PostgresPro

Стратегия движения к универсальной СУБД (OLTP/OLAP/HTAP)

Марк Ривкин

Начальник отдела технического консалтинга,
Постгрес Про

m.rivkin@postgrespro.ru

Типы систем

- **OLTP - Системы для обработки большого количества коротких транзакций в реальном времени.**
  - Оптимизированы для CRUD-операций (Create, Read, Update, Delete)
  - Высокая доступность и скорость отклика
  - Нормализованные структуры данных
  - ACID-требования (Atomicity, Consistency, Isolation, Durability)
  - Горизонтальное масштабирование через шардинг
  - Репликация для отказоустойчивости

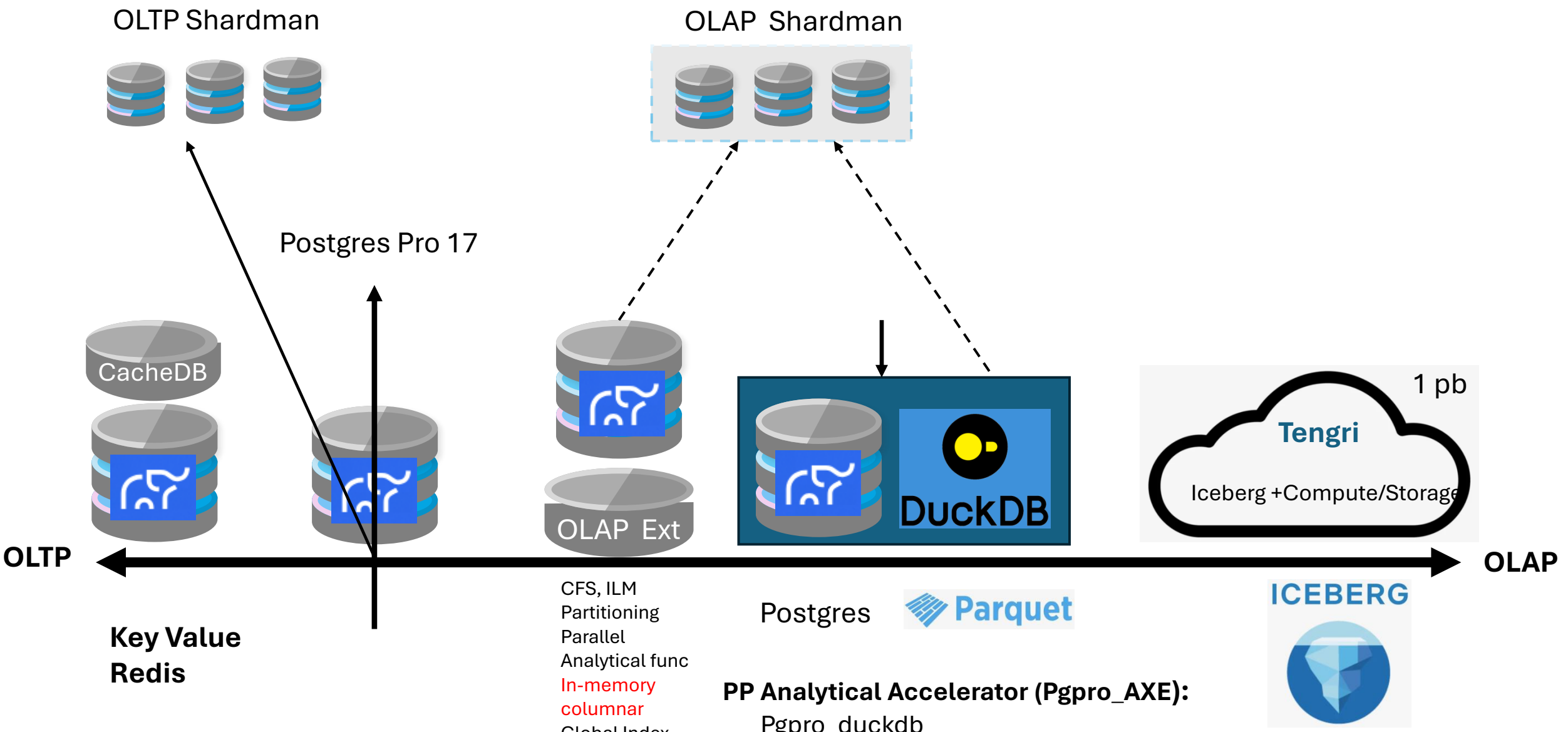
- **Аналитические системы (OLAP) - Системы для сложного анализа больших объемов исторических данных.**
  - Оптимизированы для чтения и агрегации
  - Денормализованные структуры (звезда, снежинка)
  - Пакетная обработка данных
  - Редкие изменения

OLTP, OLAP, HTAP

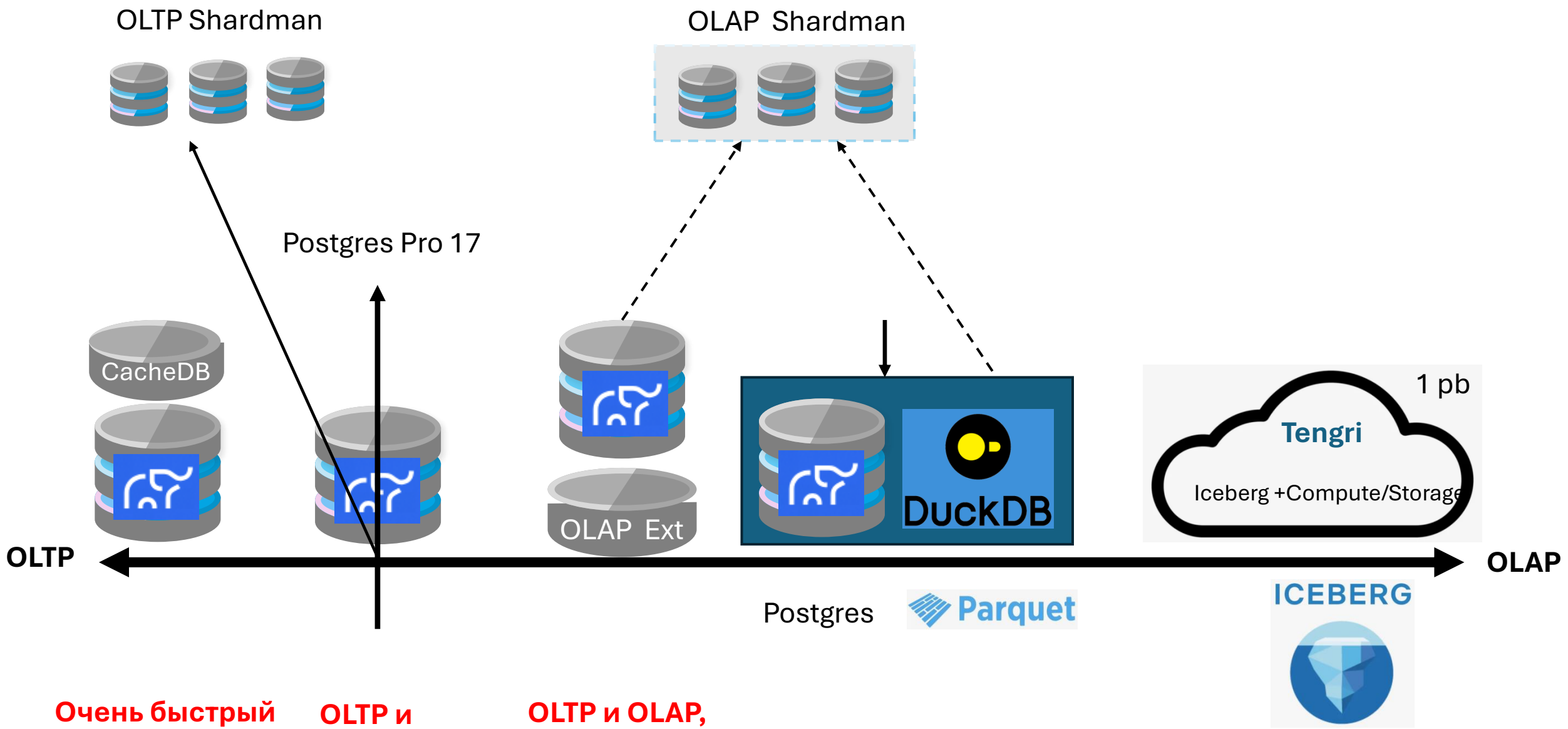
Критерий	OLTP (транзакционные)	OLAP (аналитические)
Цель	Оперативная обработка	Анализ исторических данных
Тип запросов	Короткие, простые	Сложные, агрегирующие
Данные	Актуальные	Исторические (годы)
Пример	Банковская транзакция	Отчет по продажам за 5 лет

В реальной жизни часто видим смесь нагрузок - HTAP

Можно иметь 2 разные СУБД (для OLTP и аналитики, но это дорого и аналитика отстает от OLTP
Можно сделать универсальную/конвергентную СУБД (как Oracle) и работать на свежих данных


OLTP Shardman
OLAP Shardman
Postgres Pro 17
CacheDB
OLAP Ext
DuckDB
1 pb
Tengri
Iceberg +Compute/Storage
OLTP
OLAP
Key Value
Redis
CFS, ILM
Partitioning
Parallel
Analytical func
In-memory
columnar
Global Index
Result cache
Optimizer
iHeap table
Append optimized
table .....
Postgres
Parquet
ICEBERG
PP Analytical Accelerator (Pgpro_AXE):
Pgpro_duckdb
Pgpro_metastore

Позиционирование



Очень быстрый OLTP

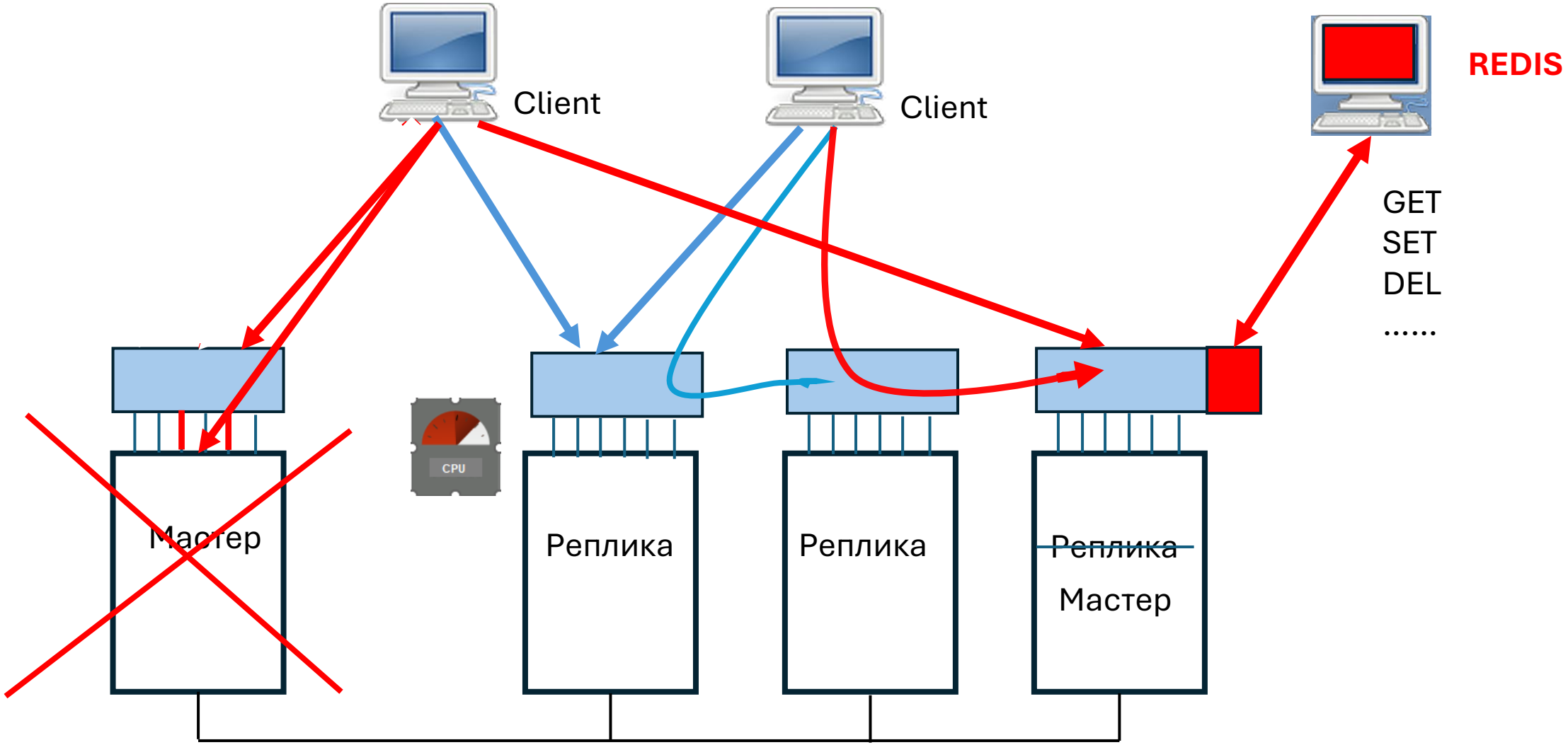
OLTP и неболь-шие DWH

OLTP и OLAP, DWH, HTAP

OLTP и большие OLAP

Очень большие OLAP

PROXIMA = Pooler + Proxy + Load Balancer

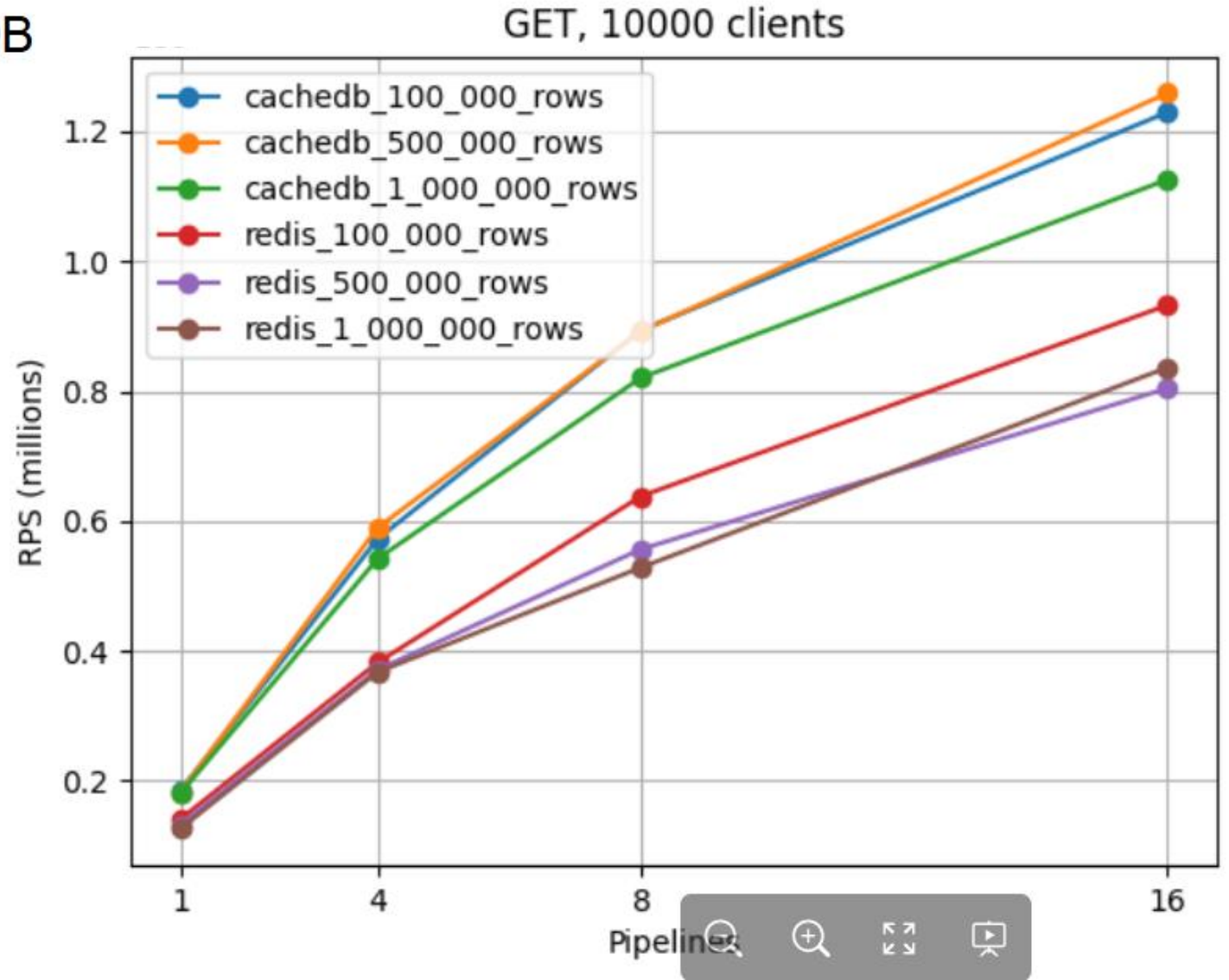


Эмуляция REDIS

- REDIS - REmote DIctionary Server
- СУБД ключ-значение в памяти. Доступ по ключу
- Быстрая выборка значений по ключу
- Изменение/удаление строк
- Инвалидация при изменении в БД (когерентный кэш)
- Подмножество команд REDIS
  - SET - создать, изменить объект
  - GET - прочитать объект по ключу
  - GETSET - изменить и вернуть старое значение
  - DEL - удалить поле
  - .....

CacheDB vs REDIS

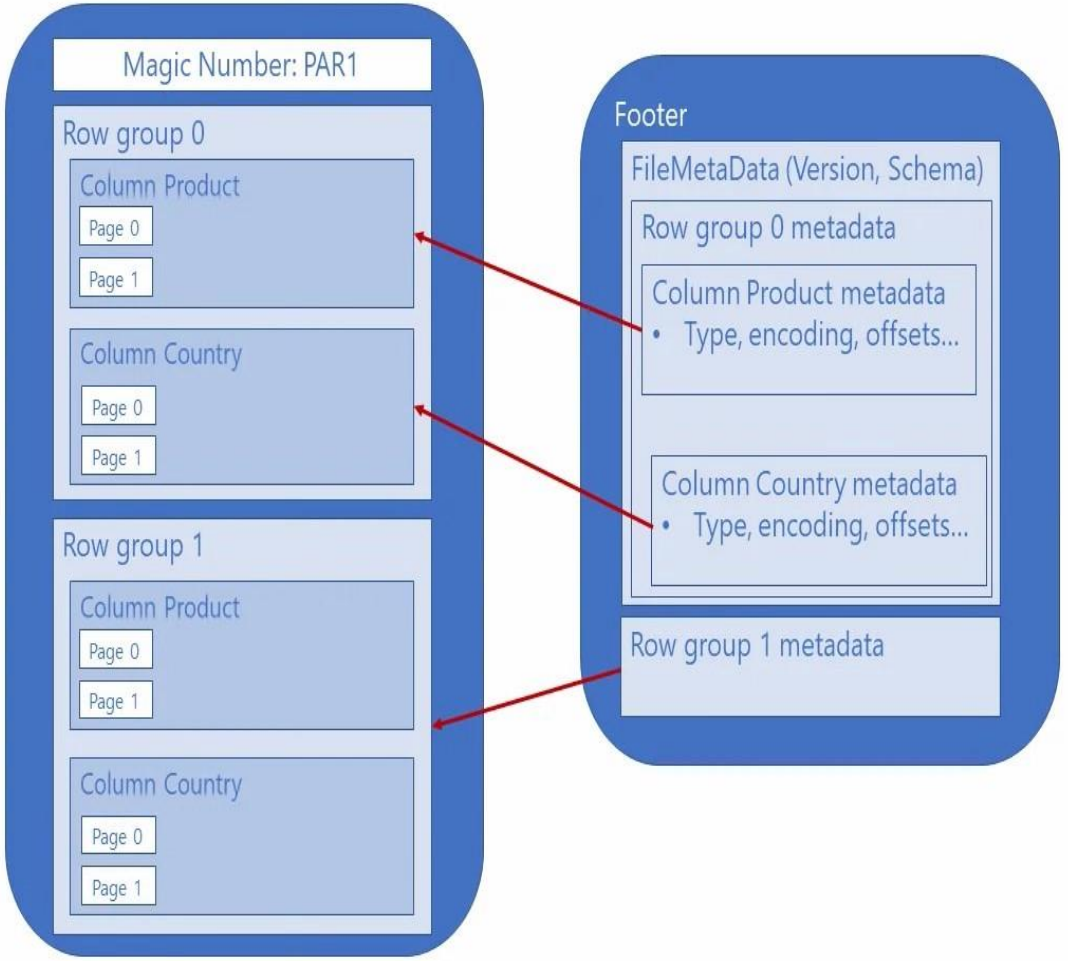
Row size = 519B



Postgres Pro Analytical Accelerator (AXE):

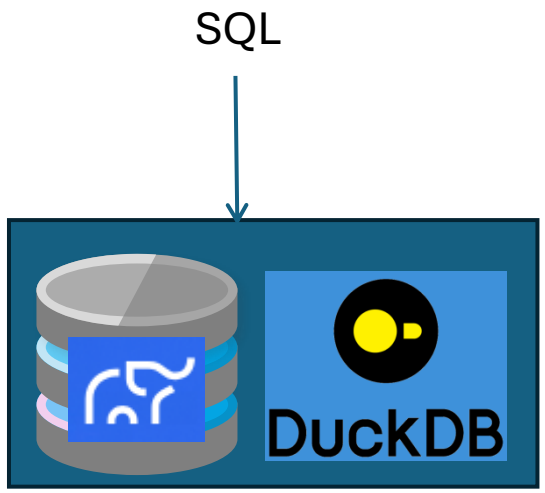
Postgres Pro Enterprise + AXE

- Структура Parquet идеальна для OLAP
- Паркетные файлы быстры т к читаются не все строки и не все колонки + сжатие – т е малый I/O (сжатие в 50 раз в пилоте)
- SIMD и параллелизм
- Файлы parquet создаются из таблиц Postgres с помощью утилиты ProCopy или команды COPY
- 10 млн записей преобразуются в Parquet за 8 сек
- Можно автоматически строить view в Postgres Pro, ссылающуюся на Parquet



Postgres Pro Enterprise + pgpro_AXE

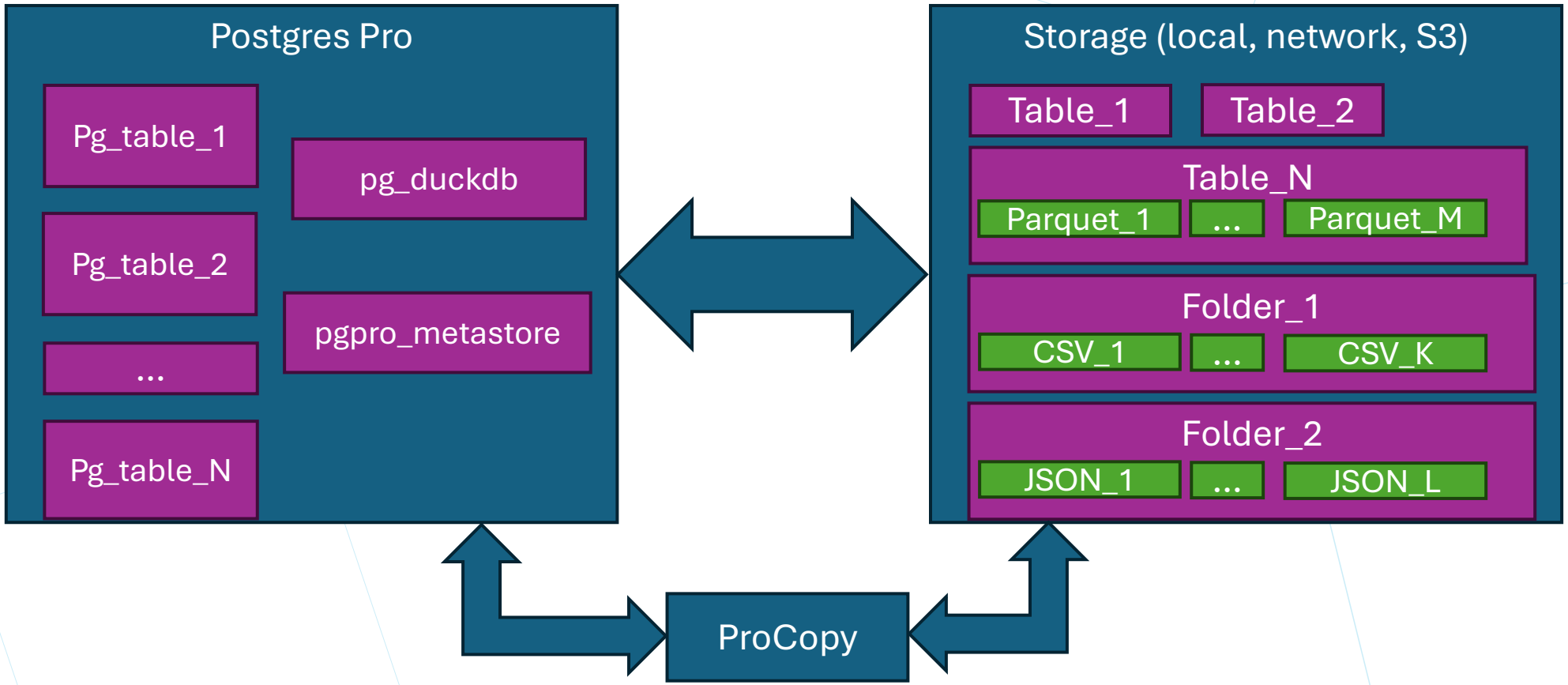
- Pgpro_duckdb - расширение к Postgres Pro Enterprise
- На том же компьютере, что и Postgres Pro
- Единая точка входа и единый SQL
- Таблицы Postgres Pro хранятся как обычно, данные AXE – в паркетных файлах на S3 или локально
- Запрос может быть к тем или иным таблицам или смешанный
- 2 оптимизатора запросов
- Для масштабирования аналитики можно создавать доп узлы с пустой БД, они работают только с parquet на S3



Pgdata

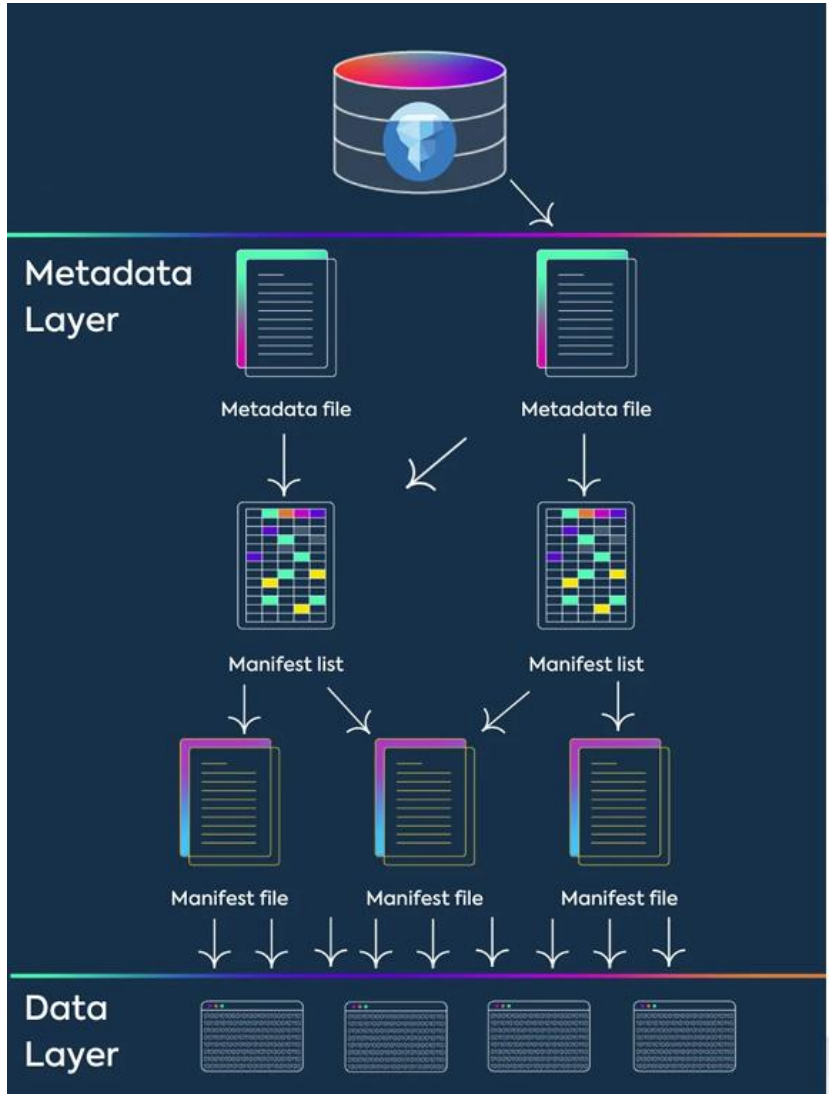
Parquet на S3 или локально

Архитектура первого релиза



Pgpro_metastore

- В версии 1 паркет только на чтение, в версии 2 (Q4 2025) будет расширение pgpro_metastore (изменение и транзакции)
- Pgpro_metastore - расширение к Postgres Pro Enterprise, аналог Apach Iceberg, но хранит каталог с метаинформацией в БД
- Механизм обновления Parquet файлов
  - Старые транзакции работают со старой версией
  - Новые транзакции во время формирования новой версии файла реконструируют snapshot
  - Новые транзакции после создания новой версии используют ее
- Metadata Layer хранит информацию о всех файлах паркетной таблицы
- Можно добавлять партиции в parquet таблицу
- Интеграция с Pgpro_backup



Результаты тестов

- Для ClickHouse, AXE, Postgres Pro - компьютер с 8 vCPU, 16 гб RAM
- Для CITUS и NN – гораздо больше
- 50 миллионов записей
- Select * from read_parquet(‘/data/……/file_pf_50_1*’) r where r[‘name’] = …. ;
- Время в секундах

Параметры	AXE	ClickHouse	Citus	Postgres Pro
Время операции 1	0,7	0,37	8,77	30,01
Время операции 2	0,68	0,64	9,29	42,31

```
Select r[‘file_name’], r[‘row_group_size’], r[‘statistics’]
   From duckdb_query(‘FROM parquet_metadata(“/data/…..file1.parquet”)’) r;
```

Postgres Pro без и с AXE

64 ядра, 64 Гб RAM, 6 млн строк

Запрос N	PG tables	AXE	Ускорение в
1	7.4186s	0.128s	57
2	4.0139s	0.175s	22
3	1.6309s	0.0373s	43
4	1.9727s	0.0423s	46
5	1.5942s	0.0359s	44
6	1.9371s	0.0346s	55
7	3.4457s	0.0713s	48
8	5.2353s	0.167s	31
9	1.6342s	0.0567s	28
10	2.0166s	0.0426s	47
11	1.6746s	0.0408s	41

Тесты другого заказчика

#	Результаты	PG heap 6 cores * 36 GB	AXE with SIMD 6 cores * 36 GB
1	Запрос актуальных данных для Хаба 1 (Вариант 1: DISTINCT ON)	3.585s	0.344s
2	Запрос актуальных данных для Хаба 1 (Вариант 2: LATERAL JOIN)	3.486s	0.984s
3	Запрос актуальных данных для Хаба 1 (Вариант 3: Подзапросы с MAX(load_date))	11.342s	0.818s
4	Запрос актуальных данных для Хаба 1 (Вариант 4: Подзапросы с MAX(load_date))	11.933s	0.804s
5	Запрос актуальных данных для Хаба 1 (Вариант 5: ROW_NUMBER)	4.230s	0.692s
6	Запрос актуальных данных для Хаба 1 (Вариант 6: CTE с агрегацией)	16.283s	0.274s
7	Запрос актуальных данных для Хаба 1 (Вариант 7: EXISTS)	6.426s	0.798s
8	Запрос актуальных данных для Хаба 1 (Вариант 8: Подзапросы с LIMIT 1)	3.310s	0.999s
9	Запрос актуальных данных для Хаба 1 (Вариант 9: LATERAL JOIN с ROW_NUMBER)	5.045s	0.962s
10	Запрос актуальных данных для Хаба 1 (Вариант 10: FIRST_VALUE)	4.997s	1.42s
11	Запрос полной истории для Хаба 1 (Вариант 1: LATERAL JOIN с подзапросами)	32.111s	5.93s
12	Запрос полной истории для Хаба 1 (Вариант 2: FIRST_VALUE)	25.192s	3.32s
13	Запрос полной истории для Хаба 1 (Вариант 3: ORDER BY/LIMIT)	27.029s	5.00s
14	Запрос полной истории для Хаба 1 (Вариант 4: EXISTS с подзапросами)	49.087s	9.51s
15	Запрос полной истории для Хаба 1 (Вариант 5: UNION ALL + FIRST VALUE)	1m:20.278s	14.06s
16	Запрос на точку во времени для Хаба 1 (Вариант 1: Подзапросы)	3.918s	0.637s
17	Запрос на точку во времени для Хаба 1 (Вариант 2: DISTINCT ON)	1.887s	0.262s
18	Запрос на точку во времени для Хаба 1 (Вариант 3: ROW_NUMBER)	2.866s	0.380s
19	Запрос на точку во времени для Хаба 1 (Вариант 4: EXISTS)	2.921s	0.390s

Tengri

Tengri – Основные преимущества

- Собственная разработка
- Использование современных технологий (parquet, iceberg, S3 и т д)
- Быстрое и простое развертывание
- Разделение слоев Compute и Storage
- Поддержка очень больших БД - петабайты !!!
- Высокая скорость выполнения сложных аналитических запросов

Tengri – СУБД для больших БД

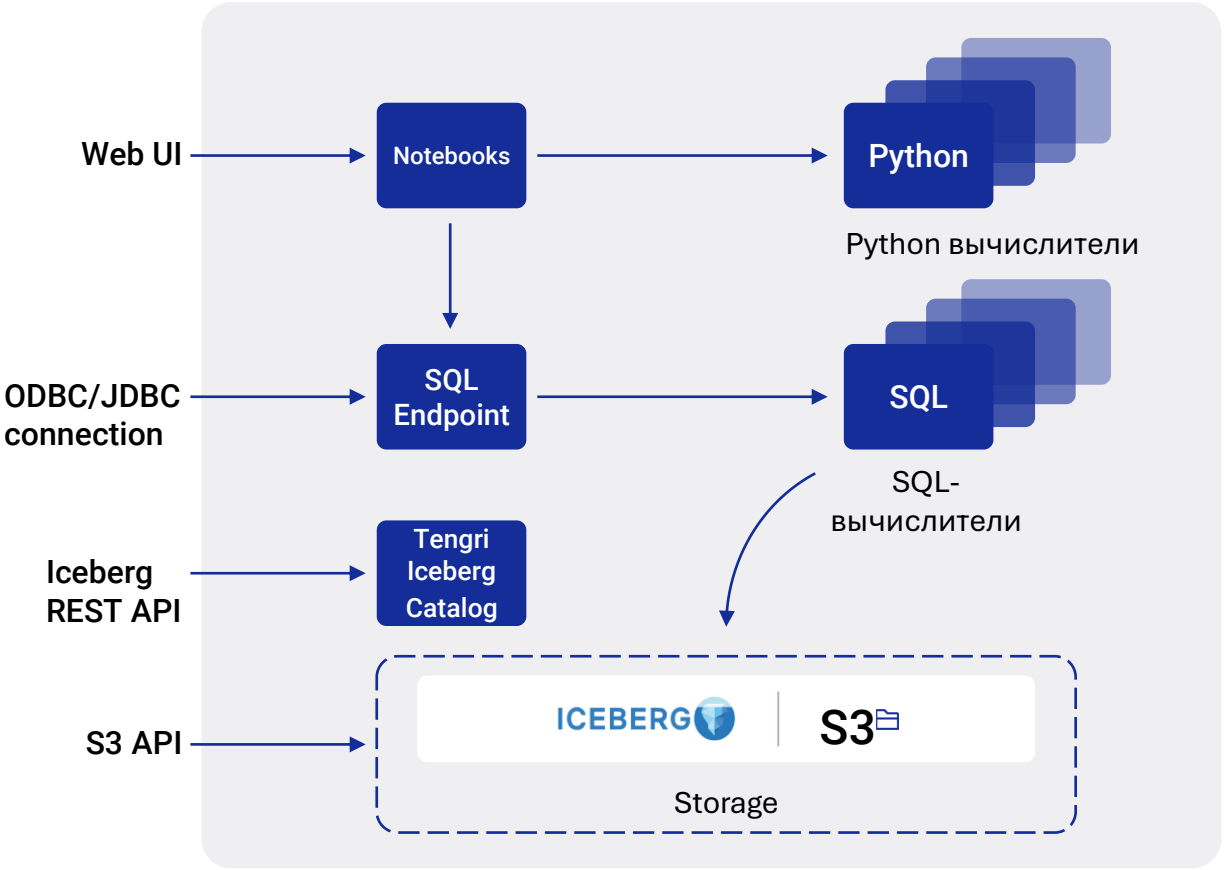
- СУБД для облака или on-prem а не расширение
- Хранит данные в Iceberg+Parquet на S3
- Поддержка диалекта Postgres SQL
- Собственная дефрагментация для Iceberg
- ACID, JDBC драйвер для Postgres
- Поддержка SQL и Python
- Графический интерфейс для администрирования, оркестрирования и сложных расчетов

Tengri – СУБД для больших БД - performance

- Колоночный формат хранения
- Сжатие колонок 5+ раз
- Morsel параллелизм
- SIMD команды
- Неограниченное масштабирование

Tengri Data Platform: Архитектура

- Каждый аналитик работает со своим SQL вычислителем
- SQL вычислитель - легковесный контейнер, запускаемый на аппаратной кластере за ~100 ms
- SQL вычислитель - получает эксклюзивные ядра (vCPU) и RAM бюджет: аналитики не мешают друг другу
- Вычислители разных размеров – S, M, L, …..
- SQL вычислитель - подгружает данные и выполняет SQL запросы.
- SQL вычислитель без нагрузки - отключается системой после периода ожидания.
- Часть оптимизации на SQL Endpoint (есть реплики), часть на вычислителе
- Агент запускает вычислители на разных узлах, данные пересылаются на вычислитель



Postgres Pro для VLDB, OLTP, OLAP и HTAP

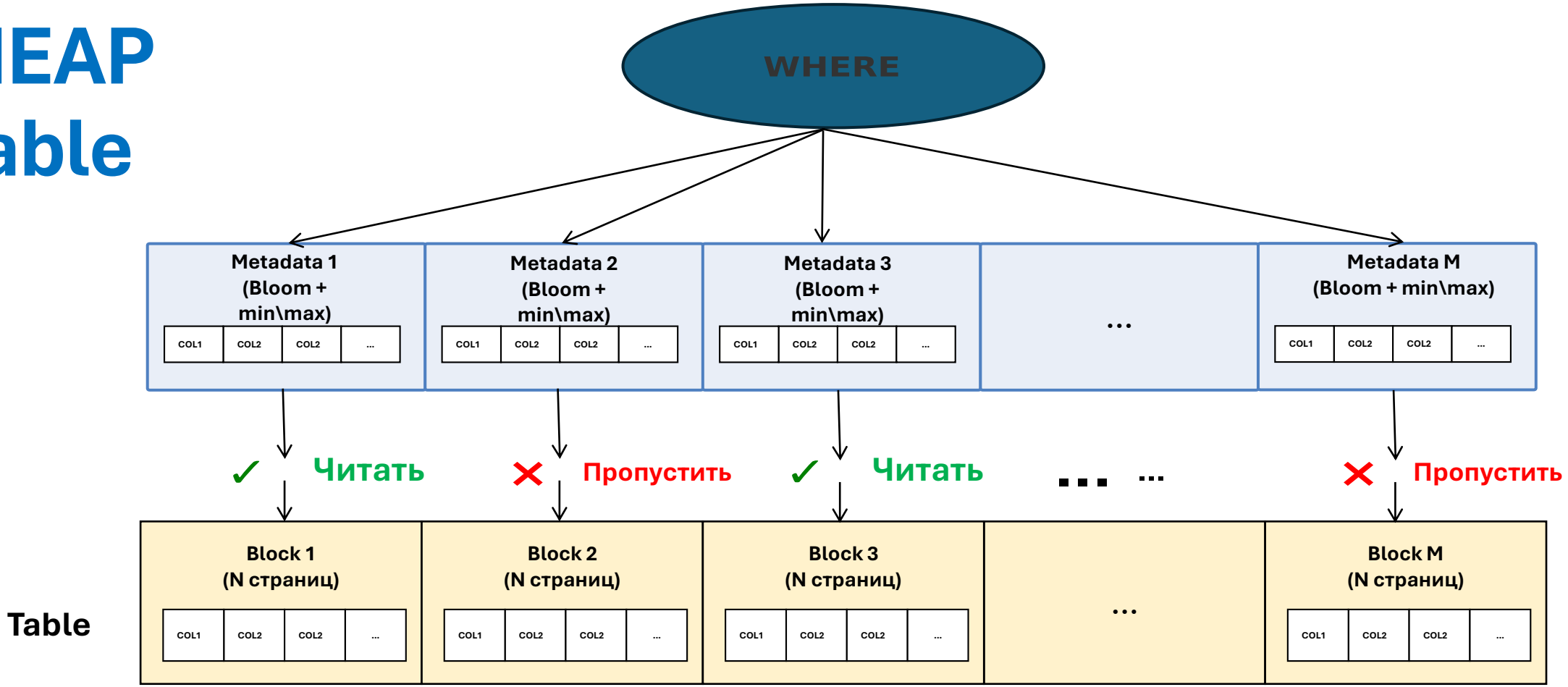
Что уже есть для OLAP и VLDB

- Небольшие DWH и витрины на Postgres Pro
- Partitioning
- Interval, reference partitioning (18)
- Сжатие
- ILM
- Глобальные индексы
- Шардинг
- Аналитические и оконные функции
- ROLAP, CUBE
- Materialized View
- Parallel DML и DDL, backup, restore

Что доделываем в Postgres Pro Enterprise 18

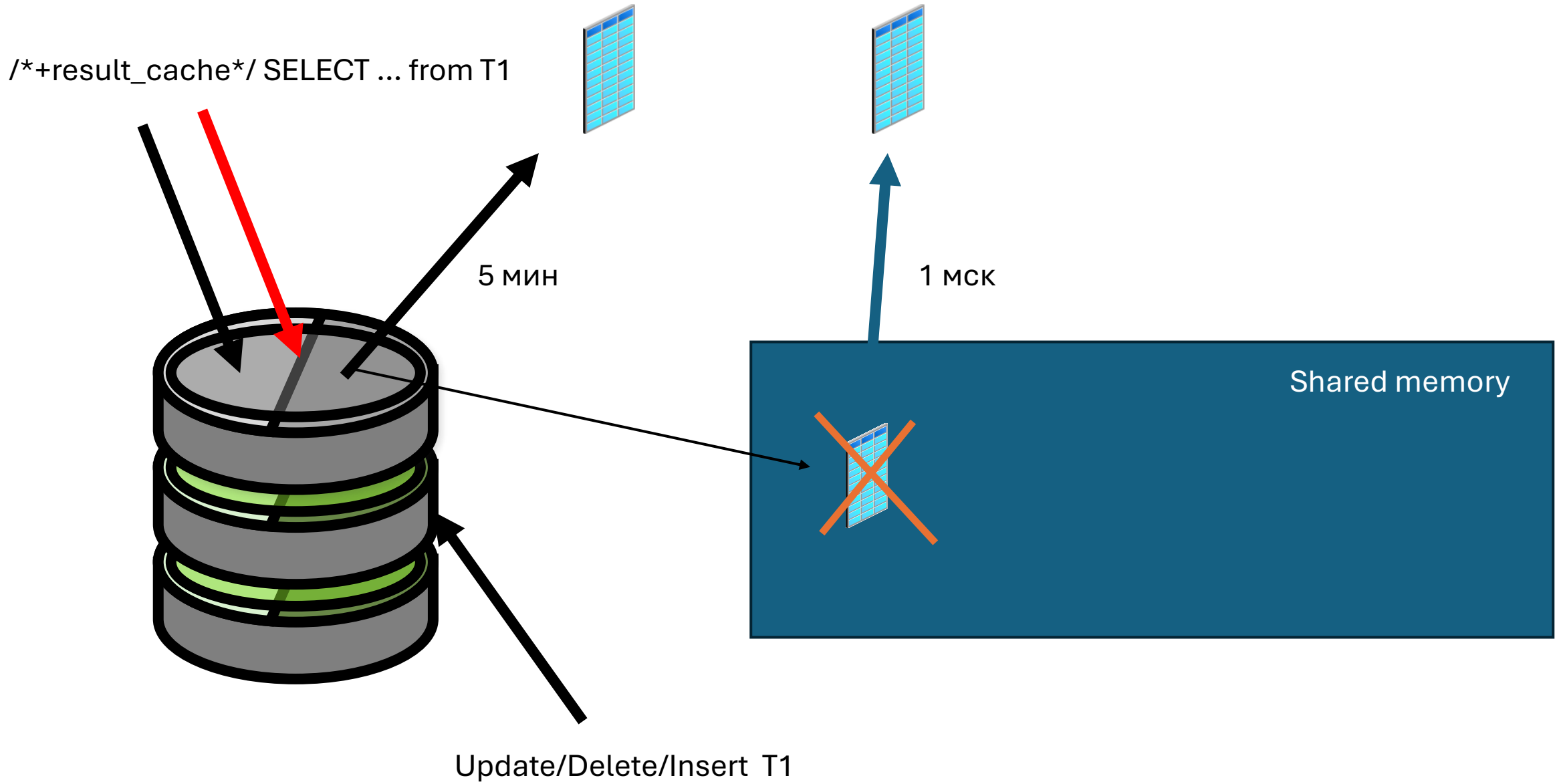
- IHeap таблицы
- Result cache
- In-memory columnar cache (HIMERA)
- Append optimized table

HEAP table



- Таблица делится на блоки страниц
- Для каждой колонки каждого блока в спецключах хранится сигнатура фильтра Блума и мин/макс значение содержимого колонки на страницах в блоке
- При сканировании таблицы условия проверяются по спецключам для каждого блока страниц. Не удовлетворяющие условиям блоки пропускаются целиком.
- Инвалидация спецключей производится во время Vacuum, либо после DML операций
- Ускоряет сканирование с условием

Result cache - быстрое выполнение одинаковых запросов разных сессий

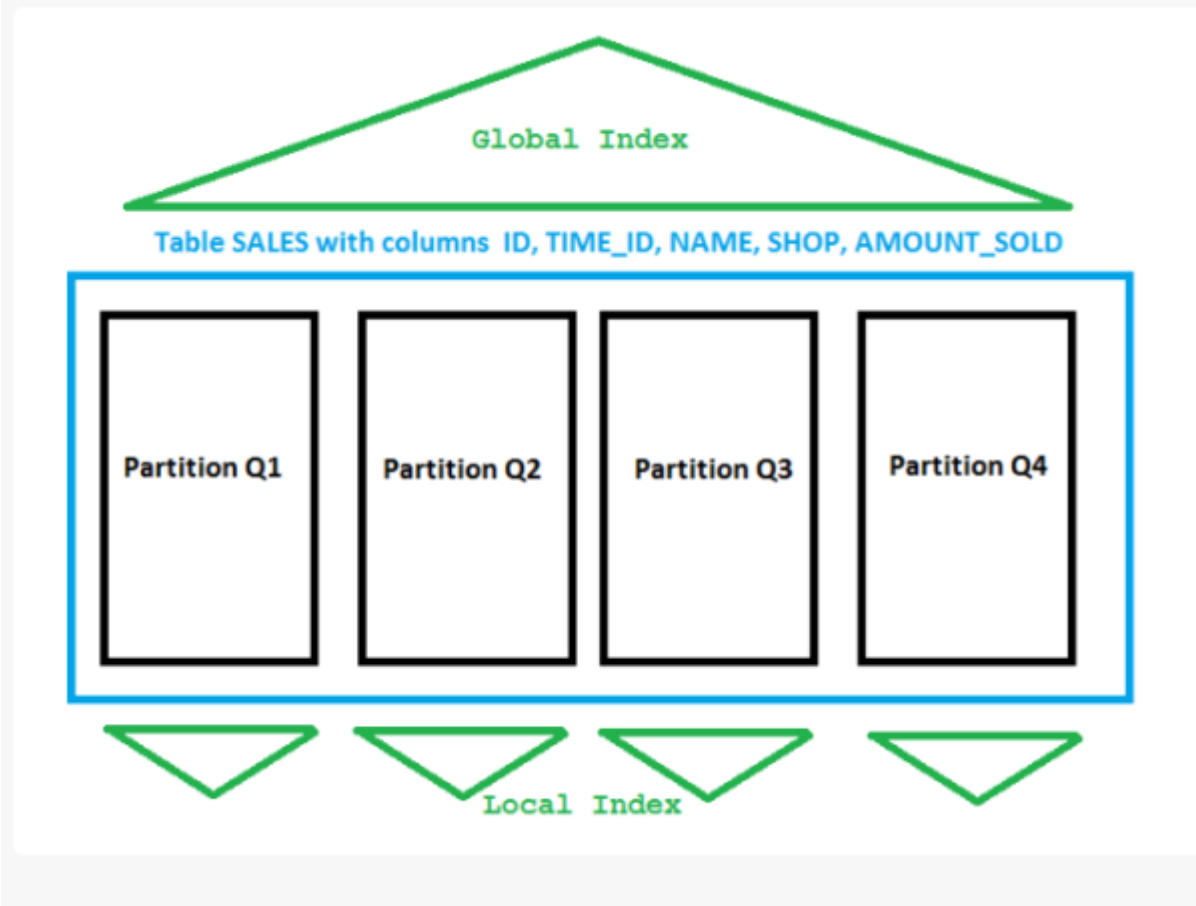


Result cache

- Запросы повторяются т к у всех пользователей тот же клиент/сервер приложений
- Долгие сложные запросы с небольшим результатом (несколько строк, агрегат и т д) на редко меняемых таблицах
- Быстрые OLTP запросы, часто выполняемые на редко изменяемых данных (справочники, курс валют, погода на будущее, свойства пользователей и т д)

Глобальные индексы

- Индексы ускоряют выборку
- При секционировании локальные индексы строятся для каждой секции и указывают только на ее строки и включают ключ секционирования
- Но часто существуют запросы, требующие индексации по колонкам, не совпадающим с ключом секционирования
- Те индекс должен указывать на строки разных секций – глобальный индекс
- Глобальный индекс может быть уникальным

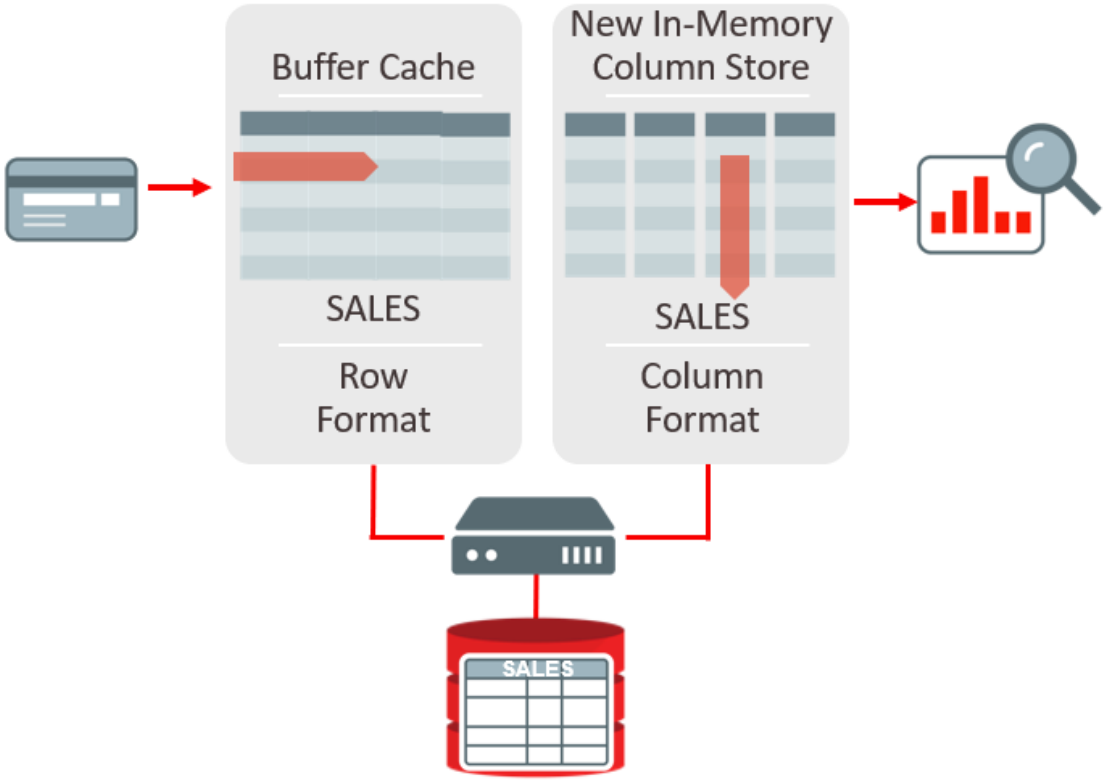


OLTP vs OLAP

- Традиционные СУБД не справляются с гибридной нагрузкой
- Либо OLTP, либо OLAP - обычные решения не обладают высокой производительностью для решения обеих задач одновременно
- Разделение OLTP и OLAP = отставание данных OLAP, высокая стоимость владения, новые точки отказа
- Часто не явные OLTP/OLAP а смешанная нагрузка

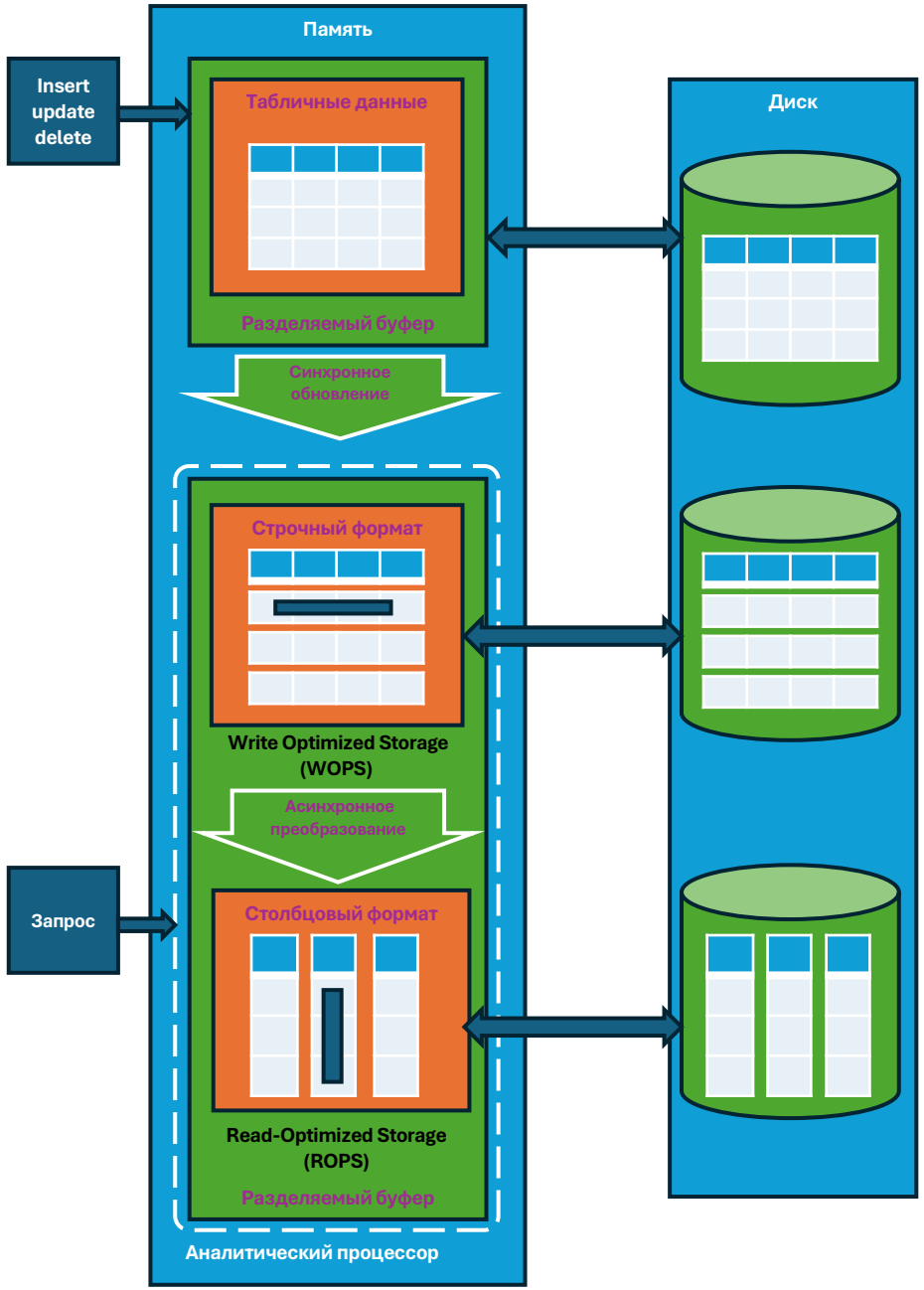
In-memory columnar cache

- 2 представления данных таблицы – построчное и поколоночное
- Построчное для OLTP
- Поколоночное для OLAP
- Оптимизатор выбирает оптимальный план
- Обновления на построчном и синхронизация кэшей
- Сильно ускоряет аналитические запросы



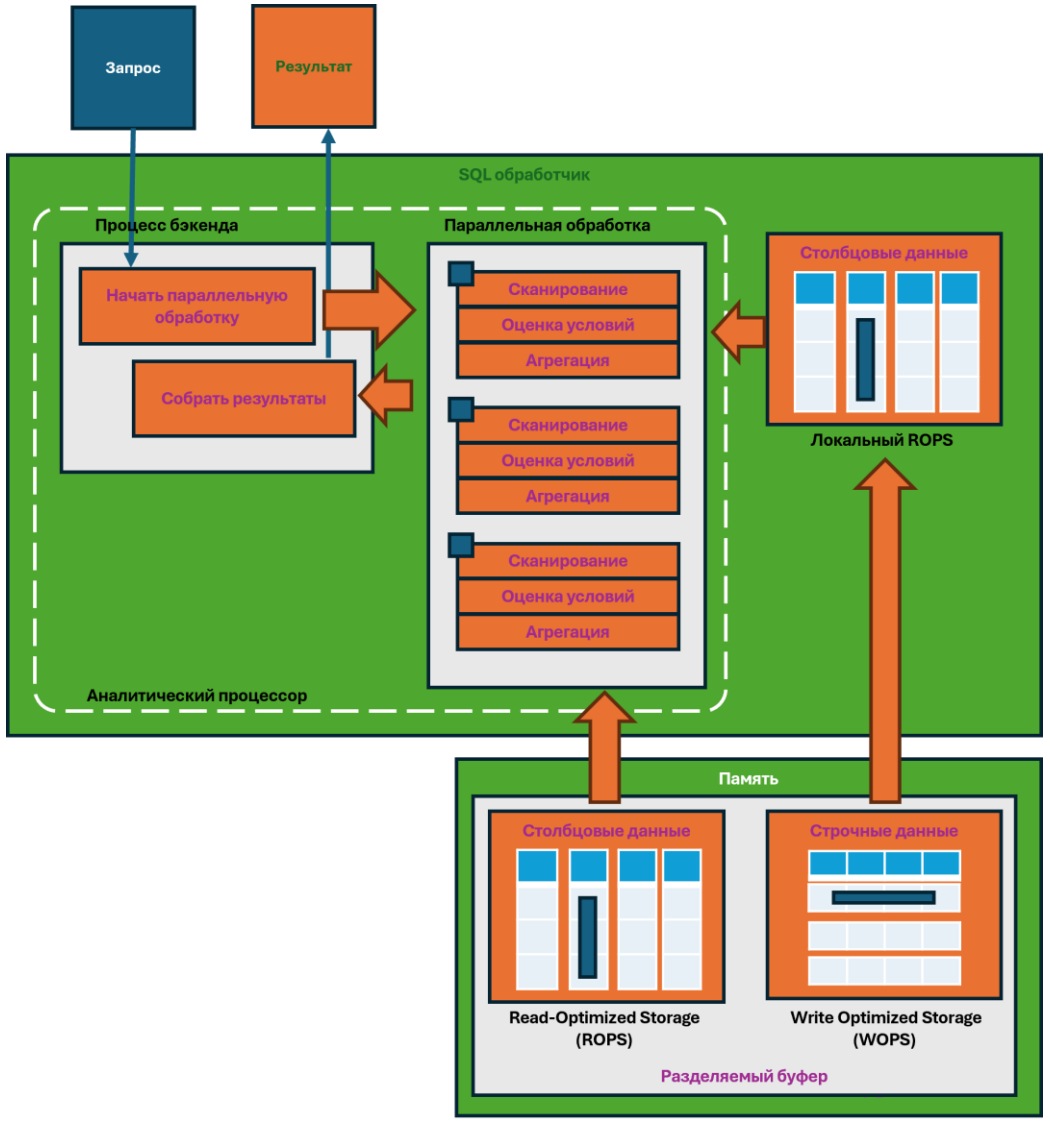
WOPS + ROPS = транзакционность и аналитика

- Write Optimized Storage (WOPS) — для записи OLTP-данных (без индексов и сжатия)
- Read Optimized Storage (ROPS) — для анализа OLAP с индексами и сжатием
- Данные из WOPS асинхронно преобразуются в ROPS



Эффективная гибридная архитектура

- In-Memory обработка данных
- Хранит данные как в строковом (OLTP), так и в поколоночном (OLAP) формате
- Синхронные данные
- Поддержка MVCC
- Производительность аналитики до 10 раз выше, при той же производительности в OLTP от Postgres
- Аналитические индексы можно удалить для таблиц с часто меняемыми данными
- Агрегаты по колонкам



HIMERA (Hybrid In-Memory Real Time Analytics)

- Создается командой CREATE INDEX .....on T1 using himera(c1, c2, c3);
- Ограничение – 32 колонки
- 3 +1 области памяти
- Shared buffer
  - Buffer cache
  - WOPS – изменения копятся но применяются по commit
  - ROPS – меняется асинхронно
- Для отображения всех зафиксированных изменений – в памяти backend локальный ROPS (ROPS+изменения из WOPS) => согласованность
- Оптимизатор учитывает стоимость поколоночных шагов
- Области на диске – для paging, они и WOPS – вспомогательные объекты

Заключение

- OLTP и OLAP решают разные бизнес-задачи
- Правильный выбор архитектуры критичен для производительности
- Современные системы стирают границы между двумя подходами
- Postgres Pro предлагает спектр решений, что позволяет реализовать оптимальную архитектуру системы


Q&A