Introduction to Virtualization

Cloud computing, which refers to both the applications delivered as services over the Internet and the hardware along with systems software in the data centers that provide those services [1], has taken center stage in information technology in recent years. As a key component of cloud computing, virtualization technology has gained tremendous attention. With virtualization technology, the underlying hardware resources can be shared by multiple virtual machines or domains with each running its own operating system (OS). This gives rise to higher hardware utilization and lower power consumption. The key component in system-level virtualization is the Virtual Machine Monitor (VMM), also referred to as a hypervisor. A VMM is responsible for isolating each running instance of an operating system from the physical machine. The VMM translates or emulates special instructions of a guest OS. A VMM itself is a complex piece of software, which typically runs with the highest privilege level (higher than a guest OS), so it is vital to ensure a VMM is as bug-free as possible.

1. Types of VMM

Goldberg [2] classifies VMMs into two types:

**Type 1- Bare metal VMM (aka native VMM)**

Bare metal VMM refers to the type of VMM that runs directly on the underlying hardware. As shown in Figure 1, a guest OS runs on a higher level (lower priority) above the VMM. The management of type 1 VMM, such as the creation of a virtual machine or domain, is implemented by a management console, which is an additional piece of software. The management console usually operates in command mode. Xen is a classic example of type 1 VMM.

