CSCI 381/780
Cloud Computing

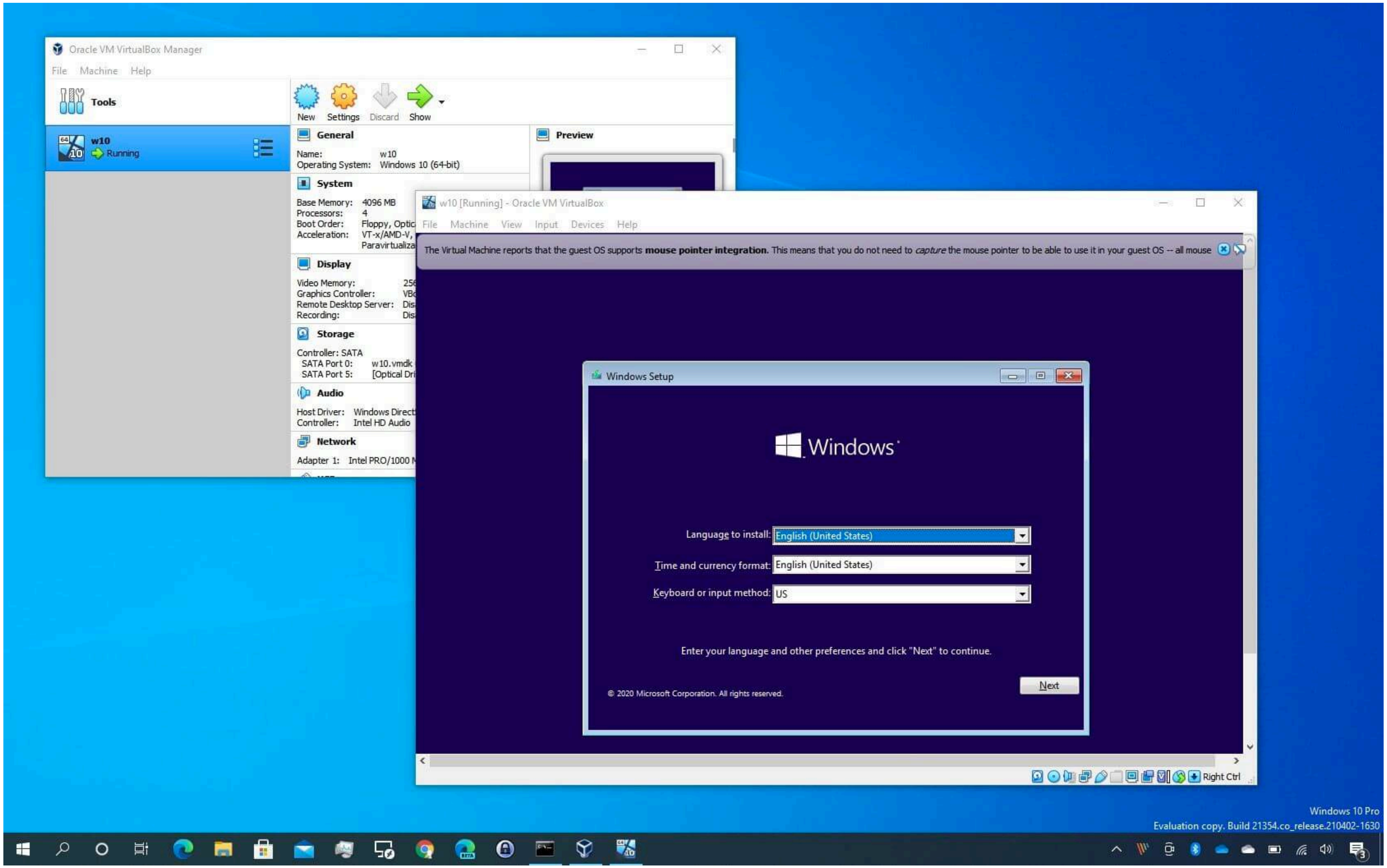
Virtualization 1: Virtual Machine

Jun Li
Queens College

Have you installed a virtual machine on your computer?

What software do you use to host your virtual machine?

