# Query processing and optimization

# Definitions

- Query processing
  - translation of query into low-level activities
  - evaluation of query
  - data extraction
- Query optimization
  - selecting the most efficient query evaluation

# Query Processing (1/2)

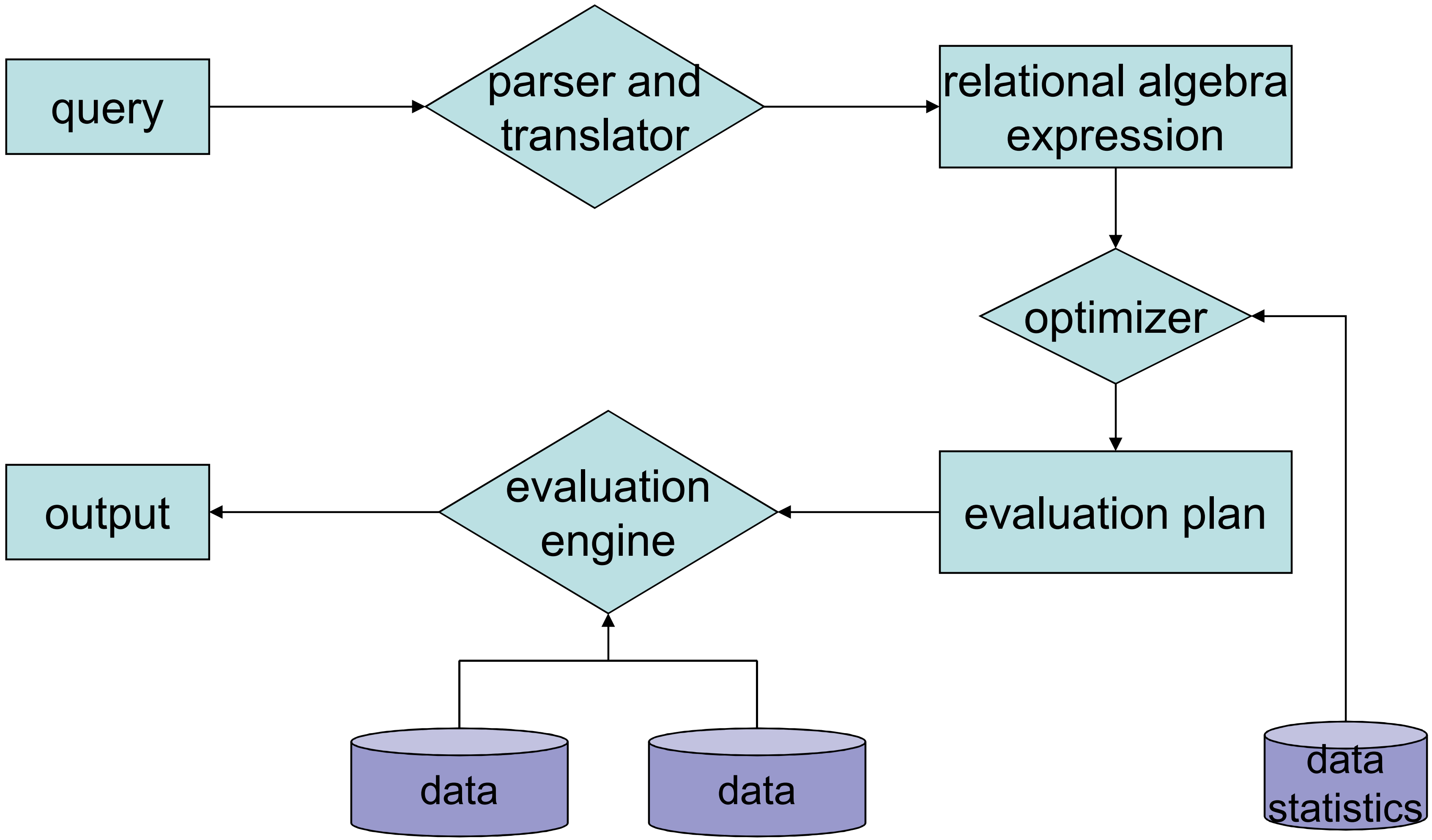
- SELECT * FROM student WHERE name=Paul
- Parse query and translate
  - check syntax, verify names, etc
  - translate into relational algebra (RDBMS)
  - create **evaluation plans**
- Find best plan (optimization)
- Execute plan

| student | |
|---|---|
| cid | name |
| 00112233 | Paul |
| 00112238 | Rob |
| 00112235 | Matt |

| takes | |
|---|---|
| cid | courseid |
| 00112233 | 312 |
| 00112233 | 395 |
| 00112235 | 312 |

| course | |
|---|---|
| courseid | coursename |
| 312 | Advanced DBs |
| 395 | Machine Learning |

# Query Processing (2/2)



# Relational Algebra (1/2)

- Query language
- Operations:
  - select: $\sigma$
  - project: $\pi$
  - union: $\cup$
  - difference: -
  - product: x
  - join: $\bowtie$

# Relational Algebra (2/2)

- SELECT * FROM student WHERE name=Paul
  - $\sigma_{name=Paul}(student)$
- $\pi_{name}(\ \sigma_{cid<00112235}(student)\ )$
- $\pi_{name}(\sigma_{coursename=Advanced\ DBs}((student \bowtie_{cid} takes) \bowtie_{courseid} course)\ )$

| student | |
|---|---|
| cid | name |
| 00112233 | Paul |
| 00112238 | Rob |
| 00112235 | Matt |

| takes | |
|---|---|
| cid | courseid |
| 00112233 | 312 |
| 00112233 | 395 |
| 00112235 | 312 |

