

WARP Graphs: Canonical State Evolution and Deterministic Worldlines

AIΩN FOUNDATIONS SERIES — PAPER II

James Ross

Independent Researcher

ORCID: 0009-0006-0025-7801

December 2025

© 2025 James Ross

This work is licensed under the
Creative Commons Attribution 4.0 International License (CC BY 4.0).

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may not imply endorsement by the author.

For the full legal code of the license, see:

<https://creativecommons.org/licenses/by/4.0/>

Abstract

Paper I introduced WARP graphs (WARPs) as a well-founded “graphs all the way down” state object. This paper equips WARP states with an operational semantics based on double-pushout rewriting with interfaces (DPOI) in an adhesive category of typed open graphs.

Our central result is a deterministic concurrent semantics at the level of a *tick*: for any scheduler-admissible batch of skeleton rewrites, the committed successor is independent (up to typed open graph isomorphism) of the internal *serialisation* order. We package states as two-plane objects $U = (G; \alpha, \beta)$ with a skeleton $G \in \mathbf{OGraph}_T$ and recursively attached WARPs over each vertex and edge. We prove that attachment-plane rewrites commute with skeleton publication up to canonical transport along preserved structure, formalised via a projection functor $\pi : \mathbf{WState} \rightarrow \mathbf{OGraph}_T$ equipped with a chosen reindexing (cleavage).

To connect the mathematics to deterministic replay, we isolate the scheduler as a total policy $\sigma : \mathbf{WState} \rightarrow \mathbf{Batch}$ and define a *tick receipt* whose core is a “tick event poset” recording which candidate matches were accepted and which were rejected by deterministic conflict resolution. Under tick confluence, this poset quotients to the same committed state, providing a clean bridge to provenance traces (Paper III).

Contents

1	Introduction	5
2	Preliminaries: Typed Open Graphs and DPOI Rewriting	6
2.1	Typed open graphs	6
2.2	DPOI rules and steps	6
3	WARP States as Two-Plane Objects	7
3.1	State objects	7
3.2	Morphisms of states and the projection functor	8
4	Two-Plane Operational Semantics and Ticks	9
4.1	Attachment-plane steps	9
4.2	Skeleton-plane steps and batches	9
4.3	Atomic ticks	9
4.4	Schedulers and deterministic runtimes	10
5	Footprints, Independence, and Scheduler-Admissible Batches	10
5.1	Footprints	10
5.2	Independence	10
5.3	Scheduler-admissible batches	11
6	Tick-Level Confluence on the Skeleton Plane	11
7	Deterministic Scheduling and Tick Receipts	11
7.1	Candidate matches and a deterministic policy	11
7.2	Tick receipts and tick-event posets	12
7.3	Quotienting receipts to committed state	13
8	Two-Plane Commutation and Deterministic Tick Semantics	13
8.1	No-delete/no-clone-under-descent	13
8.2	Two-plane commutation	14
9	Global Confluence	14
10	Discussion and Outlook	15
11	Notation Summary	16
A	Appendix: A Minimal Independence Example	17
	References	18

1 Introduction

Modern systems are concurrent and stateful. That combination is powerful and, left unmanaged, hostile to replay: the next state may depend on accidental interleavings, machine-specific scheduler behaviour, and untracked micro-decisions. For the AION programme, replay is not a debugging feature; it is part of the semantic contract.

Paper I defined WARP graphs as a minimal recursive state object: a finite skeleton whose vertices and edges carry attached WARPs, with finite well-founded nesting [Ros25]. The point is not “more structure for its own sake”, but compositionality: you can attach explanations, logs, optimisation traces, and higher-level artefacts directly to the structure that caused them.

This paper provides the dynamics. We model computation as categorical rewriting:

- local evolution inside attachments uses DPOI rewriting in the attachment category;
- global evolution of wiring uses DPOI rewriting on the skeleton in a typed open graph category \mathbf{OGraph}_T ;
- concurrency is constrained by an explicit independence discipline on skeleton matches.

Ticks and what “deterministic” means. A *tick* is our unit of concurrency: it groups a finite family of attachment-plane rewrites together with a scheduler-selected batch of independent skeleton rewrites. We do *not* assume there is only one possible next match; a state can admit many candidate rewrites. Determinism here is conditional and compositional:

- Given a fixed scheduler-admissible batch B , the tick commits a unique successor (up to isomorphism), independent of how B is serialised.
- Attachment updates commute with skeleton publication up to canonical transport, so the two-plane discipline is semantically robust.
- If the scheduler is a total function of state, entire runs become deterministic worldlines¹.