VirtualBox


Oracle VM VirtualBox Manager
File Machine Help
Tools
New Settings Discard Show
w10 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows Setup
Windows
Next
Right Ctrl

w10
Running

General
Name: w10
Operating System: Windows 10 (64-bit)

System
Base Memory: 4096 MB
Processors: 4
Boot Order: Floppy, Optic
Acceleration: VT-x/AMD-V,
Paravirtualiza

Display
Video Memory: 25
Graphics Controller: VB
Remote Desktop Server: Dis
Recording: Dis

Storage
Controller: SATA
SATA Port 0: w10.vmdk
SATA Port 5: [Optical Dr

Audio
Host Driver: Windows Direct
Controller: Intel HD Audio

Network
Adapter 1: Intel PRO/1000

Preview

The Virtual Machine reports that the guest OS supports **mouse pointer integration.** This means that you do not need to *capture* the mouse pointer to be able to use it in your guest OS -- all mouse

Language to install: English (United States)
Time and currency format: English (United States)
Keyboard or input method: US

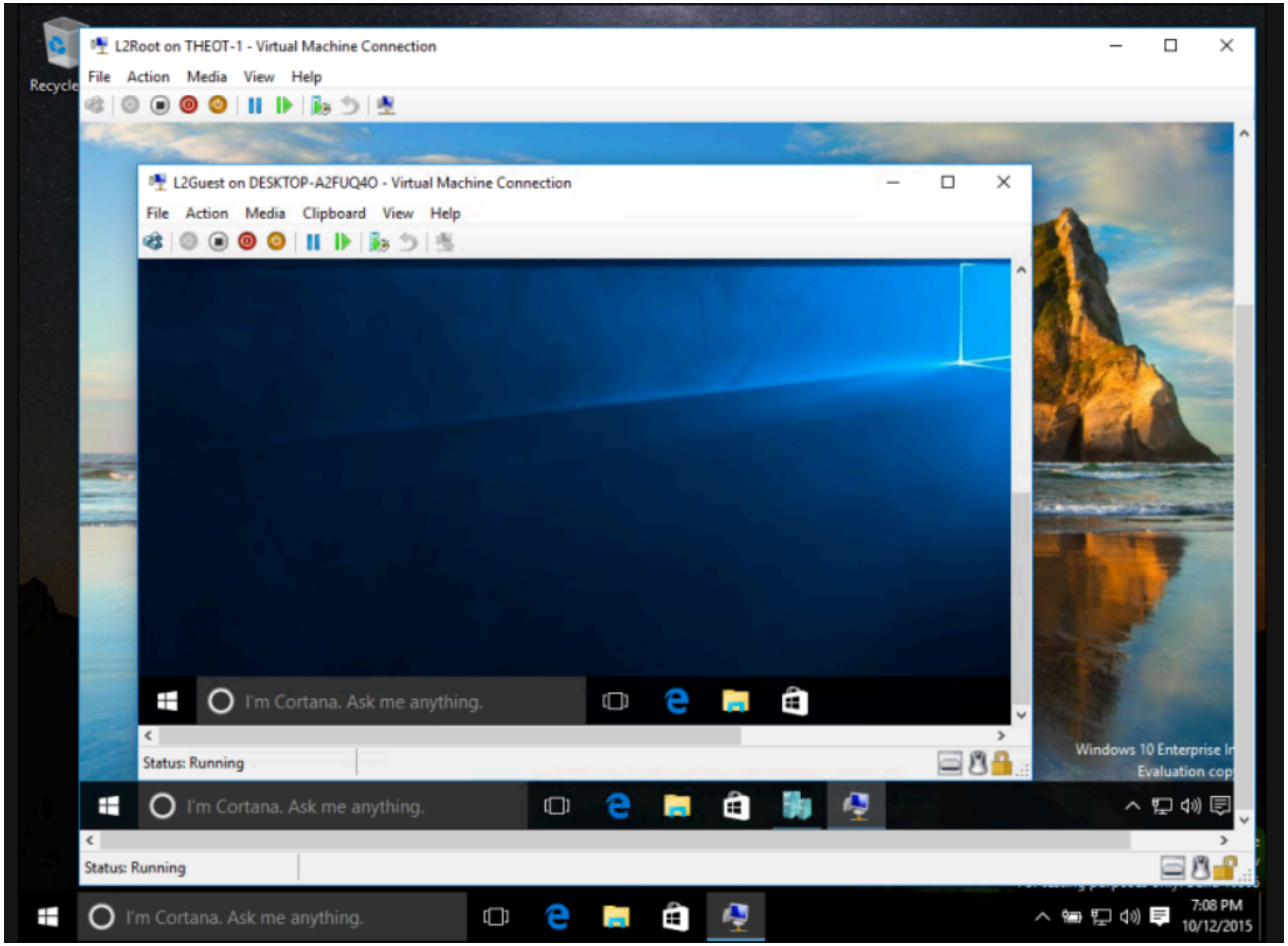
Enter your language and other preferences and click "Next" to continue.

© 2020 Microsoft Corporation. All rights reserved.

