DPDK

DPDK Bus Updates

Ferruh Yigit, Intel

DPDK Summit Userspace – Dublin - 2017

Legal Notices and Disclaimers

Who am I

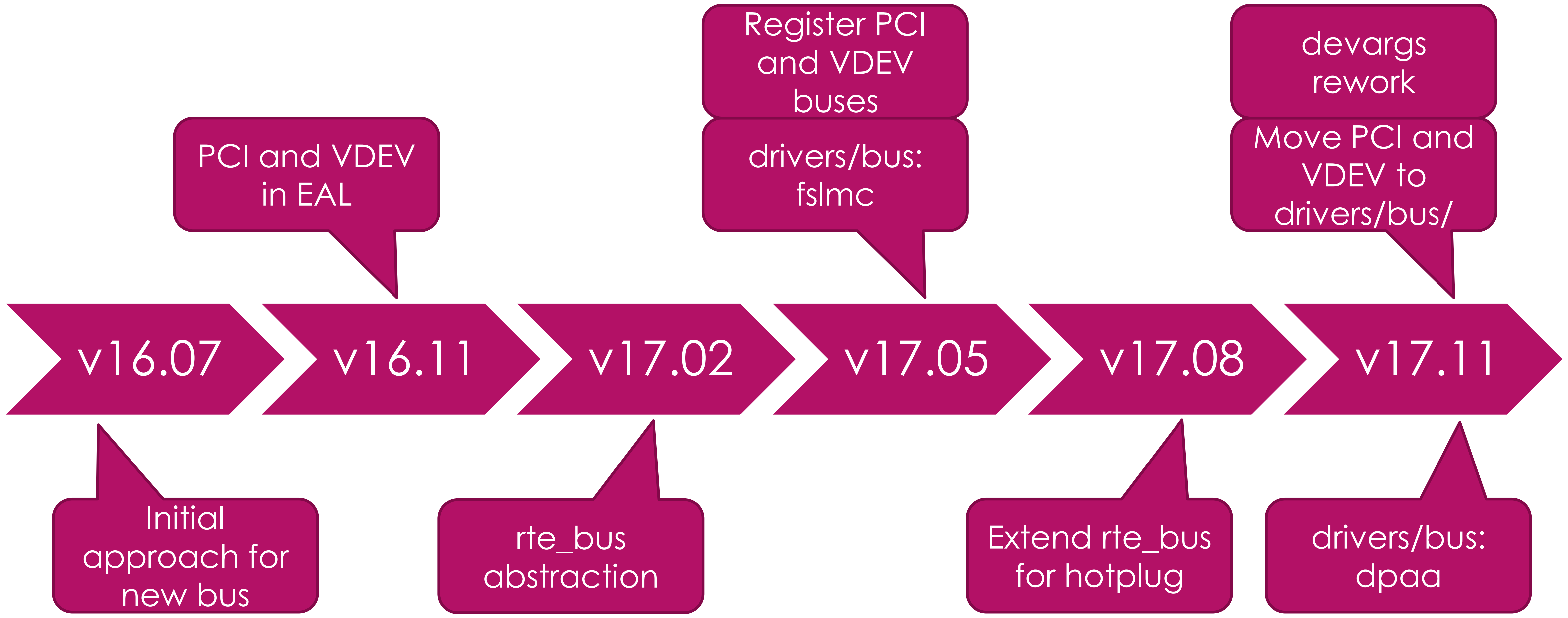
- Software Engineer in Intel and a DPDK developer
- DPDK next-net maintainer

Agenda

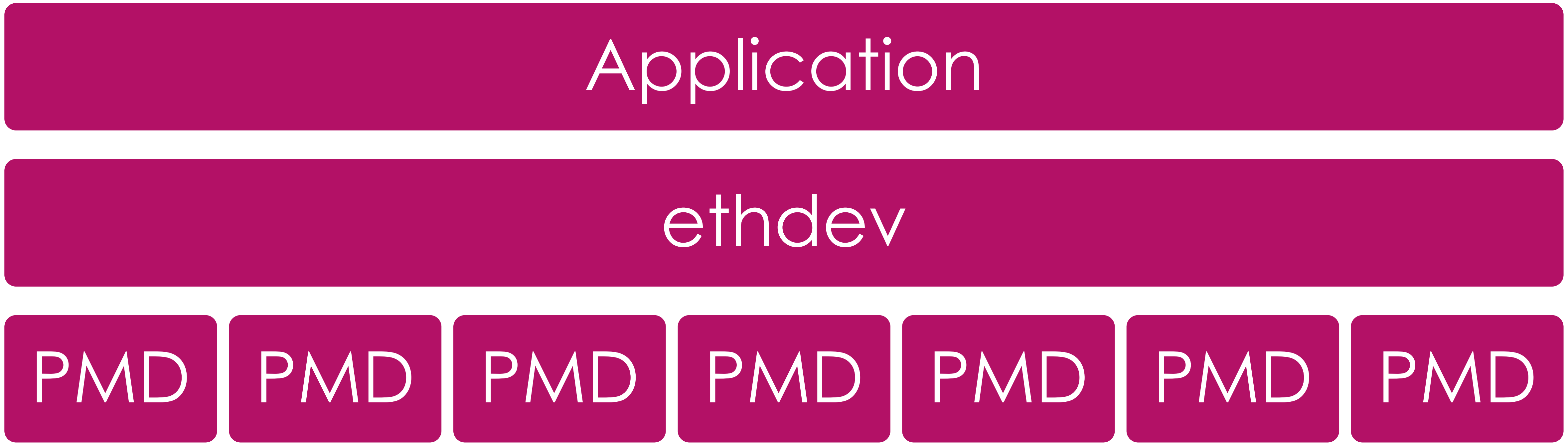
- What is the bus infrastructure in DPDK?
- History of the rte_bus.

