RDMA

(Remote Direct Memory Access)

Contents

1. What's RDMA:

Remote Direct Memory Access (**RDMA**) is a technology that allows computers in a network to exchange data in main memory without involving the processor, cache or operating system of either computer.

- **Remote**: transfer data in a network.
- **Direct**: no Operating System Kernel involvement and everything offload onto interface card.
- **Memory**: user space application virtual memory, virtual memory.
- **Access**: send, receive, read(only in **OFA**), write(only in **OFA**) and atomics (only in **IB**) operations.

1.1 Message passing

- a common programming model
- useful for communicating between any two process
- a message is a abstract chunk of data
- often used in parallel computing and interprocess communication
- processes send and receive messages to each other
- processes can use message passing to synchronize, by waiting for a message

RDMA is a way of transporting messages between servers or between a server and a client. RDMA is a message passing paradigm.

2. Benefits

- Zero-copy : applications can perform data transfer without the network software stack involvement and data is being send received directly to the buffers without being copied between the network layers. data moves directly from user memory on one side to user memory on the other side.
- Kernel bypass : applications can perform data transfer directly from user space without the need to perform context switches. User has direct access to the Channel Adapter.
- No CPU involvement : applications can access remote memory without consuming any CPU in the remote machine. The remote memory machine will be read without any intervention of remote process (or processor). The caches in the remote CPU(s) won't be filled with the accessed memory content.
- Scatter/gather entries support : RDMA supports natively working with multiple scatter/gather entries i.e. reading multiple memory buffers and sending them as one stream or getting one stream and writing it to multiple memory buffers
- Message based transactions : the data is handled as discrete messages and not as a stream, which eliminates the need of the application to separate the stream into different messages/transactions.
- asynchronous operation : threads not blocked during I/O transfers

3. Goals of RDMA

- High Bandwidth Utilization
- Low Latency
- Low CPU utilization

4. RDMA Technologies

there are several network protocols which support RDMA :

- **RoCE**: RDMA over Converged Ethernet. **RoCE** is a network protocol that enables RDMA over an Ethernet network by defining how it will perform in such an environment.
- **iWARP**: Internet Wide Area RDMA Protocol. **IWARP** leverages the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) to transmit data.
- **InfiniBand**: RDMA is the standard protocol for high-speed InfiniBand network connections.

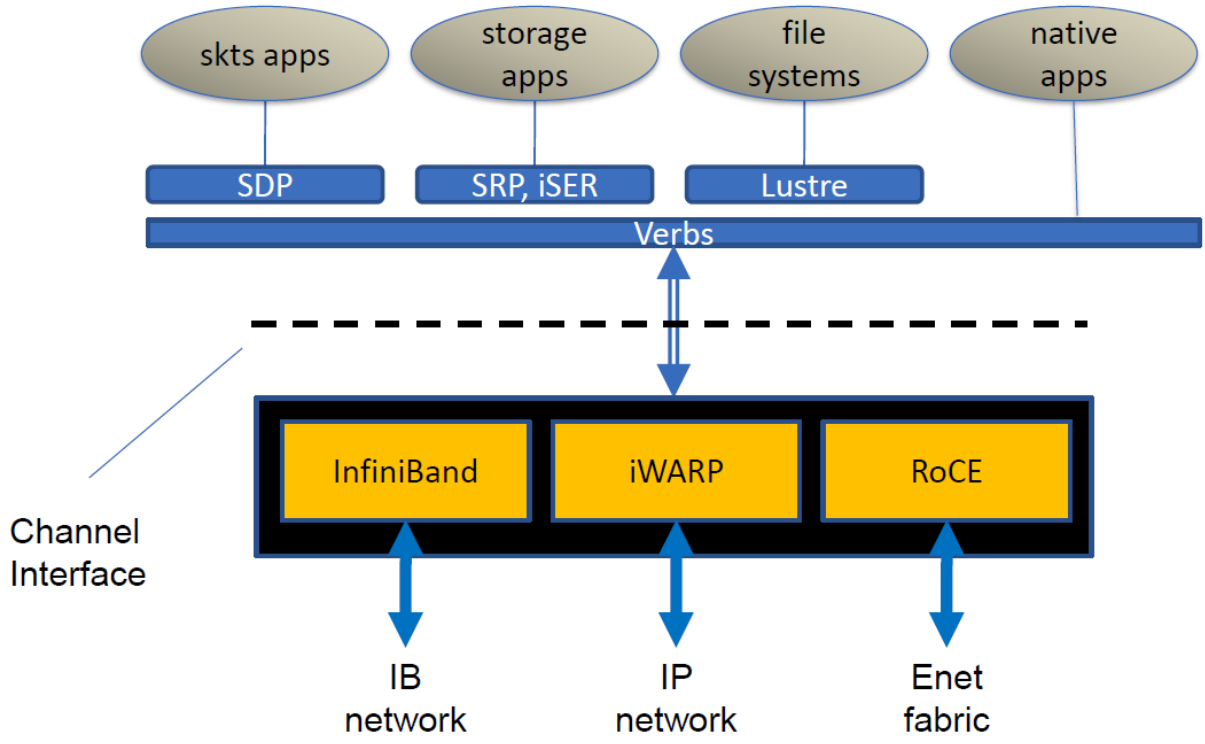

Figure 1 wire protocols

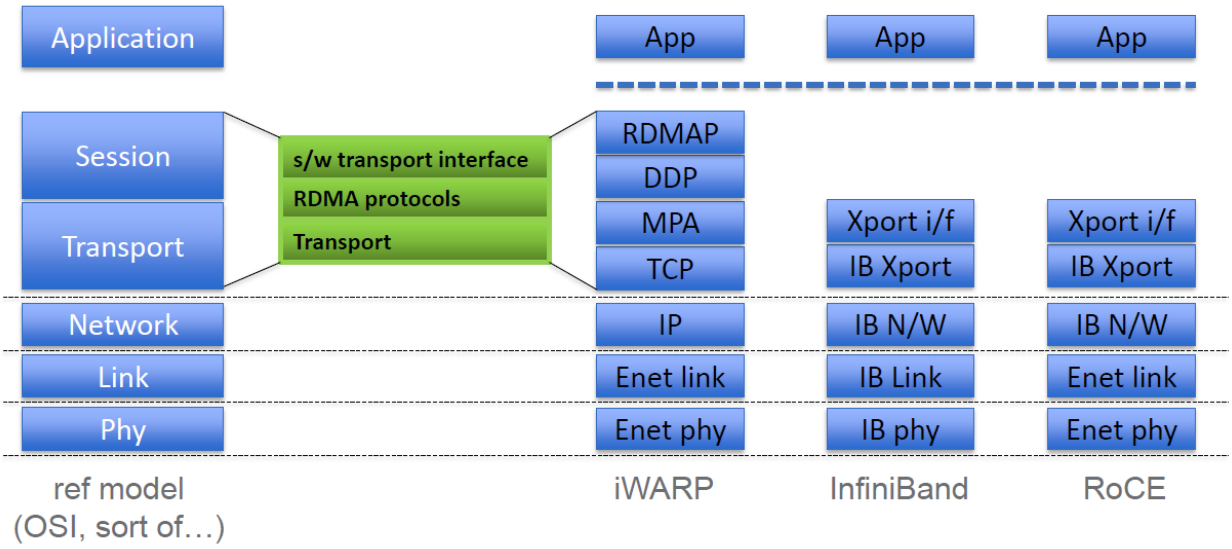

Figure 2 wire level protocols

5. message passing:

