PostgresPro

Performance tuning
part II. instance tuning

Владимир Пудовченко

Что такое тюнинг?

"диагностика неисправностей и поиск и устранение неэффективностей"

Потоки получения информации:

1. логи субд
2. семплирование ожиданий
3. общая системная информация об утилизации основных ресурсов
4. представления с оперативными и накопительными статистиками pg_stat_* (внутренние системные статистики postgres)
5. Профили запросов (top-SQL)

USE – Utilisation Saturation Errors
RED – Rate Errors Duration
LETS – Latency Errors Traffic Saturation (Goodle 4 Golden Signals)
STELA – (LETS+A) + Availability

tuning

PostgresPro

performance tuning

сопровождение

- настройка резервирования
- обновление
- мониторинг

поиск неисправностей

- субд почему-то упала
- упала и не включается
- регулярно падает

Подход к настройке

Задача тюнинга СУБД заключается в сокращении времени на обработку запроса сервером И оптимизации объявленных показателей (TPS, время отклика).

SQL tuning

Проблемное место известно. Долго загружается какая-то страница приложения или долго строится отчёт.

Instance tuning

- Широкий(количественный) взгляд на работу сервера (TPS , latency и тд.)
- Семплирование ожиданий
- Нахождение узкого места в работе сервера

Hardware\OS tuning

- Ulimits, ФС, SWAP.
- параметры ядра ОС sysctl
- BIOS/UEFI

tuning OS

Доклады Михаила Жилина:

- **Лучшие практики по OS + PostgreSQL**
- **Ускоряем железо и OS под PostgreSQL**
- **Разгоняем железо и операционную систему на максимальную производительность. Бенчмаркаем на PostgreSQL**

Простой вариант – tuned

• Позволяет системным администраторам легко и динамично менять настройки ядра
• Не нужно вносить изменения в /etc/sysctl

#sudo apt install tune

```
student:~$ tuned-adm list | grep  postgresql
- postgresql                    - Optimize for PostgreSQL server
```

```
grep -v "^\s*[#\;]\|^\s*$" /usr/lib/tuned/postgresql/tuned.conf
[main]
summary=Optimize for PostgreSQL server
include=throughput-performance
[cpu]
force_latency=1
[vm]
transparent_hugepages=never
[sysctl]
vm.dirty_background_ratio = 0
vm.dirty_ratio = 0
vm.dirty_background_bytes = 67108864
vm.dirty_bytes = 536870912
vm.swappiness=3
kernel.sched_autogroup_enabled = 0
[scheduler]
sched_min_granularity_ns = 10000000
sched_migration_cost_ns = 50000000
```

Instance tuning

Instance tuning

цель: выжить из дорогостоящего железа максимум, обеспечить стабильную работу (без скачков).

1. первичная настройка экземпляра из коробки.
2. донастройка на основе мониторинга под специфическую нагрузку конкретного приложения.

В PG порядка 150+ GUC-параметров (postgresql.conf)
из которых 20-30 параметров для которых известны «правильные значения»
Параметры хранения (табличные параметры)

Примеры

1. switchover узлов кластера под управлением кластерного ПО patroni из-за высокой загрузки процессора.
2. отставание синхронной реплики
3. вымывание кэша после ночных джобов
4. autovacuum упирается в потолок производительности -> разрастание таблиц и индексов

«медленно работает база»

подходы

Повышение производительности – устранение проблем с производительностью

1. Для диагностики проблем с производительностью требуется знание внутреннего устройства СУБД:
- архитектура
- фоновые процессы
- области памяти и тд.

2. Книги рецептов не существует:
- каждый случай уникален
- 100% однозначных закономерностей не существует
- пока ещё нет автономных БД

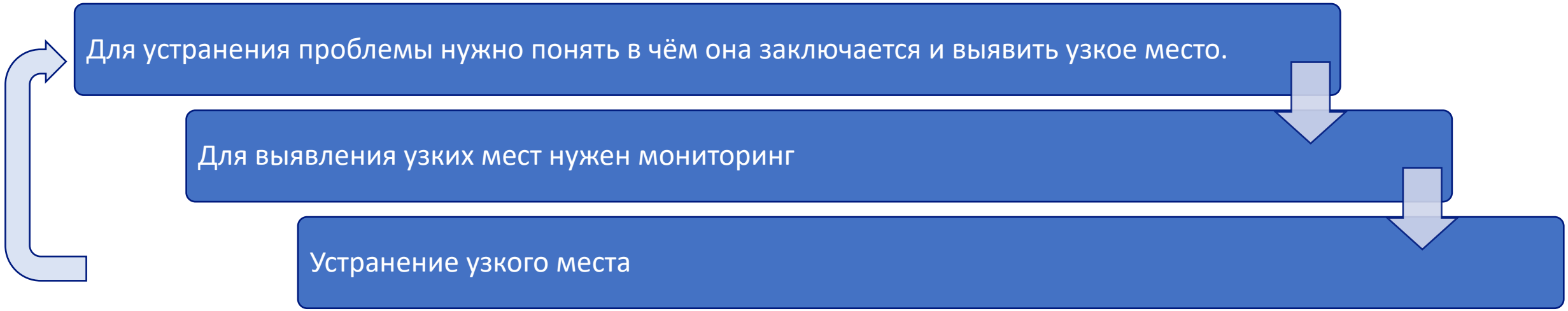
куда теряется производительность

Instance tuning

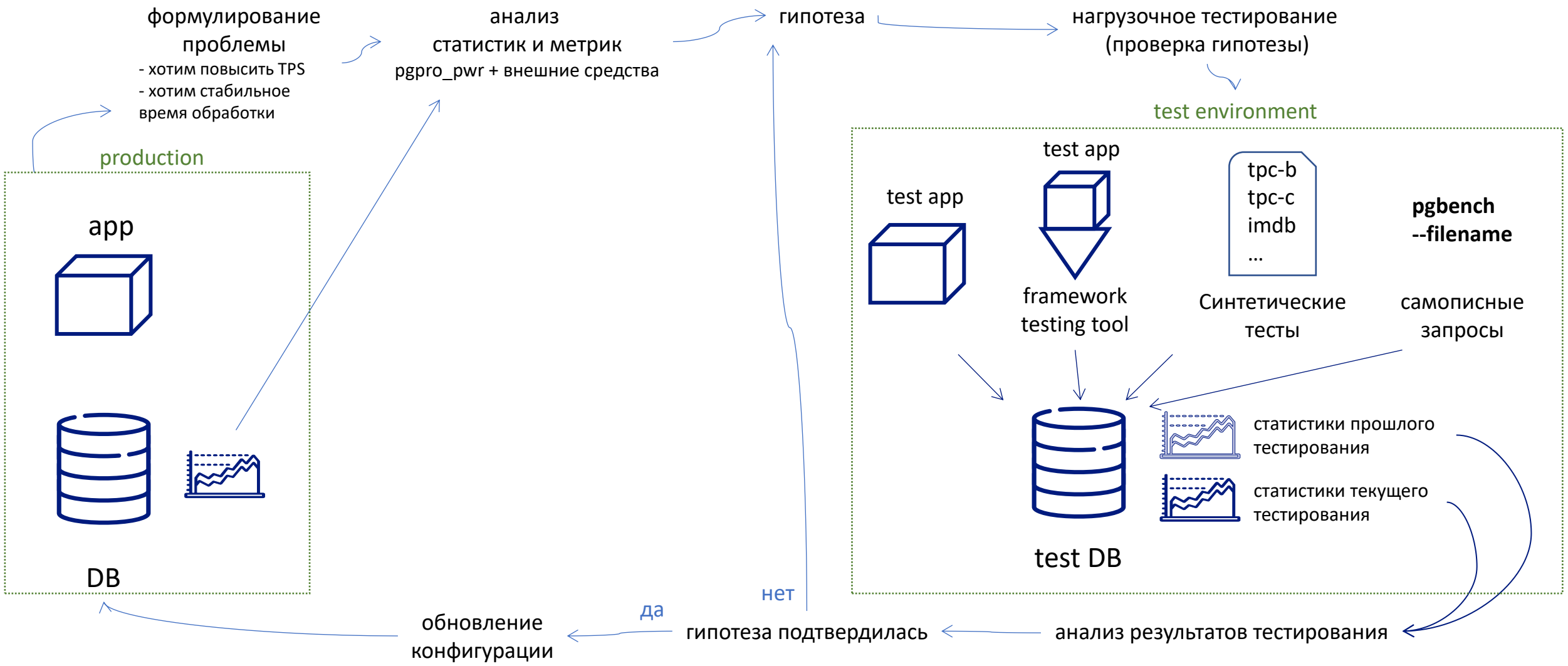
PostgresPro

Количественный взгляд на работу СУБД (TPS, latency, etc.)

- растёт БД (как в ширину, так и в глубину)
  - появляется необходимость дополнительно что-то сохранять в БД
  - растут исторические данные
- увеличивается число пользователей
- появляется аналитика и тяжёлые OLAP-запросы



Тестирование (CI/CD pipeline)



Мониторинг внешними средствами

htop — информативная версия **top**.
atop — продвинутый монитор процессов
iotop — **мониторинг** ввода/вывода
И тд.

Если это выделенный под СУБД сервер, то и нагрузку на систему создаёт только СУБД.
Внешними средствами мы везде будем видеть только Postgres.

Внутренний мониторинг

pg_stat_activity. Мониторинг активностей и ожиданий

Когда процесс ожидает чего-либо, этот факт отражается в представлении **pg_stat_activity**
wait_event_type — тип ожидания
wait_event — имя конкретного ожидания

Информация только на текущий момент
- единственный способ получить картину во времени — семплирование
- достоверная картина только при большом числе измерений

Семплирование

pg_wait_sampling. Сбор статистик с некоторой периодичностью

pg_wait_sampling — это расширение Postgres Pro для периодического сбора статистики по событиям ожидания.

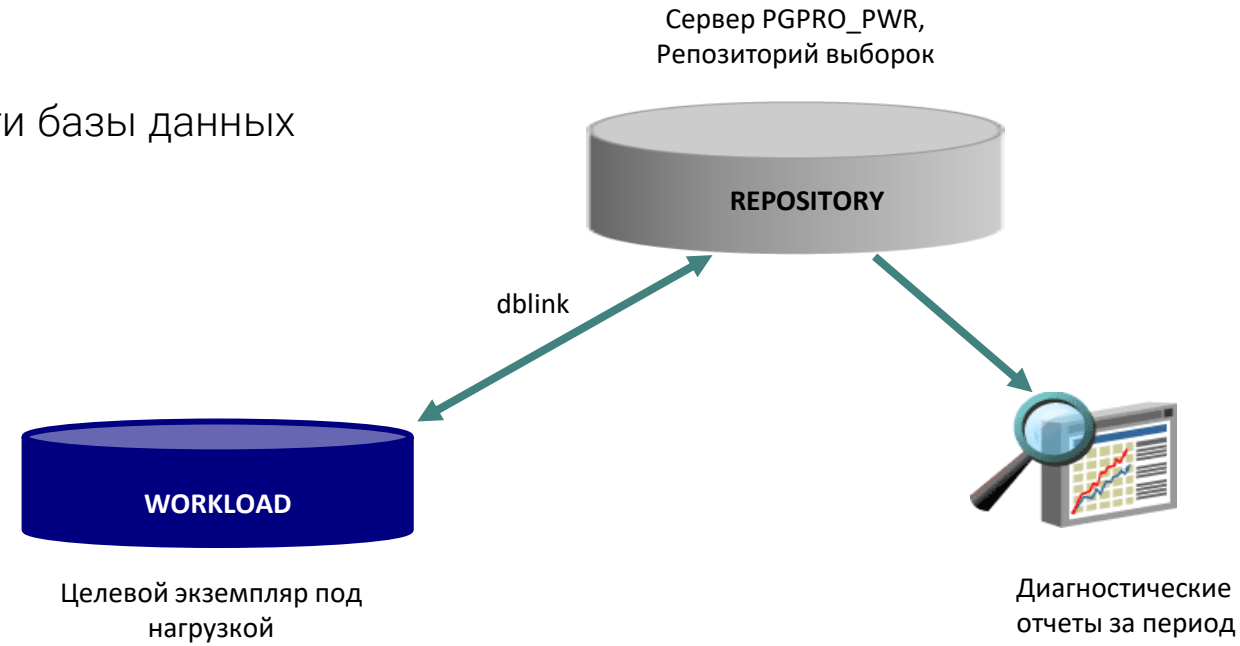
pid	backend_type	app	event_type	event	count
23559	checkpointer		Activity	CheckpointerMain	684
23559	checkpointer		IO	DataFileSync	5
23559	checkpointer		IO	SLRUFlushSync	4
23559	checkpointer		IO	ControlFileSyncUpdate	3
23559	checkpointer		IO	WALSync	2
23560	background writer		Activity	BgWriterMain	708
23561	walwriter		Activity	WalWriterMain	705
23561	walwriter		IO	WALSync	3
23562	autovacuum launcher		Activity	AutoVacuumMain	708
23565	logical replication launcher		Activity	LogicalLauncherMain	708
23600			IPC	CheckpointDone	24
23600			Client	ClientRead	10
23600			IO	CopyFileRead	1
23651	client backend	psql	Client	ClientRead	302
23651	client backend	psql	IO	WALSync	1

Отчёты pgpro_pwr

Абсолютные значения ожиданий сами по себе мало информативны. Нужно понимать за какой период они собирались. (семплирование для семплирования).

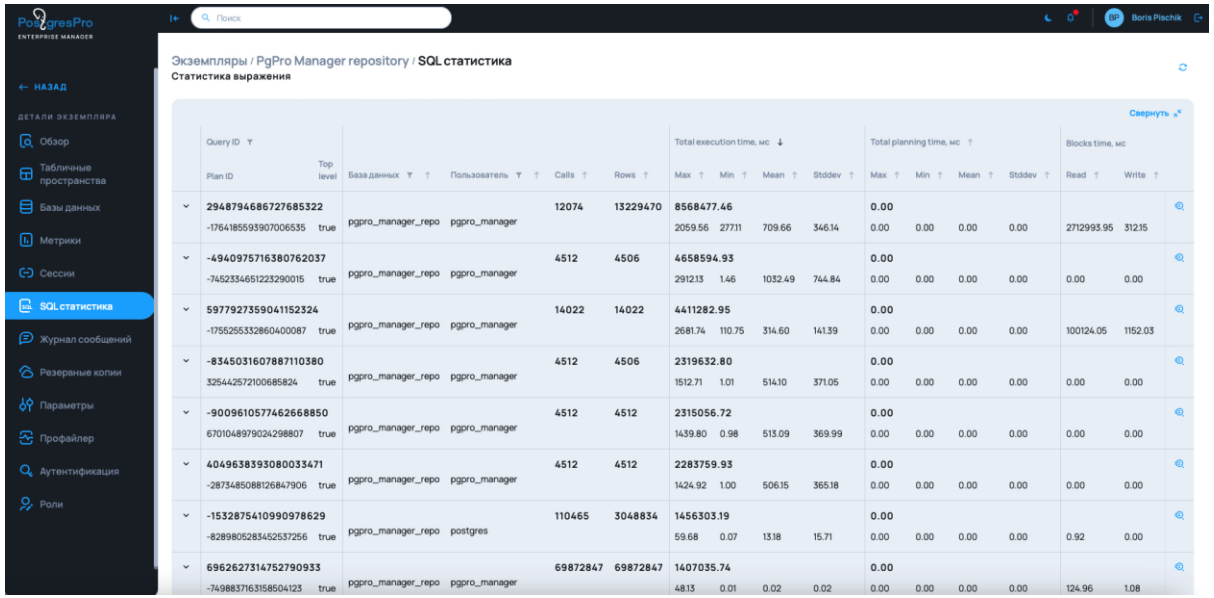
В каждый снимок pgpro_pwr попадает кумулятивная статистика ожиданий, после чего вызывается pg_wait_sampling_reset(). Построив отчёт по снимкам pgpro_pwr можно получить детальную картину о том какие ожидания преимущественно были между интервалов снимков.

