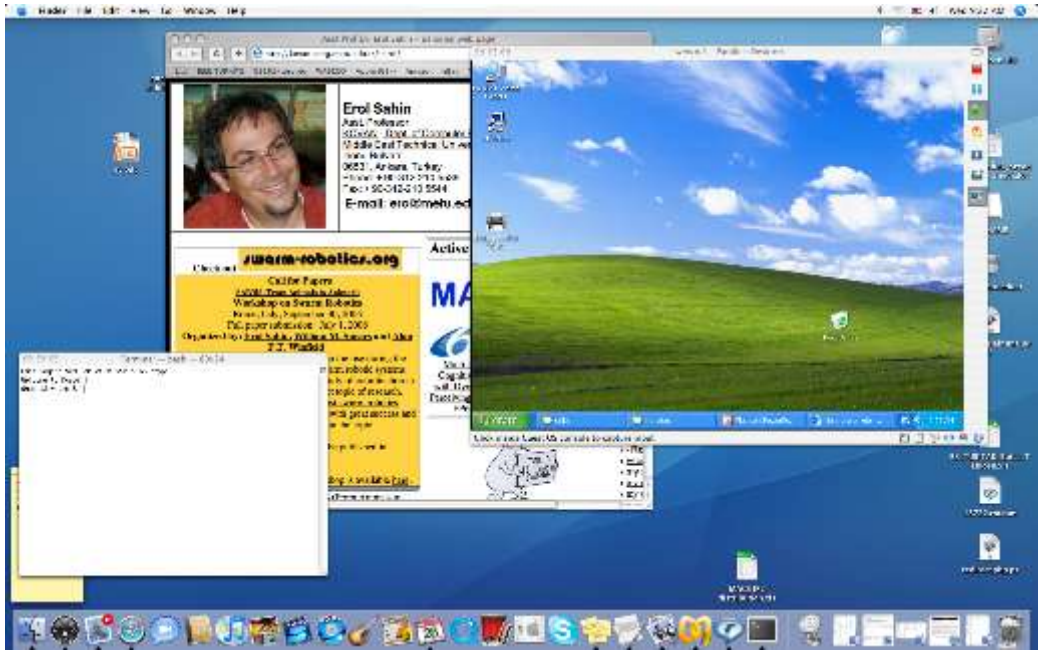
Virtualization

Virtualization

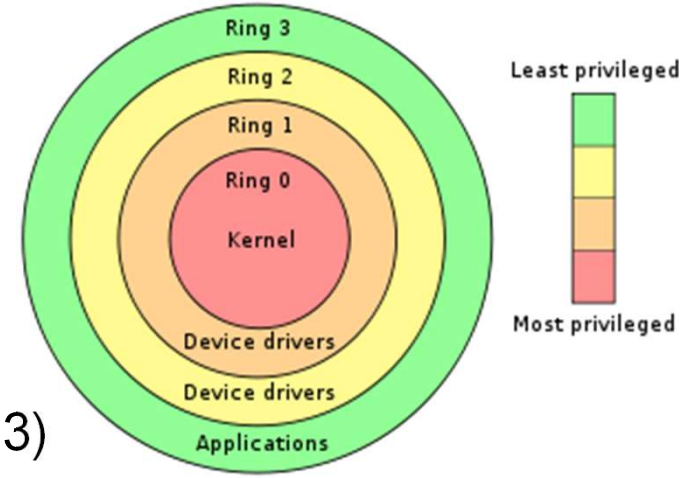


- A single computer system hosting multiple virtual machines each potentially running a different OS.
- A 40 year old technology dating back to IBM/370 that is making a come-back in the recent years.

- Robustness against software failures
- Able to run legacy applications on OS's that are not supported on current hardware or available OS's.
- Good for software development that targets different OS's.

