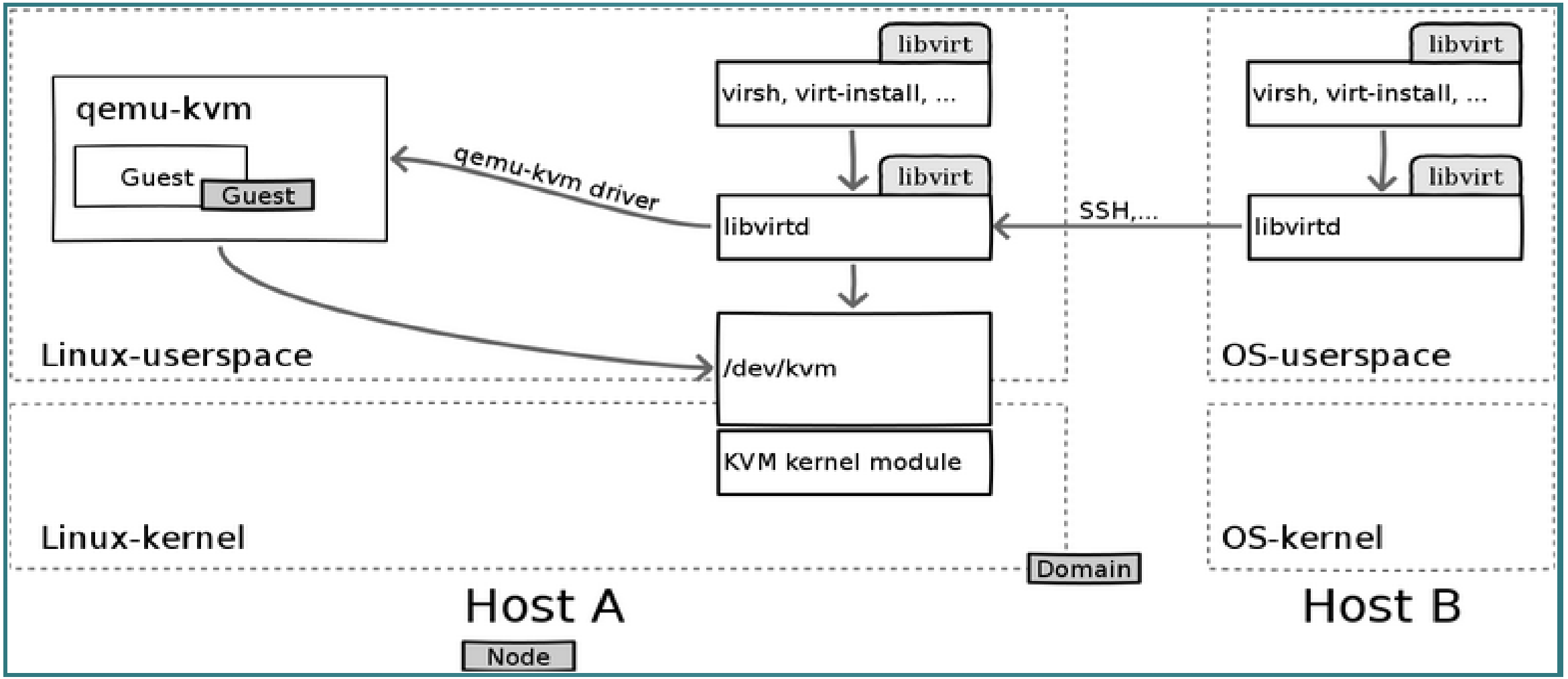
Workload Consolidation in Retail Market

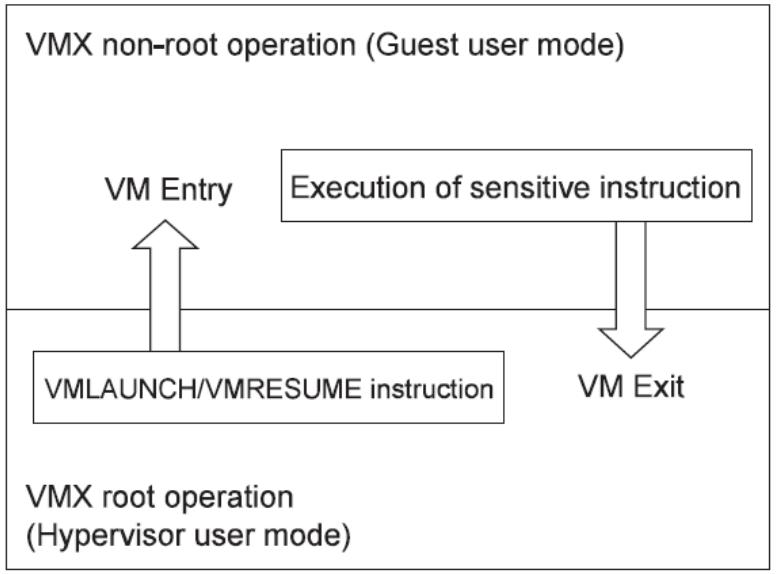
- Goal
  - Quick deployment
  - Minimize the cost of maintainance
  - Differentiate standard hardware devices by software-defined
- Key Technologies
  - Virtualization
    - KVM for frontend-interaction type services, especially for accommodating legacy software on new hardware platform
    - Docker for backend type services
  - Intel Machine Vision solution
    - Cloud-based inference
    - Edge/Fog-based inference

KVM-based Virtualization Runtime Model



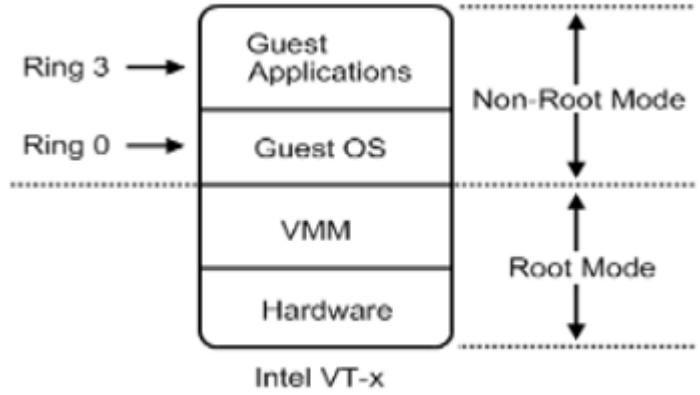
KVM Solution with Intel VT-x and VT-d

- Intel VT-x can be viewed as a “function” that switches processing to the hypervisor on detecting the execution of sensitive instruction by CPU via “multiple rings execution modes”, i.e. “VMX root operation” and “VMX non-root operation”
- Intel VT-d plays address conversion mechanism for I/O devices (IOMMU)
- Extended Page Table (EPT)
- Virtio, Device Assignment (Passthrough)
- Kernel Samepage Merging (KSM)



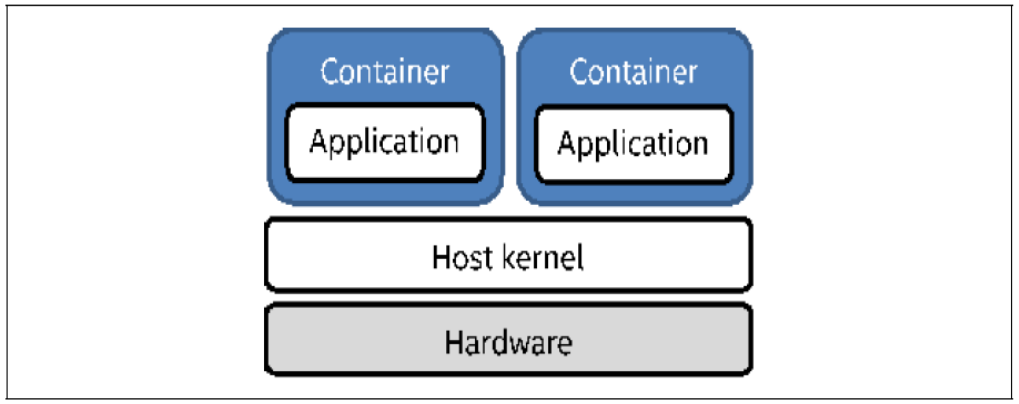
KVM正是基于Intel VT-x硬件辅助全虚拟化来实现的的，Intel VT-x CPU实现了不同的ring-X modes，这样可对guest OS正要执行的敏感指令进行ring mode切换，然后由KVM模块捕捉进行处理，同时又不影响到HostOS。

Reference: https://serverfault.com/questions/208693/difference-between-kvm-and-qemu 说明了从QEMU->KQemu->KVM过程。

