Linux Network Drivers

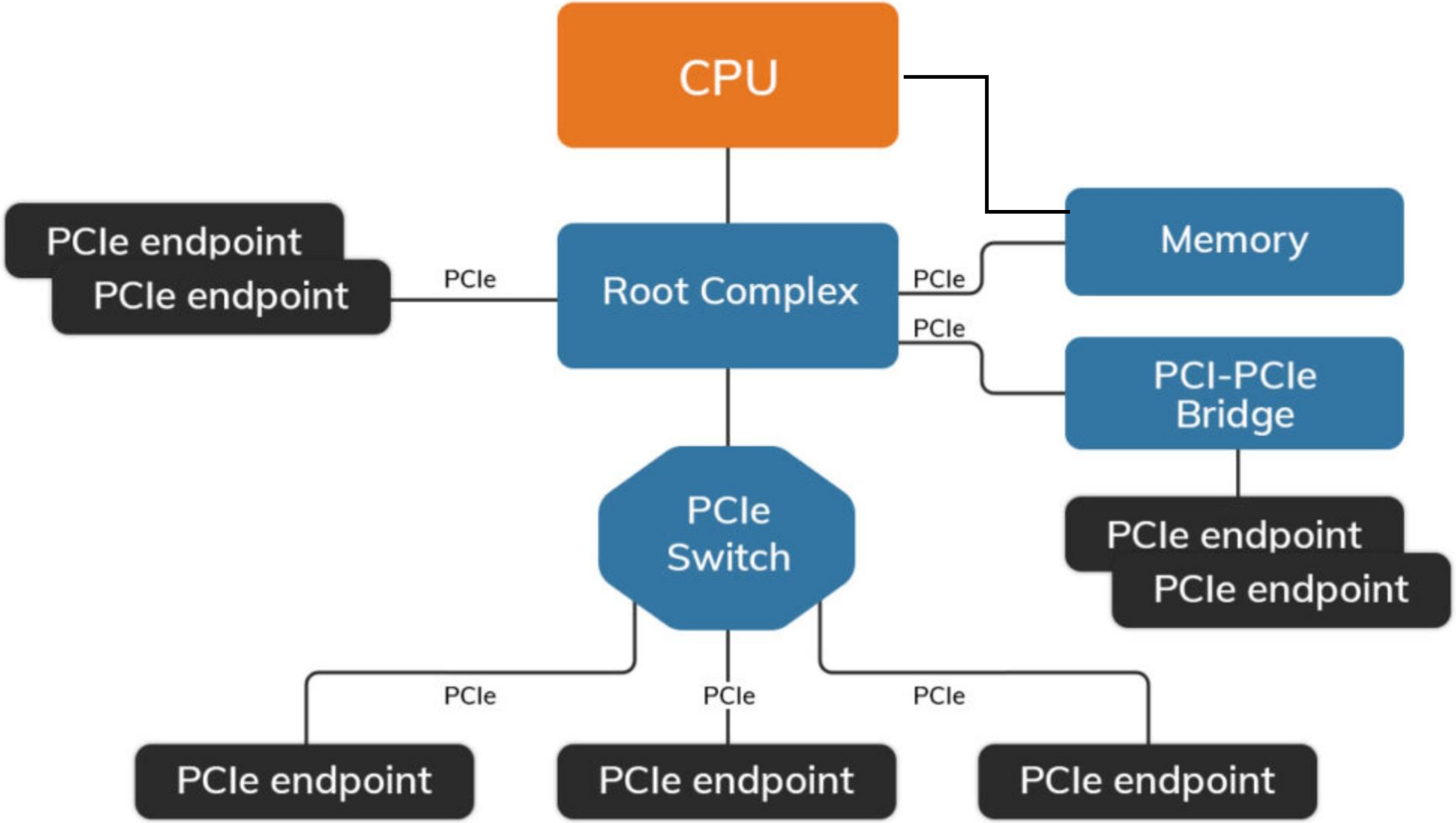
Team Emertxe

EMERTXE

Topics

- PCI Overview
- Network interface Overview
- Linux Network stack and Driver
- Dummy Network Driver code walkthrough

PCI Overview



PCI Device Configuration Space

<table>
<tr><th></th><th colspan="2">0x0</th><th colspan="12"></th><th colspan="2">0xF</th></tr>
<tr><td>0x00</td><td colspan="2">Vendor ID</td><td colspan="2">Device ID</td><td colspan="2">Command Register</td><td colspan="2">Status Register</td><td>Rev ID</td><td colspan="3">Class Code</td><td>Cache Line</td><td>Lat Timer</td><td>Header Type</td><td>BIST</td></tr>
<tr><td>0x10</td><td colspan="4">Base Address 0</td><td colspan="4">Base Address 1</td><td colspan="4">Base Address 2</td><td colspan="4">Base Address 3</td></tr>
<tr><td>0x20</td><td colspan="4">Base Address 4</td><td colspan="4">Base Address 5</td><td colspan="4">CardBus CIS pointer</td><td colspan="2">Subsystem Vendor ID</td><td colspan="2">Subsystem Device ID</td></tr>
<tr><td>0x30</td><td colspan="4">Expansion ROM Base Address</td><td colspan="8">Reserved</td><td>IRQ Line</td><td>IRQ Pin</td><td>Min Gnt</td><td>Max Lat</td></tr>
</table>

ΣMERTXE

PCI
Configuration Space...

- During Bootup

–By BIOS

–By PCI Core (Bootloader or Kernel)

- <linux/pci.h>
- pci_read_config_byte/word/dword(struct pci_dev *dev, int where, u8/16/32 *val);
- pci_write_config_byte/word/dword(struct pci_dev *dev, int where, u8/16/32 *val);

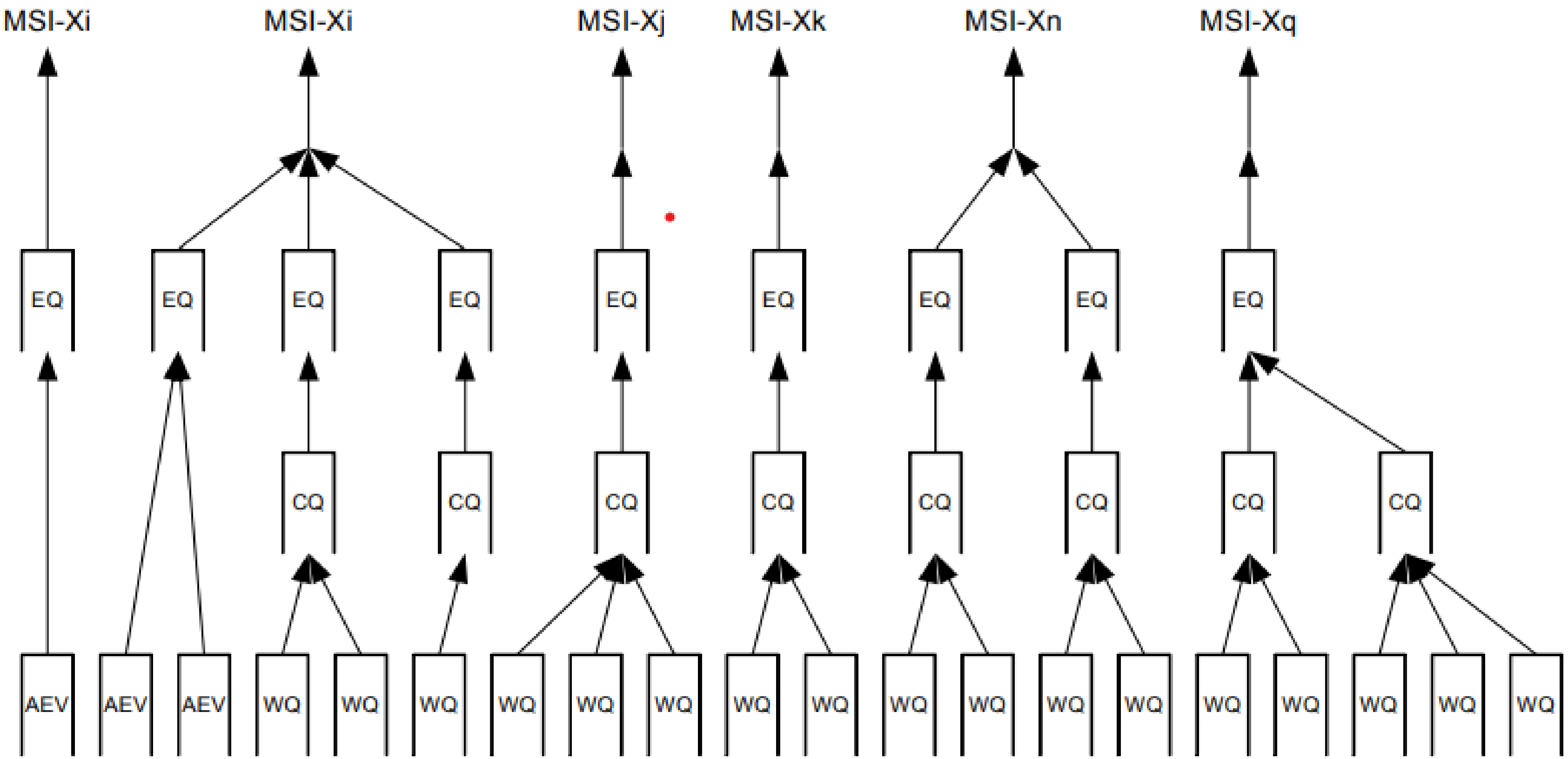
PCI Driver Registration

- int pci_register_driver(struct pci_driver *drv);
- int pci_unregister_driver(struct pci_driver *drv);
- struct pci_driver
  - const char *name
  - const struct pci_dev_id *id_table;
    - PCI_DEVICE(vendor, device);
    - PCI_DEVICE_CLASS(dev_class, dev_class_mask);
  - int (*probe)(pci_dev, id_table);
  - void (*remove)(pci_dev);

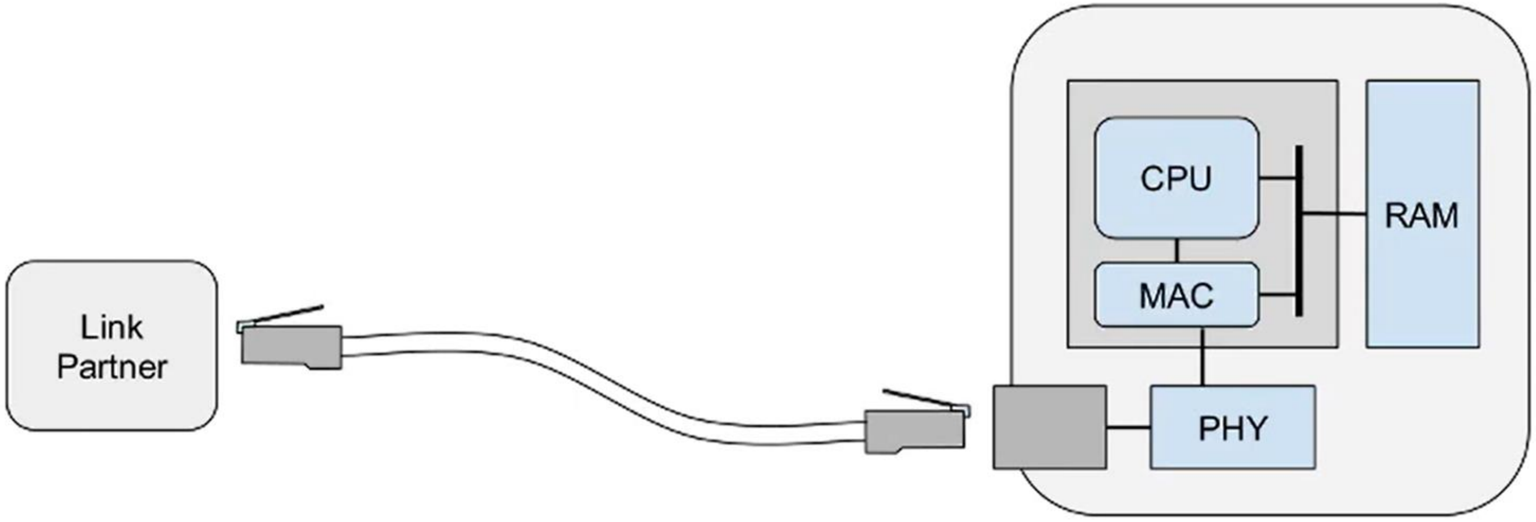
The 'probe' Function

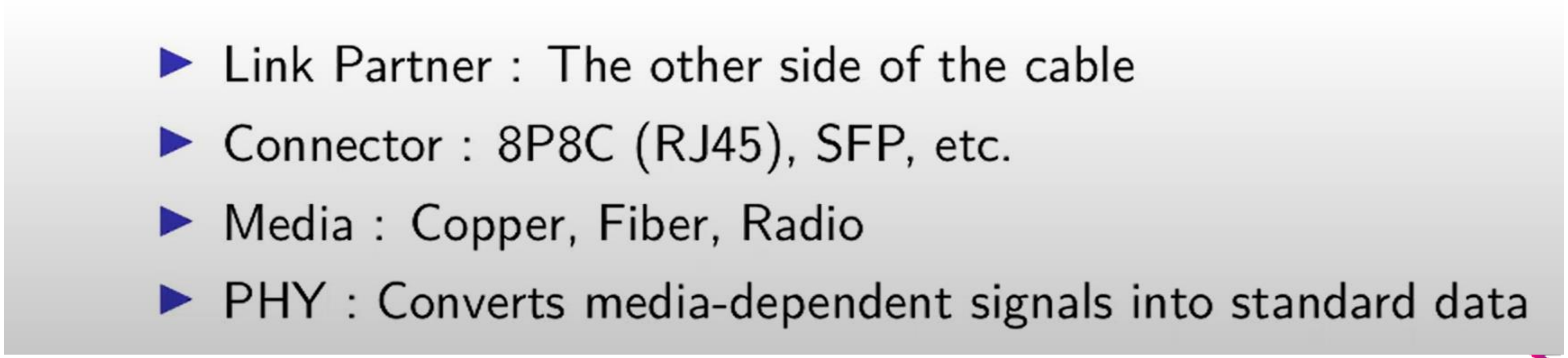
```
int probe(struct pci_dev *d, struct pci_dev_id *id)
{
   /* Initialize the PCI Device */

   ...

   /* Enable the PCI Device */

   pci_enable_device(d);


   ...

   return 0; /* Claimed. Negative for not Claimed */

}
```

PCI MSIX


MSI-Xi
MSI-Xi
MSI-Xj
MSI-Xk
MSI-Xn
MSI-Xq
EQ
EQ
EQ
EQ
EQ
EQ
EQ
EQ
EQ
CQ
CQ
CQ
CQ
CQ
CQ
CQ
CQ
AEV
AEV
AEV
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ
WQ

Network interface Overview

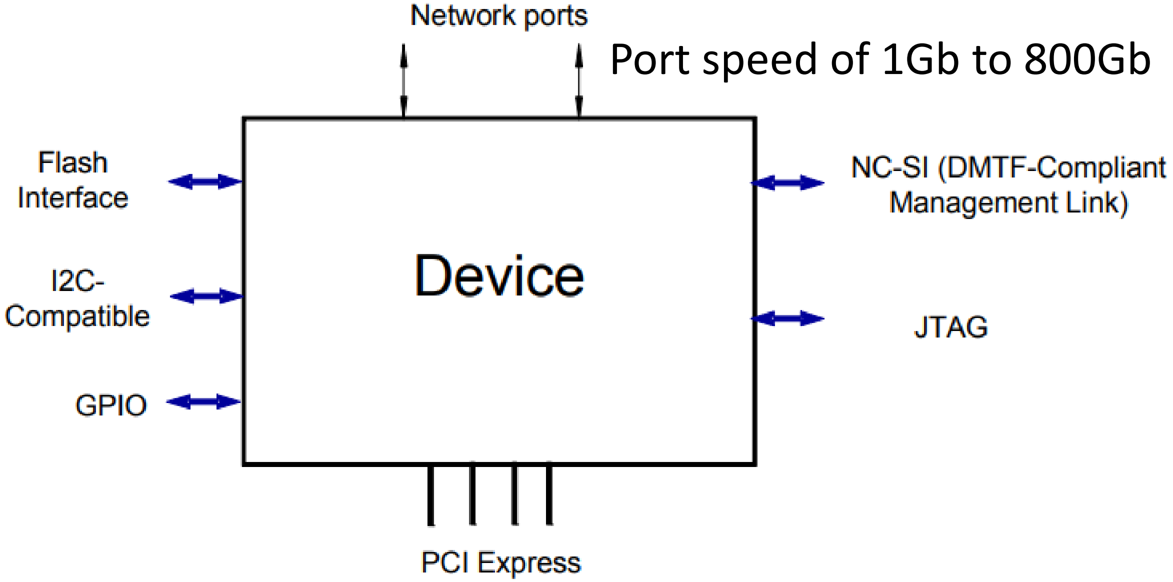




- ▶ Link Partner : The other side of the cable
- ▶ Connector : 8P8C (RJ45), SFP, etc.
- ▶ Media : Copper, Fiber, Radio
- ▶ PHY : Converts media-dependent signals into standard data

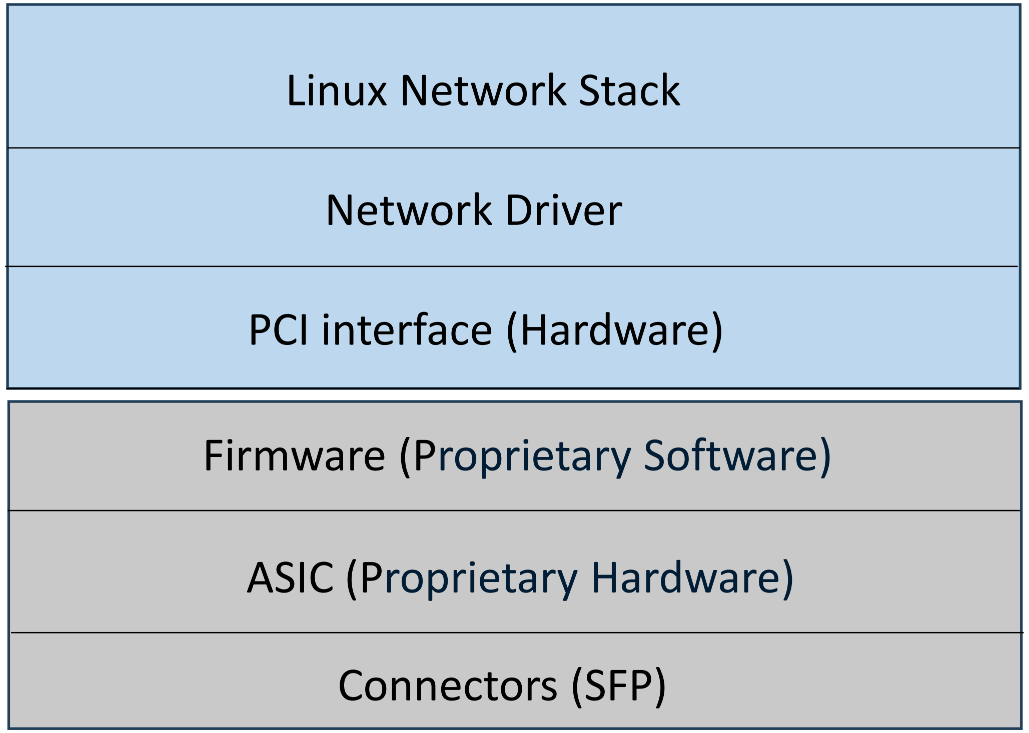
Network interface Overview

Figure 1: System Interface



Port speed of 1Gb to 800Gb

Network interface Overview



Linux Network Stack

Network Driver

PCI interface (Hardware)

Firmware (Proprietary Software)

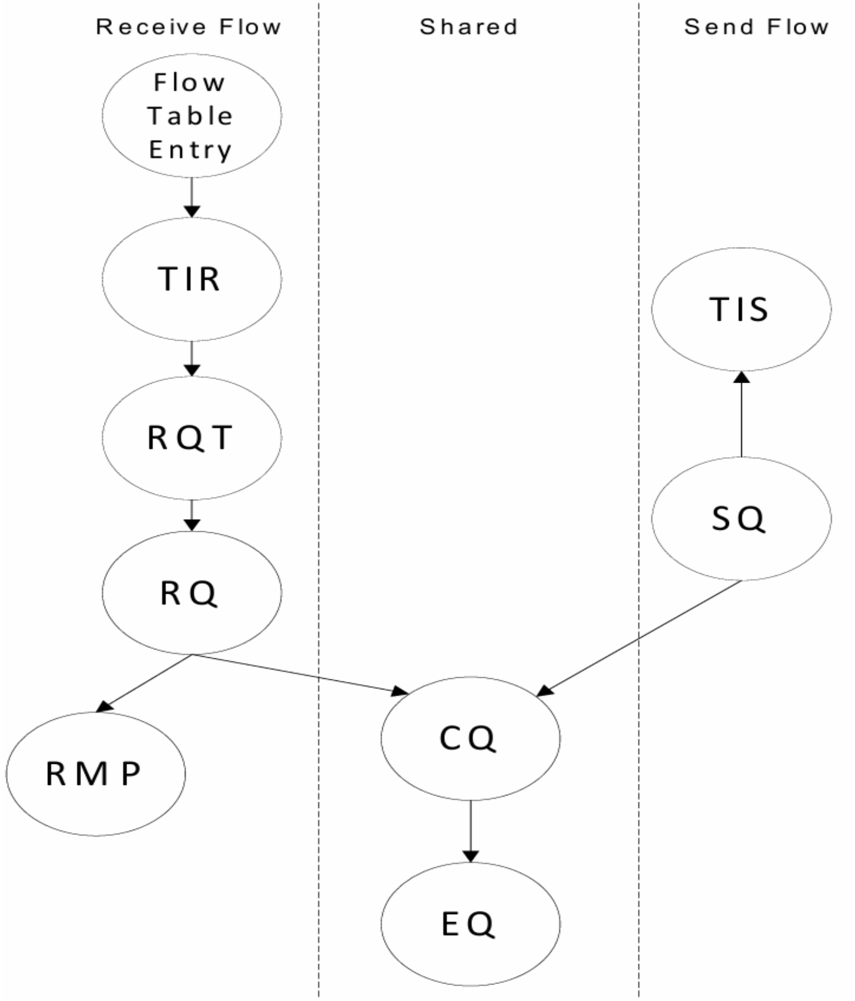
ASIC (Proprietary Hardware)

Connectors (SFP)

Network interface Overview

-Firmware

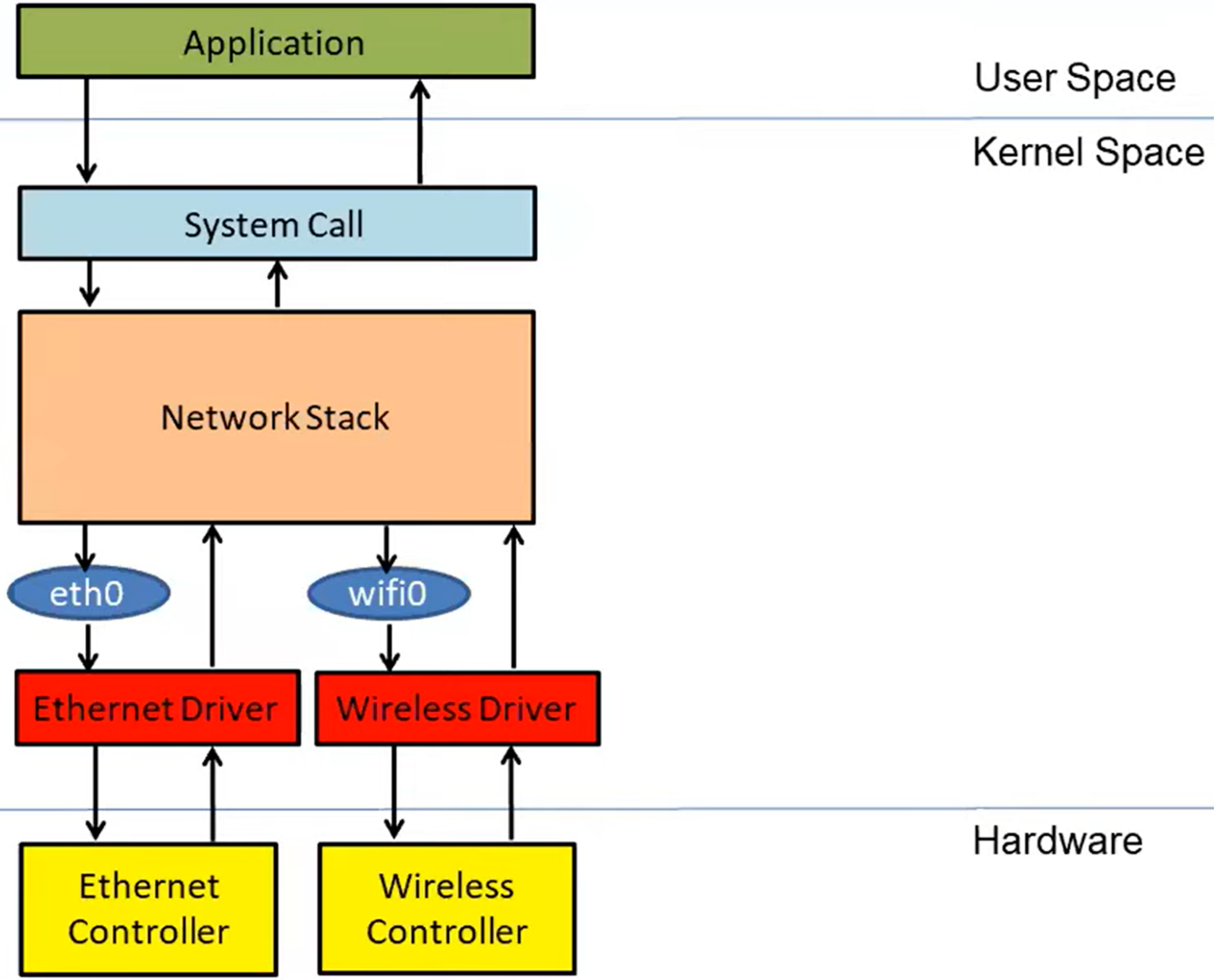
Figure 3: Networking Transport Objects



Network interface Overview

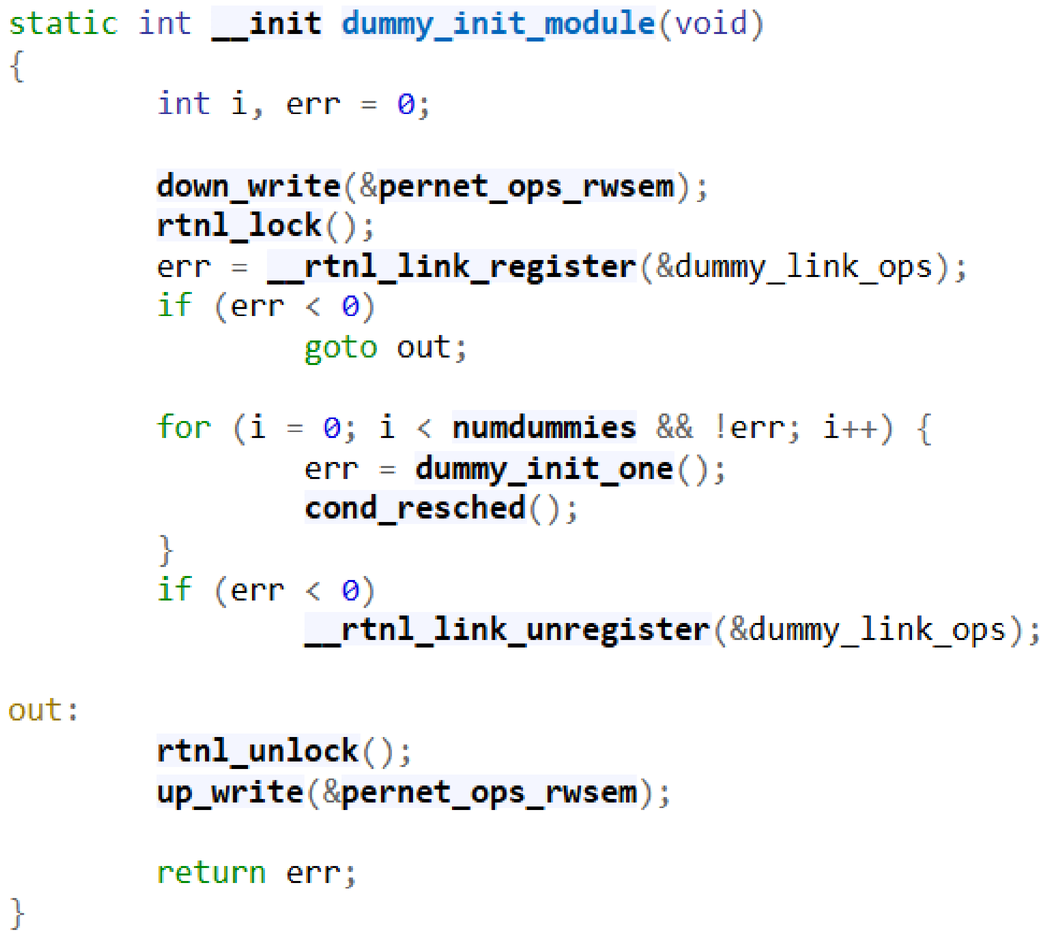
- PCI : Probe, BAR, MSIX
- IEEE 803 standards
- Linux kernel concept: IRQ, threads, memory, etc.
- Linux network stack: sk_buff, net_device_ops, ethtool
- Tools: RSS settings, Wireshark, iPerf, etc.

Network Interface in Network Stack



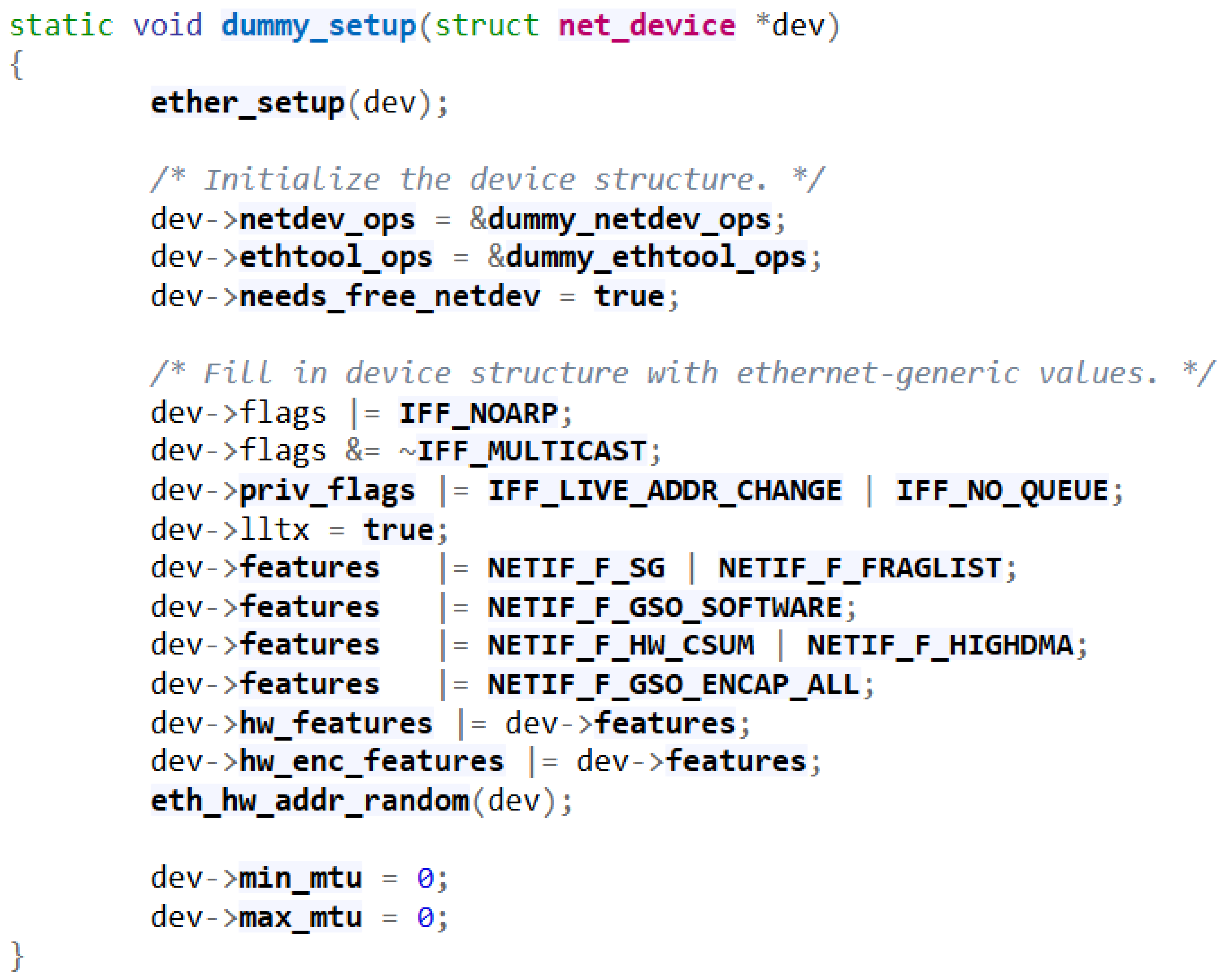
Linux Network Driver

Driver registration



```
static int __init dummy_init_module(void)
{
        int i, err = 0;

        down_write(&pernet_ops_rwsem);
        rtnl_lock();
        err = __rtnl_link_register(&dummy_link_ops);
        if (err < 0)
                goto out;

        for (i = 0; i < numdummies && !err; i++) {
                err = dummy_init_one();
                cond_resched();
        }
        if (err < 0)
                __rtnl_link_unregister(&dummy_link_ops);

out:
        rtnl_unlock();
        up_write(&pernet_ops_rwsem);

        return err;
}
```

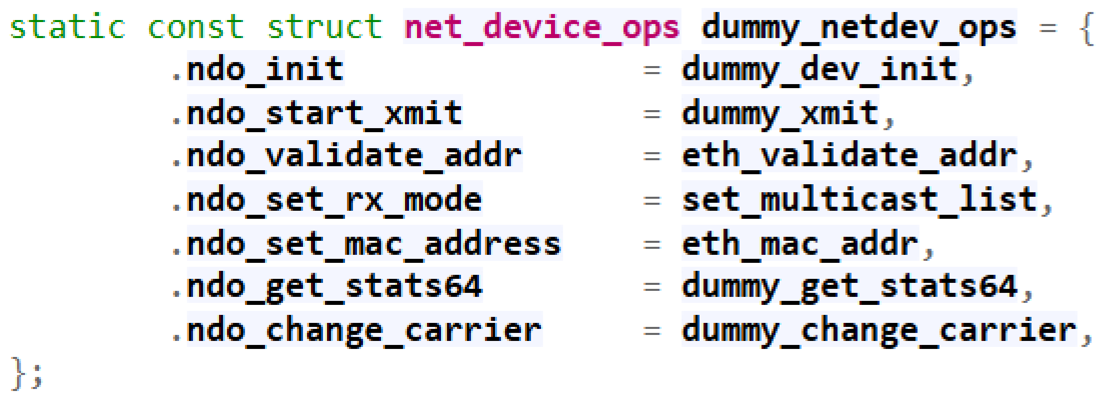
Linux Network Driver



```
static void dummy_setup(struct net_device *dev)
{
        ether_setup(dev);

        /* Initialize the device structure. */
        dev->netdev_ops = &dummy_netdev_ops;
        dev->ethtool_ops = &dummy_ethtool_ops;
        dev->needs_free_netdev = true;

        /* Fill in device structure with ethernet-generic values. */
        dev->flags |= IFF_NOARP;
        dev->flags &= ~IFF_MULTICAST;
        dev->priv_flags |= IFF_LIVE_ADDR_CHANGE | IFF_NO_QUEUE;
        dev->lltx = true;
        dev->features   |= NETIF_F_SG | NETIF_F_FRAGLIST;
        dev->features   |= NETIF_F_GSO_SOFTWARE;
        dev->features   |= NETIF_F_HW_CSUM | NETIF_F_HIGHDMA;
        dev->features   |= NETIF_F_GSO_ENCAP_ALL;
        dev->hw_features |= dev->features;
        dev->hw_enc_features |= dev->features;
        eth_hw_addr_random(dev);

        dev->min_mtu = 0;
        dev->max_mtu = 0;
}
```

Linux Network Driver

Device registration with network stack



```
static const struct net_device_ops dummy_netdev_ops = {
        .ndo_init               = dummy_dev_init,
        .ndo_start_xmit         = dummy_xmit,
        .ndo_validate_addr      = eth_validate_addr,
        .ndo_set_rx_mode        = set_multicast_list,
        .ndo_set_mac_address    = eth_mac_addr,
        .ndo_get_stats64        = dummy_get_stats64,
        .ndo_change_carrier     = dummy_change_carrier,
};
```