Protection Rings – x86

Protection rings



APPS

User space (lower privilege: ring 3)

System call/ trap

Kernel space (high privilege: ring 0)

OS
(supervisor mode)

Have rights to access some special CPU instructions

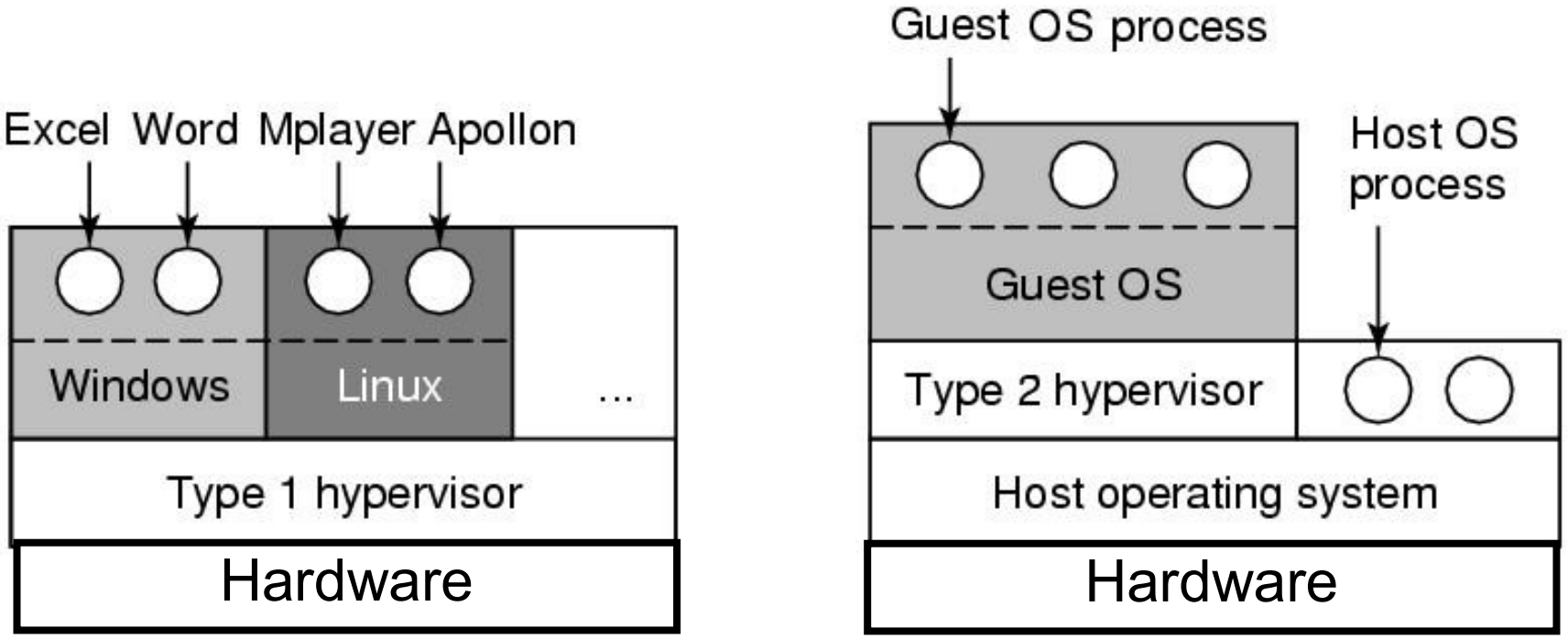
interrupt

Hardware

Requirements for virtualization

- **Approach: trap the "privileged instructions" in the user mode**
  - typically related to kernel functions, such as I/O instructions or instructions that changes MMU settings
  - emulate them within the guest OS
- **Requires help from hardware**
  - AMD and Intel CPU's have created virtualization support after 2005
    - Containers in which virtual machines can run
    - The program runs until it creates a trap
    - Traps handled by the hypervisor to emulate the desired behavior