Hard to resist

Quick History



DPDK Bus



- Where is the bus here?
- Nowhere! Only PMDs should know about the bus, internal to DPDK.

What is the bus infrastructure in DPDK

- It doesn’t drive bus controllers. DPDK is not Linux.
- It is for logically grouping devices.
- Bus infrastructure responsibilities:
  - Scan devices on given bus.
  - Match device – drivers on given bus.
  - Plug / unplug a device on given bus.
- It enables creating helper functions for PMDs.

Bus scope and expectations

EAL

- Adding new buses should be easy
- Adding a new bus should not effect the core EAL code
- Bus specific code should be moved from EAL to bus

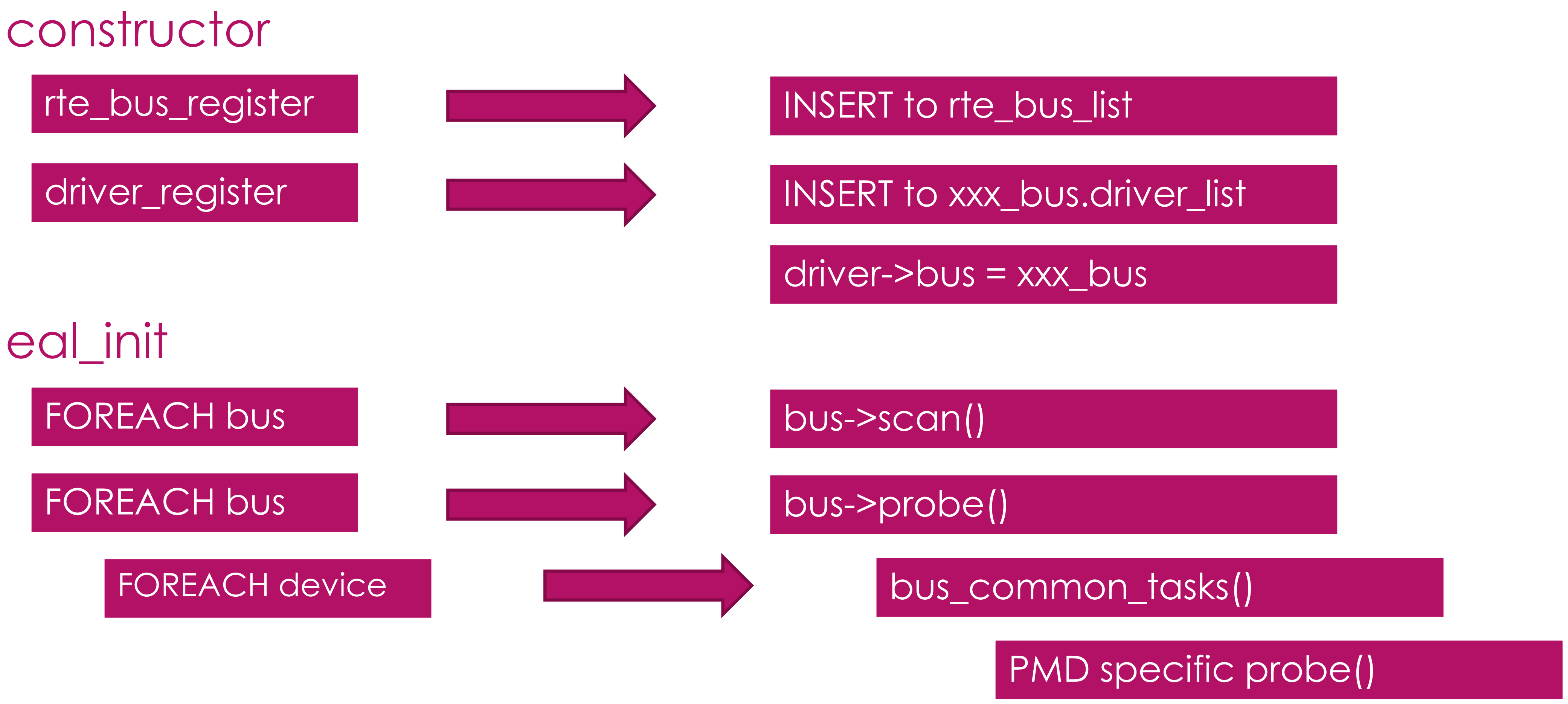
ethdev

- Functional device layers should be bus agnostic
- Adding new bus types should not require change in ethdev
- Bus related information should be saved in ethdev in a generic way

