FTF FREESCALE TECHNOLOGY FORUM 2014

Benchmarking Virtualization Solutions for QorIQ Processors

FTF-SDS-F0028

Alexandru Marginean | Senior Member of Technical Staff

APR.2014

freescale™

External Use

Session Introduction

- This session explores performance considerations when using virtualization on QorIQ processors
  - This presentation will evaluate the overhead of different virtualized environments by using various benchmark methodologies
  - The attendees will understand the reasons of virtualization overhead and how they can be evaluated and minimized

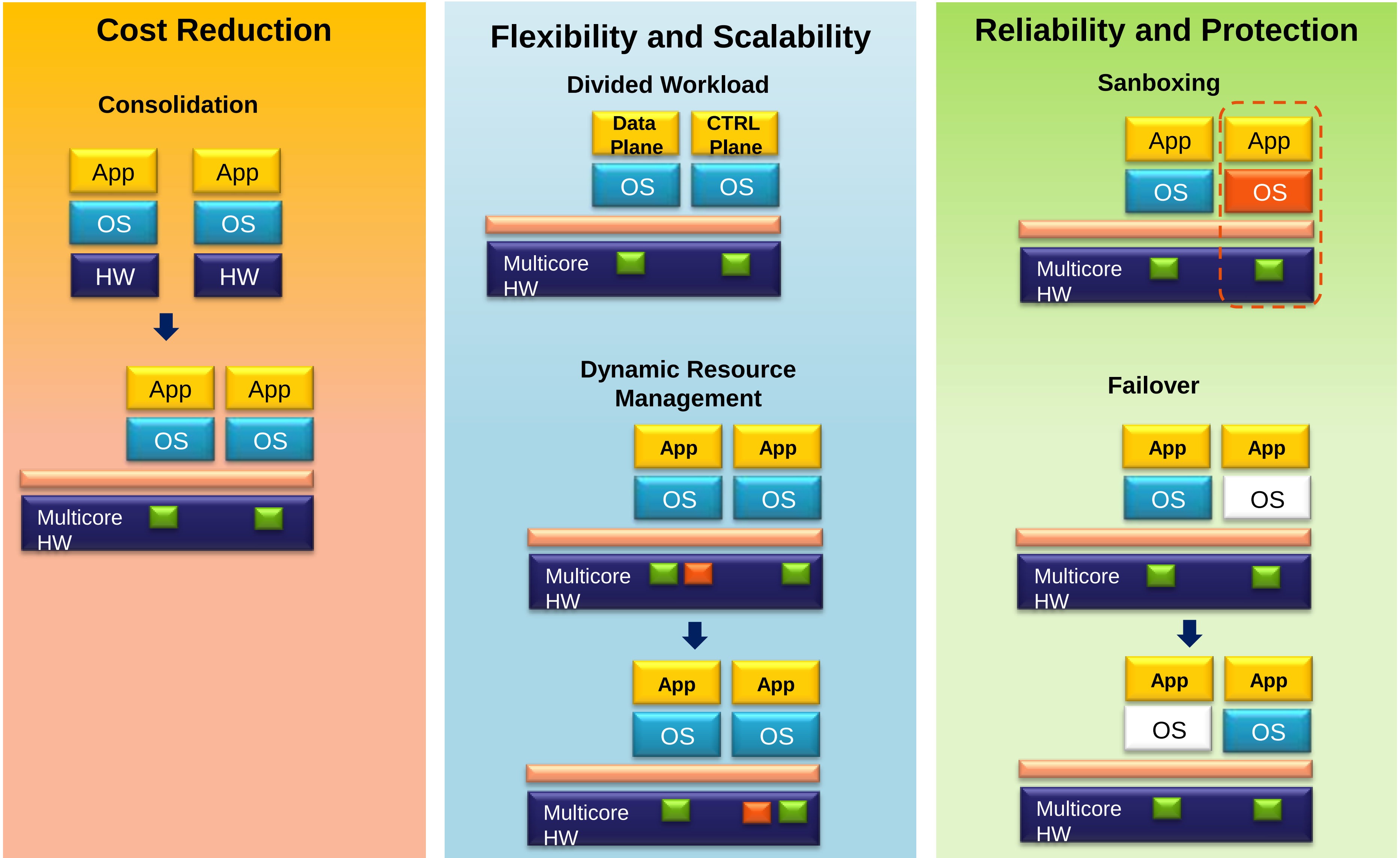
Session Objectives

- After completing this session you will be able to:
  - Understand the virtualization benefits
  - Evaluate virtualization performance on QorIQ devices
  - Identify reasons of performance degradation
  - Understand the HW/SW mechanisms that reduce virtualization overhead
  - Compare different virtualization technologies on QorIQ devices

Agenda

- Freescale Virtualization Technologies for QorIQ processors
- Sources of Virtualization Overhead
- Hardware/Software Mechanisms that Reduce Overhead
- Benchmarking Hypervisors
- Performance Considerations & Recommendations

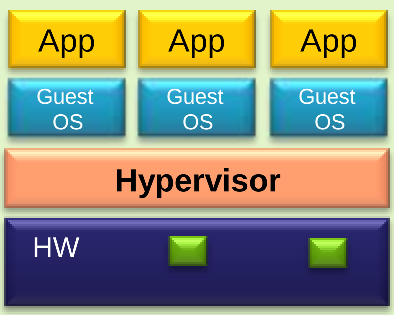
Virtualization Use Cases



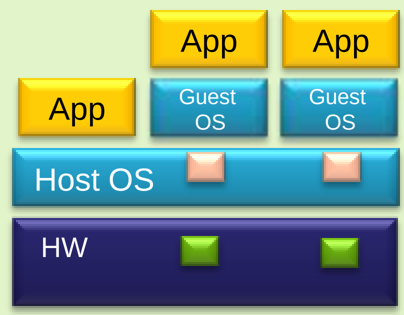
Virtualization & Hypervisors

- **Virtualization** – Hardware and software technologies that provide an abstraction layer that enables running multiple operating systems on a single computer system

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

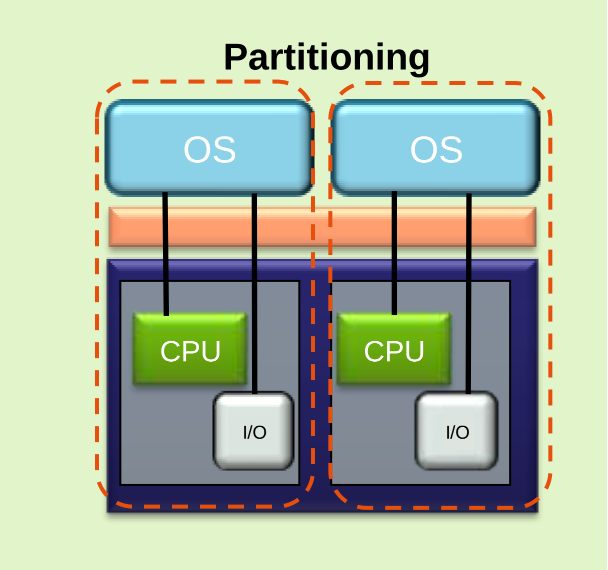


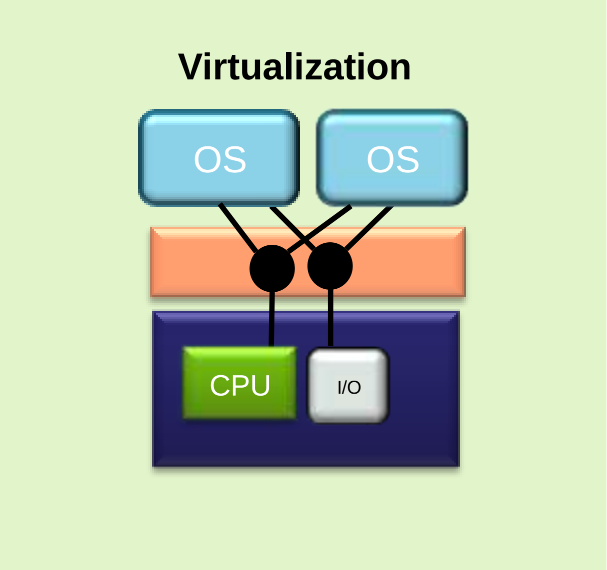
- Hypervisor runs “bare metal”

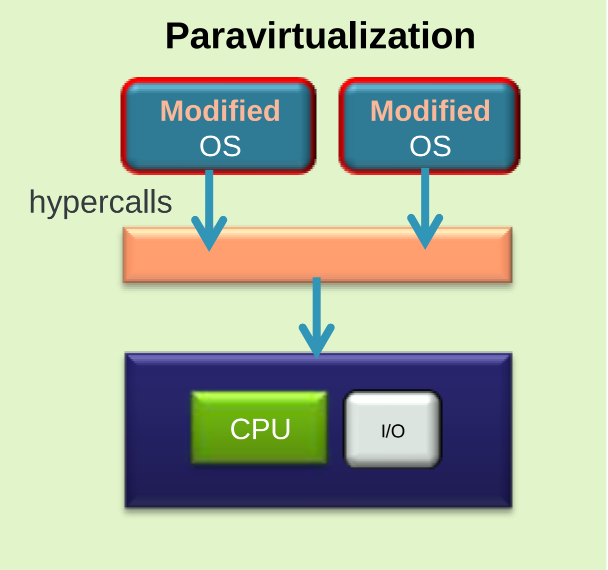


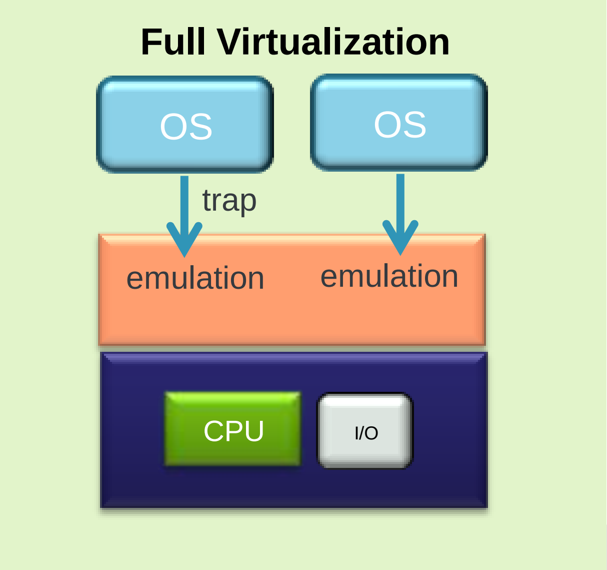
- Hypervisor is integrated in Host OS
  - Reuses OS infrastructure
- Host OS runs other applications

