2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

A 3-Tiered Agentic AI Framework for Verification Regression

Jin Choi, Sangwoo Noh, Seonghee Yim, Youngsik Kim
Samsung Electronics

Contents

- Motivation
- Main Idea
- Results
- Conclusion & Future Works

Motivation

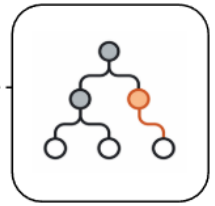
The Complexity Crisis in Modern SoC Verification

- **Unprecedented Design Complexity:** Modern SoCs integrate tens of millions of gates and hundreds of interdependent IP blocks.
- **Resource Intensity:** Continuous design changes require massive, repetitive test execution, draining computational resources
- **Structural Limitation:** Current CI tools and simple LLM patches fail to address the fundamental flaw: the Monolithic pipeline.

Current Automation Optimizes Execution, *not Architecture*

Localized Optimization

Schedules daily/weekly runs.
Reduces manual overhead.



ML Triage

Automates failure sorting or ownership assignment.

LLM / RAG

Summarizes logs and extracts failure patterns.

The limitation: Localized Optimization

These tools operate within the constraints of existing monolithic infrastructures.
They improve specific stages but do not address the underlying structural rigidity.

Monolithic Regression in the Era of Scale

In large-scale environments, verification is split across multiple teams. A flat, centralized regression system cannot adapt to design hierarchy or change locality, creating integration bottlenecks.

- **Distributed Friction:** Verification is distributed across teams, but the pipeline forces a centralized bottleneck.
- **Flat Structure:** Systems execute broad test cases without adapting scope to team-specific needs.

Why Centralized Pipelines Fail at Scale

- **Coordination Hurdles**: Maintaining a unified view is nearly impossible when verification is spread across multiple teams and infrastructures.
- **High Manual Overhead**: Aggregating results from heterogeneous environments requires significant manual intervention.
- **Structural Mismatch**: Monolithic pipelines cannot keep up with the distributed nature of modern workflows.

Monolithic Architectures Reject Agentic AI.

Embedding emerging technologies – Agentic AI, LLMs, and Model Context Protocol (MCP) – into a flat pipeline creates technical debt.

Technical Barriers

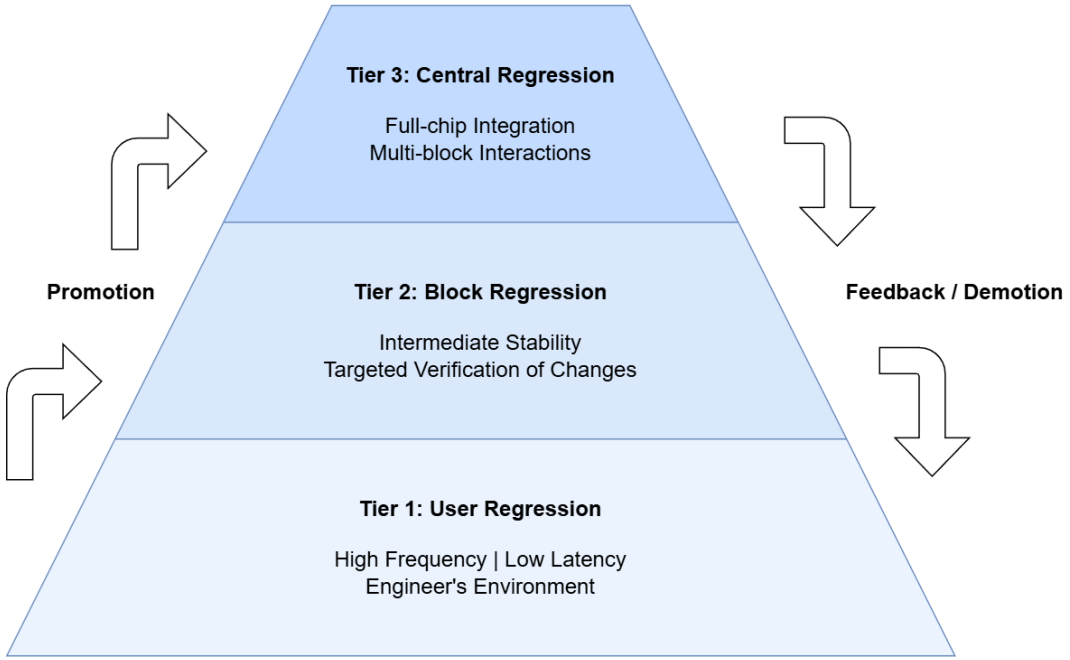
- **Scalability**: Centralized control limits the autonomy required for agent workflows.
- **Maintainability**: Adding complex agents to rigid pipelines increases verification overhead.
- **Context**: Monoliths lack the granular context (User vs. Block) required for high-performing AI.

To leverage Agentic AI, the infrastructure must be as modular as the agents themselves.

Main Ideas

Restructuring Verification: 3-Tiered Regression Framework

Aligning regression scope with verification granularity – User, Block, and Central



Architectural Philosophy

- **Granularity**: Specific scopes and objectives for each level.
- **Intelligence**: Distributed, not concentrated.
- **Coordination**: Asynchronous communication.

The Workforce: Three Agent Types

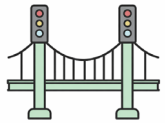
Regression Type Agents

The Brain

Role: Manage verification objective and decision making.

Examples: User Agent, Block Agent, Central Agent.

Bridge Type Agents



The Connectors

Role: Mediate data exchange via Pub-Sub mechanism.

Examples: User-Block Bridge, Gate Agent.

Operation Type Agents

The Hands

Role: Execute auxiliary, stateless tasks.

Examples: Test Execution, Error Classification.

Tier 1: User Regression Tier

The Environment: Individual engineer workspaces operating concurrently.

The Behavior: Frequent execution of identical test sets.

The Goal: Sanity Regressions. Prioritizes fast turnaround time.

Value Add: Converts informal ad-hoc testing into measurable and analyzable component. (Frequency, Latency, Signatures)

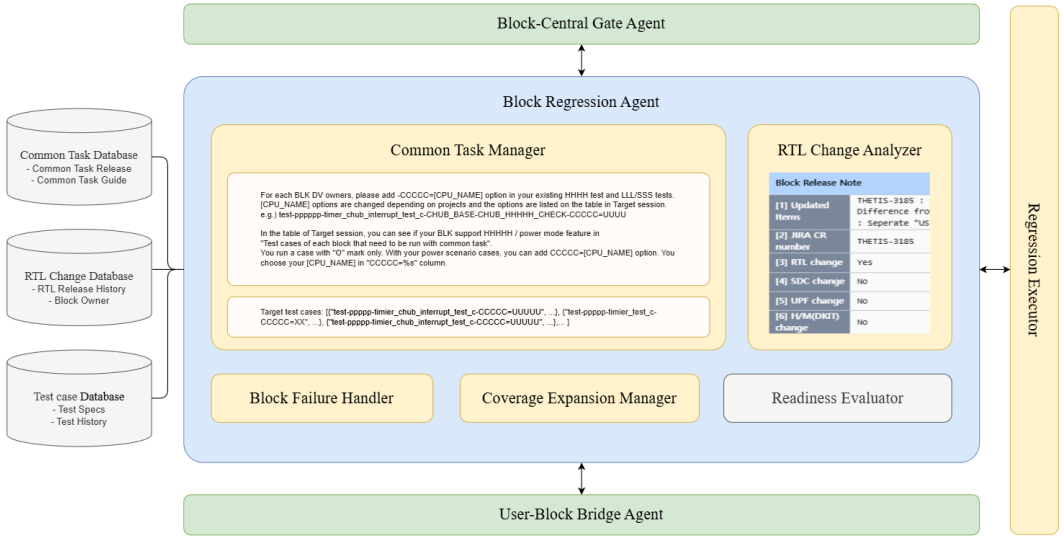
Tier 2: Block Regression Tier

The Firewall: Prevents unstable blocks from reaching full chip.

Workflow managed by Block Regression Agent:

1. **Monitor**: Scans RTL release notes.
2. **Identify**: Detects affected design blocks.
3. **Execute**: Runs block-scoped tests + randomized scenarios.

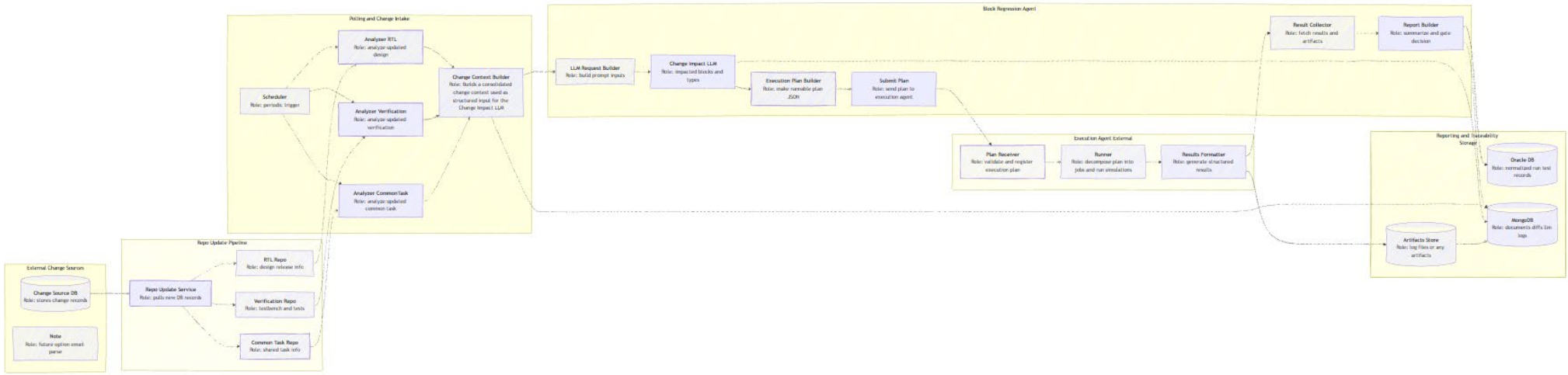
Outcome: Confines failure analysis to the specific modified block.



Automation Impact: Eliminates the 2-3 day manual delay for identifying affected test cases.

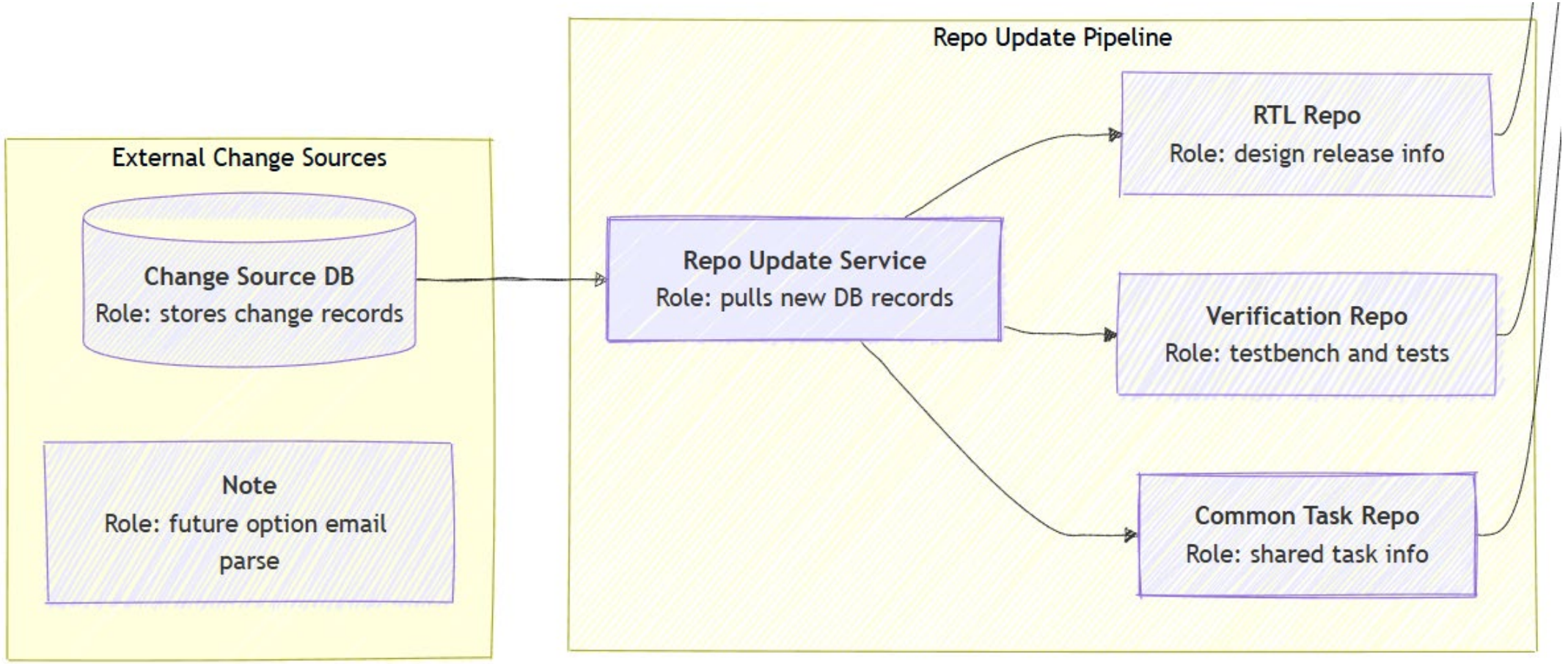
Automating Impact Analysis & Test Selection

Eliminating manual test selection to ensure no change-affected scenario is missed.

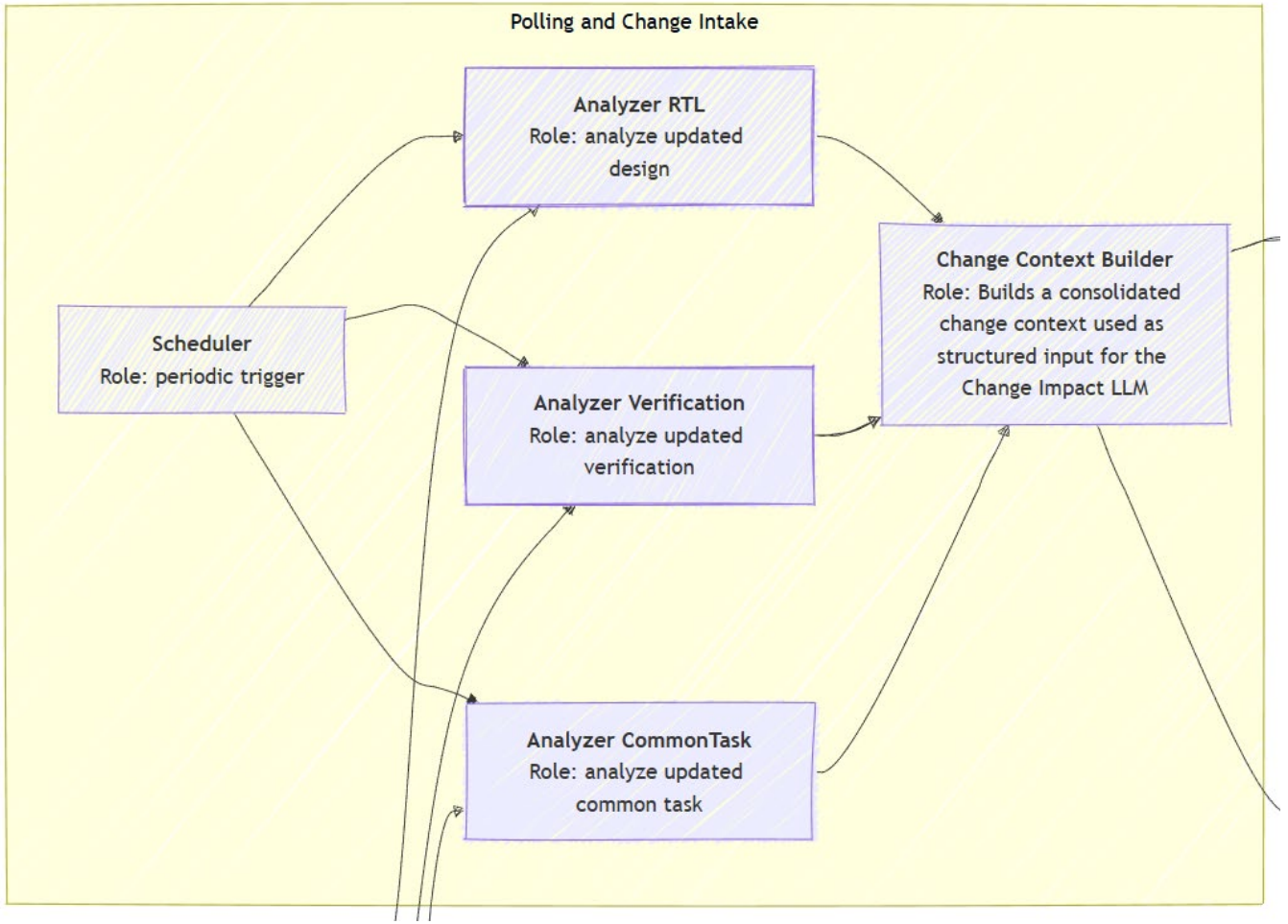


- Problem: Engineers miss affected test cases during manual selection.
- Solution: Agent derives tests directly from documented design/verification changes.
- Coverage Expansion: Automatically adds randomized tests to reduce bias.

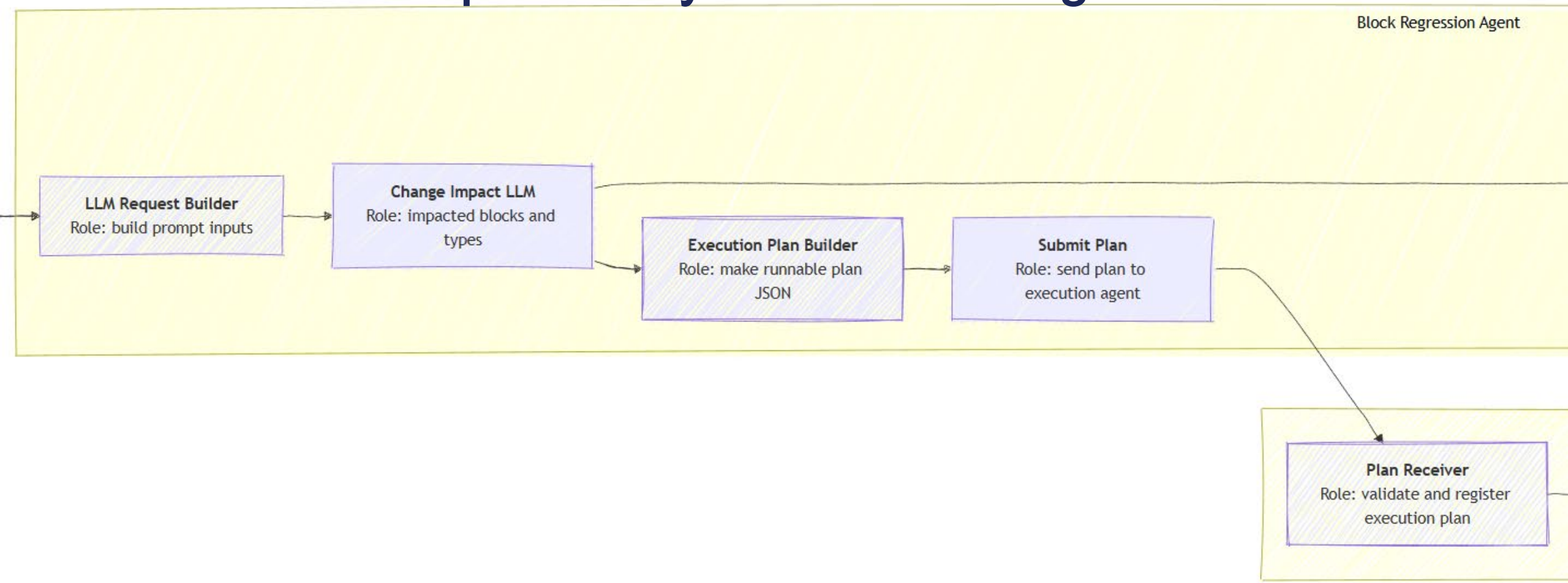
Change Ingestion and Repository Update



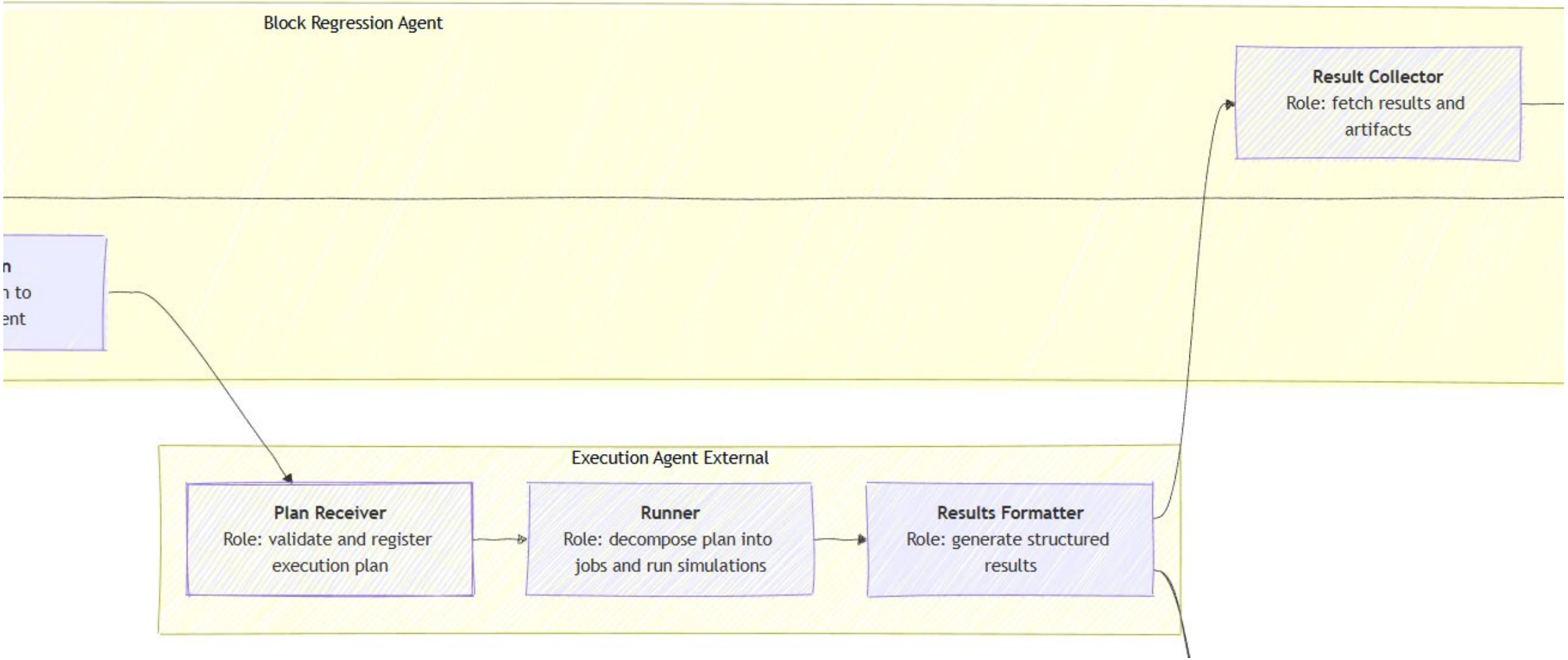
Polling-Based Change Analysis



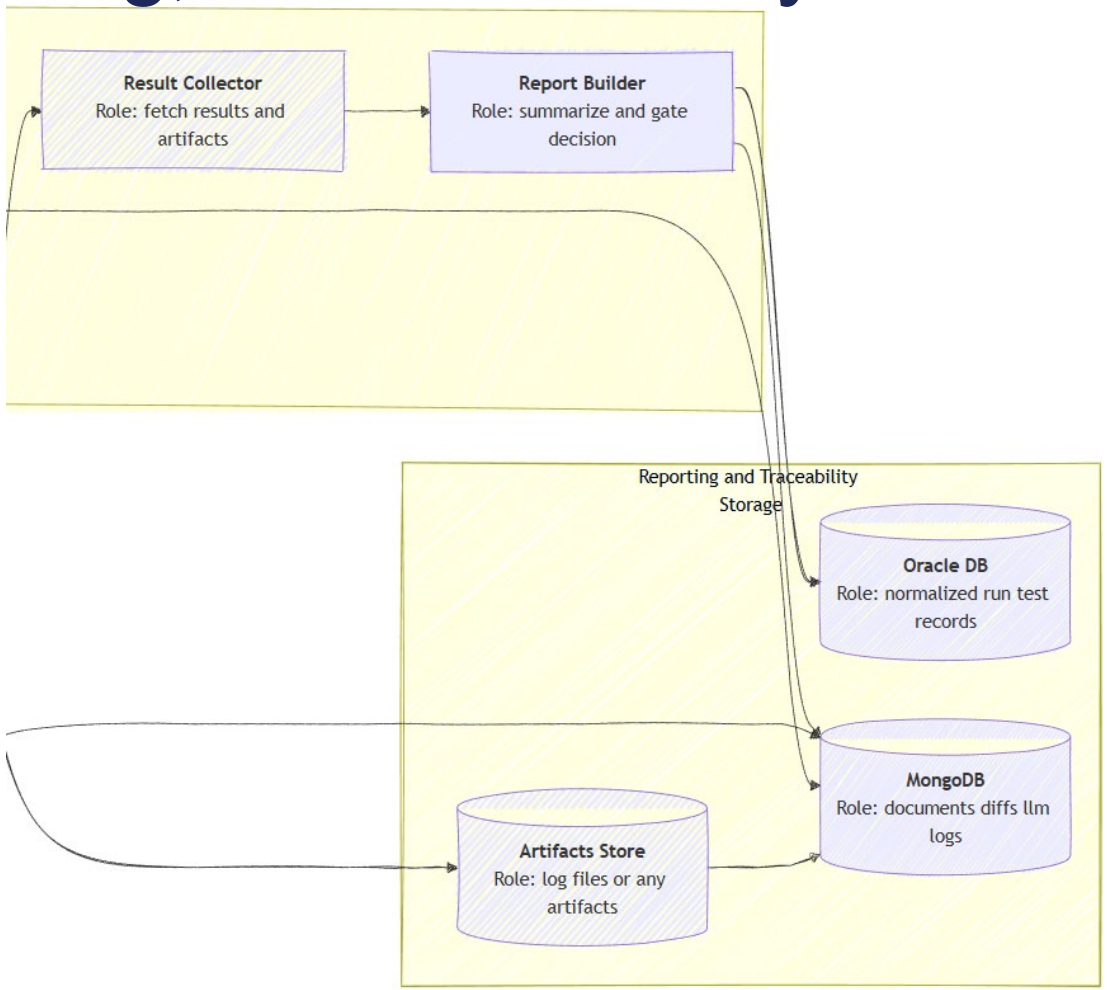
LLM-Driven Impact Analysis and Planning



Decoupled Execution and Result Normalization



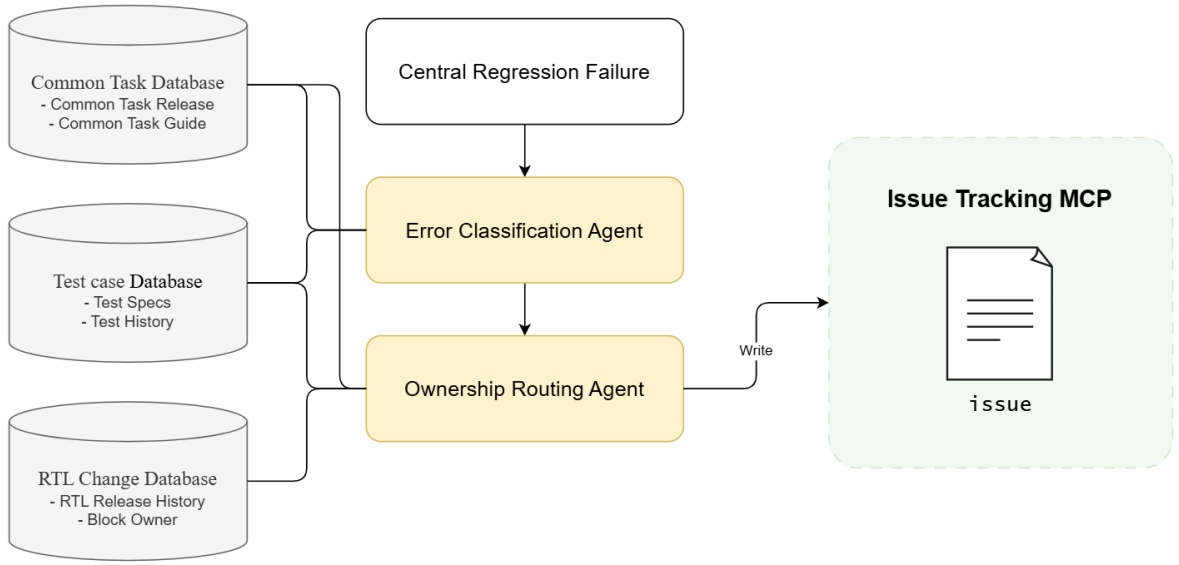
Reporting, Gating, and Traceability



Tier 3: Central Regression Tier

The Gating Strategy: Execution is only initiated when block-level pass rates hit the threshold.

- **Error Classification Agent**: Distinguishes Common Task vs. Domain Specific errors.
- **Ownership Routing Agent**: Maps failures to owners via JIRA API. End-to-end traceability.

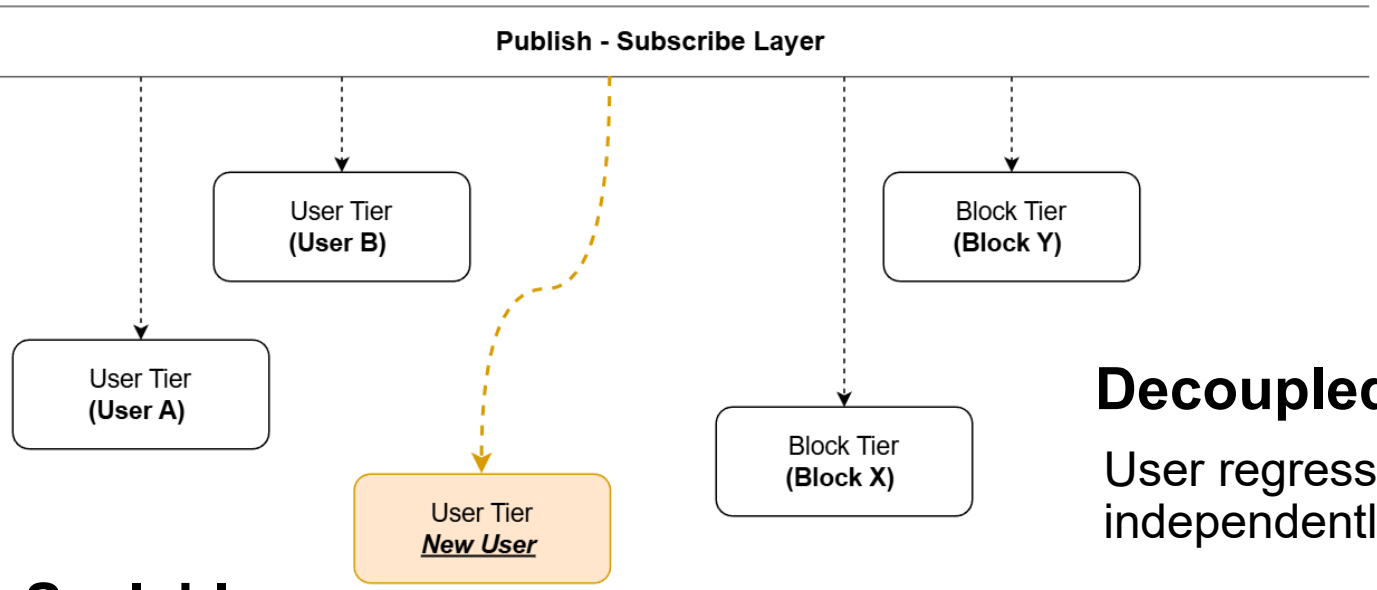


Advanced Failure Handling: Distinguishes between generic and specific errors. Users similarity scoring to route tickets to the proper owner automatically, bypassing manual triage.

Loosely Coupled Coordination via Publish-Subscribe.

Asynchronous

Tiers do not block each other.



Decoupled

User regression operates independently of other tier state.

Scalable

New subscribers (Agents) can be added without breaking the pipeline.

Optimize Resources via Gating Thresholds

The Central Regression Tier handles complex chip-level verification and multi-block interactions and these processes are resource-intensive. The Gate Agent acts as a dedicated coordination component between Block and Central regression tiers.

- **The Gating Strategy**: Execution is only initiated when block-level pass rates satisfy the stability threshold.
- **Benefits**
  - Enforces stability thresholds to prevent premature chip-level integration.
  - Reduces waste from known block-specific issues.

Database Design – Hybrid Data Model

Relational Database

The **What** – Official Record

- **Purpose**: Optimized for rigid reporting and joins.
- **Characteristics**: Joinable, immutable history, structured schema.
- **Key Entities**
  - Run Status
  - Promotion Decisions
  - Block Metrics

NoSQL Database

The **Why** – Context & Evidence

- **Purpose**: Optimized for deep debugging and AI analysis.
- **Characteristics**: Flexible schema, high volume, bulk payloads.
- **Key Documents**
  - Raw Event Payloads
  - Policy Snapshots
  - Failure Signatures

Decision Engine – Reproducible Decision Making

Decisions must be reproducible. By linking **GATE_EVAL** to a specific version of **GATE_POLICY**, we can explain why a decision was made even if rules change later.

- Re-evaluation logic: If a policy changes, a new **GATE_EVAL** row is created for the same `block_run_id`. Queries filter for the latest `evaludated_at` timestamp.

GATE_EVAL (The Decision)	
`decision`	PROMOTE / REJECT / QUARANTINE
`score`	Calculated stability metric
`policy_id (FK)`	Links to Rule Set

GATE_POLICY (The Rule Set)	
`min_pass_rate`	e.g., 0.85
`allow_known_issues`	Y / N
`policy_version`	e.g., v2.1

Decision Engine NoSQL Schema

Answering the question: ***Why did this happen***?

- `gate_events`: The raw immutable payload for debugging and replay. Ultimate debuggability and allows re-running the gate logic against historical data.
- `gate_decision_docs:` Contains full evidence for every gate decision. This document embeds the policy and metrics as they existed at the moment of decision.

```
{
  _id: String,
  source_event_id: String,
  run_id: String,
  block_run_id: String,
  received_at: Date,

  source: {
    tier: String,
    publisher: String,
    repo_ref: String,
    base_sha: String,
    head_sha: String
  },

  payload: Object
}
```

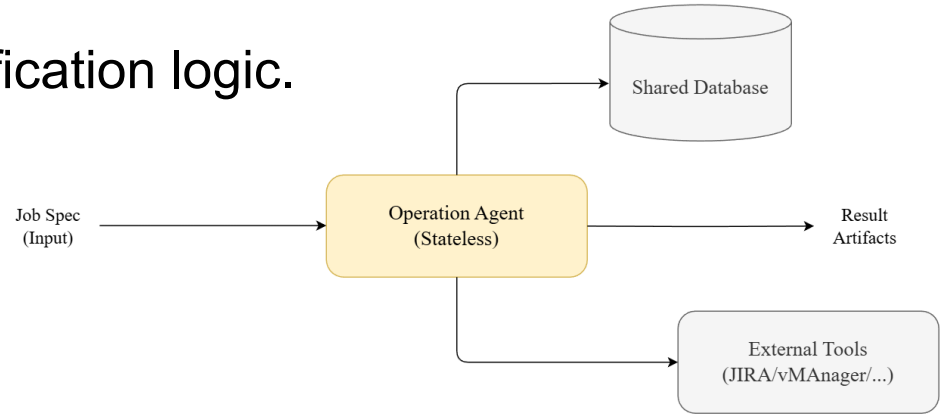
```
{
  _id: String,
  gate_eval_id: String,
  block_run_id: String,
  run_id: String,
  created_at: Date,

  policy_snapshot: {
    policy_id: String,
    policy_version: String,
    min_pass_rate: Number,
    max_flaky_score: Number,
    allow_known_issues: Boolean
  },

  metrics: {
    pass_rate: Number,
    required_complete: Boolean,
    fail_count: Number,
    pass_count: Number,
    timeout_count: Number,
    crash_count: Number,
    timeout_ratio: Number,
    flaky_score: Number
  },
```

This architecture moves us from checking if it passed
to ***understanding why it is safe to proceed the central regression.***

Operation Type Agents: Stateless Execution

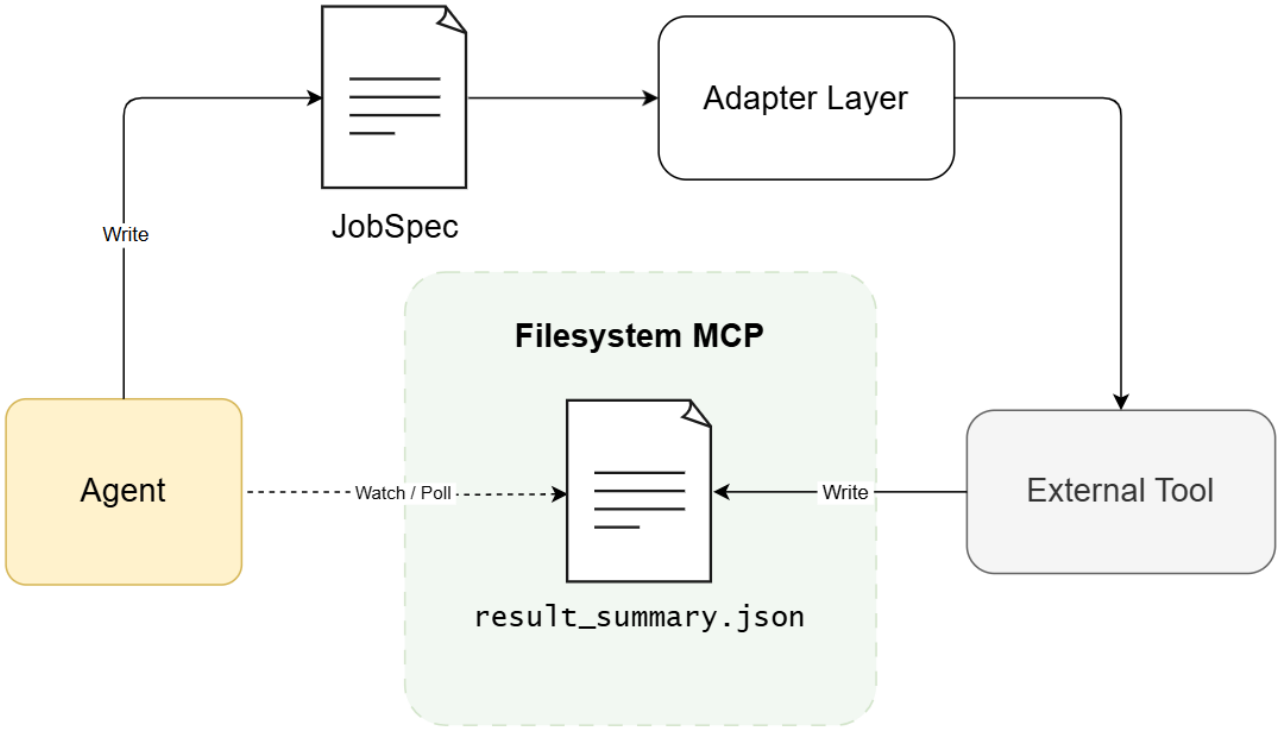
These agents don’t know why they are testing; they just execute concrete system-level functions.

- **Stateless**: Retrieve context from database, persist results to database
- **Interface**: Explicit inputs/outputs
- **Benefit**: Upgrade tools without breaking verification logic.



Regression Execution Agent

Materialize abstract decisions into concrete workloads.



A Modular Execution Framework

The concrete execution backbone for evolving regression workflows. Modular design allows agents to be swapped without breaking the pipeline.

- **Decoupled:** Verification logic evolves independently of execution mechanics.
- **Recoverable**: Data-driven state ensures resilience against infrastructure instability.
- **Standardized**: Explicit JSON schemas create a universal interface for all tiers.

Results

Solving the Distributed Verification Challenge

The Challenge

Heterogeneous infrastructures and geographically distributed teams make it hard to maintain a unified view.

The Governance Solution

- **Unified Governance**: Tiered agents ensure consistent policy across different infrastructures.
- **Gap Reduction**: Explicit scope definitions (User/Block/Central) minimize coverage gaps.
- **Structured Visibility**: Managers see health at every level, not just a binary system Pass/Fail.

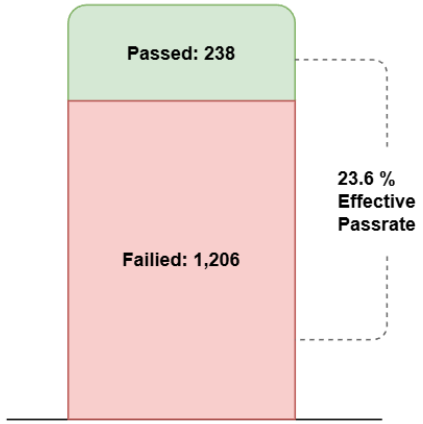
Quantifying Monolithic Failure (The Baseline)

Without hierarchical gating, significant portions of the design enter regression “blind”.

- **Data**: In a study of 33 blocks (20k+ test cases), nearly 14% had never been run before central regression.
- **Oversight**: One block had > 1,100 unexecuted test cases due to manual error.
- **Consequence**: Critical bugs pushed to last stage integration.

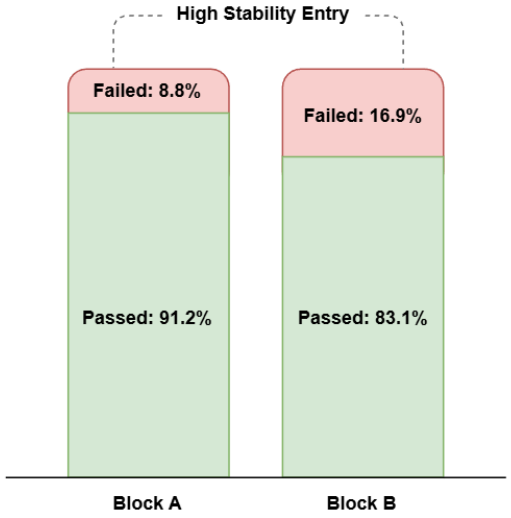
Empirical Evidence: Impact of Gating

Traditional Flow (Baseline)



High Noise, Low Value.

Proposed Framework



Clean State, Effective Analysis.

Takeaway: ensures Central Regression operates on a clean state, reducing noise and enabling chip-level analysis.

Conclusion & Future Works

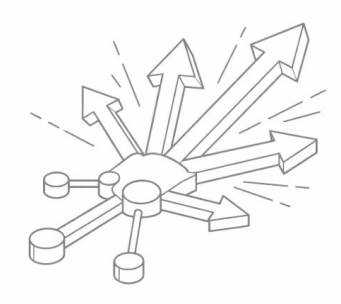
Conclusion- Structured and Hierarchical Regression Flow

- **Clear Scope Alignment**: Verification activities are explicitly aligned with the design hierarchy (user → block → chip), ensuring that changes are validated at the appropriate scope and preventing premature or excessive regression execution.
- **Governed and Predictable Flow**: Regression execution is transformed from an ad-hoc process into a controlled and repeatable workflow, with explicit qualification and readiness checks enforce disciplined promotion across regression tiers.

Conclusion – Data-Driven and Quality-Governed Verification

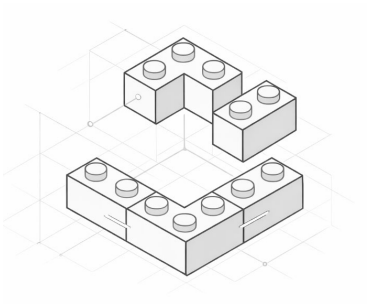
- **Data-Driven Visibility and Decision Making**: Regression results, change context and failure signatures are systematically captured, providing distributed teams with a consolidated and objective view of regression health and stability.
- **Enforced Coverage Constraints**: Each regression tier enforces clearly defined and scope-appropriate coverage requirements to ensure verification completeness.

A Robust Foundation for the future of SoC Verification



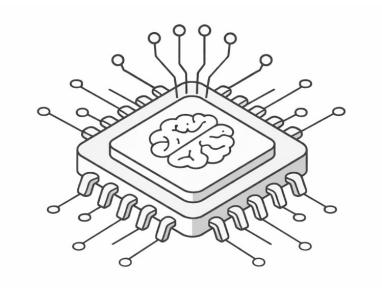
Scalability

Handles millions of gates through hierarchical distribution.



Maintainability

Modular agents are easier to update than monolithic scripts.



AI-Readiness

Native support for Model Context Protocol (MCP) and agent workflows.

The 3-Tiered Agentic AI Regression Framework is the **necessary structural evolution** to sustain the next generation of SoC design and verification.

Future Works

Extending the framework to multi-project environments and deeper AI integration.

- **Multi-Project Scope**: Handling cross IP dependencies and shared infrastructure.
- **Richer Context**: Using LLM to analyze unstructured data (design discussions, debug traces).
- **MCP Standardization**: Deepening Model Context Protocol integration for seamless tool interoperability.

The End☺