



# Архитектура PostgreSQL



Взаимодействие процессов

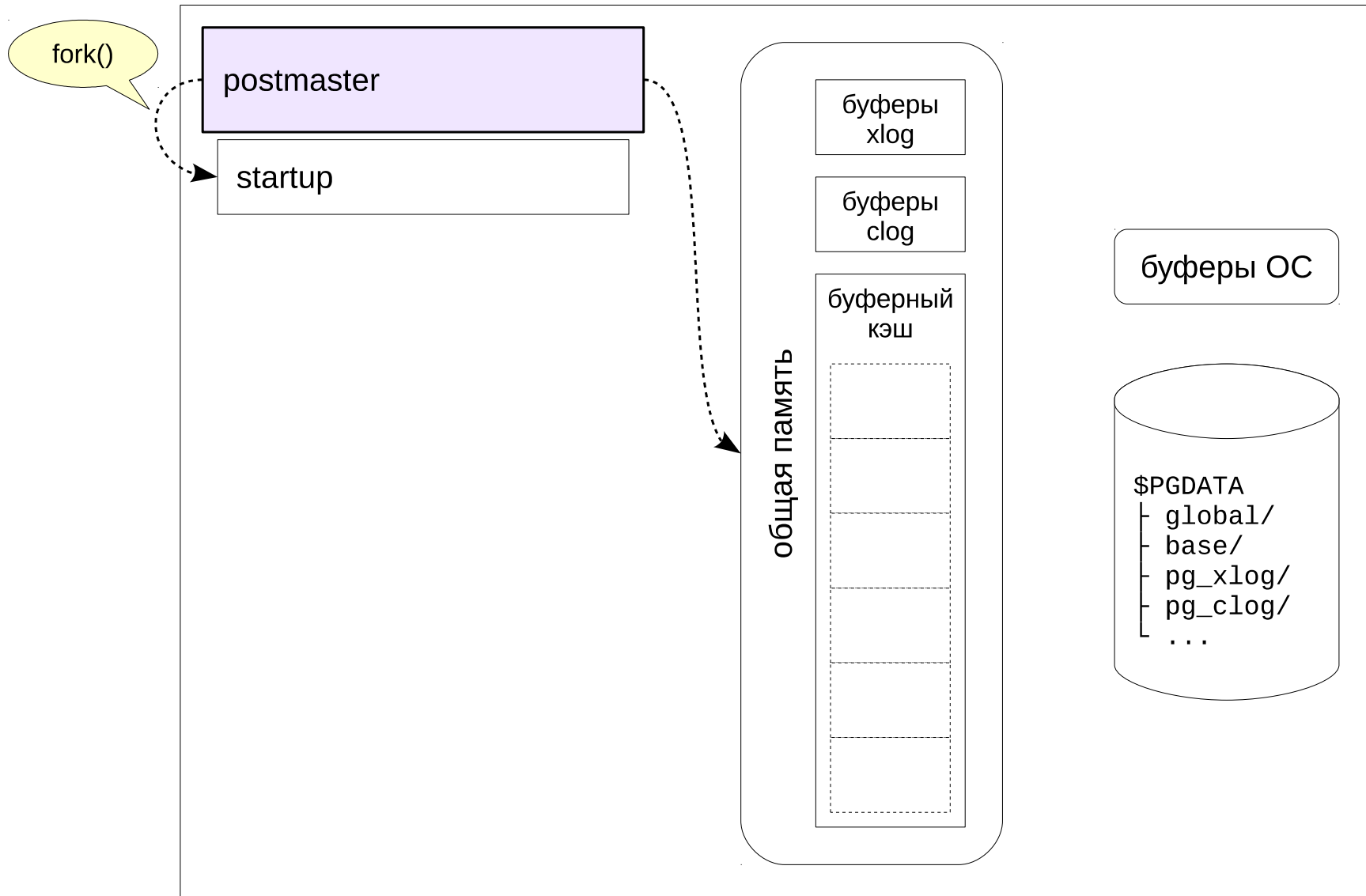
Структуры памяти

Подключение клиентов и выполнение запросов

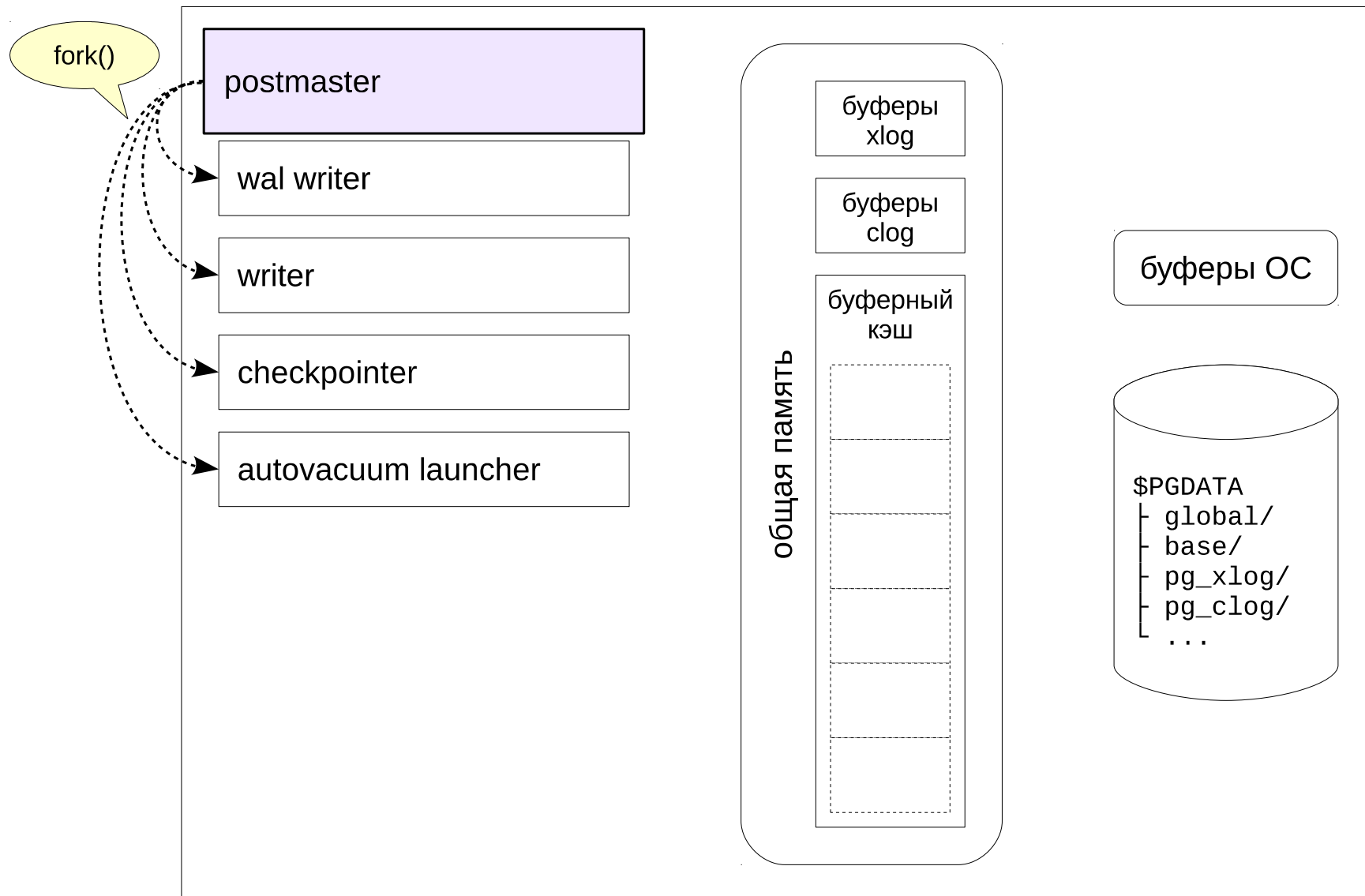
Механизмы обеспечения транзакционности

Процессы записи журналов, записи буферов,  
выполнения контрольной точки, автоочистки