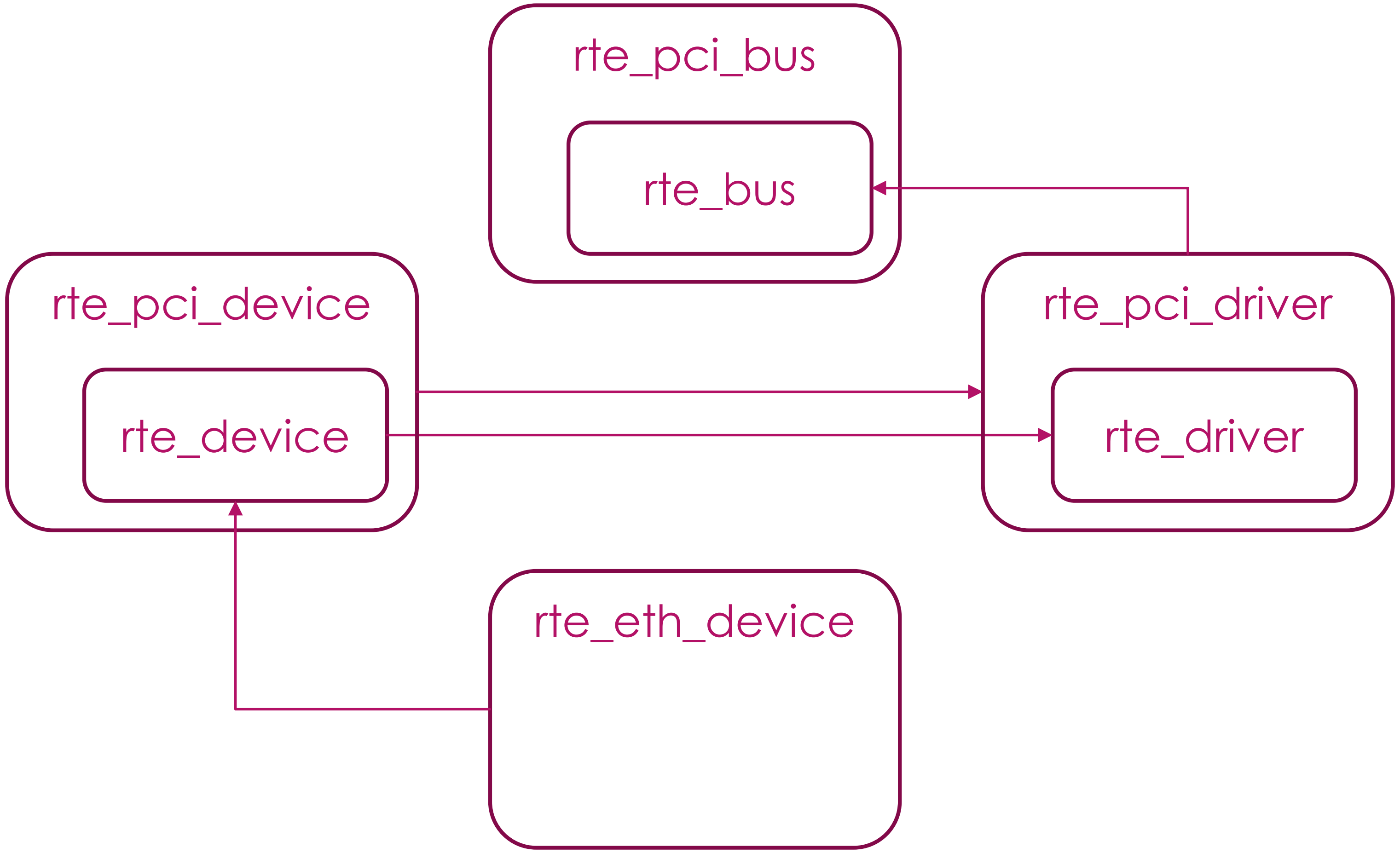
PMD

- PMDs knows about the bus
- PMDs need information from EAL related to bus
- Common tasks on given bus should be easy to do for PMDs

Bus process flow (as 17.08)



Related data structures (as 17.08)



16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

ata structures

```
struct rte_device {
        TAILQ_ENTRY(rte_device) next; /**< Next device */
        const struct rte_driver *driver;/**< Associated driver
        int numa_node;                /**< NUMA node connectio
        struct rte_devargs *devargs;  /**< Device user arguments
};

struct rte_driver {
        TAILQ_ENTRY(rte_dr
        const char *name;
        const char *alias;
};

struct rte_pci_device
        TAILQ_ENTRY(rte_pc                                       CI device. */
        struct rte_device                                        device */
        struct rte_pci_addr                                      n. */
        struct rte_pci_id id;            /**< PCI ID. */
        struct rte_mem_resour            esource[PCI_MAX_RESOURCE];
                                         /**< PCI Memory Resource */
        struct rte_intr_ha       intr_handle;  /**< Interrupt handle */
        struct rte_pci_d    ver *driver;       /**< Associated driver */
        uint16_t max_v  s;                     /**< sriov enable if not zero */
        enum rte_kernel_driver kdrv;           /**< Kernel driver passthrough */
};

struct rte_pci_driver {
        TAILQ_ENTRY(rte_pci_driver) next;      /**< Next in list. */
        struct rte_driver driver;              /**< Inherit core driver. */
        pci_probe_t *probe;                    /**< Device Probe function. */
        pci_remove_t *remove;                  /**< Device Remove function. */
        const struct rte_pci_id *id_table;     /**< ID table, NULL terminated. */
        uint32_t drv_flags;                    /**< Flags contolling handling of device. */
};
```

```
struct rte_vdev_driver {
        TAILQ_ENTRY(rte_vdev_driver) ne                   . */
        struct rte_driver driver;                         tion. */
        rte_vdev_probe_t *probe;                          ction. */
        rte_vdev_remove_t *remove;
};
```

