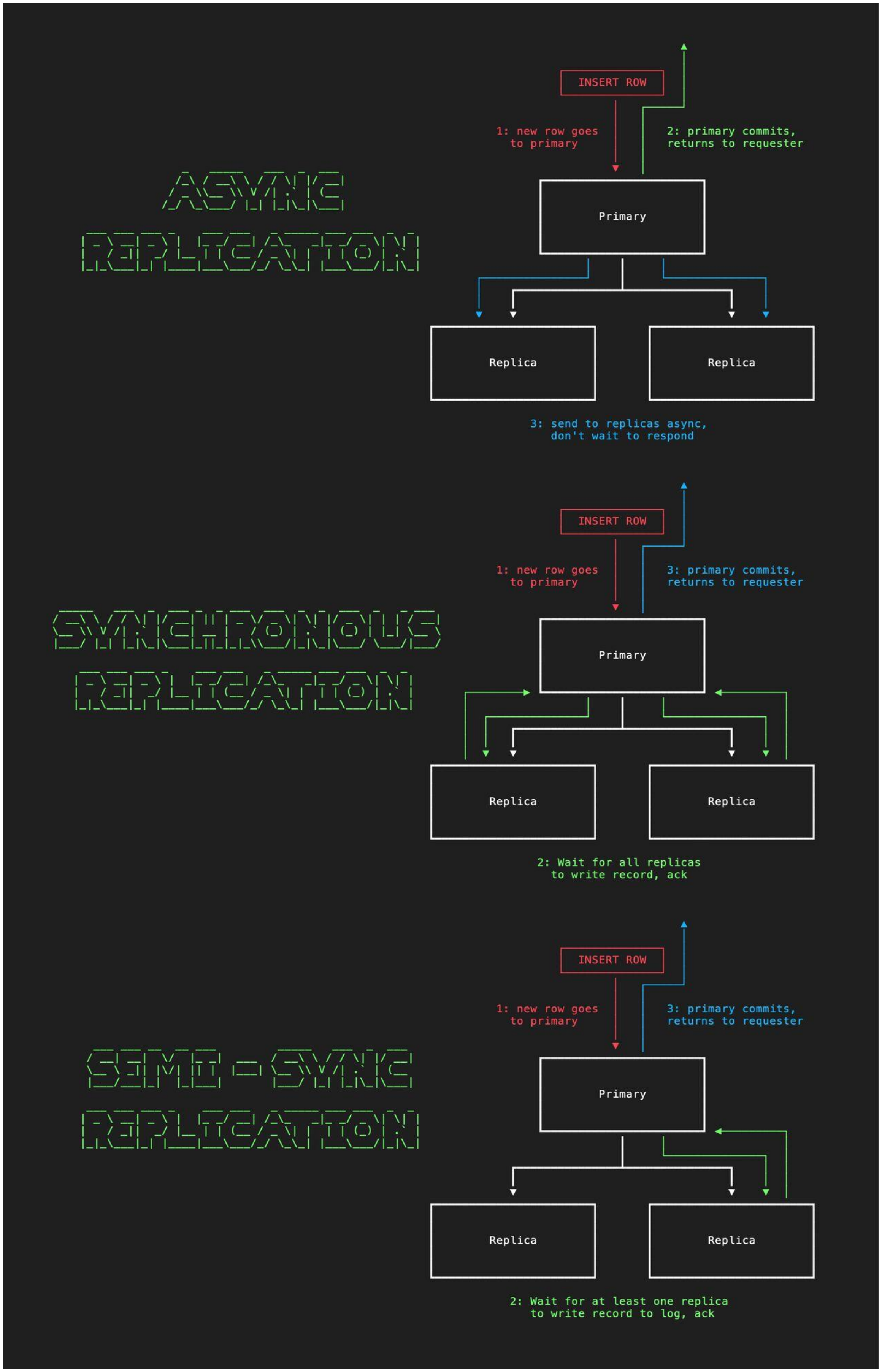

ASYNC REPLICATION
INSERT ROW
1: new row goes to primary
2: primary commits, returns to requester
Primary
Replica
Replica
3: send to replicas async, don't wait to respond
SYNCHRONOUS REPLICATION
INSERT ROW
1: new row goes to primary
3: primary commits, returns to requester
Primary
Replica
Replica
2: Wait for all replicas to write record, ack
SEMI-SYNC REPLICATION
INSERT ROW
1: new row goes to primary
3: primary commits, returns to requester
Primary
Replica
Replica
2: Wait for at least one replica to write record to log, ack

Synchronous vs Asynchronous Replication in PostgreSQL

PostgreSQL supports two main types of streaming replication modes: asynchronous and synchronous. Both have different trade-offs between performance and data safety. Below is a detailed explanation.

1. Asynchronous Replication

Flow:
1. Primary writes the transaction to WAL.
2. WAL is sent to the replica.
3. Primary does NOT wait for the replica to acknowledge before committing.
4. The client sees the commit as successful immediately.

Pros:
✅Faster commits.
✅Primary is not slowed by slow replicas.

Cons:
❌Data loss possible if primary crashes before the replica receives the latest WAL.

2. Synchronous Replication

Flow:
1. Primary writes the transaction to WAL.
2. WAL is sent to the replica.
3. Primary waits for the replica to write WAL to disk and acknowledge.
4. Only then does the primary commit and respond to the client.

Pros:
✅Data loss is minimal — replica always has all committed data.

Cons:
❌Slower commits because the primary waits for network + replica write.
❌If replica is slow or down, primary waits.

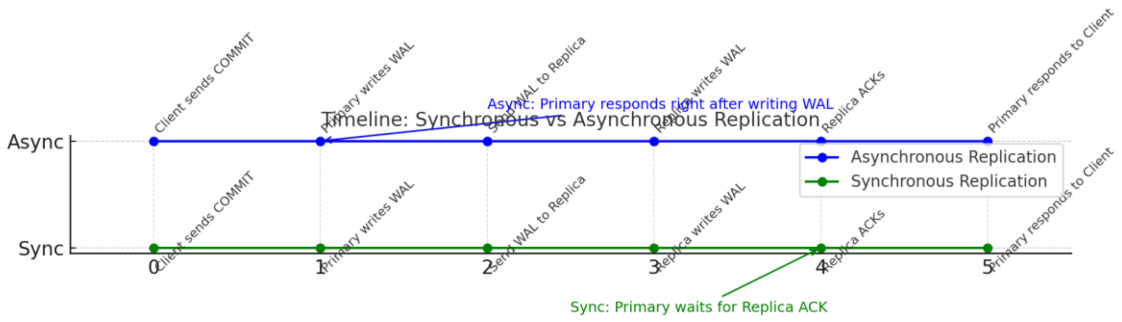
3. Quick Comparison Table

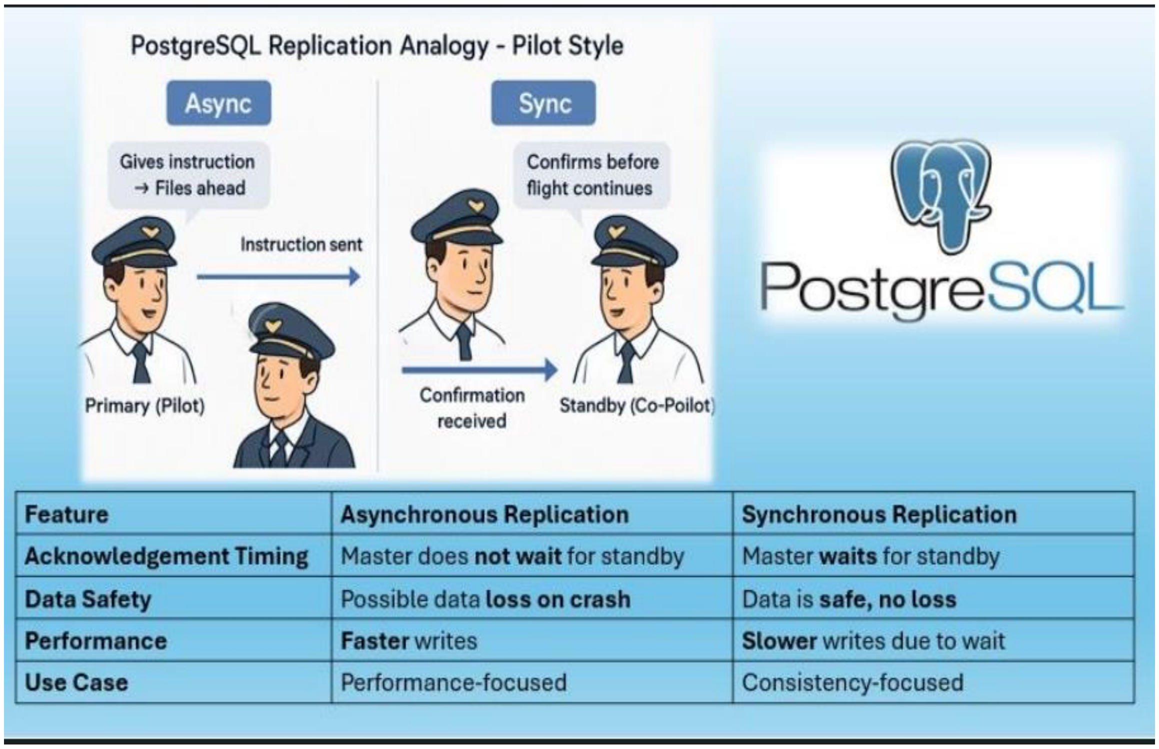
Feature	Asynchronous	Synchronous

Commit speed	Fast	Slower
Data loss risk on primary crash	Possible	Minimal
Primary waits for replica ack	No	Yes
Best for	Performance priority	Data consistency priority

4. Timeline Diagram

The diagram below illustrates when the primary responds to the client in each mode:




PostgreSQL Replication Analogy - Pilot Style
Async
Gives instruction → Files ahead
Instruction sent
Primary (Pilot)
Sync
Confirms before flight continues
Confirmation received
Standby (Co-Poilot)
PostgreSQL

Feature	Asynchronous Replication	Synchronous Replication
Acknowledgement Timing	Master does **not wait** for standby	Master **waits** for standby
Data Safety	Possible data **loss on crash**	Data is **safe, no loss**
Performance	**Faster** writes	**Slower** writes due to wait
Use Case	Performance-focused	Consistency-focused

PostgreSQL Replication — Explained with a Pilot Analogy

When it comes to PostgreSQL replication, we often hear: Synchronous vs Asynchronous Replication — but what does it really mean? Let's break it down with a flight analogy. Imagine you're a pilot (Primary DB server), and you have one or more co-pilots (Standby servers) you're communicating with during flight (database operations).

1. **Asynchronous Replication = Send and Fly**

- You (Pilot) give an instruction to the co-pilot via radio.
- As soon as you say it, you assume the co-pilot received it and move on.
- You don't wait to hear back.
- If your radio fails (Primary crash), maybe the co-pilot didn't catch your last instruction.

--> Fast and efficient, but risky.

--> Data might be lost if crash happens before co-pilot records it.

2. **Synchronous Replication = Confirm Before Flying Further**

- You give an instruction to the co-pilot.
- You wait for the co-pilot to repeat it back to confirm they heard it.
- Only after you hear the confirmation, you move on.

--> Slower, but safer.

--> You’re sure the co-pilot has the same latest instructions even if you crash.

Asynchronous:

COMMIT on primary returns immediately after writing to WAL, and sends to replica in the background.

--> Data may not be on standby if crash occurs.

Synchronous:

COMMIT waits until at least one standby confirms the data is written to its WAL.

--> Ensures no data loss, but can slow down writes.

Choose your replication strategy like a pilot:

Know when to fly fast and when to fly safe.