Structured Knowledge Layer

SKL — Complete Reference Specification v1.4

A Coordination Layer for Small Concurrent LLM Agent Teams

Governing Principle

> *SKL guarantees that coordination-relevant uncertainty, assumptions, and cross-scope dependencies cannot propagate implicitly. It is not a safety system. It is a fail-stop coordination system with bounded damage and explicit uncertainty.*

1. What SKL Is and Is Not

SKL is an auditability layer and a mistake-surfacing system for small teams of concurrent LLM coding agents. Its goal is narrow: make parallelism debuggable, not safe. Every design decision reflects that goal.

1.1 What SKL Guarantees

- Coordination-relevant uncertainty cannot propagate implicitly across agents.
- Shared assumptions between agents cannot remain implicit — conflicts trigger review before merging.
- Cross-scope dependencies cannot be silently introduced — the enforcement hook detects them mechanically.
- Architectural decisions cannot be implemented without mechanically checkable acceptance criteria.
- When things go wrong, they do so locally, transparently, and with bounded blast radius.

1.2 What SKL Will Never Claim

> *SKL must never claim to prevent bad code from being merged. It must never claim formal or probabilistic correctness guarantees for general-purpose software. Any future extension requiring either of those claims is outside the scope of this system.*

- SKL does not prevent agents from making bad implementation decisions within their scope.
- SKL does not ensure the codebase is well-designed.
- SKL does not scale to large codebases without modifications described in Section 7.
- SKL does not solve general verification — it solves coordination transparency.

1.3 Design Target

Parameter	Value	Notes
Concurrent agents	2 – 6	Queue review degrades above 6
Codebase size	Up to ~80 State entries	Beyond this, State needs query layer
Task complexity	Well-defined feature work	Deep refactors require human Orchestrator
Primary language	Python 3.x (v1.4)	Multi-language requires additional hook work

2. Repository Layout

All SKL artifacts live in a .skl/ directory at the repository root. Every file is human-readable and version-controlled via Git.

```
.skl/
├── knowledge.json          # Core shared memory (State, Queue, Invariants)
├── scope_definitions.json  # Formal semantic scope taxonomy
├── rfcs/                   # Open and resolved RFCs
│   ├── RFC_001.json
│   └── RFC_002.json
├── adrs/                   # Architecture Decision Records (append-only)
│   └── ADR_001.json
├── orchestrator_log/       # Per-session Orchestrator handoff logs
│   └── session_001.json
└── scratch/                # Per-agent ephemeral notes (deleted at session end)
    ├── agent_1.md
    └── agent_2.md
```

3. The Knowledge File — knowledge.json

The single source of truth for the system. Four top-level sections. Updated exclusively through Git commits, making every change auditable.

3.1 Invariants

Read-only facts that cannot change without explicit human action. Set by the human operator at project initialization. No agent or Orchestrator may modify them. Changes require a direct human commit with an explanatory message.

> *Risk: Invariants are only as good as their initial definition. The system has no mechanism to detect when the codebase has drifted from what Invariants describe. This is a human discipline problem, not a technical one.*

```
"invariants": {
  "tech_stack": ["Python 3.11", "FastAPI", "SQLAlchemy 2.0"],
  "auth_model": "JWT over Bearer header",
  "data_storage": "PostgreSQL only",
  "security_patterns": [
    "@require_auth", "@login_required",
    "verify_", "check_permission", "is_authorized"
  ]
}
```

3.2 State

The live map of all registered files and modules. Written only by the Orchestrator. Each record uses a structured schema — not free text — to slow semantic drift.

3.2.1 Full State Record Schema

```
{
  "id": "router_auth",
  "path": "app/routers/auth.py",
  "semantic_scope": "auth",
  "scope_schema_version": "1.0",
  "responsibilities": "Login, token refresh, password reset endpoints.",
  "dependencies": ["app/utils/tokens.py", "app/schemas.py"],
  "invariants_touched": ["auth_model"],
  "assumptions": [
    {
      "id": "assume_jwt_secret_env",
      "text": "JWT secret is always available as an environment variable at
startup.",
      "declared_by": "Agent-1",
      "scope": "auth",
```

```
      "shared": false
    }
  ],
  "owner": "Agent-1",
  "version": 4,
  "uncertainty_level": 1,
  "uncertainty_reduced_by": "tests/test_auth.py",
  "last_reviewed_at": "2025-06-01",
  "change_count_since_review": 2
}
```

