Writing Application Programs for RDMA using OFA Software Part 1

Presented in three parts

Open Fabrics Alliance

Copyright Statement

Instructor team

Paul Grun – pgrun@systemfabricworks.com

Bob Russell – rdr@iol.unh.edu

Rupert Dance – rsdance@soft-forge.com

Useful references

- InfiniBand Architecture – www.infinibandta.org
  - download of the IB specifications
- Introduction to InfiniBand™ for End Users
  - http://members.infinibandta.org/kwspub/Intro_to_IB_for_End_Users.pdf
  - free download, no registration required
- iWARP RFCs – www.ietf.org/rfc.html
  - RFC 5040 – A Remote Direct Memory Access Protocol Specification
  - RFC 5041 – Direct Data Placement over Reliable Transports
  - RFC 5044 – Marker PDU Aligned Framing for TCP Specification
  - others
- Mellanox User's Manual
  - http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

Course overview

"The purpose of this course is to provide students with knowledge and experience in writing application programs using RDMA."

Goals:

1. Provide a limited introduction to RDMA concepts and theory
2. Provide detailed classroom instruction in writing an application to the verbs API
3. Provide hands-on experience writing, compiling and executing an application program using the OFA stack and software tools.

A two part course

Part One: **Introduction to RDMA**	Part Two: **Programming with RDMA**
Programming with RDMA is different from sockets programming (for good reason) This part gives you the background needed to understand Part Two	Here are the details of how to write an application program to take full advantage of RDMA

Agenda – Introduction to RDMA

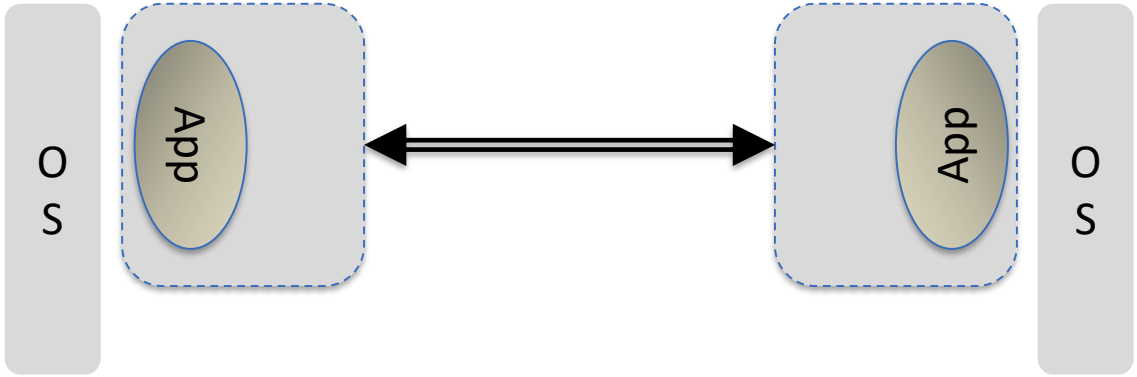
- Some key terminology
- Introduction – I/O architecture
- A Messaging Service
- Address translation and network operations
- The RDMA Architecture
- Verbs Introduction
- The OFED Stack
  - Data structures and Queues,
  - RDMA Protocols
  - Reliable Transport
  - Using the channel
  - Connections
- Introduction to wire protocols

Everything you need to fully understand Part Two…and one penny more

OPENFABRICS ALLIANCE

Useful terminology

RDMA – Remote Direct Memory Access



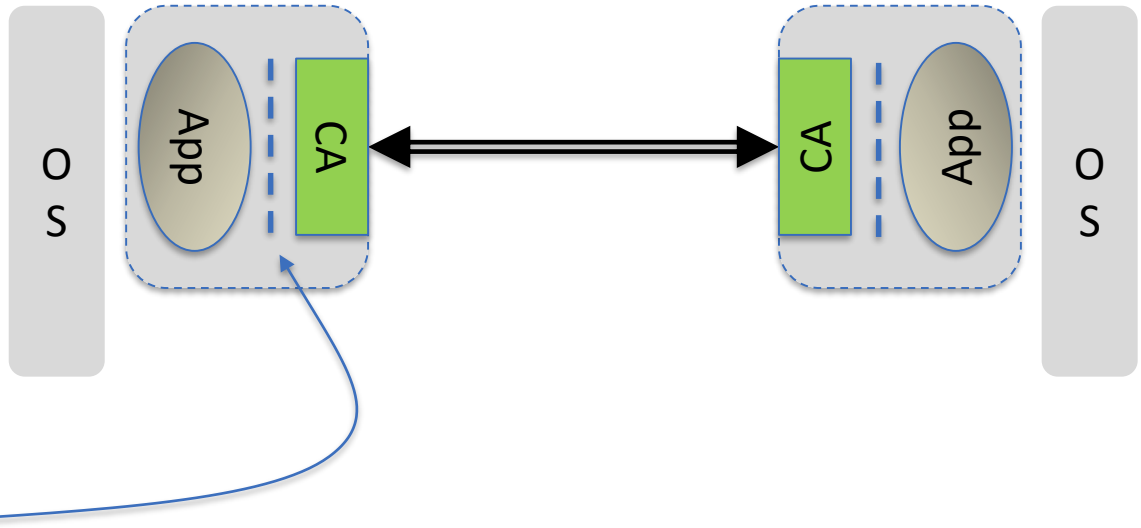
Remote Direct Memory Access: application-to-application communication

-Remote: communication at a distance

-Direct: does not require a 'higher authority' for each access

-Memory: virtual-to-virtual transfers…even across a network

Verbs

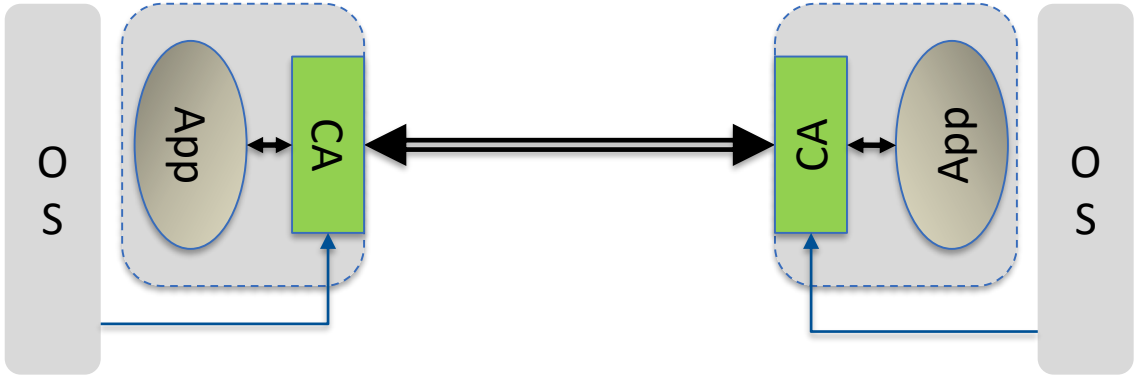


Verbs

An API used by an application to control and conduct an RDMA operation.

(It has another meaning too, which will be described later.)

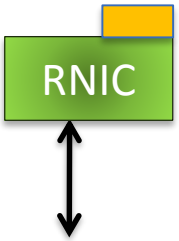
Channel Adapter - CA



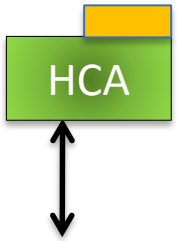
Channel Adapter (CA)

An I/O device that allows an application to conduct RDMA operations *directly*

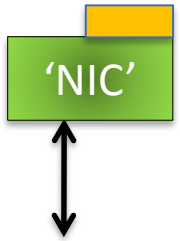
Three types of Channel Adapters



iWARP



InfiniBand



RoCE

Generically, all three are called *Channel Adapters (CA)*.
We'll see why in a minute.

We will use the expression Channel Adapter, or CA, consistently.

A channel adapter is comprised of both hardware and software.

There is wide latitude in h/w – s/w partitioning.

Requester, Responder

initiates and controls an RDMA operation

responds to requests

- A request flows from the requester to the responder
- A requester may request a READ or a WRITE operation
- A WRITE operation transfers data from the requester to the responder
- A READ operation transfers data from the responder to the requester

OPENFABRICS ALLIANCE

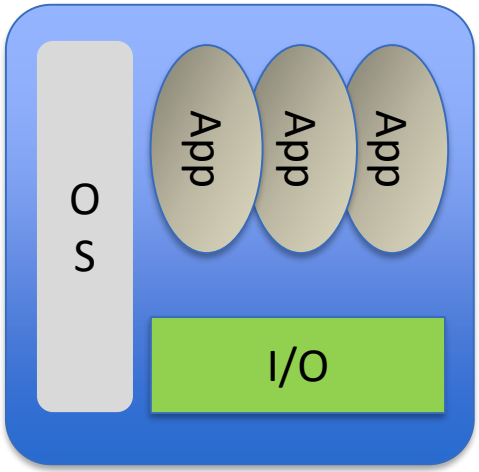
Introduction

Why RDMA?

Lots of reasons, and many value propositions. Architecturally…

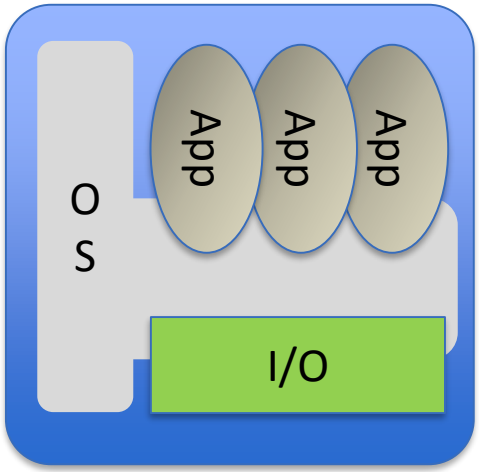
1. delivers low latency
   - stack bypass, copy avoidance
2. reduces CPU utilization
3. reduces memory b/w bottlenecks
4. delivers high bandwidth utilization
5. …

Quick Review of I/O



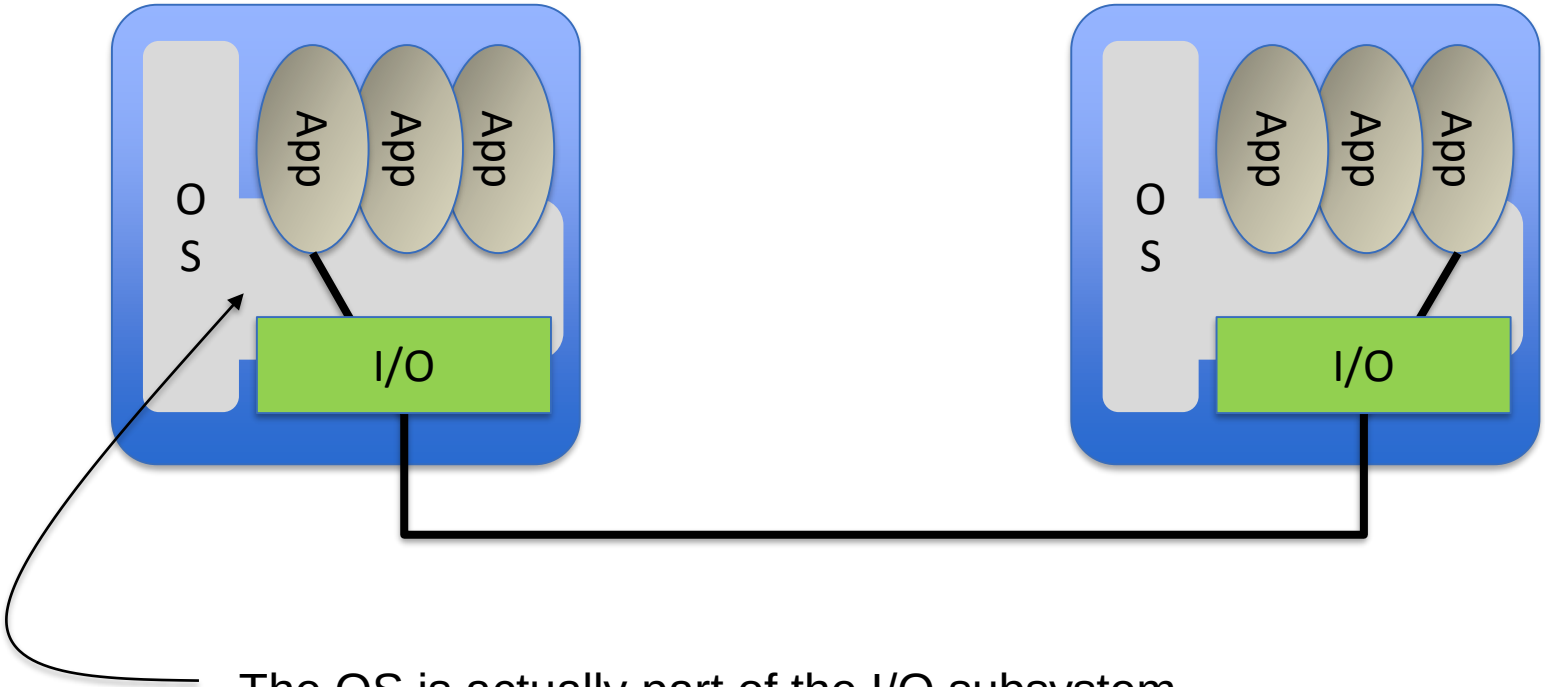
- Server hardware consists of a CPU/memory complex and an I/O subsystem
- Server hardware resources are owned by the OS
- The OS provides I/O services to its supported applications

Quick Review of I/O



- Applications exist in virtual memory
- I/O devices exist in physical memory
- All the above managed by the OS

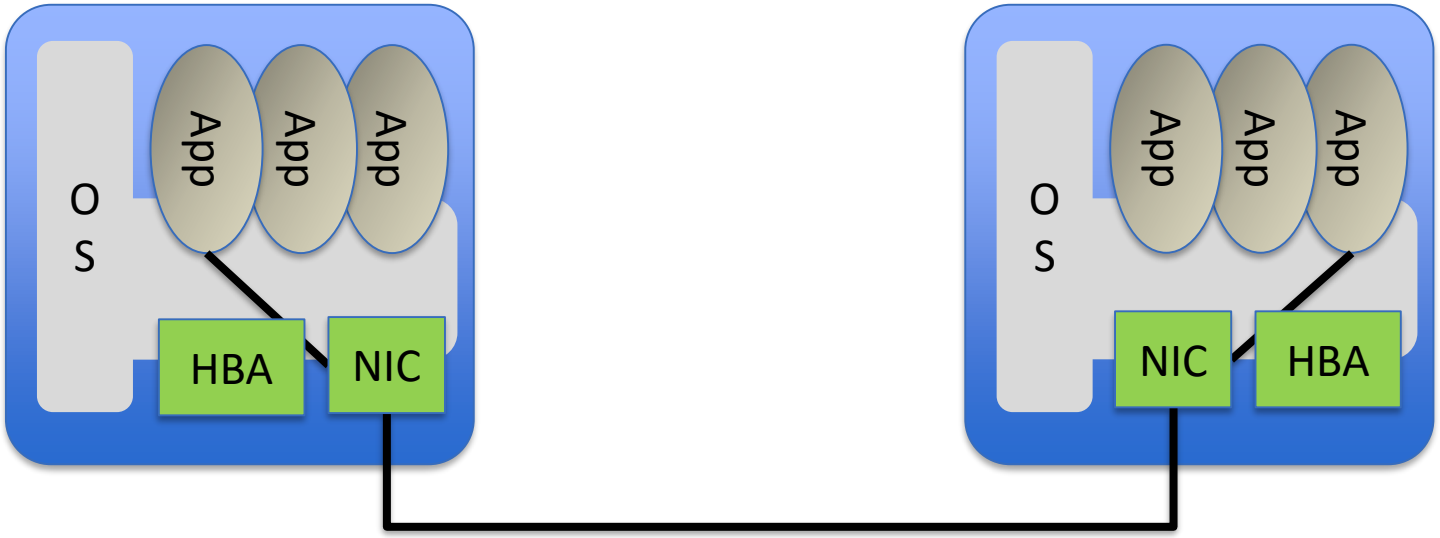
Quick Review of I/O



The OS is actually part of the I/O subsystem

For nearly every I/O operation, the application makes calls to the OS

Quick Review of I/O



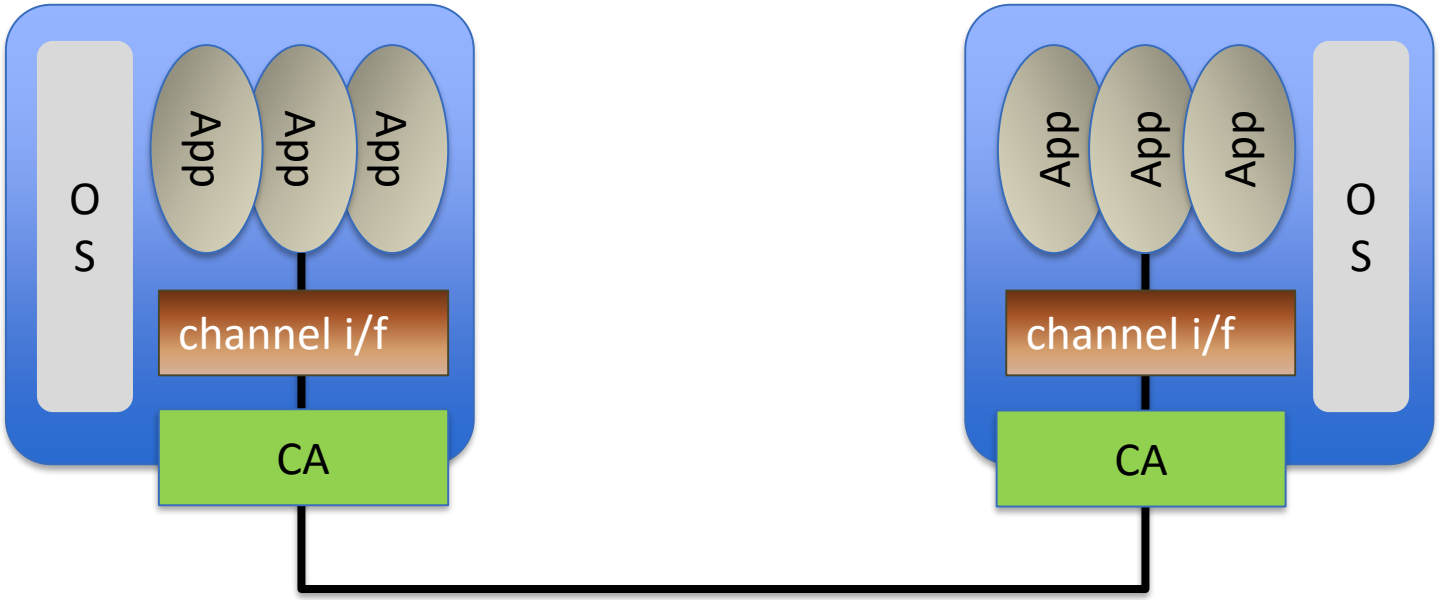
- Typically, applications perform three types of I/O - storage, networking and IPC
- Usually, there is a different wire, and I/O protocol for each
- And the OS is usually involved in each of them

A different I/O concept: Channel I/O

- An *I/O channel* is a conduit between applications
- The OS establishes the channel, thus a channel is isolated and protected
- *But the OS is not itself part of the channel!*

Channel I/O allows applications to communicate very efficiently

Physically



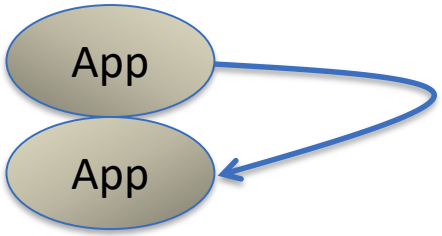
-Physically, a channel is created between two Channel Adapters
- An application accesses the channel via a thin software interface layer.
- This thin software layer is the *channel interface*
- ***Important: The channel interface runs in user space***

OPENFABRICS ALLIANCE

Message Passing

Message passing

- ‘Message Passing’ is a very common programming model
- Useful for communicating between any two processes
- A message is an abstract chunk of data, meaningful only to the processes

