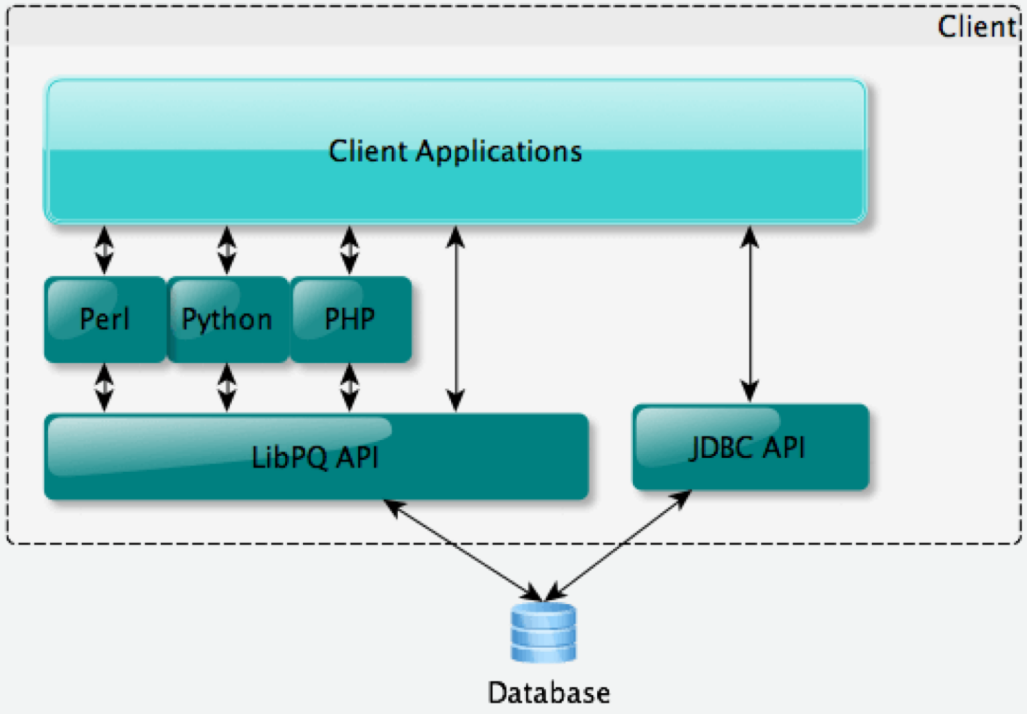
PostgreSQL Architecture

Client Architecture



aws

Client Side

- Databases are accessed via some sort of client
  - Libraries available for many languages
    - C (native)
    - C++
    - Java
    - Perl
    - Python
    - etc…
  - Some clients re-use the provided C-api, others (i.e. JDBC) implement the protocol natively
- These clients need to be able to speak to the database

Client Side

- Connectivity
  - PostgreSQL can be connected to via standard TCP/IP networks. It has a wire-level protocol commonly referred to as ‘libpq’
  - Note: libpq is also the name of the client-side library that implements the protocol
- Language
  - Once connected, you interface with postgres by sending commands to it.
    The language used is a combination of SQL:2008 compliant statements and Postgres maintenance commands.

