CLOUD COMPUTING

Virtualization - Types of Hypervisor

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

What is Virtualization?

❖Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others.

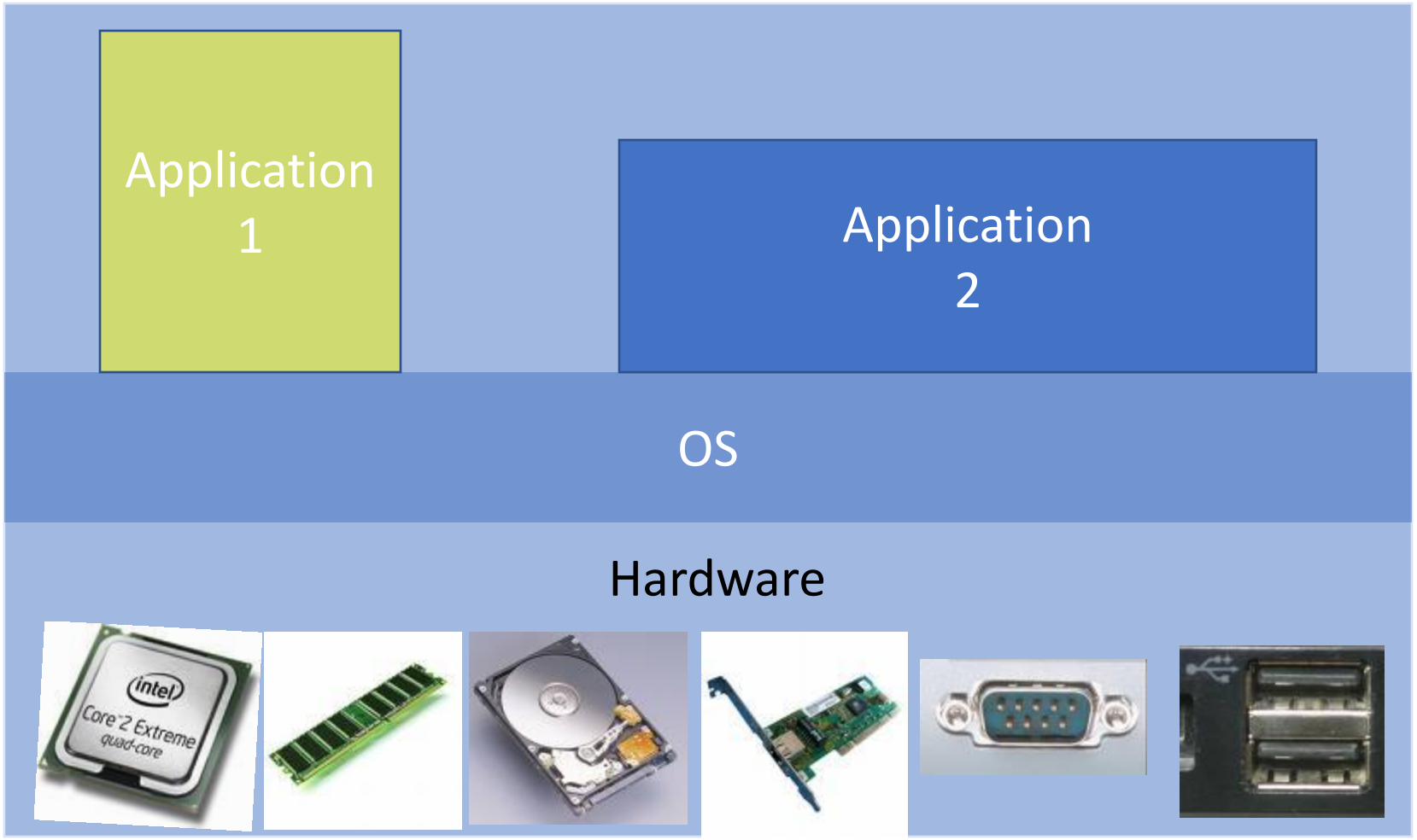
❖Practice of presenting and partitioning computing resources in a **logical** way rather than partitioning according to **physical** reality.

VM is an isolated duplicate of the physical machine.

- ➢It's an execution environment (logically) identical to a physical machine, with the ability to execute a full operating system

CLOUD COMPUTING

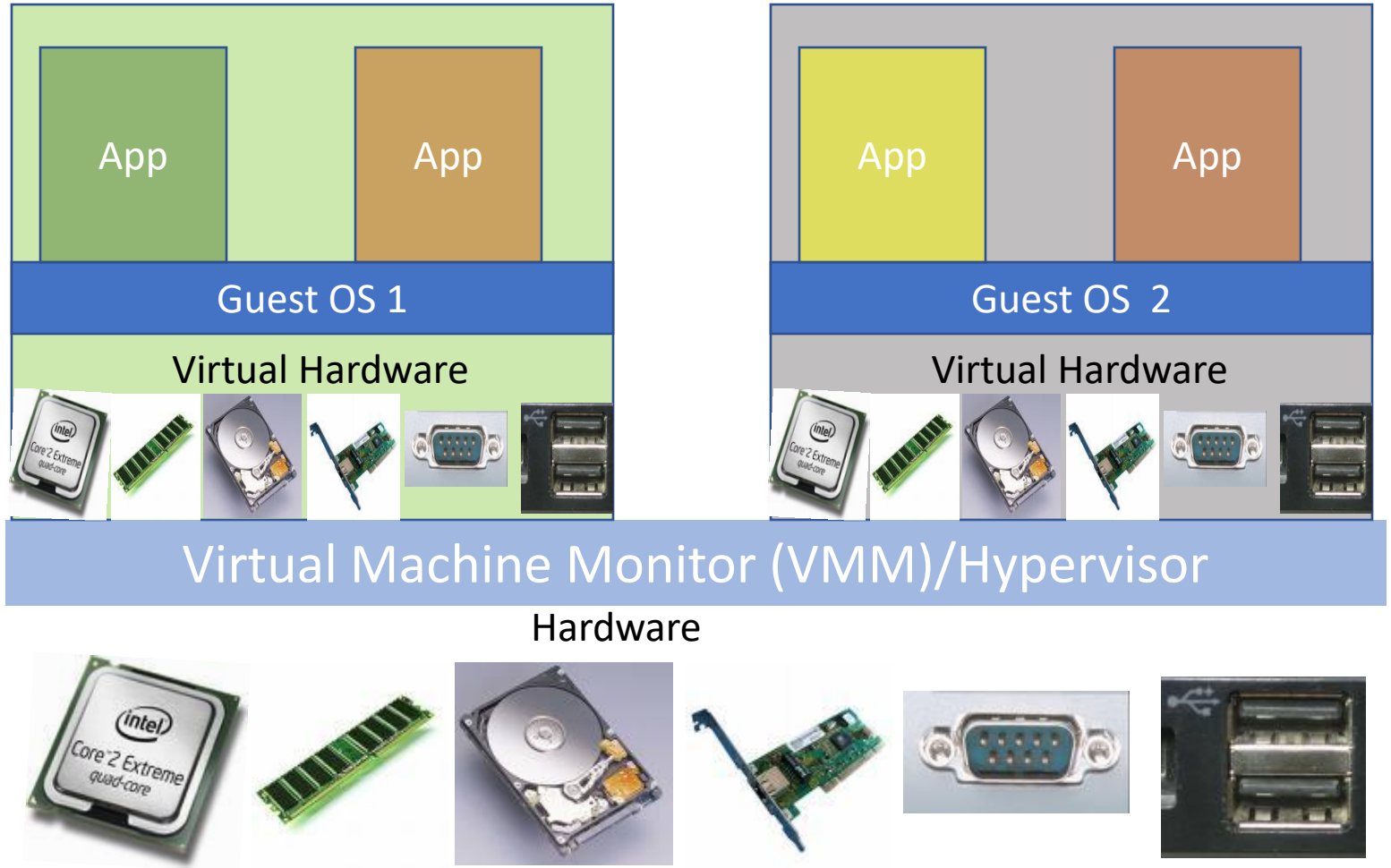
Traditional computer without virtualization (bare metal)



OS handles the bare hardware (CPU, memory, and I/O) directly.

CLOUD COMPUTING

With virtualization



Why Virtual Machines?

Operating system diversity: Can run both Linux and Windows on same h/w

Security/Isolation: Hypervisor separates the VMs from each other and isolates VMs from H/W

Rapid provisioning/Server consolidation: On demand provisioning of hardware resources

High availability/Load balancing: Ability to live-migrate a VM to other physical server

Encapsulation: The execution environment of an application is encapsulated within VM

CLOUD COMPUTING

VMM/Hypervisor

- Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This virtualization layer is commonly called the **Virtual Machine Monitor** (VMM) or **hypervisor**
- VMM manages the hardware resources of a computing system.
- Each time programs access the hardware, the VMM captures the process. In this sense, the VMM acts as a traditional OS.
- Three requirements for a VMM:
  - First, a VMM should provide an environment for programs which is essentially identical to the original machine.
  - Second, programs run in this environment should show, at worst, only minor decreases in speed.
  - Third, a VMM should be in complete control of the system resources.

Virtualization and Cloud Computing

A key technology in Cloud Computing

Enables different users to share the same physical resources

Types of Virtual Machine

Type 1 (Bare metal): VMM runs on bare metal; directly control the physical machine

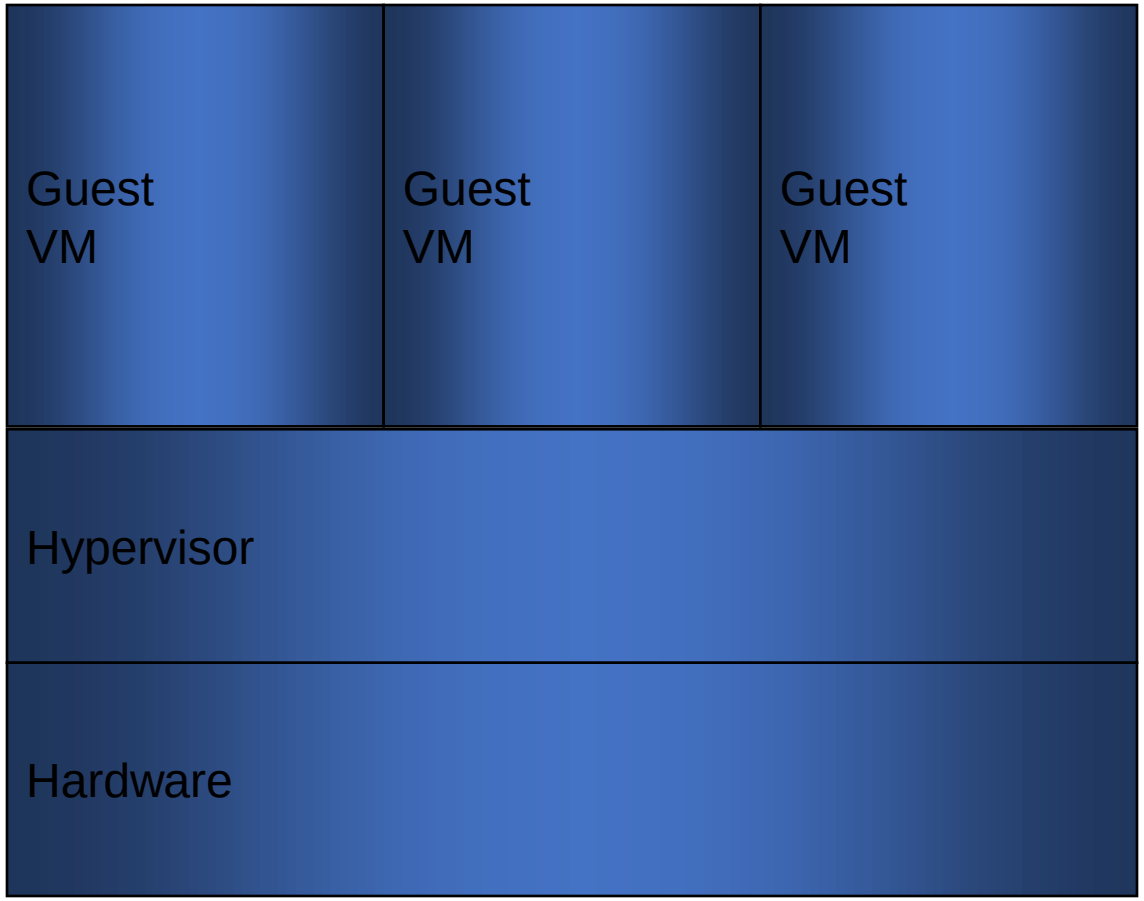
e.g., Xen, VMWare ESX server

Type 2 (Hosted): VMM runs as part of/on top of an OS

e.g., KVM, VMWare workstation

CLOUD COMPUTING

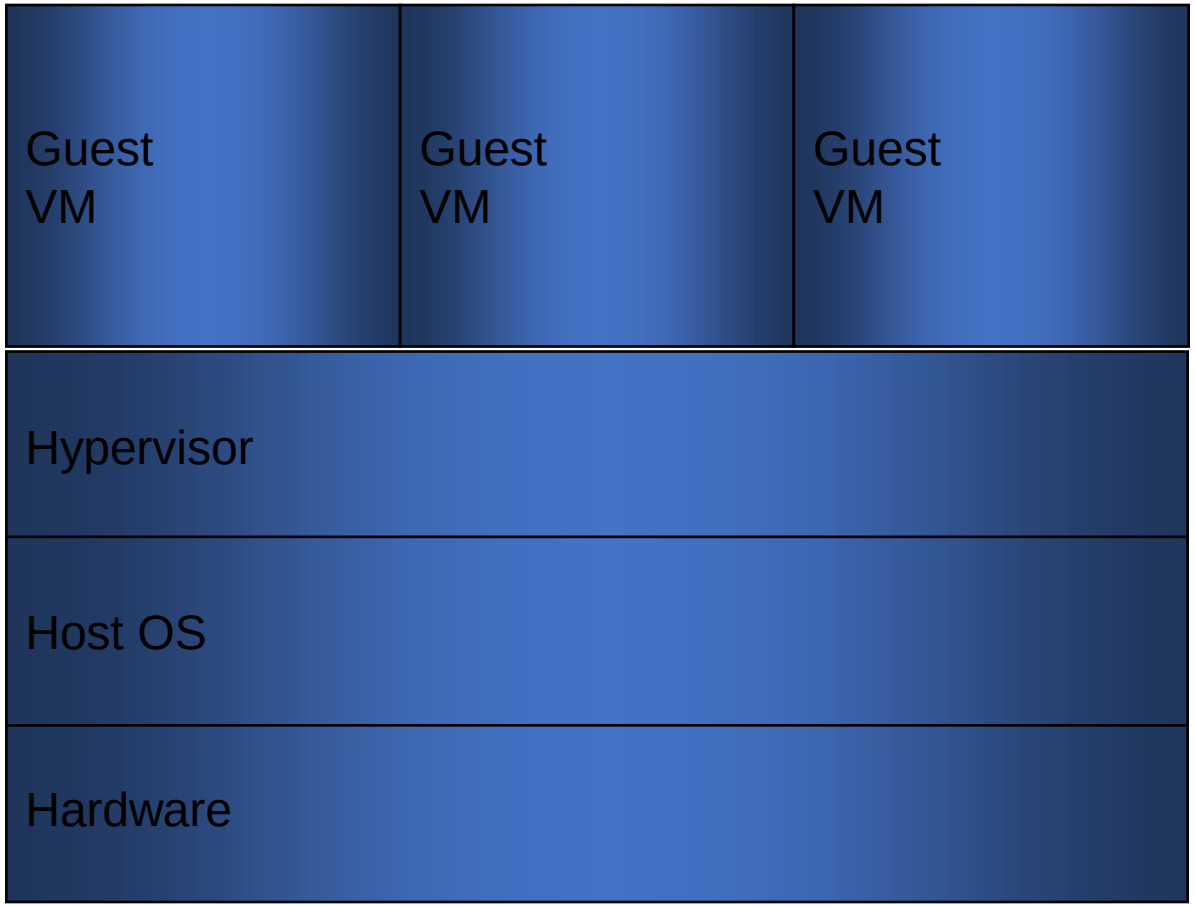
Bare Metal Hypervisors (Type 1)



IBM CP/CMS
VMware ESX
Windows Virtualization (2008)
Xen
Virtual Iron

CLOUD COMPUTING

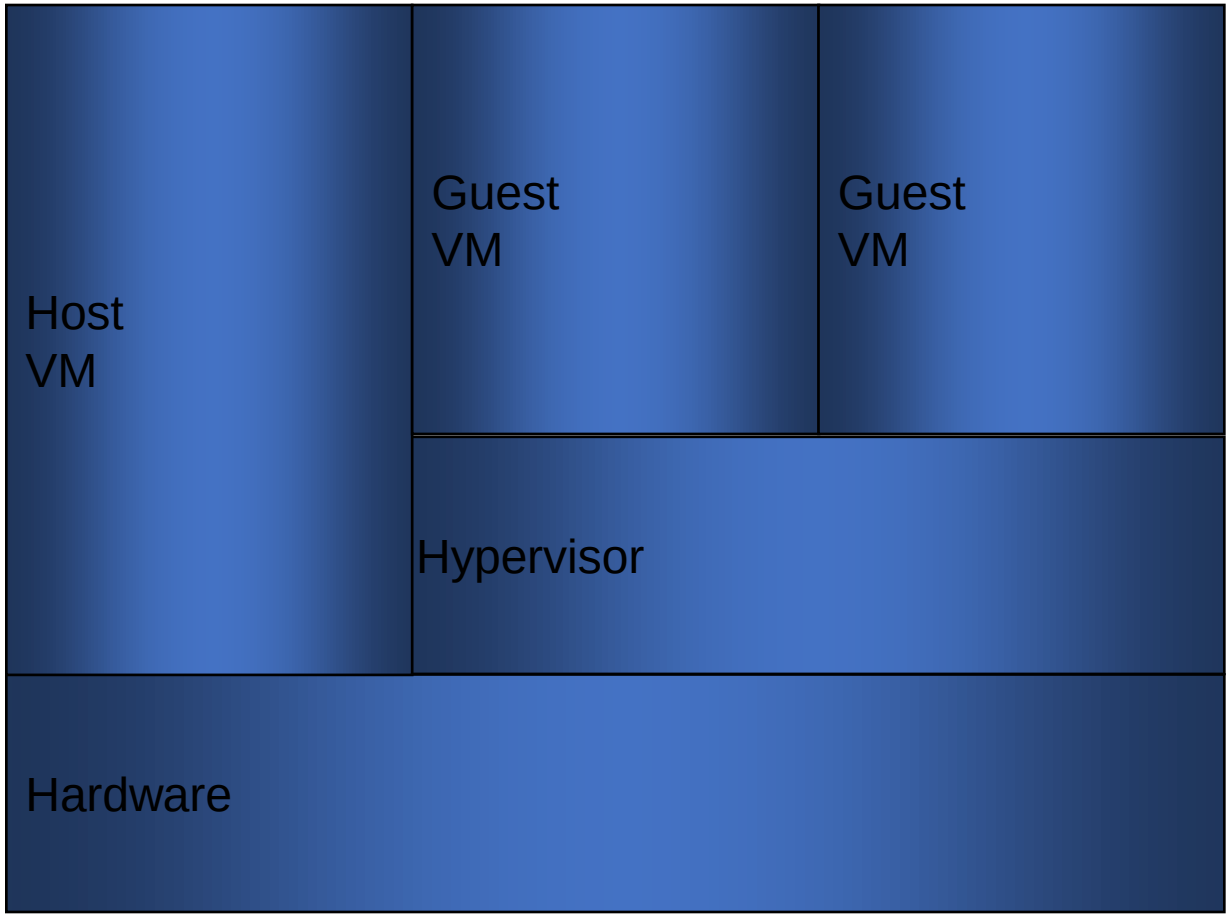
Hosted Hypervisors (Type 2)



VirtualBox
VMWare Workstation

CLOUD COMPUTING

Hybrid Hypervisors



MS Virtual Server
MS Virtual PC

CLOUD COMPUTING

Hypervisor Type 1 vs. Type 2

Criteria	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal or Native	Hosted
Definition	Runs directly on the system with VMs running on them	Runs on a conventional Operating System
Virtualization	Hardware Virtualization	OS Virtualization
Operation	Guest OS and applications run on the hypervisor	Runs as an application on the host OS
Scalability	Better Scalability	Not so much, because of its reliance on the underlying OS.
Setup/Installation	Simple, as long as you have the necessary hardware support	Lot simpler setup, as you already have an Operating System.
System Independence	Has direct access to hardware along with virtual machines it hosts	Are not allowed to directly access the host hardware and its resources
Speed	Faster	Slower because of the system's dependency
Performance	Higher-performance as there's no middle layer	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less Secure, as any problem in the base operating system affects the entire system including the protected Hypervisor
Examples	• VMware ESXi • Microsoft Hyper-V • Citrix XenServer	• VMware Workstation Player • Microsoft Virtual PC • Sun's VirtualBox

CLOUD COMPUTING

Additional references

What is a Hypervisor?
https://www.vmware.com/topics/glossary/content/hypervisor
https://youtu.be/EvXn2QiL3gs
https://www.vmware.com/topics/glossary/content/bare-metal-hypervisor
Types of hypervisor
https://www.youtube.com/watch?v=dckSc61WS6w
Bare metal vs hypervisor
https://blog.servermania.com/bare-metal-vs-hypervisor-which-is-right-for-your-project/

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Paravirtualization and Transparent Virtualization

Venkatesh Prasad
Department of Computer Science

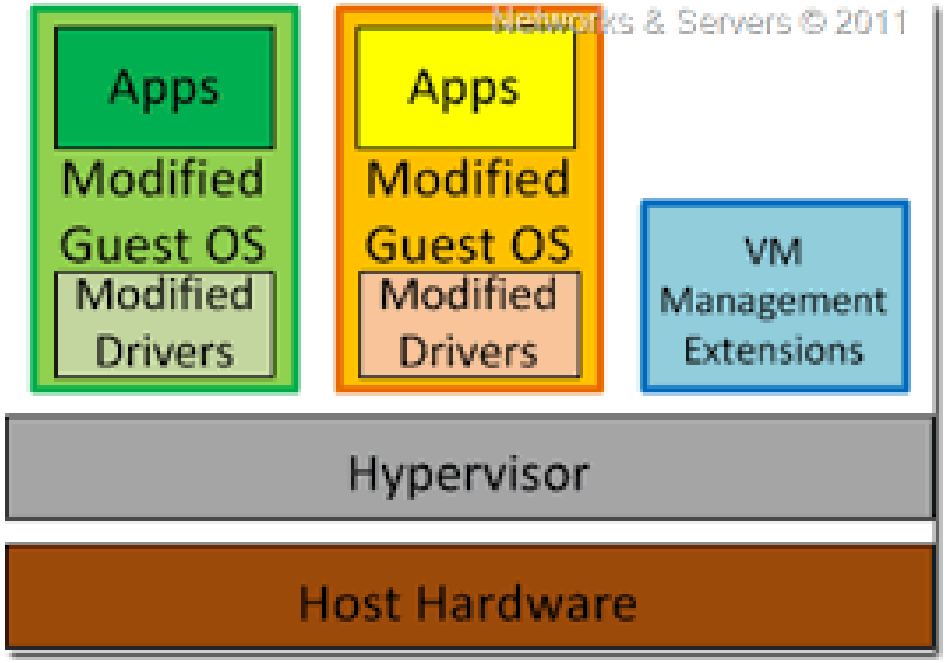
Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

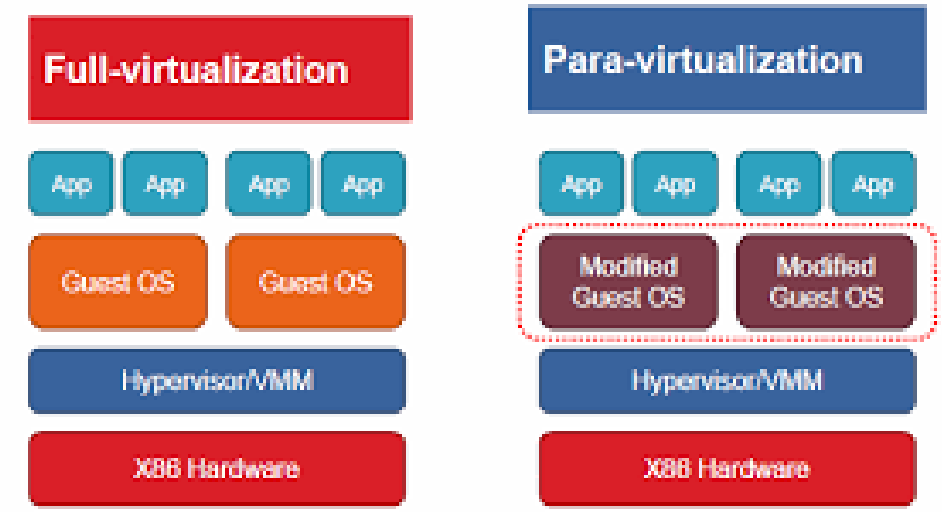
Types of Virtualization

- **Paravirtualization**
  - Modify OS to run on a hypervisor
    - Xen
  - VMM provides APIs for guest OS
  - Useful if source code of OS is modifiable
    - IBM: MVS, VM
    - Linux
    - Microsoft: Windows
- Full (**transparent**) **virtualization**
  - OS runs without modification
    - VMWare
    - kvm



CLOUD COMPUTING

Full vs Para Virtualization



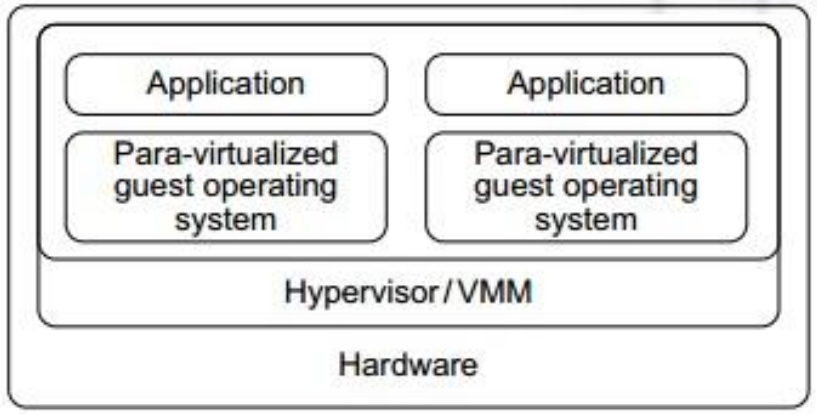
- Full virtualization architecture intercepts and emulates privileged and sensitive instructions at runtime
- Para-virtualization handles these instructions at compile time.
- The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervisor or VMM.
  - Xen assumes such a para-virtualization architecture.

Paravirtualization Overview

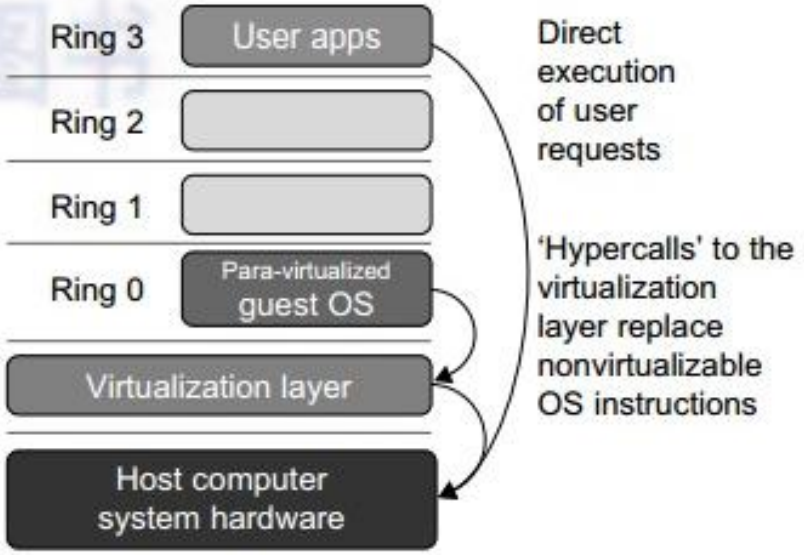
- Paravirtualization needs to modify the guest OS.
- Modify guest so that it interfaces better with hypervisor
  - Leverage hypervisor APIs (hypercall)
    - Special I/O APIs vs emulating hardware I/O
  - For improved performance
- A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications
- Performance degradation is a critical issue of a virtualized system.
  - No one wants to use a VM if it is much slower than using a physical machine.
- Para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.

CLOUD COMPUTING

Paravirtualization Overview (Cont.)



Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization process



The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls.

