RPMEM 2.0

Intel next generation platform support for RDMA with PMEM

09/24/19

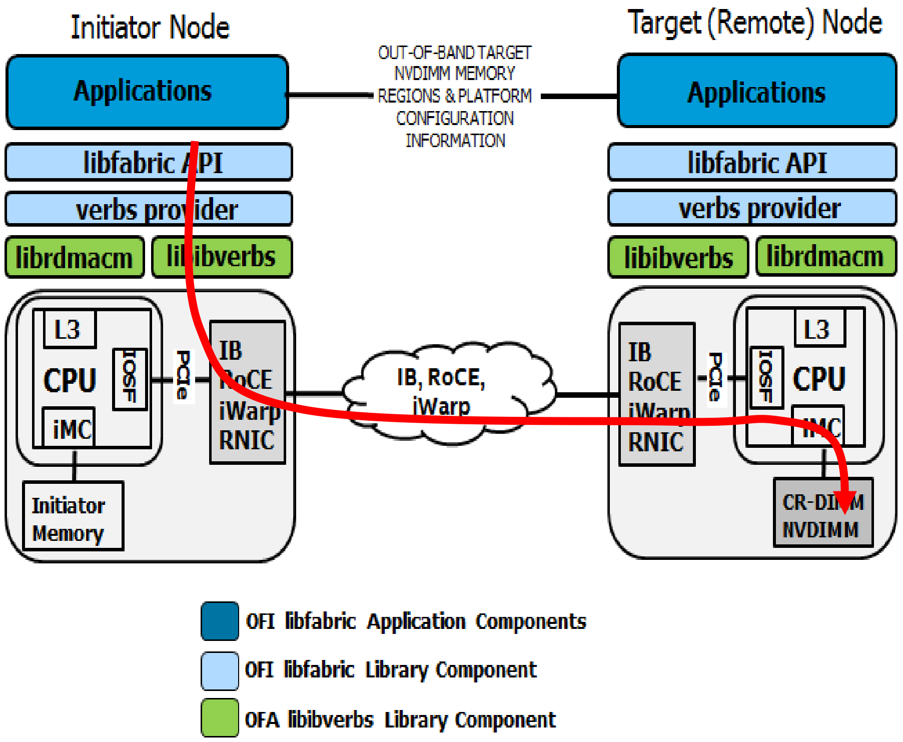
Chet Douglas - DCG NVMS

DCG
Data Center Group

Remote Persistent Memory (RPMEM) 2.0

Intel Platform Value Prop (Cloud, Enterprise, HPC)

- Increase performance with PMEM Wire protocol changes
  - Remove usage of non-allocating Writes
  - Remove extra messaging for some use cases
  - Eliminate Target node interrupts & core message handling
- Accelerate and reduce CPU utilization through improved NIC offload
- Create a standards-based solution for native remote persistent memory support. PMEM extensions being added to the following public standards/API:
  - IBTA IB/RoCE wire protocol
  - IBTA verbs spec
  - IETF iWARP wire protocol
  - ACPI PCI Firmware Spec
  - OFA libibverbs & libfabric API



- **High-performance standards-based solution in collaboration with network partners**
- **Native PMEM HW support for RNICs, CPUs, and platforms**

RPMEM 1.0 – Basic RDMA with PMEM for Intel Purley platform

See backup slides for basic overview of the 2 RPMEM 1.0 solutions Intel supports

PROS of using the RPMEM 1.0 solution

- ULP SW changes only
  - No wire protocol changes
  - No OFA library changes
  - No HW RNIC, cpu, chipset changes required

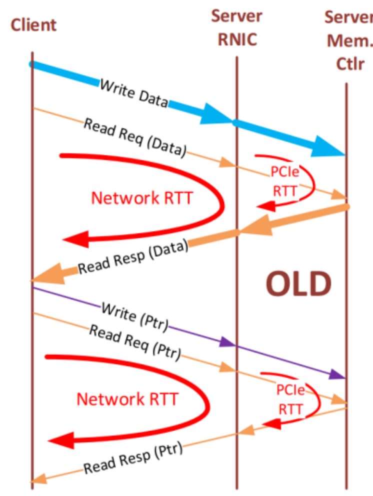
CONS of using these RPMEM 1.0 techniques

- Vendor specific implementation does not work for all platforms and does not foster industry wide acceptance
  - ULP SW changes could become a defacto protocol (not ideal!)
  - Appliance Method only works on Intel late model server platforms
- Initiator Node drives 2 possible flushing mechanisms
  - Forces target node configuration to be understood by initiator node – Not a sustainable or scalable programming model
- Performance Impact:
  - Extra messaging required to make writes durable – for some use cases
  - RPMEM 1.0 customers prefer the non-allocating write “Appliance Method” since the latency is dramatically reduced versus the DDIO “General Purpose Server Method”
  - Architectural limitations force the user/admin to place an entire PCIe Root Port in to non-allocating write mode
    - Requires intimate/internal knowledge of motherboard layout, add-in card strategy, etc
    - Impacts write performance for all devices on the root port
    - Alters the way all SW running on that node and data caching strategy
    - Applications that normally see quick read performance when accessing data that was RDMA’d potentially now will see significant latency as the first touch pulls the RDMA Write data in to LLC cache from CR

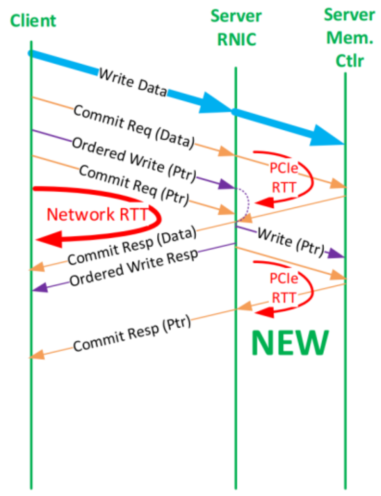
RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform

Why we need "RDMA extensions for PMEM"?

- Platform Performance:
  - One less round trip on the wire for tail of log SQL use case (see picture on the right)
    - The Data write, its Commit, the Ordered tail pointer Write and its Commit can all be launched by client in pipelined fashion.
    - The serialization of the tail pointer write relative to the Data Commit must still occur **within** the target system, but those in-system serialization latencies are noticeably lower than the equivalent serialization over the network.
  - Platform steering tags in combination with on the wire knowledge of RDMA Writes to persistent memory allow specific IO to bypass LLC cache for better efficiency
  - No longer need to place entire root port in to non-allocating mode, allowing non RPMEM users to see expected platform DDIO performance not possible with the ULP SW solution of RPMEM 1.0
- Platform efficiency/scalability, Initiator Node SW simplicity:
  - Only those applications require RPMEM need to enable it
  - Initiator node connection no longer needs to understand specific setup and configuration of each target node
  - No longer need to utilize different flush semantics for different target nodes
- Flexibility for future HW and platform architectures
  - Steering tags allow precise routing of network transactions to separate HW entities

