# Overview of QorIQ Processor Solutions for **Virtualization Features** on T2080-based iNIC

## EUF-NET-T0967

Peter Van Ackeren

J A N . 2 0 1 5

External Use

# Session Introduction

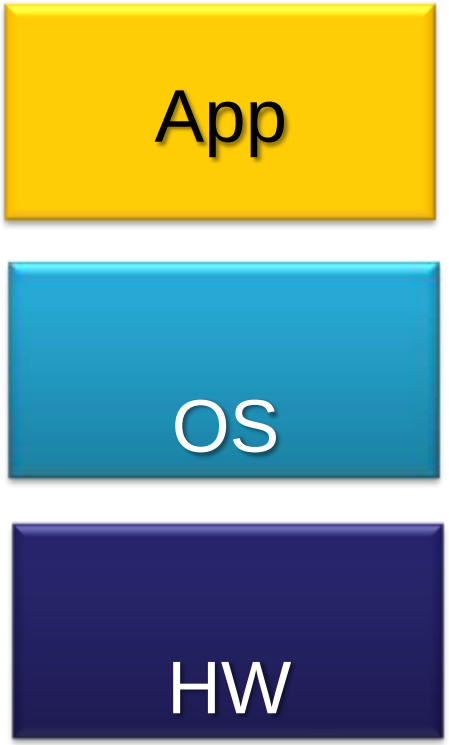
- This session explores the virtualization solutions available for on QorIQ communication processors

- After completing this session you will have become familiar with :
  - General virtualization principles and use cases
  - The features, benefits and some details of KVM/QEMU, Linux® containers and libvirt implementations as supplied in the QorIQ Linux SDK
  - Architecture specific details of Power Architecture and ARM® based platforms as applicable to virtualization use cases

# Introduction

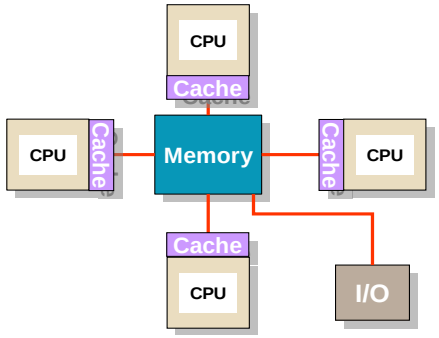
# Traditional System Model

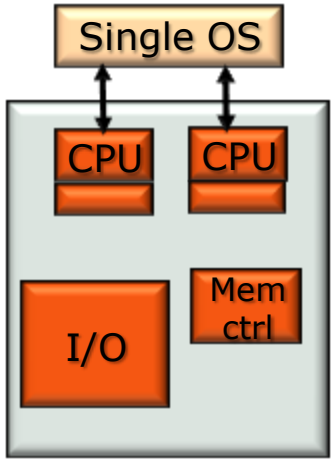
- Single OS runs on a single or multi-core hardware system



# Multi-processing Usage Models
## SMP

- SMP = Symmetrical Multi-processing
  - multi-processor homogeneous computer architecture
  - where two or more identical processors are connected to globally shared main memory and I/O
  - processors could be separate devices, multiple cores in a single device or a mix
  - a single OS instance runs on and manages the cores, with shared memory, I/O and interrupt resources
  - Each processor can run independent processes and threads
  - Any idle processor can be assigned any task



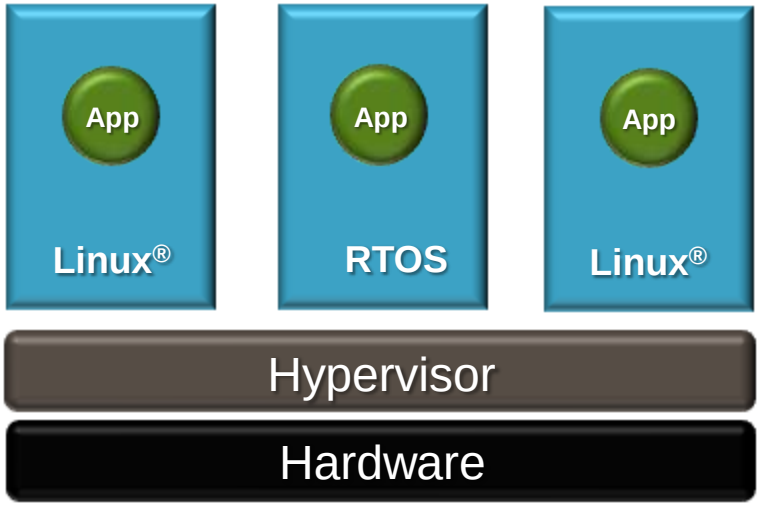
# Multi-processing Usage Models
## AMP

- AMP = Asymmetrical Multi-processing
  - Individual CPUs are dedicated to particular tasks, like running an OS
  - different software instances run on separate CPUs
    - e.g. 2 copies of Linux image are loaded at different physical address locations

- Both CPUs must cooperate to share the resources
  - Neither OS can own the whole system
  - I/O and interrupts are separated
  - Static configuration of resources

# What is Virtualization ?

- Virtualization – an abstraction layer that enables running multiple operating systems on a single system, implemented using hardware and software technologies




- A hypervisor is a software component that creates and manages virtual machines which can run operating systems.

# Types of Virtualization

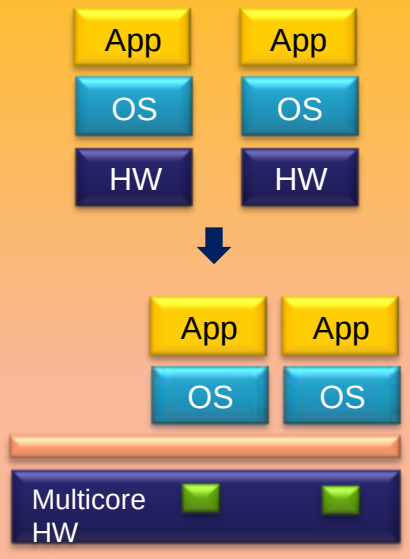
- Full Virtualization:
  - Virtual machine behaves identically to physical hardware
  - OS code unchanged
  - Advantage: no changes to the GOS
  - Disadvantage: can be lower in over-all performance
  - Common examples: KVM, VirtualBox

- Para-virtualization:
  - Guest is hypervisor-aware and uses defined API for some services
  - Guest OS is modified
  - Advantage: potentially better performance
  - Disadvantage: modifications to Guest OS (e.g. drivers)
  - Examples: Freescale Embedded HV

# Virtualization Use Cases

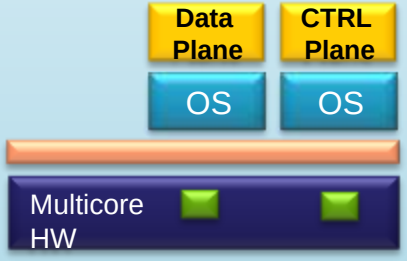
## Cost Reduction

### Consolidation

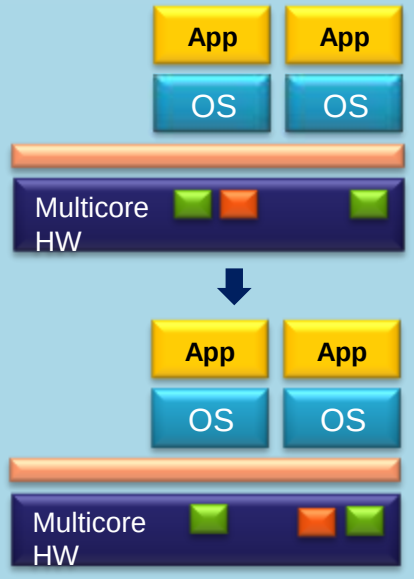



## Flexibility and Scalability

### Split Workload

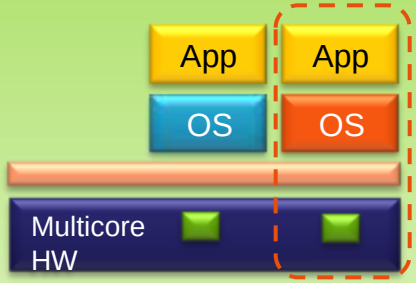



### Dynamic Resource Management

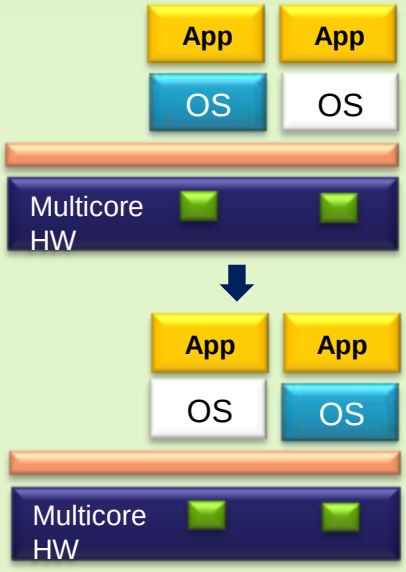



## Reliability and Protection

### Sandboxing

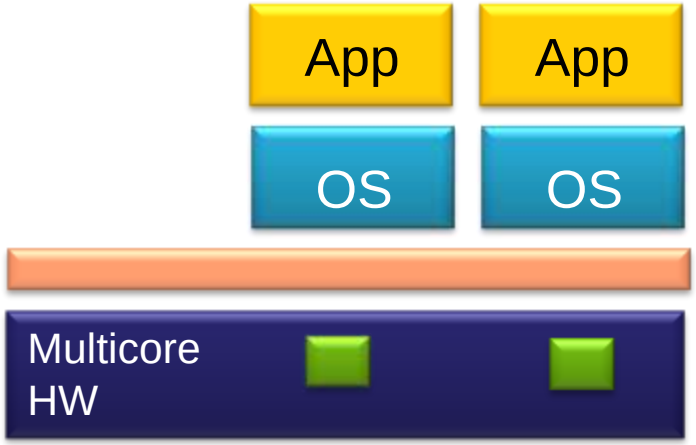



### Failover



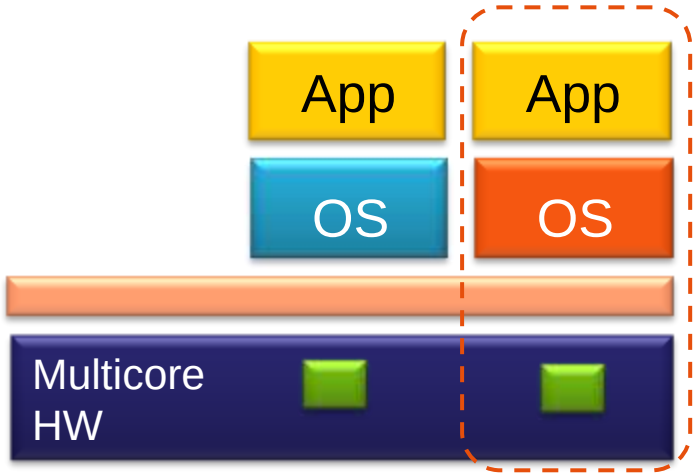
# Virtualization Use Cases – Cost Reduction

