

1. PostgreSQL Introduction

Introduction to PostgreSQL



PostgreSQL

AGENDA

- PostgreSQL and databases
- PostgreSQL history and milestones
- PostgreSQL features
- PostgreSQL architecture























POSTGRESQL

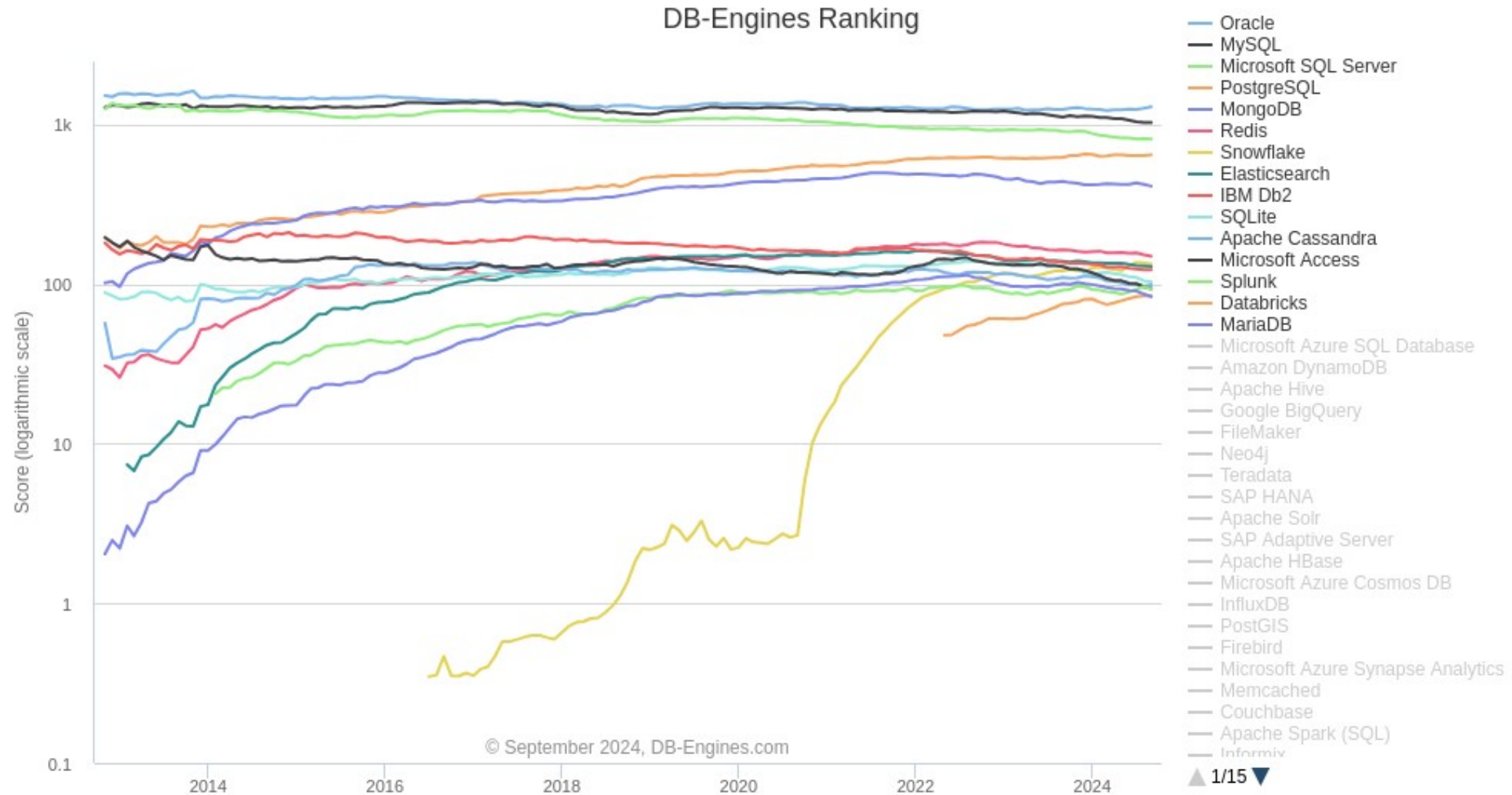
- PostgreSQL is an enterprise scale, open-source relational database management system
- Competitive in the enterprise marketplace with the major commercial databases like Oracle and SQL Server.
- Also competitive with the more lightweight database MySQL and the MariaDB fork.



DB-ENGINE RANKINGS

Rank			DBMS	Database Model	Score		
Sep 2024	Aug 2024	Sep 2023			Sep 2024	Aug 2024	Sep 2023
1.	1.	1.	Oracle 	Relational, Multi-model 	1286.59	+28.11	+45.72
2.	2.	2.	MySQL 	Relational, Multi-model 	1029.49	+2.63	-82.00
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	807.76	-7.41	-94.45
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	644.36	+6.97	+23.61
5.	5.	5.	MongoDB 	Document, Multi-model 	410.24	-10.74	-29.18
6.	6.	6.	Redis 	Key-value, Multi-model 	149.43	-3.28	-14.26
7.	7.	 11.	Snowflake 	Relational	133.72	-2.25	+12.83
8.	8.	 7.	Elasticsearch	Search engine, Multi-model 	128.79	-1.04	-10.20
9.	9.	 8.	IBM Db2	Relational, Multi-model 	123.05	+0.04	-13.67
10.	10.	 9.	SQLite 	Relational	103.35	-1.44	-25.85

DB-ENGINE ADOPTION TRENDS

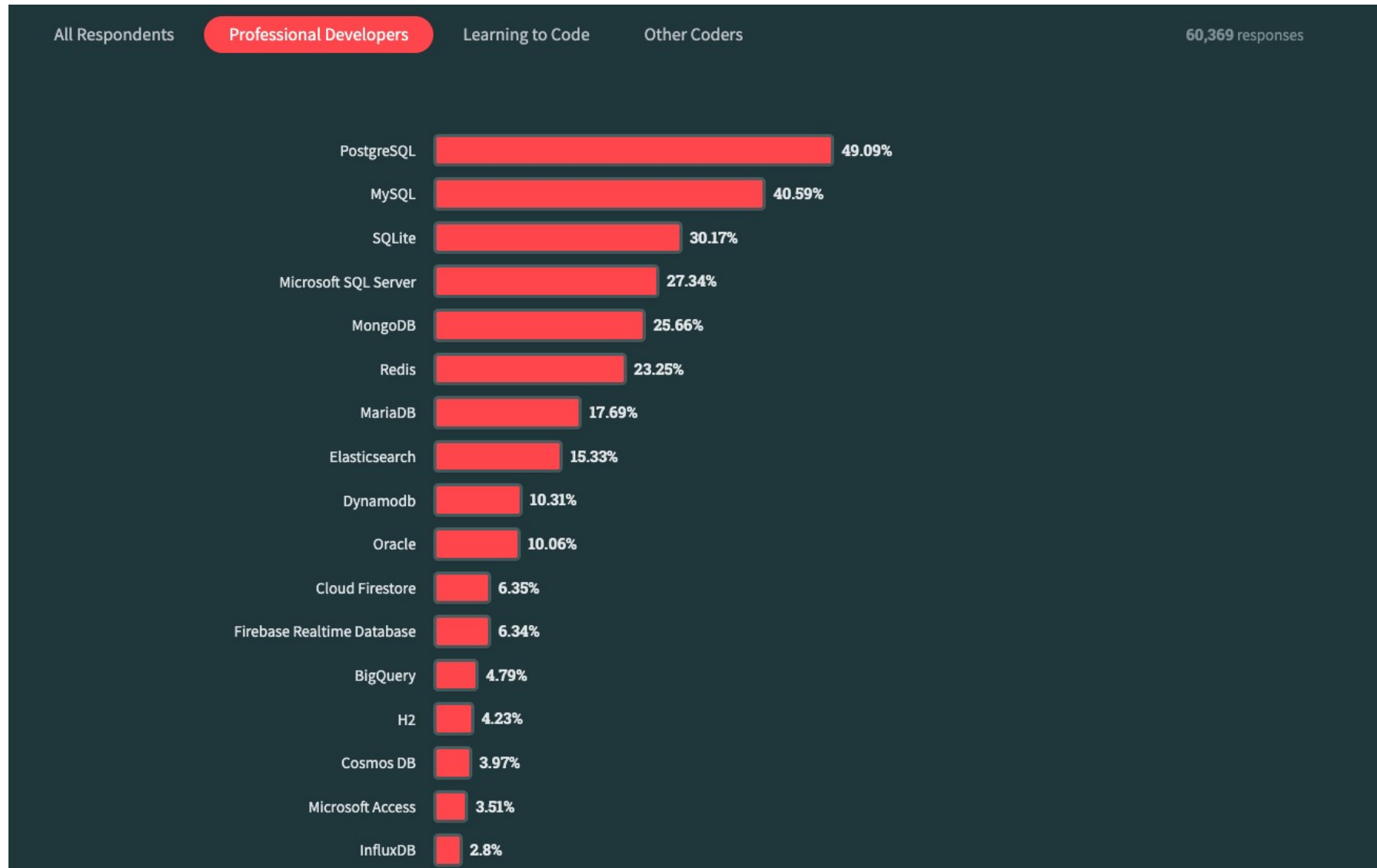


MIGRATION EXAMPLE

- Amazon originally hosted all its data on Oracle
 - Decision was made to reduce dependency on commercial database solutions
 - Cost and vendor locking were two main reasons
 - All of Amazon data was migrated to cloud based versions of PostgreSQL by 2019
- Post-migration, Amazon reported significant cost savings and performance improvements.
 - The open source model gave the freedom to modify the architecture while still maintaining PostgreSQL compatibility
 - For example, separating storage and compute onto different nodes for greater efficiency



STACK OVERFLOW – FAVORITE DATABASE



CLOUD IMPLEMENTATION

- Because PostgreSQL is both industrial strength and open source, it has been the basis for cloud-based RDBs.
 - Amazon RDS for PostgreSQL2.
 - Amazon Aurora with PostgreSQL Compatibility
 - Google Cloud SQL and AlloyDB for PostgreSQL
 - Azure Cosmos DB for PostgreSQL
 - IBM Cloud Databases for PostgreSQL
- And others
 - The open source model gives cloud providers the freedom to architect the design as they need to for their physical infrastructure



HISTORY

- The INGRES (**I**nteractive **G**raphics and **R**etrieval **S**ystem) started in 1973 at U of C Berkeley by Professor Michael Stonebraker and Eugene Wong
- One of the first implementations of the 1973 relational model developed by Ted Codd at IBM
 - IBM was developing its own prototype relational DB called System R.
 - In 1982, Stonebraker left the project to develop a commercial version of Ingres
 - He returned to Berkeley in 1985, and began a post-Ingres project called POSTGRES (Post-INGRESS).



HISTORY

- The goal of POSTGRES was to address the problems in 1980s implementations of the relational model such as:
 - Supporting constraint relationships
 - Complex data types
- POSTGRES introduced a number of major innovations now standard features in modern relational databases. Examples:
 - “Time travel,” which allowed for historical data to be stored and queried without complex database schema changes, later codified in ISO SQL 2011
 - The use of rules and triggers which are now standard features in modern relational databases.



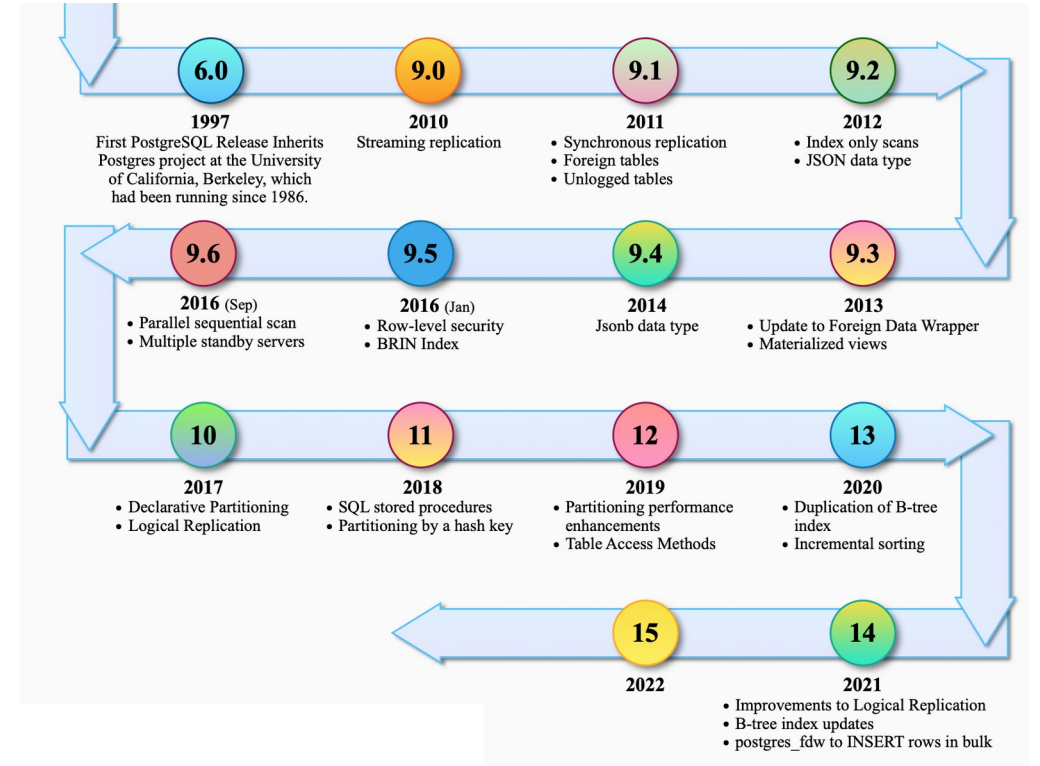
HISTORY

- POSTGRES used a custom query language, PostQUEL
- Eventually replaced by SQL, aligning it with industry standards
- In 1994, the project was open-sourced project and “Postgres95.”
- In 1996, SQL was adopted as the query language replacing the original PostQUEL query language, and was renamed PostgreSQL



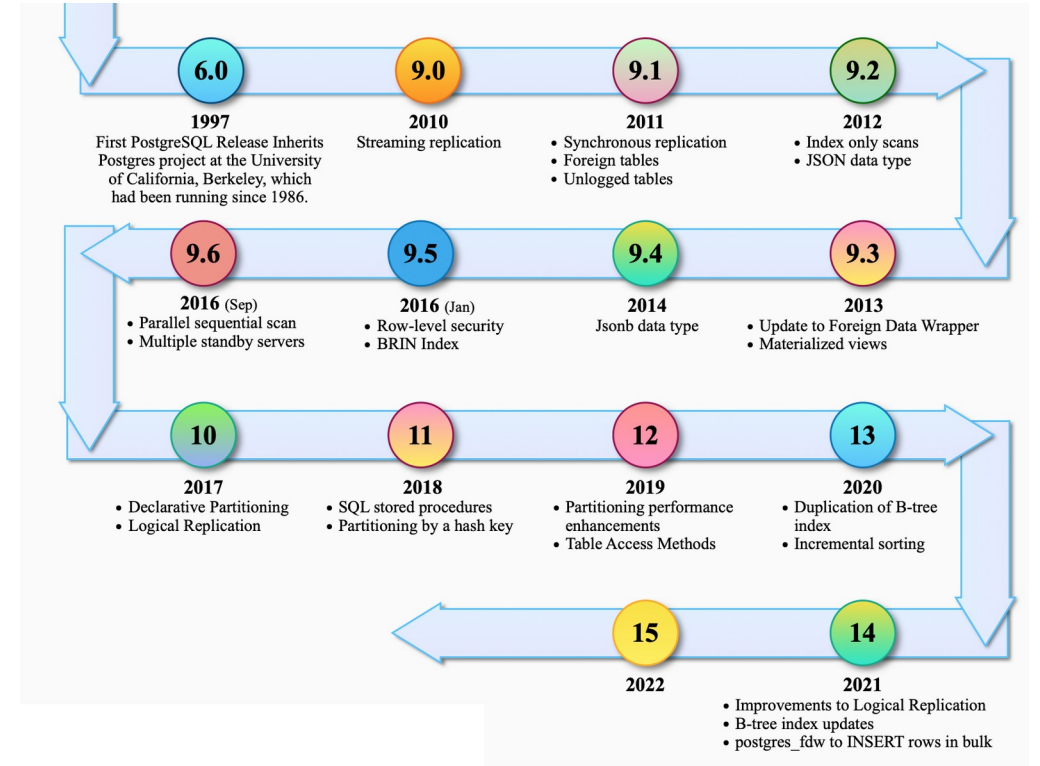
MILESTONES

- The product has evolved through a series of key milestones
- 1997-2000
 - Initial support for MVCC (Multi-Version Concurrency Control)
 - Improved database performance
 - Allowed higher levels of concurrent access without associated locking issues.
- 2003-2005
 - Point-in-Time Recovery (PITR), tablespaces, and savepoints
 - Improved the robustness and manageability of PostgreSQL in operational environments.



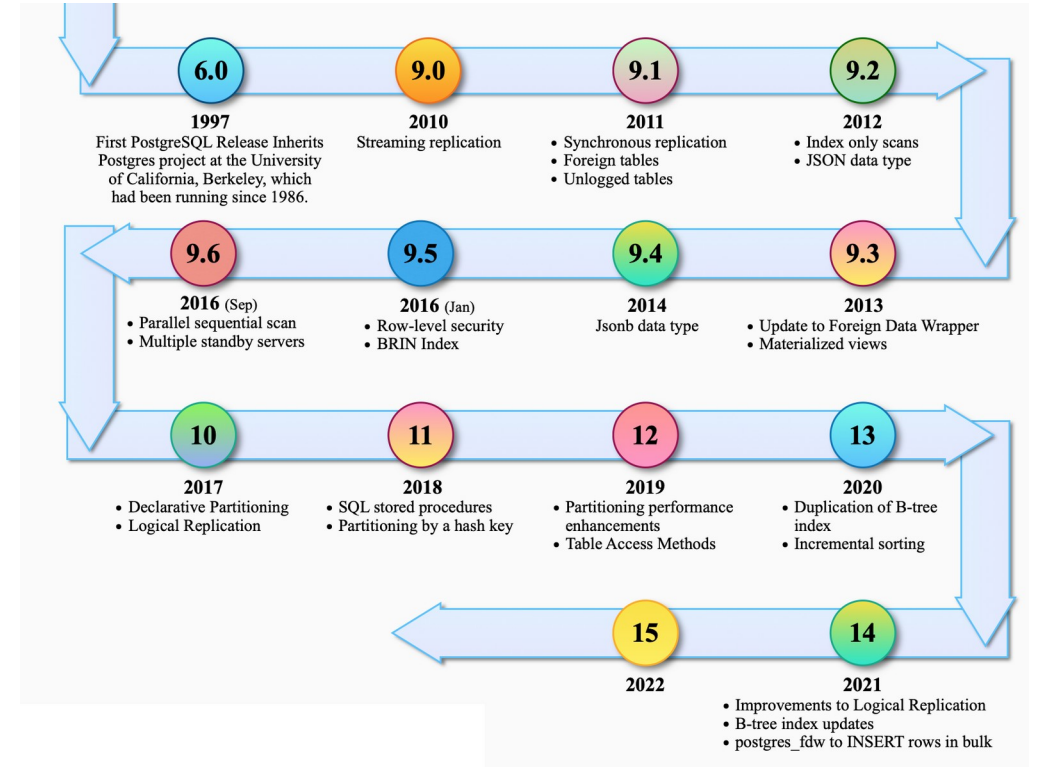
MILESTONES

- 2008-2010
 - Expanded support for semi-structured data types, like JSON
 - Added support for NoSQL databases.
 - Hot Standby and Streaming Replication were introduced, improving its high-availability capabilities.
- 2013-2014
 - JSONB, a binary storage format for JSON data, was introduced
 - Improved performance for document-based workloads.



MILESTONES

- 2016-2017
 - Introduced parallel query execution to take advantage of multicore processors for faster query processing.
 - Introduced declarative partitioning and logical replication to support managing large datasets and complex replication configurations.
- 2019-Present
 - Performance enhancements
 - Improvements in partitioning and indexing,
 - Expanding extensibility with support for custom data types, foreign data wrappers, and other features.



KEY FEATURES

- Some features of PostgreSQL relevant to modern enterprise data applications
- ACID Compliance and Data Integrity
 - Full ACID (Atomicity, Consistency, Isolation, Durability) for transaction management and data integrity.
- Advanced SQL Capabilities
 - Supports a wide range of SQL standards
 - Including advanced features like Common Table Expressions (CTEs), window functions, and full-text search.
- Extensibility and Customization
 - Users can create custom data types, operators, and functions using programming languages like PL/pgSQL, PL/Python, and PL/Perl.
 - Databases can be customized to integrate custom logic directly into the database environment.



KEY FEATURES

- Support for NoSQL Features:
 - Such as JSON and JSONB data types, Hstore, and key-value storage to handle both structured relational data and unstructured document-based data.
- Scalability and Performance Optimization:
 - Built-in tools and configurations for optimizing performance in large-scale applications
 - Indexing options (B-tree, GiST, GIN, etc.), partitioning, parallel query execution, and advanced caching mechanisms.



KEY FEATURES

- Strong Community and Ecosystem:
 - Rich ecosystem of tools, extensions, and third-party integrations.
- High Availability and Replication:
 - Options for replication and high availability
 - Including streaming replication, logical replication
 - Clustering solutions like Patroni and pgPool.
 - Supports enterprises that require minimal downtime and robust disaster recovery strategies.

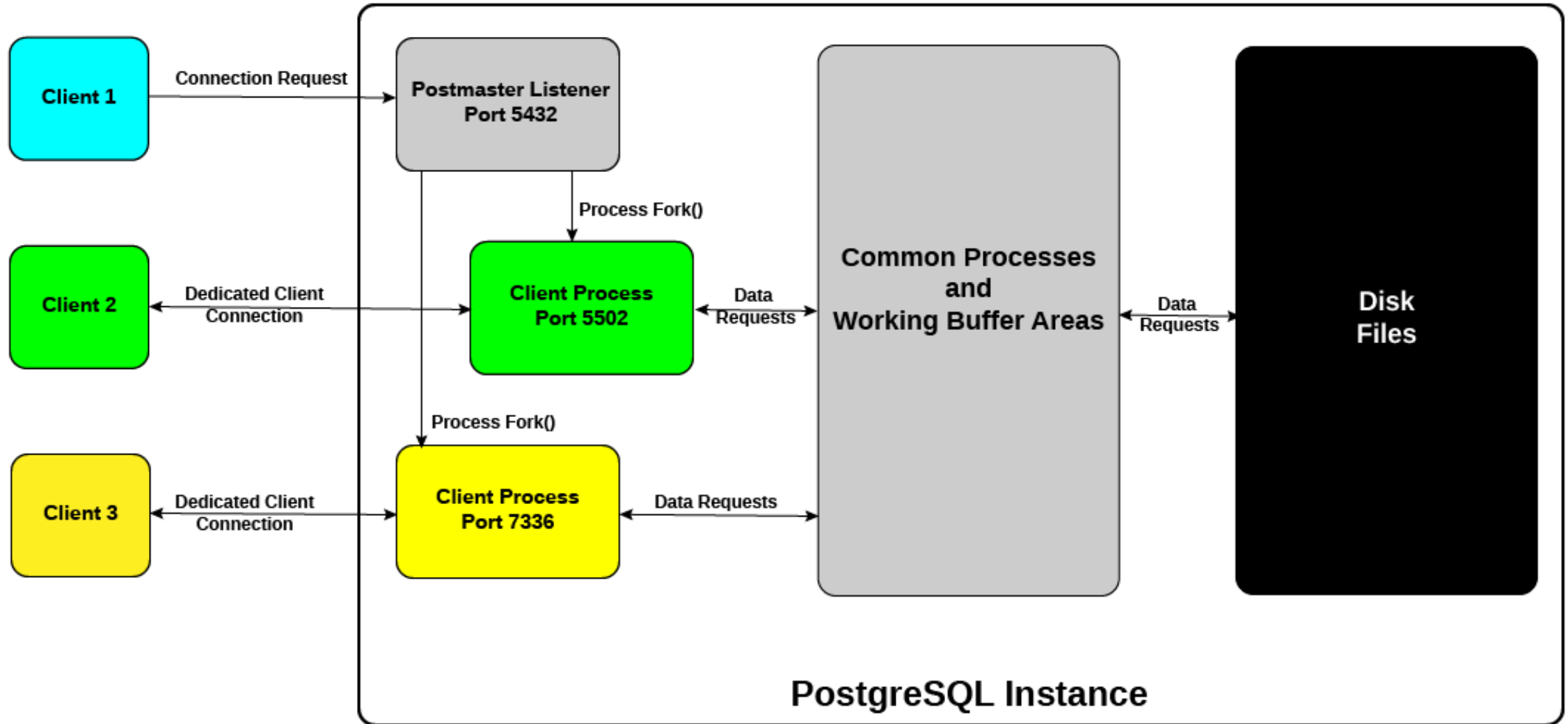


KEY FEATURES

- Cross-Platform Support and Portability:
 - Runs on all major operating systems
 - Has thorough, well-maintained documentation
 - Commercial support and consultancy services are also available from various vendors
- Compliance and Standards:
 - Complies with SQL standards
 - Integrates with various development environments, APIs, and middleware.
 - Supports ODBC, JDBC, and other common database interfaces



PROCESS ARCHITECTURE



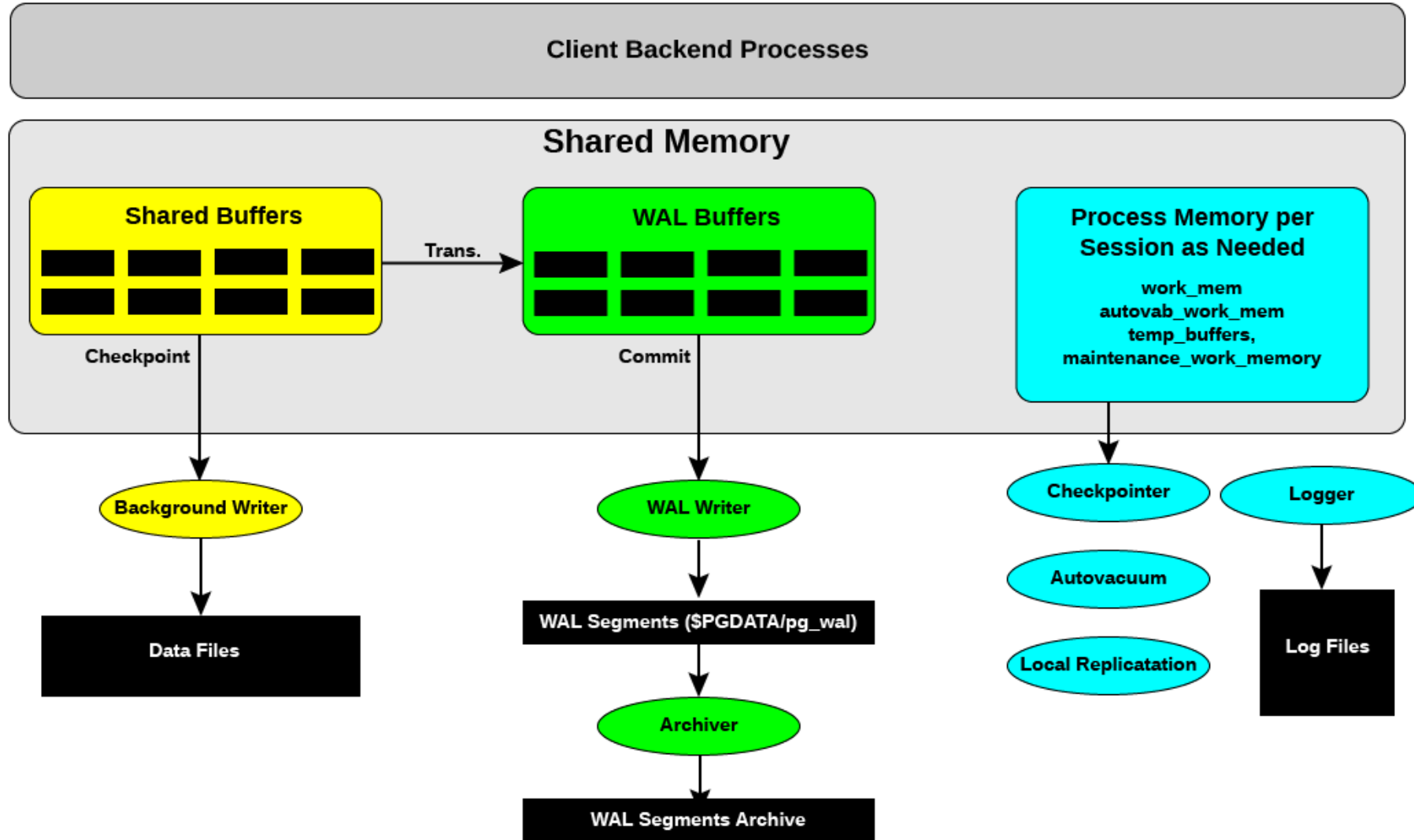
PROCESS ARCHITECTURE

- Postmaster is the initial process created when PostgreSQL starts
 - Acts as a primary process that monitors, starts and restarts various utility processes if they terminate for some reason.
 - Acts a listener on a specified port (5432 by default) for any new connection requests from the clients.
 - Authenticates and authorizes all incoming requests.
 - If the request is allowed, Postmaster spawns a new dedicated back end postgres process for the client.
- The back end client process handles parsing, planning, and executing queries received from the client.
 - Results of the queries are sent back to the client over the established connection.
 - Clients can initiate transactions, which are managed by the server using MVCC
 - *This allows multiple transactions to be processed concurrently while maintaining data integrity.*

SHARED MEMORY

- Clients do not read or write directly to disk.
 - Instead, clients work in shared data buffers
 - Utility processes are responsible for synchronizing the buffers and disk files
- The *Shared Memory Segments* are the buffer cache in memory reserved for transactions and maintenance activity.
 - There are different Shared Memory Segments allocated to different utility processes.

SHARED MEMORY



SHARED BUFFER

- The shared buffers segment is used for caching database pages that are read from disk
 - Allowing subsequent operations to be faster by accessing data directly in memory rather than on disk.
 - Crucial for performance, as it reduces the number of disk I/O operations needed.
 - The size of shared buffer is set by the *shared_buffer* parameter in the *postgresql.conf* file.
- This buffer is used for any insert, update, delete, or select operation.
 - Any data that is modified or updated is referred to as *dirty data* and is written to disk by the utility *writer* process
- Pages from tables and indexes are stored in the shared buffer
 - This memory area is shared by all the back-end client processes.

WAL BUFFER

- WAL (Write Ahead Log) buffers are an in-memory structure that temporarily stores WAL records before they are written to disk.
 - These records contain information about transactional changes made to the database, including inserts, updates, deletes, and schema modifications.
 - WAL records are generated by transactions, ensuring that changes can be replayed in case of a crash.
 - The WAL mechanism ensures that all changes to the database are logged before the changes are committed to the actual data files
- If a crash occurs, the WAL buffers ensure that all committed changes are safely stored in WAL files on disk, even if the changes have not yet been written to the main data files.
 - During recovery, WAL records are replayed to bring the database back to a consistent state, applying all changes that were committed before the crash.

OTHER BUFFERS

- **Commit Log** (CLOG) buffer, also known as the Transaction Status Log (pg_clog), keeps track of the status of transactions.
- The **lock table** is used to manage locks held by back end processes on database objects such as tables, rows, and indexes.
- **Work Memory** area is used for sort operations that includes Order By, Distinct, Merge join, and hash table operations that includes hash-join, hash-based aggregation, or when an *IN* clause is involved in the database SQL query
- **Shared Memory for Background Processes**: Various background processes, such as the autovacuum daemon, checkpointer, writer, and background writer, use shared memory segments to coordinate their activities and share information with backend processes.
- There are also a number of other specialized buffer areas that are beyond the scope of this introduction

UTILITY PROCESSES

- The main utility or background processes are:
- Bgwriter\Writer:
 - Responsible for periodically writing dirty pages (modified pages) from the shared buffers to the data files on disk.
 - Reduces the workload of the checkpoint process and smooths out disk I/O by continuously flushing modified pages in the background ensuring that sufficient clean buffers are available for new operations.
- Checkpoint:
 - Responsible for periodically writing all modified data pages (also known as "dirty" pages) from the shared buffers to the disk, along with updating the Write-Ahead Logging (WAL) records.
 - The writer process spreads the I/O load by incrementally writing dirty pages to disk, while the checkpoint process ensures data consistency and durability by writing all dirty pages at specific intervals.

UTILITY PROCESSES

- Auto-Vacuum:
 - Automatically cleans up dead tuples (deleted or updated rows) from tables and indexes to reclaim storage space and maintain database performance.
 - Helps prevent table bloat and ensures the visibility map and statistics are up-to-date, which optimizes query planning and execution.
- StatsCollector:
 - Responsible for collecting and maintaining statistics about database activity, including table access counts, index usage, and query performance metrics.
 - These statistics are used by the query planner to optimize query execution and by administrators to monitor database performance and usage.
- Logwriter\Logger:
 - Responsible for handling and writing log messages generated by the database system to log files.

UTILITY PROCESSES

- Archiver
 - Saves completed Write-Ahead Log (WAL) files to a secure, long-term archive storage location.
 - Ensures that all changes recorded in the WAL files are preserved, allowing for point-in-time recovery (PITR) and safeguarding against data loss in case of system failures.
- We will examine the file structure of a PostgreSQL instance in a later module.

PROCESSES

- The screenshot below shows the main PostgreSQL processes on an Ubuntu server instance

```
postgres 722 0.0 0.7 223000 30800 ? Ss 14:35 0:01 /usr/lib/postgresql/16/bin/postgres -D /var/lib/postgresql/16/main
postgres 4445 0.0 0.2 223136 8572 ? Ss 20:48 0:00 postgres: 16/main: checkpointer
postgres 4446 0.0 0.1 223000 6140 ? Ss 20:48 0:00 postgres: 16/main: background writer
postgres 4447 0.0 0.1 223000 5884 ? Ss 20:48 0:00 postgres: 16/main: walwriter
postgres 4448 0.0 0.2 224628 8572 ? Ss 20:48 0:00 postgres: 16/main: autovacuum launcher
postgres 4449 0.0 0.2 224576 8060 ? Ss 20:48 0:00 postgres: 16/main: logical replication launcher
ubuntu 4451 0.0 0.0 7080 2048 pts/2 S+ 20:48 0:00 grep --color=auto postgres
```

- And with client processes. Note the port numbers of the client processes

```
ubuntu@ip-172-31-24-118:~$ ps aux | grep postgres
postgres 722 0.0 0.7 223000 30800 ? Ss 14:35 0:01 /usr/lib/postgresql/16/bin/postgres -D /var/lib/postgresql/16/main
postgres 4445 0.0 0.2 223136 8572 ? Ss 20:48 0:00 postgres: 16/main: checkpointer
postgres 4446 0.0 0.1 223152 7292 ? Ss 20:48 0:00 postgres: 16/main: background writer
postgres 4447 0.0 0.2 223000 10108 ? Ss 20:48 0:00 postgres: 16/main: walwriter
postgres 4448 0.0 0.2 224628 8572 ? Ss 20:48 0:00 postgres: 16/main: autovacuum launcher
postgres 4449 0.0 0.2 224576 8060 ? Ss 20:48 0:00 postgres: 16/main: logical replication launcher
postgres 4452 0.1 0.5 227260 20644 ? Ss 20:48 0:00 postgres: 16/main: postgres postgres 98.111.240.238(53435) idle
postgres 4482 0.0 0.3 225092 14588 ? Ss 20:53 0:00 postgres: 16/main: std0 std0 174.115.137.145(57282) idle
ubuntu 4484 0.0 0.0 7080 2048 pts/2 S+ 20:53 0:00 grep --color=auto postgres
```

LAB 1

- The lab description and documentation is in the Lab directory in the class repository



End Module



PostgreSQL