RESTRUCTURING DPDK DEVICE-DRIVER FRAMEWORK

Expanding DPDK to non-PCI, non-virtual devices

SHREYANSH JAIN, HEMANT AGRAWAL

NXP

21/OCT/2016

PUBLIC USE

SECURE CONNECTIONS
FOR A SMARTER WORLD

About Me...

- An engineer with NXP's Digital Networking Software team
  - Leveraging NXP's hardware accelerators using Open-source datapath frameworks: DPDK, ODP
  - New to DPDK community (May'16)
- Reach out @
  - shreyansh.jain@nxp.com
  - IRC: #DPDK (nick: shreyansh.)

Agenda: Next ~30 minutes

- Introducing NXP SoC
  - NXP DPAA: Datapath Acceleration Architecture
  - SoC and Standardization - incongruous
- Integrating NXP PMD with DPDK: Stumbling block(s)
  - Solution(s) and what needs to be done for it
  - Allowing Custom Scanning for devices
  - Or, introducing a new Device Model: Bus ⬄ Device ⬄ Driver
- Summarizing the status
- Questions/Comments

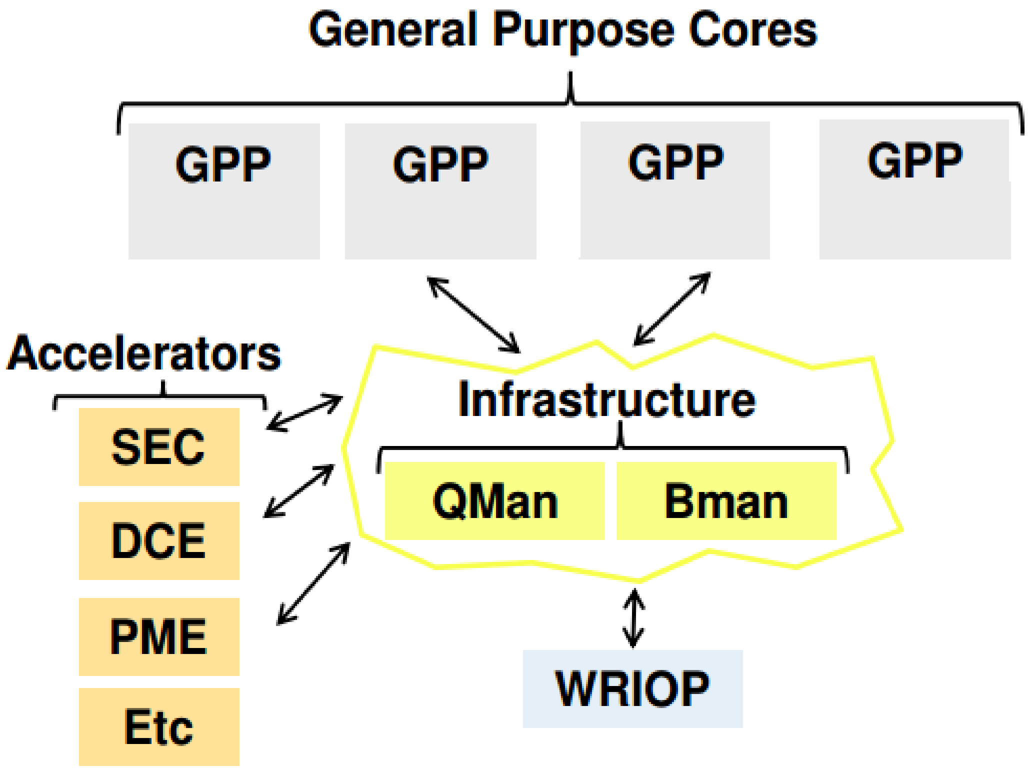
Agenda

- Introducing NXP SoC
  - NXP DPAA: Datapath Acceleration Architecture
  - SoC and Standardization - incongruous
- Integrating NXP PMD with DPDK: Stumbling block(s)
  - Solution(s) and what needs to be done for it
  - Allowing Custom Scanning for devices
  - Or, introducing a new Device Model: Bus ⇔ Device ⇔ Driver
- Summarizing the status
- Questions/Comments

DPDK on NXP SoCs

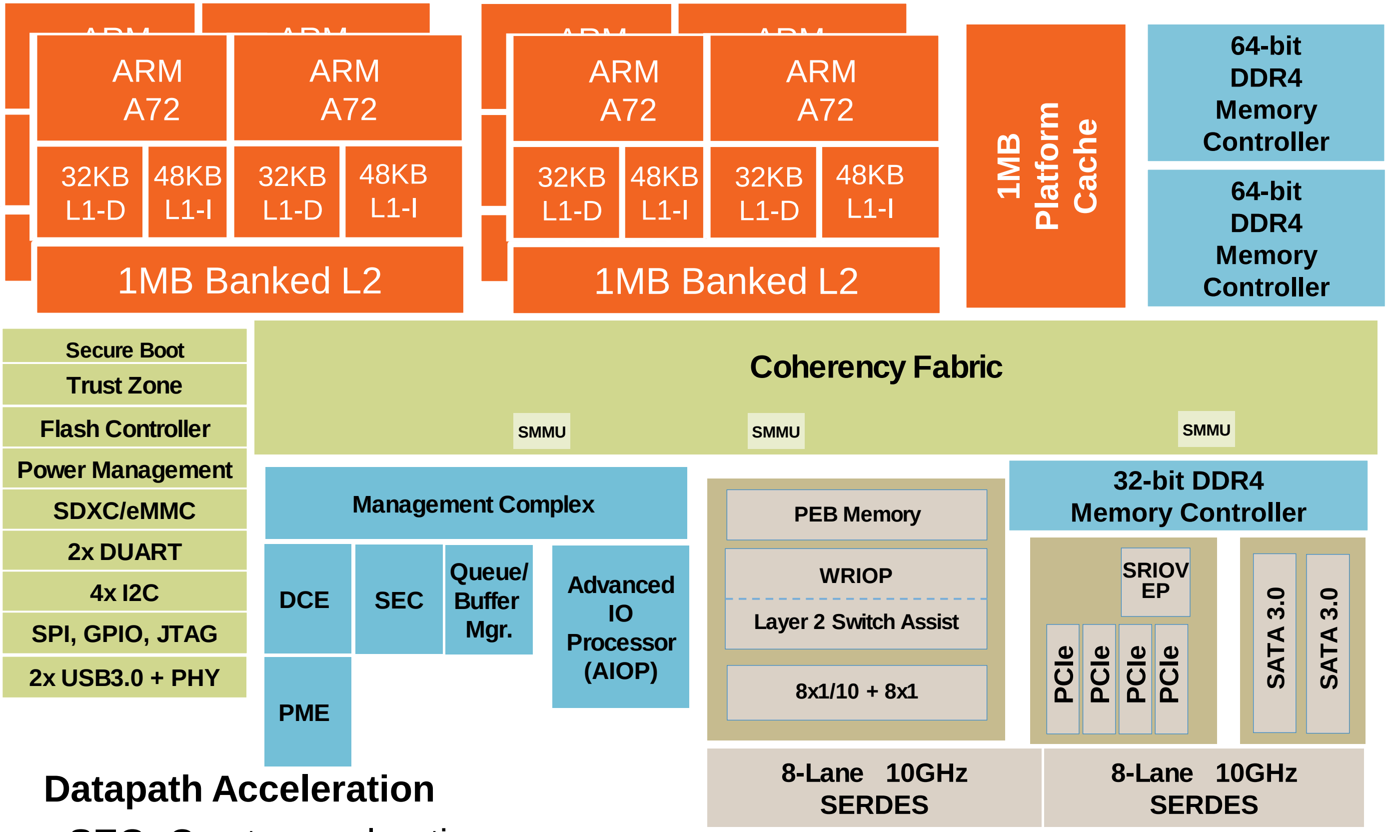
Main Goal is to add NXP SoC support in DPDK