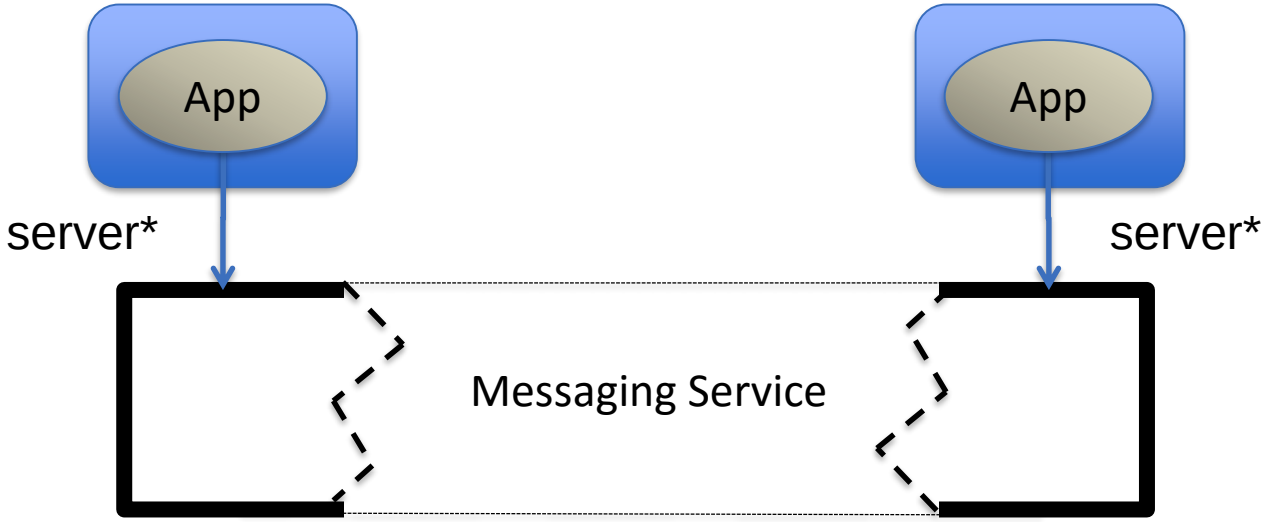


- Often used in parallel computing and interprocess communication
- Processes send and receive messages to each other
- Processes can use message passing to synchronize, by waiting for a message

A message passing service

-There are lots of ways to implement a message passing service
-Think of the *messaging service* as a black box
-The application doesn't need to know how messages are passed…
…but it will help if you do
-The application only needs to know what the service does and how to use it

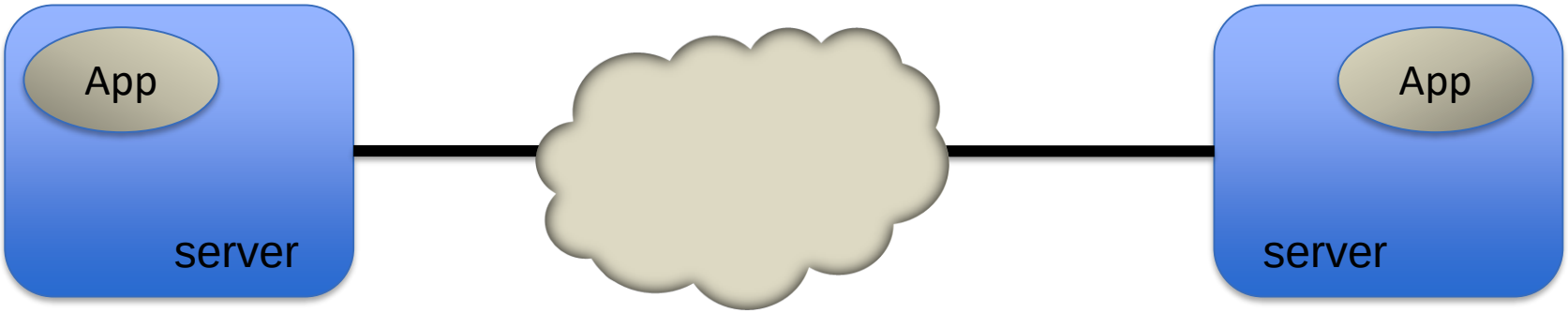
Message passing over a distance



- It gets more interesting when the two processes reside in disjoint *physical address spaces*
- This requires a networking construct to transport messages across a network

* the word ‘server’ implies a single physical address space under control of a single entity such as an OS (or a hypervisor)

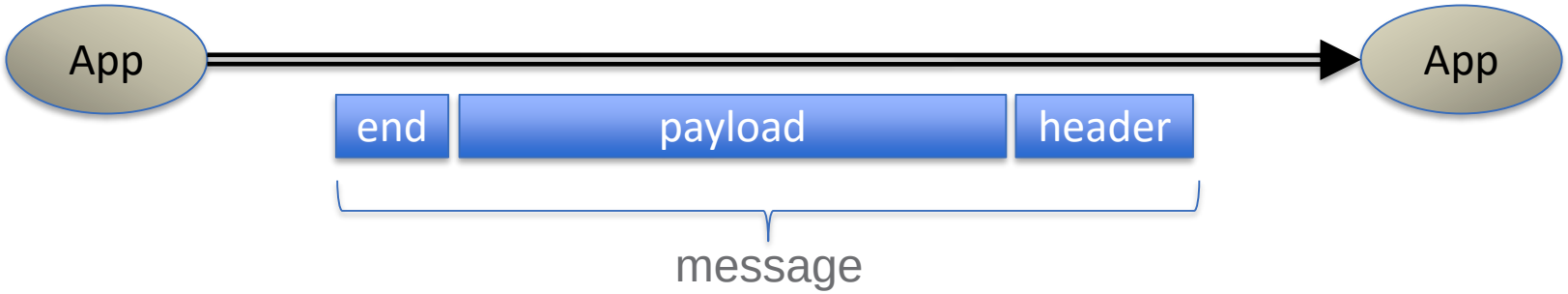
A remote message passing service



- RDMA is a way of transporting messages between servers or between a server and a client
- it has some characteristics of a network

RDMA = Remote Direct Memory Access

"RDMA" is a message passing paradigm



- The message itself is agnostic to the I/O protocol
  a payload is a payload is a payload…
- This means that messages can be transmitted and received simply and efficiently
- The I/O protocol is carried within the message *payload*
  e.g. SCSI, IPC, MPI…

Message passing for I/O

- An I/O protocol can be conducted using message passing
- I/O commands, data, and responses are passed as messages

Sockets based IPC (TCP)

An IPC application creates a queue of messages to be sent

IPC message
IPC message
IPC message
IPC message

The queued messages are copied to a socket buffer…

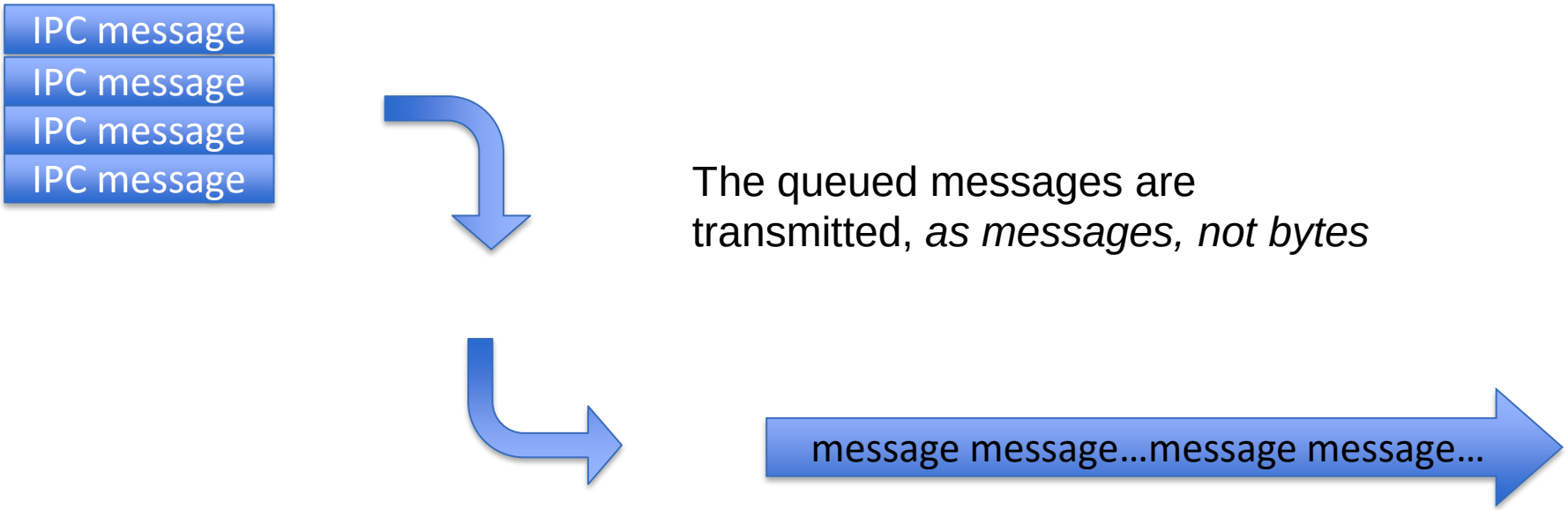
…message message message message message, message…

…and transmitted as a stream of bytes

byte byte … byte byte byte…

Message oriented IPC

An IPC application creates a queue of messages to be sent



Compared to sockets

IPC is naturally message-oriented
Sockets transfers IPC messages as a stream of byte.

Channel I/O, on the other hand, transfers messages.
It is oriented around a message as the discrete unit of work.

-a message is sent as a singular unit of work
-a message is received as a singular unit of work
-an application posts a message to the transport for transmission
-the transport delivers a completely received message to the application

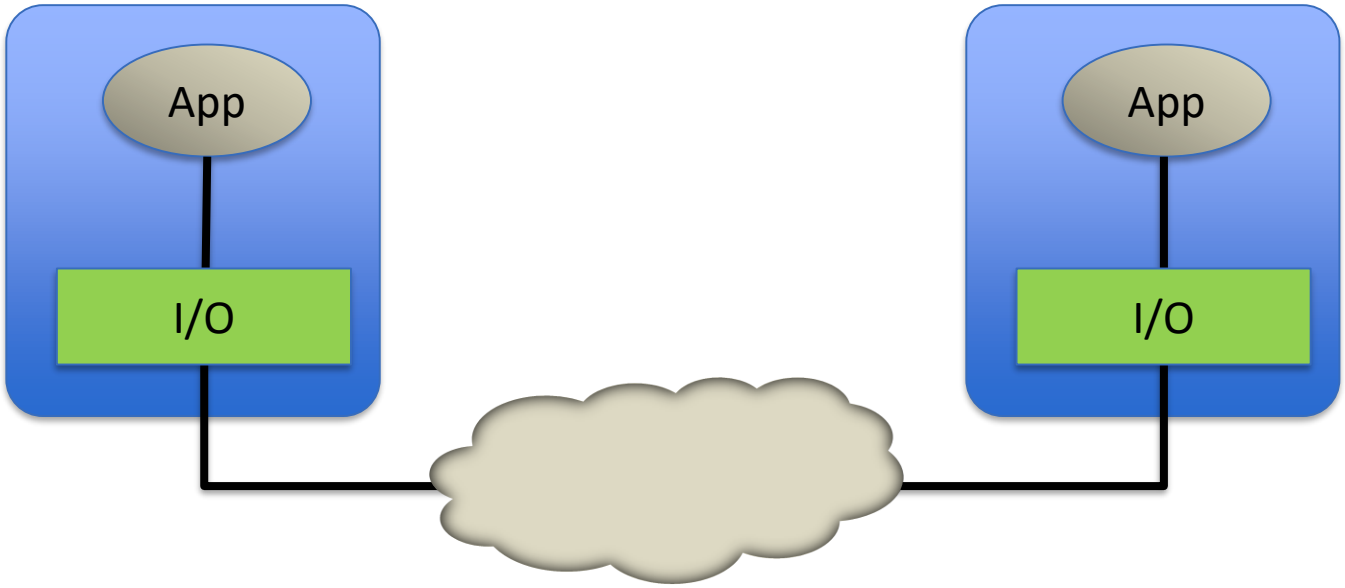
Messages can range in size from very small up to very, very large.
The transport handles the transmission of the whole message.

Don’t confuse a ‘message; with a ‘packet’...

OPENFABRICS ALLIANCE

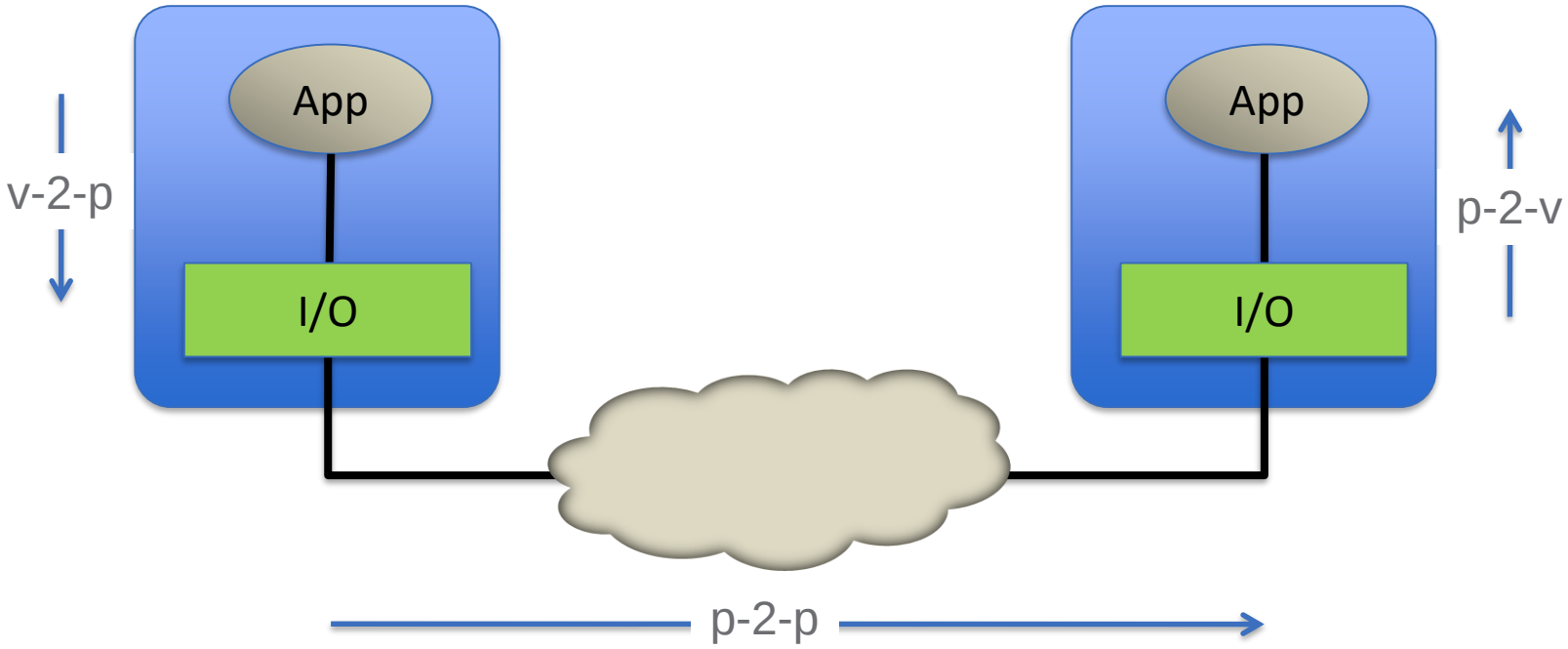
Address translation and network operations

Addressing



- Applications exist in virtual space
- Device adapters, (NICs, HBAs…) exist in physical space
- Each server occupies a disjoint physical address space

Three address translations

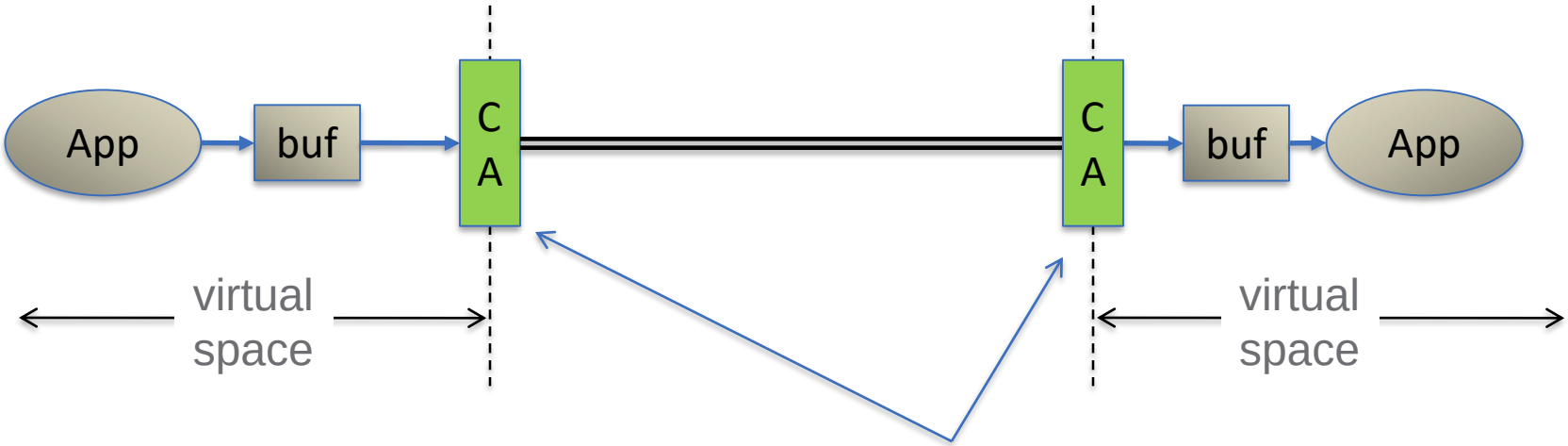


Therefore, three address translations are required end-to-end:
- a virtual-to-physical address translation
- a physical-to-physical translation, and
- a physical-to-virtual translation

OS-based address translation

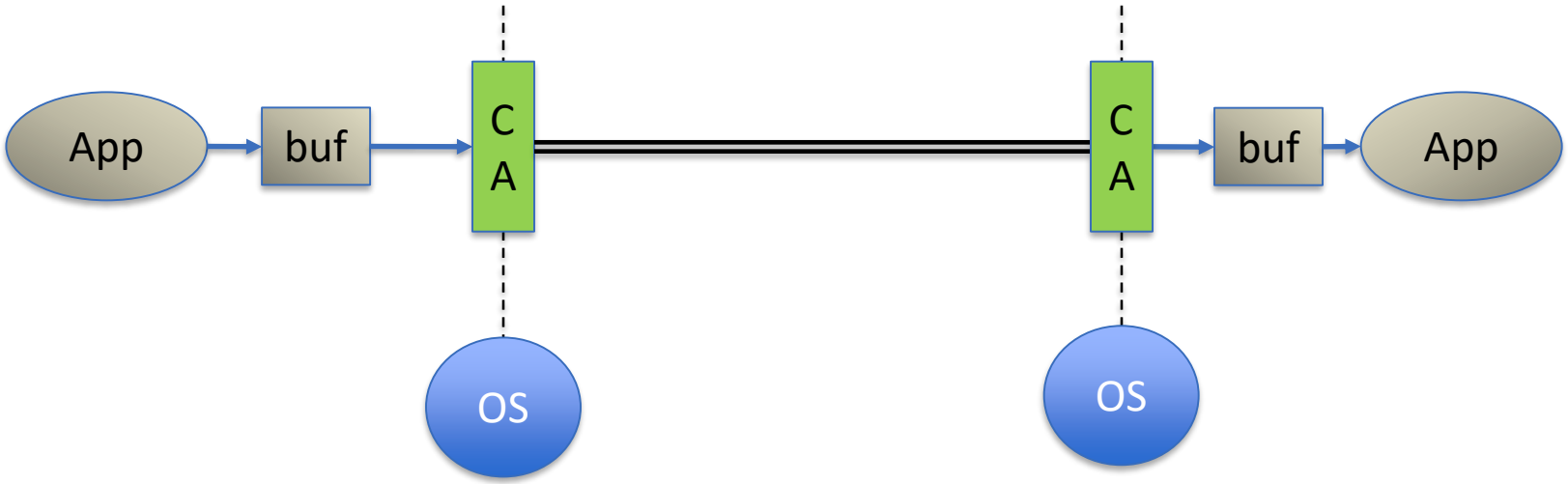
- The OS performs the v-p and p-v translations
- This ensures isolation of each application's virtual memory space
- The OS is integral to the I/O subsystem and is a part of moving every message