Windows 10 Pro
Evaluation copy. Build 21354.co_release.210402-1630

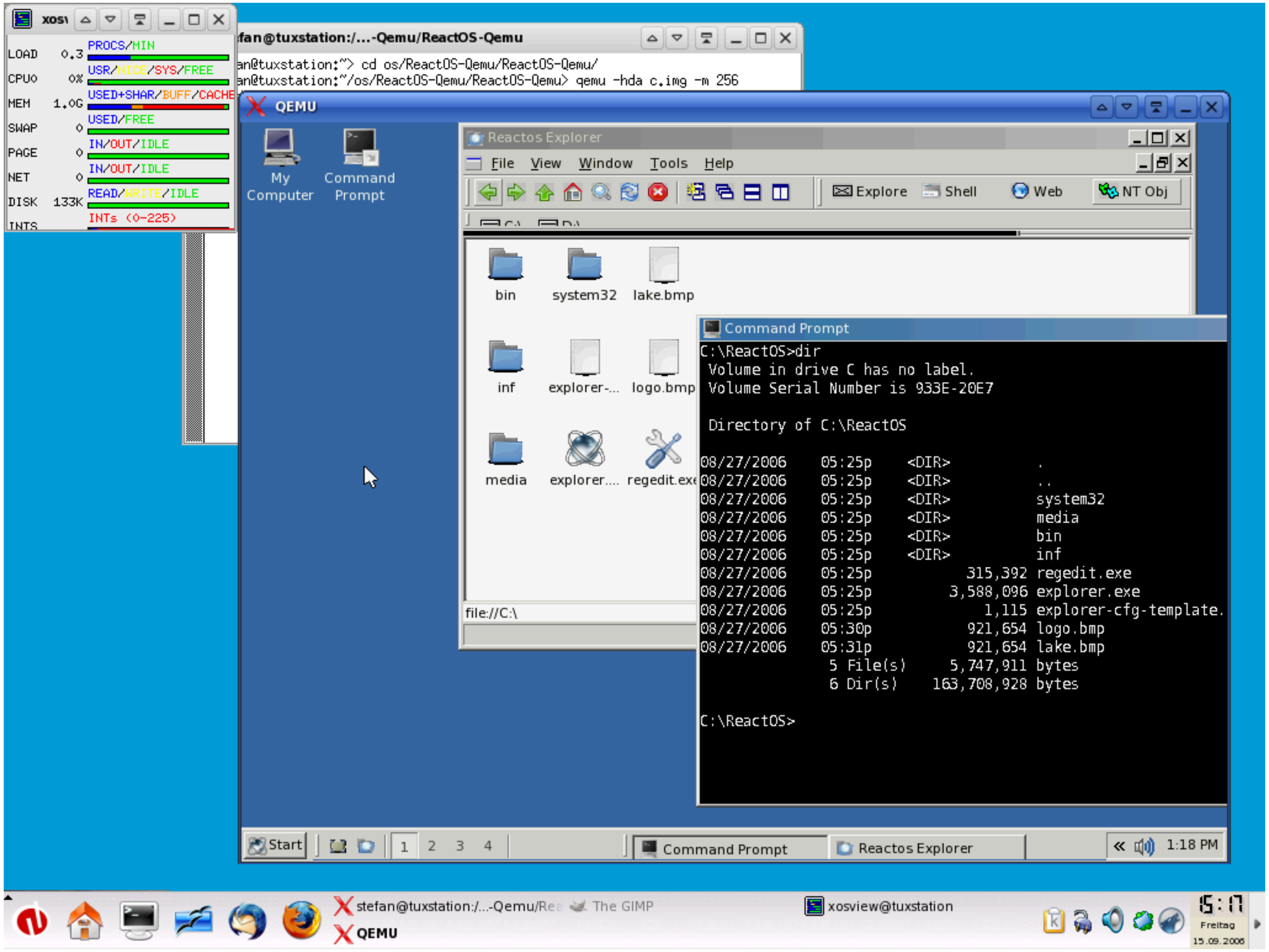
VMware

Hyper-V



Recycle
L2Root on THEOT-1 - Virtual Machine Connection
File Action Media View Help
L2Guest on DESKTOP-A2FUQ4O - Virtual Machine Connection
File Action Media Clipboard View Help
I'm Cortana. Ask me anything.
Status: Running
Windows 10 Enterprise I
Evaluation cop
I'm Cortana. Ask me anything.
Status: Running
I'm Cortana. Ask me anything.
7:08 PM
10/12/2015

