LOXODATA

BE PRO-ACTIV ON POSTGRESQL PERFORMANCE

WARSAW

24/10/2017

Lætitia AVROT
Stéphane SCHILDKNECHT
Loxodata

WHO

Lætitia Avrot

- PostgreSQL advisor and teacher
- DBA PostgreSQL for more than 10 years (and also Oracle and SQL Server)
- @l_avrot

WHO

Stéphane Schildknecht

- Founder of Loxodata
- PostgreSQL lover for more than 15 years
- Founder of PostgreSQLFr (chairman from 2005 to 2010)
- @saschild

LOXODATA

Company built on 3 essential pillars

PostgreSQL

DevOps

Cloud

LOXODATA

LOXODATA

A comprehensive service offer

- Architecture
- Consulting
- Teaching
- Administration
- Audit
- Support

WHO

You

- Who has never run postgreSQL in production?
- Are you a DBA?
- Are you a developper?

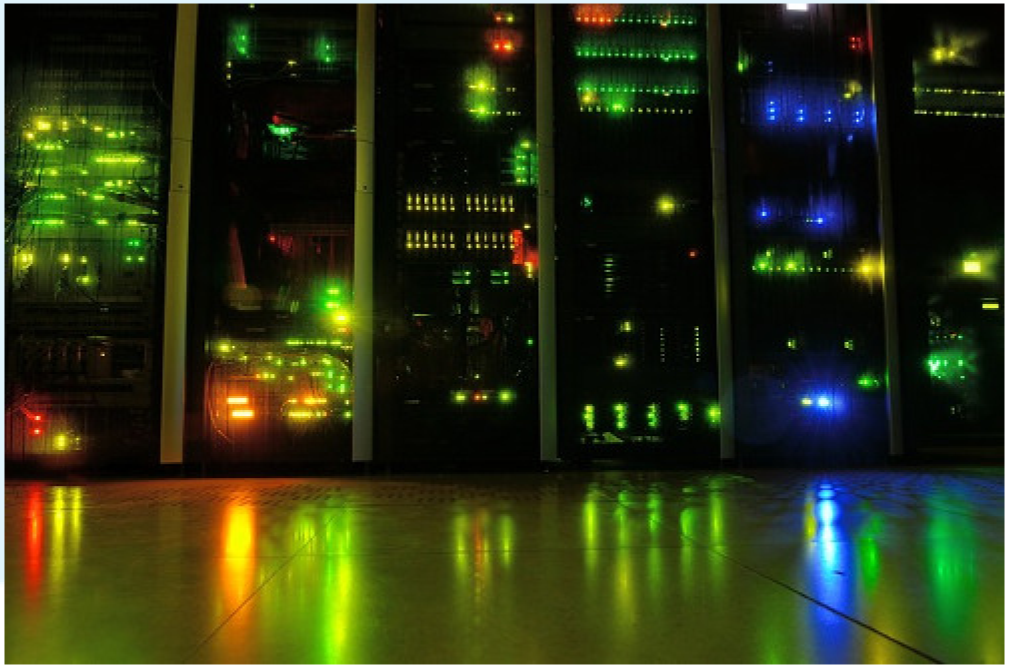
LOXODATA

WHAT

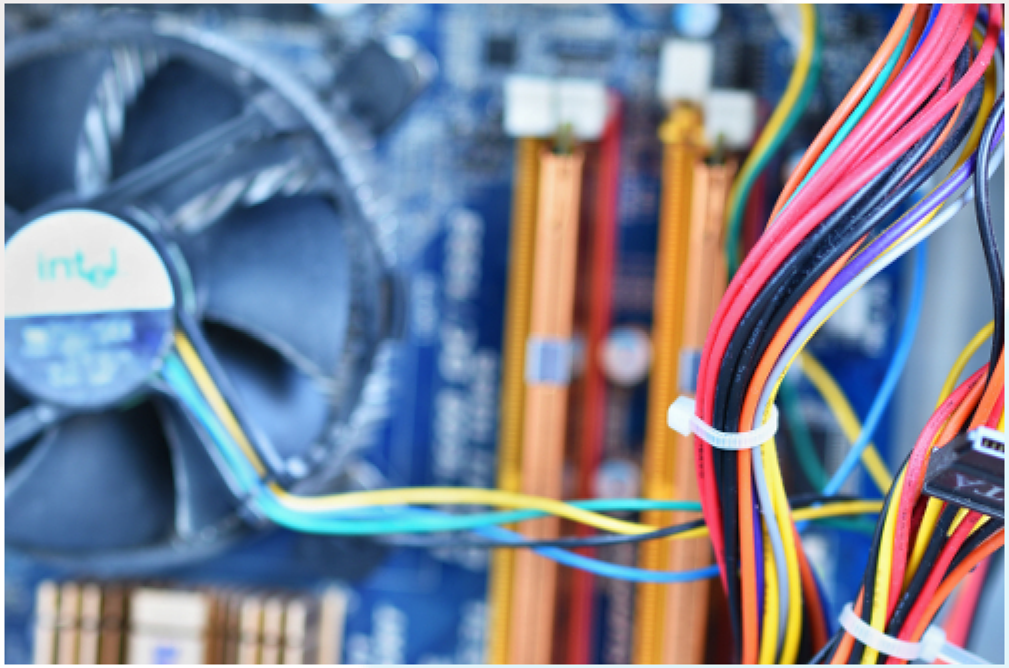
Get the best from your PostgreSQL

- Who already had performance issue with PostgreSQL?
- How can you speedup your PostgreSQL server?

HARDWARE



LOXODATA



First things first!

- IO
- RAM
- CPU

Don't trust your vendor, bench!

LOXODATA

MODELING YOUR DATABASE

A good model is a good start

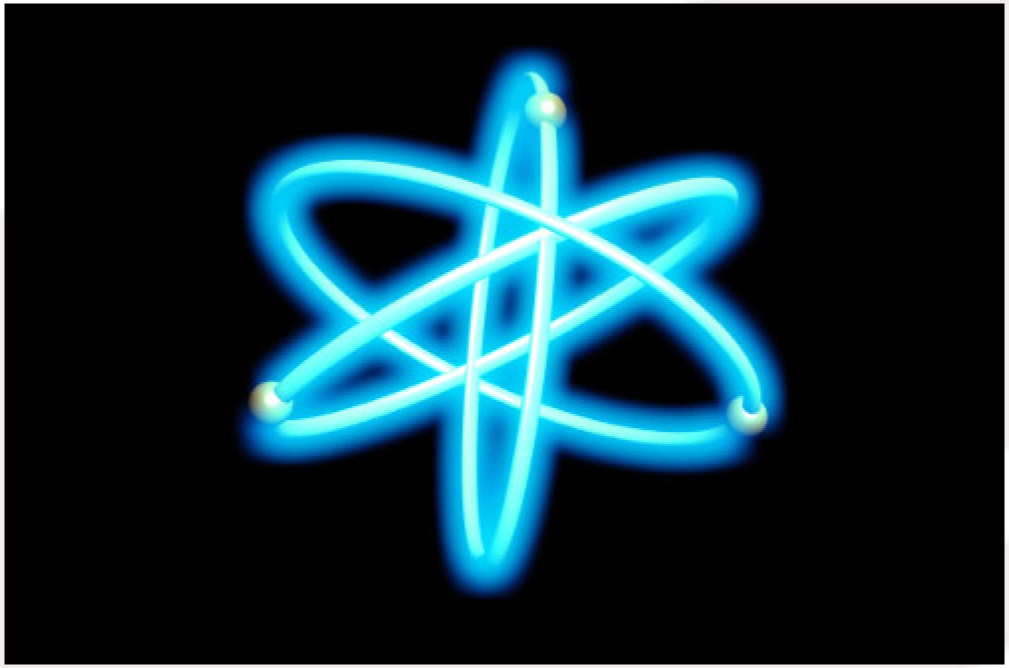
- Types
- Type alignement
- Indexing?

SOME THEORY

ACID

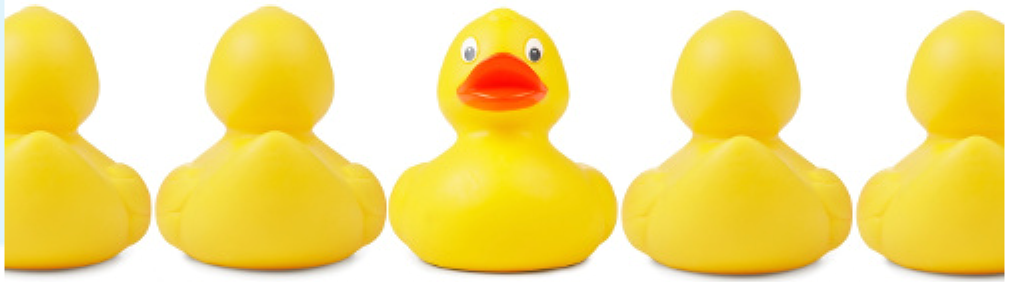
- Atomicity
- Consistency
- Isolation
- Durability

LOXODATA



Atomicity

> *Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.*



Consistency

The consistency property ensures that any transaction will bring the database from one valid state to another.

LOXODATA

Isolation

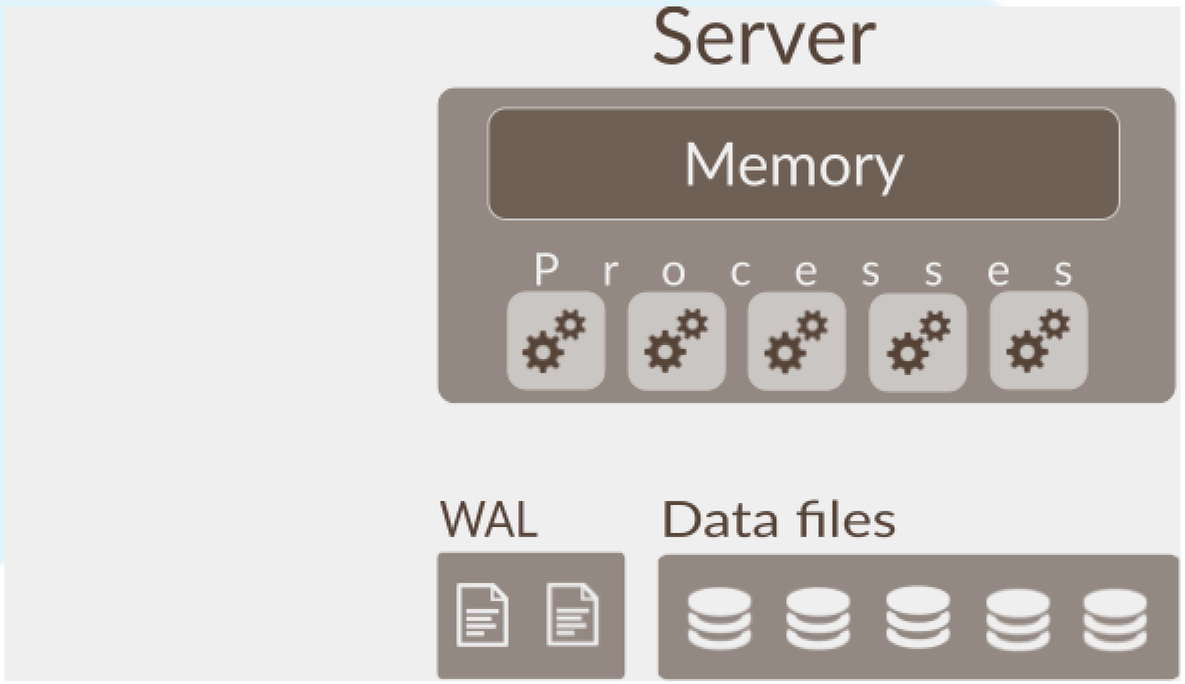
The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other.

Durability

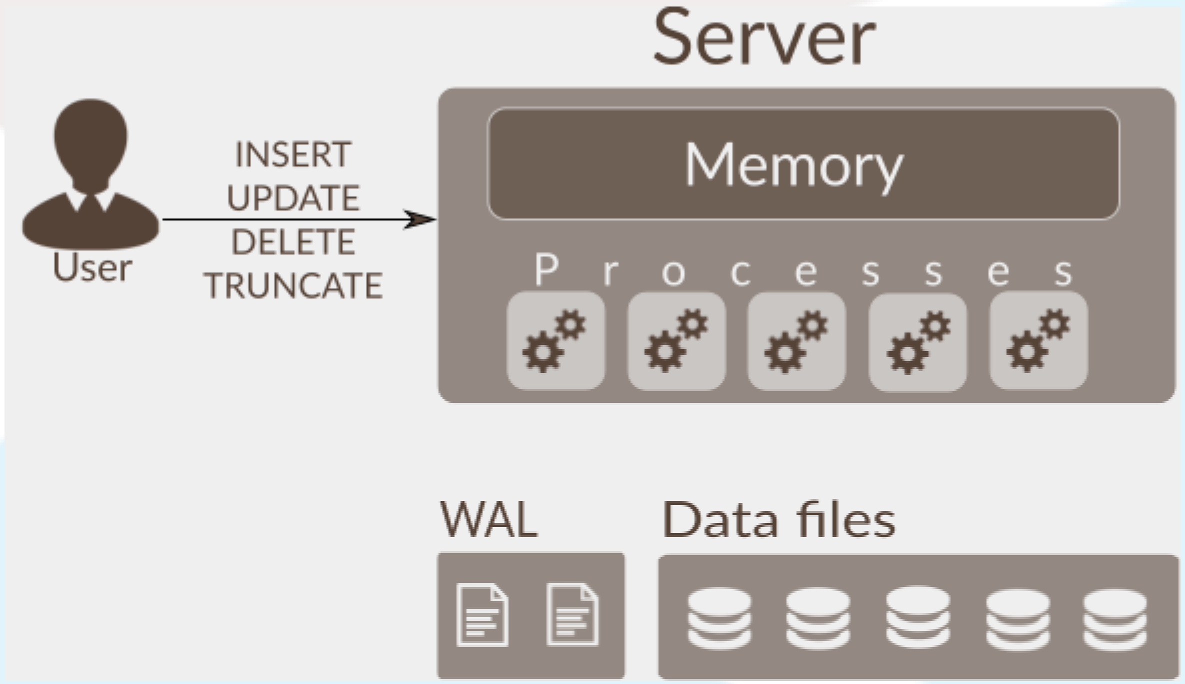
> *The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.*

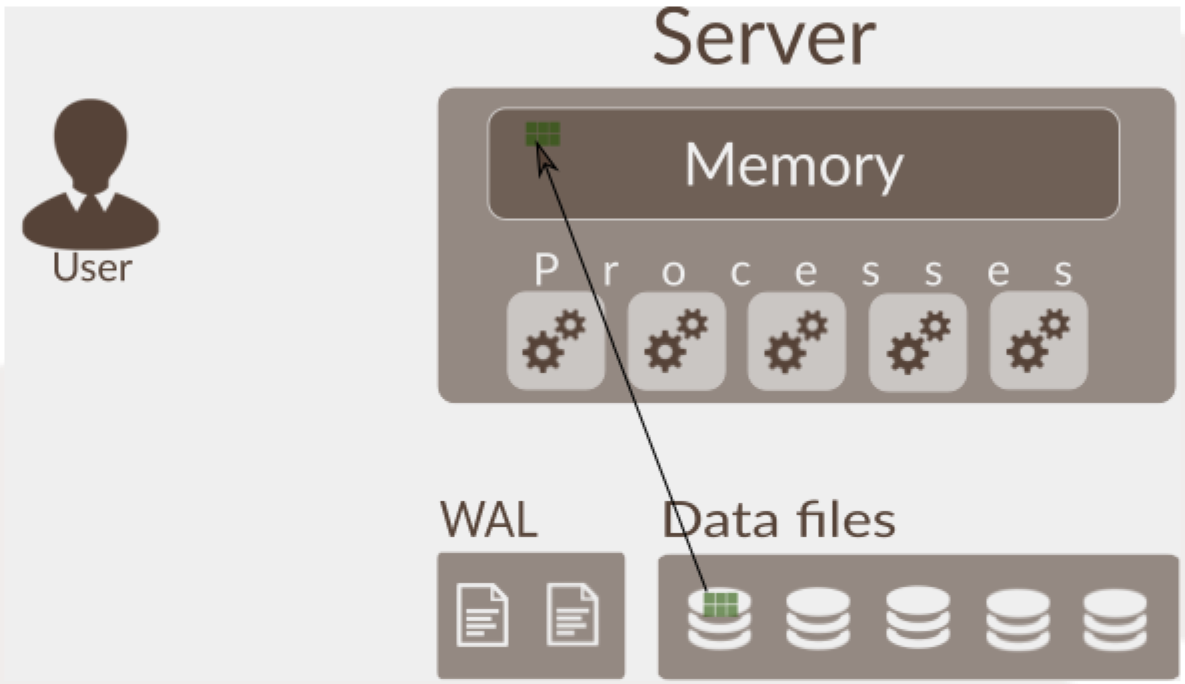
(All quotes from Wikipedia)