Channel address translation



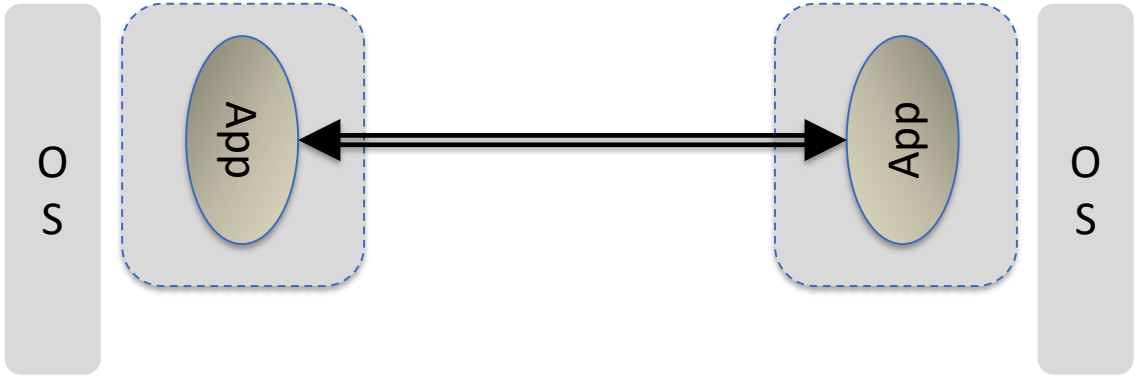
-The I/O channel performs the v-p, p-v translations
- In this way, messages are transferred without OS intervention

Channel protection mechanism



- Translation and protection tables in the CA are created by the OS
- Equivalent (at least) level of protection and isolation, but without requiring OS involvement
- The OS *supervises* the channel

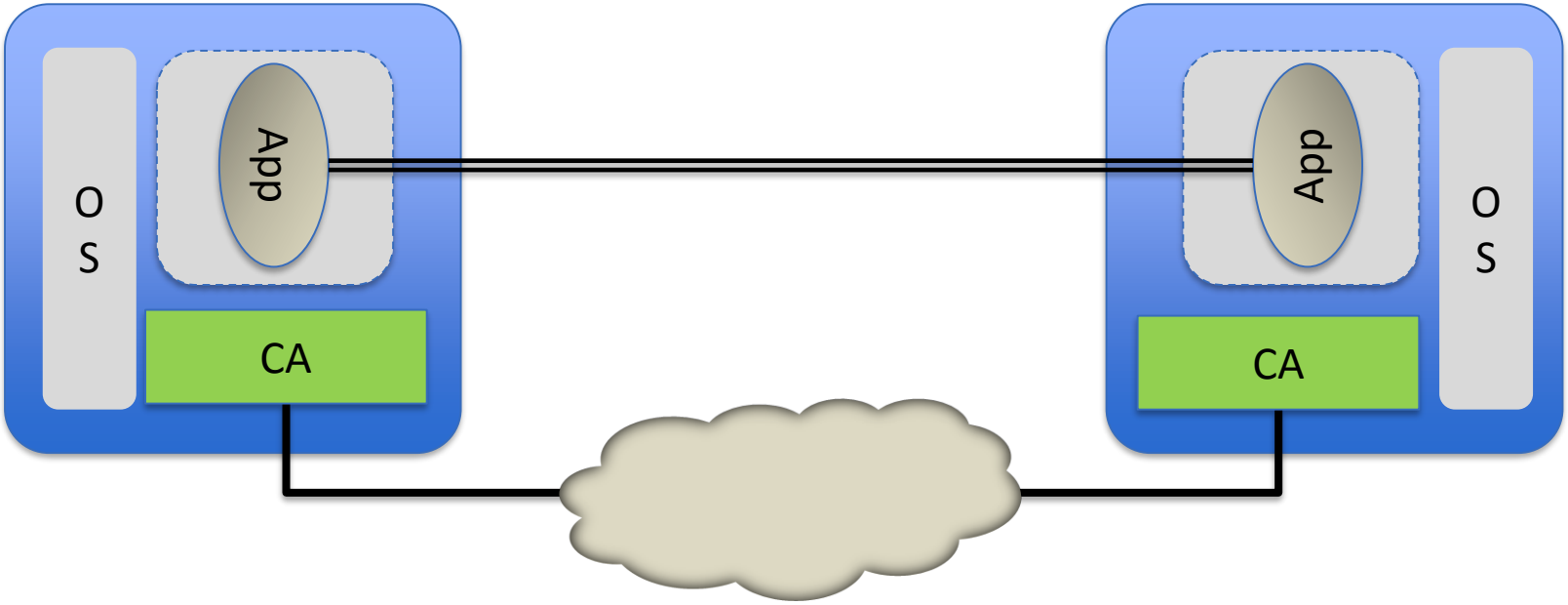
Virtual-to-virtual transfers



An I/O Channel connects virtual address spaces. The virtual address spaces can exist in disjoint physical address spaces.

Remote Direct Memory Access means that an application can directly *access remote virtual memory*

Network addressing

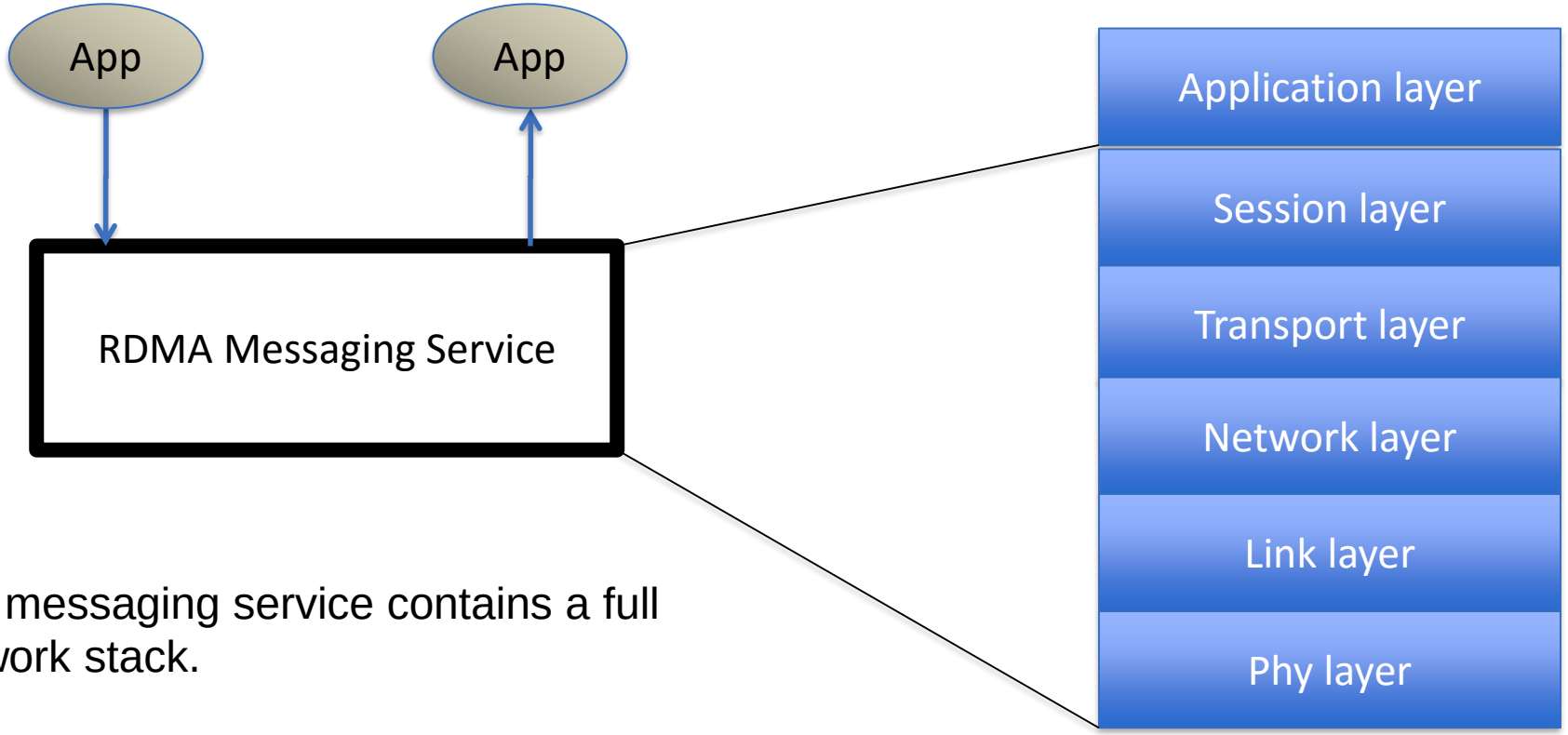


A channel spans disjoint physical address spaces
That's why the RDMA messaging service is built on a networking model
Networks allow communication between disjoint physical address spaces

OPENFABRICS ALLIANCE

Inside the messaging service RDMA architecture

The RDMA messaging service

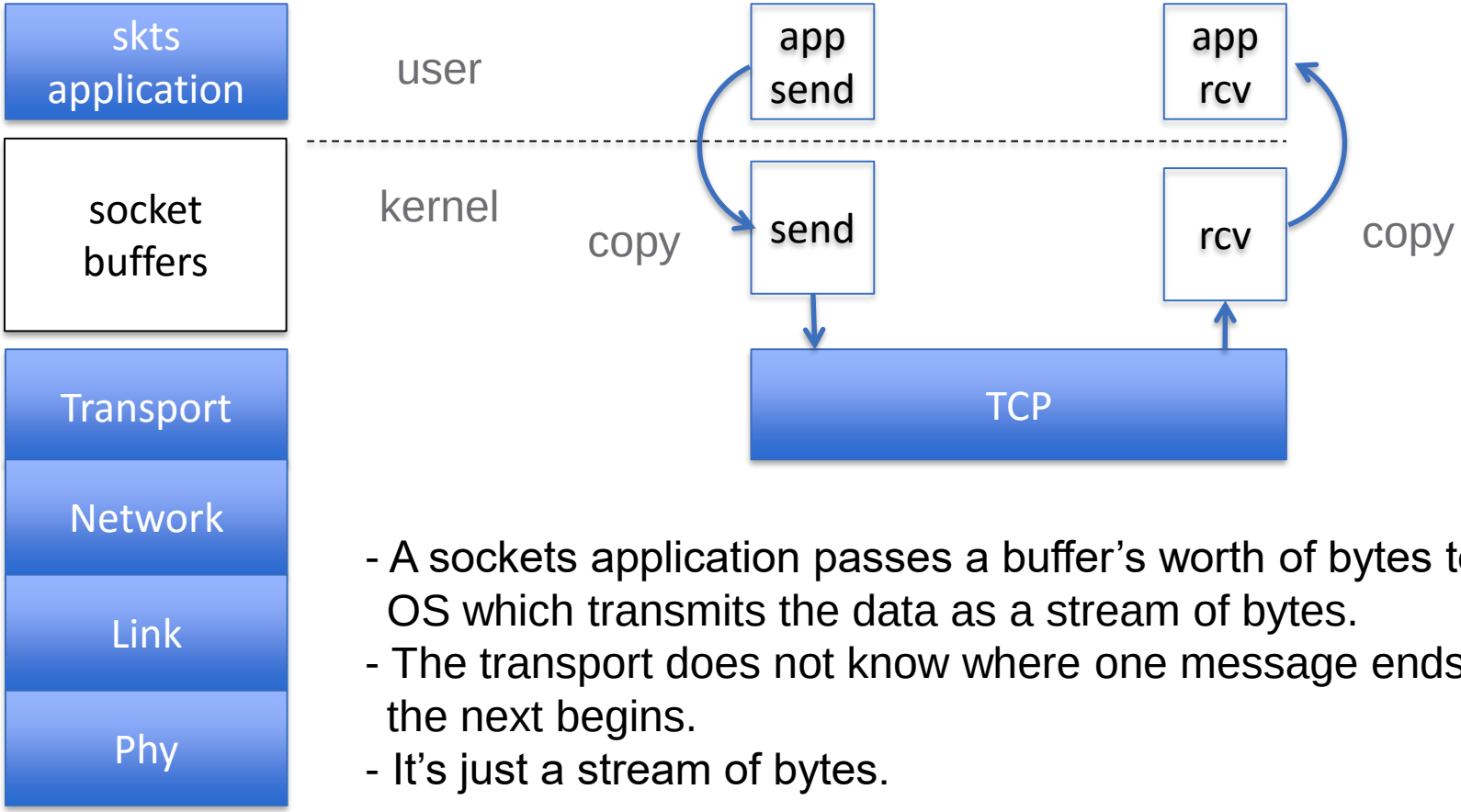


The messaging service contains a full network stack.

The question is: how does the application access the network stack??

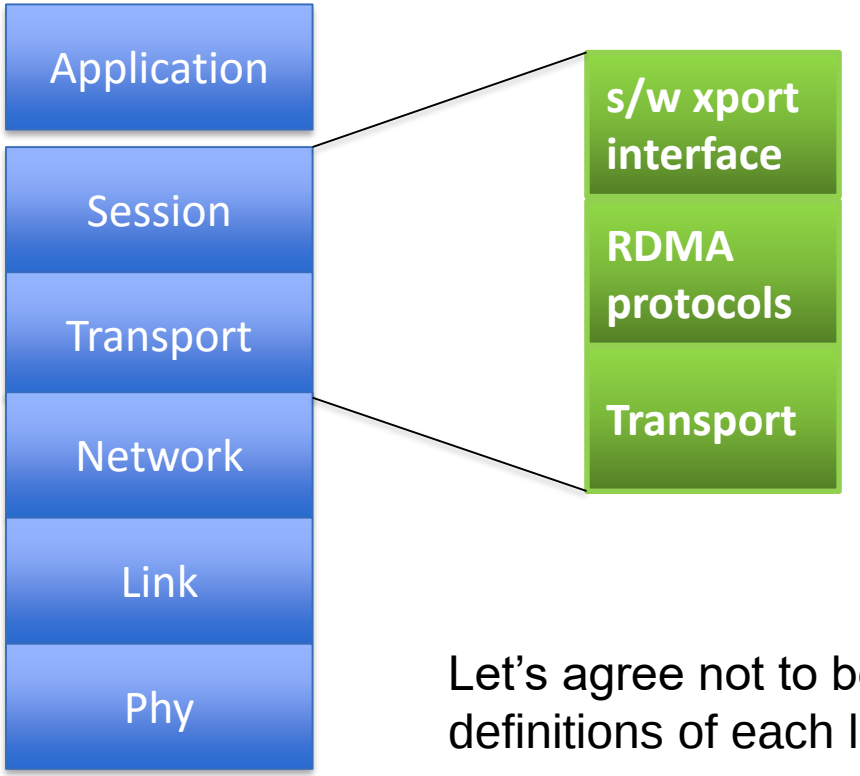
OSI-like reference model

How TCP sockets does it



- A sockets application passes a buffer's worth of bytes to the OS which transmits the data as a stream of bytes.
- The transport does not know where one message ends and the next begins.
- It's just a stream of bytes.

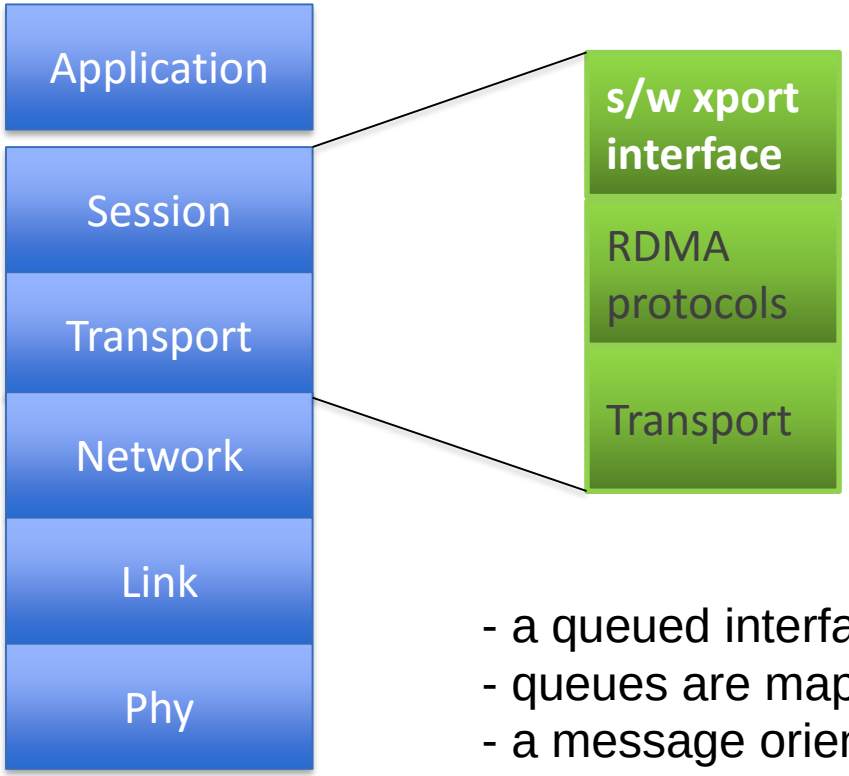
RDMA architecture



- Software Transport Interface
- RDMA Protocols
- Network Transport Service

Let's agree not to be too literal about the functional definitions of each layer.

Software transport interface



- Virtual channel adapter architecture

- a queued interface → ordered operations
- queues are mapped into application virtual space
- a message oriented architecture

A queued interface

QPs – the queue of work waiting to be executed
CQs – completion queues

-The s/w transport interface contains a series of queues
- Work queues (QPs) drive the channel interface
- CQs signal completed work
- A CQ can be associated with more than one QP

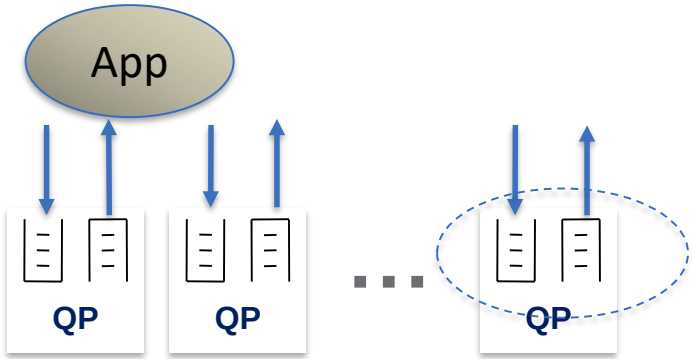
Virtual adapter architecture

Creating a work queue requires a call to the kernel

- Work queues are mapped into the app's virtual address space
- So, the application has direct access to the channel
- The OS merely 'supervises' the operation
- Unlike traditional I/O, the OS is not part of the I/O subsystem

This simple observation accounts for much of the RDMA value proposition:

Work queues, queue pairs

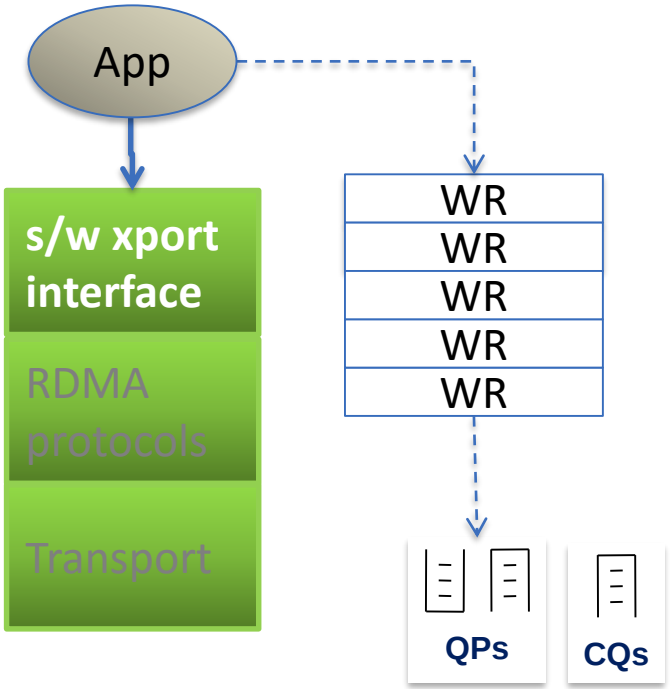


Work queues always come in pairs, hence the concept of a *Queue Pair* – a QP

QP= send queue + receive queue

- An application can create many QPs
- Each QP is associated with exactly one application

Work requests

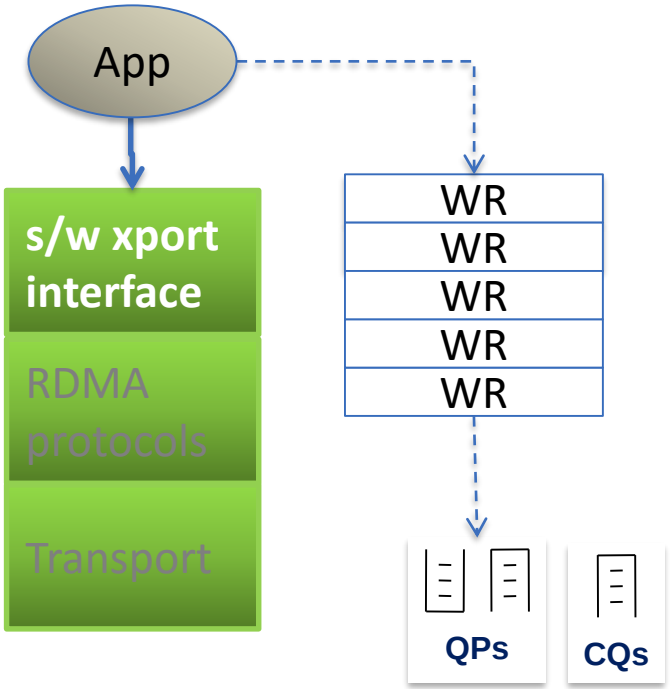


A *Work Request* (WR) is a data structure that describes a piece of work to be completed:
-a message to be sent,
-a message to be received…

An application posts a WR to a queue.

