- Talk in detail about 5 kinds of virtualization
- Look at why x86 is not virtualizable in more detail.
- Recap:
  1) Hosted Interpretation:
     - Emulating hardware
     - Every hardware instruction that emulated device execute, we take it & execute in software.
     - Very flexible but very slow.
       ↳ can be migrated to any hardware

→ Q. Why not load the binary of application in the hypervisor instead of trapping on every instruction?

A.) Hypervisor: Ring-0.
Guest App: Ring-3.

① Load all malicious code in Ring-0. May be some security vulnerabilities?

② If the code crashes, does it bring down hypervisor?

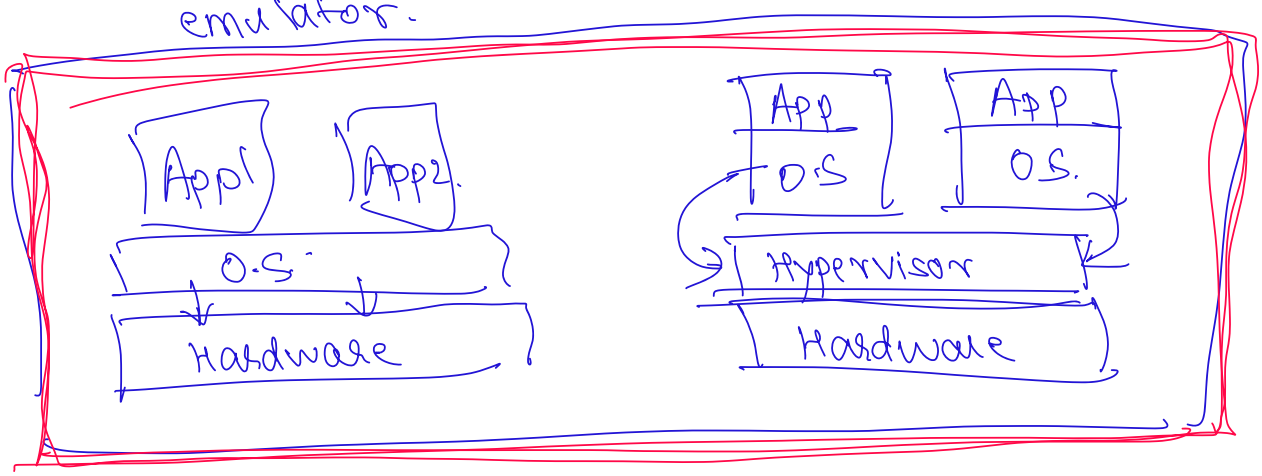
(3) Trapping itself is not the problem. Problem is "emulating" hardware instructions in software.
All checks, & emulation still required,

(4) The way to put code of Ring-3 in Ring-0, build a kernel module, & then add code in that.

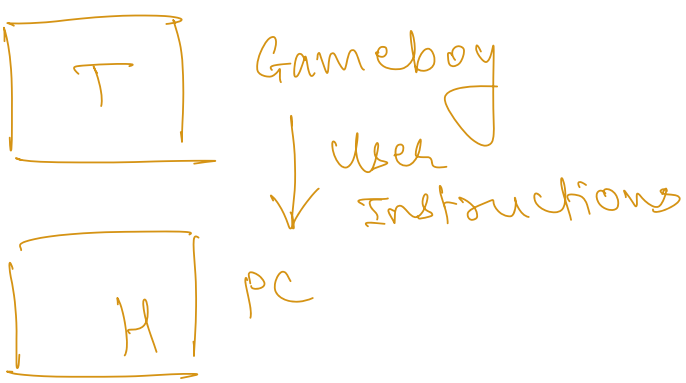
(5) How does it know when to load code in hypervisor? No trap generated, after compiling code.

