IO Virtualization

Kedar & Ozzie

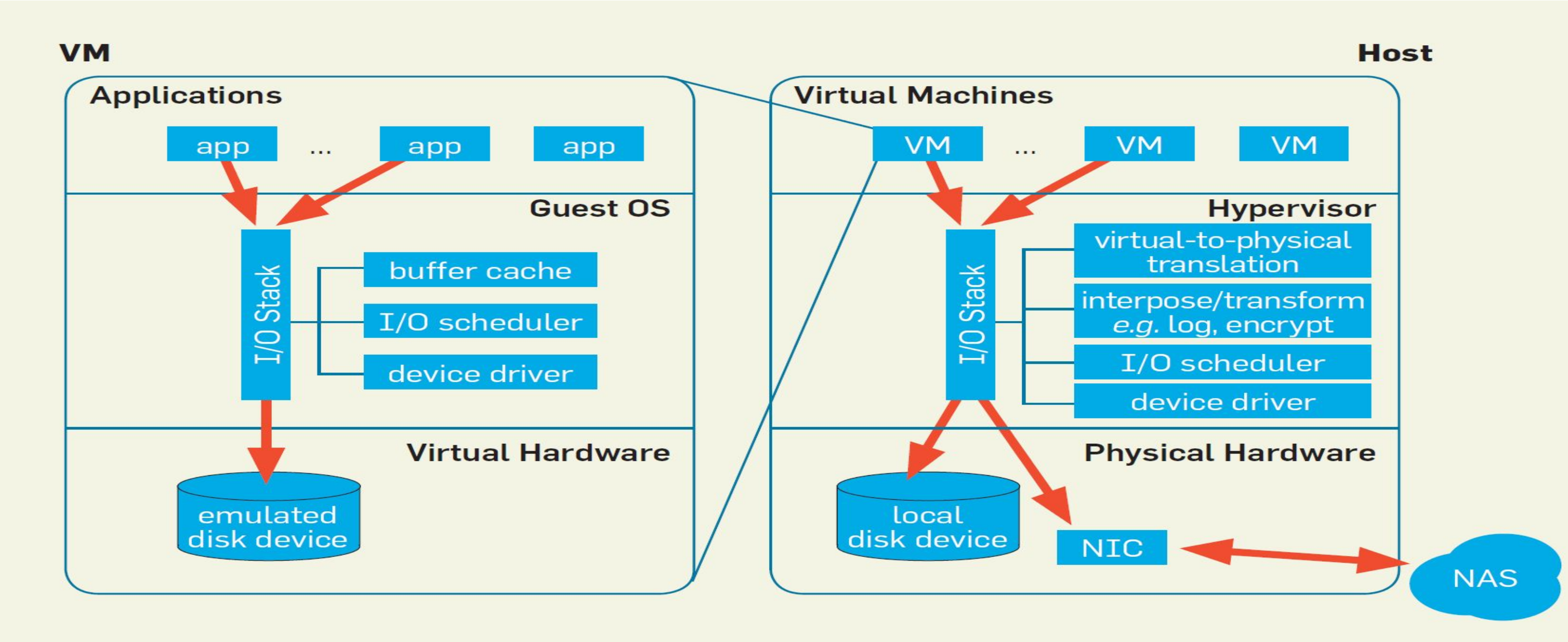
Overview

- Benefits
- Challenges
- Full Virtualization
- Paravirtualization
- Front-ends, Back-ends
- Pass through mode

Virtualization : Review

- Create a Virtual machine that can emulate all hardware resources
- Present an abstract or emulation of resource to outside world
- Encapsulate the physical resource
- Map logical resource with a physical resource (one-one, many-one, one-many)
- Advantages - Efficient utilization of resources, scalability, security

IO Virtualization



Source: Paper by Carl Waldspurger

Examples of IO Virtualization

- Computer Storage
  - Logical disk in PCs backed by partition or storage on network
- Computer Networking
  - Virtual private N/W - isolation created using cryptographic methods underlying is the public internet

IO Virtualization

- Encapsulates physical IO
- Decouples Virtual IO from Physical IO (enables portability)
- Introduce a level of indirection between abstract and concrete

Two techniques to handle IO Virtualization - software or hardware support

We will cover the software support for IO Virtualization.

Benefits

- Enables hypervisor to encapsulate entire state of VM
- Hypervisor can encode state of IO
  - Suspend VM (source server)
  - Store the encoded representation (copy to target server)
  - Resume execution at a later point
- Provide one-one, many-one, one-many mappings
- Allow hypervisor to add new features not supported by physical IO
  - Replicate data on storage devices
- Optimization to the memory images of VMs

Challenges

- Ensure good IO performance despite layer of indirection and interposition
  - IO opertions need to traverse 2 IO stacks (guest , hypervisor)
- Preserving semantics for virtual devices and interfaces
- Ensuring IO performance despite overhead due to additional functionalities added by hypervisor like security checks on n/w packets , encrypting disk writes.
- Prevent VM from monopolizing the resource and avoid scheduling delays
- Scheduling could impact VM performance
  - Contention for CPU resources could decrease TCP network performance.
  - TCP connections define RTT for flow control. CPU time-multiplexing distorts RTT, congestion windows grow slowly, degrades throughput.

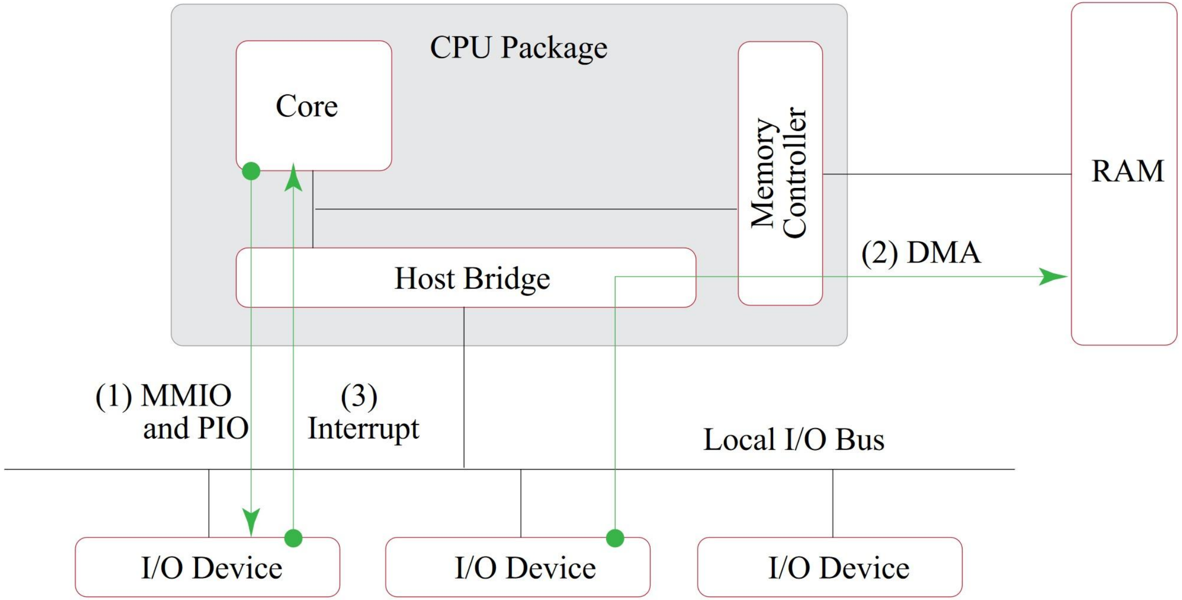
Emulation [Full Virtualization]

- Guest OS believes exclusive control on IO devices.
- Hypervisor cannot allow that. (Guest OS on newly starting VM might initialize the IO devices if allowed direct access)
- Hypervisor traps the IO related operations and emulates them

Types of Interaction between OS/Device

- OS discovers and talks to devices through MMIO & PIO operations
  - Bios associates addresses with registers of IO devices. If addresses from memory address space - MMIO, if separate address space - PIO
- Devices respond by triggering interrupts, reading/writing from/to DMA

Interactions with IO devices



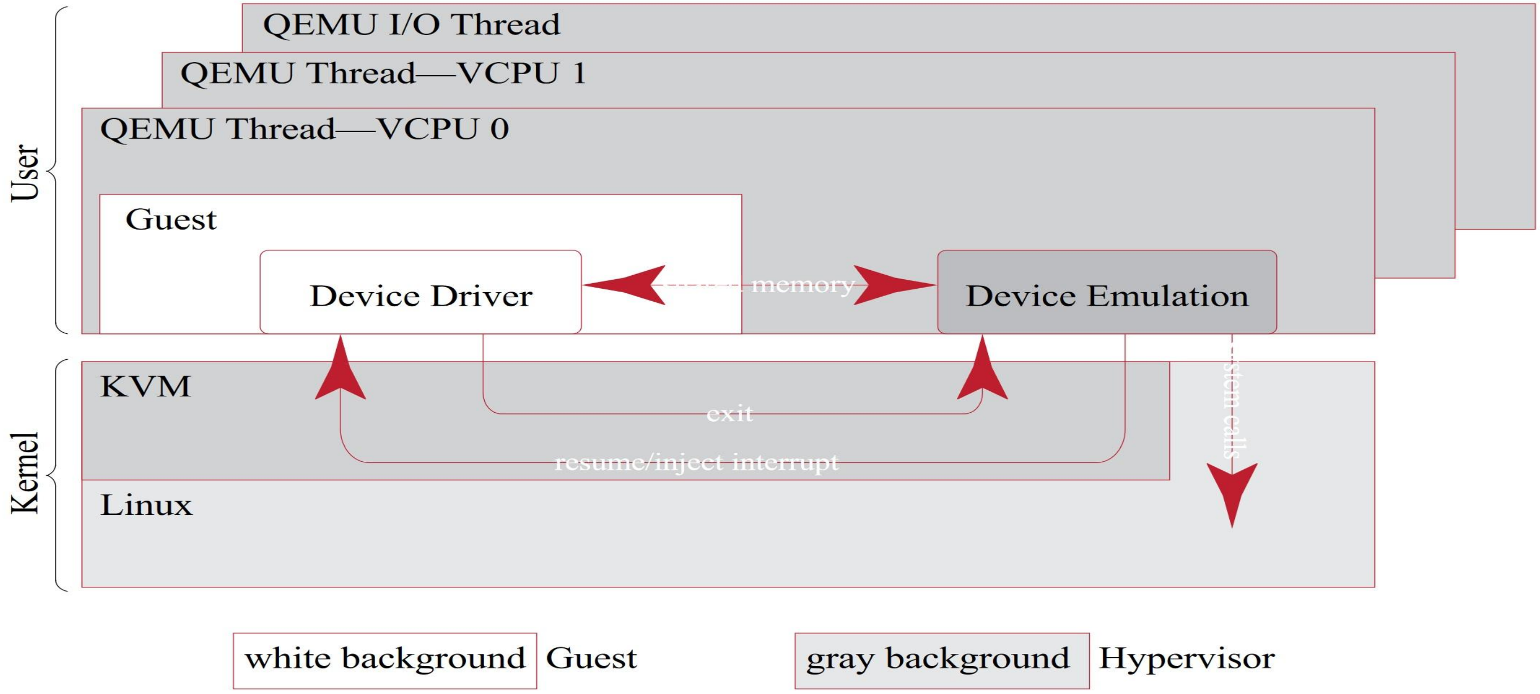
Source: H/W & S/W support for Virtualization

Emulation [Full Virtualization]

Hypervisor Virtualizes by :

- Trapping every MMIO , PIO operations of guest OS
  - MMIO - regular load /store instructions from/to guest memory pages.
  - Hypervisor traps by mapping pages as reserved/not-present (for both load/store) or as read-only for store
  - Guest PIO are privileged instructions, hypervisor configures guest's VMCS to trap them
- Emulating - interrupts, read/write to DMA

Linux Implementation



Source: H/W & S/W support for Virtualization

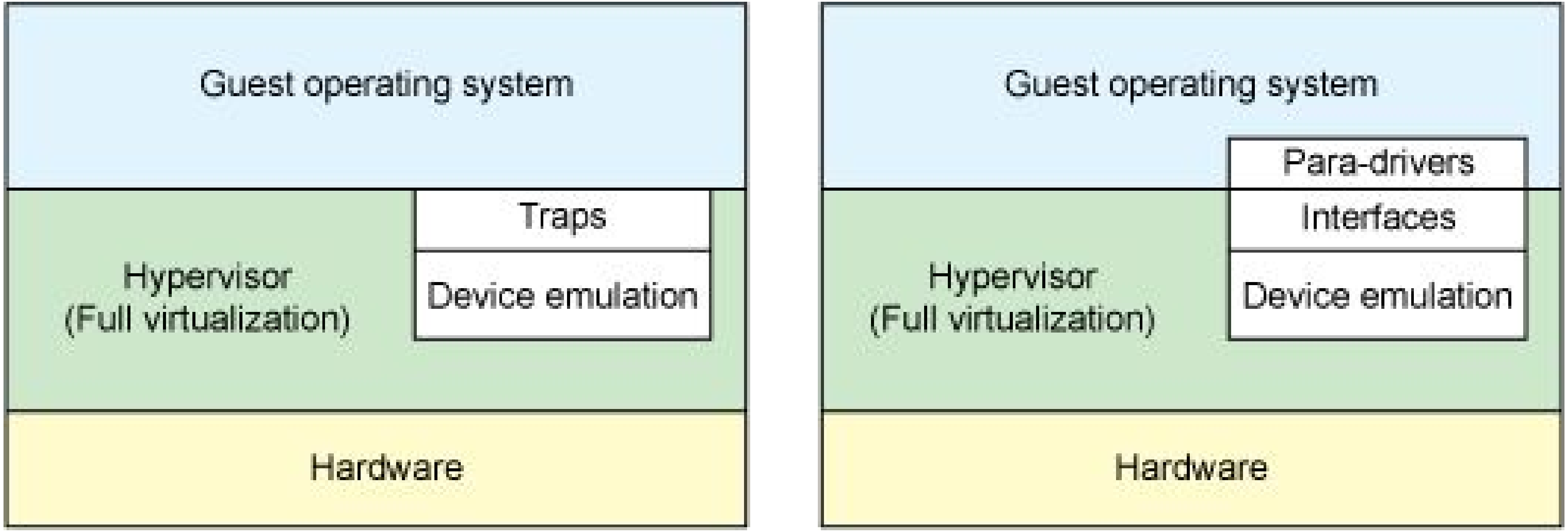
Linux Implementation

- Each VM encapsulated in Qemu process.
- Each virtual core(VCPU) represented by a thread
  - Each VCPU thread has 2 execution contexts - guest VM and host QEMU
  - Host context - for handling exits of guest VCPU context.
- Qemu creates “IO thread” for each virtual device.
  - IO thread handles asynchronous activity like handling network packets
- Here , there are 2 VCPUs and one virtual device
- Guest VM device driver issues MMIO/PIO instructions to drive the device - directed at read/write protected memory locations - suspend VCPU context - invoke KVM
- KVM relays events to same thread but to the host execution context
- Events are handled by the device emulation layer of host context through regular system calls
- Device emulation layer emulates DMA by read/write from guest IO buffers - accessible through shared memory
- Resumes guest execution context via KVM injecting interrupts to signal the guest about IO operation

I/O Paravirtualization

- Drivers and hardware were not designed for virtualization
    - Every operation can result in numerous traps
    - Layout of registers in memory tightly packed
- Redesign virtual device and its interactions
    - Minimize overhead associated with emulate
    - Guest uses specialized driver for optimized virtual hardware
- Performance comes at cost of abstraction
    - Installation of paravirtual drivers required
    - Drivers must be implemented for each type of OS
- Can be supported with emulation
    - Usually for legacy reasons

I/O Paravirtualization



Source: Virtio: An I/O virtualization framework for Linux

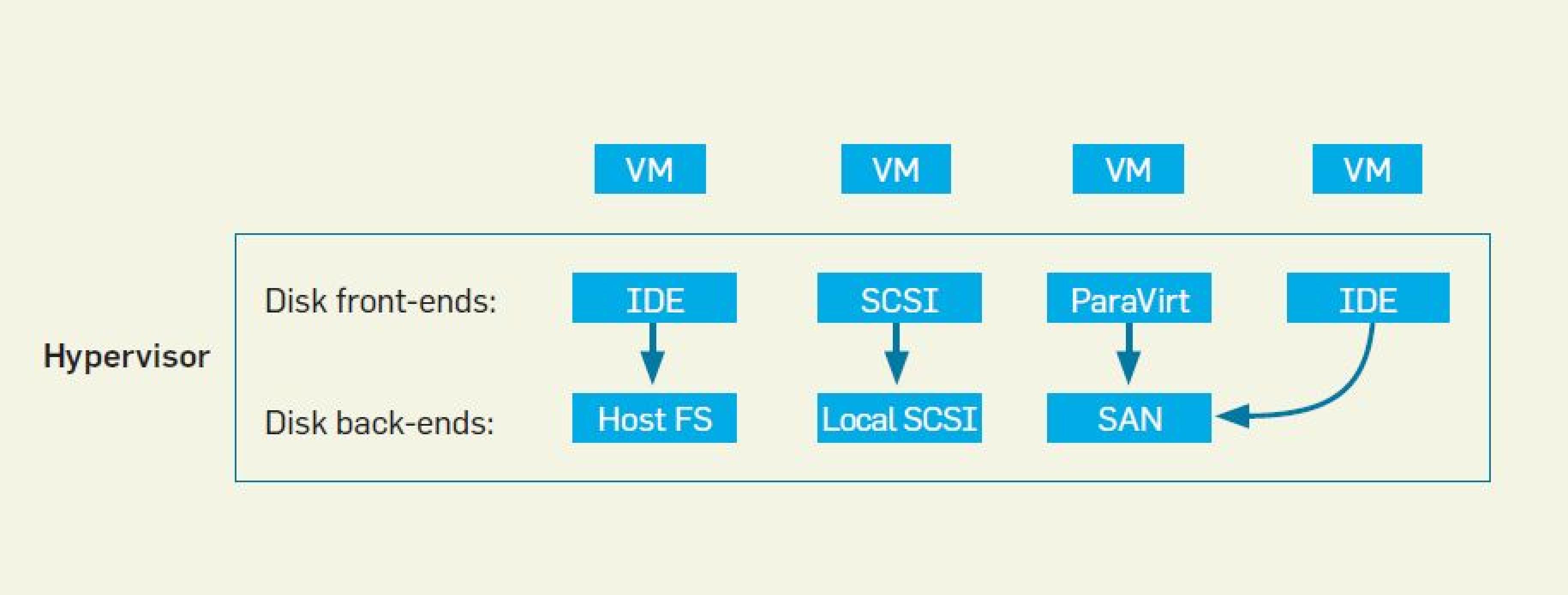
I/O Paravirtualization: Implementation

- Minimize the number of exits
  - Virtio uses *virtqueues* to perform **explicit** exits
  - Two modes so Guest and Host don't step on each other
- Utilize a shared memory segment
  - Write commands for emulation layer to access
- Reduce number of context switches
  - Vhost-net handles packet processing in Linux kernel
  - Operates with virtio-net enhancement
- Usually results in major performance enhancements
  - Virtio-net much better than e1000 (throughput, exits/secs, interrupts/secs)

Front-Ends and Back-Ends

- Front-End: Device interface
  - Guest driver and emulated device
- Back-End: Device implementation
  - Host physical resources
- Decouples ends allow for “plug-and-play”
  - Disk storage backed by file
  - Use new HW for Guest assuming older HW
- Additional functionality easily interposable
  - Packet sniffing, disk encryption, snapshot logging
- Active research to reduce overhead and interpose functionality

Front-Ends and Back-Ends



Source: I/O Virtualization

Pass-Through Mode

- Guest is able to access the device directly
- Virtually eliminates all emulation and back-end overhead
- Each device is limited to use by one VM
- Introduces strong coupling between Guest and hardware
  - Inability to interpose processes
  - Option of live migration no longer viable
- Issue of “correct” and “safe” DMA access not solved
- Active area of research to make viable
  - Hardware support making progress here

I/O Virtualization

- Emulation (Full Virtualization)
  - Best option for correctness and abstraction
  - High performance cost
- Paravirtualization
  - Optimize driver and virtual device interaction
  - Guest is “aware” of virtualization
- Pass-Through Mode
  - Best option for performance
  - Strong coupling with hardware