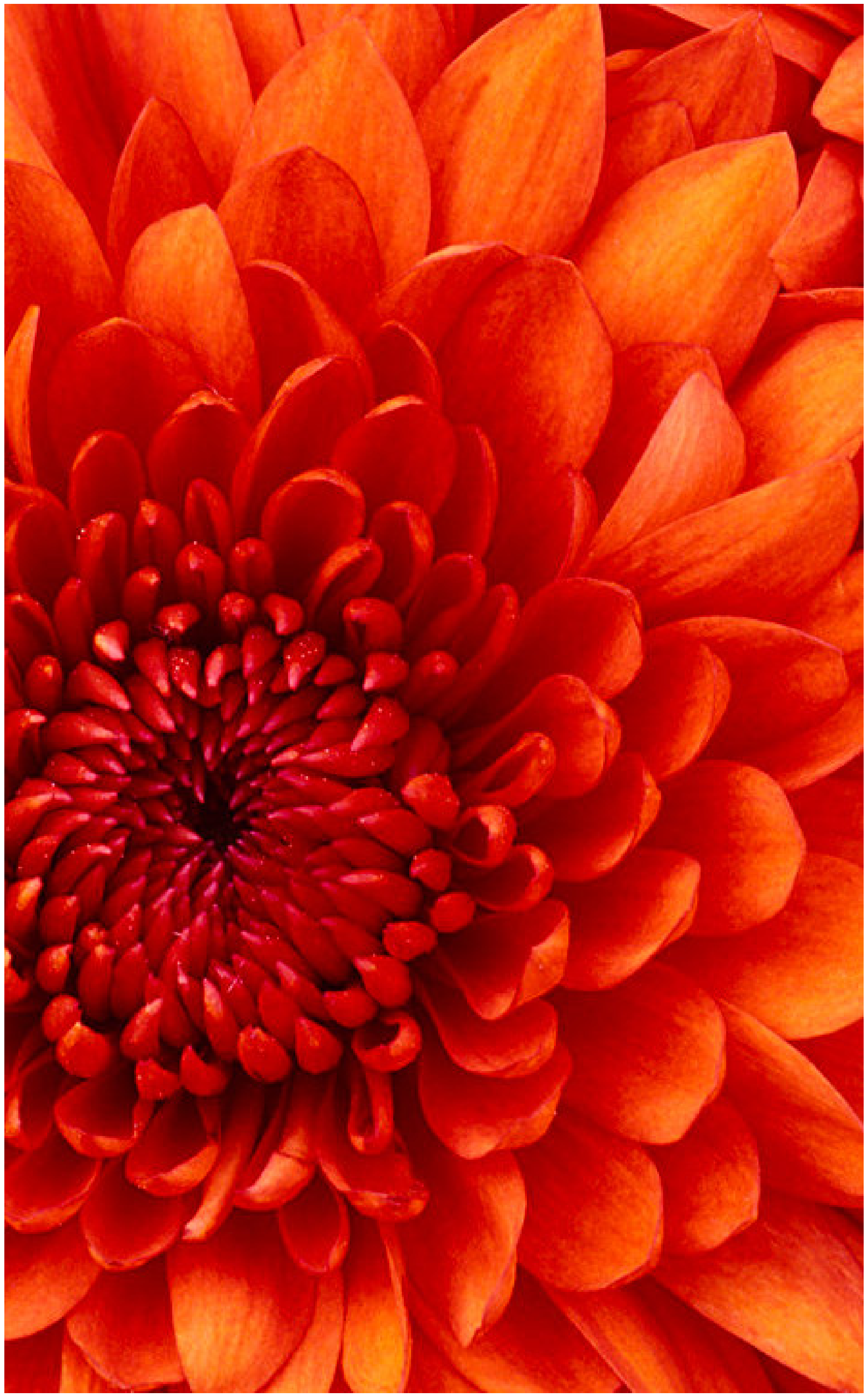
LOGICAL ABSTRACTION AND RESOURCE MANAGEMENT USING THE MANAGEMENT COMPLEX

NIR EREZ

22.MAR.2016

EXTERNAL USE

SECURE CONNECTIONS FOR A SMARTER WORLD



Agenda

- Why Management Complex?
- DPAA2 Logical Objects
  - Example: Data Path Network Interface (DPNI)
- DPAA2 Resource Management
- DPAA2 Network-on-Chip
- DPAA2 Boot Sequence

WHY MANAGEMENT COMPLEX?

QorIQ Layerscape

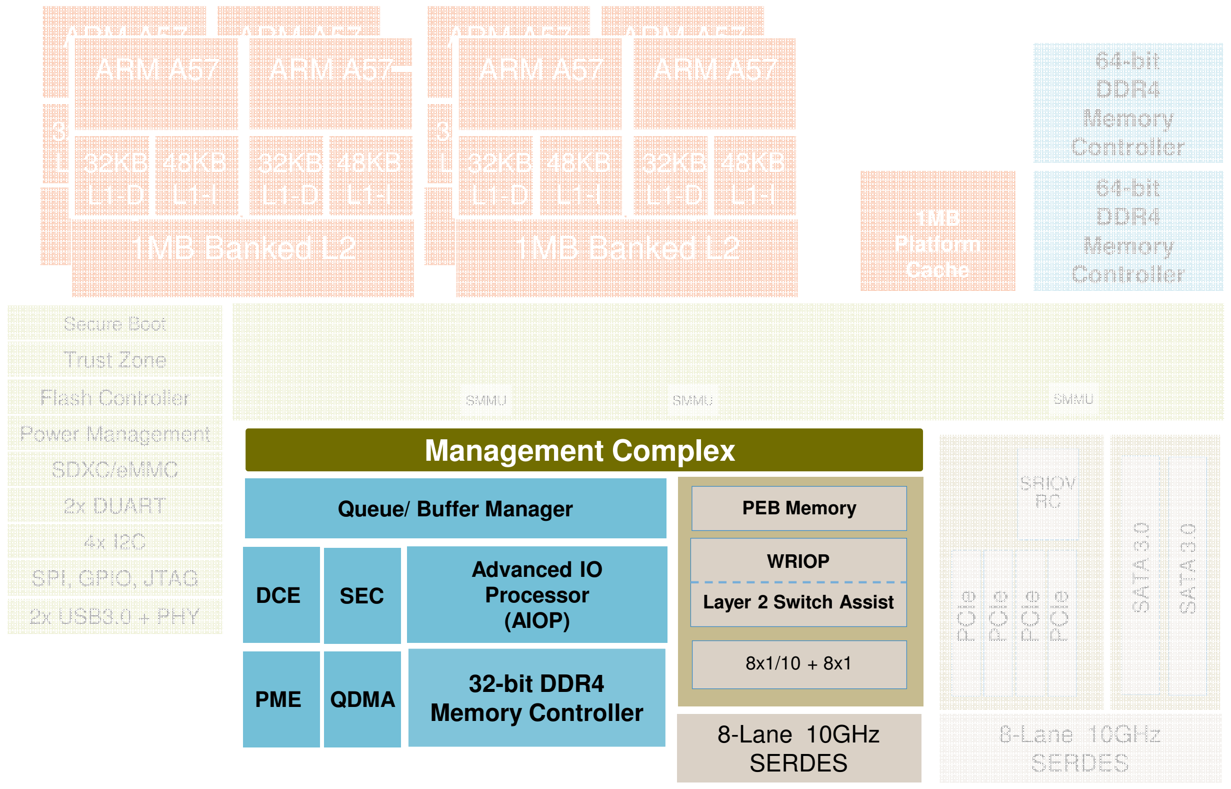
Breakthrough, software-defined approach to advance the world's new virtualized networks

New, high-performance architecture built with ease-of-use in mind
Groundbreaking, flexible architecture that abstracts hardware complexity and enables customers to focus their resources on innovation at the application level

Optimized for software-defined networking applications
Balanced integration of CPU performance with network I/O and C-programmable datapath acceleration that is right-sized (power/performance/cost) to deliver advanced SoC technology for the SDN era

Extending the industry's broadest portfolio of 64-bit multicore SoCs
Built on the ARM® Cortex®-A57 architecture with integrated L2 switch enabling interconnect and peripherals to provide a complete system-on-chip solution

LS2085A SoC – 1st DPAA2 System



Datapath Acceleration

- **SEC** – crypto acceleration
- **DCE** – Data Compression Engine
- **PME** – Pattern Matching Engine
- **QDMA** – Queue-enabled DMA Engine
- **L2 Switching** -- via Datapath Acceleration Hardware
- **Management Complex** – Configuration and Control Abstraction

General Purpose Processing

- 8x ARM® A57 CPUs, 64b, 2.0GHz
  - 1MB L2 cache
- HW L1 & L2 Prefetch Engines
- Neon SIMD in all CPUs
- 1MB L3 platform cache w/ECC
- 4MB Coherent Cache
- 2x64b DDR4 up to 2.4GT/s

Express Packet IO

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

Accelerated Packet Processing

- 20Gbps SEC - crypto acceleration
- 10Gbps Pattern Match/RegEx
- 20Gbps Data Compression Engine
- Advanced I/O Processor
  - 20Mpps advanced forwarding

Network I/O

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

MC Makes it Easy

- ✓ **Presents hardware as logical objects**
- ✓ **Virtualizes and isolates objects**
- ✓ **Hides complex sequences**
- ✓ **Sets up a Network-on-Chip**
- ✓ **Manages resources**
- ✓ **Supports recovery scenarios**

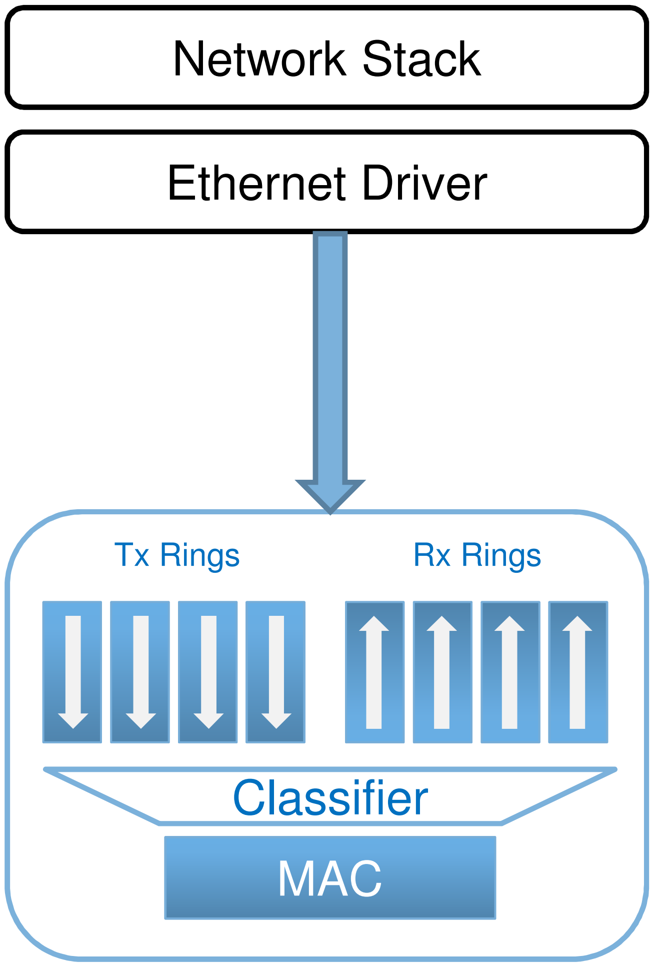
Legacy Ethernet Controller

… translates between network stack's standard features and HW implementation.

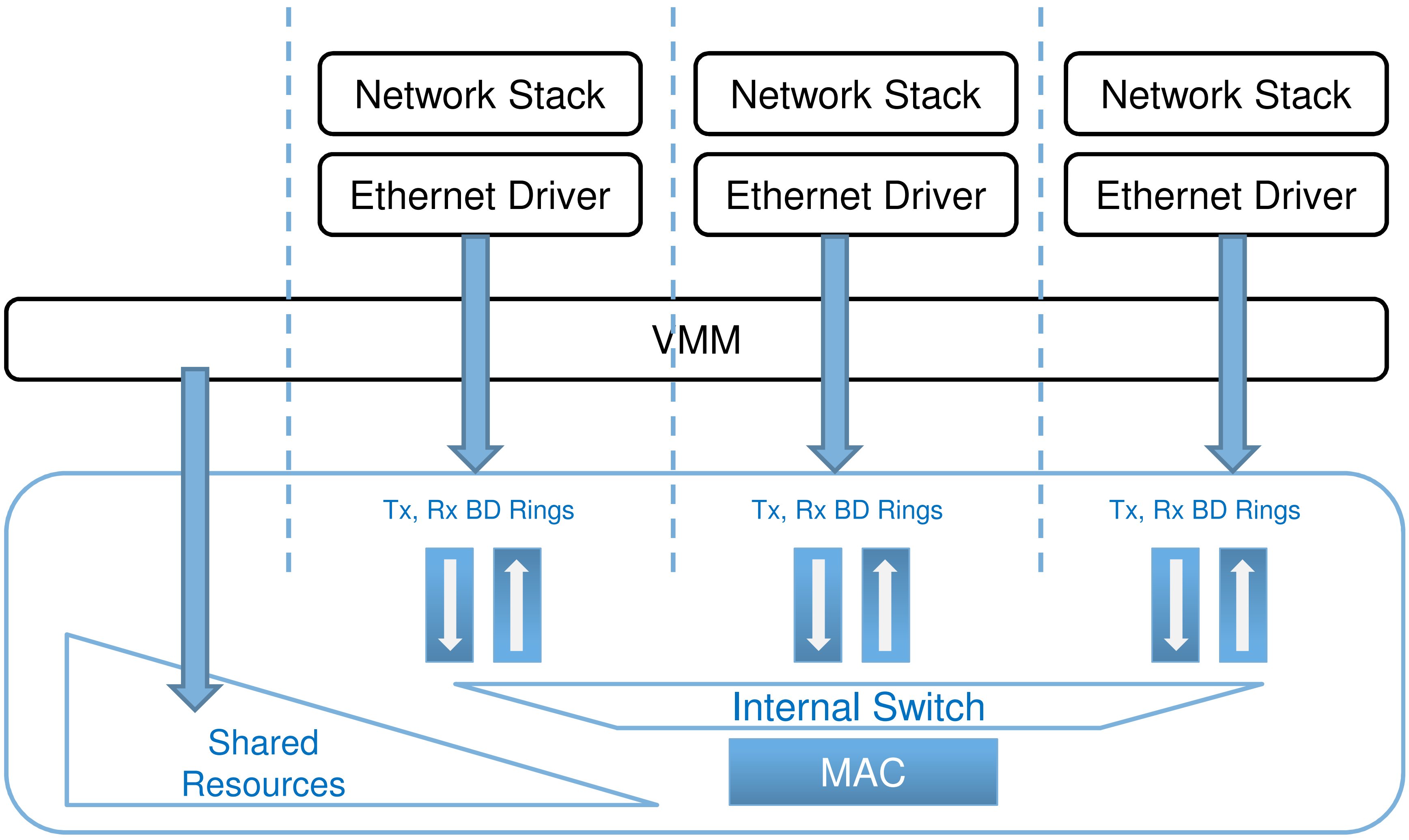
… owns all hardware resources needed to operate the Ethernet device.

All functions are in a single HW block.

Configuration is mostly independent of other blocks.

