How PostgreSQL replication works

1 - Intro

Have you ever heard about Eric Brewer's CAP theorem ? It says that it's impossible for a distributed system to provide both Consistency, Availability and Partition tolerance. There is one more theorem: PACELC which is a bit more complete than CAP, but I'd rather not spend too much time on theory.

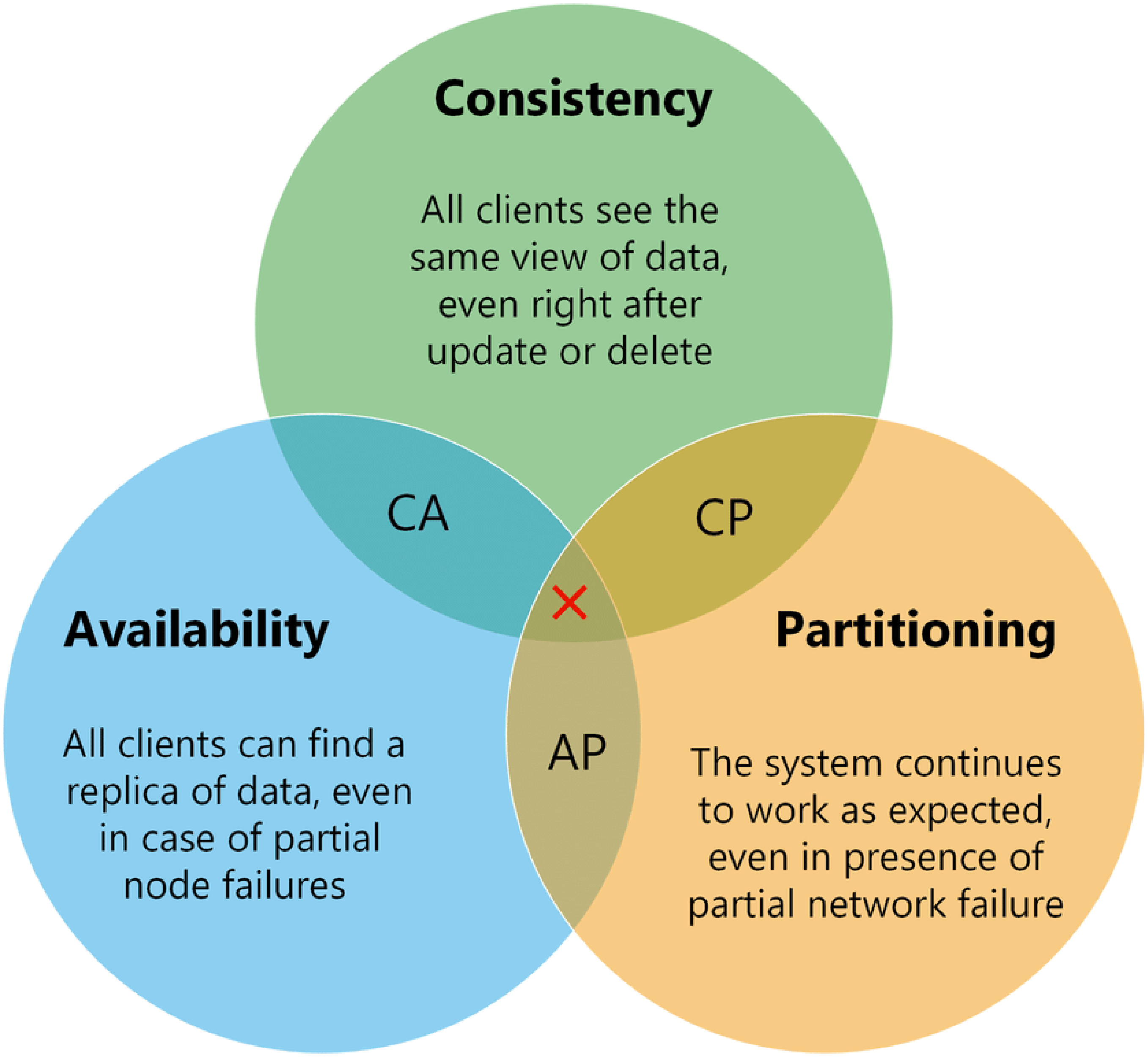

Consistency
All clients see the same view of data, even right after update or delete
CA
CP
✕
Availability
Partitioning
All clients can find a replica of data, even in case of partial node failures
AP
The system continues to work as expected, even in presence of partial network failure

Diagram by Hamzeh Khazaei from ResearchGate

PostgreSQL isn't immune of that and for what concerns Availability and Consistency, there are different solutions each of which with pros and cons.
Let's see what we mean with these three properties:

- We have **consistency** when all nodes in the cluster view the same data at the same time.
- **Availability** is the property to have at least one node in the cluster who can always serve writing and reading operations.
- **Partitioning** is a bit more tricky to define. Generally spoken, we have partitioning when something is going wrong in the cluster (node failures, network issues, ecc.). In a distributed environment partitioning is something that can always occur at some point in the time and we cannot avoid it.

The following is true for PostgreSQL 11, currently the latest available version on Amazon RDS.

2 - Glossary

First of all let’s clarify some terms:

- **Master** or **Primary** A server that can modify data.
- **Standby** or **Secondary** Servers that track changes in the master.
- **Warm Standby** A standby that cannot be connected to until it is promoted to a master server.
- **Hot Standby** A standby that can accept connections and serve read-only queries.
- **Failover** The process of switching to standby after an abnormal termination of the master server. [Learn more]
- **High Availability** The capability of a PostgreSQL cluster to ensure a higher up-time of a single node and the ability to work together with other nodes to ensure that there is always at least one node who can reply writes and queries.