Virtualization & Partitioning



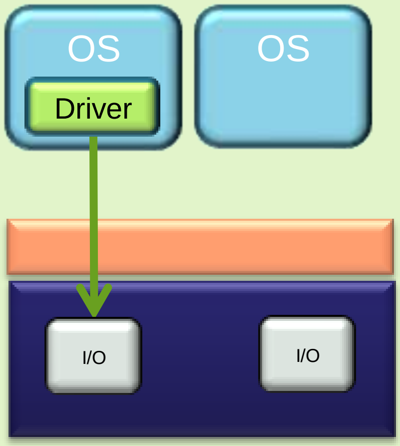




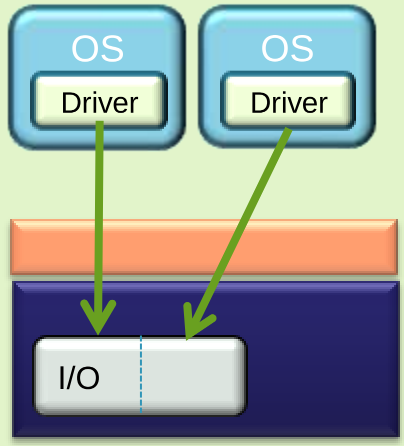


Device Usage in Virtual Environments

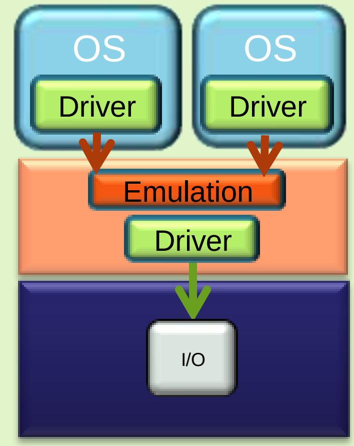
Direct Access
- Fastest native performance
- Direct access to hardware



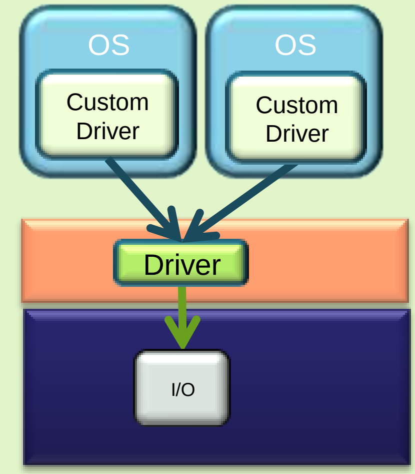
Partitionable HW
- Hardware partitioned
- One hardware block



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



Para-Virtualized
- Driver in Hypervisor
- Modified Drivers in Guest



Hardware/software access

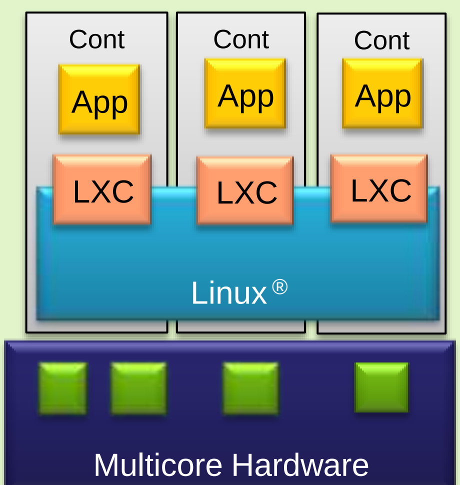
Hypercalls

Traps

Freescale Virtualization Technologies for QorIQ processors

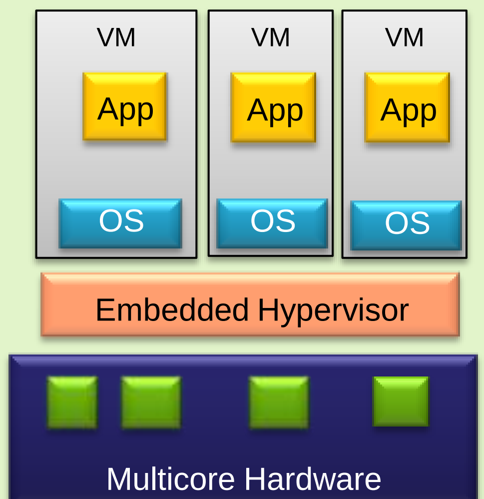
OS Virtualization (LXC)

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



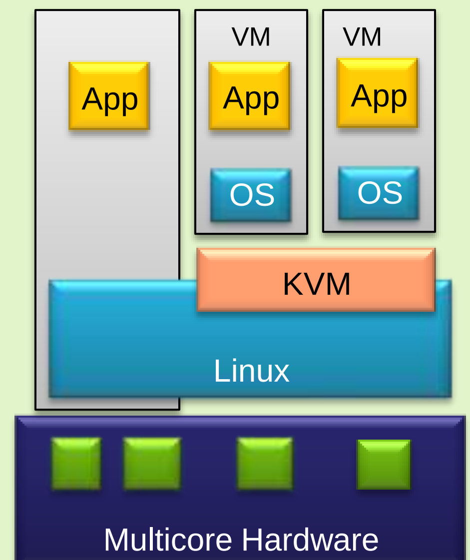
Freescale Embedded Hypervisor

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

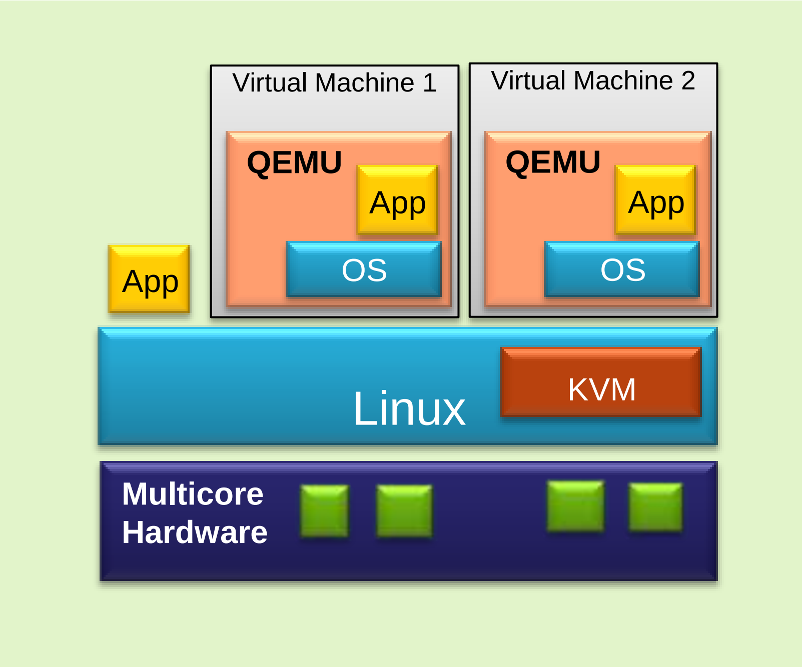


KVM

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



KVM/QEMU – Overview

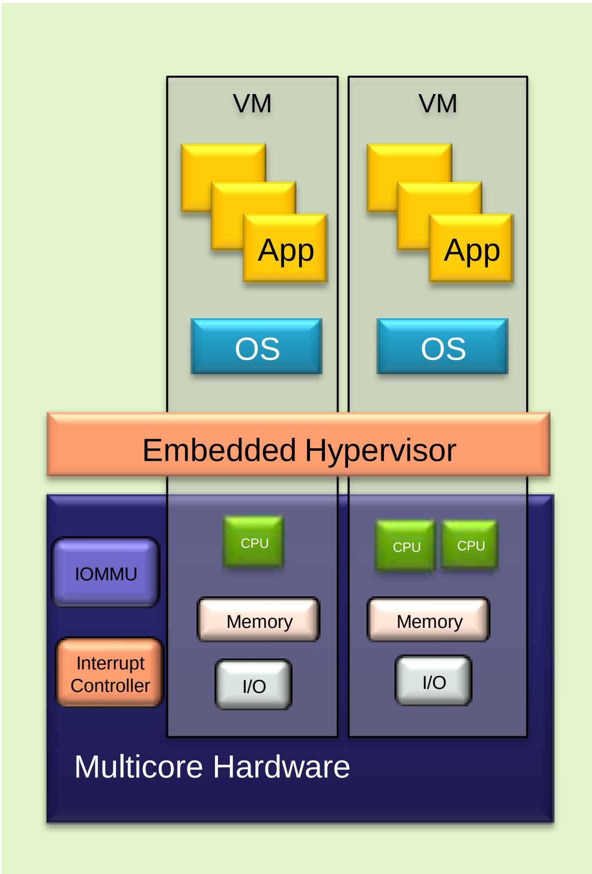


- KVM/QEMU– open source virtualization technology based on the Linux® kernel
- KVM is a Linux kernel module
- QEMU is a user space emulator that uses KVM for acceleration
- Run virtual machines alongside Linux applications
- No or minimal OS changes required
- Virtual I/O capabilities
- Direct/pass thru I/O – assign I/O devices to VMs