3.2.2 Field Definitions

Field	Type	Writable By	Description
id	string	Orchestrator	Unique identifier for this State entry
path	string	Orchestrator	Relative file path from repo root
semantic_scope	string	Orchestrator	Must match a key in scope_definitions.json
scope_schema_version	string	Orchestrator	Scope definition version used at last validation
responsibilities	string	Orchestrator	Plain-language description of file's single responsibility
dependencies	string[]	Orchestrator	Direct dependencies; validated by import scanner
invariants_touched	string[]	Orchestrator	Which Invariant keys this file's behavior depends on
assumptions	object[]	Orchestrator	Explicit assumptions agents declared about this module
owner	string	Orchestrator	Agent ID of last approved proposer
version	integer	Orchestrator	Increments on every approved change
uncertainty_level	0–3	Orchestrator / Human / CI	See Section 3.2.3 — can only decrease via external proof
uncertainty_reduced_by	string	CI / Human	Test file or check that last reduced uncertainty
last_reviewed_at	date	Human	Date of last human review of this entry
change_count_since_review	integer	Orchestrator	Increments per approved change; reset on human review

3.2.3 Uncertainty Levels

Uncertainty is a first-class invariant. It can only decrease through external proof or human action — never automatically.

Level	Name	Meaning	How to Reduce
0	Verified	Passed a mechanically checkable acceptance criterion	CI test/schema check passes; referenced in uncertainty_reduced_by
1	Reviewed	Approved and in active human-reviewed digest without concerns	Human marks review; default for approved entries
2	Proposed	Recently approved but not yet verified against any acceptance criterion	Default for new State entries; reduced by passing tests
3	Contested	Open assumption conflict, rejected RFC, or circuit-breaker trigger	Human must explicitly reduce; no automated merges permitted

> *Uncertainty level 0 is not a correctness certificate. A module at level 0 has passed a specific named check. It may still contain errors outside that check's scope.*

3.2.4 Routing Rules by Uncertainty Level

- Level 3: Proposal automatically blocked. Added to escalation queue. Human notified immediately.
- Level 2: Mandatory individual Orchestrator review regardless of change_type classification.
- Level 0–1: Standard review path applies.

3.3 Queue

Pending proposals submitted by agents. Each proposal is populated partly by the agent and partly by the enforcement hook.

3.3.1 Full Queue Proposal Schema

```
{
  "proposal_id": "prop_042",
  "agent_id": "Agent-1",
  "path": "app/utils/tokens.py",
  "semantic_scope": "auth",
  "scope_schema_version": "1.0",
  "change_type": "behavioral",
  "responsibilities": "Token generation for password reset links.",
  "dependencies": ["app/routers/auth.py"],
  "invariants_touched": [],
  "assumptions": [
    {
      "id": "assume_token_ttl_config",
      "text": "Token TTL is configurable via APP_TOKEN_TTL env variable.",
```

```
      "declared_by": "Agent-1",
      "scope": "auth",
      "shared": false
    }
  ],
  "uncertainty_delta": "+1",
  "rationale": "No existing utility covered secure reset token generation.",
  "out_of_scope": false,
  "cross_scope_flag": false,
  "branch": "feature/auth-reset",
  "risk_signals": {
    "touched_auth_or_permission_patterns": false,
    "public_api_signature_changed": false,
    "invariant_referenced_file_modified": false,
    "high_fan_in_module_modified": false,
    "ast_change_type": "behavioral",
    "mechanical_only": false
  },
  "classification_verification": {
    "agent_classification": "behavioral",
    "verifier_classification": "behavioral",
    "agreement": true,
    "stage1_override": false
  },
  "dependency_scan": {
    "undeclared_imports": [],
    "stale_declared_deps": [],
    "cross_scope_undeclared": []
  },
  "agent_reasoning_summary": "Considered using PyJWT directly in the router but
extracted to utility to keep router thin and allow reuse.",
  "status": "pending",
  "submitted_at": "2025-06-01T14:23:00Z"
}
```


3.3.2 change_type Values and Routing

Value	Meaning	Review Path
mechanical	AST-identical to base; only whitespace/comments changed	Eligible for auto-approval if stage1 confirms mechanical_only: true
behavioral	Function bodies change without signature changes	Individual Orchestrator review required
architectural	Class or function signatures change; new dependencies introduced	RFC gate triggered; human pre-clearance required

change_type is submitted by the agent but overridden by Stage 1 deterministic signals from the AST analysis. Agent self-classification is advisory, not authoritative.

3.4 Scratchpad

A directory at .skl/scratch/, one file per agent. Agents write freely. The Orchestrator does not read it. Deleted at session end. Agents may optionally promote reasoning notes to agent_reasoning_summary in their proposals before pushing — this is the only persistence mechanism for agent reasoning.

4. Scope Definitions — scope_definitions.json

Semantic scopes are formally defined and machine-validated. This prevents taxonomy drift — the gradual expansion of scope labels into catch-alls that lose enforcement value. Maintained by the human operator alongside Invariants.

> *Scope definition quality is a load-bearing assumption. A poorly written scope definition produces false confidence in enforcement. Scope definitions should be treated with the same care as Invariants and reviewed at the same cadence.*

4.1 Schema

```
{
  "scope_definitions": {
    "version": "1.0",
    "scopes": {
      "auth": {
        "description": "Authentication and authorization logic only.",
        "allowed_paths": [
          "app/routers/auth.py",
          "app/utils/tokens.py",
          "app/middleware/auth.py"
        ],
        "allowed_path_prefixes": ["app/auth/"],
        "forbidden_path_prefixes": ["app/models/", "app/infra/"],
        "permitted_responsibilities": [
          "Token generation and validation",
          "Login and logout endpoints",
          "Permission checking"
        ],
        "forbidden_responsibilities": [
          "Database schema management",
          "Email delivery",
          "Infrastructure configuration"
        ],
        "owner": "human-operator"
      },
      "persistence": {
        "description": "Database models, migrations, and query logic.",
        "allowed_path_prefixes": ["app/models/", "app/migrations/",
"app/repositories/"],
        "forbidden_path_prefixes": ["app/routers/", "app/infra/"],
        "permitted_responsibilities": [
          "SQLAlchemy model definitions",
          "Alembic migrations",
          "Repository pattern query methods"
        ],
        "forbidden_responsibilities": [
          "HTTP request handling",
```

```
            "Token generation",
            "Infrastructure provisioning"
          ],
          "owner": "human-operator"
        }
      }
    }
}
```

4.2 Taxonomy Governance

- When a scope becomes too broad, the human operator splits it and updates scope_definitions.json.
- All State records carrying the old scope label have change_count_since_review set to the review threshold, forcing surfacing in the next digest.
- Scope definition version is bumped; scope_schema_version in State records identifies entries created under an older definition.
- The system cannot detect when scope definitions have drifted from the actual architecture. Human audit is required at the same cadence as Invariant review.

5. The Enforcement Hook

A deterministic pre-push Git hook (~80 lines of Python). The most important component in the system. Removes the dependency on agent self-reporting for file-level changes and replaces it with mechanical verification. Runs automatically on every agent branch push.

> *Without the enforcement hook, the Queue is only as accurate as agents are disciplined. With it, the Queue is a complete record of everything that changed and why, regardless of agent behavior.*

5.1 Five Sequential Checks

The hook runs checks in this order, failing fast at checks 1, 2, and 5 before running the slower AST-based checks 3 and 4.

Check 1 — File Scope Validation

Diffs branch against main. Compares modified files against the agent's assigned file scope list. Any file modified outside the scope list triggers an automatic out-of-scope proposal appended to the Queue. The agent does not need to report this — the hook does it mechanically.

Check 2 — Semantic Scope Validation

Validates that all modified files fall within the allowed_paths and allowed_path_prefixes defined for the agent's assigned semantic scope in scope_definitions.json. A file that is in the assigned file scope but violates the semantic scope definition is flagged as a cross-scope modification and marked cross_scope_flag: true in the proposal.

Check 3 — AST Risk Signal Generation

For each modified Python file, runs an AST diff against the base version. Produces the risk_signals object for the proposal. Runs in under 1 second per file using Python's standard ast module.

Signal	Set When	Consequence
touched_auth_or_permission_patterns	Known security-sensitive identifiers found (configured in Invariants)	Mandatory individual review regardless of change_type
public_api_signature_changed	Function/class definition signature changed between base and head	Mandatory individual review regardless of change_type

Signal	Set When	Consequence
invariant_referenced_file_modified	Modified file appears in dependencies of any State entry with invariants_touched populated	Mandatory individual review regardless of change_type
high_fan_in_module_modified	Modified file appears as dependency in 3+ State records	Individual review with downstream impact check
ast_change_type	mechanical / behavioral / structural	Used in Stage 1 classification override
mechanical_only	ast_change_type is mechanical AND no other signals are set	Eligible for auto-approval

The security pattern list is only as good as what the human operator defines in Invariants at setup. Patterns not on the list are not detected. This is an accepted limitation.

Check 4 — Dependency Validation (Import Scanning)

For each modified file, extracts all static import statements using AST parsing and resolves them against known State record paths. Produces two lists appended to the proposal:

- undeclared_imports: imports found in code but absent from declared dependencies field. Logged as warnings, not blocking, unless they cross a semantic scope boundary.
- stale_declared_deps: dependencies declared in State record that no longer appear as actual imports. Logged as warnings.
- cross_scope_undeclared: undeclared imports that resolve to a file registered under a different semantic scope. Upgraded to blocking flag — requires Orchestrator review.

Import scanning only detects direct static imports. Dynamic imports, configuration-driven module loading, and indirect dependencies through shared state are not detected. The dependency field remains partially manual.

Check 4 adds approximately 1–3 seconds per modified file. For typical pushes of 3–8 files this is acceptable.

Check 5 — Queue Budget Enforcement

If the Queue contains more than the configured maximum of unresolved proposals (default: 15), the push is blocked. The agent receives a message instructing it to wait for the Orchestrator to process the Queue. This is a hard throttle on throughput.

6. Classification Verification

Agent self-classification is a trust boundary. Research on multi-agent systems identifies self-assessment as a primary failure mode, with approximately 20% of agent errors attributable to agents incorrectly evaluating their own outputs. SKL uses a two-stage process to reduce misclassification without introducing LLM judgment where deterministic checks are possible.

6.1 Stage 1 — Deterministic Overrides

Deterministic signals from the AST analysis override agent self-classification. The agent's submitted change_type is treated as advisory input. Stage 1 applies before any LLM judgment:

- If mechanical_only is true: proposal is classified as mechanical regardless of agent label.
- If touched_auth_or_permission_patterns, public_api_signature_changed, or invariant_referenced_file_modified is true: proposal is classified as at minimum behavioral regardless of agent label.
- If cross_scope_flag is true: proposal is classified as at minimum behavioral regardless of agent label.

6.2 Stage 2 — Verifier Pass for Ambiguous Proposals

When Stage 1 does not produce a deterministic override, the Orchestrator runs a verifier pass before reviewing the proposal. This is a structured prompt to the same LLM with a narrow task: classify the change as mechanical, behavioral, or architectural given the diff and the current State record, and output a one-sentence justification.

The Orchestrator receives both the agent's self-classification and the verifier classification. If they agree, the Orchestrator proceeds with the agreed classification. If they disagree, the Orchestrator treats the proposal as the higher-risk of the two and records both in the Queue entry.

> *The verifier is a second LLM judgment call, not a deterministic check. Two LLMs can agree on a wrong classification. The verifier reduces the error rate by introducing independent judgment; it does not eliminate it. Deterministic Stage 1 signals are the more reliable check for high-stakes proposals.*

6.3 Circuit Breaker

If a single agent has produced three or more classification disagreements in a session — cases where verifier and agent disagreed — the Orchestrator flags that agent's subsequent proposals for mandatory individual review regardless of classification. This is session-scoped and resets when the agent session ends. The threshold of 3 is configurable.

7. The Orchestrator

An LLM agent with a limited, defined role. It does not write code. It is a dispatcher, reviewer, and merger.

7.1 Responsibilities

- Task Assignment: Read feature request, consult State, produce scoped tasks for agents.
- Queue Review: Process proposals sequentially, apply classification verification, make approve/reject/escalate/RFC decisions.
- State Writing: The only entity that writes to the State section of knowledge.json.
- Session Handoff: Produce a structured session log before session ends.

7.2 Review Order for Each Proposal

The Orchestrator follows this ordering for every Queue proposal. Rationale quality is higher when classification and dependency questions are resolved before reasoning about approval:

1. Read risk_signals. Apply Stage 1 deterministic overrides.
2. If ambiguous, run verifier pass. Record both classifications.
3. Check proposing agent's circuit breaker status.
4. Review dependency_scan results. Flag cross-scope undeclared dependencies.
5. Check for assumption conflicts with other pending proposals.
6. Check for conflicts with existing State records.
7. Make decision: auto-approve / approve / reject / escalate / RFC. Write rationale.
8. Update State entry: increment version, set uncertainty_level, update owner.

7.3 Decision Types

Decision	Condition	Action
Auto-approve	mechanical_only: true AND no risk signals AND no assumption conflicts	Merge branch, update State, brief annotation in rationale
Approve	behavioral/architectural without conflicts or RFC triggers	Merge branch, update State, write full rationale
Reject	Conflicts with existing State entry	Discard branch; provide specific conflicting State record and re-scoping instructions
Escalate	Target module at uncertainty_level 3	Block proposal; notify human operator immediately; do not review

Decision	Condition	Action
RFC	Architectural proposal, invariant modification, or shared assumption conflict	Generate RFC document; block merge; notify human immediately

7.4 Rationale Requirements

Every non-auto-approval decision includes a written rationale committed alongside the State update. The rationale header includes a decision_type field of either implementation or architectural. Architectural rationales trigger a human digest entry. These rationales accumulate in commit history and constitute the architectural decision record of the project.

> *Silent Orchestrator choices are how architectural drift accumulates invisibly. The rationale field is not optional.*

7.5 Session Budget

Each Orchestrator instance operates within a session with hard limits to prevent context accumulation and quality degradation.

Limit	Default	Rationale
Maximum proposals reviewed	15	LLM reasoning quality degrades with accumulated context before throughput ceilings are hit
Maximum session duration	90 minutes	Prevents formulaic rationales from long continuous operation
Self-uncertainty threshold	2 consecutive uncertain proposals	Self-reported; triggers session reset
Manual reset	Human operator trigger	Available at any time

When a session ends, the Orchestrator writes a structured handoff log to .skl/orchestrator_log/session_NNN.json before closing. A new Orchestrator instance initializes from the current knowledge.json plus the most recent session log only — not the full review history of prior sessions. This prevents context accumulation across sessions.

7.5.1 Session Log Schema

```
{
```

```
  "session_id": "session_003",
  "proposals_reviewed": 12,
  "session_start": "2025-06-01T09:00:00Z",
  "session_end": "2025-06-01T10:23:00Z",
  "recurring_patterns_flagged": [
    "Agent-2 consistently misclassifies cross-scope utility imports as behavioral"
  ],
  "escalations": ["prop_038 escalated: uncertainty level 3 on router_auth"],
  "rfcs_opened": ["RFC_005"],
  "uncertain_decisions": ["prop_041 — two valid approaches; recommend human
review"],
  "circuit_breakers_triggered": []
}
```

8. Assumption Declaration

Agents are required to declare assumptions in their Queue proposals. An assumption is a belief about the state of the system that the agent's change depends on but does not itself verify.

> *Assumption declaration partially depends on agent self-reporting. An agent that fails to declare an assumption it holds is not caught by deterministic enforcement. The audit trail will reveal the gap after a failure, but not before. This is a residual limitation.*

8.1 The Shared Assumption Rule

The enforcement-relevant component. When the Orchestrator reviews Queue proposals and finds two or more proposals with potentially conflicting or mutually dependent assumptions — assessed by the verifier pass — both proposals are flagged and an RFC is triggered automatically.

- The RFC template for a shared assumption conflict contains both assumption statements, the modules involved, and a required resolution.
- Resolution options: promote the assumption to the Invariants section as a confirmed system invariant, or identify which module's assumption is wrong and require correction before either proposal merges.
- False positives — superficially similar assumptions that are actually independent — produce unnecessary RFCs. This is overhead, not a safety risk.
- False negatives — missed genuine conflicts — remain a residual risk.

9. The RFC Gate

Architectural decisions require human pre-clearance before the implementing branch is merged. This replaces post-hoc digest review for consequential decisions. Architecture is path-dependent: a small number of unchecked decisions can lock in patterns and make reversals expensive.

9.1 RFC Trigger Conditions

- Proposal change_type is architectural.
- Proposal would require modifying an Invariant.
- Proposal introduces a new external dependency not in the current tech stack.
- Proposal changes the interface contract of a module that three or more other modules depend on.
- Proposal resolves an ambiguity in the Invariants by choosing a path not explicitly covered.
- Shared assumption conflict detected between two or more concurrent proposals.

> *The Orchestrator will sometimes misclassify — flagging behavioral changes as architectural or missing genuine architectural choices. The system is tuned to over-flag rather than under-flag. False positives produce human review overhead; false negatives produce silent architectural drift.*

9.2 RFC Schema


```
{
  "id": "RFC_004",
  "status": "resolved",
  "created_at": "2025-06-01T14:23:00Z",
  "triggering_proposal": "prop_039",
  "decision_required": "Whether to introduce a Redis cache layer for session
management.",
  "context": "Agent-2 proposes storing session tokens in Redis rather than
PostgreSQL
    to reduce auth query load. This contradicts Invariant data_storage: PostgreSQL
only.",
  "option_a": {
    "description": "Approve as proposed. Add Redis to tech stack.",
    "consequences": "Requires updating Invariants. Adds operational dependency."
  },
  "option_b": {
    "description": "Reject. Require Agent-2 to implement session caching within
PostgreSQL.",
    "consequences": "Maintains single data store. May not fully resolve performance
concern."
  },
  "option_c": {
```

```
    "description": "Defer. Scope a performance investigation task before
deciding.",
    "consequences": "Delays feature. Produces better information for the decision."
  },
  "orchestrator_recommendation": "option_b",
  "orchestrator_rationale": "Current load does not justify adding operational
dependency.",
  "resolution": "option_b",
  "human_rationale": "Maintain single data store. Redis adds operational risk at
current scale.",
  "acceptance_criteria": [
    {
      "id": "ac_001",
      "description": "Auth query P95 latency must not exceed 150ms under standard
load test.",
      "check_type": "performance_test",
      "check_reference": "tests/load/auth_latency.py",
      "status": "passed"
    },
    {
      "id": "ac_002",
      "description": "API schema for /auth/login must conform to
openapi/auth.yaml.",
      "check_type": "schema_conformance",
      "check_reference": "openapi/auth.yaml",
      "status": "passed"
    }
  ],
  "merge_blocked_until_criteria_pass": true,
  "human_response_deadline": "2025-06-02T14:23:00Z",
  "promoted_to_adr": "ADR_003"
}
```


9.3 Acceptance Criteria

Every RFC resolution must include at least one acceptance criterion: a specific, mechanically checkable condition that must pass before the implementing branch can be merged. Criteria are written by the human operator as part of their resolution decision.

- merge_blocked_until_criteria_pass is a hard gate. The implementing branch cannot be merged until all criteria have status: passed.
- Criteria status is updated by the enforcement hook when it detects the referenced test or schema check has passed. This is deterministic — the hook checks CI artifacts, not LLM judgment.
- RFC acceptance criteria must be specific to the architectural decision at stake — not generic quality gates like 'all tests pass.'

> *Acceptance criteria written under time pressure will sometimes be incomplete. A module passing all its criteria may still have errors outside the criteria scope. This mechanism prevents violations of specifically agreed-upon invariants, not bad code in general.*

9.4 Human Response Window

Default: 24 hours. If no response is received within the window, the RFC escalates to a blocking state and all agent work touching the same semantic scope pauses. This is a deliberate forcing function. The 24-hour window is short enough to prevent silent drift and long enough to be realistic for a human operator who is not monitoring continuously.

9.5 ADR Promotion

When a human resolves an RFC, the decision is automatically promoted to an Architecture Decision Record at .skl/adrs/ADR_NNN.json. ADRs are append-only and permanent. Future Orchestrator sessions include the ADR collection in their initialization context. This prevents re-litigation of resolved architectural questions.

10. Full Workflow — Concrete Example

Feature request: Add a password reset endpoint.

Step 1 — Human Assignment

Human passes the feature request to the Orchestrator with any relevant context.

Step 2 — Orchestrator Task Assignment

- Orchestrator reads knowledge.json State and ADR collection.
- Determines: requires modifying app/routers/auth.py (scope: auth) and app/services/email.py (scope: api).
- Creates two scoped tasks with assignment_rationale recorded.
- Task A: scope_files: [app/routers/auth.py], semantic_scope: auth → Agent-1
- Task B: scope_files: [app/services/email.py], semantic_scope: api → Agent-2

Step 3 — Agents Work in Parallel

- Agent-1 works on branch feature/auth-reset-router. Realizes it needs app/utils/tokens.py, which is not in its file scope. Creates the utility anyway. Declares assumption: 'Token TTL is configurable via APP_TOKEN_TTL env variable.' Pushes branch.
- Agent-2 works on branch feature/auth-reset-email within its scope. No out-of-scope modifications. Pushes branch.

Step 4 — Enforcement Hook Runs on Both Pushes

On Agent-1's push:

  - Check 1: Detects app/utils/tokens.py is outside file scope. Auto-appends out-of-scope proposal to Queue.
  - Check 2: Validates all modified files are within auth semantic scope. app/utils/tokens.py passes (matches allowed_path pattern).
  - Check 3: AST analysis on both files. No high-risk signals. ast_change_type: behavioral on tokens.py.
  - Check 4: Import scan on tokens.py. No undeclared cross-scope imports.
  - Check 5: Queue has 2 proposals. Below budget threshold. Push allowed.

On Agent-2's push:

  - All checks pass. No out-of-scope modifications. One proposal added to Queue for the behavioral change to email.py.

Step 5 — Orchestrator Reviews Queue

Proposal 1 (Agent-2, email.py — behavioral, no conflicts):

- Stage 1: No deterministic override. ast_change_type: behavioral.
- Stage 2: Verifier agrees: behavioral. Agreement: true.
- No assumption conflicts with other proposals.
- No State conflicts. Decision: Approve. Merge branch. Update State entry for email.py to version N+1. uncertainty_level: 2. Write rationale.

Proposal 2 (Agent-1, tokens.py — out-of-scope, behavioral):

- Stage 1: No deterministic override. mechanical_only: false.
- Stage 2: Verifier agrees: behavioral. Agreement: true.
- Assumption check: Agent-1 declared 'Token TTL configurable via APP_TOKEN_TTL'. No conflicting assumptions in other proposals.
- tokens.py not in State — no conflict. Decision: Approve. Create new State entry for tokens.py. uncertainty_level: 2. Write rationale. Merge branch.

Step 6 — State Updated

- knowledge.json now contains State entries for both email.py (updated) and tokens.py (new).
- Both entries at uncertainty_level: 2 (Proposed). To reach level 1, they need human digest review. To reach level 0, they need a passing named test.
- Agent-1's declared assumption is recorded in the tokens.py State entry.

Step 7 — Human Periodic Review

- Human reviews the architectural decision digest. Sees two new State entries. Marks them reviewed. uncertainty_level drops to 1.
- CI runs test suite for auth. tests/test_auth.py passes. Orchestrator updates uncertainty_reduced_by on router_auth entry. uncertainty_level drops to 0.

11. Human Operator Responsibilities

SKL makes specific, non-optional demands on the human operator. These are not suggested practices — they are required for the system to function as designed.

Responsibility	When	Consequence of Neglect
Define Invariants accurately	Project initialization	Agents operate from a false foundation; system cannot detect this
Define scope taxonomy in scope_definitions.json	Project initialization	Enforcement hook cannot validate scope assignments
Define security pattern list in Invariants	Project initialization	auth/permission pattern detection does not fire
Review architectural decision digest	Every 10 architectural decisions (default)	SKL degrades from governance tool to audit log
Review flagged State entries (change_count_since_review at threshold)	Each digest cycle	Semantic drift in State descriptions becomes invisible
Respond to RFCs within 24-hour window	When RFC is opened	Agent work in the affected semantic scope pauses
Write RFC acceptance criteria	When resolving each RFC	RFC resolution is incomplete; merge gate has no criterion to check
Update Invariants when fundamental constraints change	When project evolves	Invariants describe a different system than what exists
Review scope definitions for drift	Same cadence as Invariant review	Scope enforcement degrades silently

12. Operational Limits

These are hard design constraints, not soft guidelines. Operating outside them degrades the system in predictable ways.

Constraint	Limit	Failure Mode if Exceeded
Concurrent agents	2–6	Queue review becomes a context management problem; scope prediction for complex features becomes unreliable
State entries	~80	Full State loading into Orchestrator context becomes unreliable; requires query layer not in this spec
Orchestrator proposals per session	15 (default)	Rationale quality degrades; formulaic approvals increase
Orchestrator session duration	90 min (default)	Same as above
Queue size	15 unresolved (default)	Hard push block triggers; agent work pauses
RFC response window	24 hours (default)	Agent work in affected scope pauses on window expiry
Simultaneous agent task completions	~3–4	Session reset may trigger mid-review; adds latency

Tasks not suitable for SKL at v1.4: deep refactors, framework migrations, performance work touching infrastructure, cross-cutting changes where correct scope is unknowable until implementation is underway. These generate high RFC and Queue volume and may require the human operator to act as Orchestrator for the duration.

13. Accepted Risks

These are known failure modes explicitly accepted rather than mitigated. They are stated here so that anyone extending the system knows what ground they are standing on.

13.1 Semantic Drift in State Descriptions

The structured schema slows description drift; it does not stop it. After enough changes, responsibilities fields become directional rather than precise. The change_count_since_review field surfaces drift; it does not prevent it. Accepted.

13.2 File-Path Scope is a Leaky Abstraction

In-scope file modifications can have broad behavioral impact not captured by path matching. The semantic scope layer and AST risk signals reduce this gap; they do not eliminate it. A change that is in-scope by both file and semantic criteria can still introduce unexpected coupling. Accepted.

13.3 Stale State Reads

Agents working simultaneously may act on State that becomes outdated mid-task. Version numbers make stale reads visible after the fact; they do not prevent them. At 2–4 agents this is an occasional nuisance. At 6 agents on interconnected tasks it is a recurring source of re-work. Accepted.

13.4 Agent Incentive Misalignment

In-scope work merges faster than out-of-scope work. Agents are implicitly rewarded for staying within scope even when scope-crossing would produce better outcomes. The human operator should periodically issue explicit refactor tasks. The system does not enforce this. Accepted.

13.5 Assumption Declaration is Partially Self-Reported

Agents that fail to declare load-bearing assumptions are not caught by deterministic enforcement. The audit trail will reveal the gap after a failure. The shared assumption conflict detection only works on assumptions that were declared. Accepted.

13.6 Human Review Fatigue

The digest and change heatmap reduce volume over time, but a long-running project will eventually see the human operator beginning to skim. SKL's value in that scenario degrades to audit trail rather than active governance. Still useful; different. Accepted.

13.7 Verifier Misclassification

Two LLMs can agree on a wrong classification. The verifier reduces the error rate by introducing independent judgment; it does not eliminate it. Deterministic Stage 1 overrides are the more reliable path for high-stakes proposals. Accepted.

13.8 Import Scanning Ceiling

Dynamic imports, configuration-driven coupling, and runtime dependencies through shared state are not detected by the import scanner. The dependency field remains partially manual. Undeclared indirect dependencies will cause stale State reads and incorrect scoping decisions. Accepted.

13.9 Acceptance Criteria Incompleteness

RFC acceptance criteria are written by humans under time pressure and will sometimes be incomplete. A module passing all its criteria may still have errors outside the criteria scope. This mechanism prevents violations of specifically agreed-upon invariants, not bad code in general. Accepted.

14. What Each Version Added

Version	Key Addition
v1.0	Single-file knowledge store, Git-based audit trail, single-Orchestrator model, escalation-and-wait pattern
v1.1	Structured State schema (replacing free text), semantic scope tags, change_count_since_review, rationale requirements for all Orchestrator decisions
v1.2	AST risk signals replacing diff_size_lines as control signal, Orchestrator session budget with structured handoff logs, RFC gate for architectural pre-clearance, ADR promotion
v1.3	Formal scope_definitions.json with machine-validated scope enforcement, static import scanning for dependency validation, two-stage classification verification with verifier pass, circuit breaker for repeated misclassification, scope_schema_version field
v1.4 (current)	uncertainty_level as first-class State invariant with routing consequences, assumption declaration with shared-assumption RFC trigger, RFC acceptance criteria as mechanically checkable contracts, propagation guarantee stated explicitly as governing principle

> *SKL guarantees that coordination-relevant uncertainty, assumptions, and cross-scope dependencies cannot propagate implicitly. It does not promise correctness. It ensures that when things go wrong, they do so locally, transparently, and with bounded blast radius.*