# Cache Coherency & Memory Model in RDMA-Backed Software-Coherent DSM

Zhengyi Chen

January 30, 2024

## Table of Contents

## 1. Overview

- DSM used to be constrained by NIC bandwidth & transfer rate (e.g., during the 1990s).
- The advent of high(er) transfer rate NICs allows the DSM idea to be revived.
- Orthogonally, hardware acceleration resources are scarce and highly valuable.
  - Traditional Scheduling Mechanisms within a Cluster cannot dynamically allocate hardware accelerators without high overhead.
- Ideally, via high-speed NICs, hardware accelerator could be statically allocated such that:
  - Every node have access to the hardware accelerator node in a time-shared fashion.
  - Accelerator-attached node can access remote memory much like attaching accelerator over, say, PCIe.

## Heterogeneous Memory Management

- **HMM** facilitates shared address space and transparent data migration between CPU and peripherals. Specifically:
  - HMM provides interface for duplicating the CPU page table with that of the device's, which are transparently synchronized.
  - It also provides corresponding `struct page` representation of device memory pages, which are faulted between the CPU and device.
- Theoretically, this should allow for devices in remote nodes to perform HMM using the DMA-capable NIC as a "proxy HMM device".
- Details of implementation of DSM-over-HMM is beyond this thesis's scope.
  - This thesis focuses on studying and implementing cache coherency and later, memory model for the DSM subsystem of this wider, ongoing project.

## Cache Coherency, and Why It Matters Here

- Cache-incoherent RDMA (e.g., mlx) performs DMA without synchronization with CPU cache.
- We cannot assume MMU to magically automatically maintain coherence.
- At transportation time:
  - Send to remote: flushes cache into memory before posting send message.
  - Receive from remote: invalidate cache entry after worked recv message.
- Example: Linux kernel tree, *smbdirect* implementation.
  - *smbdirect* opportunistically establish SMB over RDMA-capable network.
  - `smbd_post_send` cleans cache entry prior to posting send request.
  - `recv_done` invalidates cache entry after exiting softirq for recv request (as callback from RDMA driver).

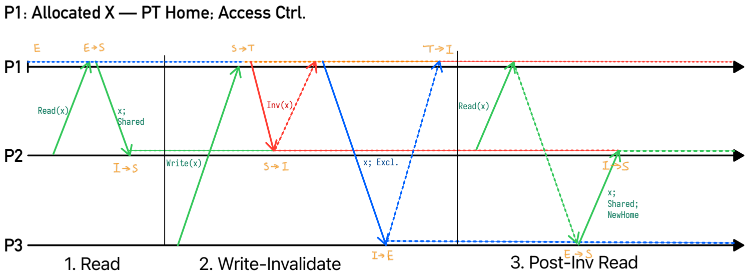
## Consistency Model and Protocol

- Majority of DSM literatures apply **release consistency** as the system's memory model.
- With **single-writer** protocol, however, the memory model can be strengthened with little increase in code complexity.
  - *DSPM*[1], for example, achieves a *de-facto* TSO consistency from its multi-writer release consistency counterpart – assuming correct memory barriers within each node's CPU, distributed writes are never reordered, and distributed reads can overtake writes.
  - Consequently, one can easily achieve sequential consistency by designating the entire write-access duration as a critical section.
- HMM's “CPU-or-device” data migration model also strongly implies a single-writer consistency protocol.

## 2. Design

- Designing a DSM necessitates designing:
  - Consistency Model.
  - Coherence Protocol and State Machine.
  - Access Control.
- Care needs to be taken to ensure that the in-kernel implementation is:
  - Correct,
  - Performant,
  - Exploits RDMA's traits.

# Protocol Excerpt: Write-Invalidate



The $T$-state indicates a transitionary state for some shared page.