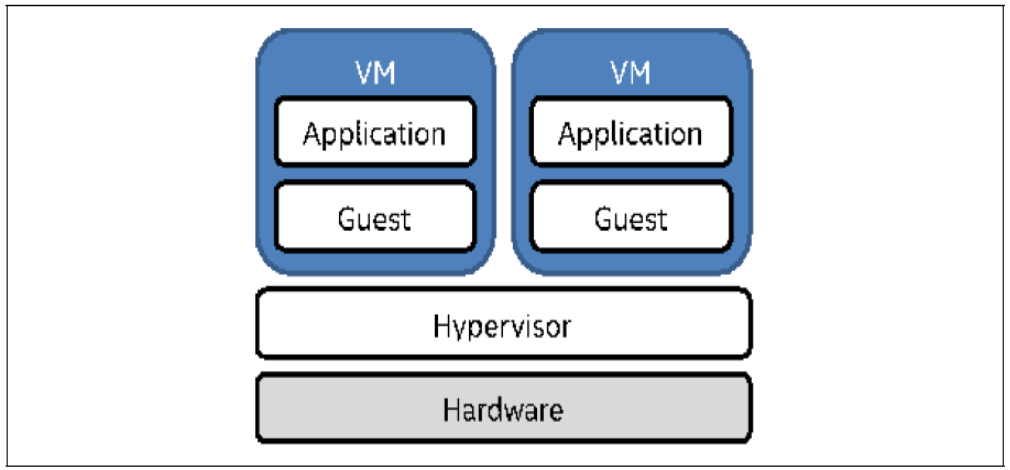


Comparison Between Docker and KVM

Representation of Two Containers



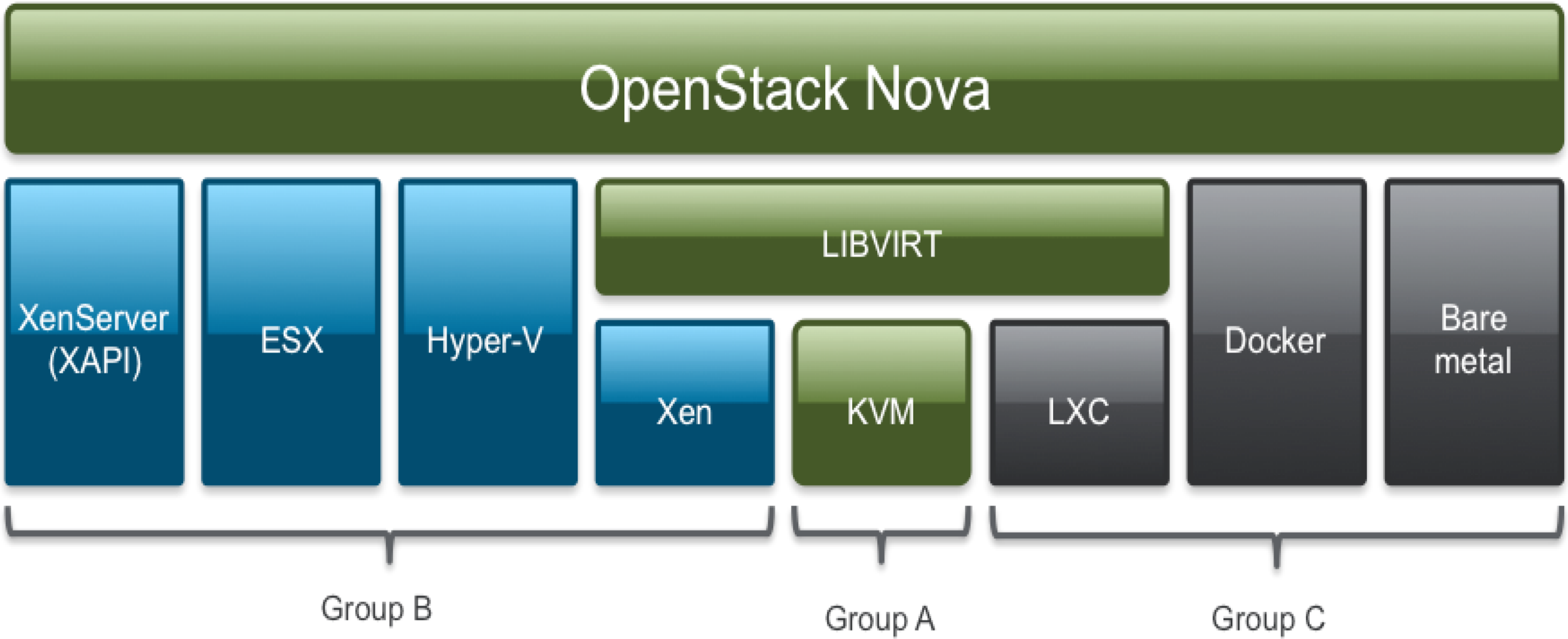
Representation of Two Virtual Machines



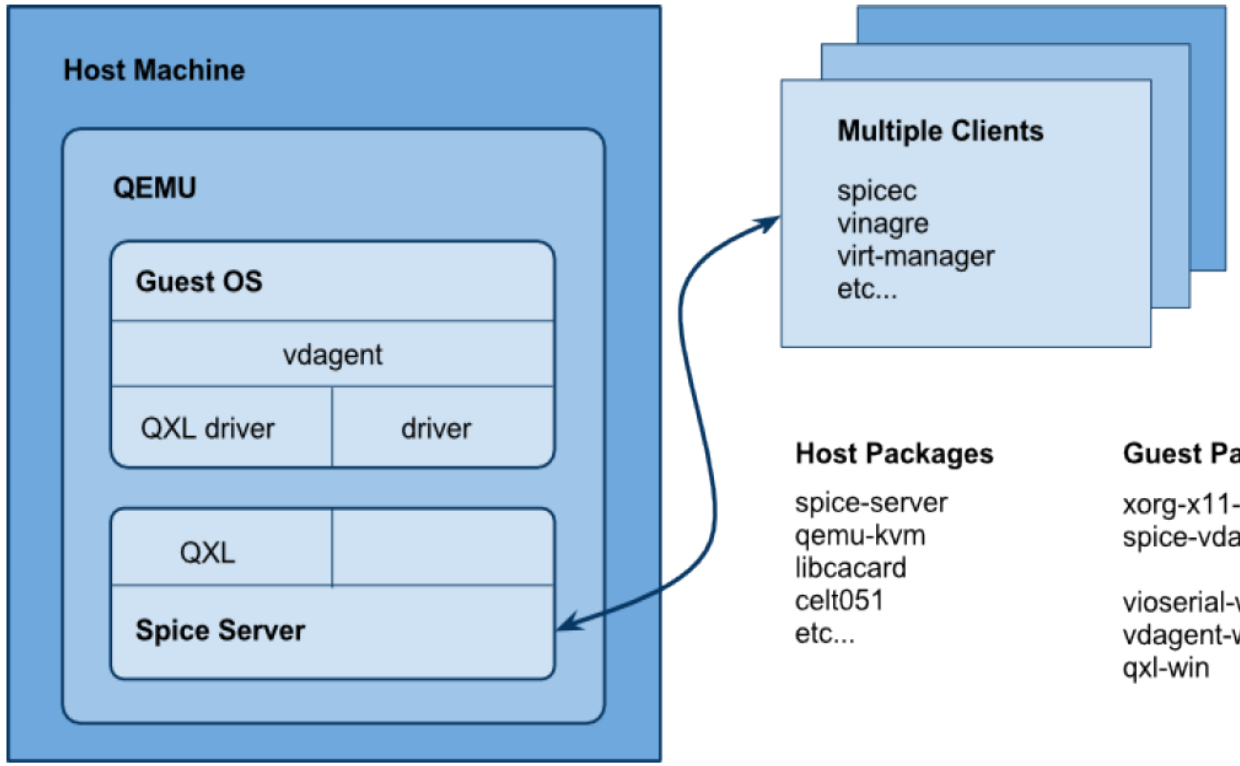
Comparison Between Docker Containers and KVM

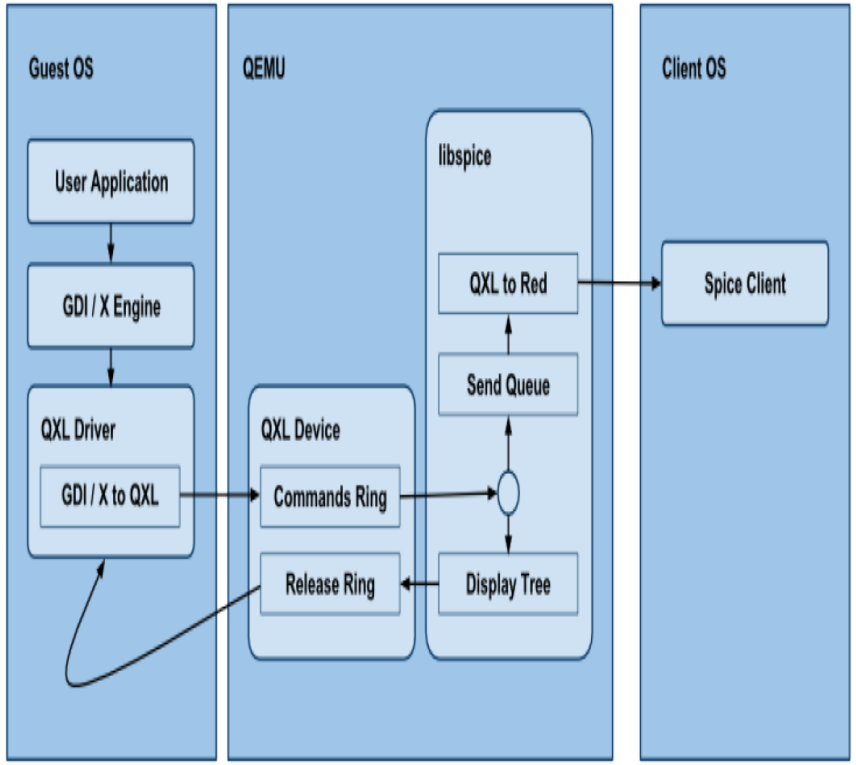
	Containers	Hypervisor-Based Virtual Machines
Choice of Operating System	Variant of host OS only	Any
Startup Time (Unoptimized Default)	1.5s	21s
Hardware Abstraction and Device Emulation	No	Yes
Disk Footprint	Small	Large
Memory Footprint	Small	Large
Density	High	Low
Configurability of Application	Flexible	Complex
Tunability from Guest	Insight into host	Black box machine
Security	Not mature and complex	Mature Security models

KVM & Docker VM Integration Scheme – Server Side



KVM Integration Scheme – Client Side





QEMU 可以使用一下几个图形输出：std, cirrus, vmware, qxl, xenfs 和 vnc。

Performance Comparison between KVM and Linux Container

TABLE I. RESULTS FOR PXZ, LINPACK, STREAM, AND RANDOMACCESS. EACH DATA POINT IS THE ARITHMETIC MEAN OF TEN RUNS. DEPARTURE FROM NATIVE EXECUTION IS SHOW WITHIN PARENTHESES "()". THE STANDARD DEVIATION IS SHOWN WITHIN SQUARE BRACKETS "[]".

Workload		Native	Docker	KVM-untuned	KVM-tuned
PXZ (MB/s)		76.2 [±0.93]	73.5 (-4%) [±0.64]	59.2 (-22%) [±1.88]	62.2 (-18%) [±1.33]
Linpack (GFLOPS)		290.8 [±1.13]	290.9 (-0%) [±0.98]	241.3 (-17%) [±1.18]	284.2 (-2%) [±1.45]
RandomAccess (GUPS)		0.0126 [±0.00029]	0.0124 (-2%) [±0.00044]	0.0125 (-1%) [±0.00032]	Tuned run not warranted
Stream (GB/s)	Add	45.8 [±0.21]	45.6 (-0%) [±0.55]	45.0 (-2%) [±0.19]	
	Copy	41.3 [±0.06]	41.2 (-0%) [±0.08]	40.1 (-3%) [±0.21]	
	Scale	41.2 [±0.08]	41.2 (-0%) [±0.06]	40.0 (-3%) [±0.15]	
	Triad	45.6 [±0.12]	45.6 (-0%) [±0.49]	45.0 (-1%) [±0.20]	

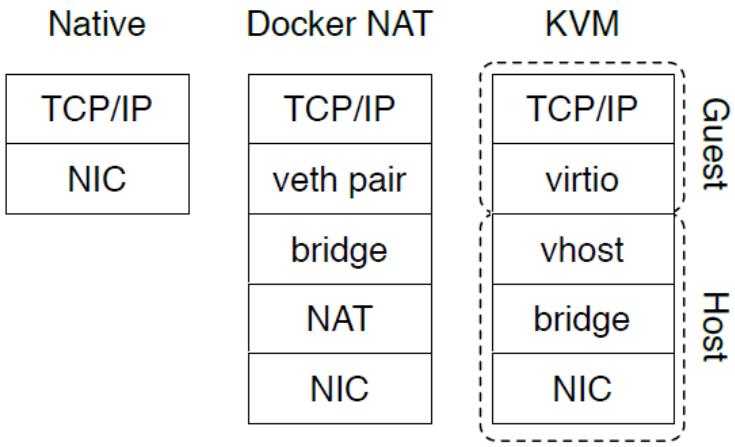


Fig. 1. Network configurations

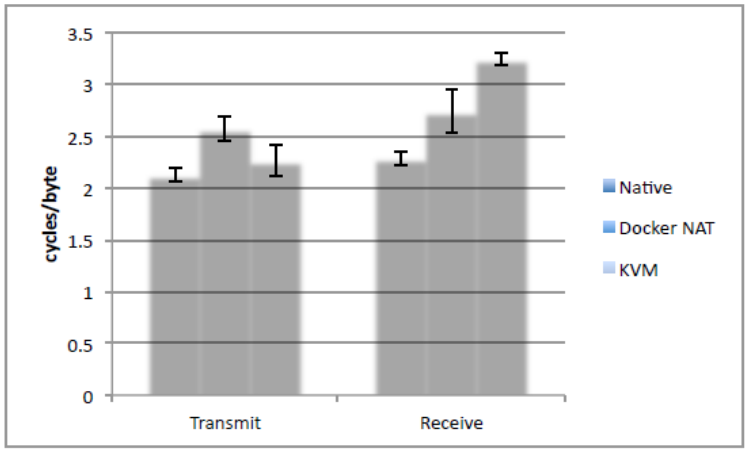


Fig. 2. TCP bulk transfer efficiency (CPU cycles/byte)

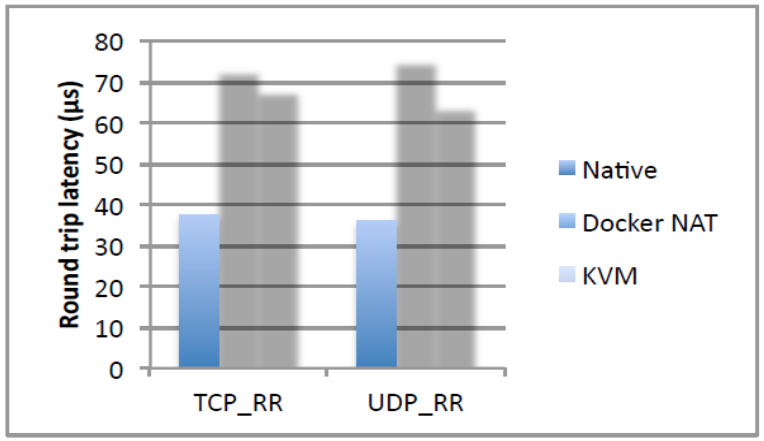


Fig. 3. Network round-trip latency (μs).

Reference:
https://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf
http://www.spec.org/virt_sc2010/results/specvirt_sc2010_perf.html

Intel Machine Vision Solution for Retail

- Edge/Fog-based Solution
  - CV-SDK
    - Support Network Models
  - Use Cases
    - Face detection & classification
    - Gesture recognition
    - Items recognition

alexnet	classification
densenet-121	
densenet-161	
densenet-169	
densenet-201	
faster-rcnn-vgg16	detection
faster-rcnn-zf	
place205-alexnet	scene recognition
place205-googlenet	
resnet-101	classification
resnet-152	
resnet-18	
resnet-50	
squeezenet1.0	
squeezenet1.1	
vgg16	
vgg19	
yolo-tiny-v1	detection
yolo-tiny-v2	
yolo-v2	
googlenet-v1	classification
googlenet-v2	
googlenet-v3	
fcn-alexnet	semantic segmentation
fcn-vgg16-8s	
fcn-vgg16-32s	
faster-rcnn-resnet	detection
ssd300	
ssd512	
dssd	
R-FCN-resnet-50	
R-FCN-resnet-101	
mtcnn-p	
mtcnn-r	
mtcnn-o	
icvface-0004	
icvnet-1.1	
icvface-0022	
icvface-0034	
icvface-0035	
icvface-0042	
icvface-0043	
pvanet-fr1	feature extraction
icv-age-gender	object attributes
icv-tsr-eu	classification
icv-tsr-us	
icv-tlr	
icv-reid-embedding	object attributes
icv-photo-transfer-googlenet	(blank)
icv-reid-distance	object attributes
icv-pc0026	detection
icv-lpr	ocr
icv-vechicle	detection
icv-facenet	
icv-mobilenet-ssd	
icv-resnet10-ssd	
icv-resnet10-sssd	
icv-headpose-cnt.1	head pose
pva-fd-s1-g	detection
pva-fd-s2-r	
pva-fd-s3-o	
icvface-0020	

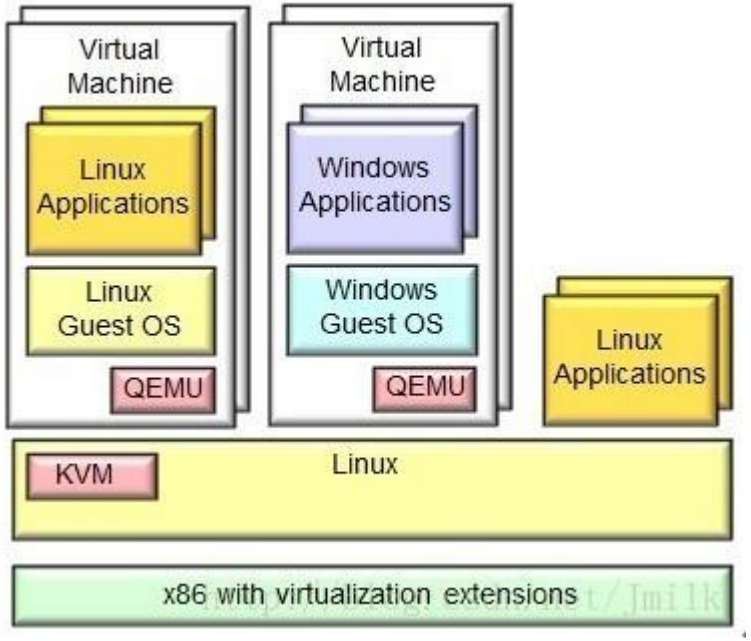
References

- http://blog.csdn.net/Jmilk/article/details/68947277?locationNum=6&fps=1
- http://blog.csdn.net/Jmilk/article/details/51853511?locationNum=2&fps=1
- http://www.cnblogs.com/echo1937/p/7338582.html
- https://codefresh.io/blog/debug_node_in_docker/
- https://www.digitalocean.com/community/tutorials/how-to-provision-and-manage-remote-docker-hosts-with-docker-machine-on-ubuntu-16-04

Backup

What is KVM?

