

Chapter 1

The Agentic Finance Framework

Chapter Deliverables

This chapter defines the book's production spine: belief formation, typed proposal, admissibility, execution, memory, and objective. It gives the concrete finance examples and software contracts that later mathematical chapters formalize.

1.1 Why Finance Technically Needs Agents

Finance does not need agents because agents are fashionable, because language models are fluent, or because every workflow should be automated. Finance needs agentic architecture for a narrower and more technical reason: many financial problems are not single predictions, single optimizations, or fully specified workflows. They are controlled sequences of decisions under partial observation, heterogeneous evidence, changing constraints, side effects, reflexive feedback, and audit obligations.

A standard predictive model answers a fixed-input question,

$$\hat{y} = f_{\theta}(x).$$

A classical optimizer answers a fixed-feasible-set question,

$$a^* \in \arg \min_{a \in \mathcal{K}(x)} C(a; x).$$

A deterministic workflow engine answers a rule-complete question: if the state is known and all branches are specified, follow the prescribed transition. These objects remain indispensable. Pricing libraries, risk engines, limit systems, portfolio optimizers, reconciliation rules, settlement ledgers, and approval tables should not be replaced by agents.

The agentic problem begins when the system must choose not only an answer, but a path of inquiry and control. The financial question becomes

Given a replayable history H_t , incomplete evidence, and live constraints, which sources should be retrieved, which tools should be called, which proposal should be formed, which validator should bind, what action is admissible, and what evidence must be stored?

The output is not merely a label. It may be a typed proposal, a tool call, a portfolio adjustment, a child-order instruction, a compliance escalation, a client-response package, a collateral workflow, a rejected action, or an audit record. The technical object is therefore not the language model alone. It is the controlled system that turns uncertain evidence into a constrained and replayable state transition.

The first principle of this chapter is conservative. An agent is justified only when a financial workflow requires a controlled loop. If the task is a static formula, use the formula. If it is a clean optimization problem, use the optimizer. If it is a stable classification problem, use the classifier. If it is a rule-complete case, use the rule engine. But when the system must gather evidence, choose tools, interpret intent, call risk engines, check constraints, propose an intervention, mutate state, monitor consequences, and preserve an audit packet, the architecture has become agentic whether or not the institution uses the word “agent.”

1.1.1 *Partial Observation*

Financial decisions rarely begin from a complete state vector. A desk receives a client email with an ambiguous instruction. A portfolio manager reads a research note whose assumptions conflict with the risk system. A collateral team sees a margin dispute with stale prices, missing legal terms, and inconsistent time zones. A compliance officer investigates a communication whose meaning depends on prior conversations, restricted lists, and policy versions. In each case, the system must form a belief state

$$b_t = \rho_t(H_t),$$

where H_t is a history of observations, documents, tool outputs, approvals, rejections, and state updates. The belief is not the prompt. It is the controlled representation of what the institution currently knows, what it does not know, and which sources support each claim.

1.1.2 Endogenous Information Acquisition

A non-agentic model consumes the information it is given. A financial agent often has to decide which information should be acquired next. It may need to retrieve a mandate before proposing a trade, query a limit service before drafting an execution instruction, request missing tax status before recommending a product, or rerun valuation before resolving a collateral dispute. Formally, the system chooses an information action q_t before choosing the financial action a_t :

$$q_t \in \mathcal{Q}_t(H_t), \quad o_{t+1}^q \sim \mathcal{O}_t(\cdot \mid H_t, q_t), \quad a_t \in \mathcal{A}_t(H_t, o_{t+1}^q).$$

Once the next admissible financial action depends on information that the system itself must choose to acquire, the workflow is no longer a static prediction problem.

1.1.3 Sequential Tool Use

Many financial workflows cannot choose their final action in one step. The system must first parse the request, retrieve contracts or policies, normalize market data, check units, compute exposures, compare limits, ask for missing information, run a pricing or risk tool, propose a candidate action, validate the post-action state, and either execute, reject, or escalate. The order matters. A system that must decide the next computation before it knows the final action is already solving a control problem. The question is not only “what is the answer?” It is “what should the system do next, given what it has learned so far?”

1.1.4 Side Effects

Financial outputs can change the institution. A routed order changes inventory, market exposure, and execution risk. A limit override consumes scarce risk capacity. A client communication creates conduct, suitability, and recordkeeping consequences. A collateral instruction changes funding and settlement state. A case escalation alters the audit trail and human accountability chain. Once an output has side effects, the architecture must distinguish a raw proposal $\tilde{a}_t$ from an admissible action a_t :

$$\tilde{a}_t \xrightarrow{\Pi_t} a_t.$$

The language model may help produce $\tilde{a}_t$, but the institution must own the projection, shield, rejection, or escalation rule Π_t .

1.1.5 External Constraints

Mandates, suitability rules, concentration limits, restricted lists, liquidity constraints, model-risk approvals, market-abuse policies, capital budgets, conduct rules, and escalation thresholds are not stable facts inside a prompt. They are external, versioned, institutionally owned constraints. Therefore the core financial question is not whether a model can explain a plausible action. The core question is whether the proposed action remains admissible after applying the current state, current mandate, current policy version, current client status, current risk budget, and current approval authority.

1.1.6 Audit Replay

A regulated financial institution must often reconstruct what was known, which source was used, what the model proposed, which validator fired, who approved, what was executed, and how the state changed. Audit replay is not a logging convenience. It is part of the control system. A system that cannot reconstruct its decision as evidence objects is not suitable for delegated financial authority, even if its model outputs are usually correct.

1.1.7 Reflexivity

Financial agents are not passive predictors. Their actions can change the environment they are meant to optimize against. Orders move liquidity. Public recommendations affect beliefs. Similar models can create crowding. Risk reductions by many institutions can amplify stress. Margin calls can force sales, and forced sales can create more margin calls. This is why a financial agent must be understood as a controller embedded in a strategic environment. Its autonomy has to shrink when feedback becomes expansive.

These features are individually familiar. Their combination is what creates the technical need for agents.

