RC 24500 (W0802-069), 14 February 2008, Revision 1, 21 July 2008
Computer Science

IBM Research Report

Security and Performance Trade-Offs in I/O Operations for Virtual Machine Monitors

Paul A. Karger and David R. Safford

IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598, USA

Research Division
Almaden – Austin – Beijing – Delhi – Haifa – T.J. Watson – Tokyo – Zurich

This paper has been submitted for publication in **IEEE Security & Privacy** magazine.

Security and Performance Trade-Offs in I/O Operations for Virtual Machine Monitors

Paul A. Karger and David R. Safford
IBM Corporation, Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
karger@watson.ibm.com safford@watson.ibm.com

ABSTRACT

VMMs have become an attractive way to improve system security by providing strong isolation between different guest operating systems. However, the primary reason to believe that a VMM can create stronger isolation between guest operating systems than the operating system can create isolation between processes is that the VMM can be much smaller and simpler than the operating system. Madnick and Donovan [31] first proposed VMMs for security in 1973 by pointing out that "… since virtual machine monitors tend to be shorter, simpler, and easier to debug than conventional multiprogramming operating systems, … the VMM is less error-prone." One difficult problem in keeping a VMM small is the complexity of modern I/O architectures and devices. Many current hypervisors move the large, complex, and sometimes proprietary device drivers out of the VMM, into one or more guest partitions which causes inherent tradeoffs in complexity, security and performance. This paper discusses these tradeoffs.

1 Introduction

This paper covers two separate issues in virtualization of I/O. One involves the size and complexity of the software needed to handle the I/O, which impacts the security of the system. The other issue is the performance of the I/O, which impacts all hypervisors, regardless of their security requirements. Resolving these issues is not simple, because they are tightly intertwined in the implementation of I/O in the hypervisor. These two issues are particularly relevant to modern hypervisors.

In 1970, when Virtual Machine Monitors (VMM) were first developed [33] for the IBM System/360 Model 67 mainframe computer [6] the strategy for I/O emulation was quite straightforward as all I/O is done by channel programs. A channel is essentially a special purpose stored-program computer that is optimized for high-performance I/O, and a mainframe typically has many of them. When a device driver wishes to perform I/O, it writes a channel program that can perform many I/O operations, including conditional operations, all triggered by a single privileged Start I/O instruction.

Virtualization of I/O is straightforward, as the Start I/O from a guest operating system will trap to the hypervisor which can easily re-write the instructions to reference only the I/O devices and memory locations authorized to the virtual machine. As this single virtualization mechanism

covers all channel attached devices, including disk and network, the hypervisor code can be quite small and simple. Also, since each Start I/O trap can perform many I/O operations, the overhead for these hypervisor traps is minimized. (The only security problem is with self-modifying channel programs, because the addresses in question could not be relocated at Start I/O time, since the addresses could change dynamically. Initially hypervisors banned self-modifying channel programs, until PR/SM introduced I/O MMUs. Note that I/O channels have nothing to do with covert or side channel security problems.)

This type of I/O virtualization using channel programs is still very relevant today, as it contributes to IBM's PR/SM product receiving an EAL5 Common Criteria evaluation [4], the highest assurance level for any hypervisor product on the market today.

Problems developed in virtualizing I/O interfaces as soon as there was an attempt to build a VMM for a computer that did not have channels. The UCLA PDP-11 Virtual Machine Monitor [38] had to deal with UNIBUS devices that could be affected by almost any unprivileged CPU instruction. Their solution was a special hardware device [43] to generate the necessary traps. Furthermore, the vast array of devices available for the UNIBUS had little or no standardization of interface. Thus, a simple re-writing of channel programs was no longer sufficient to make the virtualization work.

With modern I/O architectures, the virtualization of I/O devices has become even more complex, and modern hypervisors have responded to this complexity in a number of different ways.

2 Security Benefits of Hypervisors

It is a commonly held belief that hypervisors provide significant security benefits, because they are much smaller and simpler than full operating systems. Madnick and Donovan [31] first proposed VMMs for security in 1973 by pointing out that "… since virtual machine monitors tend to be shorter, simpler, and easier to debug than conventional multiprogramming operating systems, … the VMM is less error-prone." The early hypervisors, such as CP/67-CMS [33] were much smaller than operating systems of the day, and their successors, such as z/VM are similarly still quite small and simple. However, the reality of this security benefit is nuanced, as modern hypervisors, such as Xen [12] are actually much larger and more complex, because they include full operating systems in their special privileged I/O partitions. Why these partitions must be fully trusted is discussed in Section 4.1.

The feasibility of using some hypervisors for very high levels of security has been demonstrated by the KVM/370 project [20] and DEC's A1-secure VMM project [29]. Note that KVM/370 should not be confused with kvm, the Linux Virtual Machine Monitor [30].

3 Classes of Hypervisors

A key aspect of modern hypervisor design concerns three issues in providing virtualized access to the current complex devices:

- Are devices drivers shared?
- Are the device drivers trusted?
- Where are the device drivers located?

Different hypervisors answer these questions differently, with inherent tradeoffs in complexity, security, and performance. Pure isolation hypervisors, such as IBM's PR/SM [18] on zSeries mainframes and MILS [11] let each guest have its own dedicated I/O hardware and device drivers, with no sharing of devices between guests. This comes with an obvious limitation of scalability, and all of the security issues of any sharing are simply punted up to the guest level. Other sharing hypervisors, such as Xen [12] and IBM's PHYP [17, 44] on pSeries systems, place the device drivers in a specially privileged partition, and this partition shares the devices with other guests through the VMM. The use of a privileged I/O partition comes with inherent performance and security issues. The tradeoffs actually have little to do with the content of the drivers themselves. This paper discusses these inherent tradeoffs that come from where the drivers are placed in the system.

3.1 Pure Isolation Hypervisors on a Server

The simplest and most secure case is a pure isolation hypervisor on a server system (Figure 1), such as PR/SM on an IBM System z server, a MILS separation kernel [45], or a server version of Xen with I/O MMU support on x86 hardware. In this case, no devices are directly shared, with each guest partition having its own dedicated storage and network, and, as a server, there is no physical display or keyboard to deal with.

In this case, the hypervisor can be very small and simple, the device drivers are located in the guests, and the device drivers are not trusted. As there is no sharing of a given device, most of the difficult timing, allocation, and traffic flow issues go away. The hypervisor still needs to be high assurance, but it can be a simple, small code base. However, side channel attacks on processor cache [15, 36] and/or branch predication units [9, 10] remain serious threats.

If pure isolation is all that is desired, then this is sufficient. A MILS (Multiple Independent Levels of Security) separation kernel isolates each level of security and does not directly permit sharing across levels. A Top Secret process or guest VM would not be allowed to read data from an Unclassified process or guest VM. This is in contrast to a traditional multi-level security (MLS) model, such as Bell and LaPadula [14], which would allow a Top Secret process read-only access to the data belonging to an Unclassified process. If there needs to be sharing across security levels in a pure isolation system, such as MILS, that can be accomplished by creating a special Guard partition above the separation kernel that allows selective flows between levels. However, such Guard partitions have several issues that are not present in MLS kernels, such as Multics [46] or the DEC A1-secure VMM [29]:

- performance cost - extra context switching from guest to guest
- incompatibility – if two different MLS layers are built on top of a pure isolation kernel by two different contractors, applications may not be compatible back and forth. The DoD has suffered from this type of incompatibility many times in the past.
- difficulty of composite evaluation – MLS layer design and evaluation will need complete information about exactly how the hypervisor is built. If the MLS layer contract is won by a different contractor (who competes with the first), that information may not be available. Worse still, if two different evaluators are used, the evaluation reports may be proprietary and not available. See [27] for a discussion of how difficult composite

evaluation can be and how composite evaluation can completely miss serious security vulnerabilities.

