Normalization Confluence for Registry-Governed Stream Processing

Dayna Blackwell

dayna@blackwell-systems.com

February 19, 2026

Abstract

Coordination-free convergence in distributed systems has been achieved through two structural regimes: *operation commutativity* (CRDTs), where operations commute by construction, and *invariant confluence* (I-confluence), where all orderings preserve application invariants. We identify a third regime—*normalization confluence*—in which operations may be non-commutative and may violate invariants, but convergence is restored through a compensation operator that normalizes states to validity.

We formalize this regime by introducing *registry-governed stream processing*, where an external registry of invariant predicates defines valid states and a compensation operator repairs violations. We construct a *governance rewrite system* whose configurations pair a state with a buffer of pending events, and whose rewrite rules include both event application (in any causality-respecting order) and compensation. Two structural conditions—*well-founded compensation* (WFC) and *compensation commutativity* (CC)—ensure termination and confluence respectively. Under both, Newman's Lemma guarantees unique normal forms: any two processors consuming the same events agree on a valid state, regardless of the order in which they apply those events. A third condition—*uniformly bounded compensation* (UBC)—yields constant per-event overhead and $O(\log n)$ total cost matching conventional stream processing. We show UBC is tight: without it, compensation depth grows without limit. We show CC is necessary: without it, different orders can reach different valid states.

Compensation commutativity is strictly weaker than operation commutativity: operations need not commute, only their compensated results. The gap between these two conditions is precisely the design space that normalization confluence occupies.

To make CC practically verifiable, we develop a *verification calculus* that reduces the global CC obligation to local checks. An *invariant footprint* mapping identifies which invariants each event can affect, and a *decomposable repair* condition ensures per-invariant repairs commute. Together, these yield sufficient conditions for CC that can be checked per-event-pair rather than globally, and a product composition theorem that enables modular verification of multi-registry systems.

Keywords: Stream Processing, Normalization Confluence, Semantic Consistency, Rewrite Systems, Newman's Lemma, Compensation Commutativity, Distributed Systems

1   Introduction

Coordination-free convergence—the ability of distributed processors to agree on a result without explicit synchronization—is among the central goals of distributed systems design. The literature has identified two structural regimes under which it is achievable:

1. **Operation commutativity** (CRDTs [2]): Operations are designed to commute algebraically. Any application order produces the same state.
2. **Invariant confluence** (I-confluence [5]): Operations individually preserve application invariants under all orderings. No repair is needed because validity is never violated.

Both regimes achieve convergence by *preventing* inconsistency. CRDTs prevent it through algebraic design; I-confluence prevents it through invariant-preserving operation selection. Neither accommodates operations that are non-commutative *and* may violate invariants—yet such operations are pervasive in practice. An order processing pipeline, for example, must handle events (orders, payments, inventory adjustments, shipments) that do not commute and that individually violate business rules (no shipment without payment, no negative inventory).

We formalize the missing piece as a *registry*: any authoritative definition of validity external to the stream processor. Registries already exist in practice under various names—policy engines, constraint services, schema validators, compliance controllers—but their role in convergence has not been characterized formally. Our model abstracts these as a tuple of invariant predicates and a compensation operator.

This paper identifies a third regime: **normalization confluence**. Operations may be non-commutative. They may violate invariants. But a compensation operator normalizes states to validity after each operation, and the key structural condition is that these *compensated results* commute—even though the operations themselves do not. Under this condition, together with well-founded compensation, Newman's Lemma guarantees that all causality-respecting processing orders reach the same valid state.

The three regimes form a hierarchy of progressively weaker requirements:

1. Operation commutativity $\implies$ compensation commutativity $\implies$ convergence.
2. Invariant confluence $\implies$ no compensation needed $\implies$ convergence.
3. Compensation commutativity alone $\implies$ convergence (this paper).

Normalization confluence occupies the design space between CRDT commutativity (which requires operations to commute) and I-confluence (which requires operations to preserve invariants). It permits operations that satisfy neither, provided their compensated results agree.

1.1 Convergence via Normalization

Convergence is conventionally treated as a property of *operations*: operations must be designed to commute, or designed to preserve invariants. Our result reframes convergence as a property of the *normalization rewrite system*: the system that interleaves event application with invariant repair. The operations themselves need no special algebraic structure. What matters is that the rewrite system—encompassing both application and compensation—is terminating and confluent. This shifts the design burden from operation algebra to compensation design.

1.2 Contributions

1. **Normalization confluence regime** (Section 3): A governance rewrite system on configurations (σ, B) pairing a state with a pending-event buffer, with rules for both event application and compensation. Nondeterminism arises from the choice of which enabled event to apply next.
2. **Convergence theorem** (Section 5): Proof that processors consuming the same events converge to the same valid state, regardless of application order, under well-founded compensation (WFC) and compensation commutativity (CC).

3. **Necessity of both conditions** (Section 6): Counterexamples demonstrating that without UBC, compensation depth is unbounded, and without CC, different orders reach different valid states.
4. **Complexity analysis** (Section 7): Proof that registry governance adds $O(\log n)$ overhead per event under UBC, preserving practical scalability.
5. **Three-regime characterization** (Section 8): Together with CRDTs and invariant confluence, a partition of the coordination-free design space into operation commutativity, invariant confluence, and normalization confluence.
6. **Verification calculus** (Section 9): Sufficient conditions—invariant footprints and decomposable repair—that reduce the global CC obligation to local, per-event-pair checks. Product composition theorems enable modular verification of multi-registry systems.

2   Related Work

Conflict-Free Replicated Data Types.   CRDTs [2] guarantee convergence through algebraic structure: replicas that apply the same set of commutative, associative, idempotent operations converge to the same state. Our setting is strictly more general in one dimension: we do not require operations to commute, only that their *compensated results* commute. This permits operations that individually violate invariants, which CRDTs cannot accommodate. Conversely, CRDTs require no finiteness or boundedness assumptions. Neither approach dominates: CRDTs are preferable when operations can be designed to commute, while registry governance is preferable when business rules constrain valid states in ways that preclude commutativity.

CALM and Coordination Avoidance.   The CALM theorem [3, 4] establishes that monotone programs can achieve consistency without coordination. Bailis et al. [5] extend this to identify *invariant confluence*: a set of operations is invariant-confluent if all possible orderings preserve application invariants. When invariant confluence holds, no compensation is needed. Our framework addresses the complement: operations that are *not* invariant-confluent. We prove that compensation restores convergence under bounded depth and commutativity. Together, invariant confluence and our result partition the design space (Section 8.1).

Abstract Rewrite Systems.   Newman's Lemma [1] states that a terminating abstract rewrite system is confluent if and only if it is locally confluent. The lemma has been applied extensively in term rewriting [6] and lambda calculus. Our contribution is its application to a rewrite system with *genuine nondeterminism*: the choice of which event to apply next from a buffer of enabled events. Local confluence is nontrivial because different application orders produce different intermediate states; our compensation commutativity axiom is the condition that closes the resulting critical pairs.

Eventual Consistency.   Classical eventual consistency [7] guarantees that replicas converge given sufficient time without new updates. It does not address whether converged states satisfy application-level invariants—only that replicas agree on the same value. Our work guarantees convergence to *valid* states under continuous event arrival.

Stream Processing Theory.   Formal models of stream processing include synchronous dataflow [9], Kahn process networks [10], and the windowed computation models underlying modern engines [11]. These focus on operational semantics—what a stream processor computes—rather

than semantic validity of computed states. Our registry layer sits above any of these operational models.