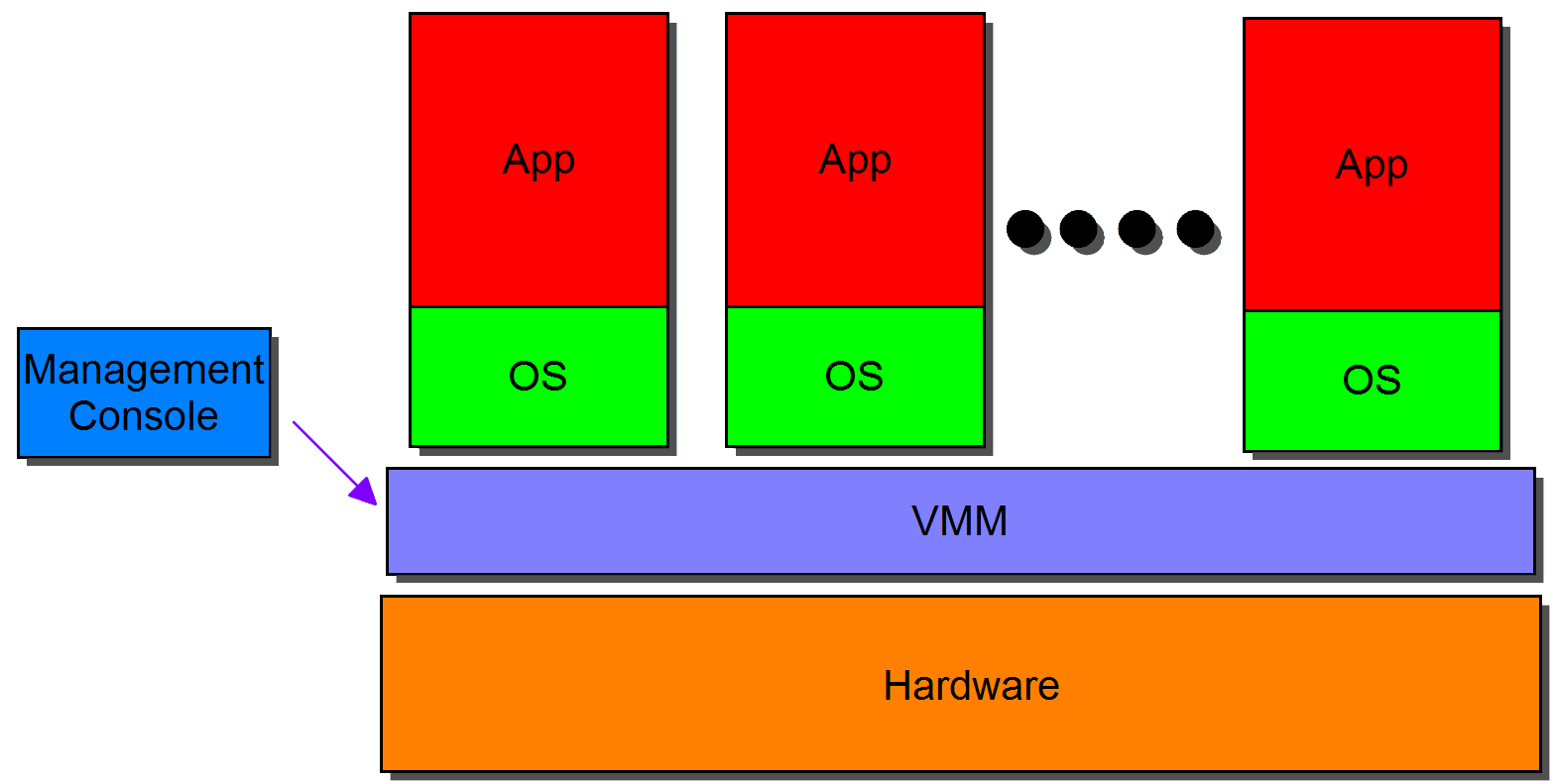


Figure 1. A bare metal VMM

**Type 2- Hosted VMM ((aka hosted VMM))**

The type 2 VMM is a hosted VMM, because it needs to reside in an existing OS environment (known as a hosted OS). As shown in Figure 2, a hosted OS is directly installed on the underlying hardware. A type 2 VMM is installed in the hosted OS as an application, and instances of OSes are installed on the type 2 VMM. The hosted OS typically provides a convenient procedure for users to manage a type 2 VMM via a graphical user interface (GUI). A type 2 VMM usually has lower performance than a type 1 VMM. The Kernel-based Virtual Machine (KVM) is an example of a type 2 VMM. Under KVM, virtual machines are created by opening a device node (/dev/kvm). Creating and running virtual machines is achieved through ioctl() system calls. Currently, KVM only supports full virtualization. All I/O accesses are forwarded to the user space of the hosted OS where the Quick EMUlator (QEMU) is used to emulate their behavior.

