SE 4455 – Cloud Computing

Introduction to Virtualization

Dr. Jagath Samarabandu

jagath@uwo.ca

TEB 351

519-661-2111 x80058

Virtualization

- A mapping layer at its most basic form
- Exposes a set of interfaces and resources
- Maps these interfaces and resources to the underlying system
- Origins
  - Virtual memory (late 50's)
  - Multi-processing – virtual run-time environment
  - Multi-users – virtual login environment

Characteristics of Virtualization

- Abstraction – allows simplifying the complexity of the underlying system
- Replication – allows creation of multiple instances of a given set of interfaces and resources
- Isolation – activities and data of one instance is isolated from other instances

Levels of Virtualization

- At an application level – multi-tenancy
- At OS level – containers
- At hardware level – virtual machines
- Virtualization in other areas
  - Networking – virtual networks, software defined networking (SDN)
  - Communication – software defined mobile networks (SDMN)
  - Infrastructure – software defined infrastructure

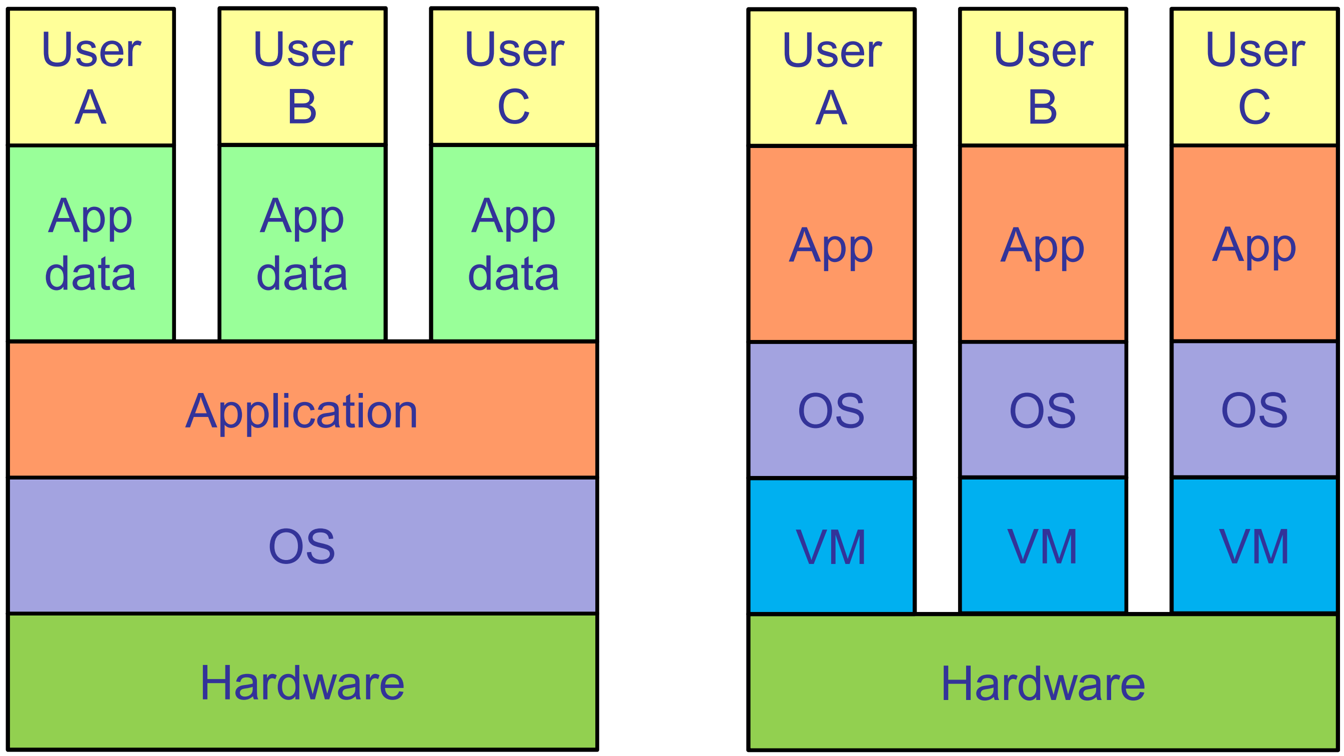
Levels of Virtualization

- **At an application level – multi-tenancy**
- At OS level – containers
- At hardware level – virtual machines

Multi-tenancy

- Single instance of application
- Virtualized application interface
- Replicate instances of application interface
- Isolate activity of each instance from other instances
- Most efficient in terms of resources
- Least flexibility – application specific
- Key requirement for SaaS

Multi-tenancy Vs Single Tenancy



Multi-tenancy – Example

- Apache virtual hosts
- Single server process bound to one IP address
- Serve separate sites for multiple domains
- Application interface isolated within a configuration directive
- Support for multiple configuration directives allows replication
- Server process provides isolation

Apache Configuration Example

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /var/www/example.com/public_html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Source - https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-ubuntu-14-04-lts

Other Multi-tenant Applications

- Salesforce
- Google App Engine
  - Support for developing multi tenant applications

Multi-tenancy – Advantages

- Efficient resource utilization
- Ability to provide stronger isolation
  - Boundary is constrained by application
  - Smaller set of interactions to be isolated
  - Resource usage is constrained by application
- Application can be offered as a service (SaaS)

Multi-tenancy – Security

- Application must be designed to offer proper isolation
- Difficult when other components are used
  - E.g. databases
- Strength of security can differ wildly
  - Depends on quality of software
- Retrofitting multi-tenancy to existing software can cause serious design issues

Multi-tenancy – Disadvantages

- Limited customization for individual users
- Complex architecture requiring significantly more development and maintenance effort
- Isolation requires through understanding of the application
- Isolation techniques are application specific
- Single point of failure

Future of Multi-tenant Applications

- Computing resources are becoming cheaper
- Multiple instances of single tenant applications are less complex
- Isolation can be moved to virtual machine level
- No single point of failure

Levels of Virtualization

- At an application level – multi-tenancy
- **At OS level – containers**
- At hardware level – virtual machines

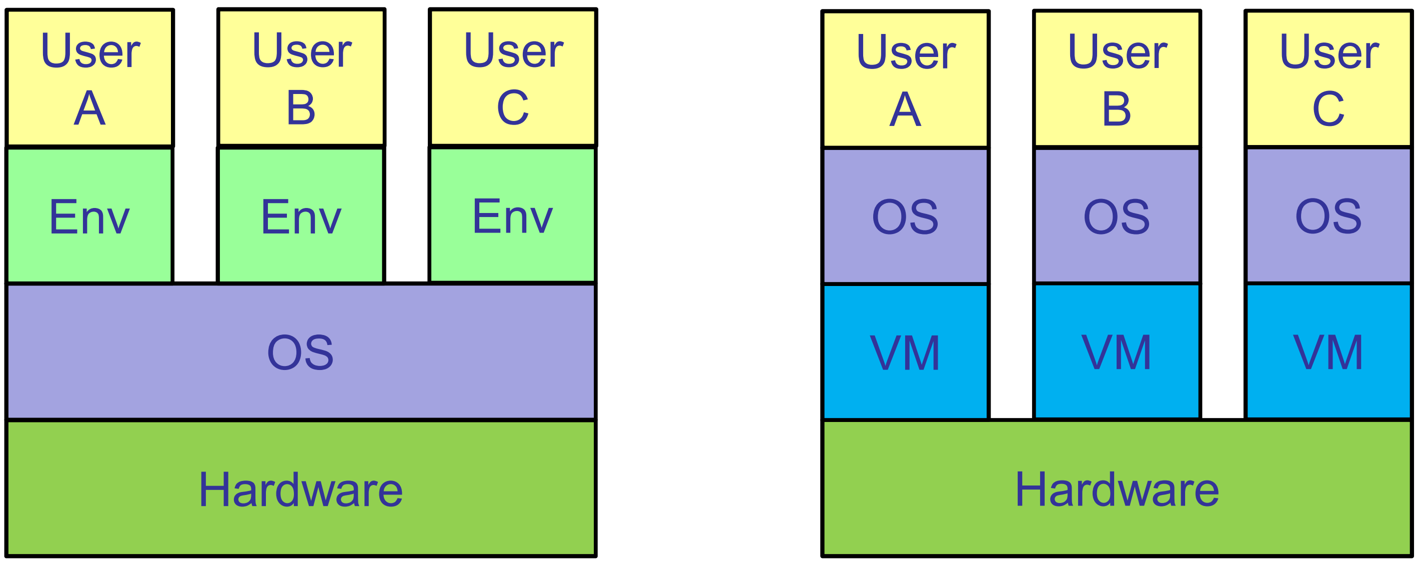
Comparison of different container implementations:

Wikipedia article on Operating-system-level virtualization

Containers

- Single instance of OS
- Virtualized user environment
- Replicate instances of user environment
- Isolate activity of each environment from other environments
- Less efficient than multi-tenancy in terms of resources
- More flexibility, but OS specific
- Recently popularized by Docker

Containers Vs Virtual Machines



History of Containers

- Originated from "chroot" in 1979
  - Provides new root file system and a process table for it and all child processes
  - Files outside this new root are not visible
- Later gave rise to FreeBSD "jails"
  - Added isolation to users and network resources
- Google developed "control groups" (cgroup) in 2006 and contributed it to Linux kernel

Linux Kernel Support for Containers

- Controlling groups of processes: **cgroup**
  - Can restrict memory usage and prioritize CPU and I/O usage of process groups
  - Allows tracking resource usage
  - Allows checkpointing, freezing and restarting process groups
- Isolating resources: **Namespaces**
  - Six namespaces: file system mount points (mnt), host name (uts), IPC, process ID (PID), network (net) and user/group ID (user)

Source: https://lwn.net/Articles/531114/

Container Implementations

- Linux – OpenVZ (2005)
- Linux – LXC (2008)
- Linux – LMCTFY (2013)
- Linux – Docker (2013)
  - Originally based on LXC, evolved later to use kernel support for containers directly
- Windows containers (Sep. 2016)

Source: https://dzone.com/articles/evolution-of-linux-containers-future

Linux Containers - LXC

- LXC is a container management layer built on top of **cgroups** and **namespaces**
- Each container has its own virtual CPU, memory, file system and network resources
- Each container has its own "init" process, which can be used to start other services. Much like a regular Linux server
- Each container can start any number of child processes

File Systems in LXC

- Avoid file system duplication with union-based file system (e.g. OverlayFS)
  - Union of a "base" (or lower) file system and a "overlay" (or upper) file system
  - Base file system is typically read-only
  - Overlay file system is read-write and contains any modifications to the base file system
  - Allows easy patching of base system
- Other copy-on-write file systems may also be used

Networking in LXC

- Bridges and taps and veth oh my!
- Just like with physical servers, you need network switches (bridge) with ports (tap), connected to Ethernet (veth)
- Network namespaces allow multiple networks to exist without address collisions
- More details later

Security of LXC

- Stigma of weaker security due to vulnerabilities in earlier versions
- Current version (1.0.x) is claimed to be as secure as a virtual machine
- UID namespace allows an unprivileged user to start a container and have root privileges within the container
- Larger providers may start offering containers as LXC matures

Linux Containers - Docker

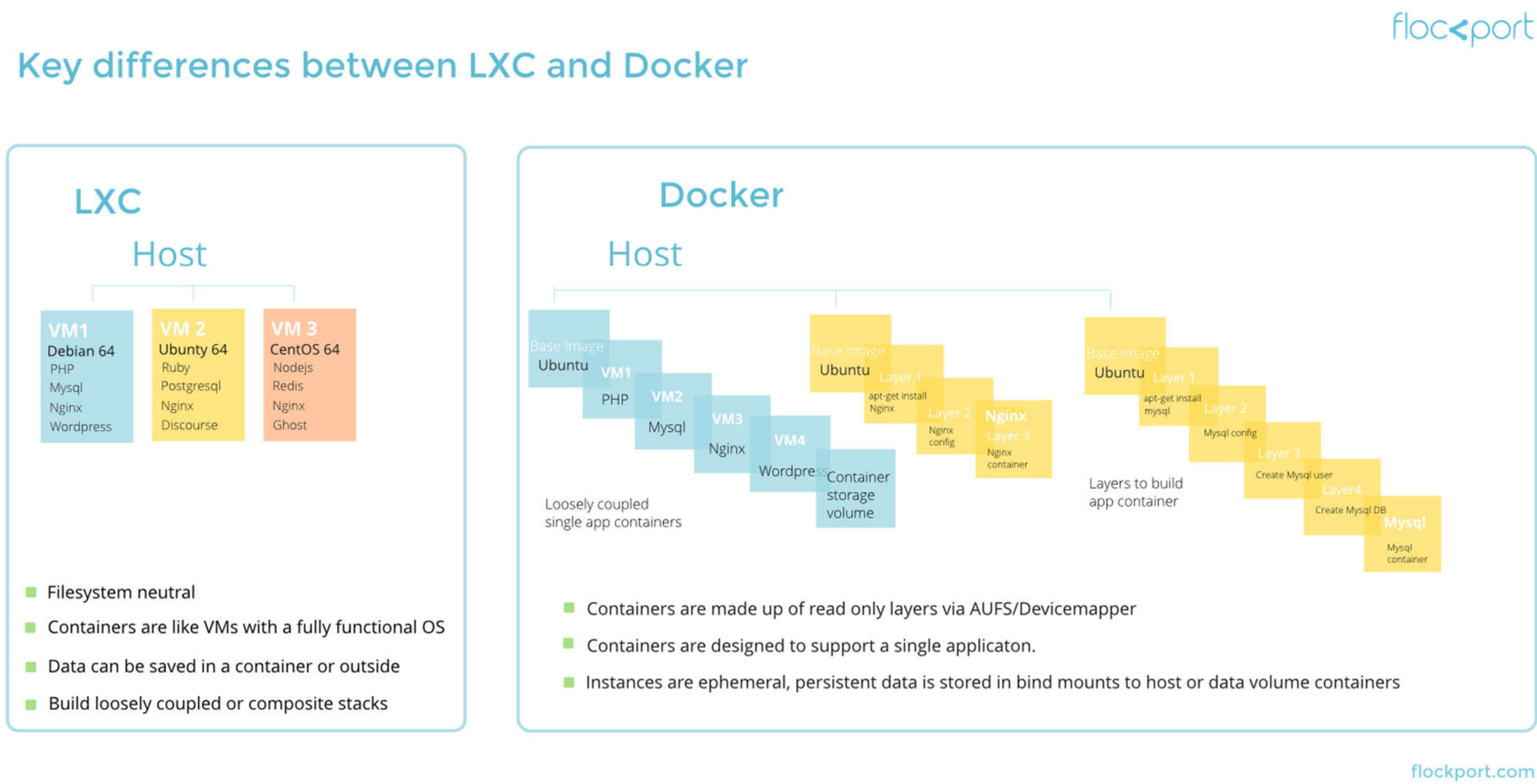
- Popular Linux container management layer
  - Like LXC
- Only run one process per container
  - Unlike LXC
- Now available for Windows Server 2016
  - For managing Windows containers
- Offers a rich set of management tools
- Written in Go language

Docker Features

- Portable deployment across machines
- Optimized for applications
- Automatically build container images
- Versioning images
- Component reuse (layering)
- Sharing (Docker hub)
- Tool echo system