You will learn a great deal about these data structures in the next part of the course.

Architecturally…

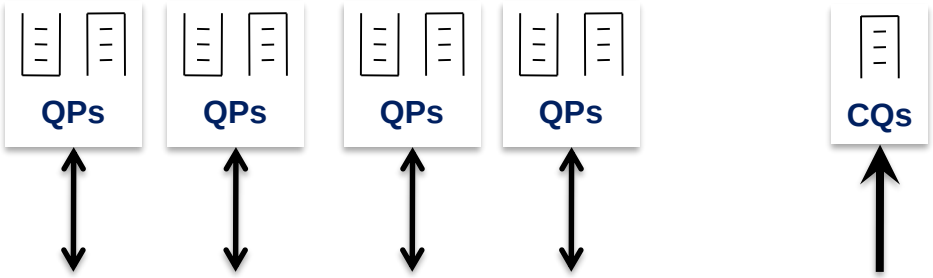


Once posted to a work queue, a WR becomes an element of that queue, called a Work Queue Element (WQE – pronounced wookie)

Similarly, the elements on a completion queue are called CQEs – pronounced cookie

For our purposes in this section of the course, it is convenient to distinguish between a WR and a WQE. However, the verbs API deals strictly in Work Requests…you won't find a reference to WQEs in the code. (But you will find them in the architecture, if you go looking).

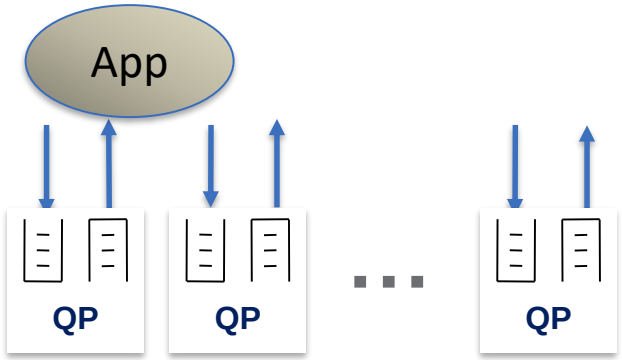
An asynchronous interface



- Once posted to the send or receive queue, the application can go off to do other work

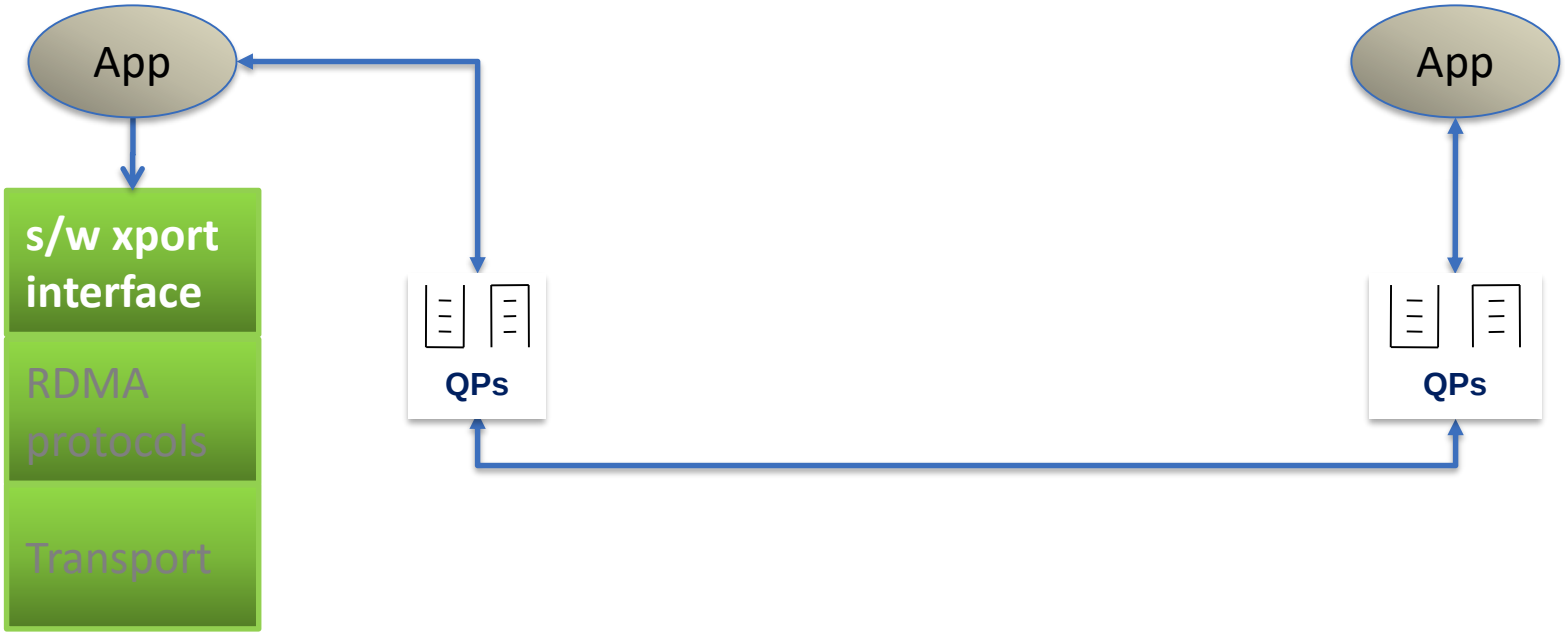
- The channel interface notifies the application when a WR is complete by posting a response to the Completion Queue (CQ)
- The application controls which QPs are associated with which CQs – it is not a 1:1 mapping.

Ordering



-Ordering is guaranteed for all WRs submitted to a given send queue
-Ordering is guaranteed to all WRs submitted to a given receive queue
-There are no ordering guarantees between send and receive queues
-There are no ordering guarantees between QPs

Connecting QPs



- For *connected services*, a connection exists between a pair of QPs
- Thus a message on a send queue can only be delivered to the other end of the connection
- Conversely, no other application can send a message using this connection.

Using the S/W transport interface

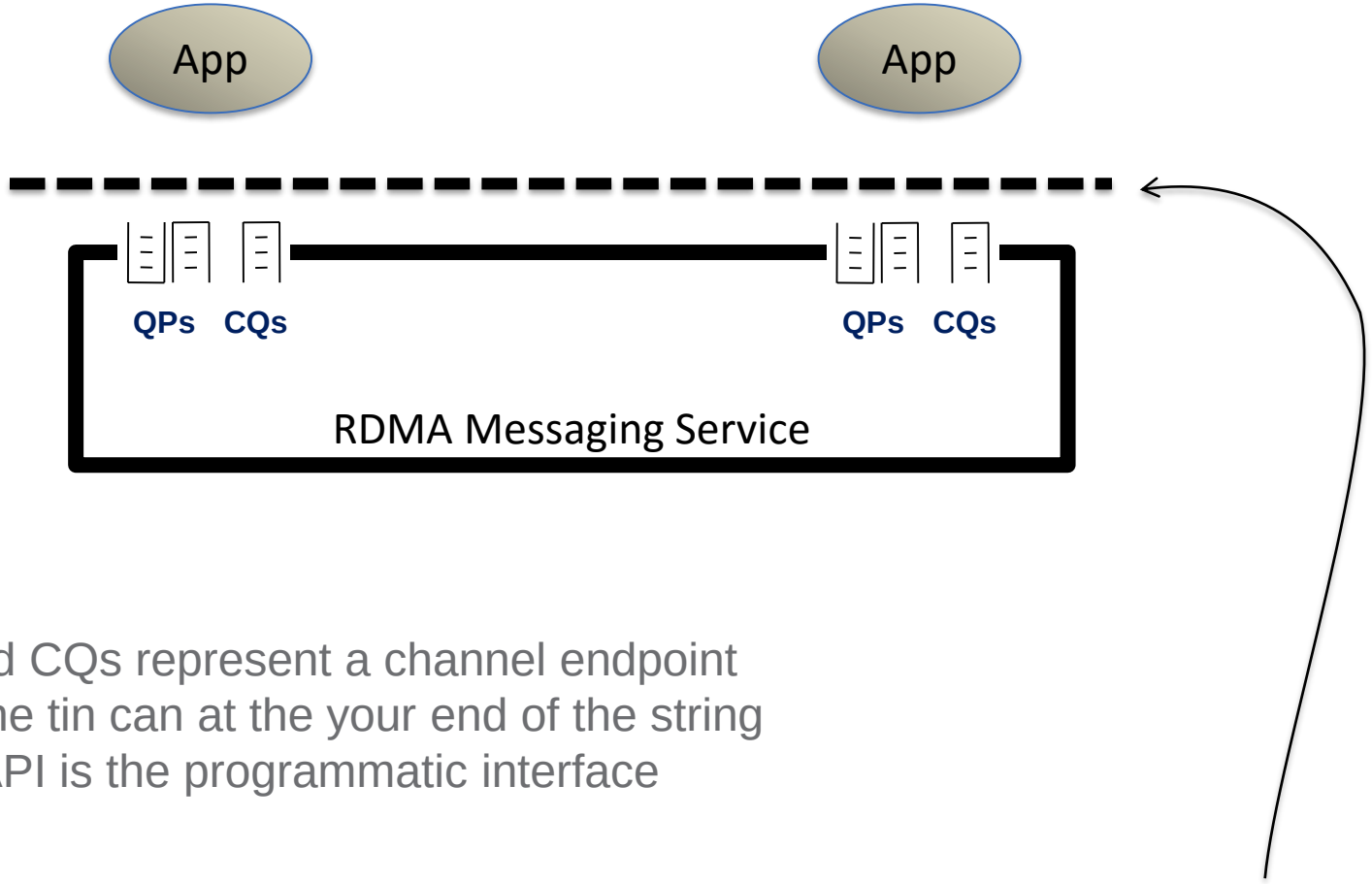
- A user creates a ***Work Request*** to either send or receive a message

- To send a message the user uses a ***Post Send Request , or Post Receive Request*** verb to post a WR on the send or receive queue.

- To prepare the hardware to receive a message, the user uses a ***Post Receive Request*** verb to post a WR on the receive queue.

You will learn a great deal more about these verbs during the next section.

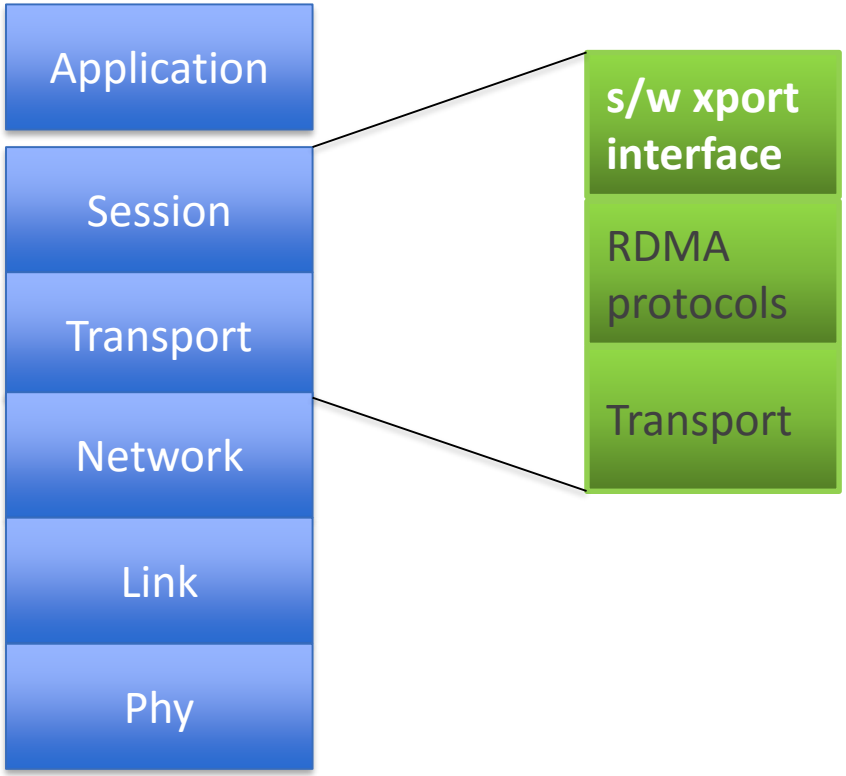
A black box model



-The QPs and CQs represent a channel endpoint
- Kinda like the tin can at the your end of the string
- The verbs API is the programmatic interface

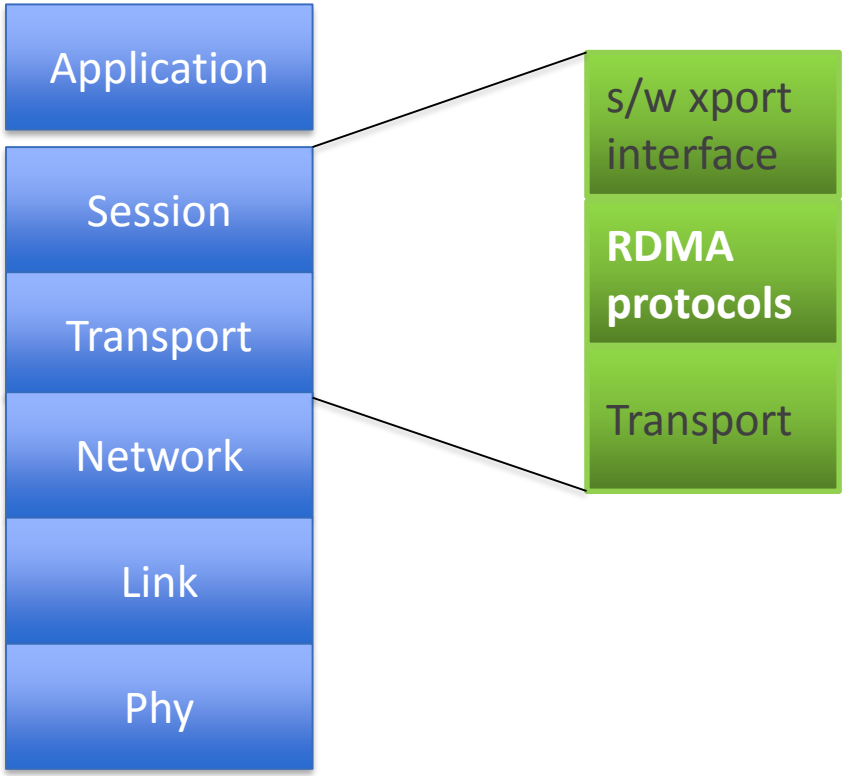
• Thus, the Verbs plus the Queues represent the crucial *channel interface*

Summarizing the s/w transport interface



- Consists of a series of work queues and completion queues
- An application uses the channel by posting work requests (WR) to the queues, and receiving completions through the CQs
- Work Requests (WRs) are data structures that succinctly describe the work to be accomplished.
- Think of WRs as commands from the application to the channel hardware.

RDMA protocols

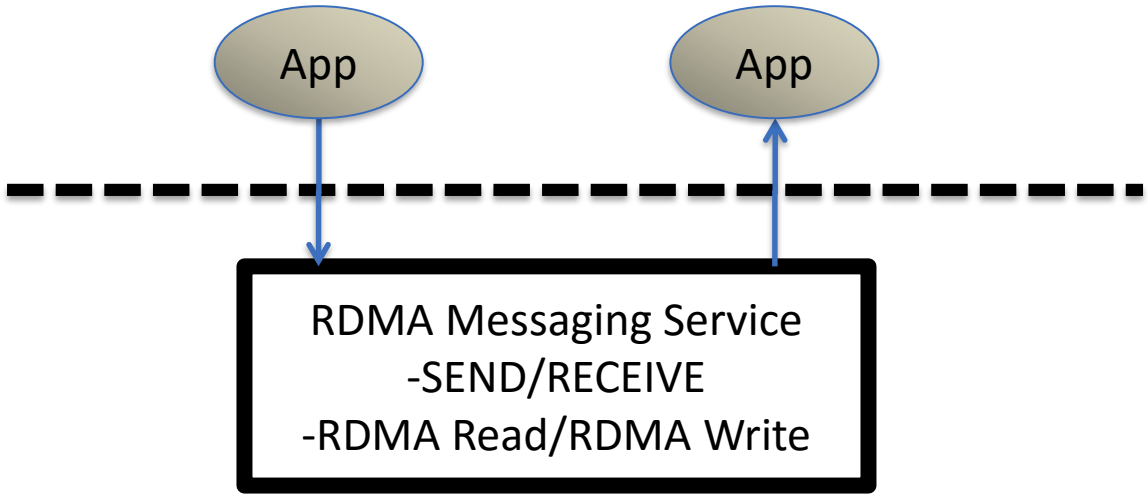


- Memory-to-memory transfer protocols

RDMA operations

A choice of message passing operations

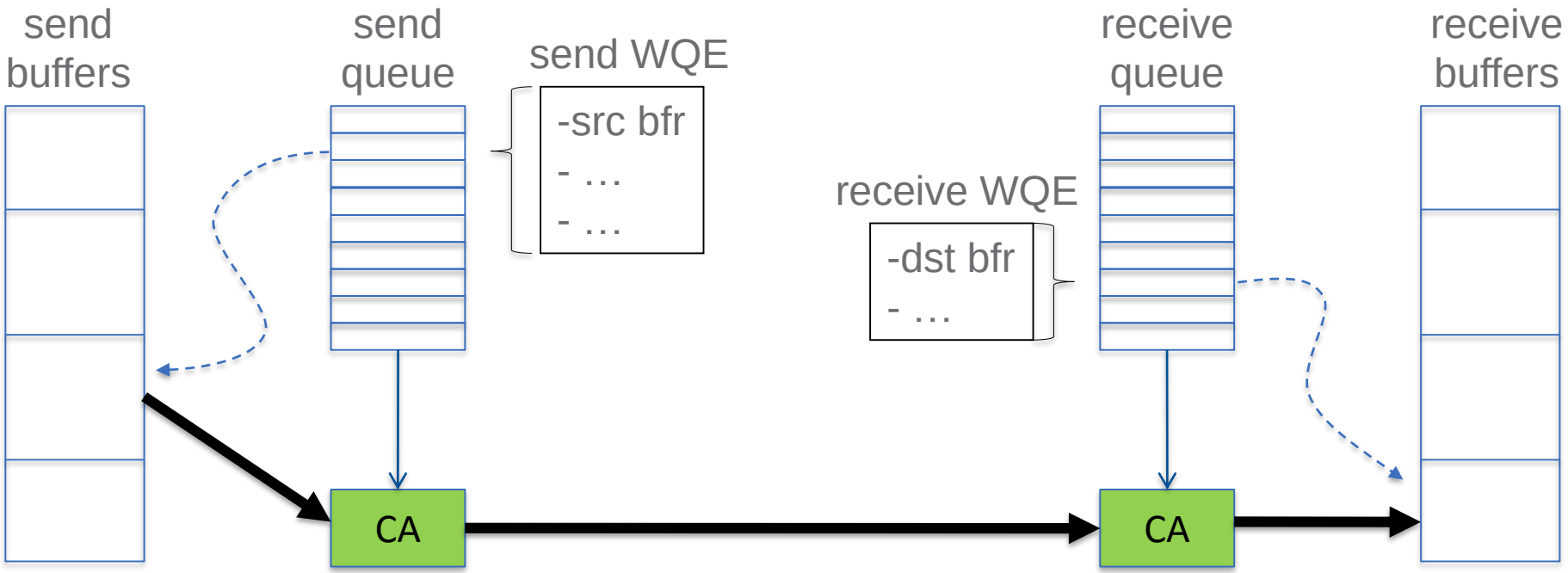
- SEND/RECEIVE – a ‘channel semantic‘
- RDMA READ/RDMA WRITE – a ‘memory semantic’