3 - Failover

As we previously said, failover is the process of switching to standby after an abnormal termination of the master server. Many failover systems use to connect Master and Standby with some kind of heartbeat which tests connection and master liveness. However, PostgreSQL does not provide an official software component to identify a failure on the primary and notify the standby. This monitoring tools are often third party plugins, distributed as open source projects or provided by SaaS vendors.

Typically when a master database crashes the most up to date replica server is restarted as new master.

What if a master restarts after a new primary is promoted? There is a mechanism, known as STONITH (Shoot The Other Node In The Head), which avoids the situation where both systems think they are the primary.

4 - Configurations

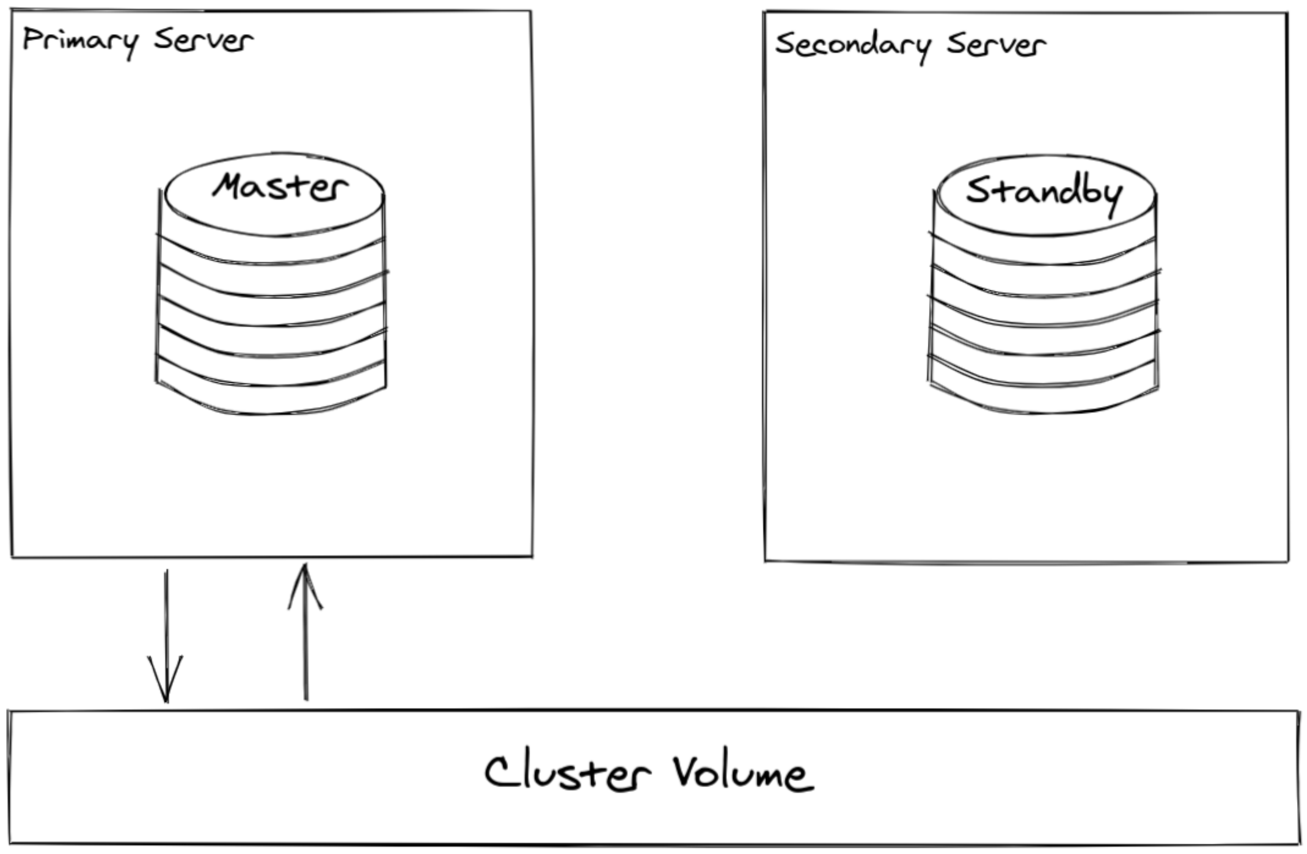
We can identify 4 main kinds of configurations:

1. File or disk based
2. Log shipping based or WAL based
3. SQL based
4. Commercial solutions and/or combined solutions

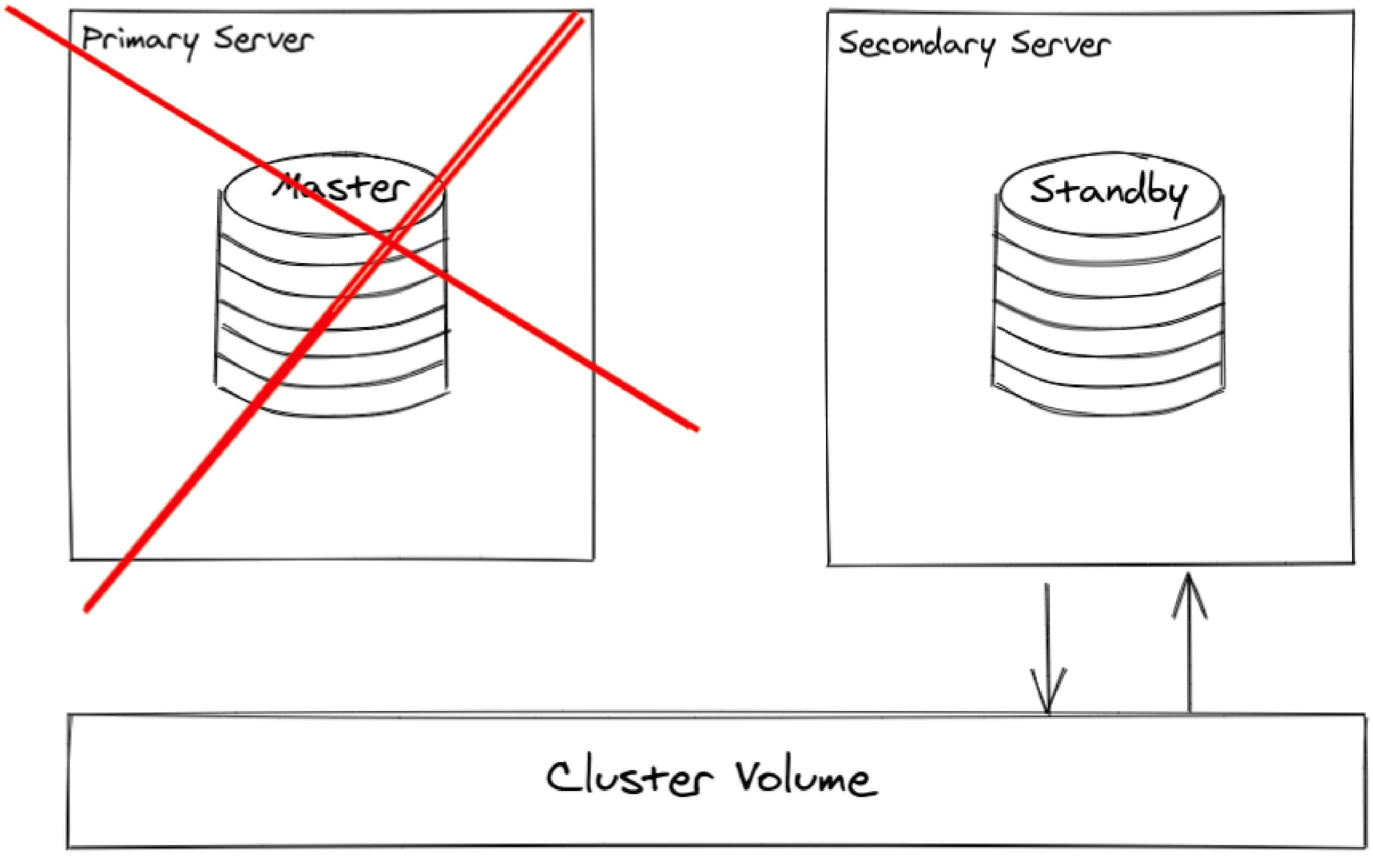
4.1 - File or disk based

These configurations are somehow physical replications since it happens at low or almost hardware level. In this category we have two main configurations: **Shared Disk** and **File System Replication (or Disk Mirroring)**.

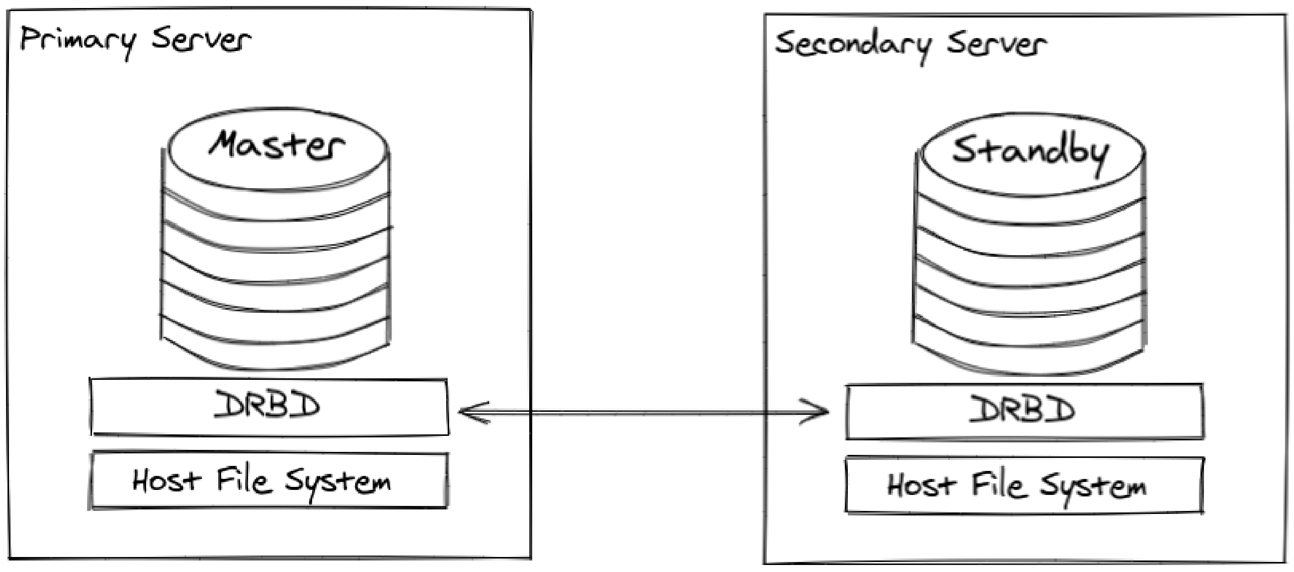
- **Shared Disk** uses a single disk array that is shared by multiple servers.