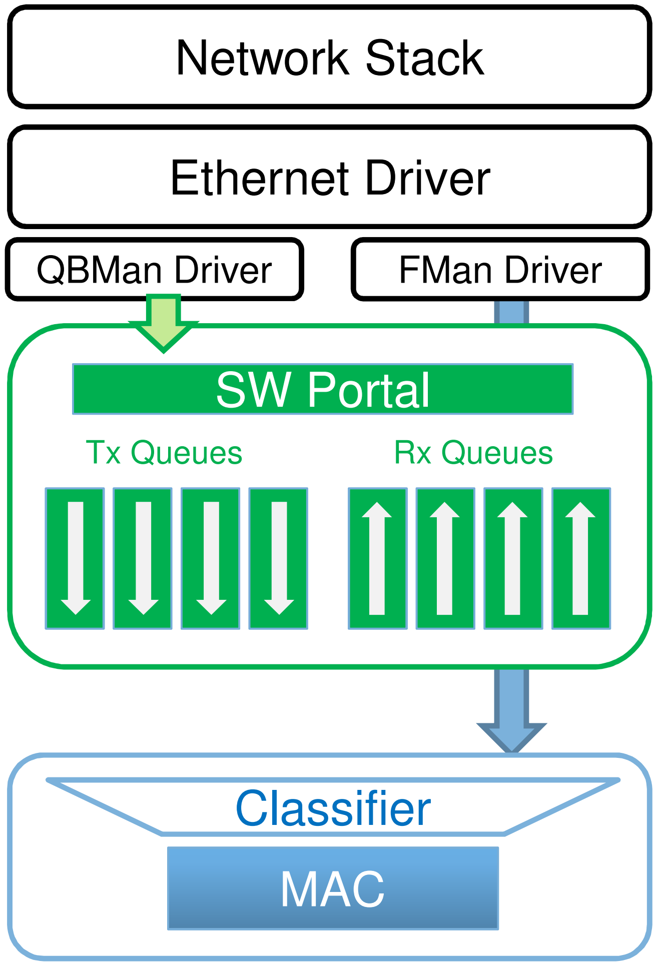


Ethernet Controller with Virtualization Support

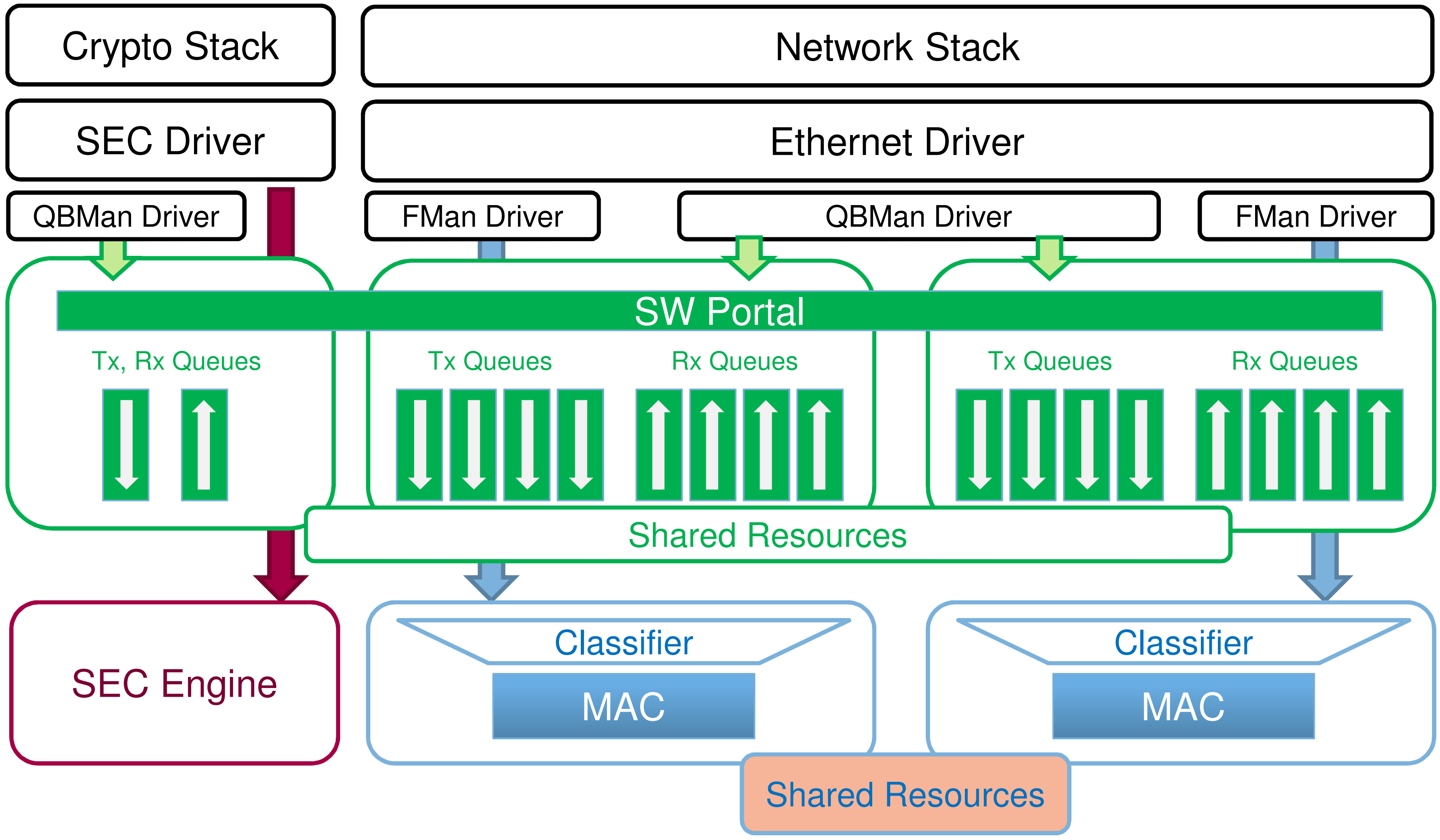


DPAA 1.x Ethernet Controller

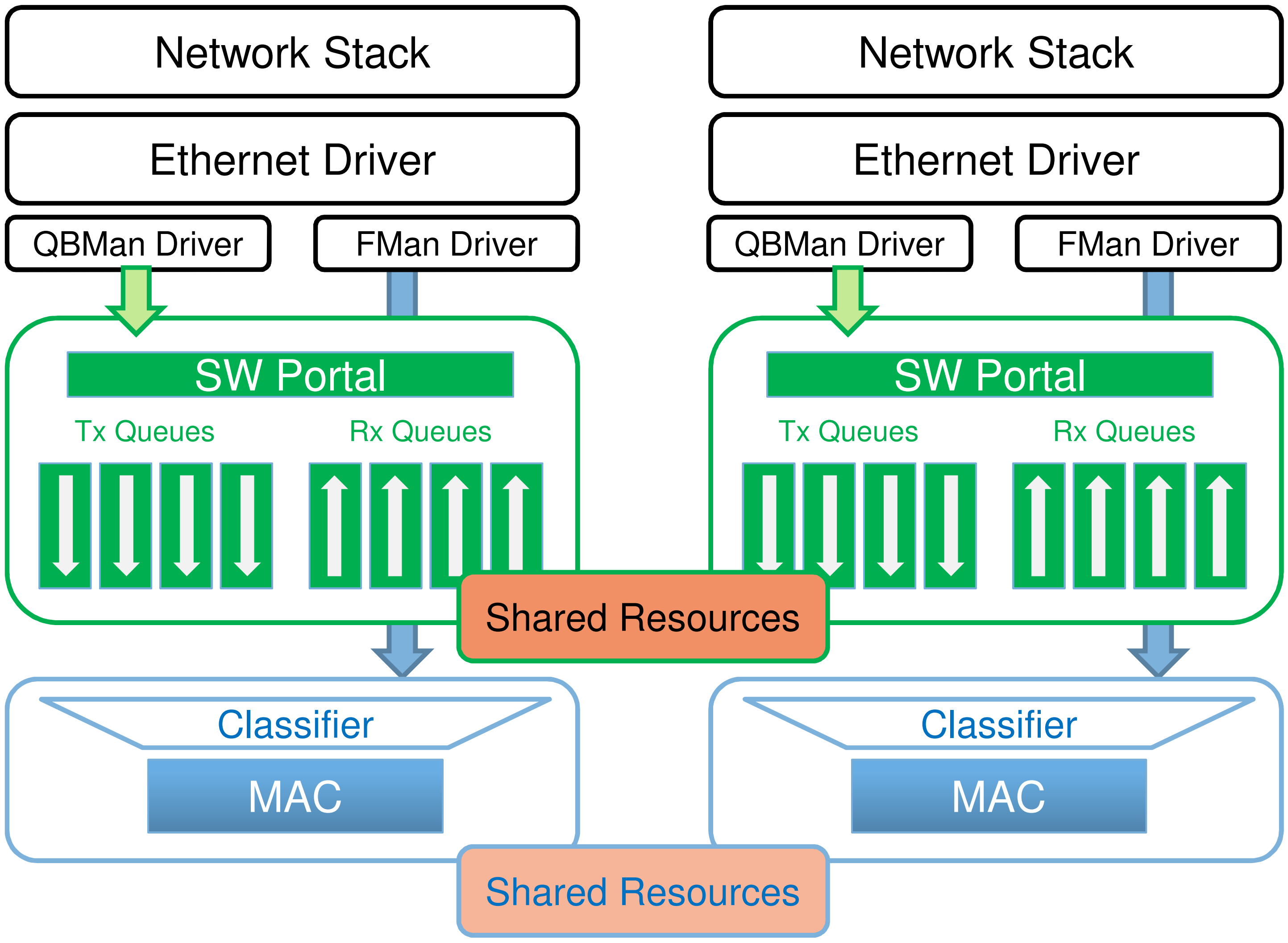
- The Ethernet driver does not own all hardware resources needed to operate the device.
- QBMan resources may serve also other drivers or instances
- Ethernet Controller functions are achieved by multiple HW blocks, and configuration has several dependencies.



DPAA 1.x Ethernet and Crypto Functions (SMP System)



DPAA 1.x Ethernet Controllers (Partitioned System)



Goal: Easy-to-Use Logical Objects

Lack of Virtualization

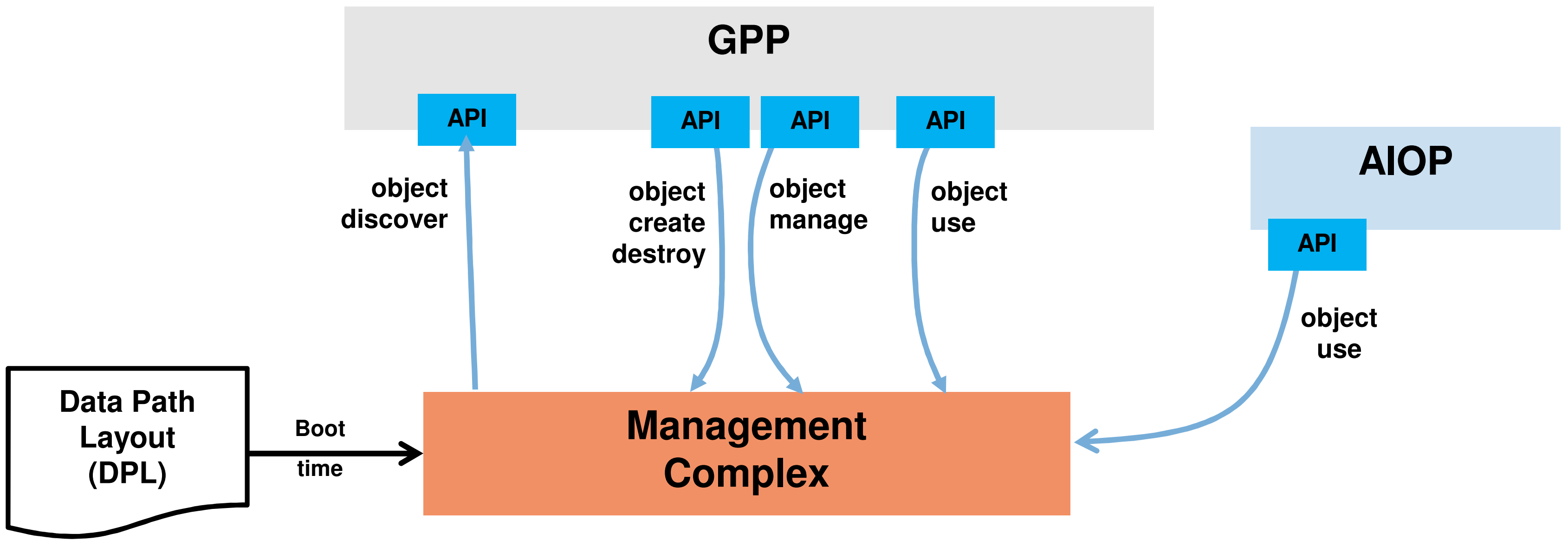
- Centralized resource piles
- Sharing needed but complex
- Requires Hypervisor or IPC

Complex Sequences

- Driver dependencies
- Resource cleanup
- Performance tuning

Management Complex (MC) Concept

- The Management Complex provides **NXP-owned abstraction and control firmware**.
- MC exports **software-defined and standard-oriented interfaces** to GPP and AIOP software, and thus hides configuration complexity from customers.
- **MC is a trusted entity and only executes NXP-supplied trusted firmware.**
  It is isolated from the rest of the SoC so it cannot be compromised by malicious or buggy software running on the GPPs.



Management Complex Roles

- **DPAA Boot and Global Initialization**
  - Global initialization of DPAA hardware blocks (QBMan, WRIOP, AIOP, SEC, etc.)
- **Configuration and Abstraction of Logical Objects**
  allocates the right set of resources and configures them as a logical object:
  - Network interfaces (basic or high-function interfaces)
  - L2 switches and Demux objects (MAC partitioning, VEB/VEPA)
  - Link aggregation groups
  - Accelerator interfaces (SEC, DCE, PME, QDMA)
  - Inter-Partition Communication interfaces (GPP ↔ AIOP, GPP ↔ GPP)
- **DPAA Objects Discovery and Control per Software Context**
- **DPAA Resource Management**
  - Allocation, tracking and recovery in fault scenarios
- **Support DPAA Hardware Virtualization**

DPAA2 LOGICAL OBJECTS

Main Attributes of MC Objects

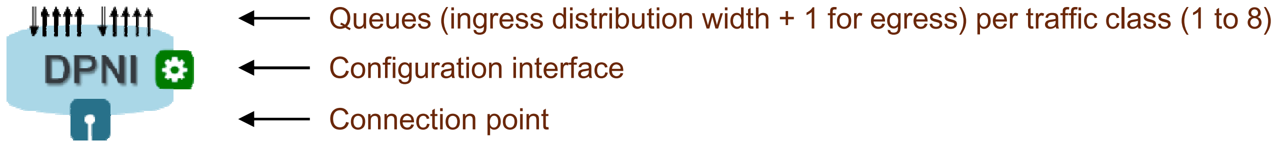
- **Object can be created dynamically**
  - Allocates hardware resources and configures them to initial state
- **Object can be destroyed dynamically**
  - Gracefully shuts down and releases all its hardware resources

- **Object can be enabled and disabled**
- **Object can be reset to initial state**

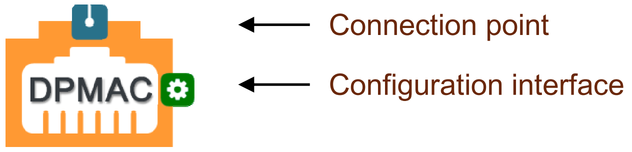
- **Object belongs to a single software context (at a time)**
- **Object can be assigned to another software context**
- **Object may interrupt its software context**

DPAA2 Logical Objects

- **Network Interfaces:**
  - **Data Path Network Interface (DPNI)**
    - A standard network interface (L2 and up), as expected by standard network stacks/applications.
    - Offers a wide range of standard offloads:
      MAC & VLAN Filtering, QoS, checksums, time-stamping, policing, IPR, IPF, IPSec, RSC, GSO, etc.
    - Configurable as a tunnel/fast-path interface (non-L2 packet), suitable for GPP-AIOP interaction.