- 一般所说的 KVM 是广义 KVM, 即: **一套 Linux 全虚拟化解决方案** , 下面是主要角色：
- KVM驱动模块: 在利用 Linux kernel 所提供的部分操作系统能力(如: 任务调度/内存管理/硬件设备交互)的基础上, 再为其加入了虚拟化能力, 使得 Linux kernel 具有了 **转化** 为 Hypervisor(虚拟化管理软件) 的条件，这个模块本身只能提供 CPU 和内存的虚拟化
- KVM 包含一个提供给 CPU 的 *底层虚拟化可加载核心模块* kvm.ko(kvm-intel.ko/kvm-AMD.ko)
- KVM 需要在具备 Intel VT 或 AMD-V 功能的 x86 平台上运行, 所以 KVM 也被称之为 **硬件辅助的全虚拟化实现**.
- 由于 *KVM 内核模块本身只能提供CPU 和内存的虚拟化*, 所以 KVM 需要一些额外的虚拟化技术组件来为虚拟机提供诸如 *网卡/ IO 总线/显卡*等硬件的虚拟化实现. 最终变成了我们所使用到的 **Linux 虚拟化解决方案**.



KVM虚拟化 = KVM内核模块 + /dev/kvm + QEMU

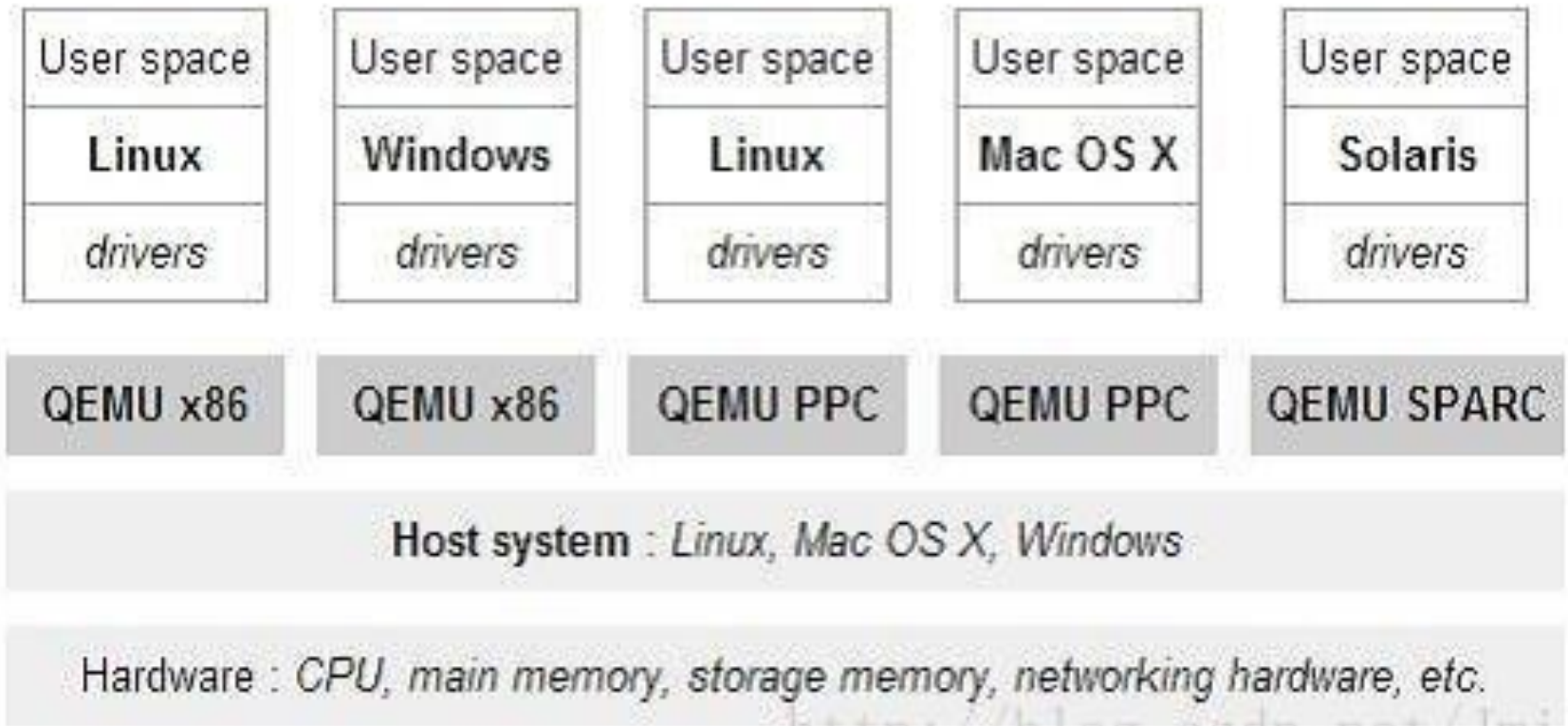
KVM

- KVM驱动模块: 在利用 Linux kernel 所提供的部分操作系统能力(如: 任务调度/内存管理/硬件设备交互)的基础上, 再为其加入了虚拟化能力, 使得 Linux kernel 具有了 转化 为 Hypervisor(虚拟化管理软件) 的条件，这个模块本身只能提供 CPU 和内存的虚拟化;
- KVM 包含一个提供给 CPU 的 *底层虚拟化可加载核心模块* kvm.ko(kvm-intel.ko/kvm-AMD.ko);
- KVM 需要在具备 Intel VT 或 AMD-V 功能的 x86 平台上运行, 所以 KVM 也被称之为 硬件辅助的全虚拟化实现;
- 由于 *KVM 内核模块本身只能提供 CPU 和内存的虚拟化*, 所以 KVM 需要一些额外的虚拟化技术组件来为虚拟机提供诸如 *网卡/ IO 总线/显卡*等硬件的虚拟化实现. 最终变成了我们所使用到的 Linux 虚拟化解决方案.

QEMU

- QEMU(Quick Emulator) 是一个广泛使用的开源计算机 仿真器和虚拟机. QEMU 作为一个独立 Hypervisor(不同于 KVM 需要嵌入到 kernel), 能在应用程序的层面上运行虚拟机. 同时也支持兼容 Xen/KVM 模式下的虚拟化.
  - 当 QEMU 作为仿真器时, QEMU 通过动态转化技术(模拟)为 GuestOS 模拟出 CPU 和其他硬件资源, 让 GuestOS 认为自身直接与硬件交互. QEMU 会将这些交互指令转译给真正的物理硬件之后, 再由物理硬件执行相应的操作. 由于 GuestOS 的指令都需要经过 QEMU 的模拟, 因而相比于虚拟机来说性能较差.
  - 当 QEMU 作为一个虚拟机时, QEMU 能够通过直接使用物理机的系统资源, 使虚拟机能够获得接近于物理机的性能表现.

QEMU

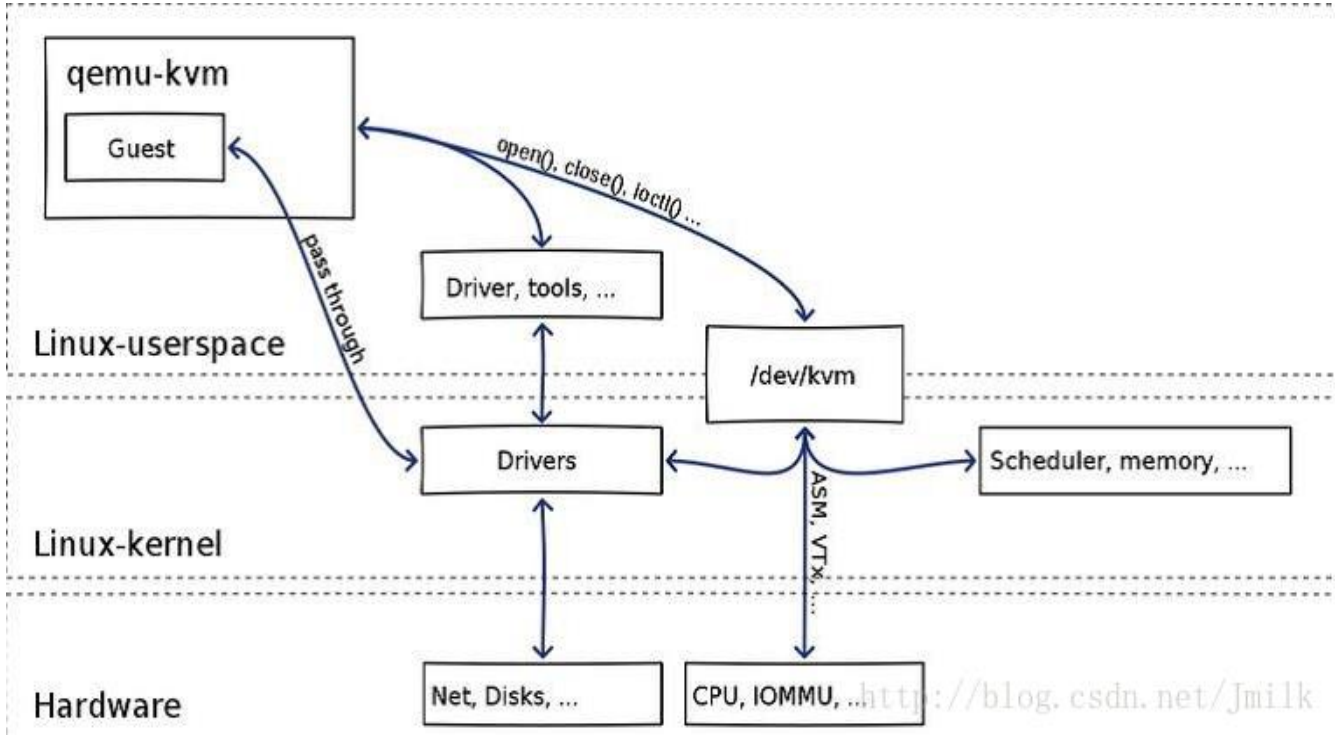


KVM与QEMU

- KVM 作为 Linux 的内核模块, 需要被加载后, 才能进一步通过其他工具的辅助以实现虚拟机的创建.
- QEMU 可以通过 KVM 对外暴露的 /dev/kvm 接口来进行调用, 官方提供的 KVM 下载有 QEMU 和 KVM 两大部分, 包含了 KVM 模块、QEMU 工具以及二者的合集 qemu-kvm 三个文件.
- KVM 负责提供 CPU 虚拟化和内存虚拟化, QEMU 负责提供硬件设备的虚拟化, 以此弥补来 KVM 的缺陷. 同时, 为了提高 QEMU 虚拟出来的虚拟硬件设备性能, 于是产生了 pass through 半虚拟化设备virtio_blk/virtio_net. KVM + QEMU 才能实现真正意义上虚拟化. qemu-kvm 就是将两者整合到了一起的qemu版本.

KVM与QEMU

- qemu-kvm 通过 ioctl 调用 KVM 的 /dev/kvm 接口, 将 KVM 内核模块相关的 CPU 指令传递到内核模块中执行.



libvirt

- Libvirt 是目前使用最为广泛的异构虚拟化管理工具及 API, 其具有一个 Libvirtd Daemon, 可供本地或远程的 virsh 调用.
- libvirt 由 应用程序编程接口库、libvirtd 守护进程、virsh CLI 组成. 其中 libvirtd 守护进程负责调度管理虚拟机, 而且这个守护进程可以分为 root 权限的 libvirtd和普通用户权限的 libvirtd 两种. 前者权限更大, 可以虚拟出物理主机的各种设备.

Virtualization Technologies

Reference To: http://csiewiki.crboy.net/embedded/xvisor

What is Virtualization?

