System Virtualization and OS Virtual Machines

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

History of Virtual Machines

- VM introduced in the sixties on IBM/370 series
- Co-Designed VM: IBM AS/400
  - High level ISA including I/Os
  - Proprietary CISC → PowerPC
- Application VMs
  - Sun Java, Microsoft Common Language Infrastructure
- OS VMs
  - VMware (Windows/Linux on Intel)
  - Connectix (Windows/PC emulation on Mac OS)

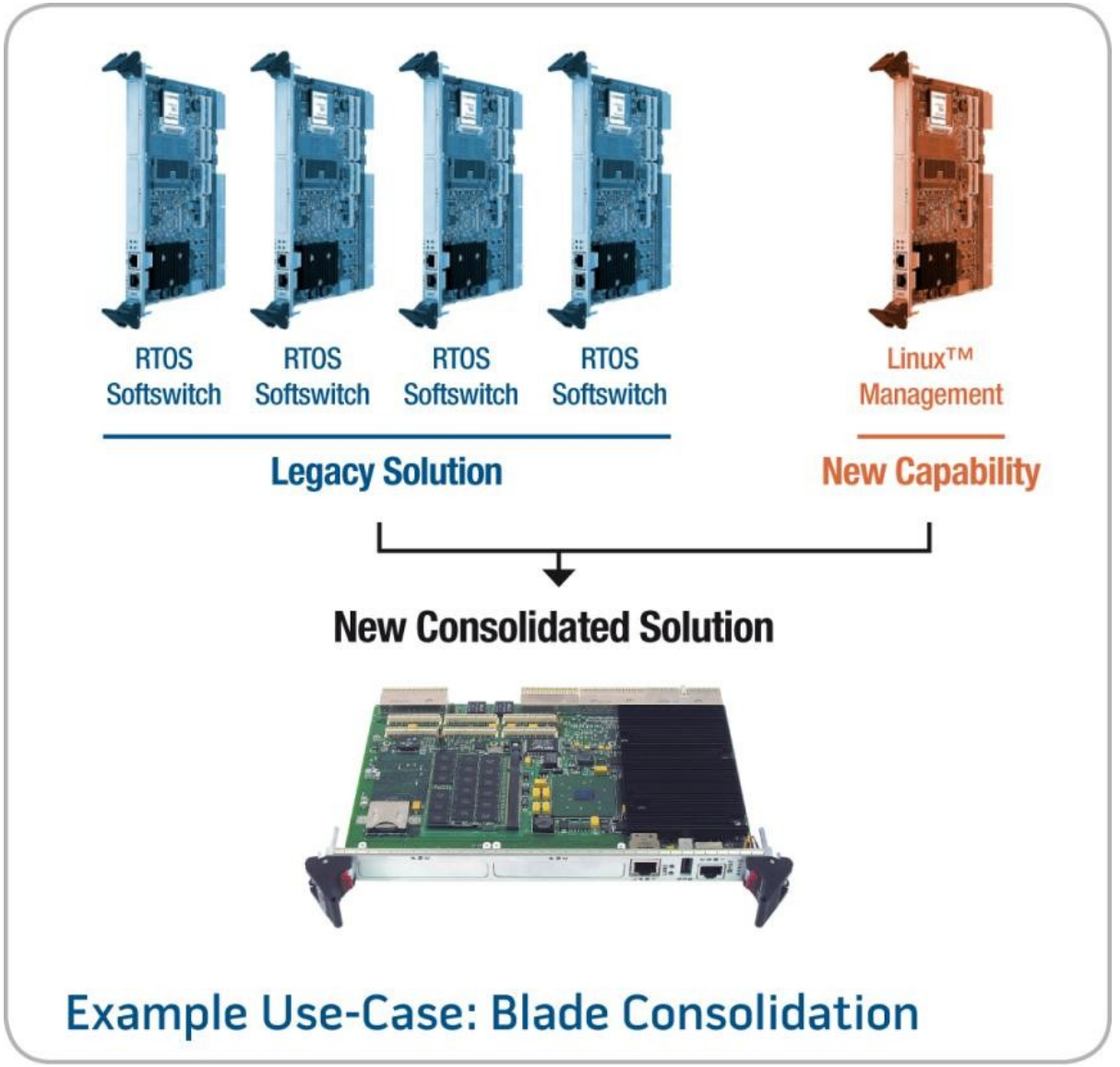
Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Goals of System Virtualization

- Reduction of Total Cost of Ownership (TCO)
  - Increase utilisation of server resources
- Reduction of Total Cost of Functioning
  - Energy consumption
  - Cooling
  - Occupied Space
- Hardware Consolidation
  - Reduction of Build Of Material (BOM) for high-volume low-end products

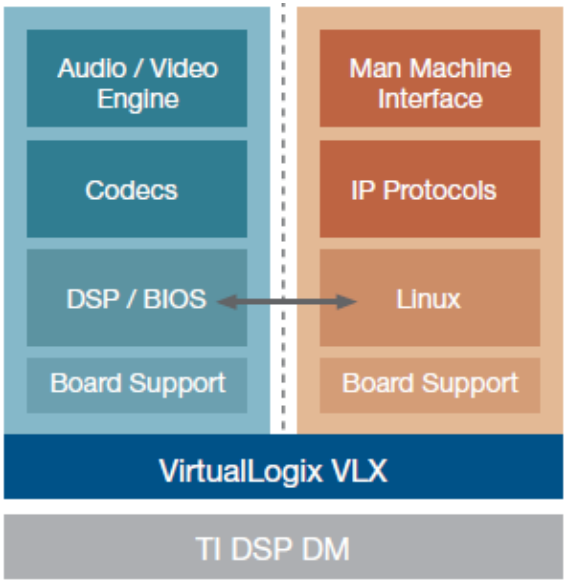
Virtualization in high-throughput network equipments



Virtualization in Multimedia devices

- Reduction of Build Of Material (BOM) for high-volume low-end products
  - No need for a General Purpose Processor
    - ~ 20 to 25 % BOM reduction
  - Run Linux together with OS supporting

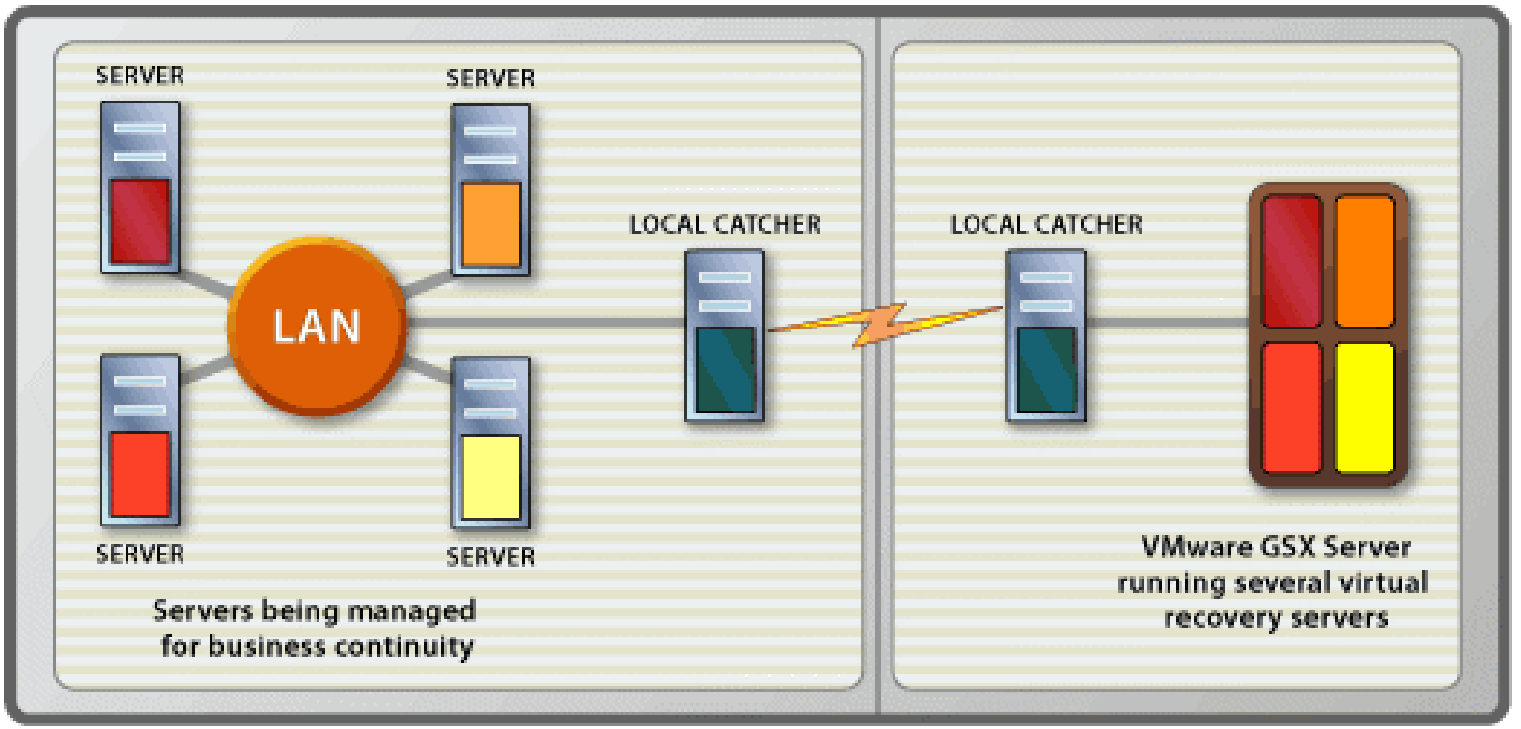
    Codecs on a single TI DSP
  - Leverage Linux environment
  - Reuse existing DSP software



Usages of Virtual Machines

- Server virtualization
  - Web sites hosting
- OS fault recovery
- OS kernel development
  - Test machine = development host
- OS/kernel education & training
- Keep backward compatibility of legacy software
  - Hardware no more available
- Run applications not supported by host OS

Recovery Servers



Multi-Core CPU Issues (1)

- CPU power gain
  - No more achieved through Frequency/Speed increase
  - But obtained with higher density & multi-core chips
- Many RTOS designed with mono-processor assumption
  - Adding multi-processor support is complex & costly
  - Scaling requires time, at best...
- Legacy RT applications also designed for mono-processor
  - Adaptation to multi-pro even more difficult than RTOS

Multi-Core CPU Issues (2)

- OS virtualization allows to run simultaneously on a multi-cores CPU multiple [instances of] mono-processor OS's
- Each OS instance is run in a [mono-processor] Virtual Machine assigned to a single CPU core
- No need to change legacy software
- Scalability managed at virtualization level

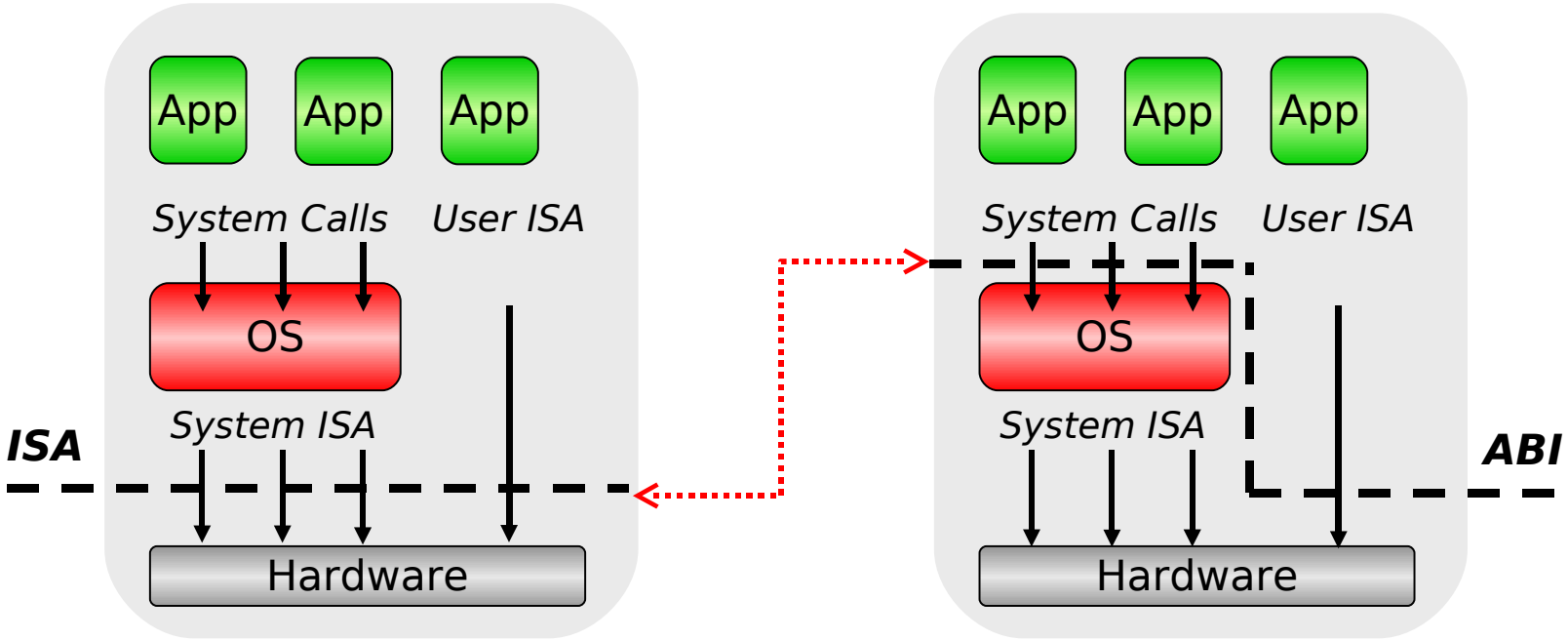
Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

System Virtualization Principles

- Run multiple OS's on the same machine
- By design, an OS assumes to have full control over all physical resources of the machine
- Manage sharing/partitioning of machine resources between Guest OS's
  - CPU
  - Physical memory & MMU
  - I/O devices

Machine Interfaces



- ISA = **I**nstruction **S**et **A**rchitecture
  - System level interface
  - All CPU instructions, memory architecture, I/O
- ABI = **A**pplication **B**inary **I**nterface
  - Process level interface
  - User-level non privileged ISA instructions + OS systems calls