(*Courtesy of VMWare [71]*)

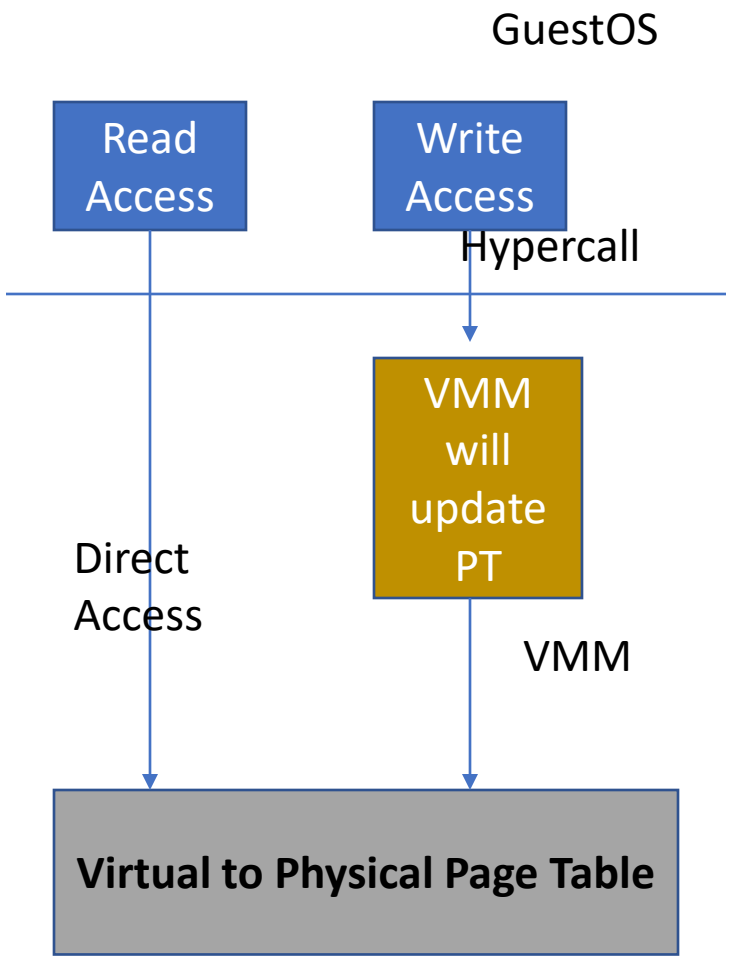
Paravirtualization Overview (Cont.)

- The guest operating systems are para-virtualized.
- They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls
- The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed.
- The OS is responsible for managing the hardware and the privileged
- instructions to execute at Ring 0, while user-level applications run at Ring 3.

CLOUD COMPUTING

Paravirtualization Example

- Xen writeable page tables
  - Guest OS has enabled writeable page tables option
  - Guest OS tries to write page tables
  - Xen makes that page table page temporarily writeable
    - But removes the page from the page table
  - Re-starts guest
  - Guest can continue to modify page table, then call a Xen routine
  - Xen then updates the real page table



Paravirtualization Example (Cont.)

- Suppose Guest OS wants to start Oracle; Oracle needs 1GB of virtual memory
- Page table stores mapping of virtual memory to physical memory
  - Assuming 4K pages, there are 1GB/4K = 250,000 page table entries
- Under pure trap and emulate virtualization, there will be a trap for each entry
  - 250,000 traps

Paravirtualization Example (Cont.)

- Since we need a page table with 250,000 entries, the page table itself is stored in memory
  - Suppose we need 4MB for the page table
  - This is 4MB/4K = 1000 pages
- With Xen writeable page table support
  - The first time the guest OS tries to write a page in the page table, there will be a trap
  - Xen will remove page from page table
  - After that, the guest OS can modify entries in page as needed
  - When completed; guest OS will call Xen API
  - Xen will validate page table entries and put page back in page table
- 1000 traps vs 250,000 traps

Paravirtualization Issues

- First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well.
- Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications.
- Finally, the performance advantage of para-virtualization varies greatly due to workload variations.
- Compared with full virtualization, para-virtualization is relatively easy and more practical.
- The main problem in full virtualization is its low performance in binary translation. To speed up binary translation is difficult. Therefore, many virtualization products employ the para-virtualization architecture. The popular Xen, KVM, and VMware ESX are good examples.

KVM

- This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel.
- Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine.
- KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

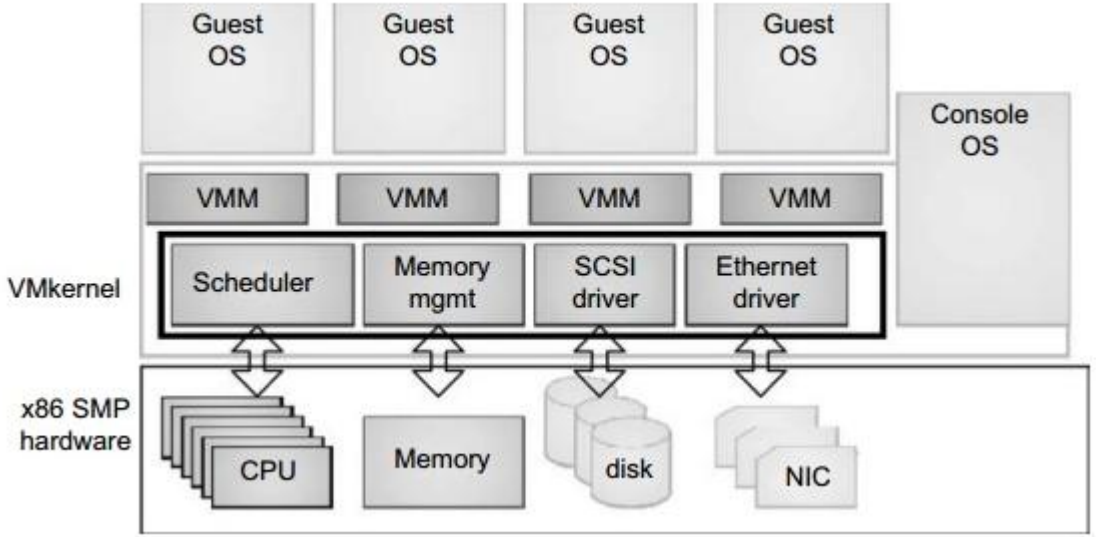
Paravirtualization with Compiler Support

- Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time.
- The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervisor or VMM.
- Xen assumes such a para-virtualization architecture.
- The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0.
  - This implies that the guest OS may not be able to execute some privileged and sensitive instructions.
- The privileged instructions are implemented by hypercalls to the hypervisor.
- After replacing the instructions with hypercalls, the modified guest OS emulates the behavior of the original guest OS.
- On an UNIX system, a system call involves an interrupt or service routine. The hypercalls apply a dedicated service routine in Xen.

CLOUD COMPUTING

VMware ESX Server architecture using Para-Virtualization

- ESX is a VMM or a hypervisor for bare-metal x86 symmetric multiprocessing (SMP) servers.
- It accesses hardware resources such as I/O directly and has complete resource management control.
- To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.



An ESX-enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console.

VMware ESX Server architecture using Para-Virtualization (Cont.)

- The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources.
- The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created.
- Hardware interface components are the device drivers and the VMware ESX Server File System.
- The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.

Summary

- Transparent virtualization
  - Trap-and-emulate
  - Binary translatiom
- Paravirtualization
- Example: Page tables
  - Transparent: shadow page tables
  - Paravirtualization: special page table APIs (Xen)

Additional References

Para vs Full vs Transparent virtualization

https://www.geeksforgeeks.org/difference-between-full-virtualization-and-paravirtualization/

https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Virtualization

Software – trap and emulate, binary translation

Venkatesh Prasad

Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Direct Execution

Idea: Run *most* instructions of the VM directly on the hardware

Advantage: Performance *close* to native execution

Challenge: How to ensure isolation/protection ?

If all instructions of the VM are directly executed then VMM has no control !

Where the original intent for a VMM and VM is to run an unmodified guest OS on the VM as if it were executing directly on the hardware, an alternative is to modify the guest OS replacing hard to virtualize instructions and/or providing more convenient abstractions especially of IO devices. Paravirtualization often results in more efficient code, but of course limits the range of usable guest OSs and the convenience of their use.

Direct Execution + Trap and Emulate

Idea:

- Trap to hypervisor when the VM tries to execute an instruction that could change the state of the system/take control (i.e., impact safety/isolation).
- Emulate execution of these instruction in hypervisor
- Direct execution of any other innocuous instructions on h/w that cannot impact other VMs or the hypervisor

How to do Trap and Emulate?

- Generally two categories of instructions:
  - User instructions:
    Typically compute instructions
    e.g., *add, mult, ld, store, jmp*
  - System instructions:
    Typically for system management
    e.g., *iret, invlpg, hlt, in, out*

invlpg— priviledged instruction to invalidate TLB Entries

How to do Trap and Emulate? (Cont.)

Two modes of CPU operation:

- **User mode** (in x86-64 typically ring 3)
- **Privileged mode** (in x86-64 ring 0)
- Attempt to execute system instructions in user mode generates trap/general protection fault (gpf)

System state:

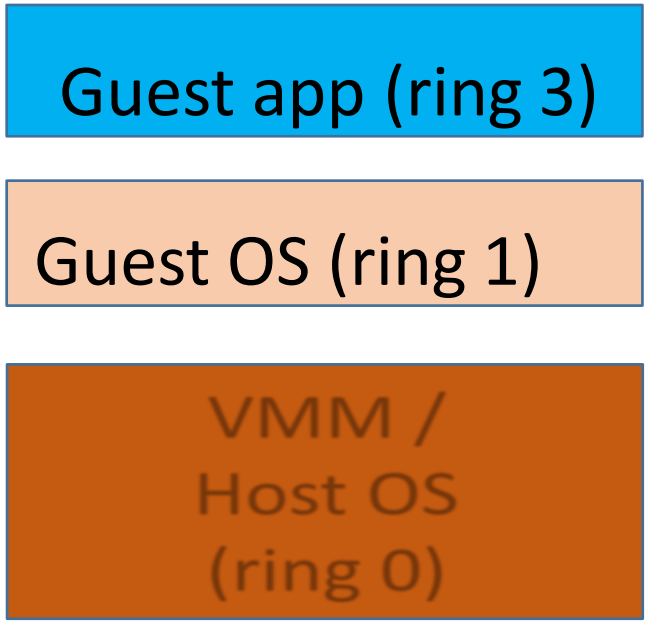
- Example, control registers like *cr3*
- Access to system state in user mode trigger gpf

Idea:

- Run the VM in user mode (ring 3), while the hypervisor in privileged mode (ring 0)
- Anytime VM tries to execute an system instruction or tries to access system state, trap to VMM

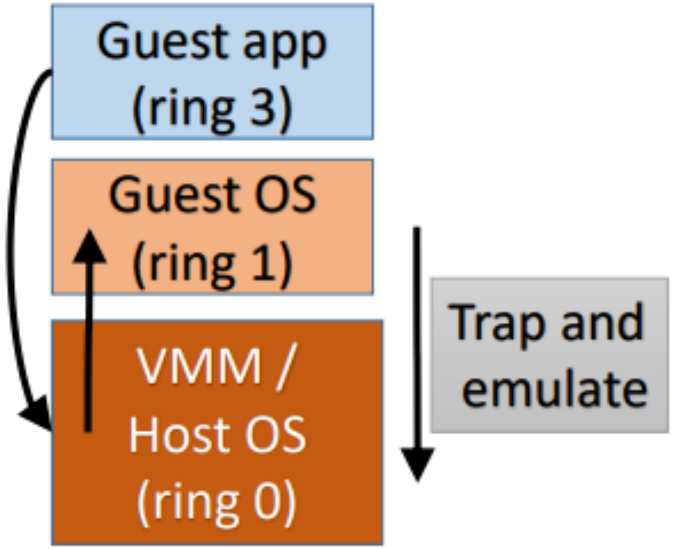
Trap and Emulate VMM

- All CPUs have multiple privilege levels
  - Ring 0,1,2,3 in x86 CPUs
- Normally, user process in ring 3, OS in ring 0
  - Privileged instructions only run in ring 0
- Now, user process in ring 3, VMM/host OS in ring 0
  - Guest OS must be protected from guest apps
  - But not fully privileged like host OS/VMM
- Trap-and-emulate VMM: guest OS runs at lower privilege level than VMM, traps to VMM for privileged operation



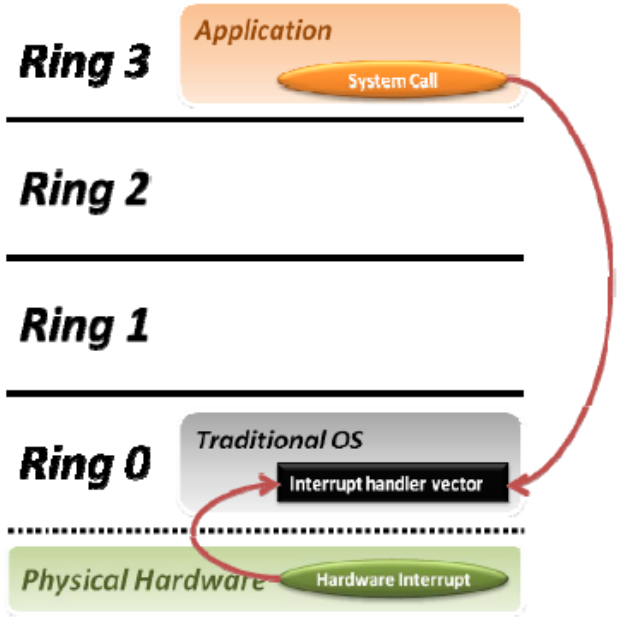
Trap and Emulate VMM (Cont.)

- Guest app has to handle syscall/interrupt
  - Special trap instr (int n), traps to VMM
  - VMM doesn't know how to handle trap
  - VMM jumps to guest OS trap handler
  - Trap handled by guest OS normally
- Guest OS performs return from trap
  - Privileged instr, traps to VMM
  - VMM jumps to corresponding user process
- Any privileged action by guest OS traps to VMM, emulated by VMM
  - Example: set IDT, set CR3, access hardware
  - Sensitive data structures like IDT must be managed by VMM, not guest OS



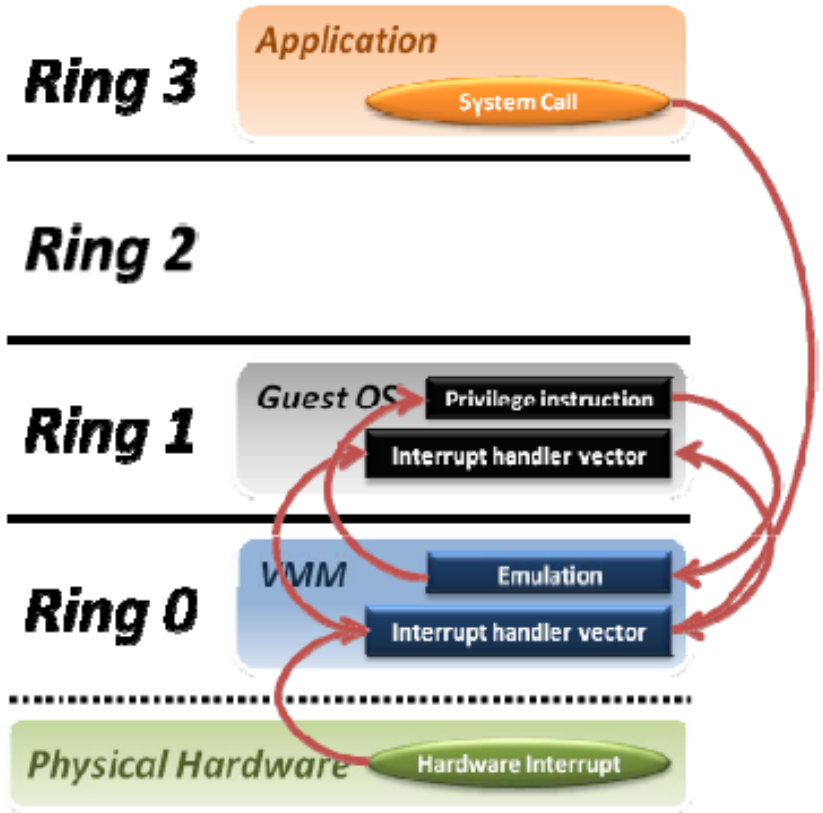
Trap and Emulate

- Privilege Rings in Hardware
- Typically OS runs in high privilege mode (Ring 0)
- And application in Ring 3 (less privilege)
- But with a VMM, the
  - VMM runs in ring 0
  - Which ring does Guest OS run in?



Trap and Emulate Contd.

Run the Guest OS in Ring 1
On executing a privileged instruction in Ring 1
- Trap into the VMM
- VMM emulates the instruction
  - Does everything that the original instruction was supposed to do but without executing it
- Then go back to the Guest OS



CLOUD COMPUTING

Trap and Emulate: Earliest virtualization Technique

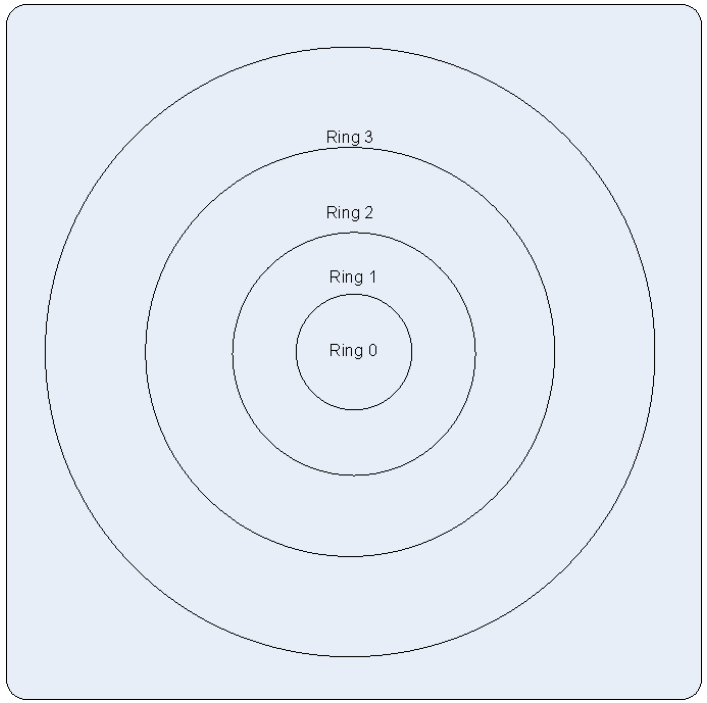
Transparent virtualization

Hypervisor traps privileged instructions and emulates

- Access to physical pages
- Physical I/O devices
- Control registers

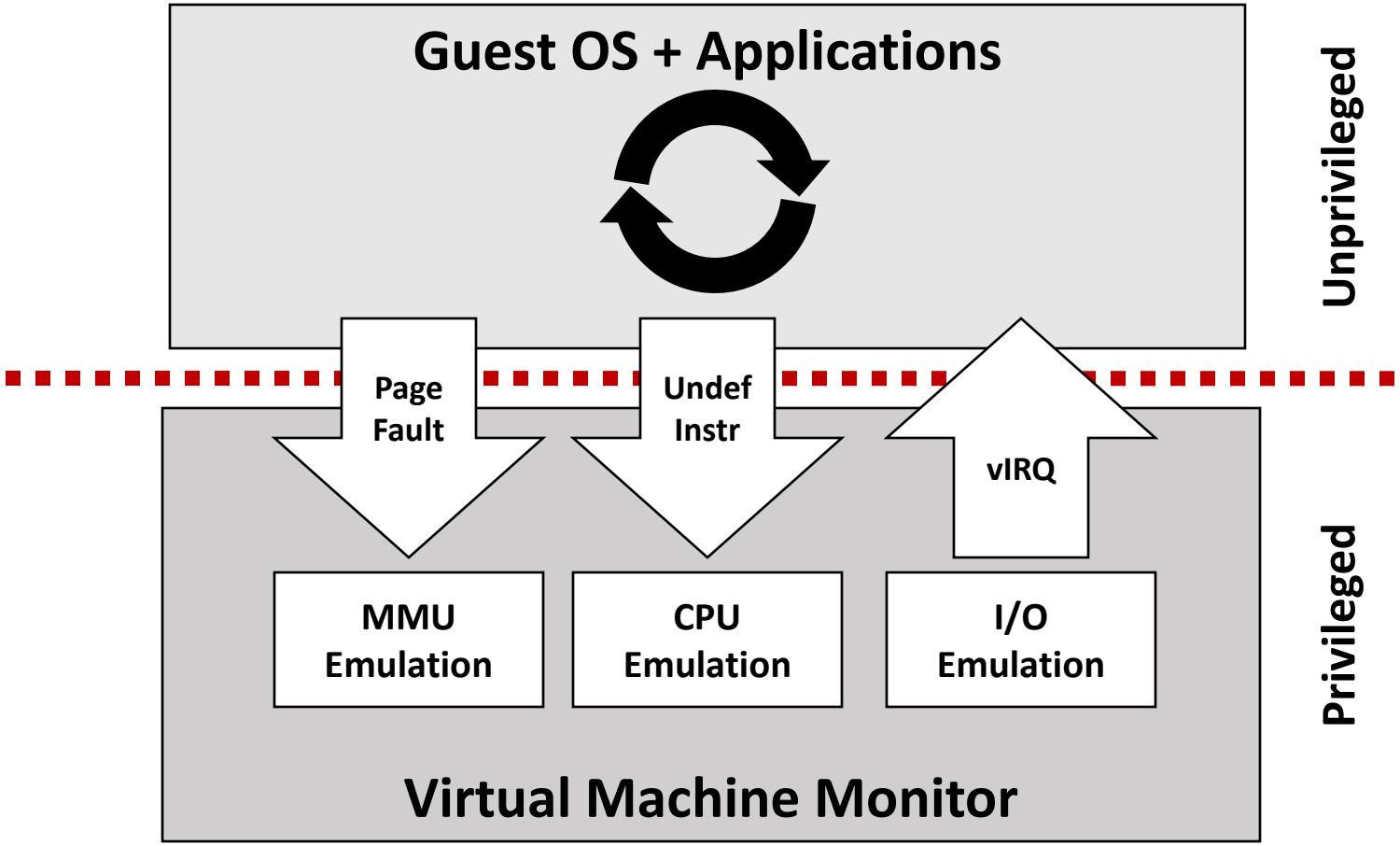
Handling privileges

- All processors have rings of privilege
- Run hypervisor in highest privilege ring (ring 0)
- Run guest in lower ring



CLOUD COMPUTING

Trap and Emulate (Summary)



Source : https://slideplayer.com/slide/5346246/ Scott Devine, VMWare

Problems with Trap and Emulate Virtualization

- Guest OS may realize it is running at lower privilege level
  - Some registers in x86 reflect CPU privilege level (code segment/CS)
  - Guest OS can read these values and get offended!
- Some x86 instructions which change hardware state (sensitive instructions) run in both privileged and unprivileged modes
  - Will behave differently when guest OS is in ring 0 vs in less privileged ring 1
  - OS behaves incorrectly in ring1, will not trap to VMM
- Why these problems?
  - OSes not developed to run at a lower privilege level
  - Instruction set architecture of x86 is not easily virtualizable (x86 wasn't designed with virtualization in mind)

Limitations of Trap and Emulate Virtualization

State of the processor is privileged if
- access to that state breaks the virtual machine's isolation boundaries
- it is needed by the monitor to implement virtualization

Defining privileged state this way is a very virtualization specific way. The architecture manual will tell you what the processor designers believe is privileged. They will make access to this state restricted to system mode code.

But basically its pretty simple, the monitor must protect against access to certain parts of the processor and memory.

Limitations of Trap and Emulate Virtualization (Cont.)

Examples of privileged state is the MMU and interrupt subsystem.

The MMU is privileged because it allows access to any piece of physical memory. The monitor needs to divide the memory among the VMs some access to the MMU by a VM would allow it to access another VM's physical memory thus breaking the boundaries. Similarly the CPU is shared through time multiplexing so access to the interrupt enable flag by the VM would allow it to take over the CPU by disabling all interrupts.

Examples of non-privileged state would be the general purpose registers and the floating point unit. Note that there may be cases where the monitor needs to reserve this state and thus may become privilged.

"Strictly Virtualizable"

A processor or mode of a processor is strictly virtualizable if, when executed in a lesser privileged mode:

- all instructions that access privileged state trap
- all instructions either trap or execute identically

Again another operational definition. This idea behind strictly virtualizable is whether trap and emulate will work.

Issues with Trap and Emulate Virtualization

- Performance Overhead
  - E.g., trapping privileged instructions
- Trap costs may be high
- Not all architectures support it
  - x86 architecture is non-virtualizable
  - Depends on the fact that executing an instruction at a lower privilege than required will cause a **trap** to the VMM.

Recent x86 systems try to facilitate virtualization by dealing with the sensitive instructions that are not privileged and by facilitating memory management with hardware. They have introduced a new mode, root mode. Now the mode will consist of the usual four levels paired with a normal/root mode. All sensitive instructions cause transfer to the root mode.

x86 is not virtualizable via Trap and Emulate

- Example: *popf* (pop flags)
  - Can be used in user mode to change ALU flags
  - Can be used in privileged mode to change system state flag (e.g., interrupt delivery flag)
  - Trouble: No trap is *popf* is attempted to alter interrupt flag in user mode --- CPU just ignores it !

    → Behavior sensitive instruction but is not privileged
- There are about 17 such instructions in x86

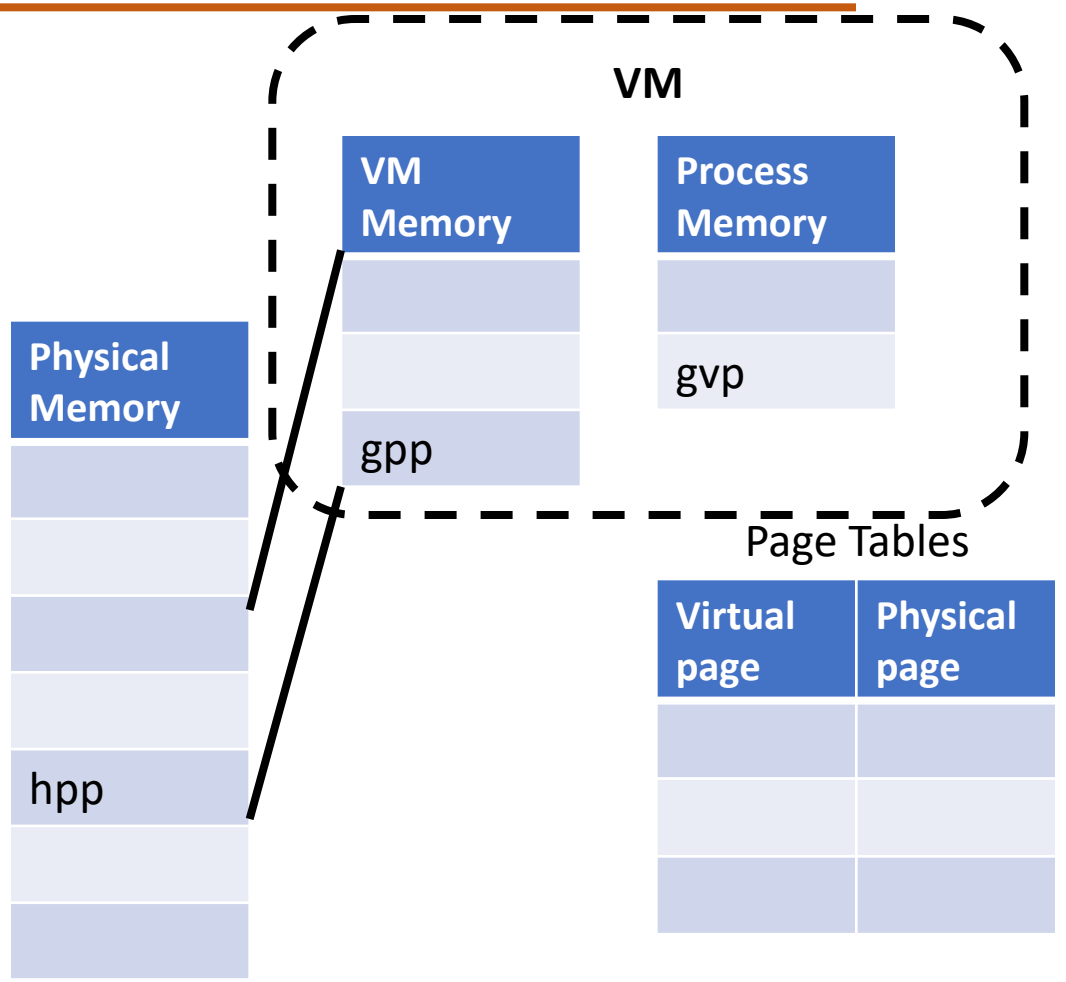
x86 virtualization

- X86 ISA (pre-2005) does not meet the Popek & Goldberg requirements for virtualization
- ISA contains 17+ sensitive , unprivileged instructions:
  - SGDT, SIDT, SLDT, SMSW, PUSHF, POPF, LAR, LSL, VERR, VERW, POP, PUSH, CALL, JMP, INT, RET, STR, MOV
- Most simply reveal that the kernel is running in user mode
  - PUSHF
- Some execute inaccurately
  - POPF
- Virtualization is still possible, requires workarounds

CLOUD COMPUTING

Trap and Emulate Exercise

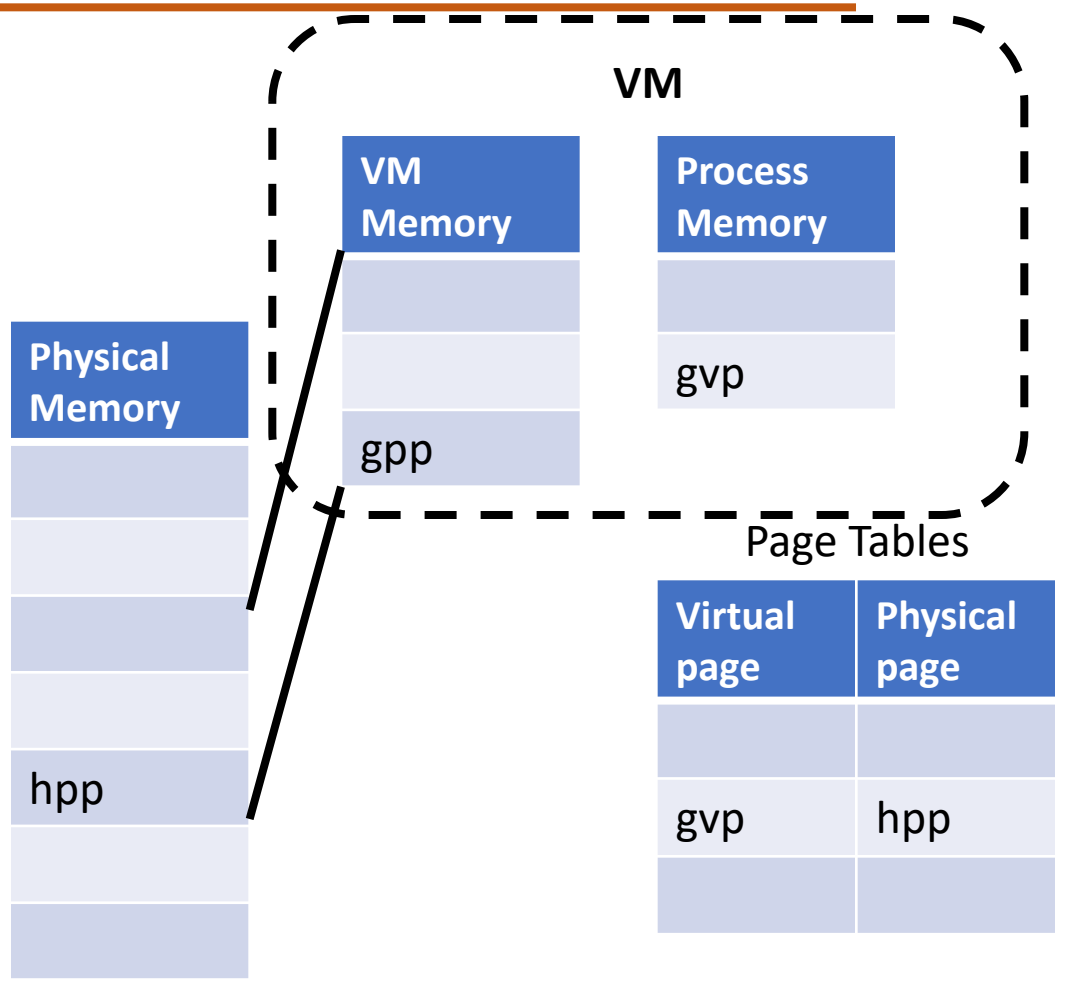
- Guest OS (e.g., Linux) is trying to define new page *gvp* for process
- It finds an unused page *gpp* in VM memory
- It tries to load (gvp,gpp) into page table
- hpp is a free physical page
- What happens?



