Virtualization

Concepts

Concepts

References and Sources

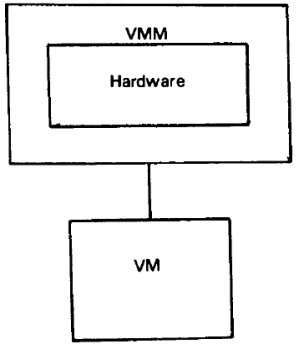
- James Smith, Ravi Nair, "The Architectures of Virtual Machines," IEEE Computer, May 2005, pp. 32-38.
- Mendel Rosenblum, Tal Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," IEEE Computer, May 2005, pp. 39-47.
- L.H. Seawright, R.A. MacKinnon, "VM/370 – a study of multiplicity and usefulness," IBM Systems Journal, vol. 18, no. 1, 1979, pp. 4-17.
- S.T. King, G.W. Dunlap, P.M. Chen, "Operating System Support for Virtual Machines," Proceedings of the 2003 USENIX Technical Conference, June 9-14, 2003, San Antonio TX, pp. 71-84.
- A. Whitaker, R.S. Cox, M. Shaw, S.D. Gribble, "Rethinking the Design of Virtual Machine Monitors," IEEE Computer, May 2005, pp. 57-62.
- G.J. Popek, and R.P. Goldberg, "Formal requirements for virtualizable third generation architectures," CACM, vol. 17 no. 7, 1974, pp. 412-421.

Definitions

- Virtualization
  - **A layer mapping its visible interface and resources onto the interface and resources of the underlying layer or system on which it is implemented**
  - **Purposes**
    - Abstraction – to simplify the use of the underlying resource (e.g., by removing details of the resource's structure)
    - Replication – to create multiple instances of the resource (e.g., to simplify management or allocation)
    - Isolation – to separate the uses which clients make of the underlying resources (e.g., to improve security)
- Virtual Machine Monitor (VMM)
  - **A virtualization system that partitions a single physical "machine" into multiple virtual machines.**
  - **Terminology**
    - Host – the machine and/or software on which the VMM is implemented
    - Guest – the OS which executes under the control of the VMM

Origins - Principles

Fig. 1. The virtual machine monitor.



"an *efficient, isolated duplicate* of the real machine"

- Efficiency
  - **Innocuous instructions should execute directly on the hardware**
- Resource control
  - **Executed programs may not affect the system resources**
- Equivalence
  - **The behavior of a program executing under the VMM should be the same as if the program were executed directly on the hardware (except possibly for timing and resource availability)**

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Communications of the ACM, vol 17, no 7, 1974, pp.412-421

Origins - Principles

Instruction types

- Privileged
  an instruction traps in unprivileged (user) mode but not in privileged (supervisor) mode.
- Sensitive
  - **Control sensitive –**
    attempts to change the memory allocation or privilege mode
  - **Behavior sensitive**
    - Location sensitive – execution behavior depends on location in memory
    - Mode sensitive – execution behavior depends on the privilege mode
- Innocuous – an instruction that is not sensitive

Theorem

For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

Signficance

The IA-32/x86 architecture is not virtualizable.

Origins - Technology

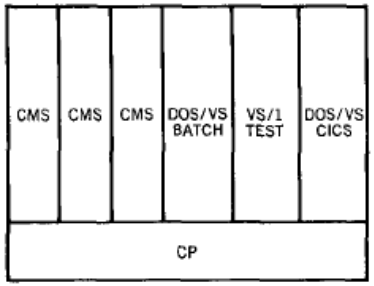
VM/370—a study of multiplicity and usefulness

by L. H. Seawright and R. A. MacKinnon

The productivity of data processing professionals and other professionals can be enhanced through the use of interactive and time-sharing systems. Similarly, system programmers can benefit from the use of system testing tools. A systems solution to both areas can be the virtual machine concept, which provides multiple software replicas of real computing systems on one real processor. Each virtual machine has a full complement of input/output devices and provides functions similar to those of a real machine. One system that implements virtual machines is IBM's Virtual Machine Facility/370 (VM/370).[1]

