Database Administration

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Roadmap

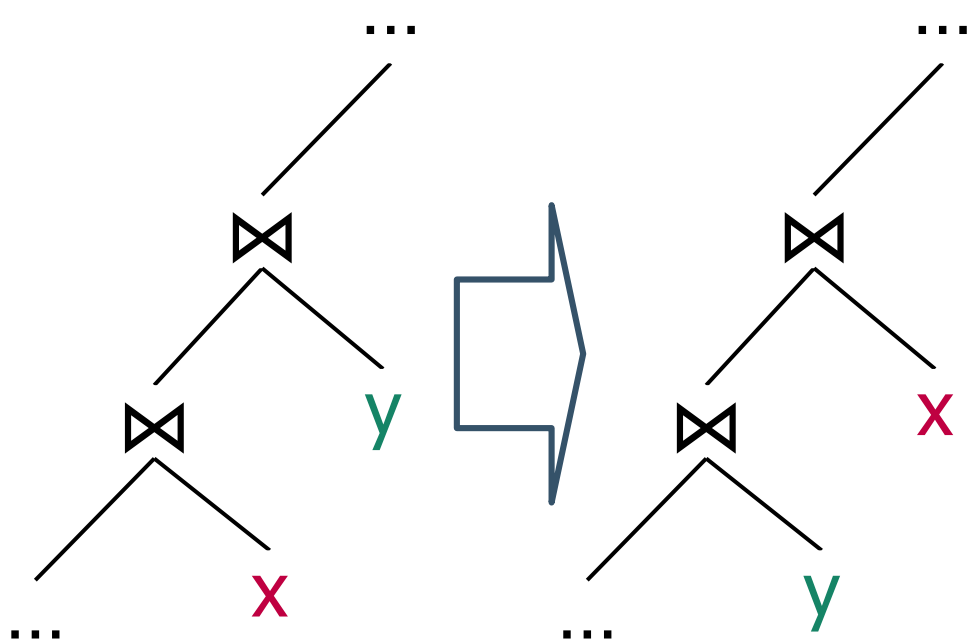
- How to estimate the cost of a plan?

- How to find alternative plans?

Search space

- The set of possible alternative plans (search space) is determined by a set of rules
  - Equivalent relational algebra expressions
  - Physical implementation of single operators or plan fragments
  - Enforcing physical properties
- The set of rules is the main configuration point for extensible query optimizers

