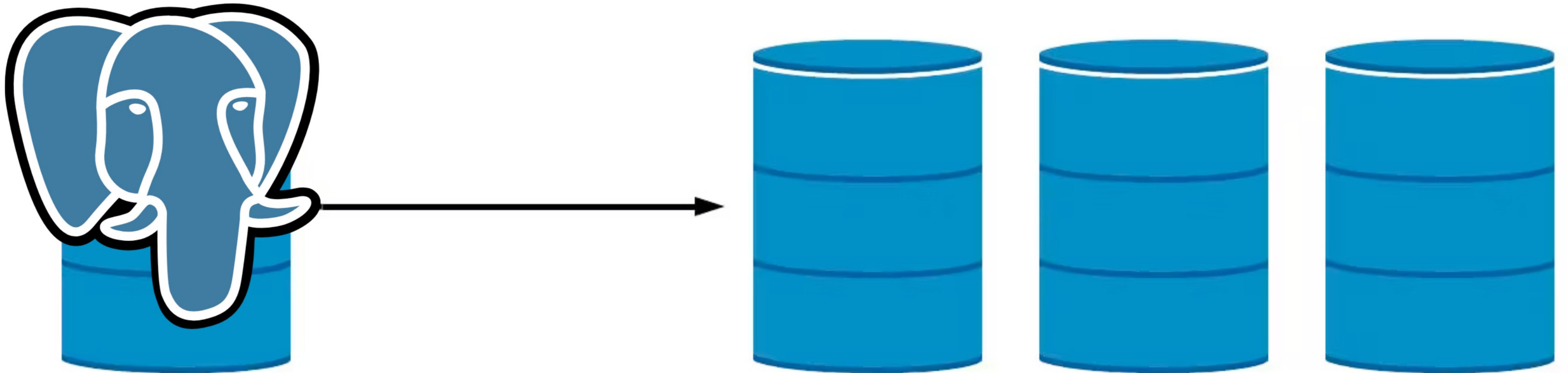


Масштабирование баз данных

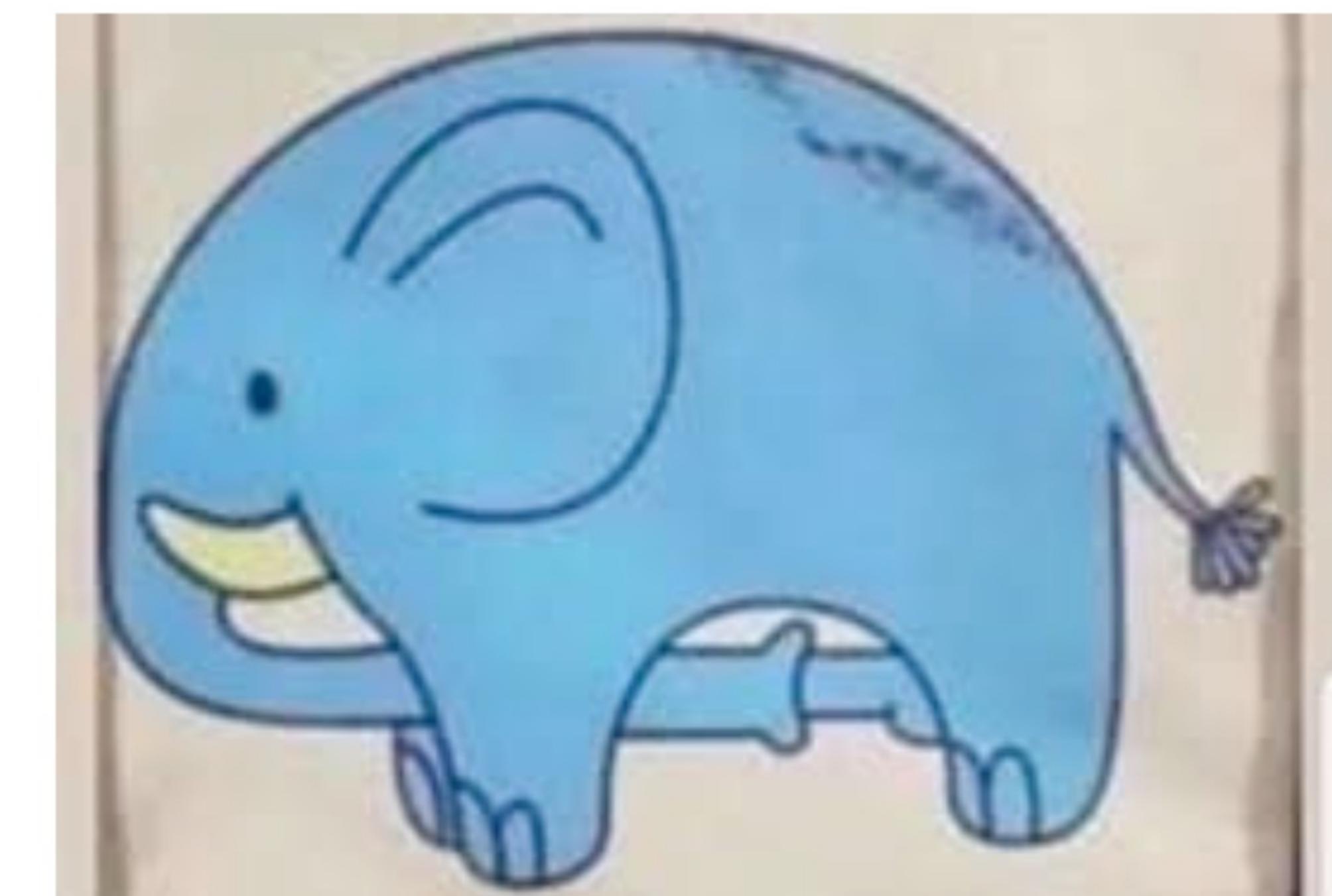


от оптимизации к шардированию

ШАГ 0. Работаем с тем что есть

- Переписываем запросы
- Добавляем недостающие индексы
- Перенастраиваем пулы, греем кеши
- Выносим данные в другие бд
- ...

<https://youtu.be/ZF5SLfrJ9Jw>



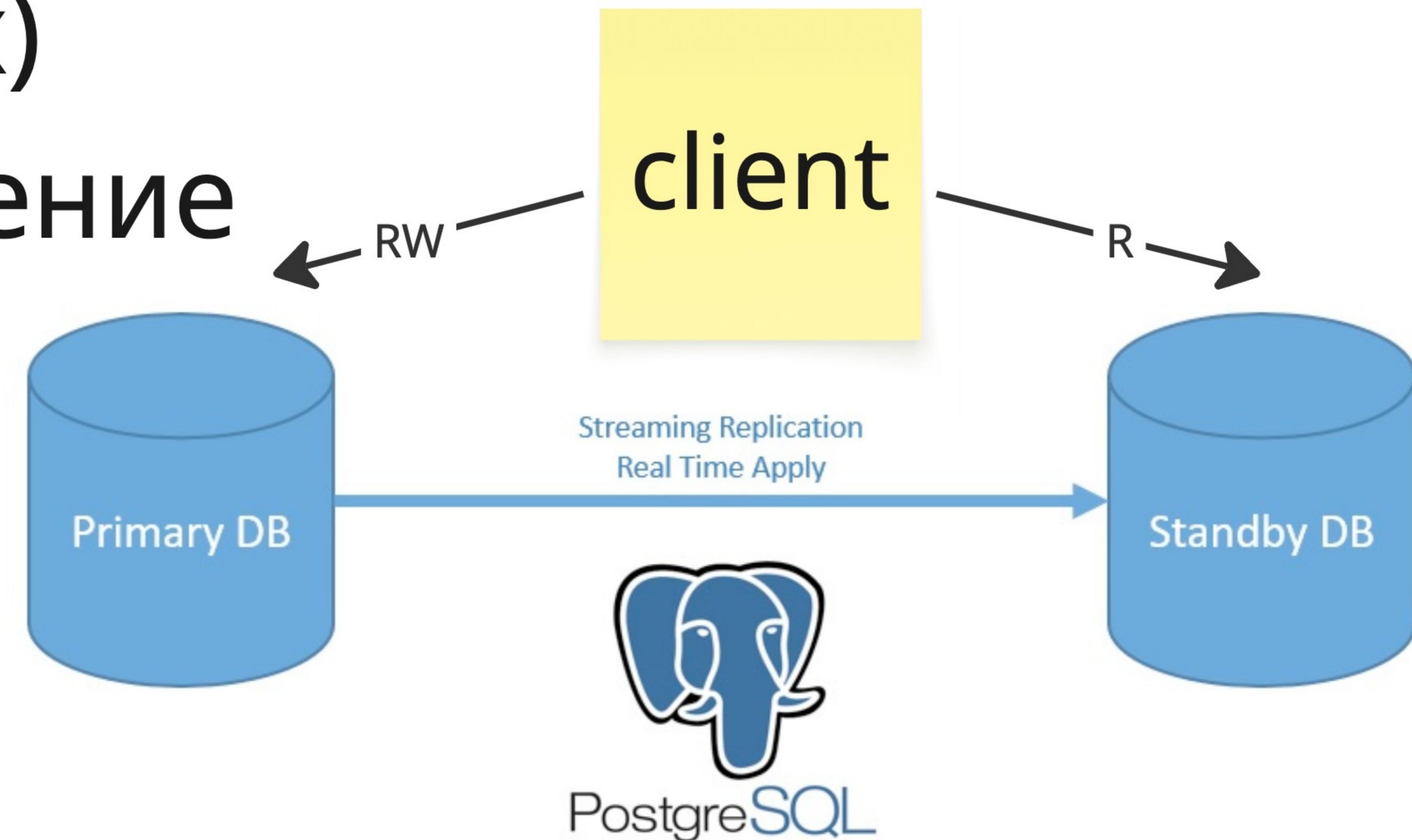
ШАГ 1. Добавляем ресурсов

- На 1 узел до **1TB** хранимых данных
(только сырых простых данных)
- До **100Gb** на 1 таблицу
- 1 узел 128CPU и 1TB ram может обработать ~2000 коннектов без bouncer и в 10 раз больше с



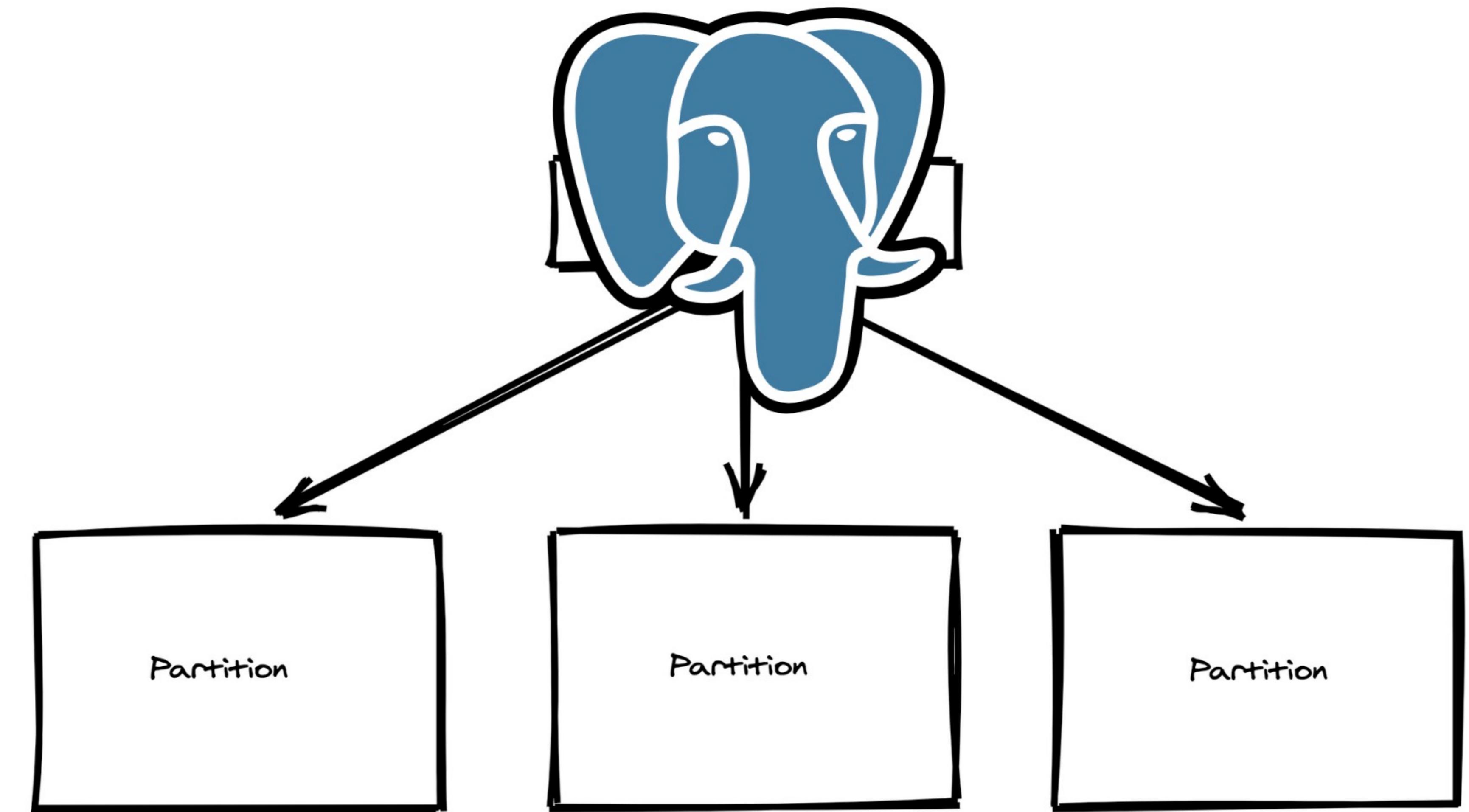
ШАГ 2. База тормозит: чтения

- Добавляем репликацию
 - синхронную или асинхронную
(от типа данных)
- На реплике только чтение



ШАГ 3. База тормозит: запись в один узел

- Система все еще
 - До 1ТВ общий размер
 - Размер таблицы 50-100GB
 - Более 100М строк
- Работает на уровне Бд