- SEND/RECEIVE : 'channel semantic', double ended operation.it involves both the Requester and the Responder. never uses rkey field in struct ibv_mr and never uses access rights IBV_ACCESS_REMOTE_READ or IBV_ACCESS_REMOTE_WRITE.
  - sender must issue listen() before client issues connect()
  - both sender and receiver must actively participate in all data transfer()
  - sender doesn't know remote receiver's virtual memory location
  - receiver doesn't know remote sender's virtual memory location

!*NOTE*: order and timing of send() and recv() are relevant. memory involved in transfer is untouchable between start and completion of transfer.

Between Posting and Completion user memory containing message data is undefined and should NOT be changed by user program.

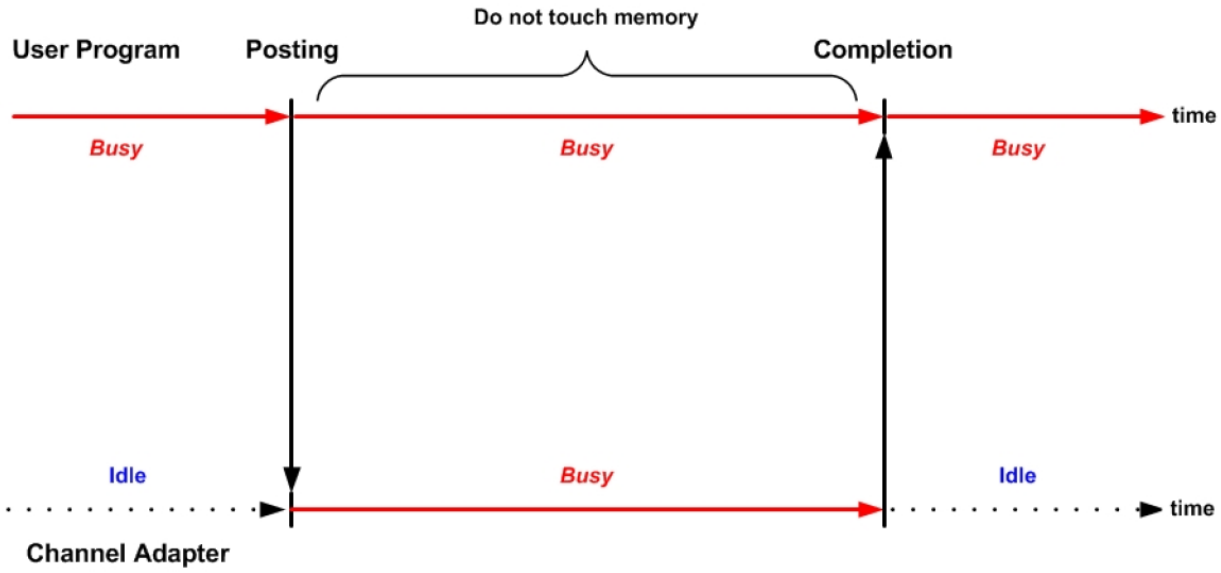

Figure 3 memory status

- RDMA READ/RDMA WRITE : 'memory semantic'
  - active side calls rdma_post_write()
  - prior to issuing this operation, active side must obtain passive side's address and key
  - passive side provides "metadata" that enables the data "push", but doesn't participate in it.
  - never use rkey filed in struct ibv_mr
  - never use access rights IBV_ACCESS_REMOTE_READ or IBV_ACCESS_REMOTE_WRITE

!*NOTE*: on iWARP the client must be the size to send the first message.