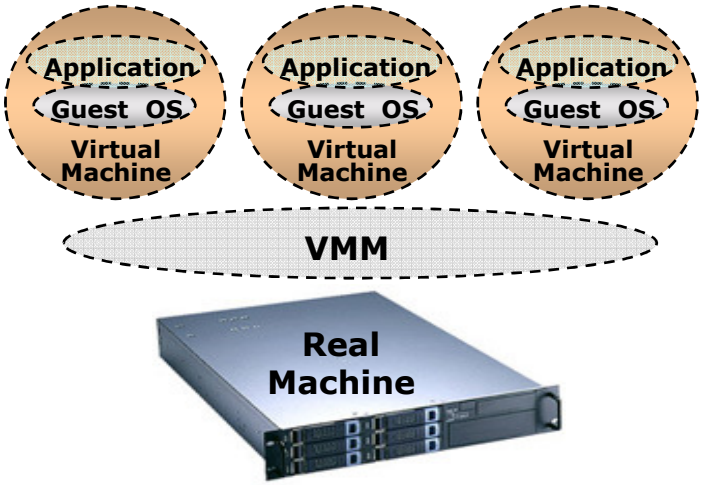
IBM Systems Journal, vol. 18, no. 1, 1979, pp. 4-17.

Figure 1 A VM/370 environment

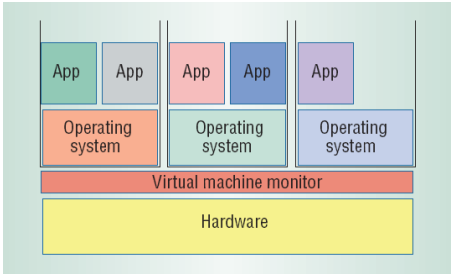


- Concurrent execution of multiple production operating systems
- Testing and development of experimental systems
- Adoption of new systems with continued use of legacy systems
- Ability to accommodate applications requiring special-purpose OS
- Introduced notions of "handshake" and "virtual-equals-real mode" to allow sharing of resource control information with CP
- Leveraged ability to co-design hardware, VMM, and guestOS

VMMs Rediscovered

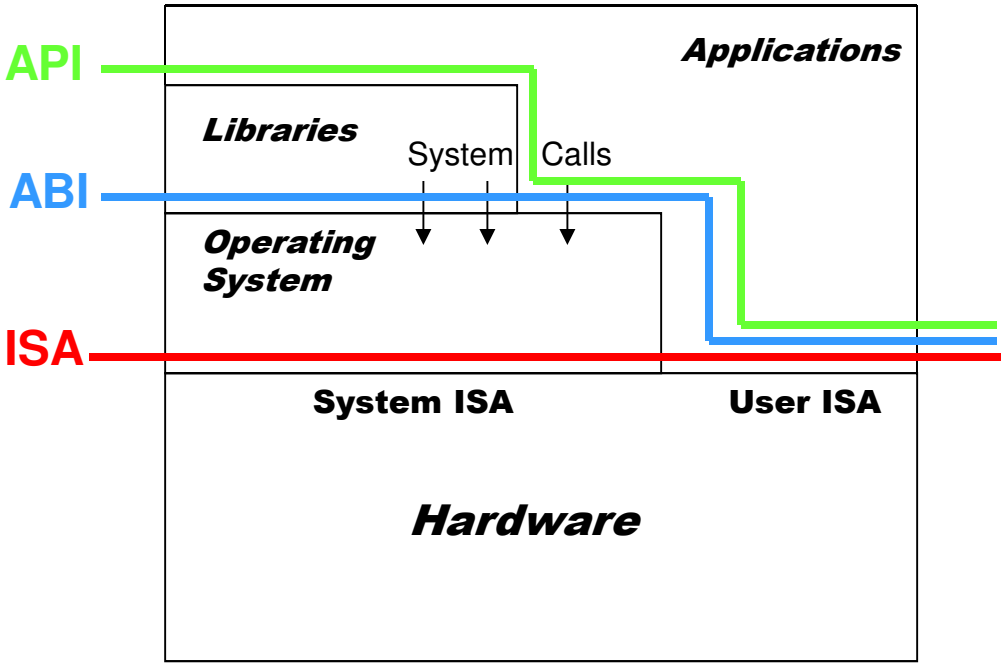


- Server/workload consolidation (reduces "server sprawl")
- Compatible with evolving multi-core architectures
- Simplifies software distributions for complex environments
- "Whole system" (workload) migration
- Improved data-center management and efficiency
- Additional services (workload isolation) added "underneath" the OS
  - **security (intrusion detection, sandboxing,...)**
  - **fault-tolerance (checkpointing, roll-back/recovery)**