Two approaches

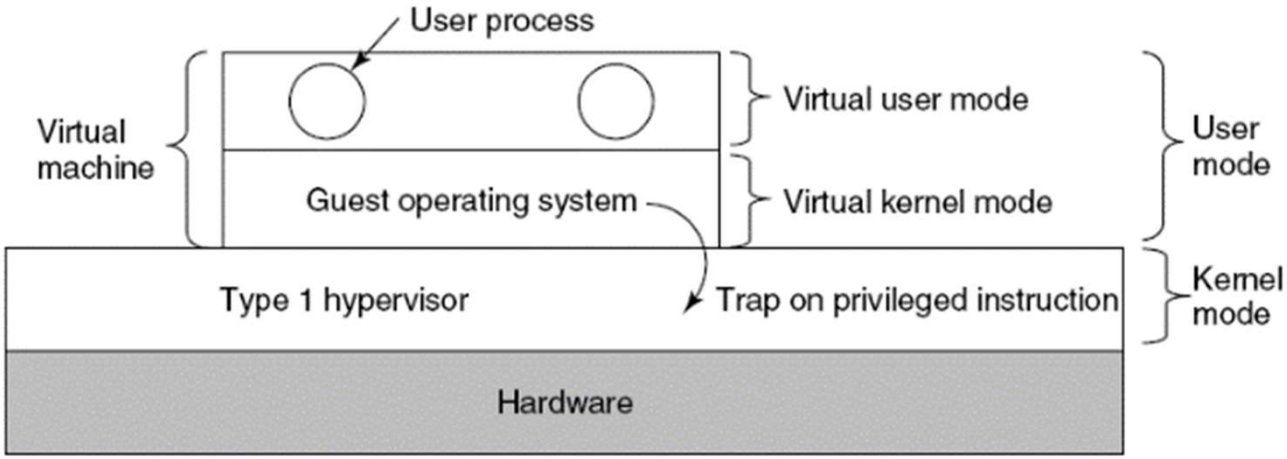


- **Virtualization is implemented by hypervisors that act just like the real hardware.**
- **Type 1 hypervisor**
  - Virtualization is implemented as part of the hosting OS at the kernel level.
  - Support multiple copies of the actual machine, called virtual machines.
- **Type 2 hypervisor**
  - Virtualization is implemented as a user program running at the user level.
  - It “interprets” the machine’s instruction set which also creates a virtual machine.

https://www.youtube.com/watch?v=FZR0rG3HKIk

Type 1 Hypervisors

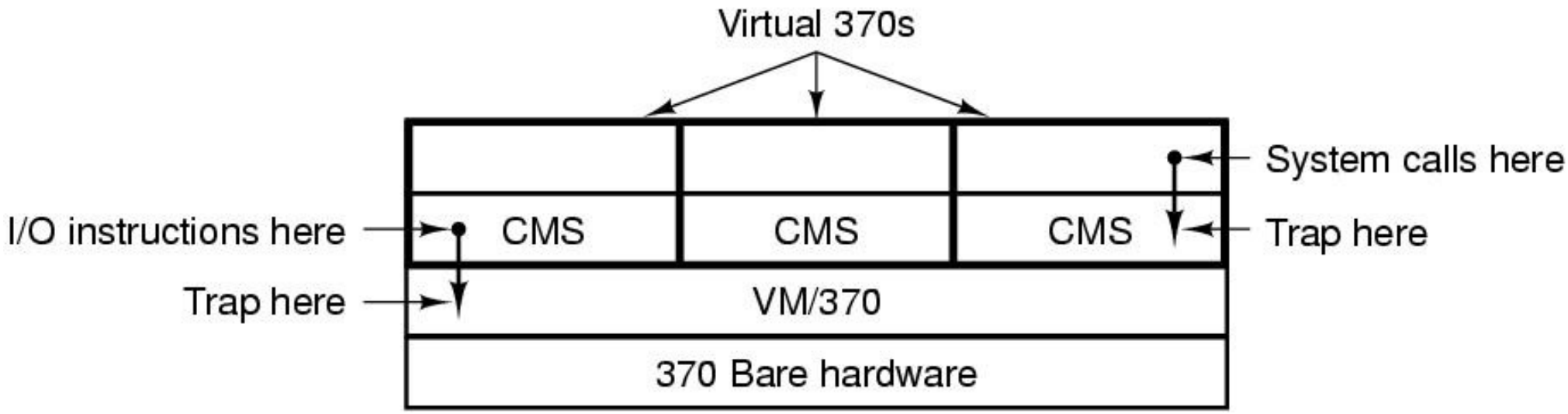
- The hypervisor runs on the bare hardware as part of the OS.
- The virtual machine runs as a user process in user mode.
  - Is not allowed to execute “privileged instructions”.
- The guest OS thinks it is running in kernel mode, although it is in user mode. (virtual kernel mode)
- When the guest OS executes a “privileged instruction”, it generates a trap (thanks to the VT support from hardware) in the host OS kernel.



- The hypervisor inspects the instruction if it was issued by the guest OS running in the virtual machine.

- If so, the hypervisor emulates what the real hardware would do when confronted with that “privileged instruction” executed in user mode.

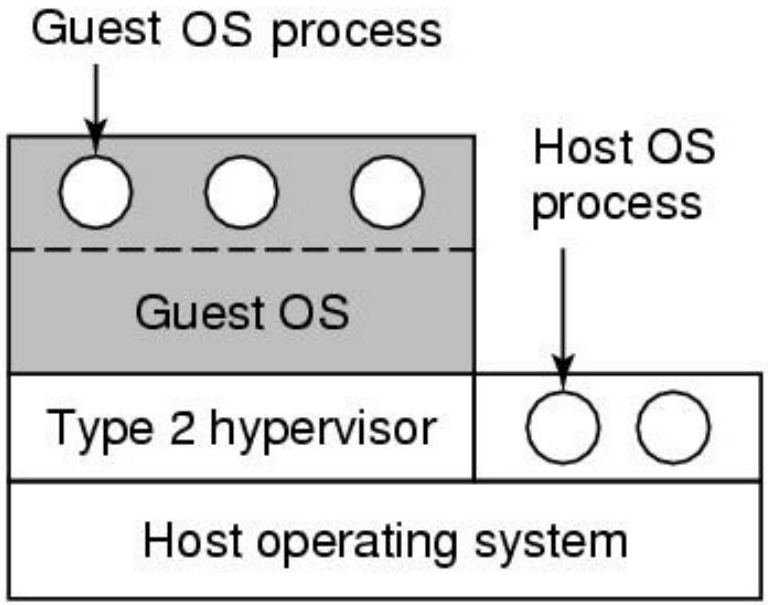
Type I: The structure of IBM VM/370



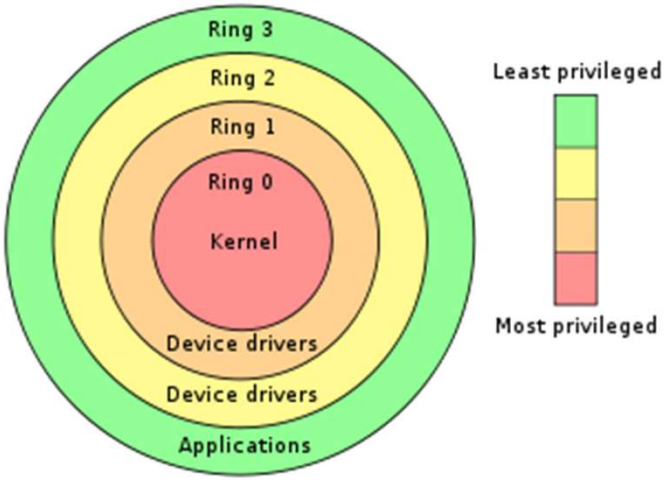
- When a CMS program executed a system call, the call is trapped by the CMS (guest OS)
- CMS then issued the normal hardware I/O instructions for reading its virtual disk, etcetera.
- These I/O instructions were trapped by the VM/370, which then performed them as part of its simulation of the real hardware.
- In its modern incarnation, z/VM can run multiple OS's such as AIX's or Linux.

Type 2 Hypervisors

- The hypervisor runs as a user process on the host OS.
- The virtual machine runs as a user process in user mode.
  - Is not allowed to execute “privileged instructions”.
- The guest OS thinks it is running in kernel mode, although it is in user mode. (virtual kernel mode)
  - When the guest OS executes a “privileged instruction”, the hypervisor emulates it.



x86 virtualization



APPS

User space (ring 3)

System call/ trap

Privilege (ring 1/2)

OS (VM)

Have rights to access some special instructions

Hypervisor

Privilege: ring 0

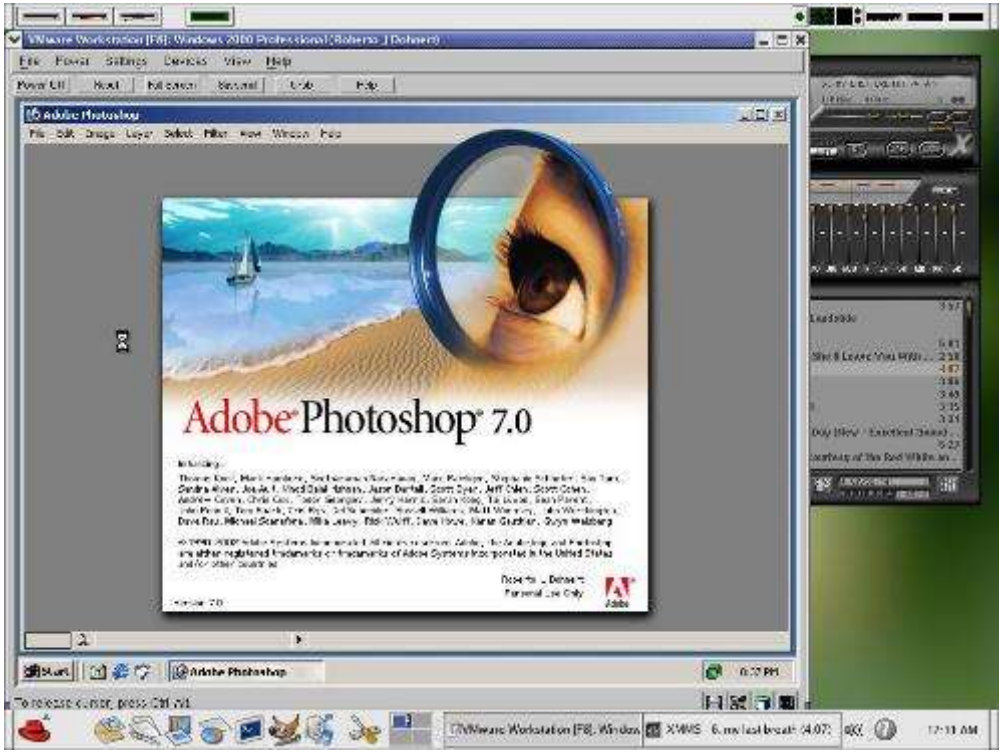
Have rights to access some special instructions

interrupt

Hardware

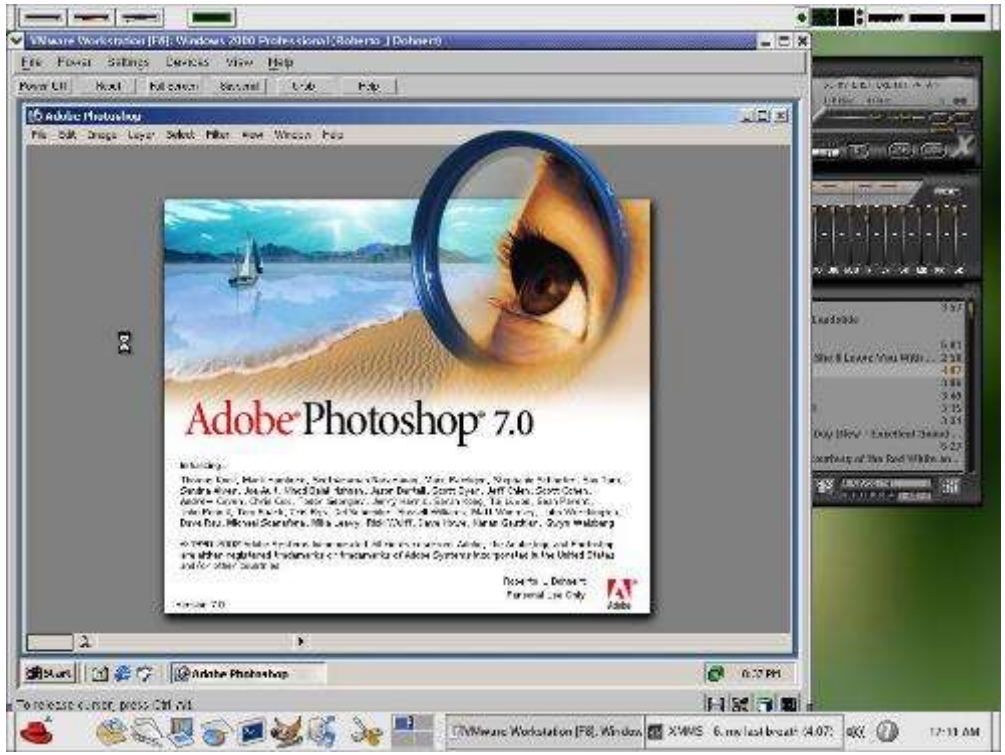
Type II: VMware: A case study

- VMware runs as a user program on the host OS, such as Windows or Linux.
- When it starts it acts like a newly booted computer, and expects to find a CD-ROM containing an OS. It then installs the (guest) OS on a virtual disk.
- Once the guest OS is installed on the virtual disk, it can be booted to run.



VMware: A case study

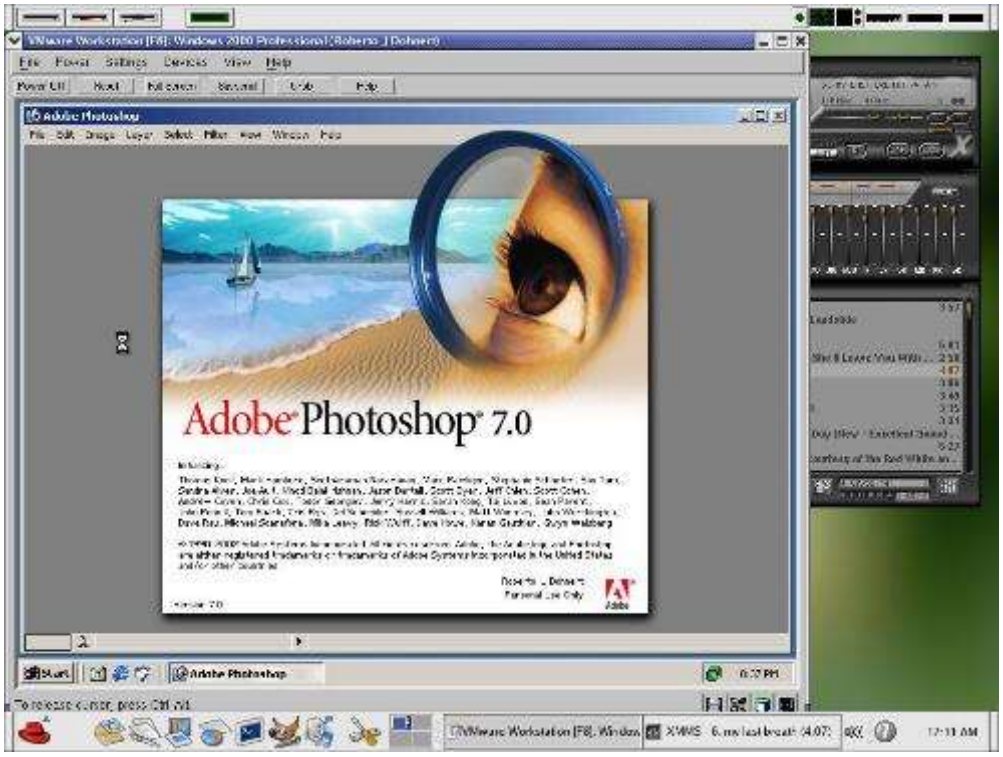
- When executing a binary program, it scans the code looking for basic blocks, that is straight runs of instructions ending in a jump, call, trap or other instructions that change the flow of execution.
- The basic block is inspected if it contains any “privileged instructions”. If so, each one is replaced with a call to VMware procedure that handles it. The final instruction is also replaced with a call into VMware.
- Once these steps are made, the basic block is cached inside VMware, and then executed.
- A basic block not containing any “privileged instructions” will execute as fast as it will on a bare machine, because it is running on the bare machine.
- “Privileged instructions” that are caught in this way are emulated. This is known as binary translation.



VMware: A case study

- After the execution of a basic block is completed, the control returns back to VMware, which picks its successor that comes next.
- If the successor is already translated, it can be executed immediately.
- Eventually, most of the program will be in cache and will run at close to full speed.

- There is no need to replace sensitive instructions in user programs. The hardware will ignore them any way.



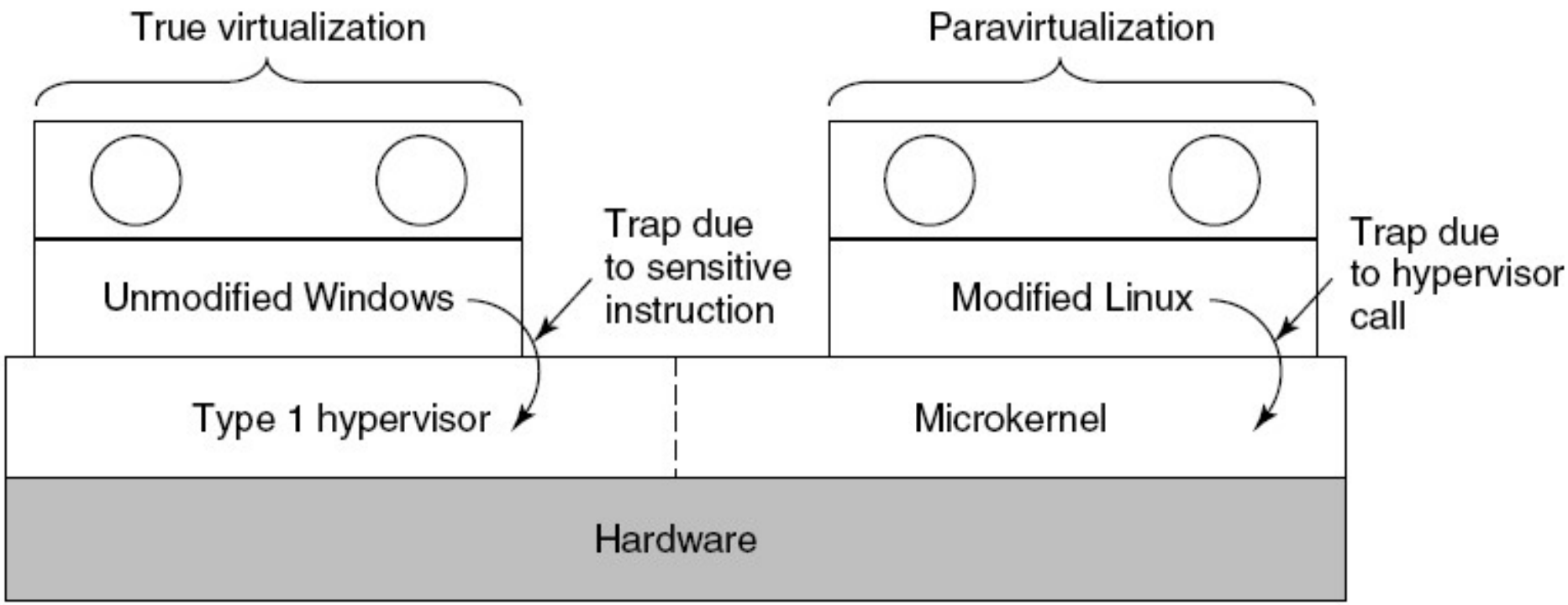
Type I vs Type 2 hypervisors

- Unlike what you would expect, Type 1 is not a winner at all times.
- It turns our that trap-and-emulate approach creates a lot of overhead due to the handling of the traps (including context switches) and that the binary translation approach (combined with caching of translated blocks) can run faster.
- For this reason, some type 1 hypervisors also perform binary translation for speeding up.

Paravirtualization

- Both type 1 and type 2 hypervisors work with unmodified guest OS's.
- A recent approach is to modify the source code of the guest OS such that it makes hypervisor calls instead of executing "privileged instructions".
- For this the hypervisors have to define an API, as an interface to the guest OS's.
  - But this approach is similar to the microkernel approach.
  - A guest OS whose "privileged instructions" are replaced with hypervisor calls is said to be paravirtualized.

Paravirtualization



- **A hypervisor supporting both true virtualization and paravirtualization.**

Virtualization issues

- **Memory virtualization**
  - How to integrate the paging of the host OS with the paging of the guest OS
- **I/O virtualization**
  - Do we use the device drivers of the host OS or the guest OS?
- **Multi-core**
  - Each core can run a number of virtual machines
  - Sharing memory among virtual machines?
- **Licensing**
  - Some software is licensed per-CPU basis.
  - What is you run multiple virtual machines?

For those who are curious…

- **Virtualization**
  - https://www.youtube.com/watch?v=FZR0rG3HKIk
- **Cgroups**
  - https://en.wikipedia.org/wiki/Cgroups
- **Container**
  - https://www.youtube.com/watch?v=EnJ7qX9fkcU
- **Containers versus Containers**
  - https://www.youtube.com/watch?v=L1ie8negCjc