LOXODATA

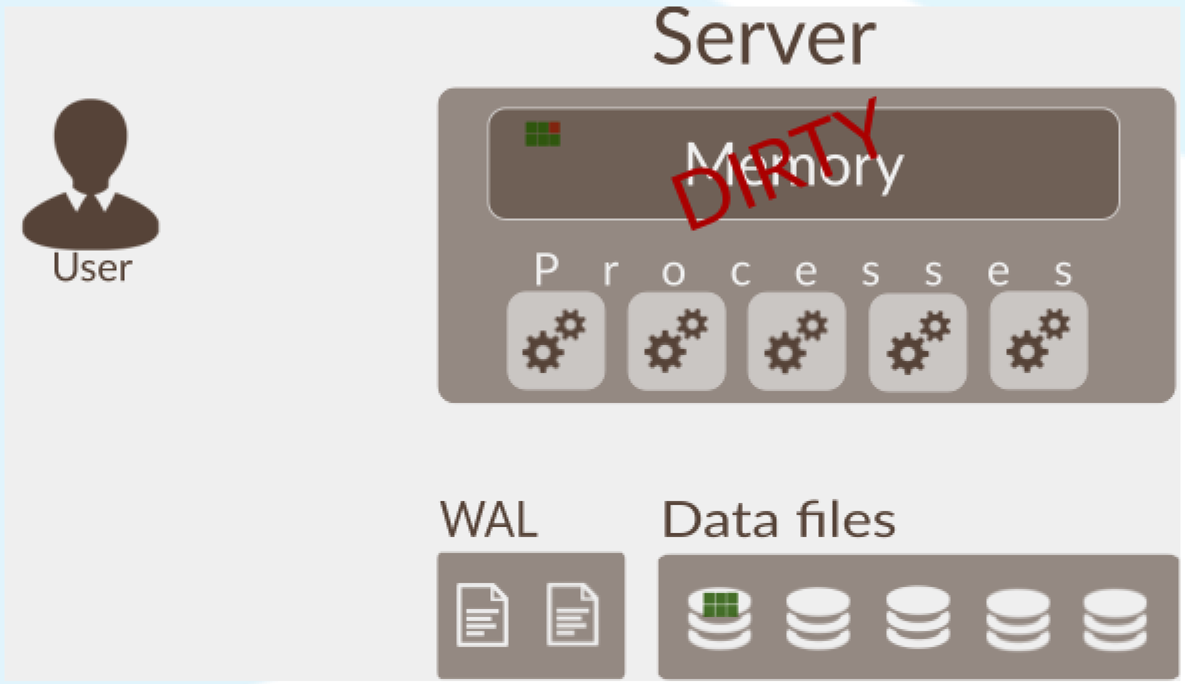


LOXODATA

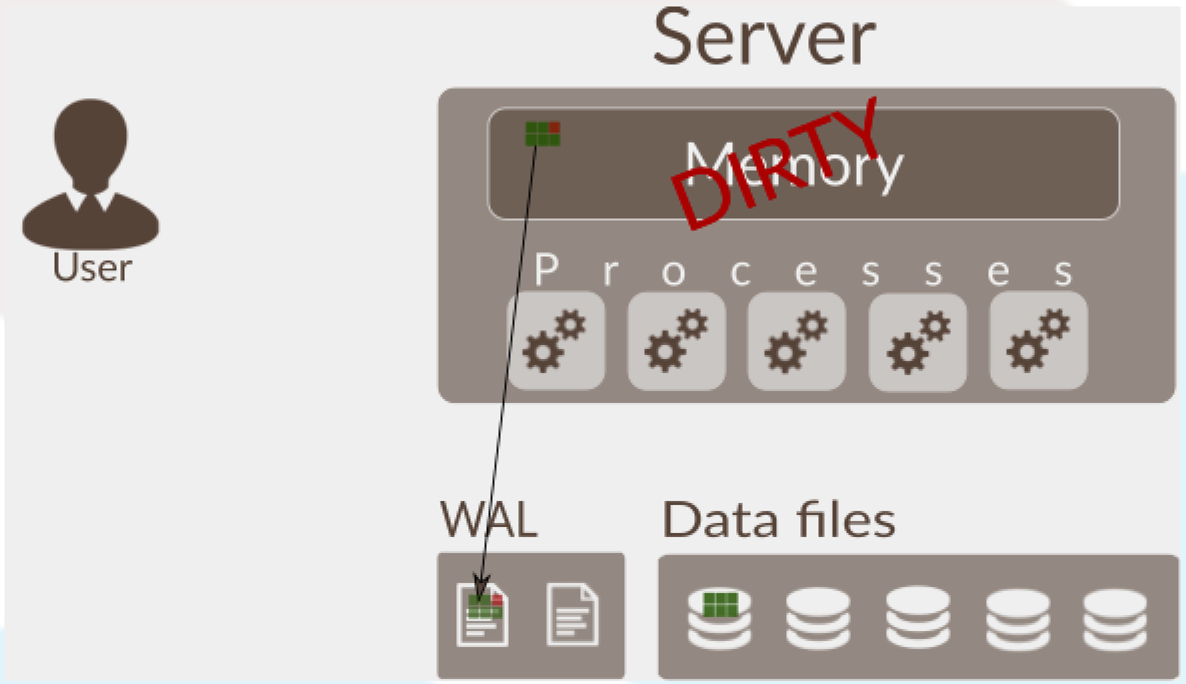

Server
User
INSERT
UPDATE
DELETE
TRUNCATE
Memory
Processes
WAL
Data files

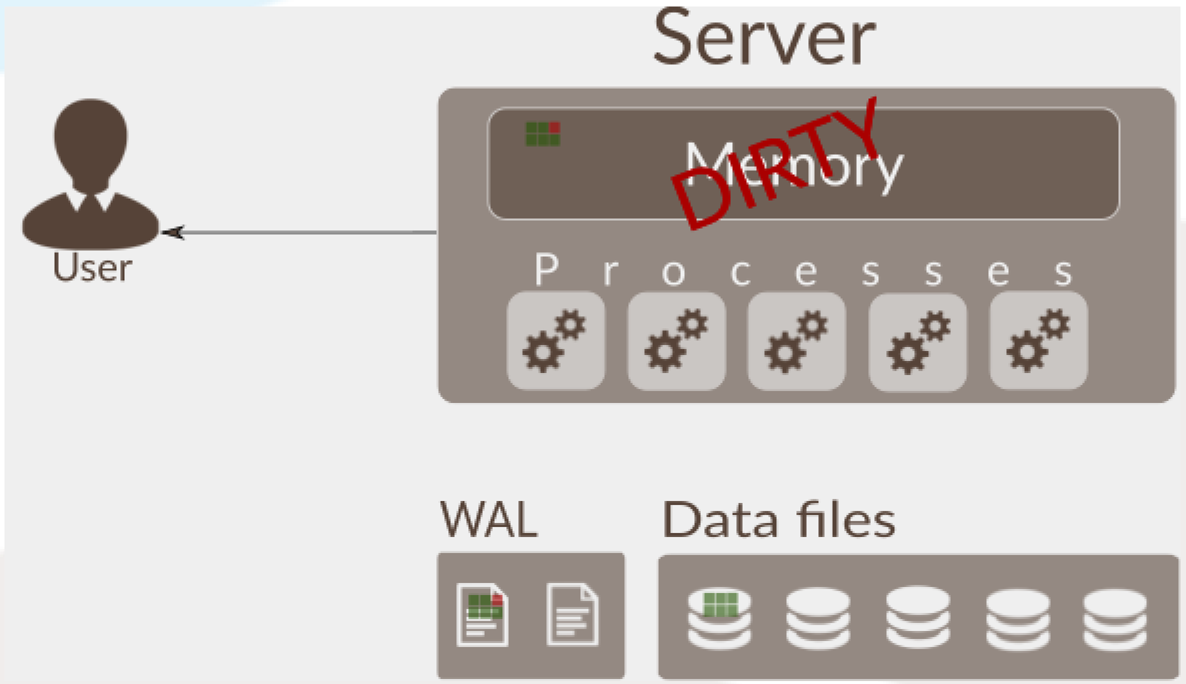

Server
User
Memory
P r o c e s s e s
WAL
Data files

LOXODATA

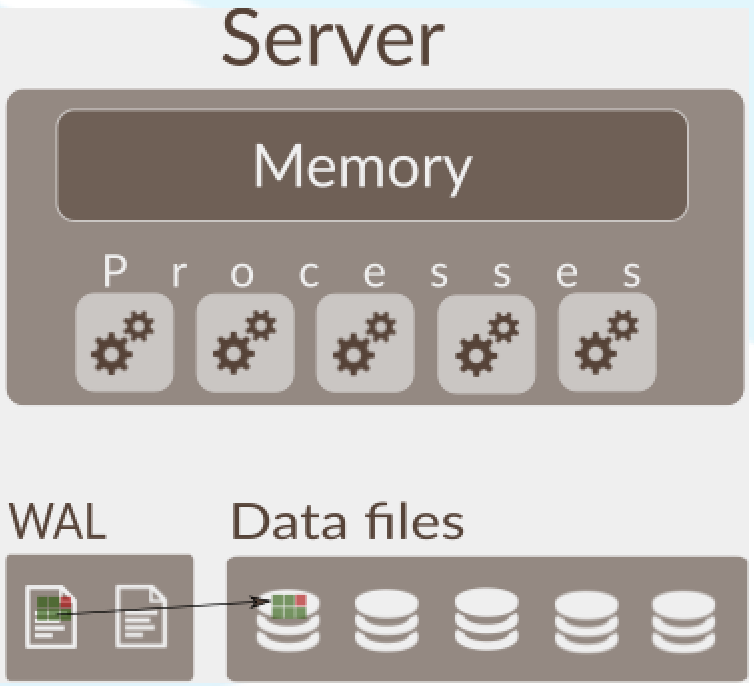


LOXODATA




Server
Memory
DIRTY
Processes
User
WAL
Data files

Checkpoint

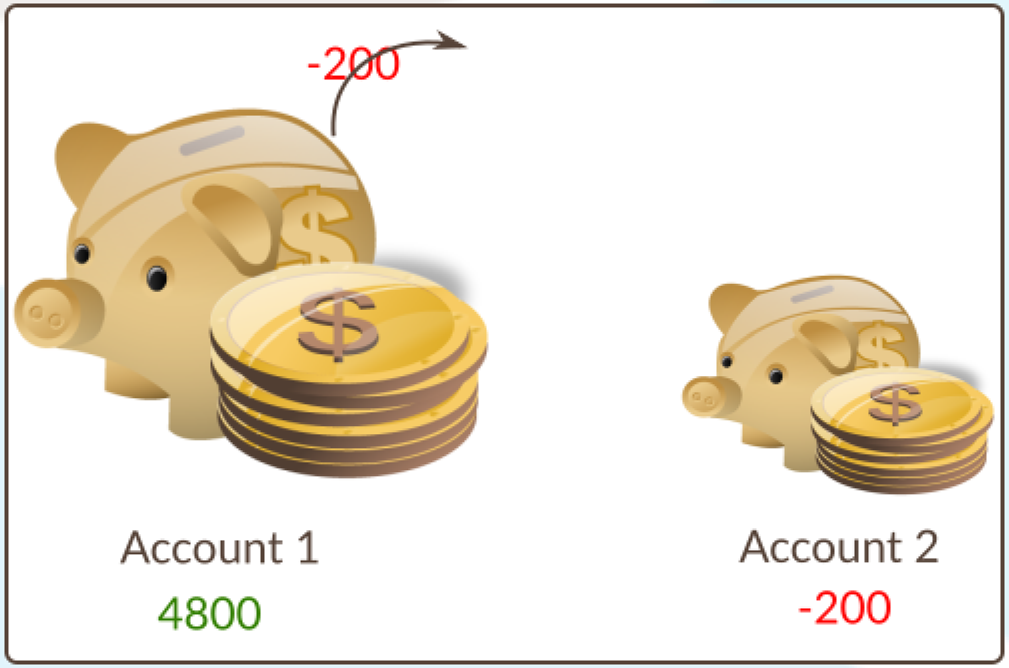


Transaction isolation level

- Read uncommited (not implemented in PostgreSQL)
- Read commited (default)
- Repeatable read
- Serializable

The problem with concurrent transactions

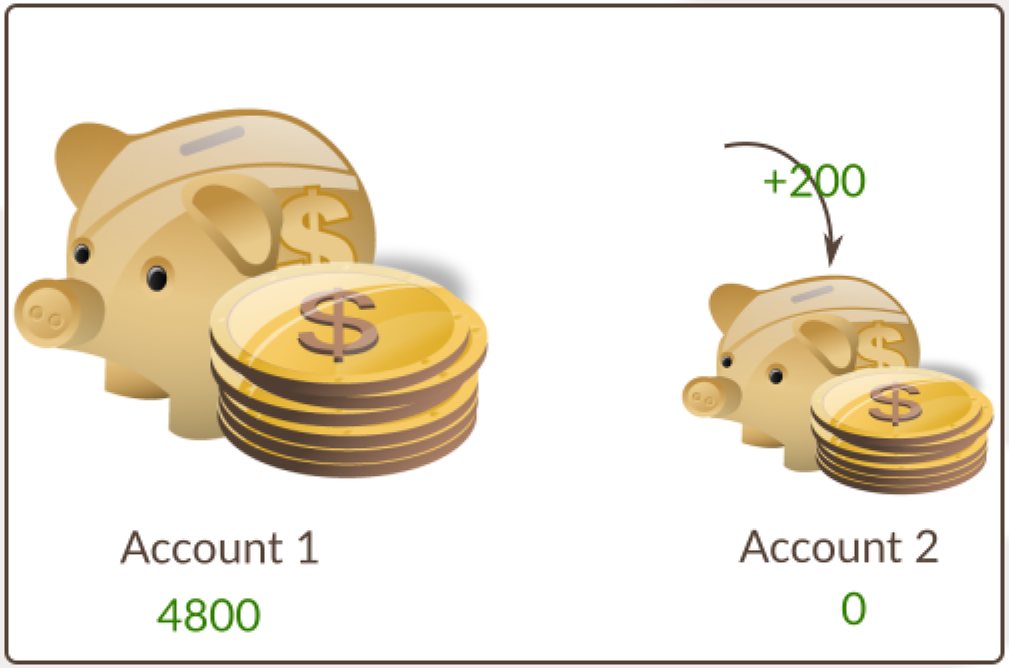
```
SELECT balance FROM account WHERE name = "account 1";
>5000
SELECT balance FROM account WHERE name = "account 2";
>-200
```



The problem with concurrent transactions

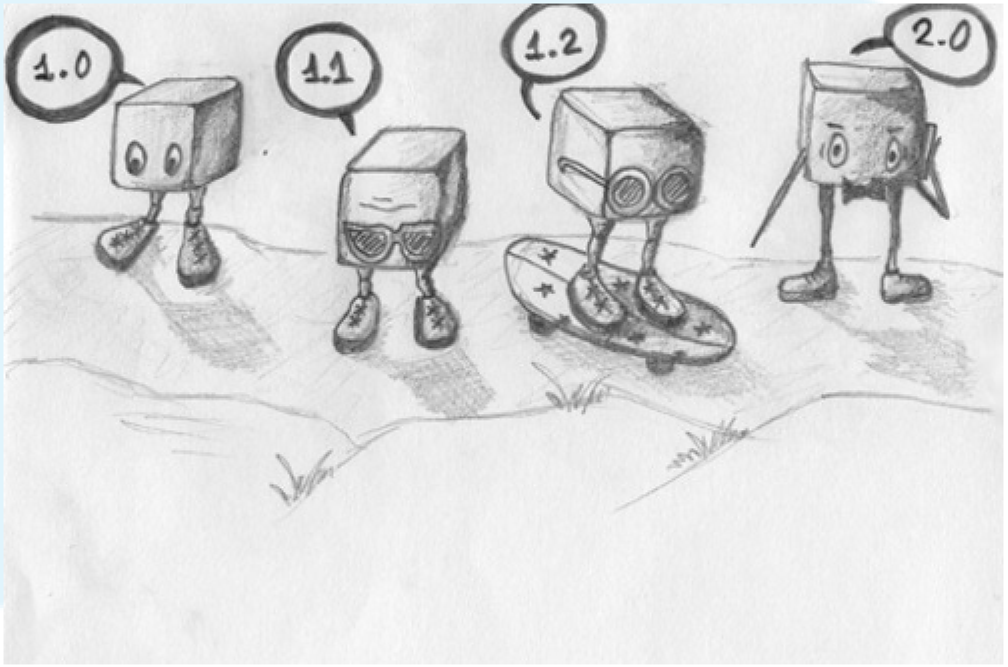
```
SELECT balance FROM account WHERE name = "account 1";
>4800
SELECT balance FROM account WHERE name = "account 2";
>-200
```

LOXODATA



The problem with concurrent transactions

```
SELECT balance FROM account WHERE name = "account 1";
>4800
SELECT balance FROM account WHERE name = "account 2";
>0
```



MVCC

- MultiVersion
- Concurrency
- Control

MVCC

xmin	xmax	id	name
100		1	Anatasia
101		2	Betty
102		3	Chris
157		4	Daniel

New transaction (Transaction_id = 157)

```
INSERT INTO employees (id, name) VALUES (4, 'Daniel');
```

```
>INSERT 1
```

LOXODATA

MVCC

xmin	xmax	id	name
100		1	Anatasia
101		2	Betty
102		3	Chris
157		4	Daniel

New transaction (Transaction_id = 158)

```
DELETE FROM employees WHERE name ="Betty";
```

LOXODATA

MVCC

xmin	xmax	id	name
100		1	Anatasia
101	158	2	Betty
102		3	Chris
157		4	Daniel

New transaction (Transaction_id = 158)

```
DELETE FROM employees WHERE name ="Betty";
```

```
>DELETE 1
```

MVCC

xmin	xmax	id	name
100		1	Anatasia
101	158	2	Betty
102		3	Chris
157		4	Daniel

New transaction (Transaction_id = 159)

```
UPDATE employees SET name = "Christian" WHERE name ="Chris";
```

LOXODATA

MVCC

xmin	xmax	id	name
100		1	Anatasia
101	158	2	Betty
102	159	3	Chris
157		4	Daniel
159		3	Christian

New transaction (Transaction_id = 159)

```
UPDATE employees SET name = "Christian" WHERE name ="Chris";
```

```
>UPDATE 1
```

Cleaning "old" rows

- VACUUM command
- autovacuum deamon

LOXODATA

SETTINGS

LOXODATA

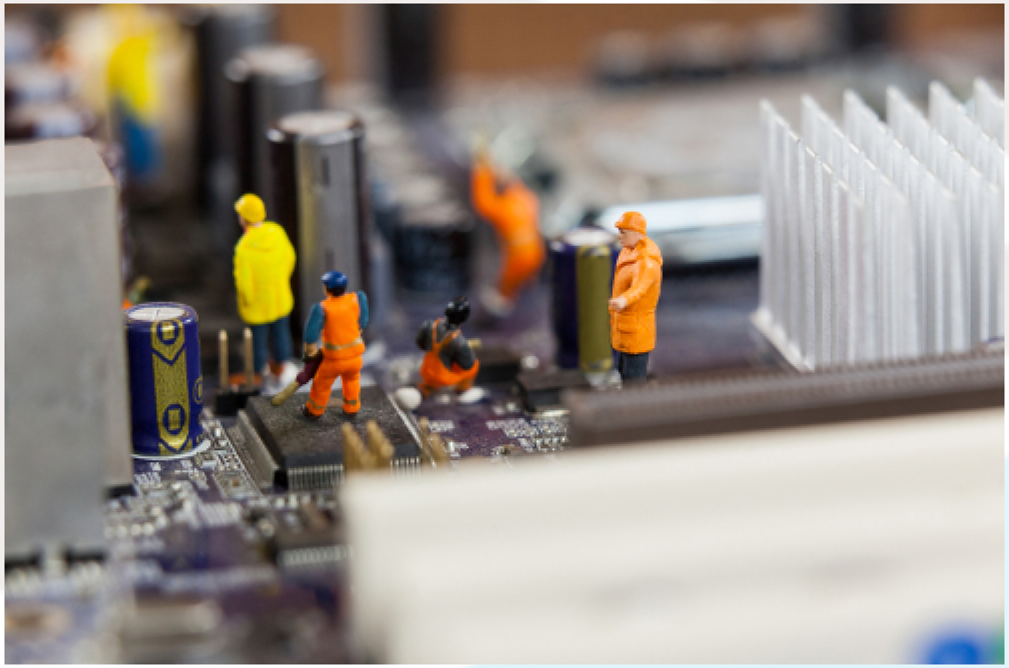
Shared_buffers

- Used for caching data
- Not too low, not too high
- 1/4 of your system's RAM is good (start <8Gb)