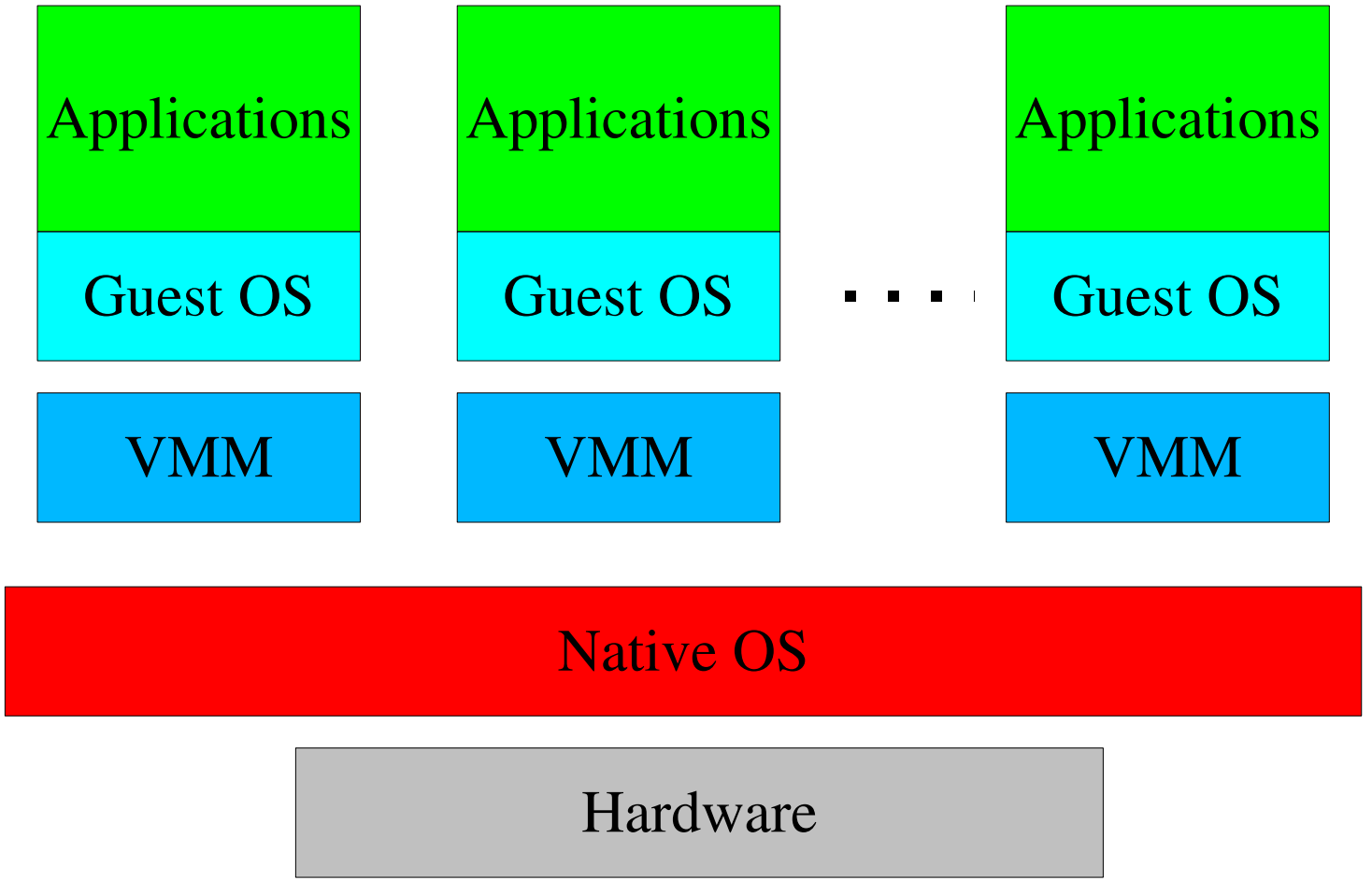
Virtualization Taxonomy

- Process level virtualization
  - Emulation of Operating System ABI
  - Virtual Servers
- System level virtualization
  - Standalone / Hosted Virtualization
  - Machine Emulation / Machine Virtualization

Hosted versus Standalone Virtualization

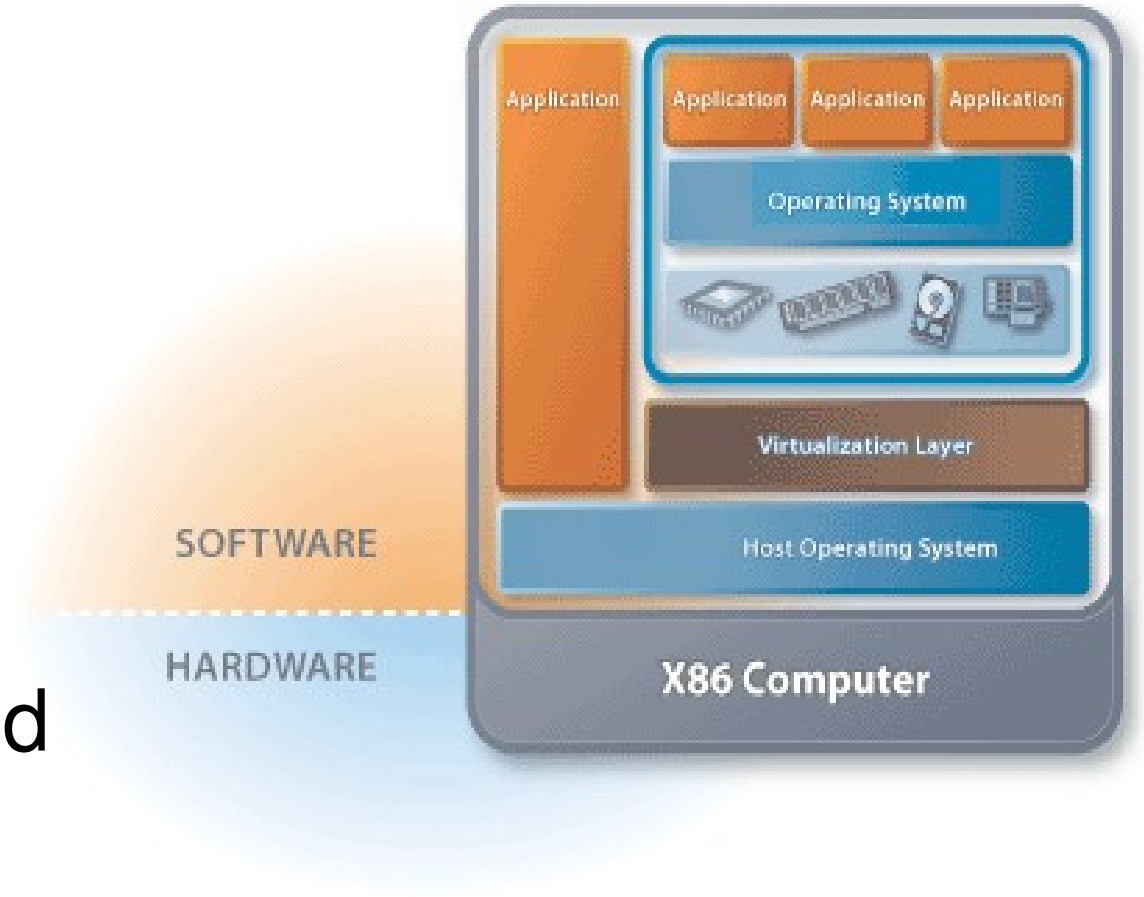
- Hosted Virtualization
  - Hosted VM Monitor (VMM) runs on top of native OS
  - VMware WKS, Microsoft VirtualPC, QEMU, UML
- Standalone Virtualization
  - VMM directly runs on bare hardware
  - VMware ESX, IBM/VM, Xen, VLX, KVM
- OS run in a VM is named a Guest OS

Hosted Virtualization

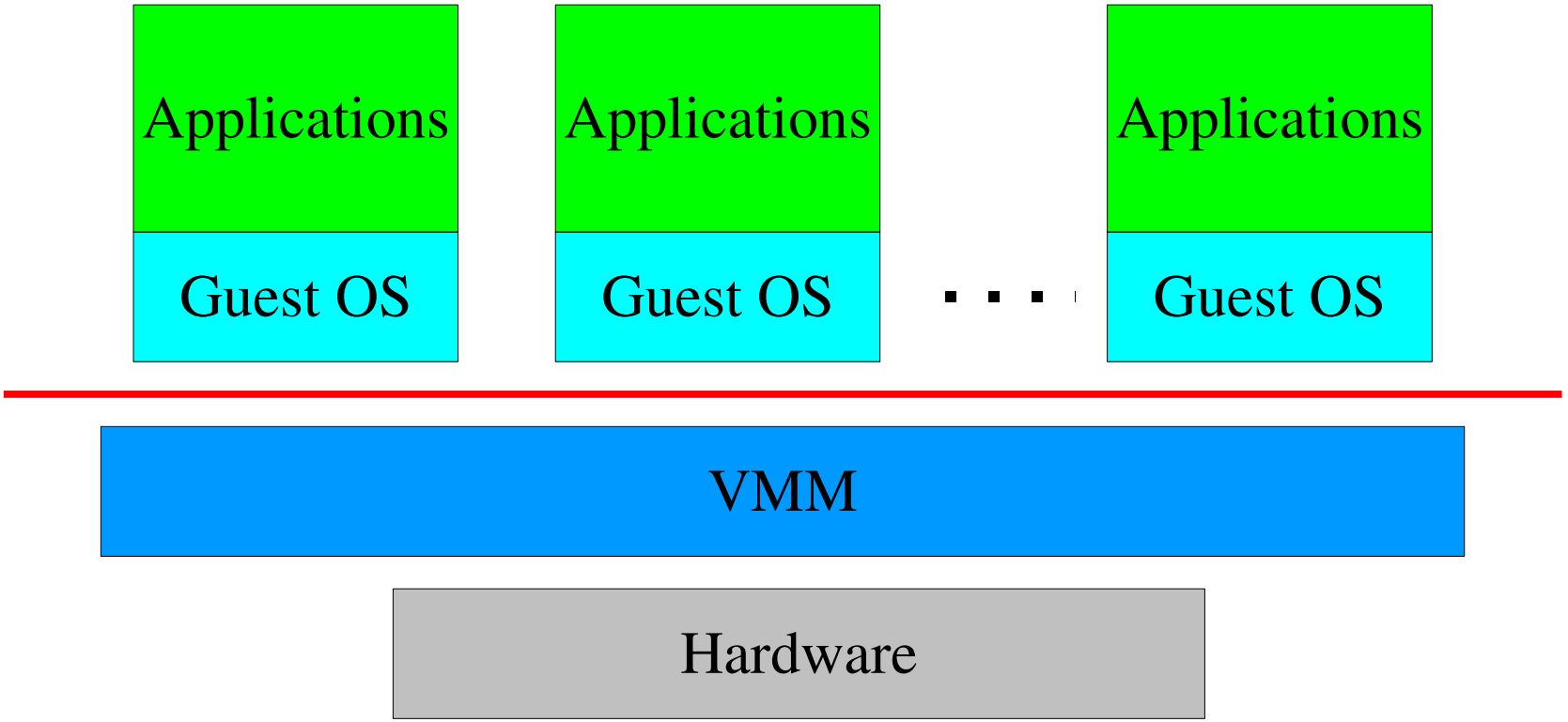


Example : VMware Workstation

- Hosted VM
- Unmodified OSes
- Specific device drivers
- X86 only
- Guest OS executed in user mode

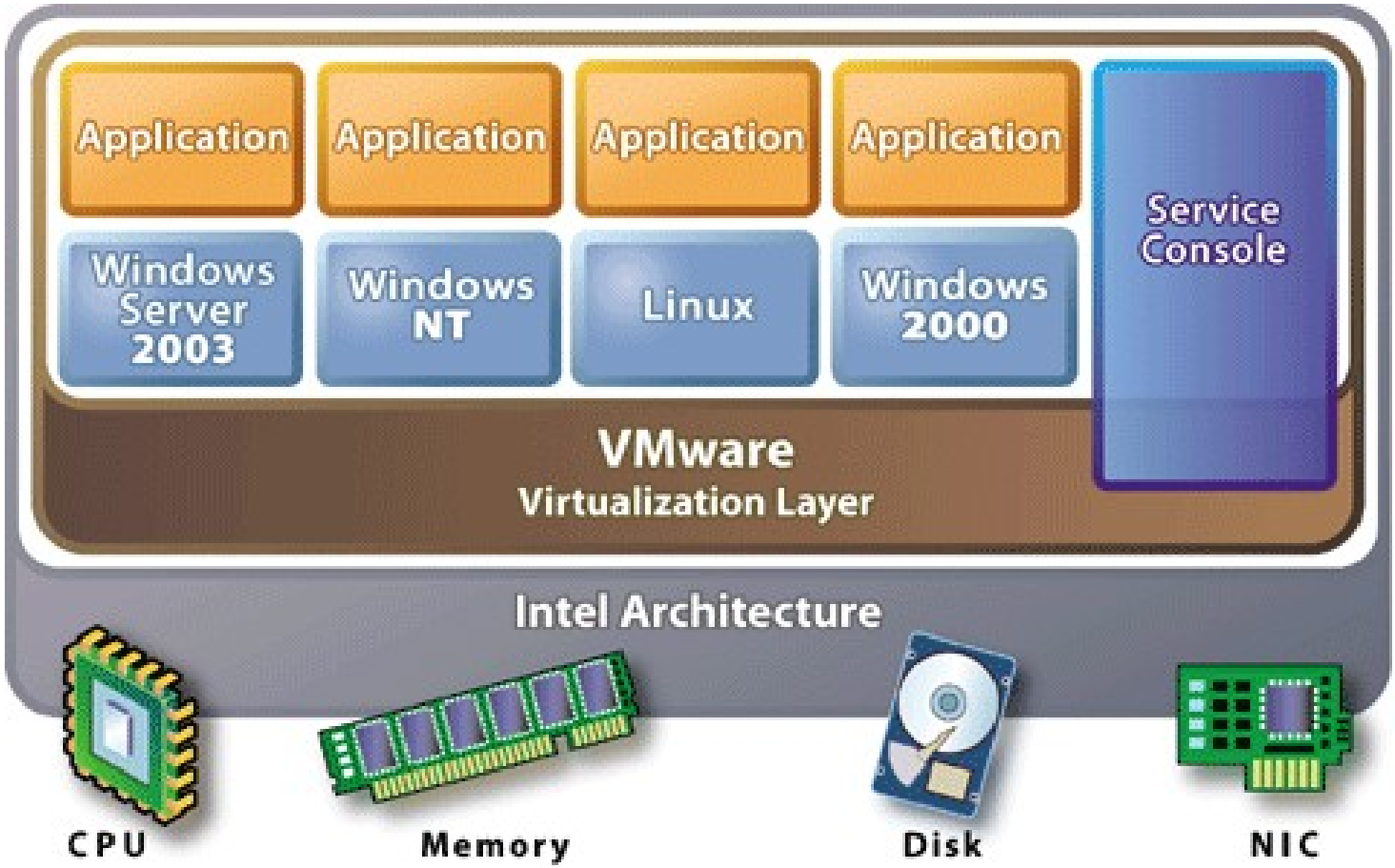


Standalone Virtualization



VMware ESX

- Standalone VM
- Supports unmodified OS binaries
  - Configuration with appropriate device drivers
- X86 only
- OS hosted in user mode



Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Process level ABI Emulation

- Goal: execute binary applications of a given system **X** on the ABI of another system **Y**
- Emulate system **X** ABI on top of system **Y** ABI
  - Emulation done by application-level code
- System **Y** must provide services equivalent to those of system **X** (file system, sockets, etc.)