aws

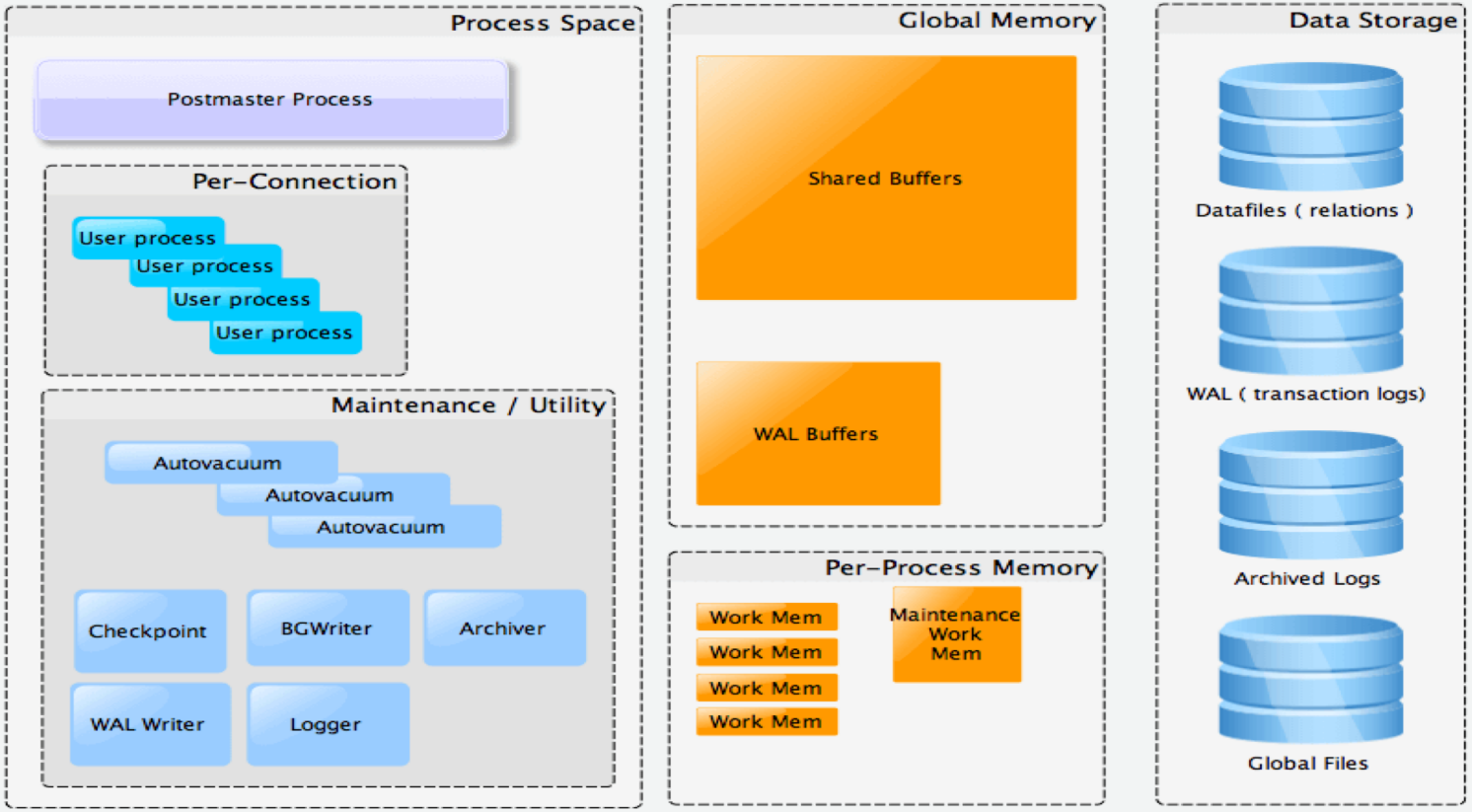
Client Components

- Libpq
  - Native (C-based), client-side API
  - Implements wire-level protocol for server communication
- JDBC
  - Java, client-side API
  - Does not re-use libpq library, implements protocol directly
  - This provides simple use for java users, no need to install the native libpq library

Server Overview

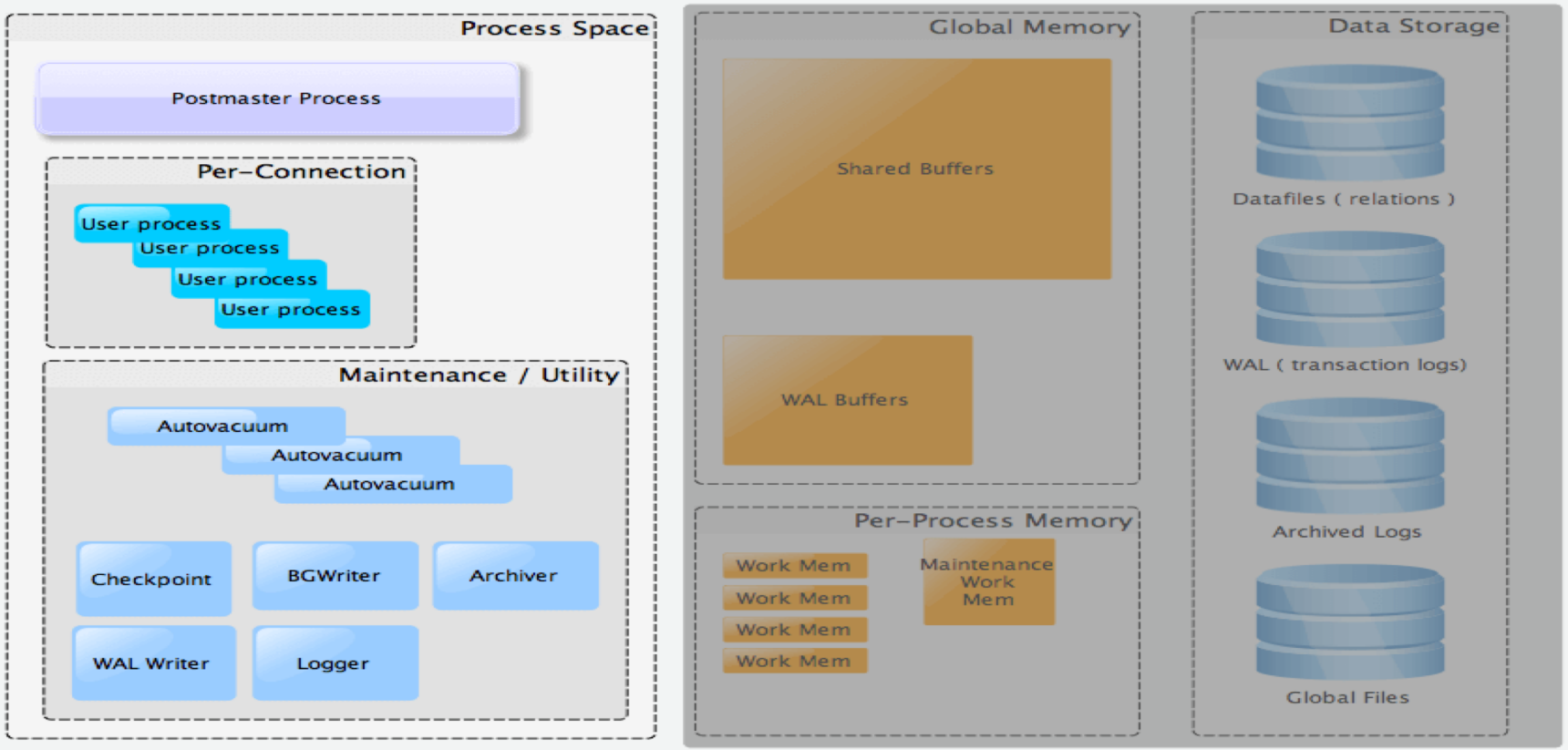
- PostgreSQL utilizes a multi-process architecture
- Similar to Oracle's 'Dedicated Server' mode
- Types of processes
  - Primary (postmaster)
  - Per-connection backend process
  - Utility (maintenance processes)

Server Overview



aws

Process Components



aws

Process Components

- Postmaster
  - Master database control process
  - Responsible for startup & shutdown
  - Handling connection requests
  - Spawning other necessary backend processes
- Postgres backend
  - Dedicated, per-connection server process
  - Known as a ‘worker’ process
  - Responsible for fetching data from disk and communicating with the client

Server Process Components - Utility

- Autovacuum
  - Dedicated backends for providing vacuum services
  - Essentially, a garbage collect of the datafiles
  - Covered later in ‘Maintenance’ section
- Writer
  - Background writer
  - Flushes memory cache to disk
- Wal-writer
  - Responsible for maintaining transaction log (journal)
  - Only used for asynchronous commits

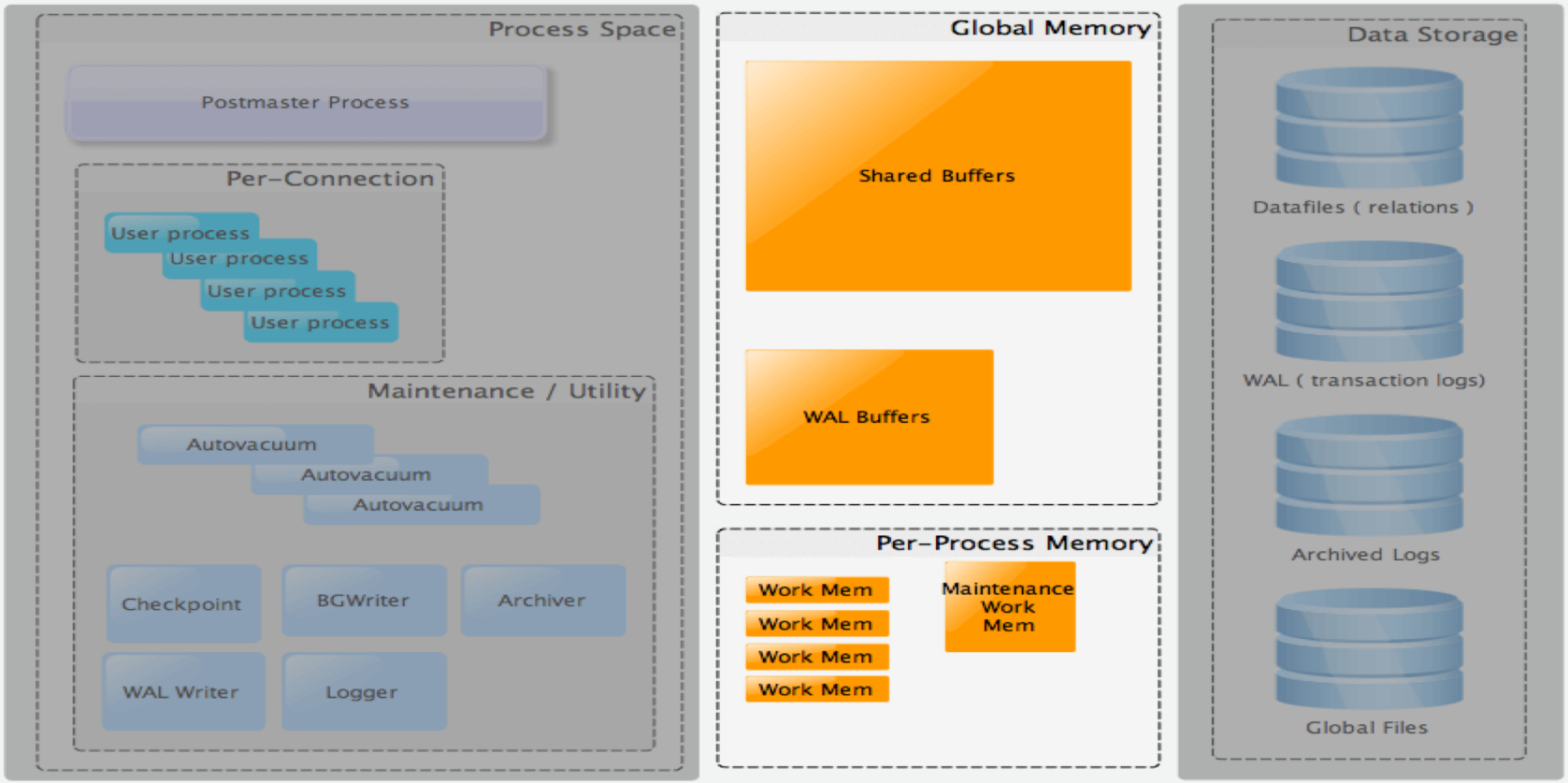
Server Process Components - Utility

- Checkpointer
  - This process performs checkpoints
  - checkpoints flush all dirty buffers to disk
- Archiver
  - Saves WAL files to a specified location for backup purposes
  - Can also be used to achieve replication
- Logger
  - Responsible for writing information logs
  - Errors, warnings, slow running queries, etc…
  - Not used if writing to syslog

Server Process Components - Utility

- Stats Collector
  - Supports collection and reporting of information about server activity
  - Can count accesses to tables and indexes
  - Information about vacuum and analyze actions
- Bgworkers
  - Supports external extensions
  - Logical replication

Memory Components



aws

Memory Components - Shared Buffers

- Shared Buffers is the primary cache component for the server which stores the disk blocks from the database files.
- All the data sets accessed from the disk are placed in shared buffers allowing subsequent reads to be memory reads.
- All writes are performed in shared buffers creating ‘dirty’ pages. The bgwriter and checkpoint processes will write this out to disk.
- Shared buffers contain “free buffers” (buffers that are never used or freed after using) and “dirty buffers” (buffers which are resultant of DML)

aws

Memory Components - WAL Buffers

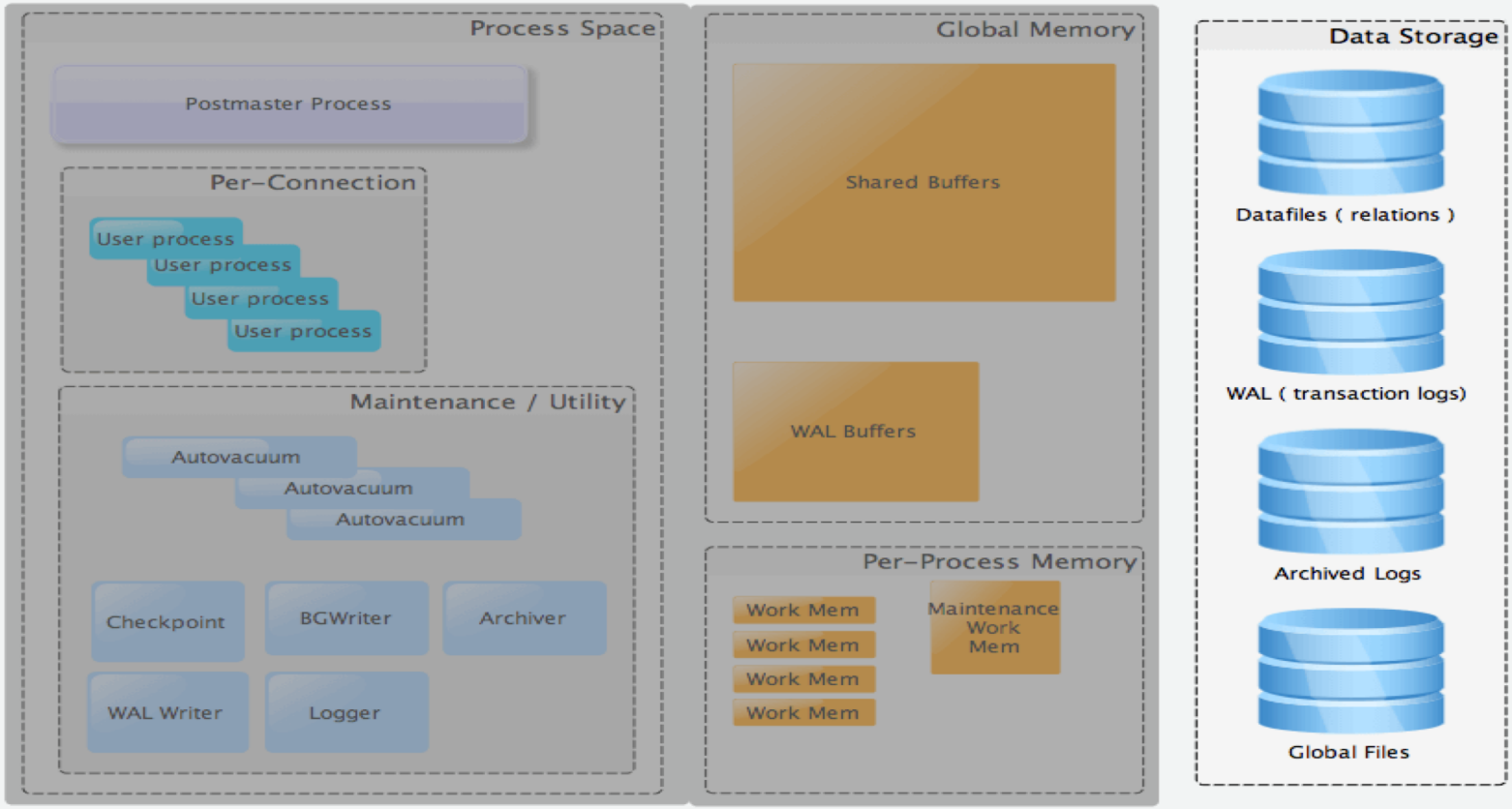
- Stores intermediate Write Ahead Log records
- Written on commit by walwriter process (or when full)

aws

Per-Session Memory

- work_mem
  - Per-Backend sort / hash memory
  - Used during certain types of JOINs and for ORDER BY operations
  - Set globally, but can be modified per-session
- maintenance_work_mem
  - Used for certain types of maintenance operations (vacuum, index creation, re-index)
  - Allocated per session that uses it (i.e. multiple autovacuum workers)

On-Disk Components



aws

Data Directory

- All of the on-disk components of a database instance are stored in a data-directory
- This directory contains multiple sub-directories
- Some sub-directories can be moved by configuration, others by symlink (unix/linux only)
- Very few of the files are user-readable or modifiable

aws

Data Directory

Commonly referred to as $PGDATA

- PG_VERSION
  - Version string of the database instance
- pg_hba.conf
  - Host-based access control (built-in firewall)
- pg_ident.conf
  - Ident-based access control
- postgresql.conf
  - Primary configuration file
- postgresql.auto.conf
  - Used by ALTER SYSTEM

aws

Data Directory

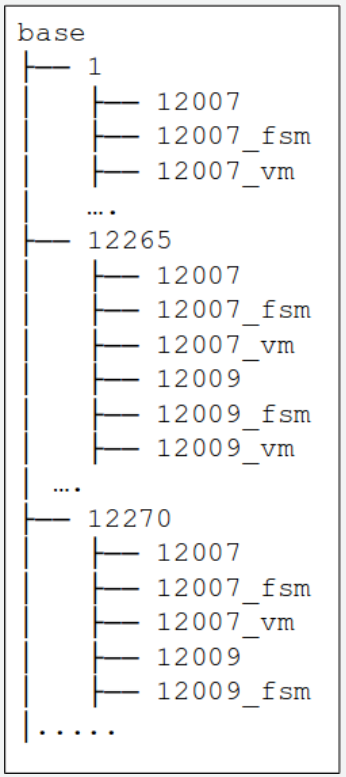
- postmaster.opts
  - What options were used to start the instance
- postmaster.pid
  - The process ID of the instance
- server.crt
  - Server certificate for SSL
- server.key
  - Server private key
- root.crt
  - Trusted certificate authorities

On-Disk Components

- Datafiles
  - Also referred to as ‘relations’ or ‘base’ files
  - Stored in the ‘base’ directory
  - Contains table or index data
    - Tables have a set of segments
    - Indexes have a set of segments
  - Stored on-disk as 1 gigabyte segments
    - A 5 GB table will be 5 1GB files
    - Segments are automatically added, no intervention required
  - Filled with data in 8kb blocks (on-disk) / pages (in-memory)
    - Blocks and pages are identical, the name changes based on where it currently resides, when they are in memory they are referred to as pages.

base Directory

- Contains the datafiles for your relations
  - sub-directories identify each database in the instance
  - each database subdirectory, files correspond to relations
  - These filenames map to a table's relfilenode
    - queryable via pg_class
  - _fsm and _vm refer to per -table maintenance structures and are discussed later

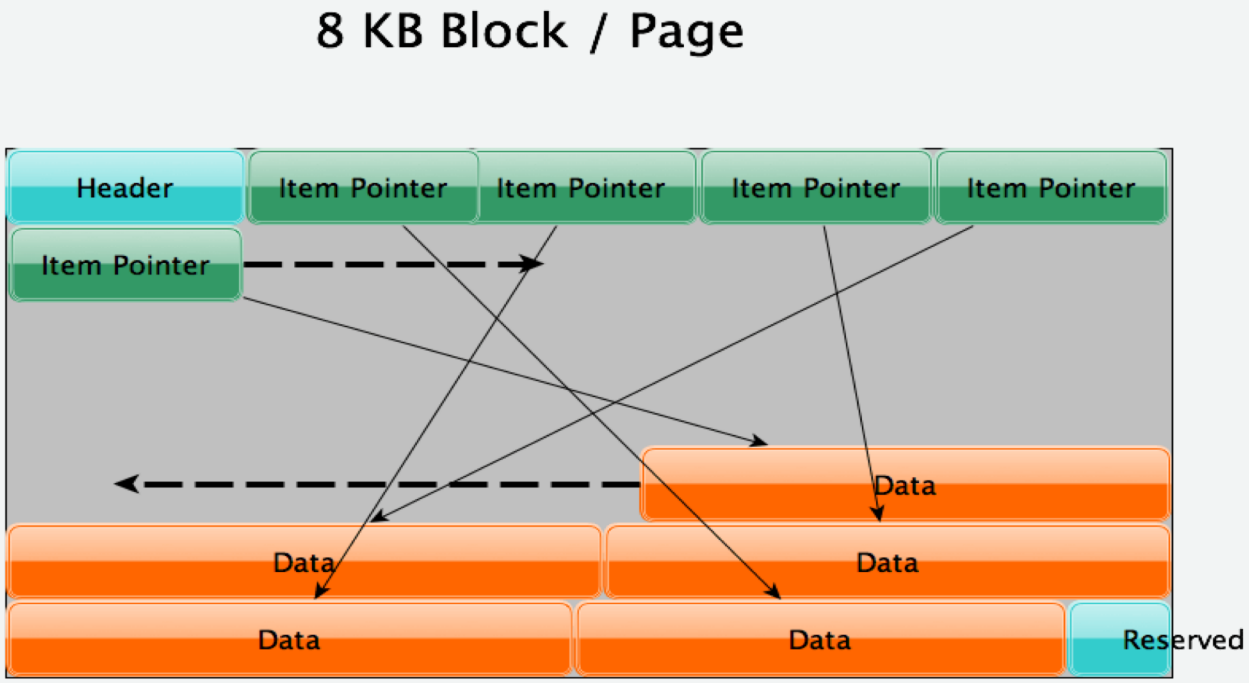


aws

Block / Page Layout

- Each 1 GB datafile is made up of 8 KB blocks
  - 8 KB is change-able at compile time
- Blocks (disk) and pages (memory) are identical
  - Page Header (20 bytes)
  - Item pointers
    - An array of pointers to the actual data in the page
    - Filled from the front to the back
  - Data
    - Filled from the rear of the page towards the front
  - Free space
    - Between item pointers and data
  - Reserved section (primarily for index pages)

On-Disk Components



Storing Large Values

- Uses TOAST tables for large values
- The Oversized Attribute Storage Technique
- Fixed length types are not TOASTable
- Compression using zlib is possible

aws

TOAST Table

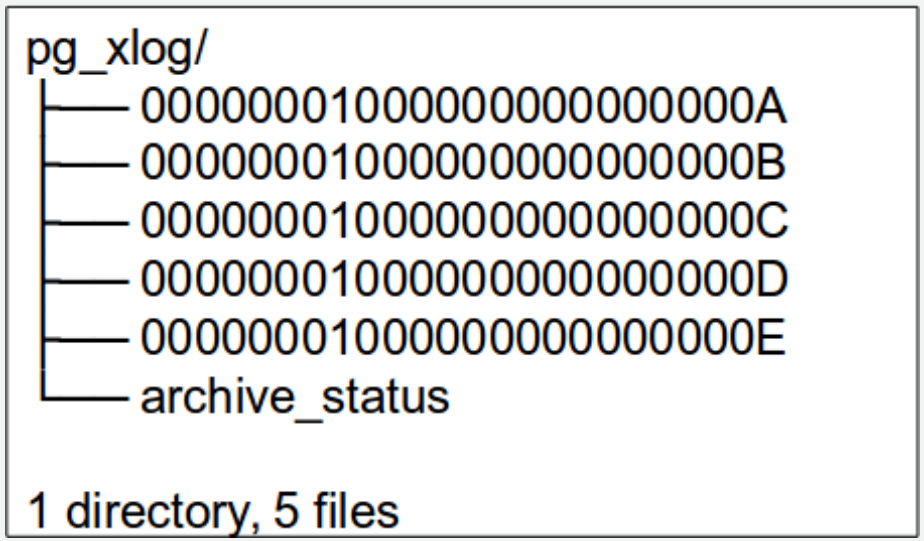
```
TOAST table "pg_toast.pg_toast_12304"
   Column    |  Type
-------------+---------
 chunk_id    | oid
 chunk_seq   | integer
 chunk_data  | bytea
```

On-Disk Components

- Write Ahead Log (WAL) Files
  - This is the transaction journal
  - Also known as: WAL, xlog, pg_xlog, transaction log, journal, REDO
  - These files contain a persistent record of COMMITs
  - Success is not returned to the client until this is safely persisted to disk
  - During crash recovery, this journal can be used to reconstruct all transactions
  - These files are written by the walwriter process

pg_wal Directory

- Contains active WAL segments
  - Segments are recycled after time
  - Segments are fixed at 16 MB
  - Can be changed at instance ‘init’ time
  - Variable number of files
    - checkpoint_segments (< 9.5)
    - max_wal_size (9.5+)
  - Move pg_wal (symlink) to dedicated disk
  - Was referred to as pg_xlog (< 10)



aws

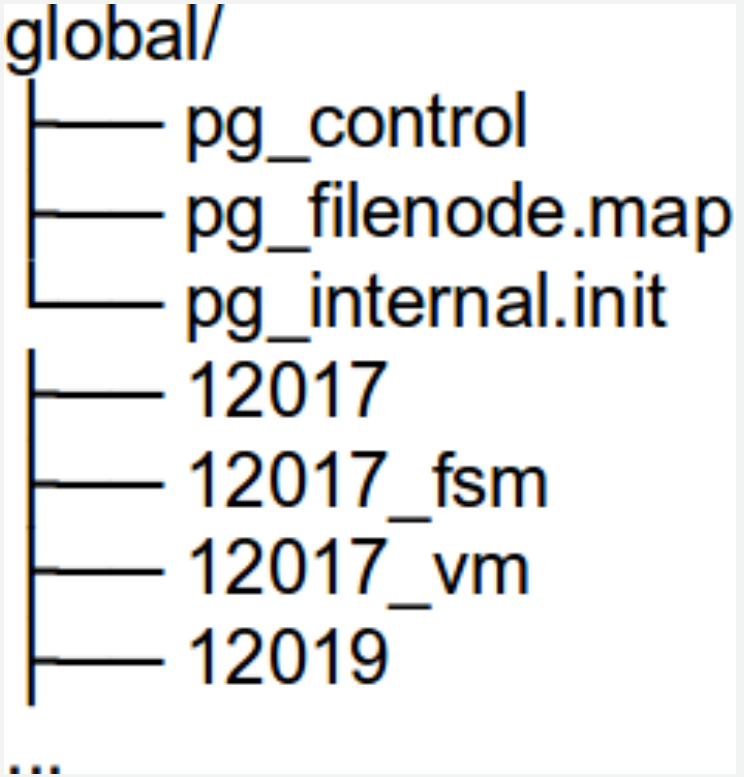
On-Disk Components

- Archived Logs
  - Archived versions of the WAL
  - An archived log has the same format as a non-archived log, it is just kept in a [user-defined] directory other than pg_wal
  - After a WAL file is closed, the ‘archiver’ process will perform the archive operation on it
    - Essentially, just a ‘cp’ or ‘rsync’
    - Command is user-definable in postgresql.conf
  - Archiving can be enabled or disabled

aws

On-Disk Components

- Global Area
  - Shared catalogs
  - System views
  - Control file
  - Can be accessed across databases
  - Cannot be modified except under special circumstances



aws

In-Motion

- How is this better than a text file?
  - Accept client requests and operate on them
  - Ensure that the data is safe on storage
  - Enforce A.C.I.D.
- Let’s see some examples of how the architecture supports these two equally important goals

aws

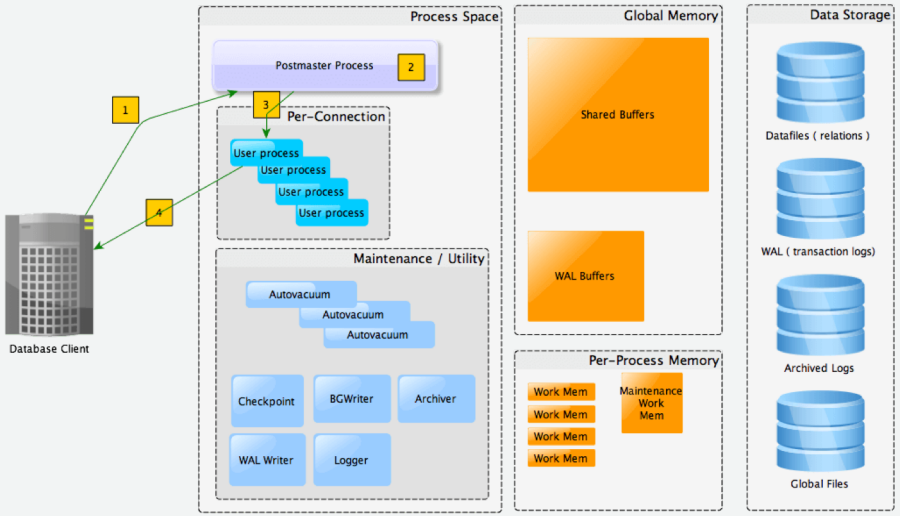
Connect Request

1. A client connection is sent to the postmaster

2. Authentication is performed

3. The postmaster spawns a user-backend process

4. The user-backend calls back to the client to continue operation



Reading Data

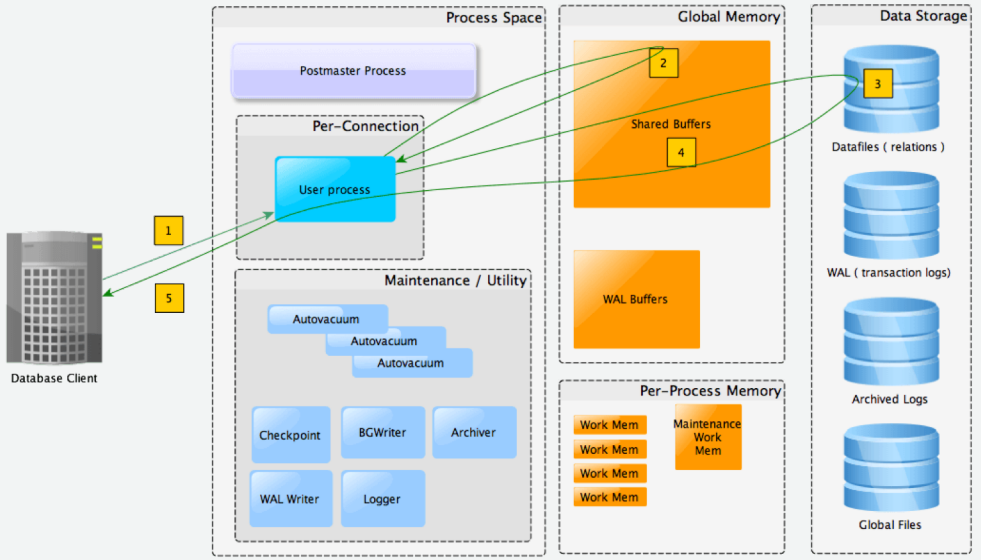
1. The client issues a query, the user-backend performs the read