If sharing applications are needed, then it would be better to select a sharing hypervisor, rather than trying to implement the sharing on top of an isolation hypervisor. See [25] for details on why sharing hypervisors are important for many DoD multi-level security (MLS) applications.

3.2 Sharing Hypervisors on a Server

With a sharing hypervisor on a server, we fortunately don't have to deal with the display and keyboard problems, but still do need to handle shared network and shared storage devices. For the network device, if the guests encrypt all traffic, then the data cannot be directly leaked by the device driver, but there are still covert timing channel [22] and traffic flow analysis channels [13, 26, 35]. Some of this could be blinded from the device driver by the guest and/or hypervisor using continuous transmission protocols and slotted transmission allocations, but it is difficult for the device driver to be completely blinded. Examples of how high-assurance hypervisors can handle such covert channels can be found in the work of Schaefer, et. al. [39], Hu [22], and Karger and Wray [28].

In a similar vein, the device driver for shared storage, even if all data is encrypted, must be trusted, because it is difficult to blind it from covert timing and traffic flow analysis channels. An untrusted storage driver can learn a lot from traffic flow analysis, as discussed in [42], and it can also be subject to covert channel attacks [28].

Sharing drivers cannot be located in the guests, but must be located either in the hypervisor (Figure 2), or in a privileged I/O partition (Figure 3). The main point here is that simply moving the shared driver out of the hypervisor does not remove it from the TCB – instead, the TCB is now expanded to include the driver and any parts of the privileged partition with control over the driver.

In the existing implementations of privileged I/O partitions, such as Xen [12] or [17], the privileged I/O partition includes an entire operating system. As a result, the size and complexity of the code is comparable to the operating systems themselves. The Linux-based kvm [30] does not use a privileged I/O partition. Instead it places the hypervisor in a full Linux kernel which means that again the size and complexity is comparable to a guest operating system.

3.3 Sharing Hypervisors on a Client

By far, the most difficult situation is a client where all devices (display, keyboard, storage, network) must be shared. All of the sharing issues from the server (shared storage, and network) apply, but we must now also try to solve the display and keyboard problem. Fortunately, keyboard drivers are small and reasonably simple, and it is easy for the hypervisor to virtualize them safely, and to provide secure attention key at the same time. The trusted display device driver is an extremely difficult, unsolved problem. Unless the display device can accept encrypted data (such as with High-bandwidth Digital Content Protection (HDCP) [5]) displays, the data can't be encrypted, the device drivers are often very large (2-3 MB), and typically are proprietary, and thus very difficult/impossible to modify. Since the driver deals with unencrypted

data at multiple levels, and the driver is low assurance, there is high risk of direct leakage, let alone timing channel attacks [37].

4 Special Privileged I/O Partitions

Modern VMMs, such as Xen [12] and PHYP [17] address the difficulties of virtualizing the huge numbers of I/O drivers by creating special privileged partitions, called Dom0 in Xen and VIOS in PHYP. The VMMs then run a full Linux or AIX system in the special partitions and redirect all I/O requests from guest virtual machines to these special partitions which are granted the privilege to directly control the real I/O devices. This approach has been very attractive to VMM developers, because it allows them to support a very large number of possible I/O devices very quickly, with little development work required, and greatly helps keep the VMM small. In particular, the use of a special privileged I/O partition can significantly reduce the time to market for a new hypervisor, and this can be critical in getting market acceptance of a product.

However, this choice of special privileged partitions for I/O drivers is not without costs. This paper will examine the costs of the special partitions to see what the VMM designers have given up in exchange for a smaller VMM and getting large numbers of I/O drivers running quickly and easily. These costs come in two places – security and performance.

4.1 Security Implications of Special Privileged I/O Partitions

The first cost of special privileged I/O partitions is security. The problem is that the special privileged I/O partition is effectively part of the VMM, and it contains an entire Linux operating system. Now, rather than the VMM being smaller and simpler than the guest operating system, it is actually larger and more complex, since it consists of the small VMM kernel plus an entire operating system. In Xen, Dom0 runs an entire Linux system. While some may argue that the I/O guests are de-privileged with respect to the VMM, running in real ring 1 (virtual ring 0) while the VMM runs in real ring 0, the drivers still have to ensure secure sharing, particularly with respect to issues such as side and covert channels, and thus must remain trusted.

From a theoretical perspective, this is a fundamental limitation of protection ring systems. In his PhD thesis, Schroeder [40, p. 27] shows how a protection ring system, such as the one he and Saltzer had earlier developed for Multics [41], cannot support mutually suspicious subsystems in a single process. This is because all programs in a given protection ring must trust one another, and the hierarchic nature of rings means that they cannot be used for mutual suspicion. A special privileged I/O partition, such as Xen's Dom0, is performing I/O on behalf of multiple different, mutually suspicious guest VMs. Schroeder's theoretical results show the fallacy here. Even though the special privileged I/O partition is running in a less privileged protection ring, that does nothing to guarantee that the mutually suspicious guests are protected from one another. If the all I/O devices can each be dedicated to a single partition, then one could run a separate I/O Partition for each guest and avoid the security problems, because separate partitions avoid the hierarchic nature of protection rings. With an IOMMU [2, 23], one could assign the devices directly to the guest partitions and also avoid the performance problems described below. (IOMMUs were first proposed for Multics [16] and the first implementation was for the Honeywell SCOMP [19].) However, some devices, most notably display devices, must inherently be shared and therefore must deal with sensitive data from multiple guests. For those

cases, neither special privileged I/O partitions nor assigning directly to the guest partition will be sufficient.

In principle, the special privilege I/O partition need not run a full operating systems, but only a set of device drivers. In that case, however, the drivers could not simply be copied from an existing operating system, such as Linux, but would need significant modifications. No current hypervisor have tried this approach.

4.2 Performance Cost of Special Privileged I/O Partitions

The second cost of special privileged I/O partitions is performance. VMMs have always performed best on compute-bound workloads and worst on I/O-bound workloads, because of the cost of translating the I/O operations. Even on mainframes with I/O channels, the cost of translating the channel programs is significant. However, the special privileged I/O partitions introduce significant additional performance costs. In this section, we use the term *context switch* to include both ring crossings and process switches. In most machines, ring crossings are less expensive than process switches. However, that distinction serves only to complicate the analysis for this paper.

With no VMM at all, an I/O operation requires a context switch from user mode to supervisor mode to start the I/O operation, and later when the I/O interrupt occurs, a context switch from supervisor mode to user mode. A traditional VMM doubles that cost, because there must be context switches from user mode to the guest operating system and from the guest operating system to the VMM. When you introduce the special I/O partitions, the number of context switches significantly increases, as the VMM must send the I/O operations to the special I/O domain which then must initiate the actual I/O operations. There can also be additional costs due to either copying or mapping the data into the I/O partition.

In the following figures, the hypervisor is shown running in real ring 0, and the guest operating system is shown running in virtual ring 0. There are multiple approaches to virtualizing protection rings, and the concepts in this paper are the same, regardless of the approach chosen. Intel and AMD have chosen to add an extra protection ring for the hypervisor [1, 7], essentially a ring -1. That technique does not support self-virtualization (running a hypervisor in a partition on top of a hypervisor). Alternatively, the technique of ring compression [21] could be used which does support self-virtualization. It is important to note that the addition of hardware virtualization support by Intel and AMD has no direct impact on I/O performance. In fact, as will be seen in *Figure 7*, the software in Xen that uses hardware virtualization in Dom0 actually makes the performance worse. In addition, Intel [23] and AMD [3] have added IOMMU support, and *Figure 6* and *Figure 7* for the Xen I/O partition cases, assume the use of the IOMMU.

Figure 4 shows the context switches required for an asynchronous I/O operation initiated from an application running in ring 3 on an operating system that is running directly on the hardware without an underlying hypervisor. Each numbered step corresponds to a context switch shown in the figure.