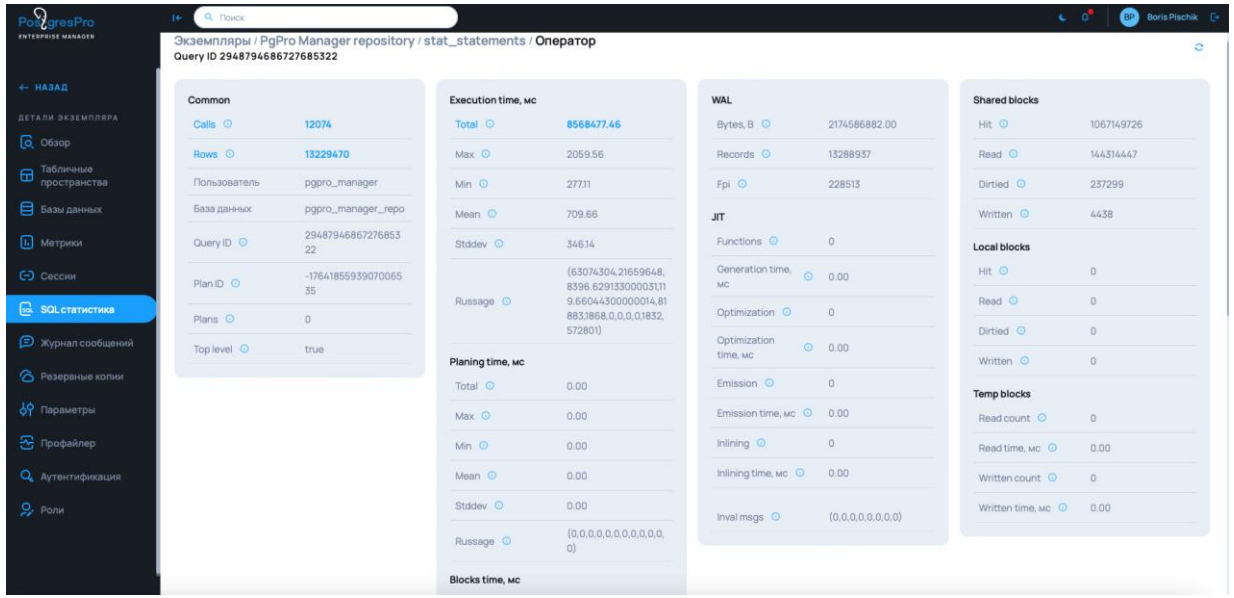
- «Диагностическая карта» производительности базы данных
- Периодические снимки статистики - samples
- Репозиторий – хранилище истории снимков
- Детальный отчет за выбранный период

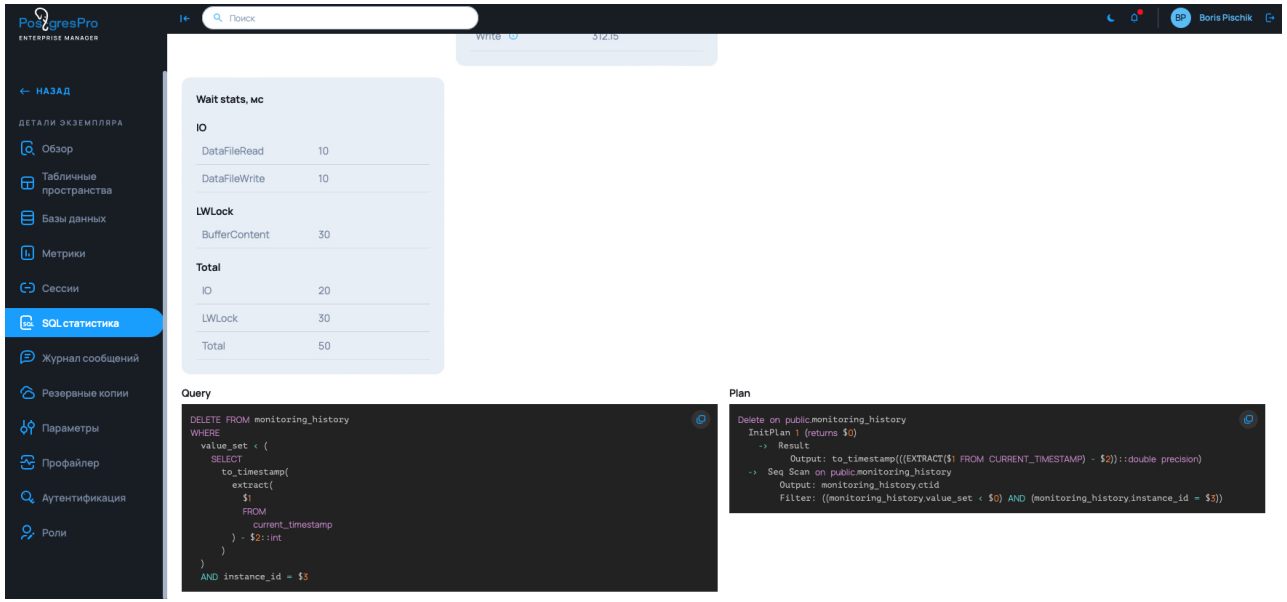


* дополнительно Борис Пищик: **https://postgrespro.ru/video/how-to**

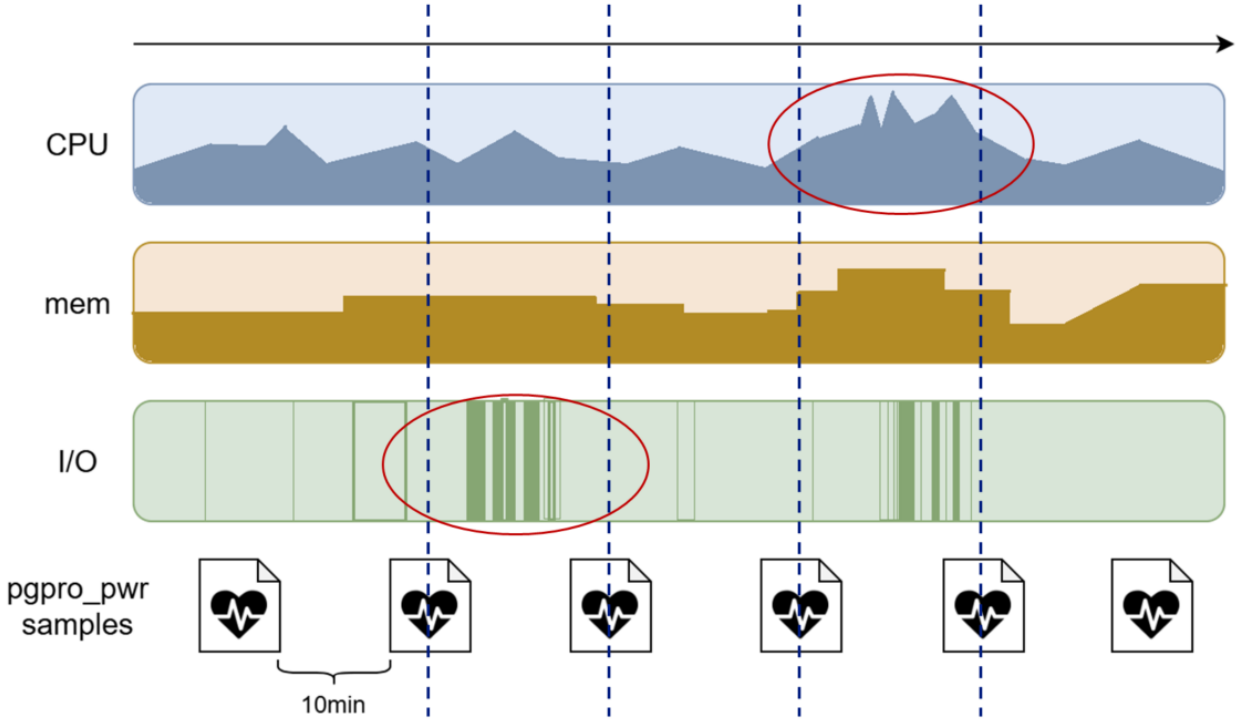
PGPRO_STATS – графический интерфейс PPEM



