



# PostgreSQL для администраторов баз данных и разработчиков

Тема вебинара

# Физический уровень PostgreSQL



**Игорь Тоекин**

**Ведущий разработчик СУБД**

Специалист в области разработки и проектировании витрин данных в PostgreSQL, а также в области разработки хранимых процедур в таких СУБД как PostgreSQL и Oracle



# Правила вебинара



Активно  
участвуем



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначения



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или  
задайте вопрос

# Маршрут сессии



Работа с PostgreSQL

Серверные процессы и память

Физическая структура данных

Практика, рефлексия

# Цели сессии

1. Иметь представление об устройстве PostgreSQL
2. Знать процессы PostgreSQL и структуру памяти
3. Изучить, как PostgreSQL работает с данными на физическом уровне

# Подключение к PostgreSQL

# Клиенты

- Клиент PostgreSQL
  - работает через **TCP** и **Unix socket**
  - самописный софт как правило использует библиотеку **jdbc**
  - протокол взаимодействия открыт
- Для каждого клиентского соединения порождается выделенный серверный процесс
  - при этом клиентский процесс называют - **frontend**
  - а серверный - **backend**
- Чаще всего работаем через **psql**
  - есть и GUI, например pgAdmin3/4, dBeaver, DataGrip

# Клиент psql

- Запуск
  - `$ psql -d база -U роль -h узел -p порт`
- Новое подключение в psql
  - `=> \c[onnect] база роль узел порт`
- Информация о текущем подключении
  - `=> \conninfo`



# Клиент psql

- Интерактивный интерфейс командной строки для базы данных

```
psql [<option> ...] [<dbname> [<username>]]
```

- Позволяет в интерактивном режиме вводить запросы, передавать их в БД и просматривать результаты запроса. Кроме того, запрос может быть введен из файла.
- Кроме того, в терминале имеется ряд **метакоманд** и различных **shell-подобных** функций, облегчающих написание скриптов и автоматизацию множества задач.

# Клиент psql

- -a | **–echo-all**
- -A | **–no-align**
- -c '**command**' | **–command='command'**
- -e | **–echo-queries**
- -E | **–echo-hidden**
- -f filename | **–file=filename**
- -F separator | **–field-separator=separator**
- -H | **–html**
- -l | **–list**
- -L filename | **–log-file=filename**
- -n | **–no-readline**
- -o filename | **–output=filename**
- -P assignment | **–pset=assignment**
- -q | **–quiet**
- -R separator | **–record-separator=separator**
- -s | **–single-step**
- -S | **–single-line**
- -t | **–tuples-only**
- -T table\_options | **–table-attr= table\_options**
- -v assignment | **–set=assignment** | **–variable=assignment**
- -V | **–version**
- -x | **–expanded**
- -X | **–no-psqlrc**
- -z | **–field-separator-zero**
- -0 | **–record-separator-zero**
- -1 | **–single-transaction**
- -? | **–help**

# Клиент psql

## Параметры подключения

- -h host | -host=host
- -p port | -port=port
- -U username | -username=username
- -W | -password
- -w -no-password
- -d dbname | -dbname=dbname

# Клиент psql

- Если встречается аргумент, не относящийся ни к одной из опций, то он интерпретируется как имя базы данных (или имя пользователя, если имя базы данных уже задано).
- Если значения по умолчанию не подходят, можно сэкономить время на вводе текста, установив соответствующие значения переменных окружения **PGAPPNAME**, **PGDATABASE**, **PGHOST**, **PGPORT** и **PGUSER**.
- Альтернативным способом указания параметров соединения является строка conninfo или URI, которая используется вместо имени базы данных.
  - **psql "service=myservice sslmode=require"**
  - **psql postgresql://gpcoordinator:5433/mydb?sslmode=require**

# Клиент psql - метакоманды

- \c | \connect [dbname [username] [host] [port]] | conninfo
- \cd [directory]
  - To print your current working directory, use \!pwd.
- \conninfo
- \copy {table [(column\_list)] | (query)} {from | to} {'filename' | program 'command' | stdin | stdout | pstdin | pstdout} [with] (option [, ...]) ]
- \d [relation\_pattern] | \d+ [relation\_pattern] | \dS [relation\_pattern]
- \db List all tablespaces
- \dn List all schemas
- \d\_\dt List all tables
- \dv List all views
- \d table-name Show table definition
- \du List all users
- \l List all databases in this cluster
- \q | \quit
- \s [filename]
- \timing [novalue | on | off]
- \?
- \x

# Клиент psql

- При запуске psql выполняются команды, записанные в двух файлах — **общесистемном** и **пользовательском**.
- **Общий системный файл** называется **psqlrc** и располагается в каталоге **/usr/local/pgsql/etc** при обычной сборке из исходных кодов. Расположение этого каталога можно узнать командой **pg\_config --sysconfdir**
- **Пользовательский файл** находится в домашнем каталоге пользователя ОС и называется **.psqlrc**. Его расположение можно изменить, задав переменную окружения **PSQLRC**.

# Клиент psql

- В эти файлы можно записать команды, настраивающие psql — например, изменить приглашение, включить вывод времени выполнения команд и т. п.
- История команд сохраняется в файле `.psql_history` в домашнем каталоге пользователя.
  - Расположение этого файла можно изменить, задав переменную окружения **PSQL\_HISTORY** или переменную psql HISTFILE.
  - По умолчанию хранится 500 последних команд; это число можно изменить переменной psql HISTSIZE

# Файлы конфигураций

- Как посмотреть конфигурационные файлы?
- Зависит ли путь от ОС?



# Файлы конфигураций

- # show hba\_file;
- # show config\_file;

## Все параметры:

- # show all;
- # select name, setting, context, short\_desc from pg\_settings;

<https://postgrespro.ru/docs/postgrespro/17/view-pg-settings>



# Файлы конфигураций

- По умолчанию postgres слушает только localhost
- Что нужно сделать для подключения?

# Файлы конфигураций

1. Включаем listener в postgresql.conf
  - `listen_addresses = '*'` # IP адреса, на которых принимает постгрес подключения, например localhost, 10.\*.\*.\*;
  - `netstat -a|grep post`
  - *# show listen\_addresses; -второй вариант*
  - *# alter system set listen\_addresses = '\*';*
2. Включаем вход по паролю в pg\_hba.conf и меняем маску подсети
3. Задаем пароль юзеру postgres
4. Перезагружаем сервер

# Метод аутентификации

- При настройке pg\_hba.conf обратите внимание на метод аутентификации
- Обязательно md5 (13), scram-sha-256(14)
  - если password - пароль не шифруется при передаче по сети
  - <https://postgrespro.ru/docs/postgresql/17/auth-pg-hba-conf>