A simple rule for Join order

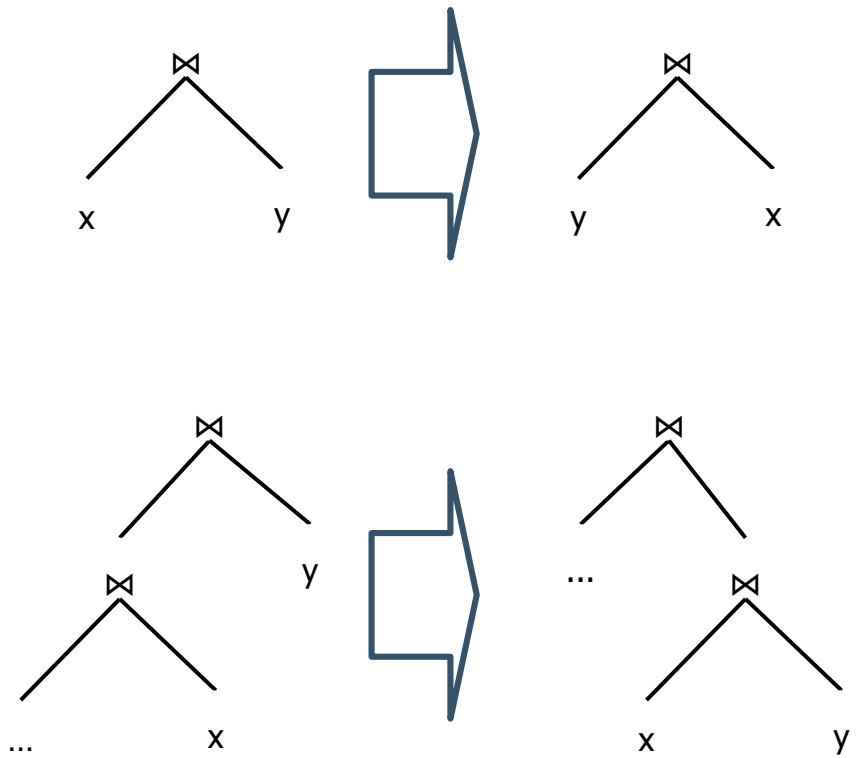


- Inner join is commutative and associative

  (...⋈x)⋈y =
  ...⋈(x⋈y) =
  ...⋈(y⋈x) =
  (...⋈y)⋈x

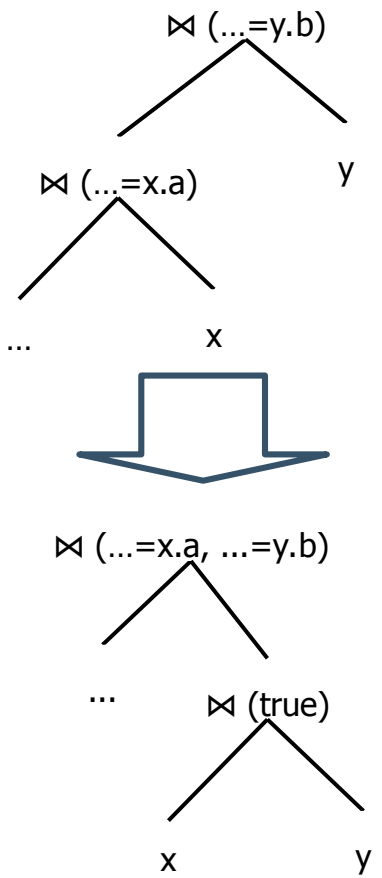
- This allows exploring all left-deep trees
  - n! permutations
- Does not consider bushy trees

Join commutativity and associativity



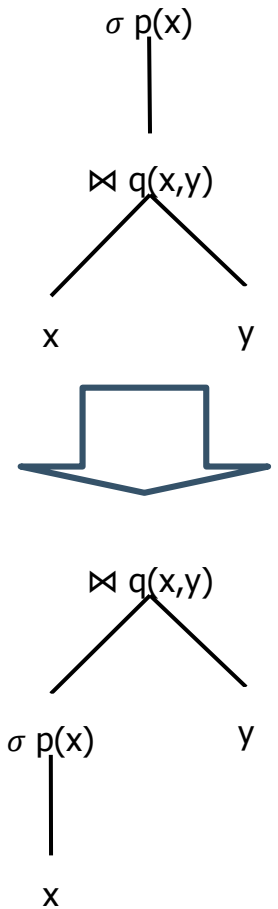
- Separately considering commutativity and associativity rules
- Allows exploring all possible join expressions:
  - Including bushy trees

Cross-products



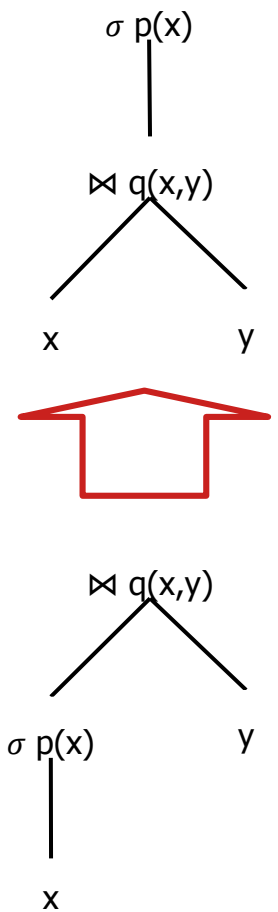
- Associativity rules may result in cross-products
- These are not useful as physical plans, as they result in a lot of intermediary data
- May be useful as an interim transformation for an useful plan

Selection push-down



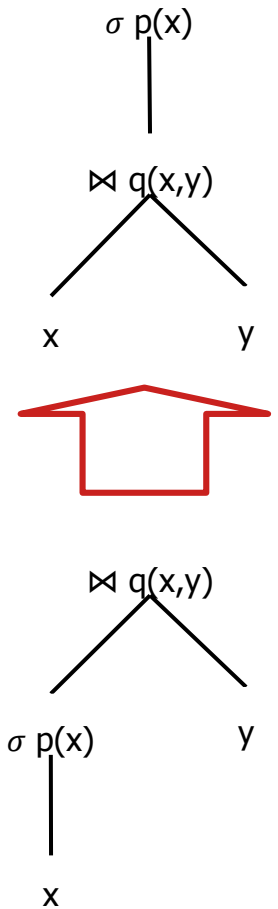
- Possible when the predicate involves only one of the branches
  - Otherwise, merge it in the join condition!
- If the predicate is highly selective, reduces the amount of work in join
- Selectivity is the key criteria for join ordering!!!

Selection pull-up



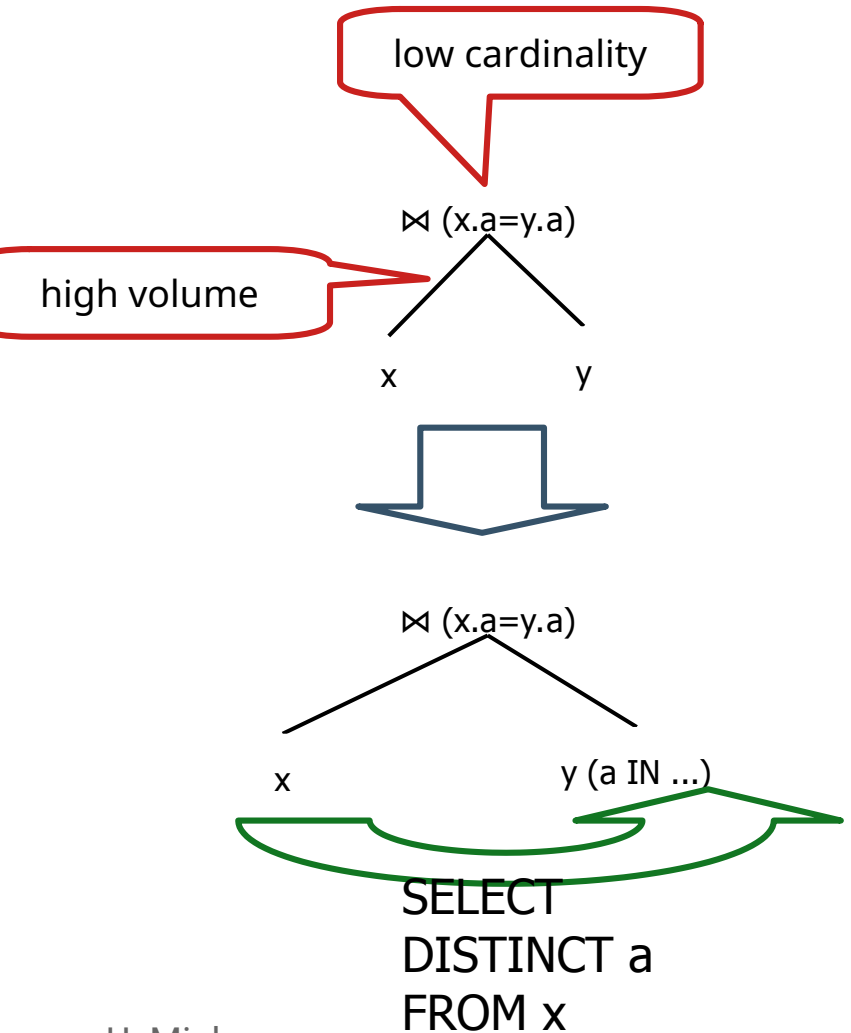
- Always possible, but…
- Does it ever make sense?!?!

Selection pull-up



- Assume that q is highly selective
- Assume that p is very costly to execute:
  - e.g. call into an LLM to check if “sentiment” on a textual column is “positive”
- Can be useful!

Sideways information passing



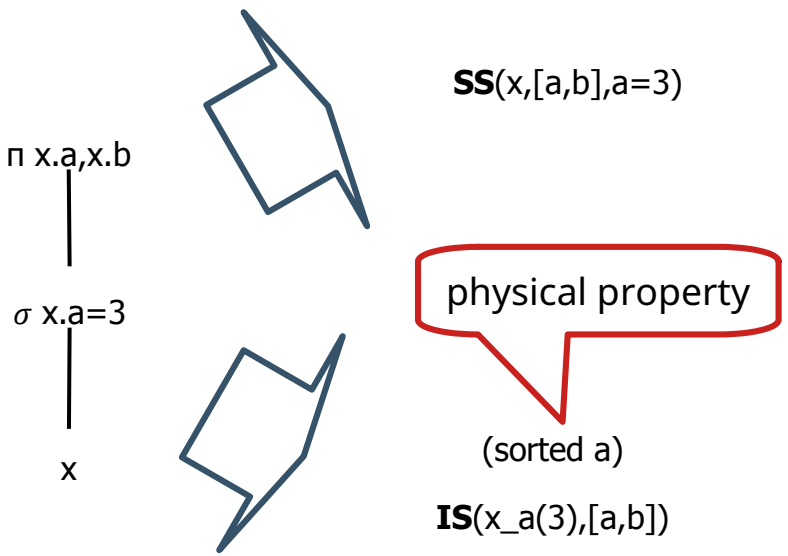
- When implemented as a hash-join:
  - Builds a very large in-memory table from y
  - Uses only a few items
- Sideways information passing from left to right side of join:
  - List of relevant keys for filtering
- Very important in federated / disaggregated systems!

More transformation rules

- Selection and projection push-down (and pull-up) through other operators
- Push down group-by through join
- Sub-query decorrelation
- …

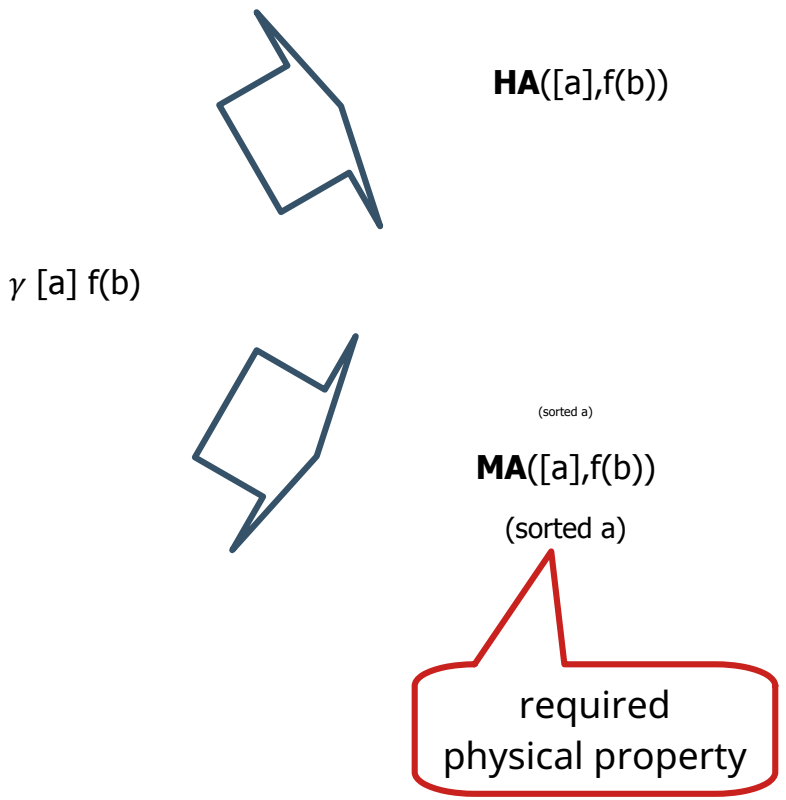
- Note that logical operators have ∞ cost because they cannot be actually executed!

Access path implementation



- Physical access operators (scans) can filter and project, reducing the amount of data movement
  - Using indexes
  - Using columnar storage
- Indexes can also be used to ordered traversal of data (without filtering)

Grouping and aggregation implementation



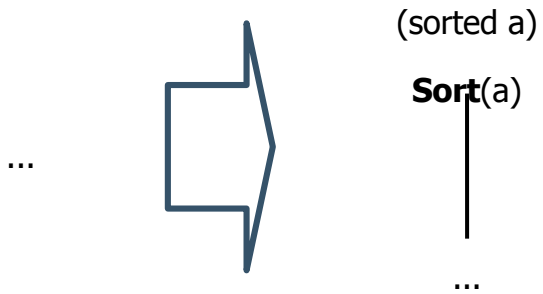
- Two-pass algorithms for grouping (and join) depend on sorted input
  - Expressed as a required "physical property"
- Sorting is preserved

Other implementation rules

- Other access path operators:
  - Index-only scan
  - Bitmap indexes
  - ...
- Join implementations

- The cost of physical operators is not ∞ and can be estimated, leading to executable plans!

Enforcers

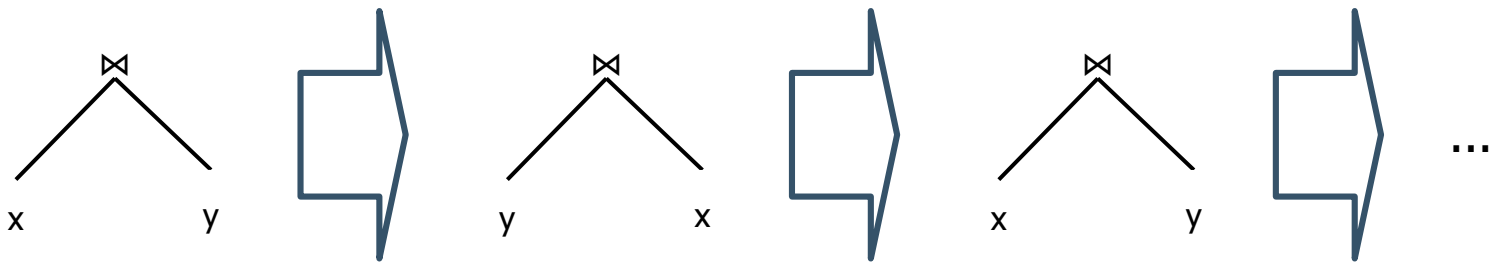


- Do not change data
- Enforce a specific physical property on data
- Have finite cost
- Examples:
  - Order → Sort operator
  - Partition → Exchange
- See search algorithm for more details

Search space

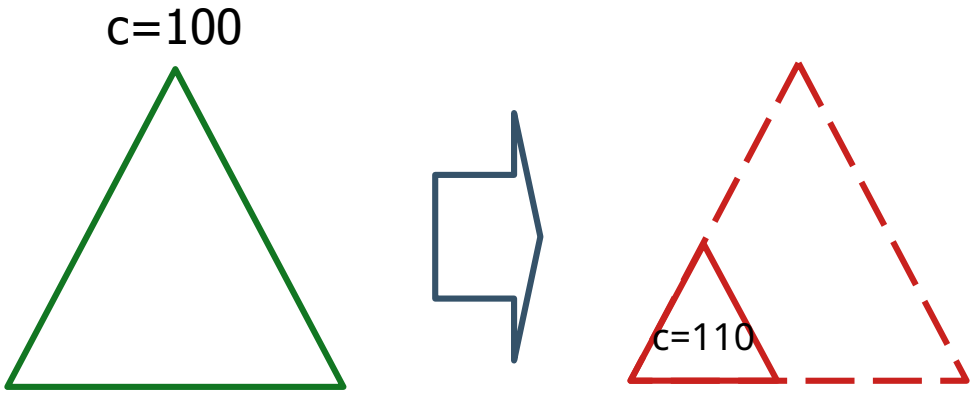
- Rule sets that produce a larger search space:
  - More likely to contain the optimum → faster execution
  - More work to evaluate all alternatives → faster planning
- Query execution is the sum of planning and execution
- Best overall performance:
  - Depends on workload
  - Found as a compromise of planning and execution

Search algorithm



- Repeated application of rules can result in an infinite loop
- Solution:
  - Remember all plans to check for repetitions

Search algorithm



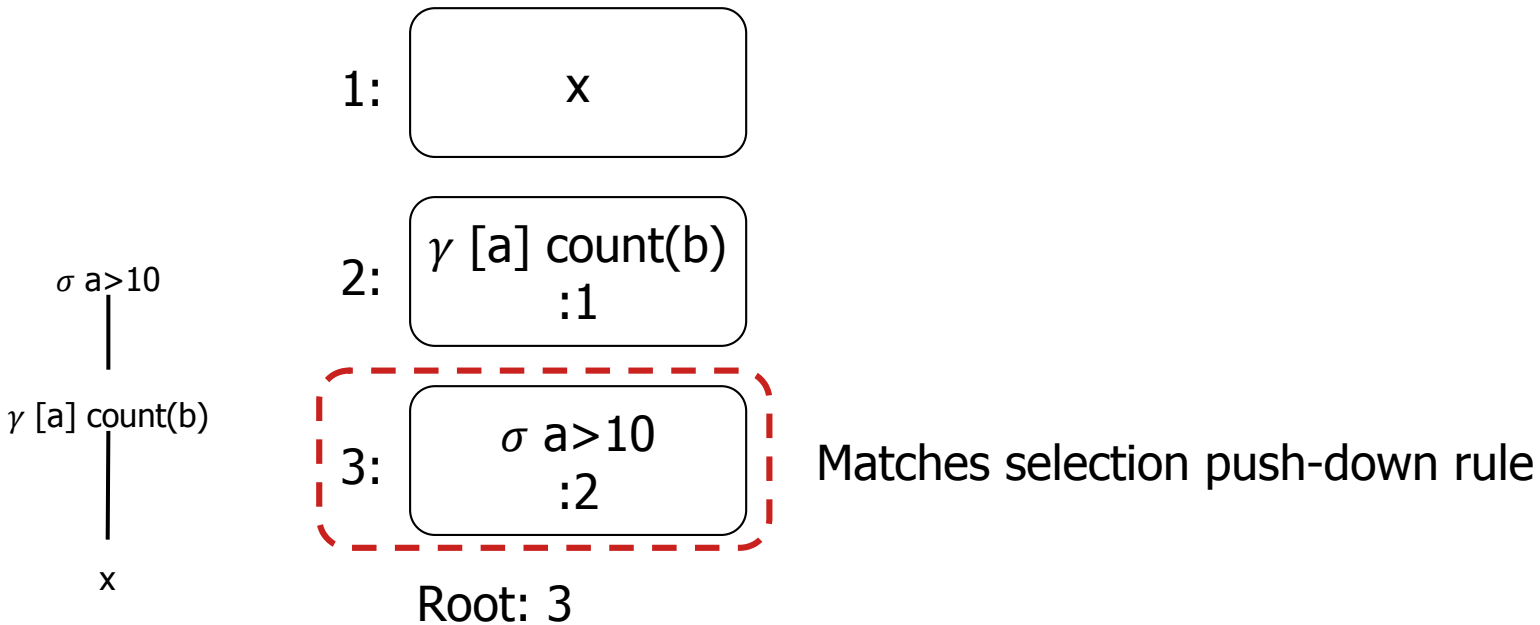
- The cost of a sub-plan of an alternative being explored may be greater that the total cost of the best know alternative
- Solution:
  - Prune search based on current best estimate

Search algorithm

- Some sequences of rule applications tend to converge faster to the optimum plan
  - The sooner that we get a “good” estimate, the more alternatives that can be pruned
- Solution:
  - Order available rules by their heuristic promise

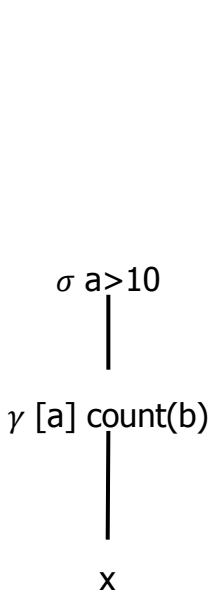
Example

SELECT a, COUNT(*) FROM x WHERE a > 10;



Example

SELECT a, COUNT(*) FROM x WHERE a > 10;



1: x

2: γ [a] count(b) :1

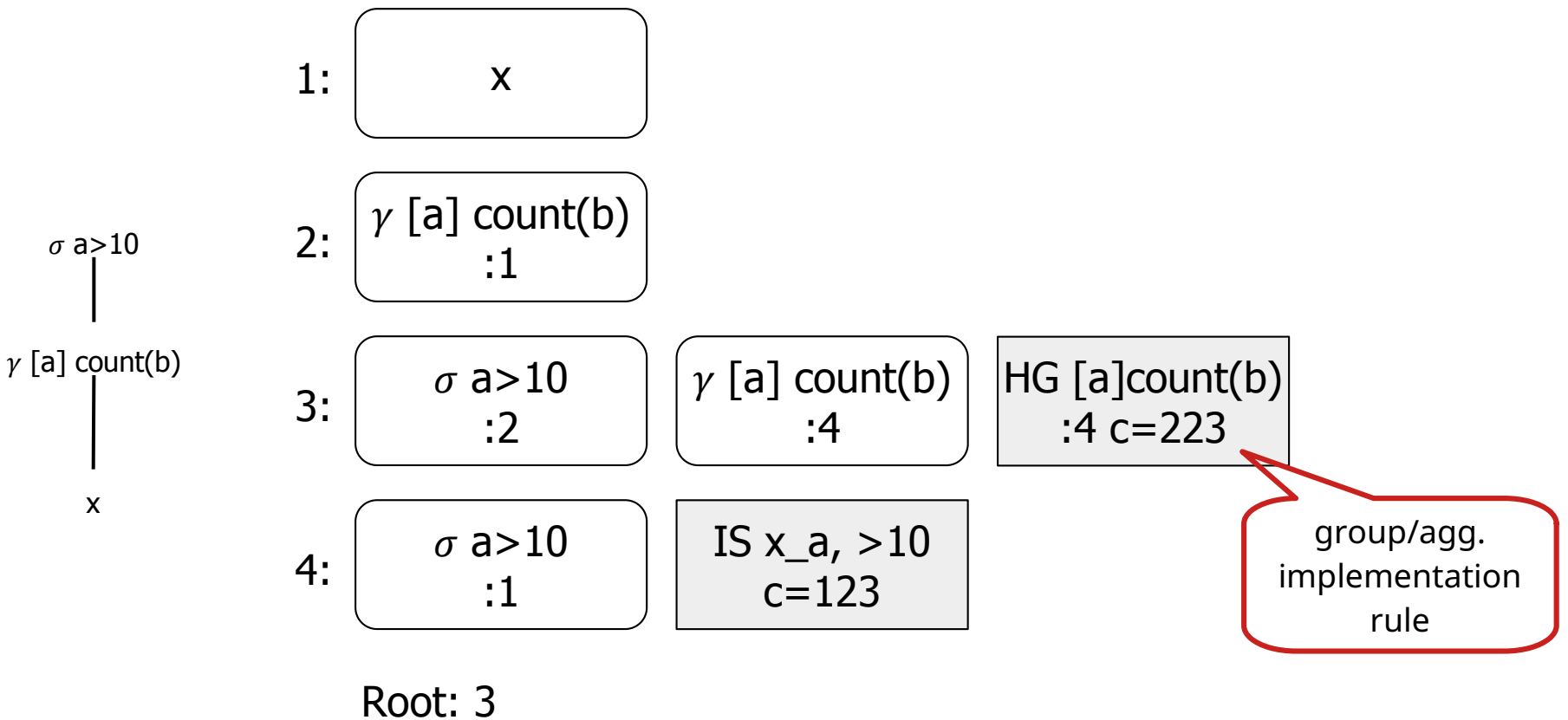
3: σ a>10 :2 — γ [a] count(b) :4

4: σ a>10 :1 — Matches index scan implementation rule

Root: 3

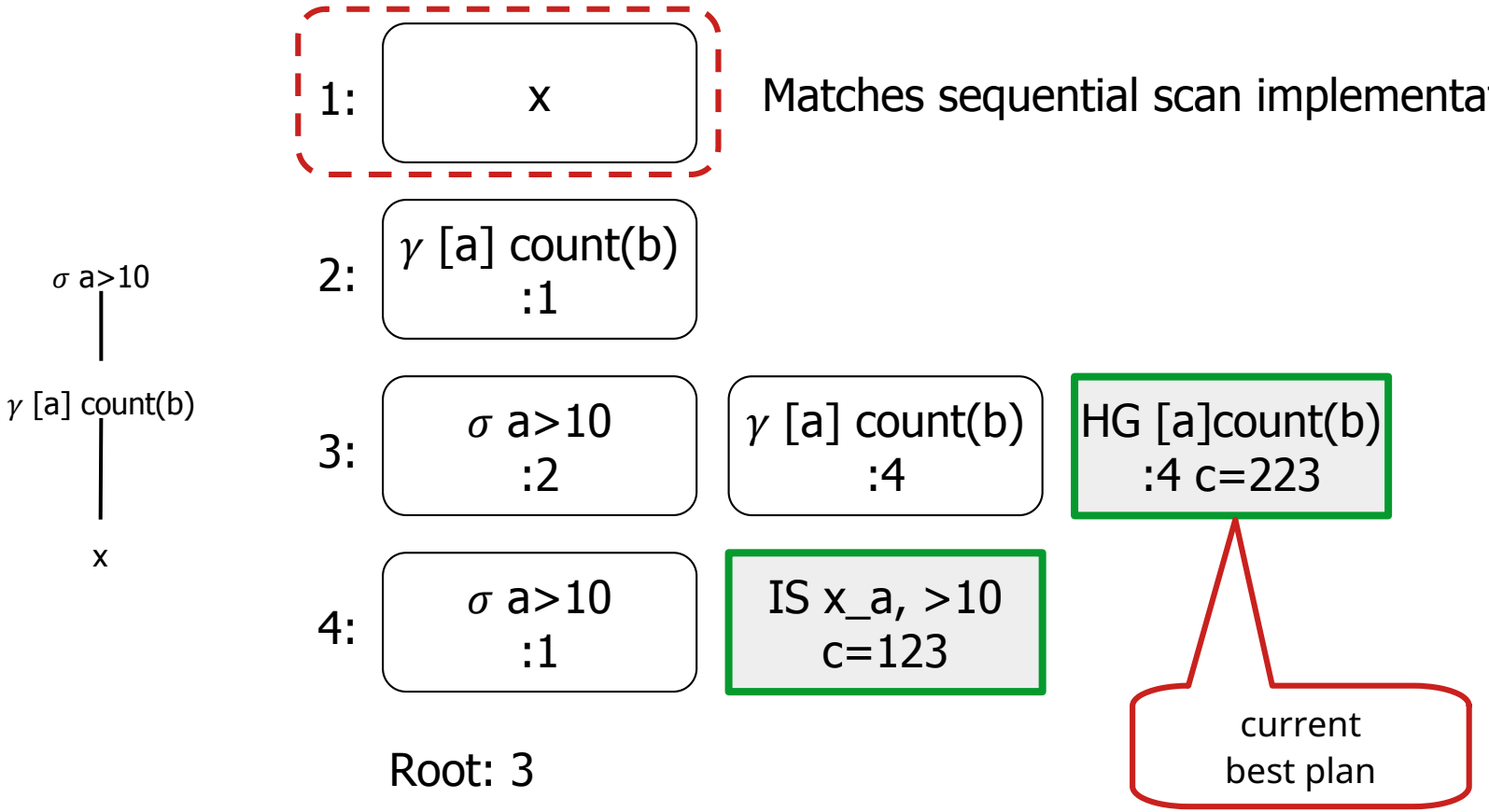
Example

SELECT a, COUNT(*) FROM x WHERE a > 10;