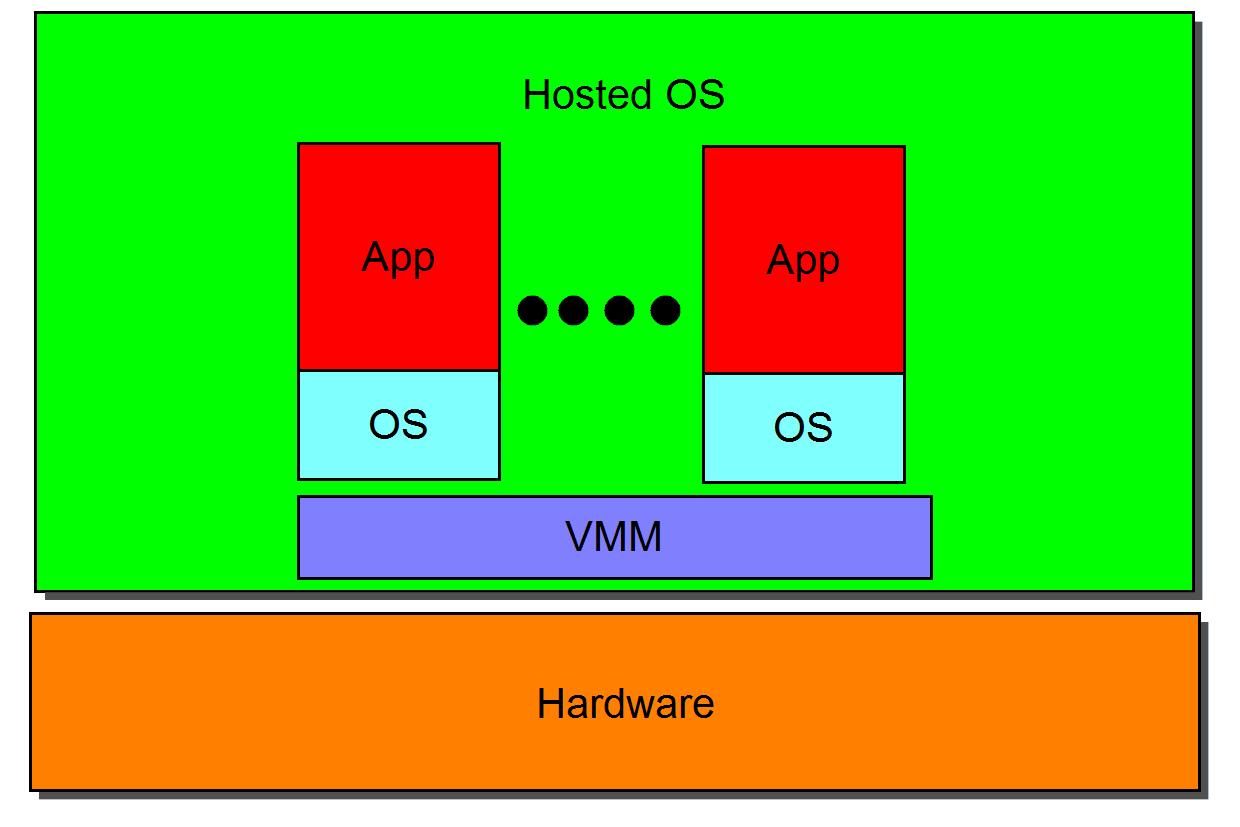


Figure 2. A hosted VMM

2. x86 Virtualization

Popek and Goldberg classify instructions into 3 categories [3]:

1. **Privileged instructions**

Those that trap if the processor is in user mode and do not trap if it is in system mode, such as instructions which are intended to change the value of a control register.

1. **Control sensitive instructions**

Those that attempt to change the configuration of resources in the system, such as CLI, which is intended to clear the interrupt flag (IF).

1. **Behavior sensitive instructions**

Those whose behavior or result depends on the configuration of resources, such as the instruction, INT N, which calls the Nth interrupt handler from the interrupt descriptor table (IDT).

The requirement for a system to be virtualizable is that sensitive instructions should be a subset of privileged instructions. However, 17 sensitive instructions of x86 do not satisfy this requirement; SIDT for example, which is used to store the content of the interrupt descriptor table register. Several types of virtualization have been proposed to solve the problem. The following are the three most popular:

1. **Full virtualization**

Full virtualization enables an unmodified OS to run in a virtual machine that simulates the entire hardware environment for a running OS. The traditional idea of full virtualization is to trap privileged instructions that are executed in the unprivileged mode in a guest OS, and emulate the behaviours of these privileged instructions in the VMM or the hosted OS. A disadvantage of full virtualization is poor performance caused by the trap-and-emulate overhead. Another example of full virtualization is the VMware binary rewriting approach. This VMM scans the instruction stream, and translate all privileged instructions to their emulated versions at runtime.

1. **Paravirtualization**

Paravirtualization requires a guest OS to be modified to run in a virtual machine. The idea is to modify the guest OS to run in ring 1, so that the VMM can run exclusively in ring 0. The guest OS is aware that it is in a paravirtualized environment. A guest OS cannot execute any privileged instructions, such as updating page tables, because it runs in lower priority and its privileged operations are ignored by the VMM. The VMM typically offers a hypercall mechanism for a guest OS to request privileged operations. A disadvantage of paravirtualization is that it is not feasible to virtualize a legacy OS, such as Windows. However, it is still a popular virtualization solution, due to its high efficiency for performing I/O [4].

1. **Hardware-assisted virtualization**

Hardware-assisted virtualization (HVM) is a special kind of full virtualization. HVM requires support of the underlying CPU for virtualization. Both Intel and AMD have introduced HVM support in processors released after 2007. The fundamental idea of HVM is to add an extra executing mode (which can be thought of as “ring -1”), such as VMX for Intel processors, which has higher priority than ring 0. An unmodified guest OS continues to run in ring 0 (non-root mode), while the VMM stays in “ring -1” (ring 0 root mode). The VMM traps and emulates privileged instructions for guest OSes. Compared to paravirtualization, HVM performs better with CPU intensive workloads, and not as effectively with I/O intensive workloads [5].