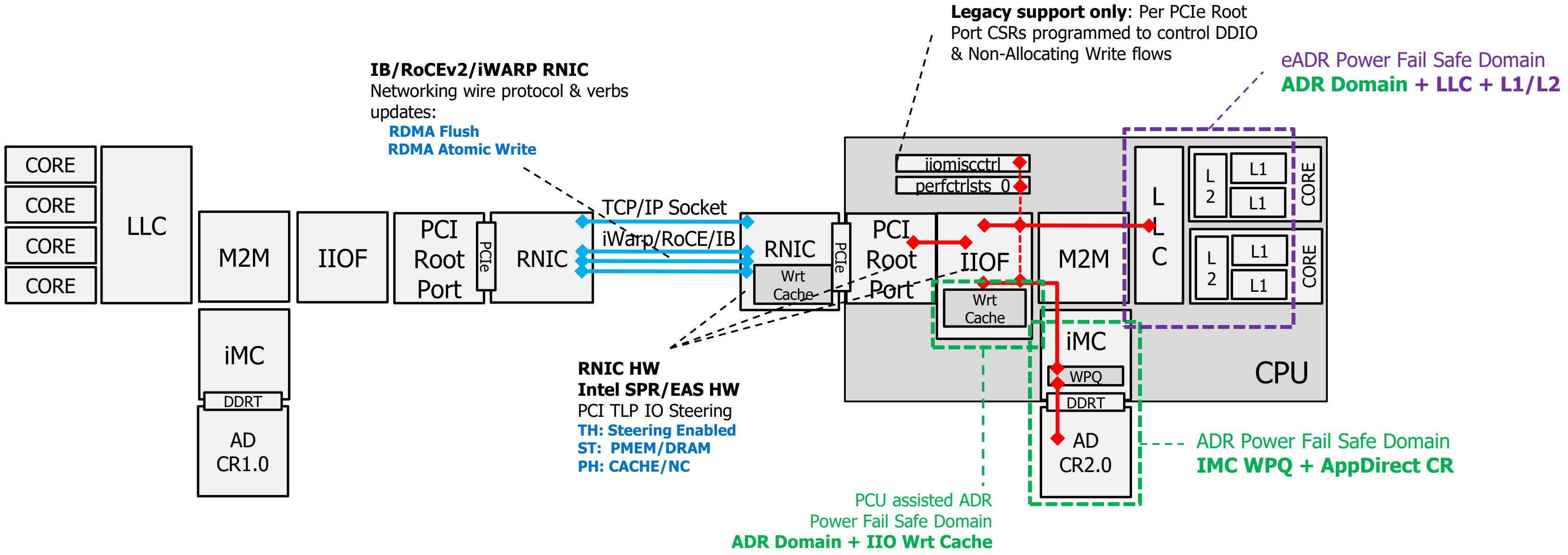


INTEL RPMEM 1.0



INTEL RPMEM 2.0

RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform



RPMEM 2.0 Intel Eagle Stream Platform HW Architecture

RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform

RDMA Flush placement attributes:

-Attributes sent with the RDMA Flush command that directs the HW on how to handle the RDMA Writes and flushes to the given QP
FI_PLACEMENT_PMEM – The data being flushed it targeted at PMEM vs DDR
FI_PLACEMENT_GO - The data being flushed must be Globally Ordered (visible) before the flush command completes

RDMA Flush:

-Force flush of previous RDMA Write data utilizing placement hints to optimize how the flushing takes place
Ordering:
Behaves the same as RDMA Read
RDMA Flush shall not begin execution until all previous operation on the QP have been executed
Subsiquent RDMA Write & RDMA Atomics may bypass RDMA Flush
RDMA Atomic Write may not bypass RDMA Flush

RDMA Atomic Write:

-8 Byte pointer update only
-Separating from RDMA Write - Allows RNIC pipelining & ordering for these updates orthogonal to RDMA Write
-Optional IMMEDIATE flag to generate target node SW interrupt
Ordering:
Wont begin execution until ALL previous received inbound requests have begun executing
Subsequent inbound RDMA Write, RDMA Send, RDMA Atomic operations can all bypass the Atomic Write
Atomic Write shall not complete in memory until all preceding operations, except RDMA Read, have been completed in memory

IEFT RDMA Write w Validate:

-End-to-end CRC generation and checking when writing directly in to PMEM.
-Important for Push model with PMEM
Ordering:
Write completion back to the initiator after CRC validation is complete

PCI TLP Processor Hint Definition

- TH – indicates the presence of TLP Processing Hints (TPH) in the TLP header and optional TPH TLP Prefix (if present for 16-bit ST)
- PH bits are hints used by completer to optimize internal data placement for anticipated subsequent access by device or host.
- Steering Tags are system-specific values that provide information about the host or cache structure in the system cache hierarchy. These values are used to associate processing elements within the platform with the processing of Requests.

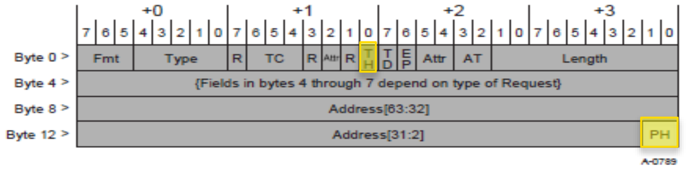


Figure 2-20: Location of PH[1:0] in a 4 DW Request Header

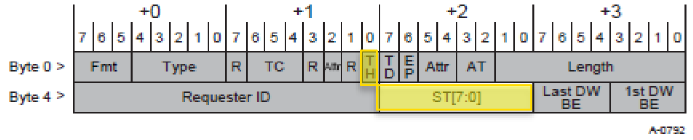
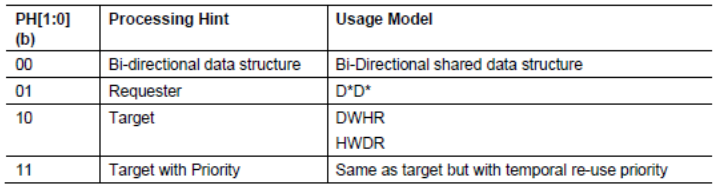


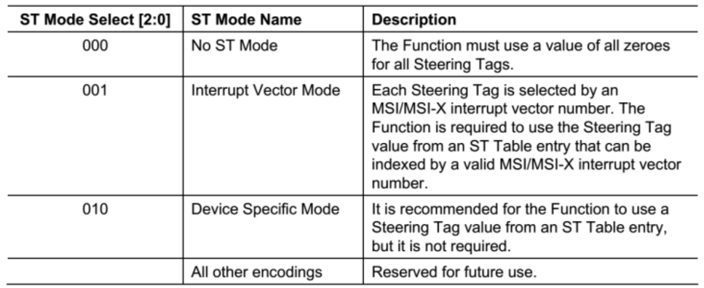
Figure 2-22: Location of ST[7:0] in the Memory Write Request Header

Table 6-11: Processing Hint Mapping



PH[1:0] (b)	Processing Hint	Usage Model
00	Bi-directional data structure	Bi-Directional shared data structure
01	Requester	D*D*
10	Target	DWHR HWDR
11	Target with Priority	Same as target but with temporal re-use priority

Table 6-12: ST Modes of Operation



ST Mode Select [2:0]	ST Mode Name	Description
000	No ST Mode	The Function must use a value of all zeroes for all Steering Tags.
001	Interrupt Vector Mode	Each Steering Tag is selected by an MSI/MSI-X interrupt vector number. The Function is required to use the Steering Tag value from an ST Table entry that can be indexed by a valid MSI/MSI-X interrupt vector number.
010	Device Specific Mode	It is recommended for the Function to use a Steering Tag value from an ST Table entry, but it is not required.
	All other encodings	Reserved for future use.

