# LOCALLY SAFE, GLOBALLY UNSAFE: COMPOSITION-INDUCED FAILURE IN NETWORKED AI AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Autonomous agents are increasingly certified one at a time and then deployed in groups onto shared infrastructure. We show that this practice certifies almost nothing. Our running case is an autonomous cloud whose deployment, reliability and capacity agents each satisfy their own specification and which, composed, take the service down with no agent making a bad decision. The mechanism is a *regenerative dissipative interaction cycle*: a loop that rebuilds its own trigger while consuming a shared resource. In a Petri-net formulation we prove a depletion bound and show it is attained, that screening for such a cycle is a linear program rather than a search over markings, that one matrix product per agent pair decides whether a local certificate survives composition, and that a cycle consuming nothing leaves the probability of failure exactly unchanged while inflating the time to it by up to $2\times$ in the running example.

The evidence comes from artifacts built by other people. AppWorld's authors wrote, for each of $486$ tasks, an assertion naming which database models a correct run may change; read as local certificates, only $21.0\%$ of $235{,}710$ ordered task pairs survive composition, falling to $5.7\%$ once twenty undocumented cross-app effects are counted. Two task executors sharing one live AppWorld account then take the benchmark's own score from $24/24$ to $6/24$, and cutting exactly the fusions our test flags restores it where the intuitive repair does not. Benchmarks that score one agent at a time cannot certify behavior in a world that agent shares.

## 1 INTRODUCTION

A system built from safe parts need not be safe. This is old news in concurrency, distributed systems and control, but it is becoming urgent for AI, because agents no longer merely exchange text. They act, and they are deployed in groups onto the same infrastructure. Cloud vendors now ship autonomous mitigation agents with execution guardrails (Google, 2026), DevOps agents that investigate and respond to production incidents on their own (Amazon Web Services, 2026; 2025; PagerDuty, 2025), and multi-agent frameworks (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Li et al., 2023) make a crew of agents a default architecture. The trajectory is not one agent with tools. It is an autonomous software factory in which deployment, reliability, capacity and security agents, each built by a different team and benchmarked alone, act on shared infrastructure at machine speed.

That is exactly the configuration in which local guarantees stop meaning what they appear to mean. Practitioners know it: Google's guidance pairs per-action guardrails with explicitly system-level machinery, naming cascading feedback, centralized actuation verification, capacity checks, circuit breakers and global rate limits (Google, 2026). What is missing is an account of *why* that layer is unavoidable, and of what exactly the local layer fails to certify. Section 3 supplies one: three agents, each satisfying a per-action rule of the kind a guardrail expresses, that jointly take a service down. It is a counterexample to local-only certification, not to any published architecture; those safeguards are, in the vocabulary of this paper, precisely cuts to the couplings our screen flags. The question is not whether agents are individually well designed, but which behaviors become possible only because they are composed:

$$\mathcal{N}_i \models \varphi_i \;\; \forall i \quad \not\Longrightarrow \quad \|_i \mathcal{N}_i \;\models\; \varphi_{\mathrm{sys}}. \tag{1}$$



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

**The same gap sits inside how we evaluate agents.** An agentic benchmark scores one agent at a time on a private copy of a world, and to make scoring possible it must say what a correct run is allowed to change. AppWorld (Trivedi et al., 2024) says it precisely: for 486 of its tasks the evaluator asserts `changed_model_names == {...}`, meaning these database models changed *and no others did*. Read as a formal object that is a local certificate of exactly the kind Theorem 3 is about, written by people testing tool use rather than composition. Composing those certificates against each other therefore tests the benchmark's implicit assumptions at one matrix product per pair, and invents nothing. It fails for $79.0\%$ of ordered pairs — not a flaw in AppWorld's assertions, which are right for one agent, but a property of per-agent evaluation itself: a score obtained under exclusive access to an environment does not transfer to a shared one.

**Why a Petri net.** The failure is a *resource* failure, a *composition* failure in which composition is shared state rather than declared channels, and a failure that is *possible* before it is *likely*. Nets make all three first-class: the resource is a place, the spend an arc, coupling is place fusion, and a stochastic layer on the same structure (Ajmone Marsan et al., 1995; Molloy, 1982) separates whether from when, while the static analysis stays linear algebra on the incidence matrix. Appendix A compares the alternatives one at a time.

**Contributions.**

(i) **A failure you can check by hand** (§3): three cloud agents, each provably unable to exhaust healthy capacity, whose composition provably does (Prop. 1), plus a second pathology in which two of them undo each other forever (Prop. 2).
(ii) **The mechanism has a name and a bound** (§4.2): a *regenerative dissipative interaction cycle* rebuilds its own trigger while spending a shared resource; Thm. 1 bounds the rounds to failure and Cor. 1 shows the bound is attained.
(iii) **Finding one is a linear program** (§4.3), not a search over markings (Thm. 2), with Prop. 5 closing the state-equation/reachability gap on the class the program returns.
(iv) **A decidable test for whether a local certificate survives** (§4.4): one matrix product per agent pair (Thm. 3, Prop. 4), with emergence monotone in the coupling (Prop. 3), so cutting an interface that starts empty can never introduce a hazard.
(v) **Not every cycle is dangerous** (§4.5): a cross-agent cycle consuming nothing leaves the probability of failure exactly unchanged while strictly increasing the time to it (Thm. 4).
(vi) **Validation on our nets and on other people's artifacts** (§5): exact reachability, exact Markov-chain analysis and a scaling study confirm every claim; $79.0\%$ of $235{,}710$ ordered AppWorld task pairs then break, rising to $94.3\%$ once twenty undocumented cross-app effects are counted, and two executors on one live account take AppWorld's own score from $24/24$ to $6/24$, restored by cutting exactly the fusions Thm. 3 flags. On MAST (Cemri et al., 2025), $14.1\%$ of $1{,}237$ annotated failures carry an inter-agent fault and no local one.

**Scope, and one word used narrowly.** We assume each agent's execution is already mediated and behaves as specified: no hallucination, no prompt injection, no adversary. The failures studied here are what remains after that assumption holds, which is why component-level assurance cannot see them. Throughout, *locally safe* means *satisfies its declared local specification* $\varphi_i$ and nothing stronger, not safe under all environmental assumptions. Each of the three agents in §3 is locally safe in this sense and none is safe in the stronger one, because each was specified against an environment in which the other two do not act.

## 2 Setup: nets, and what composition means

A **place** (circle) holds **marks** (dots) and denotes a condition or a countable resource; a **transition** (bar) is an event, enabled when all its input places hold enough marks, and firing moves marks accordingly. Petri-net texts say *token*; since that word already denotes what a language model consumes we say **mark** throughout. Two facts do all the work below: a transition needs *all* of its inputs, and firing *spends* them, which is why depletion rather than mere sequencing is expressible. Formally $\mathcal{N} = (P, T, Pre, Post, M_0)$ with $Pre, Post \in \mathbb{N}^{|P|\times|T|}$; $t_j$ is enabled at $M$ iff $M \geq Pre(:, t_j)$, and firing gives $M' = M + C(:, t_j)$ with $C := Post - Pre$. The reachable set $\mathcal{R}(\mathcal{N}, M_0)$ says which configurations can occur and the firing language $L(\mathcal{N}, M_0)$ which event sequences are executable;

we define emergence on the language and hazard on the reachable set. A **linear place invariant** is $w$ with $w^\top C = 0$, giving $w^\top M = w^\top M_0$ along every firing sequence. Appendix B is a self-contained primer, with Figure B1 illustrating firing and fusion.

**Definition 1** (Interface places, fusion map, composition)**.** Let $\mathcal{N}_i = (P_i, T_i, Pre_i, Post_i, M_i^0)$, $i = 1, \ldots, n$, have pairwise disjoint places and transitions, and let each declare *interface places* $I_i \subseteq P_i$, those through which it reads from or writes to a shared environment; the rest are *internal*. A *fusion map* $\Phi$ is an equivalence relation on $\bigsqcup_i P_i$ such that every class either is a singleton containing one internal place, or is a set of interface places holding at most one place from each component. Initial markings must be *compatible*: all places in a class carry the same initial mark count, written $M_\Phi^0(\gamma)$. The *composition* $\|_\Phi \mathcal{N}_i$ has places $P_\Phi = (\bigsqcup_i P_i)/\Phi$, transitions $\bigsqcup_i T_i$, and for each class $\gamma \in P_\Phi$ the entries $Pre_\Phi(\gamma, t) = \sum_{p \in \gamma} Pre_i(p, t)$ and $Post_\Phi(\gamma, t) = \sum_{p \in \gamma} Post_i(p, t)$.

Place fusion is the right operator because these agents do not exchange typed messages over declared channels: they read and write a shared substrate, and one agent's output signal is literally another's input signal. A fused place, drawn dashed and written $\gamma$, holds places from two or more components, while internal places pass through unchanged, the formal content of "nothing inside a component is modified". Fuse nothing and there is no emergence. The results below hold for any operator that lets one component put marks into places another reads, so explicit channels would serve equally; under transition synchronization composition can only remove behavior. Appendix D derives a composed net from three separate teams' orchestration configs, which is where the fusions are actually discovered, and Appendix F treats the additive reading of a shared pool.

## 3 The autonomous cloud meltdown

Consider a platform running $N$ healthy production instances under three autonomous agents of the kind cloud vendors now ship (Google, 2026; Amazon Web Services, 2026; PagerDuty, 2025): a deployment agent $A_D$, a reliability agent $A_S$ and a capacity agent $A_C$, each built by a different team and passing its own benchmark. Write $R$ for a release ready to ship, $V$ for the release being live, $U$ for measured CPU pressure, $H$ for healthy instances, $K$ for a capacity deficit, $Q$ for quarantined instances and $B$ for a suspected regression. The rules are

$$\begin{aligned} A_D: &\quad t_D: R \to V + U, && t_{D'}: B \to R, \\ A_S: &\quad t_S: U + H \to Q + K, && t_{S,\mathrm{rb}}: U + V \to B, \\ A_C: &\quad t_C: K \to U. \end{aligned} \tag{2}$$

That is, $t_D$ ships a validated release whose warm-up transiently raises CPU; $t_S$ quarantines one instance showing an anomaly; $t_{S,\mathrm{rb}}$ rolls back the release correlated with the anomaly; $t_{D'}$ re-attempts a withdrawn release; and $t_C$ reroutes traffic off lost capacity, raising CPU on the survivors. Each rule is what the corresponding production system is supposed to do. Let the unsafe set be total loss of serving capacity, $\mathcal{B} = \{M : M(H) = 0\}$, and the local specifications be $\varphi_D, \varphi_C$: "never decreases $M(H)$", and $\varphi_S$: "removes at most one instance per anomaly consumed". All three hold, for the reasons in Figure 1(a). Now run the agents on one platform, which fuses the interface places $U$, $K$, $V$ and $B$. A release ships, warm-up raises CPU, the reliability agent reads elevated CPU as an anomaly and quarantines an instance, the capacity agent reroutes traffic onto the survivors, their CPU rises, and the reliability agent quarantines another. The interface returns to exactly the state that started the round, but one instance is gone:

$$M(U) = 1,\ M(K) = 0,\ M(H) = k \quad \xrightarrow{t_S\, t_C} \quad M(U) = 1,\ M(K) = 0,\ M(H) = k - 1. \tag{3}$$

**Proposition 1** (Local safety does not imply compositional safety)**.** *For every $N \geq 1$, $A_D$ and $A_C$ in isolation cannot change $M(H)$, $A_S$ in isolation satisfies $M(H) \geq N - 1$, and yet $M_0 \xrightarrow{\sigma_N} M_B$ with $M_B(H) = 0$, where $\sigma_N = t_D\,(t_S t_C)^{N-1}\, t_S$ has length $2N$ and is a shortest such sequence. Hence $\varphi_D, \varphi_S, \varphi_C$ all hold while $\varphi_{\mathrm{sys}}$, namely $M(H) \geq 1$, fails, an instance of* (1)*.*

*Proof.* $P_D$ and $P_C$ do not contain $H$, so no column of $C_D$ or $C_C$ is nonzero on $H$. In $\mathcal{N}_S$ alone $U$ has no producer, so $t_S$ fires at most once. In the composition $t_D$ produces one mark in $U$; then (3) applies whenever $M(H) \geq 1$, and $N$ firings of $t_S$ take $M(H)$ from $N$ to 0, the $N$th enabled because $M(H) = 1$ at its start. Only $N - 1$ firings of $t_C$ are needed, since after the last $t_S$ the marking is already in $\mathcal{B}$. Minimality is Prop. 6; Appendices F and H give the trace and the full argument. □

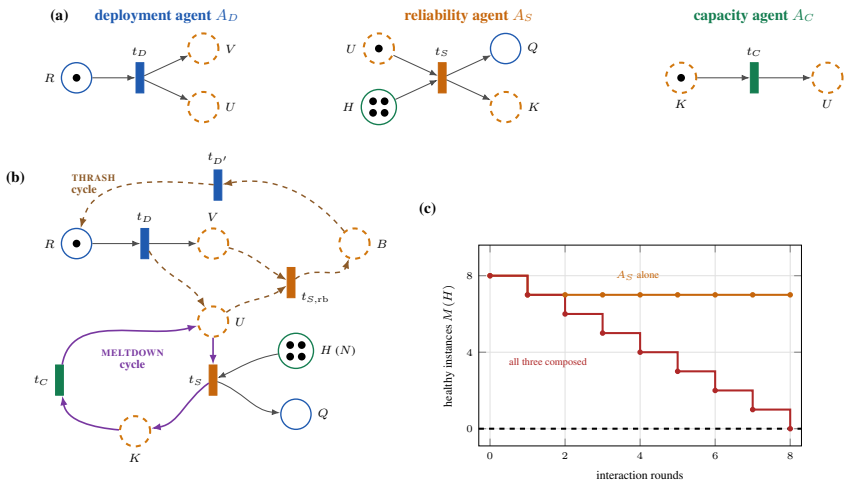



Figure 1: **The whole paper in one figure.** **(a)** Three agents, each provably satisfying its own specification; dashed orange circles are *interface* places, signals an agent reads from or writes to a shared environment. **(b)** Running them on one platform fuses $U, K, V, B$. Nothing inside any agent is modified, yet two directed cycles appear that exist in no component: the violet MELTDOWN cycle spans $A_S$ and $A_C$ and destroys one mark of $H$ per traversal while restoring the interface exactly, and the brown dashed THRASH cycle spans $A_D$ and $A_S$ and consumes nothing, so by Theorem 4 it changes when the system fails and not whether. **(c)** The consequence for $N = 8$: component testing sees one quarantine and a floor at $N - 1$, the deployed composition reaches zero.

**Proposition 2** (Mutual undoing)**.** *Each isolated net is acyclic and every firing sequence in it is finite. The composition admits the infinite sequence* $(t_D\, t_{S,\mathrm{rb}}\, t_{D'})^\omega$*, along which* $V$ *is marked and unmarked infinitely often and no terminal marking is reached.*

The two cycles compete for the same CPU mark, which §4.5 shows has a precise and initially surprising consequence.

## 4 Why it happens, and how to find it without enumerating

### 4.1 Emergent traces, defined against a baseline

Let $L_i = L(\mathcal{N}_i, M_i^0)$ be the isolated firing language of agent $i$. Were the agents independent, joint behavior would be the shuffle language $L_\otimes = \mathrm{Shuffle}(L_1, \ldots, L_N)$, while the composed system has $L_\parallel$.

**Definition 2** (Composition-emergent trace)**.** $\sigma$ is *composition-emergent* if $\sigma \in L_\parallel$ and $\sigma \notin L_\otimes$.

In the meltdown $t_D\, t_S\, t_C\, t_S$ is emergent: after the first $t_S$ the isolated reliability net cannot fire $t_S$ again, since nothing in it produces $U$, and the capacity agent supplied the missing mark. The definition asks nothing mystical, only that the behavior be absent from an explicitly declared independent baseline and enabled by coupling. Emergence is not hazard, since coordination is emergent too. Hazard is a separate predicate on the reachable set: a *composition-induced safety violation* occurs when the components satisfy their specified local safety conditions while $\mathcal{R}(\mathcal{N}_1 \parallel \cdots \parallel \mathcal{N}_N, M_0) \cap \mathcal{B} \neq \varnothing$. The operative phrase is "specified local": a global property over shared resources need not decompose into independent local predicates, because local tests omit exactly the coupling variables that in isolation never move. Two structural objects organize the rest, both in Appendix F: the *agent interaction graph* $G_A$, with an edge $i \to j$ when a transition of $i$ can feed an interface place consumed by $j$, and a *cross-agent causal cycle*, whose transitions carry at least two agent labels.

**Proposition 3** (Monotonicity in the coupling)**.** *Let* $\Phi' \subseteq \Phi$ *be fusion maps, and compare the two compositions, which share a transition set but not a place set, through the* aggregation map $\alpha : \mathbb{N}^{P_{\Phi'}} \to \mathbb{N}^{P_\Phi}$, $\alpha(M)(\gamma) = \sum_{\gamma' \subseteq \gamma} M(\gamma')$*, well defined because each* $\Phi'$*-class lies inside one*

*$\Phi$-class. Suppose the merged classes start empty. Then $L_{\Phi'} \subseteq L_\Phi$ and $\alpha(\mathcal{R}_{\Phi'}(M^0_{\Phi'})) \subseteq \mathcal{R}_\Phi(M^0_\Phi)$, with $L_\otimes$ attained when $\Phi'$ is the identity.*

Emergence therefore does not come from having many agents; it comes from coupling. A tenth isolated agent adds nothing, whereas one edge closing a cycle can change everything. The proposition also licenses the repair strategy of §4.3 when the merged classes start empty, since removing such a fusion removes firing sequences and so makes no hazardous marking newly reachable; it does *not* follow that the cut system is better in every respect, because a cut can deadlock a component or strand another agent's goal, which is the trade E6 measures.

### 4.2 Regenerative dissipative interaction cycles

A cross-agent cycle alone is not the pathology, since the thrash cycle consumes nothing. The meltdown cycle *regenerates the condition that enables it* while *consuming a finite resource*. Let $I \subseteq P$ be interface places with projection $\pi_I(M)$, and let $V(M) = v^\top M$ with $v \geq 0$ be a nonnegative resource functional: healthy instances, scale-out quota, remaining budget, unspent rate limit. A firing sequence $\sigma$ is *interface-regenerative* at $M$ if $M \xrightarrow{\sigma} M'$ with $\pi_I(M') = \pi_I(M)$, and *$d$-dissipative* if additionally $V(M') \leq V(M) - d$ for some $d > 0$; if it uses transitions of at least two agents we call it a **regenerative dissipative interaction cycle**. The meltdown's $\sigma_c = t_S t_C$, with $I = \{U, K\}$ and $V(M) = M(H)$, is 1-dissipative; if the capacity agent may also scale out against a quota it is 1-dissipative in $V(M) = M(H) + M(\beta)$ instead (Remark 1).

**Theorem 1** (Depletion)**.** *Let $\mathcal{B}$ contain all markings with $V(M) < V_{\min}$. Suppose there is a reachable $M^\star$ and a cross-agent sequence $\sigma$ such that (i) for every reachable $M$ with $\pi_I(M) = \pi_I(M^\star)$ and $V(M) \geq V_{\min}$, the whole of $\sigma$ is executable from $M$; (ii) $M \xrightarrow{\sigma} M'$ implies $\pi_I(M') = \pi_I(M)$; and (iii) $V(M') \leq V(M) - d$ with $d > 0$. Then a bad marking is reachable from $M^\star$ after at most $K = \min\{k \in \mathbb{N} : V(M^\star) - kd < V_{\min}\}$ repetitions of $\sigma$.*

**Corollary 1** (The repetition bound is attained)**.** *Suppose in addition that the net's only $V$-decreasing transition occurs once in $\sigma$ and decreases $V$ by exactly $d$. Then every firing sequence from $M^\star$ into $\mathcal{B}$ contains at least $K$ $V$-decreasing firings, and $K$ repetitions of $\sigma$ achieve it.*

Hypothesis (i) must reach down to $V(M) = V_{\min}$ and not to $V_{\min} + d$, since it is the last traversal that does the damage and it starts from a marking that is still safe. The corollary lower-bounds *$V$-decreasing firings* and nothing more; fixing the length of a shortest witness needs the enabling structure, and for the meltdown it settles directly at $2N$ (Prop. 6, Appendix H). The theorem names a reusable mechanism: **the interface resets but the substrate does not.** Because every round looks identical at the interface, no interface-level monitor sees anything change while the system deteriorates monotonically, which is also why rate limiting the interface does not catch the failure: the loop is not anomalous in frequency, only in cumulative effect.