- DPDK 16.07 supports NXP platform configuration (without NXP PMDs)
  - `defconfig_arm64-dpaa2-linuxapp-gcc`
- NXP Networking SoC:
  - Have in-built MAC and they are non-PCI based
  - Have in-built accelerators to support packet processing
    - BMAN - Packet buffer to be allocated & managed by HW
    - QMAN - Packet Queues mapped to hardware queues
    - CAAM – Crypto accelerator offload



DPAA2 Hardware

NXP DPAA2 Architecture (1/3)



Datapath Acceleration

- **SEC**- Crypto acceleration
- **DCE** - Data Compression Engine
- **PME** – Pattern Matching Engine
- **L2 Switching** -- via Datapath Acceleration Hardware
- **Management Complex** – Abstraction of configuration of HW

General Purpose Processing

- 8x 64-bit ARMv8 A72 CPUs up to 2.0GHz
- 1MB L2 cache in each 2xA72 core cluster
- HW L1 & L2 Prefetch Engines
- Neon SIMD in all CPUs
- 1MB L3 platform cache
- 2x64b DDR4 up to 2.133GT/s

Accelerated Packet Processing

- 20Gbps SEC-Crypto acceleration
- 10Gbps Pattern Match/RegEx
- 20Gbps Data Compression Engine

High Speed IO

- Supports1x8, 4x4, 4x2, 4x1 PCIe Gen3 controllers
  - SR-IOV, End Point, Root Complex
- 2 x SATA 3.0, 2 x USB 3.0 with PHY

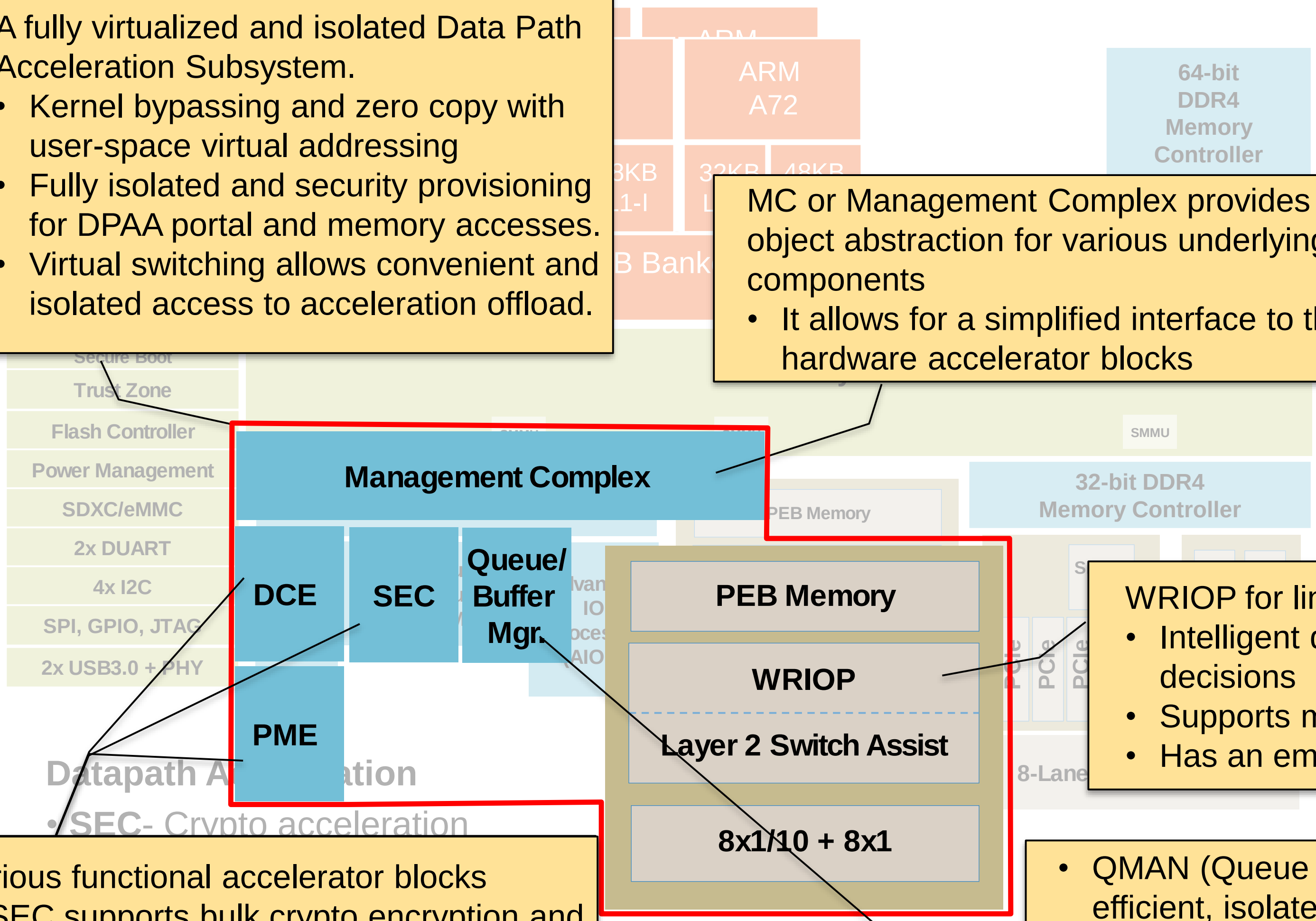
Network IO

- Wire Rate IO Processor:
  - 8x1/10GbE + 8x1G
  - XAUI/XFI/KR and SGMII/QSGMII
  - MACSec on up to 4x 1/10GbE
  - Layer 2 Switch Assist

NXP DPAA2 Architecture (2/3)

A fully virtualized and isolated Data Path Acceleration Subsystem.
- Kernel bypassing and zero copy with user-space virtual addressing
- Fully isolated and security provisioning for DPAA portal and memory accesses.
- Virtual switching allows convenient and isolated access to acceleration offload.

MC or Management Complex provides object abstraction for various underlying components
- It allows for a simplified interface to the hardware accelerator blocks



WRIOP for line rate Networking
- Intelligent distribution, queuing and drop decisions
- Supports multiple Interface profiles
- Has an embedded L2 switch

Various functional accelerator blocks
- SEC supports bulk crypto encryption and various protocol offload
- PME or Pattern Match Engine
- DCE or Data Compression Engine

- QMAN (Queue Manager) provides efficient, isolated, high-bandwidth packet queues
- BMAN or Buffer Manager allows packets to be stored on DDR, allowing pool segregation and lock-free access.