Receipts and internal provenance. A runtime that is deterministic still makes choices: which matches were considered, which were excluded for overlap, and why. We capture this by introducing a *tick receipt* whose core is a tick-event poset: a partial order over candidate events induced by a deterministic “left-most wins” filter. This receipt refines a tick without affecting its committed state, and it forms a natural bridge into provenance traces (Paper III).

Contributions. The specific contributions of this paper are:

1. A definition of WARP *states* as two-plane objects: a typed open graph skeleton together with attachment WARPs over each vertex and edge (Section 3).
2. A two-plane operational semantics based on DPOI rewriting, packaged into a *tick* notion that isolates concurrency from serialisation (Section 4).
3. A footprint-based independence discipline on skeleton matches, and a definition of scheduler-admissible batches (Section 5).

¹Here “worldline” means the \mathbb{N} -indexed sequence of committed tick transitions. We do not globalise the run into a single poset in Paper II; instead, each tick carries an internal event-poset receipt (Section 7), which Paper III uses as a provenance object.

4. Tick-level confluence on the skeleton plane: any two serialisations of a scheduler-admissible batch yield isomorphic successors (Section 6).
5. Deterministic scheduling as a total policy together with a tick receipt object (the tick-event poset) that records conflict resolution without changing the committed state (Section 7).
6. A two-plane commutation theorem: attachment updates commute with skeleton publication up to canonical transport, formalised via a projection $\pi : \mathbf{WState} \rightarrow \mathbf{OGraph}_T$ equipped with reindexing functors (Section 8).

Roadmap. Sections 2–4 set up DPOI rewriting, WARP states, and ticks. Section 5 defines footprints and scheduler admissibility. Section 6 proves within-tick confluence on the skeleton plane. Section 7 isolates deterministic scheduling and introduces the tick-event poset receipt. Section 8 proves two-plane commutation. Finally, Section 9 records a standard global confluence criterion and Section 10 discusses implications for replay and provenance.

2 Preliminaries: Typed Open Graphs and DPOI Rewriting

This paper assumes familiarity with algebraic graph transformation, especially DPO rewriting in adhesive categories [EL97, EEPT06, LS08]. We keep the review minimal and tailor it to the two-plane setting.

2.1 Typed open graphs

Fix a finite set T of types.

Let \mathbf{OGraph}_T denote a category of T -typed open graphs (e.g. Lack–Sobociński [LS08]): an object is an open graph given by a pair of monomorphisms $I \hookrightarrow G$ and $O \hookrightarrow G$ (inputs and outputs), and a morphism is a commuting map of such cospans. This category is adhesive, so pushouts along monos exist and satisfy Van Kampen properties.

Remark (skeleton categories). Paper I defined WARP graphs over plain finite multigraph skeletons, but explicitly noted that the recursive construction lifts to richer skeleton categories such as \mathbf{OGraph}_T . We will work over \mathbf{OGraph}_T throughout.

2.2 DPOI rules and steps

Definition 2.1 (DPOI rule). A *DPOI rule* in \mathbf{OGraph}_T is a span of monomorphisms

$$p = (L \xleftarrow{\ell} K \xrightarrow{r} R)$$

with L the left-hand side, K the interface (preserved part), and R the right-hand side.

Definition 2.2 (Match and step). Let $p = (L \xleftarrow{\ell} K \xrightarrow{r} R)$ be a DPOI rule and let G be a host graph. A *match* is a mono $m : L \hookrightarrow G$ satisfying the usual gluing conditions (dangling and identification). A *DPOI step* $G \Rightarrow_p H$ is obtained by computing a pushout complement and a pushout in the standard DPO square.

Remark. We will freely use standard concurrency and confluence results for DPO rewriting in adhesive categories, especially the parallel independence theorem [EEPT06] and its typed-open-graph variants [LS08].

3 WARP States as Two-Plane Objects

Paper I defines the category of WARP graphs, written **WARP**, and notes that in later work we may write \mathcal{W} for brevity. In this paper we distinguish:

- the *recursive object language* of WARPs (attachments, payload substructure);
- the *skeleton plane* on which concurrency and publication are coordinated.

3.1 State objects

Intuitively, a WARP *state* is a skeleton in \mathbf{OGraph}_T plus a WARP attachment over each skeleton vertex and edge.