# Вопросы?



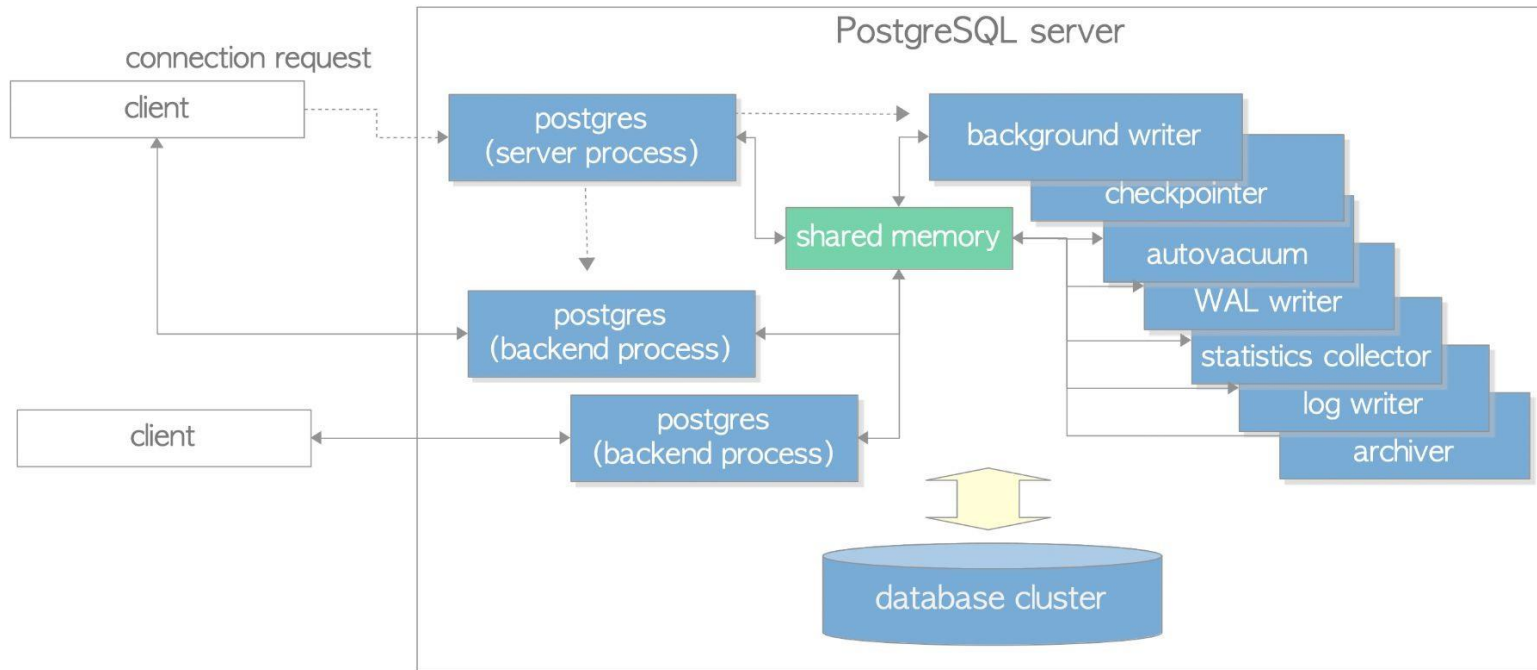
Ставим “+”,  
если вопросы есть



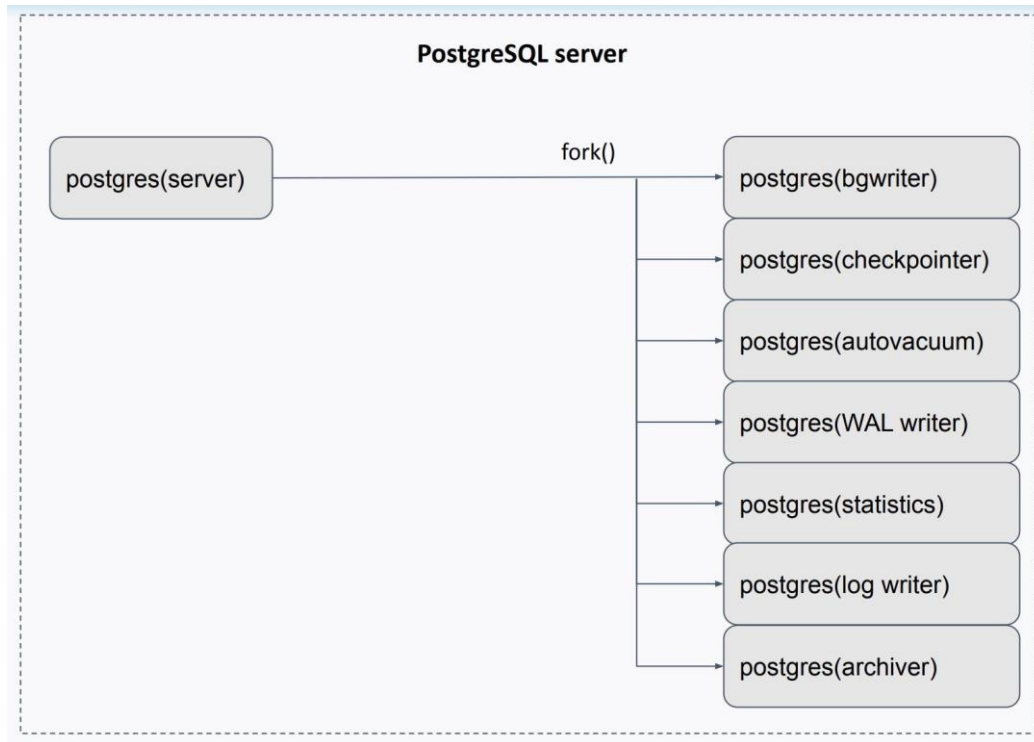
Ставим “-”,  
если вопросов нет

# Серверные процессы и память

# Process Architecture

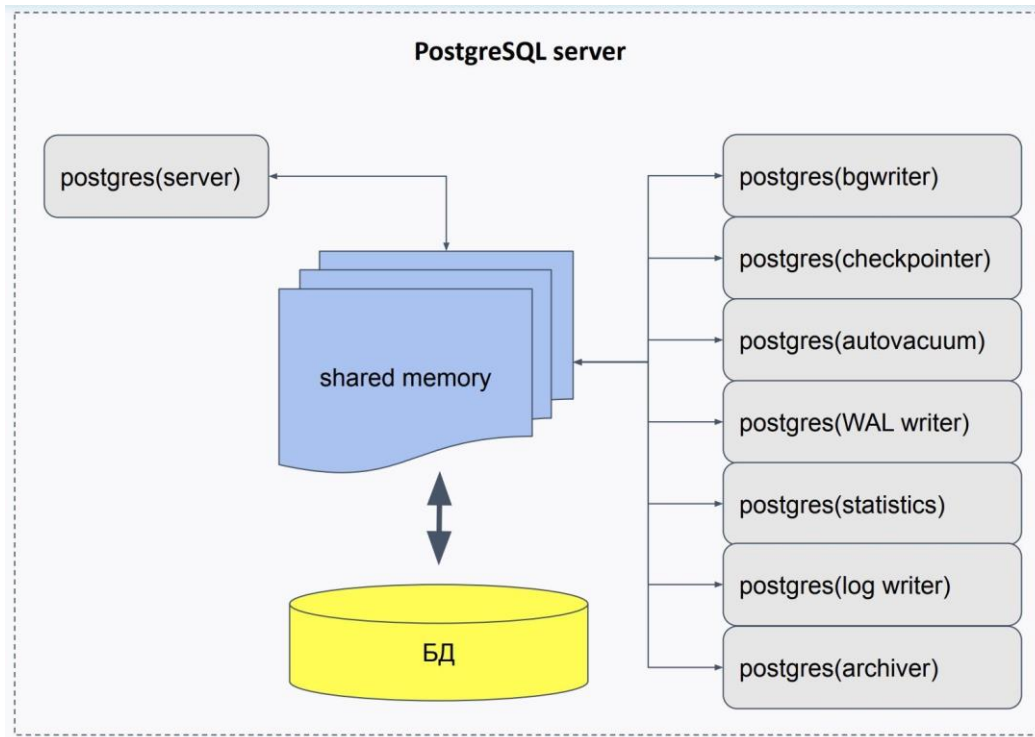


# Process Architecture

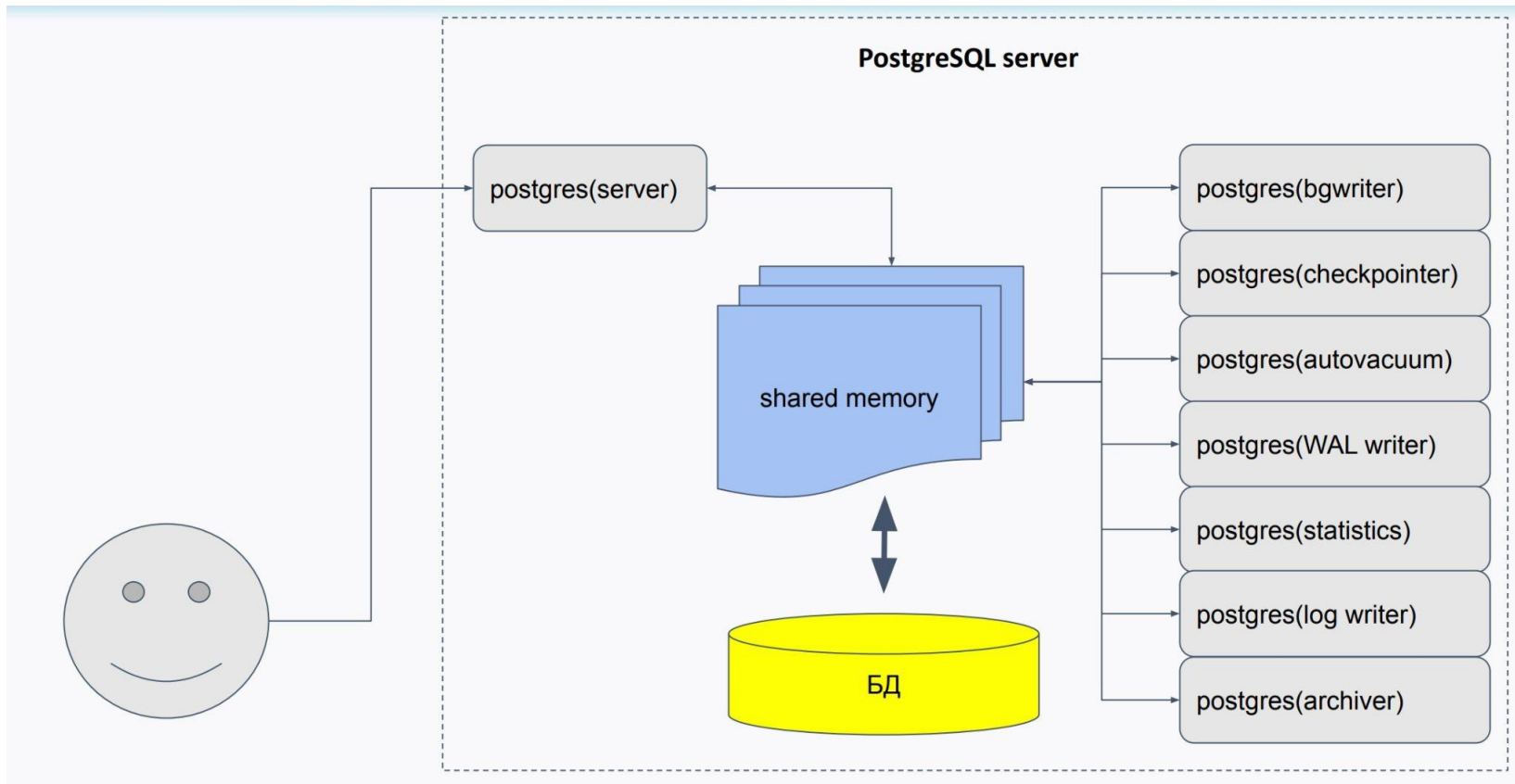




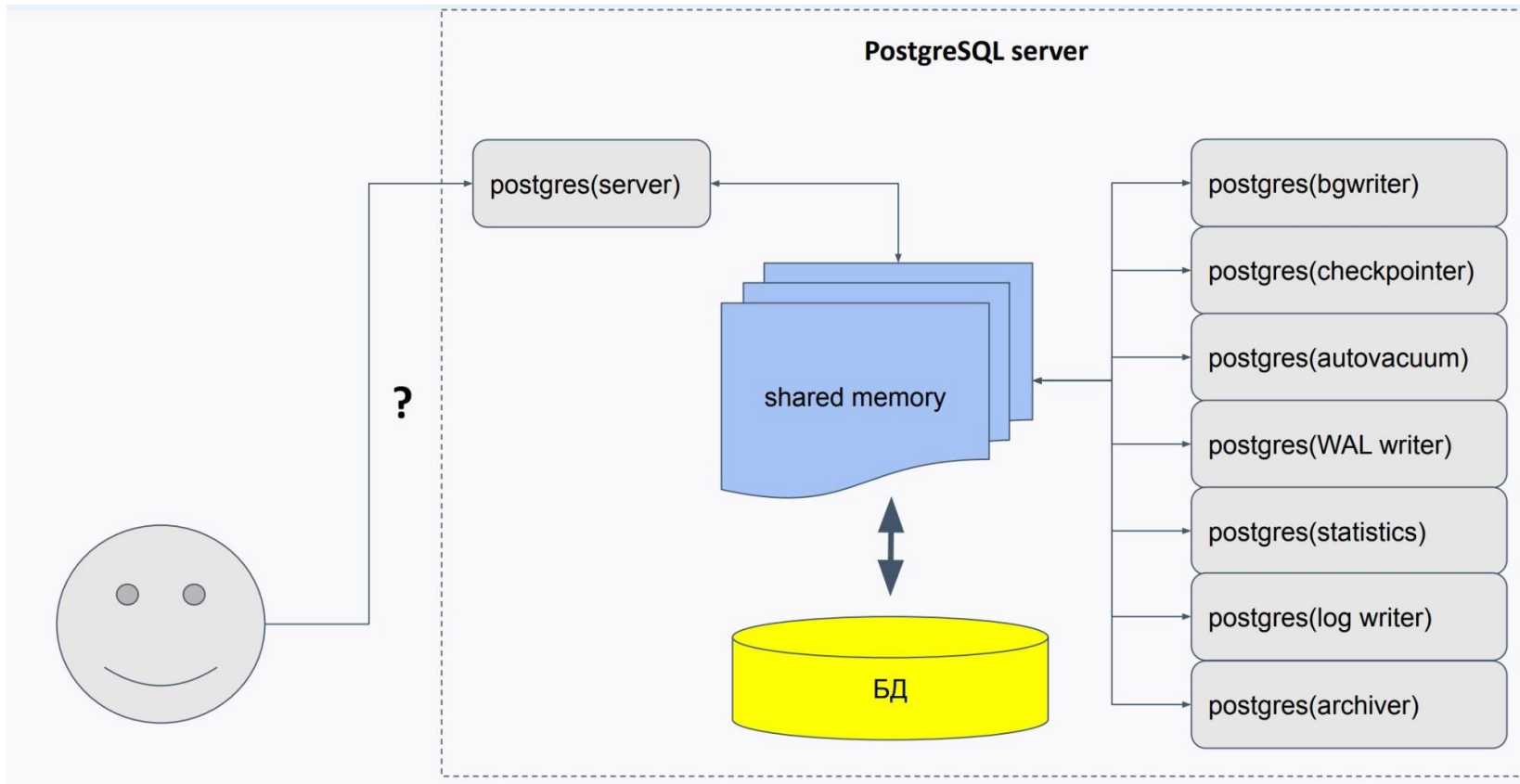
# Process Architecture



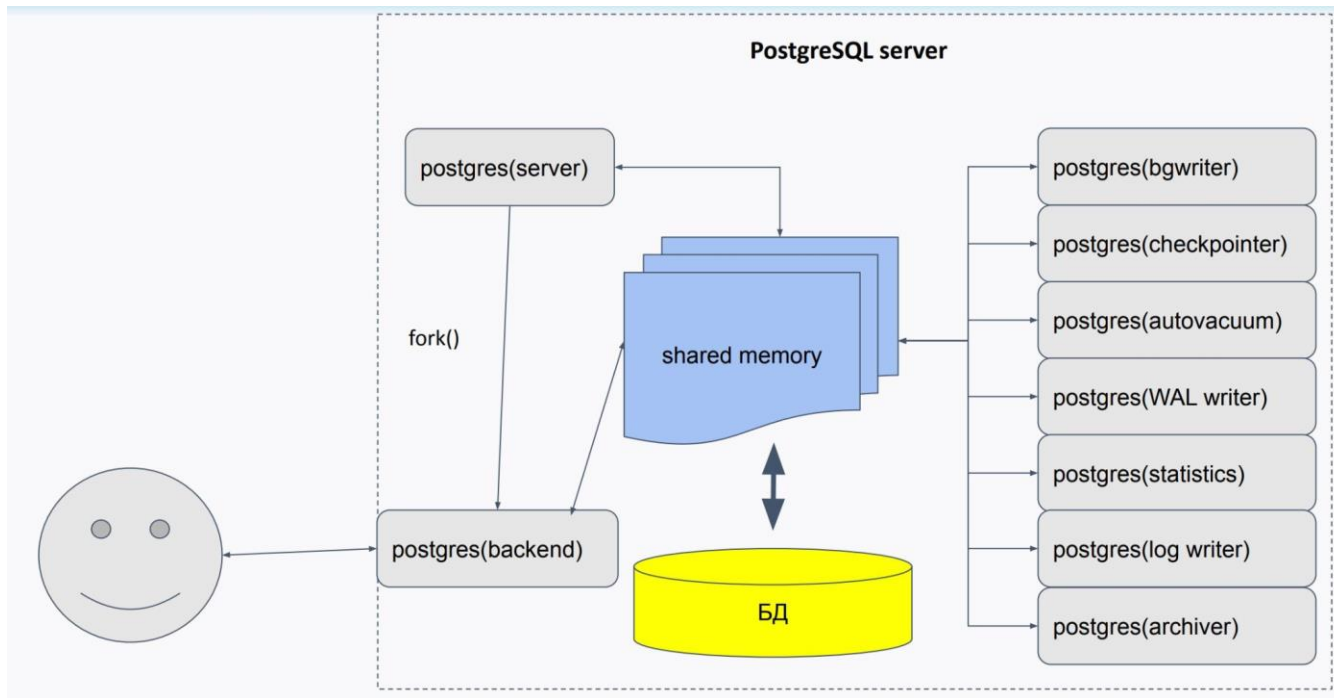
# Process Architecture



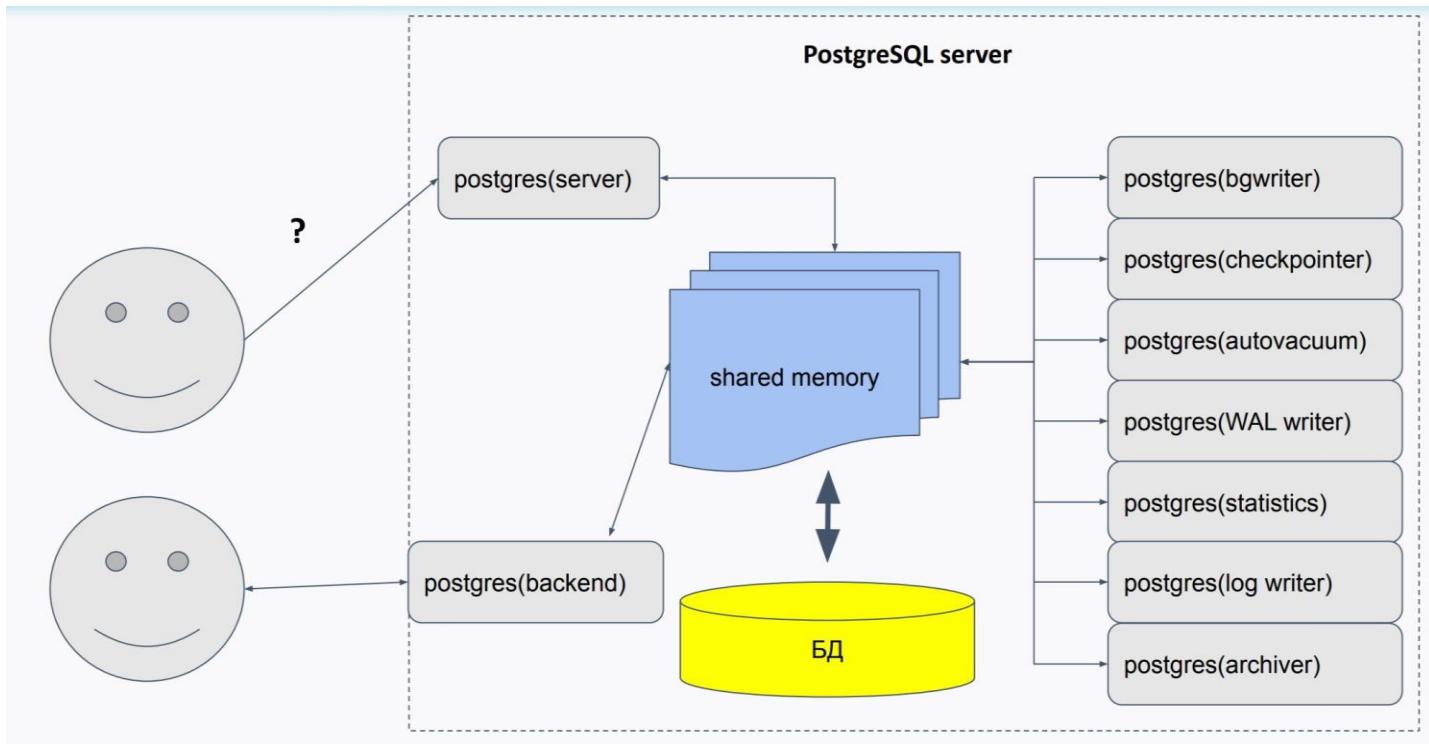