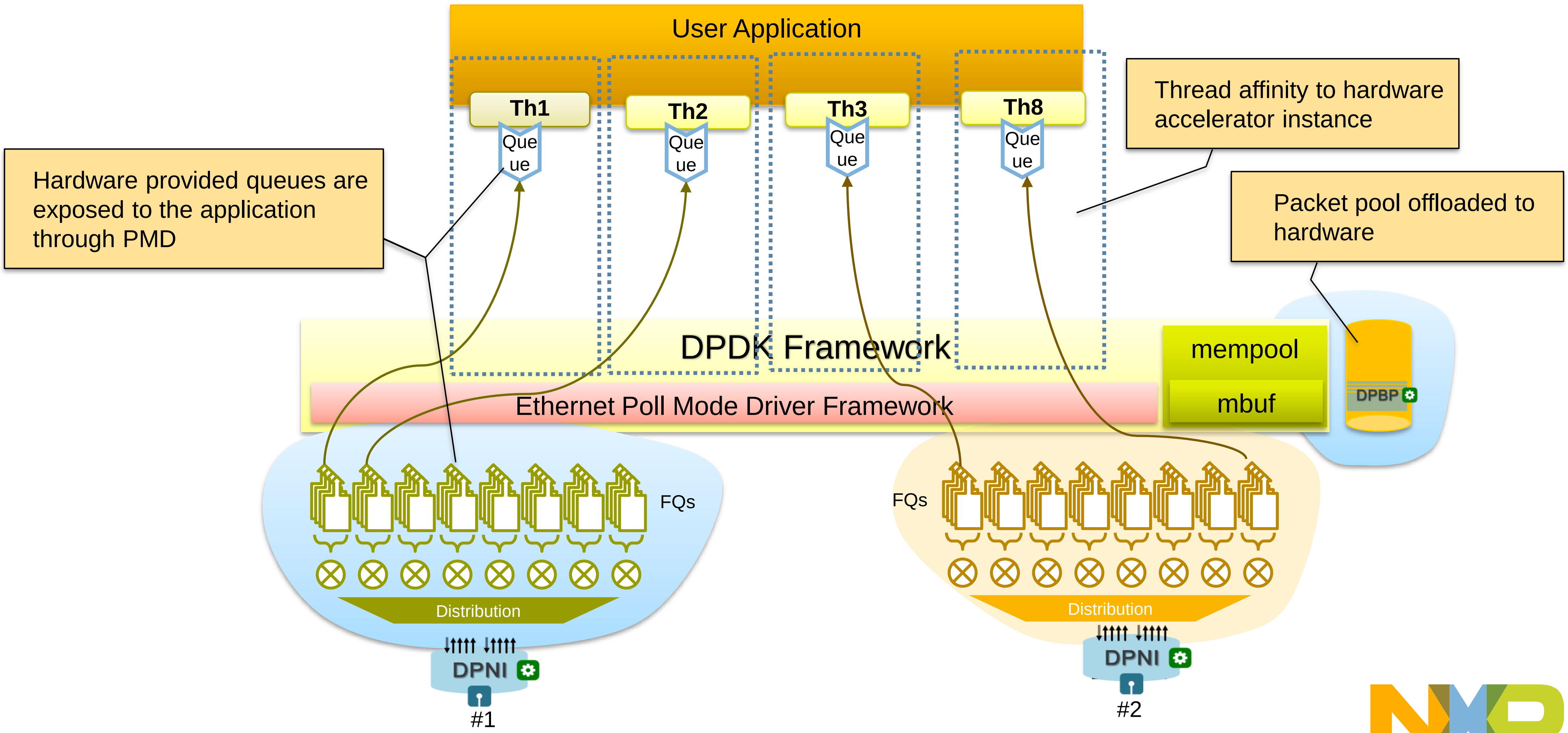
Datapath A... ation
- SEC- Crypto acceleration
...eleration Hardware
...tion of configuration of HW

General Purpose Processing
- 8x 64-bit ARMv8 A72 CPUs up to 2.0GHz
- 1MB L2 cache in each 2xA72 core cluster
- HW L1 & L2 Prefetch Engines
- Neon SIMD in all CPUs
- 1MB L3 platform cache
- 2x64b DDR4 up to 2.133GT/s

Accelerated Packet Processing
- 20Gbps SEC-Crypto acceleration
- 10Gbps Pattern Match/RegEx
- 20Gbps Data Compression Engine

High Speed IO
..., 4x1 PCIe Gen3 controllers
...oot Complex
... 3.0 with PHY
...r:
- 8x1/10GbE + 8x1G
...and SGMII/QSGMII
... to 4x 1/10GbE
... Assist

NXP DPAA2 Architecture (3/3)



SoCs are not necessarily standardized

A typical networking SoC contains one or more MAC within the chip

- Different ways to connect the peripherals (or MAC):
  - Platform Bus e.g. NXP DPAA (LS1043)
  - PCI Bus e.g. Cavium ThunderX
  - Or any other proprietary bus e.g. NXP DPAA2 (fsl-mc bus for LS2088)
- SoCs do not have an standard definition of bus like PCI
  - Even assuming a platform bus is not right
  - Non-standardized way of realizing the devices in user-space (or kernel)

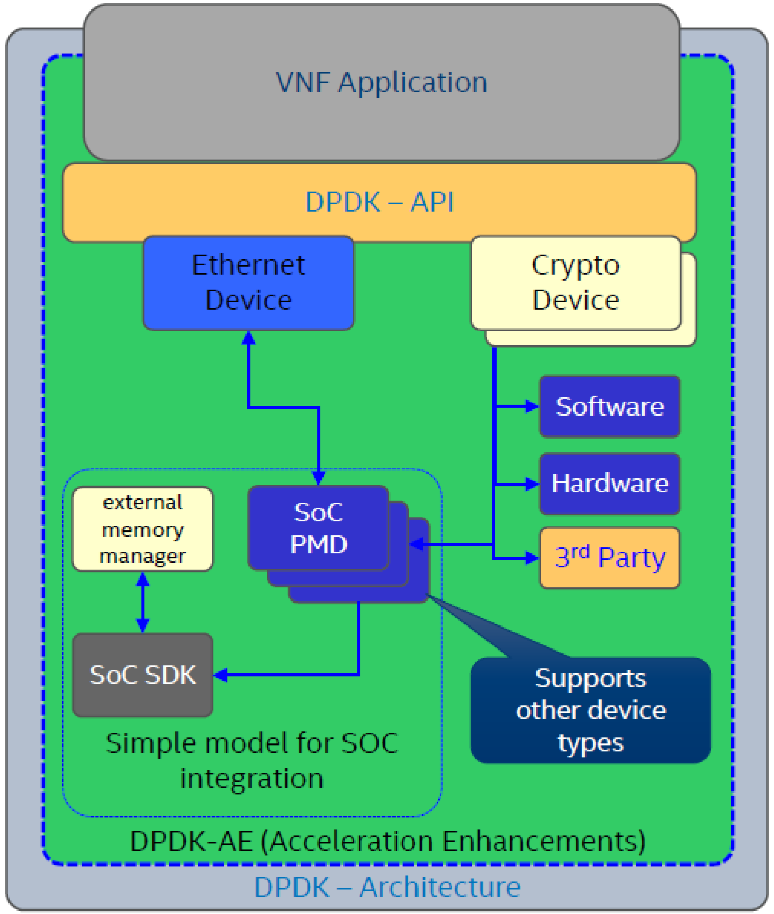
Agenda

- Introducing NXP SoC
  - NXP DPAA: Datapath Acceleration Architecture
  - SoC and Standardization - incongruous
- Integrating NXP PMD with DPDK: Stumbling block(s)
  - Solution(s) and what needs to be done for it
  - Allowing Custom Scanning for devices
  - Or, introducing a new Device Model: Bus ⇔ Device ⇔ Driver
- Summarizing the status
- Questions/Comments

