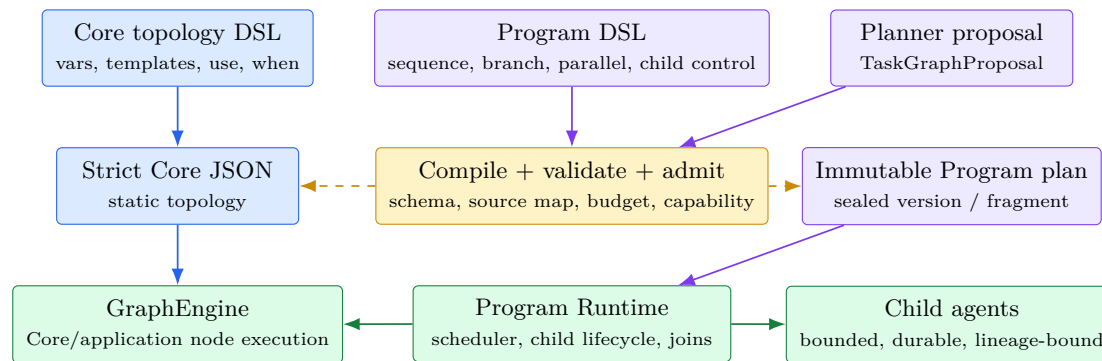


NeoGraph

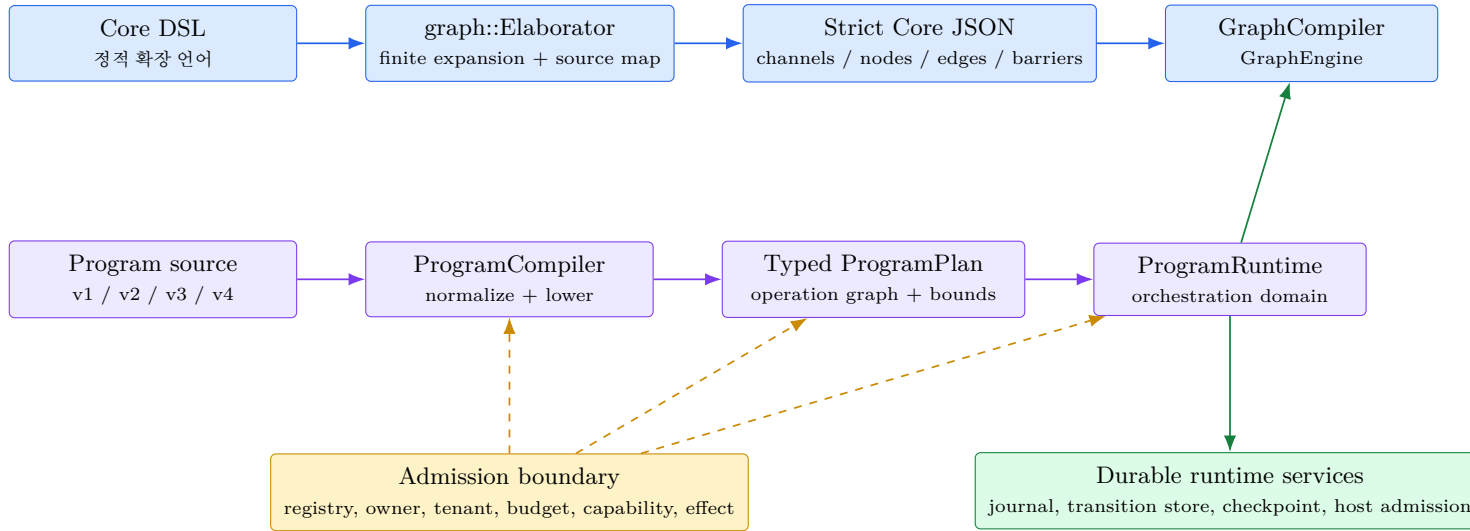
Bounded Self-Evolving Multi-Agent Runtime

Core DSL, Program DSL, Harness, admission, durable child orchestration, and versioned evolution



읽는 법. NeoGraph는 하나의 통합 DSL이 아니다. Core DSL은 정적 Core topology를 만들고, Program DSL은 sealed Core와 child Program을 유한하게 제어한다. 모델이 제안한 값은 실행 권한이 아니라 검증 대상인 데이터이며, admission을 통과한 immutable version만 실행된다. 기준 상태: 2026-08-07