Source: https://docs.docker.com/engine/faq/

LXC Vs. Docker



Source: https://www.flockport.com/lxc-vs-docker/

Challenges when using Docker

- Single process per container
  - Applications that require multiple processes (e.g. web server, database etc) need multiple containers and set up communication links
- No persistent storage
  - Containers will need separate file system mounts to access persistent data
- Network setup is rigid
  - Claims of inability to change default behaviour of assigning IP addresses

Docker Articles (Required Reading)

- Docker fundamentals (Infoworld)
- Pro-Docker:
  - Containers 101: Linux Containers and Docker Explained (Infoworld)
- Pro-LXC:
  - Differences between LXC and Docker (Flockport)

Linux Containers – Rocket (Rkt)

- Open source project developed by CoreOS
- A recent competitor to Docker
- Developed app container format "appC"
  - Defines image format, runtime environment and image discovery
- Compatible with Docker container format
- Claims to use a modular and secure design

Linux Containers - LMCTFY

- Google solution to adopt containers within their infrastructure
- Attempt to provide an abstraction to cgroup
  - Isolate internal software from changes in cgroup interface due to rapid development
- Allows specifying priority and latency requirements
  - LMCTFY converts these to appropriate cgroup rules

Description from a core developer

"Resource management API: LXC API is built for namespace support and exports cgroup support almost transparently. Linux cgroup API is unstable and hard to deal with. With lmctfy, we tried to provide an intent-based resource configuration without users having to understand the details of cgroups.

Priority - Overcommitment and sharing: lmctfy is built to provide support for resource sharing and for overcommitting machines with batch workloads that can run when the machine is relatively idle. All applications specify a priority and latency requirements. lmctfy manages all cgroup details to honor the priority and latency requirements for each task.

Programmatic interface: lmctfy is the lowest block of app management for Google's cloud. It's built to work with other tools and programs. We feel it's much better specified and stable for building more complicated toolchains above it.

We have lmctfy managing all of Google's resource isolation needs since 2007. So far, it was mangled into other pieces of Google Infrastructure. During a redesign, we were able to separate this layer out cleanly and thought it would be fun to put it out and give back." – Stackoverflow answer

Whither LMCTFY

- "libcontainer" was a Docker initiative to isolate container backend
  - Defines container format and runtime (runC)
- In May 2015 LMCTFY was given over to libcontainer project
- In June 2016, libcontainer project was given over to Open Container Initiative (OCI)

Open Container Initiative

- "… for the express purpose of creating open industry standards around container formats and runtime" - *opencontainers.org*
- Attempts to merge multiple evolving standards for container format, and runtime format
- Develops the container specification (image-spec) and runtime (runtime-spec)
- LXC, Docker and CoreOS are members
  - with 40+ major cloud technology companies

Recommended Reading

- Secure distribution of Docker images (v)
- Rkt vs others (from CoreOS)

Levels of Virtualization

- At an application level – multi-tenancy
- At OS level – containers
- **At hardware level – virtual machines**

Virtualization Requirements

- Proposed by Popek and Goldberg in 1974
  - “Formal Requirements for Virtualizable Third Generation Architectures”
- VM: Virtual machine
  - Isolated and efficient copy of the real machine
- VMM: Virtual Machine Monitor
  - Supervisory software layer which provides abstraction, replication and isolation of hardware
  - Now called "hypervisor"

Properties of VMM

- Popek & Goldberg defines three properties
- Equivalence (fidelity)
  - A program running under VMM should be identical to it running without a VMM
- Resource control (safety)
  - VMM has complete control of all hardware resources
- Efficiency (performance)
  - Majority of instructions executed without VMM intervention (i.e. directly, without emulation)

Virtualization Requirements

- Two processor modes
  - User mode
  - Supervisor mode
- Three categories of machine instructions
  - Privileged instructions
  - Control-sensitive instructions
  - Behaviour-sensitive instructions

Instruction Types

- Privileged instructions
  - Execute normally when run in supervisor mode
  - Traps when run in user mode
- Control-sensitive instructions
  - Attempts to change resource parameters or processor mode
- Behaviour-sensitive instructions
  - Effect of execution is dependant on resource parameters or processor mode

Virtualization Theorem 1

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

- i.e. All sensitive instructions must trap when run in user mode

Virtualization Theorem 2

A conventional third generation computer is recursively virtualizable if it is: a) virtualizable and b) a VMM without any timing dependencies can be constructed for it.

Virtualization - History

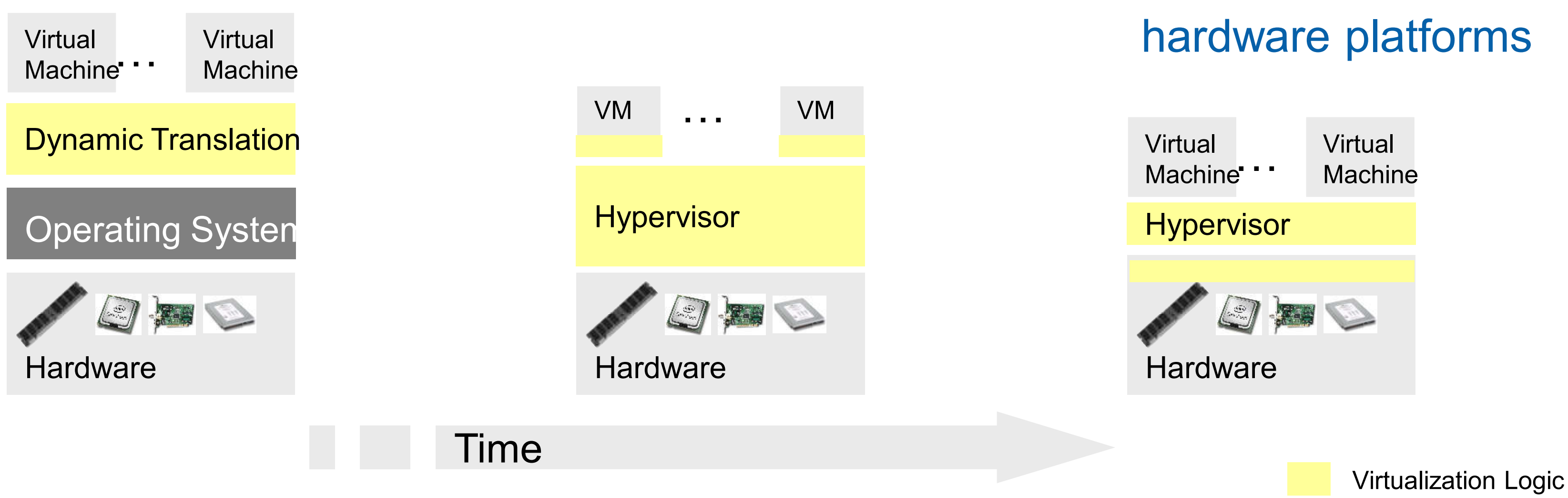
- IBM used a form of virtualization for S360 series in late 60s
  - IBM CP-40 supported 14 virtual S360 instances
- Allowed replacing multiple mainframes with a single mainframe with virtual systems
- Low server utilization was a motivator
  - 12-18% utilization on a typical x86 server
- Virtualization helped increase utilization to over 70%

Early Challenges

- Most early CPUs violated theorem 1
  - Sensitive instructions that are not privileged
  - E.g. IA-32 (i386) had 17 sensitive instructions that are not privileged
- Clever hack: replace these with a privileged instruction and add a trap to handle it
  - E.g. Binary translation in VMWare circa 1998