Architecture & Interfaces

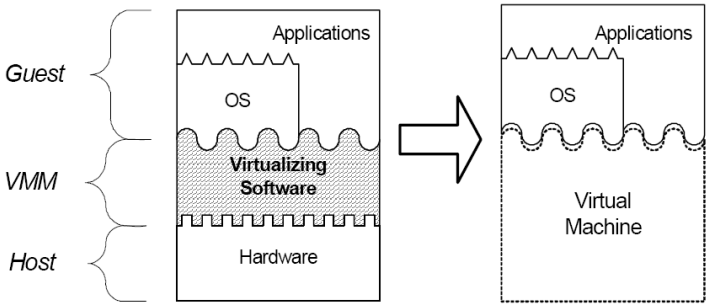
- Architecture: formal specification of a system's interface and the logical behavior of its visible resources.



- API – application binary interface
- ABI – application binary interface
- ISA – instruction set architecture

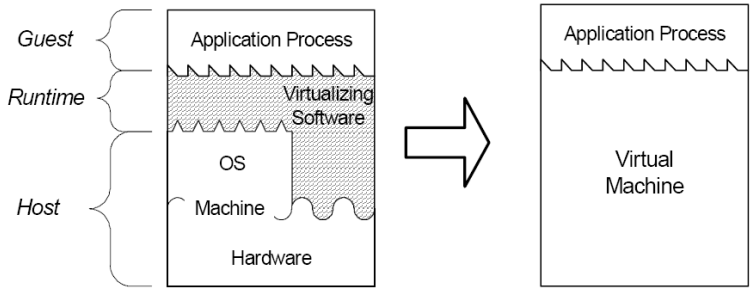
VMM Types

System



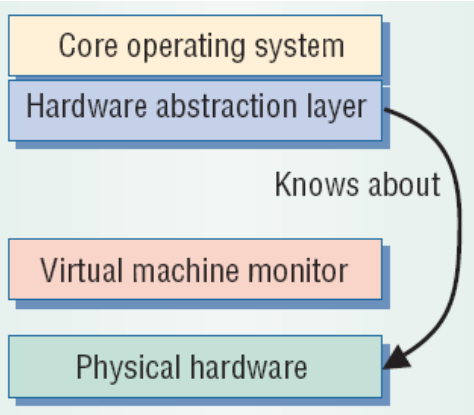
- Provides ABI interface
- Efficient execution
- Can add OS-independent services (e.g., migration, intrustion detection)

Process

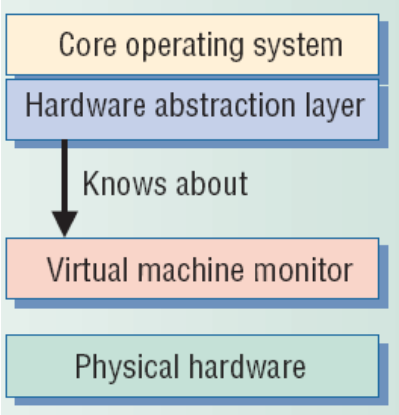


- Provdes API interface
- Easier installation
- Leverage OS services (e.g., device drivers)
- Execution overhead (possibly mitigated by just-in-time compilation)