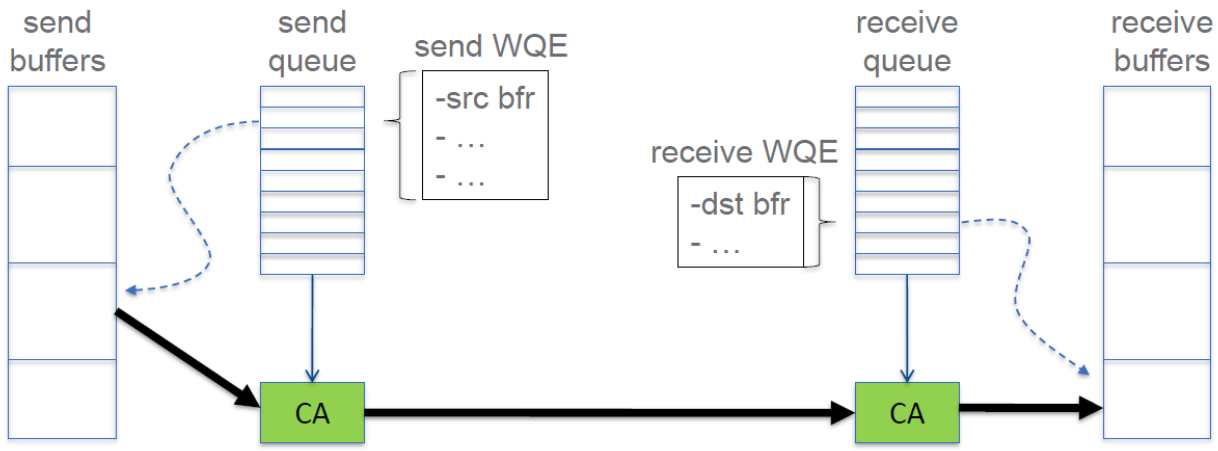

Figure 4 RDAM SEND/RECEIVE

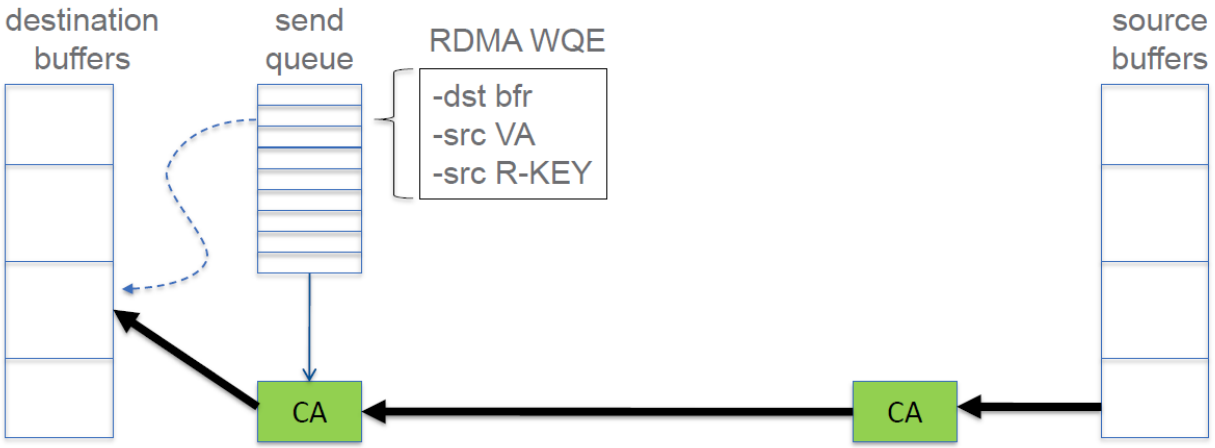

Figure 5 RDMA READ

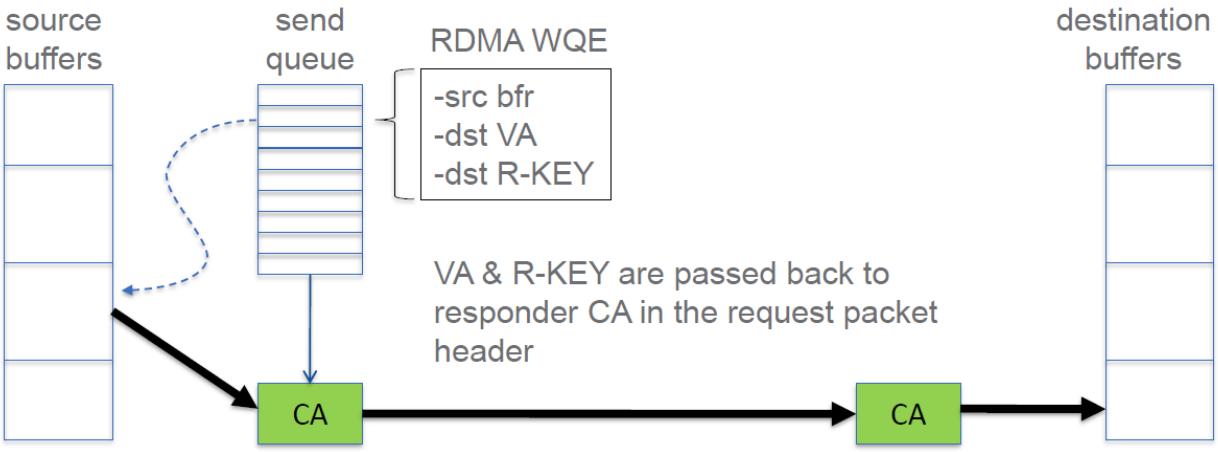

Figure 6 RDMA WRITE

6.Verbs:

An API used by an application to control and conduct an RDMA operation. the programmatic interface to the RDMA Messaging Service.

- InfiniBand specification
- OpenFabrics Alliance (OFA) Verbs API

Libraries that access RDMA:

- MPI(Message Passing Interface): OpenMPI, MVAPICH, Intel MPI; Main tool for HPC(High Performance Computing)
- File System: Lustre, NFS_RDMA
- Storage: SRP(SCSI RDMA Protocol), Iser(iSCSI extensions for RDMA)
- SDP : Socket Direct Protocol
- mva: Mellanox Messaging Accelerator
- SMC-R

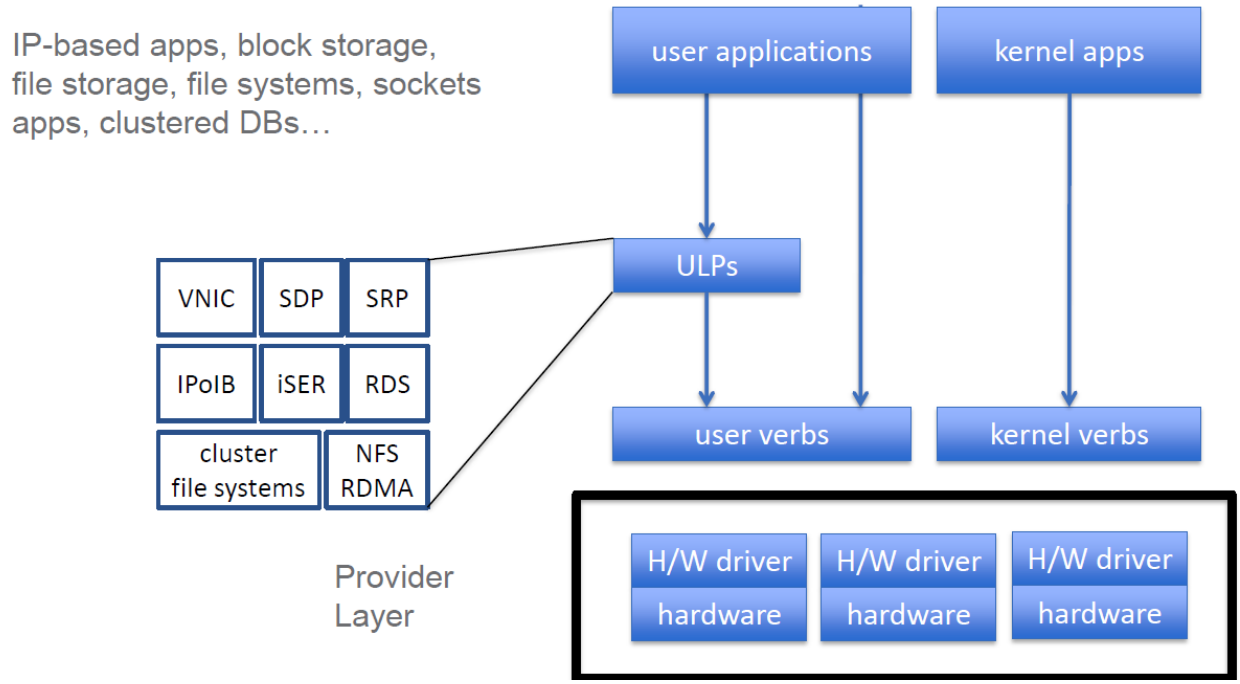

Figure 7 high level look

the programmatic interface to the RDMA Messaging Service and How to use Verbs:

- Open an HCA
- Create a Protection Domain
- Create a Queue Pair
- Create a Completion Queue

- Register memory regions
- Post work requests
- Wait for completions

How to setup a connection:

- Create communication identifier – cm_id
- Bind client to RDMA device
    - translate DNS name into internal address
    - resolve address to local RDMA device
    - resolve reoute to server
- setup queue paire
    - allocate protection domain
    - create completion queue
    - create queue pair
- set up client buffers
- connect client to server

7.Other Concept:

7.1 Protection Domain(PD) :

a sort of container that is used to associate a QP with a memory region and ensure that only the QPs that are registered to that virtual memory can access it. which enables Channel Adapters to quickly determine if an operation is allowed. data transfers on a Queue-Pair can only utilize Memory Regions in that Queue-Pair's PD.