Evolution of Software solutions

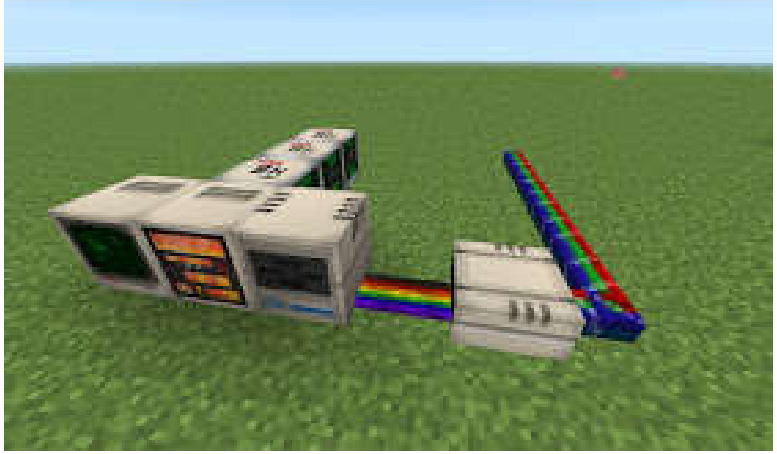
- 1st Generation: Full virtualization (Binary rewriting)
  - Software Based
  - VMware and Microsoft
- 2nd Generation: Paravirtualization
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- 3rd Generation: Silicon-based (Hardware-assisted) virtualization
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms



Source: Intel

1st Generation Virtualization

- Full system emulation
- Each instruction is interpreted and updates software representation of machine state
- Very accurate, but very slow
- E.g. Simics (1998), Gameboy emulator, Minecraft 6502 emulator



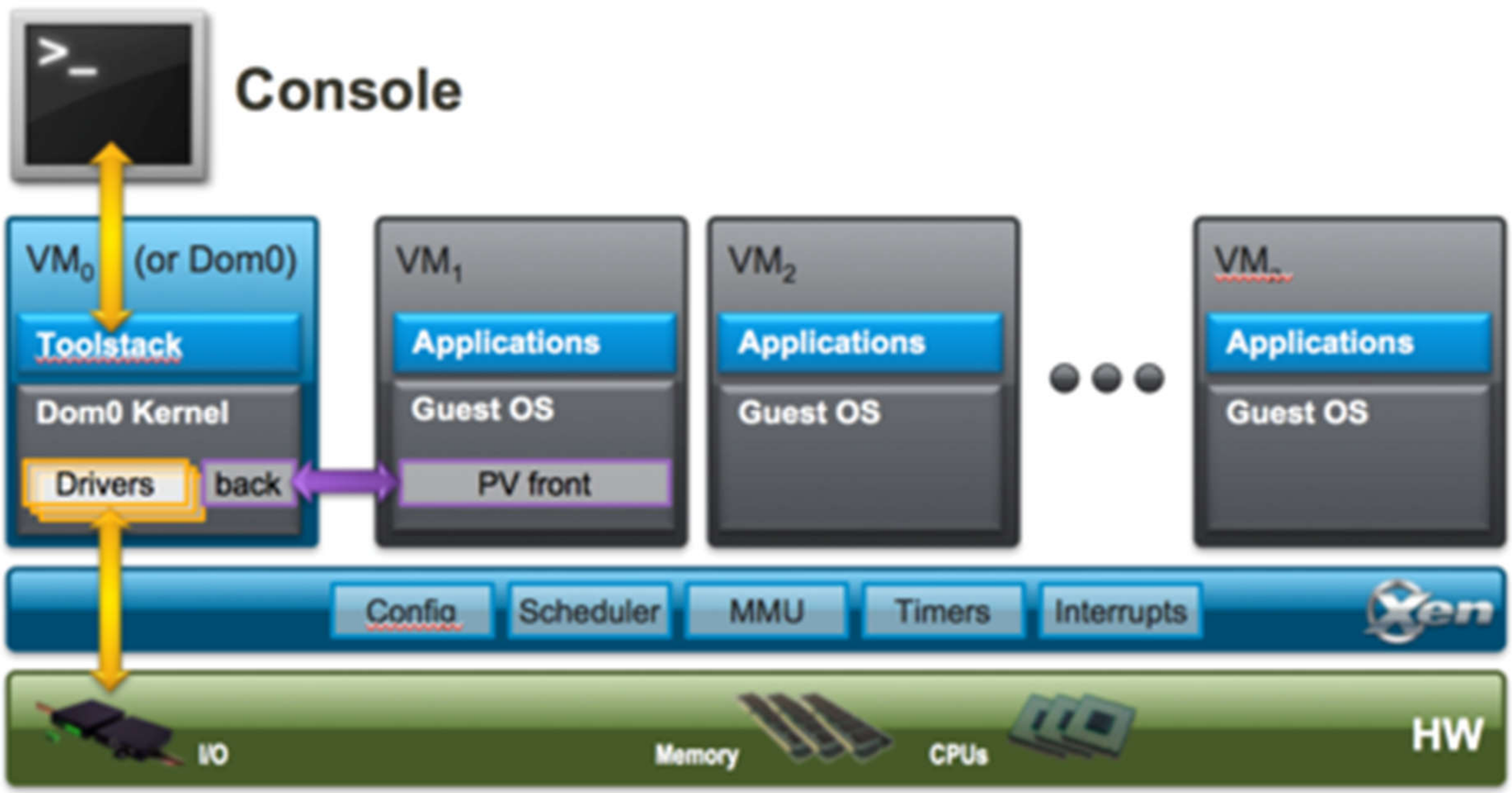
Generation 1: Partial Emulation

- Emulate just enough of the system
- E.g. QEMU, Bochs, DOSBox
- Faster than full emulation
- Take a few shortcuts to simplify hardware
- Some programs work well; Others don't work at all

Generation 2: Paravirtualization

- Make guest OS be aware of virtualization
  - Replace operations requiring privileged instructions with calls to VMM directly
  - Guest user code remains unmodified
- Only works with modifiable OS
- Instead of emulating devices, provide virtual devices through a device driver
- E.g. Xen (2003)

Paravirtualization with Xen



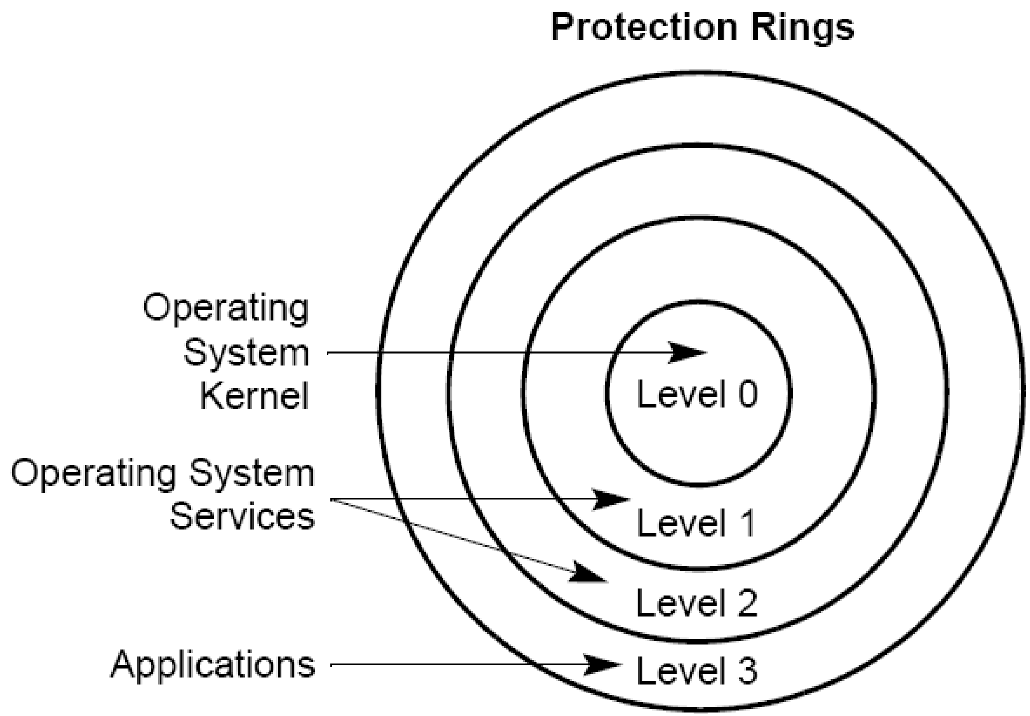
Source: wiki.xen.org

Paravirtualization with Xen

- Hypervisor is a thin layer of software running on "bare metal"
  - Starts the host kernel which is a special VM that runs in control domain (`dom0`)
  - Provides virtual devices to guest kernels
  - Supports Linux and NetBSD as host kernels
- Guest OS runs in user domain (`domU`)
  - Must be modified to use virtualized devices
  - Supports Linux, NetBSD, FreeBSD and OpenSolaris

Four Privilege Levels of X86

- Kernel normally runs in ring-0
- Guest OS must be modified to run in level 1
- VMM and host OS runs in ring-0



Device Driver with Paravirtualization

- Normal kernel device communication:

```
void nic_write_buffer(char *buf, int size) {
  for (; size > 0; size--) {
    nic_poll_ready();          // many traps
    outb(NIC_TX_BUF, *buf++); // many traps
  }
}
```

- Kernel device communication with vmm

```
void nic_write_buffer(char *buf, int size) {
  vmm_write(NIC_TX_BUF, buf, size); // one trap
}
```

Hardware Support for Virtualization

- Modern CPUs have special instructions for virtualization
  - Intel VT-x (2005)
  - AMD AMD-V (2006)
- Intel VT-x introduces two CPU modes
  - VMM runs in "VMX root" mode
  - Guest OS runs in "VMX non-root" mode
- Guest kernels can now use ring-0 in "VMX non-root" mode

Generation 3: Hardware Assisted

- VT-x and AMD-V provide a privilege level orthogonal to ring 0-4
- This offloads many challenges to writing a VMM
- Ring 0 in "VMX root" mode is now a more privileged ring-0
  - Sometimes called ring -1 (inaccurate?)

Pro/Con of 3rd Gen. Virtualization

- Benefits
  - Allows running unmodified OS as a guest
  - Faster than full emulation
- Drawbacks
  - May be slower than paravirtualization
    - Newer features like Extended Page Tables (Intel) or nested page tables (AMD) may speed up VMM
  - Unmodified OS cannot take advantage of virtualized resources
- Hybrid schemes have been proposed

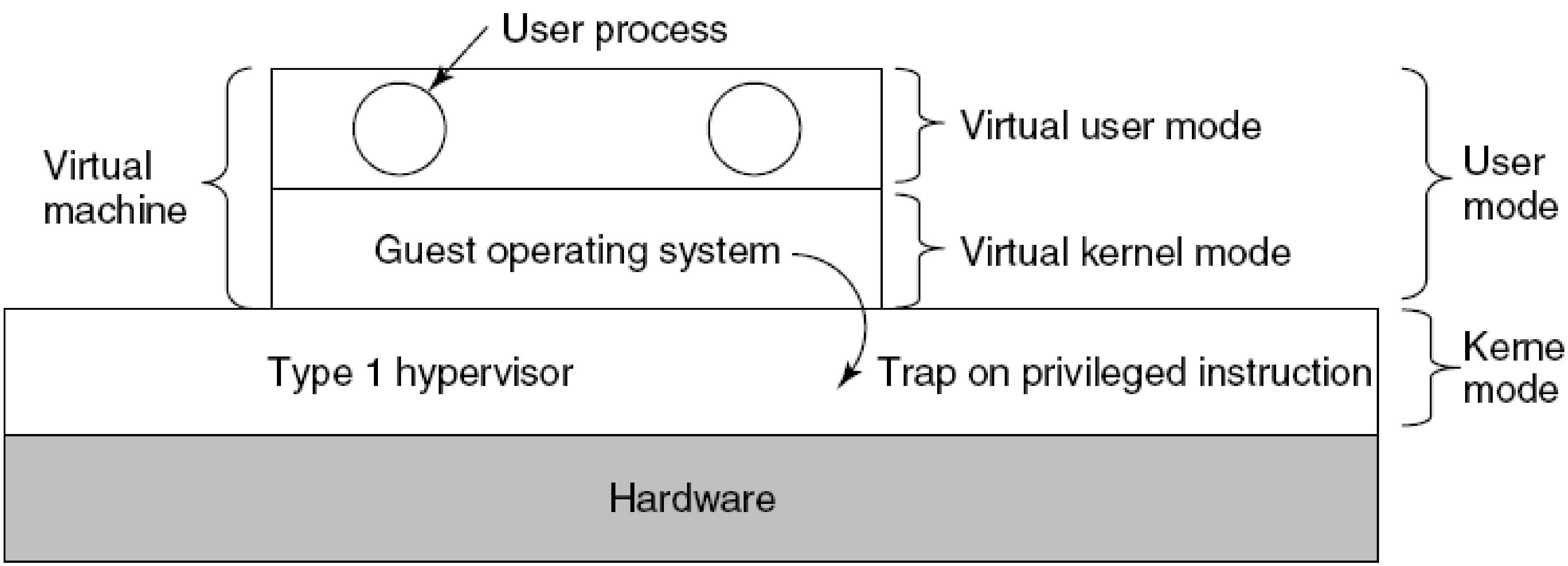
Hypervisor Types

- Type 1 hypervisor
  - Runs on "bare metal"
  - Typically has a small resource footprint
- Type 2 hypervisor
  - Runs from within a host OS
  - Requires a large resource footprint

Type 1 Hypervisor

- Virtual machines run in user mode
- Guest OS in VM thinks it is running in kernel mode – Virtual kernel Mode
- If guest OS calls sensitive instructions, hypervisor will trap and execute the instructions.
- If application on guest OS calls sensitive instructions (system calls), hypervisor traps to guest OS.

Type 1 Hypervisor



When the operating system in a virtual machine executes a kernel-only instruction, it traps to the hypervisor if virtualization technology is present.