Definition 3.1 (WARP state). A WARP *state* is a triple

$$U = (G; \alpha, \beta)$$

where:

- $G \in \mathbf{OGraph}_T$ is the *skeleton*;
- α assigns a WARP attachment to each vertex of G ;
- β assigns a WARP attachment to each edge of G .

Attachments are themselves WARP graphs, so this definition is recursively closed under nesting.

Two-plane intuition. The skeleton G carries the coarse wiring and interfaces. Attachments carry nested state and provenance payloads. Rewriting can occur *inside* attachments without changing the skeleton, and the skeleton can be rewritten in a way that preserves and transports attachments.

Attachment depth. Paper I defines a well-founded *depth* function on WARPs (the height of the recursive “graphs all the way down” nesting) and proves that all states have finite depth [Ros25]. We use only a light wrapper of that notion: for a state $U = (G; \alpha, \beta)$ and a skeleton position x (vertex or edge), write

$$\text{depth}_U(x) := \begin{cases} \text{depth}(\alpha(v)) & \text{if } x = v \in V(G), \\ \text{depth}(\beta(e)) & \text{if } x = e \in E(G). \end{cases}$$

Intuitively, $\text{depth}_U(x) = 0$ means there is no descended substructure attached at x ; $\text{depth}_U(x) \geq 1$ means there is nontrivial lineage beneath x .

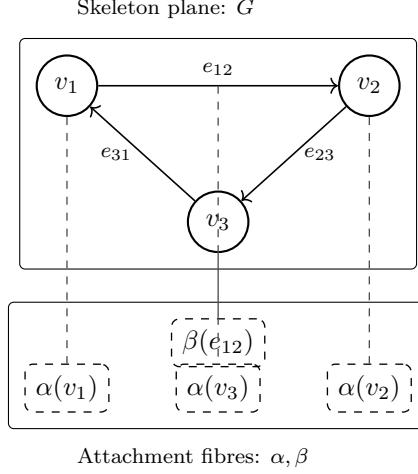


Figure 3.1: A WARP state $U = (G; \alpha, \beta)$ as a two-plane object: the skeleton G lives in \mathbf{OGraph}_T , while WARP attachments α, β sit over vertices and edges and are transported along preserved structure.

3.2 Morphisms of states and the projection functor

We package WARP states into a category \mathbf{WState} whose morphisms extend WARP morphisms by carrying a skeleton morphism plus fibrewise attachment morphisms.

Definition 3.2 (WARP state morphism). Let $U = (G; \alpha, \beta)$ and $U' = (G'; \alpha', \beta')$ be WARP states. A morphism $f : U \rightarrow U'$ consists of:

1. a morphism of skeletons $f_S : G \rightarrow G'$ in \mathbf{OGraph}_T ;
2. for each skeleton vertex v a WARP morphism $f_v : \alpha(v) \rightarrow \alpha'(f_S(v))$, and for each skeleton edge e a WARP morphism $f_e : \beta(e) \rightarrow \beta'(f_S(e))$,

with the obvious compatibility conditions with sources/targets inherited from the skeleton map.

Proposition 3.3. WARP states and their morphisms form a category \mathbf{WState} .

There is a forgetful/projection functor $\pi : \mathbf{WState} \rightarrow \mathbf{OGraph}_T$ sending $(G; \alpha, \beta) \mapsto G$ and dropping attachment components on morphisms.

Remark 3.4 (Fibration viewpoint and a chosen cleavage). For each skeleton $G \in \mathbf{OGraph}_T$, write \mathbf{WState}_G for the fibre of π over G : objects are attachment assignments (α, β) over G , and morphisms are attachment morphisms whose skeleton component is id_G . Given a skeleton morphism $g : G \rightarrow G'$ in \mathbf{OGraph}_T , postcomposition with g induces a canonical reindexing (transport) functor

$$\tau_g : \mathbf{WState}_{G'} \rightarrow \mathbf{WState}_G, \quad \tau_g(G'; \alpha', \beta') := (G; \alpha' \circ g, \beta' \circ g),$$

and similarly on morphisms. Choosing these τ_g for all g is a standard cleavage²; equivalently, $\pi : \mathbf{WState} \rightarrow \mathbf{OGraph}_T$ is a (split) Grothendieck fibration [Bén85, Jac99]. We use this transport notation to state and prove the two-plane commutation theorem; no further fibration machinery is required.

²Informally: τ_g just precomposes an attachment assignment with g . You can read it as “rename attachment slots along g ” so that attachment updates can be compared before and after a skeleton rewrite.

4 Two-Plane Operational Semantics and Ticks

This section defines the operational step of computation for WARP states. The guiding design choice is to make *ticks* primary: a tick is a bounded unit of concurrent work whose outcome is treated atomically.

4.1 Attachment-plane steps

Fix a set of DPOI rules for attachments. An *attachment-plane step* rewrites some attachments $\alpha(v)$ and/or $\beta(e)$ while leaving the skeleton G unchanged.

Definition 4.1 (Attachment-plane step). Let $U = (G; \alpha, \beta)$ be a WARP state. An *attachment-plane step* is a finite family of DPOI steps applied to a selection of attachments $\alpha(v)$ and $\beta(e)$, producing a new state $(G; \alpha_A, \beta_A)$ with the same skeleton G .

4.2 Skeleton-plane steps and batches

A *skeleton-plane step* rewrites the skeleton G by a DPOI rule p and match $m : L \hookrightarrow G$, producing a new skeleton G' . To obtain a successor WARP state $(G'; \alpha', \beta')$, we transport attachments along the preserved structure:

- attachments over preserved skeleton positions are reindexed along the induced map on those positions;
- deleted positions drop their attachments, subject to the no-delete-under-descent invariant (Section 8).

A *batch* is a finite family of skeleton matches intended to be committed concurrently. In this paper the only batches we ever commit are *scheduler-admissible* (pairwise independent) batches, defined in Section 5. For such a batch B , the parallel-independence theorem implies that any serialisation of the steps in B yields the same successor up to isomorphism (Theorem 6.1).

4.3 Atomic ticks

Definition 4.2 (Tick). A *tick* on a WARP state $U = (G; \alpha, \beta)$ consists of:

1. a finite family of attachment-plane steps on attachments over G , producing $(G; \alpha_A, \beta_A)$; and
2. a scheduler-selected skeleton batch B on G , committed as a family of DPOI steps to produce a new skeleton G' and transported attachments (α', β') .

We write $U \Rightarrow_{\text{Tick}} U'$ for “one tick commits U' ”.

Remark 4.3 (Atomicity and failure). A tick is treated as an *atomic commit*: either all selected component rewrites succeed and the committed successor is observed, or the state is unchanged. Operationally, an implementation may stage intermediate results and roll them back, but the abstract semantics exposes no partial effects. When a tick aborts we model this as a stutter step $U \Rightarrow_{\text{Tick}} U$ together with a receipt object describing the abort (Section 7); this aligns the semantics with standard transaction theory [GR92].

Remark 4.4 (Why attachments-before-skeleton?). The attachment-then-skeleton order is an operational discipline: local work settles inside attachments, then a publication step updates global wiring. Section 8 proves that, under standard invariants, swapping these two phases commutes up to canonical isomorphism.

4.4 Schedulers and deterministic runtimes

The remaining source of nondeterminism is *which* scheduler-admissible batch is chosen on a given tick.

Definition 4.5 (Scheduler policy). A *scheduler* is a function (or relation) that, given a state U , selects a scheduler-admissible batch B of skeleton matches to be committed in the current tick. A scheduler is *deterministic* if it is a total function $\sigma : \mathbf{WState} \rightarrow \mathbf{Batch}$. Only the selected batch is executed; non-selected candidates are deferred rather than “attempted and allowed to fail”.

5 Footprints, Independence, and Scheduler-Admissible Batches

We model concurrency by an explicit independence discipline on skeleton matches. The discipline is stated in terms of read/write footprints: each match specifies the items it may delete and the items it may use (read).

5.1 Footprints

Definition 5.1 (Footprint). Let $m_S : L_S \hookrightarrow G_S$ be a skeleton match for rule p with interface $K_S \subseteq L_S$. The *delete set* is

$$\text{Del}(m_S) = m_S(L_S \setminus K_S),$$

the image of the part of the left-hand side not preserved by the interface. The *use set* is

$$\text{Use}(m_S) = m_S(K_S),$$

the image of the entire left-hand side. The *footprint* is the pair $\text{Foot}(m_S) = (\text{Del}(m_S), \text{Use}(m_S))$.

5.2 Independence

Definition 5.2 (Independence). Two skeleton matches $m_{1,S} : L_{1,S} \hookrightarrow G_S$ and $m_{2,S} : L_{2,S} \hookrightarrow G_S$ are *independent* if

$$\text{Del}(m_{1,S}) \cap \text{Use}(m_{2,S}) = \emptyset \quad \text{and} \quad \text{Del}(m_{2,S}) \cap \text{Use}(m_{1,S}) = \emptyset.$$

Intuitively: nothing deleted by one step is read/used by the other.

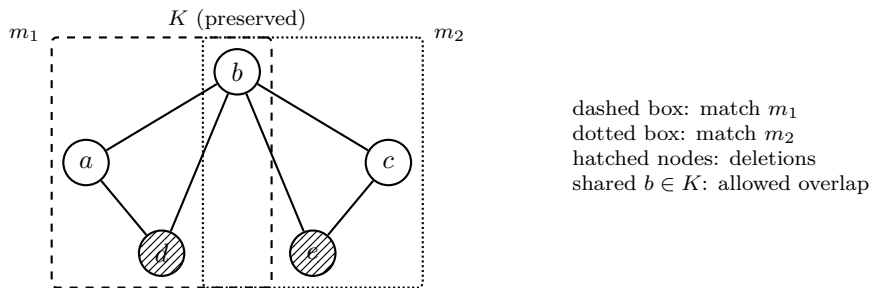


Figure 5.1: Independence via footprints. Two matches may overlap on preserved interface items (a shared pivot $b \in K$ is typical). Independence forbids only delete–use interference: $\text{Del}(m_1) \cap \text{Use}(m_2) = \emptyset$ and $\text{Del}(m_2) \cap \text{Use}(m_1) = \emptyset$.

5.3 Scheduler-admissible batches

Definition 5.3 (Scheduler-admissible batch). Let $U = (G; \alpha, \beta)$ be a WARP state. A finite family of skeleton matches $B = \{m_{i,S} : L_{i,S} \hookrightarrow G_S\}_{i \in I}$ is *scheduler-admissible* if the matches are pairwise independent.

6 Tick-Level Confluence on the Skeleton Plane

This section proves the core scheduler confluence theorem: inside a tick, serialisation order doesn't matter.

Theorem 6.1 (Skeleton-plane tick confluence). *Let $U = (G; \alpha, \beta)$ be a WARP state and let B be a scheduler-admissible batch. Then any two serialisations of the corresponding skeleton-plane DPOI steps yield isomorphic successor skeletons. Consequently, the induced successor WARP states are isomorphic (up to skeleton isomorphism in \mathbf{OGraph}_T and attachment isomorphisms transported along preserved structure).*

Proof. Let $B = \{(p_i, m_i)\}_{i \in I}$ be a scheduler-admissible batch, so the matches are pairwise independent (Definition 5.3). By the parallel-independence theorem for DPOI rewriting in an adhesive category [EEPT06, LS08], any two independent steps commute: if $i \neq j$ then executing i followed by j yields (up to isomorphism) the same result as executing j followed by i .

Any two serialisations of a finite family differ by a finite sequence of adjacent swaps. Swapping adjacent independent steps preserves the result up to isomorphism, so by induction along the swap sequence, all serialisations of B yield isomorphic successor skeletons. Transporting attachments along preserved structure is functorial (Remark 3.4), hence the induced successor WARP states are isomorphic as claimed. \square

Corollary 6.2 (Within-tick worldline uniqueness). *Fix a state U and a scheduler-admissible batch B . The tick outcome determined by B is unique up to isomorphism, independent of the internal serialisation order used by the runtime.*

7 Deterministic Scheduling and Tick Receipts

Theorem 6.1 isolates a key fact: once a scheduler-admissible batch B is fixed, the committed successor is independent of the internal serialisation used to realise B . To obtain *whole-run* determinism, we must also control how B is chosen. To obtain *provenance-ready* runs, we additionally want a faithful record of that choice.

7.1 Candidate matches and a deterministic policy

Let $U = (G; \alpha, \beta)$ be a state. Write $\mathbf{Cand}(U)$ for the set of all skeleton matches $m : L \hookrightarrow G$ that satisfy the DPOI gluing conditions for their rules. Assume that skeleton items carry stable identifiers and that each rule is given a fixed name. Then each match $m \in \mathbf{Cand}(U)$ can be assigned a deterministic *key* (e.g. a lexicographically ordered tuple of the identifiers of its image, together with the rule name), inducing a total order \prec on $\mathbf{Cand}(U)$.

A deterministic scheduler is then simply a total function $\sigma : \mathbf{WState} \rightarrow \mathbf{Batch}$. One canonical choice is a greedy “left-most wins” filter.

Definition 7.1 (Left-most scheduler). Fix a total order \prec on candidate matches. Define $\sigma_{\text{lm}}(U)$ by iterating the list $m_1 \prec m_2 \prec \dots \prec m_n$ of $\mathbf{Cand}(U)$ and accepting m_i iff it is independent of all

previously accepted matches. The resulting accepted set $B \subseteq \text{Cand}(U)$ is scheduler-admissible by construction.

7.2 Tick receipts and tick-event posets

For replay, recording the chosen batch B is enough. For provenance, debugging, and auditability, it is useful to record *why* certain candidates were excluded. This motivates an optional refinement object returned by a tick: a receipt.

Definition 7.2 (Tick receipt). A *tick receipt* for a state U is a tuple

$$\rho = (E, \preceq, E_{\text{acc}}, E_{\text{rej}}, \text{meta})$$

where:

- $E \subseteq \text{Cand}(U)$ is the finite set of candidate events considered by the scheduler;
- $E_{\text{acc}} \subseteq E$ is the set of accepted events (the batch), and $E_{\text{rej}} = E \setminus E_{\text{acc}}$;
- (E, \preceq) is a finite poset (the *tick-event poset*) recording scheduling causality (e.g. “ e blocks e' ”);
- meta is optional metadata (stable identifiers, rule names, rejection reasons, and/or an abort flag).

For the left-most scheduler, a canonical choice of \preceq is generated by the blocking relation: when a candidate m_i is rejected because it overlaps an already accepted match m_j with $j < i$, record $m_j \prec_{\text{blk}} m_i$ and take \preceq to be the reflexive transitive closure. The resulting structure is a standard sort of partial-order trace object [Maz87, Win87].

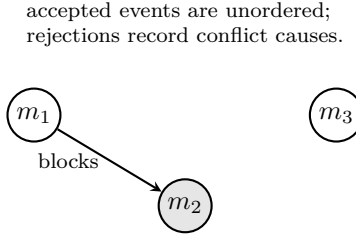


Figure 7.1: A minimal tick-event poset induced by deterministic (left-most) conflict resolution. Accepted events are unordered; rejected events are causally after the event that blocked them.

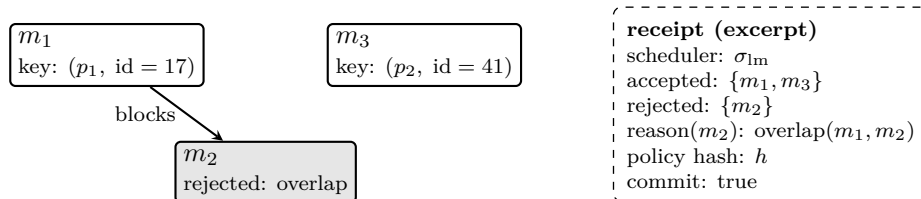


Figure 7.2: A tick receipt at small scale. Each candidate event carries a deterministic key (rule name and stable identifiers). Rejected events record a reason and the accepted event that blocked them.

7.3 Quotienting receipts to committed state

The receipt ρ is an *observational refinement*: it carries more information than the committed successor state, but (by confluence) it does not change that successor.

Proposition 7.3 (Receipts refine ticks). *Let U be a state and let ρ be a tick receipt with accepted set $E_{\text{acc}} = B$. Then the committed successor produced by executing the tick depends only on B , not on the particular linearisation used to realise B nor on the additional structure (E, \preceq) . Equivalently: forgetting ρ down to B and then applying the tick yields the same successor (up to isomorphism) as any execution consistent with ρ .*

Proof. By definition, B is scheduler-admissible, hence its steps are pairwise independent. By Theorem 6.1, any two serialisations of B yield isomorphic successors. The additional receipt data only records (i) rejected candidates, which are not executed, and (ii) causal explanation of those rejections, which does not affect the committed DPOI construction. \square

Worldlines. Given a deterministic scheduler σ and a deterministic policy for attachment-plane updates, a run produces a canonical \mathbb{N} -indexed trace

$$U_0 \xrightarrow{\text{Tick}_1, \rho_1} U_1 \xrightarrow{\text{Tick}_2, \rho_2} \dots$$

where each ρ_i is a receipt for tick i . We keep the global history linear (a worldline) while allowing each tick to carry internal partial-order structure; Paper III exploits this refinement for provenance.

8 Two-Plane Commutation and Deterministic Tick Semantics

Tick-level confluence on the skeleton plane is not enough by itself: WARP ticks also include attachment updates. This section shows that the two planes commute (up to canonical transport), so “attachments-then-skeleton” is equivalent to “skeleton-then-transported-attachments”.

8.1 No-delete/no-clone-under-descent

The key operational invariant is that skeleton publication cannot destroy or duplicate attachment lineage.

Definition 8.1 (No-delete/no-clone-under-descent). Let $U = (G; \alpha, \beta)$ be a WARP state and let B be the skeleton batch committed on G during a tick. Write $\text{Del}(B) = \bigcup_{m \in B} \text{Del}(m)$ for the overall delete set on skeleton items.

The tick satisfies *no-delete/no-clone-under-descent* if:

1. (**No delete under descent**) for any skeleton position x with $\text{depth}_U(x) \geq 1$, we have $x \notin \text{Del}(B)$.
2. (**No clone under descent**) any skeleton position x with $\text{depth}_U(x) \geq 1$ has a unique preserved image in the successor skeleton G' (so attachment transport is single-valued). In ordinary DPOI rewriting with monomorphisms this uniqueness is automatic on the preserved interface, but we state it explicitly because it is the precise condition required for unambiguous attachment transport.

8.2 Two-plane commutation

Theorem 8.2 (Two-plane commutation). *Let $U = (G; \alpha, \beta)$ be a WARP state. Let $A : (G; \alpha, \beta) \Rightarrow (G; \alpha_A, \beta_A)$ be an attachment-plane step. Let B be a scheduler-admissible skeleton batch on G that commits $G \Rightarrow_B G'$, and assume the tick satisfies no-delete/no-clone-under-descent (Definition 8.1).*

*Write S for the induced state update that commits B on the skeleton and transports attachments along preserved structure using the chosen cleavage τ (Remark 3.4). Then there exists an attachment-plane step A' over G' such that the square in Figure 8.1 commutes up to canonical isomorphism in **WState**.*

$$\begin{array}{ccc} (G; \alpha, \beta) & \xrightarrow{A} & (G; \alpha_A, \beta_A) \\ \downarrow S & & \downarrow S_A \\ (G'; \alpha', \beta') & \xrightarrow{A'} & (G'; \alpha'', \beta'') \end{array}$$

Figure 8.1: Two-plane commutation. The horizontal arrows A represent purely local updates to attachments. The vertical arrows S represent structural changes to the skeleton. Because the skeleton update respects No-Delete/No-Clone invariants, the diagram commutes up to canonical isomorphism.

Proof. By Remark 3.4, we have fixed a transport functor τ_g along each skeleton morphism g (a chosen cleavage). Committing the skeleton batch B yields a successor skeleton G' together with a canonical identification of preserved items (the usual “tracking” of DPOI rewriting on the interface and context). The no-delete/no-clone-under-descent conditions ensure that this identification is defined, and single-valued, on every skeleton position that carries nontrivial descended attachment structure. Transporting attachments across S is therefore just reindexing along the induced map on preserved positions.

Attachment-plane steps act entirely inside fibres: they rewrite the WARP objects $\alpha(v)$ and $\beta(e)$ without changing G . Reindexing is functorial, so transporting after applying A is canonically isomorphic to transporting first and then applying the transported family of fibrewise rewrites A' over G' . This is exactly the commutativity (up to canonical isomorphism) asserted by Figure 8.1. \square

Corollary 8.3 (Deterministic tick outcome). *Fix a WARP state U and suppose a tick selects a set of attachment-plane rewrites and a scheduler-admissible skeleton batch satisfying ND/NC. Then the tick outcome is unique up to isomorphism, independent of serialisation order and the interleaving of attachment/skeleton updates.*

9 Global Confluence

Theorems 6.1 and 8.2 give *per-tick* determinism. Whole-run determinism requires either a deterministic scheduler policy or confluence of the underlying rewrite relation.

Theorem 9.1 (Newman’s lemma (modulo isomorphism)). *Let \rightarrow be the one-tick reduction relation induced on skeletons (or states), and consider it modulo typed open graph isomorphism. If \rightarrow is terminating and locally confluent (modulo typed open graph isomorphism), then \rightarrow is confluent (modulo typed open graph isomorphism).*

Proof. This is the standard Newman argument [New42]. Termination ensures that every reduction sequence from a given start object is finite, and local confluence ensures that every one-step divergence can be joined. An induction on the sum of reduction lengths (or, equivalently, on the well-founded termination order) then yields confluence. For non-terminating systems, one can replace termination with alternative criteria such as decreasing diagrams [vO94], but we do not pursue that refinement here. \square

In practice, local confluence often reduces to a finite critical-pair analysis in the DPOI setting.