3. Introduction to the Xen VMM

Xen, a paravirtualizing open-source VMM, was first released as a paravirtualization solution in 2003, mainly for x86 platforms [6]. Xen has supported HVM since Intel and AMD processors began supporting virtualization. HVM allows unmodified OSes to run over a VMM, but it results in low I/O performance.

### 3.1. Priority Levels in the Xen VMM

On x86 platforms, ring 0 is set as the highest priority, and ring 3 as the lowest priority. Ring 0 is designed for an OS to stay, and applications usually run in ring 3. In a paravirtualized environment each running OS needs to be modified to run in ring 1, while the Xen VMM runs exclusively in Ring 0, guarding accesses to all privileged operations and hardware resources. In this case, an OS is not permitted to implement privileged operations they could implement before, such as updating page tables. Instead, the Xen VMM offers a hypercall mechanism, which is the only method for these running OSes to interact with the Xen VMM to request privileged operations.

### 3.2. Domains

Xen refers to each running virtual machine as a domain. Xen supports one privileged domain, domain 0 (Dom0), and multiple unprivileged domains (DomUs). As the Xen VMM has the highest priority, and is responsible for all the privileged operations in the entire paravirtualized system, the Xen VMM needs to be as bug-free as possible. Therefore, the Xen VMM does not include any device drivers. Dom0 is usually delegated as the driver domain, which includes all the drivers for the underlying hardware. This is why Dom0 is a special domain, with higher privilege than other domains.

The unprivileged domains are not given direct access to the underlying hardware; instead they require the use a split device driver model. As shown in Figure 3, Dom0 has backend drivers installed, and each DomU has frontend drivers. In order to access the underlying hardware, a virtual frontend driver in a DomU communicates with the related virtual backend driver in Dom0 (driver domain), which is usually achieved by using shared memory. The latter then forwards the received I/O requests to the corresponding real device driver. Dom0 in a split device driver model typically consists of a backend driver, a device driver and a multiplexer that handles multiplexing multiple requests from DomUs to access the shared hardware.