# Process Architecture



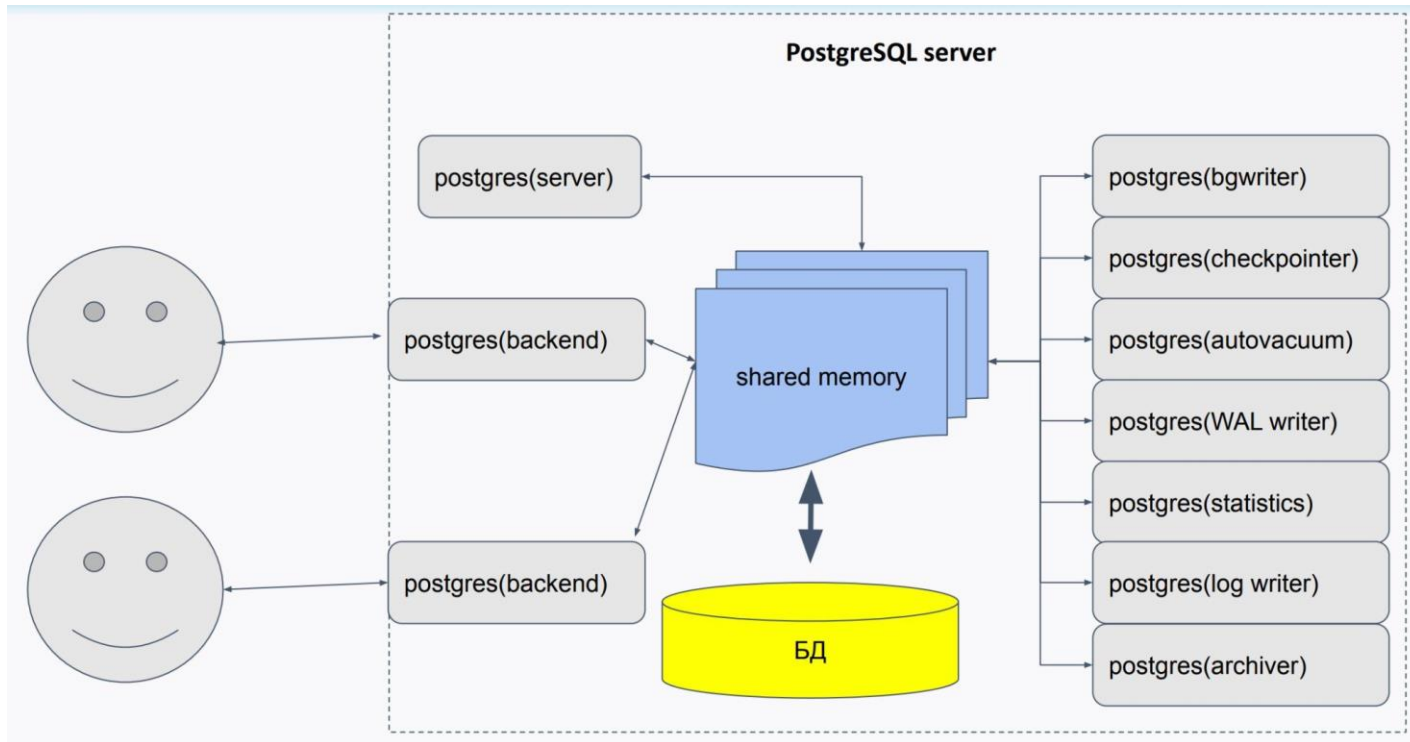
# Process Architecture



# Process Architecture



# Process Architecture



# postmaster

postgres server process (aka postmaster)

- первый процесс postgres
- запускается при старте сервиса
- порождает все остальные процессы
- создает shared memory
- слушает TCP и Unix socket

# backend processes

- запускается postmaster'ом
- обслуживает сессию
- работает пока сессия активна
