The Kernel Trilogy

K-12, G-9, and M-7 source specifications · archival working edition

Ryan Hunter, with AI research collaborators

2025–2026 · public source edition September 2026

These three source specifications describe one bounded agent episode, coordination among episodes, and exchange across organizations. They are presented as archived working designs. Some later work narrows claims in the M-7 draft; the accompanying essay is the current plain-language guide.

K-12 — Episode Kernel (Core)

Goal. Define the minimal environment (10 slots) and laws for one bounded Episode: given a message, produce a (possibly streaming) output under quotas, policy, and auditing.

Terminology Note — Exactly-Once Claim

K-12 documents idempotency and audit ordering only; the "exactly-once within a retry horizon" guarantee is defined in `spec/idempotency.md` and bound to *configured TTLs and retry windows*. Implementers MUST avoid stronger claims in user-facing docs. See RFC-0012 §3 and §5 for precise language.

---

1. Slots (Tags)

Symbol	Name	Purpose
Σ	Ephemeral	`FiberRef<State>` scratch-pad for the episode
M	Memory	merge-safe facts (CRDT / append-only facts)
E	ToolRegistry	effectful APIs invoked by the episode
L	Ledger	append-only audit trail (hash-chained events)
C	Capabilities	scoped entitlements (structured tuples; see `capabilities.md`)
R	RNG	deterministic randomness
κ	Quota	token/$/latency budgets; counters are monotone
Π	Policy	monitor to redact/reject IO (content, privacy, safety)
Λ	Locale	set of residency/jurisdiction tags; forks may only narrow
δ/λ	Transition/Act	pure control & output; λ streams chunks to caller
id	Identity	DID + attestation note (model/code digests, optional)

Σ Scope. Σ is strictly per-episode and MUST NOT be migrated across revisions; durable state belongs in M.
Slot Count. K-12 defines 10 environment slots (Σ, M, E, L, C, R, κ, Π, Λ, id). δ/λ are not slots; they are the episode's pure transition/act functions that run over the provided environment.
Approvals (HITL). Human-in-the-loop gating is exposed via the Approvals service, layered alongside E/Π/κ. Episodes MUST consult Approvals whenever Sequencer policy or K-12X Gate G1 requires manual consent; implementations typically provide the auto/audited/mailbox layers from the Approvals module.

---

2. Laws (Normative)

1. Purity: `δ, λ` are pure and deterministic: for any fixed `(Σ, msg)` and fixed M snapshot, they produce the same outputs and state deltas without observing external time, randomness, or IO. (All IO must pass via E.)

2. Effect containment: All IO passes through E; memory writes tag data with current Λ.
3. Quota monotonicity: `κ.consume(n)` is monotone decreasing within an episode.
4. Locale inheritance: Forked episodes MUST have `Λ_child ⊆ Λ_parent`. Capabilities granted in the child episode MUST be valid within `Λ_child`.
5. Audit order: A span MUST be written to L before mutating Σ or M. If L is unavailable or the append fails, the Episode MUST fail closed (no mutation and no egress).
6. Streaming output: λ MAY return an `AsyncIterable<Chunk>`; internal events go to L by default, not to the caller.
7. Capability validation: All capability grants/revokes MUST be validated against the current Locale (Λ) set. Policy Π MUST reject capability operations where the capability is not authorized within the active Λ domain.

Profile Gates (K-12X). Jurisdiction/UX-driven gates (HITL for consequential decisions, AI disclosure, appeals, synthetic provenance, consequential transparency) are specified in `spec/k12x-gates.md` and are enforced only when enabled by Π/Λ rules or explicit attributes per RFC-0016.

---

3. Episode Type (Normative, Effect A-E-R)

```ts
export type Episode =
  (msg: Msg) => Effect<
    AsyncIterable<Chunk>,          // A: output (user-visible stream)
    AgentError,                    // E: error
    K12Env                         // R: required Tags (Σ, M, E, L, C, R, κ, Π, Λ, id)
  >
```

3.1 Stream Chunk Envelope (Normative)

To ensure interop and back-pressure aware streaming, λ SHALL emit frames matching the canonical envelope:

```ts
export type Chunk =
  | { kind: "text";  delta: string }
  | { kind: "json";  delta: unknown }
  | { kind: "action-request"; action: string; arguments: unknown; idempotencyKey: string }
  | { kind: "action-result"; action: string; outcome: unknown; idempotencyKey: string }
  | { kind: "action-error";  action: string; error: { code: string; message: string; retryable: boolean };
  | { kind: "system"; event: "start" | "heartbeat" | "eos" | "cancel" | "error"; detail?: unknown }
```

Implementations MAY extend with additional kinds if consumers understand them, but MUST NOT break the above semantics.

3.1.1 Transport mappings (Normative / Informative) SSE:

- `event:` names map as: `text`, `json`, `action-request`, `action-result`, `action-error`, `system`.
- The data field is a JSON string of the `Chunk`.

gRPC (proto oneof sketch):

```
message Chunk {
  oneof kind {
    Text text = 1;
    Json json = 2;
    ActionRequest action_request = 3;
    ActionResult action_result = 4;
    ActionError action_error = 5;
    System system = 6;
  }
}
```

- Flow control MUST honor back-pressure; producers MUST NOT buffer unboundedly.

---

4. Λ (Locale) Rules (Normative)

- Λ is a finite set (e.g., `{ EU, DE }`).
- Copy on fork; only narrowing is permitted between parent and child.

- Policy Π MUST reject any read/write where `locale(data) ⊄ Λ_current`.
- Delegation to sub-agents MUST satisfy `Λ_child ⊆ Λ_parent`, or transform/redact data so derived facts are legal in the target Λ.

---

5. What λ streams vs. what L records (Informative)

- Primary stream: user-visible tokens/chunks from λ.
- Side-channel: action requests/results, budget ticks, spans → L (durable ledger). Implementations MAY expose a dev mirror of side-channel events.

---

6. Cancellation, Back-pressure & ResourceScope (Normative)

- Cancellation: An Episode MUST promptly honor fiber interruption and deadlines. On cancel, λ MUST close the output stream and emit `Chunk { kind: "system", event: "cancel" }`.
- Back-pressure: λ's stream MUST NOT buffer unboundedly. If the consumer stalls, either bounded buffering (`dropOldest` documented) or suspend production.
- ResourceScope: E MUST provide finalizers for acquired resources (connections, file handles, temp files). All finalizers MUST run on episode termination, including cancellation and failure.

---

7. Error Taxonomy (Normative)

Episodes MUST classify failures into:
- `PolicyViolation` — Π rejection. Non-retryable.
- `QuotaExceeded` — κ budget exhausted. Retryable after budget replenishment.
- `ToolTransient` — transient E failure. Retryable.
- `ToolFatal` — deterministic E failure. Non-retryable.
- `CallerCancel` — cancelled by upstream/deadline. Non-retryable.

Schedulers MUST map these to policies (e.g., backoff for transient; dead-letter for fatal). See `errors.md`.

---

8. Right-to-Erasure & Redaction (Normative)

- Memory (M): MUST support redaction tombstones that logically remove personal data while preserving merge integrity. See `memory.md`.
- Ledger (L): MUST support redaction events that either (a) encrypt personal payloads with erasable keys or (b) store only references and remove blobs upon erasure, leaving a verifiable audit trail. See `ledger.md`.

---

9. Idempotency (Normative)

Tool calls MUST carry idempotency keys and record them in L. See `idempotency.md` for key derivation, dedup windows, and outbox/inbox patterns.

G-9 — Graph Kernel (Core)

Goal. Typed coordination of many Episodes/Personas with durability, scheduling, and routing.

---

1. Slots

- N — nodes (finite set)

- E — edges (directed relation)
- Φ — flow types mapping `node → (Input, Output)` (schemas)
- R — Router (channels, delivery)
- S — Scheduler (ready-node selection strategy)
- L — Ledger (append-only event log)
- Ω — Metrics (counters, timers, histograms)
- M — Monitor (CTL/LTL predicate checker)
- id — Graph metadata (identifier and revision)

---

2. Axioms (Normative)

1. Typing: For every `(u, v) ∈ E`, `Φ(u).O ≅ Φ(v).I`.
2. Cycles: If a directed cycle exists, a termination predicate MUST be declared and verifiable by M (or a bounded-iteration limit explicitly configured).
3. Routing idempotence: `R.send` MUST be idempotent with respect to `L.append` to enable safe retries.
4. Join semantics: Fan-in nodes MUST declare `all|any|k-of-n` and a deduplication strategy (idempotency key or sequence).

---

3. Graph Port (Normative)

```ts
export interface GraphPort<I, O> {
  run(input: I): Promise<O>
  describe(): GraphDescription    // JSON AST { nodes, edges, schemas, meta }
}
```

`GraphDescription` MUST conform to `spec/schemas/GraphDescription.schema.json`. The `enforceGraphLaws` procedure MUST validate typing, adapters, and cycle guards before execution.

Describe() as Execution Truth. A scheduler MUST NOT execute behavior that cannot be derived from `describe()`; adapters MUST appear as first-class nodes with stable `witness` identifiers so that execution can be reproduced and audited.

---

4. Schema Evolution & Adapters (Normative)

- Structural subtyping: An output `O'` MAY flow into an input `I` if `O'` is a structural subtype of `I`, or an explicit adapter exists for `O' → I`.
- Adapters as nodes: Adapters are first-class nodes with a stable `witness` identifier; usage MUST be recorded by `describe()` and persisted in L for reproducibility.
- Validation: Missing adapters or incompatible types MUST cause `enforceGraphLaws` to fail.

---

5. Scheduling & Parallelism (Normative)

- Readiness: A node MUST NOT be scheduled until all required predecessors have successfully recorded outputs in L (guards considered).
- Strategies: Implementations SHOULD support at least one of: `fifo`, `fair`, `priority`, `deadline`.
- Partition ordering: For messages sent with a partition key, delivery to a given successor MUST preserve ordering per key.
- Concurrency limits: Per-node concurrency MUST be configurable; cancellation MUST propagate downstream.
- Join semantics: Fan-in nodes MUST declare whether they require `all`, `any`, or `k-of-n` predecessors, and how to de-duplicate by idempotency key or sequence number.

---

6. Idempotency & Exactly-Once (Normative)

See `idempotency.md` for key derivation, dedup windows, and outbox/inbox patterns. Briefly:

- Router: at-least-once delivery with dedup store keyed by `idempotencyKey` (TTL ≥ retry horizon).
- Ledger: `append` MUST be idempotent on `idempotencyKey` (store once or flag duplicates).

---

7. Execution (Informative)

The scheduler picks ready nodes, the router delivers inputs, nodes execute in isolated Effect scopes (with their own K-12 environments), outputs are broadcast to successors, and the ledger persists a `LedgerEvent` as per schema. Monitors may block/route based on predicates; lineage in meta enables hot-swap.

Join Semantics Reminder. For any fan-in, explicitly declare `all|any|k-of-n` and the de-dup strategy keyed to upstream outputs; omission MUST fail preflight. See §2.4 and §5 for the normative rules and scheduler interaction.

M-7 Market Kernel Specification

Version: v1.0 (Consolidated from Swarm Outputs) Status: Normative Companion to: K-12 Episode Kernel, G-9 Graph Kernel Math References: Rosetta 14e (Game Theory/IC), 14j (Quantales for Costs/Additivity)

Abstract

M-7 equips K-12 episodes and G-9 graphs with economic rails for cross-organization agent systems: discovery (D), attestation (A), offers/contracts (a), metering (β), settlement (σ), dispute (Δ), and reputation (ρ). β derives from Ledger (L) events for retry-safety; prices are subadditive/monotone; IC via commit-charge or attempt+rebate. Ports are Effect TS Context.Tags, with layers for adapters (in-memory/cluster). Semantics: Priced traces over G-9 LTS, quotiented for retries. Proofs: No double-charge (β from idempotent L), additivity (monoid), IC (expected utility), bisimulation (weak under stuttering).

Mathematical Sufficiency

M-7 conveys the platonic shape of economic rails as a functor F: Traces → Economies, where Traces is the category of G-9 LTS (states as graphs, morphisms as priced transitions), and Economies is the category of quantales ([0,∞]^d, ⊕, 0) for multi-dimensional costs. The 7-port tuple encodes F's components: D/a as discovery/contracting (objects), β as the additive functor (metering), σ/Δ as attested morphisms (settlement/dispute), ρ as the feedback monad (reputation decay). Laws are natural transformations preserving structure (14c), ensuring IC (Nash-stable strategies, 14e) and verifiability (traces quotients to proofs). This minimal form is sufficient for the ideal: composable, verifiable coordination over stochastic traces, scaling O(1) per unique event under assumptions (honest nodes, finite horizons).

Platonic Shape

The tuple ⟨D, A, a, β, σ, Δ, ρ⟩ is the irreducible product in a monoidal category, with laws as guardians: Law 2 (additivity) ensures compositionality (markets as Kleisli category), Law 5 (IC) stability (Nash via Folk theorem), Law 6 (Λ×Π) boundaries (lattices for locales/policies, 14h). Gaps (e.g., ZK for privacy, multi-party escrows) are extensions, not flaws—v1 approximates the ideal via ledger-derived proofs.

Motivation

Agent systems scale to markets: agents discover capabilities, attest trust, contract SLAs, meter usage, settle payments, resolve disputes, and build reputation. M-7 provides verifiable economics without central trust, deriving from K-12/G-9 (e.g., β from L unique events, σ using idempotency keys).

Definition: 7-Port Tuple

M-7 is the product ⟨D, A, a, β, σ, Δ, ρ⟩, with laws ensuring ledger-derived, IC economics.

Ports

- D Directory: find/resolve offers by capabilities/Λ.
- A Attestation: issue/verify proofs (e.g., sign invoices).
- a Offers/Contracts: publish/accept with typed I/O, subadditive/monotone prices, SLAs.
- β Metering: Derive additive from unique L events (monoid, 14j).
- σ Settlement: Invoices from β×a, attested, idempotent pay/refund.
- Δ Dispute: Ledger-derived resolution (event diffs).
- ρ Reputation: Monotone updates from Δ, query-time decay.

Laws (Normative)

1. Ledger-Derived Metering: β from unique L events (keys ensure 1-safeness).
2. Additivity: β commutative/associative post-dedup.
3. Attested Settlement: σ invoices with L hash proofs.
4. Contract Soundness: Typed I/O, subadditive/monotone P, monitorable SLAs.
5. IC Pricing: Commit-charge or attempt+rebate (unrebated attempt non-IC).
6. Λ×Π Compliance: Enforce locales/policies in ports.
7. Δ→ρ Monotonicity: Updates from disputes, query-time decay.
   Lemma: σ idempotent on (invoiceId, key) via outbox/inbox.

Effect TS Mapping

Ports as Context.Tags; layers compose adapters.

Port	Tag	Methods
D	DirectoryTag	find(q: CapabilityQuery): Effect<OfferHead[]>; resolve(id: OfferId): Effect
A	AttestationTag	issue(c: VerifiableCredential): Effect; verify(p: Proof, hash: string): Effect
a	OffersTag	publish(o: Offer): Effect; accept(id: OfferId, opts?: Partial): Effect; get(id: ContractId): Effect
β	MeteringTag	record(e: UsageEvent): Effect<void, MeterError, LedgerTag>; rollup(f: MeterFilter): Effect<MeterVector, MeterError, LedgerTag>
σ	SettlementTag	invoice(c: ContractId, window: Interval): Effect<Invoice, SettlementError, [LedgerTag, AttestationTag, OffersTag]>; pay(inv: Invoice, proof?: PaymentProof): Effect<Receipt, SettlementError>; refund(inv: Invoice, key?: string): Effect<CreditNote, SettlementError, LedgerTag>
Δ	DisputeTag	file(c: Claim): Effect<CaseId, DisputeError, LedgerTag>; resolve(id: CaseId, v: Verdict): Effect<Award, DisputeError, LedgerTag>
ρ	ReputationTag	update(e: ReputationEvent): Effect<void, never, LedgerTag>; score(p: PartyId, atTime?: Instant): Effect

Layers: layerMarket = Layer.mergeAll([Metering, Settlement, ...]); adapters (InMem/Cluster/Browser).

Operational Semantics

Configuration: ⟨τ (G-9 trace), M (MeterVector), C (Contracts)⟩.
- Meter Step: For chargeable ℓ in τ with key k (unseen): M ⊕ m(ℓ).
- Settlement: Σ(W) = $\sum$_c P_c(β_c(W)), with β additive, P subadditive/monotone.
- Quotient: Traces modulo τ (retries/stuttering); weak bisimulation to durable workflows.

Properties/Proofs

- No Double-Charge: Per-key 1-safeness + dedup ⇒ β counts once (14f Petri).
- Additivity: β monoid homomorphism (14j quantale).
- IC Pricing: Commit/rebate aligns EU; unrebated attempt non-IC (expected utility sketch: p=0.7, m=3 ⇒ E[A]=1.39 attempts, rebate restores alignment).
- Bisimulation: Priced traces ~ workflows modulo τ (14l coalgebra).

Appendices

A. Schemas (Effect.Schema)

- UsageEvent: {idempotencyKey, graphId, nodeId?, meter: Partial, time}.
- PriceSchedule: {quote(m: MeterVector): Money, currency, terms: {commitCharge?: boolean, attemptChargeRebate?: boolean}}.
- Invoice: {id, contractId, window, meter, amount, inclusionProof}.
- Contract: {id, offerId, terms: PriceSchedule, sla: MonitorablePredicates}.

B. Adapters

- InMemMarketLive: Array/map stubs.
- ClusterMarketLive: SQL for D/a, Stripe mock for σ.
- BrowserMarketLive: LocalStorage for β/ρ.

C. Conformance

- β additivity: Retry sim, assert meter=1.
- σ idempotency: Double-pay test, assert receipt unchanged.
- IC: Simulate failures, assert rebate/commit alignment.

D. Mathematical Sufficiency

M-7 conveys the platonic shape of economic rails as a functor F: Traces → Economies, where Traces is the category of G-9 LTS (states as graphs, morphisms as priced transitions), and Economies is the category of quantales ([0,∞]^d, ⊕, 0) for multi-dimensional costs. The 7-port tuple encodes F's components: D/a as discovery/contracting (objects), β as the additive functor (metering), σ/Δ as attested morphisms (settlement/dispute), ρ as the feedback monad (reputation decay). Laws are natural transformations preserving structure (14c), ensuring IC (Nash-stable strategies, 14e) and verifiability (traces quotients to proofs). This minimal form is sufficient for the ideal: composable, verifiable coordination over stochastic traces, scaling O(1) per unique event under assumptions (honest nodes, finite horizons).

Platonic Shape

The tuple ⟨D, A, a, β, σ, Δ, ρ⟩ is the irreducible product in a monoidal category, with laws as guardians: Law 2 (additivity) ensures compositionality (markets as Kleisli category), Law 5 (IC) stability (Nash via Folk theorem), Law 6 (Λ×Π) boundaries (lattices for locales/policies, 14h). Gaps (e.g., ZK for privacy, multi-party escrows) are extensions, not flaws—v1 approximates the ideal via ledger-derived proofs.

At cross-organization scale, agents transact capabilities and usage. M-7 defines ports (interfaces) to enable discovery, attestation, contracting, metering, settlement, and reputation. Laws will be added as the ecosystem matures.

Slot	Meaning
D	Directory — capability discovery (public metadata, endpoints)
A	Attestation — code/model/TEE proofs; verifiable credentials
Gamma	Offers/Contracts — typed offers, SLAs, rights, prices; capability leases
β	Metering — standard counters (tokens/latency/bytes) to back invoices
σ	Settlement — payment rails, escrow, refunds
Δ	Dispute/Policy — arbiters, sanctions lists, policy packs
ρ	Reputation — signed ratings, slashing; sybil resistance hooks

Ports (sketch):

```
export interface Directory {
  find(q: CapabilityQuery): Promise<AgentOffer[]>
}
export interface Metering {
  record(e: UsageEvent): Promise<void>
}
```

```
export interface Settlement {
  invoice(e: UsageEvent[]): Promise<InvoiceId>
}
```

Binding to G-9: Metering SHOULD derive from G-9 ledger events.

Source documents and references

1. Ryan Hunter, *K-12 Episode Kernel*, Foundation3 working specification.
2. Ryan Hunter, *G-9 Graph Kernel*, Foundation3 working specification.
3. Ryan Hunter, *M-7 Market Kernel*, Foundation3 working specification.
4. Ryan Hunter, *Typed-Effect Kernels for Reliable Agent Systems*, August 2025 working preprint; the companion PDF in this library.

Source edition notes

The three specifications were copied from the Foundation3 source mine into a single public reading edition. Some mathematical glyphs were transliterated to ASCII in the source edition; the LaTeX render restores math symbols where the paper font requires it. Names of companion specifications refer to documents in the original archive; they are plain text here because those files are not part of this public edition.