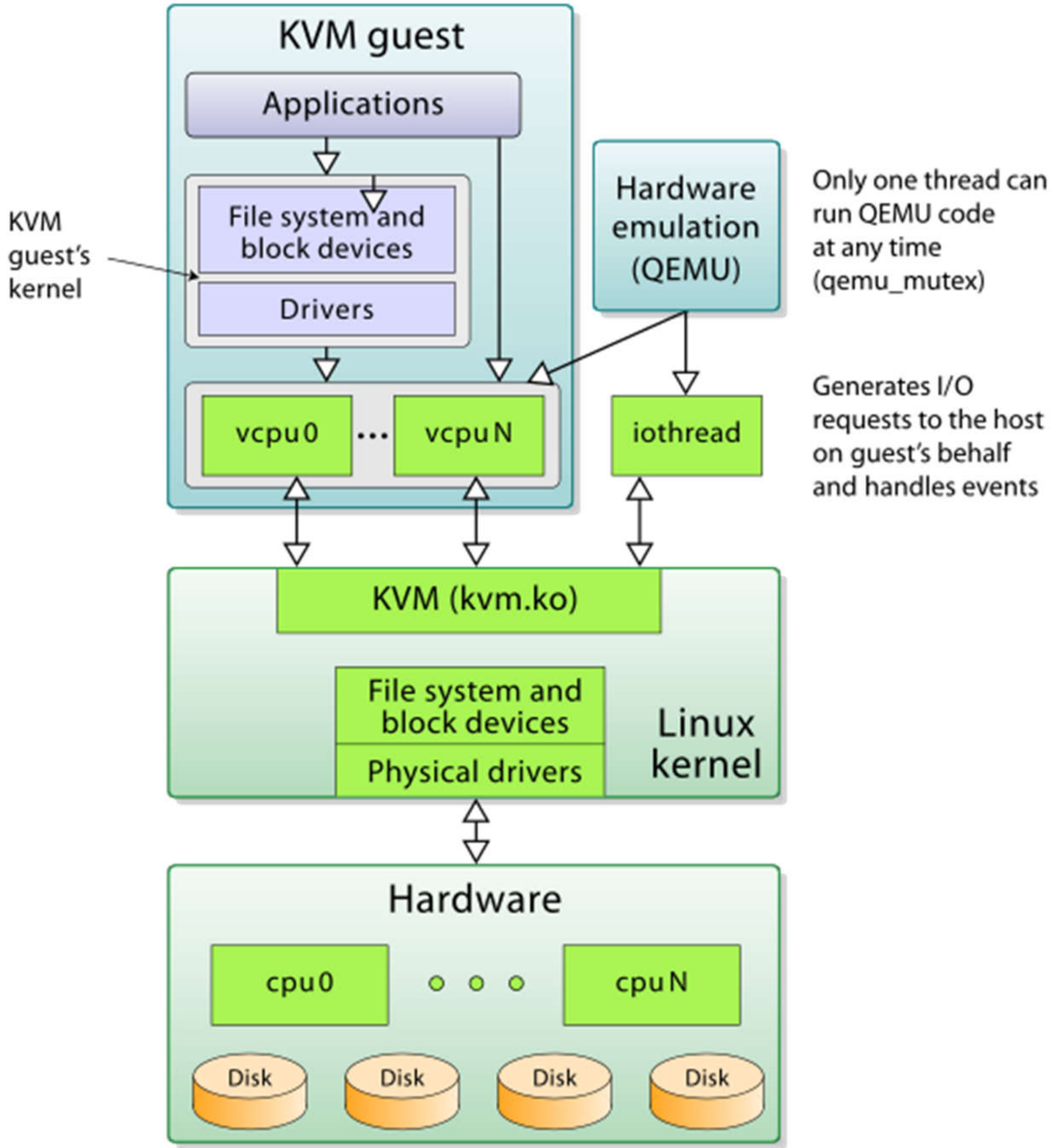
Source: Tanenbaum, Modern Operating Systems

Type 1 Hypervisor Examples

- Linux KVM (Kernel-based Virtual Machine)
- VMware ESXi
- Citrix Xen server
- Microsoft Hyper-V
- Oracle VM Server for Sparc (originally Sun)

Linux KVM

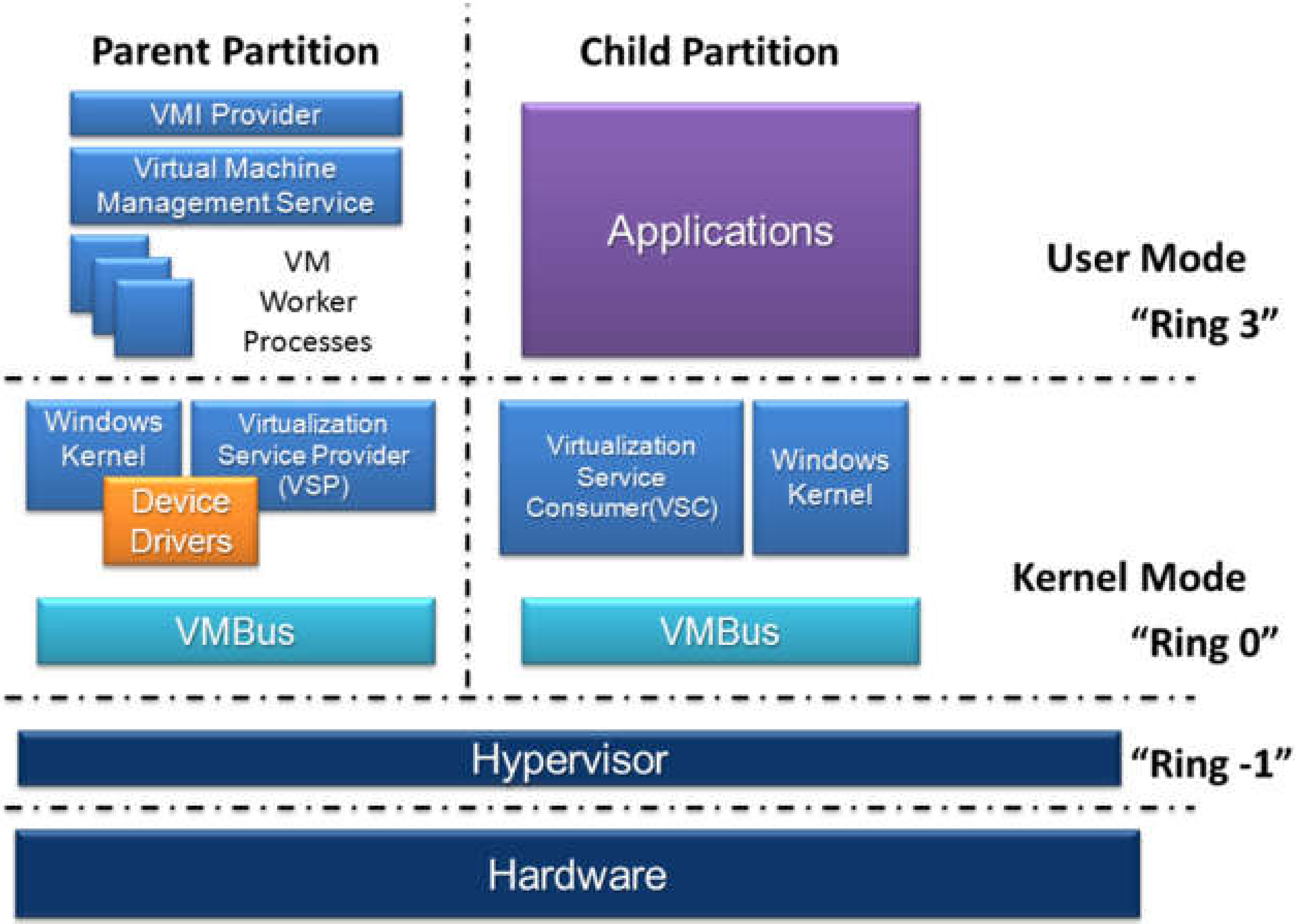
- Uses Linux to launch the hypervisor
- Some (mistakenly) call this a type-2
- Now supports ARM, IA-64 (Itanium), PowerPC and S/390 CPUs
- Supports many guest Oss
- Paravirtualization support for some devices
  - VirtIO API for Ethernet, disk I/O, memory controller and VGA



Source: Wikipedia

Microsoft Hyper-V

- Offered as a free product
- Supports Linux, FreeBSD and most versions of Windows
- Also used on XBox



Source: Wikipedia

Type 2 Hypervisor

- Runs as an application of the host OS
- Some may perform binary translation to replace sensitive instructions with calls to hypervisor
  - With run-time translation and caching, may approach near-native speed
- Distinction between type-1 and type-2 may be blurry