1. The application issues a system call to initiate an asynchronous I/O operation.
2. The operating system starts the I/O operation on the I/O device.

3. The operating system returns to the application, since this is asynchronous I/O.
4. Some time later, the I/O operation completes and sends an interrupt to the operating system.
5. The operating system generates an interrupt to the application with the results of the completed I/O operation.

Figure 5 shows the context switches for an asynchronous I/O operation when the operating system is running above a conventional hypervisor in which the I/O drivers are in the hypervisor itself. Such hypervisors include CP/67 [33], z/VM [8], and the DEC A1-secure VMM for the VAX [29]. Each numbered step corresponds to a context switch shown in the figure.

1. The application issues a system call to initiate an asynchronous I/O operation.
2. The guest operating system starts the I/O operation. In a fully virtualized hypervisor, the START IO instruction traps to the hypervisor. In a para-virtualized hypervisor, the operating system issues an HCALL to the hypervisor.
3. The hypervisor initiates the I/O operation on the I/O device.
4. The hypervisor returns to the guest operating system, as this is an asynchronous I/O request.
5. The guest operating system returns to the application program, as this is an asynchronous I/O request.
6. Some time later, the I/O operation completes and sends an interrupt to the hypervisor.
7. The hypervisor generates a virtual interrupt to the guest operating system.
8. The guest operating system generates an interrupt to the application with the results of the completed I/O operation.

Figure 6 shows the context switches required for an asynchronous I/O operation in which the hypervisor I/O control is entirely done in a special privileged I/O partition in virtual ring 0 of that partition. This is how Xen operates for paravirtualized domains. Each numbered step corresponds to a context switch shown in the figure.

1. The application issues a system call to initiate an asynchronous I/O operation.
2. The guest operating system starts the I/O operation. In a fully virtualized hypervisor, the START IO instruction traps to the hypervisor. In a para-virtualized hypervisor, the operating system issues an HCALL to the hypervisor.
3. The hypervisor passes the I/O operation to the special privileged I/O partition.
4. The hypervisor returns to the guest operating system, as this is an asynchronous I/O request.
5. The guest operating system returns to the application program, as this is an asynchronous I/O request.
6. The operating system in the special privileged I/O partition uses the IOMMU to gain access to the I/O device and start the I/O.
7. Some time later, the I/O operation completes and sends an interrupt to the hypervisor.
8. The hypervisor generates a virtual interrupt to the special privileged I/O partition operating system.
9. The special privileged I/O partition operating system completes the I/O and makes an HCALL to the hypervisor to wake up the guest operating system.

10. The hypervisor delivers the I/O completion interrupt to the guest operating system.
11. The guest operating system generates an interrupt to the application with the results of the completed I/O operation.

Figure 7 shows how Dom0 in the Xen hypervisor [12] actually works for fully virtualized I/O. Xen controls the I/O, not from virtual ring 0 of Dom0, but rather from virtual ring 3 (the user ring) of Dom0. This introduces still more context switches. Each numbered step corresponds to a context switch shown in the figure.

1. The application issues a system call to initiate an asynchronous I/O operation.
2. The guest operating system starts the I/O operation. In a fully virtualized hypervisor, the START IO instruction traps to the hypervisor. In a para-virtualized hypervisor, the operating system issues an HCALL to the hypervisor.
3. The hypervisor passes the I/O operation to Dom0.
4. The hypervisor returns to the guest operating system, as this is an asynchronous I/O request.
5. The guest operating system returns to the application program, as this is an asynchronous I/O request.
6. The operating system in Dom0 now starts the I/O virtualization software in virtual ring 3.
7. The I/O virtualization software now issues an asynchronous I/O request to the Dom0 operating system.
8. The Dom0 operating system uses the IOMMU to start the I/O device.
9. The Dom0 operating system returns to the virtual ring 3 I/O virtualization software, as this is an asynchronous I/O request.
10. Some time later, the I/O operation completes and sends an interrupt to the hypervisor.
11. The hypervisor sends a virtual interrupt to the Dom0 operating system.
12. The Dom0 operating system generates a virtual interrupt to the virtualization software in virtual ring 3.
13. The virtualization software in virtual ring 3 completes the I/O and issues a system call to the Dom0 operating system to wake up the guest operating system..
14. The Dom0 operating system issues an HCALL to the hypervisor to wake up the guest operating system.
15. The hypervisor delivers the I/O completion interrupt to the guest operating system.
16. The guest operating system generates an interrupt to the application with the results of the completed I/O operation.

We can see that the Xen Dom0 approach requires roughly 3 times the number of context switches of an operating system running directly on the hardware and roughly double the number of context switches of a conventional hypervisor with I/O drivers in the hypervisor itself. These extra context switches can explain some of the performance issues seen by Menon, et. al. [32].

4.3 Analyzing the Performance Implications

The previous section predicted the performance implications of special privileged I/O partitions, by counting context switches. Several other papers have measured actual performance tradeoffs, consistent with these counts.

Nakajima and Mallik [34] discuss the performance tradeoffs between software, hardware, and para-virtualization in the Intel environment. They measure several aspects of file I/O performance on KVM, and on Xen with paravirtualization with the drivers in DOM0. They did not measure Xen's worst case of full virtualization, in which the DOM0 application level provides device emulation. Even with the milder para-virtualization case, Xen's I/O performance was two to four times slower than KVM's, which fits well with our analysis.

Menon, et. al. [32] measure the DOM0 performance for TCP/IP, and show a throughput cap roughly one third that of a native Linux kernel on the same hardware. In their analysis, the dominant issue was TLB misses and flushes, with context switch overhead second.

Zhang, et. al. [47] also discuss the performance impact of DOM0 based TCP/IP, showing a full order of magnitude reduction in throughput of the para-virtualized Xen/DOM0 case compared to native Linux. Their measurements indicated that the performance problem was due mainly to the overhead of multiple hypercalls, and second to the number of TLB flushes associated with the context switches.

These studies show anywhere from a factor of 2 to 10 overhead in I/O for the para-virtualized DOM0 case compared to native, for file and network operations, which is consistent with our predictions. Interestingly none of the studies measured the worst case of full virtualization in DOM0.

It is important to note that processors with address space tags in the TLBs can reduce some of the performance costs of the context switches. Karger [24] includes a much more in-depth study of reducing the cost of context switching in high-security systems. He examines not just TLB tagging, but also register save/restore requirements. His measurements were based on the design of DEC's A1-secure virtual machine monitor [29], but the hypervisor is not actually mentioned in [24], as it was still an unannounced product.

5 Conclusions

We have discussed the difficulty in virtualization of modern complex I/O architectures. Different hypervisors have approached this complex I/O virtualization in different ways, with differing tradeoffs in implementation complexity, security, and performance. Depending on the goals and requirements for a given system, particularly with respect to the type of devices present, and whether or not they need to be shared, different hypervisors will be more or less suitable. The most difficult case is when devices, particularly a display adapter need to be shared. In this case, the choice of a separate I/O partition provides significant ease of implementation, at the cost of large TCB and poorer performance due to a doubling of context shifts.

6 Acknowledgements

We must thank David Toll, J. R. Rao, and Michael Steiner for their comments and suggestions on the paper, and particularly Stefan Berger for his assistance on *Figure 6* and *Figure 7*.

