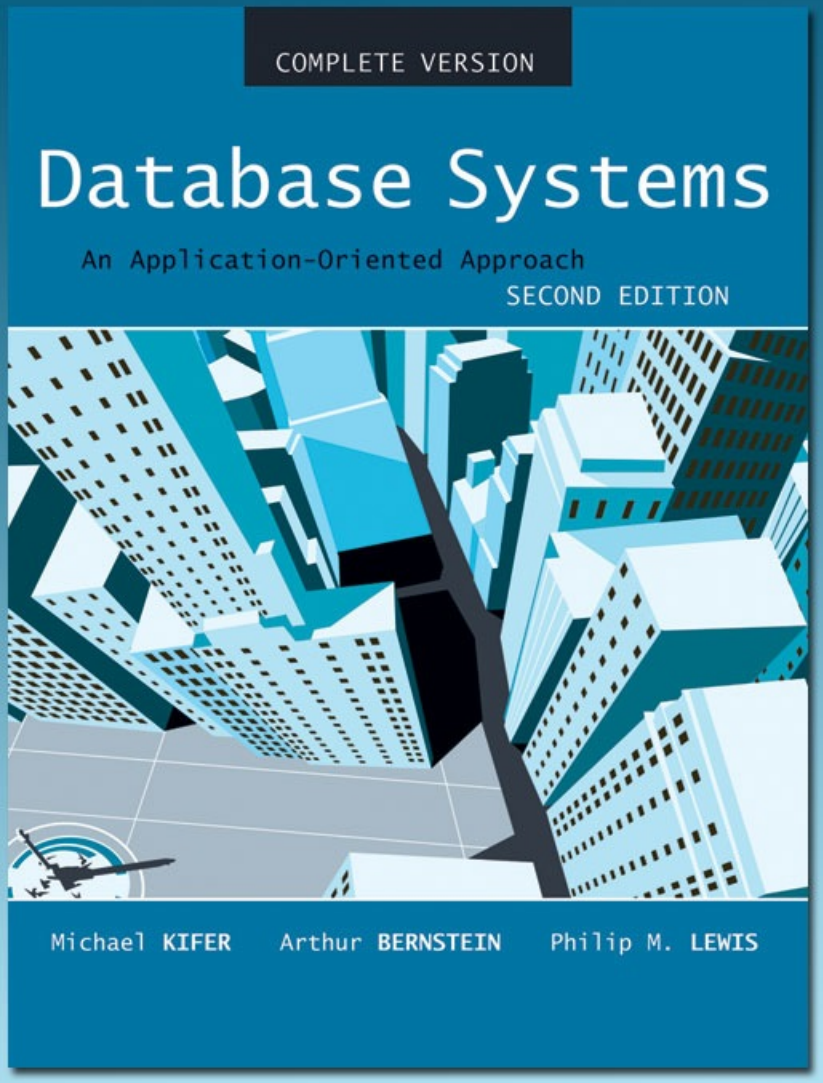


Chapter 11

An Overview of Query Optimization

PEARSON Addison Wesley
Copyright © 2006 Pearson Addison-Wesley. All rights reserved.

Query Evaluation

- ***Problem***: An SQL query is declarative – does not specify a query execution plan.
- A relational algebra expression is procedural – there is an associated query execution plan.
- ***Solution*****:** Convert SQL query to an equivalent relational algebra and evaluate it using the associated query execution plan.
  - *But which equivalent expression is best?*

Naive Conversion

SELECT DISTINCT *TargetList*
FROM R1, R2, …, RN
WHERE *Condition*

is equivalent to

$$\pi_{TargetList}(\sigma_{Condition}(R1 \times R2 \times \ldots \times RN))$$

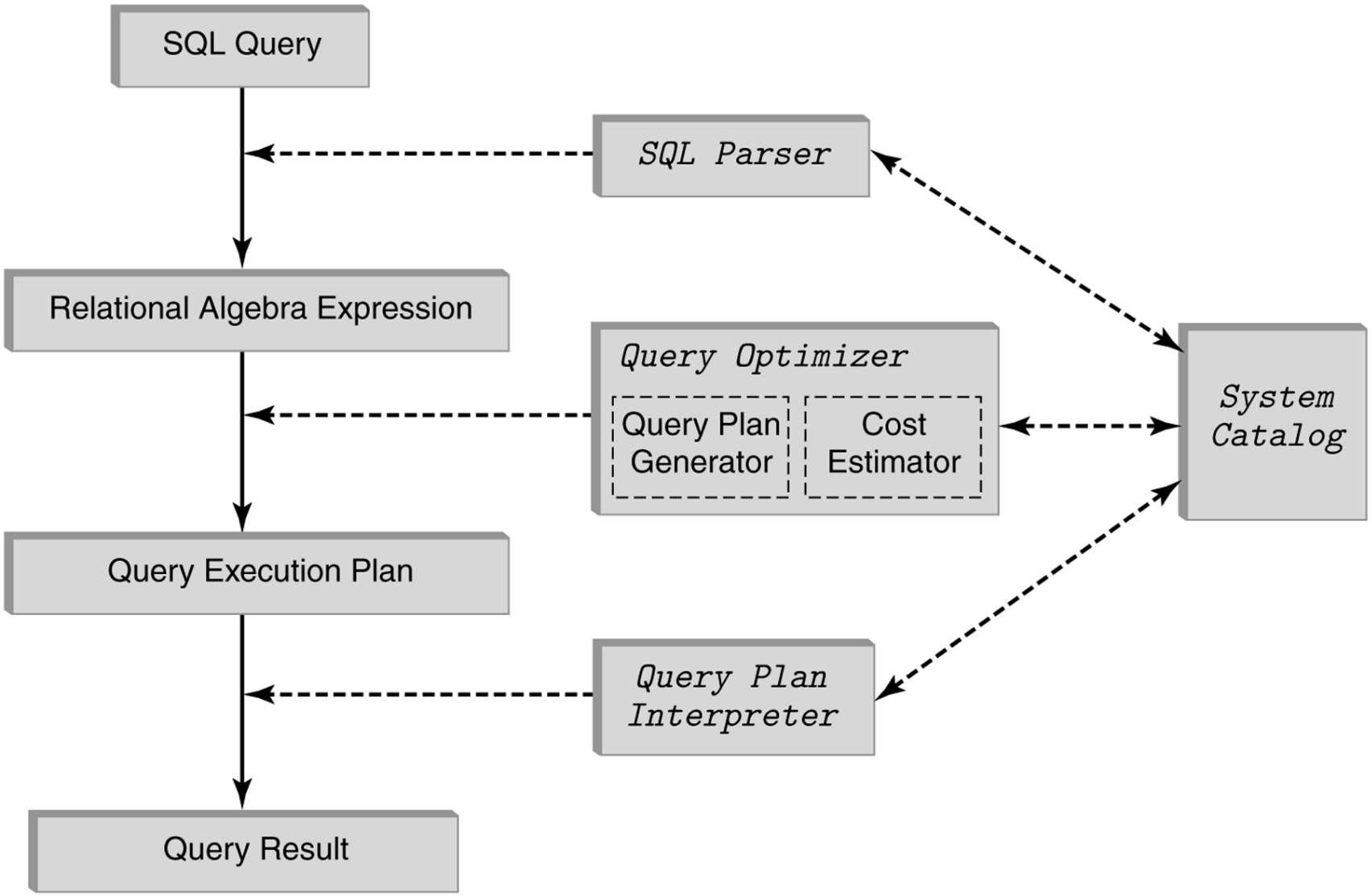
but this may imply a very inefficient query execution plan.

Example: $\pi_{Name}(\sigma_{Id=ProfId \wedge CrsCode='CS532'}(\text{Professor} \times \text{Teaching}))$

- Result can be < 100 bytes
- But if each relation is 50K then we end up computing an intermediate result Professor×Teaching of size 500M before shrinking it down to just a few bytes.

Problem: Find an *equivalent* relational algebra expression that can be evaluated "*efficiently*".

Query Processing Architecture



Query Optimizer

- Uses heuristic algorithms to evaluate relational algebra expressions. This involves:
  - estimating the cost of a relational algebra expression
  - transforming one relational algebra expression to an equivalent one
  - choosing access paths for evaluating the subexpressions
- Query optimizers do not "optimize" – just try to find "reasonably good" evaluation strategies

Equivalence Preserving Transformations

- To transform a relational expression into another equivalent expression we need transformation rules that preserve equivalence
- Each transformation rule
  - Is provably correct (ie, does preserve equivalence)
  - Has a heuristic associated with it

Selection and Projection Rules

- Break complex selection into simpler ones:
  - $\sigma_{Cond1 \wedge Cond2}(R) \equiv \sigma_{Cond1}(\sigma_{Cond2}(R))$
- Break projection into stages:
  - $\pi_{attr}(R) \equiv \pi_{attr}(\pi_{attr'}(R))$, if $attr \subseteq attr'$
- Commute projection and selection:
  - $\pi_{attr}(\sigma_{Cond}(R)) \equiv \sigma_{Cond}(\pi_{attr}(R))$,

    if $attr \supseteq$ all attributes in *Cond*

Commutativity and Associativity of Join
(and Cartesian Product as Special Case)

- Join commutativity: $R \bowtie S \equiv S \bowtie R$
  - used to reduce cost of nested loop evaluation strategies (smaller relation should be in outer loop)
- Join associativity: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$
  - used to reduce the size of intermediate relations in computation of multi-relational join – first compute the join that yields smaller intermediate result
- N-way join has $T(N) \times N!$ different evaluation plans
  - $T(N)$ is the number of parenthesized expressions
  - $N!$ is the number of permutations
- Query optimizer <u>cannot</u> look at all plans (might take longer to find an optimal plan than to compute query brute-force). Hence it does not necessarily produce optimal plan

Pushing Selections and Projections

- $\sigma_{Cond}(R \times S) \equiv R \bowtie_{Cond} S$
  - *Cond* relates attributes of both R and S
  - Reduces size of intermediate relation since rows can be discarded sooner
- $\sigma_{Cond}(R \times S) \equiv \sigma_{Cond}(R) \times S$
  - *Cond* involves only the attributes of R
  - Reduces size of intermediate relation since rows of R are discarded sooner
- $\pi_{attr}(R \times S) \equiv \pi_{attr}(\pi_{attr'}(R) \times S)$,

  if $attributes(R) \supseteq attr' \supseteq attr \cap attributes(R)$
  - reduces the size of an operand of product

Equivalence Example

- $\sigma_{C1 \wedge C2 \wedge C3}(R \times S) \equiv \sigma_{C1}(\sigma_{C2}(\sigma_{C3}(R \times S)))$