QEMU/KVM



```
fan@tuxstation:/...-Qemu/ReactOS-Qemu
an@tuxstation:~> cd os/ReactOS-Qemu/ReactOS-Qemu/
an@tuxstation:~/os/ReactOS-Qemu/ReactOS-Qemu> qemu -hda c.img -m 256
```

```
C:\ReactOS>dir
 Volume in drive C has no label.
 Volume Serial Number is 933E-20E7

 Directory of C:\ReactOS

08/27/2006    05:25p    <DIR>          .
08/27/2006    05:25p    <DIR>          ..
08/27/2006    05:25p    <DIR>          system32
08/27/2006    05:25p    <DIR>          media
08/27/2006    05:25p    <DIR>          bin
08/27/2006    05:25p    <DIR>          inf
08/27/2006    05:25p           315,392 regedit.exe
08/27/2006    05:25p         3,588,096 explorer.exe
08/27/2006    05:25p             1,115 explorer-cfg-template.
08/27/2006    05:30p           921,654 logo.bmp
08/27/2006    05:31p           921,654 lake.bmp
               5 File(s)      5,747,911 bytes
               6 Dir(s)     163,708,928 bytes

C:\ReactOS>
```

Demo

Adoptions

The new EC2 hypervisor provides **consistent performance and increased compute and memory resources** for EC2 virtualized instances by removing host system software components ... This hardware enables the new hypervisor to be very small and uninvolved in data processing tasks for networking and storage. Nov 7, 2017

https://www.theregister.com › 2017/11/07 › aws_writes_n...

AWS adopts home-brewed KVM as new hypervisor

Google Compute Engine uses **KVM** as the hypervisor, and supports guest images running Linux and Microsoft Windows which are used to launch virtual machines based on the 64 bit x86 architecture.

https://en.wikipedia.org › wiki › Google_Compute_Eng...

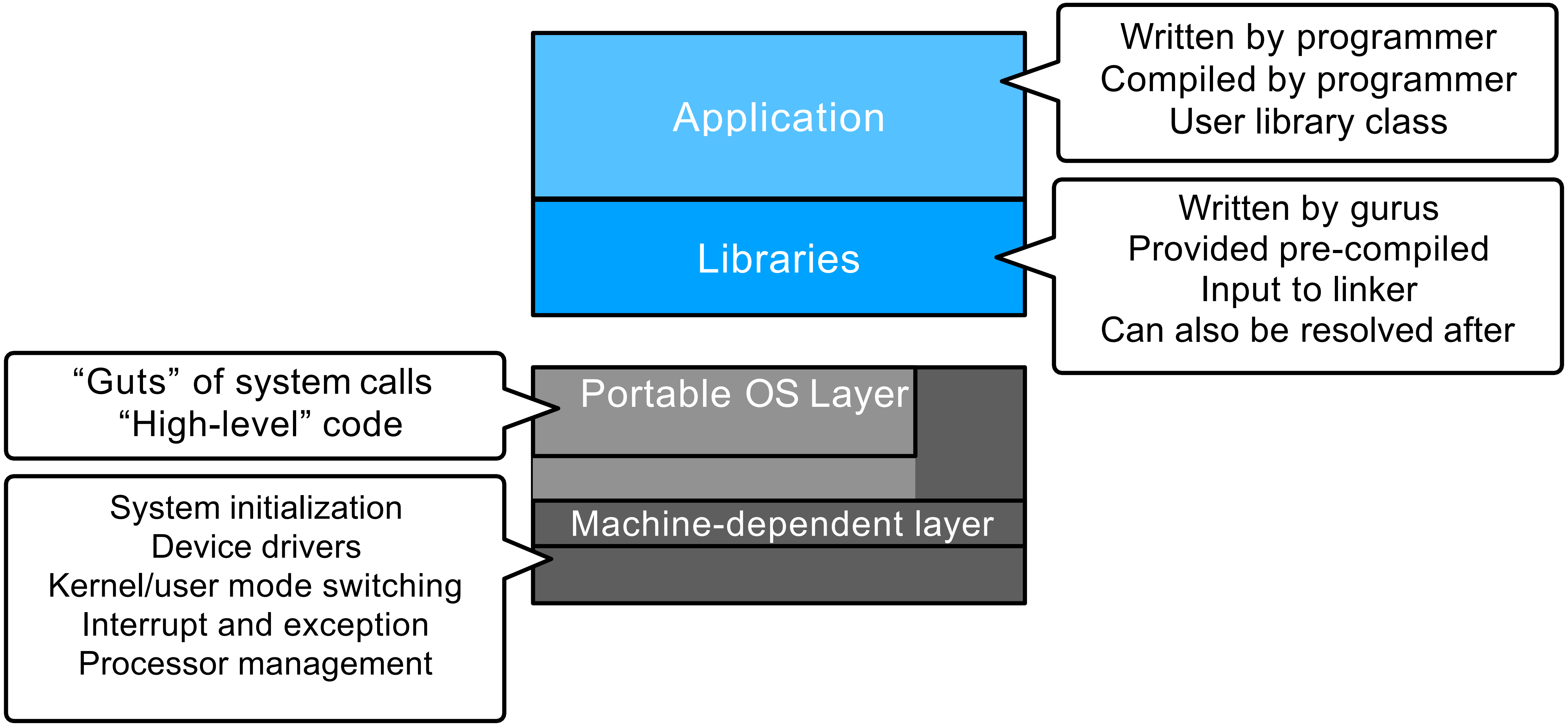
Google Compute Engine - Wikipedia

Azure Hypervisor is referred to the native hypervisor in Azure Cloud Services platform that **enables in creating virtualized machines and servers on Azure** Cloud Platform. It is similar to Microsoft Hyper V, but it is customized specifically for Azure Platform.

https://www.netreo.com › microsoft-azure › what-is-azure...

What is Azure Hypervisor? - Netreo Azure Network Monitoring

Typical OS Structure

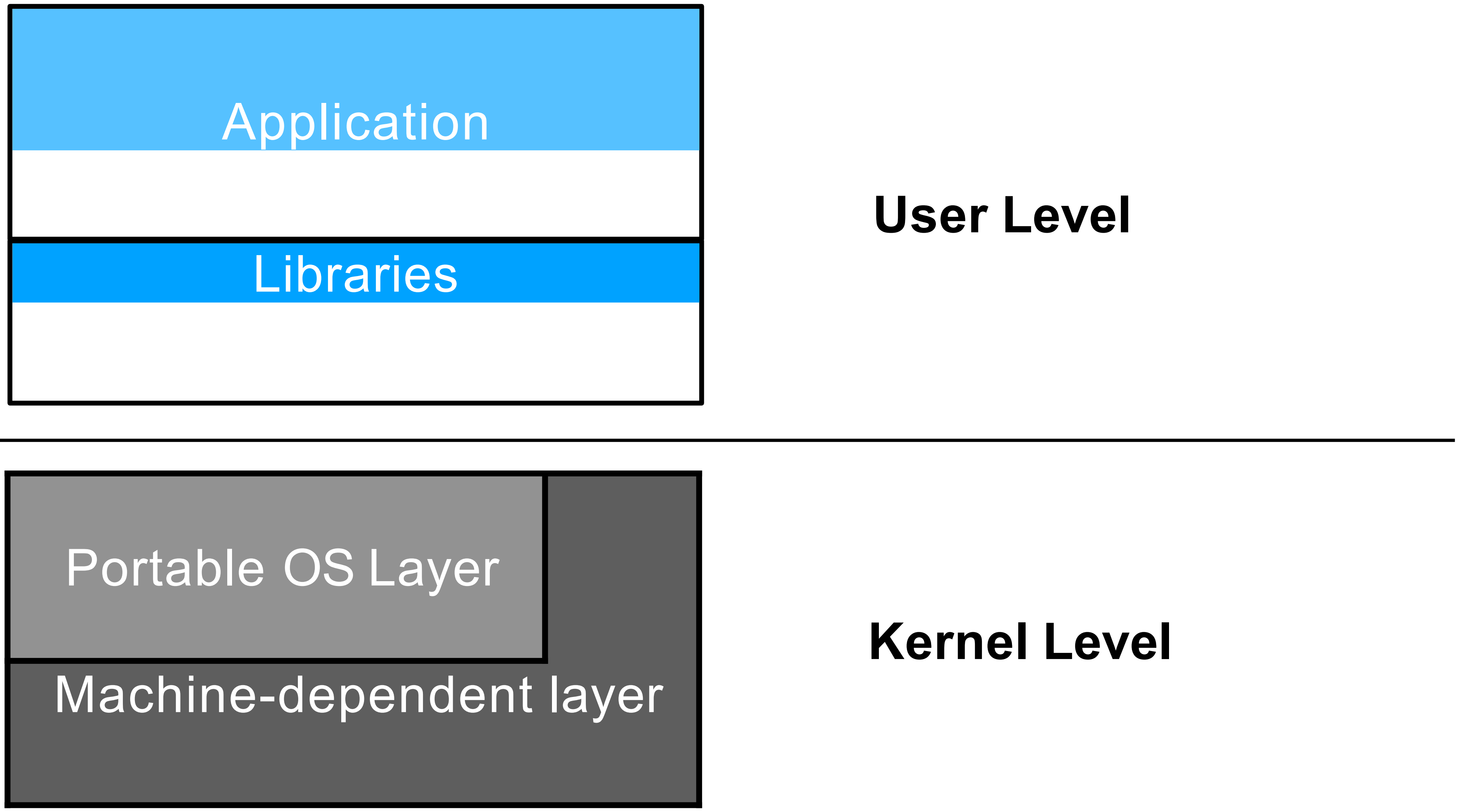


Dual-Mode Operation

- OS manages shared resources
- OS protects programs from other programs

  ➔ OS needs to be “privileged”
- Dual-mode operation of hardware
  - Kernel mode – can run privileged instructions
  - User mode – can only run non-privileged instructions

Typical OS Structure



Interrupt

- A mechanism for coordination between concurrently operating units of a computer system (e.g. CPU and I/O devices) to respond to specific conditions within a computer
- Results in transfer of flow of control (to interrupt handler in the OS), forced by hardware
- Hardware interrupts
  - I/O devices: NIC, keyboard, etc.
  - Timer
- Software interrupts
  - Exception: a software error (e.g., divided by zero)
  - System call

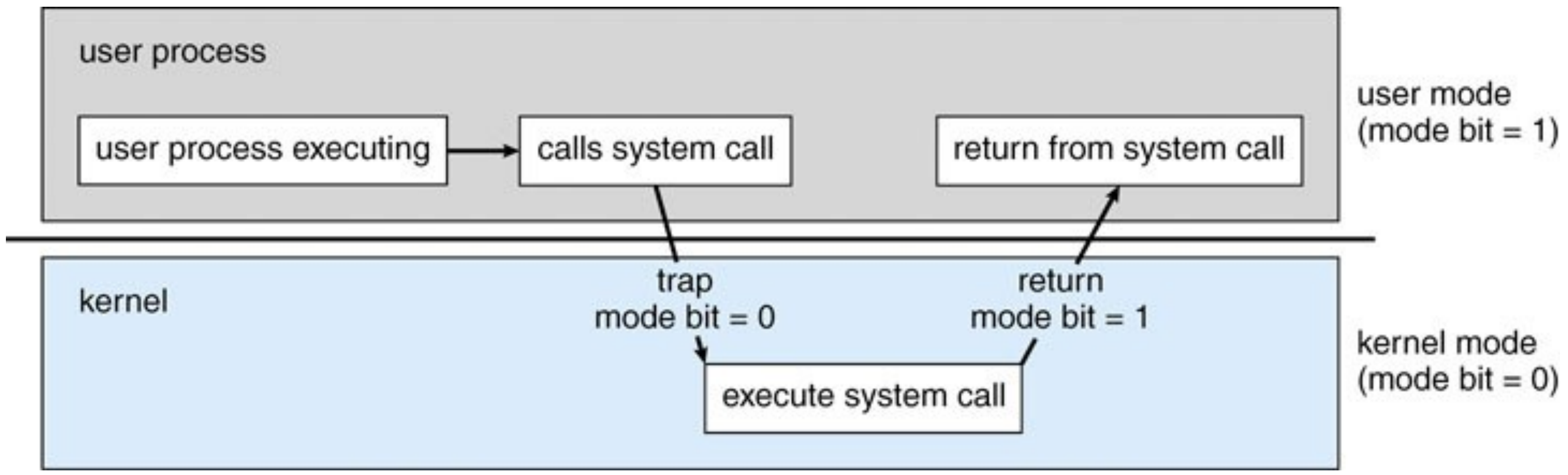
Handling Interrupts (1)

- Incoming interrupts are disabled (at this and lower priority levels) while the interrupt is being processed to prevent a lost interrupt
- Interrupt architecture must save the address of the interrupted instruction
- Interrupt transfers control to the interrupt service routine
  - generally, through the interrupt vector, which contains the addresses of all the service routines

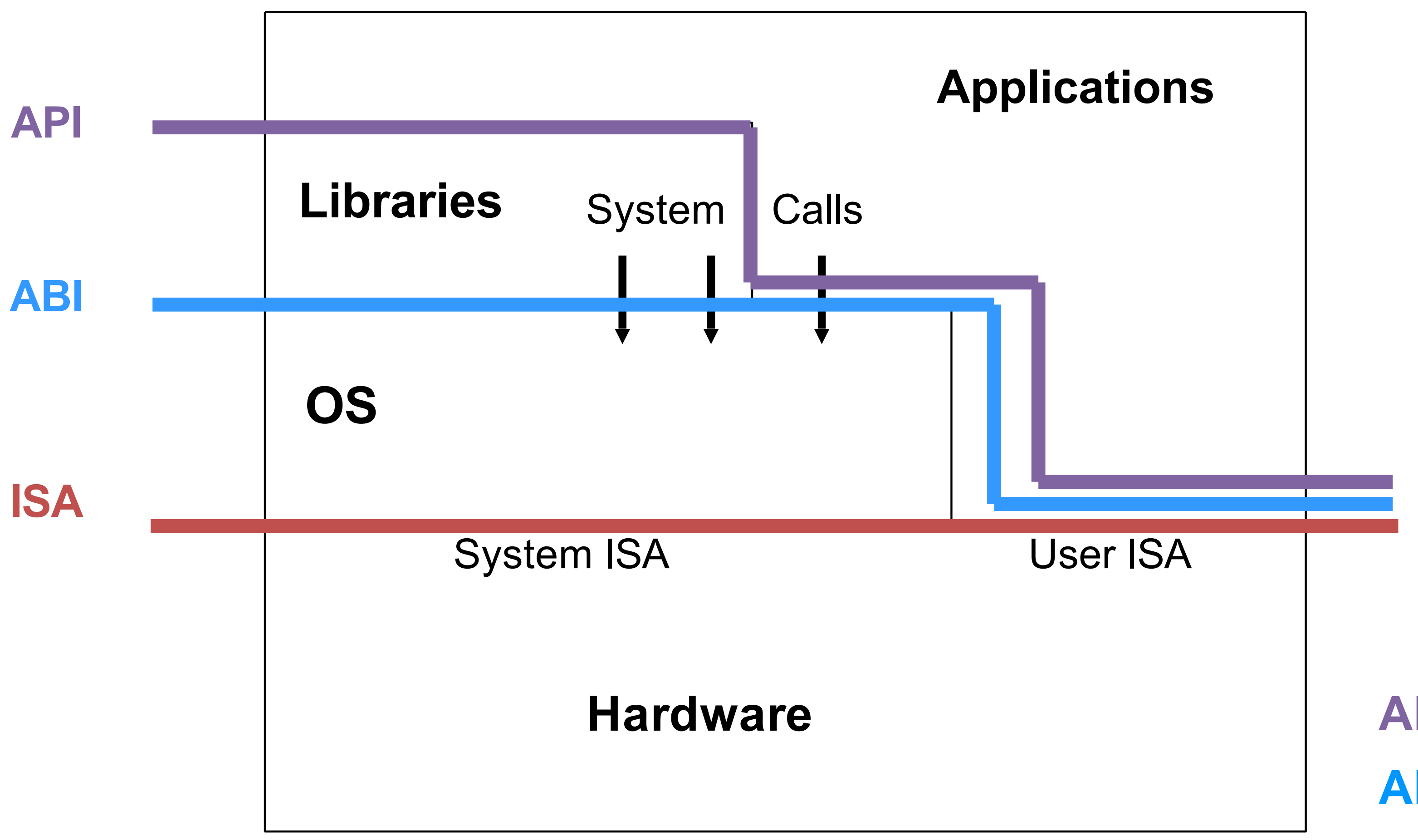
Handling Interrupts (2)

- If interrupt routine modifies process state (register values)
  - save the current state of the CPU (registers and the program counter) on the system stack
  - restore the state before returning
- Interrupts are re-enabled after servicing current interrupt
- Resume the interrupted instruction