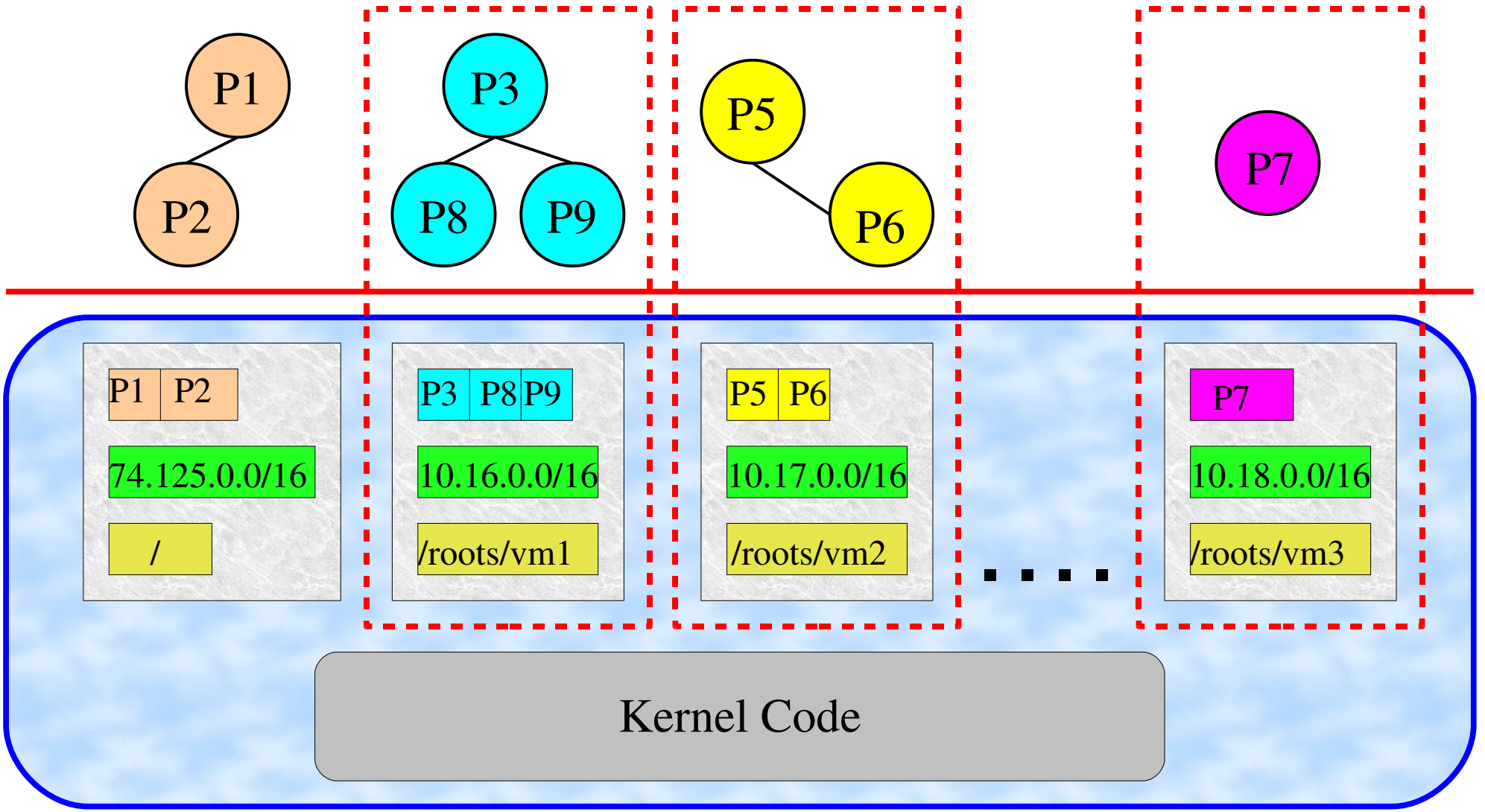
Process Level (ABI) Emulators

- Wine - **Win**dows **E**mulator on Unix/Linux
  - Windows API in userland
  - Adobe Photoshop, Google Picasa, ...
- Cygwin
  - Unix emulation on Windows
  - POSIX library
  - Bash shell + many Unix commands
  - GNU development tool chain (gcc, gdb)
  - X Window, GNOME, Apache, sshd, ...

Virtual Servers

- Single OS kernel / Multiple resource instances
- Isolated kernel execution environments
  - Root file system
  - IP tables
  - Process for signals
- Solaris 10 Containers
- Linux-VServer
- FreeBSD Jail

Virtual Servers



Virtual Servers

- Pro's
  - CPU independent
  - Lightweight
    - Low memory footprint
    - Low CPU overhead
  - Scalable
- Con's
  - No OS heterogeneity (no GPOS/RTOS combination)
  - Single OS binary instance (common point of failure)
  - Intrusive: must modify OS & follow OS updates

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

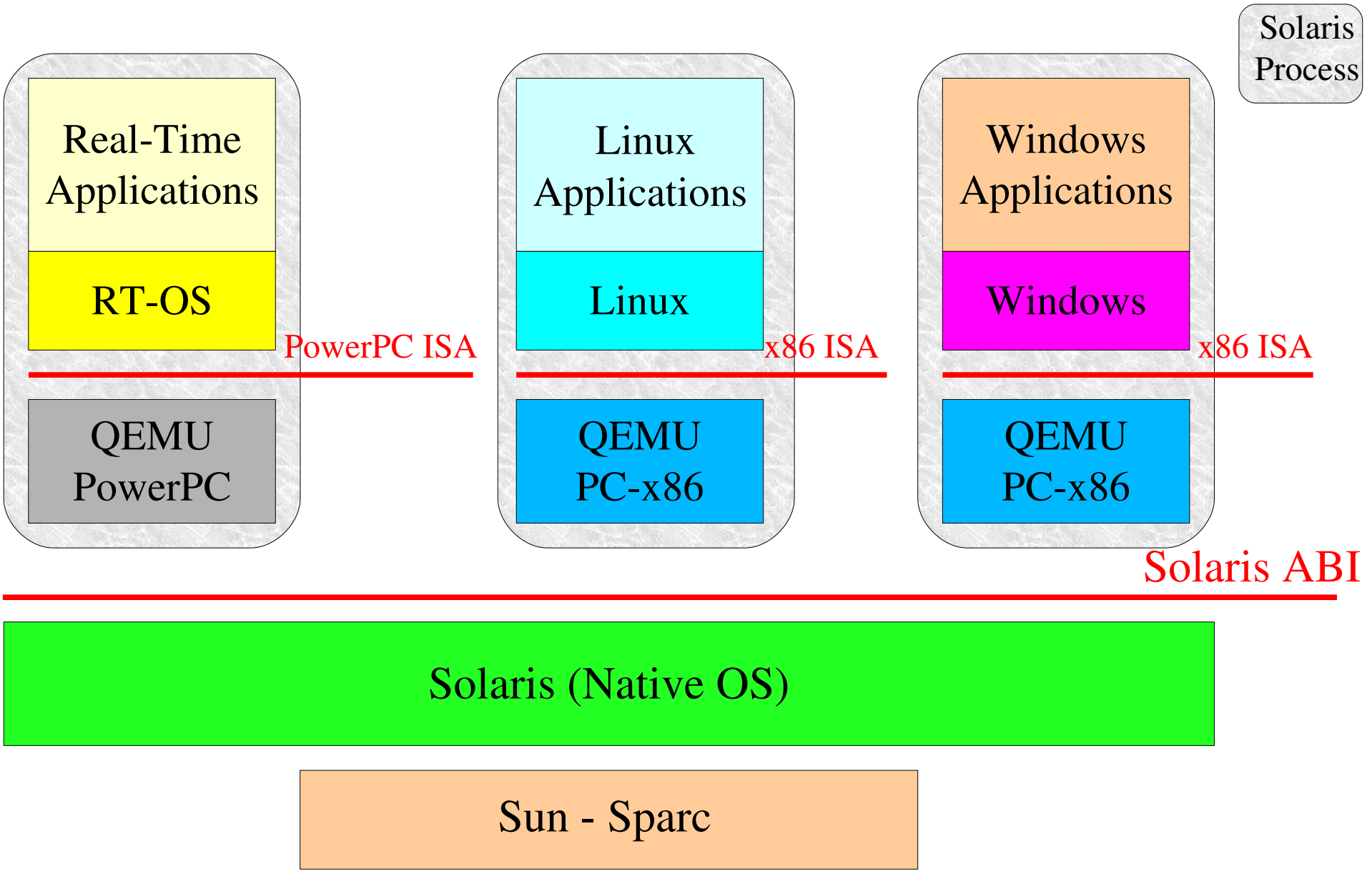
Transparent Hardware Emulation

- Run unmodified OS binaries
- Includes emulation of physical devices
- Cross ISA Emulation
  - QEMU
- Same ISA Emulation
  - VirtualBox (Intel x86)

Transparent Hardware Emulation

- Emulate machine X on top of machine Y
- Interpretation
  - 1 instruction of X executed by N instructions of Y
  - Huge slow down method (1/1000 if X ≠ Y)
- Dynamic Binary Translation
  - Convert blocs of X instructions in Y instructions
- Application-level emulator runs on a native OS
- One VM running a single Guest OS

QEMU Architecture



QEMU: Hosted Hardware Emulator

- Cross ISA Emulation
  - Emulate machine X on top of machine Y
- Interpretation + translation when X ≠ Y
- Intel x86, PowerPC, ARM, Sparc architectures
- Emulation of SMP architectures
- Emulates physical I/O devices
  - Hard Disk drives, CD-ROM, network controllers, USB controllers, …
  - Synchronous emulation of device I/O operations

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Transparent Hardware Virtualization

- Share machine resources among multiple VMs
- Execute native/unmodified OS binary images
- Provide in each VM a complete simulation of hardware
  - Full CPU instruction set
  - Interrupts, exceptions
  - Memory access and MMU
  - I/O devices

Full CPU Virtualization

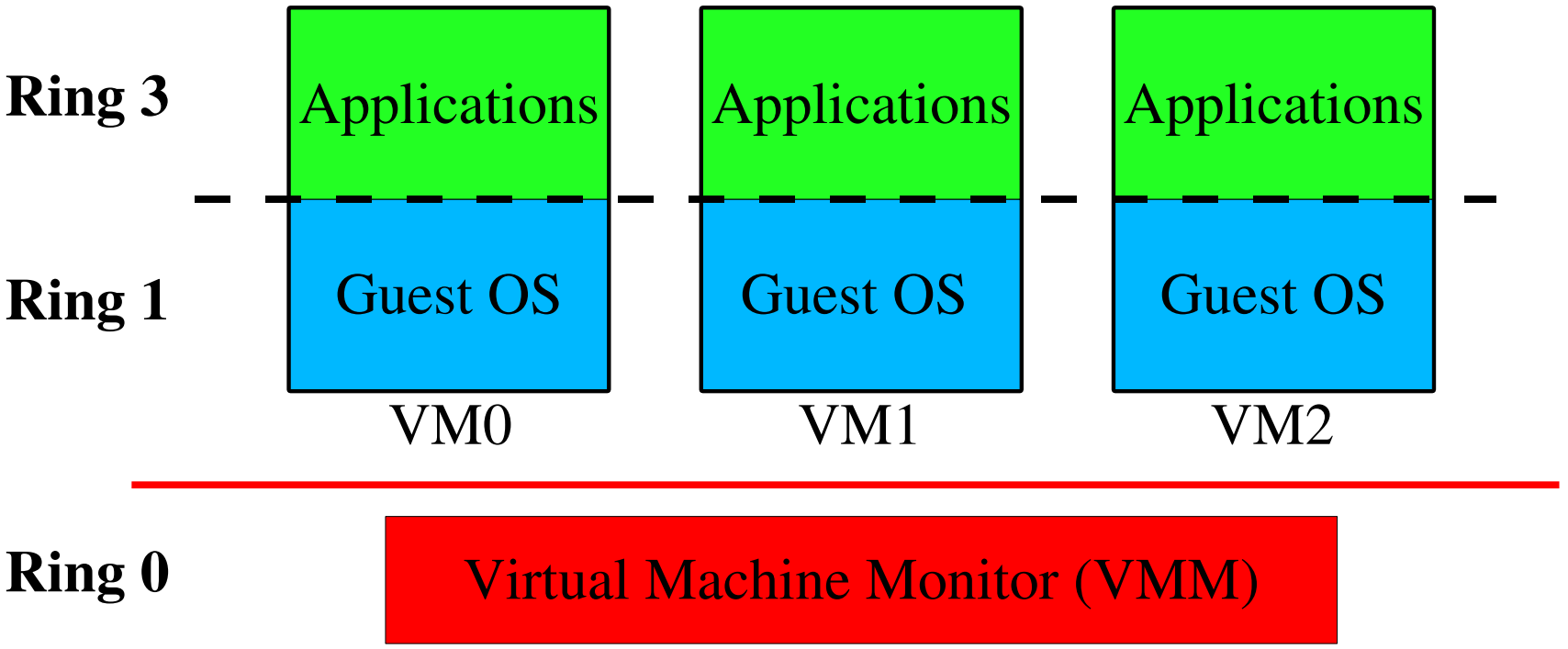
- Present same functional CPU to all Guest OSes
- VMM manages a CPU context for each VM
  - saved copy of CPU registers
  - representation of software-emulated CPU context
- VMM shares physical CPUs among all VMs
  - VMM includes a VM scheduler
    - round-robin
    - priority-based

Full CPU Virtualization

- Relationships between a VMM and VMs similar to relationships between native OS and applications
  - Guarantee mutual isolation between all VMs
  - Protect VMM from all VMs
- Directly execute native binary images of Guest OS's in non-privileged mode
- VMM emulates access to protected resources performed by Guest OSs

CPU Virtualization

- Run each Guest OS in non-privileged mode
  - Ex: ring compression on Intel x86



« Hardware-Sensitive » Instructions

- Interact with protected hardware resources
  - Privileged Instructions
  - Critical Instructions
- Cannot be directly executed by Guest OS's
- Must be detected and faked by VMM
- Dynamic Binary Translation of kernel code
  - Done once, saved in Translation Cache
  - Example: Vmware

Privileged Instructions Virtualization

- Only allowed in supervisor mode
  - Ex: cli/sti to mask/unmask interrupts on Intel x86
- When executed in non-privileged mode
  - CPU automatically detects a privilege violation
  - Triggers a “privilege-violation” exception
- Caught by VMM which fakes the expected effect of the privileged instruction
  - Ex: cli/sti
    - VMM does not mask/unmask CPU interrupts
    - records « interrupt mask status » in context of VM

Critical Instructions Virtualization (1)

- Hardware-sensitive instructions
- Ex: Intel IA-32 pushf/popf

```
pushf /* save EFLAG reg. to stack */
cli   /* mask interrupts => clear EFLAG.IF */
…
popf /* restore EFLAG reg. => unmask interrupts */
```

- When executed in non-privileged mode
  - The `cli` instruction triggers an exception caught by VMM => VMM record interrupts masked for current VM
  - But no exception for `popf` => VMM not aware of Guest OS action (unmask interrupts)

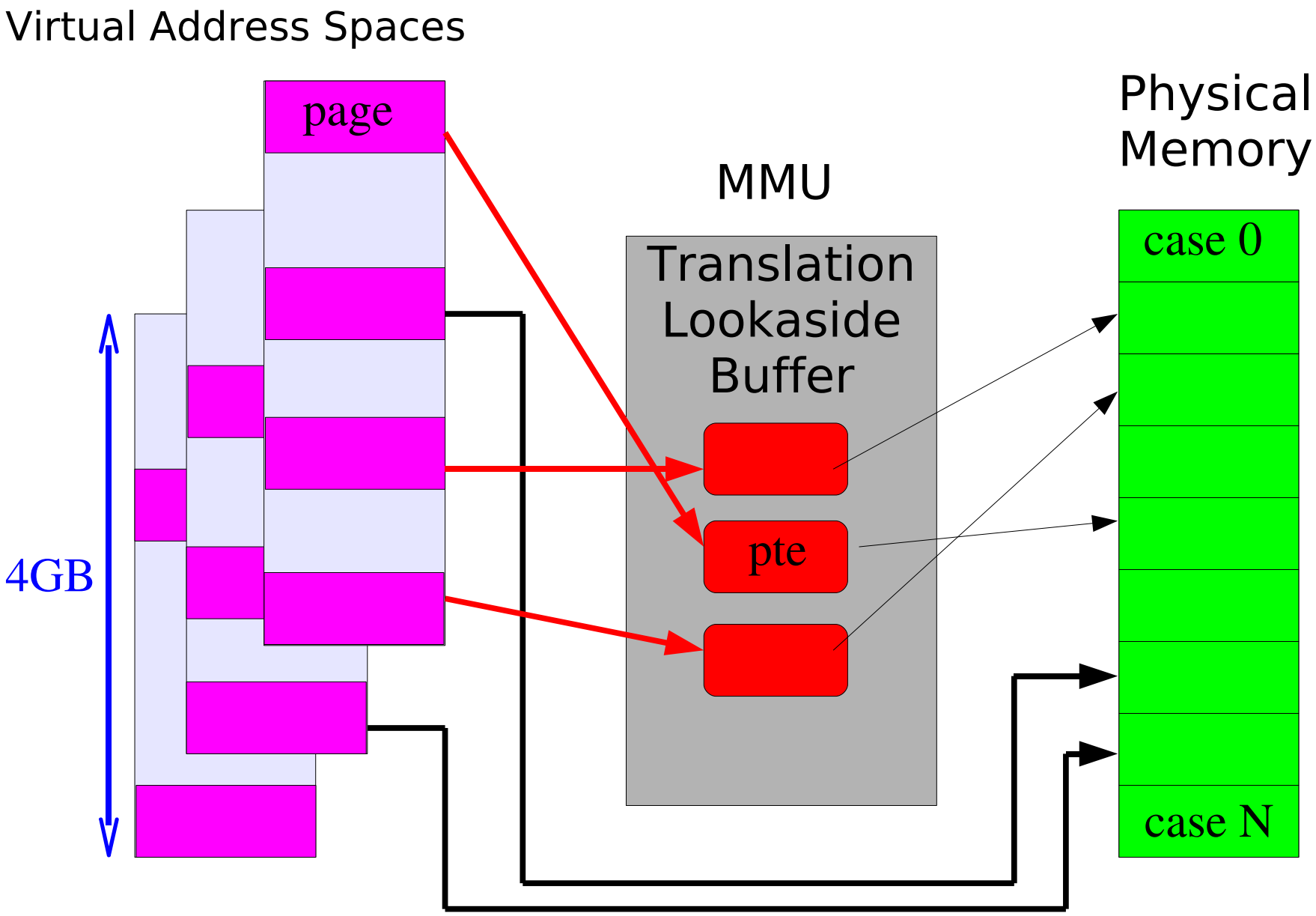
Critical Instructions Virtualization (2)

- Must be detected and emulated by VMM
- VMM dynamically analyses Guest OS binary code to find critical instructions
- VMM replaces critical instructions by a « trap » instruction to enter the VMM
- VMM emulates expected effect of critical instruction, if any.
  - Ex: pushf/popf combined with cli/sti instructions

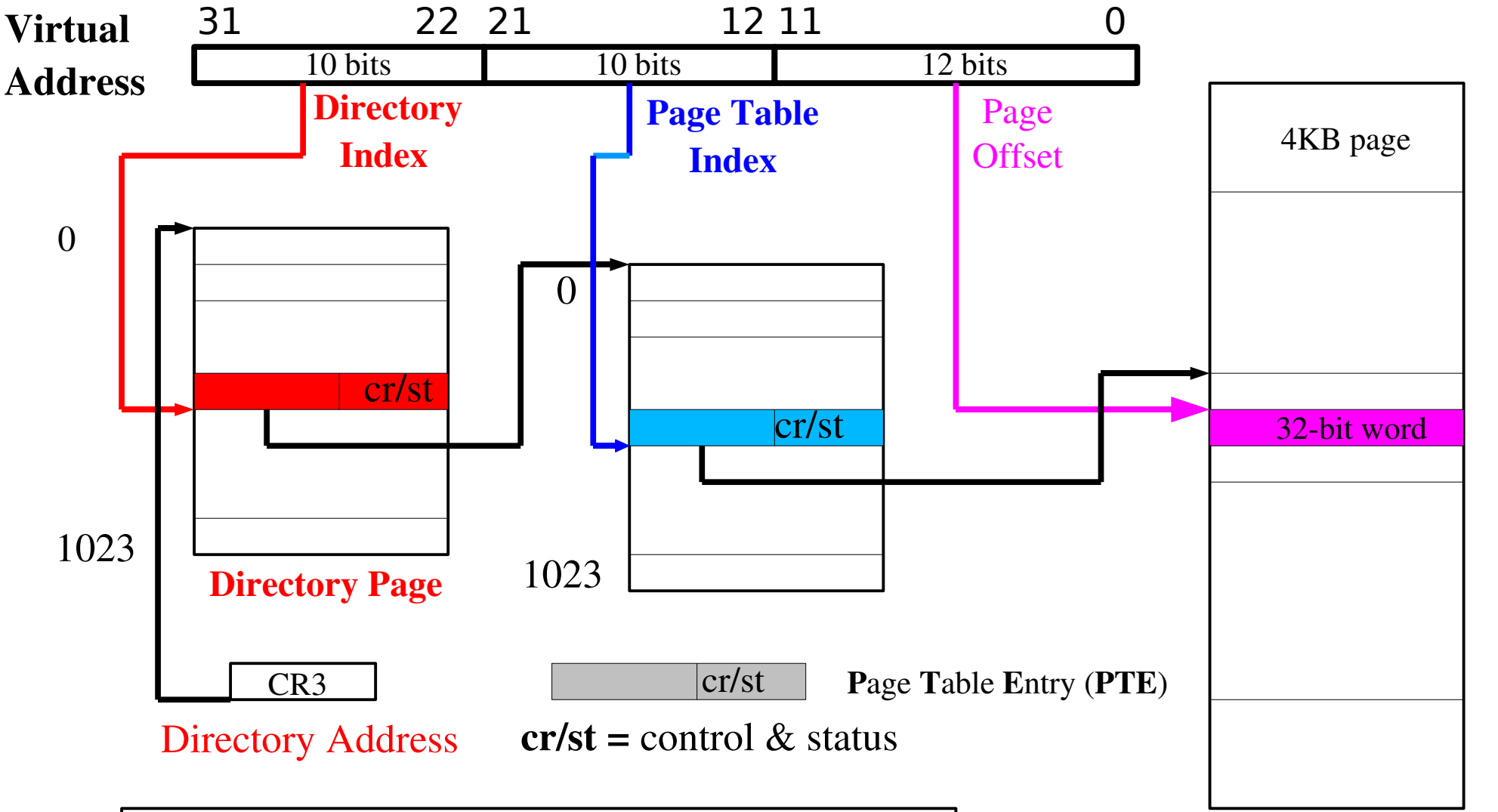
Full Memory Virtualization

- CPU include a Memory Management Unit (MMU)
  - Isolated memory addressing spaces
  - Independant of underlying physical memory layout
  - Run mutually protected applications in parallel
- Virtual Memory managed by OS kernel
  - Provides a virtual address space to each process
    - 4 GB on most 32-bit architectures (Intel x86, PowerPC)
  - Manages virtual page → physical case mappings
  - Manages « swap » space to extend physical memory

MMU & Virtual Address Space



Intel x86 MMU



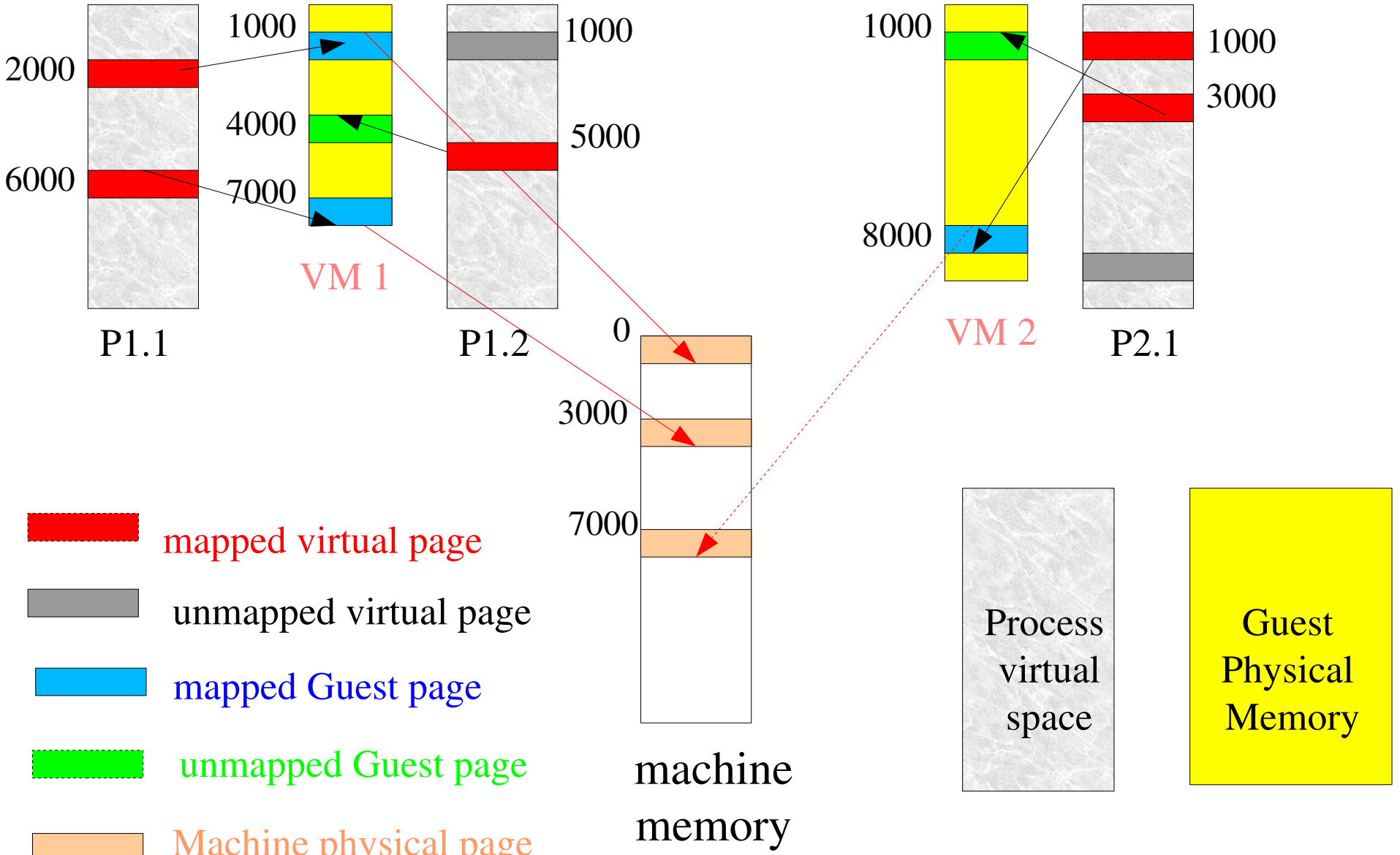
Memory Virtualization

- Machine Physical Memory
  - Physical memory available on the machine
- Guest OS Physical Memory
  - Part of machine memory assigned to a VM by VMM
  - ∑ Guest Physical Memory can be > Machine Memory
    - VMM uses « swap » space
- Guest OS Virtual Memory
  - Guest OS manages virtual address spaces of its processes

Memory Virtualization

- Guest OS manages Guest Physical Pages
  - Manages MMU with its own page entries
  - Translates Virtual Addresses into Guest Physical Addresses (GPA)
- VMM transparently manages Machine Physical Pages
  - Guest Physical Address ≠ Machine Physical Address
  - VMM dynamically translates Guest Physical Pages into Machine Physical Pages

Memory Virtualization



Memory Virtualization

- VMM maintains Shadow Page Tables
  - Copies of Guest OS translation tables
- VMM catches updates operations of translation tables performed by a Guest OS
  - Write-protect all guest OS page tables
  - Emulates operation in shadow page table
  - Updates effective MMU page table entry, if needed

Memory Virtualization

- PTE entries can be tagged with a context ID
  - Avoids to flush TLB when switching current address space upon scheduling of a new process
  - usually PTE tag = OS process identifier
- Processes of different Guest OSes can be assigned the same Process ID
  - => VMM must flush TLB when switching VMs

Memory Virtualization

- VMM must respect Guest OS virtual page faults
  - Not map virtual pages unmapped by Guest OS
  - When Guest OS unmaps a virtual page:
    - VMM must delete the associated real-page/physical page mapping, if any.
- Conversely, VMM can transparently:
  - Introduce & resolve real-page faults for Guest OSes
  - Share physical pages between Guest OS's
    - Pages with same content's (e.g. zero-ed pages)

Memory Virtualization

- VMM can swap real pages of a VM
  - on « swap » space managed by VMM
- VMM can dynamically distribute physical memory among VM's
  - Needs a specific support in Guest OS (Linux module)
  - VMM asks Guest OS to release memory
    - Guest OS self-allocates [real] pages
    - => no more available for normal [kernel] allocation service
    - VMM assigns same amount of physical pages to other VM's

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Paravirtualization

- OS adaptation to avoid binary translation overhead
- Requires access to OS source code
- Include drivers of virtual devices
- Examples
  - Xen
  - **U**ser **M**ode **L**inux (UML)

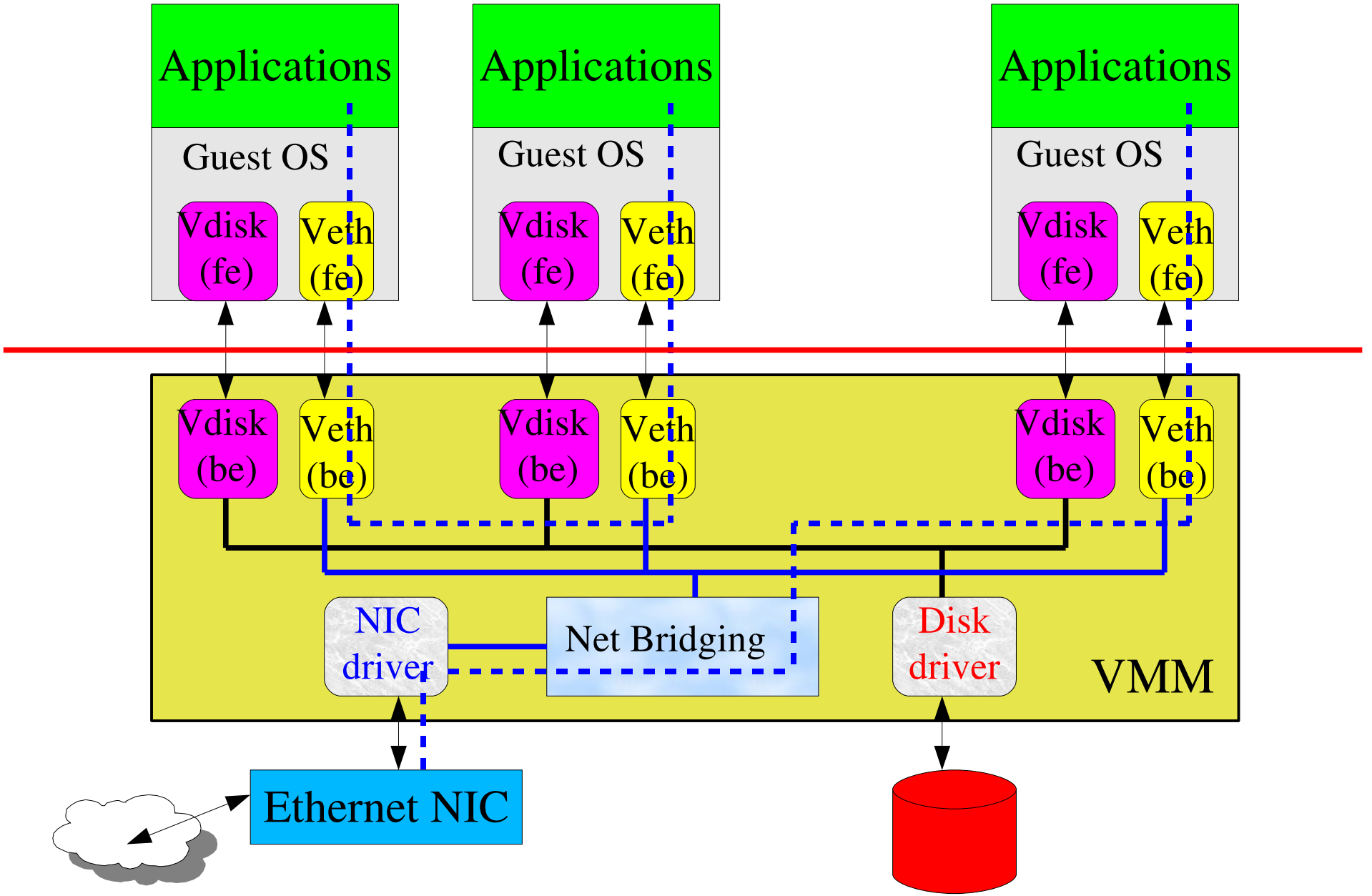
Paravirtualization Principles

- Still run each Guest OS in non-privileged mode
- But with minimal virtualization overhead
- => Modified Guest OS kernel
  - Remove Hardware-Sensitive Instructions
    - Use fast VMM system calls instead, if needed
  - Minimise usage of Privileged Instructions
- Only affect Machine/CPU dependant part of OS
- OS portage on new architecture with same CPU
  - Without system ISA

Paravirtualization (2)

- Guest OS only use Virtual I/O Devices
  - Front-end driver in Guest OS
  - Back-end driver in VMM
- Data transfer through asynchronous I/O rings
- Avoid extra I/O data copies of Full Virtualization
- VMM multiplex VM Virtual Devices on physical devices
  - Virtual Ethernet
  - Virtual Disks

Virtual I/O Devices



Paravirtualization Example: Xen

- Objectives
  - Support more than 100 VM
  - Share resources of Server machines
- Intel IA-32, x86-64 and ARM architectures
- Special first Guest OS called Domain 0
  - Run in privileged mode
  - Have access (and manages) all physical devices
  - Modified version of Linux, FreeBSD

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Hardware-Assisted Virtualization

- Support of Virtualization in Hardware
- Run unmodified OS binaries
- With minimal virtualization overhead
- Simplify VMM development
- Examples
  - KVM (Intel-VT, AMD-V)
  - VMware (Intel-VT)

Hardware Assisted Virtualization

- CPU virtualization
  - AMD-V
  - Intel VT-x (x86), Intel VT-i (Itanium) architectures
  - ARM Cortex-A15
- MMU virtualization
  - Intel **E**xtended **P**age **T**ables (EPT)
  - AMD **N**ested **P**age **T**ables (NPT)

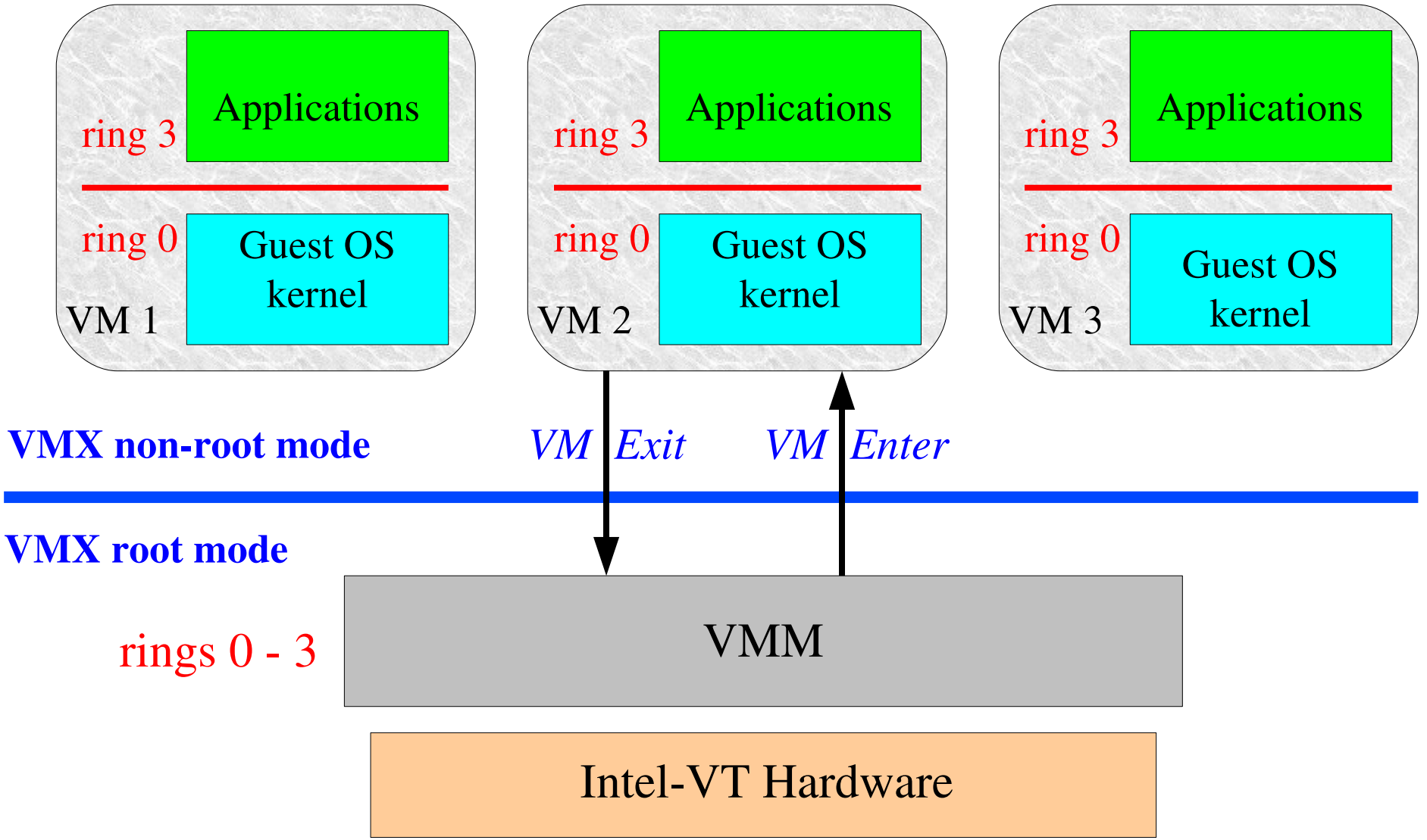
Hardware Assisted Virtualization

- DMA virtualization
  - IO-MMU
- I/O Device virtualization
  - Self-Virtualizing devices
  - **S**ingle **R**oot **I**/**O** **V**irtualization and Sharing Specification (SR-IOV)
  - Extensions to PCIe (PCI Express) Bus standard

Intel VT-x Architecture

- Support unmodified Guest OS with no need for paravirtualization and/or binary code translation
- Simplify VMM tasks & improve VMM performances
- Minimize VMM memory footprint
  - Suppress shadowing of Guest OS page tables
- Enable Guest OS to directly manage I/O devices
  - Without performance lost
  - While enforcing VM isolation and mutual protection

Intel VT-x Architecture Overview



Intel VT-x CPU Virtualization

- **V**irtual **M**achine e**X**tension (VMX)
  - Two new meta-modes of CPU operation
- VMX root mode
  - Behaviour similar to IA-32 without VT
  - Intended for VMM execution
- VMX non-root mode
  - Alternative IA-32 execution environment
  - Controlled by a VMM
  - Designed to run unchanged Guest OS in a VM
- Both modes support rings 0-3 privilege levels
  - Allow VMM to use several privilege levels

Intel VT-x CPU Virtualization

- Two additional CPU mode transitions
- From VMX root-mode to VMX non-root mode
  - Named *VM Enter*
- From VMX non-root mode to VMX root mode
  - Named *VM Exit*
- VM entries & VM exits use a new data structure
  - **V**irtual **M**achine **C**ontrol **S**tructure (VMCS) per VM
  - Referenced with a memory physical address
  - Format and layout hidden
  - New VT-x instructions to access a VMCS

Intel VT-x CPU Virtualization

- Guest State Area
  - Saved value of registers loaded by VM Exits (e.g., Segment Registers, CR3, IDTR)
  - Hidden CPU state (e.g., *CPU Interruptibility State*)
- Host State Area
- VM Control Fields
  - Interrupt Virtualization
  - Exceptions bitmaps
  - I/O bitmaps
  - Model Specific Register R/W bitmaps
  - Execution rights for CPU Privileged Instructions

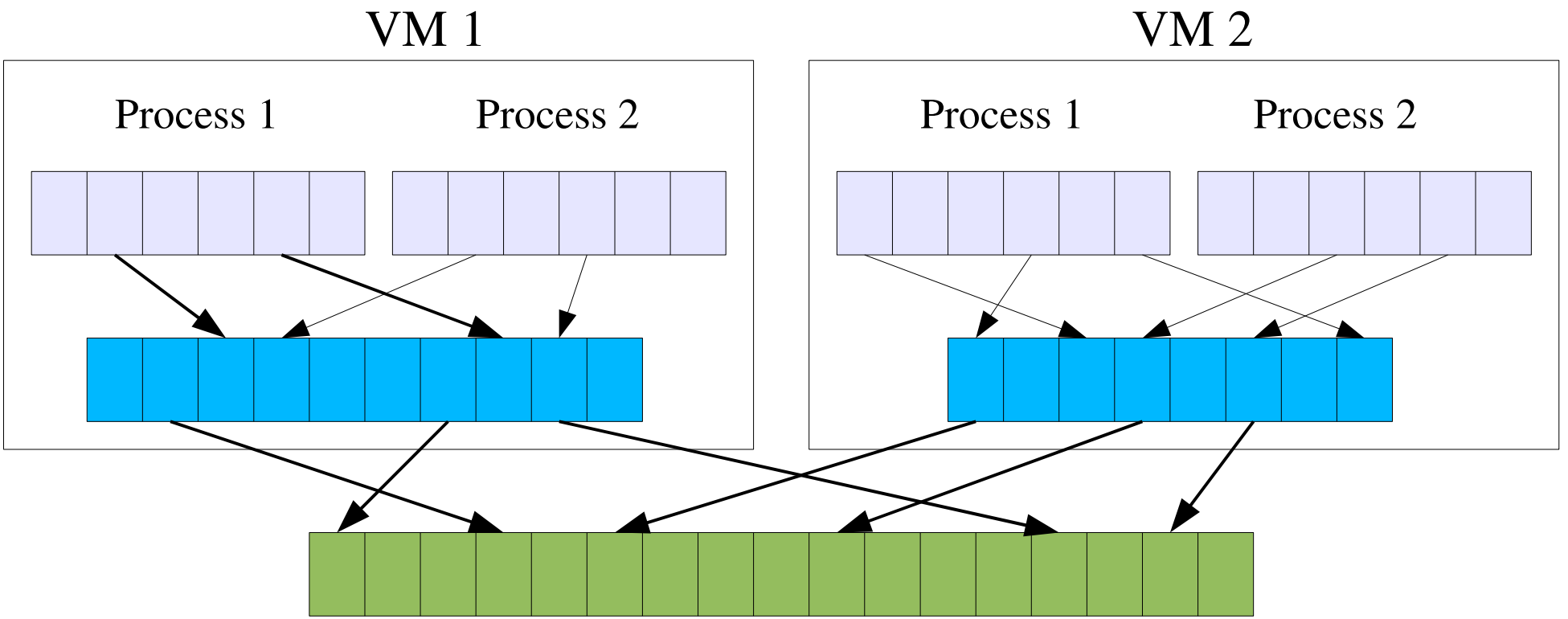
Intel VT-x Interrupt Virtualization

- VMCS External Interrupt Exiting
  - All external interrupts cause *VM Exits*
  - Guest OS cannot mask external interrupts when executing Interrupt Masking instructions
- VMCS Interrupt Window Exiting
  - *VM Exit* occurs whenever Guest OS ready to serve external interrupts
- Used by VMM to control VM interrupts

Intel VT-x MMU Virtualization

- **E**xtended **P**age **T**ables (EPT)
  - Second level of Page Tables in MMU
  - Translate Guest OS Physical Address into Machine Physical Address
  - Controlled by VMM
- **V**irtual **P**rocessor **ID**entifier (VPID)
  - Used to tag TLB entries
  - Avoid to flush TLB upon VM switch