1. 전체 시스템 지도: 두 언어와 하나의 실행 계층



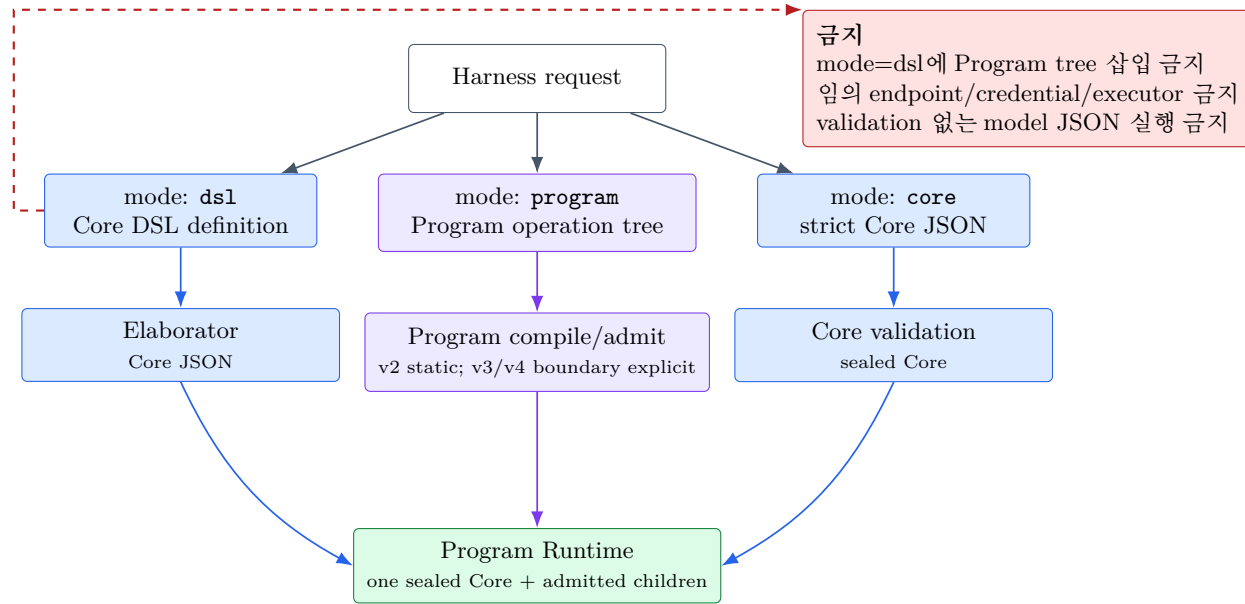
의도된 bridge. Core DSL은 strict Core JSON으로만 내려가고, Program은 그 sealed Core를 `call_core`로 호출한다. Core DSL 자체가 Program control VM으로 변하는 것은 아니다.

핵심 구분. Core DSL은 반복 작성과 topology 재사용을 해결한다. Program DSL은 유한한 orchestration과 child lifecycle을 해결한다. 두 언어를 합치지 않는 이유는 Core DSL의 결정성·정적 검증·source map을 보존하기 위해서다.

Core DSL	Program DSL	Program Runtime
vars, interpolation, templates, use, when	sequence, branch, bounded loop/retry, parallel, race, quorum, map, parallel_map, expand_task_graph	spawn, await, cancel, checkpoint, scheduling, joins, recovery
static Core topology	typed control tree / task graph	durable execution and authority enforcement

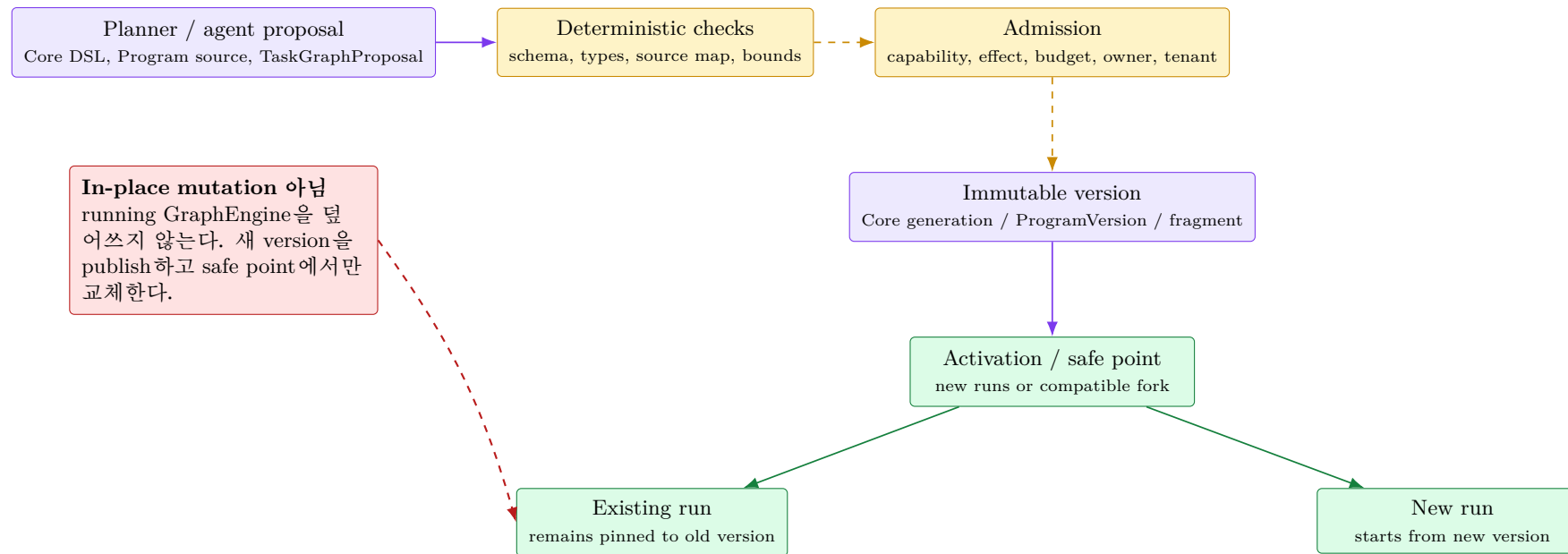
범례. 파랑 = Core, 보라 = Program, 황색 = admission/policy, 초록 = runtime/evidence. 실선은 data/execution flow, 점선은 constraint 또는 authority boundary를 뜻한다.

2. Harness와 외부 authoring 경계



현재 외부 노출의 중요한 사실. Direct Program API가 v3 `parallel_map`과 v4 `expand_task_graph`를 단계적으로 지원하더라도 Harness의 `program` profile은 별도의 schema/version 계약을 가진다. 따라서 “compiler가 받는다”와 “Harness가 외부에 공개한다”는 동일한 주장이 아니다.

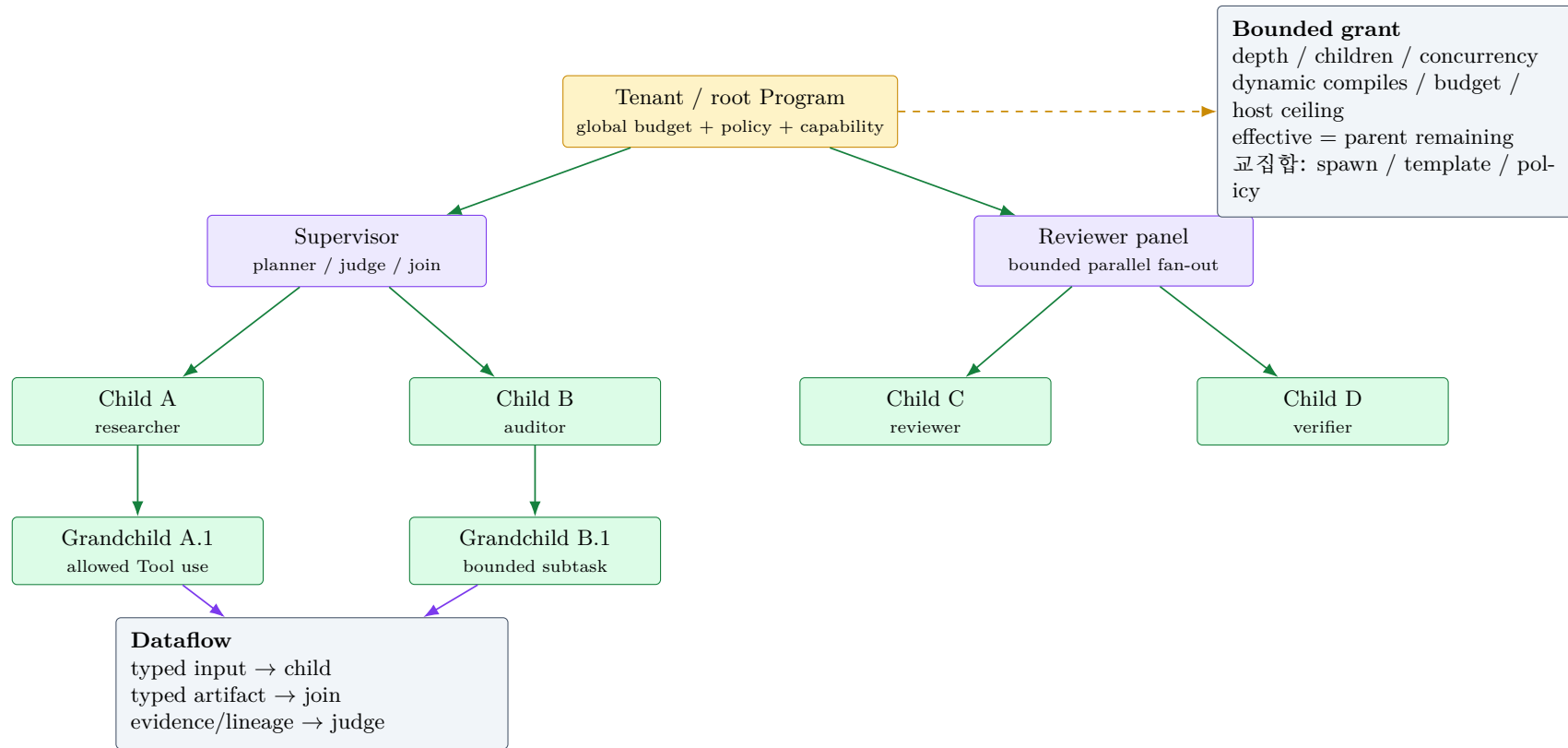
3. Self-evolution과 version swap



Self-evolution은 “모델이 원하는 JSON을 실행” 하는 것이 아니라 다음의 versioned pipeline이다.

1. proposal은 untrusted data다.
2. compiler가 유한성과 typed contract를 증명한다.
3. admission이 권한·effect·budget을 결정한다.
4. immutable version/fragment가 publish된다.
5. 기존 run은 기존 version에 pin되고, 새 run 또는 safe-point fork만 새 version을 사용한다.

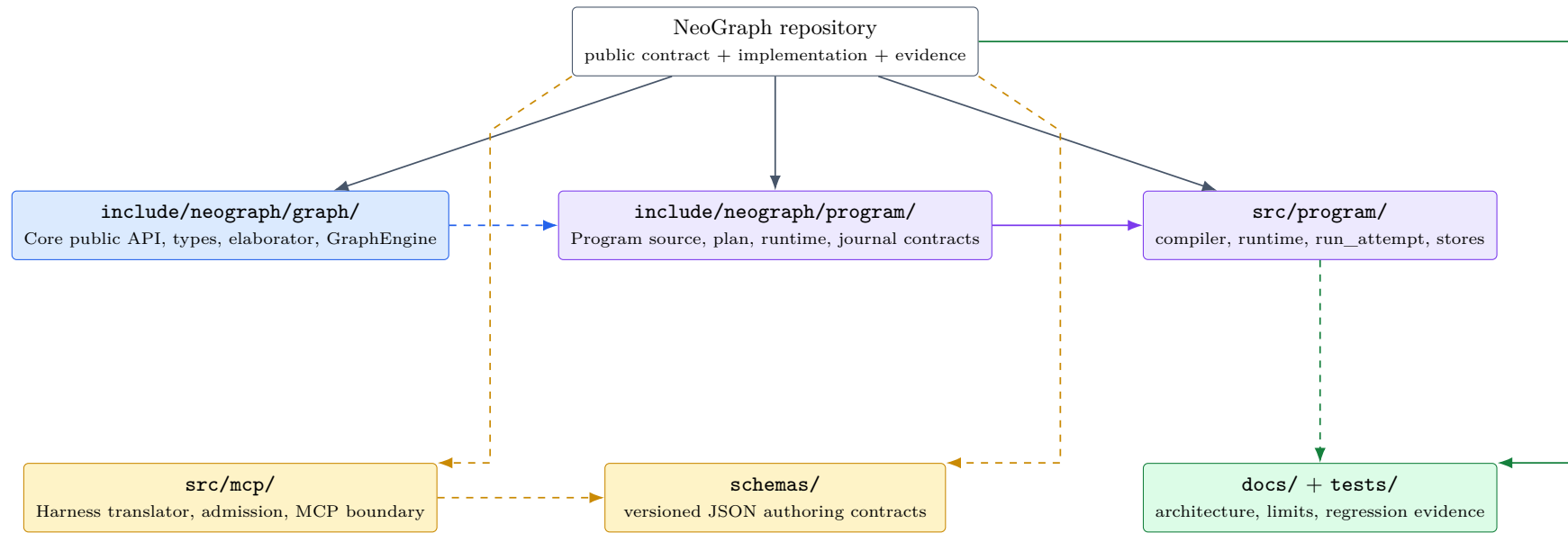
4. 계층형 sub-agent runtime



Child가 할 수 있는 일. 승인된 template의 instance 생성, typed task 수행, host-backed Tool 호출, bounded retry/parallel, 결과 artifact 발행, 위임된 범위의 descendant 생성, await/cancel/checkpoint/recovery가 가능하다.

Child가 할 수 없는 일. arbitrary shell/HTTP/MCP endpoint, credential 생성, executor/provider 임의 선택, capability self-escalation, unbounded recursion, 다른 tenant 상태 접근, live graph mutation은 허용하지 않는다. Tool 권한은 `can_use`와 `can_delegate`를 분리하며, descendant로 내려갈수록 교집합을 취한다.

5. Repository surface map



Repository를 읽는 순서. 먼저 `include/`에서 public contract를 보고, `src/`에서 compiler/runtime/Harness 구현을 확인한다. 그 다음 `schemas/`에서 외부 authoring 계약을, `docs/`와 `tests/`에서 의도와 회귀 증거를 대조한다.

Public contract	Implementation	Boundary and evidence
include/neograph/graph/ Core surface	src/program/ compiler, runtime, stores	schemas/ versioned JSON contracts
include/neograph/program/ Program surface	src/mcp/ Harness integration	docs/ + tests/ architecture, limits, regression evidence

6. Delivery map과 현재 상태



완료

P0 TaskGraphProposal IR, P1 static Program surface, P2 bounded parallel_map

진행

P3는 schema/compiler/typed plan과 runtime publication/dispatch/recovery를 연결해야 한다.

예정

P4 N-level durable hierarchy, P5 single-agent baseline과 비용/효용 평가

지금 안정적으로 가능한 것	아직 조심해야 하는 것	P5 이후에도 비목표
정적 Core topology 생성, 유한 static Program, 직접 Program API의 bounded parallel_map, sealed template 기반 작업	Harness에서 최신 v3/v4 dynamic surface 공개, v4 runtime publication, N-level recovery, 대규모 fairness/throughput	arbitrary code/endpoint/credential, 무제한 recursion, in-place mutation, 다중 agent의 자동 진실 보장

설명: 이 리포를 이해하는 순서

1. 먼저 Core DSL과 Program DSL을 분리해서 본다. Core DSL은 graph를 만들고 Program DSL은 graph/child를 제어한다.
2. 그 다음 admission을 본다. 모델이 제안한 JSON은 실행 권한이 아니며, template·Tool·budget·tenant policy의 교집합을 통과해야 한다.
3. 그 다음 runtime을 본다. ProgramRuntime은 GraphEngine을 대체하지 않고, child lifecycle·join·budget·recovery를 담당하는 별도 scheduling domain이다.
4. 마지막으로 self-evolution을 본다. evolution은 live mutation이 아니라 새 immutable version을 만들고 activation/safe point에서 교체하는 과정이다.

이 문서는 현재 설계와 보고된 P0-P2 완료 상태, P3 진행 경계를 시각화한 *architecture map*이다. 구현이 변경되면 상태 페이지와 *operation/version* 표를 함께 갱신해야 한다.