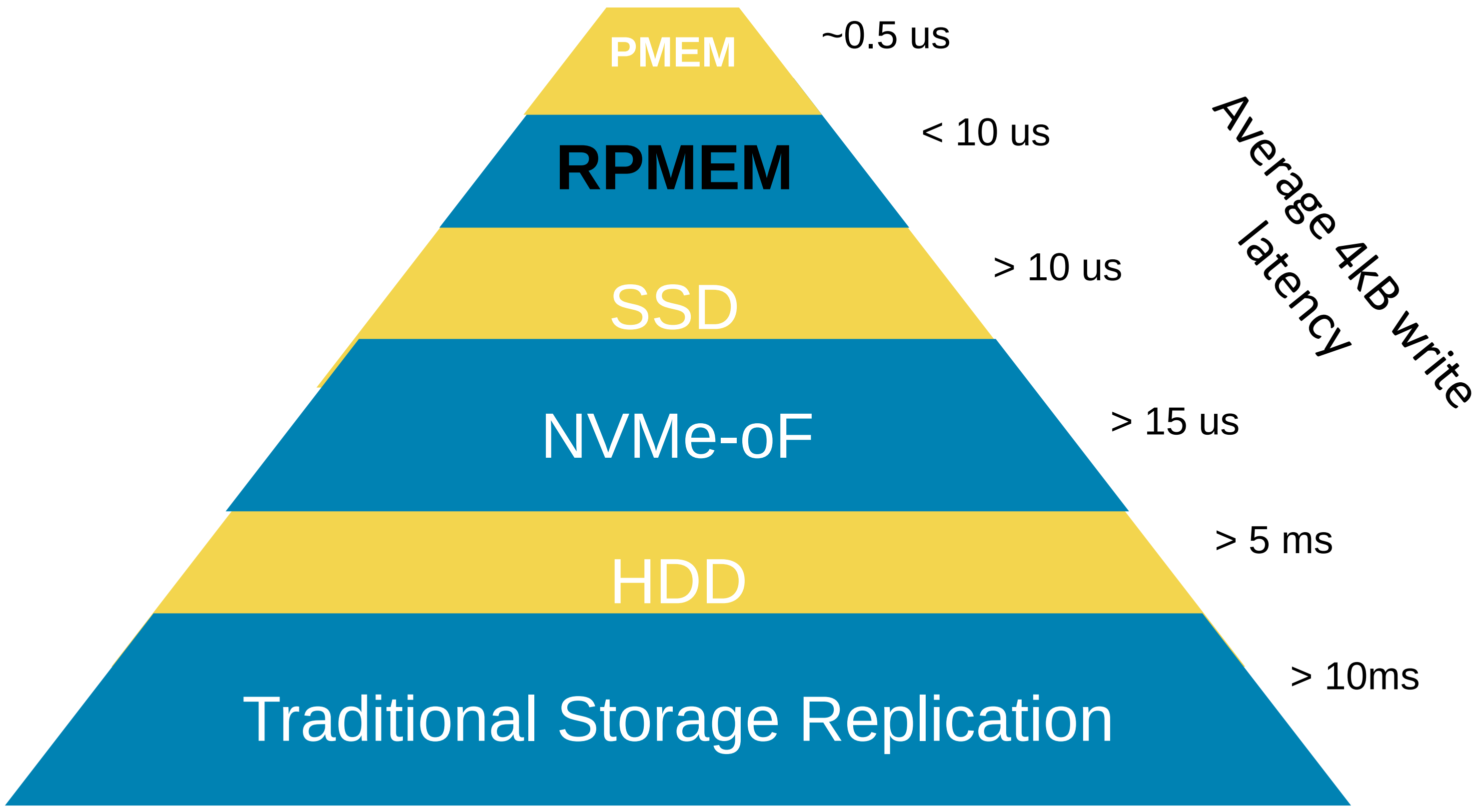
REMOTE PERSISTENT MEMORY ACCESS API - THE SECOND APPROACH

Tomasz Gromadzki, Software Architect
Jan M. Michalski, Software Engineer
Intel Corporation