  $\equiv \sigma_{C1}(\sigma_{C2}(R) \times \sigma_{C3}(S))$ $\equiv$

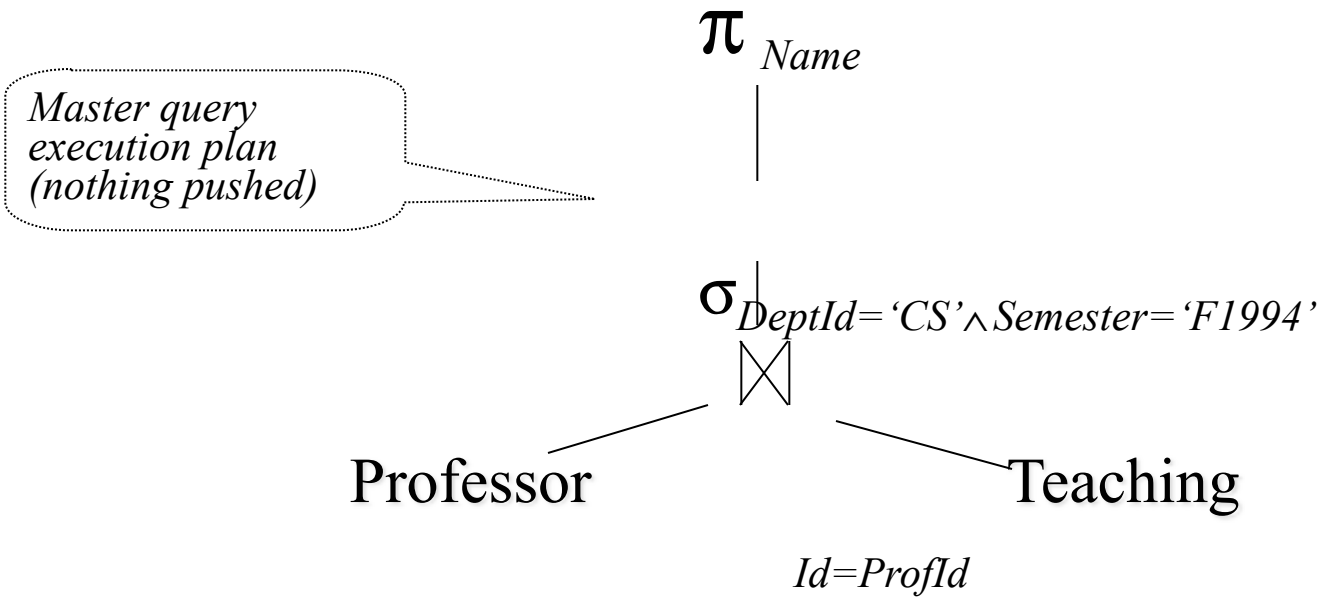
  $\sigma_{C2}(R) \bowtie_{C1} \sigma_{C3}(S)$

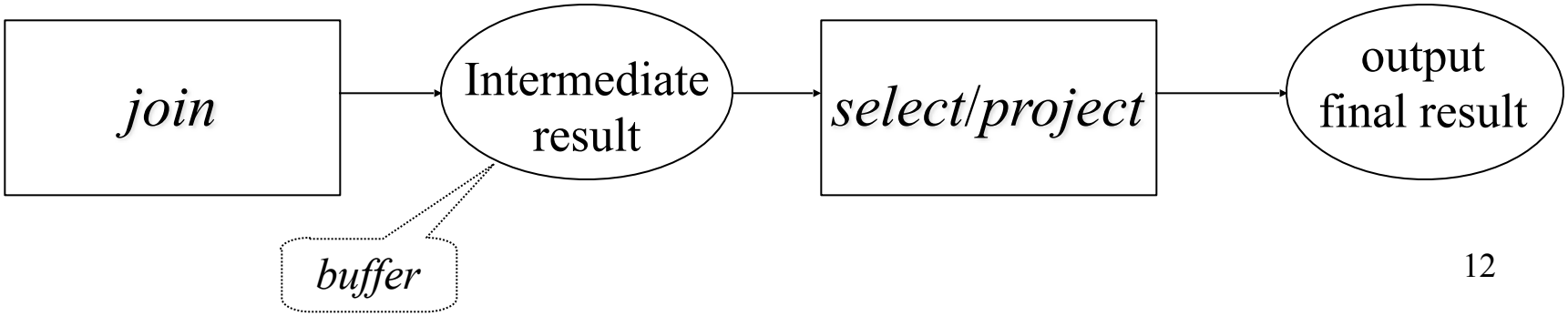
  assuming *C2* involves only attributes of R,
  C3 involves only attributes of S,
  and *C1* relates attributes of R and S

Cost - Example 1

SELECT P.*Name*
FROM Professor P, Teaching T
WHERE P.Id = T.*ProfId* -- *join condition*
AND P. *DeptId* = 'CS' AND T.*Semester* = 'F1994'

$$\pi_{Name}(\sigma_{DeptId='CS' \wedge Semester='F1994'}(\text{Professor} \bowtie_{Id=ProfId} \text{Teaching}))$$



Pipelining

- **Solution**: use *pipelining:*
  - *join* and *select*/*project* act as coroutines, operate as producer/consumer sharing a buffer in main memory.
    - When join fills buffer; *select*/*project* filters it and outputs result
    - Process is repeated until *select*/*project* has processed last output from join
  - Performing select/project adds no additional cost