Cloud Computing

- Server Virtualization

Agenda

- Virtualization Technique
  - CPU Virtualization
    - Emulation techniques
    - Trap and emulate model
    - Hardware assistance
  - Memory Virtualization
    - Shadow page table
    - Hardware assistance
  - IO Virtualization
    - Overview
    - Device model
    - Hardware assistance
- Ecosystem
  - VMware
  - Xen
  - KVM
- Other Issues
  - Live migration
  - Cloud properties

Emulation techniques

Trap and emulate paradigm

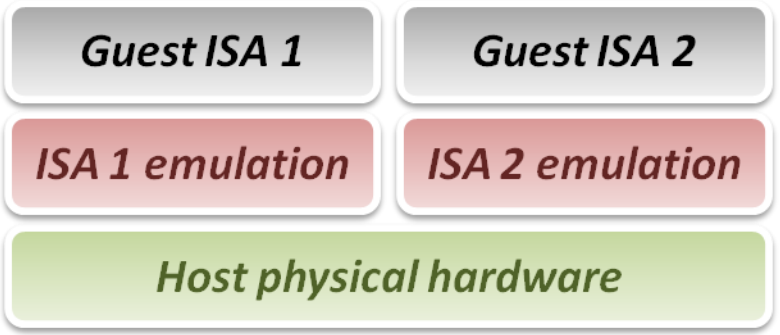
Hardware assistance

CPU VIRTUALIZATION

Emulation Technique

- Why do we talk about emulation ?
  - In fact, virtualization technique can be treated as a special case of emulation technique.
  - Many virtualization techniques were developed in or inherited from emulation technique.

- Goal of emulation :
  - Provide a method for enabling a (sub)system to present the same interface and characteristics as another.



Emulation Technique

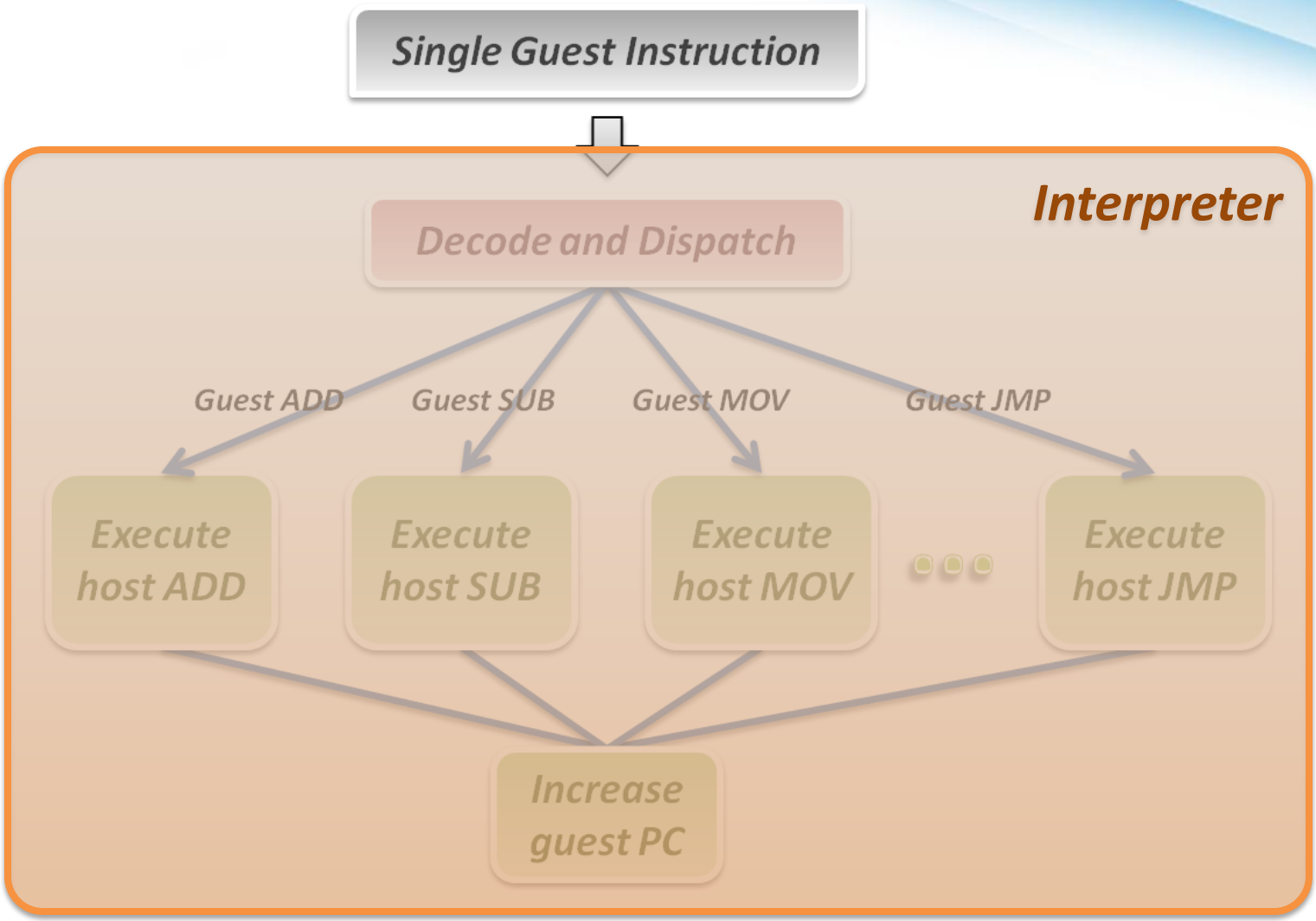
- Three emulation implementations :
  - Interpretation
    - Emulator interprets only one instruction at a time.
  - Static Binary Translation
    - Emulator translates a block of guest binary at a time and further optimizes for repeated instruction executions.
  - Dynamic Binary Translation
    - This is a hybrid approach of emulator, which mix two approaches above.
- Design challenges and issues :
  - Register mapping problem
  - Performance improvement

Interpretation

- Interpreter execution flow :
  1. Fetch one guest instruction from guest memory image.
  2. Decode and dispatch to corresponding emulation unit.
  3. Execute the functionality of that instruction and modify some related system states, such as simulated register values.
  4. Increase the guest PC (Program Counter register) and then repeat this process again.

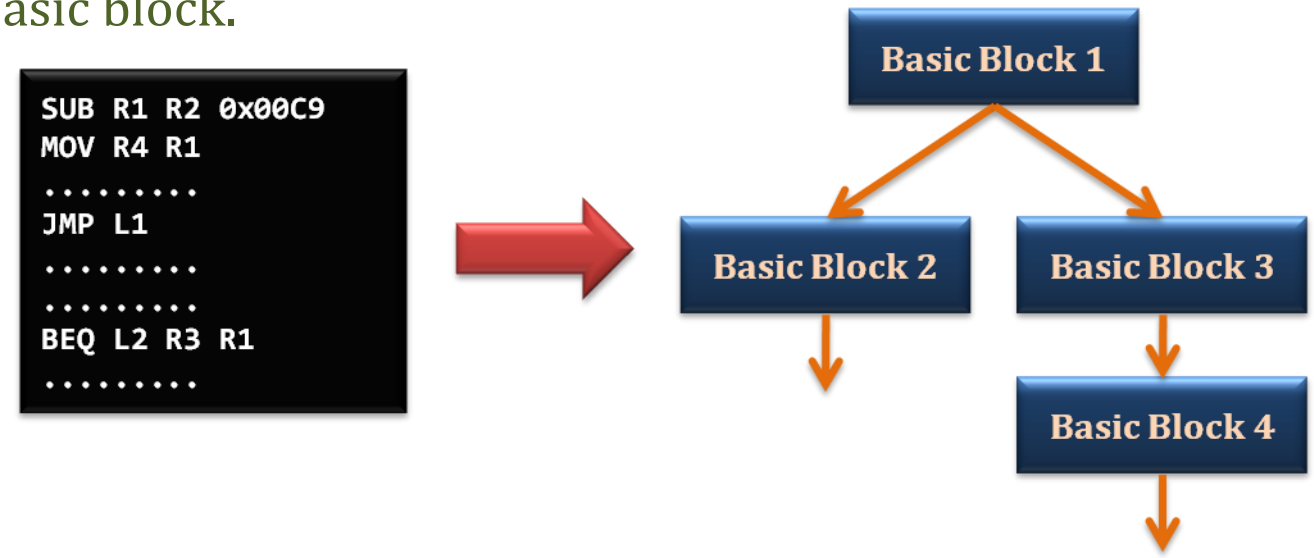
- Pros & Cons
  - Pros
    - Easy to implement
  - Cons
    - Poor performance

Interpretation



Static Binary Translation

- Using the concept of basic block which comes from compiler optimization technique.
  - A basic block is a portion of the code within a program with certain desirable properties that make it highly amenable to analysis.
  - A basic block has only one entry point, meaning no code within it is the destination of a jump instruction anywhere in the program.
  - A basic block has only one exit point, meaning only the last instruction can cause the program to begin executing code in a different basic block.

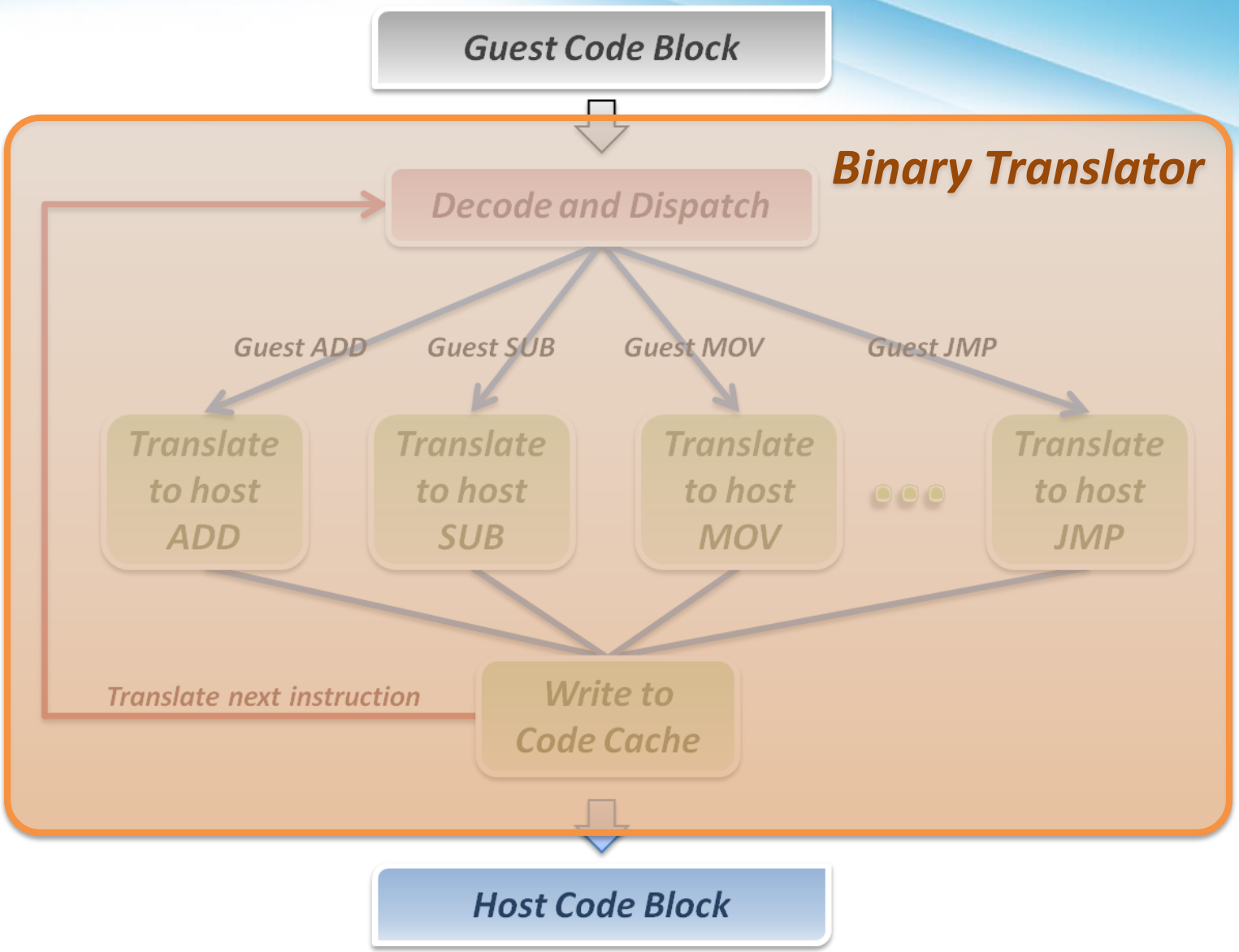


Static Binary Translation

- Static binary translation flow :
  1. Fetch one block of guest instructions from guest memory image.
  2. Decode and dispatch each instruction to the corresponding translation unit.
  3. Translate guest instruction to host instructions.
  4. Write the translated host instructions to code cache.
  5. Execute the translated host instruction block in code cache.

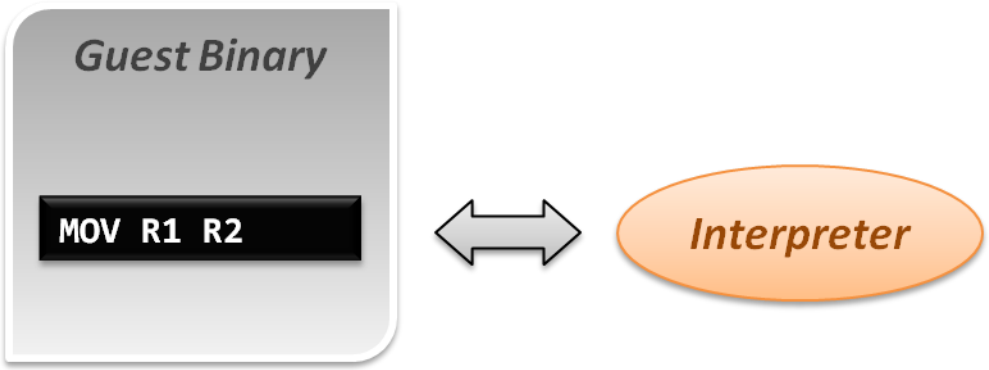
- Pros & Cons
  - Pros
    - Emulator can reuse the translated host code.
    - Emulator can apply more optimization when translating guest blocks.
  - Cons
    - Implementation complexity will increase.

Binary Translation

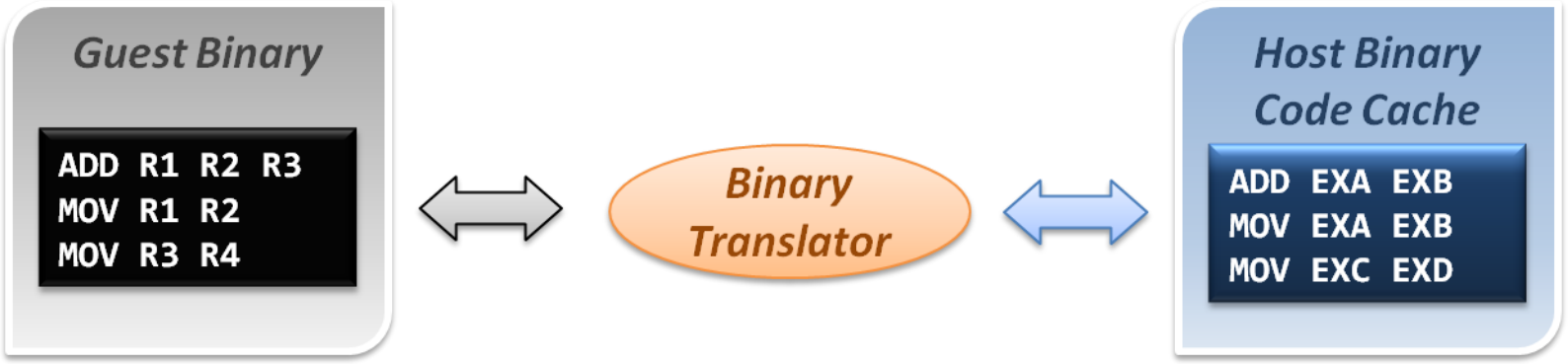


Comparison

- Interpretation implementation



- Static binary translation implementation

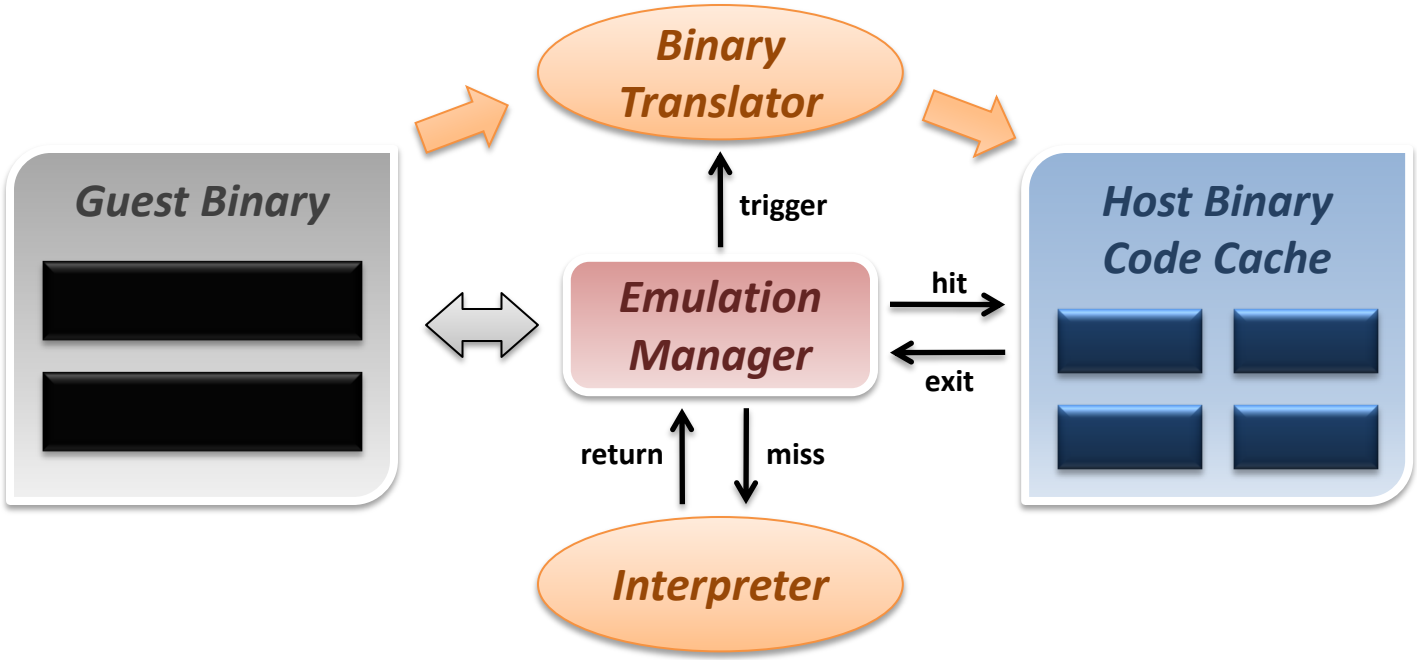


Dynamic Binary Translation

- A hybrid implementation
  - For the first discovered codes, directly interpret by interpreter and record these codes as discovered.
  - As the guest codes discovered, trigger the binary translation module to translate the guest code blocks to host code blocks, and place them into code cache.
  - When execute the translated block of guest code next time, jump to the code cache and execute the translated host binary code.
- Pros & Cons
  - Pros
    - Transparently implement binary translation.
  - Cons
    - Hard to implement.

Dynamic Binary Translation

1. First time execution, no translated code in code cache.
2. Miss code cache matching, then directly interpret the guest instruction.
3. As a code block discovered, trigger the binary translation module.
4. Translate guest code block to host binary, and place it in the code cache.
5. Next time execution, run the translated code clock in the code cache.



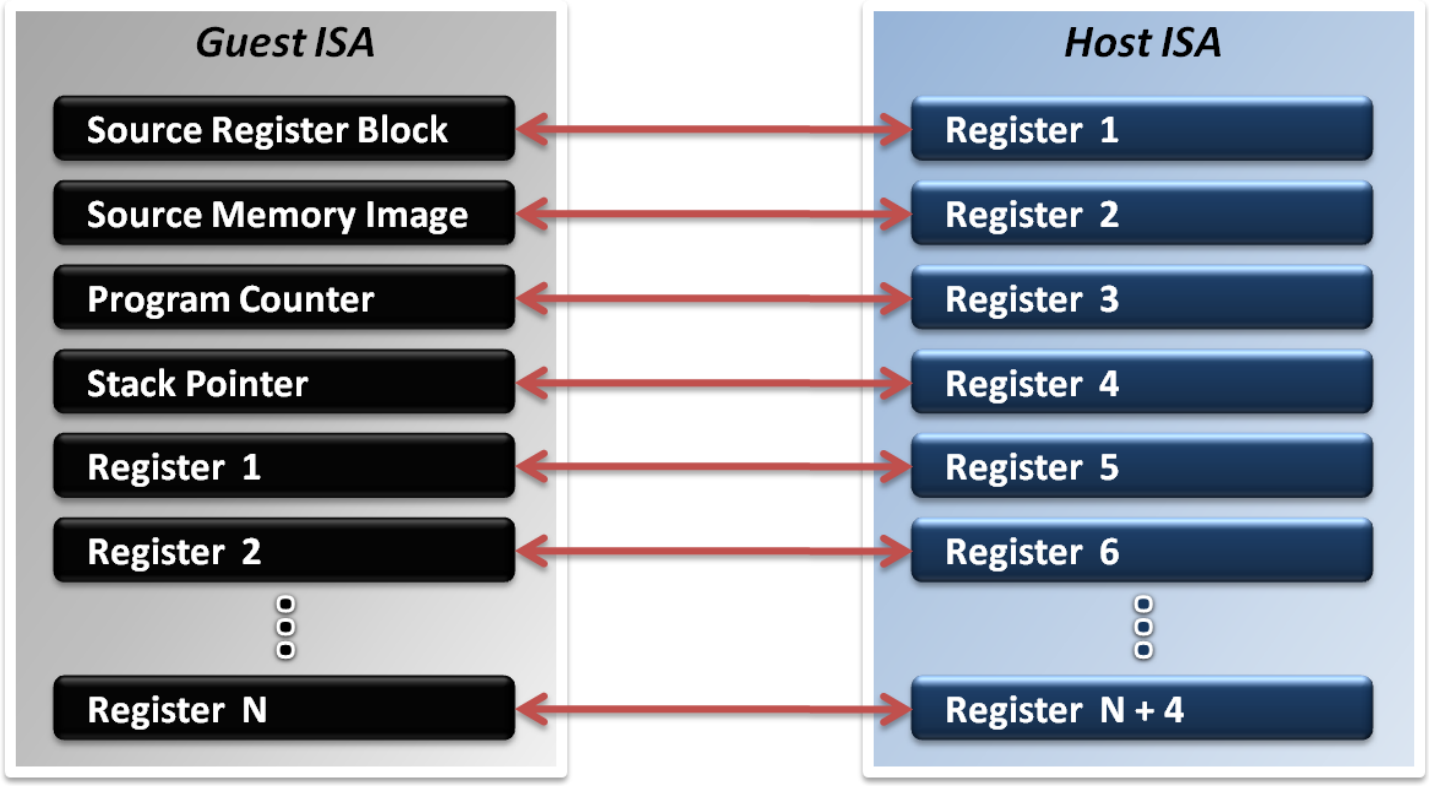
Register mapping problem

Performance improvement

Design challenges and issues

Register Mapping Problem

- Why should we map registers ?
  - Different ISA will define different number of registers.
  - Sometimes guest ISA even requires some special purpose register which host ISA does not define.



Register Mapping Problem

- Mapping different purpose of registers :
  - Map special purpose registers
    - Program Counter Register
    - Stack Pointer Register
    - Page Table Register
    - System Statues Register
    - Special Flags Register
  - Hold guest context and memory image
  - Map general purpose registers
  - Map intermediate values

Register Mapping Problem

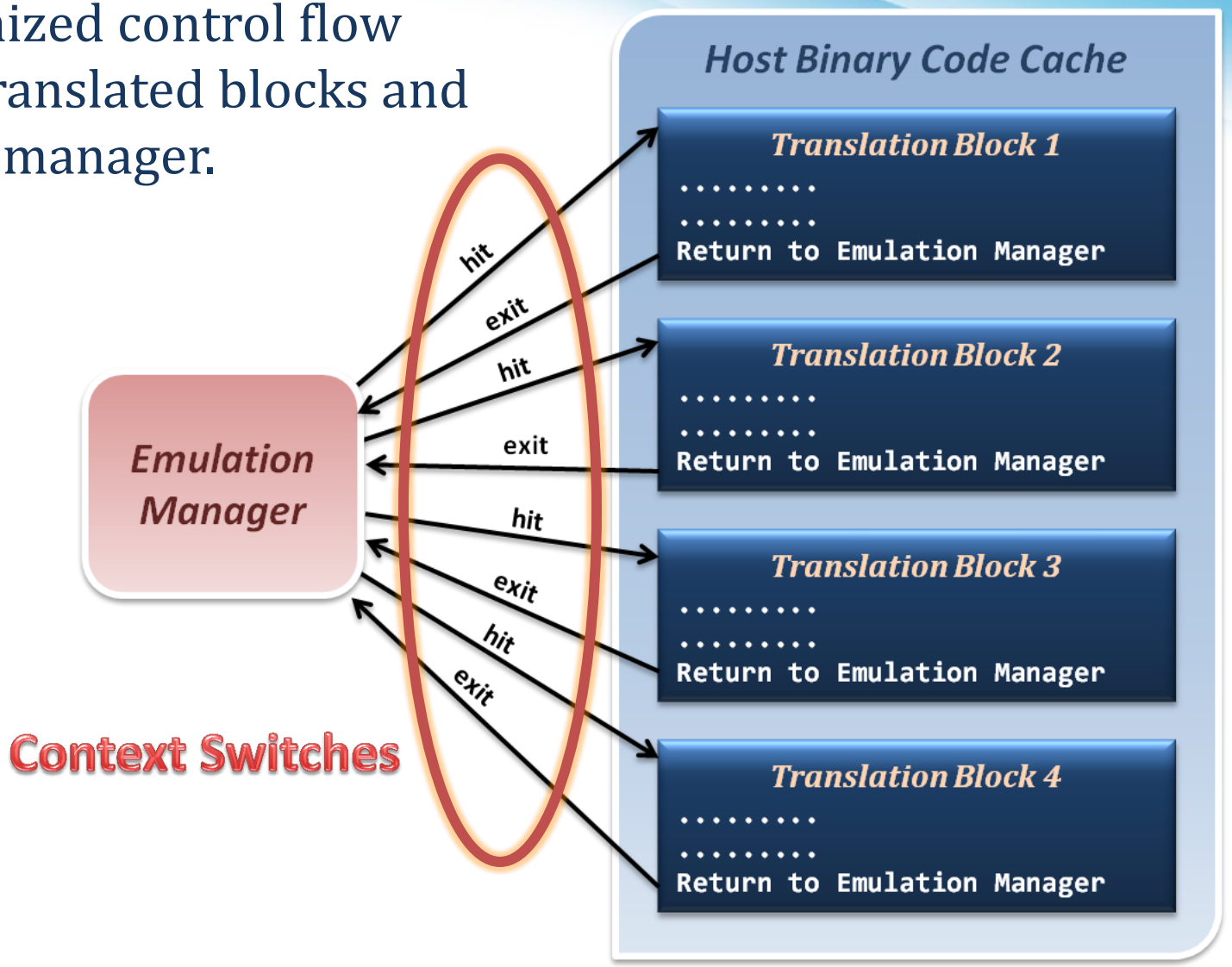
- If the number of host registers is larger the guest
  - That will be an easier case for implementation.
  - Directly map one register of guest to one of host, and make use of the rest registers for optimization.
  - Example :
    - Translating x86 binary to RISC
- If the number of host registers is not enough
  - That should involve more effort.
  - Emulator may only map some frequently used guest registers to host, and left the unmapped registers in memory.
  - Mapping decision will be critical in this case.

Performance Improvement

- What introduces the performance hit ?
  - Control flow problem
    - Highly frequent context switches between code caches and emulation manager will degrade performance.
  - Target code optimization
    - Translate guest code block in instruction-wise (translate one instruction at a time) will miss many optimization opportunities.
- Solutions :
  - Translation Chaining
  - Dynamic Optimization

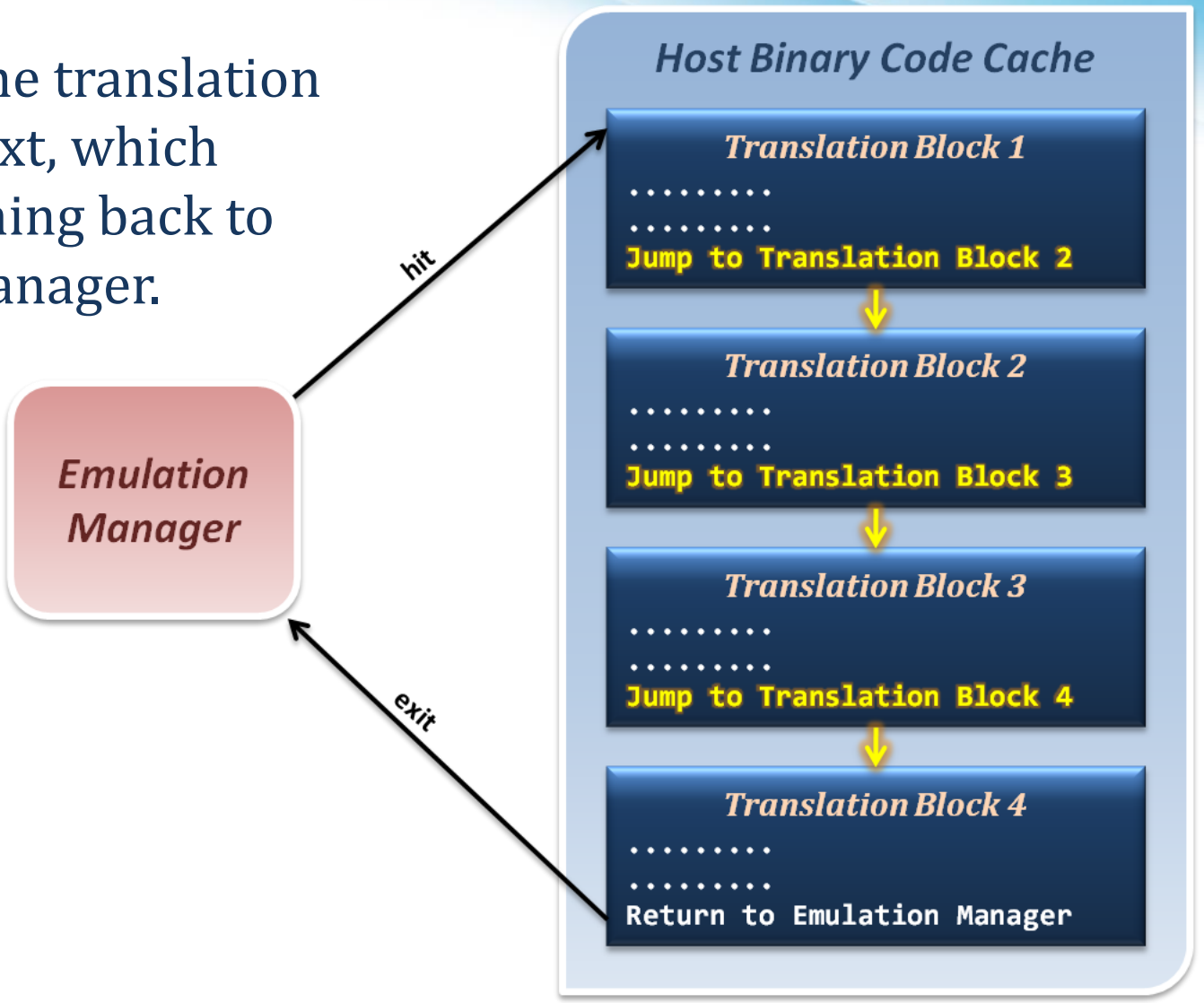
Translation Chaining

- Non-optimized control flow between translated blocks and emulation manager.



Translation Chaining

- Jump from one translation directly to next, which avoids switching back to emulation manager.



Dynamic Optimization

- How to optimize binary codes ?
  - Static optimization (compiling time optimization)
    - Optimization techniques apply to generate binary code base on the semantic information in source code.
  - Dynamic optimization (run time optimization)
    - Optimization techniques apply to generated binary code base on the run time information which relate to program input data.
- Why we use dynamic optimization technique ?
  - Advantages :
    - It can benefit from dynamic profiling.
    - It is not constrained by a compilation unit.
    - It knows the exact execution environment.

Dynamic Optimization

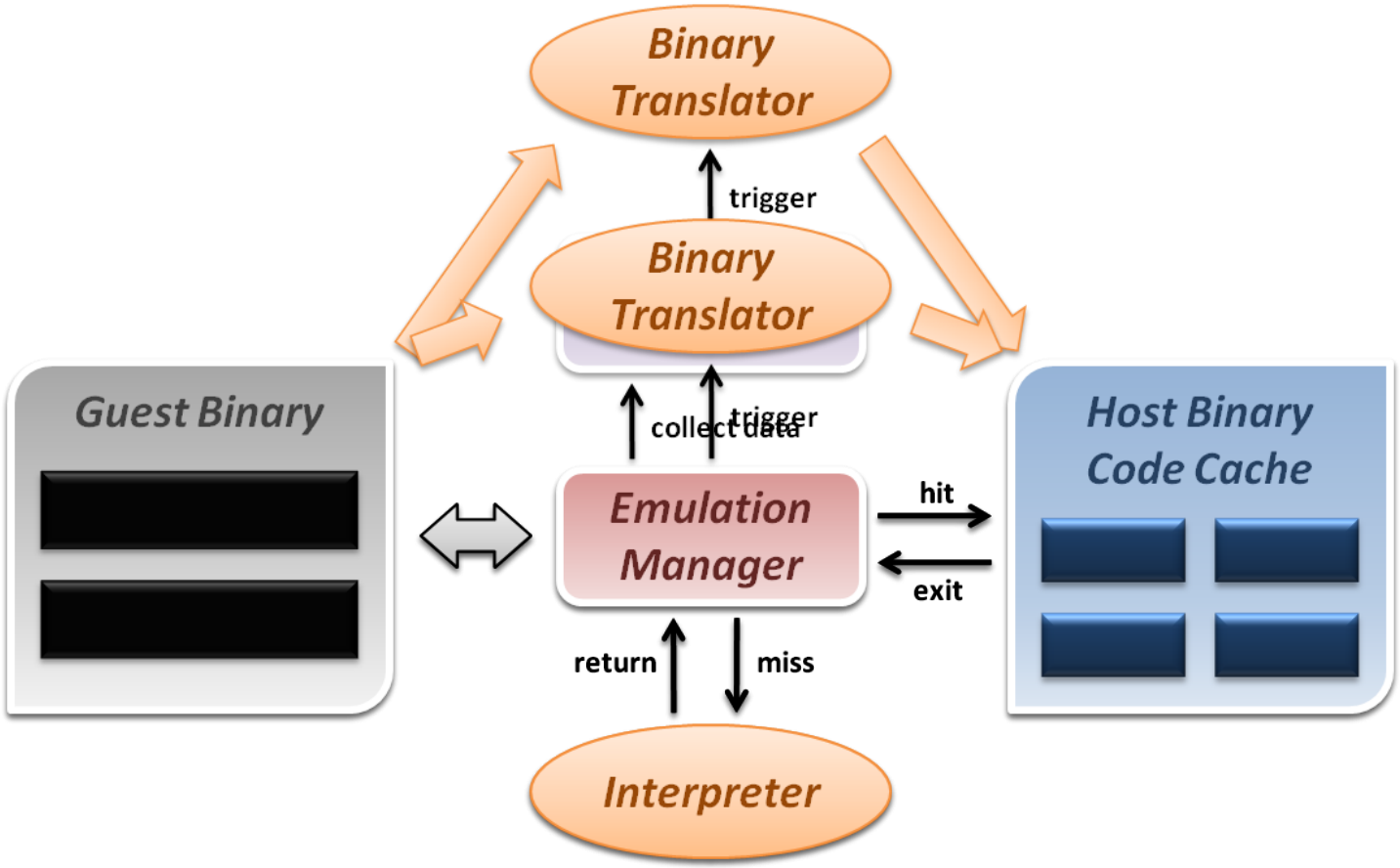
- How to implement dynamic optimization ?
  - Analyze program behavior in run time.
  - Collect run time profiling information based on the input data and host hardware characteristics.
  - Dynamically translate or modify the binary code by reordering instructions or other techniques.
  - Write back the optimized binary into code cache for next execution.

Dynamic Optimization

- How to analyze program behavior and profile ?
  - Collect statistics about a program as it runs
    - Branches (taken, not taken)
    - Jump targets
    - Data values
    - Cache misses
  - Predictability allows these statistics to be used for optimizations to be used in the future

Dynamic Optimization

- Dynamic binary translation and optimization :



Emulation techniques

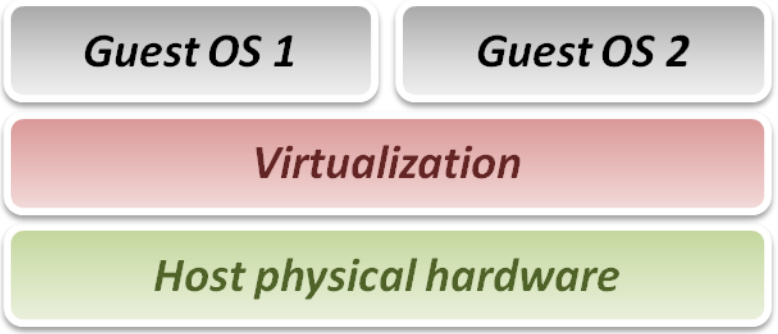
Trap and emulate model

Hardware assistance

CPU VIRTUALIZATION

Virtualization Technique

- From emulation to virtualization :
  - While emulation techniques emulate guest on host, whose ISA differ from guest, in virtualization techniques, guest and host have the same ISA.
  - Some problems in emulation will not exist in virtualization :
    - No need to translate each binary instruction to host ISA.
    - No need to worry about unmatched special register mapping.
  - Some new problems didn't exist in emulation exist now :
    - Instruction privileges should be well-controlled.
- Goal of virtualization :
  - Run or simulate all instructions of guest OS.



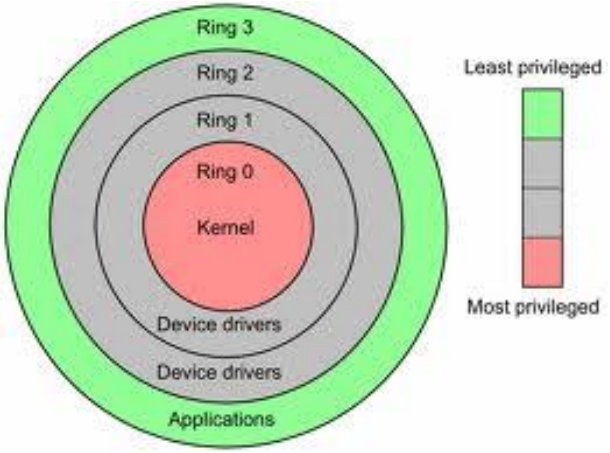
Virtualization Technique

- Virtualization requirements from Popek and Goldberg :
  - Popek and Goldberg provide a set of sufficient conditions for a computer architecture to efficiently support system virtualization.
  - Popek and Goldberg provide guidelines for the design of virtualized computer architectures.

- In Popek and Goldberg terminology, a VMM must present all three properties :
  - Equivalence (Same as real machine)
  - Resource control (Totally control)
  - Efficiency (Native execution)

CPU Architecture

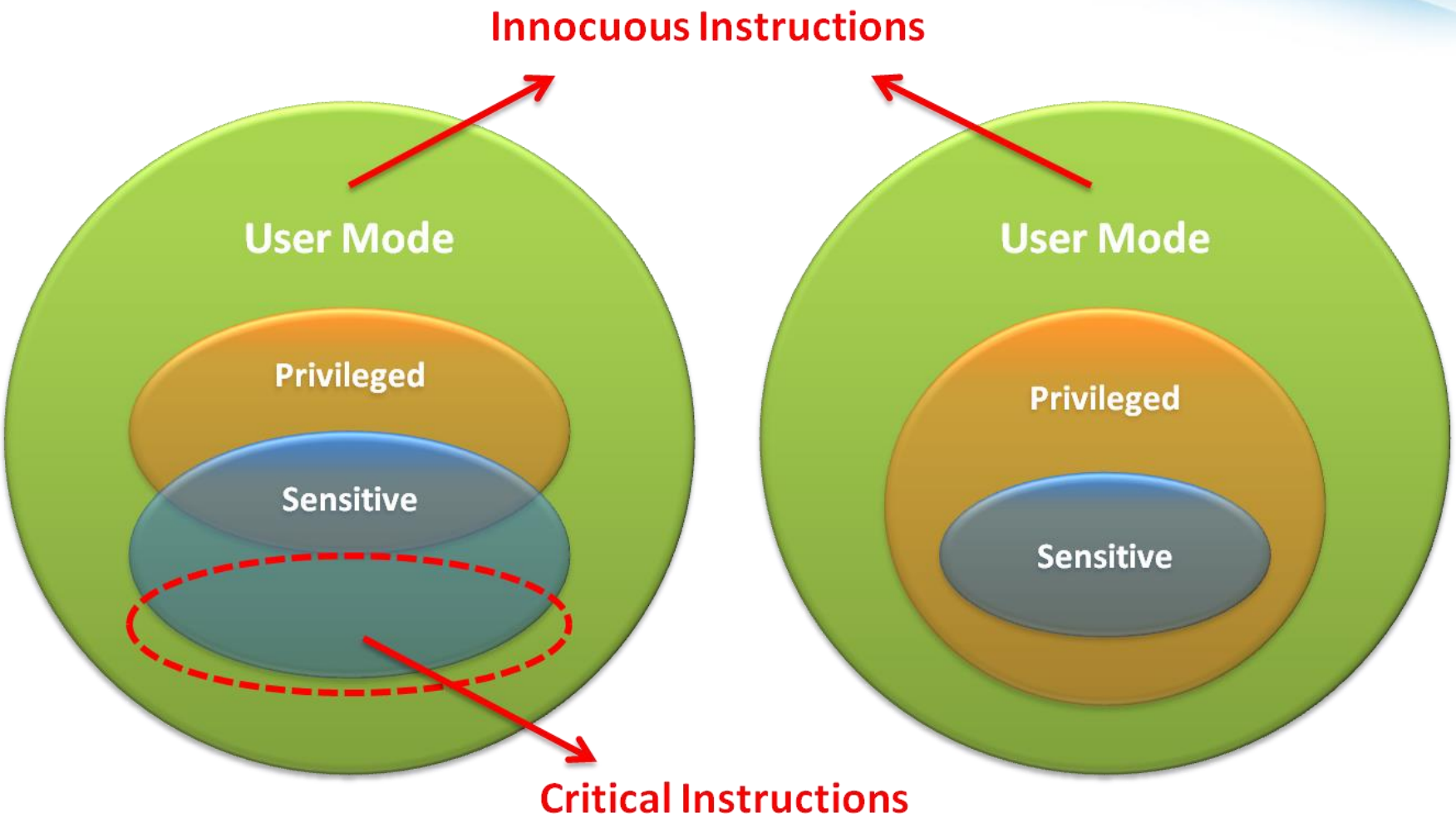
- Modern CPU status is usually classified as several modes.
- In general, we conceptually divide them into two modes :
  - Kernel mode (Ring 0)
    - CPU may perform any operation allowed by its architecture, including any instruction execution, IO operation, area of memory access, and so on.
    - Traditional OS kernel runs in Ring 1 mode.
  - User mode (Ring 1 ~ 3)
    - CPU can typically only execute a subset of those available instructions in kernel mode.
    - Traditional application runs in Ring 3 mode.



CPU Architecture

- By the classification of CPU modes, we divide instructions into following types :
  - Privileged instructions
    - Those instructions that trap if the machine is in user mode and do not trap if the machine is in kernel mode.
  - Sensitive instructions
    - Those instructions that interact with hardware, which include control-sensitive and behavior-sensitive instructions.
  - Innocuous instructions
    - All other instructions.
  - Critical instructions
    - Those sensitive but not privileged instructions.

CPU Architecture



CPU Architecture

- What is trap ?
  - When CPU is running in user mode, some internal or external events, which need to be handled in kernel mode, take place.
  - Then CPU will jump to hardware exception handler vector, and execute system operations in kernel mode.
- Trap types :
  - System Call
    - Invoked by applications in user mode.
    - For example, application ask OS for system IO.
  - Hardware Interrupts
    - Invoked by some hardware events in any mode.
    - For example, hardware clock timer trigger event.
  - Exception
    - Invoked when unexpected error or system malfunction occur.
    - For example, execute privilege instructions in user mode.

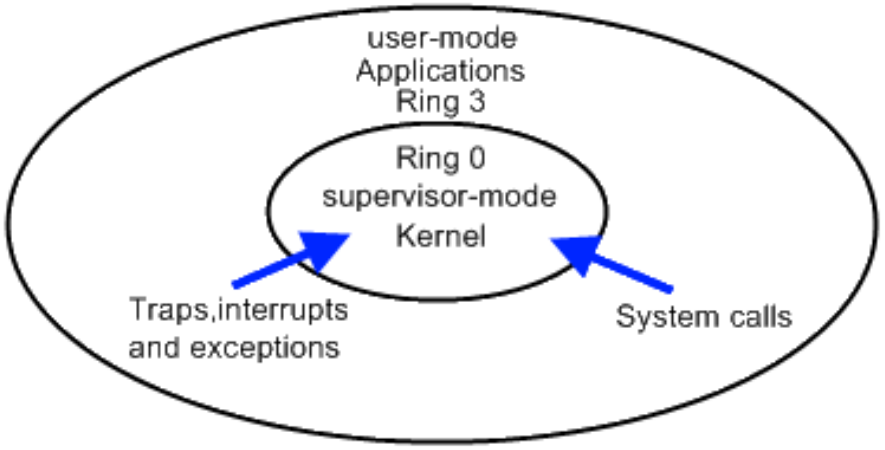
Trap and Emulate Model

- If we want CPU virtualization to be efficient, how should we implement the VMM ?
  - We should make guest binaries run on CPU as fast as possible.
  - Theoretically speaking, if we can run all guest binaries natively, there will NO overhead at all.
  - But we cannot let guest OS handle everything, VMM should be able to control all hardware resources.

- Solution :
  - Ring Compression
    - Shift traditional OS from kernel mode (Ring 0) to user mode (Ring 1), and run VMM in kernel mode.
    - Then VMM will be able to intercept all trapping events.

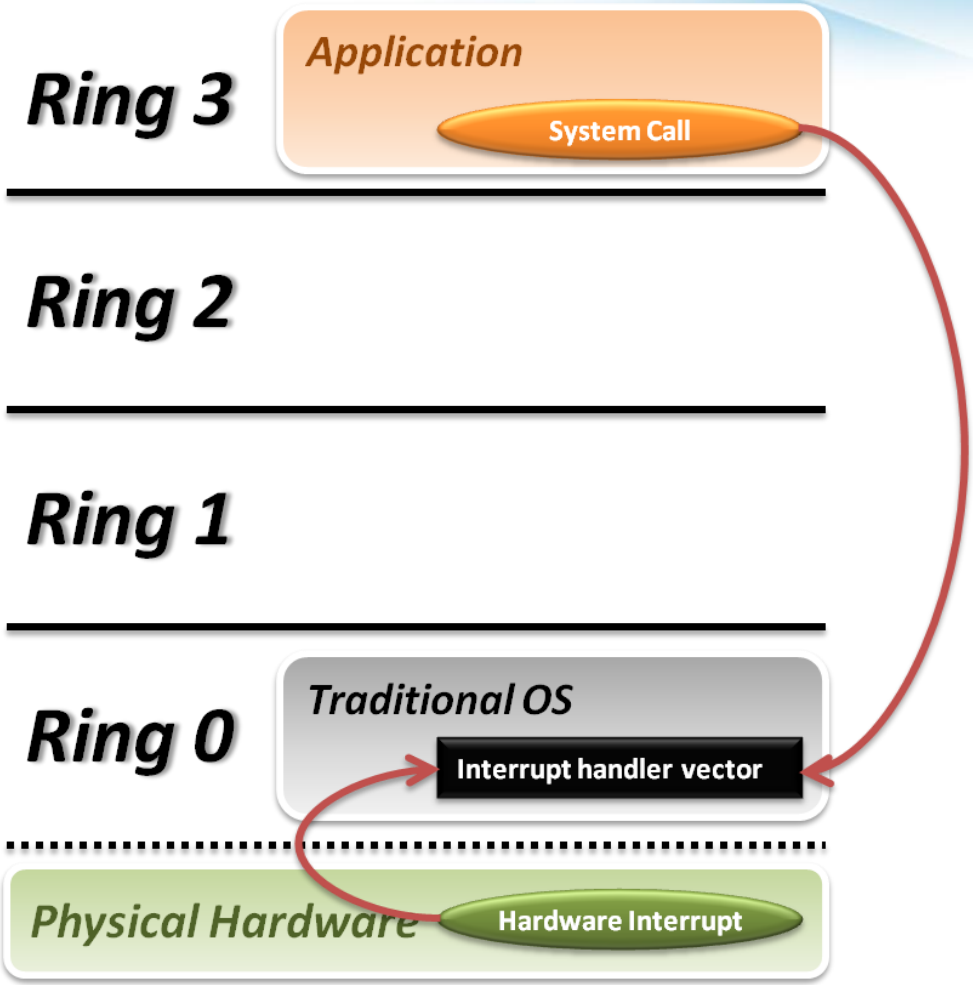
Trap and Emulate Model

- VMM virtualization paradigm *(trap and emulate)* :
  1. Let normal instructions of guest OS run directly on processor in user mode.
  2. When executing privileged instructions, hardware will make processor trap into the VMM.
  3. The VMM emulates the effect of the privileged instructions for the guest OS and return to guest.



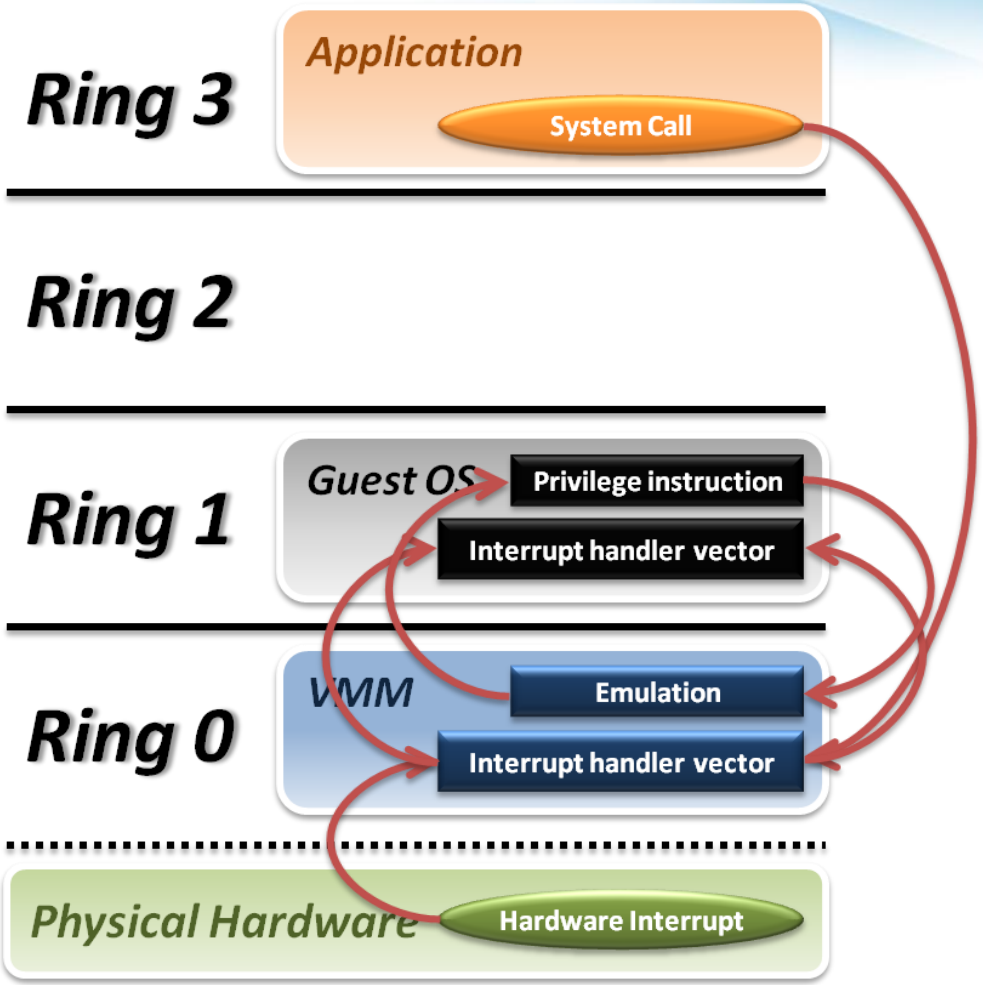
Trap and Emulate Model

- Traditional OS :
  - When an application invokes a system call :
    - CPU will trap to interrupt handler vector in OS.
    - CPU will switch to kernel mode (Ring 0) and execute OS instructions.
  - When hardware event :
    - Hardware will interrupt CPU execution, and jump to interrupt handler in OS.



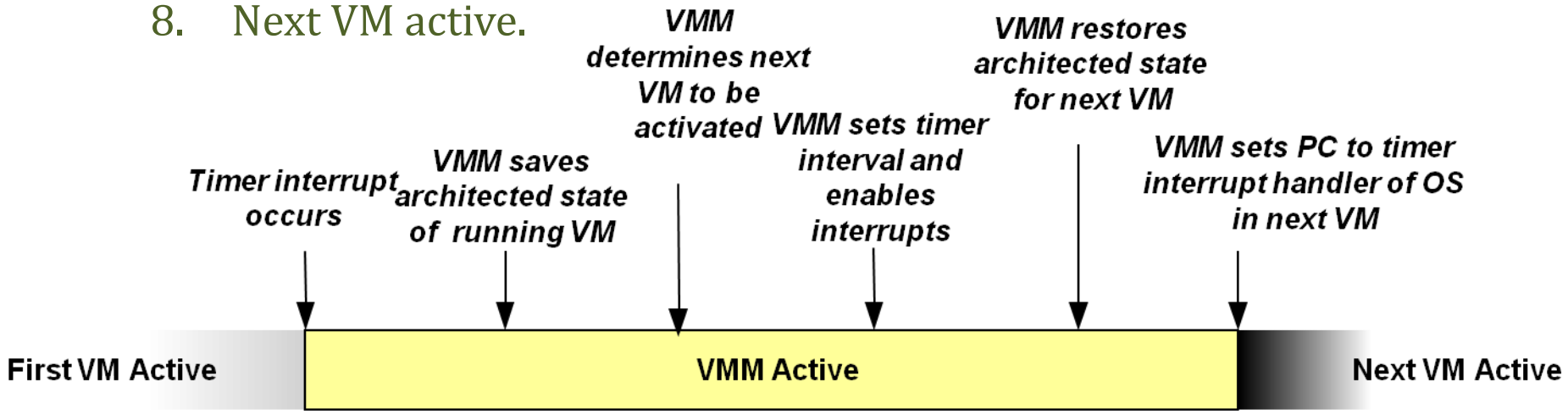
Trap and Emulate Model

- VMM and Guest OS :
  - System Call
    - CPU will trap to interrupt handler vector of VMM.
    - VMM jump back into guest OS.
  - Hardware Interrupt
    - Hardware make CPU trap to interrupt handler of VMM.
    - VMM jump to corresponding interrupt handler of guest OS.
  - Privilege Instruction
    - Running privilege instructions in guest OS will be trapped to VMM for instruction emulation.
    - After emulation, VMM jump back to guest OS.



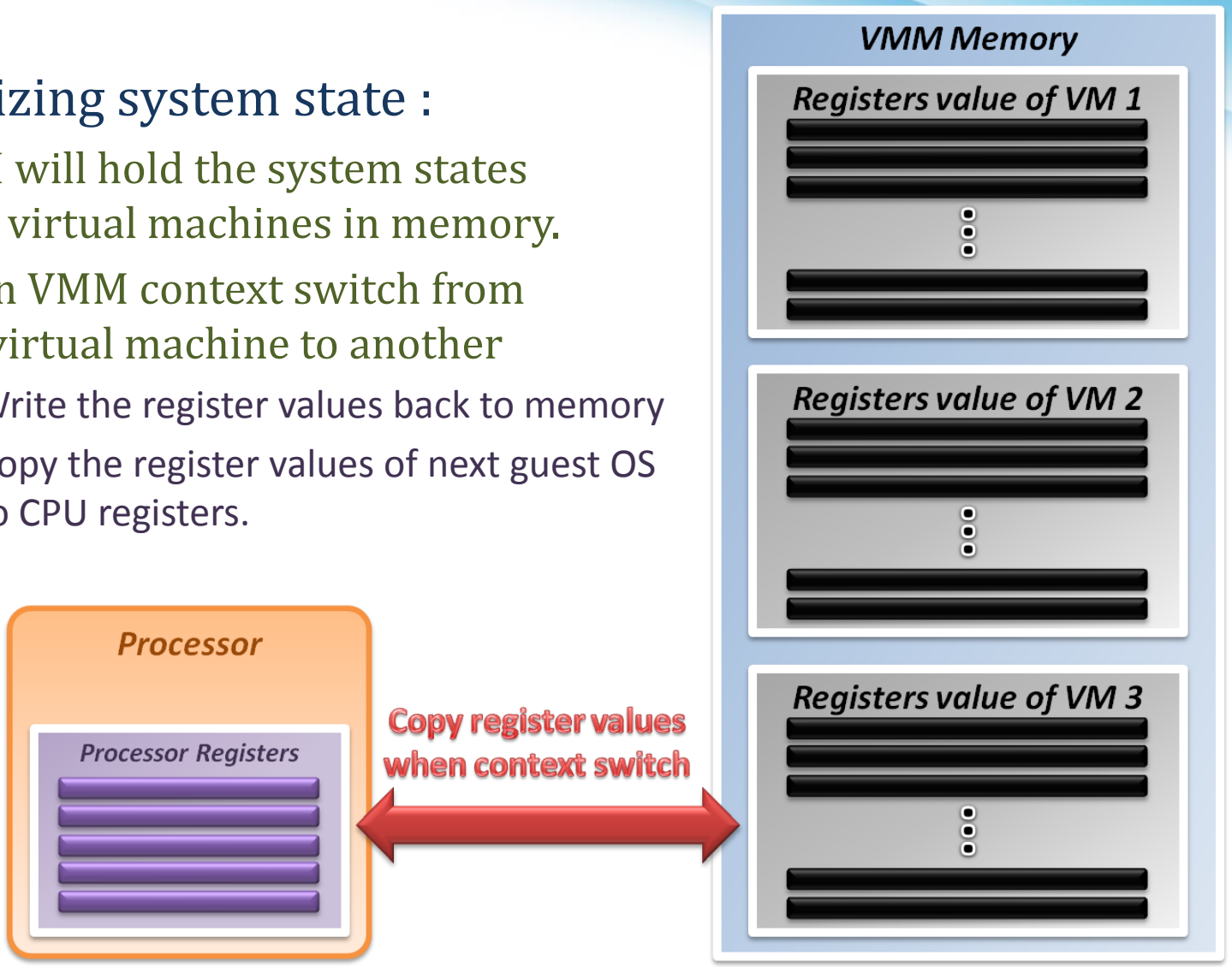
Context Switch

- Steps of VMM switch different virtual machines :
  1. Timer Interrupt in running VM.
  2. Context switch to VMM.
  3. VMM saves state of running VM.
  4. VMM determines next VM to execute.
  5. VMM sets timer interrupt.
  6. VMM restores state of next VM.
  7. VMM sets PC to timer interrupt handler of next VM.
  8. Next VM active.



System State Management

- Virtualizing system state :
  - VMM will hold the system states of all virtual machines in memory.
  - When VMM context switch from one virtual machine to another
    - Write the register values back to memory
    - Copy the register values of next guest OS to CPU registers.



Virtualization Theorem

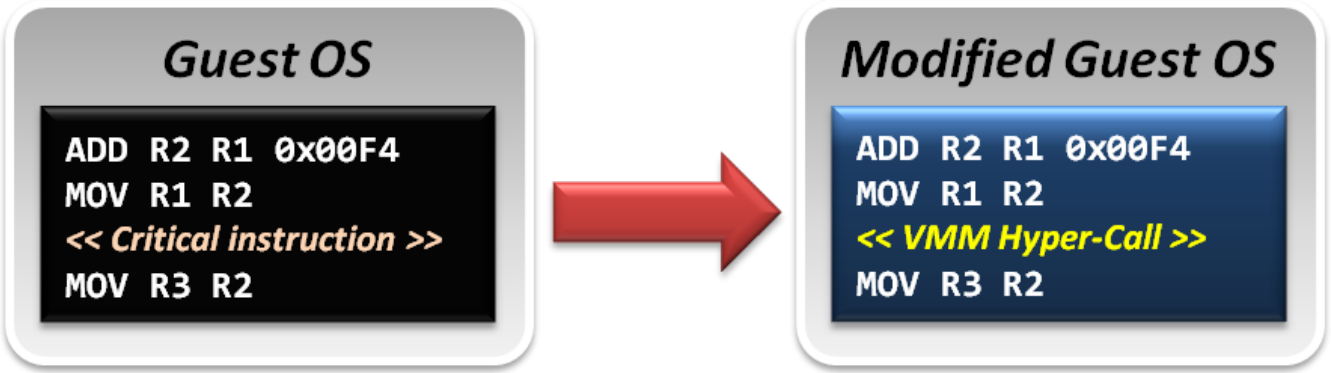
- Subset theorem :
  - For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.
- Recursive Emulation :
  - A conventional third-generation computer is recursively virtualizable if
    - It is virtualizable
    - VMM without any timing dependencies can be constructed for it.
- Under this theorem, x86 architecture cannot be virtualized directly. Other techniques are needed.

Virtualization Techniques

- How to virtualize unvirtualizable hardware :
  - Para-virtualization
    - Modify guest OS to skip the critical instructions.
    - Implement some hyper-calls to trap guest OS to VMM.
  - Binary translation
    - Use emulation technique to make hardware virtualizable.
    - Skip the critical instructions by means of these translations.
  - Hardware assistance
    - Modify or enhance ISA of hardware to provide virtualizable architecture.
    - Reduce the complexity of VMM implementation.

Para-Virtualization

- Para-Virtualization implementation :
  - In para-virtualization technique, guest OS should be modified to prevent invoking critical instructions.
  - Instead of knowing nothing about hypervisor, guest OS will be aware of the existence of VMM, and collaborate with VMM smoothly.
  - VMM will provide the hyper-call interfaces, which will be the communication channel between guest and host.



Binary Translation

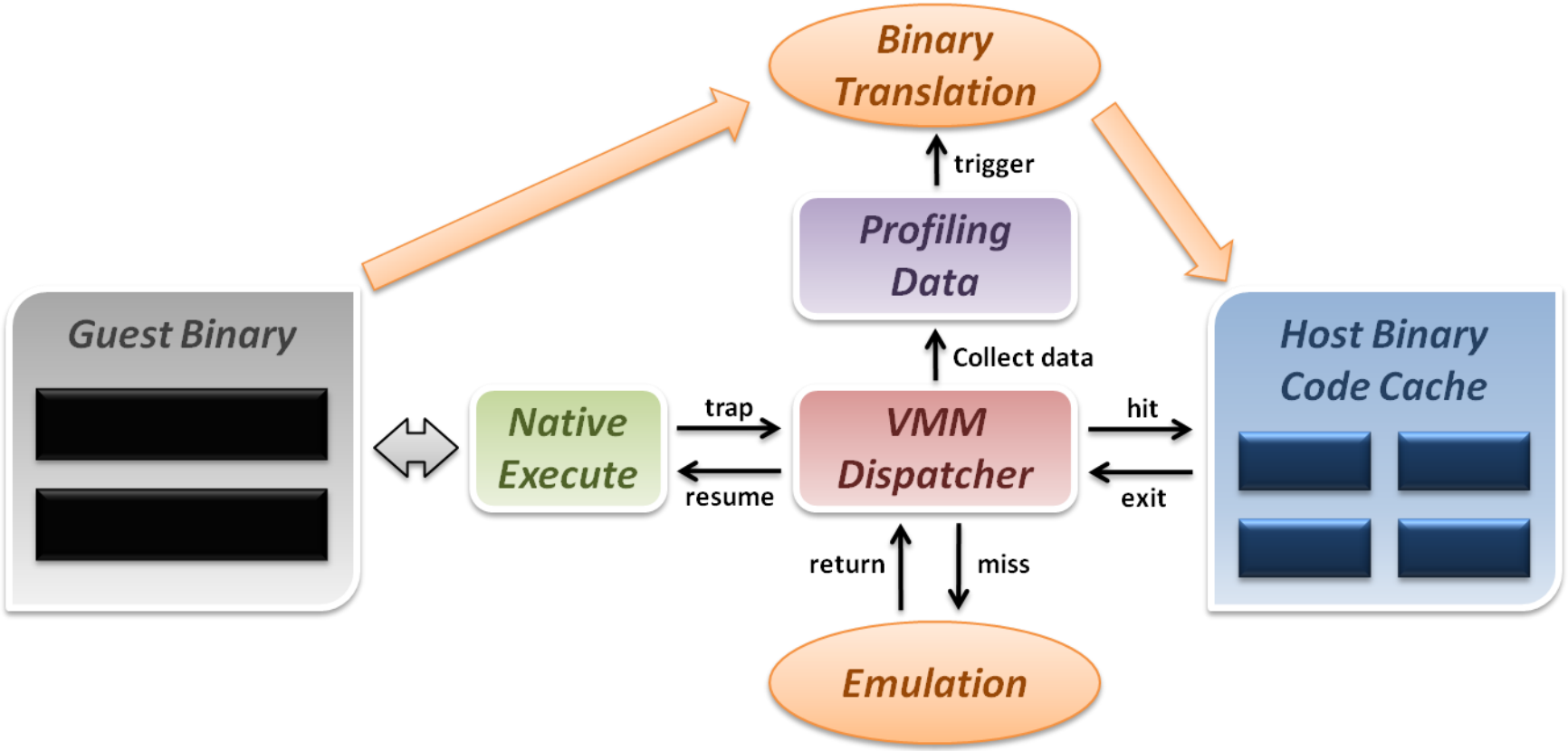
- In emulation techniques :
  - Binary translation module is used to optimize binary code blocks, and translate binaries from guest ISA to host ISA.
- In virtualization techniques :
  - Binary translation module is used to skip or modify the guest OS binary code blocks which include critical instructions.
  - Translate those critical instructions into some privilege instructions which will trap to VMM for further emulation.

Binary Translation

- Static approach vs. Dynamic approach :
  - Static binary translation
    - The entire executable file is translated into an executable of the target architecture.
    - This is very difficult to do correctly, since not all the code can be discovered by the translator.
  - Dynamic binary translation
    - Looks at a short sequence of code, typically on the order of a single basic block, translates it and caches the resulting sequence.
    - Code is only translated as it is discovered and when possible, branch instructions are made to point to already translated and saved code.

Binary Translation

- Dynamic binary translation and optimization
  - VMM can dynamically translate binary code and collect profiling data for further optimization.



Some Difficulties

- Difficulties of binary translation :
  - Self-modifying code
    - If guest OS will modify its own binary code in runtime, binary translation need to flush the responding code cache and retranslate the code block.
  - Self-reference code
    - If guest code need to reference(read) its own binary code in runtime, VMM need to make it referring back to original guest binaries location.
  - Real-time system
    - For some timing critical guest OS, emulation environment will lose precise timing, and this problem cannot be perfectly solved yet.
- Difficulty of para-virtualization :
  - Guest OS modification
    - User should at least has the source code of guest OS and modify its kernel; otherwise, para-virtualization cannot be used.

Emulation techniques

Trap and emulate model

Hardware assistance

CPU VIRTUALIZATION

Hardware Solution

- Why are there so many problems and difficulties ?
  - Critical instructions do not trap in user mode.
  - Even if we make those critical instructions trap, their semantic may also be changed; which is not acceptable.
- In short, legacy processors were not designed for virtualization purpose at the beginning.
  - If processors can be aware of the different behaviors between guest and host, the VMM design will be more efficient and simple.

Hardware Solution

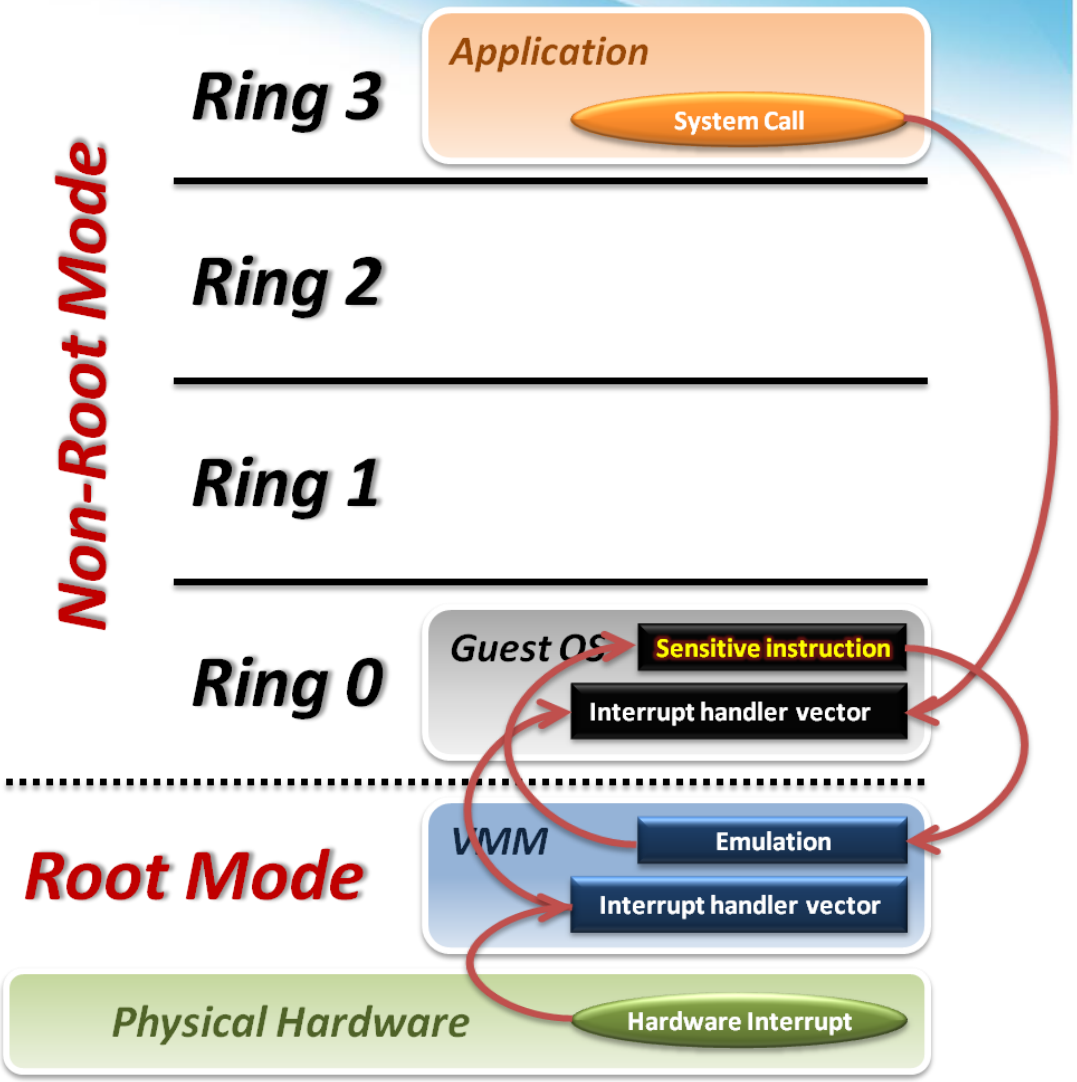
- Let's go back to trap model :
  - Some trap types do not need the VMM involvement.
    - For example, all system calls invoked by application in guest OS should be caught by gust OS only. There is no need to trap to VMM and then forward it back to guest OS, which will introduce context switch overhead.
  - Some critical instructions should not be executed by guest OS.
    - Although we make those critical instructions trap to VMM, VMM cannot identify whether this trapping action is caused by the emulation purpose or the real OS execution exception.
- Solution :
  - We need to redefine the semantic of some instructions.
  - We need to introduce new CPU control paradigm.

Intel VT-x

- In order to straighten those problems out, Intel introduces one more operation mode of x86 architecture.
  - VMX Root Operation (Root Mode)
    - All instruction behaviors in this mode are no different to traditional ones.
    - All legacy software can run in this mode correctly.
    - VMM should run in this mode and control all system resources.
  - VMX Non-Root Operation (Non-Root Mode)
    - All sensitive instruction behaviors in this mode are redefined.
    - The sensitive instructions will trap to Root Mode.
    - Guest OS should run in this mode and be fully virtualized through typical "*trap and emulation model*".

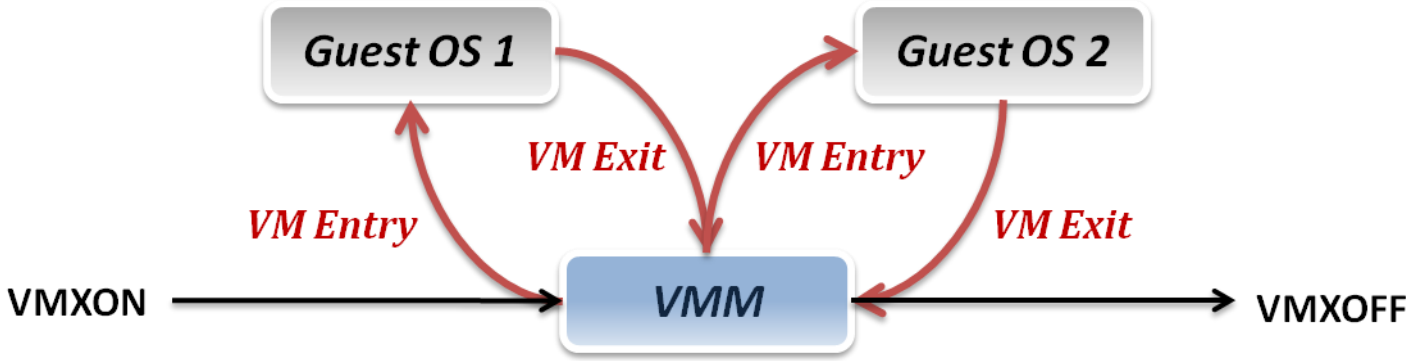
Intel VT-x

- VMM with VT-x :
  - System Call
    - CPU will directly trap to interrupt handler vector of guest OS.
  - Hardware Interrupt
    - Still, hardware events need to be handled by VMM first.
  - Sensitive Instruction
    - Instead of trap all privilege instructions, running guest OS in Non-root mode will trap sensitive instruction only.



Context Switch

- VMM switch different virtual machines with Intel VT-x :
  - **VMXON/VMXOFF**
    - These two instructions are used to turn on/off CPU Root Mode.
  - VM Entry
    - This is usually caused by the execution of **VMLAUNCH/VMRESUME** instructions, which will switch CPU mode from Root Mode to Non-Root Mode.
  - VM Exit
    - This may be caused by many reasons, such as hardware interrupts or sensitive instruction executions.
    - Switch CPU mode from Non-Root Mode to Root Mode.

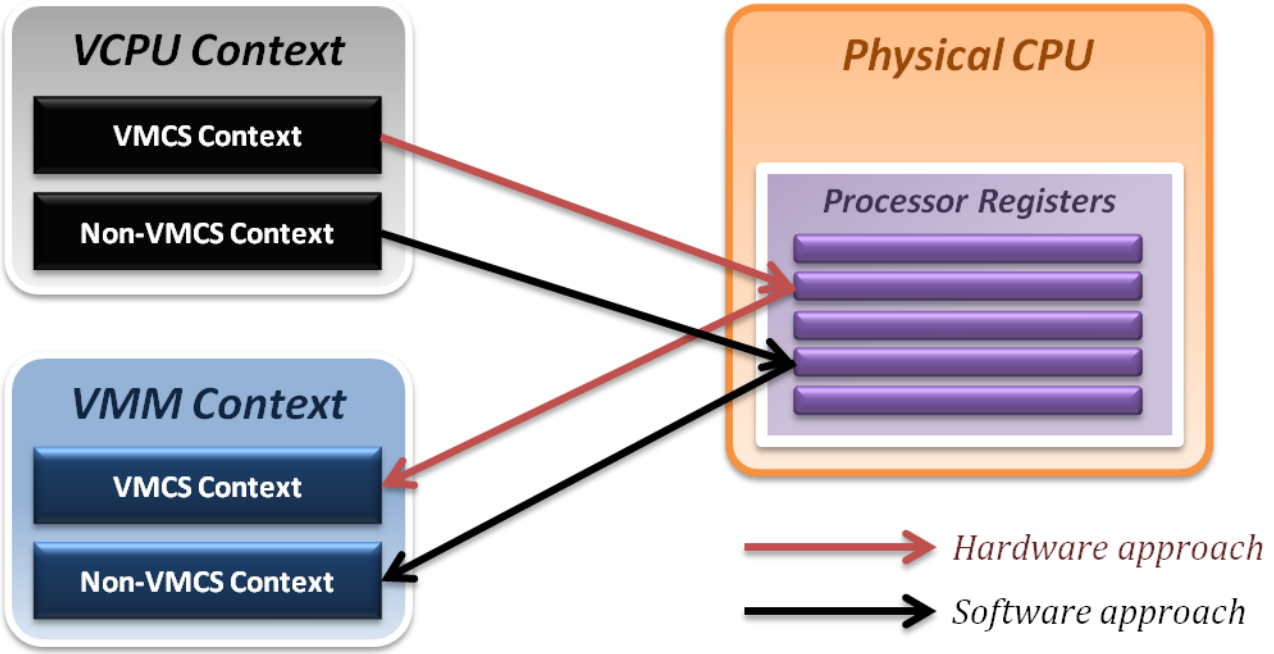


System State Management

- Intel introduces a more efficient hardware approach for register switching, **VMCS** (*Virtual Machine Control Structure*) :
  - State Area
    - Store host OS system state when VM-Entry.
    - Store guest OS system state when VM-Exit.
  - Control Area
    - Control instruction behaviors in Non-Root Mode.
    - Control VM-Entry and VM-Exit process.
  - Exit Information
    - Provide the VM-Exit reason and some hardware information.
- Whenever VM Entry or VM Exit occur, CPU will automatically read or write corresponding information into VMCS.

System State Management

- Binding virtual machine to virtual CPU
  - VCPU (Virtual CPU) contains two parts
    - VMCS maintains virtual system states, which is approached by hardware.
    - Non-VMCS maintains other non-essential system information, which is approached by software.
  - VMM needs to handle Non-VMCS part.



CPU Virtualization Summary

- Emulation technique
  - Interpretation and binary translation approaches
  - System state mapping and performance issue
    - Translation chaining, Dynamic binary optimization
- Virtualization technique
  - Modern CPU architecture
  - Trap and emulation model
  - Critical instruction issue
    - Para-virtualization, Dynamic binary translation
- Hardware assistance
  - Intel VT-x approach
    - Root Mode & Non-Root Mode

VMware , Xen , KVM

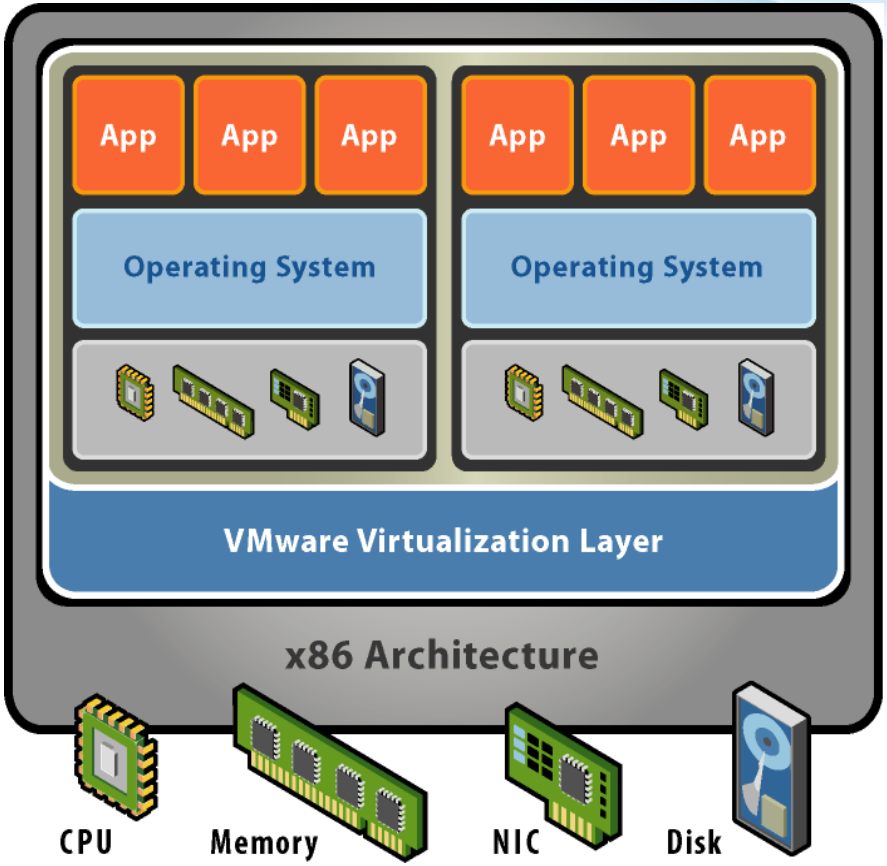
ECOSYSTEM

Venders and Projects

- Virtual machine venders :
  - VMware
    - The company was founded in 1998 and is based in Palo Alto, California. The company is majority owned by EMC Corporation.
    - Implement both type-1 and type-2 VM.
  - Xen
    - First developed in University of Cambridge Computer Laboratory.
    - As of 2010 the Xen community develops and maintains Xen as free software, licensed under the GNU General Public License (GPLv2).
    - Implement para-virtualization.
- Virtual machine project :
  - KVM (Kernel-based Virtual Machine)
    - A Linux kernel virtualization infrastructure.
    - As of 2010, KVM supports native virtualization using Intel VT-x or AMD-V.

VMware

- Basic properties :
  - Separate OS and hardware – break hardware dependencies
  - OS and Application as single unit by encapsulation
  - Strong fault and security isolation
  - Standard, HW independent environments can be provisioned anywhere
  - Flexibility to chose the right OS for the right application



VMware Virtualization Stack

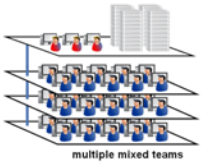
3 Management Automation

Infrastructure Optimization

Desktop Management

Business Continuity

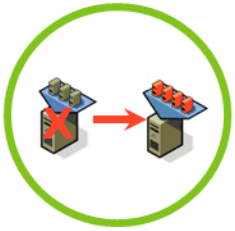
Software Lifecycle



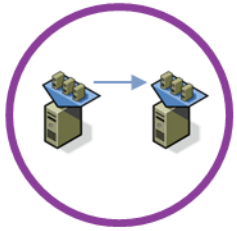
2 Distributed Virtualization

Resource Mgt

Availability

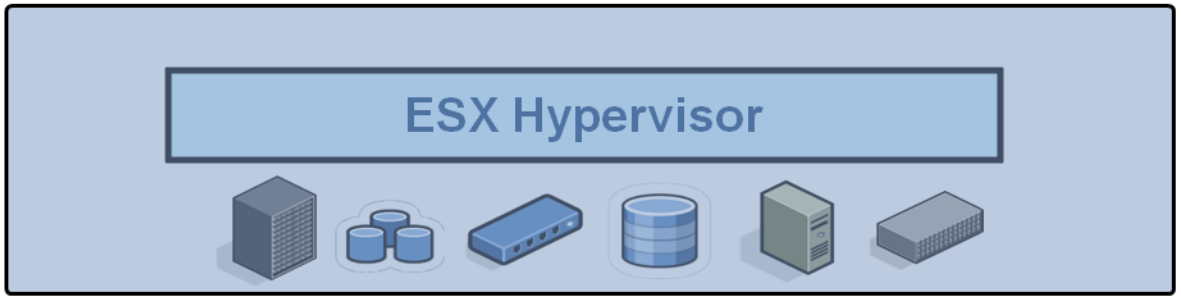


Mobility



Security

1 Virtualization Platforms



VMware Major Products

- VMware GSX Server
  - Run multiple servers on your server
  - Hosted Architecture
  - Available for Linux hosts and Windows hosts
- VMware ESX Server
  - Quality of Service
  - High-performance I/O
  - Host-less Architecture (bare-metal)

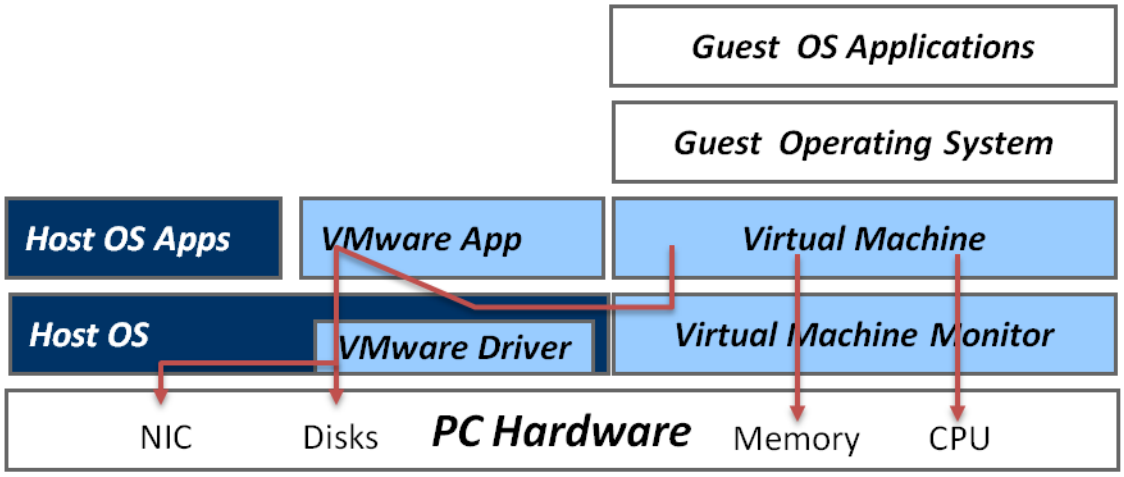
VMware GSX Server Architecture

Host Mode

VMware, acting as an application, uses the host to access other devices such as the hard disk, floppy, or network card

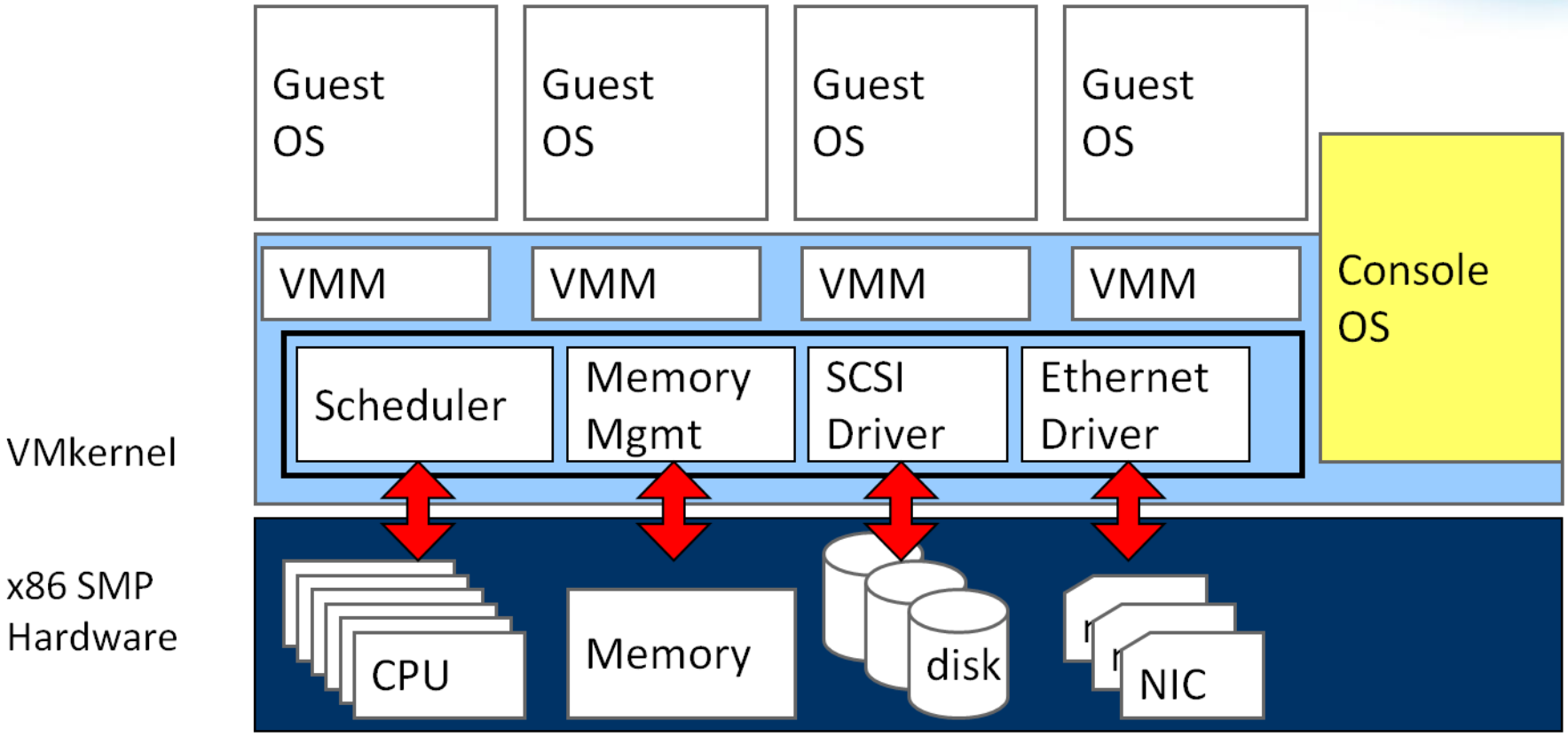
VMM Mode

The VMware Virtual machine monitor allows each guest OS to directly access the processor (direct execution)



VMware achieves both near-native execution speed and broad device support by transparently switching between Host Mode and VMM Mode.

VMware ESX Server Architecture



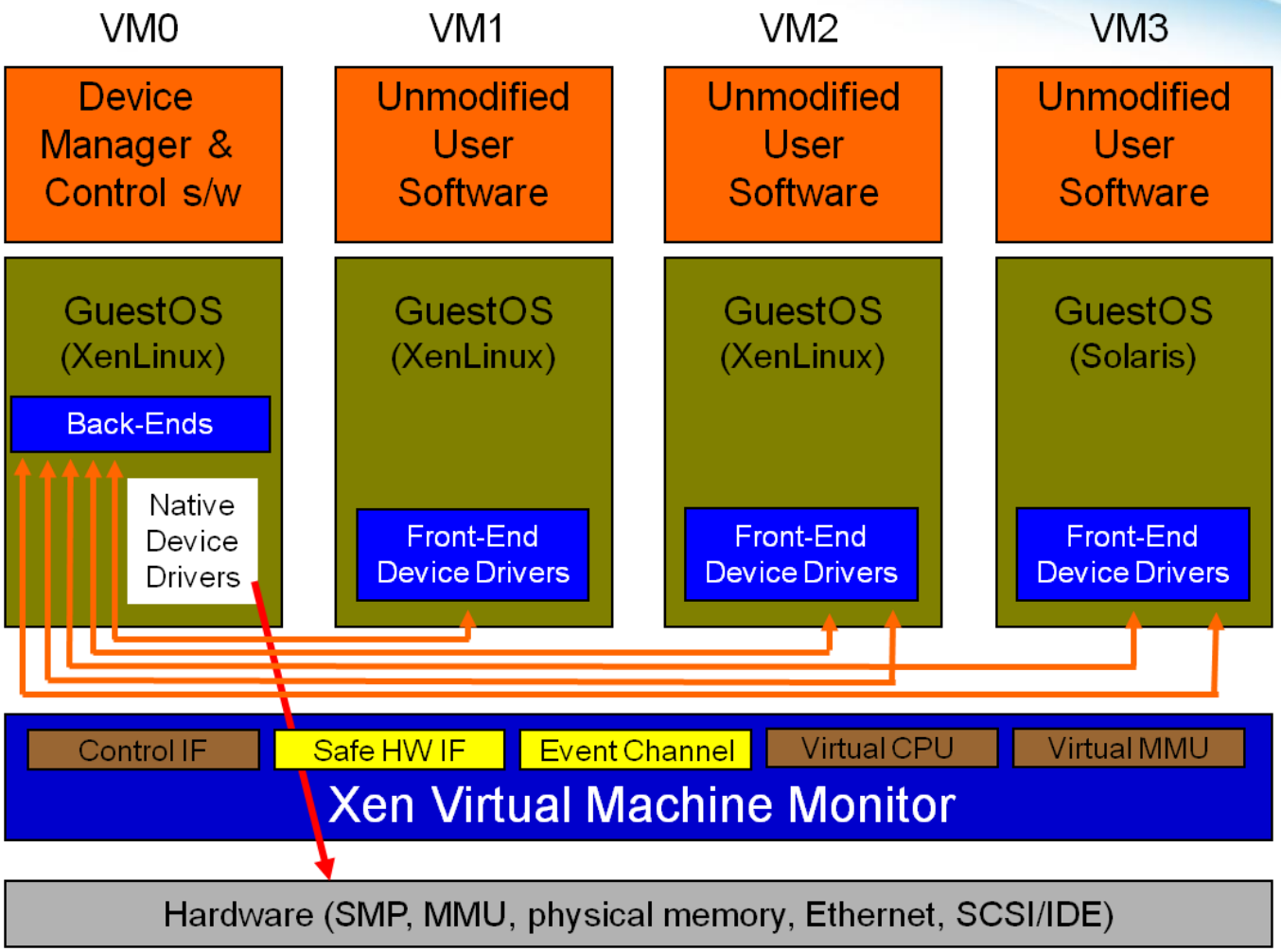
Xen

- Basic properties :
  - Para-virtualization
    - Achieve high performance even on its host architecture (x86) which has a reputation for non-cooperation with traditional virtualization techniques.
  - Hardware assisted virtualization
    - Both Intel and AMD have contributed modifications to Xen to support their respective Intel VT-x and AMD-V architecture extensions.
  - Live migration
    - The LAN iteratively copies the memory of the virtual machine to the destination without stopping its execution.
- Implement system:
  - Novell's SUSE Linux Enterprise 10
  - Red Hat's RHEL 5
  - Sun Microsystems' Solaris

Para-virtualization in Xen

- Xen extensions to x86 arch
  - Like x86, but Xen invoked for privileged instructions
  - Avoids binary rewriting
  - Minimize number of privilege transitions into Xen
  - Modifications relatively simple and self-contained
- Modify kernel to understand virtualized environment
  - Wall-clock time vs. virtual processor time
    - Desire both types of alarm timer
  - Expose real resource availability
    - Enables OS to optimize its own behaviour

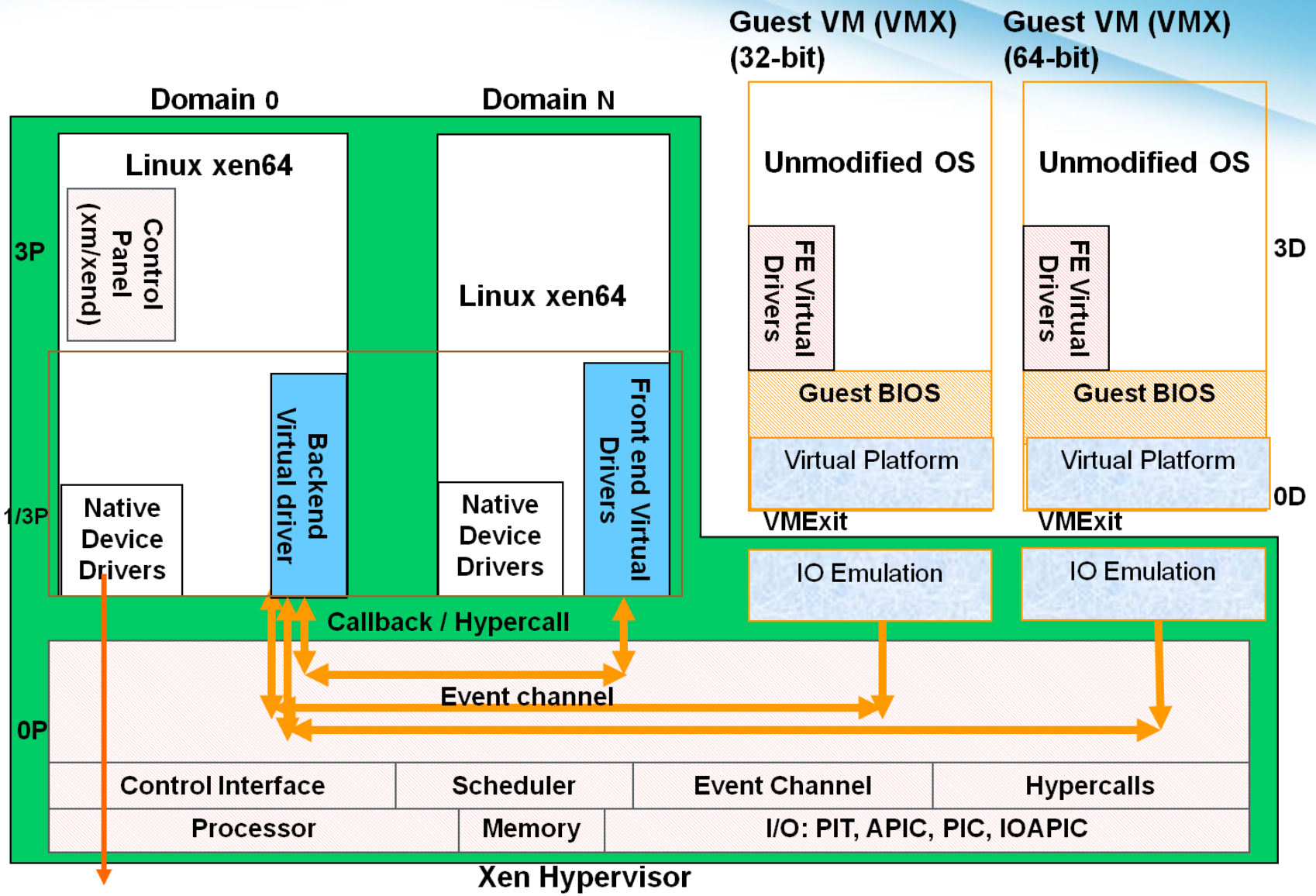
Original Xen Architecture



Hardware Assistance in Xen

- Hardware assistance :
  - CPU provides **`VMExit`** for certain privileged instructions
  - Extend page tables used to virtualize memory
- Xen features :
  - Enable Guest OS to be run without modification
    - For example, legacy Linux and Windows
  - Provide simple platform emulation
    - BIOS, apic, iopaic, rtc, Net (pcnet32), IDE emulation
  - Install para-virtualized drivers after booting for high-performance IO
  - Possibility for CPU and memory para-virtualization
    - Non-invasive hypervisor hints from OS

New Xen Architecture

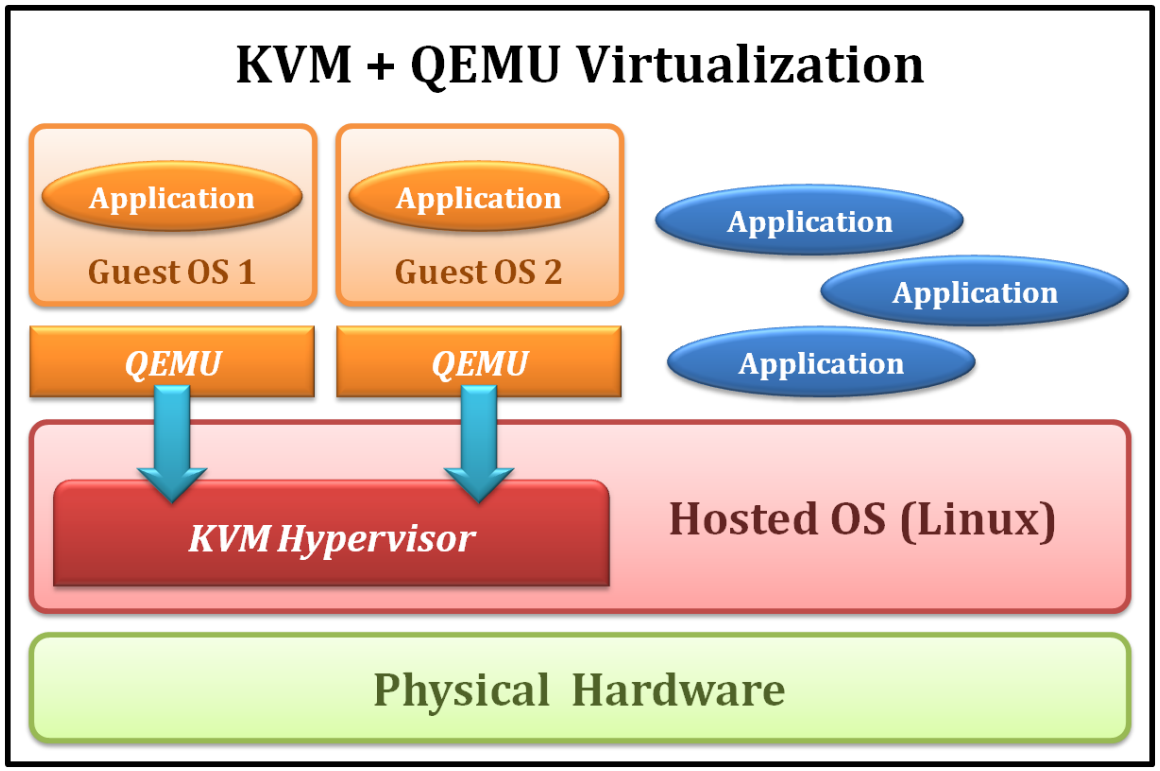


KVM

- KVM (Kernel-based Virtual Machine)
  - Linux host OS
    - The kernel component of KVM is included in mainline Linux, as of 2.6.20.
  - Full-virtualization
    - KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V).
    - Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images.
  - IO device model in KVM :
    - KVM requires a modified QEMU for IO virtualization framework.
    - Improve IO performance by **`virtio`** para-virtualization framework.

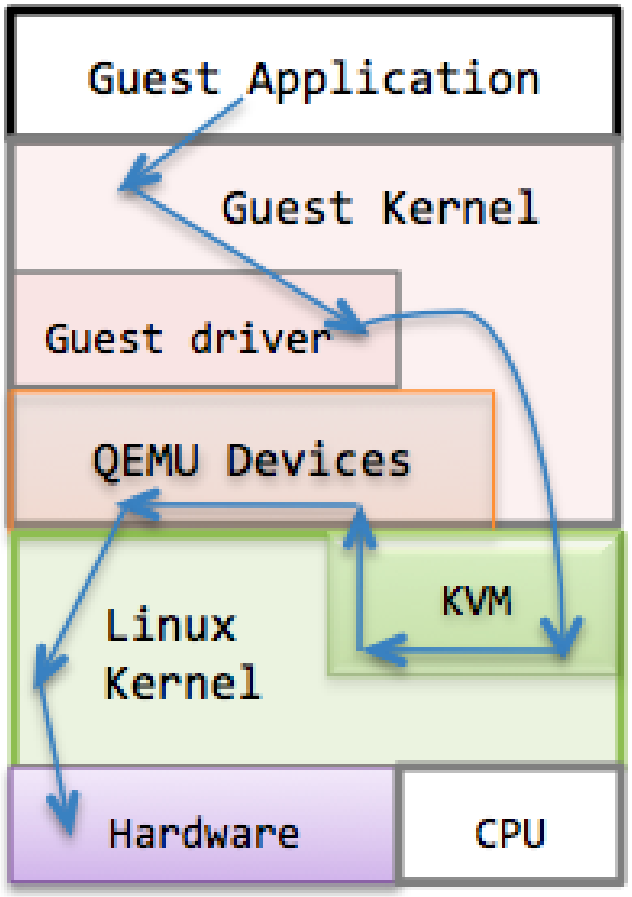
KVM Full Virtualization

- It consists of a loadable kernel module
  - `kvm.ko`
    - provides the core virtualization infrastructure
  - `kvm-intel.ko` / `kvm-amd.ko`
    - processor specific modules



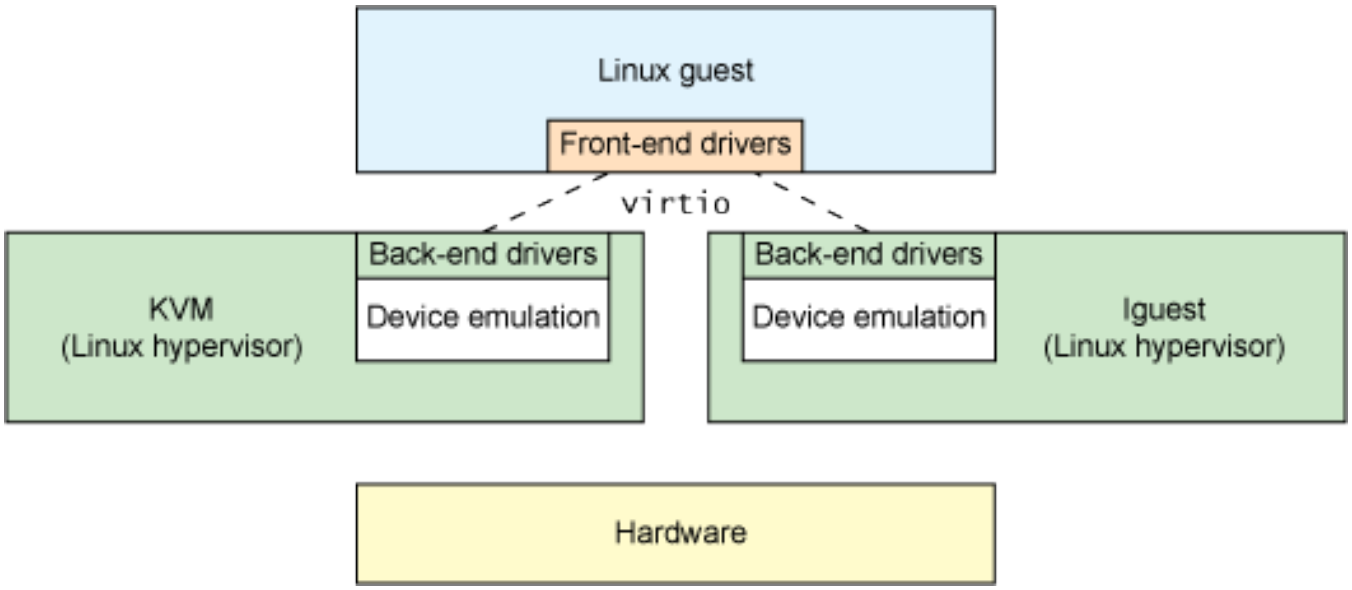
IO Device Model in KVM

- Original approach with full-virtualization
  - Guest hardware accesses are intercepted by KVM
  - QEMU emulates hardware behavior of common devices
    - RTL 8139
    - PIIX4 IDE
    - Cirrus Logic VGA



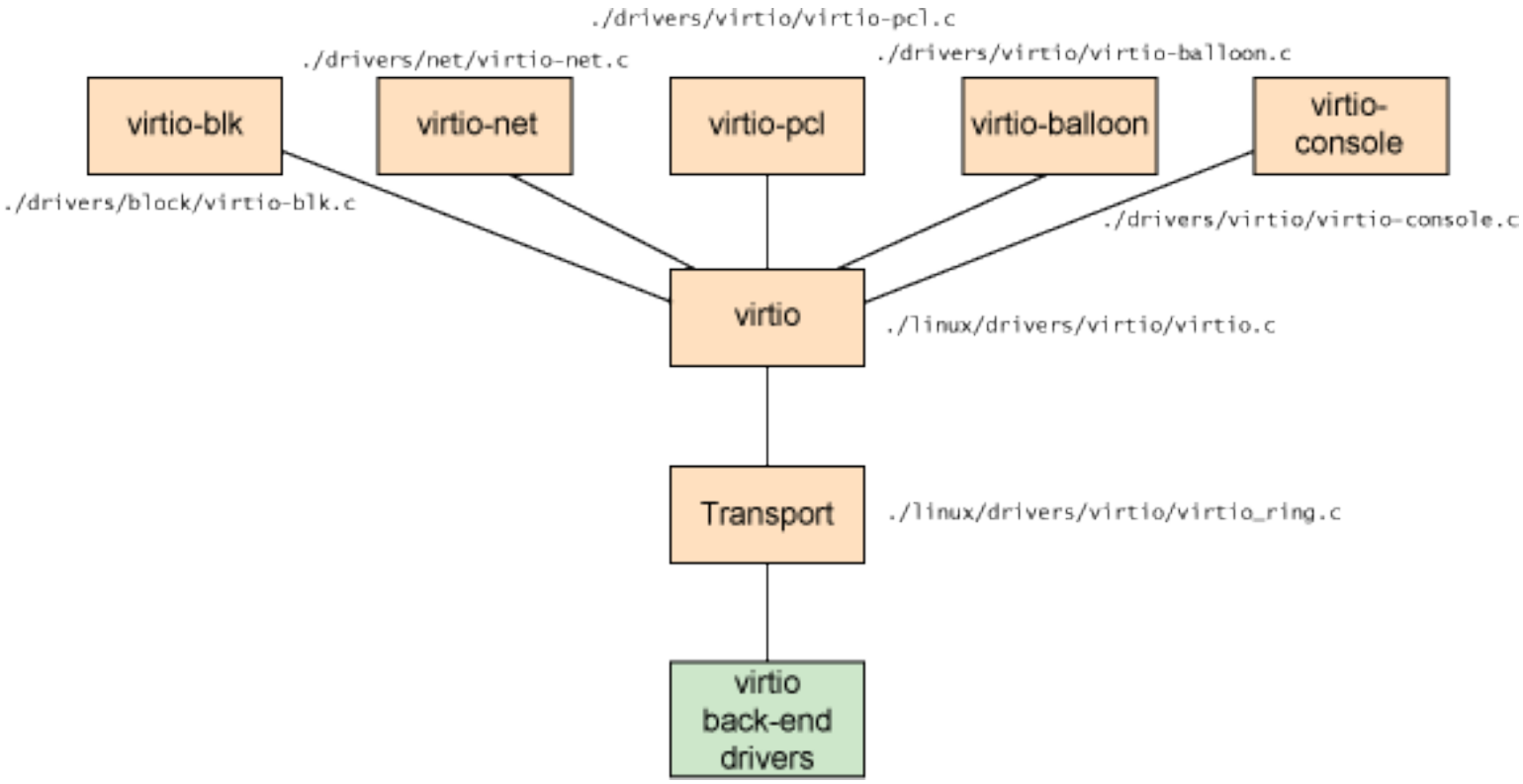
IO Device Model in KVM

- New approach with para-virtualization



IO Device Model in KVM

- **virtio** architecture



Live migration

Cloud properties

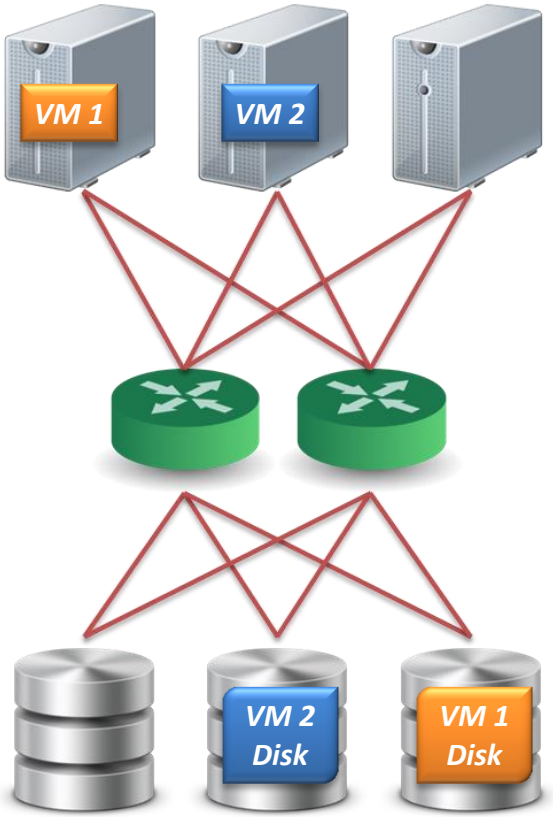
OTHER ISSUES

Other Issues

- Essential technique of cloud properties implementation
  - Live migration of virtual machines
    - Migrate a virtual machine from one physical machine to another in the run time with a small amount of performance down grade.
- Virtualization enabled cloud properties :
  - Scalability
    - Virtual machine system automatic scale up
  - Availability
    - Fault tolerance of hardware and software
  - Manageability
    - Automatic physical to virtual system transformation
  - Performance
    - Dynamically virtual machine level load balancing

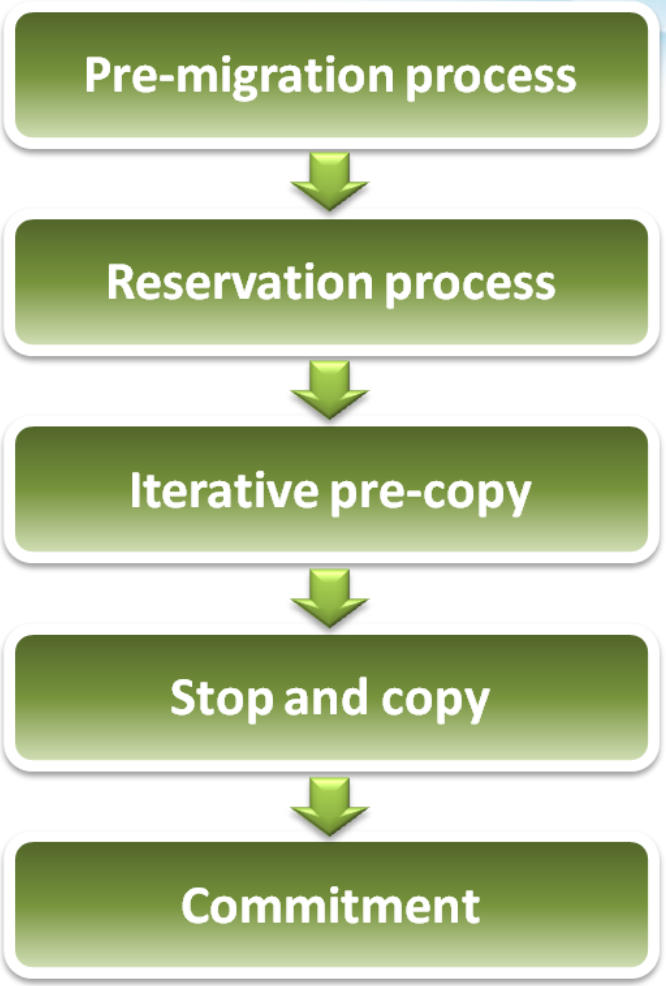
Live Migration Technique

- Pre-assumption :
  - We assume that all storage resources are separated from computing resources.
  - Storage devices of VMs are attached from network :
    - ***NAS***: NFS, CIFS
    - ***SAN***: Fibre Channel
    - ***iSCSI***, network block device
    - ***drdb*** network RAID
  - Require high quality network connection
    - Common L2 network (LAN)
    - L3 re-routing

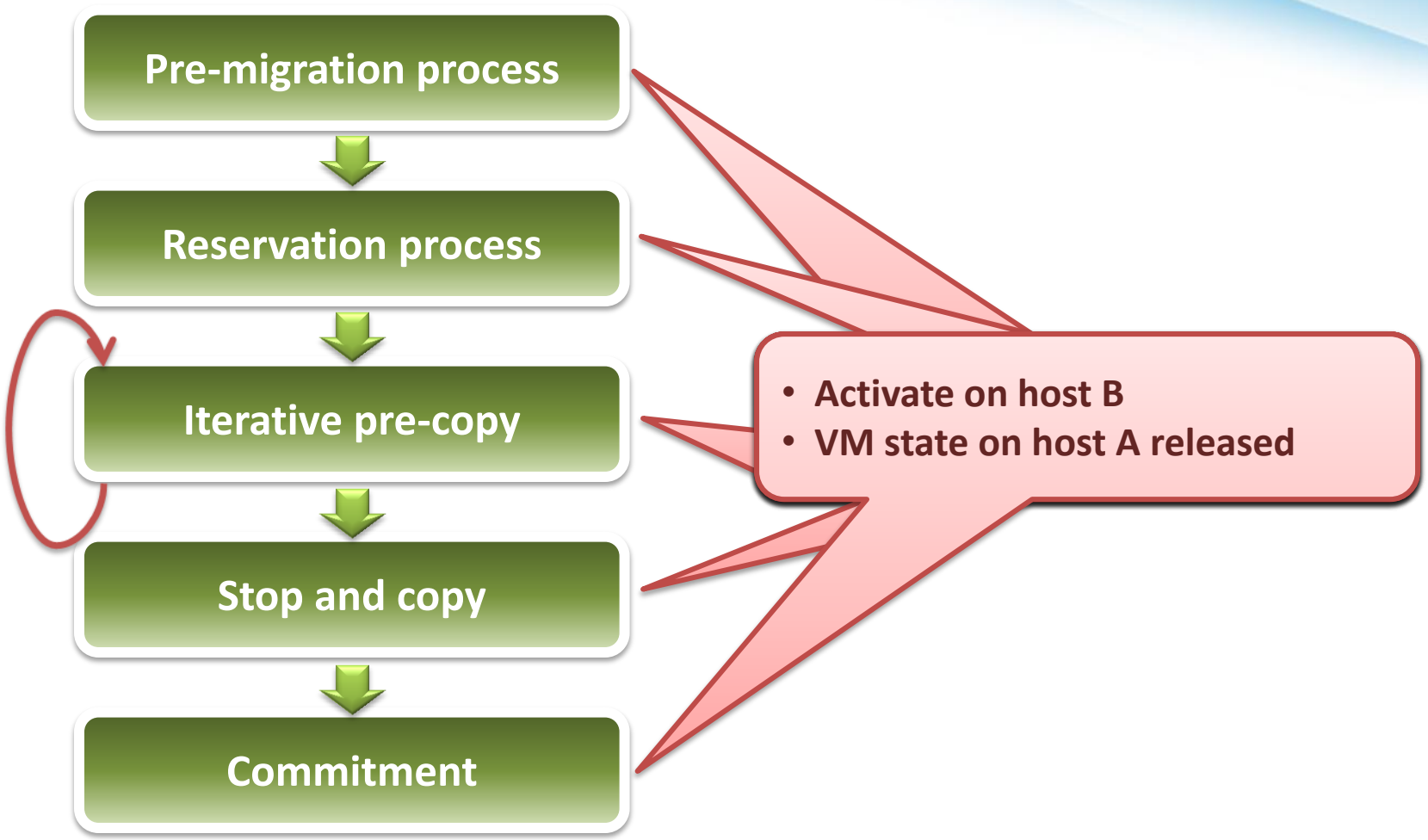


Live Migration Technique

- Challenges of live migration :
  - VMs have lots of state in memory
  - Some VMs have soft real-time requirements :
    - For examples, web servers, databases and game servers, ...etc.
    - Need to minimize down-time
- Relocation strategy :
  1. Pre-migration process
  2. Reservation process
  3. Iterative pre-copy
  4. Stop and copy
  5. Commitment



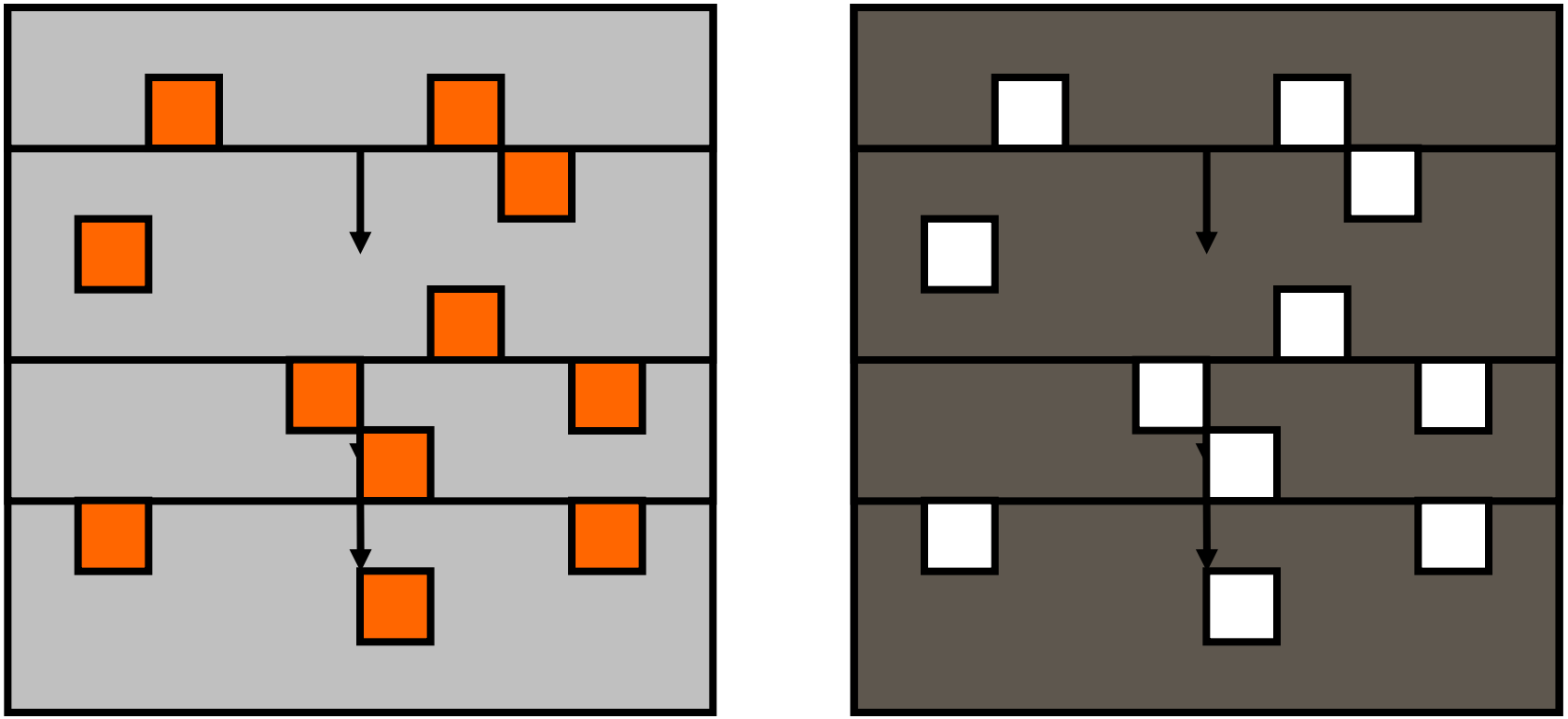
Live Migration Technique



Live Migration Technique

- Live migration process :

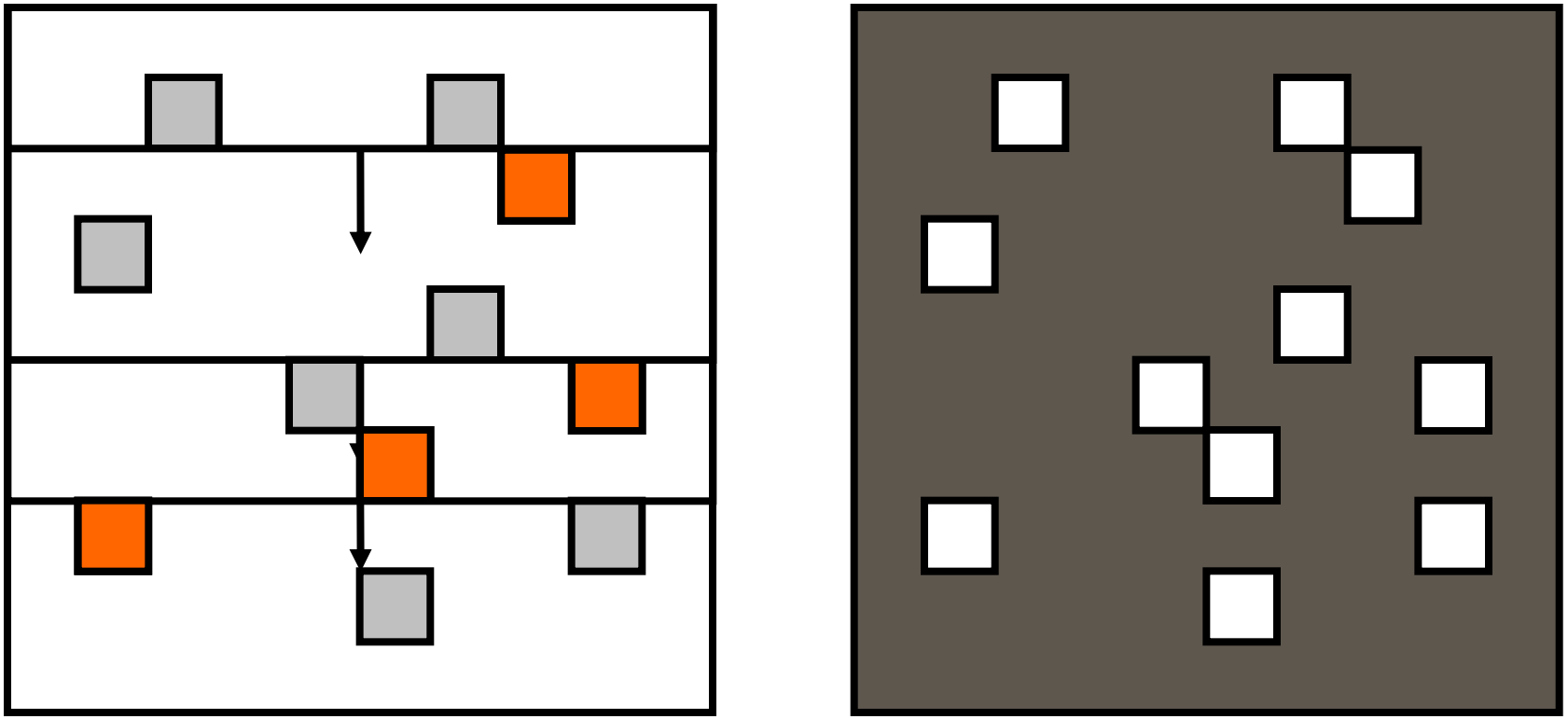
Pre-copy migration : Round 1



Live Migration Technique

- Live migration process :

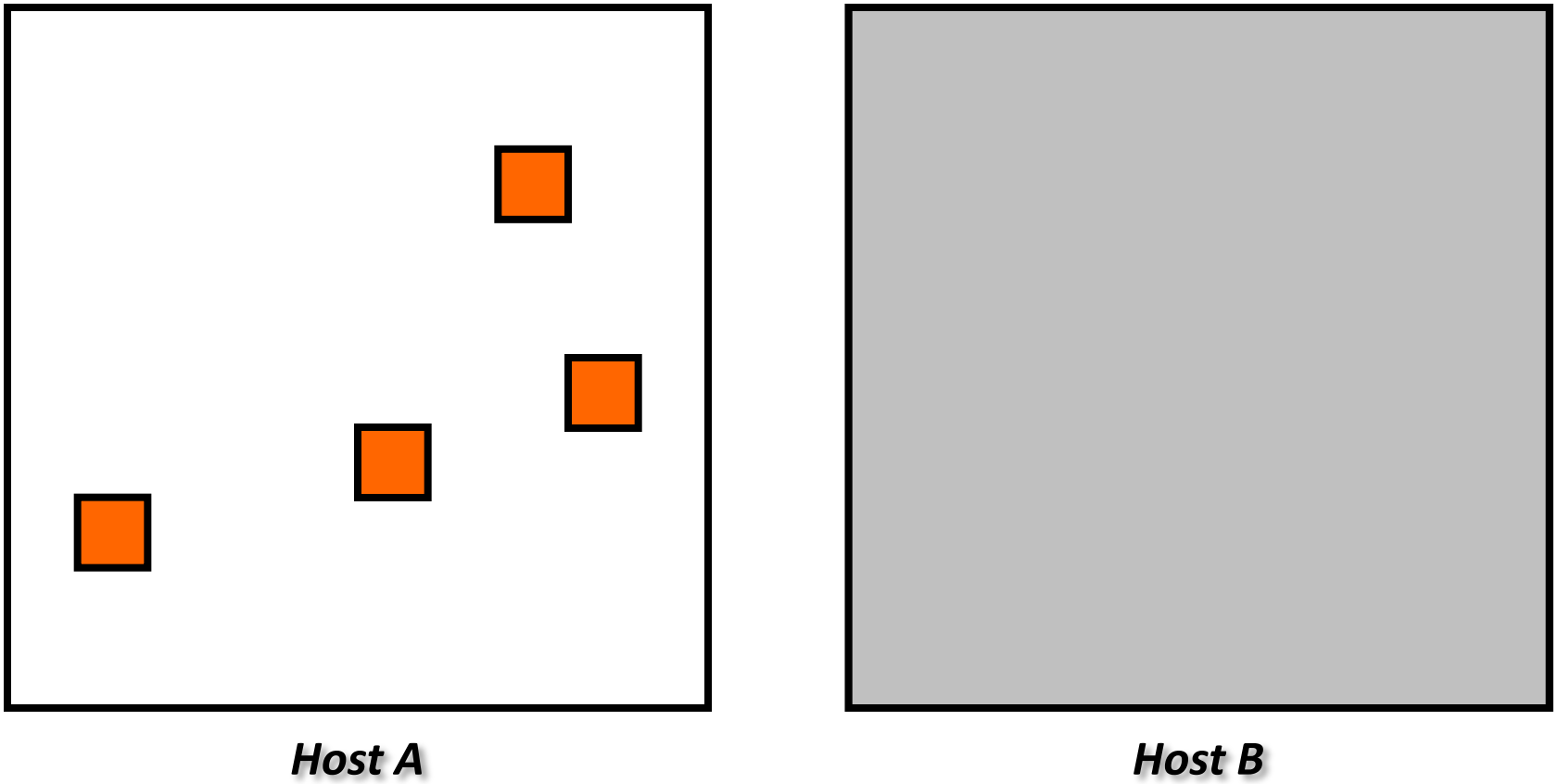
Pre-copy migration : Round 2



Live Migration Technique

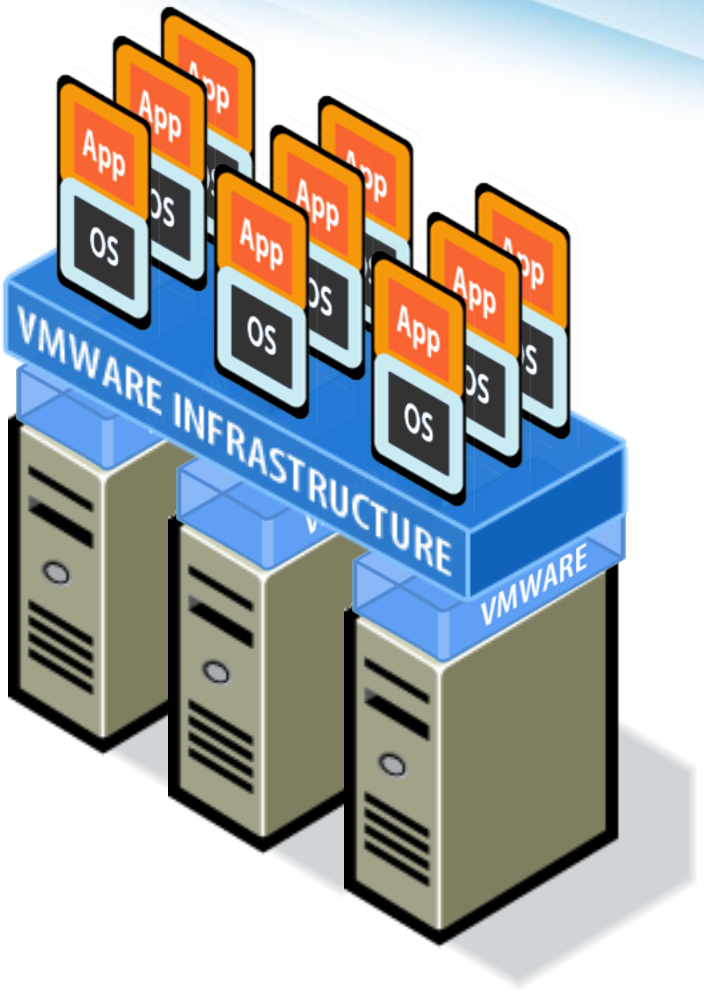
- Live migration process :

Stop and copy : Final Round



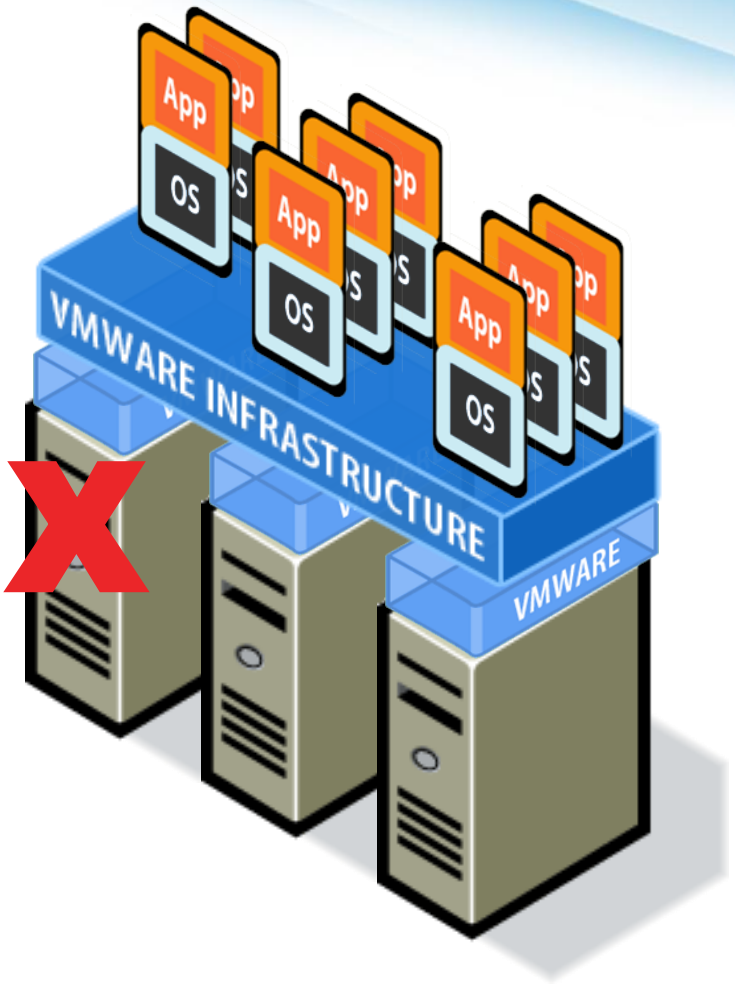
Scalability in Virtualization

- Scalability implement by VMware:
  - VMware VMotion, makes it possible to move Virtual Machines, without interrupting the applications running inside.
  - Dynamically scale up virtual machine system among physical servers.



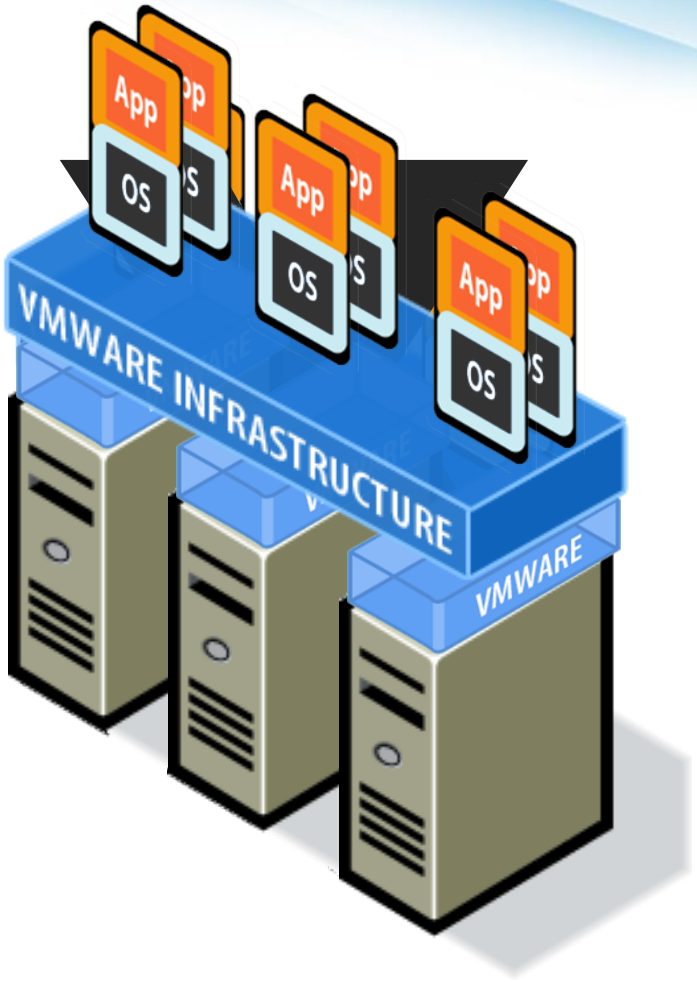
Availability in Virtualization

- Fault tolerance system :
  - VMware makes all Servers and Applications protected against component and complete system failure.
  - When system failure occurs, virtual machines will be automatically restarted on other physical servers.



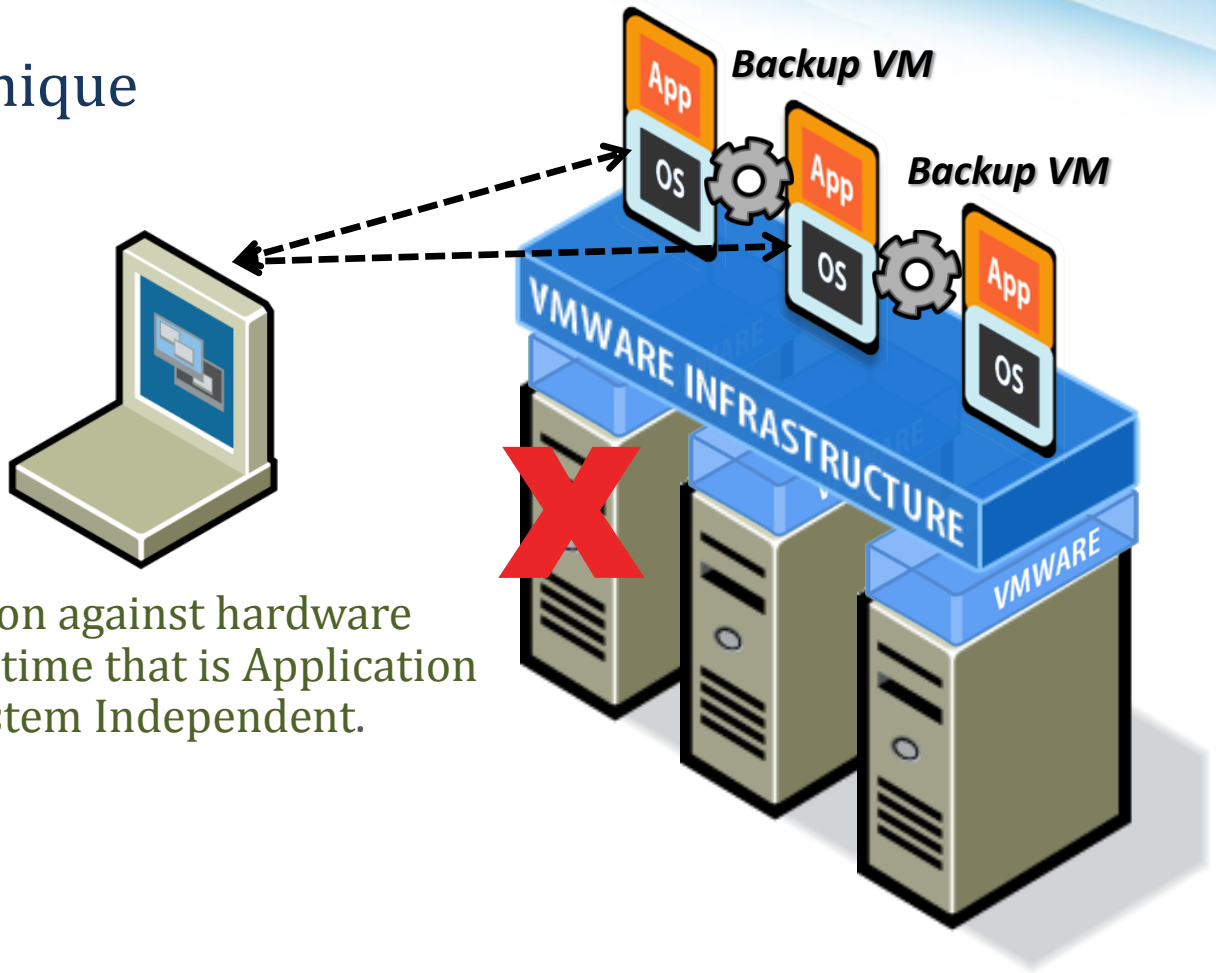
Availability in Virtualization

- Disaster recovery :
  - VMware Site Recovery Manager enables an easy transition from a production site to a Disaster Recovery site.
  - Easy Execution for real Disaster
  - Easy Testing for good night sleep



Availability in Virtualization

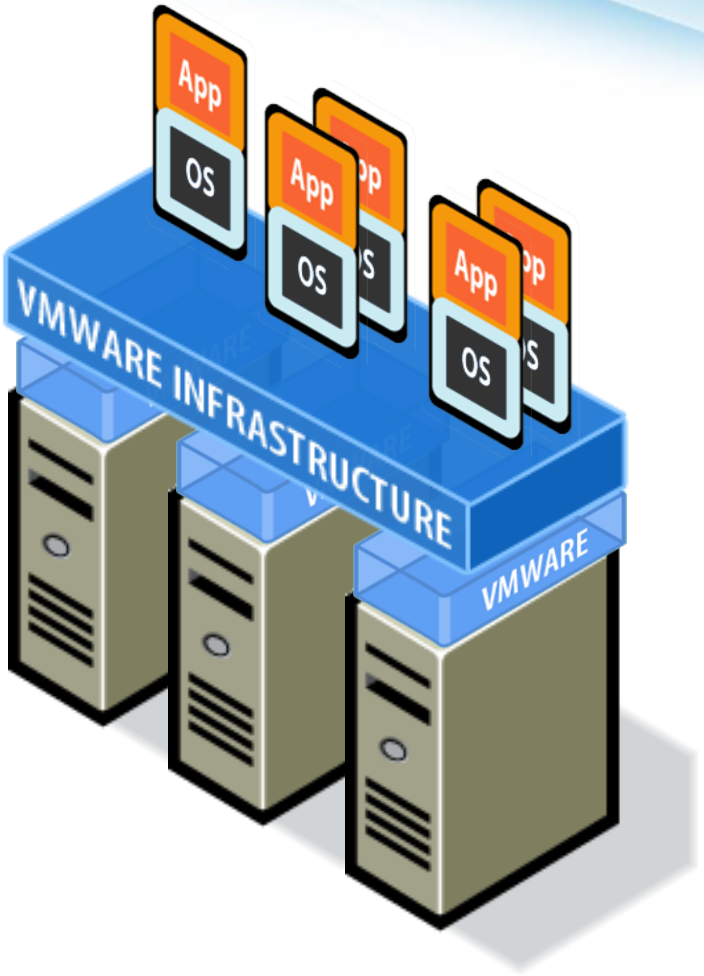
- Fail over technique



Application protection against hardware failures, with NO down time that is Application and Operating System Independent.

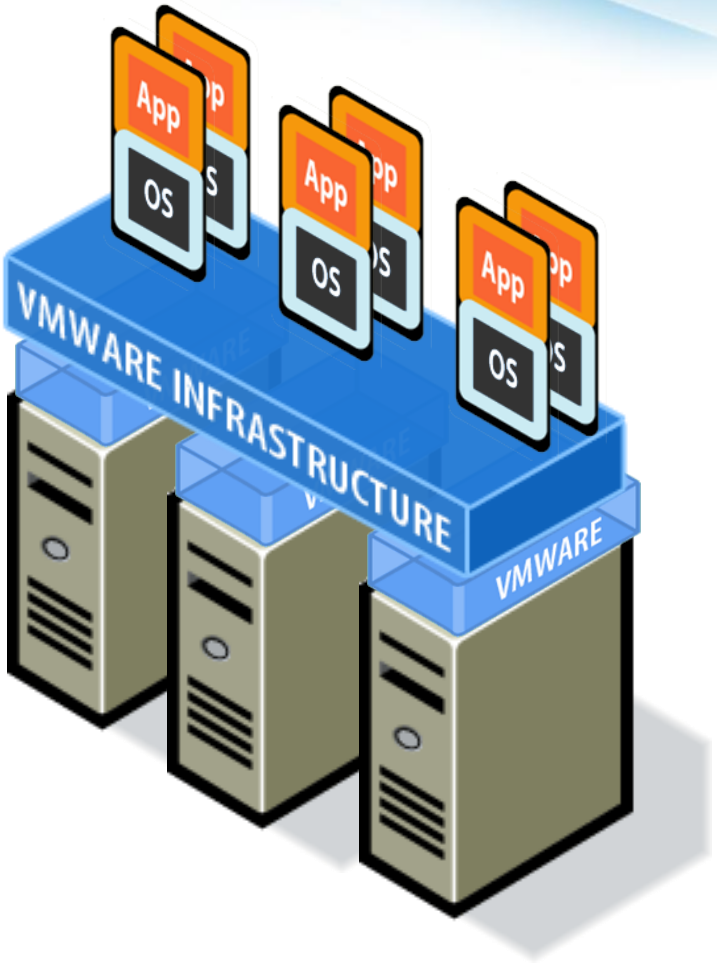
Manageability in Virtualization

- Provide physical to virtual translation :
  - Consolidation Management with the VMware Infrastructure software will automate the migrati[illegible]rom physical to vi[illegible]nes.



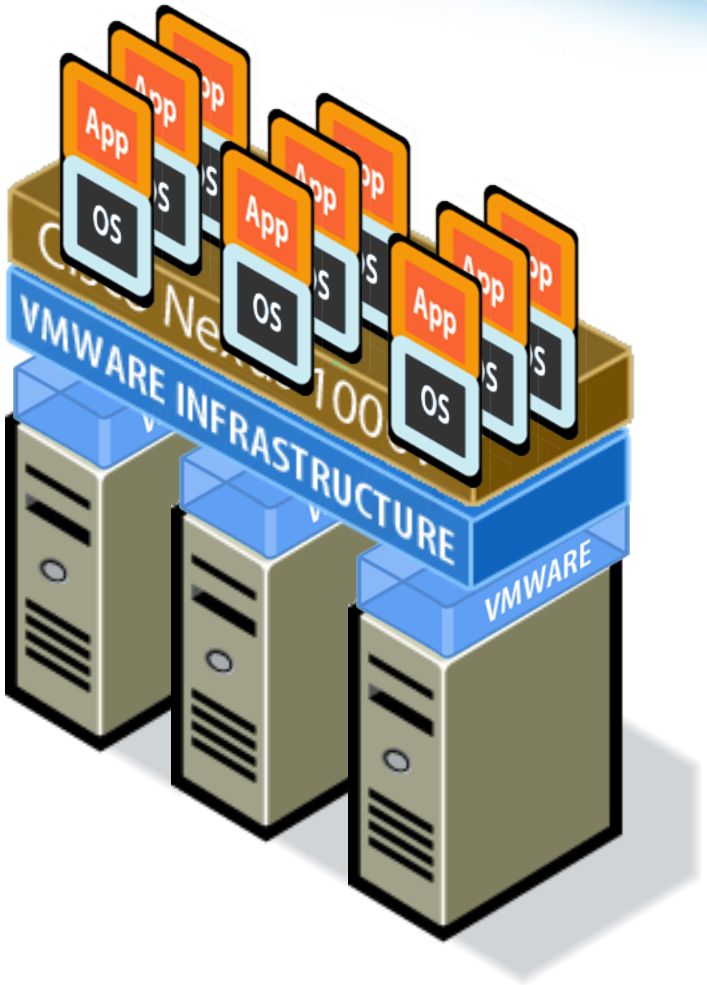
Performance in Virtualization

- Dynamic load balancing :
  - VMware Distributed Resource Scheduler automatically balances the Workloads according to set limits and guarantees.
  - Removing the need to predict resource assignment.



Performance in Virtualization

- Optimize network access :
  - VMware and Cisco are collaborating to enhance workload mobility and simpler management with virtualization-aware networks.

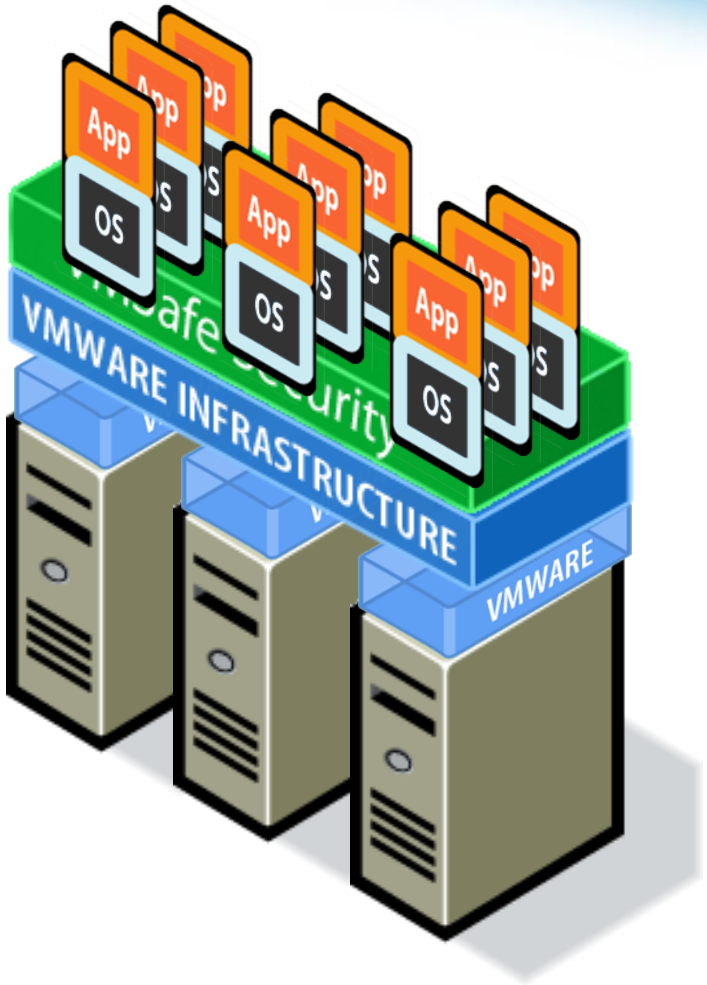


CISCO

Nexus 1000V

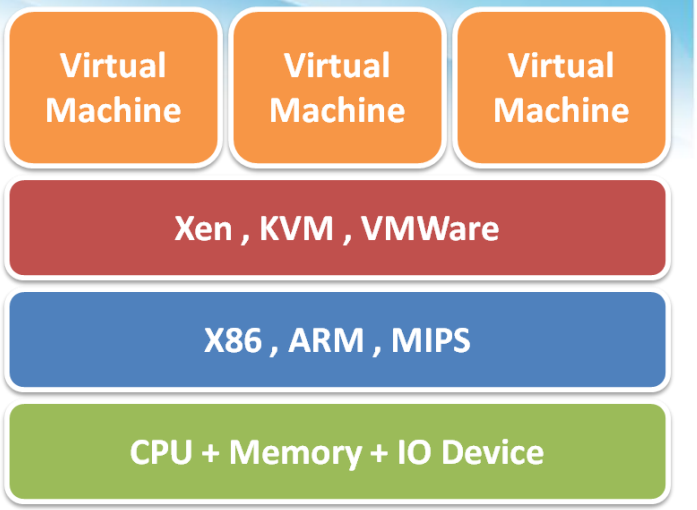
Security in Virtualization

- Enhance virtual machine security protection :
  - The Application vService VMSafe allows security vendors to add superior security solutions inside the VMware Infrastructure.



Summary

- Server virtualization technique :
  - CPU virtualization
    - Ring compression, Intel VT-x, …etc
  - Memory virtualization
    - Shadow page table, Intel EPT, …etc
  - IO virtualization
    - Device model, Intel VT-d, PCIe SR-IOV, …etc



- Ecosystem :
  - VMware implements both type-1 & type-2 virtualization
  - Xen implements both para and full virtualization
  - KVM implements in Linux mainstream kernel
- Cloud properties :
  - Enabled by live migration technique
  - Scalability, Availability, Manageability and Performance

References

- Books :
  - James E. Smith & Ravi Nair, ***Virtual Machines***, Elsevier Inc., 2005
- Web resources :
  - Xen project *http://www.xen.org*
  - KVM project *http://www.linux-kvm.org/page/Main_Page*
  - IBM VirtIO survey *https://www.ibm.com/developerworks/linux/library/l-virtio*
  - PCI-SIG IO virtualization specification *http://www.pcisig.com/specifications/iov*
- Other resources :
  - Lecture slides of “Virtual Machine” course (5200) in NCTU
  - Vmware Overview Openline presentation slides *http://www.openline.nl*
  - Xen presentation *http://www.cl.cam.ac.uk/research/srg/netos/papers/2006-xen-fosdem.ppt*