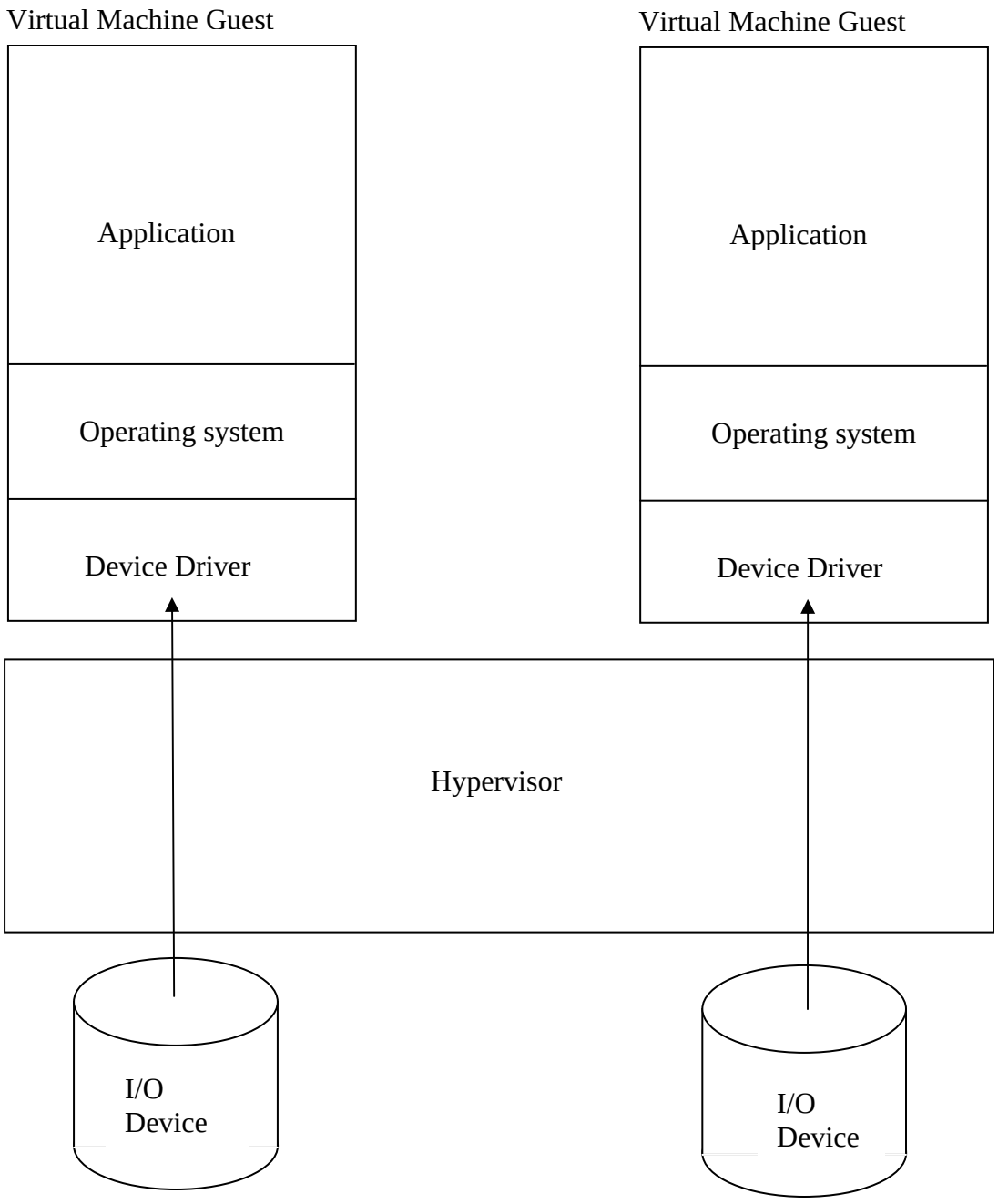

Figure 1. Pure Isolation Hypervisor with device drivers in Guest partitions

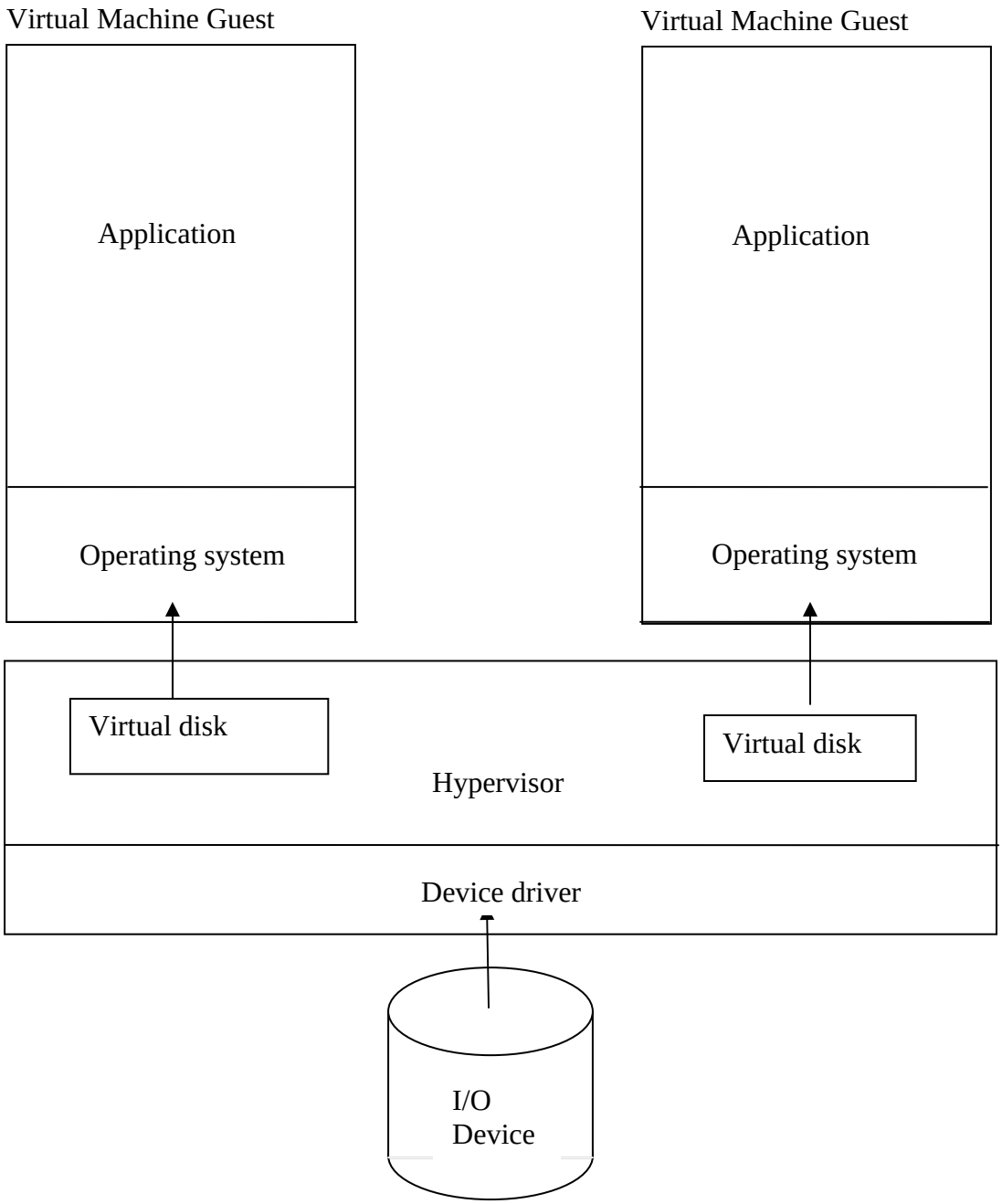

Figure 2. Sharing Hypervisor with Device Drivers in the Hypervisor

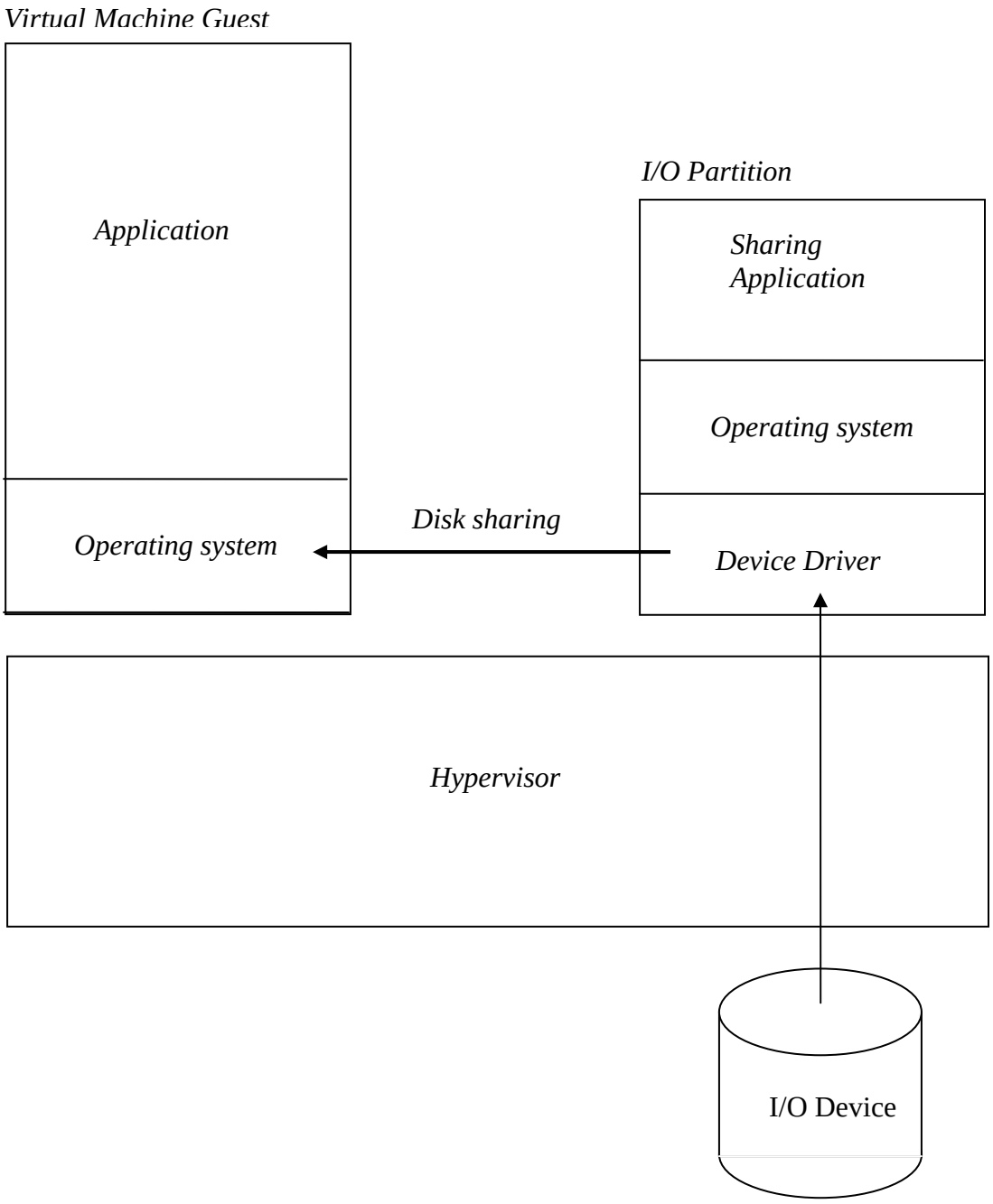

Figure 3. Sharing Hypervisor with Device drivers in Privileged I/O Partition

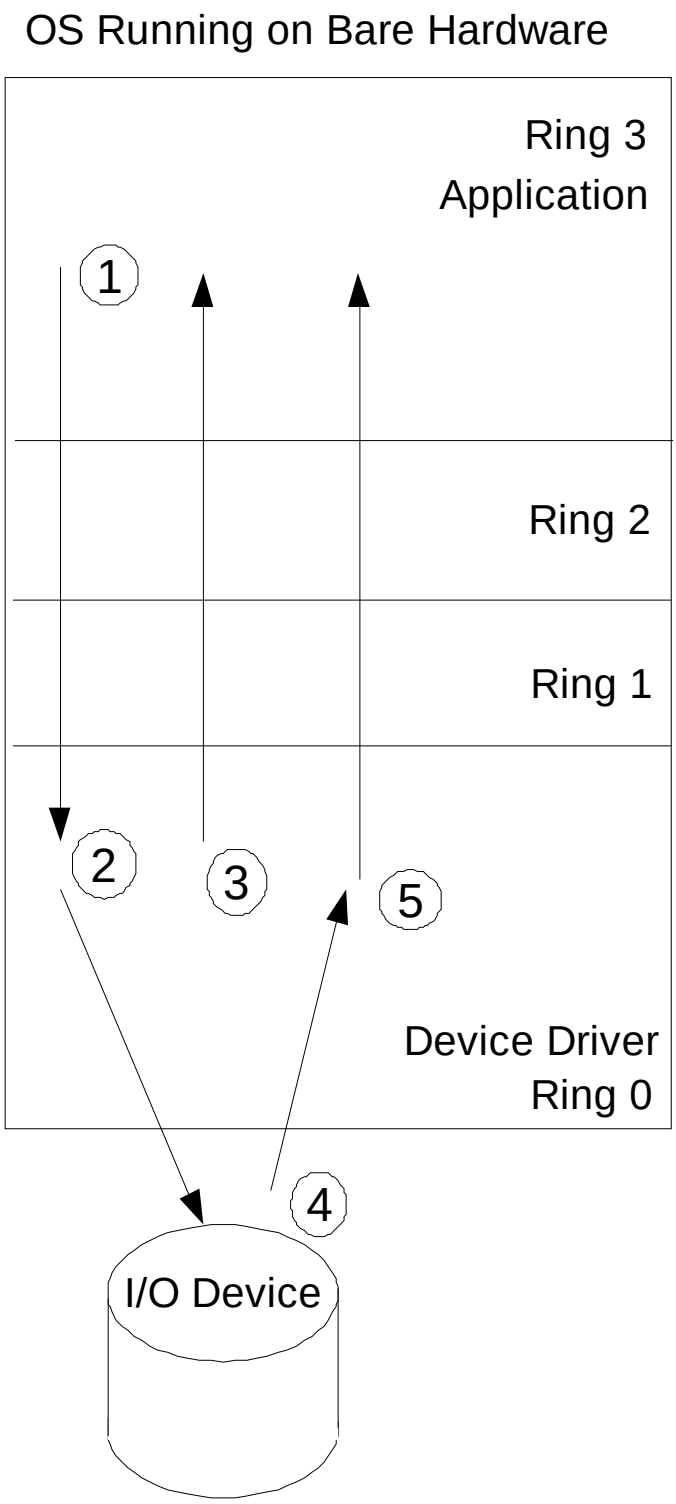

Figure 4. . I/O in an Operating System without Virtualization

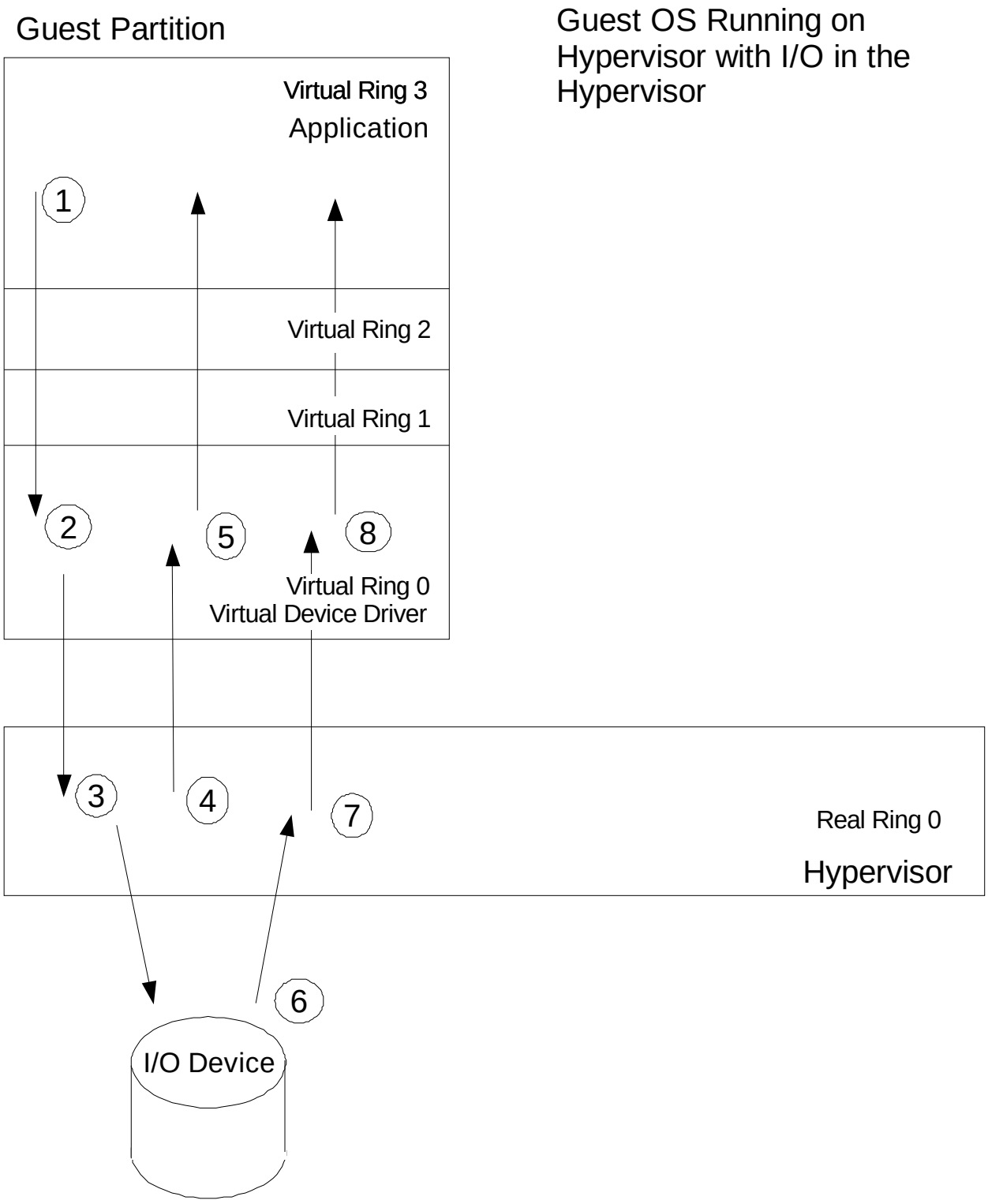

Figure 5. . Guest OS Running on a Hypervisor with I/O in the Hypervisor

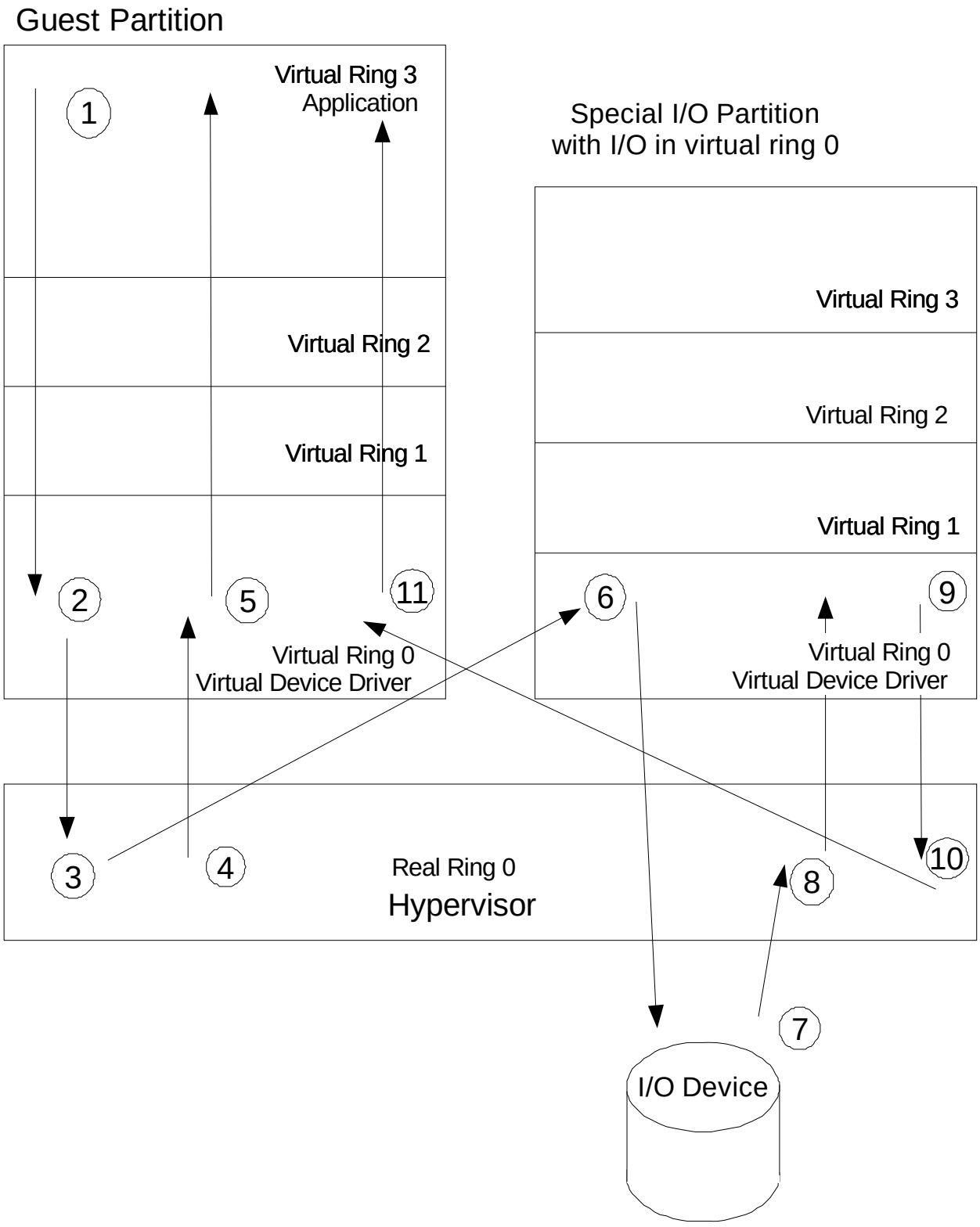

Figure 6. . Special I/O Partition with I/O drivers in virtual ring 0

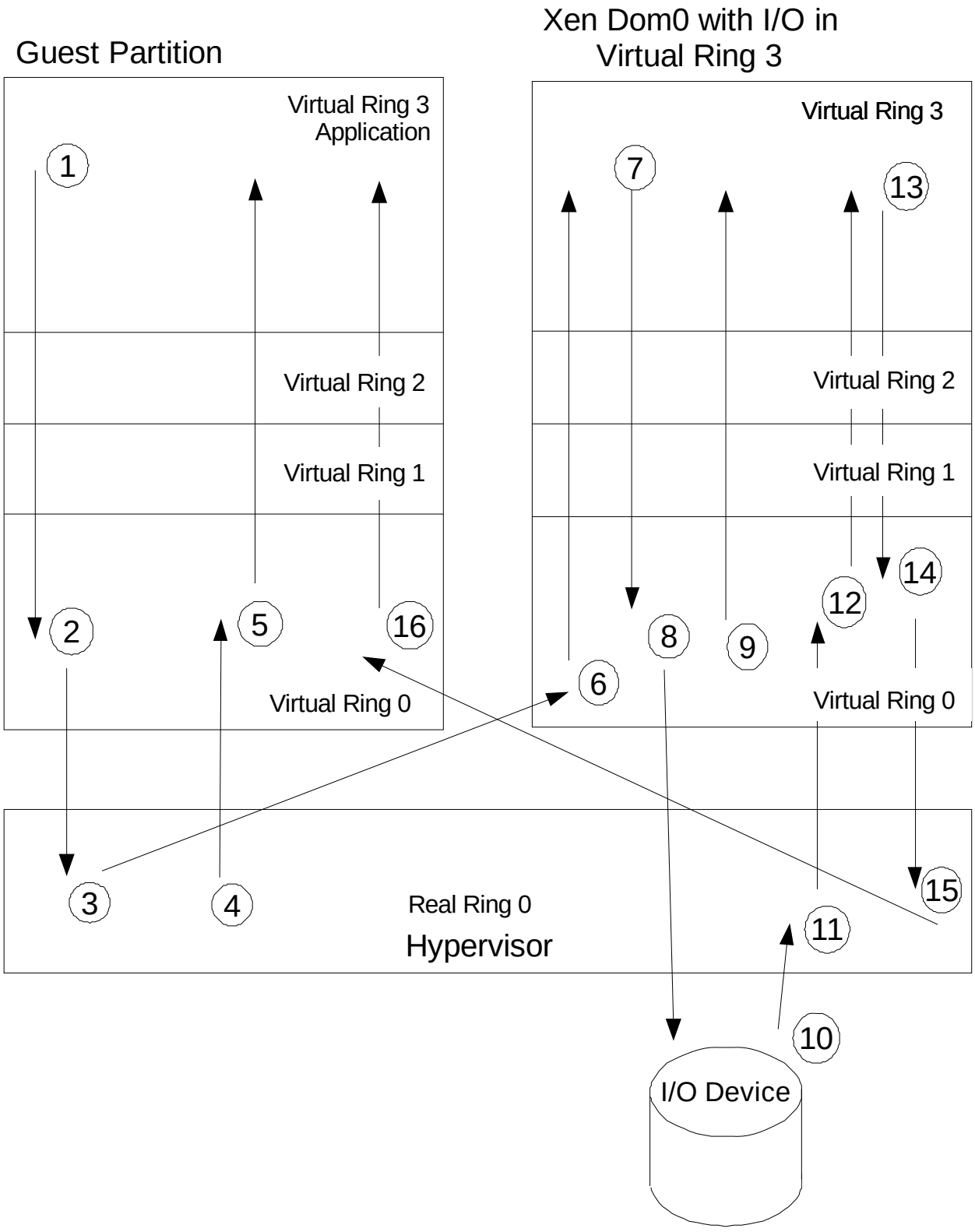

Figure 7. Xen Dom0 with I/O controlled from virtual ring 3

7 References

1. *AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual*, Publication No. 33047, Revision 3.01, May 2005, Advanced Micro Devices: Sunnyvale, CA. URL: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/33047.pdf

2. *AMD I/O Virtualization Technology (IOMMU) Specification*, Publication No. 34434, 3 February 2006, Advanced Micro Devices. URL: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf

3. *AMD I/O Virtualization Technology (IOMMU) Specification*, Publication # 34434, Revision 1.20, February 2007, Advanced Micro Devices. URL: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf

4. *Certification Report for PR/SM LPAR for the IBM System z9 Enterprise Class and the IBM System z9 Business Class*, BSI-DSZ-CC-0378-2006, 4 September 2006, Bundesamt für Sicherheit in der Informationstechnik: Bonn, Germany. URL: http://www.commoncriteriaportal.org/files/epfiles/0378a.pdf

5. *High-bandwidth Digital Content Protection System*, Revision 1.3, 21 December 2006, Digital Content Protection LLC: Beaverton, OR. URL: http://www.digital-cp.com/files/static_page_files/8006F925-129D-4C12-C87899B5A76EF5C3/HDCP_Specification%20Rev1_3.pdf

6. *IBM System/360 Model 67 Functional Characteristics*, A27-2719-0, 1967, IBM Corporation: Kingston, NY. URL: http://www.bitsavers.org/pdf/ibm/360/funcChar/A27-2719-0_360-67_funcChar.pdf

7. *Intel Virtualization Technology Specification for the IA-32 Intel Architecture*, C97063-002, April 2005, Intel Corporation.

8. *z/VM: CP Programming Services*, SC24-6084-02, May 2006, IBM Corporation: Poughkeepsie, NY. URL: http://publibz.boulder.ibm.com/epubs/pdf/hcse5b11.pdf

9. Aciiçmez, O., Ç.K. Koç, and J.-P. Seifert. *On the Power of Simple Branch Prediction Analysis*. in **Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security**. 20-22 March 2007, Singapore: ACM. p. 312-320. URL: http://islab.oregonstate.edu/koc/papers/c40.pdf