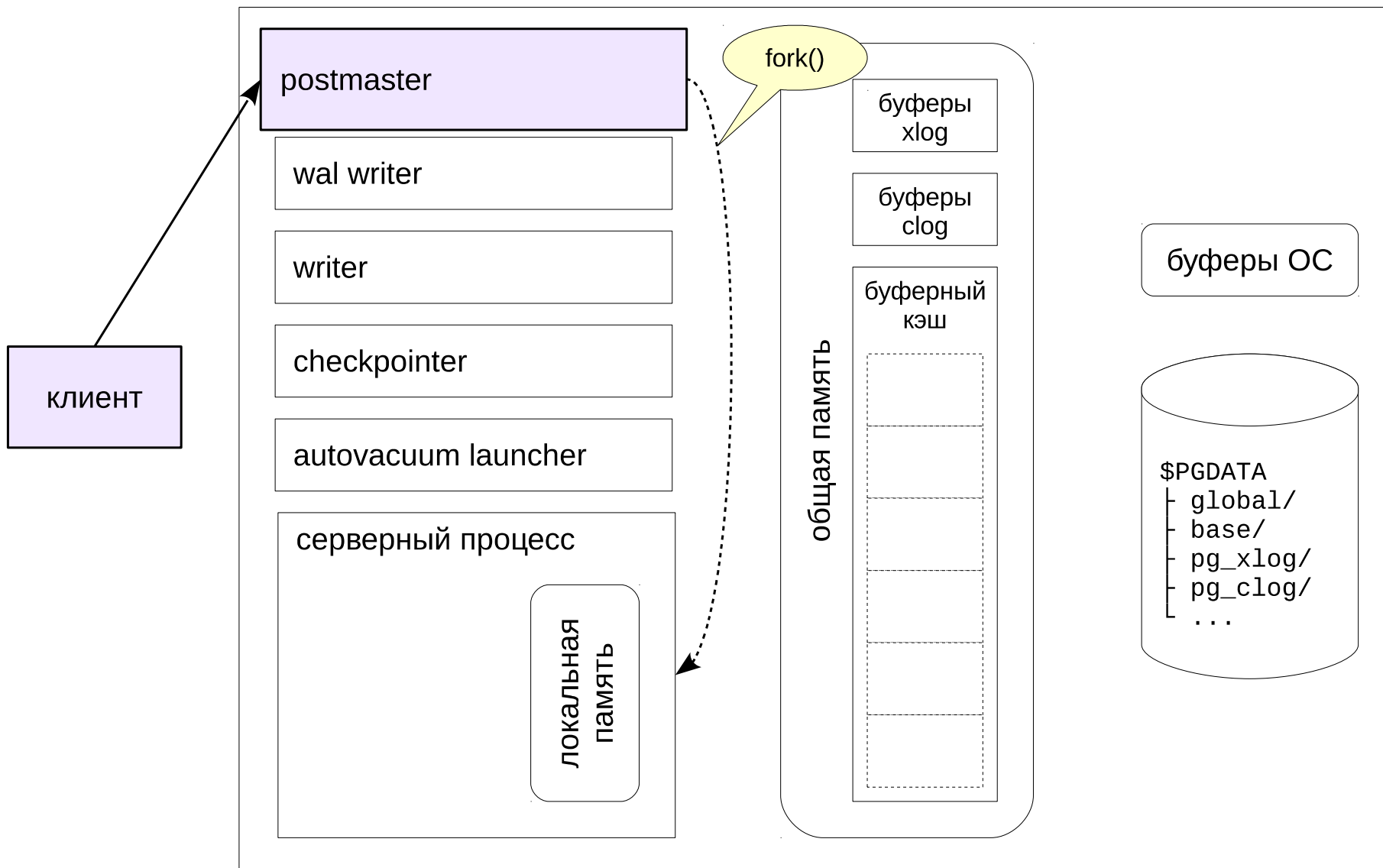
# Запуск сервера



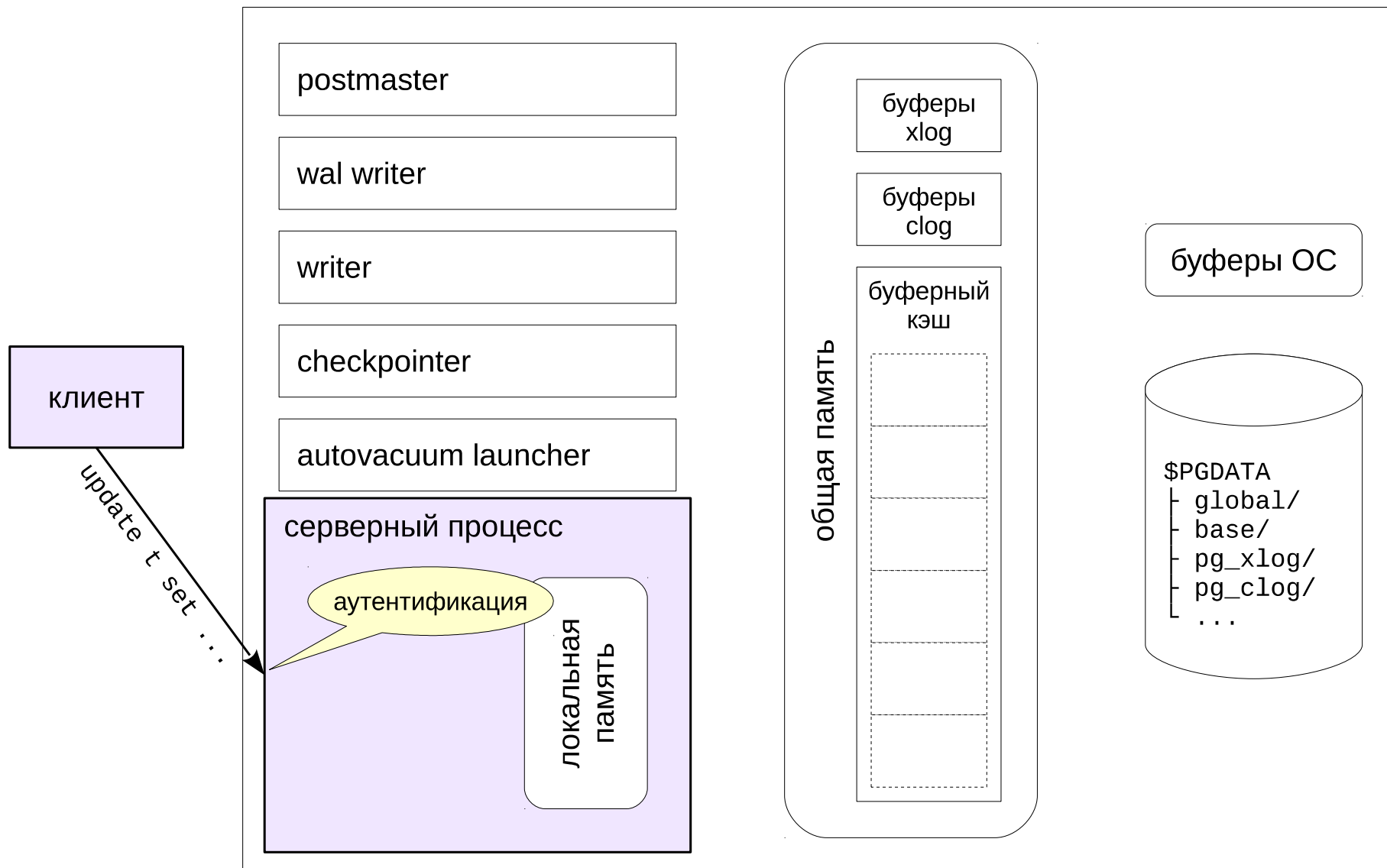
# Запуск сервера



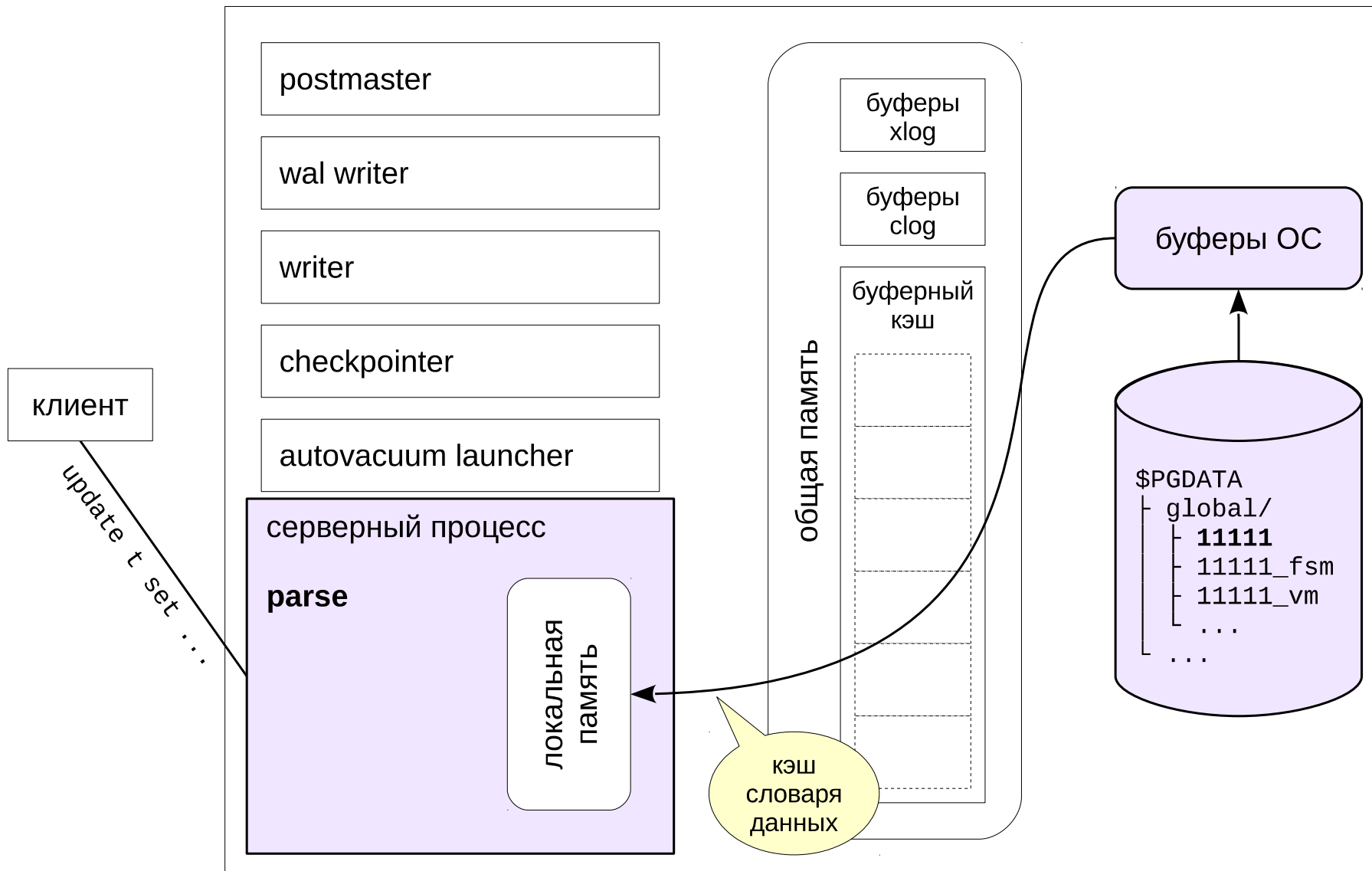
# Подключение клиента



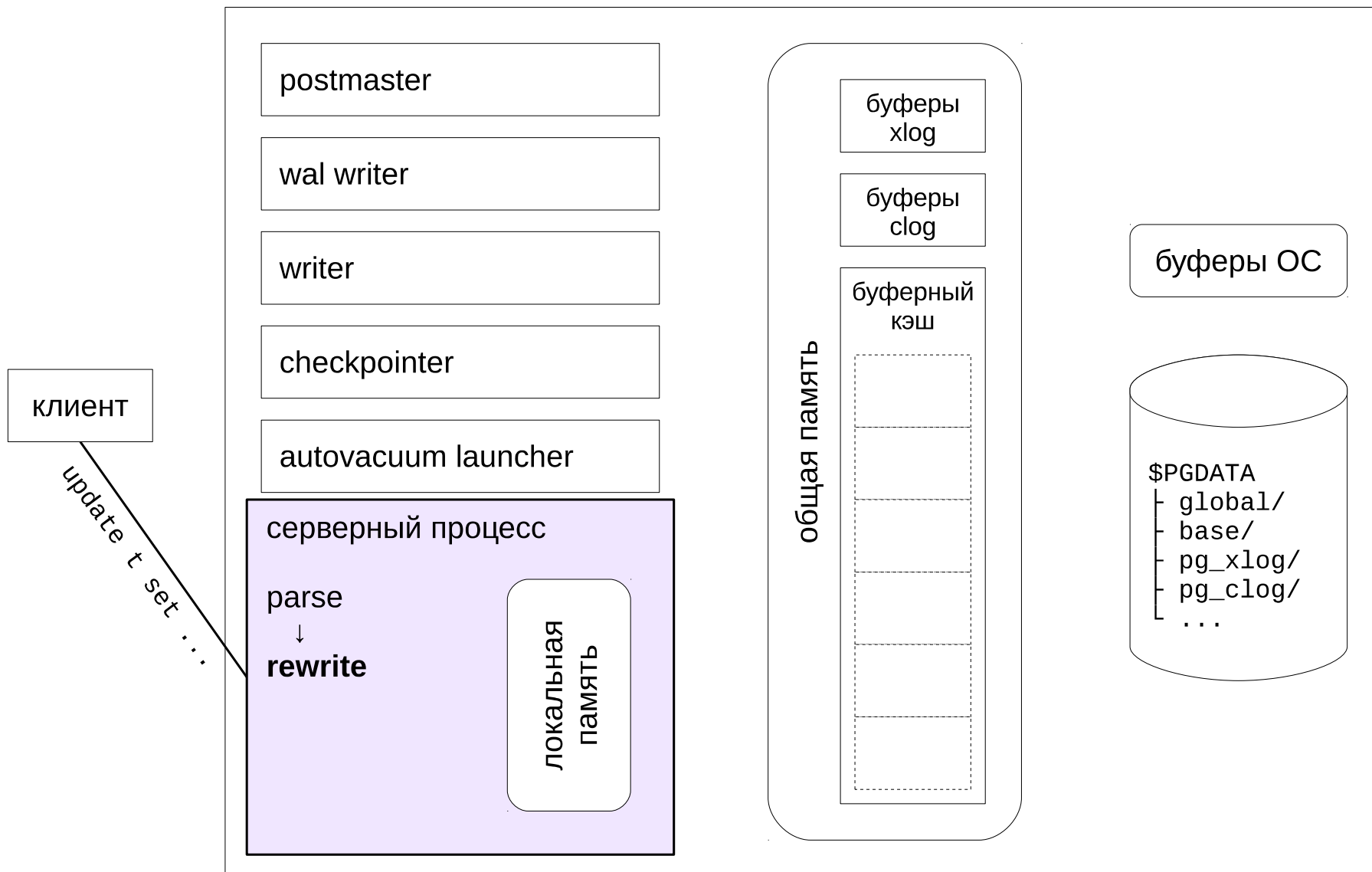
# Аутентификация



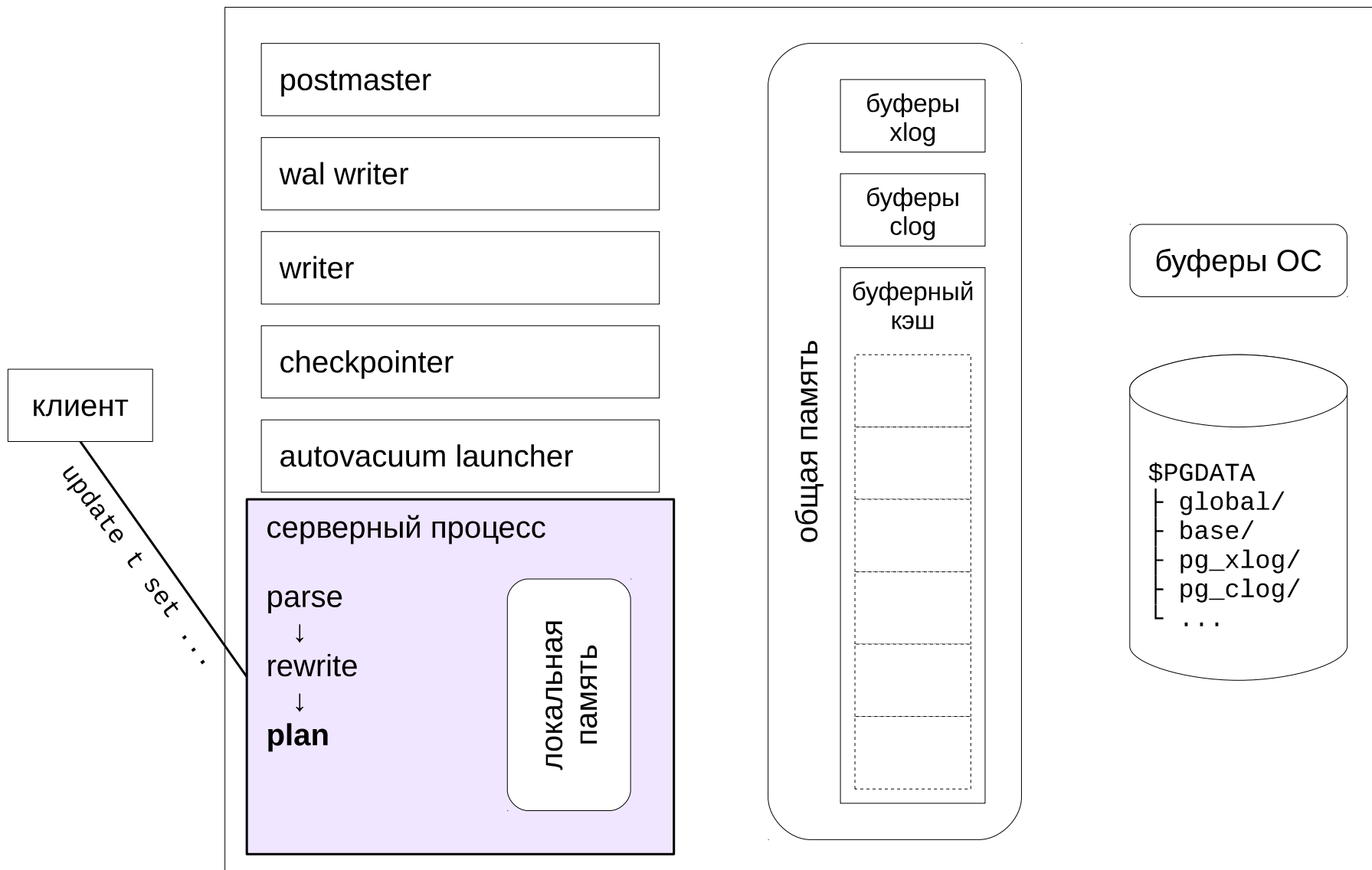
# Выполнение запроса



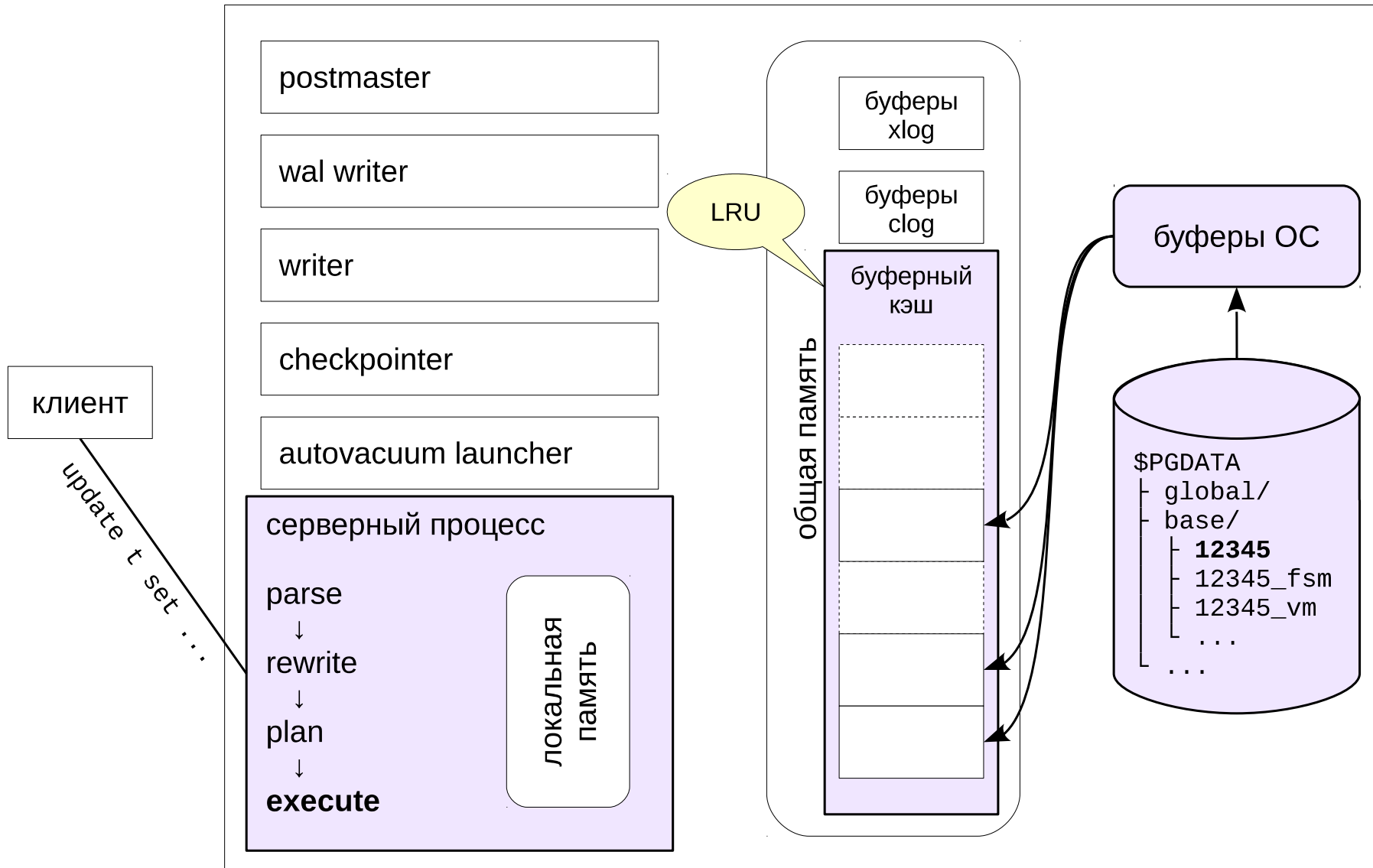
# Выполнение запроса



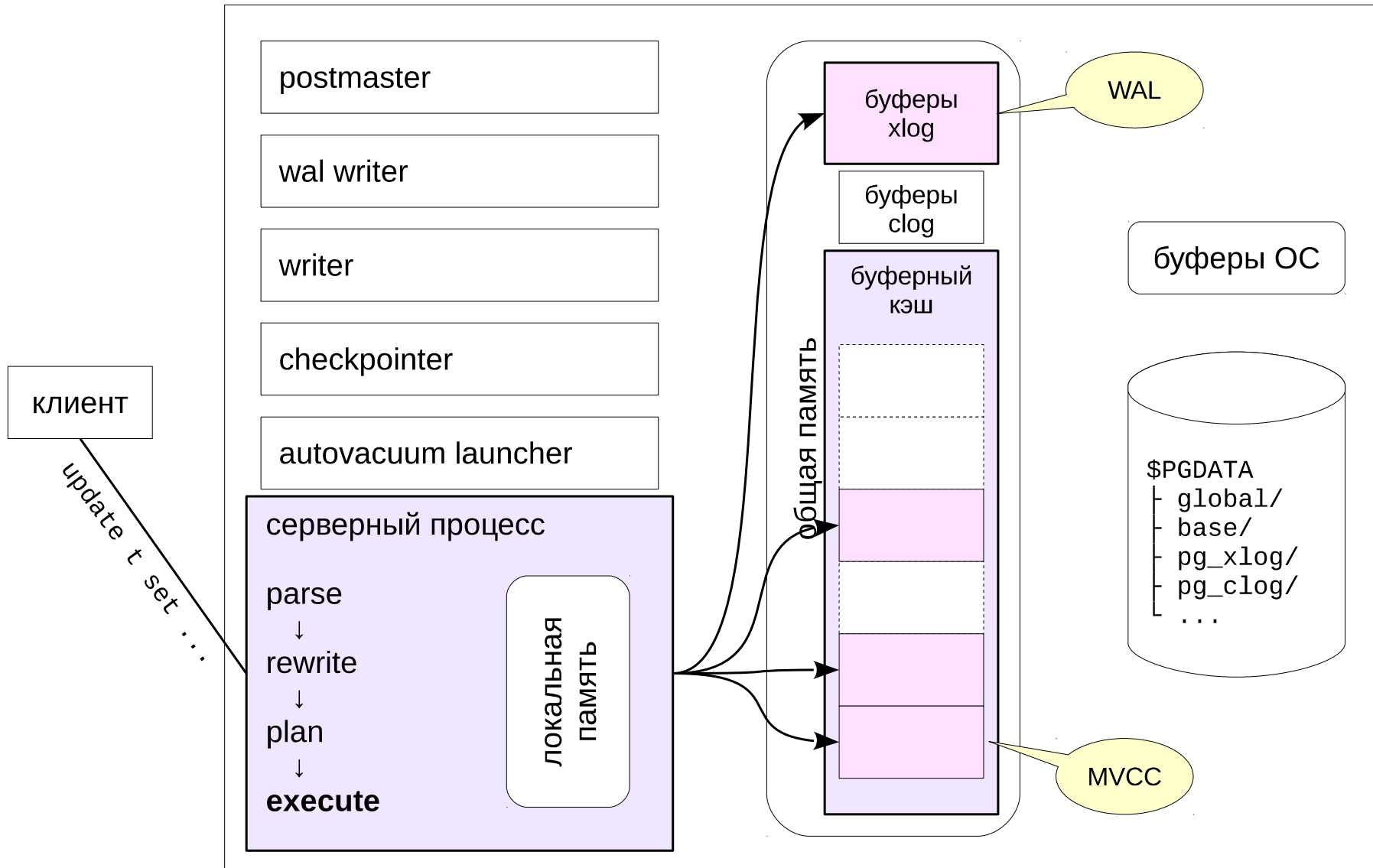
# Выполнение запроса



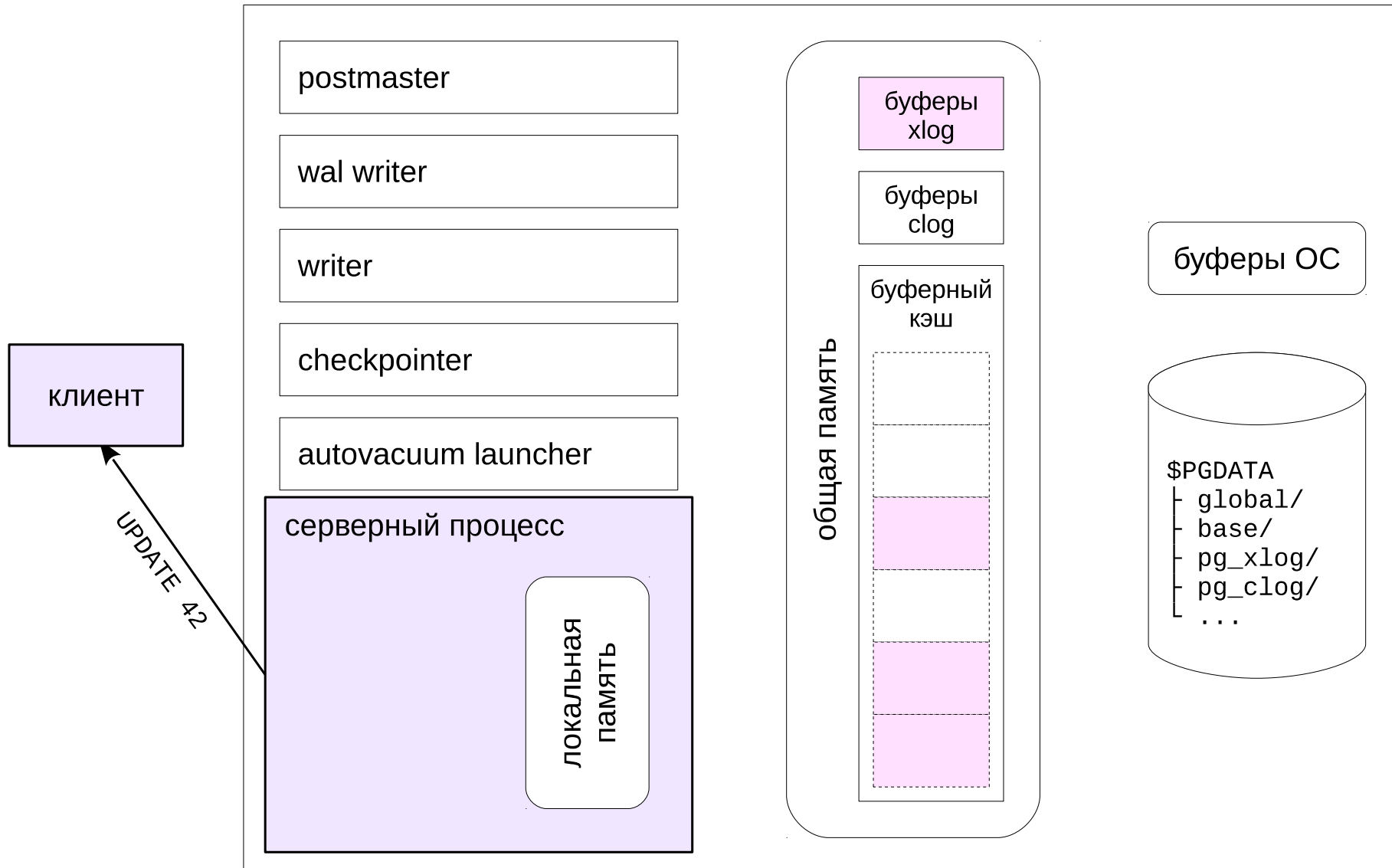
# Выполнение запроса



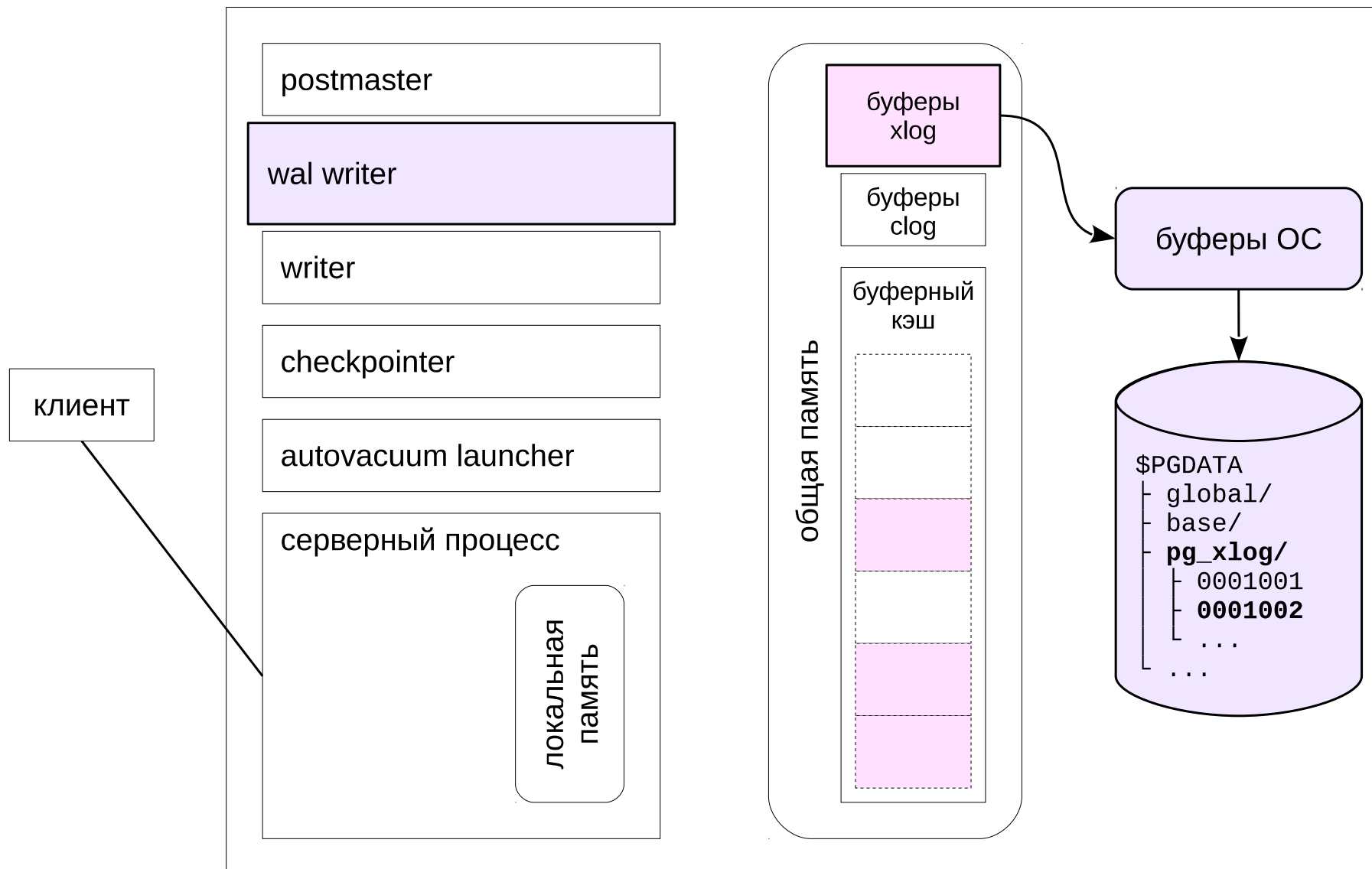
# Выполнение запроса



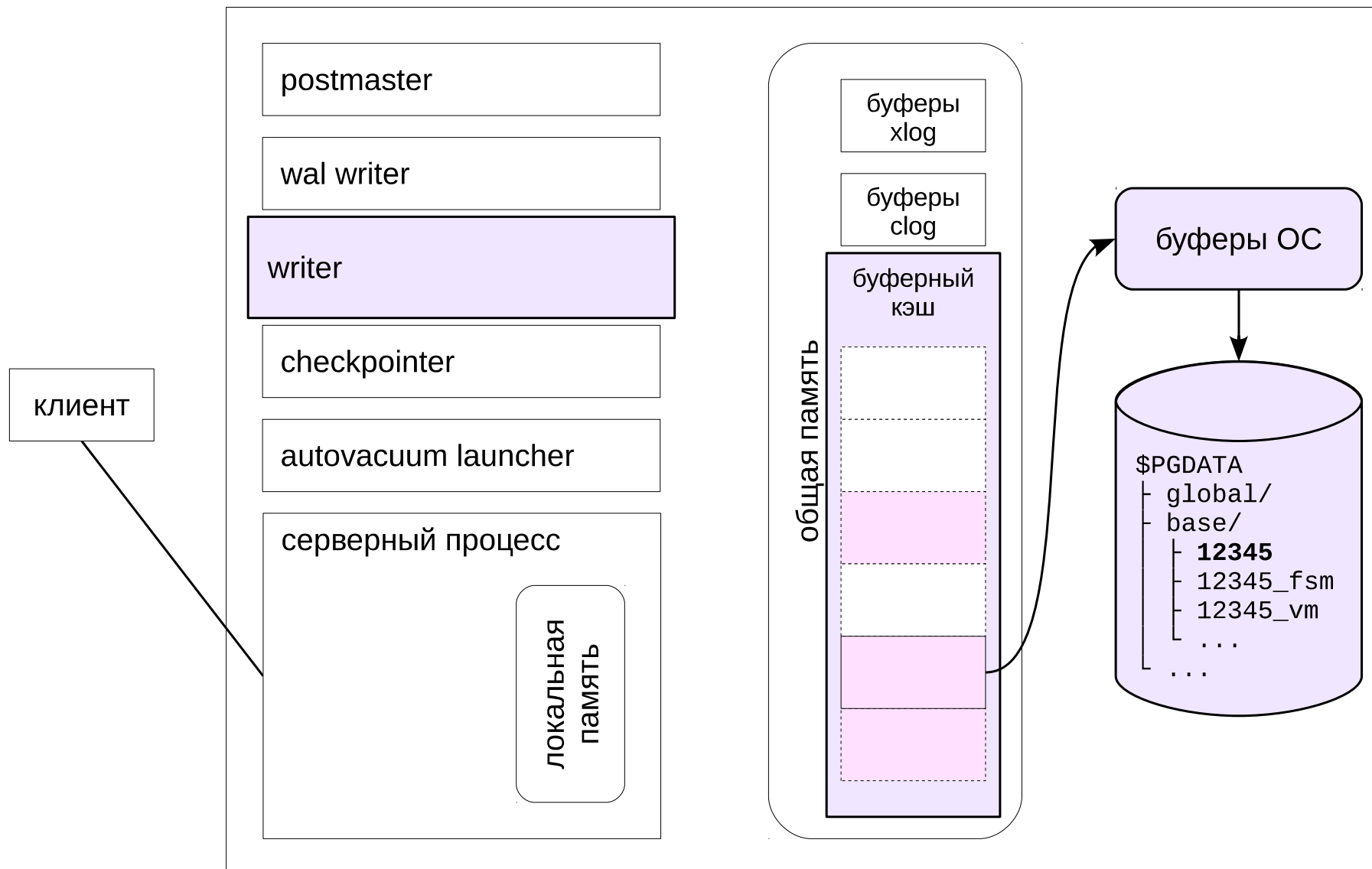
# Выполнение запроса



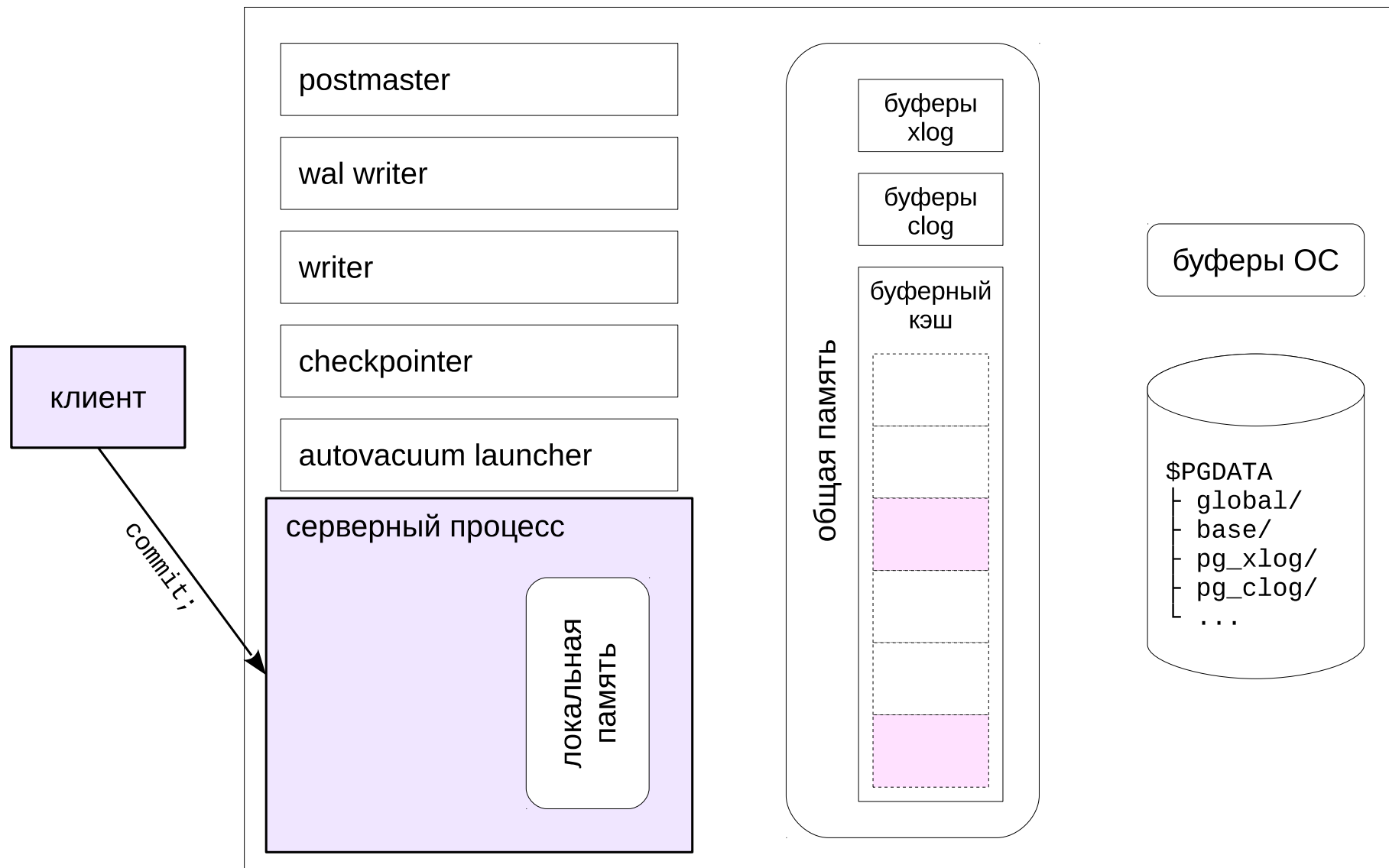
# Процесс записи журналов



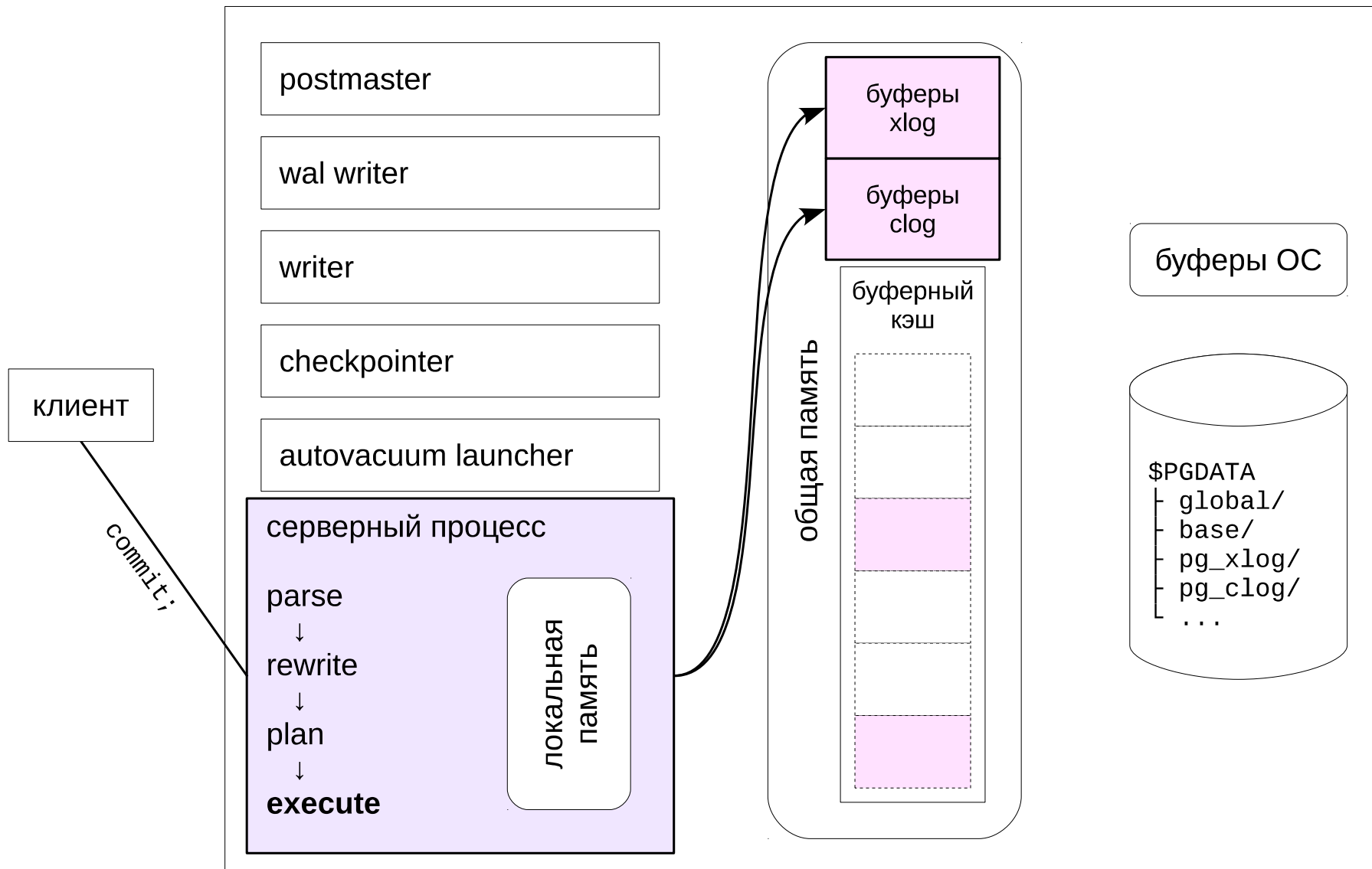
