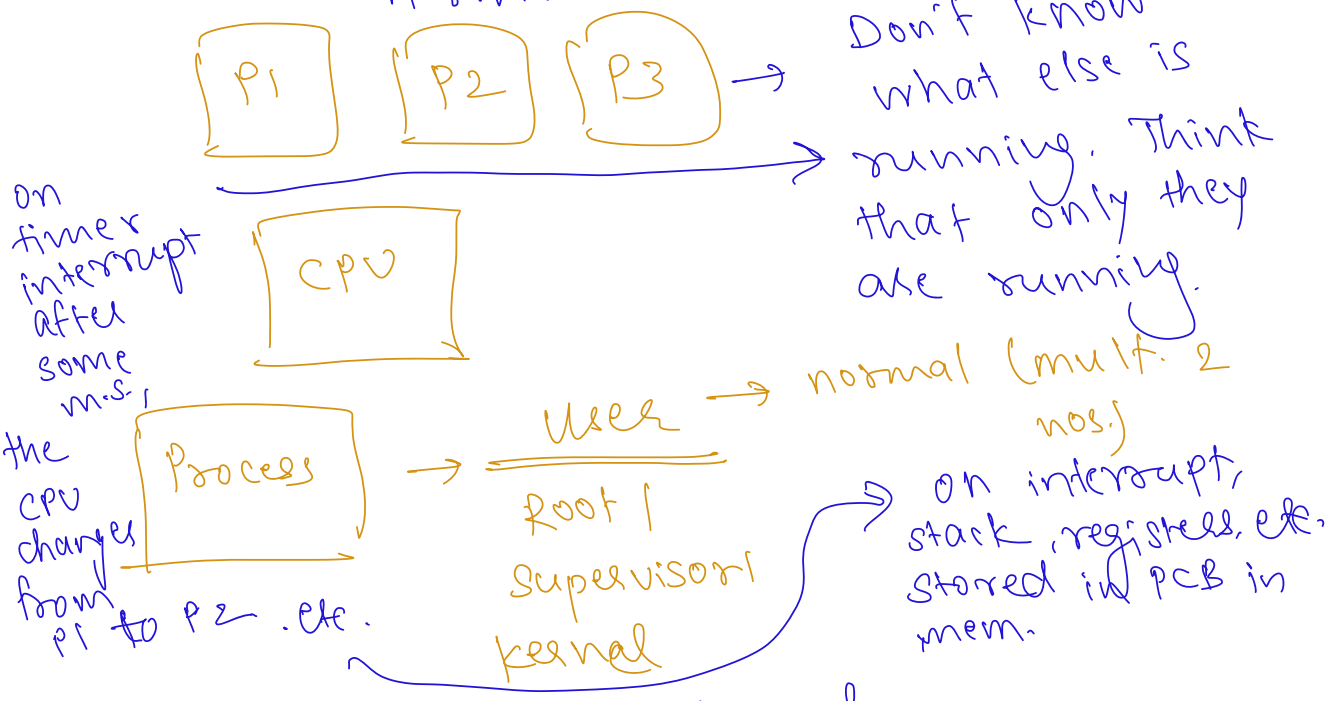
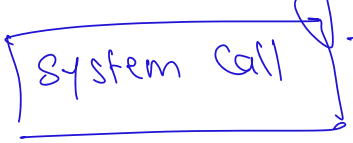
TYPES OF VIRTUALIZATION

- 1st topic: Virtual Machines
- Basics of V.Ms.
  - encourage: ask ques., keep it interactive.
- Before V.Ms., helps to understand
  - How CPUs virtualized in normal O.S.s.:



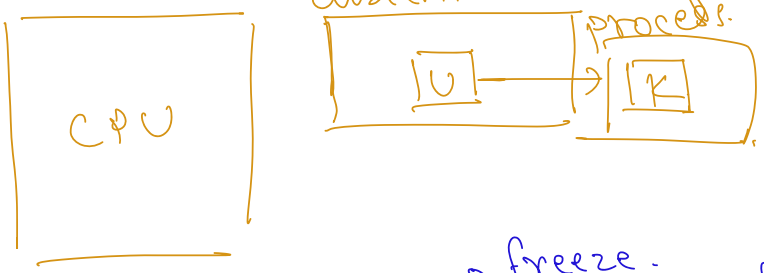
Whenever accessing



→ kernel invoked, does something on behalf of proc. & return to user mode

Kernel:

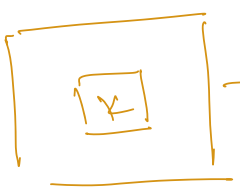
- Normal stuff: understood
- How does it work in hardware?



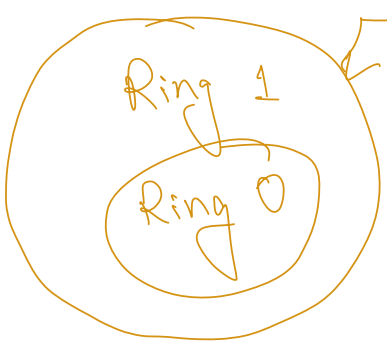
- on syscall: TRAP → freeze.
→ enter kernel mode
→ use data of user process
→ get result, return to process

e.g. write() system → call.

- kernel reads the syscall args & handles syscall.



→ Ring: Privilege level.

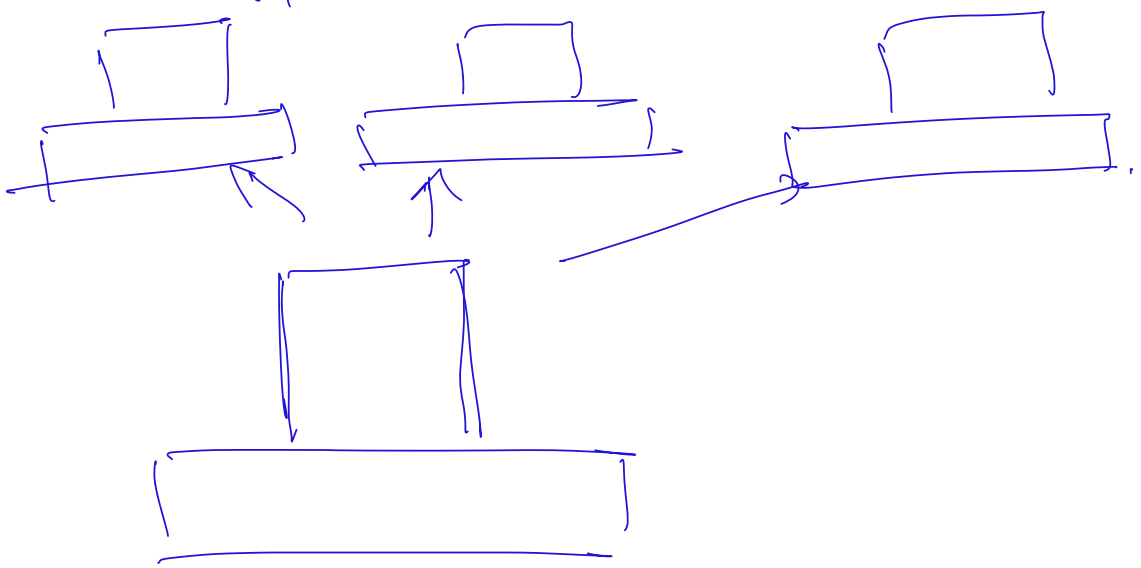


provided by hardware.

certain code only executed for certain privilege level.

All this required in V.MS

VIRTUAL MACHINE.



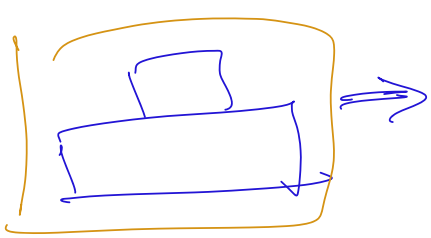
1974 ⇒ CP-67 IBM.

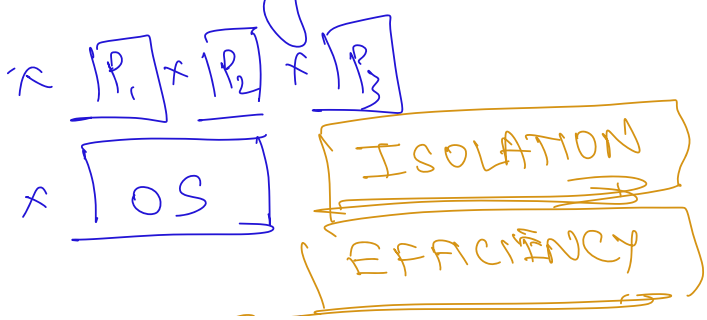
- We have big machine, severely underutilized, how should we increase utilization?
- Idea: virtual machine. to each user

→ A step higher than just running processes:

① Running a process. One malicious bring down whole kernel.
- no strict isolation

— VM: much stronger isolation.





ENCAPSULATION

→ All state neatly wrapped up & moved between diff. physical machine. (PORTABILITY).

↳ somewhere close to perf. of orig. machine
- 1000 ops is same ballpark.
- person running V.M., seemless.

Recap → ISOLATION, EFFICIENCY, ENCAPSULATION.

seminal work: Goldberg & Popek.
- 1974: consider same things as today → PhD thesis
— textbook.

TYPES OF VIRTUALIZATION

- 3 mainly used today. Focus on them in class, but not all.

1) Hosted Interpretation.

→ user mode
→ read / write

→ Tells which privilege level.

MOV R1, 1

MOV R2, 3

MVLT

CLEAR → If bit is 1.

CANNOT WRITE '0' TO ANY REGISTER.

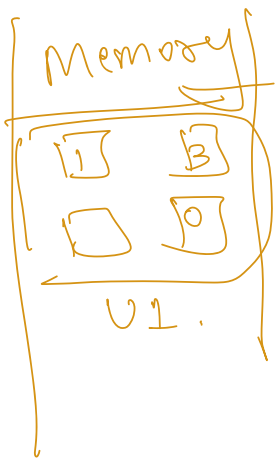
NEED TO USE CLEAR COMMAND

3 users, see all registers. Each think they are only ones using h/w.

—we need to virtualize hardware

→ We need more than the resources to virtualize.

→ User 1 running, save state, some place to keep state.



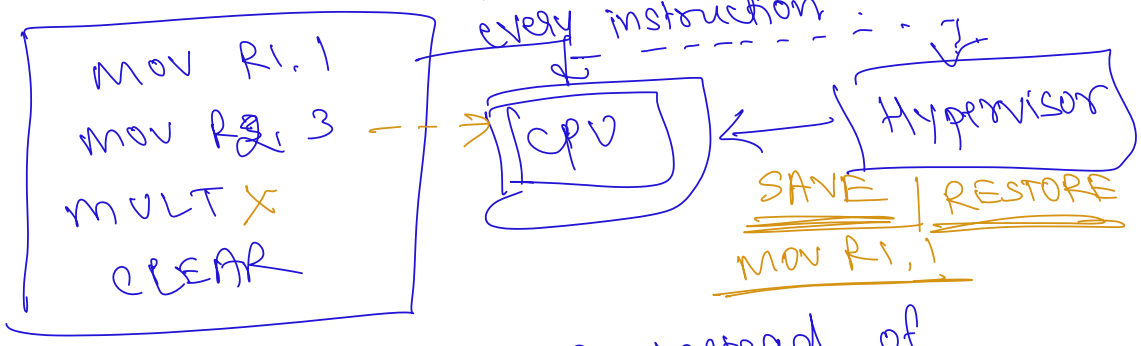
→ slower than R1, R2, R3.

→ Any time switch, save state & slow.

→ Memory 10x slower than registers.

HOSTED INTERPRETATION → Trap for each instruction.

e.g. User 1.



Every instruction: TRAP, instead of user doing, we (hypervisor) execute.

Raw instruction: never on CPU

when interrupted, Hypervisor: SAVE TRAP; interpret what it is trying to do; do it.

Hosted Interpretation: Hardware can be wildly different.

: Very slow

→ Trap on every instruction. Emulate with software if hardware doesn't supported.

→ How to run gameboy console on P.C?

↳ Hosted interpretation.

→ Mario game on browser? → Hosted interpretation.

→ Works well for games where speed of game is human interaction.

= Baremetal: 1B inst/sec.
even 100x slower: very fast.

II Trap and Emulate

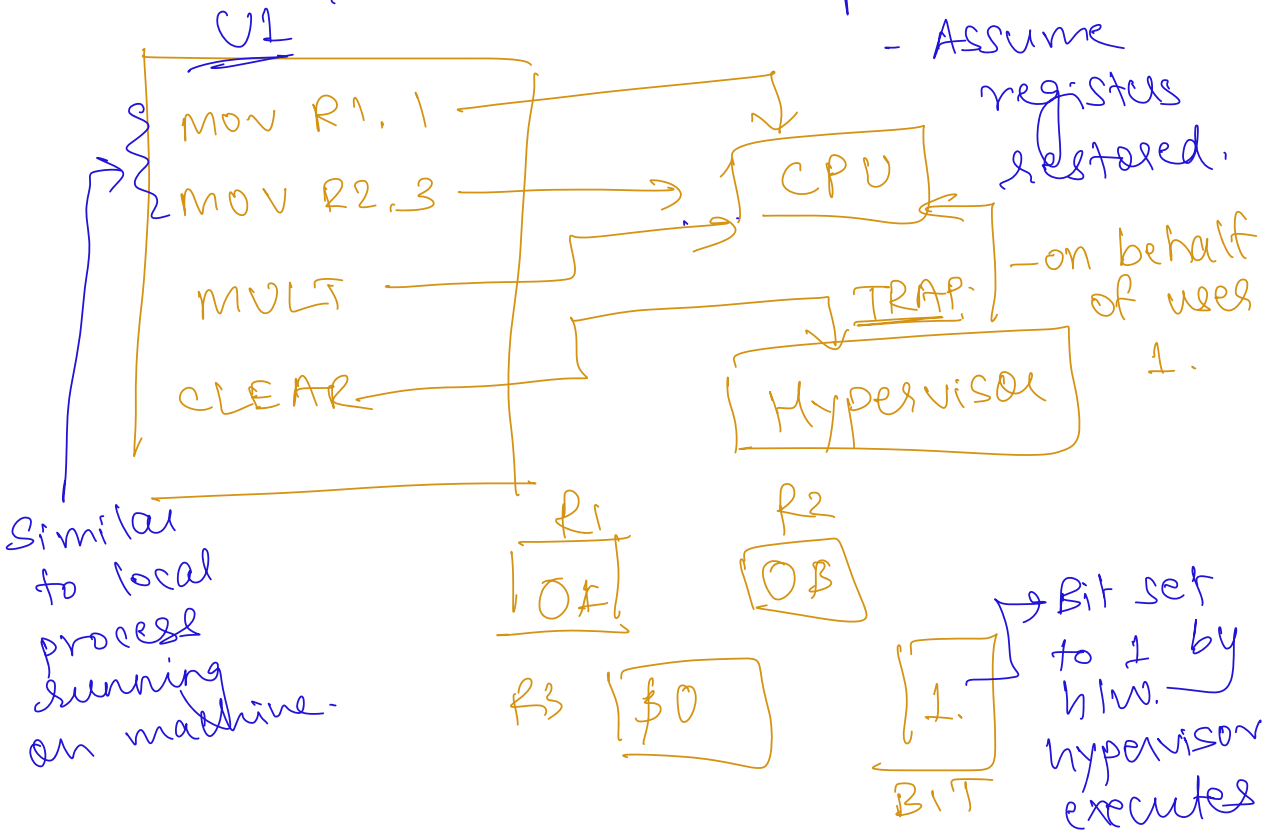
- Disadvantage: trapping, interpreting & emulating every inst.
- If inst. was supported by h/w, you don't actually need to do anything special.
- Trap & Emulate takes advantage of this - most instructions don't do anything.
  → Just restore state & let run.
- But every instruction not allowed. But privileged instruction need trap.

→ TRAP on PRIVILEGED INST.

Otherwise it would just fail...

- After trapping, hypervisor can emulate the instruction for v.m.

Security:- Big security consideration
- cannot allow v.m to run privileged instruction
  - modify hardware.
  - wipe out everything & take over machine
- Every privileged inst. executes results in a trap.



PROBLEM

A different instruction:

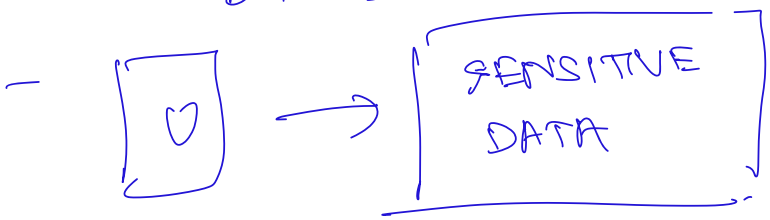
READ BIT

↳ If there are certain instructions where once it is done, you expect the bits to be in certain state.

- e.g. After CLEAR, expect to be in privilege state.

In non-virt., if you read bit, it is set to 1.

- If Bit 1 not synchronized with bit in memory
- After SAVE & RESTORE, if the bit is not set to 1, then problem.



- on read of SENSITIVE DATA, should trap.
- If instr. set allows to read SENSITIVE DATA w/o trap, you can know that you're not in hypervisor mode & break through illusion

- Physical machine: BIT would be 1
  VM: may not be 1.
  - Identify whether VM or physical

machine.

---

x86 Hardware.

EFLAGS Register.

one bit.

supervisor mode.

NO TRAP!

Non-virt.:

OS ⇒ Bit is 1.

Virt. ⇒ Bit is 0.

→ if USER APP, expect it to be 0

→ If O.S of VM, expect it to be 1.

↳ This is why we needed to wait for VMs

→ x86 was not considered to be virtualizable

↳ Read SENSITIVE DATA WITHOUT TRAP.

→ 1998

So, trap on everything.

1974: Goldberg & Popec: Conditions for whether you can virtualize something or not.

(3) Binary Translation.

- Trap & Emulate:
  Lets execute: If safe, execute
  if not safe, trap &
  make hypervisor emulate
  it.

Reactive

Binary Translation : Proactive.

Take all instructions,
translate program so that you
replace privilege instructions
with safe ones.
e.g with CLEAR

- change privilege instructions to
  SAFE instructions that operate
  on the STATE of VM instead
  of hardware.

- COMPILATION PHASE
  - VMware founded on this idea in

early 2000's.
- No problem with EFLAGS.

(4) Hardware-supported Virtualization.

- Binary Translation:
  problematic is O.S
  - translate O.S in the VM.
  - 50-100 M LOC
  - Do at runtime.
    ↳ overhead.

- Intel & AMD: Build into Hardware.
  → H/w gives copies of h/w state for each V.M.
  - e.g: RESTORE state for VM-1. restores from particular memory area.
  - SENSITIVE DATA: trap.

- 2005-2009
  ↳ No longer need Binary Translation.

(5) PARAVIRTUALIZATION

- Every new O.S. support paravirtualization.