Saga Pattern and Compensating Transactions. The saga pattern [8] decomposes long-lived transactions into sequences of local transactions with compensating actions for rollback. Our compensation operator generalizes this: rather than undoing individual operations, it repairs arbitrary invariant violations in the current state. The convergence guarantee we prove—that different interleaving orders of application and compensation reach the same valid state—has no analogue in the saga literature.

3 Registry-Governed Stream Model

3.1 Registries and Semantic Projection

Definition 3.1 (Registry). A *registry* is a tuple $R = (\Sigma_R, I_R, V_R, \rho_R)$ where:

- Σ_R is a set of *semantic states*,
- $I_R = \{\psi_1, \ldots, \psi_k\}$ is a finite set of *invariant predicates* $\psi_i : \Sigma_R \to \{\top, \bot\}$,
- $V_R : \Sigma_R \to \{\top, \bot\}$ is the *validation function* defined by $V_R(\sigma) = \bigwedge_{i=1}^{k} \psi_i(\sigma)$,
- $\rho_R : \Sigma_R \to \Sigma_R$ is a *compensation operator* satisfying: if $V_R(\sigma) = \bot$, then $\Phi(\rho_R(\sigma)) < \Phi(\sigma)$ for a well-founded measure Φ (defined below); if $V_R(\sigma) = \top$, then $\rho_R(\sigma) = \sigma$.

Definition 3.2 (Semantic Projection). A registry is defined over a semantic state space Σ_R together with a *projection* $\pi : \Sigma_{\text{raw}} \to \Sigma_R$ from unbounded raw states to semantic states. All invariants, validation, and compensation are defined on Σ_R. Two raw states s_1, s_2 are *registry-indistinguishable* if $\pi(s_1) = \pi(s_2)$. The application function apply $: \mathcal{E} \times \Sigma_R \to \Sigma_R$ is understood as "apply then project": given a raw-level application $\text{apply}_{\text{raw}} : \mathcal{E} \times \Sigma_{\text{raw}} \to \Sigma_{\text{raw}}$, the semantic application is $\text{apply}(e, \sigma) = \pi(\text{apply}_{\text{raw}}(e, s))$ for any s with $\pi(s) = \sigma$. Well-definedness requires that the projected result is independent of the representative s.

Remark 3.3 (Scope of Finiteness). Finiteness is required of the *semantic* state space Σ_R, not the raw data. An order-processing registry tracking status across states *pending, paid, shipped, delivered, cancelled* has $|\Sigma_R| = 5$ regardless of how many bytes the underlying order record occupies. More generally, any domain whose logic is captured by k boolean predicates has $|\Sigma_R| \leq 2^k$: the semantic states are the equivalence classes under the invariant vector $(\psi_1(\sigma), \ldots, \psi_k(\sigma))$. Business domains are overwhelmingly of this form.

Definition 3.4 (Registry Equivalence). Two states $\sigma_1, \sigma_2 \in \Sigma_R$ are *registry equivalent*, written $\sigma_1 \approx_R \sigma_2$, if they satisfy the same invariants: $\forall \psi \in I_R : \psi(\sigma_1) = \psi(\sigma_2)$.

Lemma 3.5. *Registry equivalence $\approx_R$ is an equivalence relation on Σ_R.*

Proof. Each condition $\psi(\sigma_1) = \psi(\sigma_2)$ is reflexive, symmetric, and transitive. A conjunction of equivalence relations is an equivalence relation. $\square$

3.2 Well-Founded and Uniformly Bounded Compensation

Termination and complexity require two properties of the compensation operator, which we state separately because they play different roles.

Axiom 3.6 (Well-Founded Compensation (WFC)). There exists a *compensation measure* $\Phi :$ $\Sigma_R \to \mathbb{N}$ such that every compensation step strictly decreases Φ: if $V_R(\sigma) = \bot$, then $\Phi(\rho_R(\sigma)) < \Phi(\sigma)$.

WFC guarantees that compensation terminates from any state: iterated application of ρ_R reaches a valid state in finitely many steps.

Axiom 3.7 (Uniformly Bounded Compensation (UBC)). The compensation measure Φ from Theorem 3.6 is uniformly bounded: there exists $M \in \mathbb{N}$ such that $\Phi(\sigma) \leq M$ for all $\sigma \in \Sigma_R$.

UBC strengthens WFC by bounding compensation depth *uniformly* across all states. WFC alone suffices for termination and for defining iterated compensation ρ_R^*. UBC is additionally needed for the uniform step bound in the termination lemma and the constant per-event compensation cost in the complexity analysis.

Lemma 3.8 (Finite State Spaces Imply UBC). *If $|\Sigma_R| < \infty$ and WFC holds, then UBC holds with $M = \max_{\sigma \in \Sigma_R} \Phi(\sigma)$.*

Proof. The maximum exists because Σ_R is finite. $\qquad\square$

Remark 3.9. WFC is the load-bearing condition for termination; UBC is the load-bearing condition for uniform step bounds and per-event complexity. Finiteness of Σ_R is a convenient sufficient condition for both, but UBC does not require full finiteness of Σ_R; it requires only bounded compensation height. The counterexample in Section 6 satisfies WFC (compensation terminates for every fixed event set) but violates UBC (compensation depth grows without bound as event sets grow).

3.3 Events and Streams

Definition 3.10 (Event Stream). An *event stream* $\mathcal{S} = \langle e_1, e_2, \ldots \rangle$ is a (possibly infinite) sequence of events drawn from event space $\mathcal{E}$. Each event e_i carries a unique identifier $\mathrm{id}(e_i)$, a timestamp $\tau(e_i)$, and a set of causal dependencies $\mathrm{deps}(e_i) \subseteq \{e_1, \ldots, e_{i-1}\}$. An *application function* $\mathrm{apply} : \mathcal{E} \times \Sigma_R \to \Sigma_R$ maps each event and current state to a new state.

Definition 3.11 (Causality-Respecting Order). An ordering π of a finite event set E *respects causality* if whenever $e_i \in \mathrm{deps}(e_j)$, event e_i precedes e_j under π. Multiple causality-respecting orders may exist for the same event set.

3.4 Governance Rewrite System

We now define the rewrite system that is the core technical device of this paper. Unlike the approach of fixing a canonical event order—which would make agreement trivial by determinism—our rewrite system admits genuine nondeterminism in the choice of which event to apply next.

Assumption 3.12 (Causal Closure). The received event set E is causally closed: for every $e \in E$, $\mathrm{deps}(e) \subseteq E$.

Under causal closure, every dependency of every event in E is either still pending in B or has already been applied (removed from B). This makes the enabled predicate below well-defined.

Definition 3.13 (Configuration). A *configuration* is a pair (σ, B) where $\sigma \in \Sigma_R$ is a semantic state and $B \subseteq E$ is a set of events that have been received but not yet applied. An event $e \in B$ is *enabled* if $\deps(e) \cap B = \varnothing$ (all causal dependencies of e have already been applied).

Definition 3.14 (Governance Rewrite System). Fix a finite set of received events E and initial state σ_0. The *governance rewrite system* $\mathcal{G} = (\mathcal{C}, \rightarrow)$ operates on configurations (σ, B) with $\sigma \in \Sigma_R$ and $B \subseteq E$. The rewrite relation $\rightarrow$ is the union of:

1. **Apply step:** $(\sigma, B) \rightarrow (\mathrm{apply}(e, \sigma), B \setminus \{e\})$ whenever $e \in B$ is enabled.
2. **Compensation step:** $(\sigma, B) \rightarrow (\rho_R(\sigma), B)$ whenever $V_R(\sigma) = \bot$.

A configuration is in *normal form* if $B = \varnothing$ and $V_R(\sigma) = \top$.

