Postgres Pro Shardman

PGMeetup.DBA

Распределенная горизонтально масштабируемая СУБД

Павел
Конотопов

PostgresPro

PGMeetup.DBA

ПАВЕЛ КОНОТОПОВ

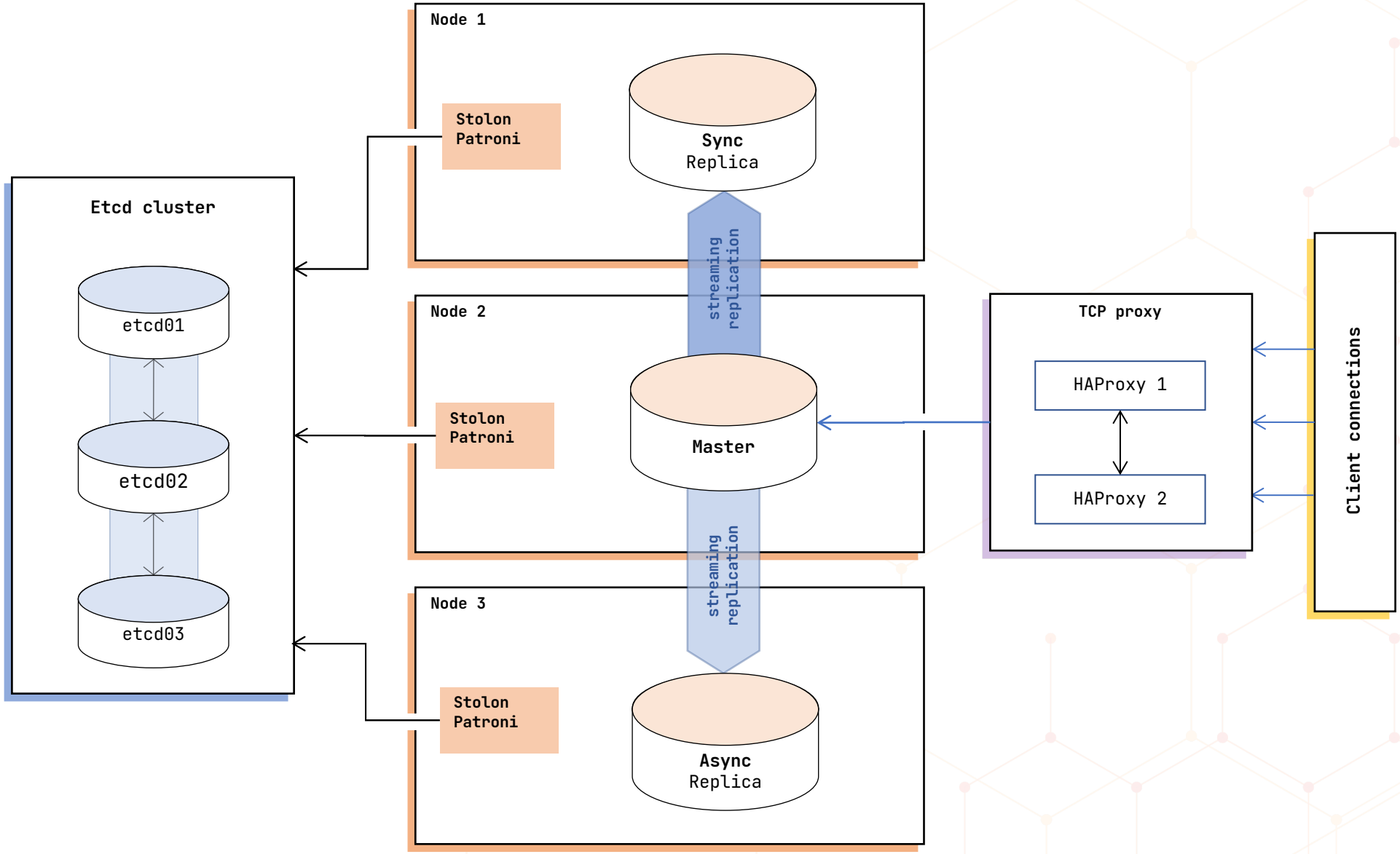
руководитель кластерной группы департамента внедрения и технической поддержки

PostgresPro

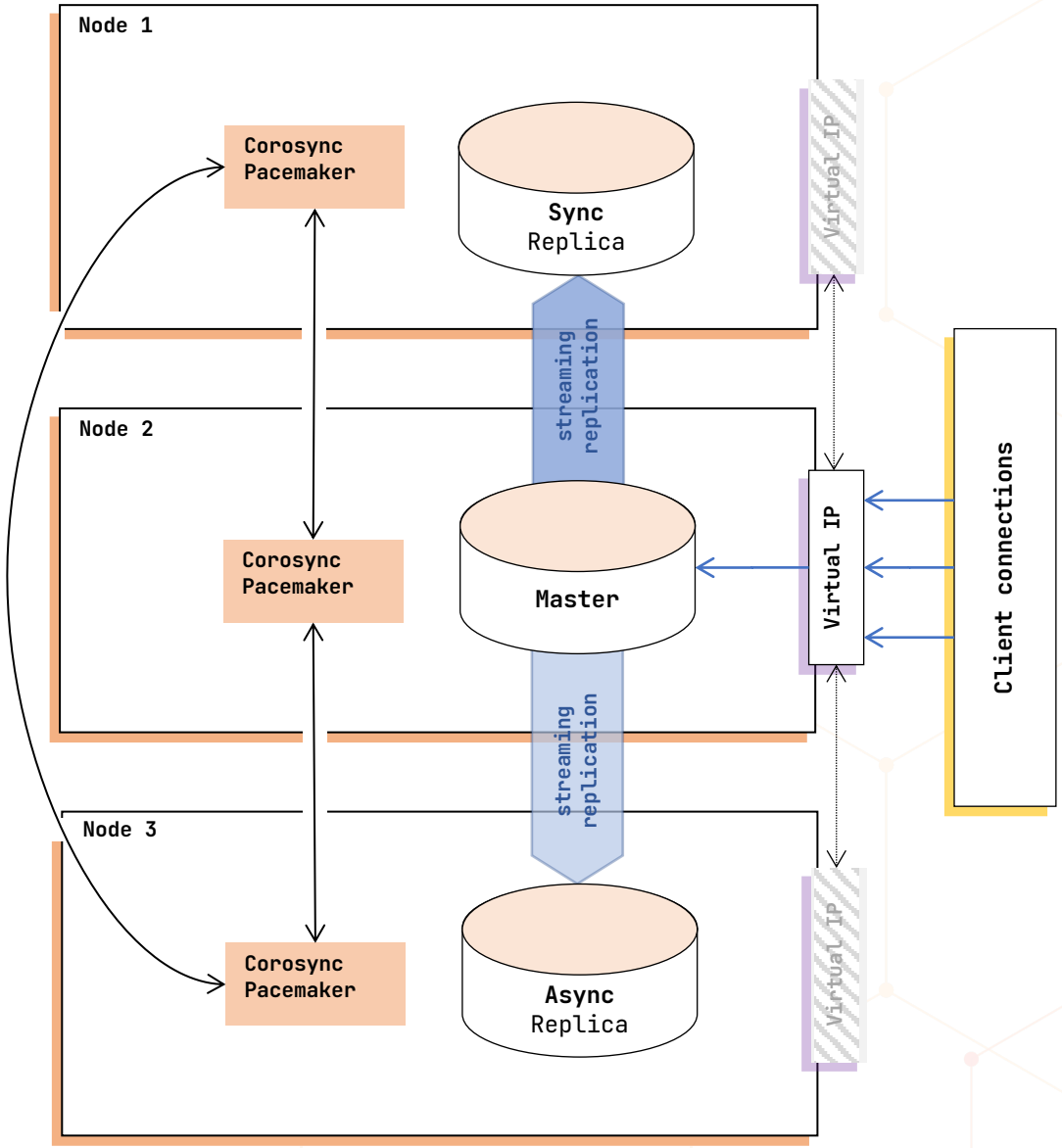
- 20+ лет в ИТ
- Инженер по отказоустойчивости PostgreSQL
- Опыт администрирования около 300 PostgreSQL кластеров в продуктивном окружении
- Последние пять лет работаю с PostgreSQL
- Последние два года работаю в Postgres Professional

LinkedIn:
https://www.linkedin.com/in/pavel-konotopov
Email:
pavel.konotopov@postgrespro.ru

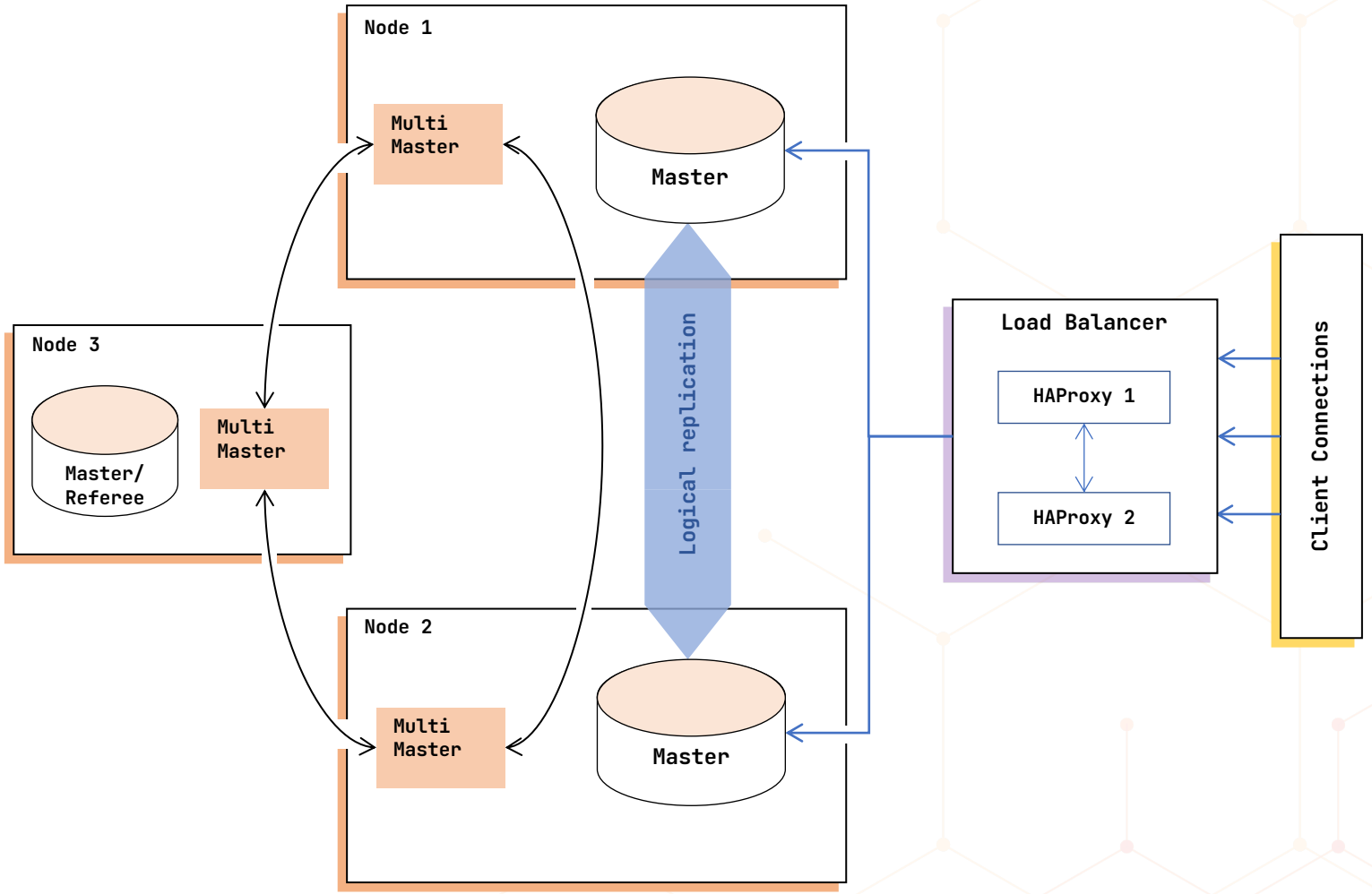
Полугоризонтальное масштабирование



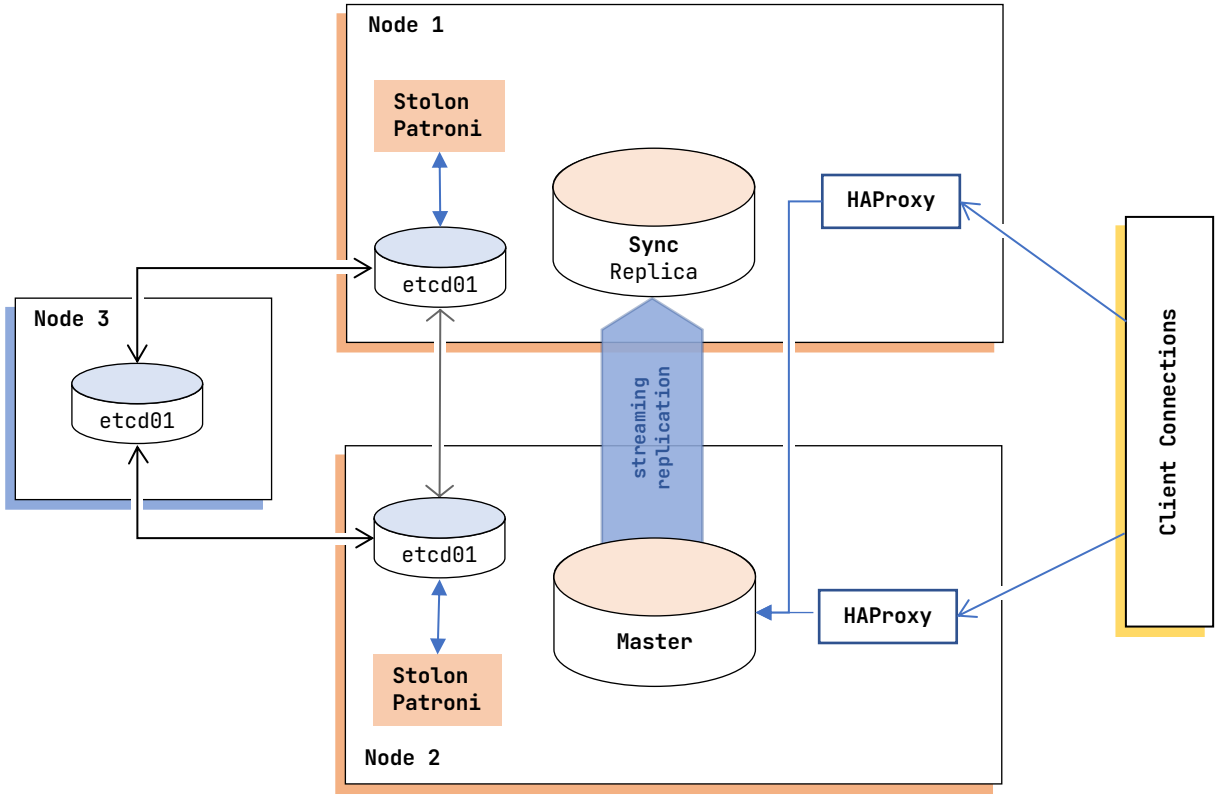
Полугоризонтальное масштабирование

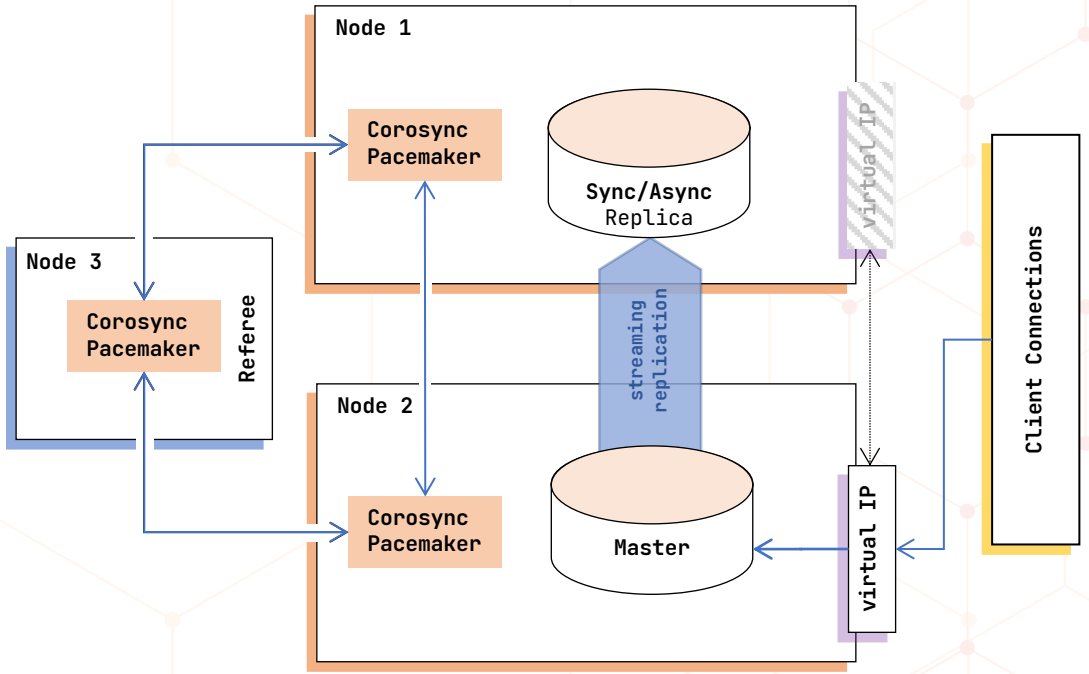


Полугоризонтальное масштабирование



Экономвариант





Преимущества Shardman

PostgresPro

- Прозрачное горизонтальное масштабирование
- Рассчитан на OLTP нагрузку
- Доступ в кластер через любой узел
- Встроенная поддержка отказоустойчивости
- Избыточность данных
- Целостность данных

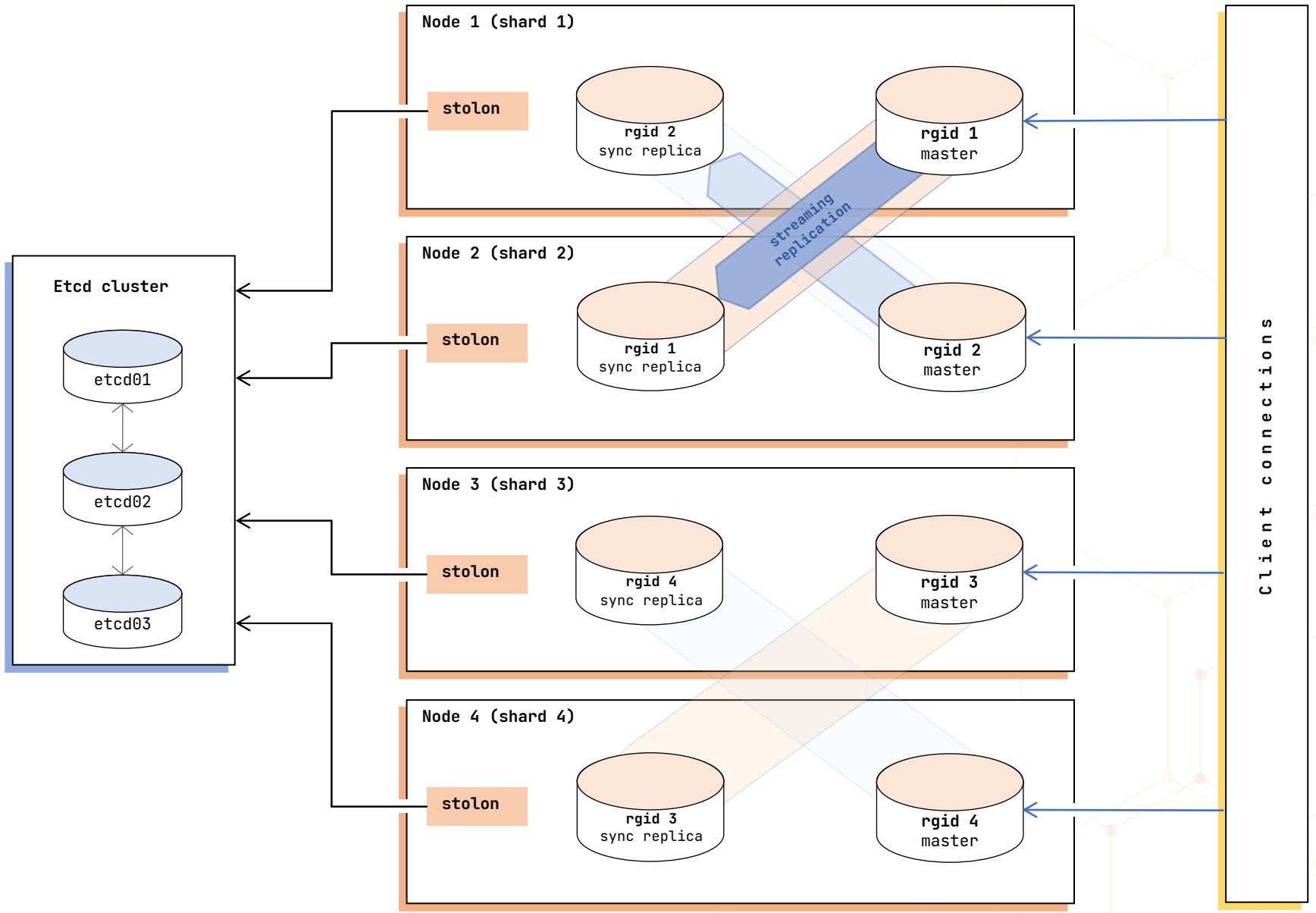
Программные компоненты

- PostgreSQL 14
- Stolon
- расширение Shardman
- Shardmanctl
  управление кластером
- shardmand
  сервис, отвечающий за кластерное ПО
- Etcd
  распределенное хранилище ключ-значение

Используемые технологии

- Стандартное секционирование – partitioning
- Foreign data wrapper – расширение postgres_fdw
- Синхронная физическая репликация
- CSN – commit sequence number (Clock-SI)

Архитектура



Развертывание кластера

- В ИТ инфраструктуре нужно иметь развернутую систему DCS
- Установка пакетов
  - postgrespro-sdm-14-server
  - postgrespro-sdm-14-contrib
  - postgrespro-sdm-14-client
  - postgrespro-sdm-14-libs
  - shardman-services
  - shardman-tools
  - stolon-sdm
- shardmanctl
  - инициализация конфигурации кластера в DCS
  - добавление узлов
  - строка подключения

Конфигурация кластера

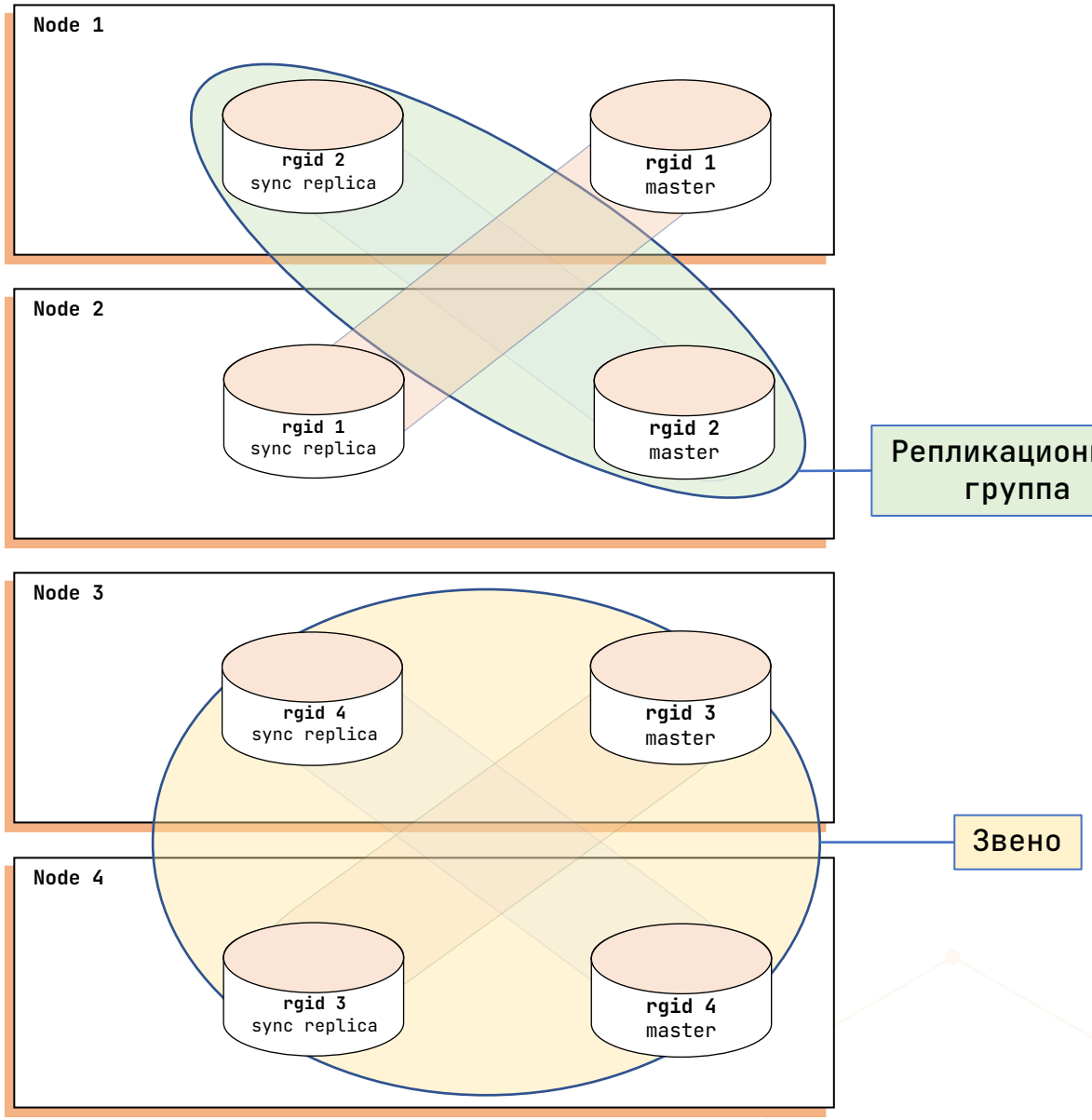
```
{
"LadleSpec": // Параметры для всех узлов кластера
 {
    "DataDir":"/var/lib/pgpro/sdm-14/data",
    "StolonBinPath":"/opt/pgpro/sdm-14/bin",
    "PgBinPath":"/opt/pgpro/sdm-14/bin",
    "PGsInitialPort": 5432,
    "ProxiesInitialPort": 5442,
    "Repfactor": 1,
    "MonitorsNum": 1
 },

"ClusterSpec": // Шаблон конфигурации Stolon
{
"PgSuAuthMethod":"trust",
"PgSuUsername":"postgres",
"PgReplUsername":"repluser",
"PgReplAuthMethod":"trust",
"UseProxy":false,
"StolonSpec":{
     "synchronousReplication": false,
     "minSynchronousStandbys": 1,
     "maxSynchronousStandbys": 0,
     "pgHBA":[
          "local all all trust",
          "host all all 0.0.0.0/0 trust",
          "host replication postgres 0.0.0.0/0 trust",
          "host replication postgres ::0/0 trust",
          "host all repluser 0.0.0.0/0 trust"
       ],
```

```
"pgParameters":
{
    "shared_buffers": "512MB",
    "work_mem": "32MB",
    "autovacuum_analyze_scale_factor": "0.05",
    "autovacuum_analyze_threshold": "100",
    "autovacuum_max_workers": "1",
    "autovacuum_vacuum_scale_factor": "0.02",
    "autovacuum_vacuum_threshold": "1000",
    "log_autovacuum_min_duration": "0",
    "log_checkpoints": "true",
    "log_statement":"none",
    "log_line_prefix":"%m [%r][%p]",
    "log_min_messages":"INFO",
    "wal_level":"logical",
    "wal_compression": "on",
    "shared_preload_libraries":"postgres_fdw,shardman",
    "max_prepared_transactions":"200",
    "default_transaction_isolation":"repeatable read",
    "track_global_snapshots":"on",
    "global_snapshot_defer_time":"20",
    "postgres_fdw.use_global_snapshots":"on",
    "postgres_fdw.use_repeatable_read":"on",
    "enable_partitionwise_aggregate":"on",
    "enable_partitionwise_join":"on",
    "max_worker_processes":"16",
    "max_logical_replication_workers": "9"
  }
```

Управление кластером

- shardmanctl init -f config.json
- shardmanctl nodes add/rm -n node1,…,nodeN
- shardmanctl backup, probackup, cleanup
- shardmanctl rebalance, update, getconnstr
- shardman-loader [OPTIONS] ... <connstr> <table_name>

Узлы кластера



Репликационная группа

мастер и реплика

мастер и несколько реплик.

Звено

несколько узлов на которых размещаются репликационные группы.

Размещение репликационных групп и количество узлов в звеньях регулируется параметром **repfactor**.

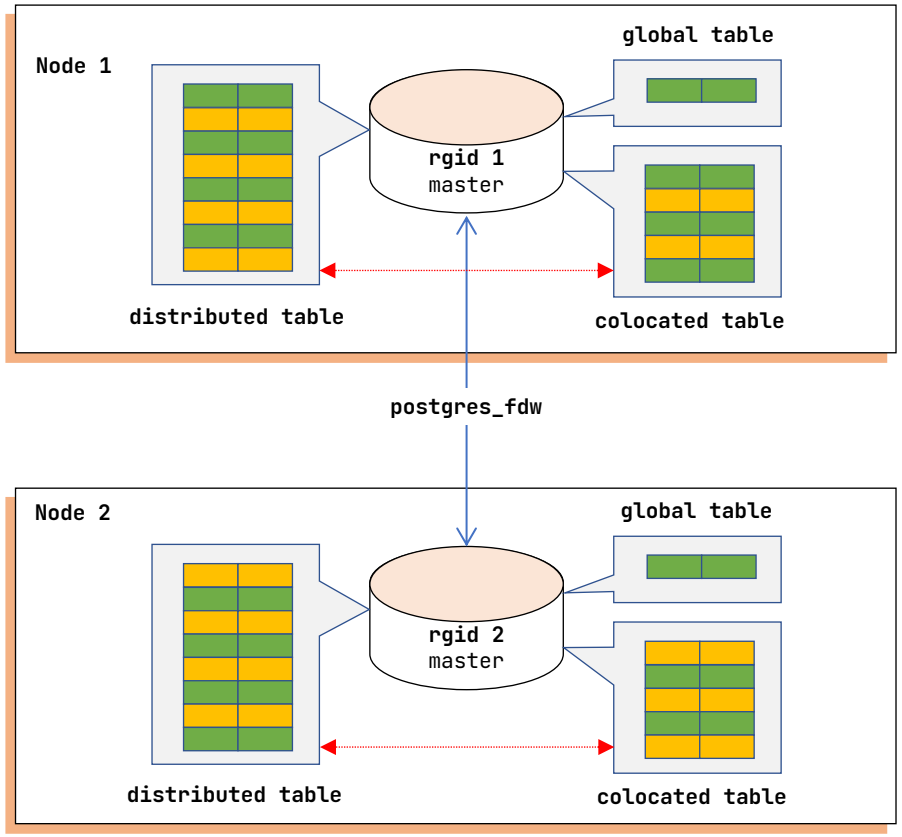
Например, если repfactor=1 (как на слайде), то звено представляет собой два узла, на которых будет две репликационные группы.

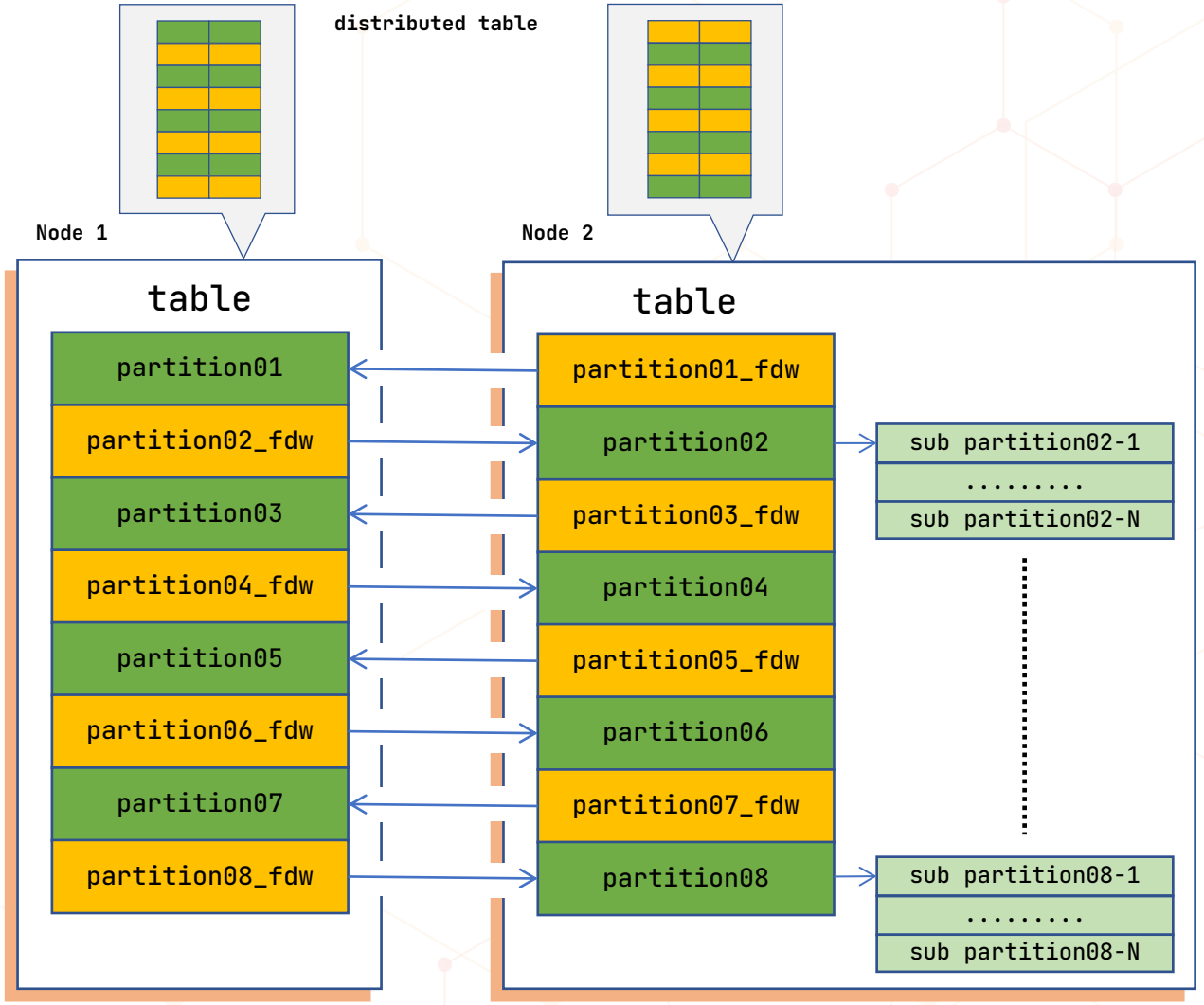
Общее правило: минимальное количество узлов в кластере должно быть кратным параметру repfactor+1.
repfactor=1 - минимум 2 узла
repfactor=2 - минимум 3 узла и т.д.

Шард

- Содержит часть данных таблиц *(следующий слайд)*
- Отказоустойчивый кластер с синхронной физической репликацией
- Состоит из набора секций – микрошардов, находятся физически на одном узле кластера
- Число секций фиксируется при создании таблицы
- Репликационные группы размещаются на нескольких узлах (кросс-репликация)

Секционирование





Распределенные таблицы

Распределенная таблица (distributed table)

- доступна на запись на всех узлах кластера
- секционированная таблица, состоящая из N секций
- количество секций – постоянная величина
- секции таблицы находятся на всех узлах кластера
- секции связаны между собой через *postgres_fdw*
- в случае выхода одного или нескольких узлов кластера из строя данные не будут потеряны.

Соразмещенная таблица (colocated table)

- секционированная таблица, состоящая из N секций
- секции размещены с секциями распределенной таблицы

Глобальная таблица (global table)

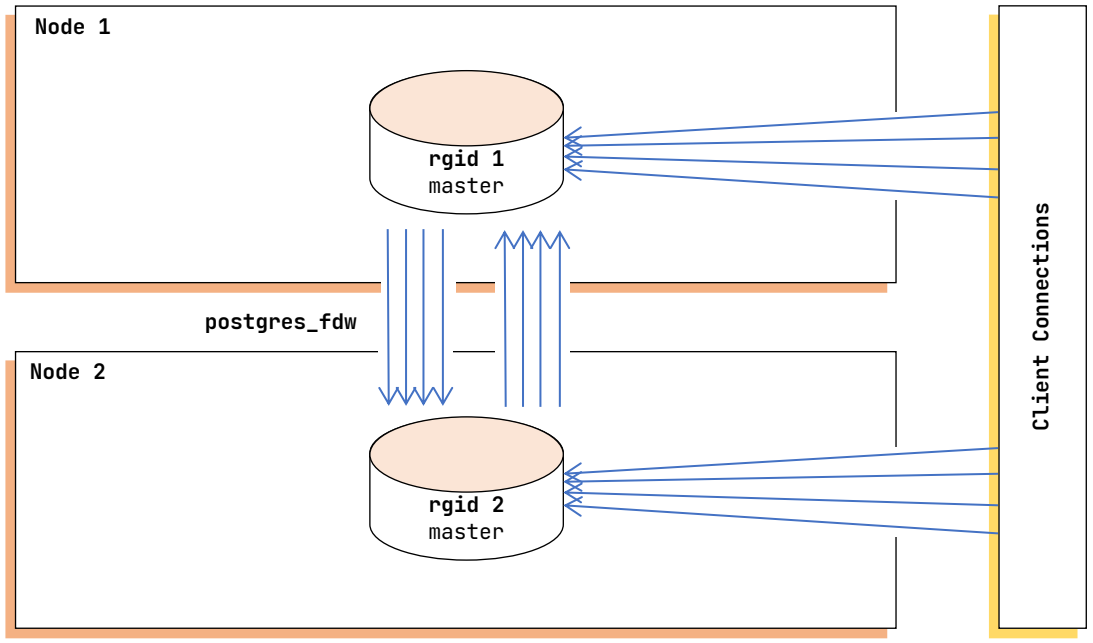
- таблица, размещенная на всех узлах кластера
- содержит идентичные данные на всех шардах

Распределенные таблицы

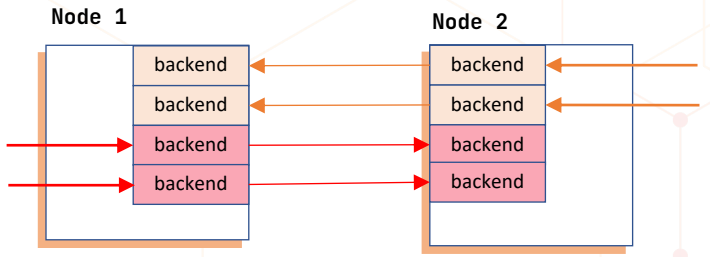
PostgresPro

```
CREATE TABLE town (
  id int primary key,
  name text
)
WITH (global);

CREATE TABLE orgs (
  id int primary key,
  town_id int not null, -- reference town(id)
  name text
)
WITH (distributed_by = 'id', num_parts = 4);

CREATE TABLE employees (
  id int not null,
  orgs_id int not null, -- reference orgs(id)
  name text,
  primary key (orgs_id, id)
)
WITH (distributed_by = ‘orgs_id', colocate_with = 'orgs', num_parts = 4);
```

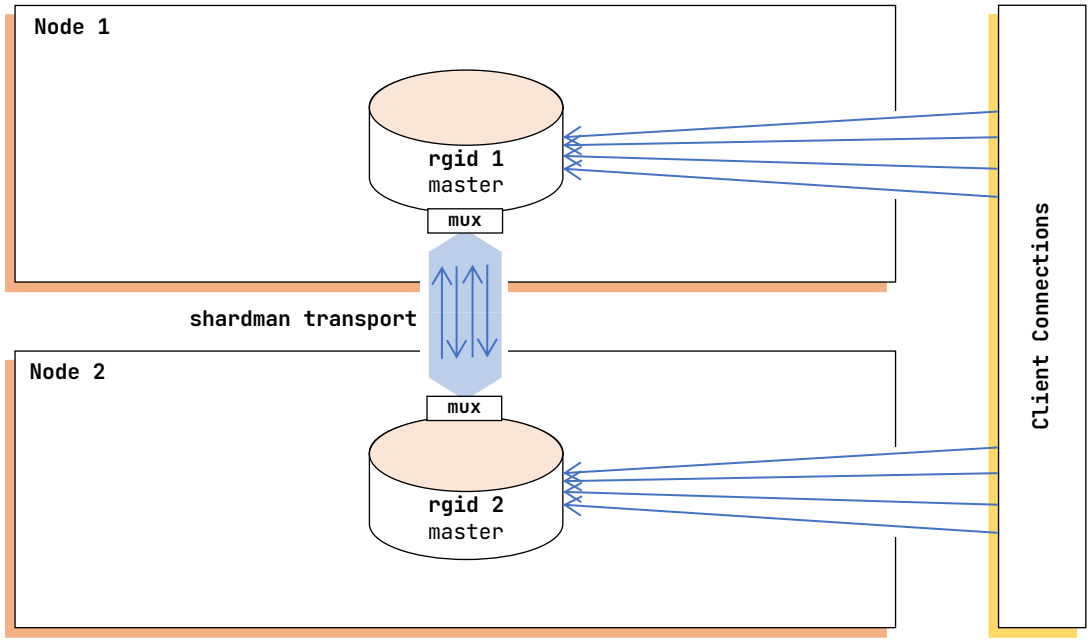
Транспорт

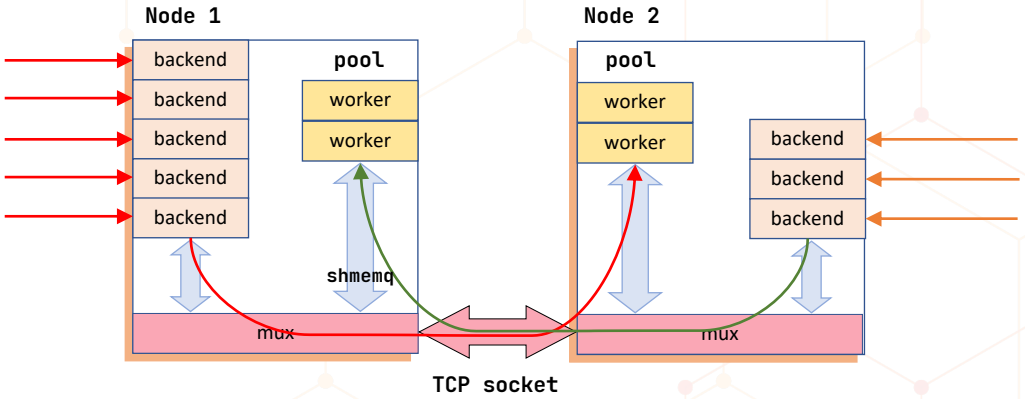


Количество подключений и процессов в кластере – m x N
m – количество подключений,
N – количество узлов



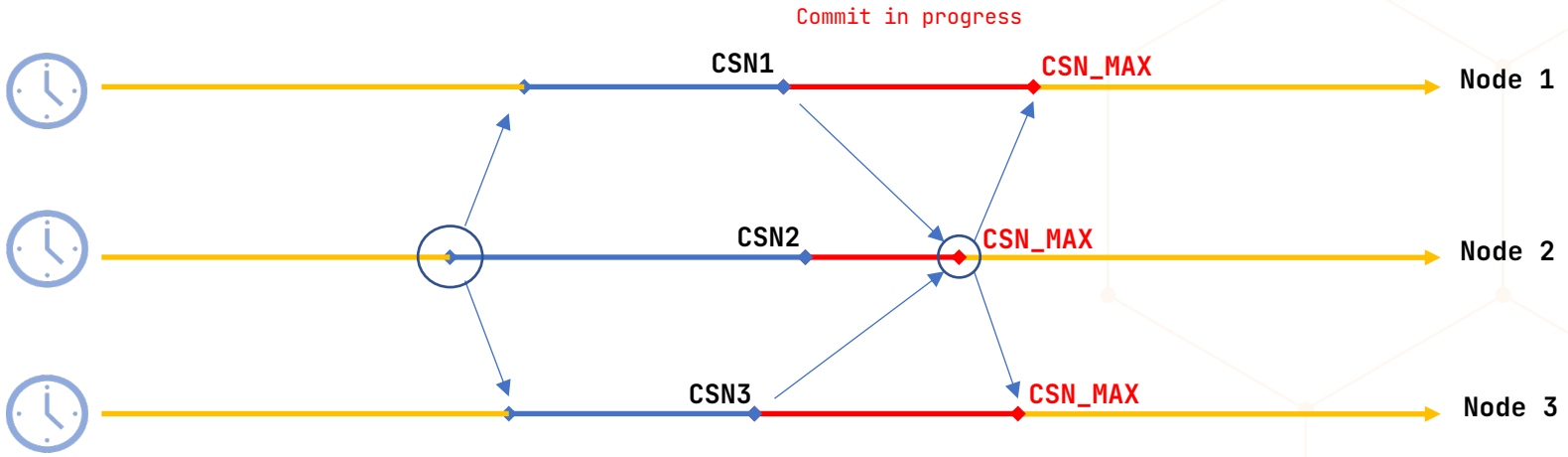
Транспорт





Количество подключений и процессов в кластере – m+(N x w)
m – количество подключений,
N – количество узлов,
w – количество воркеров

CSN - commit sequence number

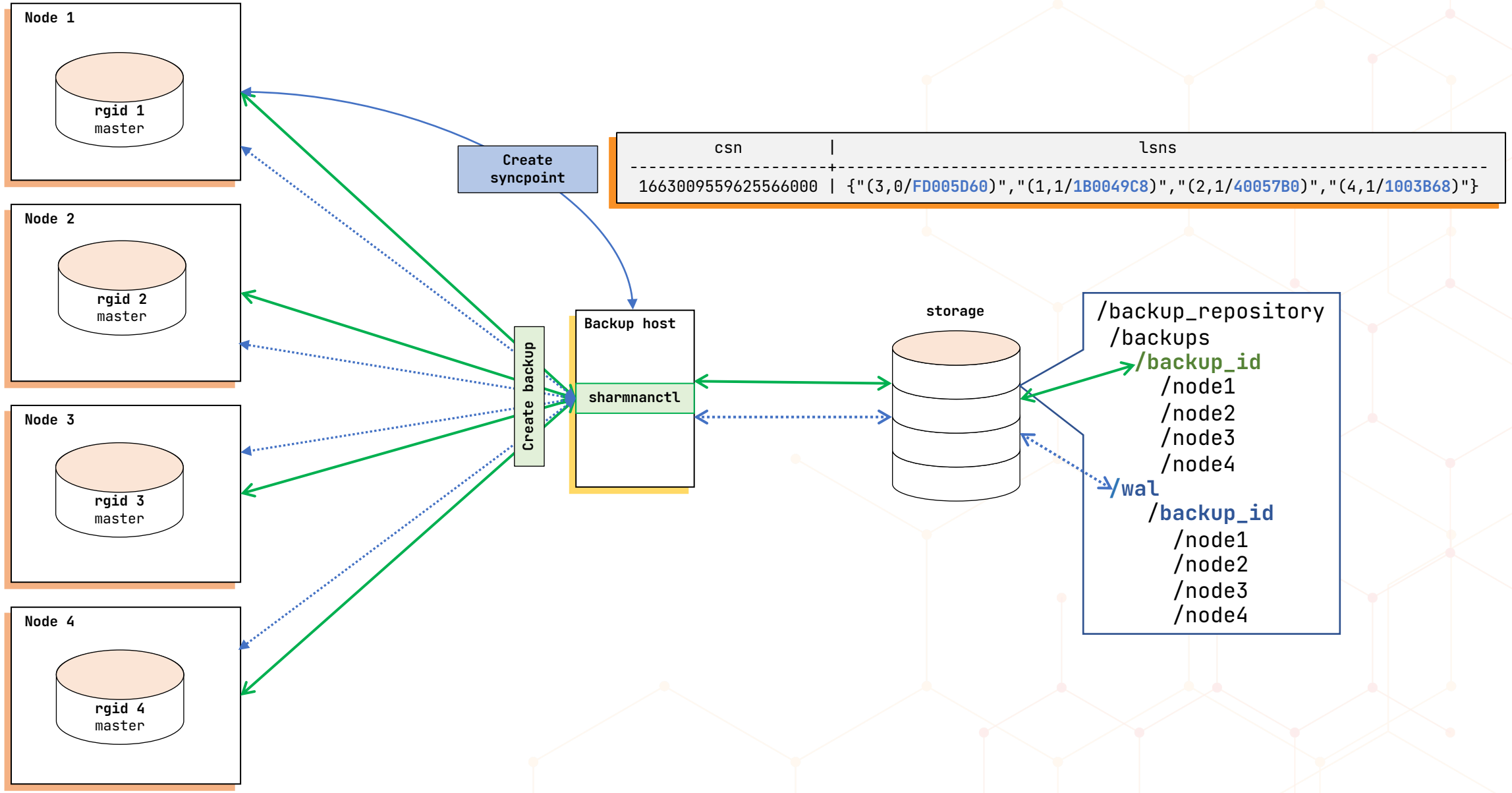


CSN_MAX = MAX(CSN1,CSN2,CSN3)

CSN - время
CSN(LSN_NODE1,LSN_NODE2,LSN_NODE3)

```
# CALL shardman.create_syncpoint(NULL, NULL);

         csn          |                                  lsns
----------------------+-------------------------------------------------------------
 16630095596255660000 | {"(3,0/FD005D60)","(1,1/1B0049C8)","(2,1/40057B0)","(4,1/1003B68)"}
```

Резервное копирование и восстановление



Пример

```
create table doc (
 id uuid primary key,
 added_at timestamp not null,
 author_id uuid references author(id),
 content text not null,
 img bytea,
 status_id uuid references rstatus(id)
);
```

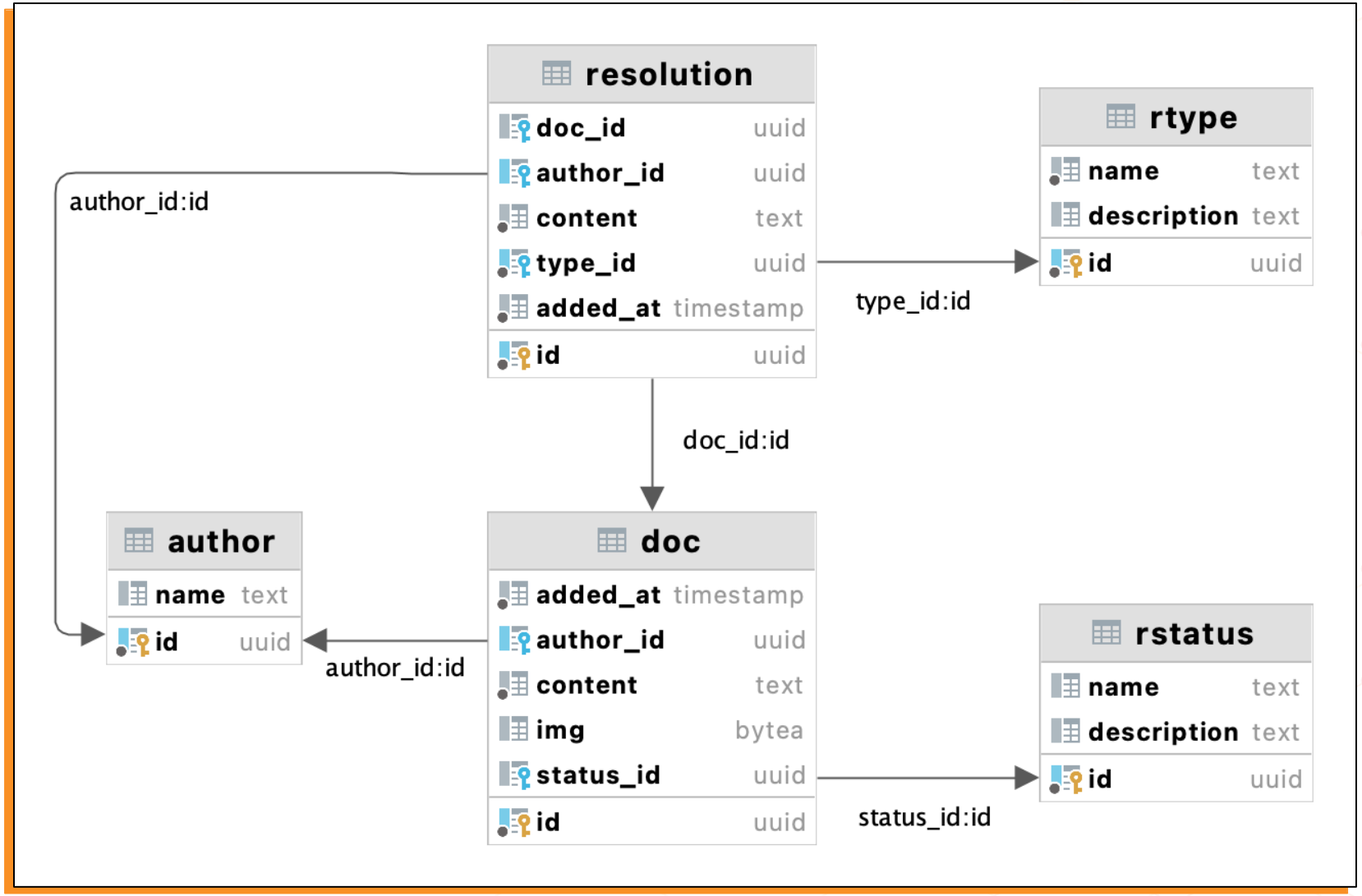
```
create table resolution (
  id uuid primary key,
  doc_id uuid references doc(id),
  author_id uuid references author(id),
  content text not null,
  type_id uuid not null references rtype(id),
  added_at timestamp not null
);
```

```
create table author (
  id uuid primary key,
  name text
);
```

```
create table rstatus (
 id uuid primary key,
 name text,
 description text
);
```

```
create table rtype (
 id uuid primary key,
 name  text not null,
 description text
);
```

Исходная схема



Условия для OLTP

PostgresPro

Требования

- Разбить данные и запросы равномерно по шардам
- Связанные данные локализовать в одном шарде
- Копию справочников разместить на каждом шарде

Преобразование схемы:

- Первичные ключи распределенных таблиц и внешние ключи между двумя распределенными таблицами должны включать колонку объекта распределения
- Справочники сделать глобальными таблицами
- В запросах добавить фильтр по объекту распределения

Схема для шардирования

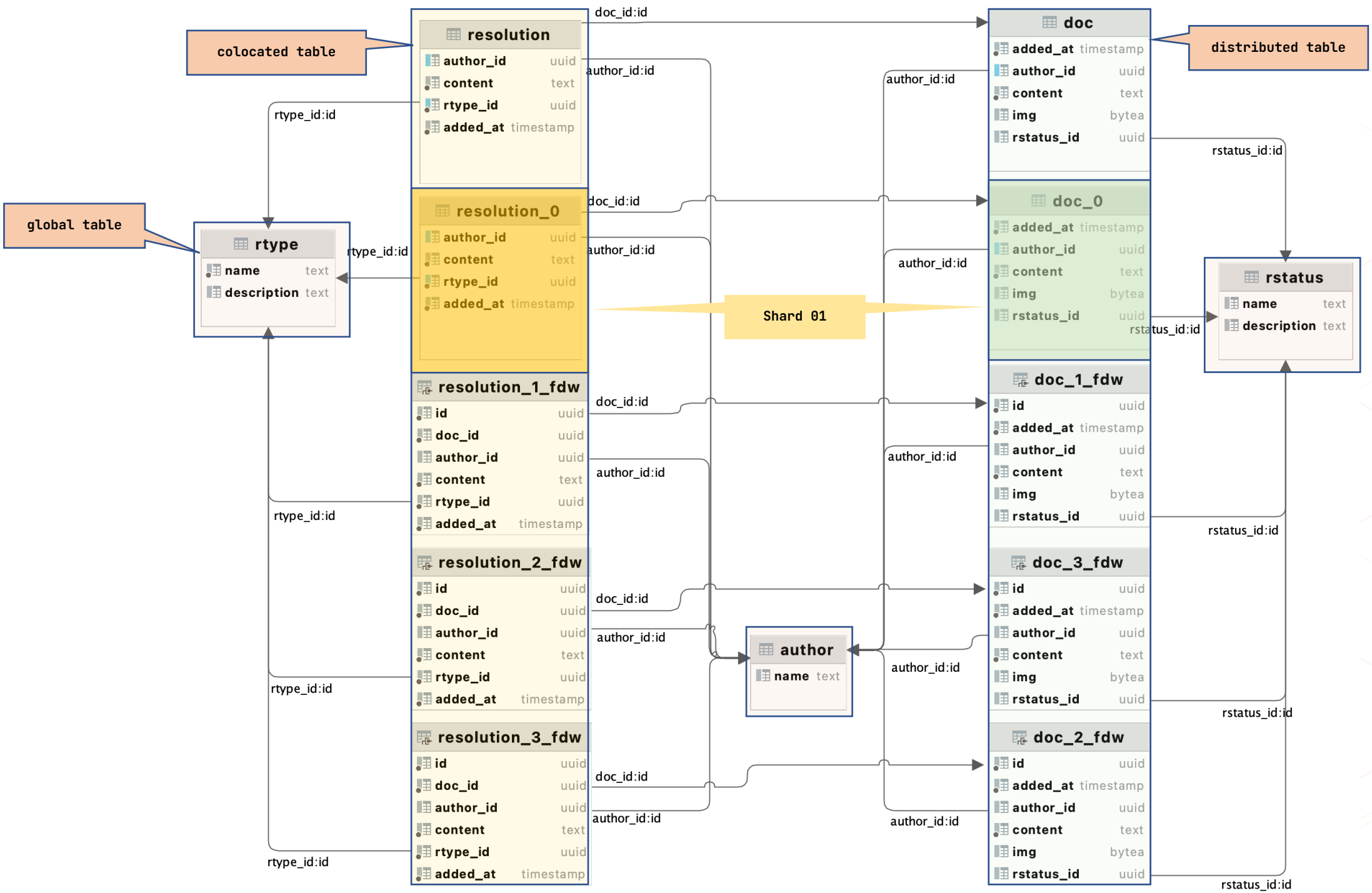
```
create table doc (
 id uuid primary key,
 added_at timestamp not null,
 author_id uuid,
 content text not null,
 img bytea,
 rstatus_id uuid
) with (
distributed_by = 'id', num_parts = 4
);
```

```
create table resolution (
 id uuid,
 doc_id uuid,
 author_id uuid,
 content text not null,
 rtype_id uuid not null,
 added_at timestamp not null,
 primary key (doc_id, id)
) with (
distributed_by = 'doc_id', num_parts = 4, colocate_with = 'doc'
);
```

```
create table author (
 id uuid primary key,
 name text
 )
with (global);
```

```
create table rstatus (
 id uuid primary key,
 name text,
 description text
 )
with (global);
```

```
create table rtype (
 id uuid primary key,
 name text not null,
 description text
 )
with (global);
```


colocated table
distributed table
global table
Shard 01
resolution
author_id uuid
content text
rtype_id uuid
added_at timestamp
resolution_0
author_id uuid
content text
rtype_id uuid
added_at timestamp
resolution_1_fdw
id uuid
doc_id uuid
author_id uuid
content text
rtype_id uuid
added_at timestamp
resolution_2_fdw
id uuid
doc_id uuid
author_id uuid
content text
rtype_id uuid
added_at timestamp
resolution_3_fdw
id uuid
doc_id uuid
author_id uuid
content text
rtype_id uuid
added_at timestamp
rtype
name text
description text
author
name text
doc
added_at timestamp
author_id uuid
content text
img bytea
rstatus_id uuid
doc_0
added_at timestamp
author_id uuid
content text
img bytea
rstatus_id uuid
doc_1_fdw
id uuid
added_at timestamp
author_id uuid
content text
img bytea
rstatus_id uuid
doc_3_fdw
id uuid
added_at timestamp
author_id uuid
content text
img bytea
rstatus_id uuid
doc_2_fdw
id uuid
added_at timestamp
author_id uuid
content text
img bytea
rstatus_id uuid
rstatus
name text
description text
doc_id:id
author_id:id
rtype_id:id
rstatus_id:id

Запросы

```
SELECT d.*, r.id
FROM doc        d
JOIN resolution r
ON    d.id = r.doc_id
WHERE d.author_id = 'c7b502a5-adaf-4565-bb1d-988623646cdb'
AND   d.id        = '000900c2-22fa-4d6b-9f3f-476eb1b8b75f';
```

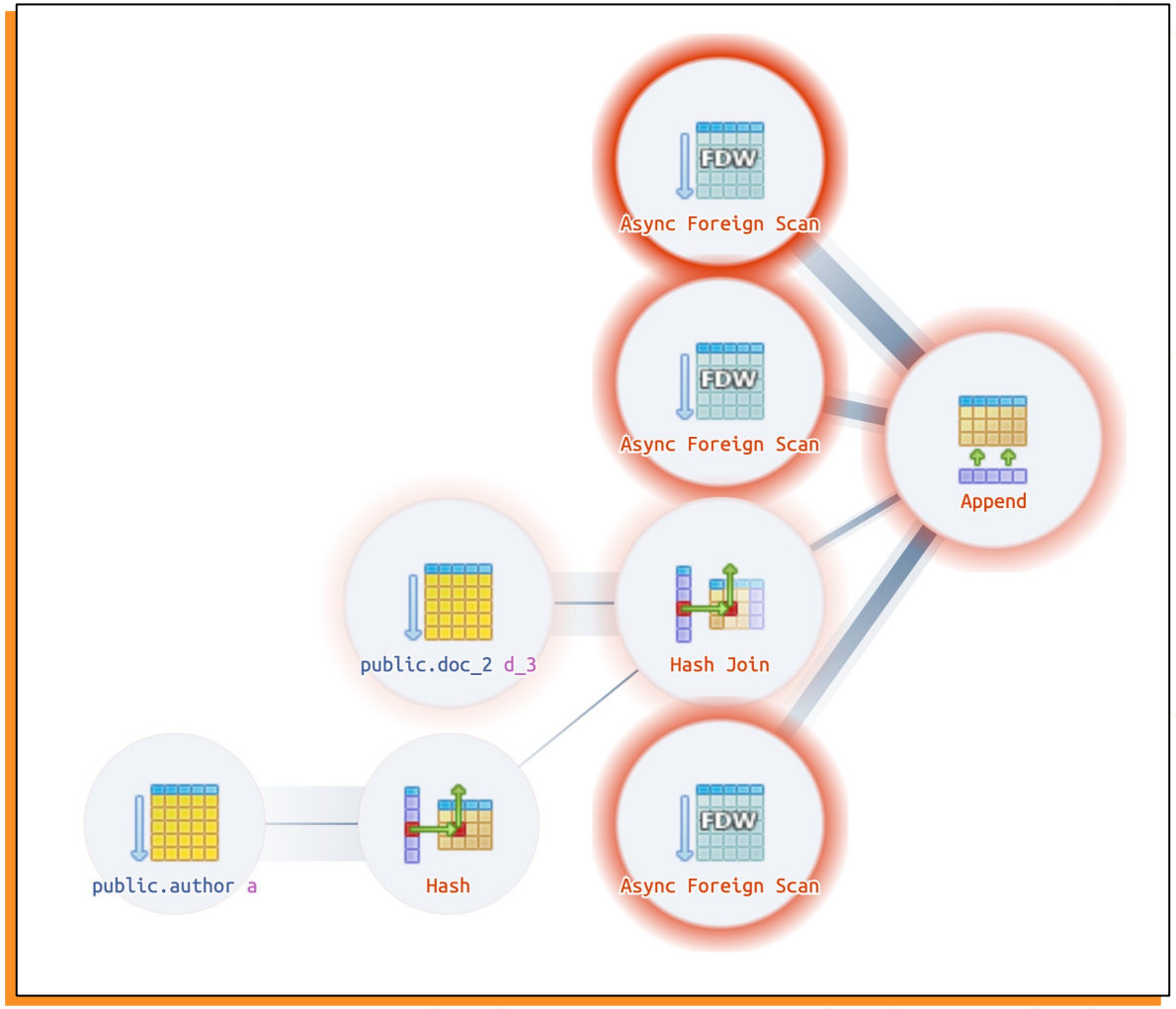
```
Foreign Scan  (cost=100.00..245.52 rows=1 width=106) (actual time=0.759..0.760 rows=2 loops=1)
  Output: d.id, d.added_at, d.author_id, d.content, d.img, d.rstatus_id, r.id
  Relations: (public.doc_3_fdw d) INNER JOIN (public.resolution_3_fdw r)
  Remote SQL: SELECT r4.id, r4.added_at, r4.author_id, r4.content, r4.img, r4.rstatus_id, r5.id
              FROM (public.doc_3 r4 INNER JOIN public.resolution_3 r5
              ON  (((r5.doc_id = '000900c2-22fa-4d6b-9f3f-476eb1b8b75f'))
              AND ((r4.id = '000900c2-22fa-4d6b-9f3f-476eb1b8b75f'))
              AND ((r4.author_id = 'c7b502a5-adaf-4565-bb1d-988623646cdb'))))
  Network: FDW bytes sent=389 received=498
```

Запросы

```
SELECT d.*, a.name
FROM doc d
JOIN author a
ON a.id = d.author_id;
```

```
Append  (cost=1.99..161.08 rows=10000 width=48) (actual time=5.891..20.909 rows=10000 loops=1)
  Network: FDW bytes sent=606 received=332581
  ->  Async Foreign Scan  (cost=1.99..2.00 rows=2520 width=48) (actual time=6.155..6.611 rows=2520 loops=1)
        Output: d_1.id, a.name
        Relations: (public.author a) INNER JOIN (public.doc_0_fdw d_1)
        Remote SQL: SELECT r4.id, r2.name FROM (public.author r2 INNER JOIN public.doc_0 r4 ON (((r4.author_id = r2.id))))
        Network: FDW bytes sent=202 received=112319
  ->  Async Foreign Scan  (cost=1.99..2.00 rows=2506 width=48) (actual time=4.276..4.677 rows=2506 loops=1)
        Output: d_2.id, a.name
        Relations: (public.author a) INNER JOIN (public.doc_1_fdw d_2)
        Remote SQL: SELECT r5.id, r2.name FROM (public.author r2 INNER JOIN public.doc_1 r5 ON (((r5.author_id = r2.id))))
        Network: FDW bytes sent=202 received=111632
  ->  Hash Join  (cost=34.08..105.09 rows=2534 width=48) (actual time=0.081..1.937 rows=2534 loops=1)
        Output: d_3.id, a.name
        Inner Unique: true
        Hash Cond: (d_3.author_id = a.id)
        ->  Seq Scan on public.doc_2 d_3  (cost=0.00..64.34 rows=2534 width=32) (actual time=0.024..0.785 rows=2534 loops=1)
              Output: d_3.id, d_3.author_id
        ->  Hash  (cost=20.70..20.70 rows=1070 width=48) (actual time=0.039..0.040 rows=40 loops=1)
              Output: a.name, a.id
              Buckets: 2048  Batches: 1  Memory Usage: 19kB
              ->  Seq Scan on public.author a  (cost=0.00..20.70 rows=1070 width=48) (actual time=0.010..0.019 rows=40 loops=1)
                    Output: a.name, a.id
  ->  Async Foreign Scan  (cost=1.99..2.00 rows=2440 width=48) (actual time=4.177..4.631 rows=2440 loops=1)
        Output: d_4.id, a.name
        Relations: (public.author a) INNER JOIN (public.doc_3_fdw d_4)
        Remote SQL: SELECT r7.id, r2.name FROM (public.author r2 INNER JOIN public.doc_3 r7 ON (((r7.author_id = r2.id))))
        Network: FDW bytes sent=202 received=108630
```

Запросы

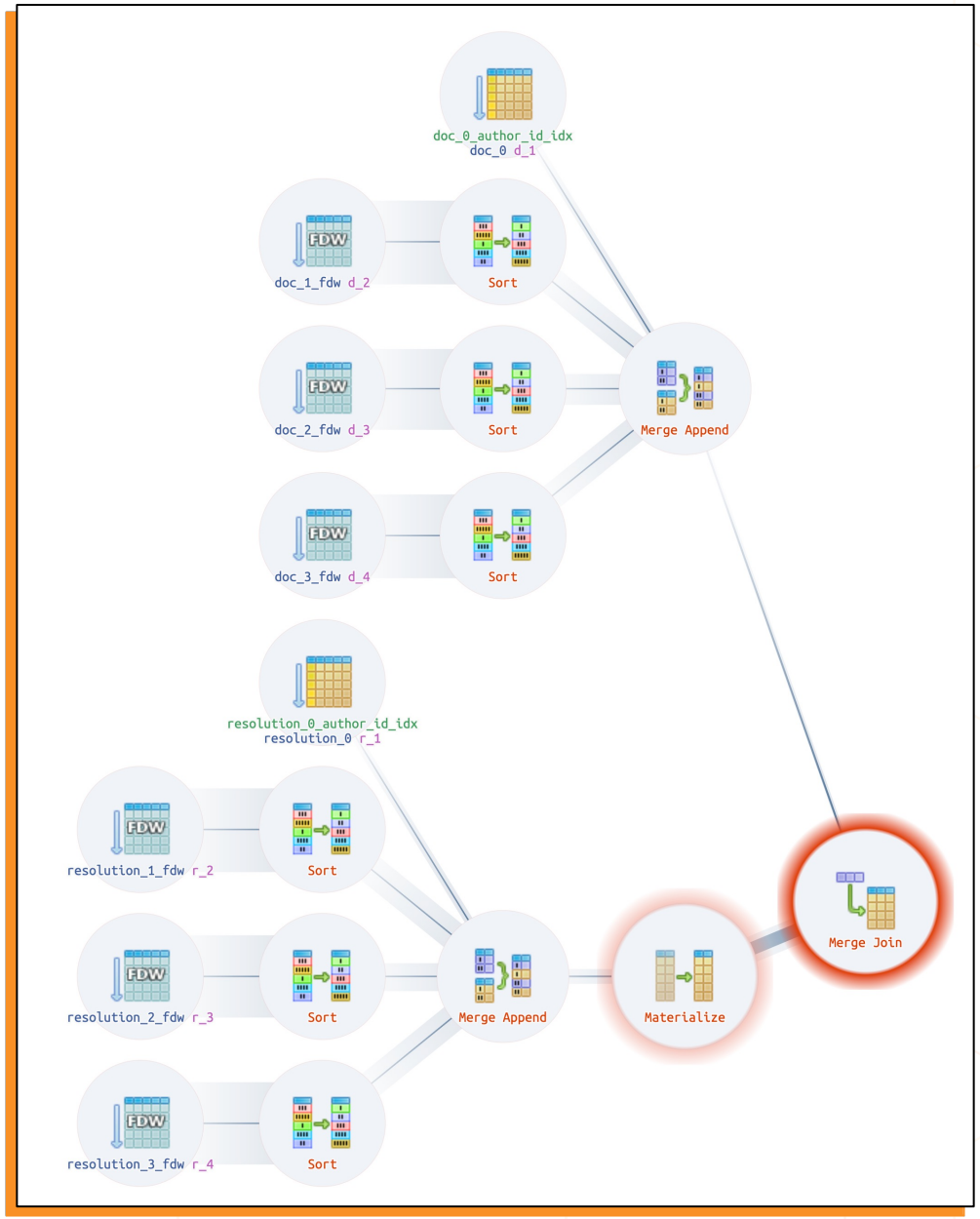

FDW
Async Foreign Scan
FDW
Async Foreign Scan
Append
public.doc_2 d_3
Hash Join
public.author a
Hash
FDW
Async Foreign Scan

Запросы

```
SELECT d.*, r.id FROM doc d JOIN resolution r ON d.author_id = r.author_id;
```

```
Join
  ->  Merge Append
        ->  Scan Table doc_0
        ->  Process
              ->  Scan Table Foreign doc_1_fdw
        ->  Process
              ->  Scan Table Foreign doc_2_fdw
        ->  Process
              ->  Scan Table Foreign doc_3_fdw
  ->  Materialize
      -> Merge Append
        ->  Scan Table resolution_0
        ->  Process
              ->  Scan Table Foreign resolution_1_fdw
        ->  Process
              ->  Scan Table Foreign resolution_2_fdw
        ->  Process
              ->  Scan Table Foreign resolution_3_fdw
```

Запросы

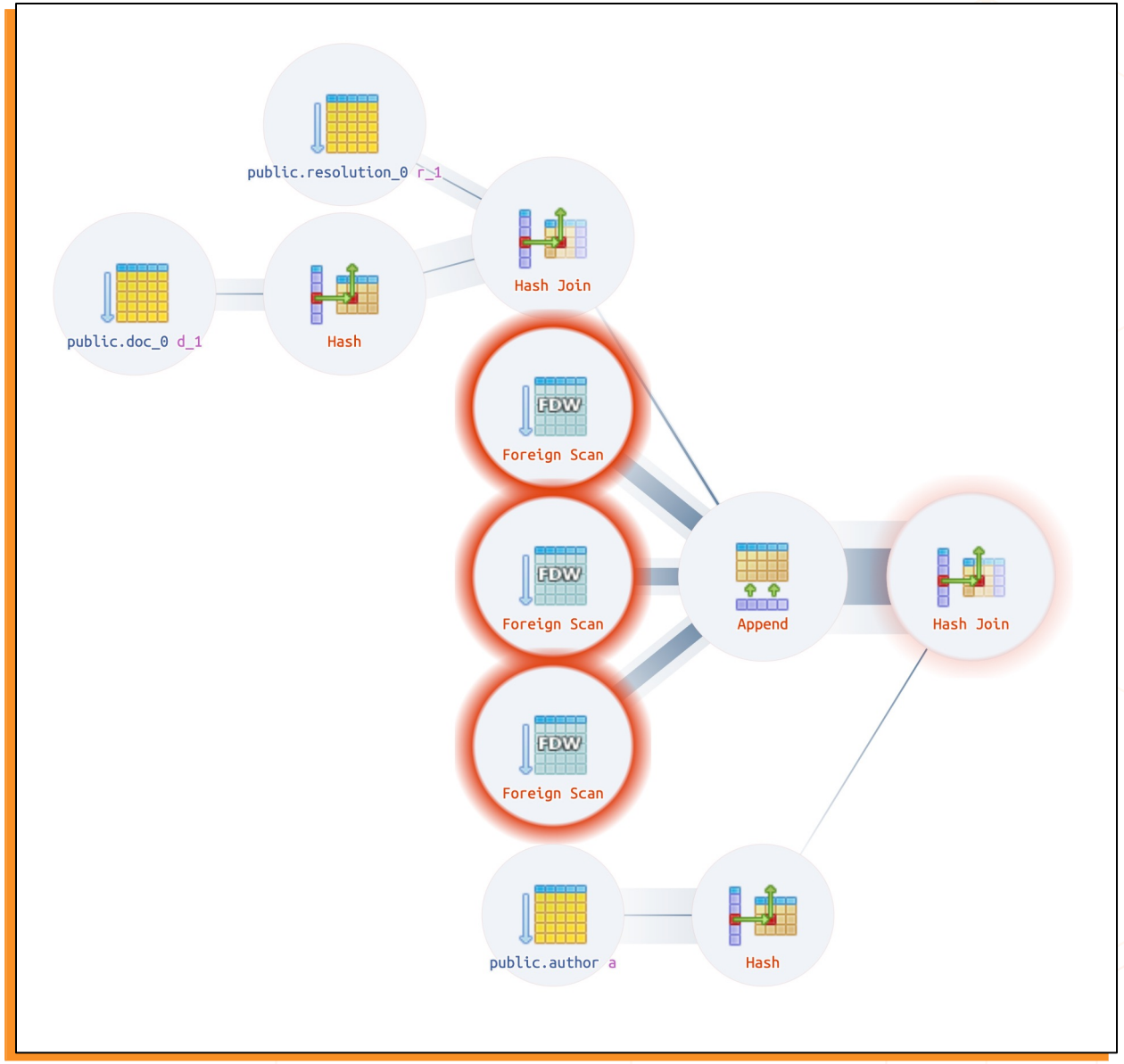

doc_0_author_id_idx
doc_0 d_1
FDW
doc_1_fdw d_2
Sort
FDW
doc_2_fdw d_3
Sort
Merge Append
FDW
doc_3_fdw d_4
Sort
resolution_0_author_id_idx
resolution_0 r_1
FDW
resolution_1_fdw r_2
Sort
FDW
resolution_2_fdw r_3
Sort
Merge Append
Materialize
Merge Join
FDW
resolution_3_fdw r_4
Sort

Запросы

```
SELECT d.*, r.id FROM doc d JOIN resolution r ON d.id = r.doc_id
                            JOIN author     a ON a.id = d.author_id;
```

```
Hash Join
  Output: d.id, d.added_at, d.author_id, d.content, d.img, d.rstatus_id, r.id
  Inner Unique: true
  Hash Cond: (d.author_id = a.id)
  ->  Append
        ->  Hash Join
              Hash Cond: (r_1.doc_id = d_1.id)
              ->  Seq Scan on public.resolution_0 r_1
              ->  Hash
                    Buckets: 2048  Batches: 1  Memory Usage: 166kB
                    ->  Seq Scan on public.doc_0 d_1
        ->  Foreign Scan
              Output: d_2.id, d_2.added_at, d_2.author_id, d_2.content, d_2.img, d_2.rstatus_id, r_2.id
              Relations: (public.resolution_1_fdw r_2) INNER JOIN (public.doc_1_fdw d_2)
              Remote SQL: SELECT r7.id, r7.added_at, r7.author_id, r7.content, r7.img, r7.rstatus_id, r11.id
                          FROM (public.resolution_1 r11 INNER JOIN public.doc_1 r7 ON (((r7.id = r11.doc_id))))
        ->  Foreign Scan
              Relations: (public.resolution_2_fdw r_3) INNER JOIN (public.doc_2_fdw d_3)
        ->  Foreign Scan
              Relations: (public.resolution_3_fdw r_4) INNER JOIN (public.doc_3_fdw d_4)
  ->  Hash
```

Запросы

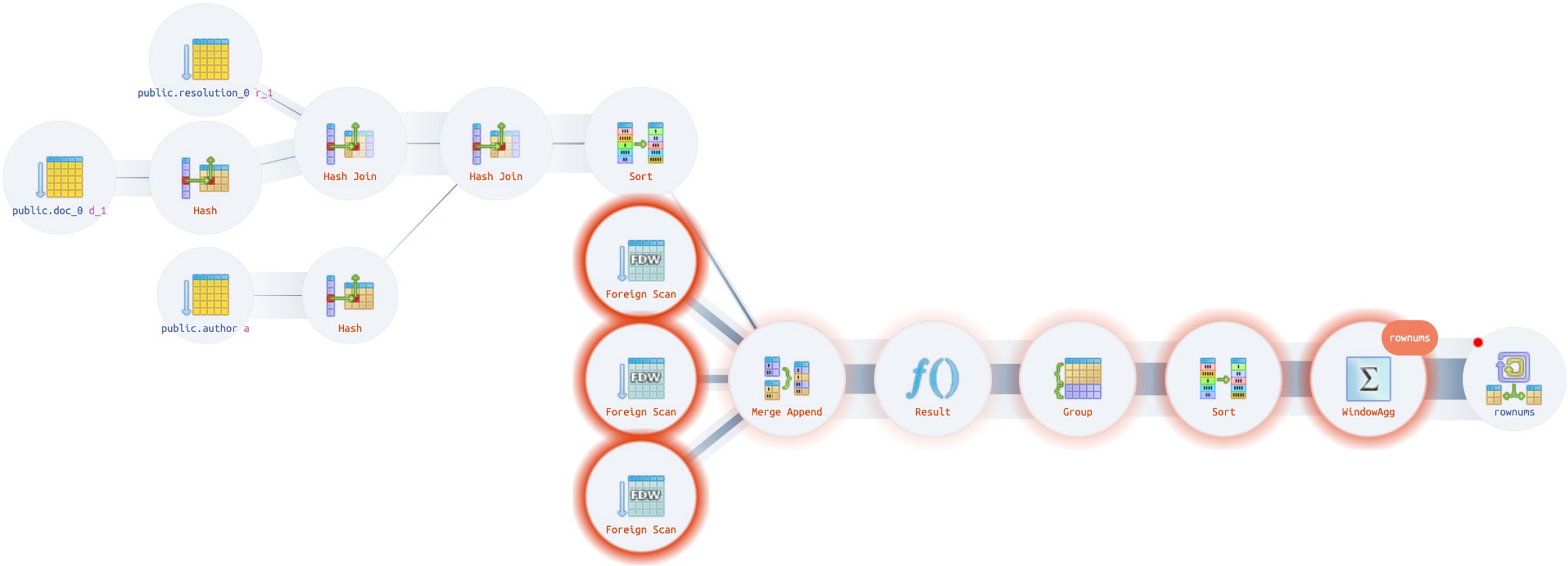

public.resolution_0 r_1
public.doc_0 d_1
Hash
Hash Join
FDW
Foreign Scan
FDW
Foreign Scan
FDW
Foreign Scan
Append
Hash Join
public.author a
Hash

Запросы

```
WITH rownums AS
     (
       SELECT row_number() OVER (partition by d.id order by a.name) AS row,
              r.id AS res,
              d.id AS doc,
              a.name as name
       FROM doc d
       JOIN resolution r ON d.id = r.doc_id
       JOIN author     a ON a.id = d.author_id
       GROUP BY r.id, d.id, a.name
     ) SELECT row, doc, name FROM rownums WHERE row > 3;
```

```
Scan Subquery
 -> WindowAgg
     -> Result
         -> Merge Append
             -> Process
                 -> Join
                     -> Scan Table public.author
                     -> Scan Table public.doc_0
                     -> Scan Table public.resolution_0
             -> Table Foreign
             -> Table Foreign
             -> Table Foreign
```

Запросы

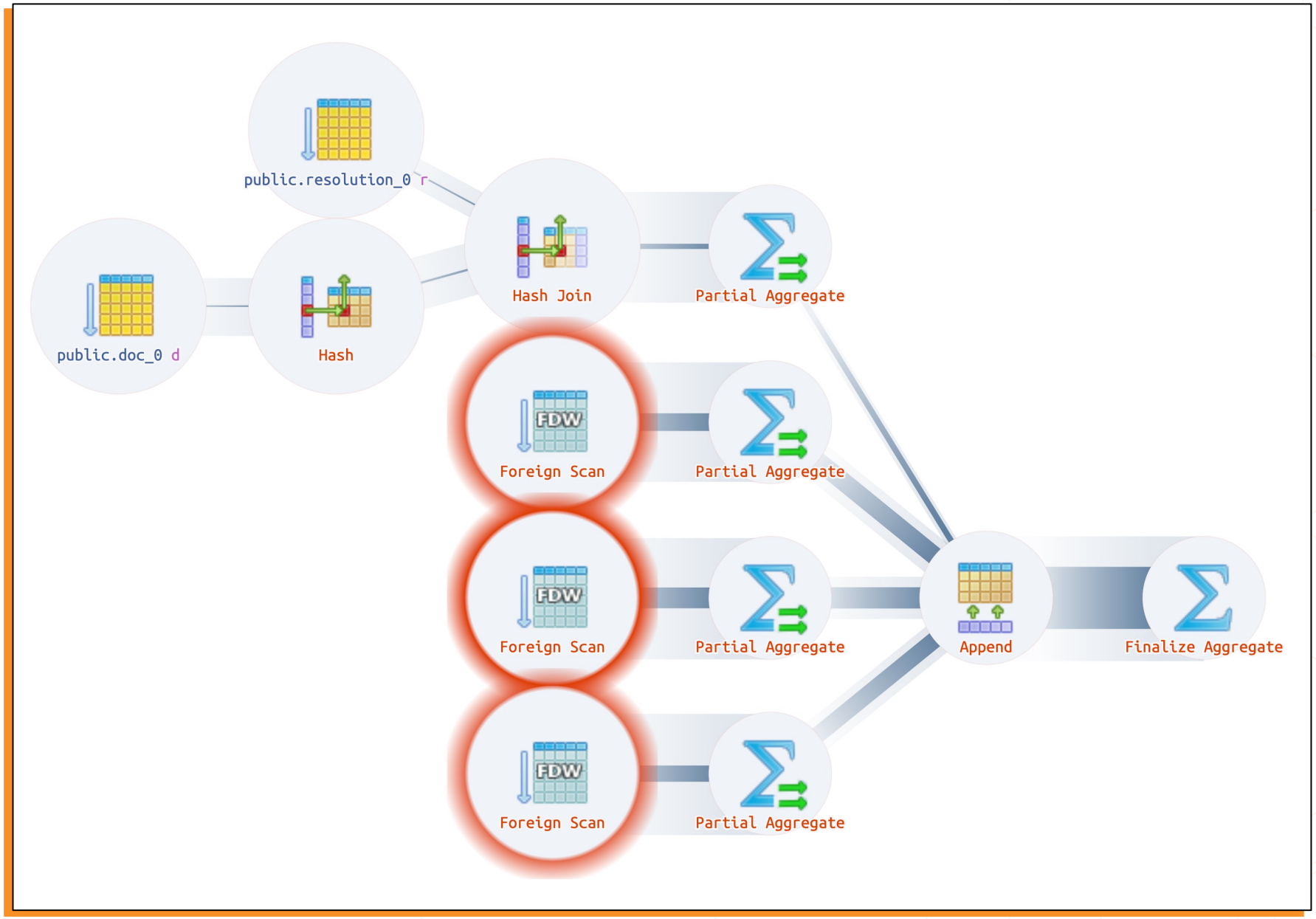

public.resolution_0 r_1
public.doc_0 d_1
Hash
Hash Join
Hash Join
Sort
public.author a
Hash
FDW
Foreign Scan
FDW
Foreign Scan
FDW
Foreign Scan
Merge Append
f()
Result
Group
Sort
rownums
WindowAgg
rownums

Запросы

```
SELECT count(d.id) as cnt FROM doc d JOIN resolution r ON d.id = r.doc_id;
```

```
Finalize Aggregate
 Output: count(d.id)
->  Append
      ->  Partial Aggregate
          Output: PARTIAL count(d.id)
          ->  Hash Join
              Output: d.id
              Hash Cond: (r.doc_id = d.id)
              ->  Seq Scan on public.resolution_0 r
                  Output: r.doc_id
              ->  Hash
                  Output: d.id
                  Buckets: ...
                  ->  Seq Scan on public.doc_0 d
                      Output: d.id
      ->  Partial Aggregate
          Output: PARTIAL count(d_1.id)
      ->  Partial Aggregate
          Output: PARTIAL count(d_2.id)
      ->  Partial Aggregate
          Output: PARTIAL count(d_3.id)
```

Запросы


public.resolution_0 r
public.doc_0 d
Hash
Hash Join
Partial Aggregate
FDW
Foreign Scan
Partial Aggregate
FDW
Foreign Scan
Partial Aggregate
FDW
Foreign Scan
Partial Aggregate
Append
Finalize Aggregate

TPC-B

PostgresPro

Параметры теста/количество узлов	1	2	10	25	50
Без ОУК*	1650	1399	3500	7000	10600
ОУК**	970	730	1720	3430	5600
Адаптированный без ОУК			5700	11800	17500

* Без прокси, без реплик, 50 секций
** С прокси и одной репликой, 50 секций

```
Данные:              pgbench -i -I gv -s 1000 (~14GB )
Тест:                pgbench -n -P 5 -c 10 -j 2 -T 60
Узлы кластера:       2cores/2GB ram, диск 10k IOPS и 125MB/s
Серверы для
создания нагрузки: 8cores/16Gb ram
```

Адаптированный тест TPC-B под шардинг:

- данные, относящиеся к одному branch лежат на одной шарде
- это позволяет создавать не больше одной удаленной транзакции

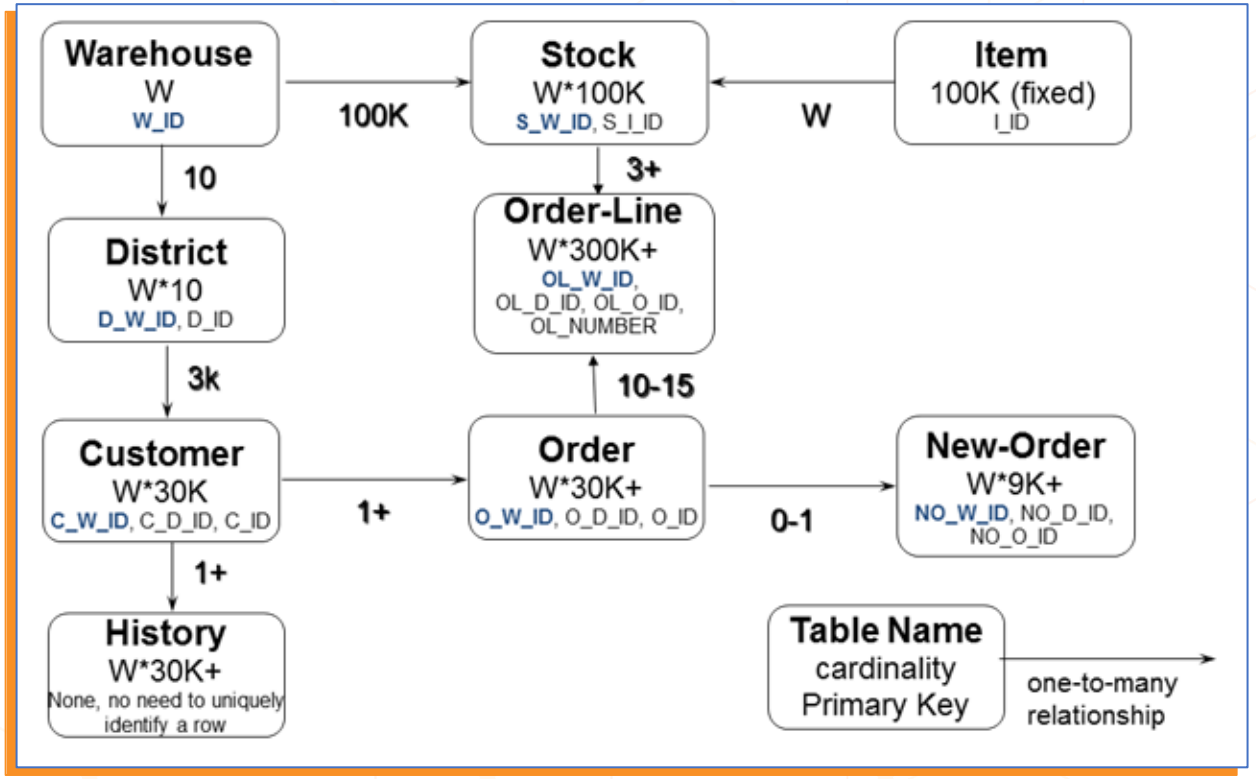
Тесты – TPC-C

Тест TPROC-C основана на спецификации TPC-C: система для выполнения заказов клиентов на поставку продукции компании.

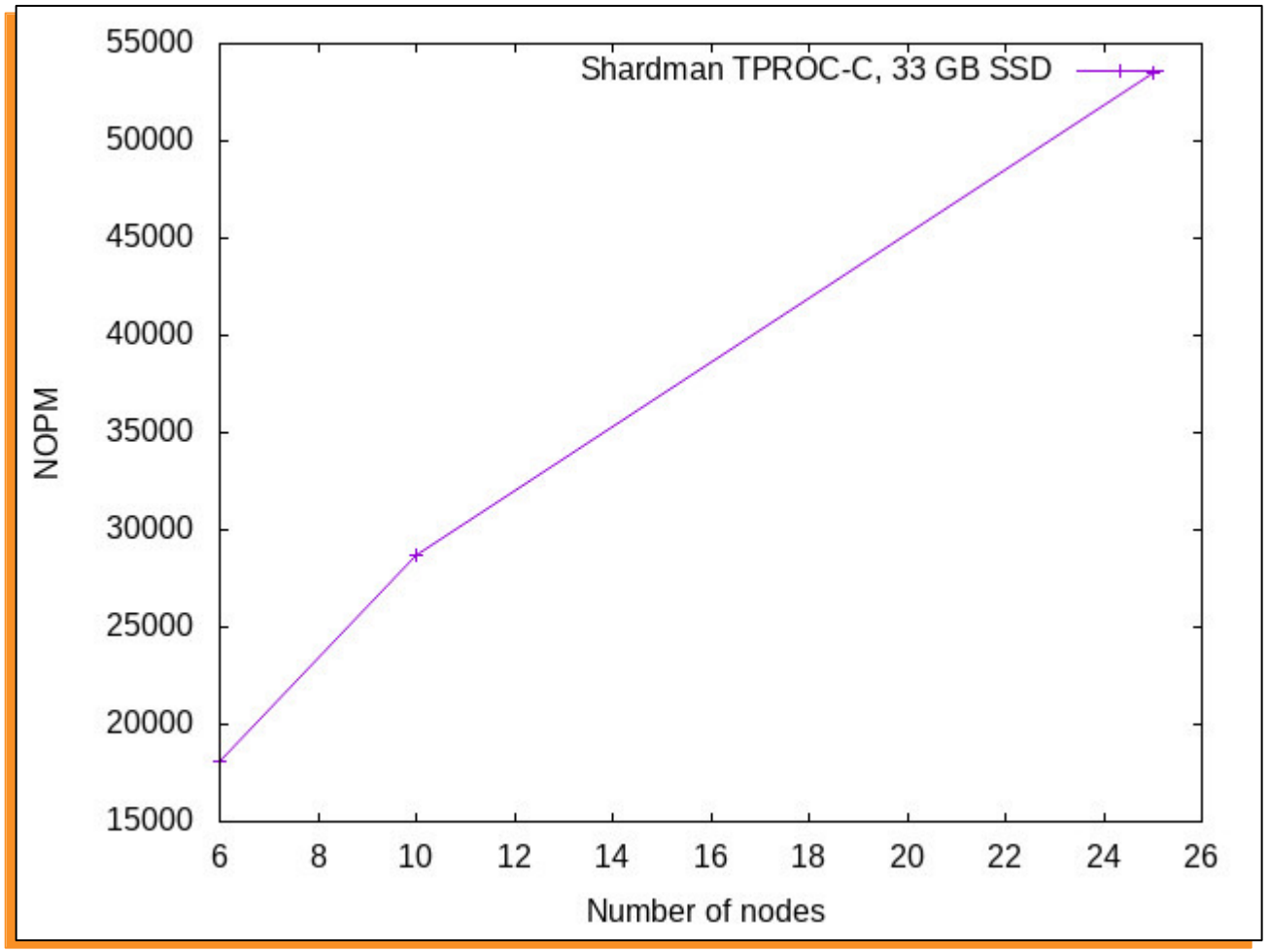
- Компания продает 100 000 наименований товаров и хранит свои запасы на складах
- Каждый склад имеет 10 торговых районов
- Каждый район обслуживает 3000 клиентов
- Клиенты звонят в компанию, операторы принимают заказ
- Каждый заказ содержит определенное количество товаров
- Заказы выполняются с местного склада
- Небольшое количество товаров поставляется с другого склада
- Размер компании не фиксирован, и по мере роста компании могут добавляться склады и районы продаж
- Тестовая схема может быть, как маленькой, так и довольно большой
- Более крупная схема требует более мощной компьютерной системы для обработки возросшего уровня транзакций

Схема TPROC-C: количество строк во всех таблицах, кроме таблицы ITEM, которая является фиксированной, зависит от количества складов.

Новый заказ: получение нового заказа: 45%
Оплата: обновление баланса клиента при платеже: 43%
Доставка: асинхронная доставка заказов: 4%
Статус заказа: получение статуса заказа клиента: 4%
Уровень запасов: состояние запасов склада: 4%



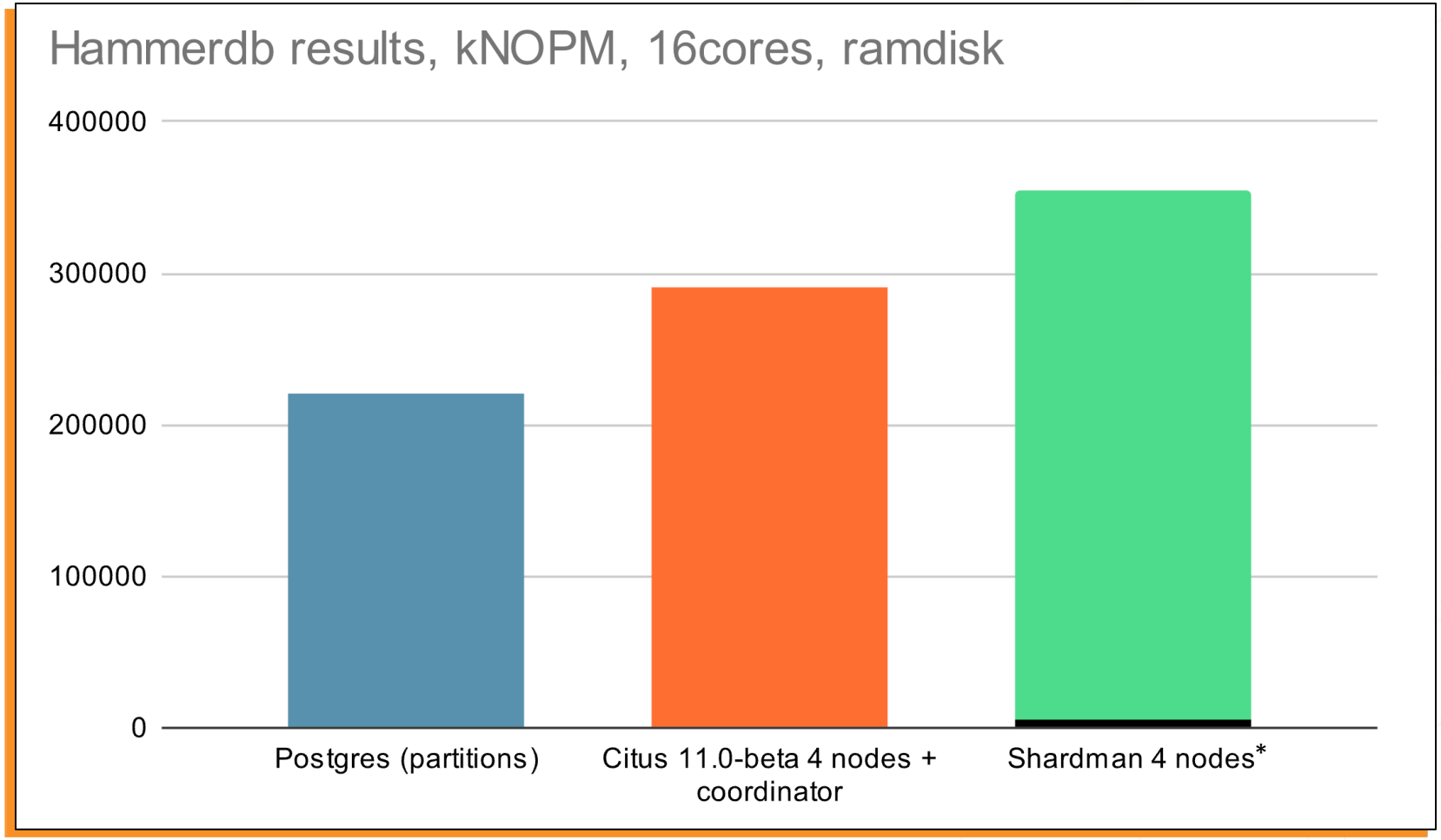
TPC-C



Тестовые сервера – Intel Cascade Lake, 2 vCPU, 8 ГБ RAM, данные располагаются на SSD
Количество warehouses: 1000
Количество разделов таблиц: 100
Объем данных 33Гб

Конфигурация	Производительность
6 узлов, 6 клиентов 4 подключения на клиента	18143 NOPM
10 узлов 10 клиентов 4 подключения	28718 NOPM
25 узлов 25 клиентов 4 подключения	53509 NOPM

TPC-C



* Исходная производительность Postgres FDW отмечена черным.

Ограничения

- Распределение данных между шардами поддерживается только по хэшу
- Все формы ALTER TABLE невозможны, кроме:
  - Изменения владельца таблицы,
  - SET/DROP NOT NULL,
  - ADD/DROP COLUMN, кроме type serial (автоинкремент)
- Нельзя создавать распределенные временные таблицы
- Нельзя изменить количество секций распределенных таблиц
- Внешние ключи не поддерживаются
- Невозможно переименовывать глобальные роли
- Только один тип хранения данных – таблицы PostgreSQL
- Не позволяет использовать реплики для чтения данных
- Транспорт работает только для SELECT
- Транспорт на реплике не поддерживается

Документация, тестовые пакеты

PostgresPro

Ссылки:

- «Распределенные транзакции и путешествия во времени»
  https://youtu.be/DFLEkWi-vRc
- «A New Approach to Sharding for Distributed PostgreSQL»
  https://postgrespro.com/blog/pgsql/5969681
- The Internals of PostgreSQL. Chapter 4.
  Foreign Data Wrappers and Parallel Query
  https://www.interdb.jp/pg/pgsql04.html
- Визуализация плана выполнения запроса
  https://explain.tensor.ru

Документация:

- Postgres FDW
  https://postgrespro.ru/docs/postgresql/14/postgres-fdw
- Shardman
  http://repo.postgrespro.ru/doc/pgprosm/14.4.8/en/html

Тестовая сборка:

- http://repo.postgrespro.ru/pgprosm-14

Докер:

- https://github.com/pkonotopov/shardman-docker

PGMeetup.DBA

С П А С И Б О !

postgrespro.ru