13th ANNUAL WORKSHOP 2017

VERBS KERNEL ABI

Liran Liss, Matan Barak

Mellanox Technologies LTD

[March 27th, 2017]

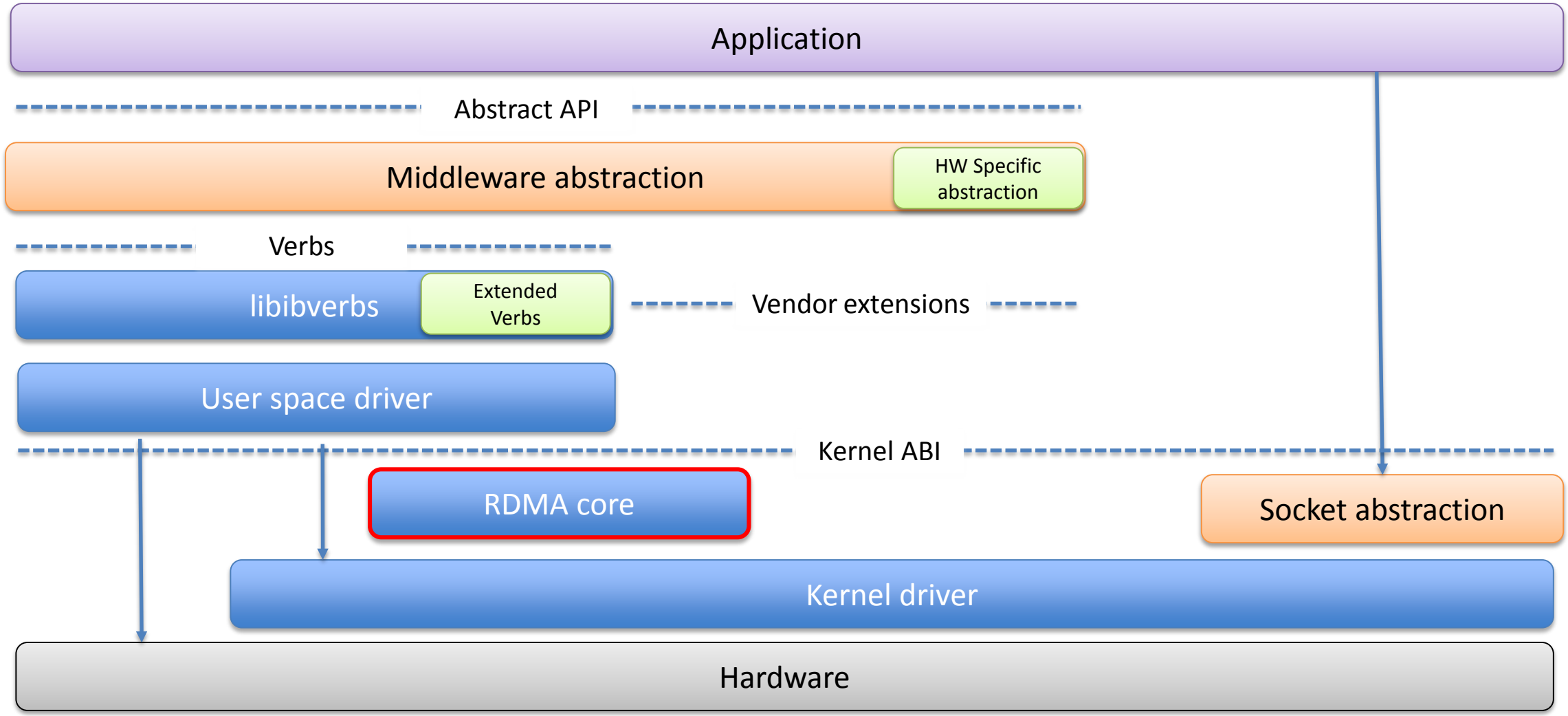
AGENDA

- **System calls and ABI**
  - The RDMA ABI challenge
    - Usually abstract HW details
    - But RDMA is all about exposing HW to user-space!
- **Current ABI**
  - Security concern
  - Feature diversity and tough extensibility
  - Code duplication
- **New proposed ABI**
  - Object oriented model
  - Using parse tree
  - device specific entities
  - More shared code
  - Maintaining backward compatibility
- **Summary**

SYSTEM CALLS AND ABI

- **Usually abstract HW details**
- **But RDMA is all about exposing HW to user-space!**

SYSTEM CALLS AND ABI



THE RDMA CORE FRAMEWORK

- **Dispatching handles**
- **Validating parameters**
- **Resource tracking**
- **Mapping user handles to kernel objects**
- **Common code**
- **Resource cleanup upon process termination**
- **Ensure backward and forward compatibility**

THE CURRENT ABI

- **Well-defined objects**
  - QPs
  - CQs
  - SRQs
  - MRs and MWs
  - AHs
  - PDs
- **Well-defined actions (methods)**
  - E.g., create, modify
- **Well-defined attributes**
- **device specific attributes in existing verbs**

SECURITY CONCERNS

- **write() syscalls are inappropriate for providing parameters**
- **Keep write semantics, and use IOCTLs for control**

CVE-2016-4565

A flaw was found in the way certain interfaces of the Linux kernel's Infiniband subsystem used write() as bi-directional ioctl() replacement, which could lead to insufficient memory security checks when being invoked using the splice() system call. A local unprivileged user on a system with either Infiniband hardware present or RDMA Userspace Connection Manager Access module explicitly loaded, could use this flaw to escalate their privileges on the system.

Source: https://access.redhat.com/security/cve/CVE-2016-4565

From: Jason Gunthorpe <jgunthorpe at obsidianresearch.com>

The drivers/infiniband stack uses write() as a replacement for bi-directional ioctl(). This is not safe. There are ways to trigger write calls that result in the return structure that is normally written to user space being shunted off to user specified kernel memory instead.

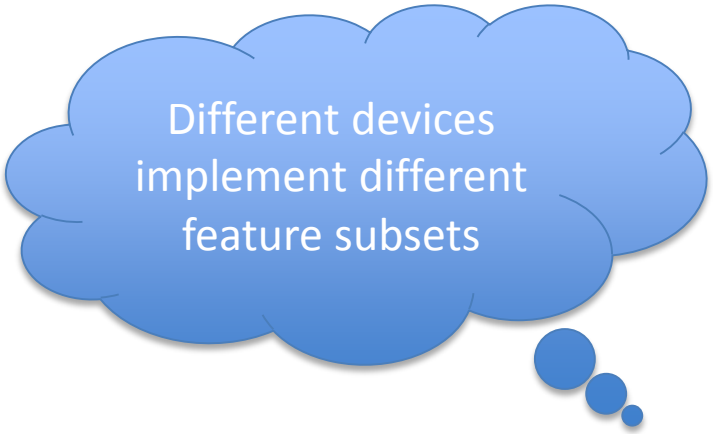
For the immediate repair, detect and deny suspicious accesses to the write API.

For long term, update the user space libraries and the kernel API to something that doesn't present the same security vulnerabilities (likely a structured ioctl() interface).

The impacted uAPI interfaces are generally only available if hardware from drivers/infiniband is installed in the system.

RDMA EXTENSIBILITY

- **Evolution of the specification**
  - New transports (e.g., XRC)
  - New memory models
  - New features
- **Evolution of Linux**
  - Demand paging for IO
  - Raw Ethernet queues
- **Evolution of HW**
  - Optimizations
  - Features



Thin abstraction layer in ABI to minimize performance cost

RDMA EXTENSIBILITY

Problem 1: Cumbersome method extension – Maintaining backward compatibility

- **Extend to new features, but still need to maintain forward and backward compatibility**
- **Hard to extend existing verbs (methods) while maintaining backward compatibility**

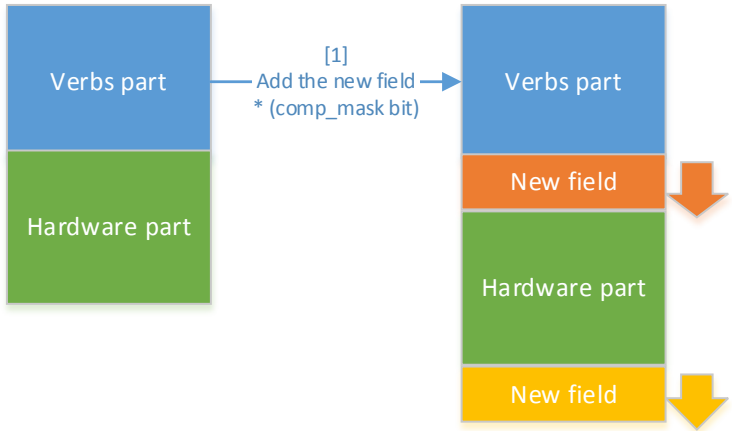
	New user-space libraries	Old user-space libraries
New kernel	✓	✓
Old kernel	✓	✓

* user-space libraries: both abstraction library (libibverbs) and the user-space device drivers

RDMA EXTENSIBILITY

Problem 1: Cumbersome method extension - – Maintaining backward compatibility

- **Different parts the standard part and device specific part**
- **Extending by growing structures**
  - When a field is added, it's always passed to the kernel → No optional attributes
  - Maintaining backward and forward compatibility requires a lot of non-trivial offset calculations and checks:




```
/* Check that the command is supported by the kernel (all bits are known) */
if (cmd.comp_mask & ~ALLOWED_COMP_MASK_BITS ||
    (uverbs_cmd_size > sizeof(cmd) &&
     memchr_inv((void *)cmd + sizeof(cmd), 0,
                uverbs_cmd_size - sizeof(cmd)))
     return -EOPNOTSUPP;

/* FOR EVERY COMMAND FIELD: check that the user gave us this new field */
if (uverbs_cmd_size >= offsetof(struct cmd, newfld) +
                       sizeof(cmd.newfld)) {
      Handle_new_fld(cmd.newfld);
}

/* FOR EVERY RESPONSE FIELD */
if (cmd.resp_size > offsetof(struct resp, resp_fld) + sizeof(resp.resp_fld)) {
    resp.response_length = offsetof(struct resp, resp_fld) + sizeof(resp.resp_fld)
    resp.resp_fld = some_value;
}

copy_to_user(cmd.resp, resp, resp.response_length);
```

1. Familiar with all comp_mask bits
2. If the given command is bigger to what is known to the kernel, the rest are zeroes.

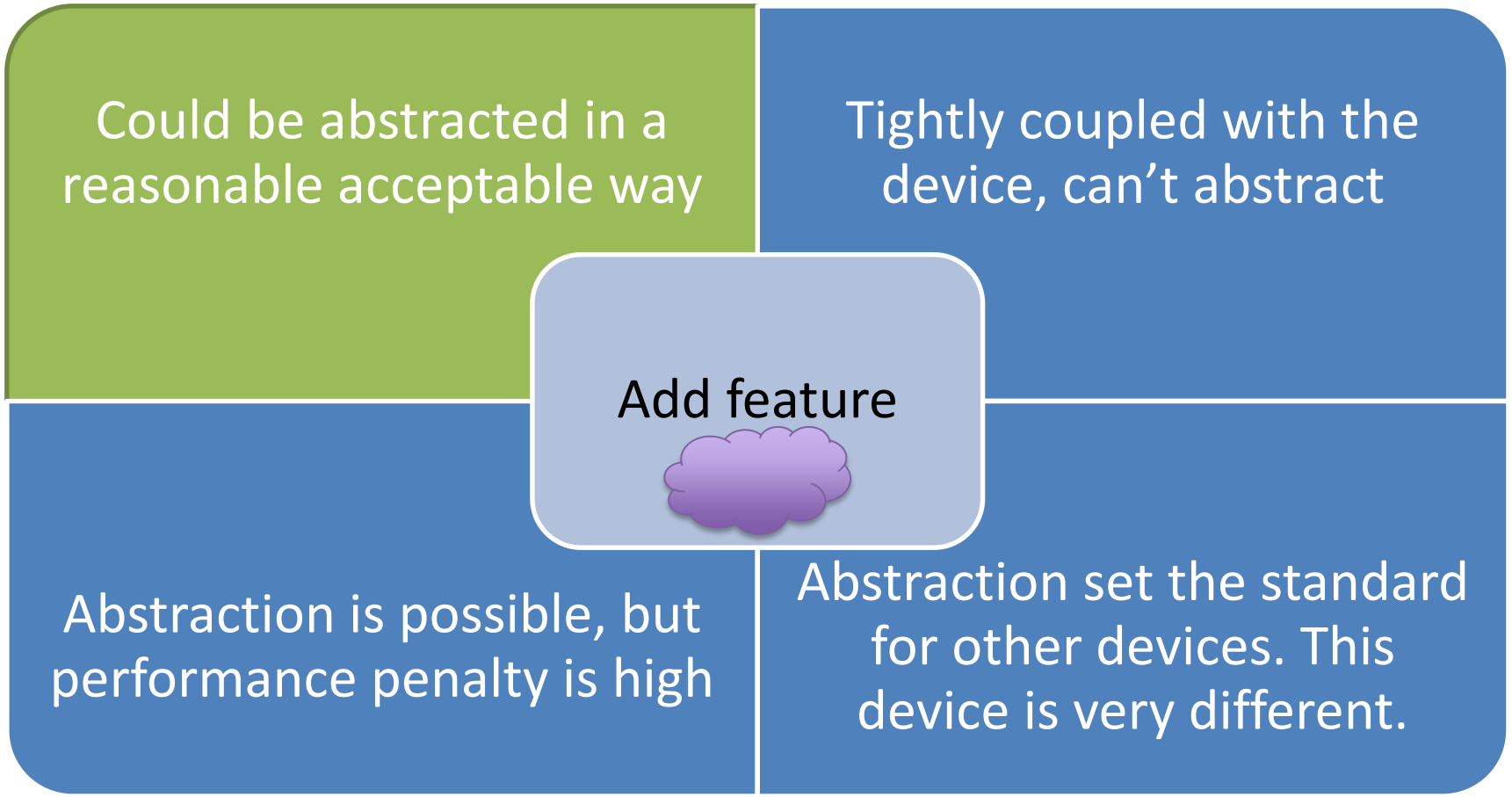
Is this command field part of the given command?

Is this response field part of the allocated user-space response?

HARD AND PRONE TO ERRORS!

RDMA EXTENSIBILITY

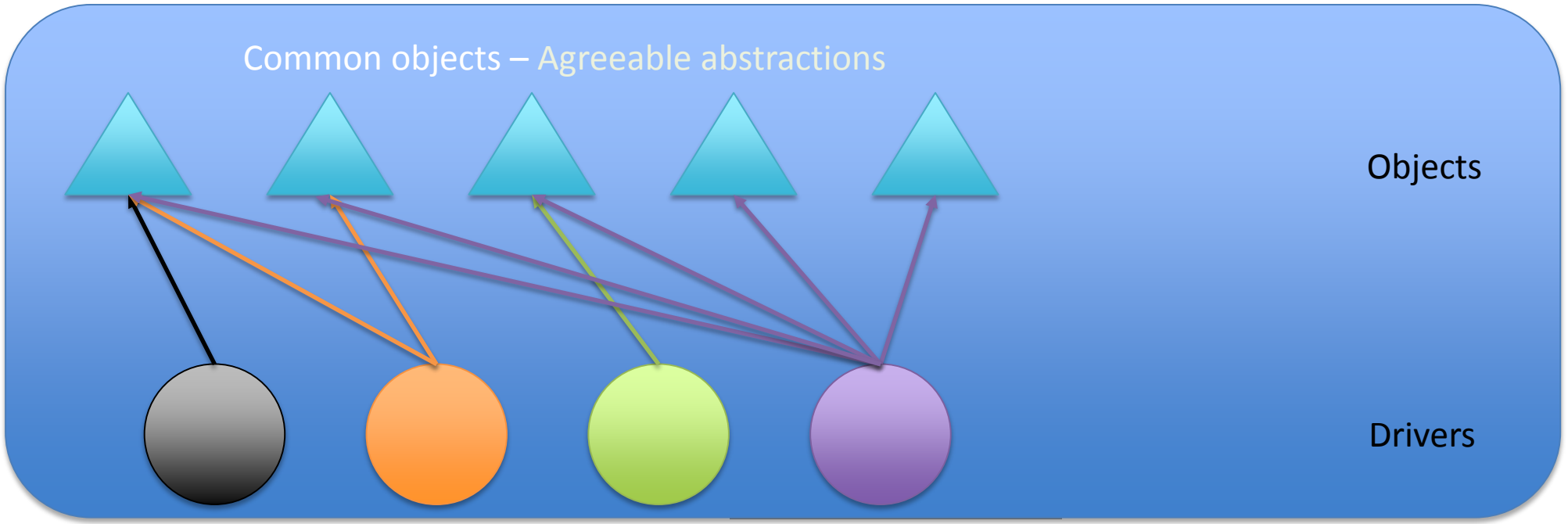
Problem 2: Feature diversity – When simple extensions are not enough



RDMA EXTENSIBILITY

Problem 2: Feature diversity – When simple extensions are not enough

- **Feature diversity**
  - Currently single flat namespace
  - Many optional APIs and device features
    - Different vendors implementing different subsets

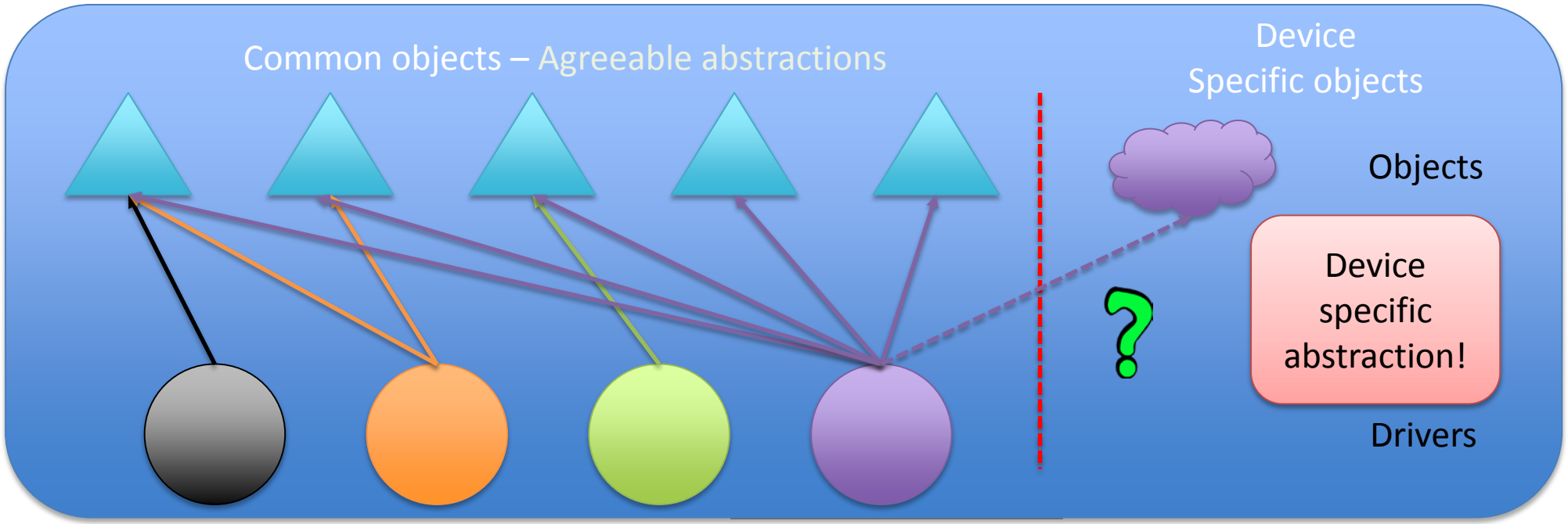


RDMA EXTENSIBILITY

Problem 2: Feature diversity – When simple extensions are not enough

- **Feature diversity**
  - Currently single flat namespace
  - Many optional APIs and device features
    - Different vendors implementing different subsets
    - Allow vendors to expose unique device capabilities
  - Device-specific optimizations aggravates the problem

No way to add a device unique objects!



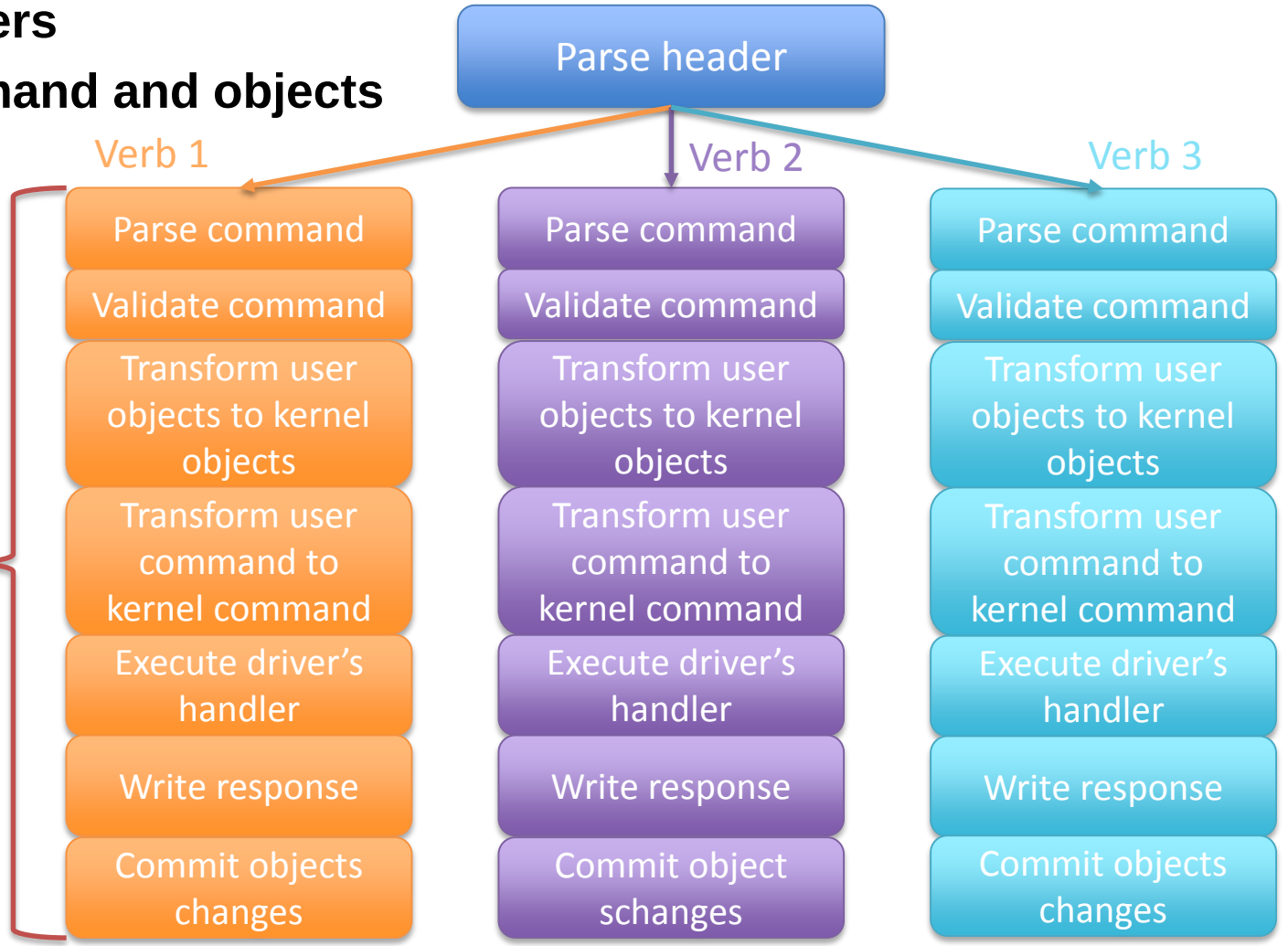
Some new features are device dependent and can't be abstracted (or at high cost!)

RDMA EXTENSIBILITY

Problem 3: Duplicate code between verbs handlers

- **A one generic method to parse headers**
- **Every verb (method) parses its command and objects by using a careful hand crafted code**
- **A lot of work to add a new verb**
- **Potential bugs!**
- **Some of the code could be changed to a shared mechanism**

A lot of parts here are duplicated!



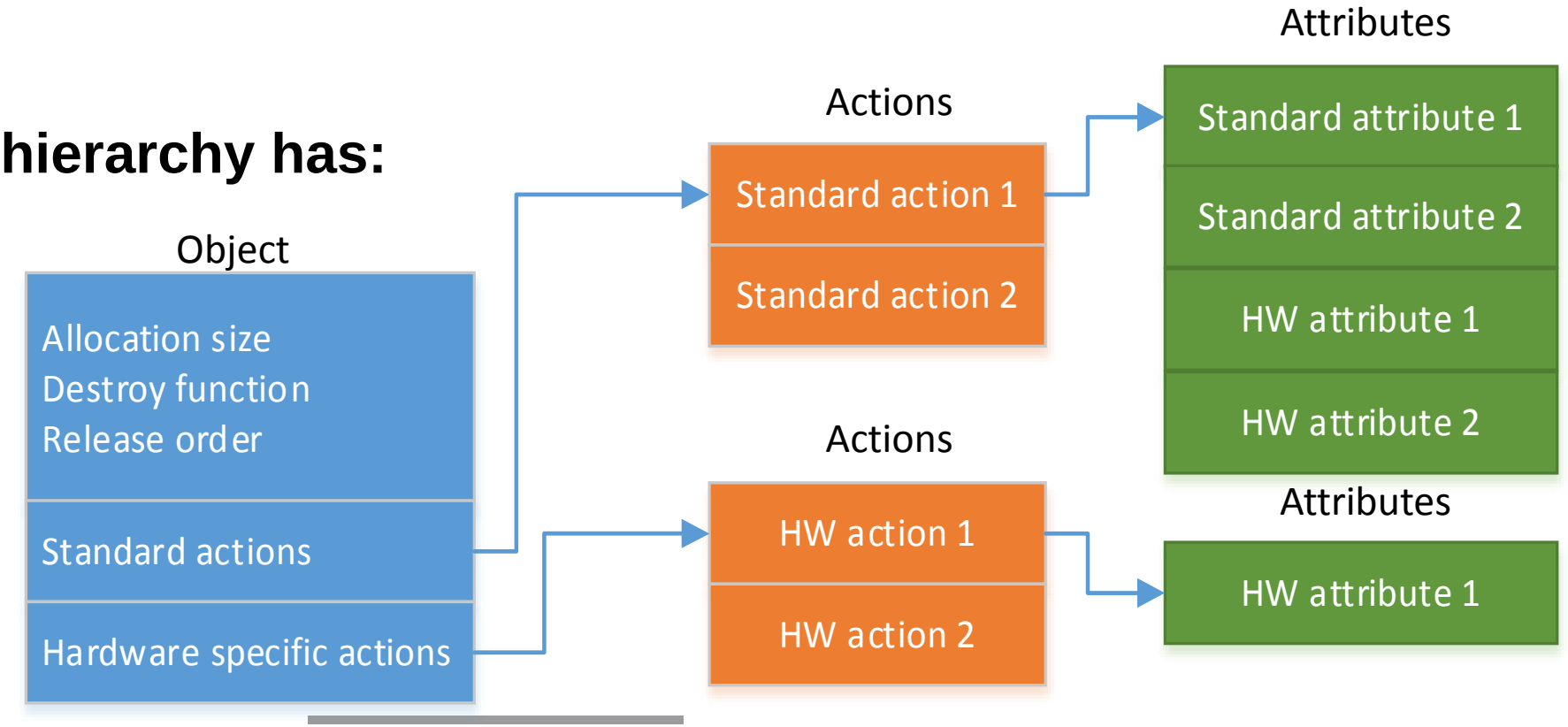
THE NEW ABI

- **Security**
  - Move to IOCTL system call
- **Extensibility**
  - Solving cumbersome extensions
    - Move to TLVs (Type-Length-Value) attributes
  - Solving feature diversity
    - Move to object oriented schema – Scales better
    - Define Objects, Methods and attributes per device
    - Parsing is done according to a specific driver and device requirements
- **Code sharing**
  - Infrastructure to take care of most hand crafted code:
    - Parsing
    - Syntax validation
    - Transform user-space objects to kernel objects
  - Keep the notion of standard objects, methods and attributes share the same code

EXTENDIBILITY

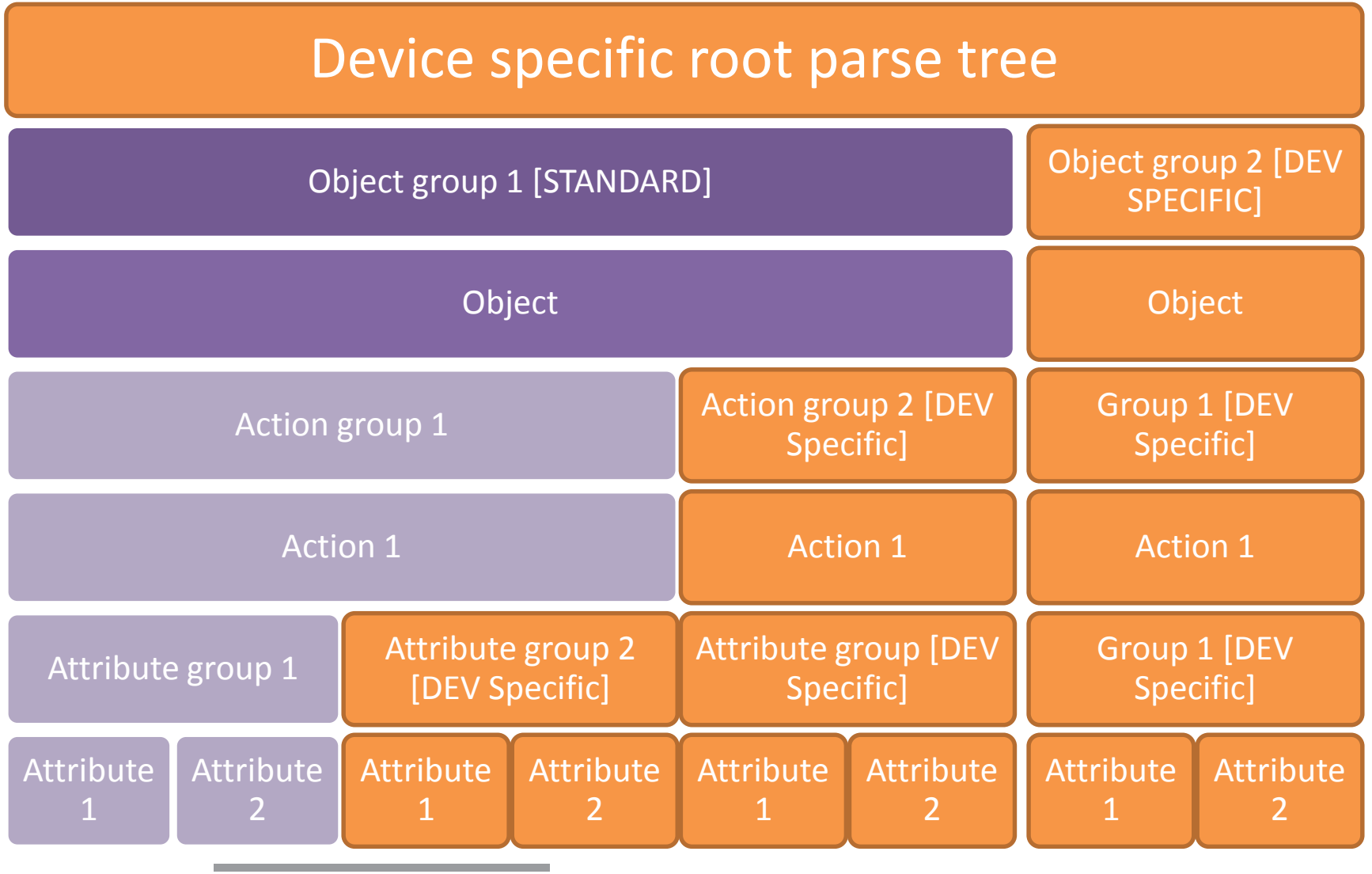
- **Replace current global method table with a hierarchy of entities:**
  - Objects
    - Actions
      - Attributes

- **Each layer in the hierarchy has:**
  - Standard entities
  - Device specific entities



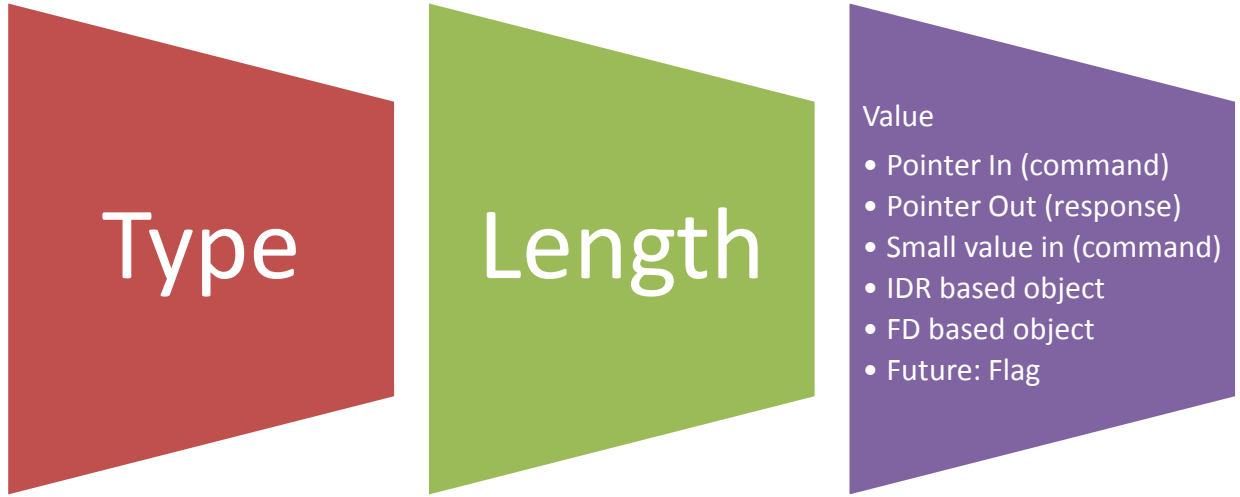
THE PARSE TREE

- Objects, methods and attributes are represented in kernel by a per-device parse tree.
- Every layer in the parse tree has 2 groups:
  - Standard entities
  - Device specific entities
- Allows to extend standard objects with device specific
  - Actions
  - Attributes
- Allows to have device specific objects



ATTRIBUTE TLVS

- **Passing Type-Length-Value based attributes**
- **Attribute classes**
  - IDR user object handle (e.g., QP, CQ)
  - FD user object handle (e.g., completion channel)
  - Pointer input (pointer to a command part)
  - Pointer out (pointer to response part)
  - Flags (group of bits)
- **Attributes can be either common or device-specific**
- **The user-space passes an array of attributes in any order.**



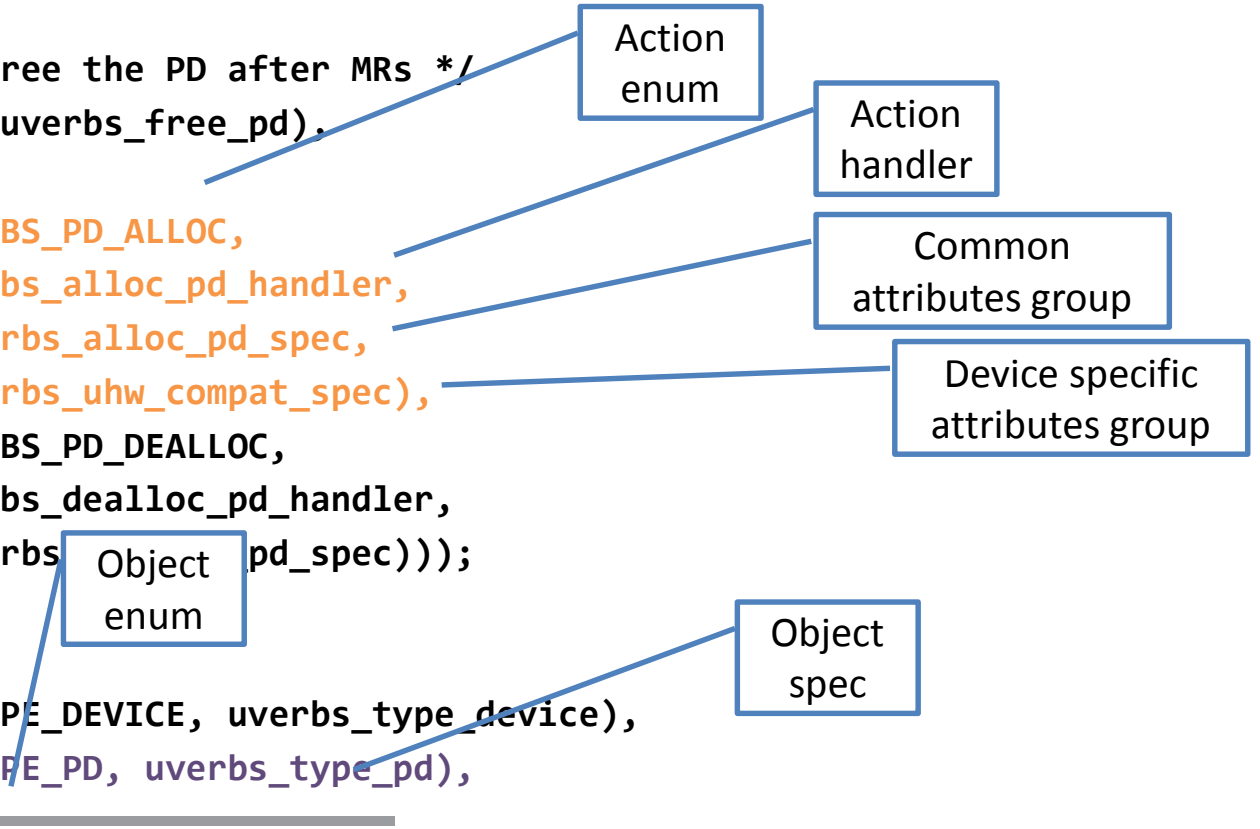
PARSE TREE SYNTAX – OBJECTS AND ACTIONS

Parse tree is defined in kernel using DSL (domain specific language):

```
int uverbs_alloc_pd_handler(struct ib_device *ib_dev, struct ib_uverbs_file *file, struct uverbs_attr_array *ctx,
                            size_t num); /* Action Handle */

DECLARE_UVERBS_TYPE(uverbs_type_pd,
                    /* 2 is used in order to free the PD after MRs */
                    &UVERBS_TYPE_ALLOC_IDR(2, uverbs_free_pd),
                    &UVERBS_ACTIONS(
                        ADD_UVERBS_ACTION(UVERBS_PD_ALLOC,
                                          uverbs_alloc_pd_handler,
                                          &uverbs_alloc_pd_spec,
                                          &uverbs_uhw_compat_spec),
                        ADD_UVERBS_ACTION(UVERBS_PD_DEALLOC,
                                          uverbs_dealloc_pd_handler,
                                          &uverbs_dealloc_pd_spec)));

DECLARE_UVERBS_TYPES(uverbs_common_types,
                     ADD_UVERBS_TYPE(UVERBS_TYPE_DEVICE, uverbs_type_device),
                     ADD_UVERBS_TYPE(UVERBS_TYPE_PD, uverbs_type_pd),
);
```



PARSE TREE SYNTAX – ATTRIBUTES

Define the attribute group:

```
DECLARE_UVERBS_ATTR_SPEC(
        uverbs_alloc_pd_spec /* Name of spec */,
        UVERBS_ATTR_IDR(ALLOC_PD_HANDLE, UVERBS_TYPE_PD,
                        UVERBS_IDR_ACCESS_NEW));
```

IDR object

```
DECLARE_UVERBS_ATTR_SPEC(
        uverbs_query_device_spec /* Name of spec */,
        UVERBS_ATTR_PTR_OUT(QUERY_DEVICE_RESP, struct ib_uverbs_query_device_resp,
                            UA_FLAGS(UVERBS_ATTR_SPEC_F_MANDATORY)),
        UVERBS_ATTR_PTR_OUT(QUERY_DEVICE_ODP, struct ib_uverbs_odp_caps),
        UVERBS_ATTR_PTR_OUT(QUERY_DEVICE_TIMESTAMP_MASK, __u64),
        UVERBS_ATTR_PTR_OUT(QUERY_DEVICE_HCA_CORE_CLOCK, __u64),
        UVERBS_ATTR_PTR_OUT(QUERY_DEVICE_CAP_FLAGS, __u64));
```

Attribute group name

Attribute number

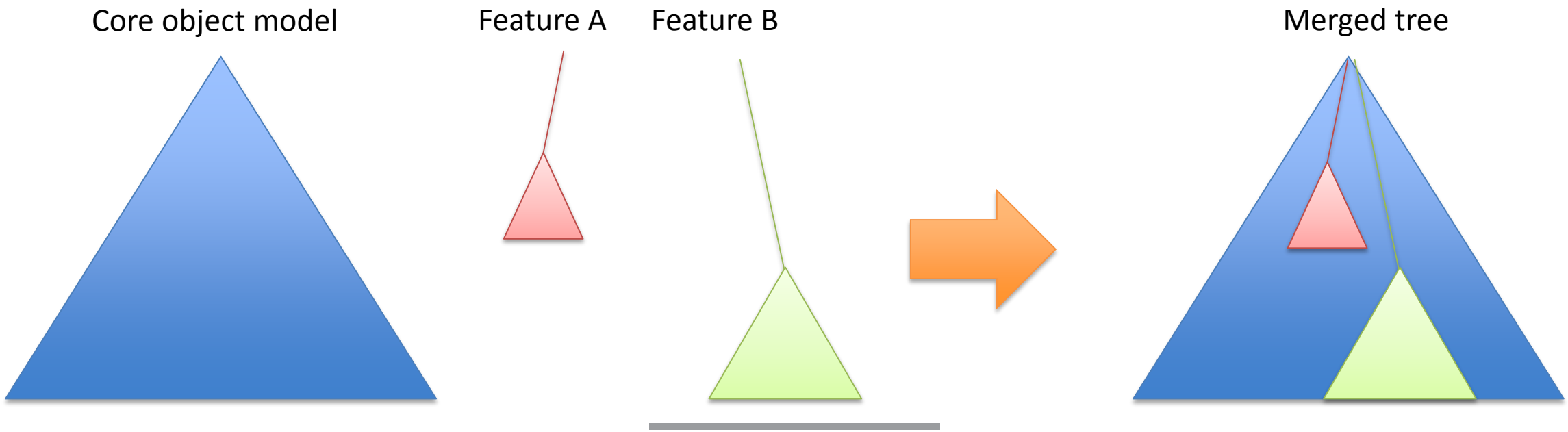
Attribute size (validated automatically)

Attribute that the user must pass (validated automatically)

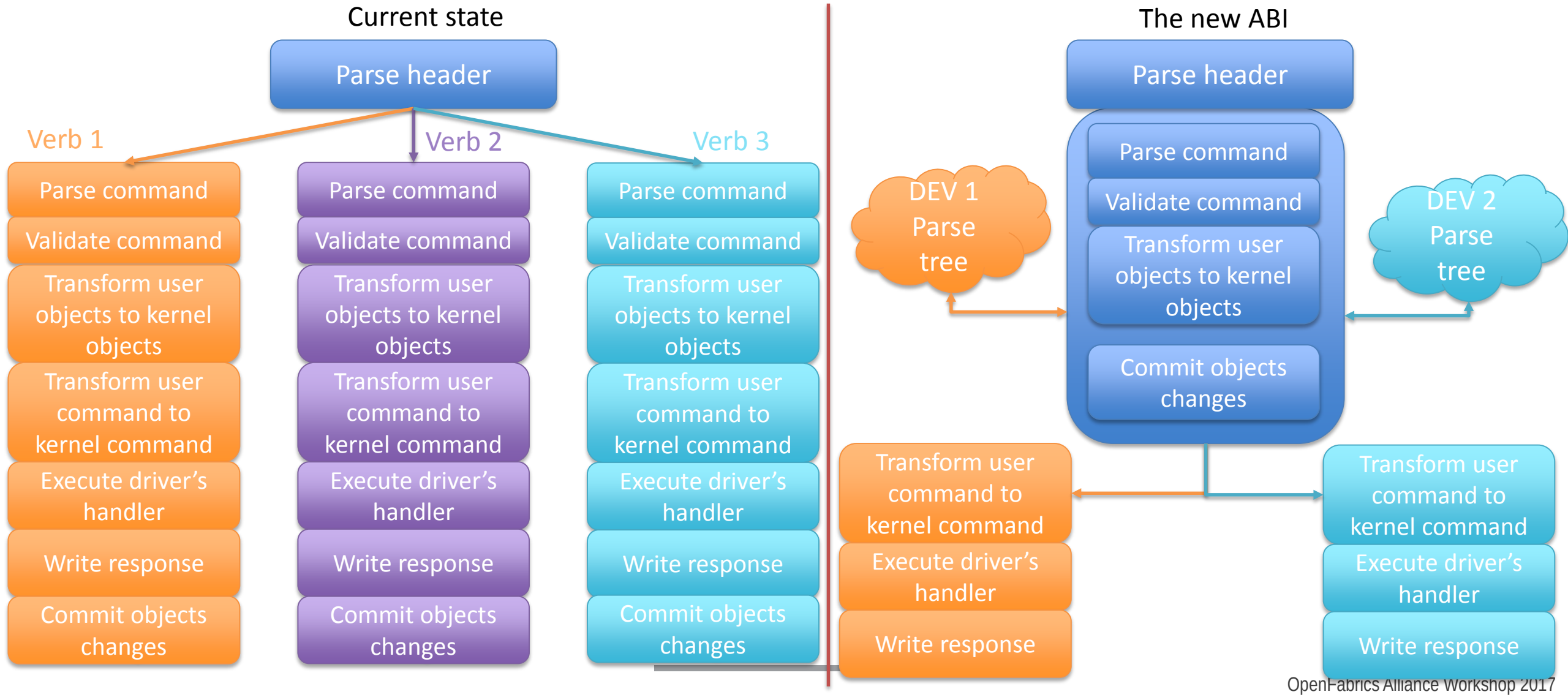
RDMA core has all the information it needs for parsing, syntax validation and objects transforming

PARSE TREE MERGING

- **Core RDMA features are described by a core parse tree**
  - Implemented by every device
- **Every feature is described by a dedicated parse tree**
  - Serves as the feature ABI specification
- **Every device indicates the feature parse trees that it supports**
  - Upon initialization, the parse trees are merged



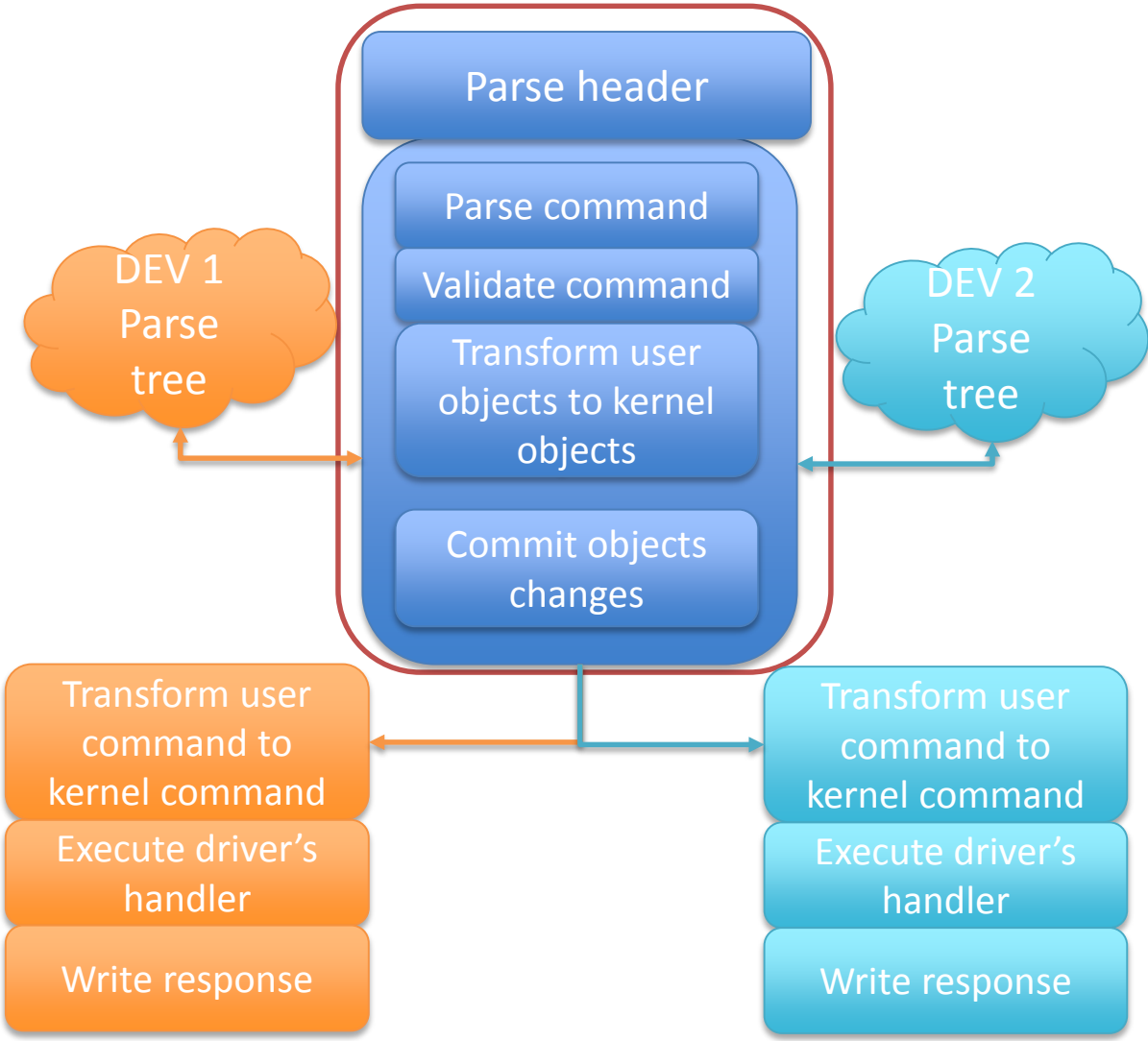
CODE SHARING



CODE SHARING

Infrastructure for parsing, validation and object transforming

- **The infrastructure is responsible for**
  - Parsing and validating the command header
  - Parsing the command attributes
    - IDR/FD → Transform to kernel object
    - Pointer IN/OUT→ Validate the size
    - Allocate require objects
    - Check that all mandatory objects were passed from user-space
  - Kernel objects locks
  - Commit the required objects
    - Allocate objects
    - Destroy objects
  - Re-order the attributes to match the kernel defined order



MAINTAINING BACKWARD COMPATIBILITY

- **For the foreseeable future, maintain the write() system-call**
- **Implement by transforming the command to the ioctl() style**
- **Straightforward mapping of existing ABI**
  - Flexible extensions for new functions and object types
- **In context teardown**
  - Release objects according to the release order defined in the parse tree
    - Reason: Sometimes we bind/unbind objects in user-space/hardware. For example, MW could created before its bounded MR, as the binding is done via user-space and the unbind could be done even only in hardware. Thus, destroy all MWs before MRs.

SUMMARY

- **Addresses the security concerns**
- **Provides an extendible mechanism**
  - Standard code
  - Device specific actions and attributes in standard objects
  - Device specific objects
- **More maintainable**
  - Infrastructure for automatic parsing, validation and transformation
  - Less error prone
- **Asynchronous interface (FD based objects)**
  - Allow future extensions to support an asynchronous interface in a standard manner
- **Enable vendors to easily introduce new features and optimizations**
- **Proper process teardown**

OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

THANK YOU

Liran Liss, Matan Barak

Mellanox Technologies LTD

HANDLER EXAMPLE

```
int uverbs_create_cq_handler(struct ib_device *ib_dev, struct ib_uverbs_file *file,
                             struct uverbs_attr_array *ctx, size_t num)
{
        struct uverbs_attr_array *common = &ctx[0];
        struct ib_ucontext *ucontext = file->ucontext;
        struct ib_ucq_object           *obj;
        struct ib_udata uhw;
        int ret;
        u64 user_handle = 0;
        struct ib_cq_init_attr attr = {};
        struct ib_cq                   *cq;
        struct ib_uverbs_completion_event_file    *ev_file = NULL;

        /* COPY MANDATORY ATTRIBUTES */
        ret = uverbs_copy_from(&attr.comp_vector, common, CREATE_CQ_COMP_VECTOR);
        ret = ret ?: uverbs_copy_from(&attr.cqe, common, CREATE_CQ_CQE);
        if (ret)
                return ret;

        /* Optional params, if they don't exist, we get -ENOENT and skip them */
        if (uverbs_copy_from(&attr.flags, common, CREATE_CQ_FLAGS) == -EFAULT ||
            uverbs_copy_from(&user_handle, common, CREATE_CQ_USER_HANDLE) == -EFAULT)
                return -EFAULT;

        /* Get completion channel if given */
        if (uverbs_is_valid(common, CREATE_CQ_COMP_CHANNEL)) {
                struct ib_uobject *ev_file_uobj =
                        common->attrs[CREATE_CQ_COMP_CHANNEL].obj_attr.uobject;

                ev_file = container_of(ev_file_uobj,
                                       struct ib_uverbs_completion_event_file,
                                       uobj_file.uobj);
                kref_get(&ev_file_uobj->ref);
        }

        /* Temporary, only until drivers get the new uverbs_attr_array */
        create_udata(ctx, num, &uhw);
```

```
        /* Handler logic from here */
        if (attr.comp_vector >= ucontext->ufile->device->num_comp_vectors)
                return -EINVAL;

        obj = container_of(common->attrs[CREATE_CQ_HANDLE].obj_attr.uobject,
                           typeof(*obj), uobject);
        obj->uverbs_file           = ucontext->ufile;
        obj->comp_events_reported  = 0;
        obj->async_events_reported = 0;
        INIT_LIST_HEAD(&obj->comp_list);
        INIT_LIST_HEAD(&obj->async_list);


        cq = ib_dev->create_cq(ib_dev, &attr, ucontext, &uhw);
        if (IS_ERR(cq))
                return PTR_ERR(cq);

        cq->device        = ib_dev;
        cq->uobject       = &obj->uobject;
        cq->comp_handler  = ib_uverbs_comp_handler;
        cq->event_handler = ib_uverbs_cq_event_handler;
        cq->cq_context    = &ev_file->ev_file;
        obj->uobject.object = cq;
        obj->uobject.user_handle = user_handle;
        atomic_set(&cq->usecnt, 0);

        /* Write response to user space */
        ret = uverbs_copy_to(common, CREATE_CQ_RESP_CQE, &cq->cqe);
        if (ret)
                goto err;

        return 0;
err:
        ib_destroy_cq(cq);
        return ret;
};
```