DWHR: Device writes then host reads soon

HWDR: Device reads data that the Host is believed to have recently written

D*D*: Device writes/reads, then device reads/writes soon

Includes DWDW, DWDR, DRDW, DRDR

Bi-Directional: Data structure that is shared and has equal read/write access by host and device.

RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform

TLP Hints:	TH	PH	ST	Durable/Volatile
Volatile Memory Write (DRAM)	0	na	na	Global Visible Order
Volatile Write w/o caching hint	1	00/11	!A	Global Visible Order
Volatile Memory Write **NO caching**	1	01	!A	
Volatile Memory Write **w/ Caching**	1	10	!A	
Persistent Write w/o caching hint	1	00/11	A	Persistent Ordered
Persistent Write **NO caching**	1	01	A	
Persistent Write **w/ Caching**	1	10	A	

PCI TLP Processor Hint Definition

A = Value programmed via new ACPI interfaces or default value if one is programmed

-fi_mr_reg memory registration attributes -RDMA Write memory registration attributes -RDMA Flush placement attributes	Backend RNIC PCIe Steering Tag output	Description
-FI_PLACEMENT_PMEM -FI_PLACEMENT_GO	-TH = 1 -ST [7:0] = A -PH [1:0] = 10	-Access PMEM with global visibility
-FI_PLACEMENT_GO	-TH = 1 -ST [7:0] = A -PH [1:0] = 10	-Access DDR with global visibility
All other combinations	Target RNIC rejects RDMA Flush sent with any unsupported placement	-Access PMEM or DDR without global visibility which we don't support on Intel platforms (Intel platforms only support Global Ordering)

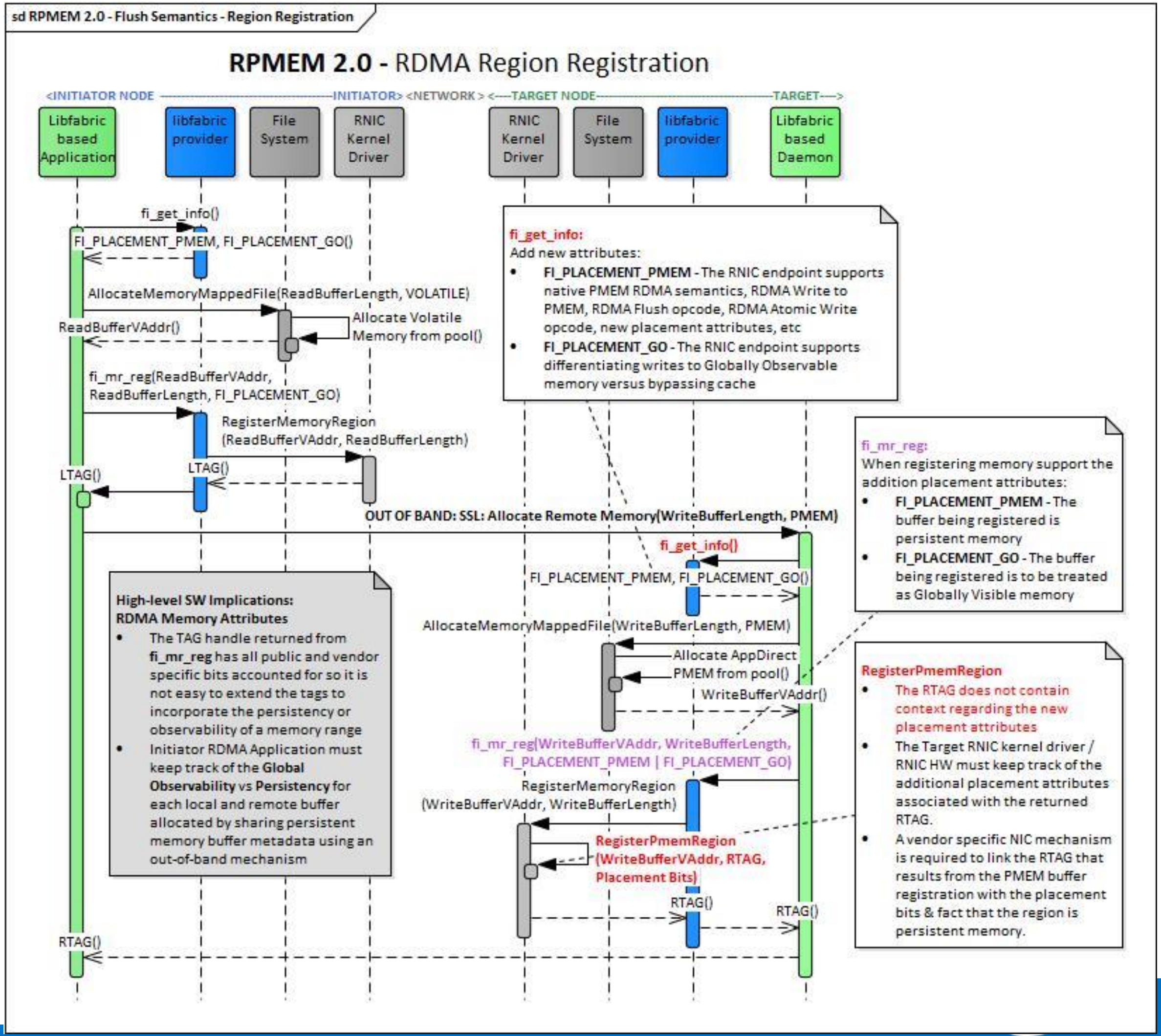
RNIC PCIe Steering Tag based transaction handling

Inbound RDMA Request	Backend RNIC PCIe Request
RDMA Read	PCIe Read
RDMA Write	PCIe Write
RDMA Flush	PCIe Flushing Read w Length=0

RNIC PCIe transaction handling

RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform

Putting it all together


sd RPMEM 2.0 - Flush Semantics - Region Registration
RPMEM 2.0 - RDMA Region Registration
<INITIATOR NODE ----INITIATOR> <NETWORK> <----TARGET NODE----TARGET---->
Libfabric based Application
libfabric provider
File System
RNIC Kernel Driver
RNIC Kernel Driver
File System
libfabric provider
Libfabric based Daemon
fi_get_info()
FI_PLACEMENT_PMEM, FI_PLACEMENT_GO()
AllocateMemoryMappedFile(ReadBufferLength, VOLATILE)
Allocate Volatile Memory from pool()
ReadBufferVAddr()
fi_mr_reg(ReadBufferVAddr, ReadBufferLength, FI_PLACEMENT_GO)
RegisterMemoryRegion (ReadBufferVAddr, ReadBufferLength)
LTAG()
LTAG()
OUT OF BAND: SSL: Allocate Remote Memory(WriteBufferLength, PMEM)
fi_get_info()
FI_PLACEMENT_PMEM, FI_PLACEMENT_GO()
AllocateMemoryMappedFile(WriteBufferLength, PMEM)
Allocate AppDirect PMEM from pool()
WriteBufferVAddr()
fi_mr_reg(WriteBufferVAddr, WriteBufferLength, FI_PLACEMENT_PMEM | FI_PLACEMENT_GO)
RegisterMemoryRegion (WriteBufferVAddr, WriteBufferLength)
RegisterPmemRegion (WriteBufferVAddr, RTAG, Placement Bits)
RTAG()
RTAG()
RTAG()

fi_get_info:
Add new attributes:

- **FI_PLACEMENT_PMEM** - The RNIC endpoint supports native PMEM RDMA semantics, RDMA Write to PMEM, RDMA Flush opcode, RDMA Atomic Write opcode, new placement attributes, etc
- **FI_PLACEMENT_GO** - The RNIC endpoint supports differentiating writes to Globally Observable memory versus bypassing cache

fi_mr_reg:
When registering memory support the addition placement attributes:

- **FI_PLACEMENT_PMEM** - The buffer being registered is persistent memory
- **FI_PLACEMENT_GO** - The buffer being registered is to be treated as Globally Visible memory

High-level SW Implications:
RDMA Memory Attributes

- The TAG handle returned from **fi_mr_reg** has all public and vendor specific bits accounted for so it is not easy to extend the tags to incorporate the persistency or observability of a memory range
- Initiator RDMA Application must keep track of the **Global Observability** vs **Persistency** for each local and remote buffer allocated by sharing persistent memory buffer metadata using an out-of-band mechanism

RegisterPmemRegion

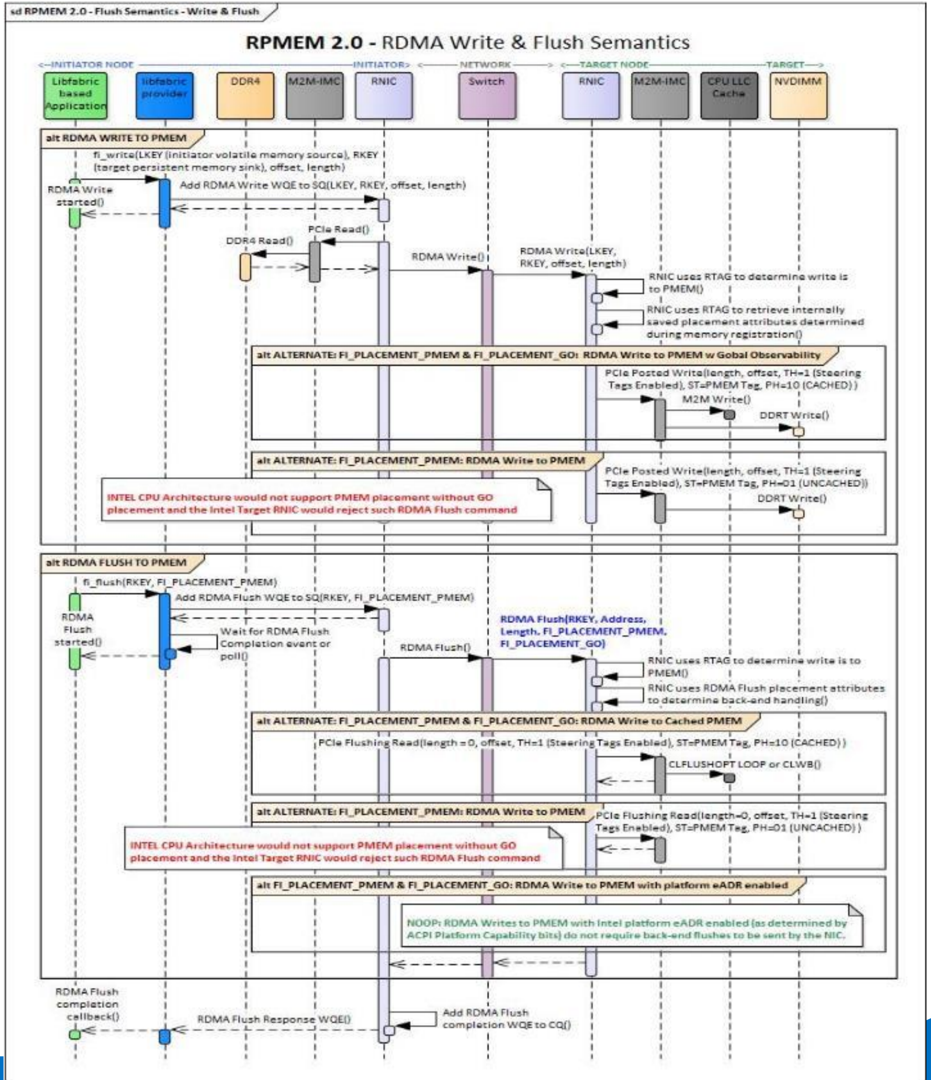
- The RTAG does not contain context regarding the new placement attributes
- The Target RNIC kernel driver / RNIC HW must keep track of the additional placement attributes associated with the returned RTAG.
- A vendor specific NIC mechanism is required to link the RTAG that results from the PMEM buffer registration with the placement bits & fact that the region is persistent memory.

RPMEM 2.0 – Native RDMA with PMEM support for Intel Eagle Stream platform

Putting it all Together:

RDMA Flush:
RKEY,
Starting VA,
Starting Length,
Placement Attribute: GO | PMEM

-VA buffer range specified by the flush must lie within one of the pre-registered target RNIC memory ranges
-The flush domain will cover all writes to the same RKEY wholly contained within the specific buffer range of the flush command
-The Target RNIC will check the RTAG registered memory range against the range specified in the RDMA Write or Flush & errors will be treated as a connection error
-The Target RNIC will send a PCIe Flushing Read with length 0 to force the read after write ordering semantics & flush behavior without the performance penalty of a data transfer
-The Target RNIC chases all the writes that are in range and insures they are flushed to the destination media