- максимальное количество определяется параметром **max\_connections** (по умолчанию 100)



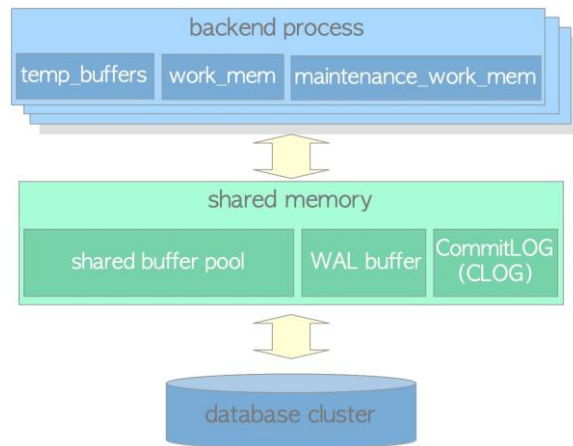
# background processes

- запускаются postmaster'ом
- при старте сервиса
- выделенная роль у каждого процесса
  - logger - запись сообщений в лог файл
  - checkpointer - запись грязных страниц из buffer cache на диск при наступлении checkpoint
  - bgwriter - проактивная запись грязных страниц из buffer cache на диск
  - wal writer - запись wal buffer в wal file
