2021 OFA Virtual Workshop

REMOTE PERSISTENT MEMORY ACCESS AS SIMPLE AS LOCAL MEMORY ACCESS

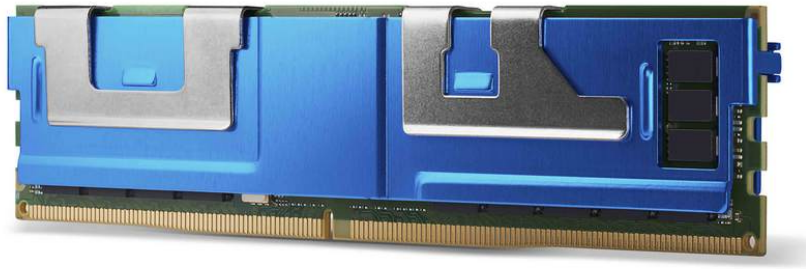
Tomasz Gromadzki, Software Architect (RPMem)

Intel Corporation

intel®

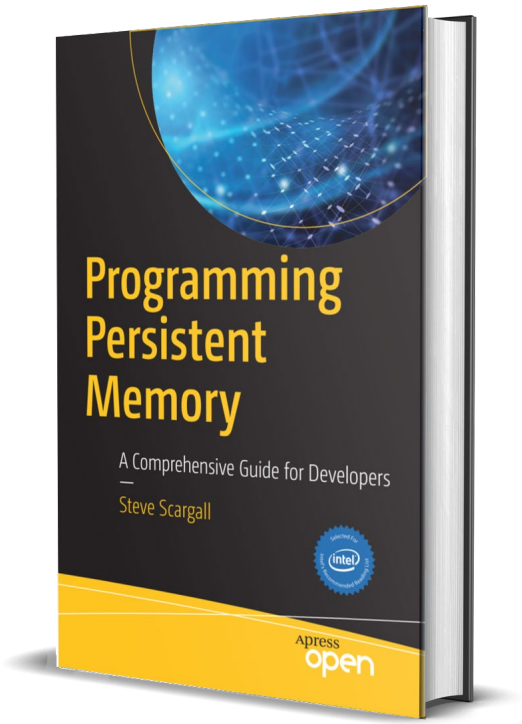
WHAT IS PERSISTENT MEMORY (PMEM)?

- allow accessing data as memory
  - directly
  - byte-addressable
- the contents are non-volatile (preserved across power cycles).
- it doesn't typically replace either memory or storage



- For more information please visit:
  - pmem.io
  - Persistent Memory Development Kit (PMDK) GitHub repo