- Consolidation :
  - redeploy multiple discrete systems/domains onto a single multi-core processor
  - Benefits:
    - Cost effective : bill-of-material, power
    - Preserve investment : software re-use
    - Improved hardware utilization
    - Flexibility

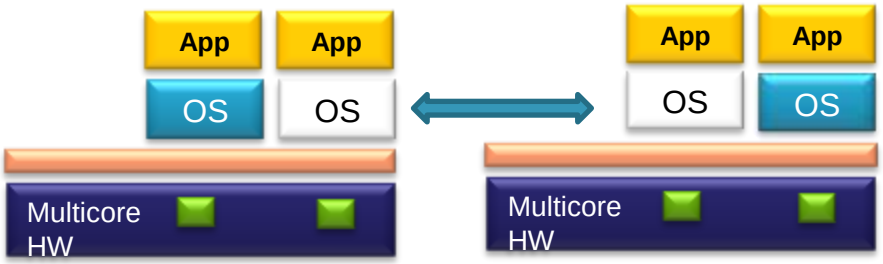


# Virtualization Use Cases – Reliability & Protection

- Sandboxing :
  - Add untrusted software to a system, e.g. operator applications
  - Run GPL based software in isolation
  - Run test software safely
  - Isolate security-sensitive tasks : access rights control, rule definitions, key management, …

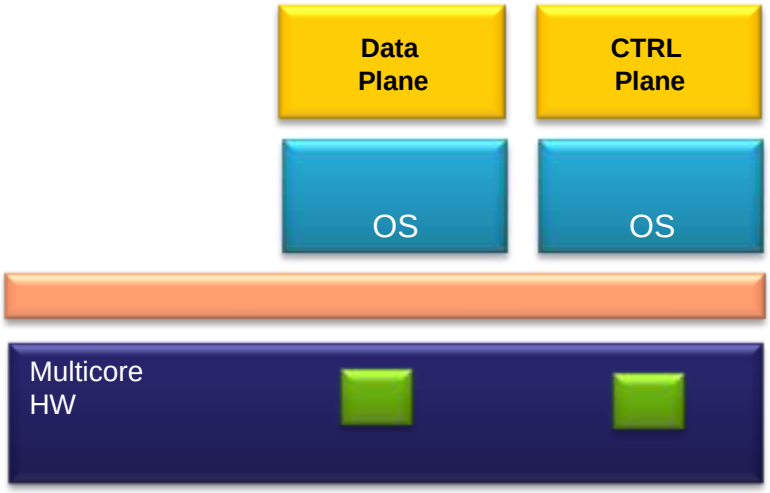



- High availability
  - active/standby configuration without additional hardware



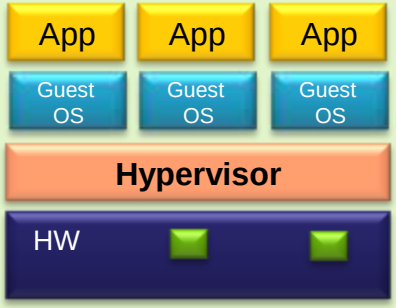
# Virtualization Use Cases – Flexibility & Scalability

- Run legacy software / OS on Linux
- Add functionality to existing system by dropping in a VM
- Use different versions of the Linux kernel
- Better resource management
  - Allocation of physical CPUs / control CPU load
  - Create/destroy VMs as needed
- Maps well to split workload
  - e.g. control plane, data plane
- In-service upgrade

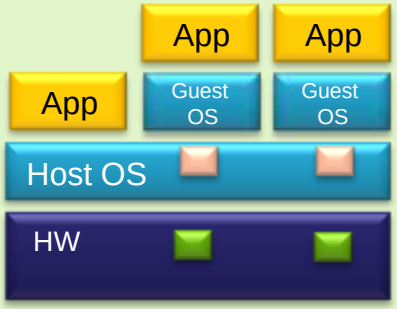


# Virtualization & Hypervisors

- Virtualization – Hardware and software technologies that provide an abstraction layer that enables running multiple operating systems on a single computer system
- Hypervisor - Software component that creates and manages virtual machines which can run guest operating systems

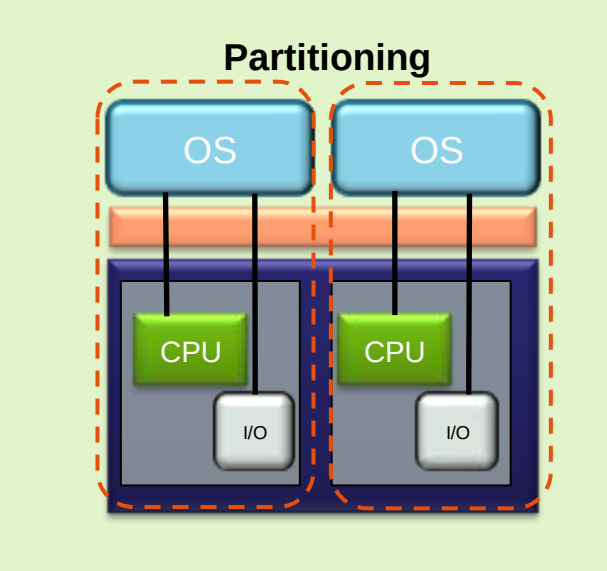


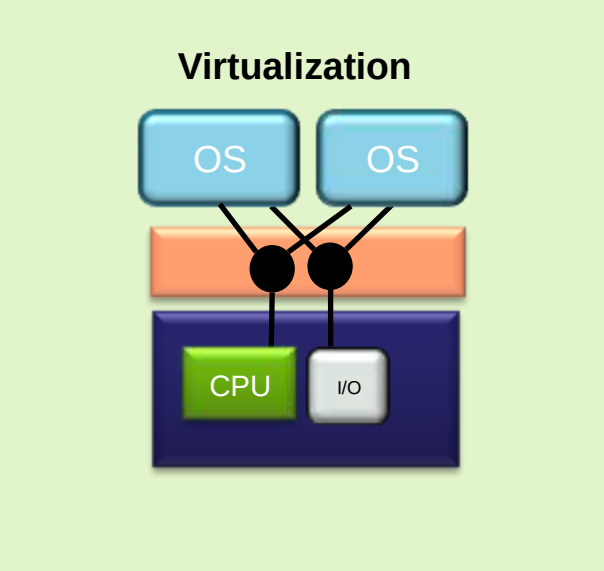

- Hypervisor runs “bare metal”

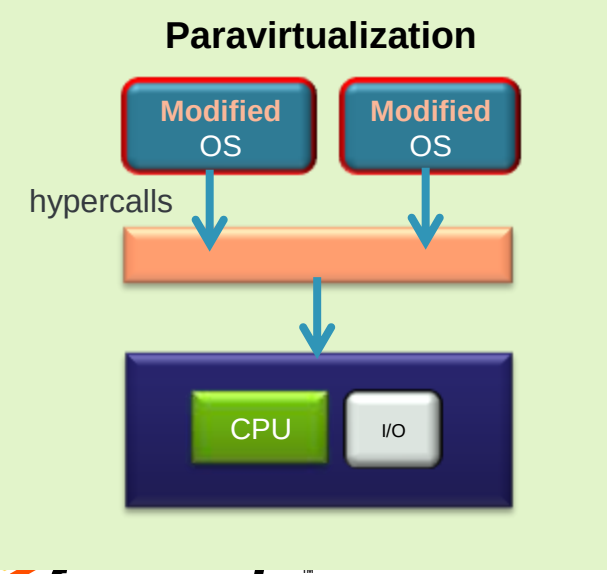


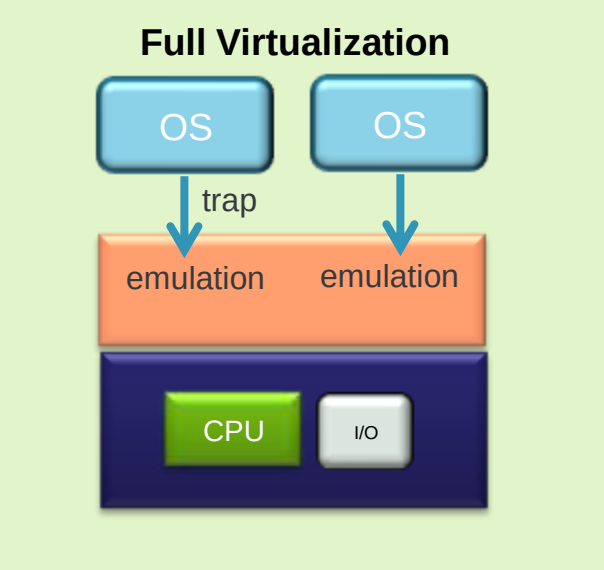

- Hypervisor is integrated in Host OS
  - Reuses OS infrastructure
- Host OS runs other applications