  - Autovacuum launcher - периодический запуск autovacuum
  - archiver - архивация и репликация WAL
  - stats collector - сбор статистики использования по сессиям и таблицам

# Memory Architecture

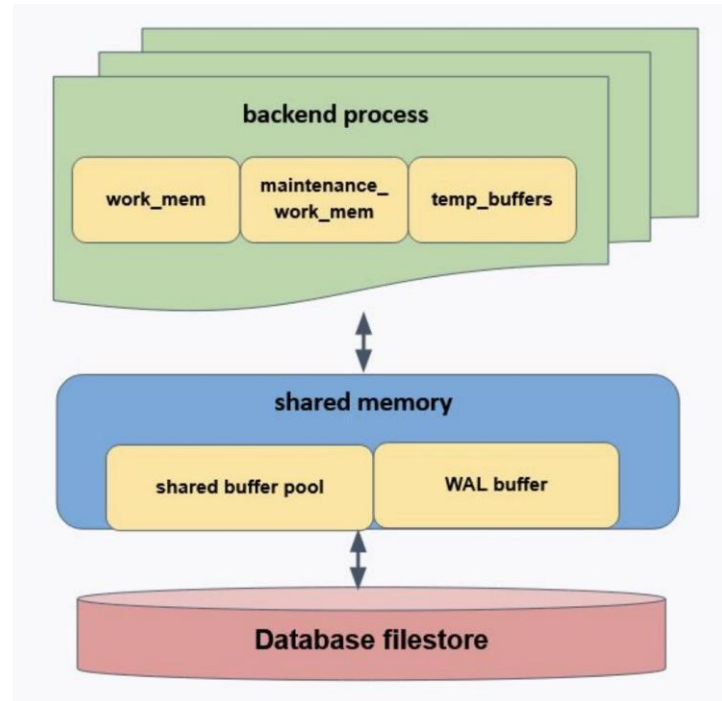
Архитектуру памяти в PostgreSQL можно разделить на две большие категории:

- **Локальная область памяти** - выделяется каждым процессом бэкенда для собственного использования.
- **Общая область памяти** - используется всеми процессами сервера PostgreSQL.