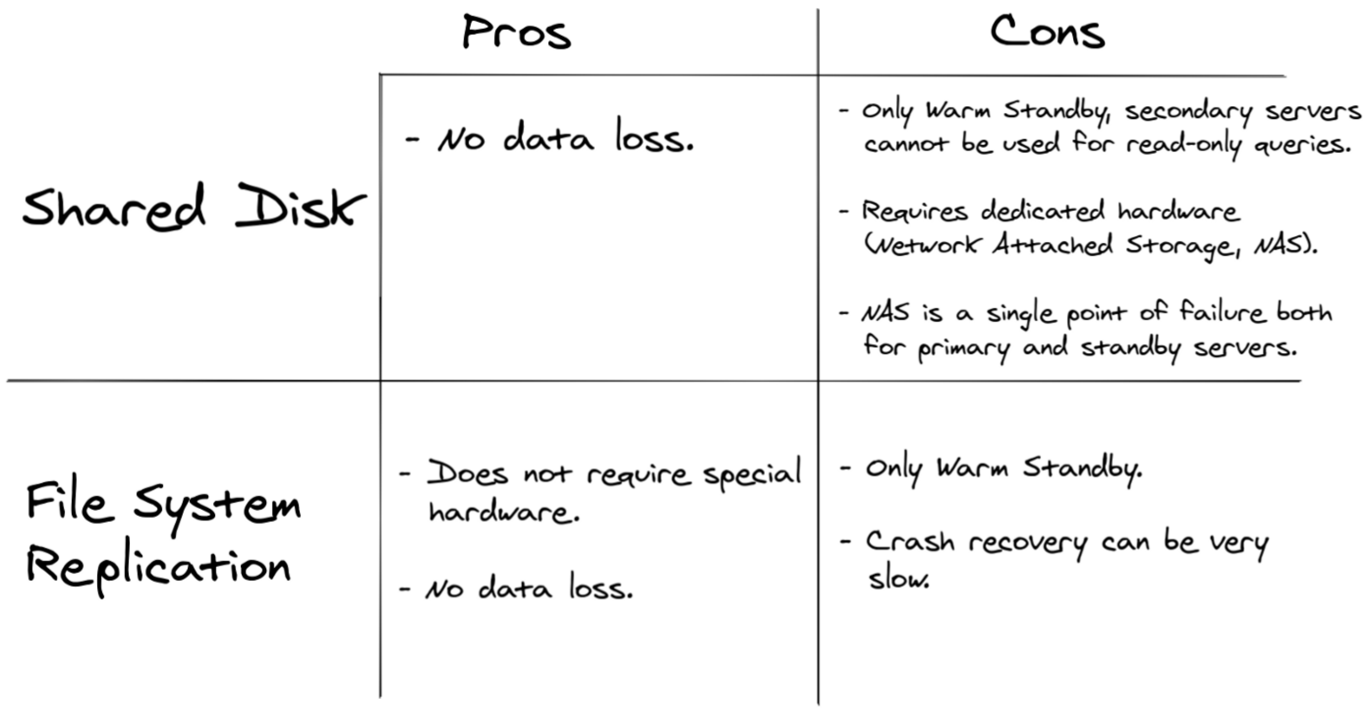


If the main database server fails, the standby server is able to mount and start the database as though it were recovering from a database crash. This allows rapid failover with no data loss.



- **File System Replication** is a common technique shared with many RDBMS. Disk Mirroring uses Distributed Replicated Block Device (DRBD), which is a distributed replicated storage system for Linux.



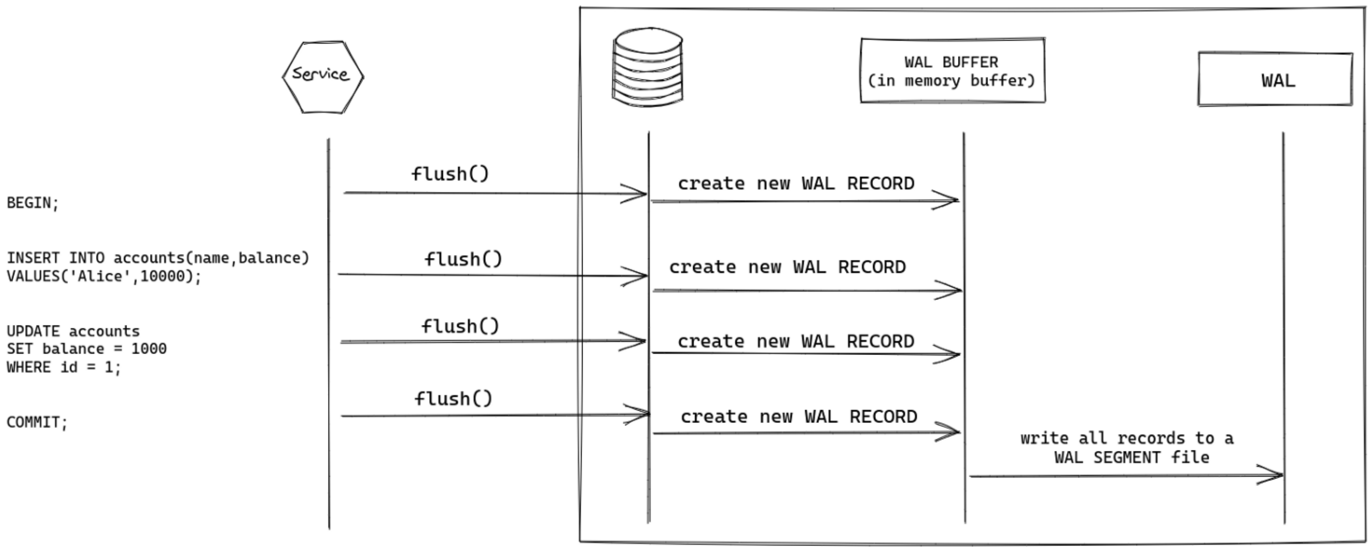


	Pros	Cons
Shared Disk	- No data loss.	- Only Warm Standby, secondary servers cannot be used for read-only queries. - Requires dedicated hardware (Network Attached Storage, NAS). - NAS is a single point of failure both for primary and standby servers.
File System Replication	- Does not require special hardware. - No data loss.	- Only Warm Standby. - Crash recovery can be very slow.

With these configurations, back to CAP theorem, we could say that we have *Consistency* and *Availability* when everything runs ok, but when things goes wrong (presence of partitioning) we keep only *Consistency*. Why we don't keep Availability ? Because there is, even if small, a downtime between the master failure and the new primary promotion.

