Chapter 2

Decision-basis failure

I used the term *decision-basis failure* a dozen times in the first chapter without formally defining it. This chapter does that. The definition matters because the vocabulary we use to name a problem shapes what we can see, measure, and respond to. “Decision-basis failure” has specific contours that distinguish it from adjacent failure modes, and getting those contours right is the first step toward being able to engineer against it.

Formal definition

Decision-basis failure is a condition in which an autonomous or semi-autonomous decision system executes a decision, every operational check in the execution stack passes, and the decision is nevertheless wrong — where *wrong* is measured against the institutional judgment the decision-making organization has encoded about this class of decision.

Three parts of this definition carry weight.

First: *autonomous or semi-autonomous*. The failure mode requires that the system acts, not merely recommends. A recommender system can give a bad recommendation, but if a human then decides based on the recommendation, the failure is not a decision-basis failure of the system — it is a human decision failure that happened to use a bad recommendation. Decision-basis failure specifically describes the case where the system itself is the deciding entity.

Second: *every operational check in the execution stack passes*. The failure is silent to the current tooling. If a check caught it, it would be a different kind of failure — a security failure, a compliance failure, a technical bug. Decision-basis failure is specifically the residual that remains after the execution stack has said everything is fine.

Third: *wrong, measured against institutional judgment*. This is where the definition departs from a merely technical one. “Wrong” is not an abstract property of the decision. It is a property of the fit between the decision and the specific organization’s accumulated judgment about how this class of decision should be made. The same decision in a different organization might not be wrong. The judgment is institutional — not personal, not legal, not universal.

The reason the definition is this specific is that it names what the governance mechanism needs to verify against. If “wrong” were universal, you could encode it once and distribute. Because “wrong” is institutional, each organization needs its own Governance Blueprint.

The definition also implies a measurable test. For each decision class, the organization can name the considerations a competent decider in that organization must weigh: the claimant’s recent history, the downstream consumers of a table, the ownership chain of a supplier. Call that set the *required basis*. A decision-basis failure is a decision whose declared basis omits a member of the required basis:

decision-basis failure(d) if and only if $R(D) \setminus B(d) \neq \varnothing$

where $R(D)$ is the signed required basis for decision class D and $B(d)$ is the basis decision d actually declared and evidenced. The test does not ask whether the outcome was bad. It asks whether what should have been considered was considered. That is why the failure can be caught before execution, and why it cannot be caught by any tool that looks only at the action.

What decision-basis failure is not

The definition is cleaner when you contrast it with adjacent failure modes.

Decision-basis failure is not an execution failure. An execution failure is a problem in the sequence of actions the agent takes — a tool call that fails, a timeout, a null pointer, a race condition. Execution failures are caught by the existing stack: observability, logs, retries. If the decision does not actually get made because of an execution failure, there was no decision to have a bad basis.

Decision-basis failure is not a hallucination. A hallucination is a model output that is factually incorrect — the model asserts something that is not true about the world. A hallucination might trigger a decision-basis failure, but the two are distinct. The claims agent in the insurance case did not hallucinate anything. Every fact it retrieved was correct. The agent did not consider the right facts.

Decision-basis failure is not misalignment in the technical AI safety sense. Misalignment, in that literature, refers to a system optimizing for a goal that is systematically different from the goal its designers intended. The agents in my three cases were not misaligned in that sense — they were optimizing exactly for what the workflow specified. The problem is that the workflow specified a subset of what the organization actually required.

Decision-basis failure is not a policy violation. A policy violation is when the agent does something the organization has explicitly forbidden. Policy violations are catchable with policy-as-code tools, access controls, audit

requirements. Decision-basis failure is when the agent does something the organization has not explicitly forbidden, has not explicitly required, but a competent decision-maker in the organization would have handled differently because of context the agent did not consult.

Decision-basis failure is not an adversarial attack. An adversarial attack is when an actor — human or automated, external or internal — manipulates an agent's inputs, prompts, tools, or workflow to produce an outcome the attacker wants. Prompt injection is an adversarial attack. Jailbreaking the agent's constraints is an adversarial attack. Supply chain compromise that modifies the agent's model or tools is an adversarial attack. Social engineering of a human reviewer whose approval the agent depends on is an adversarial attack. Each of these is a cybersecurity concern with established defenses: input sanitization, prompt injection detection, model integrity verification, zero-trust architectures, red-teaming.

Decision-basis failure is none of these. In decision-basis failure, nobody is attacking the agent. The agent is doing exactly what it was built to do. The organization simply did not fully specify what should have mattered for the decision, and the specification gap produced a decision that violated institutional judgment that was never encoded. The failure originates from inside the organization's specification process, not from outside manipulation of the operational pathway.

The distinction matters for two operational reasons. First, defending against adversarial attack does not defend against decision-basis failure, and vice versa. An organization with excellent cybersecurity can still produce decision-basis failures at scale; an organization with excellent governance infrastructure can still be compromised by adversarial attack. Both defenses are needed and must operate together. Second, the two failure modes have different origins and require different responses. Treating one as if it were the other is a category error that leaves both inadequately defended. Chapter 28, which opens Part VII, develops the relationship between the two failure modes and the integrated defense posture that production enterprise agentic deployment requires.

Decision-basis failure is not a model quality problem. Better models would not have changed the three cases in Chapter 1. A perfectly capable release agent with access to a frontier model would have done exactly what the actual agent did, because the information required to decide differently was not in its workflow at all. Model quality is upstream of decision-basis failure; it is a necessary condition for good autonomous decision-making but not a sufficient one.

Naming what decision-basis failure *is not* is almost as important as naming what it is, because the industry's instinct when a bad agent decision surfaces is to attribute it to one of the adjacent categories — the model, the workflow, the

policy configuration — and to invest in fixing that category. The investment often does not fix the actual problem, because the actual problem is at a different layer.

The anatomy

A decision-basis failure has a standard anatomy. The cases I have observed follow this structure.

Step 1: The organization has, somewhere, a piece of institutional judgment relevant to this class of decision. It may be written down (in a policy document, a Confluence page, a PDF binder). It may be unwritten (lodged in a specific person's head, enforced through social practice). It may be lodged in both (written down but also carried by someone who remembers *why* the rule exists). The judgment is real. It would change the decision if applied.

Step 2: The agent system is designed, built, and deployed. The design process consults engineering, product, perhaps security and compliance for the obvious items. It does not systematically survey every piece of institutional judgment relevant to the class of decision, because no such survey mechanism exists in most organizations. The relevant judgment is in forms that no one has turned into a reference an agent's decisions could be checked against.

Step 3: The agent begins making decisions in production. Each decision is scored against the operational checks the workflow does include. These checks catch the failure modes they were designed to catch — unauthorized access, policy violations, obvious technical errors. The decisions pass.

Step 4: Some number of decisions execute that would have been different if the missing institutional judgment had been applied. These are the decision-basis failures. In the deployments I have observed, the rate in the first months is a few percent of all agent decisions, highest where pre-existing policy was least documented, and it declines as Blueprints are built out without ever reaching zero (see the Notes on the author's own figures).

Step 5: A subset of the decision-basis failures surface as downstream problems — a fraud ring that gets missed and costs money, a compliance exposure that triggers regulatory attention, a release that breaks a downstream system. The rest never surface, which is in some ways the worst outcome, because the organization never learns that it has a systemic blind spot.

Step 6: When a surfaced failure is investigated, the common response is to patch the specific workflow — add a check for *this* specific pattern — and treat the problem as solved. The next decision-basis failure, caused by a different piece of institutional judgment that was also missed, is treated as a new and unrelated problem.

The pattern of patch-and-move-on is the industry's current response, and it is why the problem keeps recurring. Each patch is a point fix. The underlying architectural gap — that institutional judgment has no systematic path from where it lives in the organization to a reference against which agent decisions can be checked — remains open.

The failure seen as two trajectories

There is a simpler way to see the anatomy. Every decision has a **decision trajectory**: how it develops toward an outcome, what information is considered, which conditions matter, what is omitted, and when someone else should be involved. A workflow is something different. It sets which actions happen, in what order, with which tools. The carrier's workflow ran exactly as designed. Its decision trajectory did not.

Two trajectories matter for any consequential decision. The **observed trajectory** is the one the agent actually took, as far as its evidence, tool use and declared reasoning let anyone see. The **expected trajectory** is the one the organization's judgment requires: the considerations, dependencies, boundaries and escalation points any sound decision of this class must include. It is not one employee's reasoning copied step by step; experienced people reach sound decisions by different paths, and the expected trajectory holds only what every sound path must contain.

A note on the word, because engineers already use it. In agent frameworks, a *trajectory* is the sequence of messages and tool calls an agent produces, and evaluation tools score that sequence against an expected one (Chapter 3). The decision trajectory of this book is a different object, read from the same log: a sequence of considerations, not of calls. Two agents can produce the same tool calls on different grounds, and the same grounds with different calls. Where this book says trajectory, it means the considerations.

In these terms, a decision-basis failure is an observed trajectory that skips something the expected trajectory requires, while every check on the workflow passes. Most organizations have their workflows documented. Almost none have their expected trajectories written down, which is why the failure cannot be seen. Part V describes how the expected trajectory is made explicit, owned and compared.

Why it will proliferate

Decision-basis failure is going to become more common and more consequential over the next five years. The public record already points that way: a tribunal held Air Canada liable for what its customer-facing chatbot promised (Moffatt v.

Air Canada, 2024); a coding agent deleted a production database and then misreported what it had done (eWeek, 2025); and an industry survey in 2026 found agent incidents already common in enterprises (Cloud Security Alliance, 2026). Not all of these are decision-basis failures in the strict sense, but each is an agent acting on a basis nobody had verified. Three forces drive the growth.

Deployment scale. The number of enterprise agentic systems in production will grow by at least an order of magnitude in that period. Even if the per-system failure rate stays constant, the aggregate rate of decision-basis failures will scale accordingly.

Decision complexity. The decisions that agents are being asked to make are moving up the organizational stack — from simple classification and recommendation to multi-step consequential action involving money, contracts, obligations, and precedent. More consequential decisions have more institutional judgment attached to them, and therefore more surface area for decision-basis failure.

Interconnection. Agents are increasingly chained together. The output of one agent becomes the input of the next. A decision-basis failure early in the chain propagates through subsequent agents, each of which processes the output as if it were correct, because the failure is silent at every layer. Each chained agent can amplify the consequence, and in the chains I have traced the amplification per step has often been of the order of tenfold; the figure is an observation, not a law.

If you take these three forces together — more systems, bigger decisions, longer chains — the aggregate exposure of a typical enterprise to decision-basis failure in 2030 will be at least two orders of magnitude greater than today, in a context where the governance mechanisms for this failure mode are still mostly in prototype.

That is why this problem is urgent. It is not that any individual decision-basis failure is apocalyptic. Most are not. It is that the aggregate rate is going to grow fast, and the mechanism for catching them does not yet exist at scale, and the organizations that build the mechanism first will have a durable advantage over those that build it in response to the first high-profile incident.

Decision-basis failure within the broader drift taxonomy

Decision-basis failure sits inside a wider framework I published in October 2025 (Ledesma, SSRN), and the wider frame matters for what the architecture is and is not built to do.

Decision-basis failure, as this book defines it, is one specific instance of a wider phenomenon I have termed *synthetic drift* — the condition in which an agent

system's reasoning or outputs diverge from the human intent that originally authorized it. Synthetic drift is one of three coupled drift modes that threaten the coherence of human-AI governance.

The other two are:

Human drift. The condition in which humans, by unconsciously over-delegating critical functions to agent systems, lose agency, presence, or coherence. The compliance officer who stops reviewing outputs because the automated reports consistently appear accurate is experiencing human drift. The founder whose cognitive load is being carried by AI tools to the point that they can no longer explain their own strategic rationale is experiencing it too. Human drift is the subject of a companion volume to this book, *The Inner Edge*.

Historical drift. The condition in which an autonomous system retrospectively reconstructs or selectively rewrites its own cognitive history — its chain of reasoning, its evidential trail, its memory of why past decisions were made. Historical drift is the most insidious of the three, because once it has occurred, no actor can be reliably held responsible for anything. The architectural response to historical drift is the Governance Ledger (Chapter 18), which preserves the record of what actually happened in a form that cannot be rewritten.

The three drift modes are not independent occurrences; they are mutually reinforcing. As synthetic drift accelerates, humans tend to over-delegate (human drift) because the AI's output appears coherent. As humans disengage, agents are freer to drift (synthetic drift). As drift accumulates, systems trained for self-justification generate post-hoc rationalizations (historical drift) that make the earlier drift difficult to recover from.

This book addresses synthetic drift primarily, with specific focus on decision-basis failure as its most consequential manifestation in enterprise contexts. The other two drift modes receive brief treatment where they affect the architecture (human drift is addressed in Chapters 5–6 through the attention budget; historical drift is addressed in Chapter 18 through the Governance Ledger), and are the focus of other work in the broader program this book is part of.

The three-drift framing is important because it clarifies what the architecture of Parts V and VI is and is not designed to do. The architecture is a specific response to synthetic drift, with specific protections against historical drift. It does not solve human drift, which requires a different kind of intervention operating at the layer of individual human cognition and organizational practice. Each drift mode requires its own response. The three responses together constitute what I have begun calling, in other work, an integrated governance architecture for agentic enterprises.

The question that organizes the rest of the book

If decision-basis failure is a real and growing failure mode, and if the existing execution stack cannot catch it, then the question that organizes the rest of this book is this:

What is the architecture — organizational, technical, and theoretical — that catches decision-basis failure at machine speed, in production, consistently, across the full surface of an organization's agent deployments?

The answer, which the book develops in the chapters ahead, has three layers.

At the *theoretical* layer, we need a reconstruction of management theory that takes the decision as the fundamental unit, treats it as an object with measurable properties, and derives the organizational shape change (from pyramid to network) that follows. Parts III and IV of the book build this.

At the *organizational* layer, we need a new way of surfacing, naming, and signing institutional judgment — a Governance Blueprint constructed collaboratively by the responsible parties for each domain. Part V describes the artifact and the process.

At the *technical* layer, we need an architecture that evaluates each agent decision against the signed graph at the right scrutiny level, in real time, with a graduated response ladder when something fails. Part V also describes the architecture.

Before we get to any of that, we need to understand why the existing tools do not catch this failure mode. That is the subject of Chapter 3.

The full pre-release edition of *The Decision* and every chapter are available from MeaningStack. Contact: luciana@meaningstack.com