sd RPMEM 2.0 - Flush Semantics - Write & Flush
RPMEM 2.0 - RDMA Write & Flush Semantics
INITIATOR NODE
INITIATOR
NETWORK
TARGET NODE
TARGET
Libfabric based Application
libfabric provider
DDR4
M2M-IMC
RNIC
Switch
RNIC
M2M-IMC
CPU LLC Cache
NVDIMM
alt RDMA WRITE TO PMEM
fi_write(LKEY (initiator volatile memory source), RKEY (target persistent memory sink), offset, length)
Add RDMA Write WQE to SQ(LKEY, RKEY, offset, length)
RDMA Write started()
PCIe Read()
DDR4 Read()
RDMA Write()
RDMA Write(LKEY, RKEY, offset, length)
RNIC uses RTAG to determine write is to PMEM()
RNIC uses RTAG to retrieve internally saved placement attributes determined during memory registration()
alt ALTERNATE: FI_PLACEMENT_PMEM & FI_PLACEMENT_GO: RDMA Write to PMEM w Gobal Observability
PCIe Posted Write(length, offset, TH=1 (Steering Tags Enabled), ST=PMEM Tag, PH=10 (CACHED))
M2M Write()
DDRT Write()
alt ALTERNATE: FI_PLACEMENT_PMEM: RDMA Write to PMEM
INTEL CPU Architecture would not support PMEM placement without GO placement and the Intel Target RNIC would reject such RDMA Flush command
PCIe Posted Write(length, offset, TH=1 (Steering Tags Enabled), ST=PMEM Tag, PH=01 (UNCACHED))
DDRT Write()
alt RDMA FLUSH TO PMEM
fi_flush(RKEY, FI_PLACEMENT_PMEM)
Add RDMA Flush WQE to SQ(RKEY, FI_PLACEMENT_PMEM)
RDMA Flush started()
Wait for RDMA Flush Completion event or poll()
RDMA Flush()
RDMA Flush(RKEY, Address, Length, FI_PLACEMENT_PMEM, FI_PLACEMENT_GO)
RNIC uses RTAG to determine write is to PMEM()
RNIC uses RDMA Flush placement attributes to determine back-end handling()
alt ALTERNATE: FI_PLACEMENT_PMEM & FI_PLACEMENT_GO: RDMA Write to Cached PMEM
PCIe Flushing Read(length = 0, offset, TH=1 (Steering Tags Enabled), ST=PMEM Tag, PH=10 (CACHED))
CLFLUSHOPT LOOP or CLWB()
alt ALTERNATE: FI_PLACEMENT_PMEM: RDMA Write to PMEM
INTEL CPU Architecture would not support PMEM placement without GO placement and the Intel Target RNIC would reject such RDMA Flush command
PCIe Flushing Read(length=0, offset, TH=1 (Steering Tags Enabled), ST=PMEM Tag, PH=01 (UNCACHED))
alt FI_PLACEMENT_PMEM & FI_PLACEMENT_GO: RDMA Write to PMEM with platform eADR enabled
NOOP: RDMA Writes to PMEM with Intel platform eADR enabled (as determined by ACPI Platform Capability bits) do not require back-end flushes to be sent by the NIC.
RDMA Flush completion callback()
RDMA Flush Response WQE()
Add RDMA Flush completion WQE to CQ()

RPMEM 2.0 – PMDK Memory Replication Solution - Native RDMA with PMEM support for Intel Eagle Stream platform

