CPU registers (x86)

CR3 → Page table ← Hidden from programs

EIP → Instruction pointer ← Autoinc. by CPU

ESP → Stack pointer. Top of stack

EBP → Bottom of call frame

Managed by Compiler

General: EAx, EBX, ECx, ...

---

x86 rings

ring 3 — user app 1, app2

Privileged instructions
- Disable Intrpt
- Change CR3

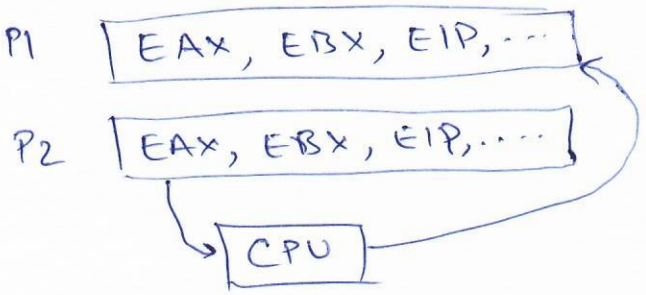
ring 0 — kernel

sys calls

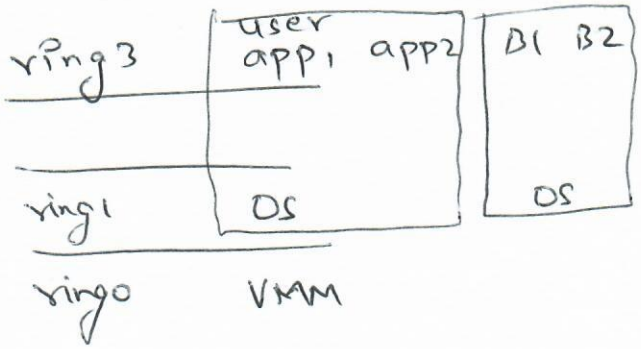
Context switch

atomic instructions

Process control block



Virtualizing OS



Virtualization in OS

Let process think it owns the hardware

- CPU register - Direct execution
- Address space

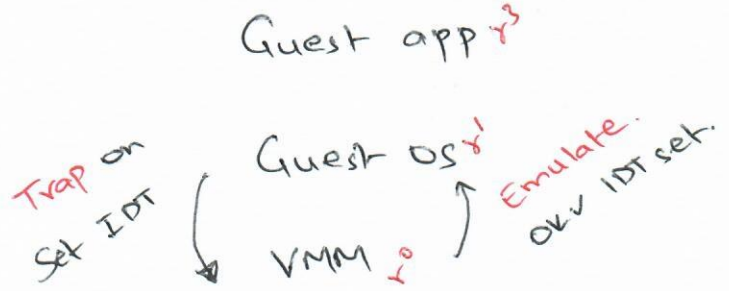
* Isolation *

Virtualization in VMM

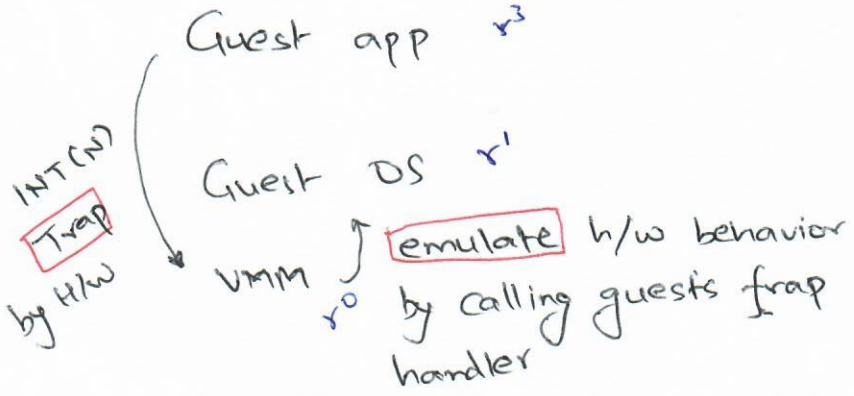
Let guest OS think it owns hardware

- CPU registers
(Guest) EAX, EBX, .. CR3, Interrupt flags
- Physical address space
- Interrupt descriptor table
- I/O devices

Trap and emulate hypervisors ②



Trap and emulate hypervisors ②



Correctness requirements ②

- All sensitive instructions must trap. Popek Goldberg theorem 1974s
- What happens if change CR3
  disable interrupt
  doesn't trap.

EFLAGS register ③

- Interrupt enable flag (IF)

not set by popf instruction
in ring 1 ↳ Does not trap

Why forget Popek Goldberg theorem? ①

Hot areas in 60s - 70s

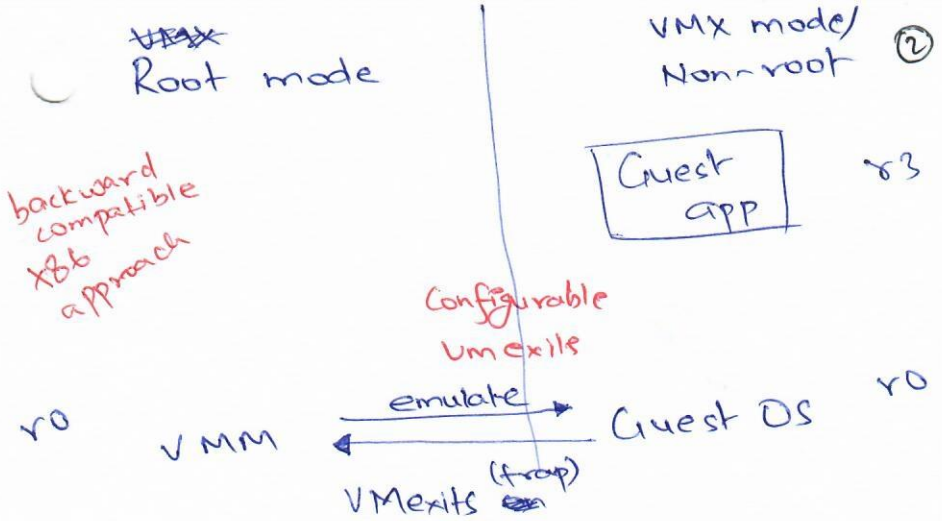
- expensive mainframe
- multiple operating system (no posix)
- let users run own OS ⇒ no rewrite of apps
- develop new OS

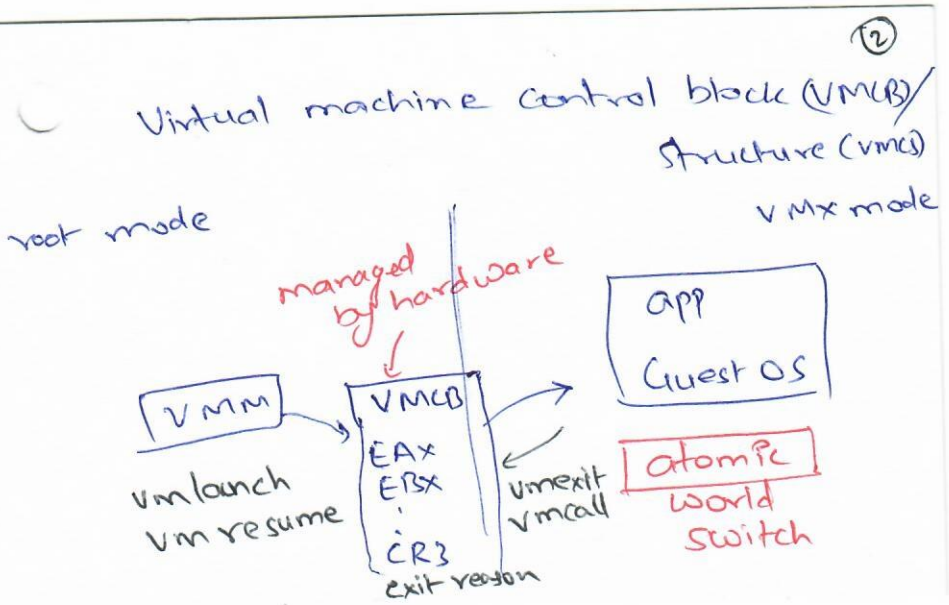
Not so hot in late 70s - late 90s ②

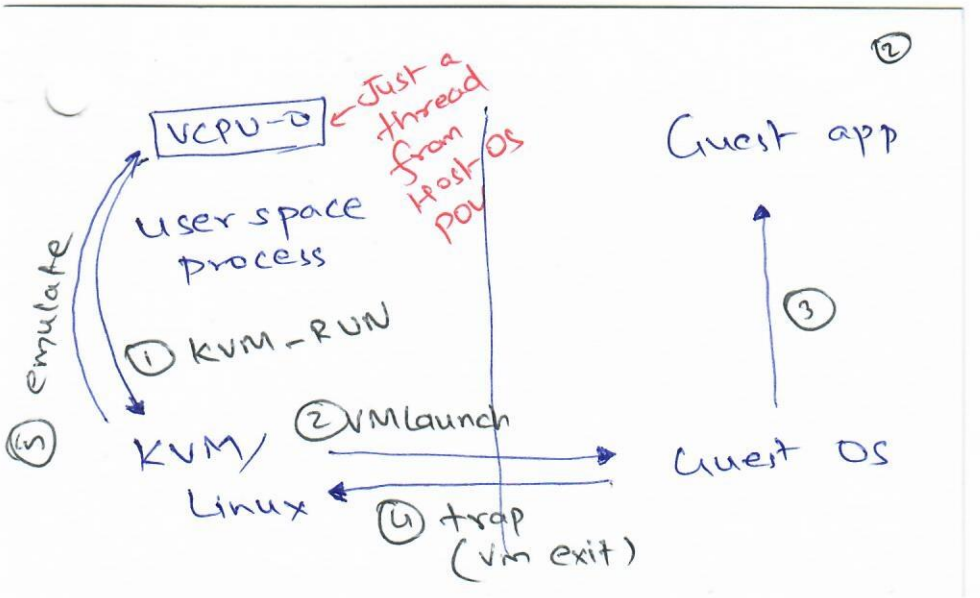
- Personal computing
- My own computer. One OS

Not again from late 90s

- Intrusion detection
- Fault tolerance
- multiple OS


~~VMX~~
Root mode
VMX mode/
Non-root
②
backward compatible x86 approach
Guest app
r3
Configurable VM exits
r0
VMM
emulate
Guest OS
r0
VMexits (trap)


②
Virtual machine control block (VMCB)/ Structure (VMCS)
root mode
VMX mode
managed by hardware
VMM
VMCB
app
Guest OS
EAX
EBX
CR3
VM launch
VM resume
VM exit
VM call
atomic
world switch
exit reason


②
VCPU-0
Just a thread from Host OS pov
Guest app
user space process
emulate
① KVM-RUN
③
KVM/ Linux
② VMLaunch
Guest OS
④ trap (VM exit)

Direct execution

①

When CPU goes to VMX mode, none of host processes are running

(Similar to when process runs, Os does not run)

KVM?

①

Abstracts over hardware details

Intel <> ARM <> AMD

different instructions

Semantics

Kvm API

---

ioctl based : KVM

①

- Create VM
- allocate mem to VM
- read/write virt registers
- Run vcpu
- Inject interrupts