Virtual Memory Virtualization



Guest OS Virtual Memory

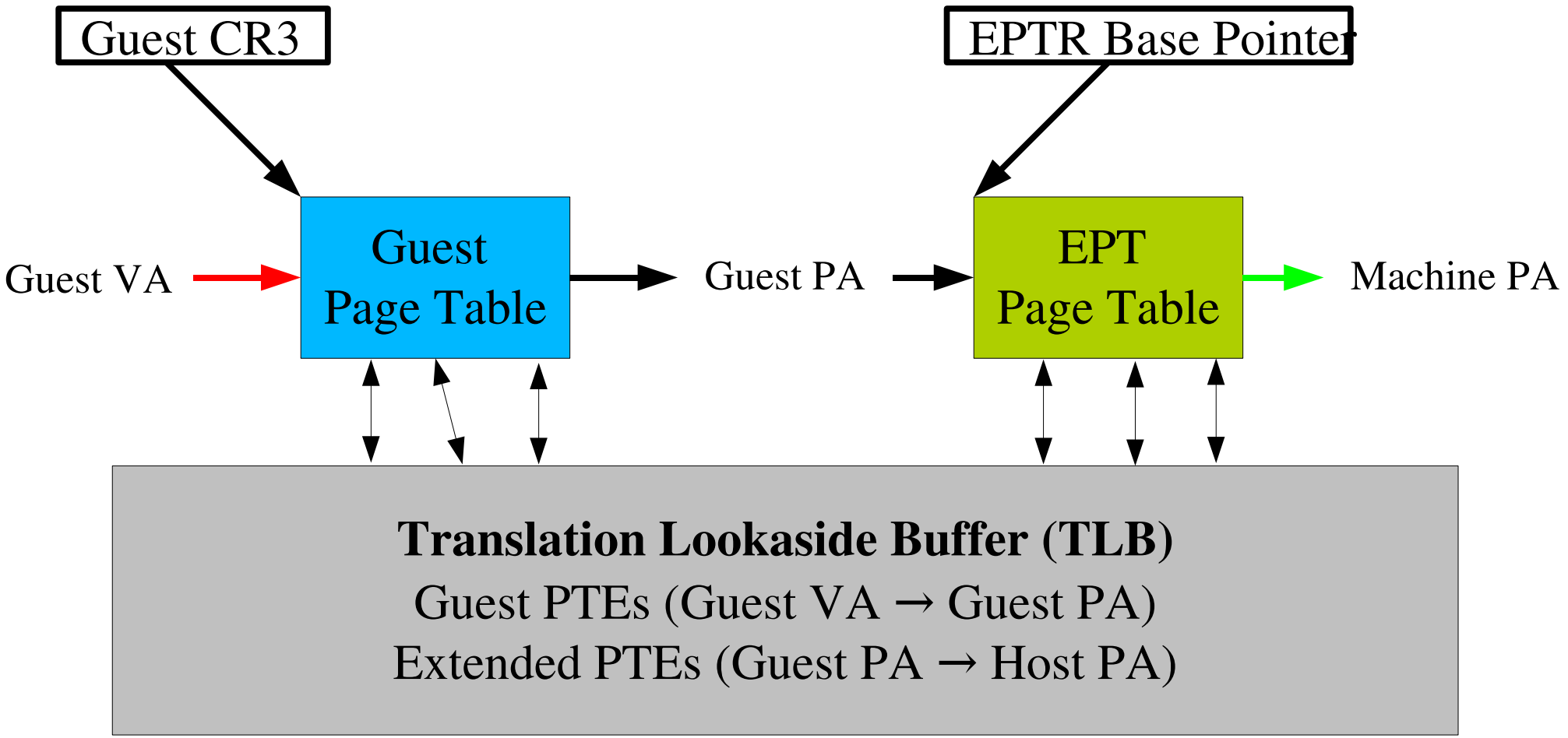
Guest OS Physical Memory

Machine Physical Memory

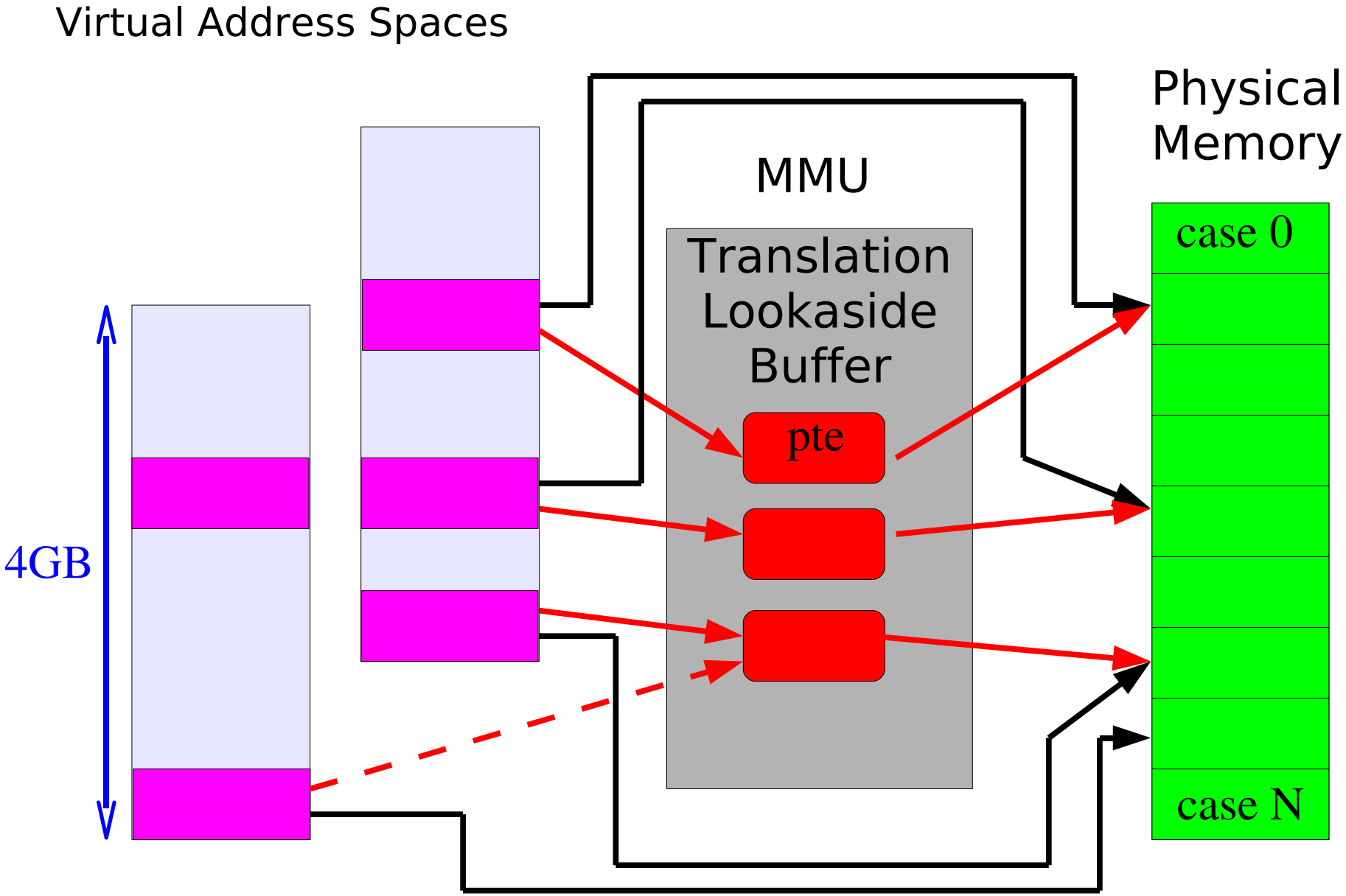
Intel VT-x Extended Page Tables

- VMM controls Extended Page Tables
- EPT used in VMX non-root operation
  - Activated on *VM Enter*
  - Desactivated on *VM exit*
- EPTP register points to Extended Page Tables
  - Instanciated by VMM
  - Saved in VMCS
  - Loaded from VMCS on *VM entry*

Intel VT-x Extended Page Tables



TLB Flush Issue



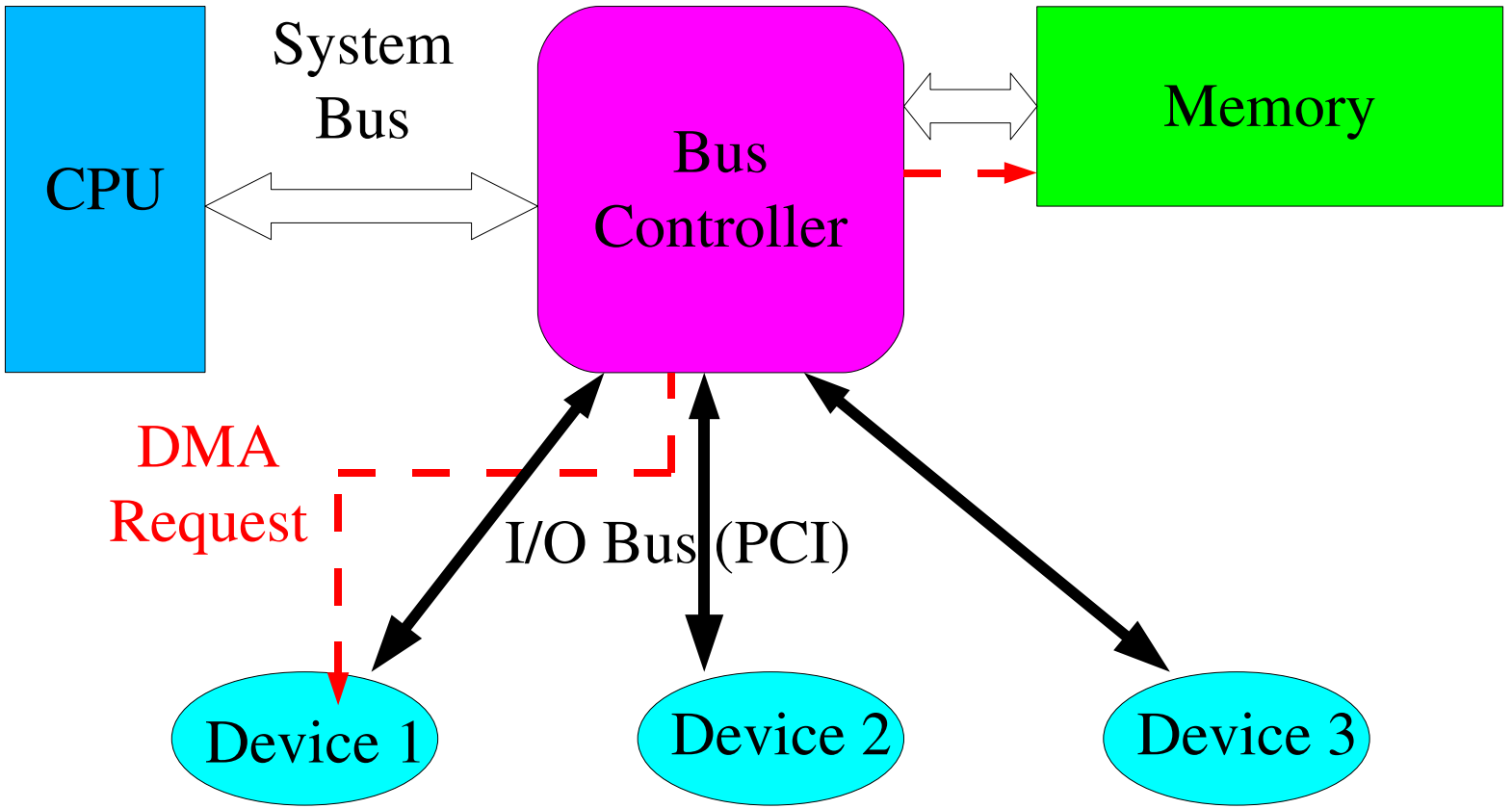
Intel VT-x Virtual Processor Identifier

- 16-bit VPID used to tag TLB entries
  - Enabled by VMM in VMCS
  - Unique VPID is assigned by VMM to each VM
  - VPID 0 reserved for VMM
- Current VPID is 0x0000 when
  - Outside VMX operation
  - In VMX root mode operation
  - In VMX non-root mode if VPID disabled in VMCS
- VPID loaded from VMCS on *VM Enter*

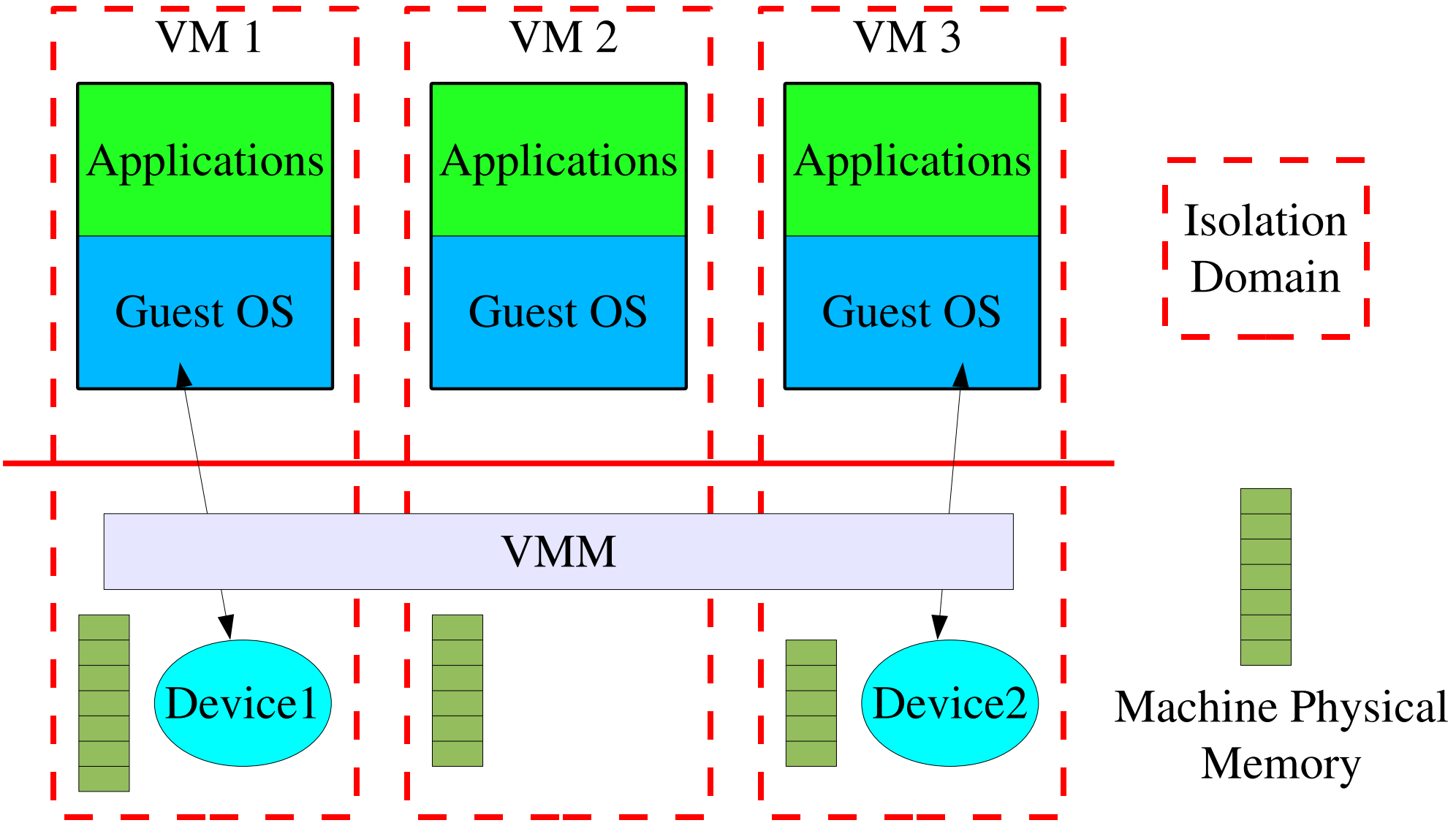
DMA Virtualization

- Enable Guest OS to manage I/O devices
  - I/O devices assigned by VMM to Guest OSes
- Transparent mode
  - Use native device driver of Guest OS
  - Unaware of physical memory Virtualization
- Enforce isolation between Guest Oses
  - Guest OS only view hardware ressources assigned by VMM (memory, devices)

DMA Principles



DMA Virtualization



DMA Virtualization Issue

- Guest OS driver setup I/O registers of device with Guest Physical Address of I/O buffers
- Guest Physical Address must be translated into its corresponding Machine Physical Address when used for DMA operations by device
- GPA Translation cannot be done by VMM
  - VMM cannot catch device-specific driver operations to setup I/O buffers addresses

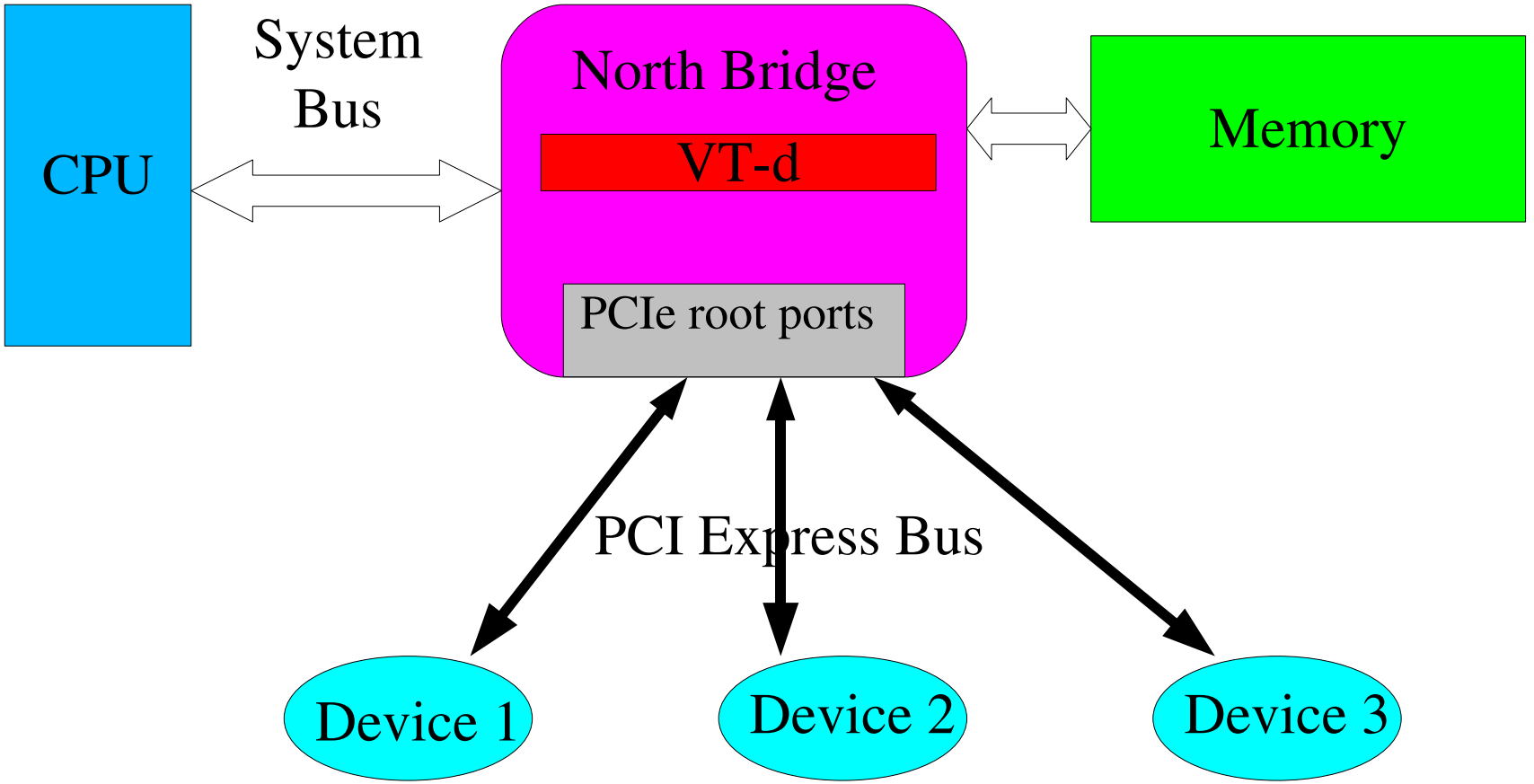
Intel VT-d Protection Domains

- Intel VT-d provides DMA Protection Domains
  - Extension of IOMMU translation mechanism
  - Isolated context of a subset of the Machine Physical Memory (MPA)
  - Correspond to the portion of Machine Physical Memory allocated to a VM
- I/O devices assigned by VMM to a DMA Protection Domain
  - Achieves DMA isolation by restricting memory view of I/O devices through DMA address translation

Intel VT-d DMA Translation

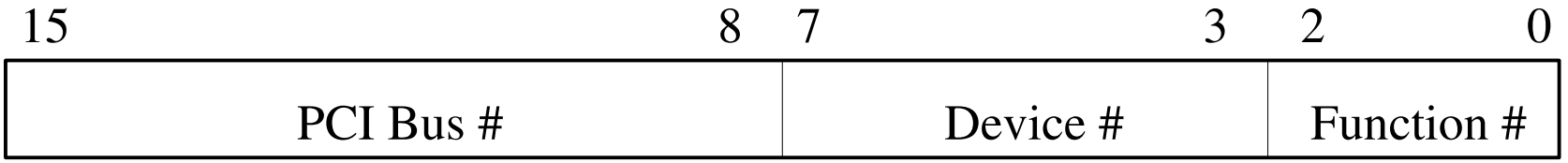
- VT-d hardware treats address specified in DMA request as **D**MA **V**irtual **A**ddress (DVA)
- DVA = GPA of the VM to which the I/O device is assigned
- VT-d translates the DVA into its corresponding Machine Physical Address
- Support of multiple Protection Domains
  - DVA to MPA translation table per Protection Domain
  - Must identify the device issuing a DMA request

VT-d PCI Express North Bridge



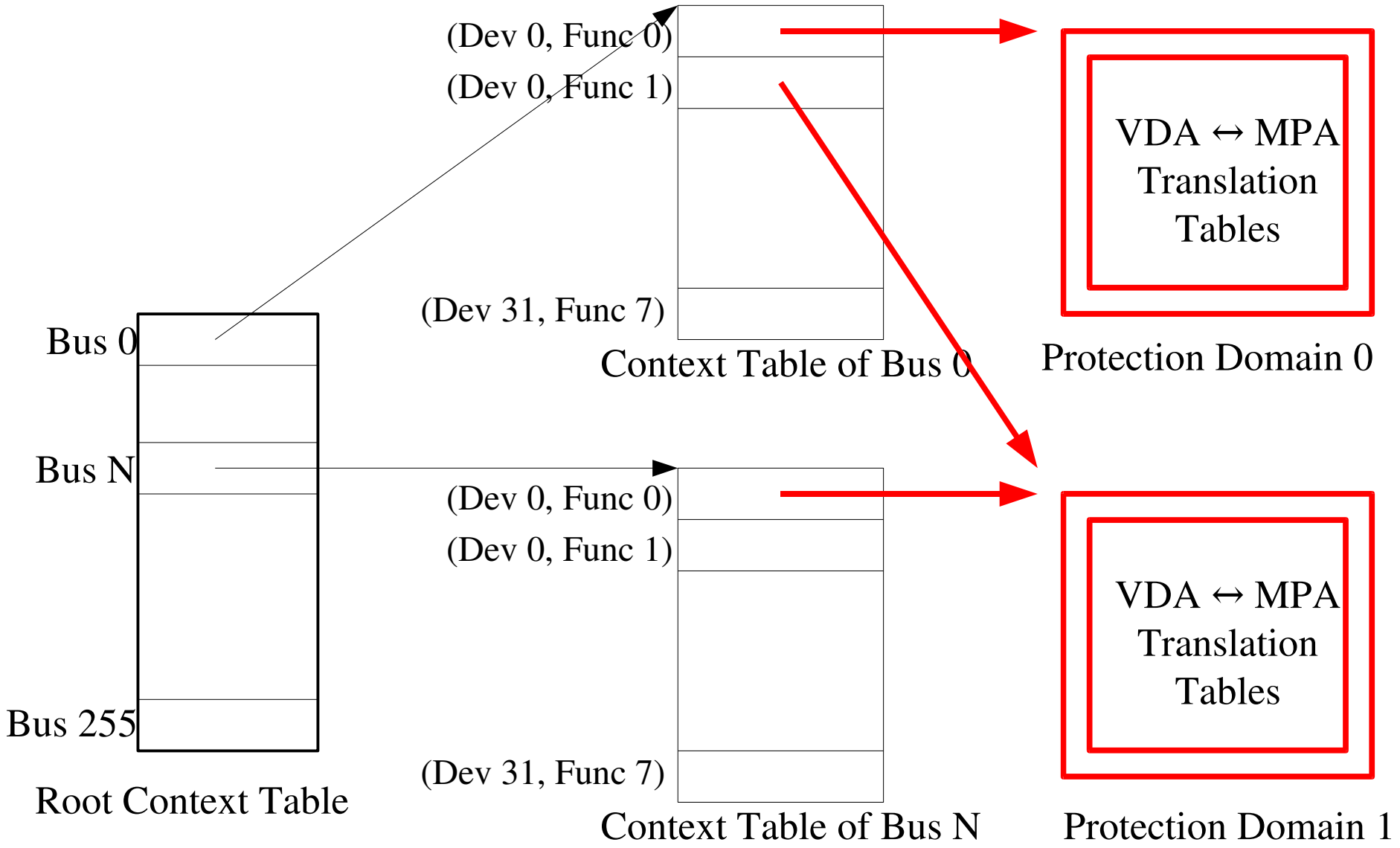
PCI DMA Requester Identification

- Mapping between PCI Device and Protection Domains
- 16-bit PCI DMA Requester Identifier



- Assigned by PCI configuration software
- Bus # indexes Bus Context Table in Root Context Table
- (Device #, Function #) indexes Device Protection Domain in Bus Context Table

Device / Protection Domain Mapping



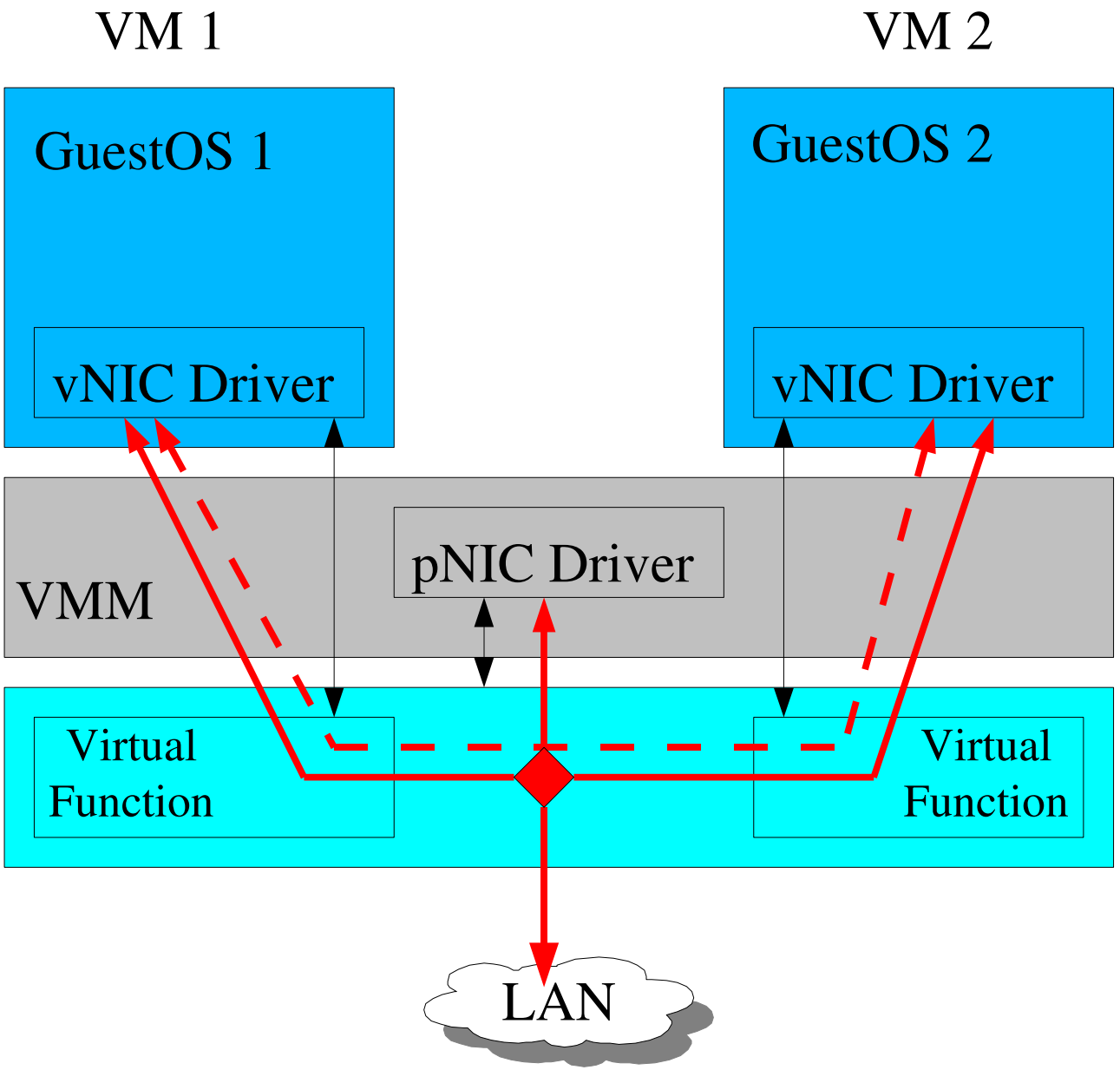
Virtual DMA Address Translation

- VDA ↔ MPA VT-d Page Tables similar to IA-32 processor Page Tables
- 4KB or larger page size granularity
- Read/Write permissions
- Protection Domains managed by VMM
  - Initialized at VM creation time
  - With same translations of the VM Extended Page Table

Device Virtualization

- Share I/O device among multiples VMs
  - With no performance lost
  - While enforcing VM isolation and protection
- Move device virtualization from the VMM to the device itself
- Requires support from the device
- Example of Ethernet controllers

Ethernet Device Virtualization



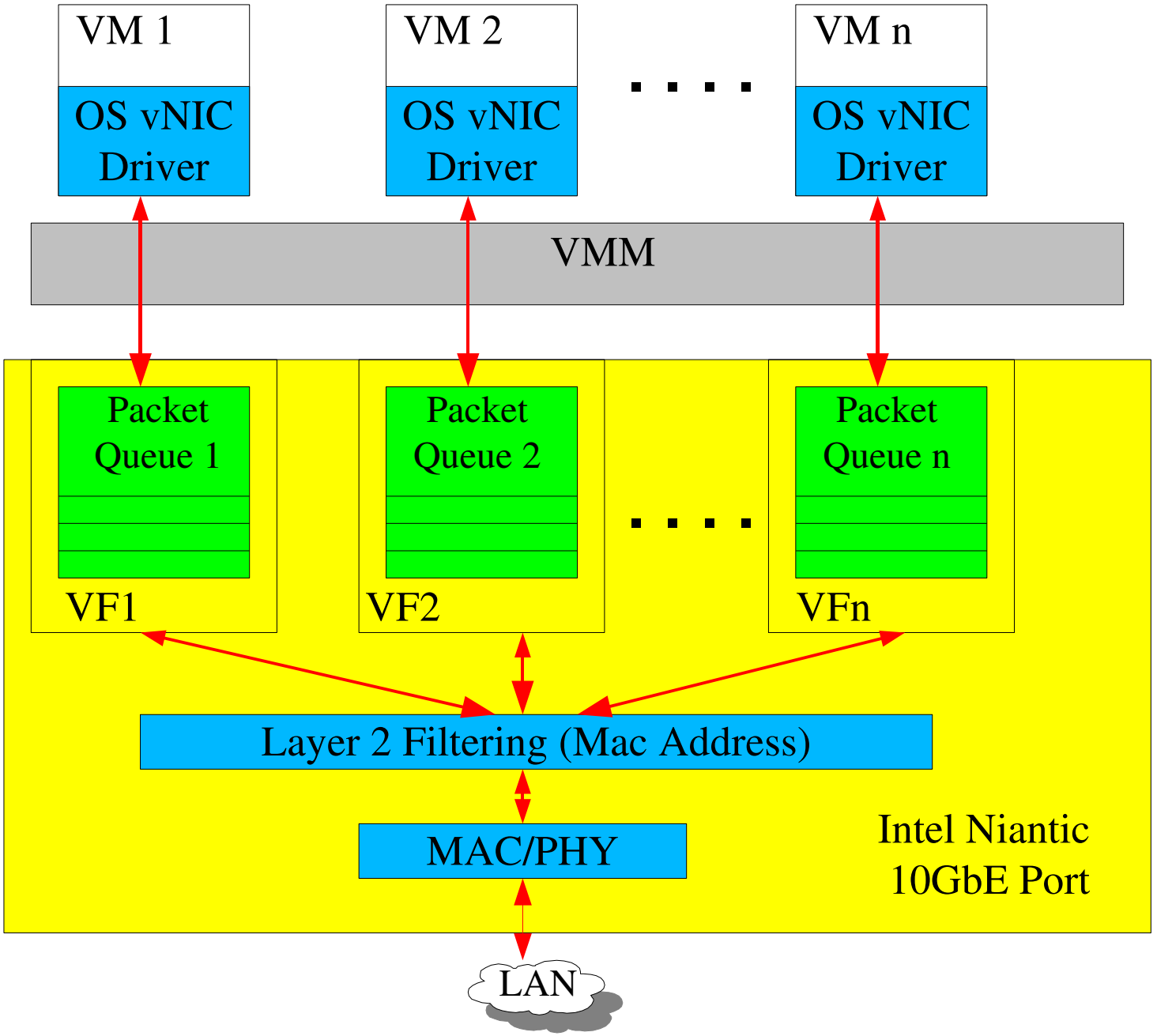
Intel **S**ingle **R**oot **I/O** **V**irtualization

- SR-IOV capable PCI Device can be partitionned into multiple Virtual Functions
- SR-IOV Device appears in PCI configuration space as multiple PCI Virtual Functions
- Each Device Virtual Function includes
  - PCI configuration registers
  - DMA streams
  - Interrupts
- Requires VT-d for DMA virtualization

Intel SR-IOV

- VMM manages physical PCI device
- Create a PCI Virtual Function for each VM
  - Include it into VM PCI configuration space to be probed by VM GuestOS kernel
  - Map it to Protection Domain of VM
- Programs the sharing of physical devices ressources between VFs
- PCI Device Virtual Functions directly managed by specific VF-Aware GuestOS drivers (kind of Para-Virtualization)

Intel SR-IOV


VM 1
OS vNIC Driver
VM 2
OS vNIC Driver
VM n
OS vNIC Driver
VMM
Packet Queue 1
Packet Queue 2
Packet Queue n
VF1
VF2
VFn
Layer 2 Filtering (Mac Address)
MAC/PHY
Intel Niantic 10GbE Port
LAN

Intel SR-IOV - Ethernet example

- Intel Kawela (1GB) / Niantic (10GB) Ethernet NICs
  - Multiple RX/TX packet queues per port
- **V**irtual **D**evice **M**achine **Q**ueues
  - 1 RX paquet queue per VF
- Filters multiple unicast Ethernet Addresses
- Layer-2 paquet filtering based on Ethernet Destination Address
- Duplicate Broadcast / Multicast packets for all VFs
- Load balancing between TX paquets sent by VFs

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Old Embedded Systems

- Relatively simple architecture
- Single-purpose devices
- Dominated by hardware constraints
  - Memory, battery charge
- Dedicated functionalities, with moderated software size and complexity
- Real-time constraints

Old Embedded Systems (2)

- Closed environment (« black boxes »)
- Fixed hardware configuration
- Full software provided by device vendor
- No dynamic loading of applications
- Software updates rareful

Embedded Systems Now (1)

- Take on features of general-purpose OS's
- Growing functionalities
  => growing complexity and size
- Run applications originally developed for PC's
  - Sophisticated **H**uman **M**achine **I**nterfaces (HMI)
  - Safari Web browser on iPhones
- Dynamic loading of applications
  - Iphone
  - Google Android

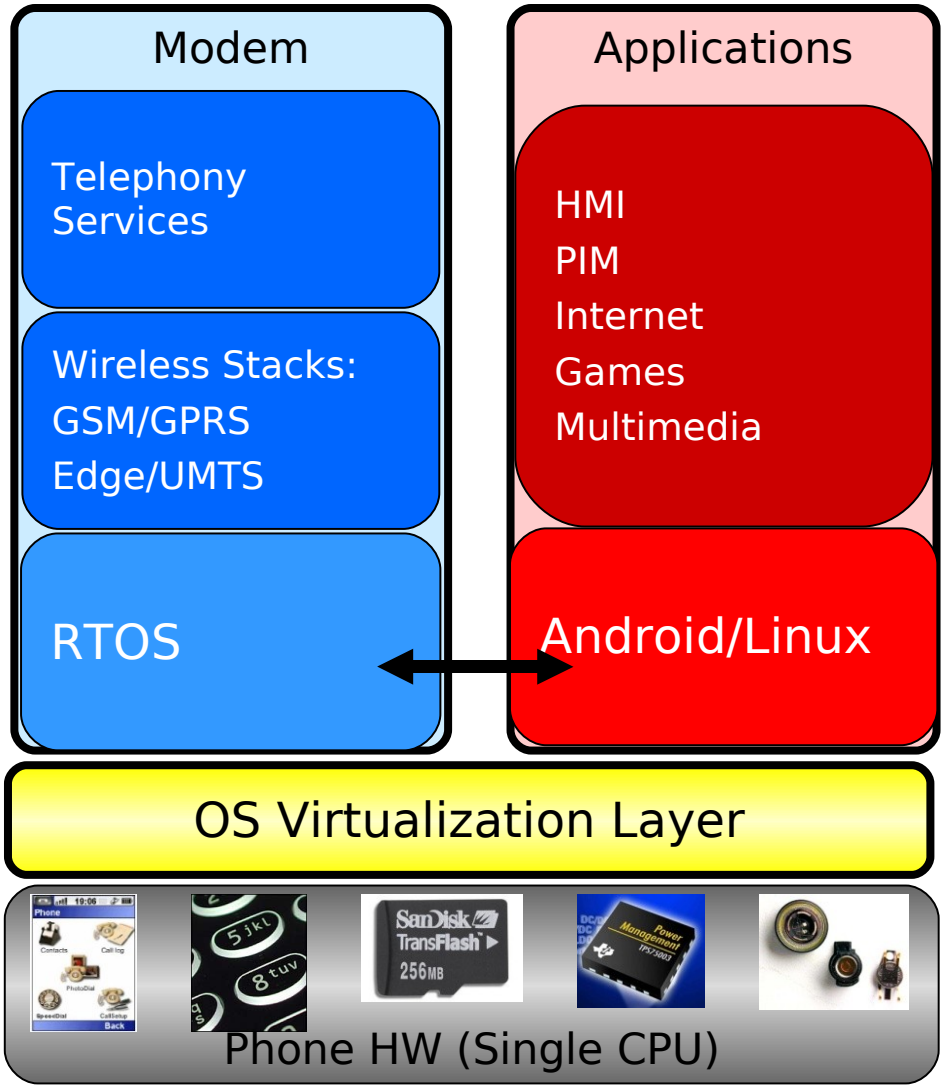
Embedded Systems Now (2)

- Dynamically load device's owner specific applications
  - Games
- Applications developped by engineers with no expertise in embedded systems
  - Java applications
- Need for exchanges with external world
  - USB, Bluetooth, Wi-Fi
  - TCP/IP
- Need for open API's, and openness in general
- Need for high-level systems (Linux, Windows)

Embedded Systems Challenges

- Still Real-Time systems (part of it)
  - Baseband stack of mobile phones
- Still hardware constraints
  - Battery
  - Memory (to minimize device's cost)
- Also used in mission/life critical situations
  - Weapons
  - Cars
- High requirements on reliability and security

Mobile Handsets



- Run Android/Linux applications on baseband processor
- Re-use existing legacy modem software stack with its RTOS (no changes)
- Support of Linux at a minimal development cost
- Operating System independence for future evolutions
- Security & Protection through OS isolation

HMI: **H**uman-**M**achine-**I**nterface
PIM: **P**ersonal **I**nformation **M**ngt.

Virtualization in Embedded Systems

- Support for heterogeneous OS's environments
- Real-time OS
  - Legacy software
  - Dedicated applications whose real-time constraints cannot be achieved by General-Purpose systems
  - Licence issues (« GPL contamination »)
- General Purpose OS
  - Openness
  - HMI

Virtualization in Embedded Systems

- Concurrent execution of RTOS and GP-OS on the same CPU
- Reduces cost (**B**ill **O**f **M**aterial)
- Requires the underlying VMM to provide
  - Memory isolation between OS's
  - CPU scheduling among OS's, with higher priority to the RTOS
  - Device partitionning
  - Communication mechanism between OS's

Virtualization in Embedded Systems

- Leverage multi-cores support with virtual machine abstraction
- 1 core per OS => no need for CPU scheduling
- 2 low-performance cores consume less power than a single high performance CPU
  => simplify power management
- New model of software distribution, shipping application with its own OS
  - No OS configuration/version incoherency

Security Through Virtualization

- Notion of Trusted Computing Base (TCB)
  - Part of the system that provides security foundations
  - Should only include hardware and VMM
  - May also include RTOS, for performance/legacy reasons
- Run GP OS in an isolated Virtual Machine
  - Avoid damaged GP OS to compromise the secure parts (data, services) of the system

Embedded + Virtualization Challenges (1)

- Full isolation of VM's does not fit cooperation requirements between OS's
- Efficient communication mechanisms between VM's
- Global scheduling, with interleaved priorities
- Global Energy Management

Embedded + Virtualization Challenges (2)

- Efficient communication mechanisms between VM's
  - Virtual Ethernet device not adapted
  - Need VMM-controlled shared memory transfers
- Example: Video streaming on a Smartphone
  - Video data received via the baseband managed by RTOS
  - Video data displayed by a Media Player running on GPOS
  - Avoid copy of video data transfered between the 2 OS's !

Task Scheduling Issues

- Standard server-oriented Virtualization model
  - The VMM schedules VM's on the CPU
  - The OS on each VM runs its own scheduler
- Interleaved priorities in Embedded Systems
  - Baseband task of RTOS with a high priority
  - But GPOS Media-Player must have a higher priority than some low-priority tasks of RTOS
  - Enable a VM to yield the CPU
    - Use a RT task as a proxy of GP OS application, and make it yield the CPU

Multi-Users Devices

- Mobile phone has 3 types of users, each with specific private data to protect from the others
  - The person owning the device, with address book, emails, documents, etc.
  - Different wireless providers, for example private and professionnal: network access properly authenticated, ensure correct billing !
  - Third-party service providers, for instance multimedia providers.
- Owner and third-parties must be granted secure financial transactions

Virtualization in Hardware

- Only way to build a real TCB
  - Without penalizing performances
- Should include support for
  - Memory Partitionning
  - Physical Memory / Machine Memory mapping
  - Coupled with multi-cores
  - Device Partitioning
    - Interrupt routing
    - I/O DMA coupled with memory partitioning & Physical Memory / Machine Memory mapping

Plan

- History
- Virtualization Usages
- Virtualization Taxonomy
- Process Level Virtualization
- Transparent Hardware Emulation
- Transparent Hardware Virtualization
- Paravirtualization
- Hardware-Assisted Virtualization
- Virtualization and Embedded Systems
- Evolution of Virtualization

Evolutions of Virtualization

- VMM shipped with hardware
  - By CPU/Motherboard constructor
  - With extensions of Machine constructor
  - Plays BIOS role
- VMM still a boot option ?
- VMM available as free software ?
- VMM available as Open Source ?

Evolutions of Virtualization

- Public & Open VMM API
  - Ease port of OS on top of VMM
- Device Virtualization
  - Included in VMM API
  - Allow generic device drivers in OS's
  - Increase OS portability