

# Журналы



Проверить, идет ли запись

# Меня хорошо видно && слышно?



Ставим "+", если все хорошо  
"-", если есть проблемы



Тема вебинара

# Журналы



**Тоескин Игорь**

Старший разработчик баз данных

Telegram: @ureasyel



# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе ТГ



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу



# Маршрут вебинара

1. Буферный кэш

2. Журнал предзаписи WAL

3. Контрольная точка

4. Настройки журнала

# Цели вебинара

К концу занятия вы сможете

1. Настраивать журналирование;
2. Корректно настроить схемы контрольных точек;



# Смысл

## Зачем вам это уметь

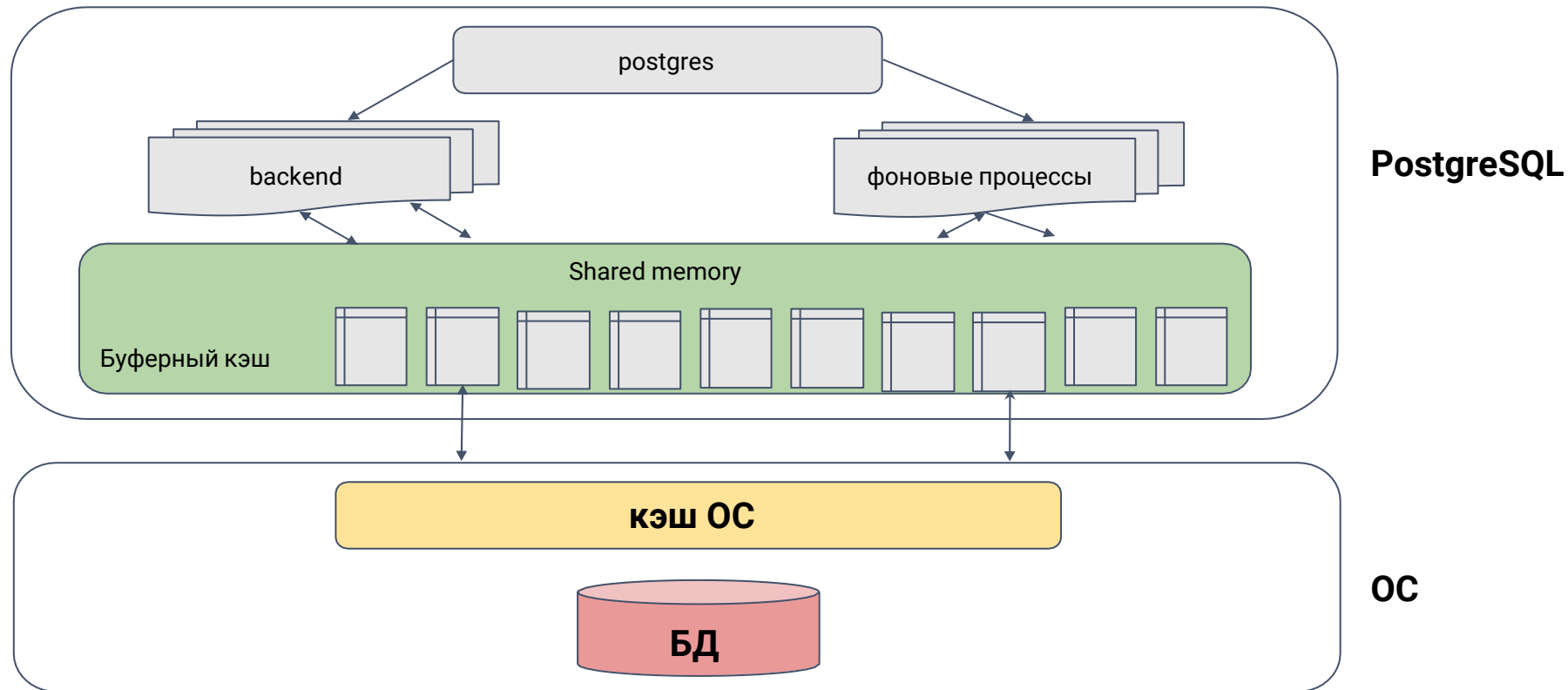
1. Обеспечить высокую надежность;
2. Обеспечить оптимальную производительность;



# Буферный кэш



# Буферный кэш



# Буферный кэш. Зачем?

Ускоряем работу всей системы.

- Оперативная память очень быстра, но ее мало;
- HDD огромный, но медленный.

# Буферный кэш. Состав

Каждый буфер состоит из одной страницы данных и заголовка. Размер по умолчанию 8 кб. Заголовок содержит:

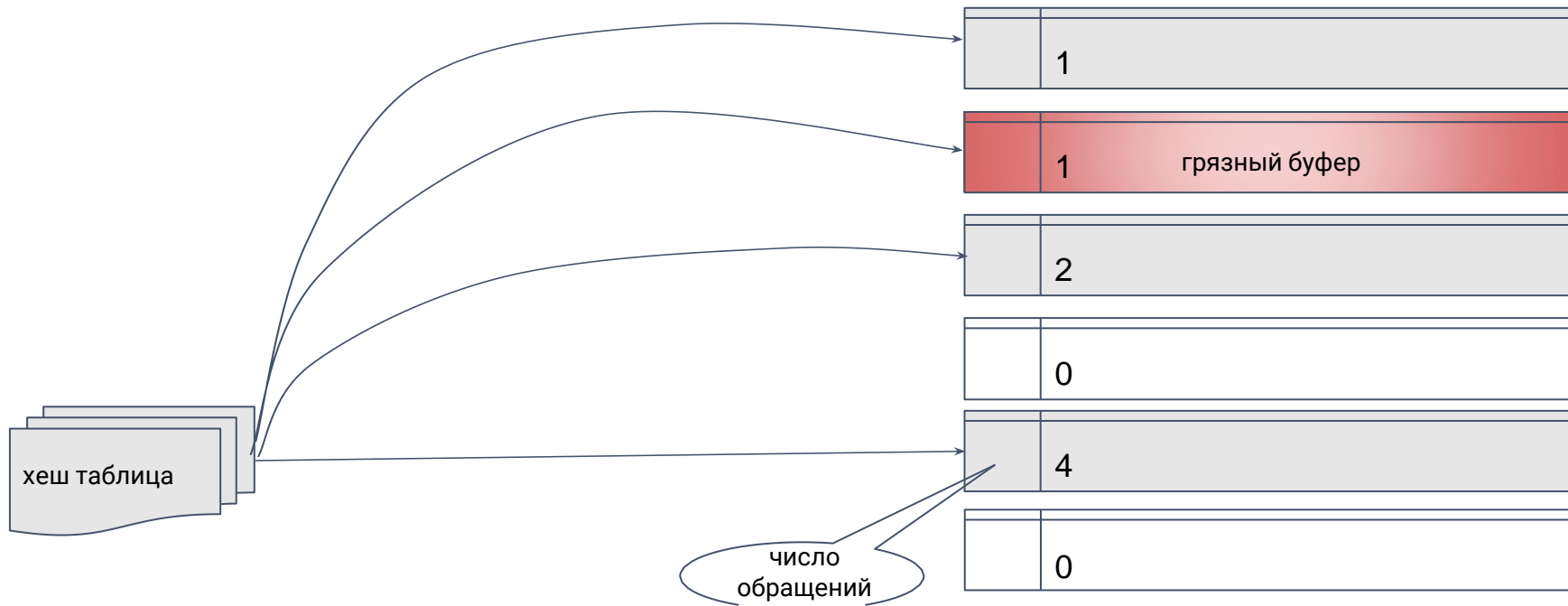
- расположение страницы на диске (файл и номер страницы в нем);
- число обращений к буферу (счетчик увеличивается каждый раз, когда процесс читает или изменяет буфер, максимально значение 5);
- признак того, что данные на странице изменились и рано или поздно должны быть записаны на диск (грязный буфер).

Изначально кэш содержит:

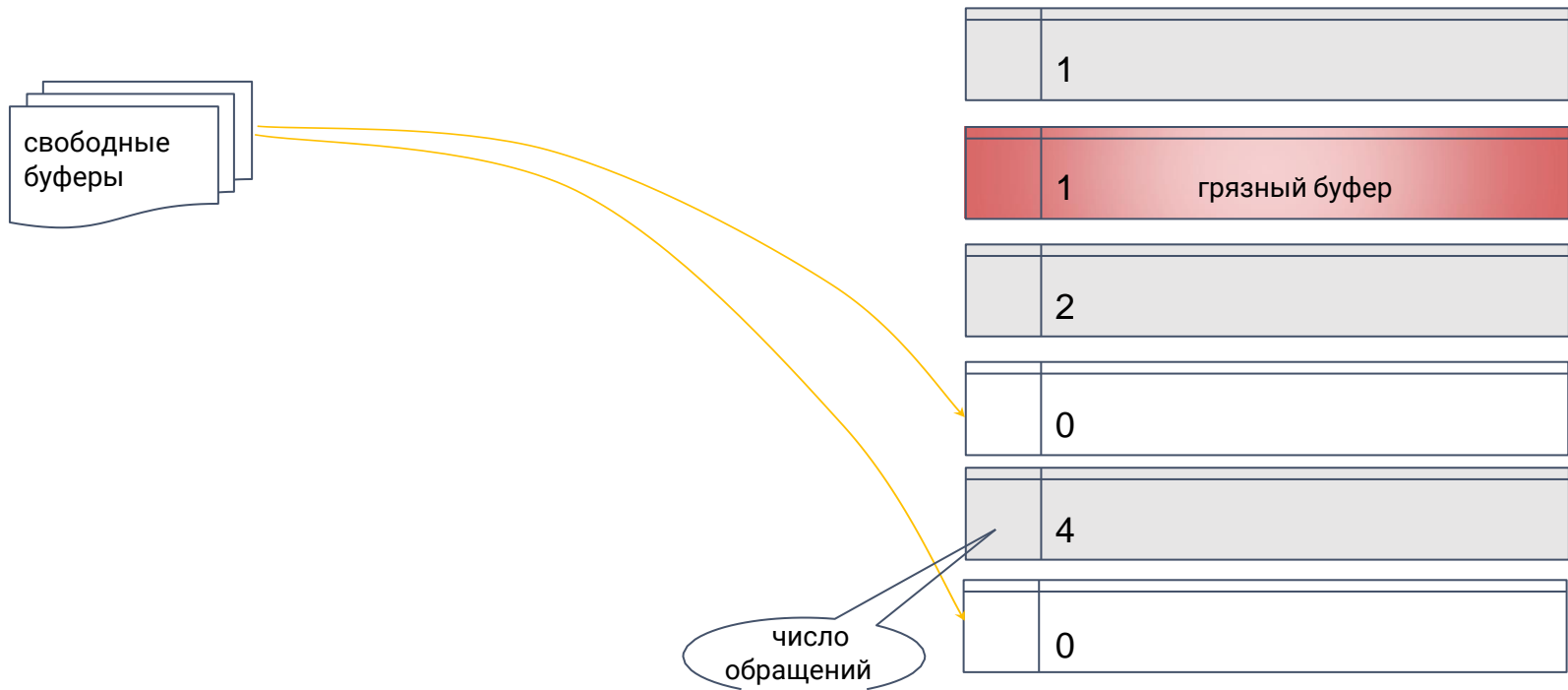
- пустые буферы, и все они связаны в список свободных буферов;
- указатель на «следующую жертву» при вытеснении старых буферов;
- также используется хеш-таблица, чтобы быстро находить нужную страницу в кэше.

Размер буферного кэша задается параметром **shared\_buffers**. Его изменение **требует перезапуска** сервера.

# Буферный кэш. Механизм работы

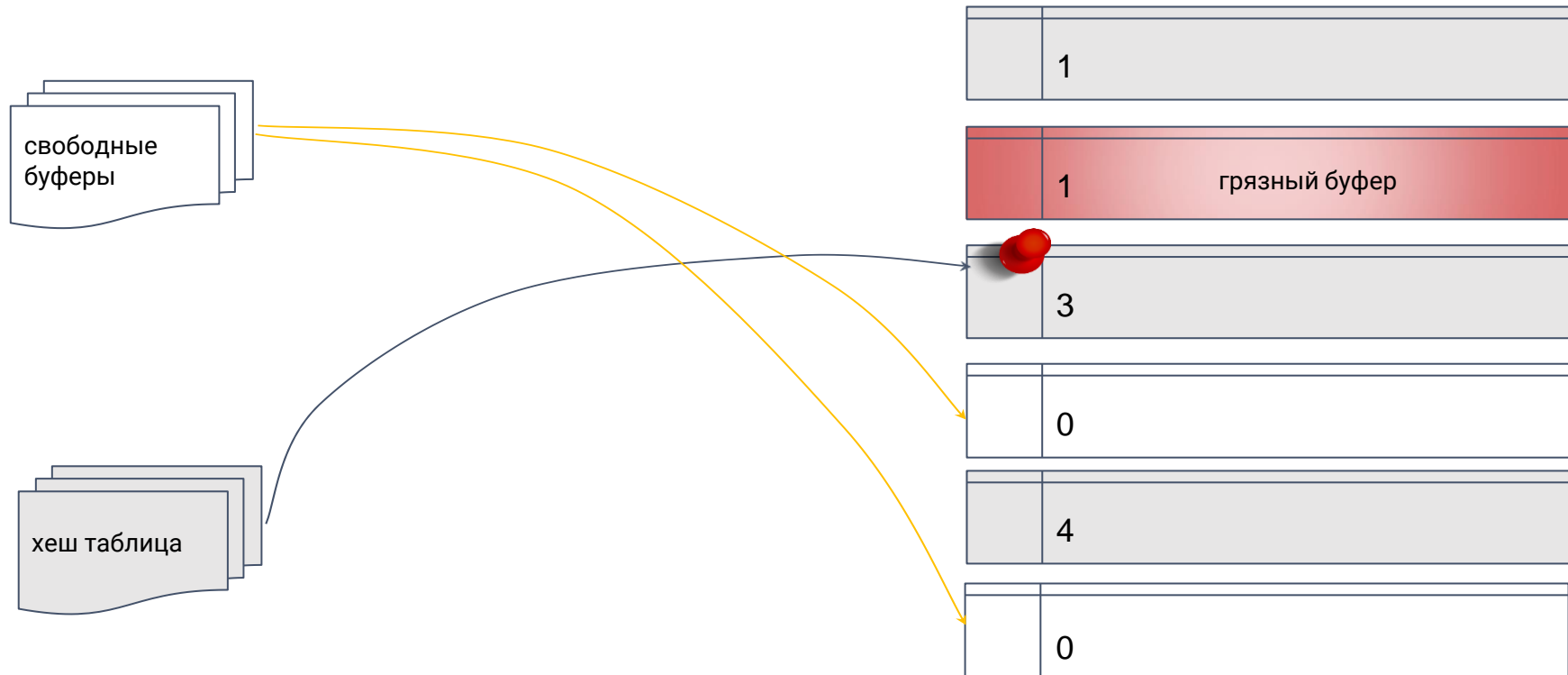


# Буферный кэш. Механизм работы



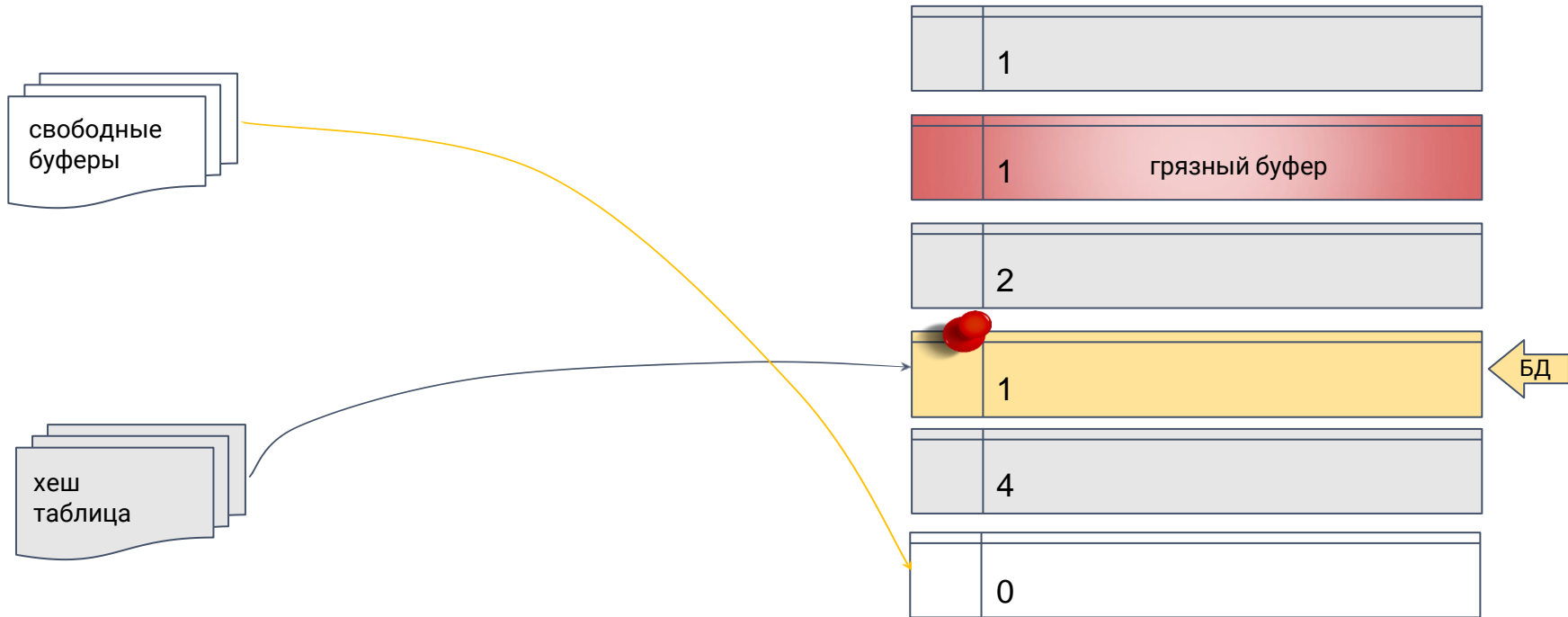
# Буферный кэш. Механизм работы

Сначала ищем в буферном кэше по хешу. Если нужная страница найдена в кэше, процесс должен «закрепить» буфер (увеличить счетчик pin count) и увеличить число обращений (счетчик usage count).