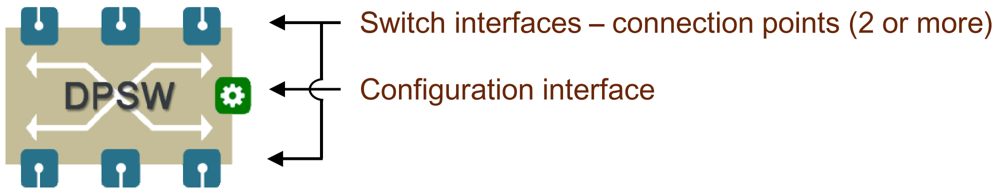

DPNI
Queues (ingress distribution width + 1 for egress) per traffic class (1 to 8)
Configuration interface
Connection point

- **Physical Interfaces:**
  - **Data Path MAC (DPMAC)**
    - Serves for physical MAC and MDIO control

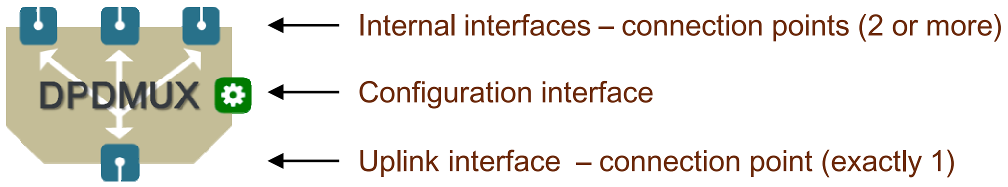


DPAA2 Logical Objects (cont.)

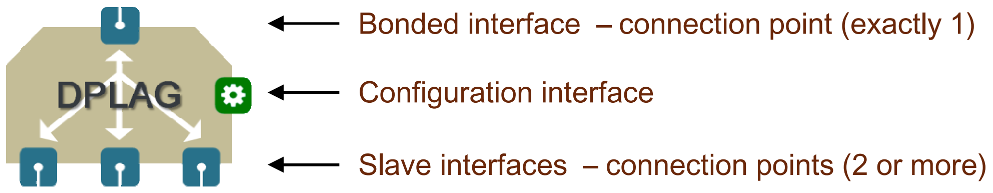
- **Switching and Aggregation:**
  - **Data Path Switch (DPSW) –** Standard implementation of L2 Switch.



  - **Data Path Demux (DPDMUX) –** Allows partitioning of a physical interface into multiple (isolated) logical interfaces. May be used for setting up different Ethernet Virtual Bridging (EVB) objects, such as VEB, VEPA, or S-Component.

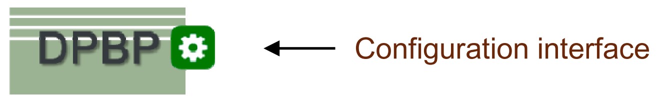

DPDMUX
Internal interfaces – connection points (2 or more)
Configuration interface
Uplink interface – connection point (exactly 1)

  - **Data Path Link Aggregator (DPLAG) –** aggregates multiple physical links into a single logical link. (NOT available in LS2085 rev-1)

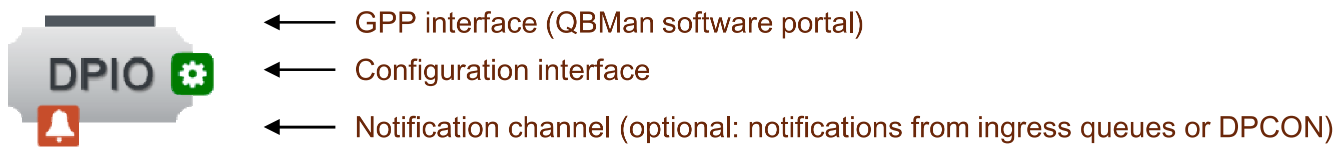

DPLAG
Bonded interface – connection point (exactly 1)
Configuration interface
Slave interfaces – connection points (2 or more)

DPAA2 Logical Objects (cont.)

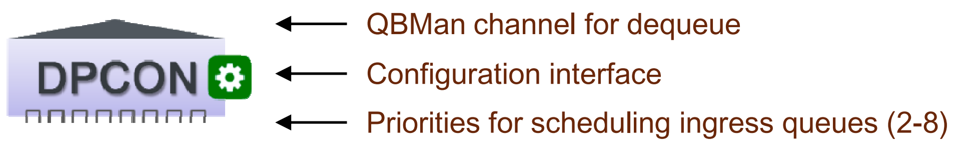
- **Supporting Infrastructure Objects:**
  - **Data Path Buffer Pool (DPBP)** – An abstraction of BMan buffer pool



  - **Data Path I/O Portal (DPIO)** – Enables enqueue/dequeue via QMan portals and getting ingress notifications

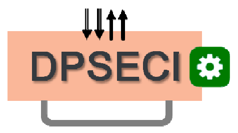

DPIO
GPP interface (QBMan software portal)
Configuration interface
Notification channel (optional: notifications from ingress queues or DPCON)

  - **Data Path Concentrator (DPCON)** – Scheduling object for advanced scheduling of ingress packets from multiple interfaces.


DPCON
QBMan channel for dequeue
Configuration interface
Priorities for scheduling ingress queues (2-8)

DPAA2 Logical Objects (cont.)

- **Accelerator Interfaces:**
  - **Security Accelerator Interface (DPSECI)**



Queues (1-8 for ingress, 1-8 for egress)
Configuration interface

  - **Data Compression Accelerator Interface (DPDCEI)**

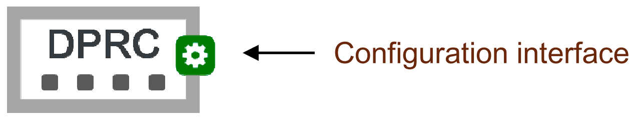
Queues (1 for ingress, 1 for egress)
Configuration interface

  - **DMA Accelerator Interface (DPDMAI)**

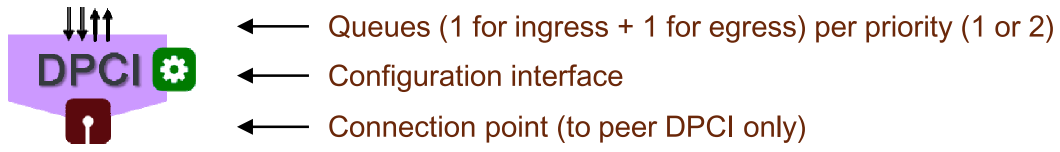
Queues (1-2 for ingress, 1-2 for egress)
Configuration interface

DPAA2 Logical Objects (cont.)

- **Management Objects:**
  - **Data Path Resource Container (DPRC):**
    - Allows software context to assign DPAA objects and resources.
    - Allows software context to create network topology by connecting network objects.
    - Functions as virtual bus, so software context may query DPAA objects and associate with OS device drivers.



- **Inter-Partition Communication:**
  - **Data Path Communication Interface (DPCI)** – allows communication between different software contexts through QMan infrastructure, which is not limited to network packet format. Useful for IPC between two GPP software entities, or between GPP and AIOP entities. The communication protocol is user-defined.


DPCI
Queues (1 for ingress + 1 for egress) per priority (1 or 2)
Configuration interface
Connection point (to peer DPCI only)

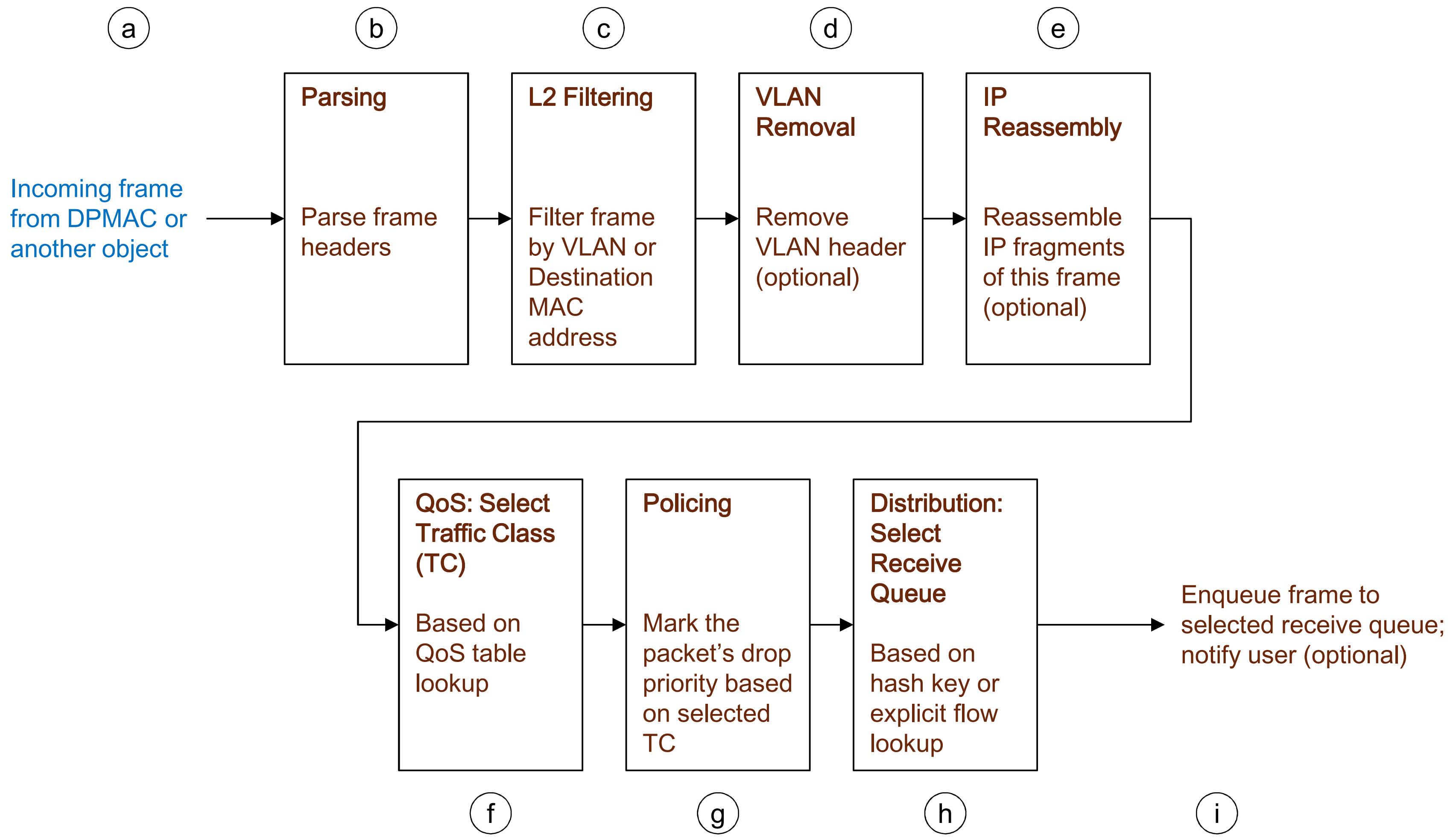
DPNI (Data Path Network Interface)

- **Wire-Speed Frame Parsing**
  - Parsing results may be visible in frame annotation area
- **Filtering of Received Frames**
  - Exact-match filtering based on destination MAC address and/or VLAN IDs
  - Unicast promiscuous and Multicast promiscuous modes
- **QoS Support**
  - Packet classification up to eight traffic classes, based on user-defined key
  - Policing based on classification result (tail-drop or WRED)
- **Distribution to Receive Queues**
  - Statistical distribution based on hash-generated key (RSS)
  - Explicit flow steering based on user-defined key
- **Up to Eight Different Buffer Pools**
- **Various Scheduling Options for Received Packets**

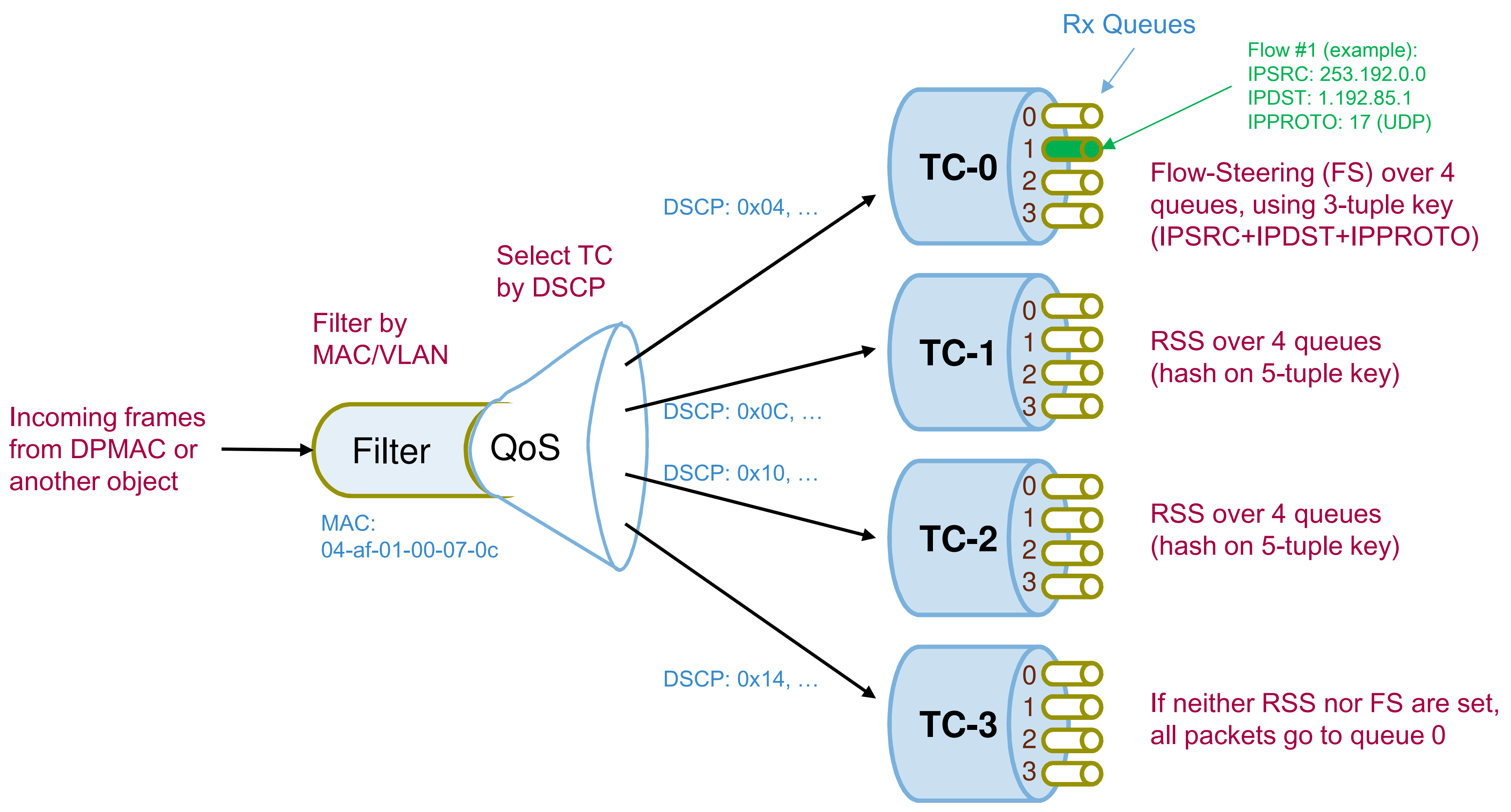
DPNI (Data Path Network Interface)

- **Traffic Shaping of Transmitted Packets**
  - Up to eight transmit queues (traffic classes)
  - Rate limit
- **Various Offload Functions:**
  - L3 and L4 checksum generation and validation
  - VLAN add/remove
  - IP Reassembly and Fragmentation
  - GRO and GSO
  - IPSec transport
- **Link-ased and Priority-based Flow Control (PFC)**
  - Supporting queue congestion and/or buffer pool depletion
- **PTP** (IEEE 1588) time-stamping
- **Network Interface Statistics**
- **Network Interface Enable, Disable, Reset**

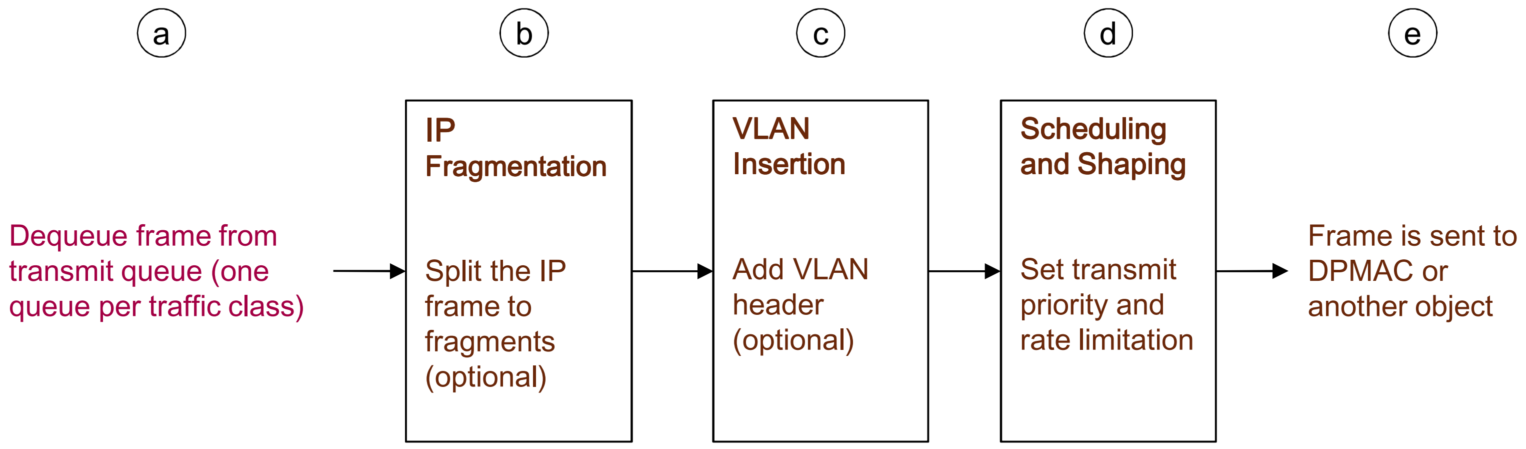
DPNI Ingress Frame Processing


a
b
c
d
e
Incoming frame from DPMAC or another object
Parsing
Parse frame headers
L2 Filtering
Filter frame by VLAN or Destination MAC address
VLAN Removal
Remove VLAN header (optional)
IP Reassembly
Reassemble IP fragments of this frame (optional)
QoS: Select Traffic Class (TC)
Based on QoS table lookup
Policing
Mark the packet's drop priority based on selected TC
Distribution: Select Receive Queue
Based on hash key or explicit flow lookup
Enqueue frame to selected receive queue; notify user (optional)
f
g
h
i

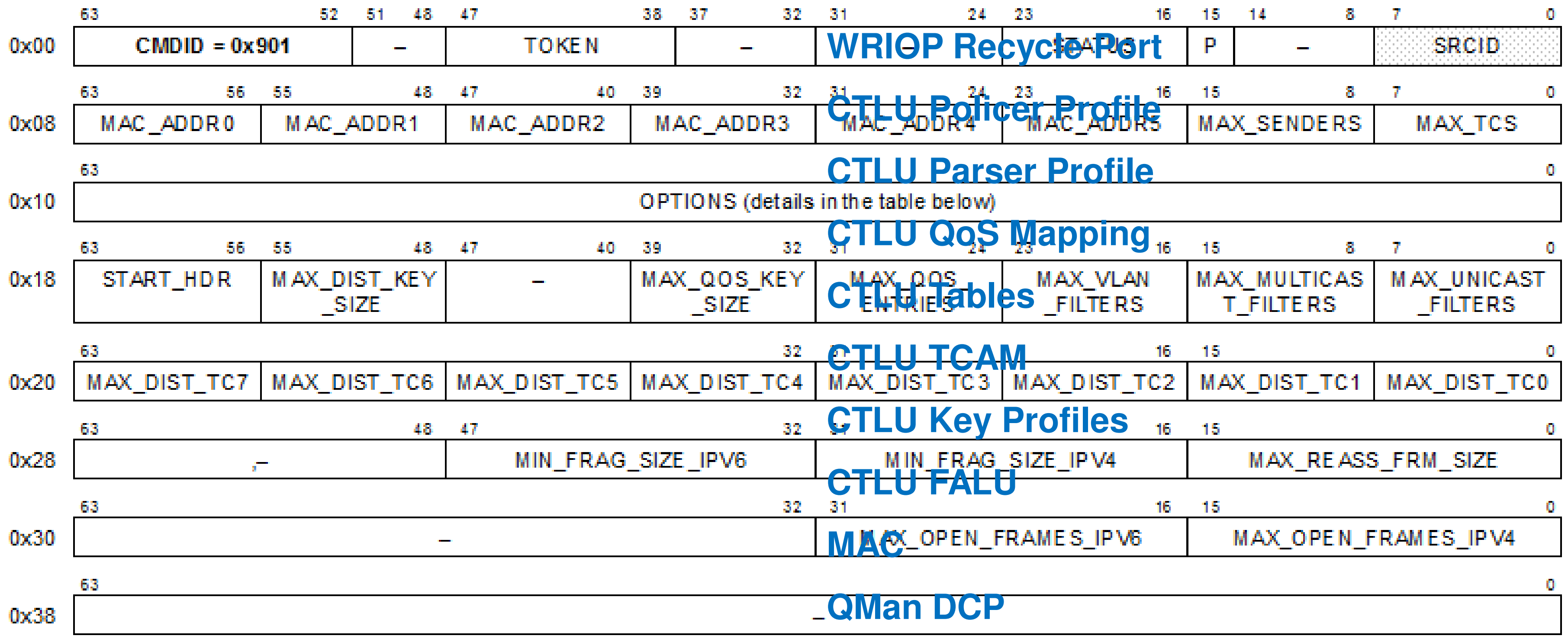
DPNI Ingress Example


Rx Queues
Flow #1 (example):
IPSRC: 253.192.0.0
IPDST: 1.192.85.1
IPPROTO: 17 (UDP)
TC-0
0
1
2
3
Flow-Steering (FS) over 4 queues, using 3-tuple key (IPSRC+IPDST+IPPROTO)
DSCP: 0x04, …
Select TC by DSCP
Filter by MAC/VLAN
TC-1
0
1
2
3
RSS over 4 queues (hash on 5-tuple key)
DSCP: 0x0C, …
Incoming frames from DPMAC or another object
Filter
QoS
DSCP: 0x10, …
TC-2
0
1
2
3
RSS over 4 queues (hash on 5-tuple key)
MAC:
04-af-01-00-07-0c
DSCP: 0x14, …
TC-3
0
1
2
3
If neither RSS nor FS are set, all packets go to queue 0

DPNI Egress Frame Processing


a
b
c
d
e
Dequeue frame from transmit queue (one queue per traffic class)
IP Fragmentation
Split the IP frame to fragments (optional)
VLAN Insertion
Add VLAN header (optional)
Scheduling and Shaping
Set transmit priority and rate limitation
Frame is sent to DPMAC or another object

Objects Configuration – Easy to Use Commands



Offset	Bits	Field
0x00	63–52	CMDID = 0x901
0x00	51–48	–
0x00	47–38	TOKEN
0x00	37–32	–
0x00	31–24	–
0x00	23–16	STATUS
0x00	15	P
0x00	14–8	–
0x00	7–0	SRCID
0x08	63–56	MAC_ADDR0
0x08	55–48	MAC_ADDR1
0x08	47–40	MAC_ADDR2
0x08	39–32	MAC_ADDR3
0x08	31–24	MAC_ADDR4
0x08	23–16	MAC_ADDR5
0x08	15–8	MAX_SENDERS
0x08	7–0	MAX_TCS
0x10	63–0	OPTIONS (details in the table below)
0x18	63–56	START_HDR
0x18	55–48	MAX_DIST_KEY_SIZE
0x18	47–40	–
0x18	39–32	MAX_QOS_KEY_SIZE
0x18	31–24	MAX_QOS_ENTRIES
0x18	23–16	MAX_VLAN_FILTERS
0x18	15–8	MAX_MULTICAST_FILTERS
0x18	7–0	MAX_UNICAST_FILTERS
0x20	63–56	MAX_DIST_TC7
0x20	55–48	MAX_DIST_TC6
0x20	47–40	MAX_DIST_TC5
0x20	39–32	MAX_DIST_TC4
0x20	31–24	MAX_DIST_TC3
0x20	23–16	MAX_DIST_TC2
0x20	15–8	MAX_DIST_TC1
0x20	7–0	MAX_DIST_TC0
0x28	63–48	–
0x28	47–32	MIN_FRAG_SIZE_IPV6
0x28	31–16	MIN_FRAG_SIZE_IPV4
0x28	15–0	MAX_REASS_FRM_SIZE
0x30	63–32	–
0x30	31–16	MAX_OPEN_FRAMES_IPV6
0x30	15–0	MAX_OPEN_FRAMES_IPV4
0x38	63–0	–

WRIOP IFP

WRIOP PPID

WRIOP Recycle Port

CTLU Policer Profile

CTLU Parser Profile

CTLU QoS Mapping

CTLU Tables

CTLU TCAM

CTLU Key Profiles

CTLU FALU

MAC

QMan DCP

QDID

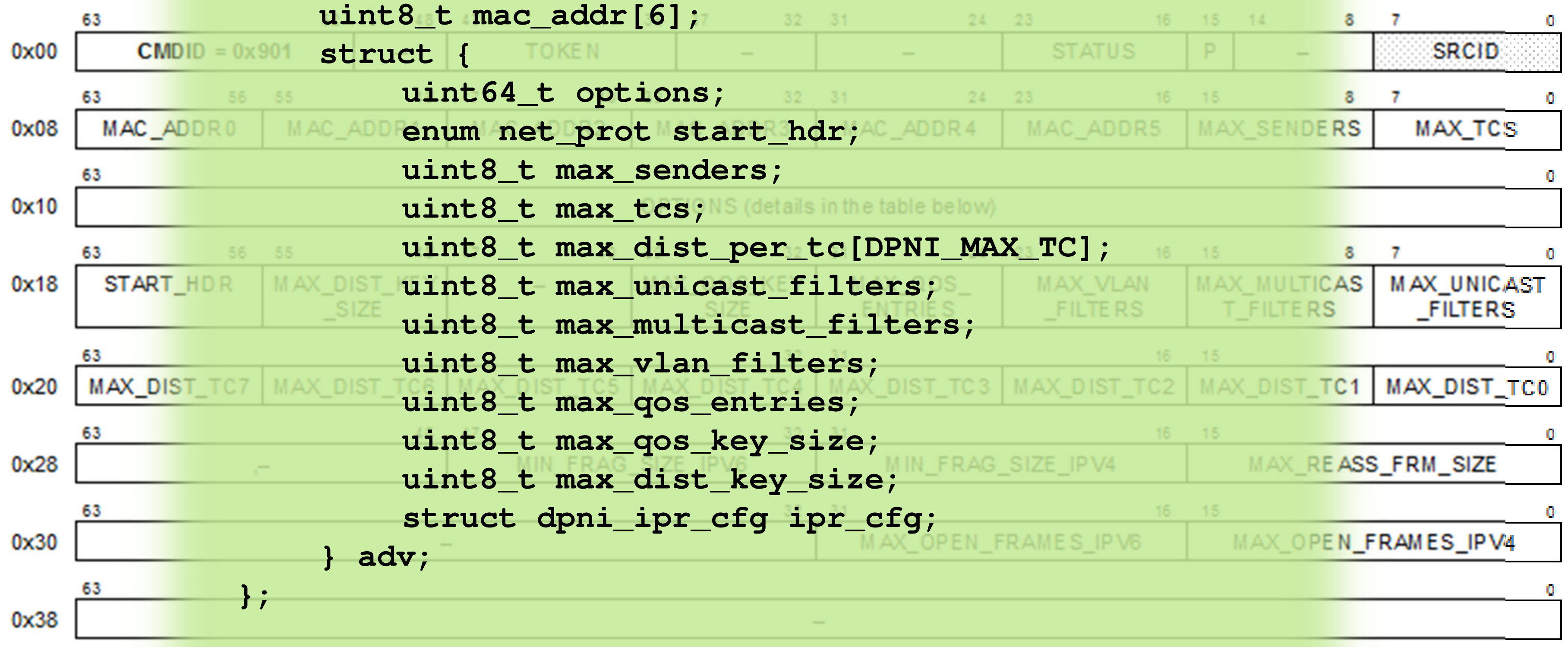
QPR

FQIDs

Congestion Groups

WQ Channels

Objects Configuration – Easy to Use API



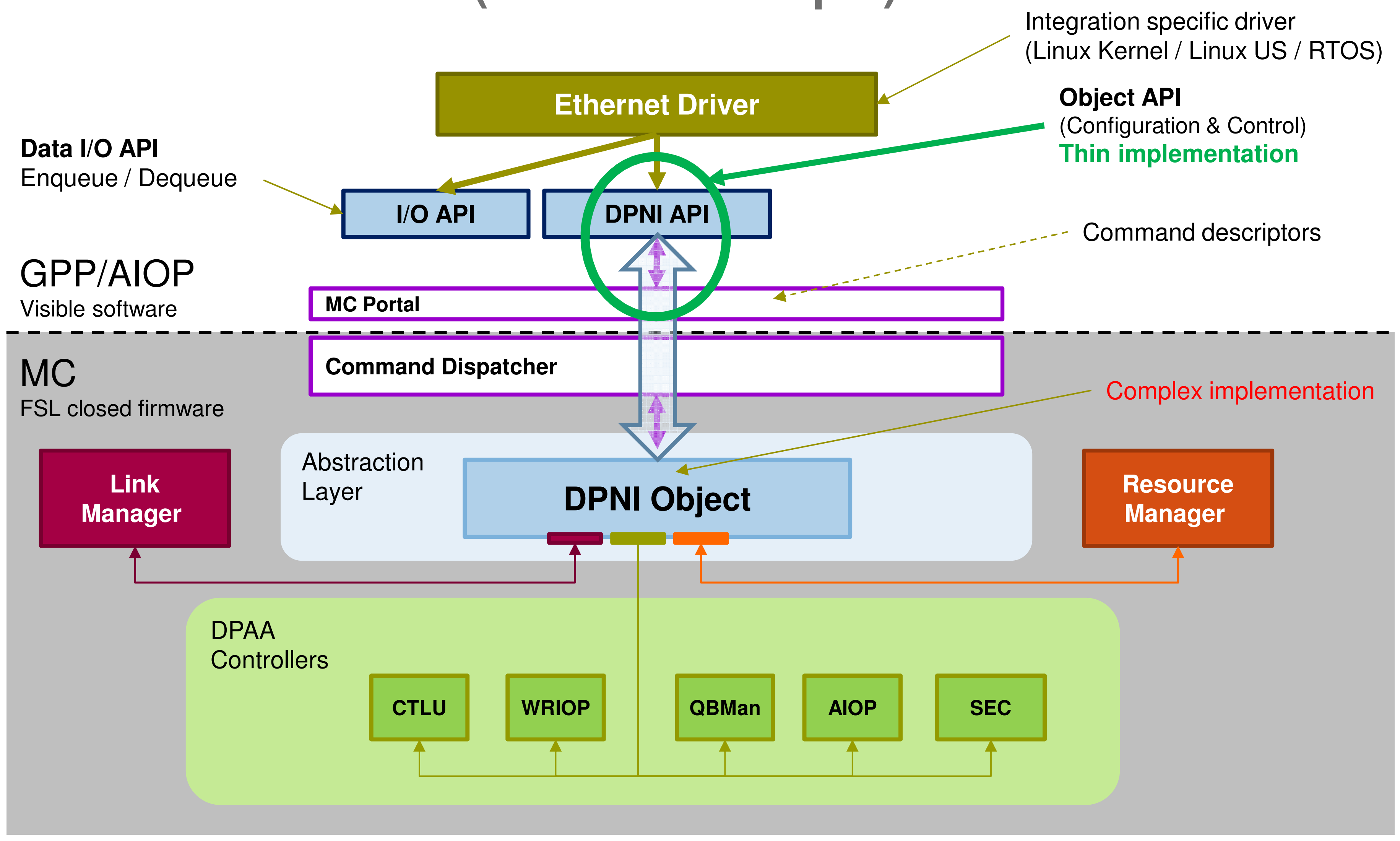
```
struct dpni_cfg {
    uint8_t mac_addr[6];
    struct {
        uint64_t options;
        enum net_prot start_hdr;
        uint8_t max_senders;
        uint8_t max_tcs;
        uint8_t max_dist_per_tc[DPNI_MAX_TC];
        uint8_t max_unicast_filters;
        uint8_t max_multicast_filters;
        uint8_t max_vlan_filters;
        uint8_t max_qos_entries;
        uint8_t max_qos_key_size;
        uint8_t max_dist_key_size;
        struct dpni_ipr_cfg ipr_cfg;
    } adv;
};

int dpni_create(struct fsl_mc_io *mc_io,
                const struct dpni_cfg *cfg,
                uint16_t *token);
```

Example: Ethernet Driver Sequence

```
/* (1) DPIO creation */
dpio_cfg.channel_mode = DPIO_LOCAL_CHANNEL;
dpio_cfg.num_priorities = 4;
dpio_create(drv->mc_io, &dpio_cfg, &token);
dpio_enable(drv->mc_io, token);


/* (2) DPBP creation */
dpbp_create(drv->mc_io, &dpbp_cfg, &token);
dpbp_enable(drv->mc_io, token);
dpbp_get_attributes(drv->mc_io, token &dpbp_attr);
/* use dpbp_attr.bpid to fill buffers pool*/
```

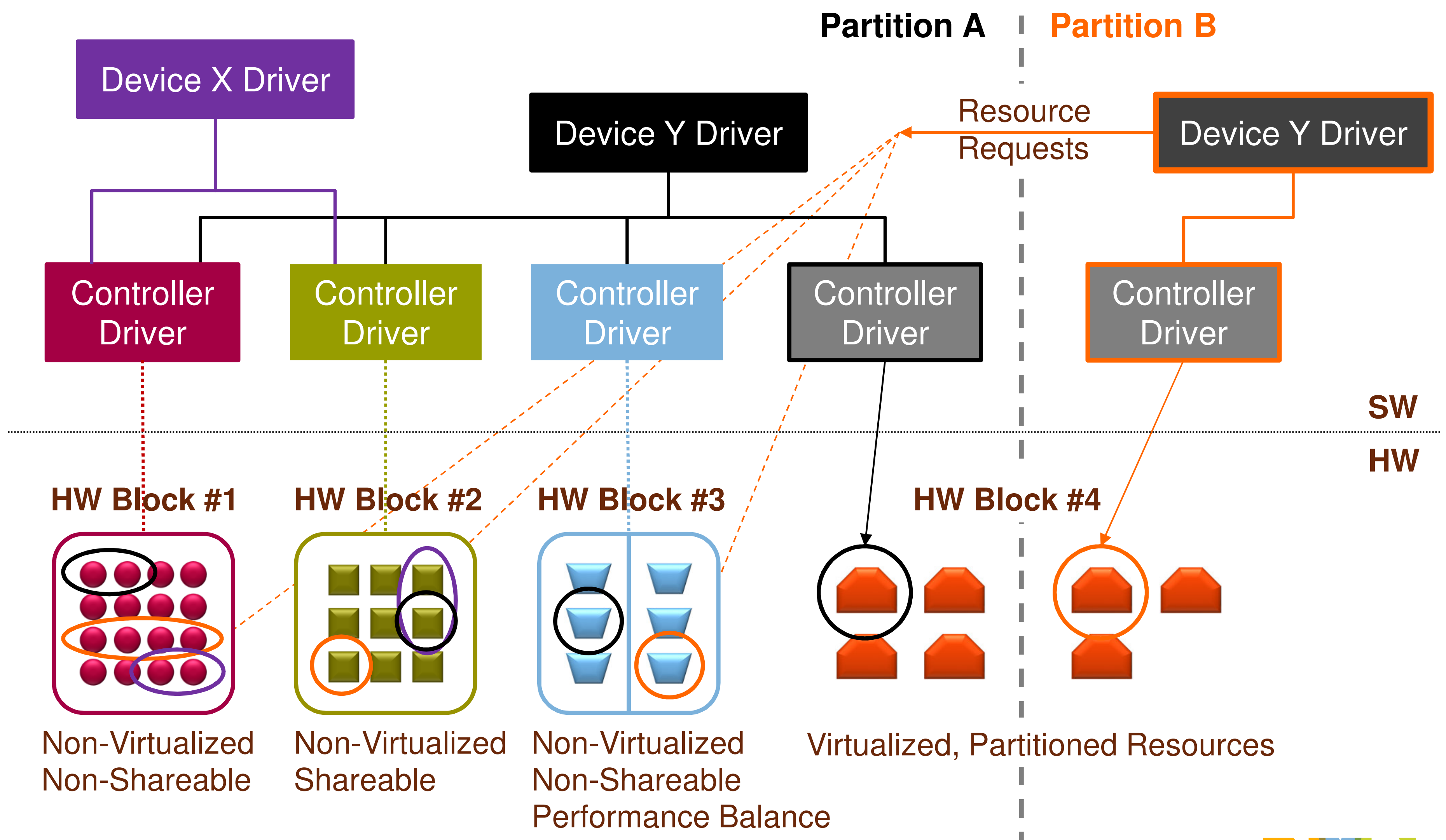
```
/* (3) DPNI creation */
dpni_cfg.mac_addr = { ... };
dpni_cfg.adv.max_tcs = 4;
dpni_cfg.adv.max_unicast_filters = 32;
(+ other standard features / offload features)
dpni_create(drv->mc_io, &dpni_cfg, &token);

/* set buffer pools */
pools_cfg.num_dpbp = 1;
pools_cfg.pools[0].dpbp_id = dpbp_attr.id;
pools_cfg.pools[0].buffer_size = 512;
dpni_set_pools(drv->mc_io, token, &pools_cfg);

dpni_enable(drv->dpni);

/* runtime control operations */
dpni_add_vlan_id(drv->mc_io, token, 0x0200);
```

DPAA2 Ease of Use (DPNI as example)

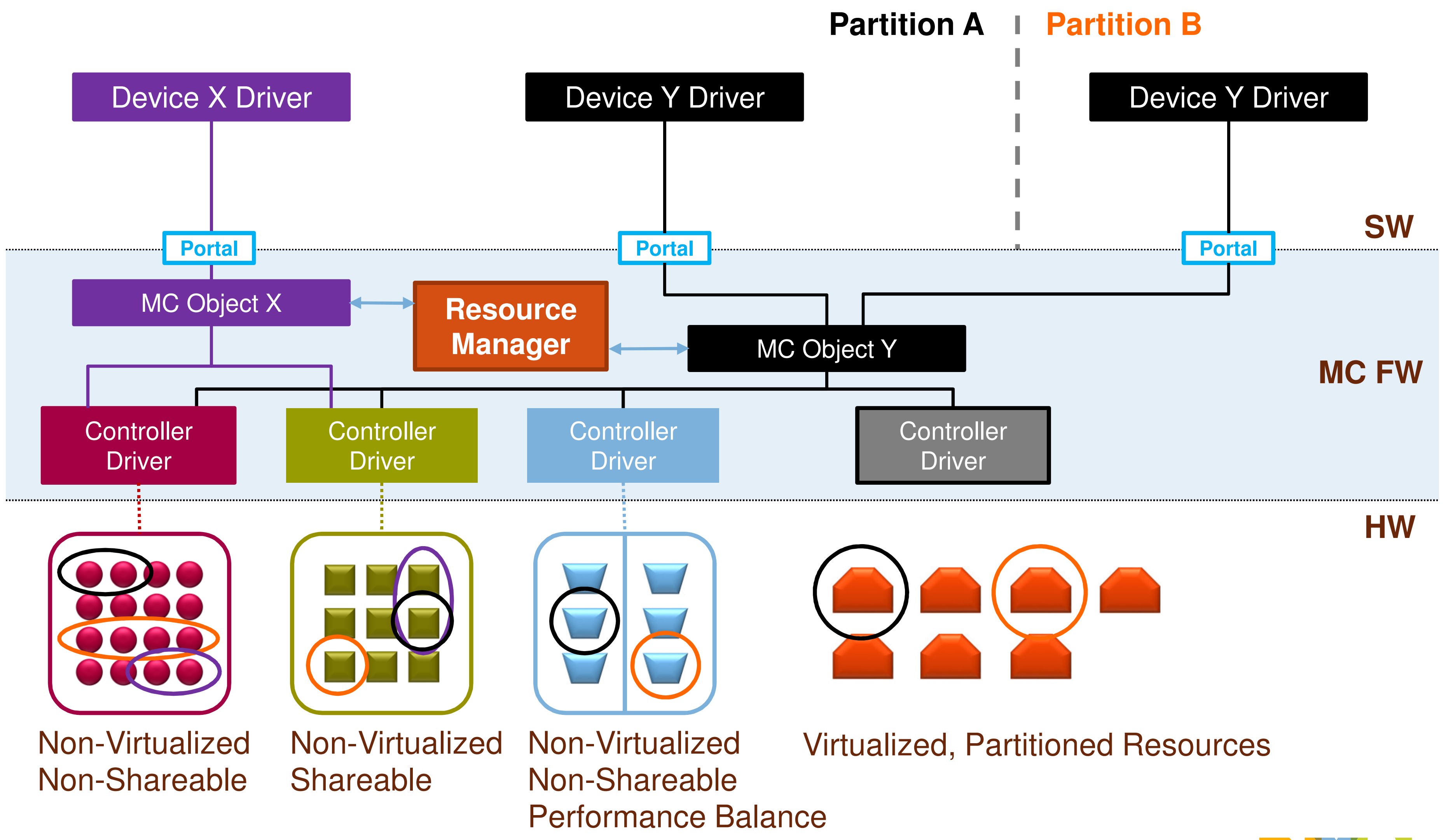


DPAA2 RESOURCE MANAGEMENT

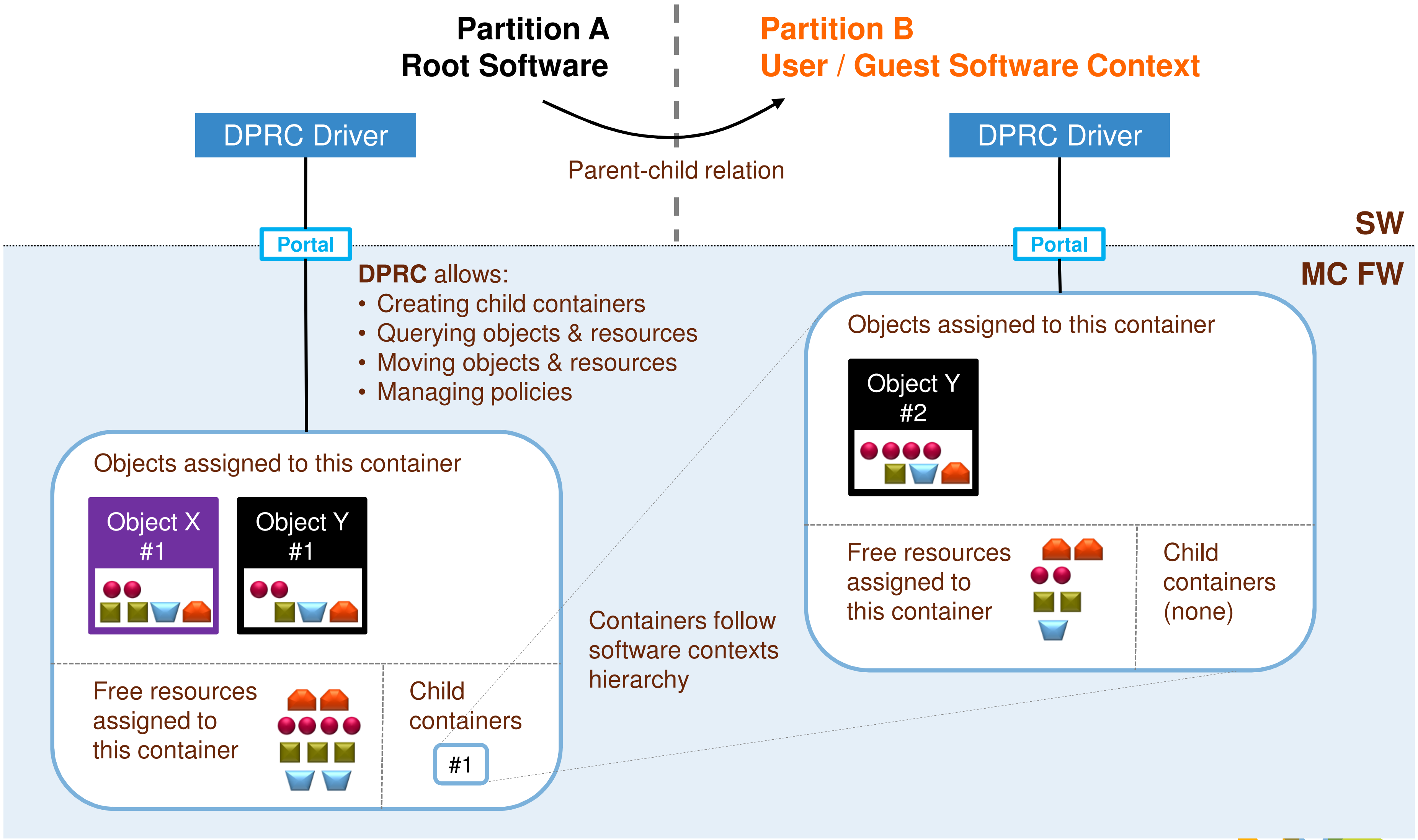
Problem: DPAA Hardware is Hard to Use



MC Presents: Logical Objects Abstraction



MC Presents: Data Path Resource Containers (DPRC)

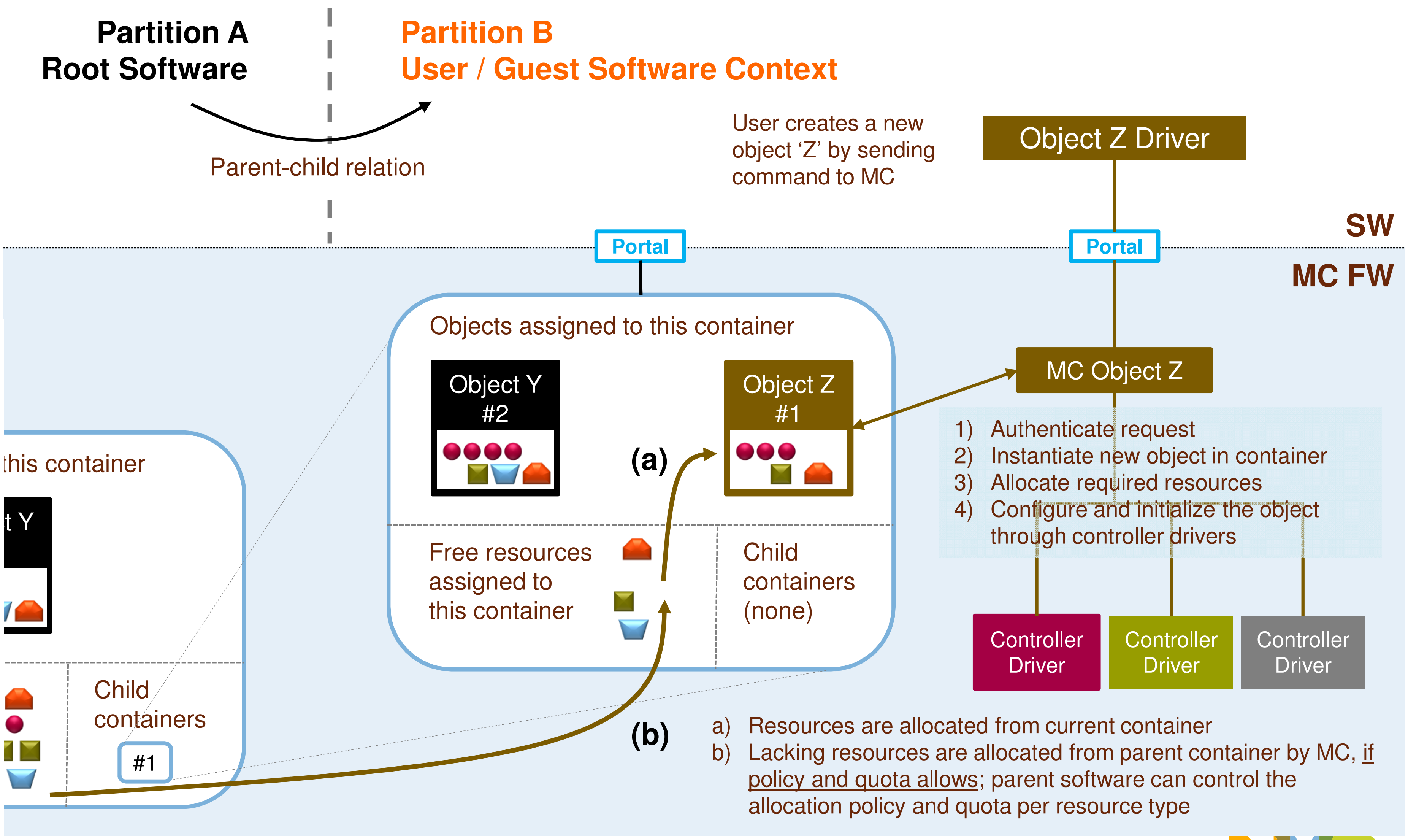


Partition A
Root Software
Partition B
User / Guest Software Context
Parent-child relation
DPRC Driver
DPRC Driver
Portal
Portal
SW
MC FW
Objects assigned to this container
Object X #1
Object Y #1
Free resources assigned to this container
Child containers
#1
Containers follow software contexts hierarchy
Objects assigned to this container
Object Y #2
Free resources assigned to this container
Child containers (none)

