DPDK

A framework for representation, configuration, and management of virtual function ports

Declan Doherty (Intel)

DPDK Summit Userspace - Dublin- 2017

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit **http://www.intel.com/performance**.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

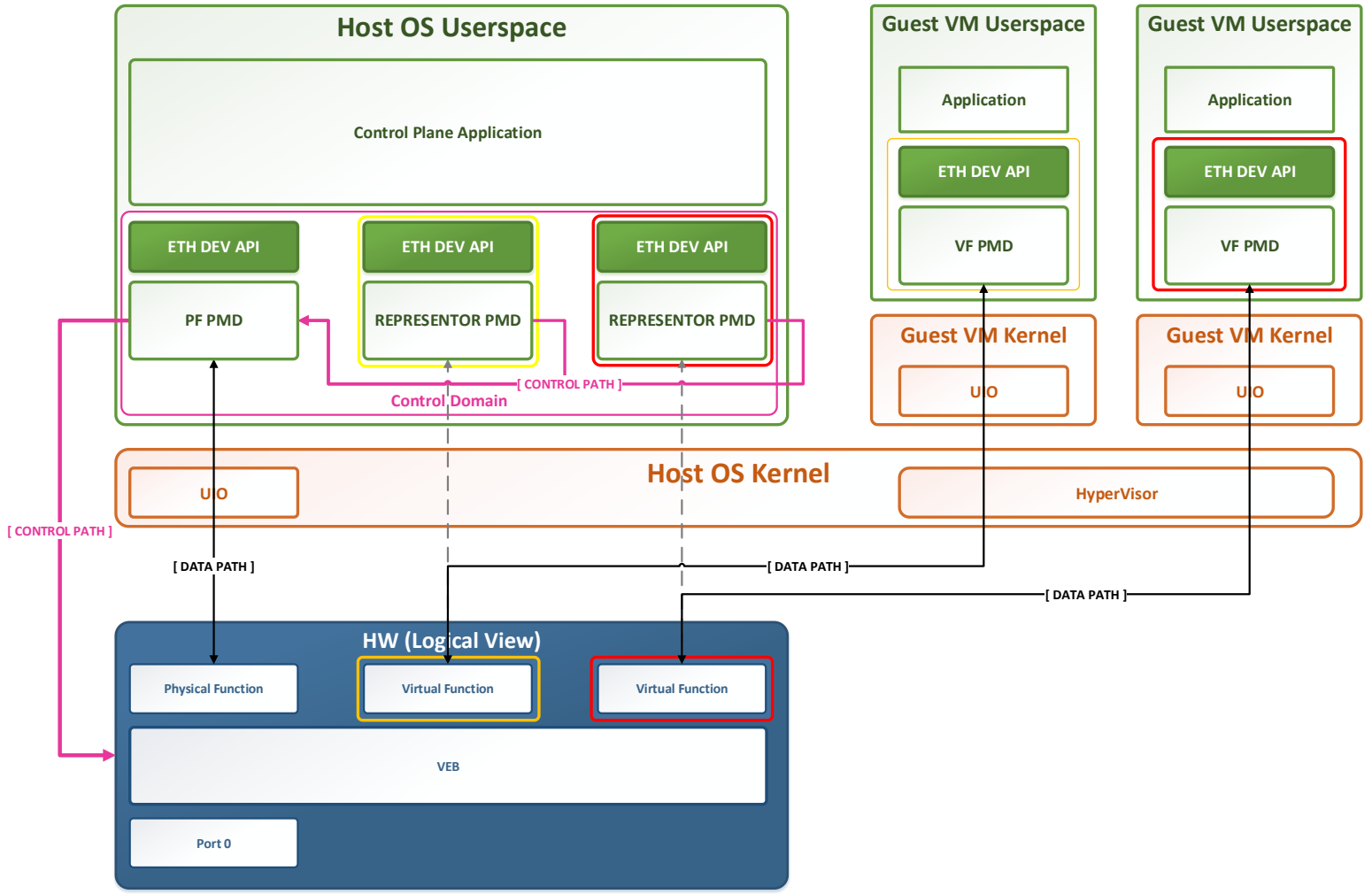
Agenda

- Port Representor Concepts
  - SR-IOV NIC
  - Multi-Port NIC
- Library Implementation Details
  - Object Model
  - Broker APIs
  - Port Representor APIs
  - Initialization Sequence
  - Example eth_dev_ops function
- Future Work