The nondeterminism is genuine: when multiple events are enabled and the state is invalid, a configuration may step via different apply rules (choosing different events) or via the compensation rule. Confluence of this system means that the choice does not affect the final result.

3.5 Compensation Commutativity

Local confluence of $\mathcal{G}$ is nontrivial. Two critical pair types arise: (1) two apply steps choosing different events, and (2) an apply step versus a compensation step. The following axiom ensures both types close.

Definition 3.15 (Iterated Compensation). Under WFC (Theorem 3.6), define $\rho_R^*(\sigma)$ to be the unique state obtained by iterating ρ_R from σ until reaching validity: $\rho_R^*(\sigma) = \rho_R^m(\sigma)$ for the least m such that $V_R(\rho_R^m(\sigma)) = \top$. Existence and finiteness of m follow from WFC; uniqueness follows from determinism of ρ_R.

Axiom 3.16 (Compensation Commutativity (CC)). The registry R satisfies:

(CC1) Order independence. For any $\sigma \in \Sigma_R$ and events e_1, e_2 that are causally independent and simultaneously enabled in some configuration (σ, B):

$$\rho_R^*(\mathrm{apply}(e_2, \rho_R^*(\mathrm{apply}(e_1, \sigma)))) = \rho_R^*(\mathrm{apply}(e_1, \rho_R^*(\mathrm{apply}(e_2, \sigma)))).$$

(CC2) Compensation absorption. For any $\sigma \in \Sigma_R$ with $V_R(\sigma) = \bot$ and any enabled event e:

$$\rho_R^*(\mathrm{apply}(e, \sigma)) = \rho_R^*(\mathrm{apply}(e, \rho_R(\sigma))).$$

Remark 3.17 (Relationship to CRDT Commutativity). CRDT commutativity requires $\mathrm{apply}(e_1, \mathrm{apply}(e_2, \sigma)) = \mathrm{apply}(e_2, \mathrm{apply}(e_1, \sigma))$ for all σ. This implies CC1 (take $\rho_R^* = \mathrm{id}$ when all states are valid) and trivially satisfies CC2 (when all states are valid, compensation never fires). Compensation commutativity is strictly weaker: it permits operations that individually violate invariants, requiring only that the *repaired* results commute. The gap between operation commutativity and compensation commutativity is precisely the space our result occupies.

Remark 3.18 (Compensation Absorption). Condition CC2 states that compensating an invalid state before applying an event does not change the final compensated result. Intuitively, the repair that compensation performs "would have been performed anyway" during the post-application compensation phase. CC2 holds whenever compensation repairs invariant violations in a way that is determined by the violation pattern rather than the path taken to reach the violation. By induction on the number of compensation steps, CC2 generalizes: $\rho_R^*(\mathrm{apply}(e, \sigma)) = \rho_R^*(\mathrm{apply}(e, \rho_R^*(\sigma)))$. In rewrite-theoretic terms, CC2 ensures that the normalizer is stable under interleaving with event application: normalizing "early" (before an event) versus "late" (after) does not change the final normal form.

3.6 Stream Processors

Definition 3.19 (Stream Processor). A *stream processor* P for stream $\mathcal{S}$ under registry R is a process that, at each time t, has received some subset of events $E_P(t) \subseteq \mathcal{S}$ and maintains state $\Psi_P(t) \in \Sigma_R$. We require:

1. **Eventual delivery**: For every event $e_i \in \mathcal{S}$, there exists a time t_i such that $e_i \in E_P(t)$ for all $t \geq t_i$.
2. **Causality-respecting application**: The processor applies events in some order that respects causality (Theorem 3.11), buffering events whose causal dependencies have not yet been received. The processor is free to choose any causality-respecting order.
3. **Incremental compensation**: After applying each event, the processor applies ρ_R until validity is restored.

The processor's computation on event set $E_P(t)$ is a reduction sequence in the governance rewrite system $\mathcal{G}$ starting from $(\sigma_0, E_P(t))$.

Different processors may apply the same events in different causality-respecting orders. The convergence theorem guarantees they nonetheless arrive at the same valid state.

4 Worked Example: Order Fulfillment

We instantiate the framework on an order fulfillment pipeline, verify the structural conditions, and trace two processor orderings to the same valid state.

4.1 Registry Definition

An order has a *stage* in $\{pending, approved, held\}$ and a *balance* in $\{0, 1, 2, 3\}$. The semantic state space is $\Sigma_R = \{pending, approved, held\} \times \{0, 1, 2, 3\}$, with $|\Sigma_R| = 12$ and initial state $\sigma_0 = (pending, 0)$.

A single invariant governs fulfillment:

$$\psi(s, b) = \top \iff (s = approved \implies b \geq 2).$$

Approval requires a balance of at least 2. The only invalid states are $(approved, 0)$ and $(approved, 1)$.

Compensation. When an order is approved with insufficient balance, it is placed on hold:

$$\rho(approved, b) = (held, b) \quad \text{for } b < 2.$$

All other states are valid, and ρ acts as the identity. The *held* stage preserves the intent to approve: the system records that approval was requested but cannot yet proceed. This is a deliberate design choice—a naive alternative that simply cancels the approval (reverting to *pending*) would violate CC, as we discuss below.

Measures. $\Phi(s, b) = 1$ if $s = approved$ and $b < 2$, otherwise 0. WFC holds (ρ strictly decreases Φ on invalid states), and UBC holds with $M = 1$.

4.2 Events

Two causally independent events:

- e_{credit}: Adds 1 to the balance (capped at 3). If the order is on hold and the new balance reaches 2, the order is automatically approved:

$$\text{apply}(e_{\text{credit}}, (s, b)) = \begin{cases} (approved, \min(b+1, 3)) & \text{if } s = held \text{ and } \min(b+1, 3) \geq 2, \\ (s, \min(b+1, 3)) & \text{otherwise.} \end{cases}$$

- e_{approve}: Sets the stage to $approved$: $\text{apply}(e_{\text{approve}}, (s, b)) = (approved, b)$.

4.3 Verifying CC

CC1 (Order Independence). We verify CC1 for $(e_{\text{approve}}, e_{\text{credit}})$ at two representative starting states.

From $(pending, 0)$:

Path 1 (e_{approve} first):

$$(pending, 0) \xrightarrow{e_{\text{approve}}} (approved, 0) \xrightarrow{\rho^*} (held, 0) \xrightarrow{e_{\text{credit}}} (held, 1). \quad \text{Result: } (held, 1).$$

Path 2 (e_{credit} first):

$$(pending, 0) \xrightarrow{e_{\text{credit}}} (pending, 1) \xrightarrow{e_{\text{approve}}} (approved, 1) \xrightarrow{\rho^*} (held, 1). \quad \text{Result: } (held, 1). \checkmark$$

In Path 1, approval fires first, violates the invariant (balance $0 < 2$), and compensation places the order on hold. The subsequent credit increments the balance but does not yet reach the threshold. In Path 2, credit arrives first (no violation), then approval violates the invariant and compensation again produces the held state. Both paths reach $(held, 1)$.

From $(pending, 1)$:

Path 1: $(pending, 1) \xrightarrow{e_{\text{approve}}} (approved, 1) \xrightarrow{\rho^*} (held, 1) \xrightarrow{e_{\text{credit}}} (approved, 2).$ Result: $(approved, 2)$.

Path 2: $(pending, 1) \xrightarrow{e_{\text{credit}}} (pending, 2) \xrightarrow{e_{\text{approve}}} (approved, 2).$ Result: $(approved, 2)$. $\checkmark$

In Path 1, approval again violates the invariant; compensation holds; the subsequent credit reaches balance 2, triggering auto-approval. In Path 2, the credit arrives first, and approval at balance 2 is valid. Both reach $(approved, 2)$: the order is approved with sufficient funds.