An unfortunate bit of confusion

- There are two types of RDMA operations
- SEND/RECEIVE & RDMA READ/RDMA WRITE
- Both are lumped under the rubric ‘RDMA’
- ‘RDMA READ’ or ‘RDMA WRITE’ refers to the specific operation
- ‘RDMA’ usually refers to the generic notion of memory-to-memory transfers
  - including SEND/RECEIVE and RDMA READ/WRITE
- Usually clear from the context

SEND/RECEIVE

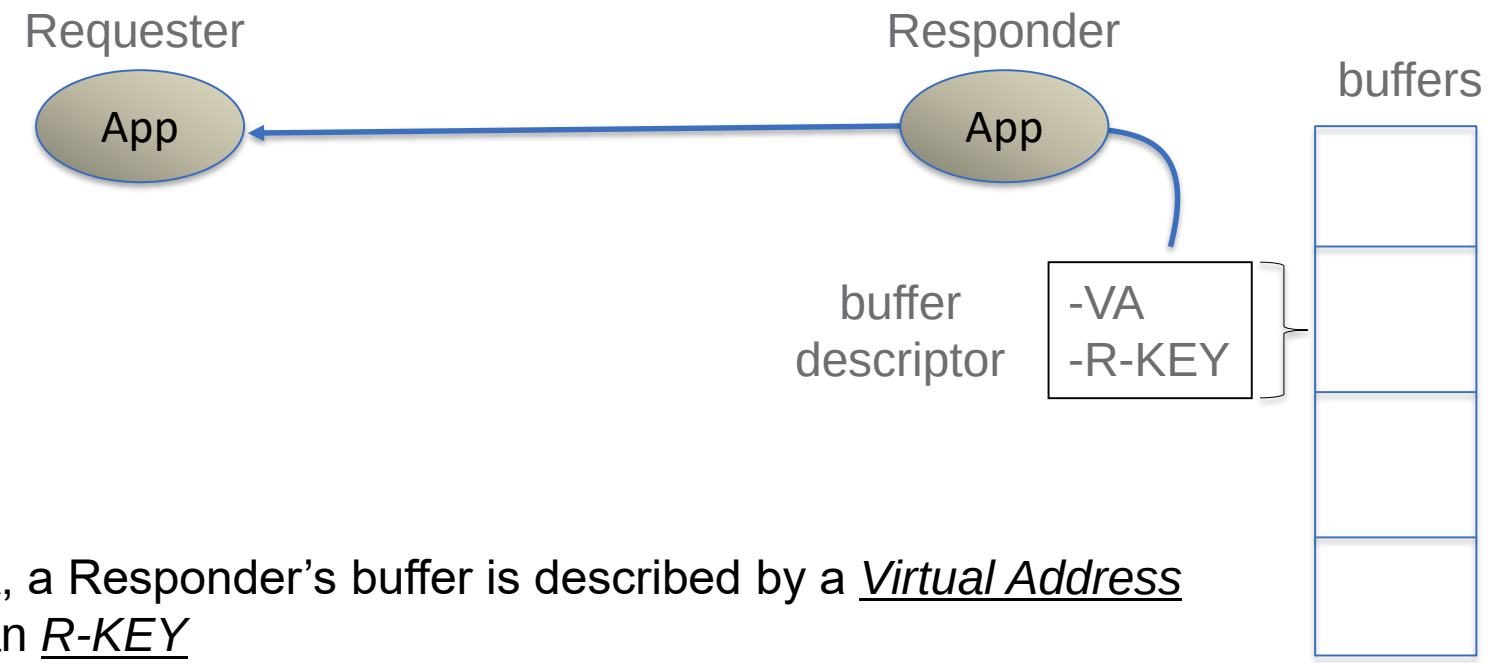


-Send WQE defines source buffer
-Receive WQE defines destination buffer
-Receive buffers must be pre-posted

SEND/RECEIVE – a channel semantic

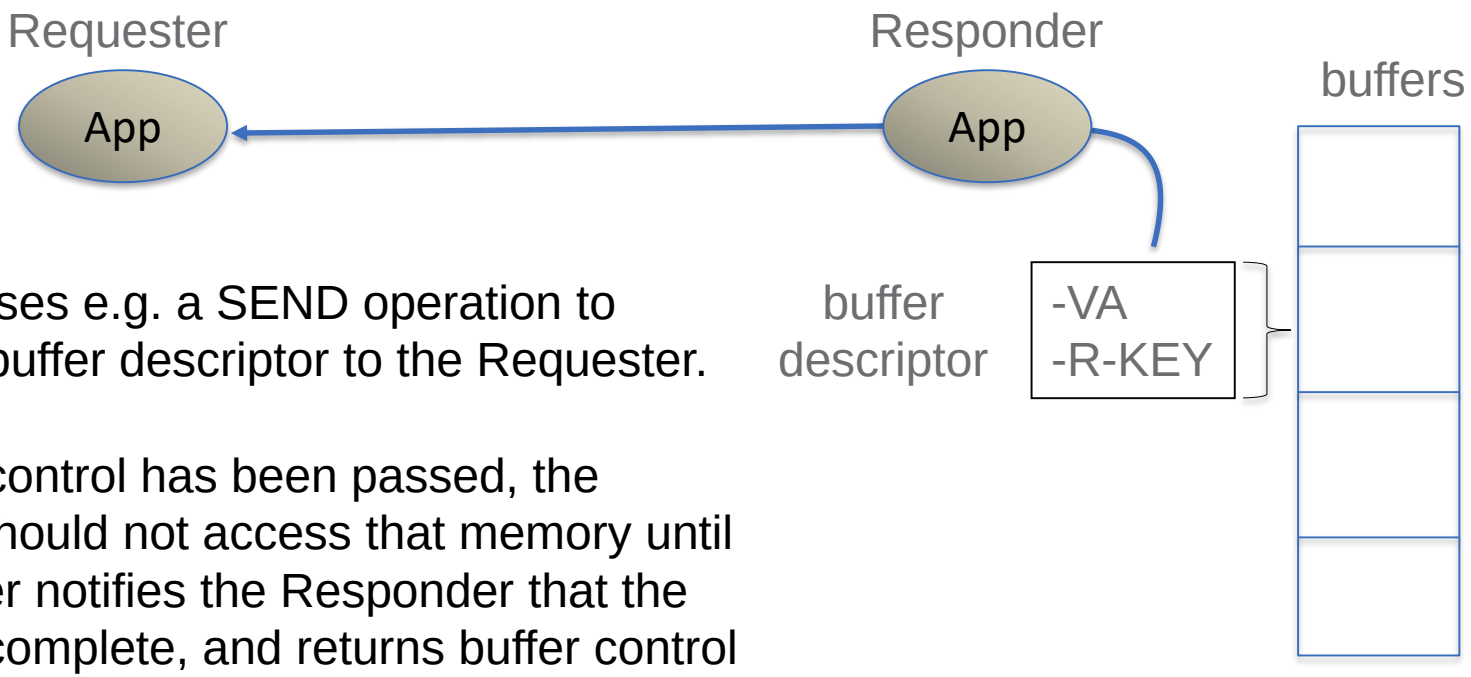
- Requester posts a SEND WR to his send queue
  - Identifies the buffer to be transmitted
  - The destination ID consists of the DLID and the destination QP#
- Responder posts a RECEIVE WR on his receive queue
  - identifies the buffer to be consumed to receive the next inbound message
- An outgoing message consumes a WQE (and a send buffer) from the send queue,
- An incoming message consumes a WQE (and a receive buffer) from the receive queue.

RDMA READ/WRITE



- For RDMA, a Responder's buffer is described by a *Virtual Address (VA)* and an *R-KEY*
- Responder passes control of the buffer to the Requester, before the op begins
- Requester controls the buffer until it passes control back to the Responder

RDMA READ/WRITE



Responder uses e.g. a SEND operation to transmit the buffer descriptor to the Requester.

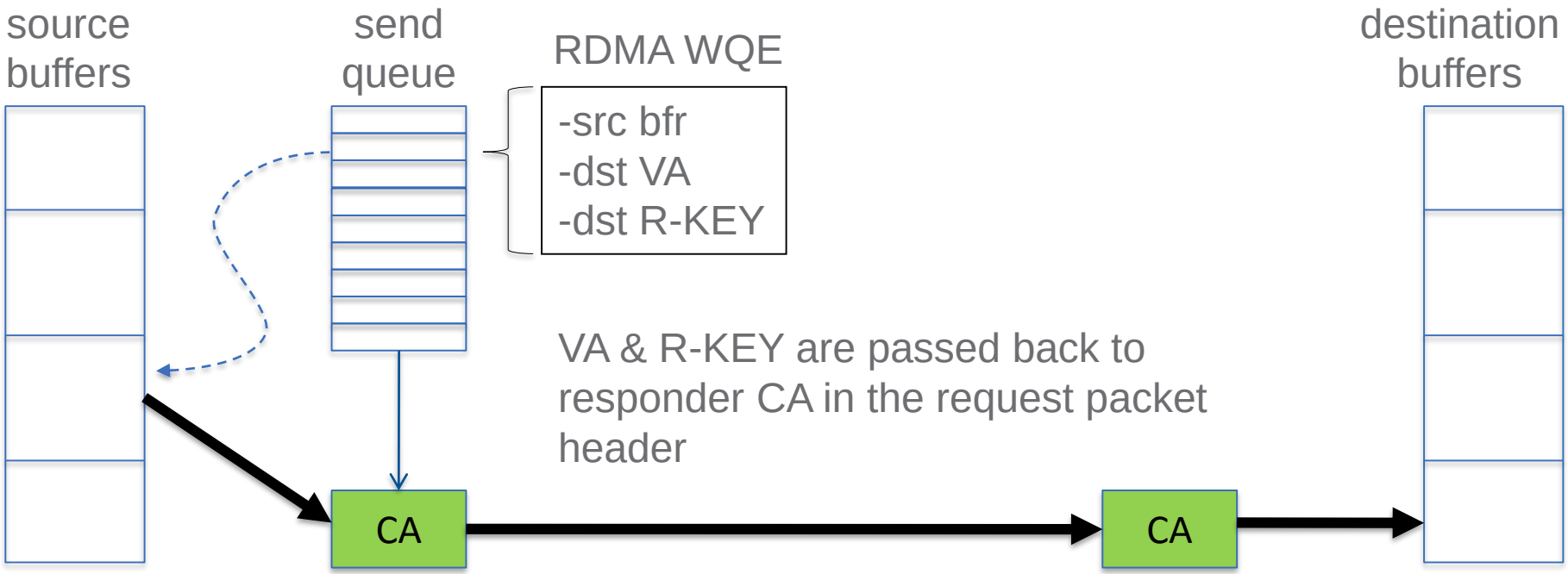
Once buffer control has been passed, the Responder should not access that memory until the Requester notifies the Responder that the operation is complete, and returns buffer control to the responder.

The means for passing control of the buffer back and forth is not specified in the RDMA architecture; it is defined by the upper layer protocol

RDMA RD/WR – a memory semantic

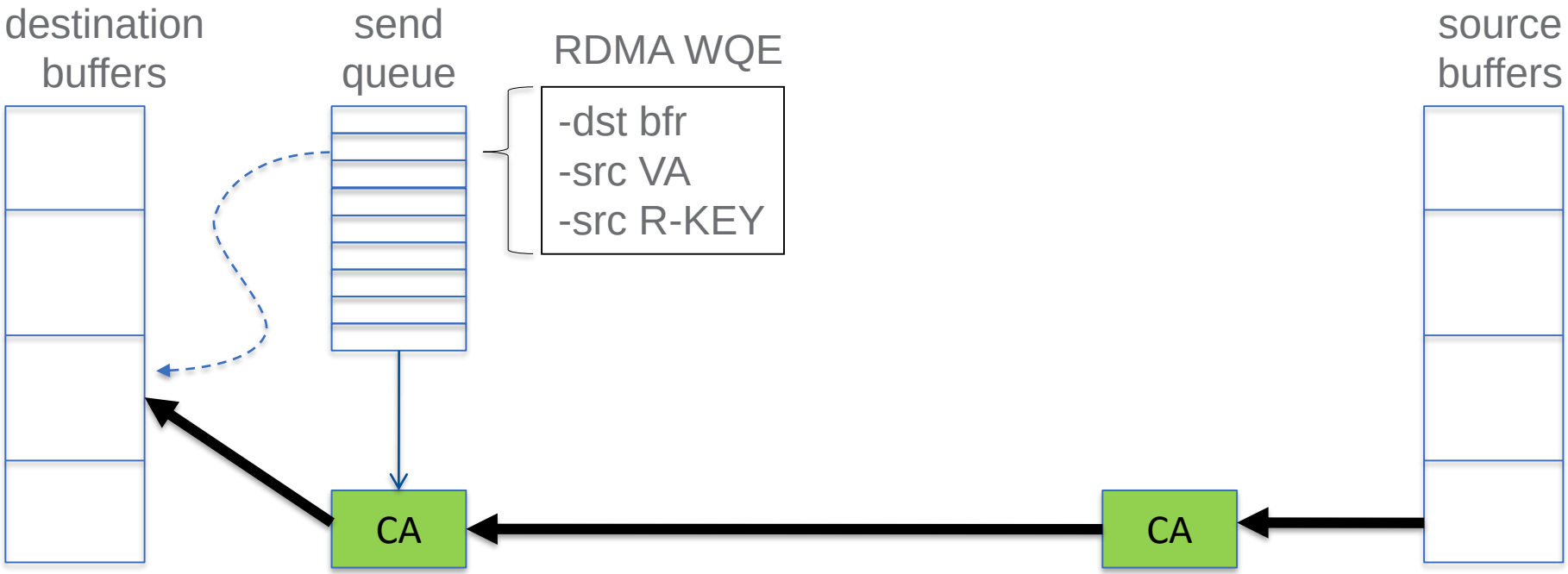
- A Responder application creates a buffer, and registers the buffer with the CA
  - The CA returns a buffer descriptor – effectively a name for the buffer
- The buffer descriptor is passed to a Requester by a Responder
  - And with it, control of the buffer is passed
- Requester can now freely do RDMA read and write operations to that buffer
- Responder application is oblivious, until control of the buffer is returned
- It takes a little more setup to get started, but…
  - …is a very efficient way to transfer large blocks of data

RDMA WRITE



- RDMA WQE defines source buffer and destination buffer addresses
- Responder CA resolves the destination buffer VA into a physical address

RDMA READ



-RDMA READ is exactly the same, except for data transfer direction

RDMA Operations - summary

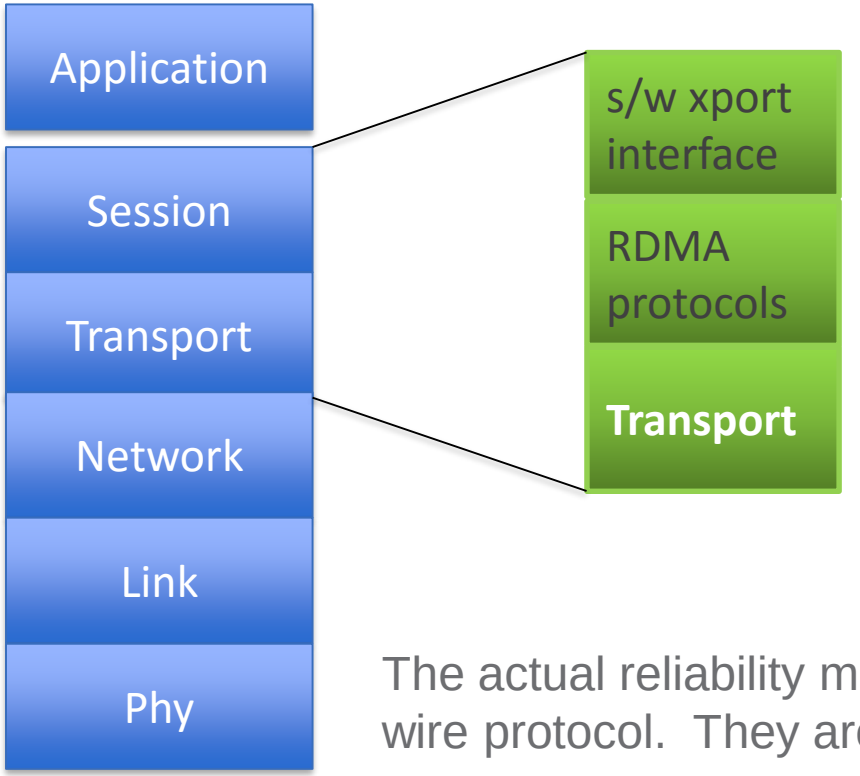
- SEND/RECEIVE is a double ended operation – it involves both the Requester and the Responder
  - Requester = SEND side, Responder = RECEIVE side
- A SEND operation consumes a WQE on both the Requester (SEND) side and on the Responder (RECEIVE) side.
- Destination buffer address is controlled by the Responder side
- RECEIVE WQE must be posted before the SEND WQE
  - failure to do so risks a race condition and a failed operation

RDMA Operations - summary

- The other application can now do memory read and memory write operations to that buffer
- The original application is oblivious, until control of the buffer is returned
- It takes a little more setup to get started, but…
  - …is a very efficient way to transfer large blocks of data

Transport service

The transport layer must provide a *Reliable, Connection-oriented* service.



- Transport service

The actual reliability mechanisms are a function of the underlying wire protocol. They are not specified by the Verbs.

Technical definition of ‘Reliable’

“Data is *delivered* to a receiver exactly once and in order, in the absence of errors”

“Exactly once” means that the receiving application is notified only once that a given message has been received.

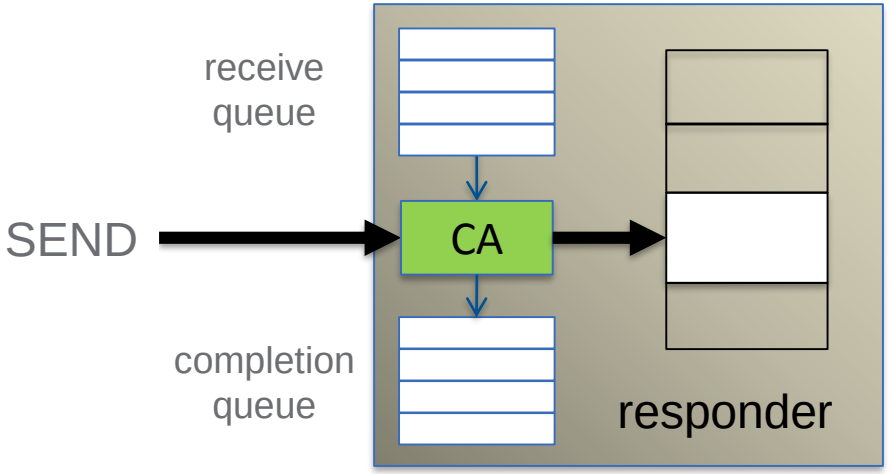
“In order” means that the transport will execute and complete each WQE in order.

“In the absence of errors” means that the transport will either recover from an error condition or report it. It won’t lie about it.

‘Delivered’

For a message-oriented transport, a message is not considered to have been delivered until all packets comprising the message have been received, and placed in the designated receive buffer in the correct order, and the receiving application has been notified (if so configured).

There are no guarantees about the state of the receive buffer memory until the CA has completely ‘delivered’ the message.
It is a bad idea to use a value in the message buffer as a flag or signal.



‘Reliable’ vs ‘Lossless’

One minor, but significant clarification.

A ‘lossless wire’ (i.e. one that does not drop packets) is NOT the same thing as a reliable transport.

Losslessness refers to the likelihood that a packet might be dropped in transit.

A reliable transport is capable of detecting and recovering from a dropped packet. Or at least reporting it. A reliable transport implements a complex protocol for this very purpose.

The InfiniBand reliability protocol is implemented by the InfiniBand Transport. Reliability for iWARP is implemented by TCP. Both are reliability protocols.

So what's a 'connection'?

A connection is established between two Queue Pairs when they have each exchanged a set of attributes, and have agreed that they are connected.

source LID	destination LID
source QP number	destination QP number

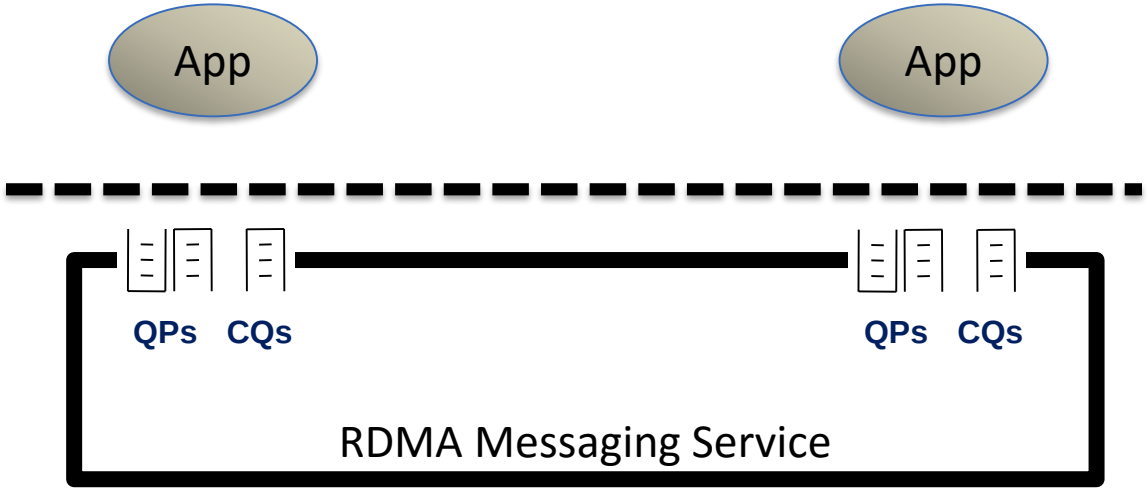
And a host of other attributes.

This exchanges allows the hardware to validate that an inbound message is, indeed destined for the QP it says it is. That's how we ensure isolation between channels.
No one else can target an operation at that QP, because it is already connected.
…

OPENFABRICS ALLIANCE

Verbs introduction

Verbs API

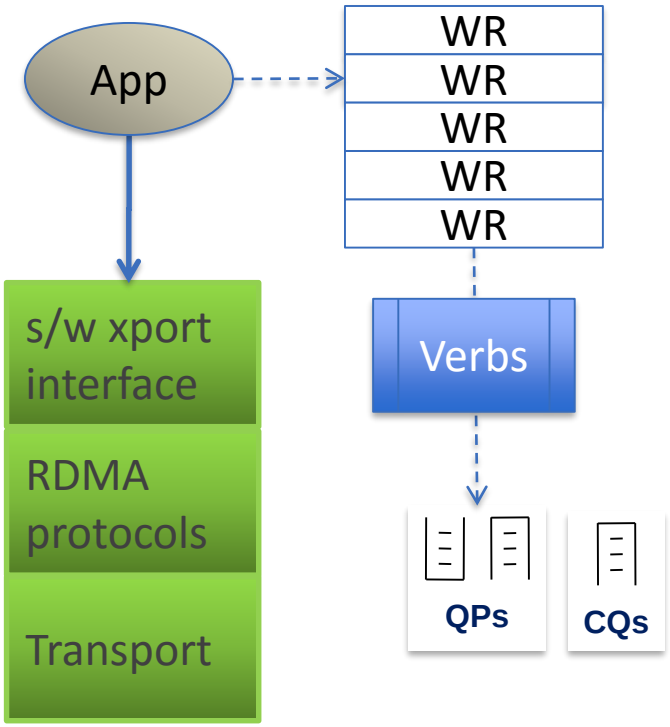


The verbs API is the programmatic interface to the RDMA Messaging Service

The messaging service is implemented as an I/O channel connecting two virtual address spaces

The QPs and CQs represent the virtual endpoints of the I/O channel

For example…



An application uses the POST SEND REQUEST verb to submit a SEND WR to the send queue.

Interesting note: POST SEND REQUEST is used to post *both* SEND and RDMA WR/RD requests to the send queue, but with a different set of attributes

A simple view of using an I/O channel

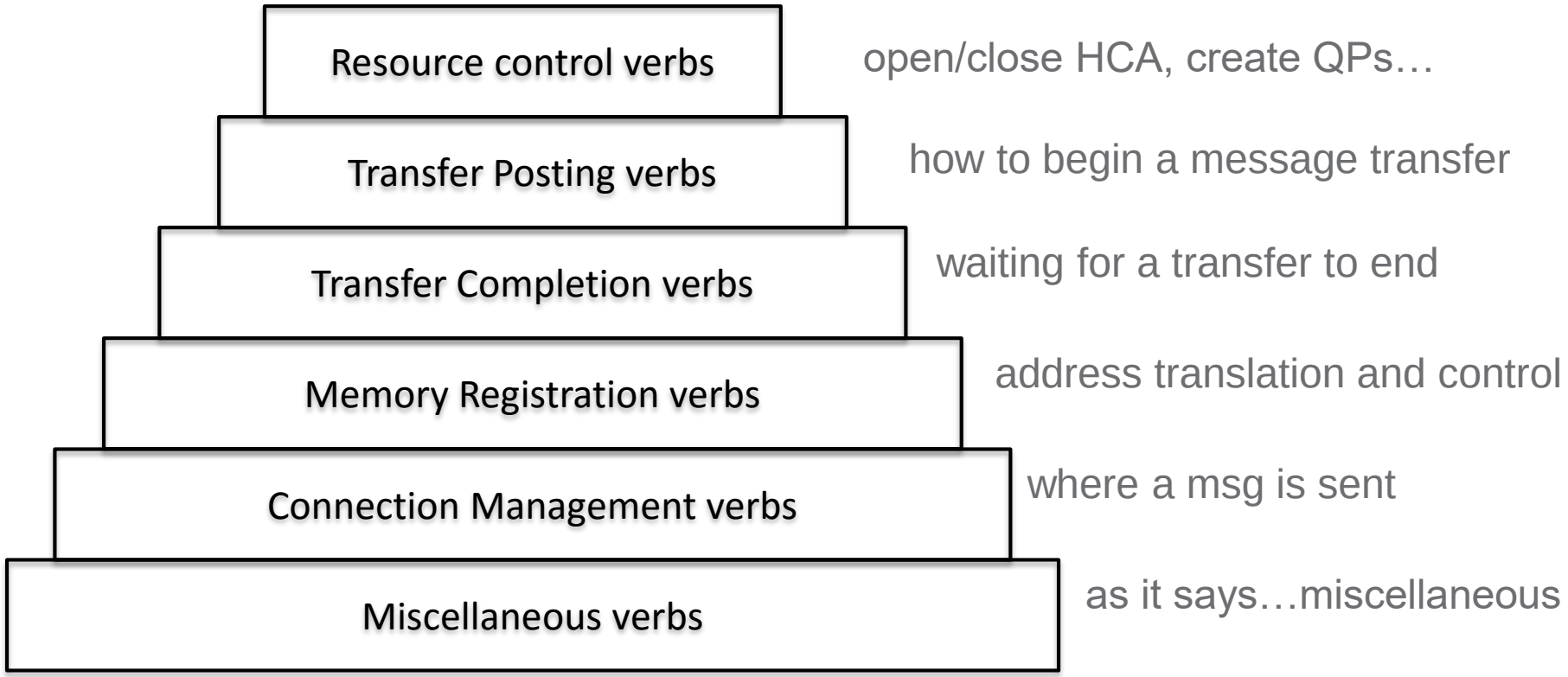
1. Open an HCA
2. Create a Protection Domain
3. Create a Queue Pair
4. Create (a) Completion Queue(s)
5. Register memory regions
6. Post work requests
7. Wait for completions

Verbs is the mechanism by which all this is done.

Part 2 of this course will take us through the verbs in some detail.

Categories of verbs

- Here is a view of the categories of verbs.
- We will go through the categories *in detail* in the next portion of this course



Verbs - a semantic description

- The concept of a ‘Verb’ was described in the InfiniBand Architecture Specification, circa 1999
- *As specified*, the Verbs are a semantic description of the required behavior of the messaging service, but with no programmatic (e.g. OS specific) details.
- Thus, verbs as *specified* are OS independent

Semantics vs an API

- Major players in the InfiniBand space recognized the value of an open source implementation of the Verbs
- The OpenFabrics Alliance was formed to create the necessary APIs
- The resulting (OS-dependent) APIs are also called ‘Verbs’
- This Verbs API is used to control an RDMA-based I/O channel

Transport independence

- Soon, others realized that there can be different implementations of an RDMA messaging service
- The RDMA Consortium spiffed up the Verbs specification while defining iWARP
- Subsequently, the InfiniBand Spec was updated to incorporate those improvements, thus preserving consistency between IB and iWARP
- A consistent set of verbs led to the notion of a common RDMA channel interface – *independent of the underlying wire-level protocol*
- This was a major breakthrough!
- Nowadays, three different 'wire protocols' are supported
  - more on these later

…and the rest is history

This bears repeating:

A consistent set of verbs led to the notion of a common RDMA transport interface

The RDMA messaging service is accessed using OFA software, regardless of the underlying transport

All three present the same interface (APIs) and semantic behavior to the consumer!

This is nearly unheard of in the known universe.

OFED currently supports InfiniBand, iWARP and RoCE.
Each looks different in terms of packets observed on the wire.

A slight conundrum

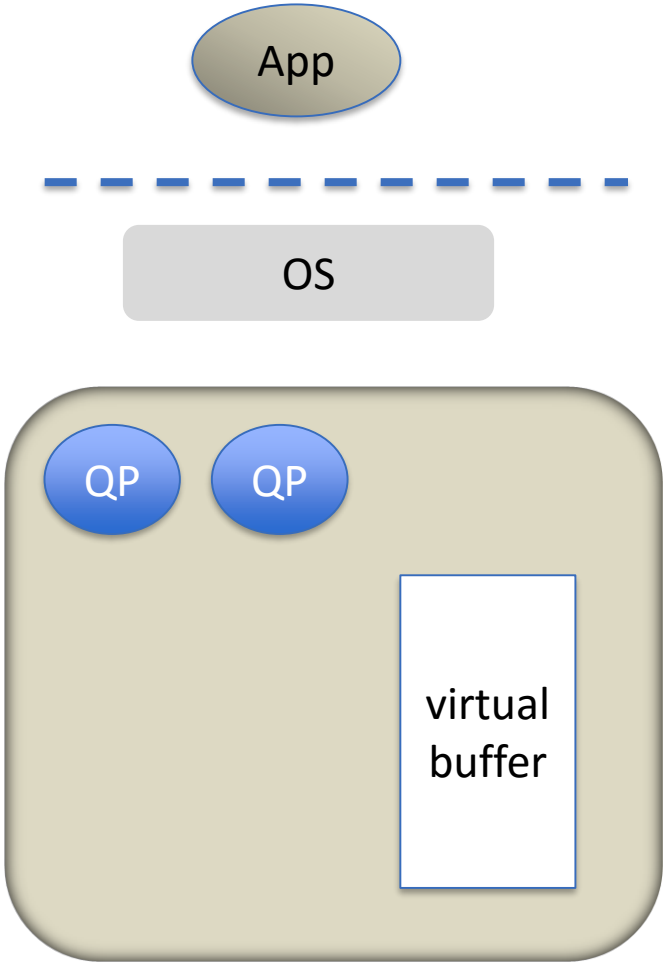
- Unfortunately…
  - Defining APIs is not in the province of a standards organization
  - Hence…
  - "**Verbs" are <u>not</u> an API** (believe it or not).

- Fortunately, The OpenFabrics Alliance came along
  - Defining APIs *is* in its province
  - Hence...
  - "**Verbs" <u>are</u> an API**

You will hear people (me) talk about Verbs, sometimes in reference to the abstract semantic description and sometimes in reference to the API.
It is rarely confusing.

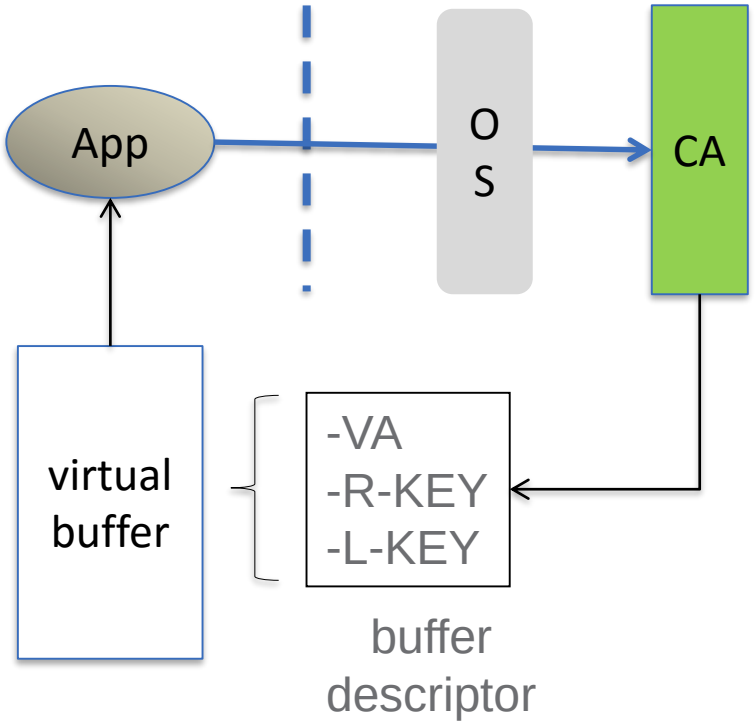
Isolation and protection mechanisms

Protection Domains



- A *Protection Domain* (PD) binds together:
  - one or more QPs
  - a region of memory, and
  - an application
- A PD is allocated. through the use of a verb
- A PD is a sort of container that is used to associate a QP with a memory region.
- The PD ensures that only the QPs that are registered to that virtual memory can access it.

Memory registration



- An app registers a region of memory with the HCA
- The HCA returns:
  - -Virtual Address (VA)
  - -R-KEY
  - -L-KEY
- The L_KEY is used when the app accesses the memory
- The R-KEY is passed to a responder during RDMA READ or WRITE operations

Summing up isolation and protection

- Protection domains control access to memory by the QPs
- App registers memory with the HCA, which returns a descriptor
  - Virtual Address, L-KEY, R-KEY.
- FMRs – (not covered in this section)
- Memory windows – not used currently, not covered in this section.

Relevant Keys:

L-KEY, R-KEY – memory binding keys. controls access to registered memory from the local user or a remote user

Q-KEY – (RD, UD only). Since datagrams are unconnected, Q-Keys are used to govern access to a remote QP. Normally, this access would be governed by comparing the SLID/DLID against QP context.

Channel isolation and protection

The WQs and the CQs are mapped into the application's virtual address space via memory registration.

The mapping is done via a **memory registration** process.

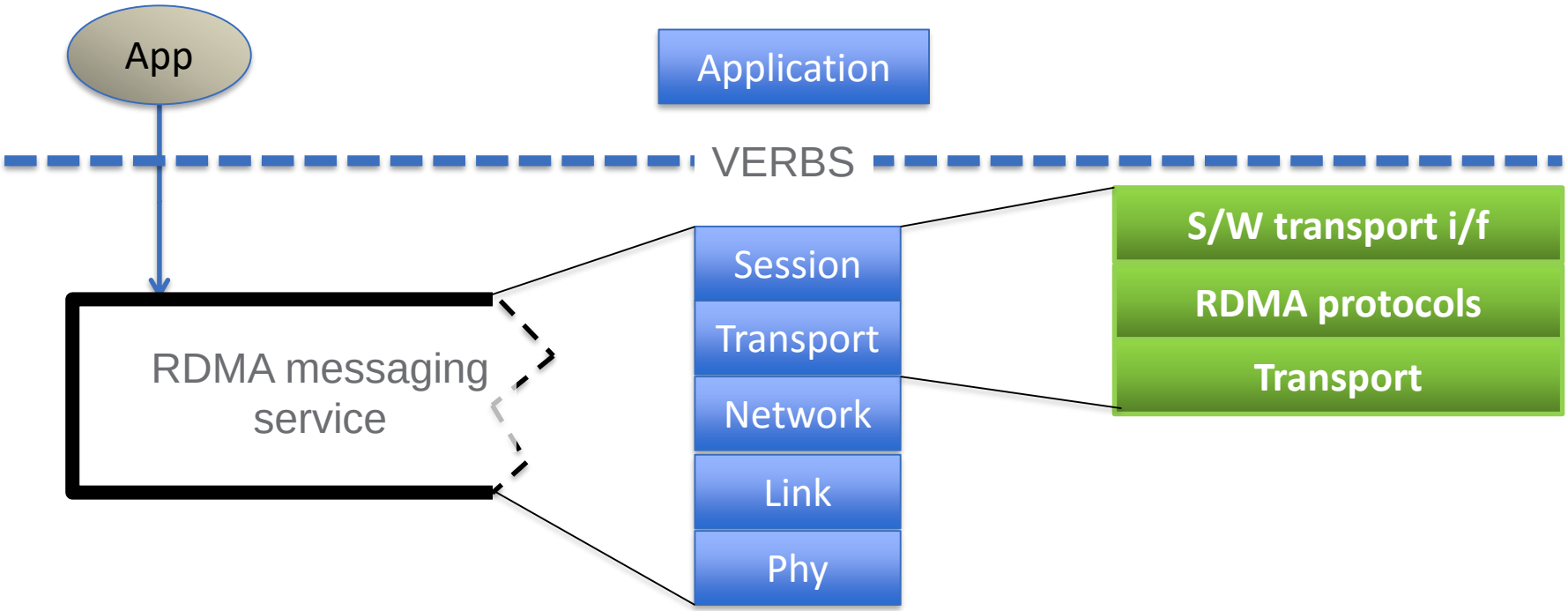
This means the application can manipulate the channel directly - no need for a context switch to a privileged entity.

Since the QPs are mapped into the application's virtual address space, they are protected and isolated from other applications.

OPENFABRICS ALLIANCE

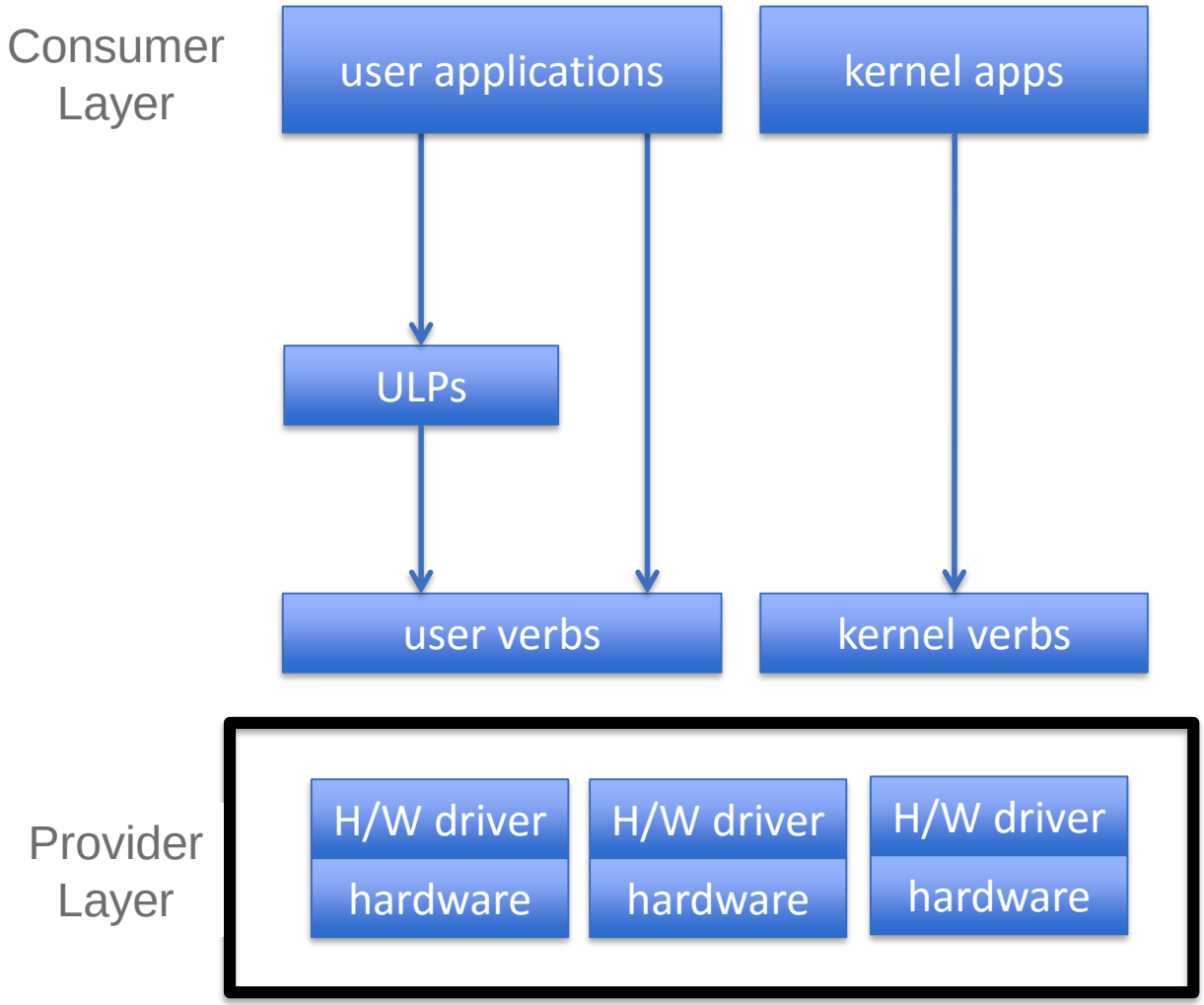
OFED stack

Quick recap

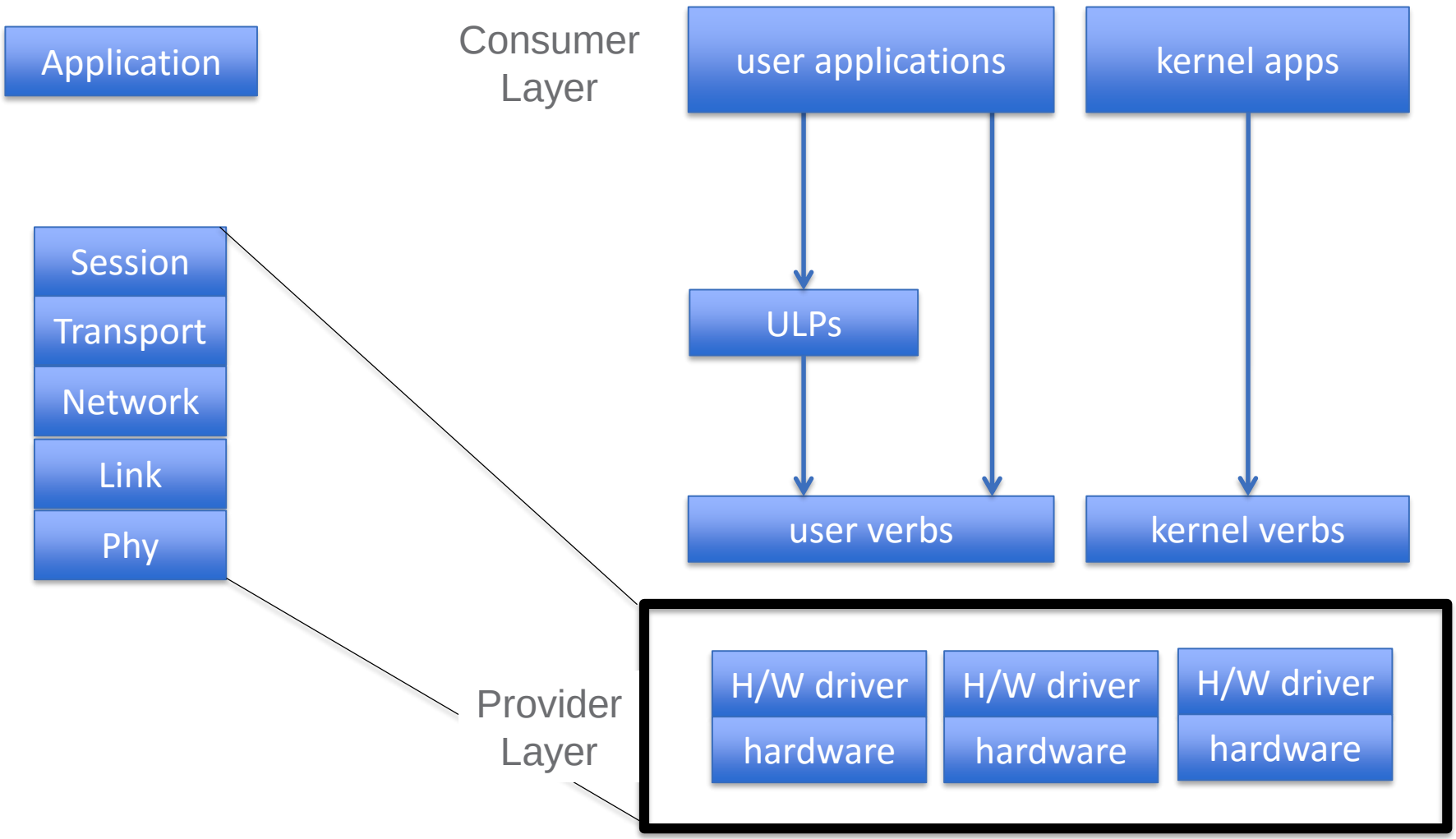


- An application accesses the RDMA messaging service by using VERBS
- The messaging service includes a complete network stack
- The principal unique elements of the stack are in the session and transport layers

A high level look at OFED



A high level look at OFED

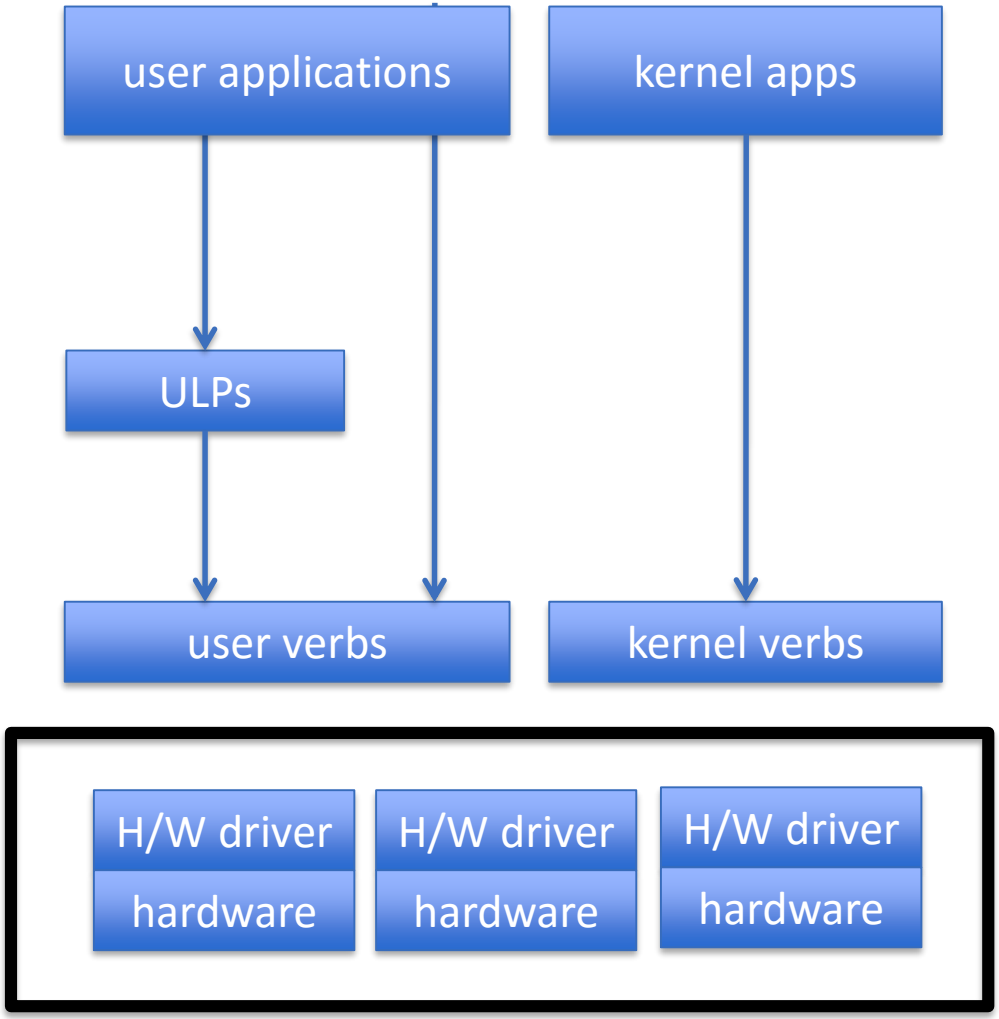


A high level look at OFED

It is not always convenient or practical to (re-) write an application to the verbs API.

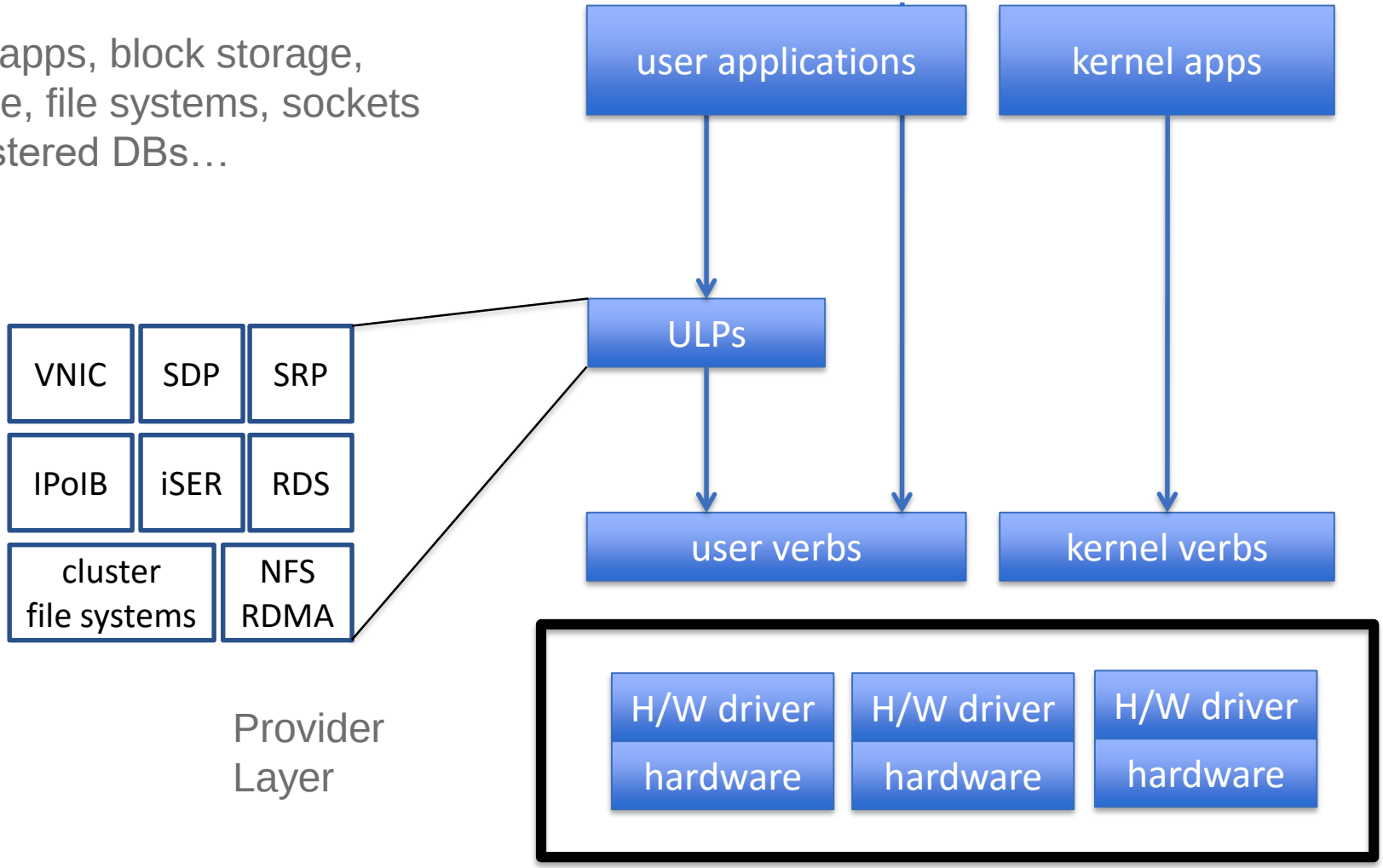
The Upper Layer Protocols (ULPs) allow a standard application to execute over an RDMA network

Provider Layer



A high level look at OFED

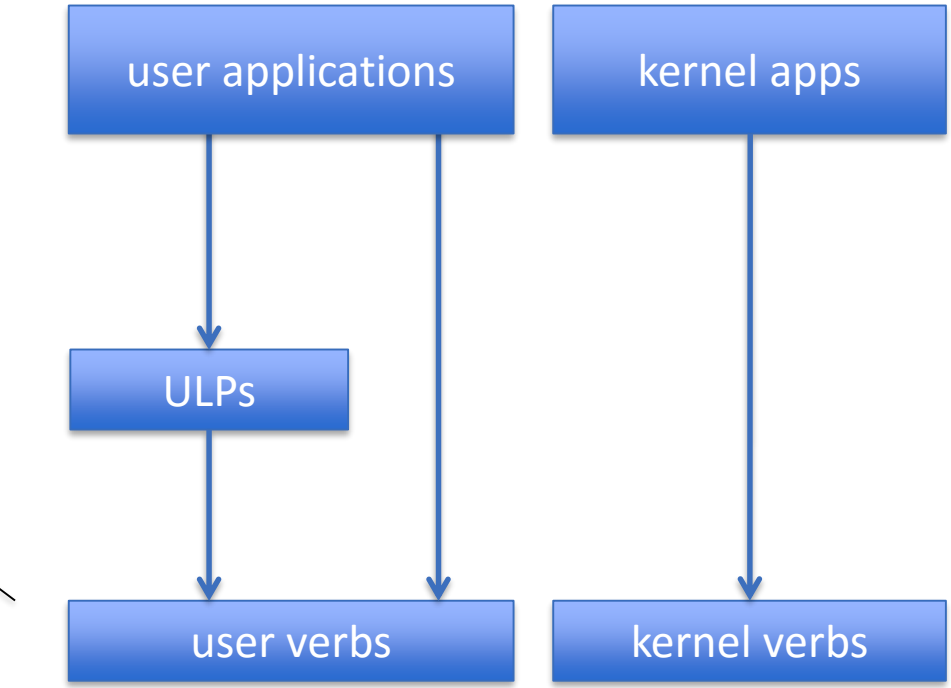
IP-based apps, block storage, file storage, file systems, sockets apps, clustered DBs…



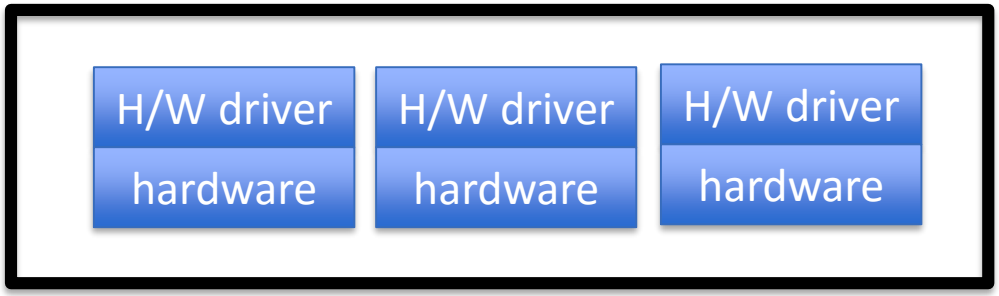
Main focus of this course…

Consumer Layer

Now that you know about the provider layer and what it does, the rest of the course is focused on the user verbs

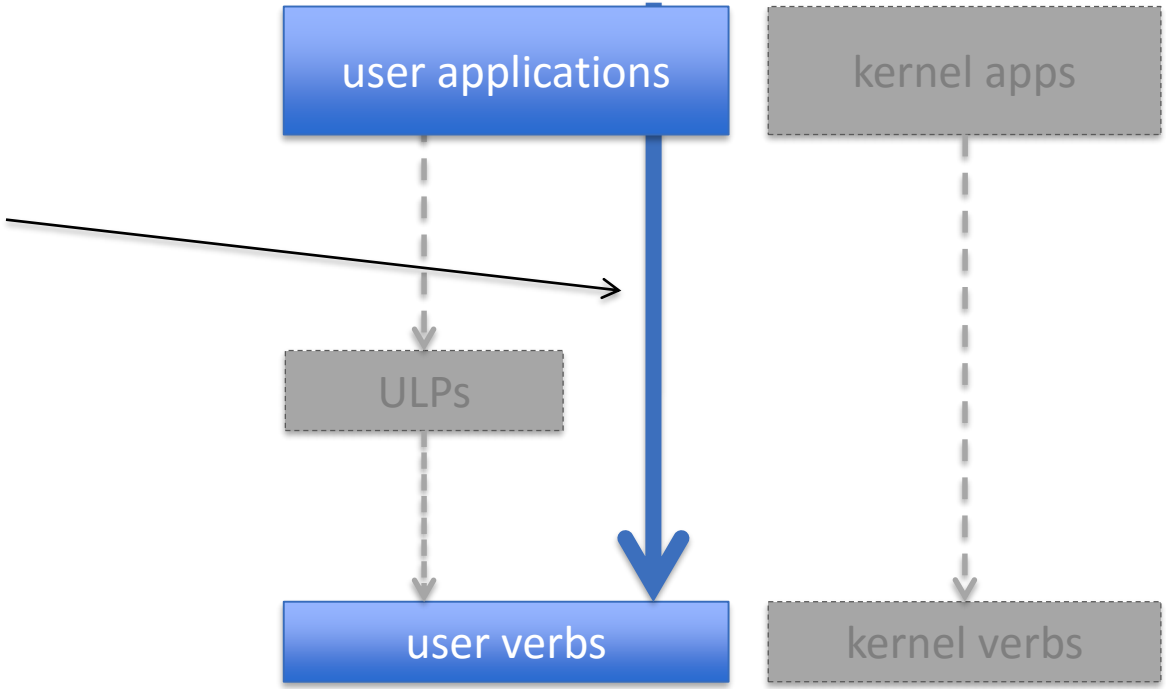


Provider Layer

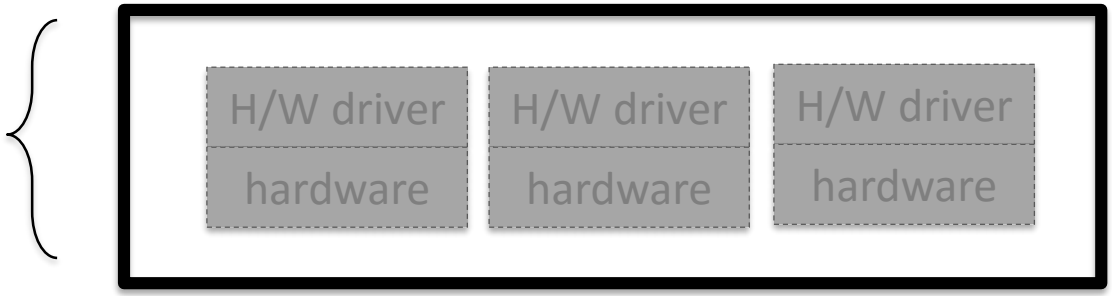


And in particular…

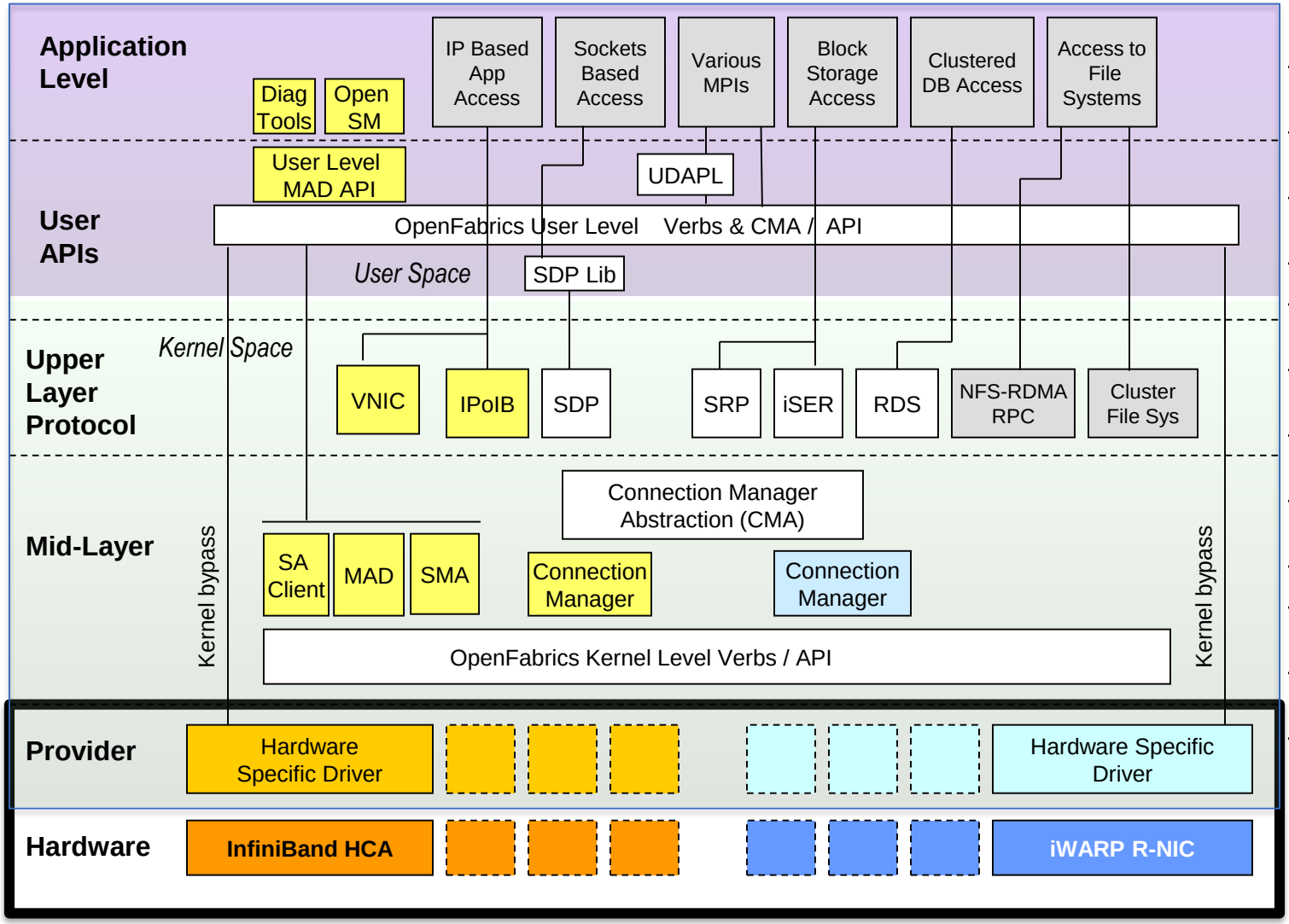
…it is about user space applications.directly to the user verbs API



you already know what the black box can do.



OFED – the details



SA	Subnet Administrator
MAD	Management Datagram
SMA	Subnet Manager Agent
PMA	Performance Manager Agent
IPoIB	IP over InfiniBand
SDP	Sockets Direct Protocol
SRP	SCSI RDMA Protocol (Initiator)
iSER	iSCSI RDMA Protocol (Initiator)
RDS	Reliable Datagram Service
VNIC	Virtual NIC
UDAPL	User Direct Access Programming Lib
HCA	Host Channel Adapter
R-NIC	RDMA NIC

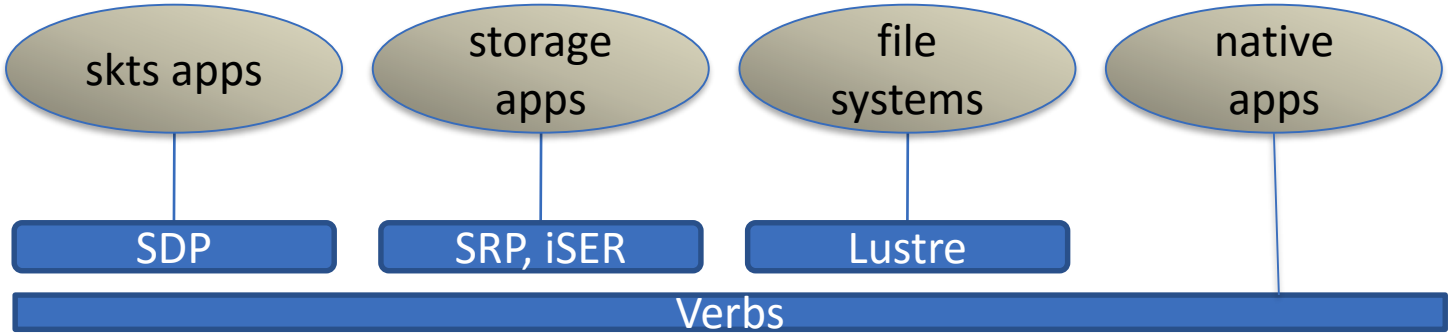
Key: Common; InfiniBand; iWARP; Apps & Access Methods for using OF Stack

OPENFABRICS ALLIANCE

Introduction to wire protocols

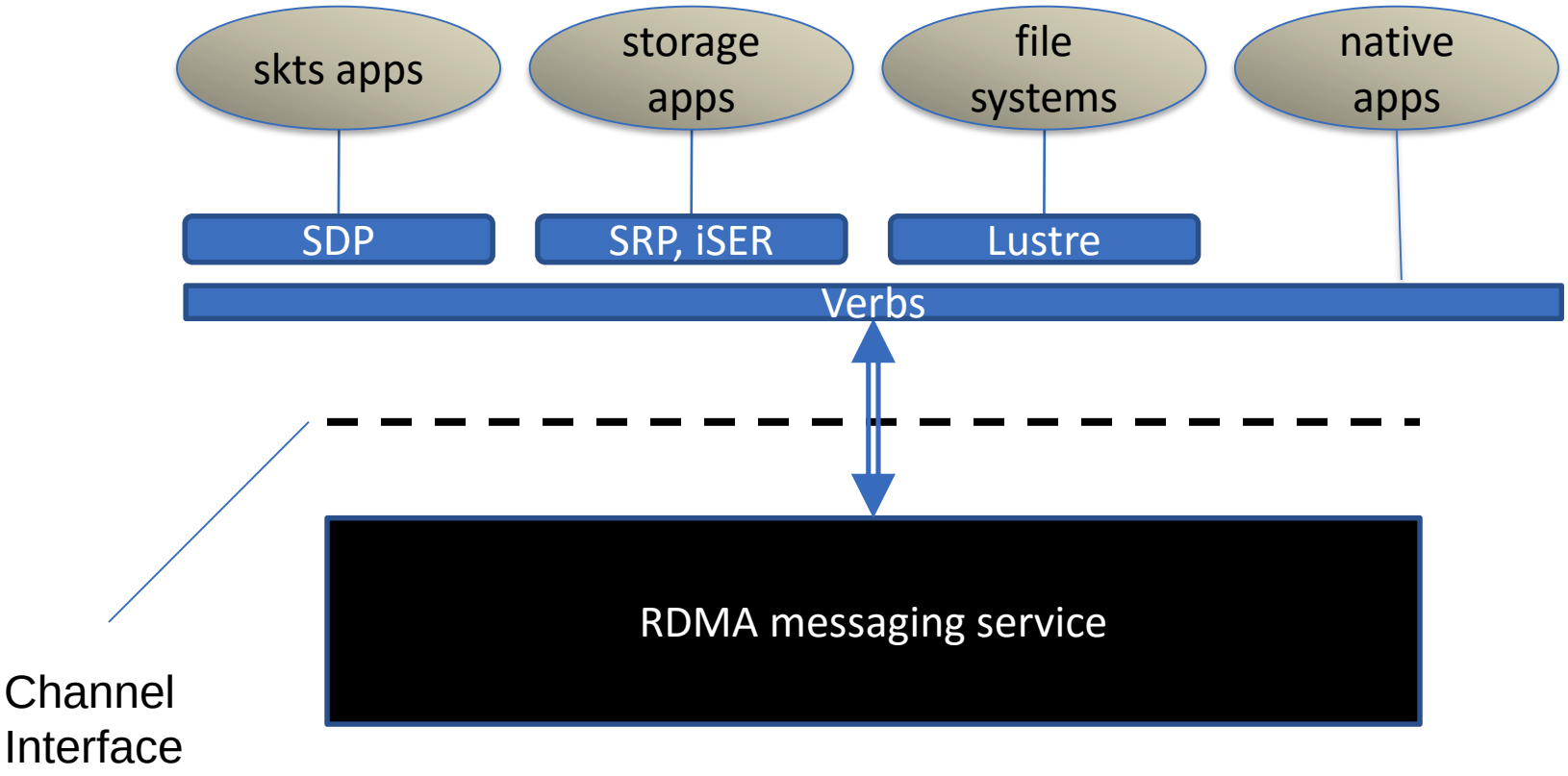
InfiniBand, iWARP, RoCE

Multiple apps



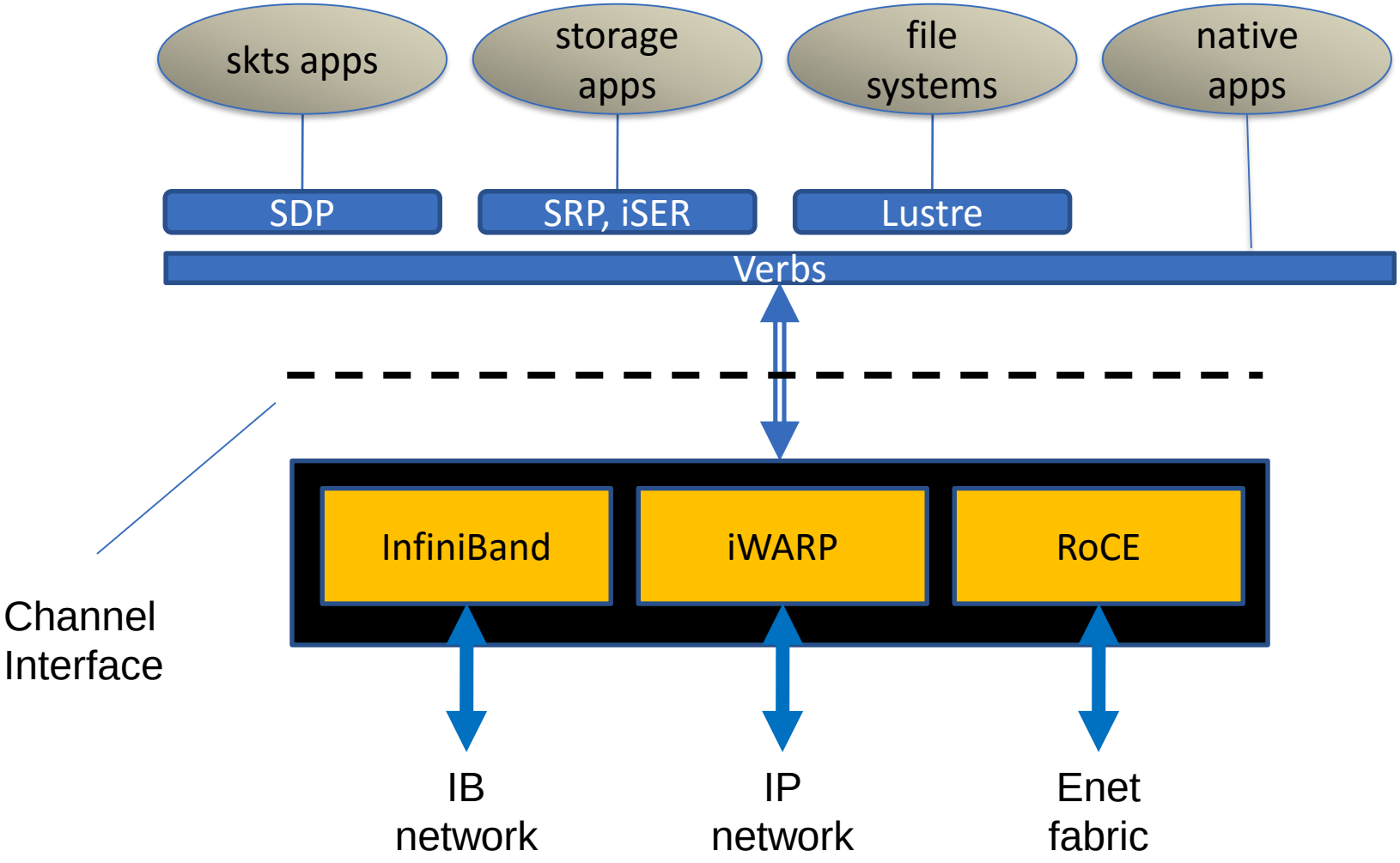
OFED is designed to support different types of applications, executing different I/O protocols

Multi apps, one channel interface

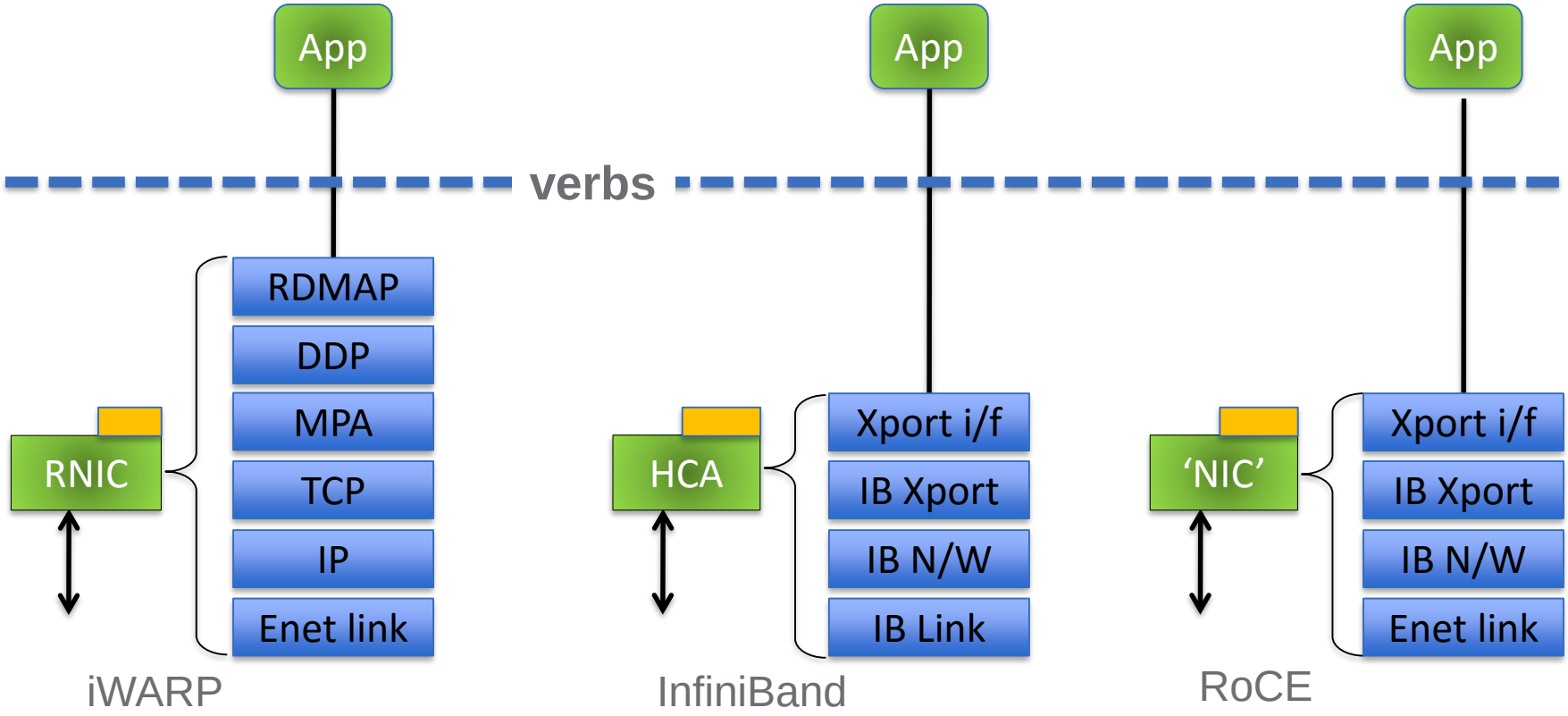


The channel interface provides a consistent programmatic interface to the message passing service

Multi apps, one CI, three wires

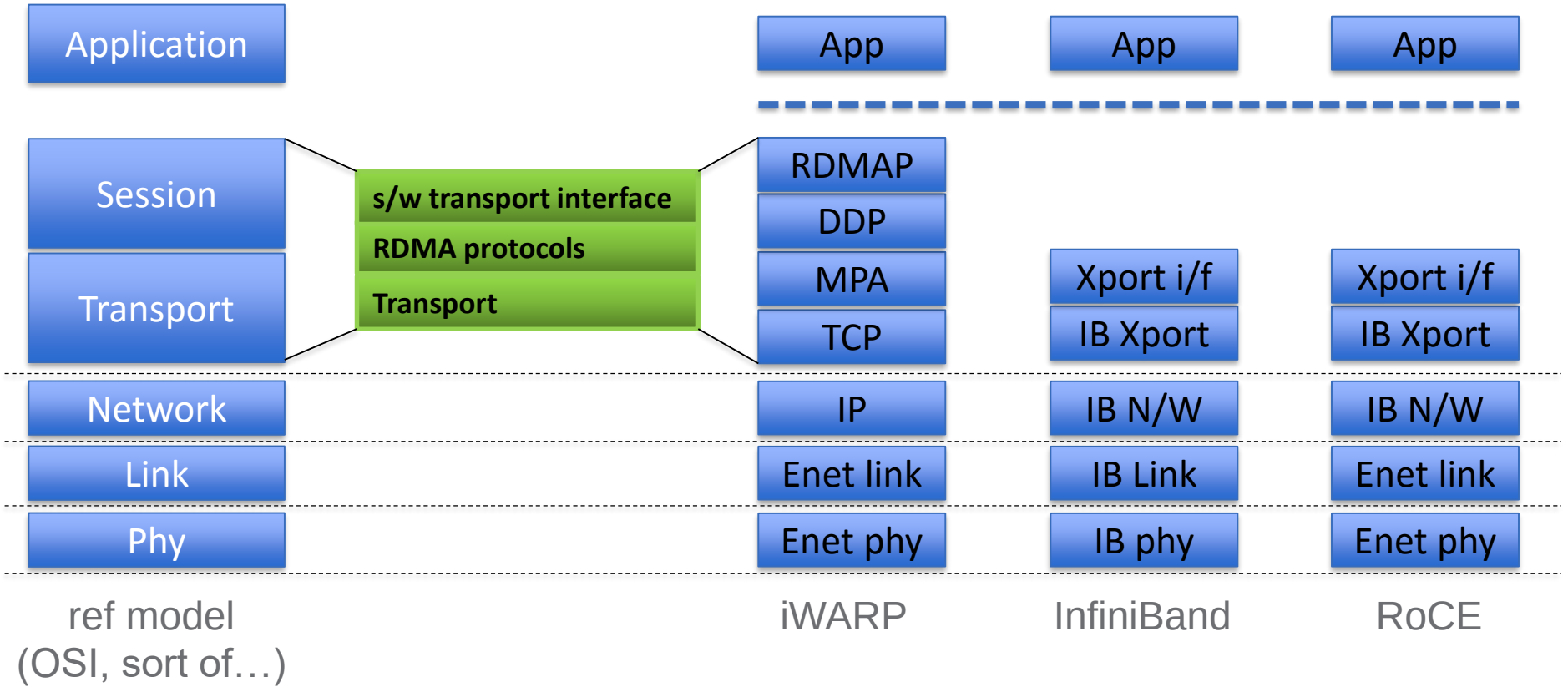


Wire level protocols



CAUTION! Don't read the hardware/software partitioning too literally!

Wire level protocols



Backup

Summing up isolation and protection

- Protection domains control access to memory by the QPs
- App registers memory with the HCA, which returns a Virtual Address, an L-KEY and an R-KEY.
- FMRs – (not covered in this section)
- Memory windows – not used currently, not covered in this section.

Relevent Keys:
P-KEY – binds certain QPs together into a partition.
L-KEY, R-KEY – memory binding keys. controls access to registered memory from the local user or a remote user
Q-KEY – (RD, UD only). Since datagrams are unconnected, Q-Keys are used to govern access to a remote QP. Normally, this access would be governed by comparing the SLID/DLID against QP context.

A list of the verbs