- 虛擬化就是要建立出一個作業系統或伺服器的虛擬版本，可以把擁有的資源作妥當的管理分配

Hypervisor (Virtual Machine Monitor, VMM)

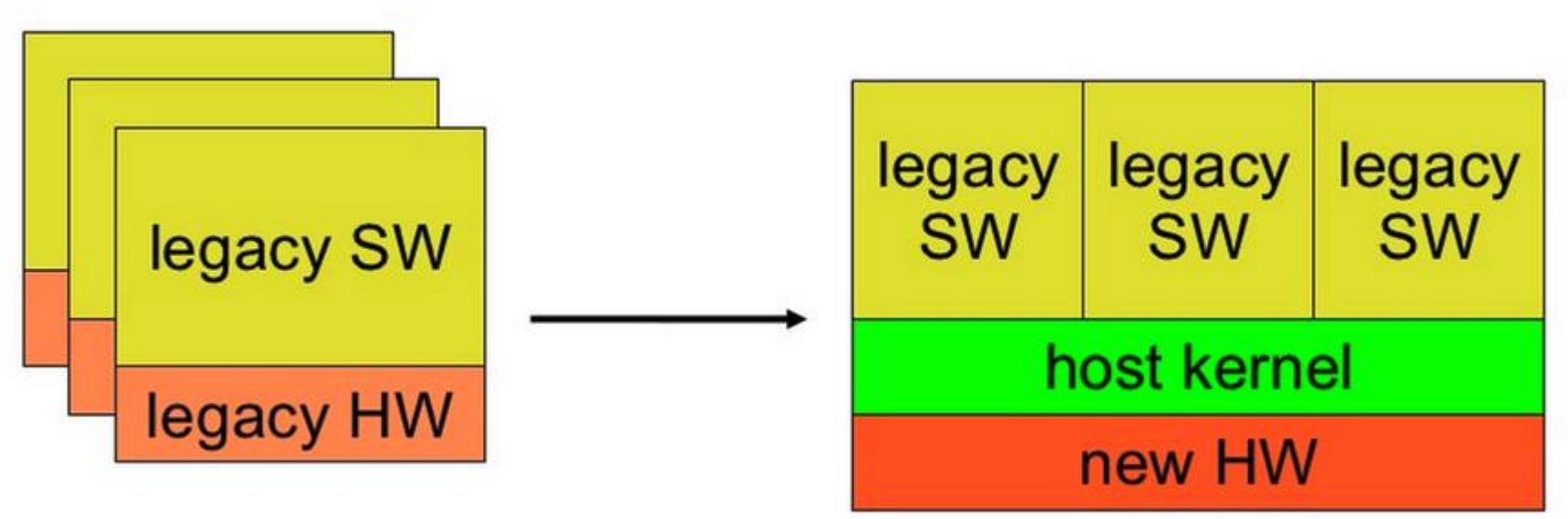
- Hypervisor的主要功用是去管理Virtual Machines (VMs)，使VM能夠各自獨立運行，同時Hypervisor也管理著底層硬體
  - Host Machine: 運行Hypervisor的實體主機，但有時運行在Hypervisor之上
  - Guest Machine: 運行在Hypervisor之上的虛擬主機
- 目前三大open source的hypervisor為Linux KVM, Xen, Xvisor

引入Hypervisor的用例及好处

- 工作負載整合 (Workload Consolidation)
  - 許多舊有的系統或程式在各自的硬體上，我們可以透過hypervisor的分配，共同使用硬體資源
  - 以VMware資料顯示，如果透過VMware vSphere來進行伺服器的虛擬化，可以降低硬體和營運成本達 50%，並減少能源成本達 80%，且每完成一個虛擬化伺服器工作負載，每年可節省的成本超過 $3,000 美元。最多可減少 70% 佈建新伺服器所需的時間。

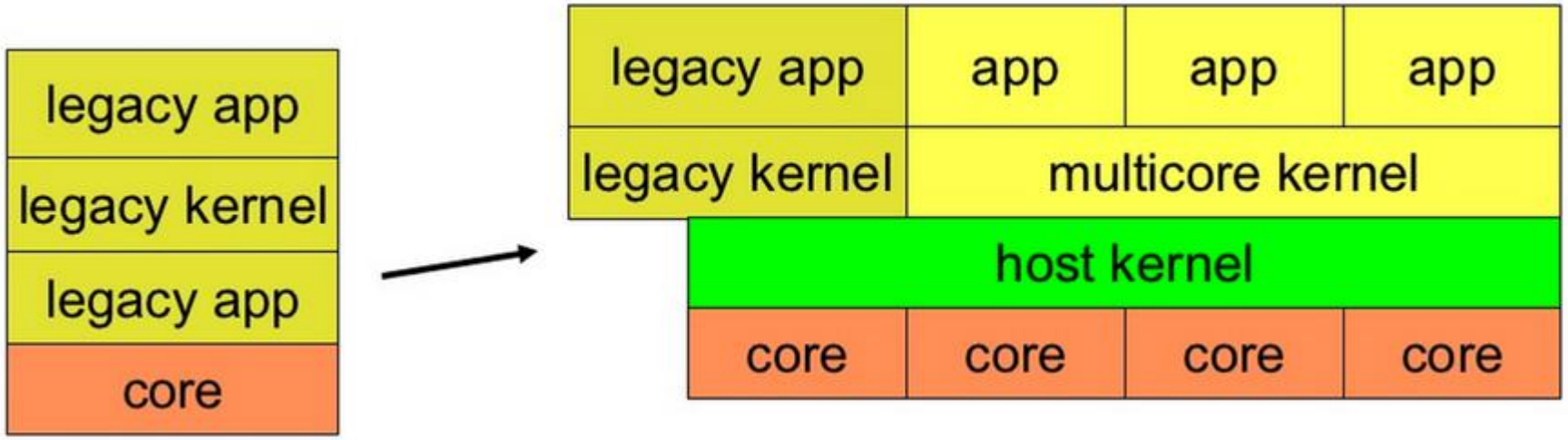
引入Hypervisor的用例及好处

- 支援舊有軟體 (Legacy Software)
  - 老舊過時的軟硬體透過虛擬化移植到新的硬體或OS上
  - App运行在不同的kernel上，可透过KVM(Kernel-based Virtual Machine)整合并切换kernel来运行app，为Bare-Metal概念



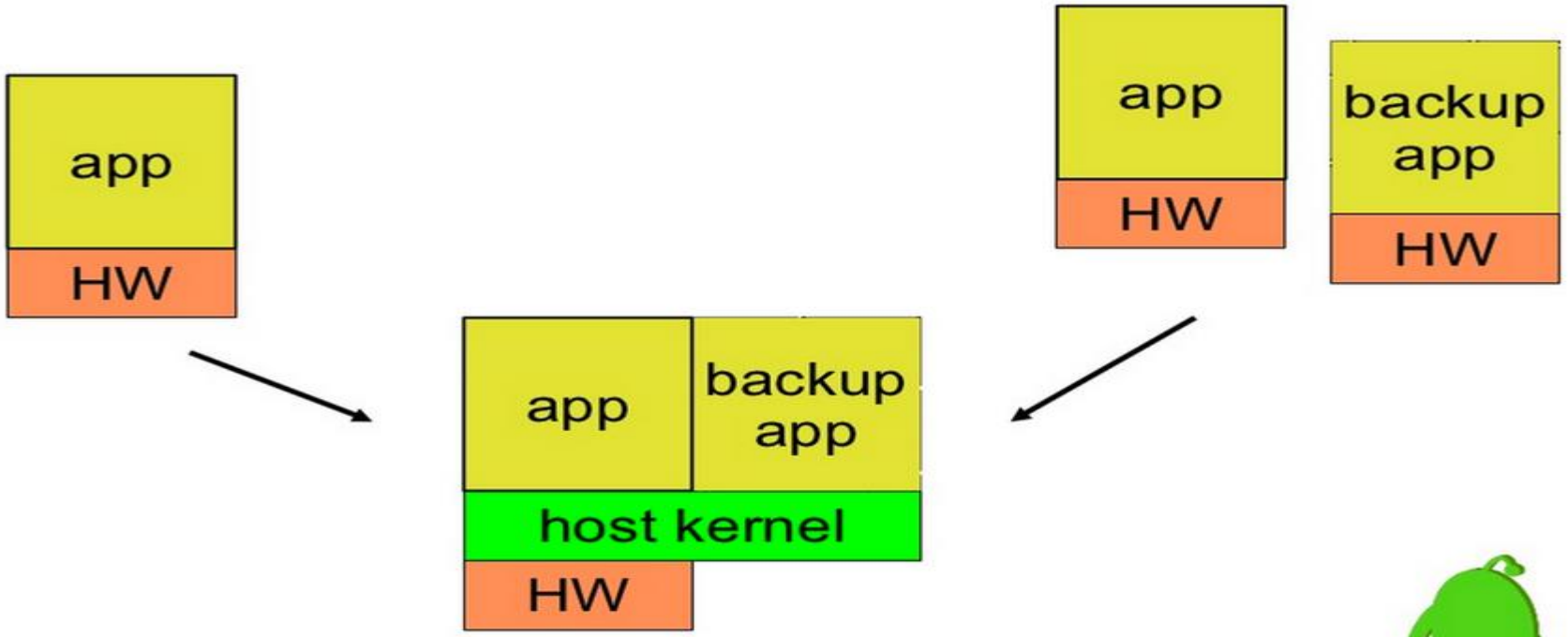
引入Hypervisor的用例及好处

- 啟用多核 (Multicore Enablement)
  - 原本舊有的APP及kernel運行在單一core上，透過虛擬化，我們可以移植到有多個core的架構，並且透過host kernel去運行



引入Hypervisor的用例及好处

- 提高可靠性 (Improved Reliability)
  - 因有host kernel，我們不需要太多的硬體

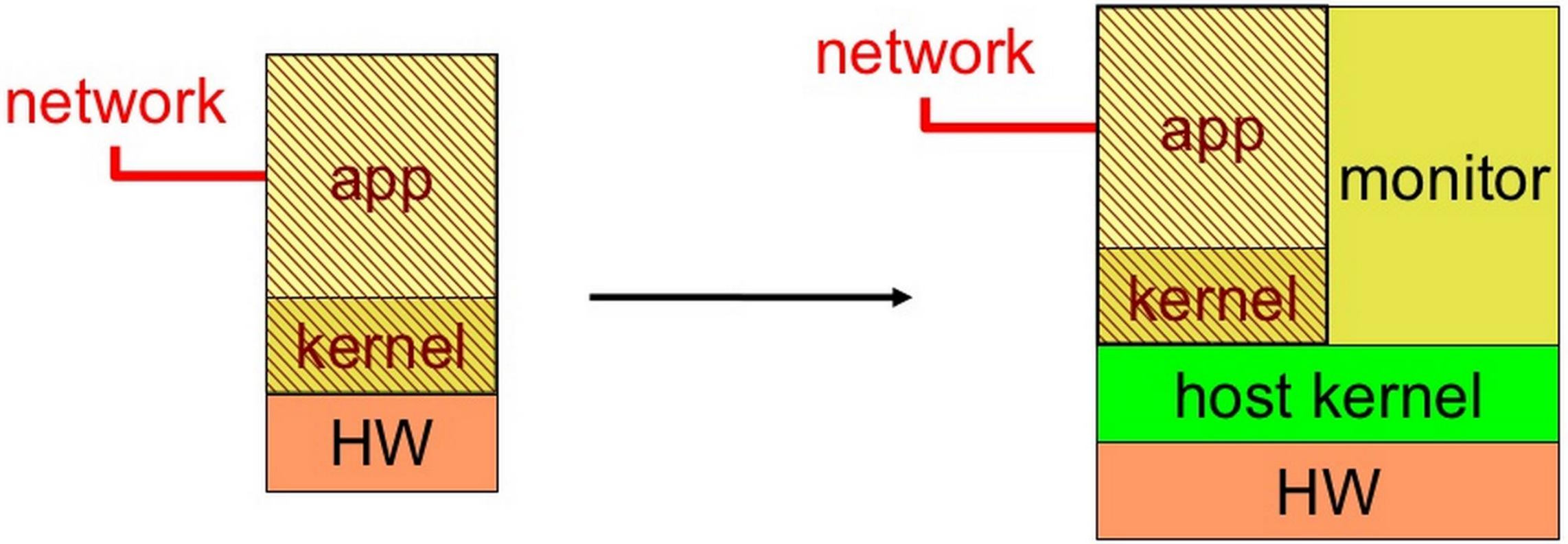


引入Hypervisor的用例及好处

- 安全監控 (Secure Monitoring)
  - ARM SMC(secure monitor call) to secure monitor mode
  - kernel等級或者是rootkits的攻擊通常都是在執行擁有特權(privilege)模式的時候發生的，而透過虛擬化，我們擁有更高權限的hypervisor去控制memory protection或程式的排程優先順序
  - 最主要我們是透過hypervisor去把security tools也就是monitor孤立出來，不會把monitor與untrusted VM放在一起，並把他們分別移到trusted secure VM，再透過 introspection 去掃描或觀察這些程式碼
  - source: http://research.microsoft.com/pubs/153179/sim-ccs09.pdf (主要探討Secure in VM-monitor)
  - source: https://www.ma.rhul.ac.uk/static/techrep/2011/RHUL-MA-2011-09.pdf
  - In-VM 有較好的performance，Out-of-VM則有較好的安全性

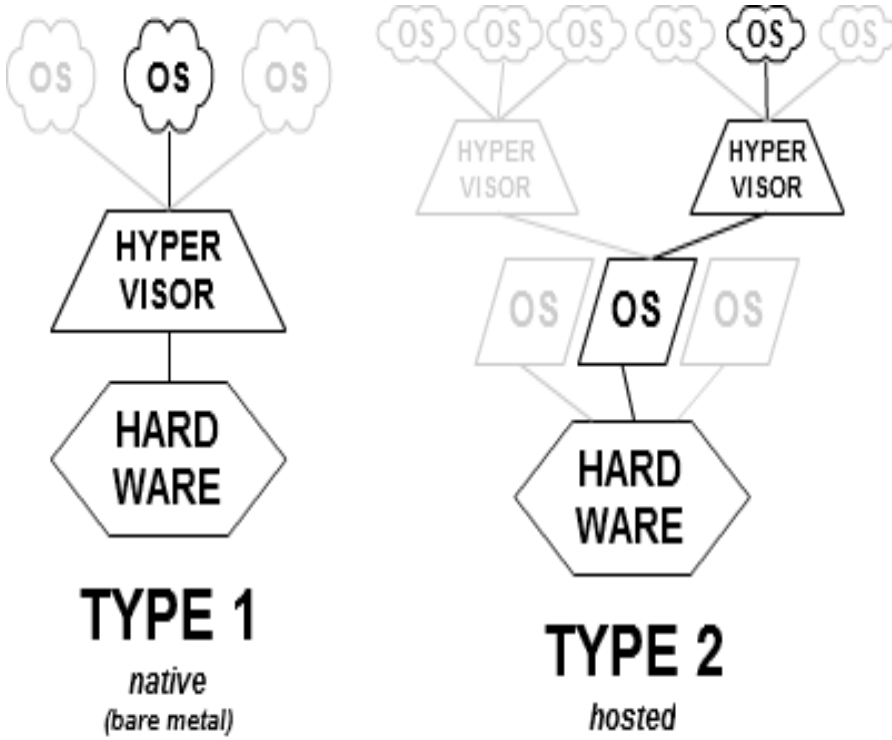
引入Hypervisor的用例及好处

- 安全监控 (Secure Monitoring)