Effective_cache_size

- A hint for the planner to know how much cache it can use
- It's an estimate, no need to be very precise
- amount of cache or 75% of RAM

LOXODATA



Work_mem

- Used for sorting data and hash join
- Queries knowledge mandatory

LOXODATA

Maintenance_work_mem

- Used for maintenance operations
- Start with 1GB

Autovacuum_work_mem & autovacuum_max_worker

- Used for autovacuum daemon operations
- If you don't know, keep default values
  (i.e. maintenance_work_mem)

Tuning autovacuum (and autoanalyze)

- autovacuum_naptime
- autovacuum_vacuum·analyze_scale_factor
- autovacuum_vacuum·analyze_threshold

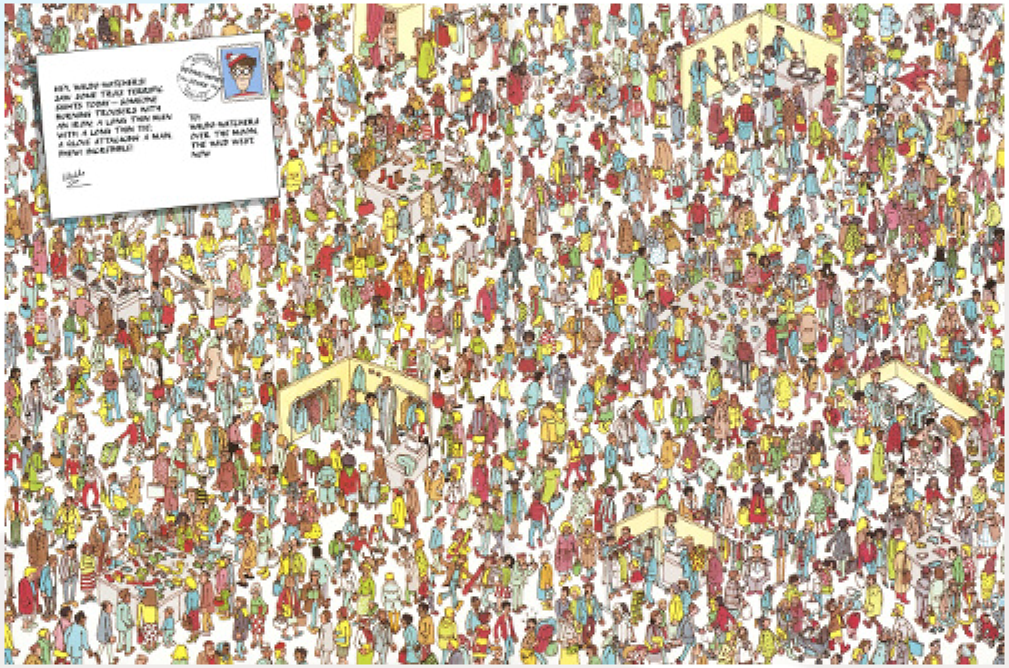
LOXODATA

Checkpoints

- checkpoint_timeout
- checkpoint_completion_target
- checkpoint_flush_after
- max·min_wal_size

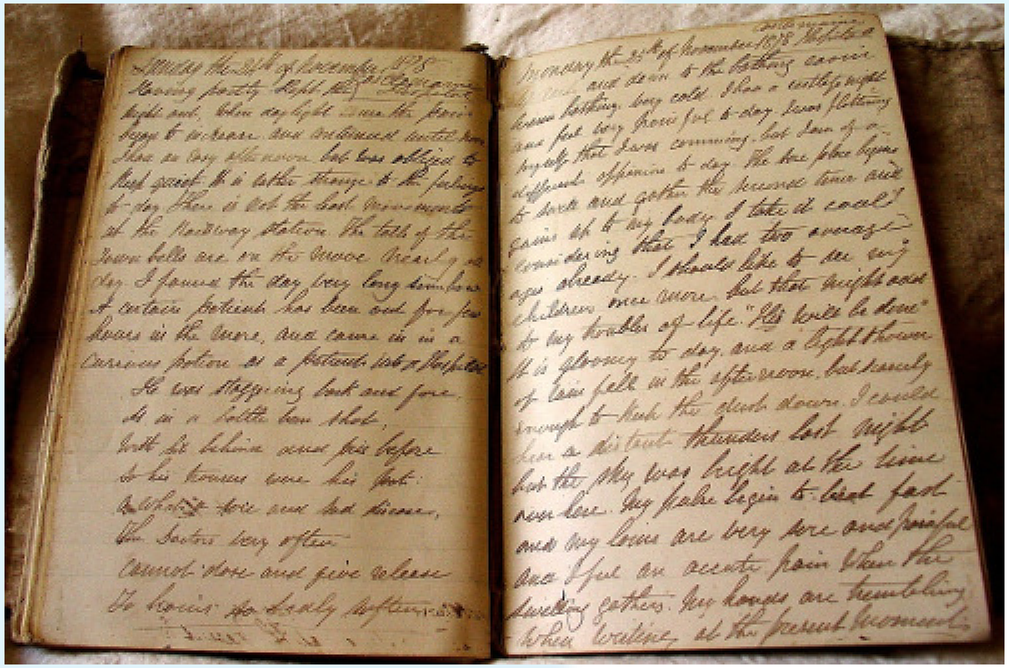
LOGGING

LOXODATA



Logging location

- logging_collector
- log_destination
- log_directory
- log_filename
- log_rotation_age·size



What to log

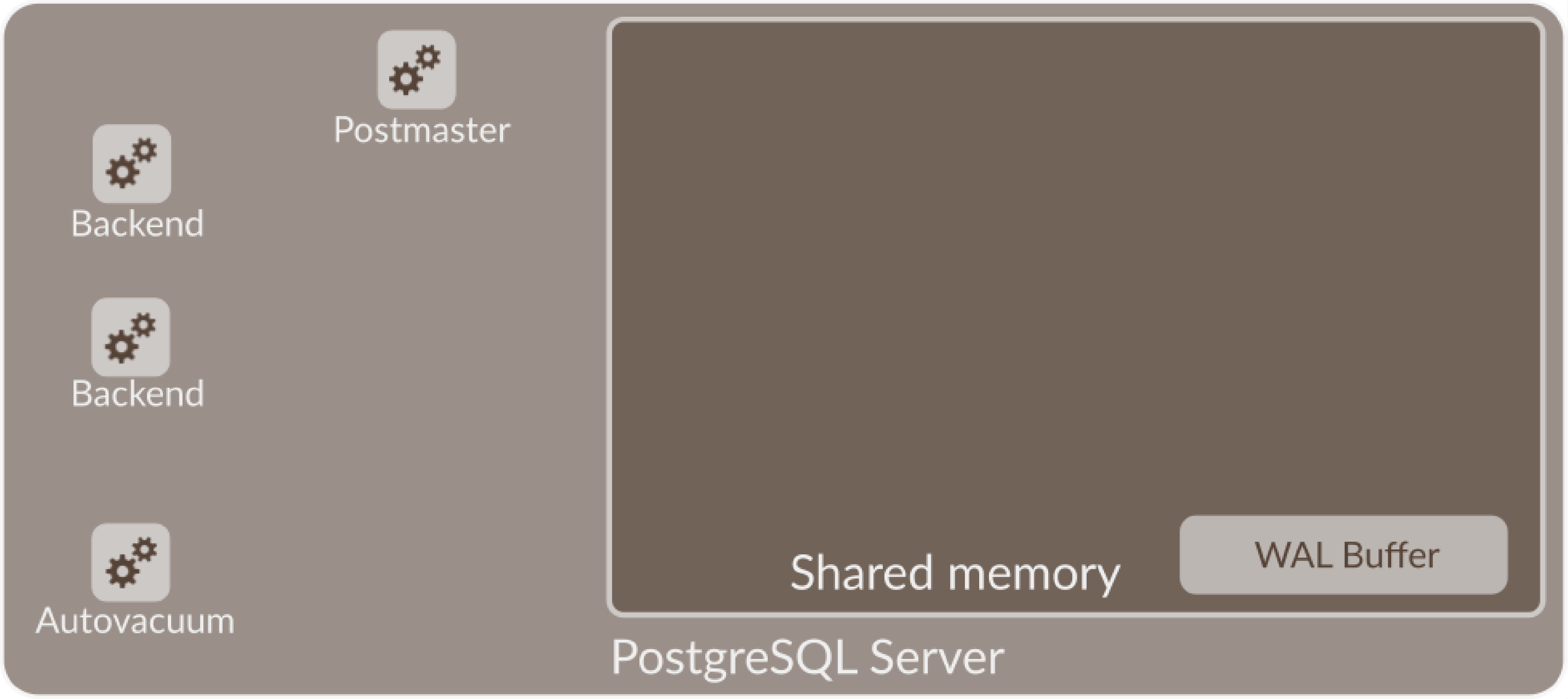
- log_min_duration_statement
- log_checkpoints = on
- log_lock_waits = on
- log_temp_files = 0

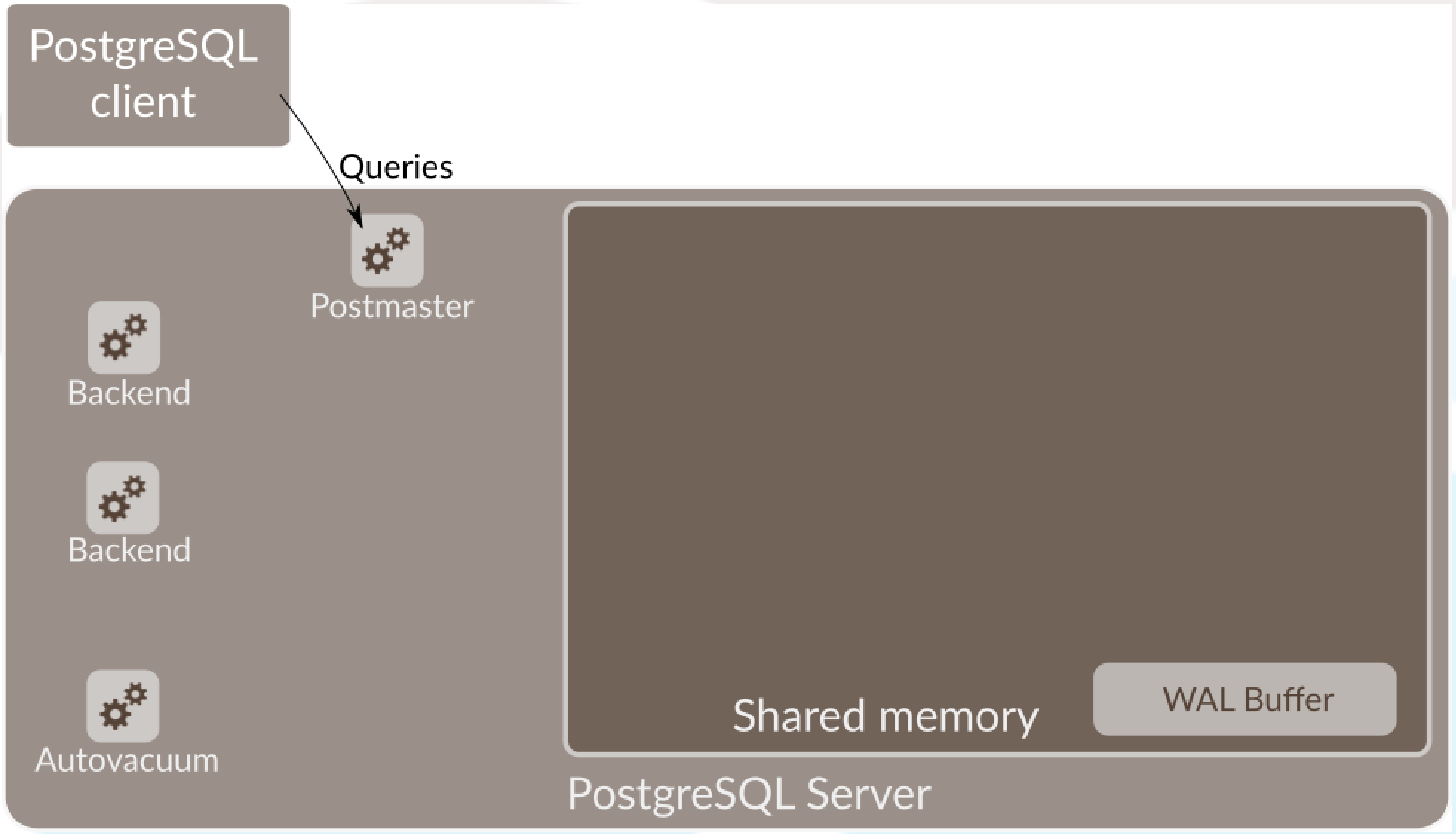
LOXODATA

What to log

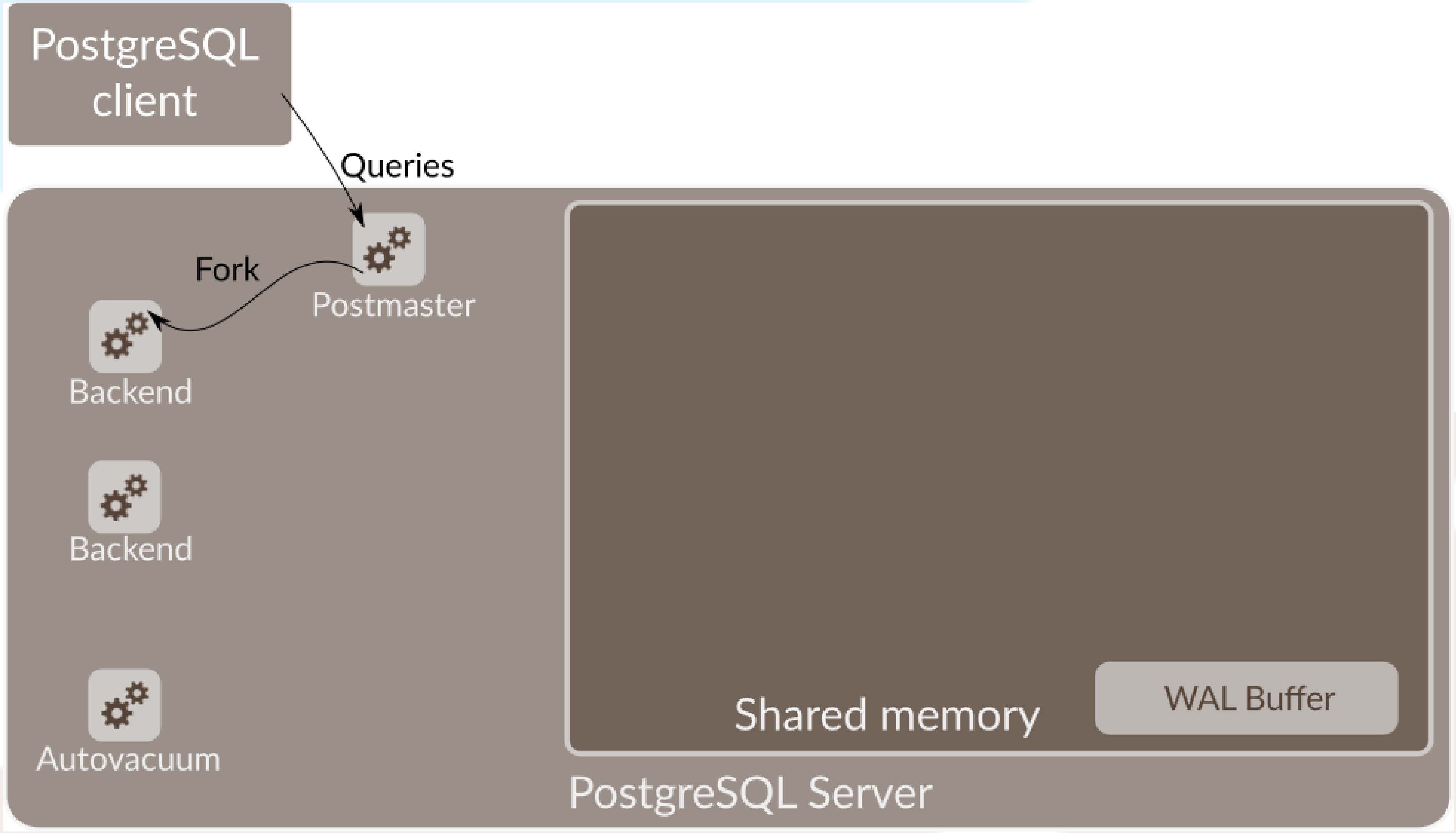
- log_line_prefix = '%t [%p-%l] %u@%d '
- log_autovacuum_min_duration = 0