Design Principle 1.1 (Technical necessity test for financial agents). *A financial workflow justifies an agentic architecture only when it requires a closed-loop composition of belief formation, evidence acquisition, tool use, proposal generation, constraint checking, state update, monitoring, and audit replay. If the task can be fully specified as a deterministic parser, database query, pricing formula, convex program, reconciliation rule, or approval table, the non-agentic substrate should dominate.*

The purpose of an agentic framework is not to make finance more theatrical. It is to give technical names, control boundaries, and evidence requirements to the parts of financial work that are already sequential, uncertain, constrained, and state-changing.

1.2 Formal Model: A Financial Agent as a Controlled Evidence System

Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \geq 0}, \mathbb{P})$ be a filtered probability space. The authoritative institutional state is

$$X_t \in \mathcal{X},$$

where X_t may include portfolios, orders, inventory, limits, client status, workflow cases, collateral records, approvals, market state, and system permissions. The agent does not observe X_t completely. It observes a history,

$$H_t = (o_0, u_0, d_0, o_1, u_1, d_1, \dots, o_t),$$

where o_i denotes observations, u_i denotes tool calls or information actions, and d_i denotes decisions, validator outputs, approvals, or rejections. The history H_i must be replayable: it is not a vague conversation transcript, but a versioned sequence of evidence objects and state transitions.

The agentic finance system is the six-part controlled architecture

$$\mathfrak{A} = (\rho, \pi, \Pi, \mathcal{E}, \mathcal{M}, \mathcal{J}).$$

Operationally,

$$H_i \xrightarrow{\rho_i} b_i \xrightarrow{\pi_i} \tilde{a}_i \xrightarrow{\Pi_i} a_i \xrightarrow{\mathcal{E}_i} X_{i+1} \xrightarrow{\mathcal{M}_i} H_{i+1}, \quad \mathcal{J} \text{ evaluates outcomes and constraints.}$$

Here b_i is the belief state, $\tilde{a}_i$ is the raw proposal, a_i is the admissible action after projection or shielding, $\mathcal{E}_i$ is the execution interface, $\mathcal{M}_i$ is the memory and audit update, and $\mathcal{J}$ is the monitored objective. The language model appears only in selected parts of ρ_i and π_i . It does not own admissibility, execution authority, memory, or institutional purpose.

A technically useful belief state is not merely a text summary. It is a structured object of the form

$$b_i = (\hat{x}_i, \mathcal{U}_i, G_i, \mathcal{C}_i, m_i, \nu_i),$$

where $\hat{x}_i$ is the normalized state estimate, $\mathcal{U}_i$ is an ambiguity set or uncertainty representation, G_i is the evidence graph, $\mathcal{C}_i$ is the active constraint set, m_i is the set of missing or contradictory fields, and ν_i contains versions of policies, models, schemas, and data sources. In a probabilistic implementation, $\mathcal{U}_i$ may be a posterior distribution over X_i . In a robust implementation, it may be a set of plausible distributions, stress regimes, or scenario measures.

The resulting control problem can be written abstractly as

$$\begin{aligned} \max_{\rho, \pi, \Pi} \quad & \mathbb{E} \left[\sum_{t=0}^{T-1} \beta^t r_t(X_t, a_t) - \sum_{t=0}^{T-1} \beta^t c_t^{\text{op}}(H_t, \tilde{a}_t, a_t) - \sum_{t=0}^{T-1} \beta^t c_t^{\text{audit}}(H_t, a_t) \right] \\ \text{s.t.} \quad & b_t = \rho_t(H_t), \\ & \tilde{a}_t = \pi_t(b_t), \\ & a_t = \Pi_t(\tilde{a}_t; b_t), \\ & a_t \in \mathcal{K}_t(b_t) \quad \text{or} \quad a_t \in \{\text{reject}, \text{escalate}\}, \\ & X_{t+1} \sim \mathcal{E}_t(\cdot \mid X_t, a_t), \\ & H_{t+1} = \mathcal{M}_t(H_t, b_t, \tilde{a}_t, a_t, X_{t+1}). \end{aligned}$$

This display is not meant to imply that every production system solves a clean Bellman equation. Its purpose is to identify the object under control. The agent is not the text generator. The agent is the controlled evidence-action-memory loop.

Table 1.1 Formal components of the agentic finance framework and their production interpretations.

Component	Question Answered	Production Interpretation
ρ_t	What do we believe?	Retrieval, normalization, provenance, uncertainty, missing-field detection, source ranking, contradiction detection.
π_t	What could be done?	LLM proposal, optimizer candidate, rule suggestion, analyst draft, or human instruction converted into a typed payload.
Π_t	What is allowed?	Schema validation, permissions, risk limits, suitability, restricted lists, liquidity checks, capital budget, escalation logic.
$\mathcal{E}_t$	What changes the world?	OMS/EMS adapter, case-management action, client-message queue, collateral system, limit-service update.
$\mathcal{M}_t$	What is remembered?	Immutable logs, source hashes, tool outputs, validator decisions, approvals, post-action state, incident records.
$\mathcal{J}$	What is success?	P&L, risk-adjusted utility, service quality, compliance, operational loss, auditability, human escalation quality.

The framework in Table 1.1 separates four objects that are often confused in informal agent discussions:

1. A **model** maps inputs to outputs. It may be a language model, classifier, forecaster, optimizer, or rules assistant.
2. A **tool** exposes a controlled operation: price an instrument, compute risk, retrieve policy, check a client status, create a draft order, or open a case.
3. A **workflow** composes tools, validators, approvals, and state transitions.
4. An **agent** chooses and coordinates the next step of a workflow under uncertainty, while remaining inside a typed and auditable control envelope.

This distinction is load-bearing. A pricing model can be excellent and still be unsafe as an agent. A language model can be useful and still have no execution authority. A rule engine can be deterministic and still be incomplete. The framework asks where each component belongs, which assumptions it owns, what it is allowed to mutate, and which evidence it must leave behind.

1.3 Admissibility as a Set-valued Control Problem

The most important object in agentic finance is not the raw model output. It is the admissible action set. Given a belief state b_t , define

$$\begin{aligned} \mathcal{K}_t(b_t) = & \mathcal{K}_t^{\text{schema}}(b_t) \cap \mathcal{K}_t^{\text{permission}}(b_t) \cap \mathcal{K}_t^{\text{mandate}}(b_t) \cap \mathcal{K}_t^{\text{suitability}}(b_t) \\ & \cap \mathcal{K}_t^{\text{risk}}(b_t) \cap \mathcal{K}_t^{\text{liquidity}}(b_t) \cap \mathcal{K}_t^{\text{capital}}(b_t) \cap \mathcal{K}_t^{\text{conduct}}(b_t) \cap \mathcal{K}_t^{\text{operations}}(b_t). \end{aligned}$$

An action is institutionally admissible only if it survives all relevant layers. The shield is therefore a set-valued projection with an explicit fail-closed state:

$$\Pi_t(\tilde{a}_t; b_t) = \begin{cases} \tilde{a}_t, & \tilde{a}_t \in \mathcal{K}_t(b_t), \\ \arg \min_{a \in \mathcal{K}_t(b_t)} D_t(a, \tilde{a}_t; b_t), & \mathcal{K}_t(b_t) \neq \emptyset \text{ and rewriting is authorized,} \\ \text{escalate,} & \text{human judgment or missing authority is required,} \\ \text{reject,} & \text{no admissible or authorized action exists.} \end{cases}$$

The four branches in the definition of Π_t should be read geometrically. The raw proposal $\tilde{a}_t$ is first placed in the full action space $\mathcal{A}$. It becomes executable only if it lies inside the intersection of all active admissibility layers. These layers include schema validity, permission, mandate, suitability, risk, liquidity, capital, conduct, and operational constraints. The intersection is the admissible set $\mathcal{K}_t(b_t)$.

Figure 1.1 gives the operational reading of this set-valued shield. If $\tilde{a}_t \in \mathcal{K}_t(b_t)$, the proposal may be allowed. If $\tilde{a}_t \notin \mathcal{K}_t(b_t)$ but rewriting is authorized, the shield may project the proposal toward an admissible action using the distance D_t . If human judgment, missing authority, or unresolved evidence is required, the shield escalates. If no admissible or authorized action exists, the shield rejects. The point is that admissibility is not a confidence score produced by the model; it is membership in a versioned, externally owned feasible set.

The distance D_t need not be Euclidean. For trading, it may penalize notional change, risk-budget consumption, turnover, impact, and tracking-error deviation. For client communication, it may penalize semantic drift from the original message and missing mandatory disclosures. For collateral workflows, it may penalize deviations from legal terms and valuation timestamps. The projection is therefore not merely mathematical; it is institutional.

Hard constraints and soft preferences must be separated. A hard constraint is a predicate $g_i(a, b_t) \leq 0$ that cannot be relaxed by model confidence. A soft preference is a term in the objective or ranking function. A common failure mode is to let a fluent explanation convert a hard rule into a soft preference. The framework forbids this. If $a \notin \mathcal{K}_t^{\text{hard}}(b_t)$, then no rationale can make a executable.

Design Principle 1.2 (Separation of proposal and authority). *The language model may propose $\tilde{a}_t$. Authority begins only after a versioned deterministic layer maps the proposal into an admissible action, rejection, or escalation. A natural-language rationale is never a substitute for membership in $\mathcal{K}_t(b_t)$.*

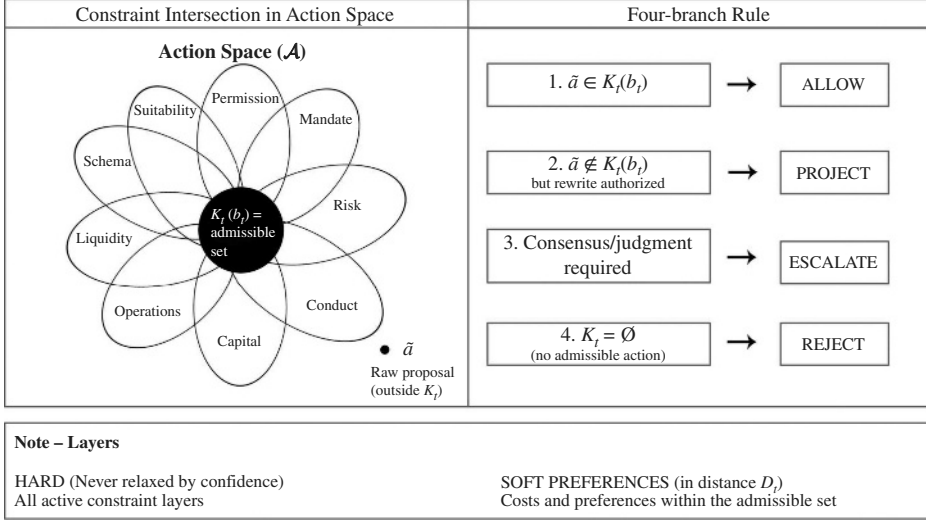


Figure 1.1 The set-valued admissibility shield. The left panel represents the admissible set $K_t(b_t)$ as the intersection of active constraint layers inside the full action space $\mathcal{A}$. A raw proposal $\tilde{a}_t$ outside this set cannot be executed merely because it is plausible or confidently explained. The right panel gives the four-branch rule: allow if the proposal is admissible; project when rewriting is authorized and an admissible alternative exists; escalate when judgment, consensus, or missing authority is required; and reject when no admissible or authorized action exists. Hard layers are never relaxed by model confidence; soft preferences enter only through the distance D_t or ranking objective.

1.4 Distributional Shielding

In finance, the most dangerous actions are often syntactically valid. A trade can have the correct symbol, side, notional, account, and approval field while still creating unacceptable tail exposure. Therefore admissibility must be evaluated on the post-action state, not on the proposal in isolation.

Let w_t denote the current portfolio, a_t the proposed change, and $w_t^+(a_t)$ the post-action portfolio after costs, fills, and financing conventions. Let

$$L_{t+1}(a_t) = -\Delta V_{t+1}(w_t^+(a_t))$$

be the one-period loss of the resulting position. A distributional shield checks risk functionals of $L_{t+1}(a_t)$ under the current belief state. A typical policy is

$$\text{VaR}_{\alpha,t}(L_{t+1}(a_t) \mid b_t) \leq B_t^{\text{VaR}},$$

$$\text{CVaR}_{\alpha,t}(L_{t+1}(a_t) \mid b_t) \leq B_t^{\text{CVaR}},$$

$$\mathbb{P}_t(L_{t+1}(a_t) > \ell_t^{\max} \mid b_t) \leq \eta_t,$$

$$\text{Conc}_t(w_t^+(a_t)) \leq B_t^{\text{conc}},$$

$$\text{LiqCost}_t(a_t) \leq B_t^{\text{liq}}.$$

When model uncertainty is material, the shield should be robustified. Let $\mathfrak{U}_t(b_t)$ be an ambiguity set of plausible conditional laws for $L_{t+1}(a_t)$. A robust distributional gate is

$$\sup_{Q \in \mathfrak{U}_t(b_t)} \text{CVaR}_\alpha^Q(L_{t+1}(a_t)) \leq B_t^{\text{CVaR}}.$$

The ambiguity set may be built from stress regimes, volatility scalings, Wasserstein balls, empirical block bootstraps, extreme-value tails, liquidity haircuts, or model-risk overlays. The important point is that the gate is about the distribution of the resulting state, not about the eloquence of the model's reasoning.

This also clarifies why bounded-difference arguments are fragile in financial agent design. If a proposed autonomy bound assumes a deterministic loss increment $L_t \in [0, b]$, the bound is only as credible as the mechanism that enforces the cap b . In liquid markets, operational losses, gap risk, slippage, wrong-way liquidity, and stress correlation are often heavy-tailed. Unless the platform has hard liquidation, notional, kill-switch, and loss-stop mechanisms that make the bound true by construction, the correct shield should use heavy-tail-aware or robust risk functionals rather than relying on a compact support assumption.

Example 1.3 (Post-action rejection under a distributional shield). *A portfolio currently holds an 8.1% weight in NVDA and the mandate caps any single-name exposure at 10%. The LLM proposes a USD 75 million buy order with an apparently reasonable narrative. The deterministic core computes the post-action state before execution:*

$$w_{\text{NVDA}}^{\text{post}} = 14.2\%, \quad \text{VaR}_{0.95}^{\text{post}} = 3.4\%, \quad \text{CVaR}_{0.95}^{\text{post}} = 5.9\%.$$

The mandate permits at most 10% single-name exposure and the risk budget caps one-day $\text{VaR}_{0.95}$ at 2.5%. The shield therefore returns `reject`. A negotiator may compute a smaller counteroffer, but the counteroffer must be re-evaluated by the same shield before it can reach the execution adapter. The audit packet records the original proposal, portfolio state, covariance snapshot, mandate version, risk metrics, shield verdict, counteroffer if any, and the committed action. The explanation is useful; the replayable packet is the evidence.

Figure 1.2 summarizes the distributional reading of the shield. The relevant object is not the narrative plausibility of the proposed trade, but the conditional law of the post-action loss $L_{t+1}(a_t)$. A proposal may be syntactically valid and still be inadmissible because its post-action distribution violates a tail-probability budget, a VaR budget, a CVaR budget, a concentration cap, or a liquidity constraint. The figure also makes explicit why bounded-loss assumptions are fragile in finance: unless the platform enforces the bound by construction, the real loss distribution may have a heavy tail that pushes tail mean beyond the budget even when a point estimate or a nominal VaR appears acceptable.

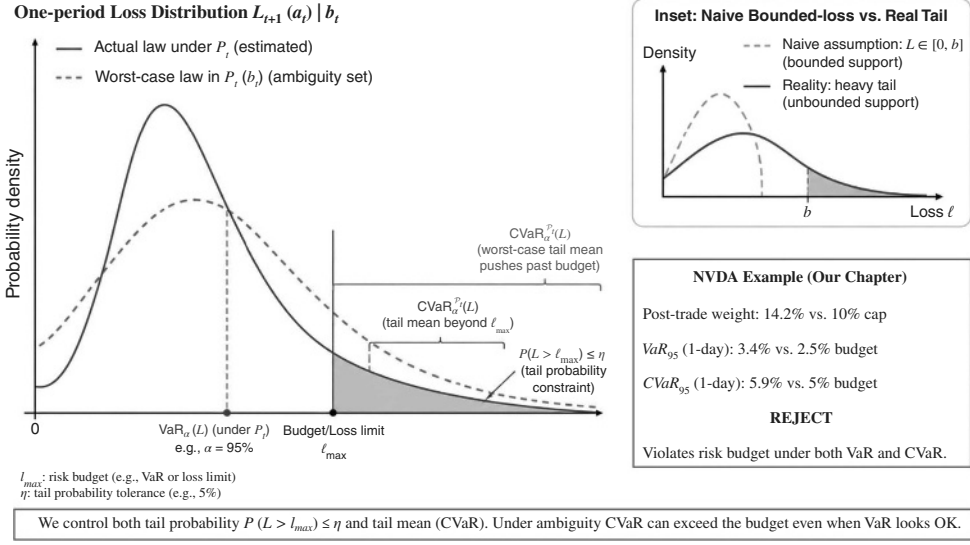


Figure 1.2 Distributional shielding for post-action loss. The shield evaluates the conditional law of $L_{t+1}(a_t)$, the one-period loss of the resulting position, rather than the raw proposal in isolation. The solid curve represents the estimated law under the current model, while the dashed curve represents a worst-case law inside the ambiguity set. The shield controls both tail probability, $P(L > \ell_{max})$, and tail severity through $CVaR_\alpha(L)$. The inset illustrates why a naive bounded-loss assumption is unsafe when real financial losses have heavy tails. The NVDA example shows a post-action rejection: the proposed trade breaches the single-name cap and violates both the VaR and CVaR risk budgets.