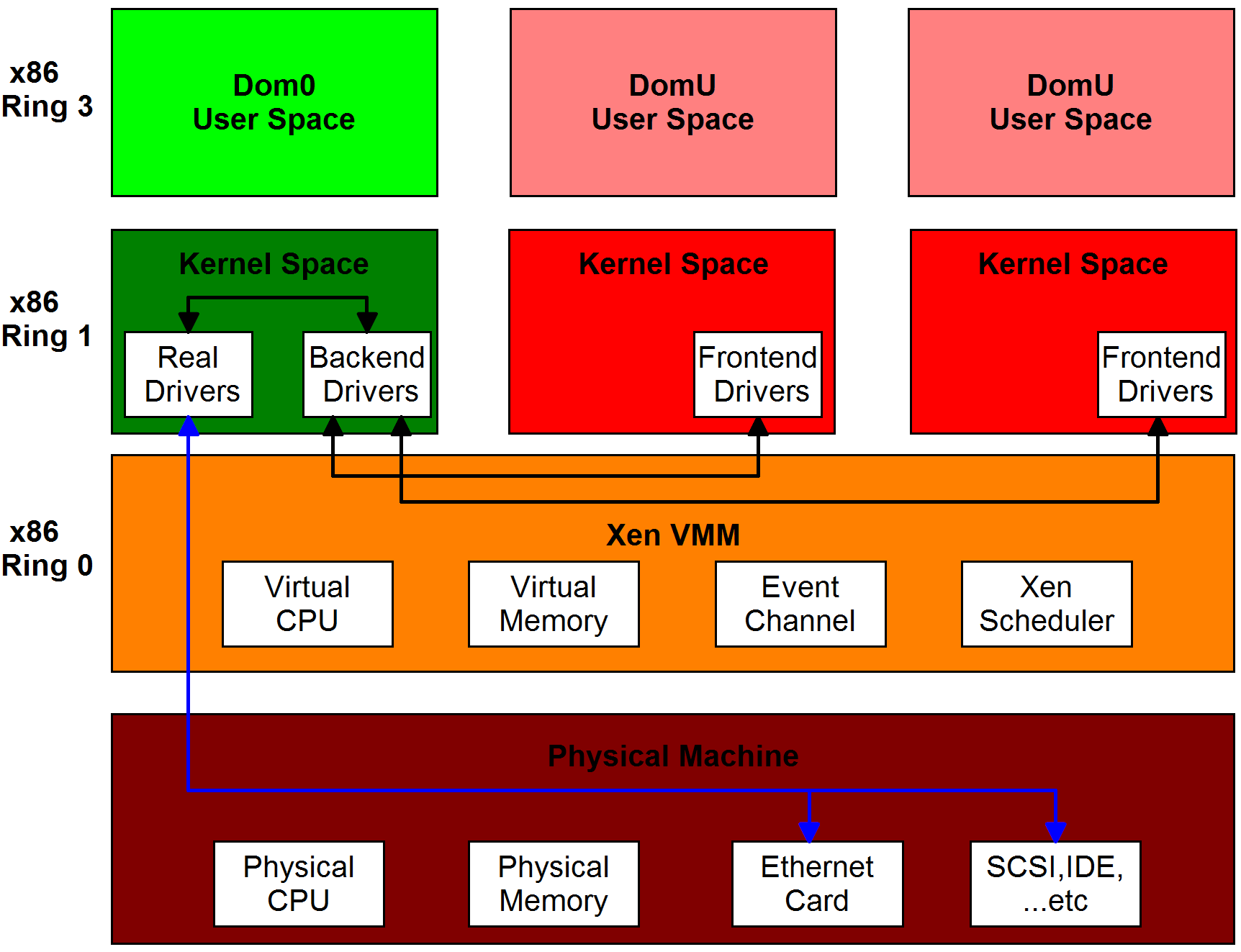


Figure 3. An overview of Xen-based paravirtualized environment

The Xen VMM recently added direct I/O access support for a domain, also known as device passthrough (e.g. PCI passthrough). The passthrough method can give a particular domain near-native I/O performance, but it violates the concept of sharing in virtualization. It also causes security problems on systems without an input/output memory management unit (IOMMU) [7].

### 3.3. Memory Management

The memory used in modern operating systems has already been virtualized, and each process has its own address space. From a process prospective, it assumes that it is the only process running on the machine, and that it has access to the entire memory space. The Xen VMM provides a pseudo-physical memory model to realize the isolation between different domains.

Figure 4 depicts a pseudo-physical memory model of the Xen VMM. The four colored blocks on top represent four virtual addresses used in a domain. The virtual addresses are first translated into pseudo-physical addresses, and the pseudo-physical addresses are then translated into real physical addresses. A guest OS allocates and maintains its own page tables, but the page tables are marked with read-only. Updating the page tables requires the OS to use an explicit hypercall. The Xen VMM validates all the updates that it deems safe [8]; for example, an update request to map a machine page belonging to another domain will fail. The Xen VMM maintains a globally readable machine-to-physical table, which records the mapping from machine (real physical) page frames to pseudo-physical ones [9]. Conversely, a guest OS is supplied with a read-only (pseudo)physical-to-machine table, which is mapped into its address space through the shared info page. The physical-to-machine table implements the translation of a pseudo-physical address to a machine address. This table is referenced when a guest OS uses a hypercall to update its page table.