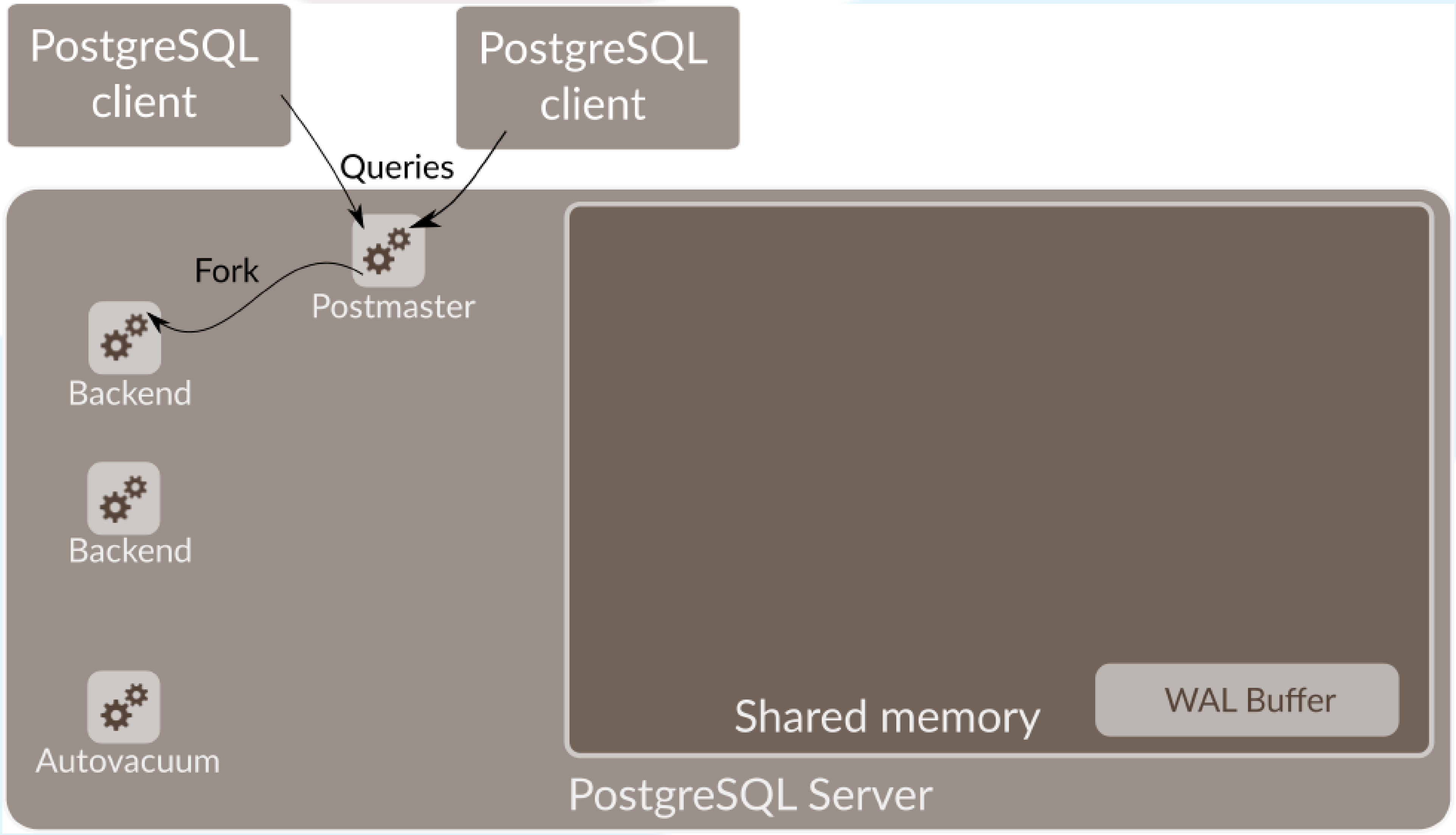
QUERIES


Postmaster
Backend
Backend
Autovacuum
Shared memory
WAL Buffer
PostgreSQL Server

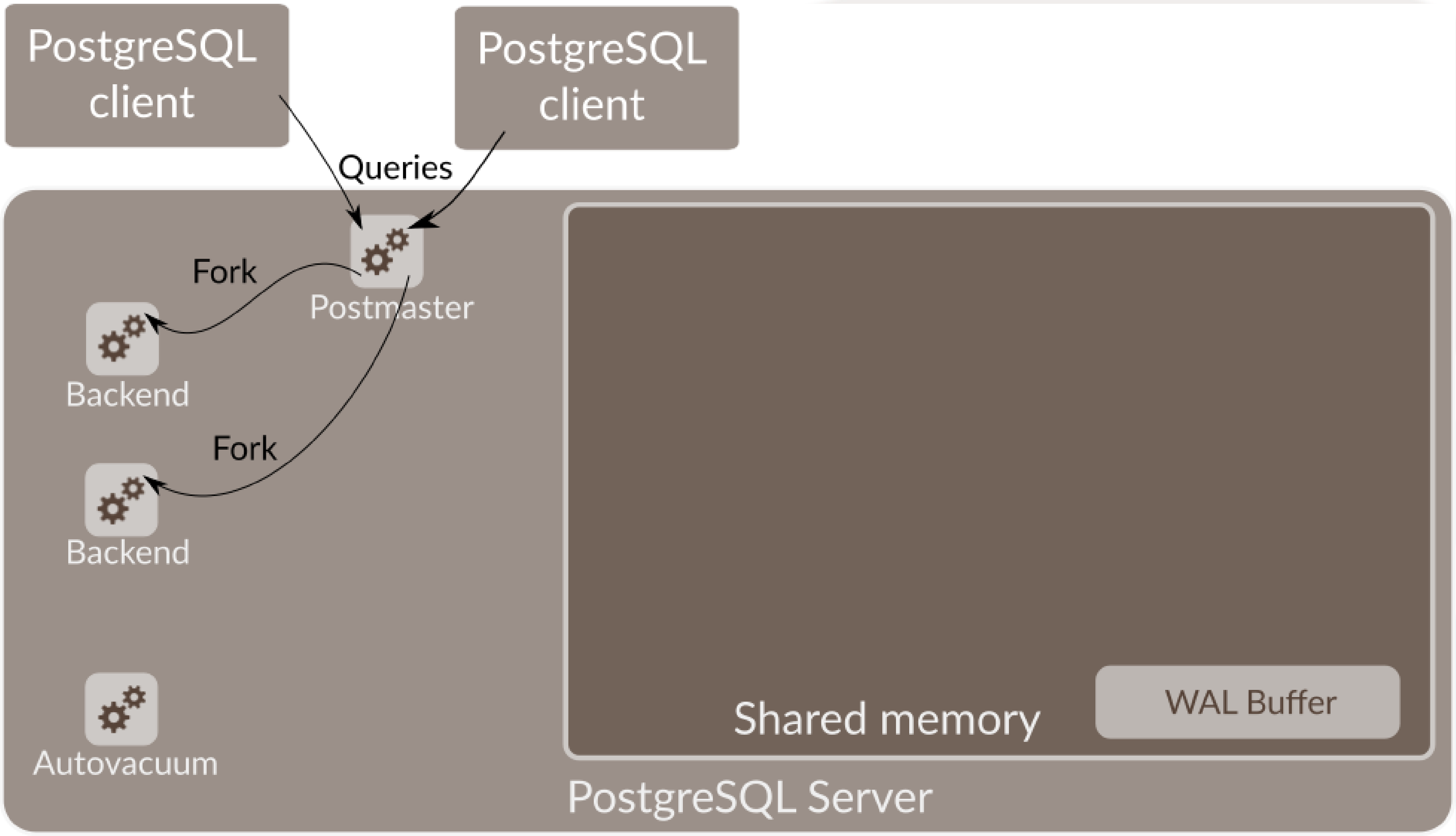

PostgreSQL client
Queries
Postmaster
Backend
Backend
Autovacuum
Shared memory
WAL Buffer
PostgreSQL Server

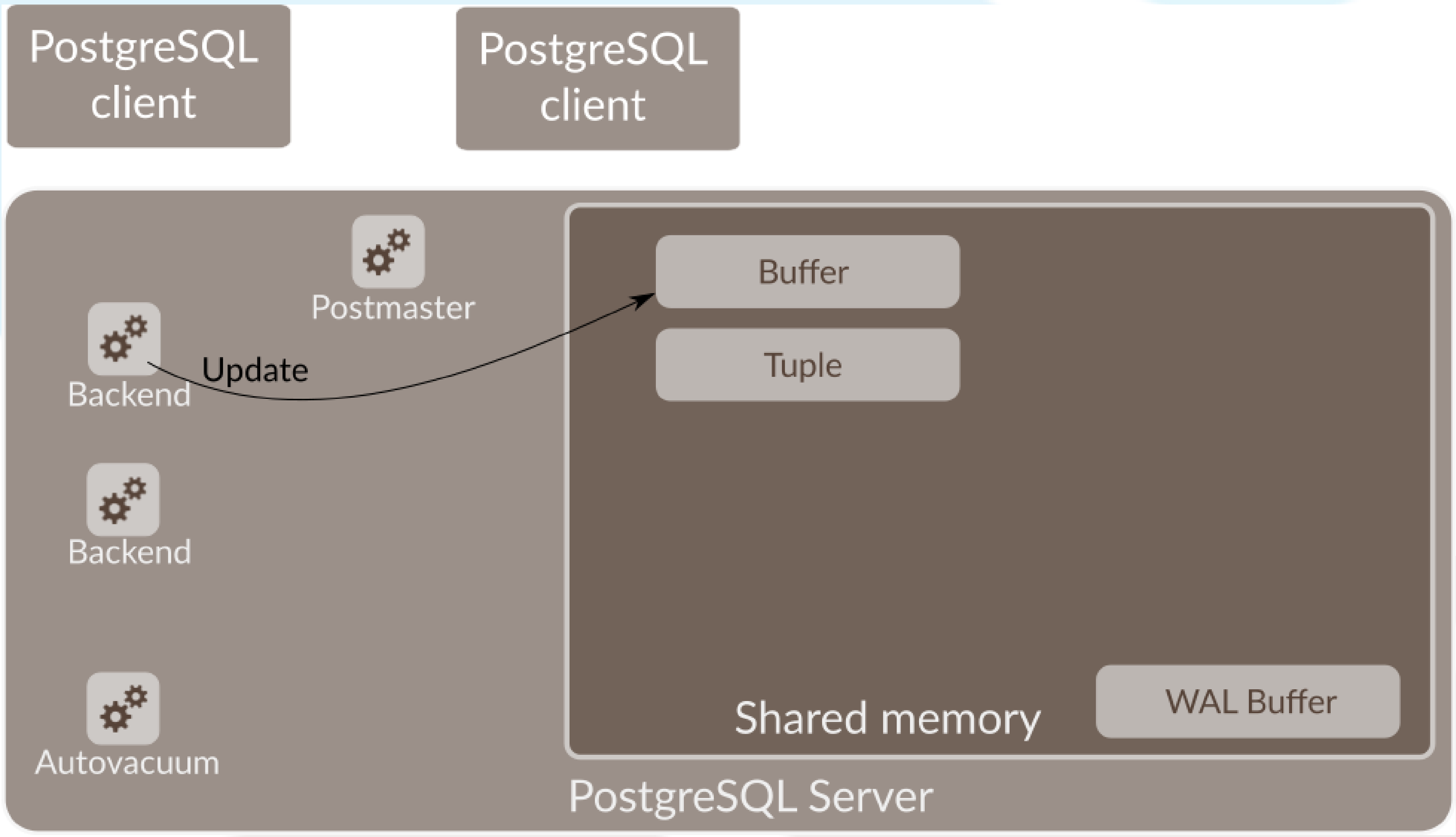
LOXODATA

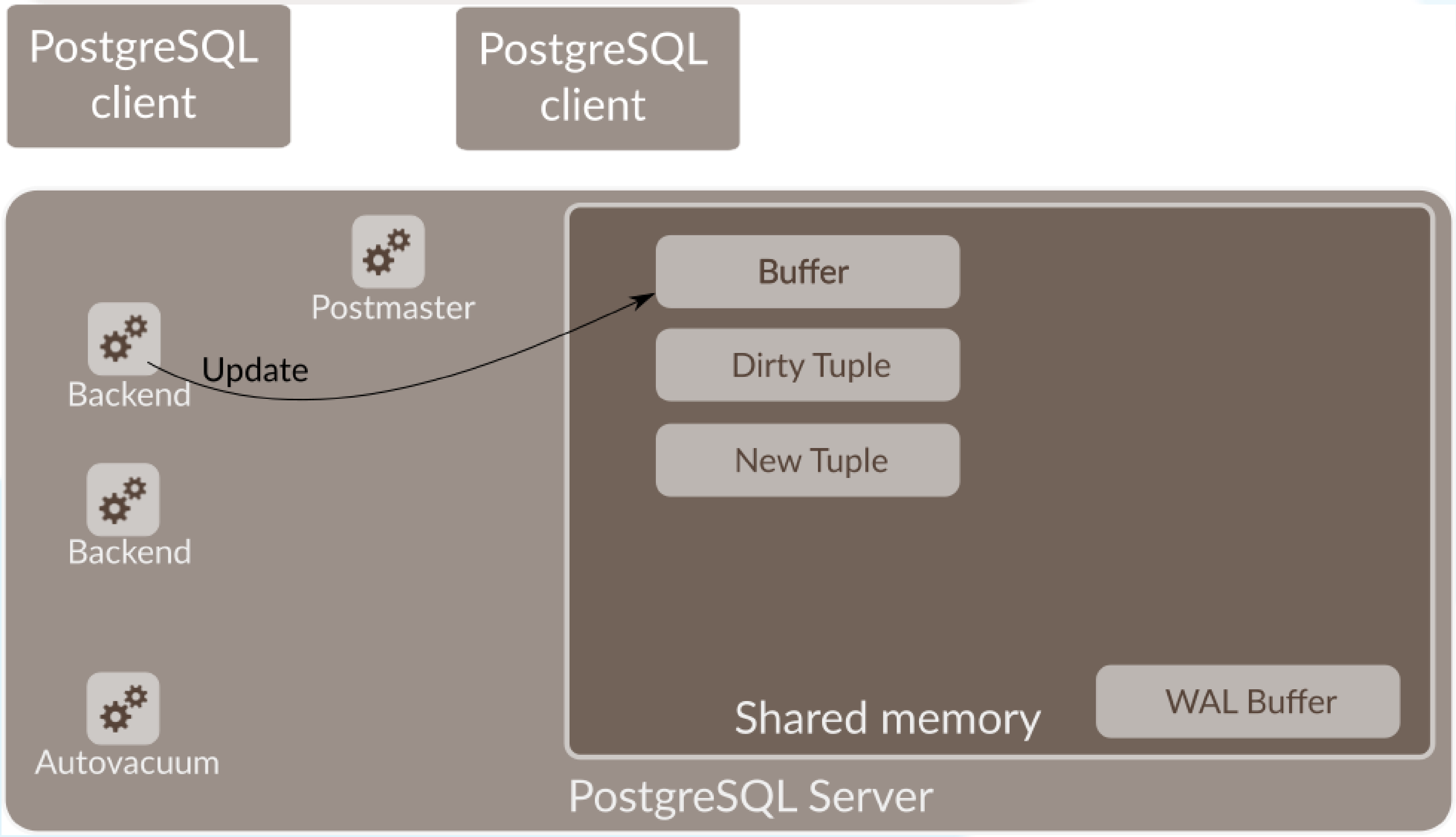

PostgreSQL client
Queries
Fork
Postmaster
Backend
Backend
Autovacuum
Shared memory
WAL Buffer
PostgreSQL Server

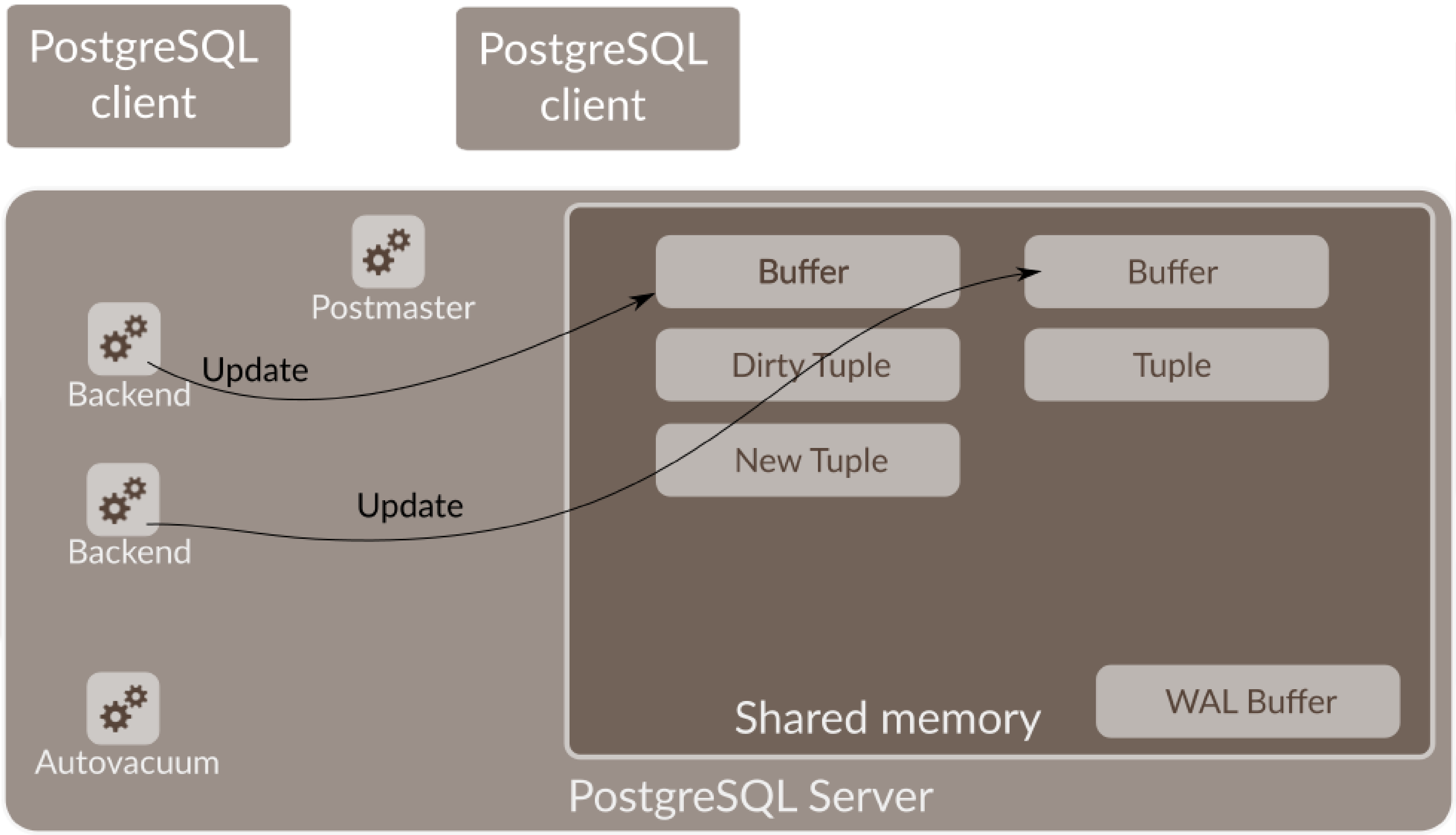

PostgreSQL client
PostgreSQL client
Queries
Fork
Postmaster
Backend
Backend
Autovacuum
Shared memory
WAL Buffer
PostgreSQL Server

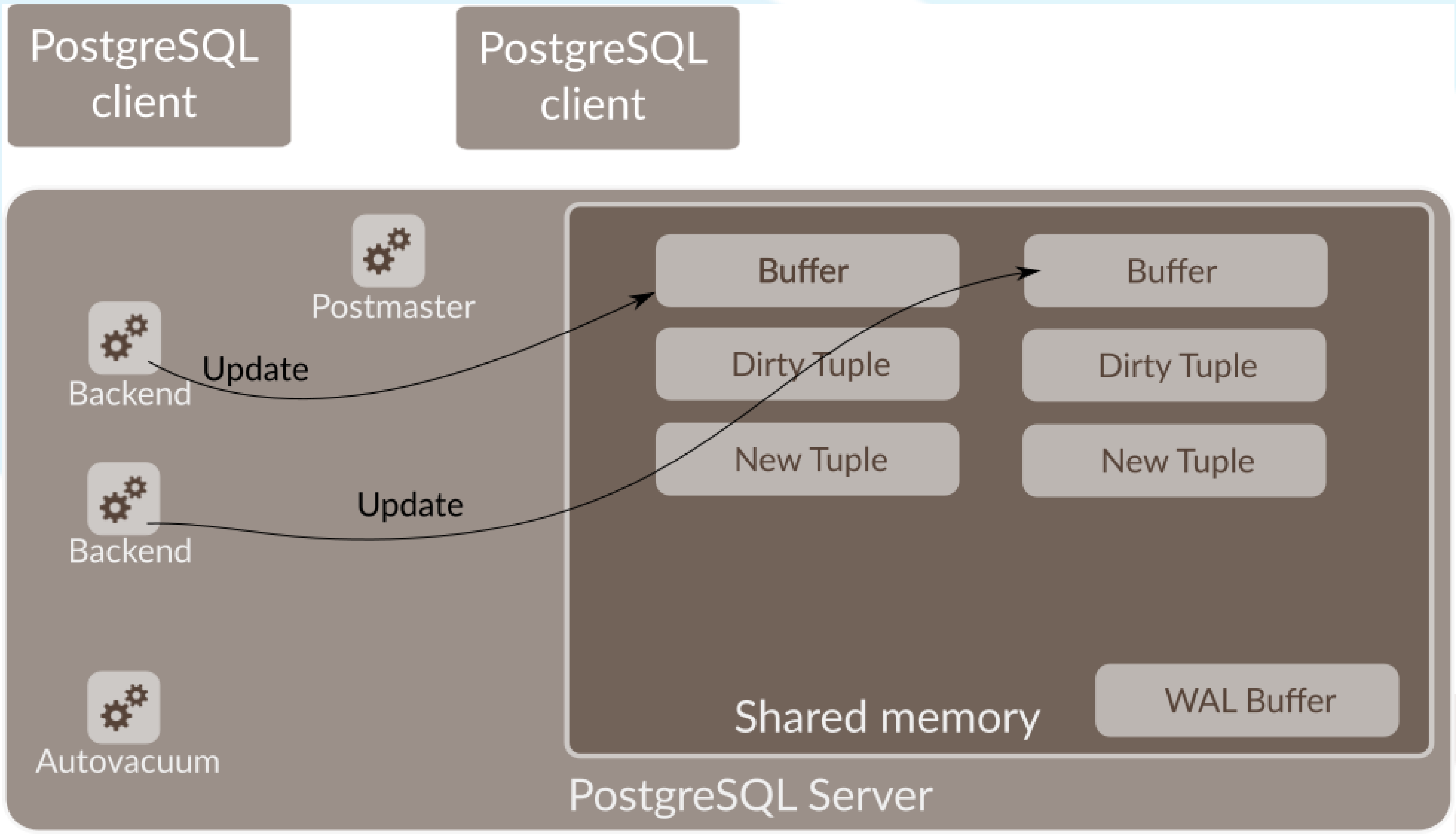
LOXODATA


PostgreSQL client
PostgreSQL client
Queries
Fork
Postmaster
Backend
Fork
Backend
Autovacuum
Shared memory
WAL Buffer
PostgreSQL Server

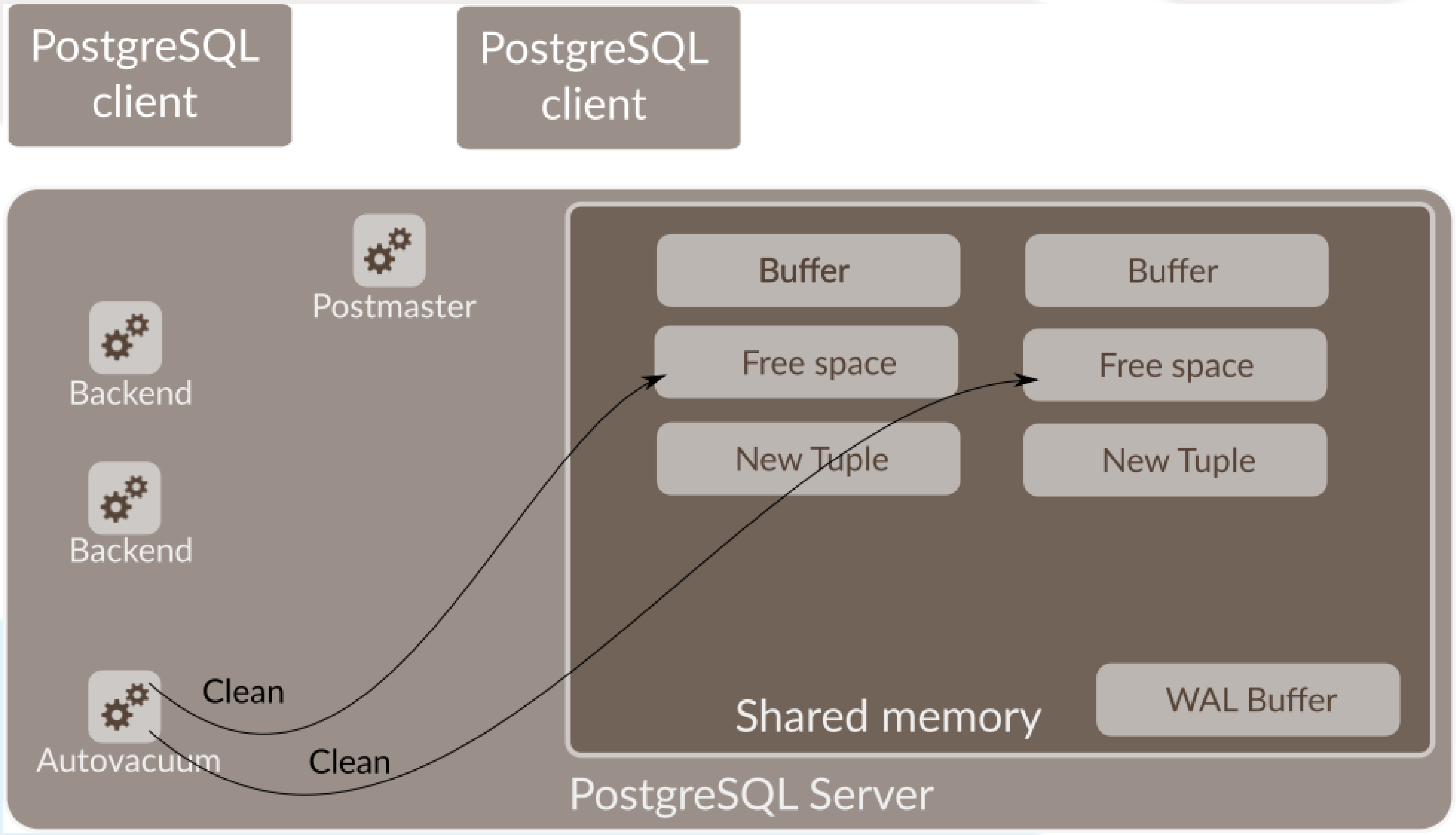

PostgreSQL client
PostgreSQL client
Postmaster
Backend
Update
Backend
Autovacuum
Buffer
Tuple
Shared memory
WAL Buffer
PostgreSQL Server


PostgreSQL client
PostgreSQL client
Postmaster
Backend
Update
Backend
Autovacuum
Buffer
Dirty Tuple
New Tuple
Shared memory
WAL Buffer
PostgreSQL Server

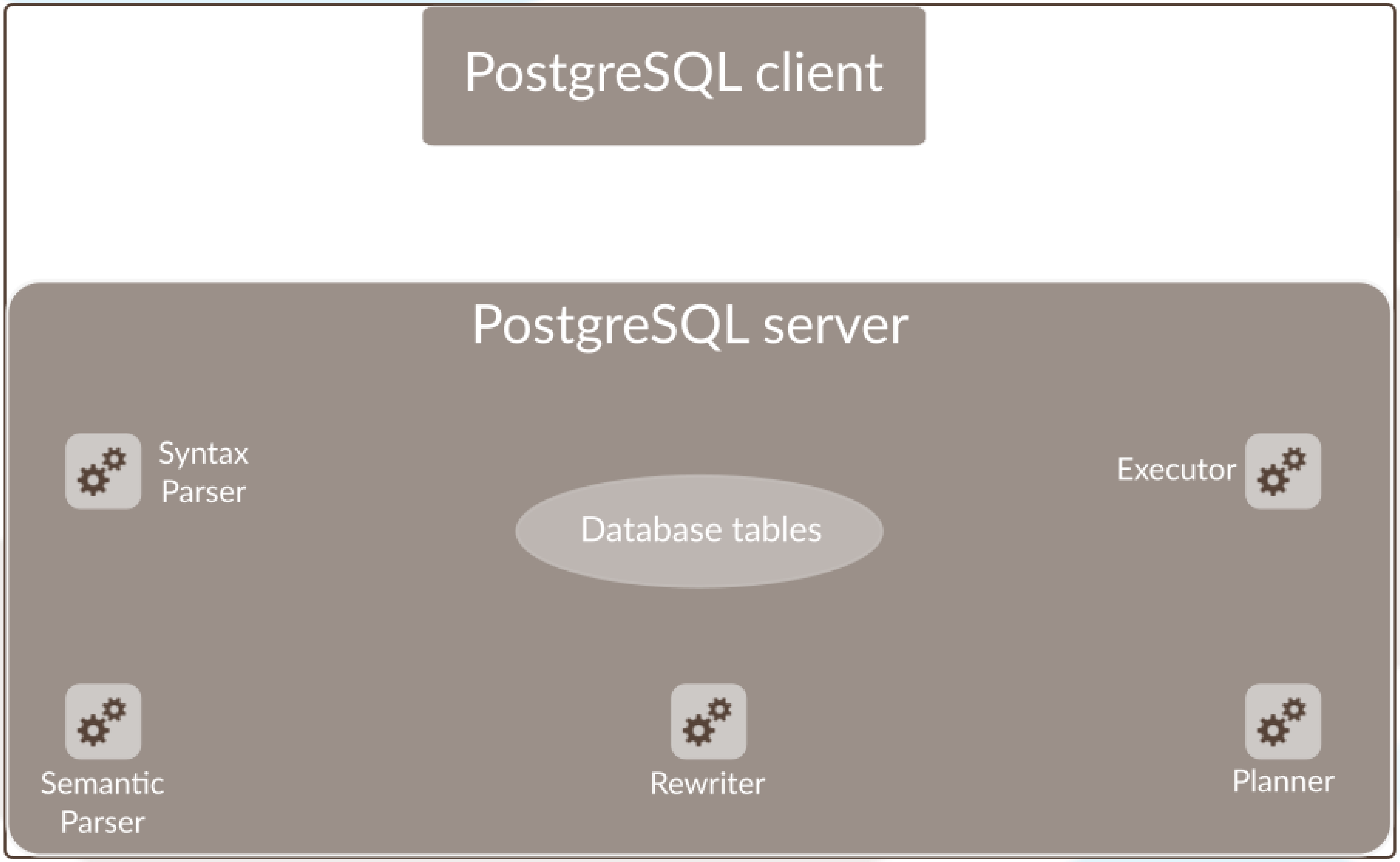

PostgreSQL client
PostgreSQL client
Postmaster
Backend
Update
Backend
Update
Autovacuum
Buffer
Dirty Tuple
New Tuple
Buffer
Tuple
Shared memory
WAL Buffer
PostgreSQL Server

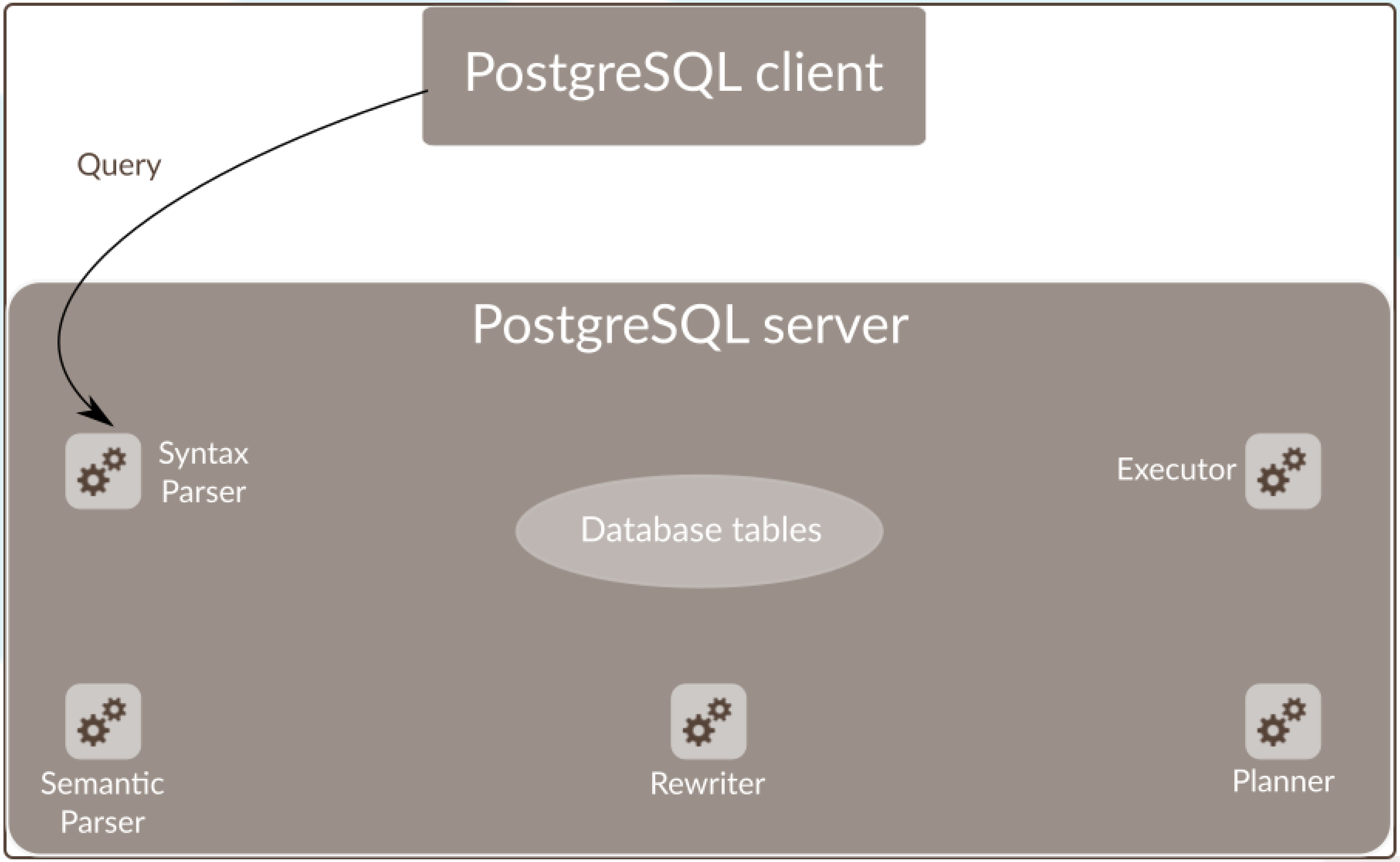

PostgreSQL client
PostgreSQL client
Postmaster
Backend
Update
Backend
Update
Autovacuum
Buffer
Dirty Tuple
New Tuple
Buffer
Dirty Tuple
New Tuple
Shared memory
WAL Buffer
PostgreSQL Server

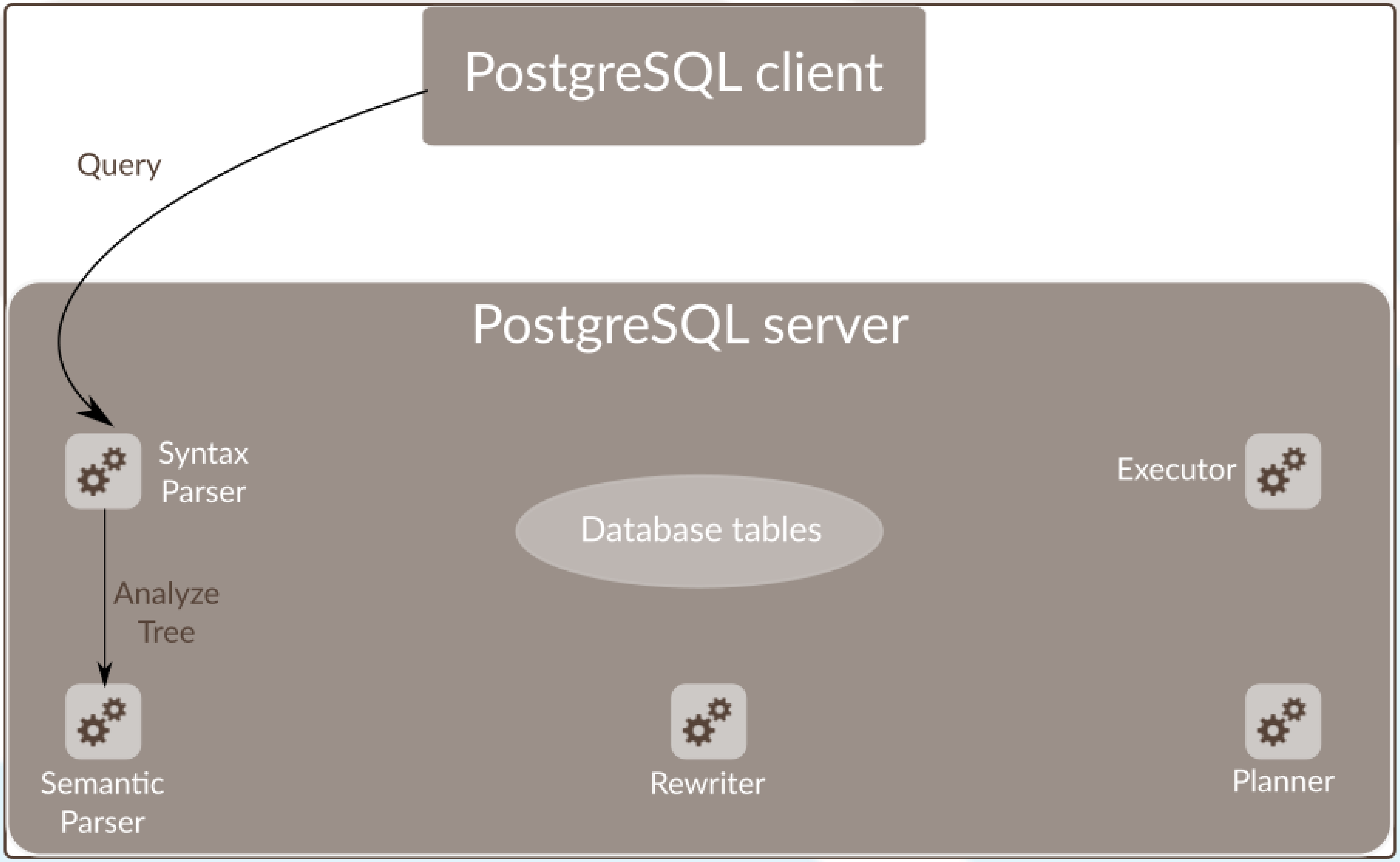
LOXODATA

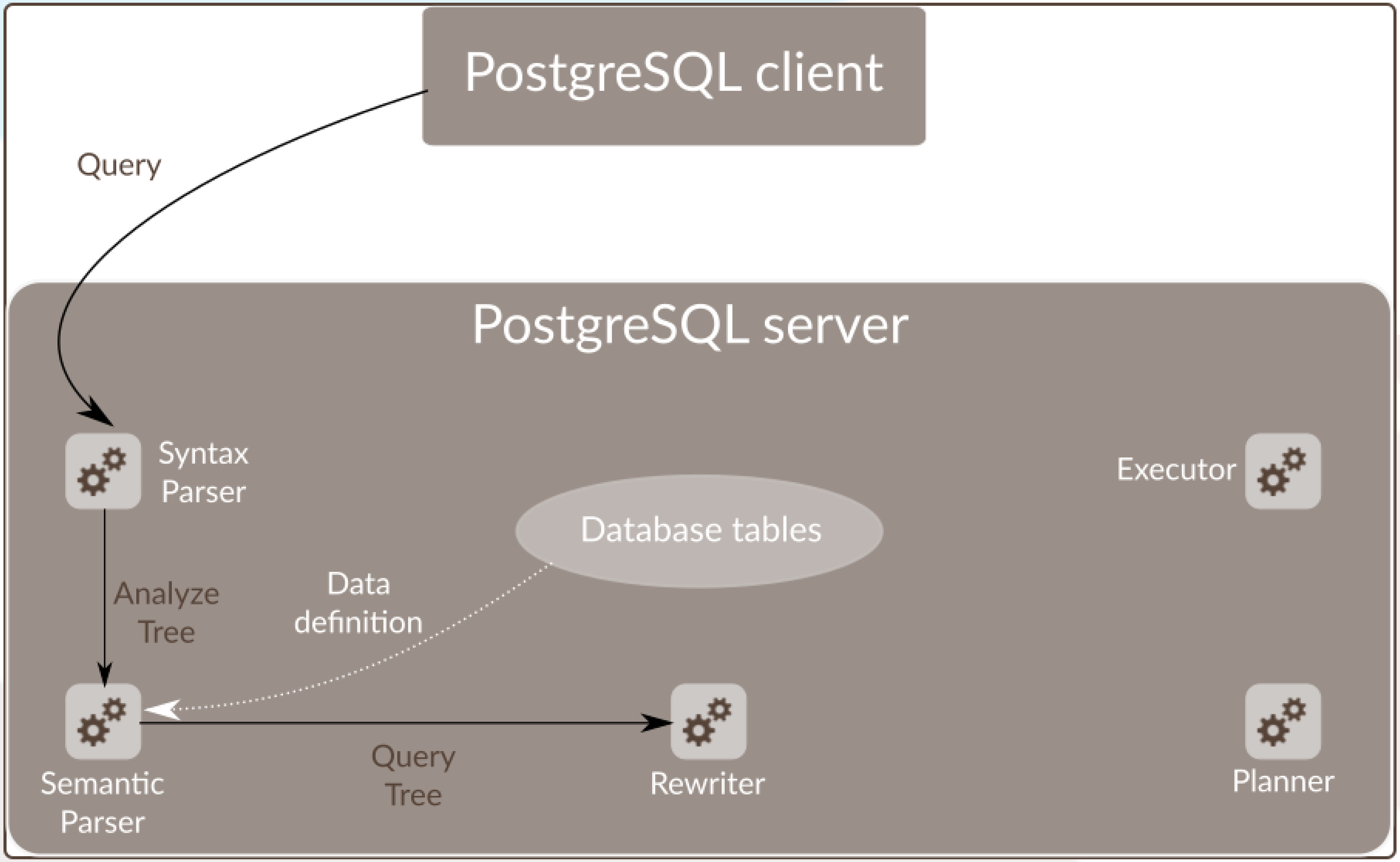

PostgreSQL client
PostgreSQL client
Postmaster
Backend
Backend
Autovacuum
Clean
Clean
Buffer
Buffer
Free space
Free space
New Tuple
New Tuple
WAL Buffer
Shared memory
PostgreSQL Server

LOXODATA

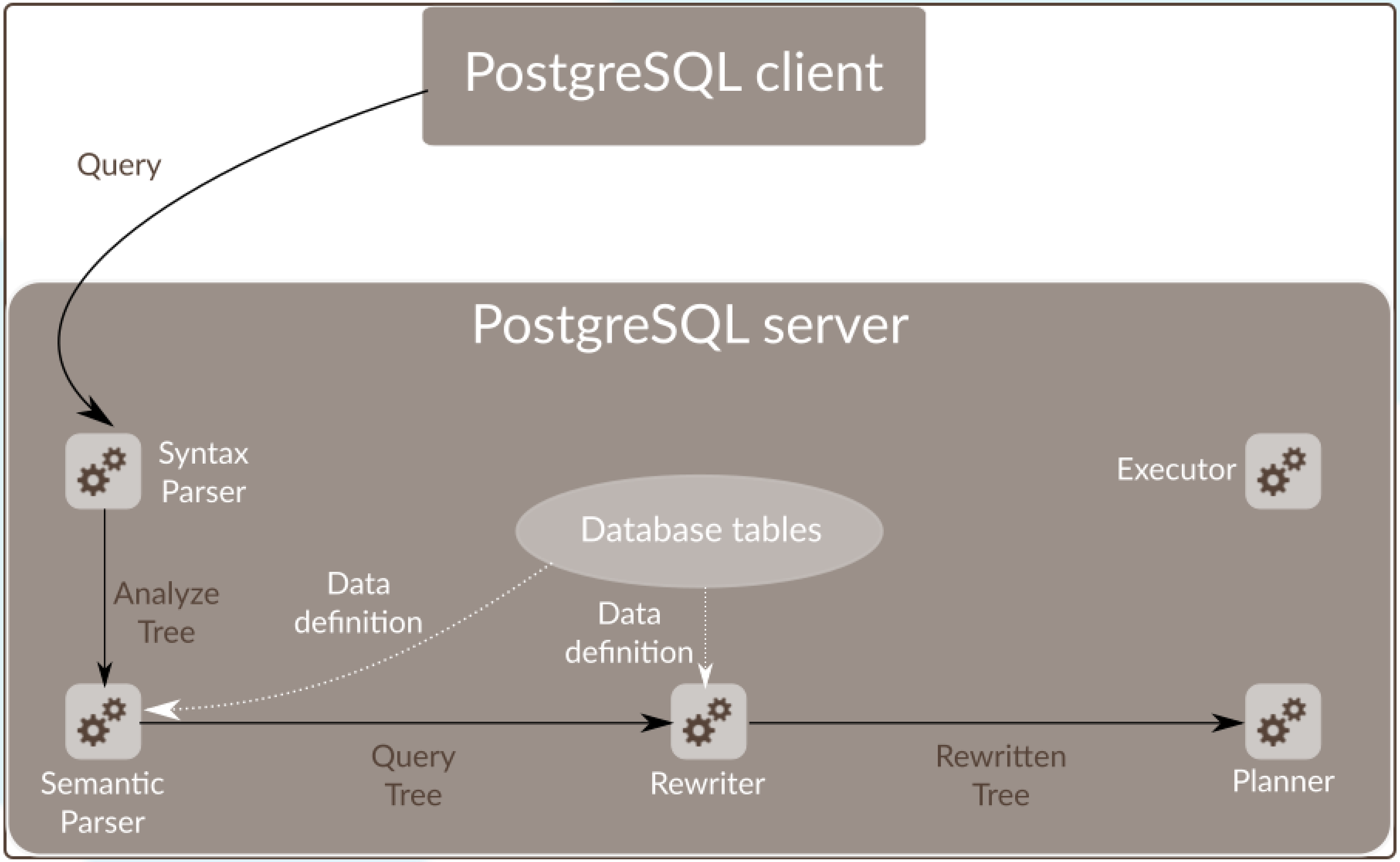

PostgreSQL client
PostgreSQL server
Syntax Parser
Executor
Database tables
Semantic Parser
Rewriter
Planner


PostgreSQL client
Query
PostgreSQL server
Syntax Parser
Database tables
Executor
Semantic Parser
Rewriter
Planner


PostgreSQL client
Query
PostgreSQL server
Syntax Parser
Analyze Tree
Semantic Parser
Database tables
Rewriter
Planner
Executor

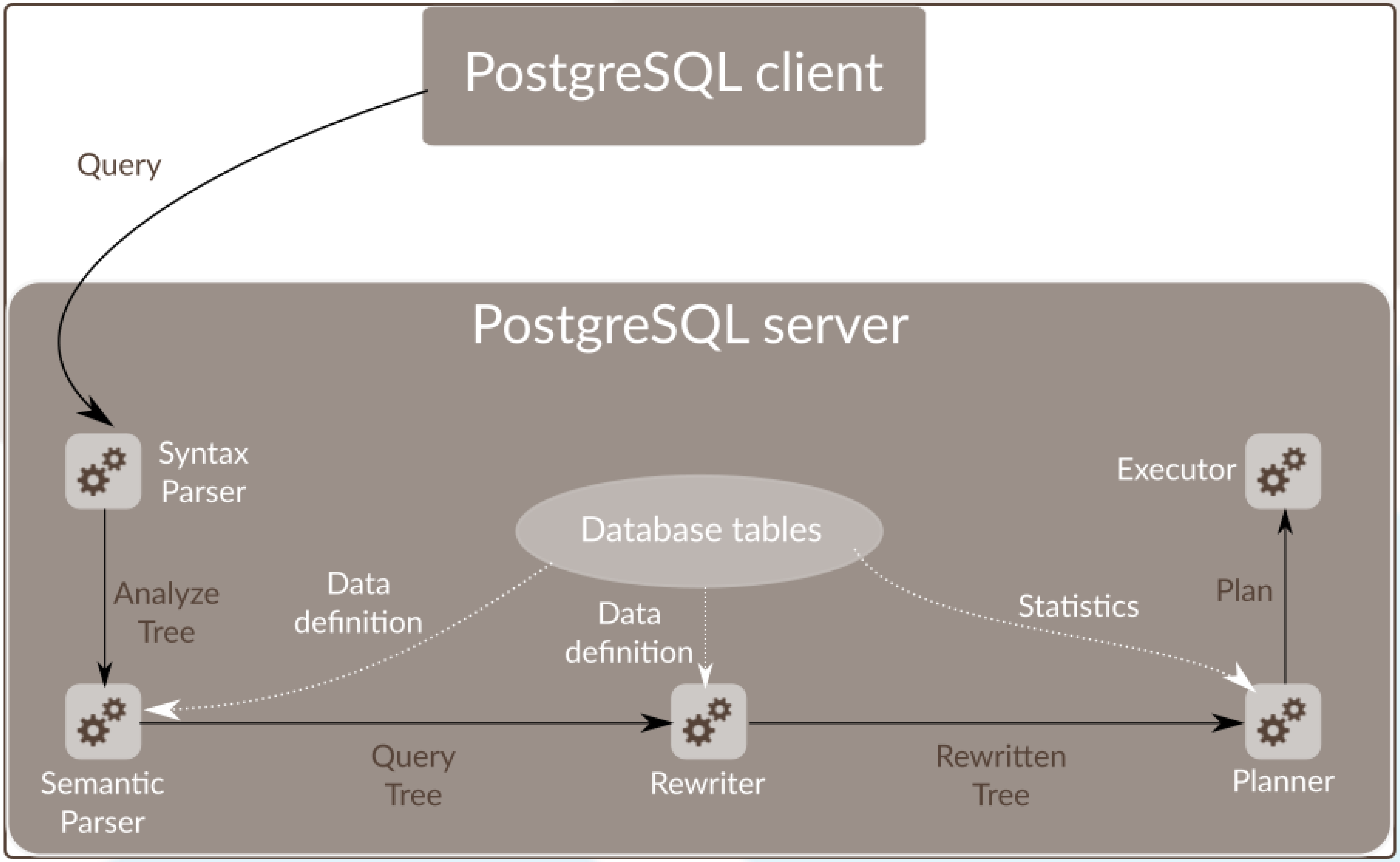

PostgreSQL client
Query
PostgreSQL server
Syntax Parser
Executor
Database tables
Analyze Tree
Data definition
Semantic Parser
Query Tree
Rewriter
Planner

LOXODATA

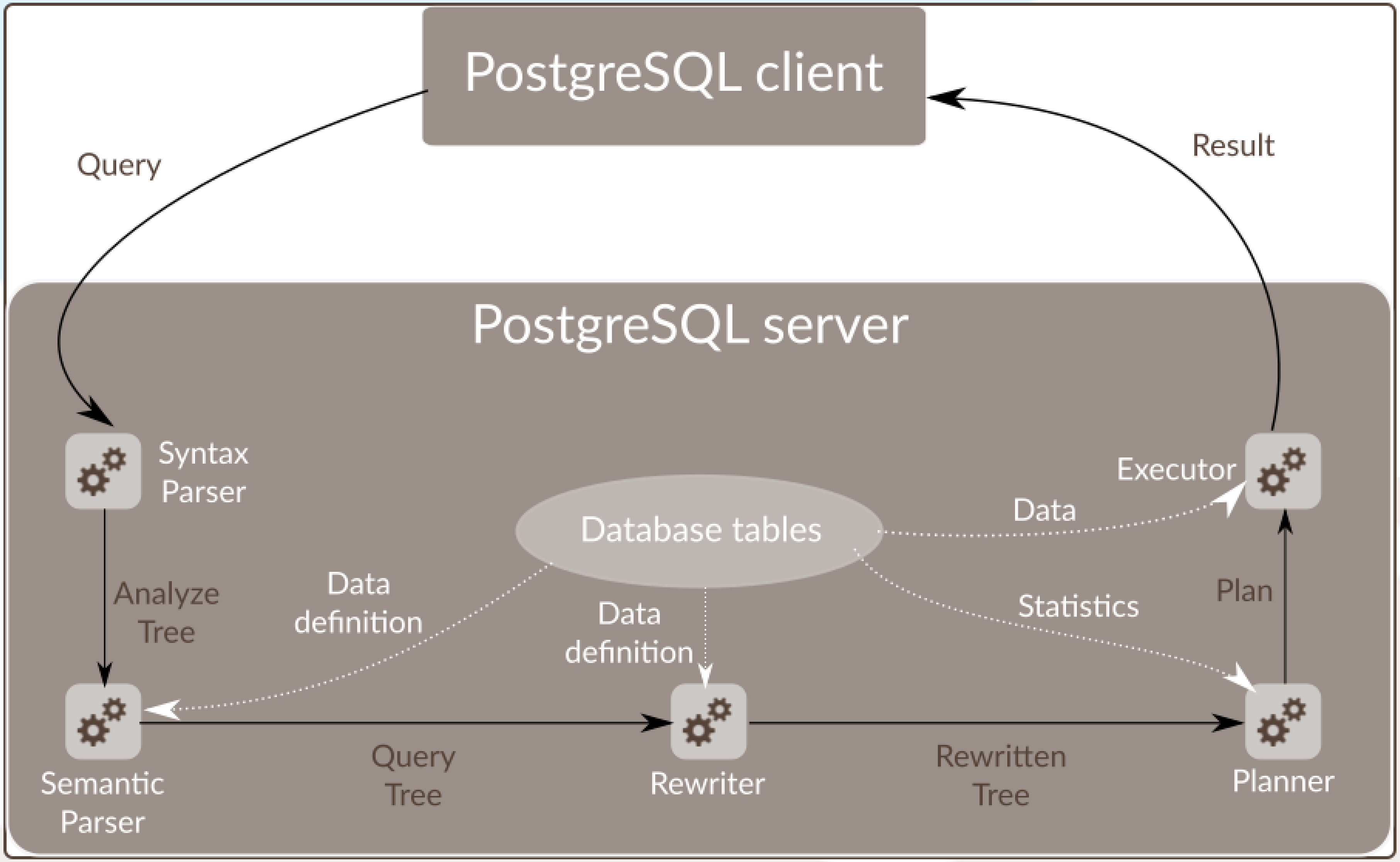


PostgreSQL performance

LOXODATA

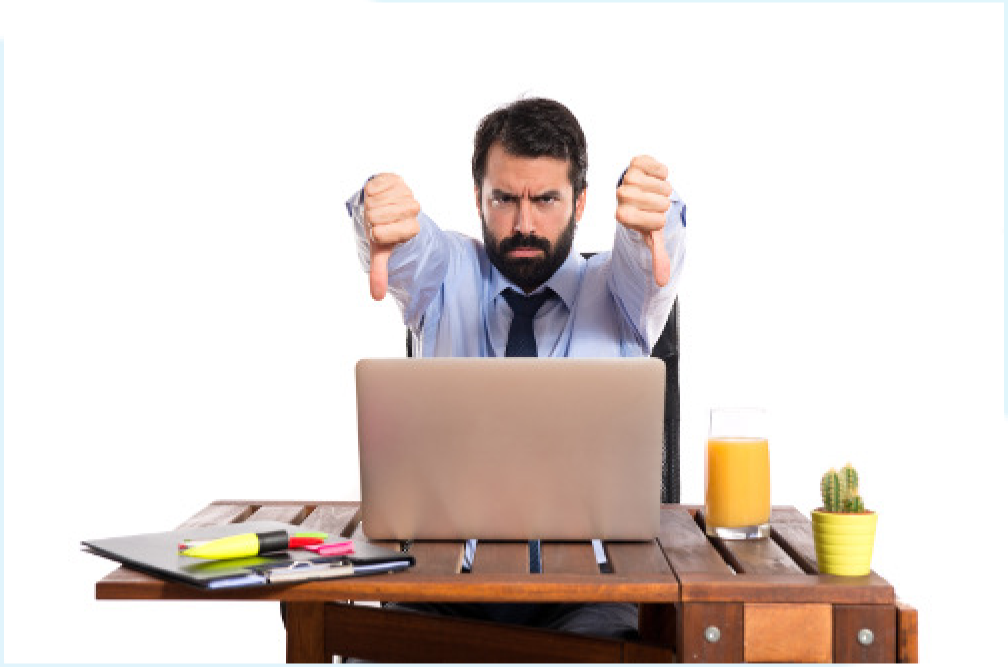

PostgreSQL client
Query
PostgreSQL server
Syntax Parser
Analyze Tree
Data definition
Database tables
Data definition
Statistics
Plan
Executor
Semantic Parser
Query Tree
Rewriter
Rewritten Tree
Planner

LOXODATA


PostgreSQL client
Query
Result
PostgreSQL server
Syntax Parser
Analyze Tree
Executor
Data
Database tables
Data definition
Data definition
Statistics
Plan
Semantic Parser
Query Tree
Rewriter
Rewritten Tree
Planner

LOXODATA

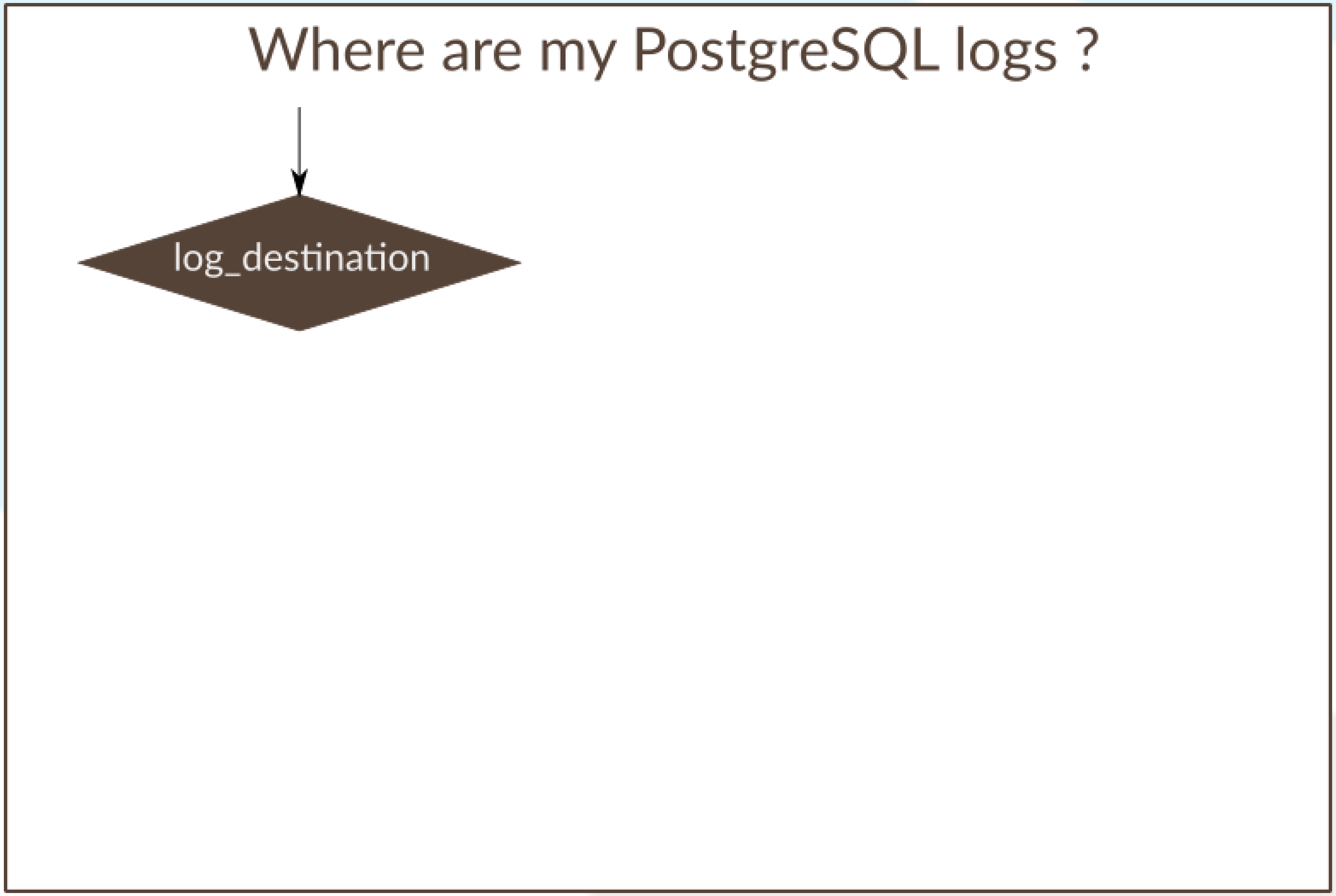
"BAD" QUERIES



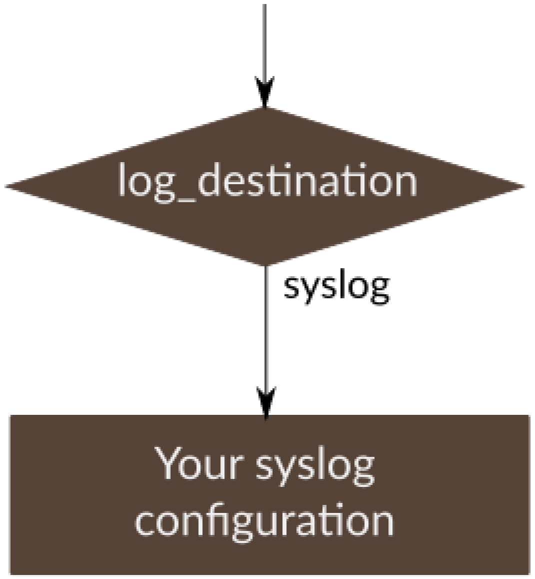
LOXODATA

Settings

- log_min_duration_statement
- log_temp_files = 0

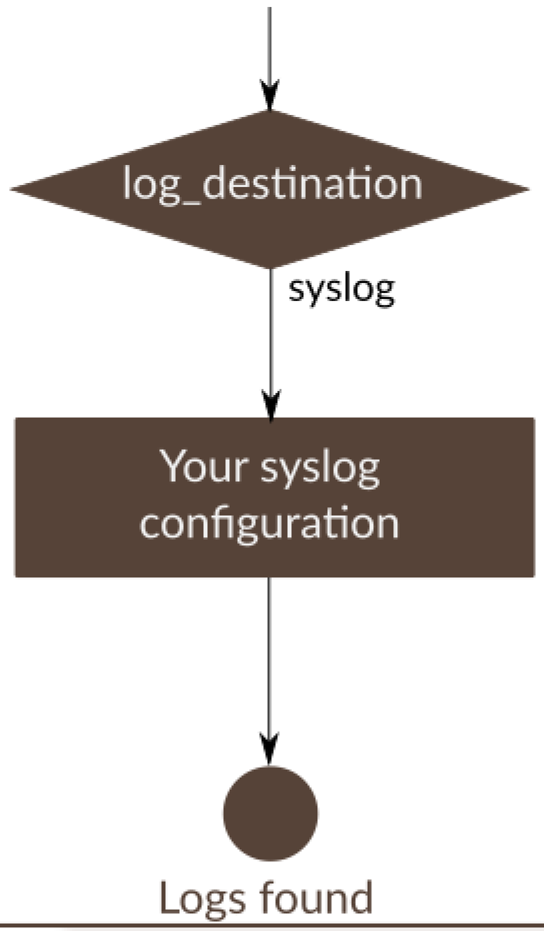

Where are my PostgreSQL logs ?
log_destination

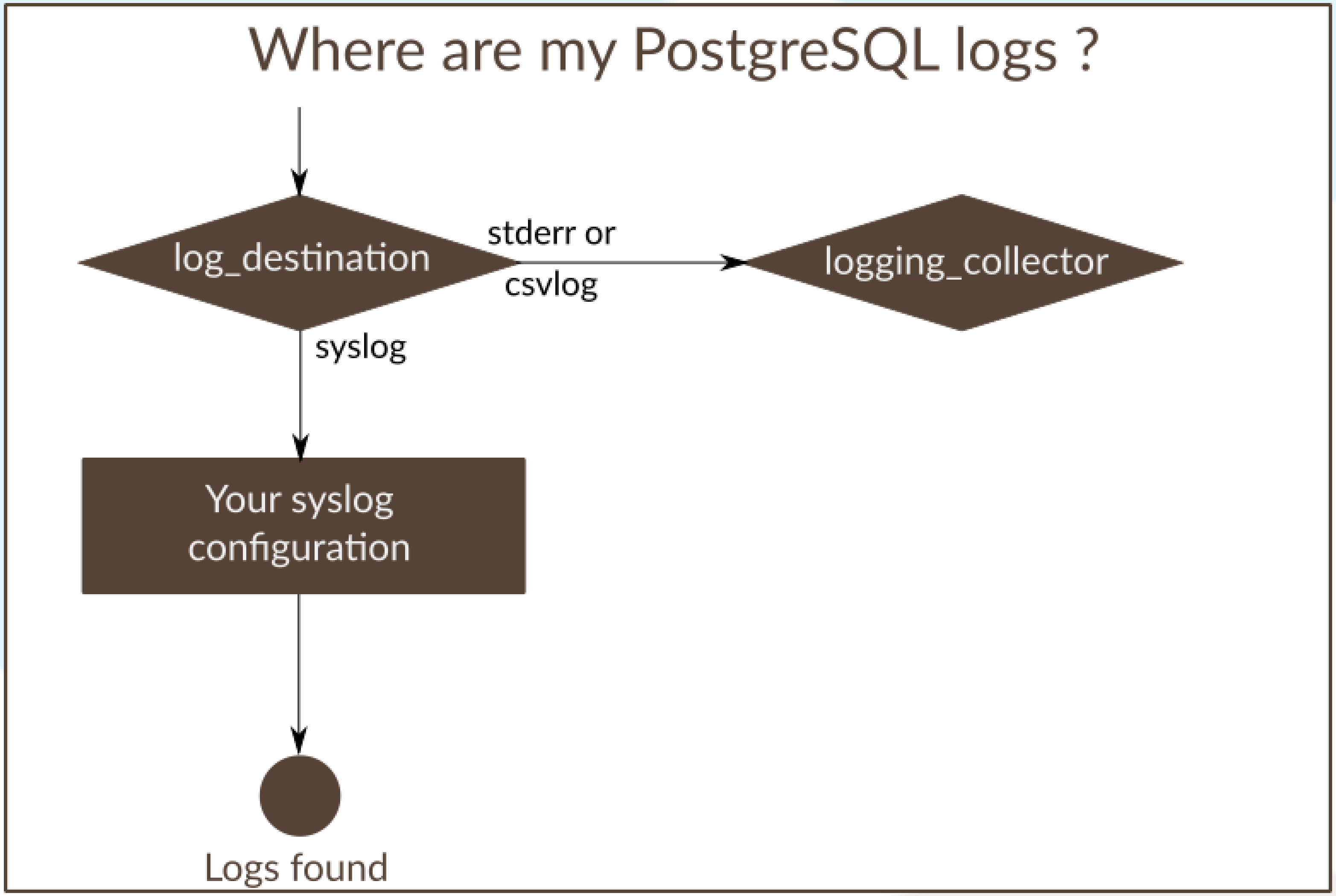
Where are my PostgreSQL logs ?

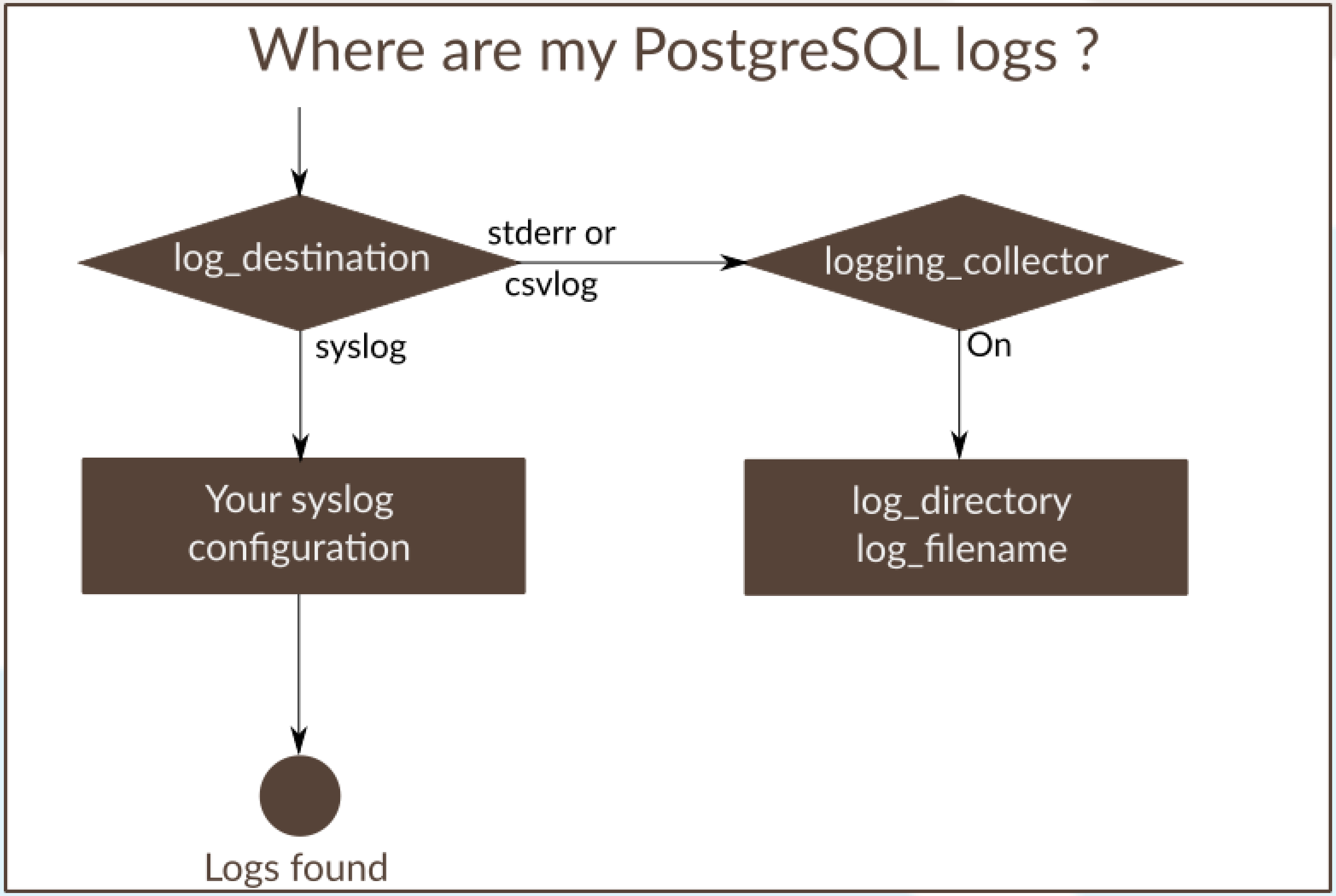


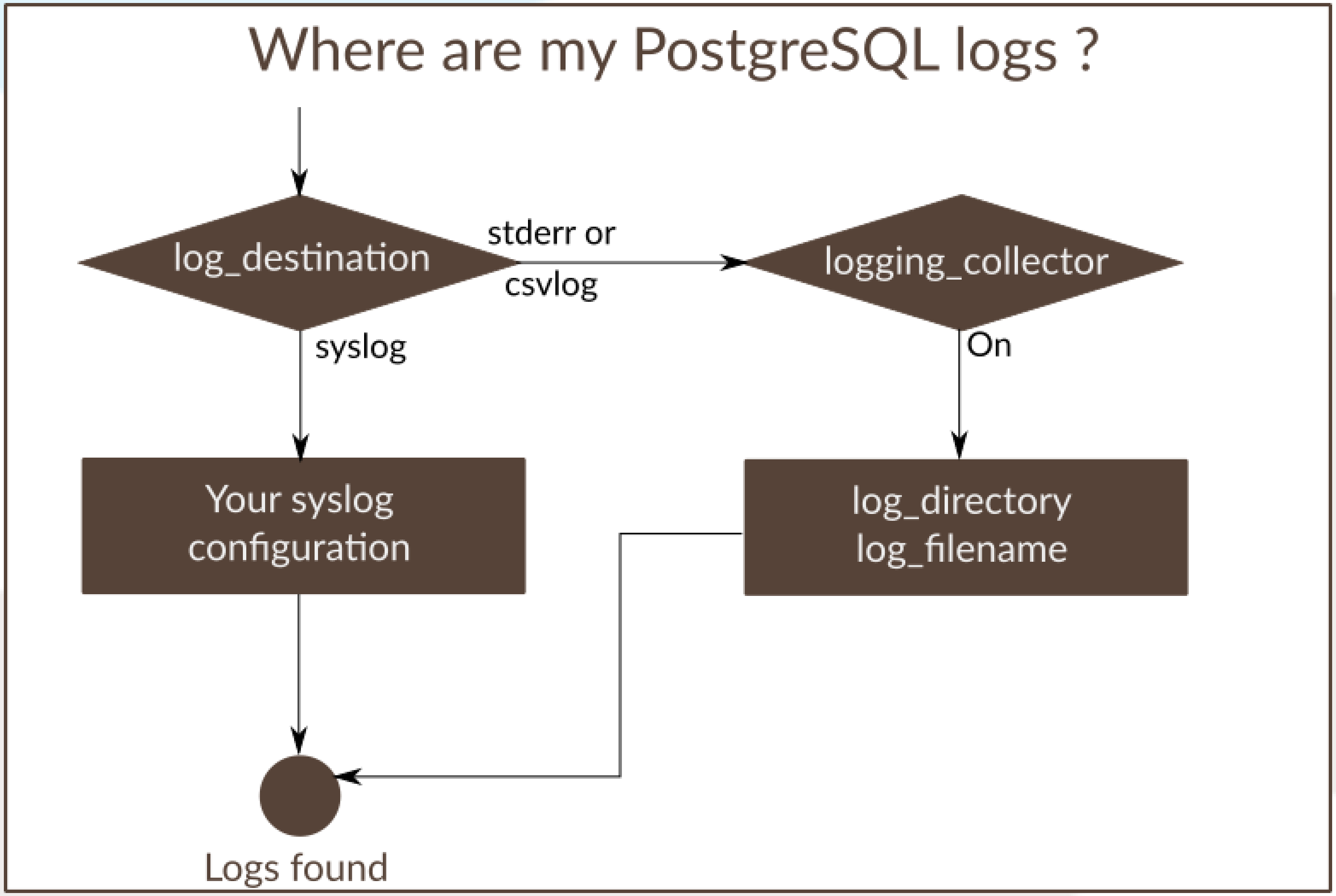
LOXODATA

Where are my PostgreSQL logs ?