Hypervisor的類型

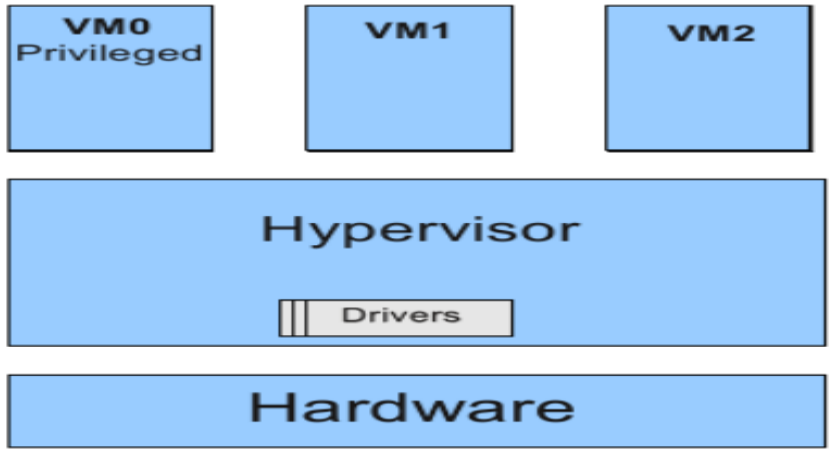
- Type-1: 本地(native)、裸機(bare-metal) hypervisors
  - Hypervisor直接運行在host的硬體上，並直接控制硬體及管理Guest作業系統，此時host把Guest作業系統當成一個process
    - 如: XenServer 、 Hyper-V 、Xvisor
- Type-2: 託管(hosted) hypervisors
  - Hypervisor運行在host的作業系統上，再去提供虛擬化服務
    - 如: VMware、VirtualBox



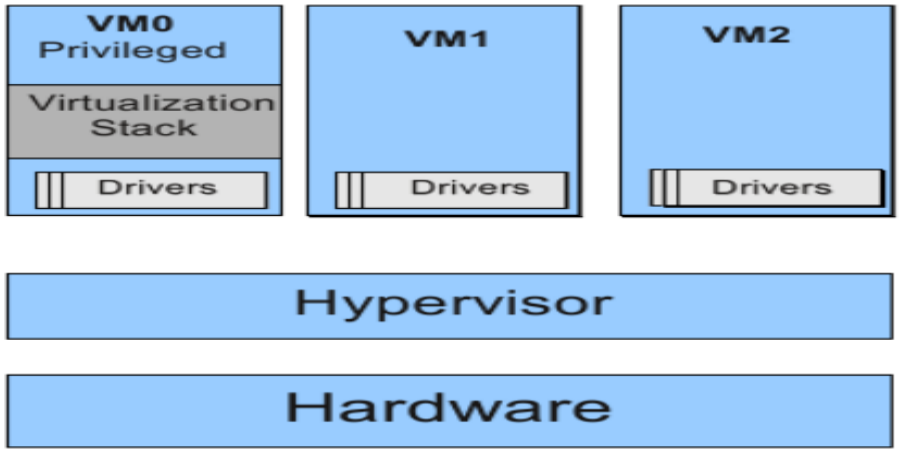
Hypervisor的類型

- Type-1 與 Type-2 比較Type-1
  - 能直接與硬體溝通，有較高的安全性(掌握在hypervisor上)，適合server虛擬化
  - 但因直接運行於硬體上，所以時常為single purpose
  - Type-1又有分兩種設計
    - monolithic
      - device driver包含在hypervisor(也就是kernel space)，因為沒有任何的intermediation，所以application及hardware間有良好的performance，但hypervisor會有大量的driver code，容易被攻擊
        - Complete monolithic
          - 此hypervisor有個software負責host的硬體存取、CPU的虛擬化及guest的IO emulation，Xvisor屬於此類
        - Partially monolithic
          - hypervisor為GPOS的附加套件(extension)，與complete monothlic不同的是，是利用user-space software，例如QEMU來支持host的硬體存取、CPU的虛擬化及guest的IO emulation，KVM屬於此類
    - microkernel
      - hypervisor為輕小的micro-kernel，利用Management Guest(例如Xen的Dom0)去提供host的硬體存取、CPU的虛擬化及guest的IO emulation
      - device driver被安裝在parent guest(也就是user space). Parent guest是privileged VM，去管理non-privileged child guest VMs. 當Child guest要存取硬體資源時，必須經過parent guest，雖然performance下降，但較monolithic安全，Xen則屬於此類
- Type-2
  - 支援較多的I/O device及服務
  - 因需多透過Host OS，效能較Type-1低，常應用於效率較不重要的客戶端

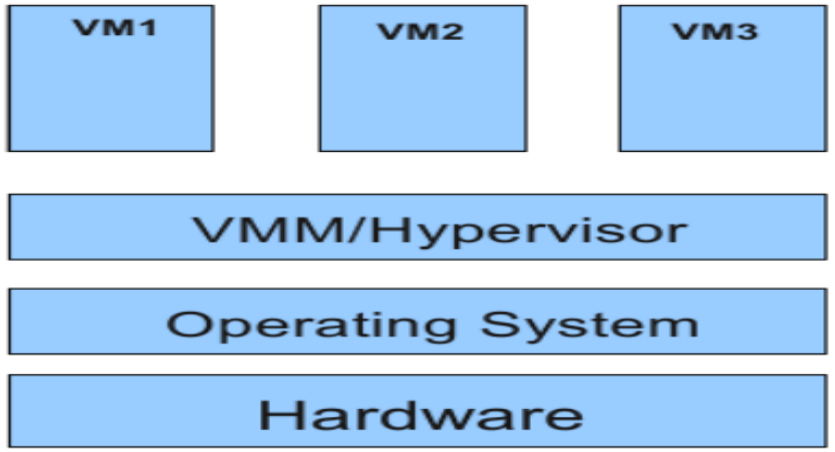
Hypervisor的類型



a) Type I Hypervisor (Monolithic)



b) Type I Hypervisor (Microkernel)



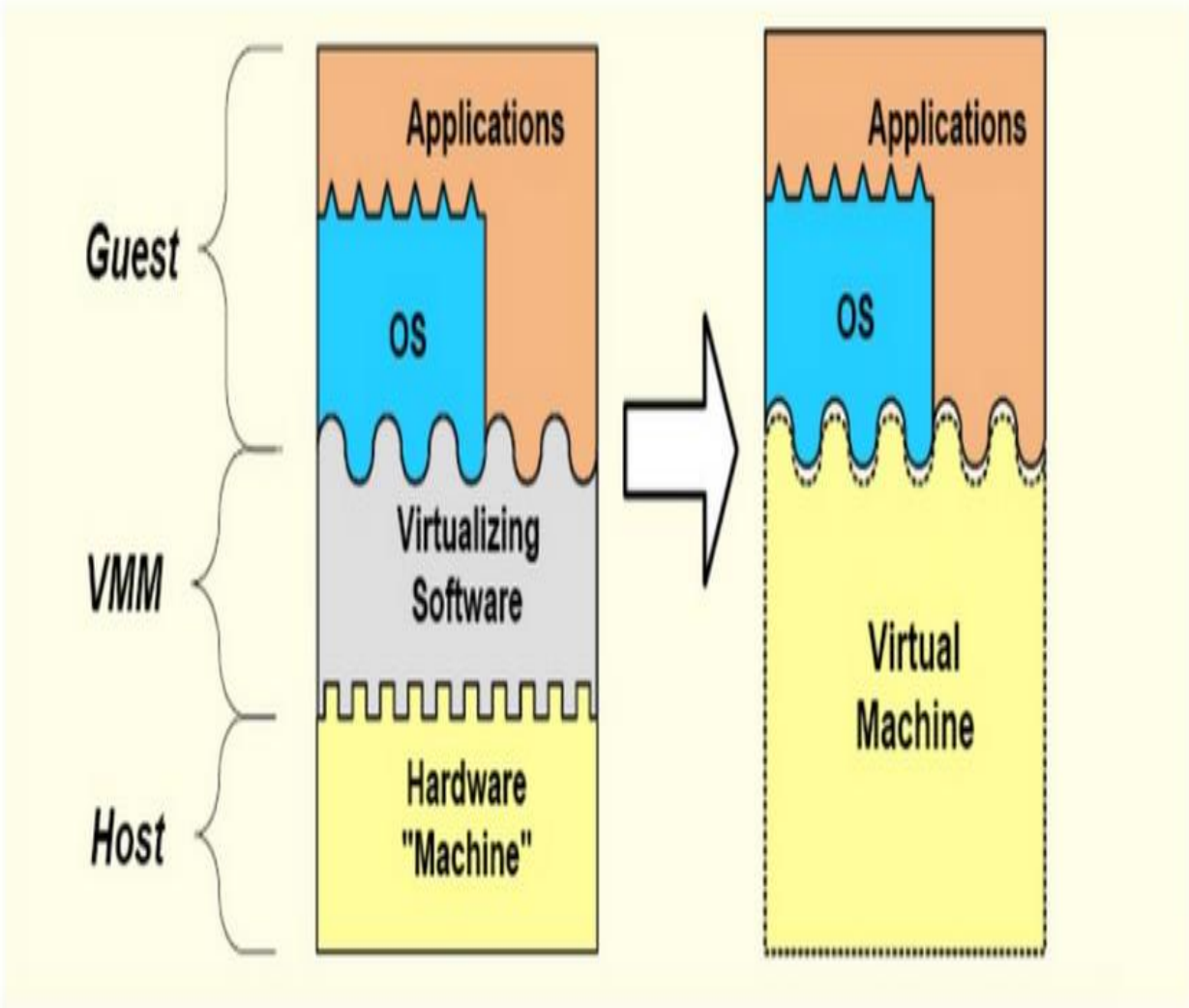
c) Type II Hypervisor

Hypervisor的類型

- Source: https://www.ma.rhul.ac.uk/static/techrep/2011/RHUL-MA-2011-09.pdf
- (補充)Type-0
  - 由於Type-1及Type-2過於複雜也不易在嵌入式系統作設定，因此美國軟體公司LynuxWorks在2012年提出Type-0 Hypervisor，此Hypervisor不需要kernel或作業系統，但有人提出這不可能實現。
- source: http://en.wikipedia.org/wiki/Hypervisor
- source: http://www.lynx.com/whitepaper/the-rise-of-the-type-zero-hypervisor
- source: http://www.virtualizationpractice.com/type-0-hypervisor-fact-or-fiction-17159

虛擬化定理

- Popek and Goldberg virtualization requirements, 1974
- Gerald Popek及Robert Goldberg二人提出的理論至今仍是非常便利的方法去決定計算機架構是否支持虛擬化，也是設計虛擬化計算機架構的導則
- VM定義
  - A virtual machine is taken to be an efficient, isolated duplicate of the real machine. It is the environment created by the virtual machine monitor.
- VMM定義
  - VMM is the piece of **software** that provides the abstraction of a virtual machine



虛擬化定理

- 當分析VMM建立的環境時，需要滿足以下三點性質：等價性 / 忠實(Equivalence / Fidelity)
  - 一個在VMM下運行的程式的所有行為應該要和原本直接運行於相同機器上一樣
  - The VMM provides an environment for programs which is **essentially identical** with the original machine
- 資源控制 / 安全性(Resource control / Safety)
  - VMM應該要有虛擬資源的完整控制權.
  - The VMM is in complete control of system resources.
- 效率性(Efficiency / Performance)
  - 常被執行的主要機器指令集部分不應該被VMM干預，而是直接被處理器執行
  - It demands that a statistically dominant subset of the virtual processor’s instructions be executed directly by the real processor, with no software intervention by the VMM.