2. If the data is in cache shared_buffers, it is a memory read

3. If not, the user-backend reads from the data files

4. The user-backend copies the data to shared_buffers

5. The data is returned to the client



Writing (or deleting) Data

1. Client issues write request

2. User-backend checks for data in shared_buffers

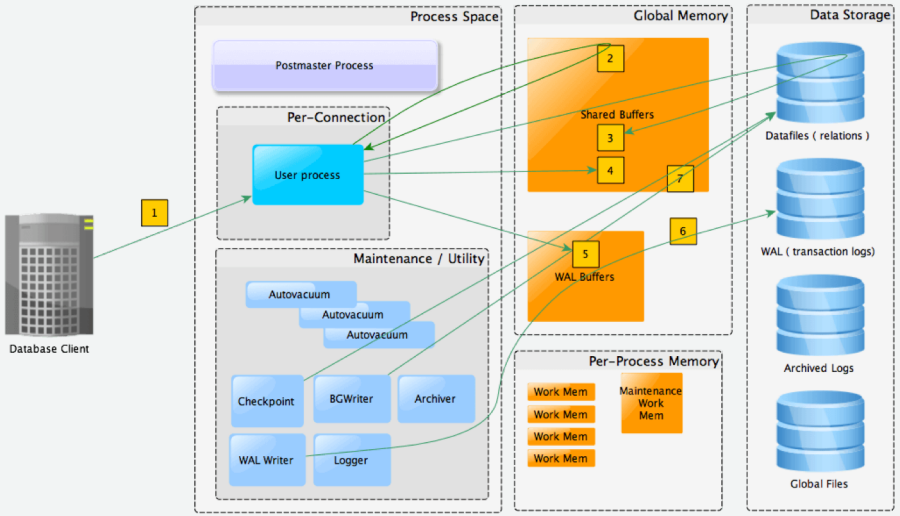
3. If not, reads into shared_buffers

4. User-backend dirties data in shared_buffers

5. User-backend records transaction in WAL

6. On commit, walwriter commits wal_buffers to pg_xlog

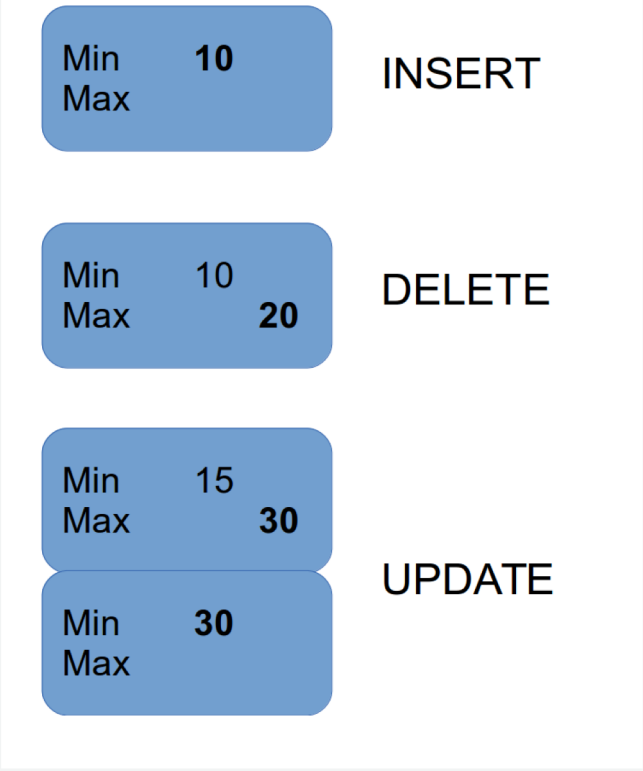
7. bgwriter (clock sweep) or checkpoint (forced) writes dirtied buffers to disk



What is MVCC?

- Multiversion Concurrency Control
  - Allows Postgres to offer high concurrency even during significant database read/write activity
  - Readers never block writers, and writers never block readers
  - Reduces locking requirements, but does not eliminate locking

MVCC Behavior



- Visibility is driven by transaction IDs (XID)
- Tuples have an XMIN and XMAX
  - XMIN is the XID that created the tuple
  - XMAX is the XID that removed the tuple

aws

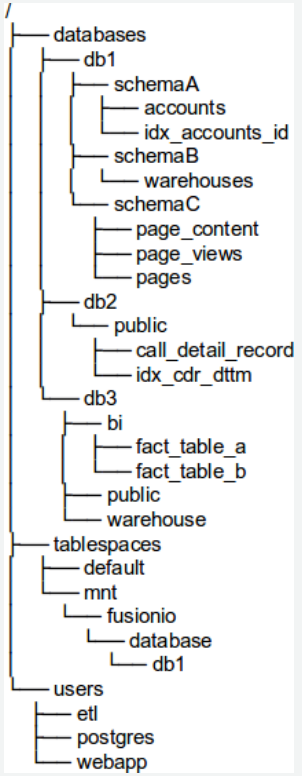
XMIN

```
postgres=# INSERT INTO foo VALUES (51);
INSERT 0 1
postgres=# SELECT xmin, xmax, * FROM foo;
 xmin | xmax | f1
------+------+----
 8531 |    0 | 51
(1 row)
```


aws

Logical Layout

- PostgreSQL has a very ‘filesystem’ like layout internally
- Logical structures may or may not tie to physical structures
- Layout is controlled by the postgres catalog tables (pg_catalog)



aws

Logical-Physical Mappings

- Instance
  - Ties to a directory (data directory)
  - 1-1 tcp/ip port
  - 1 postmaster
  - 1 shared_buffers
- Database
  - Multiple databases per instance
  - Each database maps to a directory under ‘base’ in the instance’s data-directory
- Schema / namespace
  - Purely logical grouping of relations
  - A schema only exists in the catalogs, has no real physical structure

Logical-Physical Mappings

- Tablespaces
  - A directory on disk where postgres can store relations
  - Used for tiering storage

- Users
  - Can be assigned across databases
  - Do not have physical structure (other than in catalogs)

Summary

- PostgreSQL is processed based
- Designed to be OS and file system agnostic
- Allows for easier extensibility

aws