KVM/QEMU - Details

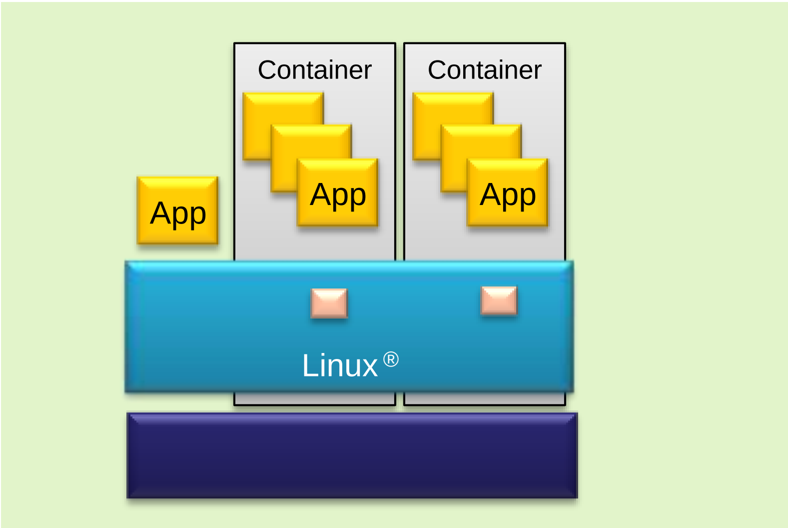
- QEMU is a user space emulator that uses KVM for acceleration
  - Uses dedicated threads for vcpus and I/O
  - KVM leverages hardware virtualization to run guest with higher privileges
  - MPIC virtual chip emulation in kernel
  - I/O
    - Provides dedicated virtio I/O devices and standard drivers 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
    - Multique virtio devices connected to multiqueue tap devices
  - Provides services for console, debug, reset, watchdog, etc

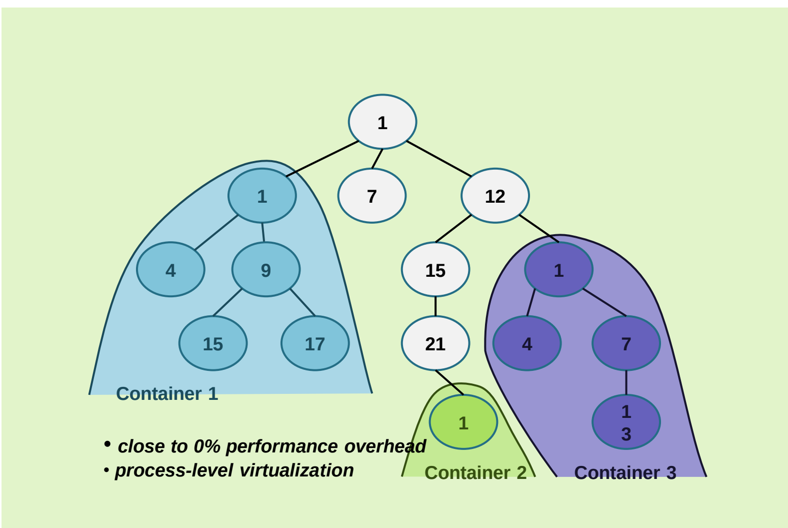
Freescale Embedded Hypervisor – Overview



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

Linux Containers – 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 a collection of technologies including kernel components (cgroups, namespaces) and user-space tools (LXC).

Virtualization Overhead

CPU Performance Considerations

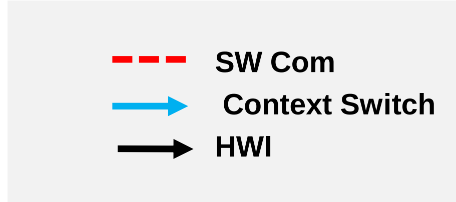
- The performance overhead when running on a hypervisor is workload and application dependent.
- Sources of overhead when running under a hypervisor
  - Handling Exceptions and Interrupts
  - Memory Management
    - Privileged instructions/registers
    - TLB miss handling
  - VM Scheduling
  - Third privilege level → Increased number of context switches
  - Device emulation
  - Privileged instructions and registers
  - Hypercalls

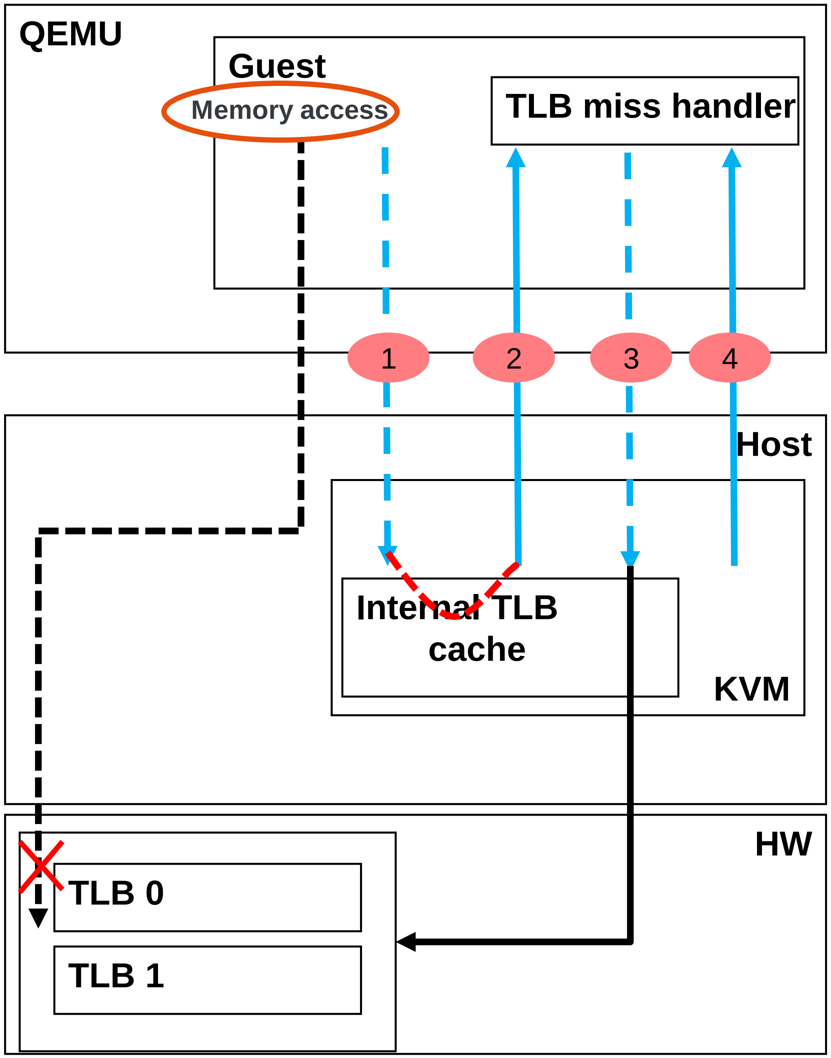
KVM Overhead

- Analysis on the KVM overhead in specific workflows for various configurations
  - Memory Management
  - RX Flow
  - TX Flow

- Virtualization overhead is evaluated by analyzing the software context switches

...ory Management e500mc/e5500





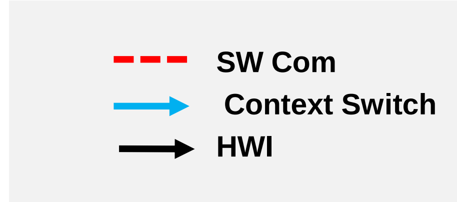
1 TLB miss

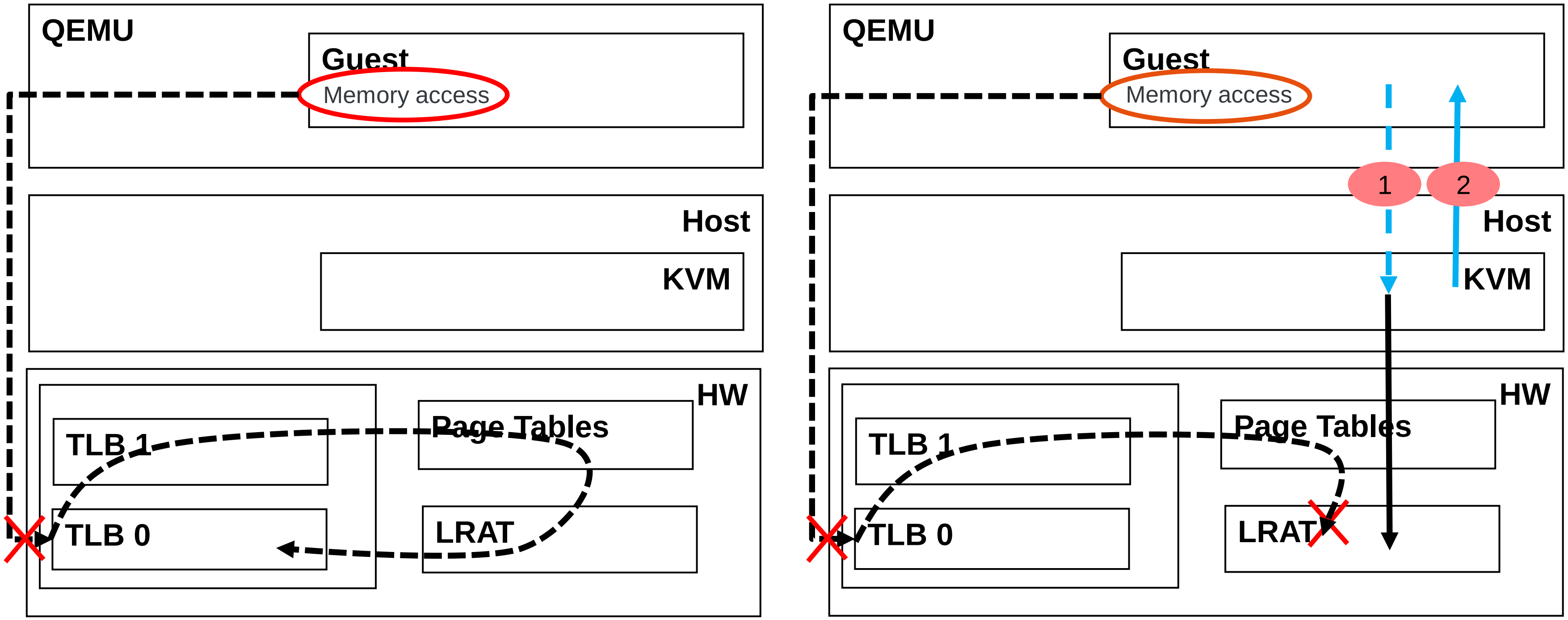
2 Return from exception

3 Privilege trap: tlbwe

4 Return from privilege trap

Memory Management e6500 (1)