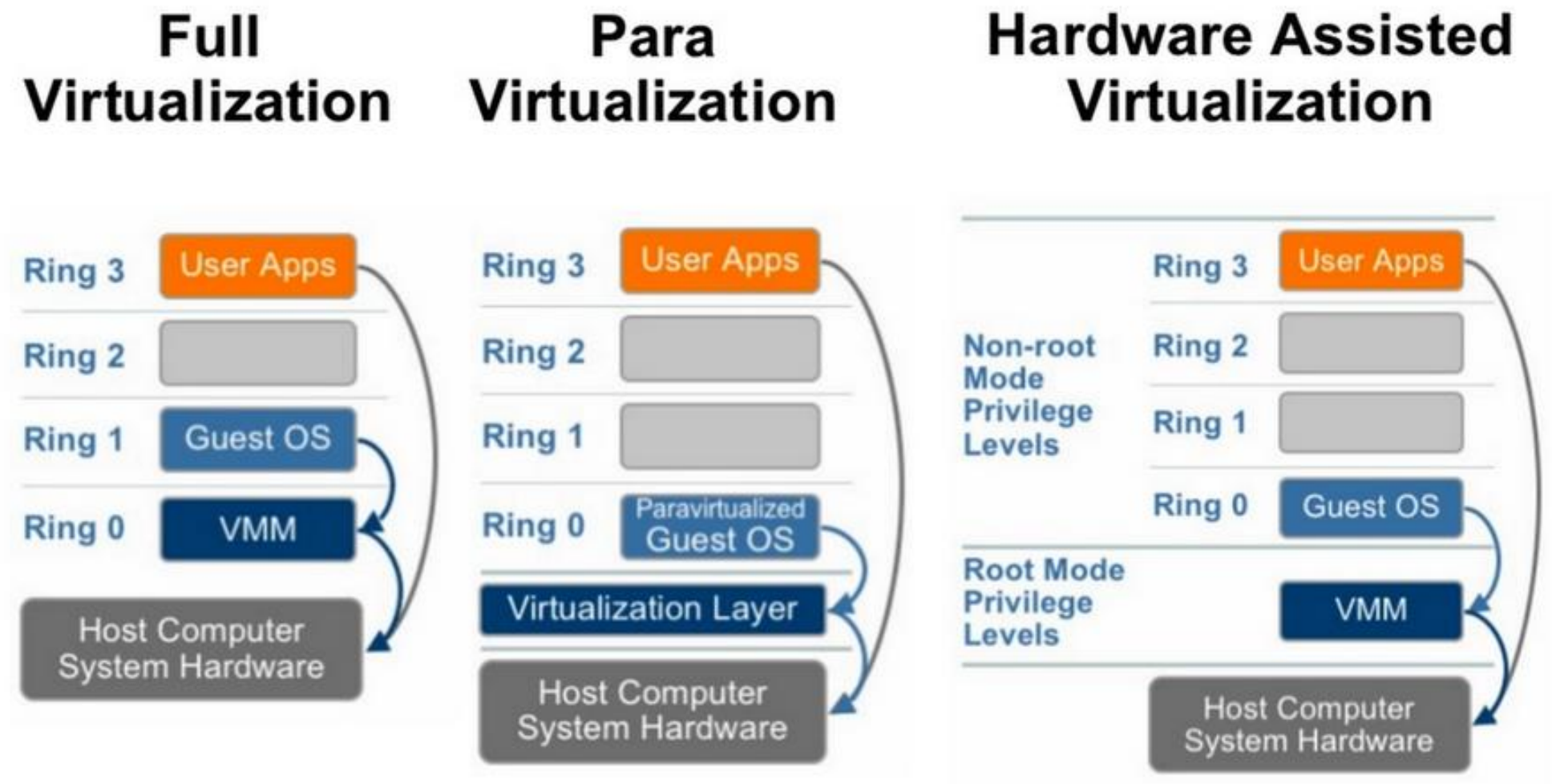
虛擬化定理

- 為了導出Virtualization theorem虛擬化理論，二人把ISA分成三個類型：(Third Generation Computer)
  - 優先級指令 (Privileged instructions)
    - 必須在擁有足夠的privilege下執行
    - 當處理器在user mode的時候，會觸發Trap
    - 當處理器在supervisor mode的時候，不會觸發Trap
    - Privileged instructions are independent of the virtualization process. They are merely characteristics of the machine which may be determined from reading the principles of operation.
    - 指令例子
      - Access I/O devices: Poll for IO, perform DMA, catch hardware interrupt
      - Manipulate memory management: Set up page tables, load/flush the TLB and CPU caches, etc.
      - Configure various “mode bits”: Interrupt priority level, software trap vectors, etc.
      - Call halt instruction: Put CPU into low-power or idle state until next interrupt
  - 控制敏感指令 (Control sensitive instructions)
    - 指令企圖去改變系統資源配置 (也就是先前提到，VMM擁有系統資源完整掌控權)
    - 指令在不經過memory trap sequence下影響處理器模式
    - It attempts to change the amount of (memory) resources available, or affects the processor mode without going through the memory trap sequence
  - 行為敏感指令 (Behavior sensitive instructions)
    - 指令根據資源配置(relocation-bound register的內容或處理器的模式)而有不同的行為結果
    - Location sensitive: load physical address (IBM 360/ 67 ERA)
    - Mode sensitive: move from previous instruction space (DeC PDP-11/45 MVPI)
- 所以根據定義，ISA如果是sensitive，就是control sensitive或者是behavior sensitive，不然就是innocuous (無害的)。另外，如果sensitive instruction為privileged，VMM則可以被覆寫

虛擬化定理

- VMM內容
  - VMM是個軟體，在此我們稱作control program，在此program中，又包含許多modules
  - control program modules可概略分成三種
    - Dispatcher
      - 可視為control module之上最高等級的control program，Dispatcher決定呼叫哪一個module
    - Allocator
      - 決定系統資源的分配，有resource table
      - 在VMM只有host一個VM的情況下，allocator只需要把VM跟VMM分開
      - 在VMM host許多VM的情況下，allocator需要避免同時把相同資源分配給VM
      - 在VM環境下執行會改變系統資源的privileged instruction，dispatcher會去啟動allocator
      - 例子：reset relocation-bounds register.
    - interpreter
      - 一個instruction，一個interpreter routine
      - 主要功用就是simulate the effect of the instruction
- Popek and Goldberg 最後提出的理論
  - Theorem 1. 對於傳統的第三代計算機，如果sensitive instruction為privilege instruction的subset，就可建立VMM
    - 更直觀點來說，當sensitive instruction影響到VMM時，會發生trap讓VMM能夠擁有完整的控制權，這樣保證前面提到的資源控制(resource control)
    - 原本由OS在kernel mode執行的敏感指令，因OS被移到user mode而無法正常執行，所以需要被trap給 hypervisor來執行。
  - Theorem 2. 一個傳統的第三代計算機是遞迴虛擬的，要符合以下兩個條件
    - 他是虛擬的
    - 新建立的VMM並沒有時間依賴性(timing dependencies)
- 論文文獻：http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4815&rep=rep1&type=pdf
- Source: http://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_requirements
- 補充：Five Generation Computer

實現虛擬化的方式

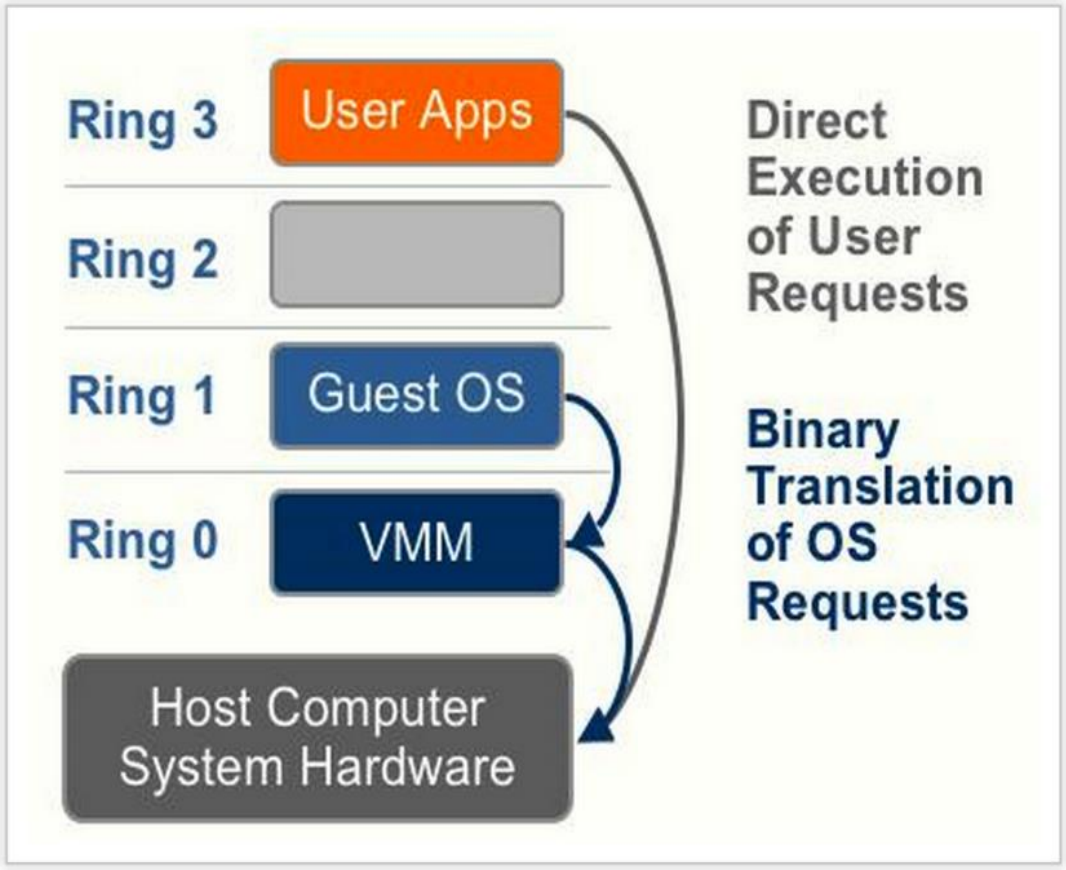

Full Virtualization
Ring 3
User Apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host Computer System Hardware
Para Virtualization
Ring 3
User Apps
Ring 2
Ring 1
Ring 0
Paravirtualized Guest OS
Virtualization Layer
Host Computer System Hardware
Hardware Assisted Virtualization
Non-root Mode Privilege Levels
Ring 3
User Apps
Ring 2
Ring 1
Ring 0
Guest OS
Root Mode Privilege Levels
VMM
Host Computer System Hardware

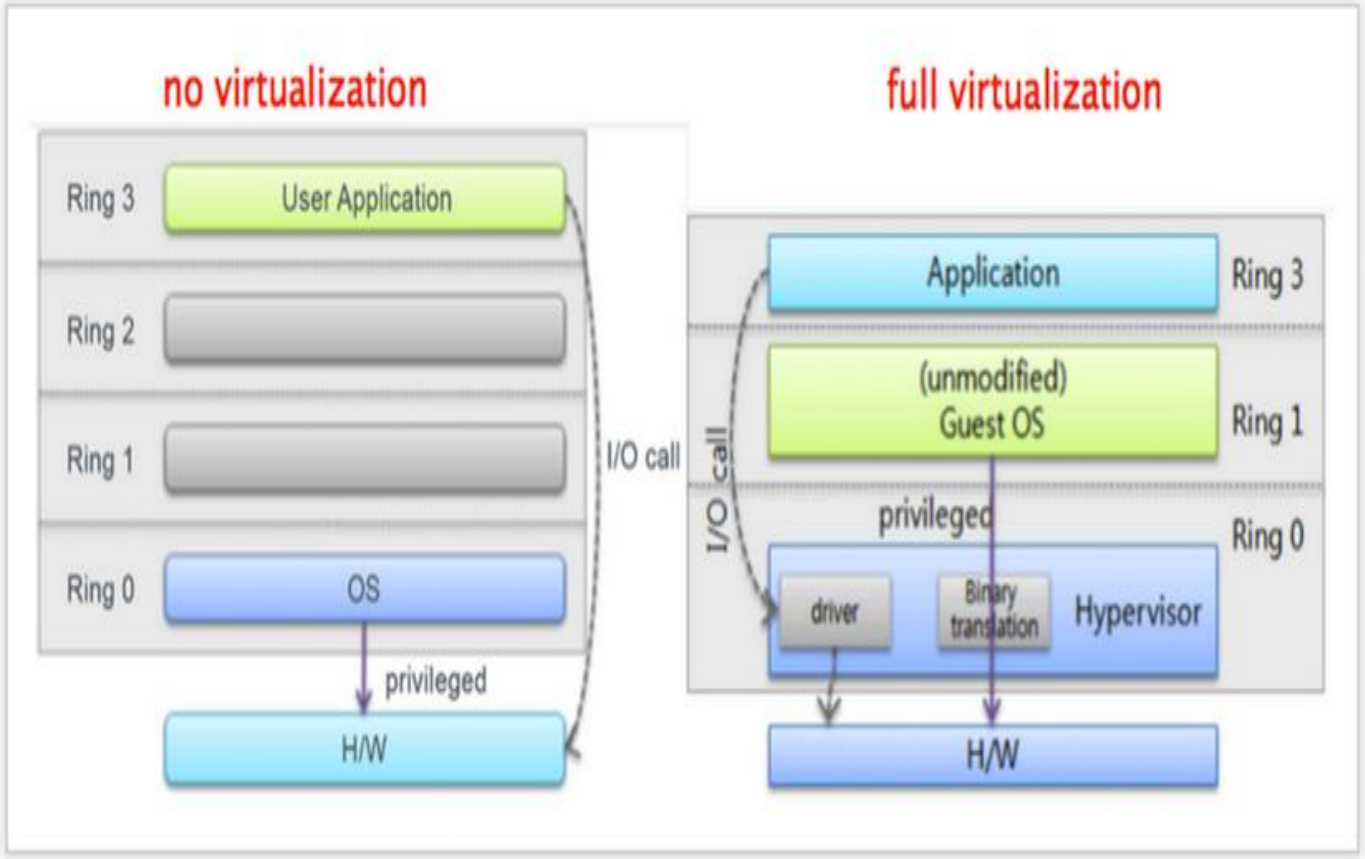
實現虛擬化的方式

- Full-Virtualization (i.e.VMWare, xen, xvisor, kvm)
  - 全部都虛擬化(I/O, interrupt, 指令集等等)，不需要硬體或OS的協助，而是透過軟體來模擬虛擬的硬體環境，直接透過Hypervisor實現
  - 不用修改OS核心，OS不知自己運行在虛擬化的環境下
  - Guest OS privilege operation需被Hypervisor intercept(攔截)，之後經過Binary Translation為machine code，以彌補x86的缺陷(only 17 instructions on privileged level)，故效能較Para-Virtualization差

實現虛擬化的方式

- Full-Virtualization (i.e.VMWare, xen, xvisor, kvm)