10. Aciiçmez, O., Ç.K. Koç, and J.-P. Seifert. *Predicting Secret Keys Via Branch Prediction*. in **Topics in Cryptology - CT-RSA 2007: The Cryptographers' Track at the RSA Conference**. 5-9 February 2007, San Francisco, CA:Lecture Notes in Computer Science Vol. 4377. Springer. p. 225-242. URL: http://security.ece.orst.edu/koc/papers/c39.pdf

11. Alves-Foss, J., C. Taylor, and P. Oman. *A Multi-layered Approach to Security in High Assurance Systems*. in **Proceedings of the 37th Hawaii International Conference on System Sciences**. 5-8 January 2004, Waikoloa, HI: IEEE Computer Society. p. 90302.2b. URL: http://csdl.computer.org/comp/proceedings/hicss/2004/2056/09/205690302b.pdf

12. Barham, P., 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 Nineteenth ACM Symposium on Operating Systems Principles (SOSP)**. 19-22 October 2003, Bolton Landing, NY: ACM Press. URL: http://www.cl.cam.ac.uk/Research/SRG/netos/papers/2003-xensosp.pdf

13. Bauer, K., D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. *Low-Resource Routing Attacks Against Tor*. in **Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society**. 29 October 2007, Alexandria, VA: ACM. p. 11-20. URL: http://systems.cs.colorado.edu/%7Ebauerk/papers/wpes25-bauer.pdf