[June 9, 2020]

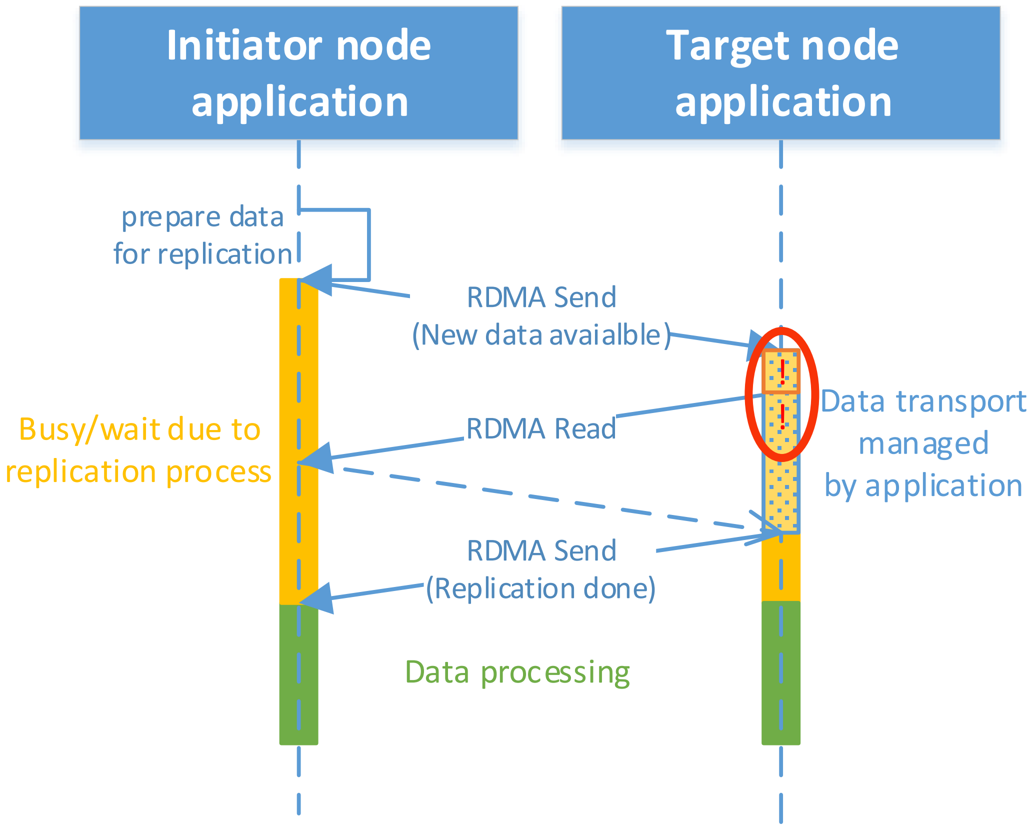
REMOTE PERSISTENT MEMORY

Remote *Direct* Persistent Memory Access

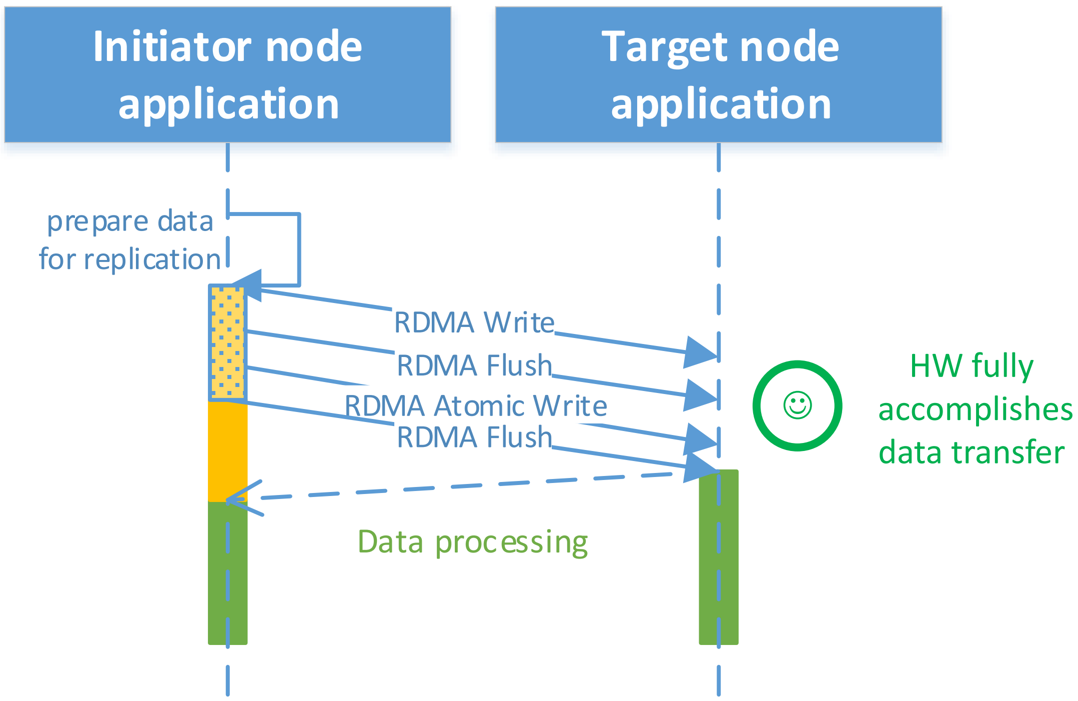


PUSH REPLICATION METHOD ALLOWS MORE DATA TO BE TRANSFERRED

Pull method - traditional RDMA



Push method - RPMEM



librpmem BASED REPLICATION

The first approach

- **Read and write access to remote persistent memory**
- **Software solution for 8 bytes atomicity guarantee**
  - The remote node's rpmem daemon
- **Read after write or send after write method selected based on remote platform configuration**
- **Designed for synchronous replication for libpmemobj**

int **rpmem_persist**(RPMEMpool *rpp, size_t offset, size_t length, unsigned lane, unsigned flags);

int **rpmem_read**(RPMEMpool *rpp, void *buff, size_t offset, size_t length, unsigned lane);

librpmem BASED REPLICA MANAGEMENT

The first approach

- **Configuration based on persistent memory pool description files**
- **SSH used for out-of-band connection**
- **rpmemd daemon controls a remote node's pool set**
- **Deeply integrated with libpmemobj**

RPMEMpool ***rpmem_create**(const char *target, const char *pool_set_name, void *pool_addr, size_t pool_size, unsigned *nlanes, const struct rpmem_pool_attr *create_attr);

RPMEMpool ***rpmem_open**(const char *target, const char *pool_set_name, void *pool_addr, size_t pool_size, unsigned *nlanes, struct rpmem_pool_attr *open_attr);

```
PMEMPOOLSET
100G /mountpoint0/myfile.part0
200G /mountpoint1/myfile.part1

# remote replica

REPLICA pmem@10.123.11.7 remotepool.set
```