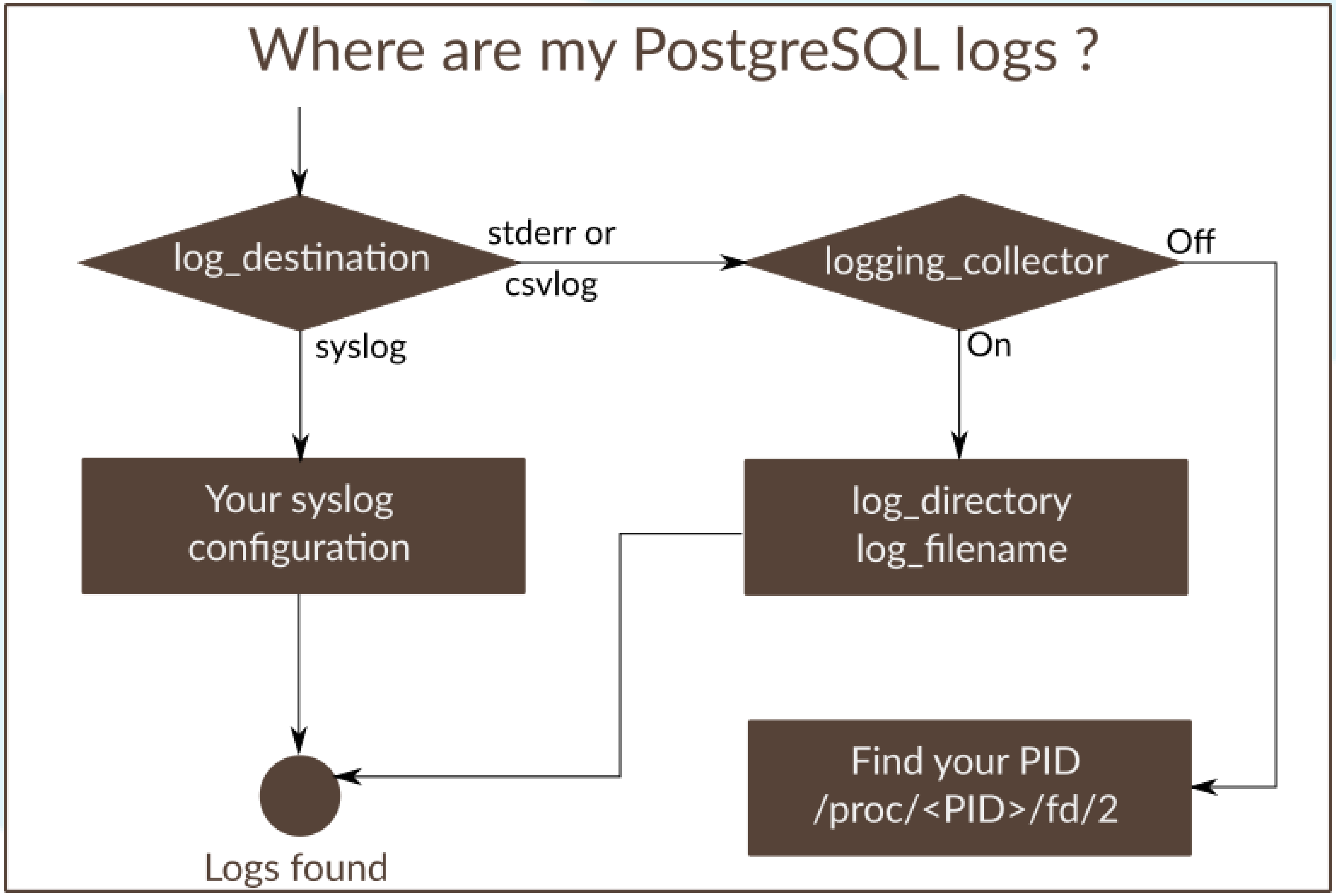



Where are my PostgreSQL logs ?
log_destination
stderr or
csvlog
logging_collector
syslog
Your syslog configuration
Logs found

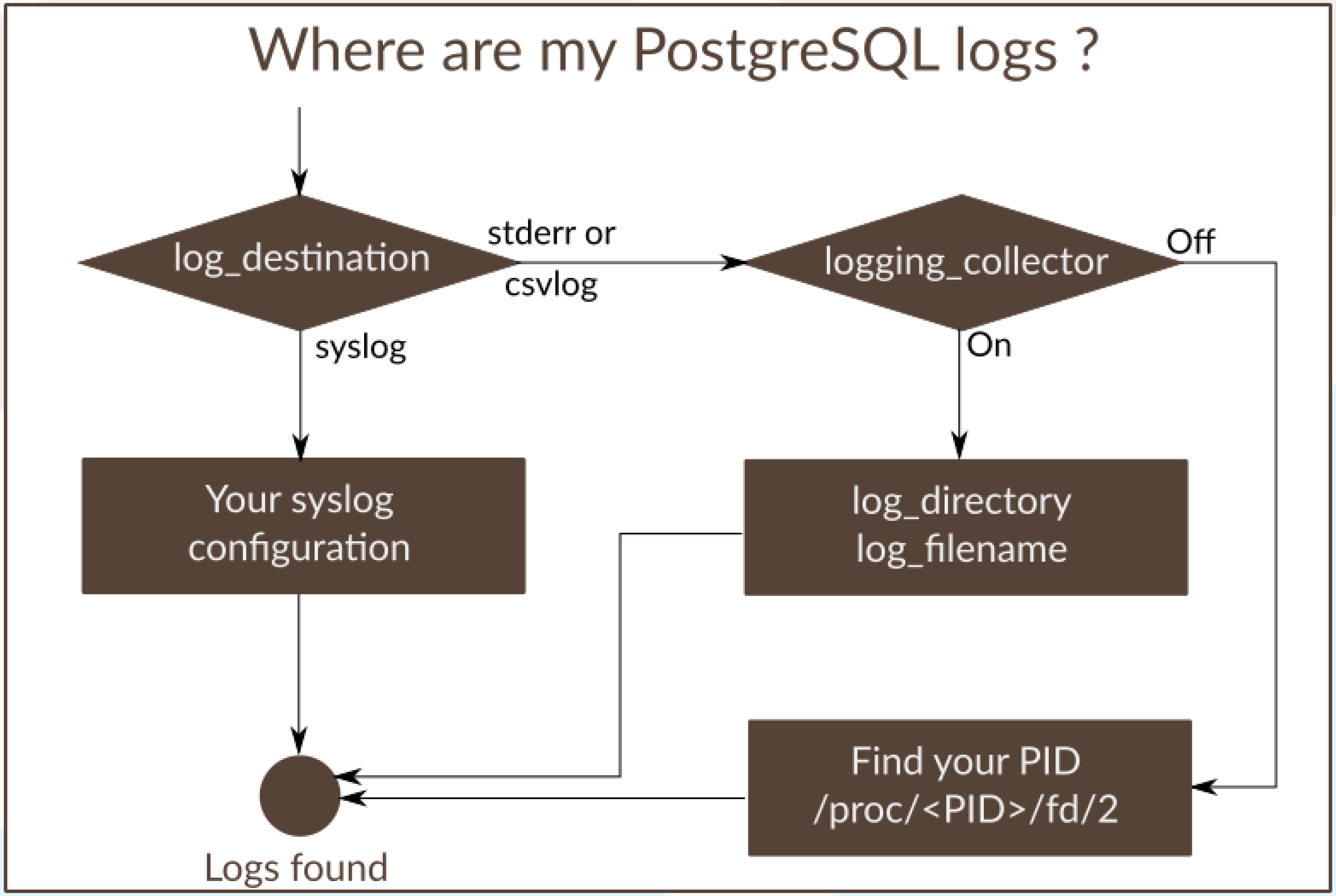

Where are my PostgreSQL logs ?
log_destination
stderr or csvlog
logging_collector
syslog
On
Your syslog configuration
log_directory
log_filename
Logs found


Where are my PostgreSQL logs ?
log_destination
stderr or csvlog
logging_collector
syslog
On
Your syslog configuration
log_directory
log_filename
Logs found

LOXODATA


Where are my PostgreSQL logs ?
log_destination
stderr or csvlog
syslog
logging_collector
Off
On
Your syslog configuration
log_directory log_filename
Find your PID /proc/<PID>/fd/2
Logs found

LOXODATA


Where are my PostgreSQL logs ?
log_destination
stderr or csvlog
syslog
logging_collector
Off
On
Your syslog configuration
log_directory log_filename
Find your PID /proc/<PID>/fd/2
Logs found

LOXODATA

pg_stat_statement

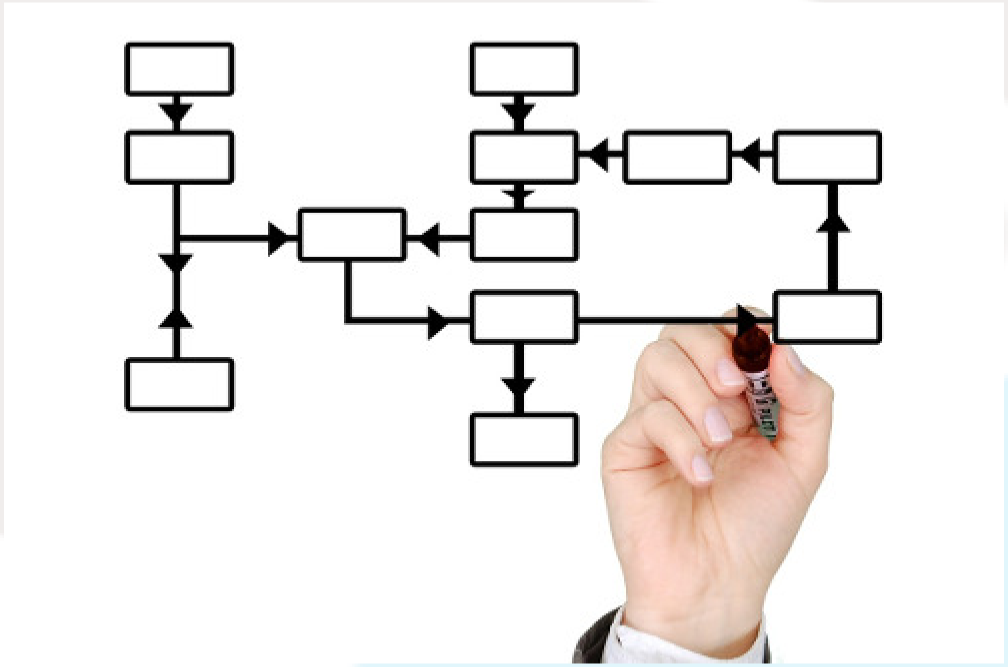
- PostgreSQL module
- Tool to track execution statistics for SQL statements
- Settings
  - pg_stat_statements.max
  - pg_stat_statements.track (top|all|none)

Tools

- PgHero
- Tsung
- PgBadger

LOXODATA

PLANNER



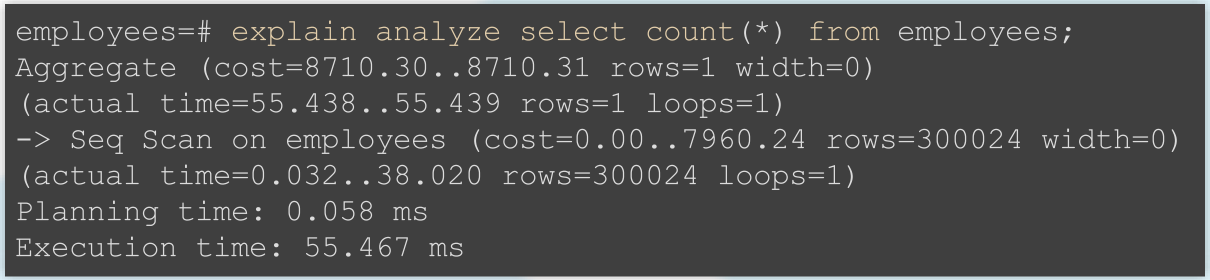
please explain

```
EXPLAIN [ANALYZE] statement
```

Examples:

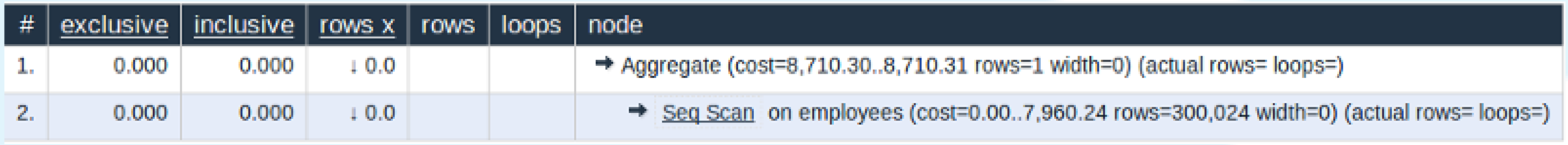
```
EXPLAIN SELECT name FROM employees WHERE salary > 10000
```

```
EXPLAIN ANALYZE SELECT name FROM employees WHERE salary > 10000
```



```
employees=# explain analyze select count(*) from employees;
Aggregate (cost=8710.30..8710.31 rows=1 width=0)
(actual time=55.438..55.439 rows=1 loops=1)
-> Seq Scan on employees (cost=0.00..7960.24 rows=300024 width=0)
(actual time=0.032..38.020 rows=300024 loops=1)
Planning time: 0.058 ms
Execution time: 55.467 ms
```

https://explain.depesz.com/



#	exclusive	inclusive	rows x	rows	loops	node
1.	0.000	0.000	↓ 0.0			➜ Aggregate (cost=8,710.30..8,710.31 rows=1 width=0) (actual rows= loops=)
2.	0.000	0.000	↓ 0.0			➜ Seq Scan on employees (cost=0.00..7,960.24 rows=300,024 width=0) (actual rows= loops=)

LOXODATA

Index and table access

- Seq Scan
- Index Scan
- Index Only Scan
- Bitmap Index Scan / Bitmap Heap Scan / Recheck Cond

LOXODATA

Joining

- Nested Loops
- Hash Join / Hash
- Merge Join

LOXODATA

Sorting and Grouping

- Sort
- GroupAggregate
- HashAggregate

Top-N Queries

- Limit
- WindowAgg

ROOTS OF THE EVIL

LOXODATA

Bad SQL...

```
SELECT DISTINCT employees.id,
  employees.last_name,
  employees.first_name,
  employees.birth_date
FROM departements
LEFT OUTER JOIN dept_manager ON
  departements.id=dept_manager.id_department
INNER JOIN employees ON
  dept_manager.id_employee=employees.id
ORDER BY employees.birth_date DESC
LIMIT 10
```

Execution time : 9128.081 ms

LOXODATA

... made better!

```sql
SELECT employees.id,
  employees.last_name,
  employees.first_name,
  employees.birth_date
FROM employees
ORDER BY employees.birth_date DESC
LIMIT 10
```

Execution time : 0.042s
217 000 times faster

How to clean SQL

1. Remove DISTINCT (if useless)
2. Remove useless sorts
3. Remove useless joins
4. Remove CROSS JOINS

Explain

- "Wrong" scan
- Good scan but too slow
- Remaining sort operations

Existing indexes

- Function on a column
- Old stats

- Bloat
- Do all these data need to be kept?

Creating indexes

- Multi-column, functional, partial
- But
  - remember it slows down writing operations
  - be sure it is used

LOXODATA

The statistic collector views

- pg_stat_user_tables
- pg_stat_user_indexes

Tapapakanga Regional Park
This natural masterpiece is cared for by the Auckland Regional Council

Materialized views

- Stored result
- Needs refreshing

Partitionning

- Easier in PostgreSQL 10
- Find a good partitioning key

CONCLUSION