K-12 / G-9 / M-7: Typed-Effect Kernels for Reliable Multi-Agent Systems

Ryan Hunter
Helaix Applied Research Institute
research@helaix.com

November 6, 2025

Abstract

We introduce three typed-effect kernels that form a minimal, compositional basis for reliable multi-agent systems. The *Episode Kernel* $\mathcal{K}_{12}$ specifies single-agent execution with twelve orthogonal components (state, memory, effects, capabilities, randomness, quota, policy, observability, locale; plus pure transition/output functions and identity). The *Graph Kernel* $\mathcal{G}_9$ lifts episodes into typed workflow graphs with explicit orchestration slots (router, scheduler, ledger, metrics, monitor) and laws for safety (deadlock guards), governance (policy non-bypass), and auditability (append-only evidence). The *Market Kernel* $\mathcal{M}_7$ sketches seven primitives for cross-organization invocation and settlement that compose with $\mathcal{G}_9$. We provide operational rules, laws (e.g., audit-before-mutation; monotone quota; locale narrowing), and proof sketches (replay reproducibility, exactly-once effects within a retry horizon, and graph liveness under guarded cycles). We show how these kernels align with effect-handler semantics and established distributed-systems patterns (CRDT convergence, SAGAs, tracing, token-bucket metering) while remaining language-agnostic. We release a reference implementation and conformance tests as supplementary artifacts.

1 Introduction

Large language model (LLM) agents are transitioning from prototypes to critical infrastructure. Production deployments require compositional guarantees: predictable local execution; safe, analyzable coordination; and auditable, cross-organization calls with verifiable evidence. We propose that these needs crystalize into a small set of *typed-effect kernels* whose laws can be checked statically and enforced at runtime.

Contributions. (1) **The K-12 Episode Kernel**—a 12-component context and law suite for single-agent episodes with pure transition/output functions. (2) **The G-9 Graph Kernel**—a typed orchestration layer with router/scheduler/ledger/metrics/monitor as first-class, checkable components. (3) **The M-7 Market Kernel**—seven primitives for discovery, capability negotiation, remote execution, and settlement. (4) **Correctness claims** with proofs or proof sketches: effect containment; audit-before-mutation; monotone budgets; deterministic replay boundary; and deadlock-freedom under guarded cycles. (5) **A conformance suite** and reference code (ancillary).

2 Background

Typed effects and handlers. Algebraic effects and effect handlers supply the semantic backbone for separating pure transition from operational interpretation and for composing effects modularly.[1]
Distributed graphs and evidence. Workflow/DAG engines (e.g., Dryad, Spark, Ray) highlight typed dataflow and scheduling. Append-only logs, tracing, CRDTs, and long-running transaction patterns (SAGAs) supply auditability, convergence, and compensation strategies. Quota/rate control follows token/leaky-bucket patterns.

3 The $\mathcal{K}_{12}$ Episode Kernel

3.1 Definition

Definition 1 (K-12)**.** The episode kernel is the tuple

$$\mathcal{K}_{12} = \langle \Sigma,\ M,\ \mathcal{E},\ C,\ R,\ \kappa,\ \Pi,\ \Xi,\ \Lambda,\ \delta,\ \lambda,\ id \rangle$$

with the following roles:

- Σ: in-episode agent state; M: durable memory (replicated; CRDT-compatible); $\mathcal{E}$: effect surface for *all* external I/O (tools, files, nets); C: capabilities (least-privilege resource descriptors); R: deterministic randomness;
- κ: monotone quota (budget for tokens/time/$); Π: policy monitor (allow/deny/redact; pre/post-conditions); Ξ: observability (metrics, tracing spans, structured logs); Λ: locale (jurisdiction/residency tags; forks may only *narrow*);
- $\delta : \Sigma \times \text{Msg} \to \Sigma$ and $\lambda : \Sigma \times \text{Msg} \to \text{Out}$ are *pure*; id is an immutable episode identifier.

3.2 Laws (selected)

Law 1 (Effect containment)**.** All externally effectful operations occur via $\mathcal{E}$; δ, λ are pure and total on their domains.

Law 2 (Audit-before-mutation)**.** Any mutation of M or external write via $\mathcal{E}$ must be *preceded* by an append to the episode ledger (recording id, capability, policy epoch, model identity, inputs/outputs digests).

Law 3 (Monotone quota)**.** All counters in κ are monotone non-decreasing; attempts to spend with $\kappa \leq 0$ are rejected by Π.

Law 4 (Locale narrowing)**.** For any forked sub-episode e', $\Lambda(e') \subseteq \Lambda(e)$; operations violating locale constraints are denied by Π at call time.

[1] Canonical references appear in the bibliography.

3.3 Operational semantics (sketch)

We model an episode configuration as $(\Sigma, M, \Lambda, \kappa, \Pi, \ldots)$ with small-step rules:

$$\frac{\lambda(\Sigma, m) = o \quad \texttt{pre}(\Pi, \Lambda, \kappa, o)}{(\Sigma, M) \xrightarrow{m/o} (\delta(\Sigma, m),\ M')}$$

where `pre` encodes policy/quota/locale checks and M' results from the audited effect if any. The *only* side effects are those mediated by $\mathcal{E}$ and accompanied by ledger appends.

3.4 Derived properties

Deterministic replay boundary. Given the recorded randomness seed(s), model identity, policy epoch, locale Λ, and the multiset of effect responses, the transcript is reproducible; non-determinism beyond $\mathcal{E}$ is excluded by construction. **Exactly-once within retry horizon.** With idempotent ledger appends and deduplicated outbox/inbox, non-idempotent tool effects are applied at most once per idempotency key (proof sketch in §7).

3.5 Assumptions

Unless stated otherwise, we assume: (1) at-least-once delivery of messages to nodes; the router enforces per-key order; (2) deterministic RNG with recorded seeds/offsets; δ, λ are pure; (3) the ledger provides idempotent `append` with durable ordering per key; (4) policy monitors (Π) are total and run before external effects; locale tags can only narrow on forks.

4 The $\mathcal{G}_9$ Graph Kernel

4.1 Purpose

$\mathcal{G}_9$ orchestrates a set of $\mathcal{K}_{12}$-compliant nodes into a typed, analyzable graph. Edges carry typed messages; orchestration slots elevate control-plane concerns to first-class components.

4.2 Definition (slots and laws)

A $\mathcal{G}_9$ deployment is a tuple with nine roles:

1. Typed node/edge set of $\mathcal{K}_{12}$ episodes.
2. *Router*: message dispatch with per-key ordering guarantees.
3. *Scheduler*: progress/fairness; bounded iteration on cyclic subgraphs.
4. *Ledger*: append-only, tamper-evident event log (episode/edge/tool events).
5. *Metrics*: structured counters, histograms, spans.
6. *Monitor*: graph-level Π that cannot be bypassed by nodes.

7. *Topology*: static checks (type compatibility, guarded SCCs).
8. *Ingress/Egress*: schema-bound boundaries with capability checks.
9. *Registry*: model/adapter identities and validation artifacts.

Graph laws. (1) *No hidden I/O*: All edge I/O is mediated by $\mathcal{E}$ and recorded in the ledger. (2) *Guarded cycles*: Strongly connected components (SCCs) must declare bounds or convergence witnesses (e.g., decreasing variant). (3) *Policy non-bypass*: All ingress/egress pass through Π; monitor verdicts (allow/deny/redact/require-human) are binding. (4) *Ordered delivery per key*: The router preserves per-key order; dedup is keyed by (graph,node,key).

Router QoS contract. The router exposes a per-key *serialization domain* such that any two messages with the same $(\text{graph}, \text{node}, \text{key})$ are delivered in program order or not at all. Deduplication is with respect to span or message keys.

Lemma 1 (Monitor non-bypass)**.** If all ingress/egress edges are mediated by the monitor and every external effect requires a ledger append recorded as an edge event, then no node can cause an externally visible effect without a monitor verdict.

Proof sketch. Any effect requires $\mathcal{E}$ and a prior append (K-12 Law); edge I/O flows through $\mathcal{G}_9$; monitors gate ingress/egress. Composition enforces the check-before-effect discipline on all outward actions.

4.3 Safety and liveness

Theorem 1 (Deadlock-freedom under guarded cycles). If every SCC has either (a) a decreasing well-founded measure or (b) a finite iteration bound, and the scheduler is fair, then every run either progresses or reaches quiescence (no enabled edges).

Theorem 2 (Exactly-once effects within a retry horizon). Assume: (i) ledger append is idempotent with a unique span key, (ii) outbox/inbox provide at-least-once delivery with dedup by span key, and (iii) non-idempotent tool effects are conditioned on unseen span keys. Then non-idempotent effects apply at most once per key despite crashes and retries.

5 The $\mathcal{M}_7$ Market Kernel

5.1 Seven primitives

We model cross-organization invocation and settlement with seven primitives:

1. **Directory** (capability discovery).
2. **Offer** (quoted terms and constraints).
3. **Contract** (capability + policy + budget binding).
4. **Invoke** (remote graph execution via typed edge).

5. **Evidence** (tamper-evident transcript: model IDs, policy epoch, inputs/outputs digests).
6. **Settle** (atomic settlement; escrow-compatible).
7. **Repute** (optional reputation signal for future offers).

The primitives compose *over* $\mathcal{G}_9$: Directory→Offer→Contract→Invoke→Evidence→Settle, with failure paths routed to compensations (SAGAs).

6 Worked Example

Consider two $\mathcal{K}_{12}$-nodes A and B with a guarded cycle: $A \to B \to A$. The scheduler enforces a bound N or a decreasing variant on a convergence metric.

1. A receives m_0, computes $o_0 = \lambda_A(\Sigma_A, m_0)$, passes `pre`, appends a span, performs an effect via $\mathcal{E}$, and emits m_1 to B.
2. B consumes m_1 (router preserves key order), repeats the audited step, and may emit m_2 back to A.
3. After N iterations or once the well-founded metric decreases to zero, the scheduler quiesces the SCC.

Ledger evidence ties each step to $(\text{graph}, \text{node}, \text{key})$; replay uses recorded responses to reproduce transcripts end-to-end.

7 Proof sketches and invariants

Effect containment. Immediate from the operational rule: all non-pure consequences require an $\mathcal{E}$ step checked by Π and recorded by the ledger.

Audit-before-mutation. Treat the ledger as the single source of durable intent; state mutation rules are guarded by the existence of a prior ledger append with a matching span id; otherwise the transition is blocked.

Monotone quota. Model κ as a commutative monoid homomorphism from event sequences to $\mathbb{N}^d_{\geq 0}$; the monitor denies transitions with insufficient residual budget.

Deterministic replay boundary. Given (R seeds, model identity hash, policy epoch, Λ) and a deterministic reduction of recorded effect-responses, the episode transcript is reconstructible by induction on the step relation.

Exactly-once within retry horizon. Standard outbox/inbox with idempotent append and dedup yields at-most-once application for non-idempotent operations. Ledger append forms the *commit point*; duplicates map to no-ops.

8 Evaluation

We propose a conformance suite and experiments:

1. **Replay determinism.** Inject seeds, crash/restart, and verify transcript equality.
2. **Budget monotonicity.** Fuzz invocation patterns; assert no decreasing counters and correct deny-on-exhaustion behavior.
3. **Locale narrowing.** Randomized graph branching; assert $\Lambda_{\text{child}} \subseteq \Lambda_{\text{parent}}$.
4. **Ledger completeness.** Mutation testing: attempt hidden I/O; verify monitor/ledger intercepts or denies.
5. **Exactly-once.** Fault injection around commit (kill-before-append / kill-after-commit) and show at-most-once externally-visible effect.
6. **SAGAs compensation latency.** Measure compensation tails under induced failures.

9 Type Discipline for Capabilities

Definition 2 (Capability type)**.** A capability is a pair (r, ϕ) where r names a resource and ϕ is a predicate over allowed operations and bounds (e.g., rate, budget, locales). The type of $\mathcal{E}$-mediated calls is indexed by (r, ϕ).

Law 5 (Least privilege)**.** For any episode, the capability context C contains only capabilities required by its declared graph edges and tools; Π rejects any call whose (r, ϕ) is not present or whose preconditions fail.

Theorem 1 (Capability monotonicity)**.** If sub-episode e' is forked from e, then $C(e') \subseteq C(e)$ and $\Lambda(e') \subseteq \Lambda(e)$. Consequently, e' cannot perform any effect that e could not perform under the same policy epoch.

Proof sketch. Forking copies and narrows contexts; monitor checks are enforced at call time.

10 Static Checks & Conformance

Graph linting. Validate (i) type compatibility of edges, (ii) SCCs have declared guards, (iii) ingress/egress traverse the monitor, (iv) capability requirements are satisfied by node declarations.

Conformance suite. Provide executable tests that (a) attempt hidden I/O, (b) attempt locale-widening, (c) attempt budget underflow, and (d) exercise exactly-once around commit points; all must be rejected or deduplicated by the runtime.

11 Implementation Status

We maintain a reference interpreter that exposes $\mathcal{K}_{12}/\mathcal{G}_9$ as TypeScript interfaces and enforces the laws in this paper: effect containment, audit-before-mutation, budget monotonicity, and locale narrowing. A conformance CLI runs the evaluation experiments in §7 and §9.

Appendix: Conformance Test Inventory (Text)

K-01 Effect containment; K-02 Audit-before-mutation; K-03 Budget monotonicity; K-04 Locale narrowing; G-01 Guarded SCC termination; G-02 Router per-key ordering; G-03 Monitor non-bypass; M-01 Evidence completeness; M-02 Settlement idempotency; M-03 Cross-org locale preservation.

12 Related Work

Typed effects and effect handlers offer principled composition of effects and separate pure semantics from operational interpretation. Dataflow systems and tracing infrastructures illustrate graph scheduling, fault-tolerance, and observability. CRDTs formalize replica convergence; SAGAs handle long-running compensations; token/leaky-bucket schemes underpin robust quota enforcement. Our kernels unify these ideas into minimal, checkable runtimes specialized for LLM-driven agent systems.

13 Limitations and Future Work

We do not claim semantic correctness of LLM outputs; instead, we bound and audit the *process*. Future work includes (i) mechanized proofs in a small-step semantics, (ii) synthesizing least-privilege capabilities from graph specs, (iii) integrating content-provenance manifests for generated media by default, and (iv) formalizing cross-organization attestations and settlement as reusable protocol modules.

14 Conclusion

K-12/G-9/M-7 isolate the essential laws for reliable agent execution. By lifting governance, evidence, and liveness into kernels with typed effects, we make multi-agent systems analyzable and auditable by construction. The reference implementation and conformance suite enable reproducible deployments and third-party verification.

References

[1] G. Plotkin and M. Pretnar. Handlers of Algebraic Effects. In *ESOP*, 2009.

[2] A. Bauer and M. Pretnar. Programming with Algebraic Effects and Handlers. *arXiv:1203.1539*, 2012.

[3] M. Shapiro et al. Conflict-free Replicated Data Types. INRIA RR-7687, 2011.

[4] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD*, 1987.

[5] B. H. Sigelman et al. Dapper: A Large-Scale Distributed Systems Tracing Infrastructure. Google TR, 2010.

[6] M. Isard et al. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EuroSys*, 2007.

[7] M. Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*, 2012.

[8] P. Moritz et al. Ray: A Distributed Framework for Emerging AI Applications. In *OSDI*, 2018.

[9] J. Heinanen and R. Guerin. A Single Rate Three Color Marker. IETF RFC 2697, 1999.

[10] N. Hardy. The Confused Deputy (or Why Capabilities Might Have Been Invented). *ACM SIGOPS*, 1988.

[11] M. S. Miller and J. S. Shapiro. Capability Myths Demolished. HP Labs TR, 2003.

[12] A. Pnueli. The Temporal Logic of Programs. In *FOCS*, 1977.

[13] C. Baier and J.-P. Katoen. Principles of Model Checking. MIT Press, 2008.

[14] P. Helland. Life Beyond Distributed Transactions: An Apostate's Opinion. In *CIDR*, 2007.

[15] P. Helland. Idempotence Is Not a Medical Condition. *ACM Queue*, 2012.

[16] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

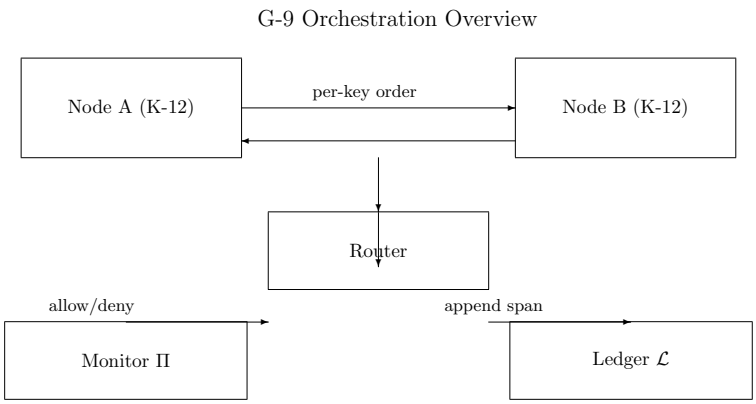

Figure 1: G-9 orchestration: K-12 nodes connected by typed edges, with router (per-key order), monitor Π enforcing policy, and ledger $\mathcal{L}$ for append-only evidence.