

# *User Direct Access Programming Library (uDAPL)*

Jie Cai

Department of Computer Science  
The Australian National University  
12/2008

---

---

# Outline

- Introduction
    - What is DAPL and uDAPL?
    - Terminology and Objects acronymns
    - Architecture and models
  - How to use uDAPL? (examples)
    - Ping-pong program with uDAPL and MPI
    - RDMA program with uDAPL
- 
-

# *What is DAPL?*

- DAT Collaborative is an industry group that has formed in order to define and submit for standardization a set of transport-independent, platform-independent Application Programming Interfaces that exploit the RDMA capabilities of next-generation interconnect technologies.
  - DAPL v1.0 released at 2002.
  - DAPL v1.2 released at 2004.
  - DAPL v2.0 released at 2007.

## *What is DAPL? (cont.)*

- Direct Access Programming Library (DAPL) contains a kernel and user space specifications in C programming language for RDMA transports, which refer to kDAPL and uDAPL respectively.
  - kDAPL and uDAPL APIs are an attempt to create a portable set of APIs for all RDMA networks.
  - User programs with uDAPL for RDMA communications.
- 
-

# *uDAPL*

- uDAPL is a generic application programming interface for network adapters capable of remote direct memory access (RDMA).
    - The uDAPL interface allows user space applications to work with RDMA adapters using a platform and transport independent API.
    - The uDAPL interface has been proposed for use in clustering, distributed systems, and network file systems.
    - Other communication library/standard for clustering system: Message-Passing Interface (MPI).
- 
-

# *Terminology (1)*

- Interface Adapter (IA): A host resident device that transfers message to and from the host memory associated with a specific Endpoint and a Fabric.
  - Consumer Notification Object (CNO): A IA-scope object that can be associated with a set of Event Dispatchers so that under certain conditions, arrival of Notification Events on those Event Queue causes activation of the CNO.
  - Data Transfer Operation (DTO): The requested data movement transfer submitted to a DAT Provider
- 
-

## *Terminology (2)*

- Endpoint (EP): The local part of a connection that supports posting DTO requests and receives.
  - Event: A structure or record that is delivered to the consumer through an Event Dispatcher to provide notice of some kind.
    - DTO completions, connections state changes, asynchronous errors, and software events.
  - Event Dispatcher (EVD): A Object that conceptually merges events from one or more Event Streams.
- 
-

## *Terminology (3)*

- **Event Stream:** A source of events for EVD: DTO completions, Connection Requests for passive side, connection reject notifications for active side, connection establishment completion notifications, disconnection notification and so on..
  - **LMR Triplet:** A type used to specify a slice within a Local Memory Region (LMR). Each LMR Triplet specifies the LMR context, the virtual address and a size.
  - **RMR Triplet:** Remote Memory Region Triplet.
- 
-