DPRC allows:

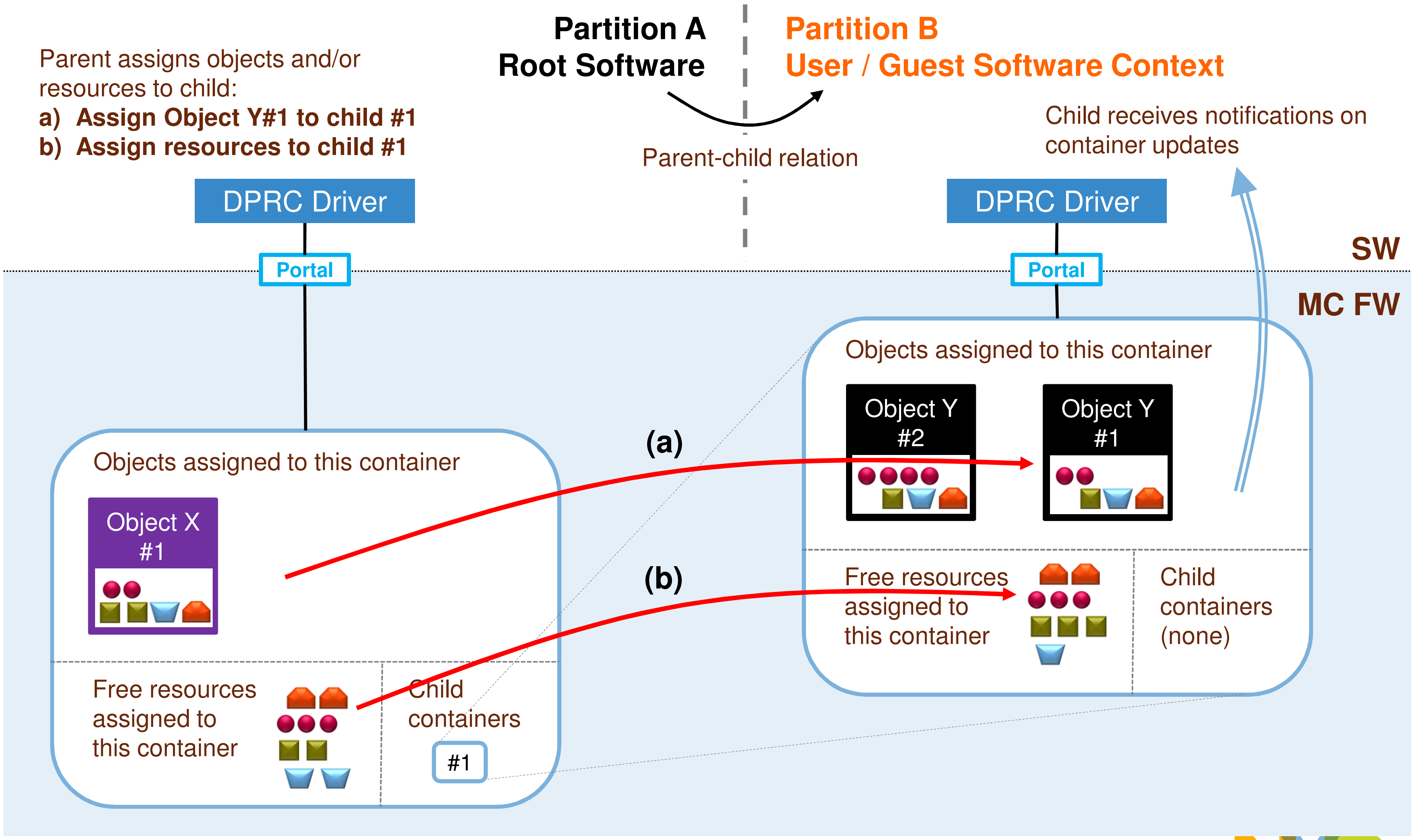
- Creating child containers
- Querying objects & resources
- Moving objects & resources
- Managing policies

Resource Management: Creating MC Objects


Partition A
Root Software
Partition B
User / Guest Software Context
Parent-child relation
User creates a new object 'Z' by sending command to MC
Object Z Driver
SW
MC FW
Portal
Portal
Objects assigned to this container
Object Y #2
Object Z #1
(a)
MC Object Z
1) Authenticate request
2) Instantiate new object in container
3) Allocate required resources
4) Configure and initialize the object through controller drivers
Free resources assigned to this container
Child containers (none)
Controller Driver
Controller Driver
Controller Driver
this container
t Y
Child containers
#1
(b)

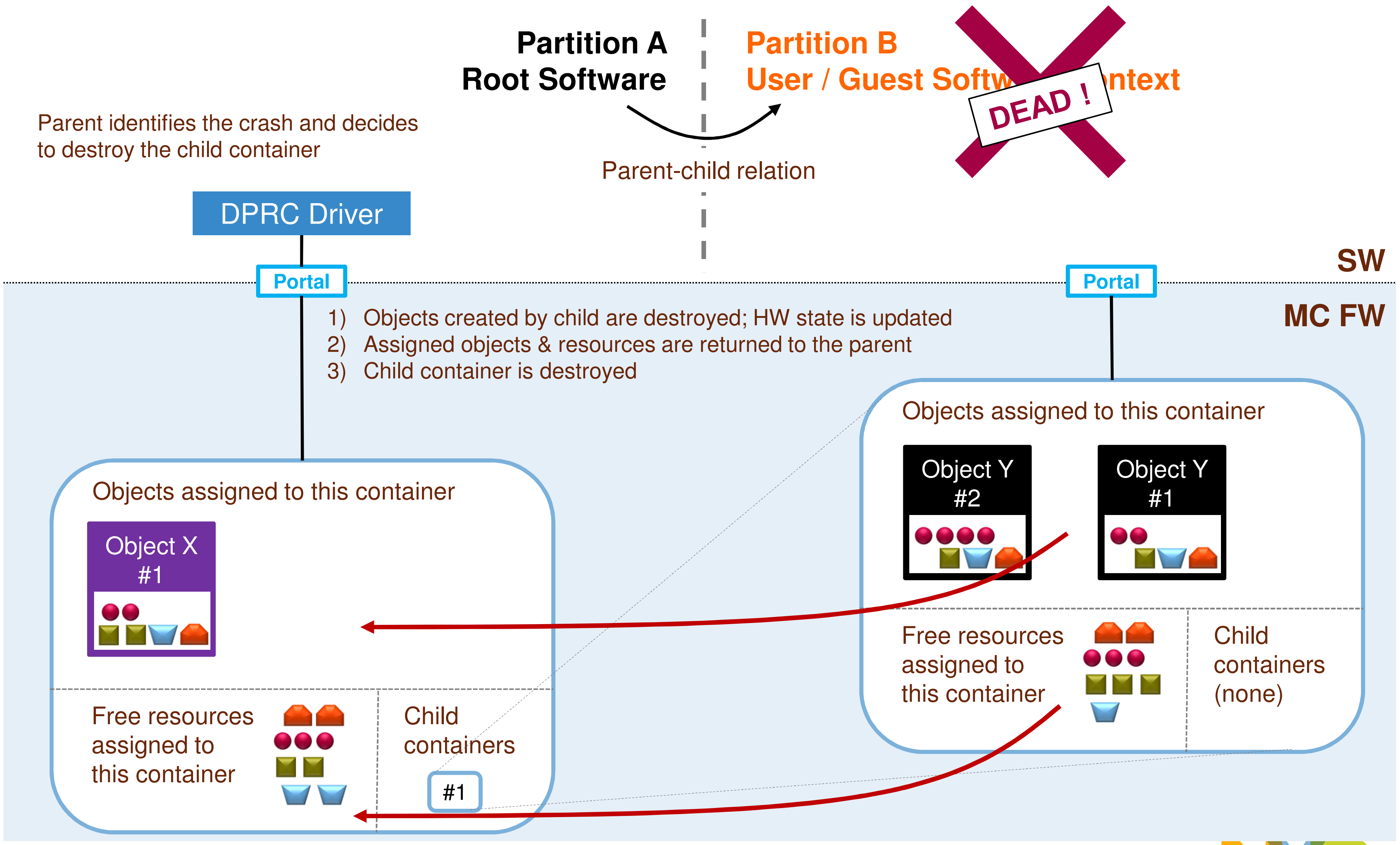
a) Resources are allocated from current container
b) Lacking resources are allocated from parent container by MC, if policy and quota allows; parent software can control the allocation policy and quota per resource type

Resource Management: Assign Objects & Resources



Partition A
Root Software
Partition B
User / Guest Software Context
Parent-child relation
Parent assigns objects and/or resources to child:
a) Assign Object Y#1 to child #1
b) Assign resources to child #1
Child receives notifications on container updates
DPRC Driver
Portal
DPRC Driver
Portal
SW
MC FW
Objects assigned to this container
Object X #1
Free resources assigned to this container
Child containers
#1
(a)
(b)
Objects assigned to this container
Object Y #2
Object Y #1
Free resources assigned to this container
Child containers (none)

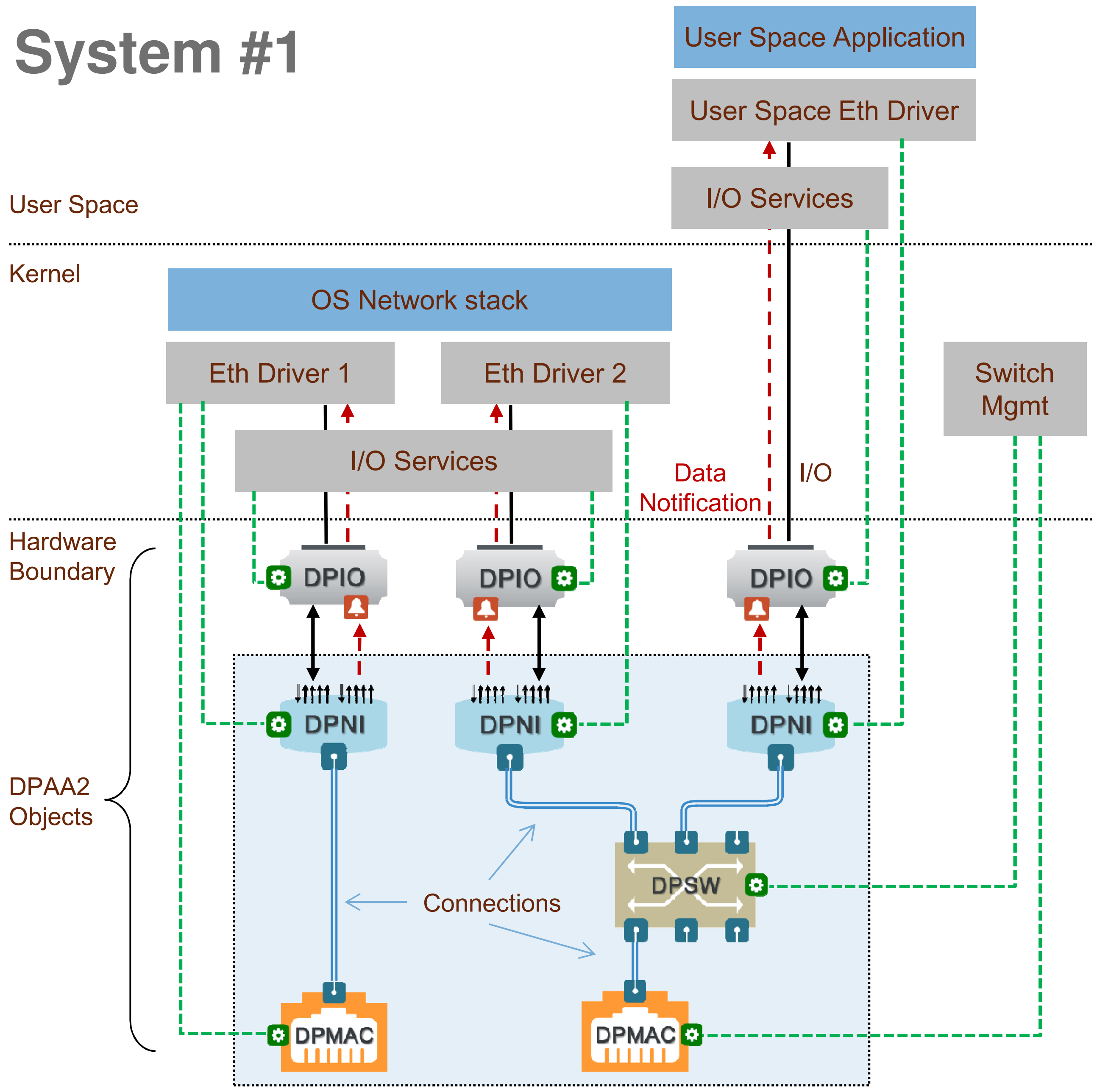
Resource Management: Resource Cleanup


Partition A
Root Software
Partition B
User / Guest Softw ntext
DEAD !
Parent identifies the crash and decides to destroy the child container
Parent-child relation
DPRC Driver
Portal
Portal
SW
MC FW
1) Objects created by child are destroyed; HW state is updated
2) Assigned objects & resources are returned to the parent
3) Child container is destroyed
Objects assigned to this container
Object X #1
Free resources assigned to this container
Child containers
#1
Objects assigned to this container
Object Y #2
Object Y #1
Free resources assigned to this container
Child containers (none)

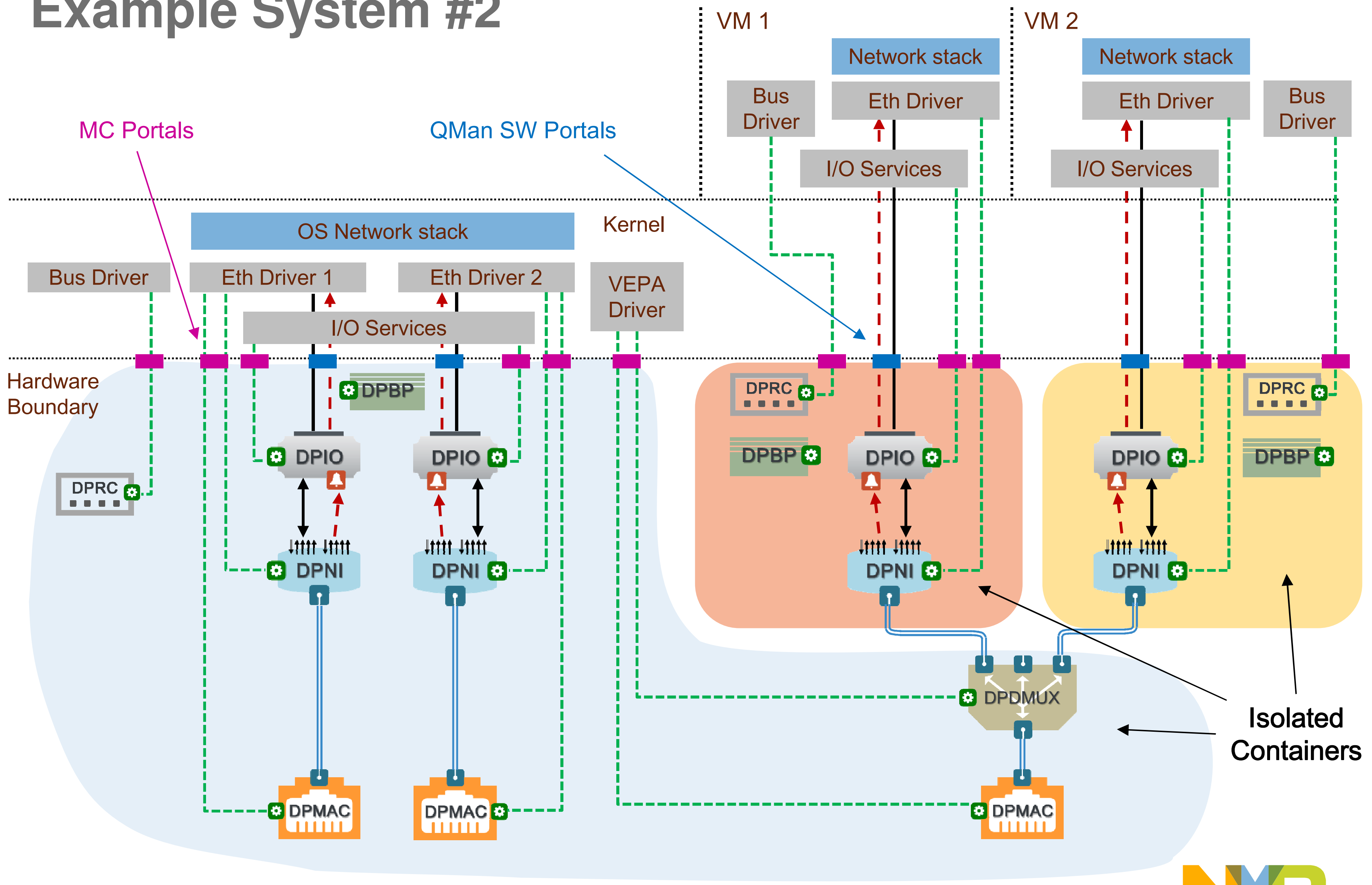
CREATE YOUR NETWORK-ON-CHIP

As easy as

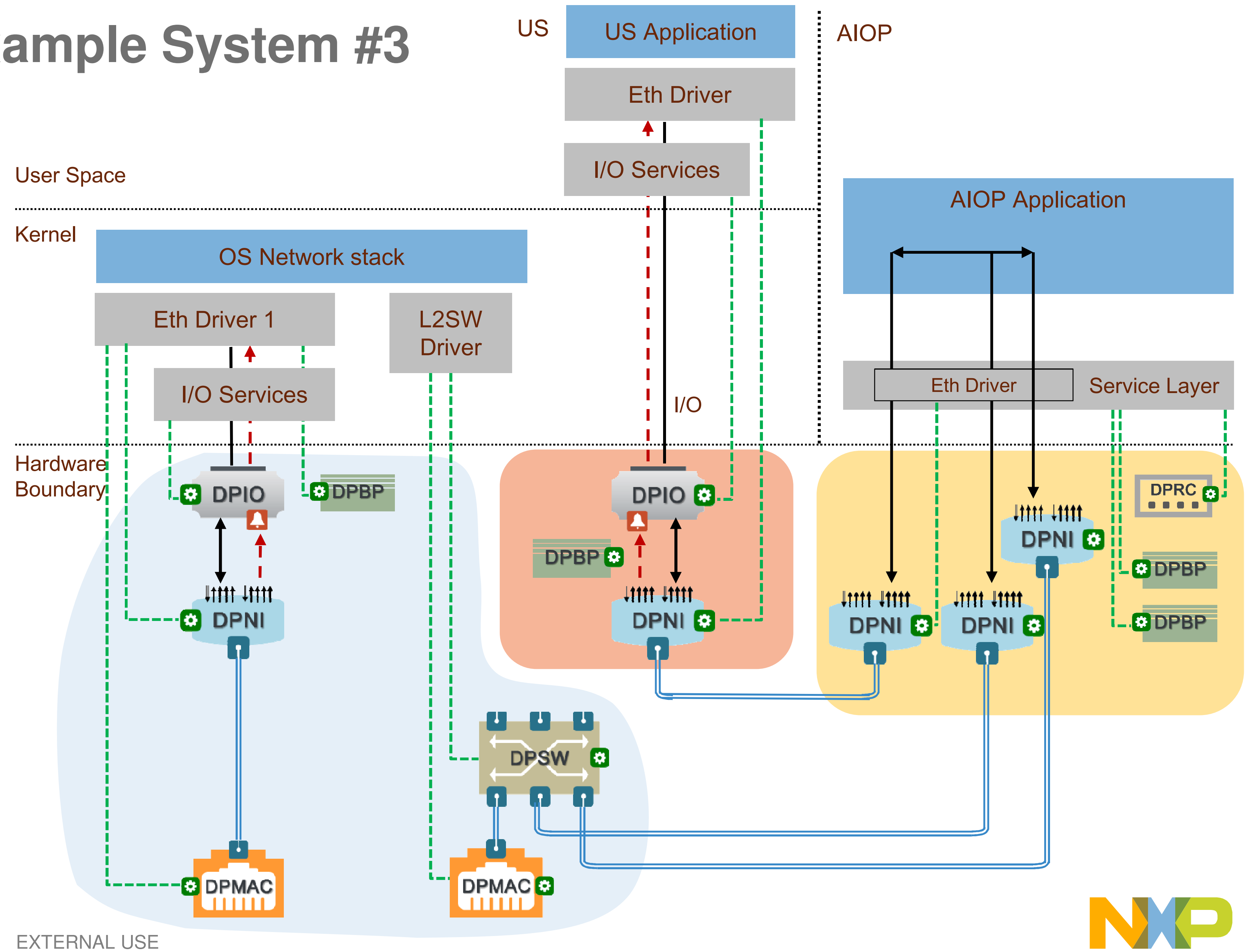
Example System #1



Example System #2



Example System #3

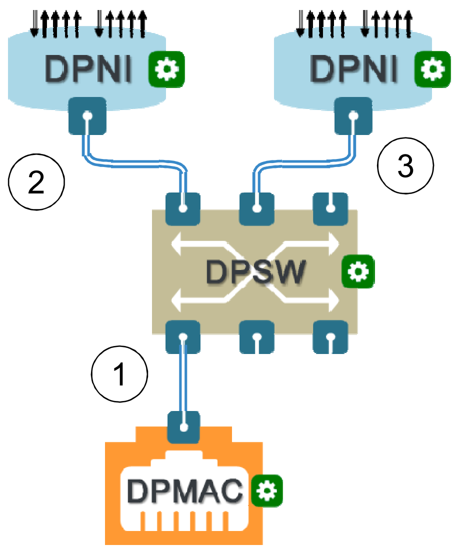


Connecting Network Objects

Connections can be declared in the DPL file:

```
connections {
        connection@1{
                endpoint1 = "dpsw@0/if@0";
                endpoint2 = "dpmac@0";
        };
        connection@2{
                endpoint1 = "dpsw@0/if@5";
                endpoint2 = "dpni@1";
                max_rate = 1000;
        };
        connection@3{
                endpoint1 = "dpsw@0/if@4";
                endpoint2 = "dpni@2";
                max_rate = 1000;
        };
};
```

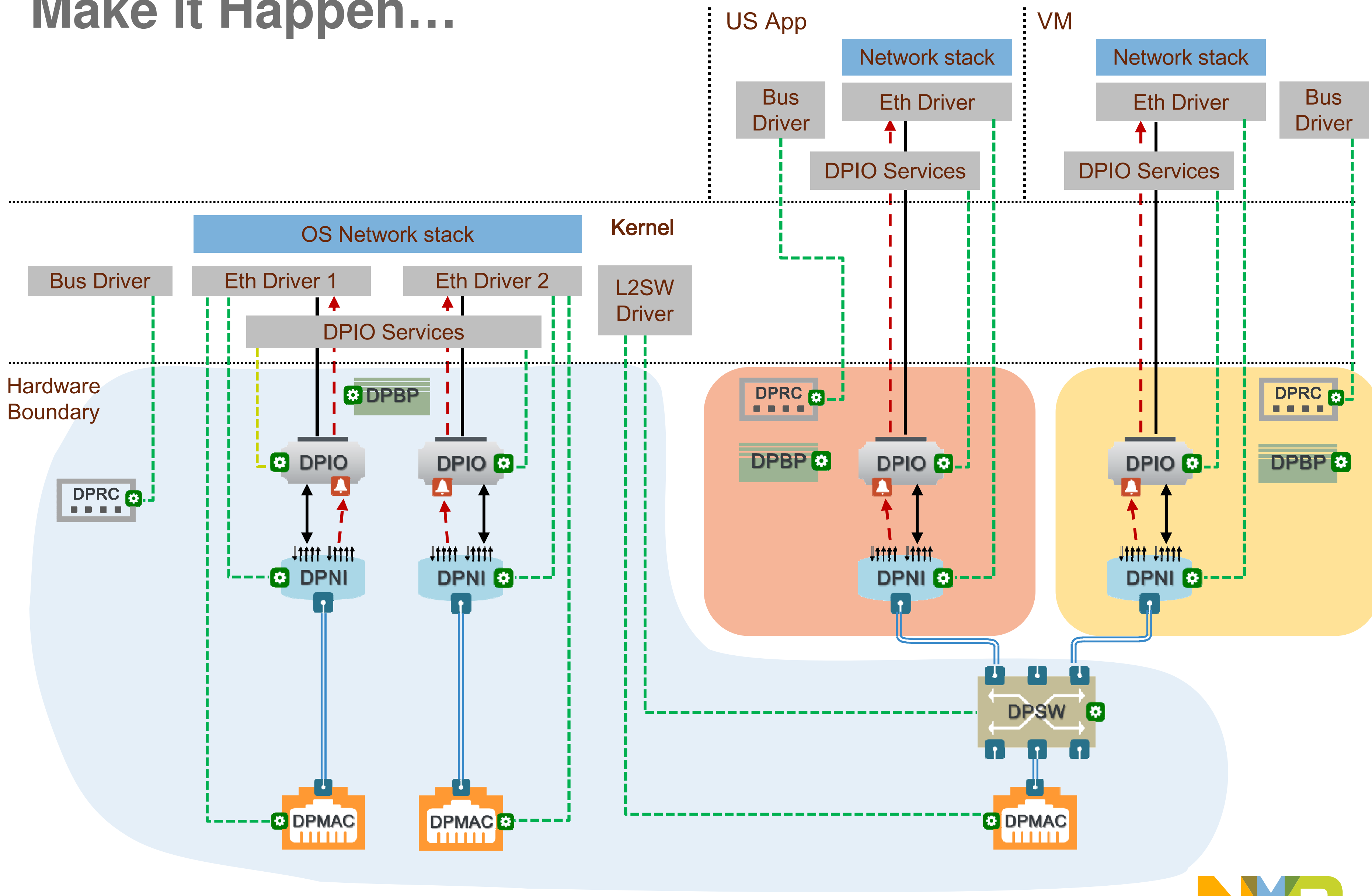
Network Topology View



…or created dynamically via DPRC API:

```
1) dprc_connect(dprc, <dpsw-0/0>, <dpmac-0>, NULL)

2) dprc_connect(dprc, <dpni-1>, <dpsw-0/5>, <rate_cfg>)

3) dprc_connect(dprc, <dpni-2>, <dpsw-0/4>, <rate_cfg>)
```

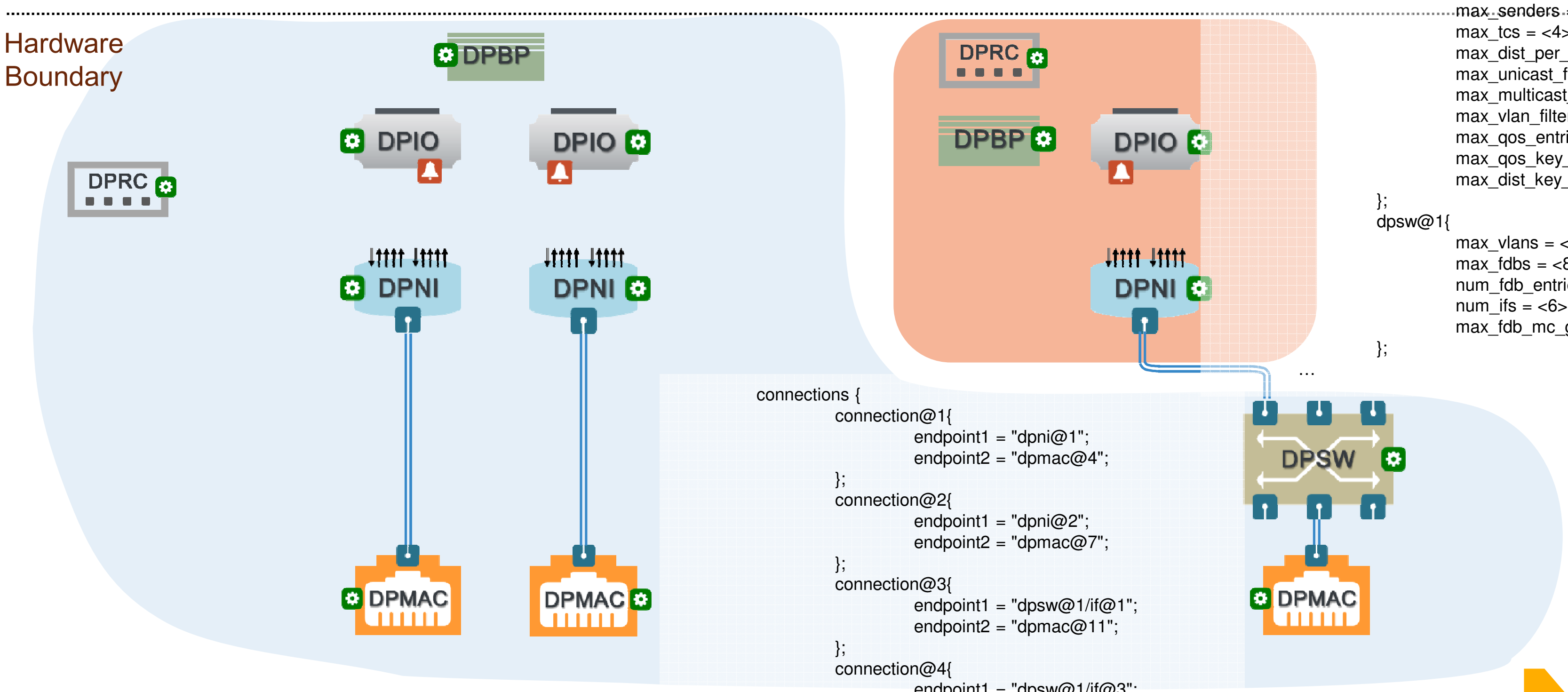
Make it Happen…



Declare Initial Objects… (DPL)

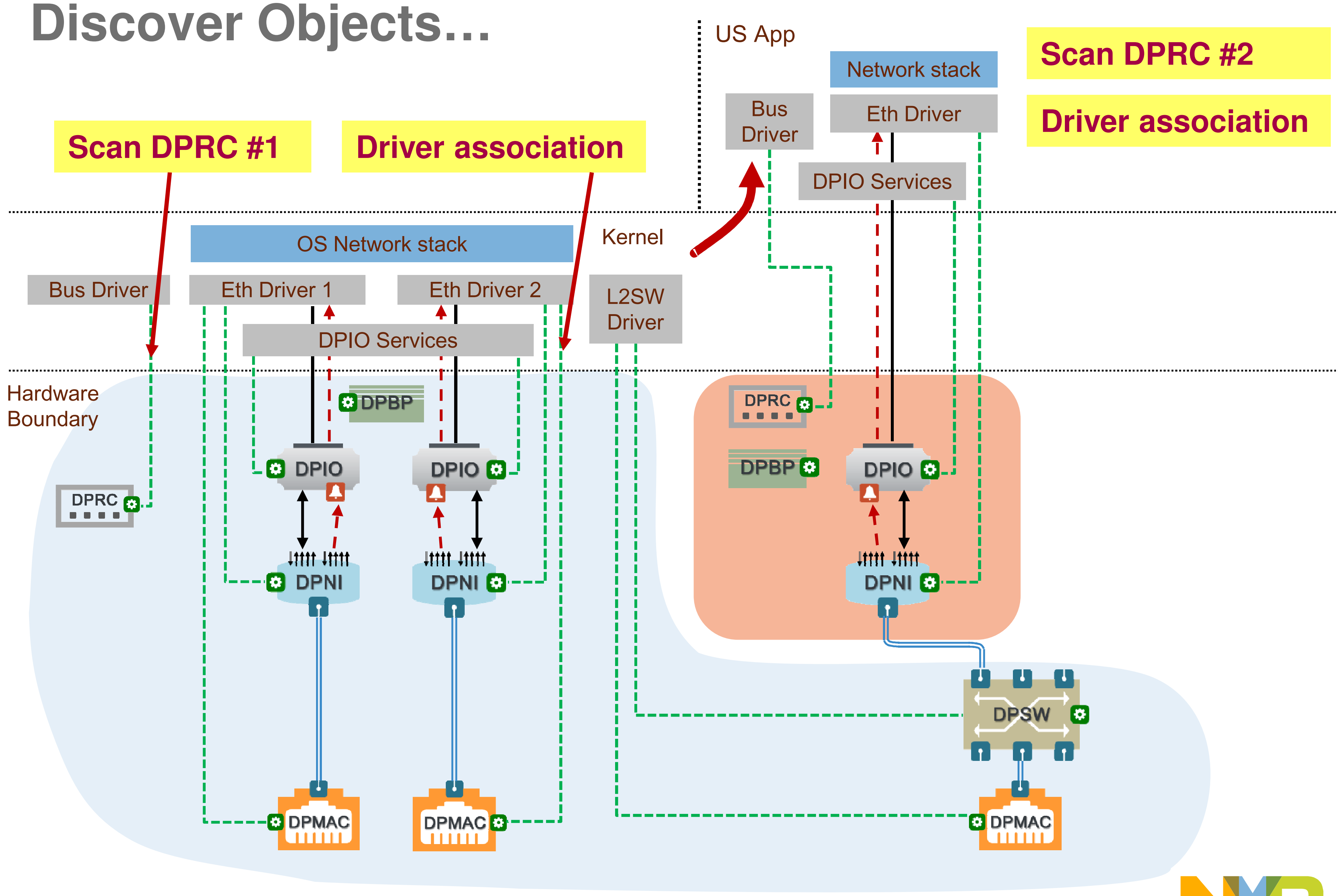
```
containers {
        dprc@1 {
                parent = "none";
                icid = <10>;
                options = "DPRC_CFG_OPT_SPAWN_ALLOWED" , "DPRC_CFG_OPT_ALLOC_ALLOWED";
                objects {
                        obj@1{  obj_name = "dpni@1";     };
                        obj@1{  obj_name = "dpni@2";     };
                        obj@2{  obj_name = "dpbp@1";     };
                        obj@3{  obj_name = "dpio@1";     };
                        obj@3{  obj_name = "dpio@2";     };
                        obj@4{  obj_name = "dpsw@1";     };
                        obj@9{  obj_name = "dpmac@4";  };
                        obj@10{ obj_name = "dpmac@7";  };
                        obj@10{    obj_name = "dpmac@11";          };
                };
        };
        dprc@2 {
                parent = "dprc@1";
                icid = <20>;
                options = "DPRC_CFG_OPT_SPAWN_ALLOWED" , "DPRC_CFG_OPT_ALLOC_ALLOWED";
                objects {
                        obj@1{  obj_name = "dpni@3";     };
                        obj@2{  obj_name = "dpbp@3";     };
                        obj@3{  obj_name = "dpio@3";     };
                };
        };
};
```

```
objects {
        dpio@1 {
                channel_mode = "DPIO_LOCAL_CHANN
                num_priorities = <8>;
        };
        dpio@2 {
                channel_mode = "DPIO_LOCAL_CHANN
                num_priorities = <4>;
        };
        dpni@1{
                mac_addr = <4 5 6 7 8 9>;
                max_senders = <1>;
                max_tcs = <4>;
                max_dist_per_tc = <2 2 4 4>;
                max_unicast_filters = <8>;
                max_multicast_filters = <16>;
                max_vlan_filters = <8>;
                max_qos_entries = <32>;
                max_qos_key_size = <24>;
                max_dist_key_size = <24>;
        };
        dpni@2{
                mac_addr = <4 5 6 7 1 2>;
                max_senders = <1>;
                max_tcs = <4>;
                max_dist_per_tc = <2 2 4 4>;
                max_unicast_filters = <8>;
                max_multicast_filters = <16>;
                max_vlan_filters = <8>;
                max_qos_entries = <32>;
                max_qos_key_size = <24>;
                max_dist_key_size = <24>;
        };
        dpsw@1{
                max_vlans = <16>;
                max_fdbs = <8>;
                num_fdb_entries = <64>;
                num_ifs = <6>;
                max_fdb_mc_groups = <16>;
        };
        …
```

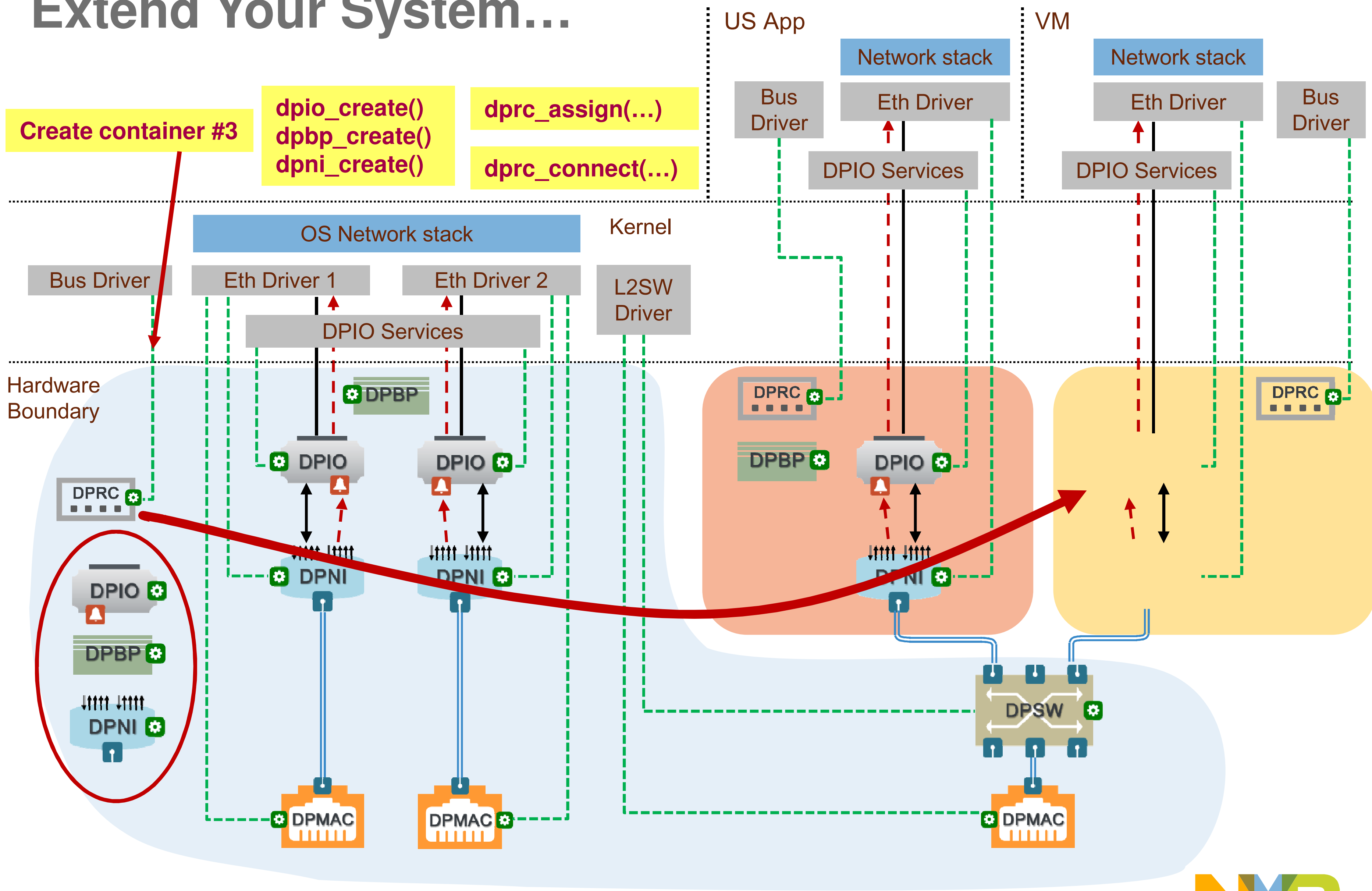


```
connections {
        connection@1{
                endpoint1 = "dpni@1";
                endpoint2 = "dpmac@4";
        };
        connection@2{
                endpoint1 = "dpni@2";
                endpoint2 = "dpmac@7";
        };
        connection@3{
                endpoint1 = "dpsw@1/if@1";
                endpoint2 = "dpmac@11";
        };
        connection@4{
                endpoint1 = "dpsw@1/if@3";
                endpoint2 = "dpni@3";
        };
};
```

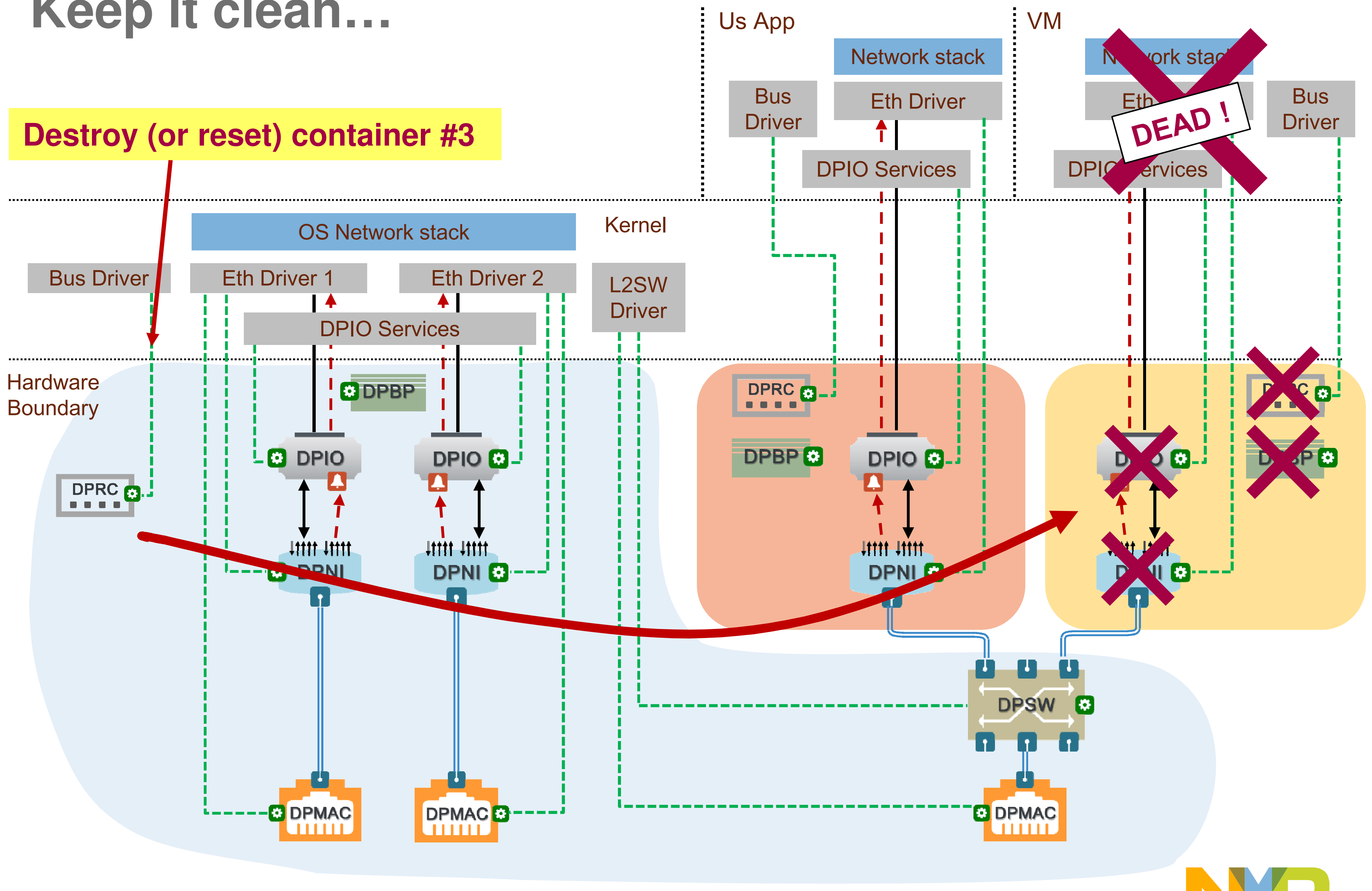
Discover Objects…



Extend Your System…



Keep it clean…



DPAA2 BOOT SEQUENCE

DPAA2 Boot Sequence

GPP

- U-Boot allocates memory for MC, verifies and loads MC image to memory
- U-Boot loads DPC and DPL to memory
- U-Boot kicks MC core #0 and waits for status
- U-Boot loads root software image and transfers control

- Root software performs its own boot sequence
- Root software scans its container and activates relevant drivers

MC

- Reads DPC
- Initializes MC platform (memory, caches, MPIC, DMA, timers, etc.)
- Initializes services (command interface, resource manager, etc.)
- Initializes all DPAA controllers
- Initializes drivers for logical objects
- Parses the DPL and creates all listed containers and objects
- Reports initialization status to boot program

- Responds to GPP queries and commands

AIOP

AIOP Boot

GPP

- GPP software creates the AIOP container and assigns required objects into it
- GPP software places AIOP image in system DDR
- GPP software sends an “AIOP load” command to MC, specifying AIOP tile id and image location

- GPP software sends an “AIOP run” command to MC, specifying AIOP tile id and selected cores

MC

- Allocates required memory as specified by AIOP image parameters
- Initializes AIOP registers and accelerators (OSM, WS, CTLU, MFLU, STE, TMAN, FDMA, CDMA, …)
- Loads AIOP image to destined memory

- Initializes MC-AIOP shared structure
- Enables the selected set of AIOP cores

AIOP

- Runs Service Layer’s boot
- Service Layer Scans the AIOP container through MC
- Marks boot completion status

MC Makes it Easy

- ✓ Presents hardware as logical objects
- ✓ Virtualizes and isolates objects
- ✓ Hides complex sequences
- ✓ Sets up a Network-on-Chip
- ✓ Manages resources
- ✓ Supports recovery scenarios

SECURE CONNECTIONS
FOR A SMARTER WORLD