CLOUD COMPUTING

Trap and Emulate Exercise Solution

1. Guest OS tries to modify page tables so that *gvp* points to *gpp*
2. Hypervisor traps this privileged instruction
3. Hypervisor maps (if not already done) *gpp* to a host physical page *hpp*
4. Loads page table to get *gvp* to point to *hpp*
5. Other page table operations (e.g., *read*) have to be trapped as well
6. We have skipped TLB manipulation



Binary Translation

Replacement of sensitive instructions by traps

Transparent virtualization technique
Complements trap and emulate virtualization
    Allows virtualization of x86 architecture
Invented by VMWare
    Mainframe architectures are virtualizable

Conceptually simple
    Replace *popf* instruction with trap to kernel and emulation
    Replace simple instructions with equivalent instructions
    Replacement is not done at run time; done before OS is run to create a modified OS image

Binary Translation (Cont.)

Translate each VM instruction to minimal set of host instructions on the fly

- Example: incl(%eax)
  - »leal mem0(%eax), %esi
  - »incl(%esi)

CPU state is kept in software structure (e.g., VCPU)

Memory is emulated/relocated

Privileged instructions gets translated to calls to emulation routine

Example: Qemu

Binary Translation (Additional Reading)

Dynamic binary translation is used in virtualizing the CPU by translating potentially dangerous (or non-virtualizable) instruction sequences one-by-one into safe instruction sequences.

It works like this:

- The monitor inspects the next sequence of instructions. An instruction sequence is typically defined as the next basic block, that is all instructions up to the next control transfer instruction such as a branch. There may be reasons to end a sequence earlier or go past a branch but for now lets assume we go to the next branch.
- Each instruction is translated and the translation is copied into a **translation cache.**

Binary Translation (Additional Reading)

❖Instructions are translated as follows:

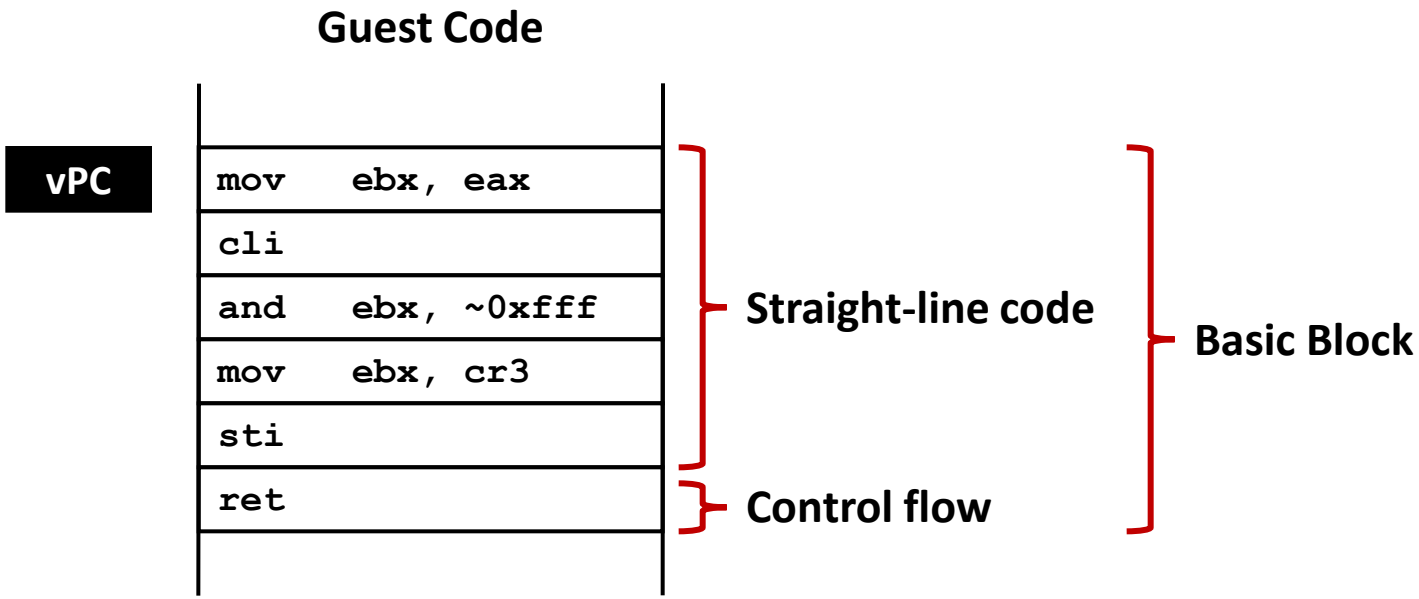
- Instructions which pose no problems can be copied into the translation cache with modification. We call these "**ident**" translations.
- Some simple dangerous instructions can be translated into a short sequence emulation code. This code is placed directly into the translation cache. We call this "**inline**" translation. An example is the modification of the **Interrupt Enable flag**.
- Other dangerous instructions need to be performed by emulation code in the monitor. For these instructions, calls to the monitor are made. These are called "Call-outs". An example of these is a change to the **page table base**.
- The branch ending the basic block needs a call out.

❖The monitor can now jump to the start of the translated basic block with the virtual registers in the hardware registers.

So dangerous instructions can be privileged instructions, non-virtualizable instructions, control flow, memory accesses.

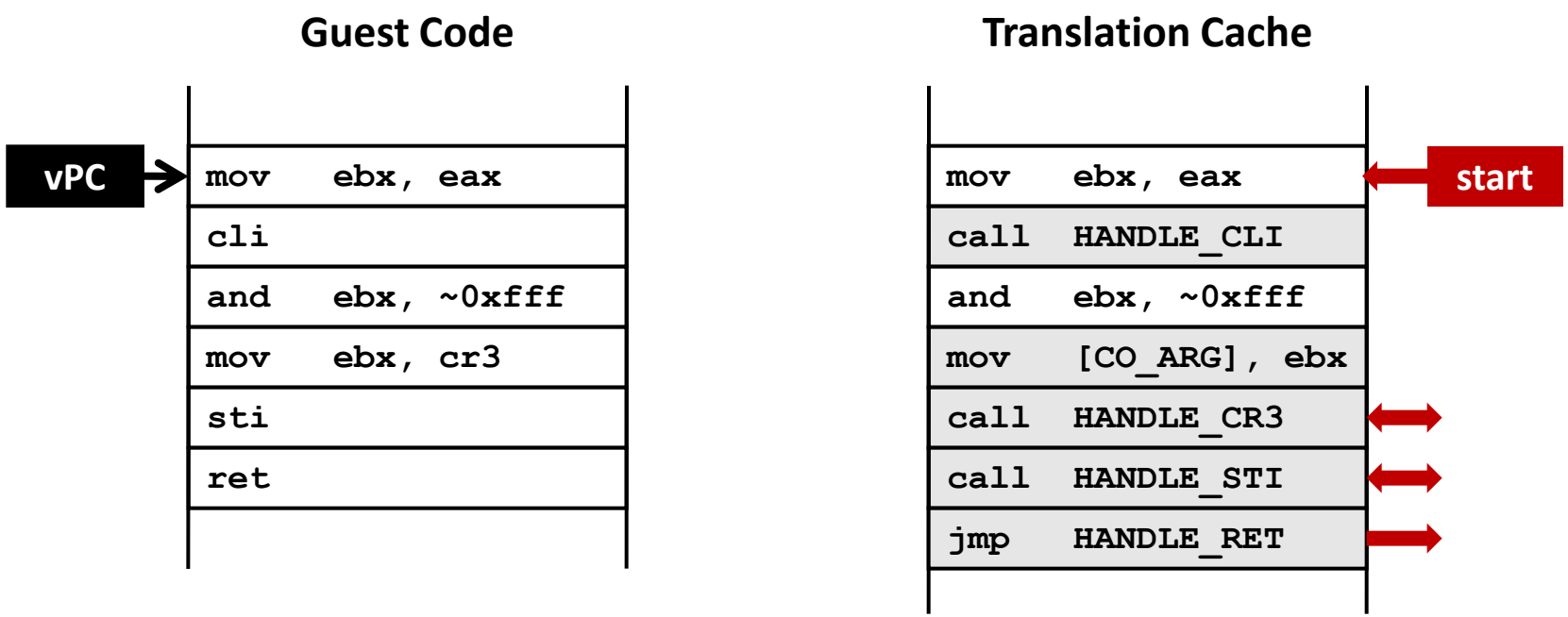
CLOUD COMPUTING

What are Basic Blocks?



CLOUD COMPUTING

Binary Translation (Cont.)



Source : https://slideplayer.com/slide/5346246/ Scott Devine, VMWare

Binary Translation Complexities

- Control flow changes
  - Translation can change addresses of instructions
  - Branches have to be re-translated
  - Keep track of branch addresses

```
isPrime:  mov   %ecx, %edi ; %ecx = %edi (a)
          mov   %esi, $2   ; i = 2
          cmp   %esi, %ecx ; is i >= a?
          jge   prime      ; jump if yes
```

```
isPrime':  mov %ecx, %edi    ; IDENT
           mov %esi, $2
           cmp %esi, %ecx
           jge (takenAddr)   ; JCC
           jmp (fallthrAddr)
```

Additional references

Binary Translation, Trap & Emulate
https://www.sciencedirect.com/topics/computer-science/binary-translation

http://www.cs.cmu.edu/~410-f06/lectures/L31_Virtualization.pdf

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Hardware Virtualization

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

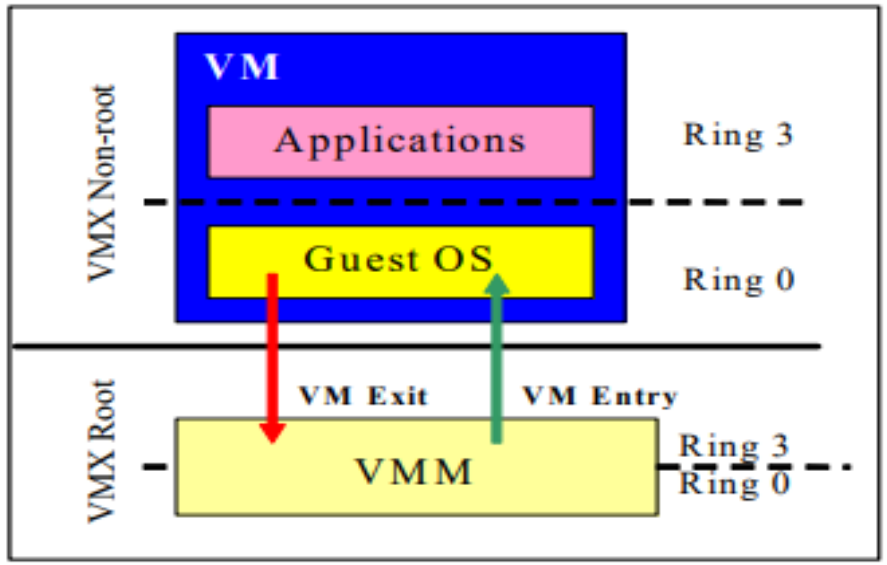
Techniques to virtualize x86

- Paravirtualization: rewrite guest OS code to be virtualizable
  - Guest OS won't invoke privileged operations, makes "hypercalls" to VMM
  - Needs OS source code changes, cannot work with unmodified OS
  - Example: Xen hypervisor
- Full virtualization: CPU instructions of guest OS are translated to be virtualizable
  - Sensitive instructions translated to trap to VMM
  - Dynamic (on the fly) binary translation, so works with unmodified OS
  - Higher overhead than paravirtualization
  - Example: VMWare workstation

CLOUD COMPUTING

Techniques to virtualize x86 (Cont.)

- Hardware assisted virtualization: KVM/QEMU in Linux
  - CPU has a special VMX mode of execution
  - Two kinds of VMX transitions: VMX Root and VMX Non-Root
    - X86 has 4 rings in both VMX root and Non-root modes
- VMM enters VMX mode to run guest OS in (special) ring 0
- Exit back to VMM on triggers (VMM retains control)



Processor behavior in VMX non-root operation is restricted and modified to facilitate virtualization. Instead of their ordinary operation, certain instructions (such as the new VMCALL instruction) and events cause VM exits to the VMM. Because these VM exits replace ordinary behavior, the functionality of software in VMX non-root operation is limited. It is this limitation that allows the VMM to retain control of processor resources.

Workaround for x86-32: Paravirtualization

- Co-design of guest OS and hypervisor
  - Advantage: Simplicity, work around corner cases
  - Disadvantage: Need **modified** guest OS
  - Example: Xen hypervisor

- Trick to virtualize x86-32:
  - Block all 17 instructions that are sensitive but not privileged
    - Just modify the guest OS to #undef them
  - Instead provide ***hypercalls*** from guest OS to VMM
    - Hypercalls are like system calls but from guest OS to VMM

Hardware assisted virtualization: Intel VT-x/AMD-v for x86-64

Two challenges of virtualizing x86-64 ISA:

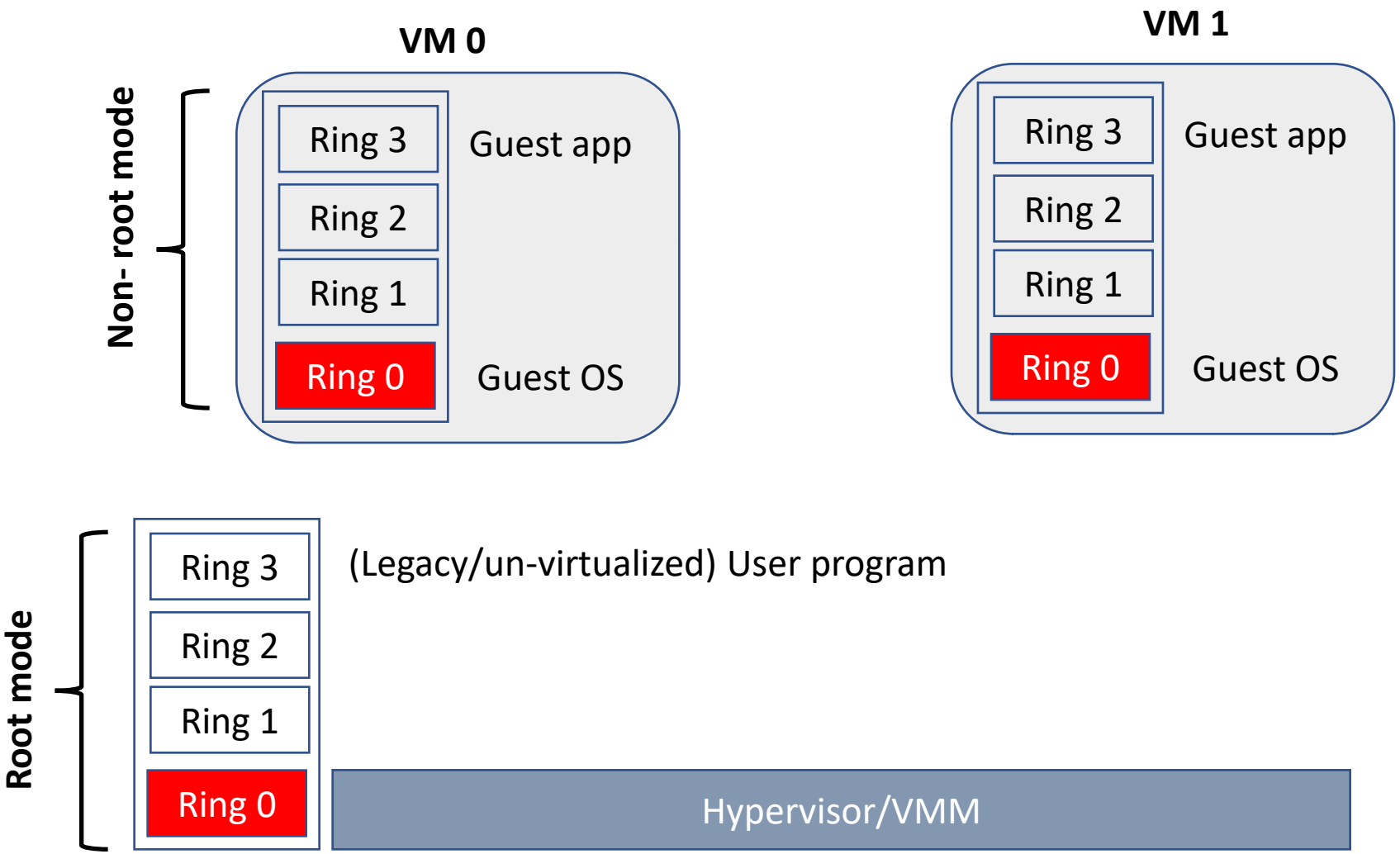
- How to hide system/privileged state from the VM?
- How to ensure a VM cannot directly change system state (e.g., interrupt flags) of the processor?

Hardware assisted virtualization (Cont.)

- Solution idea:
  - Two *new* modes of operation: root and non-root
    - Each mode has complete set of execution rings (0-3)
    - New instructions to switch between modes
  - H/W state is duplicated for each operation mode
  - Hypervisor runs in root mode and VMs in non-root mode
  - When any “sensitive” instruction executed in non-root mode it either (1) executed by processor on duplicated state or (2) trap to hypervisor

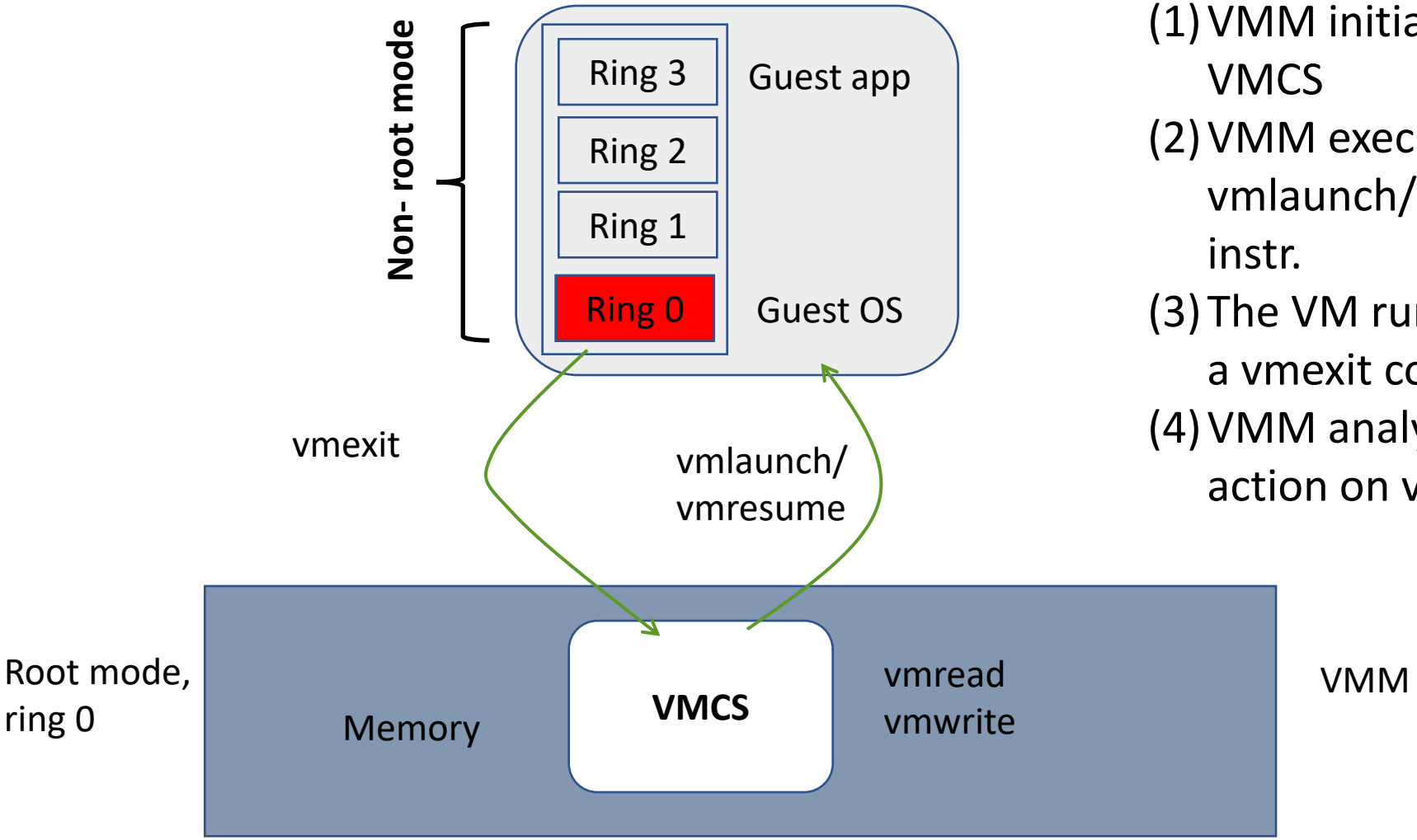
CLOUD COMPUTING

Hardware assisted virtualization (Cont.)



CLOUD COMPUTING

Using Intel VT-x/AMD-v (KVM way)



(1) VMM initializes/modifies VMCS
(2) VMM executes vmlaunch/vmresume instr.
(3) The VM runs until it hits a vmexit condition
(4) VMM analyzes and takes action on vmexit

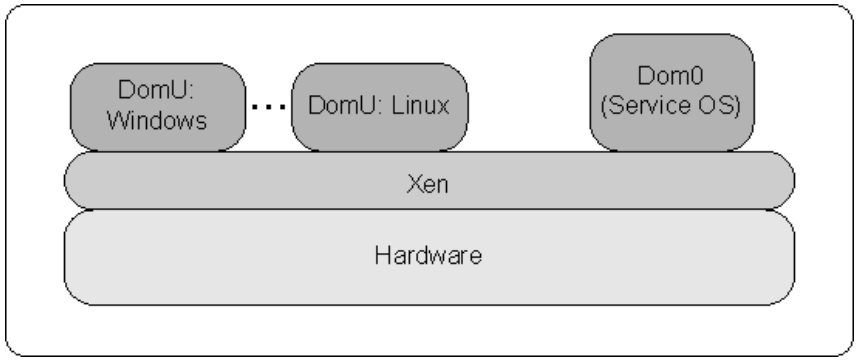
Current status of H/W assist

Almost all VMMs make use of h/w assist today
KVM is integrated part of Linux and makes heavy use of h/w assist
Even Xen and VMWare workstation uses it

CLOUD COMPUTING

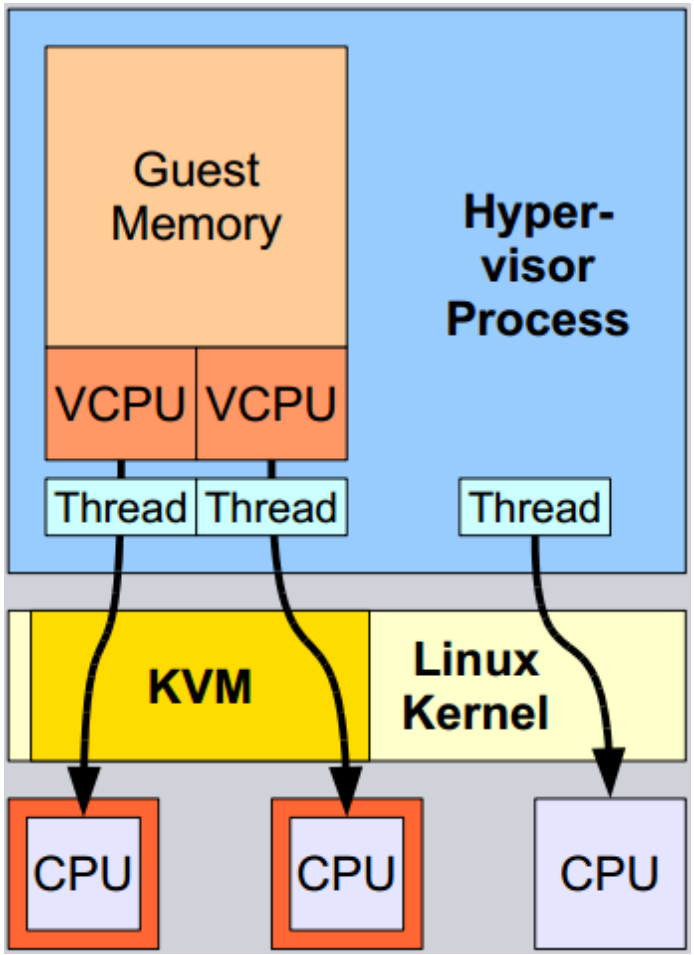
Xen architecture

- Paravirtualization hypervisor
- VMs called *domains*
- Dom0
  - Special domain
  - Based upon Linux
  - Handles all I/O
  - Windows: uses filter drivers
- Xen hypervisor
  - All functions other than I/O
  - Trap and emulate
  - Binary translation
    - Sensitive instructions replaced by hypervisor calls



KVM Architecture – Processor Support

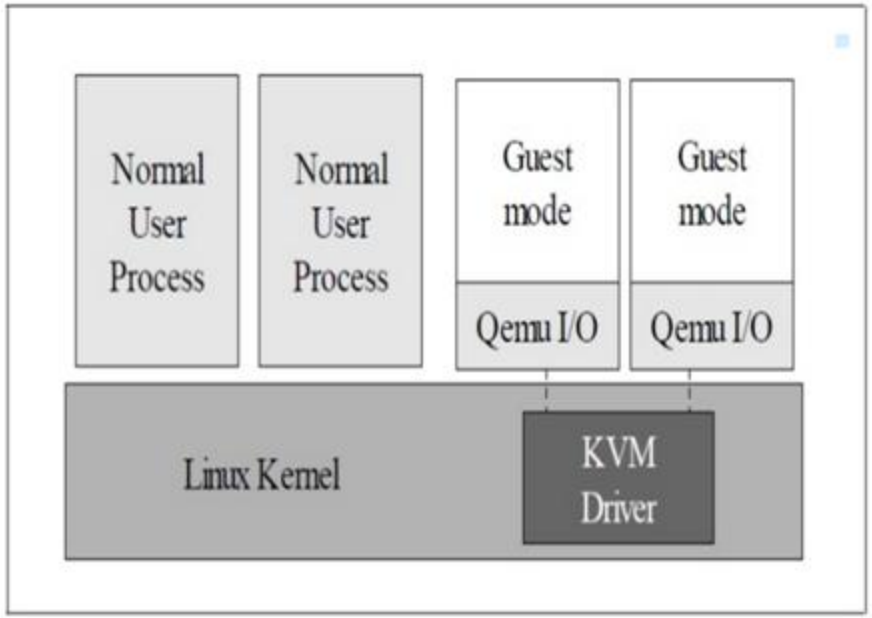
- Extends Linux kernel to add hypervisor functionality
  - Leverage Linux code for scheduling, paging, …
  - Uses VT-x, EPT
- Each VM appears as a Linux process



CLOUD COMPUTING

Linux as a Hypervisor

Paravirtualization supported via paravirtualization drivers



Full virtualization support achieved via Qemu

- Dynamic binary translator
- Emulates I/O instructions

QEMU is a type 2 hypervisor that runs within user space and performs virtual hardware emulation, whereas **KVM** is a type 1 hypervisor that runs in kernel space, that allows a user space program access to the hardware virtualization features of various processors.