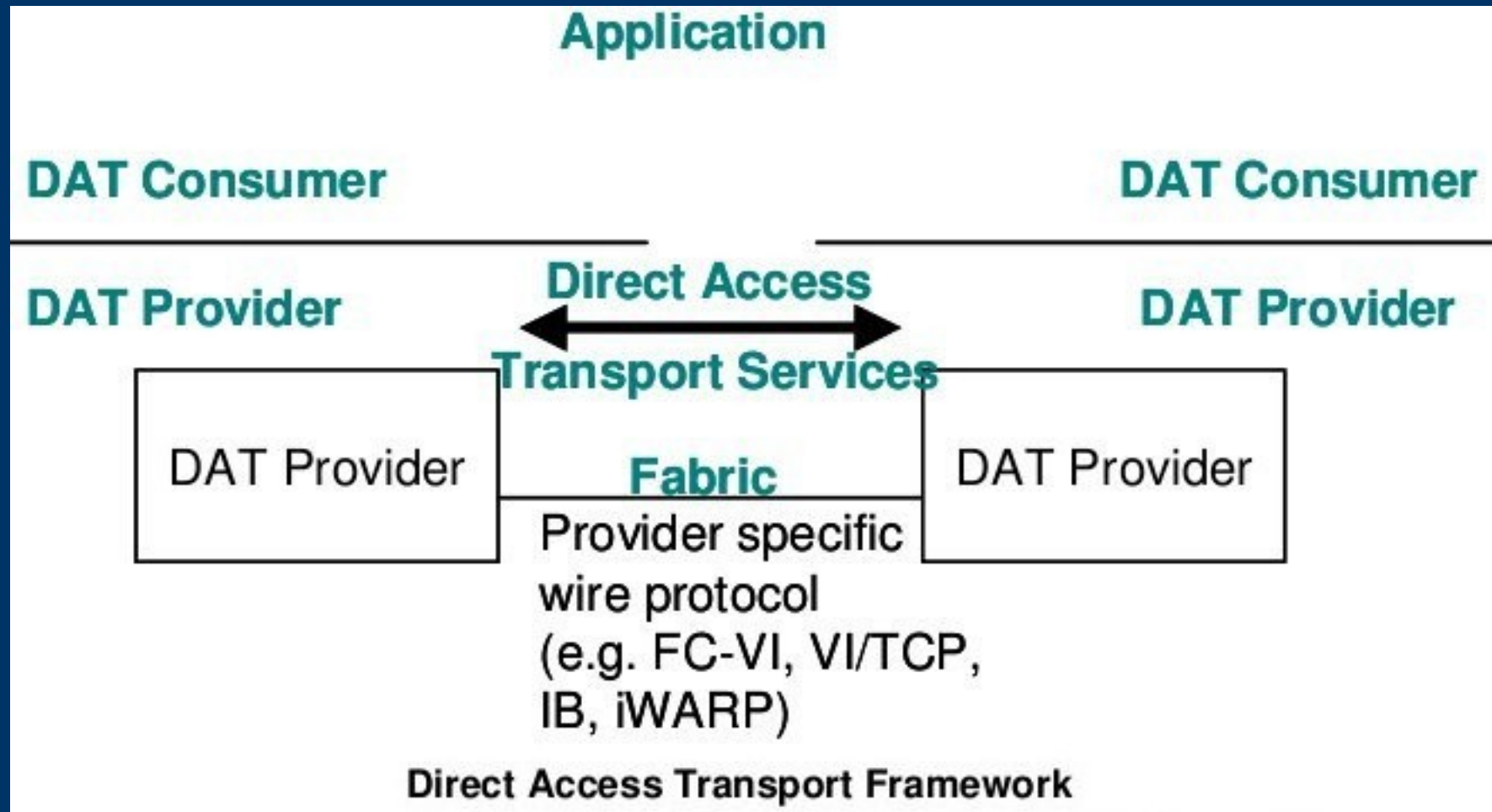
## *Terminology (4)*

- Protection Zone (PZ): A mechanism for association Endpoint and registered LMR and RMR memory of an IA that defines protection for local and remote memory access by DTO operations.
  - PSP: Public Service Point
- 
-