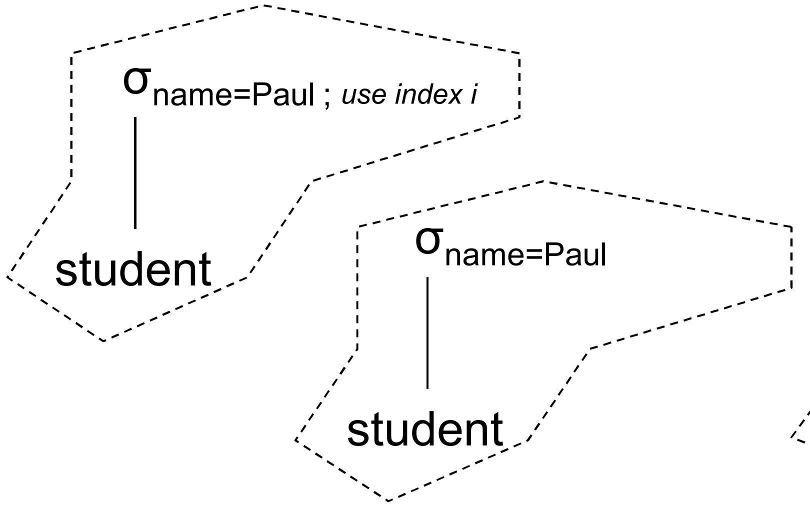
| course | |
|---|---|
| courseid | coursename |
| 312 | Advanced DBs |
| 395 | Machine Learning |

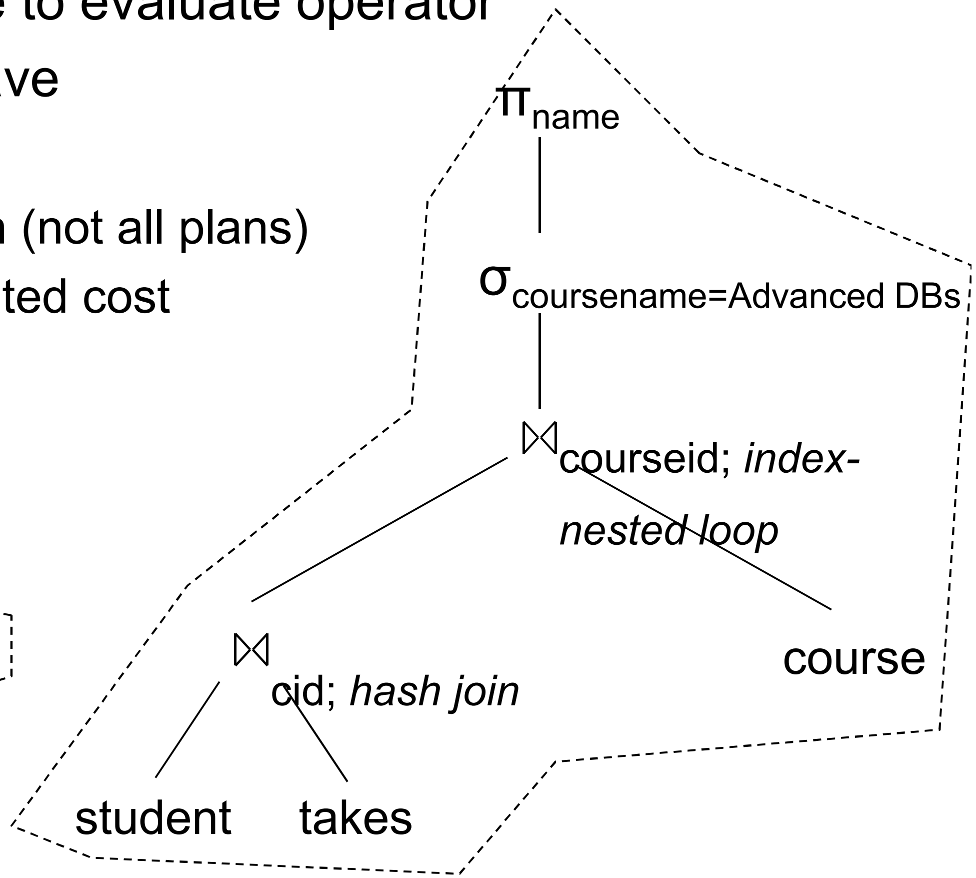
# Why Optimize?

- Many alternative options to evaluate a query
  - $\pi_{name}(\sigma_{coursename=Advanced\ DBs}((student \bowtie_{cid} takes) \bowtie_{courseid} course))$
  - $\pi_{name}((student \bowtie_{cid} takes) \bowtie_{courseid} \sigma_{coursename=Advanced\ DBs}(course)))$
- Several options to evaluate a single operation
  - $\sigma_{name=Paul}(student)$
    - scan file
    - use secondary index on student.name
- Multiple access paths
  - access path: how can records be accessed

# Evaluation plans

- Specify which access path to follow
- Specify which algorithm to use to evaluate operator
- Specify how operators interleave
- Optimization:
  - estimate the cost of each plan (not all plans)
  - select plan with lowest estimated cost



# Estimating Cost

- What needs to be considered:
  - Disk I/Os
    - sequential
    - random
  - CPU time
  - Network communication
- What are we going to consider:
  - Disk I/Os
    - page reads/writes
  - Ignoring cost of writing final output

# Operations and Costs

# Operations and Costs (1/2)

- Operations: σ, π, ∪, ∩, -, x, ⋈
- Costs:
  - $N_R$: number of records in R
  - $L_R$: size of record in R
  - $F_R$: blocking factor
    - number of records in page
  - $B_R$: number of pages to store relation R
  - V(A,R): number of distinct values of attribute A in R
  - SC(A,R): selection cardinality of A in R
    - A key: S(A,R)=1
    - A nonkey: S(A,R)= $N_R$ / V(A,R)
  - $HT_i$: number of levels in index I
  - rounding up fractions and logarithms

# Operations and Costs (2/2)

- relation *takes*
  - 700 tuples
  - student cid 8 bytes
  - course id 4 bytes
  - 9 courses
  - 100 students
  - page size 512 bytes
  - output size (in pages) of query: which students take the Advanced DBs course?
    - $N_{takes} = 700$
    - V(courseid, takes) = 9
    - SC(courseid,takes) = ceil( $N_{takes}$/V(courseid, takes) ) = ceil(700/9) = 78
    - f = floor( 512/8 ) = 64
    - B = ceil( 78/64) = 2 pages

# Selection σ (1/2)

- Linear search
  - read all pages, find records that match (assuming equality search)
  - average cost:
    - nonkey $B_R$, key $0.5*B_R$
- Binary search
  - on ordered field
  - average cost: $\left| \log_2 B_R \right| + m$
    - *m* additional pages to be read
    - $m$ = ceil( SC(A,R)/$F_R$ ) - 1
- Primary/Clustered Index
  - average cost:
    - single record $HT_i + 1$
    - multiple records $HT_i$ + ceil( SC(A,R)/$F_R$ )

# Selection σ (2/2)

- Secondary Index
  - average cost:
    - key field $HT_i + 1$
    - nonkey field
      - worst case $HT_i + SC(A,R)$
      - linear search more desirable if many matching records

# Complex selection $\sigma_{expr}$

- conjunctive selections: $\sigma_{\theta_1 \wedge \theta_2 ... \wedge \theta_n}$
  - perform simple selection using $\theta_i$ with the lowest evaluation cost
    - e.g. using an index corresponding to $\theta_i$
    - apply remaining conditions θ on the resulting records
    - $\sigma_{cid>00112233 \wedge courseid=312}(takes)$
    - cost: the cost of the simple selection on selected θ
  - multiple indices
    - select indices that correspond to $\theta_i$s
    - scan indices and return RIDs
    - answer: intersection of RIDs
    - cost: the sum of costs + record retrieval
- disjunctive selections: $\sigma_{\theta_1 \vee \theta_2 ... \vee \theta_n}$
  - multiple indices
    - union of RIDs
  - linear search

# Projection and set operations

- SELECT DISTINCT cid FROM takes
  - π requires duplicate elimination
  - sorting
- set operations require duplicate elimination
  - $R \cap S$
  - $R \cup S$
  - sorting

# Sorting

- efficient evaluation for many operations
- required by query:
  - SELECT cid,name FROM student ORDER BY name
- implementations
  - internal sorting (if records fit in memory)
  - external sorting

# External Sort-Merge Algorithm (1/3)

- Sort stage: create sorted *runs*

```
i=0;
repeat
    read M pages of relation R into memory
    sort the M pages
    write them into file R_i
    increment i
until no more pages
N = i            // number of runs
```

# External Sort-Merge Algorithm (2/3)

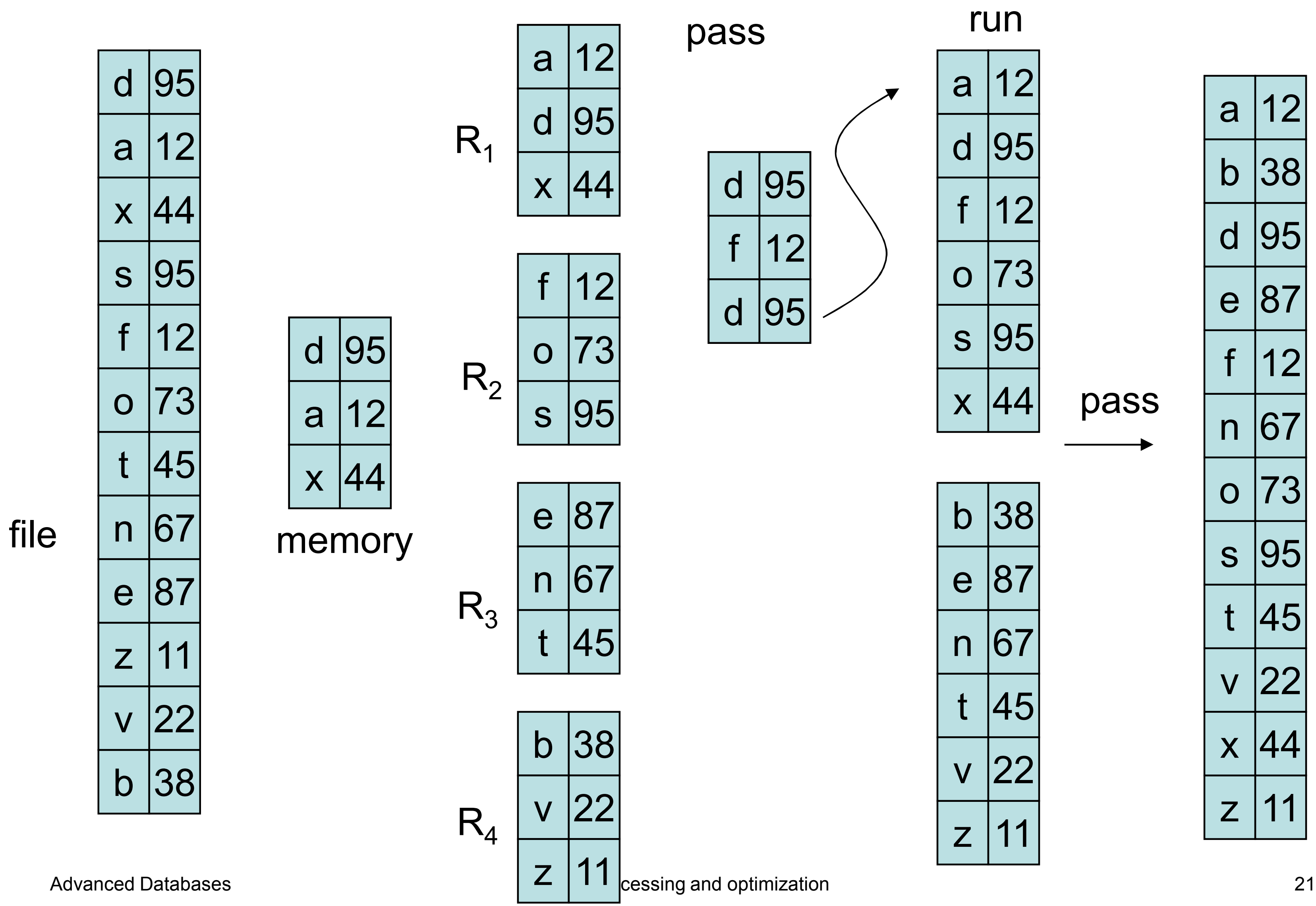
- Merge stage: merge sorted *runs*

```
//assuming N < M
allocate a page for each run file R_i        // N pages allocated
read a page P_i of each R_i
repeat
    choose first record (in sort order) among N pages, say from page P_j
    write record to output and delete from page P_j
    if page is empty read next page P_j' from R_j
until all pages are empty
```

# External Sort-Merge Algorithm (3/3)

- Merge stage: merge sorted *runs*
- What if N > M ?
  - perform multiple *passes*
  - each *pass* merges M-1 runs until relation is processed
  - in next pass number of runs is reduced
  - final *pass* generated sorted output

# Sort-Merge Example



# Sort-Merge cost

- $B_R$ the number of pages of R
- Sort stage: 2 * $B_R$
  - read/write relation
- Merge stage:
  - initially $\left\lceil \frac{B_R}{M} \right\rceil$ runs to be merged
  - each *pass* M-1 runs sorted
  - thus, total number of passes: $\left\lceil \log_{M-1}\left(\frac{B_R}{M}\right) \right\rceil$
  - at each pass 2 * $B_R$ pages are read
    - read/write relation
    - apart from final write
- Total cost:
  - $2 * B_R + 2 * B_R * \left\lceil \log_{M-1}\left(\frac{B_R}{M}\right) \right\rceil - B_R$

# Projection

- $\pi_{A1,A2\ldots}$ (R)
- remove unwanted attributes
  - scan and drop attributes
- remove duplicate records
  - sort resulting records using all attributes as sort order
  - scan sorted result, eliminate duplicates (adjucent)
- cost
  - initial scan + sorting + final scan

# Join

- $\pi_{name}(\sigma_{coursename=Advanced\ DBs}((student \bowtie_{cid} takes) \bowtie_{courseid} course))$
- implementations
  - nested loop join
  - block-nested loop join
  - indexed nested loop join
  - sort-merge join
  - hash join

# Nested loop join (1/2)

- R $\bowtie$ S

```
for each tuple t_R of R
        for each t_S of S
                if (t_R t_S match) output t_R.t_S
        end
end
```

- Works for any join condition
- S inner relation
- R outer relation

# Nested loop join (2/2)

- Costs:
  - best case when smaller relation fits in memory
    - use it as inner relation
    - $B_R + B_S$
  - worst case when memory holds one page of each relation
    - S scanned for each tuple in R
    - $N_R * B_s + B_R$

# Block nested loop join (1/2)

```
for each page X_R of R
    foreach page X_S of S
        for each tuple t_R in  X_R
            for each t_S in X_S
                if (t_R t_S match) output t_R.t_S
            end
        end
    end
end
```

# Block nested loop join (2/2)

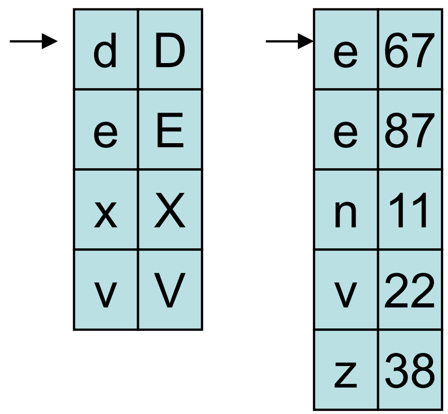
- Costs:
  - best case when smaller relation fits in memory
    - use it as inner relation
    - $B_R + B_S$
  - worst case when memory holds one page of each relation
    - S scanned for each page in R
    - $B_R * B_s + B_R$

# Indexed nested loop join

- $R \bowtie S$
- Index on inner relation (S)
- for each tuple in outer relation (R) *probe* index of inner relation
- Costs:
  - $B_R + N_R * c$
    - c the cost of index-based selection of inner relation
  - relation with fewer records as outer relation

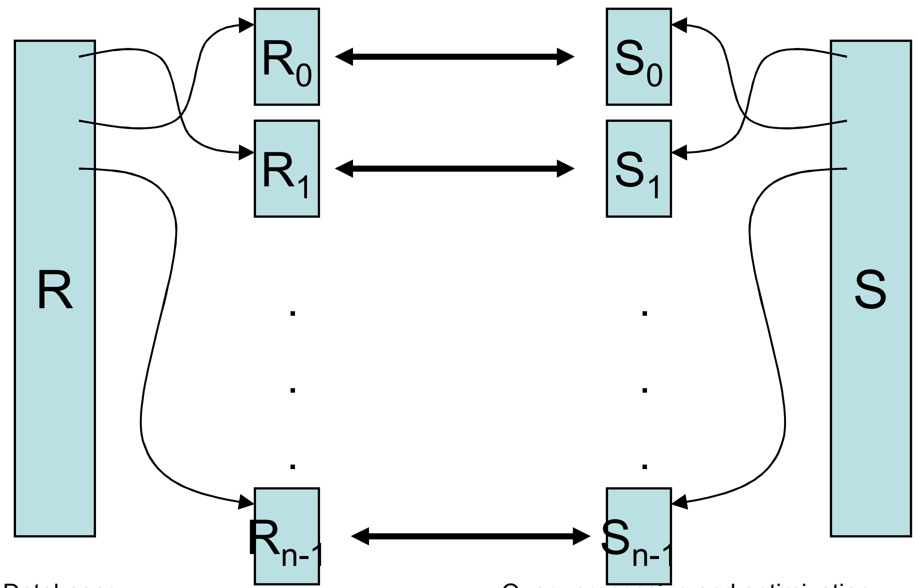
# Sort-merge join

- R ⋈ S
- Relations sorted on the join attribute
- Merge sorted relations
  - pointers to first record in each relation
  - read in a group of records of S with the same values in the join attribute
  - read records of R and process
- Relations in sorted order to be read once
- Cost:
  - cost of sorting + $B_S$ + $B_R$



# Hash join

- R ⋈ S
- use $h_1$ on joining attribute to map records to partitions that fit in memory
  - records of R are partitioned into $R_0 \ldots R_{n-1}$
  - records of S are partitioned into $S_0 \ldots S_{n-1}$
- join records in corresponding partitions
  - using a hash-based indexed block nested loop join
- Cost: $2*(B_R+B_S) + (B_R+B_S)$

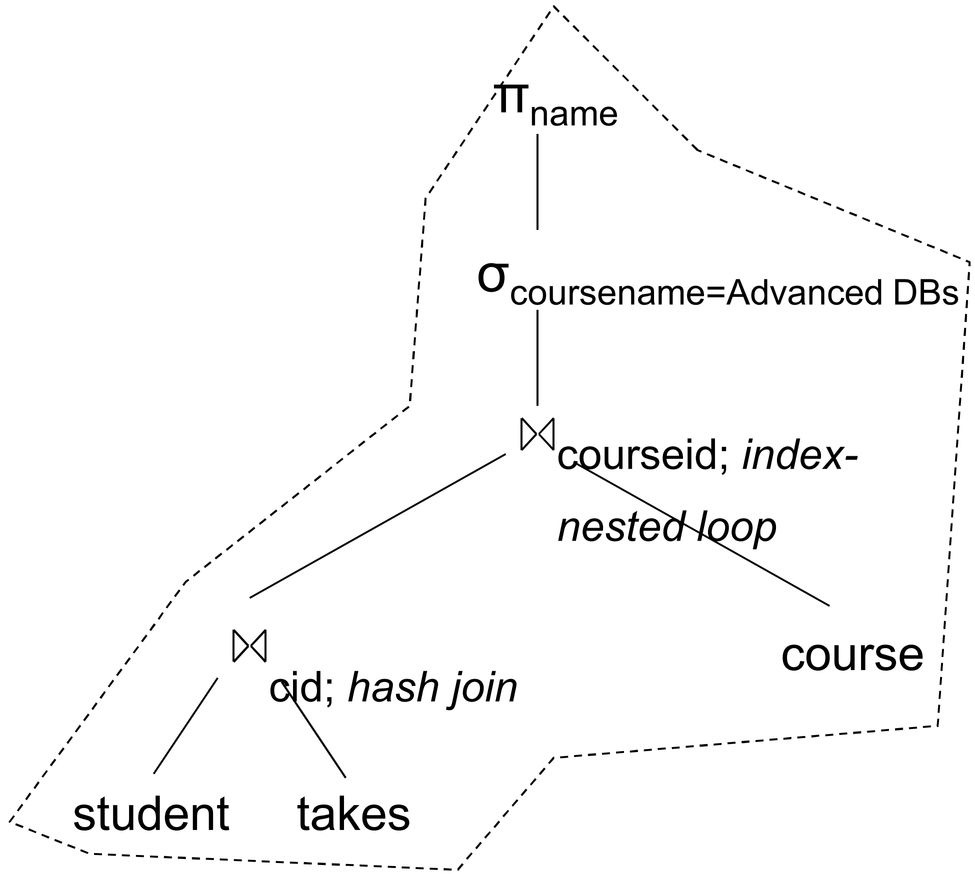


# Exercise: joins

- $R \bowtie S$
- $N_R=2^{15}$
- $B_R = 100$
- $N_S=2^6$
- $B_S = 30$
- $B^+$ index on S
  - order 4
  - full nodes
- nested loop join: best case - worst case
- block nested loop join: best case - worst case
- indexed nested loop join

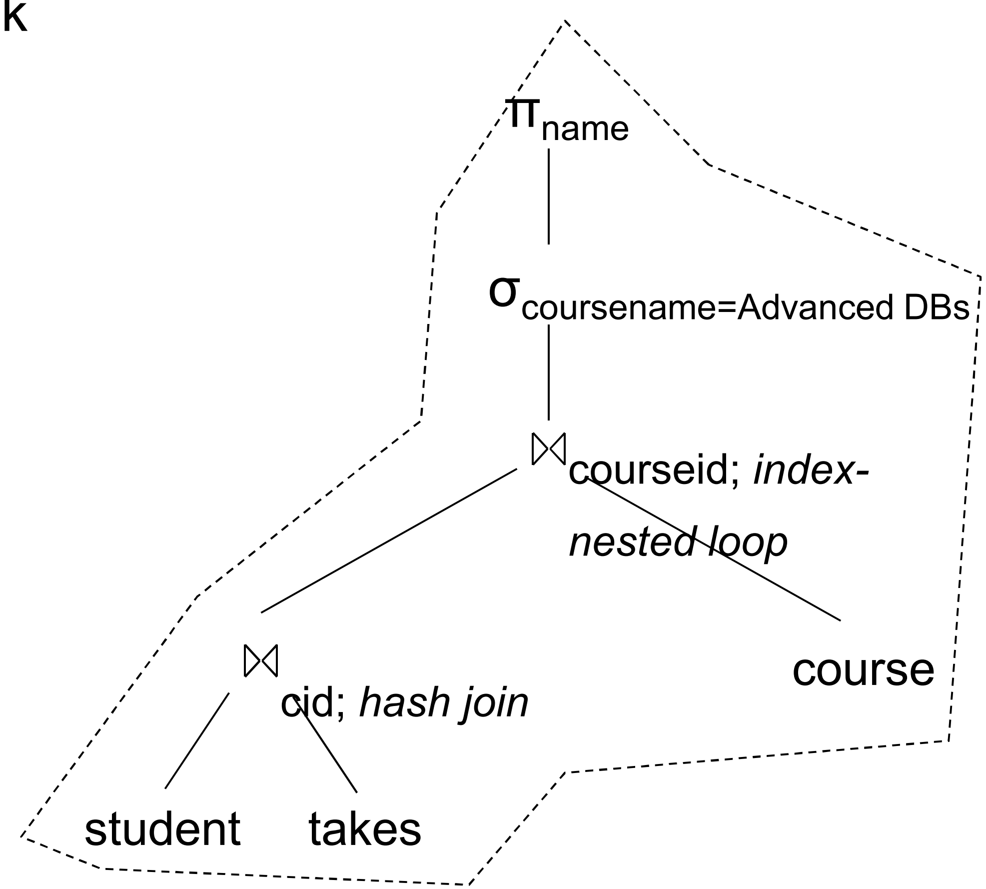
# Evaluation

- evaluate multiple operations in a plan
- materialization
- pipelining



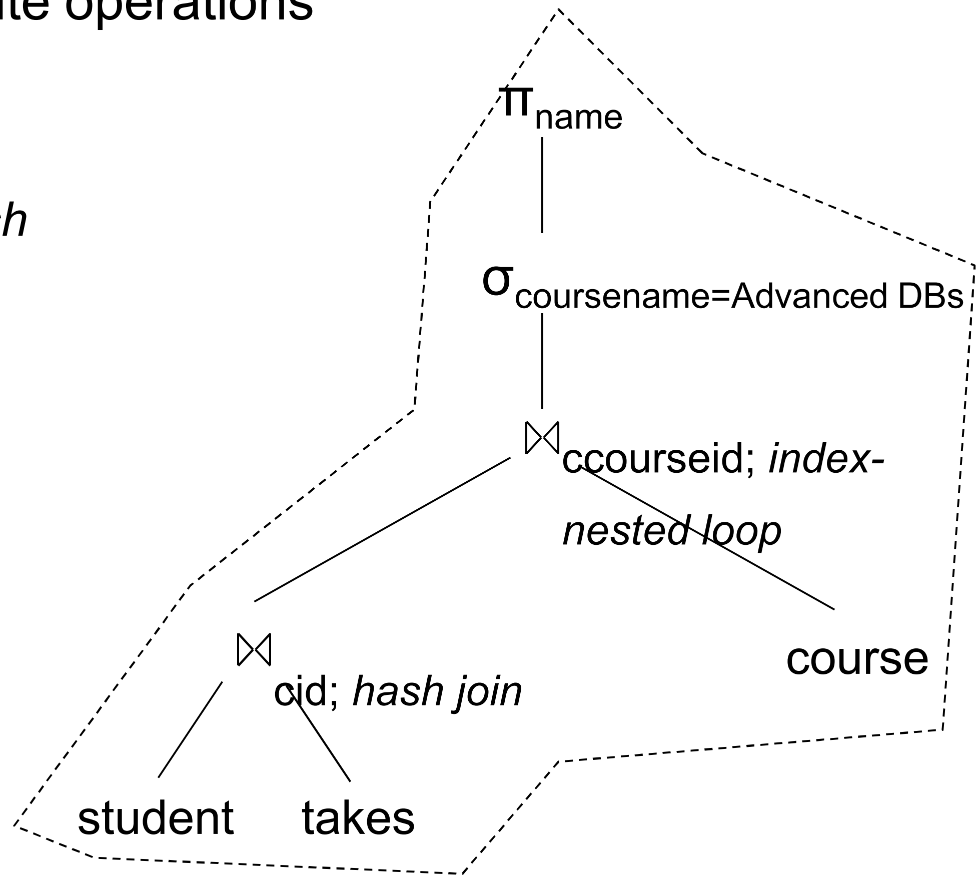
# Materialization

- create and read temporary relations
- create implies writing to disk
  - more page writes



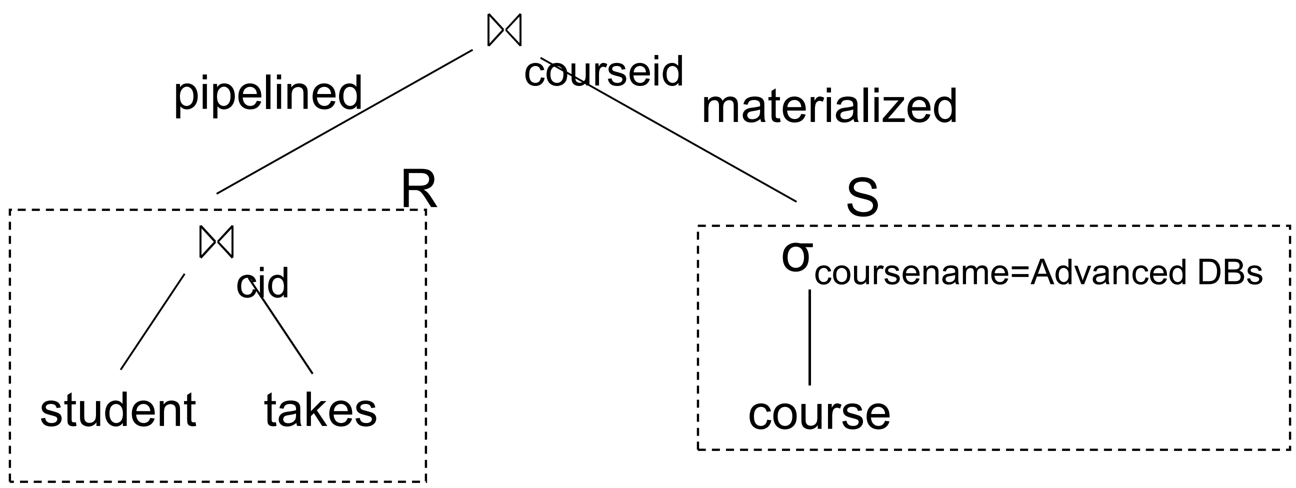
# Pipelining (1/2)

- creating a pipeline of operations
- reduces number of read-write operations
- implementations
  - demand-driven - data *pull*
  - producer-driven - data *push*



# Pipelining (2/2)

- can pipelining always be used?
- any algorithm?
- cost of R ⋈ S
  - materialization and hash join: $B_R + 3(B_R+B_S)$
  - pipelining and indexed nested loop join: $N_R * HT_i$



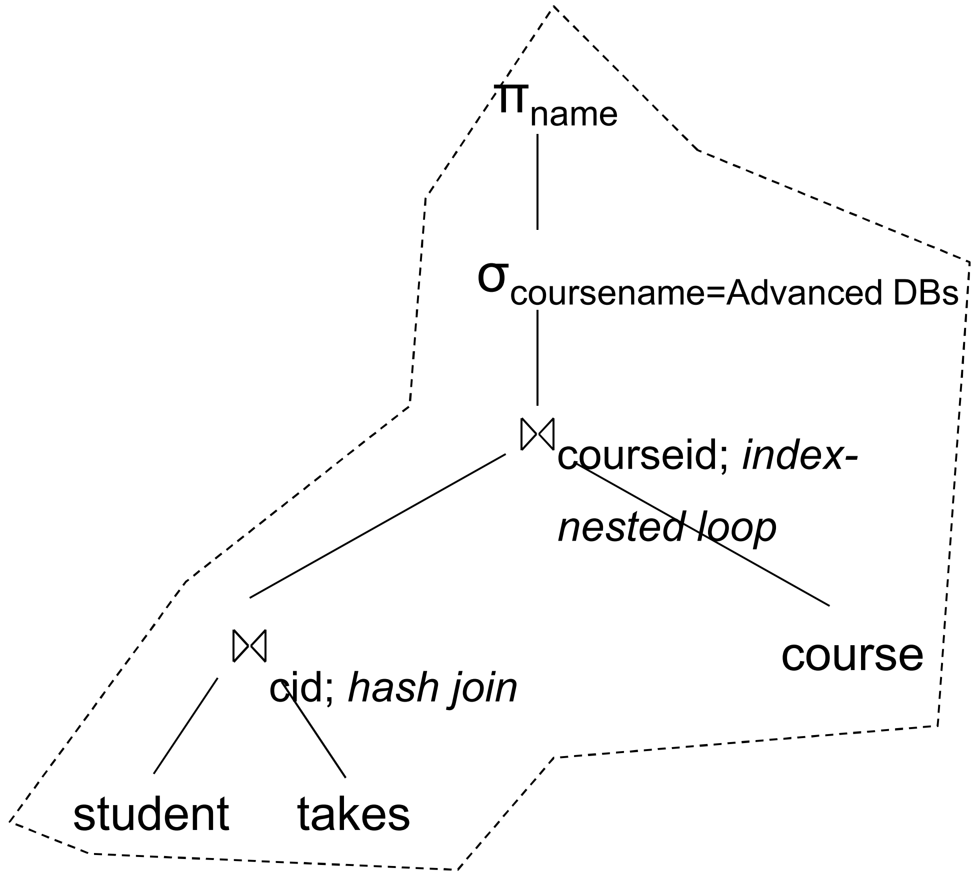
# Query Optimization

# Choosing evaluation plans

- cost based optimization
- enumeration of plans
  - $R \bowtie S \bowtie T$, 12 possible orders
- cost estimation of each plan
- overall cost
  - cannot optimize operation independently

# Cost estimation

- operation (σ, π, ⋈ …)
- implementation
- size of inputs
- size of outputs
- sorting



# Size Estimation (1/2)

- $\sigma_{A=v}(R)$
  - SC(A,R)
- $\sigma_{A \leq v}(R)$
  - $N_R * \frac{v - \min(A,R)}{\max(A,R) - \min(A,R)}$
- $\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n}(R)$
  - multiplying probabilities
  - $N_R * [(s_1/N_R) * (s_2/N_R) * \ldots (s_n/N_R)]$
- $\sigma_{\theta_1 \vee \theta_2 v \ldots \vee \theta_n}(R)$
  - probability that a record satisfy none of θ:
    $[(1 - s_1/N_R) * (1 - s_2/N_R) * \ldots * (1 - s_n/N_R)]$
  - $N_R * (1 - [(1 - s_1/N_R) * (1 - s_2/N_R) * \ldots * (1 - s_n/N_R)])$

# Size Estimation (2/2)

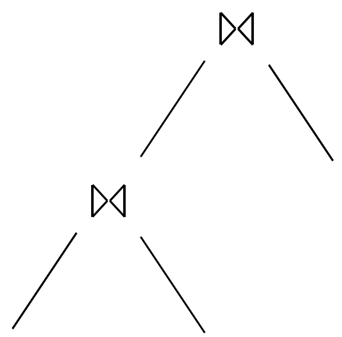
- R x S
  - $N_R * N_S$
- R⋈S
  - $R \cap S = \emptyset$: $N_R * N_S$
  - $R \cap S$ key for R: maximum output size is $N_s$
  - $R \cap S$ foreign key for R: $N_S$
  - $R \cap S = \{A\}$, neither key of R nor S
    - $N_R * N_S / V(A,S)$
    - $N_S * N_R / V(A,R)$

# Expression Equivalence

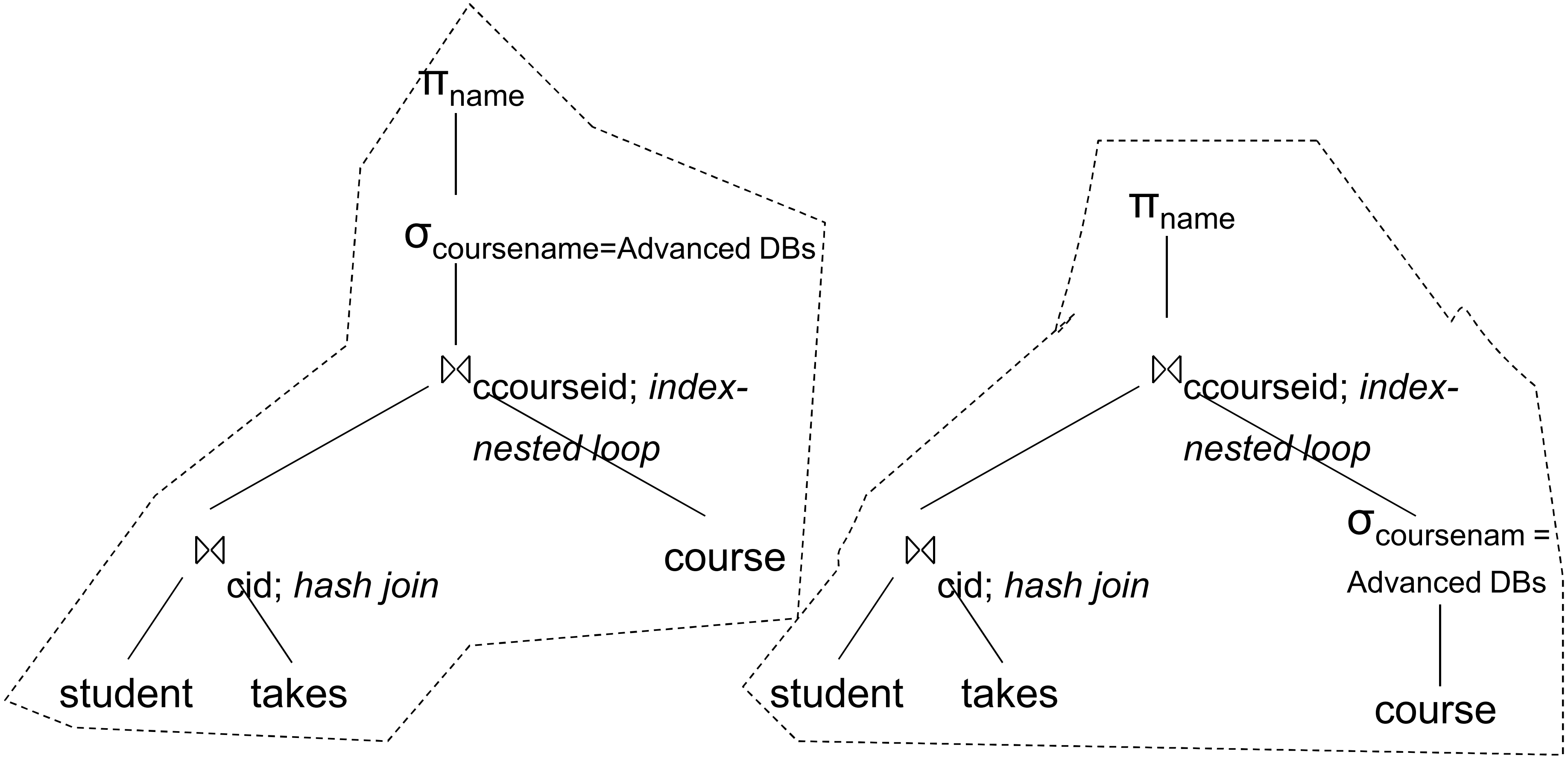
- conjunctive selection decomposition
  - $\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R))$
- commutativity of selection
  - $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$
- combining selection with join and product
  - $\sigma_{\theta 1}(R \times S) = R \bowtie_{\theta 1} S$
- commutativity of joins
  - $R \bowtie_{\theta 1} S = S \bowtie_{\theta 1} R$
- distribution of selection over join
  - $\sigma_{\theta 1 \wedge \theta 2}(R \bowtie S) = \sigma_{\theta 1}(R) \bowtie \sigma_{\theta 2}(S)$
- distribution of projection over join
  - $\pi_{A1,A2}(R \bowtie S) = \pi_{A1}(R) \bowtie \pi_{A2}(S)$
- associativity of joins: $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

# Cost Optimizer (1/2)

- transforms expressions
  - equivalent expressions
  - heuristics, *rules of thumb*
    - perform selections early
    - perform projections early
    - replace products followed by selection σ (R x S) with joins R⋈S
    - start with joins, selections with smallest result
      - create **left-deep join trees**



# Cost Optimizer (2/2)



# Cost Evaluation Exercise

- $\pi_{name}(\sigma_{coursename=Advanced\ DBs}((student \bowtie_{cid} takes) \bowtie_{courseid} course))$
- $R = student \bowtie_{cid} takes$
- S = course
- $N_S$ = 10 records
- assume that on average there are 50 students taking each course
- blocking factor: 2 records/page
- what is the cost of $\sigma_{coursename=Advanced\ DBs}(R \bowtie_{courseid} S)$
- what is the cost of $R \bowtie \sigma_{coursename=Advanced\ DBs} S$
- assume relations can fit in memory

# Summary

- Estimating the cost of a single operation
- Estimating the cost of a query plan
- Optimization
  - choose the most efficient plan