QEMU

- QEMU (short for Quick EMUlator) is a free and open-source emulator and virtualizer that can perform hardware virtualization.
- QEMU is a hosted virtual machine monitor: it emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems.
- It also can be used with Kernel-based Virtual Machine (KVM) to run virtual machines at near-native speed (by taking advantage of hardware extensions such as Intel VT-x).
- QEMU can also do emulation for user-level processes, allowing applications compiled for one architecture to run on another.

QEMU

- Qemu is a very old virtualization technology used to virtualize system components and run operating systems on it.
  - Before KVM and XEN QEMU was used heavily but it can not race with VMWARE or VIRTUAL PC.
  - But with the KVM, Qemu get superfast speed for computing by using hardware-based virtualization. QEMU acts as a hardware supplier and KVM is the CPU.
  - KVM resides in Linux kernel and there is a little configuration for it. A virtualization configuration is made on the QEMU.

Hosted Hypervisor

❖A **hosted hypervisor** is installed on a host computer, which already has an operating system installed. It runs as an application like other software on the computer.

❖Most hosted hypervisors can manage and run multiple VMs at one time.

❖While **bare metal hypervisors** run directly on the computing hardware, **hosted hypervisors** run within the operating system of the host machine.

❖Hosted hypervisors have higher latency than bare metal hypervisors because requests between the hardware and the hypervisor must pass through the extra layer of the OS.

CLOUD COMPUTING

Additional Reading

www.vmware.com/pdf/asplos235_adams.pdf
https://www.cse.iitb.ac.in/~cs695/slides_pdf/04-hwvirt-kvmqemu.pdf
https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/virtualization-tech-converged-application-platforms-paper.pdf

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Virtualization – Memory and I/O

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

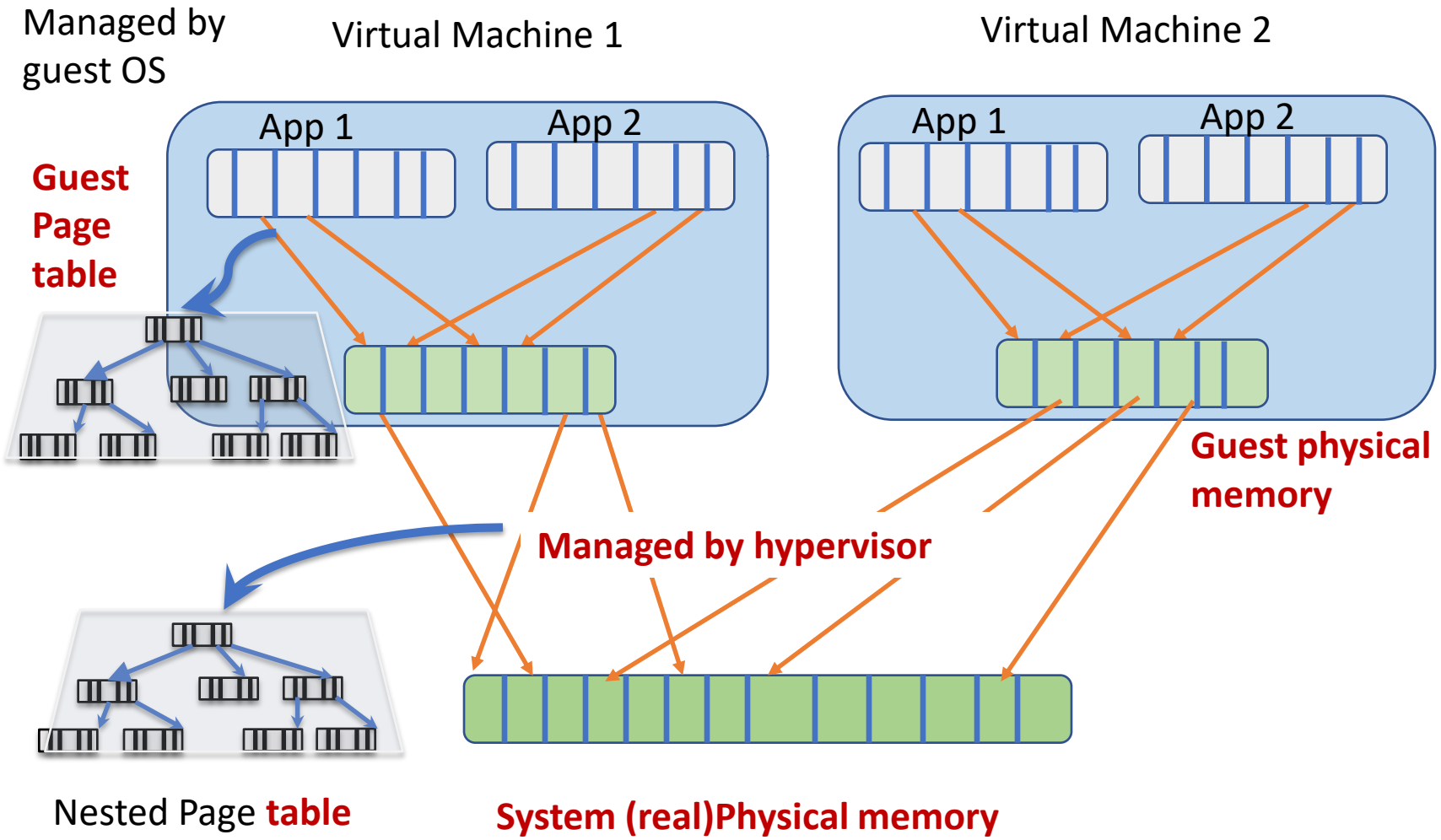
1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Requirement for memory virtualization

- **No** direct access to physical memory from VM
- **Only** hypervisor should manage physical memory
  - Why?
- But, fool the guest OS to think that it is accessing physical memory

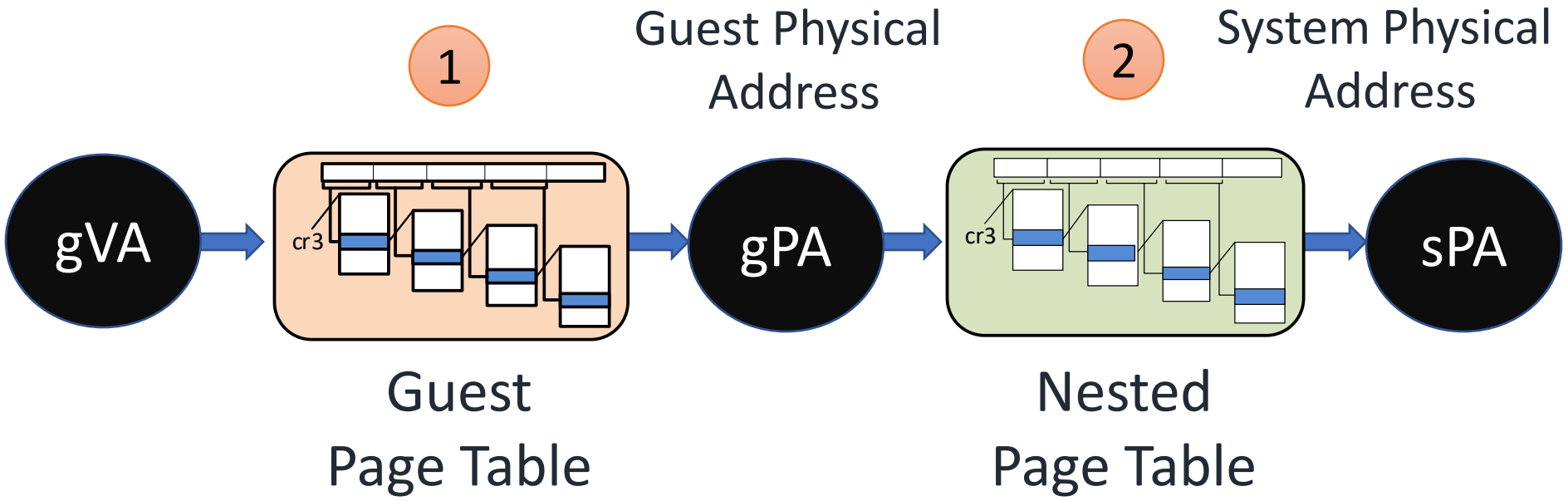
CLOUD COMPUTING

Virtualizing virtual memory



CLOUD COMPUTING

Virtualizing virtual memory (Cont.)



Two levels of address translation on each memory access by an application running inside a VM

Implementing two level address translation

- Shadow page table
  - Each page table of the guest OS has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table.
  - Shadow paging was developed as a software-only technique to virtualize memory before hardware support was implemented.

- Nested/Extended page table
  - Hardware support
  - set up by the hypervisor to perform a second level address translation

Shadow page table

- **Idea**: Let hypervisor “create” a shadow page table that maps guest VA to host PA directly
  - Made by combining guest page table with system page table
  - Hypervisor makes the *cr3* point to the shadow page table
  - A hardware or software page table walker (PTW) walks the page table
  - Control Register CR3 points to the root of page table
  - Translation from Virtual Page Number (VPN) to Physical page frame number or fault occurs

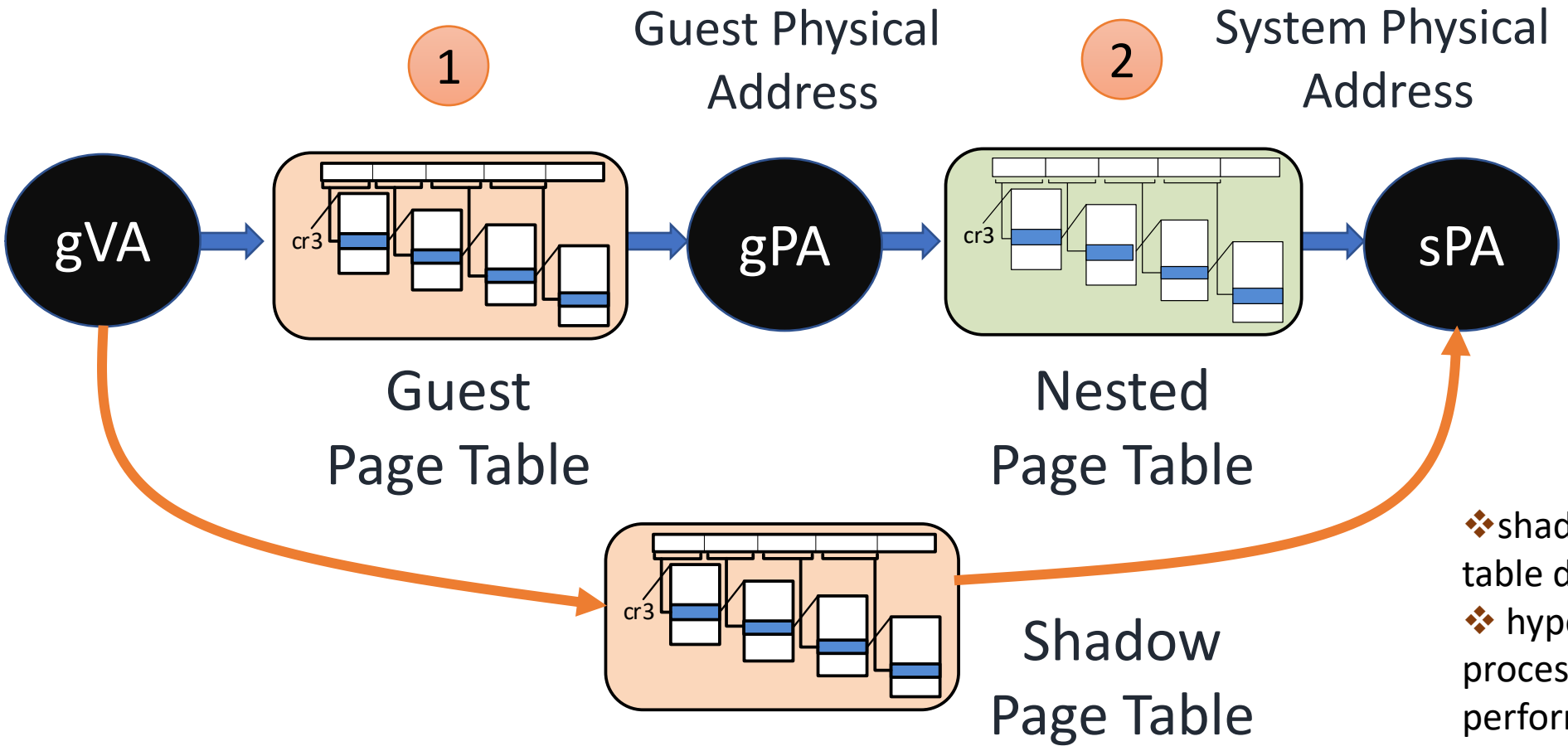
Ref: https://www.cse.iitk.ac.in/users/biswap/CASS18/VM.pdf

Shadow page table

- Software page walker is a OS handler that walks the page table
  - slower than hardware page table walker due to large address translation overhead
  - Example: SPARC (Sun/Oracle Machines)
- Hardware page walker generates load-like “instructions” to access page table
  - Example: x86 and ARM processors
- During the page walk, the page walker encounters physical addresses in CR3 register and in page table entries which point to the next level of the walk. The page walk ends when the data or leaf page is reached.
- Address translation is a very memory intensive operation because the page walker must access the memory hierarchy many times. To reduce this overhead, processors automatically store recent translations in an internal translation look-aside buffer (TLB).

CLOUD COMPUTING

Shadow page table



- shadow version of page table derived from gPT.
- hypervisor forces the processor to use sPT to perform address translation.

Shadow page tables

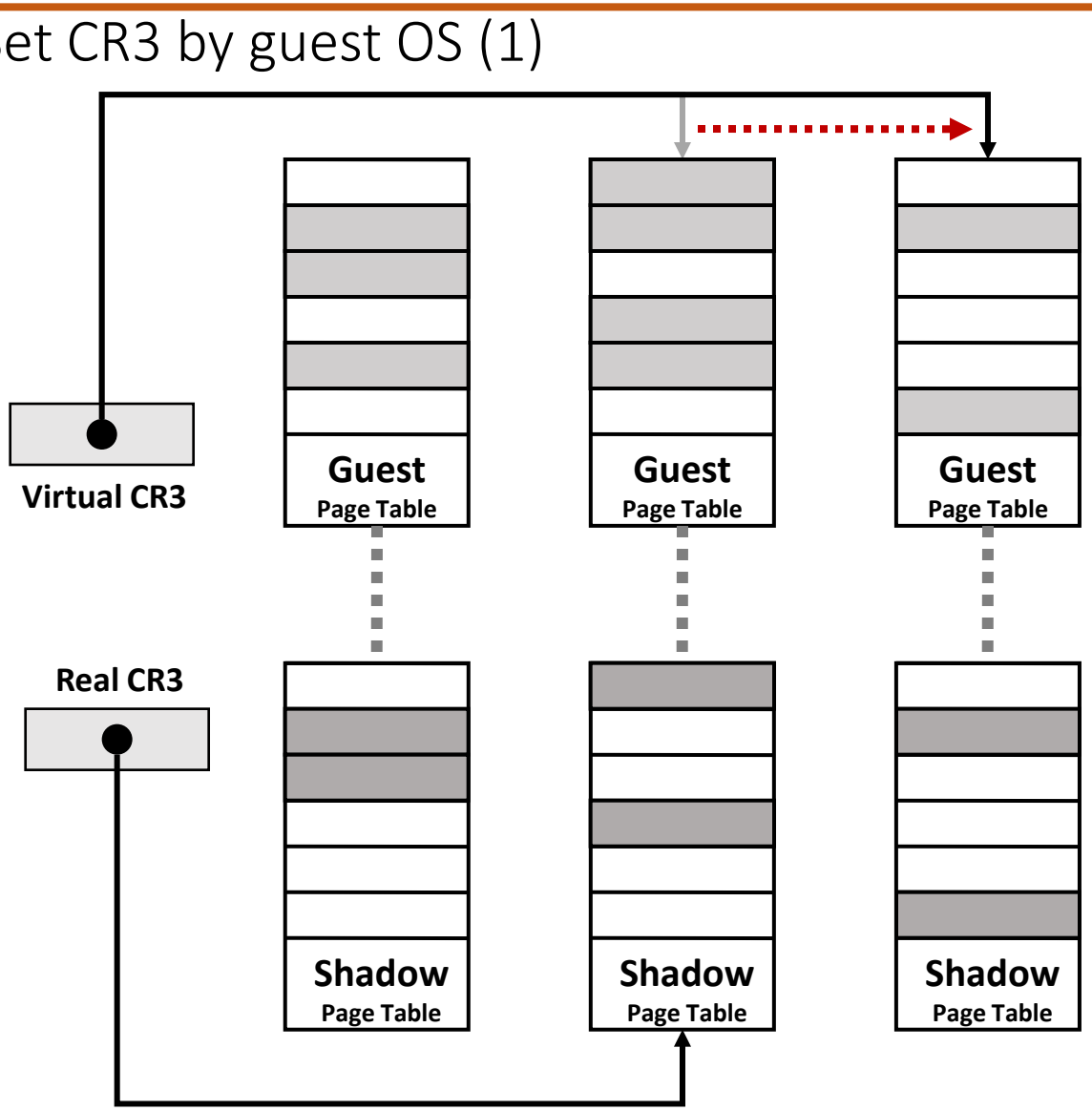
- VMM maintains shadow page tables that map guest-virtual pages directly to machine pages.
- Guest modifications to V->P tables synced to VMM V->M shadow page tables.
  - Guest OS page tables marked as read-only.
  - Modifications of page tables by guest OS -> trapped to VMM.
  - Shadow page tables synced to the guest OS tables

Shadow page tables (Cont.)

- Software-based techniques maintain a shadow version of page table derived from guest page table (gPT).
- When the guest is active, the hypervisor forces the processor to use the shadow page table (sPT) to perform address translation.
- The sPT is not visible to the guest.

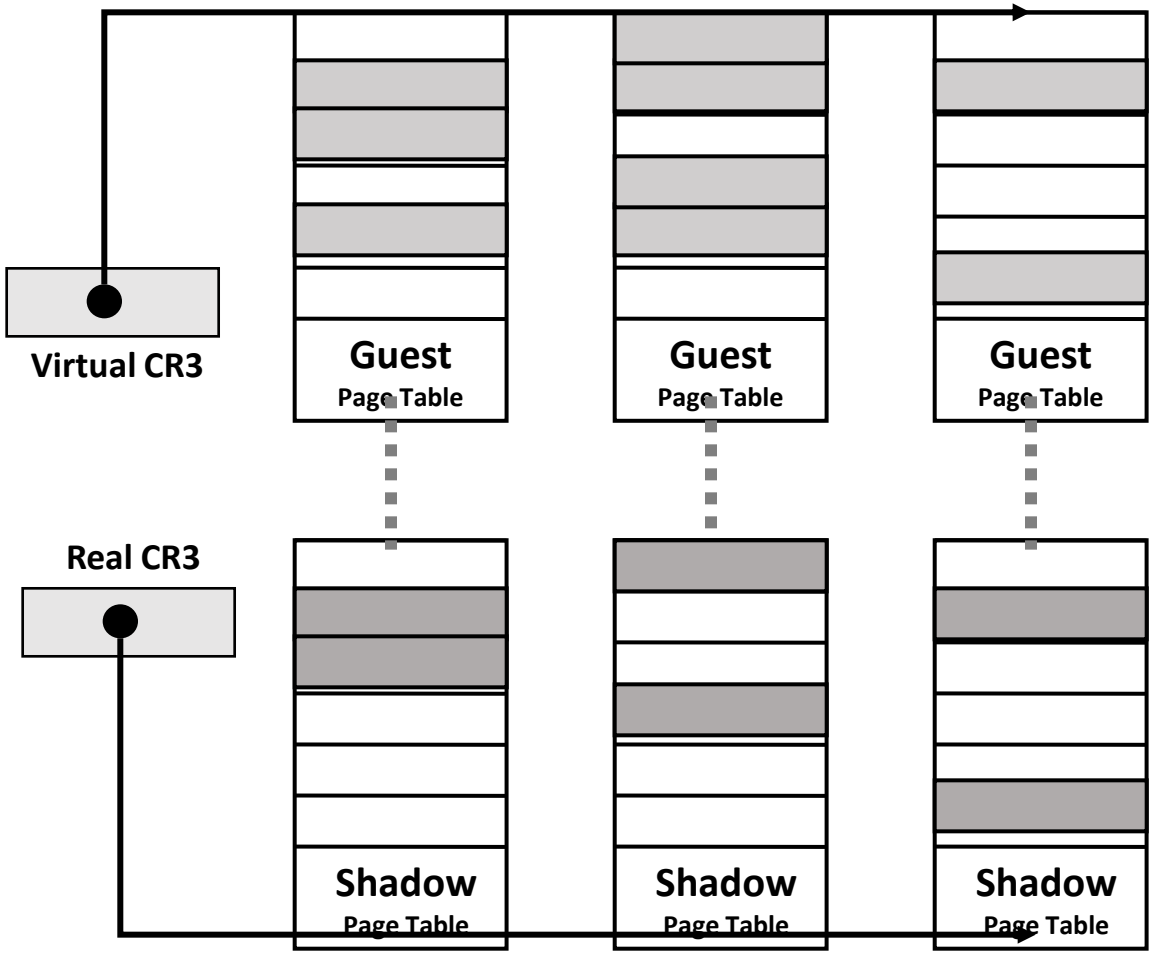
CLOUD COMPUTING

Shadow page tables (Cont.)



Shadow page tables (Cont.)

Set CR3 by guest OS (2)



Challenge of Shadow page table

- How to create a shadow page table?
  - Anytime guest OS modifies guest page table hypervisor needs to update shadow page table
  - Solution: Write protect guest page table
    - Any write access to guest page table would generate page fault → trap to hypervisor
  - Drawback: Many page faults for application that alters page tables
- For every guest application there is one shadow page table
- Every time guest application context switches trap to hypervisor to change cr3 to point to new shadow page table
- Maintaining consistency between guest page tables and shadow page tables leads to an overhead: VMM traps
- Loss of performance due to TLB flush on every “world-switch”.
- Memory overhead due to shadow copying of guest page tables.

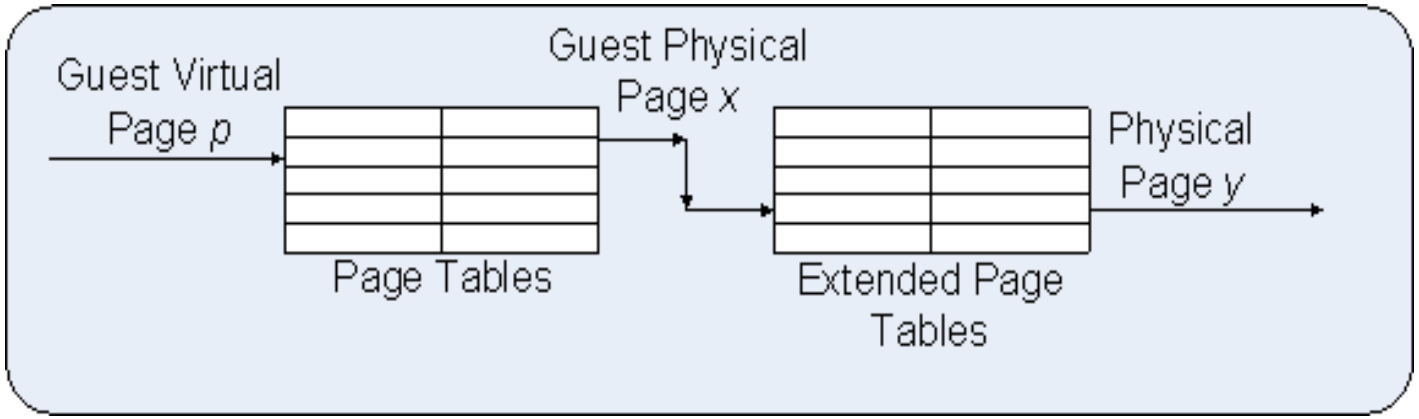
Nested/Extended page table

- **Idea**: Make hardware aware of two levels of address translation and guest and nested page table
  - Make hardware page walker walk both guest page table and nested page table on a TLB miss → Two dimensional page walk
  - Two cr3s : gCR3 → gPT; nCR3 → nPT
  - Eliminates the need of shadow page table

Ref: http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf

CLOUD COMPUTING

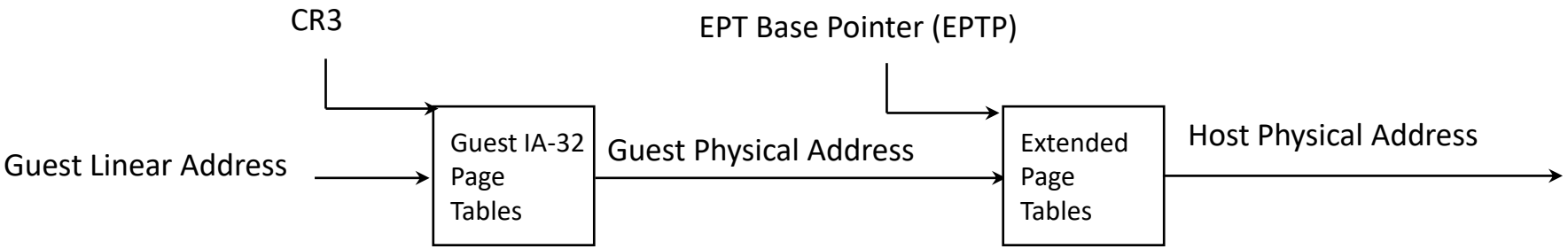
Extended page tables



Guest manages page tables
Host manages Extended Page Tables (EPT)

CLOUD COMPUTING

What is EPT?



- Extended Page Table
- A new page-table structure, under the control of the VMM
  - Defines mapping between guest- and host-physical addresses
  - EPT base pointer (new VMCS field) points to the EPT page tables
  - EPT (optionally) activated on VM entry, deactivated on VM exit
- Guest has full control over its own IA-32 page tables
  - No VM exits due to guest page faults, INVLPG, or CR3 changes

Virtual Machine Control Structure (VMCS) Field IDs used by the Hypervisor framework read and write functions.

when the guest removes a translation, it executes INVLPG to invalidate that translation in the TLB. The hypervisor intercepts this operation; it then removes the corresponding translation in sPT and executes INVLPG for the removed translation.

What is EPT?

Extended Page Tables: Performance

- **Estimated EPT benefit is very dependent on workload**
  - Typical benefit estimated up to 20%[1]
  - Outliers exist (e.g., forkwait, Cygwin gcc, > 40%)
  - Benefit increases with number of virtual CPUs (relative to MP page table virtualization algorithm)
- **Secondary benefits of EPT:**
  - No need for complex page table virtualization algorithm
  - Reduced memory footprint compared with shadow page-table algorithms
    - Shadow page tables required for each guest user process
    - Single EPT supports entire VM

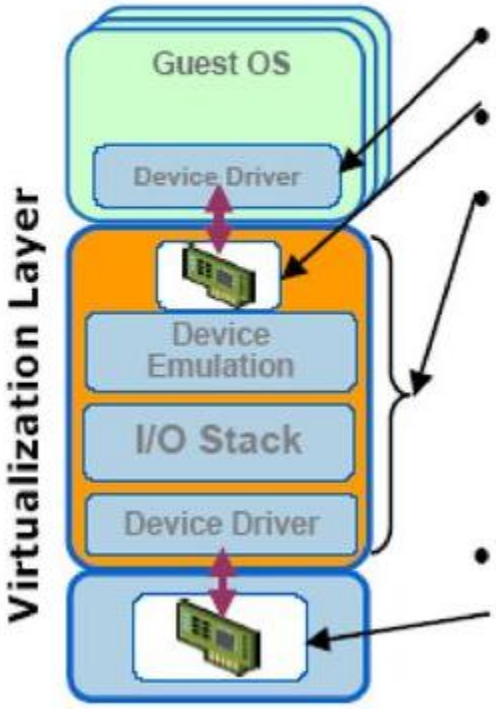
EPT improves memory virtualization performance

I/O virtualization

- I/O virtualization (IOV), or input/output virtualization, is technology that uses software to abstract upper-layer protocols from physical connections or physical transports.
- I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.
- Three ways to implement I/O virtualization:
  - full device emulation, para-virtualization, and direct I/O.
- Full device emulates well-known, real-world devices.
  - All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software.
  - This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.

Device emulation for I/O Virtualization

Current virtual I/O devices



- Guest device driver
- Virtual device
- Virtualization layer
  - emulates the virtual device
  - remaps guest and real I/O addresses
  - multiplexes and drives the physical device
  - I/O features, *e.g.*, COW disks; → Copy on write disk
- Real device
  - may be different from virtual device

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

Para-I/O-Virtualization

❖It is also known as the **split driver model** consisting of a frontend driver and a backend driver.

❖The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory.

❖The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.

❖Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

❖Para virtualization method of I/O virtualization is typically used in Xen

Direct I/O Virtualization

- Direct I/O virtualization lets the VM access devices directly.
- It can achieve close-to-native performance without high CPU costs.
- Enables a device to directly DMA to/from host memory.
  - This technique thus provides an ability to bypass the VMM's I/O emulation layer and therefore shows improved throughput for the VM's.
- The Intel VT-x technology allows a VM to have direct access to a physical address (if so configured by the VMM).
  - This ability allows a device driver within a VM to be able to write directly to registers IO device (such as configuring DMA descriptors).
  - Intel VT-d provides a similar capability for IO devices to be able to write directly to the memory space of a virtual machine, for example a DMA operation.

Direct I/O Virtualization (Contd)

❖The mechanism for doing direct assignment varies from one vendor to another. However, the basic idea is that the VMM utilizes and configures technologies such as Intel VT-x and Intel VT-d to perform address translation when sending data to and from an IO device.

The main concern with direct assignment is that it has limited scalability, a physical device can only be assigned to one VM.

Ref: https://virtualizationdeepdive.wordpress.com/about/6-2/

CLOUD COMPUTING

I/O virtualization Advantages

Flexibility: Since I/O virtualization involves abstracting the upper layer protocols from the underlying physical connections, it offers greater flexibility, utilization and faster provisioning in comparison with normal NIC and HBA cards.

Cost minimization: I/O virtualization methodology involves using fewer cables, cards and switch ports without compromising on network I/O performance.

Increased density: I/O virtualization increases the practical density of I/O by allowing more connections to exist in a given space.

Minimizing cables: The I/O virtualization helps to reduce the multiple cables needed to connect servers to storage and network.

Xen I/O Virtualization

- Xen does not emulate hardware devices
- Exposes device abstractions for simplicity and performance
- I/O data transferred to/from guest via Xen using shared-memory buffers
- Virtualized interrupts: light-weight event delivery mechanism from Xen-guest
  - Update a bitmap in shared memory
  - Optional call-back handlers registered by O/S

Additional References

https://www.cse.iitb.ac.in/~cs695/slides_pdf/06-memvirt.pdf
https://www.cse.iitb.ac.in/~cs695/papers/iovirt.pdf

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Virtualization
Goldberg/Popek Principles

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Behavior & Control sensitive instructions – Goldberg/Popek Principles

- The fundamental assumption for trap and emulate virtualization (aka classic virtualization) is
  - All sensitive instructions can be trapped
  - Sensitive instructions = instructions that must be emulated
- **Behavior sensitive**: result of instruction depends upon privilege level
  - If guest OS executes these instructions at lower privilege levels, result will be wrong
- **Control sensitive**: results that could change processor privilege level or change processor state
  - Like modify the processor registers and page tables.
- Formal Requirements for Virtualizable Third-Generation Architectures by Gerald Popek and Robert Goldberg, Comm. ACM, 1974

Popek and Goldberg principles

Popek and Goldberg in 1974 defined the requirements for an architecture to efficiently support virtualization.
They enunciated three requirements for the VMs provided by a VMM:
Equivalence: A program running on a VM should behave essentially identically to a program executing directly on the hardware.
Resource Control: The VMM must be in total control of the virtualized resources.
Efficiency: The great majority of the machine instructions must be executed without intervention of the VMM (i.e. without trap).

Ref:
https://cis.temple.edu/~ingargio/cis307/readings/virtualization.html#:~:text=Privileged%20Instructions%3A%20instructions%20that%20if,%2CSIDT%2CSLDT%2CSMSW.

Goldberg/Popek Principles (Contd.)

Virtual machine: Complete compute environment with its own isolated processing capabilities, memory and communication channels.

Virtual machine is an efficient isolated duplicate of the physical machine [Goldberg, Popek]

Virtual Machine Monitor/Hypervisor: System software that creates and manages virtual machines

Desirable qualities of a VMM [Goldberg/Popek]:

Equivalence: Virtual m/c interface similar to real m/c

Safety/Isolation: Each VM should be isolated from other

Low performance overhead: Perf. close to real m/c

Popek and Goldberg principles (Cont.)

Popek and Goldberg introduce a classification of some instructions of an ISA into 3 different groups:

Privileged instructions

Those that trap if the processor is in user mode and do not trap if it is in system mode (i.e kernel or supervisor mode).

Control sensitive instructions

Those that attempt to change the configuration of resources in the system i.e. that modify the system registers.

Examples of such instructions in x86 are: PUSHF,POPF,SGDT,SIDT,SLDT,SMSW.

Behavior sensitive instructions

Those whose behavior or result depends on the mode or configuration of the hardware (the content of the relocation register or the processor's mode).

Examples of such instructions on x86 are POP, PUSH, CALL, JMP, INT n, RET, LAR, LSL, VERR, VERW, MOV.

CLOUD COMPUTING

Popek and Goldberg Theorems

The Popek-Goldberg Theorem is a sufficient condition to build a VMM, depending on relations between control-sensitive,behavior-sensitive and privileged instructions. The main result of Popek and Goldberg's analysis can be expressed as follows.

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.

- The theorem states that to build a VMM it is sufficient that all instructions that could affect the correct functioning of the VMM (sensitive instructions) always trap and pass control to the VMM.
- Non-privileged instructions must instead be executed natively (i.e., efficiently).

Ref:
https://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_requirements

Popek and Goldberg Theorems (Cont.)

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.

• If architectures cannot be virtualized in the **classic** way, use **binary translation**, which replaces the sensitive instructions that do not generate traps

•Recursive virtualization is the ability to install a VM within a VM or the conditions under which a VMM that can run on a copy of itself can be built.

Theorem 3: A hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions.

Ref:

https://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_require

x86 virtualization

- X86 ISA does not meet the Popek & Goldberg requirements for virtualization
- ISA contains 17+ sensitive , unprivileged instructions:
  - SGDT, SIDT, SLDT, SMSW, PUSHF, POPF, LAR, LSL, VERR, VERW, POP, PUSH, CALL, JMP, INT, RET, STR, MOV
- Most simply reveal that the kernel is running in user mode
  - PUSHF
- Some execute inaccurately
  - POPF
- Virtualization is still possible, requires workarounds

Additional Reading

- https://enacademic.com/dic.nsf/enwiki/480245
- https://www.cs.cmu.edu/~410-s14/lectures/L30_Virtualization.pdf
- https://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_requirements#:~:text=The%20Popek%20and%20Goldberg%20virtualization,for%20Virtualizable%20Third%20Generation%20Architectures%22.
- https://blog.acolyer.org/2016/02/19/formal-requirements-for-virtualizable-third-generation-architectures/
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4815&rep=rep1&type=pdf
- https://www.researchgate.net/publication/252019491_Recursive_virtual_machines_for_advanced_security_mechanisms

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Virtualization – VM Migration

Venkatesh Prasad
Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Live migration process of VMs

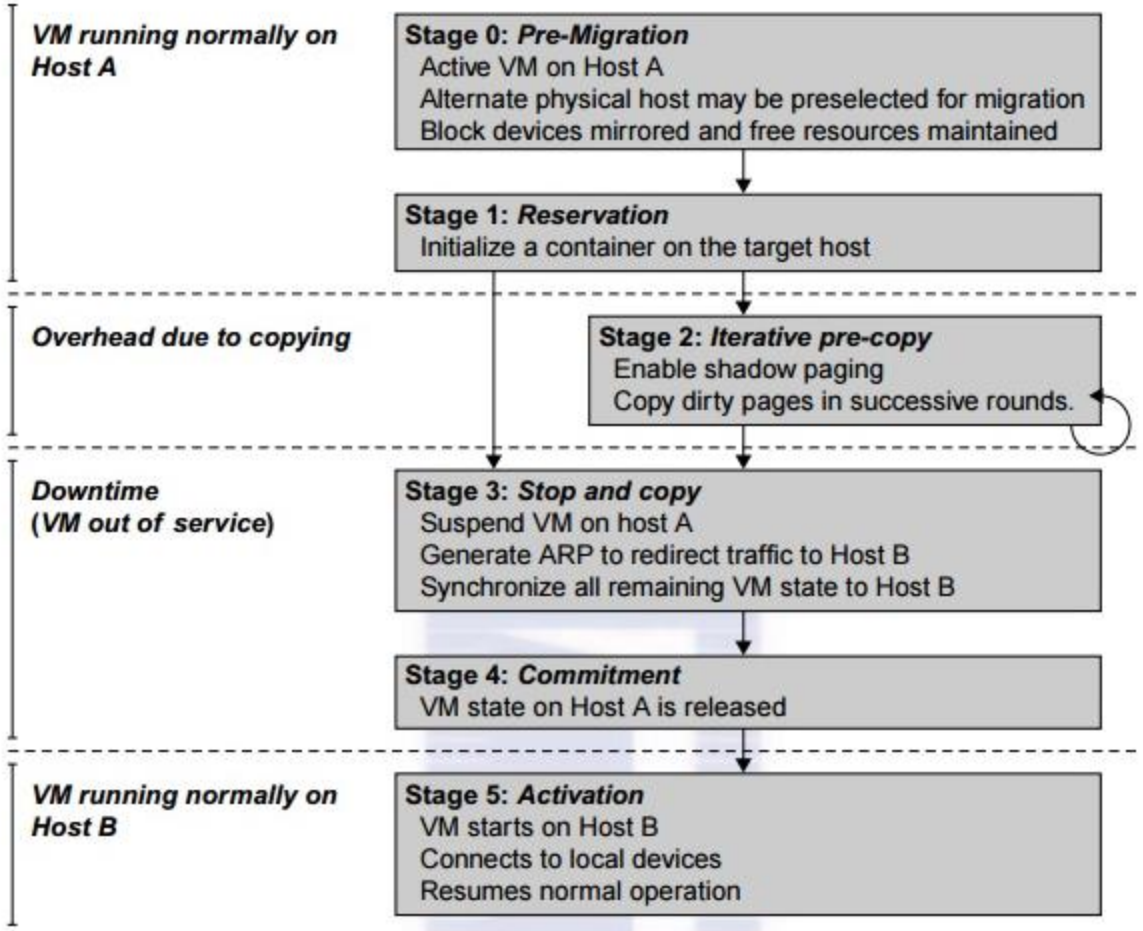
- The live migration of VMs allows workloads of one node to transfer to another node.
- However, it does not guarantee that VMs can randomly migrate among themselves. In fact, the potential overhead caused by live migrations of VMs cannot be ignored.
- Challenge is to determine how to design migration strategies to implement green computing without influencing the performance of clusters.
- Another advantage of virtualization is load balancing of applications in a virtual cluster.

Live migration process of VMs (Cont.)

- Load balancing can be achieved using the load index and frequency of user logins.
- The automatic scale-up and scale-down mechanism of a virtual cluster can be implemented based on this model.
- Consequently, we can increase the resource utilization of nodes and shorten the response time of systems.
- Mapping VMs onto the most appropriate physical node should promote performance.
- Dynamically adjusting loads among nodes by live migration of VMs is desired, when the loads on cluster nodes become quite unbalanced.

CLOUD COMPUTING

Live migration process of a VM from one host to another



Six Steps involved in Live migration of a VM

Steps 0 and 1: Start migration. This step makes preparations for the migration, including determining the migrating VM and the destination host.

•Although users could manually make a VM migrate to an appointed host, in most circumstances, the migration is automatically started by strategies such as load balancing and server consolidation.

Six Steps involved in Live migration of a VM (Cont.)

Step 2: Transfer memory. Since the whole execution state of the VM is stored in memory, sending the VM's memory to the destination node ensures continuity of the service provided by the VM.

- All of the memory data is transferred in the first round, and then the migration controller recopies the memory data which is changed in the last round.
- These steps keep iterating until the dirty portion of the memory is small enough to handle the final copy.
- Although pre-copying memory is performed iteratively, the execution of programs is not obviously interrupted.

Six Steps involved in Live migration of a VM (Cont.)

Step 3: Suspend the VM and copy the last portion of the data.

- The migrating VM's execution is suspended when the last round's memory data is transferred.
- Other non-memory data such as CPU and network states should be sent as well. During this step, the VM is stopped and its applications will no longer run. This "service unavailable" time is called the "downtime" of migration, which should be as short as possible so that it can be negligible to users.

Six Steps involved in Live migration of a VM (Cont.)

Steps 4 and 5: Commit and activate the new host.

- After all the needed data is copied, on the destination host, the VM reloads the states and recovers the execution of programs in it, and the service provided by this VM continues.
- Then the network connection is redirected to the new VM and the dependency to the source host is cleared.
- The whole migration process finishes by removing the original VM from the source host.

Issues with VM migration

Memory Migration: This is one of the most important aspects of VM migration.

- Memory migration can be in a range of hundreds of megabytes to a few gigabytes in a typical system today, and it needs to be done in an efficient manner.
- The Internet Suspend-Resume (ISR) technique **exploits temporal locality** as memory states are likely to have considerable overlap in the suspended and the resumed instances of a VM.
- Temporal locality refers to the fact that the memory states differ only by the amount of work done since a VM was last suspended before being initiated for migration.

Ref: https://www.brainkart.com/article/Migration-of-Memory,-Files,-and-Network-Resources_11346/

Issues with VM migration (Cont.)

File System Migration: To support VM migration, a system must provide each VM with a consistent, location-independent view of the file system that is available on all hosts. A simple way to achieve this is to provide each VM with its own virtual disk which the file system is mapped to and transport the contents of this virtual disk along with the other states of the VM.

However, due to the current trend of high capacity disks, migration of the contents of an entire disk over a network is not a viable solution.

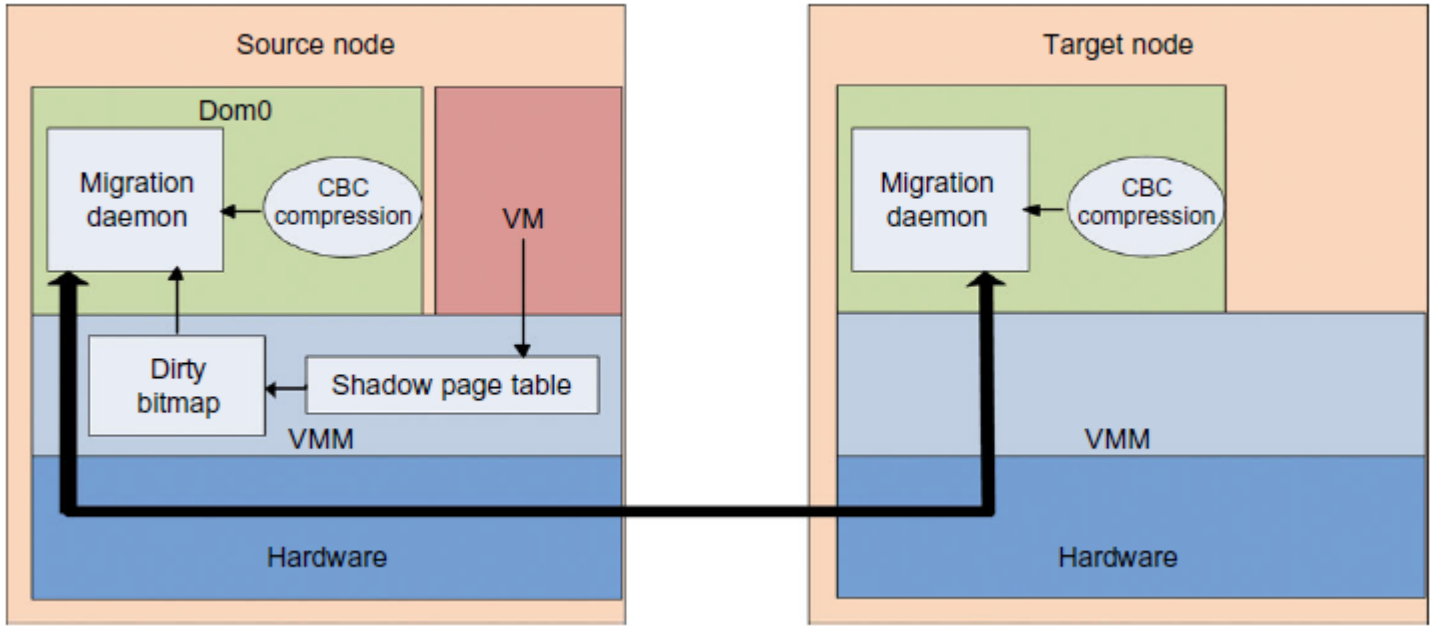
Another way is to have a global file system across all machines where a VM could be located. This way removes the need to copy files from one machine to another because all files are network accessible.

Issues with VM migration (Cont.)

Network Migration: migrating VM should maintain all open network connections without relying on forwarding mechanisms on the original host or on support from mobility or redirection mechanisms.

- To enable remote systems to locate and communicate with a VM, each VM must be assigned a virtual IP address known to other entities. This address can be distinct from the IP address of the host machine where the VM is currently located.
- Each VM can also have its own distinct virtual MAC address.
- The VMM maintains a mapping of the virtual IP and MAC addresses to their corresponding VMs.
- In general, a migrating VM includes all the protocol states and carries its IP address with it.

Example: Live migration of VM between two Xen-enabled hosts



Domain 0 (or Dom0) performs tasks to create, terminate, or migrate to another host. Xen uses a send/recv model to transfer states across VMs.

Live migration of VM between two Xen-enabled hosts explained

- Migration daemons running in the management VMs are responsible for performing migration.
- Characteristic Based Compression (**CBC**) algorithm compresses the memory pages adaptively.
  - **Adaptive compression** is a type of data **compression** which changes compression algorithms based on the type of data being compressed.
- Shadow page tables in the VMM layer trace modifications to the memory page in migrated VMs during the precopy phase. Corresponding flags are set in a dirty bitmap
- At the start of each precopy round, the bitmap is sent to the migration daemon. Then, the bitmap is cleared and the shadow page tables are destroyed and re-created in the next round.
- The system resides in Xen's management VM. Memory pages denoted by bitmap are extracted and compressed before they are sent to the destination.
- The compressed data is then decompressed on the target.

Additional Reading

https://www.cse.iitb.ac.in/~cs695/papers/livemigration.pdf

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Lightweight virtualization - Containers, namespaces, cgroups

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Containers - Motivation

- Containers and isolation features have existed for decades. Docker uses Linux name-spaces and cgroups, which have been part of Linux since 2007.
- Docker doesn't provide the container technology, but it specifically makes it simpler to use.
- Since OS already provides some isolation
  - Can we build on that to provide a lightweight isolation mechanism?
  - Containers provide a way to **virtualize** an OS so that multiple workloads can run on a single OS instance.

Containers - Motivation

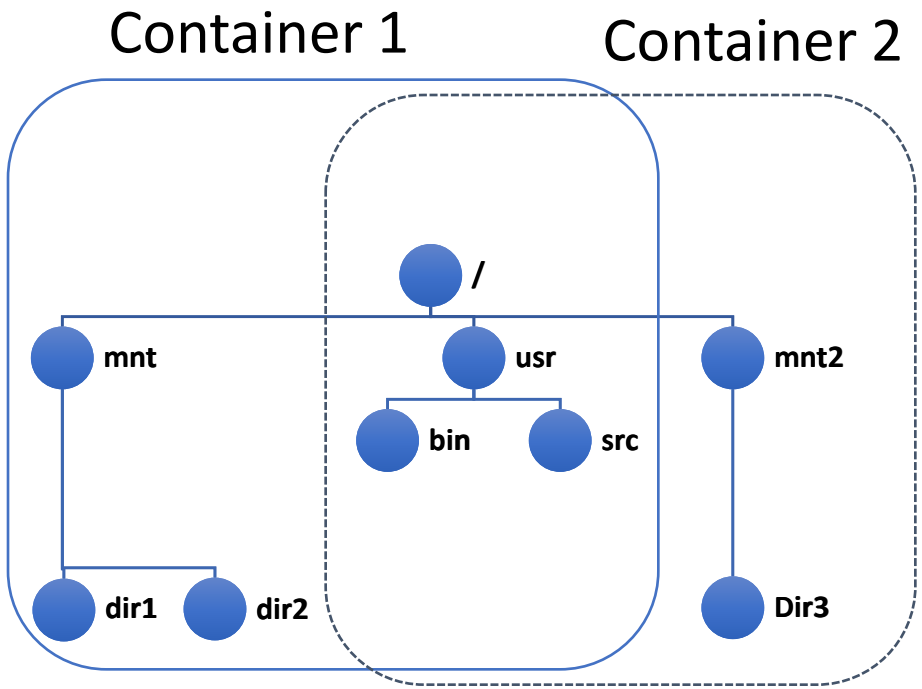
- Containers sit on top of a physical server and its host OS.
- Each container shares the host OS kernel and, usually, the binaries and libraries, too.
- Shared components are read-only.
- Containers are thus exceptionally “light”—they are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.

Containers – Access Control

- A container is
  - A sandbox (execution environment) such that
  - Processes in the container
    - Cannot access non-shared objects of other containers
    - Access only subset of objects (e.g., files) on the physical machine
- Namespaces
  - Restrict objects processes in a container can see e.g.,
    - To restrict process (and container) to a subset of files
    - Use file namespaces

Containers – Access Control

- Processes
  - In container 1 can access
    - Shared files in /usr
    - Non-shared /mnt
    - Cannot access /mnt2
  - In container 2 can access
    - Shared files in /usr
    - Non-shared /mnt2
    - Cannot access /mnt
- These are two different namespaces

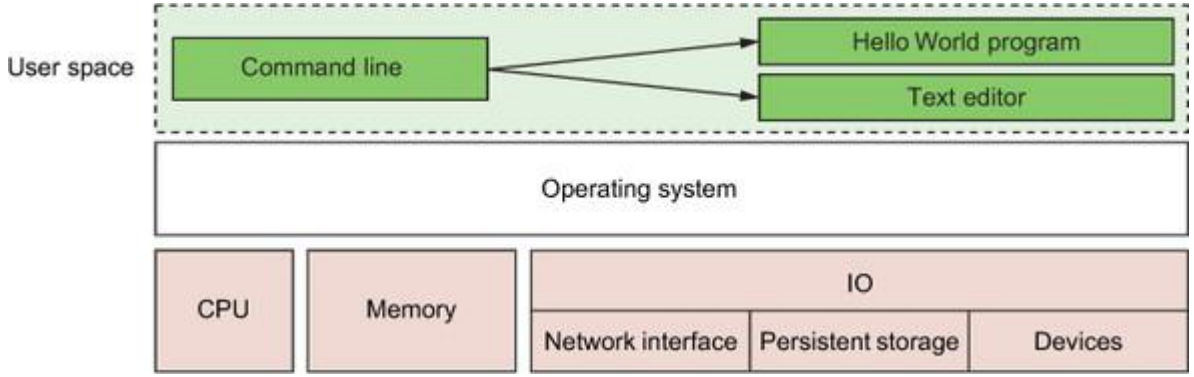


Containers – Definitions

- Linux Containers (LXC for LinuX Containers) are lightweight virtual machines (VMs) which are realized using features provided by a modern Linux kernel – VMs without the hypervisor
- Containerization of:
  - (Linux) Operating Systems
  - Single or multiple applications
- LXC as a technology not to be confused with LXC (tools) which are a user space toolset for creating & managing Linux Containers

CLOUD COMPUTING

Containers (Cont.)



A basic computer stack running two programs that were started from the command line

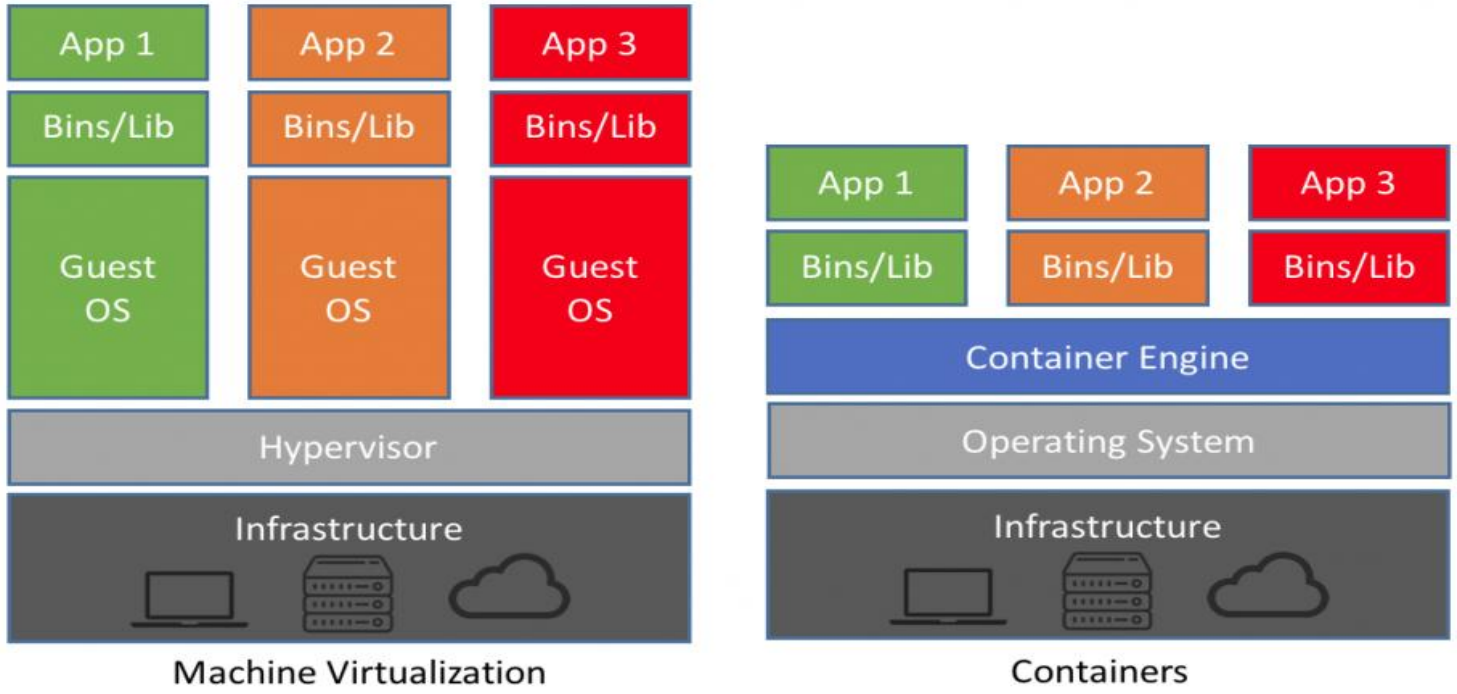
- Notice that the command-line interface, or CLI, runs in what is called user space memory, just like other programs that run on top of the operating system.
- Ideally, programs running in user space can't modify kernel space memory.
- Broadly speaking, the operating system is the interface between all user programs and the hardware that the computer is running on.

Containers vs VMs

- Containers have higher performance
  - Provisioning
    - Container creation is similar to process creation and it has speed, agility and portability.
    - VMs have to boot OS
  - Overhead
    - VMs have I/O performance overhead
- VMs are more flexible
  - Hardware is virtualized to run multiple OS instances.
  - Can’t have a Ubuntu container on a CentOS machine

CLOUD COMPUTING

Containers vs VMs



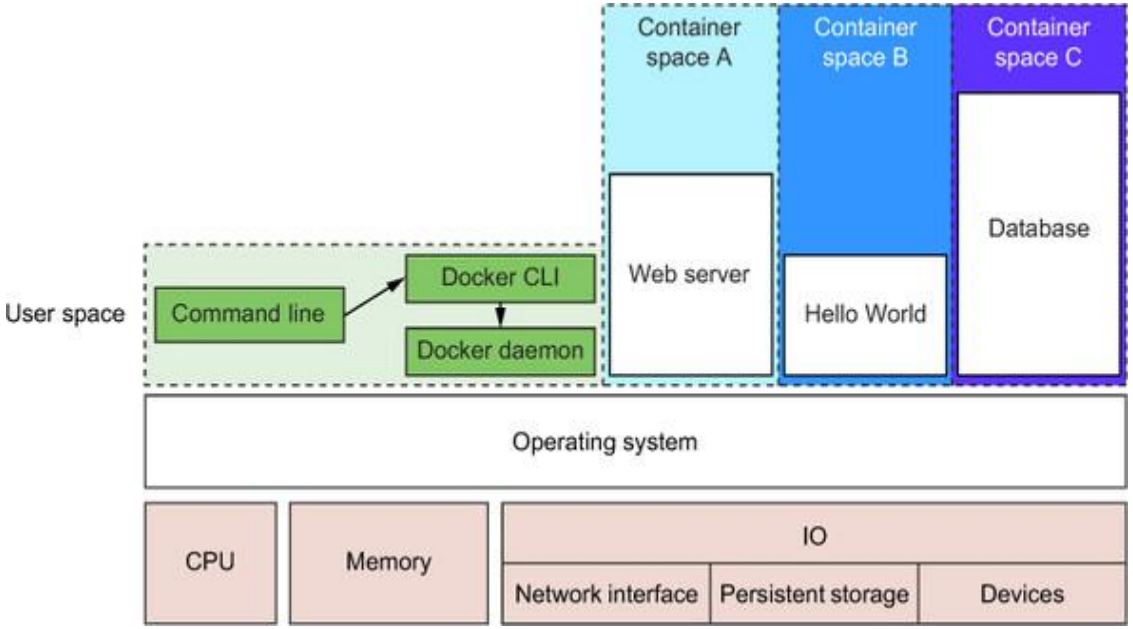
Each VM includes a separate OS image, which adds overhead in memory and storage footprint.

Containers reduce management overhead as they share a common OS, only a single OS needs to be maintained for bug fixes, patches, and so on.

CLOUD COMPUTING

Docker running three containers on a basic Linux computer system

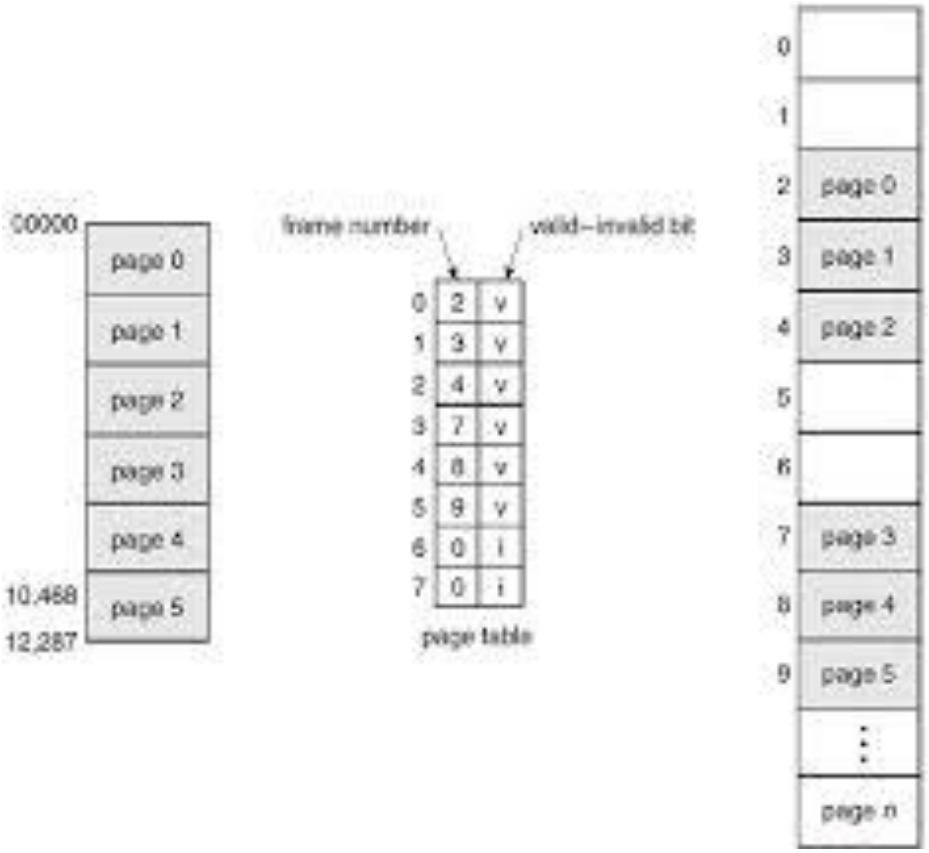
- Running Docker means running two programs in user space.
  - The first is the Docker engine that should always be running.
  - The second is the Docker CLI. This is the Docker program that users interact with to start, stop, or install software
- Each container is running as a child process of the Docker engine, wrapped with a container, and the delegate process is running in its own memory subspace of the user space.
  - Programs running inside a container can access only their own memory and resources as scoped by the container.



Docker is a set of PaaS products that use OS-level virtualization to deliver software in packages called containers.

Namespaces – What's in a Name in CS?

- If you can't name an object, you can't access it.
- Web site – if name is hidden, can't access
- Paging
  - Process can access only pages in its name space
  - Process cannot access physical page 1 (i.e frame number 1 in the diagram)



Namespaces – What's in a Name in CS? (Cont.)

- All the resources that a process sees can be considered a *namespace*
  - The files seen by a process is the *file namespace*
  - The network connections are part of *network* namespace
- The idea for namespaces came from *Plan 9 from Bell Labs*
  - Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H. and Winterbottom, P., 1995. Plan 9 from bell labs. *Computing systems*, *8*(2), pp.221-254.
  - Plan 9 is a distributed operating system, designed to make a network of heterogeneous and geographically separated computers function as a single system. In a typical Plan 9 installation, users work at terminals running GUIs, and they access CPU servers which handle computation-intensive processes. Permanent data storage is provided by additional network hosts acting as file servers and archival storage

Plan 9

- Plan 9 is based on three basic principles:
  - all objects are either files or file systems
  - communication is over a network
  - private namespaces allow their owners to access local and remote processes transparently.
- Resources are shared using standard, open protocols and consistent reusable interfaces.
- This type of distributed system is conducive to project collaboration without regard to place.
- The geographic location of the machines and the communication network are invisible to its users. It is well suited to grid computing.

CLOUD COMPUTING

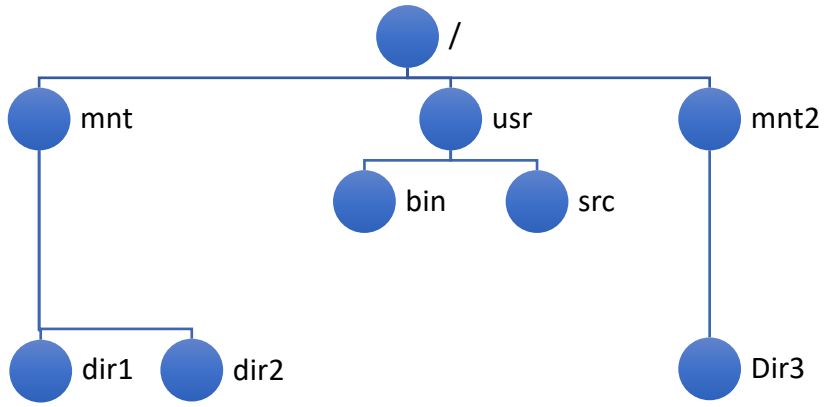
Namespaces (Cont.)

- Docker builds containers at runtime using 10 major system features.
- Docker commands to illustrate how these features can be modified to suit the needs of the contained software and to fit the environment where the container will run.
- The specific features are as follows:
  - PID namespace—Process identifiers and capabilities
  - UTS namespace—Host and domain name
  - MNT namespace—Filesystem access and structure
  - IPC namespace—Process communication over shared memory
  - NET namespace—Network access and structure
  - USR namespace—User names and identifiers
  - chroot syscall—Controls the location of the filesystem root
  - cgroups—Resource protection
  - CAP drop—Operating system feature restrictions
  - Security modules—Mandatory access controls

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

Linux Filesystems before Namespaces

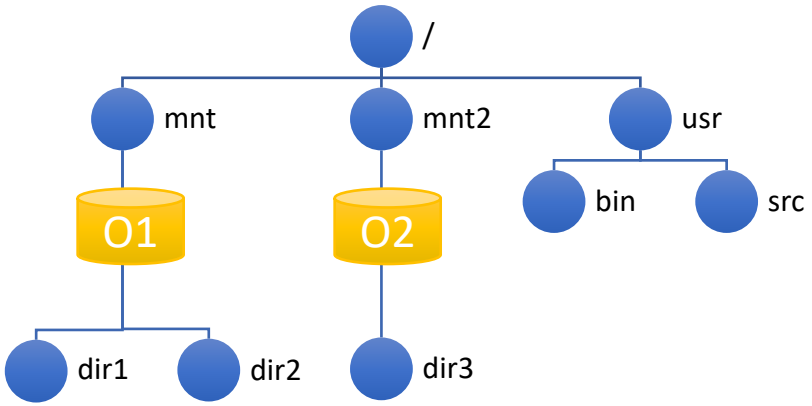
- / is the *root file system*
  - Contains *vmunix* – the OS
  - *usr, bin, src* are all subdirectories



CLOUD COMPUTING

Linux Filesystems after Namespaces

- File namespace: *mount namespace*
- *O1, O2* are volumes containing different versions of an application (e.g., Oracle)
  - Mounted at */mnt* and */mnt2*
- This namespace is visible to all processes
- Access to files and directories controlled by access rights
- Which namespace is process in?
  - Look at */proc/pid/ns*
  - *ls –l* for */proc/pidx/mnt* may show as below (4026531840 is the namespace id)



- lrwxrwxrwx. 1 mtk mtk 0 Jan 8 04:12 mnt -> mnt:[4026531840]

CLOUD COMPUTING

Linux namespaces - UTS

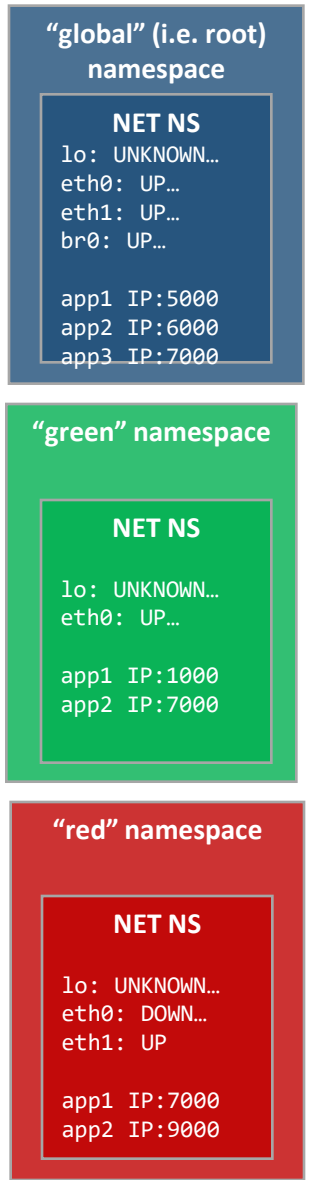
- Per namespace
  - Hostname
  - NIS domain name
- Reported by commands such as hostname
- Processes in namespace can change UTS values – only reflected in the child namespace
- Allows containers to have their own FQDN

UTS namespace is about isolating hostnames.
The **UTS namespace** is used to isolate two specific elements of the system that relate to the uname system call.
The **UTS**(UNIX Time Sharing) **namespace** is named after the data structure used to store information returned by the uname system call.

CLOUD COMPUTING

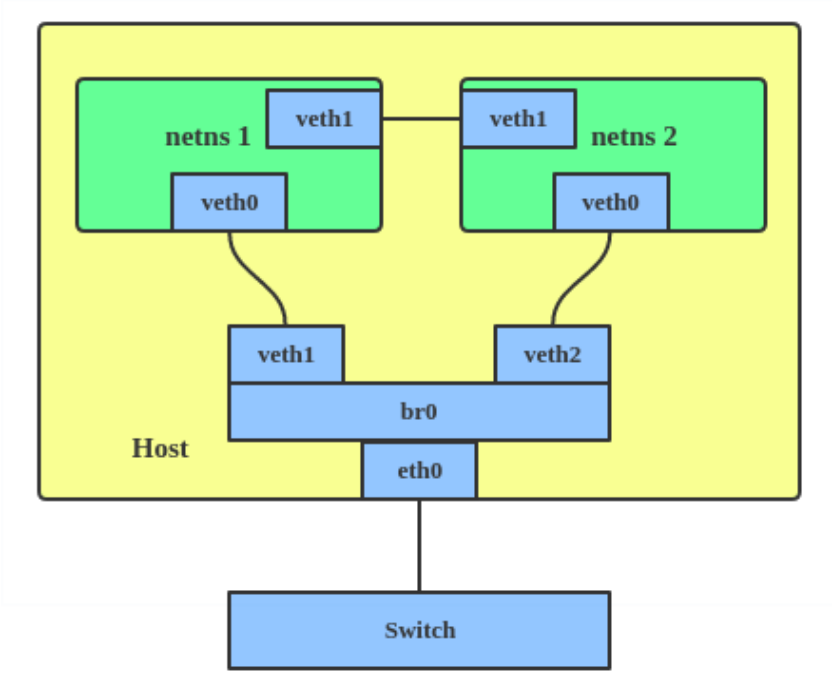
Linux namespaces - NET

- Per namespace network objects
  - Network devices (ethernets)
  - Bridges
  - Routing tables
  - IP addresses
  - ports
- Various commands support network namespace such as ip
- Connectivity to other namespaces (Diagram in next slide)
  - veths – create veth pair, move one inside the namespace and configure
  - Acts as a pipe between the 2 namespaces
- LXCs can have their own IPs, routes, bridges, etc.



Linux namespaces - NET

- The VETH (virtual Ethernet) device is a local Ethernet tunnel.
- Devices are created in pairs
- Packets transmitted on one device in the pair are immediately received on the other device.
- When either device is down, the link state of the pair is down.

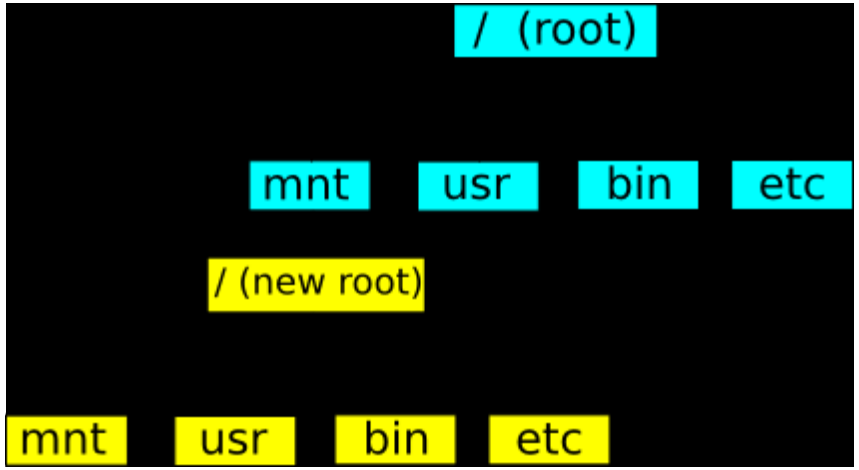


Ref: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/#:~:text=by%20Sabrina%20Dubroca.-,VETH,of%20the%20pair%20is%20down.

CLOUD COMPUTING

Changing file namespace - chroot

- The OS has a file system started at root
  - /bin, and so on, contain programs and libraries
- I want to run a program which uses a different version of /bin and so on
- Can be done using chroot
  - Mount new /bin on some place (say /mnt)
  - Issue chroot /mnt *command*
  - *command* will then see the root as /mnt, not the real root
- Using this technique, can give each process on system its own file system



Linux chroot

- Changes apparent root directory for process and children
  - Search paths
  - Relative directories
- Using chroot can be escaped given proper capabilities, thus pivot_root is often used instead
  - chroot; points the processes file system root to new directory
  - pivot_root; detaches the new root and attaches it to process root directory
  - Pivot_root info at https://man7.org/linux/man-pages/man8/pivot_root.8.html

Linux chroot (Cont.)

- Often used when building system images
  - Chroot to temp directory
  - Download and install packages in chroot
  - Compress chroot as a system root FS
- LXC realization
  - Bind mount container root FS (image)
  - Launch (unshare or clone) LXC init process in a new MNT namespace
  - pivot_root to the bind mount (root FS)

Cgroups

- Work started in 2006 by google engineers
- Merged into upstream 2.6.24 kernel due to wider spread LXC usage
- Docker uses Linux name-spaces and cgroups, which have been part of Linux since 2007.
- Control groups or Cgroups - new kernel feature - allow us to allocate resources — such as CPU time, system memory, network bandwidth, or combinations of these resources — among user-defined groups of tasks (processes) running on a system.
- We can monitor the cgroups we configure, deny cgroups access to certain resources, and even reconfigure our cgroups dynamically on a running system.
- By using cgroups, system administrators gain fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources.
- Hardware resources can be appropriately divided up among tasks and users, increasing overall efficiency.

Linux cgroups - Funtionality

- Access
  - which devices can be used per cgroup
- Resource limiting
  - memory, CPU, device accessibility, block I/O, etc.
- Prioritization
  - who gets more of the CPU, memory, etc.
- Accounting
  - resource usage per cgroup
- Control
  - freezing & check pointing
- Injection
  - packet tagging

Linux cgroups - usage

- cgroups are hierarchically structured
- Tasks are assigned to cgroups
- Each cgroup has a resource limitation
- There is a hierarchy for each resource

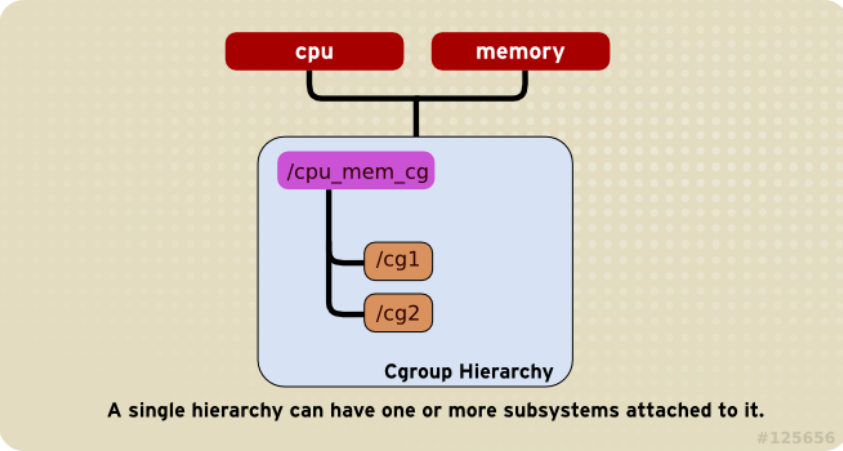
What resources can we limit?

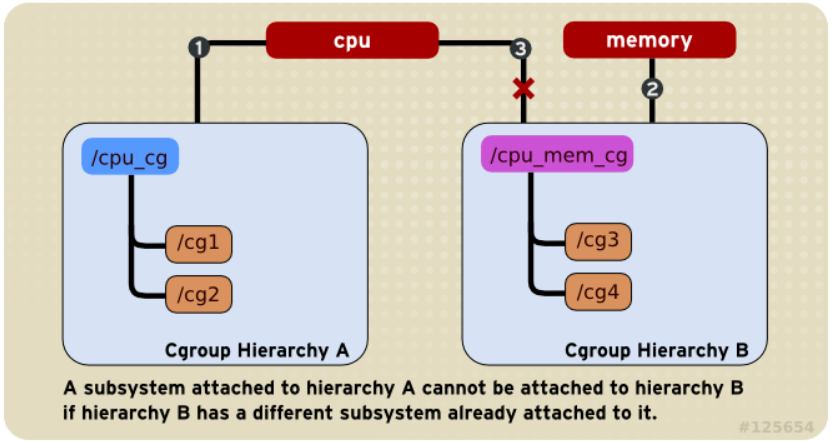
- Memory
- CPU
- BlkIO
- Devices
- Network

https://lwn.net/Articles/524935/

CLOUD COMPUTING

cgroup Examples

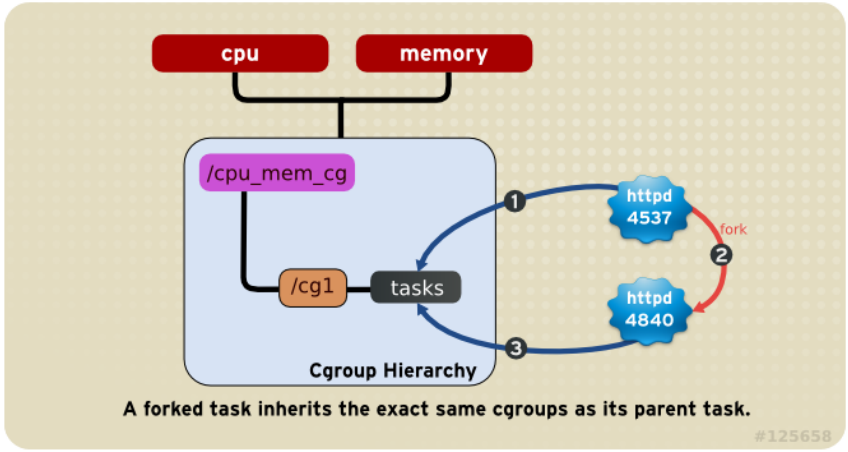




Each cgroup has a resource limitation associated with it

What resources can we limit?

- When a process creates a child process, the child process stays in the same cgroup
  - Good for servers such as NFS
  - Typical operation
    - Receive request
    - Fork child to process request
    - Child terminates when request complete
    - All children will have resource limitation of parent => the resource limitation of parent will apply to processing requests



Summary

- Use namespaces for controlling resource access
  - Docker has developed their own namespace technology called **namespaces** to provide the isolated workspace called the container.
  - When you run a container, **Docker** creates a set of **namespaces** for that container.
  - These **namespaces** provide a layer of isolation.
- Use cgroups for resource sharing

Additional References

Containers vs VMs
https://www.redhat.com/en/topics/containers/containers-vs-vms

Docker container
https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container

Understanding Linux containers
https://www.redhat.com/en/topics/containers
https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html

Additional Reading

- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/sec-Relationships_Between_Subsystems_Hierarchies_Control_Groups_and_Tasks.html
- https://en.wikipedia.org/wiki/Cgroups
- https://www.youtube.com/watch?v=sK5i-N34im8
  - Cgroups, namespaces and beyond: What are containers made from – Jerome Pettazzoni, Docker

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Deployment of cloud native applications through Docker – Unionfs

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Docker

- Tool that creates “application containers”
- Each container contains
  - Full static application stack
- Allows quick deployment of applications
- Uses LXC and other technologies
- *Docker is a tool that makes adopting* software packaging, distribution, and utilization best practices cheap and sensible defaults.
- It does so by providing a complete vision for process containers and simple tooling for building and working with them.
- Containers come up more quickly and consume fewer resources than virtual machines.

More on Docker

- Docker isn't a programming language, and it isn't a framework for building software.
- Docker is a tool that helps solve common problems such as installing, removing, upgrading, distributing, trusting, and running software.
- *Docker is an* open source project for building, shipping, and running programs.
- It's open source Linux software, which means that anyone can contribute to it and that it has benefited from a variety of perspectives.
- Uses operating system technology called **containers**
- Docker Inc. is the primary sponsor (https://docker.com/company/)

Docker (Cont.)

- Docker can be used with network applications such as web servers, databases, and mail servers, and with terminal applications including text editors, compilers, network analysis tools, and scripts.
  - In some cases, it's even used to run GUI applications such as web browsers and productivity software.
- Docker runs Linux software on most systems.
  - Docker for Mac and Docker for Windows integrate with common virtual machine (VM) technology to create portability with Windows and macOS.
    - On macOS and Windows, Docker uses a single, small virtual machine to run all the containers.
  - Docker can run native Windows applications on modern Windows server machines.

LXC vs Docker

LXC

- Virtualization technology
  - Implements namespaces for isolation
- LXC containers not portable across machines with different configs

Docker

- Management technology
  - Layer on top of LXC; tries to make LXC easier to use
- Docker containers portable across different storage, network, log configs
- Tools to simplify building container images
- Versioning of images
- Image re-use: build new images by adding more software

http://stackoverflow.com/questions/17989306/what-does-docker-add-to-lxc-tools-the-userspace-lxc-tools

CLOUD COMPUTING

VM vs Docker

Virtual Machine	Docker Container
Hardware-level process isolation	OS level process isolation
Each VM has a separate OS	Each container can share OS
Boots in minutes	Boots in seconds
VMs are of few GBs	Containers are lightweight (KBs/MBs)
Ready-made VMs are difficult to find	Pre-built docker containers are easily available
VMs can move to new host easily	Containers are destroyed and re-created rather than moving
Creating VM takes a relatively longer time	Containers can be created in seconds

More resource usage

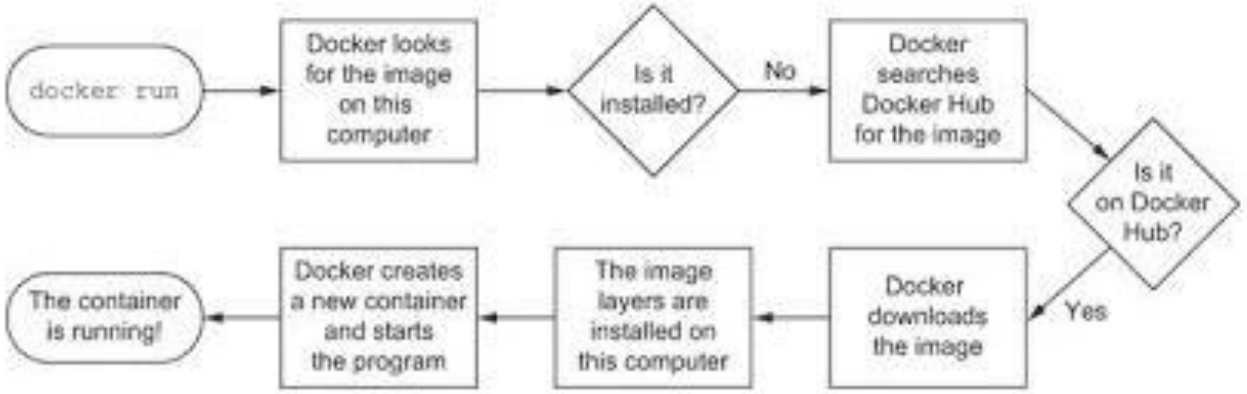
Less resource usage

CLOUD COMPUTING

Running a program with Docker

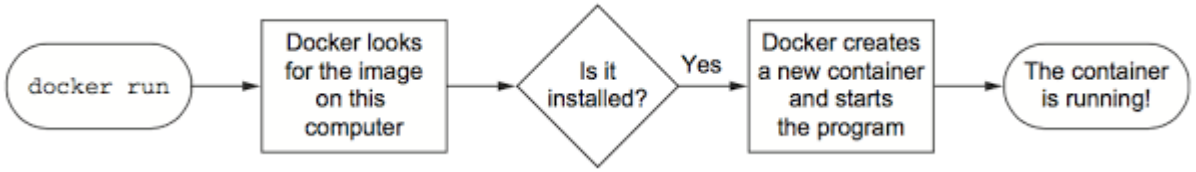
What happens after running **docker run**

The image itself is a collection of files and metadata which includes the specific program to execute and other relevant configuration details



Running docker run a second time.

The image is already installed, so Docker can start the new container right away.



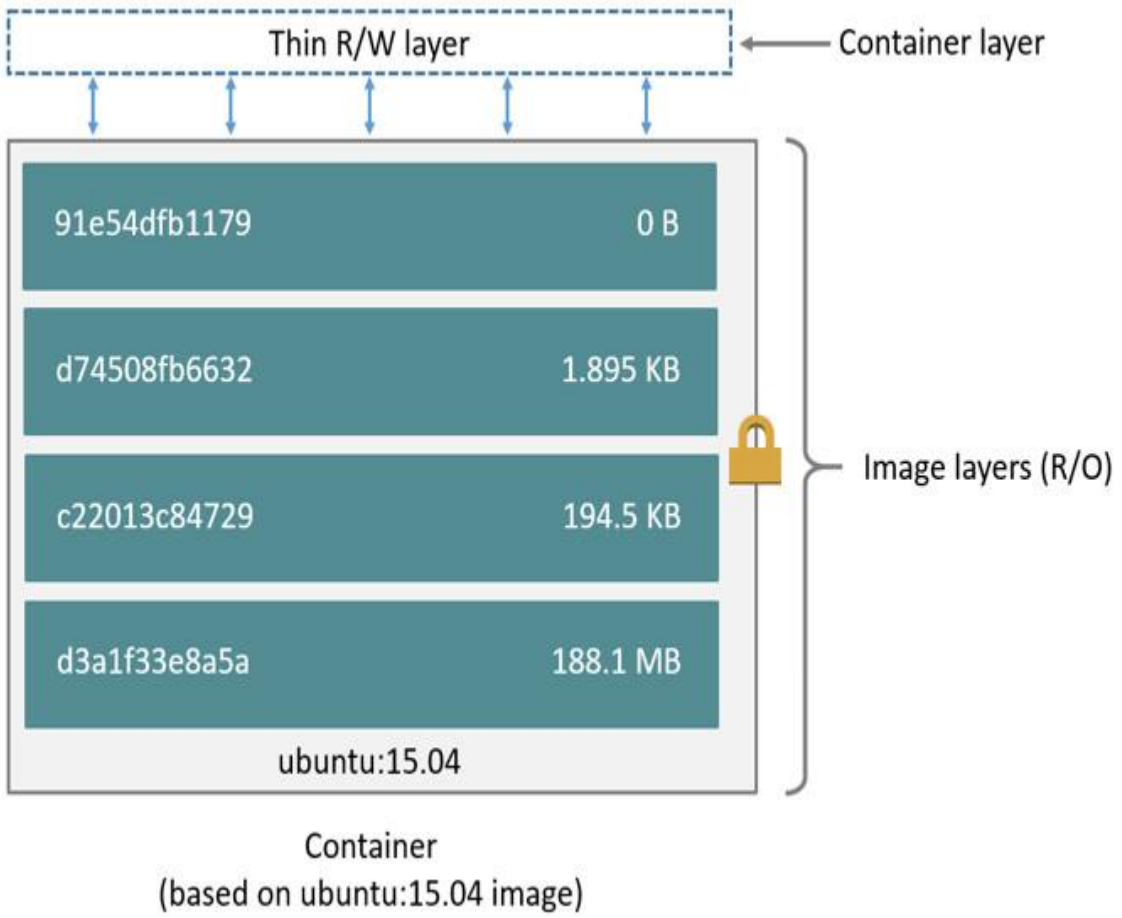
Docker Images

- A *layer is set of files and file metadata that is packaged and distributed as an atomic* unit.
- Internally, Docker treats each layer like an image, and layers are often called *intermediate images.*
- You can even promote a layer to an image by tagging it.
- Most layers build upon a parent layer by applying filesystem changes to the parent.

CLOUD COMPUTING

Docker Images (Cont.)

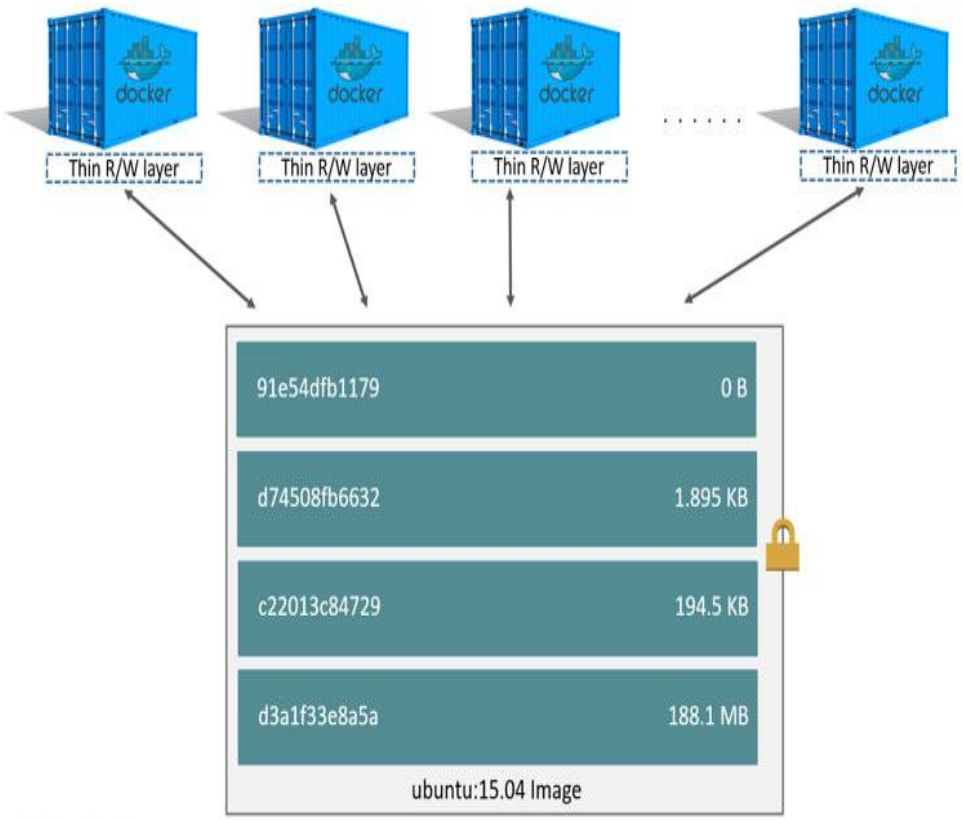
- Specification stored in Dockerfile
  - Should be for only one application
  - Only the definition of the image
- Image is built from Dockerfile
- Specifies the R/O file systems in which various programs are installed
  - For example, web server + libraries



https://docs.docker.com/storage/storagedriver/

Docker Images (Cont.)

- Image + temporary R/W file system
  - Used as temporary storage
  - Deleted when container is destroyed
- Multiple containers can use the same image and their own temporary storage



https://docs.docker.com/storage/storagedriver/

Persistent Storage in Docker

- Container storage
  - Temporary
  - Deleted when container is destroyed
- Persistent storage
  - Mount host volumes using *mount* command

Dockerfile

- A *Dockerfile is a script that describes steps for Docker to take to build a new image.*
- These files are distributed along with software that the author wants to be put into an image. In this case, you're not technically installing an image. Instead, you're following instructions to build an image.
- Distributing a Dockerfile is similar to distributing image files using your own distribution mechanisms. A common pattern is to distribute a Dockerfile with software from common version-control systems like Git or Mercurial.
    - If you have Git installed, you can try this by running an example from a public repository:

```
git clone https://github.com/dockerinaction/ch3_dockerfile.git
docker build -t dia_ch3/dockerfile:latest ch3_dockerfile
```

In this example, you copy the project from a public source repository onto your computer and then build and install a Docker image by using the Dockerfile included with that project. The value provided to the -t option of docker build is the repository where you want to install the image.

Dockerfile (Cont.)

❑ Building images from Dockerfiles is a light way to move projects around that fits into existing workflows. This approach has two disadvantages.

- First, depending on the specifics of the project, the build process might take some time.
- Second, dependencies may drift between the time when the Dockerfile was authored and when an image is built on a user's computer.
- These issues make distributing build files less than an ideal experience for a user. But it remains popular in spite of these drawbacks.

❑ When you're finished with this example, make sure to clean up your workspace:

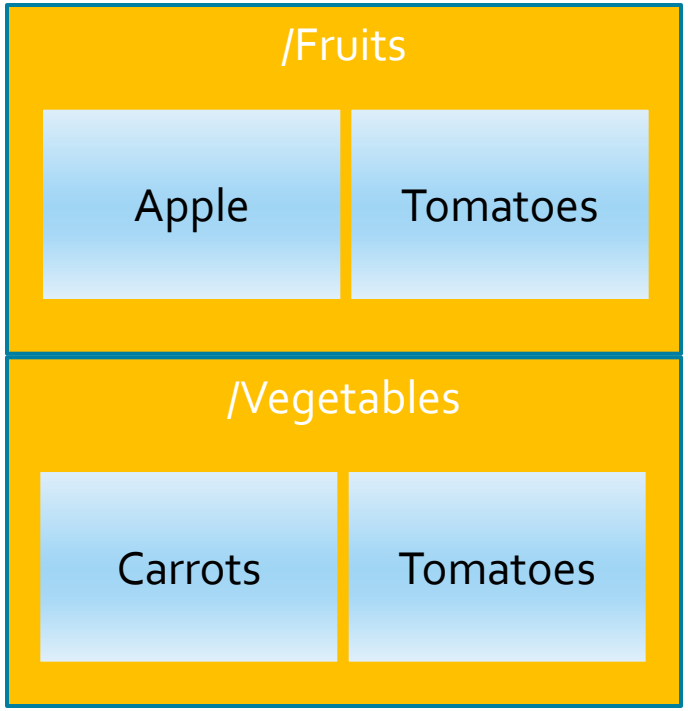
```
docker rmi dia_ch3/dockerfile
rm -rf ch3_dockerfile
```

Container Filesystem

- Programs running inside containers know nothing about image layers.
- From inside a container, the filesystem operates as though it's not running in a container or operating on an image.
- From the perspective of the container, it has exclusive copies of the files provided by the image. This is made possible with something called a *union filesystem (UFS).*
- Docker uses a variety of union filesystems and will select the best fit for your system.
- A union filesystem is part of a critical set of tools that combine to create effective filesystem isolation.
- The other tools are MNT namespaces and the chroot system call.

Unionfs features - Layering

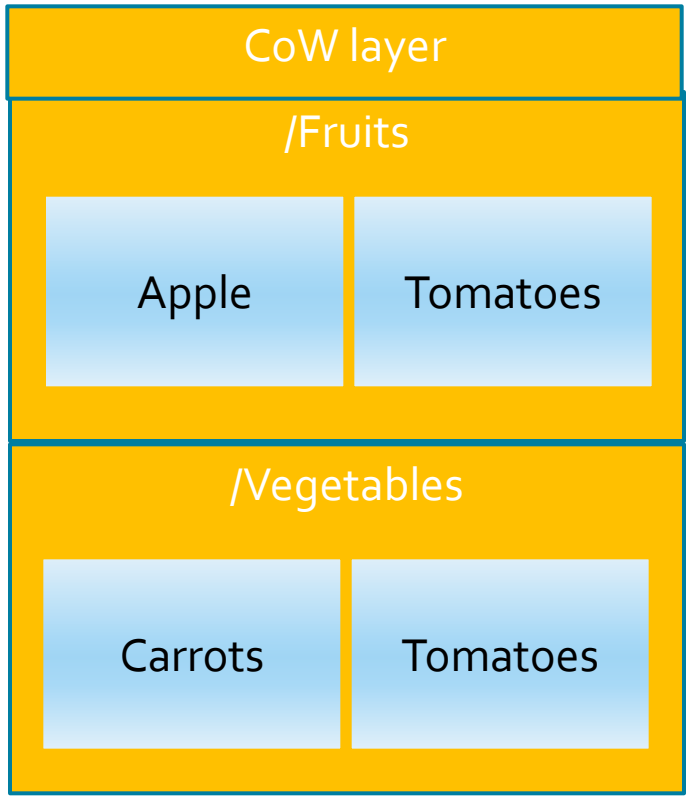
- Unionfs permits layering of file systems
  - */Fruits* contains files *Apple, Tomato*
  - */Vegetables* contains *Carrots, Tomato*
  - *mount –t unionfs –o dirs=/Fruits:/Vegetables none /mnt/healthy*
    - */mnt/healthy* has 3 files – *Apple, Tomato, Carrots*
    - *Tomato* comes from */Fruits* (1st in *dirs* option)
- As if */Fruits* is layered on top of */Vegetables*



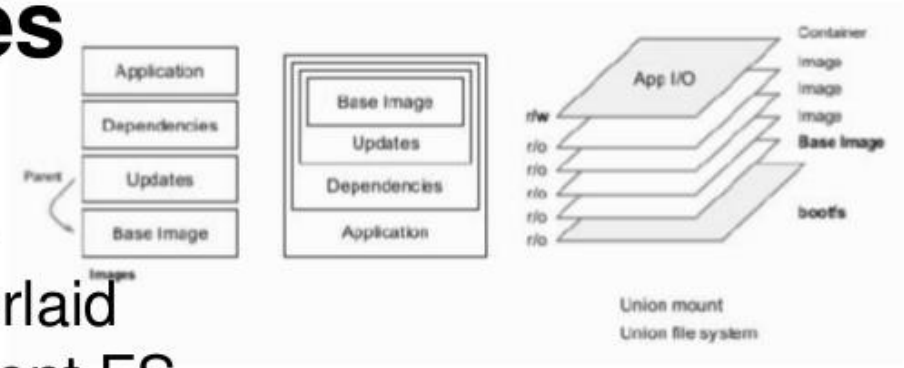
http://www.linuxjournal.com/article/7714

Unionfs features - Layering

- *-o cow* option on *mount* command enables *copy on write*
  - If change is made to a file
  - Original file is not modified
  - New file is created in a hidden location
- If */Fruits* is mounted *ro*, then changes will be recorded in a temporary layer



Incremental Images



- UnionFS
  - files from separate FS (branches) can be overlaid
  - forming a single coherent FS
  - branches may be read-only or read-write
- Docker Layers
  - each layer is mounted on top of prior layers
  - first layer = base image (scratch, busybox, ubuntu,..)
  - a read-only layer = an image
  - the top read-write layer = container

https://www.slideshare.net/cm13/introduction-to-docker-48784005

Weaknesses of Union Filesystem

- Different filesystems have different rules about file attributes, sizes, names, and characters.
- Union filesystems are in a position where they often need to translate between the rules of different filesystems. In the best cases, they're able to provide acceptable translations. In the worst cases, features are omitted.
- Union filesystems use a pattern called *copy-on-write, and that makes implementing* memory-mapped files (the mmap system call) difficult.
- Most issues that arise with writing to the union filesystem can be addressed without changing the storage provider. These can be solved with volumes
- The union filesystem is not appropriate for working with long-lived data or sharing data between containers, or a container and the host.

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

DevOps

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

Definition

- DevOps is a set of practices that combines software development and IT operations (development and operations)
- It aims to shorten the system development life cycle (SDLC) and provide continuous delivery with high software quality.
- DevOps is complementary with Agile software development; several DevOps aspects came from Agile methodology

DevOps Definition (Cont.)

- **DevOps** is a set of practices that automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably.
- The concept of **DevOps** is founded on building a culture of collaboration between teams that historically functioned in relative siloes.
- It follows Plan, Build, Deploy, Operate cycle.
- Docker and Kubernetes are 2 of 10 best DevOps tools now. Git is another one.

DevOps is not a fad; rather it is the way successful organizations are industrializing the delivery of quality software today and will be the new baseline tomorrow and for years to come. – Computer Business Review

The Dawn of DevOps

- DevOps is a set of practices that combines software development and IT operations. It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. DevOps is complementary with Agile software development; several DevOps aspects came from Agile methodology
- Before DevOps, developing and operating software were essentially two separate jobs, performed by two different groups of people.
- Developers wrote software, and they passed it on to operations staff, who ran and maintained the software in production (that is to say, serving real users, instead of merely running under test conditions).

The Dawn of DevOps (Cont.)

- Software development was a very specialist job, and so was computer operation, and there was very little overlap between the two.
- Indeed, the two departments had quite different goals and incentives, which often conflicted with each other.
- Developers tend to be focused on shipping new features quickly, while operations teams care about making services stable and reliable over the long term.

Motivational Problem

- You have built and tested the containerized application
- You learnt how to deploy your application using Docker/K8S
- As customers require more features,
  - What do you do?
- Let's look at some engineering issues

https://www.iltam.org/check_download_nopass.php?forcedownload=1&file=files/DevOpsconfpresentationSandrine1.pptx&no_encrypt=true&dlpassword=20426

SDLC – Software Development Life Cycle

- Customer generates requirements
- Do a design
  - Different teams work on different components
  - Integrated together
  - Then tested for
    - Functionality
    - Similarity to customer deployment
  - Then for deployment, need to talk to customers IT team.
    - Manages deployment, scaling, run time issues with the application

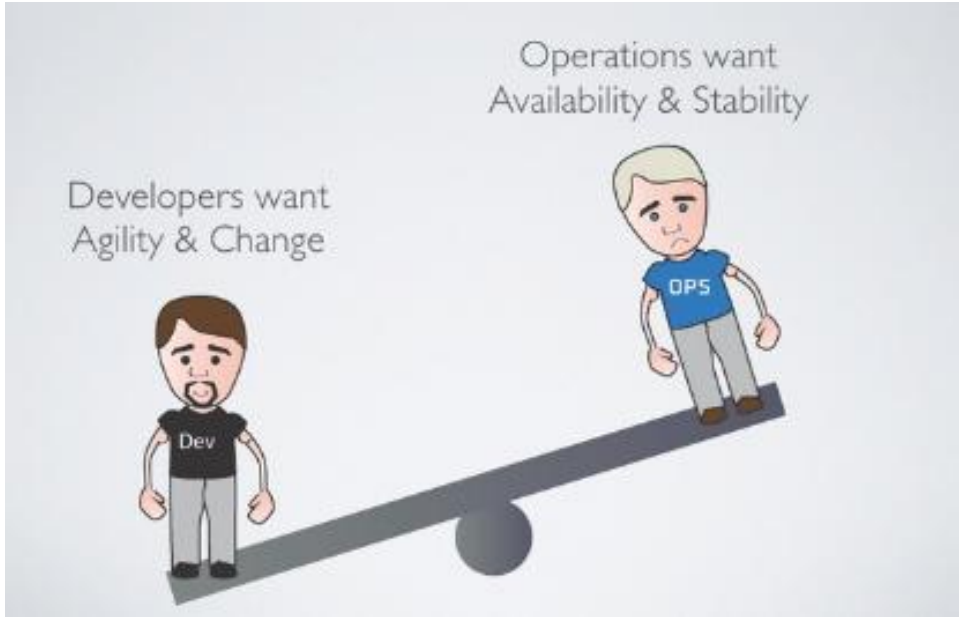
CLOUD COMPUTING

Dev & Ops Typical Converation

Put the current release live, NOW!
It works on my machine
We need this Yesterday
You are using the wrong version

- What are the dependencies?
- No machines available..
- Which DB?
- High Availability?
- Scalability?

Make the balance between Dev & Ops

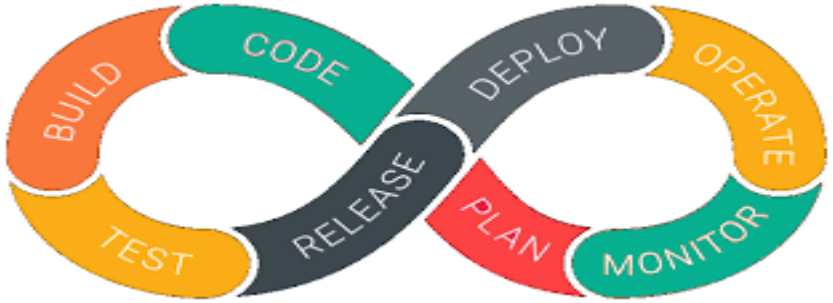


In pre-cloud era, requires a lot of manual intervention like procuring machines, etc.

With cloud – automation is possible

SDLC in Cloud

- Do a design
  - Different teams work on different components
  - Integrated together
  - Then tested for
    - Functionality
    - Similarity to customer deployment
  - Then for deployment, need to talk to cloud IT team.
    - Handles deployment, scaling, monitoring
- So, what are the challenges?



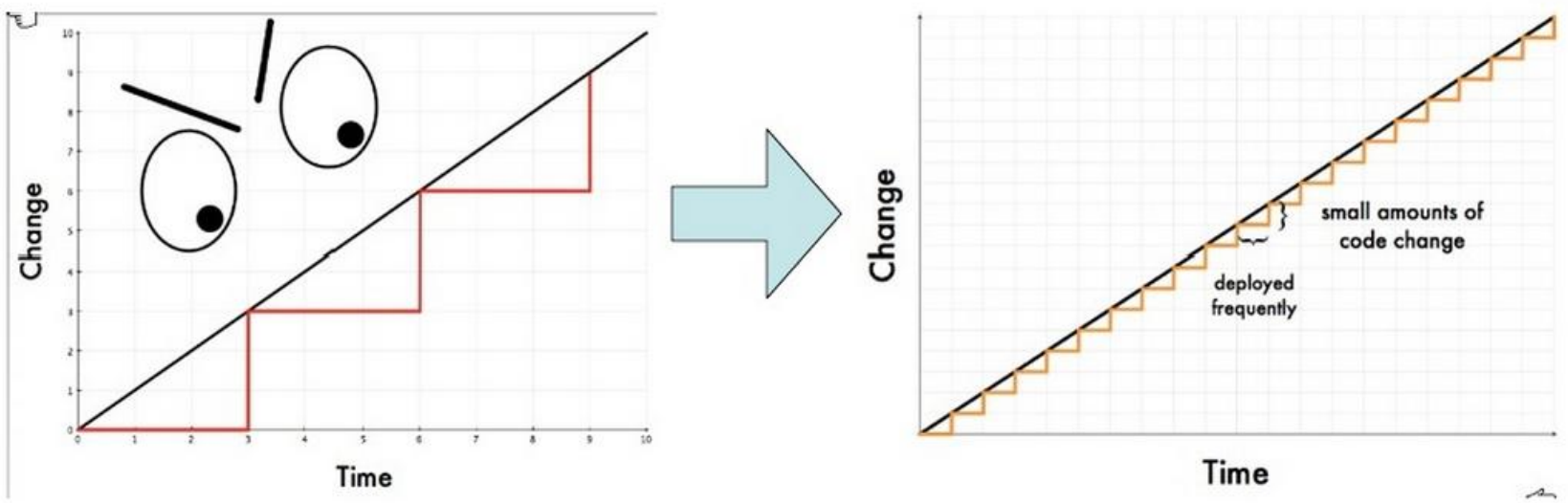
DEV **OPS**

Advantages

- Faster Time to Market – checkins more often, so can build and test more often.
- Quality of Code/Release improved because of frequent testing
- Integration Often
- Deploy Often – changes reach the customers often
- Automated
- Every one is happy (Hopefully!!)

Keep it Simple

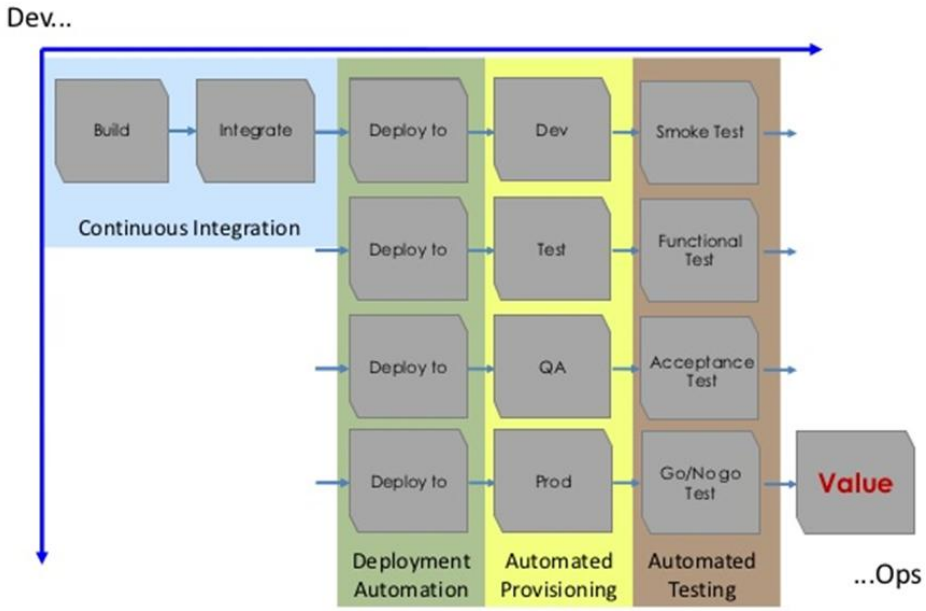
- Reduce risk of Release
- Make small changes and Test it every cycle
- Small changes can make the difference for Ops & Dev



John Allspaw: “Ops Metametrics” http://slidesha.re/dsSZIr

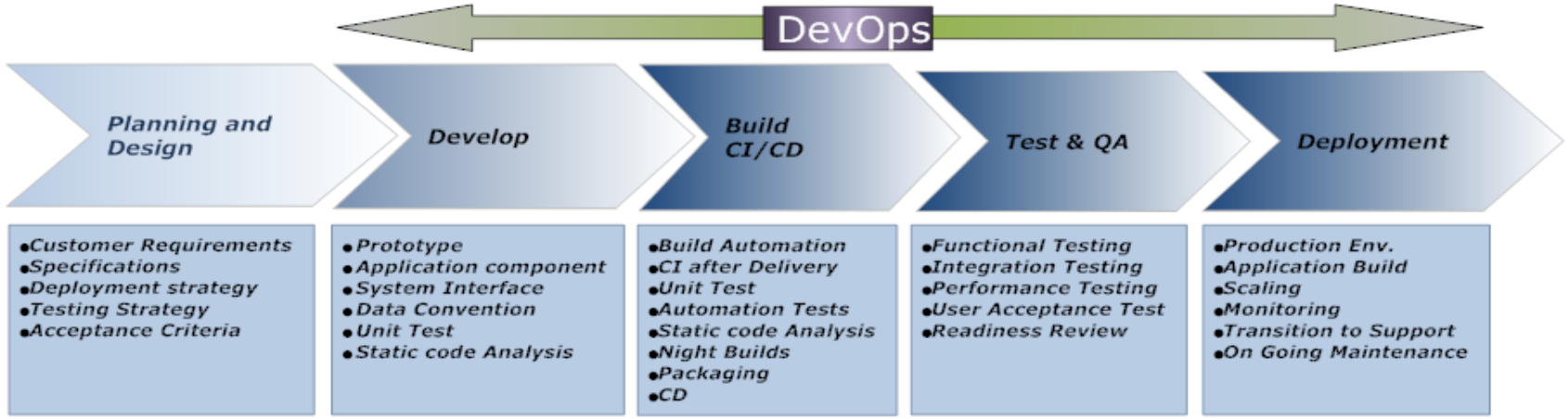
Automate all the things

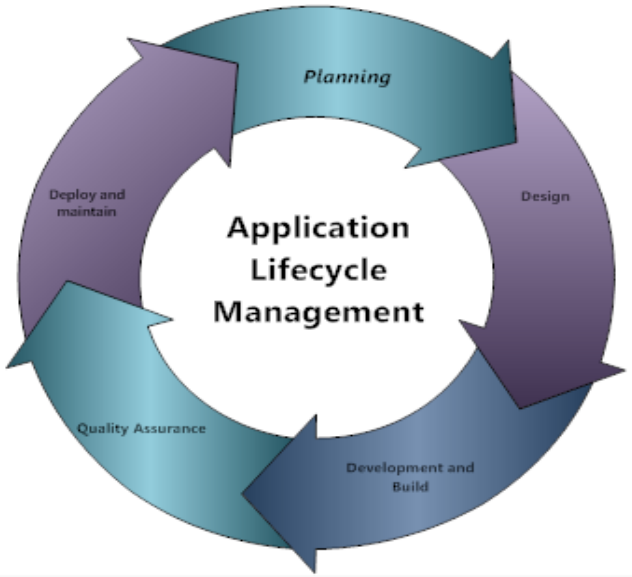
- Programming Model to enable scale
- Manage source code
- Reproducible Build
  - Build on a Prod-Like environment
  - No more “Works on my machine”
- Test
  - Testing reduces risks
  - Make you more confident
- Deploy
  - Deploy to Dev
  - Deploy to QA
  - Deploy to Pre-Prod
  - Prod



CLOUD COMPUTING

DevOps CI/CD and ALM





CI/CD - Continuous Integration and Continuous Delivery or Deployment.

CI/CD

❑ **Continuous deployment (CD)** is the automatic deployment of successful builds to production.

❑Like the test suite, deployment should also be managed centrally and automated.

❑Developers should be able to deploy new versions by either pushing a button, or merging a merge request, or pushing a Git release tag.

❑CD is often associated with **continuous integration (CI):** the automatic integration and testing of developers' changes against the mainline branch.

❑ If you're making changes on a branch that would break the build when merged to the mainline, continuous integration will let you know that right away, rather than waiting until you finish your branch and do the final merge.

❑The combination of continuous integration and deployment is often referred to as **CI/CD**.

CI/CD (Cont.)

- The machinery of continuous deployment is often referred to as a ***pipeline****: a series of* automated actions that take code from the developer's workstation to production, via a sequence of test and acceptance stages.
- pipelines automate the process for building, testing, and publishing artifacts so they can be deployed to a runtime environment.
- People often automate build pipelines with continuous integration systems such as Jenkins, Travis CI, or Drone.

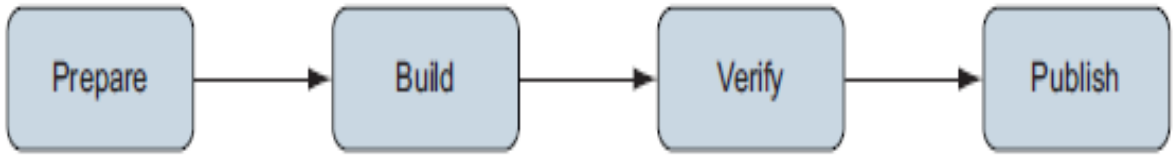
CI/CD (Cont.)

Typical pipeline for containerized applications might look like the following:

1. A developer pushes his/her code changes to Git.
2. The build system automatically builds the current version of the code and runs tests.
3. If all tests pass, the container image will be published into the central container registry.
4. The newly built container is deployed automatically to a staging environment.
5. The staging environment undergoes some automated acceptance tests.
6. The verified container image is deployed to production.

Generic Artifact Build Pipeline

- High-level process for building software or other artifacts in a pipeline.
- The goal of a build pipeline is to apply a consistent set of rigorous practices in creating deployable artifacts from source definitions.
- Continuous integration systems such as Jenkins, Travis CI, or Drone often used to automate build pipelines



Jenkins

- **Jenkins** is a very widely adopted CD tool. There is also a newer dedicated side project for running Jenkins in Kubernetes cluster, **JenkinsX**.
- Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
- Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

Jenkins (Cont.)

- Jenkins is a highly extensible product whose functionality can be extended through the installation of plugins.
- There are a vast array of plugins available to Jenkins.
- Plugins are the primary means of enhancing the functionality of a Jenkins environment to suit organization- or user-specific needs.
- There are over a thousand different plugins which can be installed on a Jenkins controller and to integrate various build tools, cloud providers, analysis tools, and much more.

References: (useful for Cloud lab experiment)

https://www.jenkins.io/doc/

https://www.jenkins.io/doc/book/pipeline/

https://jenkins-x.io/

Additional References

https://www.guru99.com/jenkins-tutorial.html
https://www.guru99.com/devops-tutorial.html

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Orchestration and Kubernetes

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

Slides Credits for all PPTs of this course

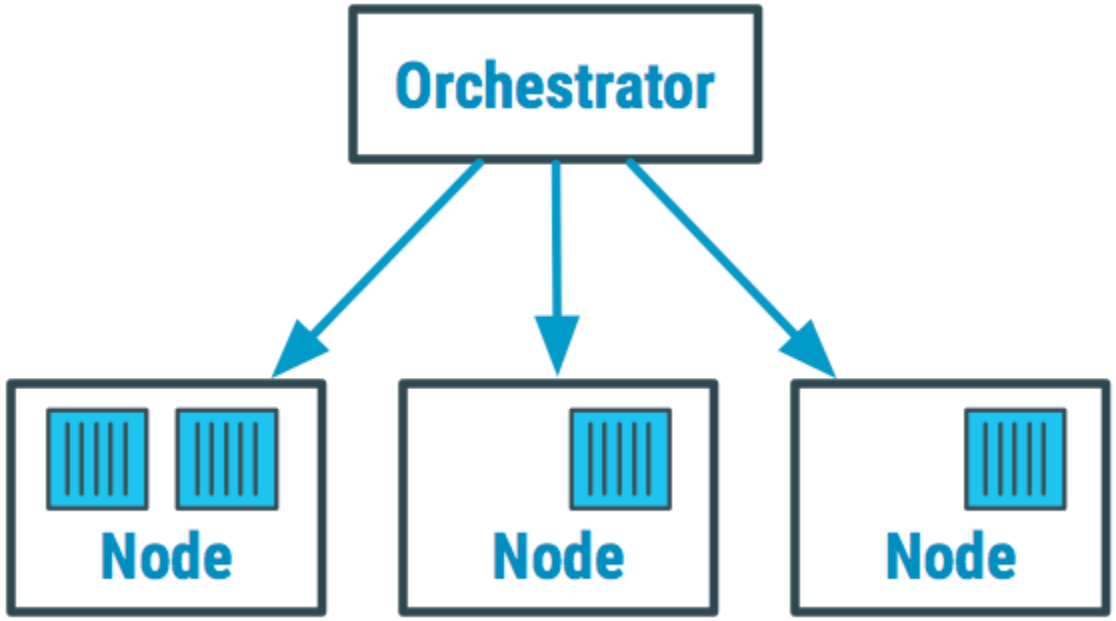
- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

CLOUD COMPUTING

Container orchestration

- Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments.
- The orchestrator manages containers across nodes



Container orchestration is the process of deploying containers on a compute cluster consisting of multiple nodes.
Orchestration tools extend lifecycle management capabilities to complex, multi-container workloads deployed on a cluster of machines.

https://devopedia.org/container-orchestration

Container orchestration

- Container orchestrator: a piece of software designed to join together many different machines into a cluster: a kind of unified compute substrate, which appears to the user as a single very powerful computer on which containers can run.

- Operations teams, too, find their workload greatly simplified by containers.
- Instead of having to maintain a sprawling estate of machines of various kinds, architectures, and operating systems, all they have to do is run a *container orchestrator*

> *Container orchestration is a process that automates the deployment, management, scaling, networking, and availability of container-based applications.*

Container orchestration

- The terms orchestration and scheduling are often used loosely as synonyms.
  - Strictly speaking, though, orchestration in this context means coordinating and sequencing different activities in service of a common goal (like the musicians in an orchestra).
  - Scheduling means managing the resources available and assigning workloads where they can most efficiently be run.

Container orchestration (Cont.)

- A third important activity is cluster management: joining multiple physical or virtual servers into a unified, reliable, fault-tolerant, apparently seamless group.
- The term container orchestrator usually refers to a single service that takes care of scheduling, orchestration, and cluster management.
- Kubernetes is the most pervasive container orchestration platform to address these challenges.
  - Kubernetes is an open-source container management (orchestration) tool.
  - It's container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
  - It lets you manage complex application deployments quickly in a predictable and reliable manner.

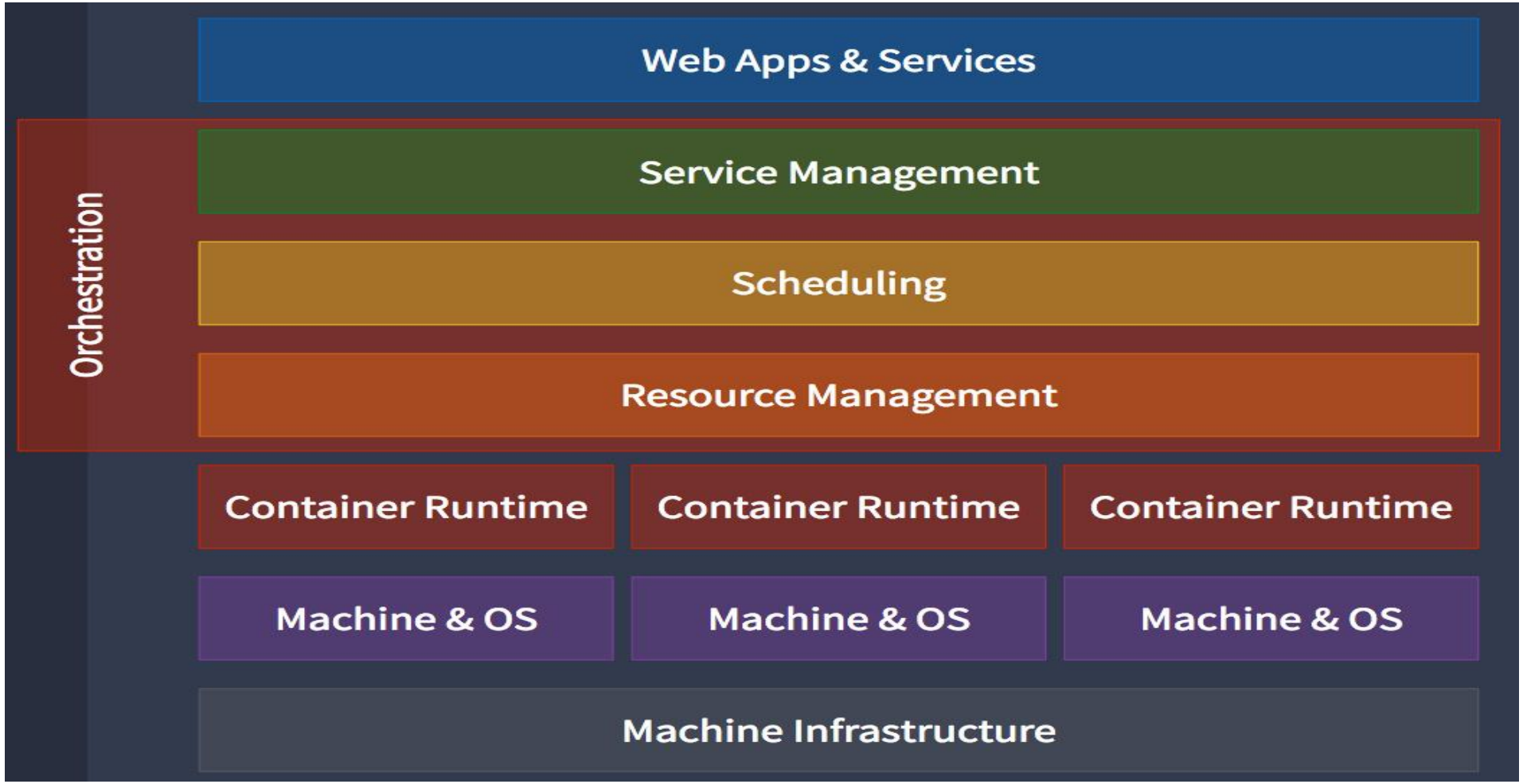
Container orchestration (Cont.)

- Container orchestration is used to control and automate many tasks:
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
  - Scaling up or removing containers to spread application load evenly across host infrastructure
  - Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
  - Allocation of resources between containers
  - External exposure of services running in a container with the outside world
  - Load balancing of service discovery between containers
  - Health monitoring of containers and hosts
  - Configuration of an application in relation to the containers running it

CLOUD COMPUTING

Where does Container orchestration fit within the system stack?

Container Orchestration mediates between the apps/services and the container runtimes



Container orchestration – Functional aspects

- **Service Management**: Labels, groups, namespaces, dependencies, load balancing, readiness checks.
- **Scheduling**: Allocation, replication, (container) resurrection, rescheduling, rolling deployment, upgrades, downgrades.
- **Resource Management**: Memory, CPU, GPU, volumes, ports, IPs.

CLOUD COMPUTING

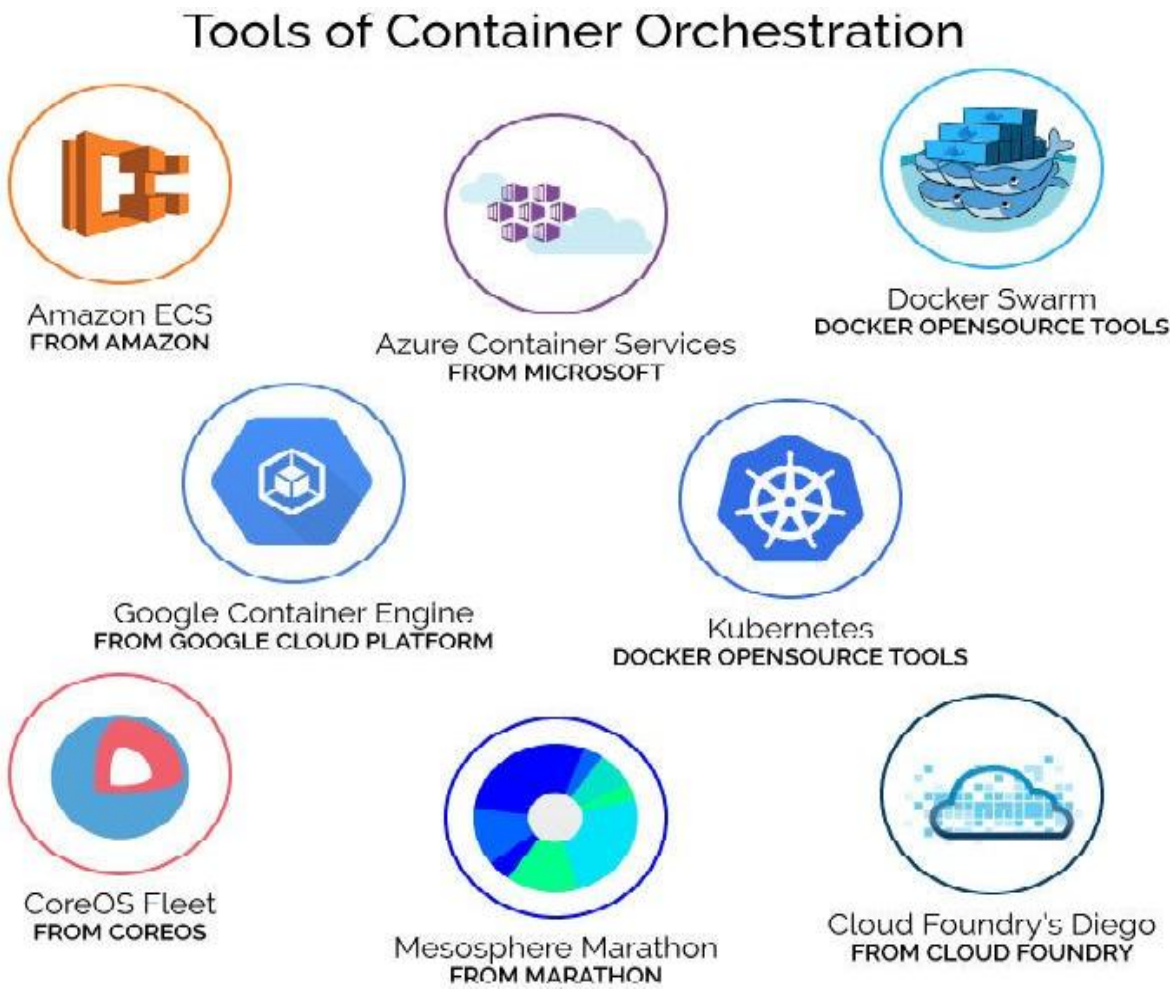
Container orchestration – Tools

Docker Swarm: Provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single, virtual Docker engine.

Google Container Engine: Google Container Engine, built on Kubernetes, can run Docker containers on the Google Cloud.

Kubernetes: An orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

Amazon ECS: The ECS supports Docker containers and lets you run applications on a managed cluster of Amazon EC2 instances.



Kubernetes Basics

- Containerisation has brought a lot of flexibility for developers in terms of managing the deployment of the applications.
- More granular application consists of more components and hence requires some sort of management for those.
- Need a solution to
  - ✓ take care of scheduling the deployment of a certain number of containers to a specific node
  - ✓ manage networking between the containers
  - ✓ follow the resource allocation
  - ✓ move containers around as they grow and much more.

CLOUD COMPUTING

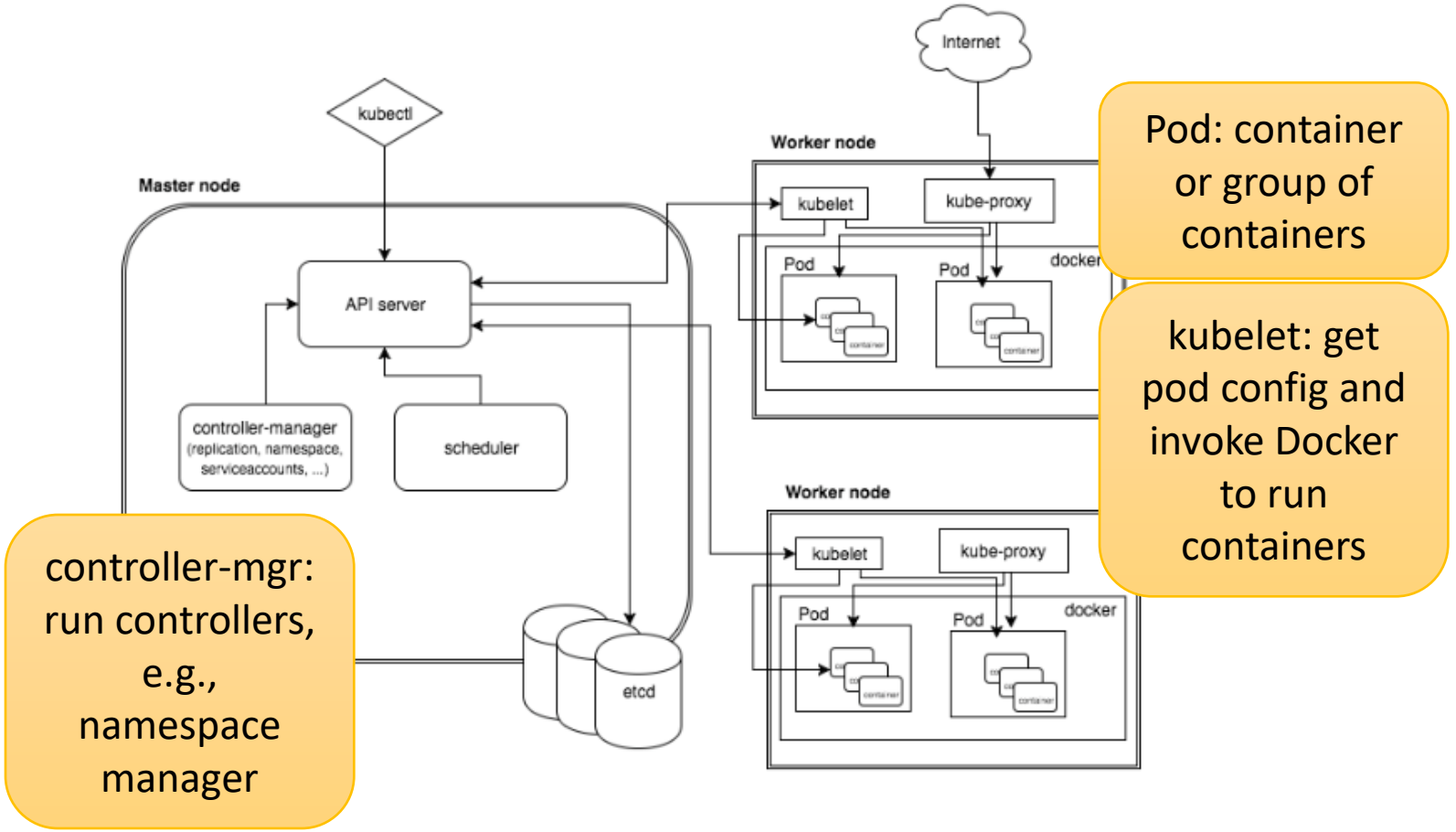
Kubernetes Basics (Cont.)

- Replication of components
- Auto-scaling
- Load balancing
- Rolling updates
- Logging across components
- Monitoring and health checking
- Service discovery
- Authentication

> *Kubernetes does the things that the very best system administrator would do: automation, failover, centralized logging, monitoring.*
> *It takes what we've learned in the DevOps community and makes it the default, out of the box.*
> ***- Kelsey Hightower,*** *Staff Developer Advocate, Google Cloud Platform at Google .*

CLOUD COMPUTING

Kubernetes Architecture



Pod: container or group of containers

kubelet: get pod config and invoke Docker to run containers

controller-mgr: run controllers, e.g., namespace manager

https://x-team.com/blog/introduction-kubernetes-architecture

Kubernetes Architecture

Kubernetes targets the management of elastic applications that consist of multiple microservices communicating with each other. Often those microservices are tightly coupled forming a group of containers that would typically, in a non-containerized setup run together on one server. This group, the smallest unit that can be scheduled to be deployed through K8s is called a *pod*.

The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node is the one taking care of orchestrating the worker nodes, where the actual services are running.

CLOUD COMPUTING

Kubernetes - Pod

- Kubernetes targets the management of elastic applications that consist of multiple microservices communicating with each other.
- Often those microservices are tightly coupled forming a group of containers that would typically, in a non-containerized setup run together on one server.
- This group, the smallest unit that can be scheduled to be deployed through K8s is called a ***pod***.
- This group of containers would share storage, Linux namespaces, cgroups, IP addresses.
- These are co-located, hence share resources and are always scheduled together.
- Pods are not intended to live long. They are created, destroyed and re-created on demand, based on the state of the server and the service itself.

Kubernetes - Service

- As pods have a short lifetime, there is no guarantee about the IP address they are served on. This could make the communication of microservices hard.
  - Imagine a typical Frontend communication with Backend services.
- Hence K8s has introduced the concept of a *service*, which is an abstraction on top of a number of pods, typically requiring to run a proxy on top, for other services to communicate with it via a Virtual IP address.
- This is where we can configure load balancing for our numerous pods and expose them via a service.

CLOUD COMPUTING

Kubernetes Components – Master Node

- Master Node
- API server
- etcd storage
- controller-manager
- Scheduler

Kubernetes Components – Worker Node

Docker
Kubelet
kube-proxy
Kubectl

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu

CLOUD COMPUTING

Kubernetes Continuation and Demo

Venkatesh Prasad
Department of Computer Science

CLOUD COMPUTING

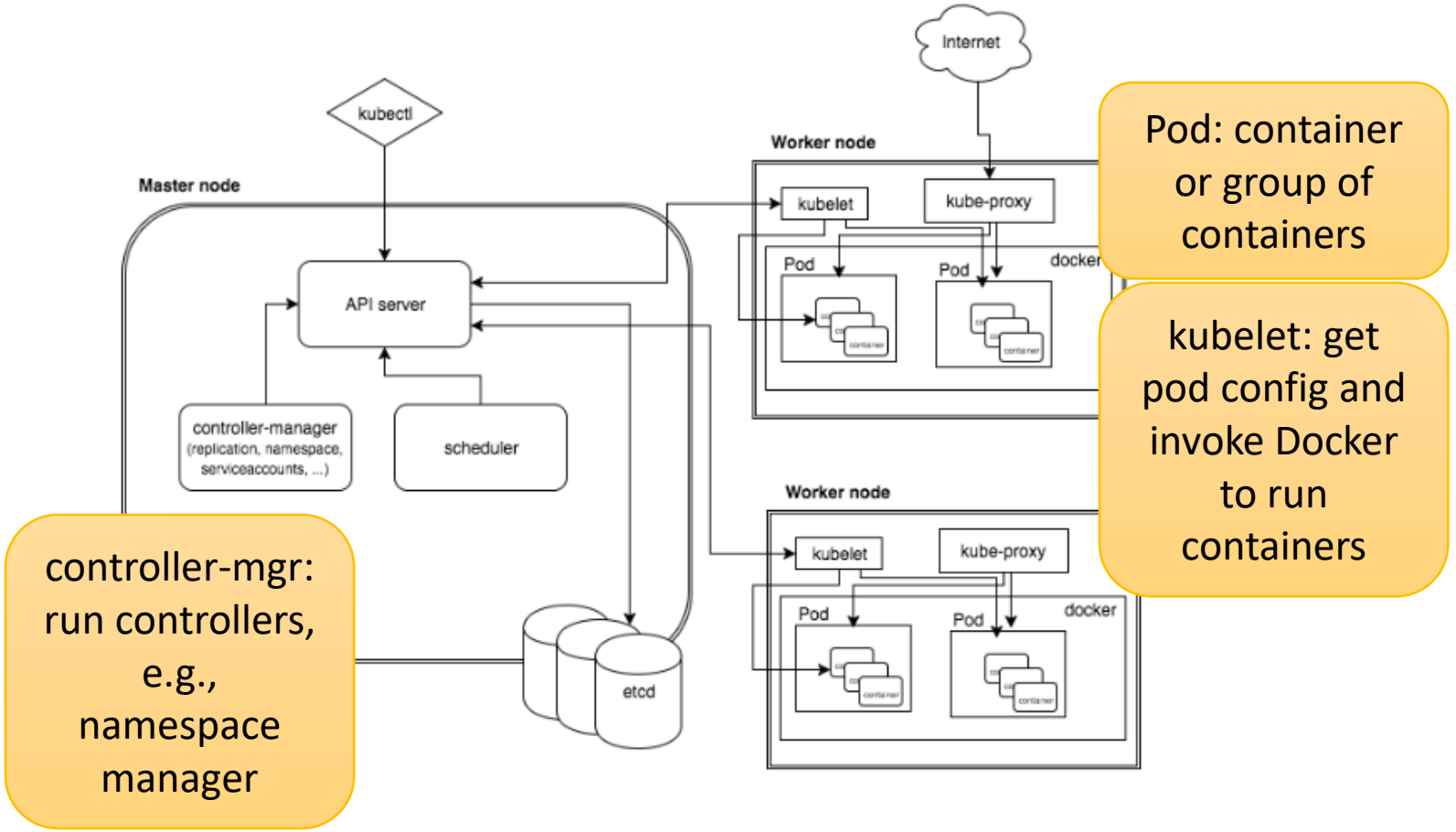
Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Prof Arkaprava Basu and Prof Sorav Bansal
2. Some slides, conceptual text and diagram from Scott Devine, Vmware
3. Text book and Reference books prescribed in the syllabus
4. Other Internet sources

CLOUD COMPUTING

Kubernetes Architecture



Pod: container or group of containers

kubelet: get pod config and invoke Docker to run containers

controller-mgr: run controllers, e.g., namespace manager

https://x-team.com/blog/introduction-kubernetes-architecture

Kubernetes Components – Master Node

Master Node

The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node is the one taking care of orchestrating the worker nodes, where the actual services are running.
Let's dive into each of the components of the master node.

API server

The API server is the entry points for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere (etcd component of the master node).

Kubernetes Components – Master Node (Cont.)

etcd storage

- **etcd** is a simple, distributed, consistent key-value store.
- It's mainly used for shared configuration and service discovery.
- It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.
  - CRUD - Create, Read, Update, and Delete, which are four primitive database operations.
- An example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc.

Kubernetes Components - Master Node (Cont.)

controller-manager

- Optionally we can run different kinds of controllers inside the master node.
- A controller uses api server to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one.
- An example of such a controller is the Replication controller, which takes care of the number of pods in the system. The replication factor is configured by the user, and it's the controller's responsibility to recreate a failed pod or remove an extra-scheduled one.
- Other examples of controllers are endpoints controller, namespace controller, and service accounts controller.

Kubernetes Components - Master Node (Cont.)

Scheduler

- Deploys configured pods and services onto the nodes
- The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

Kubernetes Components – Worker Node

Worker node

The pods are run here, so the worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled.

Docker

- Docker runs on each of the worker nodes, and runs the configured pods.
- It takes care of downloading the images and starting the containers.

Kubernetes Components - Worker Node (Cont.)

Kubelet

- kubelet gets the configuration of a pod from the api server and ensures that the described containers are up and running.
- This is the worker service that's responsible for communicating with the master node.
- It also communicates with etcd, to get information about services and write the details about newly created ones.

Kubernetes Components - Worker Node (Cont.)

kube-proxy

- kube-proxy acts as a network proxy and a load balancer for a service on a single worker node.
- It takes care of the network routing for TCP and UDP packets.

Kubectl

A command line tool to communicate with the API service and send commands to the master node.

Docker and Kubernetes – A Brief Comparison

- Docker is a containerization platform, and Kubernetes is a container orchestrator for container platforms like Docker.
- Docker provided an open standard for packaging and distributing containerized applications while Kubernetes will coordinate and schedule these containers , seamlessly upgrade an application without any interruption of service, monitor the health of an application, know when something goes wrong and seamlessly restart it.

https://www.sumologic.com/blog/kubernetes-vs-docker/

Docker and Kubernetes – A Brief Comparison (Cont.)

❑Docker's own native clustering solution for Docker containers is **Docker Swarm** , which has the advantage of being tightly integrated into the ecosystem of Docker, and uses its own API.

- Like most schedulers, **Docker Swarm** provides a way to administer a large number of containers spread across clusters of servers.
- Its filtering and scheduling system enables the selection of optimal nodes in a cluster to deploy containers.

Docker and Kubernetes – A Brief Comparison (Cont.)

❑Kubernetes is a comprehensive system for automating deployment, scheduling and scaling of containerized applications, and supports many containerization tools such as Docker.

- Kubernetes is the market leader and the standardized means of orchestrating containers and deploying distributed applications.
- Kubernetes can be run on a public cloud service or on-premises, is highly modular, open source, and has a vibrant community.
- Companies of all sizes are investing into it, and many cloud providers offer Kubernetes as a service

Additional References

Docker and Kubernetes

https://www.guru99.com/docker-tutorial.html
https://www.guru99.com/kubernetes-vs-docker.html

CLOUD COMPUTING

AWS Demo

How to create Virtual Machines - EC2 in AWS
https://www.youtube.com/watch?v=N_mP4mIqK8A

AWS Tutotial for Beginners
https://www.youtube.com/watch?v=yxhYuNLkL8A

Introduction to AWS Services
https://www.youtube.com/watch?v=Z3SYDTMP3ME

How to Create a Virtual Machine in Amazon AWS
https://www.youtube.com/watch?v=YL1hViT4Buw

AWS ECS
https://www.youtube.com/watch?v=pOal0uCrEaA

Virtualization in AWS
https://www.youtube.com/watch?v=bdnVH8Hvlls
https://www.youtube.com/watch?v=HaDDzDepFpY

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu