# Виды репликации, введение в Patroni

PostgreSQL

REC

**Проверить, идет ли запись**

# Меня хорошо видно & слышно?

# Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Вопросы вижу в чате, могу ответить не сразу

Off-topic обсуждаем в чате Telegram

# Маршрут вебинара

- Задачи репликации
- Физическая репликация
- Логическая репликация
- Примеры применения
- Рефлексия

# Цели вебинара

## К концу занятия вы сможете

1. Настраивать репликации

2. Выбирать оптимальный вид репликации

3. Обеспечивать высокую доступность / масштабируемость данных

# Задачи репликации

# Репликация

Это процесс синхронизации нескольких копий одного объекта (например, нескольких кластеров PostgreSQL на разных серверах).

Решает задачи:

- отказоустойчивости;
- масштабируемости.

# Зачем нужна репликация ???

# Зачем нужна репликация ???

1. Высокая доступность. Бэкап это хорошо, но нужно время на его развертывание.
2. Бэкап лучше делать с реплики, а не мастера.
3. Что делать, когда закончились физические ядра и память у сервера? Горизонтально масштабировать!!!
4. Геораспределение нагрузки

# Виды репликации

| Feature | Shared Disk | File System Repl. | Write-Ahead Log Shipping | Logical Repl. | Trigger-Based Repl. | SQL Repl. Middle-ware | Async. MM Repl. | Sync. MM Repl. |
|---|---|---|---|---|---|---|---|---|
| Popular examples | NAS | DRBD | built-in streaming repl. | built-in logical repl., pglogical | Londiste, Slony | pgpool-II | Bucardo | |
| Comm. method | shared disk | disk blocks | WAL | logical decoding | table rows | SQL | table rows | table rows and row locks |
| No special hardware required | | • | • | • | • | • | • | • |
| Allows multiple primary servers | | | | • | | • | • | • |
| No overhead on primary | • | | • | • | | • | | |
| No waiting for multiple servers | • | | with sync off | with sync off | • | | • | |
| Primary failure will never lose data | • | • | with sync on | with sync on | | • | | • |
| Replicas accept read-only queries | | | with hot standby | • | • | • | • | • |
| Per-table granularity | | | | • | • | | • | • |
| No conflict resolution necessary | • | • | • | | • | • | | • |

PostgreSQL: Documentation: 18: 26.1. Comparison of Different Solutions

# Физическая репликация

# Физическая репликация

1. Записи WAL передаются на реплику и применяются:
• поток данных только в одну сторону
• реплицируется кластер целиком, выборочная реплика невозможна

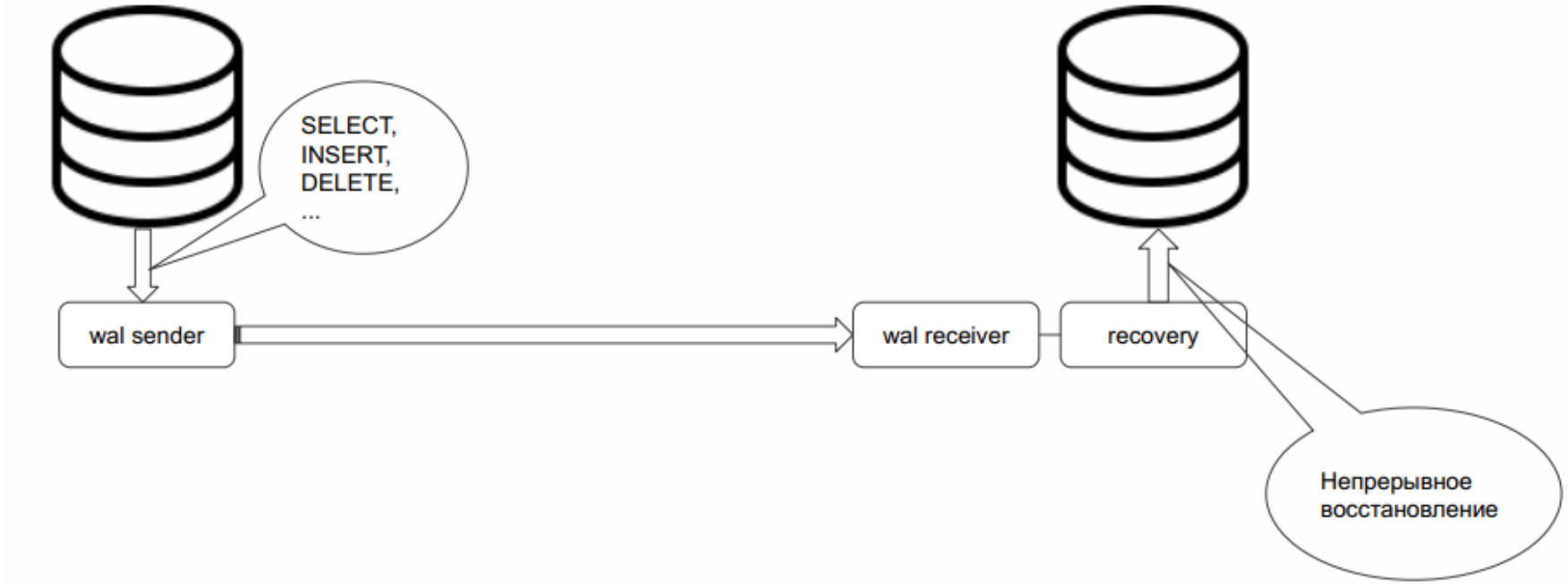
2. Реплика — точная копия мастера:
• одна и та же основная версия сервера
• полностью совместимые архитектуры и платформы

3. Реплика доступна только для чтения.

# Использование физической репликации

1. Горячий резерв для высокой доступности
2. Балансировка OLTP-нагрузки
3. Реплика для отчетов
4. Несколько реплик и каскадная репликация
5. Отложенная репликация

# Использование физической репликации



# Возможности реплики

**Допускаются**

- запросы на чтение данных (select, copy to, курсоры)
- установка параметров сервера (set, reset)
- управление транзакциями (begin, commit, rollback...)
- создание резервной копии (pg_basebackup)

# Возможности реплики

**Не допускаются**

- любые изменения (insert, update, delete, truncate, nextval...)
- блокировки, предполагающие изменение (select for update...)
- команды DDL (create, drop...), в том числе создание временных таблиц
- команды сопровождения (vacuum, analyze, reindex...)
- управление доступом (grant, revoke...)
- триггеры и пользовательские блокировки

# Виды физической репликации

1. **Асинхронная репликация** - последовательное выполнение транзакций на всех репликах.

+ хорошая пропускная способность

- появляется время отклика в течении которого отдельные реплики могут быть фактически неидентичными

synchronous_commit = off

# Виды физической репликации

2. **Синхронная репликация** – коммит на реплике должен быть доступен до того, как он будет доступен на мастере.

+ в момент отказа получают абсолютно идентичные мастер и реплику (могут потеряться последние транзакции которые еще не были зафиксированы, но мастер и реплика в любом случае идентичны)

- проблемы с производительностью: Primary ждет ответа от реплики об удачном коммите
- в случаях возникновения проблем с репликой Primary останавливается и не может работать дальше до тех пор пока реплика не будет восстановлена, либо до тех пор пока эта реплика не будет удалена из списка синхронных реплик на Primary сервере.

# Синхронная репликация

synchronous_standby_names = (список реплик)

synchronous_commit = on | remote_write | remote_apply

remote_write - дожидается только ответа о том, что информация дошла до реплики, но не факт, что произошла запись на диск

on - подтверждает, что произошла запись на диск в WAL файл

remote_apply - дает подтверждение тому, что запись применена в базе

# Куда смотреть

**На Мастере:**

Процесс можно отслеживать в представлении *pg_current_wal_lsn()*,

остальные — представление *pg_stat_replication*.

Если используется слот репликации, то информацию о нем можно получить из представления *pg_replication_slots*.

**На Реплике:**

В представлении *pg_stat_wal_receiver* и с помощью функций

*pg_last_wal_receive_lsn*() и *pg_last_wal_replay_lsn*().

# Параметры конфигурации

**max_worker_processes** - должно быть таким же или большим, чем на Мастер сервере, иначе нельзя будет делать запросы на Реплику.
**wal_level** - replica или logical
**archive_mode** - on|off
**wal_receiver_status_interval** – частота передачи сигнала от реплики мастеру (по умолчанию 10 с.)
**wal_retrieve_retry_interval** – время ожидания реплики, прежде чем повторить попытку получить WAL (5 с.)
**recovery_min_apply_delay** – время задержки на восстановление
**max_replication_slots** – количество слотов

# Создание физической реплики

Начиная с версии 10, все необходимые настройки уже присутствуют по умолчанию:
- wal_level = replica;
- max_wal_senders
- разрешение на подключение в pg_hba.conf по протоколу репликации.

Создадим 2 кластер
**$ pg_createcluster -d /var/lib/postgresql/17/main2 17 main2**
Удалим оттуда файлы
**$ rm -rf /var/lib/postgresql/17/main2**
Сделаем бэкап нашей БД. Ключ -R создаст заготовку управляющего файла recovery.conf (запуск на вторичном сервере, если другой хост то –h)
**$ pg_basebackup -p 5432 -R -D /var/lib/postgresql/17/main2**
Стартуем кластер
**$ pg_ctlcluster 17 main2 start**
Смотрим как стартовал
**$ pg_lsclusters**

# Повышение Реплики до Мастера

Перевод реплики в состояние мастера
**$ pg_ctlcluster 17 main2 promote**

*Что произойдет?*

# Повышение Реплики до Мастера

Перевод реплики в состояние мастера
**$ pg_ctlcluster 17 main2 promote**

Получим 2 разных независимых сервера.

# Повышение Реплики до Мастера

**Причины**

• плановое переключение (switchover): останов основного сервера для проведения технических работ

• аварийное переключение (failover): переход на реплику из-за сбоя основного сервера

**Процедура**

• убедиться, что мастер остановлен

• переключение вручную

• автоматическое переключение отсутствует, а в Patroni есть ☺

# Подключение старого Мастера

Простое подключение бывшего мастера — не работает:
• проблема потери записей WAL, не попавших на реплику из-за задержки

Восстановление «с нуля» из резервной копии:
• на месте бывшего мастера разворачивается абсолютно новая реплика
• процесс занимает много времени

Или воспользоваться утилитой pg_rewind

# Логическая репликация

# Логическая репликация

Встроенная логическая репликация доступна с 10 версии PostgreSQL.
Для более ранних версий аналогичный функционал доступен в расширении **pg_logical**.

Для передачи логических изменений (на уровне строк) используется протокол репликации.

Другой способ организации логической репликации состоит в использовании триггеров для перехвата изменений, помещения этой информации в очередь событий и передача ее на другой сервер. Такой способ, однако, менее эффективен, и уходит в прошлое (Slony-I).

# Отличия от физической репликации

По сути тут нет понятий Мастер и Реплика.

На сервере создается публикация, на которую другие серверы могут подписываться.

Выборочная репликация отдельных таблиц.

Подписчику передается информация об изменениях строк в таблицах: двоичная совместимость не требуется.

# Публикующий сервер

Выдает изменения данных построчно в порядке их фиксации (реплицируются команды INSERT, UPDATE, DELETE), в 11 версии добавили TRUNCATE.

Возможна начальная синхронизация.

Всегда используется слот логической репликации.

Применение изменений происходит без выполнения команд SQL и связанных с этим накладных расходов на разбор и планирование, что уменьшает нагрузку на подписчика.

Требуется установка уровня журнала **wal_level = logical**.

# Подписчики

Получают и применяют изменения: без разбора, трансформаций и планирования — сразу выполнение.

Возможны конфликты с локальными данными.

# Назначение логической репликации

1. Консолидация и общие справочники
2. Обновление основной версии сервера
3. Мастер-мастер (в будущем)

# В новых версиях

С 15 версии можно фильтровать записи и колонки.

С 16 версии стало возможным логическую репликацию настроить таким образом, что можно подписаться на публикацию на физической реплике.

# Создание логической реплики

Используем два сервера и настроим логическую репликацию.
**ALTER SYSTEM SET wal_level = logical;**
Рестартуем кластер
**$ sudo pg_ctlcluster 17 main restart**

На первом сервере создаем публикацию:
**\c db_name**
**CREATE TABLE test(i int);**
**CREATE PUBLICATION test_pub FOR TABLE test;**
**\dRp+**
**\password**

# Создание логической реплики

создадим подписку на втором экземпляре
**\c db_name**

```
CREATE TABLE test(i int);

CREATE SUBSCRIPTION test_sub
CONNECTION 'host=localhost port=5432 user=postgres password=test
dbname=replica' PUBLICATION test_pub WITH (copy_data = false);

\dRs
```

# Создание логической реплики

состояние подписки

```
SELECT * FROM pg_stat_subscription \gx
```

удаление публикации и подписки

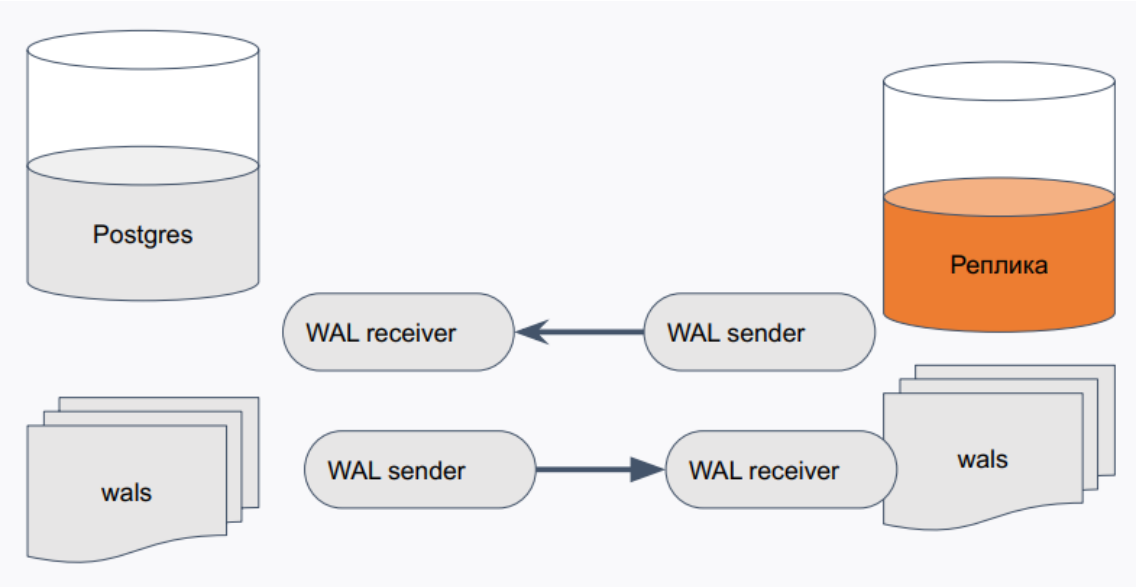
```
drop publication test_pub;
drop subscription test_sub;
```

найти проблему можно в логах

```
$ tail /var/log/postgresql/postgresql-16-main2.log
```

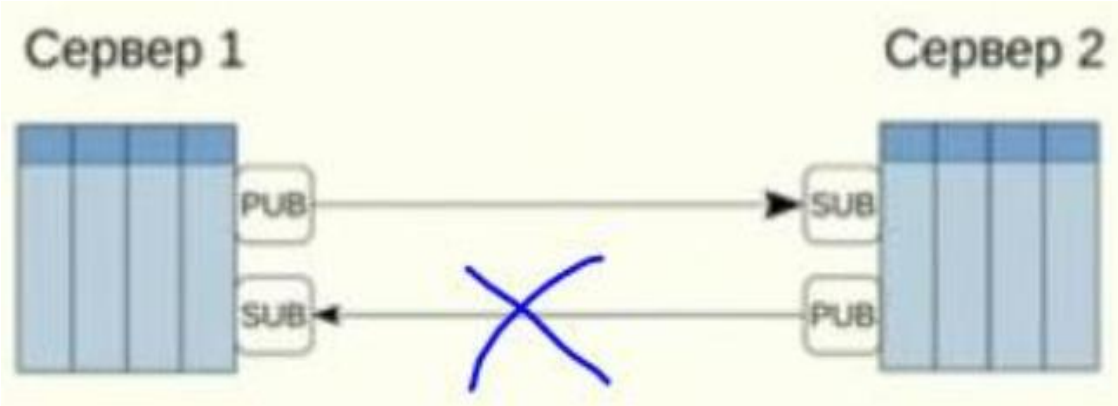
# Примеры применения

# Мастер – мастер репликация




- Любой сервер - полная копия доступная для записи
- Высока вероятность конфликтов
- Сложность в администрировании
- Сложность восстановления

# Мастер – мастер репликация



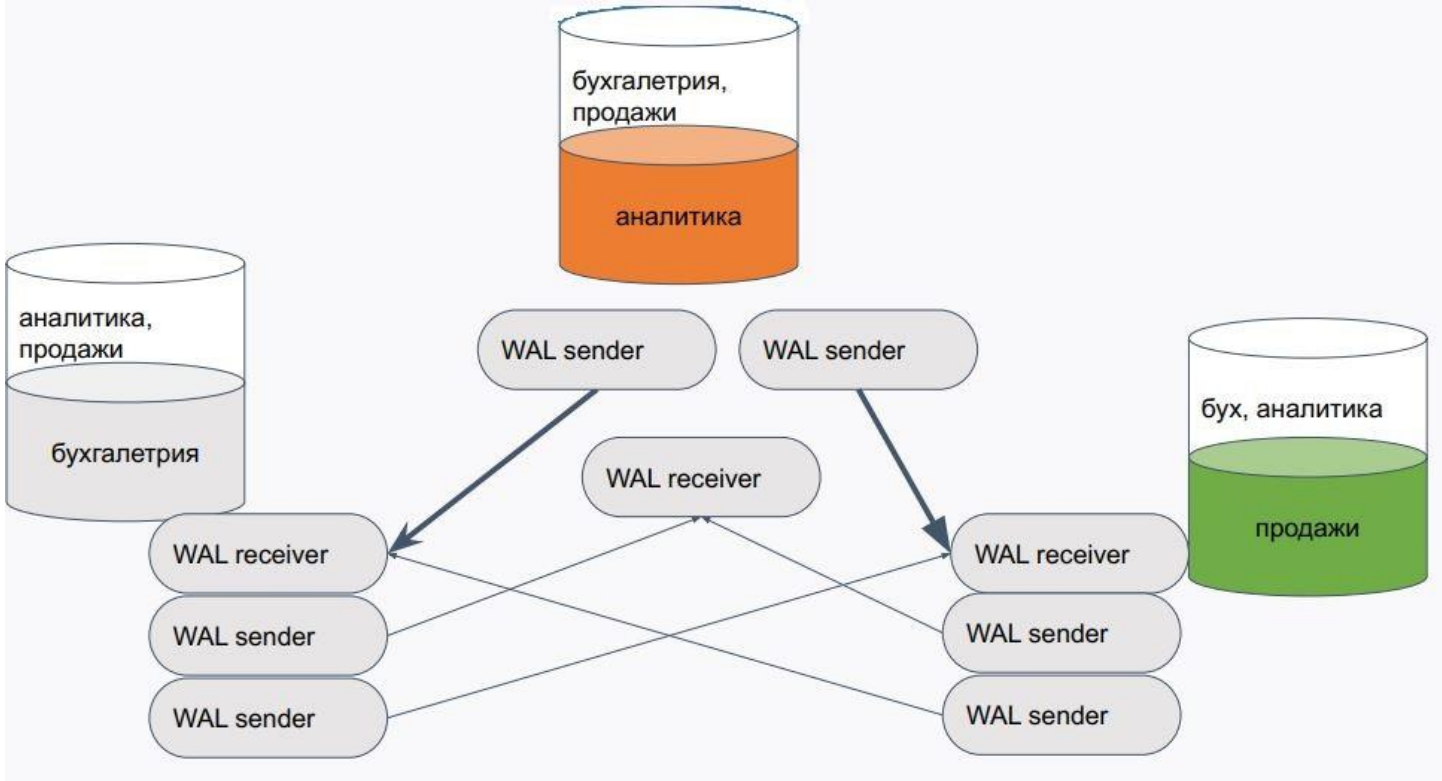

- В ванильном PostgreSQL нельзя одновременно создать на таблицу подписку и публикацию на одном сервере и зеркально сделать публикацию с подпиской на другом сервере.
- Запрещено создание подписки, порождающей цикл.

# Мастер – мастер репликация


- В ванильном PostgreSQL нельзя БЫЛО ДО 16-й версии.
- Теперь для создания этого достаточно указать опцию origin = none.

**CREATE SUBSCRIPTION test_sub**
**CONNECTION 'host=localhost port=5432 user=postgres password=test**
**dbname=replica'  PUBLICATION test_pub WITH (copy_data = false, origin = none);**

# Мастер мастер репликация: Percona eXtraDB Cluster



# Мастер – мастер репликация

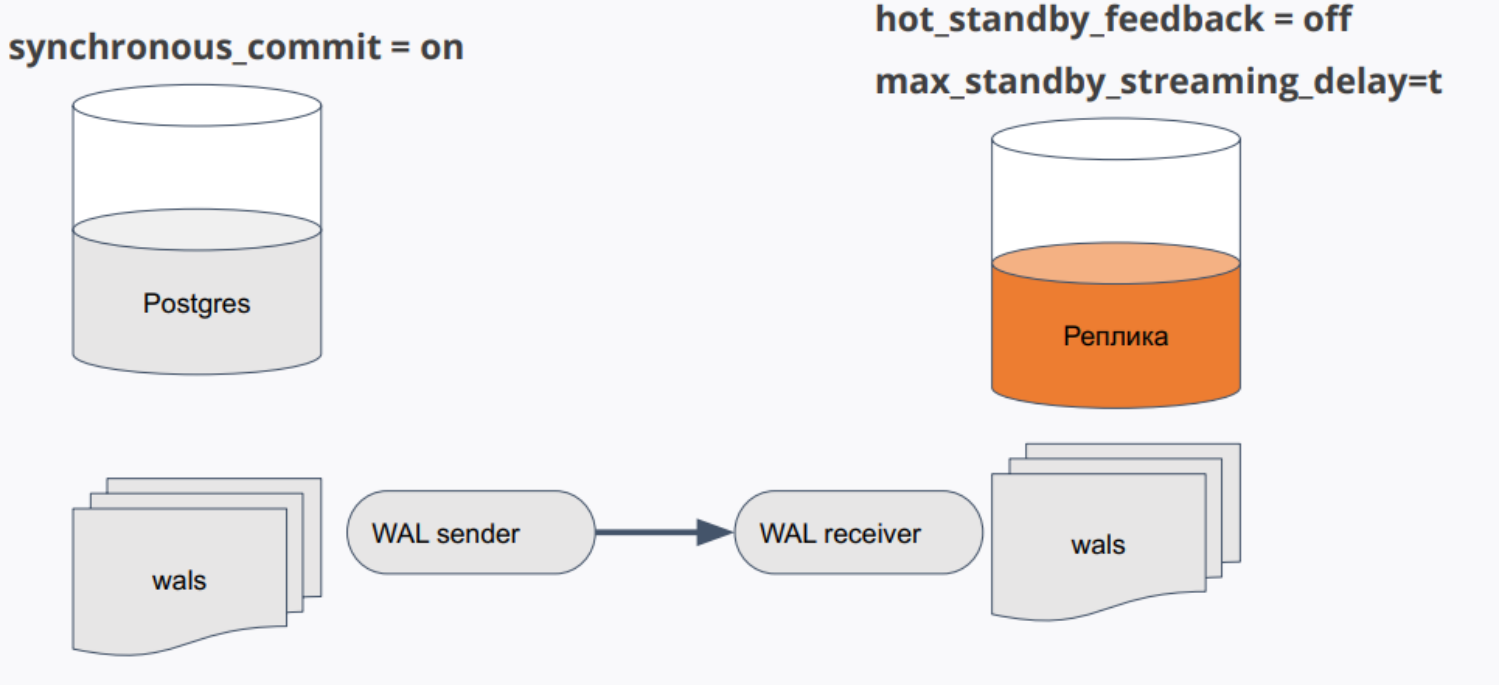
1. MultiMaster от компании Postgres Pro
   (расширение для СУБД Postgres Pro Enterprise)

2. Postgres Plus xDB Replication Server with MultiMaster Replication (MMR) от компании EnterpriseDB

3. Bucardo

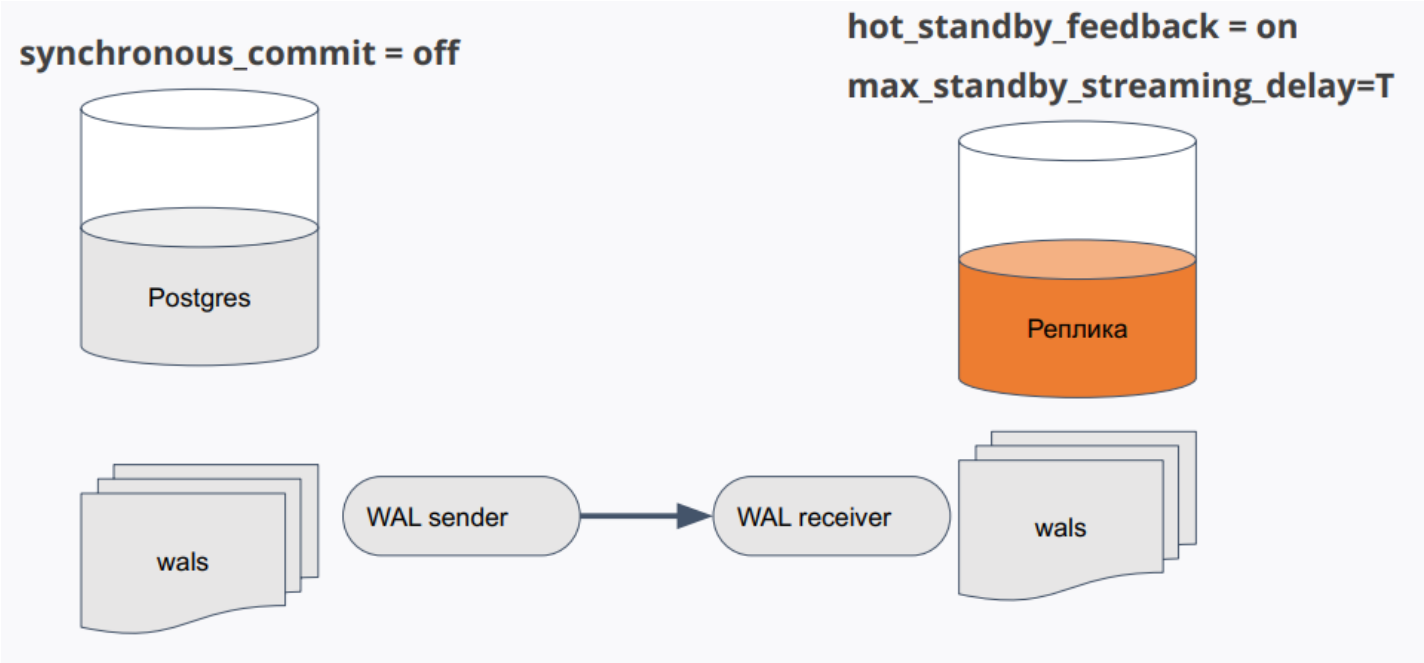
…

# Горячий резерв for high availability



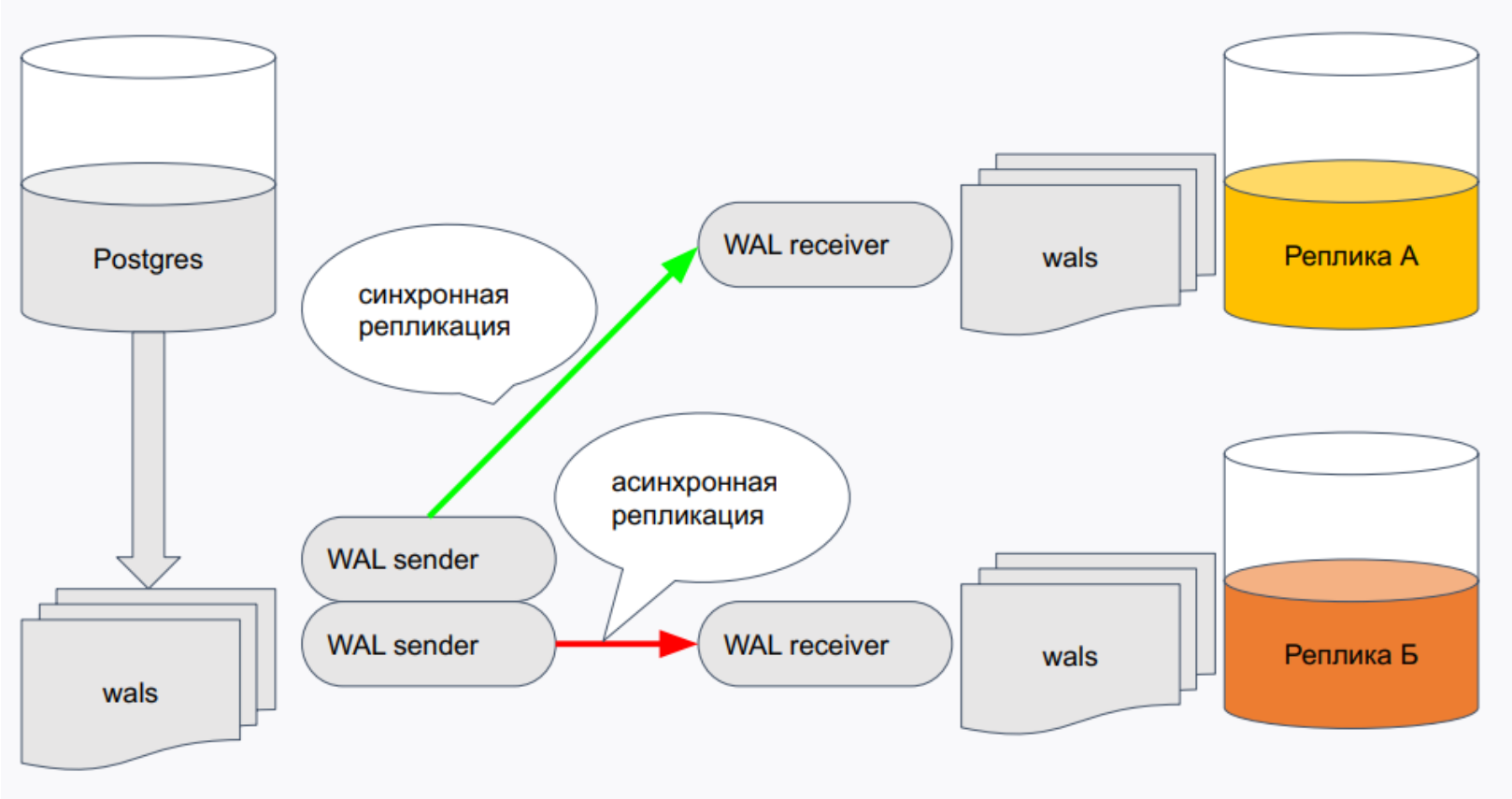

- синхронная репликация, реплика должна максимально соответствовать мастеру
- запросы к реплике возможны, но не приоритетны

# Балансировка OLTP

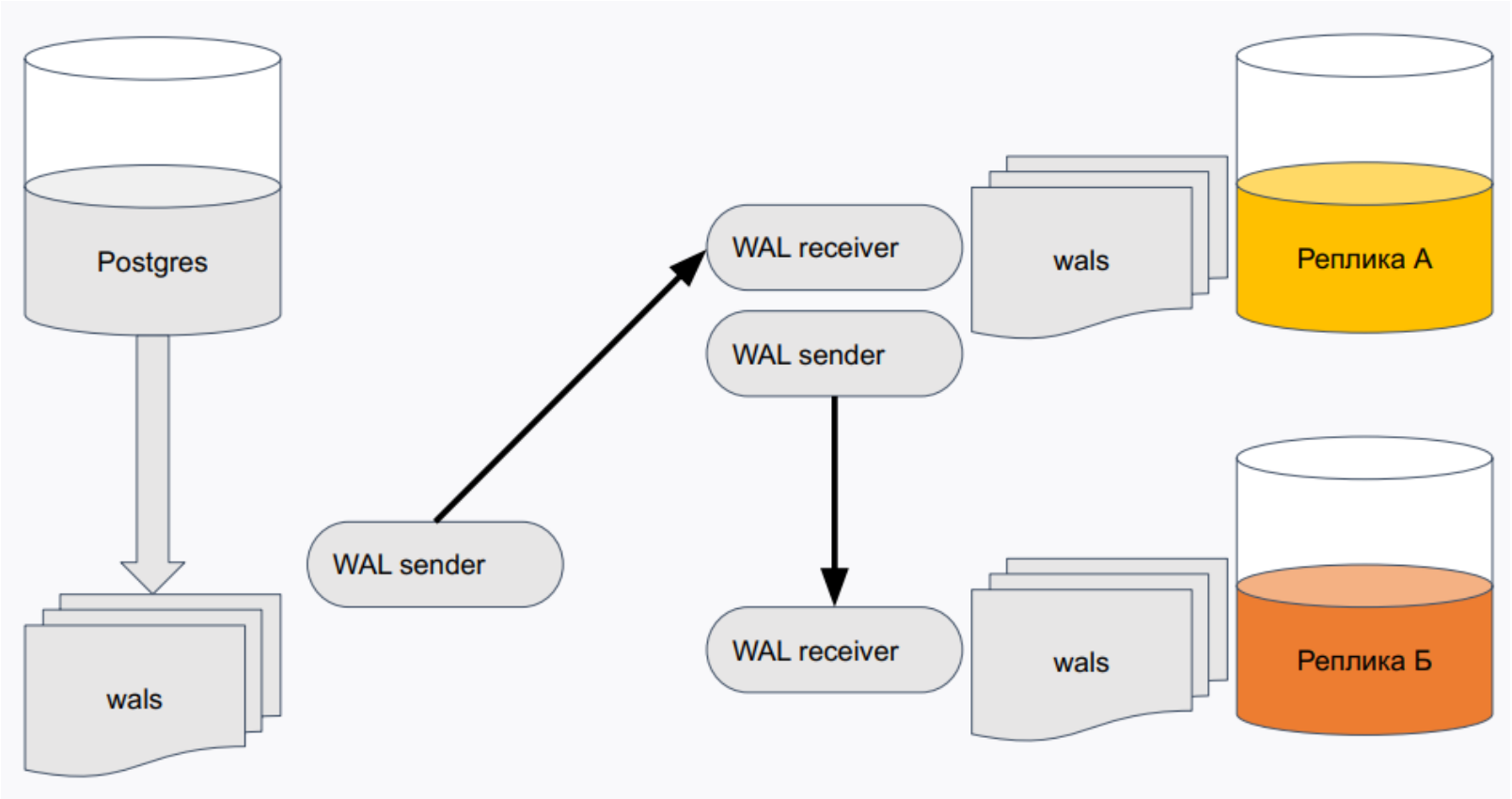



- много коротких запросов
- запросы на реплике должны отрабатываться
- долгие запросы повлияют на мастер, но в идеале их не должно быть

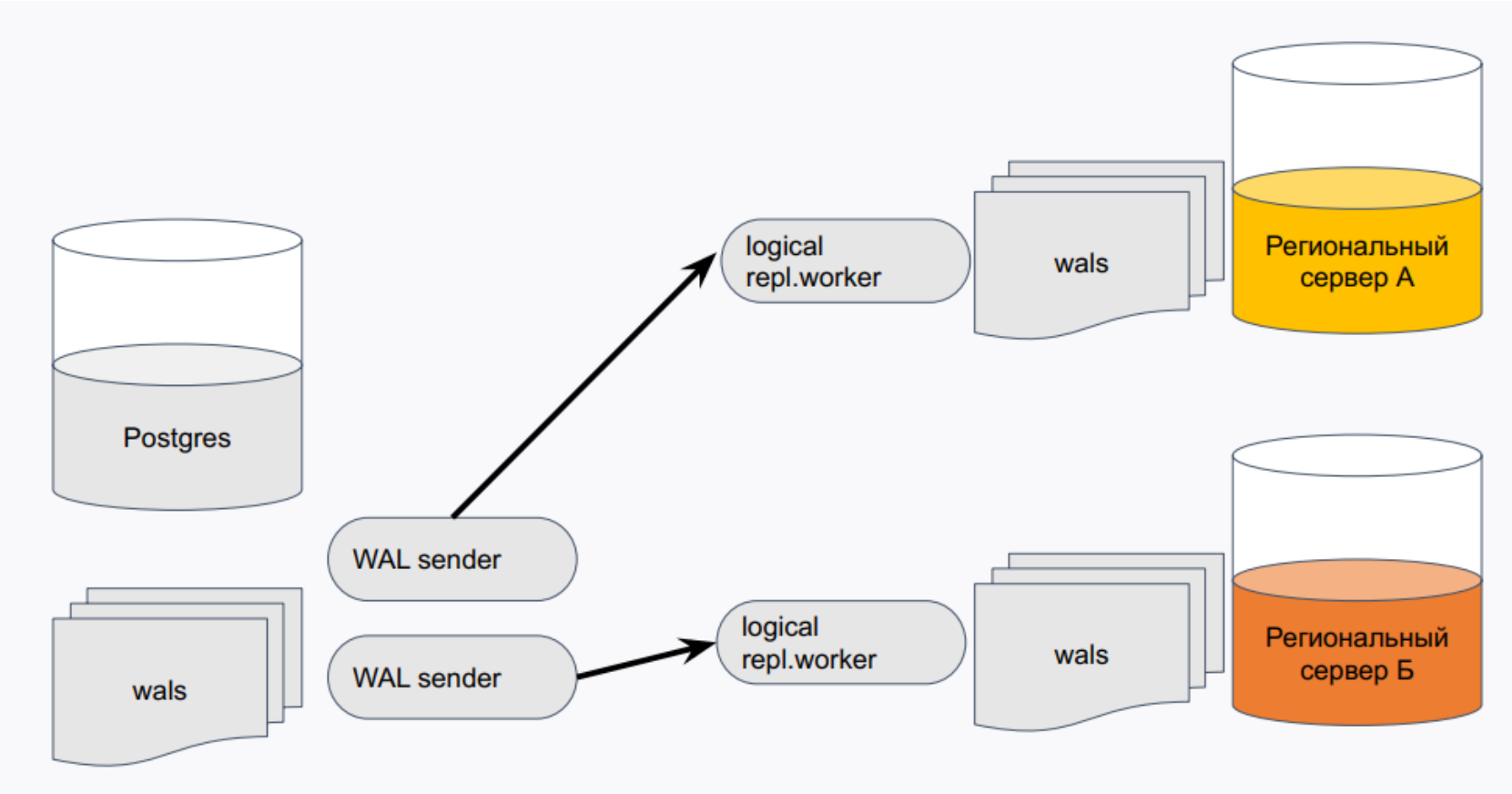
# Горизонтальное масштабирование с HA



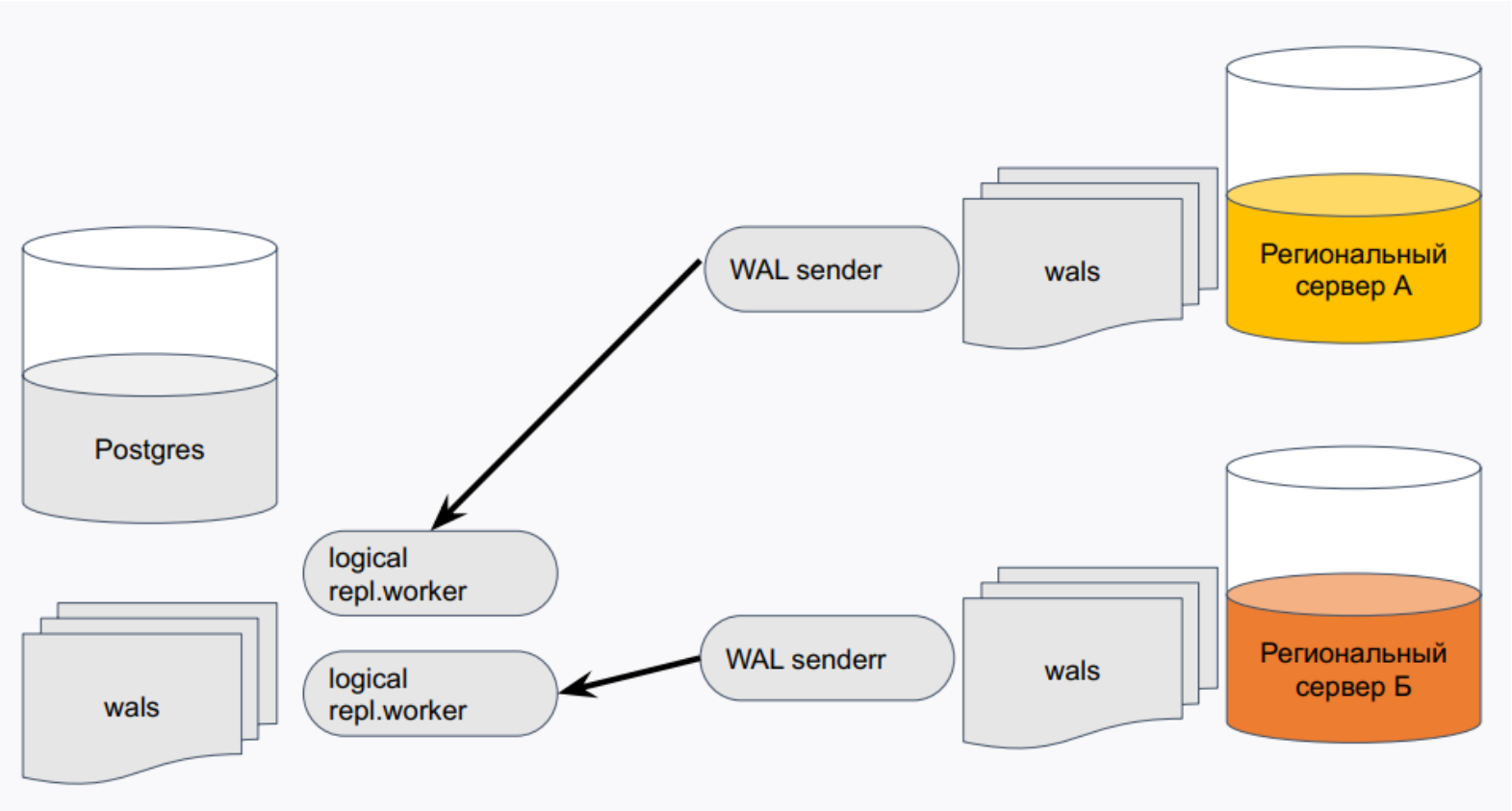
# Каскадная репликация



# Рассылка справочников



# Объединение данных



# Вопросы

+ если есть вопросы