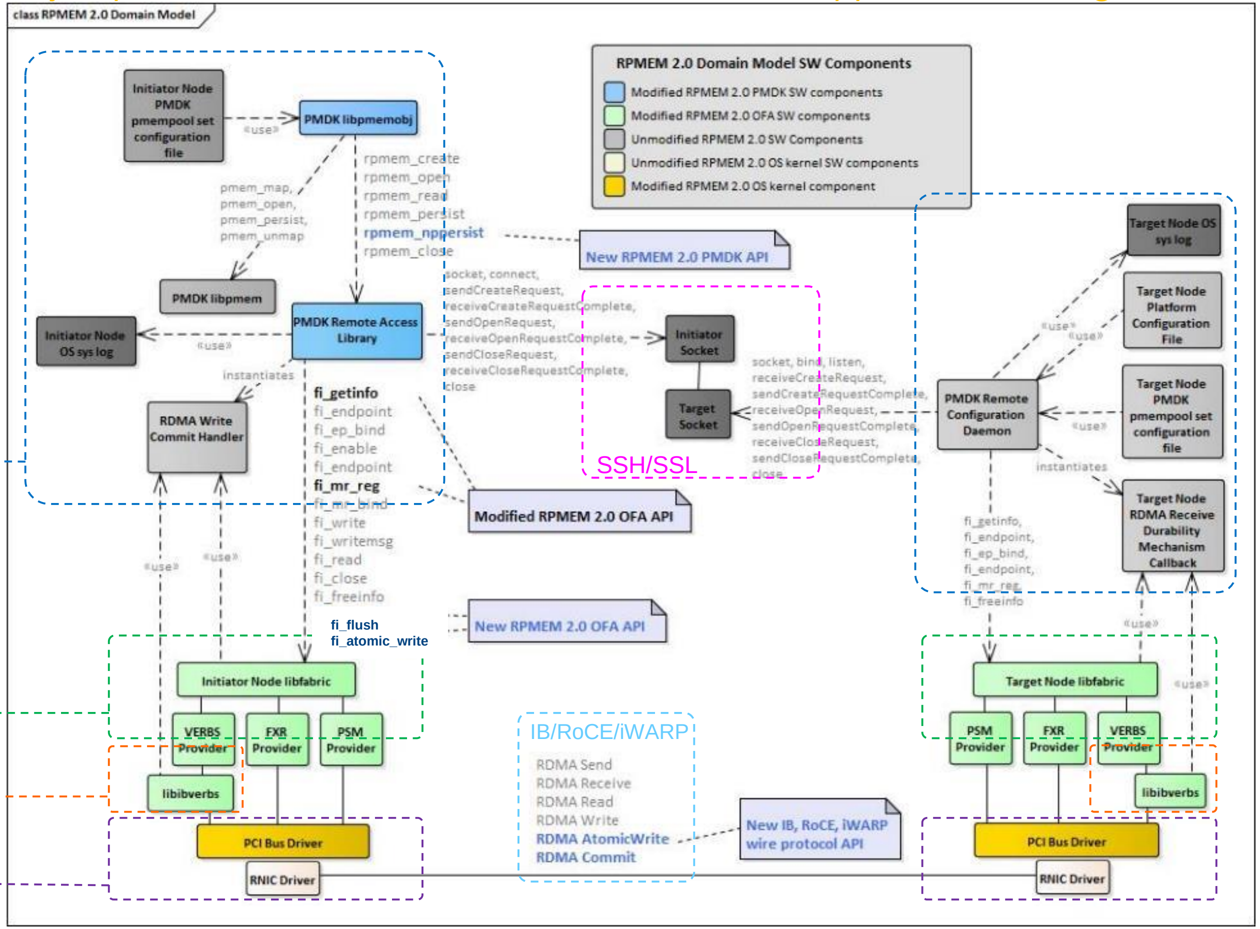
Putting it all together

Intel – PMDK \\pmem.io

OFA - libfabric

OFA - libibverbs

ACPI 6.4 Steering Tag API
Intel HW Design Guide for NIC vendors



class RPMEM 2.0 Domain Model
RPMEM 2.0 Domain Model SW Components
Modified RPMEM 2.0 PMDK SW components
Modified RPMEM 2.0 OFA SW components
Unmodified RPMEM 2.0 SW Components
Unmodified RPMEM 2.0 OS kernel SW components
Modified RPMEM 2.0 OS kernel component
Initiator Node PMDK pmempool set configuration file
«use»
PMDK libpmemobj
pmem_map, pmem_open, pmem_persist, pmem_unmap
rpmem_create
rpmem_open
rpmem_read
rpmem_persist
rpmem_nppersist
rpmem_close
New RPMEM 2.0 PMDK API
PMDK libpmem
Initiator Node OS sys log
«use»
PMDK Remote Access Library
instantiates
RDMA Write Commit Handler
socket, connect, sendCreateRequest, receiveCreateRequestComplete, sendOpenRequest, receiveOpenRequestComplete, sendCloseRequest, receiveCloseRequestComplete, close
Initiator Socket
Target Socket
SSH/SSL
socket, bind, listen, receiveCreateRequest, sendCreateRequestComplete, receiveOpenRequest, sendOpenRequestComplete, receiveCloseRequest, sendCloseRequestComplete, close
fi_getinfo
fi_endpoint
fi_ep_bind
fi_enable
fi_endpoint
fi_mr_reg
~~fi_mr_bind~~
fi_write
fi_writemsg
fi_read
fi_close
fi_freeinfo
Modified RPMEM 2.0 OFA API
fi_flush
fi_atomic_write
New RPMEM 2.0 OFA API
«use»
«use»
Initiator Node libfabric
VERBS Provider
FXR Provider
PSM Provider
libibverbs
PCI Bus Driver
RNIC Driver
IB/RoCE/iWARP
RDMA Send
RDMA Receive
RDMA Read
RDMA Write
RDMA AtomicWrite
RDMA Commit
New IB, RoCE, iWARP wire protocol API
Target Node OS sys log
Target Node Platform Configuration File
«use»
«use»
PMDK Remote Configuration Daemon
Target Node PMDK pmempool set configuration file
«use»
instantiates
Target Node RDMA Receive Durability Mechanism Callback
fi_getinfo, fi_endpoint, fi_ep_bind, fi_endpoint, fi_mr_reg, fi_freeinfo
«use»
«use»
Target Node libfabric
PSM Provider
FXR Provider
VERBS Provider
libibverbs
PCI Bus Driver
RNIC Driver

RPMEM 2.0 SOFTWARE ARCHITECTURE

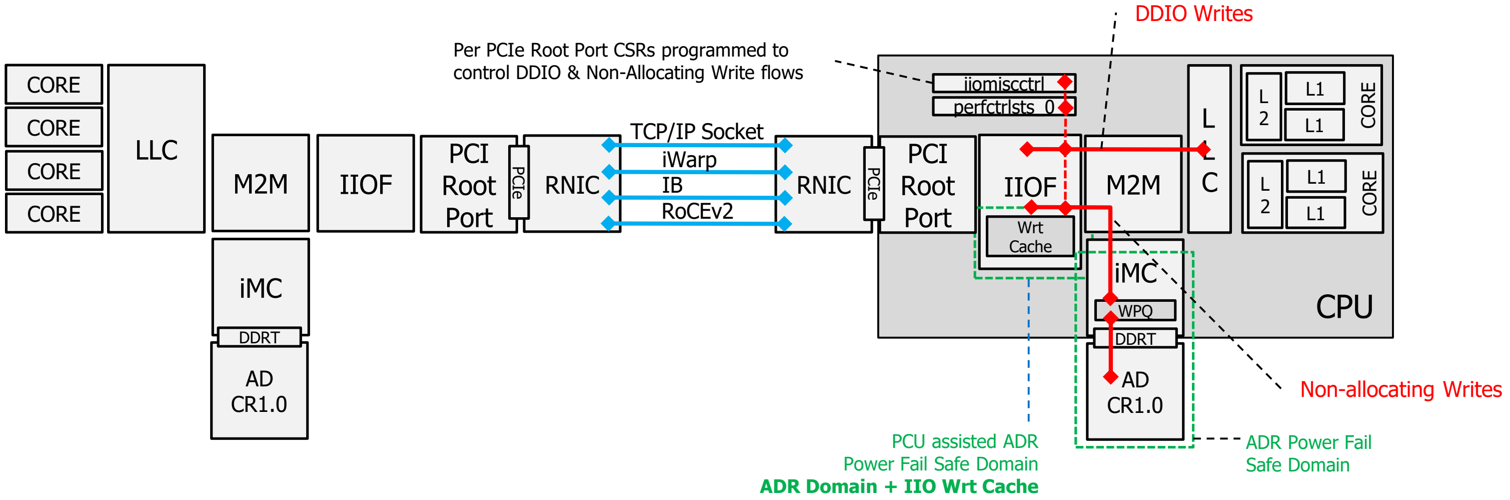
Look Inside.™

RPMEM 1.0 Backup Material

SW mechanisms to utilize PMEM on Intel Purley platforms

DCG
Data Center Group

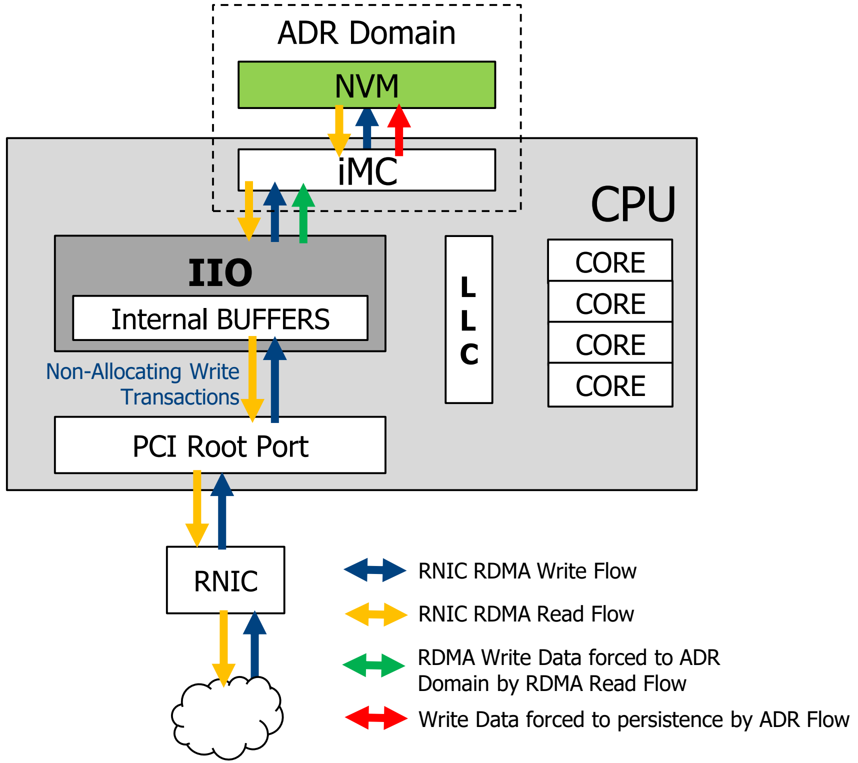
RPMEM 1.0 – Basic RDMA with PMEM for Intel Purley platform



RPMEM 1.0 Intel Purley Platform HW Architecture

RPMEM 1.0 – Basic RDMA with PMEM for Purley and CR1.0/1.5

- **ADR Domain, No DDIO**
- **"Appliance Method"**
  - Enable "non-allocating Write" transactions per Root PCI Port OR use PCIe "no snoop" bit in the PCI Transaction Header for a specific RNIC
    - Requires BIOS Enabling
    - Forces RDMA Write data directly to iMC
    - Enable on PCI Root Port with RNIC or specific RNIC
  - Follow RDMA Write(s) with RDMA Read to force remaining IIO buffer write data to ADR Domain
    - RDMA Read acts as a fencing function for the previous non-allocating write data and forces remaining write data out of the pipeline
    - Must force a PCI Read on the local PCI Root Port that handled the writes
    - Since RDMA Write and Read are silent, there is little or no change to the SW on the node supplying the Sink buffers for RDMA Write
    - RDMA Read – Read can be for any address, length



ADR Domain
NVM
iMC
CPU
IIO
Internal BUFFERS
Non-Allocating Write Transactions
L L C
CORE
CORE
CORE
CORE
PCI Root Port
RNIC
RNIC RDMA Write Flow
RNIC RDMA Read Flow
RDMA Write Data forced to ADR Domain by RDMA Read Flow
Write Data forced to persistence by ADR Flow

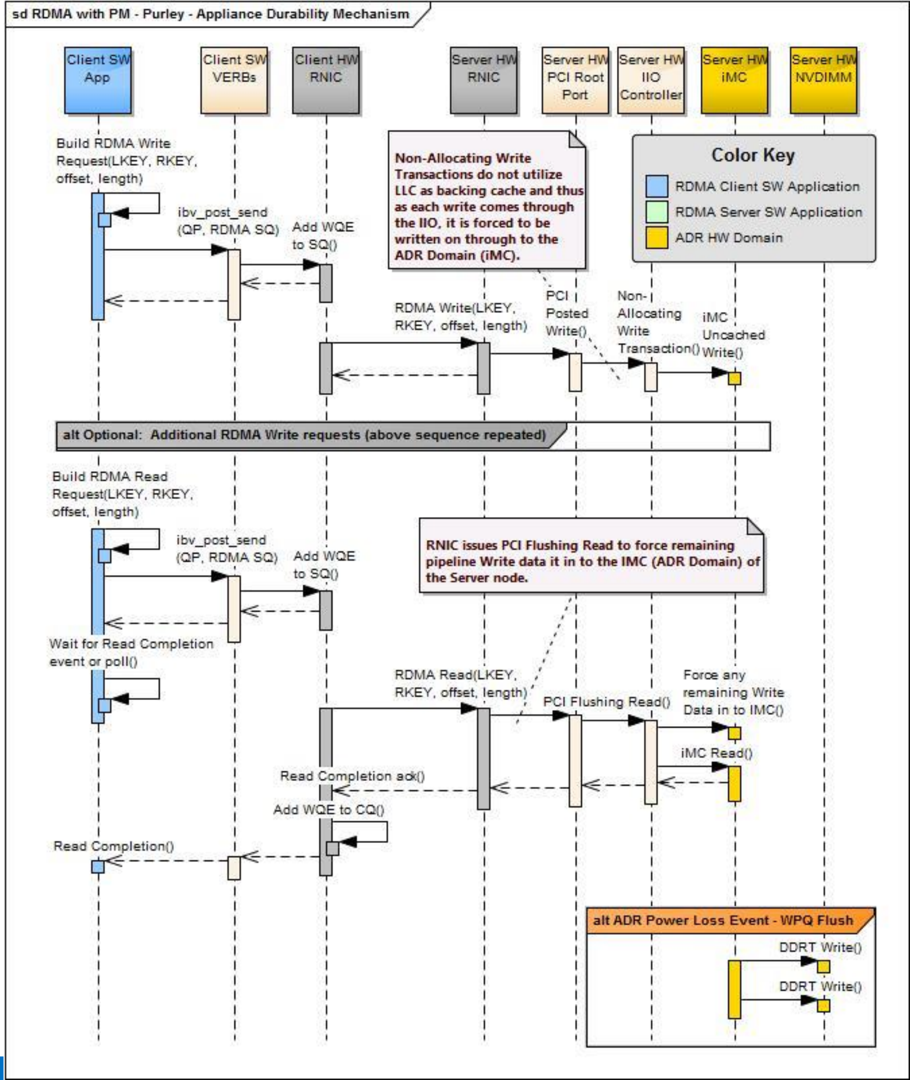
RPMEM 1.0 – Basic RDMA with PMEM for Purley and CR1.0/1.5

- **ADR Domain, No DDIO**
- **“Appliance Method”**
- **See Sequence Diagram at Right for example flow**

RDMA WRITES W NON ALLOCATING WRITE TRANSACTIONS

NEW ADDITION: RDMA READ TO FORCE REMAINING WRITE DATA IN TO THE ADR DOMAIN

NEW ADDITION: RELY ON ADR LOGIC TO MAKE WRITE DATA IN THE ADR DOMAIN DURABLE ON POWER FAIL

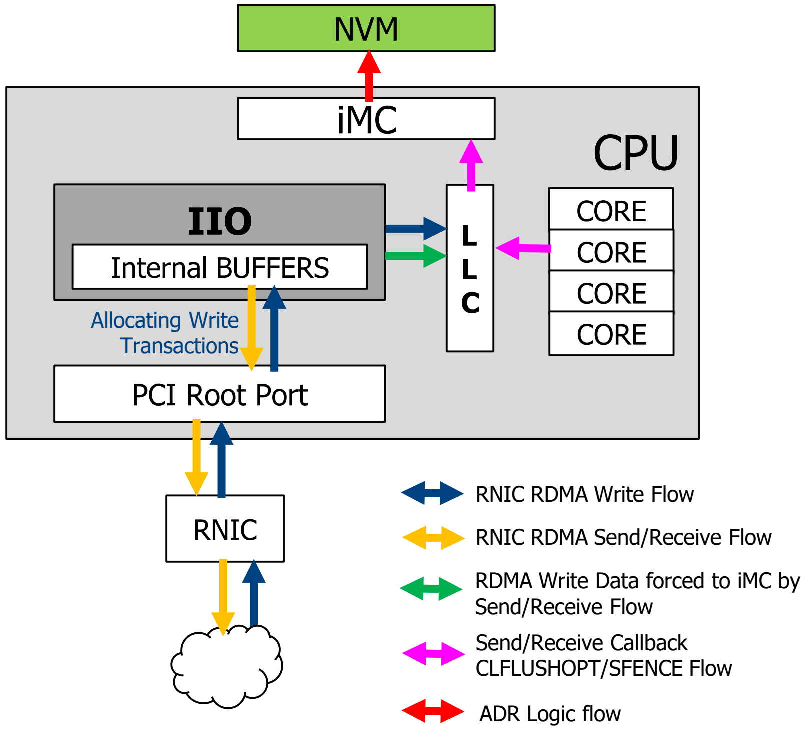

sd RDMA with PM - Purley - Appliance Durability Mechanism
Client SW App
Client SW VERBs
Client HW RNIC
Server HW RNIC
Server HW PCI Root Port
Server HW IIO Controller
Server HW iMC
Server HW NVDIMM
Color Key
RDMA Client SW Application
RDMA Server SW Application
ADR HW Domain
Build RDMA Write Request(LKEY, RKEY, offset, length)
ibv_post_send (QP, RDMA SQ)
Add WQE to SQ()
RDMA Write(LKEY, RKEY, offset, length)
PCI Posted Write()
Non-Allocating Write Transaction()
iMC Uncached Write()
alt Optional: Additional RDMA Write requests (above sequence repeated)
Build RDMA Read Request(LKEY, RKEY, offset, length)
ibv_post_send (QP, RDMA SQ)
Add WQE to SQ()
Wait for Read Completion event or poll()
RDMA Read(LKEY, RKEY, offset, length)
PCI Flushing Read()
Force any remaining Write Data in to IMC()
iMC Read()
Read Completion ack()
Add WQE to CQ()
Read Completion()
alt ADR Power Loss Event - WPQ Flush
DDRT Write()
DDRT Write()

Non-Allocating Write Transactions do not utilize LLC as backing cache and thus as each write comes through the IIO, it is forced to be written on through to the ADR Domain (iMC).

RNIC issues PCI Flushing Read to force remaining pipeline Write data it in to the IMC (ADR Domain) of the Server node.

RPMEM 1.0 – Basic RDMA with PMEM for Purley and CR1.0/1.5

- **No ADR Domain, With DDIO (Platform Default configuration)**
- **“General Purpose Server Method”**
  - Use standard “allocating Write” transactions for Root PCI Port to IIO
  - Follow RDMA Write(s) with Send/Receive to get server RDMA SW callback
    - Send/Receive will contain list of cache lines that were written (or write addresses/lengths)
    - ISA – CLFLUSHOPT/SFENCE – Flush CPU cache lines and wait for flush to complete (invalidates cache contents). The list of cache lines from the Send message is used to identify the cache lines that need to be flushed.
    - Rely on ADR logic to make write data in the ADR Domain durable on a power loss
    - Internal IIO buffers will be flushed as part of ADR logic allowing “allocating writes” to be used.



NVM
iMC
CPU
IIO
Internal BUFFERS
L L C
CORE
CORE
CORE
CORE
Allocating Write Transactions
PCI Root Port
RNIC
RNIC RDMA Write Flow
RNIC RDMA Send/Receive Flow
RDMA Write Data forced to iMC by Send/Receive Flow
Send/Receive Callback CLFLUSHOPT/SFENCE Flow
ADR Logic flow

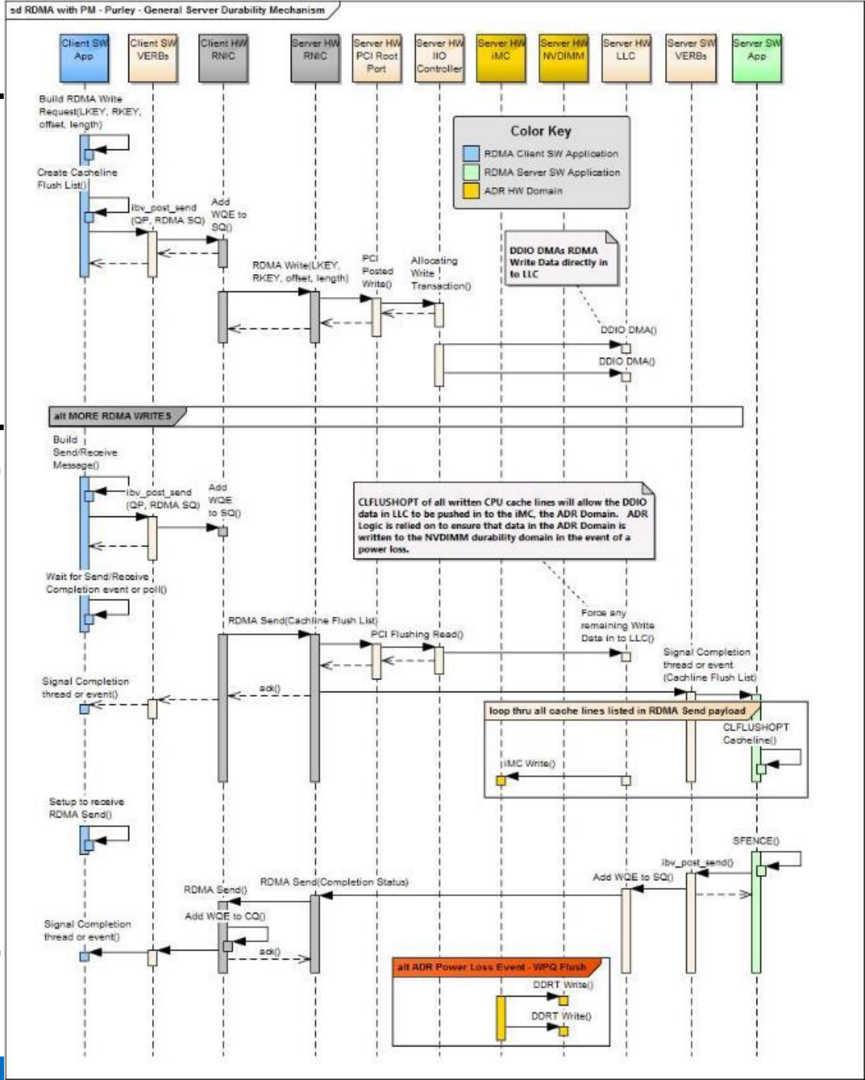
RPMEM 1.0 – Basic RDMA with PMEM for Purley and CR1.0/1.5

- **No ADR Domain, With DDIO**
- **“General Purpose Server Method”**
- **See Sequence Diagram at Right for example flow**

RDMA WRITES W STANDARD ALLOCATING WRITE TRANSACTIONS

NEW ADDITION: RDMA SEND/RECEIVE CALLBACK TO FORCE DATA IN TO IMC ADR DOMAIN

NEW ADDITION: RELY ON ADR LOGIC TO MAKE WRITE DATA IN THE ADR DOMAIN DURABLE ON POWER FAIL


sd RDMA with PM - Purley - General Server Durability Mechanism
Client SW App
Client SW VERBs
Client HW RNIC
Server HW RNIC
Server HW PCI Root Port
Server HW IIO Controller
Server HW iMC
Server HW NVDIMM
Server HW LLC
Server SW VERBs
Server SW App
Color Key
RDMA Client SW Application
RDMA Server SW Application
ADR HW Domain
Build RDMA Write Request(LKEY, RKEY, offset, length)
Create Cacheline Flush List()
ibv_post_send (QP, RDMA SQ)
Add WQE to SQ()
RDMA Write(LKEY, RKEY, offset, length)
PCI Posted Write()
Allocating Write Transaction()
DDIO DMA()
DDIO DMA()
alt MORE RDMA WRITES
Build Send/Receive Message()
ibv_post_send (QP, RDMA SQ)
Add WQE to SQ()
Wait for Send/Receive Completion event or poll()
RDMA Send(Cachline Flush List)
PCI Flushing Read()
Force any remaining Write Data in to LLC()
Signal Completion thread or event (Cachline Flush List)
Signal Completion thread or event()
ack()
loop thru all cache lines listed in RDMA Send payload
CLFLUSHOPT Cacheline()
iMC Write()
Setup to receive RDMA Send()
SFENCE()
ibv_post_send()
Add WQE to SQ()
RDMA Send()
RDMA Send(Completion Status)
Add WQE to CQ()
Signal Completion thread or event()
ack()
alt ADR Power Loss Event - WPQ Flush
DDRT Write()
DDRT Write()

DDIO DMAs RDMA Write Data directly in to LLC

CLFLUSHOPT of all written CPU cache lines will allow the DDIO data in LLC to be pushed in to the iMC, the ADR Domain. ADR Logic is relied on to ensure that data in the ADR Domain is written to the NVDIMM durability domain in the event of a power loss.