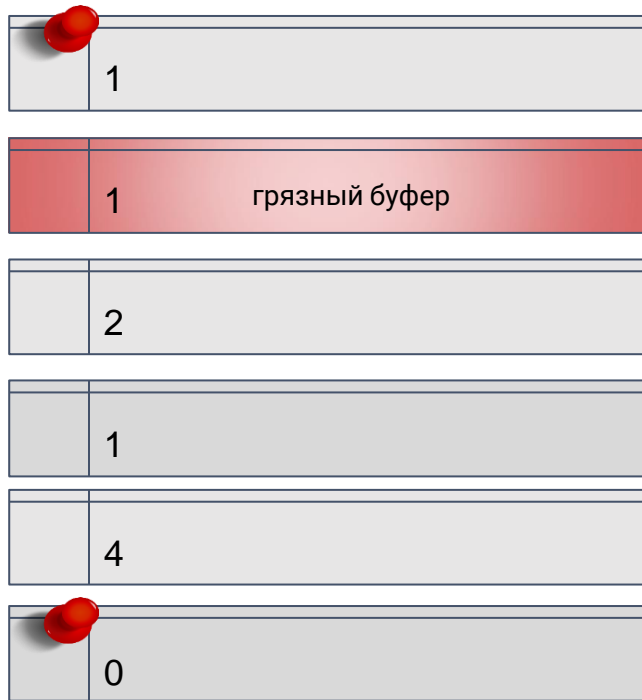
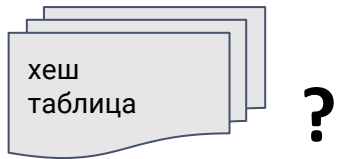
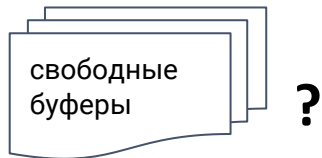
# Буферный кэш. Механизм работы

Если нет страницы в кэше, то грузим с диска в первый пустой блок и добавляем строку в хеш таблицу.



# Буферный кэш. Чтение с вытеснением

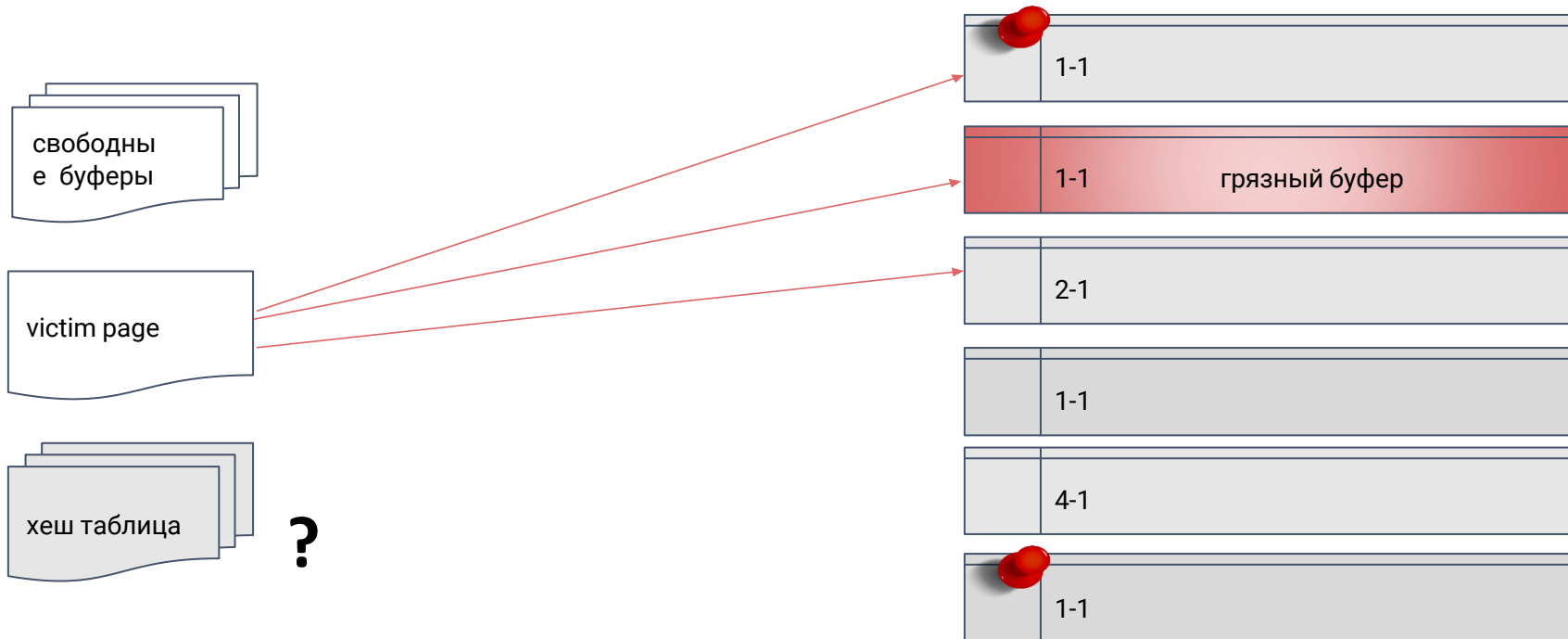
В какой-то момент все свободные блоки заканчиваются. Что делать дальше?





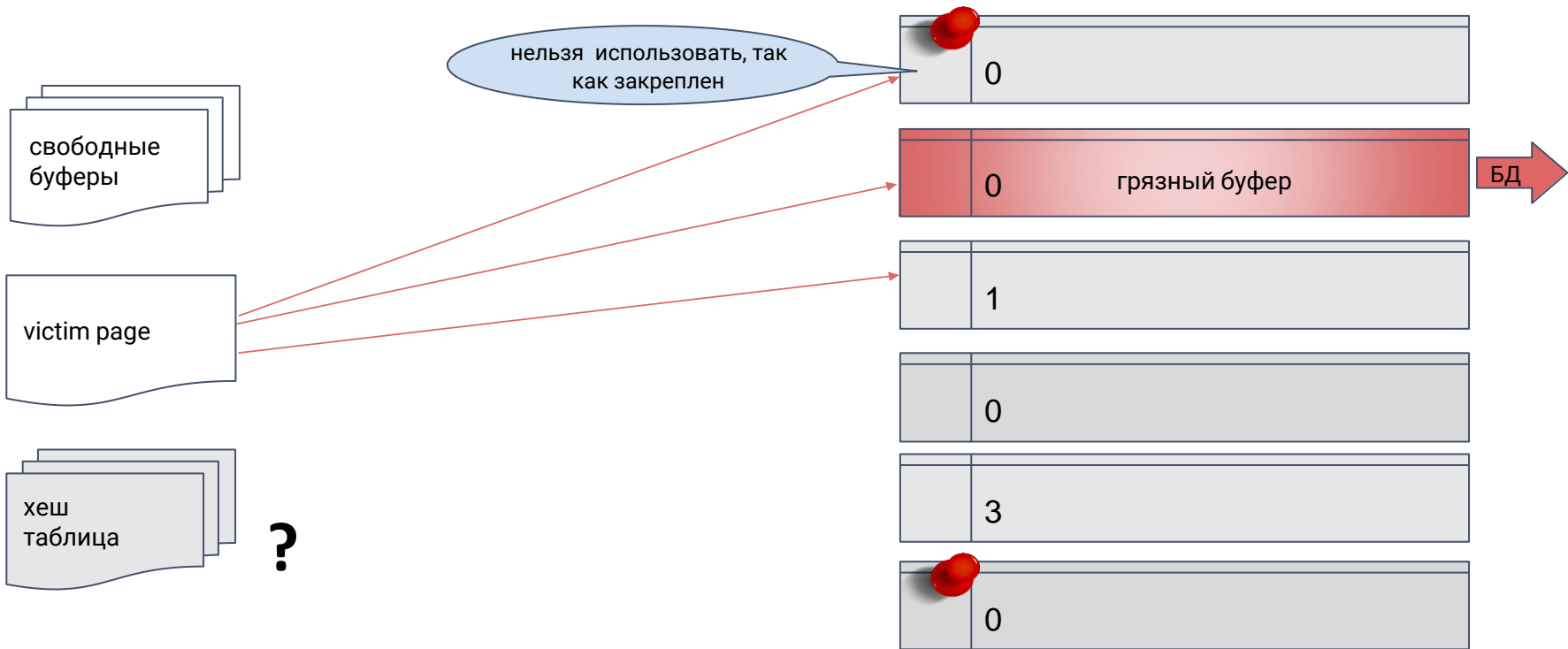
# Буферный кэш. Чтение с вытеснением

В какой-то момент все свободные блоки заканчиваются. Что делать дальше?