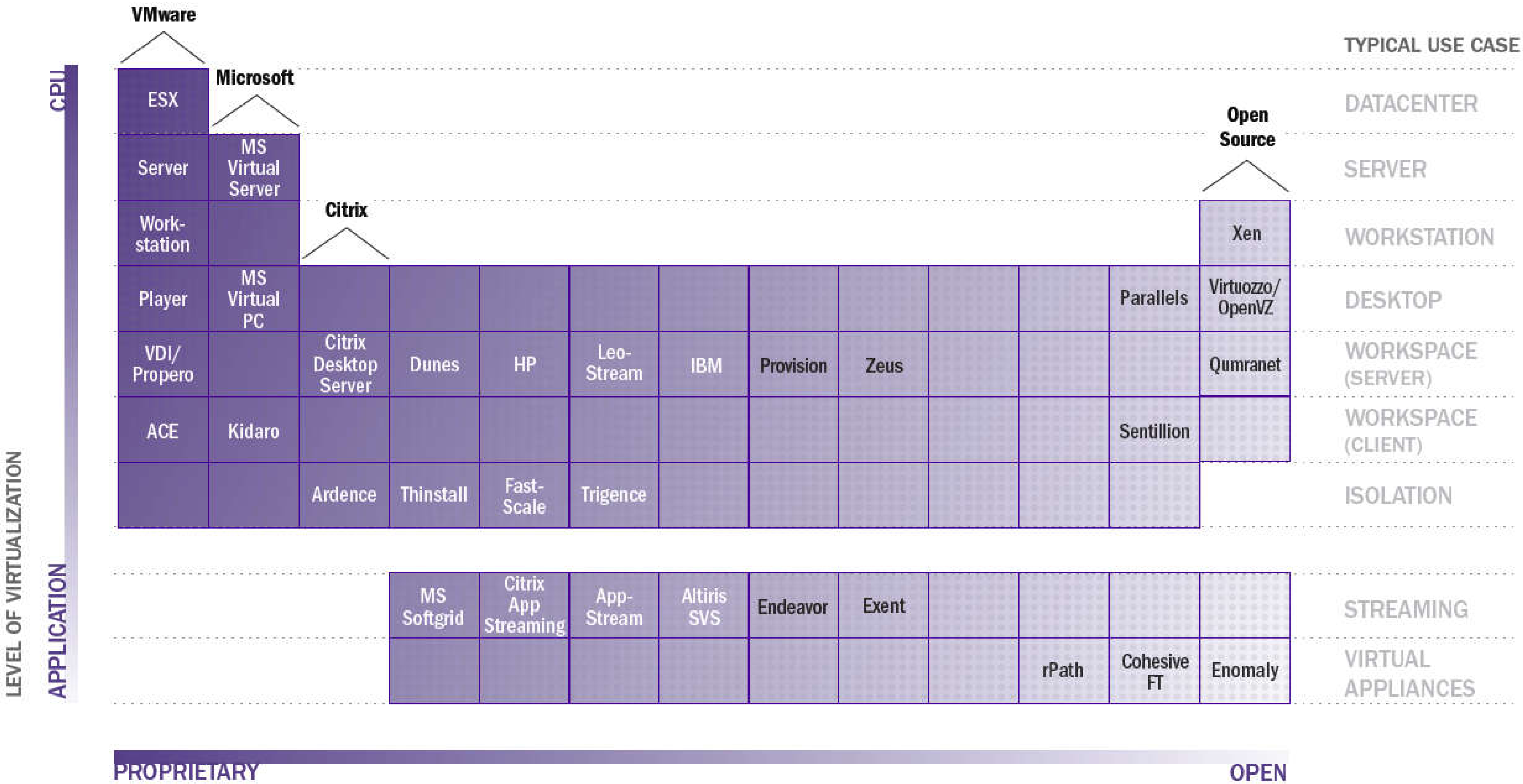
Type 2 Hypervisor Examples

- VMware Workstation
- VMware player
- VirtualBox
- Parallels for Mac
- QEMU (without KVM)

Counter Point about Type 1 / Type 2

- "The Myth of Type I and Type II Hypervisors"

Taxonomy of Virtualization



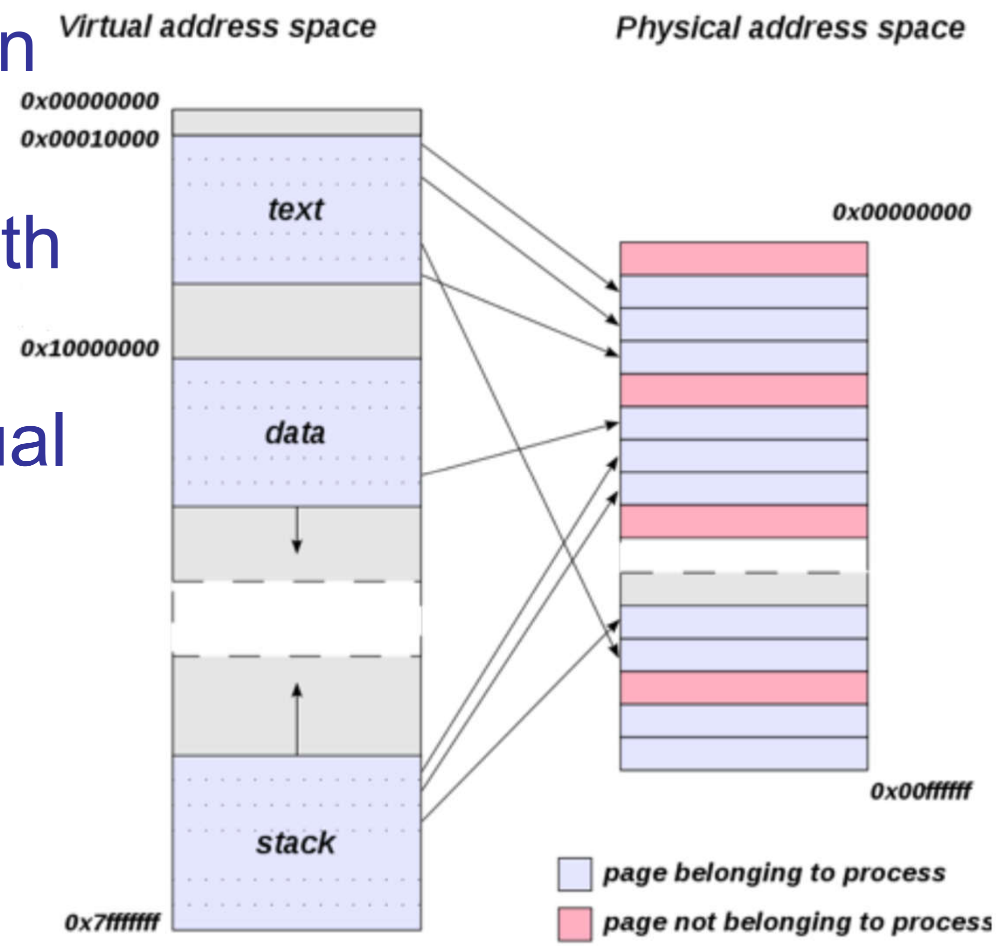
Source: Virtualization II: Desktops and applications are next – the 451 group