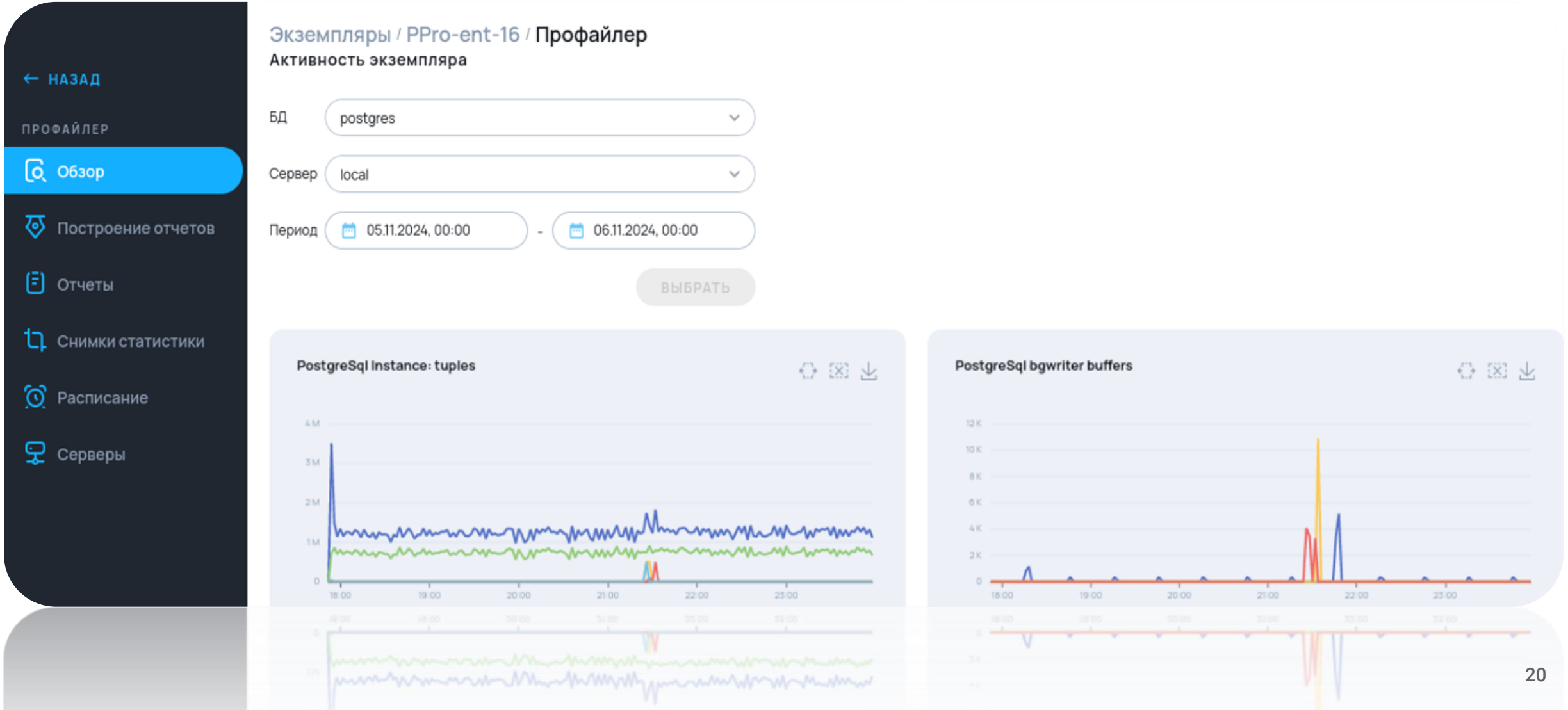


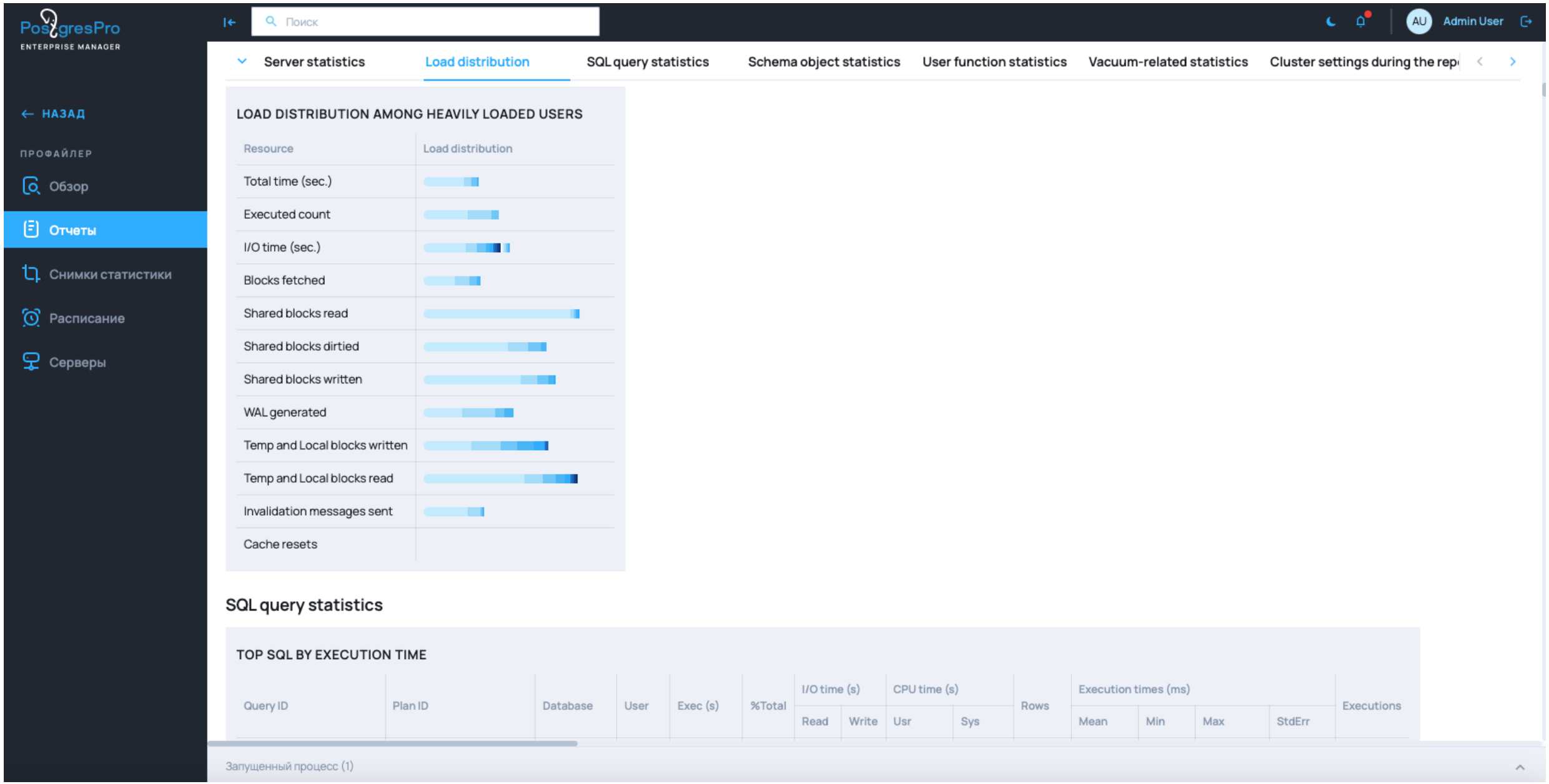


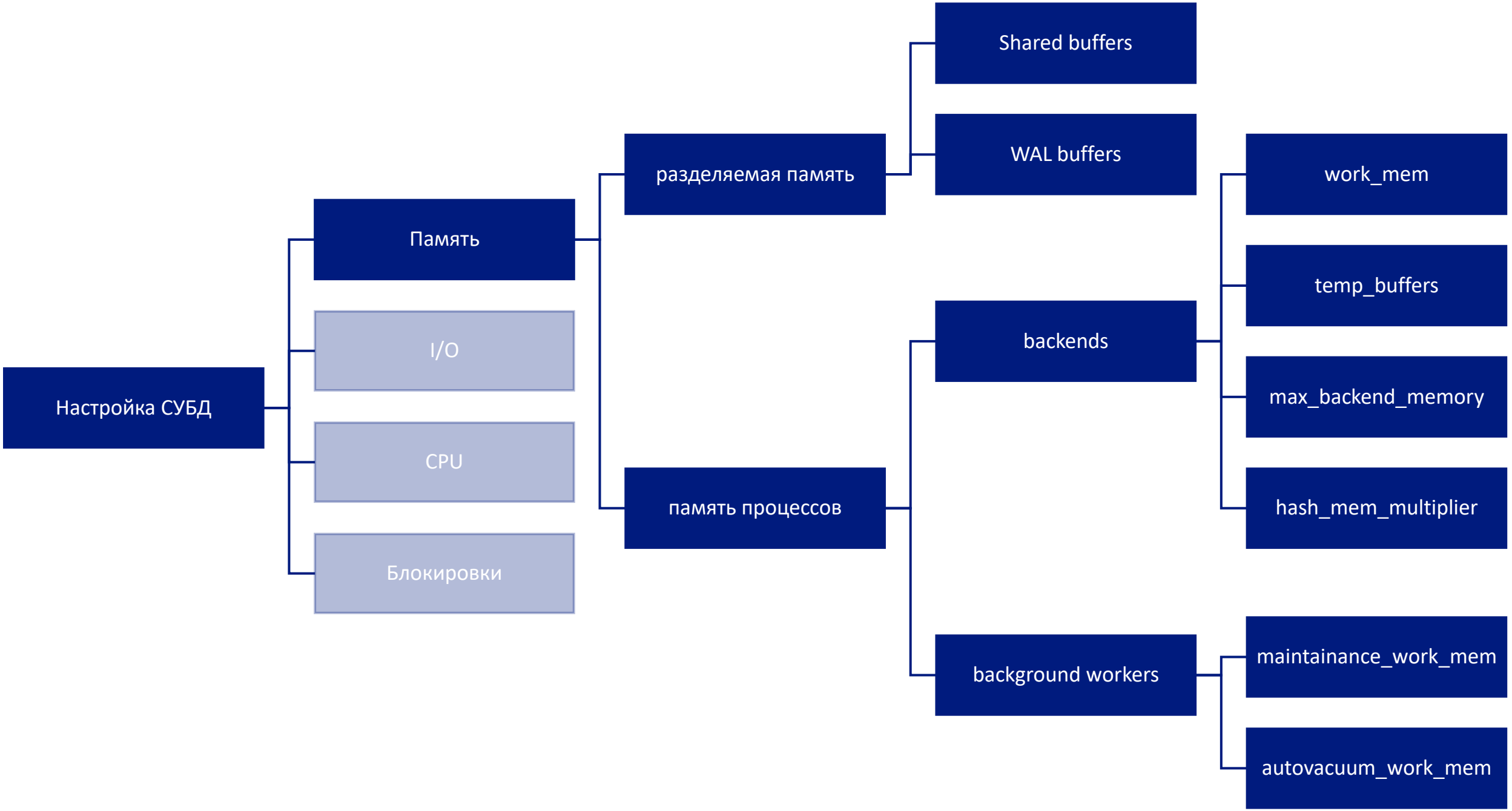
Дифференциальные отчёты pgpro_pwr



Отчёты pgpro_pwr в PPEM



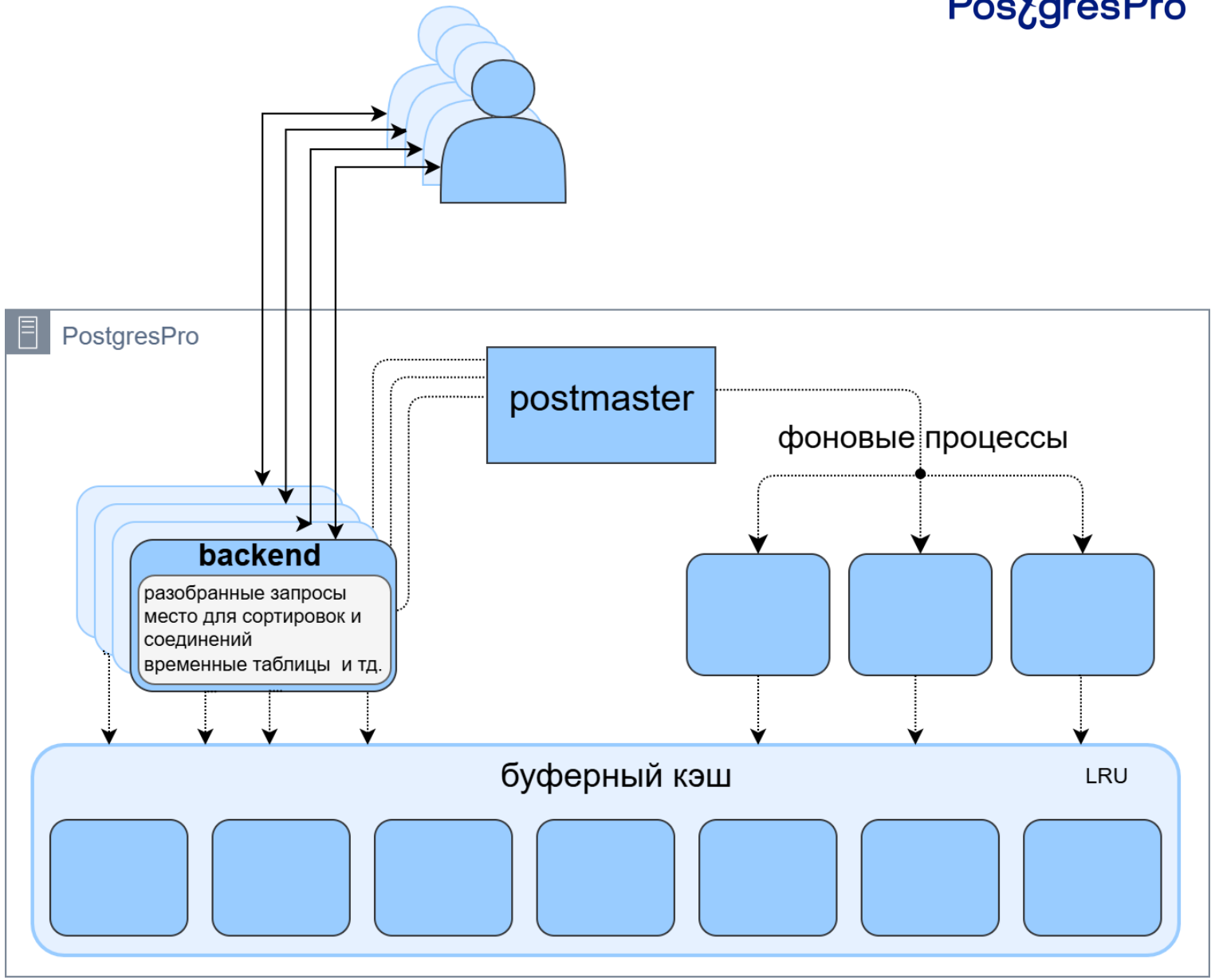

PostgresPro
ENTERPRISE MANAGER
Поиск
AU
Admin User
← НАЗАД
ПРОФАЙЛЕР
Обзор
Отчеты
Снимки статистики
Расписание
Серверы
Server statistics
Load distribution
SQL query statistics
Schema object statistics
User function statistics
Vacuum-related statistics
Cluster settings during the rep
LOAD DISTRIBUTION AMONG HEAVILY LOADED USERS
Resource
Load distribution
Total time (sec.)
Executed count
I/O time (sec.)
Blocks fetched
Shared blocks read
Shared blocks dirtied
Shared blocks written
WAL generated
Temp and Local blocks written
Temp and Local blocks read
Invalidation messages sent
Cache resets
SQL query statistics
TOP SQL BY EXECUTION TIME
Query ID
Plan ID
Database
User
Exec (s)
%Total
I/O time (s)
Read
Write
CPU time (s)
Usr
Sys
Rows
Execution times (ms)
Mean
Min
Max
StdErr
Executions
Запущенный процесс (1)


Настройка СУБД
Память
I/O
CPU
Блокировки
разделяемая память
память процессов
Shared buffers
WAL buffers
backends
background workers
work_mem
temp_buffers
max_backend_memory
hash_mem_multiplier
maintainance_work_mem
autovacuum_work_mem

Буферный кэш

В заголовке буфера:

- bufferid
- relfilenode
- reltablespace
- reldatabase
- relforknumber
- isdirty
- **usagecount**
- pinning_backends



Настройка shared_buffers

shared_buffers используется для определения того, сколько памяти будет выделено серверу для кэширования данных.

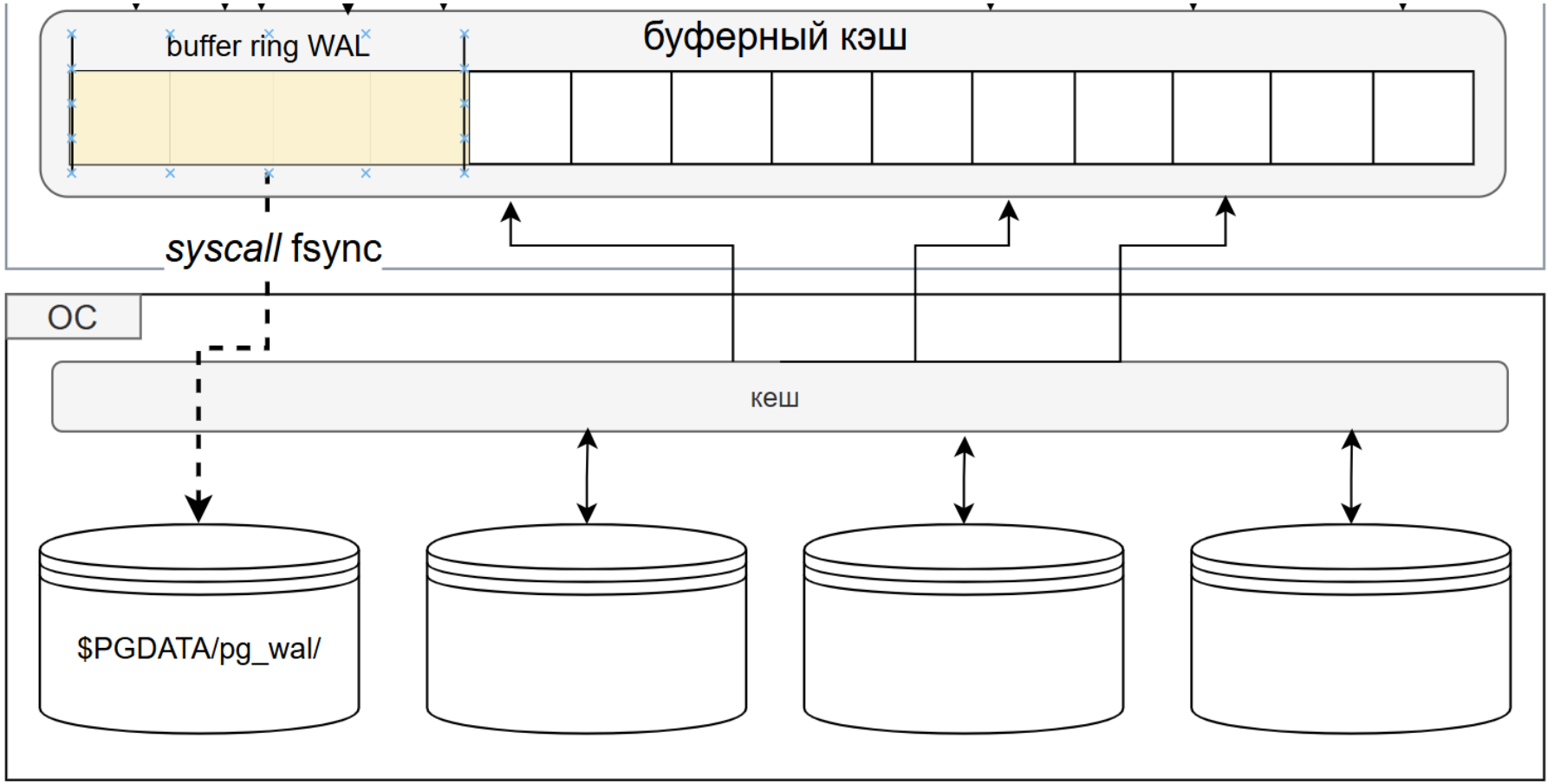
Разумное начальное значение - ¼ памяти на сервере. (99% случаев верно)

Для более прицельной настройки можно воспользоваться расширением pg_buffercache (в составе субд). Либо если серверные процессы часто ожидают блокировки «buffer pin lock»

Инспекция кэша СУБД:

```
postgres=# select usagecount, count(*) from pg_buffercache group by usagecount order by usagecount ;
 usagecount | count
------------+--------
          1 |   1029
          2 |    517
          3 |     33
          4 |    673
          5 |   6544
      <NULL> | 245156
(6 строк)
```

wal_buffers - объём разделяемой памяти, который будет использоваться для буферизации данных WAL, ещё не записанных на диск. Значение по умолчанию, равное -1, задаёт размер, равный 1/32 (около 3%) от shared_buffers



Мониторинг wal_buffers

Стоит увеличивать если:

1. Серверные процессы часто ожидают получения блокировкок «WALinsertLock» либо «WALBufMappingLock».

2. Реплика периодически отстаёт, тогда процессу wal sender приходится читать недостающие записи с диска.

Это можно определить:

- Процесс wal sender имеет не нулевой i/o на мастере. Большая разница sent_lsn и write_lsn

Использование временных таблиц

PostgresPro

temp_buffers – ограничивает максимальный объем памяти для хранения данных временных таблиц (в рамках каждого отдельного сеанса).
CREATE { TEMPORARY | TEMP } TABLE

Как определить, что используются временные таблицы?

Отчет pg_profile или pgpro_pwr!

Statement statistics by database

Database	Calls	Time (s)				Fetched (blk)		Dirtied (blk)		Temp (blk)		Local (blk)	
		Exec	Read	Write	Trg	Shared	Local	Shared	Local	Read	Write	Read	Write
	21142063357	5434617.12	13951.10	55.84		3675989789344	828719501	1039147932	149806370	294562125	305989125	710679390	326520283
pg_profile	24942263	7769.07	0.74			945853344	94006	4794711	6458	5329774	7327746	7074	9846
postgres	1983052	3119.95	0.01			60947665		3					
template1	24	0.00				67							
Total	21168988696	5445506.14	13951.85	55.84		3676996590420	828813507	1043942646	149812828	299891899	313316871	710686464	326530129

maintenance_work_mem задаёт максимальный объём памяти для операций обслуживания БД, в частности VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY. Увеличение этого значения может привести к ускорению операций очистки и восстановления БД из копии.

autovacuum_work_mem задаёт максимальный объём памяти, который будет использовать каждый рабочий процесс автоочистки. (при –1 используется maintenance_work_mem).

! Выделяется в полном объёме и может выделяться autovacuum_max_workers раз, поэтому не стоит устанавливать значение по умолчанию слишком большим.

! Недостаточный размер приводит к многократному сканированию индексов таблицы при очистке.

Мониторинг autovacuum_work_mem

<17 PostgresPro

Число индексных сканирований должно равняться числу индексов по таблице.

- pg_stat_progress_vacuum.index_vacuum_count
- VACUUM (verbose) <tablename>

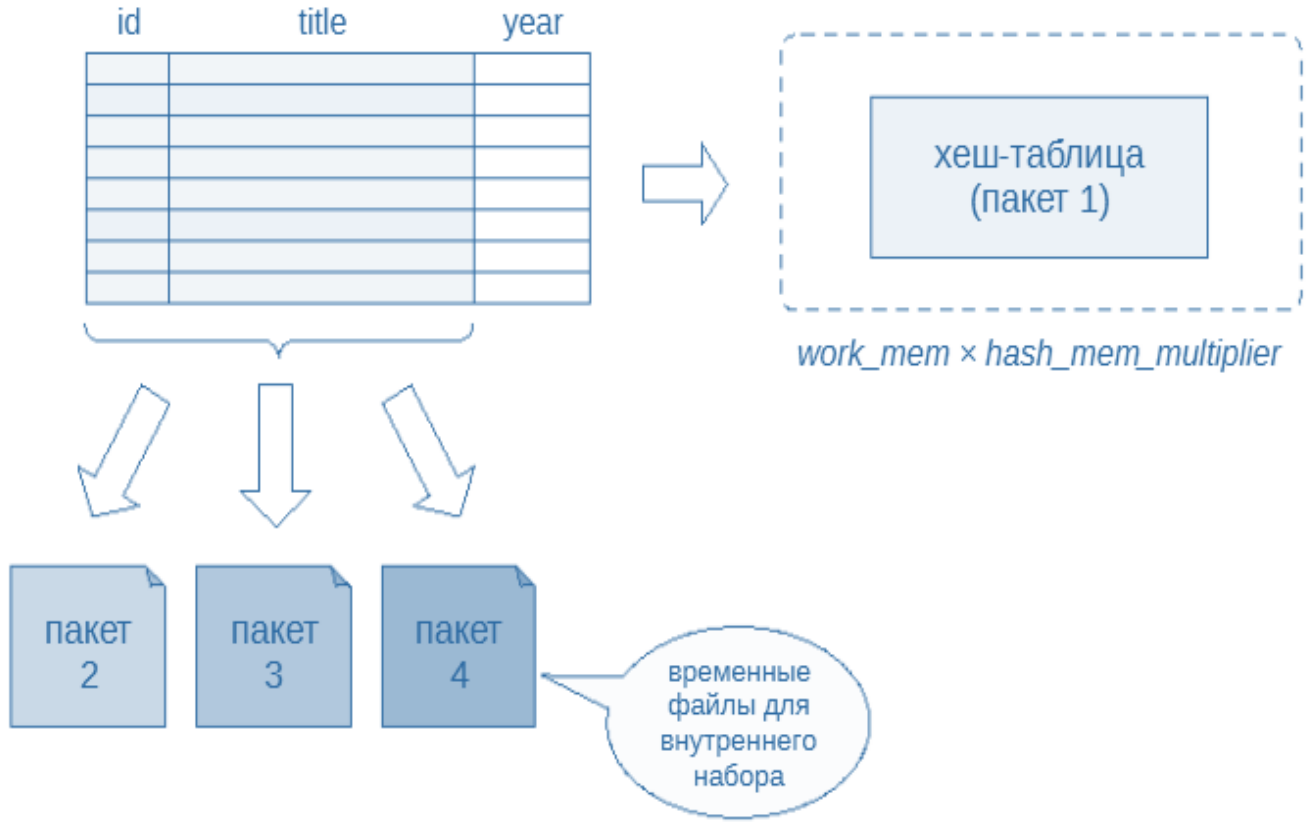
```
postgres=# vacuum (verbose) t ;
INFO:  vacuuming "postgres.public.t"
INFO:  finished vacuuming "postgres.public.t": index scans: 1
pages: 0 removed, 6431 remain, 2054 scanned (31.94% of total)
tuples: 189999 removed, 870865 remain, 0 are dead but not yet r
```

- log_autovacuum_min_duration TO '1ms'

»	Дата, время	Экземпляр	Пользователь	База данных	Тип
^	5.11.2024, 21:38:03.63	PPro-ent-16			LOG

```
automatic vacuum of table "postgres.profile.last_stat_kcache_srv1": index scans: 1pages: 3 removed, 2 rer
cutoff: 7150, which was 1 XIDs old when operation endednew relfrozenxid: 7138, which is 16 XIDs ahead of
of total) had 0 tuples frozenindex scan needed: 4 pages from table (80.00% of total) had 59 dead item ide
```

work_mem - Задаёт базовый максимальный объём памяти, который будет использоваться во внутренних операциях при обработке запросов (например, для сортировки или хеш-таблиц), прежде чем будут задействованы временные файлы на диске.



hash_mem_multiplier - Используется для определения максимального объёма памяти, который может выделяться для операций с хешированием. Итоговый объём определяется произведением work_mem и hash_mem_multiplier.

Мониторинг work_mem

Как определить, что объем work_mem достаточен?

1. Отчет **pg_profile или pgpro_pwr**!

Statement statistics by database

Database	Calls	Time (s)				Fetched (blk)		Dirtied (blk)		Temp (blk)		Local (blk)	
		Exec	Read	Write	Trg	Shared	Local	Shared	Local	Read	Write	Read	Write
	21142063357	5434617.12	13951.10	55.84		3675989789344	828719501	1039147932	149806370	294562125	305989125	710679390	326520283
pg_profile	24942263	7769.07	0.74			945853344	94006	4794711	6458	5329774	7327746	7074	9846
postgres	1983052	3119.95	0.01			60947665		3					
template1	24	0.00				67							
Total	21168988696	5445506.14	13951.85	55.84		3676996590420	828813507	1043942646	149812828	299891899	313316871	710686464	326530129

2. Параметр **log_temp_files** = '0'; + **log_statements** = 'ALL‘;

Позволяет выявить top-sql запросы, которые не поместились в work_mem, и для выполнения которых потребовалось создание временных файлов на диске.

3. PPEM метрика «Temp files created»

Сколько памяти нужно выделить?

shared_buffers +

(work_mem * hash_mem_multiplier + temp_buffers) * max_connections +

work_mem * max_worker_processes +

можно ограничить максимальное использование памяти **max_backend_memory**

autovacuum_max_workers * autovacuum_work_mem

- не должен превышать максимальный объем памяти на сервере, чтобы избежать принудительного завершения основного процесса PostgreSQL OOM killer. Можно защититься отдав пространство под SWAP.

max_connections – не нужно выставлять огромным, т.к. это резервирует память.

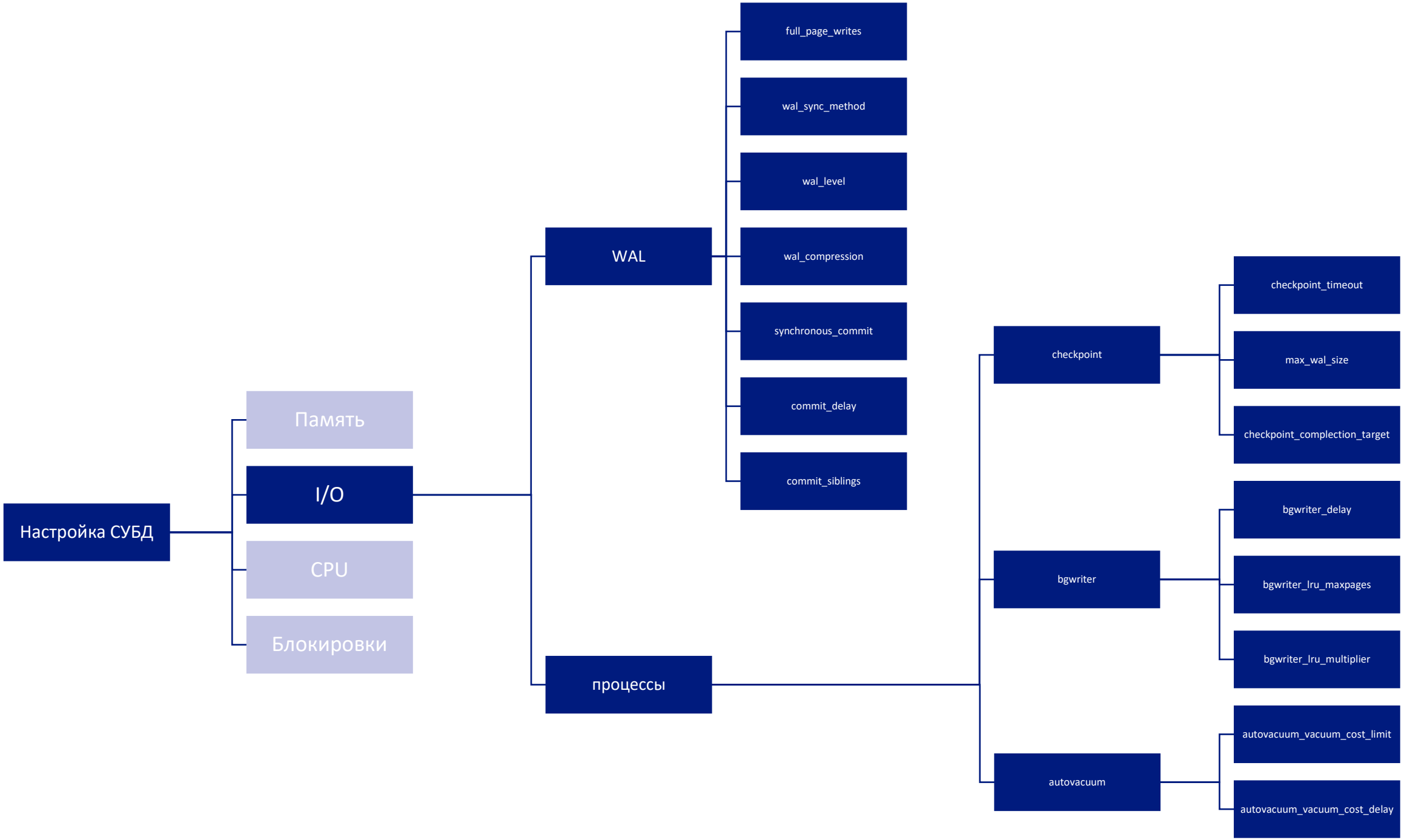
Прогрев кэша

Проблема: после перезагрузки СУБД кэш пустой.

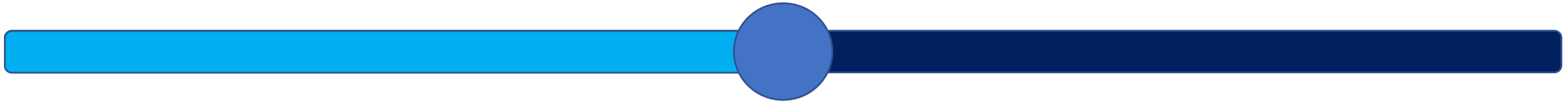
Решение: pg_prewarm

```
# postgresql.conf
shared_preload_libraries = 'pg_prewarm'

pg_prewarm.autoprewarm = true
pg_prewarm.autoprewarm_interval = 300s   --Задаёт интервал между обновлениями файла autoprewarm.blocks.
(mapping shared cache)
```


Настройка СУБД
Память
I/O
CPU
Блокировки
WAL
full_page_writes
wal_sync_method
wal_level
wal_compression
synchronous_commit
commit_delay
commit_siblings
процессы
checkpoint
checkpoint_timeout
max_wal_size
checkpoint_complection_target
bgwriter
bgwriter_delay
bgwriter_lru_maxpages
bgwriter_lru_multiplier
autovacuum
autovacuum_vacuum_cost_limit
autovacuum_vacuum_cost_delay

WAL



производительность

надёжность

~~fsync = {on , off}~~

wal_sync_method

- open_datasync
- fdatasync
- fsync
- open_sync
- …

full_page_writes = {on , off}

Утилита pg_test_fsync позволяет выбрать способ, наиболее подходящий для конкретной ОС и конкретной файловой системы.

```
root@sa:~# /opt/pgpro/ent-17/bin/pg_test_fsync
на тест отводится 5 сек.
O_DIRECT на этой платформе не поддерживается для open_datasync и open_sync.

Сравнение методов синхронизации файлов при однократной записи 8 КБ:
(в порядке предпочтения для "wal_sync_method", за исключением того, что в Linux предпочитается fdatasync)
        open_datasync                     106,654 оп/с       9376 мкс/оп
        fdatasync                         103,019 оп/с       9707 мкс/оп
        fsync                              52,562 оп/с      19025 мкс/оп
        fsync_writethrough                           н/д
        open_sync                          50,871 оп/с      19658 мкс/оп

Сравнение методов синхронизации файлов при двухкратной записи 8 КБ:
(в порядке предпочтения для "wal_sync_method", за исключением того, что в Linux предпочитается fdatasync)
        open_datasync                      54,458 оп/с      18363 мкс/оп
        fdatasync                         104,376 оп/с       9581 мкс/оп
        fsync                              52,102 оп/с      19193 мкс/оп
        fsync_writethrough                           н/д
        open_sync                          26,865 оп/с      37223 мкс/оп
```

- вредный совет

WAL

wal_compression = pglz,lz4,zstd,on,off - только для full_page_writes

WAL

synchronous_commit =

(можно менять на уровне транзакции)

remote_write
remote_apply
on
local
off ----------->

- Периоды синхранизации каждые **wal_writer_delay** (=200ms по умолчанию).
- Гарантируется согласованность, но не долговечность
- Зафиксированные транзакции могут пропасть за (3 × wal_writer_delay)

Где можно выиграть без потери надёжности:

commit_delay (ms)
commit_siblings (int)

Алгоритм: При числе одновременно работающих в системе транзакций более чем **commit_siblings**, процесс делает паузу, определяемую параметром **commit_delay**, в расчёте на то, что за время ожидания часть транзакций успеют завершиться и можно будет синхранизировать из записи за одну операцию.

Checkpoint

Периодический сброс всех грязных буферов на диск

- гарантирует попадание на диск всех изменений до контрольной точки
- ограничивает размер журнала, необходимого для восстановления

Частота срабатывания:

checkpoint_timeout

max_wal_size

Интенсивность работы:

checkpoint_completion_target = 0.9 14

Подход к настройке

PostgresPro

Вытесняет на диск грязные страницы из буферного кэша в фоне, позволяя пользовательским процессам не тратить на это время.

```
bgwriter_delay          = 200ms
bgwriter_lru_maxpages   = 100
bgwriter_lru_multiplier = 2
```

С маленькими значениями bgwriter_lru_maxpages и bgwriter_lru_multiplier уменьшается активность ввода/вывода со стороны процесса фоновой записи, но увеличивается вероятность того, что запись придётся производить непосредственно серверным процессам, что замедлит выполнение запросов.

Instance tuning

Сначала настраивается контрольная точка из соображений допустимого времени на восстановление, затем донастраивается bgWriter, таким образом, чтобы пользовательским процессам не приходилось вытеснять грязные буферы из кэша.

pg_stat_bgwriter. buffers_checkpoint	Количество буферов, записанных при выполнении контрольных точек
pg_stat_bgwriter. buffers_clean	Количество буферов, записанных фоновым процессом записи
pg_stat_bgwriter. buffers_backend	Количество буферов, записанных самим серверным процессом

15

pg_stat_io

```
select writes,write_time from pg_stat_io where backend_type = 'client backend' and object = 'relation‘ ;
```

Очистка

PostgresPro

VACUUM высвобождает пространство, занимаемое «мёртвыми» кортежами. При обычных операциях PostgreSQL кортежи, удалённые или устаревшие в результате обновления, физически не удаляются из таблицы; они сохраняются в ней, пока не будет выполнена команда VACUUM.

Регулирование агрессивности(производительности) автоматической очистки:

Процесс чередует работу и ожидание:
примерно vacuum_cost_limit условных единиц работы, затем засыпает на vacuum_cost_delay мс

vacuum_cost_limit = 200
autovacuum_vacuum_cost_delay = 2ms

Определяет общий пул работы для всех autovacuum_max_workers

bloat таблиц и индексов

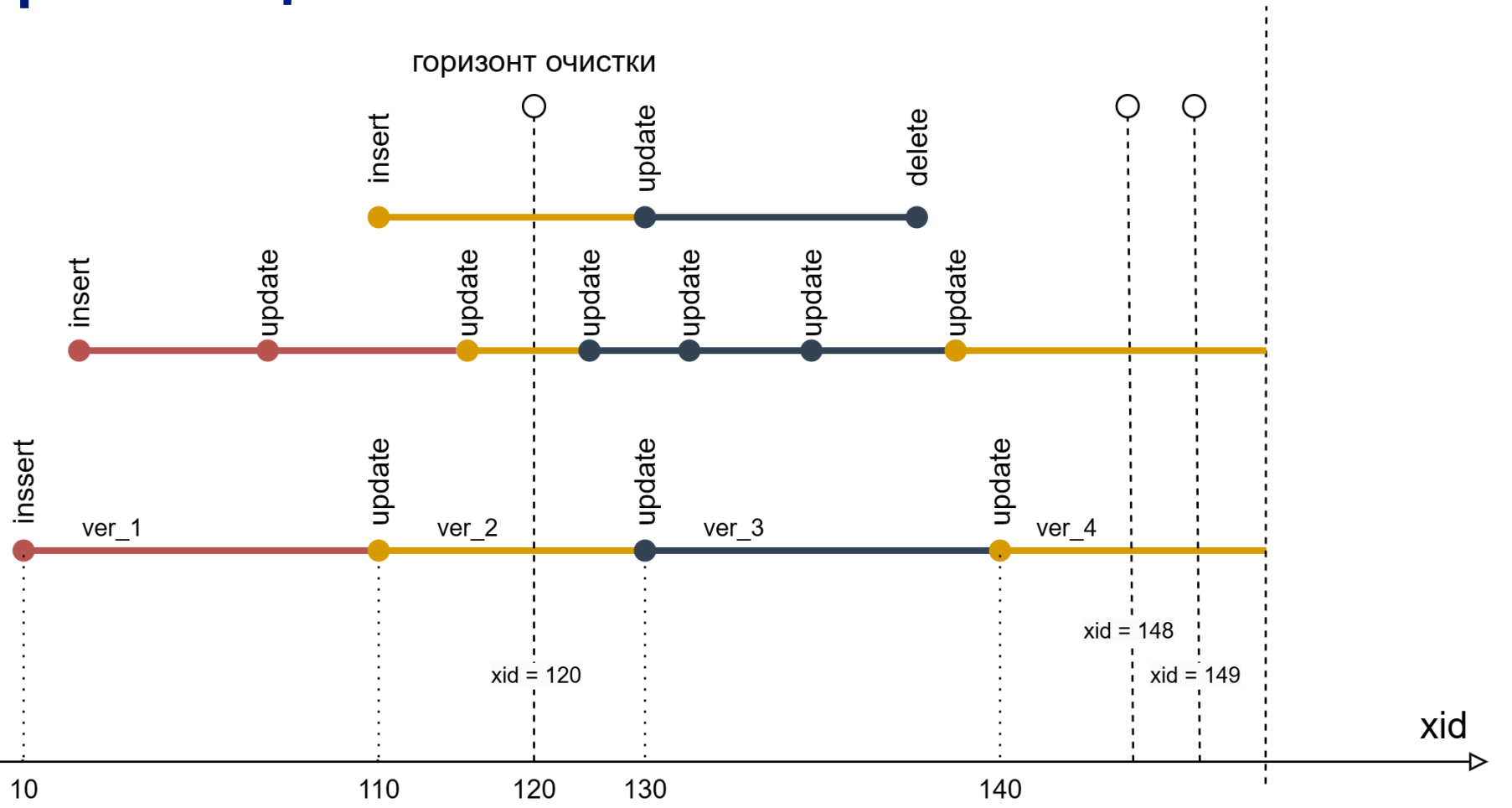
Механизм MVCC предполагает появление мёртвых строк.

Негативное влияние:

- перерасход места на диске
- замедление последовательного просмотра таблиц
- уменьшение эффективности индексного доступа (лишние переходы из индекса на мёртвые табличные строки, т.к. в индексной записи нет информации о статусе табличной строки)

Долгие транзакции

pg_xact (clog)	
xid	status
110	commit
...	...
120	active
...	...
130	commit
140	commit
141	commit
...	...
148	active
149	active



current_txid =150
текущее значение счётчика транзакций

может быть очищина autovacuum

попадает в снимок активной транзакции, не может быть очищена

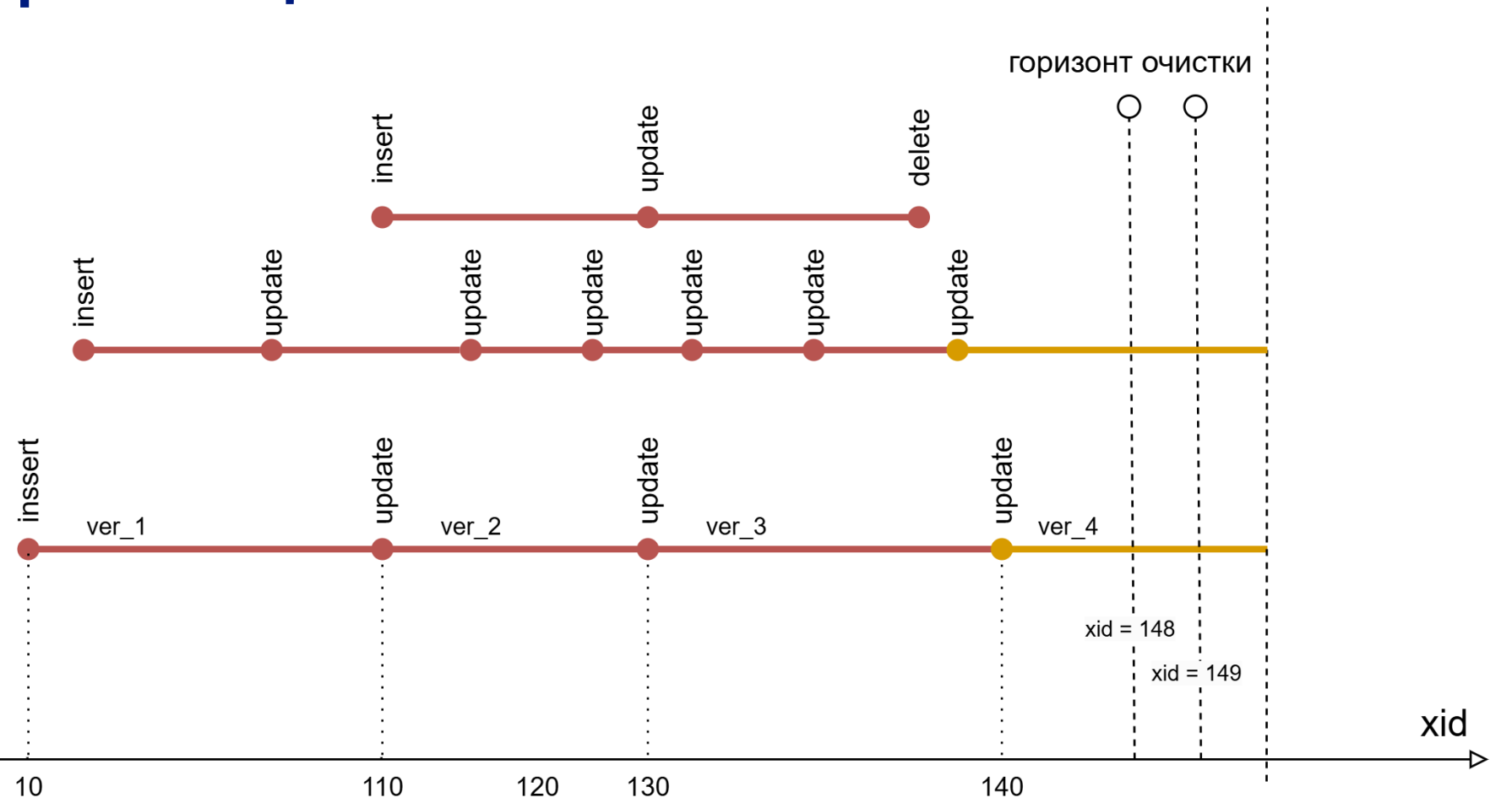
неактуальная версия, не видна ни в какой транзакции, но не может быть очищена

Долгие транзакции

PostgresPro

pg_xact (clog)

xid	status
110	commit
...	...
120	commit
...	...
130	commit
140	commit
141	commit
...	...
148	active
149	active



может быть очищина autovacuum

попадает в снимок активной транзакции, не может быть очищена

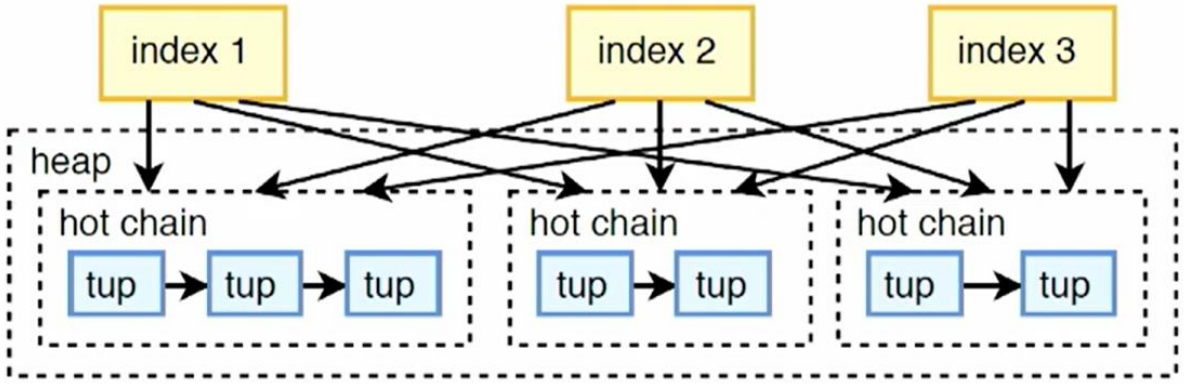
неактуальная версия, не видна ни в какой транзакции, но не может быть очищена

current_txid =150
текущее значение счётчика транзакций

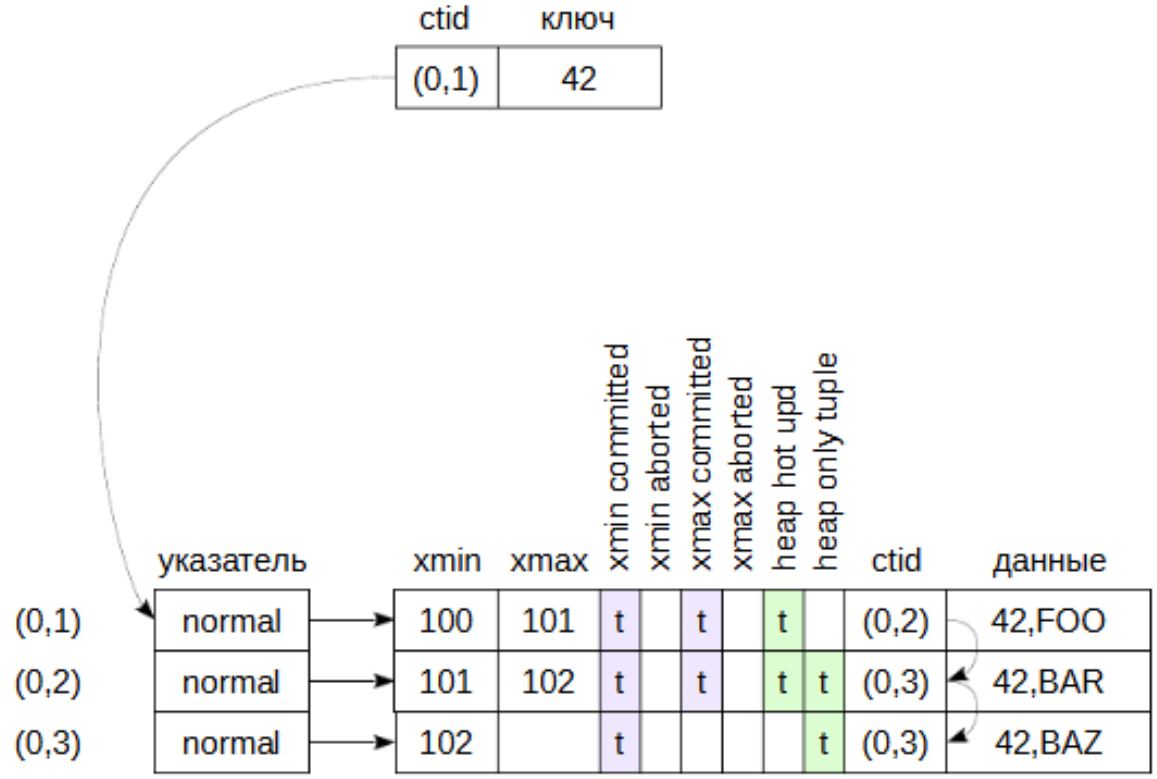
HOT-update

heap only tuple. Цепочка обновлений — только в пределах одной страницы

Для высоконагруженных UPDATE-ами таблиц.



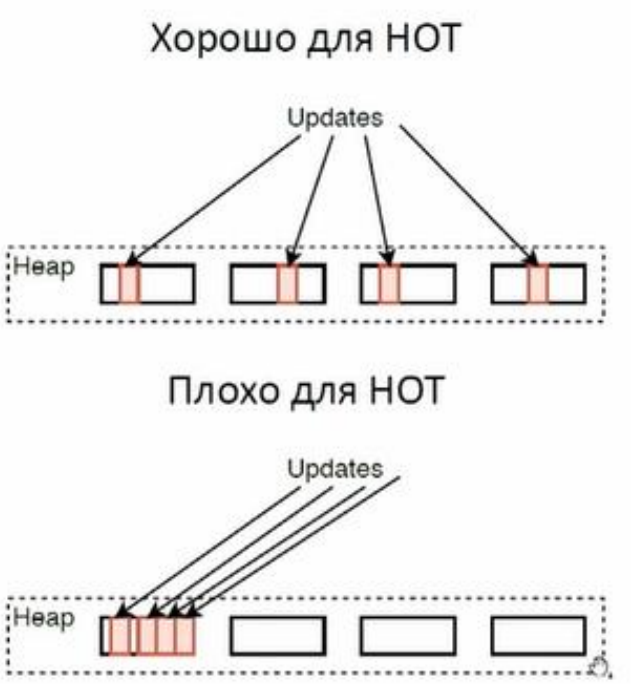
- если в табличной странице не хватает места для новой версии, цепочка обрывается (как если бы оптимизация не работала)
- место в странице можно зарезервировать, уменьшив параметр хранения таблицы fillfactor (100 % → 10 %)



fillfactor

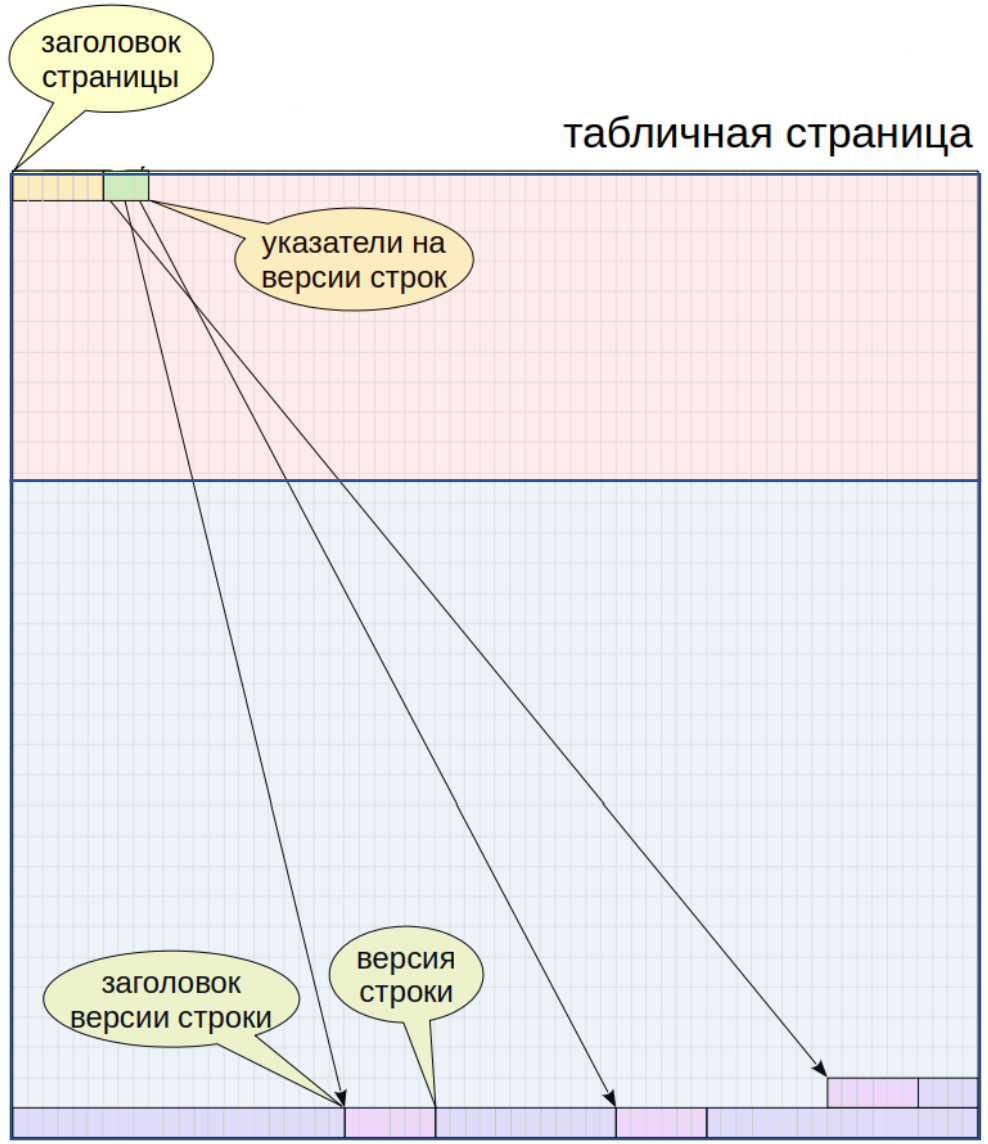
Плотность заполнения блоков

ALTER TABLE … SET (fillfactor = 70);



Резерв под UPDATE

Свободное место



fillfactor=70

Настройка fillfactor

Большая разница между:
pg_stat_all_tables.n_tup_upd
pg_stat_all_tables.n_tup_hot_upd

=> Уменьшать fillfactor

```
postgres=# select relname, n_tup_upd, n_tup_hot_upd from pg_stat_all_tables where relname ~ 'pgbench_accounts' \gx
-[ RECORD 1 ]-+-----------------
relname       | pgbench_accounts
n_tup_upd     | 1245448
n_tup_hot_upd | 226944
```

Стандартные рекомендации по сокращению I/O

- **Выделенный СХД под WAL**

Заменить $PGDATA/pg_wal симлинком, указывающим на другое расположение

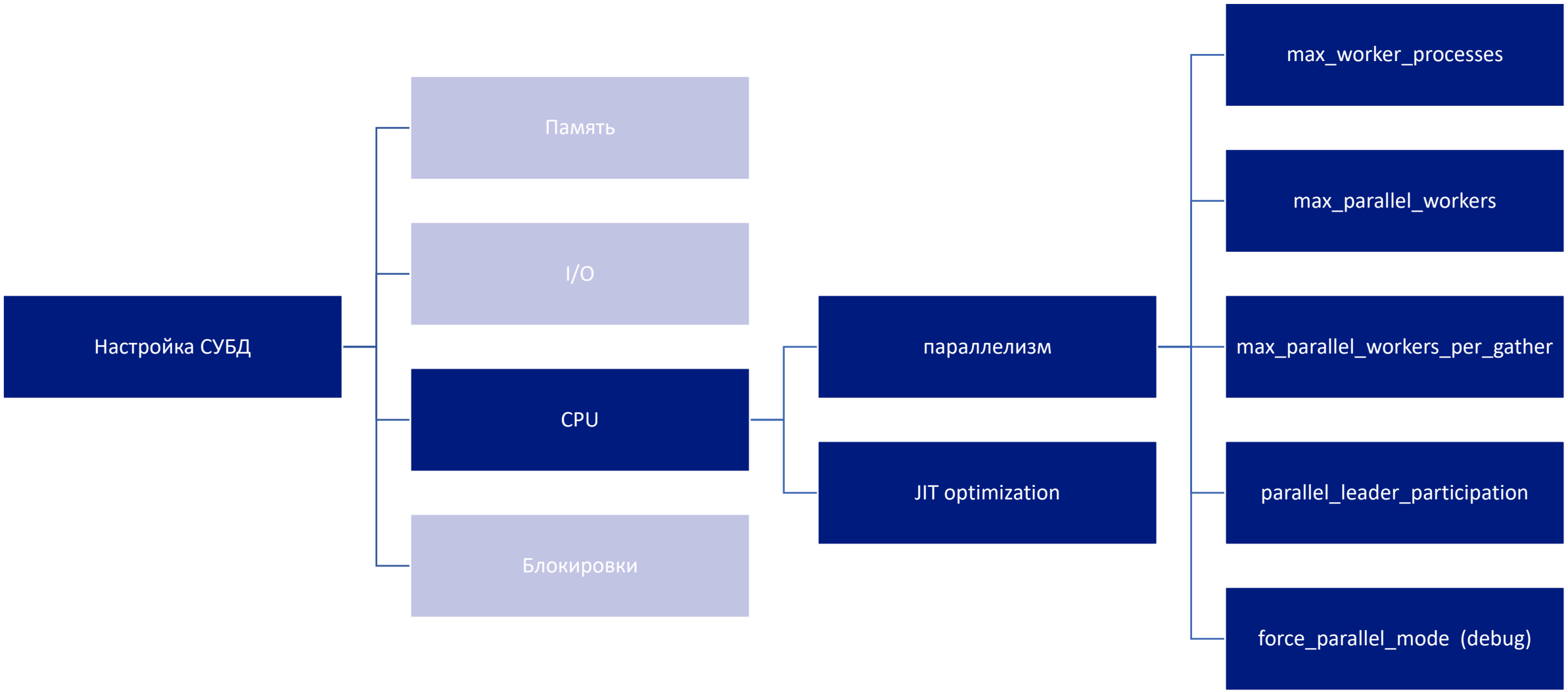
- **temp_tablespaces (string)**

Эта переменная задаёт табличные пространства, в которых будут создаваться временные объекты (временные таблицы и индексы временных таблиц), которые не поместились в **temp_buffers**. В этих табличных пространствах также создаются временные файлы для внутреннего использования, если не поместились в **work_mem.**

- **Размещение temp_tablespaces** в tmpfs

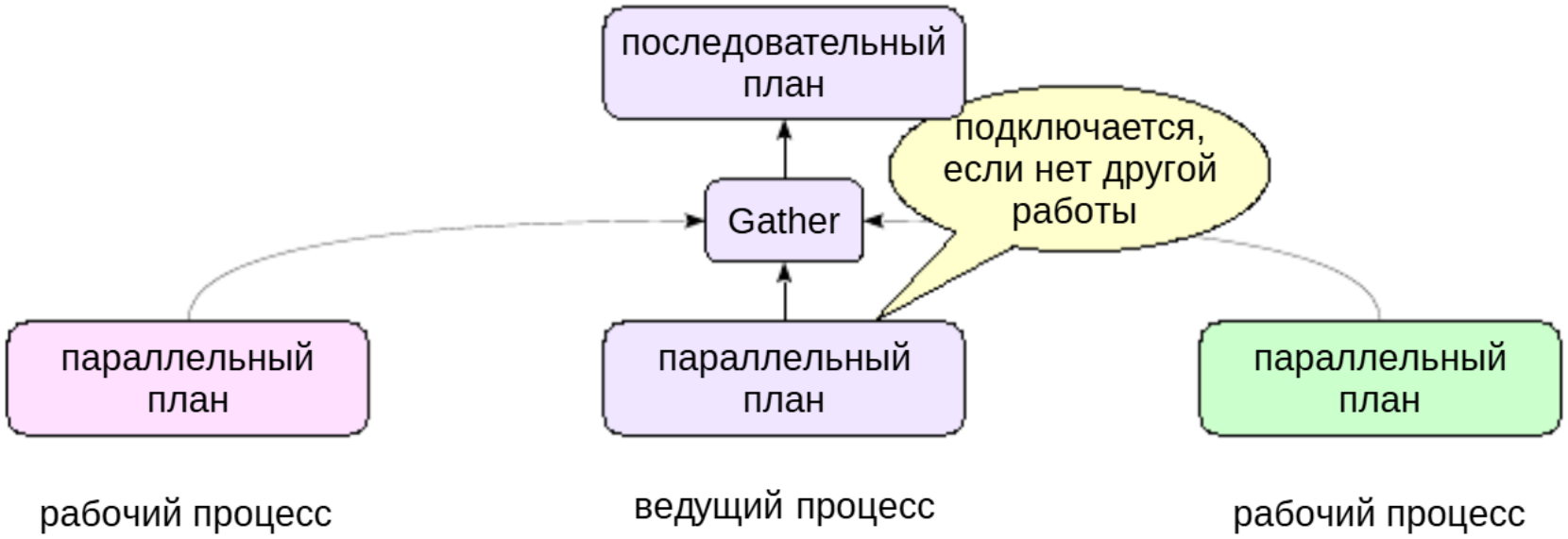
- Секционирование нагруженных таблиц и размещение секций в разных пространствах (ILM).

- **CFS (сжатый кэш на уровне ОС) +** wal_compression


Настройка СУБД
Память
I/O
CPU
Блокировки
параллелизм
JIT optimization
max_worker_processes
max_parallel_workers
max_parallel_workers_per_gather
parallel_leader_participation
force_parallel_mode (debug)

Параллелизм

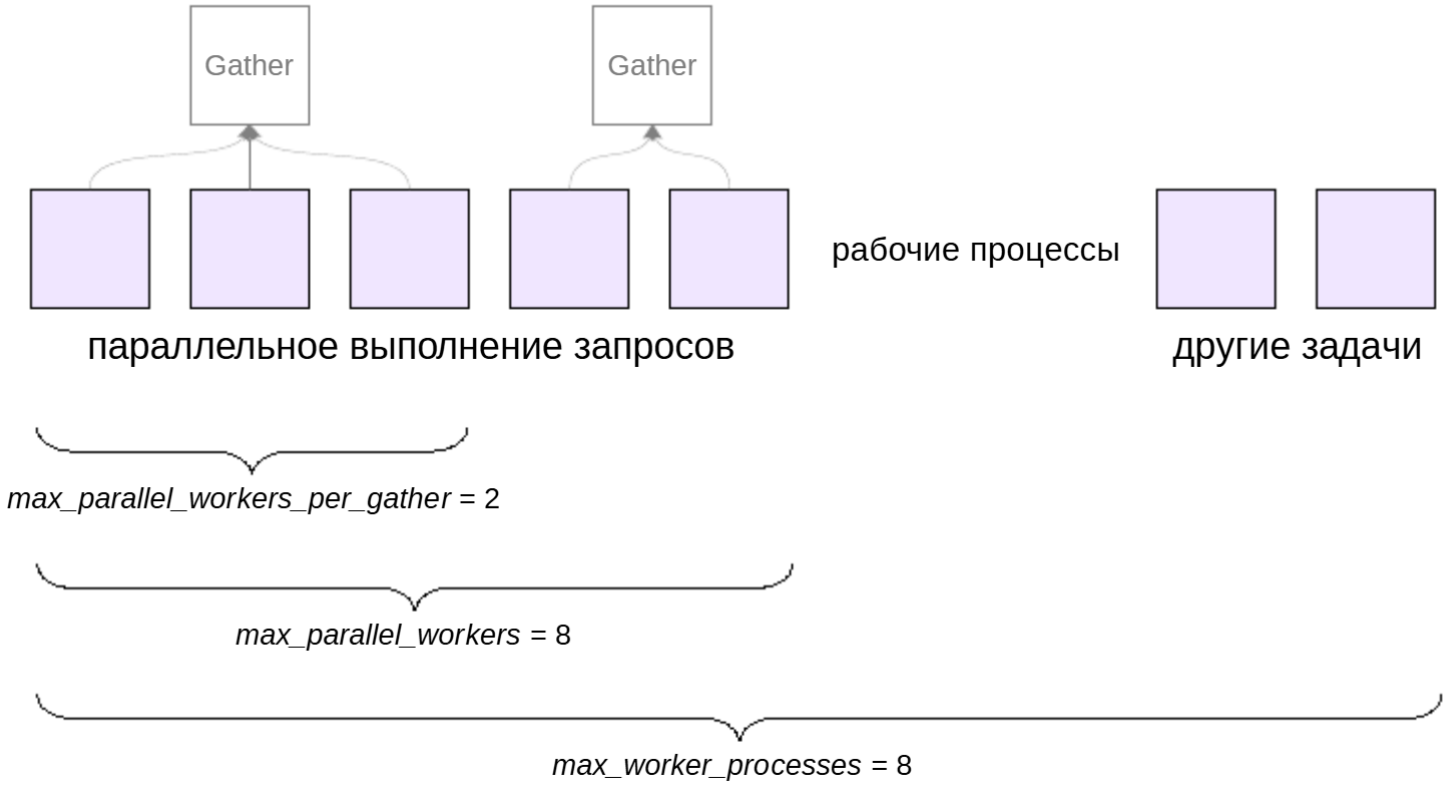
сжечь больше ресурсов, чтобы быстрее выполнить задачу



Параллелизм

Параметр конфигурации	Описание параметра	Значение по умолчанию
max_worker_processes	Задаёт максимальное число фоновых процессов, которое можно запустить в текущей системе.	8
max_parallel_workers	Задаёт максимальное число рабочих процессов, которое система сможет поддерживать для параллельных операций. Заметьте, что значение данного параметра, превышающее max_worker_processes, не будет действовать, так как параллельные рабочие процессы берутся из пула рабочих процессов, ограничиваемого этим параметром	8
max_parallel_workers_per_gather	Задаёт максимальное число рабочих процессов, которые могут запускаться одним узлом Gather	2

Параллелизм



Возникающие ожидания в параллельных планах

Событие ожидания **IPC**	Описание
MessageQueueInternal	Ожидание подключения другого процесса к общей очереди сообщений.
MessageQueuePutMessage	Ожидание записи сообщения протокола в общую очередь сообщений.
MessageQueueReceive	Ожидание получения байтов из общей очереди сообщений
MessageQueueSend	Ожидание передачи байтов в общую очередь сообщений

Накладные расходы

pid	wait_event	wait_event_type	duration	state	query
5930	[null]	[null]	00:00:03.452916	active	select subject.subject_id, addr.address_id.....
8665	MessageQueueSend	IPC	00:00:03.452310	active	select subject.subject_id, addr.address_id...
11330	MessageQueueSend	IPC	00:00:03.322182	active	select subject.subject_id, addr.address_id.....
13185	[null]	[null]	00:00:02.452919	active	select subject.subject_id, addr.address_id.....
17007	MessageQueueSend	IPC	00:00:02.442918	active	select subject.subject_id, addr.address_id.....
31220	MessageQueueSend	IPC	00:00:02.442917	active	select subject.subject_id, addr.address_id.....
47051	[null]	[null]	00:00:02.432916	active	select subject.subject_id, addr.address_id.....
55207	MessageQueueSend	IPC	00:00:02.432915	active	select subject.subject_id, addr.address_id.....

JIT

Just-In-Time compilation, компиляция «во время»

- JIT компиляция выполняется непосредственно во время выполнения запроса.
- JIT компиляция преобразует интерпретируемый вариант исполнения программы в машинный код.
- Применяется для ускорения вычисления сложных выражений в предикатах запросов.
- Если сложную функцию, вычисляемую в предикате, скомпилировать в машинный код, который выполняет процессор, то она, возможно, будет выполняться быстрее.

Оптимизации:

- Преобразование кортежей (Tuple deforming) — перевод кортежа с диска в представление в памяти.
  Преобразование кортежей можно ускорить, создав функции, предназначенные для определённой структуры таблицы и количества извлекаемых столбцов.
- Для сокращения затрат JIT компиляция может встраивать (inline) код функций непосредственно в код выражений, вызывающих эти функции.
- LLVM поддерживает оптимизации скомпилированного машинного кода

JIT

jit_above_cost

Устанавливает предел стоимости запроса, при превышении которого включается JIT-компиляция

cost > 100_000

jit_inline_above_cost

Устанавливает предел стоимости, при превышении которого будет допускаться встраивание функций и операторов в процессе JIT-компиляции.

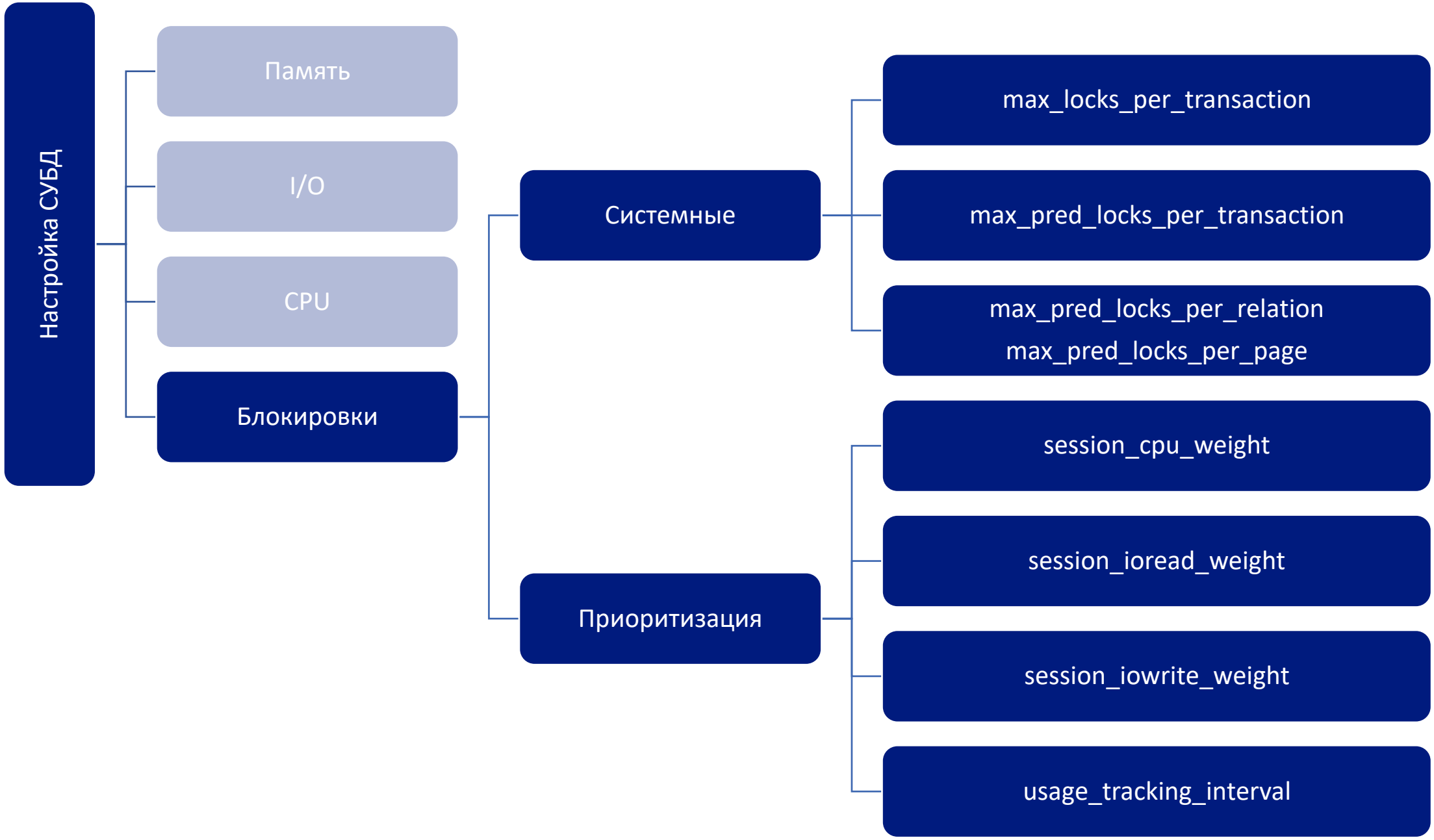
jit_optimize_above_cost

Устанавливает предел стоимости, при превышении которого в JIT-компилированных программах может применяться дорогостоящая оптимизация.

cost > 500_000

OLTP – отключать

HTAP\OLAP – возможно включать


Настройка СУБД
Память
I/O
CPU
Блокировки
Системные
Приоритизация
max_locks_per_transaction
max_pred_locks_per_transaction
max_pred_locks_per_relation
max_pred_locks_per_page
session_cpu_weight
session_ioread_weight
session_iowrite_weight
usage_tracking_interval

Пул блокировок

Информация в общей памяти сервера
представление pg_locks:
locktype — тип блокируемого ресурса,
mode — режим блокировки
Ограниченное количество: max_locks_per_transaction × max_connections

Реализация уровня изоляции serializable

Типы ресурсов
Relation — отношение
Page — страница
max_pred_locks_per_page

max_pred_locks_per_relation
tuple — версия строки

Режим:
SIRead

Resource Management Plans pgpro_rp

PostgresPro

16.1

- Приоритеты для пользователей и сессий при ограниченном ресурсе
- Методы выделения ресурсов:
  - CPU, чтение, запись
  - Приоритеты 1, 2, 4, 8
  - По умолчанию - 4
- Группы потребителей
  - пользователь может быть членом нескольких групп
  - только одна группа активна во время сеанса
  - группа по умолчанию назначается пользователю при соединении с БД
- Планы и приоритеты можно менять не останавливая PG
- Автоматическое назначение в группу при login
- Автоматическое/ручное переключение планов
- Сеансы работают без ограничений, если ресурсы свободны

WEEKDAY_PLAN

Resource Consumer Group	Resource Plan Directives
Order Entry	CPU = 1 Приоритет по чтению = 4
Shipping	CPU = 2 Приоритет по чтению = 4

NIGHT_PLAN

Resource Consumer Group	Resource Plan Directives
Billing	CPU = 4 Приоритет по чтению = 4
Order Entry	CPU 2 Приоритет по записи = 2
Shipping	CPU = 2 Приоритет по чтению = 2

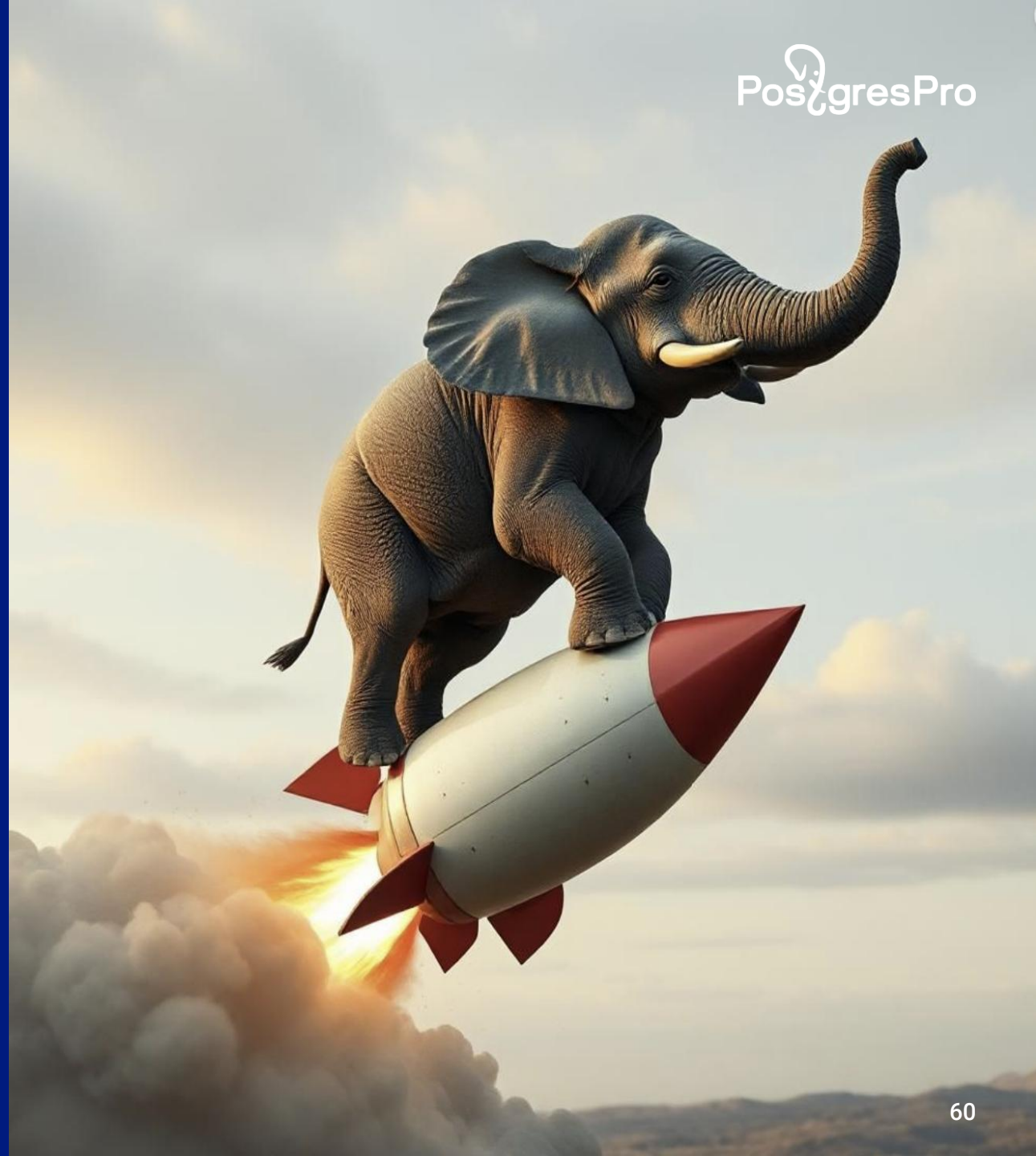
https://postgrespro.ru/docs/enterprise/17/pgpro-rp

pgpro_tune

База данных на максимум одним нажатием.