## Consistency Model: TSO

- Total Store Ordering allows Reads to bypass Stores.
- Assuming correct use of node-local synchronization on all nodes, applying TSO in a home-based DSM allows for:
  - When another node tries to read T-page from access-control node: W→R violation.
  - When another node tries to read S-page from data-provider nodes: W→R violation (if e.g., the invalidation message from access-control node was received afterwards).
  - Data-provider and access-control nodes work on one request at a time: no R→W violation.
  - Write-accesses serialized at access-control node: no W→W violation.

## Consistency Model: Strengthen to Sequential

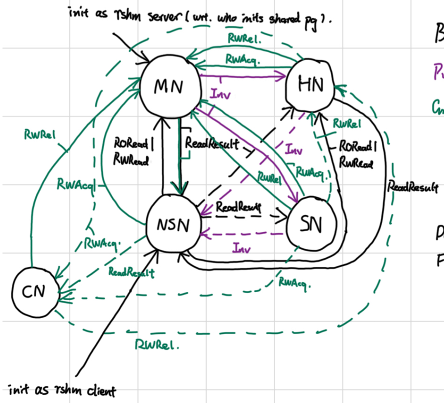
- By corollary, can reverse the previous page's statements to strengthen to sequential consistency:
    - Disallow T-pages from being serviced until new page content is installed: lengthens critical section.
    - Abolish data-provider nodes: access-control nodes become bottleneck.

## Coherence Protocol: Possible Features

- Multi-data-provider Protocol: Instead of having one data-provider, have multiple data-provider nodes that are automatically write-back to prevent network bottleneck.
  - Data provider nodes may be dynamically assigned.
  - Extra metadata can limit scalability.
- Auto-share: likewise, write-back pages to non-data-provider nodes to take advantage of 1-sided communications.
- Request aggregation: aggregate RDMA transfers for optimal transfer performance.
  - Need to be coherent with program sequence!
  - Enables write-request merging.

## Stateful Nodes & Transitions (Provisional)

- Nodes (e.g., within the cluster) become tightly bound with the properties of each shared page(s).



Black — Non-committal sharing path.

Purple — Invalidation path.

Green — Committal sharing path.

Dashed transitions are internal, happens @ reception.

Filled transitions are messages.

*. Cache entries can be Valid / Invalid.

## Stateful Nodes & Transitions (Provisional) (Cont.)

- MN (Manager Nodes): Provide access-control and (fallback) data-provision.
- HN (Home Nodes): Provide data-provision. Can be write-back or write-invalidate.
- SN (Sharer Nodes): Share data within a reader-only “epoch”. Can be write-back or write-invalidate.
- NSN (Non-sharer Nodes): Nodes in network without sharing the particular page(s).
- CN (Commit Node): Node that acquired the single-writer access to the shared page.
- Message variants are not finalized:
  - Goal: Composable message chains that allow for “piggy-backing” of multiple procedures.

## Stateful Nodes: Transition Paths

- Filled line transitions indicate local requests remote to perform state transition.
- Dashed line transitions indicate local implicitly transitions prior to sending request to remote.
- *Non-committal* path concerns about read-only and copy-on-write sharing. Sharers cannot make global modification to cached local data.
- *Invalidation* path is duo with commit operations (due to write-invalidation).
- *Committal* path concerns about global write sharing. Only one writer is allowed to write and commit at one time.
- Problem: How exactly to integrate RDMA remote read/write into this?

## Progress

- Goal: in-kernel implementation of software cache-coherency via non-coherent RDMA hardware.
- Optimistic Goal: in-kernel implementation of memory model in DSM.
- Progress: studied and isolated mechanism for data cache invalidation/flushing in ARM64, which allows the DSM to run in heterogeneous ISA clusters.
- Integration with kernel & main DSM kernel module remains at hand: is it absolutely necessary to export new symbols for such an important operation?

## On-demand Coherency in ARM64

- ARMv8 defines two levels of cache coherence:
  - *Point-of-Unification*:
  - *Point-of-Coherence*:

## Kernel Patch for On-demand Coherency

## Proof-of-Concept Kernel Module