Replayable Agent Mission Trace & Failure Attribution

1. Summary

Adds a **mission-level execution trace** to Go2 agent pipeline.

Goal: make every mission replayable and explainable:

user → agent → tool → MCP → robot → environment → verification → result

No new observability stack—built on **memory2, MCP, ToolStream, SkillResult**.

2. Problem

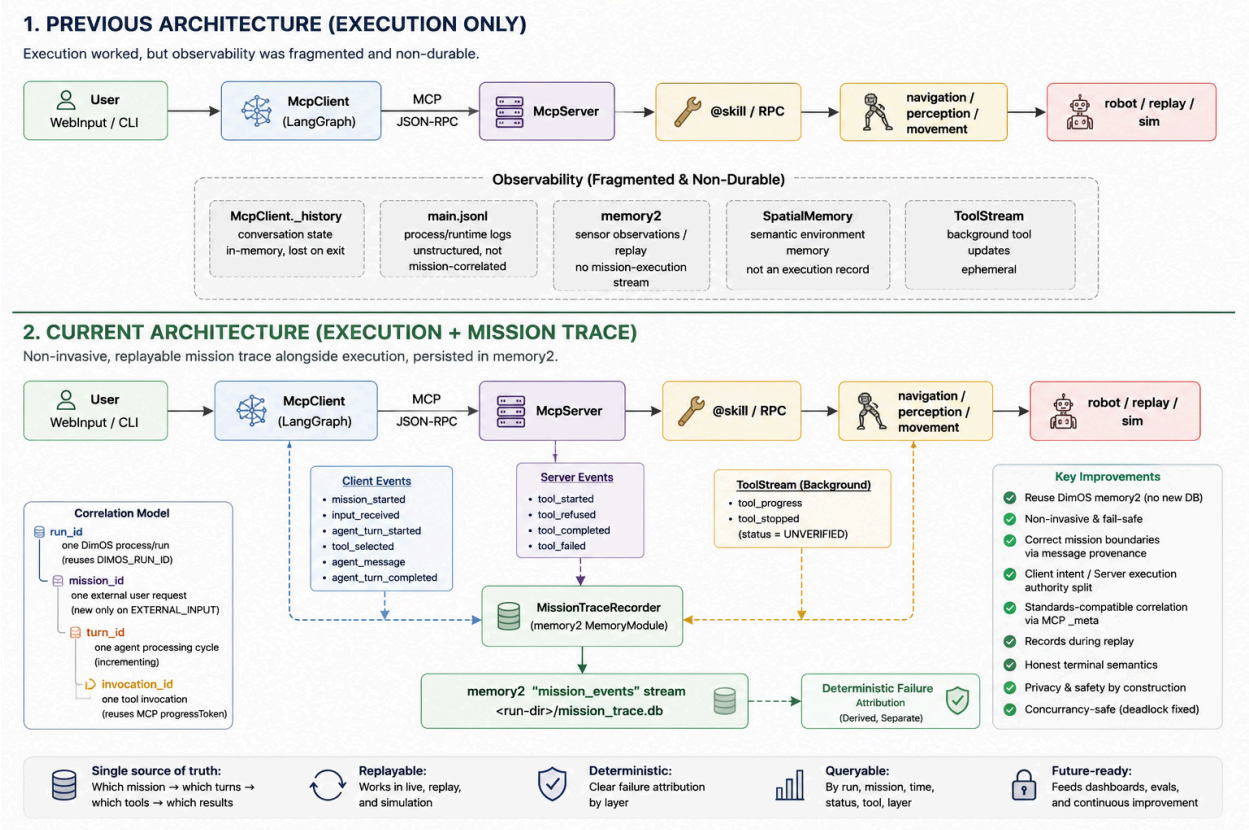
When the agent is given a physical task, the record of what actually happened was fragmented and non-durable across several sources:

- **McpClient._history:** Lost on exit (in-memory only).
- **main.jsonl:** Unstructured and not mission-correlated.
- **memory2:** Lacked a dedicated mission-execution stream.
- **SpatialMemory:** Semantic only, not an execution record.
- **ToolStream:** Ephemeral background updates.

3. Solution

Introduce a **mission trace layer** stored in `<run-dir>/mission_trace.db` that records agent decisions, tool calls, MCP execution, and failure reasons. This ensures every mission is replayable and explainable.

4. Architecture & Design Principles



Key Design Principles	
Principle	**Description**
1. No control impact	Tracing is side-channel only: it never blocks execution and never alters MCP behavior.
2. Clean correlation model	Uses a hierarchical structure: `run_id → mission_id → turn_id → invocation_id`.
3. Event split	Differentiates between **Client** (intent: mission start, tool selected, agent turn) and **Server** (execution: tool started, completed/failed, duration/errors).
4. Background execution awareness	Tool completion (RPC finished) is distinguished from physical task completion.

Key Design Principles	
Principle	**Description**
5. Message safety	Only external input creates new missions; `TOOL_PROGRESS` updates are attached to existing missions.
6. Failure attribution	Derived from execution evidence (e.g., tool selection, perception, infrastructure) rather than model introspection.

5. Physical Verification & Sensor Correlation

The `MissionVerifier` layer closes the loop on physical execution by correlating event timestamps with memory2 sensor frames (LIDAR, Camera, Odom) to identify goal state and automated verification (SUCCESS/FAILURE).

6. Capabilities Enabled

Missions are transformed into structured, end-to-end records enabling:

- Initial input and agent decision
- Tool execution and robot action
- Physical outcome and verification
- Attributed failure reason

7. Current Implementation Status

- **Trace Schema:** Completed
- **MCP Correlation:** Completed
- **ToolStream Integration:** Completed
- **Memory2 Persistence:** Completed
- **Replay + Simulation:** Completed
- **Failure Attribution:** Completed
- **Integration/Replay/Correlation Tests:** Completed

1. **Reuse over Reinvention:** Correlation leverages existing `DIMOS_RUN_ID`, `memory2`, and `SkillResult`.
2. **Non-Invasive & Fail-Safe:** Tracing is additive. Errors cannot propagate to the execution path; design becomes a no-op if the recorder is disconnected.
3. **Provenance-Based Boundaries:** Uses message envelopes to distinguish genuine user input from automated `tool_progress` updates, preventing spurious mission spawning.

4. **Authority Separation:** Client owns intent (missions, turns, tool selection); Server owns execution (starts, failures, duration). No overlap.
5. **Standards-Compatible:** Correlation data propagates via MCP `_meta` fields. No protocol changes; maintains compatibility with external MCP clients.
6. **Replay-Native:** Subclasses `MemoryModule` to capture new execution events during replay, even when standard sensor recording is disabled.
7. **Honest Semantics:** `tool_completed` strictly signifies RPC return, not physical success. Background tasks stay UNVERIFIED rather than auto-assuming success.
8. **Secure by Construction:** Attributes are sanitized. Sensitive data, chain-of-thought, and raw payloads are excluded.
9. **Concurrency Correctness:** Resolved potential deadlocks with short-lived correlation state locks.

8. Roadmap & Next Steps

10. **Physical-world verification (in progress)** — `MissionVerifier` correlating `/odom` + `/goal_reached` to emit `task_verification` with measured pose deltas, + skill/ToolStream outcome signals (Section 9).
11. **Web dashboard** — read-only mission timeline / failure view over `mission_trace.db` (queries the existing `memory2` stream).
12. **Larger tracing & eval framework** — cross-run comparison across replay / sim / models / hardware; regression and success-rate evals built on the trace as ground truth.
13. **Skill changes** — standardize key skills on `SkillResult` and background lifecycle so execution outcomes are authoritative rather than inferred.
14. **Automatic reasoning-failure attribution** — conservative, oracle/benchmark or human-label driven; still no inspection of hidden chain-of-thought.

End Goal

A single mission becomes:

fully replayable
fully attributable
physically verifiable
evaluation-ready