System-level Design Approaches

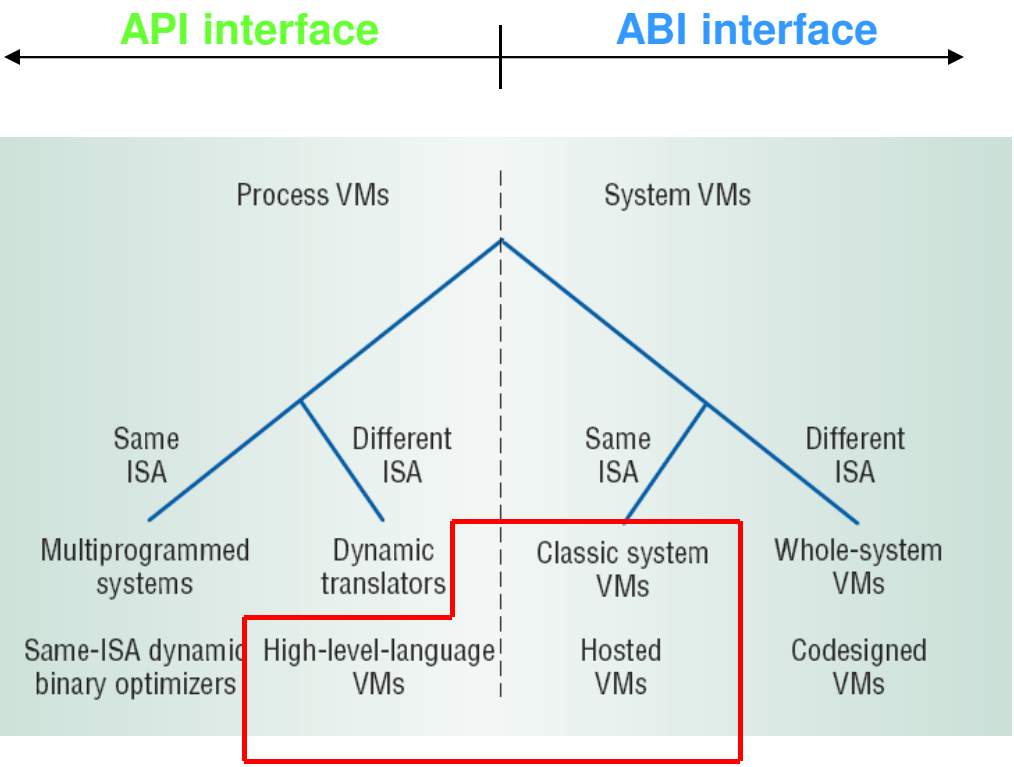


- Full virtualization (direct execution)
  - **Exact hardware exposed to OS**
  - **Efficient execution**
  - **OS runs unchanged**
  - **Requires a "virtualizable" architecture**
  - **Example: VMWare**



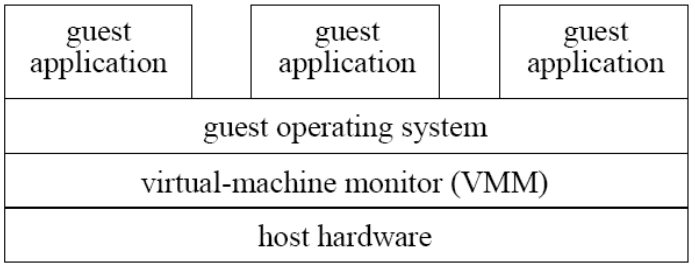
- Paravirtualization
  - **OS modified to execute under VMM**
  - **Requires porting OS code**
  - **Execution overhead**
  - **Necessary for some (popular) architectures (e.g., x86)**
  - **Examples: Xen, Denali**

Design Space (level vs. ISA)

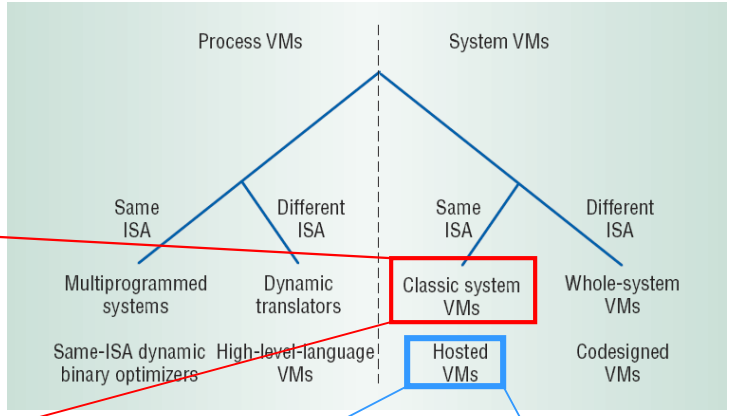


- Variety of techniques and approaches available
- Critical technology space highlighted

