

Максим Милютин
m.milyutin@gmail.com

Каскадная оптимизация запросов на примере GP ORCA

О докладчике

- R&D SQL-движков в московском исследовательском центре Huawei
 - улучшение планирования запросов методами ML
- Более 7 лет работы с базами данных и PostgreSQL
 - SQL прикладная разработка в e-commerce
 - DBA и консалтинг решений вокруг PostgreSQL
 - Разработка патчей к PostgreSQL и Greenplum и расширений к ним
 - R&D разработка вокруг GaussDB

Два основных подхода к оптимизации запросов

- System R подход
 - разработан в 70-х годах на заре реляционных СУБД
 - динамическое программирование при планировании джоинов снизу-вверх
 - cost-based модель оценки альтернатив дерева плана
- Volcano / Cascade фреймворк
 - разработан в 90-х годах
 - планирование джоинов сверху-вниз
 - формализация логических и физических трансформаций (query rewrites) к изначальному/производному дереву плана
 - динамическая расширяемость набора правил трансформаций
 - формализация физических свойств (упорядоченность по атрибуту, распределение результата по кластеру и др.) выражений плана

System R подход. PostgreSQL оптимизатор

- Эвристические (необратимые) преобразования изначальной структуры запроса (Query)
 - предвычисление константных скалярных выражений
 - проброс (push down) предикатов под дерево джоинов
 - преобразование IN / EXISTS подзапросов в semi-join
 - слияние (pull-up) подзапросов и др.
- Cost-based планирование джоинов
 - для каждой таблицы, пары таблиц, троек и т.д. ищутся наиболее эффективные варианты планов (Paths), а также планы с выгодных порядком результирующих атрибутов
- Планирование GROUP BY агрегатов
 - следует за планированием джоинов и не смешивается

Volcano / Cascade фреймворк

- Планирование через мемоизацию выражений плана
 - структура *Мето* для хранения альтернатив дерева плана
- Применение логических/физических трансформаций к изначальному/производному дереву логических операторов запроса внутри *Мето*
 - каждая трансформация подвергается стоимостной оценкой, либо применяется эвристически
- Добавление свойств порядка, распределения и др. (property enforcement) к физическим узлам и выбор оптимального по стоимости плана
 - Стоимостная модель (cost model) абстрагирована и заменяема

Greenplum и оптимизатор ORCA

- Greenplum - распределенная СУБД для MPP обработки
 - потоки данных внутри плана запроса могут быть:
 - перераспределены по хэшу выбранного ключа - узел плана *Redistribute Motion*
 - реплицированы по всем узлам кластера - *Broadcast Motion*
 - собраны внутри одного узла - *Gather Motion*
- ORCA - каскадный оптимизатор для планирования распределенных запросов на замену “legacy” PostgreSQL оптимизатору
 - изначально разрабатывалась как подключаемый компонент для внешних движков исполнения (например, HAWQ - SQL поверх Hadoop)
 - на данный момент полностью включена в дерево исходников Greenplum

Greenplum и оптимизатор ORCA

-- Таблица test1 соединяется не по ключу распределения (shuffle join)

explain (costs off)

select count(1) from test1 join test2 using (b);

QUERY PLAN

Aggregate

slice0

-> Gather Motion 2:1 (slice2; segments: 2)

slice2

-> Aggregate

-> Hash Join

Hash Cond: (test1.b = test2.b)

-> Redistribute Motion 2:2 (slice1; segments: 2)

Hash Key: test1.b

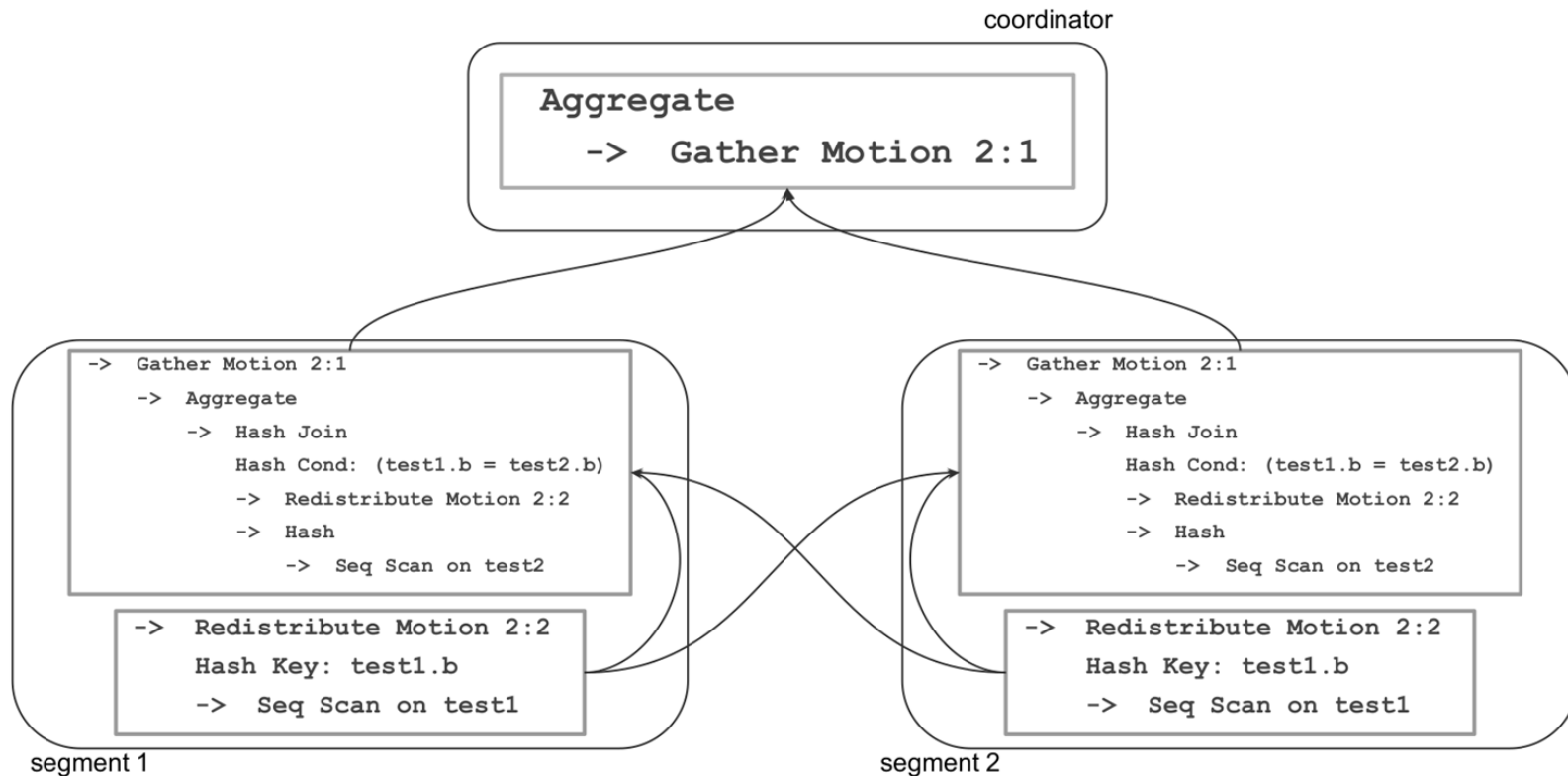
-> Seq Scan on test1

slice1

-> Hash

-> Seq Scan on test2

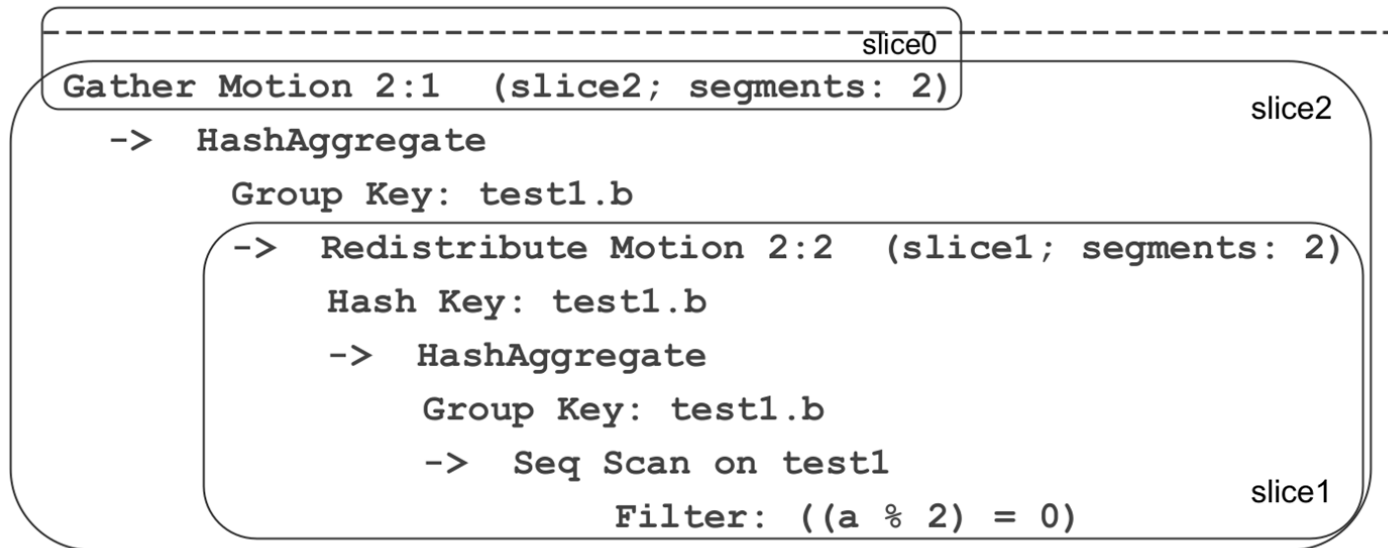
Greenplum и оптимизатор ORCA



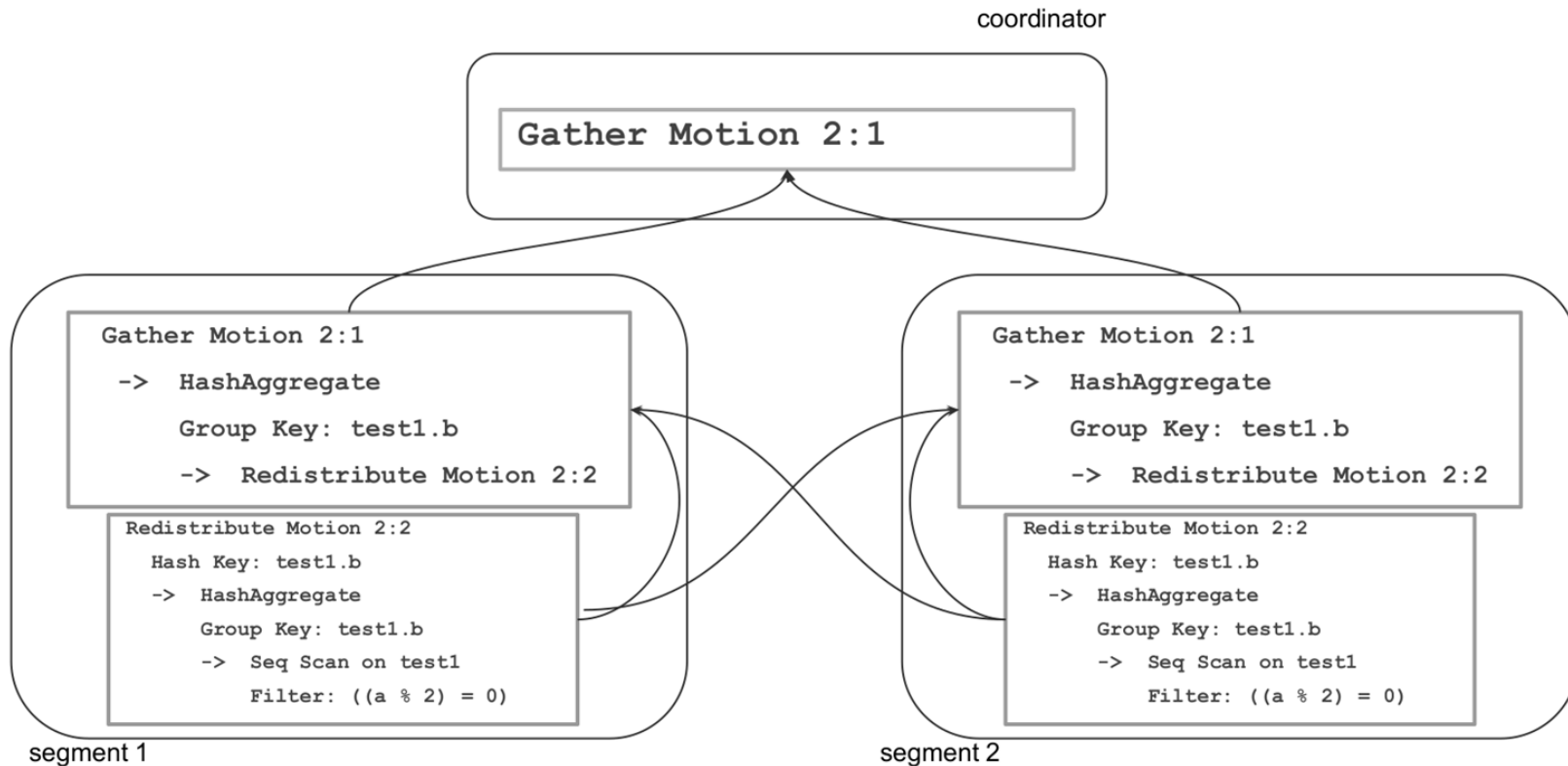
Greenplum и оптимизатор ORCA

```
-- Таблица test1 группируется не по ключу распределения  
explain (costs off)  
select count(1) from test1 where a%2=0 group by b;
```

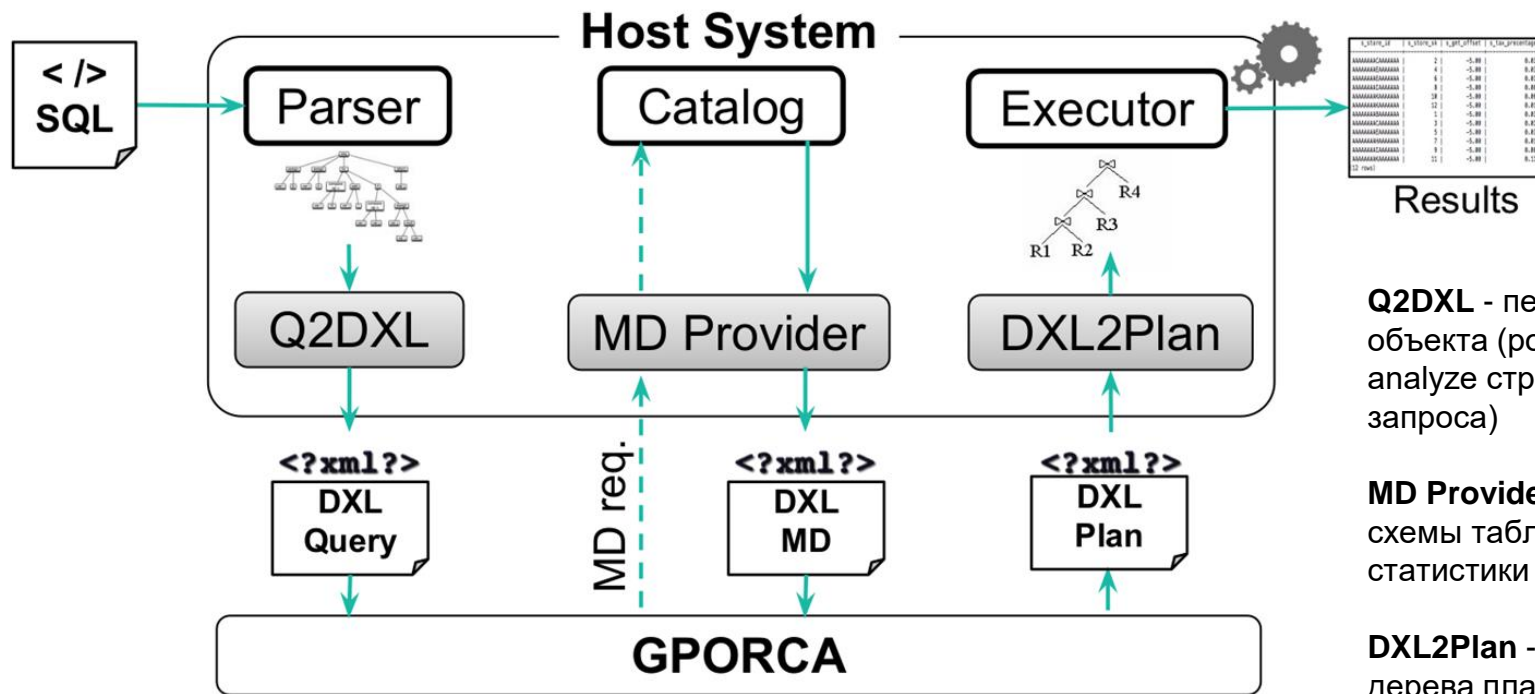
QUERY PLAN



Greenplum и оптимизатор ORCA



ORCA. Взаимодействие с движком СУБД

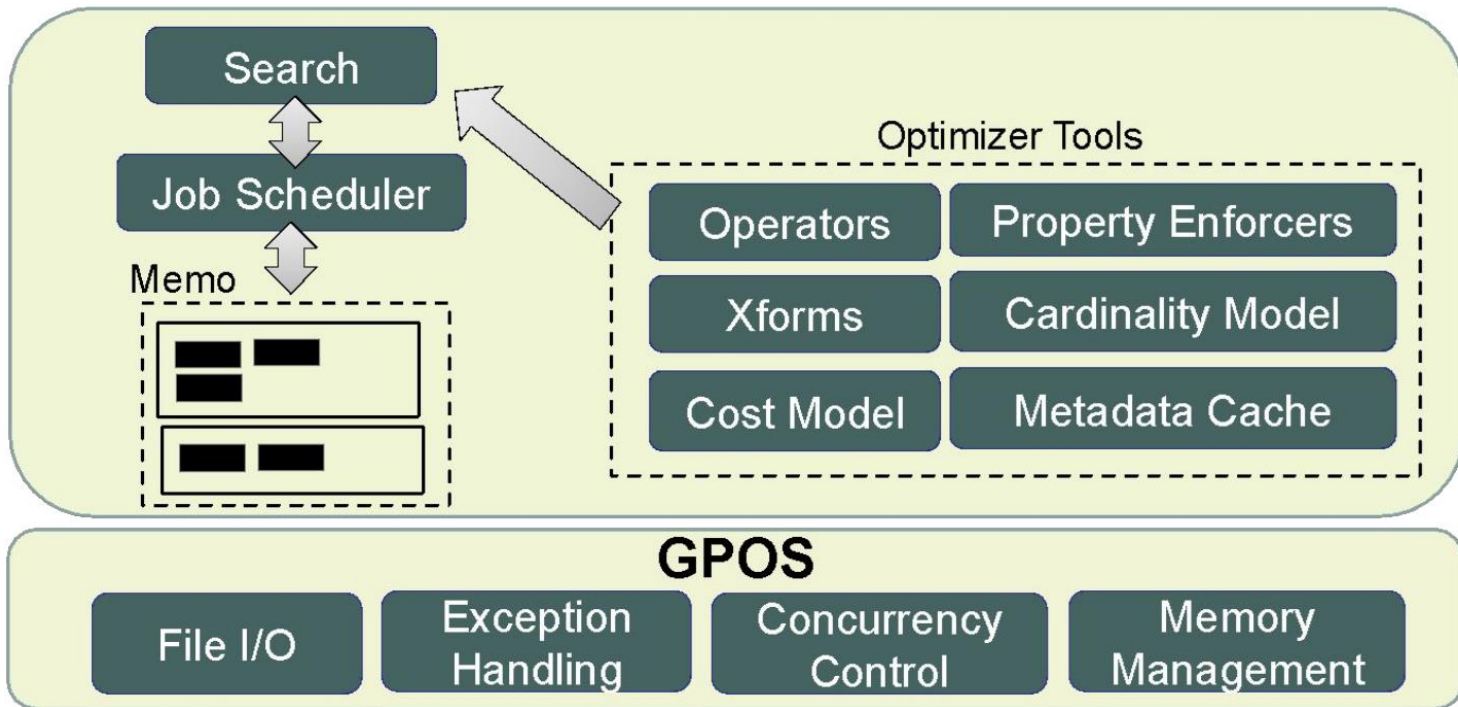


Q2DXL - передача Query объекта (post-parse-analyze структуры запроса)

MD Provider - передача схемы таблиц и статистики по колонкам

DXL2Plan - возврат дерева плана

ORCA. Внутренние компоненты



Operators - объекты логических/физических операторов

Xforms - преобразования над операторами

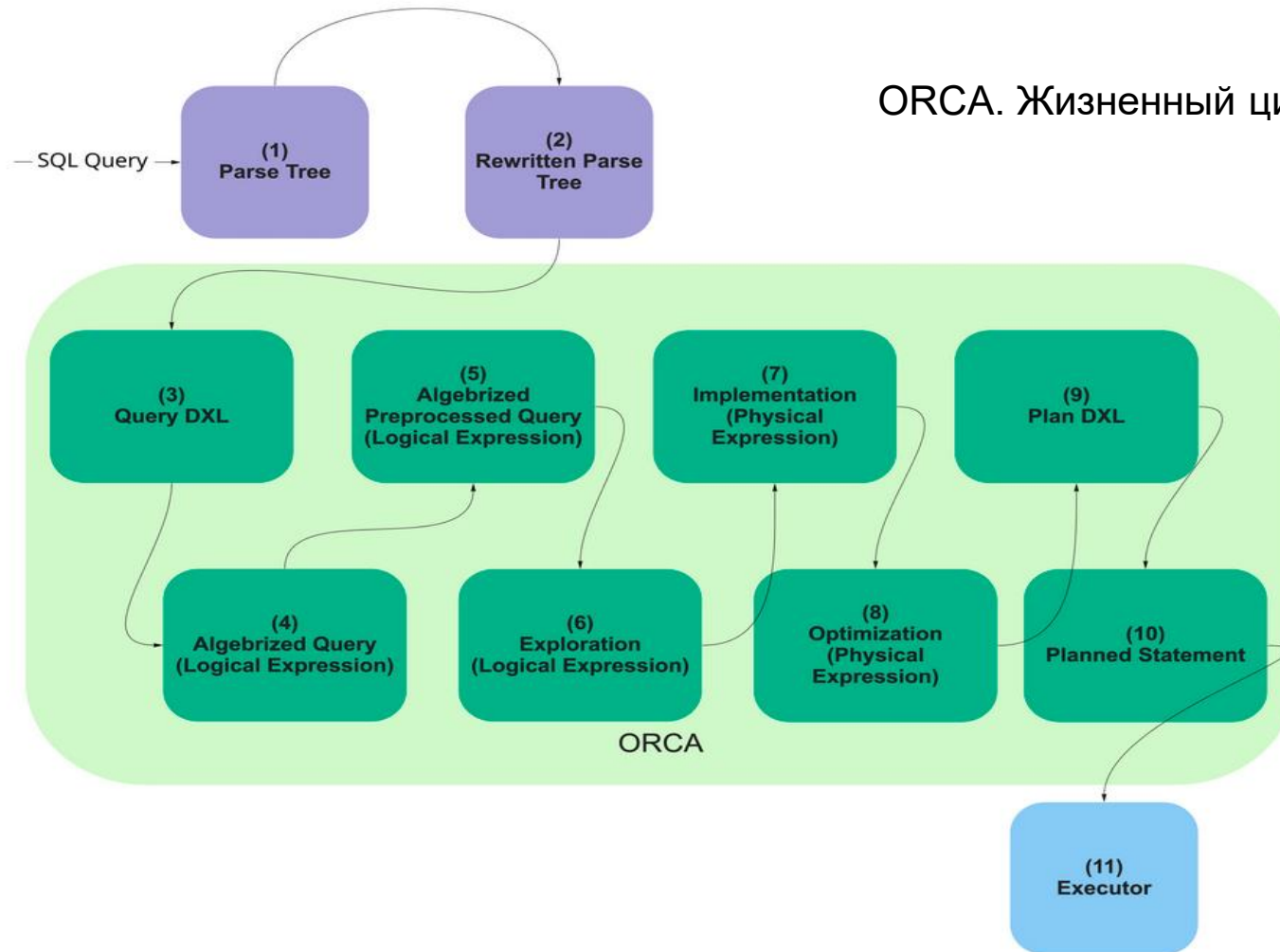
Property Enforcers (EnfdProp) - физические свойства

Metadata Cache - кэш для схемы таблиц и статистики колонок

ORCA. Стадии оптимизации

- Pre-process
 - нормализация / применение упрощающих преобразований
- Exploration
 - применение логических трансформаций внутри Memo
- Statistics Derivation
 - вычисление гистограмм по атрибутам и кардинальности для каждой группы эквивалентных выражений
- Implementation
 - преобразование логических операторов в физические
- Optimization
 - property enforcement
 - оценка альтернатив плана (физических операторов) и выбор оптимального

ORCA. Жизненный цикл запроса



ORCA. Пример запроса

```
explain (costs off)
select i, count(1) from test left join test_1 using(i)
where test_1.i is not null
group by i;
```

QUERY PLAN

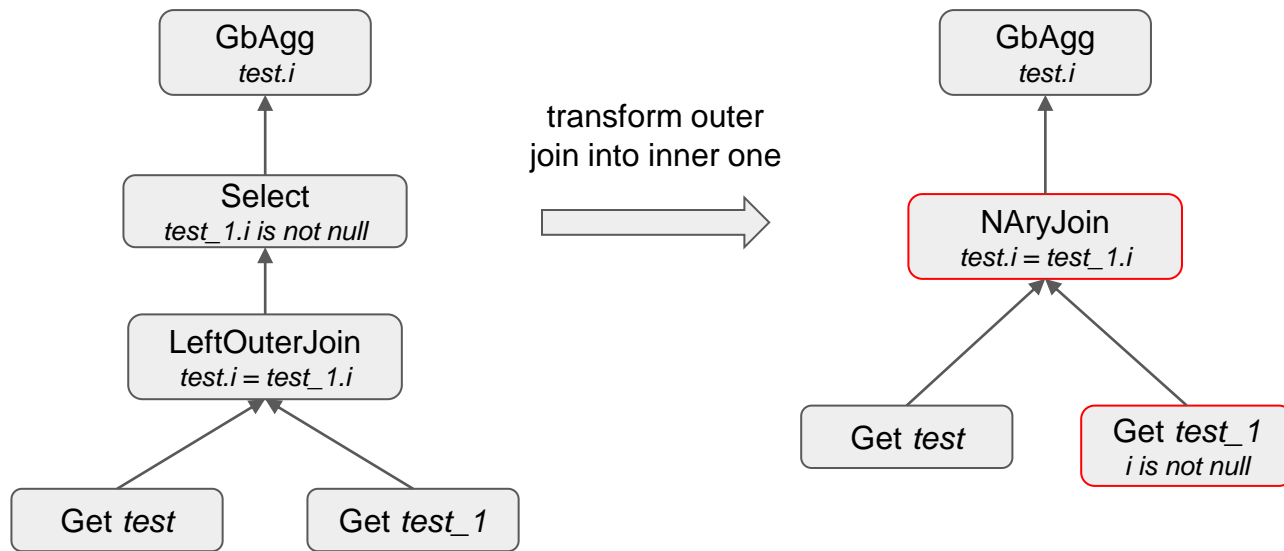
```
Gather Motion 2:1 (slice2; segments: 2)
-> Hash Join
    Hash Cond: (test.i = test_1.i)
    -> HashAggregate
        Group Key: test.i
        -> Redistribute Motion 2:2 (slice1; segments: 2)
            Hash Key: test.i
            -> Seq Scan on test
    -> Hash
        -> Seq Scan on test_1
            Filter: (NOT (i IS NULL))
```

Optimizer: Pivotal Optimizer (GPORCA)

```
Table "public.test"
Column | Type   | Modifiers
-----+-----+-----
i       | integer |
Distributed randomly
```

```
Table "public.test_1"
Column | Type   | Modifiers
-----+-----+-----
i       | integer |
Indexes:
    "test_1_i_key" UNIQUE
CONSTRAINT, btree (i)
Distributed by: (i)
```

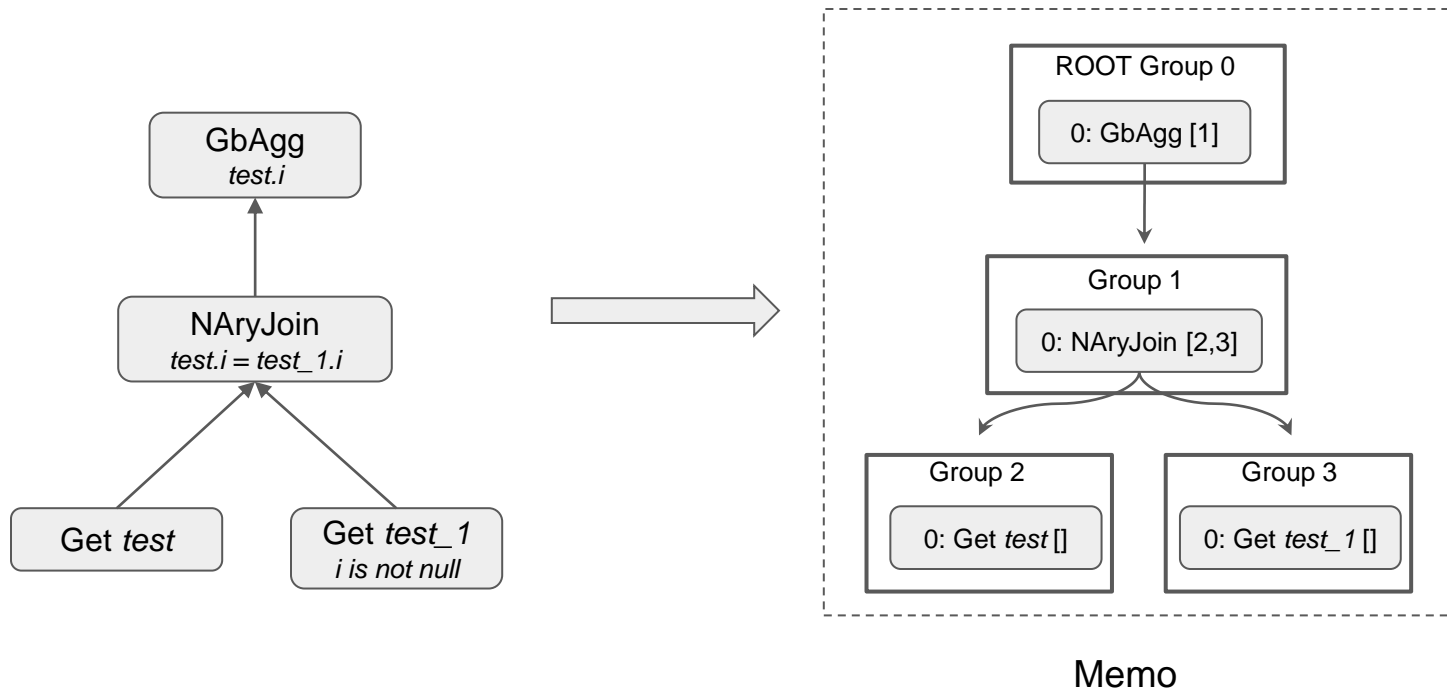
ORCA. Pre-process



- 28* преобразований
- большинство про нормализацию выражений и дерева плана

* на текущее состояние версии 6X <https://github.com/greenplum-db/gpdb/blob/e58a08f3aba64ef005624ff9d0a0963f8b6c056e/src/backend/gporca/libgpopt/src/operators/CEXpressionPreprocessor.cpp#L3060-L3267>

ORCA. Exploration. Memoization

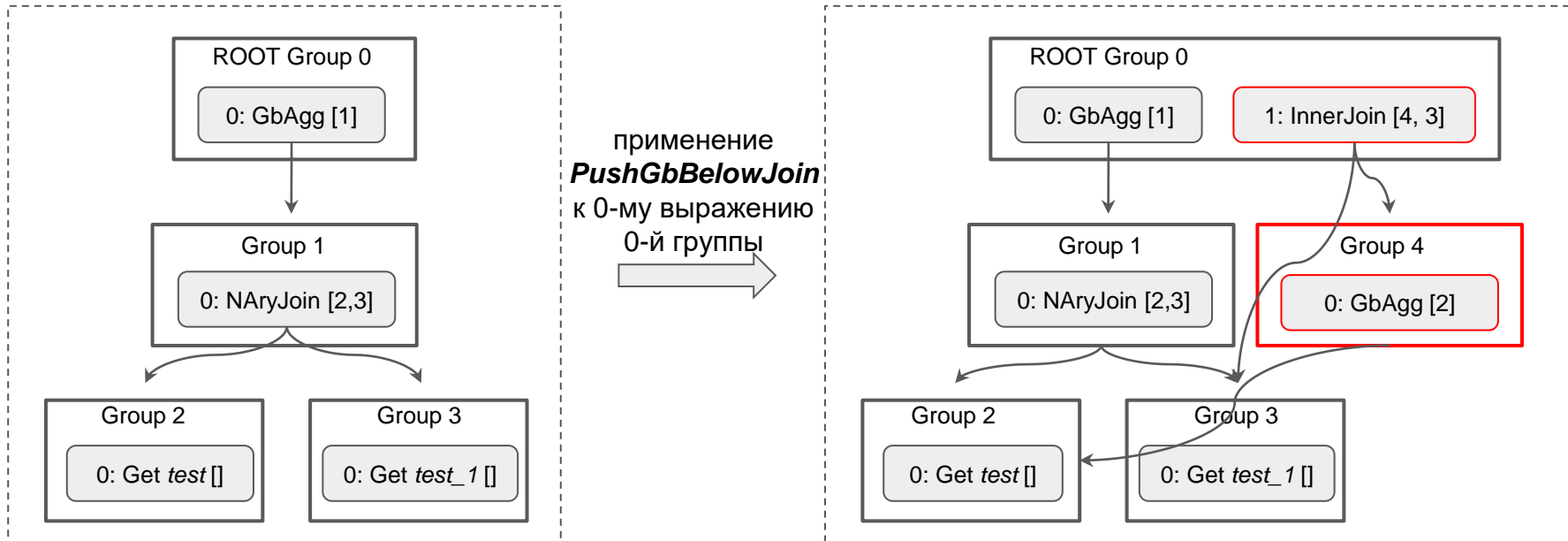


Memo - And-Or граф для компактного представления альтернатив плана в памяти

Group - совокупность эквивалентных выражений

Group Expression - оператор, ссылающийся на другие группы

ORCA. Exploration



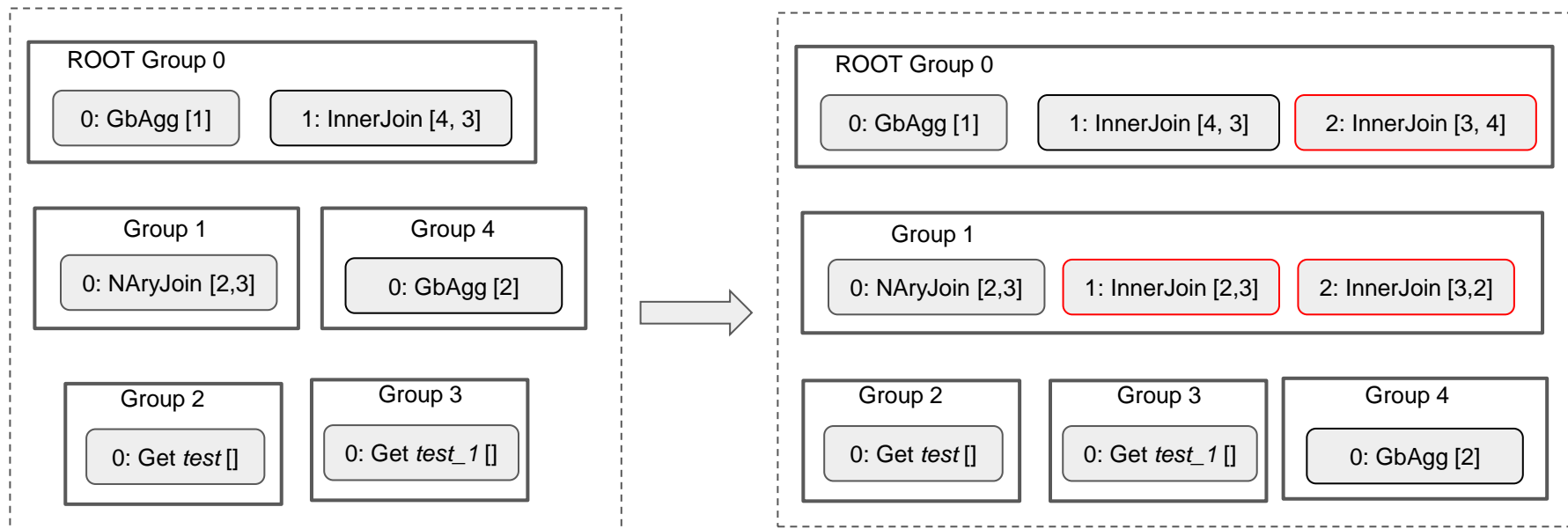
ORCA. Exploration

NAryJoinGreedy

к 0-му выражению 1-й группы

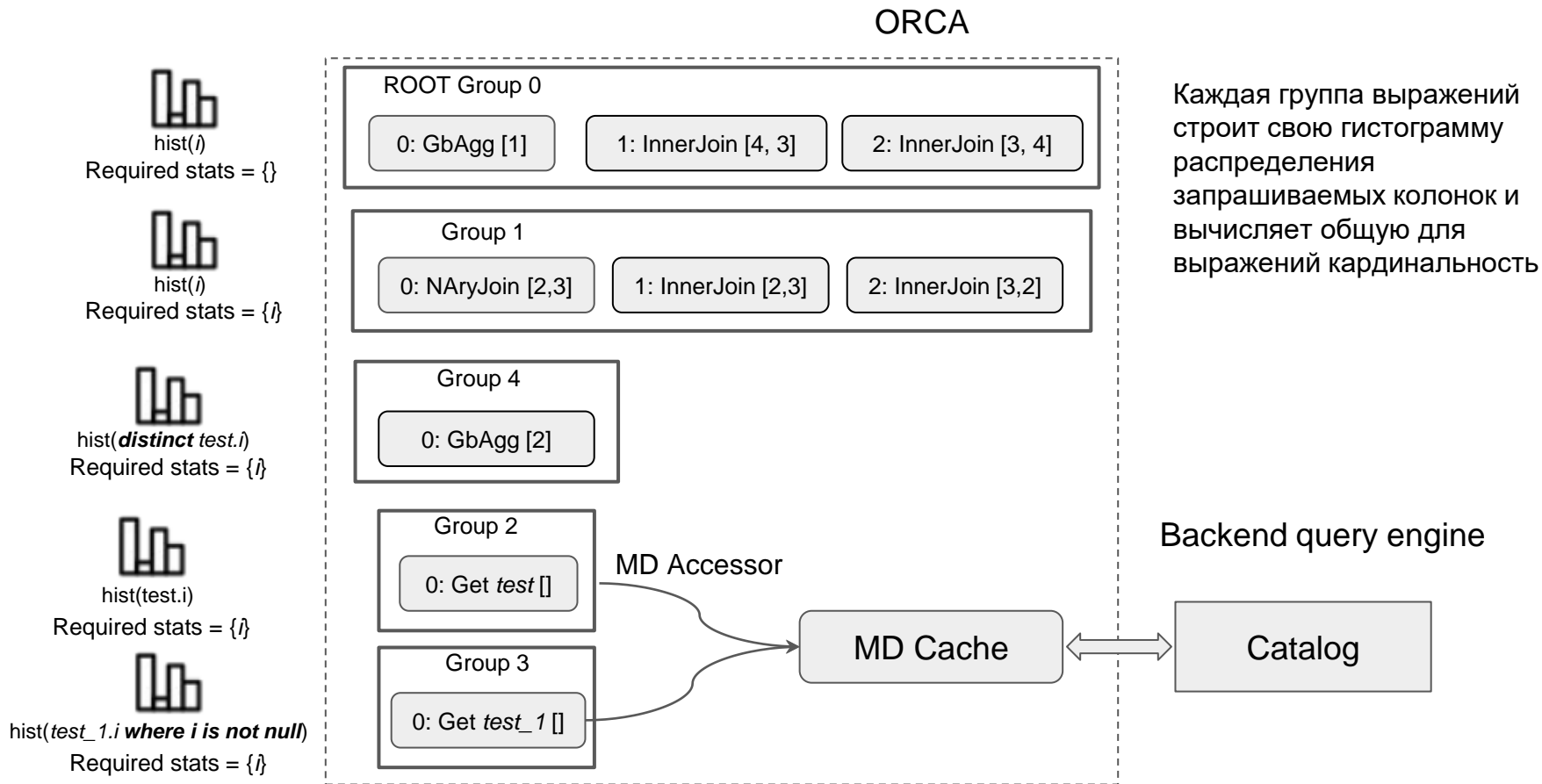
JoinCommutativity

к 1-му выражению 0-й группы
и 1-му выражению 1-й группы

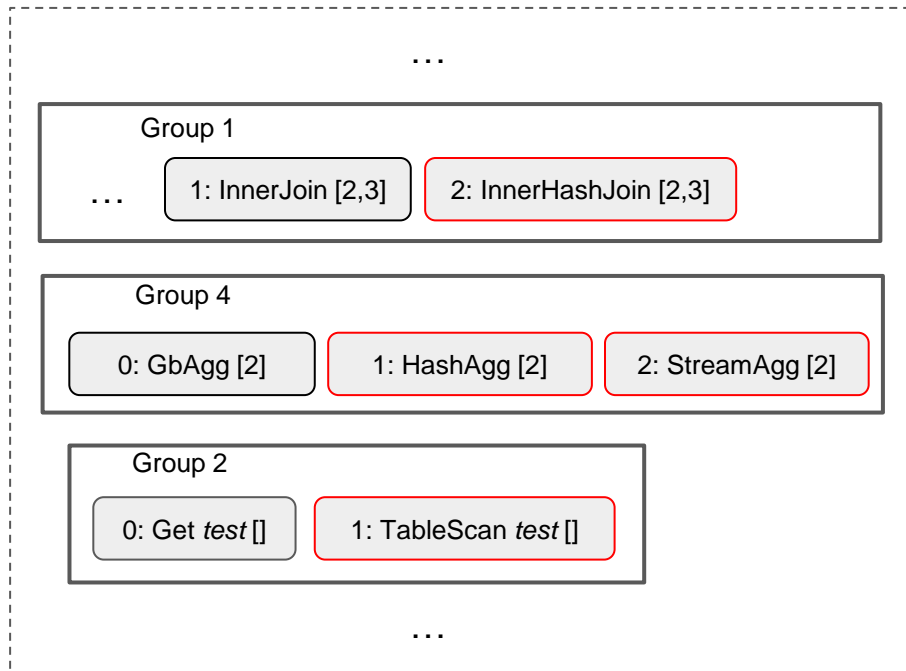


NAryJoinGreedy - разложение N-арного джоина на бинарные с минимальными кардинальностями на промежуточных джоинах и откладыванием cross джоинов на конец. Одна из вариаций GOO (Greedy Operator Ordering)

ORCA. Statistics Derivation

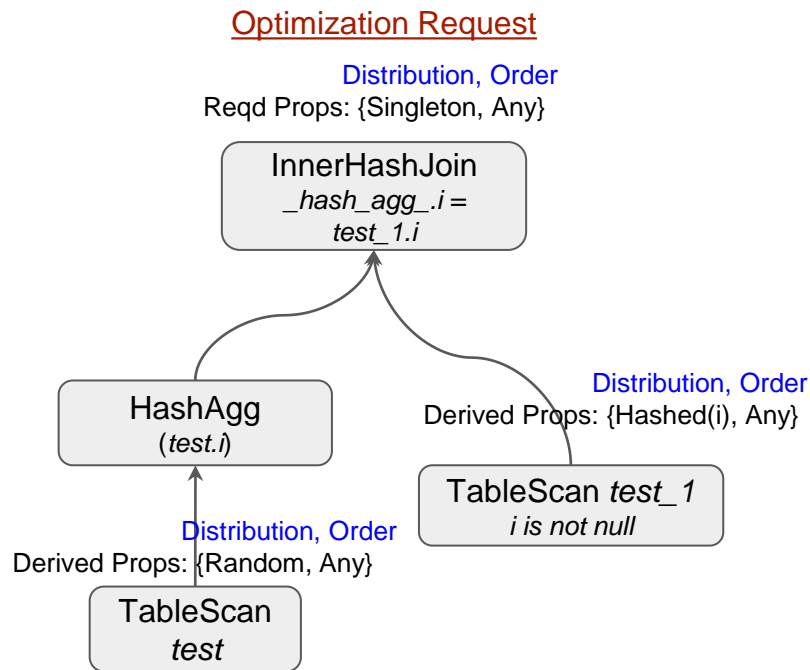


ORCA. Implementation



- Каждому конечному логическому оператору задаётся физический
- *InnerJoin* транслируется в *HashJoin* (в приоритете), либо в *NestedLoop*
- Пока ничего не говорится об упорядоченности потоков данных и о распределении потоков по узлам кластера

ORCA. Optimization. Requirement



Reqd Props - физические свойства, запрашиваемые у оператора сверху на результирующий поток данных

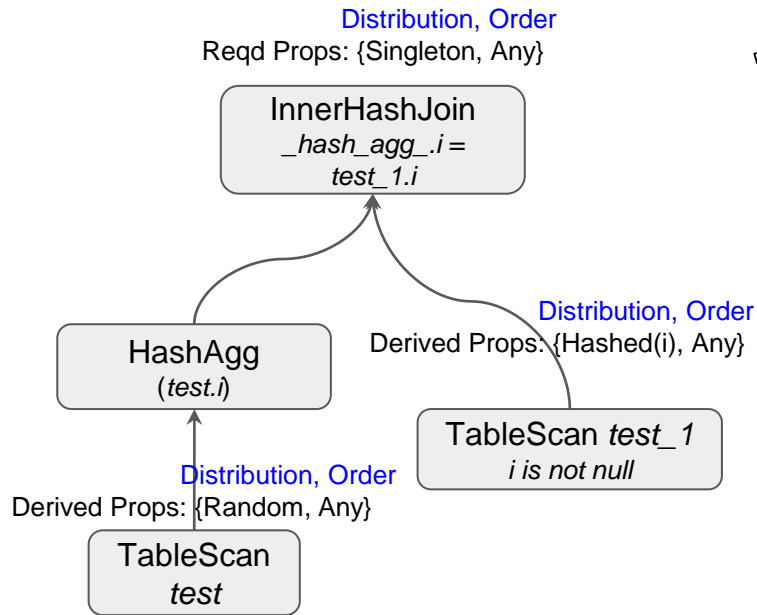
Derived Props - свойства, которые обеспечивает оператор

Задача: максимально эффективно сопоставить эти свойства

Физические свойства в ORCA:
Distribution, Order, Rewindability, Column Output и др.

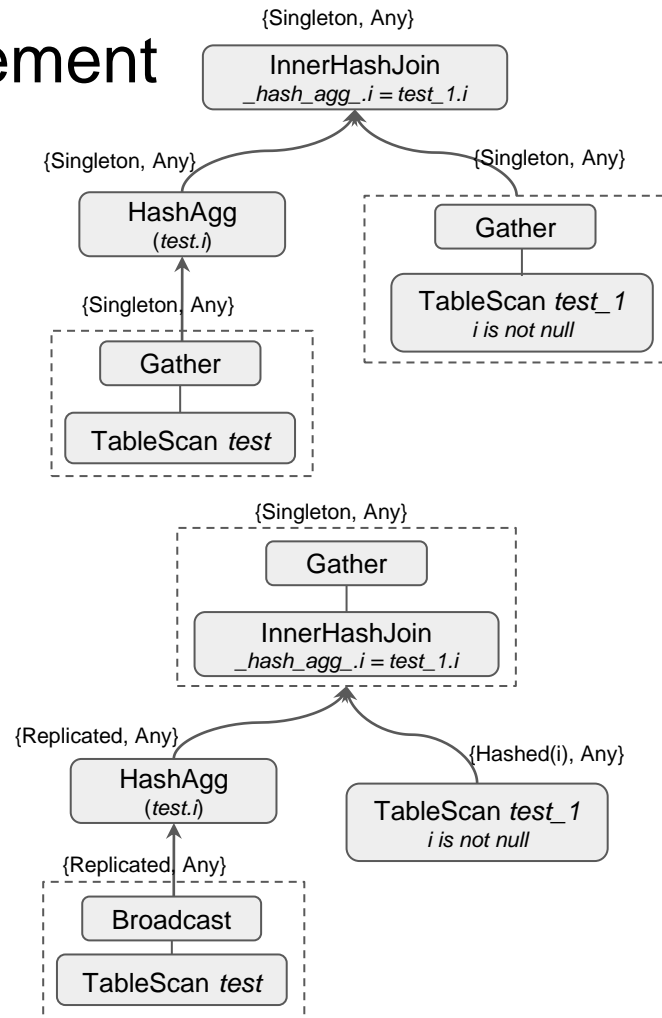
ORCA. Optimization. Property Enforcement

Optimization Request



проброс сбора данных на
координаторе к сканам

реплицировать левое
поддерево джоина

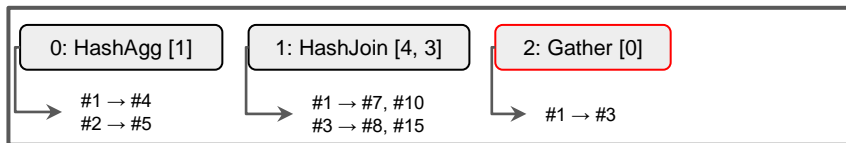


ORCA. Optimization. Costing. Property Enforcement

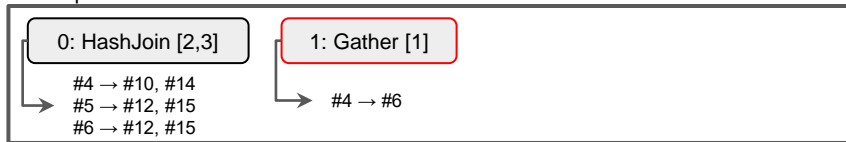
Groups Hash Tables

Memo

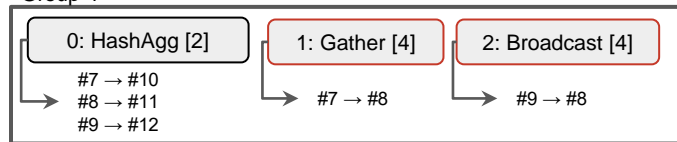
ROOT Group 0



Group 1



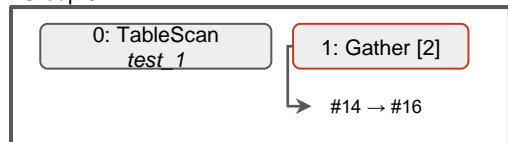
Group 4



Group 2



Group 3



#	Opt. Request	Best Expr
1	Singleton, Any	2
2	Hashed(i), Any	1
3	Any, Any	1

#	Opt. Request	Best Expr
4	Singleton, Any	1
5	Hashed(i), Any	0
6	Any, Any	0

#	Opt. Request	Best Expr
7	Singleton, Any	1
8	Hashed(i), Any	0
9	Replicated, Any	2

#	Opt. Request	Best Expr
10	Singleton, Any	1
11	Hashed(i), Any	2
12	Replicated, Any	3
13	Any, Any	0

#	Opt. Request	Best Expr
14	Singleton, Any	1
15	Hashed(i), Any	0
16	Any, Any	0

- Общий набор требуемых физических свойств в виде *optimization request* пробрасывается в родительскую группу и распространяется в различных вариациях по всем выражениям дочерних групп
- Gather, Redistribute, Broadcast, Sort, Materialize* и др. - узлы enforcer`ы для достижения требуемых свойств результирующего потока данных
- Groups Hash Table** - отображение *optimization request* (запрашиваемых физических свойств на операторы группы) в **наилучшее по стоимости** выражение группы
- Каждое выражение хранит свою карту (наиболее оптимального?) отображения *optimization request* к этому выражению на запрашиваемые *optimization request* к дочерним группам

ORCA. Optimization. Building final plan

Groups Hash Tables

Opt. Request Best Expr

1 Singleton, Any 2

2 Hashed(i), Any 1

3 Any, Any 1

Opt. Request Best Expr

4 Singleton, Any 1

5 Hashed(i), Any 0

6 Any, Any 0

Opt. Request Best Expr

7 Singleton, Any 1

8 Hashed(i), Any 0

9 Replicated, Any 2

Opt. Request Best Expr

10 Singleton, Any 1

11 Hashed(i), Any 2

12 Replicated, Any 3

13 Any, Any 0

Opt. Request Best Expr

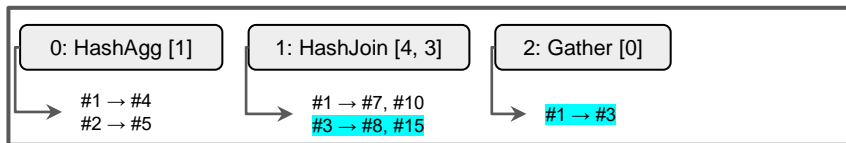
14 Singleton, Any 1

15 Hashed(i), Any 0

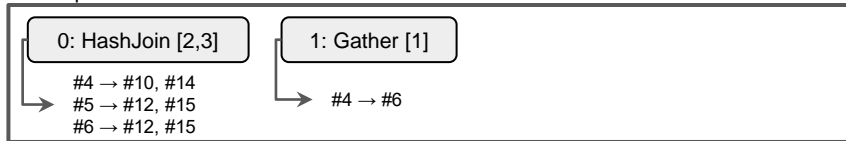
16 Any, Any 0

Memo

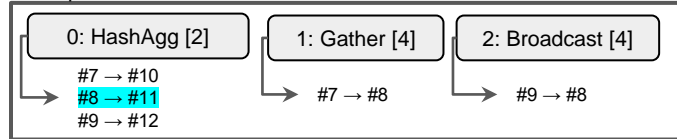
ROOT Group 0



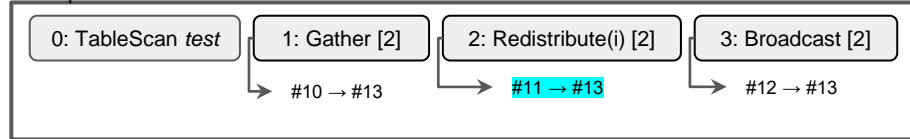
Group 1



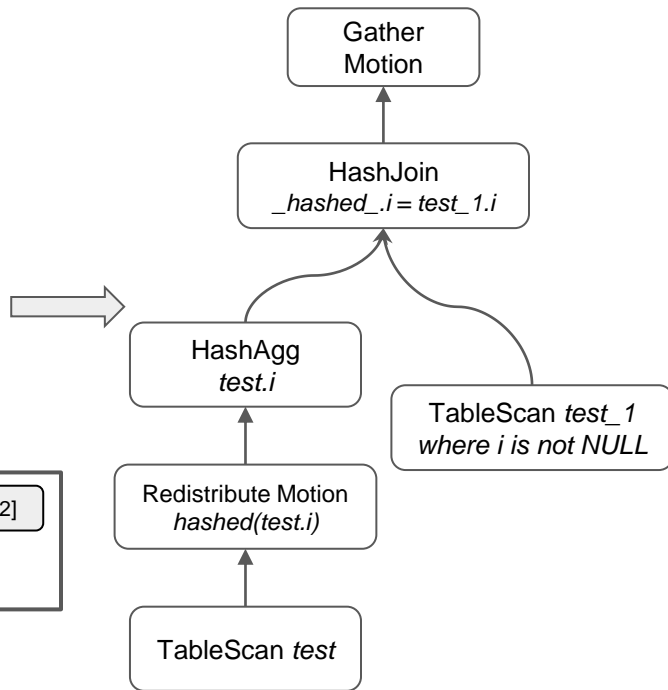
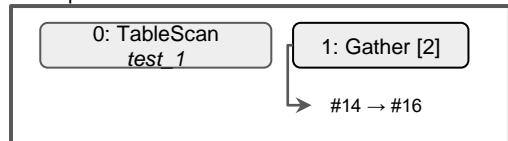
Group 4



Group 2



Group 3

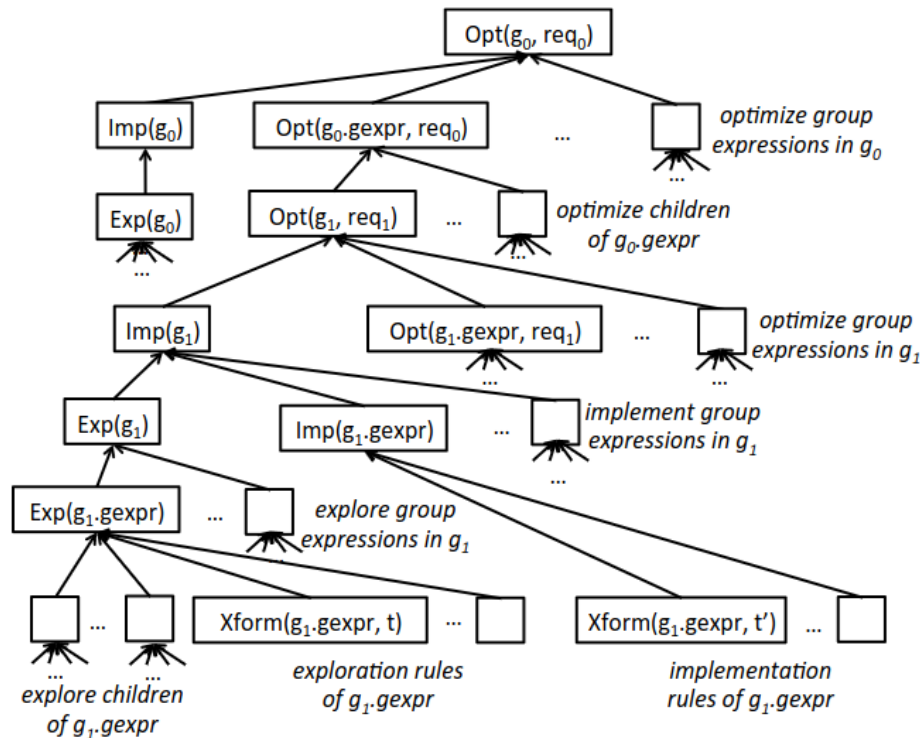


ORCA. Полезные команды для дебага/трассировки

- **Логирование стадий оптимизации:**
 - дерево запроса (начальное и после стадии pre-process):
 - `set optimizer_print_query=on`
 - структура Мемо на различных стадиях:
 - `set optimizer_print_memo_after_exploration=on`
 - `set optimizer_print_memo_after_implementation=on`
 - `set optimizer_print_memo_after_optimization=on`
 - вместе с трансформациями `set optimizer_print_xform=on`
 - различная статистика исполнения
 - `set optimizer_print_optimization_stats=on`
- **Активация/деактивация трансформаций**
 - `select enable_xform('CXformExpandNAryJoinGreedy')`
 - `select disable_xform('CXformExpandNAryJoinGreedy')`

ORCA. Распараллеливание оптимизатора

Граф зависимостей задач




- Весь процесс оптимизации поделен на задачи (jobs):
 - $CJobGroupOptimization$ (“Opt”) - optimization стадия для группы
 - $CJobGroupImplementation$ (“Imp”) - implementation стадия для группы
 - $CJobTransformation$ (“Xform”) - применение конкретной трансформации по выражению и др.
- Управляет всеми задачами Job Scheduler
- Имея граф зависимостей между задачами, можно распараллелить независимые задачи на пуле потоков
- На данный момент возможность распараллеливания выпилена [1], но разбиение на job`ы и сам Job Scheduler остался

ORCA. Инфраструктура для тестирования

- ORCA может тестироваться изолированно от движка СУБД
- Имея необходимые данные системного каталога &&
тело запроса &&
фиксированные значения GUC-параметров (про планирование) &&
детерминированное поведение планировщика,
можно сравнивать реальный план с ожидаемым
- Утилиты тестирования:
 - *gporca_test* - запуск конкретного теста в форме minidump (XML-формат) файла
 - *ctest* - запуск группы тестов

ORCA. Полезные ссылки для практики

- Советы по дебагу
 - <https://gist.github.com/darthunix/bd9467e89dca6e18b37b2125c49f2205>
- Training Boot Camp 
 - <https://github.com/greenplum-db/gpdb/wiki/ORCA-Training-Boot-Camp>
- Внутреннее устройство && навигация по кодовой базе && советы по дебагу
 - <https://github.com/Aaaaaaaron/database-papers/blob/main/optimizer/The%20Internals%20of%20GPORCA%20Optimizer.pdf>

Спасибо за внимание!
Вопросы. Критика. Пожелания...

Максим Милютин milyutinma@gmail.com