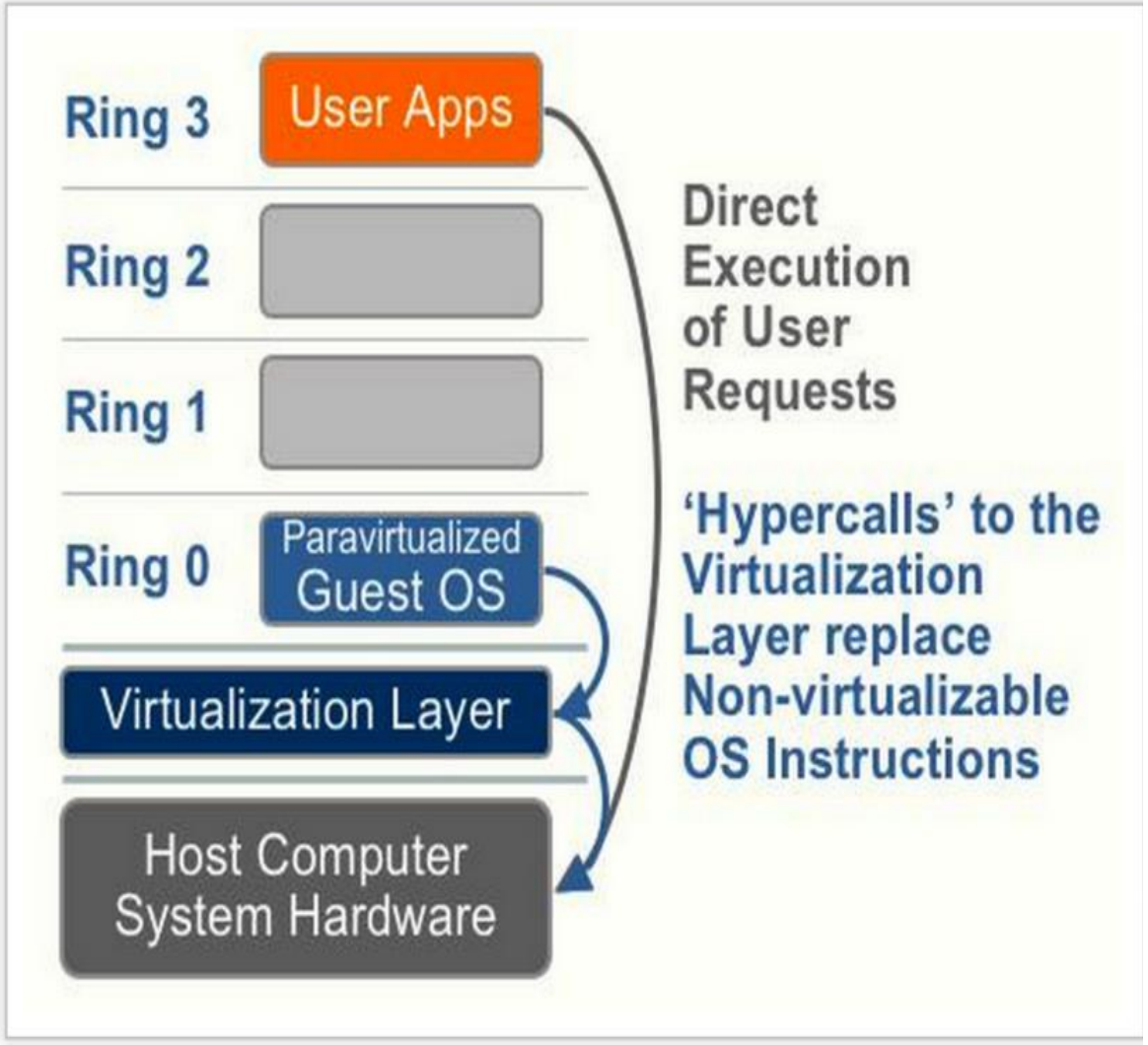


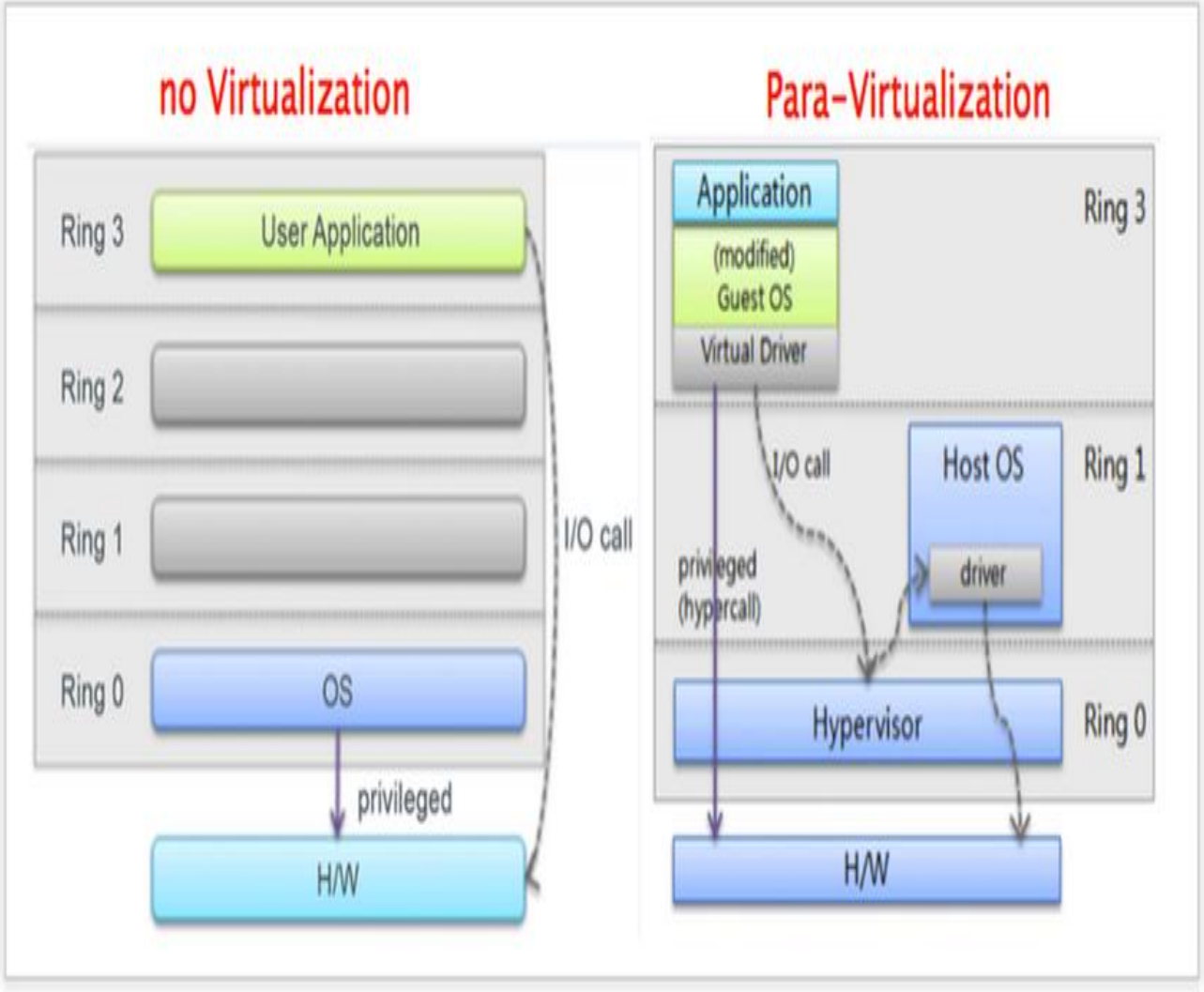
實現虛擬化的方式

- Para-Virtualization (i.e.Xen, xvisor, kvm)
  - 必須要修改OS Kernel，並且增加Hypercall
  - 由於某些Non-Virtualizable OS指令不能被hypervisor trap，所以Guest OS利用Hypercall呼叫Hypervisor對缺少的指令進行更換，這也是為何要修改OS的核心
  - Guest OS知道自己活在虛擬化的環境下，Guest OS知道有其他Guest OS的存在，並且看到的都是真實的硬體，Host OS則不用模擬CPU，Guest OS自行互相調配
  - 而在Hypervisor之上，還跑著一個Host OS(Dom0 in Xen)，是用來管理Hypervisor的，並且利用native OS的方法管理硬體上的所有I/O driver，所以hypervisor不作硬體模擬，而Guest OS不需要driver，可以直接找Host OS

實現虛擬化的方式

- Para-Virtualization (i.e.Xen, xvisor, kvm)