NXP PMD over DPDK: Integration stumbling block

- Inherently a PCI inclined architecture
  - DPDK was designed around PCI devices; there are traces of this across framework
    - `rte_eth_dev` has `rte_pci_device` as a member
    - `eth_driver` has `rte_pci_driver` as a member
      - All Ethernet devices are not PCI
      - Adding support for more bus type possible, but breaks the ABI everytime
  - EAL initialization scans the PCI bus (and VDEV) only
    - Assumes that all devices are discoverable from `sysfs`
    - It is possible to include more ‘standard’ scan functions (Platform, AMBA…)
    - But, ideal would be to have a pluggable model – let Drivers (or bus) perform the scan

There have been proposals to fix this… (1/2)



SoC PMD: Poll Mode driver model for SoC devices

Provides a clean integration of SoC via a PMD in DPDK

- Hardware abstraction in DPDK is at the PMD layer
- DPDK-API: A generic API extended to support SoCs
  - DPDK provides a two layer device model to support many devices at the same time/binary, which can include SoC devices
  - Need to enhance DPDK with some SoC specific needs or features to support SoC hardware
    - Non-PCI configuration
    - External memory manager(s) (for hardware based memory)
    - Event based programming model
- SoC-PMD: Poll Mode Driver model for SoC
  - Allows SoC SDK's to remain private
- Supports ARM and MIPS DPDK ports to utilize these SoC designs

Source: DPDK SF Summit 2015: **"Future Enhancements to DPDK Framework"** by Keith Wiles, Principal Engineer, Intel Corporation

There have been proposals to fix this… (2/2)

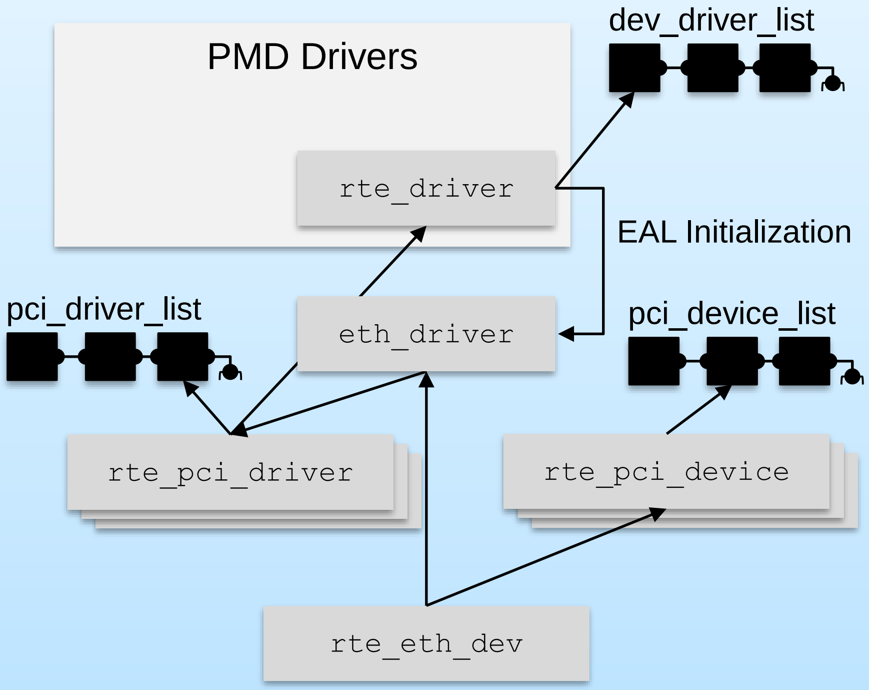
- First series of SoC related improvement sent on ML in Jan'16 [1]
  - Introduces rte_soc_driver, rte_soc_device (and other internal structures)
    - SoC registration and de-registration methods and their invocation from rte_eal_init()
    - Maintaining new linked-lists for SoC devices/drivers (soc_driver_list, soc_device_list)
    - Scanning of SoC devices from platform bus only
  - Subsequent updates allowed for 'default' scan and match [2]
    - PMDs can implement their own scan which is hooked on from EAL initialization
    - Default implementation from first series continued as helpers
  - Overall model allows for a new type of device parallel to PCI
    - SoC or non-PCI?
- Next Step: There is a need for a better model
  - One that is agnostic to such 'device type' changes in long run

V4 of patch-set is under review

[1] http://dpdk.org/ml/archives/dev/2016-January/030915.html
[2] http://dpdk.org/ml/archives/dev/2016-October/048949.html

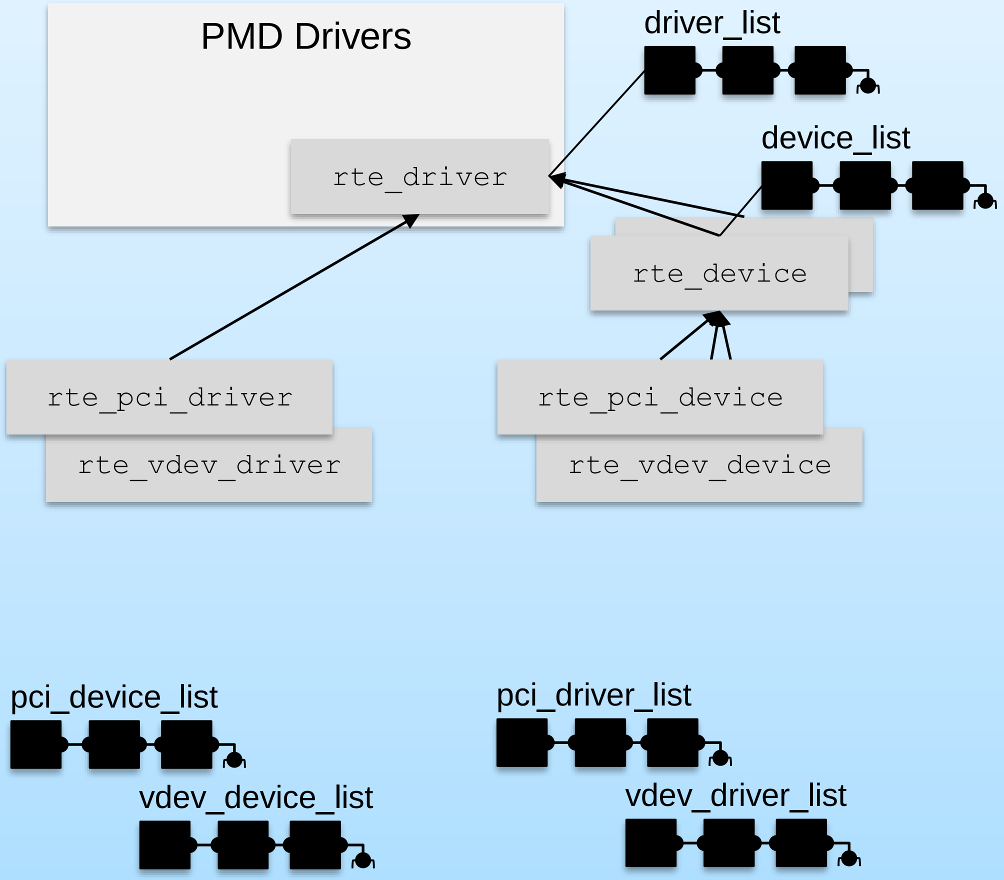
Remodeling Device-Driver Relationship (1)

Existing structure



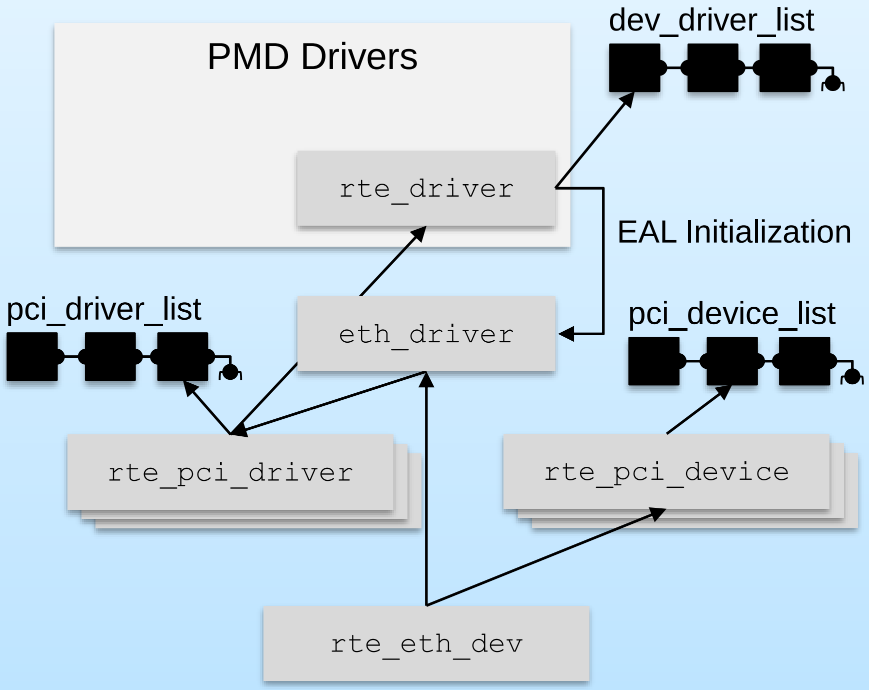
- Virtual devices are also represented by a type of rte_driver (PMD_VDEV)
- No space for non-PCI/non-virtual devices

Recently changed (Sep’16)



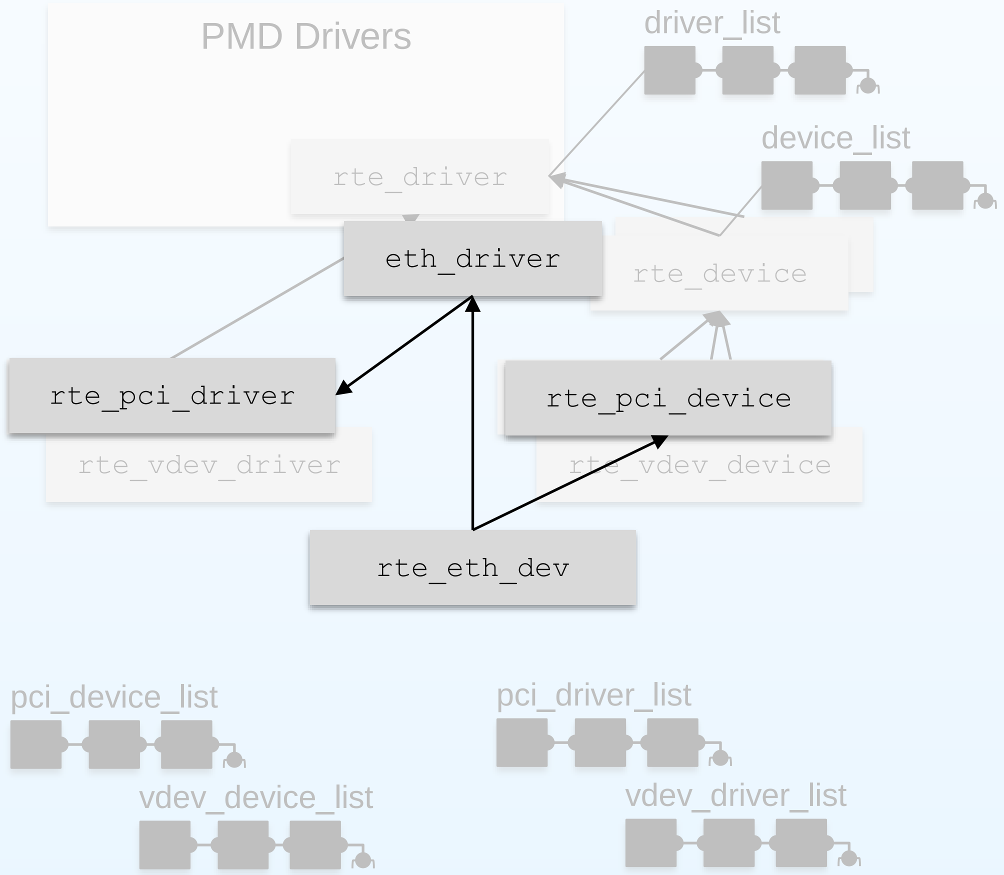
Remodeling Device-Driver Relationship (2)

Existing structure



- Virtual devices are also represented by a type of rte_driver (PMD_VDEV) and treated as PCI devices
- No space for non-PCI/non-virtual devices

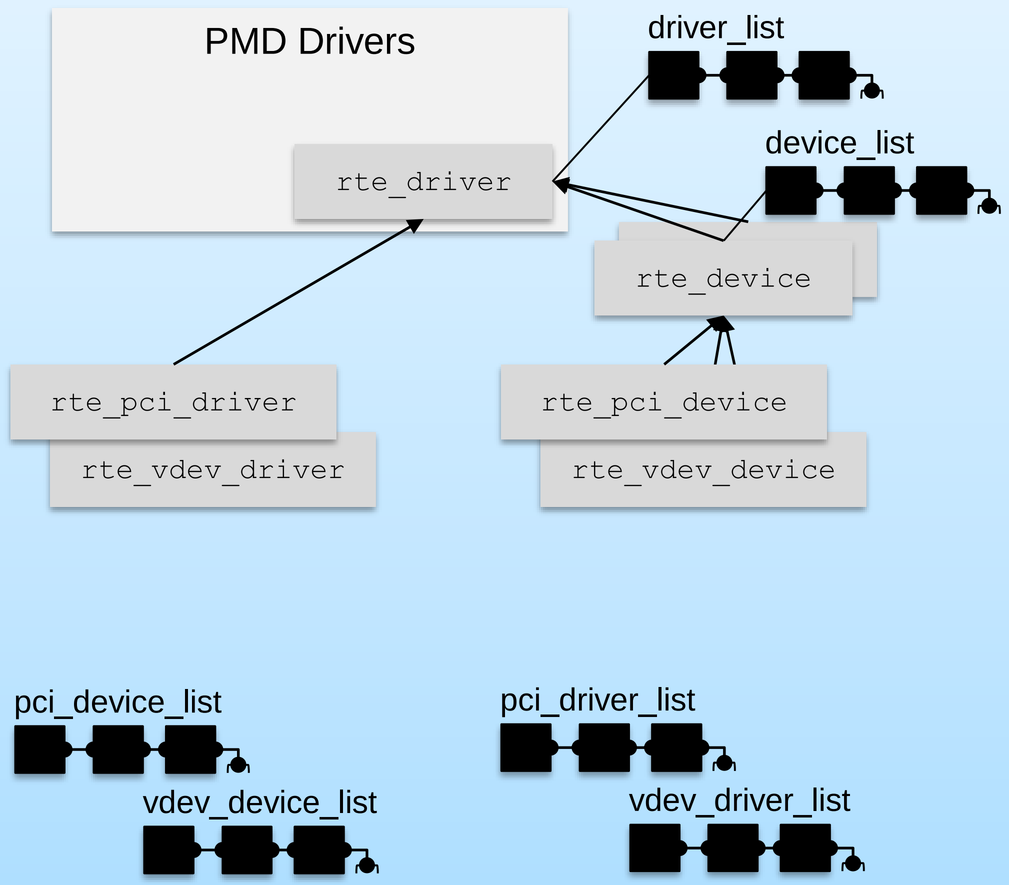
Ethernet devices are still PCI…



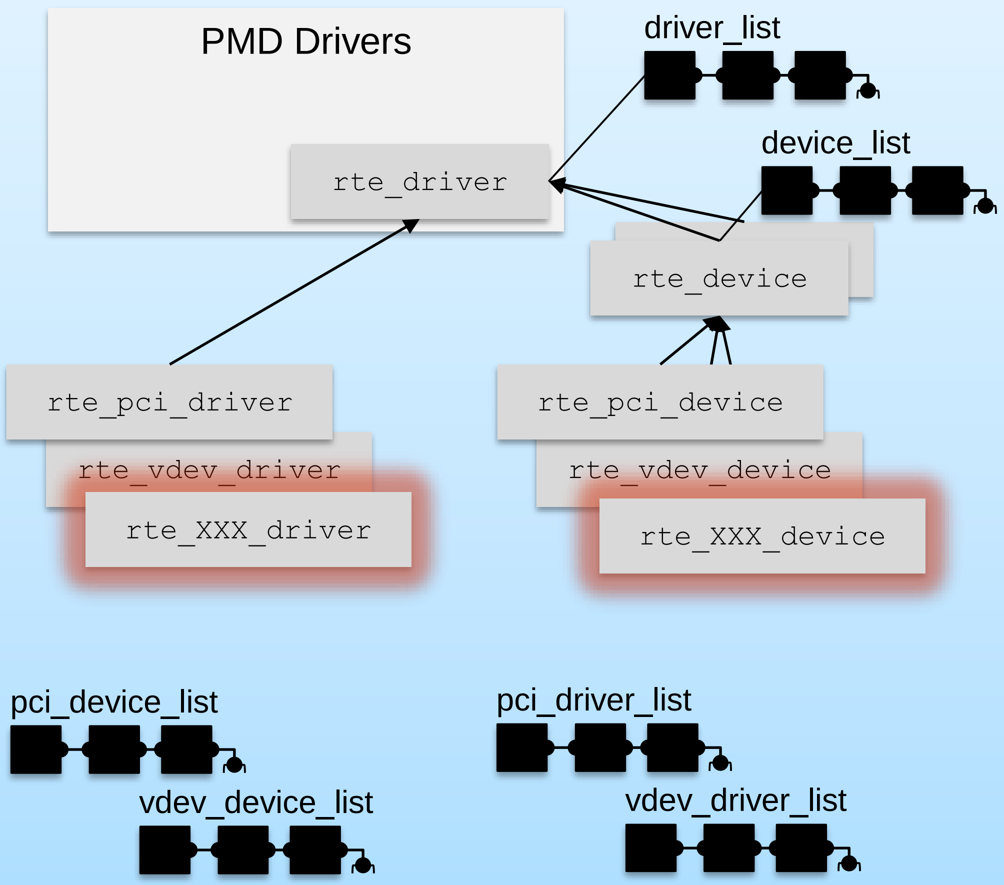
Similar is the case for Crypto devices/driver

Remodeling Device-Driver Relationship (3)

Recently changed (Sep'16)



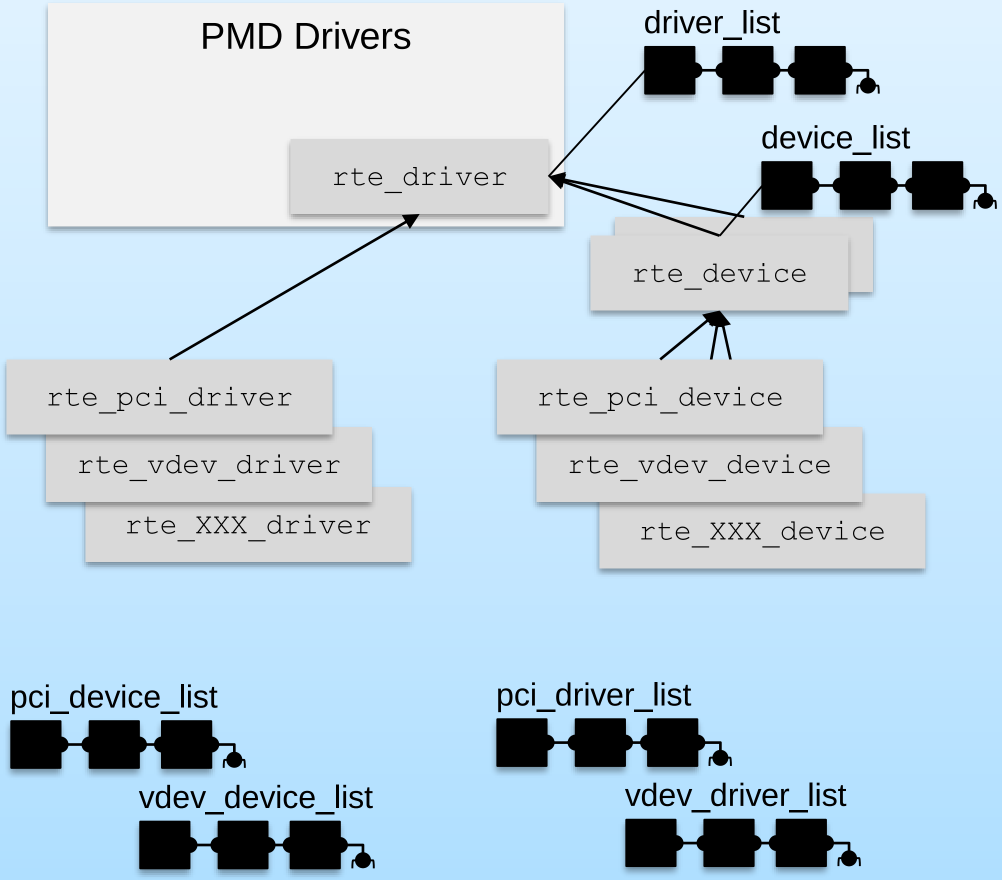
SoC Patch-set introduces this model



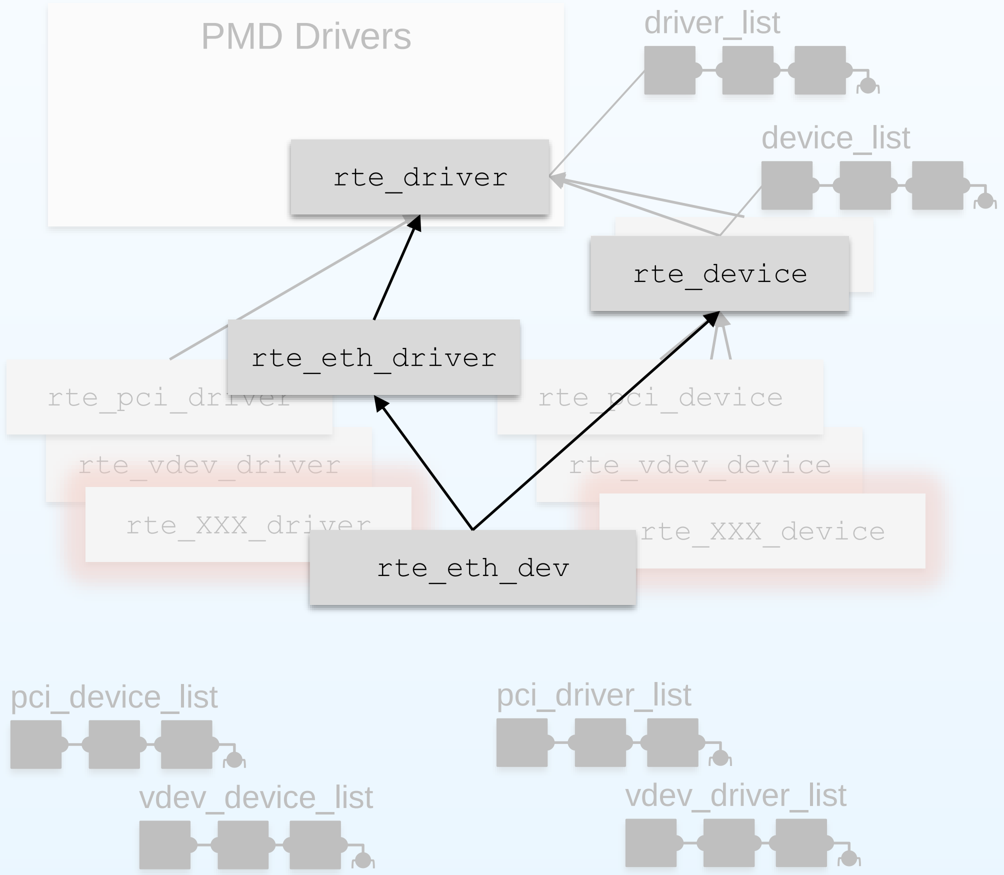
A unified list for each device/driver type?

Remodeling Device-Driver Relationship (4)

Recently changed (Sep’16)



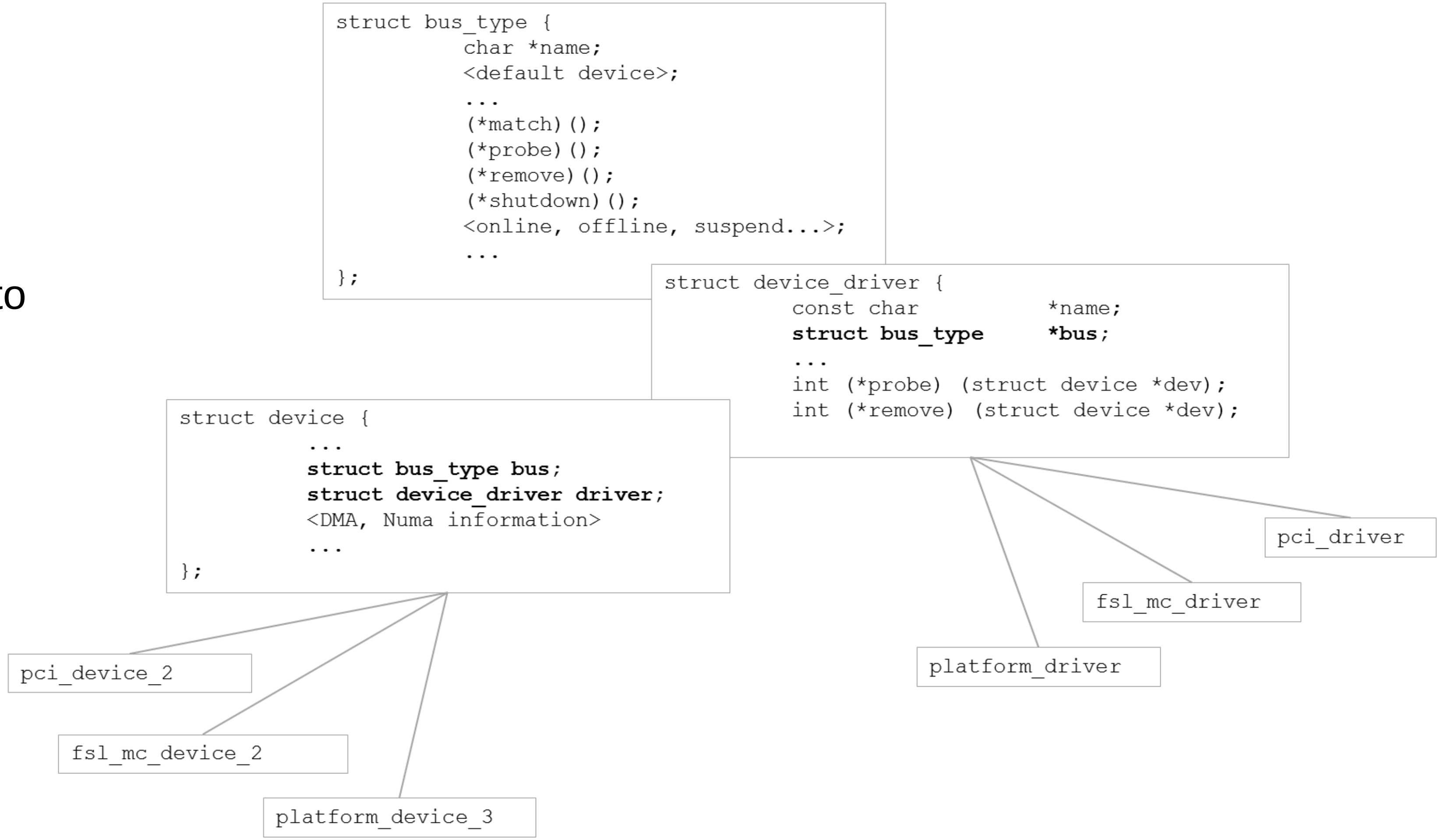
Step 2: Ethernet device bus agnostic



How to link back to exact driver from ethdevice?

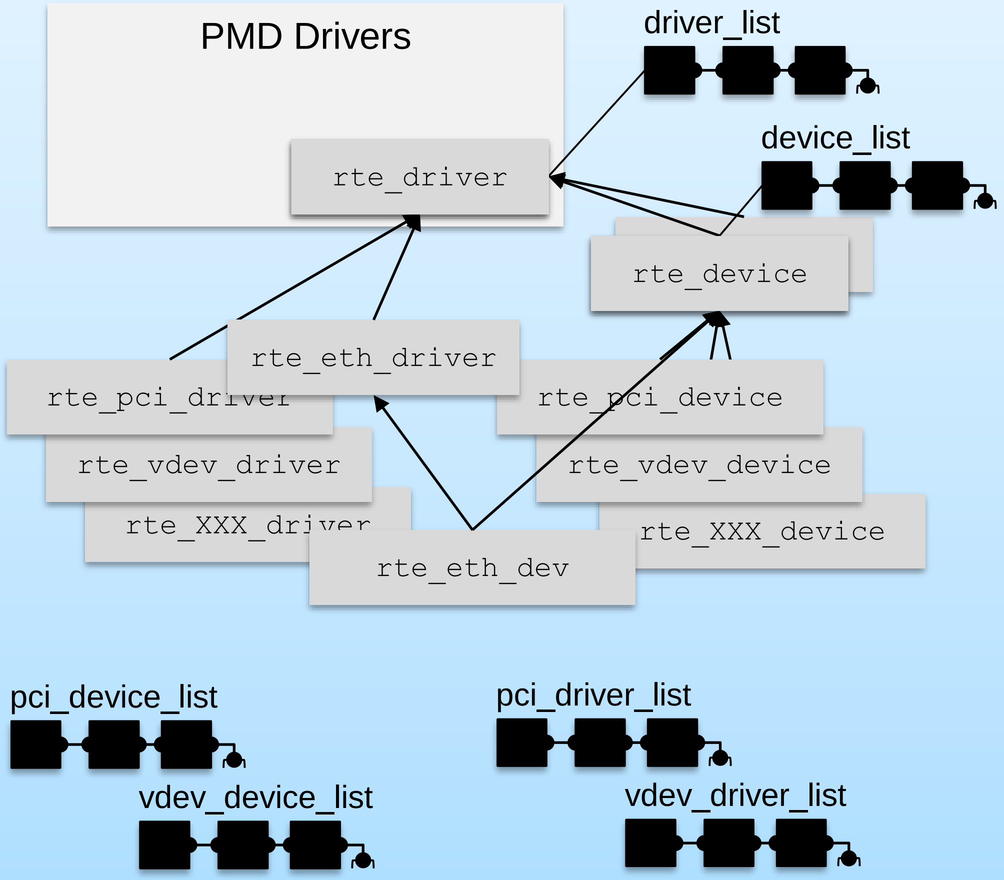
Taking cue from Linux Device Model

- Bus ⇔ Devices ⇔ Drivers
  - Device attaches on a bus
  - Drivers services a device
- DPDK…
  - `rte_pci_device` is attached to a `rte_pci_bus`
  - `rte_pci_driver` services a `rte_pci_device`
    - Service includes probe, remove, hot-plugging
  - EAL init and hot-plugging calls `rte_pci_bus->probe`
    - This in turn calls `rte_pci_driver->probe`



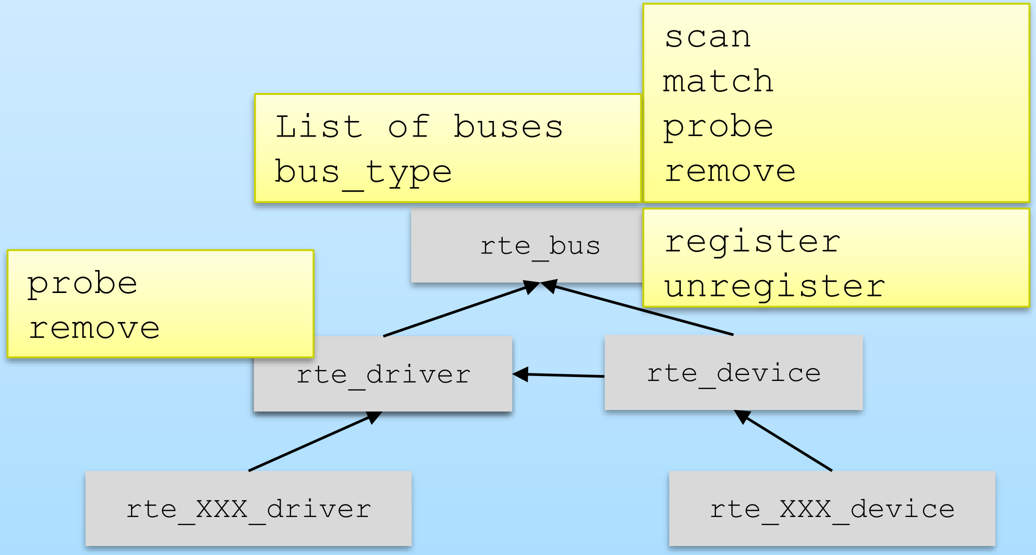
Remodeling Device-Driver Relationship (4)

Step 2



Step 3: Bus ⇔ Driver ⇔ Device

```
rte_eal_init()
`-> rte_eal_devices_init()
    `-> for each bus, call its scan
    `-> For each device scanned, call match
    `-> For each match, call driver->probe
PMDs = Bus + Driver
`-> Register for a Bus
    `-> either existing or create a new
```



Something similar has already been proposed on ML [1].

Agenda

- Introducing NXP SoC
  - NXP DPAA: Datapath Acceleration Architecture
  - SoC and Standardization - incongruous
- Integrating NXP PMD with DPDK: Stumbling block(s)
  - Solution(s) and what needs to be done for it
  - Allowing Custom Scanning for devices
  - Or, introducing a new Device Model: Bus ⇔ Device ⇔ Driver
- Summarizing the status
- Questions/Comments

Balancing long and short term goals

- Complete overhaul is a long term goal
  - It requires quite a lot of deliberation
  - Changes can be transparently done for PMDs, but impact (performance, ABI) is there
- Step-by-Step approach
  - Bring in the pluggable scan way and allow non-PCI device to be introduced
  - Once, more PMDs come in, better picture of use-cases
    - For example, whether platform bus is default or not

What else can be improved

- Updated semantics for External or Offloaded memory pool
  - Applications would prefer non-platform specific implementations
  - Application should be hardware offloading agnostic
    - That’s a platform property
- Possible approach
  - Clear semantics for Packet Mempool which can be offloaded and other mempools
    - APIs should be different. For example, `rte_mempool_create` is only for non-packet buffers
  - In case of unavailability of offloaded pool, transparent fallback to non-offloaded pool
    - Use-case: NFV where applications can be deployed to heterogenous host environment
    - A patch for supporting fallback is posted on ML
      - Transparent fallback vs exposing API for checking support

NXP SoC in DPDK – Status Check

- Run time services for non-IA
  - ☺ Available for ARM, Power8 and other architecture
- Mempool offload framework – to use external or hardware memory managers
  - ☺ Merged in 16.07
- non-PCIe devices support
  - Multiple discussions and patchsets – not much progress in terms of review
  - Phased approach would allow non-PCI PMDs to be introduced
  - Complete overhaul is fairly long term
- Event Driven Programming model
  - RFC posted by Cavium; Intel and NXP contributed in review

QUESTIONS?