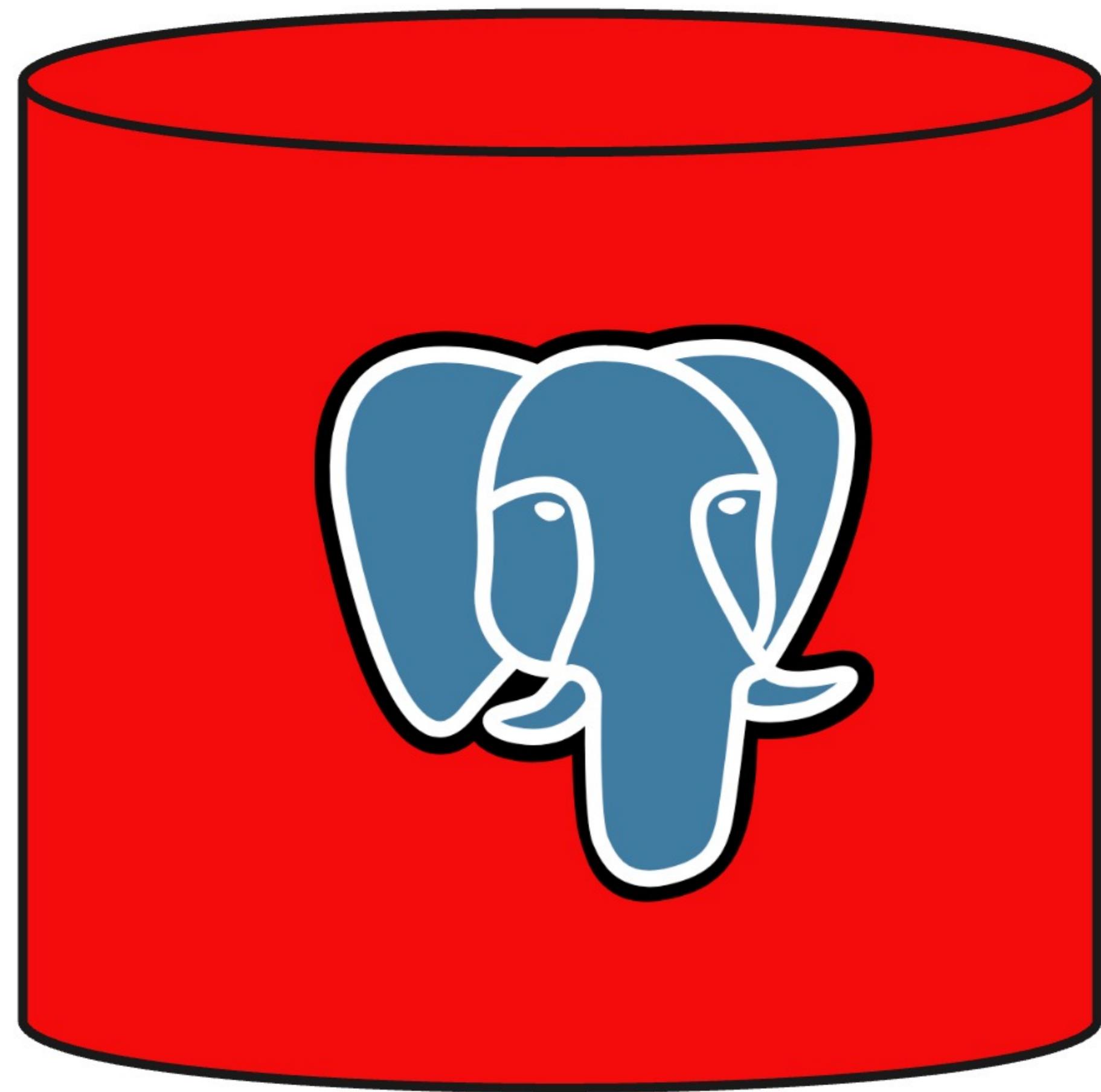


ШАГ 3. База тормозит: запись в один узел

- Не сильно поможет, когда запросы по ID
 - `PARTITION BY HASH (id);` будет искать по всем
 - Придется добавлять `tenant_id` для попадения в нужную партицию (почти изобрели шардирование) может быть шагом 1

```
1 CREATE TABLE clients (
2     id BIGSERIAL,
3     tenant_id INT NOT NULL, -- partition pk
4     created_at TIMESTAMPTZ DEFAULT NOW(),
5     PRIMARY KEY (id, tenant_id)
6 ) PARTITION BY LIST (tenant_id);
```