# Virtualization & Partitioning



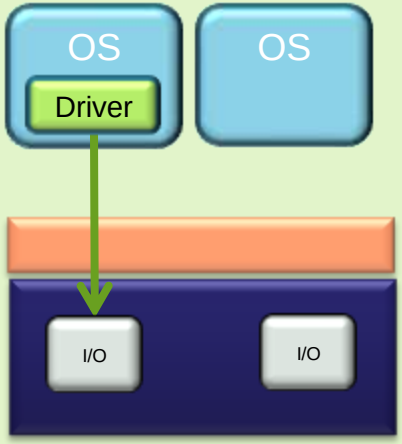



# Device Usage in Virtual Environments

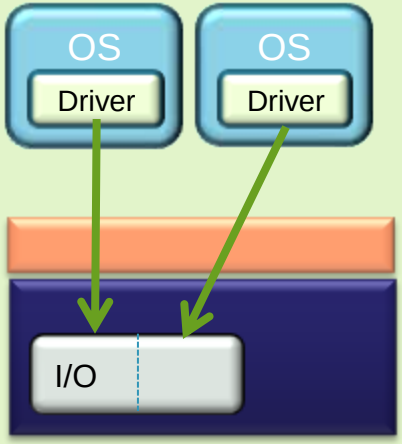
## Direct Access

- Best native performance
- Direct access to hardware



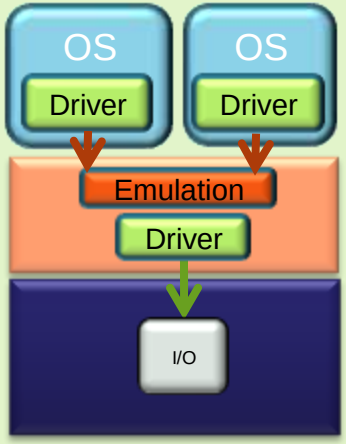

## Partitionable HW

- Hardware partitioned
- One hardware block



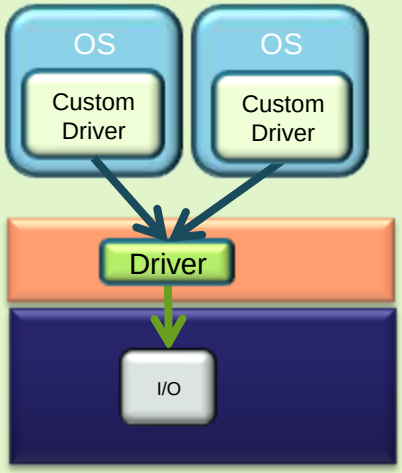

## Emulated

- Driver in Hypervisor
- Emulation in Hypervisor
- Unmodified Drivers in Guest




## Para-Virtualized

- Driver in Hypervisor
- Modified Drivers in Guest

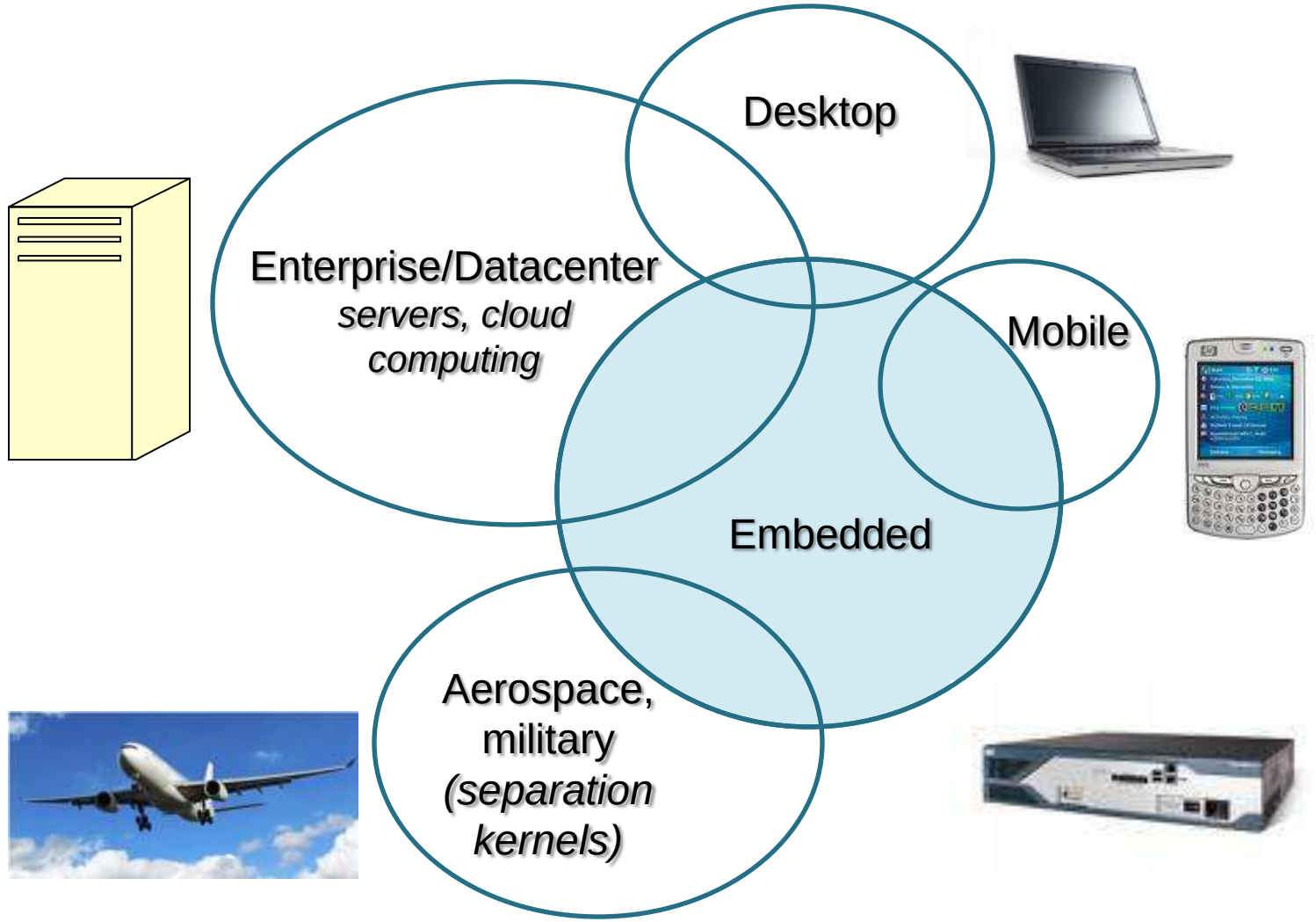



Hardware/software access

Hypercalls

Traps

# Multi-OS Systems

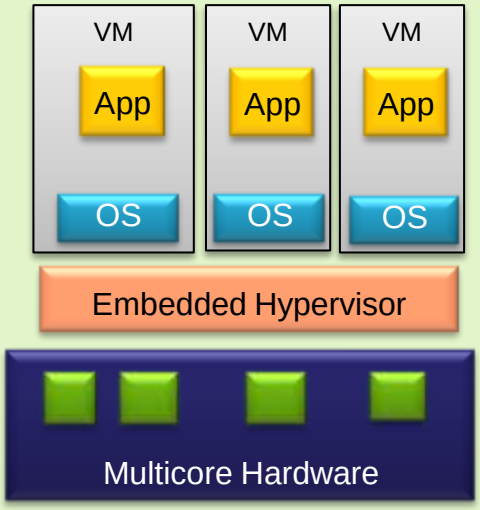


# Virtualization Technologies for QorIQ

# Virtualization Technologies for QorIQ

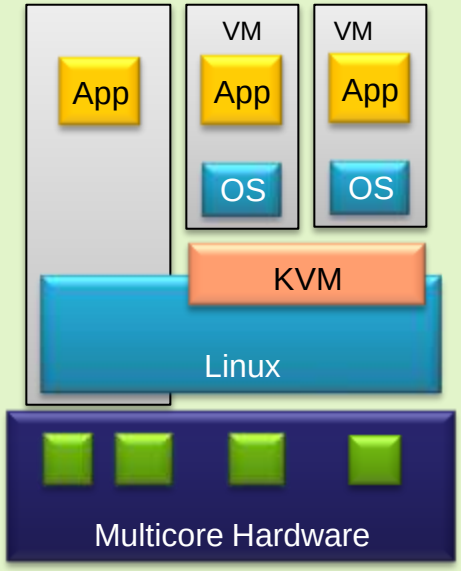
## Freescale Embedded Hypervisor

- Lightweight Hypervisor
- Resource Partitioning
- Para-Virtualization
- Failover support
- 3rd Party OSs



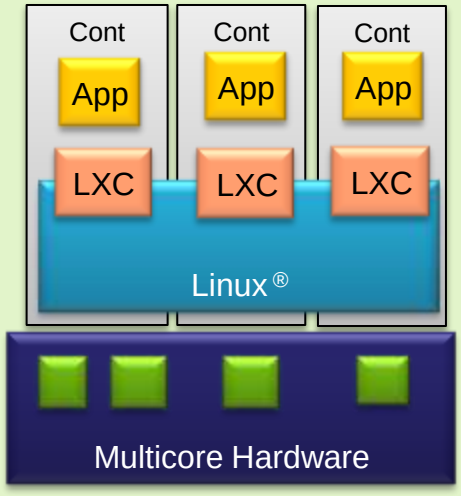

## KVM

- Linux® Hypervisor
- Resource Virtualization
- Resource Oversubscription
- 3rd Party OSs



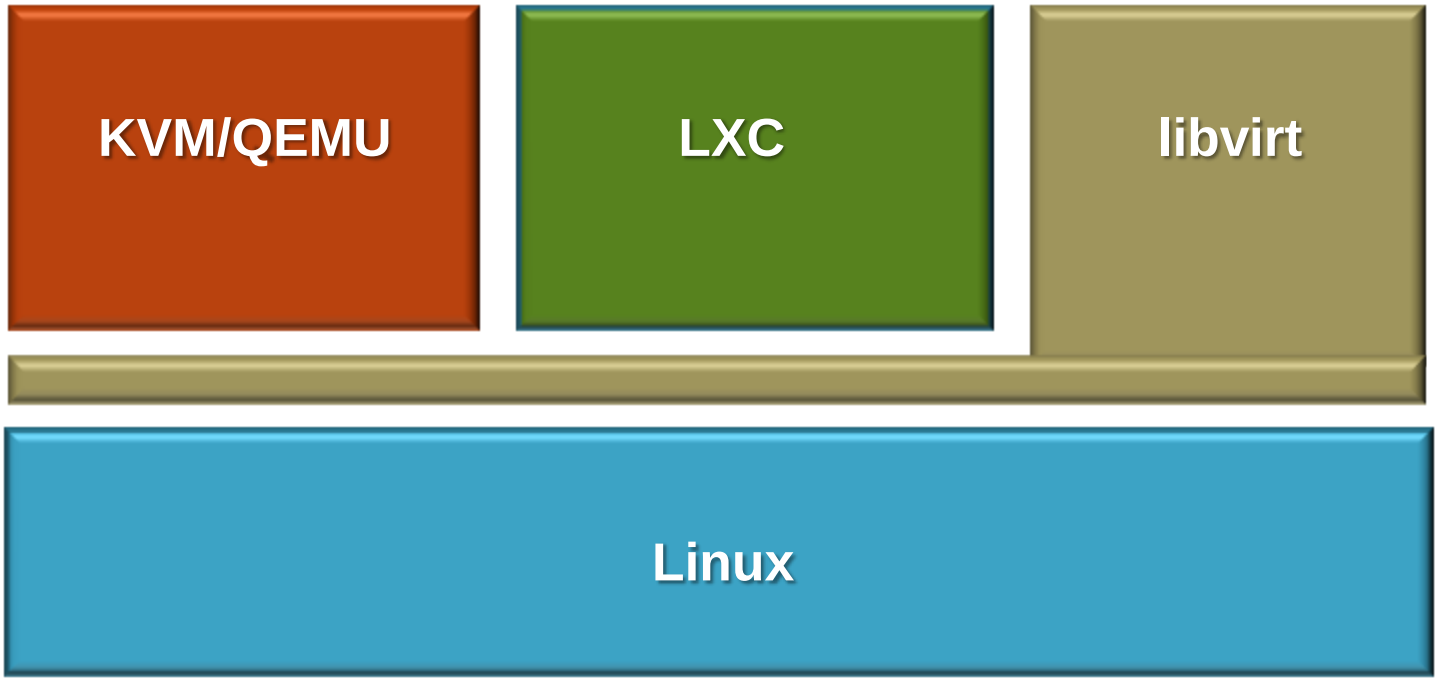

## OS Virtualization (LXC)

- Low Overhead
- Isolation and Resource Control in Linux®
- Decreased Isolation (Kernel sharing)

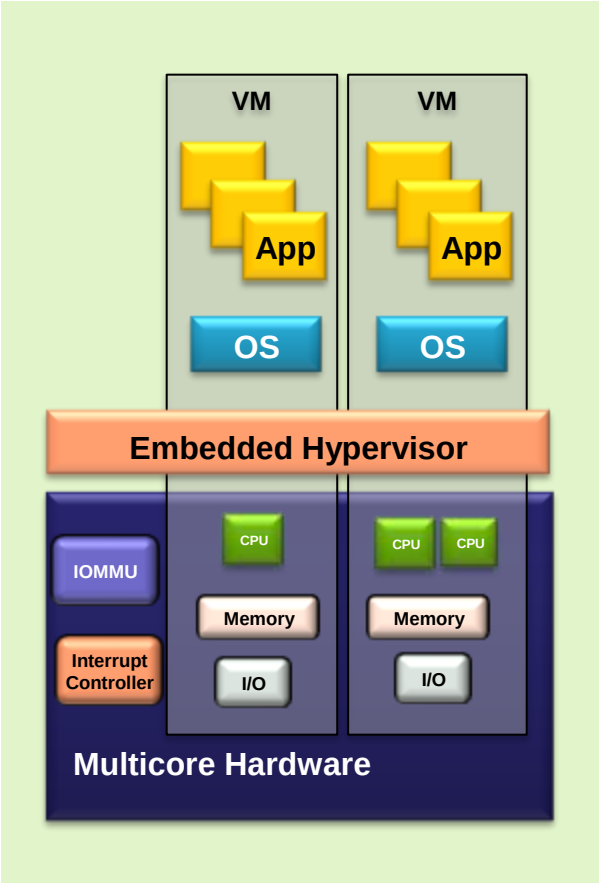


# Virtualization Technologies for QorIQ

- Linux-based
- Delivered with the QorIQ Linux SDK for Power and ARMv7 architecture based products (ARMv8 : planned 2015)

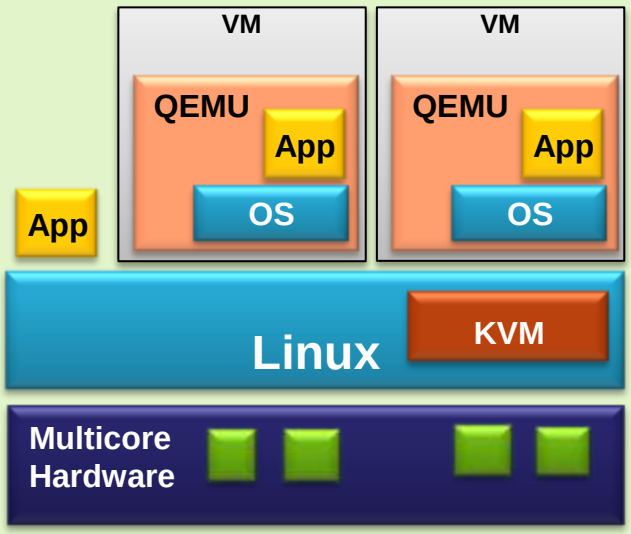


# Freescale Embedded Hypervisor – Overview




- Lightweight hypervisor offering partitioning and isolation
- Only one OS per core
- Combination of full virtualization and para-virtualization
- OS has direct control on high speed peripherals
- Provides good performance and minimal changes to Guest OS

# KVM / QEMU – Overview




- Open source virtualization technology based on the Linux kernel
- Boot operating systems in virtual machines alongside Linux applications
- KVM is a Linux kernel module
- QEMU is a user space emulator that uses KVM for acceleration
- Run virtual machines alongside Linux applications
- No or minimal OS changes required
- Virtual I/O capabilities : virtual disk, network interfaces, serial, etc.
- Direct/pass thru I/O – assign I/O devices to VMs

# KVM / QEMU – Details

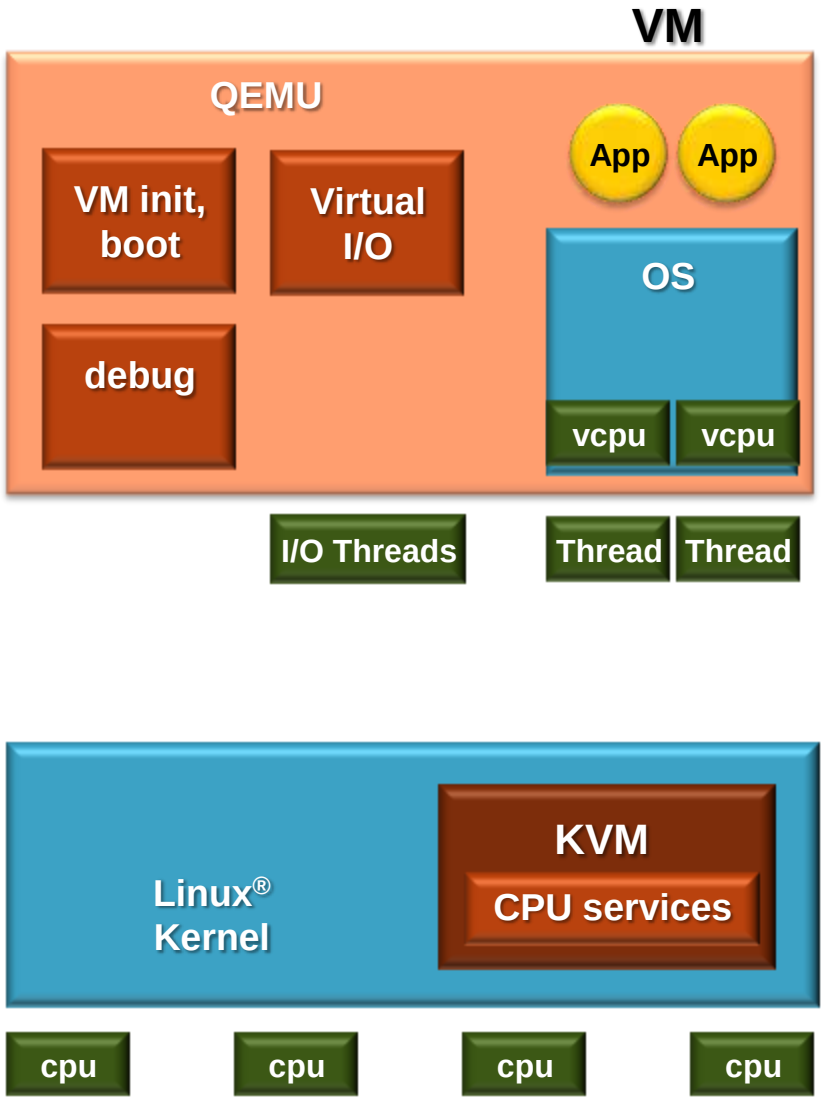
- QEMU is a user space emulator that uses KVM for acceleration
  - Uses dedicated threads for vcpus and I/O
  - KVM leverages hardware virtualization to run guest with higher privileges
  - MPIC virtual chip emulation in kernel
  - I/O
    - Provides dedicated virtio I/O devices and standard drivers in Linux kernel
    - Uses vfio Linux framework to direct assign physical PCI devices
    - Direct notifications between I/O threads and KVM using `eventfds`
    - Vhost provides virtio emulation and I/O thread and in kernel
    - Multi-queue virtio devices connected to multi-queue tap devices
  - Provides services for console, debug, reset, watchdog, etc.

# KVM / QEMU

- QEMU provides …
  - Virtual machine setup and initialization
  - Memory allocation
  - Virtual I/O services
  - Debug stub
- KVM provides …
  - Isolation– using hardware mechanisms and virt extensions
  - Virtual CPU services
  - API used by QEMU
- Linux Kernel provides …
  - Scheduling
  - Memory management

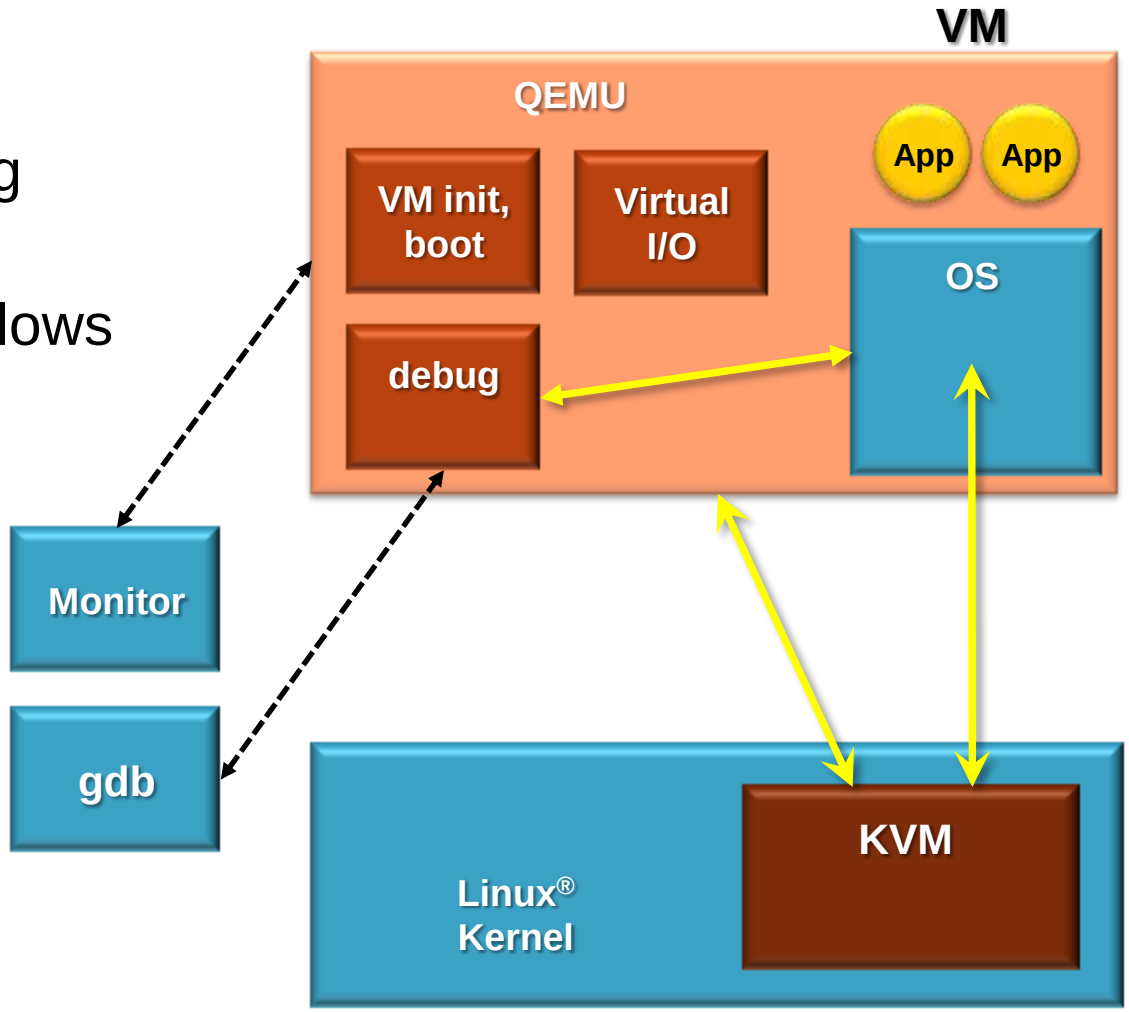
# KVM / QEMU
## Virtual CPUs

- Each vcpu is a Linux thread
  - created by QEMU
- Full capabilities of the Linux scheduler can be used to manage VCPUs/threads
  - CPU affinity
  - priority
  - isolcpus
  - LXC



# KVM / QEMU
## Debugging

- Debug stub in QEMU allows guest debugging using GDB
- QEMU monitor shell allows examining VM state



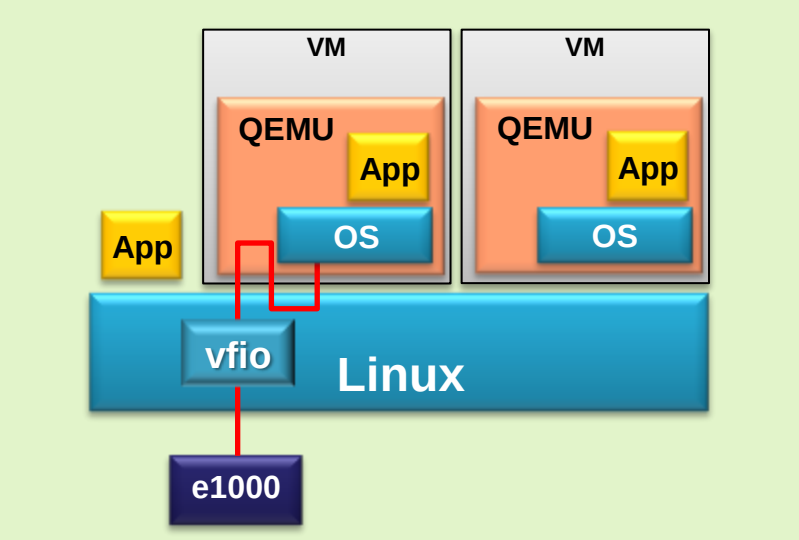
# KVM Status

- KVM on QorIQ P-series and T-series
  - available since QorIQ SDK 1.2 (2012)
  - upstreamed
- KVM on QorIQ Layerscape 1
  - 1st release in QorIQ SDK 1.7 (dec-2014)
- KVM on ARM is available now in upstream Linux since :
  - 32-bit : kernel 3.9
  - 64-bit : kernel 3.11
- KVM and QEMU have an active open source community developing the technology– led by ARM and Linaro

# I/O and KVM

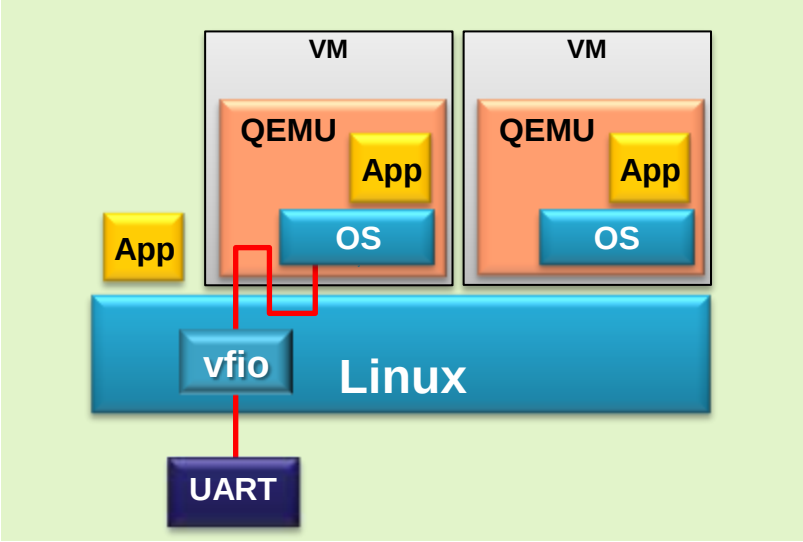
# KVM – I/O Pass Through of PCI Devices

- Assign a physical PCI device to a KVM virtual machine
- Device becomes a private resource of the VM
- OS gets direct access to device registers
- DMA is direct to OS buffers
- QEMU mediates all interrupts from the PCI device
- QEMU presents PCI device on a virtual PCI bus



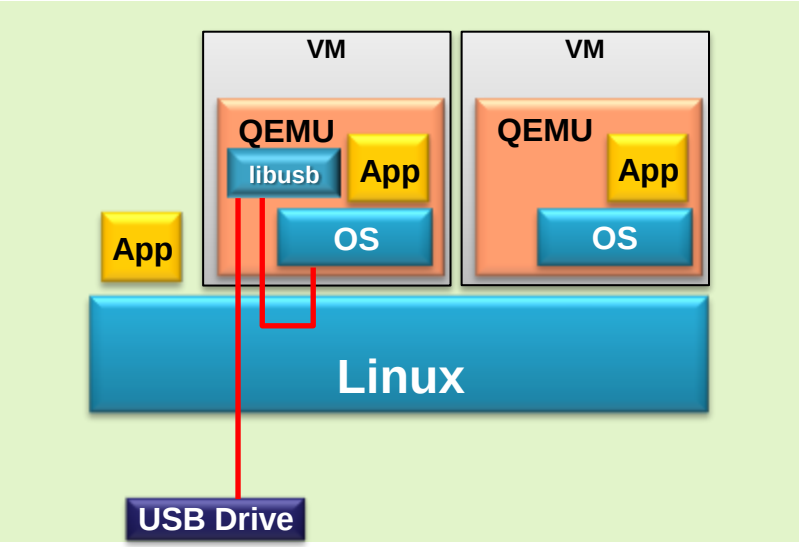
# KVM – I/O Pass Through of Platform Devices

- Assign a physical SoC device (e.g. UART) to a VM
- Device becomes a private resource of the VM
- OS gets direct access to device registers
- DMA is direct to OS buffers
- Guest sees standard device node in its device tree
- QEMU mediates all interrupts from the device



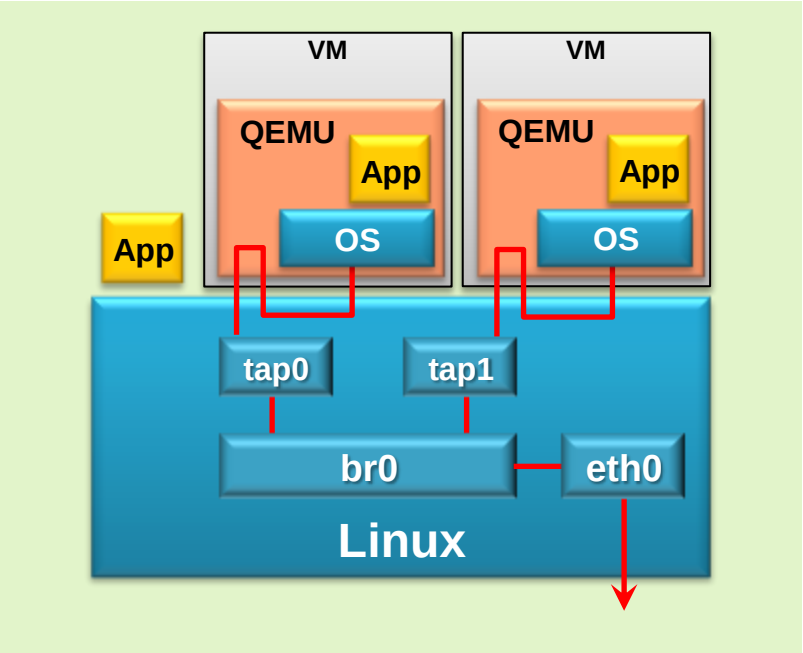
# Pass Through of USB Devices

- Assign a physical USB device or port to a KVM virtual machine
- QEMU mediates device access
- Guest sees a virtual USB controller on a virtual PCI bus



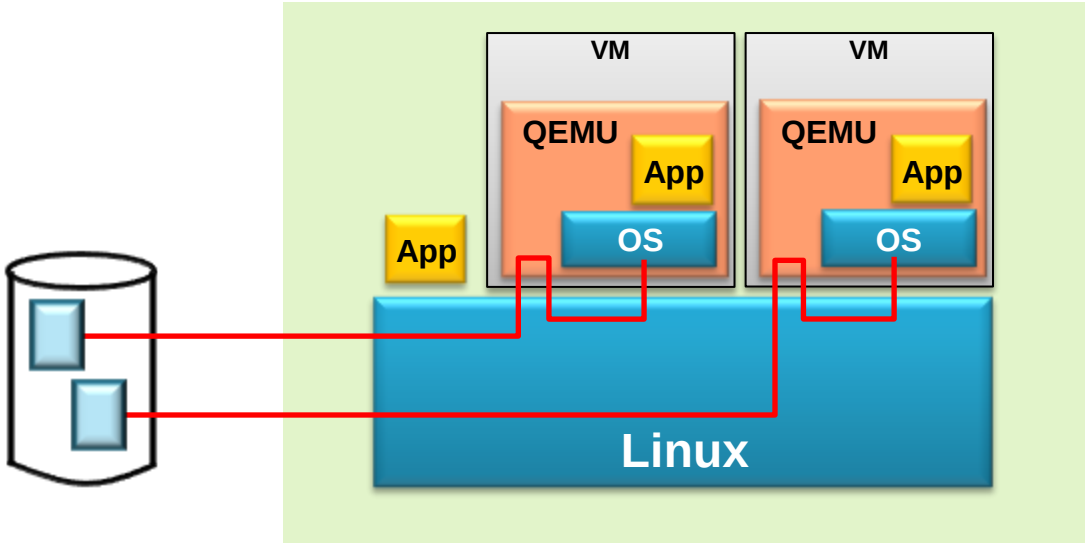
# Virtio Networking

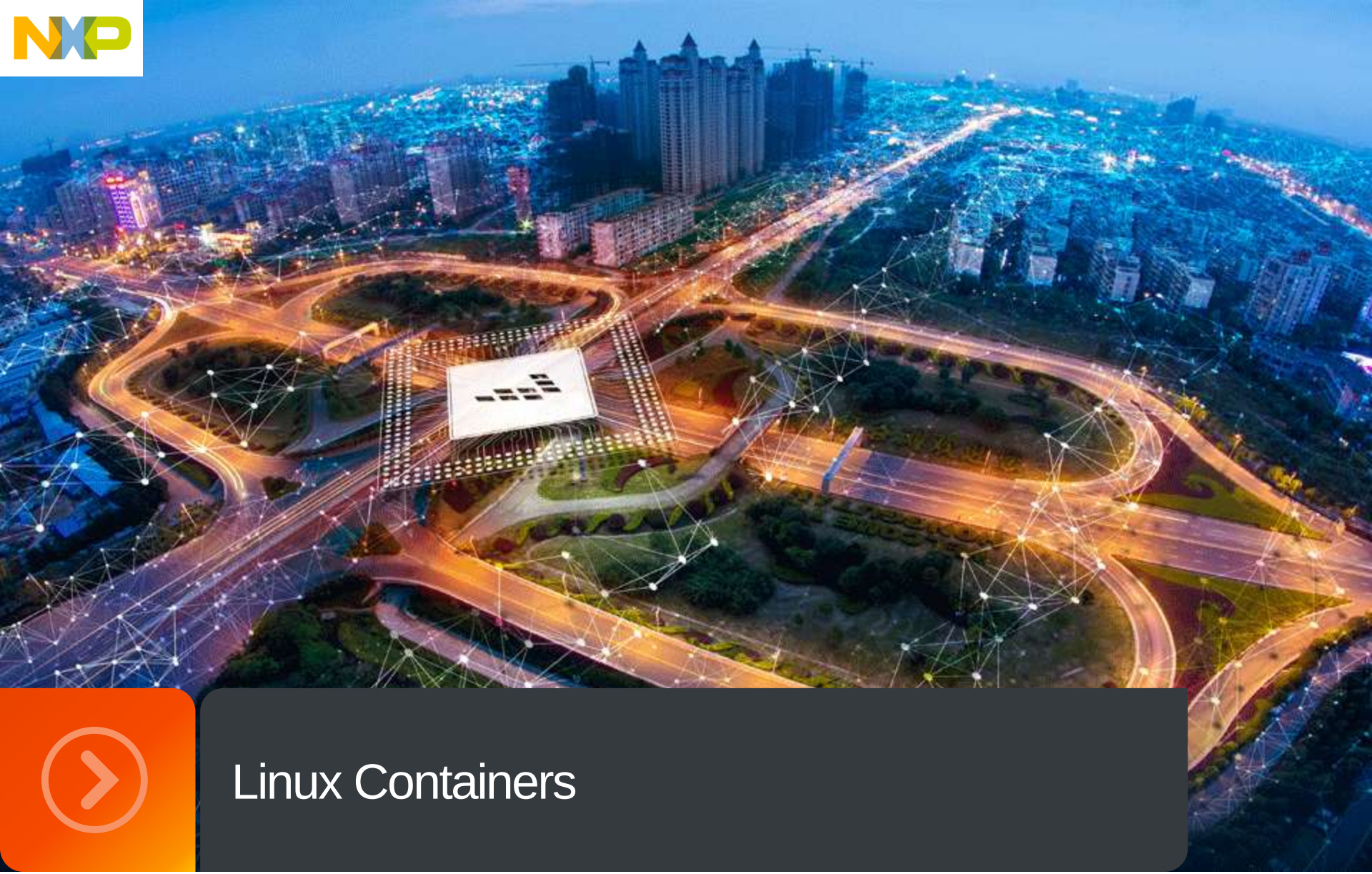
- Enables sharing of host network interfaces
- Host : - Bridge (virtual switch) is connected to physical host interface
  - QEMU uses tun/tap device connected to the bridge
- Guest : - Sees a private `virtio` network device on PCI bus
  - Virtio network driver is needed in guest



# Virtio Block

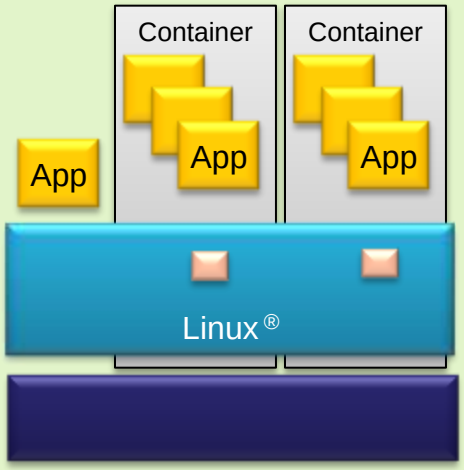
- Give each virtual machine a private storage device
- Virtual disk could be single binary image on host file system or logical volume on the host's disk
- Guest sees a private "virtio" network device on PCI bus
- Virtio block driver is needed in guest

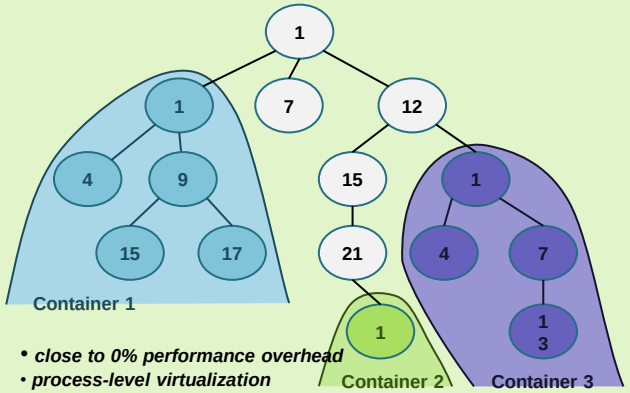




# Linux Containers

# Linux Containers (LXC) – Overview




- OS level virtualization
- Guest kernel is the same as the Host kernel, but OS appears isolated
- Provides low overhead, lightweight, secure partitioning of Linux applications into different domains
- Can control resource utilization of domains– CPU, memory, I/O bandwidth
- LinuX Containers is based on kernel components (cgroups, namespaces) and user-space tools (LXC)
- KVM virtual machines can be run in containers

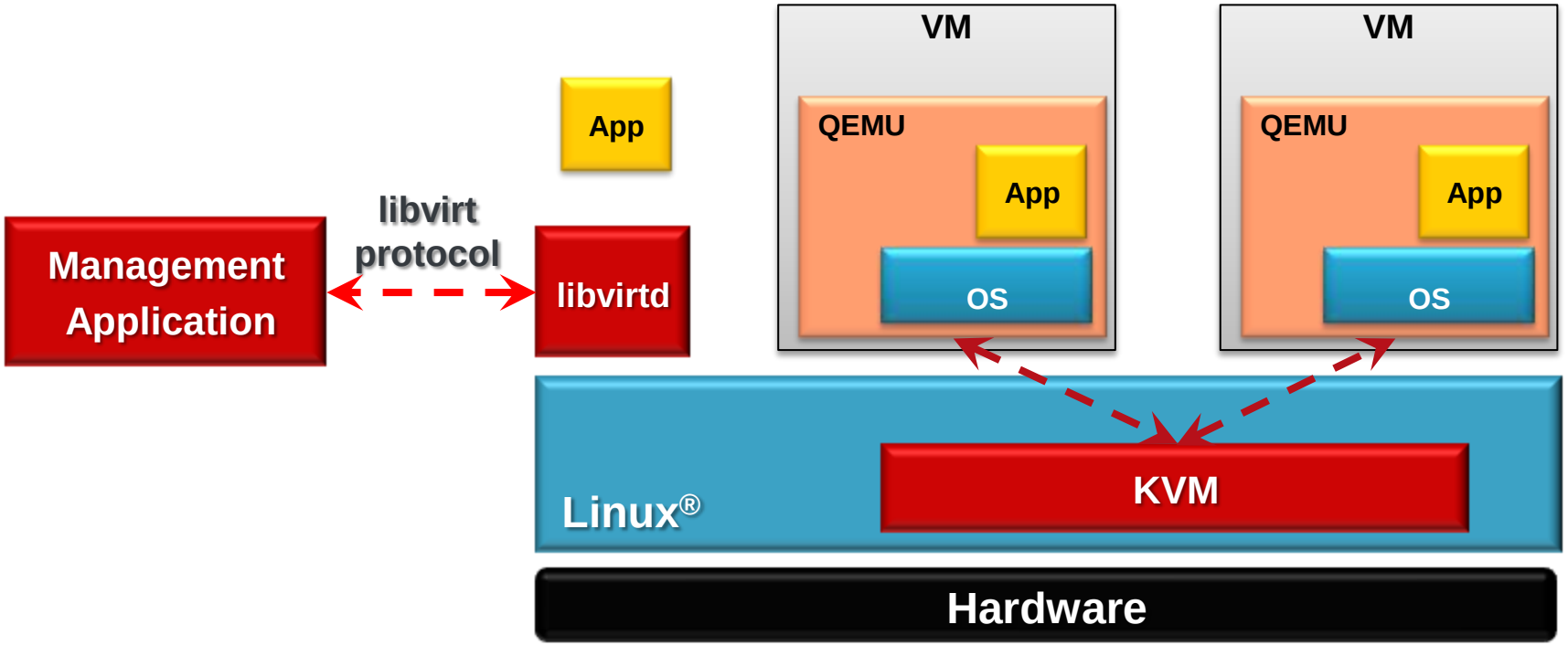
# Libvirt Overview

http://libvirt.org/

- A toolkit to interact with the virtualization capabilities of OS-es and hypervisors
- Goal : provide common and stable layer sufficient to securely manage domains on a node, possibly remote
- Has drivers for KVM/QEMU and Linux containers
- Many management applications supported

# Libvirt Overview

http://libvirt.org/



# Virtualization Hardware Comparison

# Comparison of Processor Virtualization Capabilities

- ARM, Power, x68 architectures all support similar mechanisms to support virtualization.

| Capabilities | ARM | Power | x86 |
|---|---|---|---|
| 3rd privilege level | Yes | Yes | Yes |
| Extended Address space | Yes | Yes | Yes |
| Hardware guest physical address translation (2-stage) | Yes | Yes (LRAT) | Yes (EPT/NPT) |
| Direct guest interrupt management | Yes | Yes | Yes (x2 APIC) |
| IOMMU | Yes (SMMU) | Yes (PAMU) | Yes (VT-d) |

# Virtualization Features in QorIQ Silicon

- CPU
  - e500mc / e5500
  - 3rd privilege level
  - Partition ID / extended virtual address space
  - Key registers duplicated for guests
  - Direct system calls
  - Direct external hardware interrupts to guest
  - LRAT– gphys -> phys translation in hardware
- SoC
  - IOMMU (PAMU) provides isolation from I/O device memory accesses

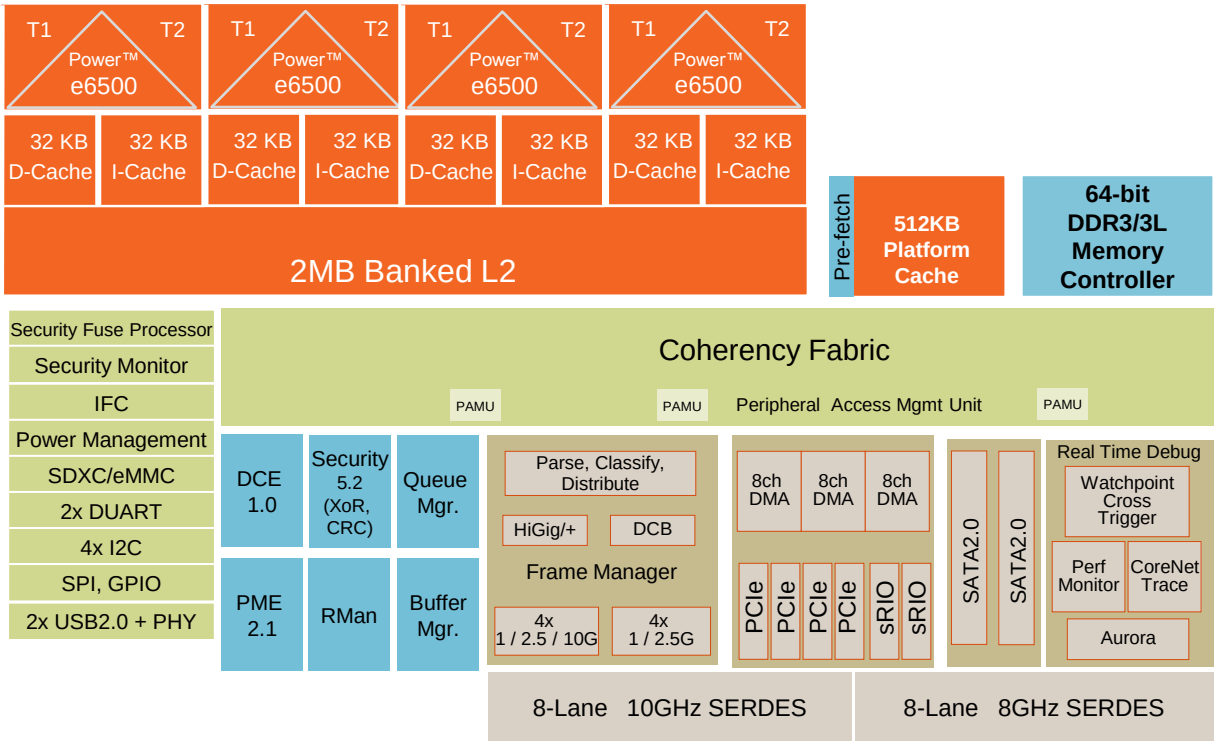
# ARM Hardware Virtualization Extensions

- A7/A15/A53/A57 Core
  - New Hypervisor mode
    - Privileged operations trap to hypervisor
    - Banked registers
  - Two stage address translation
    - Virtual address (VA) -> Intermediate physical (IPA)
    - Intermediate (IPA) -> Physical (PA)
  - Direct system calls
  - Guest timer in core
  - Guest GIC (interrupt controller) interface for ACK, EOI

- SoC
  - IOMMU (SMMU) provides isolation from I/O device memory accesses

# Networking Virtualization
# T-Series 10 GbE iNIC

# QorIQ T2080 Power Optimized Multicore Solution




**Network IO**

- Up to 25Gbps Simple PCD each direction
  - 4x1/10 GE, 4x1 GE or 2.5 Gb/s SGMII
  - XFI, 10 GBase-KR, XAUI, HiGig, HiGig+, SGMII, RGMII, 1000Base-KX

**Processor**

- 4x e6500, 64b, 1.2 - 1.8 GHz
- Dual threaded, with 128 b AltiVec
- 2MB shared L2; 256 KB per thread

**Memory Subsystem**

- 512 KB Platform Cache w/ECC
- 1x DDR3/3L Controller up to 2.1 GHz
- Up to 1 TB addressability
  - 40-bit physical addressing
- HW Data Prefetching

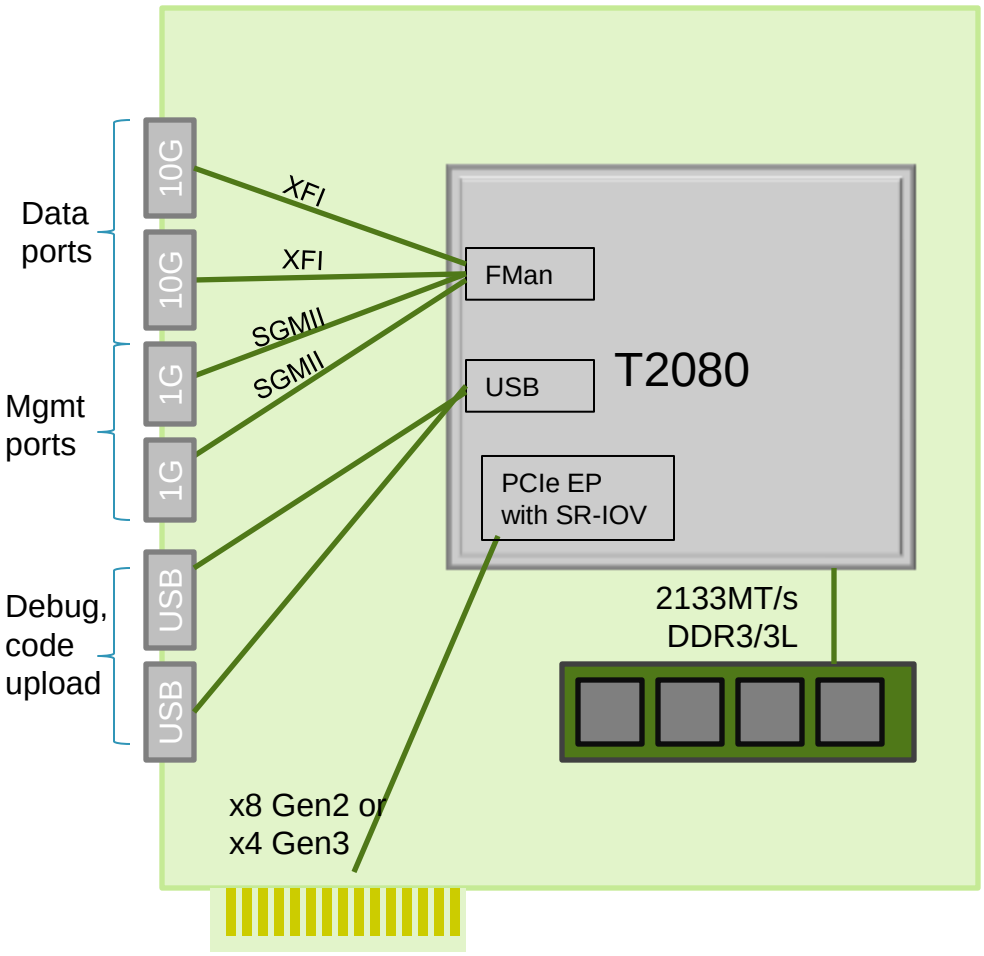
**Switch Fabric**

**High Speed Serial IO**

- 4x PCIe Controllers: Gen1.1/2.0/3.0
  - 1 with SR-IOV support
  - x8 Gen2
- 2x sRIO Controller
  - Type 9 and 11 messaging
  - Interworking to DPAA via RMan
- 2 SATA 2.0 3Gb/s
- 2 USB 2.0 with PHY
- **SEC**- crypto acceleration
- **DCE** - Data Compression 17.5 Gbps
- **PME** – Pattern Matching to 10 Gbps

# T2080 RDB System

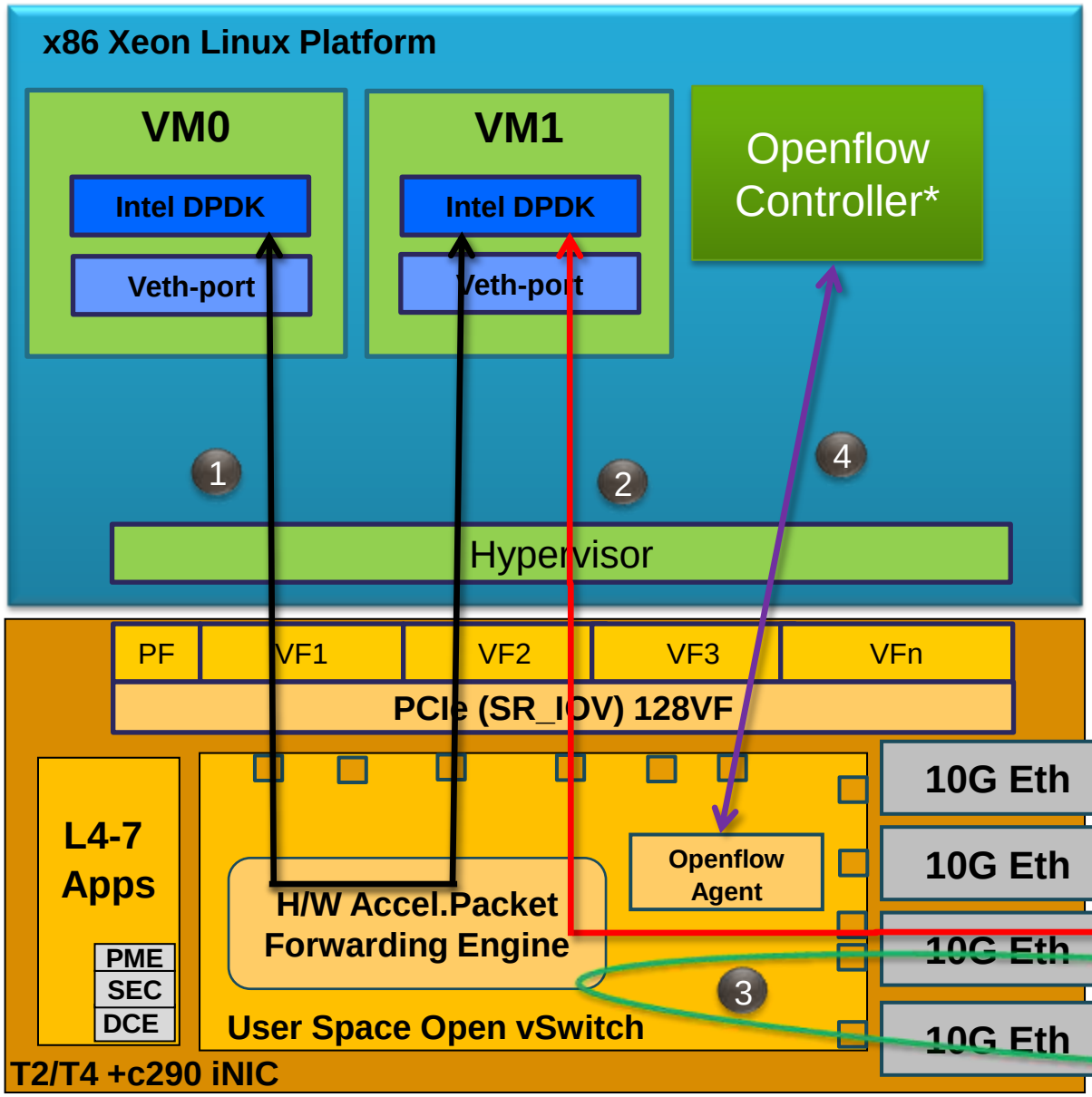
# Target Application: 20 Gb/s iNIC




- Well-balanced device for 20 Gb/s bi-directional application:
  - FMan moves about 25 Gb/s
  - 3x DMA engines move about 20 Gb/s
  - x4 Gen3 or x8 Gen2 PCIe moves 32 Gb/s
- SR-IOV allows virtual machines on host to see a private iNIC
- 15.5 W power fits in 30 W slot-provided power budget
- Improved PCIe Endpoint capabilities support customization of Device ID, Class Code, and Vendor ID. Driver can be stored in Expansion ROM
- Offload accelerators for services cards: 10 Gb/s IPSEC or Kasumi, 10 Gb/s pattern matching, 17.5 Gb/s data compression
- PCIe card reference board available

# Server with Freescale iNIC

## T4 iNIC demo Traffic Flow




* Can be external

**Enhanced L4-7 Functionality**

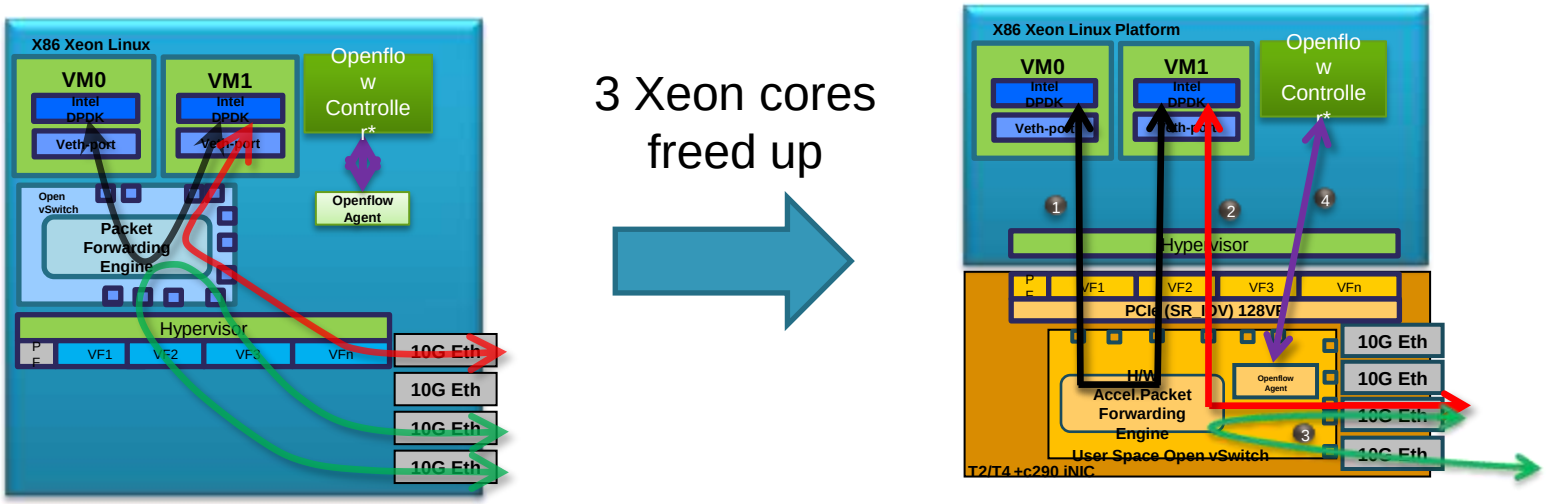
- NFV/SDN/Firewall/ACL
- IPSEC
- TCP offload
- Data Compression
- Deep Packet Inspection
- Load Balancing
- OpenSSL + record offload
- Vendor defined applications

**Benefits**

- Offloading of x86 CPU to increase aggregate with application performance cost effectively.
- Increase top end server performance
- Scalable iNIC platform performance T2080 to T4240. Reusable software.
- Hardware acceleration for Data Path, Pattern Matching, Security and Decompression /Compression, PKC/Record offload.

# Freescale iNIC Performance Advantage

- Intel Xeon platforms with a standard NIC require 4 cores of the Xeon CPU to run OVS (3 cores) and VMM (1 core).
- With a Freescale iNIC, 1 Xeon core continues running the VMM; the 3 cores running OVS are offloaded to the Freescale CPU.
- Additionally Freescale processors contain network application oriented hardware accelerators (security, compress/decompress, pattern matching ) which accelerate key iNIC use cases.



# Q&A


www.Freescale.com