Virtualization

Mythili Vutukuru
CSE, IIT Bombay

Virtualization and cloud computing

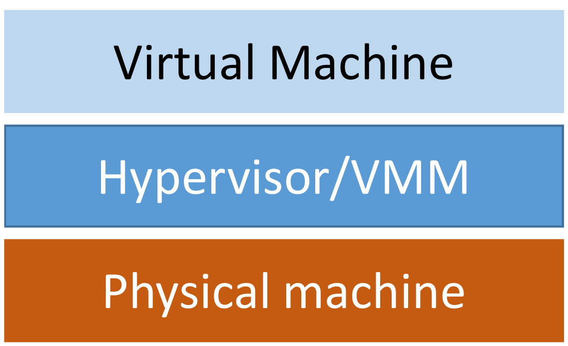
- What is the cloud?
  - Commodity servers with lots of compute and storage, connected with high speed networking, located in data centers
- What is virtualization?
  - Multiple virtual machines (VMs) can run inside a physical machine (PM)
  - VM gives user an illusion of running on a physical machine
  - Containers are like lightweight VMs
- Virtualization is a building block for cloud computing
  - Virtualization enables multiple clients share the cloud's compute resources
  - Multiple users on VMs/containers can share same cloud server
- In addition to compute, clouds also manage large amounts of data
  - Cloud storage/big data systems for efficient storage and retrieval of data

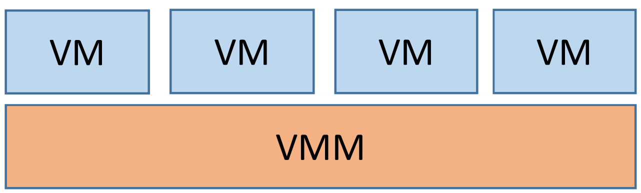
Why cloud computing?

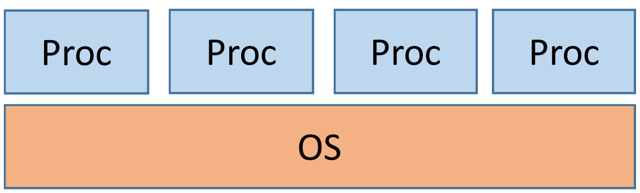
- Public cloud providers (Amazon AWS, Microsoft Azure, Google Cloud etc) setup and maintain data centers with high-end servers
  - Powerful CPUs, lots of memory, disk storage etc., available to users
  - Organizations can also run a private cloud only for their users
- Why run applications on cloud and not on “bare metal” servers?
  - Multiplexing gains: multiple VMs can share the system resources
  - Lower overhead of maintenance: hardware/software maintained by providers
  - Flexibility: VMs can move to another machine if one fails
  - Pay as per usage: no need to invest in servers if only lightly used
- Disadvantages of running applications on cloud
  - Performance: longer delay to access servers via internet
  - Higher cost if heavily used

Hypervisor (VMM)

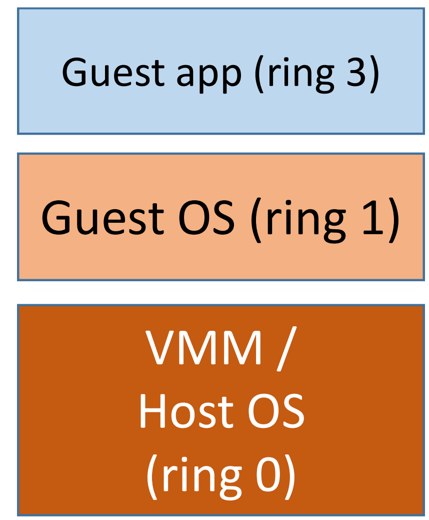
- Hypervisor or virtual machine monitor (VMM): a piece of software that allows multiple VMs to run on a physical machine (PM)
- Multiple VMs running on a PM – multiplex the underlying machine
  - Similar to how OS multiplexes processes on CPU
- Guest OS expects complete control over hardware, but VMM must multiplex multiple guest OSes on the same hardware – how?







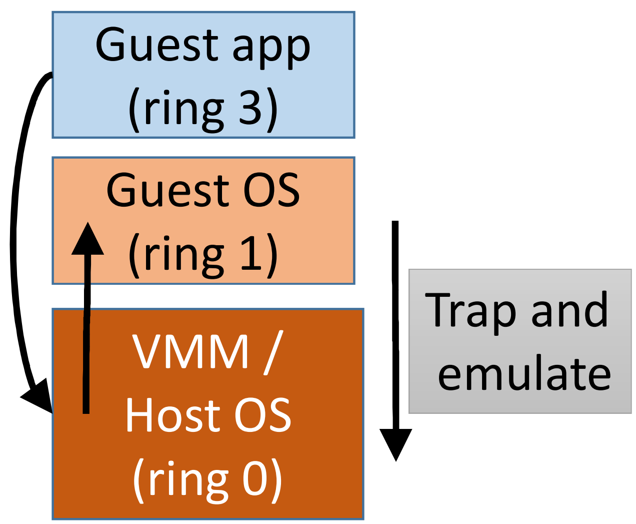
Basic idea: Trap and emulate VMM



- 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
  - Can run in ring 1?
- Trap-and-emulate VMM: guest OS runs at lower privilege level than VMM, traps to VMM for privileged operation

Trap and emulate VMM

- 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

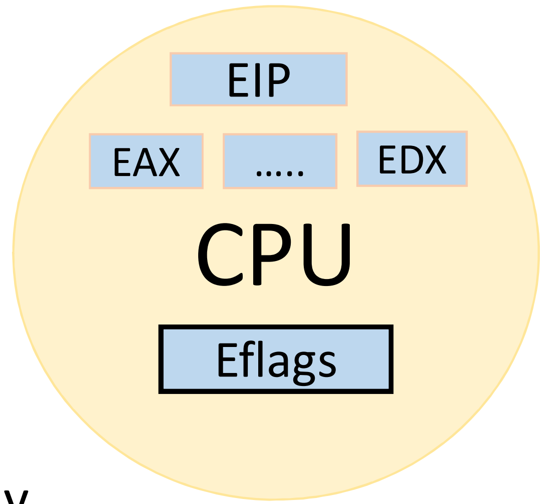


Problems with trap and emulate

- 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)

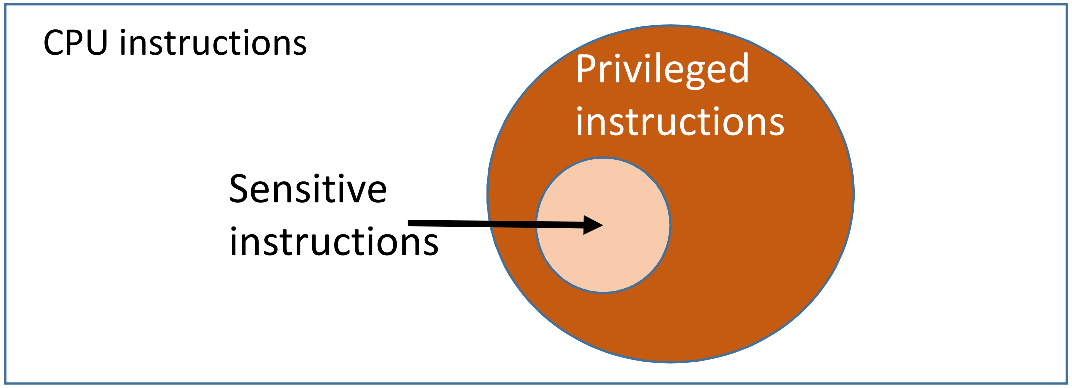
Example: Problems with trap and emulate

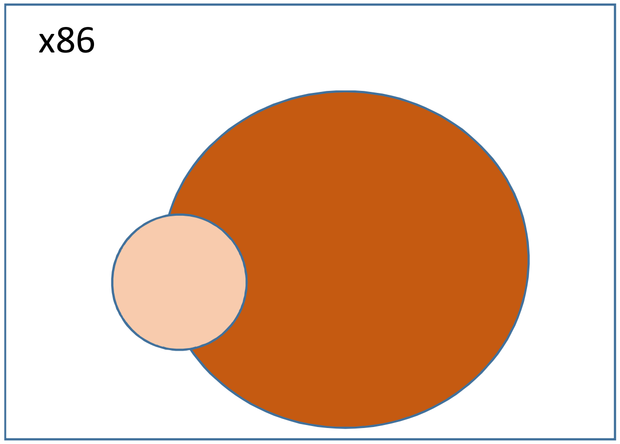
- Eflags register is a set of CPU flags
  - IF (interrupt flag) indicates if interrupts on/off
- Consider the popf instruction in x86
  - Pops values on top of stack and sets eflags
- Executed in ring 0, all flags set normally
- Executed in ring 1, only some flags set
  - IF is not set as it is privileged flag
- So, popf is a sensitive instruction, not privileged, does not trap, behaves differently when executed in different privilege levels
  - Guest OS is buggy in ring 1



Popek Goldberg theorem

- Sensitive instruction = changes hardware state
- Privileged instruction = runs only in privileged mode
  - Traps to ring 0 if executed from unprivileged rings
- In order to build a VMM efficiently via trap-and-emulate method, sensitive instructions should be a subset of privileged instructions
  - x86 does not satisfy this criteria, so trap and emulate VMM is not possible





Techniques to virtualize x86 (1)

- 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

Techniques to virtualize x86 (2)

- Hardware assisted virtualization: KVM/QEMU in Linux
  - CPU has a special VMX mode of execution
  - X86 has 4 rings on non-VMX root mode, another 4 rings in VMX mode
- VMM enters VMX mode to run guest OS in (special) ring 0
- Exit back to VMM on triggers (VMM retains control)