10 Discussion and Outlook

Paper I provided the state object; this paper provides the operational semantics. The main methodological point is that concurrency is treated as a semantic design choice rather than an implementation accident: admissible parallelism is defined by an independence discipline on skeleton matches, and publication is constrained by invariants that prevent the skeleton plane from deleting or duplicating descended attachment structure. With those constraints in place, standard results from categorical graph rewriting supply the desired determinacy properties: within a tick, the committed successor is independent of internal serialisation; across the two planes, attachment evolution commutes with skeleton publication up to canonical transport.

Relation to other concurrency models. Our “maximal independent batch per tick” discipline is closest in spirit to step semantics for Petri nets and to trace/event-structure models of concurrency [Jen97, Maz87, Win87]. The difference is that independence is decided syntactically from DPOI match footprints (delete/use sets) rather than from token enabling. Bigraphs and related formalisms emphasise locality, mobility, and binding structure [Mil02]; WARP graphs instead separate global wiring (the skeleton plane) from fibred state (attachments), and enforce determinism by isolating the remaining choice into a policy σ and recording it via receipts.

From the perspective of replay, the only remaining degree of freedom is the scheduler policy σ . If σ (and the attachment-update policy) is deterministic, then the semantics yields a canonical worldline trace. From the perspective of provenance, this paper deliberately separates *what is committed* (the successor state) from *how that commitment was chosen* (the tick receipt and its event poset). Paper III builds on this separation by treating receipts as first-class provenance payloads rather than informal log files.

Finally, we emphasise that we have modelled ticks as atomic commits (Remark 4.3): a tick either happens or it does not. More refined implementation strategies (e.g. partial rollback with compensation logs) can be analysed as refinement theorems against this atomic model, without expanding the mathematical surface area of Paper II.

11 Notation Summary

Symbol	Meaning
\mathbf{OGraph}_T	category of T -typed open graphs (adhesive)
$p = (L \xleftarrow{\ell} K \xrightarrow{r} R)$	DPOI rule
$m : L \hookrightarrow G$	match of a rule in a host graph
$U = (G; \alpha, \beta)$	WARP state: skeleton G with vertex/edge attachments
\mathbf{WState}	category of WARP states
$\pi : \mathbf{WState} \rightarrow \mathbf{OGraph}_T$	projection to skeletons
τ_g	transport (reindexing) functor along a skeleton morphism g
$\text{depth}_U(x)$	attachment depth at skeleton position x
$\text{Del}(m_S), \text{Use}(m_S)$	delete and use sets of a skeleton match
$\text{Foot}(m_S)$	footprint (Del, Use)
$\text{Cand}(U)$	set of candidate skeleton matches in state U
B	scheduler-admissible batch (pairwise independent)
σ	deterministic scheduler policy $\mathbf{WState} \rightarrow \mathbf{Batch}$
ρ	tick receipt (including a tick-event poset)
Tick	one tick of the two-plane operational semantics

A Appendix: A Minimal Independence Example

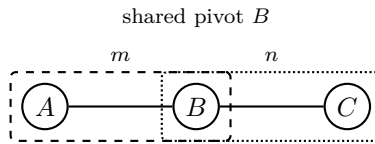


Figure A.1: A shared-pivot pattern. Two matches may overlap on a preserved node B provided neither deletes it; footprints make this check explicit.

References

- [Bén85] Jean Bénabou. Fibered categories and the foundations of naive category theory. *The Journal of Symbolic Logic*, 50(1):10–37, 1985.
- [EPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
- [EL97] Hartmut Ehrig and Michael Löwe. Parallel and distributed graph transformation. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 433–456. World Scientific, 1997.
- [GR92] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.
- [Jac99] Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 1999.
- [Jen97] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1997.
- [LS08] Stephen Lack and Paweł Sobociński. Adhesive categories. *Foundations of Computational Mathematics*, 8(2):191–210, 2008.
- [Maz87] Antoni Mazurkiewicz. Trace theory. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 278–324. Springer, 1987.
- [Mil02] Robin Milner. Bigraphs and mobile processes. *Theoretical Computer Science*, 274(1–2):1–51, 2002.
- [New42] M. H. A. Newman. On theories with a combinatorial definition of “equivalence”. *Annals of Mathematics*, 43(2):223–243, 1942.
- [Ros25] James Ross. WARP Graphs: A Worldline Algebra for Recursive Provenance. *AIΩN Foundations Series — Paper I*, December 2025. Version cited: December 2025 PDF.
- [vO94] Vincent van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.
- [Win87] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1987.