12th ANNUAL WORKSHOP 2016

KERNEL VERBS API UPDATE

Leon Romanovsky

Mellanox Technologies

[April 4th, 2016]

AGENDA

- Memory registration API
- CQ polling API
- Draining QP
- Generic RDMA WRITE/READ API

MEMORY REGISTRATION API

MULTIPLE MEMORY REGISTRATION METHODS

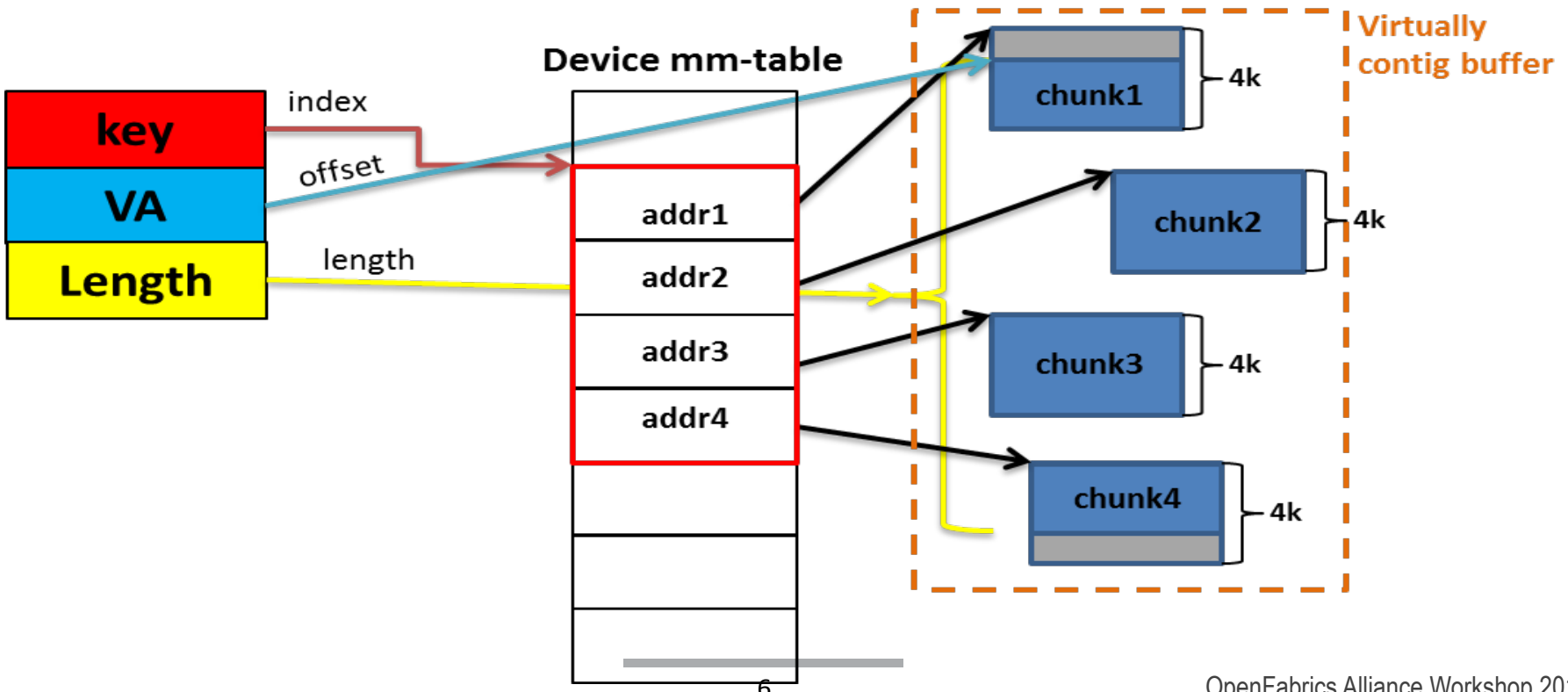
- **Physical memory regions (MR)**
  - Synchronous interface
  - Every registration causes to new MR
- **Fast memory registration (FMR)**
  - Fast synchronous interface
  - Weak deregistration semantics
  - Not widely adopted
- **Memory windows (MW)**
  - Fast, asynchronous interface
  - Binds continuous apertures to existing memory regions
    - Not relevant to kernel ULPs
  - Removed from kernel API
- **Fast memory registration mode (FRWR)**
  - Asynchronous interface
  - Maps blocks of physical memory
  - Widely adopted

MULTIPLE MEMORY REGISTRATION METHODS

- **Physical memory regions (MR)**
  - Synchronous interface
  - Every registration causes to new MR
- **Fast memory registration (FMR)**
  - Fast synchronous interface
  - Weak deregistration semantics
  - Not widely adopted
- **Memory windows (MW)**
  - Fast, asynchronous interface
  - Binds continuous apertures to existing memory regions
    - Not relevant to kernel ULPs
  - Removed from kernel API
- **Fast memory registration mode (FRWR)**
  - Asynchronous interface
  - Maps blocks of physical memory
  - Widely adopted

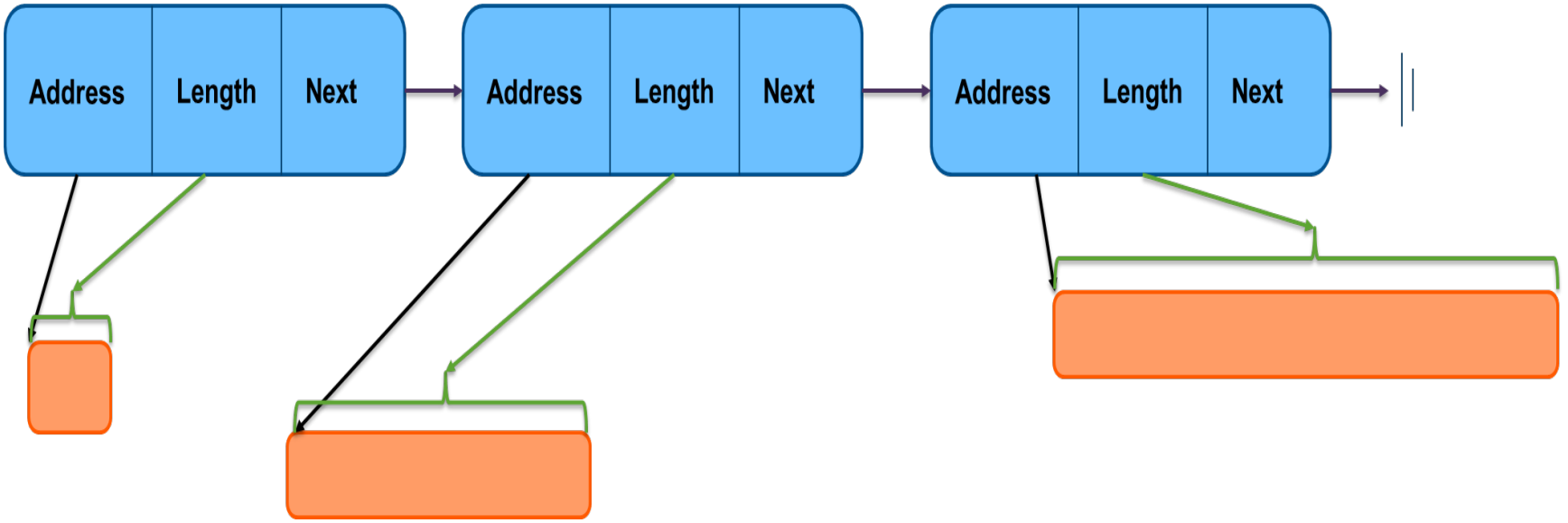
FRWR MAPPINGS

- All buffers must be block aligned (page shift)
- The first buffer can have first byte offset (FBO)
- The last buffer can end before the end of block (EOB)



FRWR DRAWBACKS

- **Allocation of free memory region and a fast_reg_page_list**
- **Translation from S/G list to page vector for each ULP**
- **No ability to support arbitrary S/G**
  - Each ULP bridged this semantic gap



OLD FRWR API

OP	API
Allocate	`frpl = ib_alloc_fast_reg_page_list()` `mr = ib_alloc_fast_reg_mr()`
Free	`ib_free_fast_reg_page_list(frpl)` `ib_dereg_mr(mr)`

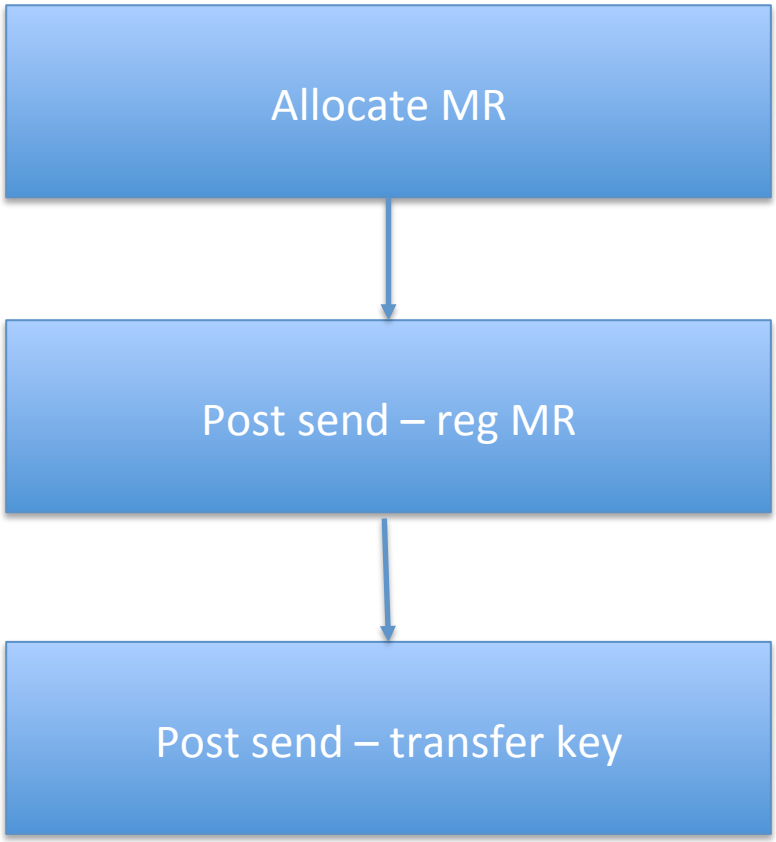
Register interface (post_send)	`struct {`
	`  u64 iova_start;`
	`  struct ib_fast_reg_page_list *page_list;`
	`  unsigned int page_shift;`
	`  unsigned int page_list_len;`
	`  u32 length;`
	`  int access_flags;`
	`  u32 rkey;`
	`} fast_reg;`

Opcode	`IB_WR_FAST_REG_MR`

NEW FRWR API

OP	API
Allocate	`mr = ib_alloc_mr(pd, max_num_sg, mr_type)`
Free	`ib_dereg_mr(mr)`
S/G list mapping	`nents = ib_map_map_mr_sg(mr, sg, sg_nents, page_size);`
Register interface (post_send)	`struct { struct ib_send_wr wr; struct ib_mr *mr; int access; u32 key; } ib_reg_mr;`
Opcode	`IB_WR_REG_MR`

ULP REGISTRATION FLOW



CQ POLLING API

CQ POLLING CONSIDERATIONS

- **Different completion contexts**
  - Kernel threads
  - Work queues
  - Software IRQs
  - Hardware IRQs
- **Fairness between multiple CQs**
- **CQ re-arm policy**
- **Handling missed events**

CQ POLLING USAGE UNCERTAINTY

- Work requests (WR) context return path
- Work request IDs (wr_id) (un)reliable
- Multiple sources of post_send completions
- Polling in batches or one-by-one
- Multiple CPUs (affinity)
- CPU/NUMA locality

CQ POLLING API

OP	API

Decide on polling type	`enum ib_poll_context {`
	`  IB_POLL_DIRECT, /* caller context, no hw completions */`
	`  IB_POLL_SOFTIRQ, /* poll from softirq context */`
	`  IB_POLL_WORKQUEUE, /* poll from workqueue */`
	`};`

Allocate	`struct ib_cq *ib_alloc_cq(dev, private, nr_cqe, comp_vector, poll_ctx);`
Free	`void ib_free_cq(cq)`

Completion return (filled in WR)	`struct ib_cqe {`
	`  void (*done)(struct ib_cq *cq, struct ib_wc *wc);`
	`};`

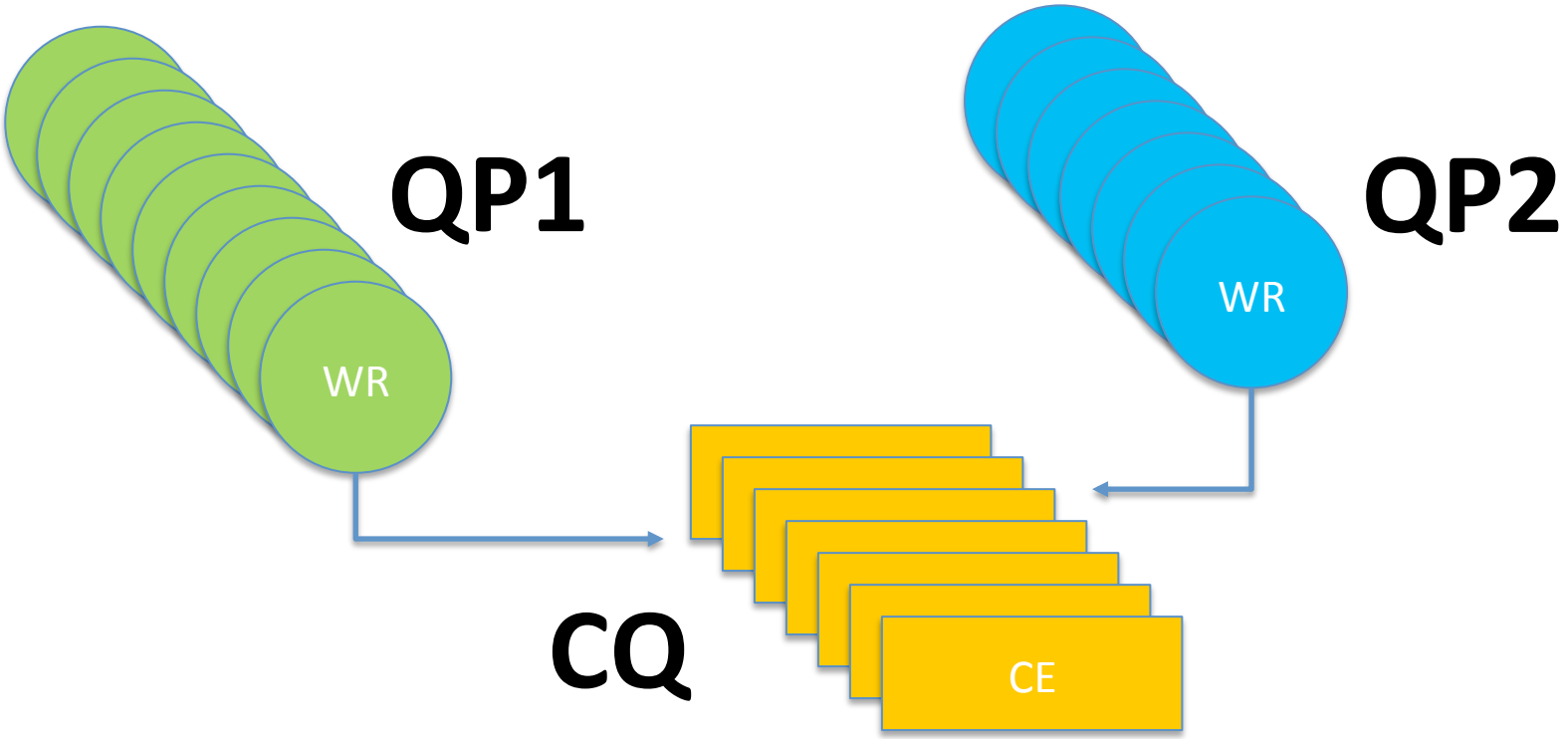
CQ POLLING SUMMARY

- **Unify completion queue logic from different ULP clients**
- **Simplify completion queue polling and interrupt handling**
  - No need to poll, handle events and maintain logic
- **Resolve the error completions unreliability**
- **Support different polling schemes**
- **Performance optimized**

DRAINING QP

PROBLEM

- Unknown when WRs are completed after ceasing posts

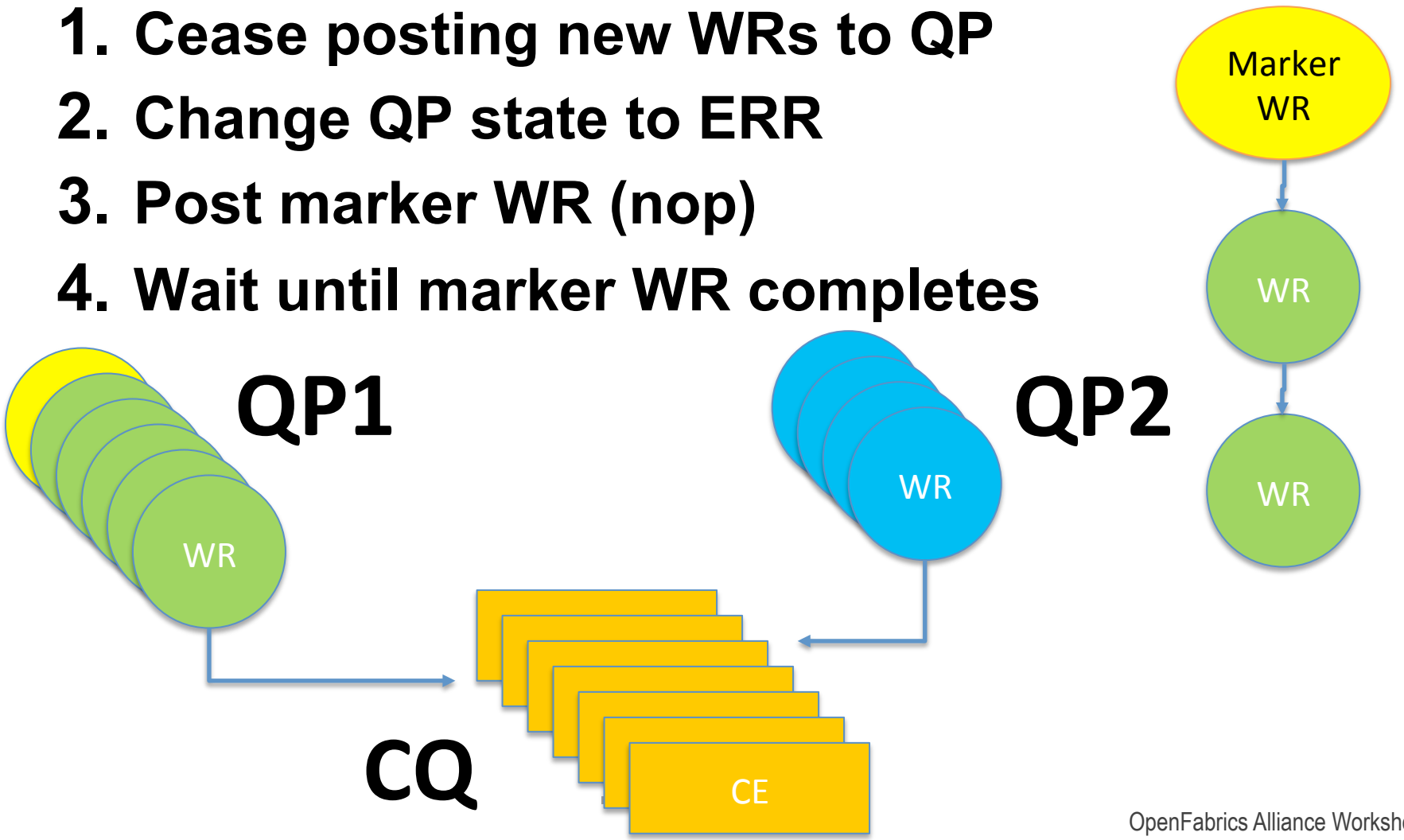


COMMON METHODS

1. Wait until all previously posted WRs complete and then destroy QP - may be indefinite
2. Destroy QP without waiting for completions - need to maintain a shadow state of all in-flight WQEs in order to free related state
3. Modify QP to error, and then poll related CQ until it is empty - works only if CQ is associated with only the same QP

ROBUST GENERIC DRAINING

1. Cease posting new WRs to QP
2. Change QP state to ERR
3. Post marker WR (nop)
4. Wait until marker WR completes



DRAIN QP API

OP	API
Drain SQ	`void ib_drain_sq(qp)`
Drain RQ	`void ib_drain_rq(p)`
Drain QP	`void ib_drain_qp(qp)`

- **IB/core code to drain SQ/RQ/both in single function call**
- **Protect from use-after-free error flow**
- **Synchronized operation in one place**

GENERIC RDMA READ/WRITE (WIP)

MOTIVATION

- RDMA READ/WRITE is common operation for storage protocols
- Every ULP has own implementation
- Lack of support for generic S/G lists
- HCA aware implementations
- Reuse pre-allocated MRs (MR pool)
- Support large number of S/G entries

REFERENCES

- IB: new common API for draining queues by Steve Wise
- Generic RDMA READ/WRITE API by Christoph Hellwig
- Completion queue abstraction by Christoph Hellwig and Sagi Grimberg
- New fast registration API by Sagi Grimberg

12th ANNUAL WORKSHOP 2016

THANK YOU

Leon Romanovsky

Mellanox Technology