- one or more QPs
- a region of memory or many memory regions
- data transfers on a QP can only utilize Memory Regions in that QP's PD.

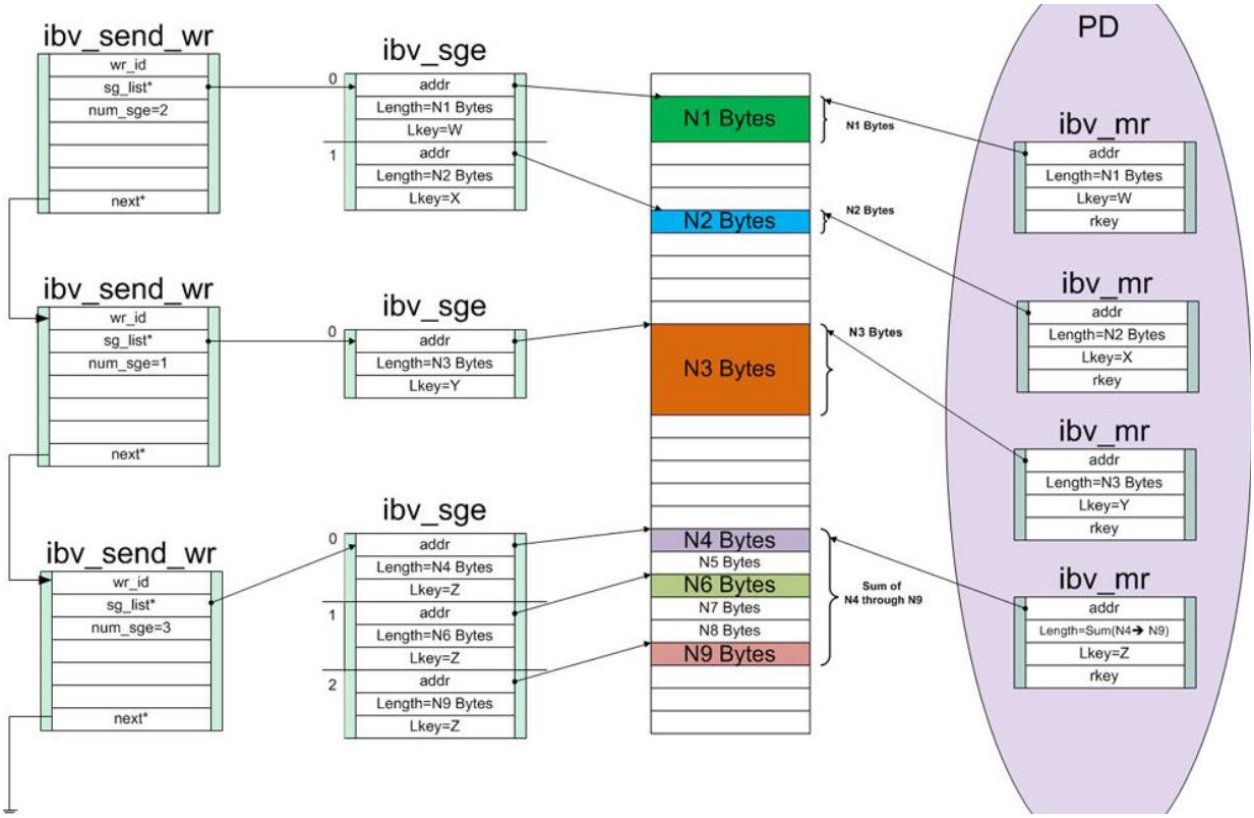

Figure 8 protect domain

7.2Memory registration :

register a region of memory with the HCA to enable to transfer data directly to/from host memory without CPU intervention. this operation will map the WQs and CQs into the application's virtual address space. This means the application can manipulate the channel directly – no need for a context switch to a privileged entity.

- Virtual Address(VA)
- R-KEY : passed to responder, used by remote Channel Adapter to access MR.
- L-KEY : the app access the memory, used by local Channel Adapter to access MR.

User need to specify Memory Region of contiguous virtual memory, MR's physical pages are pinned in memory, so they are NOT swapped out during data transfer, the mapping between virtual address to physical address is written to Channel Adapter during registration process.

7.3 Memory Region(MR) :

user-defined area of memory registered with certain user-defined access rights in a Protection Domain. each memory region is a member of one PD.

7.4 MR Access rights:

- IBV_ACCESS_LOCAL_WRITE

  allow local CA to write to local registered memory.

- IBV_ACCESS_REMOTE_WRITE

  allow remote CA to write to local registered memory.

- IBV_ACCESS_REMOTE_READ

  allow remote CA to read from local registered memory.

By default. local CA always allowed to read from local registered memory. ibv_post_send() with MRs with only default local read access and ibv_post_recv() requires MRs with at least IBV_ACCESS_LOCAL_WRITE.

!*NOTE*: iWAPR requires IBV_ACCESS_LOCAL_WRITE if IBV_ACCESS_REMOTE_WRITE is used.

7.5 Relevant Keys:

- L-KEY, R-KEY: control access to registered memory from the local user or a remote user.
- Q-KEY(RD, UD only): used to govern access to a remote QP. Normally, this access would be governed by comparing the SLID/DLID against QP context.

7.6 Scatter-gather:

scatter-gather list(sge, array) : describes chunk of virtual memory for transfer

- scatter : allow one receive operation to split data from single message “on the wire” into different chunks of local virtual memory. each element in the sg_list array describes on chunks of virtual memory.

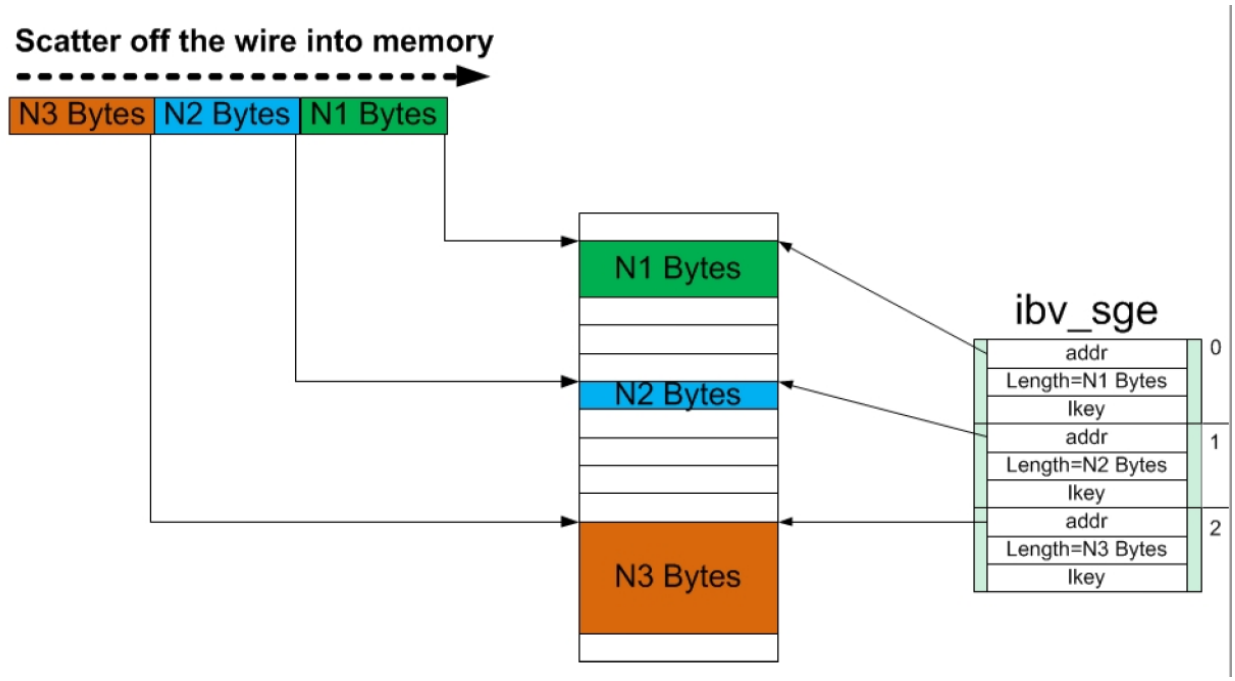

Figure 9sge scatter

- gather : allows on SWR to pull together data from different chunks of local virtual memory into single message “on the wire”. each element in sg_list array describes one chunk of virtual memory.

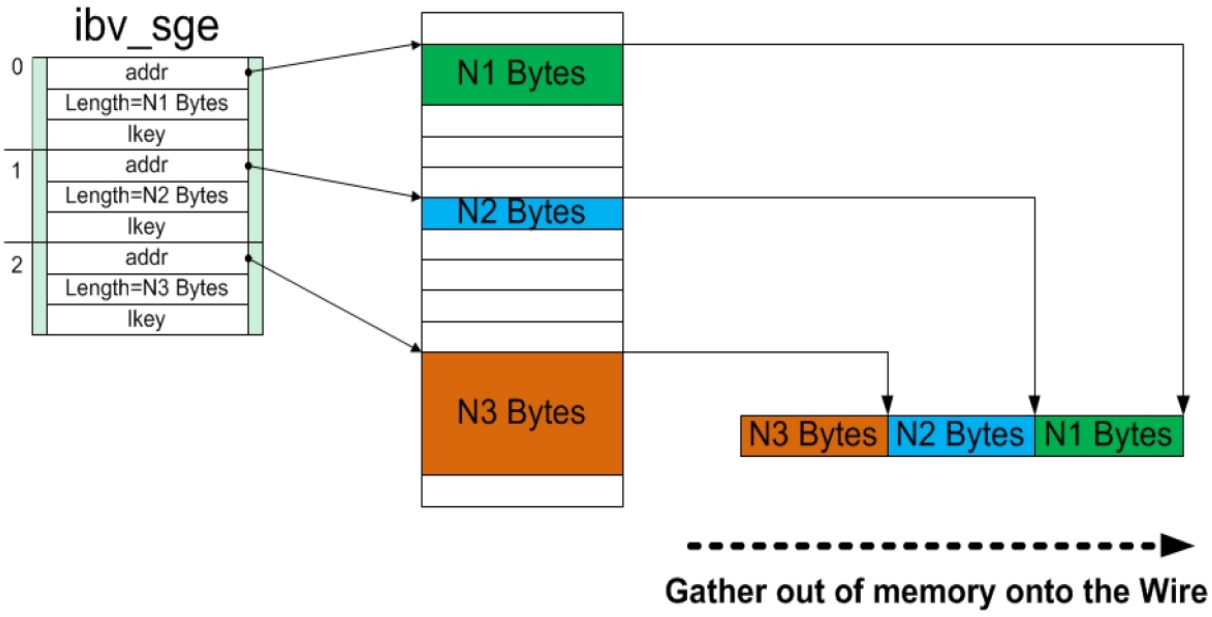

Figure 10 sge gather

In practice, most sg_list has only one element no matter what’s the type of it.

8. Verbs functions

OFED currently supports InfiniBand, iWARP and RoCE. they present the same interface(API) and semantic behavior.

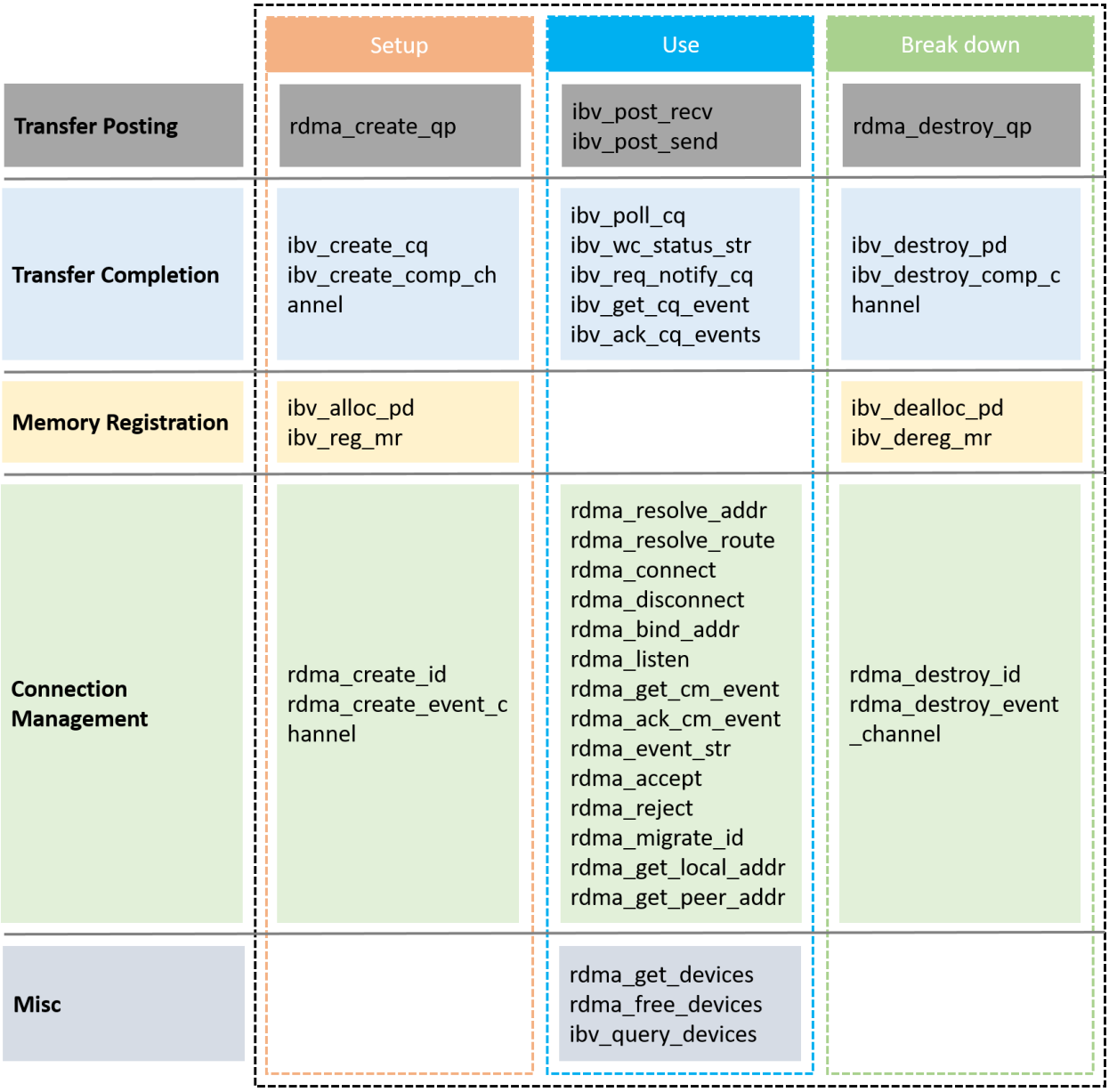

Figure 11 Verbs component functions

ibv_post_recv()

parameters:

- QP: Queue Pair

- Pointer to list of Receive Work Requests – RWR

- Pointer to bad RWR in list in case of error

return value:

- == 0 all RWRs successfully added to recv queue(RQ)

- != 0 error code

ibv_post_send()

parameters:

- QP: Queue Pair

- Pointer to linked list of Send Work Requests – SWR

- Pointer to bad SWR in list in case of error

return value:

- == 0

- != 0

ibv_poll_cq()

purpose: detect completion of a transfer.

parameters:

- Completion Queue - CQ

- Number of slots in Work Completion array

- Array of Work Completion slots – WC

return value:

- >=0 – number of WC slots filled in

- <= error code

ibv_create _cq()

parameters:

- cm_id

- desired total number of slots in new queue

- user-defined identification of this queue

- completion channel

- comp_vector

return value:

- pointer to new instance of struct ibv_cq

ibv_destry _cq()

parameters:

- pointer to struct ibv_cq returned by ibv_create_cq()

ibv_alloc_pd()

parameters:

- Context – verbs filed of associated cm_id

return value:

- pointer to new instance of struct ibv_pd

ibv_dealloc_pd()

parameter:

- pointer to struct ibv_pd returned by ibv_alloc_pd()

ibv_reg_mr()

parameters:

- protection domain

- start address of region

- byte length of region

- access rights for region

return value:

pointer to new instance of struct ibv_mr

ibv_reg_mr()

paramters:

- pointer to struct ibv_mr returned by ibv_reg_mr()

rdma_create_id()

parameters:

- Channel for reporting events

- Storage for returning pointer to struct rmda_cm_id

- user-defined identification of the cm_id

- port space RDMA_PS_TCP

return value:

- ==0 cm_id created successfully.

- !=0 error code stored in global errno

rdma_destroy_id()

parameters:

- pointer to struct rdma_cm_id returned by redma_create_id()

return value:

- ==0 sucessful

- !=0 error code stored in global errno

rdma_resolve_addr()

parameters:

- cm_id , communication identifier

- src_addr, client's sockaddr

- dst_addr, server's sockaddr

- timoute, maximum time to wait in milliseconds.

return value:

- 0, cm_id successfully bound to local RDMA device

- -1, error code stored in global errno

rdma_resolve_route ()

purpose: establish a route through fabric to server.

parameters:

- cm_id, communication identifier

- timeout, maximum time to wait in milliseconds

return value:

- 0 cm_id found route to remote RDMA device

- -1 error code stored in globa errno

rdma_create_qp ()

parameters:

- cm_id, communication identifier

- protection domain

- structure of values to initialize new qp

return value:

- 0, qp successfully created.

- -1, error code stored in global errno.

rdma_destroy_qp ()

parameters:

pointer to struct ibv_cq returned by rdma_create_qp() in qp field of struct rdma_cm_id

return value:

- == 0, ok

- != 0, error code

rdma_connect ()

parameters:

- cm_id

- structure of values to initialize new connection

return value:

- 0, connection was successfully established

- -1, error code stored in global errno.

!*NOTE*: on InfiniBand, must have subnet manager running before rdma_connect() will succeed.

Once client has connected, it can start transmitting data, When finished transmitting, client must disconnect, then break-down data structures in the reverse order they were set up.

rdma_disconnect ()

parameters:

- cm_id

return value:

- 0, connection was successfully disconnected.

- -1, error code stored in global errno.

!*NOTE*: error code EINVAL means remote size disconnected first.

9. Verbs data structures

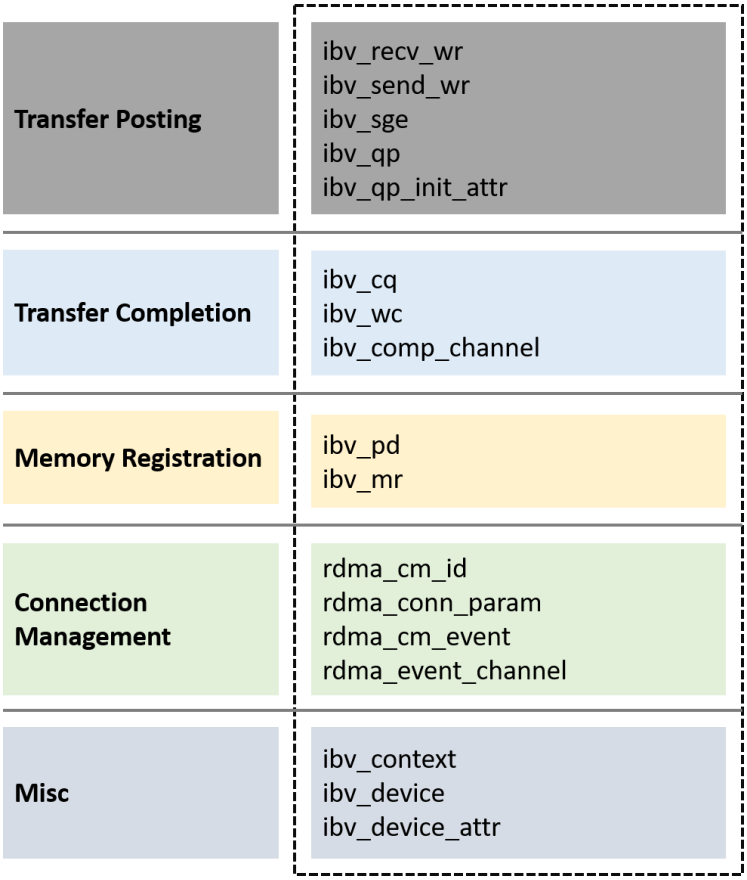

Figure 12 Verbs component data structures

ibv_recv_wr:

tell channel adapter where in virtual memory to put data It receives.

must fill these fields before calling ibv_post_recv()

next : pointer to next RWR(Receive Work Request) in linked list, in practice, most WR lists contain only 1 WR. so this filed usually set to NULL.

wr_id : user_defined id of this RWR

sg_list : array of scatter-gather elements(SGE)

num_sge : no. of elements in sg_list array

ibv_send_wr:

tell channel adapter what data to send.

must fill these fields before calling ibv_post_recv()

next : pointer to next SWR in linked list, in practice, most WR lists contain only 1 WR. so this filed usually set to NULL.

wr_id : user-defined identification of thiw SWR

sg_list : array of scatter-gather elements(SGE)

opcode : IBV_WR_SEND

num_sge : no. of elements in sg_list array

send_flags : IBV_SEND_SIGNALED

ibv_sge

Programmer must fill in these fields before calling ibv_post_recv() or ibv_post_send()

lkey : local memory registration key, used by local Channel Adapter to access MR

must cover all bytes in memory chunk

must belong to protection domain of QP

addr : base address of virtual memory chunk

length : number of bytes in virtual memory chunk

ibv_qp

magor structure to access others

specifies initial values for these fields when QP is creted.

pd : protection domain

recv_cq : recv completion queue

send_cq : send completion queue

qp_context : user_defined id of the QP

ibv_cq

hould information from CA about completed work requests until program “picks them up”

ceq : max number of slots wanted in this CQ.

context : field of associated cm_id.

channel : allows CA to return completion events(NULL ok).

cq_context : user-defined id of this CQ.

ibv_wc

network adapter returns status information about a work request. interrogates these fields after ibv_poll_cq() returns

wr_id : copy of user-defined wr_id from WR

status : code for sucess or failure of WR

opcode : code for type of completed WR(IBV_WC_SEND or IBV_WC_RECV)

byte_len : number of bytes transferred by WR

ibv_pd

enables QPs to utilize MRs for transfers

context : verbs field of associated cm_id, cm_id created during connection management.

ibv_mr

define MR for RDMA transfers. specifies values for pd, addr, and length, system returns values lkey and rkey.

pd : protection domain

lkey : access key for local CA to use

rkey : access key for remote CA to use

addr : starting virtual address of memory region

lengthe : number of bytes in memory region

rdma_cm_id

cm_id is the unifying data structure for all connections

major structure to tie others together

qp : associated queue pair

ps : port space

verbs : interface to driver and CA

channel : for delivery of connection events to program

ibv_qp_init_attr

fill all fields before passing this structure as parameter to rdma_creae_qp()

rdma_conn_param

package set of values needed to initialize a new connection. fill in values for all fields before passing this structure as a parameter to a rdma_connect().

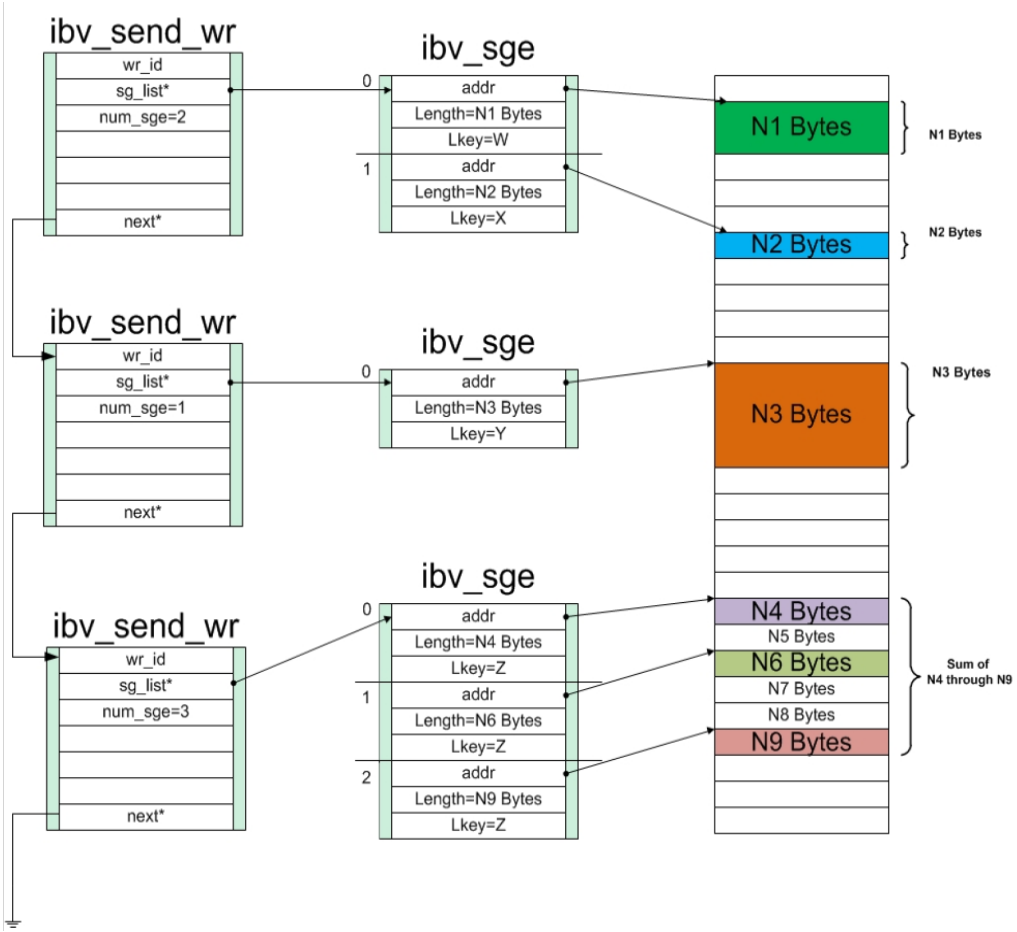

Figure 13 wr relation with sge

Note:

- between posting and completion user memory containing message data is undefined and should not be changed by user program.
- by default. local CA always allowed to read from local registered memory(access code value is 0)

【1】NVMe over Fabrics: Fibre Channel vs. RDMA : https://www.networkcomputing.com/storage/nvme-over-fabrics-fibre-channel-vs-rdma/239739929

【2】Remote Direct Memory Access (RDMA) : https://searchstorage.techtarget.com/definition/Remote-Direct-Memory-Access

【3】Introduction to Remote Direct Memory Access (RDMA) : http://www.rdmamojo.com/2014/03/31/remote-direct-memory-access-rdma/

【4】http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

【5】RDMA Programming - Base on linux-rdma : http://hustcat.github.io/rdma-programming/

【6】RDMAmojo : https://www.rdmamojo.com/page/9/

【7】github-RDMA-Tutorial : https://github.com/jcxue/RDMA-Tutorial/wiki

【8】https://blog.zhaw.ch/icclab/infiniband-an-introduction-simple-ib-verbs-program-with-rdma-write/

NVDIMM : Non-volatile dual in-line memory module

iWARP : internet Wide Area RDMA Protocol

RoCE : RDMA over Converged Ethernet

OFED : OpenFabrics Enterprise Distribution

PD : Protection Domains

WR: Work Request, a data structure that describe a piece of work to be completed, an application posts a WR to a queue.

WQE: Work Queue Element,(wookie)

CQE: Completion Queue Element,(cookie)

CQ : Completion Queues

QP : Queue-Pairs, send queue + receive queue, an application can create many QPs, each QP is associated with exactly one application. play the role of socket “fd” in data transfer operations.

SRQ : Shared Receive Queues

AH : Address Handles

MR : Memory Regions

VA : Virtual Address

ULPs : Upper Layer Protocols. allow a standard application to execute over an RDMA network

HCA : Host Channel Adapter. definition for iWARP.

RNIC : RDMA Network Interface Card. definition for RoCE.

CA : Channel Adapter(iWARP->RNIC, InifiBand->HCA, RoCE->'NIC'),an I/O device that allow an application to conduct RDMA operation directly. ~~contain address translation tables, performs the v(virtual address)-p(physical address), p-v translations.~~

CM_ID : unify data structure for all connections, each cm_id has an associated QP used for all transfers on the connection.

```
enum ibv_wc_status {
        IBV_WC_SUCCESS,
        IBV_WC_LOC_LEN_ERR,
        IBV_WC_LOC_QP_OP_ERR,
        IBV_WC_LOC_EEC_OP_ERR,
        IBV_WC_LOC_PROT_ERR,
        IBV_WC_WR_FLUSTH_ERR,
        IBV_WC_MW_BIND_ERR,
        IBV_WC_BAD_RESP_ERR,
        IBV_WC_LOC_ACCESS_ERR,
        IBV_WC_REM_INV_REQ_ERR,
        IBV_WC_REM_ACCESS_ERR,
        IBV_WC_REM_OP_ERR,
        IBV_WC_RETRY_EXC_ERR,
        IBV_WC_RNR_RETRY_EXC_ERR,
        IBV_WC_LOC_RDD_VIOL_ERR,
        IBV_WC_REM_INV_RD_REQ_ERR,
        IBV_WC_REM_ABOUT_ERR,
        IBV_WC_INV_EECN_ERR,
```

IBV_WC_INV_EEC_STATE_ERR,

IBV_WC_FATA_ERR,

INV_WC_RESP_TIMEOUT_ERR,

INV_WC_GENERAL_ERR

}

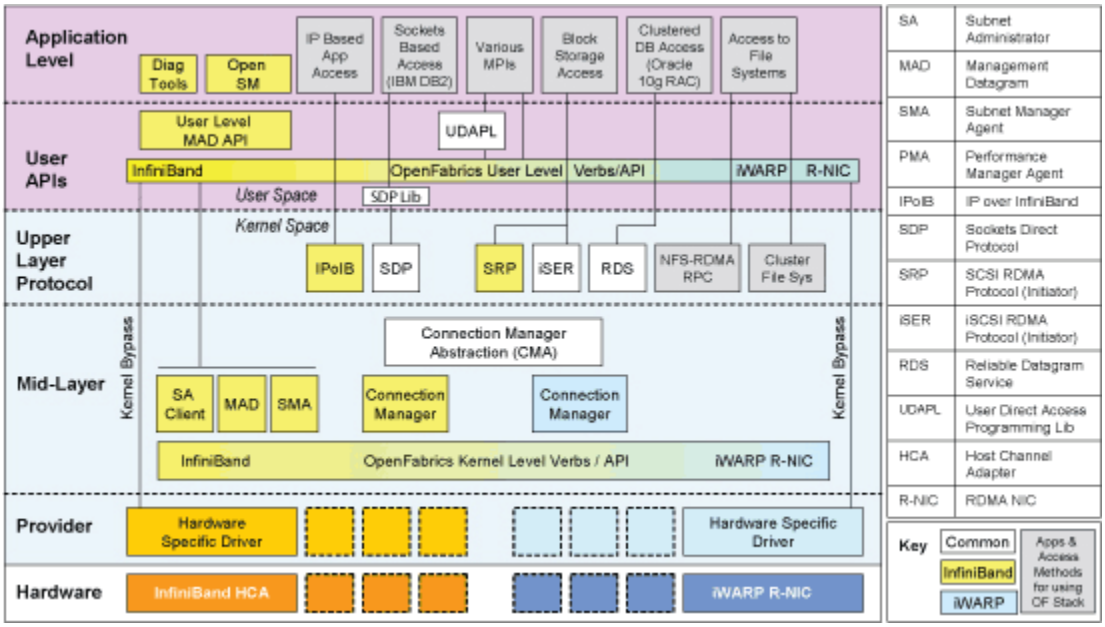

Figure 14 OFED details