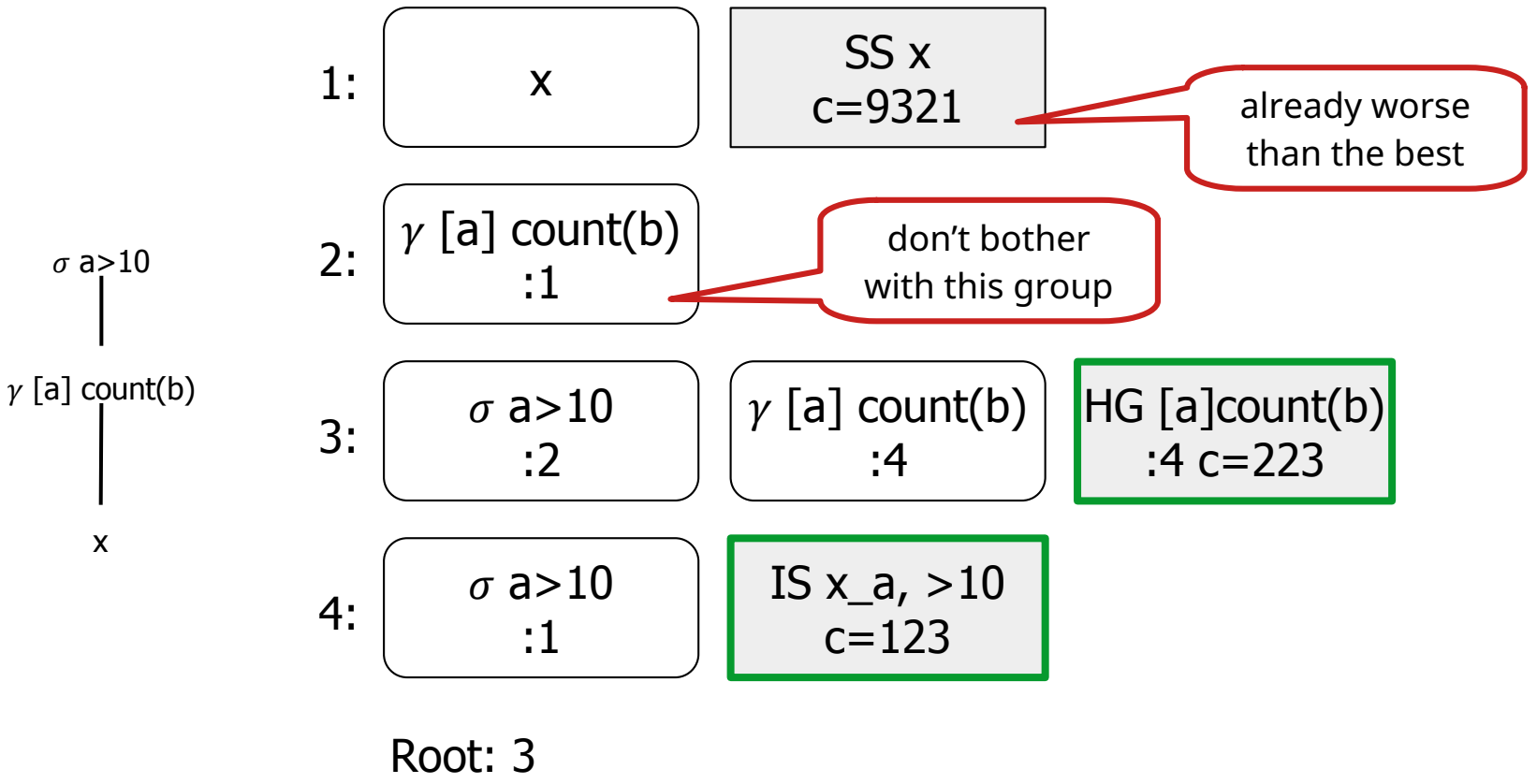
Example

SELECT a, COUNT(*) FROM x WHERE a > 10;



Example

SELECT a, COUNT(*) FROM x WHERE a > 10;



Volcano/Cascades search

- Top down search, from root
- When entering a group, for each expression:
  - Optimize inputs to expression
  - Apply each rule expression:
    - Order rule application by promise
- Reconsider each group (and ancestors) when new expressions are found

- Volcano and Cascades differ how rule matching tasks are scheduled

Conclusions

- Make sure that the best option is available:
  - Indexes and materialized views
  - Sufficient memory for all operators
- Make sure the best option is selected:
  - Tune relative weights
  - Provide current and sufficiently detailed statistics

PostgreSQL

- Simpler bottom-up optimization algorithm
  - Limits use of materialized views
- Not designed for extensibility
- No hinting:
  - Considered a bug!
- Optional genetic solver for join ordering

Apache Calcite

- Implements Volcano/Cascades in Java
- Large general purpose rule collection
  - Including materialized view substitution
- Aimed at federated database systems:
  - Multiple sets of physical implementation operators, that map to requests to different systems
  - Native physical implementations (Enumerable)