Writing Application Programs for RDMA using OFA Software
Part 2

Open Fabrics Alliance

C 2011 OpenFabrics Alliance, Inc

Copyright Statement

Copyright (C) 2016 OpenFabrics Alliance

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

The license itself is at https://www.gnu.org/licenses/fdl-1.3.en.html.

Remote Direct Memory Access

- Goals of RDMA (Remote Direct Memory Access)
  - –High Bandwidth Utilization
  - –Low Latency
  - –Low CPU utilization
- Current technologies that implement RDMA
  - –InfiniBand
  - –iWARP (internet Wide Area RDMA Protocol)
  - –RoCE (RDMA over Converged Ethernet)

Terms for Channel Adapters

Infiniband

– **HCA** Host Channel Adapter

iWARP

– **RNIC** RDMA Network Interface Card

RoCE

– ???

Generic

– **CA** Channel Adapter of any type capable of RDMA

The term used throughout this course

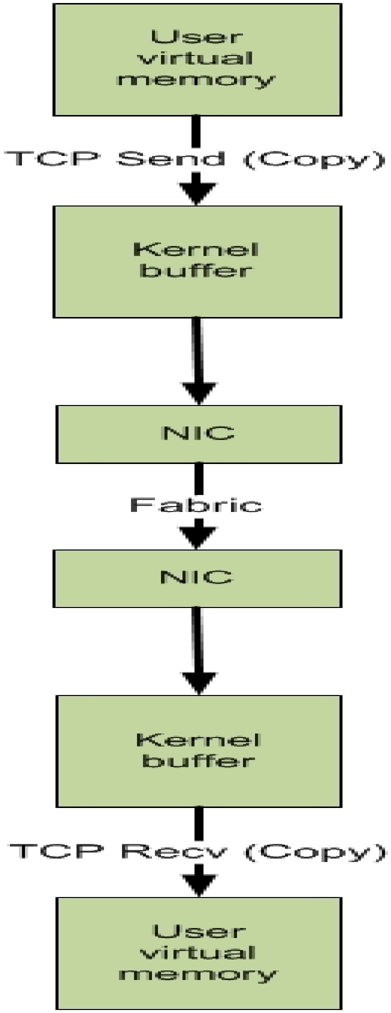
Major RDMA Features

- Zero-copy data transfers
  - Data moves directly from user memory on one side to user memory on the other side
  - No CPU intervention
  - No intermediate buffering
- Kernel by-pass
  - User has direct access to the Channel Adapter
  - No kernel-level protocol handling
  - All protocol handling done on the Channel Adapter

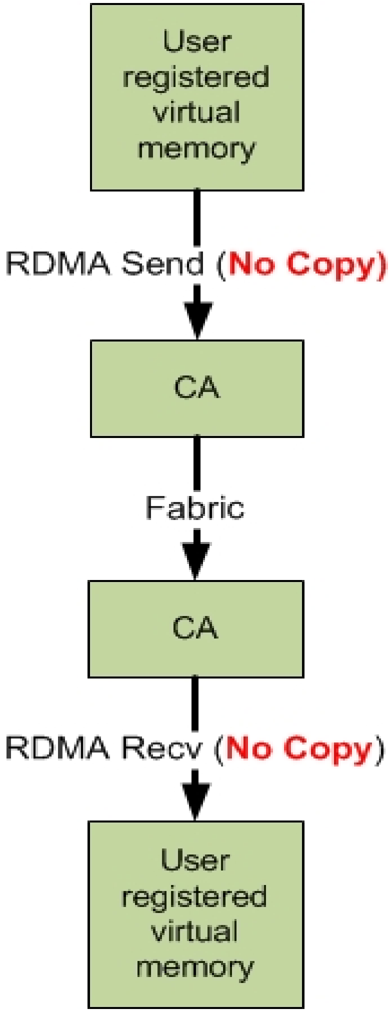
AF_INET compared to RDMA

- Both utilize the client-server model
- Both require a connection for reliable transport
- Both provide a reliable transport mode
  - TCP in AF_INET provides a reliable, in-order **stream of bytes**
  - RDMA_PS_TCP provides a reliable, in-order **sequence of messages**
- TCP uses buffer copying, RDMA uses no buffers

TCP transfer has buffer copies



RDMA transfer has no buffer copy



Important Points

- All RDMA transfers use discrete messages
  - **no** TCP streams!
- We will mostly discuss reliable transport
  - Unreliable transport not implemented for iWARP
  - Unreliable transport used primarily for IB multicast

Client-Server Model

- Server
  - Listener
    - Waits for connections from clients
  - Agent
    - Transfers data with one client
- Client
  - Connects to a server's listener
  - Transfers data with a server's agent

Client-Server Analogy

- Server (Business Enterprise)

–Listener (Call Center)

- Create connection to listen for clients (switchboard)
- Advertise DNS-name, port (1-800 number)
- Loop forever

– Listen for new connection (call) from client (customer)

– Hand off new connection (call) to agent (customer rep)

–Agent (Customer Rep)

- Accept connection (call) from listener (call center)
- Transfer information (talk) with client (customer)
- Close (hang up) the connection

Client-Server Analogy continued

- Client (Customer)
  - Find out DNS-name, port (1-800 number)
  - Create connection (dial 1-800 number)
  - Transfer information (talk) with agent (customer rep)
  - Close (hang up) the connection

C 2011 OpenFabrics Alliance, Inc

OFA API

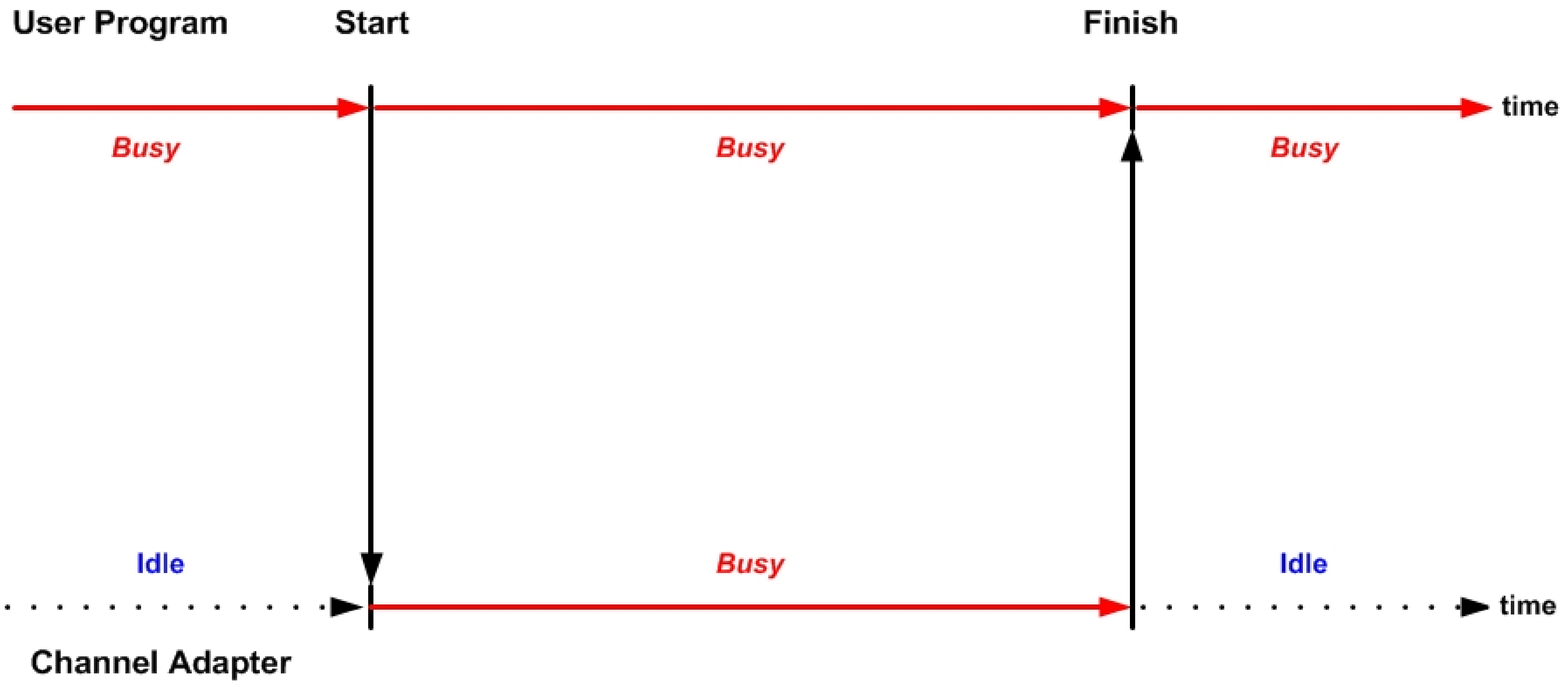
- Some general similarities with sockets
- Many new concepts for sockets programmers
  - Verbs (in OFA API is just another name for functions)
  - Protection Domains
  - Memory Registration
  - Connection Management
  - Explicit Queue Manipulation
  - Explicit Event Handling
  - Explicit Asynchronous Operation
  - Explicit Manipulation of System Data Structures

C 2011 OpenFabrics Alliance, Inc

OFA API Programming Style

- (Almost all) operations to CA are asynchronous
  - Program calls a function to start the operation
    - Function conveys request to CA and returns at once
  - CA operation proceeds in parallel with program
  - CA "informs" program when operation has finished
    - CA conveys status reply about operation to program
- Many applications use threads to handle parallelism
- Many system-level structures are visible
- Many new terms, acronyms and abbreviations

Asynchronous CA operation



Asynchronous Data Transfer

•Posting

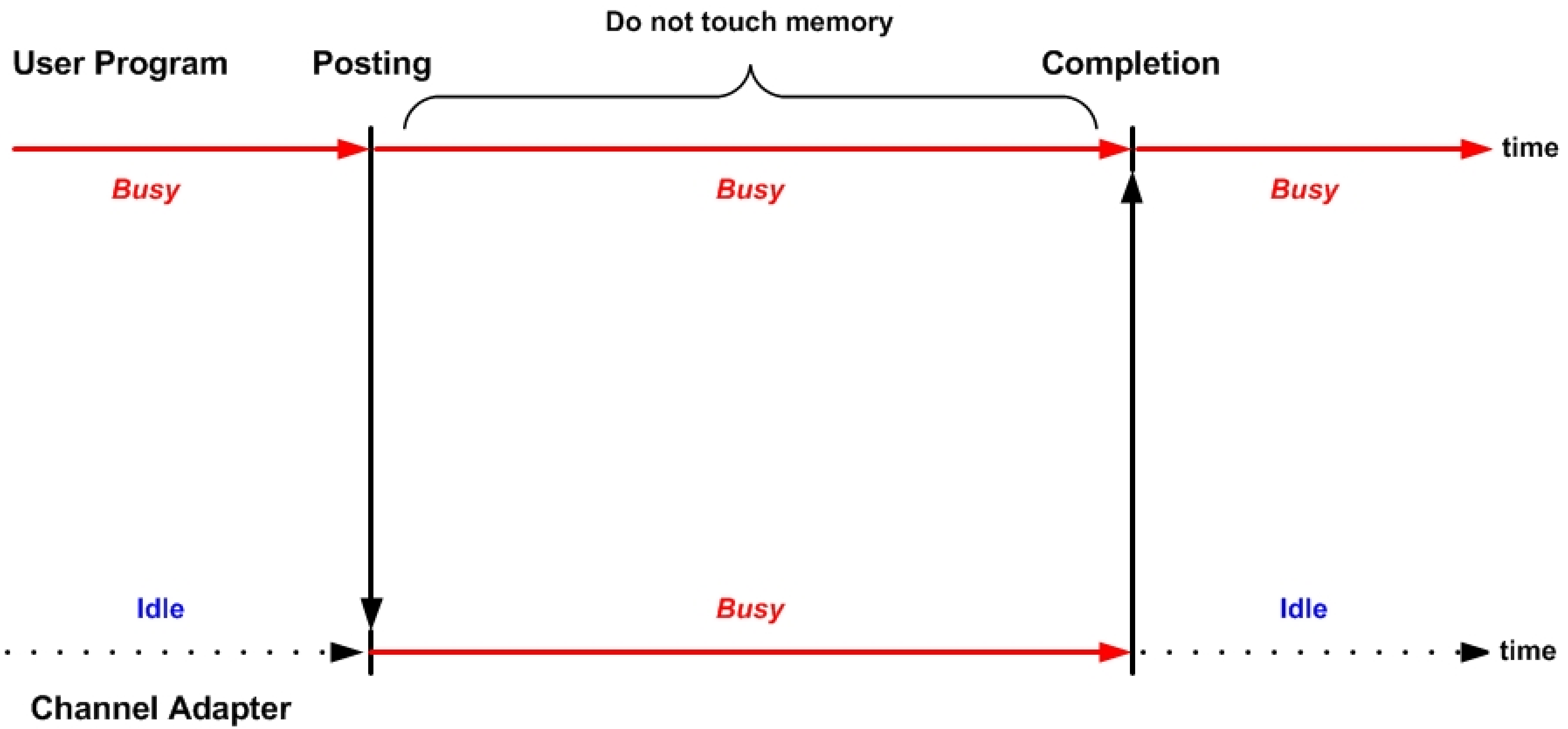
–Term used to initiate data transfer operation's start

•Completion

–Term used to ascertain data transfer operation's end

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

Posting – Completion



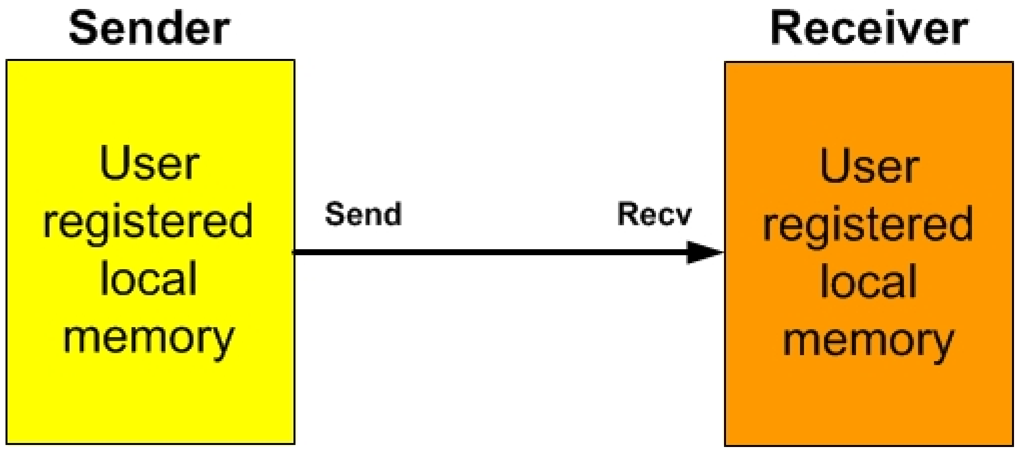
OFA Data Transfer Types

- Send/Receive – similar to “normal” sockets
- RDMA_WRITE – only in OFA
- RDMA_READ – only in OFA
- Atomics – only in IB, optional to implement

Discuss Send/Receive first in order to introduce basic concepts

- Same verbs and data structures used by all types

Send/Recv data transfer



Major components of program

1. Transfer Posting
2. Transfer Completion
3. Memory Registration
4. Connection Management
5. Miscellaneous

- Discuss these in top-down order

C 2011 OpenFabrics Alliance, Inc

Each component must consider

- Purpose
- Data structures
- Verbs to setup the data structures
- Verbs to utilize the data structures
- Verbs to tear-down the data structures
- Relationships with other data structures

Component Pyramid of verbs

- Diagram showing pyramid with all verbs in each component at one level having 3 parts (columns):
  - Setup
  - Use
  - Break-Down
- Each part will contain the relevant verbs
- Serves as a road map for where we have been and what follows
- As introduce new verbs will highlight them in map

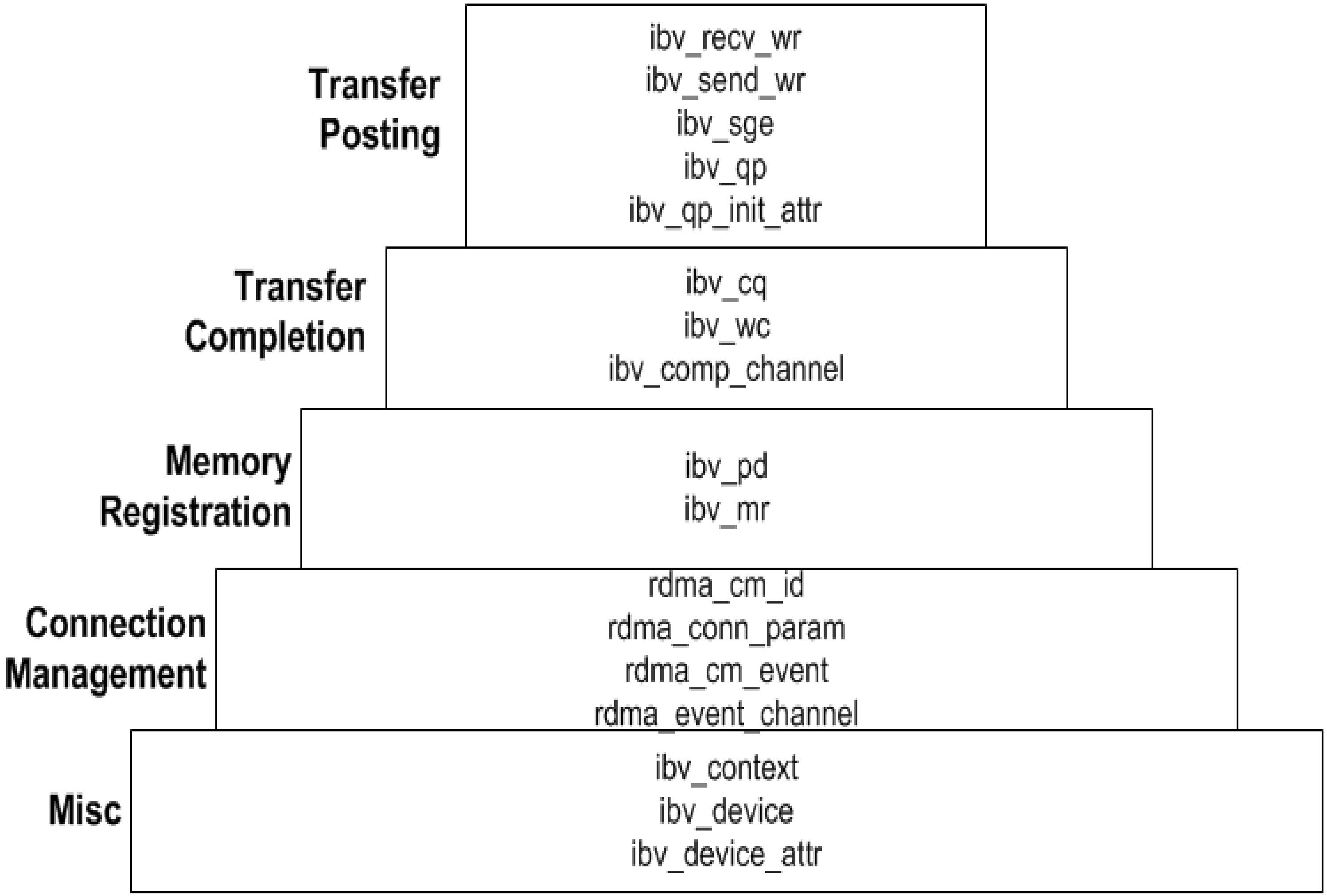
	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

C 2011 OpenFabrics Alliance, Inc

Pyramid of data structures

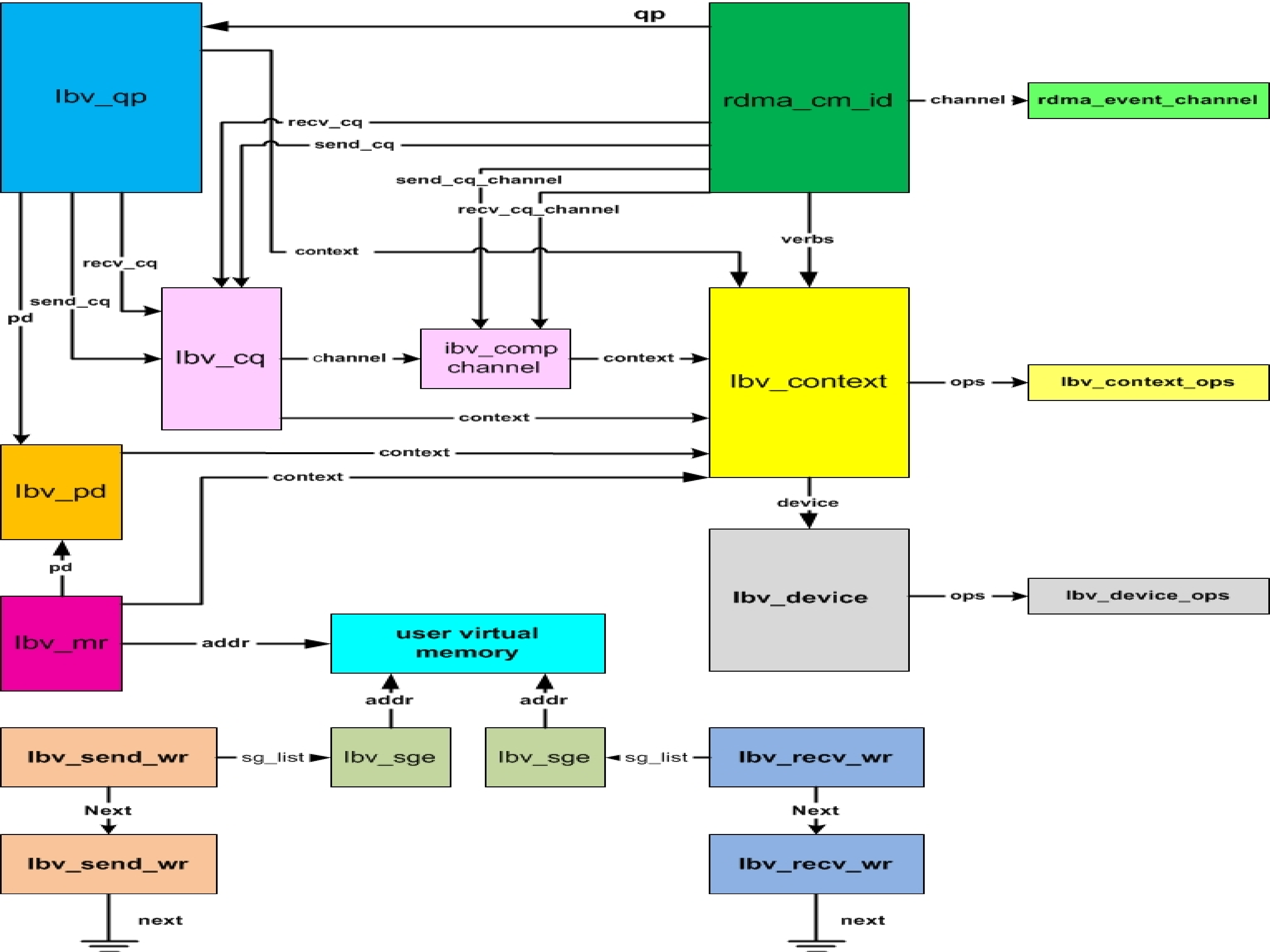
- Diagram showing pyramid of each component as a level
- Each level will contain the relevant data structures set up and broken down by the verbs in that component
- Serves as road map for where we have been and what follows
- As introduce new data structures will highlight them in this map

Pyramid of Data structures



Data Structure Relationships

- Chart showing all major OFA data structures and their interrelationship
- Serves as a road map for where we have been and what follows



qp
Ibv_qp
rdma_cm_id
channel
rdma_event_channel
recv_cq
send_cq
send_cq_channel
recv_cq_channel
context
verbs
recv_cq
send_cq
pd
Ibv_cq
channel
ibv_comp channel
context
Ibv_context
ops
Ibv_context_ops
context
context
Ibv_pd
context
device
pd
Ibv_device
ops
Ibv_device_ops
Ibv_mr
addr
user virtual memory
addr
addr
Ibv_send_wr
sg_list
Ibv_sge
Ibv_sge
sg_list
Ibv_recv_wr
Next
Next
Ibv_send_wr
Ibv_recv_wr
next
next

Approach in this presentation

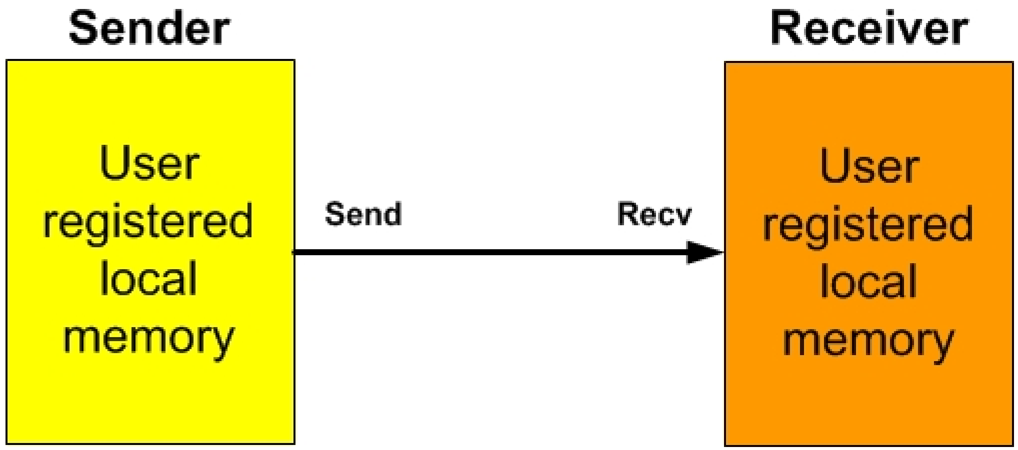
- Start with simple concepts and examples, then elaborate later
- Top-down introduction using the pyramids
  - Major concepts
  - How programmers view finished program
  - Data transfer (at top) is most interesting, important
- Discussion of components
- Bottom-up construction using the pyramids
  - How programmers use the API to construct programs
- Deal with API components relevant to all RDMA technologies, so same program runs on them all

Working code

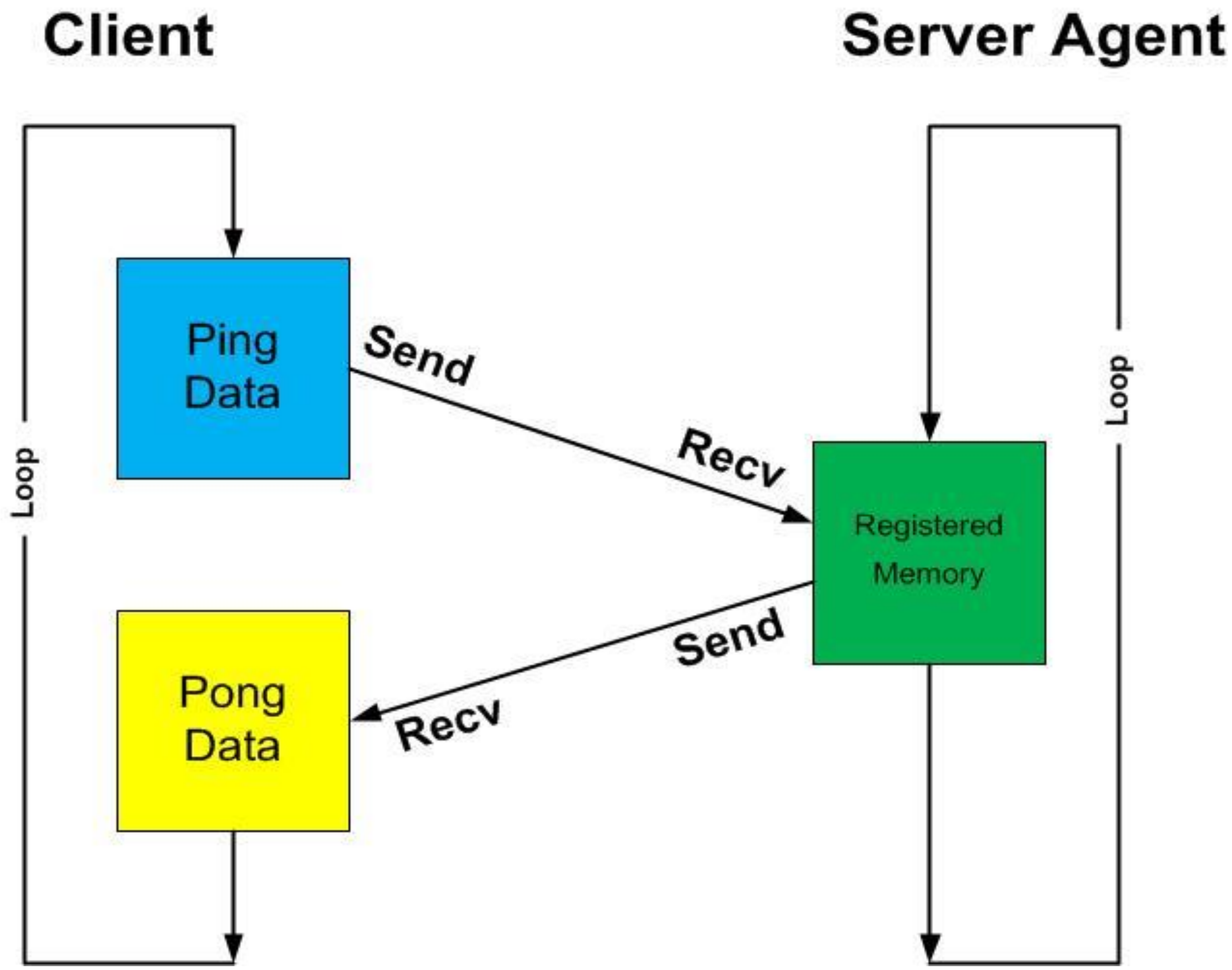
- All concepts are illustrated with working code that runs unchanged on all RDMA technologies
- Code written in C for basic examples:
  - Ping-pong
  - Blast
  - Publish-Subscribe
- Basic code is elaborated as new concepts and techniques are introduced
- Goal is to write portable, reusable code

C 2011 OpenFabrics Alliance, Inc

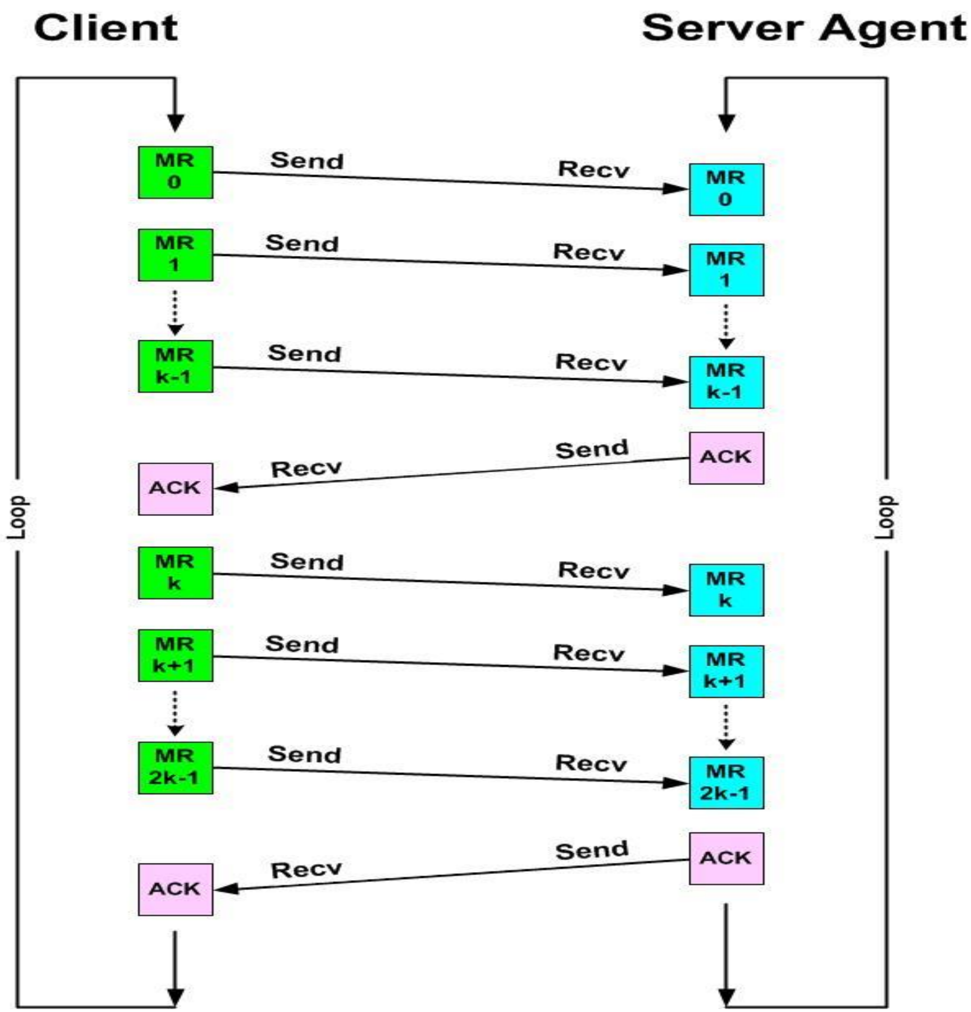
Send/Recv data transfer



Ping-Pong using Send/Recv



Blast using Send/Recv



Major components of Program

1. **Transfer Posting**
2. Transfer Completion
3. Memory Registration
4. Connection Management
5. Miscellaneous

Discuss these in top-down order

C 2011 OpenFabrics Alliance, Inc

1. Transfer Posting

The mechanism by which a program gives the CA all information necessary to start data transfer over RDMA

Send/Recv Similarities with Sockets

- Client must establish a connection with Server prior to any data transfer
- Sender does not know remote receiver’s virtual memory location
- Receiver does not know remote sender’s virtual memory location
- Each **send** of a message must match a corresponding **recv** of that message

(Both sides actively participate in the transfer)

Send/Recv Differences with Sockets

- "normal" socket transfers are buffered (the order of **send** relative to **recv** is irrelevant)
- OFA transfers are not buffered (the **recv** MUST be posted BEFORE the **send**)
- When using OFA, virtual memory on each side MUST be registered on that side ("normal" sockets have no notion of registered memory)
- OFA transfers operate asynchronously ("normal" socket transfers operate synchronously)

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	**Ibv_post_recv** **Ibv_post_send**	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

Posting to receive data

- Verb: **ibv_post_recv()**
- Parameters:
  - Queue Pair - QP
  - 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

Posting to send data

- Verb: **ibv_post_send()**
- Parameters:
  - –Queue Pair - QP
  - –Pointer to linked list of Send Work Requests – SWR
  - –Pointer to bad SWR in list in case of error
- Return value:

    == 0 all SWRs successfully added to send queue (SQ)

    != 0 error code

ibv_post_recv() code snippet

```
int
our_post_recv(struct our_control  *conn,  struct ibv_recv_wr  *recv_work_request,
                 struct our_options  *options)
{
struct ibv_recv_wr      *bad_wr;
            int          ret;
errno = 0;
ret = ibv_post_recv(conn->queue_pair,  recv_work_request,  &bad_wr);
if (ret != 0) {
      if (our_report_wc_status(ret,  "ibv_post_recv",  options)  !=  0)  {
            our_report_error(ret,  "ibv_post_recv",  options);
      }
}
return ret;
}     /* our_post_recv */
```

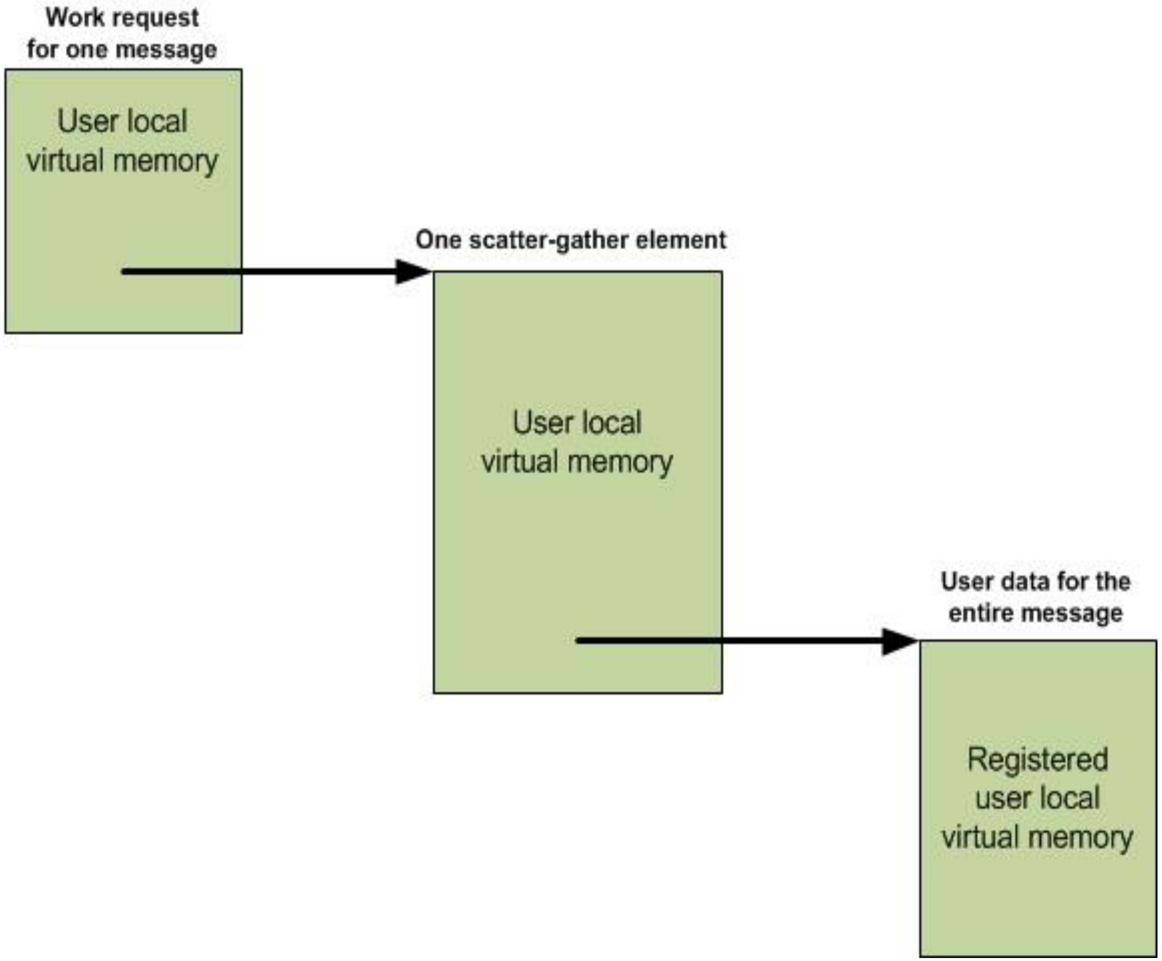
ibv_post_send() code snippet

```
int
our_post_send(struct our_control  *conn, struct ibv_send_wr  *send_work_request,
              struct our_options  *options)
{
struct ibv_send_wr      *bad_wr;
int   ret;

errno = 0;
ret = ibv_post_send(conn->queue_pair,  send_work_request,  &bad_wr);
If (ret != 0) {
      if (our_report_wc_status(ret,  "ibv_post_send",  options) != 0)  {
            our_report_error(ret,  "ibv_post_send",  options);
      }
}
return ret;
}     /* our_post_send */
```

Work Requests in DS pyramid

Transfer Posting	ibv_recv_wr ibv_send_wr ibv_sge ibv_qp ibv_qp_init_attr
Transfer Completion	ibv_cq ibv_wc ibv_comp_channel
Memory Registration	ibv_pd ibv_mr
Connection Management	rdma_cm_id rdma_conn_param rdma_cm_event rdma_event_channel
Misc	ibv_context ibv_device ibv_device_attr

Simplest Work Request construct



Receive Work Request (RWR)

- Purpose: tell channel adapter where in virtual memory to put data it receives
- Data structure: **struct ibv_recv_wr**
- Fields visible to programmer:

next	pointer to next RWR in linked list
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

- Programmer must fill in these fields before calling **ibv_post_recv()**

Send Work Request (SWR)

- Purpose: tell channel adapter what data to send
- Data structure: **struct ibv_send_wr**
- Fields visible to programmer:

next	pointer to next SWR in linked list
wr_id	user-defined identification of this SWR
sg_list	array of scatter-gather elements (SGE)
opcode	**IBV_WR_SEND**
num_sge	number of elements in **sg_list** array
send_flags	**IBV_SEND_SIGNALED**

- Programmer must fill in these fields before calling **ibv_post_send()**

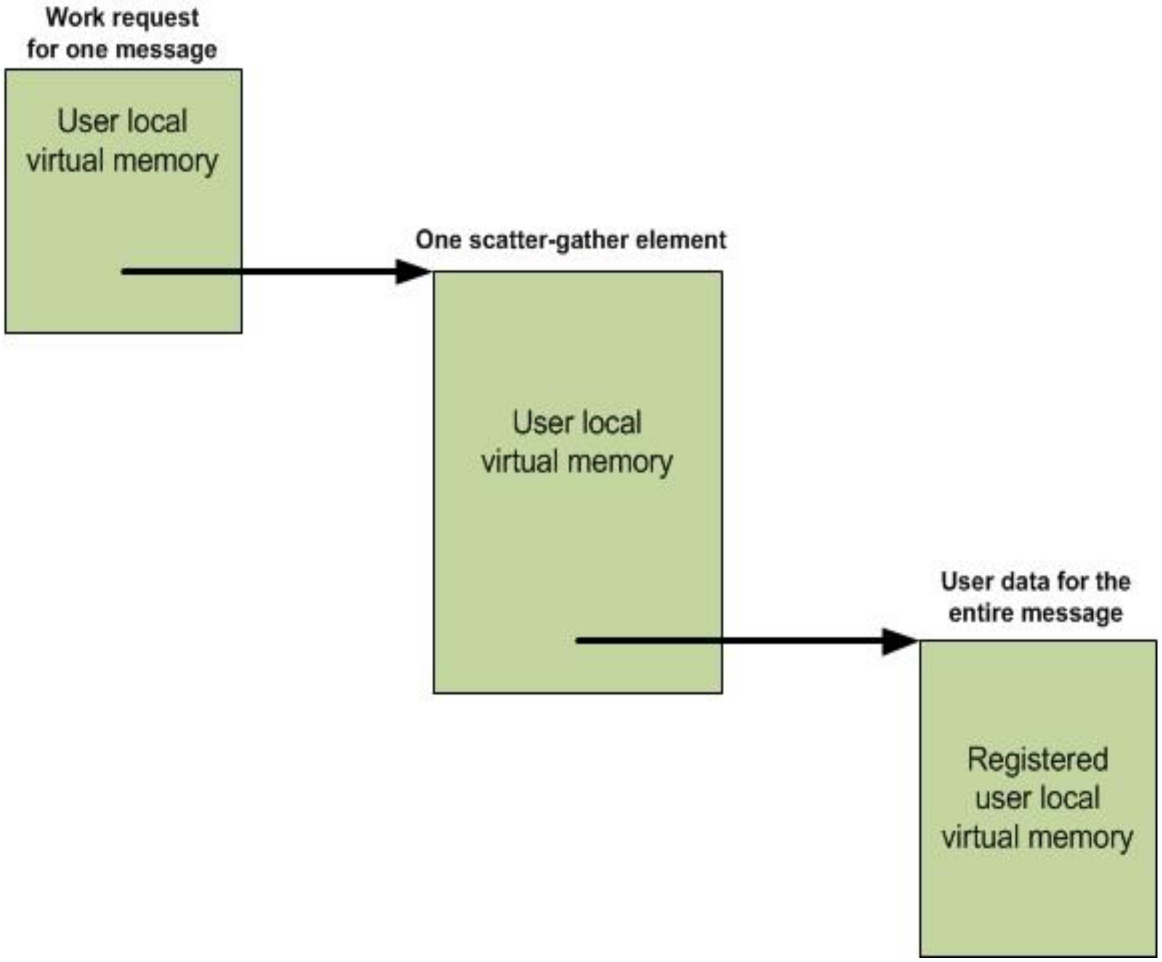
Work Request List

- Allows one **ibv_post_recv()** or one **ibv_post_send()** to convey multiple transfer operations to the CA
- Each WR in list is independent
- Each WR in list generates its own completion
- All WRs in list must be same type – Recv or Send

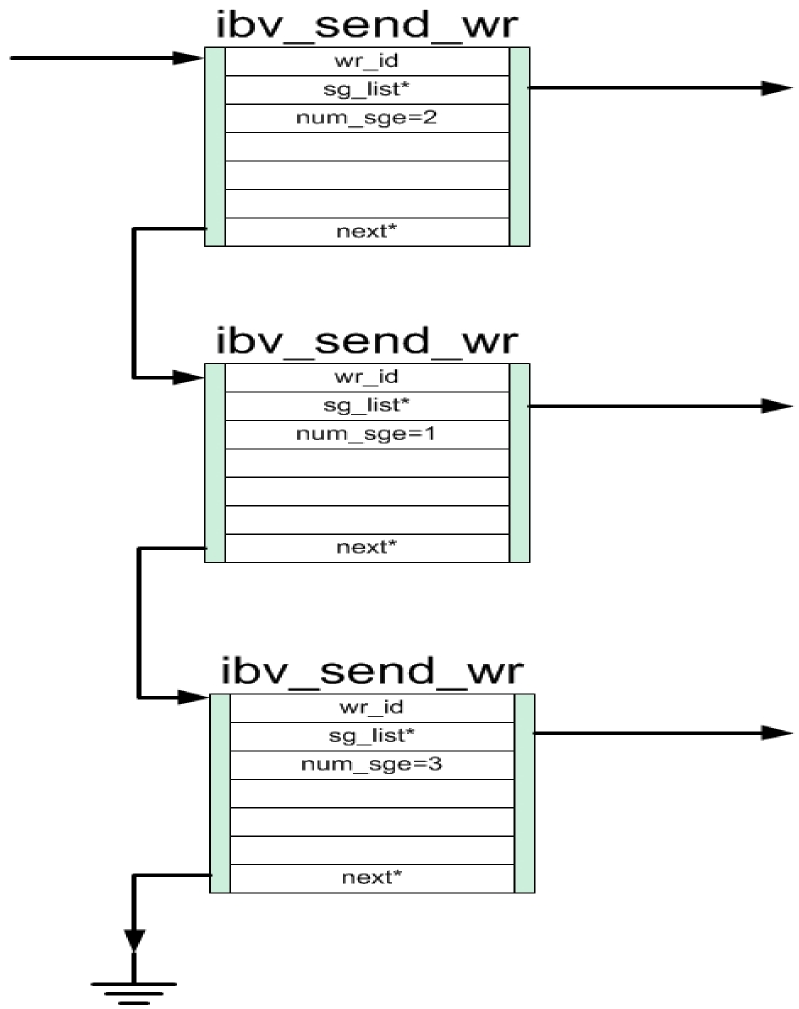
- In practice, most WR lists contain only 1 WR

C 2011 OpenFabrics Alliance, Inc

Simplest Work Request construct



Send work request list



our_setup_recv_wr() code snippet

```
static void
our_setup_recv_wr(struct our_control  *conn,  struct ibv_sge  *sg_list,
                int  n_sges,  struct ibv_recv_wr  *recv_work_request)
{
/* set the user's identification to be pointer to itself */
recv_work_request->wr_id = (uint64_t)recv_work_request;

/* not chaining this work request to other work requests */
recv_work_request->next = NULL;

/* point at array of scatter-gather elements for this recv */
recv_work_request->sg_list = sg_list;

/* number of scatter-gather elements in array actually being used */
recv_work_request->num_sge = n_sges;
}        /* our_setup_recv_wr */
```

our_setup_send_wr() code snippet

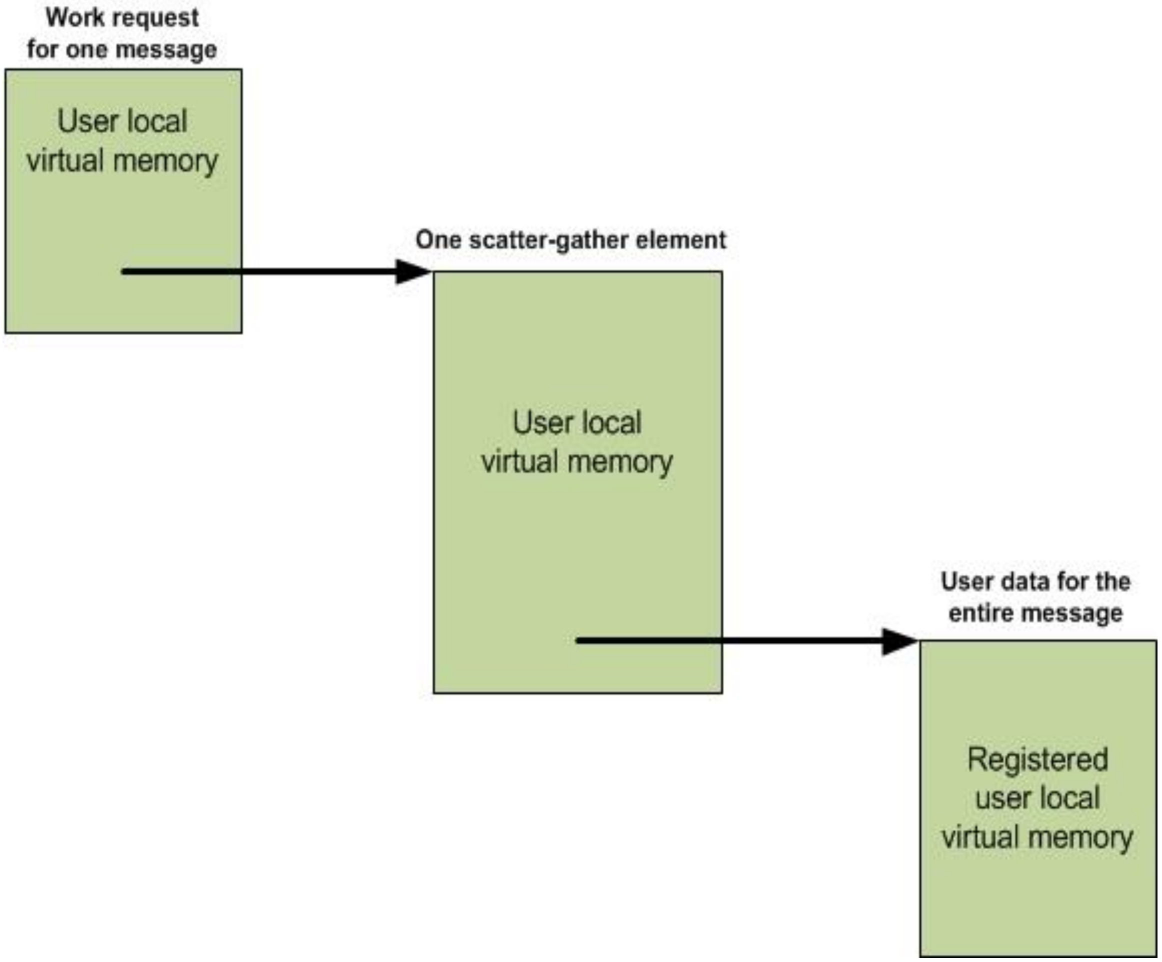
```
static void
our_setup_send_wr(struct our_control  *conn,  struct ibv_sge  *sg_list,
                enum ibv_wr_opcode  opcode,  int  n_sges,
                struct ibv_send_wr  *send_work_request)
{
/* set the user's identification to be pointer to itself */
send_work_request->wr_id = (uint64_t)send_work_request;

/* not chaining this work request to other work requests */
send_work_request->next = NULL;

/* point at array of scatter-gather elements for this send */
send_work_request->sg_list = sg_list;

/* number of scatter-gather elements in array actually being used */
send_work_request->num_sge = n_sges;

/* the type of send */
send_work_request->opcode = opcode;

/* set SIGNALED flag so every send generates a completion */
send_work_request->send_flags = IBV_SEND_SIGNALED;

/* not sending any immediate data */
send_work_request->imm_data = 0;
}               /* our_setup_send_wr */
```

Scatter-gather lists

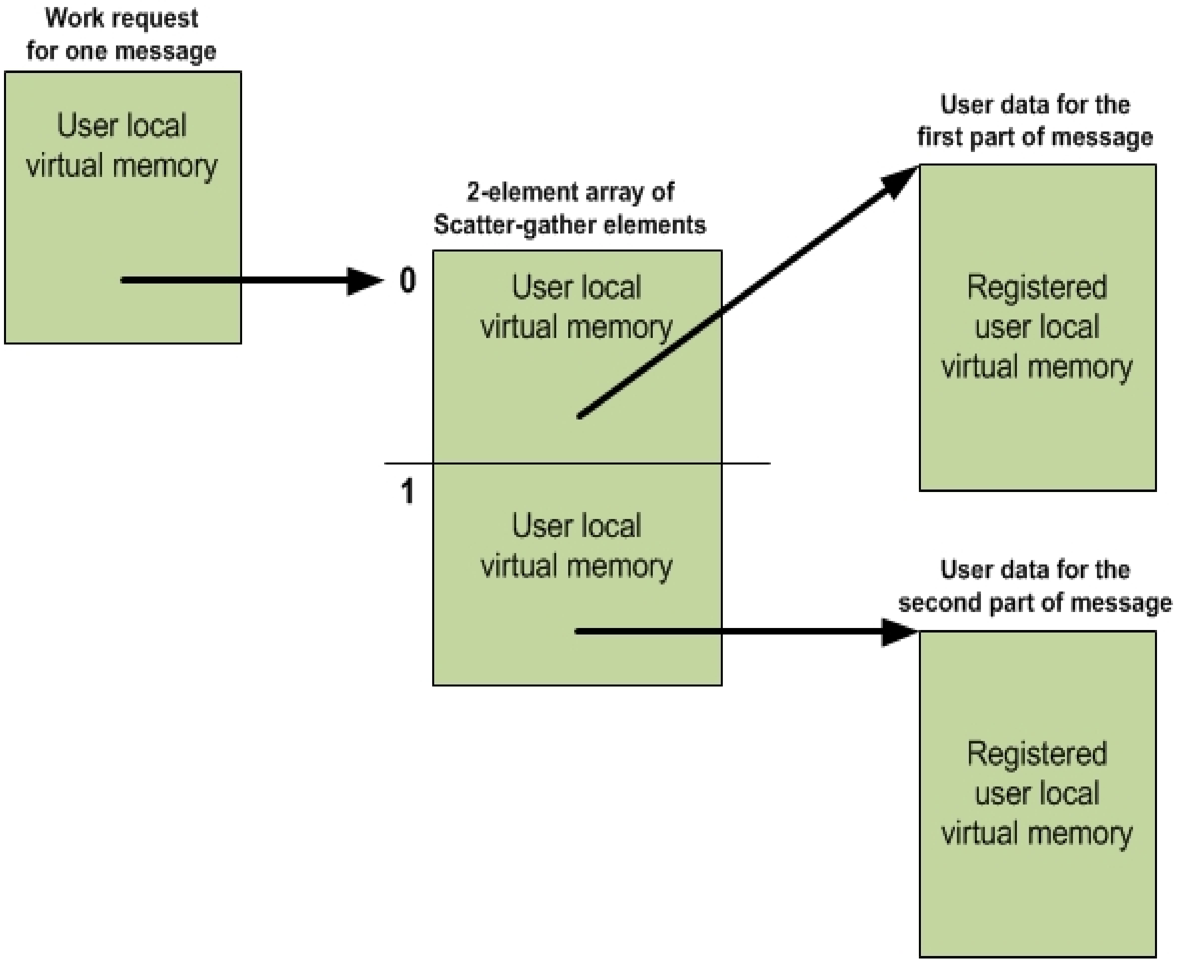
- Each work request points to a scatter-gather list
  - **sg_list** field
- List is array of scatter-gather elements (SGEs)
- Each element in array has type **struct ibv_sge**
- Work request says how many elements in array
  - **num_sge** field

- In practice, most scatter-gather lists contain only 1 SGE

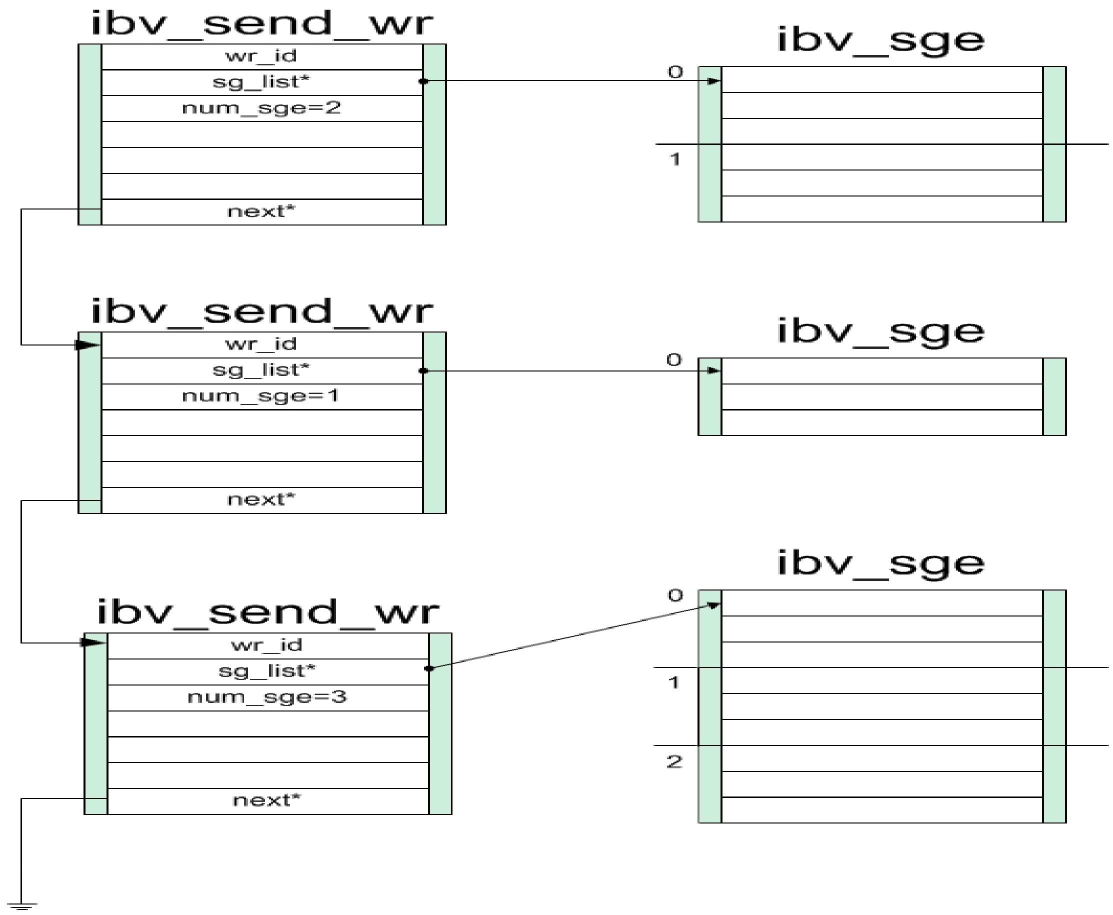
Simplest Work Request construct



One WR with two SGEs



WR list with scatter-gather lists

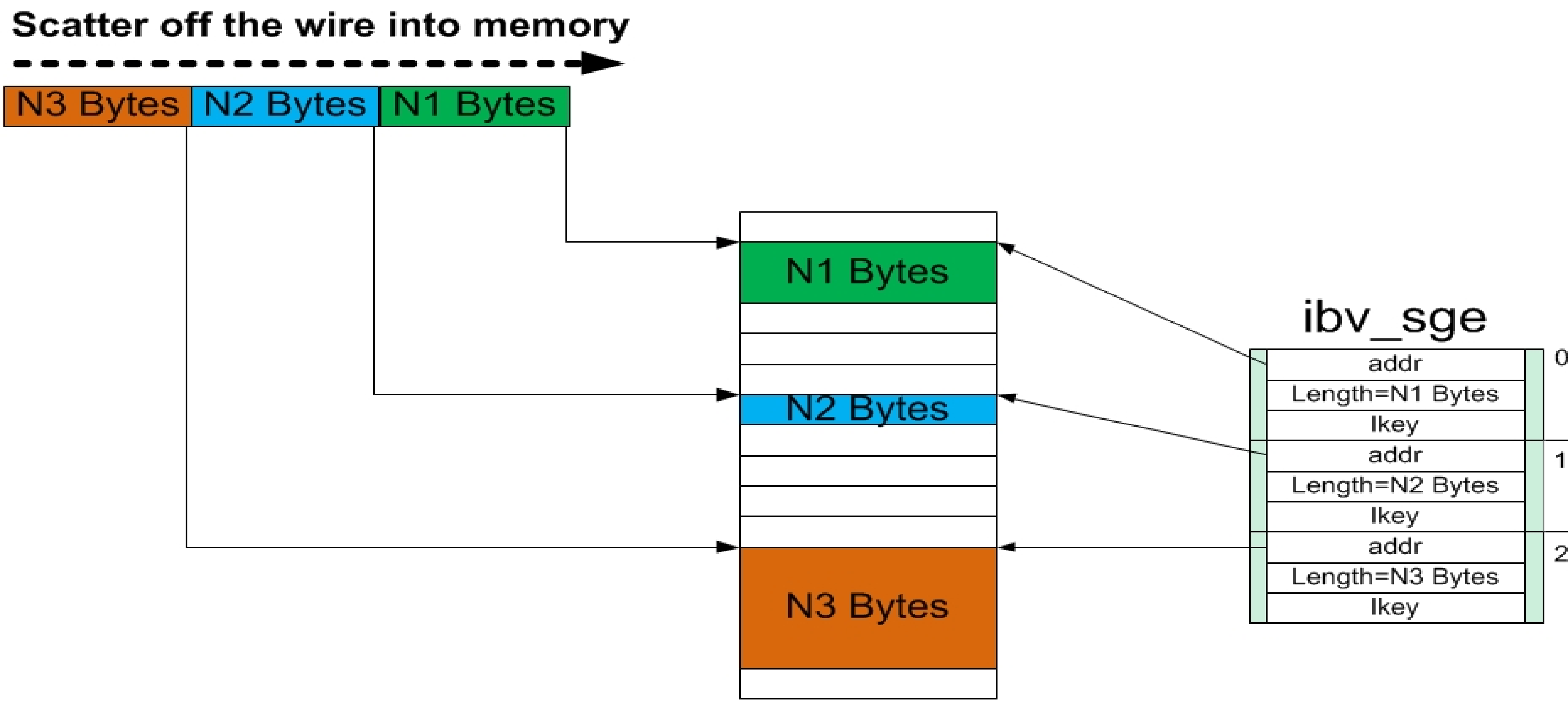


Recv scatter-gather list (array)

- Goal: allows one recv operation to split (i.e., **scatter**) data from single message "on the wire" into different chunks of local virtual memory
- Useful if 1 message contains fixed-length header followed by maximum-length data
  - header will be stored in one area of memory
  - data will be stored in another area of memory
- Each element in the **sg_list** array describes one chunk of virtual memory

- In practice, most **sg_list**s contain only 1 SGE

Scatter during ibv_post_recv()

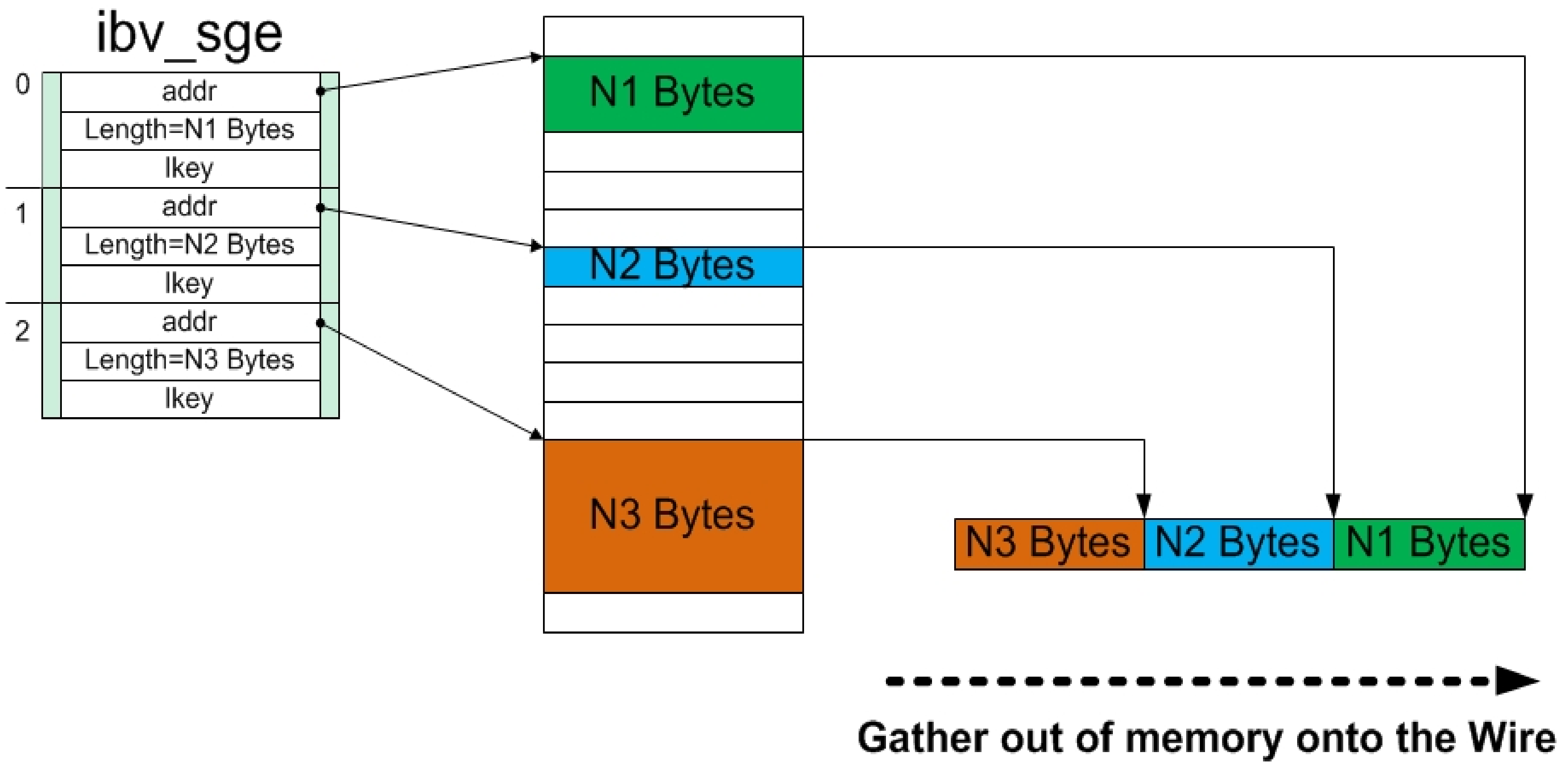


Send scatter-gather list (array)

- Goal: allows one SWR to pull together (i.e., **gather**) data from different chunks of local virtual memory into single message "on the wire"
- Each element in the sg_list array describes one chunk of virtual memory
- Useful if 1 message has fixed-length header followed by variable-length data
  - fixed-length header in one area of memory
  - variable-length data in another area of memory

- In practice, most **sg_list**s have only 1 element

Gather during ibv_post_send()



Scatter-Gather Element (SGE)

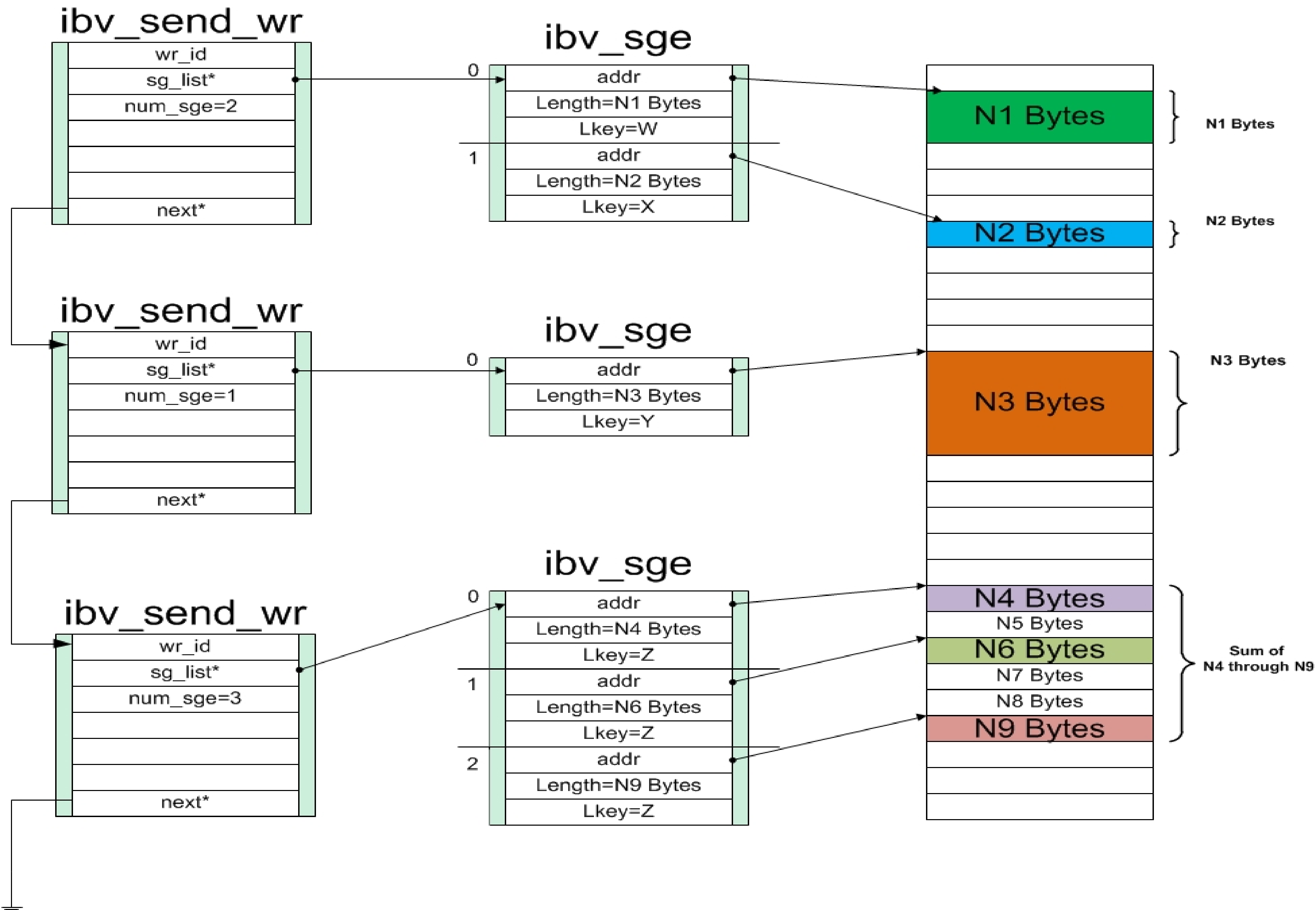
- Purpose: describe chunk of virtual memory for transfer
- Data structure: **struct ibv_sge**
- Fields visible to programmer:

lkey	local memory registration key
	- 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

- Programmer must fill in these fields before calling **ibv_post_recv()** or **bv_post_send()**

our_setup_sge() code snippet

```
/* fill in scatter-gather element for length bytes at registered addr */
static void
our_setup_sge(void  *addr,  unsigned int  length,  unsigned int  lkey,
        struct ibv_sge  *sge)
{
/* point at the memory area */
sge->addr = (uint64_t)(unsigned long)addr;

/* set the number of bytes in that memory area */
sge->length = length;

/* set the registration key for that memory area */
sge->lkey = lkey;
}       /* our_setup_sge */
```


ibv_send_wr
wr_id
sg_list*
num_sge=2
next*
ibv_sge
0
addr
Length=N1 Bytes
Lkey=W
1
addr
Length=N2 Bytes
Lkey=X
N1 Bytes
N1 Bytes
N2 Bytes
N2 Bytes
ibv_send_wr
wr_id
sg_list*
num_sge=1
next*
ibv_sge
0
addr
Length=N3 Bytes
Lkey=Y
N3 Bytes
N3 Bytes
ibv_send_wr
wr_id
sg_list*
num_sge=3
next*
ibv_sge
0
addr
Length=N4 Bytes
Lkey=Z
1
addr
Length=N6 Bytes
Lkey=Z
2
addr
Length=N9 Bytes
Lkey=Z
N4 Bytes
N5 Bytes
N6 Bytes
N7 Bytes
N8 Bytes
N9 Bytes
Sum of
N4 through N9

Queue Pair (QP)

- A unifying data structure for all transfers
- Plays role of socket “fd” in data transfer operations

- Major components:

–Protection domain

Registered memory regions this QP can use in transfers

–Send completion queue

Completed send operations user has not “picked up” yet

–Receive completion queue

Completed receive operations user has not “picked up” yet

C 2011 OpenFabrics Alliance, Inc

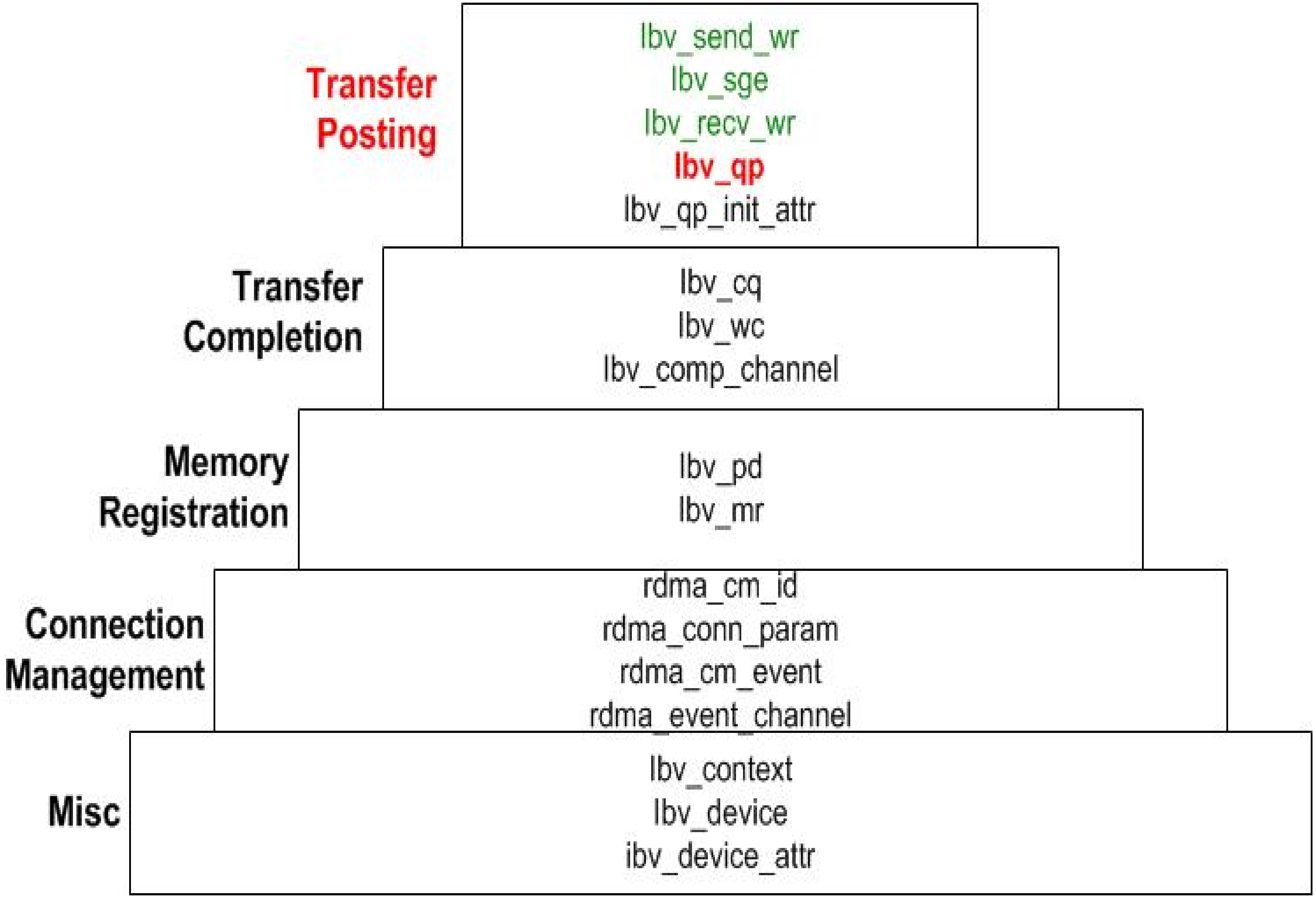
Queue Pair Data Structure

- Purpose – major structure to access others
- Data structure: **struct ibv_qp**
- Fields visible to programmer:

pd	protection domain
recv_cq	recv completion queue
send_cq	send completion queue
qp_context	user-defined id of this QP

- Programmer specifies initial values for these fields when QP is created

QP in data structure pyramid



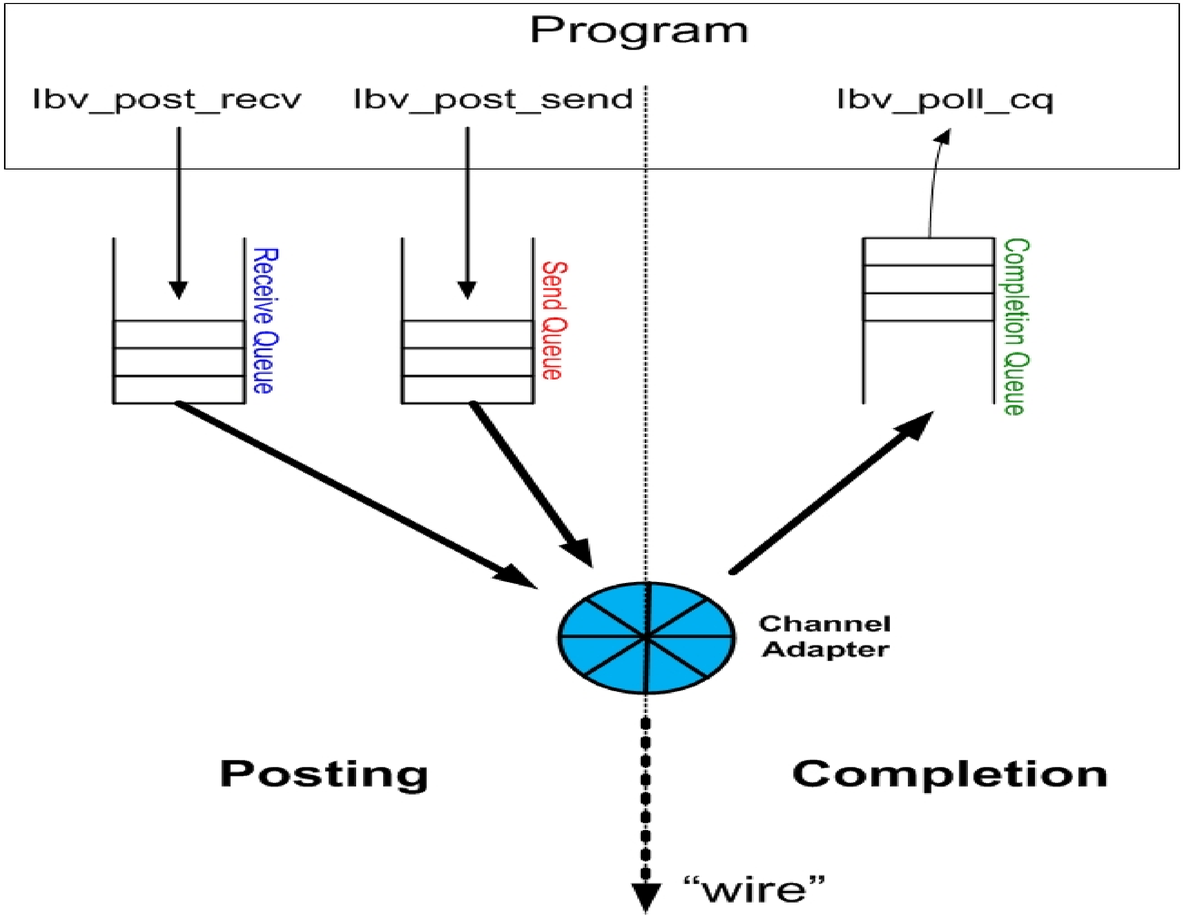
Major components of program

1. Transfer Posting
2. **Transfer Completion**
3. Memory Registration
4. Connection Management
5. Miscellaneous

- Discuss these in top-down order

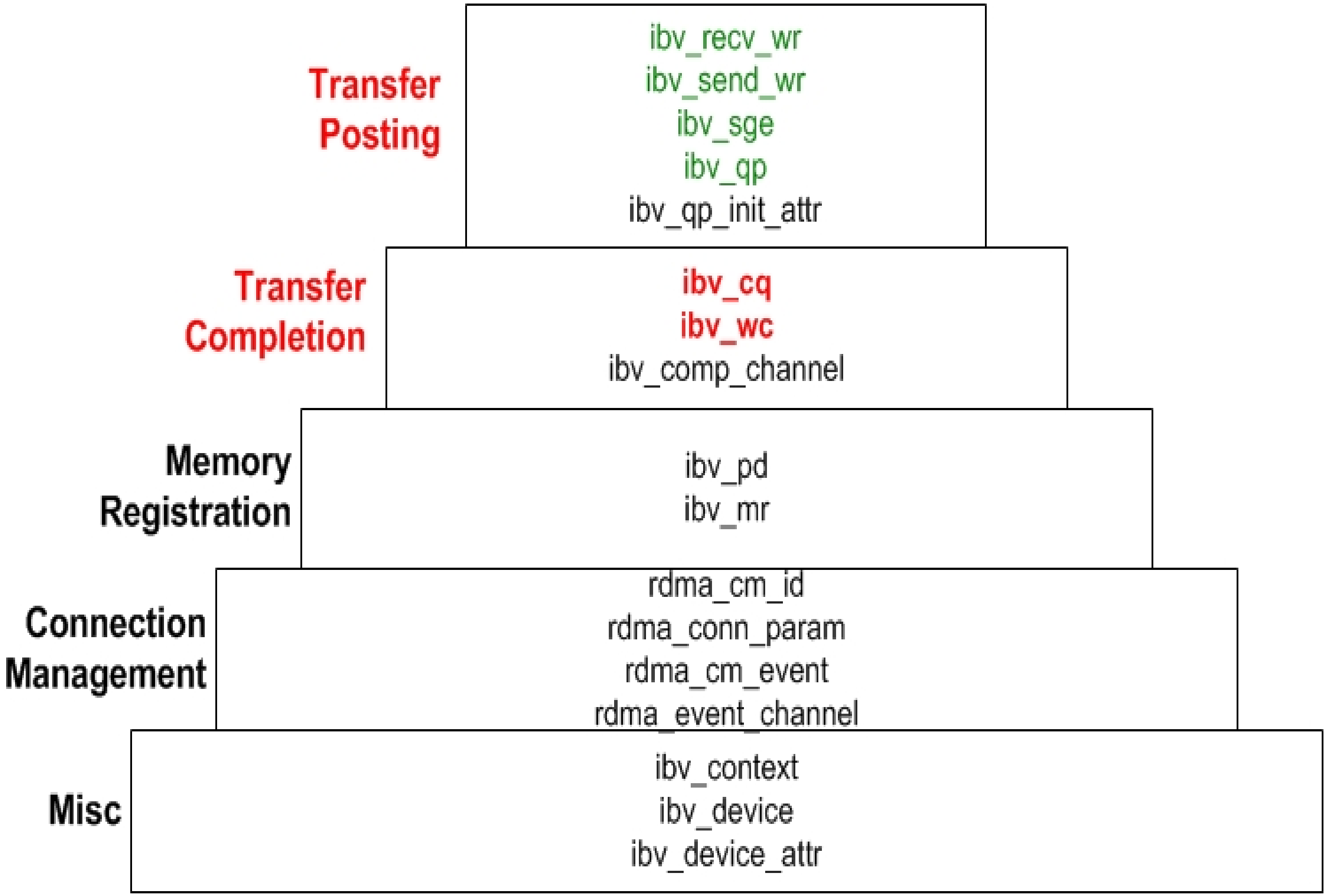
C 2011 OpenFabrics Alliance, Inc

Transfer and completion queues



	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	**ibv_create_cq** ibv_create_comp_channel	**Ibv_poll_cq** **Ibv_wc_status_str** Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	**ibv_destroy_cp** ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

CQ and WC in DS pyramid



2. Transfer Completion

- Purpose: To detect completion of a transfer
- Verb: **ibv_poll_cq()**
- Parameters:
  - Completion Queue - CQ
  - Array of Work Completion slots – WC
  - Number of slots in Work Completion array
- Return value:

  >= 0 number of WC slots filled in

  < 0 error code

C 2011 OpenFabrics Alliance, Inc

Completion Queue (CQ) Data Structure

- Purpose: hold information from CA about completed work requests until program "picks them up"
- Data structure: **struct ibv_cq**
- Fields visible to programmer:

cqe	max number of slots wanted in this CQ
context	verbs field of associated cm_id
channel	allows CA to return completion events (NULL ok)
cq_context	user-defined id of this CQ

- Programmer gives initial values when CQ is created
- System returns value for **cqe** that may be greater than size supplied by programmer

completion queue setup

- Verb: **ibv_create_cq()**
- Parameters:
  - verbs field of associated cm_id
  - desired total number of slots in new queue
  - user-defined identification of this queue
  - completion channel (may be NULL)
  - comp_vector - ???
- Return value:
  - Pointer to new instance of **struct ibv_cq**

completion queue break-down

- Verb: **ibv_destroy_cq()**
- Parameter:
  - Pointer to **struct ibv_cq** returned by **ibv_create_cq()**

our_create_cq() - code snippet

```
static int
our_create_cq(struct our_control  *conn,  struct rdma_cm_id  *cm_id,  struct our_options  *options)
{
int     ret;
errno = 0;
conn->completion_queue = ibv_create_cq(cm_id->verbs,  options->send_queue_depth * 2,
                                                                  conn,  NULL,  0);
if (conn->completion_queue == NULL) {
        ret = ENOMEM;
        our_report_error(ret,  "ibv_create_cq",  options);
        return ret;
}
our_trace_ptr("ibv_create_cq",  "created completion queue",  conn->completion_queue,  options);
our_trace_ulong("ibv_create_cq",  "returned cqe",  conn->completion_queue->cqe,  options);
our_trace_ptr("ibv_create_cq",  "returned completion queue channel",
                                             conn->completion_queue->channel,  options);
return 0;
}       /* our_create_cq */
```

Work Completion (WC)

•Data structure to hold “results” of a posted operation that is now finished

•Space provided by user as parameter to **ibv_poll_cq()**

•CA fills in work completion fields with information taken from first item in completion queue

–Identification of work request that generated this WC

–Status information about why it finished

== 0 successful (IBV_WC_SUCCESS)

!= 0 error code

Work Completion (WC) Data Structure

- Purpose: means by which network adaptor returns status information about a work request
- Data structure: **struct ibv_wc**
- Fields visible to programmer:

wr_id	copy of user-defined wr_id from WR
status	code for success 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

- Programmer interrogates these fields after **ibv_poll_cq()** returns

ibv_poll_cq() code snippet

```
int
our_await_completion(struct our_control *conn,
                    struct ibv_wc *work_completion,
                    struct our_options *options)
{
int         ret;

/* busy wait for next work completion to appear in completion queue */
do    {
        errno = 0;
        ret = ibv_poll_cq(conn->completion_queue,  1,  work_completion);
} while (ret == 0);
if (ret != 1) {
        /* ret cannot be 0,  and should never be > 1,  so must be < 0 */
        our_report_error(ret,  "ibv_poll_cq",  options);
} else {
        ret = our_check_completion_status(conn,  work_completion,  options);
}
return ret;
}       /* our_await_completion */
```

our_check_completion_status()

```
static int
our_check_completion_status(struct our_control *conn,
                            struct ibv_wc *work_completion,
                            struct our_options *options)
{
int     ret;

ret = work_completion->status;
if (ret != 0) {
        if (ret == IBV_WC_WR_FLUSH_ERR) {
                our_report_string( "ibv_poll_cq", "completion status", "flushed", options);
        } else if (our_report_wc_status(ret, "ibv_poll_cq", options)) {
                our_report_ulong("ibv_poll_cq", "completion status", ret, options);
        }
}
return ret;
}       /* our_check_completion_status */
```

C 2011 OpenFabrics Alliance, Inc

our_report_wc_status()

```
/* on entry, ret is known to be != 0
 *
 * Returns == 0 if ibv_wc_status message was printed (ret was valid status code)
 *              != 0 otherwise
 */
int
our_report_wc_status(int ret, const char *verb_name, struct our_options *options)
{
/* ensure that ret is an enum ibv_wc_status value */
if (ret < IBV_WC_SUCCESS || ret > IBV_WC_GENERAL_ERR)
        return ret;

/* print the status error message */
fprintf(stderr, "%s: %s returned status %d %s\n",
                        options->message, verb_name, ret, ibv_wc_status_str(ret));
return 0;
}       /* our_report_wc_status */
```

C 2011 OpenFabrics Alliance, Inc

enum ibv_wc_status – in verbs.h

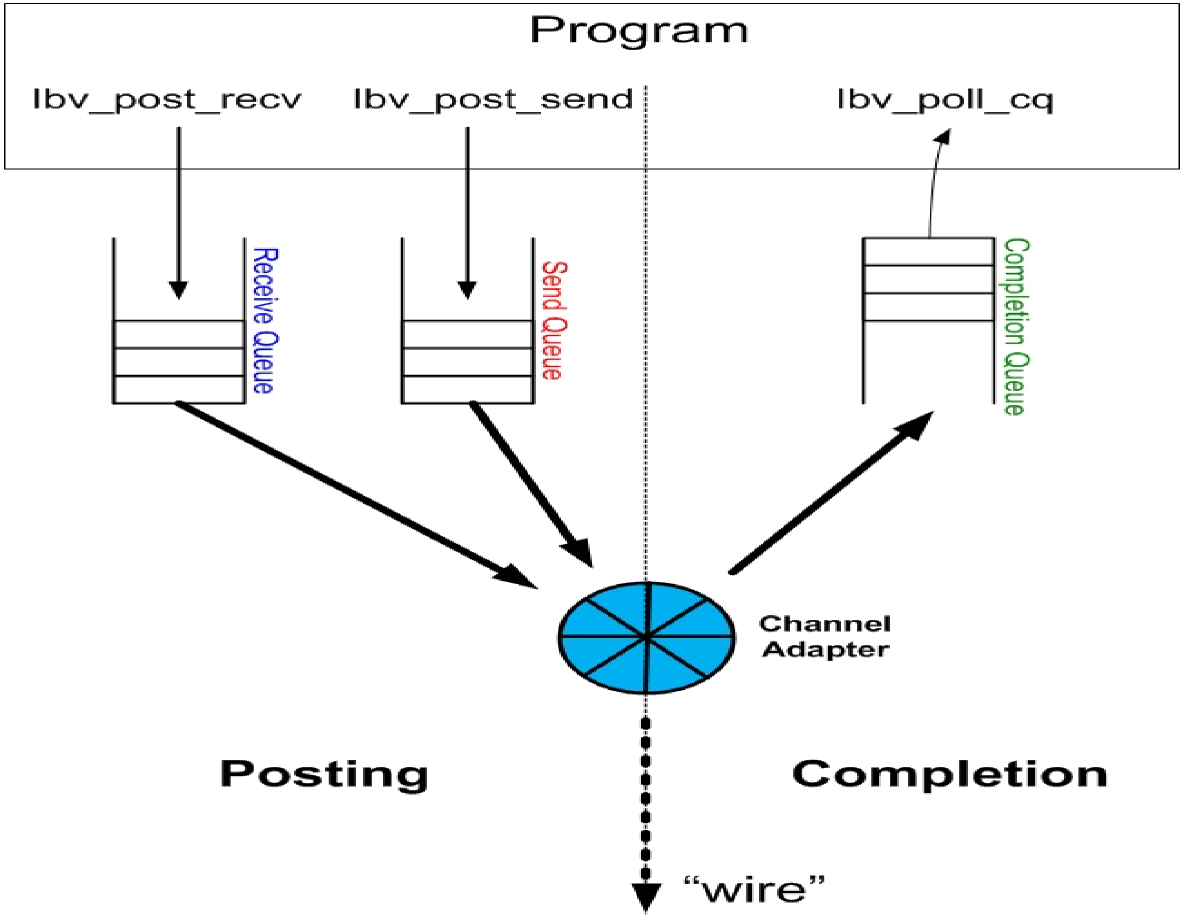
```
enum ibv_wc_status {
IBV_WC_SUCCESS,
IBV_WC_LOC_LEN_ERR,
IBV_WC_LOC_QP_OP_ERR,
IBV_WC_LOC_EEC_OP_ERR,
IBV_WC_LOC_PROT_ERR,
IBV_WC_WR_FLUSH_ERR,
IBV_WC_MW_BIND_ERR,
IBV_WC_BAD_RESP_ERR,
IBV_WC_LOC_ACCESS_ERR,
IBV_WC_REM_INV_REQ_ERR,
IBV_WC_REM_ACCESS_ERR,
IBV_WC_REM_OP_ERR,
IBV_WC_RETRY_EXC_ERR,
IBV_WC_RNR_RETRY_EXC_ERR,
IBV_WC_LOC_RDD_VIOL_ERR,
IBV_WC_REM_INV_RD_REQ_ERR,
IBV_WC_REM_ABORT_ERR,
IBV_WC_INV_EECN_ERR,
IBV_WC_INV_EEC_STATE_ERR,
IBV_WC_FATAL_ERR,
IBV_WC_RESP_TIMEOUT_ERR,
IBV_WC_GENERAL_ERR
};
```

That does it – for data transfer

- Transfer initiation
  - ibv_post_recv()
  - ibv_post_send()
  - Work Request – RWR, SWR
  - Scatter-Gather Element – SGE
  - Queue Pair - QP
- Transfer completion
  - ibv_poll_cq()
  - Completion Queue - CQ
  - Work Completion - WC

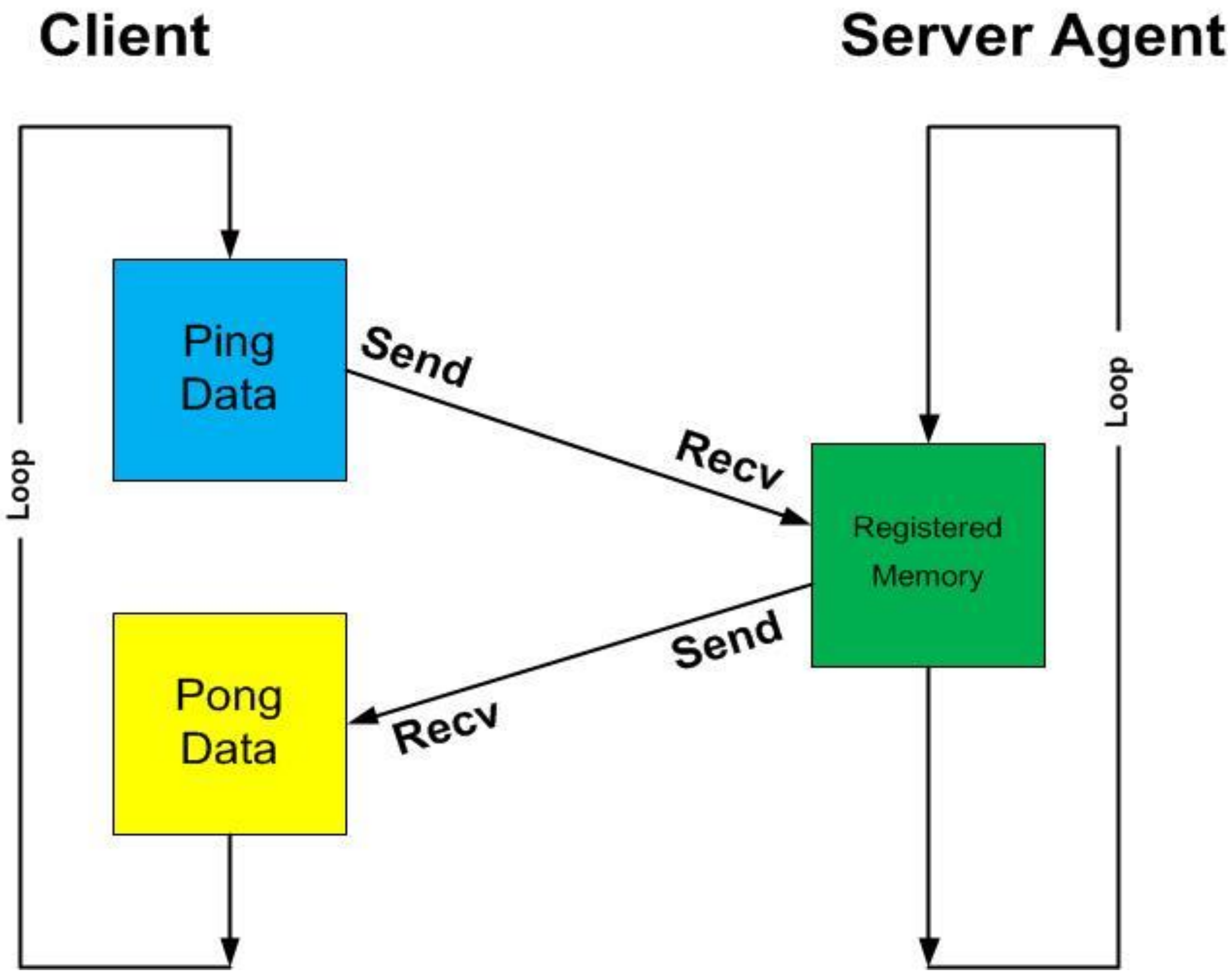
C 2011 OpenFabrics Alliance, Inc

Transfer and completion queues



C 2011 OpenFabrics Alliance, Inc

Ping-Pong using Send/Recv



Synopsis of client's ping-pong loop

```
client_conn->wc_recv = client_conn->wc_send = 0;
while (client_conn->wc_recv < client_conn->limit) {

call our_post_recv() for client_conn->user_data_recv_work_request[0];

call our_post_send() for client_conn->user_data_send_work_request[0];
call our_await_completion() to get IBV_WC_SEND work_completion;
client_conn->wc_send++;

call our_await_completion() to get IBV_WC_RECV work_completion;
if (work_completion.byte_len != client_conn->data_size)
                break;
optionally call our_verify_data() to check recv data against send data

client_conn->wc_recv++;
}       /* while */
```

C 2011 OpenFabrics Alliance, Inc

Server-agent's ping-pong loop

```
call our_post_recv() for agent_conn->user_data_recv_work_request[0];

agent_conn->wc_recv = 0;
agent_conn->wc_send = 0;
while (agent_conn->wc_send < agent_conn->limit) {

call our_await_completion() to get IBV_WC_RECV work_completion;
if (work_completion.byte_len != agent_conn->data_size)
        break;
agent_conn->wc_recv++;

call our_post_recv() for agent_conn->user_data_recv_work_request[0];

call our_post_send() for agent_conn->user_data_send_work_request[0];
call our_await_completion() to get IBV_WC_SEND work_completion;
agent_conn->wc_send++;
}       /* while  */
```

C 2011 OpenFabrics Alliance, Inc

Major components of Program

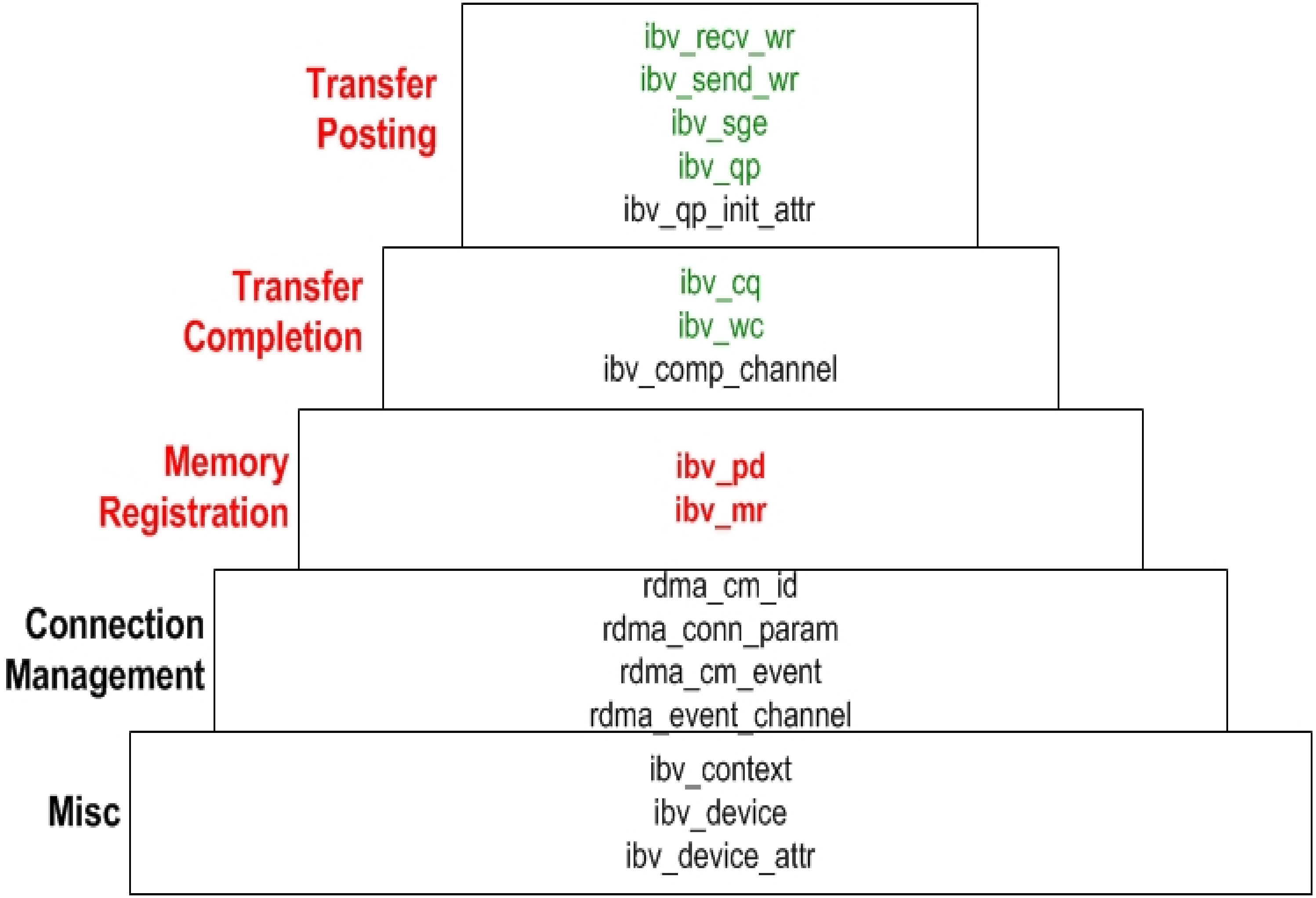
1. Transfer Posting
2. Transfer Completion
3. **Memory Registration**
4. Connection Management
5. Miscellaneous

Discuss these in top-down order

C 2011 OpenFabrics Alliance, Inc

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	**Ibv_alloc_pd** **Ibv_reg_mr**		**Ibv_dealloc_pd** **Ibv_dereg_mr**
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

PD and MR in DS pyramid

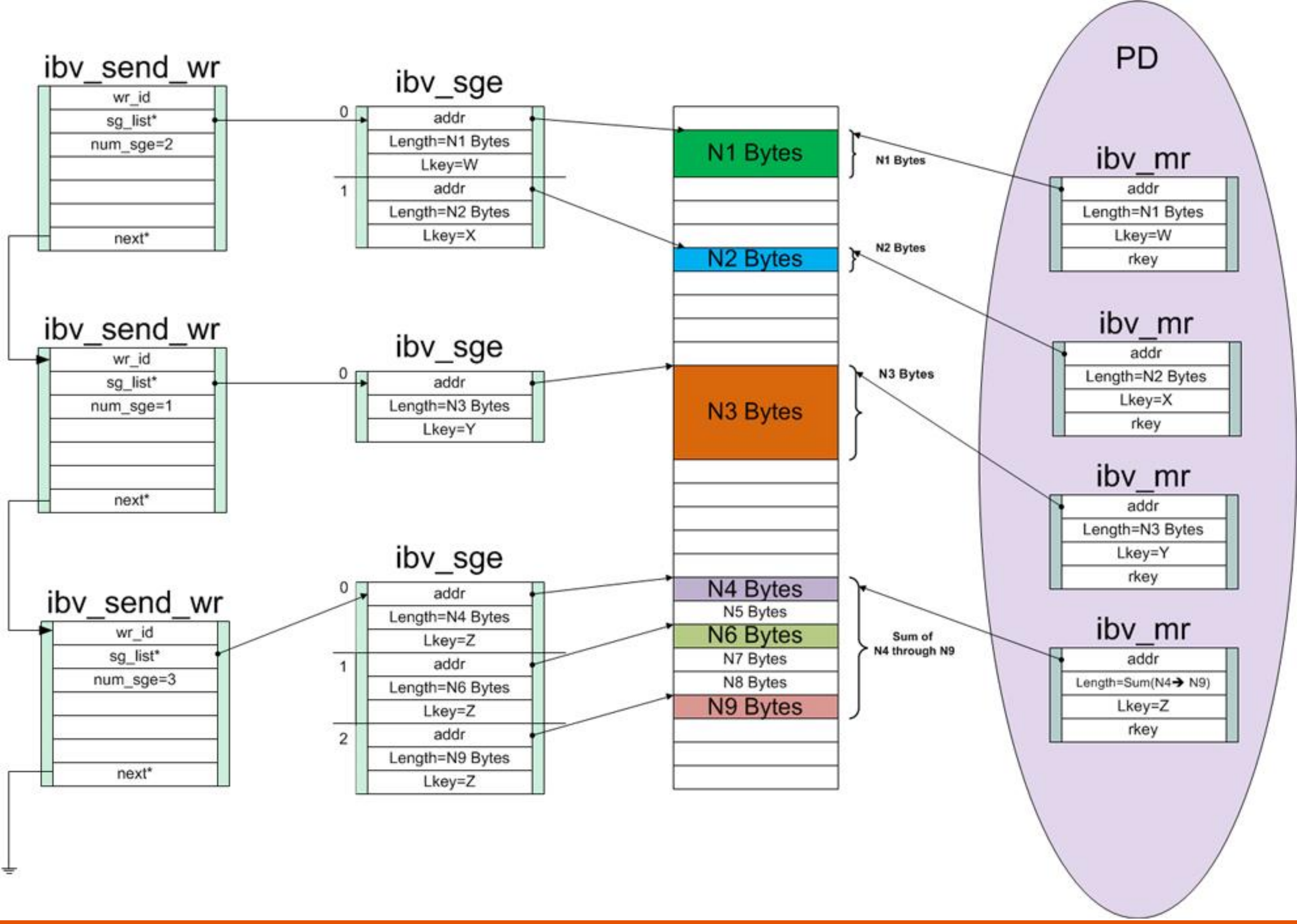


C 2011 OpenFabrics Alliance, Inc

3. OFA Memory Registration

- Goal: to enable Channel Adapter to transfer data directly to/from host memory without CPU intervention
- Protection Domain (PD): user-defined collection of QPs and MRs that enables Channel Adapters to quickly determine if an operation is allowed
- Memory Region (MR): user-defined area of memory registered with certain user-defined access rights in a Protection Domain

C 2011 OpenFabrics Alliance, Inc


ibv_send_wr
wr_id
sg_list*
num_sge=2
next*
ibv_sge
0
addr
Length=N1 Bytes
Lkey=W
1
addr
Length=N2 Bytes
Lkey=X
ibv_send_wr
wr_id
sg_list*
num_sge=1
next*
ibv_sge
0
addr
Length=N3 Bytes
Lkey=Y
ibv_send_wr
wr_id
sg_list*
num_sge=3
next*
ibv_sge
0
addr
Length=N4 Bytes
Lkey=Z
1
addr
Length=N6 Bytes
Lkey=Z
2
addr
Length=N9 Bytes
Lkey=Z
N1 Bytes
N2 Bytes
N3 Bytes
N4 Bytes
N5 Bytes
N6 Bytes
N7 Bytes
N8 Bytes
N9 Bytes
N1 Bytes
N2 Bytes
N3 Bytes
Sum of
N4 through N9
PD
ibv_mr
addr
Length=N1 Bytes
Lkey=W
rkey
ibv_mr
addr
Length=N2 Bytes
Lkey=X
rkey
ibv_mr
addr
Length=N3 Bytes
Lkey=Y
rkey
ibv_mr
addr
Length=Sum(N4→ N9)
Lkey=Z
rkey

Protection Domain - PD

•Means for controlling Channel Adapter access to host system memory

•Each Memory Region is a member of one PD

–Many Memory Regions may belong to same PD

•Each Queue-Pair is a member of one PD

–Many Queue-Pairs may belong to same PD

•Data transfers on a Queue-Pair can only utilize Memory Regions in that Queue-Pair’s PD

C 2011 OpenFabrics Alliance, Inc

Protection Domain Data Structure

- Purpose: enables QPs to utilize MRs for transfers
- Data structure: **struct ibv_pd**
- Fields visible to programmer:

  context verbs field of associated cm_id

- Programmer specifies initial value for this field when PD is created
- cm_id created during connection management

PD Setup/Break-down

- Protection Domain setup

Verb: **ibv_alloc_pd()**

Parameter:

–Context – verbs field of associated cm_id

Return Value:

Pointer to new instance of **struct ibv_pd**

- Protection Domain break-down

Verb: **ibv_dealloc_pd()**

Parameter:

–Pointer to **struct ibv_pd** returned by **ibv_alloc_pd()**

C 2011 OpenFabrics Alliance, Inc

Ibv_alloc_pd() code snippet

```
static int
our_alloc_pd(struct our_control  *conn,  struct rdma_cm_id  *cm_id,
        struct our_options  *options)
{
int        ret = 0;
errno = 0;
conn->protection_domain = ibv_alloc_pd(cm_id->verbs);
if (conn->protection_domain == NULL) {
           ret = ENOMEM;
           our_report_error(ret,  "ibv_alloc_pd",  options);
} else {
           our_trace_ptr("ibv_alloc_pd",  "allocated protection domain",
                                   conn->protection_domain,  options);
}
return ret;
}          /* our_alloc_pd */
```

Memory Registration Process

- Sets up mechanisms to enable Channel Adapter to transfer data directly to/from host memory
- User specifies Memory Region (MR) of contiguous virtual memory
- MR’s physical pages are “pinned” in memory, so they are NOT swapped out during data transfer
- MR’s physical <> virtual mapping is made “permanent” (until deregistration)
- MR’s physical <> virtual mapping is written to Channel Adapter during registration process

Memory Registration (continued)

- User specifies Protection Domain (PD)
- User specifies access rights (shown below) for local and remote Channel Adapters (CAs)
- System generates local key (lkey) and remote key (rkey) for the Memory Region
  – **lkey** used by local Channel Adapter to access MR
  – **rkey** used by remote Channel Adapter to access MR

Memory Region Data Structure

- Purpose – define MR for RDMA transfers
- Data structure: **struct ibv_mr**
- Fields visible to programmer:

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
length	number of bytes in memory region

- Programmer specifies values for **pd**, **addr** and **length**
- System returns values for **lkey** and **rkey**

C 2011 OpenFabrics Alliance, Inc

Memory Region Setup

- Verb: **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() code snippet

```
static struct ibv_mr *
our_setup_mr(struct our_control  *conn,  void  *addr,  unsigned int length,
             int  access,  struct our_options  *options)
{
struct ibv_mr       *mr;

errno = 0;
mr = ibv_reg_mr(conn->protection_domain,  addr,  length,  access);
if (mr == NULL) {
      our_report_error(ENOMEM,  "ibv_reg_mr",  options);
}
return mr;
}     /* our_setup_mr */
```

Access rights (can be OR'd)

Says what access each CA has to local memory:

–IBV_ACCESS_LOCAL_WRITE

Allows local CA to write to local registered memory

–IBV_ACCESS_REMOTE_WRITE

Allows remote CA to write to local registered memory

–IBV_ACCESS_REMOTE_READ

Allows remote CA to read from local registered memory

By default, local CA always allowed to read from local registered memory (access code value is 0)

NOTE: iWARP requires IBV_ACCESS_LOCAL_WRITE if IBV_ACCESS_REMOTE_WRITE is used

Send/Recv access rights

- **ibv_post_send()** with WR opcode IBV_WR_SEND
  - –requires MRs with only default local read access
  - –local CA reads from local memory “out onto the wire”

- **ibv_post_recv()**
  - –requires MRs with at least IBV_ACCESS_LOCAL_WRITE
  - –local CA writes into local memory “in off the wire”

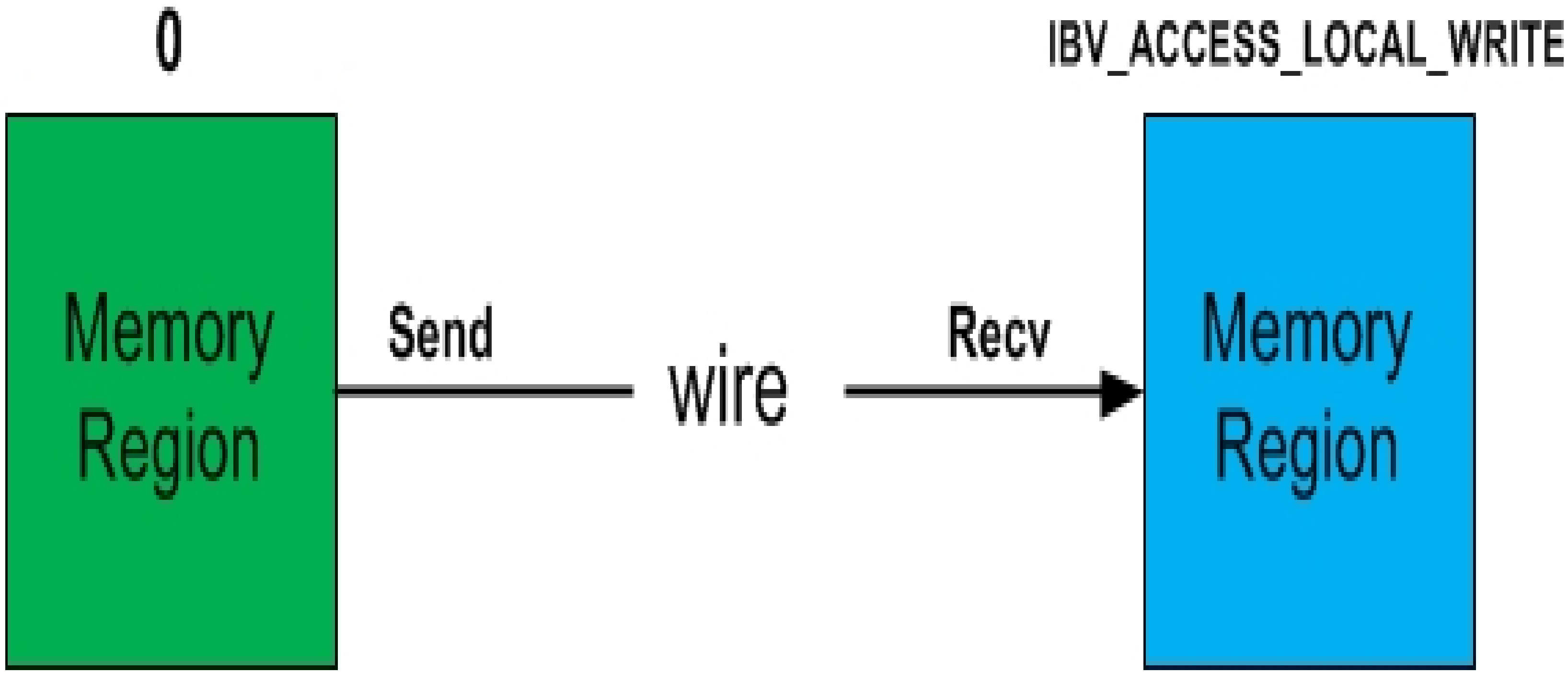
C 2011 OpenFabrics Alliance, Inc

Send/Recv Notes

- Send/Recv never uses **rkey** field in **struct ibv_mr**

- Send/Recv never uses access rights IBV_ACCESS_REMOTE_READ or IBV_ACCESS_REMOTE_WRITE

Send/Recv data flow



Memory Region Break-down

- Verb: **ibv_dereg_mr()**
- Parameter:
  - Pointer to **struct ibv_mr** returned by **ibv_reg_mr()**

Setting up client buffers

- "ping-pong" client needs 2 buffers

  (to allow verification of returned "pong" data)

  – One contains original "ping" data

    default local read access to send to remote agent

  – Other gets returned "pong" data

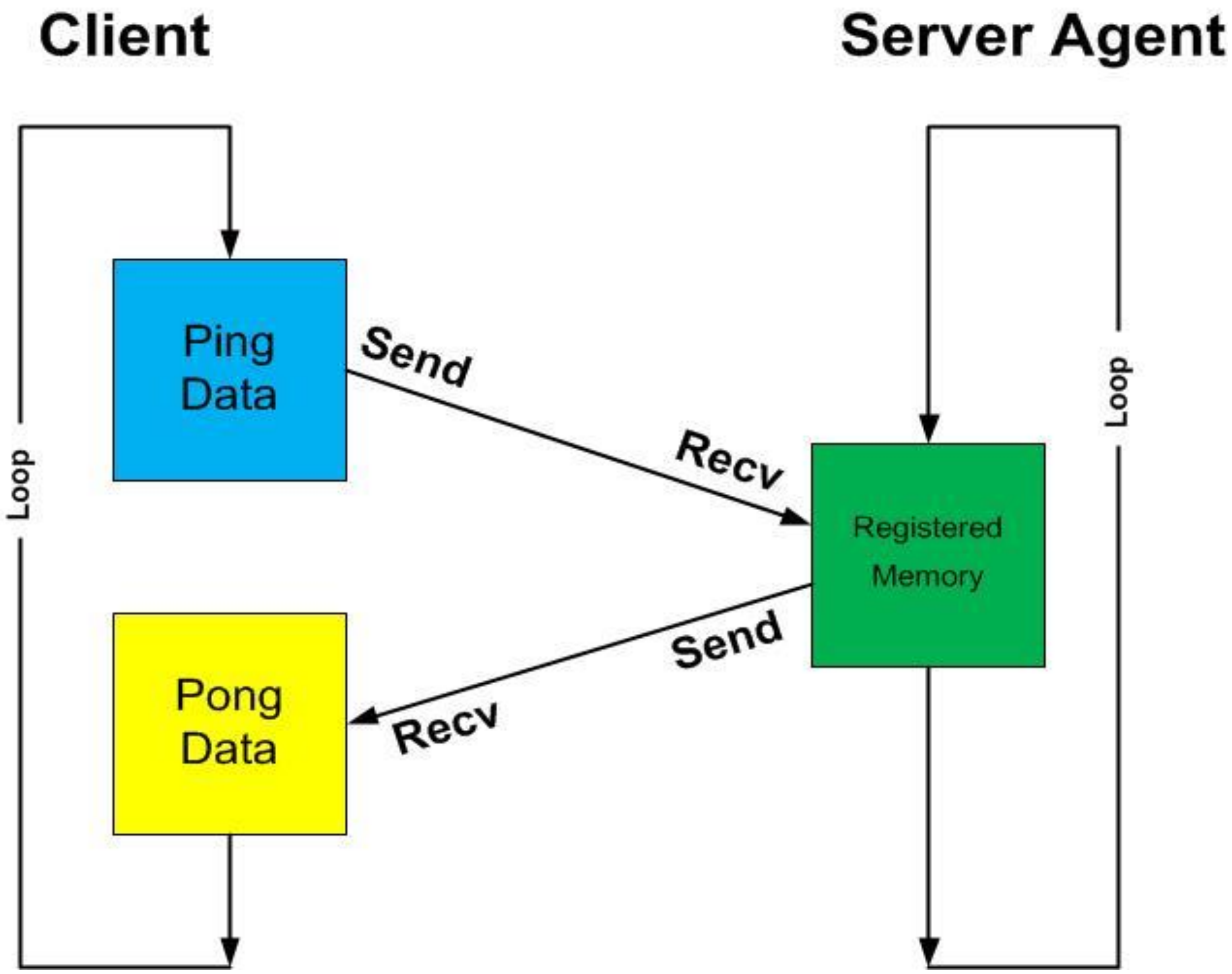
    IBV_ACCESS_LOCAL_WRITE to receive from remote agent

- Client needs 2 work requests

  – One to send "ping" data to agent

  – One to recv "pong" back data from agent

Ping-Pong using Send/Recv



our_setup_client_buffers()

```
int
our_setup_client_buffers(struct our_control  *conn,  struct our_options  *options)
{
int       ret;
          int       access[2] = {0,  IBV_ACCESS_LOCAL_WRITE};
/* client needs 2 buffers,  first to send(),  second to recv() */
if ((ret = our_setup_user_data(conn,  2,  access,  options)) != 0)
          goto out0;
/* client needs 2 work requests,  first to send(),  second to recv() */
our_setup_send_wr(conn,  &conn->user_data_sge[0],  IBV_WR_SEND,  1,
                              &conn->user_data_send_work_request[0] );
our_setup_recv_wr(conn,  &conn->user_data_sge[1],  1,
                              &conn->user_data_recv_work_request[0] );
out0:
return ret;
}         /* our_setup_client_buffers */
```

our_setup_user_data() - part 1

```
static int
our_setup_user_data(struct our_control  *conn,  int  n_user_bufs,  int  access[],
                        struct our_options  *options)
{
int   ret, i;

conn->n_user_data_bufs = 0;
for (i = 0;  i < n_user_bufs;  i++) {
      /* allocate space to hold user data,  plus 1 for '\0' */
      conn->user_data[i] = our_calloc(options->data_size + 1,  options->message);
      if (conn->user_data[i] == NULL) {
            ret = ENOMEM;
            goto out1;
      }
```

our_setup_user_data() - part 2

```
    /* register each user_data buffer for appropriate access */
    ret=our_setup_mr_sge(conn, conn->user_data[i], options->data_size,access[i],
                         &conn->user_data_mr[i], &conn->user_data_sge[i], options);
    if (ret != 0) {
        free(conn->user_data[i]);
        goto out1;
    }
    /* keep count of number of buffers allocated and registered */
    conn->n_user_data_bufs++;
}   /* for */
return 0; /* all user_data buffers set up ok */
out1:
our_unsetup_buffers(conn, options);
out0:
return ret;
}   /* our_setup_user_data */
```

our_setup_mr_sge()

```
/* register a memory addr of length bytes for appropriate access
 * and fill in scatter-gather element for it
 */
static int
our_setup_mr_sge(struct our_control  *conn,  void  *addr,  unsigned int  length,
                 int  access,  struct ibv_mr  **mr,  struct ibv_sge  *sge,
                 struct our_options  *options)
{
/* register the address for appropriate access */
*mr = our_setup_mr(conn,  addr,  length,  access,  options);
if (*mr == NULL)
    return -1;
/* fill in the fields of a single scatter-gather element */
our_setup_sge(addr,  length,  (*mr)->lkey, sge);
return 0;
}     /* our_setup_mr_sge */
```

Setting up agent buffers

- Agent needs 1 buffer
  - **IBV_ACCESS_LOCAL_WRITE** to receive from remote client
  - default local read access to send back to remote client

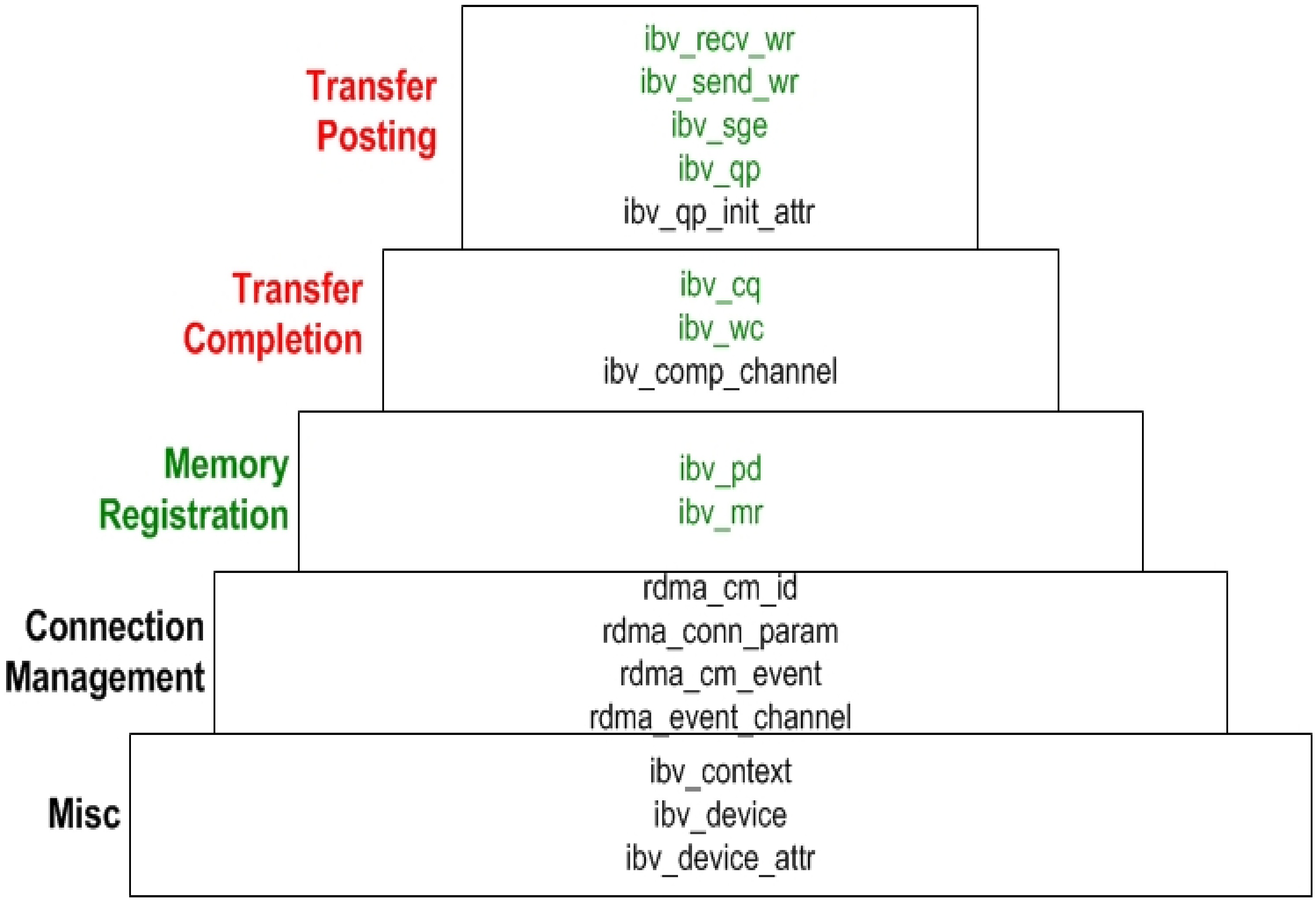
- Agent needs 2 work requests
  - One to receive “ping” data from client
  - One to send “pong” data back to client

C 2011 OpenFabrics Alliance, Inc

our_setup_agent_buffers()

```
int
our_setup_agent_buffers(struct our_control  *conn,  struct our_options  *options)
{
int       ret;
          int                 access[1] = {IBV_ACCESS_LOCAL_WRITE};
/* agent needs 1 buffer for both recv() and send() */
if ((ret = our_setup_user_data(conn,  1,  access,  options))  !=  0)
          goto out0;
/* fill in fields of user_data's work requests for agent's data buffer */
our_setup_recv_wr(conn,  &conn->user_data_sge[0],  1,
                                  &conn->user_data_recv_work_request[0] );
our_setup_send_wr(conn,  &conn->user_data_sge[0],  IBV_WR_SEND,  1,
                                  &conn->user_data_send_work_request[0] );
out0:
return ret;
}         /* our_setup_agent_buffers */
```

DS pyramid so far



C 2011 OpenFabrics Alliance, Inc

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

Major components of Program

1. Transfer Posting
2. Transfer Completion
3. Memory Registration
4. **Connection Management**
5. Miscellaneous

Discuss these in top-down order

C 2011 OpenFabrics Alliance, Inc

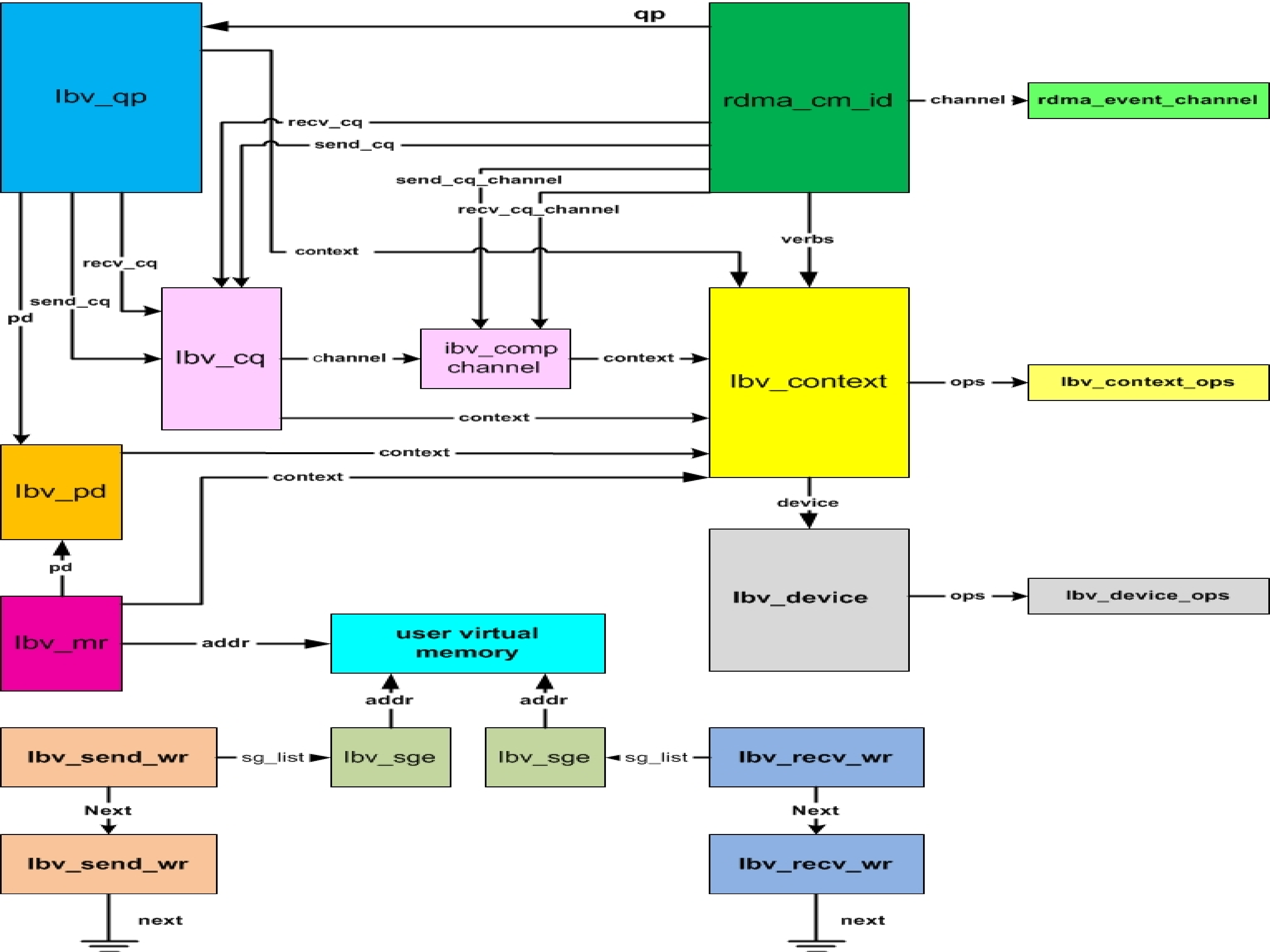
4. OFA Connection Management

- Goal: To establish, maintain and release communication between endpoints
- Same client-server model as "normal" sockets
- As with sockets, different steps taken by client and server in order to "rendezvous"
- More details to consider than "normal" sockets
- For limited situations, synchronous operation available; in practice, asynchronous operation

Client steps to make connection

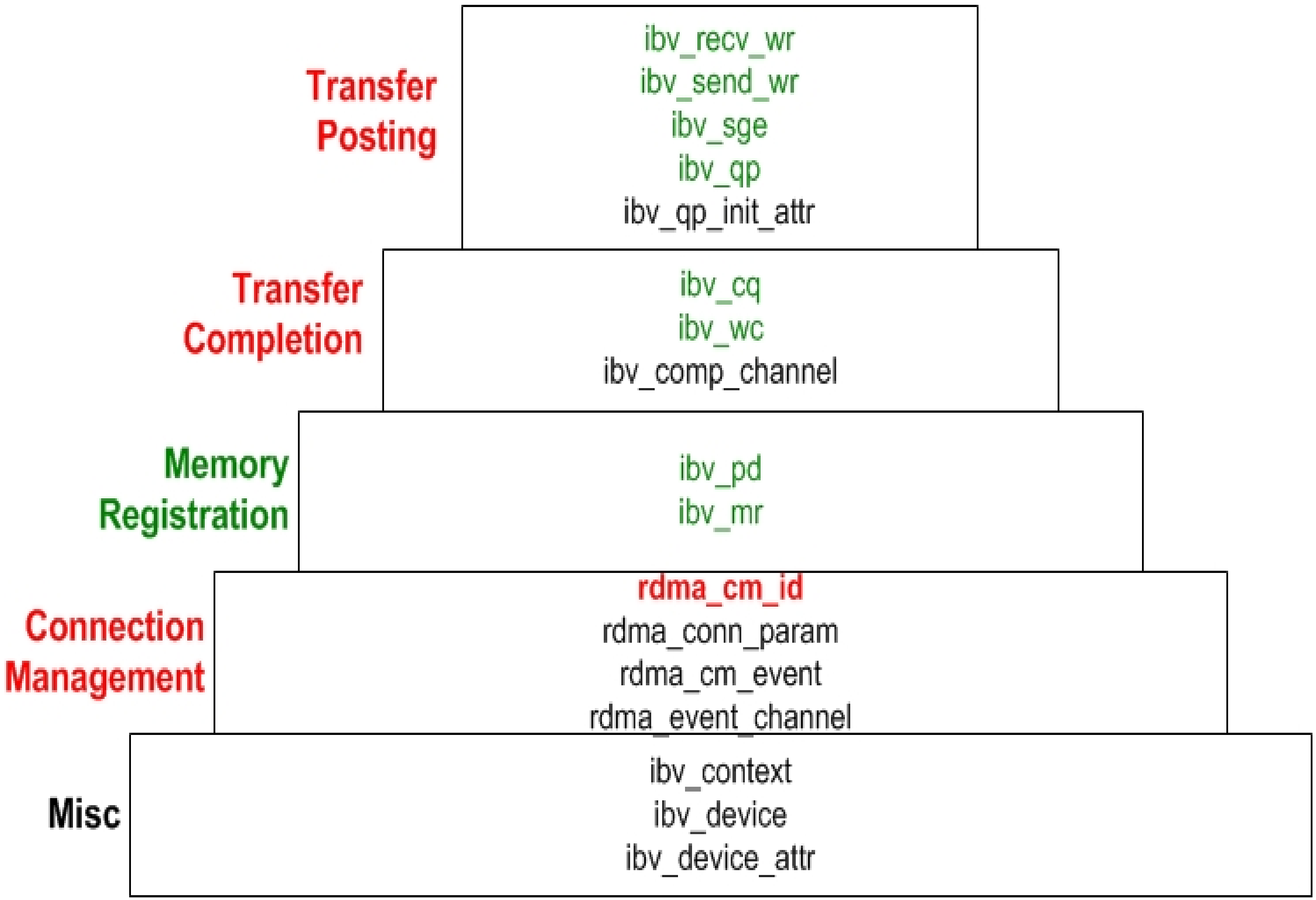
1. Create communication identifier - cm_id
2. Bind client to RDMA device
    1. Translate DNS name into internal address
    2. Resolve address to local RDMA device
    3. Resolve route to server
3. Setup queue pair
    1. Allocate protection domain (already done)
    2. Create completion queue (already done)
    3. Create queue pair
4. Set up client buffers (already done)
5. Connect client to server

C 2011 OpenFabrics Alliance, Inc



Ibv_qp
qp
rdma_cm_id
channel
rdma_event_channel
recv_cq
send_cq
send_cq_channel
recv_cq_channel
context
verbs
recv_cq
send_cq
pd
Ibv_cq
channel
ibv_comp
channel
context
Ibv_context
ops
Ibv_context_ops
context
context
Ibv_pd
context
device
pd
Ibv_device
ops
Ibv_device_ops
Ibv_mr
addr
user virtual
memory
addr
addr
Ibv_send_wr
sg_list
Ibv_sge
Ibv_sge
sg_list
Ibv_recv_wr
Next
Next
Ibv_send_wr
Ibv_recv_wr
next
next

cm_id in DS pyramid



C 2011 OpenFabrics Alliance, Inc

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

C 2011 OpenFabrics Alliance, Inc

Communication Identifier (cm_id)

- The unifying data structure for all connections
- Serves role of a socket "fd" for connection operations
  - Each **cm_id** has an associated QP used for all transfers on the connection

rdma_cm_id data structure

- Purpose – major structure to tie others together
- Data structure: **struct rdma_cm_id**
- Fields visible to programmer:

qp	associated queue pair
ps	port space
verbs	interface to driver and CA
channel	for delivery of connection events to program

- Programmer specifies initial values for **ps**, **verbs** and **channel** when **struct rdma_cm_id** is created

C 2011 OpenFabrics Alliance, Inc

rdma_cm_id setup

- Verb: **rdma_create_id**
- Parameters:

–Channel for reporting events from CA to program
(NULL for cm verbs to run synchronously)

–Storage for returning pointer to **struct rdma_cm_id**

–User-defined identification of this cm_id

–Port space RDMA_PS_TCP

- Return Value:

== 0 cm_id created successfully

!= 0 error code stored in global errno

rdma_cm_id break-down

- Verb: **rdma_destroy_id()**
- Parameter:

–Pointer to **struct rdma_cm_id** returned by **rdma_create_id()**

- Return Value:

== 0 successful

 != 0 error code stored in global errno

C 2011 OpenFabrics Alliance, Inc

our_create_id() code snippet

```
int
our_create_id(struct our_control  *conn,  struct our_options  *options)
{
int    ret;
errno = 0;
ret = rdma_create_id(NULL,  &conn->cm_id,  conn,  RDMA_PS_TCP);
if (ret != 0) {
        our_report_error(ret,  "rdma_create_id",  options);
        goto out0;
}
our_trace_ptr("rdma_create_id",  "created cm_id",  conn->cm_id, options);
/* report new communication channel created for us and its fd */
our_trace_ptr("rdma_create_id",  "returned cm_id->channel",  conn->cm_id->channel,
                                                                      options);
our_trace_ulong("rdma_create_id",  "assigned fd",  conn->cm_id->channel->fd,  options);
out0:
return ret;
}      /* our_create_id */
```

Client steps to make connection

1. Create communication identifier - cm_id
2. Bind client to RDMA device
    1. Translate DNS name into internal address
    2. Resolve address to local RDMA device
    3. Resolve route to server
3. Setup queue pair
    1. Allocate protection domain (already done)
    2. Create completion queue (already done)
    3. Create queue pair
4. Set up client buffers (already done)
5. Connect client to server

C 2011 OpenFabrics Alliance, Inc

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	**rdma_resolve_addr** **rdma_resolve_route** rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

C 2011 OpenFabrics Alliance, Inc

Bind client to RDMA device

More explicit steps than with “normal” sockets

1. Translate server’s DNS name (or IP address) to “addrinfo” and “sockaddr” structures (can use “normal” getaddrinfo())
2. Bind server address structures to local RDMA address and local RDMA device using local routing tables
3. Establish route to server address

Resolve address

- Verb: **rdma_resolve_addr()**
- Parameters:

–communication identifier – cm_id

–client's (local) sockaddr (can be NULL) – src_addr

–server's (remote) sockaddr – dst_addr

–maximum time to wait in milliseconds - timeout

- Return value:

0 cm_id successfully bound to local RDMA device

-1 error code stored in global errno

- System will fill-in non-NULL src_addr

C 2011 OpenFabrics Alliance, Inc

Resolve route

- Purpose: establish a route through fabric to server
- Verb: **rdma_resolve_route()**
- Parameters:
  - communication identifier – cm_id
  - maximum time to wait in milliseconds - timeout
- Return value:

    0 cm_id found route to remote RDMA device

    -1 error code stored in global errno

our_client_bind() - part 1

```
int
our_client_bind(struct our_control *client_conn, struct our_options *options)
{
struct addrinfo                    *aptr, hints;
int                          ret;
/* get structure for remote host node on which server resides */
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
ret=getaddrinfo(options->server_name, options->server_port, &hints, &aptr);
if (ret != 0) {
        fprintf(stderr, "%s getaddrinfo server_name %s port %s: %s\n",
                options->message, options->server_name,
                options->server_port, gai_strerror(ret) );
        return ret;
}
```

our_client_bind() - part 2

```
errno = 0;
ret = rdma_resolve_addr(client_conn->cm_id, NULL,
                                        (struct sockaddr *)aptr->ai_addr, 2000);
if (ret != 0) {
        our_report_error(ret, "rdma_resolve_addr", options);
        goto out1;
}
/* in this demo, rdma_resolve_addr() operates synchronously */
errno = 0;
ret = rdma_resolve_route(client_conn->cm_id, 2000);
if (ret != 0) {
        our_report_error(ret, "rdma_resolve_route", options);
        goto out1;
}
/* in this demo, rdma_resolve_route() operates synchronously */
/* everything worked ok, fall thru, because ret == 0 already */
out1:
freeaddrinfo(aptr);
return ret;
}       /* our_client_bind */
```

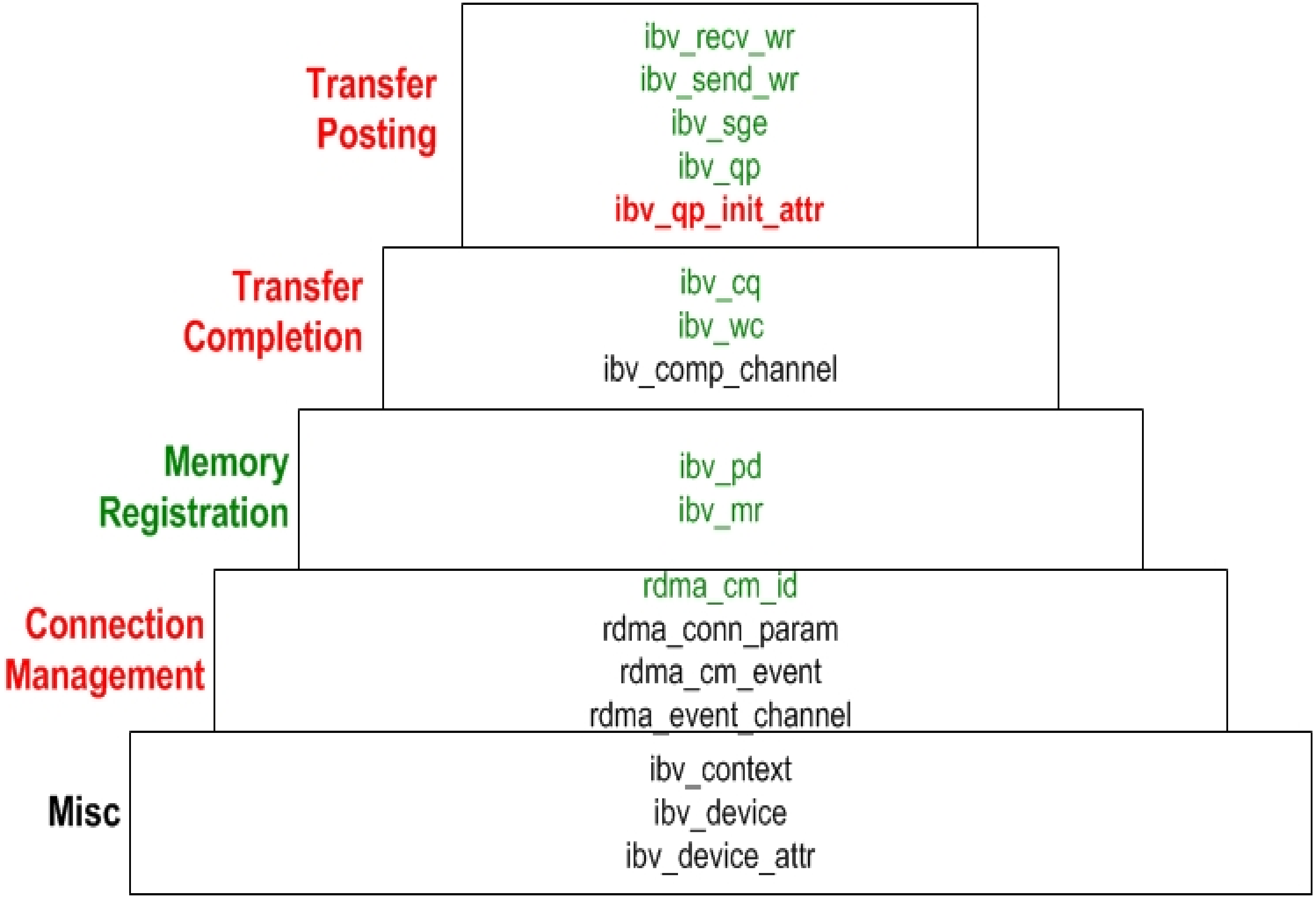
Client steps to make connection

1. Create communication identifier - cm_id
2. Bind client to RDMA device
    1. Translate DNS name into internal address
    2. Resolve address to local RDMA device
    3. Resolve route to server
3. Setup queue pair
    1. Allocate protection domain (already done)
    2. Create completion queue (already done)
    3. Create queue pair
4. Set up client buffers (already done)
5. Connect client to server

C 2011 OpenFabrics Alliance, Inc

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

ibv_qp_init_addr in DS pyramid



Steps to setup queue pair

1. Allocate protection domain (already covered)
2. Create completion queue (already covered)
3. Create queue pair

our_setup_qp() - part 1

```
int
our_setup_qp(struct our_control *conn, struct rdma_cm_id *cm_id,
        struct our_options *options)
{
        int     ret;
/* create a protection domain */
ret = our_alloc_pd(conn, cm_id, options);
if (ret != 0)
                goto err0;

/* create a completion queue */
ret = our_create_cq(conn, cm_id, options);
if (ret != 0)
        goto err1;
```

our_setup_qp() - part 2

```
/* create a queue pair */
ret = our_create_qp(conn, options);
if (ret != 0)
                goto err2;

/* everything worked ok */
        return ret;

err2:
        our_destroy_cq(conn, options);
err1:
our_dealloc_pd(conn, options);
err0:
return ret;
}       /* our_setup_qp */
```

Queue pair setup

- Verb: **rdma_create_qp()**
- Parameters:
  - Communication identifier – cm_id
  - Protection domain
  - Structure of values to initialize new qp
- Return value:

  0 new qp successfully created

  -1 error code stored in global errno
- System fills in qp field of cm_id with newly created instance of **struct ibv_qp**

Ibv_qp_init_attr data structure

- Purpose – to package set of values needed to initialize a new queue pair
- Data structure **struct ibv_qp_init_attr**
- Fields visible to programmer:
  – all of them – the idea is to fill in values that system will use to initialize corresponding fields in a new qp
- Programmer fills in values for all fields before passing this structure as a parameter to **rdma_create_qp()**

our_setup_qp_params()

```
void
our_setup_qp_params(struct our_control *conn,
                    struct ibv_qp_init_attr *init_attr,
                    struct our_options *options)
{
memset(init_attr, 0, sizeof(*init_attr));
init_attr->qp_context = conn;
init_attr->send_cq = conn->completion_queue;
init_attr->recv_cq = conn->completion_queue;
init_attr->srq = NULL;
init_attr->cap.max_send_wr = options->send_queue_depth;
init_attr->cap.max_recv_wr = options->recv_queue_depth;
init_attr->cap.max_send_sge = options->max_send_sge;
init_attr->cap.max_recv_sge = options->max_recv_sge;
init_attr->cap.max_inline_data = 0;
init_attr->qp_type = IBV_QPT_RC;
init_attr->sq_sig_all = 0;
init_attr->xrc_domain = NULL;
}       /* our_setup_qp_params */
```

C 2011 OpenFabrics Alliance, Inc

our_create_qp() - part 1

```
static int
our_create_qp(struct our_control  *conn,  struct our_options  *options)
{
struct ibv_qp_init_attr         init_attr;
int                             ret;

/* set up parameters to define properties of the new queue pair */
our_setup_qp_params(conn,  &init_attr,  options);

errno = 0;
ret = rdma_create_qp(conn->cm_id,  conn->protection_domain,  &init_attr);
if (ret != 0) {
        our_report_error(ret,  "rdma_create_qp",  options);
} else {
        conn->queue_pair = conn->cm_id->qp;
```

C 2011 OpenFabrics Alliance, Inc

our_create_qp() - part 2

```
    our_trace_ptr("rdma_create_qp", "created queue pair",
                                conn->queue_pair, options);
    our_trace_ulong("rdma_create_qp", "max_send_wr",
                                init_attr.cap.max_send_wr, options);
    our_trace_ulong("rdma_create_qp", "max_recv_wr",
                                init_attr.cap.max_recv_wr, options);
    our_trace_ulong("rdma_create_qp", "max_send_sge",
                                init_attr.cap.max_send_sge, options);
    our_trace_ulong("rdma_create_qp", "max_recv_sge",
                                init_attr.cap.max_recv_sge, options);
    our_trace_ulong("rdma_create_qp", "max_inline_data",
                                init_attr.cap.max_inline_data, options);
    }
return ret;
}    /* our_create_qp */
```

Queue pair break-down

- Verb: **rdma_destroy_qp()**
- Parameter:

–Pointer to **struct ibv_cq** returned by **rdma_create_qp()** in qp field of **struct rdma_cm_id**

- Return value:

== 0 ok

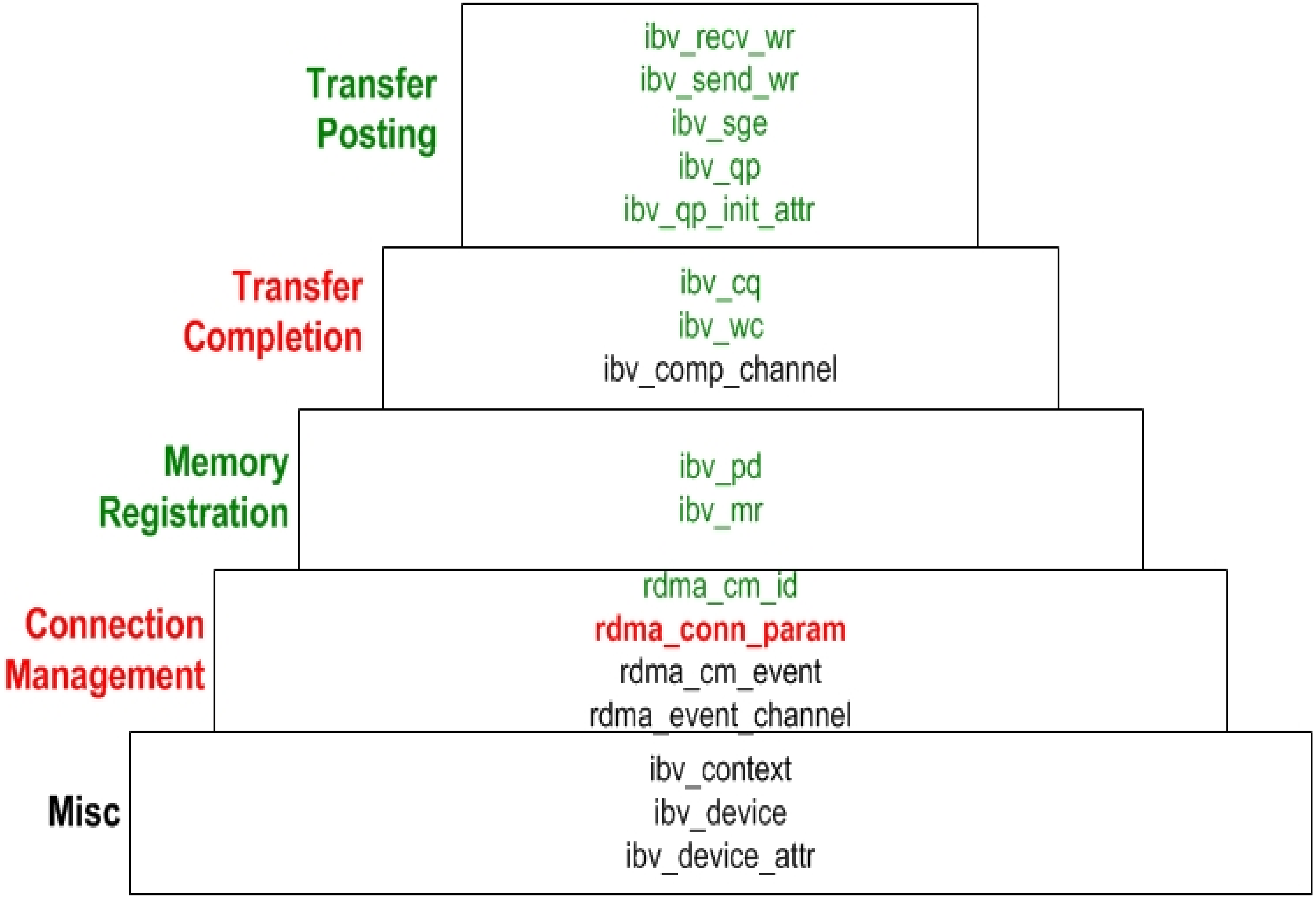
!= 0 error code

Client steps to make connection

1. Create communication identifier - cm_id
2. Bind client to RDMA device
    1. Translate DNS name into internal address
    2. Resolve address to local RDMA device
    3. Resolve route to server
3. Setup queue pair
    1. Allocate protection domain (already done)
    2. Create completion queue (already done)
    3. Create queue pair
4. Set up client buffers (already done)
5. Connect client to server

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route **rdma_connect** **rdma_disconnect** rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

rdma_conn_param in DS pyramid



Client connect to server

- Verb: **rdma_connect()**
- Parameters:

–Communication identifier – 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

C 2011 OpenFabrics Alliance, Inc

rdma_conn_param data structure

•Purpose – to package set of values needed to initialize a new connection

•Data structure **struct rdma_conn_param**

•Fields visible to programmer:

–all of them – the idea is to fill in values that system will use to initialize corresponding fields in a new connection

•Programmer fills in values for all fields before passing this structure as a parameter to **rdma_connect()**

our_setup_conn_params()

```
void
our_setup_conn_params(struct rdma_conn_param *params)
{
memset(params, 0, sizeof(*params));

params->private_data = NULL;
params->private_data_len = 0;
params->responder_resources = 2;
params->initiator_depth = 2;
params->retry_count = 5;
params->rnr_retry_count = 5;
}        /* our_setup_conn_params */
```

our_client_connect()

```
int
our_client_connect(struct our_control  *client_conn,  struct our_options  *options)
{
struct rdma_conn_param          client_params;
int                             ret;

our_setup_conn_params(&client_params);
errno = 0;
ret = rdma_connect(client_conn->cm_id,  &client_params);
if (ret != 0) {
        our_report_error(ret,  "rdma_connect",  options);
        return ret;
}
/* in this demo,  rdma_connect() operates synchronously */
/* client connection established ok */
our_report_ptr("rdma_connect",  "connected cm_id",  client_conn->cm_id,  options);
return ret;
}       /* our_client_connect */
```

Client off and running

- Once client has connected, it can start transmitting data (posting work requests and awaiting work completions)
- NOTE: on iWARP the client MUST be the side to send the first message

- When finished transmitting, client must disconnect, then break-down data structures in the reverse order they were set up

C 2011 OpenFabrics Alliance, Inc

rdma_disconnect()

- Verb: **rdma_disconnect()**
- Parameters:

–Communication identifier – cm_id

- Return Value:

0 connection was successfully disconnected

-1 error code stored in global errno

error code EINVAL means remote side disconnected first

Bottom-up Client Setup

- **rdma_create_id()** - create **struct rdma_cm_id** – identifier
- **rdma_resolve_addr()** - bind **struct rdma_cm_id** to local device
- **rdma_resolve_route()** - resolve route to remote server
- **ibv_alloc_pd()** - create **struct ibv_pd** – protection domain
- **ibv_create_cq()** - create **struct ibv_cq** – completion queue
- **rdma_create_qp()** - create **struct ibv_qp** – queue pair
- **ibv_reg_mr()** - create **struct ibv_mr** – memory region
- **rdma_connect()** - create connection to remote server

C 2011 OpenFabrics Alliance, Inc

client main program – part 1

```
int
main(int argc,  char  *argv[])
{
struct our_control*client_conn;
struct our_options       *options;
int                              result;
/* assume there is an error somewhere along the line */
result = EXIT_FAILURE;
/* process the command line options -- don't go on if any errors */
options = our_process_options(argc,  argv);
if (options == NULL)
       goto out0;
/* allocate our own control structure to keep track of new connection */
client_conn = our_create_control_struct(options);
if (client_conn == NULL)
       goto out1;
```

client main program – part 2

```
if (our_create_id(client_conn,  options) != 0)
        goto out2;
if (our_client_bind(client_conn,  options) != 0)
        goto out3;
if (our_setup_qp(client_conn,  client_conn->cm_id,  options) != 0)
        goto out3;
if (our_setup_client_buffers(client_conn,  options) != 0)
        goto out4;
if (our_client_connect(client_conn,  options) != 0)
        goto out5;

our_trace_ptr("Client",  "connected our_control",  client_conn, options);
```

Client ping-pong use

- **ibv_post_recv()** - start operation to receive pong data
- **ibv_post_send()** - start operation to send ping data
- **ibv_poll_cq()** - get work completion from send ping data
- **ibv_poll_cq()** - get work completion from recv pong data

C 2011 OpenFabrics Alliance, Inc

Bottom-up Client Break-Down

- **rdma_disconnect()** - destroy connection to remote server
- **ibv_dereg_mr()** - destroy **struct ibv_mr** – memory region
- **rdma_destroy_qp()** - destroy **struct ibv_qp** – queue pair
- **ibv_destroy_cp()** - destroy **struct ibv_cq** – completion queue
- **ibv_dealloc_pd()** - deallocate **struct ibv_pd** – protection domain
- **rdma_destroy_id()** - destroy **struct rdma_cm_id** – identifier

C 2011 OpenFabrics Alliance, Inc

client main program – part 3

```
/* the client now ping-pongs data with the server */
if (our_client_operation(client_conn,  options)  !=  0)
        goto out6;
/* the client finished successfully, continue into tear-down phase */
result = EXIT_SUCCESS;
out6:
our_disconnect(client_conn,  options);
out5:
our_unsetup_buffers(client_conn,  options);
out4:
our_unsetup_qp(client_conn,  options);
out3:
our_destroy_id(client_conn,  options);
out2:
our_destroy_control_struct(client_conn,  options);
out1:
our_unprocess_options(options);
out0:
exit(result);
}       /* main */
```

That's it – for the client!

We have shown all the steps necessary to create a simple "ping-pong" client application using the OFA API to program RDMA.

Now show sample runs of the program.

Then we'll look at implementing the server.

Ping-Pong demo0

- BASELINE
- Send/Recv semantics for ping-pong
- Synchronous processing of all cm verbs
  – No **struct rdma_event_channel** created explicitly
- Busy polling for completions
  – No **struct ibv_comp_channel** created explicitly
  – **ibv_poll_cq()** does not block

C 2011 OpenFabrics Alliance, Inc

Server participants

•Listener

–Purpose is to wait for cm event triggered by connection request from client

–Uses system provided information from the cm event to create an agent

–Never transfers any data with client

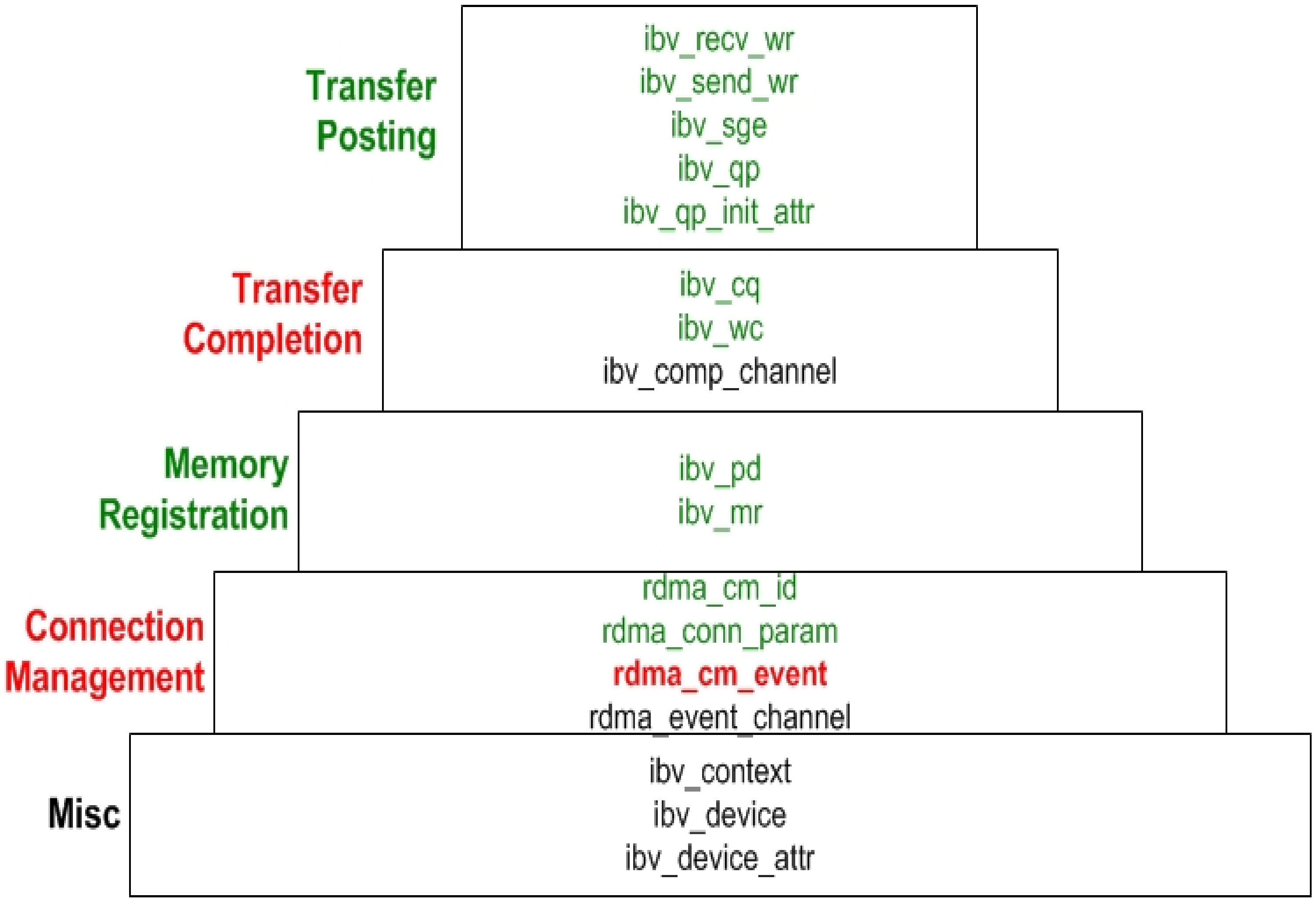
•Agent

–Purpose is to perform all data transfers with one client

–Accepts or rejects the client's connection request

–Disconnects from client when transfers are

rdma_cm_event in DS pyramid



	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect **rdma_bind_addr** **rdma_listen** **rdma_get_cm_event** **rdma_ack_cm_event** **rdma_event_str** rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

New verbs used in server

- Only in connection management, due to asymmetry in rendezvous
  - **rdma_bind_addr()**
  - **rdma_listen()**
  - **rdma_accept()**
  - **rdma_get_cm_event()**
  - **rdma_ack_cm_event()**

First 3 modeled on “normal” socket functions

- bind()
- listen()
- accept()

Bottom-up Listener Setup

- **rdma_create_id() -** create **struct rdma_cm_id** identifier

- **rdma_bind_addr()** - bind **struct rdma_cm_id** to local device

- **rdma_listen()** - establish listener backlog

Server main program - part 1

```
int
main(int argc, char *argv[])
{
struct our_control       *listen_conn;
struct our_options       *options;
struct rdma_cm_id        *event_cm_id;
int                      result;
/* assume there is an error somewhere along the line */
result = EXIT_FAILURE;
/* process the command line options -- don't go on if any errors */
options = our_process_options(argc, argv);
if (options == NULL)
        goto out0;
/* allocate our own control structure for listener's connection */
listen_conn = our_create_control_struct(options);
if (listen_conn == NULL)
        goto out1;
if (our_create_id(listen_conn, options) != 0)
        goto out2;
if (our_listener_bind(listen_conn, options) != 0)
        goto out3;
```

our_listener_bind() - part 1

```
int
our_listener_bind(struct our_control *listen_conn, struct our_options *options)
{
struct addrinfo       *aptr, hints;
int                   ret;
/* get structure for remote host node on which server resides */
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;      /* this makes it a server */
ret = getaddrinfo(options->server_name,  options->server_port, &hints,  &aptr);
if (ret != 0) {
        fprintf(stderr,  "%s: getaddrinfo server_name %s port %s %s\n",
                options->message,  options->server_name,
                options->server_port,  gai_strerror(ret));
        return ret;
}
errno = 0;
ret = rdma_bind_addr(listen_conn->cm_id,  (struct sockaddr *)aptr->ai_addr);
```

our_listener_bind() - part 2

```
if (ret != 0) {
        our_report_error(ret, "rdma_bind_addr", options);
        goto out1;
}
our_trace_ok("rdma_bind_addr", options);
our_trace_ptr("rdma_bind_addr", "returned cm_id -> channel",
                                        listen_conn->cm_id->channel, options);
errno = 0;
ret = rdma_listen(listen_conn->cm_id, OUR_BACKLOG);
if (ret != 0) {
        our_report_error(ret, "rdma_listen", options);
        goto out1;
}
our_trace_ok("rdma_listen", options);
our_trace_ptr("rdma_listen", "returned cm_id -> channel",
                                        listen_conn->cm_id->channel, options);
/* everything worked ok, fall thru, because ret == 0 already */
out1:
freeaddrinfo(aptr);
return ret;
}       /* our_listener_bind */
```

Bottom-up Listener Use

- **rdma_get_cm_event()** - get **struct rdma_cm_event** with type RDMA_CM_EVENT_CONNECT_REQUEST which creates new **struct rdma_cm_id** for agent

- **rdma_ack_cm_event()** - acknowledge **struct rdma_cm_event**

- **rdma_event_str()** - return printable string explaining an **enum rdma_cm_event_type** code

Bottom-up Listener Break-Down

- **rdma_destroy_id()** - destroy **struct rdma_cm_id** – identifier

Server main program – part 2

```
/* listener all setup,  just wait for a client to request a connect */
if (our_await_cm_event(listen_conn,  RDMA_CM_EVENT_CONNECT_REQUEST,
                              "listener",  &event_cm_id, options)  !=  0)
        goto out3;
/* hand the client's request over to a new agent */
if (our_agent(event_cm_id,  options) != 0)
        goto out3;
/* the agent finished successfully, continue into break-down phase */
result = EXIT_SUCCESS;
out3:
our_destroy_id(listen_conn,  options);
out2:
our_destroy_control_struct(listen_conn,  options);
out1:
our_unprocess_options(options);
out0:
exit(result);
}       /* main */
```

C 2011 OpenFabrics Alliance, Inc

our_await_cm_event() - part 1

```
int
our_await_cm_event( struct our_control  *conn,
                    enum rdma_cm_event_type  this_event_type,
                    char  *name,   struct rdma_cm_id  **cm_id,
                    struct our_options  *options)
{
struct rdma_cm_event*     cm_event;
int                       ret;

if (options->flags & TRACING) {
    fprintf(stderr,  "%s: %s awaiting next cm event %d (%s) our_control %p\n",
                     options->message,  name,  this_event_type,
                     rdma_event_str(this_event_type),  conn);
}
```

our_await_cm_event() - part 2

```
/* block until we get a cm_event from the communication manager */
errno = 0;
ret = rdma_get_cm_event(conn->cm_id->channel,  &cm_event);
if (ret != 0) {
        our_report_error(ret,  "rdma_get_cm_event",  options);
        goto out0;
}
if (options->flags & TRACING) {
        fprintf(stderr,  "%s: %s got cm event %d (%s) cm_id %p "
                        "our_control %p status %d\n",
                        options->message, name,
                        cm_event->event,  rdma_event_str(cm_event->event),
                        cm_event->id,  conn, cm_event->status);
}
```

our_await_cm_event() - part 3

```
if (cm_event->event != this_event_type) {
        fprintf(stderr, "%s: %s expected cm event %d (%s)\n",
                        options->message, name,
                        this_event_type, rdma_event_str(this_event_type));
        ret = -1;
} else {
        if (cm_id != NULL) {
                *cm_id = cm_event->id;
        }
}

/* all cm_events returned by rdma_get_cm_event() MUST be acknowledged */
rdma_ack_cm_event(cm_event);
out0:
return ret;
}       /* our_await_cm_event */
```

server participants

- Listener
  - Purpose is to wait for cm event triggered by connection request from client
  - Uses system provided information from the cm event to create an agent
  - Never transfers any data with client
- Agent
  - Purpose is to perform all data transfers with one client
  - Accepts or rejects the client's connection request
  - Disconnects from client when transfers are

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str **rdma_accept** **rdma_reject** rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

Agent Overview

- Utilizes **struct rdma_cm_id** from connect request
- Sets up all its own structures
- Calls **rdma_accept()** or **rdma_reject()**
- Does all the data transfers with client
- Calls **rdma_disconnect()** when finished

C 2011 OpenFabrics Alliance, Inc

Bottom-up Agent Setup

•**our_create_control_struct()** - create **struct our_control**

•**our_migrate_id()** - move **struct rdma_cm_id** from connect request into new **struct our_control**

•**ibv_alloc_pd()** - create **struct ibv_pd** – protection domain

•**ibv_create_cq()** - create **struct ibv_cq** – completion queue

•**rdma_create_qp()** - create **struct ibv_qp** – queue pair

•**ibv_reg_mr()** – create **struct ibv_mr** – memory region

•**ibv_post_recv()** - start receive of first message from client

•**rdma_accept()** - accept client's connection request

our_agent() - part 1

```
static int
our_agent(struct rdma_cm_id  *event_cm_id,  struct our_options  *options)
{
struct our_control       *agent_conn;
int                      result;
/* assume there is an error somewhere along the line */
result = EXIT_FAILURE;
agent_conn = our_create_control_struct(options);
if (agent_conn == NULL)
        goto out0;
if (our_migrate_id(agent_conn,  event_cm_id,  options) != 0)
        goto out1;
if (our_setup_qp(agent_conn,  agent_conn->cm_id,  options) != 0)
        goto out2;
if (our_setup_agent_buffers(agent_conn,  options) != 0)
        goto out3;
/* post first receive on the agent_conn */
if (our_post_recv(agent_conn,  &agent_conn->user_data_recv_work_request[0],  options) != 0)
        goto out4;
if (our_agent_connect(agent_conn,  options) != 0)
        goto out4;
```

our_migrate_id()

```
int
our_migrate_id(struct our_control *conn, struct rdma_cm_id *new_cm_id,
               struct our_options *options)
{
/* simple when we have not created our own channel */
conn->cm_id = new_cm_id;
new_cm_id->context = conn;

/* report new cm_id created for us */
our_trace_ptr("our_migrate_id", "migrated cm_id", conn->cm_id, options);
return 0;
}     /* our_migrate_id */
```

our_agent_connect()

```
int
our_agent_connect(struct our_control *agent_conn, struct our_options *options)
{
struct rdma_conn_param       agent_params;
int                          ret;

our_setup_conn_params(&agent_params);
errno = 0;
ret = rdma_accept(agent_conn->cm_id, &agent_params);
if (ret != 0) {
        our_report_error(ret, "rdma_accept", options);
} else {
        /* in this demo, rdma_accept() operates synchronously */
        /* agent connection established ok */
        our_report_ptr("rdma_accept", "accepted cm_id", agent_conn->cm_id, options);
}
return ret;
}       /* our_agent_connect */
```

Agent ping-pong use

- **ibv_poll_cq()** - get work completion from recv ping data
- **ibv_post_recv()** - start operation to receive next ping data
- **ibv_post_send()** - start operation to send pong data
- **ibv_poll_cq()** - get work completion from send pong data

Bottom-up Agent Break-Down

- **rdma_disconnect()** - destroy connection to remote server
- **ibv_dereg_mr()** - destroy **struct ibv_mr** – memory region
- **rdma_destroy_qp()** - destroy **struct ibv_qp** – queue pair
- **ibv_destroy_cp()** - destroy **struct ibv_cq** – completion queue
- **ibv_dealloc_pd()** - deallocate **struct ibv_pd** – protection domain
- **rdma_destroy_id()** - destroy **struct rdma_cm_id** – identifier

C 2011 OpenFabrics Alliance, Inc

our_agent() - part 2

```
our_trace_ptr("Agent", "accepted our_control", agent_conn, options);
/* the agent now ping-pongs data with the client */
if (our_agent_operation(agent_conn, options) != 0)
        goto out5;
/* the agent finished successfully, continue into tear-down phase */
result = EXIT_SUCCESS;
out5:
our_disconnect(agent_conn, options);
out4:
our_unsetup_buffers(agent_conn, options);
out3:
our_unsetup_qp(agent_conn, options);
out2:
our_destroy_id(agent_conn, options);
out1:
our_destroy_control_struct(agent_conn, options);
out0:
return result;
}       /* our_agent */
```

Ping-Pong example ping-sr-0

- BASELINE
- Send/Recv semantics for ping-pong
- Synchronous processing of all cm verbs
  - No **struct rdma_event_channel** created explicitly
  - **rdma_get_cm_event()** blocks
- Busy polling for completions
  - No **struct ibv_comp_channel** created explicitly
  - **ibv_poll_cq()** does not block

Problems with ping-sr-0

- Problem 1
  - Parameters for number of messages and/or size of messages could be specified differently on client and server
- Problem 2
  - One side could disconnect unexpectedly (network failure, program crashed or was killed)
- Problem 3
  - CPU utilization is very high

Solution to Problem 1

- Have client communicate its values for number of messages and size of messages to server BEFORE transferring any data
- Client **rdma_connect()** can carry private data from client to server
- Listener **rdma_get_cm_event()** gets private data in RDMA_CM_EVENT_CONNECT_REQUEST
- Agent started by listener uses this private data to set its **data_size** and **limit** values

C 2011 OpenFabrics Alliance, Inc

Ping-Pong example ping-sr-1

- Requires small changes to a few files in ping-sr-0
  - New **struct our_connect_info** to define contents of private data (values on the wire MUST be in network byte order)
  - Before client calls **rdma_connect()** fill in new structure with client's values for **data_size** and **limit** using htonll()
  - After listener calls **rdma_get_cm_event()** copy private data out of **struct rdma_cm_event** into new structure
  - When **struct rdma_cm_id** is migrated to agent, set **data_size** and **limit** from new structure using ntohll()

C 2011 OpenFabrics Alliance, Inc

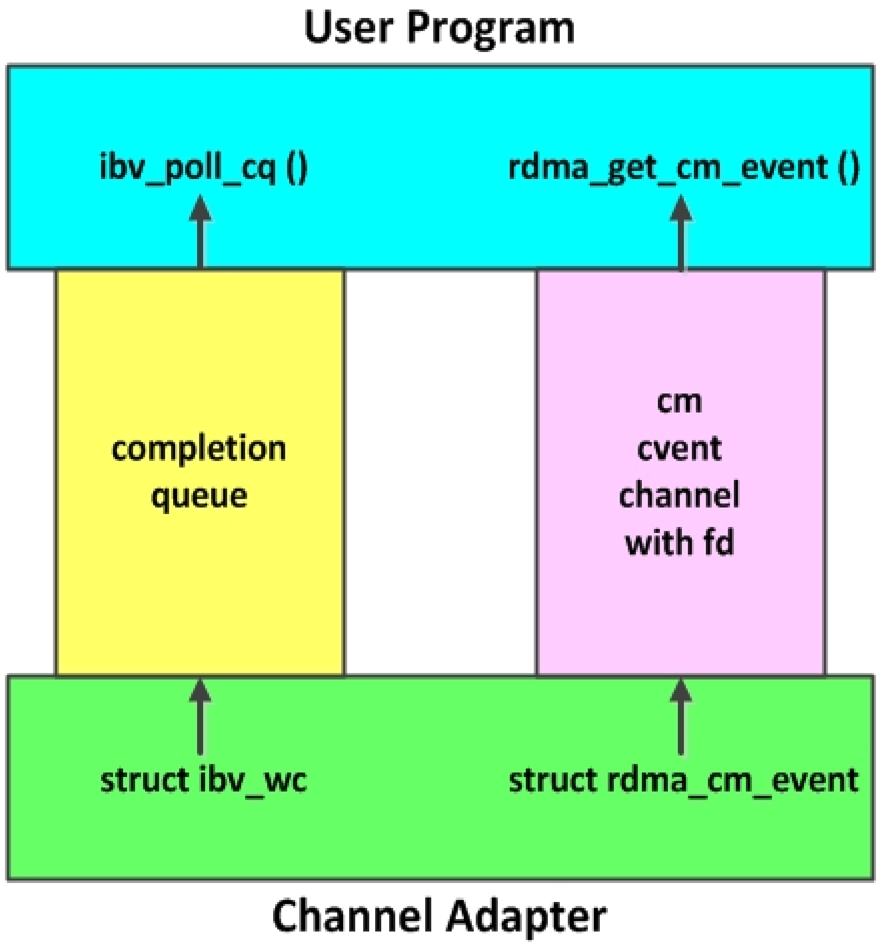
Solutions to Problem 2

- Problem arises because:
  - completions are handled asynchronously

    ibv_poll_cq() is non-blocking
  - cm events are handled synchronously

    rdma_get_cm_event() blocks
- Both completions and cm events are examples of status info flowing back from CA to the program
  - completions convey results of data transfer operations
  - cm events convey results of connection management operations

Status flows from CA to program



Differences in status flow types

- CA delivers completions by enqueuing work completion structures (struct ibv_wc) in completion queues (struct ibv_cq)
  - **struct ibv_wc** carries results relevant to data transfer operations
- CA delivers cm events by enqueuing event structures (struct rdma_cm_event) in event channels (struct rdma_event_channel)
  - **struct rdma_cm_event** carries results relevant to connection management operations

Default status flow types

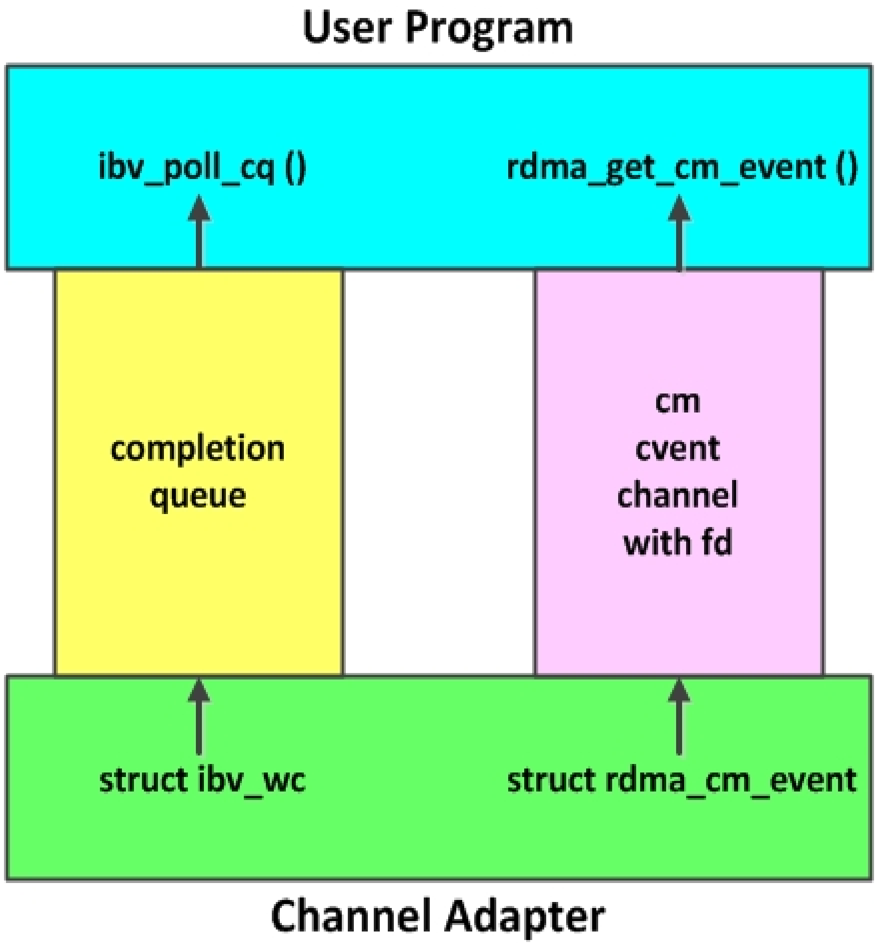
- Work Completions – contain data transfer final status
  - If ibv_comp_channel NOT explicitly created then all completion verbs are **asynchronous**
  - **ibv_poll_cq()** does not block, it just polls

- CM events - contain connection management results
  - If rdma_event_channel NOT explicitly created then all cm verbs are **synchronous**
  - **rdma_get_cm_event()** blocks, it does not poll

- Notice the differences in default synchronicity

© 2011 OpenFabrics Alliance, Inc

Status flows from CA to program



Handling status flows from CA

- Three possibilities

1. Do not explicitly create a channel (the default)
2. Explicitly create a channel
3. Explicitly create a channel and put it into O_NONBLOCK mode

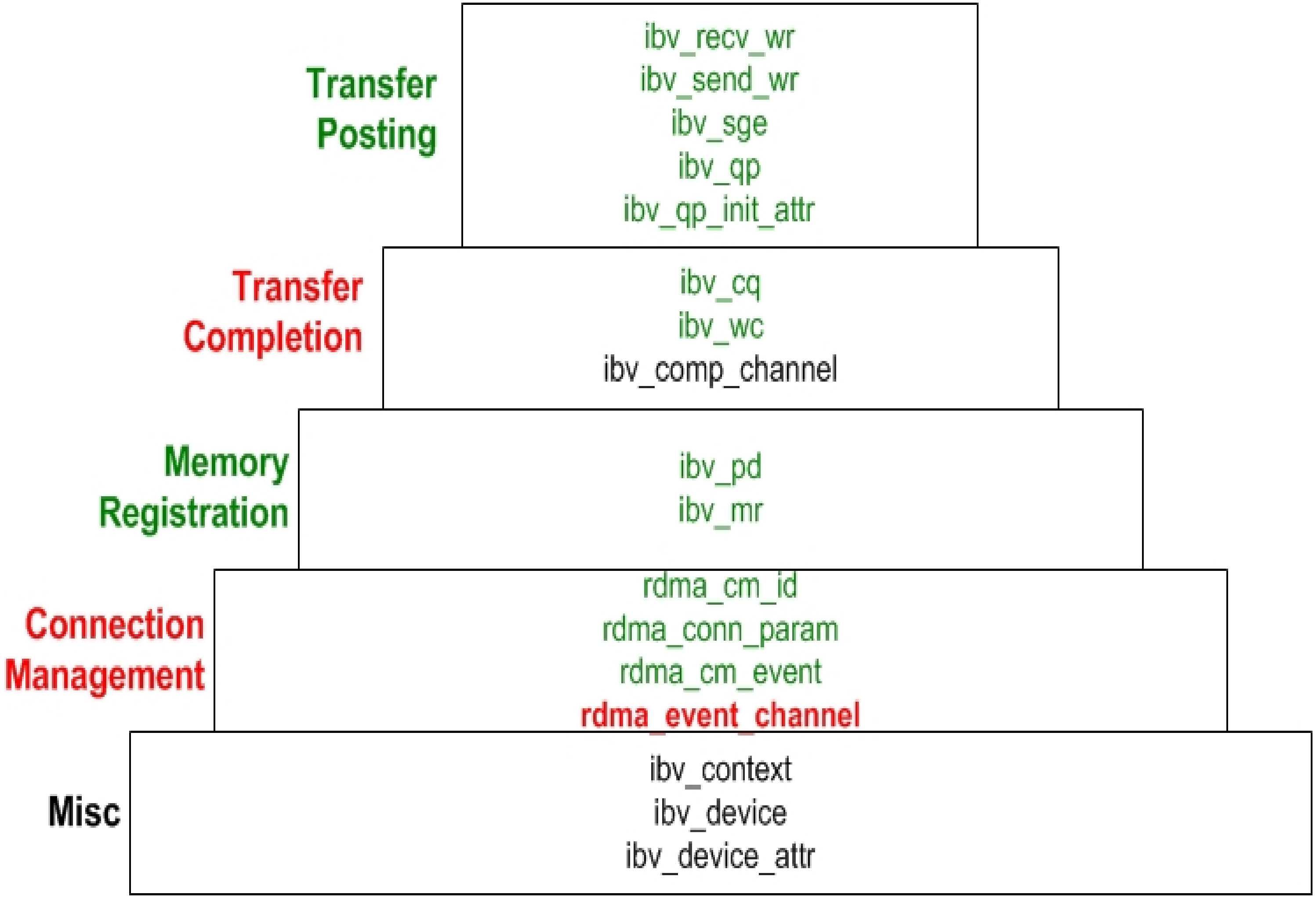
- All three possibilities apply to both types of flow
  - cm events
  - completions
- Result – 9 combinations to consider

Ping-Pong exercise ping-sr-2e

- Based on ping-sr-1
- Explicitly create **rdma_event_channel**
- Many cm verbs now operate asynchronously
- Exception: **rdma_get_cm_event()** still blocks

C 2011 OpenFabrics Alliance, Inc

rdma_event_channel in DS pyramid



	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject **rdma_migrate_id** rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

event_channel data structure

•Purpose: report asynchronous events to user program from connection management

•Data structure: **struct rdma_event_channel**

•Fields visible to programmer:

fd operating system file descriptor (small integer)

•System fills in **fd** field with newly assigned system file descriptor number

event_channel setup

- Verb: **rdma_create_event_channel()**
- Parameters: none
- Return value:
- Pointer to new instance of **struct rdma_event_channel**
- System fills in **fd** field of event_channel with newly assigned system file descriptor number

event_channel break-down

- Verb: **rdma_destroy_event_channel()**
- Parameter:
  - Pointer to **struct rdma_event_channel** returned by **rdma_create_event_channel()**

our_create_event_channel()

```
static int
our_create_event_channel(struct our_control  *conn,  struct our_options  *options)
{
int    ret;
errno = 0;
conn->cm_event_channel = rdma_create_event_channel();
if (conn->cm_event_channel == NULL) {
        ret = ENOMEM;
        our_report_error(ret,  "rdma_create_event_channel",  options);
        return ret;
}
/* report the new communication channel created by us and its fd */
our_trace_ptr("rdma_create_event_channel",  "created cm_event_channel",
                                          conn->cm_event_channel,  options);
our_trace_ulong("rdma_create_event_channel",  "assigned fd",
                                     conn->cm_event_channel->fd,  options);
return 0;
}      /* our_create_event_channel */
```

our_migrate_id() - part 1

```
/* already have a communication identifier,
 * migrate it to use a new channel and set its context to be this new conn
 */
int
our_migrate_id(struct our_control *conn, struct rdma_cm_id *new_cm_id,
                struct our_connect_info *connect_info,
                struct our_options *options)
{
int    ret;

/* replace agent's limit and data_size with values from connect_info */
our_trace_uint64("option", "count", options->limit, options);
options->limit = ntohll(connect_info->remote_limit);
our_report_uint64("client", "count", options->limit, options);

our_trace_uint64("option", "data_size", options->data_size, options);
options->data_size = ntohll(connect_info->remote_data_size);
our_report_uint64("client", "data_size", options->data_size, options);
```

our_migrate_id() - part 2

```
/* create our own channel */
ret = our_create_event_channel(conn, options);
if (ret != 0)
        goto out0;
errno = 0;
ret = rdma_migrate_id(new_cm_id, conn->cm_event_channel);
if (ret != 0) {
        our_report_error(ret, "rdma_migrate_id", options);
        our_destroy_event_channel(conn, options);
} else {
        conn->cm_id = new_cm_id;
        new_cm_id->context = conn;
        /* report new cm_id created for us */
        our_trace_ptr("rdma_migrate_id", "migrated cm_id", conn->cm_id, options);
}
out0:
return ret;
}       /* our_migrate_id */
```

Exercise ping-sr-2e

- Based on ping-sr-1
- Explicitly creates **rdma_event_channel**
- Many cm verbs now operate asynchronously
- Exception: **rdma_get_cm_event()** still blocks
- Does NOT solve problem 2, because it simply waits in-line after every asynchronous cm verb by calling **our_await_cm_event()** – for example, see **our_bind_client()**

- Exercise for the Lab

Solution to Problem 2

- We have created an event channel, but **rdma_get_cm_event()** still blocks, so can't just poll it periodically
- Instead create new user thread to wait on blocking **rdma_get_cm_event()**
  - Thread copies event info into fields in struct our_control
  - **our_await_completion()** checks this info in already existing "busy wait" loop around **ibv_poll_cq()**
  - **our_await_cm_event()** synchronizes with new thread

Threads, not child processes

- OFA resources (connections, data structures, registered memory regions) cannot be inherited by a child process from it's parent process
- Similarly, OFA resources do not "live" across an exec() system call.
- A new child process or prgram can, of course, create it's own new OFA resources (connections, data structures, registered memory)
- Therefore, a listener cannot create agent processes, only agent threads

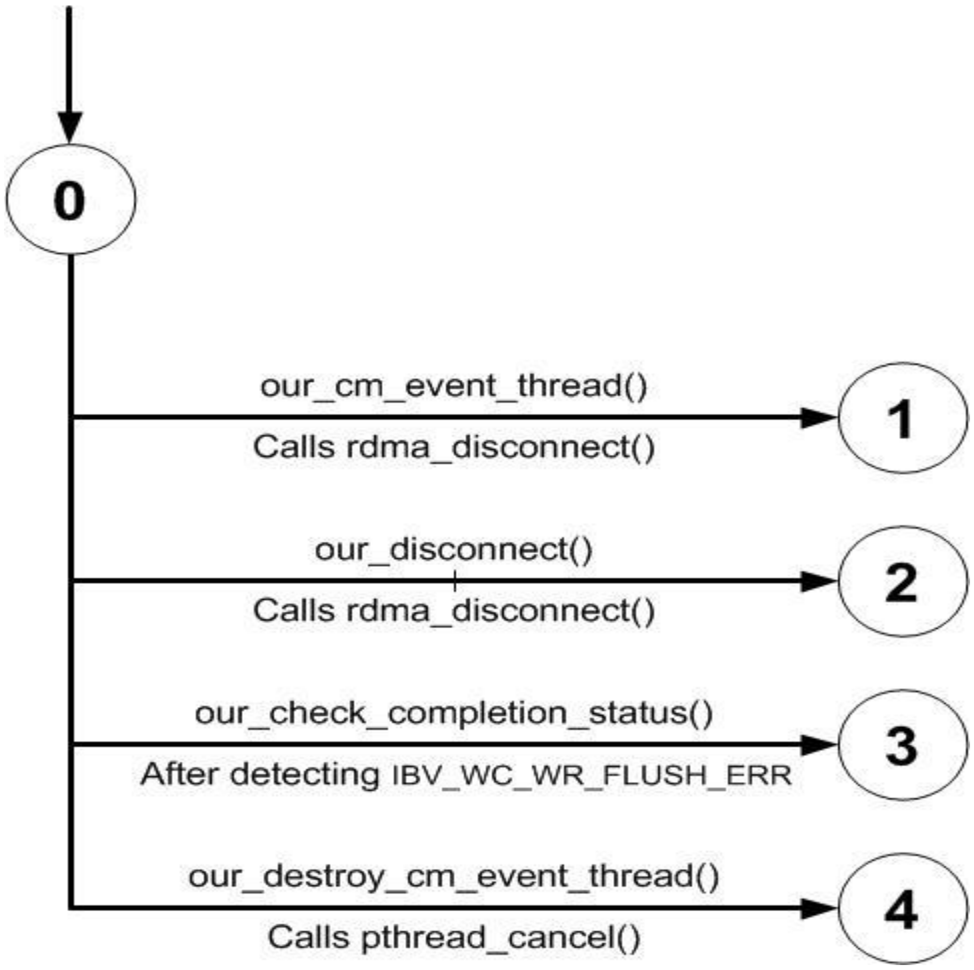
C 2011 OpenFabrics Alliance, Inc

New thread for cm events

- In prototypes.h:
  - add fields in struct our_control to hold latest cm event info and a mutex_lock so updates are atomic
- In client.c and agent.c:
  - add call in main() to **our_create_cm_event_thread()**
- In process_cm_events.c:
  - add functions **our_create_cm_event_thread()** and **our_cm_event_thread()**
  - modify function **our_await_cm_event()**
- In process_completions.c:
  - modify **our_await_completion()**

C 2011 OpenFabrics Alliance, Inc

Coordinating disconnect events with disconnected state variable



C 2011 OpenFabrics Alliance, Inc

Ping-Pong example ping-sr-2

- Creates separate asynchronous thread to handle cm events
- That thread blocks, it does not busy wait
- When thread gets **rdma_cm_event**, it stores information from it so main thread's “busy wait” loop can find it

- Run this and walk the code

Solution to problem 3

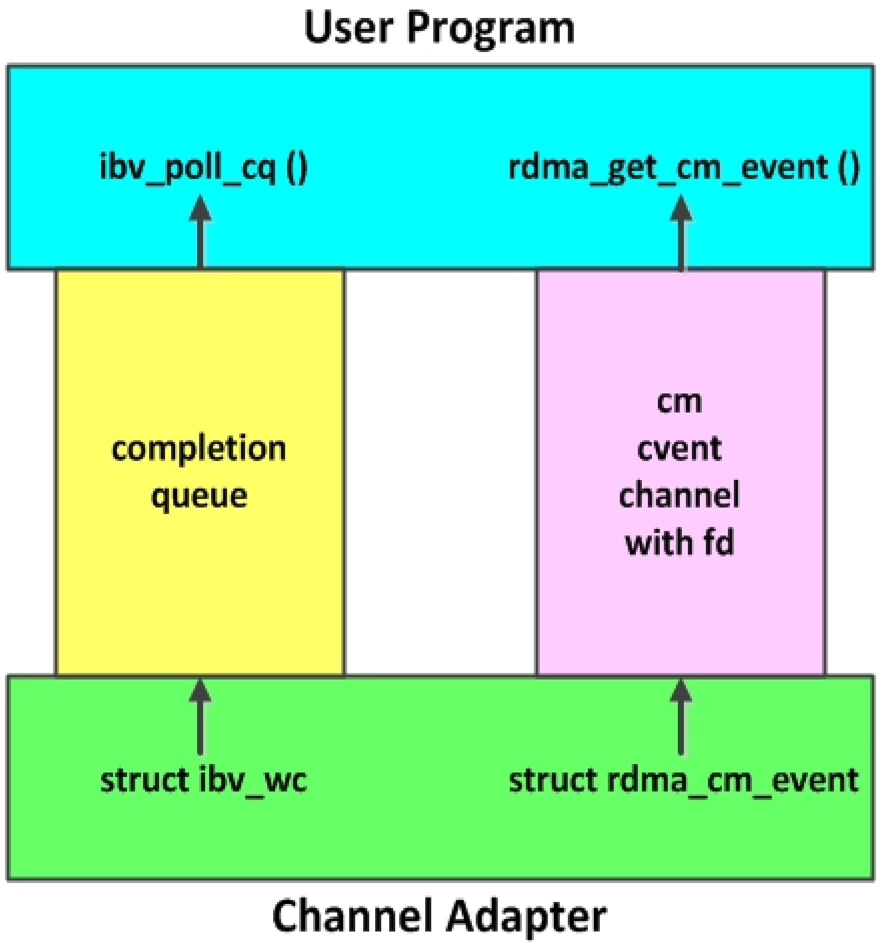
- The problem is high CPU utilization caused by "busy-waiting" on **ibv_poll_cq()** for completions
- To solve this, need to eliminate "busy-waiting"
- To do this, we need to block until an asynchronous completion occurs

Eliminating busy waiting

- Busy waiting burns CPU cycles
  - Main reason for burning cycles – to improve latency
  - OK if have lots of unused CPU cycles
- Have shown how to eliminate busy waiting when dealing with cm events through the use of an asynchronous thread
- Now look at eliminating busy waiting when dealing with completions (**ibv_poll_cq()**)

C 2011 OpenFabrics Alliance, Inc

Status flows from CA to program



Types of CM status flows from CA

1.Completely synchronously

1.All cm verbs block

2.No explicit channel creation

2.Mostly asynchronously

1.Many cm verbs non-blocking except **rdma_get_cm_event()**

2.Explicit channel creation

3.Completely asynchronously

1.Most cm verbs non-block, including **rdma_get_cm_event()**

2.Explicit channel creation with O_NONBLOCK mode

C 2011 OpenFabrics Alliance, Inc

Types of Completion status flows

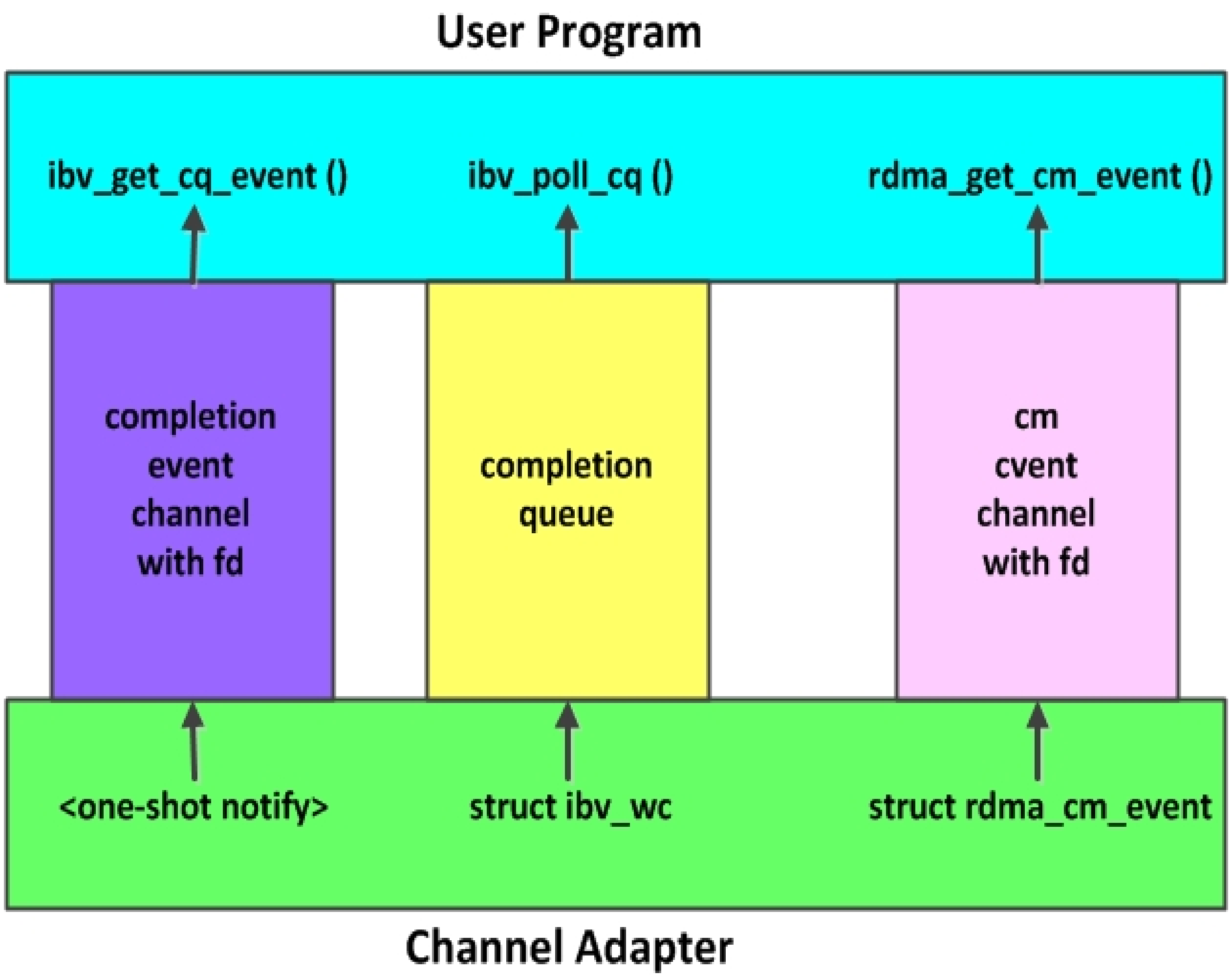
1. Completely asynchronously
   1. All completion verbs are non-blocking
   2. No explicit channel creation, need busy polling
2. Partially asynchronously
   1. Introduce completion events and **ibv_get_cq_event()**
   2. Explicit channel creation, **ibv_get_cq_event()** blocks
3. Completely asynchronously
   1. **ibv_get_cq_event()** becomes non-blocking
   2. Explicit channel creation with O_NONBLOCK

completion channel in DS pyramid

Transfer Posting	ibv_recv_wr ibv_send_wr ibv_sge ibv_qp ibv_qp_init_attr
Transfer Completion	ibv_cq ibv_wc **ibv_comp_channel**
Memory Registration	ibv_pd ibv_mr
Connection Management	rdma_cm_id rdma_conn_param rdma_cm_event rdma_event_channel
Misc	ibv_context ibv_device ibv_device_attr

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq **ibv_create_comp_channel**	Ibv_poll_cq Ibv_wc_status_str **Ibv_req_notify_cq** **Ibv_get_cq_event** **Ibv_ack_cq_events**	ibv_destroy_cp **ibv_destroy_comp_channel**
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	

Status flow with completion channel



Completion status flow from CA

- All demos so far are completely asynchronous
  - All completion verbs are non-blocking
  - No explicit completion channel creation
- What happens with explicit channel creation?
  - No change in verbs used so far! (Unlike cm channel)
  - Need to use new verbs to deal with the channel
    - **ibv_req_notify_cq()** to request events on the channel
    - **ibv_get_cq_event()** to block for events
    - **ibv_ack_cq_event()** to acknowledge events

Completion-channel data structure

- Purpose: to enable Channel Adaptor to notify program that work completions are in the CQ
- Data structure: **struct ibv_comp_channel**
- Fields visible to programmer:

fd	operating system file descriptor (small integer)
context	verbs field of associated cm_id

- System returns value for **fd** when structure is created

comp_channel setup/break-down

- Setup
  - Verb: **ibv_create_comp_channel()**
  - Parameter:
    - Context - verbs field of associated cm_id
  - Return value:
    - Pointer to new instance of **struct ibv_comp_channel**
- Break-down
  - Verb: **ibv_destroy_comp_channel()**
  - Parameter:
    - Pointer to **struct ibv_comp_channel** returned by **ibv_create_comp_channel()**

C 2011 OpenFabrics Alliance, Inc

our_create_comp_channel()

```
static int
our_create_comp_channel(struct our_control  *conn,  struct rdma_cm_id  *cm_id,
                        struct our_options  *options)
{
int     ret;
/* create a completion channel for this cm_id */
errno = 0;
conn->completion_channel = ibv_create_comp_channel(cm_id->verbs);
if (conn->completion_channel == NULL) {
        ret = ENOMEM;
        our_report_error(ret,  "ibv_create_comp_channel",  options);
} else {
        ret = 0;
        our_trace_ptr("ibv_create_comp_channel",  "created completion channel",
                                                    conn->completion_channel,  options);
        our_trace_ulong("ibv_create_comp_channel",  "assigned fd",
                                                conn->completion_channel->fd,  options);
}
return ret;
}       /* our_create_comp_channel */
```

our_await_completion() - part 1

```
int
our_await_completion(struct our_control *conn,
                        struct ibv_wc *work_completion,
                        struct our_options *options)
{
int         ret;

/* wait for next work completion to appear in completion queue */
do   {
errno = 0;
if (conn->latest_cm_event_type != RDMA_CM_EVENT_ESTABLISHED) {
     /* peer must have disconnected unexpectedly */
     ret = conn->latest_status;
     if (ret == 0) {
          ret = ECONNRESET;/* Connection reset by peer */
     }
} else {/* see if a completion has already arrived */
     ret = ibv_poll_cq(conn->completion_queue, 1, work_completion);
```

C 2011 OpenFabrics Alliance, Inc

our_await_completion() - part 2

```
        if (ret == 0) {
            conn->n_1st_poll_zero++;
            /* no completion here yet, must wait for one */
            ret = our_wait_for_notification(conn, options);
            if (ret == 0) {
                errno = 0;
                ret = ibv_poll_cq(conn->completion_queue, 1, work_completion);
                if (ret == 0)
                    conn->n_2nd_poll_zero++;
                else
                    conn->n_2nd_poll_non_zero++;
            }
        } else {
            conn->n_1st_poll_non_zero++;
        }
    }
} while (ret == 0);
```

our_await_completion() - part 3

```
/* should have gotten exactly 1 work completion */
if (ret != 1) {
    /* ret cannot be 0, and should never be > 1, so must be < 0 */
    our_report_error(ret, "ibv_poll_cq", options);
} else {
    ret = our_check_completion(conn, work_completion, options);
}
return ret;
}   /* our_await_completion */
```

our_wait_for_notification() - part 1

```
static int
our_wait_for_notification(struct our_control *conn, struct our_options *options)
{
struct ibv_cq   *event_queue;
void            *event_context;
int             ret;
/* wait for a completion notification (this verb blocks) */
errno = 0;
ret=ibv_get_cq_event(conn->completion_channel, &event_queue, &event_context);
if (ret != 0) {
        our_report_error(ret, "ibv_get_cq_event", options);
        goto out0;
}
conn->cq_events_that_need_ack++;
if (conn->cq_events_that_need_ack == UINT_MAX) {
        ibv_ack_cq_events(conn->completion_queue, UINT_MAX);
        conn->cq_events_that_need_ack = 0;
}
```

C 2011 OpenFabrics Alliance, Inc

our_wait_for_notification() - part 2

```
/* request notification when next completion arrives into empty completion queue.
 * See examples on "man ibv_get_cq_event" for how an “extra event” may be
 * triggered due to a race between this ibv_req_notify() and the subsequent
 *  ibv_poll_cq() that empties the completion queue.
 * The number of occurrences of this race will be recorded in completion_stats[0]
 * and will be printed as the value of work_completion_array_size[0].
 */
errno = 0;
ret = ibv_req_notify_cq(conn->completion_queue, 0);
if (ret != 0) {
	our_report_error(ret, "ibv_req_notify_cq", options);
}
out0:
return ret;
}	/* our_wait_for_notification */
```

Ping-Pong example ping-sr-3

- •Uses notifications to block for completion events on explicitly created completion channel
  - –Reduces CPU utilization by eliminating busy waiting
  - –But round-trip latency goes up
- •Run this
- •Walk code as necessary

C 2011 OpenFabrics Alliance, Inc

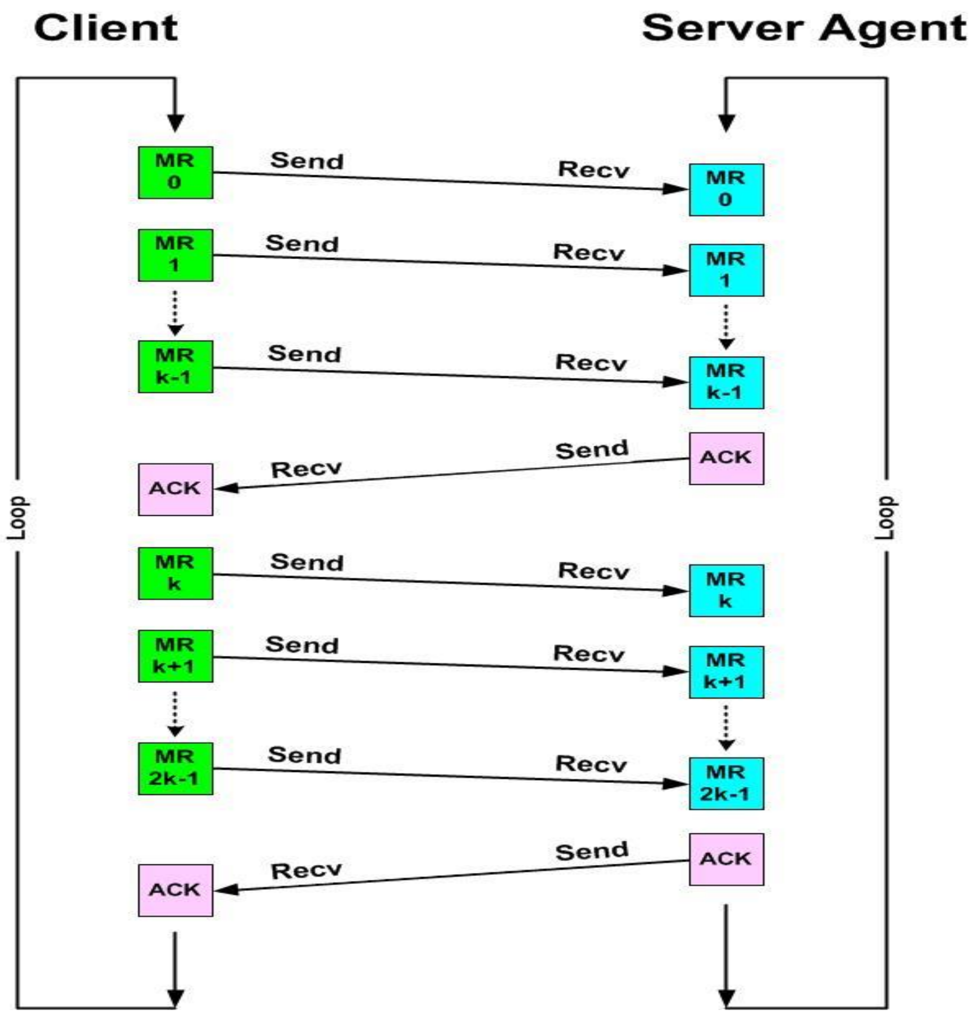
Ping-Pong example ping-sr-4

- Set both cm event channel and completion channel into O_NONBLOCK mode
- Use POSIX **poll()** to wait for both cm events and completion events
- Closest in style to “normal” sockets synchronous programming
- No need for extra thread
- Will look at this later if have time

Blast example blast-sr-2

- Blast using send/recv
- Explicitly creates **rdma_event_channel**
- Explicitly creates separate cm_event_thread
- Uses busy-waiting on **ibv_poll_cq()** to blast user data buffers from client to server

Blast using Send/Recv



Blast example blast-sr-2

- Blast using send/recv
- In Lab, run this and walk the code

C 2011 OpenFabrics Alliance, Inc

Blast exercise blast-sr-2e

- Blast program using completion channel and notifications
- In Lab, exercise for student

Finished with Send/Recv

- Lots of OFA verbs and OFA data structures
- Almost all of these are also used in RDMA_WRITE and RDMA_READ operations
- Still have to discuss
  - 2 more verbs in connection management component
  - the miscellaneous component of the pyramids

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id **rdma_get_local_addr** **rdma_get_peer_addr**	rdma_destroy_id rdma_destroy_event_channel
Misc		**rdma_get_devices** **rdma_free_devices** **ibv_query_devices**	

Miscellaneous CM Verbs

- **rdma_get_local_addr()**
- **rdma_get_peer_addr()**

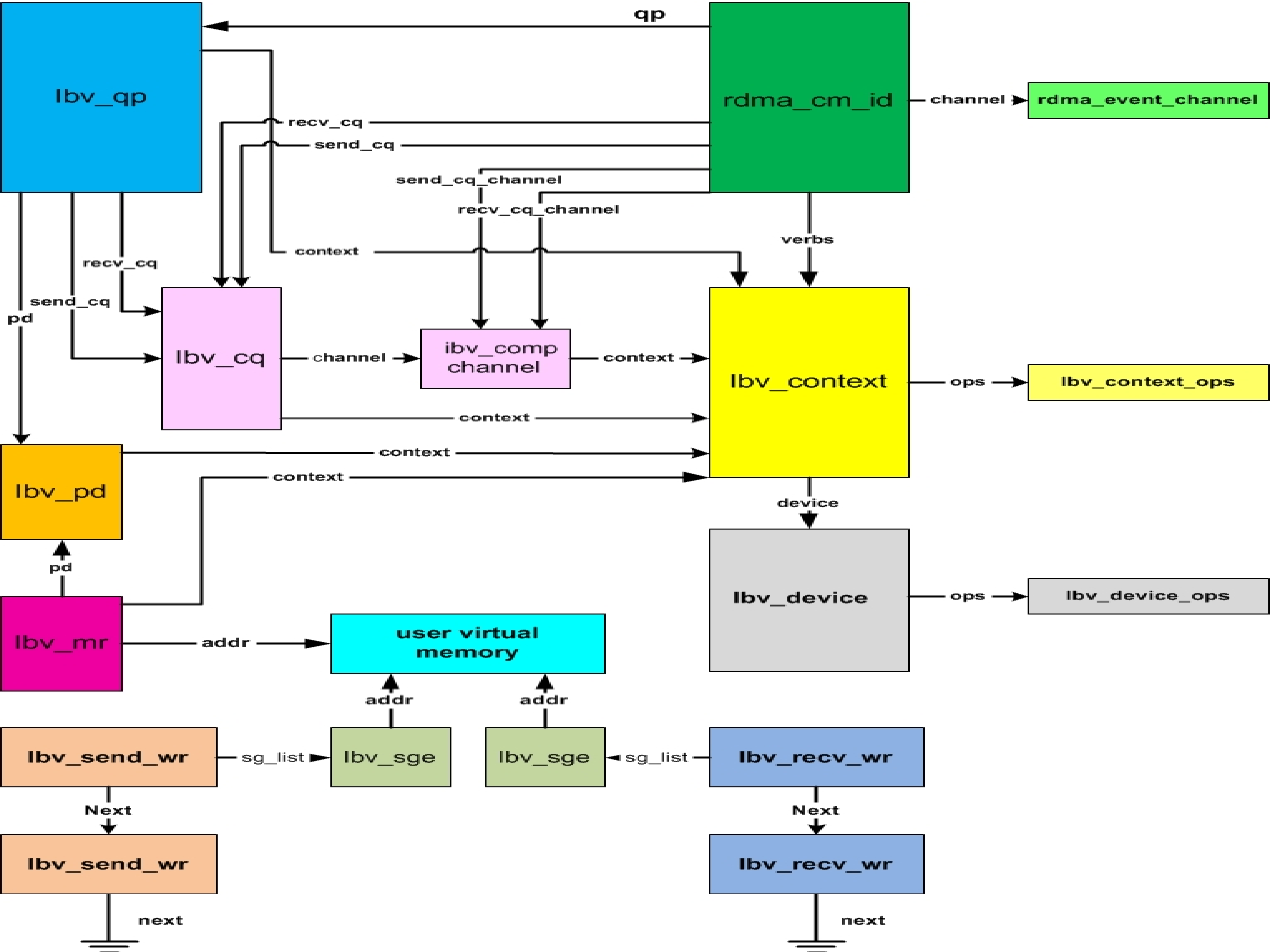
- Analogous to “normal” socket functions
- **getsockname()**
- **getpeername()**

C 2011 OpenFabrics Alliance, Inc

Misc. structures in DS pyramid

Layer	Structures
Transfer Posting	ibv_recv_wr ibv_send_wr ibv_sge ibv_qp ibv_qp_init_attr
Transfer Completion	ibv_cq ibv_wc ibv_comp_channel
Memory Registration	ibv_pd ibv_mr
Connection Management	rdma_cm_id rdma_conn_param rdma_cm_event rdma_event_channel
Misc	ibv_context Ibv_device ibv_device_attr

C 2011 OpenFabrics Alliance, Inc



qp
Ibv_qp
rdma_cm_id
channel
rdma_event_channel
recv_cq
send_cq
send_cq_channel
recv_cq_channel
context
verbs
recv_cq
send_cq
pd
Ibv_cq
channel
ibv_comp channel
context
Ibv_context
ops
Ibv_context_ops
context
context
Ibv_pd
context
device
pd
Ibv_device
ops
Ibv_device_ops
Ibv_mr
addr
user virtual memory
addr
addr
Ibv_send_wr
sg_list
Ibv_sge
Ibv_sge
sg_list
Ibv_recv_wr
Next
Next
Ibv_send_wr
Ibv_recv_wr
next
next

Struct ibv_context

- Referred to by many other structures already covered
- Contains 2 important fields

ops	big “jump table” for the verbs
device	structure representing the logical device

- Normally not used directly by programmer

–but see next demo

struct ibv_device

- Fundamental representation of CA
- Contains interesting fields

ops	“jump table” to allocate/free context
name	kernel name for the device
dev_name	name for the verbs
node_type	enum ibv_node_type
transport_type	enum ibv_transport_type

- Normally not used by programmer
  - But see next example

Getting a list of RDMA devices

- Purpose: to determine the names (and other info) of RDMA devices on a system
- Verb: **rdma_get_devices()**
- Parameter:

–integer in which number of RDMA devices is returned

- Return value

pointer to list of pointers to **struct ibv_context**

Freeing device list

- Verb: **rdma_free_devices()**
- Parameter:

  list pointer returned by **rdma_get_devices()**

ibv_query_device()

- Purpose: to return a structure containing information about an RDMA device
- Verb: **ibv_query_device()**
- Parameters:
  - pointer to **struct ibv_context**
  - pointer to **struct ibv_device_attr**
- System fills in fields of **device_attr** with information about RDMA device pointed to by **context**

struct ibv_device_attr

- Many fields visible to programmer
  - see definition in <infiniband/verbs.h>
- Each field provides information about the device
  - Most fields contain maximum number of a resource
- Useful for identification and administrative purposes

C 2011 OpenFabrics Alliance, Inc

Example devices

- Stand-alone program to print list of RDMA devices available on the host
- Uses
  - **rdma_get_devices()**
  - **rdma_free_devices()**
  - **ibv_query_device()**

and
  - **struct ibv_device_attr**
- Run this
- Walk the code

C 2011 OpenFabrics Alliance, Inc

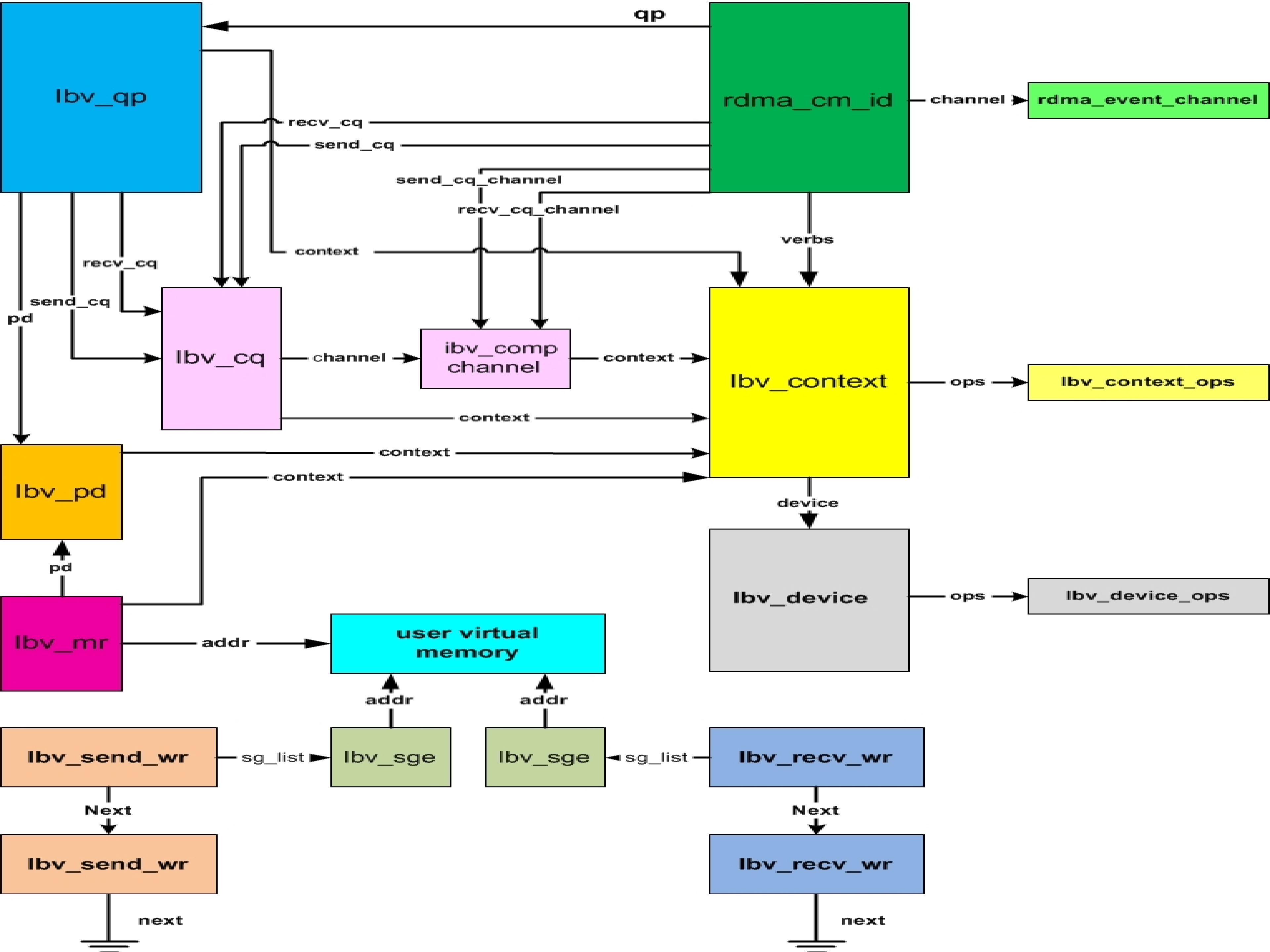
End of Send/Recv

- Almost all data structures and verbs presented so far will still be used for RDMA_WRITE and RDMA_READ transfers
- Concept is different

Complete Data Structure Pyramid

Transfer Posting	ibv_recv_wr ibv_send_wr ibv_sge ibv_qp ibv_qp_init_attr
Transfer Completion	ibv_cq ibv_wc ibv_comp_channel
Memory Registration	ibv_pd ibv_mr
Connection Management	rdma_cm_id rdma_conn_param rdma_cm_event rdma_event_channel
Misc	ibv_context ibv_device ibv_device_attr

	Setup	Use	Break-Down
Transfer Posting	rdma_create_qp	Ibv_post_recv Ibv_post_send	rdma_destroy_qp
Transfer Completion	ibv_create_cq ibv_create_comp_channel	Ibv_poll_cq Ibv_wc_status_str Ibv_req_notify_cq Ibv_get_cq_event Ibv_ack_cq_events	ibv_destroy_cp ibv_destroy_comp_channel
Memory Registration	Ibv_alloc_pd Ibv_reg_mr		Ibv_dealloc_pd Ibv_dereg_mr
Connection Management	rdma_create_id rdma_create_event_channel	rdma_resolve_addr rdma_resolve_route rdma_connect rdma_disconnect rdma_bind_addr rdma_listen rdma_get_cm_event rdma_ack_cm_event rdma_event_str rdma_accept rdma_reject rdma_migrate_id rdma_get_local_addr rdma_get_peer_addr	rdma_destroy_id rdma_destroy_event_channel
Misc		rdma_get_devices rdma_free_devices ibv_query_devices	



Ibv_qp
qp
rdma_cm_id
channel
rdma_event_channel
recv_cq
send_cq
send_cq_channel
recv_cq_channel
context
verbs
recv_cq
send_cq
pd
Ibv_cq
channel
ibv_comp
channel
context
Ibv_context
ops
Ibv_context_ops
context
context
Ibv_pd
context
device
pd
Ibv_device
ops
Ibv_device_ops
Ibv_mr
addr
user virtual
memory
addr
addr
Ibv_send_wr
sg_list
Ibv_sge
Ibv_sge
sg_list
Ibv_recv_wr
Next
Next
Ibv_send_wr
Ibv_recv_wr
next
next

End of Subpart I