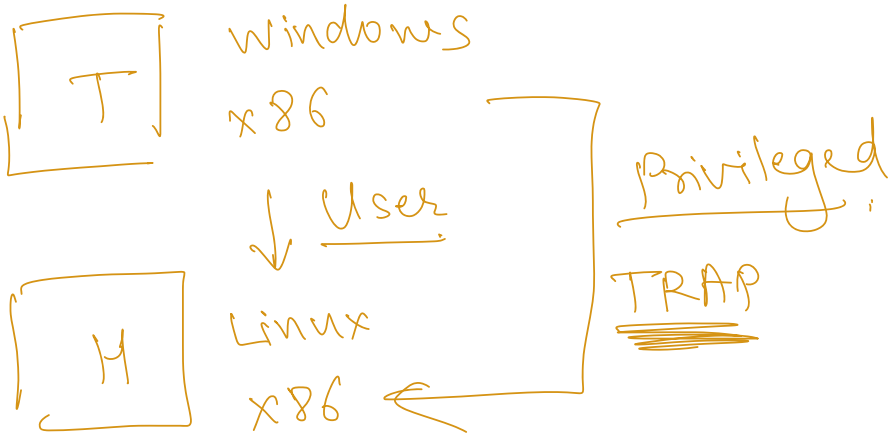
---

Q. What is diff. between hypervisor & O.S.? Hypervisor is a "hardware" emulator.




2) Trap-and-Emulate






- user instructions means instructions that don't modify privileged host state or read privileged host state
  Similar to user process in normal O.S.

- However, when we have privileged instruction, trap.
  - System traps from target to host, & host simulates what should have happened to host architecture.
  - Target arch. does not know it is virtualized, it thinks it is running on hardware
  - Host arch will do what H/w

would have done.

TRAP & EMULATE : START OF BEING ABLE TO EFFICIENTLY VIRTUALIZE SYSTEMS.

- If vast majority user mode, directly run on host arch.
  → very efficient.
- e.g. 99% time running user code

Requirements to run efficiently

Goldberg & Popek ★★

1) Two modes of execution: user mode & kernel mode
   we need to support it in hardware.
   user mode must not be able to run kernel mode code.
2) user mode → Privileged instruction
   Result in TRAP
   - Need mechanism to route the TRAP to underlying host arch.

3) Sensitive data (read or write)
Result: TRAP

- sensitive instructions must behave like privileged instructions.
- sensitive data: ① what privilege level am I in?
② what is current time on system.
③ How much time has elapsed since last event.

All this should be handled by the VMM / host, else guest will realize that it is virtualized.

x86 violates: with 17 instructions

No TRAP: Host arch. cannot intercept, so x86 using trap & emulate is not virtualizable.

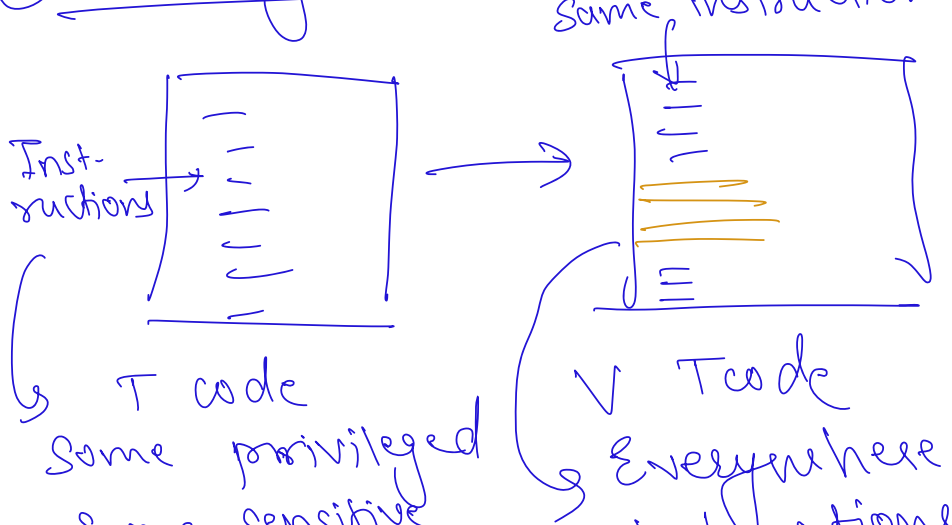
---

- Why is it bad that guest O.S. knows it is virtualized?
- Assumptions in the O.S. that it expects that it is in

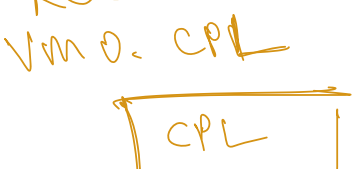
highest privilege level in kernel code. e.g. assert().
- Assert() fail: O.S. cannot run.
- If you take unmodified O.S. & run it on something like this, won't work. Assumptions incorrect.

---

## (3) Binary Translation.




e.g. Read CPL → x86 doesn't throw trap. O.S. would fail.




Anytime target says read CPL, we read

state for every V.M. maintained by hypervisor

CPL field from state.
D.S. always up-to-data.
∴ when running kernel code, CPL elevated to Ring 0. when user mode, CPL elevated to Ring 1.
We fake hardware state for V.M. to read instead of actual h/w state.

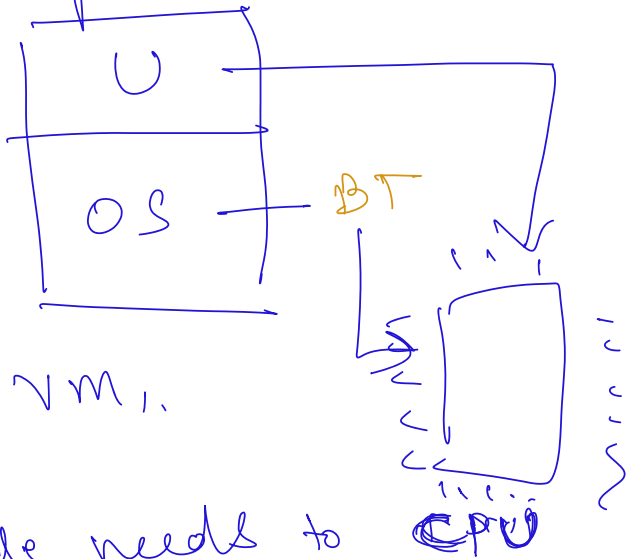
- We control fake h/w state. So we ensure we give V.M. what it expects.
- Privileged or sensitive instructions in O.S. code translated to something else.
- We don't need to wait for privileged inst. to throw trap.
  - (give path of trap.)
  - Do it in proactive way by changing code before execution.
  - code doesn't have any privileged or sensitive instructions.
  - Both replaced.

Translation was big breakthrough.
Before: people didn't know how to virtualize x86. But with B.T. you could take sensitive instructions & replace with user instructions. & run on CPU.
This allowed x86 to be virtualized

## QEMU uses binary translation

- Small systems: Binary translation can be done in one shot.
  Large systems: cannot.

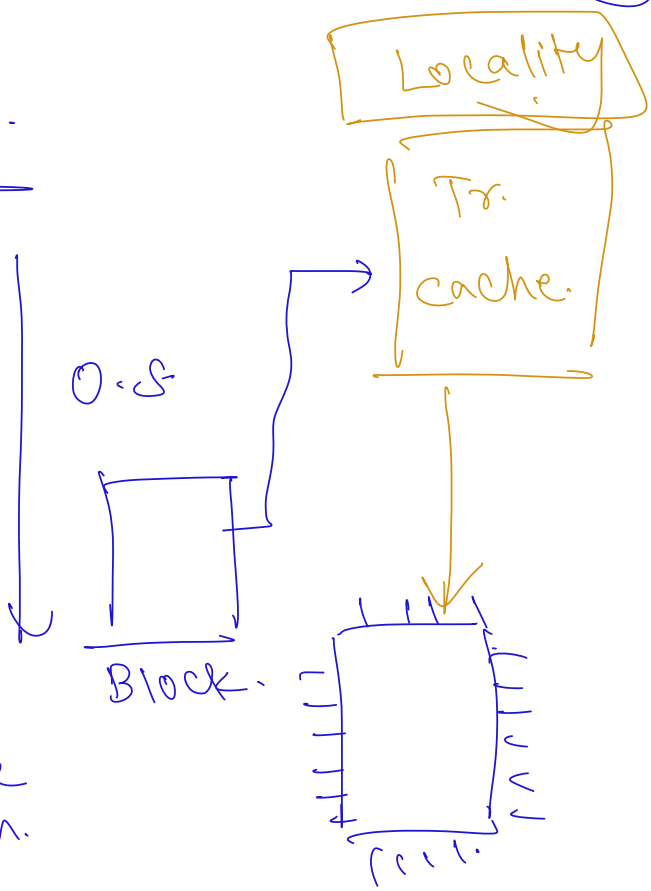
e.g.




O.S code needs to be translated
O.S. is the beast. e.g. windows has 50-100M LOC

Linux : 5OM LOC.

- Infeasible to translate entire thing & run it.
- Runtime B.T.




- we re-run code again & again.
  e.g. for loop, function called again & again
- strong temporal locality.
- So translating & keeping in cache helpful, for reusable code
- translation not cheap.
  - have to do in real time.
  - it would slow down system throughput.

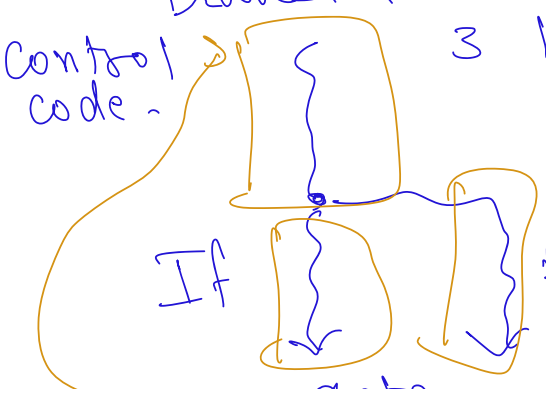
→ take block of code, put it in cache & hope to reuse it again & again.

→ makes it practical
- code with no locality B.T. very bad. Cache won't help.
- O.S. code also contains user instructions & must be translated.

---

Takeaway big ideas:
- what is binary translation?
- why does it work?
- Details are fine. Labs will make you understand details.

- when translating code to translation cache:
  3 blocks. Each straight line, one block.
  - once enter control block, want to be able to execute all instructions one after another w/o jumps.



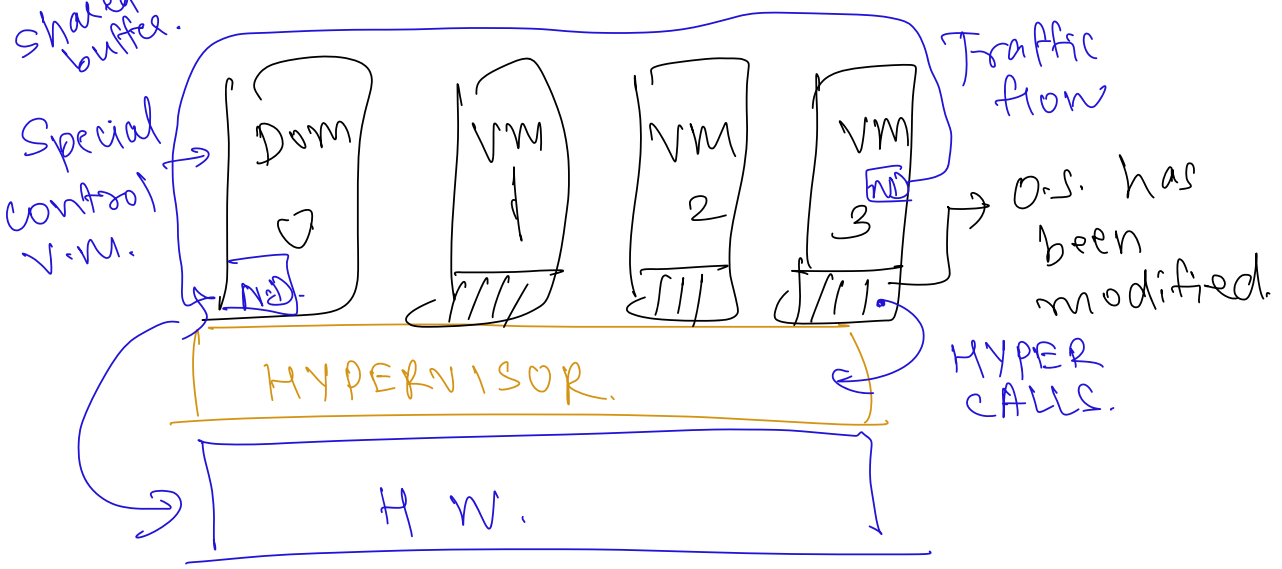
goto.
↳ Need to translate jumps.
Every Jmp receives special attention.
Translate correctly.

SELF MODIFYING CODE : code is rewriting what happens in branches
Bad: - we translate once, put in cache, don't translate again.
self-modifying: code changes everytime it runs.
- All things marked read-only
if code tries to change, we flush cache & re-translate.
- previous translation won't work.
SLOW

- Intermediate architecture:
very broad, general & can be translated to any host architecture.
- Hypervisor in B.T. does not need to be in kernel mode

## 4) Paravirtualization

- We will talk in more detail in later class
- Dedicated class for paravirtualization (Xen).




- O.S. knows it is virtualized.
- Can do Hypercalls
- Any time actual hardware state needs to be changed.
  - e.g. VM wants to add new networking driver.
  - Hypercall to hypervisor.
  - Hypervisor needs to be involved

in such things
- instead of modifying hlw, O.S. does hypercall.
- way more efficient.

Trap & Emulate: trap to do anything simple.
e.g. trap to do something, trap again.
e.g. for setting up virt. N.P., we should do 3 privileged writes

WRITE → TRAP
WRITE → TRAP
WRITE → TRAP

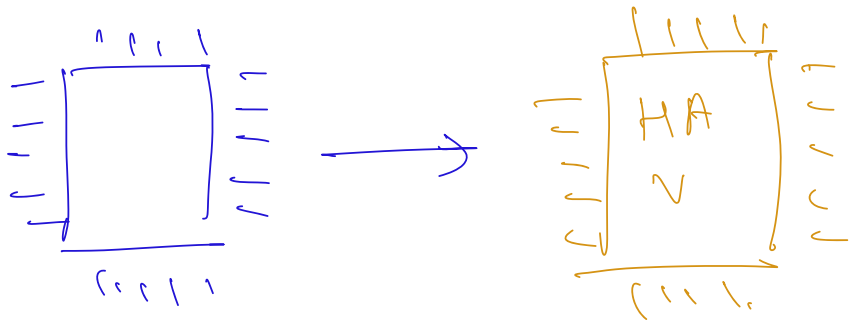
in paravirtualized:
Hypercall just once
3x gain.

Actual: Lots of traps. → Like 24 traps for one thing.
- O.S. modification
  - hard to keep up.
  - devs hate maintaining code
  - file bugs, patches

- both paravirtualized & non-virtualized.
- Amazon used to support paravirtualization
- that ended with hardware supported virtualization.

## 5) Hardware assisted Virtualization

Historical context:
- Trap & Emulate: could not emulate x86
- Binary Translation (1999) : big moment. x86 virtualization.
- ParaVirtualization
  - for a while both were used & para had better performance.
- Intel & AMD realized that virtualization was going to be big.
  - They redesigned chips to do virtualization
  - state-of-the-art.




↳ plugs holes we had with x86

↳ if sensitive data, trap generated

Goldberg & Popek condition met.

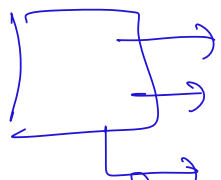
- Binary translation not needed.
- Hardware guarantees it for you.

VMCS → Virtual machine control structure

Hardware gives control. for things.

e.g. how hardware understands page table walking.

Similar, h/w understands vmcs.



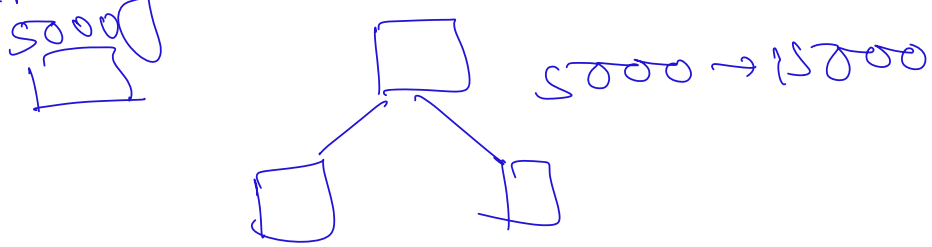
Page Tables
CPU
Trigger a trap.

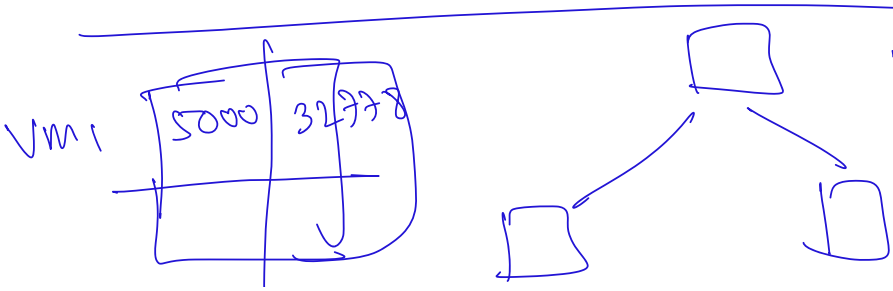
Subset of state that we would maintain for each v.m.

what inst. should trigger a trap?

Bit field: every inst. is a bit. You can mention any inst. to trap for you, via VMCS.

- We can tell hardware when to trap & when not to trap.
- We control what to consider privileged & sensitive.
- Hardware does binary translation.
- Other hardware support:
  ① memory virtualization:



if addr. of page table of
vm1 & special p.p. in vmcs,
it will figure out 5000 → 15000 &
15000 → 32778 in hardware.

without hardware support:
shadow page tables (we will learn)
SLOW, TRAPS

— Virtualization simplified significantly.
— Before: very complex &
worse performance.