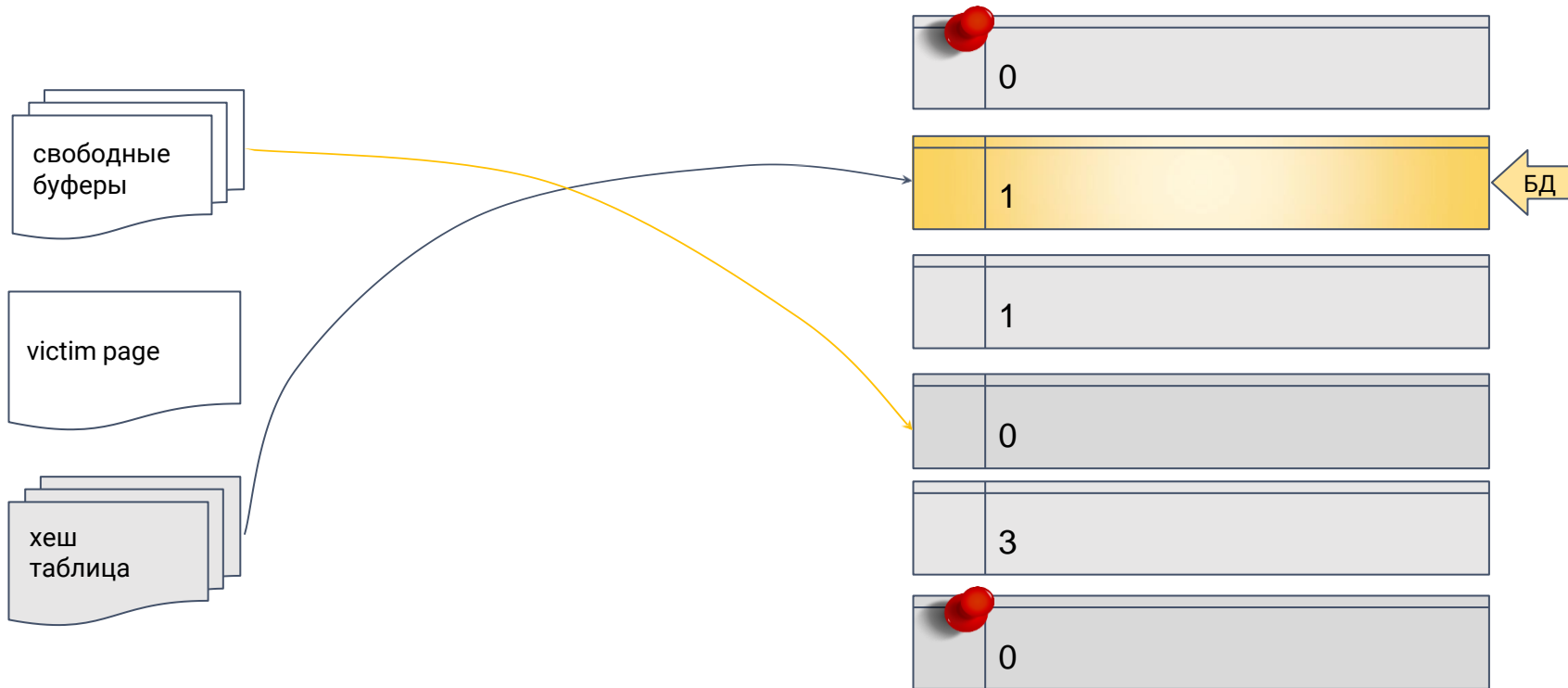


# Буферный кэш. Чтение с вытеснением

Грязную страницу нужно **заменить**, но для этого ее нужно сбросить на диск. Частично сглаживается процессом **bgwriter** и механизмом контрольных точек.



# Буферный кэш. Чтение с вытеснением



# Буферный кэш. Массовое вытеснение

## Буферное кольцо

Часть буферного кэша, выделенная для одной операции предотвращает вытеснение кэша «одноразовыми» данными.

операция	кол-во страниц	грязные буферы
последовательное чтение (несколько операций одновременно)	32	исключаются из кольца
очистка (VACUUM)	32	вытесняются на диск
массовая запись (COPY, CTAS)	$\leq 2048$	вытесняются на диск

# Буферный кэш. Настройка

По умолчанию `shared_buffers` = 128MB

Буферный кэш должен содержать «активные» данные:

- при меньшем размере постоянно вытесняются полезные страницы;
- при большем размере бессмысленно растут накладные расходы.

Начальная рекомендация — **25%** ОЗУ

Нужно учитывать двойное кэширование - если страницы нет в кэше СУБД, она может оказаться в кэше ОС, но алгоритм вытеснения ОС не учитывает специфики базы данных.

# Буферный кэш. Временные таблицы

Данные временных таблиц:

- видны только одному сеансу — нет смысла использовать общий кэш;
- существуют в пределах сеанса — не жалко потерять при сбое.

Используется локальный буферный кэш:

- не требуются блокировки;
- память выделяется по необходимости в пределах `temp_buffers`;
- обычный алгоритм вытеснения.

# Буферный кэш. Разогрев кэша

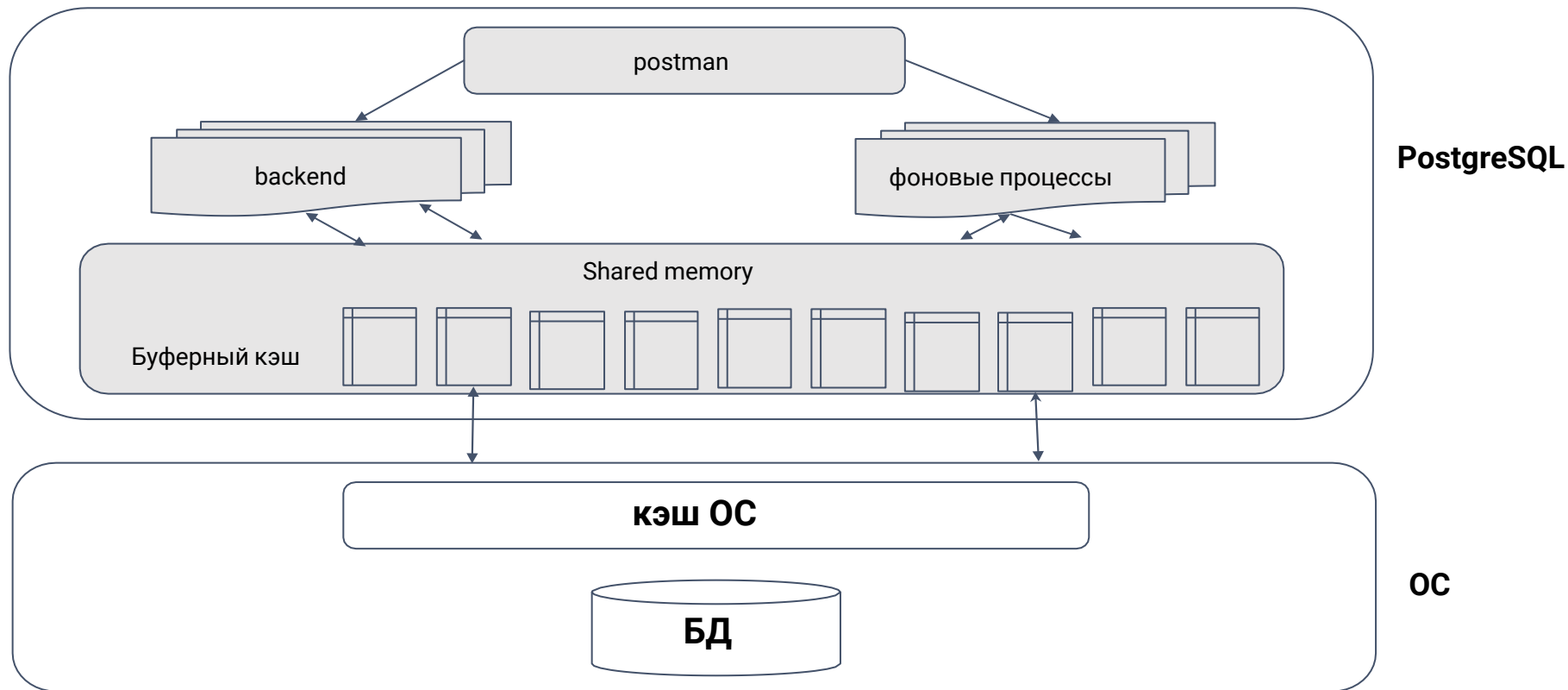
- **pg\_prewarm**
- используется после рестарта кластера
- заполняет кэш указанными таблицами

# Вопросы?



# Журнал предзаписи

# Буферный кэш & WAL



# Write ahead log - WAL

## Основная задача

- возможность восстановления согласованности данных после сбоя

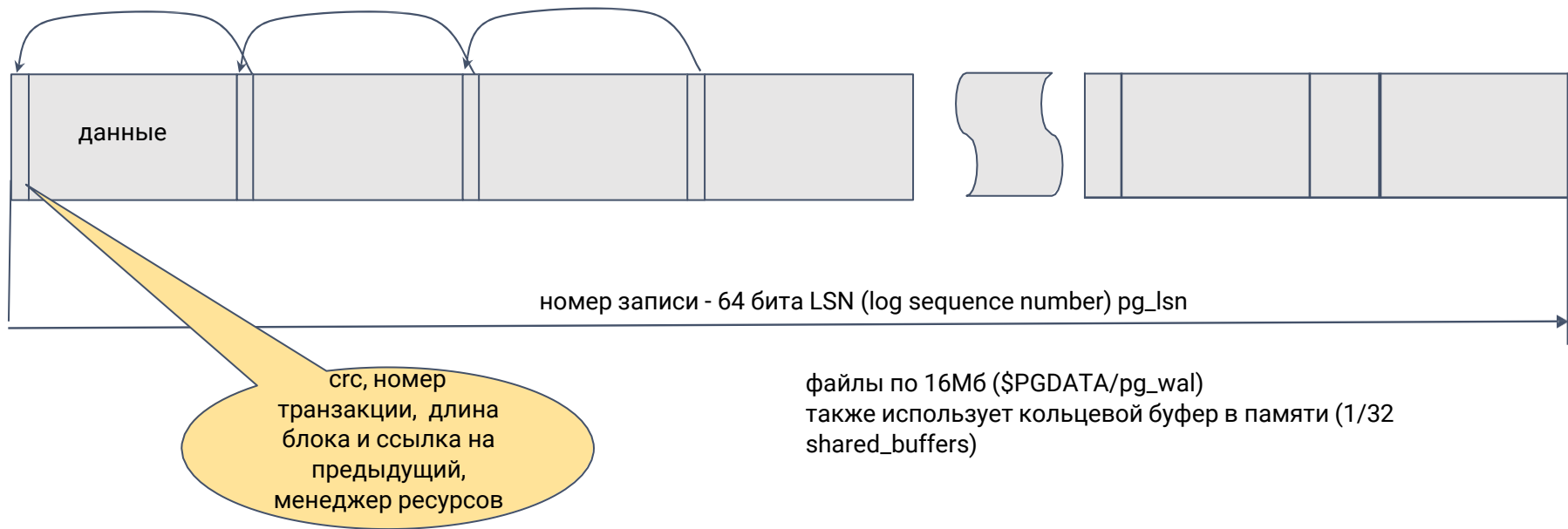
## Механизм

- при изменении данных действие также записывается в журнал журнальная запись попадает на диск раньше измененных данных
- восстановление после сбоя — повторное выполнение потерянных операций с помощью журнальных записей

## Что туда попадает

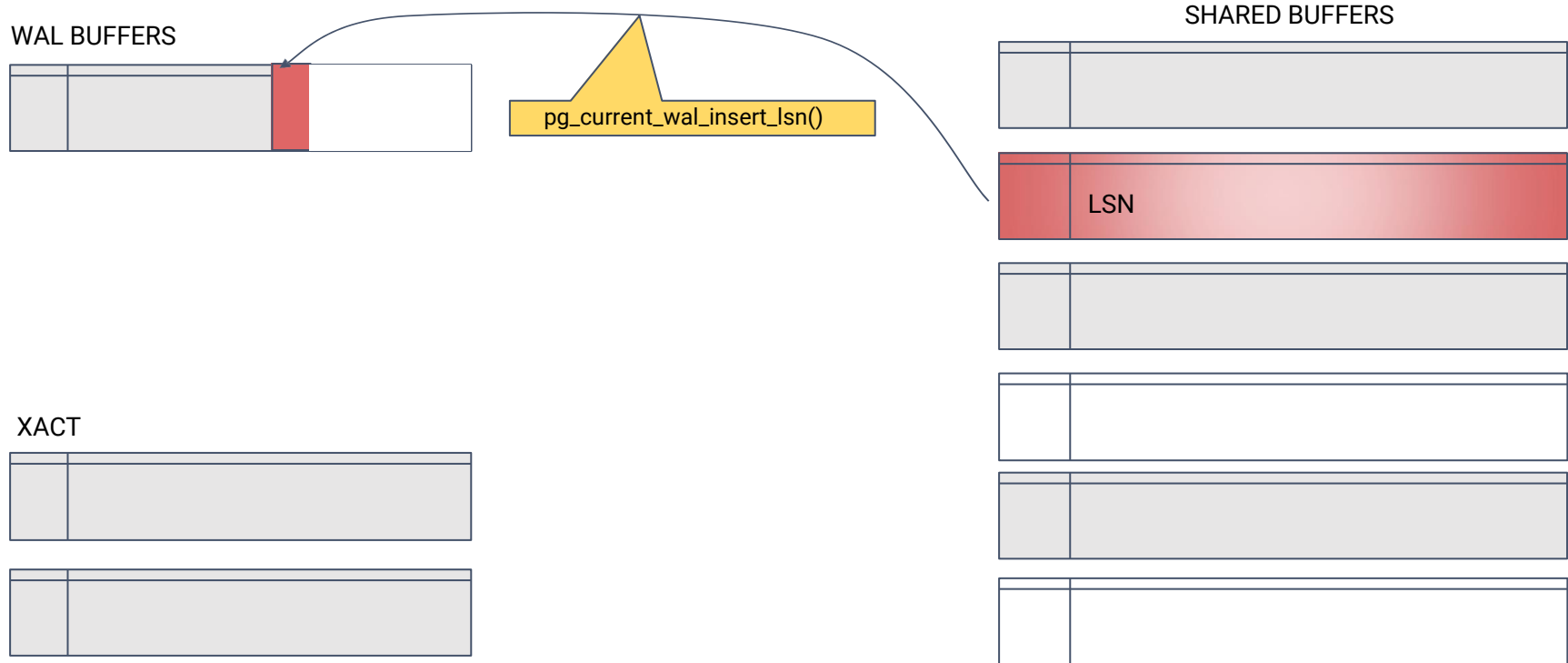
- изменение любых страниц в буферном кэше
- фиксация и отмена транзакций - буферы ХАСТ
- НЕ ПОПАДАЮТ - временные и нежурналируемые таблицы

# WAL. Устройство

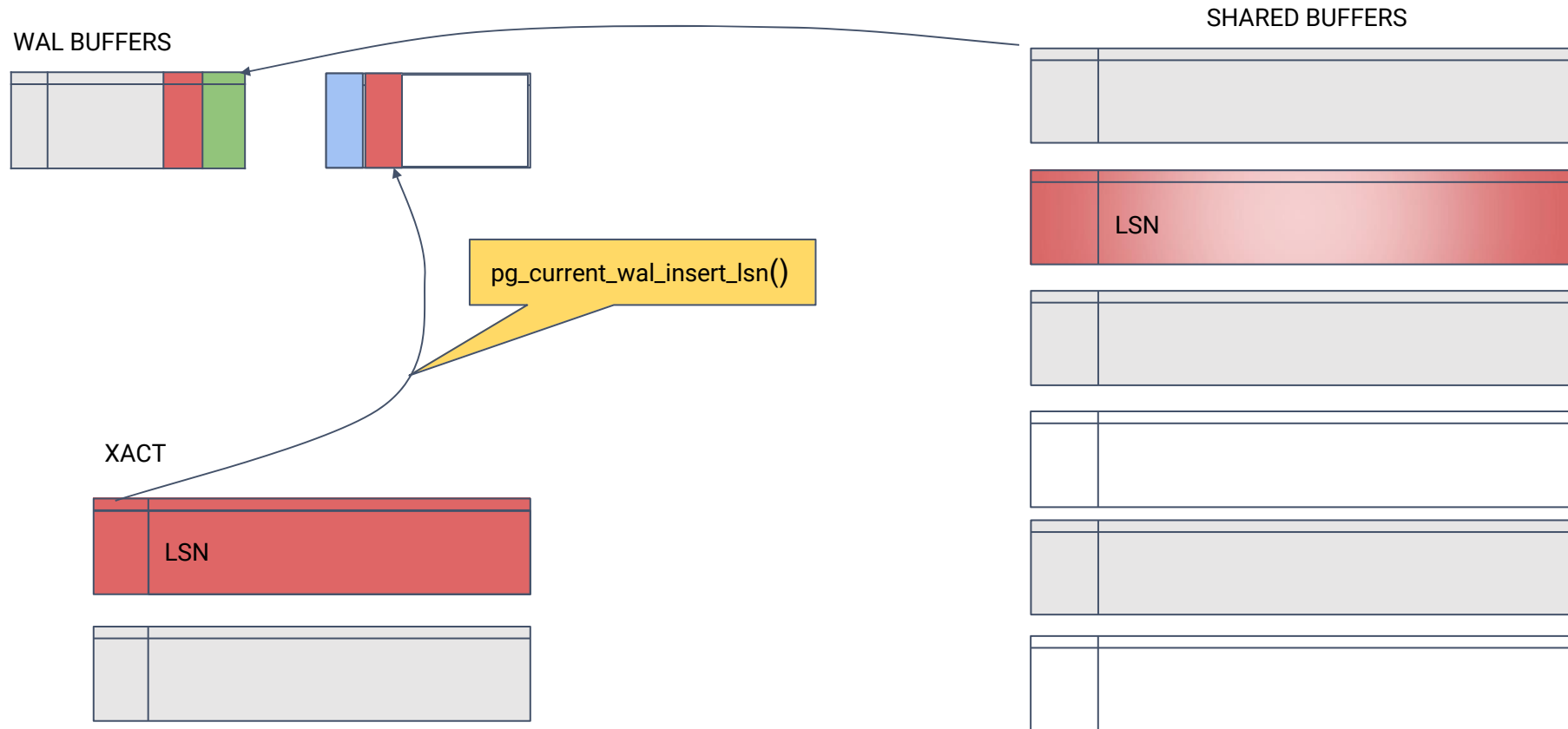


`$/usr/lib/postgresql/13/bin/pg_waldump -r list` - список менеджеров

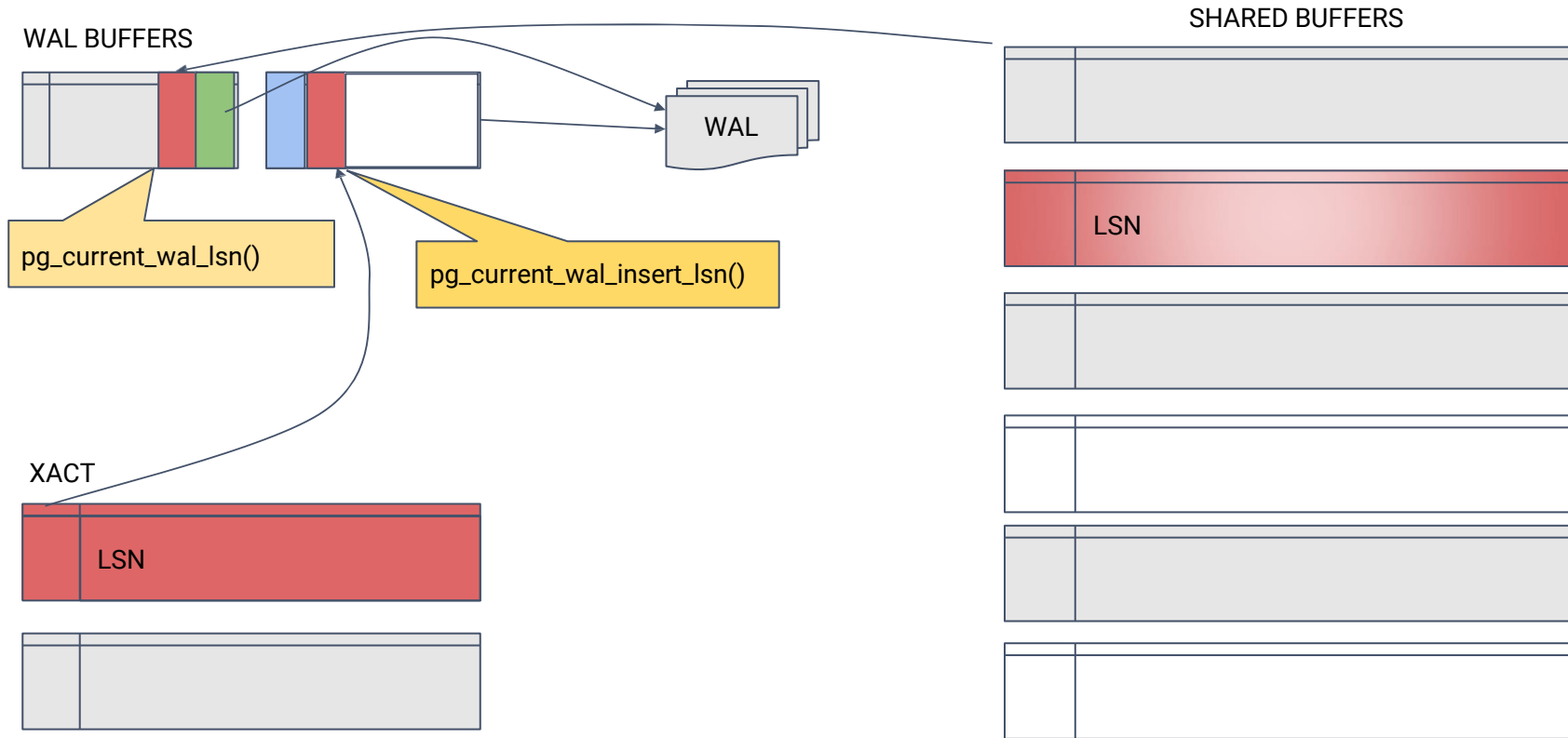
# WAL. Механизм упреждающей записи



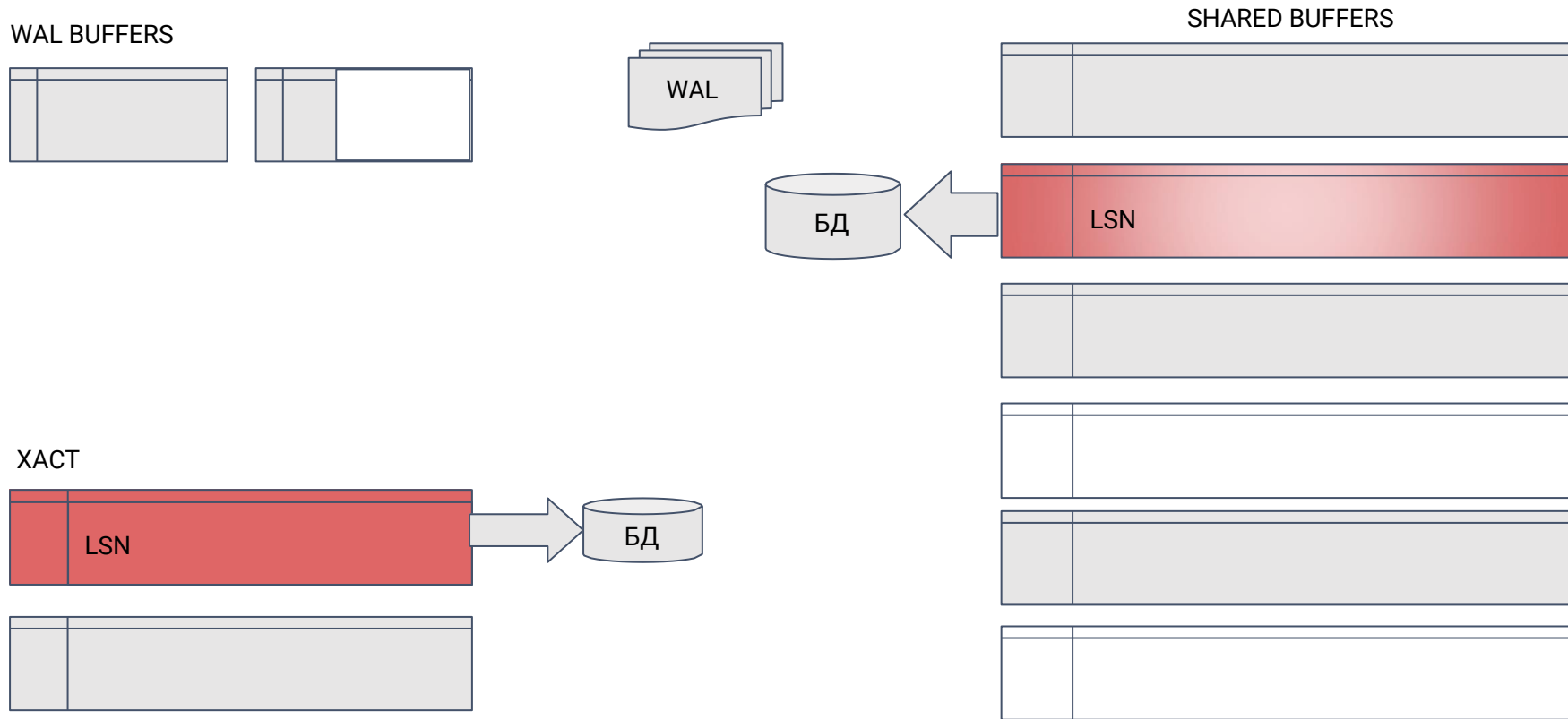
# WAL. Механизм упреждающей записи



# WAL. Механизм упреждающей записи



# WAL. Механизм упреждающей записи





# WAL. Восстановление

## При старте сервера после сбоя

(состояние кластера в pg\_control отличается от «shut down»):

- для каждой журнальной записи:
  - определить страницу, к которой относится эта запись
  - применить запись, если ее LSN больше, чем LSN страницы
- перезаписать нежурналируемые таблицы init-файлами

# Вопросы?

# Контрольная точка

# Контрольная точка

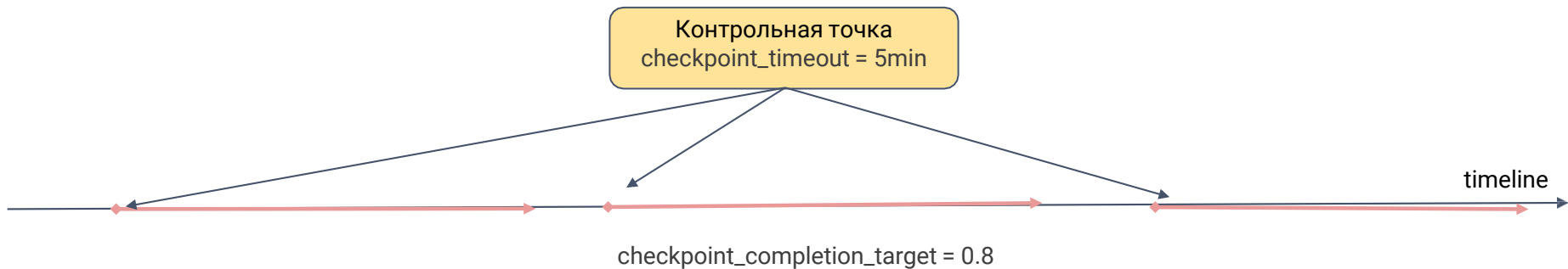
Зачем она нужна? Можно же с самого начала накатить все wal?

# Контрольная точка

Зачем она нужна? Можно же с самого начала накатить все wal?