Особенно критична становится эта проблема при нехватке времени или необходимости быстро масштабироваться. Ручная настройка может превратиться в серьёзное узкое место при развитии инфраструктуры и запуске новых сервисов.

Команда Postgres Professional реализовала возможность автоматически получить конфигурацию Postgres Pro, учитывающую особенности вашего оборудования и нагрузок, всего за несколько минут, избегая при этом длительных погружений в документацию и многочасовые эксперименты. Знакомьтесь — pgpro_tune.



Принцип работы

Решение pgpro_tune — «автотюнер» для Postgres Pro, позволяющий автоматически оптимизировать конфигурацию сервера под оборудование и задачи.

Сбор информации о системе.

pgpro_tune анализирует конфигурацию сервера, определяя количество процессорных ядер, объем оперативной памяти и другие важные параметры оборудования.

Вызов экспертной логики.

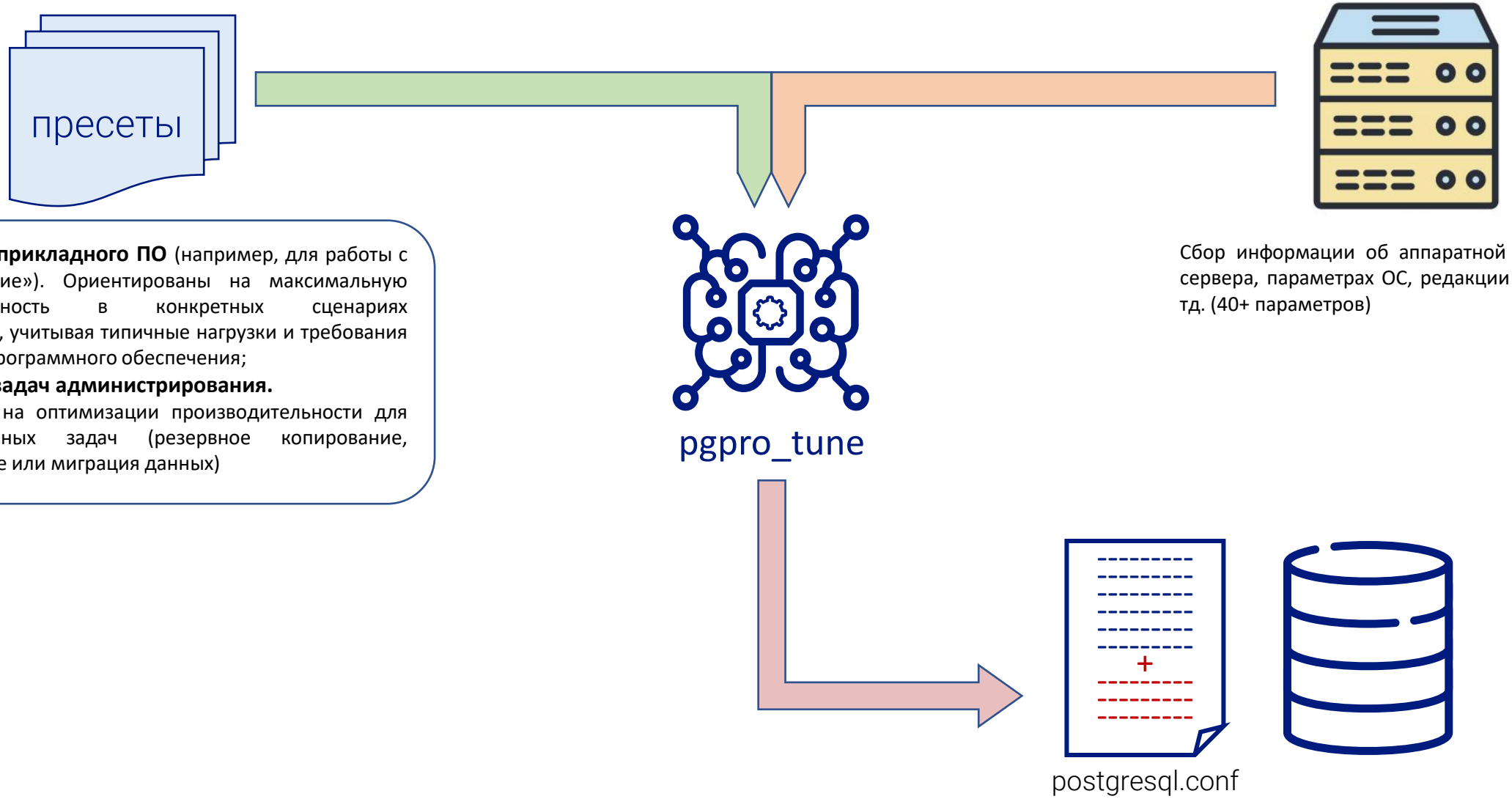
pgpro_tune передает собранную информацию о системе набору специализированных shell-скриптов, содержащих экспертные знания и логику настройки, разработанные специалистами Postgres Professional.

Формирование итоговой конфигурации.

Shell-скрипты анализируют входные данные и предлагают оптимальные значения для различных параметров конфигурации. Утилита pgpro_tune агрегирует эти предложения и генерирует итоговый блок настроек.

Применение настроек.

Утилита записывает сгенерированные настройки в файл postgresql.conf, снабжая их подробными комментариями и указанием даты и времени автонастройки.

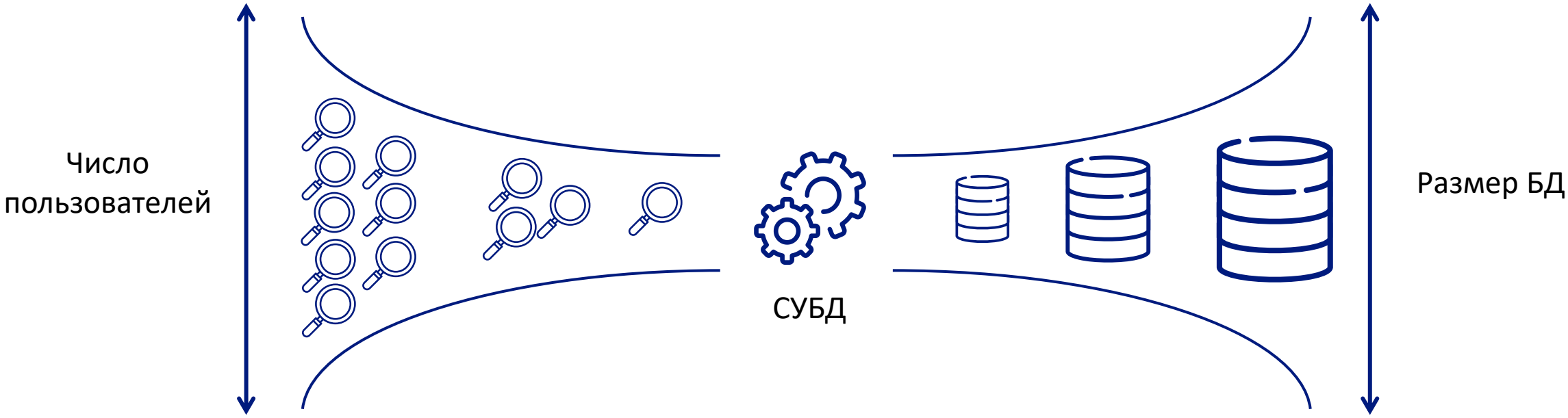


пресеты для прикладного ПО (например, для работы с «1С:Предприятие»). Ориентированы на максимальную производительность в конкретных сценариях использования, учитывая типичные нагрузки и требования прикладного программного обеспечения;

пресеты для задач администрирования. Фокусируются на оптимизации производительности для административных задач (резервное копирование, восстановление или миграция данных)

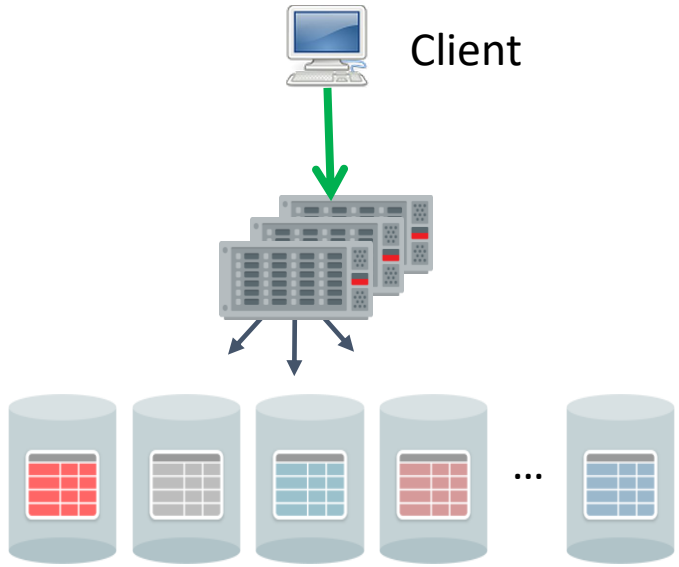
Сбор информации об аппаратной конфигурации сервера, параметрах ОС, редакции Postgres Pro и тд. (40+ параметров)

Проблема масштабируемости

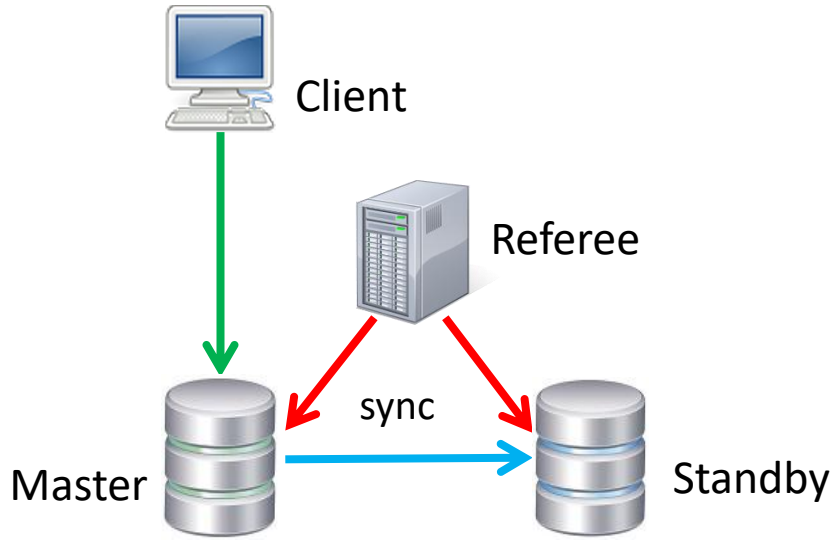


Для высоко нагруженных систем

SHARDMAN

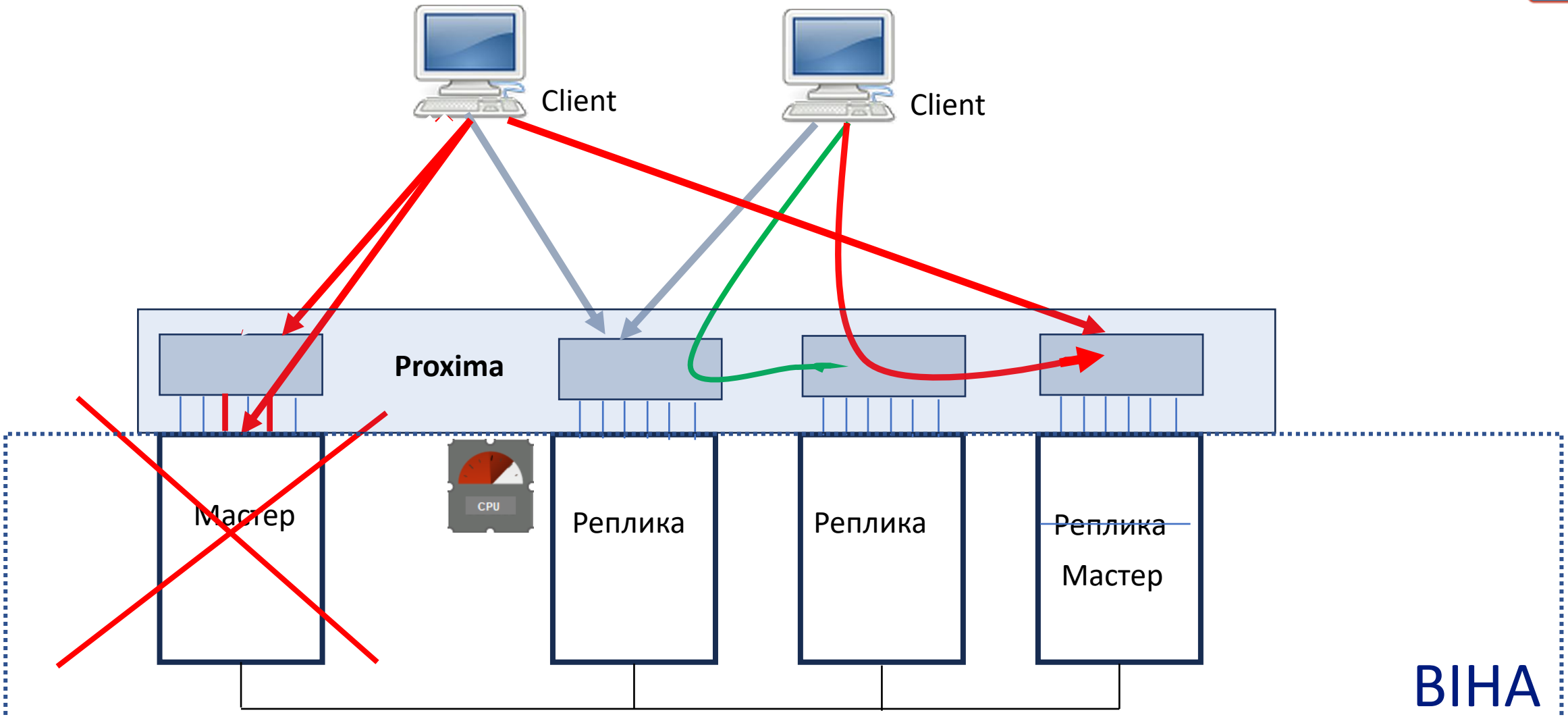


BiHA + proxima load balancer



PROXIMA = Pooler + Proxy + Load Balancer

17.2



ВІНА

Спасибо за внимание!

Q&A