# Память сессии

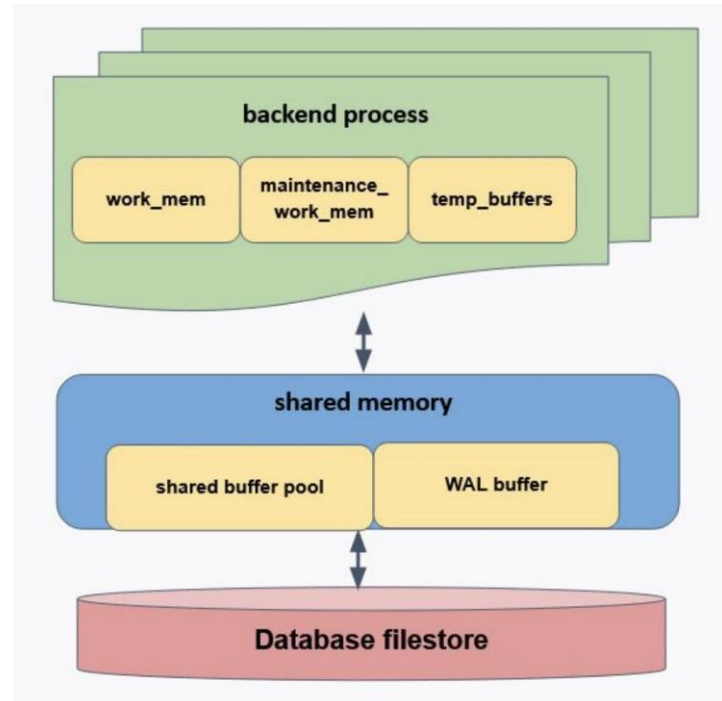
- Принадлежит backend процессу
- **work\_mem (4 MB)**
  - используется на этапе выполнения запроса для сортировок строк, например ORDER BY и DISTINCT
- **maintenance\_work\_mem (64MB)**
  - используется служебными операциями типа VACUUM и REINDEX
  - выделяется только при использовании команд обслуживания в сессии



# Память сессии

- **temp\_buffers (8 MB)**
  - используется на этапе выполнения для хранения временных таблиц

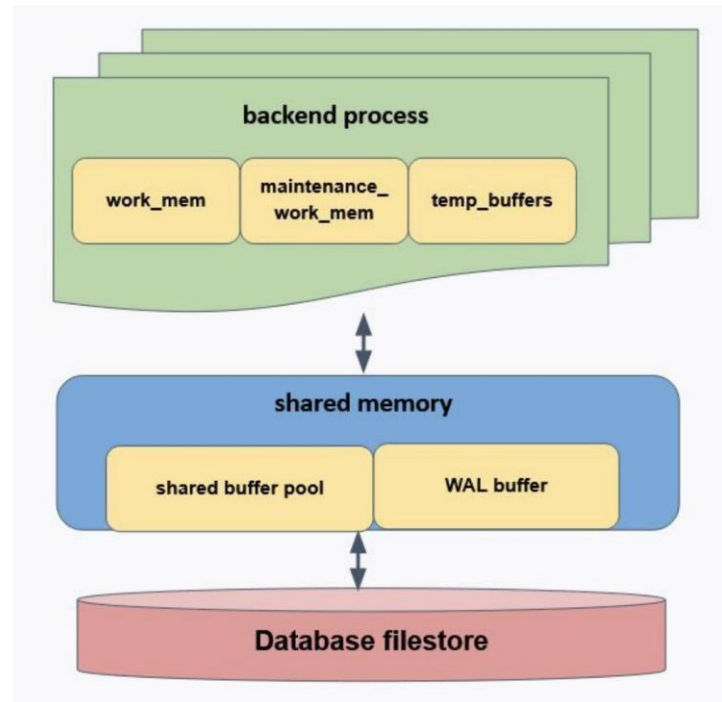
## [Process and memory Architecture](#)



# Кейс - что не так с настройками?

- Памяти у инстанса 4GB
- max\_connections = 1000
- shared\_buffers = 6GB
- work\_mem = 16MB
- maintenance\_work\_mem = 256MB

Периодически приходит Linux OOM killer

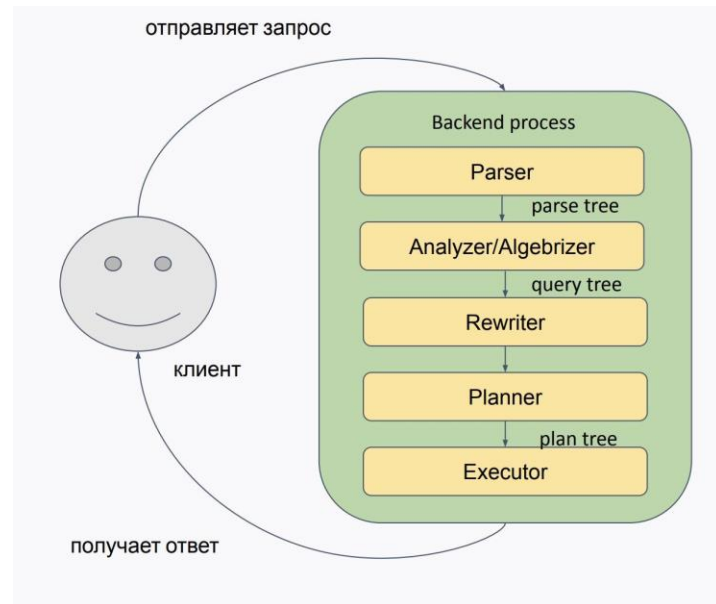


# Что происходит внутри сессии

- Выполняет запрос:
  - Parser
  - Analyser
  - Rewriter
  - Planner
  - Executor

<https://www.postgresql.org/docs/17/rules.html>

Особо управлять не можем(



# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

# Физическая структура данных



# Физическая структура данных

- PostgreSQL работает с данными на дисках только через файловую систему
  - EXT3/4 и XFS наиболее популярны
  - Raw devices не поддерживаются
- **Best practices:**
  - не хранить данные в корневой файловой системе
  - отдельная файловая система для каждого табличного пространства
  - в случае внешнего файлового хранилища - отдельный каталог для каждого табличного пространства

# Табличное пространство

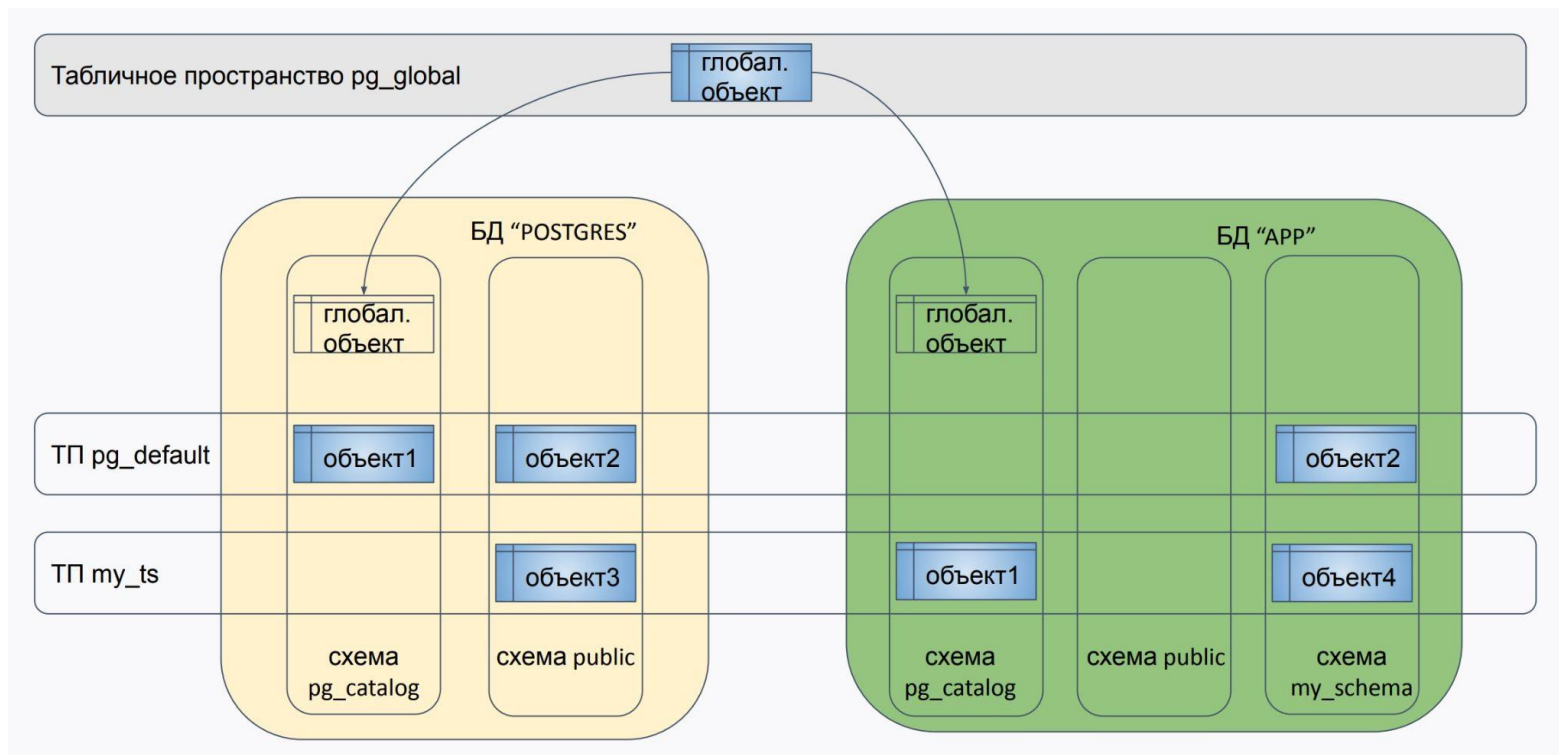
- Отдельный каталог с точки зрения файловой системы
- Лучше делать отдельную файловую систему
- Одно табличное пространство может использоваться несколькими базами данных

## По умолчанию:

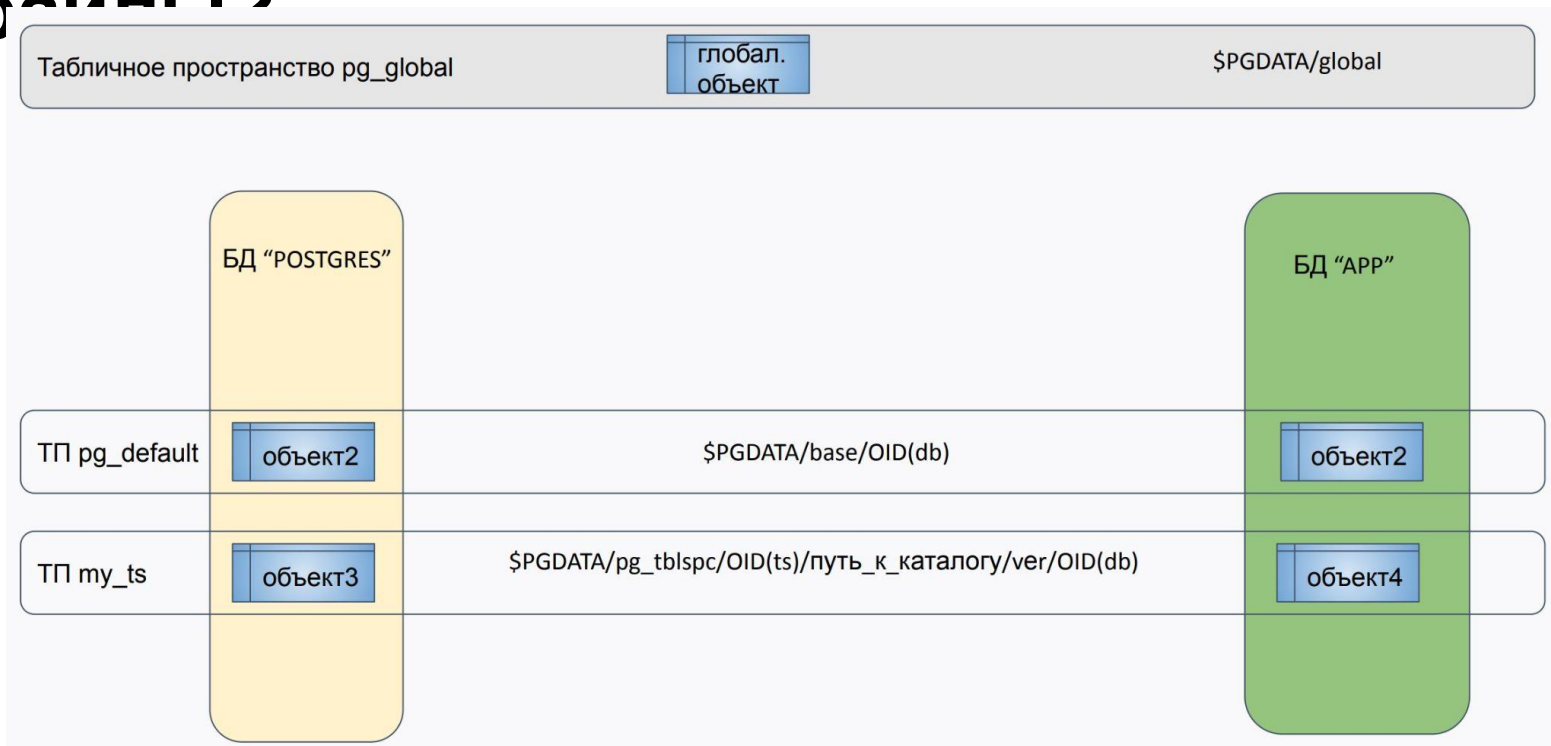
- `# select * from pg_tablespace;`
  - `pg_default` - `$PGDATA/base`
  - `pg_global` - `$PGDATA/global`

Новые табличные пространства создаются в `$PGDATA/pg_tblspc`

# Табличное пространство



# Табличное пространство. Где файлы?





# Tablespaces

Варианты оптимизации производительности:


- самые часто используемые данные на самые быстрые носители
- редко используемые архивы на медленные
- распараллелить нагрузку по разным рейд массивам
- локальный ссд без рейда для материальных представлений

# Кластер PostgreSQL

Несколько баз данных под управлением одного сервера. По умолчанию:

- **template0** 
  - для восстановления из резервной копии
  - по умолчанию даже нет прав на connect
- **template1** 
  - используется как шаблон для создания новых баз данных
  - в нем имеет смысл делать некие действия которые не хочется делать каждый раз при создании новых баз данных (create extension, create schema)

# Кластер PostgreSQL

- postgres 
  - первая база данных для регулярной работы
  - создается по умолчанию
  - хорошая практика - также не использовать, но и не удалять - иногда нужна для различных утилит

# Кластер PostgreSQL

- В качестве шаблона для создания БД можно использовать любую другую БД - при этом возникает соблазн использовать **create database with template** для копирования БД.
- Так делать не рекомендовано, согласно документации:
  - *CREATE DATABASE will fail if any other connection exists when it starts; otherwise, new connections to the template database are locked out until CREATE DATABASE completes*



# Таблицы

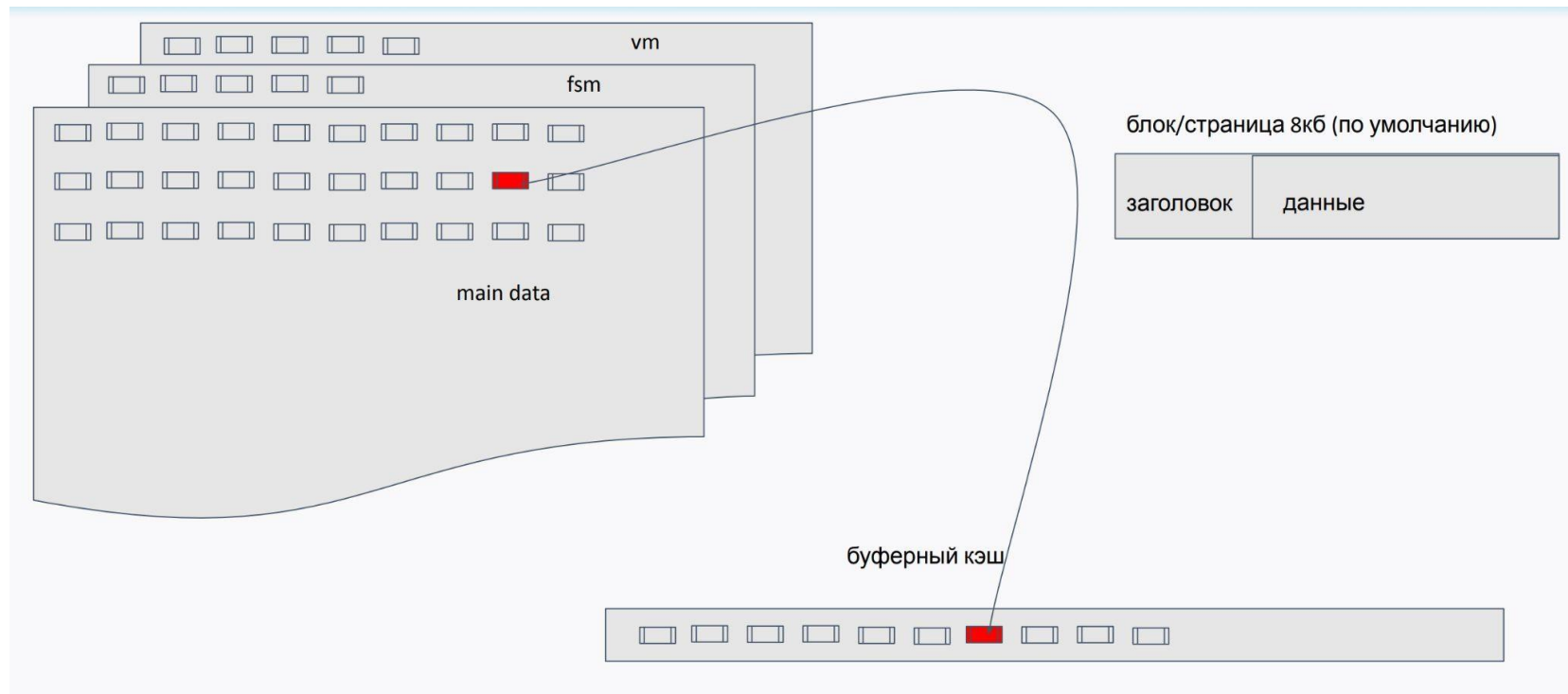
Для каждой таблицы создается до 3-х файлов, каждый до 1 Гб. Если превышает, то создается файл NNN.1 NNN.2 и т.д. Также для FSM и VM:

- **файл с данными - OID таблицы**
- **файл со свободными блоками - OID\_fsm**
  - отмечает свободное пространство в страницах после очистки
  - используется при вставке новых версий строк
  - существует для всех объектов

# Таблицы

- **файл с таблицей видимости - OID\_vm**
  - отмечает страницы, на которых все версии строк видны во всех снимках
  - используется для оптимизации работы процесса очистки и ускорения индексного доступа
  - существует только для таблиц
  - иными словами, это страницы, которые давно не изменялись и успели полностью очиститься от неактуальных версий

# Таблицы



# TOAST

- Версия строки должна помещаться на одну страницу
  - можно сжать часть атрибутов,
  - или вынести в отдельную TOAST-таблицу,
  - или сжать и вынести одновременно

# TOAST

- TOAST-таблица
  - схема pg\_toast
  - поддержана собственным индексом
  - «длинные» атрибуты разделены на части размером меньше страницы
  - читается только при обращении к «длинному» атрибуту
  - собственная версионность (если при обновлении toast часть не меняется, то и не будет создана новая версия toast части)
  - работает прозрачно для приложения
  - стоит задуматься, когда пишем `select *`

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет



# Рефлексия

Спасибо за внимание!

# Приходите на следующие вебинары



**Тоескин Игорь**

**Ведущий разработчик СУБД**

Специалист в области разработки и проектировании витрин данных в PostgreSQL, а также в области разработки хранимых процедур в таких СУБД как PostgreSQL и Oracle