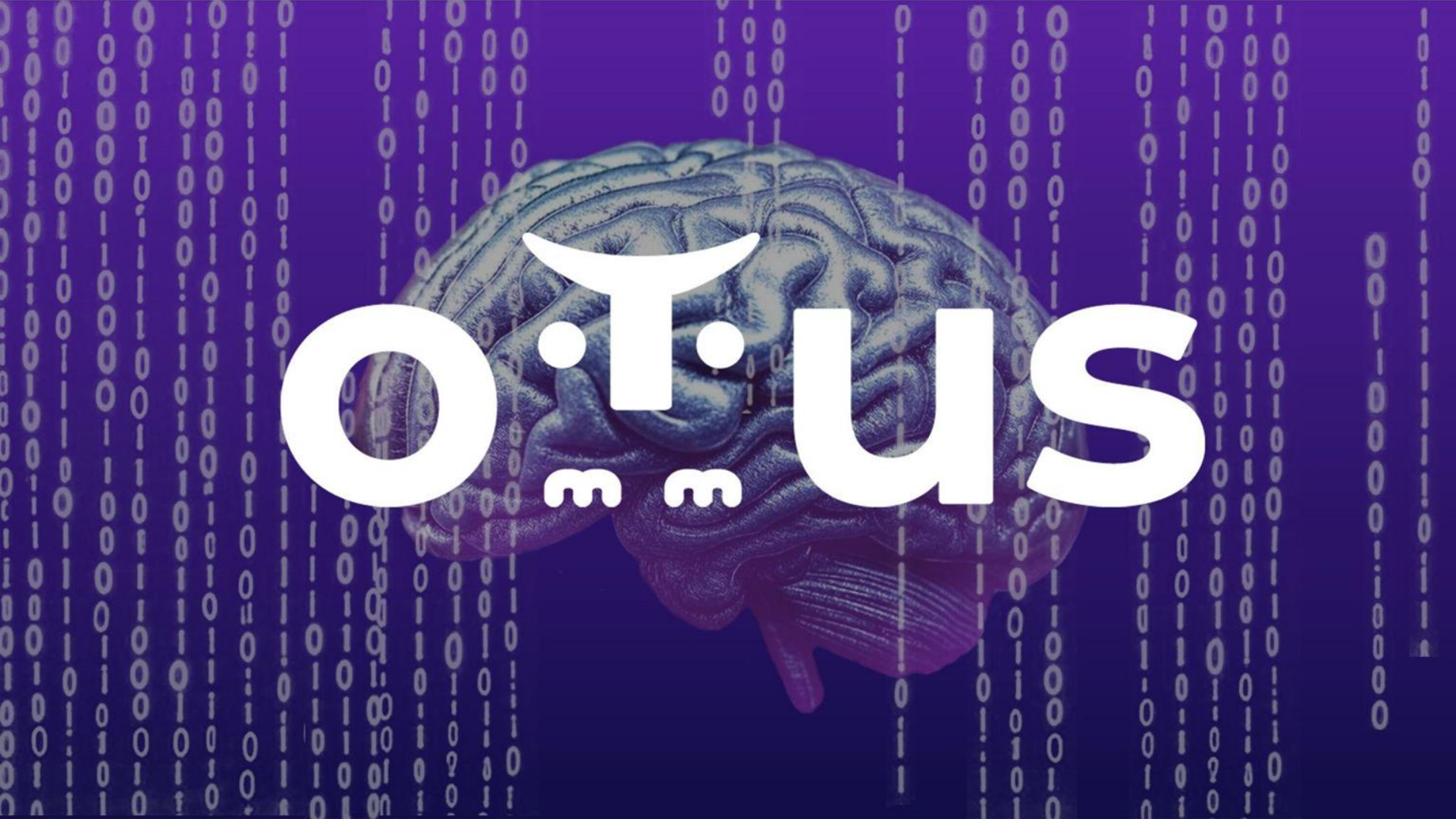
– если вопросов нет

# Рефлексия

**С какими впечатлениями уходите с вебинара?**

# Заполните, пожалуйста, опрос о занятии

Мы читаем все ваши сообщения
и берем их в работу ✏️❤️


OTIUS