14. Bell, D.E. and L.J. LaPadula, *Computer Security Model: Unified Exposition and Multics Interpretation*, ESD-TR-75-306, March 1976, The MITRE Corporation, Bedford, MA: HQ Electronic Systems Division, Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/bell76.pdf

15. Bernstein, D.J., *Cache-timing attacks on AES*, 2005, University of Illinois at Chicago: Chicago, IL. URL: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

16. Biba, K.J., S.R. Ames, E.L. Burke, P.A. Karger, W.R. Price, R.R. Schell, and W.L. Schiller, *A Preliminary Specification of a Multics Security Kernel*, WP-20119, April 1975, The MITRE Corporation: Bedford, MA.

17. Blank, A., P. Keifer, C. Sallave Jr., G. Valencia, J. Wain, and A.M. Warda, *Advanced POWER Virtualization on IBM System p5*, SG24-7940-01, December 2005, IBM Corporation: Austin, TX. URL: http://www.redbooks.ibm.com/redbooks/pdfs/sg247940.pdf

18. Borden, T.L., J.P. Hennessy, and J.W. Rymarczyk, *Multiple Operating Systems on One Processor Complex*. **IBM Systems Journal**, 1989. **28**(1): p. 104-123. URL: http://www.research.ibm.com/journal/sj/281/ibmsj2801H.pdf

19. Broadbridge, R. and J. Mekota, *Secure Communications Processor Specification*, ESD-TR-76-351, Vol. II, June 1976, Honeywell Information Systems, Inc., McLean, VA: HQ Electronic Systems Division, Hanscom AFB, MA.