Since $|\Sigma_R| = 12$, CC1 can be verified exhaustively for all valid starting states. All cases follow the same pattern: compensation preserves the approval intent in the held state, and subsequent credits eventually satisfy the balance threshold.

CC2 (Compensation Absorption). From the invalid state $(approved, 0)$ with event e_{credit}:

LHS: $\rho^*(\text{apply}(e_{\text{credit}}, (approved, 0))) = \rho^*((approved, 1)) = (held, 1)$.
RHS: $\rho^*(\text{apply}(e_{\text{credit}}, \rho(approved, 0))) = \rho^*(\text{apply}(e_{\text{credit}}, (held, 0))) = \rho^*((held, 1)) = (held, 1)$. $\checkmark$

Compensating before the credit (putting the order on hold first) produces the same result as applying the credit to the invalid state and then compensating. CC2 holds because the held state and the approved-but-invalid state respond identically to subsequent credits after compensation.

4.4 Two Processors, Three Events

Consider the stream $\mathcal{S} = \langle e_{\text{approve}}, e_{\text{credit}}, e_{\text{credit}} \rangle$ (approval followed by two credits). Two processors P_1, P_2 receive the events in different orders:

Processor P_1 (approve, credit, credit):

$$(pending, 0) \xrightarrow{e_{\text{approve}}} (approved, 0) \xrightarrow{\rho^*} (held, 0) \xrightarrow{e_{\text{credit}}} (held, 1) \xrightarrow{e_{\text{credit}}} (approved, 2).$$

Processor P_2 (credit, credit, approve):

$$(pending, 0) \xrightarrow{e_{\text{credit}}} (pending, 1) \xrightarrow{e_{\text{credit}}} (pending, 2) \xrightarrow{e_{\text{approve}}} (approved, 2).$$

Both processors arrive at $(approved, 2)$. P_1 encountered an invariant violation (approval without sufficient balance) and passed through the held state; P_2 never violated the invariant. Compensation commutativity guarantees agreement regardless of the path.

4.5 Why Compensation Design Matters

The held stage is essential. A naive compensation that simply reverts approval to *pending*—i.e., $\rho(approved, b) = (pending, b)$—would violate CC1. From $(pending, 0)$, approval-then-credit would produce $(pending, 1)$ (intent to approve lost), while credit-then-approval would produce $(held, 1)$ or a different compensated state depending on the design. The held stage preserves the approval intent so that subsequent credits can trigger auto-approval, ensuring that different orderings converge.

This illustrates the practical content of the CC conditions: they constrain *compensation design*, not operation design. The sufficient condition used here is *state-determined compensation*: ρ^* depends only on which invariants are violated (an approved order with low balance is always placed on hold), making the post-compensation state independent of how the violation was reached.

5 Convergence Theorem

5.1 Termination

Lemma 5.1 (Termination)**.** *Under WFC (Theorem 3.6), the governance rewrite system $\mathcal{G}$ is terminating: every reduction sequence from (σ_0, E) is finite. Under UBC (Theorem 3.7), the total number of steps is bounded by $|E| + (|E| + 1) \cdot M$.*

Proof. Define the measure $\mu(\sigma, B) = (|B|, \Phi(\sigma))$ taking values in $\mathbb{N} \times \mathbb{N}$ under the lexicographic order, which is well-founded.

An apply step sends (σ, B) to $(\text{apply}(e, \sigma), B \setminus \{e\})$. The first component decreases: $|B \setminus \{e\}| = |B| - 1$. Therefore μ strictly decreases regardless of the second component.

A compensation step sends (σ, B) to $(\rho_R(\sigma), B)$. The first component is unchanged; the second strictly decreases by WFC. Therefore μ strictly decreases.

Since μ maps into a well-ordered set and strictly decreases at every step, every reduction sequence is finite. Under UBC, $\Phi(\sigma) \leq M$ for all σ, so at most M compensation steps can occur between apply steps (including before the first apply), giving the uniform bound $|E| + (|E| + 1) \cdot M$. $\quad\square$

5.2 Local Confluence

Lemma 5.2 (Local Confluence)**.** *Under CC (Theorem 3.16), the governance rewrite system $\mathcal{G}$ is locally confluent: if $(\sigma, B) \to X_1$ and $(\sigma, B) \to X_2$, then there exists a configuration Z with $X_1 \to^* Z$ and $X_2 \to^* Z$.*

Proof. Three cases arise from the two rewrite rules.

Case 1: Two apply steps. $(\sigma, B) \to (\text{apply}(e_1, \sigma), B \backslash \{e_1\}) = X_1$ and $(\sigma, B) \to (\text{apply}(e_2, \sigma), B \backslash \{e_2\}) = X_2$, where $e_1 \neq e_2$ are both enabled.

Since e_1 and e_2 are both enabled, their dependencies are disjoint from B. By Theorem 3.12, $e_1, e_2 \in E$, and since $e_1 \in B$ we have $e_1 \notin \text{deps}(e_2)$ (otherwise $\text{deps}(e_2) \cap B \neq \varnothing$, contradicting enabledness of e_2), and symmetrically $e_2 \notin \text{deps}(e_1)$. Therefore e_1 and e_2 are causally independent.

From X_1: compensate to validity, obtaining $(\rho_R^*(\text{apply}(e_1, \sigma)), B \setminus \{e_1\})$; then apply e_2 (still enabled since its dependencies remain outside $B \setminus \{e_1\}$) and compensate, reaching

$$Z_1 = \left(\rho_R^*(\text{apply}(e_2, \rho_R^*(\text{apply}(e_1, \sigma)))), \ B \setminus \{e_1, e_2\} \right).$$

From X_2: symmetrically, reach

$$Z_2 = \left(\rho_R^*(\text{apply}(e_1, \rho_R^*(\text{apply}(e_2, \sigma)))), \ B \setminus \{e_1, e_2\} \right).$$

By CC1, the state components of Z_1 and Z_2 are equal. Take $Z = Z_1 = Z_2$.

Case 2: Apply vs. compensation. $V_R(\sigma) = \bot$ and e is enabled in B. $(\sigma, B) \to (\text{apply}(e, \sigma), B \setminus \{e\}) = X_1$ and $(\sigma, B) \to (\rho_R(\sigma), B) = X_2$.

From X_1: compensate to validity, reaching $(\rho_R^*(\text{apply}(e, \sigma)), B \setminus \{e\})$.

From $X_2 = (\rho_R(\sigma), B)$: apply e (still enabled, since compensation does not change B) to get $(\text{apply}(e, \rho_R(\sigma)), B \setminus \{e\})$; then compensate to validity, reaching $(\rho_R^*(\text{apply}(e, \rho_R(\sigma))), B \setminus \{e\})$.

By CC2, $\rho_R^*(\text{apply}(e, \sigma)) = \rho_R^*(\text{apply}(e, \rho_R(\sigma)))$. The buffer components are equal. Take $Z = (\rho_R^*(\text{apply}(e, \sigma)), B \setminus \{e\})$.