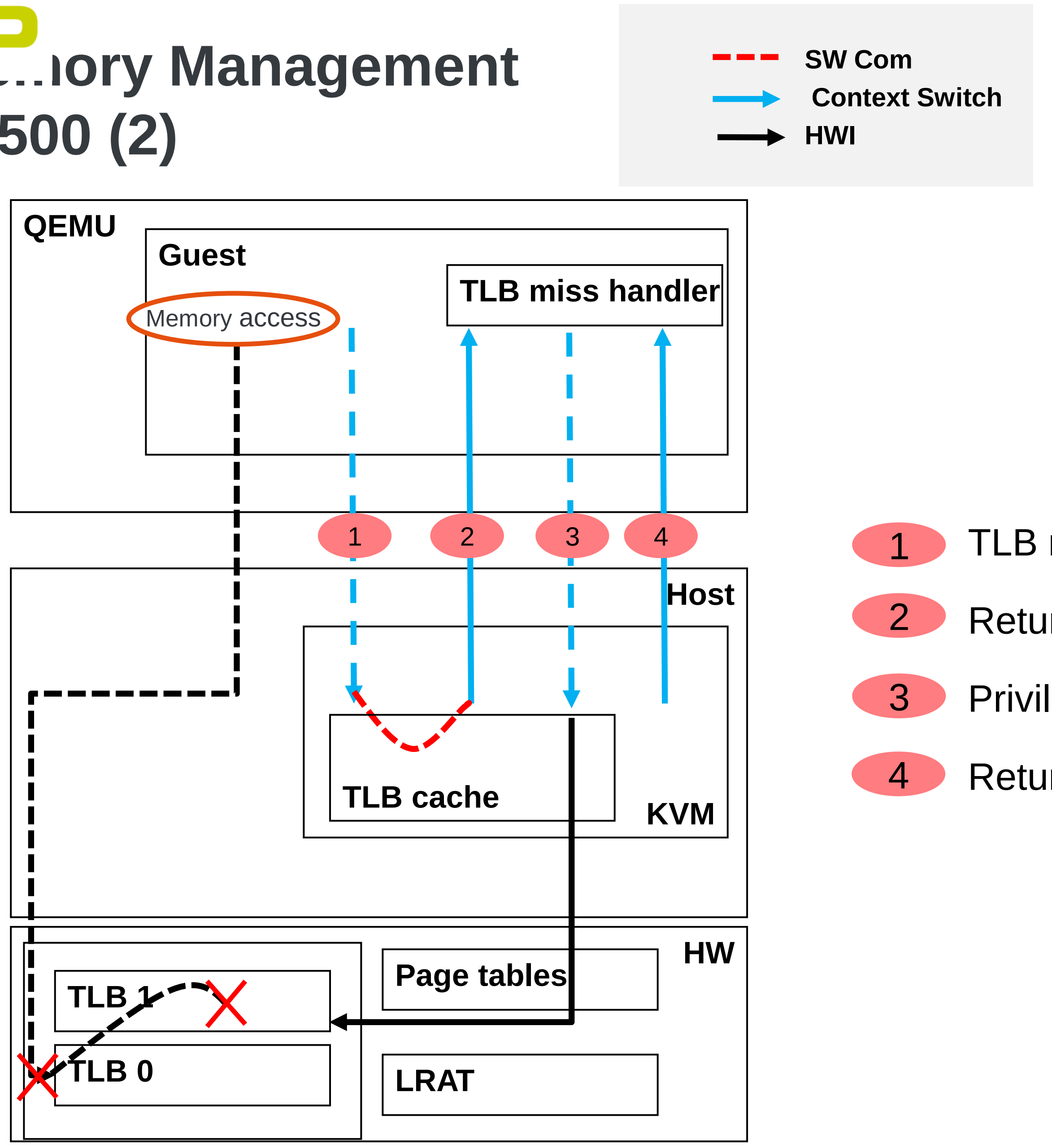




1 LRAT miss

2 Return from exception

...ory Management e6500 (2)



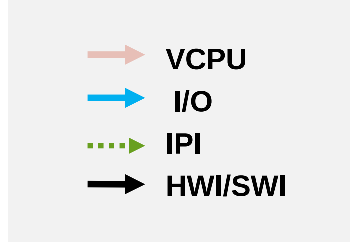
1 TLB miss

2 Return from exception

3 Privilege trap: tlbwe (indirect entry)

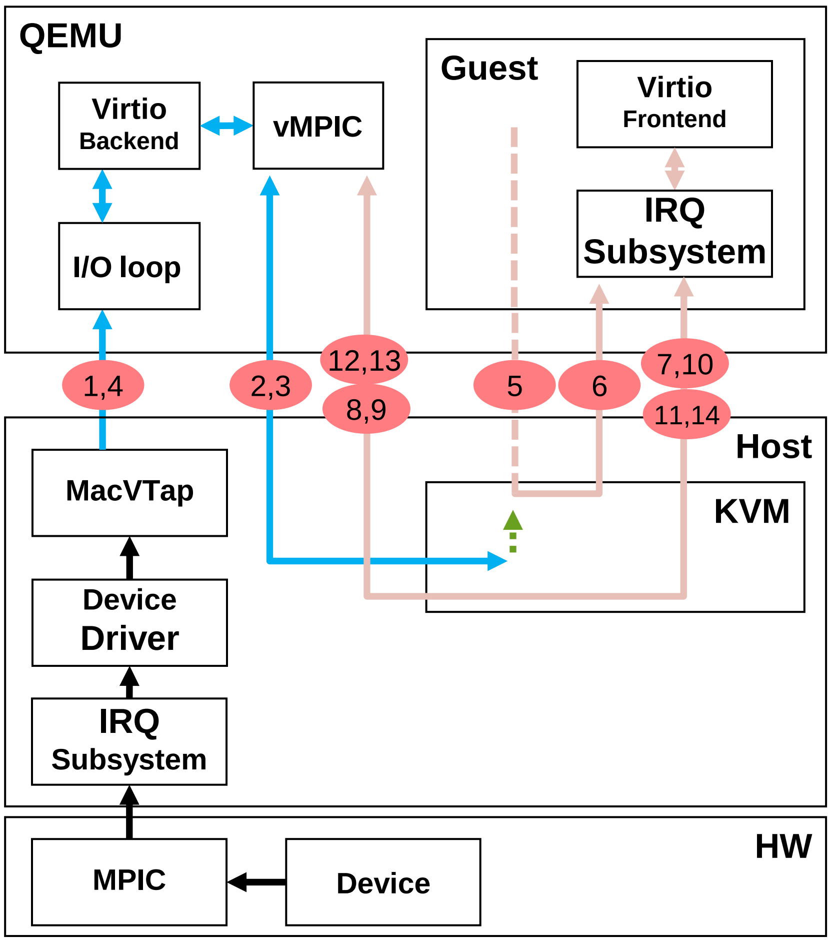
4 Return from privilege trap

IRQ Flow
QEMU Emulation



QEMU Emulation

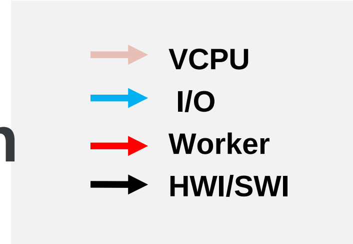
- Virtio device
- No in-kernel mpic
- No vhost

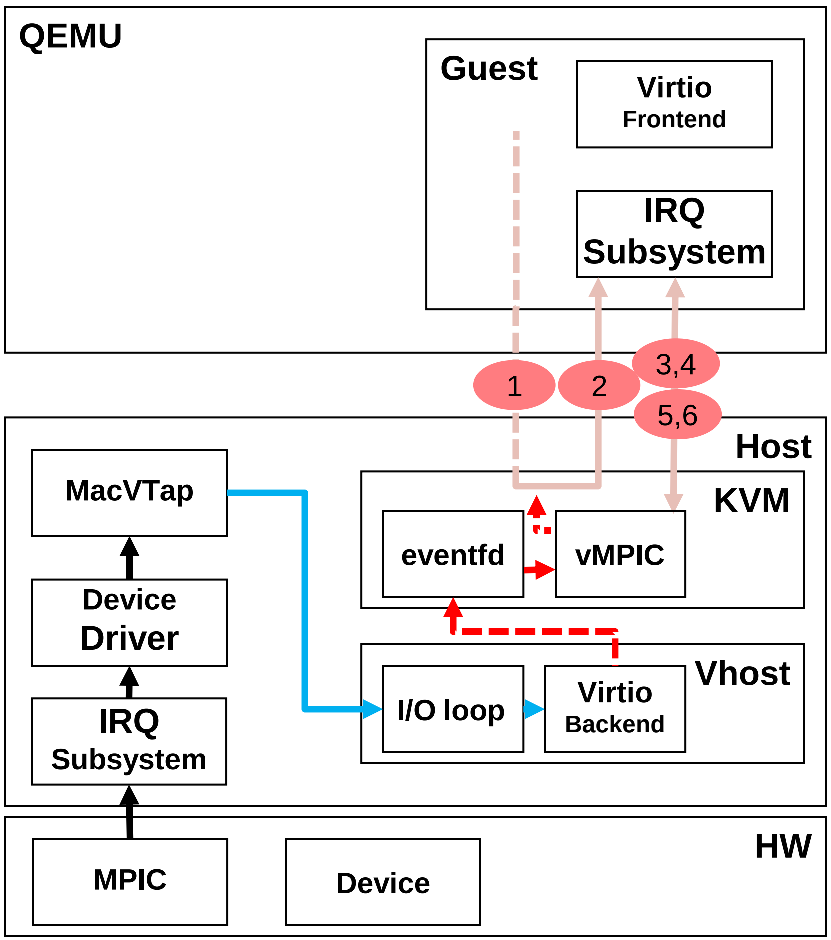


1	Wait for input
2	Inject external interrupt
3	Return
4	Wait for input
5	Reschedule vcpu
6	Inject external interrupt
7,8	Read vMPIC's shared MSI register
9,10	Return
11,12	Write vMPIC's EOI
13,14	Return

low

In-kernel Emulation



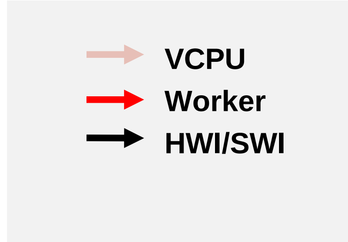


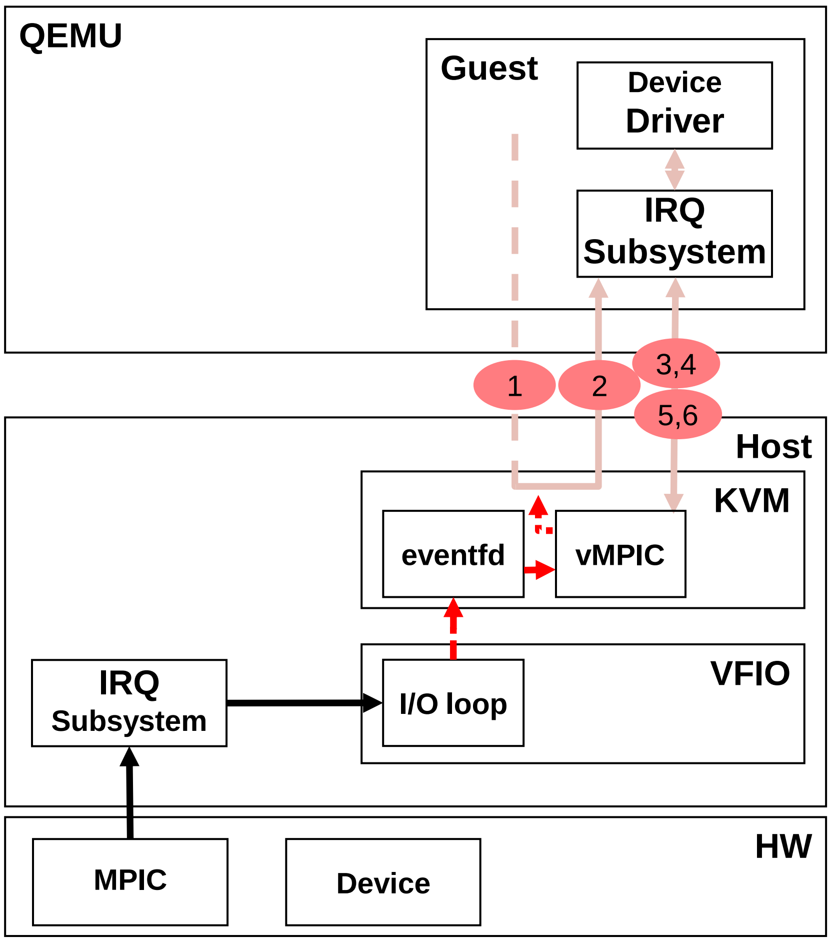
In-kernel Emulation

- Virtio device
- In-kernel mpic
- Vhost (with irqfd/ioeventfd)

1. Reschedule vcpu
2. Inject external interrupt
3. Read MPIC’s shared MSI register
4. Return
5. Write MPIC’s EOI
6. Return

low
Direct Assignment





Direct assignment

- Physical device
- Vfio (with irqfd & pamu integration)
- Hw mpic with isolated banks for msi (no guest virtualization)

1 Reschedule vcpu

2 Inject external interrupt

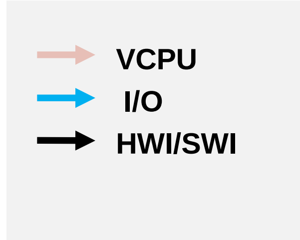
3 Read vMPIC’s shared MSI register

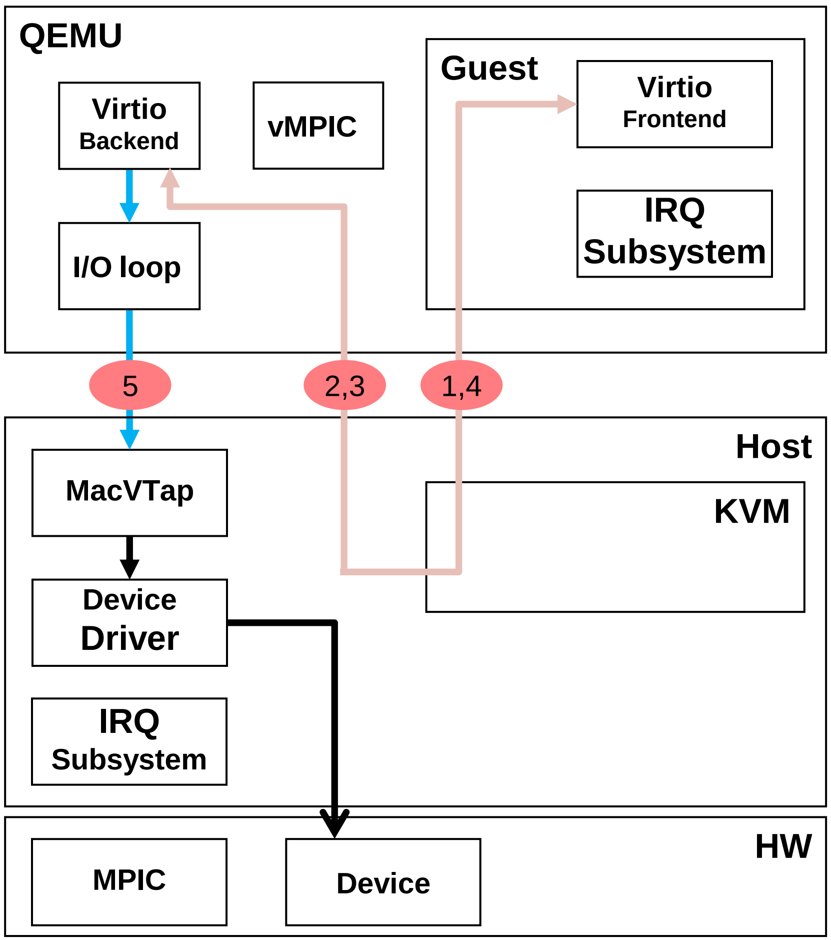
4 Return

5 Write vMPIC’s EOI

6 Return

low
QEMU Emulation



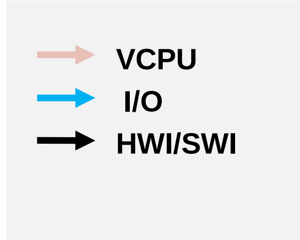


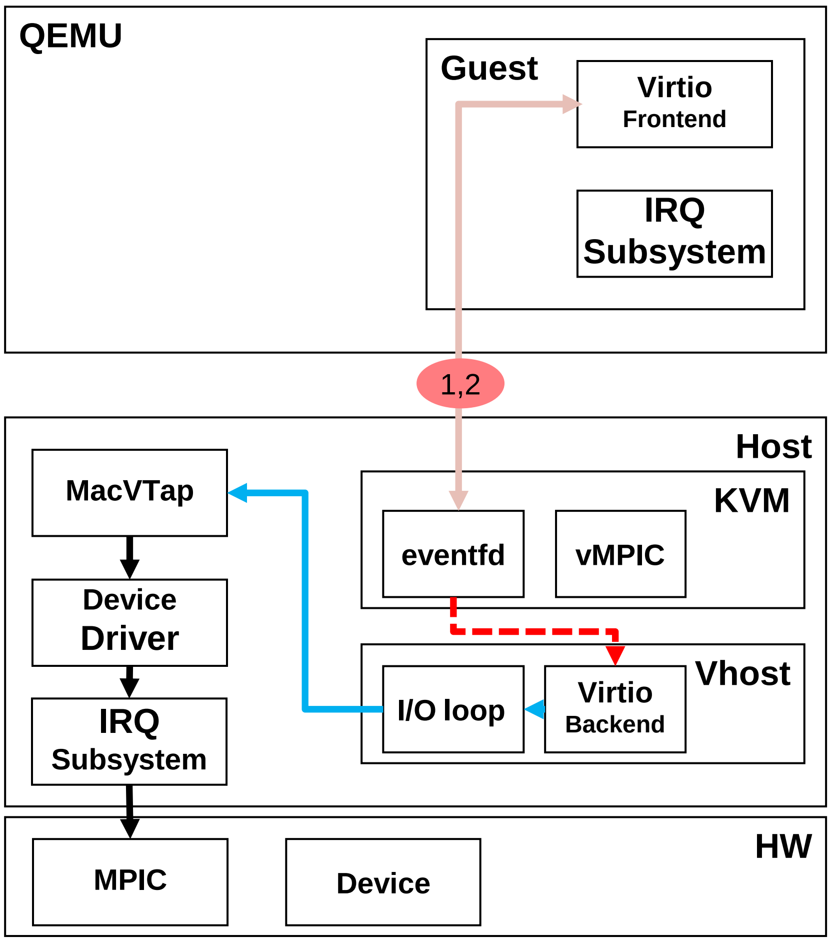
QEMU Emulation

- Virtio device
- No in-kernel mpic
- No vhost
- No irqfd/ioeventfd

1. Send command to virtio device
2. Emulate memory mapped I/O access
3,4. Run vcpu
5. Send data

low
In-kernel Emulation





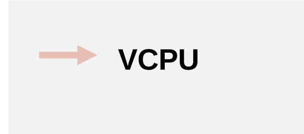
In-kernel Emulation

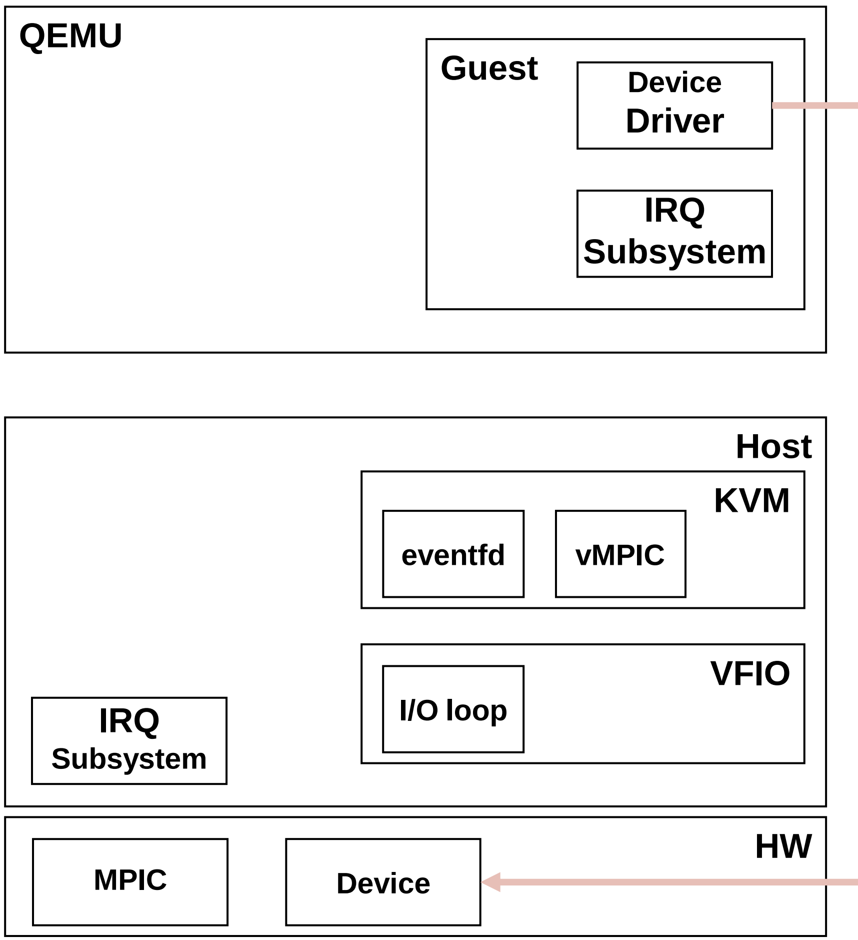
- Virtio device
- In-kernel mpic
- Vhost (with irqfd/ioeventfd)

1 Send command to virtio device

2 Return

… low
Direct Assignment





Direct Assignment

- Physical device
- Vfio (with irqfd & pamu integration)
- HW mpic with isolated banks for msi (no guest virtualization)

Benchmarking Hypervisors

Benchmarking Considerations

- Goal
  - Evaluate overhead, but
    - Use fair evaluation of alternatives
    - Benchmark the relevant scenarios

- Classification

Microbenchmarks

•Particular system operation
•Evaluate performance bottlenecks

Macrobenchmarks

•Realistic workloads
•Real-life system performance

CPU Benchmarks

•CPU Operations

System/OS Benchmarks

•Focuses on OS including memory, caches, interrupts

I/O Benchmarks

•Focuses on IO: e.g. network, disk

Synthetic Benchmarks

•Simulate certain scenarios
• May use random or extreme data
•May represent unrealistic workloads

Real World Benchmarks

• Real applications
• Execution traces from real applications

Hypervisor Benchmarking Considerations

- Goal
  - Evaluate the virtualization overhead, but:
    1. Benchmark the relevant scenario
       - A network benchmark may show different numbers than a (random) memory accesses
    2. Use the same environment for benchmarking
       - Number of cores or available memory

- May be difficult to get a real native vs virtualized performance
- No standard benchmark suite available for embedded hypervisor
- Many types of benchmarking mechanisms used but results needs to be interpreted correctly

Hypervisor Benchmarking Considerations

- Do not use Microbenchmarks for absolute performance evaluation of one virtualized environment versus a bare-metal one
  - Variance of Microbenchmarks results can be very high

- For comparing bare metal performance with virtualized performance use scenarios as close to the real targeted application.

- We have used OS/IO/system benchmarks
  - CoreMark
  - LmBench
  - Netperf

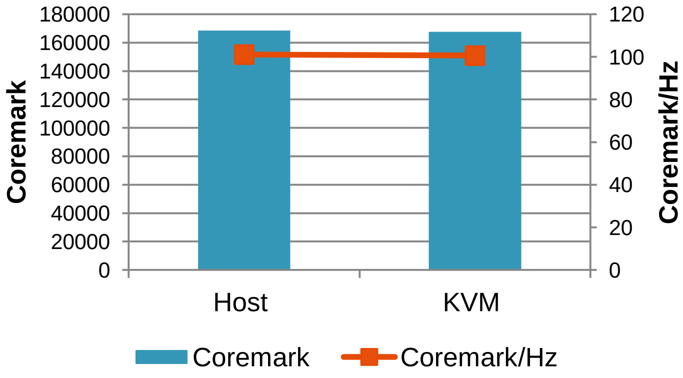
Benchmarks Overview

- **CoreMark**
  - Microbenchmark targeting the CPU core
  - Performs: list processing, matrix operations, determine if an input stream contains numbers, CRC
- **Lmbench**
  - lmbench is an extensible suite of micro-benchmarks
  - Measures latencies and bandwidth in the core, memory subsystem, network, file system and disk
- **Netperf UDP**
  - Netperf is a client-server application for network bandwidth testing between two hosts on a network
    - netperf client and netserver daemon use two socket connections
      - communication socket socket used for all internal communication
      - data socket used for performance benchmark
  - We measured UDP throughput performance

Benchmarking Results

CoreMark Results

	Scenario	CoreMark Score	CoreMark / MHz
1	Host	168447	101.07
2	KVM	167619	100.57



Environment

- T4240 platform
- GCC4.7.3 (-O3 -mcpu=e6500 -m32 -mno-altivec)
- -DMULTITHREAD=24
- Linux SMP 24 cpus

Setup Comparison

1. Host (24 cpus)
2. KVM Guest with 24 vpcus

Conclusions

- Coremark scores in virtualized environments are close to bare metal
  - CPU operations are not impacted by virtualization
- Reasons
  - Limited memory management operations performed
  - All operations are tied to the core: matrix multiplication, list processing, CRC

LMBench Results

Scenario	Host	KVM	Degradation
Processors, processes – times in microseconds			
Select on TCP fd's	**8.66**	**8.69**	**0.34 %**
Signal handler overhead	**3.48**	**3.50**	**0.57 %**
Arithmetic operations – times in nanoseconds			
Integer divide	**45.7**	**45.8**	**0.21 %**
Float divide	**12.0**	**12.0**	**0 %**
Double divide	**21.0**	**21.1**	**0.47 %**
Local communication bandwidth – MB/s			
AF UNIX	**1357**	**1306**	**3.75 %**
Memory read	**646**	**646**	**0 %**
Memory write	**5617**	**5309**	**5.48 %**
Memory read latencies in nanoseconds			
Sequential access	**83.3**	**84.5**	**1.44 %**
Random access	**125.2**	**648.3**	**417.81 %**

Environment

- T4240 platform
- GCC4.7.3 (-O3 -mcpu=e6500 -m32 -mno-altivec)
- Linux SMP 24 cpus (baremetal vs KVM)

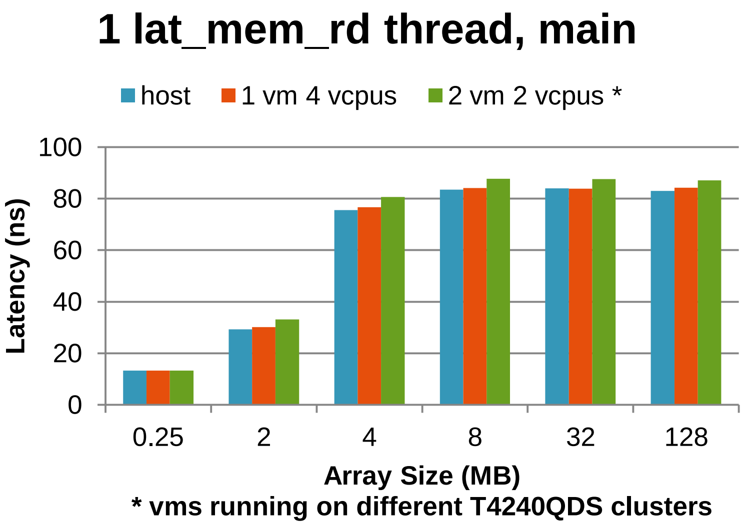
Guest Setup

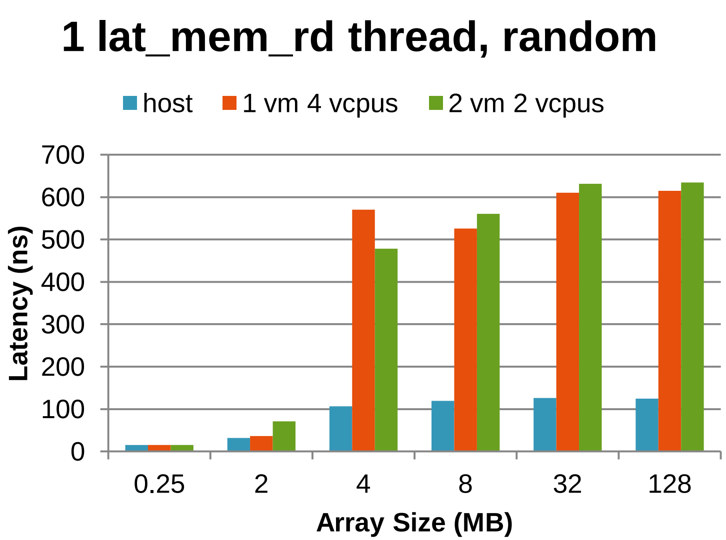
- 1 GB memory
- Hugepage backed
- SMP 24 vcpus

Conclusions

- CPU intensive operations have very low overhead
- Memory access performance depends on type of access (sequential vs random)
- Context switch results depends on scenario

LMBench Memory Latency





Guest Setup

- 1 GB memory
- Hugepage backed
- SMP <2/4> vcpus

Platform Setup

- 1 GB hugepage size
- HW PageTable walk enabled

Conclusions

- Little overhead for sequential memory access
- Overhead for random memory access
  - Triggered by the MMU subsystem virtualization
  - Pressure on TLB and cache increases with data array size
  - Pressure on TLB1 is influenced by TLB1 indirect entry size
    - Linux TLB1 indirect entry size is limited to 2MB
  - Larger hugepages will put less pressure on TLB1

NetPerf Setup

Environment

- 2 T4240 boards, connected back to back through 10G XAUI – “server” and “client”
- All tests run on client – server only used as the other endpoint, no monitoring
- GCC4.7.3 (-O3 -mcpu=e6500 -m32 -mno-altivec)
- Different Tx / Rx setups
- PCD Rx classification + interrupt steering on host
- Scatter / gather support
- GRO + GSO activated

Netperf:

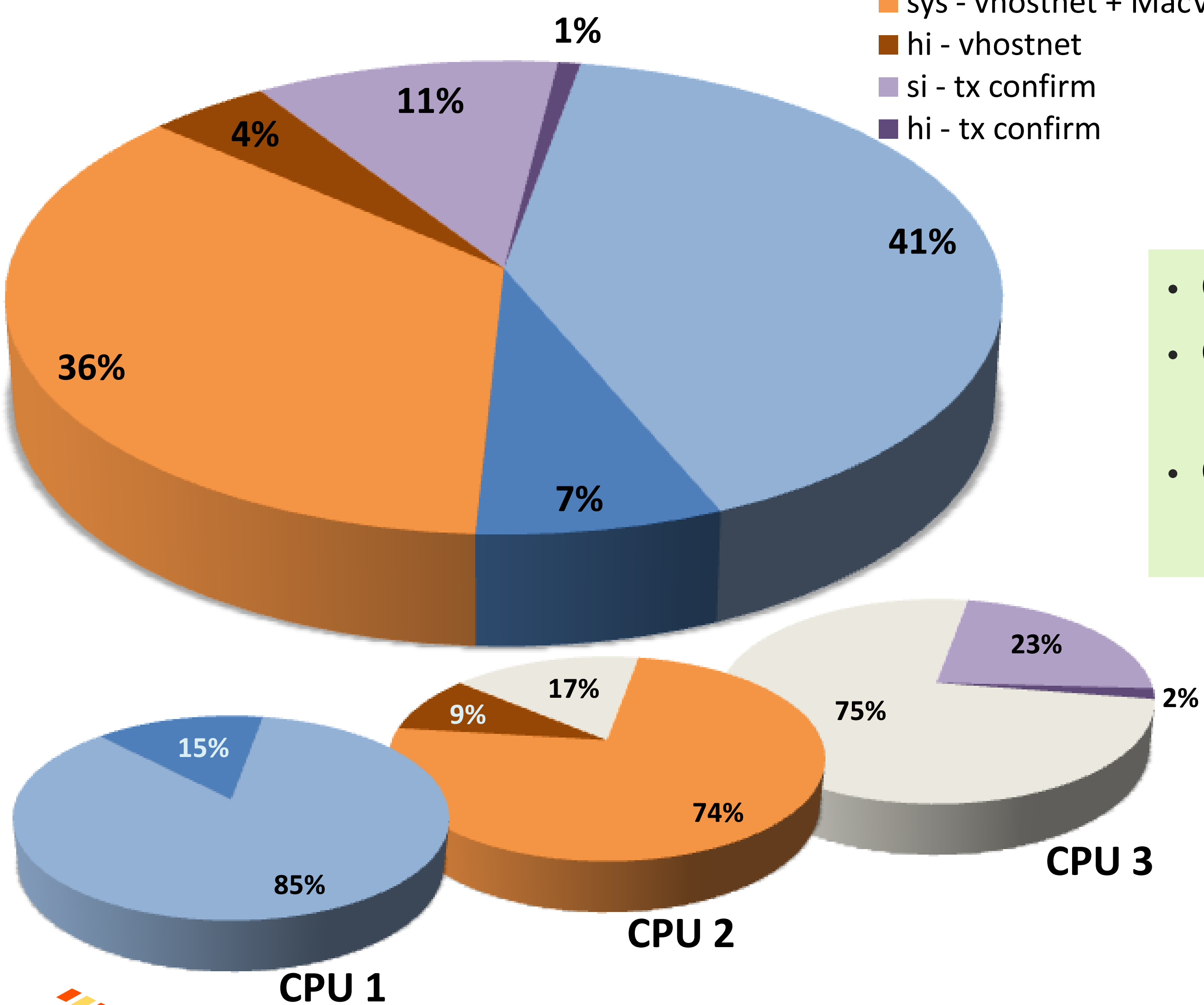
- UDP_STREAM
- Unidirectional
- Message size: 1472
- Core affinity

Guest Setup

- 1 GB memory
- Hugepage backed
- **Macvtap + vhostnet + virtio**
- Multiple single-queue interfaces
- VCPU, vhostnet process affinity

NetPerf Tx CPU breakdown – vhostnet

1 netperf flow, 1.57 Gb/s, **3.47 GHz**

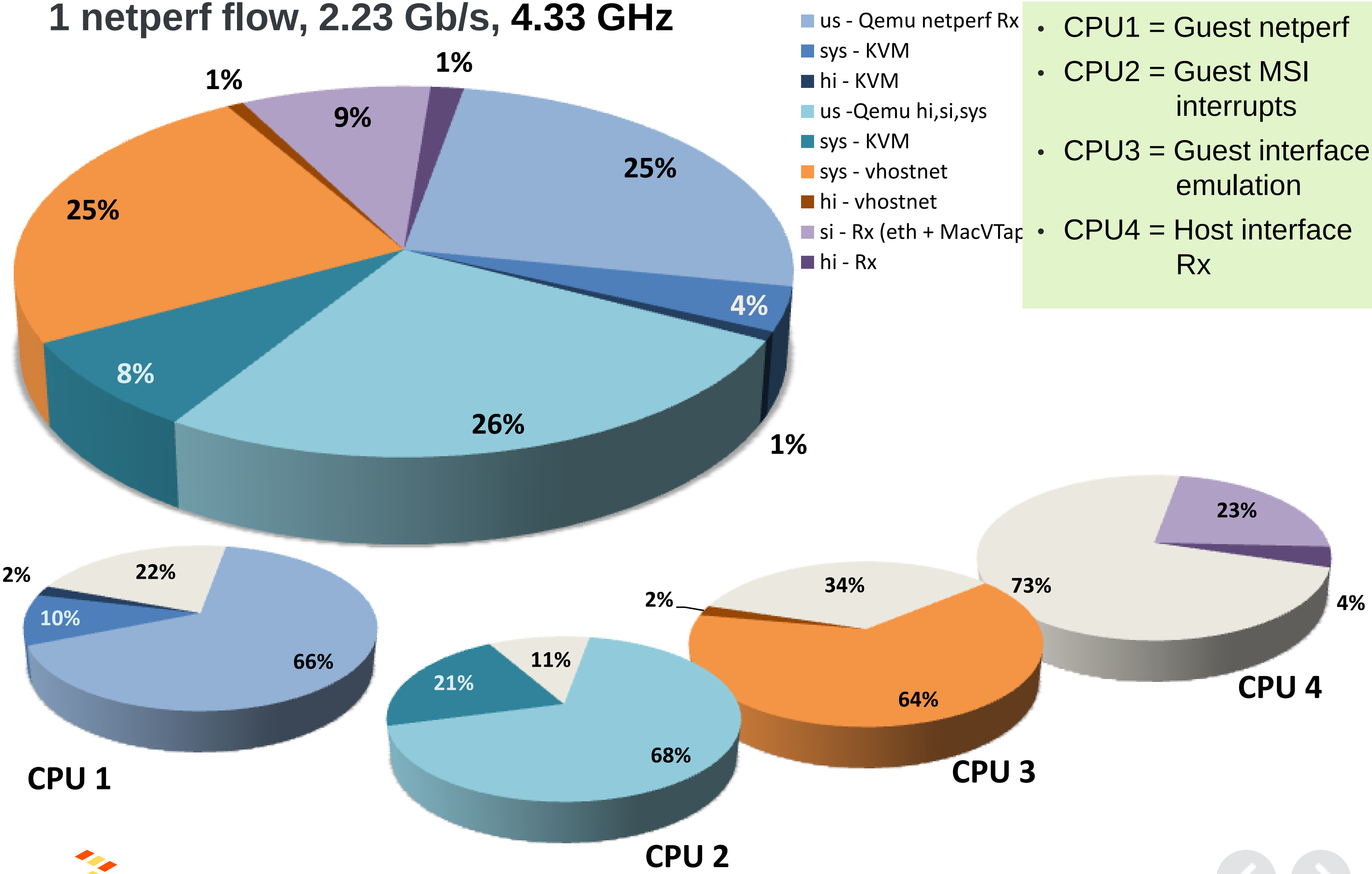


- CPU1 = Guest netperf
- CPU2 = Guest interface emulation
- CPU3 = Host interface Tx confirmation

Netperf Rx CPU breakdown – vhostnet

1 netperf flow, 2.23 Gb/s, **4.33 GHz**

- CPU1 = Guest netperf
- CPU2 = Guest MSI interrupts
- CPU3 = Guest interface emulation
- CPU4 = Host interface Rx

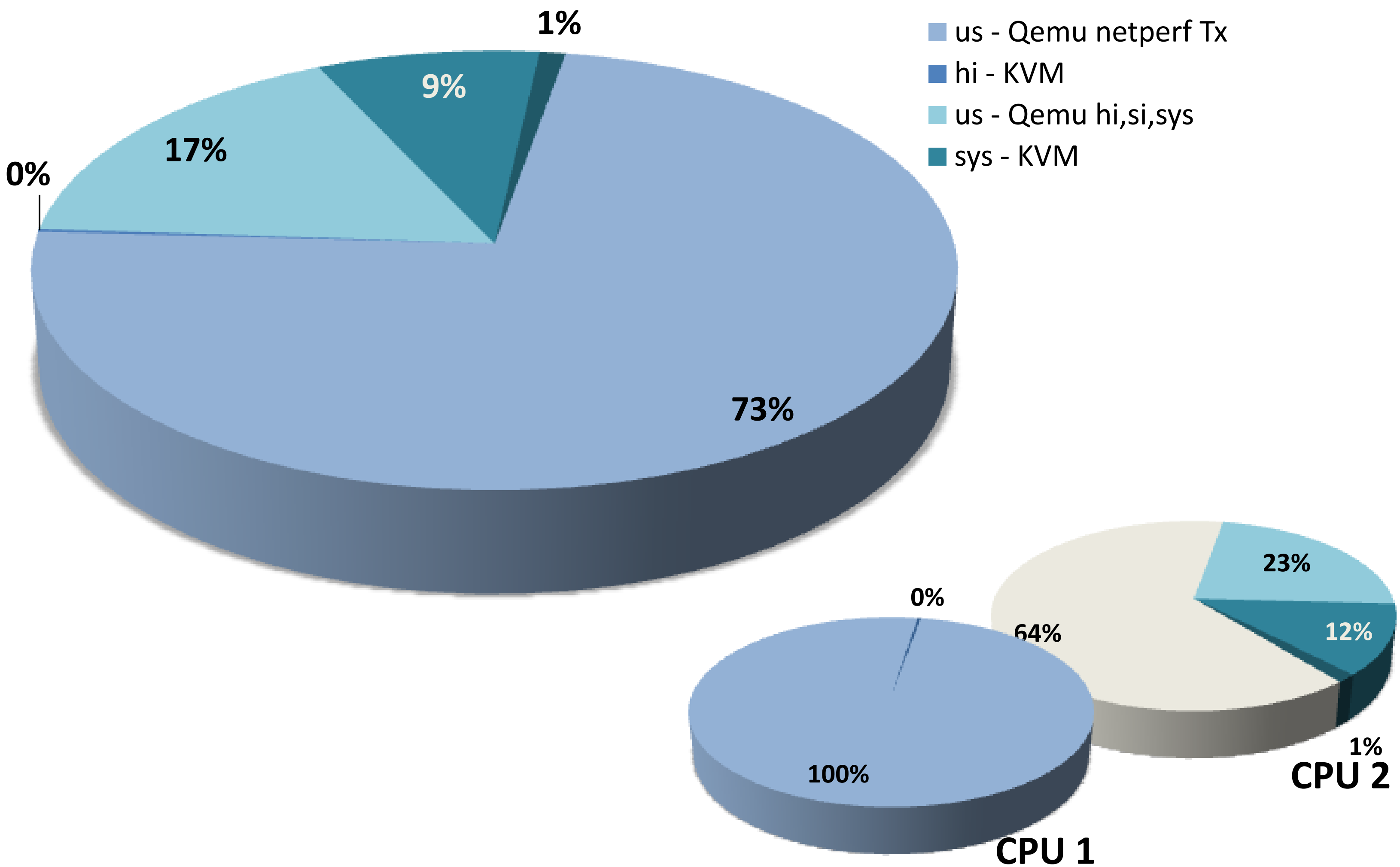


NetPerf UDP Results – cores to reach line rate (10G)

	Throughput (Mb/s)	Core load (HW Threads)	Degradation (%)
Tx physical interface	9550.4	6.23	-
Tx macvlan	9506.2	8.8	41.79
Tx guest (using macvtap)	9470	13	110.27
Rx physical interface	9570.4	4.73	-
Rx macvlan	9583.1	4.79	1.21
Rx guest (using macvtap)	9577.5	10.9	130.21

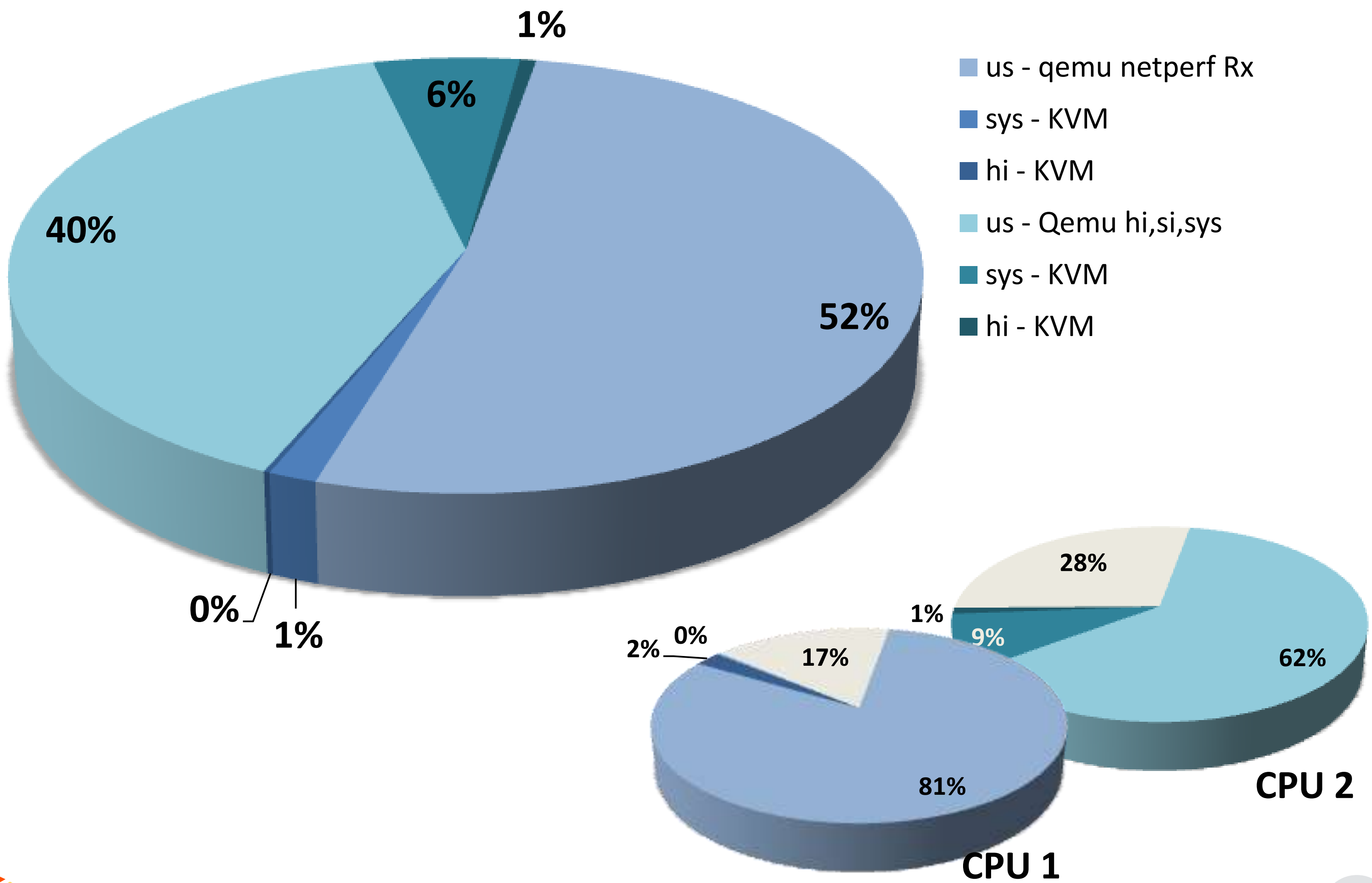
Netperf Tx CPU breakdown – PCI direct assignment

1 netperf flow, **2.1526 Gb/s, 2.306 GHz**



Netperf Rx CPU breakdown – PCI direct assignment

1 netperf flow, **2.3369 Gb/s, 2.619 GHz**



Mechanisms for investigation and debug

- KVM PPC provides exit timing information
  - Uses standard DebugFS
  - Provides timing information via /sys/kernel/debug/kvm/
    - dec, doorbell, dsi, ext_intr, guest doorbell, inst_emu, mmio, isi, itlb_r, itlb_v, dtlb_r, dtlb_v

- KVM components provide debug information
  - Uses standard dynamic debug
  - Filter messages by function/file/line via <debugfs>/dynamic_debug/control
  - Components provide debug information in dmsg
    - In-kernel mpic
    - vhost-net

- Ftrace – Display time spent in each function

- KVM provides gdb debugging support for guests

Conclusions

- The performance overhead when running on a hypervisor is workload and application dependent
- CPU intensive operations have low overhead
- Memory access performance depends on type of access and HW/SW support
  - Balancing system for optimal memory access results requires system analysis and fine tuning
- I/O bandwidth depends on available HW/SW support, system configuration and balancing
  - In-Kernel emulation reduces the number of context switches and improves performance
  - Direct assignment offers better performance

Introducing The QorIQ LS2 Family

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

QorIQ LS2 Family

Key Features

SDN/NFV Switching

Data Center

Wireless Access

Unprecedented performance and ease of use for smarter, more capable networks

High performance cores with leading interconnect and memory bandwidth

- 8x ARM Cortex-A57 cores, 2.0GHz, 4MB L2 cache, w Neon SIMD
- 1MB L3 platform cache w/ECC
- 2x 64b DDR4 up to 2.4GT/s

A high performance datapath designed with software developers in mind

- New datapath hardware and abstracted acceleration that is called via standard Linux objects
- 40 Gbps Packet processing performance with 20Gbps acceleration (crypto, Pattern Match/RegEx, Data Compression)
- Management complex provides all init/setup/teardown tasks

Leading network I/O integration

- 8x1/10GbE + 8x1G, MACSec on up to 4x 1/10GbE
- Integrated L2 switching capability for cost savings
- 4 PCIe Gen3 controllers, 1 with SR-IOV support
- 2 x SATA 3.0, 2 x USB 3.0 with PHY

See the LS2 Family First in the Tech Lab!

4 new demos built on QorIQ LS2 processors:

- Performance Analysis Made Easy
- Leave the Packet Processing To Us
- Combining Ease of Use with Performance
- Tools for Every Step of Your Design

www.Freescale.com