Intelligence Economy
INSTITUTE

Overstated Unencumbered Mint Capacity

A Case Discussion on Asynchronous Claim and Reserve State Misclassification in Fiat-Backed Stablecoins

Zakaryae Boudi
Intelligence Economy Institute

Working Paper - June 2026

Abstract

This is a case discussion of a stablecoin failure mode that requires no stolen mint key, no falsified bank statement, no broken ERC-20, and no direct contract exploit. The system overstates how much it can safely mint because *claims and reserves are measured on incompatible finality clocks*, and because the authorization join between them is left unguarded.

The reason this case is worth studying is that the failure lives in none of the components. Each part is correct in isolation: the token contract is sound, the bank report is honest, the bridge verifies its messages, the attestation is accurately scoped. **The failure lives in the seams between them**, in the joins where one ledger's notion of "done" is handed to another ledger that means something different by it. The bug is not in any box; it is in the wiring between boxes.

The thesis is narrow: *form is not finality*. Received is not available, burned is not paid, bridged is not finalized, frozen is not extinguished, attested is not continuous, and issued supply is not total claims. Stablecoins fail inside those inequalities, and each inequality is a seam.

Beyond the per-seam predicates, the deepest commitments of this analysis are **meta-properties**. There is *no global cut*: one of the four ledgers belongs to a bank the issuer can only poll and that can revise its own postings, so authorization rests on a local cut plus dated, revisable belief, and a mint against in-window belief is *provisional* until proven. The controls must *compose* within a stated latency budget, and the honest consequence is that instant settlement and post-hoc reserve verification are incompatible, so one must prefund. Prevention *probabilistically fails*, so a recovery state machine with a pre-declared breach-resolution order is required. Halting *mint* is the issuer's safe discretion; throttling *redemption* may be unlawful. The controls' guardians are *trusted not to collude*, so the collusion threshold and the single points of trust are named rather than assumed. And the issuer does *not fail alone*: the model carries systemic-correlation terms, because every canonical stablecoin death, a failed reserve bank, a peer's depeg, a shared oracle, was systemic.

This document is a technical case discussion intended for stablecoin issuers, auditors, and protocol-security engineers. It describes a class of control failure and its remediation; it is not legal, accounting, or financial advice, and the worked figures are illustrative. A note on scope: the case is the overstated-mint-capacity finding, but the remediation it requires is broad enough that the document doubles as a control model, spanning claim/reserve state synchronization, a seam-contract methodology, a recovery framework, config-plane governance, and a systemic overlay. Readers wanting only the finding can stop after §14; the remainder is the model the finding implies.

Executive summary. The system overstates mint capacity because it treats external reserve *observations* as final reserve *capacity*, and treats internal claim-*form* transitions as claim *extinguishment*. The resulting mint authorization can consume capacity that is either not yet available or already encumbered by unresolved claims. Equivalently: reserve observations become mint authority before passing the eligibility, availability, encumbrance, horizon, currency, haircut, and staleness gates, while claim-form transitions are treated as obligation extinguishment before the underlying economic/legal claim is settled, transferred, extinguished, or reclassified.

Contents

1 How a Fiat-Backed Stablecoin Is Supposed to Work **3**
2 The Finding **4**
3 Threat Model, Who Pulls the Seams Apart **4**
4 Architecture Under Discussion **5**
5 The Consistency Model, No Global Cut Exists **5**
6 Seam Contracts, Every Join Must Be Typed **7**
6.1 Why this list is closeable 7
7 The Real Object: Mint Capacity by Horizon and Currency **8**
8 The Four Reconciled Ledgers **8**
9 Claims: Legal Liability vs. Reserve-Coverage Obligation **9**
10 Reserves: The Seven Gates **10**
11 The Control Ledger: Authorization Is Not Finality **10**
12 Root Cause (Two Independent Defects) **11**
13 Worked Example, Decomposed by Root Cause **12**
14 Worked Stress Over the Full Matrix **13**
15 Exploit Routes and Severity **14**
16 The Weak Dashboard **15**
17 The Objects and the Capacity / Liability Models **16**
18 Invariant 1, No Orphan Lifecycle Transitions **17**
19 Invariant 2, The Priority Waterfall **17**
20 Invariant 3, Snapshot Freshness and Single-Use Binding **18**
21 Invariant 4, Liveness, and the Asymmetry of Halting **18**
22 The Latency Budget, Do the Controls Compose? **19**
23 Lifecycle Invariants **20**
24 Domains, Multi-Chain Control, and Valuation-Source Integrity **20**
25 Attestation: Scope vs. Inference **20**
26 Buffers Are Not a Substitute for Correctness **21**
27 The Config Plane, Governing the Parameters (T5) **21**
28 The Recovery State Machine, Prevention Fails; Then What? **21**
29 Trust Assumptions and Collusion, Name the Threshold **22**
30 Systemic Correlation, The Issuer Does Not Fail Alone **23**
31 The Assumption Ledger **23**
32 The Fix **25**
33 What an Audit Must Cover **25**
34 Final Lesson, From Seams to Meta-Properties **26**

1 How a Fiat-Backed Stablecoin Is Supposed to Work

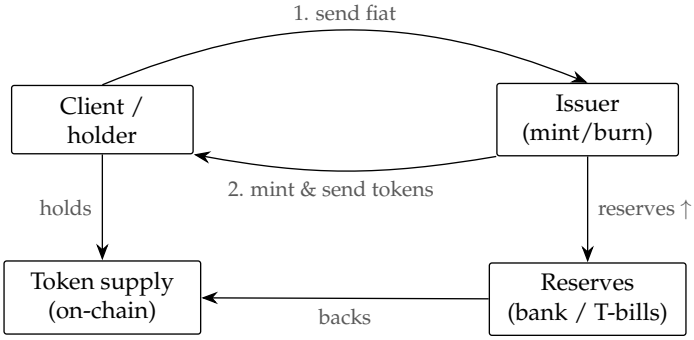
Before examining how the model breaks, it helps to state how it is *meant* to behave when everything goes right. The rest of this document is, in effect, a study of the gap between this clean picture and operational reality, so the clean picture is worth drawing first.

A fiat-backed stablecoin is a promise with three parts: *one token is worth one unit of fiat*; *anyone holding a token can redeem it for that fiat*; and *every token in circulation is backed by reserves the issuer actually holds*. The issuer's job is to keep those three promises simultaneously true at all times. There are only two operations that change the token supply, and each is supposed to be paired one-to-one with a movement of real money.

Mint (money in, tokens out). A client sends fiat to the issuer's bank. Once that money is genuinely in hand, the issuer creates ("mints") an equal value of new tokens and sends them to the client. Supply goes up; reserves go up by the same amount. The peg is preserved because the new tokens are fully backed the instant they exist.

Redeem (tokens in, money out). A holder sends tokens back to the issuer. The issuer destroys ("burns") them and pays out the equal value of fiat. Supply goes down; reserves go down by the same amount. The peg is preserved because the disappearing tokens release exactly the reserves that backed them.

Hold and transfer (in between). While tokens circulate, traded, sent, used in applications, bridged to other chains, the issuer does nothing to supply or reserves. Transfers move tokens between holders but never change the total owed or the total held.



The intended invariant, in one line: *at every moment,* `reserves_held` $=$ `tokens_outstanding` $\times$ *peg. Mint raises both sides together; redeem lowers both sides together; holding and transferring move neither. If that equality held continuously and exactly, there would be no story to tell.*

Figure 1. The nominal lifecycle. Two supply-changing operations (mint, redeem), each paired one-to-one with a reserve movement, plus a do-nothing middle (hold, transfer). The promise is that backing tracks supply at all times.

Where the clean picture quietly assumes too much. The nominal model treats four words as if they were instantaneous and certain: money is *received*, reserves are *available*, a burn means the holder is *paid*, and a token bridged elsewhere is *settled*. In a diagram these are single arrows. In reality each is a process that takes time, can be reversed, and is recorded in a different system on a different clock, the issuer's bank does not settle on the same timeline as the issuer's blockchain, which does not settle on the same timeline as a bridge or a redemption rail. The simple equality `reserves` $=$ `supply` $\times$ `peg` is therefore not one equation evaluated at one instant; it is a claim about several independently-updating ledgers that the issuer is trying to hold in agreement.

This document is about what happens in the gaps between those arrows, the moments where money *looks* received but has not cleared, or a token *looks* settled but the cash has not yet moved. Everything that follows is a careful account of those gaps and how to keep the three promises true despite them.

2 The Finding

Primary finding: overstated unencumbered mint capacity. Not necessarily fraud. Not necessarily immediate insolvency. Not necessarily a smart-contract bug. But if this overstated value feeds mint authorization, it becomes critical, because issuance is irreversible in a way that the underlying reserve event is not.

The naive control most issuers actually run is:

```
reserves >= totalSupply
```

This is necessary and nowhere near sufficient. It is a screenshot with its assumptions hidden inside it. A stablecoin security model must track supply, claims, reserves, controls, legal status, domains, evidence, *and the finality clock attached to each*, and it must guard the moment where a reserve snapshot is converted into mint authority.

Each row is a seam*: two systems agree on the word and disagree on its meaning.*

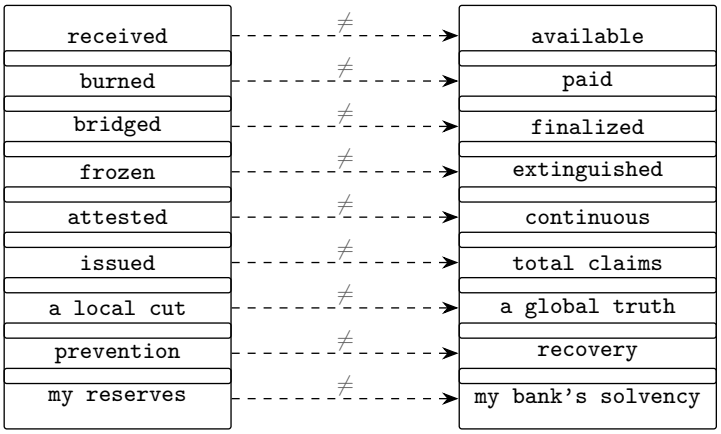


Figure 2. The seam catalogue. The first six rows are intra-issuer finality mismatches; the last three are meta-property seams (§5, §28, §30). Closing a seam means specifying the contract under which the left term is allowed to become the right term.

3 Threat Model, Who Pulls the Seams Apart

A control claim is meaningless without a named adversary and a capability set. This document defends against five distinct threat classes, because each one stresses a different seam, and conflating them is itself a common review failure.

T1, Finality-arbitraging client (active, external, authorized).
A verified institutional participant who can observe or infer bank cutoffs, mint-release policy, redemption batch windows, bridge finality, attestation cutoffs, and pause latency, and times actions to exploit the gaps. No system compromise; only timing.

T2, Correlated macro stress (passive, no adversary).
A redemption wave coinciding with delayed bank settlement, bridge congestion, and stale internal reconciliation. Control bugs do not require villains.

T3, Colluding or impaired custodian / bank (trusted-party failure).
A custodian whose "current balance" is honest but provisional, recallable, under lien, or subject to correspondent delay or capital control.

T4, Compromised or stalled bridge / domain authority.
A destination domain that releases liquidity before global cross-domain claim reconciliation, or a validator set whose finality guarantees degrade under load.

T5, Privileged insider / compromised governance (config-plane adversary).
An actor who breaks no runtime invariant but instead changes the *parameters* that make the invariants safe: relaxing `max_snapshot_age`, widening a domain limit, whitelisting an ineligible asset, loosening a haircut. The cheapest attack on the whole system. Treated in §27.

A control that defends T1 (timing) may do nothing for T3 (custodian quality), T4 (domain reconciliation), or T5 (parameter authority). Any claim that the architecture is robust is scoped to these five classes and is false outside them. An undefined threat model is the qualitative twin of an undefined coverage ratio, a reassuring statement with the seam hidden inside it.

4 Architecture Under Discussion

A fiat-backed stablecoin issuer running: L1 ERC-20 supply; issuer-recognized L2 / multi-domain supply; a mint/burn controller; an institutional portal; a treasury ledger; bank / custodian accounts; a redemption workflow; bridge accounting; reserve attestations; and compliance controls.

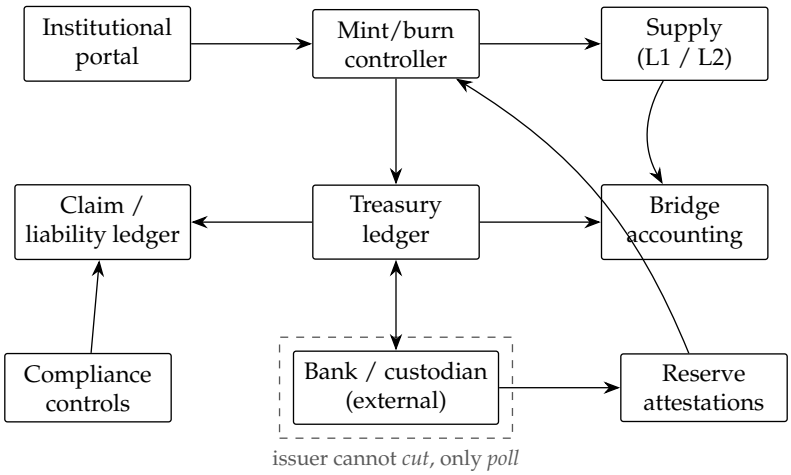


Figure 3. The components. Each can be individually correct, the ERC-20 sound, the bank report honest, the bridge messages verified, the attestation accurately scoped, and the system can *still* overstate mint capacity. The failure lives in the arrows, not the boxes. The bank sits outside the dashed boundary: the issuer can only poll it, a fact that becomes decisive in §5.

5 The Consistency Model, No Global Cut Exists

This section comes before everything else technical because it determines whether anything that follows means what it appears to mean. Every invariant in this document is a predicate over the state of four ledgers. A predicate is only as trustworthy as the *consistency model* of the state it reads. An invariant checked against a snapshot that never coherently existed proves nothing.

It is tempting to ask for a "consistent cut across all four ledgers," but that is not achievable, and seeing why is instructive. The four ledgers are not symmetric. Supply (chains), claim

(issuer records), and control (mint pipeline) are **owned by the issuer** and can in principle be cut together. But the ground truth for the **reserve** ledger lives in **bank and custodian systems the issuer does not control**: the issuer can only *poll* them, subject to their cutoffs, their finality classes, and, critically, their right to *revise* a posting after the fact. You cannot take a synchronous cut that includes a counterparty's books.

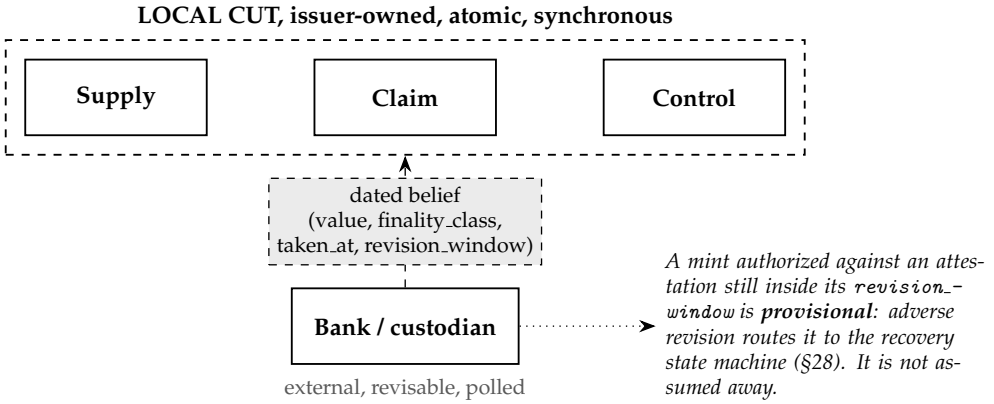


Figure 4. There is no global cut. The issuer-owned ledgers form a local cut that can be made atomic; the bank is an oracle outside it, contributing dated, revisable belief (consistent with §24). Every downstream invariant therefore holds "under beliefs as of t, subject to external revision," not "at instant t."

This lens also locates the two independent defects that §12 (Root Cause) dissects in full. Named there as Defect A and Defect B, both are write-ordering and atomicity failures, and the consistency model shows precisely where each one lives, and why one is far harder to cure than the other:

- **Defect A**, a wire that credits capacity before the funds are available, is the issuer treating an *external belief* as if it were inside the cut. *Not* curable by a database transaction; curable only by respecting finality class and revision window, and by a recovery path for adverse revision.
- **Defect B**, a burn that deletes the liability before payout finalizes, is two *issuer-owned* writes not placed in one transaction. Curable by a true atomic transaction, because both sides are inside the local cut.

```
authorize_mint(...) valid against:
  local_cut      = (supply_v, claim_v, control_v)   # issuer-owned, mutually consistent
  external_beliefs = { (attestation_i, finality_class_i, taken_at_i, revision_window_i) }

  1. LOCAL CUT: a "latest from each ledger" read is INVALID for authorization.
  2. ORDERED, ATOMIC CLAIM TRANSITIONS: supply-decrement and liability-creation
     on burn occur in one atomic unit (both inside the local cut).
  3. EXTERNAL BELIEFS ARE DATED AND REVISABLE: never a cut-synchronous fact;
     provisional authorizations are flagged.
  4. MONOTONIC SEQUENCE + STALENESS BOUND: strictly increasing per [h,c];
     if now - taken_at > max_cut_age[h] -> HALT.
```

The distinction is the whole point: **what the issuer owns can be made atomic; what the issuer merely observes can only be dated, bounded, and recovered from.** An auditor told "we take a consistent global snapshot" should treat that claim as false on its face, because one of the four ledgers belongs to someone else.

6 Seam Contracts, Every Join Must Be Typed

Naming a seam is not specifying it. A seam between two ledgers is an interface, and an interface has a *contract*: the type and unit of what crosses it, the finality class required, whether the transfer is idempotent, and the failure semantics if the receiver rejects.

```
seam_contract = {
  from_ledger, to_ledger, payload_type, unit, currency,
  required_finality_class,   # which bank/bridge/legal class is accepted
  idempotency_key,           # so retries do not double-apply
  ordering_guarantee,        # FIFO per key | causal | none
  on_reject,                 # hold | refund | retry-with-backoff | escalate
  max_staleness, evidence_id }
```

Table 1. Worked seam contracts. The contracts are deliberately asymmetric: each join requires a different finality class, reject behaviour, and idempotency key.

Seam	Required finality class	On reject	Idempotent?
bank-confirmation → control	`available/irrevocable` (not `current`)	hold mint; don't count	yes, by `bank_confirmation_id`
claim → reserve (coverage)	counted state for horizon h	exclude from due claims	yes, by `claim_id`
burn → claim	n/a (creates obligation)	block burn if unrecordable	yes, by `redemption_id`
bridge-burn → x-domain claim	`message proven` + challenge expired	refund/retry holds claim	yes, by `message_id`
reserve-snapshot → mint	fresh, single-use, monotonic	reject mint	**no**, single-use

6.1 Why this list is closeable

Enumerating seams one at a time invites the question an audit firm asks immediately: *what is your argument that no further seam exists?* A list assembled by inspection is a changelog, not a specification, and it can never answer that question. The fix is to stop enumerating and instead **derive**.

> **Generative principle.** A seam exists wherever state crosses between two ledgers *and the destination consumes it under a stronger or different semantic requirement than the source guarantees*, including finality, unit, currency, time horizon, legal status, claimant identity, valuation method, priority, or evidence scope. Equivalently: a seam is a tuple $(\mathsf{from}, \mathsf{to}, r)$ where the destination's requirement r along any of these dimensions exceeds or differs from what the source can guarantee.

The narrower "finality only" form is a special case. The broader principle matches the seam-contract schema (which already carries unit, currency, finality, idempotency, ordering, failure path, staleness, and evidence): each schema field is a dimension along which a destination can demand more than a source provides, and therefore a dimension along which a seam can arise.

	supply	claim	reserve	control
supply	′			•
claim		′	•	
reserve			′	•
control	•			′

A • marks a directed pair (from → to) carrying payload where the destination's requirement exceeds or differs from the source's guarantee along any *semantic dimension (finality, unit, currency, horizon, legal status, claimant identity, valuation, priority, evidence). The seam set is* exactly *these cells. A new seam can appear* only *via a governed change: a new ledger, a new payload path, or a new semantic dimension, each a config-plane event (§27), not a surprise.*

Figure 5. The seam-closure matrix. "Have we found all the seams?" becomes "is the set of ledgers, payload paths, and finality classes complete and frozen under governance?", a question that can be answered, audited, and re-checked when the architecture changes.

7 The Real Object: Mint Capacity by Horizon and Currency

The object that governs solvency is not a headline reserve number, not raw bank balance, and not `totalSupply`. It is `mint_capacity`$[h, c]$, capacity by settlement horizon h and currency c. Minting consumes capacity.

```
mint_capacity[h,c] = unencumbered_capacity[h,c]
                   - due_effective_claims[h,c]
                   - pending_commitments[h,c]
                   - stress_buffer[h,c]
```

The two indices are the whole point. A reserve is not simply present or absent; it is available *at a time* and *in a currency*. Cash that settles today (T+0) is a different asset from a Treasury bill that settles tomorrow (T+1), even at the same face value, because a claim due this afternoon cannot be paid with money that arrives next week. Likewise a surplus of euros does not cover a dollar claim that is due now. So capacity is not one figure but a grid of [horizon $\times$ currency] cells, and a mint draws down a *specific* cell: it consumes capacity of the right currency, available by the right time. Reporting a single blended number silently nets a T+1 euro surplus against a T+0 dollar shortfall, which is exactly the error the grid exists to prevent.

Reading the formula term by term, for each cell: start from `unencumbered_capacity`, the reserves that are genuinely free and usable for that time-and-currency slot (the output of the seven gates in §10); subtract `due_effective_claims`, everything actually owed in that slot; subtract `pending_commitments`, payouts already promised but not yet settled; and subtract `stress_buffer`, the cushion deliberately held back for correlated bad days. Whatever remains is the most that may be minted against that cell. Each subtrahend is itself a place where form can be mistaken for substance, which is why the following sections define each one precisely rather than trusting a single balance.

The bug is that the system computes this **too high**, for two structurally independent reasons that must never be reported as one number: it overstates the first term (counting reserves that are not yet available or already spoken for) and understates the claim terms (treating obligations as discharged before they are settled). §12 separates these as Defect A and Defect B.

8 The Four Reconciled Ledgers

Mint capacity is computed from four distinct ledgers, and the discipline of the whole system is keeping each one reconciled against the next rather than reading any one of them in isolation. Each ledger answers a different question, and between every adjacent pair sits an inequality that a naive system silently treats as an equality.

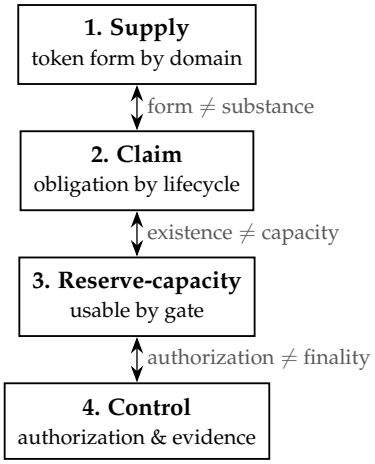


Figure 6. Four ledgers, each reconciled against the next. The inequalities between them are the seams the rest of the document closes.

The **supply** ledger records how many tokens exist, by domain and token form. The **claim** ledger records what the issuer actually owes, by obligation and lifecycle state. The **reserve-capacity** ledger records what backing is genuinely usable, gate by gate. The **control** ledger records what has been authorized and on what evidence. The three inequalities are the recurring traps: *form ≠ substance* (a token existing in some form is not the same as a settled, fully-backed obligation), *existence ≠ capacity* (a claim existing is not the same as reserves being available to meet it), and *authorization ≠ finality* (a mint being authorized against a snapshot is not the same as the underlying reserve event being final). Closing the gap at each arrow, rather than assuming it is zero, is what the invariants in the following sections enforce.

9 Claims: Legal Liability vs. Reserve-Coverage Obligation

"Claim" means *any* legal liability, contingent obligation, reserve-coverage obligation, operational commitment, or issuer-recognized claim under the issuer's terms, law, reserve policy, or bridge arrangement. The breadth is deliberate: anything that can require the issuer to part with reserves, now or on a condition, is a claim, regardless of whether an accountant or a court would yet call it a debt.

The reason the definition is wide is that two notions people assume are identical are not. **Legal liability** is what the issuer owes as a matter of law and contract. **Reserve-coverage obligation** is what the issuer's own reserve policy must stand ready to fund. These overlap heavily but neither contains the other, and the cases where they diverge are exactly where capacity is overstated:

- A **third-party wrapped token** may not be issuer legal debt, yet if the issuer recognizes it, it can still be a reserve-coverage obligation, a claim without being a liability.
- A **frozen balance** may not be immediately redeemable, yet it remains a valid claim; the freeze restricts transfer, it does not extinguish the obligation.
- A **pending bridge release** may be an operational obligation the issuer must honor without being a present legal debt on the balance sheet.

The practical danger is asymmetric. If something that *is* a claim for coverage purposes is filed under "not a liability" and therefore dropped from `due_effective_claims`, the claim term in the capacity formula shrinks and mint capacity rises by exactly that amount. **A claim misclassified as a non-claim is the raw material of overstatement**: it is the mechanism by which real obligations vanish from the number that authorizes new issuance. The safe default

is therefore inclusion, count it as a claim unless there is a documented, governed basis for excluding it, with the legal-versus-coverage status recorded explicitly rather than collapsed into a single yes/no.

10 Reserves: The Seven Gates

A reserve asset must pass **seven** tests before it can support mint capacity. Six are about the asset's own properties; the seventh, haircut adequacy under correlated stress, is the one that breaks in production.

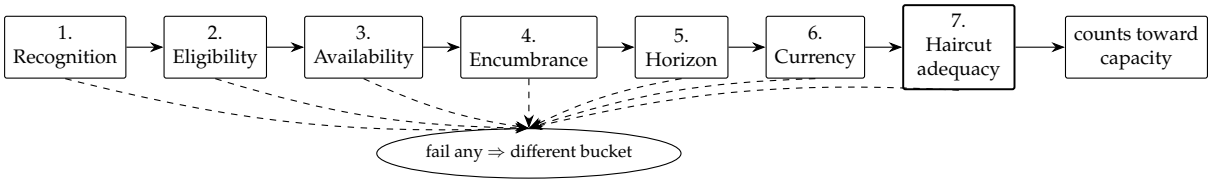


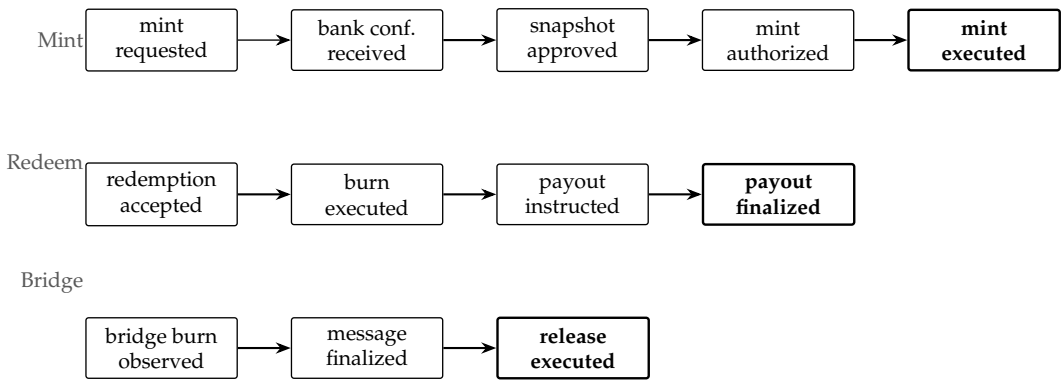
Figure 7. The seven gates in series. Gate 7 (bold) matters because reserve liquidation value and redemption demand are *negatively correlated* under stress: a T-bill can pass gates 1–6 and still fail if its haircut was calibrated for orderly markets.

The unauditable-input problem. Gate 7 depends on `haircut_required_under(scenario= correlated_run, h, c)`, the most consequential number in the reserve model, produced by a process with an obvious incentive to be optimistic. Forcing the haircut into a gate fixed the *form* of the error; it relocated the *subjectivity* into the scenario function. Two failure modes: **self-calibration** (an issuer marking its own illiquid book) and **pro-cyclicality** (a scenario calibrated on trailing volatility is loosest exactly before a stress). Therefore `haircut_required_under`, `stress_buffer`, and `max_*_age` are **governed inputs**, each carrying a methodology document, a calibration source, a change-authority record, and a version id referenced in the reserve snapshot. An audit firm cannot sign off on a capacity number whose largest determinant is an unspecified process.