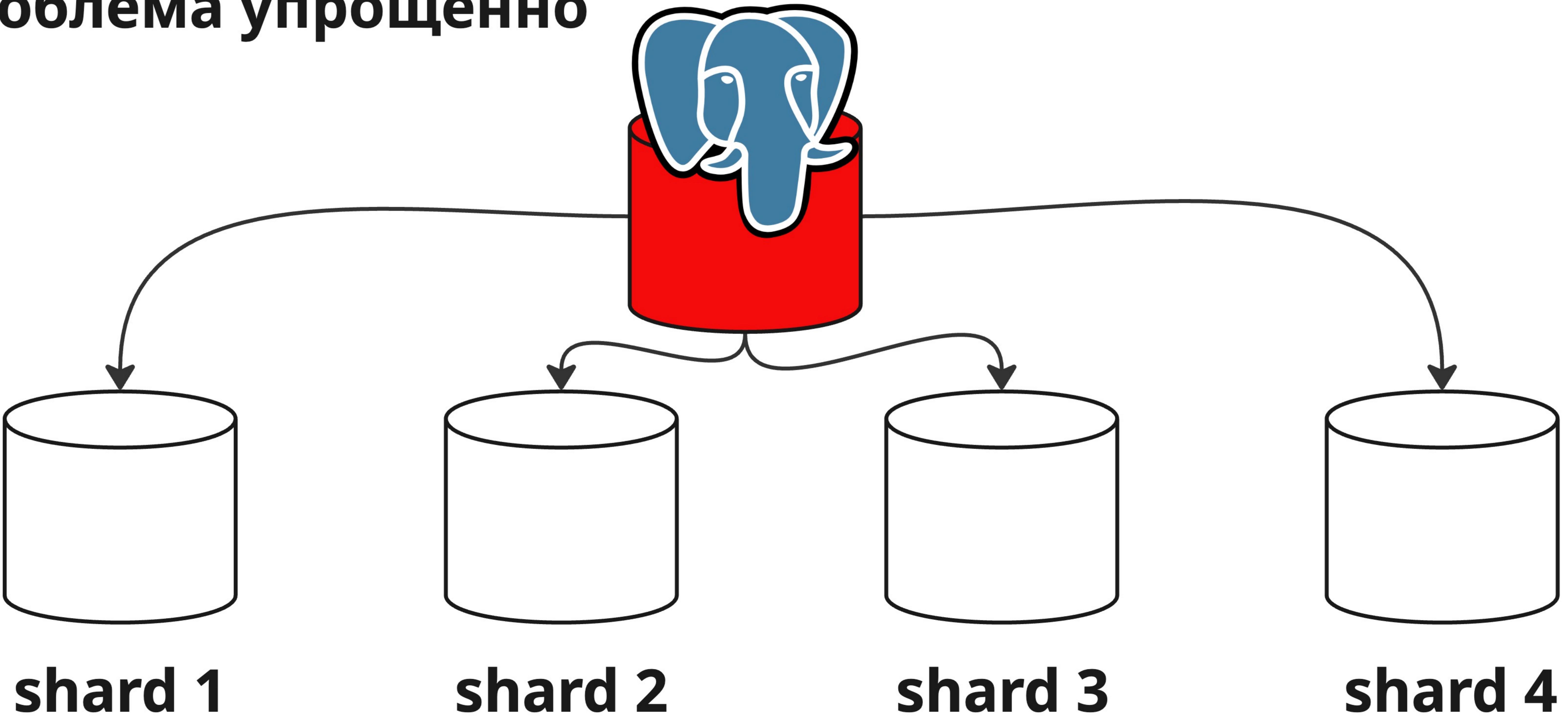
Когда шардируем



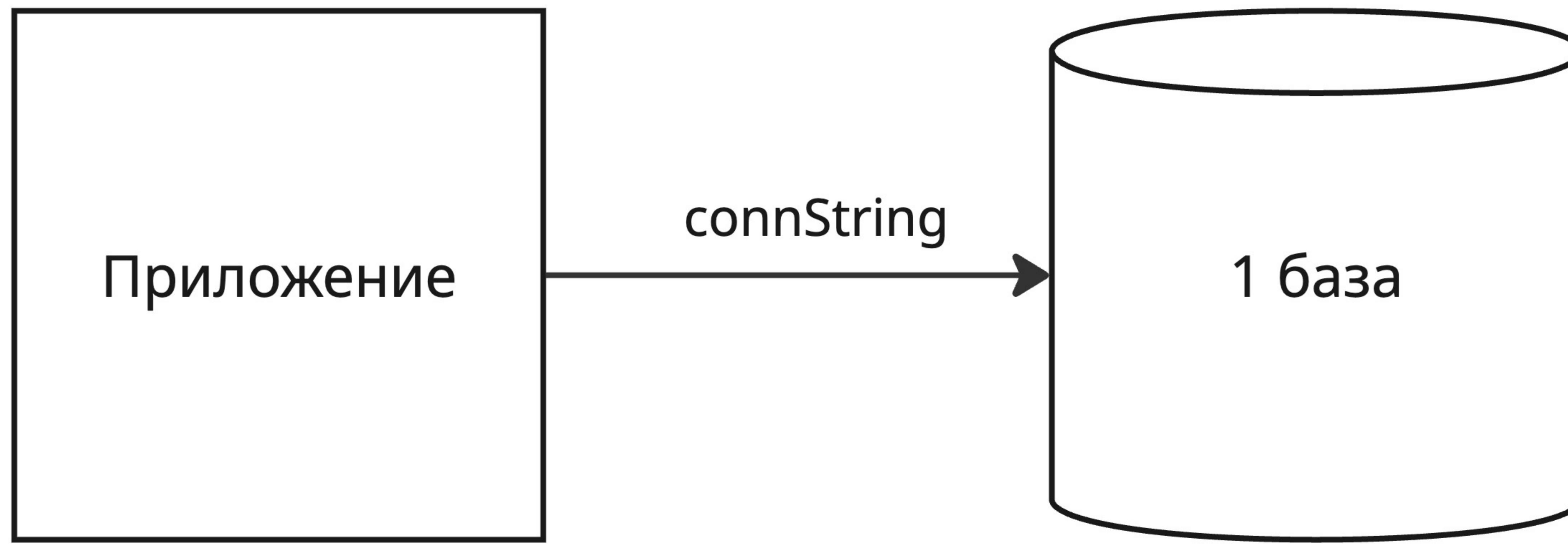
500+ GB

200k W/S

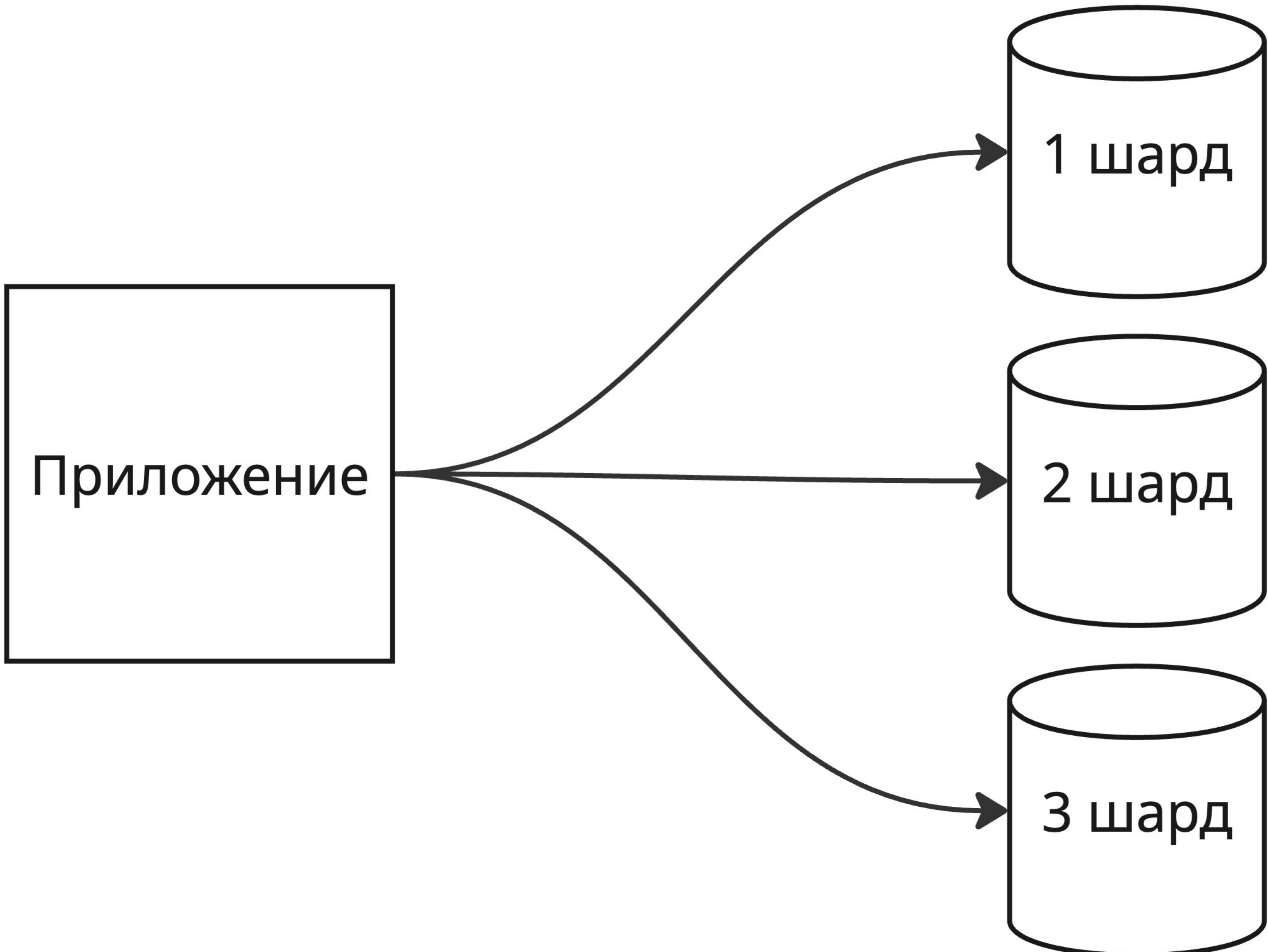
Проблема упрощенно



Как работало приложение до

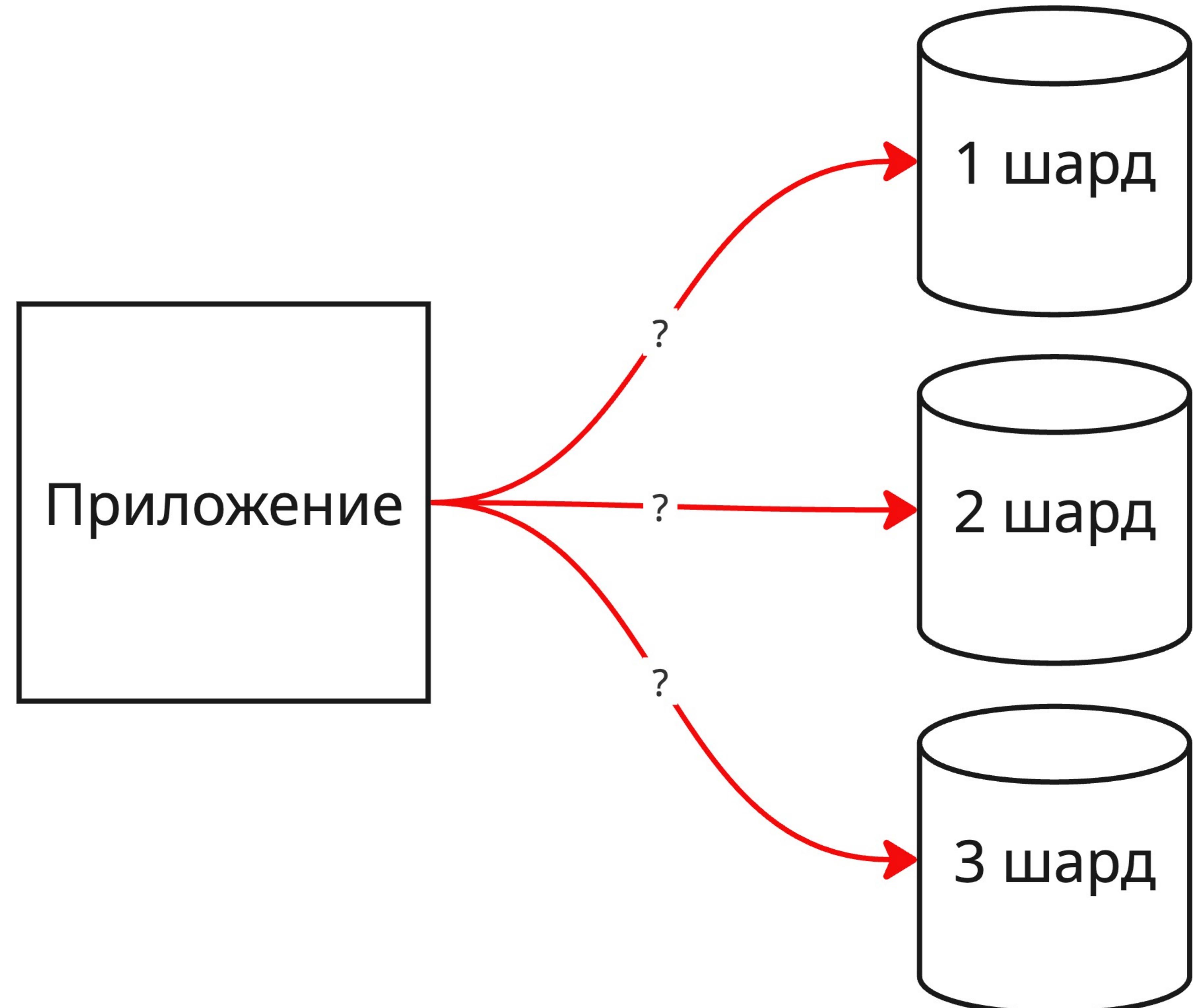


Как будет сейчас



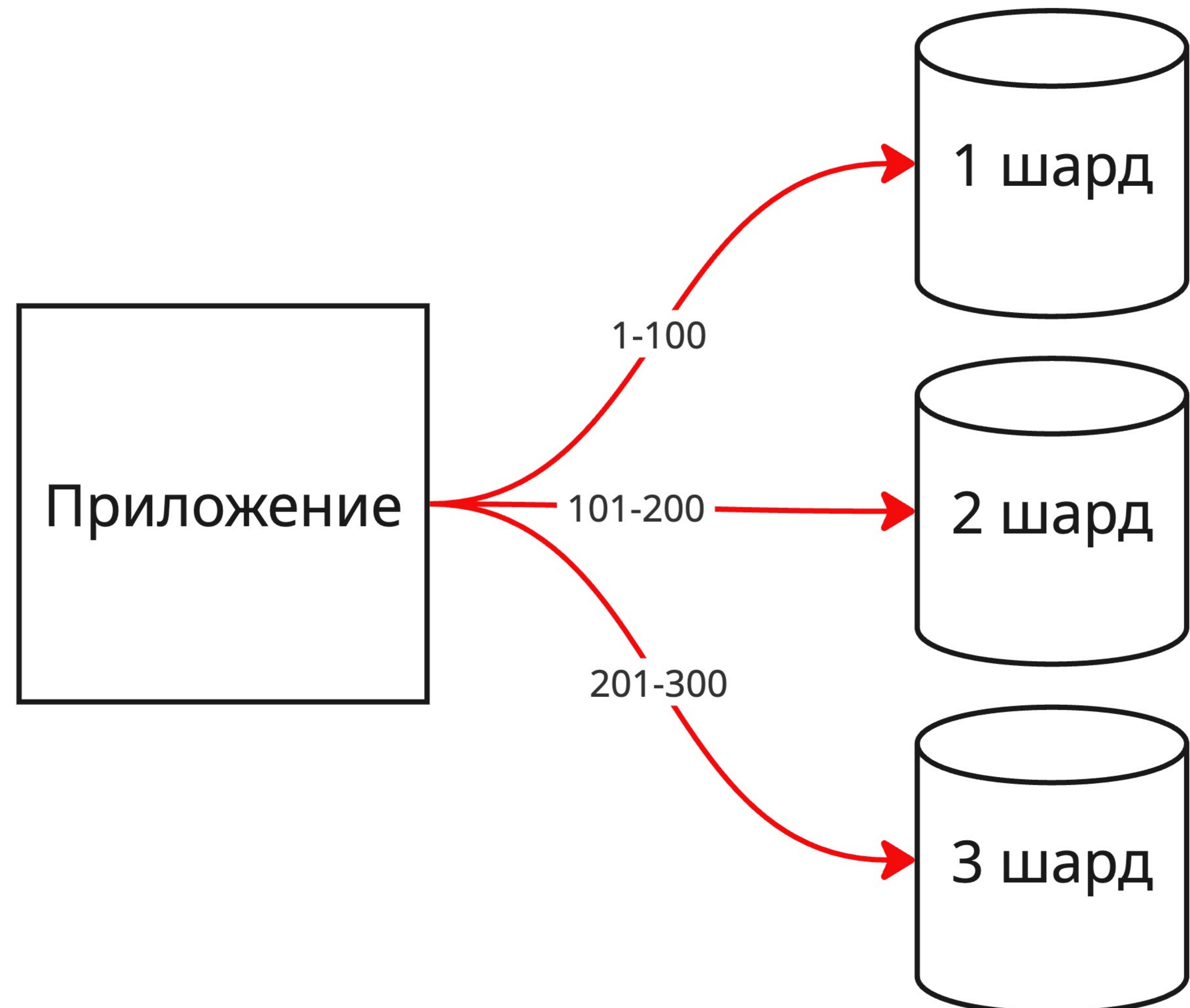
Как выбрать шард?

- Range-based sharding (по диапазону ключей)
- Hash-based sharding (по хэшу ключа)
- Directory-based sharding (шардирование через промежуточную таблицу)
- Consistent hashing (консистентный хэш)



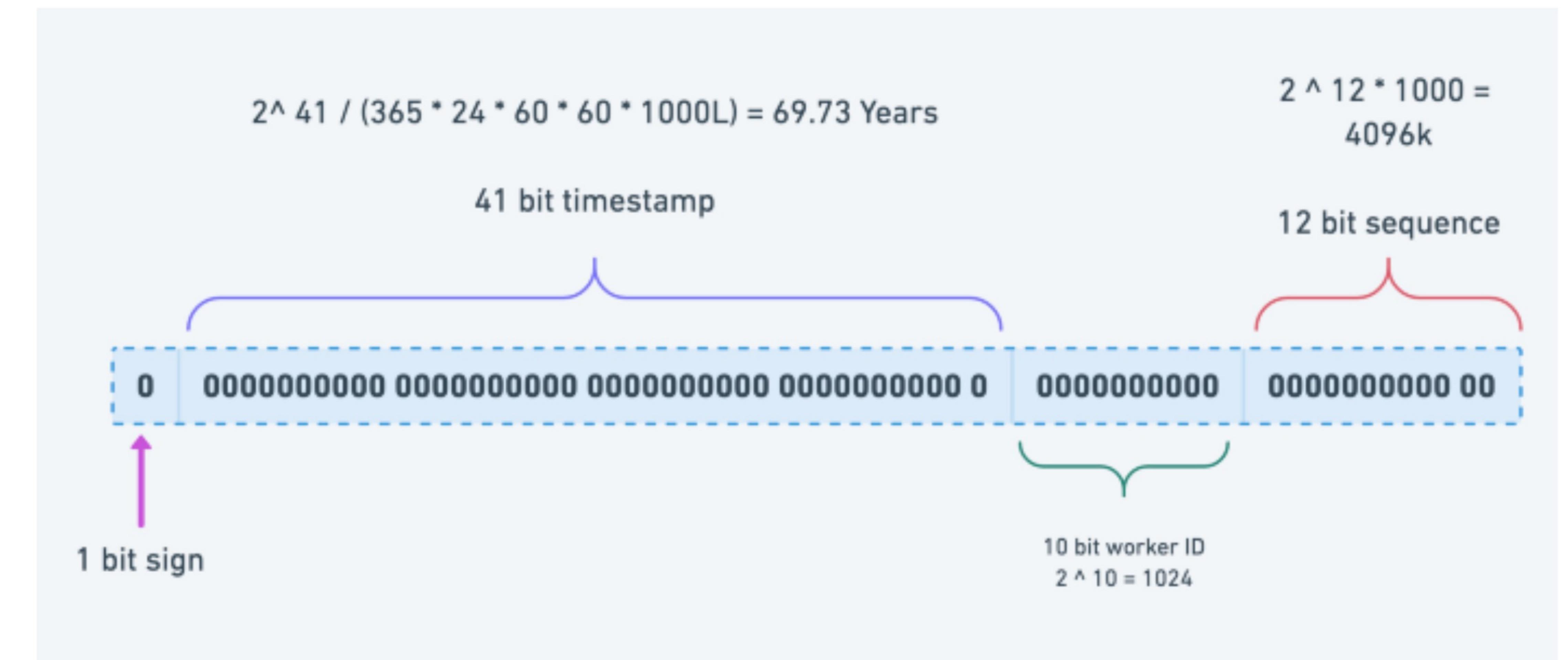
Range-based sharding

- Есть правило диапазонов где лежат данные
- На вставке генерируем **глобальный ID**
- Определяем в какой диапазон попадает ID
- Вставляем данные



Как сгенерировать глобальный ID

- Централизованный счётчик (отдельный сервис)
- Snowflake - 64 (timestamp + machine id + seq)
- ULID - base32 (ordered by timestamp)
- UUIDv7 - as ULID



<https://medium.com/prepster/unique-id-generators-4e3f898d0999>

Плюсы

- Легкость запуска и понимания как работает
- В системах заточенных на работу с диапазонами будет соблюдена локальность данных
- Ввод новых узлов не требует полного решардинга



Минусы

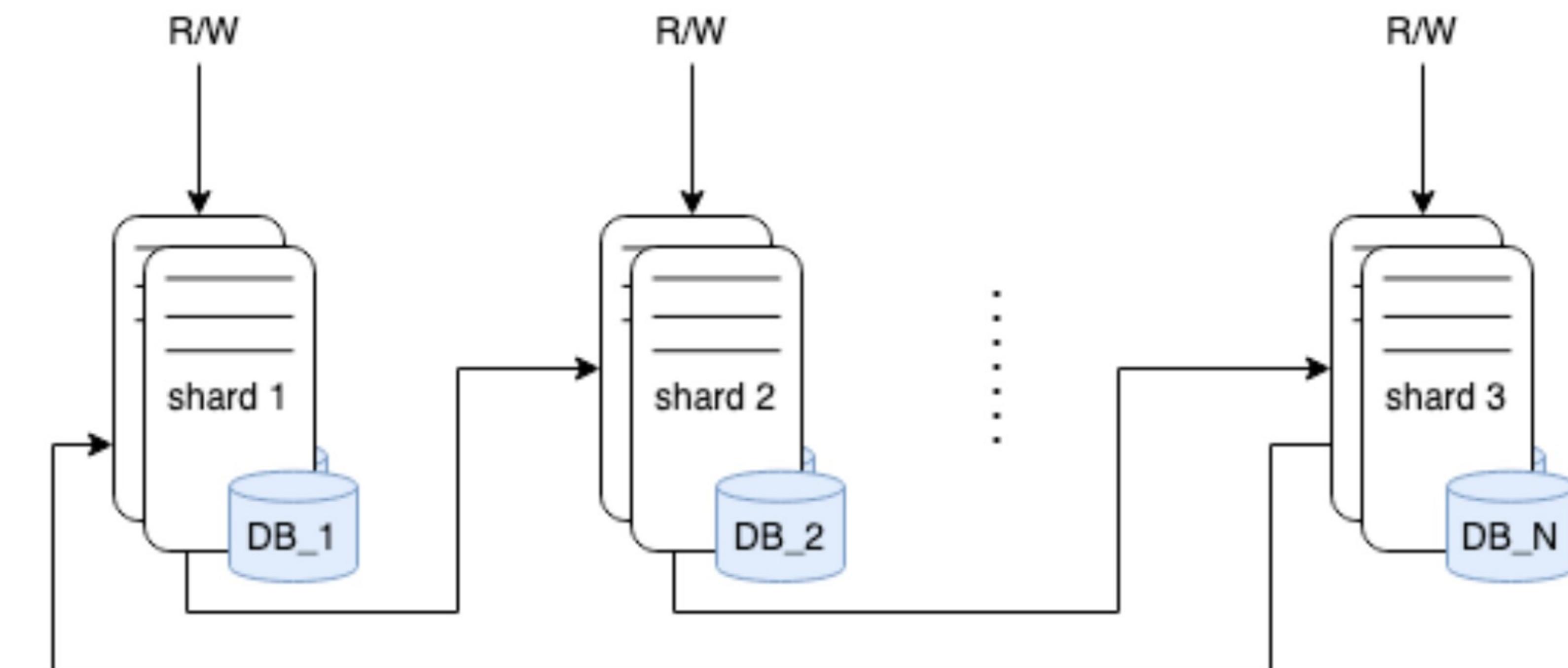
- Необходимо использовать основанные на timestamp **UUIDs (v1/v7)**
- Один из диапазонов может быть сильно перегружен при росте системы
- При монотонном увеличении ID старые диапазоны могут быть слабо использованы



```
1 CREATE TABLE clients (
2     id UUID DEFAULT gen_random_uuid(), -- pg17!
3     shard_key CHAR(1) GENERATED ALWAYS AS (
4         CASE
5             WHEN (substr(id::text, 1, 1) BETWEEN '0'
6                 AND '3') THEN '0'
7             WHEN (substr(id::text, 1, 1) BETWEEN '4'
8                 AND '7') THEN '1'
9             WHEN (substr(id::text, 1, 1) BETWEEN '8'
10                AND 'b') THEN '2'
11             ELSE '3'
12         END
13     ) STORED,
14     ...
15 )
```

Ввод новых узлов (нет решардинга)

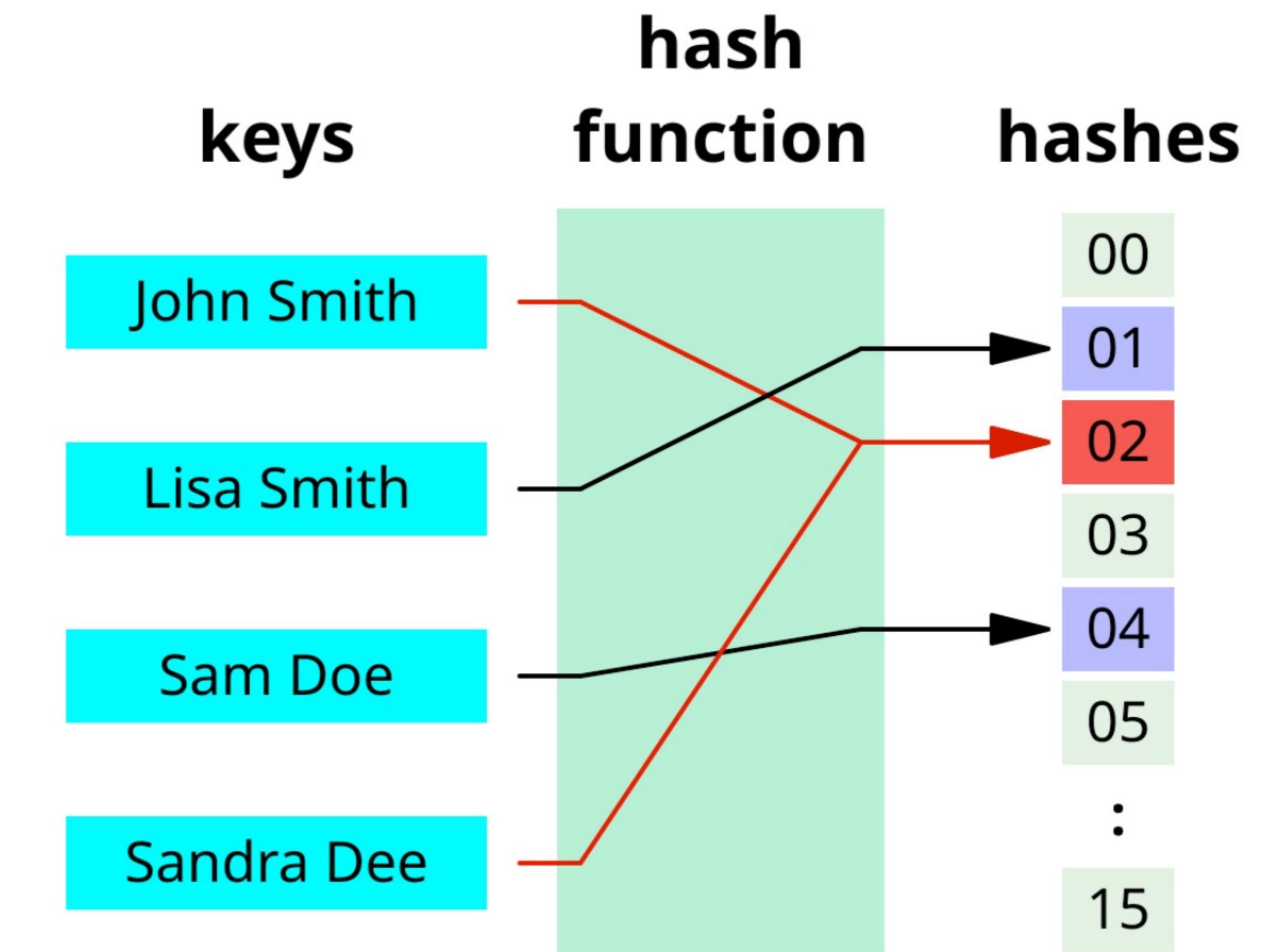
- Разбиваем данные на узлах (A: 0..999999, B: 1000000 .. ∞)
 - Shard A: 0 .. 499999
 - Shard C (новый): 500000 .. 999999
 - Shard B остается: 1000000 .. ∞
- Копируем заранее данные в Shard C
- Обновляем конфигурацию
- Удаляем дубли в шарде A



Не нужен полный решардинг, лишь один узел

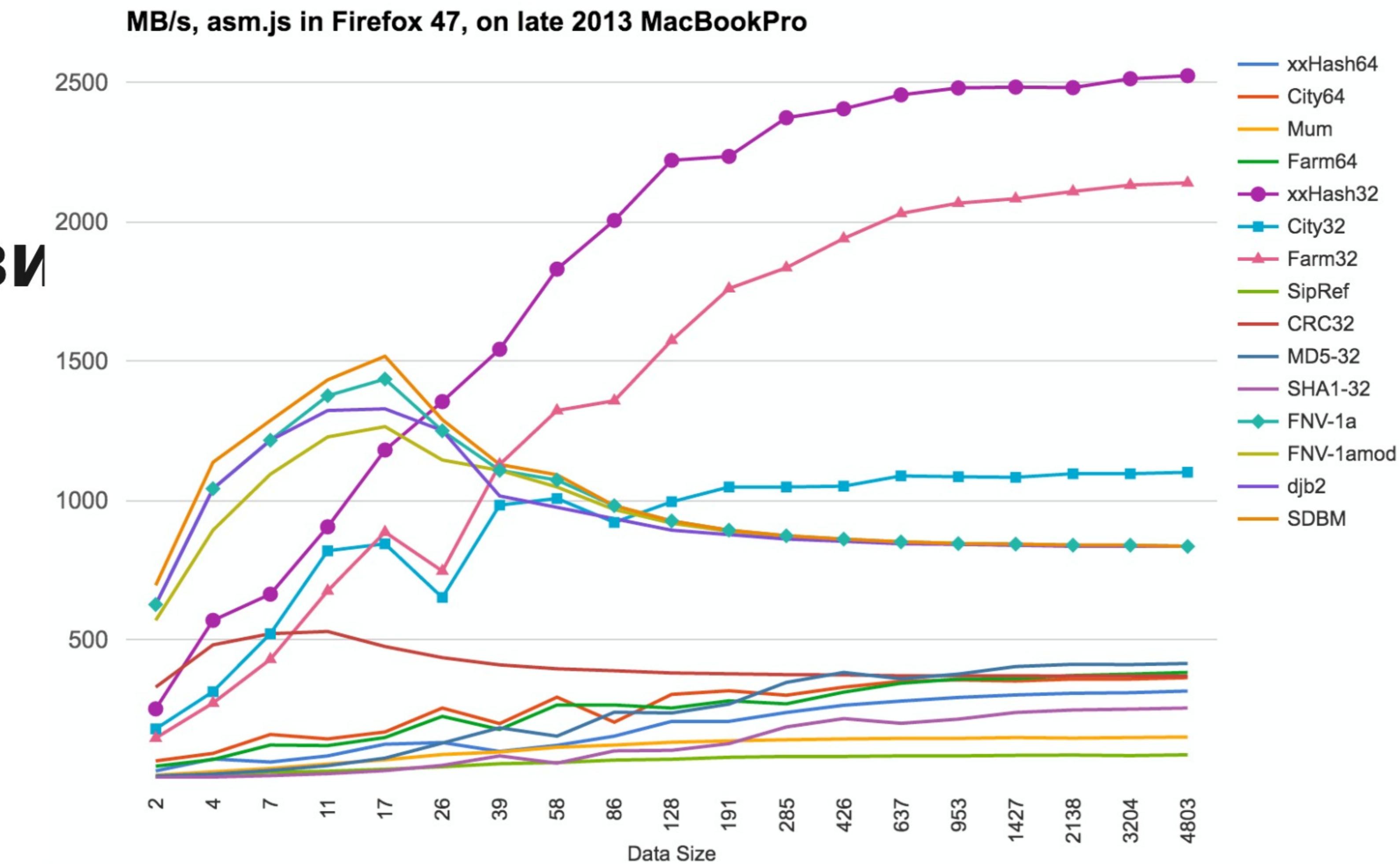
Hash-based sharding

- $\text{shardId} = \text{hash}(\text{key}) \% N$
 - Есть свойство (key) на основе которого генерируется hash (int) и от него берем остаток (mod) количества узлов (N)
 - Получаем идентификатор шарда
- Нужна хеш-функция с равномерным распределением
 - FNV-1a
 - MurmurHash3



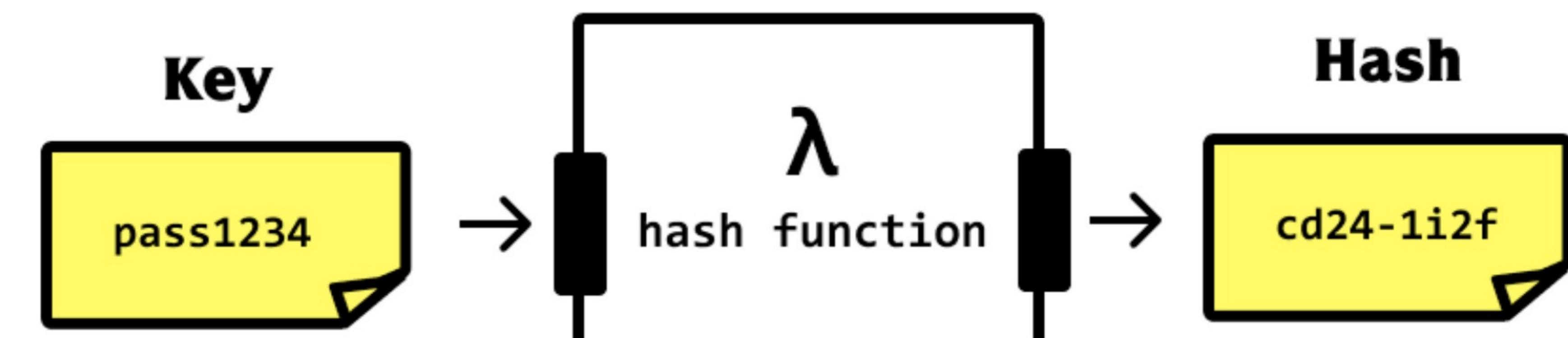
Как выбрать хеш функцию

- Равномерность распределения и коллизии
 - Отсутствие горячих шардов
 - Скорость
 - Хэш будет вычисляться на каждом INSERT/GET
 - Детерминированность
 - Для одного и того же ключа всегда должен возвращаться одинаковый хэш



Как выбрать ключ хеширования

- **Стабильность**
 - Ключ не должен меняться после создания записи
 - Пример хорошего: user_id, account_id, UUIDv7/ULID
- **Кардинальность**
 - Чем выше уникальность, тем равномернее распределяются записи по шард-базам.
- **Размер и тип**
 - int64 или короткая строка
- **Логика доступа**
 - Есть частые запросы по определённому полю
- **Из практики**
 - Если есть глобально уникальный int64/UUID, шардируйте по нему



Плюсы

- Легкость запуска и понимания как работает
- Легкость поддержки и отладки попадания данных
- Равномерное распределение в большинстве случаев

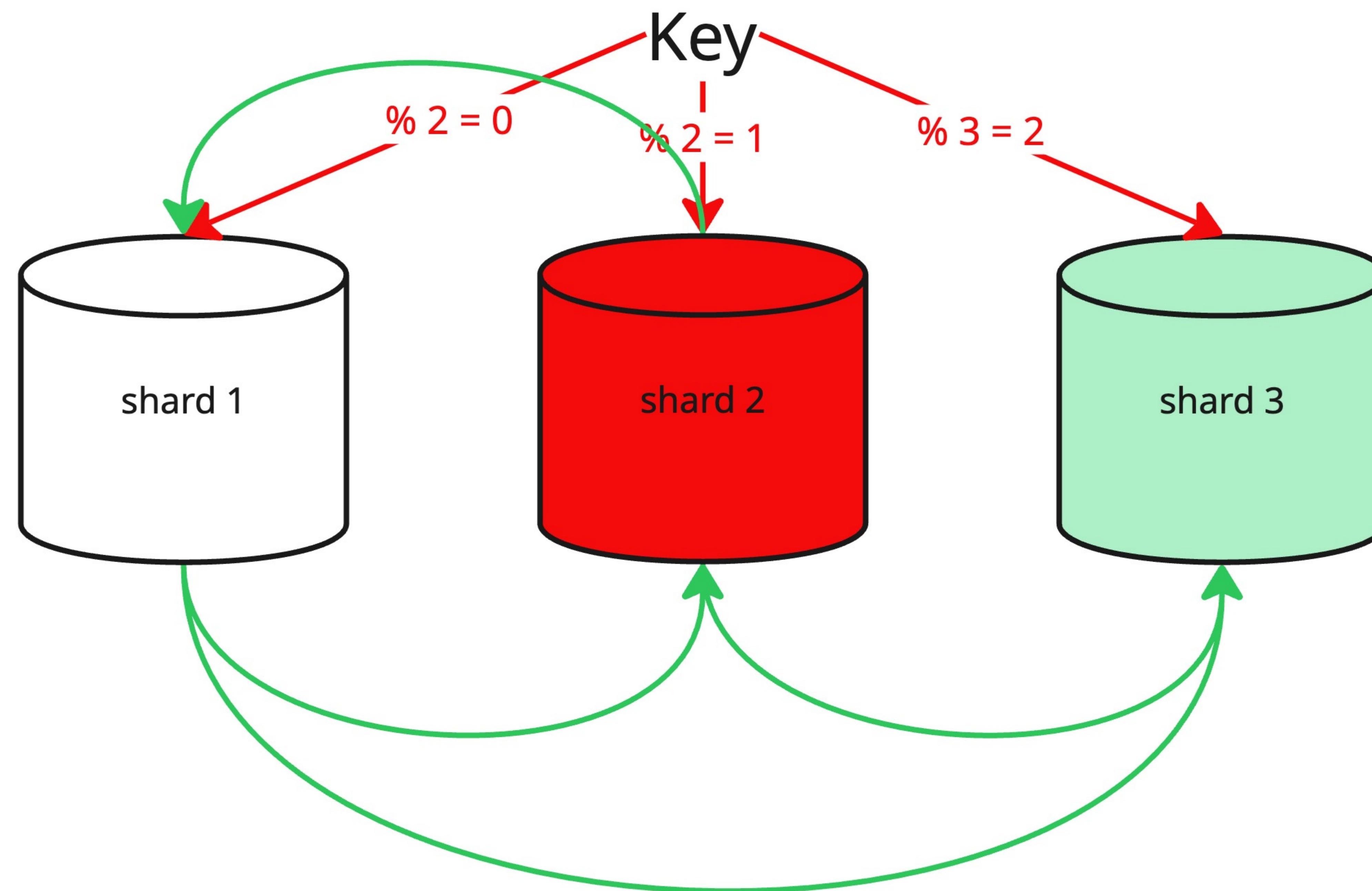


Минусы

- При вводе новых узлов придется перераспределять (решардировать базу целиком)
- При неудачном выборе количества шардов, функции или размера придется регулярно менять конфигурацию



Ввод новых узлов (полный решардинг)



To be continued

