Emerging non-volatile memory (NVRAM) technologies offer the durability of disk with byte-addressability and access latencies similar to DRAM. Future systems will likely attach these NVRAMs to a DRAM-like memory bus [1, 2]. Such systems enable the construction of high performing, in-memory, recoverable data structures (RDS) [1, 2]. The tenets of creating RDSs revolve around ordering writes to the data structure. However, existing architectures do not provide efficient mechanisms to order writes all the way through NVRAM. Pelley, Chen, and Wenisch [3] introduce persistency models (drawing on memory consistency model research) to reason about and order writes to NVRAM. Here, we introduce notation to concisely and precisely define the models to help draw better parallels to existing memory consistency models. These precise definitions make it easier to reason about the interaction between instruction execution and NVRAM write order. Achieving the desired order of NVRAM writes across threads is tricky. We show two generic coding patterns to illustrate how to leverage persistency models to achieve the desired order of writes. In particular, one code pattern (observe) leverages relaxed persistency models and can be used to enforce only the absolute minimum NVRAM write orderings required for correct recovery.

1. Memory persistency models

Memory events: We formalize Pelley’s persistency models in terms of order relations over memory events. We consider two kinds of events, loads and stores (to persistent or volatile address spaces), which we collectively call memory accesses. We use the term “persist” to refer to the act of durably writing a store to persistent memory. We assume persists are performed atomically (with respect to failures) at 8-byte granularity. By “thread”, we refer to execution contexts—cores or hardware threads. We use the following notation:

- \( L_i^a \): A load from thread \( i \) to address \( a \)
- \( S_i^a \): A store from thread \( i \) to address \( a \)
- \( M_i^a \): A load or store by thread \( i \) to address \( a \)

Persist memory order: We reason about two ordering relations over memory events. Volatile memory order (VMO) is an ordering relation over all memory events (loads, stores and their values) as prescribed by the consistency model. Persist memory order (PMO) comprises the same events, however, events are instead ordered by the constraints imposed by the persistency model. We denote these ordering relations as:

- \( A \preceq_v B \): A occurs no later than B in VMO
- \( A \preceq_p B \): A occurs no later than B in PMO

An ordering relation between stores in PMO implies the corresponding persist actions are ordered; that is, \( A \preceq_p B \rightarrow B \) may not persist before \( A \).

Strong persist atomicity: Memory consistency models often guarantee that stores to a single address are serialized (store atomicity). Persistency models could guarantee persist atomicity, persists to the same address are serialized. Pelley argues persistency models should provide strong persist atomicity (SPA), to preclude non-intuitive behavior (e.g., recovery to states unreachable under fault-free execution). SPA requires that conflicting accesses (accesses to the same address, at least one being a store) ordered in VMO are also ordered in PMO.

\[
S_i^a \preceq_v M_i^a \rightarrow S_i^a \preceq_p M_i^a
M_i^a \preceq_v S_i^a \rightarrow M_i^a \preceq_p S_i^a
\]

SPA is guaranteed by all the persistency models described below. SPA is critical to constructing precise ordering dependences across threads (see section 2).

Strict persistency: Under strict persistency, the consistency model governs both VMO and PMO; that is, the two are identical. So, for any two stores ordered by the consistency model, the corresponding persists are also ordered. Under strict persistency:

\[
M_i^a \preceq_v M_i^b \leftrightarrow M_i^a \preceq_p M_i^b
\]

While strict persistency is the most intuitive of the persistency models, it is not the best performing. By ordering persists per VMO, strict persistency enforces orderings typically not required for recovery correctness [3].

Relaxed persistency: Relaxed persistency models govern PMO independent of the memory consistency model. These models provide programmers with additional memory events, to prescribe only those ordering constraints in PMO required to ensure correct recovery.

Epoch persistency: The epoch persistency model, a relaxed persistency model, introduces a new memory event, the “persist barrier” (different from memory consistency barriers). We denote persist barriers issued by thread \( i \) as \( PB^i \). Under epoch persistency, any two memory accesses on the same thread separated by a persist barrier are ordered in PMO.

\[
M_i^a \preceq_v PB^i \preceq_v M_i^b \rightarrow M_i^a \preceq_p M_i^b
\]

Persist barriers separate a thread’s execution into ordered epochs (persists from an epoch are concurrent). Epoch persistency is similar to the model described in [2], though Pelley introduces some subtle differences.
The central intuition is to leverage the conflict-}


persists to A and B under epoch persistency using persist
barriers \( PB^1 \) and \( PB^2 \). We denote the unlock operation
on thread 1 as \( S^1_L \) and the lock operation on thread 2 as
\( S^2_L \). The program orders of thread 1, thread 2 and the
ordering property of persist barriers (eq. 3) ensures that:
\[
S^1_A \leq_v PB^1 \leq_v S^1_L \rightarrow S^1_A \leq_p S^1_L \\
(5)
\]
\[
S^2_L \leq_v PB^2 \leq_v S^2_B \rightarrow S^2_L \leq_p S^2_B \\
(6)
\]
From conflicting accesses to lock L and SPA (eq 1)
\[
S^1_L \leq_v S^2_L \rightarrow S^1_L \leq_p S^2_B \\
(7)
\]

By transitivity and eqs. 5-7, we ensure that \( S^1_A \leq_p S^2_B \). This same reasoning extends to strands (instead
of threads) under strand persistency.

Observe: Instead of relying on lock L for conflicting
accesses, we can explicitly observe (using loads) the specific
addresses after which subsequent persists should be
ordered, and then issue a persist barrier. Figure 1b illus-
trates this pattern. \( S^1_A \)'s persist is unordered with
\( S^2_B \). Again, the above reasoning extends to strands (instead
of threads) under strand persistency.

To conclude, we introduced precise definitions for per-
sistency models and used the definitions to detail generic
approaches to enforce the desired order of persists.

References

[1] J. Coburn, A. Caulfield, A. Akel, L. Grupp,
R. Gupta, R. Jhala, and S. Swanson. Ny-heaps:
Making persistent objects fast and safe with next-
generation, non-volatile memories. In Proceedings of
ASPLOS, 2011.

D. Burger, and D. Coetzee. Better i/o through byte-
addressable, persistent memory. In Proceedings of
SOSP, 2009.