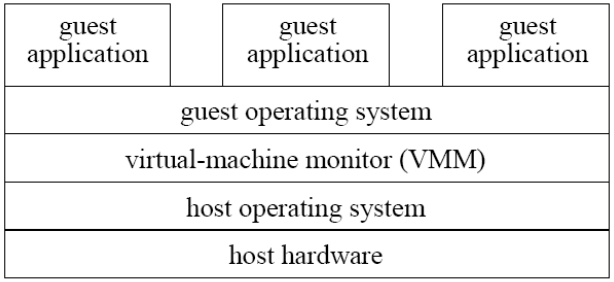
System VMMs



Type 1



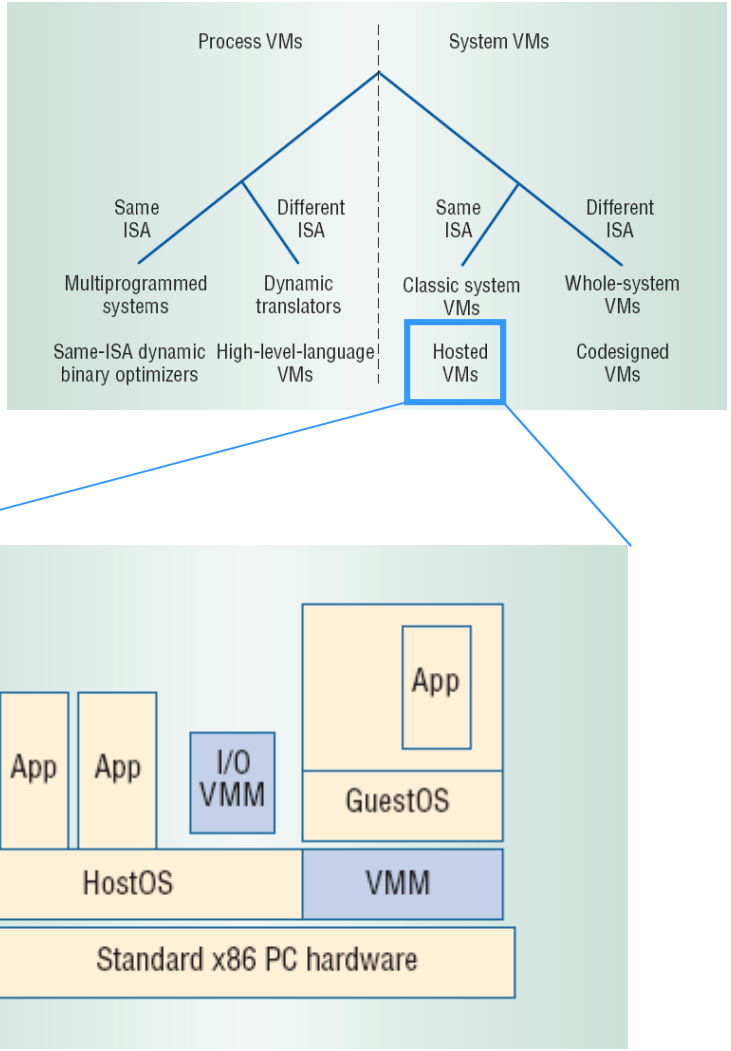
- Structure
  - **Type 1: runs directly on host hardware**
  - **Type 2: runs on HostOS**
- Primary goals
  - **Type 1: High performance**
  - **Type 2: Ease of construction/installation/acceptability**
- Examples
  - **Type 1: VMWare ESX Server, Xen, OS/370**
  - **Type 2: User-mode Linux**



Type 2

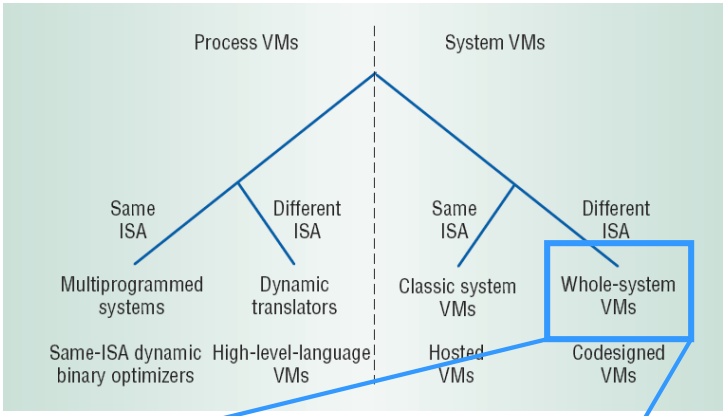
Hosted VMMs

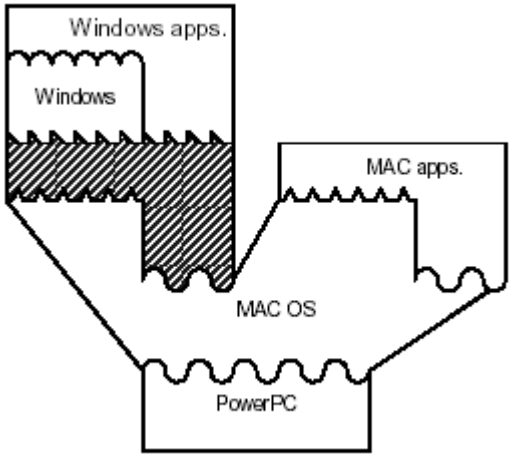
- Structure
  - **Hybrid between Type1 and Type2**
  - **Core VMM executes directly on hardware**
  - **I/O services provided by code running on HostOS**
- Goals
  - **Improve performance overall**
  - **leverages I/O device support on the HostOS**
- Disadvantages
  - **Incurs overhead on I/O operations**
  - **Lacks performance isolation and performance guarantees**
- Example: VMWare (Workstation)



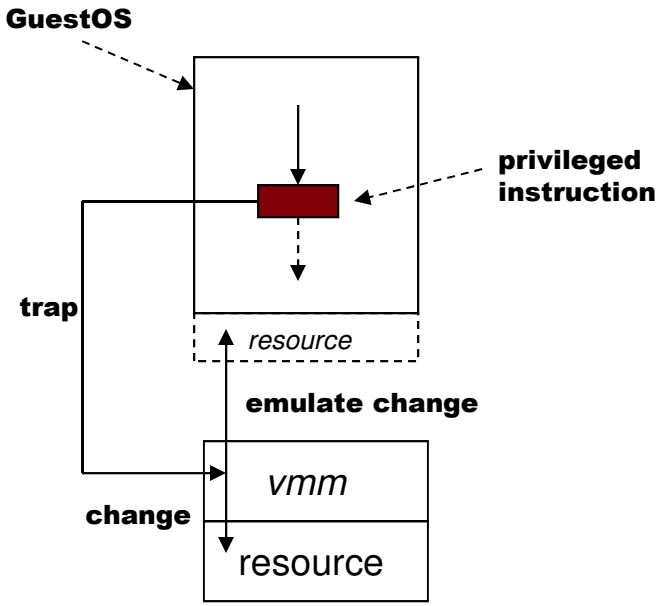
Whole-system VMMs

- **Challenge: GuestOS ISA differs from HostOS ISA**
- **Requires full emulation of GuestOS and its applications**
- **Example: VirtualPC**





Strategies

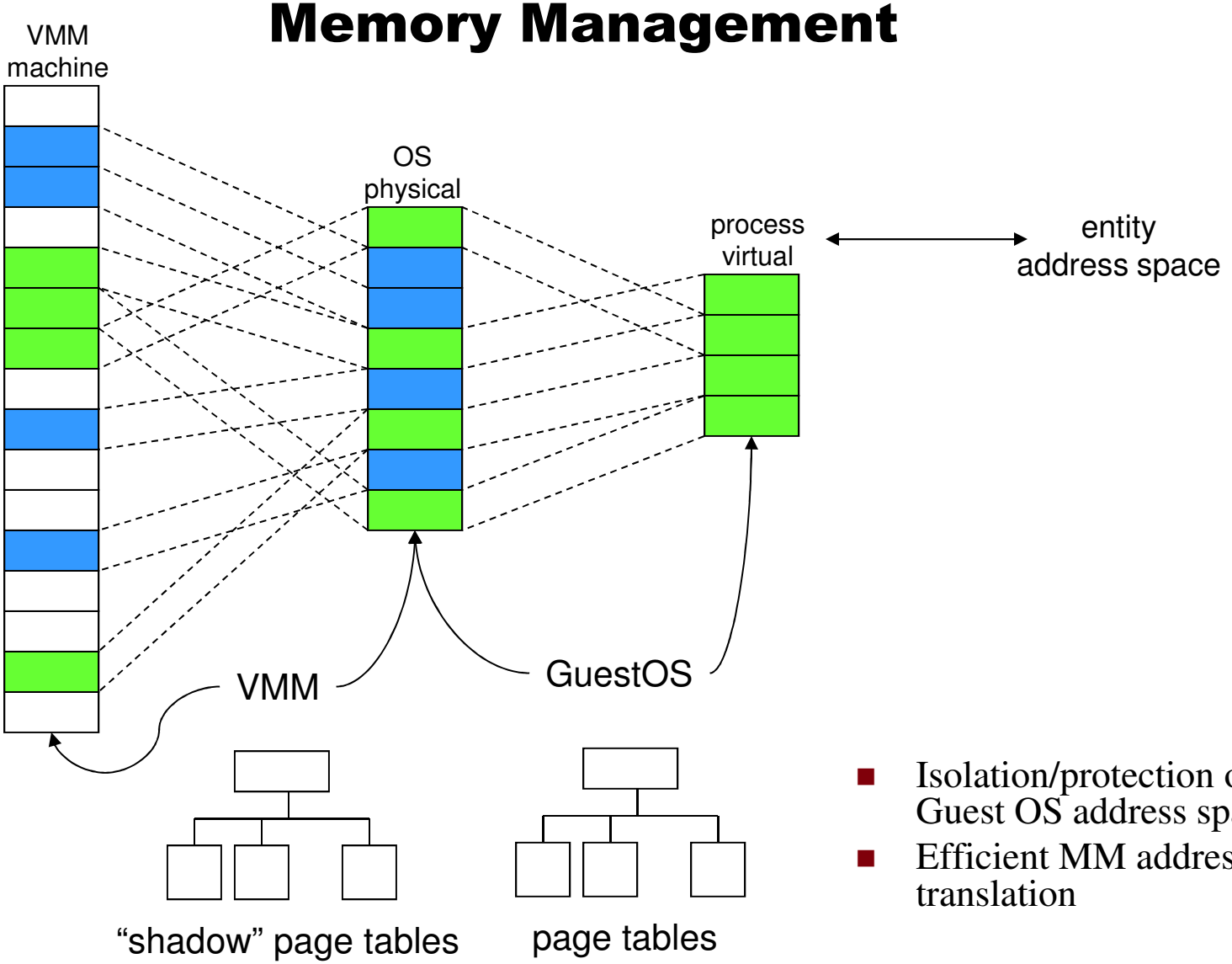


- De-privileging
  - VMM emulates the effect on system/hardware resources of privileged instructions whose execution traps into the VMM
  - aka trap-and-emulate
  - Typically achieved by running GuestOS at a lower hardware priority level than the VMM
  - Problematic on some architectures where privileged instructions do not trap when executed at deprivileged priority
- Primary/shadow structures
  - VMM maintains "shadow" copies of critical structures whose "primary" versions are manipulated by the GuestOS
  - e.g., page tables
  - Primary copies needed to insure correct environment visible to GuestOS
- Memory traces
  - Controlling access to memory so that the shadow and primary structure remain coherent
  - Common strategy: write-protect primary copies so that update operations cause page faults which can be caught, interpreted, and emulated.

Virtualizing the IA-32 (x86) architecture

- Architecture has protection rings 0..3 with OS normally in ring 0 and applications in ring 3…
- …and VMM must run in ring 0 to maintain its integrity and control
- …but GuestOS not running in ring 0 is problematic:
  - **Some privileged instructions execute only in ring 0 but do not fault when executed outside ring 0 (remember privileged vs. sensitive?)**
  - **instructions for low latency system calls (SYSENTER/SYSEXIT) always transition to ring 0 forcing the VMM into unwanted emulation or overhead**
  - **For the Itanium architecture, interrupt registers only accessible in ring 0; forcing VMM to intercept each device driver access to these registers has severe performance consequences**
  - **Masking interrupts can only be done in ring 0**
  - **Ring compression: paging does not distinguish privilege levels 0-2, GuestOS must run in ring 3 but is then not protected from its applications also running in ring 3**
  - **Cannot be used for 64-bit guests on IA-32**
  - **The fact that it is not running in ring 0 can be detected (is this important?)**

Memory Management



- Isolation/protection of Guest OS address spaces
- Efficient MM address translation