# ***DAT Objects – Acronymns***

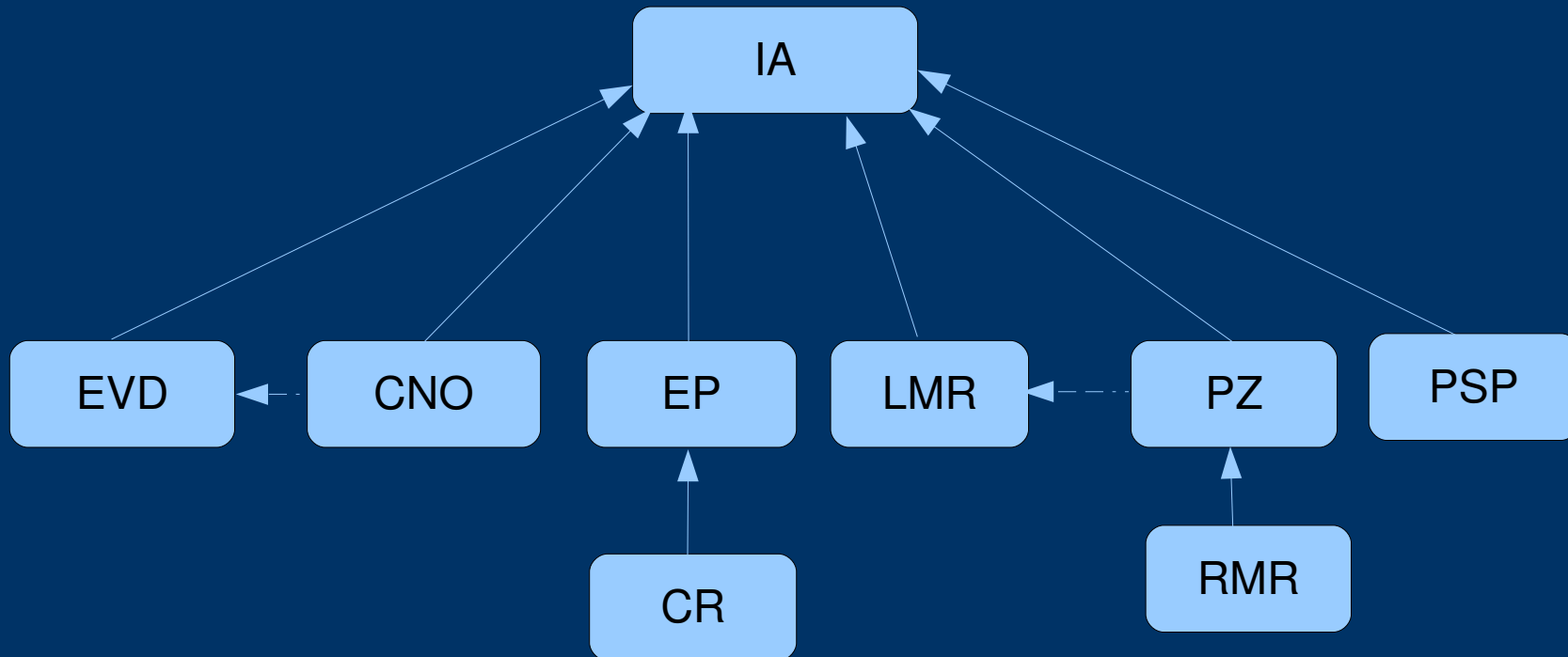
- CNO – consumer notification object
  - CR – connection request
  - EP – endpoint
  - EVD – event dispatcher
  - IA – interface adapter
  - LMR – local memory region
  - PSP – public service point
  - PZ – protection zone
  - RMR – remote memory region
  - RSP – reserved service point
- 
-

# Architecture (1) – General Model



# Architecture (2) – Objects

- DAT Objects are managed in a root-leaves fashion.

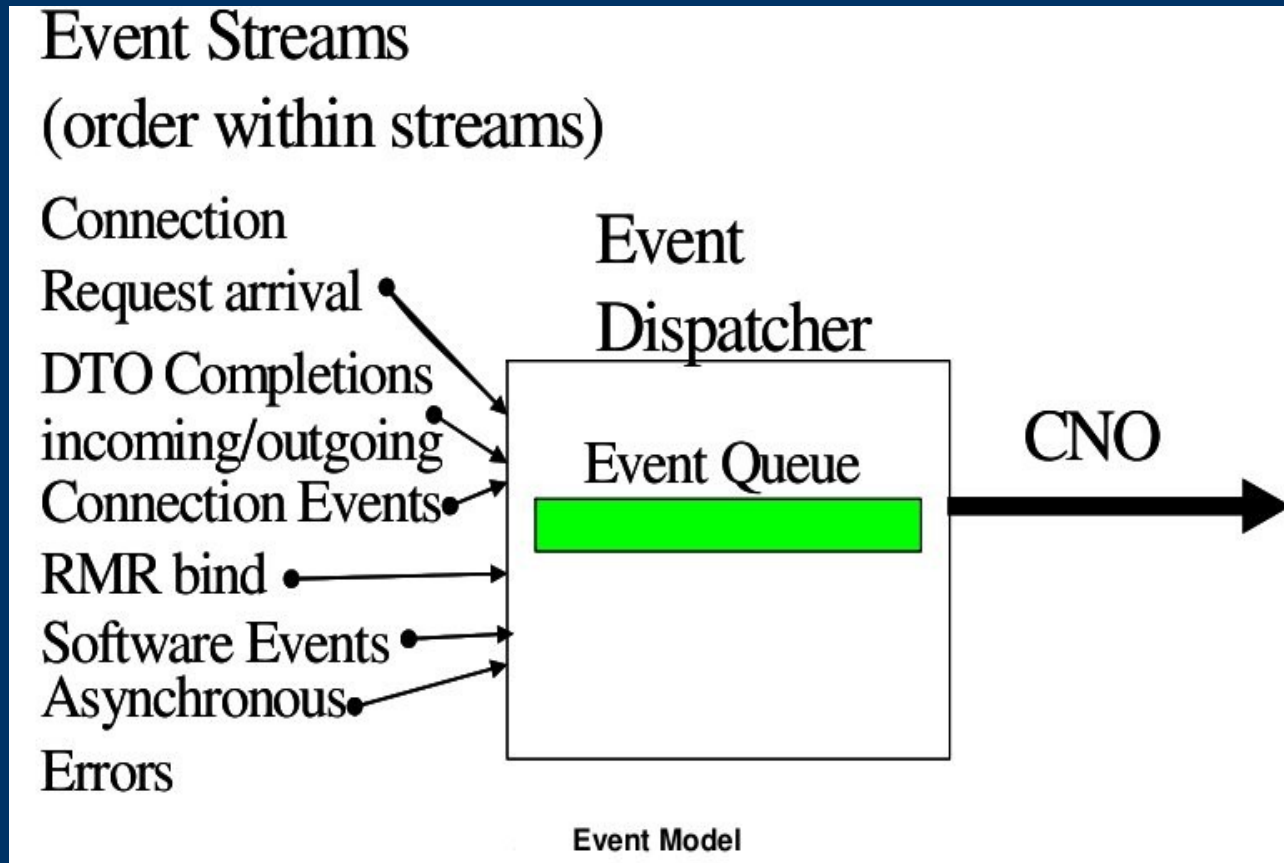


# Architecture (3)

- Local Resource Management
    - dat\_ia\_open, dat\_ia\_close, dat\_ia\_query:  
Open/Close/Query an interface adapter,
  - Event Management
    - Event Dispatcher (EVD) delivers events and event streams feed into EVD
      - Completions
      - Connection requests
      - Connection events: establishment, disconnect, timeout
      - Asynchronous errors
      - User events
- 
-

# Architecture (4) – EVD Model

- dat\_cno\_create  
dat\_cno\_free  
dat\_cno\_wait
- dat\_evd\_create  
dat\_evd\_free  
dat\_evd\_wait  
dat\_evd\_dequeue



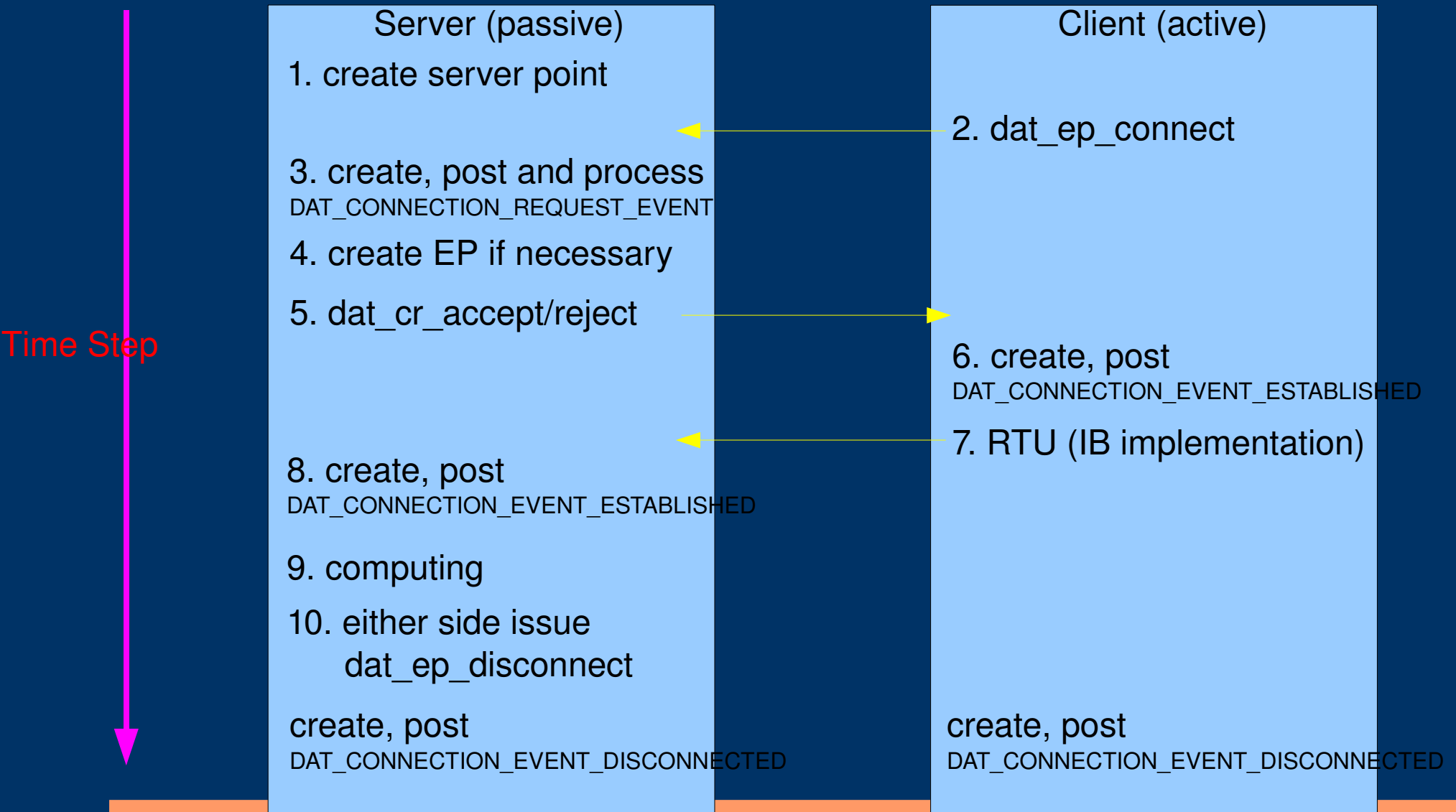
# Architecture (5) – Connection Mgmt

- Connection Management
    - Support client/server and peer-to-peer connection models
    - Uses standard socket address for IA
    - Connection Qualifiers are used to associate connection requests with the providing service
    - Public service point (PSP) is limited to a single IA
      - Allow consumer to listen for CR on a specified connection qualifier, which is advertised by a name service
      - Listen request is persistent, allow multiple connections.
      - `dat_psp_create` is used to create PSP
    - Reserved service point (RSP): peer-to-peer connection
      - `dat_rsp_create`
- 
-

# Architecture (6) – Connection Mgmt

- Connection Management (Cont.)
  - Connection Request (CR): `dat_cr_accept`
    - Establishes a connection and destroy CR object
    - User specifies endpoint, unless it is specified by CR
    - User may pass private data to remote user
  - Endpoint (EP)
    - `dat_ep_create/free`: create and release an EP
    - `dat_ep_connection`: initial a connection request to specified remote service
    - `dat_ep_reset`: reset EP to unconnected state
    - `dat_ep_disconnect`: disconnection ep
    - `dat_ep_post_send/recv`: send/recv data transfer
    - `dat_ep_post_rdma_write/read`: RDMA write/read data transfer
    - `dat_ib_post_rdma_write_immed`: RDMA write with immediate data

# Architecture (7) – Connection Diagram




# Architecture (8) – Memory Mgmt

- Memory Management
    - dat\_pz\_create
      - Allocates a protection zone on an interface adapter
      - Associates endpoints with local and remote memory regions
    - dat\_lmr\_create
      - Registers memory with a protection zone for use with an endpoint
    - dat\_rmr\_create
      - Allocates a remote memory region within a protection zone
      - No memory is associated with the RMR until bound
    - dat\_rmr\_bind
      - Enables a section of an LMR for remote access
- 
-

# *MPI Pingpong*

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myproc);
if (nprocs != 2) exit (1);
other_proc = (myproc + 1) % 2;

if (myproc == 0) {
    MPI_Send(snd_buf, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);
    MPI_Recv(rcv_buf, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
    &status);
} else {
    MPI_Recv(rcv_buf, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD,
    &status);
    MPI_Send(snd_buf, size/8, MPI_DOUBLE, other_proc, 0, MPI_COMM_WORLD);
}
```



# PingPong Example (1)

## Initialization Phase (1):

```
h_async_evd = DAT_HANDLE_NULL;  
ret = dat_ia_open( provider, 8, &h_async_evd, &h_ia );
```

```
ret = dat_pz_create(h_ia, &h_pz);
```

```
region.for_va = rbuf;  
ret = dat_lmr_create( h_ia, DAT_MEM_TYPE_VIRTUAL,  
                    region, buf_len*burst, h_pz,  
                    DAT_MEM_PRIV_ALL_FLAG,  
                    DAT_VA_TYPE_VA, &h_lmr_rcv,  
                    &lmr_context_rcv, &rmr_context_rcv,  
                    &registered_size_rcv,  
                    &registered_addr_rcv );
```

min length of  
asynchronous event  
dispatcher queue

Length of region

# PingPong Example (2)

## Initialization Phase (2):

```
region.for_va = sbuf;  
ret = dat_lmr_create(h_ia, DAT_MEM_TYPE_VIRTUAL,  
                    region, buf_len*burst, h_pz,  
                    DAT_MEM_PRIV_ALL_FLAG,  
                    DAT_VA_TYPE_VA, &h_lmr_send,  
                    &lmr_context_send, &rmr_context_send,  
                    &registered_size_send,  
                    &registered_addr_send );
```

min length of event  
dispatcher queue

```
ret = dat_evd_create(h_ia, 10, DAT_HANDLE_NULL,  
                    DAT_EVD_CR_FLAG, &h_cr_evd);  
ret = dat_evd_create(h_ia, 10, DAT_HANDLE_NULL,  
                    DAT_EVD_CONNECTION_FLAG,  
                    &h_conn_evd );
```



# PingPong Example (4)

## Connection Establishment Phase (1):

Server name (obtained from command line argument or environment variables)

### Server side:

```
dat_psp_create(h_ia, conn_id, h_cr_evd,  
              DAT_PSP_CONSUMER_FLAG,  
              &h_psp );  
dat_evd_wait( h_cr_evd,  
              SERVER_TIMEOUT, 1, &event,  
              &nmore );  
h_cr =  
    event.event_data.cr_arrival_event_data.cr_handle;  
dat_cr_query( h_cr,  
              DAT_CSP_FIELD_ALL,  
              &cr_param);  
dat_cr_accept(h_cr, h_ep, 48,  
              cr_param.private_data);
```

### Client side:

```
getaddrinfo (servername, NULL, NULL,  
            &target)  
remote_addr =  
    *((DAT_IA_ADDRESS_PTR)target  
    ->ai_addr);  
for (i=0;i<48;i++) //set pdata  
    pdata[i]=i+1;  
dat_ep_connect(  
    h_ep, &remote_addr, conn_id,  
    CONN_TIMEOUT, 48,  
    (DAT_PVOID)pdata, 0,  
    DAT_CONNECT_DEFAULT_FLAG );
```

Size of  
pdata

QoS?

# *PingPong Example (5)*

Connection Establishment Phase (2): both side

```
dat_evd_wait( h_conn_evd, DAT_TIMEOUT_INFINITE, 1, &event, &nmore );

if ( event.event_number != DAT_CONNECTION_EVENT_ESTABLISHED ) {
    fprintf(stderr, "%d Error unexpected conn event : %s\n",
            getpid(),DT_EventToStr(event.event_number));
    return( DAT_ABORT );
}

dat_ep_query(h_ep, DAT_EP_FIELD_ALL, &ep_param);
```

# PingPong Example (6)

## Ping-pong Phase (1):

```
l_iov.lmr_context      = lmr_context_rcv;  
l_iov.virtual_address = (DAT_VADDR)rcv_buf;  
l_iov.segment_length  = buf_len;  
dat_ep_post_rcv( h_ep, 1, &l_iov, cookie,  
                DAT_COMPLETION_DEFAULT_FLAG );
```

Number of lmr\_triplet in  
l\_iov,  
l\_iov can be a array of  
lmr\_triplet

```
if(!server) {  
    *snd_buf = 0x55;  
    iov.lmr_context      = lmr_context_send;  
    iov.virtual_address = (DAT_VADDR)(unsigned long)snd_buf;  
    iov.segment_length  = buf_len;  
    dat_ep_post_send( h_ep, 1, &iov, cookie,  
                    DAT_COMPLETION_SUPPRESS_FLAG);  
}
```

Suppression flag do not  
raise an event



# *Compared with MPI PingPong?*

- uDAPL is a lower layer communication library compared to MPI.
  - Initialization for uDAPL is complex, however, this phase has been hidden to user in MPI.
  - MPI provide process rank and number of processes to user. However, user need to find out corresponding information manually in uDAPL.
  - uDAPL utilizes event-driven model, `dat_ep_post_send/recv` are non-blocking and asynchronous.
  - Data transfer with uDAPL does not require to specify data type as what MPI does.
- 
-

# RDMA Example (1) – with msg

## (v1.2)

```
if(server) {  
    dat_ep_post_rdma_write( h_ep, MSG_IOV_COUNT, l_iov, cookie, &r_iov,  
                           DAT_COMPLETION_SUPPRESS_FLAG );  
    iov.lmr_context    = lmr_context_send_msg;  
    iov.virtual_address = (DAT_VADDR)(unsigned long)rmr_send_msg;  
    dat_ep_post_send( h_ep, 1, &iov, cookie,  
                     DAT_COMPLETION_SUPPRESS_FLAG);  
}
```

```
if(!server) {  
    dat_ep_post_recv( h_ep, 1, &l_iov, cookie,  
                     DAT_COMPLETION_DEFAULT_FLAG );  
    dat_evd_wait(h_dto_rcv_evd, DTO_TIMEOUT, 1, &event, &nmore);  
}
```

Suppression flag do not  
raise an event

## *Discussion on RDMA with msg*

- This function is original uDAPL function for version 1.0 to 1.2.
  - Writer need to deliver a message to remote process to indicate the completion of RDMA write.
  - Similar to MPI-2 Active mode RMA:
    - Need cooperation from remote process
  - Utilizing IB provided 4-bytes inline data to avoid the completion message.
  - Implemented in version 2.0.
- 
-

# *RDMA Example (2) – with immed data (v2.0 IB extension)*

```
if(server) {
    immed_data = 0x1111;
    dat_ib_post_rdma_write_immed(h_ep, MSG_IOV_COUNT, 1_iov, cookie,
                                &r_iov, immed_data,
                                DAT_COMPLETION_DEFAULT_FLAG);
}
dat_evd_wait(dto_evd, DTO_TIMEOUT, 1, &event, &nmore);

DAT_IB_EXTENSION_EVENT_DATA *ext_event =
    (DAT_IB_EXTENSION_EVENT_DATA *)&event.event_extension_data[0];
if (ext_event->type == DAT_IB_RDMA_WRITE_IMMED) //server event
{
    null;
} else if (ext_event->type == DAT_IB_RDMA_WRITE_IMMED_DATA)
// client event
{
    immed_data_recv = ext_event->val.immed.data;
}
```

---

---

# *Discussion on RDMA with immed*

- One network latency has been eliminated by using `dat_ib_post_rdma_write_immed`.
  - Similar to MPI-2 Passive mode RMA:
    - uDAPL: `dat_evd_wait` to capture event and immed data.
    - MPI-2: `MPI_Win_wait(win)` to ensure origin process finished RDMA write.
- 
-