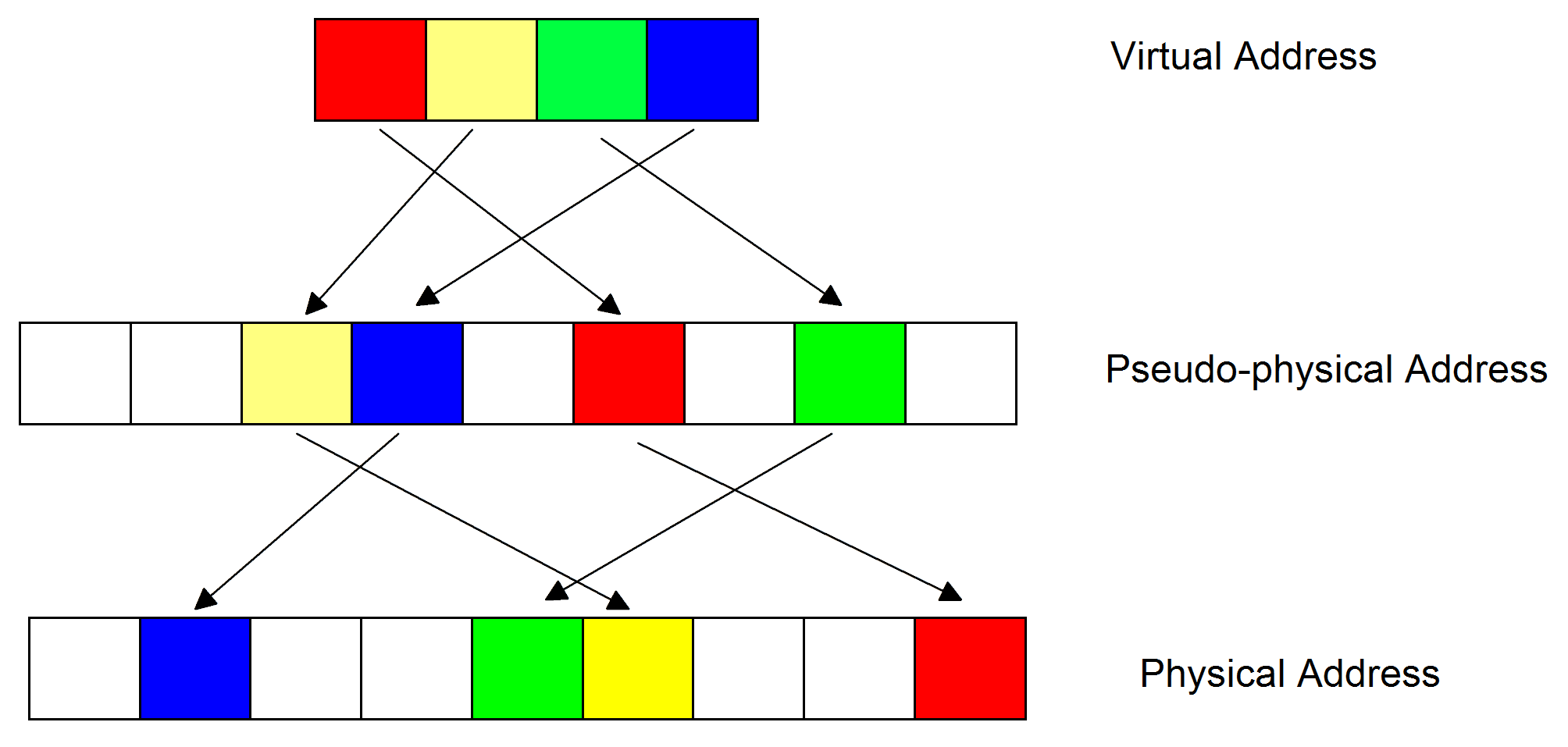


Figure 4. A pseudo-physical memory model

### 3.4. Mechanisms

The Xen VMM provides numerous mechanisms that allow domains to run correctly in the virtualized environment; the aforementioned split device driver model is an example of such a mechanism. The following are some other commonly used mechanisms supplied by the Xen VMM:

1. *Event Channel Mechanism*

An event can be understood conceptually as a virtual interrupt. An interrupt is an asynchronous notification delivered to the machine hardware, while an event is an asynchronous notification delivered to a virtual machine (domain) by the Xen VMM or another domain. A domain needs to request an event channel from the Xen VMM to deliver events to another domain. After obtaining an event channel, both of the two domains need to bind the event channel to an interrupt number, and register an interrupt handler with that interrupt number so that the corresponding interrupt handler will be invoked when an event is received. An example use of events is a split device model in which the frontend and backend drivers notify each other when an I/O request or response has been sent.