pmem.io/book



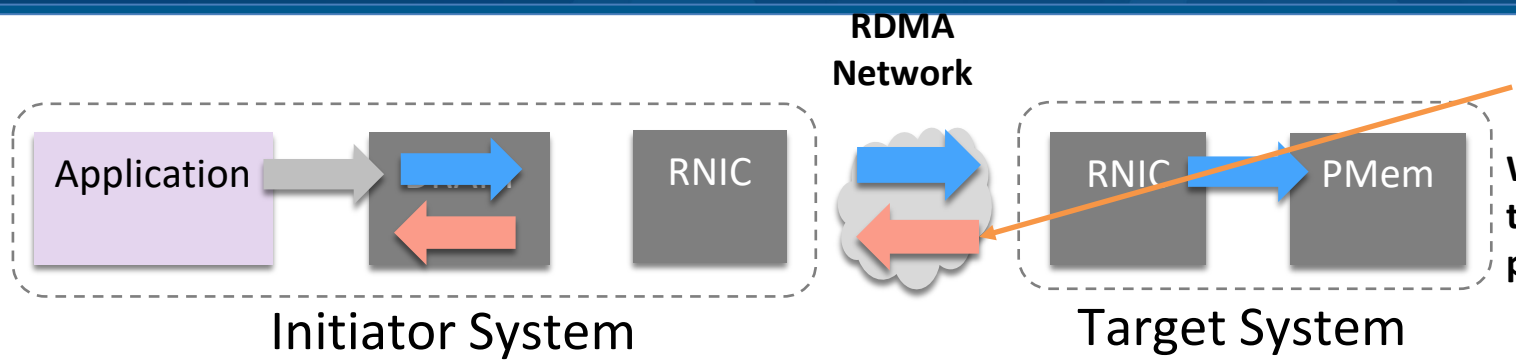
PERSISTENT MEMORY PROGRAMMING MODEL

Storage Networking Industry Association (SNIA) standardized
Software Persistent Memory (PMem) programming model

When writing data to PMem, in order for the data to be considered persistent:

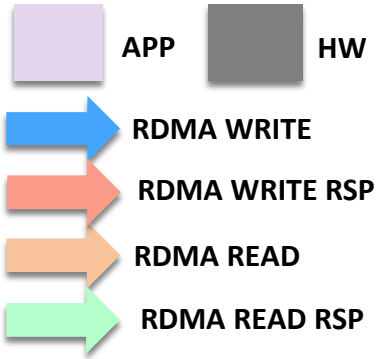
- After writing to persistent memory, the software is responsible for
  flushing data from CPU caches to the Persistence Domain
- Once the CPU caches are flushed of write data,
  the platform guarantees the write data is persistent, should power be lost

IMPLEMENTING THE PMEM PROGRAMMING MODEL OVER AN RDMA NETWORK (APM)

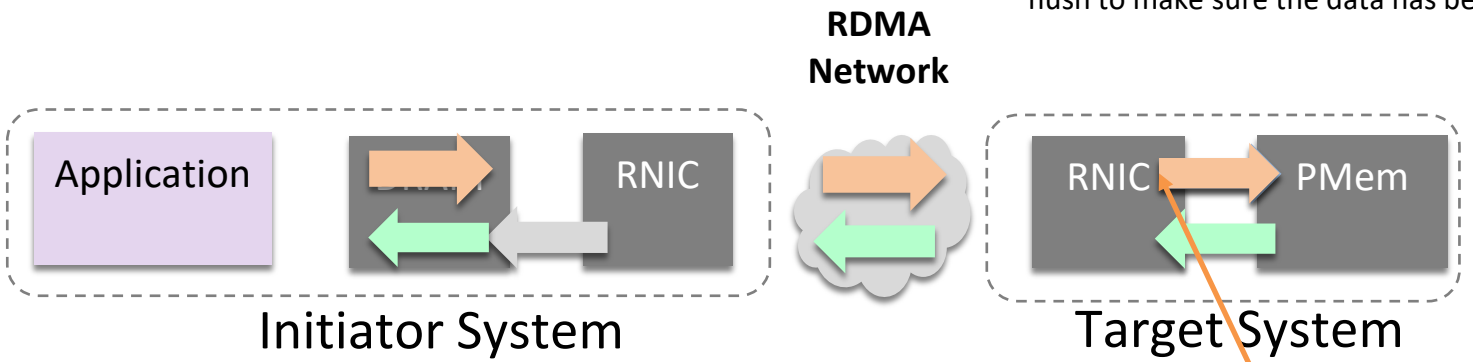


The RDMA Write response only tells the Application that the Target RNIC has sent the write data on to the final storage location in persistent memory.

Writing data to remote persistent memory



As the programming model states, the Application will need to execute a flush to make sure the data has been flushed to the persistence domain.

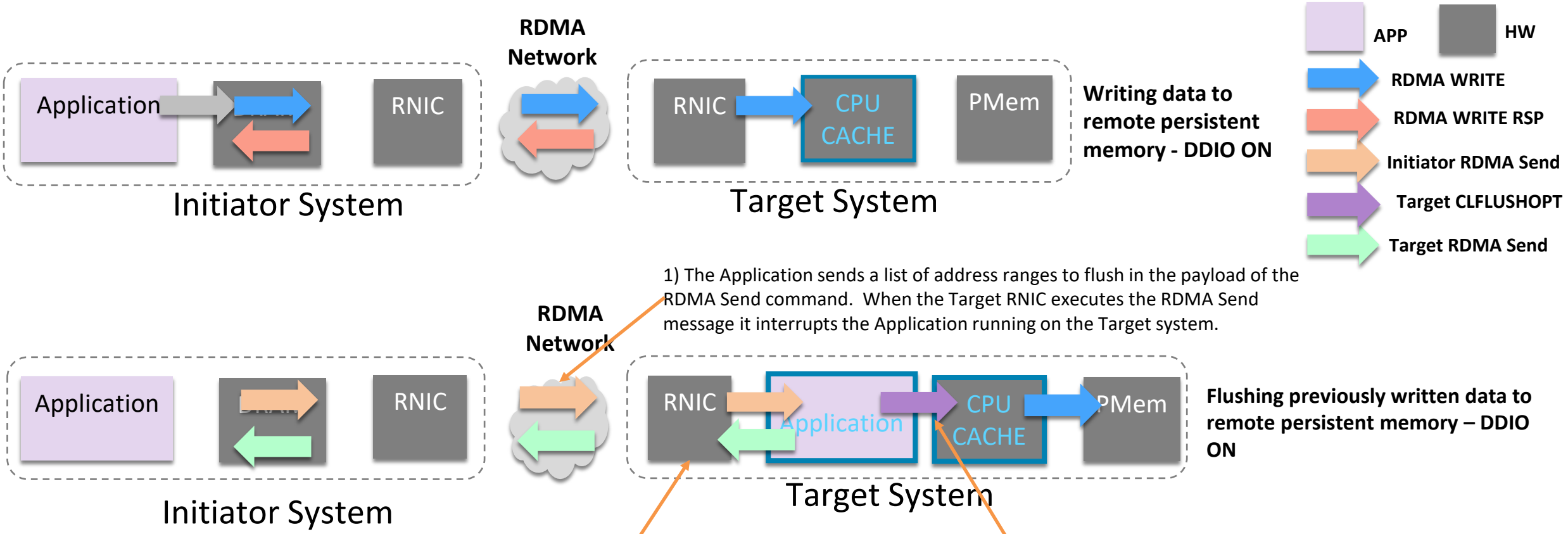


Flushing previously written data to remote persistent memory

The RDMA Read forces any write data to the persistence domain before the RDMA Read can proceed. This is because the ordering rules of both RDMA and PCIe.

"Appliance Persistency Method"

IMPLEMENTING THE PMEM PROGRAMMING MODEL OVER AN RDMA NETWORK (GPSPM)



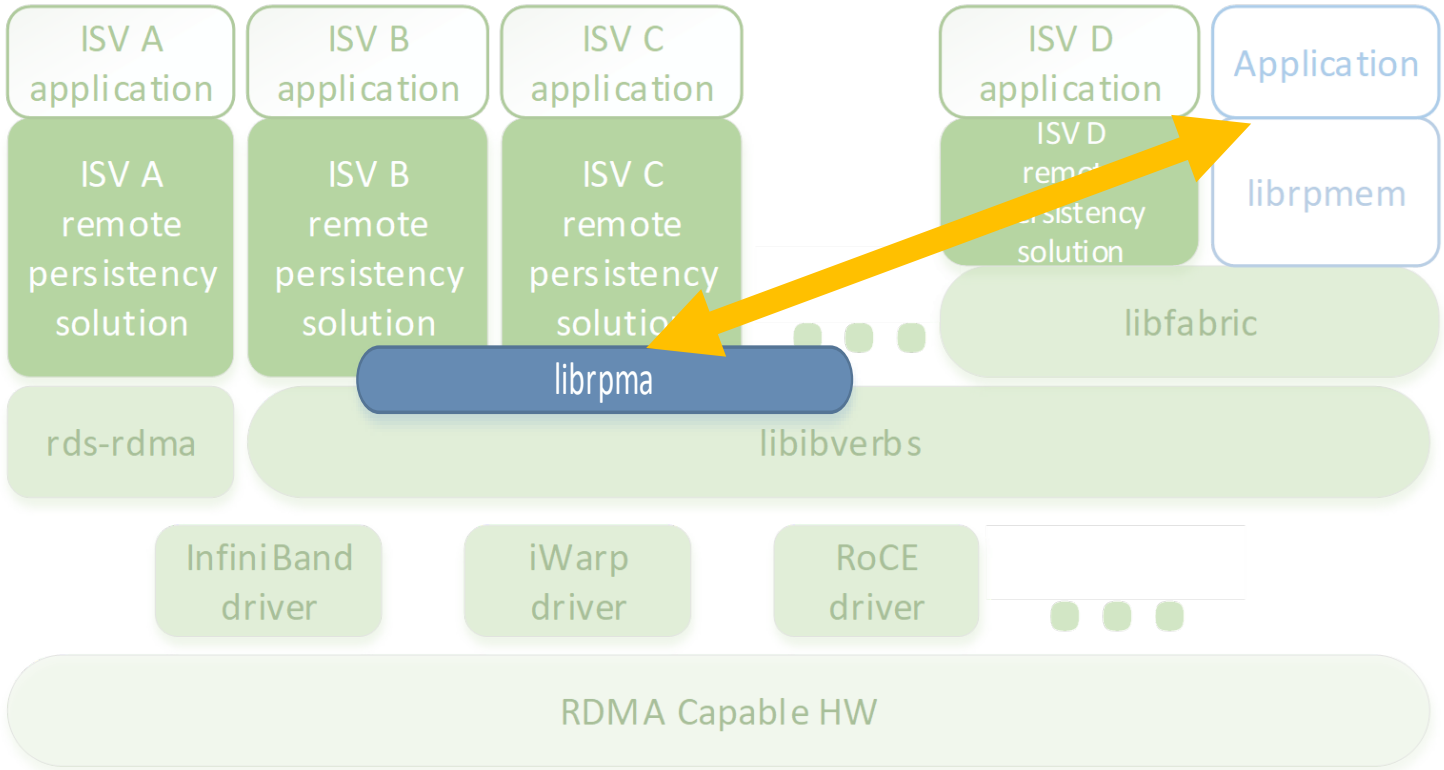
"General Purpose Server Persistency Method"

PMEM AND RDMA

RDMA accesses Intel® Optane™ PMem
in the same way it accesses DRAM

Remote PMem (RPMem)
is about well-known technologies
(like PCIe, RDMA) used in a new way

THE NEW LIBRPMA FOCUSES ON RPMEM USABILITY

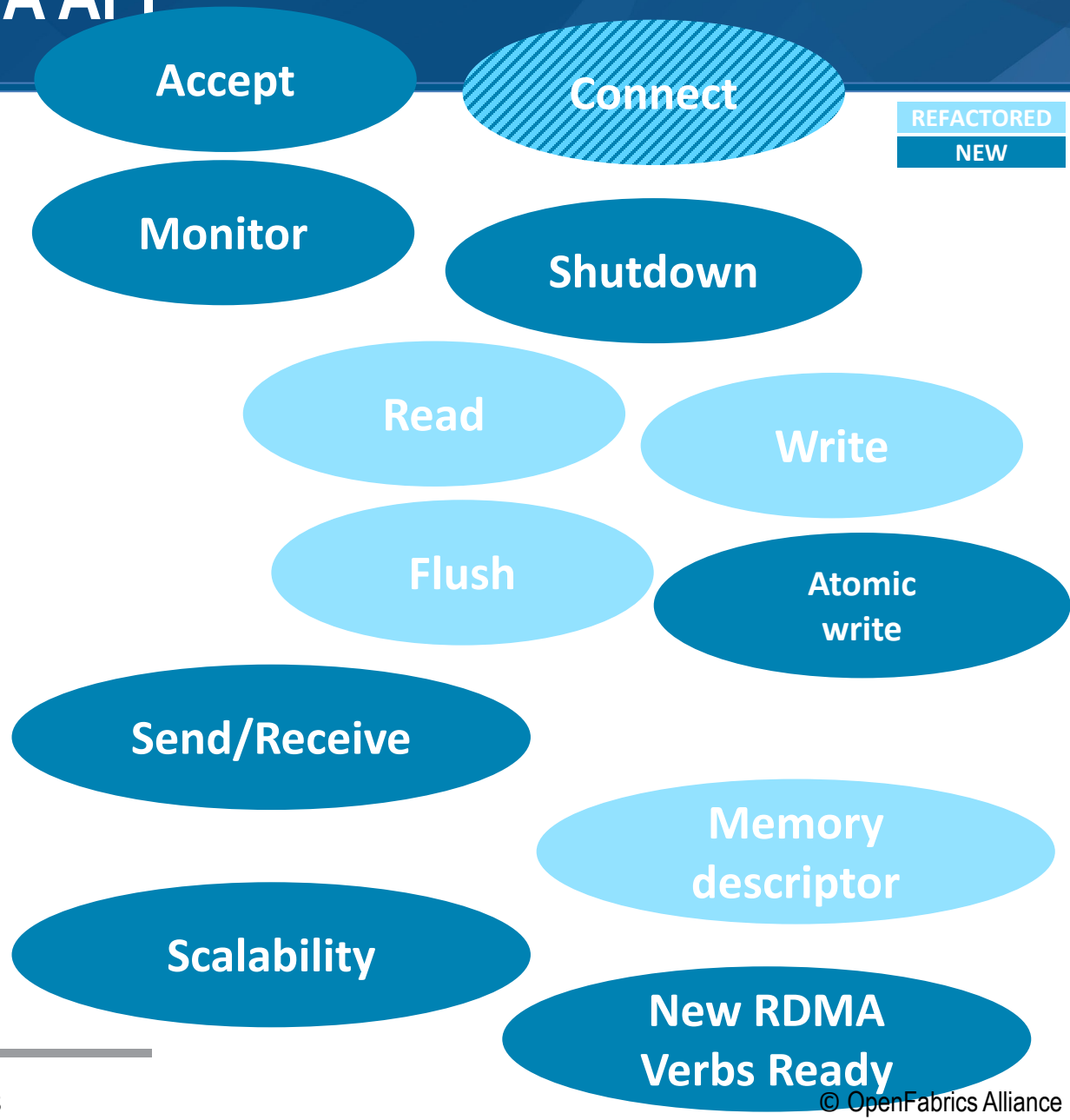


- memcpy-like API
- Hidden RDMA complexity
- Application can freely manage your PMEM all the time
- Minimum dependencies
- Enables Persistent Memory Programming Model

up to 50% RPMem source code reduction in an application that moves from libibverbs to librpma

LIBRPMA API

- Connection management
  - to ensure operations consistency
  - to hide RDMA complexity
- **Remote Persistent Memory Access (RPMA)**
  - Read, Write, Flush, Atomic write
- Messaging
  - also with PMEM-backed message buffers
- Memory management
  - r_key exchange support
- Ready to incorporate RDMA Memory Placement Extension



BASIC EXAMPLE – MEMORY REGISTRATION

Initiator node

```
rpma_mr_reg(peer,
    ptr, size,
    RPMA_MR_USAGE_WRITE_SRC,
    &src_mr);
```

Target node

```
rpma_mr_reg(peer,
    ptr, size,
    RPMA_MR_USAGE_WRITE_DST |
    RPMA_MR_USAGE_FLUSH_TYPE_PERSISTENT,
    &dst_mr);
```

BASIC EXAMPLE – REMOTE PERSISTENT MEMORY WRITE

Initiator node

Target node

```
rpma_write(conn,
    dst_mr, dst_offset,
    src_mr, src_offset,
    KILOBYTE, RPMA_F_COMPLETION_ON_ERROR, NULL);
```

BASIC EXAMPLE – REMOTE PERSISTENT MEMORY WRITE

Initiator node

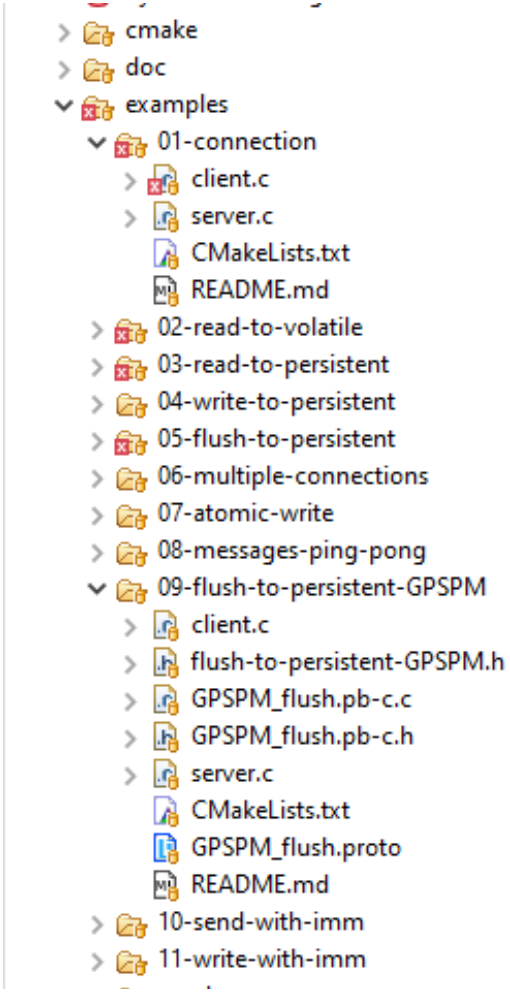
Target node

```
rpma_write(conn,
    dst_mr, dst_offset,
    src_mr, src_offset,
    KILOBYTE, RPMA_F_COMPLETION_ON_ERROR, NULL);
```

```
rpma_flush(conn,
    dst_mr, dst_offset,
    KILOBYTE, RPMA_FLUSH_TYPE_PERSISTENT,
    RPMA_F_COMPLETION_ALWAYS, FLUSH_ID);
```

https://github.com/pmem/rpma/tree/master/examples/05-flush-to-persistent

LIBRPMA EXAMPLES COLLECTION



- Connection establishment and management
- Read/write from/to DRAM/PMem
- Multiple connections (scalability)
- Atomic write
- Messaging
- Flush to persistent (both APM and GPSPM)
- Send/Write with immediate data

PEER SETUP

Client node

```
rpma_utils_get_ibv_context(SERVER_ADDR,
RPMA_UTIL_IBV_CONTEXT_REMOTE, &dev);
```

Server node

```
rpma_utils_get_ibv_context(SERVER_ADDR,
RPMA_UTIL_IBV_CONTEXT_LOCAL, &dev);
```

Common

```
rpma_peer_new(dev, &peer);
```

CONNECTION SETUP

Client node

Common

```
rpma_conn_cfg_new(&conn_cfg);
rpma_conn_cfg_set_cq_size(conn_cfg, 5);
…
```

Server node

```
rpma_conn_req_new(peer,
SERVER_ADDR, SERVER_PORT,
conn_cfg, &conn_req);
```

```
rpma_ep_listen(peer,
SERVER_ADDR, SERVER_PORT, &ep)
rpma_ep_next_conn_req(ep, conn_cfg, &con_req));
```

```
rpma_conn_req_connect(conn_req, NULL, &conn);
rpma_conn_next_event(conn, &conn_event);
assert(conn_event == RPMA_CONN_ESTABLISHED);
```

DISCONNECT

Client node

```
rpma_conn_disconnect(conn);

rpma_conn_next_event(conn, &conn_status);
assert(conn_status == RPMA_CONN_CLOSED);

rpma_conn_delete(&conn)
```

Server node

```
rpma_conn_next_event(conn, &conn_status);
if(conn_status == RPMA_CONN_CLOSED) {
    rpma_conn_disconnect(conn);
    rpma_conn_delete(&conn);
}
```

GPSPM EXAMPLE (09-FLUSH-TO-PERSISTENT-GSPSPM)

- APM's Flush and GPSPM's Flush do the same thing but they are not semantically the same
- GPSPM Flush can be combined with other application's operations
- No common API for APM and GPSPM flushes as we want to provide messaging API for applications

LIBRPMA 0.9 FEATURES

- Establishing an **RDMA connection** with a remote node and monitoring connection status
  - **Configurable** connection parameters
  - **Exchanging** small **information** during the connection establishment process
- **Read** remote **memory** (both **persistent** and volatile)
- **Write** to remote **memory** (both **persistent** and volatile**)**
- **Ensuring data placement** within the memory subsystem of a remote node (**flush**)
- Executing an **atomic write** (Intel platform-specific via standard aligned RDMA Write with appropriate fencing; native AtomicWrite-ready)
- **Polling**, in a non-blocking way, for RDMA **operations completions** and **connection status** changes
- Logging all **diagnostic/status to syslog** and/or on stderr
- A transparent FileSystem DAX support (on selected RNICs)
- Providing **easy to use RDMA Send/Receive** messaging (also with PMem-backed buffers)

- Additionally:
  - Examples collection
  - A Fio based benchmarking environment

THE LIBRPMA LIBRARY ONE YEAR LATER

- Use cases
  - The librpma is going to be used in Ceph and SQL-like engine
  - The librpma library is utilized by a number of customers as a reference solution
  - The librpma library is used as a fast RPMem ramp up tool

  - FIO engines for APM, GPSPM and AoF are based on librpma
    - all of these allows for easy, on-premise RPMem benchmarking

- API is stable but small tuning is still possible in response to customer feedback (0.9)

- New features under development(performance and usability improvements)
  - Separate Receive Completion Queue
  - More examples
    - Send/Recv with PMem
    - Connection errors handling

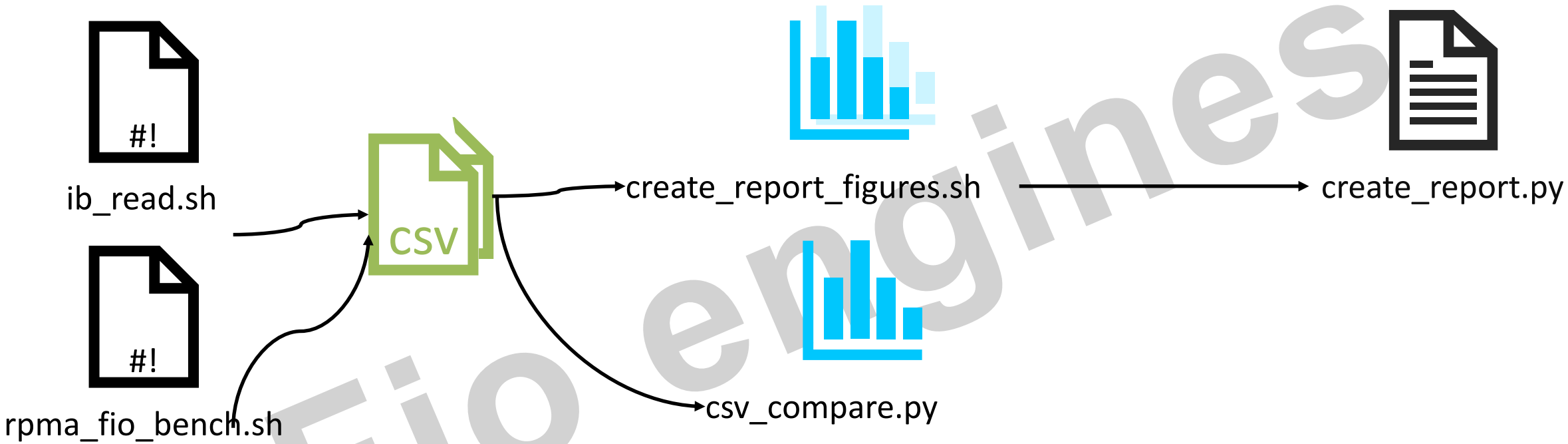
RPMEM BENCHMARKING ENVIRONMENT

- Fio engines (librpma APM/GPSPM, librpma AoF)
  `github.com/pmem/fio` => `github.com/axboe/fio/pull/1186`
  - bandwidth/latency
  - remote DRAM vs RPMEM (dev-dax, fs-dax)
  - numjobs, blocksize, iodepth, readwrite

- pmem.io/rpma
  - *Performance – Tuning* blog describing RPMem related BIOS/OS settings
  - *Direct Write to PMem* blog describing RPMem enabling on Intel servers

- github.com/pmem/rpma/tree/master/tools/perf
  - `rpma_fio_bench.sh` – to collect performance data
    - Fio job files templates
    - remote DDIO control, local/remote NUMA control, PMem/DRAM
  - `csv_compare.py` - for results comparison (research, manual analysis)
  - `create_report.sh` - for comprehensive performance report
    - `rpma_fio_bench.sh 192.168.1.1 all all all`
    - adjustable report template

BENCHMARKING TOOLSET

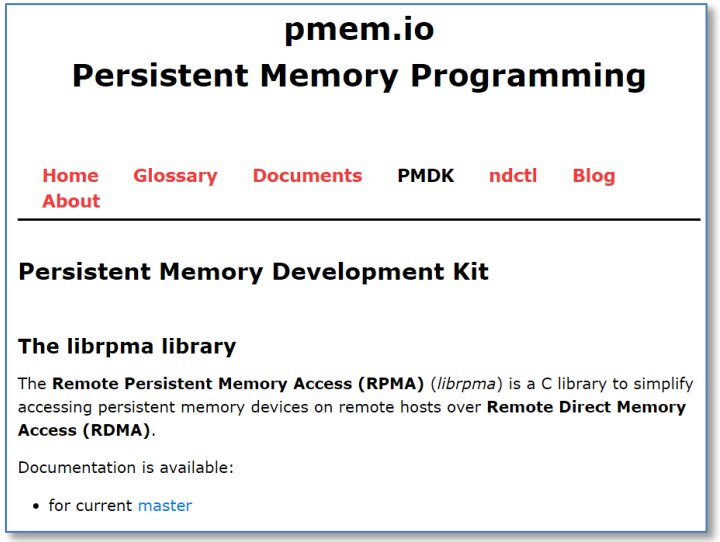


github.com/pmem/rpma/tree/master/tools/perf

MORE DOCUMENTATION AVAILABLE

Visit

- pmem.io/rpma for official documentation
- github.com/pmem/rpma to
  - build the library
  - run the examples
  - setup the benchmarking environment



You do not need neither an Intel® Optane™ PMem nor RDMA-capable NIC to start examples

You can play with RPMA examples on any Linux desktop

2021 OFA Virtual Workshop

THANK YOU

Tomasz Gromadzki, Software Architect (RPMEM)

Intel Corporation

LEGAL NOTICE AND DISCLAIMERS

This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

No License (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, Intel Xeon, and Optane Persistent Memory are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as property of others.

BACKUP

IS THE LIBRPMA ATOMIC WRITE AN ATOMIC OPERATION?

- IBTA Spec does not support atomicity for RDMA Write
- librpma implementation assumes the aligned 8-byte data won't be torn either in RDMA HW/SW stack or on the PCIe bus level
- The current implementation explicitly uses fencing to ensure ordered data processing according to the upcoming RDMA.AtomicWrite ordering rules
- The native RDMA.AtomicWrite will be used whenever and wherever it will be available