20. Gold, B.D., R.R. Linde, R.J. Peeler, M. Schaefer, J.F. Scheid, and P.D. Ward. *A Security Retrofit of VM/370*. in **AFIPS Conference Proceedings, Volume 48, 1979 National Computer Conference**. 1979, Montvale, NJ: AFIPS Press. p. 335-344.

21. Hall, J.S. and P.T. Robinson. *Virtualizing the VAX Architecture*. in **18th International Symposium on Computer Architecture**. May 1991, Toronto, ON, Canada: published in

Computer Architecture News, Vol. 19, No. 3. p. 380-389. URL: http://doi.acm.org/10.1145/115952.115990

22. Hu, W.-M. *Reducing Timing Channels with Fuzzy Time*. in **Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy**. 20-22 May 1991, Oakland, CA: IEEE Computer Society. p. 8-20.

23. Humphreys, J. and T. Grieser, *Mainstreaming Server Virtualization: The Intel Approach*, June 2006, IDC: Framingham, MA. URL: http://www.intel.com/business/technologies/idc_virtualization_wp.pdf

24. Karger, P.A., *Improving Security and Performance for Capability Systems*, Computer Laboratory Technical Report No. 149, October 1988, University of Cambridge: Cambridge, England. URL: http://domino.research.ibm.com/comm/research_people.nsf/pages/karger.pubs.html/$FILE/trthesis.pdf

25. Karger, P.A. *Multi-Level Security Requirements for Hypervisors*. in **21st Annual Computer Security Applications Conference**. 2005, Tucson, AZ: IEEE Computer Society. p. 240-248. URL: http://www.acsa-admin.org/2005/papers/154.pdf

26. Karger, P.A., *Non-Discretionary Access Control for Decentralized Computing Systems*, MIT/LCS/TR-179, May 1977, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. URL: http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-179.pdf

27. Karger, P.A. and H. Kurth. *Increased Information Flow Needs for High-Assurance Composite Evaluations*. in **Second IEEE International Information Assurance Workshop**. 8-9 April 2004, Charlotte, NC: IEEE Computer Society. p. 129-140.

28. Karger, P.A. and J.C. Wray. *Storage Channels in Disk Arm Optimization*. in **Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy**. 20-22 May 1991, Oakland, CA: p. 52-61.

29. Karger, P.A., M.E. Zurko, D.W. Bonin, A.H. Mason, and C.E. Kahn, *A Retrospective on the VAX VMM Security Kernel.* **IEEE Transactions on Software Engineering**, November 1991. **17**(11): p. 1147-1165.

30. Kivity, A., Y. Kamay, D. Laor, U. Lublin, and A. Liguori. *kvm: the Linux Virtual Machine Monitor*. in **Proceedings of the Linux Symposium**. 27-30 June 2007, Ottawa, ON, Canada: Vol. 1. p. 225-230. URL: http://ols.108.redhat.com/2007/Reprints/nakajima-Reprint.pdf

31. Madnick, S.E. and J.J. Donovan. *Application and Analysis of the Virtual Machine Approach to Information System Security*. in **Proceedings of the ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems**. 26-27 March 1973, Cambridge, MA: Association for Computing Machinery. p. 210-224. URL: http://portal.acm.org/citation.cfm?id=803961

32. Menon, A., J.R. Santos, Y. Turner, G.J. Janakiraman, and W. Zwaenepoel. *Diagnosing Performance Overheads in the Xen Virtual Machine Environment*. in **First ACM/USENIX Conference on Virtual Execution Environments**. 2005, Chicago, IL: p. 13-23. URL: http://www.hpl.hp.com/techreports/2005/HPL-2005-80.pdf

33. Meyer, R.A. and L.H. Seawright, *A Virtual Machine Time-Sharing System*. **IBM Systems Journal**, 1970. **9**(3): p. 199-218. URL: http://www.research.ibm.com/journal/sj/093/ibmsj0903D.pdf

34. Nakajima, J. and A.K. Mallick. *Hybrid-Virtualization - Enhanced Virtualization for Linux*. in **Proceedings of the Linux Symposium**. 27-30 June 2007, Ottawa, ON, Canada: Vol. 2. p. 87-96. URL: http://ols.108.redhat.com/2007/Reprints/nakajima-Reprint.pdf

35. Padlipsky, M.A., D.W. Snow, and P.A. Karger, *Limitations of End-to-End Encryption in Secure Computer Networks*, ESD-TR-78-158, August 1978, The MITRE Corporation: Bedford MA, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://stinet.dtic.mil/cgi-bin/GetTRDoc?AD=A059221&Location=U2&doc=GetTRDoc.pdf

36. Percival, C., *Cache Missing for Fun and Profit*, 2005. URL: http://www.daemonology.net/papers/htt.pdf

37. Picciotto, J. and J. Epstein. *A Comparison of Trusted X Security Policies, Architectures, and Interoperability*. in **Proceedings of the Eighth Annual Computer Security Applications Conference**. 30 November - 4 December 1992, San Antonio, TX: IEEE Computer Society. p. 142-152.

38. Popek, G.J. and C.S. Kline, *The PDP-11 Virtual Machine Architecture: A Case Study*. **Operating Systems Review**, 19-21 November 1975. **9**(5): p. 97-105. Proceedings of the Fifth ACM Symposium on Operating Systems Principles, Austin, TX.

39. Schaefer, M., B. Gold, R. Linde, and J. Scheid. *Program Confinement in KVM/370*. in **Proceedings of the 1977 ACM Annual Conference**. 16-19 October 1977, Seattle, WA: p. 404-410.

40. Schroeder, M.D., *Cooperation of Mutually Suspicious Subsystems in a Computer Utility*, Ph. D. Thesis, Department of Electrical Engineering, MAC TR-104, September 1972, Massachusetts Institute of Technology: Cambridge, MA. URL: http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-104.pdf

41. Schroeder, M.D. and J.H. Saltzer, *A Hardware Architecture for Implementing Protection Rings*. **Comm. ACM**, March 1972. **15**(3): p. 157-170.

42. Smith, S.W. and D. Safford, *Practical Server Privacy with Secure Coprocessors*. **IBM Systems Journal**, 2001. **40**(3): p. 683-695. URL: http://www.research.ibm.com/journal/sj/403/tocpdf.html

43. Vahey, M.D., *A Virtualizer Efficiency Device*, M.S. Thesis in *Department of Computer Science* 1975, University of California: Los Angeles, CA.

44. Valdez, E., R. Sailer, and R. Perez. *Retrofitting the IBM POWER Hypervisor to Support Mandatory Access Control*. in **Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)**. 10-14 December 2007, Miami Beach, FL: IEEE Computer Society. p. 221-231. URL: http://acsac.org/2007/papers/153.pdf

45. Vanfleet, W.M., J.A. Luke, R.W. Beckwith, C. Taylor, B. Calloni, and G. Uchenick, *MILS: Architecture for High-Assurance Embedded Computing.* **Crosstalk: The Journal of Defense Software Engineering**, August 2005. **18**(8): p. 12-16. URL: http://www.stsc.hill.af.mil/crosstalk/2005/08/0508Vanfleet_etal.pdf

46. Whitmore, J., A. Bensoussan, P. Green, D. Hunt, A. Kobziar, and J. Stern, *Design for Multics Security Enhancements*, ESD-TR-74-176, December 1973, Honeywell Information Systems, Inc., HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/whit74.pdf

47. Zhang, X., S. McIntosh, P. Rohatgi, and J.L. Griffin. *XenSocket: A High-Throughput Interdomain Transport for Virtual Machines*. in **ACM/IFIP/USENIX 8th International Middleware Conference**. 26-30 November 2007, Newport Beach, CA:Lecture Notes in Computer Science Vol. 4834. Springer. p. 184-203. URL: http://www.ece.cmu.edu/~griffin2/papers/2007-11-28-middleware-xensocket-a-high-throughput-interdomain-transport-for-virtual-machines.pdf