!NO vdev_device

```
struct rte_eth_dev {
        const struct eth_driver *driver;  **< Driver for this device */
        struct rte_pci_device *pci_dev; /**< PCI info. supplied by probing */
        uint8_t attached; /**< Flag indicating the port is attached */
        …
};

struct eth_driver {
        struct rte_pci_driver pci_drv;    /**< The PMD is also a PCI driver. */
        eth_dev_init_t eth_dev_init;      /**< Device init function. */
        eth_dev_uninit_t eth_d   uninit;  /**< Device uninit function. */
        unsigned int dev      ate_size;   /**< Size of device private data. */
};
```

NO vdev devices

Too small to read? That is OK! Will not discuss all these details in this talk.

PCI dev in ethdev device

eth driver inherited from PCI driver

16.11- 17.02 – 17.05 – 17.08 – 17.11

eal init flow

```
rte_eal_init
        eal_parse_args
                eal_parse_common_option
                        rte_eal_devargs_add
                                insert(devargs_list, devargs)
        rte_eal_pci_init
                rte_eal_pci_scan
                        insert(pci_device_list, dev)
        rte_eal_dev_init
                foreach(devargs, devargs_list) rte_eal_vdev_init()
        rte_eal_pci_probe
                foreach(dev, pci_device_list) probe_all
```

vdev scan

PCI scan

vdev init

PCI probe

Bus functionalities hardcoded

PMD code

```
static struct eth_driver rte_ixgbe_pmd = {
            .pci_drv = {
                    .id_table = pci_id_ixgbe_map,
                    .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC |
                            RTE_PCI_DRV_DETACHABLE,
                    .probe = rte_eth_dev_pci_probe,   !GENERIC PCI
                    .remove = rte_eth_dev_pci_remove,
            },
            .eth_dev_init = eth_ixgbe_dev_init,
            .eth_dev_uninit = eth_ixgbe_dev_uninit,
            .dev_private_size = sizeof(struct ixgbe_adapter),
};

RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd.pci_drv);


         ruct rte_vdev_driver pmd_null_drv = {
            .probe = rte_pmd_null_probe,
            .remove = rte_pmd_null_remove,


RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

related data structures

```
struct rte_device {
        TAILQ_ENTRY(rte_device) next; /**< Next device */
        const struct rte_driver *driver;/**< Associated driver */
        int numa_node;              /**< NUMA node connection */
        struct rte_devargs *devargs;  /**< Device user arguments */
};

struct rte_driver {
        TAILQ_ENTRY(rte_driver) next;  /**< Next in list. */
        const char *name;                  /**< Driver name. */
        const char *alias;             /**< Driver alias. */
};

struct rte_pci_device {
        TAILQ_ENTRY(rte_pci_device) next;       /**< Next probed PCI
        struct rte_device device;               /**< Inherit cor
        struct rte_pci_addr addr;               /**< PCI locat
        struct rte_pci_id id;                   /**< PCI ID.
        struct rte_mem_resource mem_resource[PCI_MAX_RESOURC
                                                /**< PCI Memo
        struct rte_intr_handle intr_handle;     /**< Interrupt
        struct rte_pci_driver *driver;          /**< Associated d
        uint16_t max_vfs;                       /**< sriov enable if ne
        enum rte_kernel_driver kdrv;            /**< Kernel driver passthrough */
};

struct rte_pci_driver {
        TAILQ_ENTRY(rte_pci_driver) next;       /**< Next in list. */
        struct rte_driver driver;               /**< Inherit core driver. */
        pci_probe_t *probe;                     /**< Device Probe function. */
        pci_remove_t *remove;                   /**< Device Remove function. */
        const struct rte_pci_id *id_table;      /**< ID table, NULL terminated. */
        uint32_t drv_flags;                     /**< Flags contolling handling of device. */
};
```

```
struct rte_vdev_driver {
        TAILQ_ENTRY(rte_vdev_driver) nex
        struct rte_driver driver;
        rte_vdev_probe_t *probe;                                  */
        rte_vdev_remove_t *remove;                                */
};
```

!NO vdev_device

```
struct rte_eth_dev {
        const struct eth_driver *      er;/**< Driver for this device */
        struct rte_device *device; /**< Backing device */
        uint8_t attached; /**< Flag indicating the port is attached */
        …
};

   uct eth_driver {
        struct rte_pci_driver pci_drv;     /**< The PMD is also a PCI driver. */
        eth_dev_init_t eth_dev_init;       /**< Device init function. */
        eth_dev_uninit_t eth_dev_uninit;   /**< Device uninit function. */
        unsigned int dev_private_size;     /**< Size of device private data. */

struct rte_bus {
        TAILQ_ENTRY(rte_bus) next;   /**< Next bus object in linked list */
        const char *name;            /**< Name of the bus */
        rte_bus_scan_t scan;         /**< Scan for devices attached to bus */
        rte_bus_probe_t probe;       /**< Probe devices on bus */
};
```

ethdev has generic device

New bus abstraction

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

eal init flow

PMD code

```
rte_eal_init
        eal_parse_args
                eal_parse_common_option
                        rte_eal_devargs_add
                                insert(devargs_list, devargs)
        rte_eal_pci_init
                rte_eal_pci_scan
                        insert(pci_device_list, dev)
        rte_bus_scan
                foreach(bus, rte_bus_list) bus->scan() !NO BUS REGISTERED
        rte_bus_probe
                foreach(bus, rte_bus_list) bus->probe()
        rte_eal_pci_probe
                foreach(dev, pci_device_list) probe_all
        rte_eal_dev_init
                foreach(devargs, devargs_list) rte_eal_vdev_init()
```

```
static struct eth_driver rte_ixgbe_pmd = {
           .pci_drv = {
                   .id_table = pci_id_ixgbe_map,
                   .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
                   .probe = rte_eth_dev_pci_probe,
                   .remove = rte_eth_dev_pci_remove,
           },
           .eth_dev_init = eth_ixgbe_dev_init,
           .eth_dev_uninit = eth_ixgbe_dev_uninit,
           .dev_private_size = sizeof(struct ixgbe_adapter),

TER_PCI(net_ixgbe, rte_ixgbe_pmd.pci_drv);

vdev_driver pmd_null_drv = {
te_pmd_null_probe,
rte_pmd_null_remove,

GISTER_VDEV(net_null, pmd_null_drv);
```

Bus functions are added into eal_init, no bus registered yet

vdev scan

PCI scan

rte_bus

PCI probe

vdev init

bus->scan

bus->probe

16.11- 17.02 – 17.05 – 17.08 – 17.11

related data structures

PCI bus inherited from rte_bus

vdev device struct created

eth driver removed

```
struct rte_device {
        TAILQ_ENTRY(rte_device) next; /**< Next d
        const char *name;             /**< D
        const struct rte_driver *driver;/**< Asso
        int numa_node;                /**< NUMA n
        struct rte_devargs *devargs;  /**< Device
};

struct rte_driver {
        TAILQ_ENTRY(rte_driver) next;  /**< Next in list. */
        const char *name;                  /**< Driver name. */
        const char *alias;             /**< Driver alias. */
};

struct rte_pci_device {
        TAILQ_ENTRY(rte_pci_device) next;       /**< Next probed
        struct rte_device device;               /**< Inherit c
        struct rte_pci_addr addr;               /**< PCI loca
        struct rte_pci_id id;                   /**< PCI ID.
        struct rte_mem_resource mem_resource[PCI_MAX_RESOURC
                                                /**< PCI Mem
        struct rte_intr_handle intr_handle;     /**< Interru
        struct rte_pci_driver *driver;          /**< Associat
        uint16_t max_vfs;                       /**< sriov enab
        enum rte_kernel_driver kdrv;            /**< Kernel drive
        char name[PCI_PRI_STR_SIZE+1];              /**< PCI location (ASCII) */
};

struct rte_pci_driver {
        TAILQ_ENTRY(rte_pci_driver) next;       /**< Next in list. */
        struct rte_driver driver;               /**< Inherit core driver. */
        struct rte_pci_bus *bus;                    /**< PCI bus reference. */
        pci_probe_t *probe;                     /**< Device Probe function. */
        pci_remove_t *remove;                   /**< Device Remove function. */
        const struct rte_pci_id *id_table;      /**< ID table, NULL terminated. */
        uint32_t drv_flags;                     /**< Flags contolling handling of device. */
};
```

```
struct rte_pci_bus {
        struct rte_bus bus;                 /**< Inherit the generic class */
        struct rte_pci_device_list device_list;  /**< List of PCI devices */
        struct rte_pci_driver_list driver_list;  /**< List of PCI drivers */
};

struct rte_vdev_driver {
        TAILQ_ENTRY(rte_vdev_driver) next;
        struct rte_driver driver;      /*
        rte_vdev_probe_t *probe;       /*                        */
        rte_vdev_remove_t *remove;                             . */
};

struct rte_vdev_device {
        TAILQ_ENTRY(rte_vdev_device) next;             ext attached vdev */
        struct rte_device device;               /**< Inherit core device */
};

struct rte_eth_dev {
        struct rte_device *device; /**< Backing device */
        enum rte_eth_dev_state state; /**< Flag indicating the port state */
        …
```

!eth_driver REMOVED

```
struct rte_bus {
        TAILQ_ENTRY(rte_bus) next;   /**< Next bus object in linked list */
        const char *name;            /**< Name of the bus */
        rte_bus_scan_t scan;         /**< Scan for devices attached to bus */
        rte_bus_probe_t probe;       /**< Probe devices on bus */
};
```

16.11- 17.02 – 17.05 – 17.08 – 17.11

bus code

```
struct rte_pci_bus rte_pci_bus = {
        .bus = {
                .scan = rte_pci_scan,
                .probe = rte_pci_probe,
        },
        .device_list = TAILQ_HEAD_INIT                                ,
        .driver_list = TAILQ_HEAD_INIT                                ),
};

RTE_REGISTER_BUS(PCI_BUS_NAME, rte_pci_bus.bu

struct rte_fslmc_bus rte_fslmc_bus = {
        .bus = {
                .scan = rte_fslmc_scan
                .probe = rte_fslmc_prob
        },
        .device_list = TAILQ_HEAD_INIT                                t),
        .driver_list = TAILQ_HEAD_INIT                                st),
};

RTE_REGISTER_BUS(FSLMC_BUS_NAME, rte_fslmc_bus.
```

Existing buses registered

New bus driver

```
static struct rte_bus rte_vdev_bus = {
        .scan = vdev_scan,
        .probe = vdev_probe,
};

RTE_INIT(rte_vdev_bus_register);

static void rte_vdev_bus_register(void)
{
        static int registered;

        if (registered)
                return;

        registered = 1;
        rte_vdev_bus.name = RTE_STR(virtual);
        rte_bus_register(&rte_vdev_bus);
}
```

16.11- 17.02 – **17.05** – 17.08 – 17.11

eal init flow

```
rte_eal_init
        eal_parse_args
                eal_parse_common_option
                        rte_eal_devargs_add
                                insert(devargs_list, devargs)
        rte_bus_scan
                foreach(bus, rte_bus_list) bus->scan()
        rte_bus_probe
                foreach(bus, rte_bus_list) bus->probe()
                if (vbus)
                        vbus->probe()
```

Hardcoded buses removed

bus->scan

bus->probe

PCI

vdev

fslmc

PMD code

```
eth_ixgbe_pci_probe
        rte_eth_dev_pci_generic_probe(pci_dev, eth_ixgbe_dev_init)

eth_ixgbe_pci_remove
        rte_eth_dev_pci_generic_remove(pci_dev, eth_ixgbe_dev_uninit)

static struct rte_pci_driver rte_ixgbe_pmd = {
            .id_table = pci_id_ixgbe_map,
            .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
            .probe = eth_ixgbe_pci_probe,
            .remove = eth_ixgbe_pci_remove,
 };

 RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe

 static struct rte_vdev_driver pmd_null_d
           .probe = rte_pmd_null_probe,
           .remove = rte_pmd_null_remove,
 };

 RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```

In PMD bus driver representation changed

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

related data structures

```
struct rte_device {
        TAILQ_ENTRY(rte_device) next; /**< Next device */
        const char *name;             /**< Device name */
        const struct rte_driver *driver;/**< Associated driver */
        int numa_node;                /**< NUMA node connection */
        struct rte_devargs *devargs;  /**< Device user arguments */
};

struct rte_driver {
        TAILQ_ENTRY(rte_driver) next;  /**< Next in list. */
        const char *name;                   /**< Driver name. */
        const char *alias;             /**< Driver alias. */
};

struct rte_pci_device {
        TAILQ_ENTRY(rte_pci_device) next;        /**<
        struct rte_device device;                /**
        struct rte_pci_addr addr;                /
        struct rte_pci_id id;
        struct rte_mem_resource mem_resource[PCI

        struct rte_intr_handle intr_handle;
        struct rte_pci_driver *driver;           /
        uint16_t max_vfs;                        /*
        enum rte_kernel_driver kdrv;             /**
        char name[PCI_PRI_STR_SIZE+1];           /**<
};

struct rte_pci_driver {
        TAILQ_ENTRY(rte_pci_driver) next;        /**< Next in list. */
        struct rte_driver driver;                /**< Inherit core driver. */
        struct rte_pci_bus *bus;                 /**< PCI bus reference. */
        pci_probe_t *probe;                      /**< Device Probe function. */
        pci_remove_t *remove;                    /**< Device Remove function. */
        const struct rte_pci_id *id_table;       /**< ID table, NULL terminated. */
        uint32_t drv_flags;                      /**< Flags contolling handling of device. */
};
```

New bus functions for hotplug support

```
struct rte_pci_bus {
        struct rte_bus bus;                  /**< Inherit the generic class */
        struct rte_pci_device_list device_list;  /**< List of PCI devices */
        struct rte_pci_driver_list driver_list;  /**< List of PCI drivers */
};

struct rte_vdev_driver {
        TAILQ_ENTRY(rte_vdev_driver) next; /**< Next in list. */
        struct rte_driver driver;      /**< Inherited general driver. */
        rte_vdev_probe_t *probe;       /**< Virtual device probe function. */
        rte_vdev_remove_t *remove;     /**< Virtual device remove function. */
};

struct rte_vdev_device {
        TAILQ_ENTRY(rte_vdev_device) next;      /**< Next attached vdev */
        struct rte_device device;               /**< Inherit core device */
};

struct rte_eth_dev {
              struct rte_device *devi
              enum rte_eth_dev_state
              …
};

struct rte_bus {
        TAILQ_ENTRY(rte_bus) next;
        const char *name;
        rte_bus_scan_t scan;           /*
        rte_bus_probe_t probe;         /*
        rte_bus_find_device_t                          on the bus */
        rte_bus_plug_t plug                  /**< Probe single device for drivers */
        rte_bus_unplug_t                  ;   /**< Remove single device from driver */
        rte_bus_parse_t            e;        /**< Parse a device name */
        struct rte_b     onf conf;           /**< Bus configuration */
};

struct rte_devargs {
        …
};
```

Updated devargs, removed bus-specific name or address

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

bus code

```
struct rte_pci_bus rte_pci_bus = {
        .bus = {
                .scan = rte_pci_scan,
                .probe = rte_pci_probe,
                .find_device = pci_find
                .plug = pci_plug,
                .unplug = pci_unplug,
                .parse = pci_parse,
        },
        .device_list = TAILQ_HEAD_INITIAL
        .driver_list = TAILQ_HEAD_INITIALIZ
};

RTE_REGISTER_BUS(pci, rte_pci_bus.bus);

struct rte_fslmc_bus rte_fslmc_bus = {
        .bus = {
                .scan = rte_fslmc_scan,
                .probe = rte_fslmc_probe,
                .find_device = rte_fslmc_find_device,
        },
        .device_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.device_list),
        .driver_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.driver_list),
};

RTE_REGISTER_BUS(fslmc, rte_fslmc_bus.bus);
```

To parse string representation of the device

```
static struct rte_bus rte_vdev_bus = {
        .scan = vdev_scan,
        .probe = vdev_probe,
        .find_device = vdev_find_device,
         .plug = vdev_plug,
         .unplug = vdev_unplug,
         .parse = vdev_parse,

RTE_REGISTER_BUS(vdev, rte_vdev_bus);
```

Two new APIs:

- **rte_eal_hotplug_add**
- **rte_eal_hotplug_remove**

16.11- 17.02 – 17.05 – 17.08 – 17.11

eal init flow

```
rte_eal_init
        eal_parse_args
                eal_parse_common_option
                        rte_eal_devargs_add
                                insert(devargs_list, devargs)
        rte_bus_scan
                foreach(bus, rte_bus_list) bus->scan()
        rte_bus_probe
                foreach(bus, rte_bus_list) bus->probe()
                if (vbus)
                        vbus->probe()
```

bus->scan

bus->probe

PCI

vdev

fslmc

PMD code

```
eth_ixgbe_pci_probe
        rte_eth_dev_pci_generic_probe(pci_dev, eth_ixgbe_dev_init)

eth_ixgbe_pci_remove
        rte_eth_dev_pci_generic_remove(pci_dev, eth_ixgbe_dev_uninit)

static struct rte_pci_driver rte_ixgbe_pmd = {
            .id_table = pci_id_ixgbe_map,
            .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
            .probe = eth_ixgbe_pci_probe,
            .remove = eth_ixgbe_pci_remove,
 };

 RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd);


 static struct rte_vdev_driver pmd_null_drv = {
            .probe = rte_pmd_null_probe,
            .remove = rte_pmd_null_remove,
 };

 RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```

16.11- 17.02 – 17.05 – 17.08 – 17.11

related data structures

```
struct rte_device {
        TAILQ_ENTRY(rte_device) next; /**< Next device */
        const char *name;             /**< Device name */
        const struct rte_driver *driver;/**< Associated driver */
        int numa_node;                /**< NUMA node connection */
        struct rte_devargs *devargs;  /**< Device user arguments */
};

struct rte_driver {
        TAILQ_ENTRY(rte_driver) next;  /**< Next in list. */
        const char *name;                   /**< Driver name.
        const char *alias;              /**< Driver alias. */
};

struct rte_pci_device {
        TAILQ_ENTRY(rte_pci_device) next;        /**< Next probed PCI dev
        struct rte_device device;                /**< Inherit core device */
        struct rte_pci_addr addr;                /**< PCI location. */
        struct rte_pci_id id;                    /**< PCI ID. */
        struct rte_mem_resource mem_resource[PCI_MAX_RESOURCE];
                                                 /**< PCI Memory Resource */
        struct rte_intr_handle intr_handle;      /**< Interrupt handle */
        struct rte_pci_driver *driver;           /**< Associated driver */
        uint16_t max_vfs;                        /**< sriov enable if not zero */
        enum rte_kernel_driver kdrv;             /**< Kernel driver passthrough */
        char name[PCI_PRI_STR_SIZE+1];           /**< PCI location (ASCII) */
};

struct rte_pci_driver {
        TAILQ_ENTRY(rte_pci_driver) next;        /**< Next in list. */
        struct rte_driver driver;                /**< Inherit core driver. */
        struct rte_pci_bus *bus;                 /**< PCI bus reference. */
        pci_probe_t *probe;                      /**< Device Probe function. */
        pci_remove_t *remove;                    /**< Device Remove function. */
        const struct rte_pci_id *id_table;       /**< ID table, NULL terminated. */
        uint32_t drv_flags;                      /**< Flags contolling handling of device. */
};
```

NO change (yet)

```
struct rte_pci_bus {
        struct rte_bus bus;                 /**< Inherit the generic class */
        struct rte_pci_device_list device_list;  /**< List of PCI devices */
        struct rte_pci_driver_list driver_list;  /**< List of PCI drivers */
};

struct rte_vdev_driver {
        TAILQ_ENTRY(rte_vdev_driver) next; /**< Next in list. */
        struct rte_driver driver;      /**< Inherited general driver. */
        rte_vdev_probe_t *probe;       /**< Virtual device probe function. */
        rte_vdev_remove_t *remove;     /**< Virtual device remove function. */
};

struct rte_vdev_device {
        TAILQ_ENTRY(rte_vdev_device) next;      /**< Next attached vdev */
        struct rte_device device;               /**< Inherit core device */
};

struct rte_eth_dev {
            struct rte_device *device; /**< Backing device */
            enum rte_eth_dev_state state; /**< Flag indicating the port state */
            …
};

struct rte_bus {
        TAILQ_ENTRY(rte_bus) next;   /**< Next bus object in linked list */
        const char *name;            /**< Name of the bus */
        rte_bus_scan_t scan;         /**< Scan for devices attached to bus */
        rte_bus_probe_t probe;       /**< Probe devices on bus */
        rte_bus_find_device_t find_device; /**< Find a device on the bus */
        rte_bus_plug_t plug;         /**< Probe single device for drivers */
        rte_bus_unplug_t unplug;     /**< Remove single device from driver */
        rte_bus_parse_t parse;       /**< Parse a device name */
        struct rte_bus_conf conf;    /**< Bus configuration */
};
```

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

bus code

```
struct rte_pci_bus rte_pci_bus = {
	.bus = {
		.scan = rte_pci_scan,
		.probe = rte_pci_probe,
		.find_device = pci_find_device,
		.plug = pci_plug,
		.unplug = pci_unplug,
		.parse = pci_parse,
	},
	.device_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.device
	.driver_list = TAILQ_HEAD_INITIALIZER(rte_pci_bus.drive
};

RTE_REGISTER_BUS(pci, rte_pci_bus.bus);

struct rte_fslmc_bus rte_fslmc_bus = {
	.bus = {
		.scan = rte_fslmc_scan,
		.probe = rte_fslmc_probe,
		.find_device = rte_fslmc_find_device,
	},
	.device_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.device_list),
	.driver_list = TAILQ_HEAD_INITIALIZER(rte_fslmc_bus.driver_list),
};

RTE_REGISTER_BUS(fslmc, rte_fslmc_bus.bus);
```

New bus driver

```
static struct rte_bus rte_vdev_bus = {
		.scan = vdev_scan,
		.probe = vdev_probe,
		.find_device = vdev_find_device,
		.plug = vdev_plug,
		.unplug = vdev_unplug,
		.parse = vdev_parse,
};

RTE_REGISTER_BUS(vdev, rte_vdev_bus);

struct rte_dpaa_bus rte_dpaa_bus = {
	.bus = {
		.scan = rte_dpaa_bus_scan,
		.probe = rte_dpaa_bus_probe,
		.find_device = rte_dpaa_find_device,
	},
	.device_list = TAILQ_HEAD_INITIALIZER(rte_dpaa_bus.device_list),
	.driver_list = TAILQ_HEAD_INITIALIZER(rte_dpaa_bus.driver_list),
	.device_count = 0,
};

RTE_REGISTER_BUS(FSL_DPAA_BUS_NAME, rte_dpaa_bus.bus);
```

16.11- 17.02 – 17.05 – 17.08 – 17.11

DPDK

eal init flow

```
rte_eal_init
        eal_parse_args
                eal_parse_common_option
                        rte_eal_devargs_add
                                insert(devargs_list, devargs)
        rte_bus_scan
                foreach(bus, rte_bus_list) bus->scan()
        rte_bus_probe
                foreach(bus, rte_bus_list) bus->probe()
                if (vbus)
                        vbus->probe()
```

bus->scan

bus->probe

PCI

vdev

fslmc

PMD code

```
eth_ixgbe_pci_probe
        rte_eth_dev_pci_generic_probe(pci_dev, eth_ixgbe_dev_init)

eth_ixgbe_pci_remove
        rte_eth_dev_pci_generic_remove(pci_dev, eth_ixgbe_dev_uninit)

static struct rte_pci_driver rte_ixgbe_pmd = {
            .id_table = pci_id_ixgbe_map,
            .drv_flags = RTE_PCI_DRV_NEED_MAPPING | RTE_PCI_DRV_INTR_LSC,
            .probe = eth_ixgbe_pci_probe,
            .remove = eth_ixgbe_pci_remove,
 };

 RTE_PMD_REGISTER_PCI(net_ixgbe, rte_ixgbe_pmd);


 static struct rte_vdev_driver pmd_null_drv = {
            .probe = rte_pmd_null_probe,
            .remove = rte_pmd_null_remove,
 };

 RTE_PMD_REGISTER_VDEV(net_null, pmd_null_drv);
```

TODO – 17.11 ongoing

- Move vdev and PCI into drivers/bus/ folder (patches on patchwork).
  - Looking for volunteers! For drivers/bus/* maintainership.
- Devargs rework separate it from specific bus.
  - whitelist / blacklist must be more generic (not only PCI).
- drivers/bus/net ?

TODO - Future

- Hyper-V VMBUS ?
- Add/Remove notifications for hotplug.
- More explicit and extensible devargs.
- Remove rte_bus reference from devargs.
- Bus documentation ?

Thanks

- Shreyansh Jain
- Jan Blunck
- Gaetan Rivet
- And rest that I missed.

Questions?

Ferruh Yigit

ferruh.yigit@intel.com