11 The Control Ledger: Authorization Is Not Finality

The control ledger is where evidence becomes action. It does not hold money or tokens; it holds the record of what has been *authorized* and on what basis. Its single most important property is that authorization and finality are different events separated in time: a mint can be authorized against a reserve snapshot whose underlying bank credit is not yet final, and a redemption can be recorded as a burn whose cash payout has not yet settled. Every operation therefore moves through a small pipeline of states, and **every arrow between two states is a point where the form of the previous step can be mistaken for the finality of the next.**

Three pipelines matter. On the **mint** side, a request is admitted, a bank confirmation is received, a reserve snapshot is approved, the mint is authorized, and only then executed. On the **redemption** side, a redemption is accepted, the tokens are burned, a payout is instructed, and the payout finalizes. On the **bridge**, a burn is observed on the source domain, the cross-domain message is finalized, and the release executes on the destination. The irreversible act in each, executing the mint, finalizing the payout, releasing on the destination, sits at the *end*; the danger is that an earlier, merely formal step is treated as if it were that final act.



Bold boxes are the irreversible acts. Heavy arrows are seams where form can be mistaken for finality: a received confirmation is not an available balance, an approved snapshot is not a final reserve, a burn is not a completed payout, a finalized message is not a reconciled release. The thin arrow (request to confirmation) is the one transition that asserts nothing about money.

Figure 8. The control pipelines for mint, redemption, and bridge. Authorization sits upstream of the irreversible act; each heavy arrow is a place where an earlier formal state is wrongly read as final.

12 Root Cause (Two Independent Defects)

The single most important point: **this is not one bug. It is two, with different owners and different fixes.**

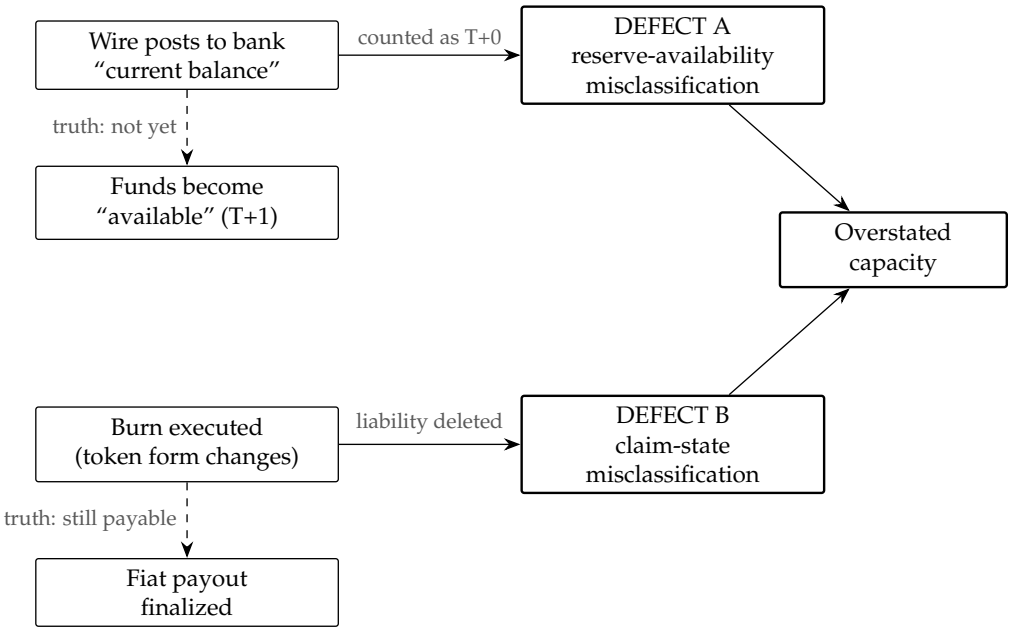


Figure 9. Two independent defects with different owners. Defect A (treasury) is cured by binding the snapshot to availability and finality class. Defect B (redemption and bridge accounting) is cured by making any supply-reducing event a claim-state *transition*, not a deletion. Conflating them into one number hides which fix is urgent.

The split is along a single axis: Defect A is a **reserve-side** error, an asset counted before it is available; Defect B is a **claim-side** error, an obligation treated as discharged before it is settled. Defect B is therefore not only the burned-but-unpaid redemption shown above. Any event that changes a token's *form* while the underlying obligation is still live belongs to it: a burn whose fiat payout has not finalized, and equally an in-flight bridge release whose destination claim is

not yet reconciled. Both are the same mistake, reading a state transition as an extinguishment, which is why the worked example in §13 counts both under Defect B.

In plain English:

```
received   != available      burned   != paid
recognized != eligible       bridged  != finalized
eligible   != liquid         frozen   != extinguished
attested   != continuous     issued   != total claims
```

13 Worked Example, Decomposed by Root Cause

§12 named the two defects; this section puts numbers on them, on a single account at a single instant, so the structure of the overstatement is visible rather than asserted. Everything is one currency (USD) and one horizon (T+0); the multi-currency, multi-horizon version follows in §14.

Start with the honest figure. Assume the tokens already in circulation are matched by reserves held elsewhere; the question here is how much *additional* minting the issuer's free cash can support, so the figures below are the cash *above* that existing backing. The issuer holds $100m of such cash usable today, but not all of it is free: $25m is already owed to redeemers whose tokens were burned but who have not yet been paid, $10m is committed to bridge releases still in flight, and $15m must be retained as the stress and operational buffer. What remains is the most that can be safely minted today.

The two redemption-related figures deserve care, because this is exactly where Defect B will strike. When a redeemer's tokens are burned, the obligation does not vanish; it *moves*. In the full claim definition (§17) it leaves `circulating_tokens` and enters `pending_redemptions_unpaid`, so the total claim count is conserved across the burn. The honest $25m line is that relocated obligation, still live because the cash has not yet left the bank. A correct ledger keeps it; a buggy one deletes it at the burn and conserves nothing.

Available cash (T+0)	$100m
Pending redemptions, burned but unpaid	$25m
In-flight bridge releases	$10m
Required stress / operational buffer	$15m
True T+0 mint capacity	$100 - 25 - 10 - 15 = \$\mathbf{50m}$

Now introduce the failure. A large incoming wire posts to the bank's **current balance**: $+\$40m$. That money is not *available* until T+1, but the buggy controller makes two independent errors at once. It credits the wire as though it were T+0 capacity (the reserve-side error), and it treats the $25m of burned-but-unpaid redemptions and the $10m of in-flight bridge claims as already gone (the claim-side error). The result is $100 + 40 - 0 - 0 - 15 = \$125m$ of believed capacity against $50m of real capacity, an overstatement of $75m.

The headline "$75m too high" is the number to distrust, because it hides that the $75m is the sum of two structurally different mistakes with two different owners. The reserve-side half is Defect A; the claim-side half is Defect B, which here has two components, both instances of the same "state transition read as extinguishment" error: the burned-but-unpaid redemption ($25m) and the in-flight bridge claim ($10m).

Table 2. The decomposition most analyses skip. The $75m headline is reported *only* alongside its split, because the two halves have different owners and different fixes.

Defect	Mechanism	Magnitude
A, reserve availability	T+1 wire credited as T+0 (asset not yet available)	$40m
B, claim state, redemption	burned-but-unpaid liability zeroed	$25m
B, claim state, bridge	in-flight bridge claim zeroed	$10m
Combined overstatement		**$75m**

The split is not cosmetic. Defect A is fixed in the treasury function, by binding the reserve snapshot to availability and finality class so an unavailable wire cannot count. Defect B is fixed in redemption and bridge accounting, by recording every supply-reducing event as a claim-state transition that persists until the underlying payout or cross-domain release finalizes. Two teams, two remediations, two different review trails. A single "we were $75m off" would tell neither team what it owns.

It is worth being precise about the *shape* of each error, because they are not symmetric. Defect A is an **over-count on the asset side**: a dollar of capacity is added that should not be there. Defect B is an **under-count on the claim side**: a dollar of obligation that should be subtracted is not. In the formula capacity $=$ assets $-$ claims $-$ buffer, the two errors push the same number up from opposite directions, which is exactly why their magnitudes add cleanly to $75m and why fixing one does nothing for the other. Defect B in particular is an error of *omission*, the liability is not mis-valued, it is simply absent, so it cannot be caught by re-checking the claims that remain on the books; it can only be caught by an invariant that insists a burned or bridged token leaves behind a claim until cash or release finalizes (§18). Both of Defect B's components, the burned-but-unpaid redemption and the in-flight bridge claim, are the identical mistake applied to two different supply-reducing events: a change in token *form* read as the extinguishment of the obligation behind it.

14 Worked Stress Over the Full Matrix

The decomposition in §13 was a single account, one currency, one instant. It showed the mechanism but not the model. This section runs the full apparatus, the $[h, c]$ capacity matrix, the priority waterfall, the non-double-count invariant, and currency stress, on one concrete scenario, because a model that is only described and never executed is a claim, not a result. The scenario has two currencies (USD and EUR), two settlement horizons (T+0 today, T+1 tomorrow), and one priority class, with a single EUR/USD rate of 1.08 applied consistently throughout.

The point of the matrix is that solvency is not a scalar. A single reserves-versus-claims number can be comfortably positive while the issuer still cannot pay what is due, because money has a *when* and a *which currency* attached to it, and those attributes do not net against each other. The matrix makes both explicit by splitting capacity into one cell per (horizon, currency) pair and refusing to let a surplus in one cell silently cover a shortfall in another.

Reserve assets

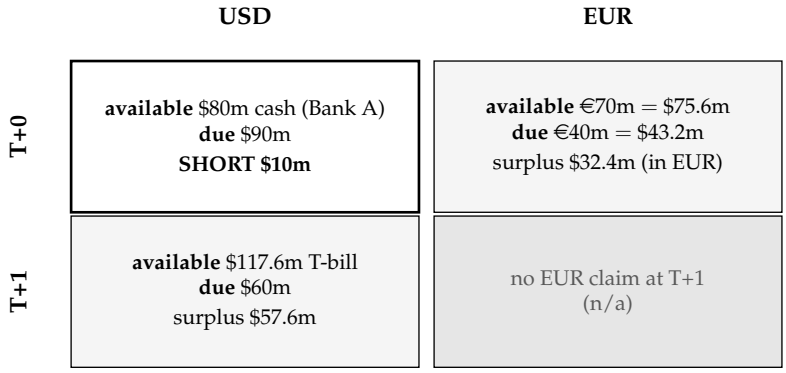
Asset	Value	By
Cash, Bank A	$80m	T+0
T-bill ladder	$120m	T+1
(2% haircut)	$117.6m	
Cash, Bank B	€70m	T+0

Claims due

Claim	Amount	Horizon
USD redemptions	$90m	T+0
USD redemptions	$60m	T+1
EUR redemptions	€40m	T+0

Capacity is checked cell by cell: each [horizon, currency] cell compares the eligible assets

that can settle in that currency by that horizon against the claims due in the same cell. An asset counts in a cell only if it clears every gate for that cell, in particular the horizon gate (it settles in time) and the currency gate (it is the right currency, or hedged into it).



Each cell pairs the assets that can settle in that currency by that horizon *against the claims due in the same cell. Three cells clear. The heavy-bordered cell, T+0 USD, is short $10m: only $80m of cash settles today against $90m of dollar claims due today.*

The trap, and why the obvious plug is illegal. The single shortfall is $10m of T+0 dollars. The tempting fix is to reach for the T+1 T-bill, "it is only a one-day gap, and T+1 has a $57.6m surplus." Two independent controls forbid it. The **decisive** one is the *horizon gate*: the T-bill settles T+1, so it cannot settle by T+0 and is *not countable* in the T+0 cell at all, the plug is illegal before any arithmetic is done. The *non-double-count invariant* (§19) blocks reuse in general, an asset's value is partitioned across the cells it backs, but note that here the numbers are not even a double-count case ($10 + 60 < 117.6$); the binding reason is horizon, not quantity. The large EUR surplus does not help either: it is the wrong currency for a dollar claim (the currency gate). **The result is a T+0 liquidity breach**, due T+0 USD claims cannot be met from eligible T+0 USD assets, which is *not* the same as balance-sheet insolvency, the issuer holds ample total assets, just not the right currency at the right time. A naive per-cell check can show all-green only by improperly reusing an asset across cells; the gates and the non-double-count invariant expose the shortfall the blended view hides.

Figure 10. The full-matrix stress. Three of four cells clear; the T+0 USD cell is short $10m. Concentration then adds the systemic layer: every dollar of T+0 USD cash, the only asset that can actually meet the T+0 USD claims, sits in a single uninsured account at Bank A. That is 100% single-name concentration in the one cell that is already short, whatever the bank looks like as a share of the total $276m reserve base. If Bank A wobbles, the cell goes from short $10m to short $90m, and the contagion trigger (§30) says the run tends to arrive precisely then.

Reading the four cells in turn makes the discipline concrete. **T+0 USD** pairs the only dollars that settle today, $80m of Bank A cash, against $90m of dollar claims due today: short $10m. **T+0 EUR** pairs €70m ($75.6m at 1.08) against €40m ($43.2m) of euro claims due today: a comfortable surplus, but one denominated in euros. **T+1 USD** pairs the $117.6m post-haircut T-bill, which settles tomorrow, against $60m of dollar claims due tomorrow: a $57.6m surplus. **T+1 EUR** has no claim and is not applicable. The blended, single-number view sums everything and reports a large surplus; the matrix view shows that the only thing that matters at T+0 in dollars is the $80m that can actually settle in dollars today, and it is $10m short. Every surplus in the grid is real, and none of it is reachable: the T+1 dollars are a day late, and the T+0 euros are the wrong currency. Solvency in aggregate coexists with a breach in one cell, which is precisely the failure a headline coverage ratio is structurally unable to see.

15 Exploit Routes and Severity

An overstated capacity number is only dangerous if something *consumes* it. This section connects the two defects to the concrete ways an overstatement is turned into a loss, and then to how

badly each one bites. The routes are not hypothetical attacks on cryptography; they are ordinary sequences of legitimate operations that pay out against capacity that was never really there.

Route A, reserve finality mismatch. A client's funds are used to authorize a mint: the issuer counts a current-but-unavailable balance as T+0 capacity, releases the tokens, and only then does the inflow turn out to be delayed, recalled, or disputed. The tokens are already irreversible; the backing never finalized. This is Defect A consumed at the mint join.

Route B, liquidity drain. A client redeems freshly minted tokens into a T+0 payout path that draws down existing liquidity, while the inflow that supposedly backed the mint is only available at T+1. Nothing is stolen; the issuer simply pays out today against money that arrives tomorrow, and the timing gap is the loss.

Route C, cross-domain drain. Tokens are minted on one domain, moved to another, and the destination releases value before the cross-domain claim is globally reconciled. The same obligation is briefly honoured twice across the seam between domains.

Routes A and B are reachable by an ordinary authorized client who merely *times* actions around known settlement windows (threat T1) or by passive correlated stress (T2); Route C additionally requires a compromised or stalled bridge or destination domain (T4).

Severity is not a property of the defect itself but of *where the bad number is consumed*. Four cases, in rough order of seriousness: **solvency**, the overstatement means eligible assets are actually below effective claims, the issuer cannot make good even given time; **liquidity**, assets available by a horizon fall short of claims due by that horizon, solvent on paper but unable to pay on time (the §14 breach); **reporting**, the bad number reaches dashboards and attestations but not the mint authorizer, misleading observers without yet causing a loss; and **exploitability**, whether an actor can deliberately force or time the mismatch rather than merely benefit from it by chance. The practical rule: if the overstated capacity feeds *mint authorization*, the severity is critical, because issuance is irreversible; if it only feeds a dashboard, it is reporting risk until the day it does not.

16 The Weak Dashboard

It is worth naming the single artifact that lets all of this hide in plain sight, because nearly every issuer reports some version of it and it looks authoritative:

```
coverage = (bank_current_balance + T_bills) / onchain_totalSupply
```

A ratio above one is read as "fully backed." It is not a control; it is a screenshot with its assumptions hidden inside it, and each of its three inputs quietly fails a gate established earlier.

The numerator is wrong on the asset side. `bank_current_balance` is a *current* figure, not an available one, so it fails the availability gate (gate 3) by counting funds that have not cleared, exactly Defect A; and it is counted at face, failing the haircut-adequacy gate (gate 7) by ignoring what the balance is worth under stress. `T_bills` are counted as if spendable now, but they fail the availability gate (gate 3) and, for any obligation due today, the horizon gate (gate 5), since a T+1 instrument cannot settle a T+0 claim.

The denominator is wrong on the claim side. `totalSupply` is on-chain *supply*, not *claims*: it omits every off-token obligation, the burned-but-unpaid redemptions, the in-flight bridge claims, the frozen-but-valid balances, the contingent reserve, exactly the claim-side terms Defect B deletes. It also ignores currency entirely, blending a dollar liability with a euro asset.

So the reassuring single number conceals at least four independent assumptions, that current equals available, that face equals stressed value, that any maturity can meet any horizon, and that supply equals claims, each of which is precisely one of the seams the rest of this document exists to close. A coverage ratio is a fine thing to publish *after* those gates have been applied; it is dangerous as a substitute for them.

17 The Objects and the Capacity / Liability Models

Everything so far has argued in prose; this section pins the argument to the two data structures the whole model stands on. If a claim or a reserve asset is missing a field, the corresponding gate or invariant simply cannot be evaluated, so the schemas below are not documentation, they are the minimum state that makes the rest of the document checkable. Two objects carry the system.

```
claim = { claim_id, claimant, amount, currency, domain,
  legal_status, reserve_coverage_status, maturity_horizon, priority,
  settlement_state, revocability, blocking_condition,
  predecessor_ref, terminating_ref, evidence_id }

reserve_asset = { asset_id, currency, legal_owner, custodian,
  recognized, eligible, available_by, encumbered_amount, valuation_method,
  stress_haircut, haircut_scenario, maturity, liquidity_horizon,
  counterparty_risk, concentration_bucket, bankruptcy_remoteness,
  segregation_status, evidence_id }
```

Several fields are load-bearing in ways the earlier sections now explain. On the claim, `legal_status` and `reserve_coverage_status` are kept *separate* precisely because a claim can be one without the other (§9); `settlement_state` together with `predecessor_ref` and `terminating_ref` is what lets a burn be recorded as a *transition* rather than a deletion, closing Defect B; and `maturity_horizon`, `currency`, and `priority` are the three axes the capacity matrix is indexed on. On the reserve asset, `available_by` encodes the horizon gate, `currency` the currency gate, `encumbered_amount` the encumbrance gate, and the pair `stress_haircut` with `haircut_scenario` the haircut-adequacy gate, the seventh and most gameable (§10). `evidence_id` appears on both because every value that feeds an authorization must be traceable to the attestation that justified it.

Horizon-, priority-, currency-aware effective claims. The claim side of the capacity formula is not a single total but a sum over distinct obligation states, each of which a naive system is liable to drop:

```
effective_claims[h,p,c] = circulating_tokens + pending_redemptions_unpaid
  + failed_payouts + in_flight_bridge_claims + frozen_but_valid_claims
  + unsettled_mint_obligations + domain_migration_claims
  + contingent_claim_reserve[h,p,c]
```

The first term is the visible supply; every other term is an obligation that exists *off* the token and is therefore invisible to a `totalSupply`-based view. Defect B is exactly the deletion of the second and fourth terms at burn and bridge time; the frozen and contingent terms are the ones compliance and legal most often argue away.

Resolving an internal contradiction: contingent claims. The legal-liability-vs-coverage distinction admits *contingent* obligations, a clawback, a court order, a regulatory freeze converting to forced redemption. A contingent claim has no determinate amount until its condition resolves, while the matrix invariant requires a number. Omitting them understates liability; including them at face value overstates it. The only honest treatment is a **scenario-bounded reserve**:

```
contingent_claim_reserve[h,p,c] =
  SUM over contingent claims k matched to (h,p,c) of severity(k) * P_resolve_by(k,h)
  # bounded, not face value, not zero; the method is a GOVERNED input
```

Unencumbered reserve capacity. The asset side mirrors the claim side: start from cash genuinely available at the horizon, add only the assets that clear all seven gates (counted at

their stressed liquidation value, not face), and then subtract everything already spoken for, so that no dollar is counted twice across commitments, holds, buffers, or domains.

```
unencumbered_capacity[h,c] = available_cash[h,c]
  + SUM_i stress_liquidation_value(asset_i,h,c)   # only assets passing all 7 gates
  - committed_payouts - bank_holds - settlement_buffer
  - operational_floor - domain_allocations - encumbrances
```

Read together, the two formulas are just the two halves of $\texttt{mint\_capacity}[h, c]$ from §7 made concrete: an asset side that counts only what survives the gates, and a claim side that counts every obligation including the ones that have left the token. The headline formula's $\texttt{due\_effective\_claims}[h, c]$ is precisely $\texttt{effective\_claims}[h, p, c]$ restricted to the obligations *due* by horizon h in currency c and summed over priority p; its $\texttt{unencumbered\_capacity}[h, c]$ is the asset expression directly above. Defect A corrupts the first half; Defect B corrupts the second.

18 Invariant 1, No Orphan Lifecycle Transitions

A value is not an invariant until it is a checkable predicate. One caveat governs every invariant that follows: each is stated as a checkable predicate *over explicitly classified inputs*, but the correctness of those classifications remains a legal, accounting, risk, or governance attestation, not a computation. Whether a frozen claim is legally extinguished, whether an asset is bankruptcy-remote, whether a haircut is adequate under correlated stress, whether a third-party wrapper is issuer debt, whether a redemption throttle is lawful, whether a claimant-transfer event is legally effective, these are predicates over *classified facts* supplied by processes outside the validator. The machine checks the predicate; the attestation establishes the classification. Conflating the two is its own form of fake rigor.

> **Definition.** Every state transition carries a `predecessor_ref` and, on completion, a `terminating_ref` (settled, extinguished, transferred, or failure-path-resolved). The set of *open* claims is exactly the set of transitions with a predecessor but no terminating reference.

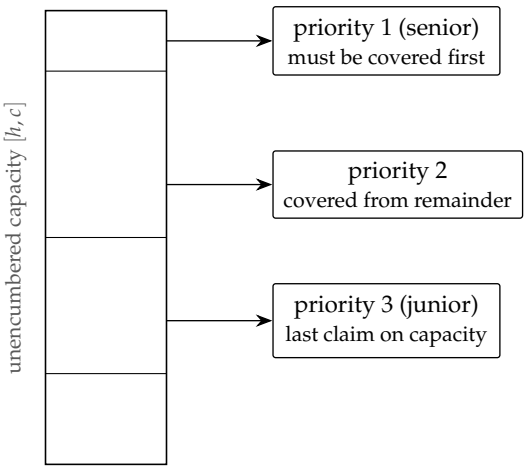
```
open_claims == { t : t.predecessor_ref != NULL and t.terminating_ref == NULL }
for every transition t:
    t.predecessor_ref != NULL                 # nothing appears from nowhere
    t.terminating_ref == NULL  XOR  t.is_open == FALSE
liability_total == SUM over open_claims of claim.amount   # ledger == open set
```

A CI validator computes both sides and asserts equality on every batch close.

19 Invariant 2, The Priority Waterfall

Ranking priorities is not allocating contested capacity. Under scarcity, capacity must be allocated *in priority order*, cumulatively.

> **Invariant.** For every horizon h, currency c, and every priority threshold p^* (most senior first):
>
> $$\texttt{unencumbered\_capacity}[h, c] \;\geq\; \sum_{p \leq p^*} \texttt{due\_effective\_claims}[h, p, c] + \texttt{stress\_buffer}[h, p^*, c].$$



The invariant must hold at every *cut p*, not per line item, otherwise a junior claim clears while a senior claim in the same bucket starves, the actual failure mode in resolution.*

Figure 11. Capacity drawn off in strict priority order. A per-line-item check produces false greens precisely when it matters most.

Two refinements resolution litigates. *(a) Within-class rationing* must be declared, pro-rata is the default that survives challenge; time-ordered or pre-emptive rules are valid but must be disclosed in advance. *(b) Shared assets* must be partitioned before per-cell checks:

```
NON-DOUBLE-COUNT INVARIANT:
  for every reserve asset a:  SUM over cells (h,c) of allocated(a,h,c) <= countable_value(a)
```

A matrix that reports green in every cell only by reusing a shared asset across cells is misreporting: it claims coverage it does not have. Whether the resulting shortfall is a liquidity breach or true insolvency depends on the eligible-asset total (§14 is a liquidity breach, not insolvency).

20 Invariant 3, Snapshot Freshness and Single-Use Binding

The mint invariant requires a `reserve_snapshot_id` but must also constrain its *age* and *consumption*. Otherwise the same favorable snapshot authorizes multiple mints (replay), or a snapshot taken before a large redemption authorizes a mint after it (staleness).

```
mint_execution(mint_id) requires reserve_snapshot_id s such that:
  1. FRESHNESS:  now - s.taken_at <= max_snapshot_age[h]
  2. ORDERING:   s.taken_at >= last_materially_adverse_event_before(mint_id)
  3. SINGLE-USE: s.consumed == FALSE -> set s.consumed = TRUE atomically on mint
  4. MONOTONIC:  s.sequence > last_consumed_snapshot_sequence[h,c]
```

This single join is where reserve evidence becomes mint authority; leaving it unguarded nullifies every other control upstream.

21 Invariant 4, Liveness, and the Asymmetry of Halting

Every prior invariant is a *safety* property. But the T1 adversary times pause latency and the T2 stress depends on stale reconciliation, seams in *time*. A safety-correct system that is unavailable or stale at the moment of stress fails depositors exactly as hard as an overstated one.

```
1. RECONCILIATION FRESHNESS: now - last_full_reconciliation > max -> HALT
2. FINALITY-CLASS FRESHNESS:  stale bank class -> treat balance as 'provisional'
```

```
3. PAUSE BOUND: time_to_effective_pause <= max_pause_latency < fastest_settlement_window
4. SNAPSHOT-TO-MINT LATENCY:  continuously monitored bound (restates Inv. 3.1)
5. GRACEFUL DEGRADATION: effective_capacity *= max(0, 1 - staleness/max_staleness)
6. HYSTERESIS: resume threshold stricter than halt threshold (no flapping)
7. FALSE-HALT COST: an unnecessary halt is a tracked control cost, not a non-event
```

Halting mint is safe; halting redemption may be unlawful. Degradation silently assumed the issuer always *wants* to stop the flow. True for **mint** (issuer discretion); false for **redemption**, which may be the holder's *right* under contract or regulation. Throttling a redemption right is a potential default or compliance violation, a legal event, not a safety action. The high-signal statement of the point: **a pause control is directional.** The same word "pause" means risk reduction on the mint side and possible default on the redemption side. Many crypto risk frameworks casually say "pause everything"; for regulated money that is not necessarily an available action.

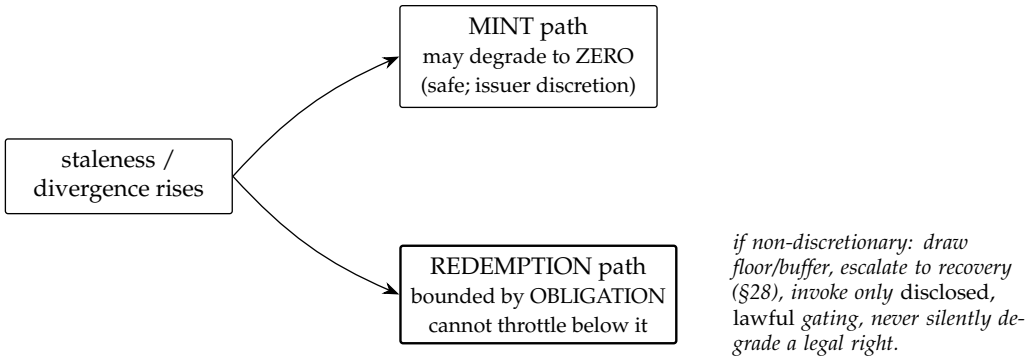


Figure 12. Asymmetric degradation. Halting issuance buys time honestly; throttling redemption converts a liquidity problem into a legal one.

22 The Latency Budget, Do the Controls Compose?

Every prior section *added* a control; none asked whether they fit together in time. A growing pile of "must" without a joint-satisfiability proof is a wish list an engineer will quietly violate, producing an undocumented deviation, worse than the gap it patched.

Conflict 1, local cut vs. pause latency. Assembling a global cut is slow; the settlement window is fast. The consistency model of §5 resolves this: the local cut is over *issuer-owned* ledgers only (cheap, synchronous); external belief is pre-validated out of band.

```
t_authorize = t_local_cut + t_belief_lookup + t_invariant_eval
REQUIRE: t_authorize       < settlement_window[h]
REQUIRE: max_pause_latency < settlement_window[h]
REQUIRE: t_local_cut is O(issuer-owned reads), NOT O(external poll)
If settlement_window[h] too short (true T+0/instant):
   the product MUST prefund -- prefunding moves the reserve event BEFORE the
   mint request, collapsing the seam. You buy latency headroom, you do not pretend.
```

The honest consequence: **instant settlement and post-hoc reserve verification are incompatible; you must prefund.**

Conflict 2, timelock vs. legitimate recalibration. A genuine operational loosening (raising `max_feed_age` during a planned oracle migration) would be blocked by the timelock exactly when needed. Resolved by distinguishing *standing-parameter loosening* (timelocked, slow) from *pre-declared operational transitions* (a named runbook approved in advance under the same m-of-n, executing within a bounded window). The principle: **a control that cannot be satisfied jointly with another is not a control, it is a latent deviation.**

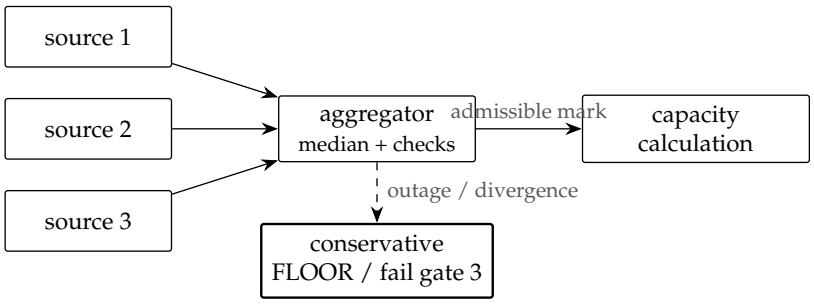
23 Lifecycle Invariants

```
MINT:   mint_execution requires fresh, single-use snapshot AND amount <= capacity[h,c]
REDEMPTION (burn moves the claim, does not delete it):
  burn(redemption_id, amount) => claim_state = pending_cash_payable
    until fiat_settlement_finalized OR failure_path_resolved
BRIDGE (settlement is a lifecycle):
  bridge_burn(message_id,...) => claim_state = in_flight_cross_domain_claim
    until destination_release_finalized OR refund_or_retry_resolved
RESERVE (countability is a controlled state):
  countable_for_mint(h,c) iff recognized && eligible && available_by(h)
    && unencumbered && stress_haircut_applied && gate_haircut_adequate
    && currency_matched_or_hedged(c)
FREEZE (compliance and liability are separate):
  freeze(addr, amount) => transferability = restricted; claim_status = unchanged
    unless legal_extinguishment_id OR claimant_transfer_id exists
```

24 Domains, Multi-Chain Control, and Valuation-Source Integrity

Tag every domain balance: `issuer_native | issuer_bridged | third_party_wrapped | synthetic | exchange_internal`. Only some are issuer-recognized liabilities. Per-domain: `domain_supply[d] + pending_outbound[d] <= domain_limit[d]`; global: `SUM(recognized domain claims) + pending_cross_domain <= global_liability_limit`. For native multi-chain issuance the issuer *is* the bridge; for third-party wrappers the issuer may be solvent while the wrapper is not, which must be disclosed.

Every capacity number depends on inputs the issuer does not produce: `stress_liquidation_value` needs marks; FX needs rates; an MMF needs a NAV strike. A stale or manipulated feed overstates capacity exactly as stale reconciliation overstates claims. §21 bounded internal staleness; this is the *external* seam.



Admissibility requires: staleness bound, multi-source agreement, manipulation resistance (TWAP / multiple venues / attested NAV), conservative fallback (never last-good held indefinitely), and an MMF break-the-buck re-gating trigger. `max_feed_age` *belongs in the liveness regime.*

Figure 13. Valuation-source integrity. A single stale FX fix overstates EUR-backed USD capacity precisely as a stale reconciliation overstates claims.

25 Attestation: Scope vs. Inference

A point-in-time attestation proves a scoped statement at a cutoff; the market hears "continuously fully backed." These are different claims, and the gap is a seam. A rigorous package discloses liability states included *and* excluded; treatment of pending redemptions, bridge claims, frozen balances, failed payouts; reserve eligibility, valuation method, haircuts *and haircut scenario*, maturity buckets, encumbrances; cutoff policy; reconciliation exceptions. Excluding a state

can be acceptable *if clearly scoped*. The failure is using an excluded state as a non-claim in mint capacity, dashboards, or public coverage ratios. **Scope must match use.**

26 Buffers Are Not a Substitute for Correctness

More reserves can *mask* bad classification: they raise the stress level required to expose the defect without removing it. A serious issuer needs both correct state accounting *and* stress buffers by horizon, priority, and currency. Compensating controls that reduce severity: mint delay until available balance, prefunding, client credit limits, redemption cutoffs, liquidity buffer, bridge caps, domain pause, dual control on reserve classification, real-time reconciliation, and an exception queue for failed payouts.

27 The Config Plane, Governing the Parameters (T5)

The cheapest real attack. Every invariant is safe *only relative to a set of parameters*: `max_snapshot_age`, `domain_limit`, the eligibility whitelist, `haircut_required_under`, `stress_buffer`, `max_feed_age`, `max_reconciliation_staleness`. Each is mutable. The most efficient way to overstate capacity is not finality arbitrage, it is to *relax a parameter*. No invariant is violated; the invariant is simply re-pointed at a weaker bound. This is the seam between **policy and mechanism**.

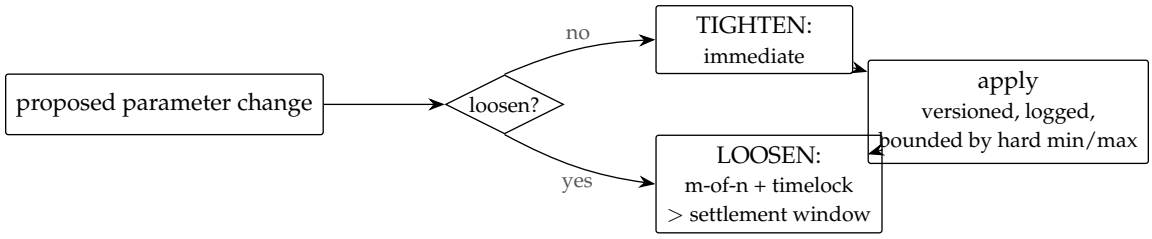


Figure 14. The config plane. Directionality is deliberate: loosening is slow and reviewed, tightening is fast and free, because the asymmetry favors safety. A model with perfect invariants but single-key parameters has merely moved the private key from the mint function to the config function.

The emergency-override trap. Timelocks, bounds, and multi-party approval are defeated by the one path most systems quietly keep: emergency powers that bypass them. That is usually where the real T5 risk sits, an insider does not loosen `max_snapshot_age` through governance, they invoke the emergency path. The override must therefore be constrained to be risk-reducing only:

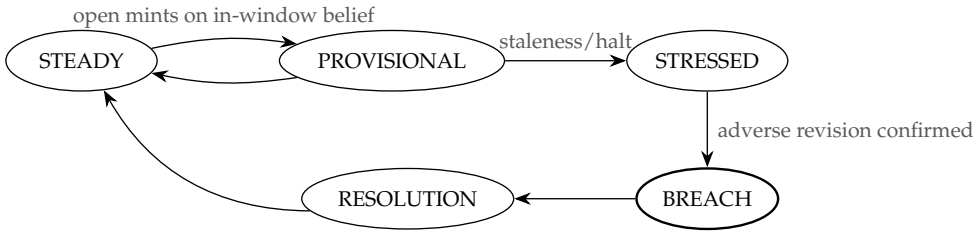
```
Emergency override:
  allowed ONLY for tightening / risk-reducing changes
  CANNOT loosen eligibility, haircut, snapshot age, domain limit,
         or liability treatment
  must expire automatically (no standing emergency state)
  must be post-reviewed and disclosed
```

Without this asymmetry, every other config-plane control is theatre: T5 simply routes around the timelock through the emergency door.

28 The Recovery State Machine, Prevention Fails; Then What?

Everything to this point is *preventive*. But the threat model contains correlated stress and custodian reversal, and §5 established that mints against in-revision-window beliefs are *provisional*.

So a mint *will* occasionally clear against capacity that later proves illusory. A specification that is all prevention and no recovery is a happy-path document.



Breach resolution order (declared in advance, not improvised under stress): *(1) ABSORB, draw operational floor + stress buffer; (2) TOP-UP , reserve replenishment from equity/backstop,* not *new issuance against the same illusory capacity; (3) CLAWBACK, pursue the reversed leg per the seam contract's* `on_reject`*; (4) SOCIALIZE, only if 1–3 exhausted and only per pre-disclosed lawful terms. Undisclosed socialization is fraud, not recovery.* ***Invariant: the public coverage representation must not say STEADY while internal state is PROVISIONAL/BREACH.***

Figure 15. The operating-state machine. BREACH is the state every prevention-only model omits. Prevention is probabilistic; recovery determines whether a bad mint becomes an embarrassment or an insolvency.

Every recovery transition must be mapped to an enforceable legal basis; otherwise it is not a recovery path, only an operational wish. The resolution order above is necessary but not self-executing. Depending on the stablecoin's terms and jurisdiction: clawback may be impossible against secondary holders; top-up may be unenforceable unless pre-funded or guaranteed; the operational buffer may be legally segregated for other uses; suspension clauses may be limited or absent; socialization may be prohibited outright; and forced redemption or claim transfer may require legal authority the issuer does not hold. A resolution runbook that lists a step with no enforceable basis behind it is planning to do something it cannot lawfully do, which, discovered mid-breach, is worse than having no plan.

29 Trust Assumptions and Collusion, Name the Threshold

T1–T5 were treated as independent. The dangerous cases are *products*, and the config-plane defenses are *themselves* non-collusion assumptions: m-of-n fails at *m* colluding signers; separation of duties fails when the separated parties cooperate.

```
COLLUSION THRESHOLD (state explicitly per critical control):
  snapshot signing             : safe unless >= m_snap of n_snap collude
  reserve-eligibility whitelist: safe unless >= m_elig of n_elig collude
  oracle aggregation           : safe unless >= m_oracle feeds collude/fail
  parameter governance         : safe unless >= m_gov of n_gov collude
  GLOBAL resistance = MINIMUM of the above (adversary targets the weakest quorum)

SINGLE POINTS OF TRUST (compromise voids MULTIPLE invariants at once):
  snapshot-signing key      -> voids Inv.3 and the local cut simultaneously
  eligibility-whitelist auth -> voids gates 1-2 for any asset
  oracle aggregator         -> voids valuation integrity and the FX leg
  pause/governance multisig -> voids liveness and the config plane
```

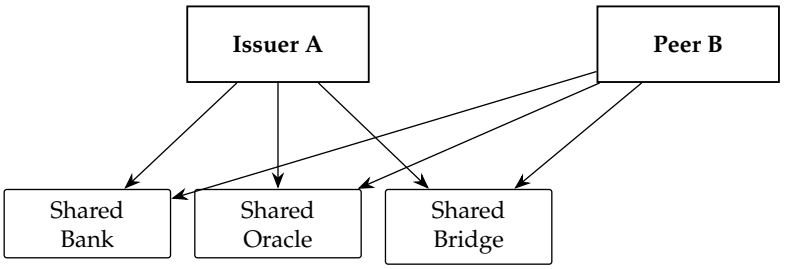
Cross-threat products to defend: **T1+T5** (insider relaxes `max_snapshot_age` for the client's exact window); **T3+T5** (jointly mark a reversed posting as final). The defense is genuine independence of guardians, different people, keys, vendors, jurisdictions, so no single collusion set spans two single points of trust. And the independence must be *infrastructural*, not just nominal: **quorum diversity is fake if the quorum members share the same bank group, cloud provider, custody**

provider, signer institution, HSM vendor, legal administrator, oracle data vendor, or incident-response operator. A threshold that is mathematically m-of-n can be operationally 1-of-1 if all signers depend on the same underlying infrastructure. **Naming the trust assumption is more valuable than adding the next mechanism, because the mechanism is only as strong as the assumption it silently rests on.**

30 Systemic Correlation, The Issuer Does Not Fail Alone

This section is an overlay, not part of the primary internal-control finding. The internal model is necessary but not sufficient; systemic correlation bounds the model from outside. It changes the claim from "stablecoin internal-control failure" to "the limits of any internal-control model," and is included so the reader does not mistake intra-issuer correctness for safety.

Every invariant so far is issuer-internal, the model's last and largest blind spot. The canonical stablecoin failures were *systemic*: a reserve bank that itself failed, a redemption wave triggered by a *different* coin's depeg, a shared oracle or custodian across competing issuers, reserves that are themselves another stablecoin.



A shared dependency means peer failure is correlated *with yours, breaking the independence the buffers assume. An issuer can pass every internal predicate and still depeg because its bank, oracle, or reserve asset was simultaneously someone else's single point of failure.*

Figure 16. Shared dependencies couple nominally independent issuers. Solipsistic correctness is not safety.

The matrix needs correlation terms it currently lacks:

```
1. COUNTERPARTY CONCENTRATION: reserve at any single bank / total <= conc_limit
2. CROSS-STABLECOIN EXPOSURE:  a reserve/settlement asset that is another
   stablecoin -> its depeg is YOUR depeg. Tag and cap.
3. SHARED-DEPENDENCY MAP:      enumerate oracle/bridge/custodian/rail shared with peers
4. CONTAGION TRIGGER:          model redemption demand as correlated with MARKET
   stress; size stress_buffer for correlated, not idiosyncratic, runs
```

The honest statement to depositors and regulators: **an issuer's safety is bounded above by the safety of its least-safe shared dependency.**

31 The Assumption Ledger

If only eight needs can be carried (for a short note or a public summary), these are the highest-severity: (1) mint only against eligible capacity; (2) burn creates a payable; (3) bridge burn creates an in-flight claim; (4) frozen balances classified legally; (5) snapshot freshness and single-use; (6) seam contracts complete and derived; (7) config-plane governance; (8) a recovery state machine. The full ledger below additionally carries liveness, valuation feeds, the latency budget, collusion thresholds, and systemic correlation.

Need	Provider	Evidence	Status if naive
Mint only against recognized, eligible, available, unencumbered, currency-matched, haircut-adequate capacity	treasury / risk	snapshot + available-balance confirmation + encumbrance ledger	**weak** if current balance used
Burn creates pending payable until fiat settlement	redemption processor	payout finality confirmation	**missing** if burn deletes liability
Bridge burn creates in-flight claim until release/refund finalizes	bridge accounting	finalized message proof + state transition	**weak** if pending burns subtracted at once
Frozen balances classified by legal claim state	compliance / legal	legal status transition	**missing** if frozen removed by default
Snapshot fresh and single-use at the mint join	mint controller	age + consumption flag + monotonic sequence	**critical** if age/replay unguarded
Halt on stale reconciliation; pause $<$ settlement window	risk / ops	reconciliation timestamp + pause monitor	**critical** if liveness unmonitored
Authorization over a *local* cut; external reserve as dated revisable belief	ledger substrate / pipeline	local-cut certificate + dated attestations + revision log	**critical** if bank treated as inside the cut
Each seam has a typed contract	architecture	seam-contract registry	**weak** if only the snapshot join is specified
Seam set derived from a closed matrix, frozen under governance	architecture	seam-closure derivation	**weak** if seams are a discovered list
Contingent claims as a bounded reserve	legal / risk	scenario-bounded estimate + method version	**missing** if contingent claims dropped
Shared assets partitioned across cells	treasury / risk	allocation ledger	**critical** if cells pass but share an asset
External feeds fresh, multi-source, manipulation-resistant, conservative fallback	oracle / treasury	feed admissibility log	**critical** if a single stale mark feeds capacity
Stress scenario / haircut floors / buffers as governed inputs	independent risk	methodology + calibration source + version	**weak** if self-calibrated or pro-cyclical
Safety-critical parameters governed (separation, m-of-n, timelock, bounds, versioned, logged)	governance	config-change log + approvals	**critical** if a single role can loosen a guard

Need	Provider	Evidence	Status if naive
Controls fit a latency budget; instant settlement implies pre-funding	architecture / risk	latency budget proof	**critical** if cut + pause cannot fit the window
Asymmetric degradation: mint may halt; non-discretionary redemption may not	risk + legal	degradation policy + redemption-obligation map	**critical** if redemption treated as a dial
Recovery state machine with pre-declared breach order	ops / governance	declared resolution runbook	**missing** if prevention-only
Each recovery transition mapped to an enforceable legal basis	legal / governance	legal-basis register per transition	**critical** if a step has no lawful basis
Emergency override is tightening-only, expiring, post-reviewed	governance	override charter + audit log	**critical** if emergency path can loosen guards
Collusion thresholds and single-points-of-trust named, distributed, monitored	governance / security	trust-assumption register	**critical** if controls assume non-collusion
Systemic correlation terms modeled	risk	systemic-exposure map	**critical** if every invariant is issuer-internal
Halts degrade gracefully with hysteresis; false-halt cost tracked	risk / ops	degradation curve + halt-cost metric	**weak** if halt is binary and grievable

32 The Fix

The fix is **not** "hold more reserves." It is: an honest consistency model (a *local* cut over issuer-owned ledgers, external reserve as dated revisable belief, no global cut); a typed contract for every seam, with the seam set *derived* from a closed matrix; state-accurate claim accounting (burn/bridge/freeze are transitions, not deletions); contingent claims as a bounded reserve; seven-gate reserve accounting including haircut adequacy under correlated stress; horizon-based liquidity controls with a priority waterfall, declared within-class rationing, and no shared-asset double-count (demonstrated, §14); currency-aware eligibility; valuation-source integrity; domain reconciliation; lifecycle-bound controls with no orphan transitions; a fresh, single-use, monotonic snapshot binding; liveness that degrades gracefully and *asymmetrically* (mint may halt, non-discretionary redemption may not); a proven latency budget (instant settlement met by prefunding, not pretended away); a governed config plane with a tightening-only, expiring emergency override; a recovery state machine in which every transition is mapped to an enforceable legal basis; named trust assumptions and collusion thresholds with infrastructural (not merely nominal) quorum diversity; and a modeled systemic overlay.

33 What an Audit Must Cover

The token contract is one line item among two dozen. The deepest items are not mechanisms at all, they are *meta-properties*: is the seam list **complete** (derivable)? Do the controls **compose** (fit a latency budget)? Is there a **recovery** path when prevention fails, and is each recovery transition

mapped to an enforceable legal basis? Who is **trusted**, and what is the collusion threshold? Does the issuer model that it **fails together** with its bank, its oracle, and its peers? A review that verifies every mechanism and none of these has audited the machine and ignored whether the machine can be built, recovered, or trusted.

34 Final Lesson, From Seams to Meta-Properties

The lesson is not that any one component was weak. Every component was correct. A stablecoin is not its components, it is the set of joins between them, and the subtle failures live in those seams.

```
received     is not   available        a snapshot   is not   fresh forever
burned       is not   paid             a pause      is not   instantaneous
bridged      is not   finalized        a local cut  is not   a global truth
frozen       is not   extinguished     prevention   is not   recovery
attested     is not   continuous       my reserves  is not   my bank's solvency
issued       is not   total claims
```

There are deeper seams still, the ones between the model and the operational reality it runs in. The consistency model is the seam between the predicate and the state it reads, and one of those states belongs to a bank that can revise it, so there is no global truth to read, only a dated belief. Valuation feeds are the seam between the model and the outside world. The config plane is the seam between mechanism and policy. The recovery state machine is the seam between prevention and the day prevention fails. The collusion threshold is the seam between a control and the people trusted to run it. Systemic correlation is the seam between the issuer and the system it cannot exit: the most rigorous internal model in the industry does not survive its bank failing.

These do not sit between the boxes in the architecture diagram; they sit between the diagram and reality. They are the hardest to see precisely because a polished framework looks complete, and a polished framework is more dangerous than a rough one: it invites reviewers to stop pushing, and it invites its authors to mistake the absence of a found flaw for the presence of correctness. The mature version of this work is not a longer list of controls, it is a model that is **derivably complete, jointly satisfiable, recovery-capable, explicit about whom it trusts, and aware that it lives in a system that fails together.** The last seam to close is the one between *looking* rigorous and *being* rigorous.

Root cause, stated once. The system overstates mint capacity because reserve observations become mint authority before passing the eligibility, availability, encumbrance, horizon, currency, haircut, and staleness gates, while claim-form transitions are treated as obligation extinguishment before the underlying economic/legal claim is settled, transferred, extinguished, or reclassified.