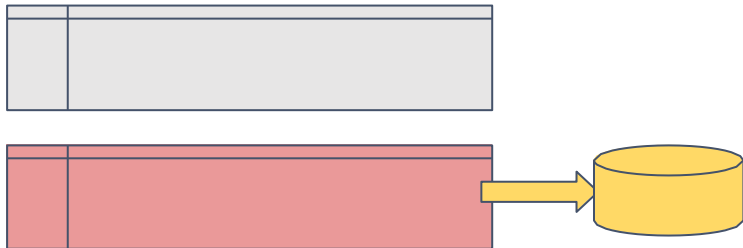
- очень большой объем информации хранить
- большое время восстановления
- сколько может страница измененная лежать в буферном кэше?

# Контрольная точка



# Контрольная точка

ХАСТ



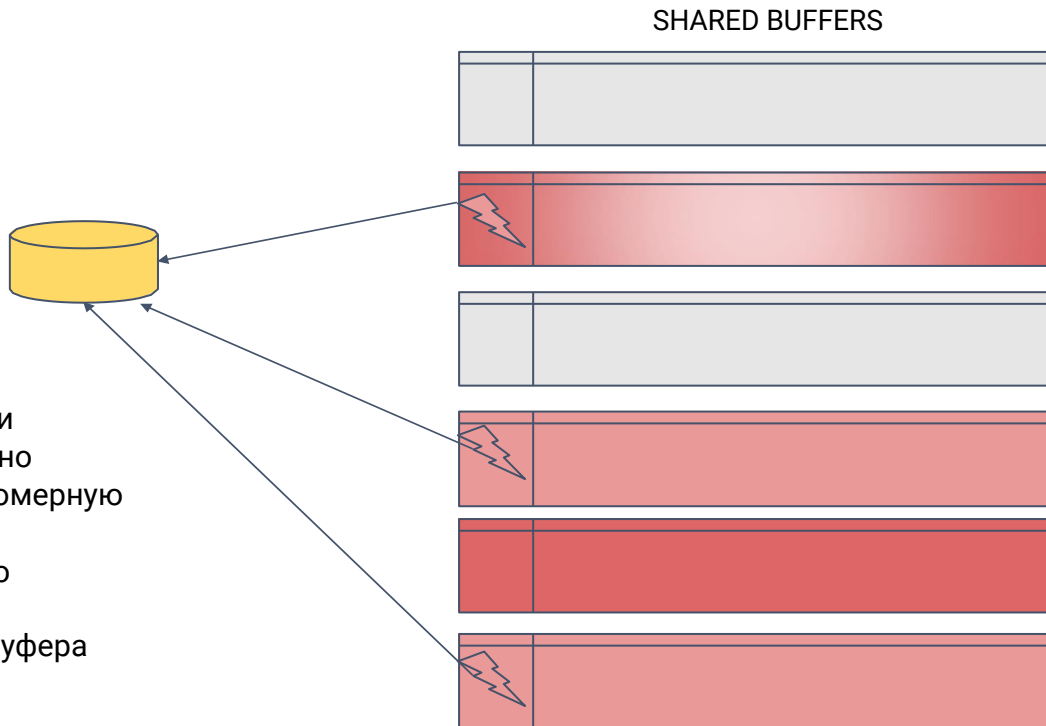
- сначала сбрасываются буферы ХАСТ
- помечаются грязные буферы

SHARED BUFFERS



# Контрольная точка

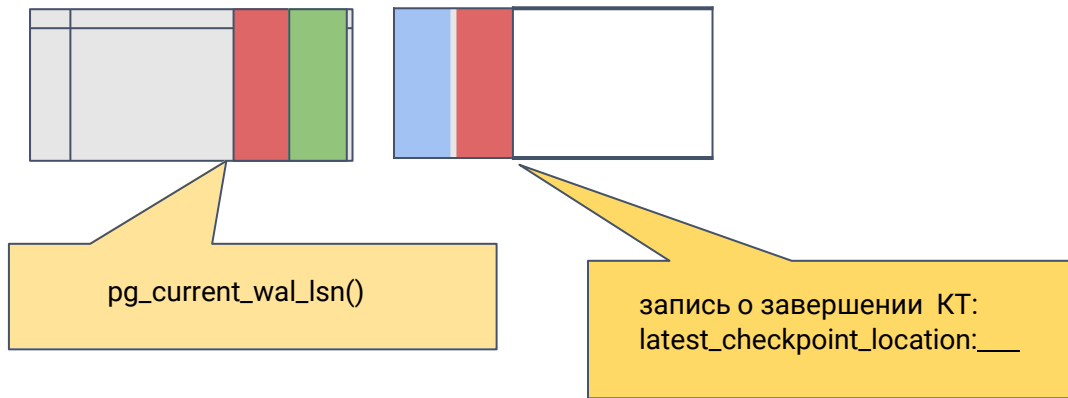
- зная значение параметра `checkpoint_completion_target = 0.5` и количество блоков, которые нужно записать мы рассчитываем равномерную запись
- помеченные страницы постепенно записываются
- пометка убирается из заголовка буфера





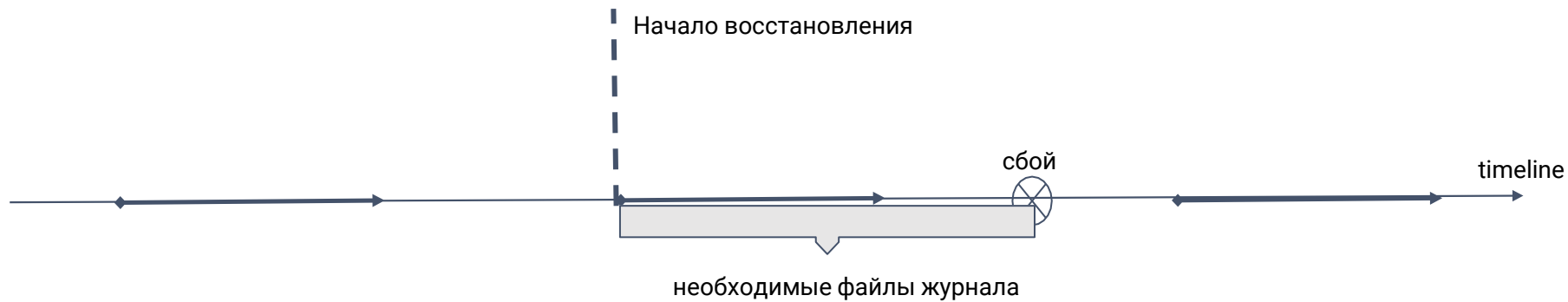
# Контрольная точка

WAL



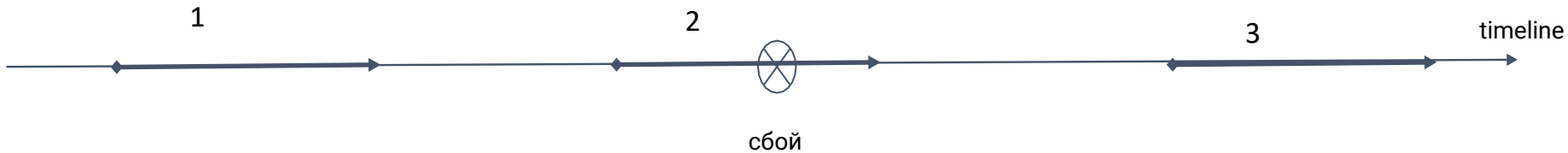
- 1. в журнале создается запись о завершении контрольной точки с указанием момента ее начала
- 1. в файл `$PGDATA/global/pg_control` записывается LSN контрольной точки
  - ...
  - Latest checkpoint location: 0/12B35EBB
  - ...

# Контрольная точка

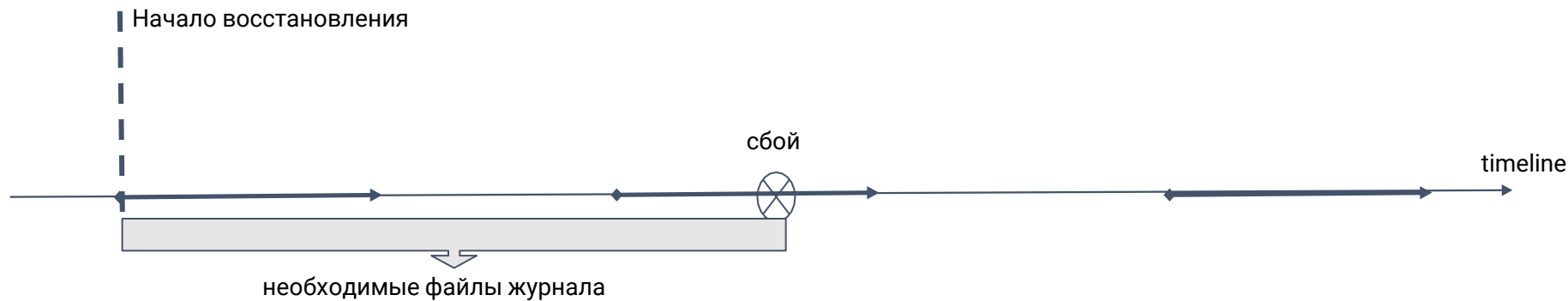


# Контрольная точка

С какой точки произойдет восстановление и за какой период нужны будут wal файлы?



# Контрольная точка



# Контрольная точка

## При старте сервера после сбоя

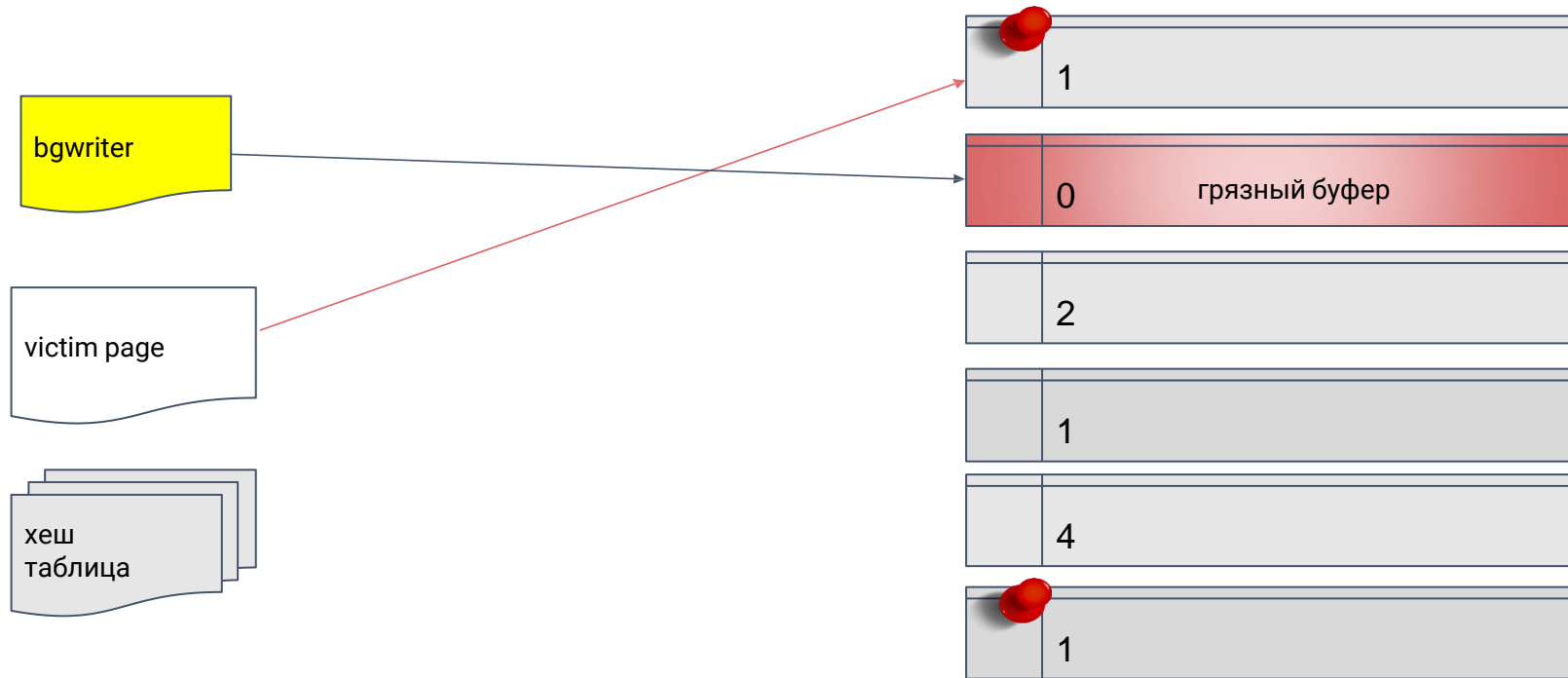
- Найти LSN0 начала последней завершенной контрольной точки;
- Применить каждую запись журнала, начиная с LSN0, если LSN записи больше, чем LSN страницы;
- Перезаписать нежурналируемые таблицы init-файлами;
- Выполнить контрольную точку.

# Контрольная точка

- **Настройка частоты срабатывания:**
  - `checkpoint_timeout = 5min`
  - `max_wal_size = 1GB`
- **Сервер хранит журнальные файлы необходимые для восстановления:**
  - $(2 \text{ (1 с 12 версии)} + \text{checkpoint\_completion\_target}) * \text{max\_wal\_size}$
  - еще не прочитанные через слоты репликации
  - еще не записанные в архив, если настроена непрерывная архивация
  - не превышающие по объему минимальной отметки
- **Настройки**
  - `max_wal_size = 1GB`
  - `min_wal_size = 100MB`
  - `wal_keep_segments = 0`

# Контрольная точка. Процесс фоновой записи

В какой-то момент все свободные блоки заканчиваются. Что делать дальше?



# Контрольная точка

- **Настройки**

- `bgwriter_delay = 200ms`
- `bgwriter_lru_maxpages = 100`
- `bgwriter_lru_multiplier = 2.0`

- **Алгоритм**

- уснуть на `bgwriter_delay`
- если в среднем за цикл запрашивается  $N$  буферов, то записать  $N * \text{bgwriter\_lru\_multiplier} \leq \text{bgwriter\_lru\_maxpages}$  грязных буферов



# Контрольная точка. Практика

# Вопросы?

# Настройка журнала

# Уровни журнала

- **Minimal**
  - восстановление после сбоя
- **Replica**
  - восстановление из резервной копии, репликация
  - + операции массовой обработки данных, блокировки
- **Logical**
  - логическая репликация
  - + информация для логического декодирования
- **Настройка**
  - `wal_level = replica`

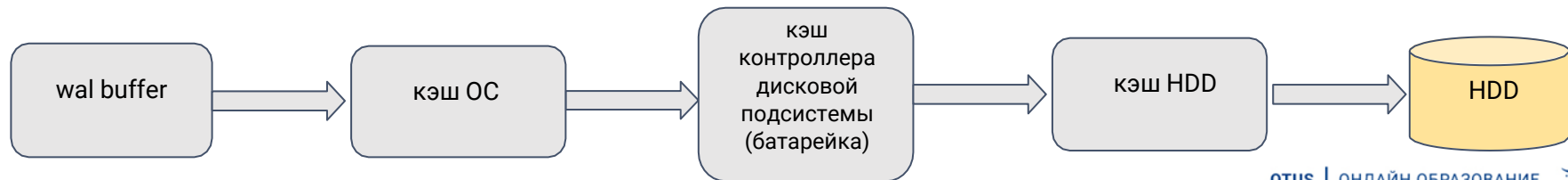
# Настройка записи на диск

## Синхронизация с диском

- данные должны дойти до энергонезависимого хранилища через многочисленные кэши
- СУБД сообщает операционной системе способом, указанным в `wal_sync_method` надо учитывать аппаратное кэширование

## Настройки

- `fsync = on;`
- `show fsync;`
- `show wal_sync_method;`
- утилита `pg_test_fsync` помогает выбрать оптимальный способ



# Повреждение данных

- **Контрольные суммы журнальных записей**
  - включены всегда, CRC-32
- **Контрольные суммы страниц (накладные расходы)**
  - По умолчанию отключены. До 12 версии можно включить только при инициализации кластера.
  - `pg_createcluster --data-checksums`
- **Настройки**
  - `show data_checksums;`
  - `ignore_checksum_failure = off;`
  - `wal_log_hints = off` (записывает все содержимое каждой страницы при изменениях даже инф.бит, неявно on при контрольных суммах страниц);
  - `wal_compression = off;`

# Характер нагрузки

## Постоянный поток записи

- характер нагрузки отличается от остальной системы
- последовательная запись, отсутствие случайного доступа
- при высокой нагрузке — размещение на отдельных физических дисках (символьная ссылка из `$PGDATA/pg_wal`)

## Редкое чтение

- при восстановлении
- при работе процессов `walsender`, если реплика не успевает быстро получать записи

# Режим записи

- синхронный режим
- асинхронный режим



# Режим синхронной записи

- **Алгоритм**

- при фиксации изменений сбрасывает накопившиеся записи, включая
- запись о фиксации
- ждет `commit_delay`, если активно не менее `commit_siblings` транзакций

- **Характеристики**

- гарантируется долговечность
- увеличивается время отклика

- **Настройки**

- `synchronous_commit = on`
- `commit_delay = 0`
- `commit_siblings = 5`

# Режим асинхронной записи

- **Алгоритм**

- циклы записи через `wal_writer_delay`
- записывает только целиком заполненные страницы
- но если новых полных страниц нет, то записывает последнюю до конца

- **Характеристики**

- гарантируется согласованность, но не долговечность
- зафиксированные изменения могут пропасть ( $3 \times \text{wal\_writer\_delay}$ )
- уменьшается время отклика

- **Настройки**

- `synchronous_commit = off` (можно изменять на уровне транзакции)
- `wal_writer_delay = 200ms`

# Рефлексия

# Вопросы?

- Кто что запомнил за сегодня?
- Сколько контрольных точек рекомендовано хранить для гарантированного восстановления?
- Как вам баланс между теорией и практикой?

# Домашнее задание

# ДЗ

1. Настройте выполнение контрольной точки раз в 30 секунд.
2. 10 минут с помощью утилиты `rgbench` подавайте нагрузку.
3. Измерьте, какой объем журнальных файлов был сгенерирован за это время. Оцените, какой объем приходится в среднем на одну контрольную точку.
4. Проверьте данные статистики: все ли контрольные точки выполнялись точно по расписанию. Почему так произошло?
5. Сравните `tps` в синхронном/асинхронном режиме утилитой `rgbench`. Объясните полученный результат.
6. Создайте новый кластер с включенной контрольной суммой страниц. Создайте таблицу. Вставьте несколько значений. Выключите кластер. Измените пару байт в таблице. Включите кластер и сделайте выборку из таблицы. Что и почему произошло? как проигнорировать ошибку и продолжить работу?

**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**