1. *Grant Table Mechanism*

The grant table mechanism allows a domain to share or transfer its memory pages, with or to another domain. Each domain has its own grant table, which is shared with the Xen VMM. The table is a data structure storing the information used for memory sharing between domains, such as which operation a grantee domain is allowed to perform on the granter’s offered memory. Entry of such a data structure in a grant table is identified by a grant reference which is an integer index. When one domain wants to share its memory with another domain, a grant reference corresponding to the data structure describing that shared memory must be passed to the grantee by the granter via some out of band mechanism, such as XenStore.

1. *XenStore*

The XenStore is a hierarchical storage system maintained by Dom0, and shared among all DomUs. It is mainly used as an extended method of transmitting small amounts of information between domains [4]. For example, the aforementioned grant reference can be put in the XenStore by one domain, and obtained by another domain.

### 3.5. Schedulers

The default and most widely used scheduler in the Xen VMM is the credit scheduler, which focuses on allocating CPU resources in a rational manner to each domain according to their pre-assigned weight. The scheduler adopts vCPUs (virtual CPU) as scheduling entities, and each domain can be assigned one or more vCPUs. The default scheduling time quantum is 30ms; that is, scheduling decisions are made every 30ms. The scheduler debits the credits of each running domain on a tick period (10ms) basis. Domains that have consumed all of their allocated credits will be put into the OVER FIFO (first-in-first-out) queue at the following scheduling point, which means they are over scheduled. While they can remain in the UNDER FIFO queue but be moved to the end of the queue if they still have credits remaining. Once the sum of all the active domains becomes negative, the scheduler will allocate new credits to all the domains at the next scheduling point according to their weight. Domains in the OVER queue will not be selected to run unless there are no domains in the UNDER queue ready to run [10]. Therefore, some domains can use more than their share of processor resources but only on the condition that the processor would otherwise have been idle.

However, domains that only have I/O-bound tasks are usually blocked when they are waiting for I/O responses. When an event is sent to a blocked domain, the scheduler will activate the domain and put it into the UNDER queue. Hence, the de facto I/O response latency is largely determined by the activated domain’s position after it is inserted in the queue. Accordingly, a BOOST state, which has higher priority than OVER and UNDER states, is introduced by the credit scheduler to reduce the latency caused by scheduling delay. With the BOOST mechanism, each time a blocked domain is activated for a pending event it will be assigned a BOOST priority, and allowed to preempt the running domain. For impartiality, the priority of an activated domain will not be boosted if it is in the OVER queue, which implies that it had both I/O and CPU bound tasks, and the previously allocated sharing resources have been totally consumed. Consequently, domains running only I/O bound tasks can realize lower I/O responsiveness, due to the BOOST mechanism [10].

References

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds: A Berkeley view of cloud computing,” University of California, Berkeley, Tech. Rep. UCB-EECS, February 2009.
2. R. P. Goldberg, “Architectural principles for virtual computer systems,” Ph.D. Thesis. Division of Engineering and Applied Physics, Harvard University Cambridge Massachusetts, 1973.
3. G. J. Popek,R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” Communications of the ACM, vol. 17, no. 7, pp. 412-421, July 1974.
4. D. Chisnall, “The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series),” Prentice Hall PTR, Upper Saddle River, USA, 2007.
5. J. Nakajima, Q. Lin, S. Yang, M. Zhu, S. Gao, P. Yu, Y. Dong, Z. Qi, K. Chen, H. Guan, M. Xia, “Optimizing virtual machines using hybrid virtualization,” Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 573-578, 2011.
6. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP’03), pp.164-177, 2003.
7. Xen PCI Passthrough, <http://wiki.xen.org/wiki/Xen_PCI_Passthrough>.
8. D. E. Williams, J. Garcia, “Virtualization with Xen including XenEnterprise, XenServer and XenExpress,” Syngress, May 2007.
9. S. Senthilvelan and M. Senthilvelan,“Study of Content-Based Sharing on the Xen Virtual Machine Monitor,” <http://pages.cs.wisc.edu/~remzi/Classes/736/Spring2005/Projects/Muru-Selva/>.
10. D. Ongaro, A. L. Cox, and S. Rixner, “Scheduling I/O in virtual machine monitors,” In Proc. of Virtual Execution Environments, 2008.