Parallel Query Optimisation

Contents

Objectives of Parallel Query Optimisation

For a relational database query

$\Downarrow$

Several relational operators are executed

$\Downarrow$

Several execution orderings of the operators are possible

$\Downarrow$

Select the best ordering!

The meaning of „best ordering”

The trivial goal: the shortest response time for the query

In multi-user databases: achieve maximal throughput with acceptable response time for every query

Another goal: the least resource consumption

A cost metric should be defined for the parallel system considering the resources shared either in time or in space for different operations.

The optimiser counts the cost of the execution plans based on the given metric and searches for the minimum.

Parallel Query Optimisation

From sequential to parallel optimisation

Two-Phase optimisation methodology

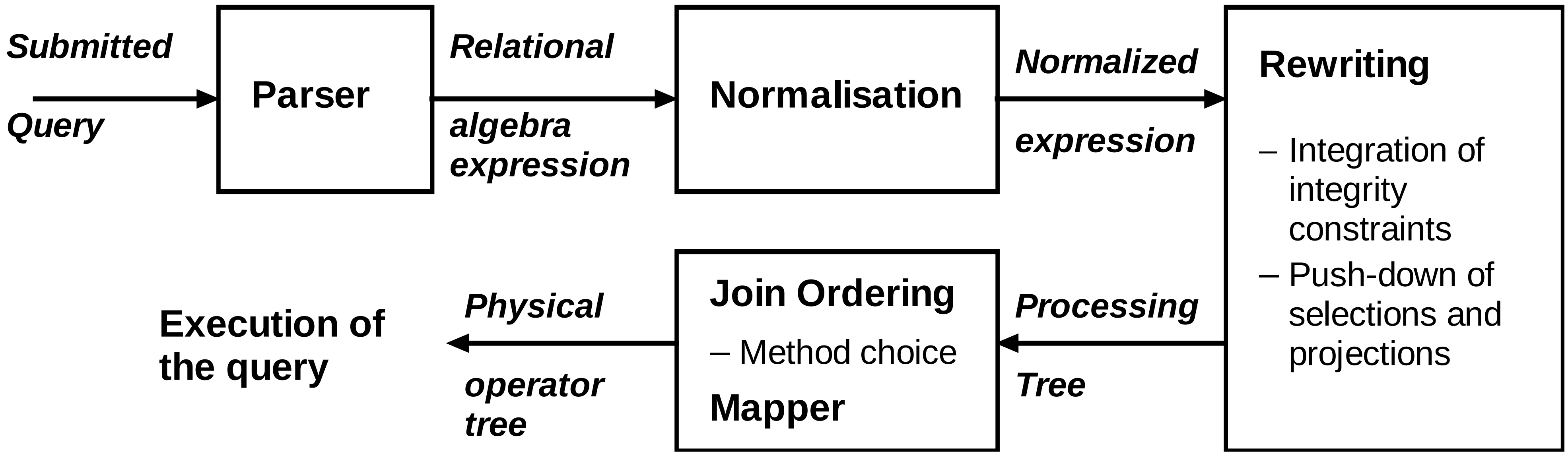
One-Phase optimisation methodology

Inter-operator parallelism oriented optimisation

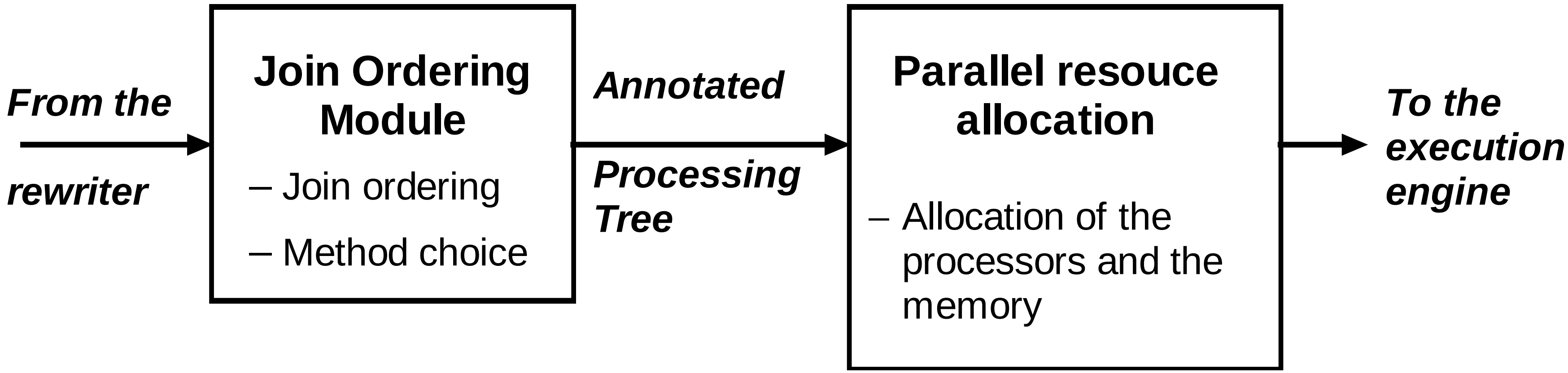
Summary

From single-query to multi-query optimisation

Schema of sequential relational query optimisation



Two-Phase parallel query optimisation



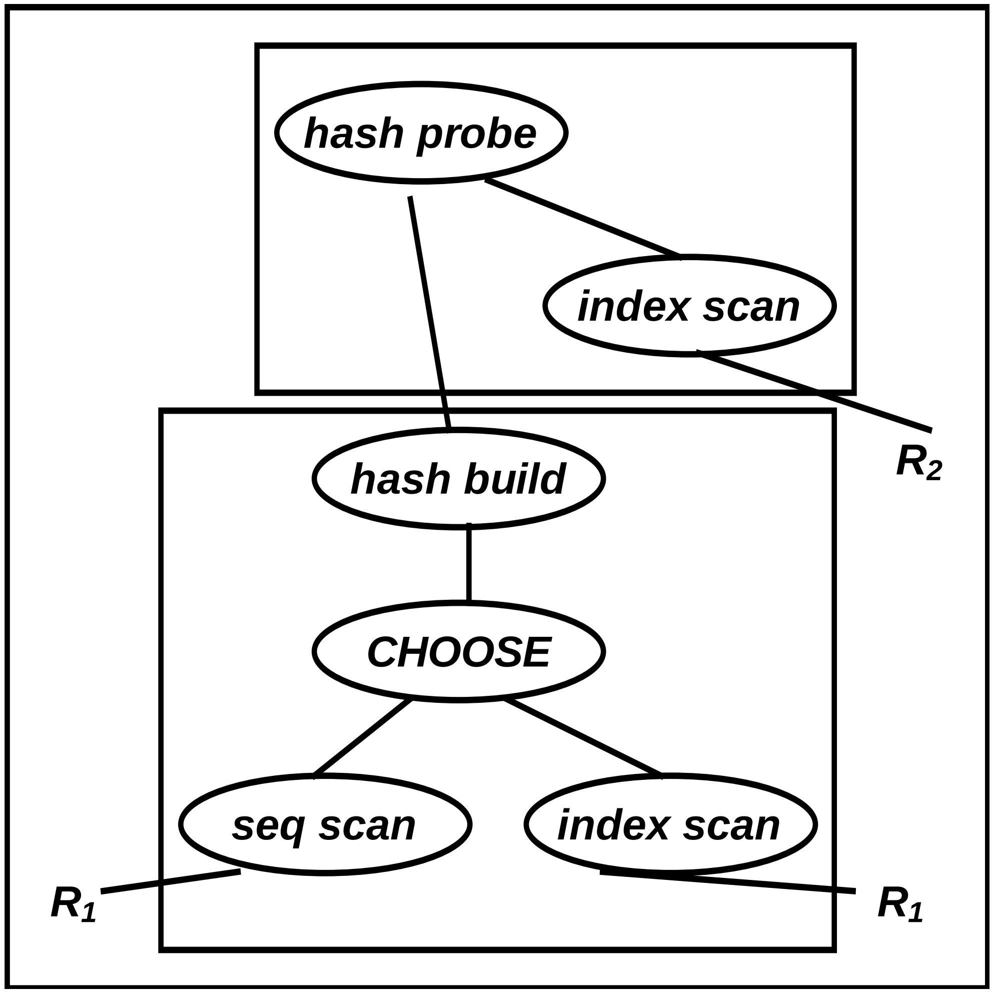
- **Simplifies the optimisation process**
- **First phase can be a uni-processor optimiser ⇒ industrial products choose such an approach**
- **Difficult to achieve optimal method choice without considering parallel resources**

Two-Phase optimisation in XPRS

First phase

A collection of good sequential processing trees for various memory sizes and processor numbers is retained

If the optimiser cannot decide between two alternative strategies, a **choose node** *is used*



Two-Phase optimisation in XPRS

- **Second phase**
  - ***The tree is split into tasks which could be executed in parallel (inter-operator parallelism)***
  - ***At any time only two tasks must be run in parallel (an I/O intensive and an another one)***
  - ***The processors and disks operate as close to their full utilisation as possible***
  - ***Dynamic parallel adjustment algorithm is used for the allocation of new tasks***
- **Good for shared-memory architecture and simple cost models**

„Two and a half Phase” method in Tandem NonStop SQL

Two Phase method

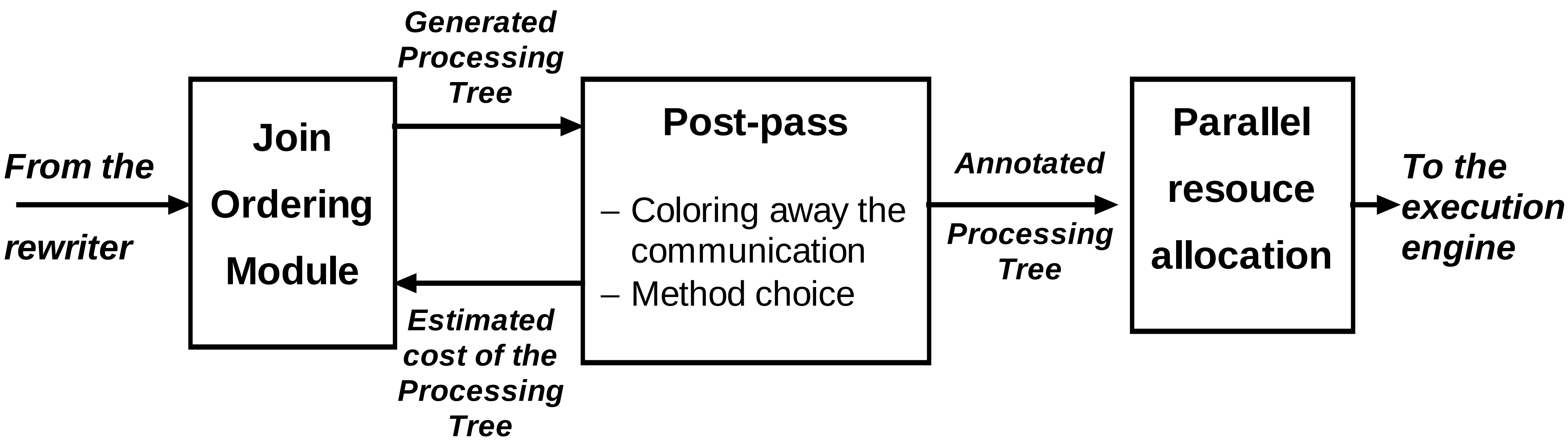
+

***post-pass* optimisation**
to the join ordering (first phase)
to optimise *redistribution cost*

Developed for distributed memory systems (shared-nothing or shared-disk)

First phase generates an optimal join ordering by taking into account the impact of redistribution

„Two and a half Phase” method in Tandem NonStop SQL



„Two and a half Phase” method in Tandem NonStop SQL

Special annotations in the processing tree

- **partition attribute**
- **strategy to achieve the desired partitioning (i.e. sorting and/or building an index)**
- **method choice for each operator, for which the redistribution costs are optimised**

Colors are associated to the nodes in the tree

- **color = (partition attribute, sorting attribute, indexing attribute)**
- **If the output color of an operator and the input color of the successor is the same, the costs of inter-operator communications are zero**

Goal: Find the optimal processing tree with the coloring meaning minimal redistribution cost

„Two and a half Phase” method in Tandem NonStop SQL

Experiments: single queries (one join operation and an aggregate function) were executed with optimised redistribution costs *three times faster* than without optimisation

The post-pass optimisation has high computational complexity

The processor allocation for join is specialised in Tandem, with general allocation method the communication costs can be integrated into the first phase in a more complicated way

Three-Phase method in IBM DB2 V.3.

1. **join ordering**
2. **computing the degree of inter-, intra-operator parallelism and possible pipeline parallelism**
3. **At run-time, the degree of intra-operator parallelism is adjusted to the actual memory space**

Developed for shared memory systems

Third phase ensures dynamic control of execution and full utilisation of the resources

Tested only for simple queries

Run-time optimisation for complex queries is cumbersome

Summary of the Two-Phase optimisation methodologies

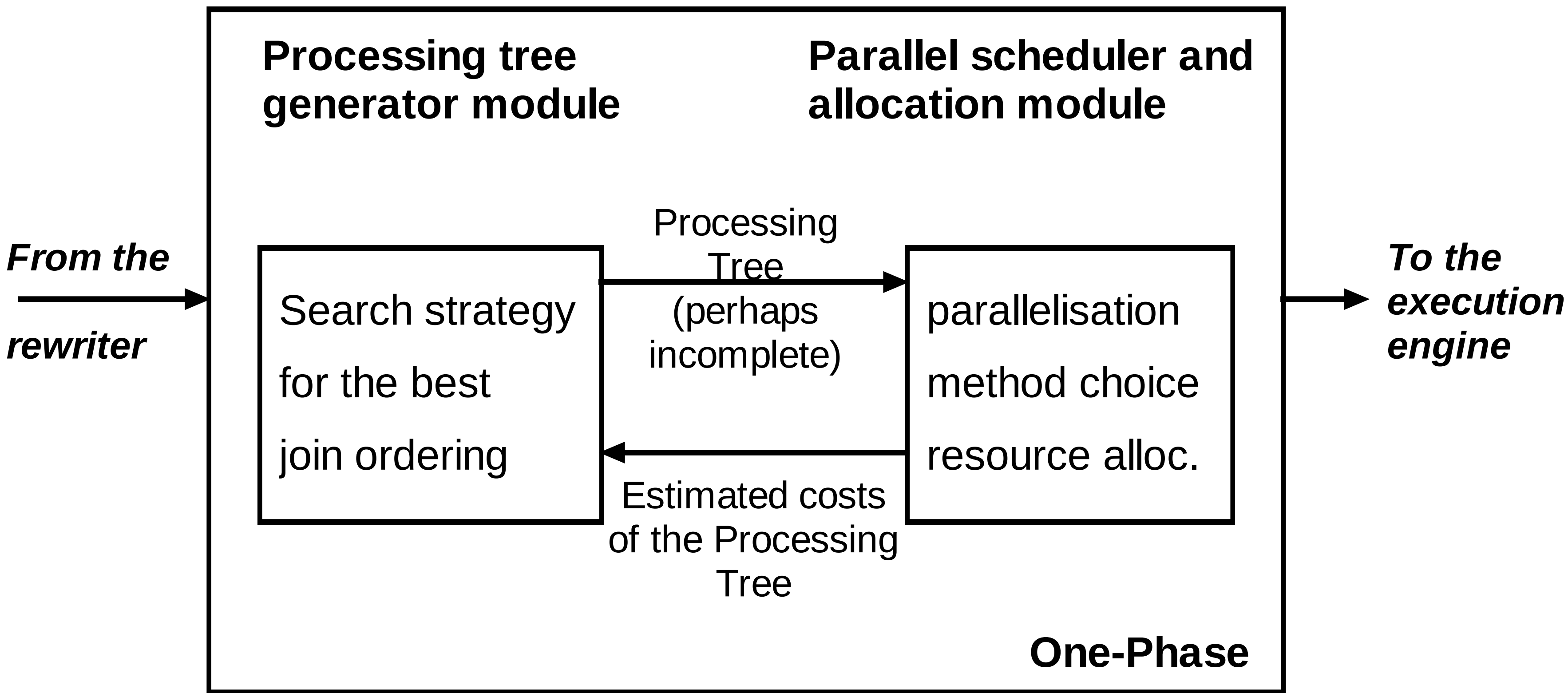
Developed for shared memory systems

Benchmarks only for small queries

For shared-nothing systems, the method may fail the optimal solution

inter-operator parallelism must be exploited for complex queries $\Rightarrow$ the join ordering must be done with regard of the parallel resources

One-Phase parallel query optimisation



One-Phase parallel query optimisation

join ordering + parallelisation strategy + parallel resource allocation in one phase

The complexity of optimum search increased significantly

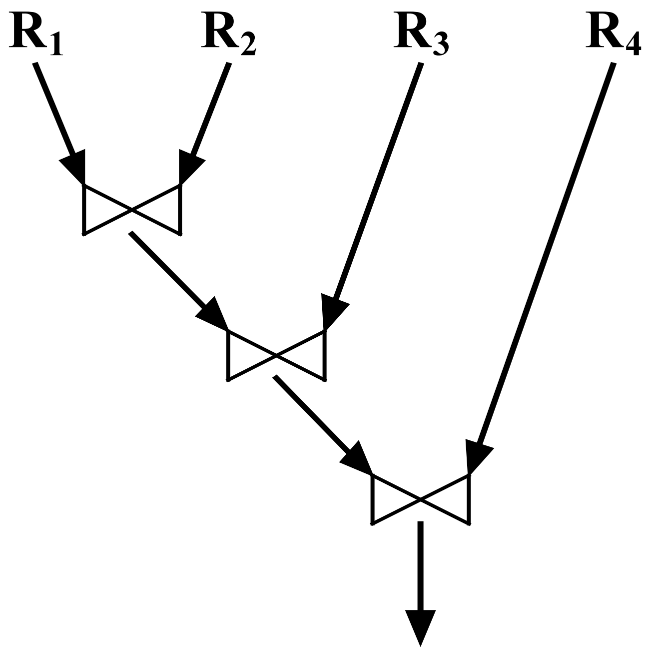
$$\Downarrow$$

Heuristical search should be done (specific processing tree shapes are considered)

left-deep tree	***right-deep tree***	***bushy tree***
segmented right-deep tree	***zigzag tree***	***serialized bushy tree***

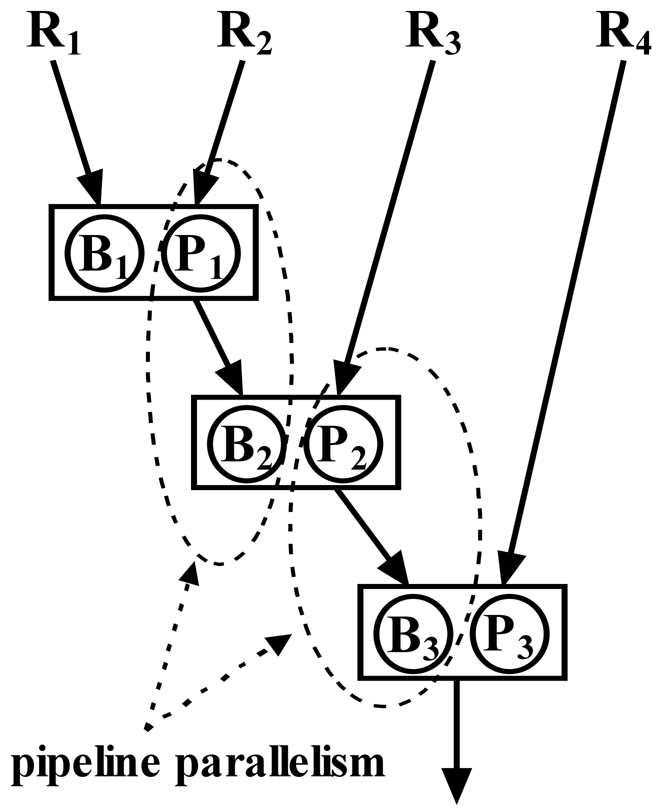
Left-deep tree

Query tree



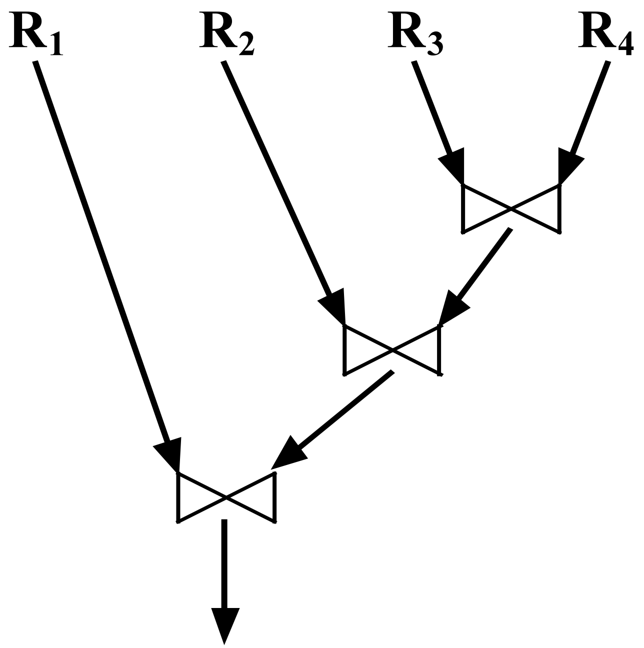
Left relation is the inner relation in a join

Physical representation

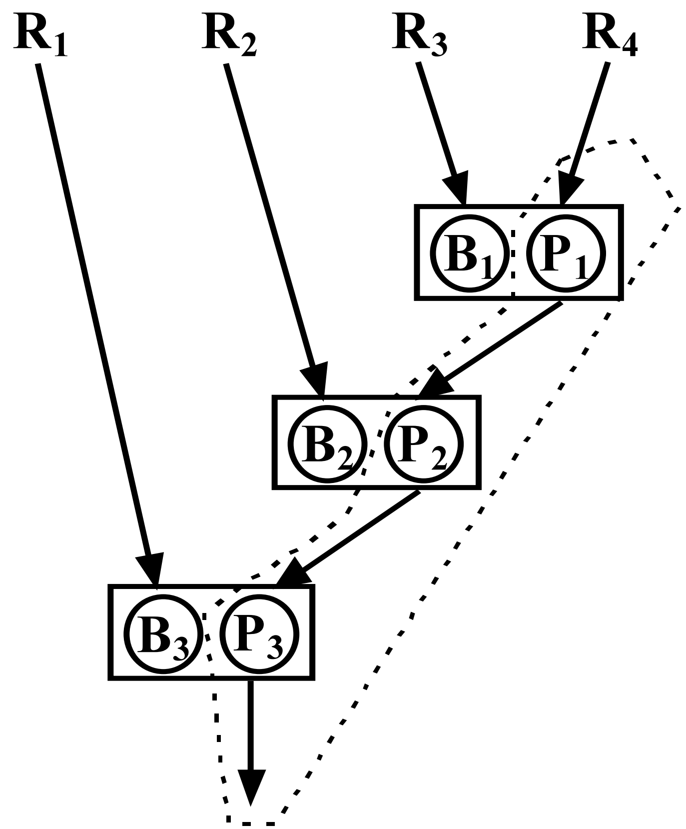


Right-deep tree

Query tree



Physical representation



Right-deep tree scheduling

Right-deep $\Rightarrow$ maximal pipeline parallelism can be exploited

If all hash tables fit into memory $\Rightarrow$ one pipeline chain
otherwise $\Rightarrow$ static right-deep tree scheduling: segmenting the chain that the hash tables of a segment fit into memory
(do not mix up with segmented right-deep trees)

Propagation of estimation errors $\Rightarrow$ dynamic scheduling: determining the next segment in run-time

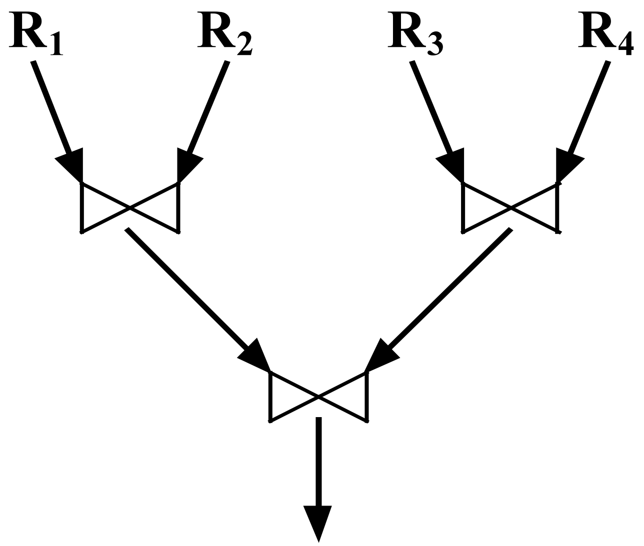
Implemented first in GAMMA, (Schneider and DeWitt)

Maximal pipeline parallelism is worth to exploit only with enough memory

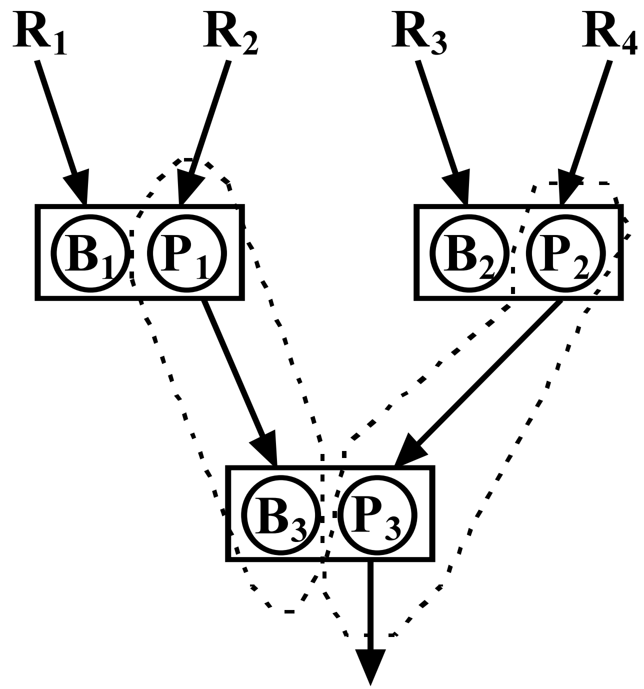
No inter-operator parallelism can be exploited

Bushy tree

Query tree



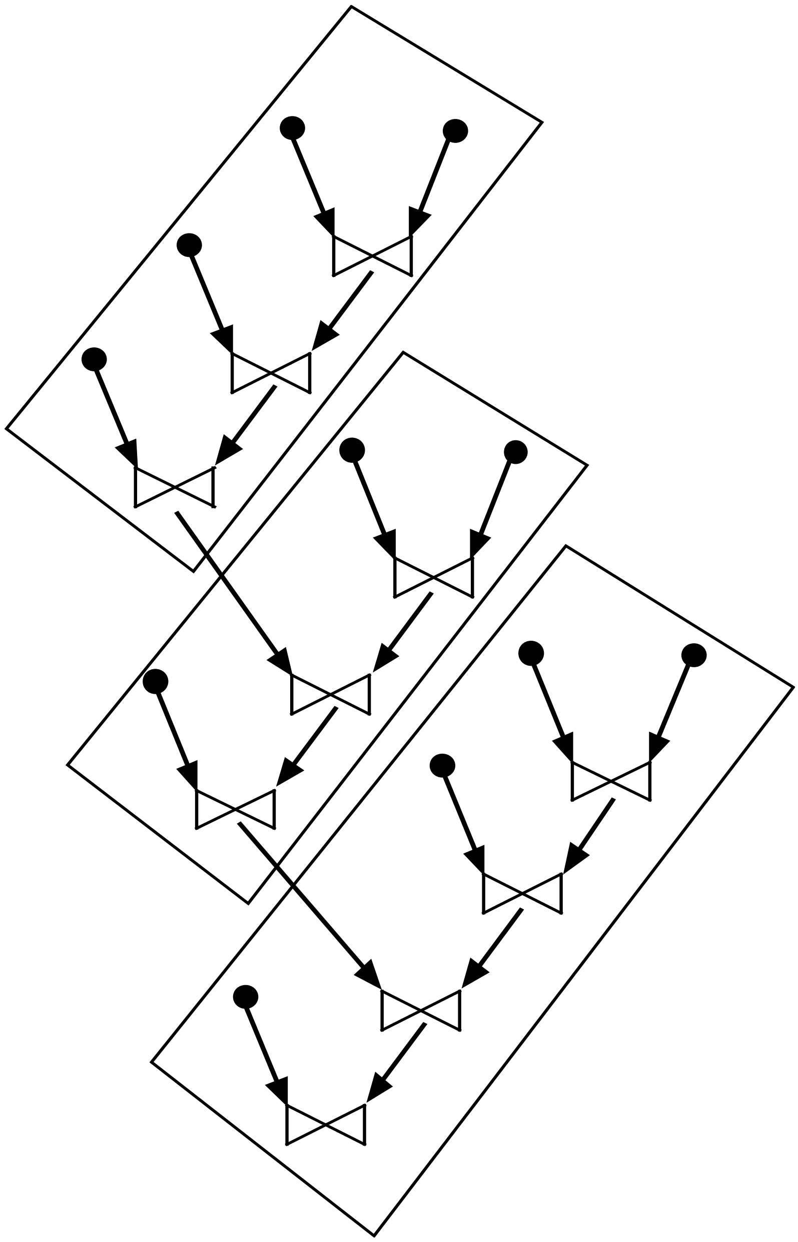
Physical representation



Segmented right-deep tree

Segmented right-deep tree = set of right-deep trees

The result relation of a segment may be *any* of the inner or outer relations of the next segment



Segmented right-deep tree scheduling

Proposed by Chen et al.

Advantage over right-deep tree

More flexible optimisation

E.g. a small result relation of a segment can be chosen to be the inner relation of the next segment $\Rightarrow$ the hash table may be built in the memory while with right-deep tree scheduling bucket processing would be needed

Advantage over bushy tree

Much simpler search space in optimisation

Problems

Redistribution costs are not integrated into the optimisation

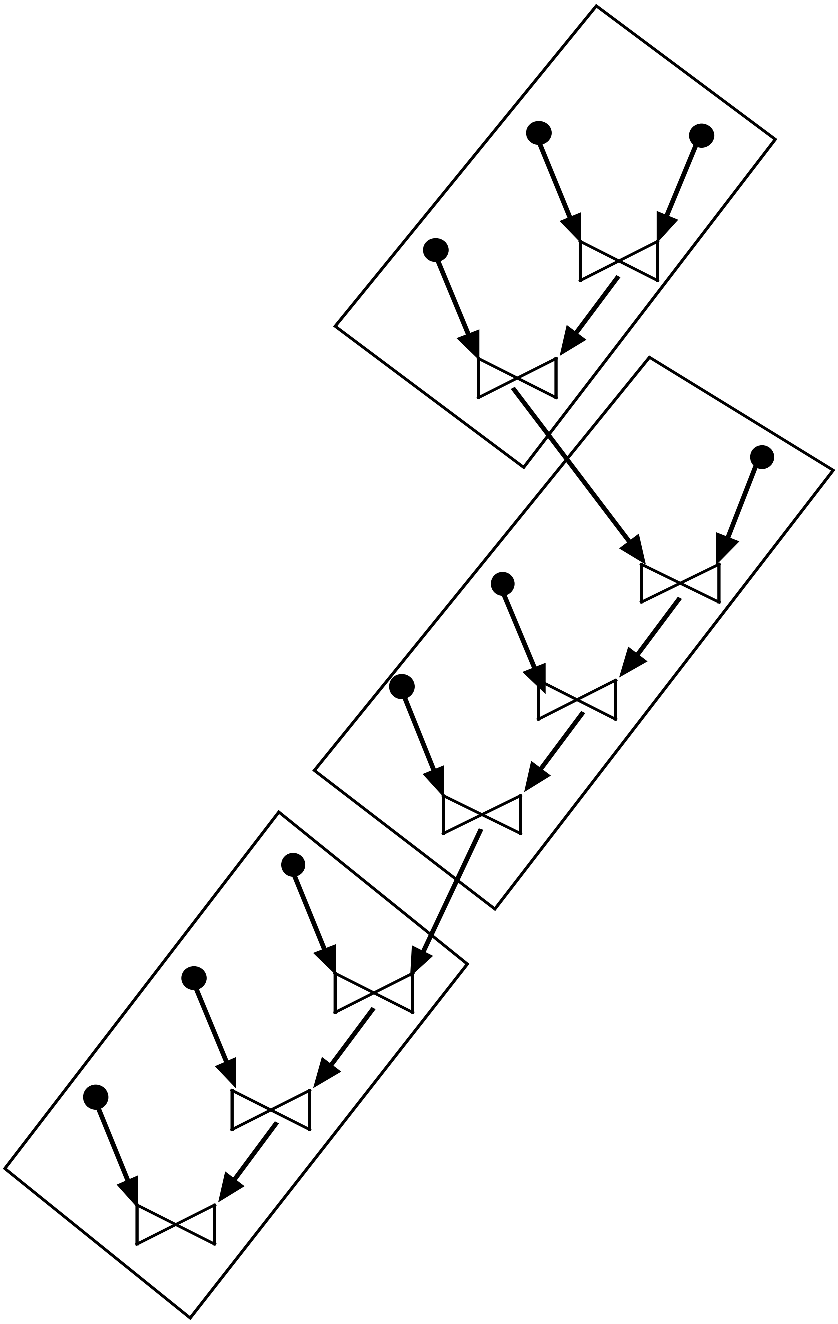
the loads of joins in a long pipeline chain are difficult to balance

inter-operator parallelism cannot be exploited

Zigzag tree

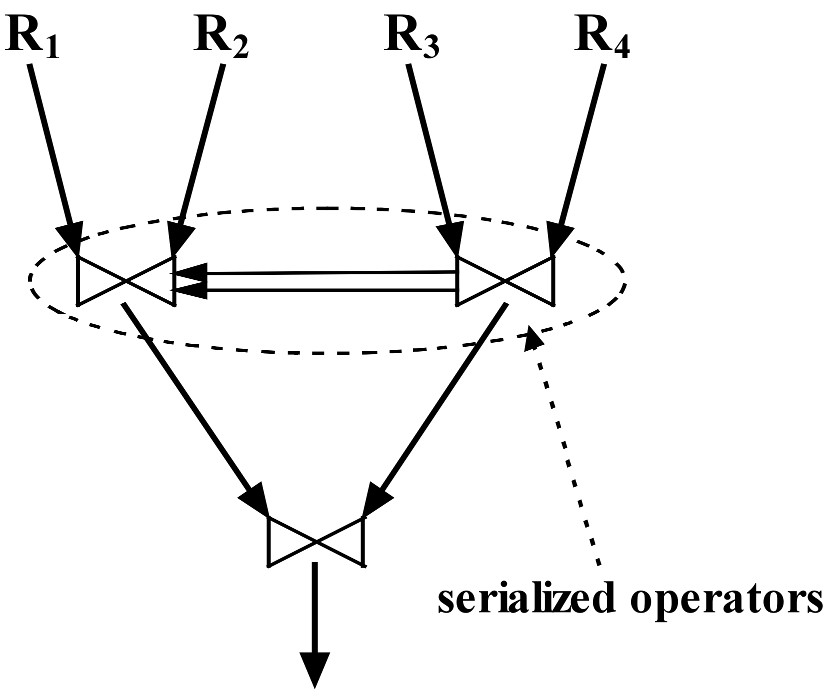
Zigzag tree = segmented right-deep tree

The result relation of a segment can be connected only to the *first join* of the next segment



Serialized bushy tree

Bushy tree + precedence edges between operations to serialize independent operators



Inter-operator parallelism oriented optimisation

Synchronised inter-operator parallelism

- proposed by Lu et al.
- no pipeline parallelism is taken into account
- the query processing is divided into synchronised steps
  - the joins executed in the next step must wait for the termination of the previous step
- greedy optimisation algorithm with polynomial complexity
- good for shared-everything systems
- extended for shared-nothing architecture by Hua et al.

Inter-operator parallelism oriented optimisation

PSA scheduling (Parallel Sheduling Algorithm)

- proposed by Hamurlain et al. in the PARIS project
- priorities are assigned to the operators ready to run
- priority is computed by estimating supplement cost of delaying an operator
- operators are executed on disjoint sets of processors (no time sharing)
- outperforms static right-deep scheduling and it is better than bushy tree scheduling up to 200 processors
- developed for single-query environments

Inter-operator parallelism oriented optimisation

Prisma/DB

- **PRISMA focuses on the problem of finding the best parallel scheduling for a given processing tree**
- **it implements right-deep tree, segmented right-deep tree and bushy tree scheduling**
- **uses synchronized parallel execution and full parallel execution strategies**
- **pipelined version of hash based join is used**

For many processors and for complex queries $\Rightarrow$ ***full parallel execution of bushy tree plans is the best***

Inter-operator parallelism oriented optimisation

IBM DB2 Parallel Edition

- **designed for shared nothing systems**
- **one-phase optimisation approach**
- **bushy tree scheduling with pruning trees from search space which exceed the best tree as of yet found**
- **only one of the three sets of processors can be assigned to perform a join: all processors, the processors accessing the inner relation ot the processors accessing the outer relation**

The best case for execution: if concurrent joins can be executed on a disjoint sets of processors

Summary of optimisation

One-Phase optimisation integrates parallelism in the optimisation process but suffers from combinatorial explosion in comparing with the Two-Phase approach

Clear tendency: exploit pipeline and inter-operator parallelism, especially for shared-nothing architectures

Two Phase	One Phase
Shared-everything architecture	Shared-nothing and hierarchial architectures
Simple queries	Complex queries

From single-query to multi-query environments

- **Inter-query + intra-query parallelism must be exploited**
- **Main problems**
  - **heterogeneous resouce availability**
  - **resource contentions between independent queries**
  - **additional costs due to sharing the resources among concurrent operations of one query (intra-query par.) and among queries**
- **Three step methodology**
  - **query execution frame (select resources for the given query)**
  - **scheduling + parallelising**
  - **dynamic control mechanism for run-time reoptimisation**

Search strategies in optimisation

Optimisation task

Generate possible physical operator trees (execution plans)

Compute the execution cost of the plan

Search for the plan with (near) minimal cost

This is a combinatorial optimisation problem

Algorithms for generating plans

- Exhaustive search
- breadth- and depth-first search with branch-and-bound
- randomised search
- polynomial heuristic search

Exhaustive search

Generates all possible operator trees

It has combinatorial complexity

Feasible for joins on few relations (max. 6 relations)

Breadth- and depth-first search with branch-and-bound

Deterministic search strategies

Iterative

- **The operator tree of *n* relations is built from the optimal subtrees of *n-i* relations combined with the optimal subtrees of *i* relations**

From a set of generated plans for *i* relations are generated the plans for *i+1* relations

Pruning function is applied at the expansion of the plan

Feasible algorithm for 10-15 joins and linear trees

Randomised search

1. The algorithm starts from a random state in the search space
2. It walks trough the search space, evaluating the cost of each state
3. The walking is controlled by transformation rules between trees and global strategy

Developed for bushy trees (inter-operator parallelism)

It cannot be guaranteed to be optimal

Polynomial heuristic search

Greedy algorithm: start from base relations and in each iteration build a new tree that has the lowest cost

Augmentation heuristic: adapt depth-first search with starting from the base relation with the least cardinality and at each expansion, only one succesor with the least cost is generated

Uniform greedy heuristic: generate complete trees starting from each base relation by augmentation heuristic and choose the least costly one