### 4.3 Screening is a linear program

Searching for such a cycle looks like a search over markings, which would be hopeless: reachability in nets is Ackermann-complete (Czerwiński and Orlikowski, 2021) and even coverability needs exponential space (Lipton, 1976; Rackoff, 1978). It is not such a search. Let $y \in \mathbb{N}^{|T|}$ count transition occurrences and let $C_I$ restrict the incidence matrix to interface places:

$$\underbrace{C_I y = 0}_{\text{interface regenerated}}, \qquad \underbrace{v^\top C y \leq -1}_{\text{resource dissipated}}, \qquad \underbrace{\textstyle\sum_{t\in T_i} y_t \geq 1,\ \sum_{t\in T_j} y_t \geq 1}_{\text{cross-agent}}, \qquad y \geq 0. \tag{4}$$

**Theorem 2** (Screening complexity)**.** *System (4) has a solution in $\mathbb{N}^{|T|}$ if and only if it has one in $\mathbb{Q}^{|T|}_{\geq 0}$. Consequently feasibility is decidable in time polynomial in the size of the composed net, by linear programming.*

Every constraint is preserved by scaling $y$ up, so a rational solution scaled by the least common multiple of its denominators is integral, and feasibility of the relaxation is a linear program (Appendix H). The linear algebra is classical: (4) is a T-semiflow condition on the interface submatrix with two extra constraints, and structural net analysis by linear programming is mature (Memmi and Roucairol, 1980; Silva et al., 1998; Murata, 1989). What is new is the reduction: that the hazard of

§4.2 is exactly expressible in that form, and that the interface restriction $C_I y = 0$ is what makes the cross-agent condition meaningful. Prop. 5 then closes the usual gap between the state equation and reachability, since whenever the returned support induces a simple circuit whose non-circuit inputs are marked and untouched, $y$ is guaranteed executable, as the meltdown's $\sigma_c = t_S t_C$ is with subnet places $\{U, K\}$ and $H$ marked.

### 4.4 Do local certificates survive composition?

Verification usually certifies each component with invariants, and composition breaks this when another component can move places appearing in one. Let $C_i = Post_i - Pre_i$, let agent $i$ have a local invariant $w_i^\top C_i = 0$, let $F_i$ embed local place vectors into the fused global space so that $C = [\, F_1 C_1 \; \cdots \; F_N C_N \,]$, and let $\widetilde{w}_i$ extend $w_i$ globally, with $\widetilde{W}_i$ stacking these for all of agent $i$'s invariants. Define the *invariant-interference block* $J_{ij} = \widetilde{w}_i^\top F_j C_j$: if $J_{ij} \neq 0$, transitions of agent $j$ can change a quantity agent $i$ treated as constant.

**Theorem 3** (Preservation of local linear invariants)**.** *If $\widetilde{W}_i^\top F_j C_j = 0$ for every $i \neq j$, then every extended local invariant is preserved by every transition of the composed net.*

**Proposition 4** (Preservation of monotone certificates)**.** *If $w_i^\top C_i \leq 0$ componentwise and $\widetilde{w}_i^\top F_j C_j \leq 0$ for every $j \neq i$, then $\widetilde{w}_i^\top M$ is nonincreasing along every firing sequence of the composed net.*

Both conditions are deliberately strong, because they are checkable without building the global reachability graph: one matrix product per agent pair. If global safety follows from the preserved invariants, this is a compositional certificate outright. Equality is not the only shape a component certificate takes, and the meltdown's is not: what the reliability agent relies on is that nothing replenishes its trigger, $e_U^\top C_S = (-1, -1) \leq 0$, so $M(U)$ is *nonincreasing* under its own firings, which is why Prop. 4 is needed beside Thm. 3. For an equality invariant any nonzero entry of $J_{ij}$ is a problem; for a monotone certificate only a *positive* entry is. In the meltdown each block has a positive entry and that is the failure: $J_{S,D} = (1, 0)$ and $J_{S,C} = (1)$, so $t_D$ and $t_C$ each put back the mark the reliability agent's argument assumes cannot return. AppWorld's `changed_model_names` assertions are equality invariants on the write witnesses, so there the test is $J_{ij} = 0$. Both conditions are sufficient, not necessary: failing one shows the local certificate is insufficient, not that the system is unsafe. Read as contracts, each agent was verified under an unwritten assumption about its input interface, latent in its test harness, and composition falsifies all three at once (Appendix I).

### 4.5 From possible to likely: how probable, and how fast

Give each timed transition a rate $\lambda_t(M) > 0$ with exponential firing time, so enabled transitions race and the marking process is a continuous-time Markov chain (Ajmone Marsan et al., 1995). Structure then decides whether the hazard probability can be nonzero at all (for every horizon and *every* positive rate assignment, which we call *distribution freedom*), while the rates decide only how fast. Without the thrash cycle,

$$\mathbb{E}[\tau_{\mathcal{B}}] = \frac{1}{\lambda_D} + \frac{N}{\lambda_S} + \frac{N-1}{\lambda_C} \qquad \text{and, with a stabilizer of rate } \mu, \qquad \mathbb{P}(\text{outage}) = \left(\frac{\lambda_S}{\lambda_S + \mu}\right)^N. \tag{5}$$

The left formula carries an uncomfortable reading: *faster agents do not make this system safer, they make it fail sooner.* Halving mean time to mitigate, the headline metric for a reliability agent, halves the $N/\lambda_S$ term without changing whether the hazard is reached at all. The thrash cycle competes for the same CPU mark, and Theorem 4 (Appendix H) shows it costs time and not risk: a cross-agent cycle that is interface-regenerative and 0-dissipative leaves the probability of advancing toward the hazard at $\lambda_S/(\lambda_S + \mu)$, free of its own entry rate $\lambda_0$ and of every rate inside it, while inflating the expected time to leave the state from $1/(\lambda_S + \mu)$ to $(1 + \lambda_0\theta)/(\lambda_S + \mu)$. The meltdown cycle therefore determines *whether* the system fails and the thrash cycle only *when*, which is why an operator sees an outage on some days and a release that will not stick on others, with no code change in between. The design consequence follows: removing a cross-agent cycle makes the outage unreachable under every parameterization, whereas a cooldown only makes it unlikely under the rates one happened to estimate, and by (5) that protection weakens as $N$ falls, exactly when the platform is already degraded.

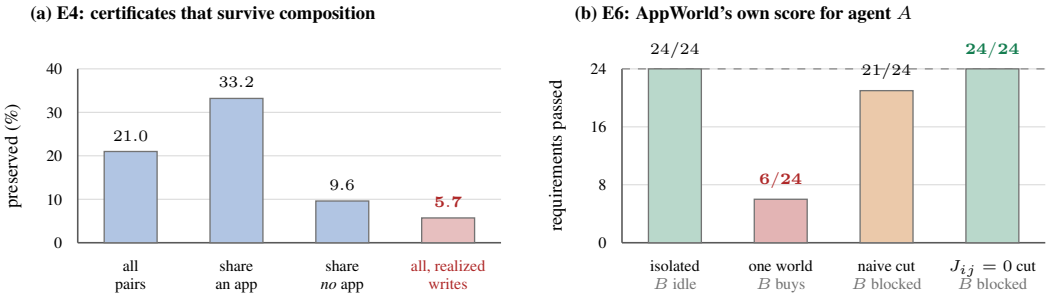



Figure 2: The two empirical results. **(a)** The fraction of AppWorld task pairs whose own evaluator certificate survives composition. Pairs sharing *no* application survive least often, because a certificate reading *and nothing else changed* is falsified by a write anywhere; the red bar recomputes the census against what the realizing APIs actually write. **(b)** AppWorld's own evaluator scoring agent $A$, whose task is unrelated to $B$'s: sharing one account costs $A$ three quarters of its requirements, the intuitive repair does not restore them, and the cut Thm. 3 prescribes does.

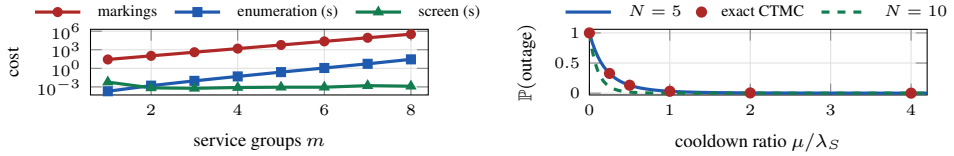



Figure 3: The screen is practical, and the stochastic theory is exact. **Left** (E2): as $m$ service groups share one pool, enumerating the reachable set grows geometrically while the linear program of Thm. 2 stays flat at a few milliseconds. **Right** (E3): the closed form of (5) for two fleet sizes against exact Markov-chain values on the full net *including* the thrash cycle, which is Thm. 4 in both halves at once.

## 5 Experiments

Two settings. On the nets of §3, where every quantity is exactly computable, we ask whether the theory is right (E1 to E3); on two artifacts built by other people for other purposes, AppWorld (Trivedi et al., 2024) and MAST (Cemri et al., 2025), we ask whether it survives contact with systems we did not design (E4 to E7). Table M7 is the map and Appendices M and N the full narratives, protocols and numbers. For E4, E5 and E7 we executed nothing, analyzing AppWorld's evaluators, 4,601 released trajectories with the database diffs they produced, and MAST's annotated traces; for E6 we ran two scripted task executors on one live AppWorld account, scored by AppWorld's own evaluator. No language model was invoked in any of the four, which holds model competence out as a confound.

**E1 to E3: the theory is right, and cheap to apply.** On the nets of §3 every claim is confirmed exactly: no component reaches $\mathcal{B}$, and the composition reaches it for every $N$ from 2 to 12 with a shortest witness of exactly $2N$, the length Prop. 6 predicts. Figure 3 shows the other two claims, the screen against enumeration as a fleet grows and the closed form against the exact chain; Appendix M gives all three in full.

**E4: a benchmark's own certificates, composed.** AppWorld scores 732 tasks one at a time, each on a private copy of a world of nine apps and 469 APIs giving one user an inbox, a file system and four accounts that move money. The situation any real assistant is in is the other one: the tasks are separate and the accounts are not, so we take each task as an agent and ask what Thm. 3 says about pairs of them on one world. The certificate is not ours: for 486 tasks the evaluator asserts `changed_model_names == {...}`, which says these models changed and, as importantly, that no others did. As an invariant it is $w_i^\top M = 0$ with $w_i$ the indicator of the write-witness places of every model outside the declared set, so $J_{ij} \neq 0$ exactly when $j$ writes something $i$'s certificate holds constant. Of 235,710 ordered pairs only 21.0% survive (Figure 2a, Table M8), and the breakdown is the surprising part: pairs whose tasks share *no* application break most reliably of all, because a

certificate saying *and nothing else changed* is falsified by a write anywhere. AppWorld's authors did not write bad assertions: they wrote the right assertion for one agent, and the composition has none to inherit. The rate is close to arithmetic once the mechanism is seen, and what it buys is a measurement of how strongly the benchmark assumes exclusive access to its environment.

Theorem 1 also needs a nonnegative $V$ and a cross-agent cycle that regenerates the interface while spending it, and AppWorld supplies both without invention on our part (Figure M7, Appendix M). One agent calls `add_product_to_cart` and another calls `place_order`: the cart is empty again and the money is gone. Alone, the first has no $V$-decreasing transition at all and the second has one that is never enabled; together they reach $\mathcal{B}$ in exactly $2K$ firings for every budget $K$ from 2 to 12, the $K$ traversals Thm. 1 predicts (Table L2), and the same shape appears on `splitwise.ExpenseShare`. The negative control makes this a measurement rather than an illustration: two agents that both spend from one Venmo balance also exhaust it, and the analysis declines to call that composition-induced, because each exhausts it alone and the screen returns an empty interface. By Definition 2 that is contention, not emergence, and the screen's cost stays flat as shoppers are added (Table L3).

**E5: the arcs the API surface omits.** A net library is a claim about an API's semantics, and AppWorld lets that claim be tested, since it publishes trajectories together with the SQL those calls applied. Held out over the three architectures not used to correct it, the library names the exact set of changed models $63.1\%$ of the time with recall $0.93$; recall is the important direction, because over-approximating what an agent writes still bounds the interference while missing a write does not. Building the library that way turned up the sharpest thing this benchmark has to say. **Twenty calls write models belonging to a different app, and not one says so in its own API documentation**, nine of them documented `GET` calls that write a file, so any analysis classifying calls by HTTP method puts them on the wrong side of the interface. The behavior is not secret (AppWorld's paper describes the worlds as coupled), but the coupling is absent from the per-API contract an implementer reads, which is the artifact any per-agent analysis is built from. Recomputing the census against what those APIs actually write drops the preserved fraction from $21.0\%$ to $5.7\%$ (Figure 2a): three quarters of the pairs that looked compositionally safe are not, and the coupling that breaks them is visible only in the state (Appendix M).

**E6: two agents, one account, and the repair.** Everything above is structural. Here two agents run live on one AppWorld account and AppWorld's own evaluator scores the first (Figure 2b). Agent $A$ exports the user's notes to a backup directory: task `6c2c621`, whose evaluator declares that a correct run changes `file_system.File` and `file_system.Directory` and nothing else. Agent $B$ buys something. The two tasks have nothing to do with each other. Alone, $A$ passes all three task variants; on one world with $B$ it fails all three, because the order's receipt is a file, it is not in the backup directory, it does not end in `.md`, and the models changed are no longer the two declared. Nothing in $B$'s behavior is wrong and nothing in $A$'s changed.

Theorem 3 identifies which fusions must be removed to restore $A$'s certificate (Table M9 and Figure L6, Appendices M and L). The obvious screen, stopping $B$ from writing the models $A$ owns, still fails all three tasks, because $B$'s cart entry is a write outside $A$'s declared set even though it is nothing $A$ cares about; the screen Thm. 3 prescribes, which refuses every call of $B$ whose effects leave $A$'s declared set, restores all three at a cost of six refused calls. That cost should be read before the result is called a repair, since $B$ manages its own purchase only in the unmediated arm: the screen restores $A$ under a priority rule rather than both agents at once, and since it only refuses calls it guarantees no new reachability hazard while saying nothing about $B$'s progress.

**E6b (specified, not run) and E7.** The complementary experiment replaces both executors with language-model agents and measures a rate rather than a mechanism; its prediction is falsifiable, that degradation in the shared arm is confined, up to the model's own error rate, to the $J_{ij} \neq 0$ pairs. As an outside check on the premise itself, MAST (Cemri et al., 2025) annotates $1{,}642$ multi-agent traces against fourteen failure modes, and $174$ of the $1{,}237$ carrying a label ($14.1\%$) carry an inter-agent mode and no specification-category mode at all: no local fault was annotated and the system failed anyway. An absent label is not a verified absence, which is why that is the strongest reading we claim, and the corpus does *not* support a quantitative link between interface failure and the topology of $G_A$, a negative result we report as such (Appendix M).

**Threats to validity.** E1 to E3 are exact analyses of formal models, so they support one claim only, that the theory is correct and computable. E4 is exact *given the library*, which E5 measures rather than asserts; E6 is three task variants under a scripted policy; and E7 rests on an LLM judge we show to be unreliable at the granularity its finer claim needs.

## 6 Related work

**Compositional verification, and net theory.** That local correctness does not compose founds assume-guarantee reasoning (Misra and Chandy, 1981; Pnueli, 1985; Abadi and Lamport, 1993), compositional model checking (Clarke et al., 1989) and interface theories (de Alfaro and Henzinger, 2001; Alur and Henzinger, 1999), whose response is to declare each component's environmental assumption and discharge it against the others. We differ in three ways that matter for agents: the assumption we falsify is declared nowhere, being latent in a test harness, so §4.4 recovers it from the component's own certificate; the failure is a resource failure, so the operator is shared-state fusion rather than synchronization on channels; and the certificates already exist in numbers, §5 composing 486 written by other people. On the net side, composition and structural analysis are classical (Petri, 1962; Murata, 1989; Winskel, 1987; David and Alla, 1994; Jensen and Kristensen, 2009; Ajmone Marsan et al., 1995; Molloy, 1982), our screen (4) is a T-semiflow condition on an interface submatrix plus two constraints, analysis by linear programming over semiflows is mature (Memmi and Roucairol, 1980; Silva et al., 1998), and the hardness of behavioral questions is standard (Karp and Miller, 1969; Lipton, 1976; Rackoff, 1978; Czerwiński and Orlikowski, 2021). The contribution is the reduction, not the linear algebra: that this hazard class is exactly a T-semiflow with a dissipation constraint, and that Prop. 5 makes the relaxation exact on the simple circuits it returns.

**Multi-agent LLM systems, and their evaluation.** Frameworks running several model-driven agents against shared state are standard (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Li et al., 2023), and empirical studies catalog how they fail (Cemri et al., 2025; Reid et al., 2025; Erisken et al., 2025); those studies supply phenomena, and we use one (Cemri et al., 2025) as an outside check on our premise rather than a source of examples. Where a taxonomy gives a label, we add a mechanism with a bound and a test. Work on agent oversight and sandboxing (Ruan et al., 2024; Chan et al., 2024) builds the mediation layer our screens presuppose, and Prop. 1 is the argument that it cannot be purely per-action. Agentic benchmarks (Trivedi et al., 2024; Zhou et al., 2024; Jimenez et al., 2024; Yao et al., 2024; Liu et al., 2024) score one agent at a time, which is exactly why their per-task correctness assertions can be read as local certificates and composed against each other; to our knowledge that reading is new, and it is what lets a composition claim be tested against a scorer nobody wrote for it.

Finally, "emergence" in machine learning usually names a capability appearing with scale (Wei et al., 2022), a framing whose metric dependence has been contested (Schaeffer et al., 2023); Definition 2 is a deliberately modest alternative, checkable and baseline-relative rather than a claim about unpredictability. And the view that accidents arise from unsafe interactions among components each of which meets its requirements is the core of systems-theoretic safety analysis (Leveson, 2011), to which we contribute a formalism where the interaction is a fused place, the hazard is a resource bound, and the analysis is a linear program rather than a guided workshop.

## 7 Discussion

The results compose into a layered analysis that avoids the global state space unless necessary: expose the interface places, build $G_A$, compute one block $J_{ij}$ per agent pair, run the screen (4), and build a reachable set only for what survives. Appendix M expands it, with three repairs for the meltdown, the limitations and four open problems. The E4 census is the part others can rerun today, needing no agent, no model and no execution: for any benchmark asserting an exact state-change footprint, compute $J_{ij}$ over ordered task pairs and report the preserved fraction. The agents of §3 are not hypothetical, and Prop. 1 says their guardrails do not add up: an agent should be approved with an interface contract naming what it emits, what it consumes, and the companions for which its certificate holds. Until then, a score earned alone is a claim about a world the agent will not be deployed into.

## AI USE STATEMENT

We used a large language model as a writing and editing assistant: condensing the exposition, drafting prose that we then revised, literature search for the related-work section, and LaTeX and figure layout. Every claim, number, citation and proof was checked by the authors against the underlying artifacts. Of the activities ICLR asks authors to disclose, generative AI was not used to develop the theory, to formulate or prove the mathematical claims, to propose hypotheses, to design the experiments, to implement the analysis code, or to generate or clean data. We take responsibility for the final content.

## REPRODUCIBILITY STATEMENT

Every quantity in §5 other than timings is exact rather than sampled. Appendix F gives the incidence matrices of the running example, Appendix H the assumptions and full proofs, Appendices N and L the protocols and complete tables, and Appendix L the construction of the AppWorld net library and its validation against the recorded database writes. E6 is scored by AppWorld's own evaluator rather than ours, no language model is invoked anywhere in the analysis, and the harness, nets and result files accompany the paper.

## ETHICS STATEMENT

This work analyzes publicly released benchmark artifacts and a stylized model, and involves no human subjects, no personal data and no new model. One risk is worth naming: a screen that locates the couplings carrying a hazard also locates where a shared substrate is most fragile. We judge the balance favorable, since those couplings are recoverable by anyone who reads an API surface against its database, and the mitigations the analysis prescribes are available to defenders at the same cost.

## REFERENCES

M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73-132, 1993.

M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.

R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7-48, 1999.

Amazon Web Services. Production operations and autonomous incident response, AWS DevOps Agent. AWS documentation, 2026. https://docs.aws.amazon.com/devopsagent/latest/userguide/production-operations-autonomous-incident-response.html.

Amazon Web Services. AWS Security Incident Response now provides agentic AI-powered investigation, November 2025. https://aws.amazon.com/about-aws/whats-new/2025/11/aws-security-incident-response-agentic-ai-powered-investigation.

M. Cemri et al. Why do multi-agent LLM systems fail? arXiv:2503.13657, 2025.

A. Chan et al. Visibility into AI agents. In *Proc. ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2024.

E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Proc. IEEE Symposium on Logic in Computer Science (LICS)*, 1989.

W. Czerwiński and Ł. Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 2021.

R. David and H. Alla. Petri nets for modeling of dynamic systems: a survey. *Automatica*, 30(2):175-202, 1994.

L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 2001.

S. Erisken, T. Gothard, M. Leitgab, and R. Potham. MAEBE: multi-agent emergent behavior framework. arXiv:2506.03053, 2025.

Google. AI engineering for reliable operations. Google SRE practices and processes, 2026. https://sre.google/resources/practices-and-processes/ai-engineering-reliable-operations/.

S. Hong et al. MetaGPT: meta programming for a multi-agent collaborative framework. In *Proc. ICLR*, 2024.

K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: can language models resolve real-world GitHub issues? In *Proc. ICLR*, 2024.

R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147-195, 1969.

N. G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011.

T. Li and Q. Zhu. Agentic AI for cyber resilience: a new security paradigm and its system-theoretic foundations. arXiv:2512.22883, 2025.

G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem. CAMEL: communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, 2023.

R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, 1976.

X. Liu et al. AgentBench: evaluating LLMs as agents. In *Proc. ICLR*, 2024.

G. Memmi and G. Roucairol. Linear algebra in net theory. In *Net Theory and Applications*, LNCS 84, pages 213-223. Springer, 1980.

J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417-426, 1981.

M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Trans. Computers*, 31(9):913-917, 1982.

T. Murata. Petri nets: properties, analysis and applications. *Proc. IEEE*, 77(4):541-580, 1989.

PagerDuty. Operations Cloud release including agentic AI delivering autonomous site-reliability operational insights and scheduling optimization agents. Press release, February 2025.

C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.

A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, pages 123-144. Springer, 1985.

C. Qian et al. ChatDev: communicative agents for software development. In *Proc. ACL*, 2024.

C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223-231, 1978.

A. Reid, S. O'Callaghan, L. Carroll, and T. Caetano. Risk analysis techniques for governed LLM-based multi-agent systems. arXiv:2508.05687, 2025.

Y. Ruan et al. Identifying the risks of LM agents with an LM-emulated sandbox. In *Proc. ICLR*, 2024.

R. Schaeffer, B. Miranda, and S. Koyejo. Are emergent abilities of large language models a mirage? In *Advances in Neural Information Processing Systems*, 2023.

M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *Lectures on Petri Nets I: Basic Models*, LNCS 1491, pages 309-373. Springer, 1998.

H. Trivedi, T. Khot, M. Hartmann, R. Manku, V. Dong, E. Li, S. Gupta, A. Sabharwal, and N. Balasubramanian. AppWorld: a controllable world of apps and people for benchmarking interactive coding agents. In *Proc. ACL*, 2024. Best Resource Paper.

J. Wei et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

G. Winskel. Petri nets, algebras, morphisms, and compositionality. *Information and Computation*, 72(3):197-238, 1987.

Q. Wu et al. AutoGen: enabling next-gen LLM applications via multi-agent conversation. arXiv:2308.08155, 2023.

Y.-T. Yang and Q. Zhu. Internet of agentic AI: incentive-compatible distributed teaming and workflow. arXiv:2602.03145, 2026.

S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. $\tau$-bench: a benchmark for tool-agent-user interaction in real-world domains. arXiv:2406.12045, 2024.

S. Zhou et al. WebArena: a realistic web environment for building autonomous agents. In *Proc. ICLR*, 2024.

## How to read this appendix

It is long because it serves three different readers, and none of them needs all of it. A reader new to Petri nets should start at Appendix B, which assumes nothing, and continue to Appendix C only if they want the probability in §4.5; the structural half of the paper needs none of it. A reader who wants to know whether this applies to *their* agents should read Appendix D, which turns three ordinary orchestration configs into one composed net and shows why the coupling appears in none of them, and then Appendix E, which works four further compositions, each chosen to exercise a different result. A reader who wants the main text at full length should read Appendix M, which holds every passage §5 and §7 compress, together with the experiment overview table, the E4 and E6 tables, the AppWorld composition figure, the screening procedure and the open problems. A reader checking the claims should read Appendix H for the proofs, Appendix F for the running example's incidence matrices and the witness marking by marking, Appendix L for how the benchmark nets are built and what they do and do not model, and Appendix N for every protocol and full result behind §5. Appendix A argues the choice of formalism against the alternatives, Appendix G is the symbol table, and Appendices I, J and K give the contract reading, the dynamic case, and a hierarchy of emergence.

**Disclosure: shared expository material.** The background appendices, namely Appendix B, Appendix C and the correspondence table of Appendix D, are tutorial, and none of their content is claimed as a contribution of this paper. Substantially the same exposition appears in companion work by the same authors, and is reproduced here so that this paper can be read on its own rather than alongside it. A reader of more than one should skip it. This paper's own contributions are the results listed in §1 and proved in Appendix H, together with its experiments.

## Notation and assumptions, restated

The appendices are read out of order, so everything they invoke is collected here. Nothing in this section is new.

**Nets.** A place/transition net is $\mathcal{N} = (P, T, Pre, Post, M^0)$ with $Pre, Post \in \mathbb{N}^{|P|\times|T|}$ the arc multiplicity matrices and $C = Post - Pre$ the *incidence matrix*. A marking $M \in \mathbb{N}^{|P|}$ is the whole state. Transition $t$ is enabled at $M$ when $M \geq Pre(:, t)$ componentwise, and firing gives $M' = M + C(:, t)$. We write $t : X \to Y$ for the transition with $Pre(:, t) = X$ and $Post(:, t) = Y$ as multisets over places, so $t_S : U + H \to Q + K$ means $Pre(U, t_S) = Pre(H, t_S) = 1$ and $Post(Q, t_S) = Post(K, t_S) = 1$ with every other entry zero. $\mathcal{R}(\mathcal{N}, M_0)$ is the reachable set and $\mathcal{B}$ the hazardous markings. Petri-net texts say *token* for a mark; since that word already denotes what a language model consumes, this paper says **mark** throughout.

**Composition.** Each component declares *interface* places $I_i \subseteq P_i$, those it reads from or writes to a shared environment; the rest are internal. A *fusion map* $\Phi$ is an equivalence relation on $\bigsqcup_i P_i$ whose classes are either a single internal place or a set of interface places taking at most one from each component, and $\|_\Phi \mathcal{N}_i$ has places $(\bigsqcup_i P_i)/\Phi$, all the transitions, and $Pre_\Phi(\gamma, t) = \sum_{p\in\gamma} Pre_i(p, t)$, likewise for $Post$ (Definition 1). $\Phi$ the identity is the uncoupled case. The baseline is $L_\otimes = \mathrm{Shuffle}(L_1, \ldots, L_N)$ against the composed language $L_\|$, and $\sigma$ is *composition-emergent* when $\sigma \in L_\| \setminus L_\otimes$ (Definition 2). Emergence is not hazard: a *composition-induced safety violation* is the separate condition that each component meets its declared local safety condition while $\mathcal{R}(\|_\Phi \mathcal{N}_i, M_0) \cap \mathcal{B} \neq \varnothing$.

**The mechanism.** Fix interface places $I$ with projection $\pi_I(M)$ and a nonnegative resource functional $V(M) = v^\top M$. A sequence $\sigma$ with $M \xrightarrow{\sigma} M'$ is *interface-regenerative* at $M$ if $\pi_I(M') = \pi_I(M)$, and *$d$-dissipative* if $V(M') \leq V(M) - d$ for some $d > 0$. Using transitions of at least two agents makes it a **regenerative dissipative interaction cycle**. In one sentence: *the interface resets but the substrate does not*, which is why no interface-level monitor sees anything change while the system deteriorates.

**The results.** Theorem 1: such a cycle reaches $V(M) < V_{\min}$ after at most $K = \min\{k : V(M^\star) - kd < V_{\min}\}$ repetitions. Corollary 1: the bound is attained; the length of a shortest witness

is settled per net, as Proposition 6 does for the meltdown. Theorem 2: the screen (4) is preserved by scaling up, so integer feasibility equals rational feasibility and the search is a linear program rather than a search over markings; Proposition 5 says when its answer is guaranteed executable. Theorem 3: local linear invariants all survive composition if $\widetilde{W}_i^\top F_j C_j = 0$ for every $i \neq j$, checkable as one matrix product per agent pair. Proposition 3: emergence is monotone in the coupling when fused places start empty, so cutting such an interface can only remove behavior and never introduce a hazard. Theorem 4: a cross-agent cycle that consumes nothing strictly increases the expected time to failure and leaves its probability exactly unchanged.

**The stochastic layer.** Each enabled transition carries a rate $\lambda_t(M) > 0$, making the marking process a continuous-time Markov chain; *admissible* means every rate is finite and strictly positive. Structural statements quantify over all admissible assignments, which is why they need no rate estimates. Appendix C develops the chain, absorption probabilities, expected hitting times and the embedded jump chain from scratch.

**The scope assumption.** Throughout, each agent's own execution is assumed already mediated and behaving as specified: no hallucination, no prompt injection, no adversary. The failures studied here are what remains *after* that assumption holds, which is precisely why component-level assurance cannot see them.

**The experiments.** E1 shows no component reaches $\mathcal{B}$ and the composition reaches it in exactly $2N$ firings. E2 compares the screen against enumeration as a fleet grows. E3 checks the closed form and the neutrality of the thrash cycle against an exact chain. E4 to E7 take the analysis to two published benchmarks. E1, E2, E3 and the structural parts of E4 to E7 are exact; anything sampled states its sample size in Appendix N.

## A WHY A PETRI NET, AND NOT SOMETHING ELSE

The meltdown of §3 has three properties at once, and the formalism has to hold all three simultaneously or it cannot see the failure. It is a *resource* failure: no protocol is violated, the system merely runs out of instances. It is a *composition* failure: it exists only in the coupling, and §3 shows each component provably cannot produce it alone. And it is a failure that is *possible* before it is *likely*: we need to say which sequences can occur before we say how often, because the two questions have different answers for the two cycles in the same net.

| Demand | Why this failure has it | Petri-net construct |
|---|---|---|
| **Counted resources** | the hazard is $M(H) = 0$, reached one instance at a time | a place is a counter; $v^\top M$ is the resource functional of Thm. 1 |
| **Composition as shared substrate** | agents meet by reading and writing shared signals, not by exchanging typed messages | place fusion; Prop. 3 makes the coupling a monotone knob |
| **Structural analysis without enumeration** | 327,681 markings at $m = 8$, and real fleets are larger | the incidence matrix $C$; screening becomes an LP (Thm. 2) |
| **Component certificates as first-class objects** | practitioners already write invariants; the question is whether they survive | $w^\top C_i = 0$; interference is one matrix product (Thm. 3) |
| **Possible before likely** | one cycle is lethal, one is merely wasteful, and they look identical | one structure, two layers; Thm. 4 separates them |

### THE ALTERNATIVES, ON THE AUTONOMOUS CLOUD

**Per-agent testing and benchmarks.** The actual incumbent, and the reason this paper exists. Each agent passes its own suite, and the caption of Figure 1(a) says why that is not carelessness: in isolation the reliability agent's trigger $U$ has no producer, so testing it at all requires injecting an anomaly by hand, after which it correctly quarantines once and stops. The test harness cannot generate the second anomaly because the component that generates it is a different team's agent. Per-agent testing without an explicit interface contract does not close this gap.

**Integration testing and multi-agent evaluation.** The obvious next answer, and it finds this failure only if it happens to sample the right interleaving. Our composed net is small, yet the hazard needs a specific alternation of $t_S$ and $t_C$ sustained for $N$ rounds; an evaluation that stops after a few steps observes one quarantine and a successful reroute, which is exactly the intended behavior. Exact reachability answers the same question with certainty on $4N + 3$ markings, and Theorem 2 answers it without enumerating anything at all.

**Automata and products of state machines.** The state explosion is the entire difficulty at fleet scale, and Table N12 measures it: a factor of about 3.9 per service group. Worse, an automaton product gives no analogue of the state equation, so there is nothing to relax into a linear program. The structural screen is not an optimization of enumeration; it is a different object that automata do not have.

**Process algebra.** CCS, CSP, and the $\pi$-calculus make parallel composition native, which is a real strength and the closest competitor on that axis (Winskel, 1987). Resources are the problem: "one healthy instance is consumed" has to be encoded as process behavior, so a depletion argument becomes a bisimulation argument instead of arithmetic on $C$, and the linear-algebraic screen has no counterpart. Nets and process algebras are formally related, and the relation is exactly why we take the side with an incidence matrix.

**Assume-guarantee reasoning.** Not a competitor but the right frame, and §4.4 is an assume-guarantee result: $J_{ij} = 0$ for all $i \neq j$ is the interface condition under which local certificates compose. What assume-guarantee needs and does not itself supply is a model with explicit shared state and counted resources in which the obligations can be discharged mechanically. The net supplies it, and reduces the discharge to one matrix product per agent pair rather than a circular-reasoning rule plus model checking.

**Orchestration graphs and workflow DAGs.** These model one planner's plan. The autonomous cloud has no planner: three agents built by three teams act on shared infrastructure with no common graph, no shared scheduler, and no coordinating node. There is simply no DAG to write, and writing one would presume away the property under study.

**Control theory.** The nearest intuition, since the meltdown is a positive feedback loop, and a continuous model of CPU and capacity would give an instability argument. Three mismatches. The hazard is discrete and combinatorial: instances are quarantined one at a time and $M(H)$ is an integer that reaches zero. The agents are event-triggered, not continuously acting. And the artifact an incident review needs is a witness sequence naming the transitions in order, which reachability produces and a stability margin does not. The stochastic layer of §4.5 recovers the quantitative half without giving up the first.

**Chaos engineering and fault injection.** Complementary and empirical, and aimed at a different target. It perturbs a system to see whether it tolerates faults. Here there is no fault to inject: every component behaves exactly as designed, on inputs that are exactly what its specification anticipated. The pathology is in the composition, not in any deviation from it.

**Guardrails and mediated execution.** Mediating a single agent's consequential actions is a different problem from this one, and the Scope paragraph of §1 states the relation: we assume every agent's execution is already mediated, with no hallucination, no injection, and no adversary. This failure is what remains *after* per-agent assurance succeeds, which is precisely why per-agent assurance cannot be the answer to it.

### What a Petri net does not give

A net is only as good as its event abstraction. If "CPU is elevated" is not faithfully a mark, the analysis is about a different system, and the abstraction step is where domain expertise enters and cannot be automated away. Rates are estimated, usually from benign traffic, and the quantitative claims of §4.5 inherit that uncertainty; the structural claims do not, which is the practical reason to prefer them. Continuous dynamics, gradual degradation, and load-dependent service curves are

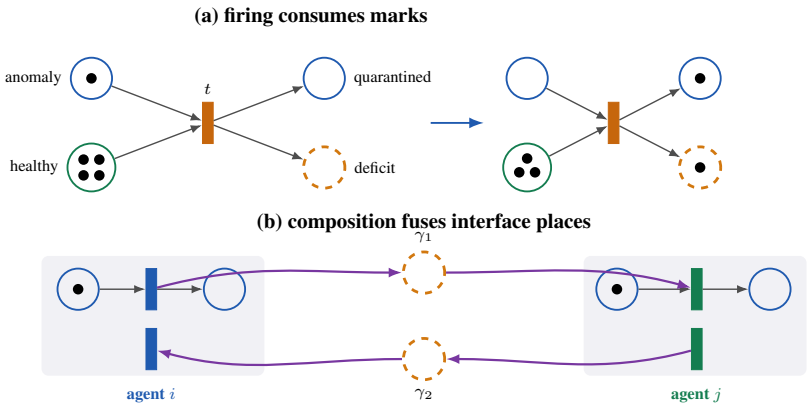



Figure B1: The formalism (a) and the composition operator (b). In (a) the healthy count has decreased after firing: marks are consumed, not copied. In (b) we compose by *fusing interface places* (Definition 1): a place one component writes is identified with a place another reads, and the merged place is written $\gamma$. The two fused places $\gamma_1, \gamma_2$ shown here are generic, standing for any such pair; in the running example of §3 they are $U$, $K$, $V$ and $B$. Two are already enough for agent $j$ to change what agent $i$ can do next, while nothing inside either component changes. A single coupling direction only adds behavior downstream, whereas the reciprocal coupling drawn here closes a directed cycle, and a cycle lets influence circulate indefinitely. Other composition semantics exist, and Section 2 explains which of them the results cover.

outside the model. And nothing here decides which of several safe designs is best; it decides which are safe.

## B PETRI NETS FROM SCRATCH

This appendix assumes no prior exposure to Petri nets. A reader who has seen them can skip to Appendix F.

**The three objects.** A net has *places* (circles), *transitions* (bars), and *arcs* between them, always place-to-transition or transition-to-place, never place-to-place. A place holds some number of *marks* (dots). The list of how many marks sit in each place is the *marking* $M$, and the marking is the entire state of the system: there is no other state variable hiding anywhere.

Read a place as a condition or a countable resource, and a transition as an event. In our running example the place $H$ holds one mark per healthy serving instance, so $M(H) = 8$ means eight instances are healthy; the place $U$ holds a mark when CPU pressure has been measured as elevated; the transition $t_S$ is the event "the reliability agent quarantines an instance".

**The one rule.** A transition is *enabled* when every place with an arc into it holds at least as many marks as that arc requires. An enabled transition may *fire*, which *removes* the required marks from its input places and *adds* marks to its output places. Two consequences do all the work in this paper.

*A transition needs all of its inputs.* $t_S$ has arcs from both $U$ and $H$, so quarantining requires both an anomaly signal and a healthy instance to take. This is conjunction, and it is structural: there is no way to write a net in which $t_S$ fires with $U$ empty.

*Firing spends the inputs.* After $t_S$ fires, the mark that was in $H$ is gone. It is not flagged as spent; it does not exist. This is what makes a net able to express depletion, and it is why "eight instances, quarantine one per anomaly" has a last round rather than continuing forever.

**Worked micro-example.** Figure B2 is this paragraph in three pictures. Take $M(U) = 1$, $M(H) = 8$, everything else 0. Then $t_S : U + H \to Q + K$ is enabled, since $U$ and $H$ both hold a mark. Firing it gives $M(U) = 0$, $M(H) = 7$, $M(Q) = 1$, $M(K) = 1$. Now $t_S$ is *not* enabled, because $U$ is

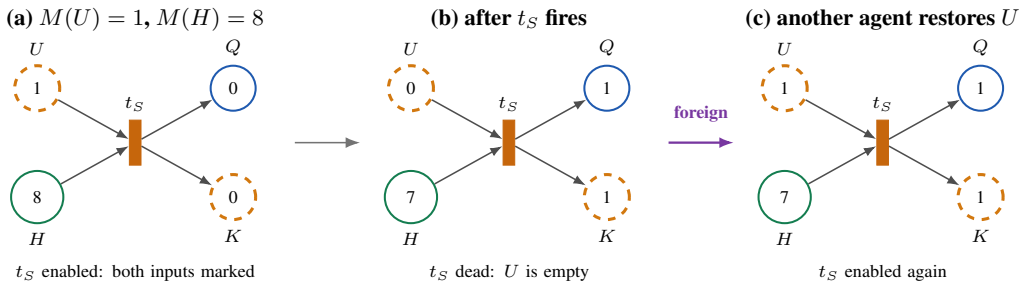



Figure B2: The one firing rule, and the entire paper in three pictures. In (a) the reliability agent's quarantine transition is enabled because both of its inputs hold a mark. Firing it (b) *spends* them: $H$ drops by one and $U$ is empty, so $t_S$ cannot fire again. One anomaly bought exactly one quarantine, which is the agent's local guarantee and it is structural: no schedule, no rate limit and no adversary is involved. In (c) some *other* component puts a mark back into $U$, which costs $H$ nothing: the marking is $M(H) = 7$ still, and $t_S$ is merely enabled again. Nothing inside the reliability agent has changed and its guarantee is still true of it, yet the next firing takes $M(H)$ to 6. Every result in this paper is about the violet arrow: which compositions can draw it, how far $H$ can fall before something breaks, and how to find out without enumerating the states.

empty: one anomaly bought exactly one quarantine. This is the isolated reliability agent, and it is why its local guarantee holds. Now suppose some other component puts a mark back into $U$. Then $t_S$ is enabled again, $M(H)$ drops to 6, and nothing about the reliability agent has changed. That sentence is the entire paper.

**Why matrices.** Number the places and the transitions. $Pre(p, t)$ is how many marks $t$ takes from $p$; $Post(p, t)$ is how many it puts back. Then $t$ is enabled at $M$ exactly when $M \geq Pre(:, t)$ componentwise, and firing gives $M' = M + C(:, t)$ for $C = Post - Pre$, the *incidence matrix*. Firing a whole sequence $\sigma$ gives $M' = M + Cy$, where $y$ counts how many times each transition occurs in $\sigma$; $y$ is the *Parikh vector* and it forgets the order. This is the *state equation*, and it is why questions about nets become linear algebra, which is what Theorem 2 exploits.

The state equation is a one-way implication: every reachable marking satisfies it, but a $y \geq 0$ satisfying it need not correspond to any executable order of firings. That gap is the reason Proposition 5 exists.

**Invariants, and why practitioners already write them.** A vector $w$ with $w^\top C = 0$ is a *place invariant*: since $M' = M + Cy$, we get $w^\top M' = w^\top M$ for every reachable $M$, so the weighted mark count $w^\top M$ never changes. Take $w = e_H + e_Q$ in our example: every quarantine moves one mark from $H$ to $Q$, so $M(H) + M(Q) = N$ always. That is a machine-checked conservation law, obtained from the matrix without exploring any states, and it is exactly the shape of the safety certificates engineers write by hand ("instances are never created or destroyed, only moved").

**The five questions one asks of a net.**

| Question | Formally | Operationally |
|---|---|---|
| Reachability | is $M \in \mathcal{R}(\mathcal{N}, M_0)$ | can this bad configuration ever occur |
| Boundedness | is $M(p) \leq k$ always | can this queue or spend grow without limit |
| Invariants | solve $w^\top C = 0$ | what is conserved, and what certifies it |
| Liveness | can every $t$ still fire | is any capability permanently dead |
| Deadlock | is some reachable $M$ with no enabled $t$ | can the system wedge |

**Two things a net is not.** It is not a flowchart: there is no single "current step", because many places can hold marks at once and that is how concurrency is represented. And it is not a probabilistic model on its own: the net says which sequences are *possible*, and only the stochastic layer of §4.5 says which are *likely*. Keeping those separate is the point of Theorem 4.

## C THE STOCHASTIC LAYER, FROM SCRATCH

The structural half of this paper needs no probability: a marking is reachable or it is not. The quantitative half needs a little, and this appendix supplies all of it. It assumes only that the reader has met the exponential distribution, and a reader who has not can take the two facts in the first paragraph on trust and lose nothing. Everything the paper computes is one of the four objects defined below, and all four are linear algebra.

**Why a rate, and what it costs.** Give each enabled transition $t$ a *rate* $\lambda_t > 0$, meaning it fires after a delay drawn from $\mathrm{Exp}(\lambda_t)$, whose mean is $1/\lambda_t$. Two properties are the reason this particular distribution is standard here rather than a modeling opinion. It is *memoryless*: having waited a while tells you nothing about how much longer you will wait, so the state of the process is the marking alone and no clock has to be carried in it. And it is closed under the race described next. The cost of the assumption is real and worth naming: exponential delays have their mode at zero and a long tail, so they are a poor description of a step whose duration is tightly concentrated, such as a fixed timeout. Where that matters one splits the step into several exponential stages, which is the standard repair; where a paper's claim quantifies over *all* rate assignments, as the structural results here do, the question does not arise at all.

**The race, which is where branching comes from.** Suppose transitions $t_1, \dots, t_k$ are enabled at a marking with rates $\lambda_1, \dots, \lambda_k$. They run concurrently and the first to fire wins. Two facts, both elementary:

$$\min_i \mathrm{Exp}(\lambda_i) \;\sim\; \mathrm{Exp}\Big(\textstyle\sum_i \lambda_i\Big), \qquad\qquad \mathbb{P}\{t_j \text{ fires first}\} = \tfrac{\lambda_j}{\sum_i \lambda_i}.$$

The first says how long the system waits; the second says where it goes. Nothing else generates the branching probabilities in this paper: wherever a number like "$60\%$ of the time" appears, it is a ratio of rates in a race, or a rate that was split into two branches to represent an outcome distribution.

**The chain and its generator.** Let $\mathcal{S}$ be the set of reachable markings, assumed finite. The construction above makes the marking process a *continuous-time Markov chain* (CTMC) on $\mathcal{S}$: a process that sits in a state for an exponential time and then jumps, with where it jumps next depending only on where it is, never on how it arrived. It is summarized by one matrix, the *generator* $\mathbf{Q}$, indexed by markings:

$$\mathbf{Q}_{M,M'} = \sum_{t:\, M \xrightarrow{t} M'} \lambda_t(M) \quad (M' \neq M), \qquad\qquad \mathbf{Q}_{M,M} = -\sum_{M' \neq M} \mathbf{Q}_{M,M'}. \tag{C1}$$

The sum in the first entry is over transitions rather than a single one because two different transitions may lead to the same successor, and then their rates add. The diagonal is fixed by the requirement that each row sum to zero, which is just the statement that the process must go somewhere. A *marking-dependent* rate $\lambda_t(M)$ is permitted throughout and changes nothing structural: it alters the numbers in $\mathbf{Q}$, not its shape.

**Absorbing states, and the first of the four computations.** A state is *absorbing* if it has no outgoing transitions: once entered, the process stays. Partition the states into transient ones $\mathcal{T}$ and absorbing ones, the latter grouped into disjoint outcome classes. Write $u(M)$ for the probability that, started at $M$, the process is eventually absorbed in a designated class $\mathcal{G}$. Then $u$ is the unique solution of

$$u(M) = \frac{1}{-\mathbf{Q}_{M,M}} \sum_{M' \neq M} \mathbf{Q}_{M,M'}\, u(M'), \qquad M \in \mathcal{T}, \tag{C2}$$

subject to $u \equiv 1$ on $\mathcal{G}$ and $u \equiv 0$ on every other absorbing class. That is one linear system of size $|\mathcal{T}|$. In words: the chance of finishing well from here is the average of the chances from wherever you go next, weighted by how likely each next state is. Solving (C2) is what this paper means by an *exact* probability. No trajectory is ever simulated, so the answer carries no sampling error and cannot be improved by running longer.

**Expected time, the second computation.** Replacing the boundary condition with a unit cost per unit time gives the expected time to absorption $h(M)$, solving

$$h(M) = \frac{1}{-\mathbf{Q}_{M,M}} + \frac{1}{-\mathbf{Q}_{M,M}} \sum_{M' \neq M} \mathbf{Q}_{M,M'}\, h(M'), \qquad h \equiv 0 \text{ on absorbing states,} \tag{C3}$$

the leading term being the mean holding time $1/\sum_i \lambda_i$ at $M$. This is the same matrix with a different right-hand side, which is why a paper can report "how likely" and "how long" at essentially the same cost, and why the two can be separated cleanly: (C2) does not contain a single holding time, so a change that alters only how long the process dawdles cannot change an absorption probability.

**The embedded jump chain, and why some results are rate-free.** Ignore the waiting times and record only the sequence of states visited. That discrete chain, the *embedded jump chain*, has transition probabilities $P_{M,M'} = \mathbf{Q}_{M,M'}/(-\mathbf{Q}_{M,M})$, which are exactly the race probabilities of the second paragraph. Equation (C2) is a statement about it alone, and this is the formal reason a claim of the form "this changes the timing and not the risk" can be made exactly rather than approximately: the holding times were never in the equation. It is also why a rate can be rescaled globally without moving any absorption probability, only the clock.

**Finite horizons and uniformization, the third computation.** A deadline turns an eventual question into a horizon question: not *will* it finish, but will it finish *by* $D$. The state distribution at time $D$ is $\pi(D) = \pi(0)e^{\mathbf{Q}D}$, and the matrix exponential is computed here by *uniformization*, which is worth one sentence of explanation because the name is opaque. Pick $\Gamma \geq \max_M(-\mathbf{Q}_{M,M})$, so that $\Gamma$ is at least as fast as anything in the chain, and write $P = I + \mathbf{Q}/\Gamma$, a genuine stochastic matrix. Then

$$e^{\mathbf{Q}D} = \sum_{n \geq 0} e^{-\Gamma D} \frac{(\Gamma D)^n}{n!}\, P^n, \tag{C4}$$

that is: run a discrete chain $P$ for a Poisson$(\Gamma D)$ number of steps. The series has non-negative terms summing to one, so truncating it at a tail mass of $10^{-12}$ gives a bound on the error rather than an estimate of it. Deadline quantities in this paper are computed this way and are exact to that tolerance.

**What is exact and what is not.** The distinction matters when reading the results, so it is stated plainly. The *computations* above are exact: given a generator, an absorption probability carries no sampling error. The *inputs* are not: a rate estimated from measured latencies has a confidence interval, and a branch probability read off an evaluation has a sample size. And the *structural* claims are of a third kind entirely: an unreachable marking has probability zero under *every* admissible assignment of rates, so no estimate of any rate can refute it. Where this paper reports a zero obtained structurally and a small number obtained stochastically, they are not two measurements of the same kind, and the difference is the point.

**Where this paper uses each.** One notational warning first: $Q$ in italics is the quarantine place of §3, and $\mathbf{Q}$ in bold is the generator; they are unrelated. The closed form (5) is the race of the second paragraph applied $N$ times, each round being a contest between the hazard-advancing $t_S$ at rate $\lambda_S$ and the stabilizer $t_{\text{cool}}$ at rate $\mu$, so each round is lost with probability $\lambda_S/(\lambda_S + \mu)$ and the rounds are independent given the marking. Theorem 4 is a statement about the embedded jump chain and nothing else, which is exactly why it can assert that a neutral cycle changes the expected time by (C3) while leaving (C2) untouched: the cycle contributes holding time, and holding times do not appear in (C2). E3 solves (C2) directly on the composed net and agrees with the closed form to $3 \times 10^{-16}$, which is machine precision rather than a good fit. Everything structural, namely reachability, the depletion bound and the screen, uses none of this appendix and holds for every admissible assignment of rates.

## D From a real agentic workflow to a net

The most common objection to this line of work is not that the theory is wrong but that nobody knows how to get from the thing they actually built to the thing we analyze. This appendix does that translation explicitly, on the running example, starting from a specification of the kind teams really write.

### THE CORRESPONDENCE

Almost all of the work is one decision made repeatedly: *is this a condition or an event?* Conditions and resources become places; events become transitions. Everything else follows.

| In the agent system | Becomes | Decision it forces |
|---|---|---|
| an agent | *not* a place: a label on transitions, and a field of the color on the marks it handles | none; agents are who acts, not where state lives |
| a task, request, or ticket | one mark flowing through control places | how many can be in flight at once |
| an agent phase (planning, waiting on a tool, awaiting review) | a control place holding that task's mark | which phases are worth distinguishing |
| an LLM call that decides between options | one transition per option, competing from the same place | the policy is the *relative rates*, not new structure |
| a tool call | a transition that takes a slot mark, a place for "in flight", and a second transition to finish | whether in-flight state matters; if a failure can strike mid-call, it must |
| tool outcomes: ok, error, timeout | competing transitions out of the in-flight place, with rates | the observed outcome distribution, which is measurable |
| a retry | an arc back to the pre-call place, plus a retry-budget place | the budget place is what makes retries bounded rather than a livelock |
| a handoff or delegation $A \to B$ | a place written by a transition of $A$ and read by one of $B$ | this place *is* the interface, and naming it is the whole design review |
| retrieved context or a memory read | a mark whose age matters, consumed through an age interval | how old is too old |
| a human approval | a place fed by a human-labeled transition and *consumed* by the action | consumed, not tested, or one approval authorizes many actions |
| a capability or permission | a field of the color, plus a guard on the tool transition | which agent may ever hold it |
| a rate limit or quota of $K$ | a pool place with $K$ marks, taken and returned | returned on *every* exit path, including failures |
| a deadline or SLA | not a net element: a horizon in the transient analysis | what counts as too late |

### WORKED TRANSLATION: THREE TEAMS, THREE CONFIGS

The point of this paper is that composition failures are not visible in any one agent's specification, so the translation must start from three of them, written by three teams who never met.

```
# team A                  # team B                  # team C
agent: deployer           agent: reliability       agent: capacity
on: release_ready         on: cpu_anomaly          on: capacity_deficit
do: ship(); warmup()      do: quarantine(1)        do: reroute()
guarantee:                guarantee:               guarantee:
never removes             <=1 instance             never removes
capacity                  per anomaly              capacity
```

Each config translates on its own exactly as in §3: the `do` steps are transitions, the pre- and post-conditions are places, and each team's `guarantee` is a local specification $\varphi_i$ that its net genuinely satisfies. Translating each one alone is easy, and it is also useless, because the failure is in none of them.

The translation step that matters is the one no single configuration file can supply, because it spans two of them. Look at what each agent *reads* and what it *writes*, and ask whether any agent's output is another's input:

| | **reads** | **writes** | **fused with** |
|---|---|---|---|
| deployer | release ready | release live, *CPU pressure* | B's trigger |
| reliability | *CPU pressure* | quarantined, *capacity deficit* | C's trigger |
| capacity | *capacity deficit* | *CPU pressure* | B's trigger |

The italicized names appear in more than one row, and each such coincidence is a fusion: the same physical signal, named separately by two teams. Applying Definition 1 to those pairs is the entire composition step, and it is what turns three acyclic nets into one containing a cycle. Note that nothing here required reading any agent's code. It required listing inputs and outputs and noticing that `cpu_anomaly` is produced by two agents and consumed by a third.

Two practical warnings. First, teams name the same signal differently, so the coincidences must be found by what the signals *are* rather than by string matching; this is the step that needs a human. Second, a signal that is written by nobody in isolation is exactly the dangerous case: the reliability agent's trigger has no producer in its own net, which is why its local guarantee holds and why testing it alone cannot expose the meltdown.

### Where this sits among agentic architectures

Agentic systems are usually presented as an escalating ladder: a chat call; chat with retrieval; an agentic RAG pipeline that rewrites, reranks and cites; tools under a shared quota; multi-agent crews handing work between them; and finally agents that act on the world, write memory, and ask humans for approval (Li and Zhu, 2025). That ladder is close to the order in which the structural features of a net become necessary, and it is worth being explicit about which rung this paper occupies.

At the bottom rung a net earns nothing: a single chat call is one event with no retained state, nothing is reachable and nothing is spent. Retrieval adds context that is read at one moment and used at another, which is a temporal property: whether a snapshot is still valid when it is used is a question about elapsed time, and it is orthogonal to the composition question asked here. An agentic pipeline adds loops, so termination and retry budgets become meaningful. Shared quotas add contention, which is the classical use of Petri nets.

This paper sits at the rung where composition begins. Multi-agent workflows, whether organized as crews handing work between them or as an orchestrator dispatching to specialized agents, introduce places that one agent writes and another reads, and those interface places are exactly where local guarantees stop composing. One modeling decision on this rung decides whether the analysis can see anything: a knowledge store or signal read and written by several crews is a *single* place, not one per crew. Modeling it per crew reproduces the very mistake, testing each agent against its own private copy of the world, that §3 is about.

One further note on translation. A workflow specified as a task graph $W = (T_W, D_W)$ with subtasks $T_W$, dependencies $D_W$, and an assignment $\phi : T_W \to A$ of subtasks to agents (Yang and Zhu, 2026) needs no encoding effort at all: subtasks become transitions, dependencies become the places between them, and $\phi$ becomes the agent label carried by each transition. Where teams already specify workflows this way, the net is a change of notation rather than a change of model.

### What not to model

More effort is wasted here than anywhere else. The net is a model of *externally relevant behavior*, not of cognition, and four things should stay out of it.

*Prompt text and reasoning traces.* What the model thought is not a condition anything can depend on. It becomes, at most, an outcome distribution on competing transitions.

*Message contents.* An email body is unbounded and would make the state space infinite. Abstract it to the finitely many classes that gate a decision: sender trusted or not, request present or not, provenance class. That abstraction is also where modeling error enters, so it is worth writing down explicitly.

*Tool internals.* A database is a rate and an outcome distribution. Modeling its query planner adds states and answers no question we asked.

Table E1: Four agent compositions. Column four is the nonnegative functional $V$ of §4.2: the substrate that each round of the cycle spends and that no interface-level dashboard reports, because by construction the interface looks identical at the start of every round.

| **Setting** | **The agents** | **The fusion nobody declared** | **The substrate $V$** | **Shows** |
|---|---|---|---|---|
| software factory | issue-fixing, dependency-bumping, flaky-test triage | "main is green" is written by one team and read as a precondition by two | tests still enforcing | Thm. 1, Cor. 1 |
| customer operations | refund, fraud review, retention | *credit issued* and *anomalous credit velocity* are one event | goodwill allowance; reviewers on shift | interface blindness |
| research crew | planner, workers, critic | the critic's verdict is also the planner's trigger | shared search quota (dissipative) vs. none (neutral) | Thm. 4 |
| platform security | secret rotation, deployment | *a credential* and *the credential an instance holds* | serving instances | Thm. 3 |

*Anything you will not act on.* If no design decision depends on distinguishing two phases, merge them. Nets grow fast and the discipline that keeps them small is refusing to represent detail that changes no answer.

### A CHECKLIST

1. List the **consequential actions**: the ones whose effects outlive the episode. Only these need modeling carefully.
2. For each, write its preconditions as things that are **held**, not things that are checked.
3. Decide, for each precondition, whether it is **spent** (an arc in) or merely **tested** (a self-loop). This is the decision that most often decides whether the analysis finds anything.
4. Name every **handoff place** between agents. These are the interfaces, and they are where composition failures live.
5. Add the **resource pools** with their real capacities, and check every exit path returns the slot.
6. Mark what is **outside your control**: incoming messages, timeouts, failures, other teams' agents, the world changing.
7. Only now attach **rates**, from measured latencies, and outcome probabilities, from evaluation logs.

Steps 1 to 6 are structural and give the guarantees that hold for every rate assignment. Step 7 is what turns the same object into the quantitative questions. Doing step 7 first is the usual mistake: it produces a simulator rather than a model, and simulators answer only the questions you already thought to ask.

## E FOUR COMPOSITIONS, WORKED

The meltdown is an infrastructure example because infrastructure is where the depleted resource is easiest to point at. The mechanism is not about infrastructure. This appendix works four compositions of agents that teams are building now, each chosen because it exercises a *different* result of the paper: the depletion bound, the blindness of interface monitors, the distinction between a neutral and a dissipative cycle, and invariant interference. They are constructions rather than incident reports, in the same sense §3 is: each agent's rule is what the corresponding production component is supposed to do, and the pathology is what remains after every local rule is obeyed.

Figure E3 draws all four to one skeleton, which is the point of collecting them. The first question in every case is the same, and it is the one the reads/writes table of Appendix D exists to answer: *which two names, written by two teams, denote one physical thing*. That coincidence is the fusion, it never appears in any single configuration file, and finding it is the only step of this method that requires a person.

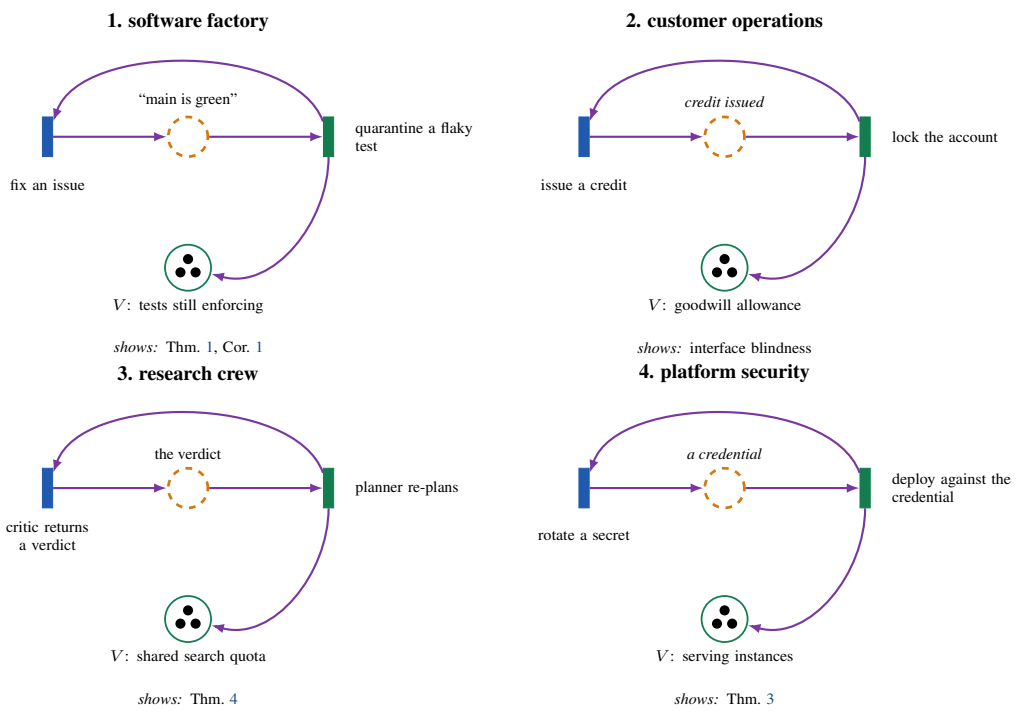



Figure E3: Four compositions of Appendix E, drawn to the same skeleton on purpose. In each, a transition of one agent marks a place a second agent consumes and the second agent's firing restores it; in three of the four a substrate that no interface-level dashboard reports goes down by one, and case 3 is the control in which it does not. The settings share no vocabulary (a green build, a goodwill budget, a search quota, a serving fleet), and the only thing that has to be discovered in each case is the same thing: *which two names, written by two teams, denote one physical object*. That is the dashed place, it appears in no single configuration file, and finding it is the one step of this method that still requires a person. Case 3 is the control: delete the arc into $V$ and the identical cycle becomes neutral, changing when the system finishes and not whether it survives (Theorem 4).

### 1. The software factory: three coding agents and a green build

Three agents on one repository, each a product several vendors now ship. $A_P$ takes an issue, writes a patch, and merges it when continuous integration is green. $A_B$ opens a dependency upgrade when a new version is published and merges it when integration is green. $A_Q$ triages the build: when a test fails in a way its classifier calls flaky, it quarantines the test, so that a transient failure does not block the queue. Write $G$ for *main is green*, $F$ for *an open build failure*, $E$ for *a test still enforcing*, $Z$ for *a test quarantined*, $J$ for *a merge queued*, and let

$$\begin{array}{llll} A_B: & t_B: G \to J, & A_P: & t_P: G \to J, \\ \text{build:} & t_{\text{ci}}: J \to F, & A_Q: & t_Q: F + E \to G + Z. \end{array}$$

Every rule is defensible in isolation and each team can say so. $A_Q$ in particular satisfies the local specification most teams would write for it: it never quarantines a test that is not currently failing, it quarantines at most one per failure, and it strictly reduces the number of open failures. It has no arc that creates a failure.

The cycle $\sigma = t_B\, t_{\text{ci}}\, t_Q$ is interface-regenerative on $I = \{G, F, J\}$: it starts and ends with main green, no open failure, and nothing queued. It is 1-dissipative in $V(M) = M(E)$. So Theorem 1 applies with $d = 1$ and $V(M^\star) = E_0$ the size of the suite, and the number of rounds to a marking with fewer than $V_{\min}$ enforcing tests is exactly $K = E_0 - V_{\min} + 1$. Corollary 1 gives $K$ as the number of traversals, and because $\sigma$ here is a three-transition circuit whose dissipating transition $t_Q$ is last, the corresponding witness is $3K$ firings; as in Proposition 6 that length is a fact about this circuit rather than a consequence of the corollary, and it holds because no transition outside $\sigma$ decreases $M(E)$ and none re-enables $t_Q$ by another route.

Two things are worth extracting. First, the terminal state is the one every engineer recognizes as the joke version of this failure, a repository whose build is always green because nothing is being checked, and it is reached without any agent doing anything it was not built to do. Second, the perverse scaling of §4.2 is on full display: the larger the test suite, the more rounds the failure takes and the more obviously benign each round is. A quarantine rate of one test per merge is unremarkable in a suite of four thousand, and the interface, which is what a dashboard shows, is identical at every round.

The repairs the model ranks are the ones a net makes visible. Removing $A_Q$'s arc to $G$ (quarantining a test does not itself re-green the build, a human or a re-run does) provably removes this cycle. Adding a quarantine quota is the same repair in a different place: it makes $Z$ bounded, which bounds $K$ directly, and it is the standard fix precisely because the quarantine quota is the fix whose accounting is legible.

### 2. Customer operations: three agents and one goodwill budget

A retention agent $A_T$ issues a service credit when a churn signal fires. A fraud agent $A_F$ locks an account whose credit velocity looks anomalous, which is a rule written to catch a real and expensive attack. A resolution agent $A_R$ handles inbound contacts, and its service-level rule is that a locked account reporting a lockout is unlocked within the window, with an apology credit for the inconvenience. Each of the three is doing what its own runbook says.

The fusion is in column three of Table E1 and it is a naming coincidence, not a design: the event $A_T$ calls *credit issued* and the signal $A_F$ reads as *credit velocity* are the same physical fact, and nothing in either team's configuration says so. The cycle is

$$t_{\text{credit}}\, t_{\text{lock}}\, t_{\text{contact}}\, t_{\text{unlock}}$$

and it is regenerative on the interface {churn signal, lock flag, open case}: after the unlock and its apology credit, every interface place is exactly where it started, including the credit-velocity signal that begins the next round. It dissipates two things at once. The goodwill allowance $V_1$ falls by the credit amount per round, and the reviewer pool $V_2$ falls by one shift-hour, because a lock-and-unlock consumes human attention in every organization that has this control at all. Theorem 1 applies to whichever of the two has the smaller $V(M^\star)/d$, which is the useful form of the statement: the composition fails at its *tightest* substrate, and that is rarely the one the agents were designed around.

This vignette is in the appendix for one specific reason. Every metric these three teams monitor is an interface metric: credits issued per contact, lock precision, unlock latency against the service level. All three are constant across rounds by construction, all three stay inside their thresholds, and the system is meanwhile spending down a budget and a staff. Operationally, "the interface resets but the substrate does not" means that the visible workflow state returns to its initial value while the underlying resource remains depleted. That is why the recommendation following from the theorem is not "add an alert" but "instrument $V$", a quantity none of the three agents references.

### 3. A research crew: which loop is worth fixing

A planner $A_L$ decomposes a question, $m$ workers $A_W$ retrieve and draft, and a critic $A_V$ accepts or returns the draft with a critique. The rejection loop is the pathology teams complain about, and the model's contribution here is to say when complaining is warranted, because *two different loops with the same shape have opposite consequences*.

Suppose the critic returns the draft and the planner re-dispatches, and suppose the workers redraft from context already in hand. The cycle is interface-regenerative (a draft exists, a critique is pending, the same question is open) and it is $0$-dissipative, since it consumes nothing that is not replenished. Theorem 4 then applies exactly: if the mission ends either by the critic accepting, at rate $\mu$, or by the deadline expiring, at rate $\lambda_S$, then the probability of missing the deadline is $\lambda_S/(\lambda_S + \mu)$ *regardless of the loop's rates*. Making the critic faster, or capping the number of revisions, changes the time to finish and does not change the probability of finishing. This is worth saying to a team, because the intuition in the room is invariably the opposite.

Now let the workers re-retrieve on each round against a rate-limited search API shared with everything else the organization runs, or against a token allowance for the mission. The cycle is unchanged in shape and now $d$-dissipative in $V(M) =$ remaining quota, and it is a regenerative dissipative

interaction cycle in the sense of §4.2. Theorem 1 bounds the rounds to exhaustion, the neutrality result no longer applies, and capping revisions is now the correct intervention rather than a placebo. The difference between the two systems is one arc, it is exactly the arc a diagram of the crew would omit, and it decides which of two contradictory pieces of advice is right. Where the substrate is an inference allowance rather than a quota, the accompanying token-budget paper makes the same object a decision problem rather than only a hazard.

### 4. Platform security: two certificates that are each true

A rotation agent $A_K$ enforces one rule: no credential is older than its maximum age, so it mints a replacement and revokes the predecessor. A deployment agent $A_D$ enforces another: no instance serves without a valid credential, so an instance whose credential is revoked is drained. Each team wrote a linear invariant and each invariant is true of its own component. For $A_K$, over the credential places, minting and revoking preserve $M(\text{current}) + M(\text{revoking}) = 1$. For $A_D$, over the fleet places, $M(\text{serving}) + M(\text{draining}) + M(\text{idle}) = N$.

The conservation invariant survives composition, since $A_K$ has no arc touching the fleet places at all. The invariant that does not survive is the one $A_D$'s safety argument actually rests on, and it is easy to miss because it is trivial in isolation. In $A_D$'s own net the credential place has neither producer nor consumer, since $A_D$ reads it and no transition of $A_D$ changes it, so $e_{\text{cred}}^{\top} C_D = 0$, and *the credential stays valid* is a local place invariant of $A_D$ in the strict sense of §4.4. That is the assumption latent in $A_D$'s test harness, where nothing else was running, and it is exactly the assumption its drain-avoidance argument consumes.

Now fuse the credential place, because $A_K$'s *current credential* and the credential $A_D$'s instances hold are one object, and compute the interference block $J_{D,K} = \widetilde{e}_{\text{cred}}^{\top} F_K C_K$. It is nonzero, since $A_K$'s revoke column moves precisely that place. The structure is identical to the meltdown, where the reliability agent's certificate depends on $U$ being unproduced and $J_{S,D}, J_{S,C} \neq 0$ say that two other agents produce it. Here the reading is that a quantity $A_D$ treated as constant is moved by another agent, and the operational consequence follows: if rotation is fleet-wide, draining is fleet-wide.

Two things follow that are worth having. The interference block is one matrix product per pair, so this is a check that runs in continuous integration across an organization's agents without anyone enumerating a global state space; that is the whole point of Theorem 3 being sufficient rather than necessary. And the repair is again structural: stage the fusion so that a credential is revoked only after the instances holding it acknowledge a replacement, which splits the fused place into two.

### Where to look for the coupling

Across all four, the fused place was one of five kinds, and this is the list we would hand a team before their first reads/writes table.

*A counted resource with more than one claimant.* Instances, runners, reviewers on shift, a quota, an allowance, seats, connection slots. These produce depletion, which is the failure mode with a bound attached.

*One event under two names.* A credit and a velocity signal, a merge and a build trigger, a mitigation and an anomaly. These are the couplings that are genuinely invisible, because both teams' documents are correct and neither mentions the other.

*A shared store.* Vector index, scratchpad, feature store, ticket system, configuration repository. A shared store is a set of fused places, one per key that two agents both touch, and the write/write pairs among them are where to start.

*A human pool.* Approvals and escalations are a resource, they are finite, and they are almost never drawn on any team's diagram, because each team drew the diagram of its own agent, in which the human is an oracle rather than a queue.

*An external rate limit.* The one substrate that is shared by construction and owned by nobody, which is why it is usually discovered during the incident rather than before it.

# F FORMAL SPECIFICATION OF THE RUNNING EXAMPLE

**Two readings of a fused initial marking.** Definition 1 requires *compatible* initial markings, so that a fused class carries one mark count and fusion means *one signal seen by several agents*. Where a class instead models a *pool* that each component contributes to, a shared budget assembled from several allocations say, the additive reading $M^0_\Phi(\gamma) = \sum_{p\in\gamma} M^0_i(p)$ is the right one. Every result in the paper holds verbatim under either, because both fix a single initial vector $M^0_\Phi$ and nothing afterwards depends on which fixed it: the incidence matrix $C_\Phi$ is unchanged, so the state equation, the screen (4) and the interference blocks $J_{ij}$ are all unaffected, and only the numerical value of $V(M_0)$, hence the bound $K$ of Theorem 1, differs. All the nets in this paper are compatible: a fused place is one account, one build status, one CPU signal.

Everything in §3 in full, so that every number in the paper can be recomputed by hand.

**Rule notation.** We write $t : X \to Y$ for the transition $t$ with $Pre(:,t) = X$ and $Post(:,t) = Y$, where $X$ and $Y$ are multisets over places written additively. So $t_S : U + H \to Q + K$ means $Pre(U,t_S) = Pre(H,t_S) = 1$, $Post(Q,t_S) = Post(K,t_S) = 1$, and all other entries of those two columns are zero. All arc weights in this paper are 0 or 1.

**The three components.** Places are $P = \{R, V, U, H, K, Q, B\}$ throughout, with the reading of §3. Each component owns a transition set and declares interface places $I_i$ (dashed in Figure 1(a)); the remaining places it touches are internal to it.

$$\begin{array}{lll} \mathcal{N}_D: & T_D = \{t_D, t_{D'}\}, & I_D = \{V, U, B\}, \quad \text{internal: } R, \\ \mathcal{N}_S: & T_S = \{t_S, t_{S,\mathrm{rb}}\}, & I_S = \{U, V, K, B\}, \quad \text{internal: } H, Q, \\ \mathcal{N}_C: & T_C = \{t_C\}, & I_C = \{K, U\}, \quad \text{internal: none.} \end{array}$$

Their incidence matrices, rows labeled by place and columns by transition, are

$$C_D = \begin{array}{c} \\ R \\ V \\ U \\ H \\ K \\ Q \\ B \end{array} \begin{array}{c} \begin{array}{cc} t_D & t_{D'} \end{array} \\ \begin{pmatrix} -1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1 \end{pmatrix} \end{array} \qquad C_S = \begin{array}{c} \\ R \\ V \\ U \\ H \\ K \\ Q \\ B \end{array} \begin{array}{c} \begin{array}{cc} t_S & t_{S,\mathrm{rb}} \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & -1 \\ -1 & -1 \\ -1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \qquad C_C = \begin{array}{c} \\ R \\ V \\ U \\ H \\ K \\ Q \\ B \end{array} \begin{array}{c} \begin{array}{c} t_C \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1 \\ 0 \\ 0 \end{pmatrix} \end{array}$$

**The fusion map and the composed net.** $\Phi$ identifies the copies of $U$ held by $\mathcal{N}_D, \mathcal{N}_S, \mathcal{N}_C$; the copies of $K$ held by $\mathcal{N}_S, \mathcal{N}_C$; and the copies of $V$ and of $B$ held by $\mathcal{N}_D, \mathcal{N}_S$. Every other class is a singleton. By Definition 1 the composed net has the 7 places above, the 5 transitions $T_D \sqcup T_S \sqcup T_C$, and incidence

$$C = \begin{array}{c} \\ R \\ V \\ U \\ H \\ K \\ Q \\ B \end{array} \begin{array}{c} \begin{array}{ccccc} t_D & t_{D'} & t_S & t_{S,\mathrm{rb}} & t_C \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{F5}$$

with $M_0 = (1, 0, 0, N, 0, 0, 0)^\top$ and $\mathcal{B} = \{M : M(H) = 0\}$.

**The two surviving invariants.** Summing rows $H$ and $Q$ of (F5) gives 0, so $(e_H + e_Q)^\top C = 0$ and $M(H) + M(Q) = N$ on every reachable marking. Summing rows $R$, $V$ and $B$ also gives 0, so $M(R) + M(V) + M(B) = 1$. Both are read off the matrix without enumerating anything.

**The certificate that does not survive.** The reliability agent's local argument is that $M(U)$ never increases, so the single injected anomaly can be spent only once and $M(H) \geq N - 1$. Formally the

argument uses $e_U^\top C_S = (-1, -1) \leq 0$. Composition breaks precisely this step:

$$J_{S,D} = e_U^\top C_D = (1,\, 0) \neq 0, \qquad J_{S,C} = e_U^\top C_C = (1) \neq 0,$$

so both the deployment agent and the capacity agent can raise $M(U)$, and Theorem 3's hypothesis fails. These are the blocks reported in E1, and computing them costs two matrix products.

**The two structural objects.** The *agent interaction graph* $G_A = (V_A, E_A)$ has a vertex per agent and an edge $i \to j$ whenever some transition of agent $i$ can add a mark to an interface place that some transition of agent $j$ consumes. Here $A_D \to A_S$ (via $U$ and $V$), $A_S \to A_C$ (via $K$), $A_C \to A_S$ (via $U$), and $A_S \to A_D$ (via $B$), so $G_A$ has two directed cycles. A *cross-agent causal cycle* is a directed cycle in the bipartite place/transition graph whose transitions carry at least two distinct agent labels; the two here are the meltdown cycle $U \to t_S \to K \to t_C \to U$ and the thrash cycle $R \to t_D \to U \to t_{S,\mathrm{rb}} \to B \to t_{D'} \to R$.

**The witness, marking by marking.** With $N = 3$, the shortest sequence into $\mathcal{B}$ is $\sigma_3 = t_D\,(t_S t_C)^2\,t_S$, of length $2N = 6$:

| step | fired | $R$ | $V$ | $U$ | $H$ | $K$ | $Q$ | $B$ |
|---|---|---|---|---|---|---|---|---|
| 0 | (initial) | 1 | 0 | 0 | 3 | 0 | 0 | 0 |
| 1 | $t_D$ | 0 | 1 | 1 | 3 | 0 | 0 | 0 |
| 2 | $t_S$ | 0 | 1 | 0 | 2 | 1 | 1 | 0 |
| 3 | $t_C$ | 0 | 1 | 1 | 2 | 0 | 1 | 0 |
| 4 | $t_S$ | 0 | 1 | 0 | 1 | 1 | 2 | 0 |
| 5 | $t_C$ | 0 | 1 | 1 | 1 | 0 | 2 | 0 |
| 6 | $t_S$ | 0 | 1 | 0 | **0** | 1 | 3 | 0 |

Rows $2 \to 4$ and $4 \to 6$ are the ratchet of (3): the interface pair $(M(U), M(K)) = (1, 0)$ at steps 1, 3 and 5 is identical every time, while $M(H)$ falls by one each round. An interface-level monitor watching $U$ and $K$ sees the same reading three times and reports nothing. Note also that no $t_C$ is needed after step 6, which is why the witness has length $2N$ and not $2N + 1$.

**Every step is legal.** At step 2 the reliability agent consumed one anomaly and removed one instance, which is exactly its specification; at step 3 the capacity agent observed a deficit and rerouted, which is exactly its specification. No agent deviates at any step, and the failure is the composition of six correct actions.

**A note on coordinates.** Appendix F lists all components over the same seven places, while the incidence matrices displayed in Appendix H omit the rows a component never touches, so that $C_D$ and $C_C$ appear without an $H$ row. The two are the same object: every component matrix is zero-extended into the common global coordinate system of Definition 1, which is what the embeddings $F_i$ do, and a missing row is a zero row. Products such as $J_{ij} = \widetilde{w}_i^\top F_j C_j$ are always taken in the global coordinates.

## G NOTATION

| Symbol | Meaning |
|---|---|
| $P, T$ | places and transitions of a net |
| $M$, $M_0$ | a marking (mark counts) and the initial one |
| $C_i = Post_i - Pre_i$ | incidence matrix of agent $i$'s net |
| $\mathcal{N}_i$, $\|_i\mathcal{N}_i$ | agent $i$'s net; the composition |
| $\Phi$ | fusion map: which interface places are identified on composition |
| $L_i$, $L_\otimes$, $L_\|$ | isolated language; shuffle baseline; composed language |
| $\varphi_i$, $\varphi_{\text{sys}}$ | agent $i$'s local specification; the system specification |
| $\mathcal{B}$ | unsafe markings; here $\{M : M(H) = 0\}$ |
| $I \subseteq P$, $\pi_I$ | interface places; projection of a marking onto them |
| $V(M) = v^\top M$ | nonnegative resource functional (healthy instances, quota, budget) |
| $d$, $V_{\min}$ | resource spent per cycle; level below which $M \in \mathcal{B}$ |
| $K$ | number of cycle repetitions to depletion, Thm. 1 |
| $y \in \mathbb{N}^{\|T\|}$ | Parikh vector: how often each transition fires |
| $J_{ij} = \widetilde{w}_i^\top F_j C_j$ | invariant-interference block: can $j$ move what $i$ held constant |
| $G_A$ | agent interaction graph |
| $\lambda_t(M)$ | firing rate of $t$ at $M$ |
| $\tau_\mathcal{B}$, $R_H$ | hitting time of $\mathcal{B}$; risk $\mathbb{P}(\tau_\mathcal{B} \le H)$ |
| $N$, $m$ | healthy instances; independent service groups sharing the pool |

## H PROOFS

**Proposition 1, in full.** Write the composed place set as $P = \{R, V, U, H, K, Q, B\}$ and the transitions as in (2), with $M_0(R) = 1$, $M_0(H) = N$, and all other places empty.

*$A_D$ and $A_C$ in isolation.* $P_D = \{R, V, U, B\}$ and $P_C = \{K, U\}$, neither of which contains $H$. Hence every column of $C_D$ and of $C_C$ has a zero entry on $H$, and by the state equation $M = M^0 + C\sigma$ the value $M(H)$ is the same at every reachable marking, so $\varphi_D$ and $\varphi_C$ hold and neither net can reach $\mathcal{B}$ from any initial marking with $M(H) \ge 1$.

*$A_S$ in isolation.* In $\mathcal{N}_S$ the places $U$ and $V$ have no producer, so $Post_S(U, \cdot) = 0$ and $Post_S(V, \cdot) = 0$, and testing requires injecting marks by hand. With $M(U) = 1$ and $M(H) = N$, exactly one of $t_S, t_{S,\text{rb}}$ can fire, each consuming the single $U$ mark, after which no transition is enabled. Thus every reachable marking has $M(H) \ge N - 1$, and at most one instance is removed per injected anomaly, which is $\varphi_S$.

*Composition.* Fusing $U, K, V, B$ makes $t_D$ a producer of $U$ and $t_C$ a second producer of $U$. From $M_0$, firing $t_D$ gives $M(V) = M(U) = 1$, $M(R) = 0$, $M(H) = N$. Now suppose $M(U) = 1$, $M(K) = 0$, $M(H) = k \ge 1$. Then $t_S$ is enabled, since $Pre(U, t_S) = Pre(H, t_S) = 1$ and $k \ge 1$, and firing gives $M(U) = 0$, $M(H) = k - 1$, $M(K) = 1$. Then $t_C$ is enabled and firing gives $M(K) = 0$, $M(U) = 1$, which is (3). Induction from $k = N$ downward: the $j$th traversal begins with $M(H) = N - j + 1 \ge 1$ for $j \le N$, so all $N$ traversals are enabled and after the $N$th we have $M(H) = 0$.

Two features of this proof are what generalize. First, the isolated-safety argument for $A_S$ rests entirely on a structural fact about the isolated net, that $U$ has no producer, and that fact is destroyed by composition without any local rule changing. This is why testing an agent against a hand-injected signal certifies nothing about its behavior when a peer generates that signal. Second, the composed argument never inspects the internals of any agent; it uses only the interface behavior (3), which is why the mechanism can be searched for structurally rather than simulated.

**Proposition 5** (Realizability of a circuit solution)**.** *Suppose $y$ solves (4) with $y_t = 1$ on its support; the support induces a subnet in which every place has exactly one input and one output transition; exactly one subnet place is marked at $M^\star$; and every non-subnet input of a subnet transition is marked at $M^\star$ and untouched by the other subnet transitions. Then the subnet is a simple circuit whose transitions fire in cyclic order from $M^\star$, so $y$ is realizable as an interface-regenerative sequence.*

**Theorem 1.** Write $M_0 = M^\star$ and let $M_k$ be the marking after $k$ repetitions. By (ii) the interface marking is $\pi_I(M^\star)$ at every $M_k$, and by (iii) $V(M_k) \le V(M^\star) - kd$. Suppose $k < K$; then

$V(M_k) \geq V_{\min}$ by the minimality of $K$, so (i) applies at $M_k$ and the whole of $\sigma$ is executable, giving $M_{k+1}$. Induction supplies $M_1, \ldots, M_K$, and $V(M_K) < V_{\min}$ by the definition of $K$, so $M_K \in \mathcal{B}$. The strength of (i) is used exactly once and it is the step that matters: at $k = K - 1$ the marking still satisfies $V \geq V_{\min}$ but may satisfy only $V = V_{\min}$, so a hypothesis requiring $V \geq V_{\min} + d$ would not license the final traversal.

**Corollary 1.** Each traversal of $\sigma$ contains the net's $V$-decreasing transition once, and it decreases $V$ by $d$, while every other transition leaves $V$ unchanged or raises it. Reaching $\mathcal{B}$ from $M^\star$ needs $V(M) < V_{\min}$, which by the definition of $K$ in Theorem 1 requires at least $K$ $V$-decreasing firings, and $K$ repetitions of $\sigma$ supply exactly that many.

Corollary 1 does not bound the *length* of a shortest witness, and it is worth saying why not, since an earlier version of this corollary claimed one. The hypotheses do not prevent a $V$-decreasing transition of $\sigma$ from being re-enabled by a route outside $\sigma$, and do not rule out a cheaper ordering of the same multiset of firings. Either breaks the arithmetic $K|\sigma| - r$, with $r$ the number of transitions of $\sigma$ after the $V$-decreasing one. A length claim needs the enabling structure, and Proposition 6 supplies it for the running example.

**Theorem 3.** A transition of agent $i$ preserves $\widetilde{W}_i$ by the local relation. A transition of agent $j \neq i$ changes the marking by a column of $F_j C_j$, whose inner product with every column of $\widetilde{W}_i$ is zero by hypothesis.

**Two variants of the model.**

**Remark 1** (If the capacity agent can also scale out)**.** Suppose $A_C$ may add instances rather than only reroute, drawing on a quota place $\beta$: $t_{C^+} : K + \beta \to H$. The meltdown does not disappear, it changes denomination. The cycle is then dissipative in the combined resource $V(M) = M(H) + M(\beta)$, still with $d = 1$, so it exhausts the quota first and the instances afterwards. With an unbounded quota the net becomes unbounded in $Q$ and in cumulative spend, so the outcome is a cost blowup instead of an outage.

**Remark 2** (Failing the test is not proof of danger)**.** If the condition fails we conclude only that the local certificate is insufficient, not that the system is unsafe. A certificate may fail to compose while the system remains perfectly safe, and the correct response is further analysis rather than automatic rejection.

**Proposition 3.** Since the merged classes start empty, $\alpha(M^0_{\Phi'}) = M^0_\Phi$. Firing a transition in $\mathcal{N}_{\Phi'}$ requires the marks in the $\Phi'$-classes it touches; each such class lies in a $\Phi$-class whose aggregated marking is at least as large, so the same transition fires in $\mathcal{N}_\Phi$ with the same effect on the aggregated state. By induction on sequence length every sequence firable under $\Phi'$ is firable under $\Phi$, and $\alpha$ commutes with firing. The converse fails because fusion adds rows of $C$ that can enable a transition under $\Phi$ with no counterpart under $\Phi'$.

**Theorem 2, in full.** Necessity is immediate. For sufficiency, let $y \in \mathbb{Q}^{|T|}_{\geq 0}$ be feasible. Every constraint in (4) is invariant under scaling by any $c \geq 1$: $C_I(cy) = cC_I y = 0$; $v^\top C(cy) = c\, v^\top C y \leq -c \leq -1$; and each support constraint likewise scales to $\geq c \geq 1$. Choose $c$ to be the least common multiple of the denominators of the entries of $y$, which is at least 1; then $cy \in \mathbb{N}^{|T|}$ and satisfies every constraint. Feasibility of the rational relaxation is a linear program, hence polynomial.

**Proposition 5.** The subnet is a directed circuit alternating places and transitions with one circulating mark. Order the transitions from the one whose input place is marked; each is enabled when reached, since its subnet input holds the mark by induction and its other inputs are marked by hypothesis and untouched. Firing all of them returns the mark, so $\pi_I$ is restored with firing count exactly $y$.

**Proposition 2.** From $M(R) = 1$: $t_D$ gives $M(V) = M(U) = 1$; $t_{S,\mathrm{rb}}$ consumes both and gives $M(B) = 1$; $t_{D'}$ restores $M(R) = 1$. The marking is identical to the start, so the sequence repeats forever. In isolation $t_{D'}$ is dead in $\mathcal{N}_D$ because $B$ has no producer, and $t_{S,\mathrm{rb}}$ is dead in $\mathcal{N}_S$ because $U$ and $V$ have none, so neither net has a cycle.

**Theorem 1, remarks on the hypotheses.** Hypothesis (i) is the one that must be checked against reachability rather than structure, since the state equation can admit a Parikh vector satisfying (ii) and (iii) that no firing order realizes; Proposition 5 gives a checkable sufficient condition. Hypothesis (ii) makes the cycle repeatable, without which the sequence may fire once and strand the interface in a state that no longer triggers it. Hypothesis (iii) with $d > 0$ strictly makes it monotone; a cycle that returns the resource, the case $d = 0$, is exactly the neutral cycle of Theorem 4. The bound $K$ counts cycle traversals rather than wall-clock time, for which see (5).

**Theorem 4, stated and proved.** The statement is deferred from §4.5.

**Theorem 4** (Neutral cycles inflate time but not risk)**.** *Let $s^\star$ be a marking of a finite continuous-time Markov chain whose only exits are a hazard-advancing transition $t_S$ with rate $\lambda_S$, an absorbing-safe transition $t_{\mathrm{cool}}$ with rate $\mu$, and a cycle $\sigma_0$ entered at rate $\lambda_0 > 0$ that is interface-regenerative and 0-dissipative and returns to $s^\star$ almost surely, in finite expected time $\theta > 0$, without changing $V$ and without passing through the hazard, the safe state, or any other exit. Then*

$$p = \mathbb{P}(t_S \text{ before } t_{\mathrm{cool}}) = \frac{\lambda_S}{\lambda_S + \mu}, \qquad \mathbb{E}[T_{\mathrm{exit}}] = \frac{1 + \lambda_0 \theta}{\lambda_S + \mu} > \frac{1}{\lambda_S + \mu},$$

*so $p$ is free of $\lambda_0$ and of every rate inside $\sigma_0$, while the expected time to leave $s^\star$ by either exit is strictly inflated by the cycle.*

Let $p$ be the probability of eventually firing $t_S$ before $t_{\mathrm{cool}}$ starting from $s^\star$. The first jump is $t_S$ with probability $a = \lambda_S/\Lambda$, $t_{\mathrm{cool}}$ with probability $b = \mu/\Lambda$, and into $\sigma_0$ with probability $c = \lambda_0/\Lambda$, where $\Lambda = \lambda_S + \mu + \lambda_0$. Since $\sigma_0$ returns to $s^\star$ almost surely without changing $V$, and $t_{\mathrm{cool}}$ is not enabled elsewhere along it, the strong Markov property at the return time gives $p = a + cp$, so $p = a/(1-c) = \lambda_S/(\lambda_S + \mu)$, which is free of $\lambda_0$ and of every rate inside $\sigma_0$. For the time, let $\theta$ be the expected duration of one excursion through $\sigma_0$, which is positive and finite because $\sigma_0$ is a finite cycle of positive rates. The number of excursions before exit is geometric with success probability $1 - c$, but each visit to $s^\star$ also costs its own holding time $1/\Lambda$, and a count of excursions alone omits those. Conditioning on the first jump instead gives the recurrence

$$\mathbb{E} = \frac{1}{\Lambda} + c\,(\theta + \mathbb{E}), \qquad \text{so} \qquad \mathbb{E} = \frac{1/\Lambda + c\theta}{1 - c} = \frac{1 + \lambda_0 \theta}{\lambda_S + \mu} > \frac{1}{\lambda_S + \mu},$$

using $1 - c = (\lambda_S + \mu)/\Lambda$. The hypothesis that $\sigma_0$ returns almost surely, in finite expected time, and offers no exit of its own is what licenses both the $p = a + cp$ step and this one; a cycle with a side exit, a hazard state or a safe state along it satisfies neither. Both halves are confirmed numerically in §5: the probability column is identical to five decimals across all cooldown ratios, while the absorption-time column inflates by a factor between 1.33 and 2.00.

**Proposition 4.** Let $t$ be any transition of the composed net, belonging to some component $j$. Then $\widetilde{w}_i^\top C_\Phi(:, t) = \widetilde{w}_i^\top F_j C_j(:, t)$, which is $\leq 0$ by hypothesis when $j \neq i$ and by $w_i^\top C_i \leq 0$ when $j = i$. So every firing changes $\widetilde{w}_i^\top M$ by a nonpositive amount, and the quantity is nonincreasing along every sequence. The equality case $\widetilde{W}_i^\top F_j C_j = 0$ of Theorem 3 is the same argument with $\leq$ replaced by $=$. Note the asymmetry the one-sided version introduces: a strictly negative entry of $J_{ij}$ preserves a monotone certificate and destroys an equality one, which is why the two tests differ.

**Proposition 6 (shortest witness for the meltdown), stated and proved.** The statement is deferred from §4.2.

**Proposition 6** (Shortest witness for the meltdown)**.** *In the composed net of §3 with $M_0(H) = N \geq 1$, every firing sequence from $M_0$ reaching $\mathcal{B} = \{M(H) = 0\}$ has length at least $2N$, and $t_D(t_S t_C)^{N-1} t_S$ attains it.*

Write $\#t$ for the number of times $t$ occurs in a sequence $\rho$ from $M_0$ reaching $\mathcal{B} = \{M(H) = 0\}$. In the composed net of §3 the only transition with $C(H, \cdot) < 0$ is $t_S$, with $C(H, t_S) = -1$, and no transition increases $M(H)$; since $M_0(H) = N$, reaching $M(H) = 0$ forces $\#t_S \geq N$, and in fact $\#t_S = N$ exactly, because $M(H)$ cannot go negative.

Each $t_S$ consumes one mark of $U$. $U$ is marked by $t_D$ and $t_C$, and $t_D$ needs $R$, which is restored only by $t_{D'}$ after $t_{S,\mathrm{rb}}$ has consumed both $U$ and $V$. Every $t_S$ after the first therefore needs its $U$ mark from either a $t_C$ or a fresh $t_D$.

A second $t_D$ costs at least the three extra firings $t_{S,\mathrm{rb}}$, $t_{D'}$, $t_D$ while supplying no net mark of $U$, since $t_{S,\mathrm{rb}}$ consumes one, whereas a $t_C$ costs one firing and supplies one. The cheapest supply is therefore one $t_D$ and $N-1$ firings of $t_C$, giving $|\rho| \geq \#t_S + \#t_D + \#t_C \geq N+1+(N-1) = 2N$.

For attainment, $t_D(t_S t_C)^{N-1} t_S$ is executable from $M_0$: $t_D$ marks $U$ and $V$; each $t_S$ consumes one $U$ and one $H$ and marks $K$ and $Q$; each $t_C$ consumes $K$ and returns $U$; after $N-1$ rounds $M(H) = 1$ and $M(U) = 1$, so the final $t_S$ fires and gives $M(H) = 0$. Its length is $1 + 2(N-1) + 1 = 2N$. The enumeration of Table N10 finds exactly this length for every $N$ from 2 to 12.

**A sufficient compositional-safety principle.** Collecting the conditions of §4.4: let $\mathcal{A}_1, \ldots, \mathcal{A}_N$ be interface agents and $\mathcal{S}$ a global safe set. Suppose each agent is safe under its contract assumption $\mathsf{A}_i$; contracts are compatible along the actual coupling graph; every local linear certificate used has zero invariant interference; the resulting global certificates imply $M \in \mathcal{S}$; and $M_0$ satisfies them. Then every reachable marking of the composition lies in $\mathcal{S}$. The proof is immediate from Theorem 3. The content is methodological rather than mathematical, since the compatibility condition forces the modeler to write down the environmental assumptions that isolated testing leaves implicit. A useful additional requirement, covering §4.2 directly, is the absence of any reachable regenerative dissipative interaction cycle for the resources that define $\mathcal{S}$.

## I THE CONTRACT FORMULATION

Attach to agent $i$ a contract $(\mathsf{A}_i, \mathsf{G}_i)$ of admissible input-interface traces and guaranteed output-interface traces. The three agents of §3 were implicitly verified under

$$\begin{aligned} \mathsf{A}_S &: \text{ CPU anomalies are exogenous, not caused by my own mitigation,} \\ \mathsf{A}_C &: \text{ rerouting-induced load does not itself trigger mitigation,} \\ \mathsf{A}_D &: \text{ a release, once deployed, stays deployed,} \end{aligned} \tag{I6}$$

and compatibility requires $\mathsf{G}_j \subseteq \mathsf{A}_i$ for every coupling $j \to i$. All three inclusions fail. Composition therefore invalidated the environmental assumptions under which each component was certified, which is the cleanest statement of what went wrong.

## J DYNAMIC COMPOSITION

If the coupling graph varies with time, a safe roster can become unsafe when one new edge closes a directed cycle. An admission monitor should therefore check not merely agent identity and capability but whether a proposed coupling turns the interaction graph cyclic, introduces a nonzero interference block, or creates a new solution to (4). All three are cheap enough to run at coalition-formation time.

## K A HIERARCHY OF COMPOSITION-INDUCED EMERGENCE

The main text treats structural emergence. Two further levels arise, and the same machinery covers all three, though not without change: the structural level needs only $V$ and the incidence matrix, the stochastic level adds rates on the same structure, and the epistemic level needs colored marks and update rules that are not in general linear. What is common is the regenerative cycle and the functional it moves. Figure K4 shows what each level leaves behind for an analysis to find.

*Structural.* Composition adds firing sequences or reachable markings absent from the independent baseline. Mechanisms include cross-agent enabling, new synchronization, circular wait and deadlock, duplicate commitment, and shared resource depletion. The meltdown cycle is of this kind; the thrash cycle is the degenerate case $d = 0$, producing non-termination rather than depletion.

*Stochastic.* A behavior was already possible in the baseline, but coupling changes its probability or time scale substantially. The signature is not a new reachable marking but a large $\Delta R_H$, or a large change in $\mathbb{E}[T_{\mathrm{abs}}]$ or in long-run availability. Theorem 4 is the clean instance: a neutral cycle changes only the time scale. This class is easy to miss precisely because reachability analysis reports nothing new, which argues for running the stochastic step even when the structural step is clean.

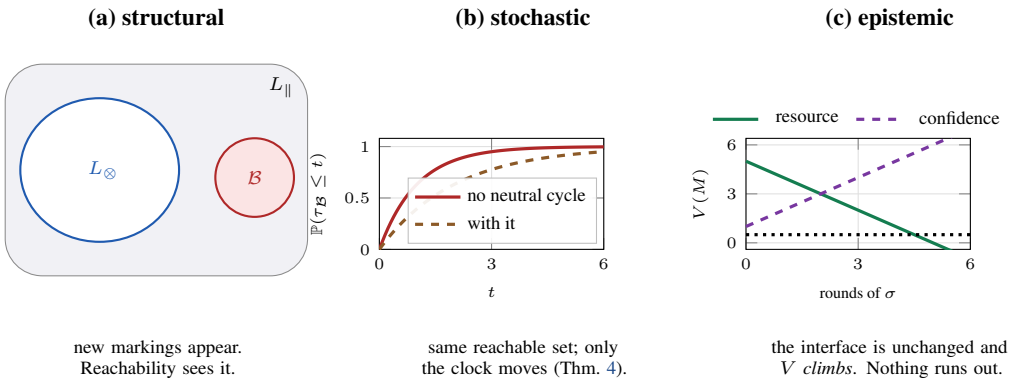



Figure K4: The three levels of composition-induced emergence, and what each one leaves for an analysis to find. **(a)** Structural: the composed language strictly contains the independent baseline, and the hazard lies in the difference. This is Definition 2, and reachability finds it. **(b)** Stochastic: nothing new is reachable, so the structural step reports a clean bill, and the coupling has still changed when the system fails. Theorem 4 is the extreme case, in which the hitting *probability* does not move at all and only the time scale does; this level is easy to miss for exactly that reason, which is the argument for running the stochastic step even after a clean structural one. **(c)** Epistemic: the same regenerative cycle with the inequality of Theorem 1 reversed. Marks carry claims, sources and confidence rather than instances, and an endorsement loop raises the confidence that makes endorsement more likely. The interface still looks identical every round; what is being consumed is the independence of the evidence, and no dashboard has a gauge for it.

*Epistemic.* Shared places may carry beliefs, evidence, confidence, or provenance rather than physical resources. With colored marks $c = (\text{claim}, \text{source}, \text{confidence}, \text{timestamp})$, one agent updates the color of a mark consumed by another, and a false-consensus loop in which endorsement raises confidence which makes endorsement more likely is a regenerative cycle whose amplified quantity is epistemic. Structurally this is Theorem 1 with the inequality reversed: the interface structure is unchanged while $V$ climbs monotonically. Detection is harder, because no resource visibly runs out; what runs out is the independence of the evidence.

## L From two benchmarks to nets

The open problem this paper shares with every applied use of nets is where the net comes from. Section 5 answers it twice, once for a benchmark that gives an API surface and once for a corpus that gives only traces, and the two answers are different in a way worth stating.

### AppWorld: a module library, not a net per task

AppWorld's published description gives nine apps, 457 APIs and 750 tasks (Trivedi et al., 2024); the release we analyze is data version `0.2.0` from repository commit `a072b7a`, retrieved 17 February 2026, which has 469 documented APIs across the nine apps and the supervisor, and 732 tasks split 90 train, 57 dev, 168 `test_normal` and 417 `test_challenge`. Every count in this paper is that release and not the one the paper describes. So: nine user-facing apps, 469 APIs, one supervisor holding the user's credentials, and 732 tasks; Figure M7 is the whole construction on one pair of agents. We write one module per app, once, against the app's own documentation, and a task's net is the restriction of that library to the APIs the task calls. No net is written per task: the library is written once against each app's own documentation and reused across every task that touches that app, which is what keeps the modeling effort proportional to the number of apps rather than to the number of tasks.

**Places are the benchmark's models** (Figure L5). AppWorld's evaluators are written in terms of names like `venmo.PaymentRequest`, so those are the places. Each model contributes two: a *cardinality* place counting how many exist, which is what a counted resource such as a cart entry

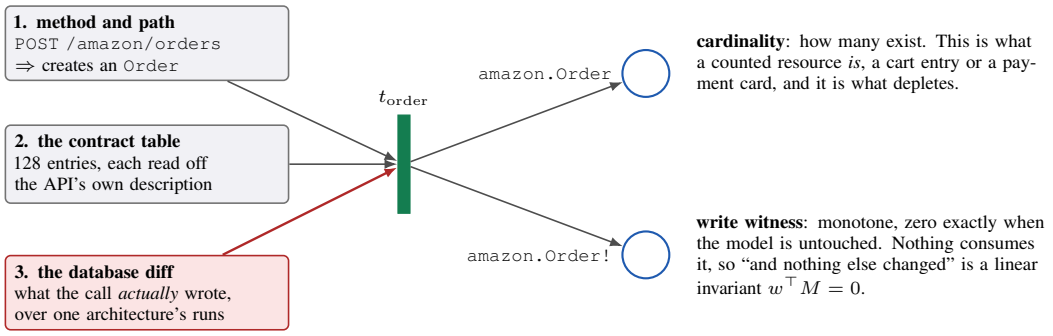



Figure L5: How one API becomes arcs, and why each model needs two places. The first two sources are a reading of AppWorld's documentation; the third is not, and it is where the twenty cross-app arcs of Table L4 come from. Splitting each model into a cardinality place and a monotone witness is what lets the same net answer both questions this paper asks: the cardinality place carries the depletion of Theorem 1, and the witness place is what makes a benchmark's `changed_model_names` assertion a linear invariant that Theorem 3 can compose. A cardinality place alone cannot express it, since an update changes a model without changing how many there are; a witness place alone cannot express depletion, since nothing ever consumes one.

or a payment card actually is, and a monotone *write-witness* place, zero exactly when the model is untouched. The witness is what turns `changed_model_names == {...}` into a linear invariant. A cardinality place cannot express it, because an update changes a model without changing how many there are; and because nothing consumes a witness, a net carrying them has an infinite reachable set, so they are on only when a certificate is being checked and off for every dynamic analysis. Leaving them on would also make the program of Theorem 2 infeasible for a trivial reason, since a place that can only increase cannot be regenerated, which would look like a negative result and would not be one.

**Transitions are APIs, and their arcs come from three sources.** The method gives the default: `GET` reads, `POST` creates, `DELETE` removes, `PATCH` updates, with the target model recovered from the path. Where the path names a parent rather than the child it writes, as `/venmo/transactions/{id}/likes` writes a like and not a transaction, an explicit contract of 128 entries says which, each read off the API's own description. Every API taking an `access_token` tests a session place that `login` produces and `logout` consumes. Nine APIs additionally move money, and those arcs are the only ones not implied by a method or a model: they are stated in the API's description ("Add money to your Venmo balance", "Place an order for the products in the cart") and they are the $V$ of Theorem 1.

**The third source is the database, and it is the interesting one.** Twenty calls write models belonging to another app and say nothing about it. We did not guess them: for every trajectory in which exactly one writing API was called, the whole recorded database delta is attributable to that call, and comparing the attribution against the library's prediction names the missing arcs. That procedure was run on one of the four published agent architectures and the resulting library evaluated on the other three, so Table L6's figures are held out. Table L4 lists what it found.

**Composition follows Definition 1.** Each task-agent gets a private copy of the nets for the apps it touches; the fusion map identifies the model and resource places across agents, because there is one Venmo account, and leaves the session places private, because each agent holds its own token. An agent's finite instruction list, where "order the four things on my list" is four adds and not arbitrarily many, is a private budget place, without which the composed net is unbounded in the cart and enumeration has nothing to report.

**What the mediator of E6 cuts.** Figure L6 draws the two readings of the interference block that Table M9 separates, and why only one of them restores the certificate.

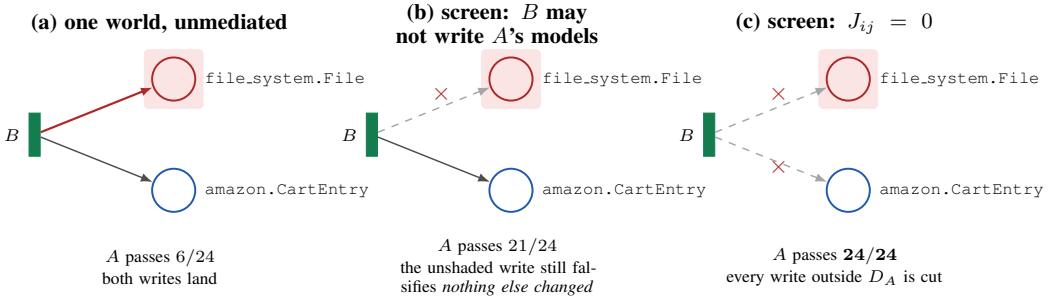



Figure L6: Why the obvious repair is not the right one (E6, Table M9). The shaded place is inside $D_A$, the set agent $A$'s evaluator names; the unshaded one is outside it, and $A$'s certificate asserts that it does *not* change. That is the point: `changed_model_names` $= D_A$ says something about *every* model place, not only about the ones in $D_A$. So cutting $B$'s writes *into* $D_A$ (b) removes the damage a reader notices, the receipt filed into the backup directory, and leaves the certificate false, because $B$'s cart entry is still a write outside the declared set. The block $J_{ij} = w_i^\top F_j C_j$ has $w_i$ supported on the witnesses *outside* $D_A$, so the cut it prescribes (c) is the complementary one, and that one restores $A$ exactly. Since both only refuse calls, neither cut can introduce a new hazard; only the second removes this one.

**Two definitions that had to be specified carefully.** The first is which fused places count as *interface* places for the screen. Three kinds do not, for one reason rather than three: a ledger nobody consumes (the order history, the sent-mail archive, a witness) records what happened and gates nothing; a place only ever tested is a monitor and not a channel through which one agent takes what another needs; and the resource itself is $V$ and must not regenerate. What is left is the set of shared places some transition strictly decreases, and $\pi_I(M') = \pi_I(M)$ is meaningful only for those. The second is what "the component alone" means. Building each component over its own places would give it a private copy of the shared account, which is the assumption under test; the components in Table L2 are therefore the sub-nets of the *composition* on one agent's transitions, in the composed place universe.

**Validation.** All 469 APIs map to arcs with none left over, and every one of the 50 models AppWorld's evaluators name is reachable by some transition of the library, a coverage check against a vocabulary we did not choose. Table L6 is the quantitative version. Thirty-four offline tests cover the reader, the library, the fusion, and each theorem's prediction, including the assertion that the interference block never misses interference the declared footprints show.

### MAST: A CORPUS WITH NO API SURFACE

MAST (Cemri et al., 2025) gives traces and labels and no interface at all, so no net is built from it. What it can supply is the agent interaction graph $G_A$ of §4, recovered per trace from the framework's own log format: nodes are the roles that spoke, and an edge $i \to j$ is a turn by $i$ followed by one by $j$. A speaker is recovered for all but five of the 1,638 usable traces. Four records carry a null annotation for every mode and are dropped rather than read as clean runs.

Two decisions are worth stating because both could have been made to flatter the result. MAST's inter-agent category is used as the interface category because its authors named it that, not because we sorted the modes; and because two of its six modes (task derailment, action-reasoning mismatch) can be described without a second agent, every headline number is recomputed on the four that cannot, with both figures reported. The human-annotated file uses three earlier drafts of the taxonomy with different numbering, so the judge-versus-human comparison maps by mode *name*; matching on the number would compare "inconsistency between reasoning and action" against "disobey role specification" and report the result as disagreement. Draft modes with no counterpart in the published fourteen are dropped and counted.

The comparison is therefore this. Of the 19 shipped human-annotated records, 17 also appear in the judged corpus and enter the agreement figure; the other two are `GAIA` records the judged corpus

Table L2: E4, depletion on AppWorld's own APIs. Two agents share one account: one fills a cart and another places the order, or one records shared expenses and another settles them. Neither can exhaust the budget alone, since the first has no $V$-decreasing transition at all and the second has one that is never enabled, and together they exhaust it in $K$ traversals of a two-transition cycle, the $K$ of Theorem 1. The length follows from the circuit directly, as in Proposition 6, and not from Corollary 1, which bounds $V$-decreasing firings and not sequence length. The last line is the negative control: two agents that both spend from one Venmo balance exhaust it, but each of them exhausts it alone, so by Definition 2 nothing there is composition-emergent and the analysis says so.

| | **cart** | | | **expenses** | |
|---|---|---|---|---|---|
| $V(M^\star)$ | predicted | measured | | measured | |
| 2 | 4 | 4 | ✓ | 4 | ✓ |
| 3 | 6 | 6 | ✓ | 6 | ✓ |
| 4 | 8 | 8 | ✓ | 8 | ✓ |
| 5 | 10 | 10 | ✓ | 10 | ✓ |
| 6 | 12 | 12 | ✓ | 12 | ✓ |
| 7 | 14 | 14 | ✓ | 14 | ✓ |
| 8 | 16 | 16 | ✓ | 16 | ✓ |
| 9 | 18 | 18 | ✓ | 18 | ✓ |
| 10 | 20 | 20 | ✓ | 20 | ✓ |
| 11 | 22 | 22 | ✓ | 22 | ✓ |
| 12 | 24 | 24 | ✓ | 24 | ✓ |

*Alone*: the shopper is $V$-invariant (9 markings, no $V$-decreasing transition); the buyer can spend but reaches 1 marking. *Control*: both Venmo agents reach $\mathcal{B}$ alone (settler 5 markings, sender 5 markings), so nothing there is composition-emergent.

Table L3: E4, screening against enumeration as more agents share one AppWorld account: $m$ shoppers and one buyer, all on the same cart. Between 2 and 7 agents the reachable set grows by a factor of 24,554 and the linear program of Theorem 2 varies by 12%, returning the same two-transition circuit at every size. Marking counts are exact; the seconds are one machine, and the screen column is the median of nine solves because a single cold call on the largest instance is an order of magnitude slower than the steady state.

| **agents** | **places** | **trans.** | **markings** | **enumerate (s)** | **screen (s)** |
|---|---|---|---|---|---|
| 2 | 13 | 5 | 35 | 0.0004 | 0.0004 |
| 3 | 15 | 8 | 390 | 0.0064 | 0.0004 |
| 4 | 17 | 11 | 3,146 | 0.0795 | 0.0004 |
| 5 | 19 | 14 | 21,903 | 0.7320 | 0.0005 |
| 6 | 21 | 17 | 140,661 | 5.9985 | 0.0005 |
| 7 | 23 | 20 | 859,420 | 55.5384 | 0.0005 |

does not contain. A mode counts as present for the humans when at least two of the three annotators marked it, and the comparison runs over the 213 (trace, mode) cells that survive the name mapping. For the 22.2% figure the denominator is 18 rather than 19, because one record carries no positive label under the mapping and so is not a failing trace. None of this is the protocol MAST used to report its own judge accuracy, and the two numbers should not be read against each other.

### Reproducing it

This appendix covers E4 to E7; the protocol and full tables for E1 to E3 are in Appendix N. `experiments/benchmarks/` holds the harness. `aw_docs.py` reads AppWorld's API documentation, task specifications and evaluators with `json` and `ast`, so the structural analysis needs no AppWorld install; `aw_net.py` is the module library; `aw_compose.py` is E4; `aw_replay.py` is E5; `run_appworld_multi.py` is E6 and does need AppWorld on Python 3.11 or 3.12; `mast_corpus.py` and `mast_analysis.py` are E7; `make_tables_p2.py` generates every benchmark table in this paper, so none is transcribed by hand. The offline analysis is `python aw_compose.py && python aw_replay.py && python mast_analysis.py && python make_tables_p2.py`, and nothing in it calls a model.

Table L4: E5, effects that cross an app boundary. Every row was found by comparing the library's prediction against AppWorld's database diff over the published trajectories, and none of them appears in the API documentation the rest of the library is built from. Of the 20, 9 are documented `GET`, so a method-based reading of the interface classifies them as reads. These are the arcs that make a file agent's certificate depend on a shopping agent.

| call | models it writes in another app |
|---|---|
| `amazon.download_order_receipt` | file_system.Directory, file_system.File |
| `amazon.download_prime_subscription_receipt` | file_system.Directory, file_system.File |
| `amazon.initiate_return` | file_system.Directory, file_system.File, gmail.Attachment, gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `amazon.place_order` | file_system.Directory, file_system.File, gmail.Attachment, gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `amazon.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `file_system.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `gmail.download_attachment` | file_system.Directory, file_system.File |
| `phone.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `simple_note.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `splitwise.download_expense_receipt_file` | file_system.Directory, file_system.File |
| `splitwise.download_payment_receipt_file` | file_system.Directory, file_system.File |
| `splitwise.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `spotify.download_premium_subscription_receipt` | file_system.Directory, file_system.File |
| `spotify.download_song` | file_system.Directory, file_system.File |
| `spotify.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `todoist.download_attachment` | file_system.Directory, file_system.File |
| `todoist.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |
| `venmo.download_bank_transfer_receipt` | file_system.Directory, file_system.File |
| `venmo.download_transaction_receipt` | file_system.Directory, file_system.File |
| `venmo.signup` | gmail.Email, gmail.GlobalEmailThread, gmail.UserEmailThread |

Table L5: E7, MAST (Cemri et al., 2025): 1,638 annotated traces from seven multi-agent systems, 1,237 of them carrying at least one failure mode. *FC2* is MAST's own inter-agent category; *relational* is the four of its six modes whose definitions name a second agent. The left-hand claim survives: a substantial share of real multi-agent failures is located at the interface with no locally attributable fault. The right-hand one does not: neither share tracks the size or the shape of $G_A$, and the reason is visible in the last line.

| system | roles | failing | FC2 | relational |
|---|---|---|---|---|
| OpenManus | 1 | 29 | 0.79 | 0.45 |
| AG2 | 2 | 286 | 0.74 | 0.27 |
| MetaGPT | 2 | 405 | 0.66 | 0.52 |
| AppWorld | 3 | 29 | 0.79 | 0.28 |
| Magentic | 3 | 153 | 0.76 | 0.56 |
| HyperAgent | 4 | 26 | 0.88 | 0.73 |
| ChatDev | 6 | 305 | 0.67 | 0.37 |

*Interface fault with no local fault*: 174 of 1237 (14.1%) by FC2; 84 (6.8%) on the relational subset; 4 of 18 (22.2%) on the human-annotated traces. $\mathbb{P}(\text{FC2} \mid \text{FC1}) = 0.74$ against $\mathbb{P}(\text{FC2} \mid \neg\text{FC1}) = 0.61$; Spearman across the seven systems $0.08$ against roles and $0.05$ against cycles. *Judge against three human annotators*: 54% agreement on 213 labels, 76 judge-only against 23 human-only. *And on 89 traces in which one role ever speaks, the judge still records* ignored other agent's input *16 times and* fail to ask for clarification *15 times.*

## M MATERIAL DEFERRED FROM THE MAIN TEXT

Sections 5 and 7 state each experiment and each recommendation in the space a nine-page body allows. This appendix holds everything that was compressed out of them, so that no step of the argument rests on a passage the reader cannot see: the experiment overview table; the full narratives of E1 to E3, E5, E6b and E7 with the threats to validity; the E4 census table, the E6 results table and the figure of the AppWorld composition; the stochastic layer of §4.5 at full length; the screening procedure together with the three inequivalent repairs for the meltdown; and the open problems. Appendix N then gives every protocol and the complete numerical tables, and Appendix L the construction of the two benchmark nets.

Table L6: E5, the net library against AppWorld's own record of what changed. 4,601 published trajectories from four agent architectures and four models; for each, the models the library says the calls write are compared with the models the database diff says were written, with the scenario's own setup subtracted. The cross-app arcs of Table L4 were discovered on the first row and the other three are held out, so the held-out line is the one to read. Recall is the more important direction, since a hazard analysis that over-approximates what an agent writes still bounds the interference, but 0.93 is not 1 and the missing 7% of realized writes are a real gap, not a conservative one. The shortfall in precision is largely proposals the environment refused, which a call log cannot distinguish from executions.

| architecture | scored | exact set | precision | recall |
|---|---|---|---|---|
| full code agent (development) | 1,236 | 62.1% | 0.51 | 0.92 |
| function calling agent | 900 | 64.7% | 0.65 | 0.94 |
| plan execute code agent | 1,048 | 65.4% | 0.77 | 0.93 |
| react code agent | 1,417 | 60.5% | 0.68 | 0.93 |
| **held out, all three** | **3,365** | **63.1%** | **0.70** | **0.93** |

1,653,611 API calls, all but 48 resolved to a documented API. Those 48 name no API at all: the agent interpolated a whole response object into the path where an identifier belonged.

Table M7: The seven experiments. All analyses are exact except where a sample size is stated; "exact" means the reachable set or the absorption linear system was solved, not sampled.

| | Claim, and how it is checked | Outcome |
|---|---|---|
| E1 | no component reaches $\mathcal{B}$, the composition does, and the shortest witness has the predicted length (Prop. 6); exact reachability, $N = 2$ to 12 | witness is exactly $2N$, every $N$ |
| E2 | screening is cheap where enumeration is not (Thm. 2); fleet of $m$ service groups | 1 ms vs 28 s at $m = 8$ |
| E3 | (5) is exact and neutral cycles cost time, not risk (Thm. 4); exact CTMC, 18 settings | $3{\times}10^{-16}$; risk flat, time up to $2\times$ |
| E4 | the benchmark's own certificates do not survive composition (Thm. 3) and depletion bites on a real API surface; $J_{ij}$ on 235,710 ordered AppWorld task pairs | 21.0% preserved, 5.7% with cascades; $2K$ witness matches $11/11$ |
| E5 | the net library says what the world does, and the API surface omits twenty cross-app effects; 4,601 trajectories vs the database diff | held-out recall 0.93, precision 0.70 |
| E6 | cutting the flagged fusions restores the certificate (Thm. 3), at the cost of the other agent; two scripted executors, AppWorld's own scorer | $24/24 \to 6/24 \to 24/24$, $B$ blocked |
| E7 | interface failure is real and not locally attributable, but does *not* track the shape of $G_A$; 1,237 annotated failures | 14.1% have no local fault; second claim is *negative* |

## THE SEVEN EXPERIMENTS AT A GLANCE

### E1 TO E3, IN FULL

**E1 to E3: the theory is right, and cheap to apply.** Enumerating the composition and each component in isolation, the latter with a hand-injected stimulus on its input interface, for every $N$ from 2 to 12: isolated $A_D$ and $A_C$ leave $M(H) = N$ untouched, isolated $A_S$ bottoms out at exactly $M(H) = N - 1$, none of the three reaches $\mathcal{B}$ for any $N$, and the composition reaches $\mathcal{B}$ for every $N$ with a shortest witness of length exactly $2N$ in all eleven cases (Table N10). Two readings matter more than the headline. The composed reachable set has only $4N + 3$ markings, so nothing is hiding in a large state space: the failure is three transitions wide and is invisible to per-component testing for a structural reason rather than a budgetary one. And the screen sees it without enumeration: two place invariants survive composition, $M(H) + M(Q) = N$ and $M(R) + M(V) + M(B) = 1$, while the reliability agent's own certificate does not, the interference blocks on the fused place $U$ being

Table M8: E4, AppWorld. Each of the 486 tasks whose evaluator declares a write footprint is one agent; a pair is the two of them on one world. A pair is *preserved* when Theorem 3's interference block $J_{ij}$ vanishes, which here means agent $j$ writes no model agent $i$'s evaluator declares unchanged. The second row uses what the APIs realizing $j$'s footprint actually write rather than what $j$'s task declares; the difference is the undocumented cross-app cascades of Table L4. Exact, no sampling.

| | **pairs** | **preserved** | **broken** |
|---|---|---|---|
| all ordered pairs | 235,710 | 49,545 (21.02%) | 186,165 |
| against realized writes | 235,710 | 13,353 (5.67%) | 222,357 |
| pairs sharing an app | 114,180 | 37,905 | 76,275 |
| pairs sharing no app | 121,530 | 11,640 | 109,890 |

*Block against the set test* on 299 sampled pairs: 250 agree, 49 block-only, 0 set-only. *Library*: 469 APIs over 10 apps, 288 of them writing, 92 model places; all 50 models the evaluators name are covered and 0 writes are unmapped.

$J_{S,D} = (1, 0)$ and $J_{S,C} = (1)$, so Thm. 3 flags the hazard from the component incidence matrices alone, before a single state is generated.

E2 scales to a fleet of $m$ independent service groups, each with its own three agents, all drawing on one shared pool of $N = 6$ healthy instances: the realistic shape of the problem, many services on one substrate. Enumeration grows by a factor of about 3.9 per group and becomes impractical by $m = 8$, where the net has 42 places, 40 transitions and 327,681 reachable markings and enumeration takes 28.3 s; the program is flat at a few milliseconds throughout and in every instance returns support exactly $\{t_S^{(0)}, t_C^{(0)}\}$, a circuit, which Prop. 5 certifies executable (Figure 3 left, Table N12). Between $m = 1$ and $m = 8$ the reachable set grows by 12,136× and enumeration time by 117,000×, while the screen changes by less than 5×.

E3 solves the absorption linear systems on the finite chain exactly, rather than simulating, for the meltdown-only model and for the full model including the thrash cycle, over 18 settings: $N \in \{3, 5, 10\}$ and $\mu/\lambda_S \in \{0, \frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$ at unit rates (Figure 3 right, Table N11). Three results. The closed form (5) is exact, not approximate: across all 18 settings it agrees with the exact chain to machine precision, maximum absolute deviation $3 \times 10^{-16}$. Adding the thrash cycle changes the outage probability by that same $3 \times 10^{-16}$, which is to say by nothing at all, in every one of the 18 settings, as Thm. 4 requires. And it inflates $\mathbb{E}[T_{\text{abs}}]$ by a factor *independent of* $N$: in all 18 settings the factor equals $1 + 2/(2 + \mu/\lambda_S)$ exactly, falling from 2.00 at $\mu = 0$ to 1.33 at $\mu = 4\lambda_S$. The fleet size sets how long the outage takes to arrive, the competing cycle sets a multiplier on it, and neither touches whether it arrives. That last point is the operationally dangerous one: a system with a busy thrash cycle fails exactly as often as one without and takes up to twice as long to do it, which is precisely the signature an operator reads as "intermittent, keep watching".

## THE E4 CENSUS TABLE

## OPEN PROBLEMS

Four questions follow directly from §7. Characterize $L_{\|} \setminus L_{\otimes}$ without constructing the global state space. Extend Proposition 5 to a necessary and sufficient condition, turning the screen of §4.3 into a decision procedure rather than a filter with a sufficient side condition. Decompose the composition-induced risk $\Delta R_H$ into per-interface contributions, a systems analogue of risk attribution, which would say not only that a coupling is hazardous but how much of the hazard it carries. And learn the net, couplings and rates included, from execution logs. The last is no longer entirely open: §5 recovers twenty real couplings from AppWorld's recorded database writes, and what it recovers is exactly the arcs the documentation omits. What is missing is a coverage guarantee, since the observed traces need not exercise every coupling present in the deployed system, and such a guarantee is what would let the method run on deployed systems rather than on designs.

### THE SCREENING PROCEDURE, AND THE THREE REPAIRS, IN FULL

**Three repairs for the meltdown, and why they are not equivalent.** Three interventions each stop the outage of §3, and they differ in what they guarantee. *Refusing to quarantine once* $M(H) \leq h_{\min}$ adds a read arc from $H$ to $t_S$ with multiplicity $h_{\min} + 1$, which breaks the cycle: the marking $M(H) = h_{\min}$ no longer enables $t_S$, so by distribution freedom (§4.5) no bad marking is reachable under any rate assignment. It also costs mitigation capability exactly when the fleet is smallest. *Coloring the CPU mark with its provenance* and guarding $t_S$ on source $\neq$ deploy-warmup breaks the *trigger* rather than the cycle, and it is the root-cause fix, because the defect it removes is a genuine epistemic gap: the reliability agent cannot currently tell pressure its own ecosystem produced from pressure caused by a customer-facing incident. It needs a colored net (Jensen and Kristensen, 2009) and a real provenance channel, which is the engineering cost. *A post-deploy cooldown* of rate $\mu$ leaves the harmful path reachable and merely unlikely, at probability $(\lambda_S/(\lambda_S + \mu))^N$ by (5). The first two are distribution free, in the sense of §4.5: they hold for every positive rate assignment, including ones nobody estimated. The third holds only for the rates one happened to estimate, and its protection *weakens as* $N$ *falls*, which is to say exactly when the platform is already degraded. That asymmetry, rather than any difference in implementation effort, is the reason to prefer a structural repair where one exists.

**A screening procedure, and a test others can run.** The results compose into a layered analysis that avoids the global state space unless necessary: expose the interface places (messages, shared resources, permissions, memory writes, handoffs, approvals); build $G_A$ and compute its strongly connected components, which is linear; compute the interference blocks $J_{ij}$, one matrix product per agent pair; then run the screen (4), a linear program by Thm. 2 and often exact by Prop. 5. Only for surviving candidates build a reachable set, and only for confirmed hazards run the stochastic analysis. Breaking all hazardous cycles with the fewest regulated interface transitions is then a hitting-set problem whose solution says where monitors, approval gates and rate limits go; since regulation only removes firing sequences, such a repair never introduces a new hazard, though E6 shows what it can cost in progress, and Appendix E works four further compositions outside infrastructure. The E4 census is the part others can rerun today: it needs no agent, no model and no execution, only a benchmark's evaluators and a map from its API surface to state. For any agentic benchmark asserting an exact state-change footprint, compute $J_{ij}$ over ordered task pairs and report the preserved fraction. The theory predicts it will be small, and that pairs sharing *no* application will break more often than pairs that share one. A benchmark reporting that number beside its accuracy would be saying something about shared-environment behavior that no current leaderboard does.

**Limitations, and what follows.** A net is only as good as its event abstraction: if an agent misclassifies a message as verified, the net operates correctly on an incorrect mark, so this analysis is orthogonal to semantic assurance rather than a substitute for it. The screen can over-report, Thm. 3 is sufficient rather than necessary, plain place-transition nets are coarse, and emergence is baseline dependent, which is why we state the baseline throughout. On the benchmark side the AppWorld library is validated to recall $0.93$ and not to $1$, E4 composes agents two at a time and measures an isolation assumption rather than operational severity, E6 restores $A$ by serializing $B$ out of the way, and the cross-app arcs were recovered from recorded runs, so a coupling no published agent triggered is one we do not have. Four questions follow: characterize $L_{\|} \setminus L_{\otimes}$ without constructing the global state space; extend Prop. 5 to a necessary and sufficient condition; decompose $\Delta R_H$ into per-interface contributions; and learn the net from execution logs with the coverage guarantee the present analysis lacks. The agents of §3 are not hypothetical, each will arrive with its own guardrail, and Prop. 1 says those guardrails do not add up. What the analysis recommends is concrete: an agent should be approved together with an interface contract naming what it emits, what it consumes, and the companions for which its certificate remains valid. Until then, a score earned alone is a claim about a world the agent will not be deployed into.

### THE APPWORLD COMPOSITION, AND THE E6 RESULTS, DRAWN

Figure M7 is the picture §5 refers to and has no room for: the two-transition cycle that an ordinary shopping task and an ordinary buying task close between them, and the writes `place_order` makes into two *other* apps. Table M9 is the E6 scoreboard, including the column that prices the repair. Both

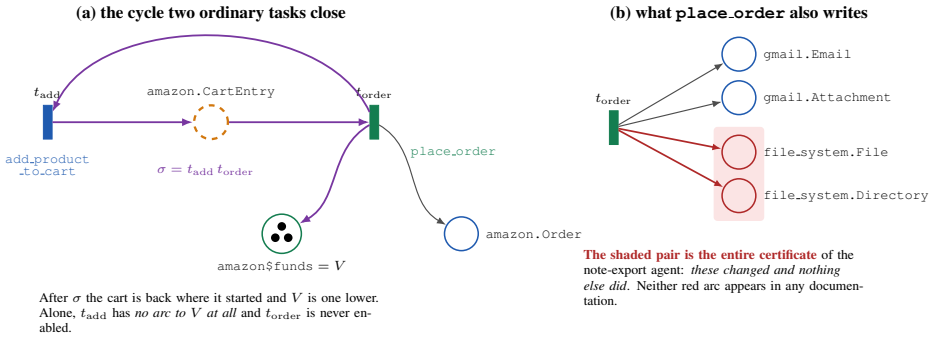



Figure M7: The AppWorld composition of E4 to E6. **(a)** A shopper adds to the cart and a buyer orders it; the cart is an interface place, so after both fire it is back where it started and the money is not. $\sigma$ is therefore the regenerative dissipative interaction cycle of §4.2 with $|\sigma| = 2$ and $d = 1$, and the shortest witness has $2K$ firings (Table L2). Neither agent can do this alone. **(b)** The same transition writes models belonging to two *other* apps, because the order files a receipt and mails a confirmation; four of six arcs are drawn. The red pair lands inside the declared footprint of an agent whose only job is to export notes to a backup directory, and E6 runs exactly that pair live. A certificate true of its own agent, falsified by an agent sharing nothing with it but an account, is this paper in one picture.

Table M9: E6, two agents on one AppWorld account, scored by AppWorld's own evaluator. $A$ exports the user's notes to a backup directory and its evaluator declares that a correct run changes `file_system.File` and `file_system.Directory` and nothing else; $B$ buys something, and placing an order files a receipt. Cutting the fusions $J_{ij}$ flags restores $A$ exactly, where cutting only the models $A$ owns does not. The refusal count is the price.

| **arm** | $A$ **passes** | $A$ **requirements** | $B$ **orders** | $B$ **refused** |
|---|---|---|---|---|
| isolated (AppWorld's own protocol) | 3/3 | 24/24 | 0/3 | 0 |
| one world, unmediated | 0/3 | 6/24 | 3/3 | 0 |
| screen: $B$ may not write $A$'s models | 0/3 | 21/24 | 0/3 | 3 |
| **screen:** $J_{ij} = 0$ | 3/3 | 24/24 | 0/3 | 6 |

are reproduced here rather than in §5 purely for space, and nothing in the body's argument depends on reading them.

### E5 AND E6B AS THEY WERE WRITTEN AT FULL LENGTH

The two passages below are the versions §5 compresses, reproduced so that nothing in the argument depends on the compression.

**E5, in full.** A net library is a claim about an API's semantics, and AppWorld lets that claim be tested, since it publishes trajectories together with the SQL those calls applied. Across 4,601 scored trajectories and 1.65 million calls the library names the exact set of changed models 62.9% of the time; held out over the three architectures not used to correct it, 63.1% exact with recall 0.93 and precision 0.70 (Table L6). Recall is the important direction, because a library that over-approximates what an agent writes still bounds the interference while one that misses a write does not, and 0.93 is not soundness: about seven percent of recorded model writes are absent from the prediction, and every E4 number inherits that gap. Building the library that way turned up the sharpest thing this benchmark has to say. **Twenty calls write models belonging to a different app, and not one says so in its own API documentation** (Table L4). AppWorld's paper describes the worlds as coupled and mentions effects of this kind, so the behavior is not secret; the point is that the coupling is absent from the per-API contract an implementer reads, which is the artifact any per-agent analysis is built from. Placing an Amazon order files a receipt into the file system and mails a confirmation with the receipt attached, the right-hand half of Figure M7, and nine of the twenty are documented `GET` calls that write a file, so any analysis classifying calls by HTTP method puts them on the wrong side of the interface. Recomputing the census against what the realizing APIs actually write rather than what the

tasks declare, the preserved fraction falls from $21.0\%$ to $5.7\%$: three quarters of the pairs that looked compositionally safe are not, and the coupling that breaks them is invisible in the documentation and visible only in the state.

**E6b, specified but not run: the language-model arm.** The complementary experiment replaces both executors with language-model agents and measures a rate rather than a mechanism: for $n$ task pairs with $J_{ij} \neq 0$ and $n$ with $J_{ij} = 0$, report AppWorld's task-goal-completion for $A$ in three arms (isolated, shared unmediated, and shared under the $J_{ij} = 0$ screen) together with how often $B$ in fact emits a call whose effects leave $A$'s declared set. The prediction is falsifiable: degradation in the shared arm should be confined, up to the model's own error rate, to the $J_{ij} \neq 0$ pairs, and the screen should recover the isolated score on those pairs at the cost of $B$'s refused calls. Such an arm would add the *frequency* with which real agents trigger a flagged coupling; it would not add the mechanism, which AppWorld's own evaluator already establishes on runs it scored.

### The stochastic layer as it was written at full length

Structure says whether a hazard is reachable; deployment also needs to know how likely and how fast. Give each timed transition a rate $\lambda_t(M) > 0$ with exponential firing time; enabled transitions race and the marking process is a continuous-time Markov chain (Ajmone Marsan et al., 1995). With $\tau_{\mathcal{B}} = \inf\{\theta : M(\theta) \in \mathcal{B}\}$ and $R_H = \mathbb{P}(\tau_{\mathcal{B}} \leq H)$, structure decides whether this can be nonzero at all, for every horizon and *every* positive rate assignment: $\mathcal{R}(\mathcal{N}, M_0) \cap \mathcal{B} = \varnothing \Rightarrow R_H = 0$, a fact we call *distribution freedom* and use twice below. Comparing against an uncoupled baseline $\mathcal{N}_{\otimes}$ gives the *composition-induced risk* $\Delta R_H = R_H(\mathcal{N}_{\|}) - R_H(\mathcal{N}_{\otimes})$, which stays meaningful when the baseline risk is exactly zero, as it is here. With rates $\lambda_D, \lambda_S, \lambda_C$ on deployment, mitigation and reroute and no competing transition the outage is certain, $\mathbb{P}(\tau_{\mathcal{B}} < \infty) = 1$, and

$$\mathbb{E}[\tau_{\mathcal{B}}] = \frac{1}{\lambda_D} + \frac{N}{\lambda_S} + \frac{N-1}{\lambda_C} \qquad \text{while with a stabilizer of rate } \mu, \qquad \mathbb{P}(\text{outage}) = \Big(\frac{\lambda_S}{\lambda_S + \mu}\Big)^N.$$

The left formula carries an uncomfortable reading: *faster agents do not make this system safer, they make it fail sooner.* A team that halves its mean time to mitigate, the headline metric for a reliability agent, halves the $N/\lambda_S$ term, so improving the metric moves the system toward the hazard, and by distribution freedom it does not change whether the hazard is reached. The stabilizer $t_{\text{cool}}$ on the right is a post-deploy cooldown, a mitigation rate limit, or a human confirmation that absorbs the CPU mark and ends the round; each round is then a race and the outage needs $N$ consecutive bad outcomes. Once $\mu > 0$ the chain has two absorbing outcomes, so $\mathbb{E}[\tau_{\mathcal{B}}]$ is infinite and is not the quantity to report; the finite timing quantity, and the one §5 sweeps, is the time to absorption either way, $T_{\text{abs}} = \min(\tau_{\mathcal{B}}, \tau_{\text{safe}})$. The thrash cycle competes for the same mark, and the answer to how much worse it makes things is exactly none.

The proof is two lines on the embedded jump chain: $p = a + cp$ with $a = \lambda_S/\Lambda$, $c = \lambda_0/\Lambda$, $\Lambda = \lambda_S + \mu + \lambda_0$, and $\mathbb{E} = 1/\Lambda + c(\theta + \mathbb{E})$ for the time, the $c\theta$ term being what a purely geometric count of excursions omits (Appendix H). The theorem separates the two cycles cleanly: the meltdown cycle determines *whether* the system fails, the thrash cycle only *when*. It also explains a familiar symptom, since the two cycles share the CPU mark: an operator sees an outage on some days and a release that will not stick on others, with no code change in between, and the split is governed by the relative rates of two mitigation policies that were never specified jointly. The design consequence is the contrast between structural and stochastic intervention. Removing the cross-agent cycle makes the outage unreachable under every parameterization; a fast cooldown leaves the harmful path reachable and merely unlikely, under the rates one happened to estimate, and by (5) that protection weakens as $N$ falls, exactly when the platform is already degraded. Only the first survives conditions nobody anticipated.

### E7, in full

**E7: an independently built taxonomy, and what it can and cannot support.** MAST (Cemri et al., 2025) annotates $1{,}642$ traces from ChatDev, MetaGPT, AG2, Magentic-One, OpenManus, HyperAgent and an AppWorld crew against fourteen failure modes its authors derived by reading the traces, grouped into three categories of their own. One of those they named *inter-agent misalignment*, so treating it as the interface category reads their label rather than imposing ours. Four records carry

a null annotation for every mode and are dropped rather than read as clean runs, leaving $1{,}638$ usable traces, of which $1{,}237$ carry at least one failure label; every rate below is over those $1{,}237$.

Of the $1{,}237$ traces, $174$ ($14.1\%$) carry an inter-agent mode and no specification-category mode at all: no local fault was annotated and the system failed anyway (Table L5). That is weaker than saying every agent met its specification, since an absent label is not a verified absence, and it is the strongest reading the data supports. Restricting to the four modes whose definitions cannot be stated about a single agent gives $6.8\%$; on the human-annotated traces it is $4/18$, or $22.2\%$. The three figures bracket the claim from both sides.

The data do *not* support a quantitative relationship between interface-failure frequency and the topology of the agent interaction graph. Prop. 3 predicts that interface failure tracks coupling, so we recovered $G_A$ per trace, taking a turn by $i$ followed by a turn by $j$ as an edge. It does not track it: across the seven systems Spearman is $0.08$ against the number of roles and $0.05$ against the share of traces with a reciprocal pair, and per trace a cyclic $G_A$ gives $0.74$ against $0.64$ for an acyclic one. With seven systems these coefficients are descriptive and we draw no inference from them. Two explanations are live and we can separate them only partly. Consecutive speech is a coarse proxy for communication, since it records who talked after whom, not who read what, so a framework routing through a shared store looks acyclic when it is not. And the labels are noisy: on $89$ traces in which exactly one role ever speaks, the judge still records *ignored other agent's input* $16$ times and *fail to ask for clarification* $15$ times, which cannot happen whatever the proxy does. Against the shipped human annotations we measure $54\%$ agreement on 213 labels, the judge flagging three times as many modes as the annotators. That figure is not comparable to the $94\%$ accuracy and $\kappa = 0.77$ MAST reports for its own judge: it is a different measurement, over the 17 shipped human traces that also appear in the judged corpus, with the human label the majority of three annotators and the earlier drafts mapped by mode name rather than number (Appendix L). We report it as what our mapping yields, not as a correction to theirs. MAST supports the qualitative claim that a large share of real multi-agent failure sits at the interface; it does not support a quantitative test of what predicts it, and we do not report one.

### Threats to validity, in full

**Threats to validity.** E1 to E3 are exact analyses of formal models, not measurements of a deployed system, and support exactly one claim: that the theory is correct and computable. The benchmark arms carry different risks. E4 is exact *given the library*, and the library is the assumption: its arcs are a reading of AppWorld's API documentation corrected against the database, and E5 measures how good that reading is rather than asserting it. The correction was fitted on one of four architectures and the figures are held out on the other three, so the recall is not self-congratulatory; but a library validated to recall $0.93$ is not a library validated to 1, and every E4 number inherits that. E4's pairs are also *pairs*: two agents at a time, not a workload. E6 is three task variants driven by a scripted policy; what it establishes is not statistical but mechanical. E7 rests on an LLM judge whose labels we show to be unreliable at the granularity the finer claim needs, which is why that claim is reported as a negative. All three artifacts are simulated, and none is a production incident.

## N Protocol and full numerical results for E1 to E3

**Environment.** Every number in this paper was produced on one machine, an Apple M3 under macOS 26.5.1, with CPython 3.14.0, NumPy 2.5.1, SciPy 1.18.0 and SymPy for the invariant nullspaces. Linear programs are solved by `scipy.optimize.linprog` with `method="highs"`, the HiGHS dual simplex and interior point solver bundled with SciPy 1.18.0. Reachability, absorption and screening are deterministic and use no random number generator, so no seed is involved; the only sampled quantity anywhere in the paper is the pair sample of the block check in E4, which uses `numpy.random.default_rng(0)`. Wall-clock columns are from this machine and are the only numbers that will not reproduce exactly. The accompanying repository records the commit that generated each result file.

This appendix covers the three experiments on the nets of §3. The four benchmark arms are a different kind of thing, because their nets have to be built from somebody else's artifact before anything can be measured, so their construction, their protocol and their tables are together in Appendix L.

Table N10: E1, exact enumeration for every $N$ from 2 to 12. Each component is tested in isolation with a hand-injected stimulus on its input interface, which is the most favorable possible reading of component testing: without the injection the reliability agent's trigger $U$ has no producer and $t_S$ is dead. The composed reachable set is $4N + 3$ markings and the shortest witness is $2N$ firings, in every case.

| | composition | | isolated, $\min M(H)$ reached | | | |
|---|---|---|---|---|---|---|
| $N$ | markings | witness | $A_D$ | $A_S$ | $A_C$ | any reaches $\mathcal{B}$? |
| 2 | 11 | 4 | 2 | 1 | 2 | no |
| 3 | 15 | 6 | 3 | 2 | 3 | no |
| 4 | 19 | 8 | 4 | 3 | 4 | no |
| 5 | 23 | 10 | 5 | 4 | 5 | no |
| 6 | 27 | 12 | 6 | 5 | 6 | no |
| 7 | 31 | 14 | 7 | 6 | 7 | no |
| 8 | 35 | 16 | 8 | 7 | 8 | no |
| 9 | 39 | 18 | 9 | 8 | 9 | no |
| 10 | 43 | 20 | 10 | 9 | 10 | no |
| 11 | 47 | 22 | 11 | 10 | 11 | no |
| 12 | 51 | 24 | 12 | 11 | 12 | no |

Table N11: E3, exact continuous-time Markov chain analysis at $\lambda_D = \lambda_S = \lambda_C = \lambda_{S,\mathrm{rb}} = 1$, $N = 5$. The closed form of Eq. (5) is exact; adding the thrash cycle changes the outage probability by nothing and the expected time to absorption by a factor between 1.33 and 2.00, as Theorem 4 requires.

| | $\mathbb{P}$(outage) | | | $\mathbb{E}[T_{\mathrm{abs}}]$ | | |
|---|---|---|---|---|---|---|
| $\mu/\lambda_S$ | closed form | meltdown | with thrash | meltdown | with thrash | factor |
| 0.00 | 1.00000 | 1.00000 | 1.00000 | 10.00 | 20.00 | 2.00 |
| 0.25 | 0.32768 | 0.32768 | 0.32768 | 6.05 | 11.43 | 1.89 |
| 0.50 | 0.13169 | 0.13169 | 0.13169 | 4.34 | 7.82 | 1.80 |
| 1.00 | 0.03125 | 0.03125 | 0.03125 | 2.91 | 4.84 | 1.67 |
| 2.00 | 0.00412 | 0.00412 | 0.00412 | 1.99 | 2.99 | 1.50 |
| 4.00 | 0.00032 | 0.00032 | 0.00032 | 1.50 | 2.00 | 1.33 |

**Implementation.** A compact pure-Python place/transition net library: explicit breadth-first reachability with shortest-witness extraction, exact absorption analysis by solving two dense linear systems on the reachable continuous-time Markov chain, and the screen of (4) solved with a standard interior-point linear-programming routine. Net definitions, analysis code, and raw result files accompany the submission.

**What is exact and what is not.** E1 and E3 involve no sampling and no approximation: reachability is complete enumeration, and the stochastic quantities are obtained by solving linear systems rather than by simulation, so the only error is floating point, bounded at $3 \times 10^{-16}$ on the reported comparisons. E2 timings are single-core wall-clock on a laptop and indicate scaling, not absolute performance; the LP time at $m = 1$ is larger than at $m = 2$ because it includes one-time solver setup. Total runtime is under one minute except for the $m = 8$ enumeration, which dominates at 28 seconds.

**E1 protocol.** The composition fuses the interface places $U$, $K$, $V$, $B$. Each component is enumerated alone from its own initial marking with one mark injected on its input interface, which is strictly more generous than real component testing: without the injection $t_S$ is dead in $\mathcal{N}_S$, so an isolated reliability agent cannot be exercised at all. Reported $\min M(H)$ is the minimum over the component's entire reachable set.

**E2 protocol.** $m$ service groups, each a full copy of the three agents with its own interface places, all consuming from one shared pool of $N = 6$ healthy instances. Enumeration is complete BFS; the screen solves (4) once on the composed incidence matrix. The screen returns support $\{t_S^{(0)}, t_C^{(0)}\}$ in every instance, which Proposition 5 certifies executable because that support induces a simple circuit.

Table N12: E2, exact reachability against the linear-program screen of Theorem 2, on a fleet of $m$ service groups sharing one instance pool ($N = 6$). Enumeration grows geometrically while the screen stays flat and returns the minimal hazardous cycle in every case. Times are single-core seconds.

| $m$ | places | transitions | reachable markings | enumeration (s) | screen (s) | screen support |
|---|---|---|---|---|---|---|
| 1 | 7 | 5 | 27 | 0.0002 | 0.0060 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 2 | 12 | 10 | 104 | 0.0016 | 0.0007 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 3 | 17 | 15 | 400 | 0.0095 | 0.0006 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 4 | 22 | 20 | 1,536 | 0.0493 | 0.0008 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 5 | 27 | 25 | 5,888 | 0.2452 | 0.0009 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 6 | 32 | 30 | 22,528 | 1.138 | 0.0009 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 7 | 37 | 35 | 86,016 | 5.369 | 0.0016 | $\{t_S^{(0)}, t_C^{(0)}\}$ |
| 8 | 42 | 40 | 327,681 | 28.26 | 0.0013 | $\{t_S^{(0)}, t_C^{(0)}\}$ |

**E3 protocol.** $N \in \{3, 5, 10\}$ and $\mu/\lambda_S \in \{0, \frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$, with $\lambda_D = \lambda_S = \lambda_C = \lambda_{S,\mathrm{rb}} = 1$; 18 settings in total. Model A is the meltdown net alone (12 to 33 states); model B adds the thrash cycle (19 to 54 states). We solve for absorption probability and expected absorption time in both, and compare against Eq. (5). Table N11 reports $N = 5$; the $N = 3$ and $N = 10$ sweeps are in the accompanying results file and agree identically, including the $N$-independence of the time-inflation factor noted in §5.