# Процесс записи буферов



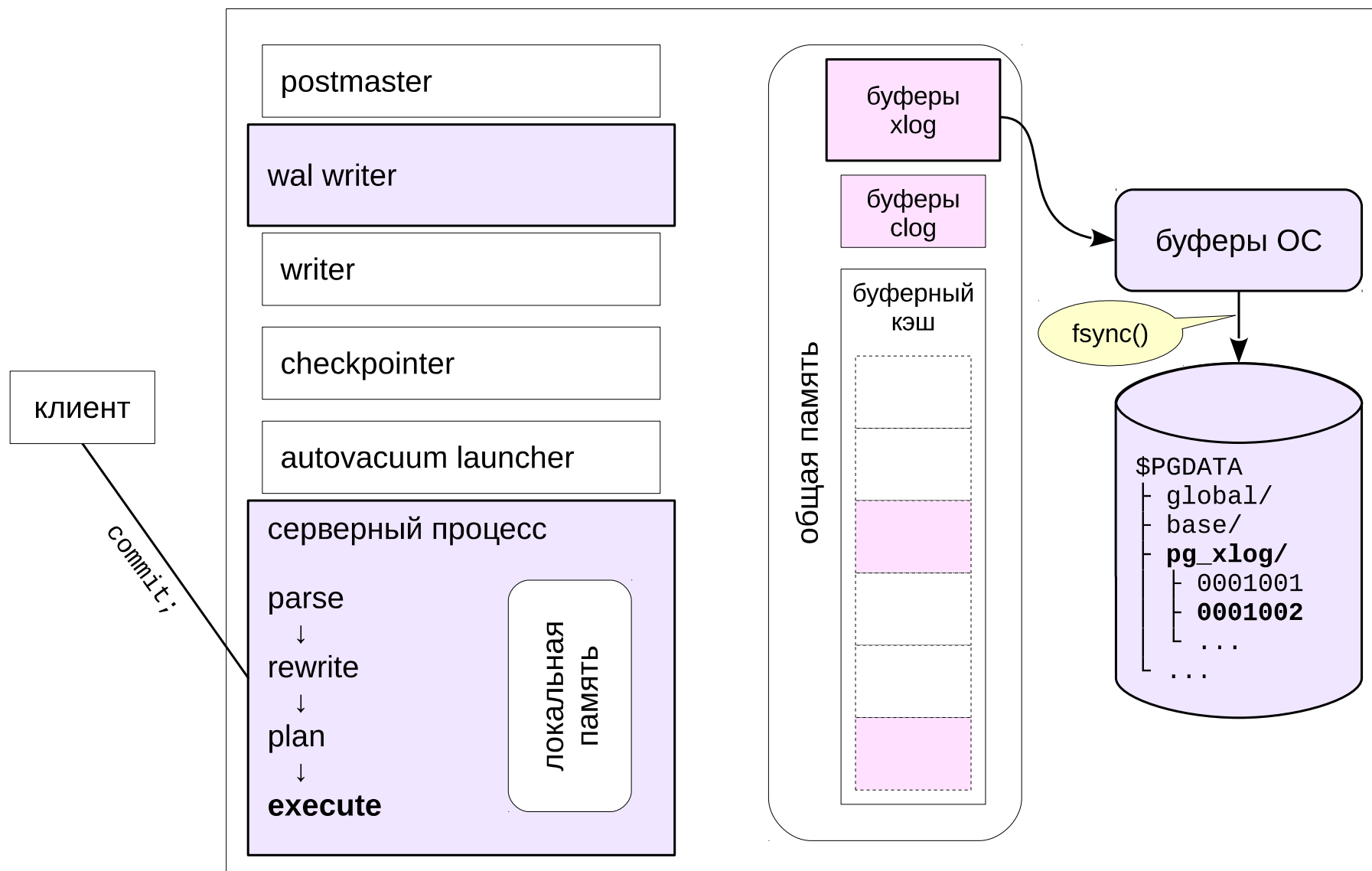
# Фиксация изменений



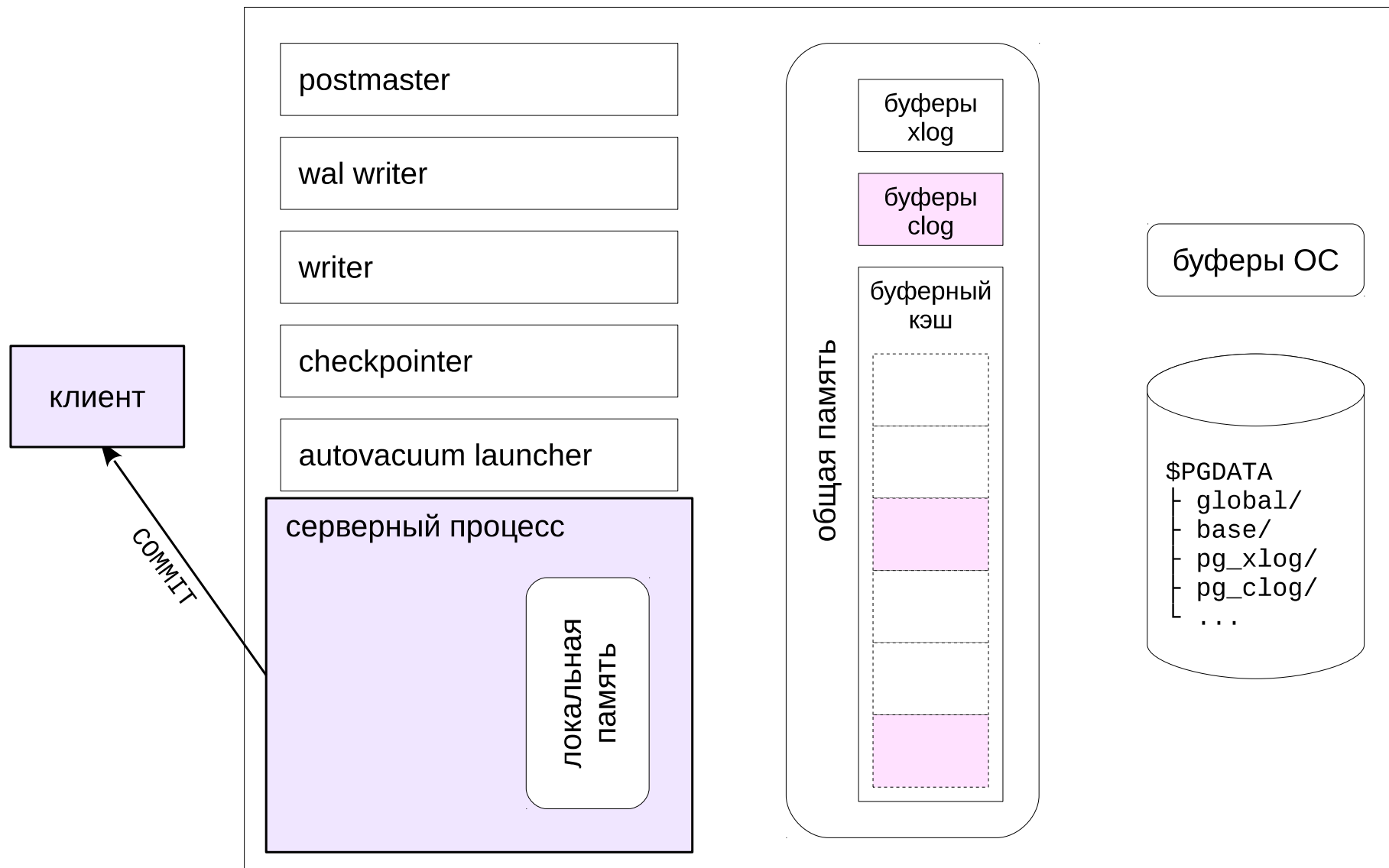
# Фиксация изменений



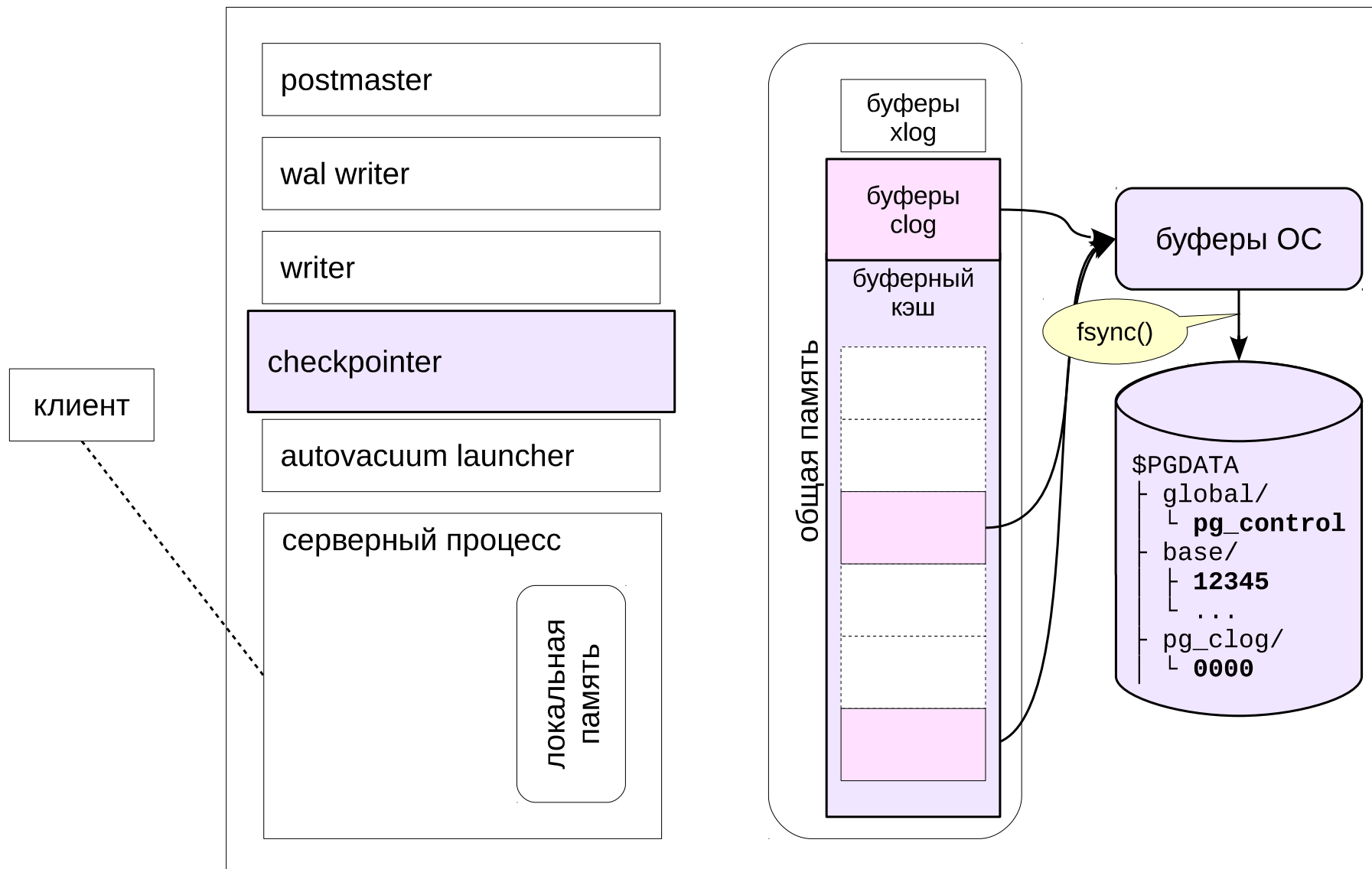
# Фиксация изменений



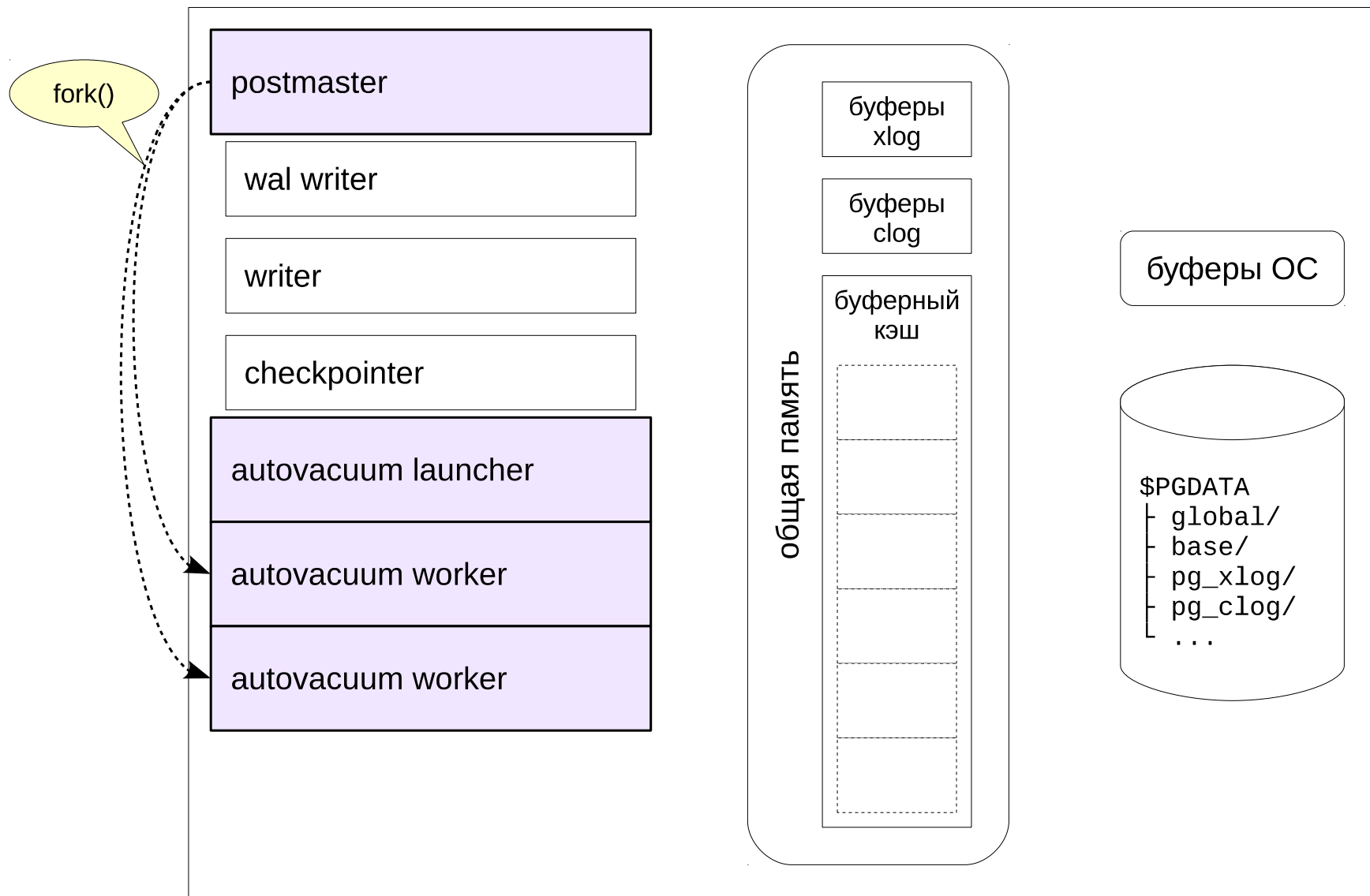
# Фиксация изменений



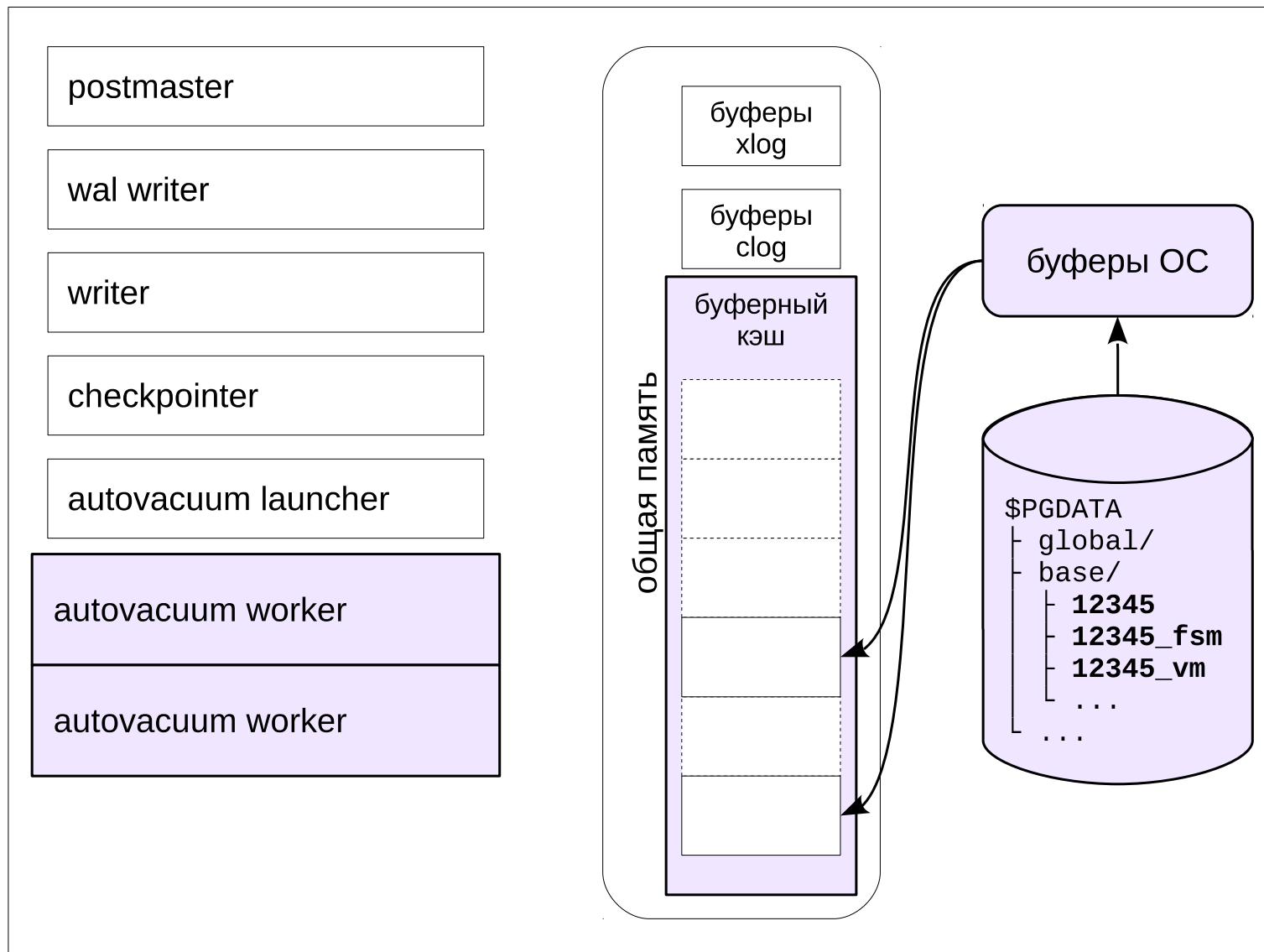
# Контрольная точка



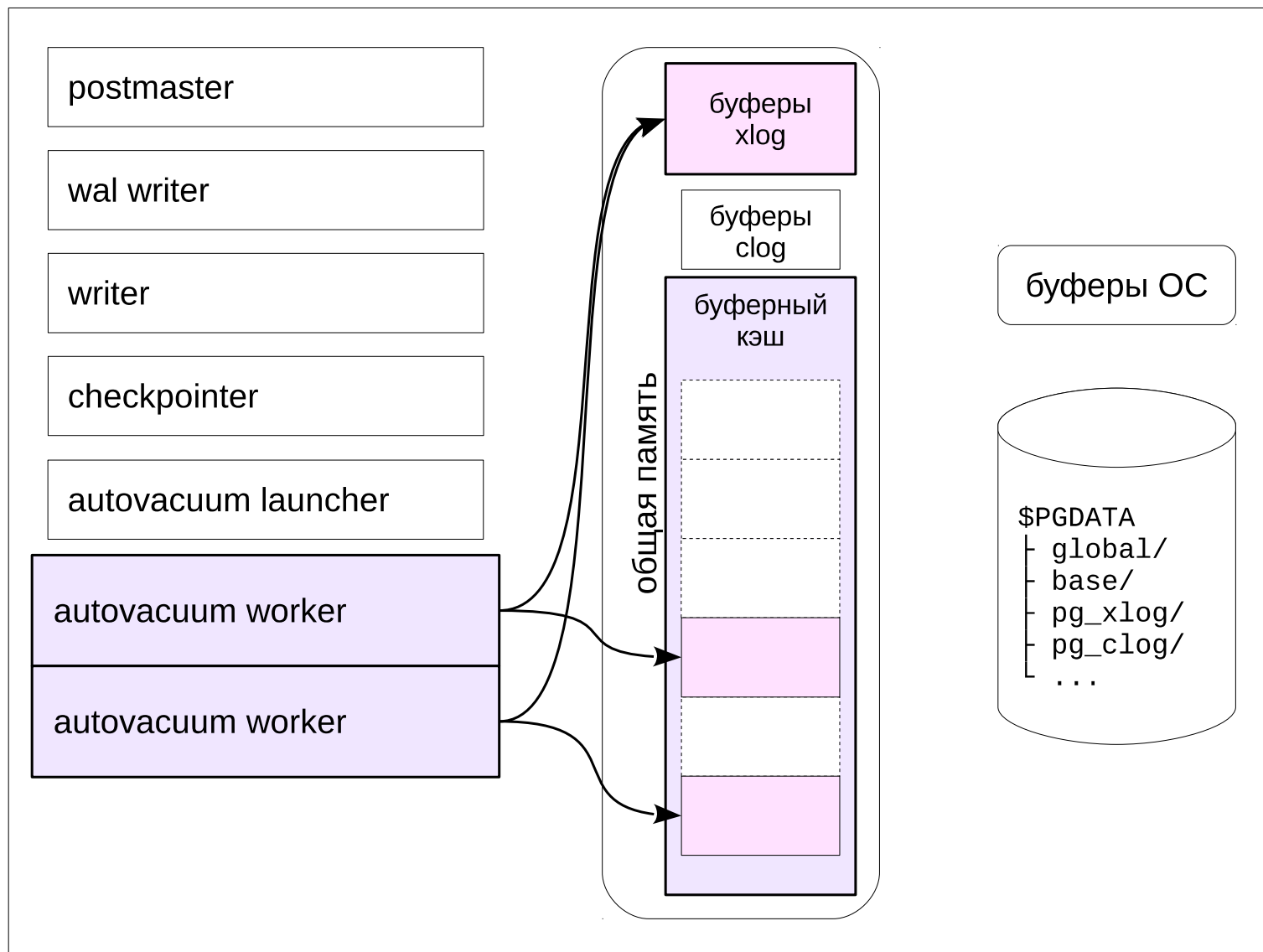
# Процесс автоочистки



# Процесс автоочистки



# Процесс автоочистки



## Архитектурно PostgreSQL:

взаимодействующие процессы

один клиент — один обслуживающий процесс

локальная и общая память, кэширование

внешняя память под управлением ОС

Большое количество фоновых процессов требуют настройки

1. Под пользователем postgres установлен сервер СУБД, создан кластер и настроена переменная окружения PGDATA. Проверьте, запущен ли сервер; если нет — запустите.
2. В каком каталоге находятся файлы кластера?
3. В каком каталоге находятся исполняемые файлы?
4. Найдите номер процесса postmaster.
5. Посмотрите процессы, порожденные postmaster, средствами операционной системы. Понятно ли назначение всех процессов?



### **Авторские права**

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 год.

Авторы: Егор Рогов, Павел Лузанов

### **Использование материалов курса**

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

### **Обратная связь**

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

### **Отказ от ответственности**

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Взаимодействие процессов

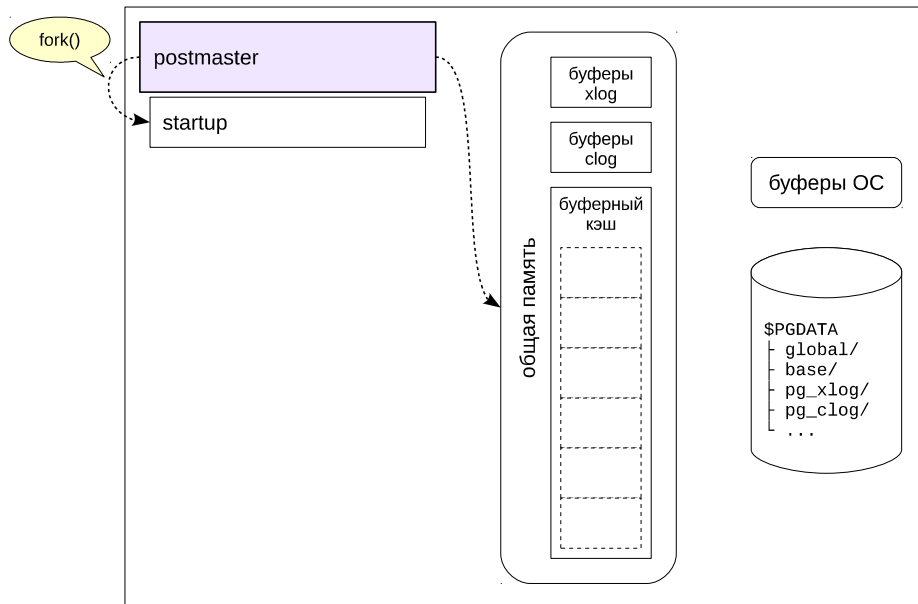
Структуры памяти

Подключение клиентов и выполнение запросов

Механизмы обеспечения транзакционности

Процессы записи журналов, записи буферов,  
выполнения контрольной точки, автоочистки

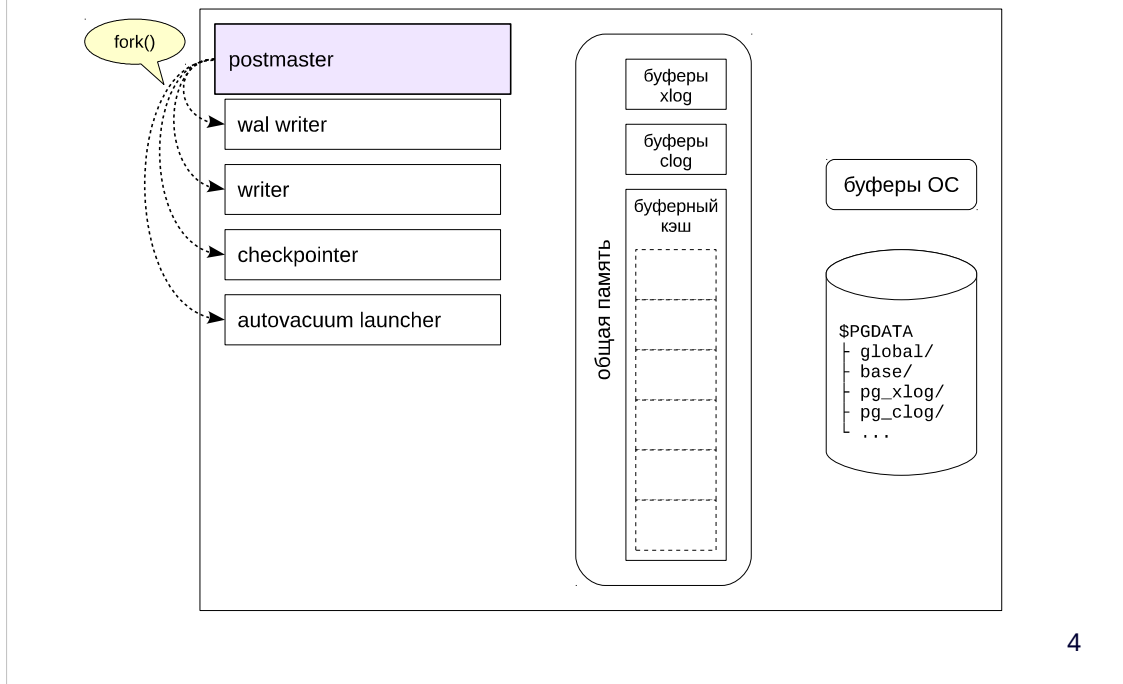
# Запуск сервера



3

Экземпляр PostgreSQL работает над базами данных (кластер БД), расположенными во внешней памяти.

Работу экземпляра обеспечивают несколько процессов. При старте сервера первым запускается основной процесс, называемый postmaster. Он выделяет память под структуры, разделяемые процессами (общая память), и запускает процесс startup. Этот процесс выполняет начальную инициализацию и, если необходимо, восстановление после сбоя. После этого процесс startup завершается.

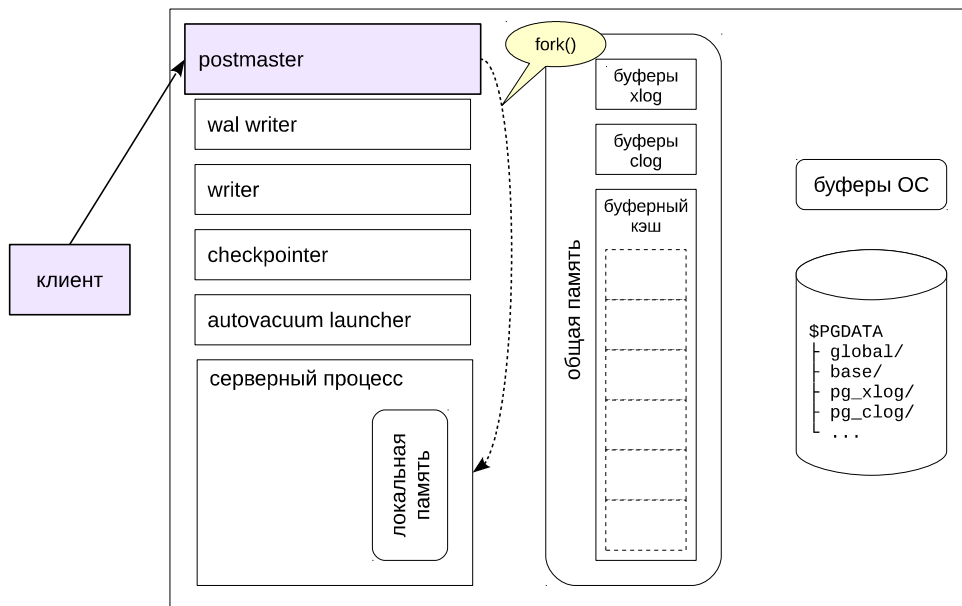


После этого postmaster порождает ряд служебных процессов, предназначение которых будет рассмотрено дальше. Процессы имеют как доступ к общей памяти, так и свою локальную память, недоступную другим процессам.

Затем задача postmaster состоит в том, чтобы:

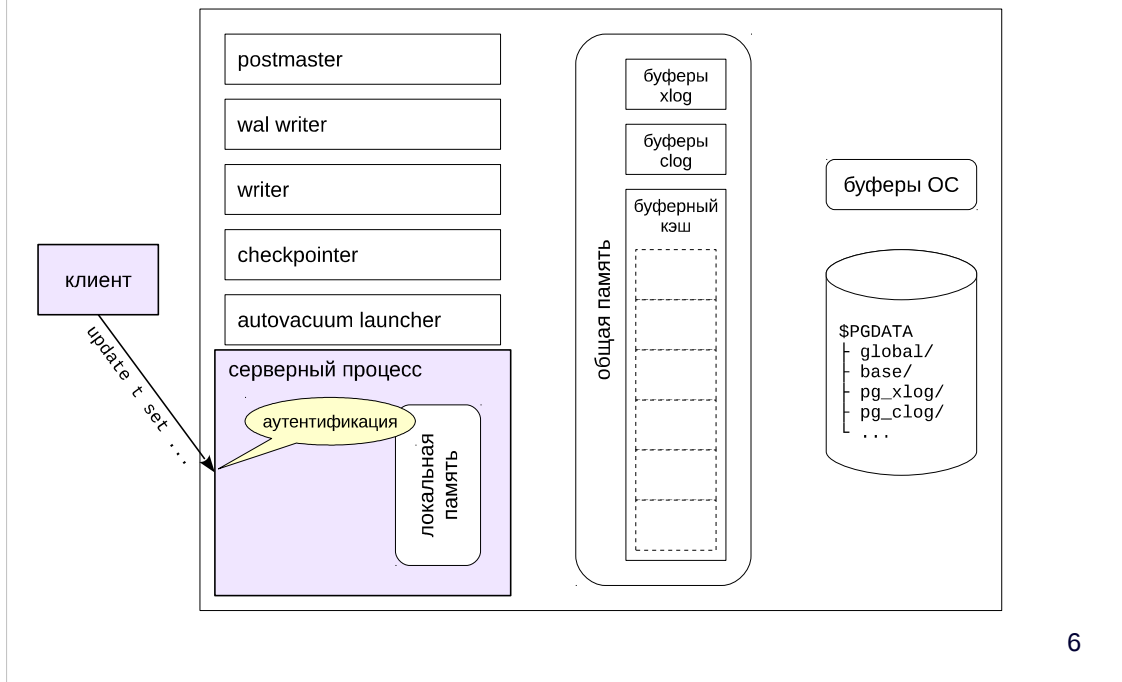
- 1) слушать порт (указанный в настройках) на предмет входящих соединений,
- 2) отслеживать состояние порожденных процессов; если в каком-либо процессе возникнет сбой, postmaster выполнит перезапуск сервера, чтобы избежать проблем из-за возможных повреждений общей памяти.

# Подключение клиента



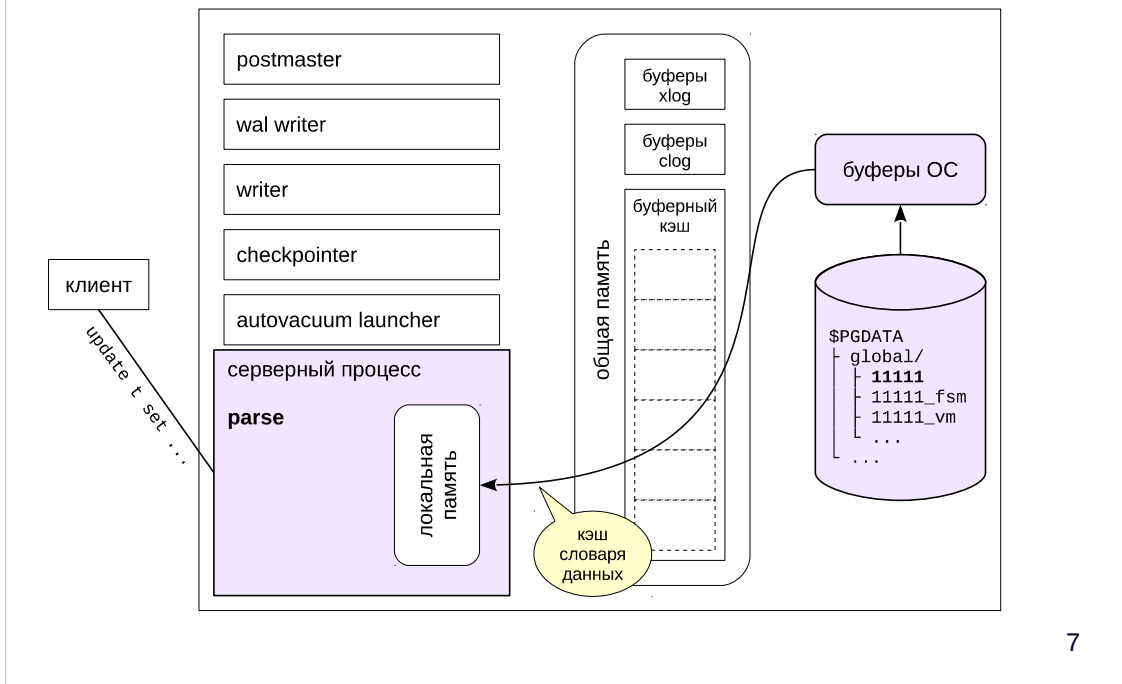
Когда клиент хочет соединиться с сервером БД, он направляет запрос на определенный порт, который обслуживается процессом postmaster. Postmaster порождает отдельный серверный процесс (backend), который и продолжает работу с клиентом. Серверный процесс завершается только при отключении клиента.

Таким образом, подключение клиента — довольно дорогая операция. Кроме того, число серверных процессов в системе равно числу активных подключений. Поэтому в приложениях часто используется пул запросов (для этого можно использовать такие расширения, как pgBouncer, PgPool).



Порожденный серверный процесс выполняет идентификацию, аутентификацию и авторизацию клиента, после чего готов к выполнению поступающих от него запросов.

# Выполнение запроса

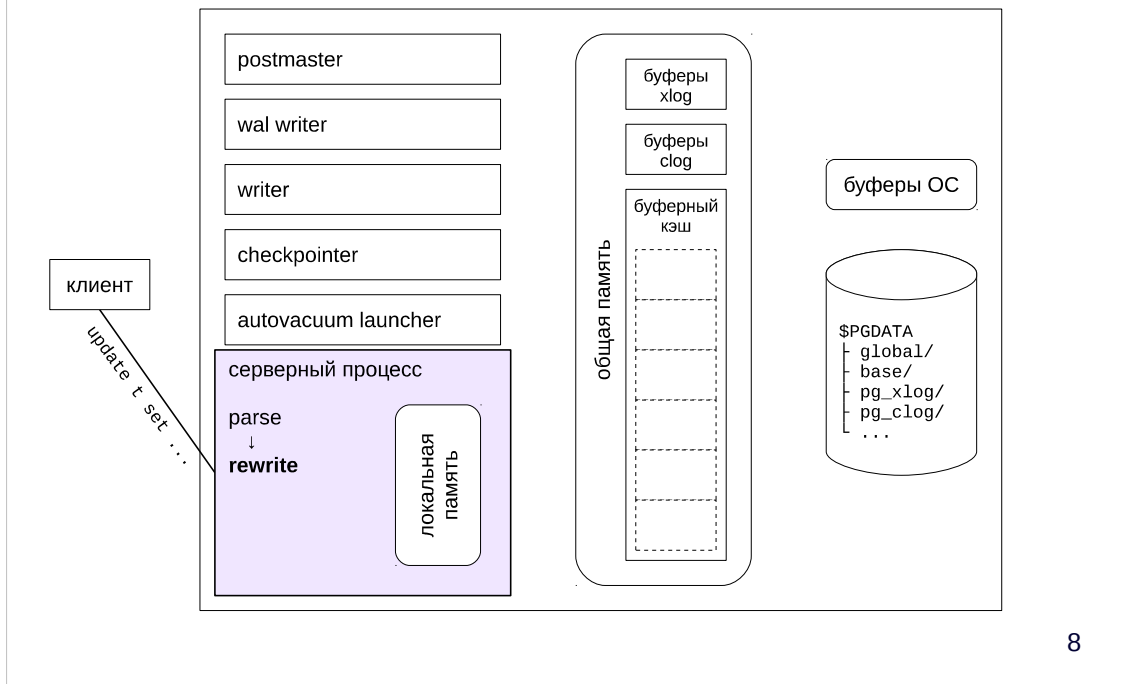


Выполнение запроса начинается с этапа разбора. Проверяется синтаксическая и семантическая корректность переданного клиентом предложения.

Для этого серверному процессу необходимо свериться с данными системного каталога, который содержит сведения о наличии и структуре объектов БД и прав доступа к ним. Чтобы не обращаться каждый раз (при каждом новом запросе) за данными к диску, они кэшируются в локальной памяти процесса.

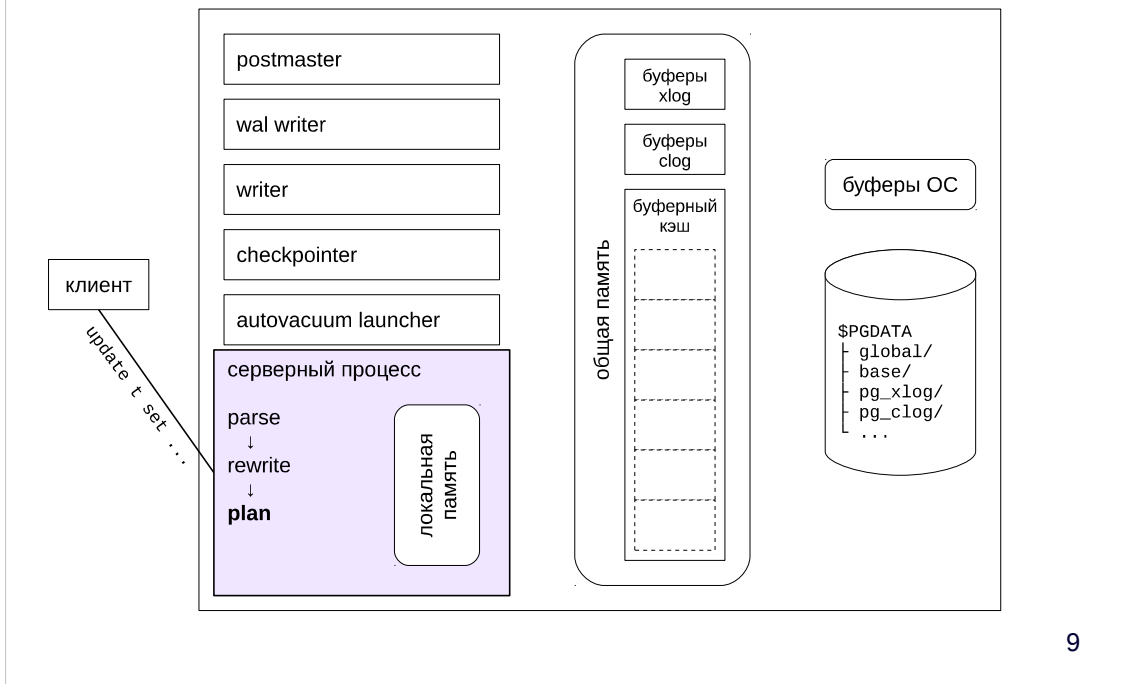
Если какой-либо процесс меняет данные системного каталога, он сообщает остальным процессам (через общую память), что такие-то данные необходимо перечитать. Таким образом поддерживается актуальное состояние данных у всех серверных процессов.

# Выполнение запроса



Далее следует этап переписывания запроса. Здесь PostgreSQL руководствуется настроенными правилами, которые позволяют видоизменить запрос или выполнить дополнительные действия. В частности, с помощью механизма правил реализована подстановка текста представления вместо его имени.

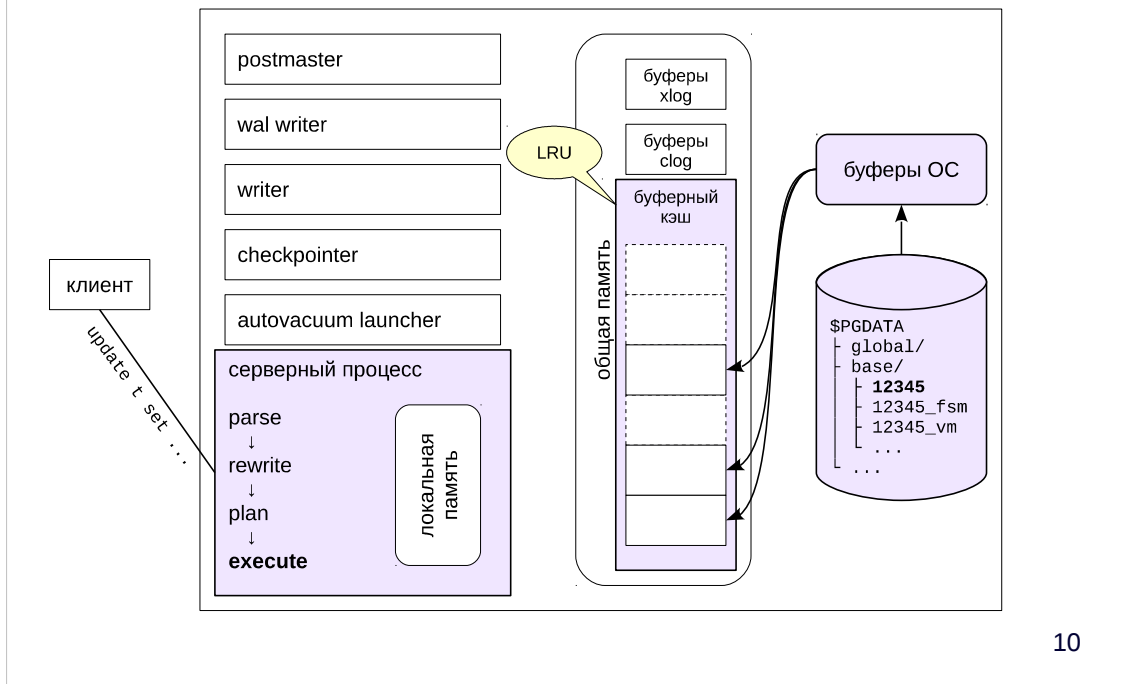
# Выполнение запроса



SQL является декларативным языком и не содержит указаний о том, как именно должен выполняться запрос. Это решение принимается на следующем этапе при планировании выполнения (оптимизации).

Оптимизатор полагается на математическую модель выполнения запроса и статистику (сведения о данных, содержащихся в таблицах). Он перебирает различные планы выполнения и останавливается на плане, имеющем наименьшую стоимость в терминах необходимых ресурсов.

# Выполнение запроса



Далее следует собственно выполнение запроса в соответствии с выбранным планом.

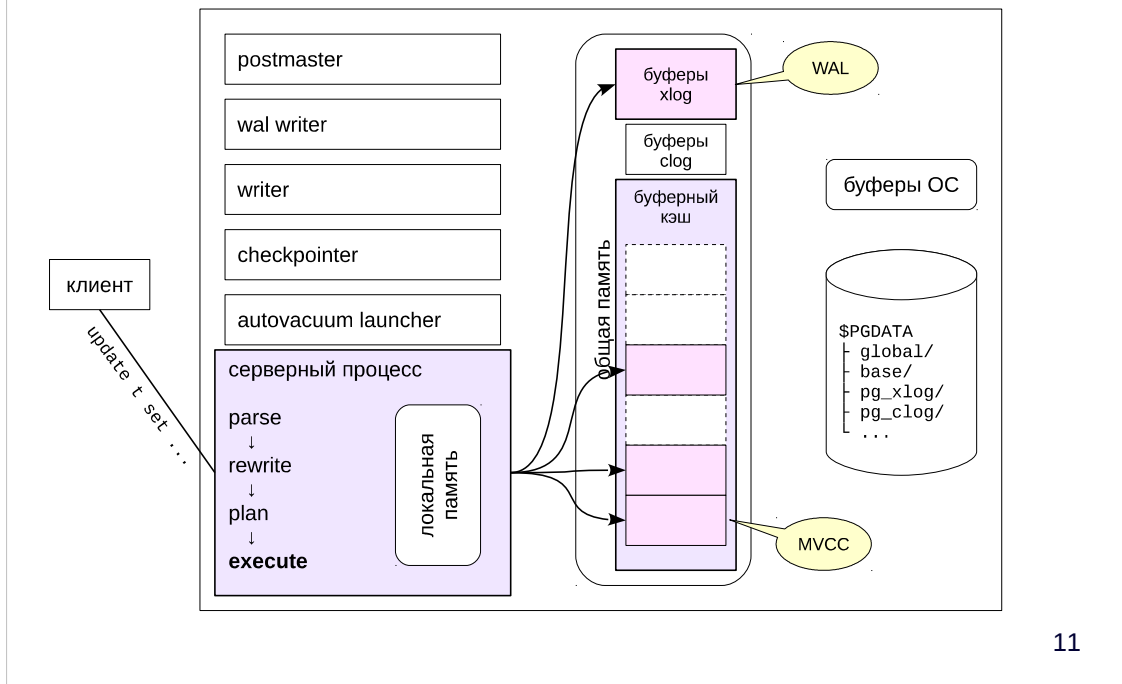
Как правило, при этом необходимо прочитать какие-то данные. Данные хранятся во внешней памяти на дисках. Данные организованы в табличные пространства, каждое из которых соответствует своему каталогу файловой системы. Каждый объект БД занимает несколько файлов. Например, таблицы занимают минимум три файла (собственно данные, а также карта свободного пространства и карта видимости); кроме того, когда файл достигает 1 GB, создается дополнительный новый файл. Таким образом, в системе с большим количеством объектов может быть много файлов, что необходимо учитывать при выборе файловой системы.

При чтении (и записи) данных PostgreSQL пользуется средствами операционной системы. Таким образом, данные сначала попадают в кэш ОС, а только затем — в буферный кэш СУБД.

Для прочитанных с диска данных применяется кэширование в общей оперативной памяти (буферный кэш, shared buffers). Данные читаются и обрабатываются постранично. Размер страницы настраивается при сборке сервера и обычно равен 8 KB.

Поскольку размер кэша мал по сравнению с объемом дисковой памяти, реализовано вытеснение страниц, которые используются реже остальных.

# Выполнение запроса



При изменении данных процесс выполняет два действия:

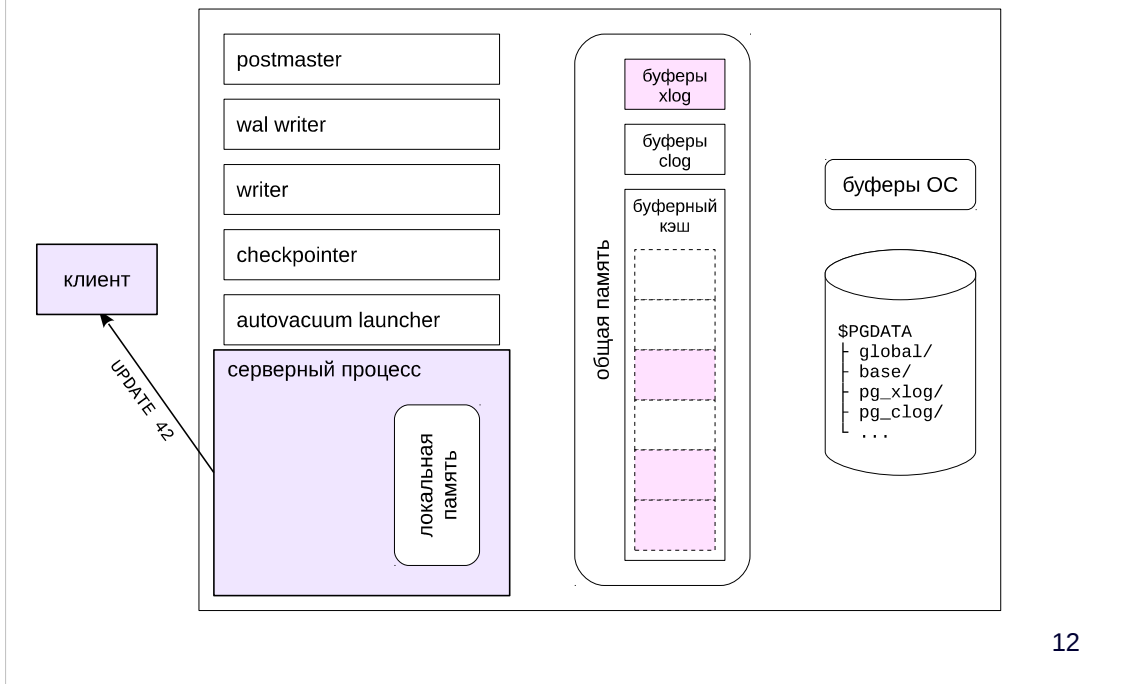
- 1) записывает информацию о выполняемых действиях в журнал упреждающей записи (WAL или XLOG);
- 2) меняет страницы в буферном кэше.

Журнал используется для того, чтобы обеспечить долговечность данных (последнее свойство ACID). PostgreSQL гарантирует, что записи WAL будут записаны на диск до того, как будут записаны сами данные (которые могут находиться в буферном кэше довольно долго). Таким образом, при сбое сервер сможет повторно выполнить все действия и не потерять изменения.

Поддержка остальных свойств ACID (атомарность, согласованность, и особенно изоляция) в значительной степени полагается на механизм MVCC. Его идея состоит в том, что на низком уровне (в страницах данных) сохраняется не только текущая версия строки, но и предшествующие ей. Например, при удалении строки она только помечается как удаленная, а при обновлении сохраняется версия строки со старыми значениями. При этом каждая транзакция видит свой снимок данных, в который входят только те версии строк, которые образуют согласованную картину данных на определенный момент.

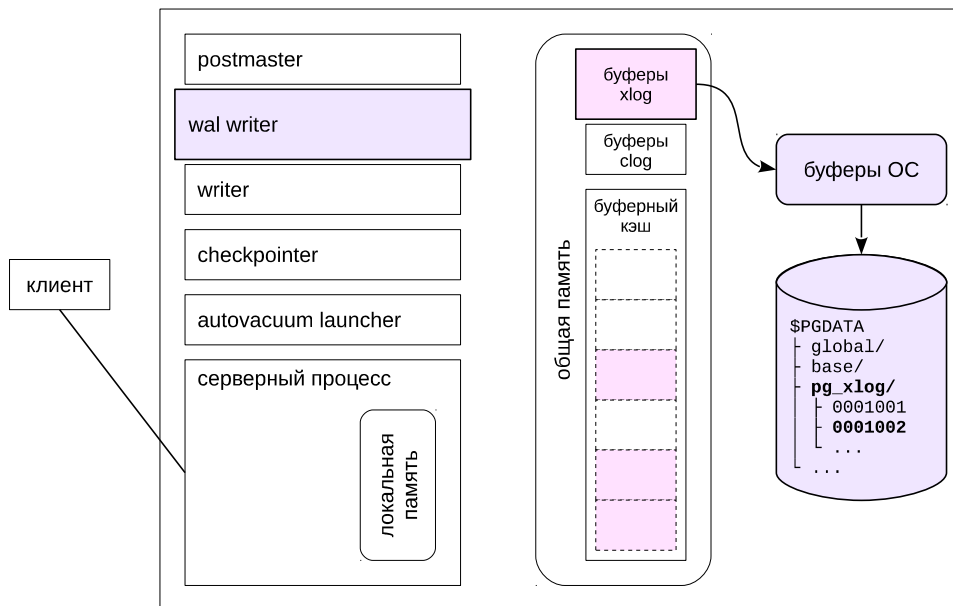
При выполнении запроса серверный процесс может использовать свою локальную память для выполнения промежуточных действий, таких, как соединения или сортировки.

# Выполнение запроса



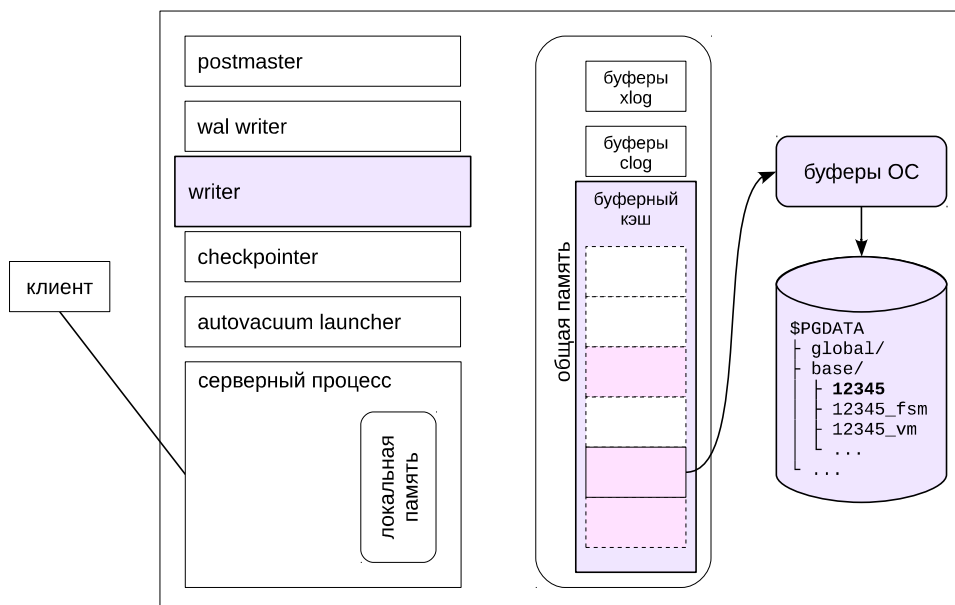
Выполнив запрос, серверный процесс возвращает результат клиенту и ожидает поступления следующего запроса.

# Процесс записи журналов



13

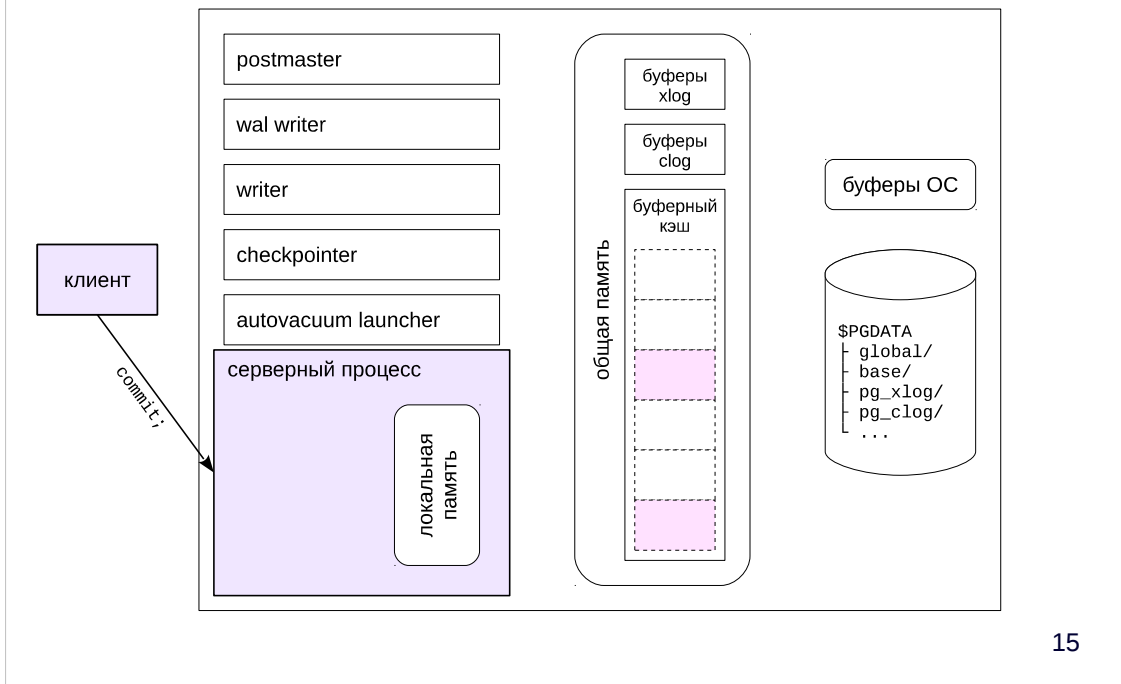
Записи журнала попадают на диск благодаря процессу wal writer. Он переносит накопившиеся записи (разных процессов) в файлы — сегменты WAL. Эти файлы имеют размер 16 MB (настраивается при сборке сервера) и располагаются в каталоге `$PGDATA/pg_xlog`.



Если серверному процессу понадобится буфер, чтобы прочитать страницу с диска, он должен будет найти подходящий (используемый реже, чем остальные). Если найденный буфер окажется грязным, серверному процессу придется самому сбросить его на диск, что плохо, так как замедлит его работу.

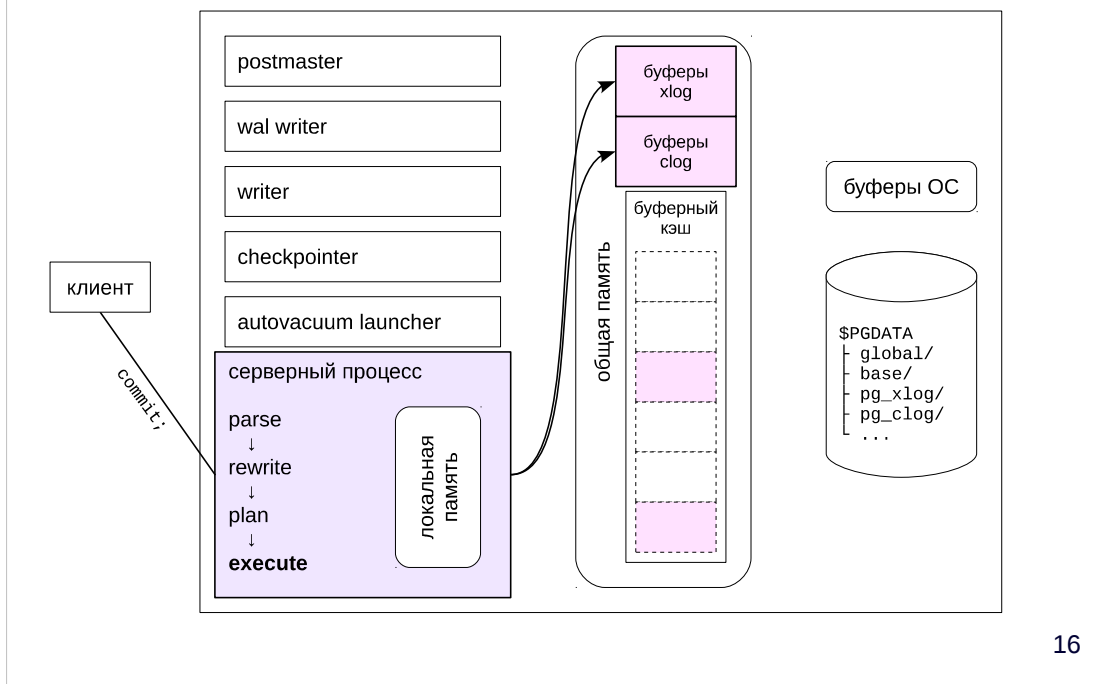
Поэтому существует еще один фоновый процесс — writer. Он выполняется периодически и записывает часть грязных буферов (которые с большой вероятностью будут вытеснены) из кэша на диск, освобождая, таким образом, место для чтения новых данных.

# Фиксация изменений



Рассмотрим теперь фиксацию изменений. Клиент посылает команду commit серверному процессу.

# Фиксация изменений



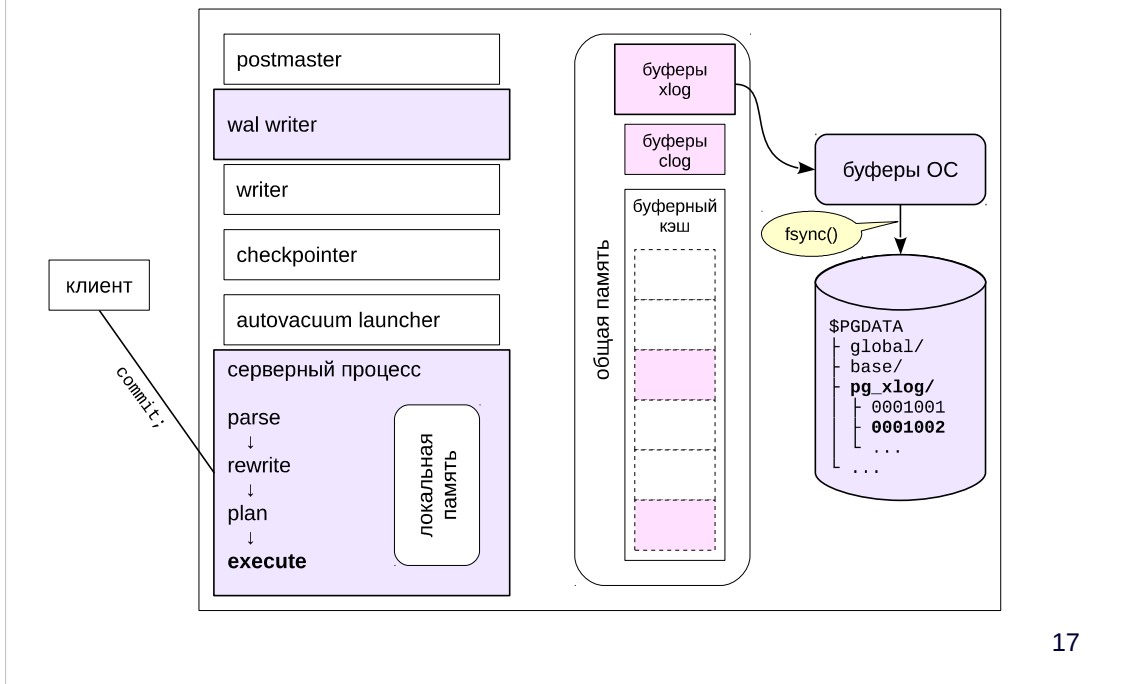
16

Серверный процесс:

- 1) записывает в журнал упреждающей записи информацию о фиксации,
- 2) записывает изменение статуса транзакции в таблицу статусов (commit log, CLOG).

Как всегда, данные должны сначала попасть в журнал, а уж затем — в CLOG.

# Фиксация изменений



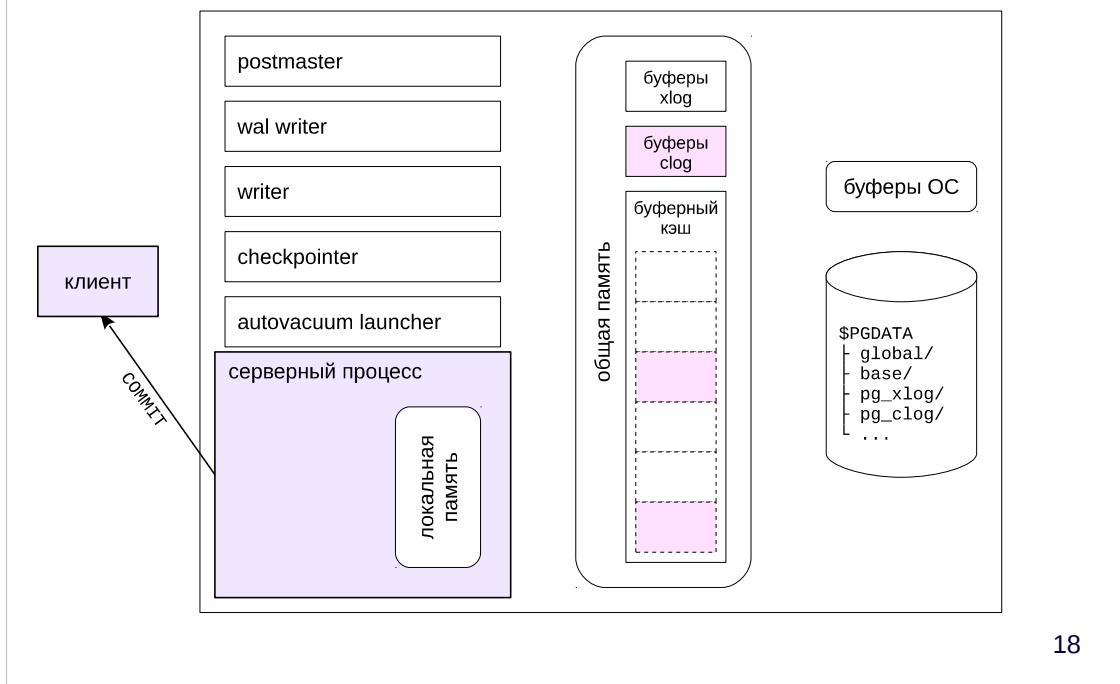
17

В зависимости от настроек, журнал может быть записан сразу же (синхронная фиксация) или позже.

В первом случае гарантируется сохранность зафиксированных данных при сбое. При этом сам серверный процесс записывает журнал на диск; запись сопровождается вызовом `fsync` для того, чтобы данные из буферов операционной системы были физически записаны на жесткий диск. Разумеется, сохранность данных обеспечивается ценой уменьшения производительности.

Во втором случае данные будут записаны спустя некоторое время процессом `wal writer`. В этом случае гарантируется только согласованность данных, но часть информации может быть потеряна при сбое.

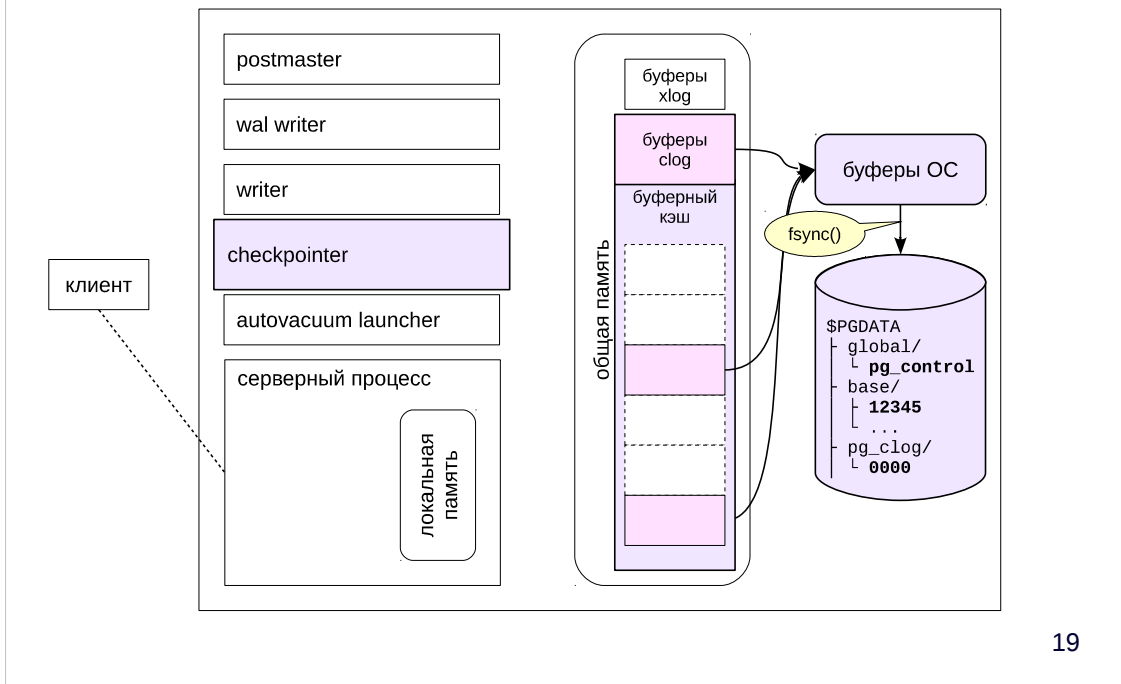
# Фиксация изменений



После того, как записи WAL попали на диск (в случае синхронной фиксации), серверный процесс возвращает результат клиенту.

При этом данные об изменении статуса транзакции остаются в буфере clog; они будут записаны на диск позже.

# Контрольная точка



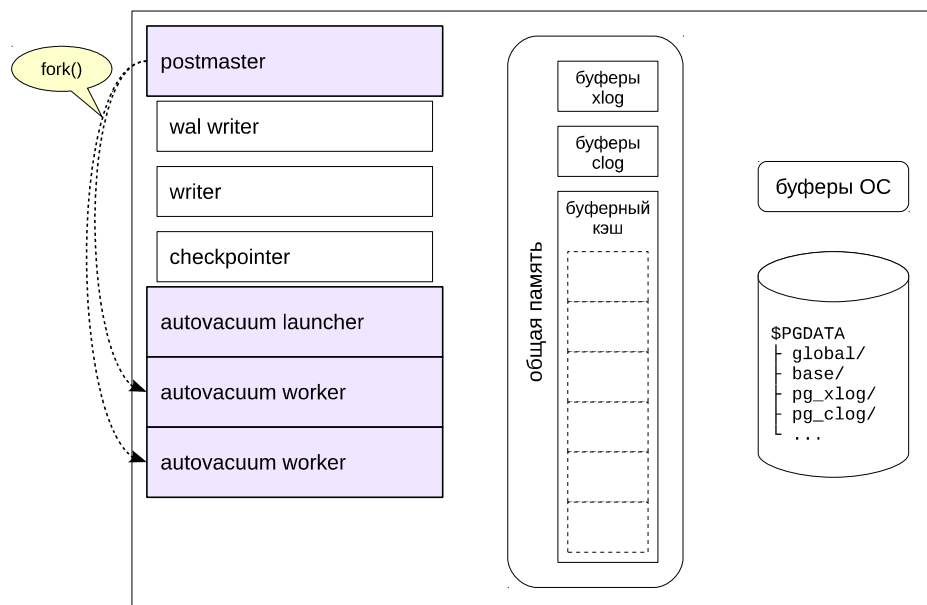
Следующий важный фоновый процесс — процесс выполнения контрольной точки (checkpointер).

Хотя наличие журнала упреждающей записи теоретически гарантирует сохранность данных при сбое, но сколько при этом надо хранить сегментов WAL? Если активно используемые данные не будут вытесняться из буферного кэша, то неопределенно много. Поэтому периодически выполняется контрольная точка, при которой все данные из буферного кэша принудительно сбрасываются на диск. Таким образом, для восстановления после сбоя можно хранить сегменты WAL только с момента последней контрольной точки.

На диск сбрасываются не только блоки данных, но и буферы slog, а также в специальный файл pg\_control записывается информация о пройденной контрольной точке.

Контрольная точка может выполняться быстро, но обычно распределяется во времени, чтобы не нагружать дисковую подсистему.

# Процесс автоочистки

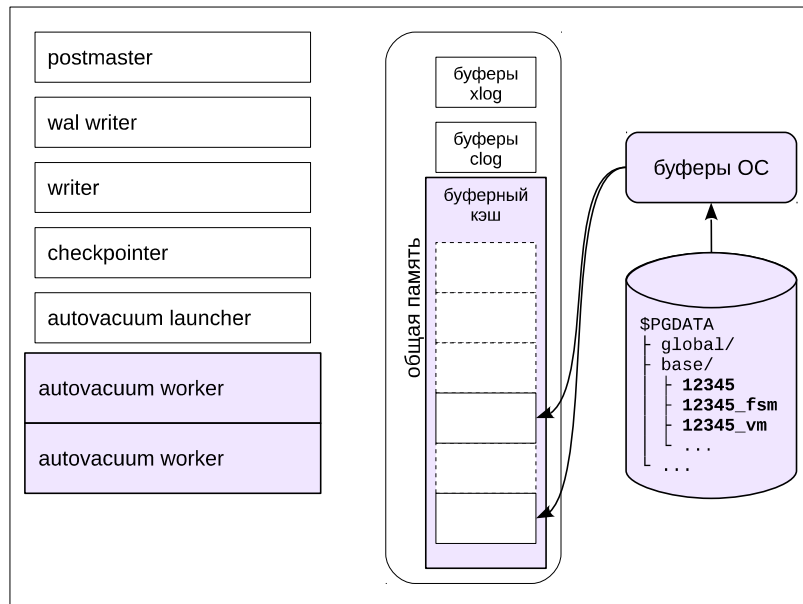


20

При отключении клиента обслуживающий его серверный процесс завершается.

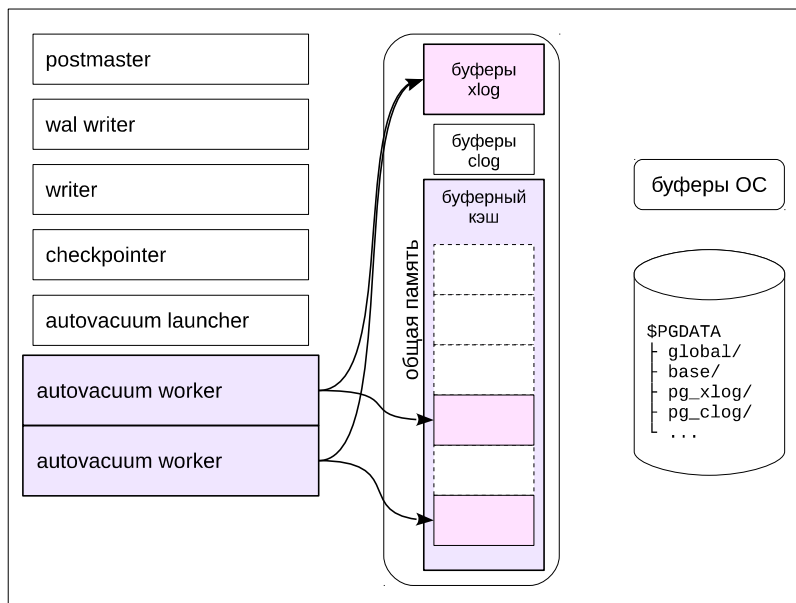
Еще один важный (хотя формально и не обязательный) фоновый процесс — автоочистка. В системе присутствует процесс `autovacuum launcher`, который порождает несколько процессов `autovacuum worker`, работающих параллельно. Но делается это с помощью `postmaster`, так как только он занимается порождением новых процессов.

# Процесс автоочистки



Собственно очистка состоит в удалении из страниц тех версий строк, которые уже не используются ни в одном снимке данных. Для этого процессы читают страницы с диска в буферный кэш (если, конечно, они уже не находятся в кэше) и вносят в них исправления.

# Процесс автоочистки



22

Как обычно, сначала записывается информация в журнал упреждающей записи, а затем — в сами страницы.

Исправления остаются в оперативной памяти до тех пор, пока не будут записаны на диск: буферы журнала — самим процессом автоочистки или процессом wal writer; буферы данных — серверным процессом либо процессами writer и checkpointer, как рассматривалось ранее.

## Архитектурно PostgreSQL:

взаимодействующие процессы

один клиент — один обслуживающий процесс

локальная и общая память, кэширование

внешняя память под управлением ОС

Большое количество фоновых процессов требуют настройки

1. Под пользователем postgres установлен сервер СУБД, создан кластер и настроена переменная окружения PGDATA. Проверьте, запущен ли сервер; если нет — запустите.
2. В каком каталоге находятся файлы кластера?
3. В каком каталоге находятся исполняемые файлы?
4. Найдите номер процесса postmaster.
5. Посмотрите процессы, порожденные postmaster, средствами операционной системы. Понятно ли назначение всех процессов?