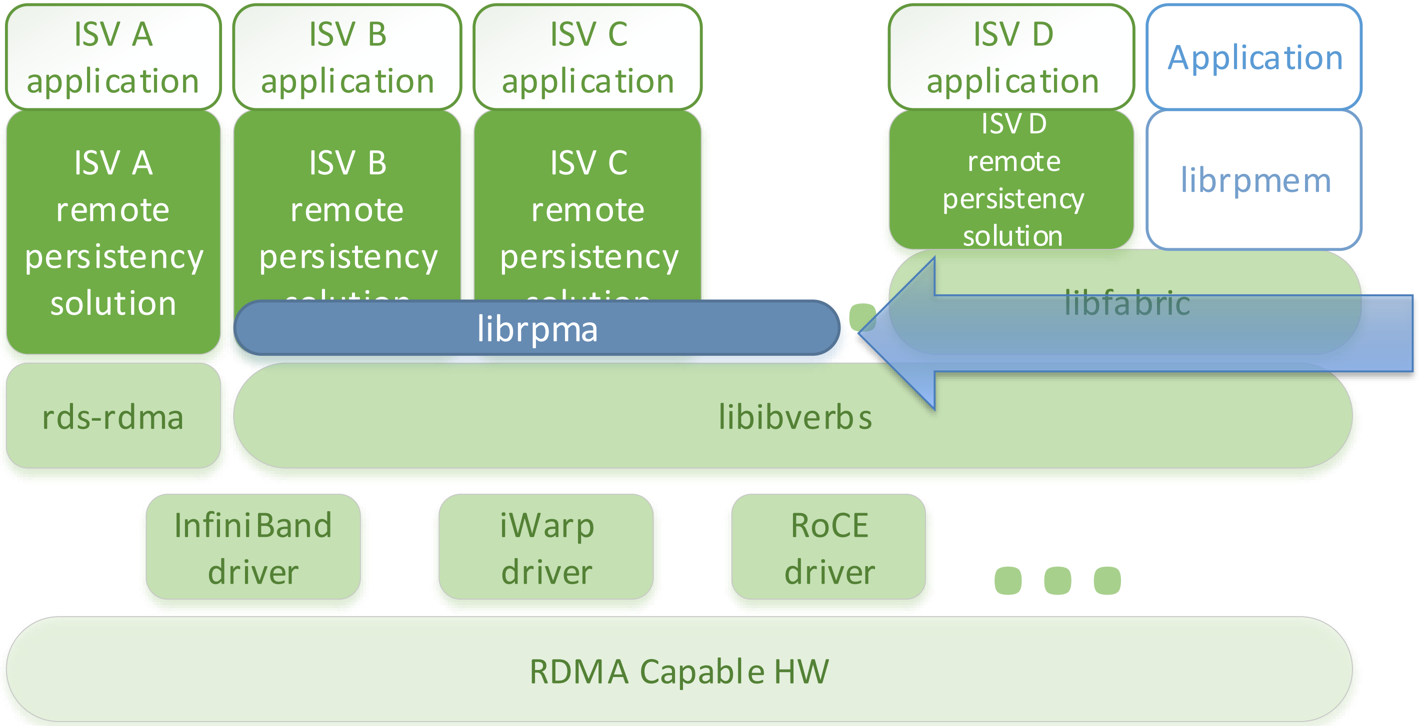
PAIN POINTS OF THE FIRST APPROACH

Customer's feedback to librpmem

PMDK (librpmem) provides	Customers expect
Replication process **tightly coupled to libpmemobj**	Replication process **controlled by app**
Poolsets semantic is used as replication basis	Replication process to **follow app data semantic**
Static replication **configuration**	Replication **configuration** might **change online** based on application needs
No access to replicated data in runtime	At least **read access** to replicated data in runtime
Focus on RDMA.Write API	RDMA.Write/Send as well as RDMA.Read support depending on application case
	Neither libfabric nor SSH dependencies

librpma THE SECOND APPROACH TO RPMEM

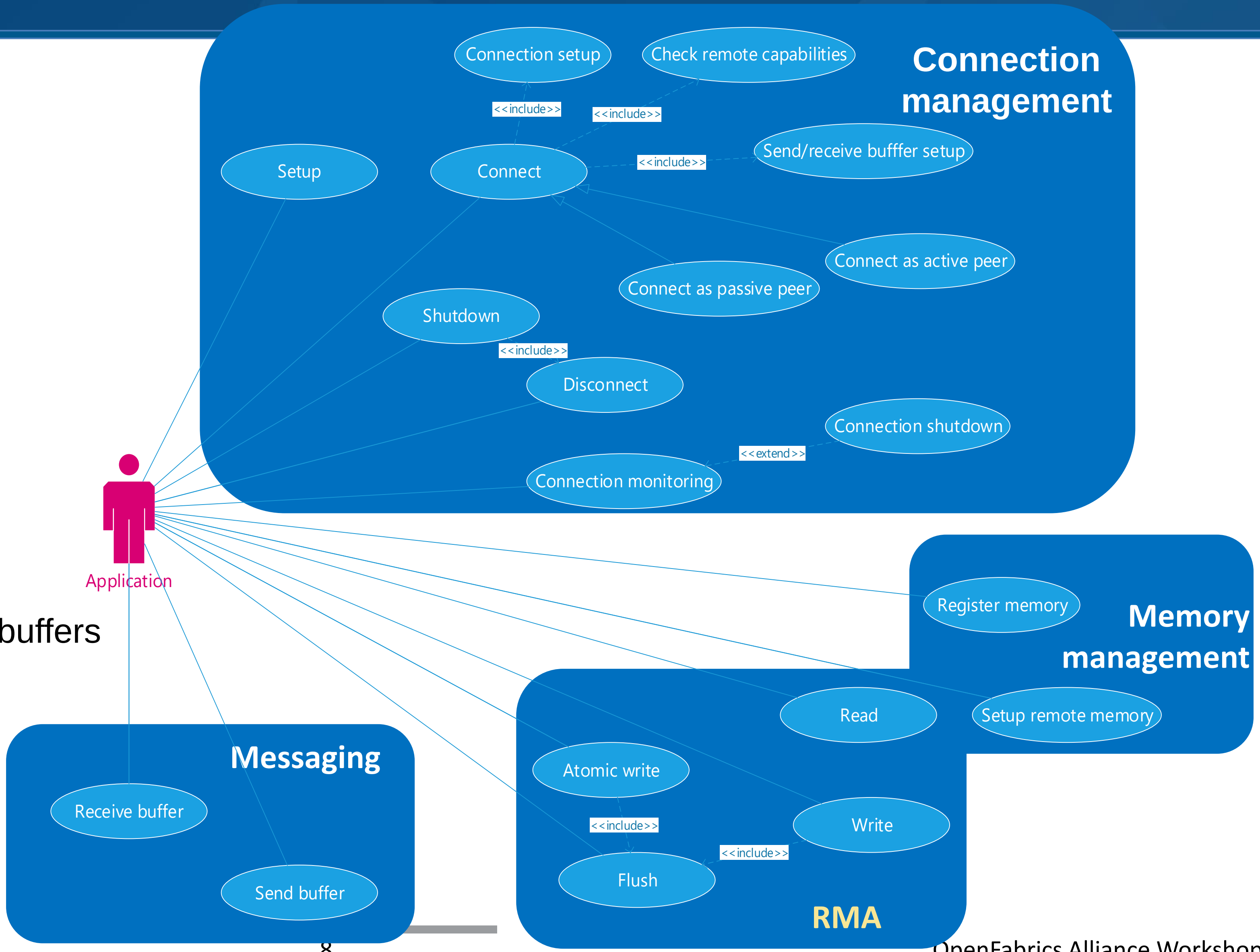
RDMA Push Transfer Method



- **memcpy-like API for RPMEM**
- **Hidden RDMA complexity**
- **Based on librpmem experience**
  - read-after-write, read-after-send
- **RDMA Memory Placement Extensions ready**
  - Flush, Atomic Write, Verify
- **PMEM management left for an application**

librpma USE CASE MODEL

- **RMA**
  - This is the core of the library
- **Connection management**
  - to ensure operations consistency
  - to hide RDMA complexity
- **Messaging**
  - also with PMEM-backed message buffers
- **Memory management**
  - e.g. to support r_key exchange



REMOTE MEMORY ACCESS

librpma API

rpma_write (dst, dst_offset, src, src_offset, len, completion)

- completion – do we expect confirmation of the request

rpma_write_8bytes(dst, dst_offset, src, src_offset, completion)

rpma_flush(dst, offset, len, placement)

- placement = either Persistent or only Global Observability

rpma_read(dst, dst_offset, src, src_offset, len, completion)

rpma_next_completion(&operation, &status)

- allows collecting confirmations to write/flush/read operations

Non-blocking API

int rpma_conn_fd(rpma_conn, fd_type)

MEMORY MANAGEMENT

librpma API

memory_local_handle **= rpma_memory_new(void *ptr, size, usage, placement, flags)**

- usage - bitwise or: read_src, read_dst, write_src, write_dst
- placement either persistent or volatile
- flags – e.g. cached/no cached write

rpma_memory_serialize(memory_local_handle, user_buffer)

- user_buffer allows delivering the local memory description to the remote side

rpma_memory_deserialize(user_buffer, memory_remote_handle)

- a remote memory handle is created from the user_buffer on the remote side

CONNECTION SETUP

librpma API

- **Active side**

rpma_conn_setup(addr, service, connection)**

/ receive buffers setup */*

rpma_connect(connection)

rpma_conn_get_remote_capabilities(…)

- **Listening side**

rpma_listen(addr, service, rpma_socket)**

rpma_socket_read (rpma_socket, connection, connection_status)

/ receive buffers setup */*

rpma_accept(rpma_conn)
or **rpma_reject(rpma_conn)**

finally
rpma_socket_delete (rpma_socket);

CONNECTION MANAGEMENT AND CONFIGURATION

librpma API

- **Connection monitoring and shutdown**

  rpma_conn_status(connection_status)

  rpma_disconnect(connection)

- **Blocking/non-blocking API**

  rpma_peer_cfg_set_blocking(blocking_API_calls)

  int rpma_socket_fd(rpma_socket)

  int rpma_conn_fd(rpma_conn, fd_type)

  - either conn_status fd or conn_next_completion fd
  - file descriptors will allow making use of generally available scalable I/O event notification mechanisms

- **Capabilities setup**

  rpma_peer_cfg_set_auto_flush(…)

  rpma_peer_cfg_set_ddio(…)

  rpma_peer_cfg_set_odp(…)

MESSAGING

librpma API

rpma_conn_recv_setup(connection, memory_local_handle, offset, entries_num, entry_size)

- receive buffers setup

rpma_conn_recv_payload(connection, memory_local_handle, offset, size)

- access to received data

rpma_conn_recv_ack(memory_local_handle, offset)

- mark memory buffer to be reused for next incoming message

rpma_conn_send(memory_local_handle, offset, size, completion)

- post a send request to the remote side

rpma_next_completion(&operation, &status)

EXAMPLES

RPMA WRITE IN BLOCKING MODE

Write to the remote persistent memory followed by Flush

```
memory_local_handle *src;
memory_remote_handle *dst;
…
/* local write to memory described by src */

/* posting a WRITE */
rpma_write(conn, user_context, dst, offset_dst, src, offset_src, len, RPMA_OP_FLAG_NO_COMPLETION);

/* post a FLUSH for flushing the preceding WRITE to persistence */
rpma_flush(conn, user_context, dst, offset_dst, len, RPMA_FLUSH_TYPE_PERSISTENT, RPMA_OP_FLAG_COMPLETION);

/* wait for the FLUSH to complete */
rpma_next_completion(conn, &op_context, &op, &status);
assert(op == RPMA_OP_FLUSH && status == RPMA_OP_STATUS_OK && op_context == user_context);

…
```

TARGET NODE MEMORY MANAGEMENT

Memory Setup and Serialization

```
…
void *pmem_ptr;
char payload[256];
size_t payload_size;
memory_local_handle *dst;

…
pmem_ptr =  pmem2_map_get_address(map);
…
rpma_memory_new(peer, pmem_ptr, pmem_size, RPMA_MR_WRITE_DST, RPMA_MR_PLT_PERSISTENT, &dst);
…
rpma_memory_serialize(dst, payload);
/* send data to initiator node to let know memory registration in remote location */

/* target node ready for incoming remote operations - READ/WRITE/FLUSH */

…
```

INITIATOR NODE MEMORY MANAGEMENT

Memory Setup, Memory Deserialization

```
char mem_buff[] = "Test";
memory_local_handle *src;
memory_remote_handle *dst;
…
/* register mem to be copied to remote node */
rpma_memory_new(peer, mem_buff, len, RPMA_MR_WRITE_SRC, RPMA_MR_PLT_VOLATILE, &src);
…


/* create remote memory handle based on data received from target node */
rpma_memory_deserialize(payload, payload_size, &dst);
…

/* initiator and target nodes ready for remote operations - READ/WRITE/FLUSH */
```

CONNECTION SETUP AND CONNECTING TO THE REMOTE NODE

```
…
    char recv_buff[CLIENT_BUFF_SIZE];
    memory_local_handle *recv;
…

    /* initialize connection */
    rpma_conn_setup(peer, SERVER_ADDR, SERVER_PORT, &conn);
…
    /* receive buffers setup */
    rpma_memory_new(peer, recv_buff, CLIENT_BUFF_SIZE, RPMA_MR_WRITE_DST, RPMA_MR_PLT_VOLATILE, &recv);
    rpma_conn_recv_setup(conn, recv, 0 /* offset */, 1, CLIENT_BUFF_SIZE);

    /* establish the connection */
    struct rpma_conn_cfg *conn_cfg;
    rpma_conn_cfg_new(&conn_cfg);
    rpma_conn_cfg_set_sq_size(conn_cfg, 10);
    rpma_conn_cfg_set_rq_size(conn_cfg, 10);
    rpma_conn_cfg_set_cq_size(conn_cfg, 10);
    rpma_connect(conn, conn_cfg);
    rpma_conn_cfg_delete(&conn_cfg);
 …
```

THANK YOU

Tomasz Gromadzki, Software Architect
Jan M. Michalski, Software Engineer

Intel Corporation

https://github.com/pmem/rpma

LEGAL NOTICE AND DISCLAIMERS

BACKUP

NEW API INSPIRATIONS

New RDMA verbs (IETF/IBTA)

- **The *RDMA Flush* operation requests that all bytes in a specified region are to be made *persistent* and/or *globally visible***

- **The *RDMA Verify** operation requests that all bytes in a specified region are to be read from the underlying storage and that an integrity hash be calculated.**

- **The *Atomic Write* operation provides a block of data (*8 bytes*) which is placed to a specified region *atomically***

*) defined only by IETF so far

Newly identified workloads

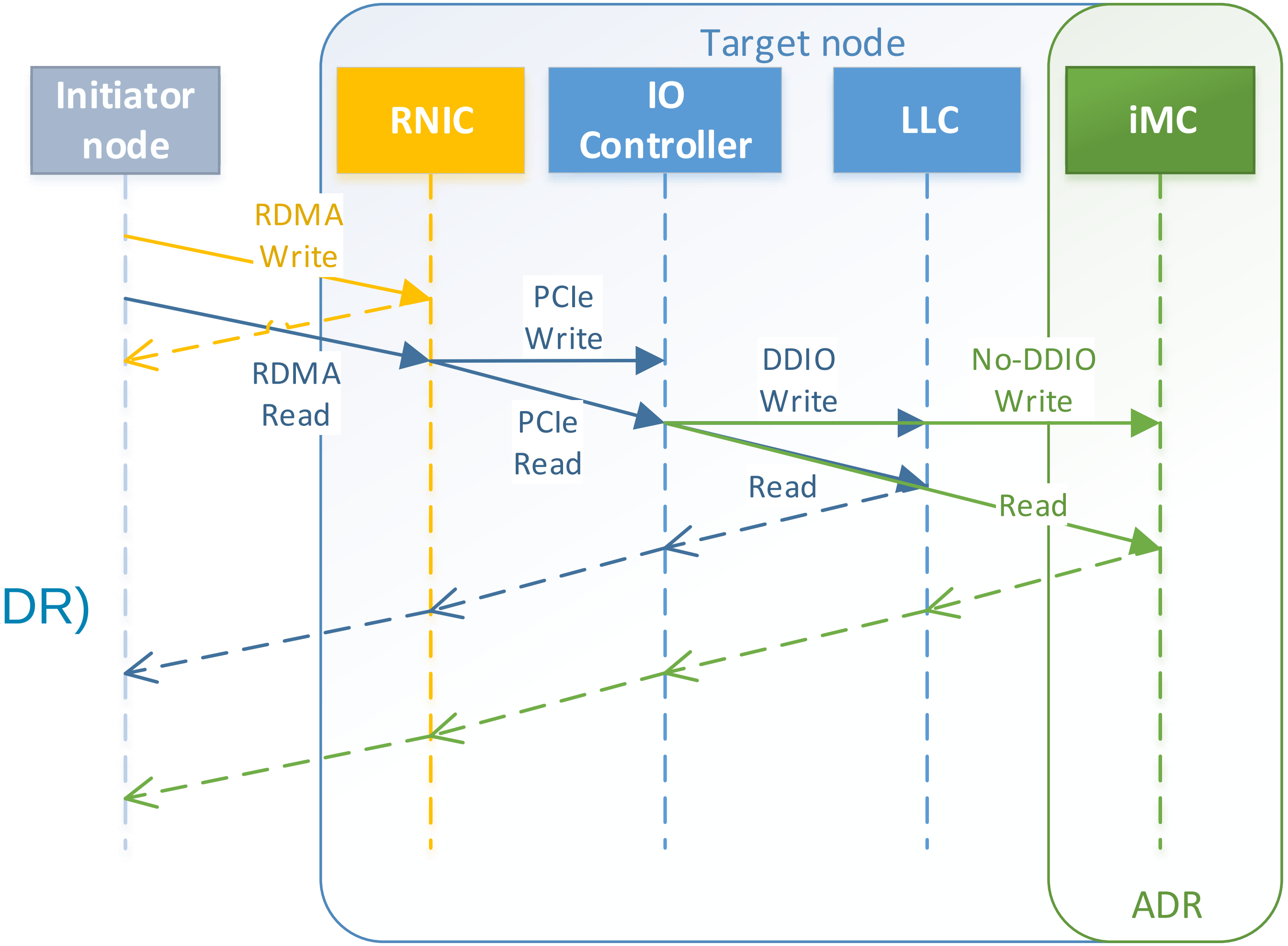
- **PMEM used in the context of post SEND and post RECV**

- **Connection’s private data utilize for**
  - **- nodes’ capabilities exchange**
  - **- r_key exchange**

- **scatter/gather list to combine an application payload and library’s private data in one network transaction**

PUSH METHOD OVER TRADITIONAL RDMA

- RDMA Write ensures only that data are delivered to RNIC (no ADR)
- RDMA Read* forces data to be pushed out form RNIC with PCIe Writes
- PCIe Read flushes all PCIe Writes to destination LLC in case of DDIO** (no to ADR)
- DDIO off ensures data are moved to persistent memory automatically



*) 8 bytes RDMA/PCIe Read is used for that purpose
**) Intel® Data Direct I/O Technology