Virtualization of Resources

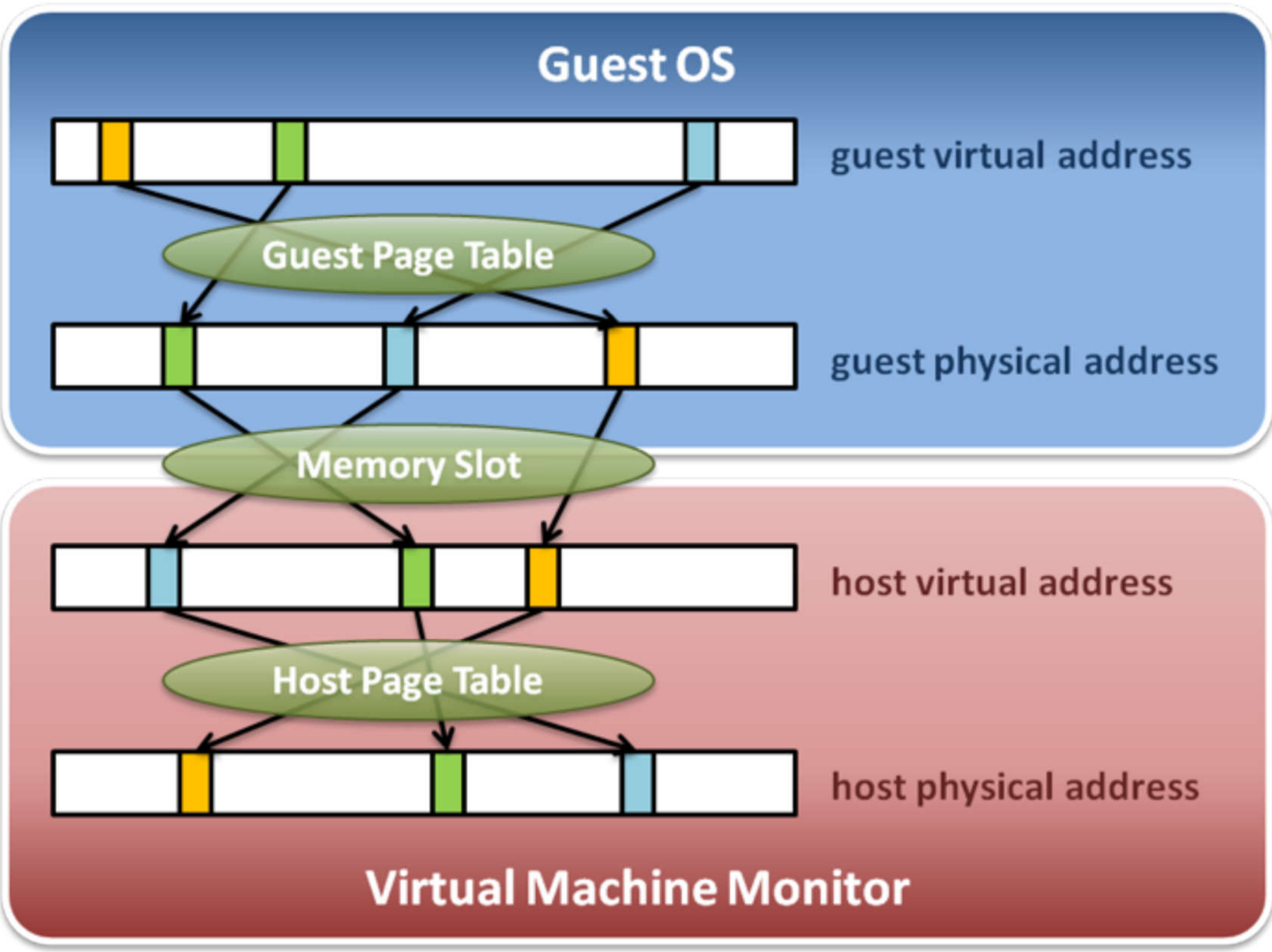
- Next challenge for efficient virtualization
- Memory virtualization
- Device (network) I/O virtualization
- Storage Virtualization

Virtual Memory

- Each process has own space
- A page is a fixed length contiguous chunk
- Page table maps virtual pages to physical pages
- Provides isolation between processes
- MMU does this (and more) in hardware



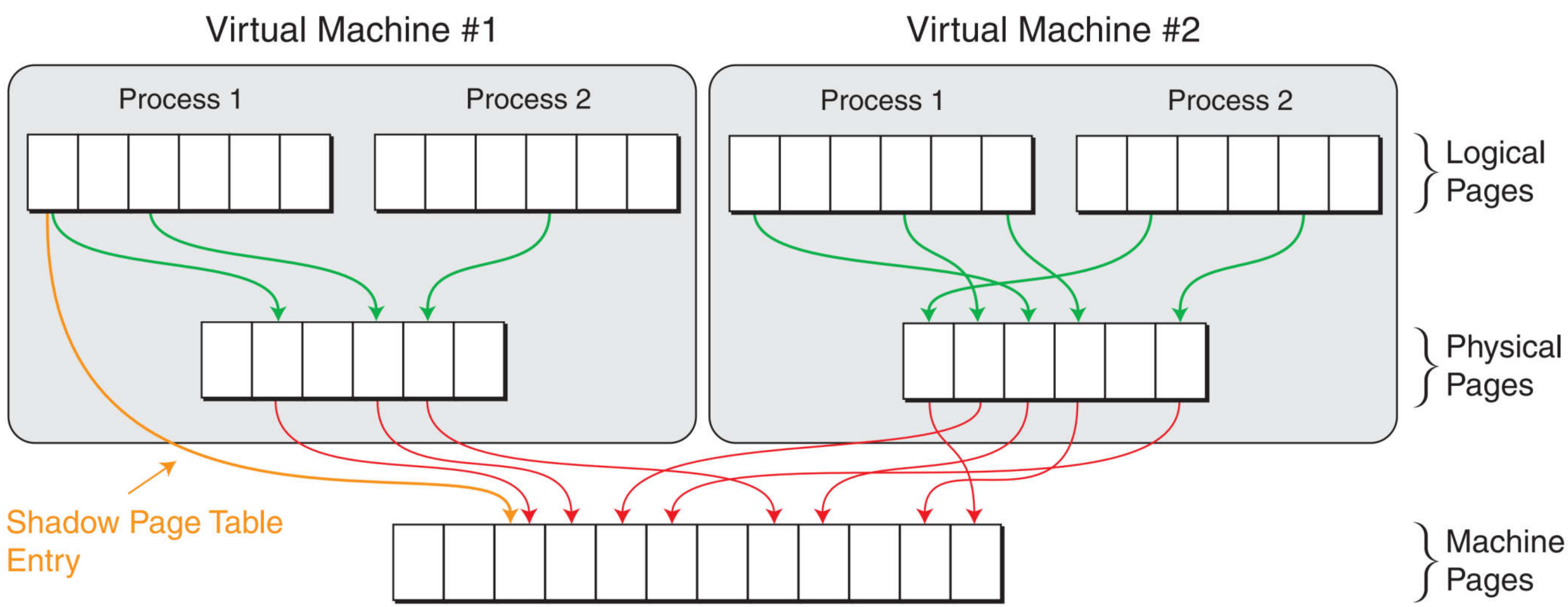
Virtual Memory in VM



Translation Lookaside Buffer (TLB)

- A MMU cache that speeds up the "page table walk"
- Search key is virtual address and search result is physical address
- Can only keep track of virtual memory of VMM (now called shadow page table)
- Each guest must keep track of its own page table (no help from MMU – E.g. TLB)

Shadow Page Table



Source: Univ. Rochester CS456 lecture slides

Hardware Support for VM Memory

- Second Level Address Translation (SLAT)
  - Intel: Extended Page Tables (EPT)
  - AMD: Rapid Virtualization Indexing (RVI, formerly Nested Page Tables or NPT)
- Simplifies VMM
  - Doesn't have to maintain a shadow page table

Virtualization of Resources

- Next challenge for efficient virtualization
- Memory virtualization
- **<u>Device (network) I/O virtualization</u>**
- Storage Virtualization

Device I/O virtualization

- I/O MMU virtualization
  - Intel: VT-d; AMD: AMD-Vi
- Make one device look like multiple virtual devices
  - Separate DMA and interrupt handling
  - Need Function Level Reset (FLR) that only reset the virtual device
- Network virtualization
  - Intel: VT-c

Virtualization of Resources

- Next challenge for efficient virtualization
- Memory virtualization
- Device (network) I/O virtualization
- **Storage Virtualization**

Storage Virtualization

- Traditional OSs requires physical disk partitions to mount on various points on the file system hierarchy
- This gets unwieldy very quickly with VMs
- Volatile demand requires storage pooling
- Block-level virtualization
  - Address space remapping to abstract storage at block level
- File-level virtualization

Implementing Storage Virtualization

- Host based
  - Separate software layer running on the host OS that provides storage virtualization
  - E.g. Logical Volume Manager (LVM) on Linux
- Storage device based
  - Using controllers that manage disk arrays
  - E.g. RAID
- Network based
  - Storage Area Networks (SAN)

Summary

- Application level virtualization
  - Multi-tenant apps
- OS level virtualization
  - Containers
- Hardware level virtualization
  - Emulation
  - Paravirtualization
  - Hardware assisted virtualization