Port Representors

- Port Representors are virtual poll mode drivers (PMD) which provide a logical representation in DPDK for a port of a multi host port device.
- Primary purpose demonstrated in our RFC is to support configuration, management and monitoring of virtual functions of a physical function bound to a userspace control plane application.
- Port Representor PMDs are associated with a parent base driver which provide the backend implementations for the representor ports.
- Allows VF ports to managed using existing DPDK APIs without the need to create and maintain a set of device specific APIs.

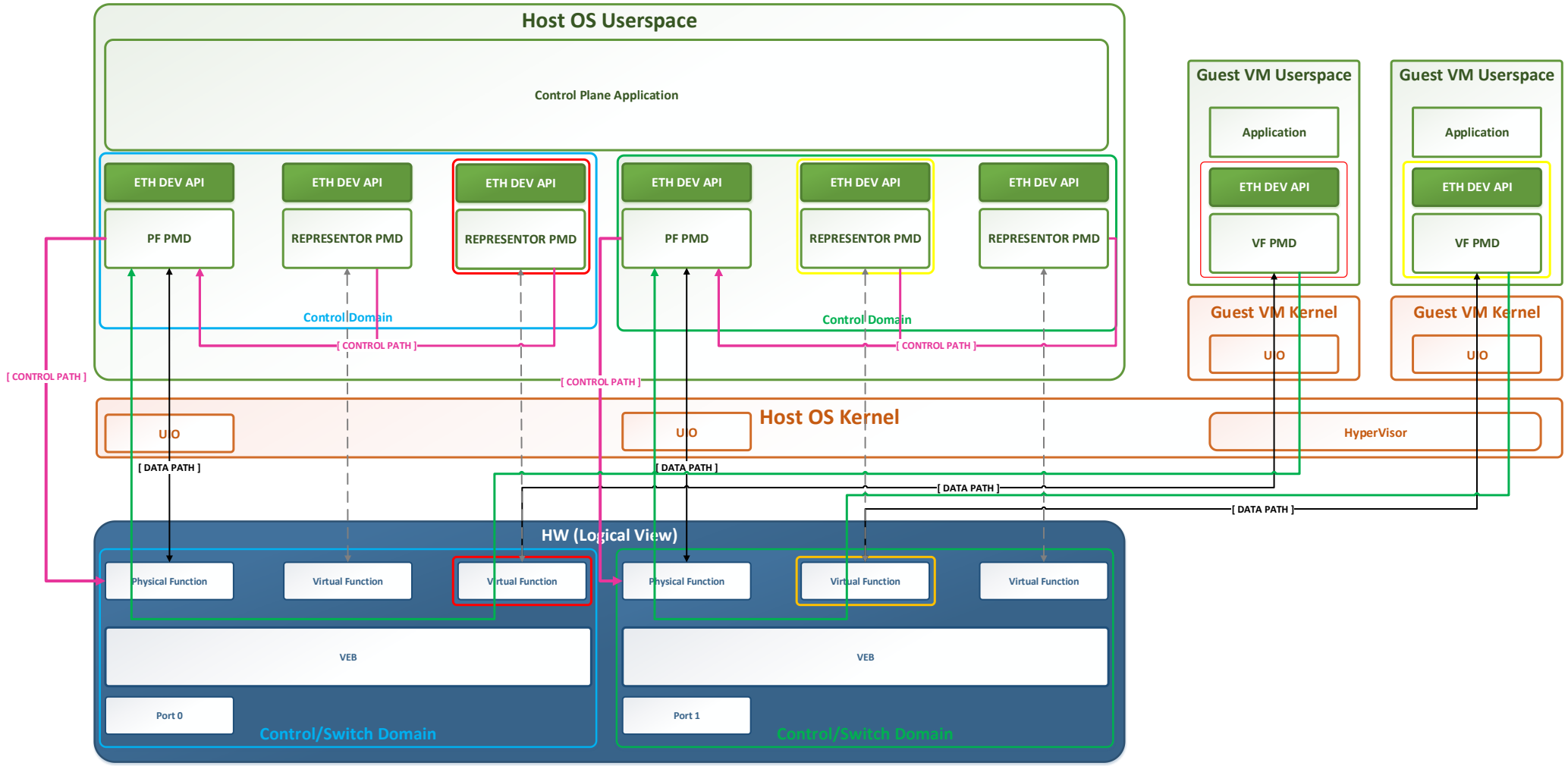
Port Representors for a NIC supporting SR-IOV



Port Representors for a NIC supporting SR-IOV

- Host Control Plane Application
  - Port Representor PMDs are created to represent each virtual function (VF) of the PF PMD.
  - Port Representor PMD control plane is through eth_dev_ops implemented by base driver (PF PMD).
  - Port properties configured through representor:
    - MAC, VLAN
    - Promiscuous Mode
    - Multicast/Broadcast
- Guest Application
  - Configuration of data path only:
    - Tx/Rx Queues
    - RSS/Flow Director
    - Offloads
- No data path supported for this use case.

Port Representors for a multi-port devices

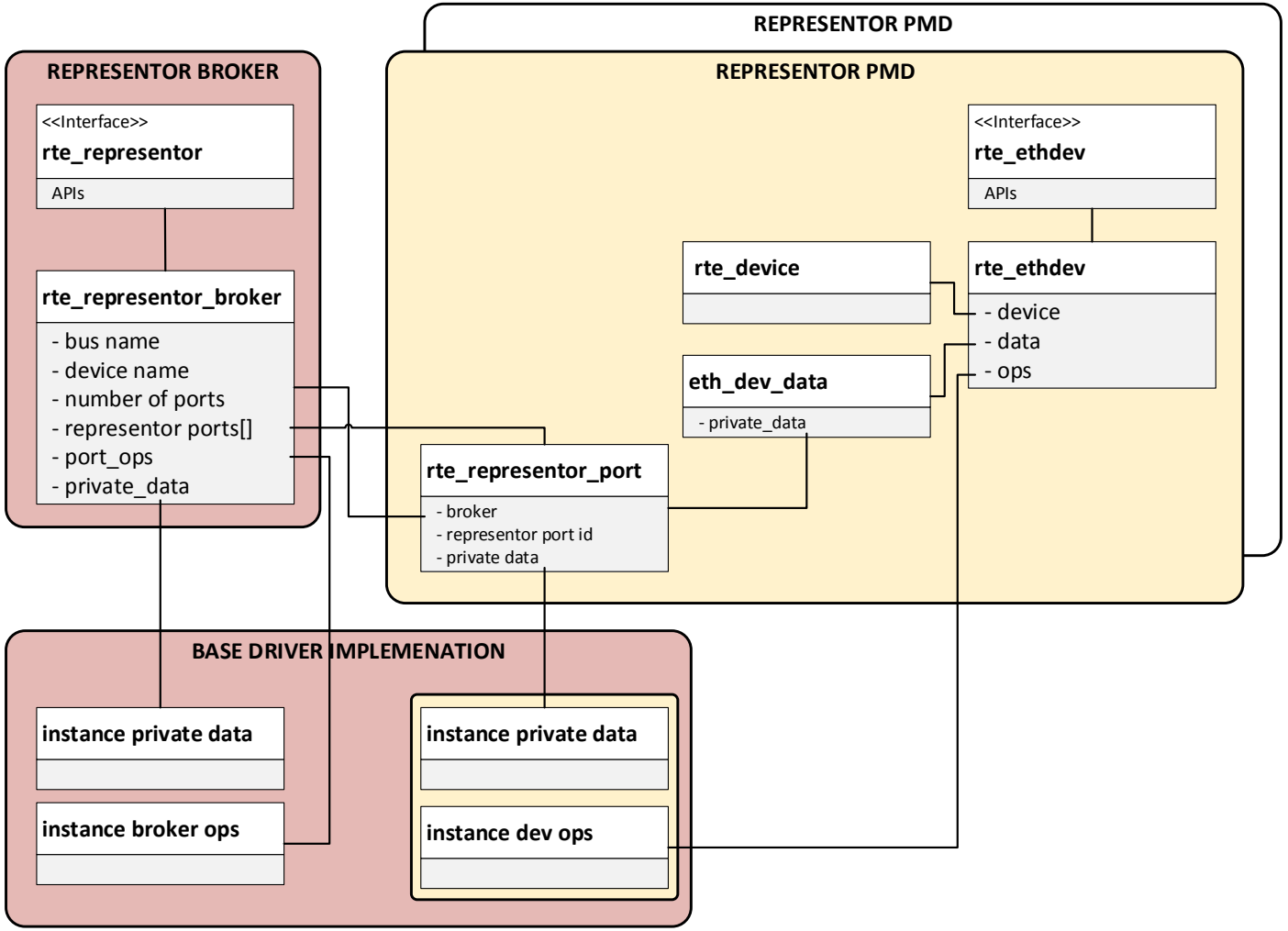


Port Representors for a multi-port devices

- Introduces the new concept of switch/control domain to ethdev's
  - Base driver defines the switch/control domain.
  - Each representor port inherits the domain from it's root device.
- If hardware supports advance port-to-port switching capabilities then switch domain can be use by application to know whether logical ports are in the same domain.

Library Implementation

Port Representor (Object Model)



Port Representor

- Representor PMD
  - Generic skeleton PMD with infrastructure for creation of representor port and registration with broker.
  - All configuration including capabilities and dev_ops functions configured by broker/ base driver.
  - No restrictions on port representor capabilities set by framework, all are controlled by the base driver.
- Representor Broker
  - Integrates into base driver (eg PF PMD)
  - Base driver is not required to be an ethdev.
  - Base driver configures number of representor ports supported and provides port configuration functions for representor port initialisation

Representor Broker APIs

- Register / Un-Register Broker in base driver
  - `int rte_representor_broker_register(struct rte_representor_broker *broker`
  - `int rte_representor_broker_unregister(struct rte_representor_broker *broker);`

```
struct rte_representor_broker {
    TAILQ_ENTRY(rte_representor_broker) next;

    const char *bus;
    const char *device;
    /**< Base Device Bus/Device Name */
    uint16_t nb_virtual_ports;
    struct rte_representor_port *virtual_ports;
    /**< Array of virtual(representor) ports */
    struct rte_representor_broker_port_ops *port_ops;
    /**< Port Initialisation Functions */
    void *private_data;
    /**< Base Driver private data */
};
```

```
struct rte_representor_broker_port_ops
{
    port_priv_data_set;
    port_priv_data_free;
    port_capabilities_set;
    port_ops_get;
};
```

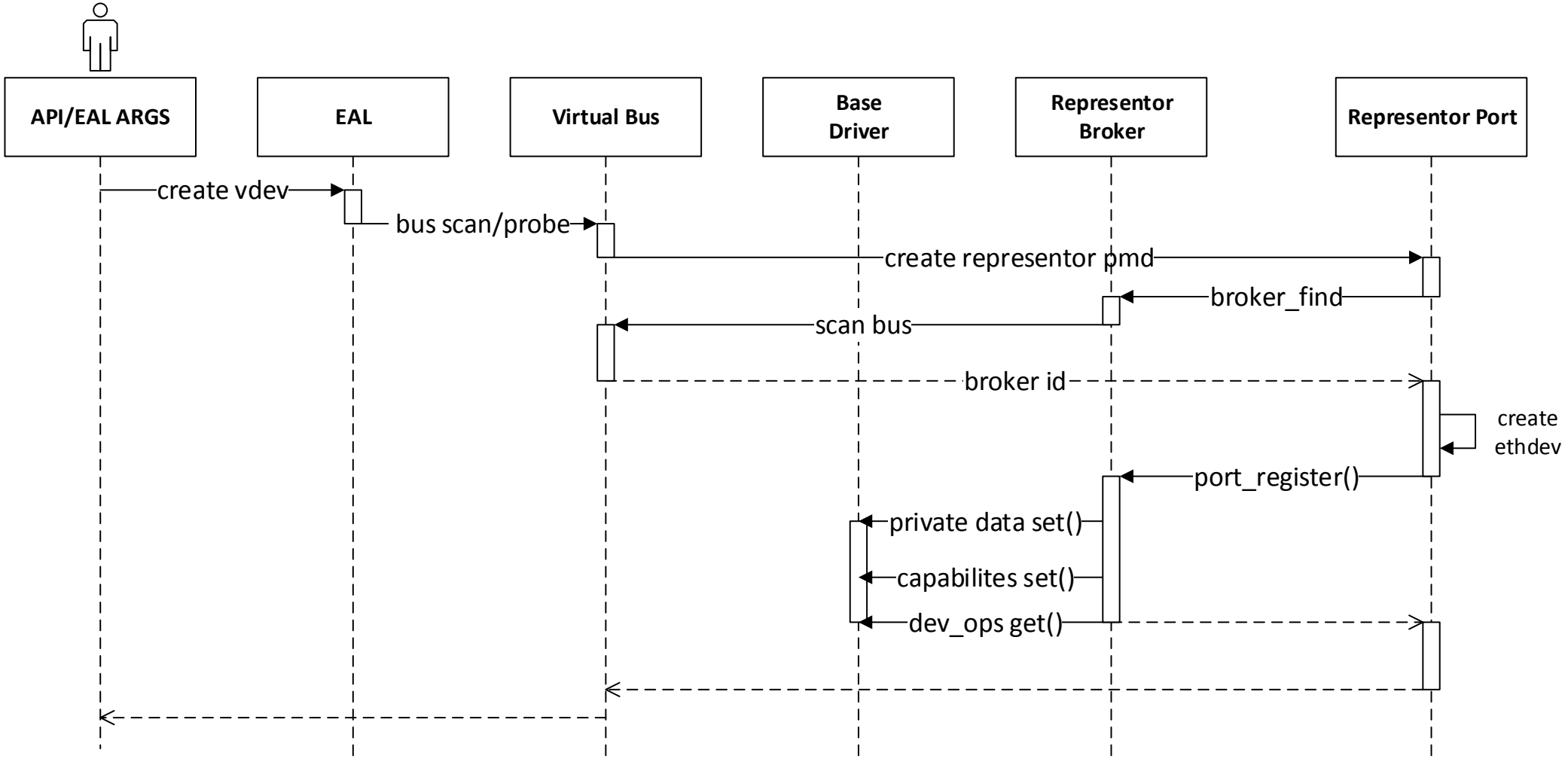
Representor Port APIs

- `struct rte_representor_broker *`
  `rte_representor_broker_find(const char *bus, const char *device);`

- `int rte_representor_port_register(struct rte_representor_broker *broker,`
  `uint32_t vport_id,`
  `struct rte_eth_dev *ethdev);`

- `Int rte_representor_port_unregister(struct rte_eth_dev *ethdev);`

Port Representor (Initialisation Sequence)



Example eth_dev_ops function


```
struct rte_representor_port {
    struct rte_representor_broker *broker;
    uint16_t id;
    struct rte_eth_dev *ethdev;
    enum {
        RTE_REPRESENTOR_PORT_INVALID,
        RTE_REPRESENTOR_PORT_VALID
    } state;
    void *priv_data;
};
```

```
struct i40e_representor_priv_data {
    struct rte_eth_dev *pf_ethdev;
};
```

```
static void
i40e_port_representor_dev_infos_get(struct rte_eth_dev *ethdev,
                struct rte_eth_dev_info *dev_info){

    struct rte_representor_port *port_rep = ethdev->data->dev_private;
    struct i40e_representor_priv_data *i40e_priv_data = port_rep->priv_data;

    /**< Function Implementation */
    …
};
```

Future Work

Possible Future Work

- Data path enablement (next talk!)
- Enable hot-plug support so representor ports get created automatically, as VF are created.
- Port-to-Port switching through rte_flow using logical port id's.
- Advance port capabilities management
  - Port representor could be used to define capabilities of the underlying port. e.g. make a VF untrusted so it can change it's MAC address etc.
  - Limit hardware resources port can use, e.g. number of flow director rules.
- Policy enforcement
  - stop VF over riding configuration applied in control plane application.
  - Would require hooks into base driver to catch configuration requests coming through hardware mailbox

Questions?

Declan Doherty

<declan.doherty@intel.com>