Flowing Relaxes POP - Proof

(supplementary material for Modelling the ARMv8 Architecture, Operationally: Concurrency and ISA)

1 Notation

For a set A, let $\mathcal{T}(A)$, the *hierarchical partitioning of* A, denote a tree whose nodes are labeled with subsets of A subject to:

- there are exactly $|A|$ many leaves and each leaf has a unique label from A,
- the label of a parent node is the (disjoint) union of the labels of its children nodes.

Note that with those constraints the label of the root node has to be A. We will consider a hierarchical partitioning of the set of thread identifiers of a given Flowing (or POP) system, and denote it as $\mathcal{T}$. Each node in $\mathcal{T}$ is called a *segment*, ranged over by $S, S', S_1, \ldots$. We write $S \geq S'$ whenever S is an ancestor of S' and $S > S'$ whenever S is a proper ancestor of S'. Let $S \oplus S'$ denote the minimal common ancestor of S and S'; i.e. the segment contained in every segment which contains $S \cup S'$.

In Table 1, we list the symbols used to denote frequently occurring components about Flowing and POP systems.

F	System states of Flowing, ranged over by $\mathsf{f}, \mathsf{f}_i, \mathsf{f}', \ldots$
P	System states of PoP, ranged over by $\mathsf{p}, \mathsf{p}_i, \mathsf{p}', \ldots$
Q	Generic system states (Flowing or PoP), ranged over by $\mathsf{q}, \mathsf{q}_i, \mathsf{q}', \ldots$
$ss(\mathsf{q})$	The storage subsystem state of the system state q
$ts(\mathsf{q})$	The thread subsystem state of the system state q

Table 1: Some notation for the main components of Flowing and POP systems.

2 The Simulation Relation, σ

That POP relaxes Flowing will be established by constructing a simulation relation $\sigma \subseteq \mathsf{F} \times \mathsf{P}$. The proof will be by induction on the length of a Flowing trace, $\mathbf{r}_{\mathcal{F}}$. The inductive step is to prove that if f is any Flowing state reached by a Flowing trace $\mathbf{r}_{\mathcal{F}}$ of length k, there is a POP state p such that $(\mathsf{f}, \mathsf{p}) \in \sigma$ and there is some Flowing transition $\mathsf{f} \xrightarrow{t}_{\mathcal{F}} \mathsf{f}'$, then there is a POP state p' such that $\mathsf{p} \xrightarrow{\mu(t)}_{\mathcal{P}}^{*} \mathsf{p}'$ and $(\mathsf{f}', \mathsf{p}') \in \sigma$.

2.1 Flowing state components

Let $\mathsf{f} \in \mathsf{F}$ be a Flowing system state. The relevant components of the state, along with the notation we are going to use in the rest of this document given in square brackets, are the following:

- `thread_ids` [$\mathsf{f}.Tid$] – set of thread identifiers comprising the system.
- `topology` [$\mathsf{f}.Topo$] – a hierarchical partitioning of $\mathsf{f}.Tid$, representing the segment interconnections.
- `thread_to_segment` [$\mathsf{f}.TofS$] – maps each thread t to the leaf segment that (exclusively) contains t.
- `buffers` [$\mathsf{f}.Buf$] – maps each segment S to the sequence of requests contained in S.
- `reordered` [$\mathsf{f}.Reord$] – set of pairs of requests which have already been reordered.
- `memory` [$\mathsf{f}.Mem$] – set of write requests, at most one per location, representing the most recent write to reach the *memory*.

Additionally, we introduce the following auxiliary components:

- [$\mathsf{f}.Evt^{act}$] – set of requests that are currently in the storage subsystem.
- [$\mathsf{f}.Evt$] – set of requests that are either in $\mathsf{f}.Evt^{act}$ or have completed execution[1].
- [$\mathsf{f}.Seg$] – maps each request in $\mathsf{f}.Evt^{act}$ to some segment in $\mathsf{f}.Topo$.
- [S_{mem}] – segment containing all requests that have flown into memory (with the requirement that S_{mem} not a segment in $\mathsf{f}.Topo$). We assume that for any f, $\mathsf{f}.Tid \subseteq S_{mem}$ and that $\mathsf{f}.Buf$ is extended to include S_{mem}.

We also define a partial order $\succ_{\mathsf{f}}$ over requests in $\mathsf{f}.Evt$ such that for any two requests $e, e' \in \mathsf{f}.Evt$ $e \succ_{\mathsf{f}} e'$ holds if one of the following condition holds:

- $\mathsf{f}.Seg(e) > \mathsf{f}.Seg(e')$,
- $\mathsf{f}.Seg(e) = S_{mem}$ and $\mathsf{f}.Seg(e') \neq S_{mem}$,
- there is some segment $S \in \mathsf{f}.Topo \cup \{S_{mem}\}$ such that $S = \mathsf{f}.Seg(e) = \mathsf{f}.Seg(e')$ and $\mathsf{f}.Buf(S)$ is of the form $E_1 \cdot e' \cdot E_2 \cdot e \cdot E_3$ for some sequences of requests E_i over $\mathsf{f}.Evt$.

Finally, for any read/write request e we use $Addr(e)$ to denote the address e accesses.

2.2 POP storage subsystem states

Let $\mathsf{p} \in \mathsf{P}$ be a POP system state. The relevant components of the state, along with the notation we are going to use in the rest of this document given in square brackets, are the following:

- `thread_ids` [$\mathsf{p}.Tid$] – set of thread identifiers comprising the system.
- `requests_seen` [$\mathsf{p}.Evt$] – set of requests accepted by the storage subsystem.

[1]This means that in case e is a read request it has been satisfied; otherwise, it has flown into memory.

- `order_constraints` [$\mathsf{p}.Ord$] – partial order over `requests_seen` (may include orderings between requests to different locations).
- `requests_propagated_to` [$\mathsf{p}.Prop$] – maps each thread identifier t to the set of requests (subset of the current value of `requests_seen`) that have already been propagated to t.

A POP state p is called $\mathcal{T}$*-conforming* if for all $e \in \mathsf{p}.Evt$, the set $\{t \mid e \in \mathsf{p}.Prop(t)\}$ is a segment in $\mathcal{T}$. In such a case, the set $\{t \mid e \in \mathsf{p}.Prop(t)\}$ is called the *segment* of e, written $\mathsf{p}.Seg(e)$. Additionally, we introduce the auxiliary component $\mathsf{p}.Mem$ which is the dual of $\mathsf{f}.Mem$ of a Flowing state.

2.3 Simulation invariant, $\mathcal{I}_\sigma$

Let f and p be system states of the Flowing and the PoP systems, respectively. Let s_f and s_p be shorthand for $ss(\mathsf{f})$ and $ss(\mathsf{p})$, respectively. Then the simulation invariant $\mathcal{I}_\sigma(\mathsf{f},\mathsf{p})$ is defined as the conjunction of the following invariants:

$\mathcal{I}_{tss}(\mathsf{f},\mathsf{p})$: $ts(\mathsf{f}) = ts(\mathsf{p})$

$\mathcal{I}_{tid}(\mathsf{f},\mathsf{p})$: $\mathsf{f}.Tid = \mathsf{p}.Tid$

$\mathcal{I}_{evt}(\mathsf{f},\mathsf{p})$: $\mathsf{f}.Evt = \mathsf{p}.Evt$

$\mathcal{I}_{ord}(\mathsf{f},\mathsf{p})$: for any two requests $e, e' \in \mathsf{p}.Evt$, $(e, e') \in \mathsf{p}.Ord$ implies $e \succ_\mathsf{f} e'$,

$\mathcal{I}_{seg}(\mathsf{f},\mathsf{p})$: p is $\mathsf{f}.Topo$-conforming and $\mathsf{p}.Seg(e) = S$ iff $\mathsf{f}.Buf(S)$ contains e.

We define the simulation relation σ by the following equivalence:

$$\forall \mathsf{f}, \mathsf{p} \,.\, \sigma(\mathsf{f},\mathsf{p}) \Leftrightarrow \mathcal{I}_\sigma(\mathsf{f},\mathsf{p})$$

3 The Transitions

In this section, the transitions of the Flowing and PoP storage subsystems will be listed and a correspondence between them, i.e. $\mu : \mathrm{Tr}_\mathcal{F} \mapsto \mathrm{Tr}^*_\mathcal{P}$, will be given.

3.1 The Flowing Storage Subsystem Transitions, $\mathrm{Tr}_\mathcal{F}$

$\langle \dashrightarrow e\rangle_\mathcal{F}$ Moves request e into the storage subsystem (actually a side-effect of a thread subsystem transition).

$\langle e \rightarrow\rangle_\mathcal{F}$ Pulls e from the head of some segment and pushes onto the tail of the next segment.

$\langle e \leftrightarrow e'\rangle_\mathcal{F}$ Swaps the positions of two requests e and e' both located in the same segment.

$\langle r \smile w\rangle_\mathcal{F}$ Matches the read request r with the write request w, consecutively located in some segment.

$\langle r \dashrightarrow\rangle_\mathcal{F}$ Matches the read request r with the contents of the memory, when r is at the head of the root segment.

$\langle w \dashrightarrow\rangle_{\mathcal{F}}$ Moves the write request w at the head of the root segment to the memory.

$\langle b \dashrightarrow\rangle_{\mathcal{F}}$ Removes the barrier request b from the head of the root segment.

$\langle \dashrightarrow w_{\mathsf{x}} \smile w\rangle_{\mathcal{F}}$ Moves the write exclusive request w_{x} which is paired with the write request w located in some segment into the storage subsystem.

$\langle \dashrightarrow w_{\mathsf{x}} \smile\rangle_{\mathcal{F}}$ Moves the write exclusive request w_{x} which is paired with the memory (i.e. the write request which has most recently updated the contents in the memory) into the storage subsystem.

3.2 The POP Storage Subsystem Transitions, $\mathrm{Tr}_{\mathcal{P}}$

$\langle \dashrightarrow e\rangle_{\mathcal{P}}$ Moves an request e into the storage subsystem; similar to $\langle \dashrightarrow e\rangle_{\mathcal{F}}$.

$\langle e_1 \,\mathsf{coh}\, e_2\rangle_{\mathcal{P}}$ Modifies the extended coherence relation by ordering two hitherto unordered requests e_1 and e_2.

$\langle e \rightsquigarrow t\rangle_{\mathcal{P}}$ Propagates request e to thread t.

$\langle t : r \smile w\rangle_{\mathcal{P}}$ Sends thread t the value observed by read request r which is matched with the write request w.

$\langle \dashrightarrow w_{\mathsf{x}} \smile w\rangle_{\mathcal{P}}$ Similar to the dual Flowing transition.

3.3 The transition mapping, μ

Here we list each transition of the Flowing storage subsystem and the corresponding sequence of transitions that are to be done by the POP storage subsystem.

t	$\mu(t)$
$\langle \dashrightarrow e\rangle_{\mathcal{F}}$	$\langle \dashrightarrow e\rangle_{\mathcal{P}}$
$\langle e \rightarrow\rangle_{\mathcal{F}}$	$\langle e \rightsquigarrow t_1\rangle_{\mathcal{P}} \cdot \langle e \rightsquigarrow t_2\rangle_{\mathcal{P}} \ldots \langle e \rightsquigarrow t_k\rangle_{\mathcal{P}}$, where $S' \setminus S = \{t_i\}$
$\langle e \leftrightarrow e'\rangle_{\mathcal{F}}$	ε
$\langle r \smile w\rangle_{\mathcal{F}}$	$\langle t : r \smile w\rangle_{\mathcal{P}}$, where t is the thread which has issued r
$\langle r \dashrightarrow\rangle_{\mathcal{F}}$	$\langle t : r \smile \hat{w}\rangle_{\mathcal{P}}$, where t is the thread which has issued r, $\hat{w}$ is the most recent complete write to $Addr(r)$
$\langle w \dashrightarrow\rangle_{\mathcal{F}}$	ε
$\langle b \dashrightarrow\rangle_{\mathcal{F}}$	ε
$\langle \dashrightarrow w_{\mathsf{x}} \smile w\rangle_{\mathcal{F}}$	$\langle \dashrightarrow w_{\mathsf{x}} \smile w\rangle_{\mathcal{P}}$
$\langle \dashrightarrow w_{\mathsf{x}} \smile\rangle$	$\langle \dashrightarrow w_{\mathsf{x}} \smile \hat{w}\rangle_{\mathcal{P}}$, where $\hat{w}$ is the most recent complete write to $Addr(r)$

4 Proof of Simulation

In this section, we prove by induction that the simulation invariant $\mathcal{I}_\sigma$ is preserved. Formally, we show that for any Flowing state f and transition t enabled at f, the transition sequence $\mu(t)$ from $\mathsf{p} = \sigma(\mathsf{f})$ is enabled and for the respective end states f' and p', we have $\sigma(\mathsf{f}') = \mathsf{p}'$.

We omit the transitions of the thread subsystem which is identical for Flowing and POP systems. The only action of the transition system that implicitly involves the storage subsystem state is the removal of a request from the storage subsystem. Assume that e is the request to be removed from the storage subsystem at state f and that $(\mathsf{f}, \mathsf{p}) \in \sigma(\mathsf{f}, \mathsf{p})$. Observe that $\mathcal{I}_\sigma(\mathsf{f}, \mathsf{p})$ implies that $ts(\mathsf{f}) = ts(\mathsf{p})$ so the POP state p is able to perform the same transition which results in the removal of e from $ss(\mathsf{p})$. Since $\mathsf{f}'.Tid = \mathsf{f}.Tid$ and $\mathsf{p}'.Tid = \mathsf{p}.Tid$ for all f' and p' reachable from f and p respectively, there is no transition that can invalidate $\mathcal{I}_{tid}(\mathsf{f}, \mathsf{p})$.[2] Since $\mathsf{f}'.Evt = \mathsf{f}.Evt \setminus \{e\}$, $\mathsf{p}'.Evt = \mathsf{p}.Evt \setminus \{e\}$ and $\mathsf{f}.Evt = \mathsf{p}.Evt$ by the assumption that $\mathcal{I}_{evt}(\mathsf{f}, \mathsf{p})$, we have $\mathcal{I}_{evt}(\mathsf{f}', \mathsf{p}')$. Similarly, it is straightforward to show that $\mathcal{I}_{seg}(\mathsf{f}', \mathsf{p}')$ holds. Finally, the removal of e from p means that $\mathsf{p}'.Ord = [(\mathsf{p}.Ord \setminus \{e\}) \cap Dep]^*$. We show in the next section (Lemma 1) that $\mathsf{p}.Ord$ is always the transitive closure of a subset of Dep. This observation and the fact that $\mathsf{p}.Ord$ and $\succ_\mathsf{f}$ do not conflict by $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$ are enough to conclude that $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$ also holds.

In what follows, we use Dep as a binary relation over requests capturing the reorder condition used in the preconditions of storage subsystem transitions (see Sec.6.4 of the main submission). That is, for any two requests e, e', $(e, e') \in Dep$ if and only if the reorder condition is not satisfied by e and e'.

4.1 Auxiliary Claims

We begin by stating several side results which will then be used in the simulation proof.

Lemma 1 ($Ord \subseteq Dep^*$)**.** *For all requests e, e' and POP states p, if $(e, e') \in \mathsf{p}.Ord$, then there exist $e_1, \ldots, e_n$ such that $e_1 = e$, $e_n = e'$ and for all $i \in [1, n)$ we have (e_i, e_{i+1}).*

Proof. Proof is done by induction on the length of trace. Base case is vacuously true since at an initial state p_0, $\mathsf{p}_0.Ord = \emptyset$. Now assume that the claim holds for all traces of length up to k. Consider a trace $\mathsf{p}_0 \xrightarrow{a_1}_\mathcal{P} \mathsf{p}_1 \ldots$ of length $k+1$ and let $\mathsf{p}_k \xrightarrow{a_{k+1}}_\mathcal{P} \mathsf{p}_{k+1}$ be its last ($k + 1^{th}$) transition. There are three cases to consider based on the transition with action a_{k+1}.

- a_{k+1} is an accept request type, of request $\hat{e}$ by thread $\hat{t}$, into the storage subsystem. By definition of the transition, $\mathsf{p}_k.Ord \subseteq \mathsf{p}_{k+1}.Ord$, and $\mathsf{p}_{k+1}.Ord$ is derived from $\mathsf{p}_k.Ord$ by first adding all pairs $(e, \hat{e})$ such that e has propagated to $\hat{t}$ and $(e, \hat{e}) \in Dep$, and then taking the transitive closure. This implies that any pair (e_1, e_2) that is in $\mathsf{p}_{k+1}.Ord \setminus \mathsf{p}_k.Ord$ we have $\hat{e} \neq e_1$ and $\hat{e} = e_2$. Assume (e, e') is in $\mathsf{p}_{k+1}.Ord$. If (e, e') is also in $\mathsf{p}_k.Ord$, then by the induction hypothesis there exist $e_1, \ldots, e_n$ satisfying the claim. If (e, e') is not in $\mathsf{p}_k.Ord$, then by the above observation we have $e' = \hat{e}$. By construction, we must have either $(e, \hat{e}) \in Dep$ or some e' such that $(e, e') \in \mathsf{p}_k.Ord$ and $(e', \hat{e}) \in Dep$. By the induction hypothesis, there must exist some $e_1, \ldots, e_m$ such that $e = e_1$, $e' = e_m$ and $(e_i, e_{i+1}) \in Dep$ for all $i \in [1, m)$. This immediately implies that $e_1, \ldots, e_m, \hat{e}$ is the desired sequence of $(e, \hat{e})$.

[2]As a matter of fact, we will in the rest of the simulation proof ignore the preservation of $\mathcal{I}_{tid}(\mathsf{f}, \mathsf{p})$ which as argued here is trivial.

- a_{k+1} is a propagate request type, of request $\hat{e}$ by thread $\hat{t}$ to thread t'. By definition of the transition, $\mathsf{p}_k.Ord \subseteq \mathsf{p}_{k+1}.Ord$, and $\mathsf{p}_{k+1}.Ord$ is derived from $\mathsf{p}_k.Ord$ by first adding all pairs $(\hat{e}, e)$ such that e has propagated to t' but not to $\hat{t}$ and $(\hat{e}, e) \in Dep$, and then taking the transitive closure. Assume (e, e') is in $\mathsf{p}_{k+1}.Ord$. If (e, e') is also in $\mathsf{p}_k.Ord$, then by the induction hypothesis there exist $e_1, \ldots, e_n$ satisfying the claim. If (e, e') is not in $\mathsf{p}_k.Ord$, then there exists a request e'' such that $(e, \hat{e}) \in \mathsf{p}_k.Ord$, $(\hat{e}, e'') \in Dep$ and $(e'', e') \in \mathsf{p}_k.Ord$. By the induction hypothesis, we have the sequences $e_1, \ldots, e_n$ and $e'_1, \ldots, e'_m$ for $(e, \hat{e})$ and (e'', e'), respectively. Then, the sequence $e_1, \ldots, e_n, e'_1, \ldots, e'_m$ is the desired sequence for (e, e').

- a_{k+1} is a remove request type, of requeset $\hat{e}$ by thread $\hat{t}$. By definition of the transition, $\mathsf{p}_{k+1}.Ord \subseteq \mathsf{p}_k.Ord$, and $\mathsf{p}_{k+1}.Ord$ is derived from $\mathsf{p}_k.Ord$ by first removing all pairs containing e, i.e. (e', e) or (e, e'), of those remaining pairs retaining only pairs in Dep and then taking the transitive closure. Since it is obtained by taking the transitive closure of a subset of Dep, the claim holds $\mathsf{p}_{k+1}.Ord$.

□

Lemma 2 (Ord and propagate)**.** *Let $\mathsf{p}_0 \xrightarrow{a_1}_{\mathcal{P}} \mathsf{p}_1 \ldots \xrightarrow{a_n}_{\mathcal{P}} \mathsf{p}_n$ be a POP trace and e_1, e_2 be two requests such that for some $1 \leq j \leq n$ we have $(e_1, e_2) \in \mathsf{p}_j.Ord$. Let $t \in \mathsf{p}_j.Tid$ be a thread to which neither request has propagated; i.e. $\{e_1, e_2\} \cap \mathsf{p}_j.Prop(t) = \emptyset$. If for all $i \in [j, n]$ we have $(e_1, e_2) \in \mathsf{p}_i.Ord$ and $e_1 \notin \mathsf{p}_n.Prop(t)$, then $e_2 \notin \mathsf{p}_n.Prop(t)$.*

Proof. Follows directly from the definition of the propagate transition $\langle e \rightsquigarrow t \rangle_{\mathcal{P}}$. □

Lemma 3 (Dep and Flow)**.** *Let $\mathsf{f}_0 \xrightarrow{a_1}_{\mathcal{F}} \mathsf{f}_1 \ldots \xrightarrow{a_n}_{\mathcal{F}} \mathsf{f}_n$ be a Flowing trace and e_1, e_2 be two requests such that $(e_1, e_2) \in Dep$. If for some $j \in [1, n]$ we have $e_1 \succ_{\mathsf{f}_j} e_2$ and e_1 is in $\mathsf{f}_n.Evt$ (so e_1 is neither restarted nor removed), then for all $j \leq i \leq n$, $e_2 \in \mathsf{f}_i.Evt$ implies that $e_1 \succ_{\mathsf{f}_i} e_2$.*

Proof. Follows directly from the fact that reordering of e_1 with e_2 is not possible when e_1 is in front of e_2 due to $(e_1, e_2) \in Dep$. □

Lemma 4 (Conflict and Ord)**.** *Let $\mathsf{p}_0 \xrightarrow{a_1}_{\mathcal{P}} \mathsf{p}_1 \ldots \xrightarrow{a_n}_{\mathcal{P}} \mathsf{p}_n$ be a POP trace. Let $e, e' \mathsf{p}_n.Evt$ be two fully conflicting requests; i.e. both (e, e') and (e', e) are in Dep. If $\{e, e'\} \subseteq \mathsf{p}_n.Prop(t)$ holds for some thread t, then either $(e, e') \in \mathsf{p}_n.Ord$ holds or $(e', e) \in \mathsf{p}_n.Ord$ holds.*

Proof. Consider the maximal suffix in which the condition $\{e, e'\} \subseteq \mathsf{p}_n.Prop(t)$ holds for at least one thread t. Let p_j be the state which begins this suffix. Without loss of generality assume that e was already propagated to t prior to p_j and e' is propagated to t by the transition a_j. There are two cases to consider. One case is when a_j is an accept request transition. In this case, by the definition of the accept request transition, Ord will be updated to incude the pair (e, e'). The other case is when a_j is a propagate request transition. In this case, again by the definition of the transition, Ord will be updated to include the pair (e', e). In either case, the pair remains in Ord, that is $\mathsf{p}_i.Ord$ includes the pair for all $i \geq j$ unless either e or e' is removed which we assumed does not happen in the suffix. □

Corollary 1. *If e and e' are fully conflicting and either one is fully propagated (for all t, $e \in \mathsf{p}.Prop(t)$), then either $(e, e') \in \mathsf{p}.Ord$ or $(e', e) \in \mathsf{p}.Ord$.*

Proof. Follows immediately from Lemma 4. □

Lemma 5 (Thread Partitions). *Let* p *be a* $\mathcal{T}$*-conforming POP state. Let* e, e' *be two requests in* $\mathsf{p}.Evt$.

- *If there exists some thread* t *such that* $e, e' \in \mathsf{p}.Prop(t)$, *then either* $\mathsf{p}.Seg(e) \geq \mathsf{p}.Seg(e')$ *or* $\mathsf{p}.Seg(e') \geq \mathsf{p}.Seg(e)$.
- *If there does not exist a thread* t *such that* $e, e' \in \mathsf{p}.Prop(t)$, *then* $\mathsf{p}.Seg(e)$ *and* $\mathsf{p}.Seg(e')$ *are proper descendants of* $\mathsf{p}.Seg(e) \oplus \mathsf{p}.Seg(e')$.

Proof. Both results follow immediately from the definition of hierarchical partitioning. □

4.2 Base Case – Initial System States

A Flowing state f_0 is *initial* implies that $\mathsf{f}_0.Buf(S) = \emptyset$ for all $S \in \mathsf{f}_0.Topo$; $\mathsf{f}_0.Reord = \emptyset$; $\mathsf{f}_0.Mem[l] = w_l$ for all addresses l where w_l is a hypothetical write request representing the initial value at address l. Furthermore, we set the auxiliary components such that $\mathsf{f}_0.Evt^{act} = \emptyset$, $\mathsf{f}_0.Evt = \mathsf{f}_0.Buf(S_{mem}) = W$ where W is a sequence of all w_l.

The POP initial state p_0 corresponding to f_0 is constructed by setting $ts(\mathsf{p}_0) = ts(\mathsf{f}_0)$, $\mathsf{p}_0.Tid = \mathsf{f}_0.Tid$, $\mathsf{p}_0.Evt = \mathsf{f}_0.Evt$, $\mathsf{p}_0.Ord = \emptyset$, $\mathsf{p}_0.Prop(t) = \{w \in W\}$ for all $t \in \mathsf{p}_0.Tid$, $\mathsf{p}_0.Mem = \mathsf{f}_0.Mem$. This implies that $\mathcal{I}_\sigma(\mathsf{f}_0, \mathsf{p}_0)$ as required.

4.3 Inductive Step – Transitions

Let f be a state of the Flowing system and let p be the state of the POP system such that $(\mathsf{f}, \mathsf{p}) \in \sigma$ holds. We show that if $\mathsf{f} \xrightarrow{a}_{\mathcal{F}} \mathsf{f}'$ holds, then there exists p' such that $\mathsf{p} \xrightarrow{\mu(a)}_{\mathcal{P}}{}^{*}\ \mathsf{p}'$ and $(\mathsf{f}', \mathsf{p}') \in \sigma$ hold.

In the following, for each transition we specify the precondition for the transition (f and p have to satisfy their corresponding preconditions for the transition to take place) and the next state by stating only those components of the state that change. We use a predicate $Block$ to abstract the conditions which capture the blocking condition regarding the avoidance of deadlock's in the presence of exclusive writes. The intended meaning of $Block(e, w_{\mathsf{x}}, w)$ is assumed to evaluate to true exactly when the exclusive write w_{x} is coupled with w and introducing e between these two requests (by flowing into a parent segment in Flowing or propagating to a thread in POP) can cause an eventual deadlock. We require $Block(e, w_{\mathsf{x}}, w)$ evaluates to true only when e fully conflict with both w_{x} and w; i.e. any pair that has e and one of w_{x} or w is in Dep. We choose this more abstract depiction for write exclusive related transitions in order to show that as long as both Flowing and POP use the same predicate, the simulation relation will hold (much like the Dep relation abstracting the reordering conditions).

◇ $a = \langle \dashrightarrow e \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- Assuming e is issued by thread t, then e is not in the (leaf) segment $\mathsf{f}.TofS(t)$ and t is in $\mathsf{f}.Tid$.

$\boxed{\text{Flowing Postcondition:}}$

- $\mathsf{f}'.TofS(t)$ is set to $e \cdot (\mathsf{f}.TofS(t))$.

- $\mathsf{f}'.Evt^{act}$ is set to $\mathsf{f}.Evt^{act} \cup \{e\}$.
- $\mathsf{f}'.Evt$ is set to $\mathsf{f}.Evt \cup \{e\}$.
- $\mathsf{f}'.Seg(e)$ is set to $\mathsf{f}.TofS(t)$.

◇ $\mu(a) = \langle \dashrightarrow e \rangle_{\mathcal{P}}$

$\boxed{\text{POP Precondition:}}$

- e is not in $\mathsf{p}.Evt$ and t is in $\mathsf{p}.Tid$.
  This follows from $\mathcal{I}_{evt}(\mathsf{f}, \mathsf{p})$ and $\mathcal{I}_{tid}(\mathsf{f}, \mathsf{p})$.

$\boxed{\text{POP Postcondition:}}$

- $\mathsf{p}'.Evt$ is set to $\mathsf{p}.Evt \cup \{e\}$.
  This establishes $\mathcal{I}_{evt}(\mathsf{f}', \mathsf{p}')$.
- $\mathsf{p}'.Prop(t)$ is set to $\mathsf{p}.Prop(t) \cup \{e\}$.
  This establishes $\mathcal{I}_{seg}(\mathsf{f}', \mathsf{p}')$.
- $\mathsf{p}'.Ord$ is set to $\mathsf{p}.Ord(t) \cup \{(e', e) \mid e' \in E\}$ where E is a subset of requests that have been propagated to t, i.e. $E \subseteq \mathsf{p}.Prop(t)$ such that $(e, e') \in Dep$.
  This establishes $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$.

◇ $t = \langle e \rightarrow \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- e is at the bottom of a segment S ($\mathsf{f}.Buf(S)$ is of the form $E \cdot e$).
- S' is the parent segment of S in $\mathsf{f}.Topo$.
- $Block(e, w_{\mathsf{x}}, w)$ evaluates to false for any exclusive couple w_{x} and w such that if $S_{\mathsf{x}} = \mathsf{f}.Seg(w_{\mathsf{x}})$ and $S_w = \mathsf{f}.Seg(w)$ where $S_{\mathsf{x}} \neq S'$ is not a descendant of S and $S_{\mathsf{x}} \oplus S = S'$, and $S_w > S$ (S_w is a strict ancestor of S).

$\boxed{\text{Flowing Postcondition:}}$

- $\mathsf{f}'.Buf(S) = E$
- $\mathsf{f}'.Buf(S') = e \cdot [\mathsf{f}.Buf((S')]$
- $\mathsf{f}'.Reord$ is set to $\mathsf{f}.Reord \setminus \{(e_1, e_2) \mid e_1 = e \vee e_2 = e\}$
- $\mathsf{f}'.Seg(e)$ is set to S'.

◇ $\mu(a) = \langle e \rightsquigarrow t_1 \rangle_{\mathcal{P}} \cdot \langle e \rightsquigarrow t_2 \rangle_{\mathcal{P}} \ldots \langle e \rightsquigarrow t_k \rangle_{\mathcal{P}}$, **where** $S' \setminus S = \{t_i\}$

$\boxed{\text{POP Precondition:}}$

- $e \in \mathsf{p}.Evt$.
  By $\mathcal{I}_{evt}(\mathsf{f}, \mathsf{p})$, $e \in \mathsf{f}.Evt$ implies $e \in \mathsf{p}.Evt$.

- $e \notin \mathsf{p}.Prop(t_i)$ for all $t_i \in S' \setminus S$.
  By $\mathcal{I}_{seg}(\mathsf{f}, \mathsf{p})$, p is a $\mathcal{T}$-conforming state and $\mathsf{p}.Seg(e) = \mathsf{f}.Seg(e)$. This implies that for any thread t' if $t' \notin S$ then $e \notin \mathsf{p}.Prop(t')$.

- for any e' such that $(e', e) \in \mathsf{p}.Ord$ and $e' \in \mathsf{p}.Prop(t)$, $e' \in \mathsf{p}.Prop(t_i)$ for all $t_i \in S' \setminus S$.
  By Lemma 1, there exists a sequence $e_1, \ldots, e_n$ of requests in $\mathsf{p}.Evt$ such that $e_1 = e'$, $e_n = e$ and for all $j \in [1, n)$ we have $(e_j, e_{j+1}) \in Dep$. By Lemma 3, each e_j except for $e_n = e$ must be in some ancestor of S'; i.e. $\mathsf{p}.Seg(e_j) \geq S'$. By the induction hypothesis, this implies that all e_i excluding $e_n = e$ (but including $e_1 = e'$) must satisfy $\mathsf{p}.Seg(e_i) \geq S'$. But this implies that for all $t' \in S' \setminus S$, we have $e_i \in \mathsf{p}.Prop(t')$ (except for $e_n = e$) as required.

- For any exclusive write w_{x} such that $w_{\mathsf{x}} \in \mathsf{p}.Prop(t') \setminus \mathsf{p}.Prop(t)$ and its coupled write w satisfies $(w, e) \in \mathsf{p}.Ord$, $Block(e, w_{\mathsf{x}}, w)$ evaluates to false.
  By the requirement of *Block* (see its definition above), if w and e do not conflict, then $Block(e, w_{\mathsf{x}}, w)$ cannot be true. Assume that they do. By the assumption that $(w, e) \in \mathsf{p}.Ord$ and by $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$, we must have $w \succ_{\mathsf{f}} e$. But under these conditions, $Block(e, w_{\mathsf{x}}, w)$ must also be checked for the Flowing transition; so it cannot be true.

$\boxed{\text{POP Postcondition:}}$

- $\mathsf{p}'.Prop(t_i)$ is set to $\mathsf{p}.Prop(t_i) \cup \{e\}$.
  This establishes $\mathcal{I}_{seg}(\mathsf{f}', \mathsf{p}')$.

- $\mathsf{p}'.Ord$ is the transitive closure of the minimal set that includes $\mathsf{p}.Ord$ and all pairs (e, e') such that $\mathsf{p}.Seg(e') \leq S' \setminus S$ holds and $(e, e') \in Dep$.
  This establishes $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$.

- $\mathsf{p}'.Evt$ is unchanged (equal to $\mathsf{p}.Evt$).
  This establishes $\mathcal{I}_{evt}(\mathsf{f}', \mathsf{p}')$.

◇ $a = \langle e \leftrightarrow e' \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- there is some segment S such that $S = \mathsf{f}.Seg(e) = \mathsf{f}.Seg(e')$, $(e, e') \notin \mathsf{f}.Reord$, $\mathsf{f}.Buf(S)$ is of the form $E_1 \cdot e \cdot e' E_2$.

- $(e', e) \notin Dep$.

$\boxed{\text{Flowing Postcondition:}}$

- $\mathsf{f}'.Reord$ is set to $\mathsf{f}.Reord \cup \{(e, e')\}$.

- $\mathsf{f}'.Buf(S)$ is set to $E_1 \cdot e' \cdot e \cdot E_2$.

◇ ε**, i.e. *no transition***

$\boxed{\text{POP Precondition:}}$

- `True`

$\boxed{\text{POP Postcondition:}}$

- $\mathsf{p}' = \mathsf{p}$.
  Observe that only $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$ is affected by the corresponding Flowing transition. By Lemma 1 and Lemma 3 and the fact that e and e' are adjacent in S, the only possibility of having $(e', e) \in \mathsf{p}.\mathit{Ord}$ is if $(e, e') \in \mathit{Dep}$ which is not true by the Flowing precondition. This implies that e and e' are not ordered in $\mathsf{p}'.\mathit{Ord}$ which establishes $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$.

◇ $a = \langle r \smile w \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- There exists some S such that $\mathsf{f}.\mathit{Seg}(r) = \mathsf{f}.\mathit{Seg}(w) = S$.
- r and w are consecutive in S; i.e. $\mathsf{f}.\mathit{Buf}(S) = E_1 \cdot r \cdot w \cdot E_2$ for some request sequences E_1, E_2.
- r and w are to the same address; i.e. $\mathit{Addr}(r) = \mathit{Addr}(w)$.

$\boxed{\text{Flowing Postcondition:}}$

- Send a read response $(w.\mathit{val})$ to thread t which issued r.
- Remove r; i.e. $\mathsf{f}'.\mathit{Evt}$ is set to $\mathsf{f}.\mathit{Evt} \setminus \{r\}$.
- Set $\mathsf{f}'.\mathit{Evt}^{act}$ to $\mathsf{f}.\mathit{Evt}^{act} \setminus \{r\}$.

◇ $\mu(a) = \langle t : r \smile w \rangle_{\mathcal{P}}$

$\boxed{\text{POP Precondition:}}$

- r and w have been propagated to the same threads; i.e. for all t', $r \in \mathsf{p}.\mathit{Prop}(t')$ iff $w \in \mathsf{p}.\mathit{Prop}(t')$.
  By $\mathcal{I}_{seg}(\mathsf{f}, \mathsf{p})$, for any requset e and thread t, $t \in \mathsf{f}.\mathit{Seg}(e)$ implies $e \in \mathsf{p}.\mathit{Prop}(t)$. The condition is implied by the fact that $\mathsf{f}.\mathit{Seg}(r) = \mathsf{f}.\mathit{Seg}(w)$.
- r and w are to the same address.
  Implied by the Flowing precondition.
- $(w, r) \in q_{\mathsf{p}}.\mathit{Ord}$.
  According to the definition of Dep, r and w are fully conflicting; i.e. both $(r, w) \in \mathit{Dep}$ and $(w, r) \in \mathit{Dep}$ hold. By Lemma 4, we must either have $(r, w) \in \mathsf{p}.\mathit{Ord}$ or $(w, r) \in \mathsf{p}.\mathit{Ord}$. By $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$, for any two requests $e, e' \in \mathsf{p}.\mathit{Evt}$ we have $(e, e') \in \mathsf{p}.\mathit{Ord}$ implying $e \succ_{\mathsf{f}} e'$. Since $w \succ_{\mathsf{f}} r$ holds, we must have $(w, r) \in \mathsf{p}.\mathit{Ord}$.
- if there is a request e ordered after w and before r, then e must be a fully propagated access to a different address; i.e. if there is $e \in \mathsf{p}.\mathit{Evt}$ such that $(w, e) \in \mathsf{p}.\mathit{Ord}$ and $(e, r) \in \mathsf{p}.\mathit{Ord}$, then $\mathsf{p}.\mathit{Seg}(e) = S_{root}$ and $\mathit{Addr}(e) \neq \mathit{Addr}(r)$.
  By $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$, $(w, e) \in \mathsf{p}.\mathit{Ord}$ and $(e, r) \in \mathsf{p}.\mathit{Ord}$ imply together that w and r cannot be adjacent in $\mathsf{f}.\mathit{Seg}(r)$, contradicting the Flowing precondition.

$\boxed{\text{POP Postcondition:}}$

- Send a read response $(w.val)$ to thread t which issued r.
  This establishes $\mathcal{I}_{tss}(\mathsf{f}', \mathsf{p}')$.
- Remove r from $\mathsf{p}.Evt$.
  This establishes $\mathcal{I}_{evt}(\mathsf{f}', \mathsf{p}') \wedge \mathcal{I}_{seg}(\mathsf{f}', \mathsf{p}')$.
- Set $\mathsf{p}'.Ord$ to the transitive closure of $(\mathsf{p}.Ord \setminus \{(e_1, e_2 \mid e_1 = r \vee e_2 = r\}) \cap Dep$.
  This establishes $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$.

◇ $a = \langle r \dashrightarrow \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- r is at the bottom of S_{root}; i.e. $q_{\mathsf{f}}.Buf(S_{root}) = E \cdot r$.

$\boxed{\text{Flowing Postcondition:}}$

- Send a read response $(\mathsf{f}.Mem[Addr(r)].val)$ to thread t which issued r.
- Remove r from storage subsystem; i.e. $\mathsf{f}'.Evt$ is set to $\mathsf{f}.Evt \setminus \{r\}$.
- Set $\mathsf{f}'.Evt^{act}$ to $\mathsf{f} \setminus \{r\}$.

◇ $\mu(a) = \langle t : r \smile \hat{w} \rangle_{\mathcal{P}}$, where $\hat{w} = \mathsf{p}.Mem[Addr(r)]$

$\boxed{\text{POP Precondition:}}$

- r and w have been propagated to the same threads; i.e. for all t', $r \in \mathsf{p}.Prop(t')$ iff $w \in \mathsf{p}.Prop(t')$.
  By $\mathcal{I}_{seg}(\mathsf{f}, \mathsf{p})$, $\mathsf{p}.Seg(r) = S_{root}$ implying that for all t we have $r \in \mathsf{p}.Prop(t)$; i.e. r is fully propagated. By definition, $\hat{w}$ is also fully propagated.
- r and $\hat{w}$ are to the same address.
  Implied by the definition of $\hat{w}$.
- $(\hat{w}, r) \in \mathsf{p}.Ord$.
  According to the definition of Dep, r and $\hat{w}$ are fully conflicting; i.e. both $(r, \hat{w}) \in Dep$ and $(\hat{w}, r) \in Dep$ hold. By Lemma 4, we must either have $(r, \hat{w}) \in \mathsf{p}.Ord$ or $(\hat{w}, r) \in \mathsf{p}.Ord$. By $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$, for any two requests $e, e' \in \mathsf{p}.Evt$ we have $(e, e') \in \mathsf{p}.Ord$ implying $e \succ_{\mathsf{f}} e'$. Since $\hat{w} \succ_{\mathsf{f}} r$ holds, we must have $(\hat{w}, r) \in \mathsf{p}.Ord$.
- if there is a request e ordered after $\hat{w}$ and before r, then e must be a fully propagated access to a different address; i.e. if there is $e \in \mathsf{p}.Evt$ such that $(\hat{w}, e) \in \mathsf{p}.Ord$ and $(e, r) \in \mathsf{p}.Ord$, then $\mathsf{p}.Seg(e) = S_{root}$ and $Addr(e) \neq Addr(r)$.
  Assume the contrary and let w be a write request to $Addr(r)$ such that both $(\hat{w}, w) \in \mathsf{p}.Ord$ and $(w, r) \in \mathsf{p}.Ord$ hold. By $\mathcal{I}_{ord}(\mathsf{f}, \mathsf{p})$, $(\hat{w}, w) \in \mathsf{p}.Ord$ and $(w, r) \in \mathsf{p}.Ord$ imply $\hat{w} \succ_{\mathsf{f}} w$ and $w \succ_{\mathsf{f}} r$, respectively. By Lemma 3 and the fact that both $(\hat{w}, w) \in Dep$ and $(w, r) \in Dep$

hold, this implies that $\hat{w}$ and w have both flown into memory before r reaches the bottom of S_{root}. In other words, the Flowing trace must be of the following form:

$$\mathsf{f}_0 \rightsquigarrow \cdot \xrightarrow{\langle \hat{w} \dashrightarrow \rangle_{\mathcal{F}}}_{\mathcal{F}} \cdot \rightsquigarrow \cdot \xrightarrow{\langle w \dashrightarrow \rangle_{\mathcal{F}}}_{\mathcal{F}} \cdot \rightsquigarrow \mathsf{f}$$

That is, $\hat{w}$ must have left S_{root} and reached memory before w did. However this implies that $\mathsf{p}.Mem[Addr(r)] \neq \hat{w}$ contradicting the assumption that $\hat{w}$ denotes $\mathsf{p}.[Addr(r)]$.

$\boxed{\text{POP Postcondition:}}$

- Send a read response ($\hat{w}.val$) to thread t which issued r.
  Same as the Flowing postcondition; this establishes $\mathcal{I}_{tss}(\mathsf{f}', \mathsf{p}')$.
- Remove r.
  Same as the Flowing postcondition; this establishes $\mathcal{I}_{evt}(\mathsf{f}', \mathsf{p}') \wedge \mathcal{I}_{seg}$.
- Set $\mathsf{p}'.Ord$ to the transitive closure of $(\mathsf{p}.Ord \setminus \{(e_1, e_2 \mid e_1 = r \vee e_2 = r\}) \cap Dep$.
  This establishes $\mathcal{I}_{ord}(\mathsf{f}', \mathsf{p}')$.

◇ $a = \langle w \dashrightarrow \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- w is at the bottom of S_{root}; i.e. $\mathsf{f}.Buf(S_{root}) = E \cdot w$.

$\boxed{\text{Flowing Postcondition:}}$

- Update memory to map the address of w to point to w; i.e. $\mathsf{f}'.Mem(Addr(w))$ is set to w.
- Set $\mathsf{f}'.Evt^{act}$ is set to $\mathsf{f}.Evt^{act} \setminus \{w\}$.
- Set $\mathsf{f}'.Buf(S_{root}) = E$.
- Set $\mathsf{f}'.Buf(S_{mem})$ to $w \cdot \mathsf{f}.Buf(S_{mem})$.

◇ $\mu(a) = \varepsilon$[3]

$\boxed{\text{POP Precondition:}}$

- True.

$\boxed{\text{POP Postcondition:}}$

- $\mathsf{p}'.Seg(w)$ is set to S_{mem}.
  This establishes $\mathcal{I}_{seg}(\mathsf{f}, \mathsf{p})$. The remaining invariants are not affected by the change to the f; i.e. for all invariants $\mathcal{I} \neq \mathcal{I}_{seg}$, we have $\mathcal{I}(\mathsf{f}, \mathsf{p})$ implies $\mathcal{I}(\mathsf{f}', \mathsf{p}')$.

[3]To be more precise we do allow a silent transition which we use to update only the auxiliary components of p. In this sense, our simulation is *weak* because it projects out observably stuttering steps. In order to relieve the reader from extra technicalities not essential to the proof, we choose to omit explicit constructions to cover these stuttering steps.

◇ $a = \langle b \dashrightarrow \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- b is at the bottom of S_{root}; i.e. $\mathsf{f}.Buf(S_{root}) = E \cdot b$.

$\boxed{\text{Flowing Postcondition:}}$

- Set $\mathsf{f}'.Evt^{act}$ to $\mathsf{f}.Evt^{act} \setminus \{b\}$.
- Set $\mathsf{f}'.Buf(S_{root})$ to E.
- Set $\mathsf{f}'.Buf(S_{mem})$ to $b \cdot \mathsf{f}.Buf(S_{mem})$.

◇ $\mu(a) = \varepsilon$

$\boxed{\text{POP Precondition:}}$

- True.

$\boxed{\text{POP Postcondition:}}$

- $\mathsf{p}'.Seg(b)$ is set to S_{mem}.
  This case is similar to the previous transition.

◇ $a = \langle \dashrightarrow w_{\mathsf{x}} \smile w \rangle_{\mathcal{F}}$

$\boxed{\text{Flowing Precondition:}}$

- The preconditions of $\langle \dashrightarrow w_{\mathsf{x}} \rangle_{\mathcal{F}}$ hold.
- The requests w_{x} and w are to the same address; i.e. $Addr(w_{\mathsf{x}}) = Addr(w)$.
- For all requests e such that both $w \succ_{\mathsf{f}} e$ and $e \in \mathsf{f}.Prop(t)$ where t is the thread issuing w_{x} hold, e is not a full barrier (dmb sy), or a write barrier (dmb st), and if e is a write request and $Addr(e) = Addr(w)$, then e is issued by t, e is not a write-exclusive request and if there is a write exclusive request w'_{x} such that $Addr(w'_{\mathsf{x}}) = Addr(e)$ and $e \succ_{\mathsf{f}} w'_{\mathsf{x}}$, then there is another write request w' such that $Addr(w') = Addr(w'_{\mathsf{x}})$, $w \succ_{\mathsf{f}} w' \succ_{\mathsf{f}} w'_{\mathsf{x}}$ hold.

$\boxed{\text{Flowing Postcondition:}}$

- Same as $\langle \dashrightarrow e \rangle_{\mathcal{F}}$ with e replaced with w_{x}.

◇ $\mu(a) = \langle \dashrightarrow w_{\mathsf{x}} \smile w \rangle_{\mathcal{P}}$

$\boxed{\text{POP Precondition:}}$

- The preconditions of $\langle \dashrightarrow w_{\mathsf{x}} \rangle_{\mathcal{P}}$ hold.
  This is implied by the Flowing precondition and by the previous analysis of $\langle \dashrightarrow e \rangle_{\mathcal{P}}$.

- w is in the storage subsystem; $w \in \mathsf{p}.Evt$.
  This is implied by the Flowing precondition and by $\mathcal{I}_{evt}(\mathsf{f},\mathsf{p})$.

- w and w_{x} are to the same address.
  This is implied by the Flowing precondition.

- w is not the immediate predecessor of a different exclusive write request. That is, there does not exist another exclusive write w'_{x} such that $(w, w'_{\mathsf{x}}) \in \mathsf{p}.Ord$ and there is no write w' to $Addr(w)$ such that $(w, w') \in \mathsf{p}.Ord$ and $(w', w'_{\mathsf{x}}) \in \mathsf{p}.Ord$.
  If such an w'_{x} existed, then by the fact that they are fully conflicting and by $\mathcal{I}_{ord}(\mathsf{f},\mathsf{p})$, the Flowing precondition would not be satisfied.

- Any write request w' such that $(w, w') \in \mathsf{p}.Ord$ and $w' \in \mathsf{p}.Prop(t)$, where t is the thread that issued w_{x}, and such that w' is coupled with some exclusive write w'_{x}, is to a different address; i.e. $Addr(w') \neq Addr(w)$.
  If such an w' did exist, then by the fact that w' and w are fully conflicting and by $\mathcal{I}_{ord}(\mathsf{f},\mathsf{p})$ and $\mathcal{I}_{seg}(\mathsf{f},\mathsf{p})$, the Flowing precondition would not be satisfied.

- For every request $e \in \mathsf{p}.Prop(t)$ and $(w, e) \in \mathsf{p}.Ord$ hold, if e is issued by thread $t' \neq t$, then $Addr(e) \neq Addr(w_{\mathsf{x}})$, e is not a full barrier (`dmb sy`) or a write barrier (`dmb st`) unless e is fully propagated.
  If e is not fully propagated, then by $\mathcal{I}_{ord}(\mathsf{f},\mathsf{p})$ and by the fact that e would be fully conflicting in any of those cases, the Flowing precondition would also not hold.

POP Postcondition:

- Same as $\langle \dashrightarrow e\rangle_{\mathcal{P}}$ with e replaced with w_{x}.

◇ $a = \langle \dashrightarrow w_{\mathsf{x}} \smile \rangle_{\mathcal{F}}$

Flowing Precondition:

- Same as the previous case where w is replaced with $\mathsf{f}.Mem[Addr(w_{\mathsf{x}})]$.

Flowing Postcondition:

- Same as $\langle \dashrightarrow e\rangle_{\mathcal{F}}$.

◇ $\mu(a) = \langle \dashrightarrow w_{\mathsf{x}} \smile \hat{w}\rangle_{\mathcal{P}}$, where $\hat{w} = \mathsf{p}.Mem[Addr(w_{\mathsf{x}})]$

POP Precondition:

- Same as the previous case.

POP Postcondition:

- Same as the previous case.