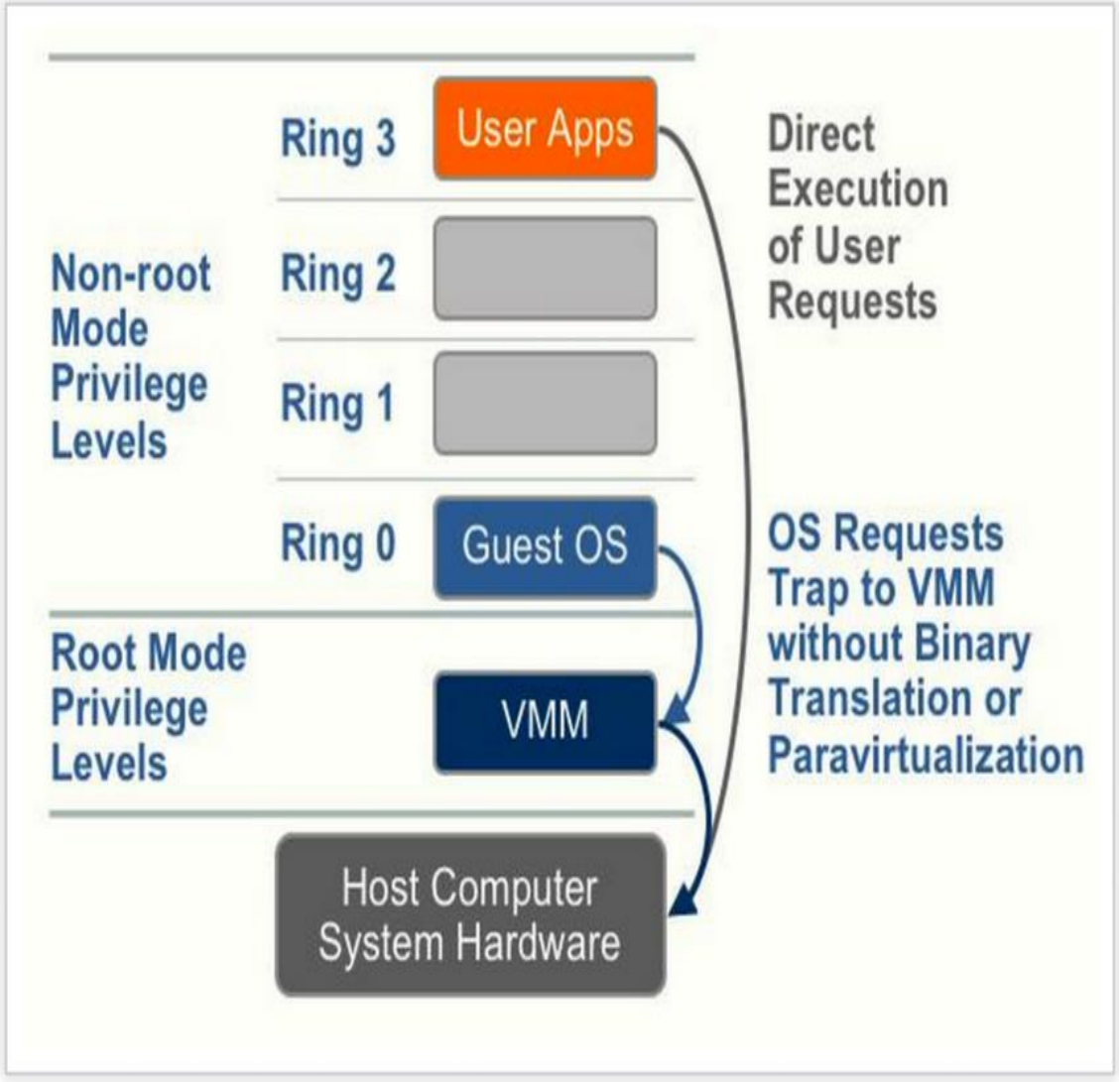


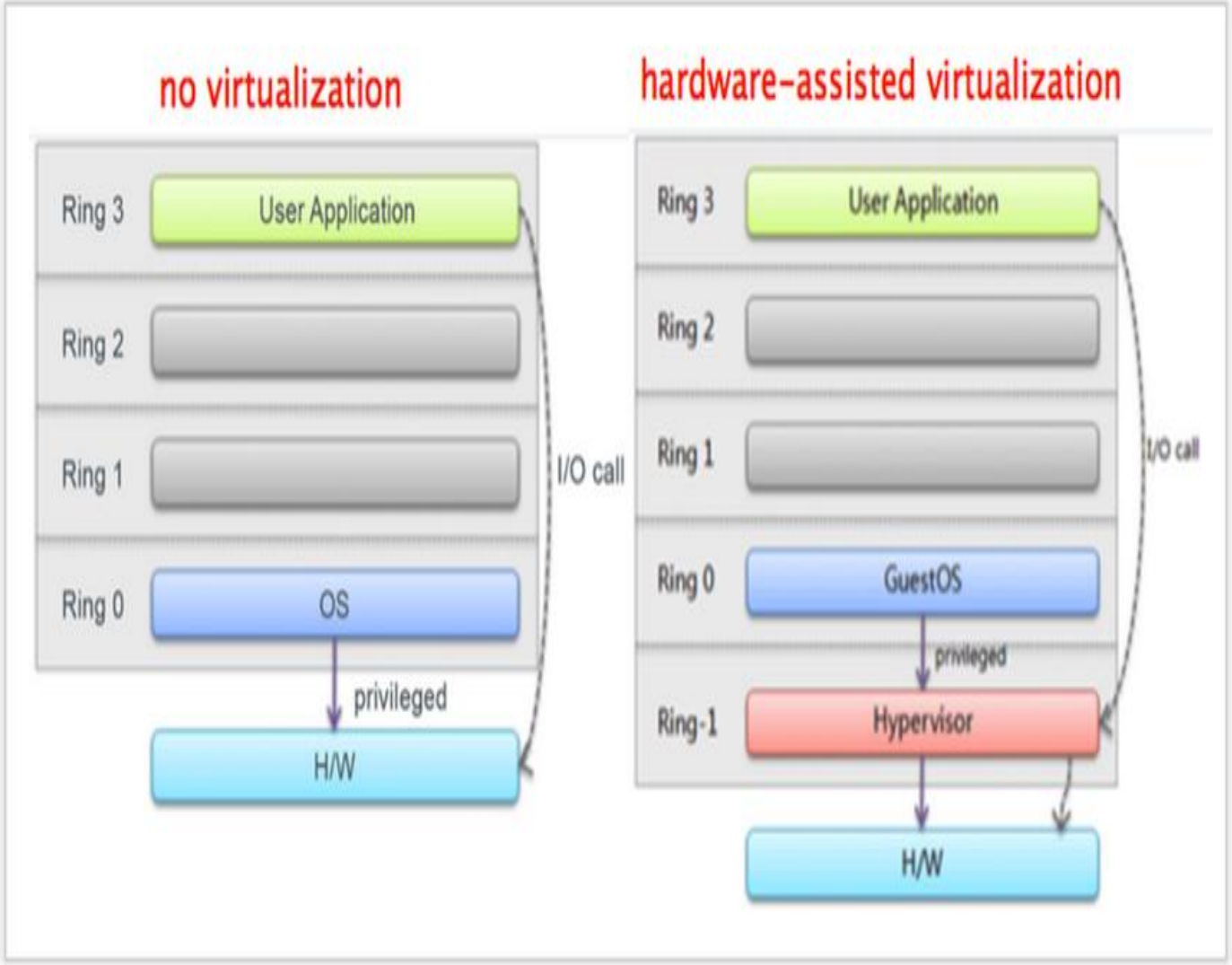
實現虛擬化的方式

- Hardware-assisted virtualization (Intel VT, AMD-V)
  - 需要硬體的支援虛擬化
  - 引入新的指令和處理器的運行模式，讓Hypervisor及Guest OS運行在不同的mode下，OS在被監視模式下運行，需要Hypervisor介入時，則透過hardware進行切換
  - 讓Guest OS發出的privilege指令被hypervisor攔截到
  - Intel VT及AMD-V增加Ring -1並提供Hypervisor x86虛擬化指令，使Hypervisor可以去存取Ring 0的硬體，並把Guest OS放在**Ring 0**，使Guest OS能在不影響其他guest或host OS的情況下，直接執行Ring 0的指令，所以不需要作Binary Translation也不需要修改OS Kernel

實現虛擬化的方式

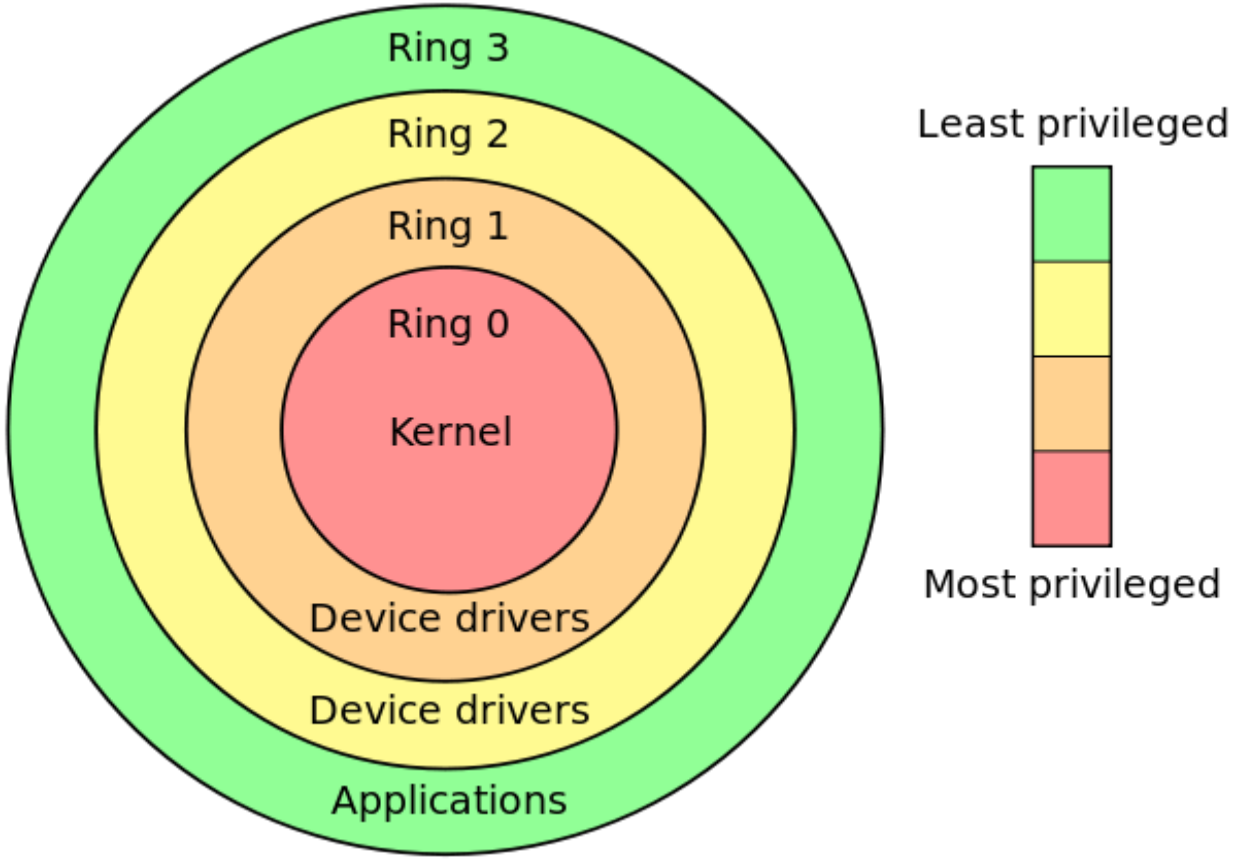
- Hardware-assisted virtualization (Intel VT, AMD-V)





實現虛擬化的方式

- Ring介紹
  - 此機制是為了保護資料或防止錯誤
  - x86結構中有四種級別Ring 0 ~ Ring 3
    - Ring 0最高，有著OS Kernal
    - Ring 3最低，也就是user mode，通常Application都在此層執行
    - Ring 1及Ring 2供驅動程式使用，很少用到
    - 權力最高不能直接控制權力最低，反之亦然，彼此是利用call gate溝通(提高CPL)，但有例外請閱此P14



Source: http://www.csie.ntu.edu.tw/~wcchen/asm98/asm/proj/b85506061/chap3/privilege.html
Source: http://tonysuo.blogspot.tw/2013/01/virtualization-3full-para-hardware.html

虛擬化指令(ARM)

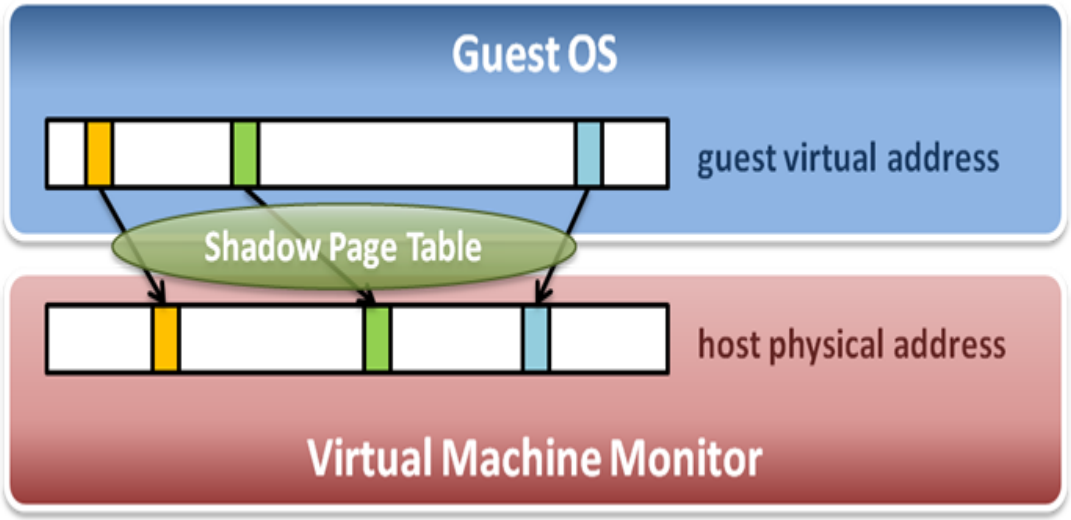
- 問題指令(Problematic Instructions)
  - Type I: 在user mode執行會產生未定義的指令異常
    - MCR、MRC: 需要依賴協處理器(coprocessor)
  - Type II: 在user mode執行會沒有作用
    - MSR、MRS: 需要操作系統暫存器
  - Type III: 在user mode執行會產生不可預測的行為
    - MOVS PC, LR: 返回指令，改變PC並跳回user mode，在user mode執行會產生不可預測的結果
  - ARM 的敏感指令:
    - 存取協處理器: MRC / MCR / CDP / LDC / STC
    - 存取SIMD/VFP 系統暫存器: VMRS / VMSR
    - 進入TrustZone 安全狀態: SMC
    - 存取 Memory-Mapped I/O: Load/Store instructions from/into memory-mapped I/O locations
    - 直接存取CPSR: MRS / MSR / CPS / SRS / RFE / LDM (conditional execution) / DPSPC
    - 間接存取CPSR: LDRT / STRT – Load/Store Unprivileged (“As User”)
    - 存取Banked Register: LDM / STM

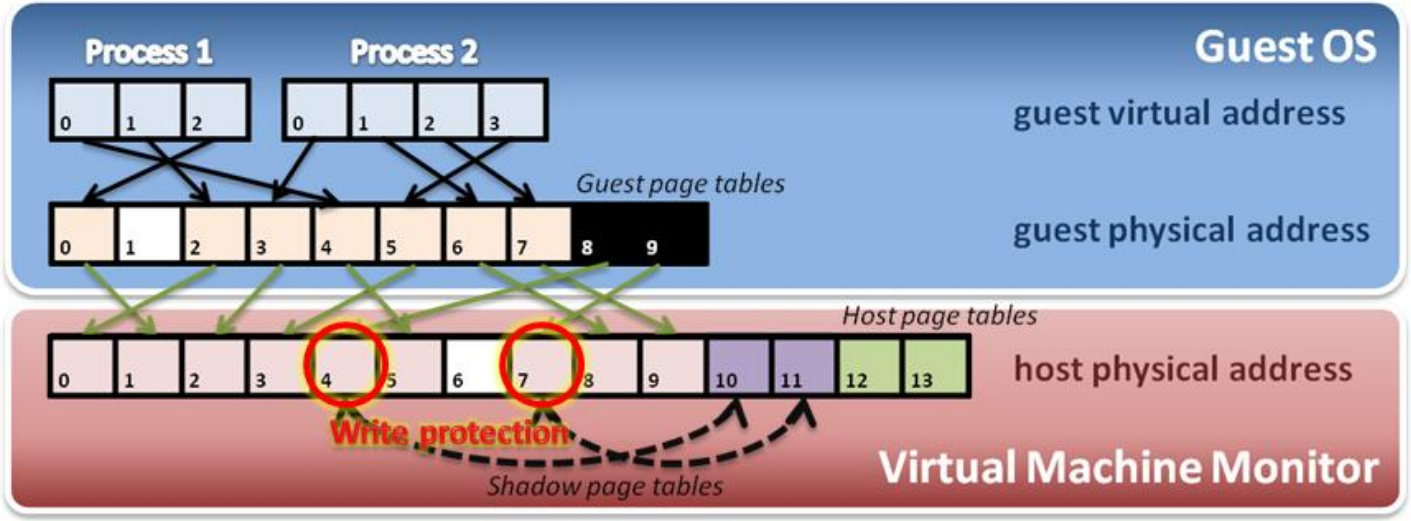
虛擬化指令(ARM)

- Solutions
  - 軟體技術: trap and emulate
    - Dynamic Binary Translation
      - 把問題指令取代為hypercall以進行trap 及emulate
    - Hypercall
      - 對type and original instruction bits進行編碼
      - trap 到 hypervisor，進行解碼及模擬指令
  - 硬體技術:
    - 特權指令轉換(待補)
    - MMU強制執行trap
    - 虛擬化擴充

虛擬化指令(ARM)

- 記憶體虛擬化(without hardware support)
  - Shadow page tables:
    - Map guest virtual address to host physical address
    - Guest OS maintain自己的page table到 guest實體記憶體框架
    - hypervisor 把所有的guest實體記憶體框架 map 到host實體記憶體框架





- 為每一個guest page table 建立 Shadow page table
- hypervisor要保護放著guest page table的host frame

ARM Virtualization Extensions

- ARMv8
  - Source:

    http://wiki.csie.ncku.edu.tw/embedded/ARMv8#%E8%99%9B%E6%93%AC%E5%8C%96-virtualization

ARM Virtualization Extensions

- CPU virtualization
  - ARM 增加運行在Non-secure privilege level 2 的 Hypervisor mode
  - CPU 虛擬化擴充
    - Guest OS kernel執行在EL1，userspace執行在EL0
    - 使大部分的敏感指令可以本地執行(native-run)在EL1上而不必trap及emulation
    - 而仍需要trap的敏感指令會被trap到EL2 (hypervisor mode HYP)
      - Guest OS's Load/Store
      - 會影響其他Guest OS的指令
      - Hypervisor Syndrome Register(HSR) 會保存被trapped的指令的資訊，因此hypervisor就能emulate它
  - Xvisor- cpu_vcpu_helper.c: