Lecture 4 CS 695

Design of VMMs / Hypervisors.

recap:

① Native: OS is the owner of all resources.
- interrupts, LDE, system calls.

② challenges to VMM design.
- who is the owner? ownership resolution question.

---

* Design principles of VMMs.

Popek & Goldberg 1974

(i) efficiency – min. VMM ovhds.

(ii) resource control – VMM is the owner of all resources

(iii) equivalence – appl'n. should execute identically on VMs & PMs.

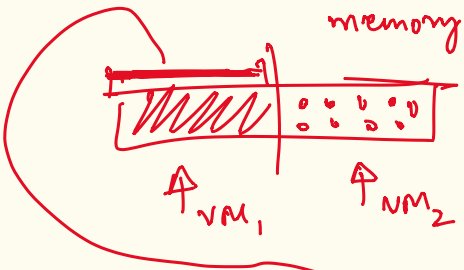
* # Types of VMMs.

① Bare-metal vs. Hosted
(vmware, xen) Type 1 (qemu-kvm, virtualbox) Type 2

② Full-virtualization vs. Para-virtualization
- Guest OS is unware of the VMM (vmware, ~~kvm~~ virtualbox)
- Guest OS "knows" about the VMM (xen, kvm)

Design 1: trap-and-emulate

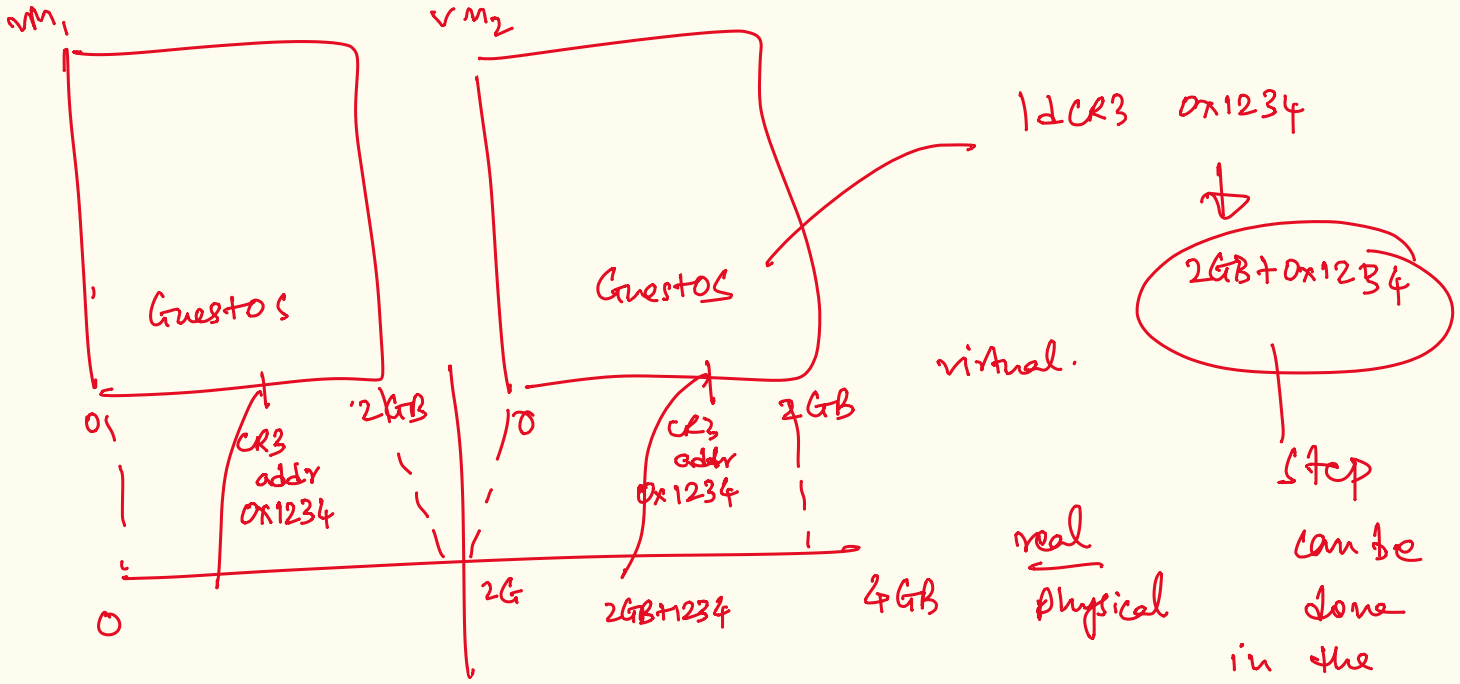
	Native	VM	
Ring 3	App	App	3
		OS	1
Ring 0	OS	VMM	0



PCs with x86.

eg: Guest OS writes to CR3 with a value (from VM1)
- ldCR3 —— MPL: 0
⇒ trap! CPL = 1 VMM's
— s/w to model PL 0 & jmp to trap handler
— inside handler —— "check" update value, modify it, write it.

- return to Guest OS. - yoohoo!



(ii) write to the IDTR.

- trap to VMM
  - check for range
  - modify / update from virtualized addr. to real addr
  - make a note of <IDT addr : VM>

(*) trap-and-emulate (virtualization handling).

CPL > MPL / RPL of instruction.

given an ISA:

P: set of privileged instructions ⇒ generate a trap when CPL < RPL

S: set of sensitive instructions. — which effect system/execution state.

- RPL = 0
- S ⊂ P

(*) x86 historically was developed w/o the VMM use-case.

e.g.: (i) popf ~ pops a word from the stack and writes to the flags register (eflags).

- one of the bits is used to enable / disable interrupts.
- needs PL=0 for correct semantics.

CPL≠0 ⇒ NOP!

= 17 or more x86 instructions which are not virtualizable.

⇒ do not generate a trap on CPL > 0

(ii) sidt ~ reads the IDTR into mem. addr/regs.

↳ works even if PL > 0

- Guest OS read succeeds ⇒ $IDT_G \neq IDTR$

⇒ knows about the hypervisor's IDT.

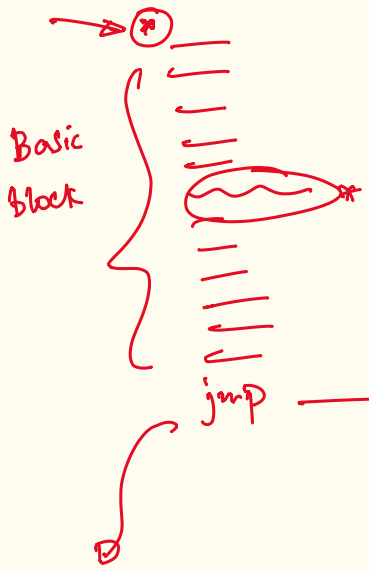
(iii) PL > 0 read code segment regs. to determine CPL. guest OS CPL > 0

Design 2: Scan-and-patch binary translation (VMware)

- scan instructions streams for critical instructions ↳ in S but not in P.
- replace w/ explicit trap to VMM.

= make critical inst^r. privileged!

↳ let trap-and-emulate take over.



w/. int. vector

trap —— < ip : instruction arguments >

(i) each block scanned before execution on CPU

(ii) scan & cache multiple bb blocks

- efficiency (?)
- resource control ✓
- equivalence ~

③ Design 3 : Para-virtualized CPU virtualization.