Большие таблицы в PostgreSQL

или как превратить 60+ Tb в 10+ Tb

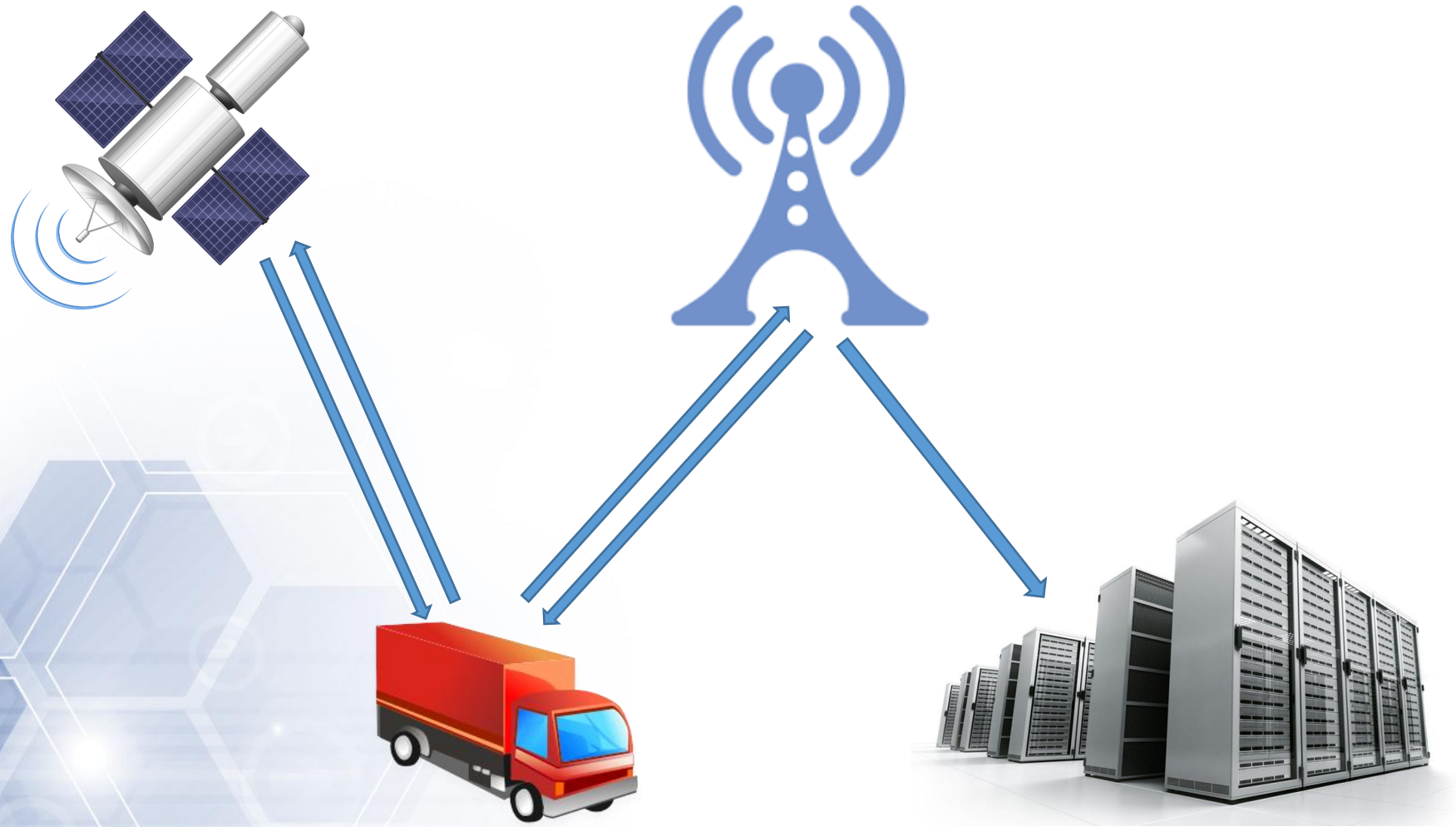
Вадим Яценко

март, 2017

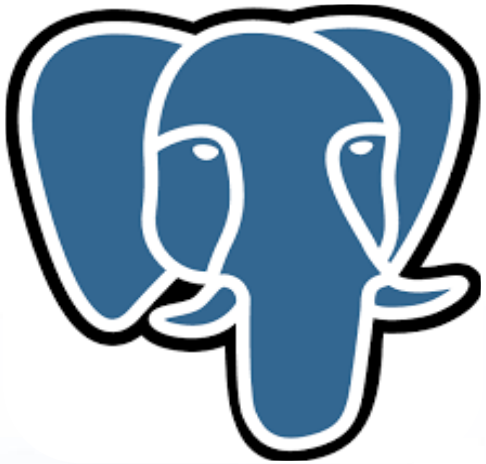
«Прогресс Софт» — российский разработчик информационных систем на базе открытых технологий.

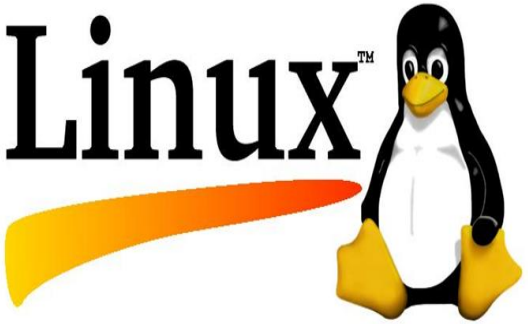
На протяжении многих лет мы разрабатываем и реализуем крупномасштабные проекты по автоматизации государственных органов власти РФ и корпораций.

Проект



Технологии





SSD

NUTANIX™

Цифры

Данные

- 2 000 000 бортовых устройств
- 2 000 000 координат/минуту
- 520 000 000 запросов в день

Нагрузка

- 2 Дата-центра
- 25 PostgreSQL серверов
- 80 000 – 100 000 Tuples Per Second

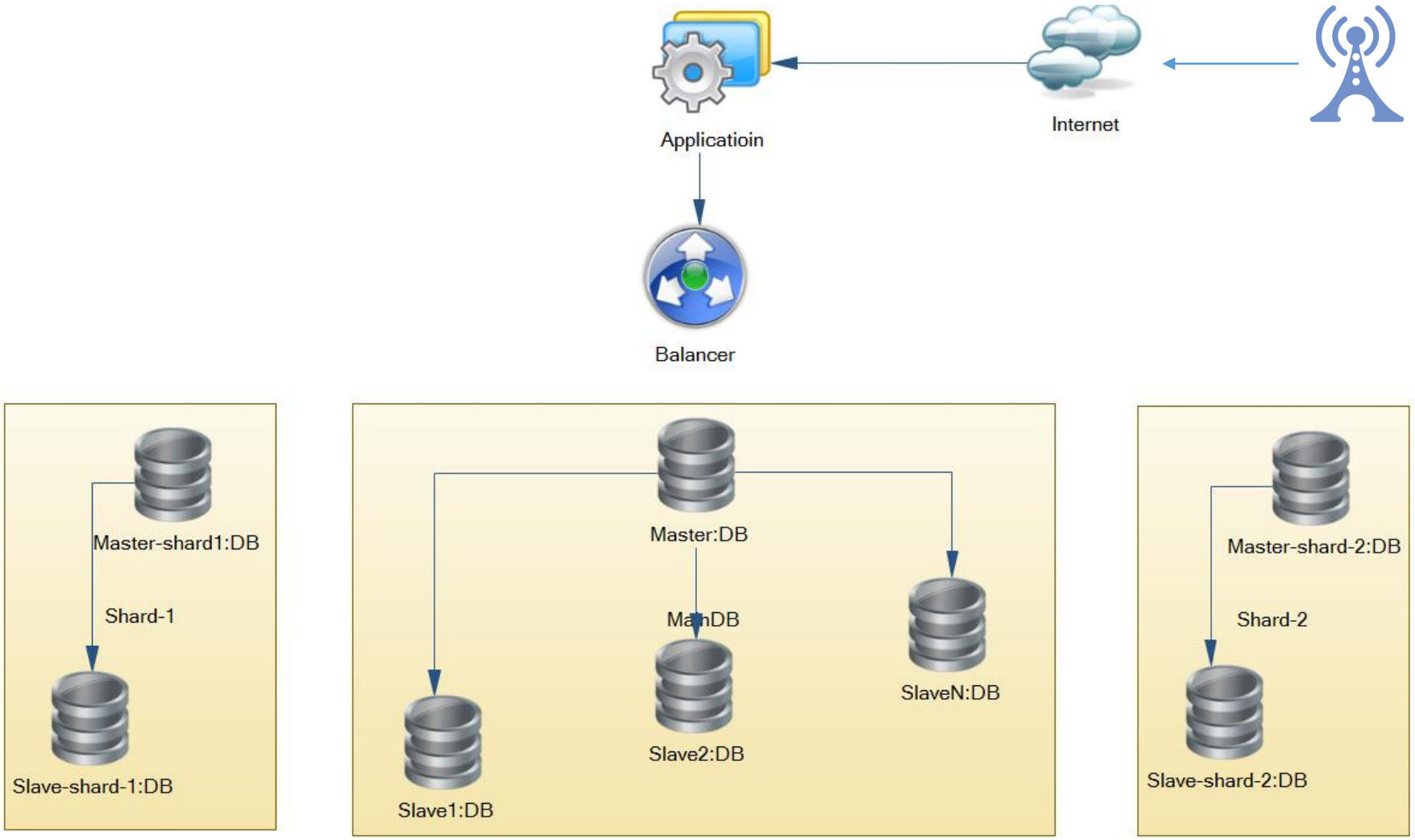
Старт проекта

N masters

+

FDW (postgres_fdw)

Старт проекта. Шардинг



Старт проекта. Кластер хранения координат

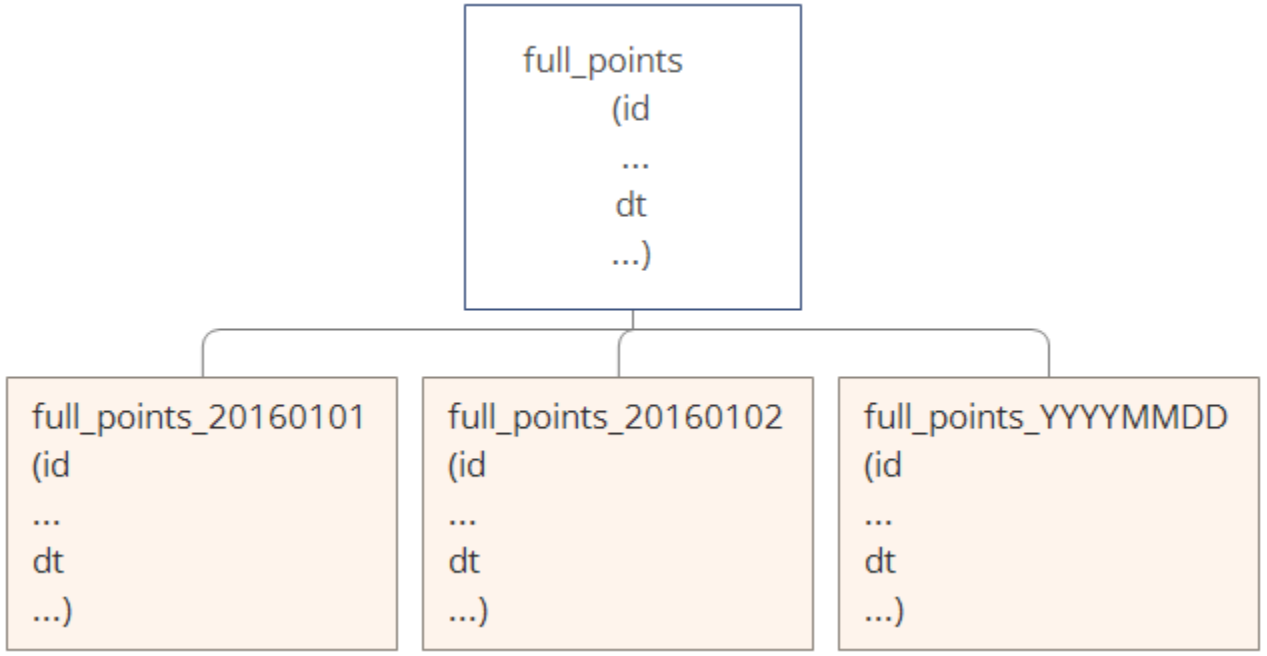
- 1 плоская таблица на сервере
- ~ 300 000 000 строк в сутки
- INSERT в таблицу
- Master – Slave (PSR)

Первый этап. Проблематика.

- Мастер не справляется с записью;
- Быстрый рост размера БД;
- Медленная выборка
- Долгий VACUUM(FULL), ANALYZE;
- Долгое резервное копирование

Первый этап. Решение

- Партиционирование:
  - RANGE = 1 сутки
  - CHECK по dt
- Замена INSERT на COPY FROM STDIN



Первый этап. Итоги

Было

- Плоская таблица
- INSERT
- 300 000 000 в день
- 50 Gb в день
- Медленный SELECT

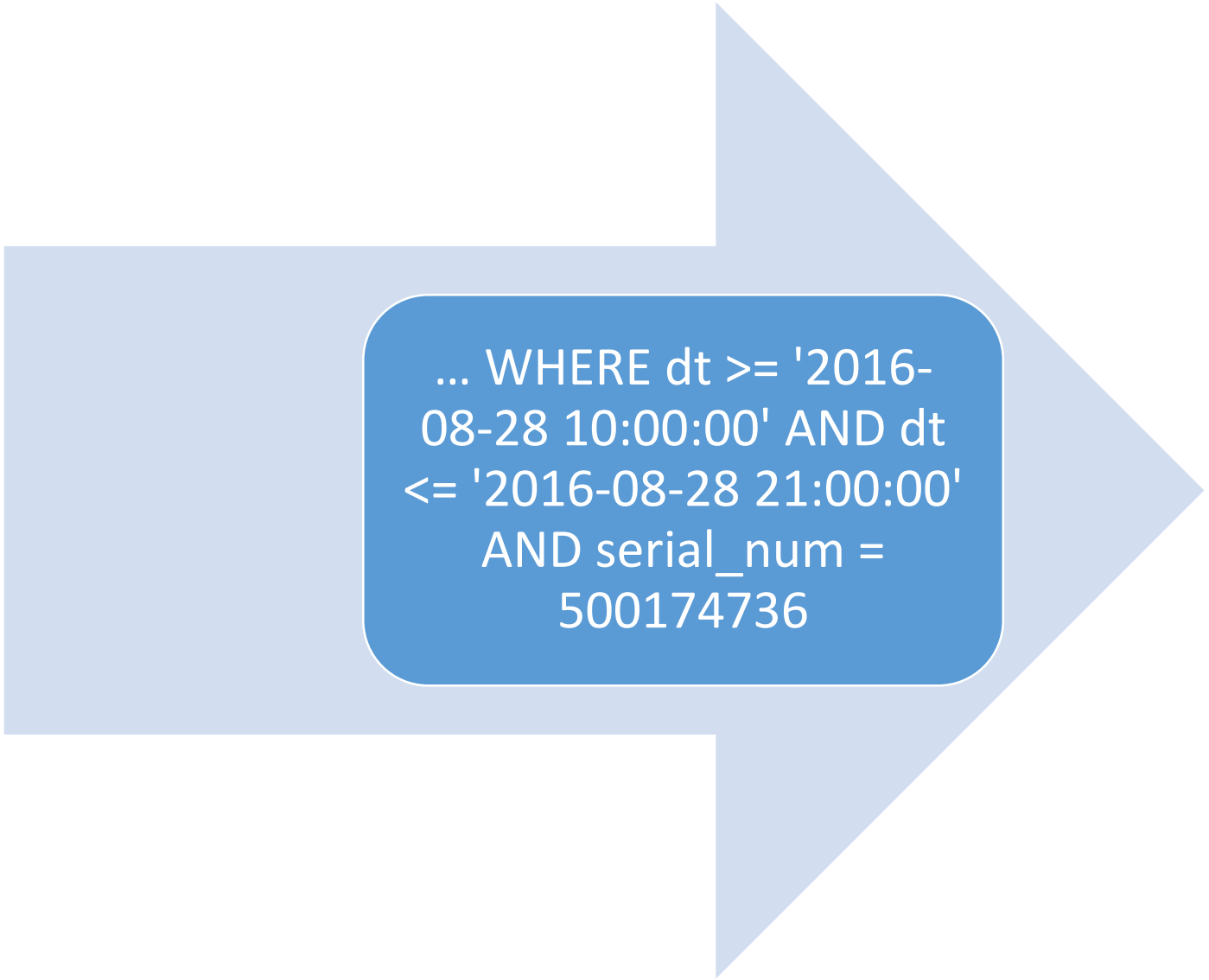
Стало

- Партиционирование
- COPY FROM STDIN
- 800 000 000 в день
- 150 Gb в день
- Быстрый SELECT

Второй этап. Проблематика

- ~150 GB в каждая таблица
- + Standby
- Постоянный рост размера данных в сутки
- Таблица > RAM
- VACUUM, ANALYZE > 2 часов

Второй этап. Типовое условие запроса



Второй этап. Решение

- JSONB вместо плоской таблицы:

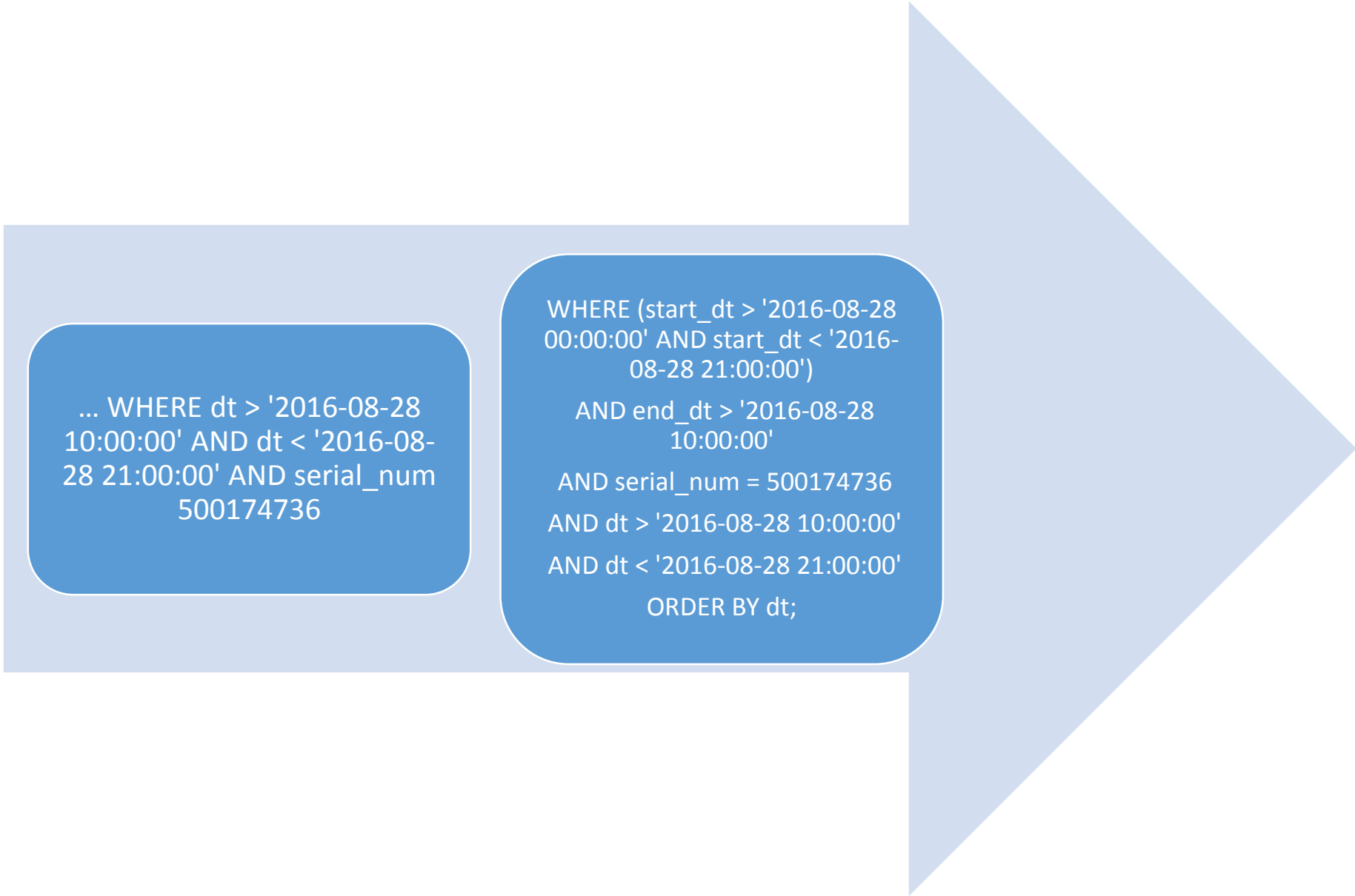
```
[{"dt": 1472361159, "alt": 186, "lat": 56.37424087524414, "lon":
38.70574188232422, "hdop": 23.100000381469727, "vdop": 23.100000381469727,
"speed": 40, …, "vehicle_class": 1}]
```

- serial_num отдельное поле
- Замена dt на start_dt и end_dt
- Использование VIEW в SELECT запросах
- Составной btree индекс по serial_num, start_dt, end_dt
- Доработка WHERE в запросе

Второй этап. VIEW

```
CREATE OR REPLACE VIEW v_full_points AS
 SELECT t.id,   t.serial_num,  (t.points ->> 'lat'::text)::double precision AS lat,
    (t.points ->> 'lon'::text)::double precision AS lon,  (t.points ->> 'alt'::text)::smallint AS alt,
    to_timestamp((t.points ->> 'dt'::text)::double precision) AS dt,
   (t.points ->> 'is_valid'::text)::boolean AS is_valid,  (t.points ->> 'speed'::text)::smallint AS speed,
    (t.points ->> 'course'::text)::smallint AS course,  (t.points ->> 'hdop'::text)::real AS hdop,
    (t.points ->> 'vdop'::text)::real AS vdop,  (t.points ->> 'vehicle_class'::text)::smallint AS vehicle_class,
    (t.points ->> 'sat_visible'::text)::smallint AS sat_visible, (t.points ->> 'sat_used'::text)::smallint AS sat_used,
    t.created_at,  t.start_dt,  t.end_dt
   FROM ( SELECT full_points.id,  full_points.serial_num,  full_points.start_dt,  full_points.end_dt,
full_points.created_at,  jsonb_array_elements(full_points.points) AS points
         FROM full_points) t;
```

Третий этап. Новый условие запроса



Второй этап. Итоги

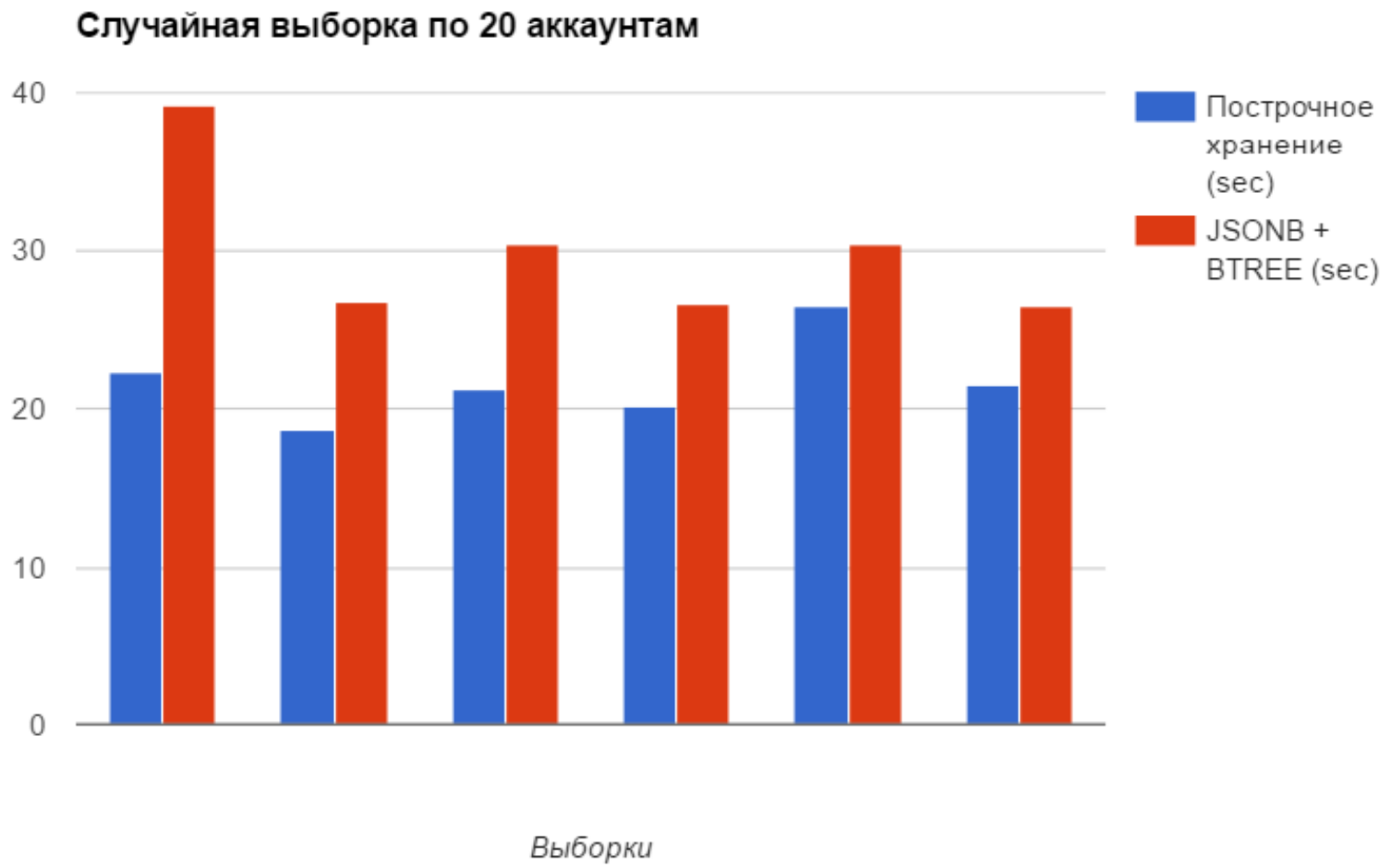
Было

- ~150 GB каждая таблица
- Построчное хранение координат
- Быстрый SELECT
- Таблица > RAM
- VACUUM, ANALYZE > 2 часов

Стало

- ~27 GB каждая таблица
- Пакеты JSONB
- SELECT медленнее на 30%
- Таблица < RAM
- VACUUM, ANALYZE ~ 2 минуты

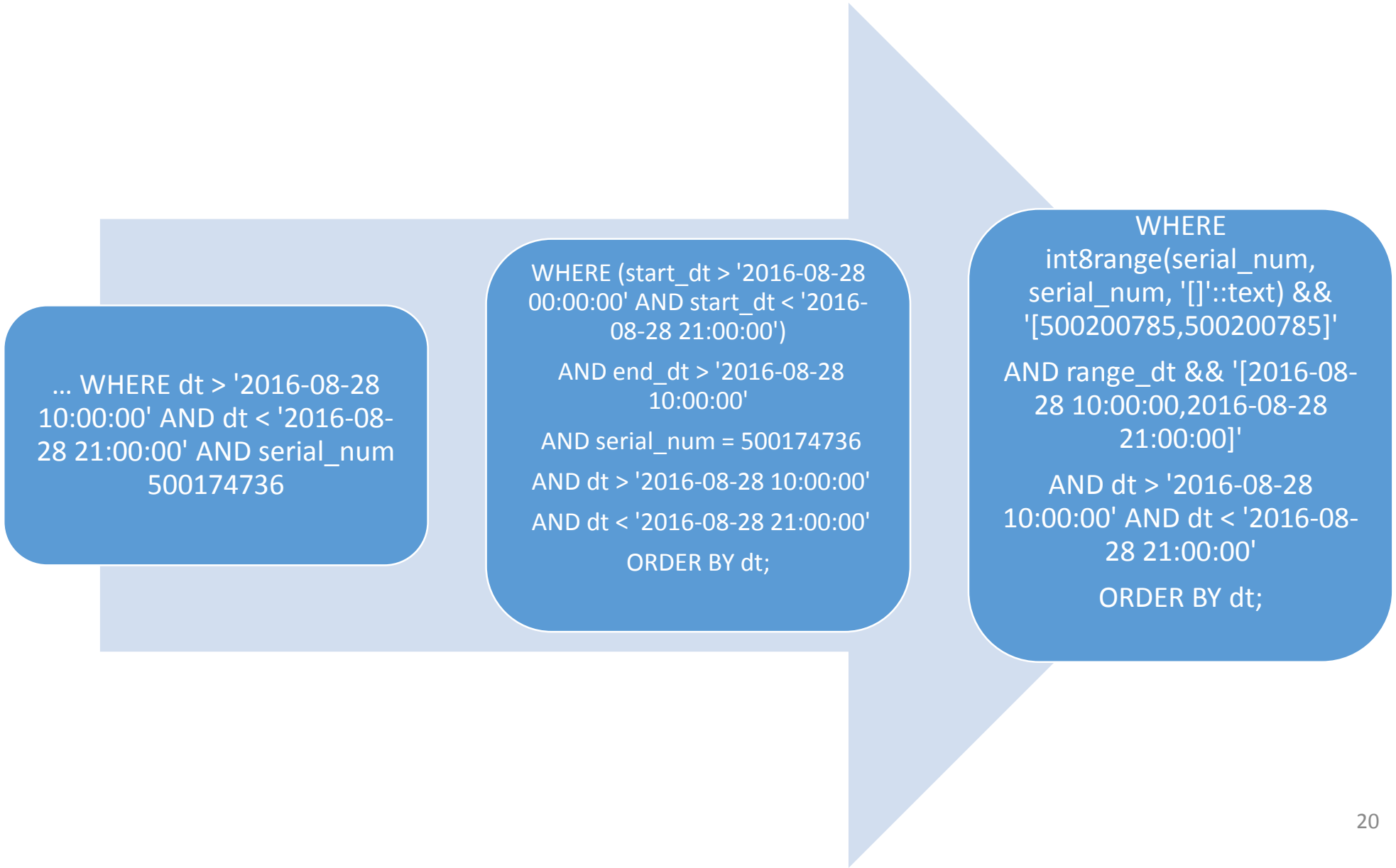
Третий этап. Проблематика



Третий этап. Решение

- Замена start_dt и end_dt интервалом tsrange:
- Используем составной GIST индекс:
- Снижаем random_page_cost до 1.5
- FILLFACTOR индексов 100%

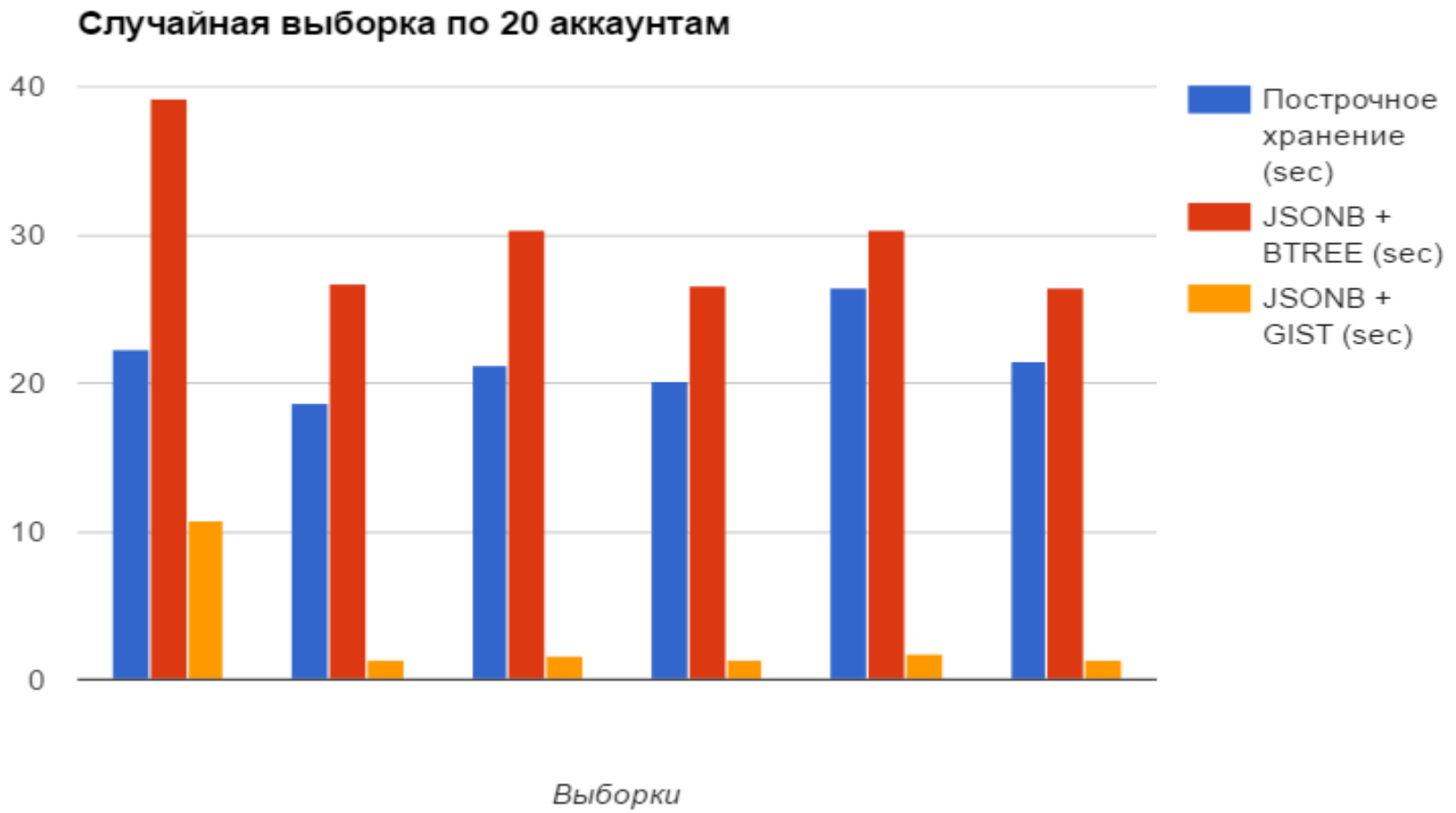
Оптимизация. Новое условие запроса



Третий этап. Итоги

Было	Стало
• ~27 GB каждая таблица • Пакеты JSONB • SELECT медленнее на 30% • Таблица < RAM • VACUUM, ANALYZE ~ 2 минуты	• ~27,5 GB каждая таблица • Пакеты JSONB • SELECT на порядок быстрее • Таблица < RAM • VACUUM, ANALYZE ~ 2 минуты

Время выполнения запроса



Profit

- Минимальные изменения бизнес функций и кода проекта
- Данные сжаты ~ в 7 раз
- В день ~ 40 GB
- Скорость чтения увеличена
- на порядок (GIST)
- Простое обслуживание БД

Архивация данных

Требования:

- Данные в оперативном доступе – 1 год
- Архивные данные – 3 года

PL/PgSQL + PL/Sh:

- PL/PgSQL - логика, транзакционность
- PL/Sh – работа с утилитами pg_dump/pg_restore

PL/PgSQL + PL/Sh

- Запуск по расписанию
- ~ 6-8 часов архивация одной партиции
- Транзакционность в рамках функции PL/PgSQL
- Запуск на мастере, дамп с Standby
- pg_xlog_replay_pause() + pg_xlog_replay_resume()

PostgreSQL 9.4

Не хватает из 9.5/6 :

- Параллельный seq scan
- pg_pathman
- Наследование FDW таблиц
- BRIN-индексы

Нет совсем в PostgreSQL:

- Check по FDW INHERIT таблицам
- Отчуждаемые таблицы/tablespace
- Планировщик не понимает оператор «&&»

Спасибо за внимание!
Ваши вопросы?

www.npoprogress.com