Case 3: Two compensation steps. Both rewrites apply ρ_R to the same state, producing the same successor $(\rho_R(\sigma), B)$. Take $Z = (\rho_R(\sigma), B)$. $\qquad \square$

5.3 Main Result

Corollary 5.3 (Unique Normal Forms)**.** *Under WFC and CC, every configuration (σ_0, E) has a unique normal form.*

Proof. By Theorem 5.1, $\mathcal{G}$ is terminating. By Theorem 5.2, $\mathcal{G}$ is locally confluent. By Newman's Lemma [1], $\mathcal{G}$ is globally confluent. A terminating, confluent rewrite system has unique normal forms. $\qquad \square$

Theorem 5.4 (Stream Convergence)**.** *Let R be a registry satisfying WFC (Theorem 3.6) and CC (Theorem 3.16), and let P_1, P_2 be stream processors (Theorem 3.19) consuming the same event stream $\mathcal{S}$. Then:*

(a) **Validity**: *For each processor, after it completes the incremental compensation phase for any applied event, its state satisfies $V_R(\Psi_{P_j}(t)) = \top$.*

(b) **Order independence**: *The state of each processor depends only on its received event set, not on the order in which events were applied.*

(c) **Agreement**: *Whenever the processors have received the same events, they are in the same state: if $E_{P_1}(t) = E_{P_2}(t)$, then $\Psi_{P_1}(t) = \Psi_{P_2}(t)$.*

Combined with eventual delivery, disagreements are transient: any event causing processors to differ will eventually be received by both, restoring agreement.

Proof. We prove each part.

Part (a): Validity. By Theorem 5.1, compensation terminates. By Theorem 3.19, the processor compensates after each event application until validity is restored. Therefore after completing the compensation phase for any applied event, $V_R(\Psi_{P_j}(t)) = \top$.

Part (b): Order independence. Each processor's computation on event set $E = E_{P_j}(t)$ is a reduction sequence in the governance rewrite system from (σ_0, E) to a normal form $(\sigma^*, \varnothing)$. Different causality-respecting orders correspond to different reduction sequences. By Theorem 5.3, all reduction sequences reach the same normal form. Therefore the final state σ^* depends only on E, not on the order.

Part (c): Agreement. If $E_{P_1}(t) = E_{P_2}(t) = E$, then both processors' computations are reduction sequences from (σ_0, E). By Theorem 5.3, both reach the same normal form $(\sigma^*, \varnothing)$. Therefore $\Psi_{P_1}(t) = \Psi_{P_2}(t) = \sigma^*$. $\square$

Remark 5.5 (Semantic Projection as a Set Function). The theorem implies that the mapping $F(E) = \mathrm{nf}(\sigma_0, E)$—the unique normal form of the governance rewrite system starting from (σ_0, E)—is a well-defined function of the event *set* E, independent of causal linearization. This is the Church–Rosser property for our operational semantics.

Corollary 5.6 (Convergence Under Quiescent Streams)**.** *If the stream $\mathcal{S}$ is finite, then there exists a valid state σ^* such that both processors converge to σ^*: there exists T with $\Psi_{P_1}(t) = \Psi_{P_2}(t) = \sigma^*$ for all $t \geq T$.*

Proof. By eventual delivery, there exists T_N such that $E_{P_1}(t) = E_{P_2}(t) = \mathcal{S}$ for all $t \geq T_N$. By Theorem 5.4(c), both processors are in the same state. Since no new events arrive, this state is fixed from T_N onward. $\square$

Remark 5.7 (Infinite Streams). For infinite streams, the received event sets grow monotonically toward $\mathcal{S}$ but may differ at any given time. When event sets coincide, Theorem 5.4(c) guarantees agreement. Eventual delivery ensures that any event causing disagreement is transient. The agreed-upon state may continue to change as new events arrive—the guarantee is perpetual agreement, not convergence to a fixed point. Theorem 5.6 recovers fixed-point convergence when the stream is finite.

6   Necessity of the Structural Conditions

We show that both UBC and CC are necessary for their respective guarantees: UBC for uniform complexity, CC for agreement.

6.1 Necessity of UBC

UBC is not merely convenient; it is necessary for a uniform bound on compensation depth, and thus for the constant per-event compensation overhead used in the complexity analysis (Section 7). Without it, compensation depth can grow without limit as event sets grow.

Theorem 6.1 (Unbounded Compensation Depth Without UBC). *For any $M \in \mathbb{N}$, there exists a registry R_∞ and a finite event set E such that some reduction sequence from (σ_0, E) in the governance rewrite system contains more than M compensation steps. Consequently, no uniform bound on compensation depth exists, and UBC fails.*

Proof. Define R_∞ with $\Sigma_{R_\infty} = \mathbb{Z}$, the single invariant $\psi(\sigma) = \top \iff \sigma = 0$, initial state $\sigma_0 = 0$, and compensation $\rho(\sigma) = \sigma + 1$ for $\sigma < 0$ and $\rho(\sigma) = \sigma - 1$ for $\sigma > 0$ (so compensation moves toward 0).

For each $n \in \mathbb{N}$, consider the single-event set $E_n = \{e_n\}$ where $\mathrm{apply}(e_n, 0) = -n$. Under the processor model (Theorem 3.19), the processor applies e_n to reach state $-n$, then compensates incrementally: $-n \to -(n-1) \to \cdots \to -1 \to 0$. This requires exactly n compensation steps.

For any proposed bound M, choose $n = M + 1$. The compensation episode after this single event contains $n > M$ steps. Therefore no finite M uniformly bounds compensation depth, and UBC fails.

The measure $\Phi(\sigma) = |\sigma|$ strictly decreases under ρ, so WFC holds and compensation terminates for each fixed E_n. The pathology is not non-termination but *unbounded* termination: the compensation depth required to restore validity grows without limit. $\qquad\square$

Remark 6.2 (What Fails). The counterexample satisfies WFC (compensation terminates for every fixed E_n) but violates UBC (compensation depth is n, unbounded). CC also holds: there is only one event per set, so order independence is vacuous, and compensation absorption holds because ρ moves deterministically toward 0. The pathology is purely one of unbounded depth, which invalidates the uniform step bound and the constant per-event compensation cost in Theorem 7.1. Finite semantic state spaces prevent this by bounding Φ (Theorem 3.8).

6.2 Necessity of CC

CC is necessary for agreement: without it, processors applying the same events in different causality-respecting orders can reach different valid states.

Proposition 6.3 (Disagreement Without CC). *There exists a registry satisfying WFC and UBC, and a finite event set E with two causality-respecting orderings, such that the governance rewrite system reaches different valid normal forms under the two orderings.*

Proof. Let $\Sigma_R = \{A, B, C, D\}$ with valid states $\{A, B\}$ and compensation $\rho_R(C) = A$, $\rho_R(D) = B$. UBC holds with $M = 1$.

Define two events e_1, e_2 with $\mathrm{deps}(e_1) = \mathrm{deps}(e_2) = \varnothing$, so both are enabled from the initial configuration $(A, \{e_1, e_2\})$ and are causally independent under Theorem 3.11. Their application is:

	A	B	C	D
e_1	C	C	C	D
e_2	D	C	C	D

Evaluate CC1 at $\sigma = A$. Under the processor model (Theorem 3.19), each processor normalizes after each application, so the comparison is between $\rho_R^*(A_{e_2}(\rho_R^*(A_{e_1}(A))))$ and the symmetric order.

Path 1 (e_1 then e_2): apply(e_1, A) $= C$, $\rho_R^*(C) = A$, apply(e_2, A) $= D$, $\rho_R^*(D) = B$. Normal form: B.

Path 2 (e_2 then e_1): apply(e_2, A) $= D$, $\rho_R^*(D) = B$, apply(e_1, B) $= C$, $\rho_R^*(C) = A$. Normal form: A.

Since $A \neq B$, CC1 is violated. Two processors consuming $\{e_1, e_2\}$ in different causality-respecting orders reach different valid states despite WFC and UBC holding. □

Remark 6.4. The key feature is that ρ_R maps distinct invalid states to *distinct* valid states ($C \mapsto A$, $D \mapsto B$), creating path-dependent repair. Contrast this with the order fulfillment example (Section 4), where compensation preserves approval intent via the held state, ensuring path-independent repair. CC captures precisely the boundary between these cases.

Remark 6.5. The UBC counterexample (Theorem 6.1) satisfies CC but violates UBC; the CC counterexample (Theorem 6.3) satisfies WFC and UBC but violates CC. Together, they show that each condition is independently necessary for its respective guarantee.

7   Complexity Analysis

Theorem 7.1 (Convergence Complexity). *For a stream prefix of length n under a registry satisfying UBC with bound M and $k = |I_R|$ invariants, the model-level cost of registry-governed stream processing is $O(n)$ for event application, validation, and compensation combined. If causality maintenance is implemented with a priority queue, the total cost is $O(n \log n)$, where k and M are treated as constants of system design.*

Proof. We analyze the per-event cost, separating model-level costs (which follow from the formalism) from scheduling costs (which depend on implementation).

Model-level per-event costs. For each event e_i ($1 \leq i \leq n$):

- *Event application*: $O(1)$ (state update).
- *Invariant validation*: $O(k) = O(1)$ (evaluate each ψ_j).
- *Compensation*: At most M steps, each $O(1)$. Since M is bounded by UBC, this is $O(1)$.

Registry governance therefore adds $O(1)$ overhead per event beyond what the base stream processor already performs.

Scheduling overhead. Maintaining a set of enabled events and resolving causal dependencies requires implementation-dependent data structures. With a priority queue keyed by timestamp and identifier, each event insertion and dependency-resolution check costs $O(\log i)$, yielding:

Total cost.

$$T(n) = \sum_{i=1}^{n} \big(\underbrace{O(1)}_{\text{model}} + \underbrace{O(\log i)}_{\text{scheduling}}\big) = O(n) + O\left(\sum_{i=1}^{n} \log i\right) = O(n \log n).$$

The dominant term is causality maintenance. Validation and compensation contribute no asymptotic overhead. Systems with native causal ordering (e.g., single-partition Kafka topics) eliminate the $O(\log i)$ term, reducing the bound to $O(n)$. □

8 Discussion

8.1 Three Regimes of Coordination-Free Convergence

Our result, combined with CRDTs [2] and invariant confluence [5], identifies three distinct structural regimes under which distributed processors converge without coordination:

1. **Operation commutativity** (CRDTs): Operations commute algebraically. Convergence follows from the algebra of operations. No invariant enforcement or compensation is needed.
2. **Invariant confluence** (I-confluence): Operations preserve application invariants under all orderings. Convergence follows because validity is never violated. No compensation is needed.
3. **Normalization confluence** (this paper): Operations may be non-commutative and may violate invariants. Compensation normalizes states to validity. WFC and CC are sufficient for semantic convergence (Theorem 5.4); UBC is orthogonal to convergence and governs scalability (Theorem 7.1). No coordination is needed.

The three regimes form a containment hierarchy. Operation commutativity implies CC (with $\rho_R^* = \mathrm{id}$). Invariant confluence implies that compensation never fires. Normalization confluence subsumes both as special cases and additionally covers systems where operations are non-commutative and invariant-violating—the common case in business applications.

The regimes partition the coordination-free design space by *where* convergence is guaranteed: in the operations (regime 1), in the invariants (regime 2), or in the normalization rewrite system (regime 3).

Remark 8.1 (Convergence as a Property of the Rewrite System). The conventional literature frames convergence as a property of operations: operations must commute, or operations must preserve invariants. Normalization confluence reframes convergence as a property of the *rewrite system* that interleaves application and compensation. The operations themselves need no special algebraic structure. This shifts the design burden from operation algebra to compensation design, and places the convergence mechanism at the semantic normalization layer—above the data-structure layer (CRDTs) and the transaction layer (I-confluence).

8.2 Verifying Compensation Commutativity

CC is easy to state but potentially hard to verify: it quantifies over all states and all independent event pairs. Section 9 develops a verification calculus that reduces CC to local, per-event-pair checks via invariant footprints and decomposable repair. Here we note three common patterns that practitioners can use as a first check before invoking the full calculus:

1. **Compensation to a canonical state.** If there exists a default valid state $\sigma_\perp$ such that $\rho_R^*(\sigma) = \sigma_\perp$ for all invalid σ, both CC1 and CC2 hold trivially. This is the "reset to safe default" pattern.
2. **Lattice-structured compensation.** If the valid states form a join-semilattice and compensation computes the join with the nearest valid state, both conditions hold by the commutativity of the join operation.
3. **Decomposable repair.** If compensation decomposes into commuting per-invariant repairs, CC can be verified via the footprint calculus of Section 9. The order fulfillment example (Section 4) uses this pattern.

8.3 Applications

The compensate-and-converge principle applies wherever systems process streams of changes under semantic constraints with bounded compensation. We highlight three domains.

Infrastructure Configuration Drift. Infrastructure-as-code tools apply streams of configuration changes to infrastructure state. Valid configurations are defined by policy. The semantic state space is finite (k boolean policy predicates yield at most 2^k equivalence classes). Remediation operators that reset violated policies to compliant defaults satisfy CC2 (compensation to a canonical state per policy). Our theorem guarantees convergence to equivalent policy-compliant states regardless of the order in which changes are applied.

Database Schema Migration. Schema migrations are event streams applied to schema state. The set of valid schema configurations is finite. Migration scripts that repair constraint violations constitute compensation operators. When such scripts resolve each constraint independently (lattice-structured compensation), CC holds and our theorem guarantees order-independent convergence.

Regulatory Compliance. Organizations processing streams of regulatory changes must maintain compliance. Compliance states are finite (each regulation is either satisfied or not). Remediation procedures that restore compliance by addressing each violated regulation independently satisfy CC. Our theorem guarantees convergence to equivalent compliance states.

8.4 Extensions

The single-registry result is the foundation for two extensions we do not attempt here. First, *federated streams* where multiple organizations maintain independent registries connected by morphisms. Second, *privacy-preserving validation* where organizations prove registry compliance via zero-knowledge proofs. Both require the single-registry convergence result as a building block.

9 Verification Calculus for CC

Compensation commutativity is a global condition: it quantifies over all states and all independent event pairs. This section develops sufficient conditions that reduce CC to local, per-event-pair checks. The key tools are an *invariant footprint* mapping and a *decomposable repair* condition.

9.1 Notation

For readability in algebraic manipulations, we write $N(\sigma) := \rho_R^*(\sigma)$ for iterated compensation to validity (Theorem 3.15). The normalizer satisfies: (i) $V_R(N(\sigma)) = \top$ for all σ; (ii) $N(\sigma) = \sigma$ if $V_R(\sigma) = \top$; (iii) $N(N(\sigma)) = N(\sigma)$ (idempotence). We write $A_e(\sigma) := \mathrm{apply}(e, \sigma)$ when convenient.

In this notation, the CC obligations become:

(CC1) $N(A_{e_2}(N(A_{e_1}(\sigma)))) = N(A_{e_1}(N(A_{e_2}(\sigma))))$ for independent e_1, e_2.
(CC2) $N(A_e(\sigma)) = N(A_e(\rho_R(\sigma)))$ for invalid σ.

9.2 Invariant Vectors and Footprints

Definition 9.1 (Invariant Vector). Define $\mathbf{v} : \Sigma_R \to \{0,1\}^k$ by $\mathbf{v}(\sigma)_i = 1 \iff \psi_i(\sigma) = \top$.

Definition 9.2 (Footprint). A *footprint* is a function $F : \mathcal{E} \to 2^{\{1,\ldots,k\}}$ such that for all σ and all $i \notin F(e)$,

$$\mathbf{v}(A_e(\sigma))_i = \mathbf{v}(\sigma)_i.$$

Intuitively, $F(e)$ is the set of invariants whose truth value can change under e. Over-approximations are sound: larger footprints make the calculus more conservative but never incorrect.

9.3 Decomposable Repair

We capture the common case where repair addresses each violated invariant independently.

Definition 9.3 (Per-Invariant Repair). A set of *per-invariant repair operators* $r_1, \ldots, r_k : \Sigma_R \to \Sigma_R$ satisfies:

(R1) **Targeted fix:** if $\psi_i(\sigma) = \bot$, then $\psi_i(r_i(\sigma)) = \top$.
(R2) **No-op on satisfied:** if $\psi_i(\sigma) = \top$, then $r_i(\sigma) = \sigma$.
(R3) **Mutual commutativity:** $r_i(r_j(\sigma)) = r_j(r_i(\sigma))$ for all i, j.

Definition 9.4 (Decomposable Normalizer). The normalizer N is *decomposable* if there exist per-invariant repairs $r_1, \ldots, r_k$ as above such that

$$N(\sigma) = r_1(r_2(\cdots r_k(\sigma) \cdots))$$

for all σ. (The order is irrelevant by mutual commutativity.)

Definition 9.5 (Repair Locality). An event e is *repair-local* with respect to footprint $F(e)$ if for all $i \notin F(e)$,

$$r_i(A_e(\sigma)) = A_e(r_i(\sigma)).$$

That is, e commutes with repairs of invariants it does not affect.

9.4 Strong Absorption Implies CC2

Theorem 9.6 (Strong Absorption). *Assume WFC and let $N = \rho_R^*$. If for all events e and states σ,*

$$N(A_e(\sigma)) = N(A_e(N(\sigma))),$$

then CC2 holds.

Proof. Let $V_R(\sigma) = \bot$. Because N iterates ρ_R to the unique valid fixpoint reachable from σ, and $\rho_R(\sigma)$ lies on the same compensation chain, we have $N(\sigma) = N(\rho_R(\sigma))$.

Applying the assumed equation at σ and at $\rho_R(\sigma)$:

$$N(A_e(\sigma)) = N(A_e(N(\sigma))) = N(A_e(N(\rho_R(\sigma)))) = N(A_e(\rho_R(\sigma))),$$

where the first and third equalities use the assumption and the middle equality uses $N(\sigma) = N(\rho_R(\sigma))$. This is CC2. $\square$

Remark 9.7. Strong absorption states: "normalizing before applying an event does not change the final normalized outcome." This holds whenever compensation repairs violations in a way determined by the violation pattern rather than by the path to the violation. The order fulfillment example (Section 4) satisfies this: the held state captures the approval intent regardless of whether the order was directly approved with low balance or arrived at the invalid state through a different path.

9.5 Footprint Disjointness Implies CC1

We prove two preparatory lemmas before the main theorem.

Lemma 9.8 (Repair-Event Commutation). *If event e is repair-local with respect to footprint $F(e)$ (Theorem 9.5), then for any index set J with $J \cap F(e) = \varnothing$ and any state σ:*

$$R_J(A_e(\sigma)) = A_e(R_J(\sigma)),$$

where $R_J = \prod_{i \in J} r_i$.

Proof. By repair locality, $r_i \circ A_e = A_e \circ r_i$ for each $i \in J$ (since $i \notin F(e)$). The result follows by induction on $|J|$: if the claim holds for $R_{J'}$ with $|J'| < |J|$, then for any $j \in J$,

$$R_J(A_e(\sigma)) = r_j(R_{J \setminus \{j\}}(A_e(\sigma))) = r_j(A_e(R_{J \setminus \{j\}}(\sigma))) = A_e(r_j(R_{J \setminus \{j\}}(\sigma))) = A_e(R_J(\sigma)),$$

using the inductive hypothesis and then $r_j \circ A_e = A_e \circ r_j$. $\qquad\square$

Lemma 9.9 (Repair Block Idempotence). *For any index set $J \subseteq \{1, \ldots, k\}$, the repair block R_J is idempotent on states that are valid on all invariants in J: if $\psi_i(\sigma) = \top$ for all $i \in J$, then $R_J(\sigma) = \sigma$. More generally, $R_J(R_J(\sigma)) = R_J(\sigma)$ for all σ.*

Proof. If $\psi_i(\sigma) = \top$ for all $i \in J$, then each $r_i(\sigma) = \sigma$ by (R2), so $R_J(\sigma) = \sigma$. For general idempotence: $R_J(\sigma)$ satisfies $\psi_i(R_J(\sigma)) = \top$ for all $i \in J$ (each r_i fixes its invariant by (R1) and does not break others by mutual commutativity and (R2)), so $R_J(R_J(\sigma)) = R_J(\sigma)$ by the first claim. $\qquad\square$

Theorem 9.10 (Footprint Commutativity). *Assume WFC and that N is decomposable via commuting per-invariant repairs $r_1, \ldots, r_k$. Let e_1, e_2 be independent events satisfying:*

(H1) Disjoint footprints: $F(e_1) \cap F(e_2) = \varnothing$.
(H2) Repair locality: each e_j is repair-local with respect to $F(e_j)$.

Then CC1 holds for (e_1, e_2).

Proof. Partition $\{1, \ldots, k\}$ into three disjoint index sets: $S_1 = F(e_1)$, $S_2 = F(e_2)$, $S_0 = \{1, \ldots, k\} \setminus (S_1 \cup S_2)$. By decomposability and mutual commutativity, $N = R_{S_0} \circ R_{S_1} \circ R_{S_2}$ in any order. We compute both sides of CC1 as chains of equalities.

Left side: $N(A_{e_2}(N(A_{e_1}(\sigma))))$.

$$
\begin{aligned}
&= N(A_{e_2}(R_{S_0}(R_{S_1}(R_{S_2}(A_{e_1}(\sigma)))))) && \text{(expand inner } N\text{)} \\
&= N(R_{S_0}(R_{S_1}(A_{e_2}(R_{S_2}(A_{e_1}(\sigma)))))) && \text{(Theorem 9.8: } S_0 \cap F(e_2) = S_1 \cap F(e_2) = \varnothing\text{)} \\
&= R_{S_0}(R_{S_1}(R_{S_2}(R_{S_0}(R_{S_1}(A_{e_2}(R_{S_2}(A_{e_1}(\sigma)))))))) && \text{(expand outer } N\text{)} \\
&= R_{S_0}(R_{S_1}(R_{S_2}(A_{e_2}(R_{S_2}(A_{e_1}(\sigma)))))) && \text{(Theorem 9.9: } R_{S_0}, R_{S_1} \text{ absorbed)}
\end{aligned}
$$

Right side: $N(A_{e_1}(N(A_{e_2}(\sigma))))$.

$$
\begin{aligned}
&= N(A_{e_1}(R_{S_0}(R_{S_2}(R_{S_1}(A_{e_2}(\sigma)))))) && \text{(expand inner } N\text{)} \\
&= N(R_{S_0}(R_{S_2}(A_{e_1}(R_{S_1}(A_{e_2}(\sigma)))))) && \text{(Theorem 9.8: } S_0 \cap F(e_1) = S_2 \cap F(e_1) = \varnothing\text{)} \\
&= R_{S_0}(R_{S_1}(R_{S_2}(R_{S_0}(R_{S_2}(A_{e_1}(R_{S_1}(A_{e_2}(\sigma)))))))) && \text{(expand outer } N\text{)} \\
&= R_{S_0}(R_{S_1}(R_{S_2}(A_{e_1}(R_{S_1}(A_{e_2}(\sigma)))))) && \text{(Theorem 9.9: } R_{S_0}, R_{S_2} \text{ absorbed)}
\end{aligned}
$$

Equating. The left side is $R_{S_0}(R_{S_1}(R_{S_2}(A_{e_2}(R_{S_2}(A_{e_1}(\sigma))))))$ and the right side is $R_{S_0}(R_{S_1}(R_{S_2}(A_{e_1}(R_{S_1}(A_{e_2}(\sigma))$
It remains to show their inner expressions are equal.

Apply Theorem 9.8 to move R_{S_2} past A_{e_1} on the left (since $S_2 \cap F(e_1) = \varnothing$):

$$A_{e_2}(R_{S_2}(A_{e_1}(\sigma))) = A_{e_2}(A_{e_1}(R_{S_2}(\sigma))).$$

Apply Theorem 9.8 to move R_{S_1} past A_{e_2} on the right (since $S_1 \cap F(e_2) = \varnothing$):

$$A_{e_1}(R_{S_1}(A_{e_2}(\sigma))) = A_{e_1}(A_{e_2}(R_{S_1}(\sigma))).$$

Both inner expressions now have the form $A_{e_j}(A_{e_\ell}(R_?(\sigma)))$ with all repairs pushed past the events. Applying R_{S_2} to the left expression and R_{S_1} to the right, then using mutual commutativity of R_{S_1} and R_{S_2}, both sides reduce to

$$R_{S_0}(R_{S_1}(R_{S_2}(A_{e_2}(A_{e_1}(R_{S_2}(R_{S_1}(\sigma))))))).$$

Since R_{S_1} and R_{S_2} commute and each is idempotent after application (Theorem 9.9), the left and right sides are equal. $\qquad\square$

Remark 9.11 (Practical import). A practitioner does not prove CC1 by global state enumeration. Instead: (i) assign each event type an invariant footprint, (ii) verify that each event commutes with repairs outside its footprint (Theorem 9.8), (iii) check that independent event pairs have disjoint footprints. All three are local obligations. The order fulfillment example (Section 4) demonstrates this workflow: e_{credit} and e_{approve} affect disjoint aspects of the state, and the held-state repair commutes with credit application.

9.6   Product Composition

Real systems comprise multiple semantic modules. The following theorem shows CC verifications compose.

Definition 9.12 (Product Registry). Let $R_1 = (\Sigma_1, I_1, V_1, \rho_1)$ and $R_2 = (\Sigma_2, I_2, V_2, \rho_2)$ with disjoint invariant sets. Define $R_1 \otimes R_2$ by:

$$\Sigma = \Sigma_1 \times \Sigma_2, \quad V(\sigma_1, \sigma_2) = V_1(\sigma_1) \wedge V_2(\sigma_2), \quad \rho(\sigma_1, \sigma_2) = (\rho_1(\sigma_1), \rho_2(\sigma_2)).$$

Events are pairs $e = (e^{(1)}, e^{(2)})$ with application componentwise.

Theorem 9.13 (Product Lifting). *If each R_j satisfies WFC and CC, then $R_1 \otimes R_2$ satisfies WFC and CC. If each normalizer is decomposable, the product normalizer is decomposable with footprints composed by union across components.*

Proof. WFC: Use measure $\Phi(\sigma_1, \sigma_2) = \Phi_1(\sigma_1) + \Phi_2(\sigma_2)$. Each compensation step strictly decreases at least one component when invalid.

CC2: The product normalizer satisfies $N(\sigma_1, \sigma_2) = (N_1(\sigma_1), N_2(\sigma_2))$ and application is componentwise, so CC2 holds iff it holds in each component.

CC1: Both sides of the CC1 equation decompose into paired equalities in R_1 and R_2 by componentwise application and normalization.

Decomposability: Per-invariant repairs in each component act on disjoint state components and thus commute across components trivially. $\qquad\square$

Remark 9.14. Product composition enables modular system design: verify CC for each semantic module independently, then compose without re-verification. An order processing system with separate inventory, payment, and shipping registries can verify CC per-module and lift to the combined system.

9.7 Partial Decomposability

Full decomposability is a strong sufficient condition. When it does not hold globally, the calculus still provides value:

1. CC may hold for event pairs with disjoint footprints even if the full normalizer is not decomposable. The footprint theorem (Theorem 9.10) applies to any event pair satisfying its hypotheses, regardless of whether *all* pairs do.
2. When footprints overlap for some event pair, the calculus identifies precisely which pairs require further analysis. A practitioner can then either redesign compensation to be canonical for the overlapping invariants or add coordination only for the interacting event types.
3. For event types that share footprints but whose normalized outcomes agree by domain-specific reasoning (e.g., both repairs clamp to the same bound), CC1 can be verified by case analysis restricted to the overlapping states. The footprint calculus reduces the case space.

10 Conclusion

We have identified *normalization confluence* as a third structural regime for coordination-free convergence in distributed systems, alongside operation commutativity (CRDTs) and invariant confluence (I-confluence). The regime is characterized by two conditions: well-founded compensation (WFC), which ensures termination, and compensation commutativity (CC), which ensures confluence. Under both, the governance rewrite system has unique normal forms by Newman's Lemma, and all processors consuming the same events converge to the same valid state. A uniform bound on compensation depth (UBC) additionally guarantees constant per-event overhead. Both conditions are necessary: without UBC, compensation depth is unbounded; without CC, different orders reach different valid states.

To make CC practically verifiable, we developed a verification calculus that reduces the global CC obligation to local checks. Invariant footprints identify which invariants each event can affect; decomposable repair ensures per-invariant fixes commute; and product composition enables modular verification across semantic modules. Together, these convert "compensate-and-converge" from a convergence theorem into a verifiable design pattern with tractable proof obligations.

Compensation commutativity is strictly weaker than the operation commutativity required by CRDTs: operations need not commute, only their compensated results. The gap between these two conditions is precisely the design space that normalization confluence occupies—systems where operations are non-commutative, invariant-violating, but convergent under repair.

The three regimes partition the coordination-free design space by where convergence is guaranteed: in the operations, in the invariants, or in the normalization rewrite system. This last regime formalizes a pattern that practitioners use implicitly—post-application repair as a convergence mechanism—and provides both the structural conditions under which it is sound and a calculus for verifying those conditions.

Acknowledgments

References

[1] M. H. A. Newman, "On theories with a combinatorial definition of 'equivalence'," *Annals of Mathematics*, vol. 43, no. 2, pp. 223–243, 1942.

[2] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," *INRIA Research Report RR-7506*, 2011.

[3] J. M. Hellerstein, "The declarative imperative: experiences and conjectures in distributed logic," *SIGMOD Record*, vol. 39, no. 1, pp. 5–19, 2010.

[4] T. J. Ameloot, F. Neven, and J. Van den Bussche, "Relational transducers for declarative networking," *Journal of the ACM*, vol. 60, no. 2, pp. 1–38, 2013.

[5] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Coordination avoidance in database systems," *Proc. VLDB Endowment*, vol. 8, no. 3, pp. 185–196, 2014.

[6] F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.

[7] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[8] H. Garcia-Molina and K. Salem, "Sagas," *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 249–259, 1987.

[9] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[10] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing*, pp. 471–475, 1974.

[11] T. Akidau, R. Bradshaw, C. Chambers, et al., "The Dataflow Model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[12] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.