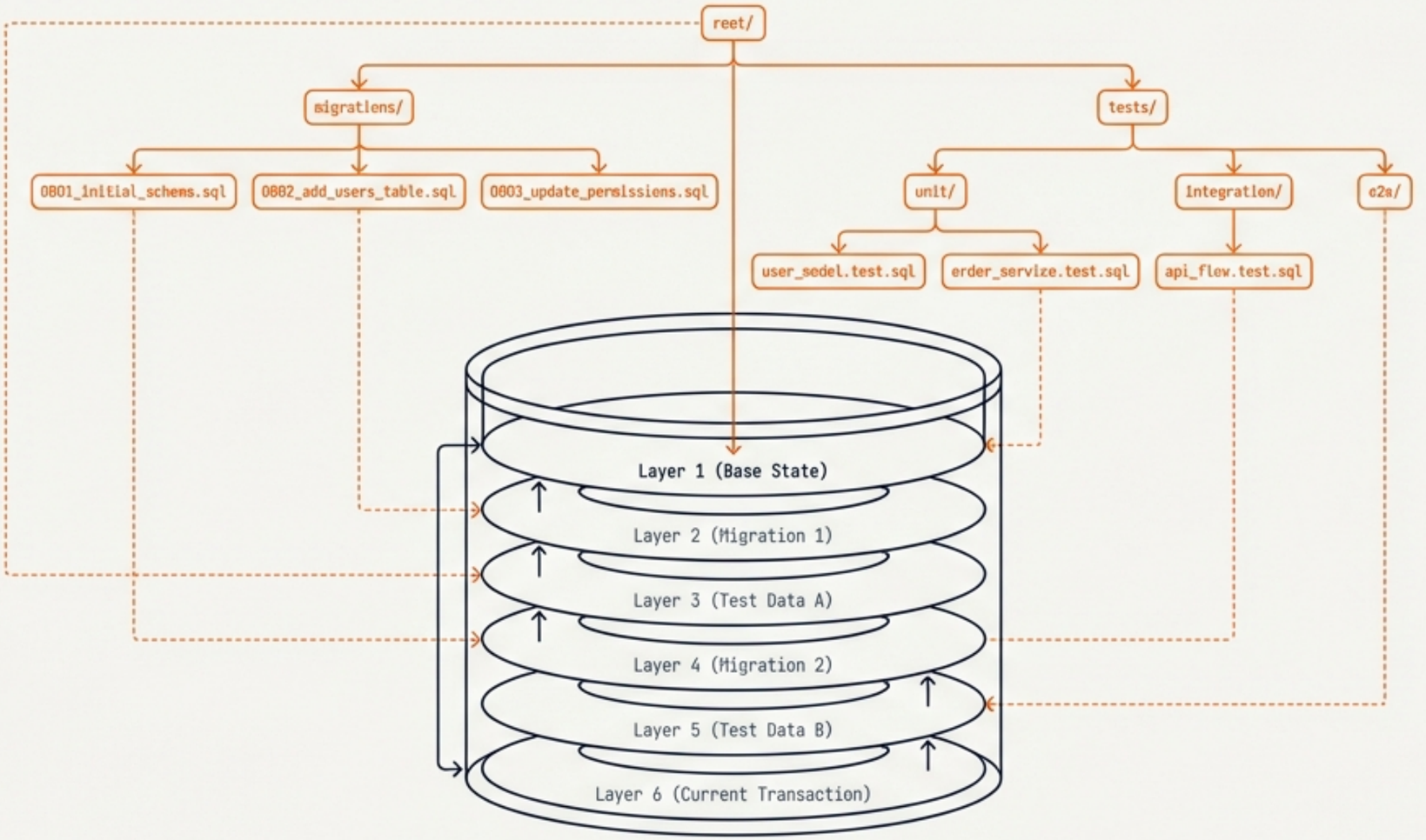
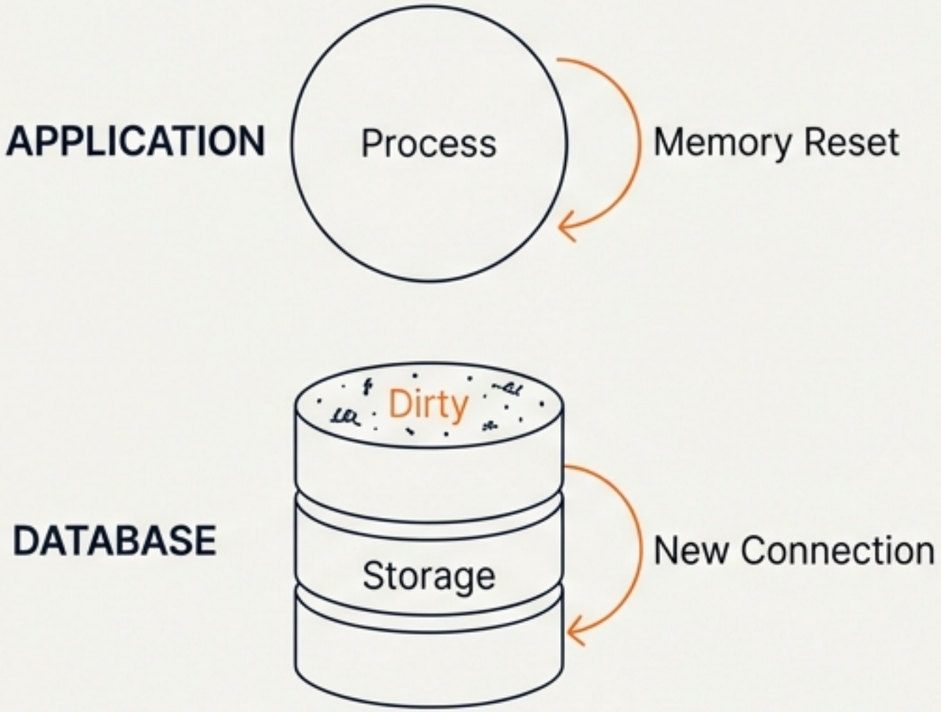
pgmi: Transactional Testing & Deployment Architecture

A first-principles analysis of zero-side-effect database testing using PostgreSQL primitives



THE PERSISTENCE PROBLEM: WHY DATABASE TESTING IS HISTORICALLY PAINFUL

THE CORE TENSION



Applications reset state on restart. Databases persist state by design. This persistence causes "residue" that contaminates tests.

THE CURRENT LANDSCAPE

	Pros	Cons
Containers (Docker)	Perfect Isolation	High Latency (2-5s startup). Disconnects testing from deployment.
Truncation / Cleanup	Fast Execution	Fragile Maintenance. Cascading deletes are messy. Requires perfect teardown scripts.
ORM Transactions	Easy Integration	App-layer only. Cannot test DDL (schema changes). Ignores DB-internal logic.

THE INSIGHT: WHAT IF THE DEPLOYMENT TOOL DIDN'T MANAGE THE TRANSACTION, BUT THE SCRIPT DID?

THE PRIMITIVES: WHY THIS ONLY WORKS IN POSTGRESQL

pgmi is an orchestration layer for these native capabilities.

TRANSACTIONAL DDL

Unlike MySQL or Oracle, PostgreSQL allows `CREATE TABLE` and `ALTER TABLE` to be rolled back.

```
BEGIN;
DROP TABLE users; -- Oops
ROLLBACK; -- Table users is restored
```

SAVEPOINTS

Named markers allow nested rollback scopes within a single transaction without aborting the whole operation.

```
SAVEPOINT test_start;
INSERT INTO ...;
ROLLBACK TO SAVEPOINT test_start;
```

SESSION-SCOPED TABLES

`pg_temp` schema visibility is restricted to the current connection. Data vanishes on disconnect.

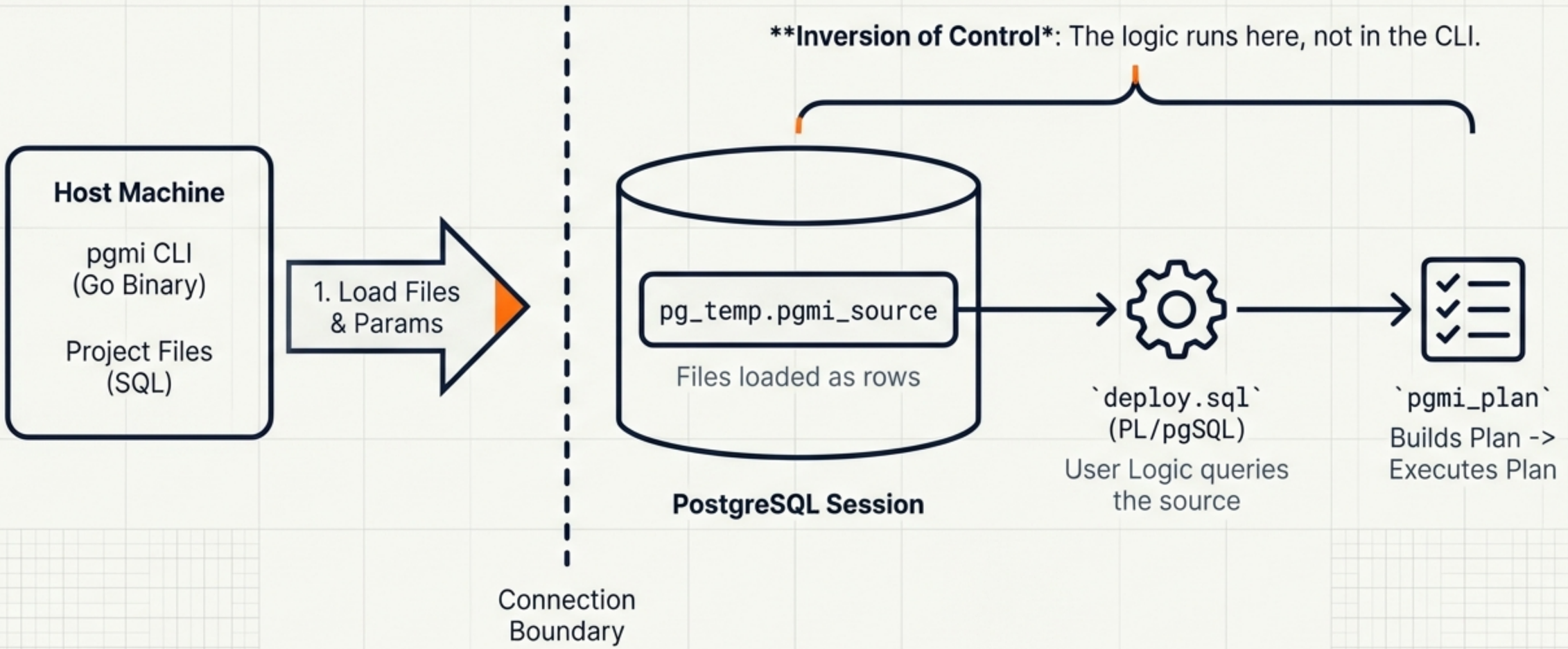
```
CREATE TEMP TABLE pgmi_source (...)
ON COMMIT DROP;
```

PL/pgSQL

Turing-complete procedural language allows loops, conditionals, and exception handling inside the DB.

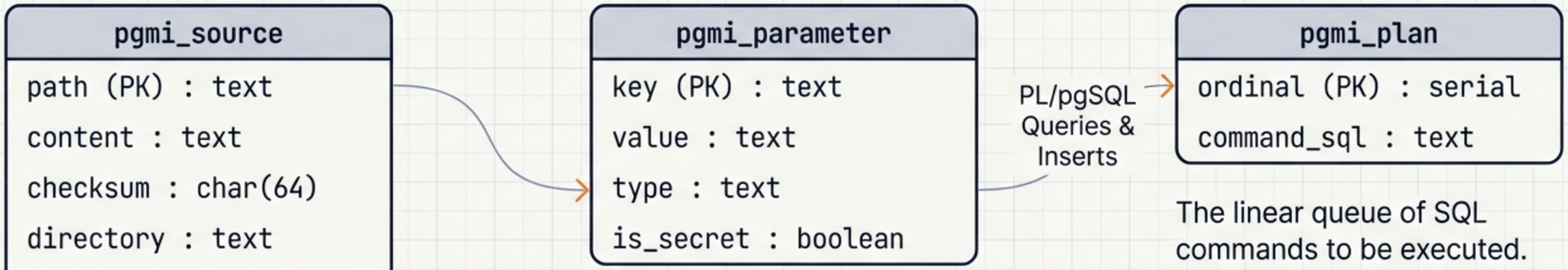
```
DO $$
BEGIN
  IF exists THEN ... END IF;
END $$;
```

Architecture: PostgreSQL as the Deployment Engine



The Session API: The Plan is Data

Deployment state is exposed as standard SQL tables in `pg_temp`.



Contains the raw project files loaded from disk.

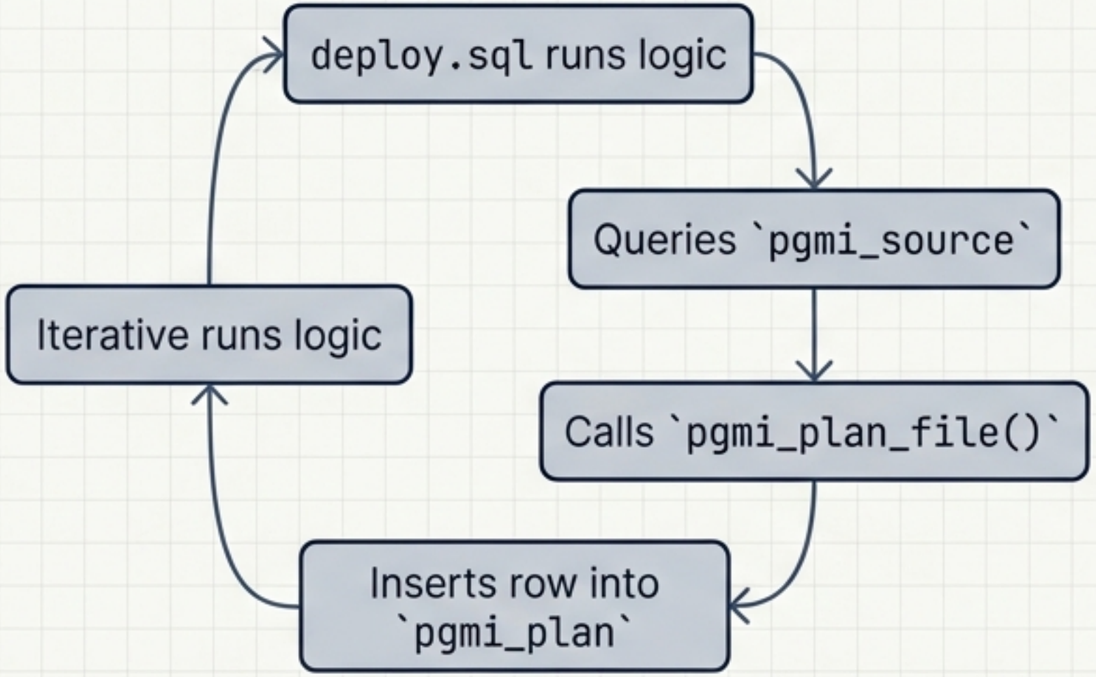
Strongly typed CLI arguments injected into the session.

The linear queue of SQL commands to be executed.

```
-- Example: Dynamic Plan Construction
INSERT INTO pgmi_plan (command_sql)
  SELECT content FROM pgmi_source
  WHERE directory = './migrations/'
  ORDER BY name;
```

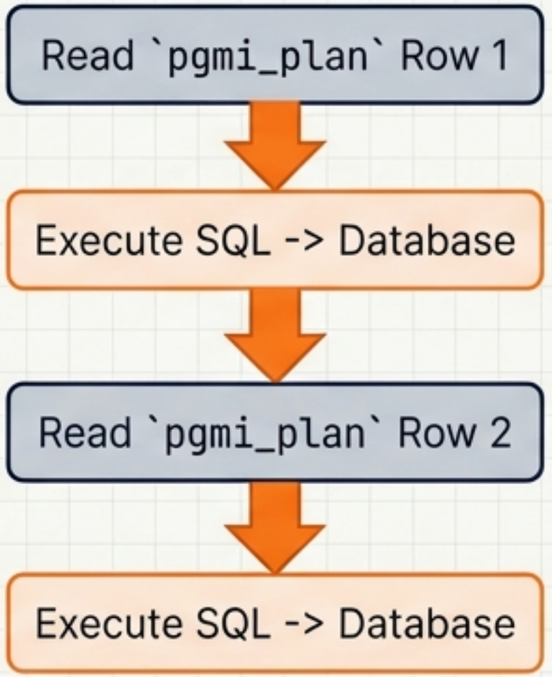
The Two-Phase Execution Model

Phase 1: Planning (PL/pgSQL)



No database changes yet. Just data entry.

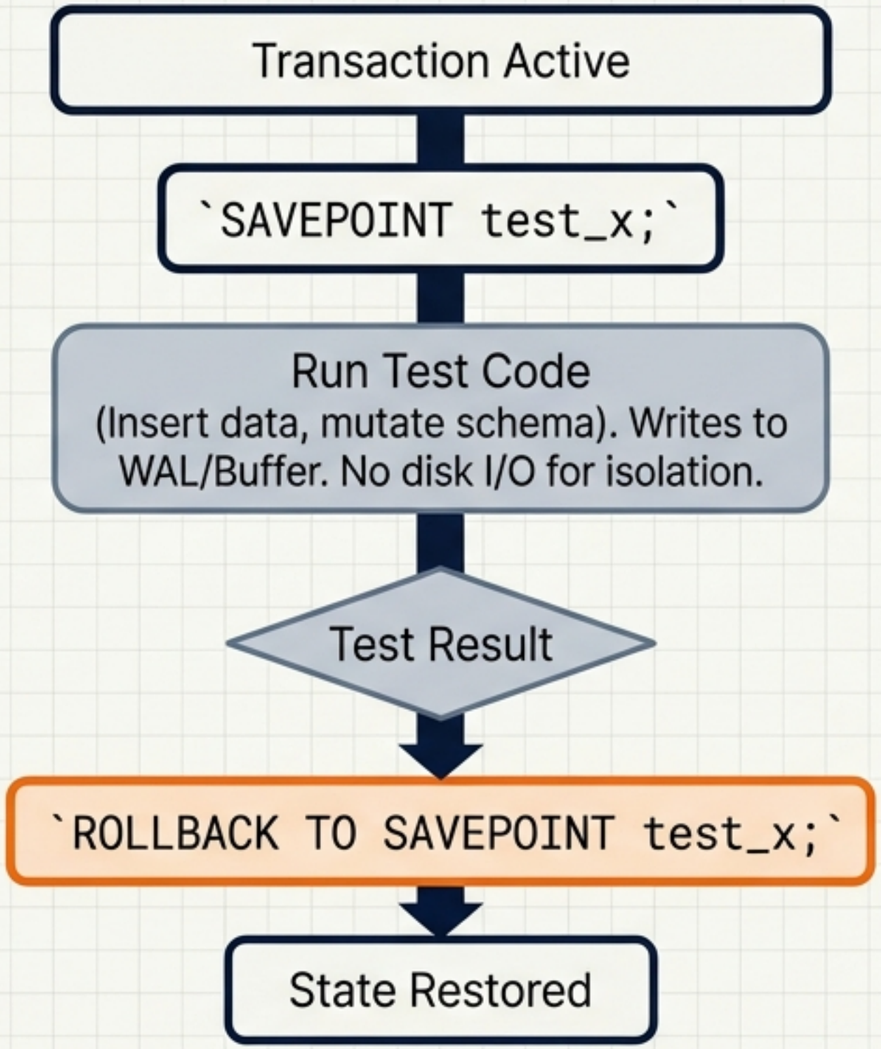
Phase 2: Execution (Go Binary)



Dumb execution. Runs strings from memory.

This separation enables dynamic test suite construction based on runtime conditions.

Isolation Strategy: The Savepont Mechanism



Performance & Constraints

- **Zero Latency:** No container startup or connection overhead.
- **Metadata Operation:** Savepoints manipulate transaction state, not data pages.
- **The 64-Limit:** PostgreSQL caches 64 subtransactions. `pgmi` avoids overflow by releasing/rolling back *before* starting the next test. Active count = Depth, not Total.

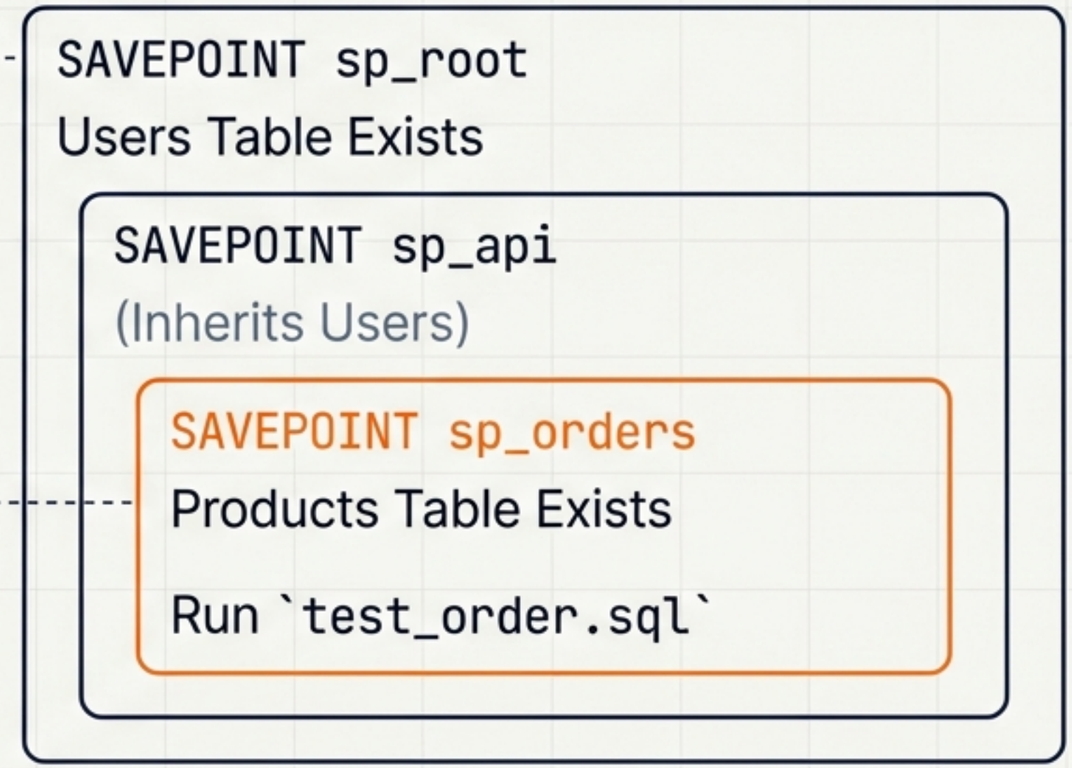
Hierarchical Fixture Composition

The directory tree *is* the data dependency tree.

File System (Tree)

```
/
├── _setup.sql  Create Users
└── /api
    └── /orders
        ├── _setup.sql  Create Products
        └── test_order.sql
```

Database Scope (Nested Containers)



Tests automatically inherit the context of all ancestor setup scripts. Cleanup happens in reverse.

The Algorithm I: Discovery & Aggregation

Transforms a flat list of files into a navigable tree structure with metadata.

Step 1: Filtering

pgmi_source (JetBrains Mono)

path (text)	type (text)
./api/schema.sql	MIGRATION
./api/orders/schema.sql	MIGRATION
./api/orders/__test__/test_create.sql	TEST
./api/orders/__test__/test_delete.sql	TEST
./core/schema.sql	MIGRATION

```
DELETE FROM pgmi_source
WHERE path ~ '/__tests?_/'
RETURNING * INTO pgmi_unittest_script
```

Test Files

Normal Migrations

Step 2: Aggregation

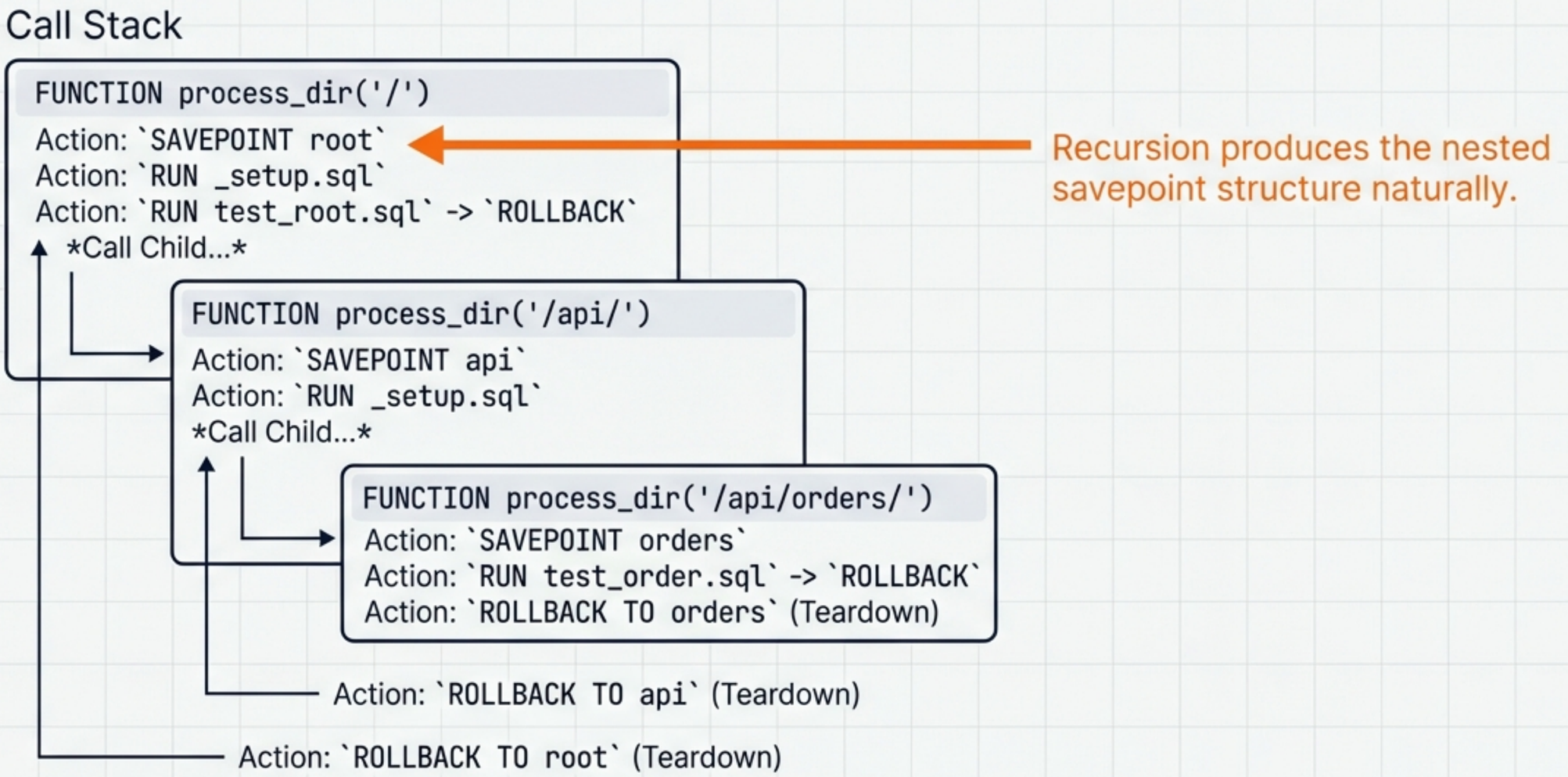
pgmi_unittest_vw_directory

```
{
  "directory": "./api/orders/",
  "parent": "./api/",
  "has_setup": TRUE,
  "has_teardown": FALSE,
  "test_scripts": [
    "test_create.sql",
    "test_delete.sql"
  ]
}
```

Transforms a flat list of files into a navigable tree structure with metadata.

The Algorithm II: Recursive Traversal

Depth-First Search logic in `pgmi_unittest_pvw_script()`



The Algorithm III: Plan Materialization

The Stream

The recursive function yields rows.

Yield

The Embedding

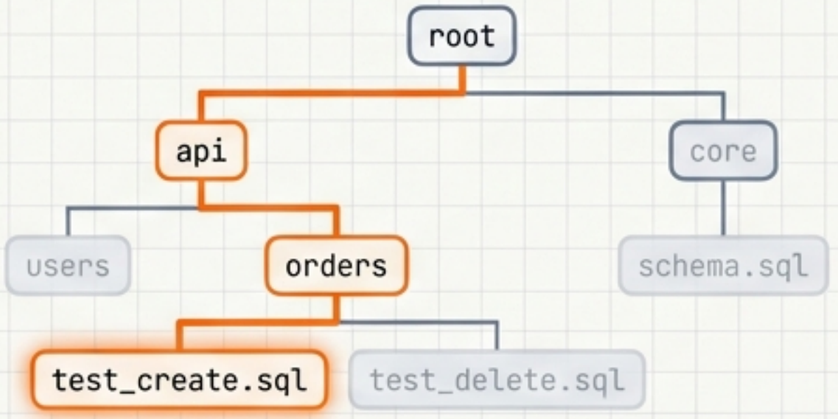
`pgmi` embeds the actual SQL content into the plan.

pgmi_plan

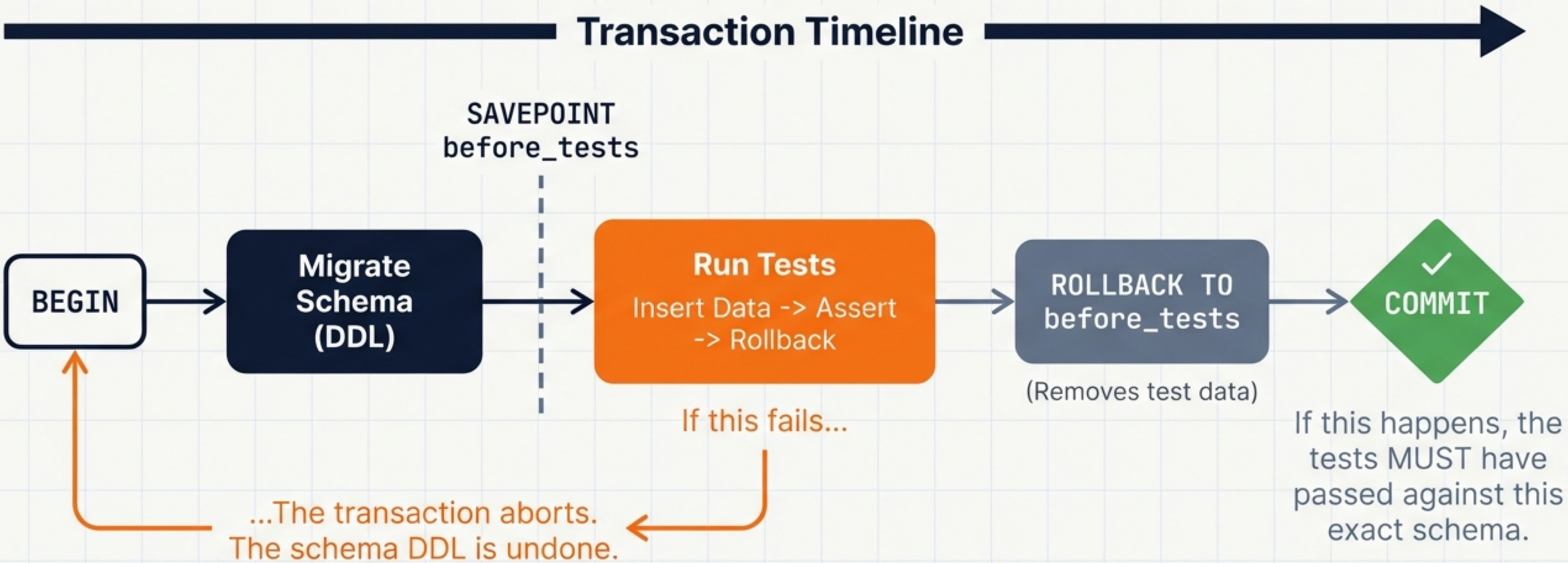
ordinal	command_sql	executable_sql
10	SAVEPOINT sp_001; -- Setup	CREATE TABLE users (id SERIAL PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

Intelligent Filtering

```
-- When filtering for "/orders/", we preserve parents:
WHERE path LIKE parent_path || '%'
```



The Payoff: The ‘Gated Deployment’ Pattern



Atomic Deploy-and-Test. Zero bad schemas in production.

Compliance & Safety Implications

Meeting PCI-DSS, SOX, and HIPAA requirements with cryptographic certainty.

Traditional Risk

Deployment without verification.

Outcome: $440M Loss.
Untested code activated in production.

Gated Safety

The Audit Trail

```
> pgmi deploy
[INFO] Migrating schema... OK
[INFO] Running tests...
       - test_orders.sql ... PASS
       - test_users.sql ... PASS
[INFO] Tests passed. Committing transaction.
Exit Code: 0
```

- ☑ Evidence of testing *during* change event.
- ☑ Atomic guarantee.
- ☑ Audit trail via `RAISE NOTICE`.

The exit code is the compliance artifact.

Tradeoffs & Constraints

Where `pgmi` hits the boundaries of the model.

No Parallel Execution.

Reason: `pg_temp` tables and Transactions are bound to a single session.

Impact: Test suite scaling is linear, not horizontal.

PostgreSQL Only

Reason: Relies on Transactional DDL and Savepoints behavior specific to Postgres.

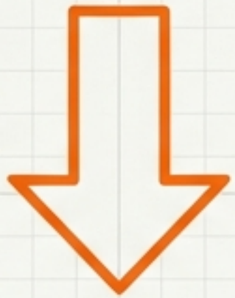
No Side-Effects.

Reason: Cannot rollback external API calls (HTTP, S3) via `ROLLBACK`.

Summary: Evolutionary Database Design

pgmi moves deployment logic from external tools (Java/Go/Perl) into the database itself (PL/pgSQL).

Test-Then-Deploy
(Race conditions, drift)



Deploy-And-Test
(Atomic guarantee)

“It requires you to think about transaction boundaries explicitly, but gives you the power to control them completely.”

`Open Source: github.com/vvka-141/pgmi | Documentation: pgmi.dev`