4.2 - Log shipping based or WAL based

4.2.1 - WAL ? What ? Before we start talking of log shipping configurations, we have to say that every transaction in PostgreSQL is written to a transaction log called Write-Ahead Log (WAL).
Let's imagine we have a service which starts a transaction and for every new statement it flushes it to PostgreSQL, here what happens on the database



```
BEGIN;

INSERT INTO accounts(name,balance)
VALUES('Alice',10000);

UPDATE accounts
SET balance = 1000
WHERE id = 1;

COMMIT;
```

A WAL record's Log Sequence Number (LSN) represents the location/position where the record is saved in the log file.

So WAL is made up of *segments* files of 16MB by default and each segment has one or more *records*. *Log Sequence Number* (LSN) is a pointer to a specific record in WAL.

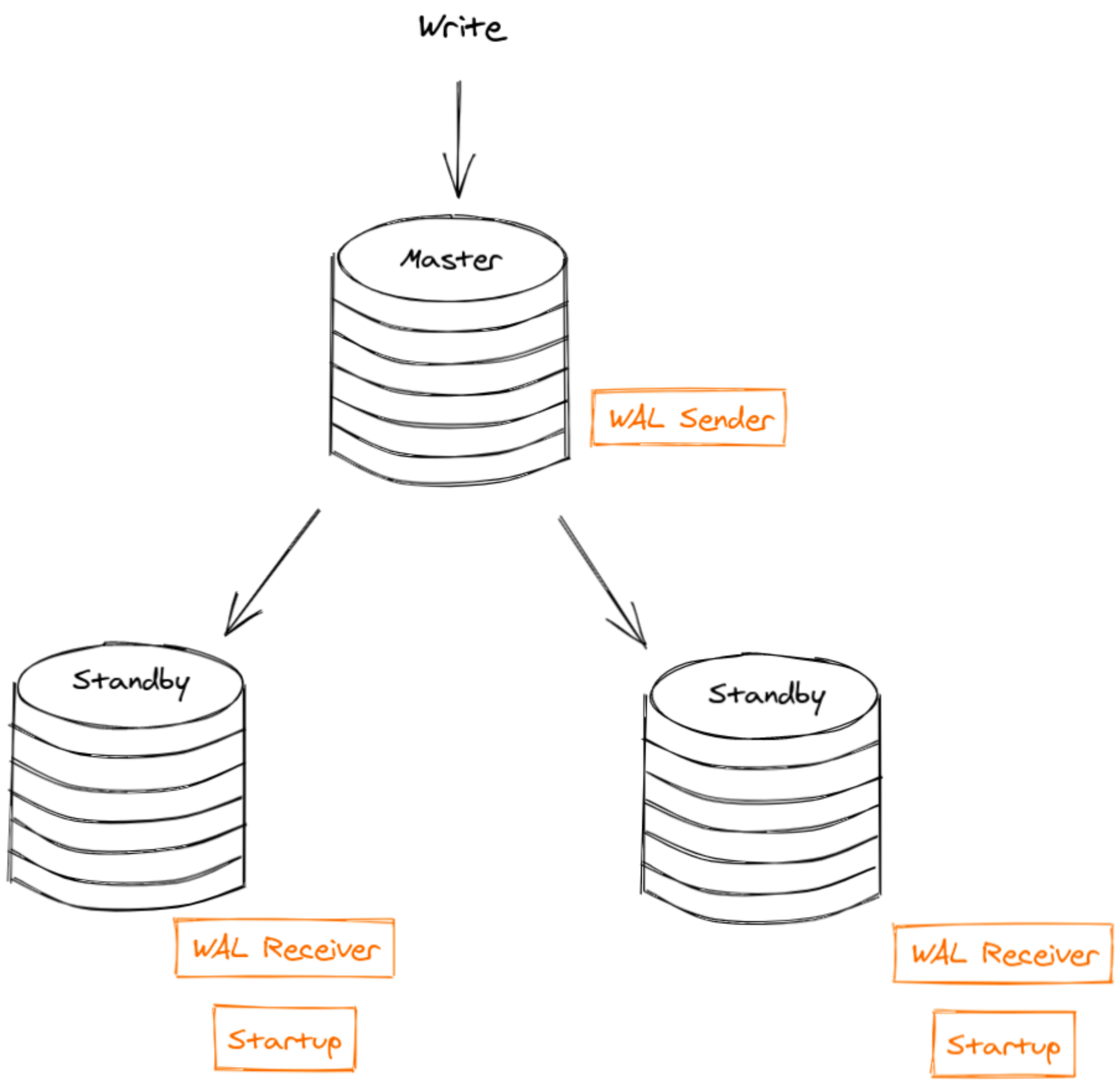
```
postgres@f0bc03174671:~/11/main/pg_wal$ ls -l --block-size=M
total 33M
postgres postgres 16M [...] 000000010000000000000001
postgres postgres 16M [...] 000000010000000000000002
postgres postgres  1M [...] archive_status
postgres@f0bc03174671:~/11/main/pg_wal$
```

4.2.2 - WAL, why ?

A standby server uses WAL segments (XLOGS in PostgreSQL terminology) to continuously replicate changes from its master.
Write-ahead logging is used to grant atomicity and durability in a DBMS by serializing chunks of byte-array data (each of one with an LSN) to a stable storage before they are applied to the database.

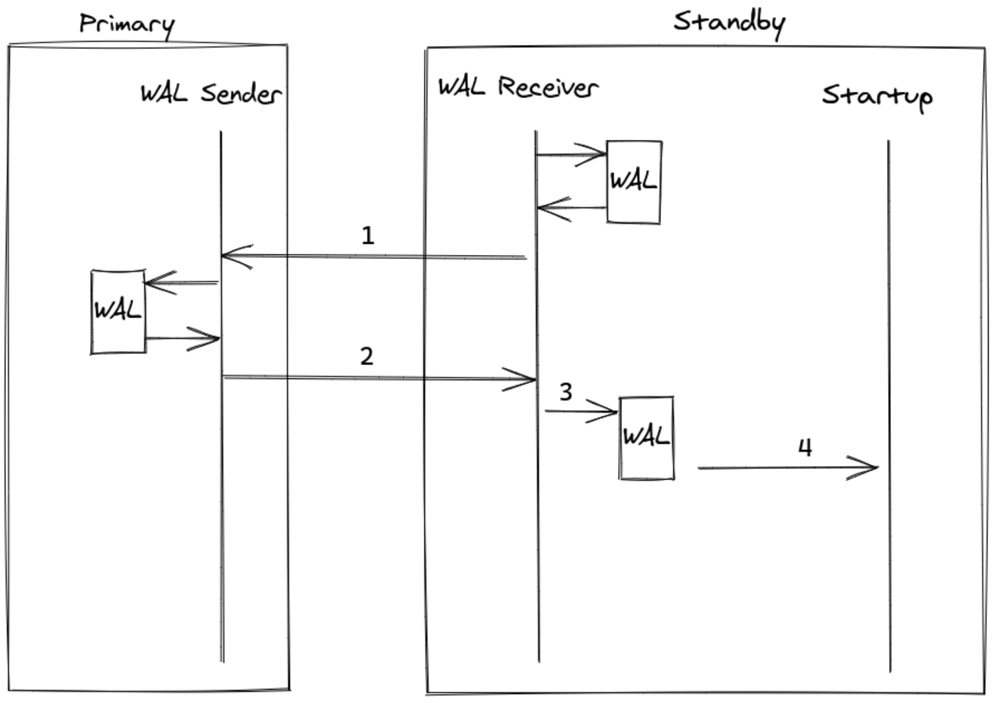
Applying a mutation to a database can result in many file system operations. How a database can then guarantee atomicity if for example the server stop working due to a power outage while it was in the middle of a file system updating? When a database boots, it starts a *replay* or *startup* process which reads the available WAL segments and compares them with the LSN stored in each data page (each data page is marked with the LSN of the latest WAL record affecting the page).
As the PostgreSQL doc says:

> *During WAL replay, we can check the LSN of a page to detect whether the change recorded by the current log entry is already applied (it has been, if the page LSN is >= the log entry's WAL location).*



WAL Sender is a process in master server which is responsible of sending segments to a receiver.
WAL Receiver is a process, in standby server, responsible of updating WAL segments as soon as it receives updates from a WAL Sender.
Startup process, in standby servers, loads transactions from WAL to database process, which then updates the database file system.

4.2.3 - What happens when a standby reboots after maintenance or crash ?In the following schema we can see a schematic description



1) WAL Receiver process sends the LSN (Log Sequence Number) up until when the WAL data has been replayed on a slave to the master.

2) WAL Sender process on master sends the WAL data from the LSN sent by the WAL receiver

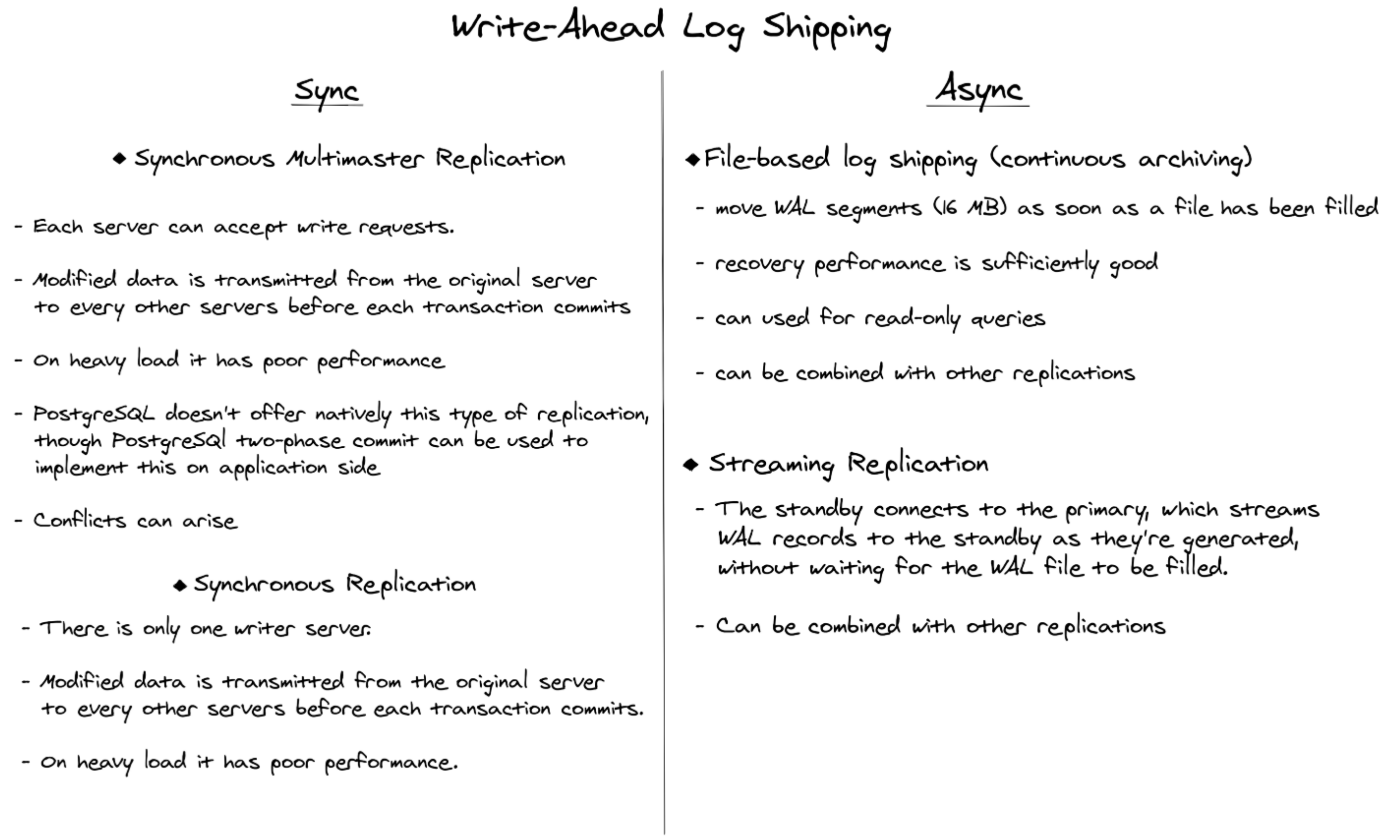
3) Wal Receiver writes the WAL data sent by WAL sender to WAL segments.

4) The Startup process on standby replays the data written to WAL segment.

Once we have an idea on what's a WAL, we can finally start talking about the Write-Ahead Log Shipping configurations.

4.2.4 - Log based configurations

- In **Synchronous Multi-master Replication** each server can accept write requests and modified data is transmitted from the original server to every other server before each transaction commits. It uses the 2PC protocol and follows the all-or-none rule.
- **Synchronous Replication** differs from the previous configuration by the presence of a single master: before each transaction commits, the master server waits until standbys confirm they received data. The advantage of this configuration is that there won't be conflicts due to simultaneous writing.
- **File-based log shipping (or continuous archiving)** is an async configuration which moves WAL segments as soon as a file of 16 MB has been filled.
- **Streaming Replication** is an async configuration where master server streams data to secondary as they are generated, without waiting for WAL segments to be filled. It allows to reduce the replication lag between primary and secondary.



Write-Ahead Log Shipping

Sync

◆ Synchronous Multimaster Replication

- Each server can accept write requests.
- Modified data is transmitted from the original server to every other servers before each transaction commits
- On heavy load it has poor performance
- PostgreSQL doesn't offer natively this type of replication, though PostgreSQl two-phase commit can be used to implement this on application side
- Conflicts can arise

◆ Synchronous Replication

- There is only one writer server.
- Modified data is transmitted from the original server to every other servers before each transaction commits.
- On heavy load it has poor performance.

Async

◆ File-based log shipping (continuous archiving)

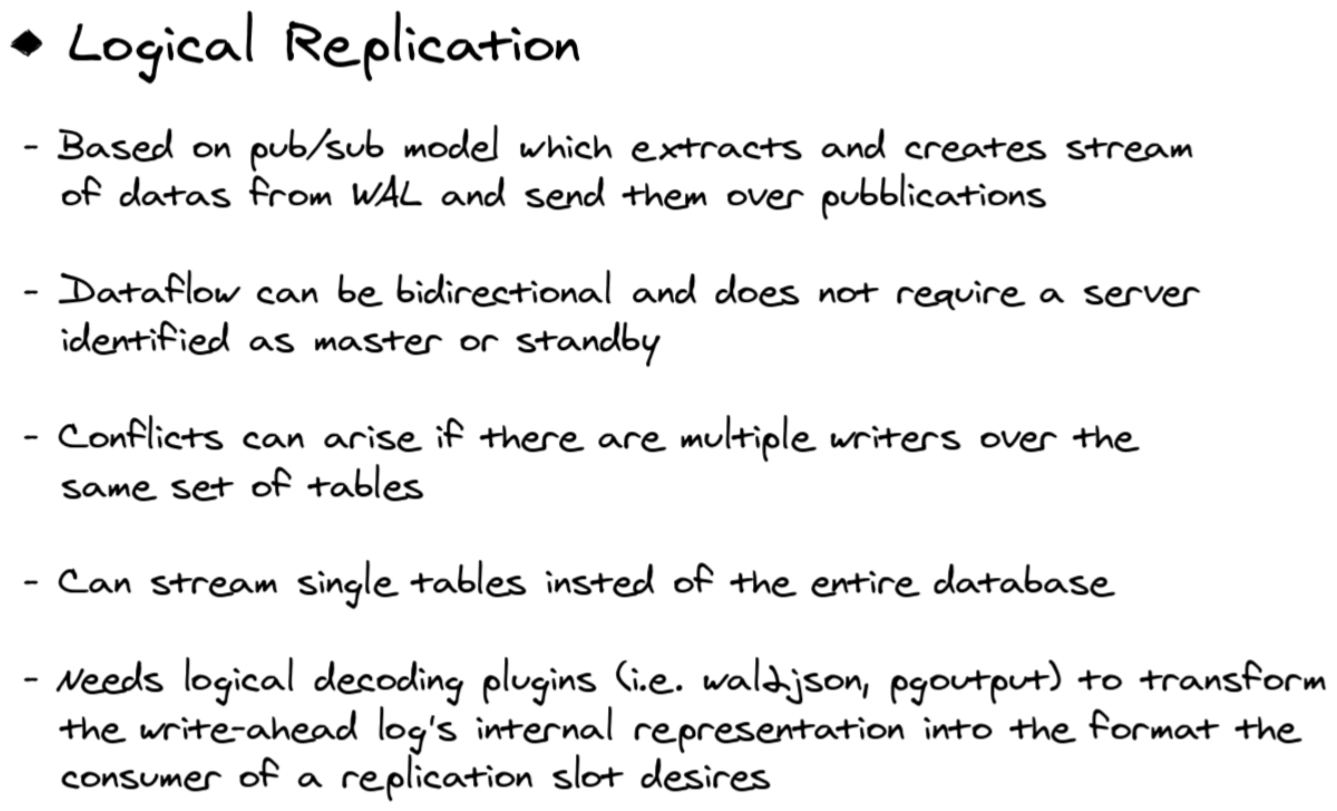
- move WAL segments (16 MB) as soon as a file has been filled
- recovery performance is sufficiently good
- can used for read-only queries
- can be combined with other replications

◆ Streaming Replication

- The standby connects to the primary, which streams WAL records to the standby as they're generated, without waiting for the WAL file to be filled.
- Can be combined with other replications

One obvious thing is that with synchronous replication there is no replication lag, while with asynchronous replication a replication lag is possible and there could be data loss in the case of master failure. The last thing to say about this approach is that Write-Ahead Log shipping can only be done for the entire database server and it's not allowed to choose a subset of tables or schema.

There is one more configuration under this category and it's called **Logical Replication**. It's based on WAL too, but it uses a quite different model since there is no Sender/Receiver process involved. It uses a pub/sub model which extracts, encodes and streams data from WAL. These data are then sent over publications.
Logical Replication can also be combined with WAL shipping methods (i.e. with synchronous replication to avoid data loss).



- Logical Replication
  - Based on pub/sub model which extracts and creates stream of datas from WAL and send them over pubblications
  - Dataflow can be bidirectional and does not require a server identified as master or standby
  - Conflicts can arise if there are multiple writers over the same set of tables
  - Can stream single tables insted of the entire database
  - Needs logical decoding plugins (i.e. wal2json, pgoutput) to transform the write-ahead log's internal representation into the format the consumer of a replication slot desires

Analyzing what we said about the previous configurations under the CAP point of view, we can say that:

- Synchronous approaches favour consistency over availability when partition occurs.
- Asynchronous approaches favour availability over consistency when partition occurs.

4.3 - SQL based

SQL based configuration, basically works with a program which intercepts all SQL mutation queries and sends them to a standby server which operates independently.

If queries are broadcast unmodified, functions like `random()`, `CURRENT_TIMESTAMP`, and sequences can have different values on different servers. This can be avoided by configuring the interceptor program to query values from the source database and replace that in the SQL statement before broadcasting it.

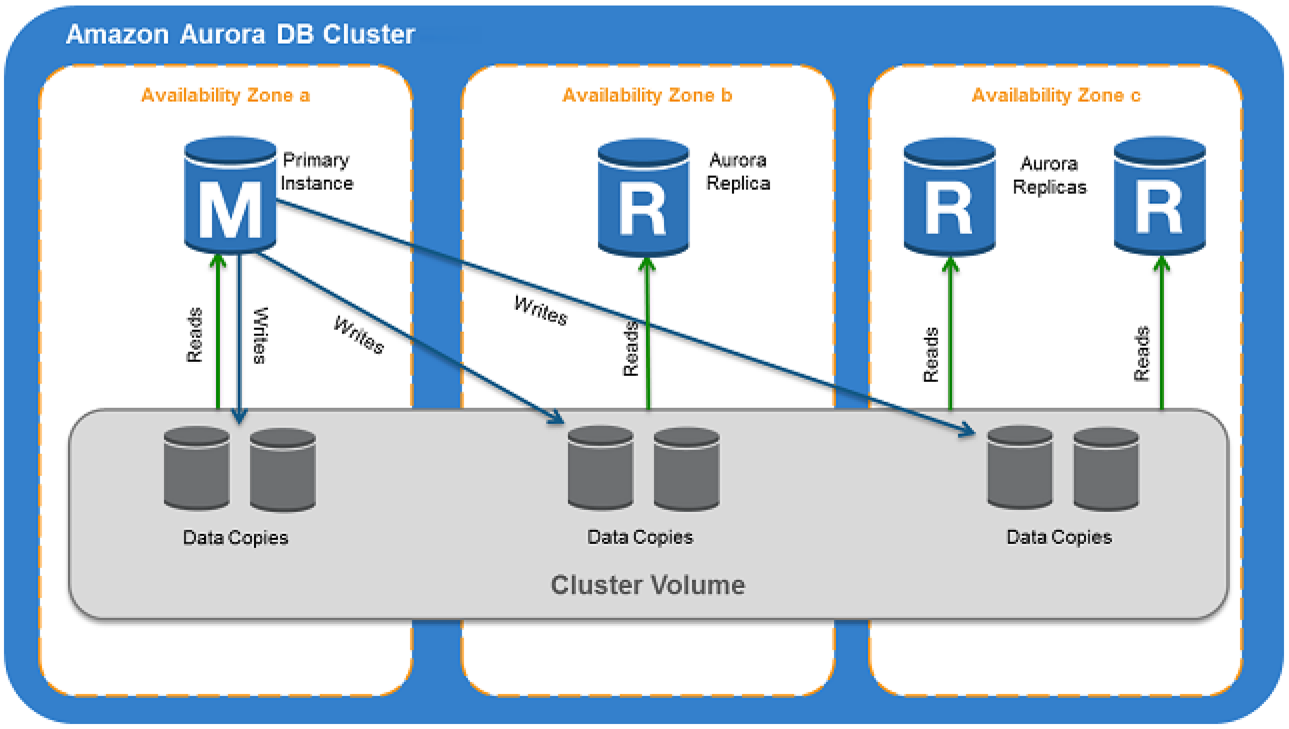
4.4 - Commercial solutions

On the last stand we have the commercial solutions, which are solutions developed starting from the PostgreSQL open source project.

In the project I'm currently working on at Moveax, we chose *Amazon Aurora DB clusters* as relational database to back our micro-services environment. Aurora has a single master replication configuration, made up of:

- Primary server which supports read and write operations.
- Aurora replicas, which are hot standby servers deployed in different availability zones

Furthermore, Aurora uses a distributed cluster volume located in different availability zones. Physically the underlying storage is made up of SSDs and the replication lag is often below the 100 milliseconds.



In conclusion on our brief tour of Postgres replication, we have to say that there are other replication implementations that may be described in another post.