1.5 Belief Formation as Evidence Engineering

The belief operator ρ_t is often the weakest part of financial agent design. Many systems treat the prompt as context and the context as truth. This is not acceptable for financial authority. A production belief state should distinguish facts, estimates, missing fields, contradictions, and unsupported claims.

Let the evidence graph be

$$G_t = (V_t, E_t),$$

where each node $v \in V_t$ is a source object with attributes

$$\text{attr}(v) = (\text{source id, type, owner, timestamp, version, permission, hash, retention policy}).$$

Edges represent extraction, transformation, reconciliation, contradiction, or approval relationships. A normalized fact z is admissible for downstream use only if it carries provenance,

$$z = (\text{field, value, unit, as-of time, source path, confidence}).$$

Thus, “client is professional” is not a belief unless the system can identify the client-classification record, version, jurisdiction, timestamp, and permission under which that classification is valid.

A robust belief state should include at least four diagnostics:

1. **Provenance coverage:** which claims have source support and which are inferred or missing
2. **Temporal consistency:** whether prices, positions, approvals, policies, and client records share compatible as-of times
3. **Authority hierarchy:** which system is authoritative when retrieved sources conflict
4. **Action relevance:** which missing fields can change the admissible action set $\mathcal{K}_i(b_i)$

A missing field is not equally important in all contexts. Missing color in a chart is harmless. Missing tax residence, mandate version, settlement date, or restricted-list status can change the feasible set and must therefore trigger escalation or rejection.

1.6 Engineering Bridge: From Theorem to System Design

This chapter is written for two technical audiences that often describe the same problem with different vocabularies. A financial engineering reader sees a controlled stochastic system: state, observations, beliefs, admissible actions, transition dynamics, costs, constraints, and a value functional. A computer science reader sees a production system: state stores, retrieval services, schemas, validators, tool adapters, queues, permissions, logs, regression tests, and incident runbooks. These are not two separate views. They are the same architecture viewed at different levels of abstraction. Table 1.2 translates the formal state-action system into its financial and production implementations.

The bridge performs three conversions:

1. **From notation to responsibility.** Each mathematical symbol is assigned to a system component with an owner, test suite, and audit record.
2. **From natural language to typed payloads.** The LLM’s output is treated as a proposal, not as an executable act. Authority begins only after the proposal satisfies a declared schema and the admissibility layer accepts it.
3. **From model behavior to institutional evidence.** A fluent explanation is not evidence. The evidence is the replayable packet: input state, retrieved sources, prompt or policy version, model output, validator decisions, shield metrics, approval state, execution result, and post-action record.

A theorem tells us what follows under assumptions. A design principle tells us what to build if we accept those assumptions. A protocol rule tells engineers what must be enforced at runtime. The mistake in many agent deployments is to jump directly from theorem to prompt. The correct path is the following extractable control chain:

theory → typed contract → validator → shield → execution adapter → audit packet

Table 1.2 Mapping mathematical objects to financial-engineering and system-engineering interpretations.

Mathematical Object	Financial-engineering Reading	System-engineering Reading
State X_t	Portfolio, inventory, client status, limits, market regime, collateral, workflow state	Authoritative databases, ledgers, case records, position stores, limit services
Observation o_t	Noisy and partial evidence: prices, filings, messages, news, risk reports	Retrieval outputs, market-data feeds, document parsers, event streams
Belief b_t	Filtered estimate, uncertainty set, active constraints, missing variables	Normalized context object with provenance, timestamps, confidence, and missing fields
Proposal $\tilde{a}_t$	Candidate trade, allocation, message, route, override, or escalation	LLM/tool output satisfying a declared action schema
Admissible action a_t	Feasible action under mandate, risk, liquidity, law, and capital constraints	Payload accepted by validators, shields, permissions, and approval workflows
Transition $\mathcal{E}_t$	Market impact, ledger mutation, client response, inventory update, operational consequence	API side effect, queue event, database write, order submission, case update
Objective $\mathcal{J}$	Utility, P&L, risk, cost, service quality, compliance, auditability	Metrics, dashboards, tests, alerts, audit scores, incident thresholds

In words, formal assumptions must be converted into a typed contract, checked by validators and shields, executed through a narrow adapter, and preserved as an audit packet.

1.7 Runtime Sequence

Every consequential agentic workflow in finance should be decomposed into the following operating sequence.

- 1. Observe.** The system collects market data, portfolio state, client state, policy documents, messages, or external evidence. Inputs are timestamped, permissioned, and source-linked.
- 2. Form a belief.** The retrieval and normalization layer builds a belief state: facts, uncertainty, provenance, active constraints, and open questions. This is ρ_t . The belief state is not whatever text happens to be inside the prompt; it is the controlled state representation used by the platform.
- 3. Propose.** The LLM, policy model, optimizer, rules assistant, or human-in-the-loop process produces a structured proposal. This is $\pi_t(b_t)$. The proposal may include rationale, evidence references, confidence, and requested action, but it has no side effect by itself.

4. **Validate and shield.** The deterministic core checks schema, permissions, restrictions, limits, concentrations, liquidity, suitability, idempotency, and cumulative budgets. This is Π_i . The shield may allow, reject, escalate, delay, or rewrite into a safer candidate that must be checked again.
5. **Execute narrowly.** Only an accepted payload reaches $\mathcal{E}_i$, the execution interface. This interface exposes the smallest sufficient action surface: for example, a child-order adapter, a case-routing action, a report-submission endpoint, or a communication draft queue.
6. **Record and monitor.** The system writes the committed trace to $\mathcal{M}_i$ and evaluates outcomes under $\mathcal{J}$. The log is not an afterthought. It is the evidence that the institution used the model inside a controlled delegation chain.

1.8 The Minimal Production Contract

The following contract is intentionally plain. It expresses the central production rule: the model does not mutate the world by emitting prose. It emits a typed proposal. The deterministic system checks the proposal. Only a checked payload reaches an execution adapter. The audit layer records the evidence, the proposal, the validator result, the shield decision, and the state transition.

```
from enum import Enum
from pydantic import BaseModel, Field
from typing import Any, Literal

class ActionType(Enum):
    EXTRACT = "extract"
    RECOMMEND = "recommend"
    EXECUTE_TRADE = "execute_trade"
    ROUTE_ORDER = "route_order"
    OPEN_CASE = "open_case"
    ESCALATE = "escalate"
    REJECT = "reject"

class EvidenceRef(BaseModel):
    source_id: str
    source_type: Literal[
        "filing", "policy", "market_data", "audit_log",
        "client_record", "risk_report", "legal_agreement"
    ]
    timestamp_utc: str
    version: str
    hash: str
    permission_scope: str

class BeliefState(BaseModel):
    case_id: str
    as_of_utc: str
```

```

    retrieved_evidence: list[EvidenceRef]
    normalized_facts: dict[str, Any]
    missing_fields: list[str]
    contradictions: list[str]
    uncertainty: dict[str, float]
    active_policy_versions: dict[str, str]

class ProposedAction(BaseModel):
    action_type: ActionType
    payload: dict[str, Any]
    intended_side_effect: str
    rationale: str
    evidence: list[EvidenceRef]
    confidence: float = Field(ge=0.0, le=1.0)
    idempotency_key: str

class RiskMetrics(BaseModel):
    var_95: float | None = None
    cvar_95: float | None = None
    tail_probability: float | None = None
    concentration_after: dict[str, float] = {}
    liquidity_cost_estimate: float | None = None
    stress_loss: dict[str, float] = {}

class ShieldDecision(BaseModel):
    status: Literal["allow", "reject", "escalate", "rewrite"]
    reason: str
    checked_rules: list[str]
    validator_versions: dict[str, str]
    risk_metrics: RiskMetrics | None = None
    rewritten_payload: dict[str, Any] | None = None

class ExecutionResult(BaseModel):
    executed: bool
    execution_adapter: str
    external_reference: str | None = None
    post_state_hash: str | None = None
    error: str | None = None

class AgentContract(BaseModel):
    belief: BeliefState #  $\rho_t(H_t)$ 
    proposal: ProposedAction #  $\pi_t(b_t)$ 
    shield_decision: ShieldDecision #  $\Pi_t(\text{proposal}; \text{belief})$ 
    execution_result: ExecutionResult
    memory_update: dict[str, Any] #  $M_t$ 
    objective_trace: dict[str, Any] #  $J$ 

```

The contract separates five validities that are often collapsed in weak agent systems:

1. **Syntactic validity:** the payload conforms to the declared schema.
2. **Semantic validity:** the fields mean what they claim to mean and are grounded in evidence.
3. **Financial validity:** the post-action state satisfies risk, liquidity, capital, and portfolio constraints.

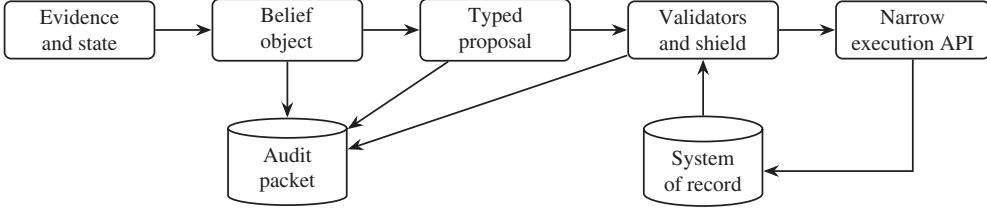


Figure 1.3 From theorem to production system. Natural language may enter at the boundary, but authority passes through typed payloads, deterministic controls, distributional shields, narrow execution adapters, and replayable logs.

4. **Authority validity:** the actor, workflow, and approval chain are allowed to perform the side effect.
5. **Replay validity:** the institution can reconstruct the state, evidence, proposal, validators, decision, and execution result.

Figure 1.3 summarizes the path from theorem to typed contract, validator, shield, execution adapter, and audit packet.

1.9 Audit Invariants

Auditability is a technical invariant, not a documentation exercise. For every action that is allowed, rejected, rewritten, or escalated, the system should write an audit tuple

$$\ell_t = \left(\text{id}_t, h(H_t), h(b_t), h(\tilde{a}_t), \nu_t^{\text{model}}, \nu_t^{\text{policy}}, \nu_t^{\text{validator}}, d_t, h(a_t), \gamma_t \right),$$

where $h(\cdot)$ denotes a hash or content address, d_t is the shield decision, and γ_t is the execution or non-execution result. Four invariants matter most:

1. **Causal ordering:** the proposal, validator output, approval, and execution result must be ordered and attributable.
2. **Version completeness:** policies, schemas, prompts, models, risk engines, and data sources must be versioned.
3. **Idempotency:** repeated calls with the same idempotency key must not create unintended repeated side effects.
4. **Replayability:** an independent reviewer must be able to reconstruct why the system allowed, rejected, rewrote, or escalated the proposal.

A system without these invariants may still be useful as an assistant, but it is not suitable for controlled financial delegation.

1.10 Multi-agent Committees and Bounded Negotiation

Many financial deployments will use several specialist agents: research, technical analysis, risk, compliance, portfolio management, operations, or legal. The correct interpretation

is not that these agents vote their way around the control system. They are advisory processes inside a larger admissibility architecture.

Let $i = 1, \dots, m$ index specialist agents, each producing a diagnostic or proposal

$$z_i^j = \pi_i^j(b_i).$$

An aggregation rule $\mathcal{C}_i$ may summarize these diagnostics into a committee proposal

$$\tilde{a}_i^{\text{com}} = \mathcal{C}_i(z_i^1, \dots, z_i^m; b_i).$$

The final action is still

$$a_i = \Pi_i(\tilde{a}_i^{\text{com}}; b_i).$$

No committee vote can bypass Π_i . No rhetorical consensus can relax a hard constraint. No specialist role can grant itself execution authority.

A negotiator shield can be useful when the raw proposal is too risky but a smaller or more conditional action may be admissible. For example, a USD 75 million order may fail concentration and liquidity constraints, while a USD 18 million staged VWAP order with a participation cap may pass. Negotiation is safe only if it is monotone with respect to hard risk controls. A simple design rule is

$$R_i(a^{k+1}; b_i) \leq R_i(a^k; b_i) \quad \text{for every hard risk functional } R_i$$

whenever the negotiator claims to reduce risk. The final counteroffer must still be passed through the same shield. Negotiation may shrink, stage, collateralize, or condition a proposal; it may not relax the mandate, ignore a restricted list, or consume an unauthorized risk budget.

1.11 Real-life Financial Examples as Controlled Traces

The following examples are stylized, but they are written as production workflows rather than toy prompts. In each case, the value of the agent is not that it “sounds smart.” The value is that it coordinates messy evidence, multiple tools, explicit constraints, and a replayable decision path.

1.11.1 *Wealth-management Instruction with Missing Suitability Context*

A relationship manager receives the message: “Please move part of my cash into short-duration bond funds today; I do not want equity risk before the central bank announcement.” A chatbot can draft a response, but it cannot safely decide what to do. The controlled agent must retrieve the client’s mandate, risk profile, restricted-product list, tax residence, cash balance, settlement constraints, previous instructions, and the current approved-product universe. It may then call a suitability tool, a duration-exposure calculator, a concentration checker, and a fee-disclosure generator. The raw proposal could be a set of candidate funds and a client message. The shield must reject any candidate outside the mandate, above concentration limits, or inconsistent with the client’s

documented risk profile. The admissible output may be a draft recommendation, an escalation to an adviser, or a rejection that asks for clarification. The audit packet must show the message, retrieved client facts, tool outputs, suitability checks, rejected alternatives, final draft, and human approval state.

1.11.2 Execution Desk Child-order Recommendation

A trader receives a parent order to buy a large block of a liquid equity before a client deadline. A static model can estimate expected volume or short-term price impact. An optimizer can produce an execution schedule if all parameters are known. The real workflow is more complicated. The system must observe the live order book, volume curve, volatility, news flags, venue status, participation limits, client urgency, restricted times, and current fill state. It must decide whether to place passive child orders, cross the spread, pause, reduce size, or escalate. The agent's useful role is to propose the next execution instruction in a typed format: symbol, side, maximum quantity, limit-price logic, participation cap, time-in-force, venue restrictions, and rationale. The shield then evaluates post-action risk: inventory, expected slippage, market-impact budget, liquidity stress, and operational constraints. If the proposal violates a participation cap or attempts to trade during a restricted window, the shield rejects it regardless of the explanation.

1.11.3 Collateral Dispute and Margin Workflow

A counterparty disputes a margin call, claiming that one curve input is stale and one trade was already terminated. Resolving the case requires retrieving the CSA, trade population, valuation timestamp, market-data source, dispute threshold, eligible collateral schedule, termination record, and prior communications. A single classifier cannot solve this problem because the correct next step may be to recalculate exposure, open an operations case, ask for missing evidence, escalate to legal, or suspend the workflow until a data issue is resolved. An agentic workflow can build the evidence packet, call valuation and data-quality tools, compare the disputed inputs with approved sources, and propose one of several controlled actions: accept the dispute, reject it, request documents, rerun valuation, or escalate. The deterministic core checks whether the user has authority to alter the margin call, whether the threshold is exceeded, and whether the counterparty agreement allows the proposed action.

1.11.4 Credit Portfolio Monitoring After Adverse News

A negative news event affects a borrower in a credit portfolio. A forecast model may estimate default probability, and a database query may list exposures. But a risk manager needs a controlled workflow: retrieve the exposure, covenants, collateral, guarantees, credit memo, latest financials, sector comparables, trading levels, watch-list status, and previous committee decisions. The system must decide whether to refresh the rating, open a watch-list case, recommend a hedge, restrict new exposure, or escalate to committee. The admissibility layer prevents the agent from changing the official rating

or hedging the exposure without the required approval. Thus the agent improves speed and completeness of investigation, while the deterministic core preserves formal credit authority.

1.11.5 Compliance Review of a Proposed Client Communication

A banker drafts a message to a client explaining a structured note, including potential upside, downside, early redemption, issuer risk, liquidity, and fees. The communication must be consistent with approved terms, fair presentation rules, local marketing policy, client classification, product restrictions, and required disclosures. A language model can help rewrite the message, but it cannot be the final compliance gate. The agent retrieves the product term sheet, policy version, client classification, approved disclosure library, and previous edits. It proposes a revised message and identifies unsupported claims. The validator checks mandatory disclosures, prohibited phrases, missing risk factors, and product-eligibility constraints. The admissible output may be an approved draft, a red-lined draft, or a compliance case.

1.11.6 Treasury Hedge Exception

A corporate treasury desk receives an urgent request to hedge foreign-exchange exposure before a deadline. The request mentions an expected invoice, but the supporting document is missing, and the notional exceeds a soft limit. A simple rule engine can block the trade, but it cannot assemble the evidence, ask the right question, calculate the proposed hedge ratio, compare policy exceptions, and prepare an approval packet. A language model alone can draft a persuasive explanation, but it should not approve the exception. An agentic system retrieves the hedging policy, invoice history, exposure report, current limits, approved counterparties, and market quotes. It calls tools to estimate exposure, hedge effectiveness, liquidity cost, and limit usage. It then proposes either a smaller hedge, a human approval request, or a rejection with missing evidence. The deterministic shield prevents execution until the exception workflow is complete. Table 1.3 distinguishes workflows that benefit from bounded agentic coordination from tasks that belong on deterministic substrates.

These examples also show the negative definition. Agents are not needed to price a vanilla swap once the curve and model are fixed. They are not needed to compute portfolio variance from a covariance matrix. They are not needed to run a well-specified reconciliation rule. They are needed when a financial system must navigate from incomplete and heterogeneous evidence to a constrained, state-changing, auditable action.

1.12 What Must Never Be Delegated to the Language Model

Because the examples are realistic, it is worth stating the boundary early. The language model may summarize documents, extract candidate fields, draft messages, compare policies, explain rejected actions, and propose typed payloads. It should not own the following functions:

Table 1.3 Financial workflows requiring controlled agentic coordination.

Workflow	Why a Simple Model Is Insufficient	Controlled Agentic Role
Client investment instruction	Intent, suitability, mandate, tax, liquidity, and product eligibility interact.	Retrieve evidence, form a proposal, route for approval, block inadmissible products.
Execution recommendation	The next action depends on live fills, liquidity, urgency, and market impact.	Coordinate market tools and propose a typed child-order instruction.
Collateral dispute	Legal terms, valuation data, trade state, and communications conflict.	Assemble evidence, call valuation tools, propose accept/reject/escalate.
Credit monitoring	News affects exposures, covenants, ratings, hedges, and committee workflows.	Build an evidence packet and recommend controlled escalation.
Compliance communication	Language must satisfy product, client, jurisdiction, and disclosure constraints.	Draft and redline, while validators enforce policy and record evidence.
Treasury hedge exception	Business urgency conflicts with limits and missing documentation.	Prepare the exception packet; do not execute until authority is granted.

- 1. Official arithmetic:** pricing, risk, margin, exposure, capital, and P&L calculations must be performed by approved numerical systems.
- 2. Admissibility:** mandate, suitability, restricted-list, liquidity, concentration, legal, and approval constraints must be checked by versioned validators.
- 3. Execution authority:** any action that mutates orders, ledgers, client records, limits, collateral, or cases must pass through a permissioned execution adapter.
- 4. Evidence ownership:** the institution, not the model, must own source control, timestamps, hashes, approvals, and audit records.
- 5. Objective definition:** the model must not silently define success. P&L, risk, conduct, service quality, and auditability must be monitored explicitly by $\mathcal{J}$.

This boundary is not anti-agent. It is the condition that makes agents usable in finance. Without it, an agent is a fluent stochastic interface to unbounded institutional authority. With it, an agent is a constrained coordinator of retrieval, tool use, proposal generation, and evidence production.

1.13 The Deterministic Core Is Not Magic

The deterministic core is not called deterministic because it is automatically correct. It is called deterministic because its semantics can be specified, tested, replayed, versioned, reviewed, and suspended. This distinction matters. A stale rule, a bad reference-data feed, a mis-specified policy exception, a broken entitlement map, a numerical-tolerance

bug, a solver failure, or a schema-version mismatch can make a deterministic control wrong. Determinism removes stochastic generation from the commitment path; it does not remove ordinary software, data, and governance risk.

Protocol Rule 1.4 (Validation obligations of the deterministic core). *Every deterministic component that can affect an admissible financial action must carry a control file with five artifacts:*

1. *the authoritative specification or policy it implements;*
2. *test vectors, including edge cases and known incidents;*
3. *versioned dependencies, data sources, tolerances, and fallback rules;*
4. *an owner with change authority and an independent reviewer;*
5. *monitoring that detects stale rules, failed lookups, solver failures, and exception drift.*

A deterministic shield without these artifacts is only a coded opinion. A deterministic shield with these artifacts is a reviewable institutional control.

This caveat strengthens the book’s thesis rather than weakening it. The financial reason to prefer deterministic commitment layers is not that legacy code is virtuous. It is that deterministic failure modes can be localized, replayed, tested, patched, and assigned to accountable owners. A prompt failure often leaves only a transcript. A control failure can leave a failing test.

1.14 The Distributional Shield as the Running Implementation

The companion implementation in the official GitHub repository, <https://github.com/Davidpazn/agnostic-finance-distributional-shield>, instantiates the bridge. Its core idea is simple: risk is a property of the resulting portfolio, not of a proposed trade in isolation. The shield evaluates the post-action state $w_t + a_t$ under a risk policy and returns allow, reject, or escalate. The notebooks implement typed action/state contracts, a synthetic market and covariance model, deterministic checks, distributional VaR/CVaR/tail-probability gates, liquidity and concentration constraints, option-delta examples, DV01 checks, basket rebalances, stress regimes, heavy-tailed scenarios, model-card summaries, and consolidated CSV audit logs.

The multi-agent extension adds specialist advisers: research, technical, risk, compliance, and portfolio-management roles. They can recommend, challenge, escalate, or reject, but they cannot bypass the shield. The negotiator extension adds bounded counteroffer logic: compliance can hard-block restricted symbols, liquidity can shrink notional size, distributional risk can search for the largest survivable notional, and execution can attach terms such as staged VWAP, participation caps, child orders, and kill switches. Negotiation may only reduce or condition risk; it may not relax a constraint. The agentic RAG layer then builds a retrieval corpus from the shield’s audit logs and answers governance questions from evidence. It explains decisions; it does not authorize them.

Table 1.4 Operational layers and rules for bounded agentic finance.

Layer	Operational Rule
Typed contracts	Malformed actions fail closed before risk is even computed.
Distributional shield	VaR, CVaR, tail probability, concentration, liquidity, and instrument-specific rules are checked on the resulting portfolio.
Multi-agent committee	Agents provide opinions and diagnostics, but executable authority remains with the shield.
Negotiator shield	Counteroffers can shrink or condition a proposal, then the final proposal is re-evaluated by the shield.
Agentic RAG	Explanations cite audit evidence and escalate when evidence is weak, missing, or contradictory.
Audit logs	Every proposal, vote, offer, verdict, metric, and rule check is written as replayable evidence.

Table 1.4 consolidates the operational layers that keep language at the boundary and authority inside contracts.

1.15 Design Maxim

The bridge can now be reduced to one design maxim:

Let language enter at the edge, but let authority pass only through contracts.

This sentence is not a slogan. It is the architecture. The language layer is useful because it can interpret, summarize, retrieve, draft, and propose under messy evidence. The contractual layer is necessary because financial authority requires typed state, admissible actions, deterministic checks, distributional risk controls, execution permissions, and replayable memory. Agentic finance is therefore not the replacement of financial engineering by language models. It is the embedding of language models inside financial engineering.

1.16 Mapping Back to $\mathfrak{A}$

The chapter defines all six components of $\mathfrak{A}$: ρ as evidence engineering, π as proposal generation, Π as admissibility, $\mathcal{E}$ as narrow execution, $\mathcal{M}$ as replayable memory, and $\mathcal{J}$ as monitored institutional objective.

