Lecture 17: Query Processing & Optimization

CS348 Spring 2025:
Introduction to Database Management

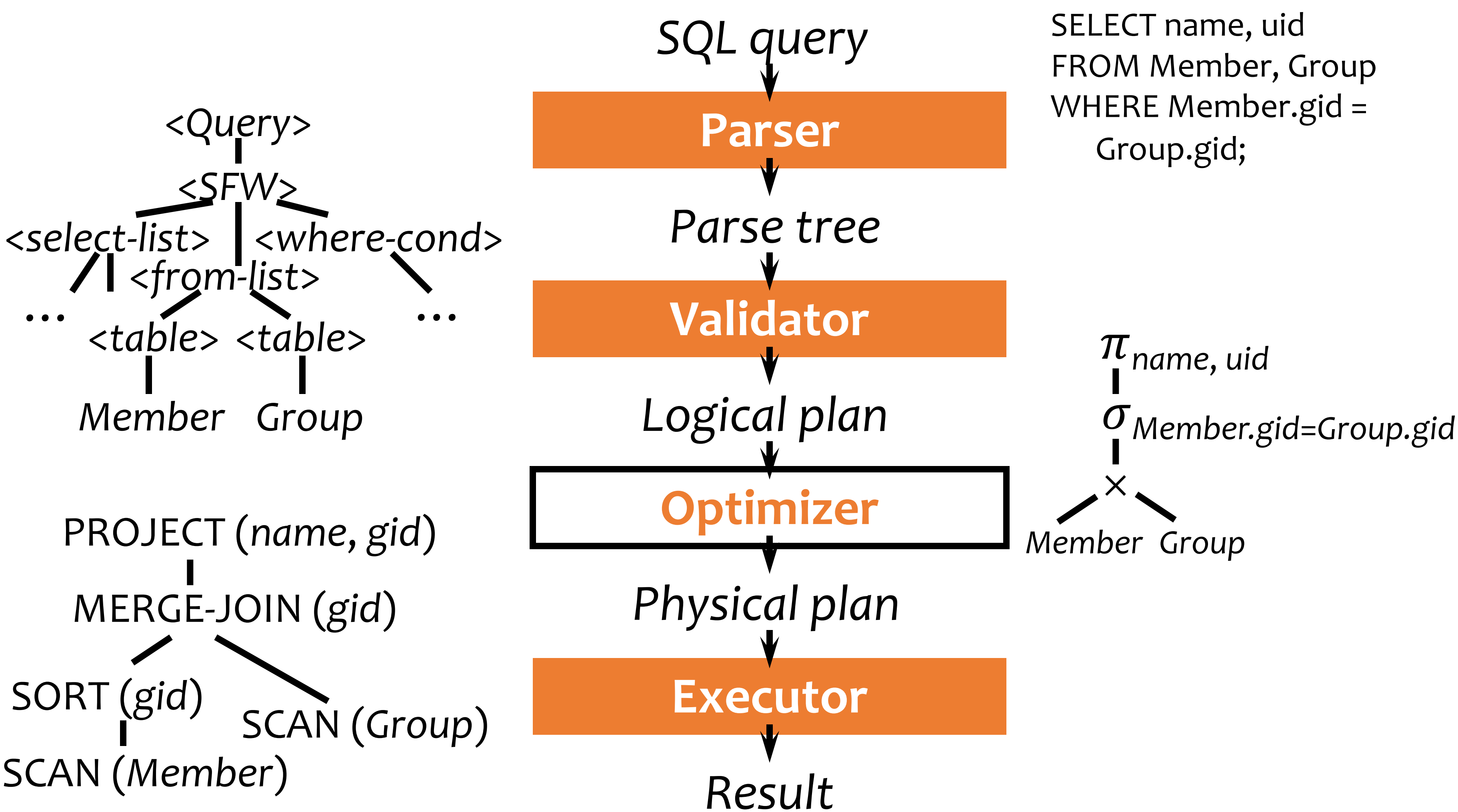
Instructor: **Xiao Hu**

Sections: 001, 002, 003

Announcements

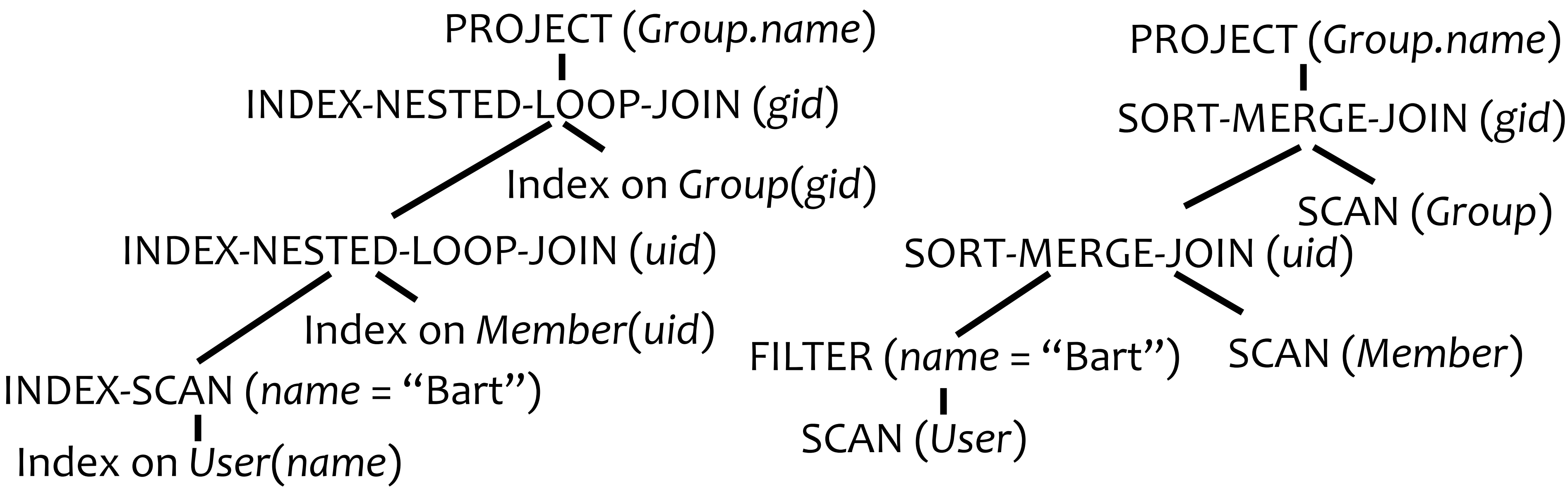
- Milestone 2 of group project
  - Due today!

A query's trip through the DBMS



(Recap) Physical plans

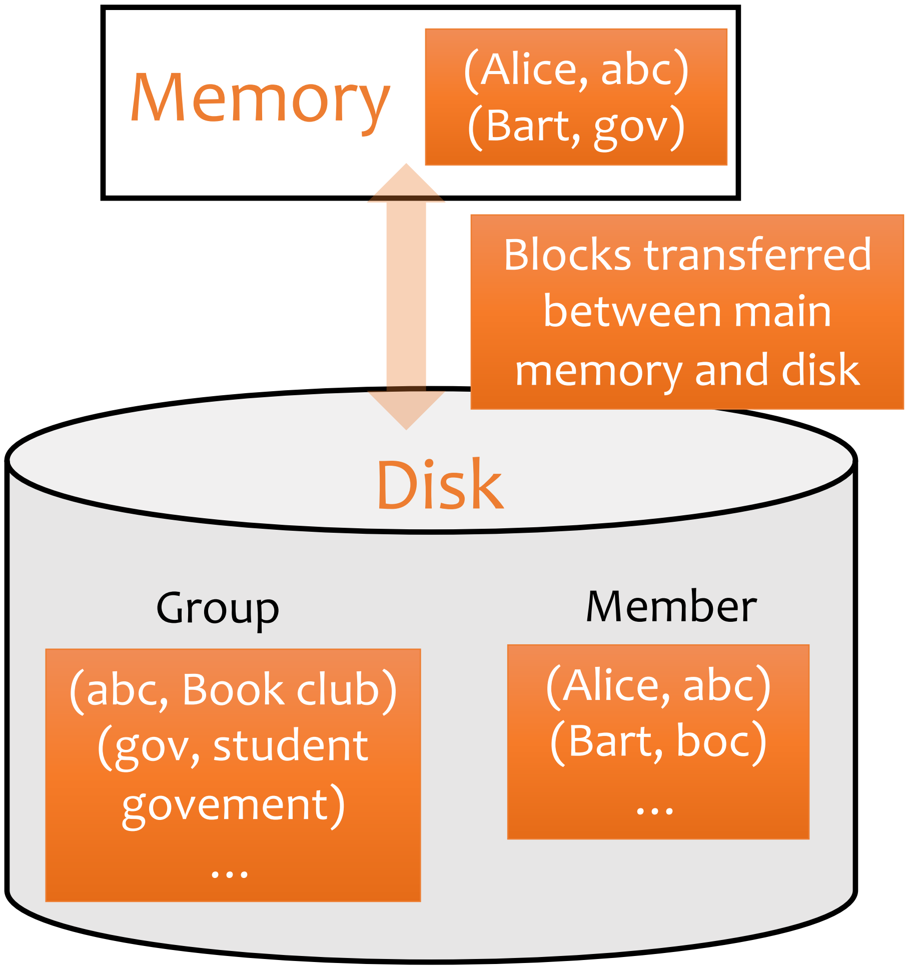
```
SELECT Group.name
FROM User, Member, Group
WHERE User.name = 'Bart'
AND User.uid = Member.uid AND Member.gid = Group.gid;
```



- Many physical plans for a single query

Outline

- Scan
  - Table scan
  - Selection, Duplicate-preserving projection
  - Nested-loop join
- Sort
  - External merge sort
  - Duplicate elimination, Grouping and Aggregation
  - Sort-merge join, Union, Difference, Intersection
- Hash
- Index



Notation and Assumption

- Relations: R, S
- Tuples: r, s
- Number of tuples: $|R|$, $|S|$
- Number of disk blocks: $B(R)$, $B(S)$
- Number of memory blocks available: M
- Cost metric
  - Number of I/O's (blocks transferred between memory and disk)
  - Memory requirement
- Not counting the cost of writing the result out
  - Same for any algorithm
  - Maybe not needed – results may be pipelined into downstream operator

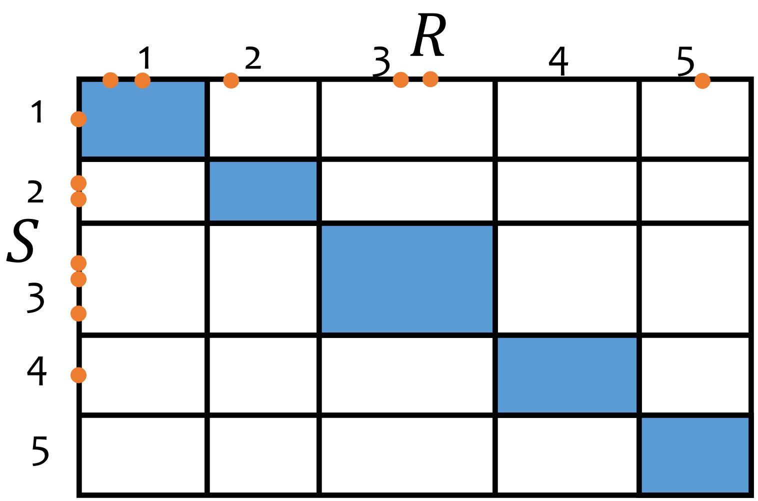
Hashing-based algorithms

http://global.rakuten.com/en/store/citygas/item/041233/

Hash join

$R \bowtie_{R.A=S.B} S$

- Main idea
  - Partition R and S by hashing their join attributes, and then consider corresponding partitions of R and S
  - If $r.A$ and $s.B$ get hashed to different partitions, they don't join

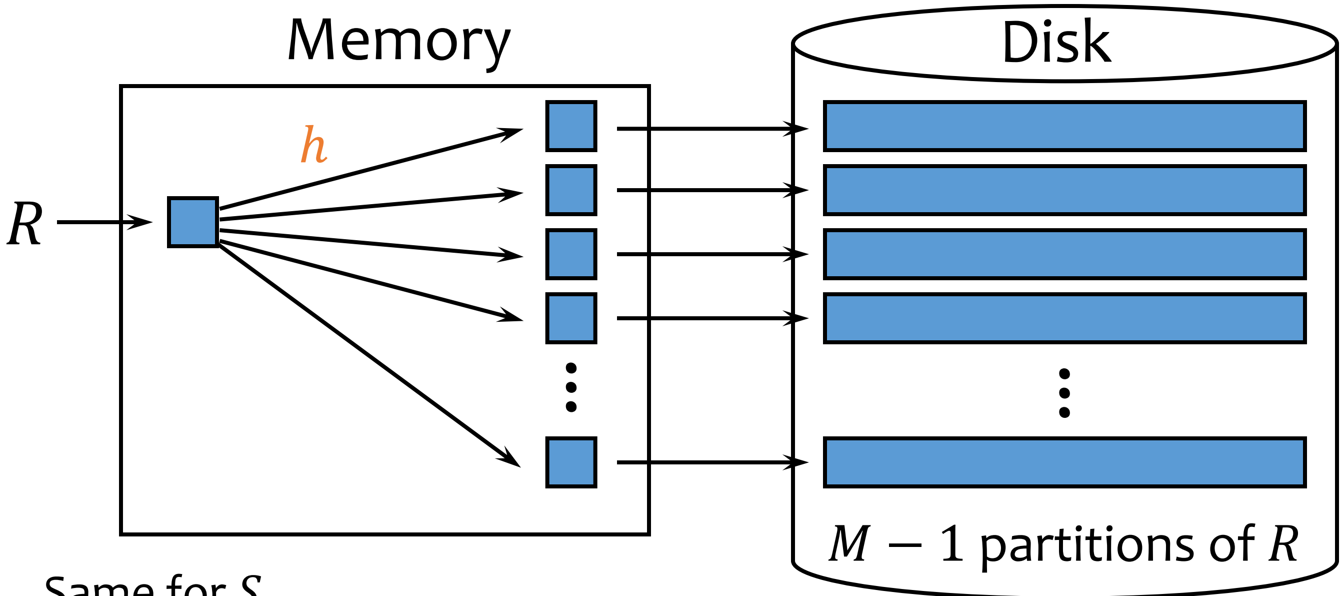


Nested-loop join considers all slots

Hash join considers only those along the diagonal!

Partitioning phase

- Partition R and S according to the same hash function on their join attributes



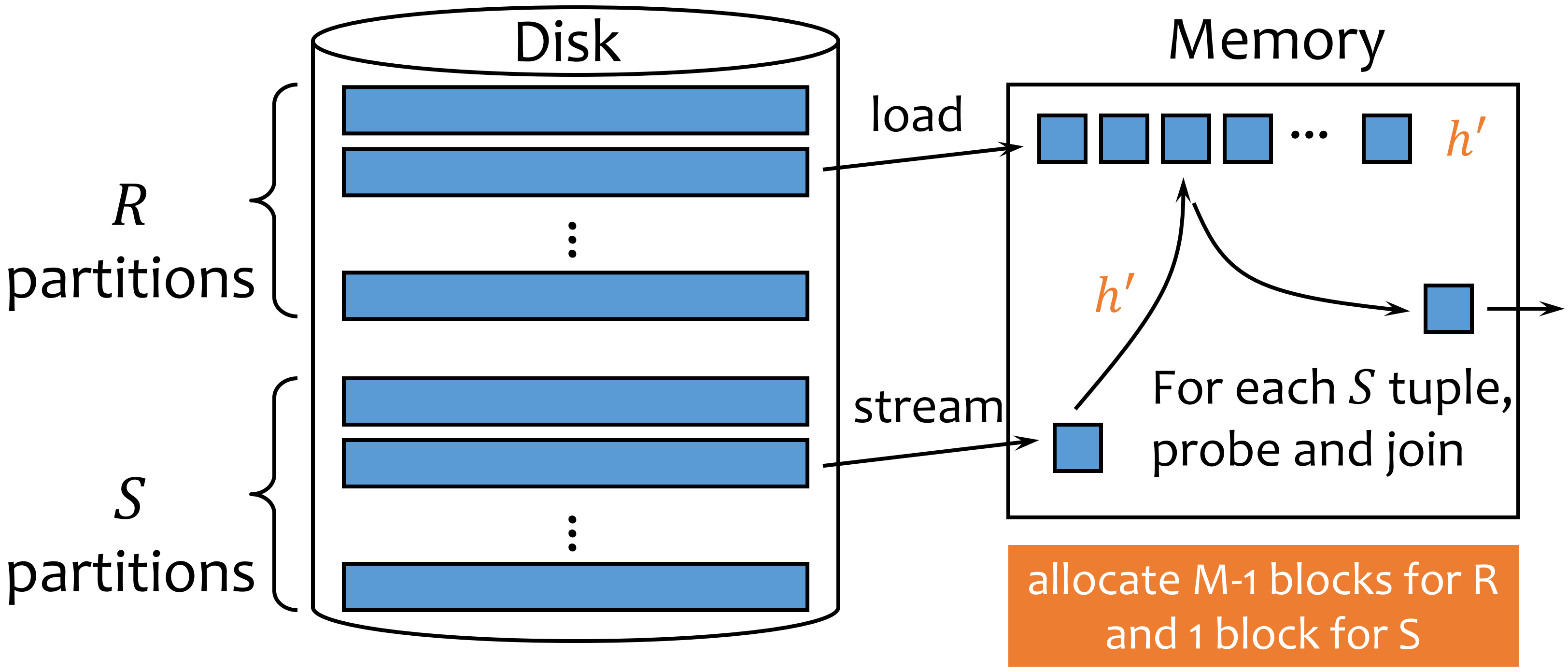
Same for S

allocate 1 for input and M-1 for output buffers

If the hash function is good, each partition has a size of B(R)/(M-1)

Probing phase

- Read in each partition of R, stream in the corresponding partition of S, join
  - Typically build a hash table for the partition of R
    - Not the same hash function used for partition, of course!



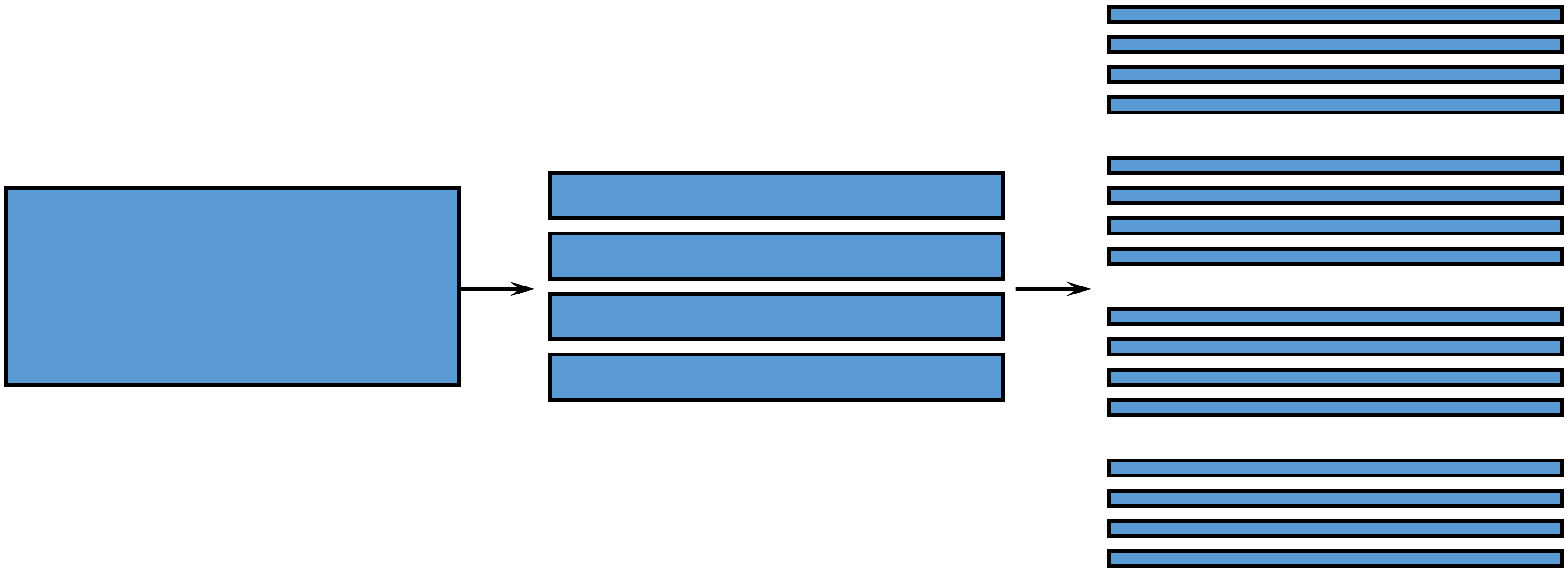
Performance of hash join

- If hash join completes in two phases:
  - I/O's: $3 \cdot \left(B(R) + B(S)\right)$
    - 1st phase: read B(R) + B(S) into memory to partition and write partitioned B(R) +B(S) to disk
    - 2nd phase: read B(R) + B(S) into memory to merge and join
  - Memory requirement:
    - In the probing phase, we should have enough memory to fit one partition of R: $M - 1 > \frac{B(R)}{M-1}$
    - $M > \sqrt{B(R)} + 1$
    - We can always pick R to be the smaller relation, so:

$$M > \sqrt{\min\left(B(R), B(S)\right)} + 1$$

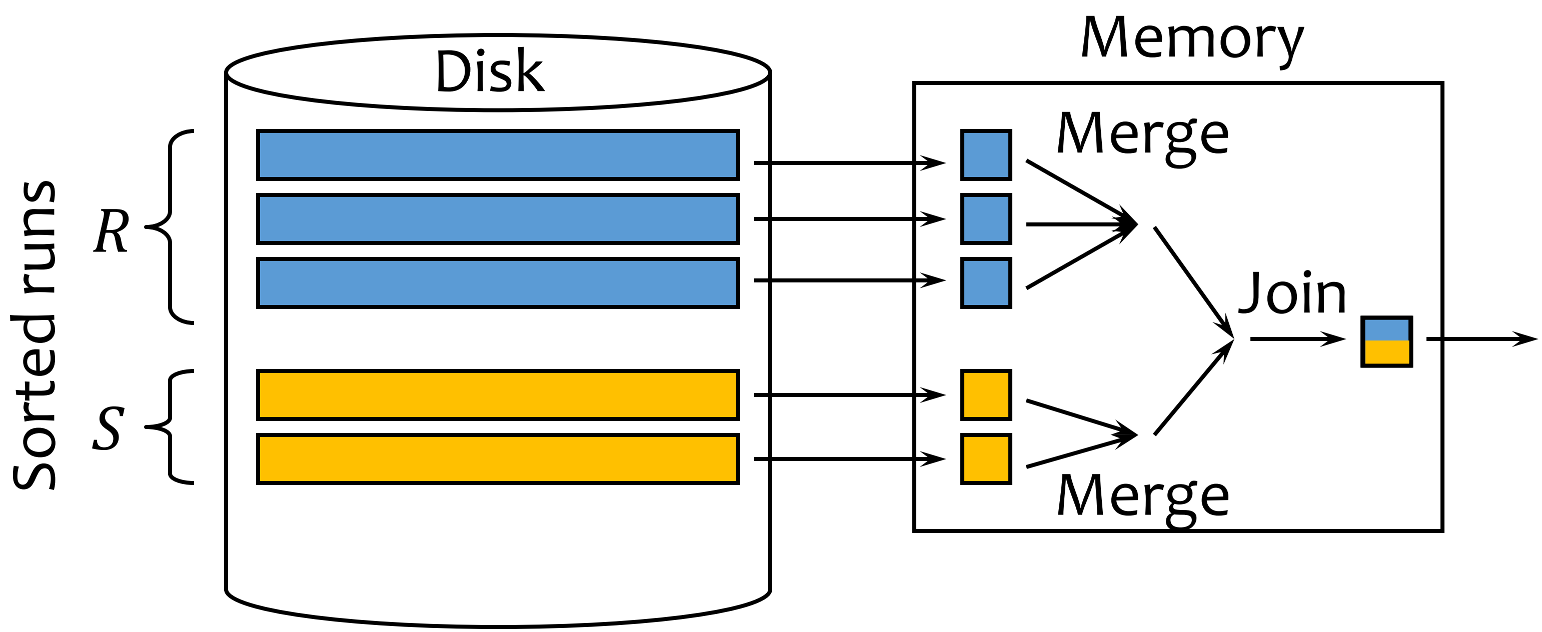
Generalizing for larger inputs

- What if a partition is too large for memory?
  - Read it back in and partition it again!
  - Re-partition $O\left(\log_M B(R)\right)$ times



(Recap) Optimized SMJ

- Produce sorted runs of R and S
- Merge the runs of R, merge the runs of S, and merge-join the result streams as they are generated

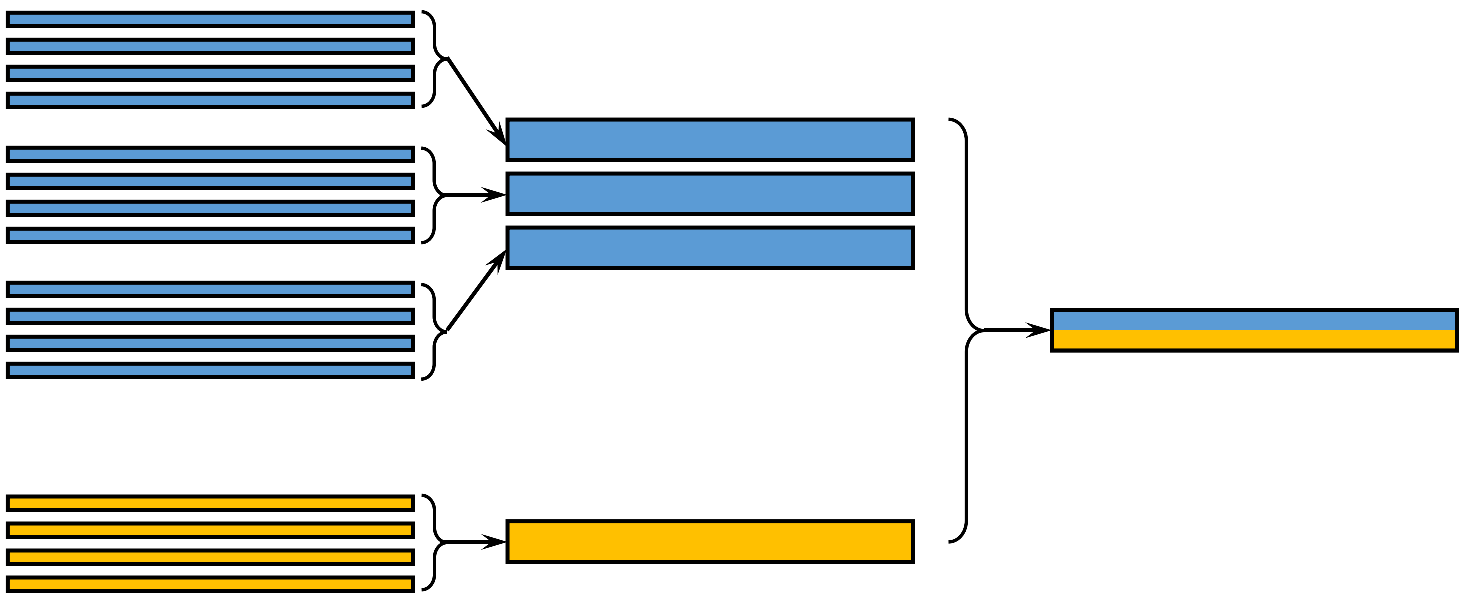


(Recap) Performance of SMJ

- If SMJ completes in two passes:
  - I/O's: $3 \cdot \big(B(R) + B(S)\big)$
    - 1st phase: read $B(R) + B(S)$ into memory to sort and write $\left\lceil \frac{B(R)}{M} \right\rceil + \lceil \frac{B(S)}{M} \rceil$ sorted runs to disk
    - 2nd phase: merge $\left\lceil \frac{B(R)}{M} \right\rceil + \lceil \frac{B(S)}{M} \rceil$ sorted runs in memory and join
  - Memory requirement
    - We must have enough memory to accommodate one block from each run: $M > \left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil$
    - Roughly $M > \sqrt{B(R) + B(S)}$

(Recap) Generalizing for larger inputs

- What if there are many number of sorted runs?
  - Repeatedly merge to reduce number of runs as necessary before final merge and join
  - Merge $O(\log_M (B(R) + B(S)))$ times



(Two-pass) Hash Join v.s. SMJ

- I/O's: same
- Memory requirement: hash join is lower
  - $\sqrt{\min\left(B(R), B(S)\right)} + 1 < \sqrt{B(R) + B(S)}$
  - Hash join wins when two relations have very different sizes
- Other factors
  - Hash join performance depends on the quality of the hash
    - Might not get evenly sized buckets
  - SMJ can be adapted for inequality join predicates
  - SMJ wins if R and/or S are already sorted
  - SMJ wins if the result needs to be in sorted order

(Multi-pass) Hash Join v.s. SMJ

For both, let I denote "input"

- # passes is $O\left(\log_M\left(\frac{B(I)}{M}\right)\right) = O(\log_M B(I))$
  - For HJ, assuming hash function is "good" enough and there is no severe data skew

- Overall I/Os is $O(B(I) \cdot \log_M B(I))$
  - For HJ
    - The partition phase takes $O(B(I) \cdot \log_M B(I))$ I/Os
    - The probing phase only takes $O\big(B(I)\big)$ I/Os
  - For SMJ, assuming no external-memory mini nested loops
    - The sorting phase takes $O(B(I) \cdot \log_M B(I))$ I/Os
    - The merge phase only takes $O\big(B(I)\big)$ I/Os

Duality of Sort and Hash

- Handling very large inputs
  - Sorting: recursive merge
  - Hashing: recursive partitioning

- I/O patterns
  - Sorting: sequential write, random read (merge)
  - Hashing: random write, sequential read (partition)

Other hash-based algorithms

- Union (set), difference, intersection
  - More or less like hash join
- Duplicate elimination
  - Check for duplicates within each partition/bucket
- Grouping and aggregation
  - Apply the hash functions to the group-by columns
  - Tuples in the same group must end up in the same partition/bucket
  - Keep a running aggregate value for each group
    - Just like in the sorting case, this trick may not always work

Outline

- Scan
  - Table scan
  - Selection, Duplicate-preserving projection
  - Nested-loop join
- Sort
  - External merge sort
  - Duplicate elimination, Grouping and Aggregation
  - Sort-merge join, Union (set), Difference, Intersection
- Hash
  - Hash join, union (set), difference, intersection, duplicate elimination, grouping and aggregation
- Index

Index-based algorithms

http://i1.trekearth.com/photos/28820/p2270994.jpg

Selection using index

- Equality predicate: $\sigma_{A=v}(R)$
  - Use an ISAM, B^+-tree, or hash index on $R(A)$

- Range predicate: $\sigma_{A>v}(R)$
  - Use an ordered index (e.g., ISAM or B^+-tree) on $R(A)$
  - Hash index is not applicable

Index versus table scan for selection

Situations where index clearly wins:

- Index-only queries which do not require retrieving actual tuples
  - Example: $\pi_A\left(\sigma_{A>v}(R)\right)$
- Primary index clustered according to search key
  - One lookup leads to all result tuples in their entirety

Index versus table scan (cont'd)

BUT(!):

- Consider $\sigma_{A>v}(R)$ and a secondary, non-clustered index on $R(A)$
  - Need to follow pointers to get the actual result tuples
  - Say that 20% of R satisfies $A > v$
  - I/O's for index-based selection: lookup + 20% $|R|$
  - I/O's for scan-based selection: $B(R)$
  - Table scan wins if a block contains more than 5 tuples!

Index nested-loop join

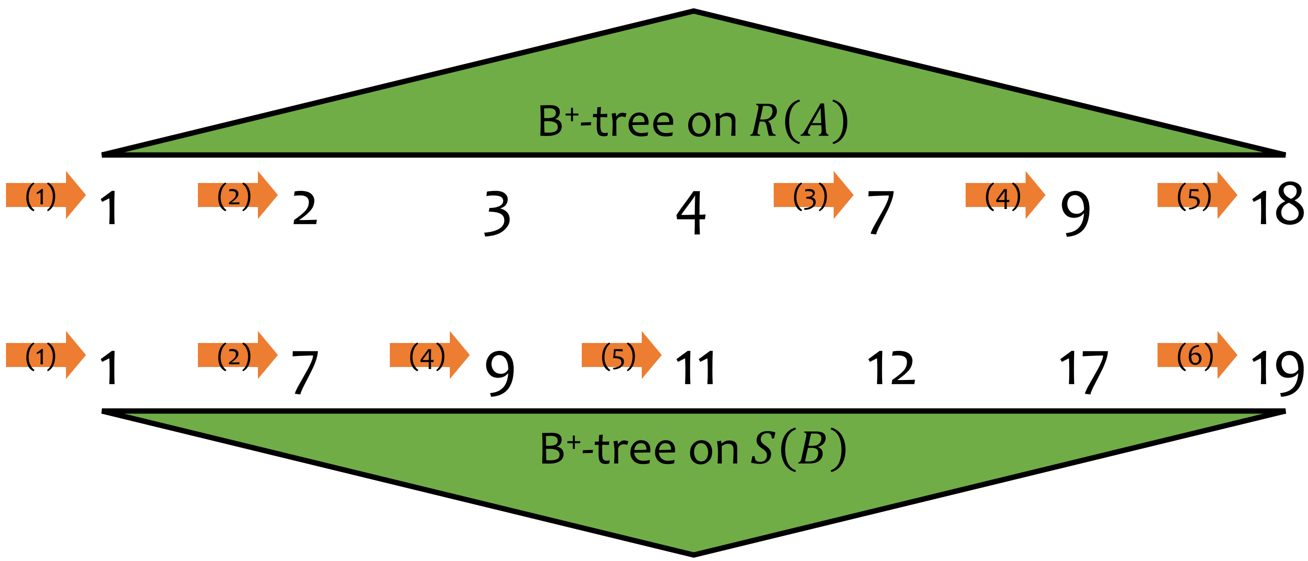
$R \bowtie_{R.A=S.B} S$

- Idea: use a value of $R.A$ to probe the index on $S(B)$
- For each block of R, and for each r in the block:
  Use the index on $S(B)$ to retrieve s with $s.B = r.A$
  Output rs
- I/O's: $B(R) + |R| \cdot \text{lookup} + \text{fetching cost}$
  - Typically, the cost of an index lookup is 2-4 I/O's
  - Beats other join methods if $|R|$ is not too big
  - Better pick R to be the smaller relation
- Memory requirement: $M \geq 3$ (blocks)

Zig-zag join using ordered indexes

$R \bowtie_{R.A=S.B} S$

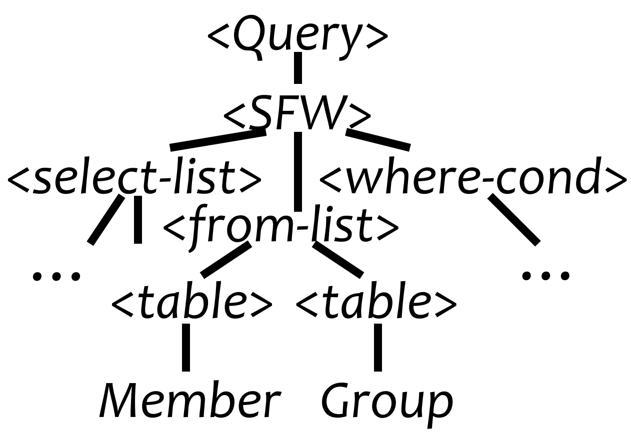
- Idea: use the ordering provided by the indexes on $R(A)$ and $S(B)$ to eliminate the sorting step of sort-merge join
- Use the larger key to probe the other index
  - Possibly skipping many keys that don't match

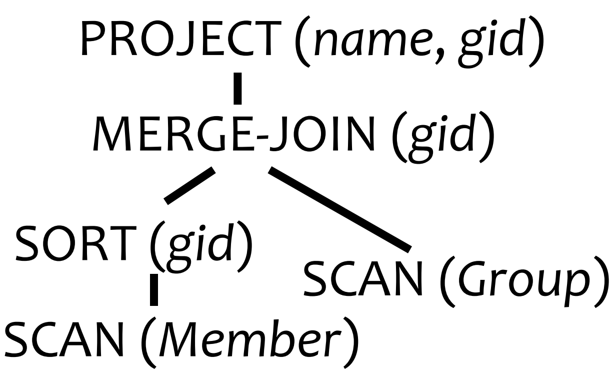


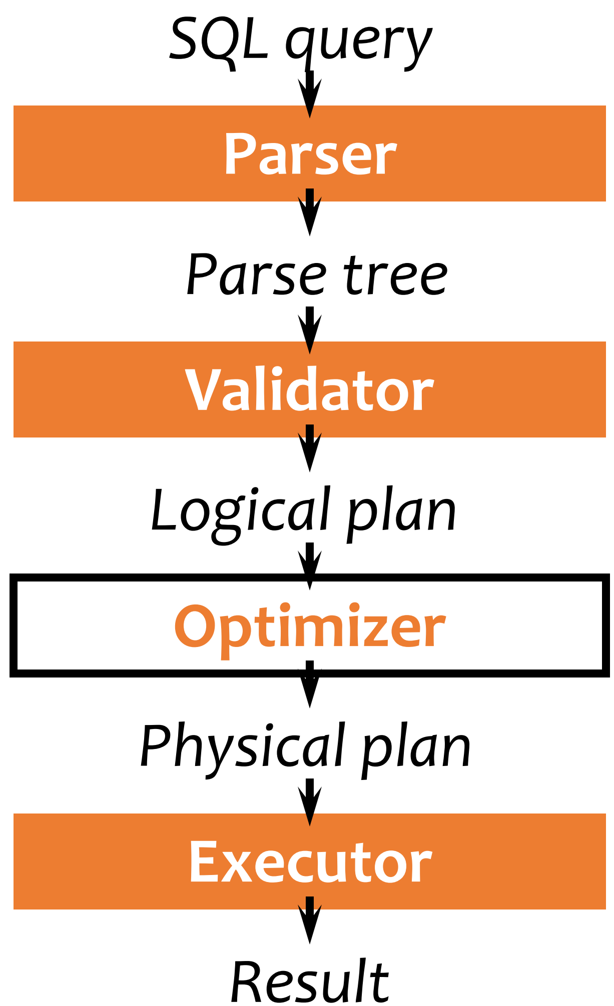
Summary of techniques

- Scan
  - Table scan
  - Selection, Duplicate-preserving projection
  - Nested-loop join
- Sort
  - External merge sort
  - Duplicate elimination, Grouping and Aggregation
  - Sort-merge join, Union (set), Difference, Intersection
- Hash
  - Hash join, union (set), difference, intersection, duplicate elimination, grouping and aggregation
- Index
  - Selection, index nested-loop join, zig-zag join

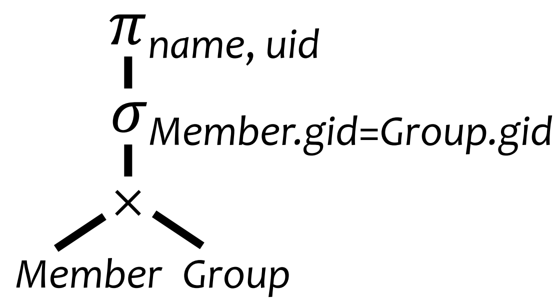
Back to the trip





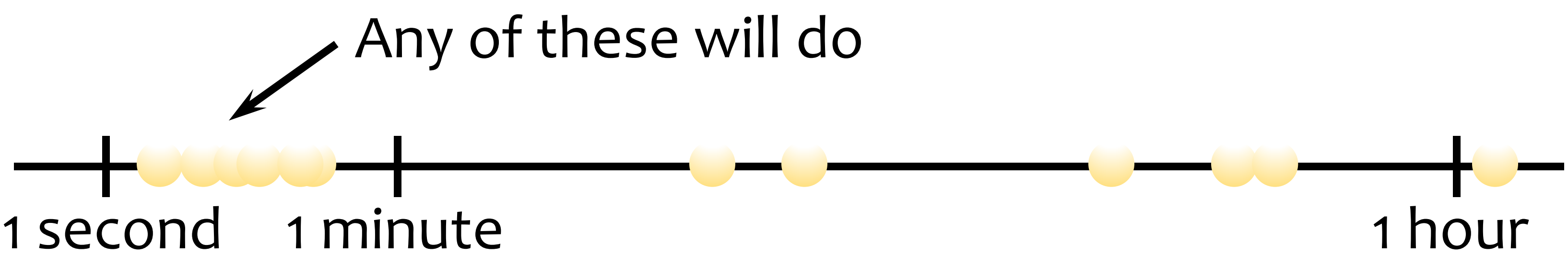



```
SELECT name, uid
FROM Member, Group
WHERE Member.gid =
    Group.gid;
```



Query optimization

- Why query optimization?
  - Many different ways of processing the same query
    - A query can have multiple logical plans
    - A logical plan can have numerous physical plans
      - Scan? Sort? Hash? Index?
  - Different ways make different assumptions about data have different performance
- Often, the goal is not getting the optimum plan, but instead avoiding the horrible ones



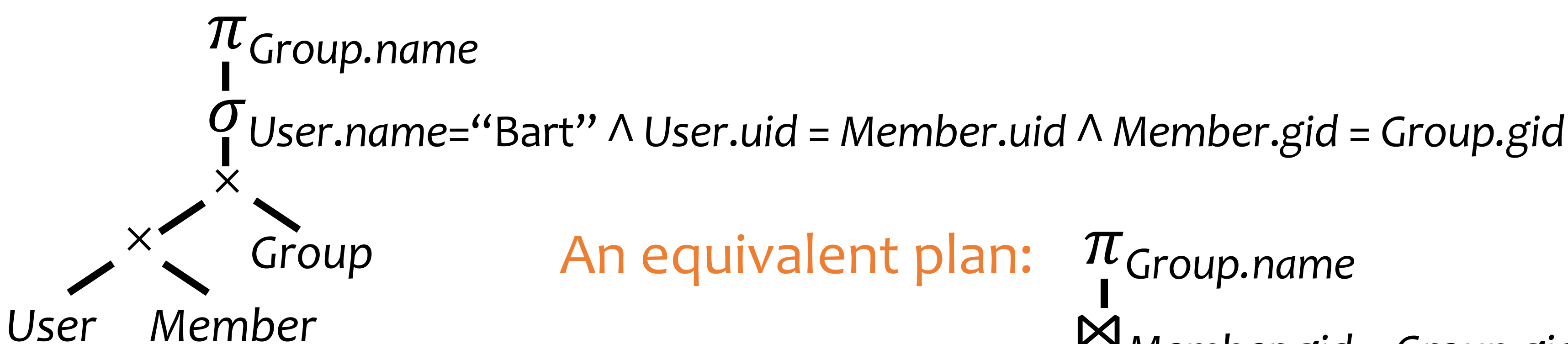
Outline

- Search space
  - What are the possible equivalent logical plans? (this lecture)
  - For each logical plan, what are the possible physical plans? (Lecture 16 - 17)

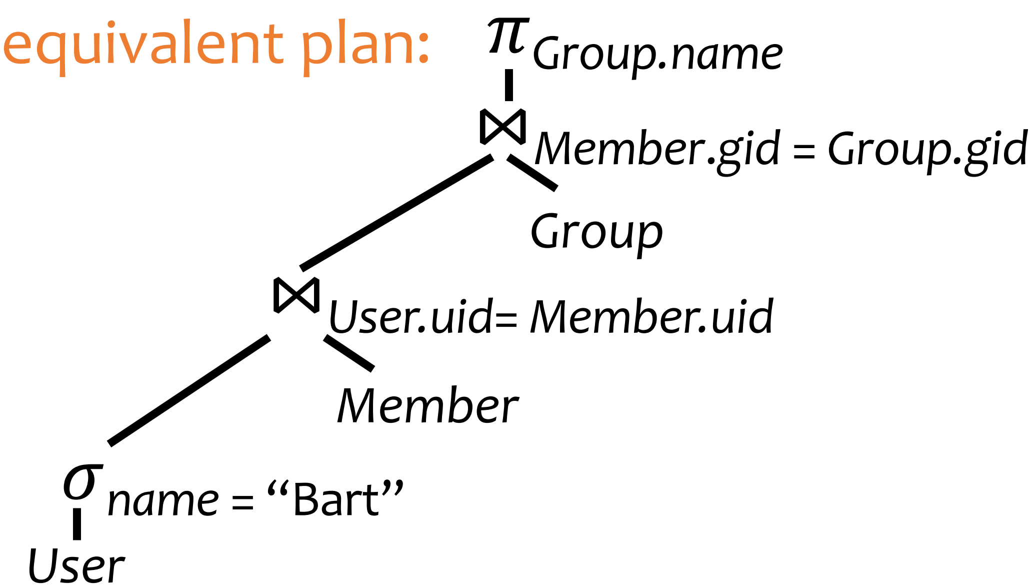
- Search strategy
  - Rule-based strategy
  - Cost-estimation-based strategy

Logical plan

- An expression tree where nodes are logical operators (often relational algebra operators)
- There are many equivalent logical plans



An equivalent plan:

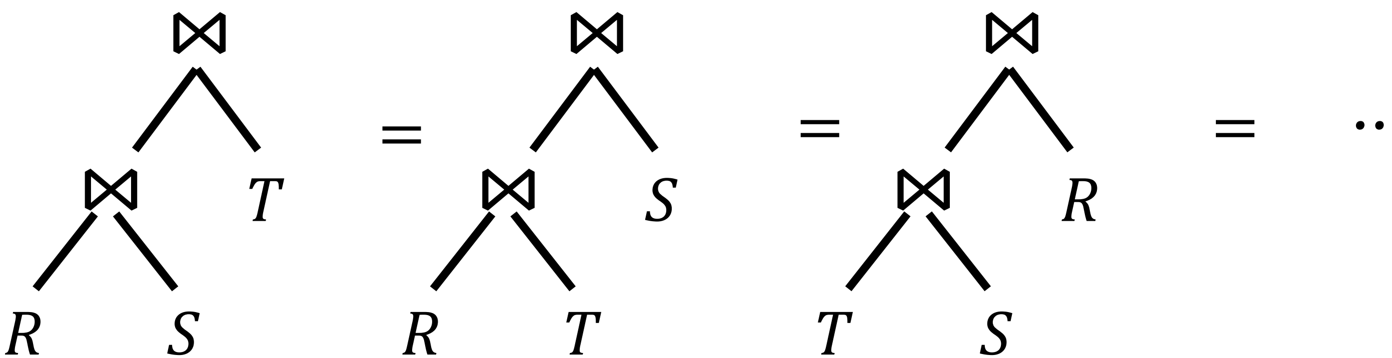


Algebraic equivalences

- Apply algebraic equivalences in relational and/or algebra to systematically transform a plan to new ones
- Convert σ_p-$\times$ to/from $\bowtie$: $\sigma_p(R \times S) = R \bowtie_p S$
- $\times$ and $\bowtie$ are associative and commutative (except column ordering, which is unimportant)
  - $R \times S = S \times R$
  - $(R \times S) \times T = R \times (S \times T)$
  - $R \bowtie S = S \bowtie R$
  - $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Algebraic equivalences

- Join reordering:



- Efficiently finding a good ordering for a join query has been a long-standing database research problem until today
  - The number of intermediate join results matters!
  - Example: User(uid, …) ⋈ Member (uid, gid) ⋈ Group(gid, …)

Algebraic equivalences

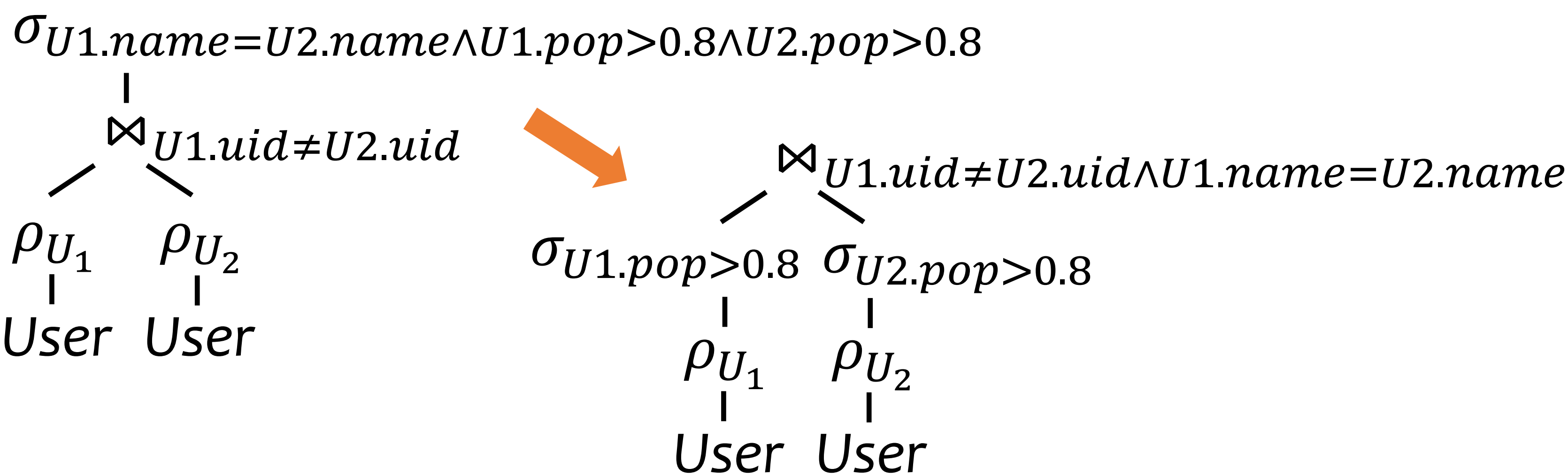
- Merge/split σ's: $\sigma_{p_1}\left(\sigma_{p_2}R\right) = \sigma_{p_1 \wedge p_2}R$
- Merge/split π's: $\pi_{L_1}\left(\pi_{L_2}R\right) = \pi_{L_1 \cap L_2}R$
- Merge/split $-$'s :
  - $(R - T) \cup (S - T) = (R \cup S) - T$
  - $(R - T) \cap (S - T) = (R \cap S) - T$
  - $R - S - T = R - (S \cup T)$

Algebraic equivalences

- Push down σ into $\bowtie$:

$$\sigma_{p \wedge p_R \wedge p_S}\left(R \bowtie_{p'} S\right) = \left(\sigma_{p_R} R\right) \bowtie_{p \wedge p'} \left(\sigma_{p_S} S\right)$$

  - p_R involves only R;
  - p_S involves only S;
  - p and p' involve both R and S



Algebraic equivalences

- Push down π into σ:

$$\pi_L(\sigma_p R) = \pi_L\left(\sigma_p(\pi_{L \cup L'} R)\right)$$

  - L' is the set of columns referenced by p

$\pi_{name,age}$

|

$\sigma_{pop>0.8 \land age<18}$

|

User

$\pi_{name,age}$

|

$\sigma_{pop>0.8 \land age<18}$

|

$\pi_{name,age,pop}$

|

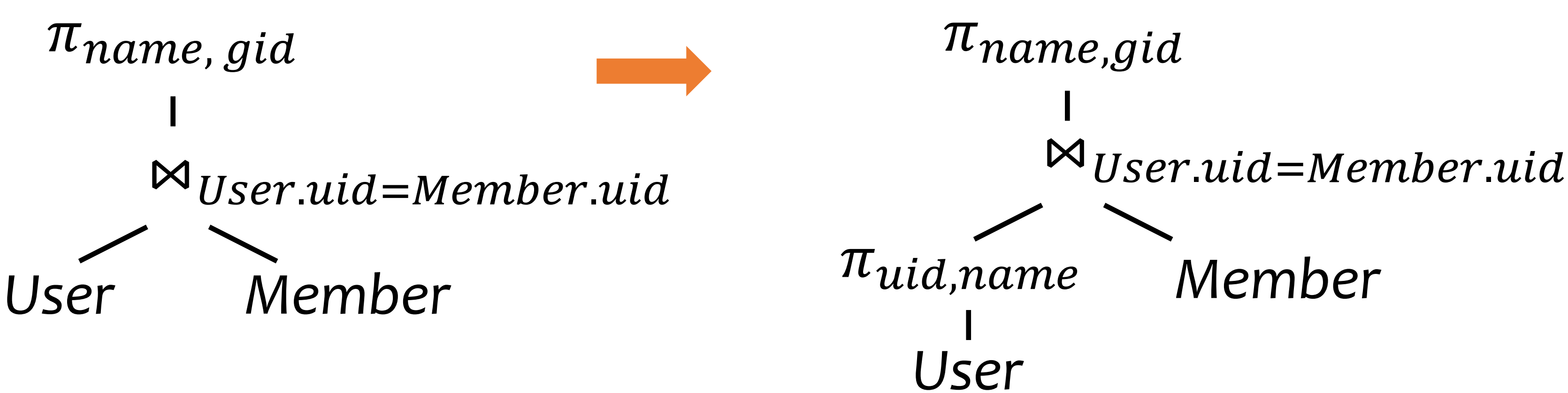
User

Algebraic equivalences

- Push down π into $\bowtie$:

$$\pi_L(R \bowtie_p S) = \pi_L\left(\left(\pi_{L_R} R\right) \bowtie_p \left(\pi_{L_S} S\right)\right)$$

  - L_R is the set of columns referenced by p and L for R
  - L_S is the set of columns referenced by p and L for S



Algebraic equivalences

- Push down $-$ into $\bowtie$:
  - Suppose R and T have the same schema:
    $$(R \bowtie S) - (T \bowtie S) = (R - T) \bowtie S$$
  - Suppose S and W also have the same schema
    $$(R \bowtie S) - (T \bowtie W) =?$$

R(A,B)⋈S(B,C,D) −T(A,B)⋈W(B,C,D) = $\big((R - T) \bowtie S\big) \cup \big(R \bowtie (S - W)\big)$

Direction ⊇:

- For any (a,b,c,d) ∈ $\big((R - T) \bowtie S\big)$, (a,b) ∈R, (b,c,d) ∈ S, so (a,b,c,d) ∈ R ⋈ S; (a,b) ∉ T, so (a,b,c,d) ∉ T ⋈ W
- Similarly, if (a,b,c,d) ∈ $\big(R \bowtie (S - W)\big)$, (a,b,c,d) ∈ R ⋈ S and (a,b,c,d) ∉ T ⋈ W

Direction ⊆:

For any (a,b,c,d) ∈ R⋈S −T⋈W, either (a,b) ∉ T or (b,c,d) ∉ W holds

- If (a,b) ∉ T, (a,b,c,d) ∈ $\big((R - T) \bowtie S\big)$
- If (b,c,d) ∉ W, (a,b,c,d) ∈ $\big(R \bowtie (S - W)\big)$

What is next?

- Search space
  - What are the possible equivalent logical plans? (this lecture)
    - Many rewrite rules to be further explored …
  - For each logical plan, what are the possible physical plans? (lecture 16 - 17)

- Search strategy
  - Rule-based strategy
  - Cost-estimation-based strategy