Transition between User/Kernel Modes



Interaction between Different Layers



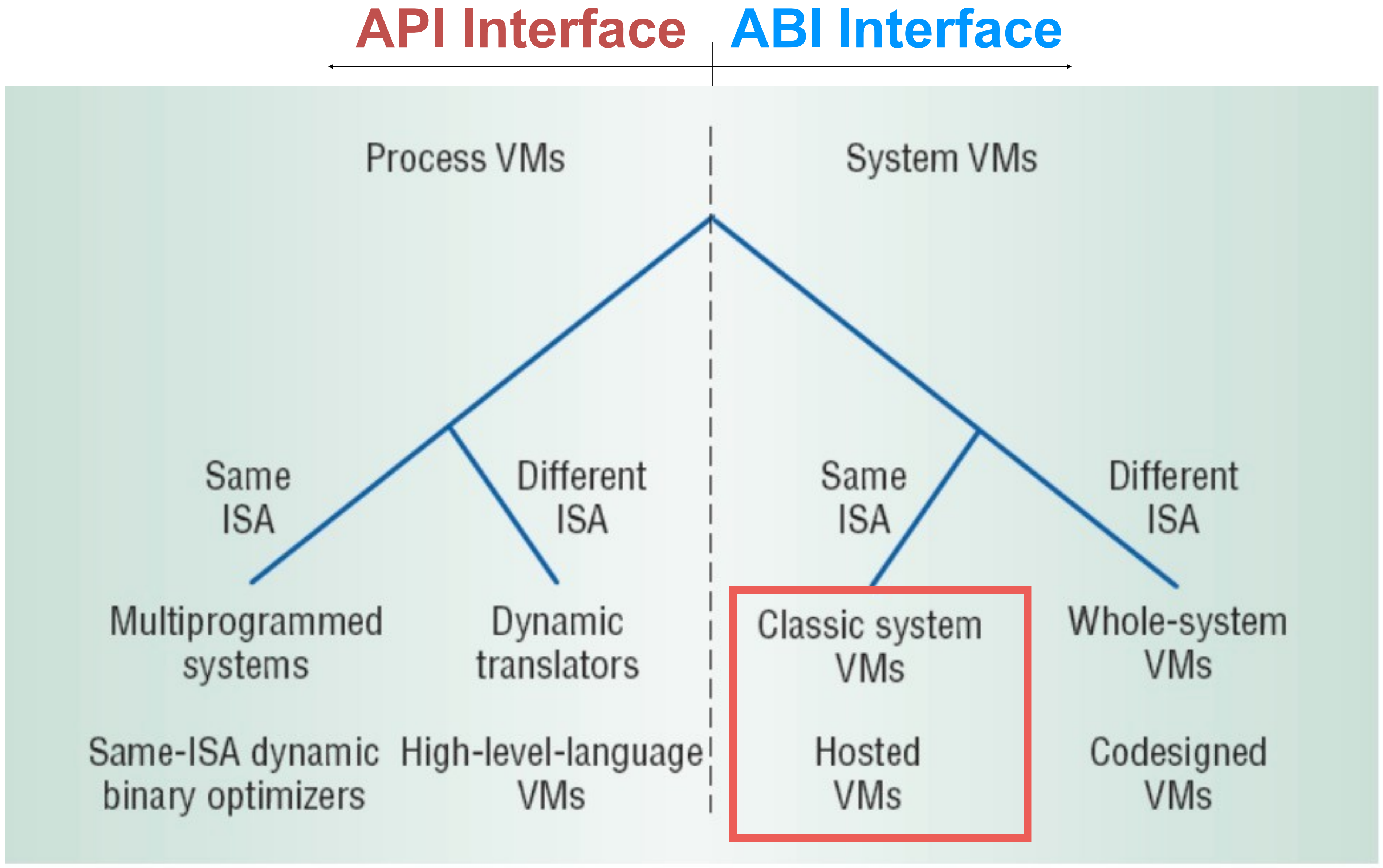
Which layer should virtualization be at?

API – application programming interface

ABI – application binary interface

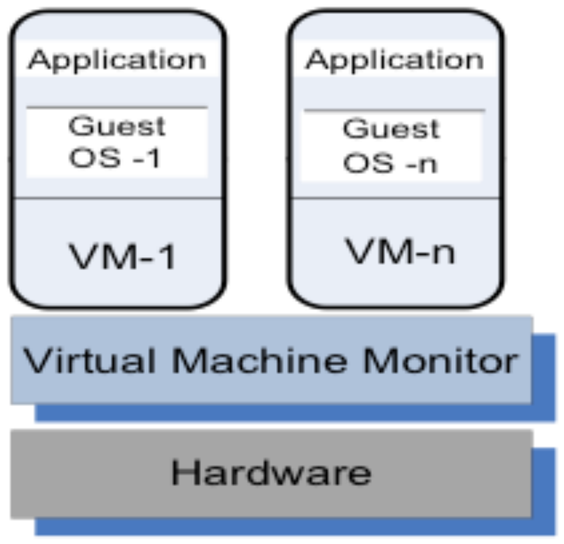
ISA – instruction set architecture

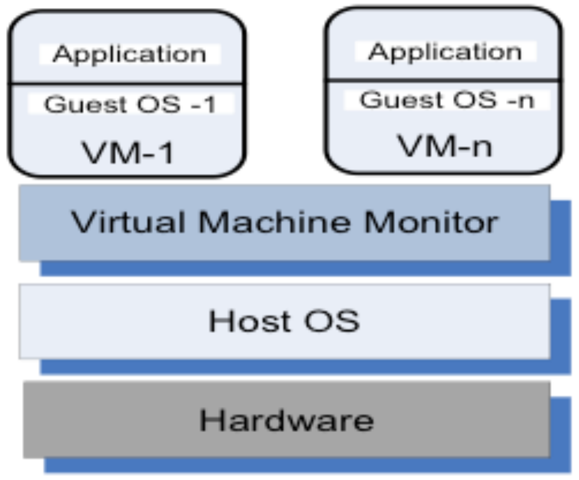
Design Space (Level vs. ISA)



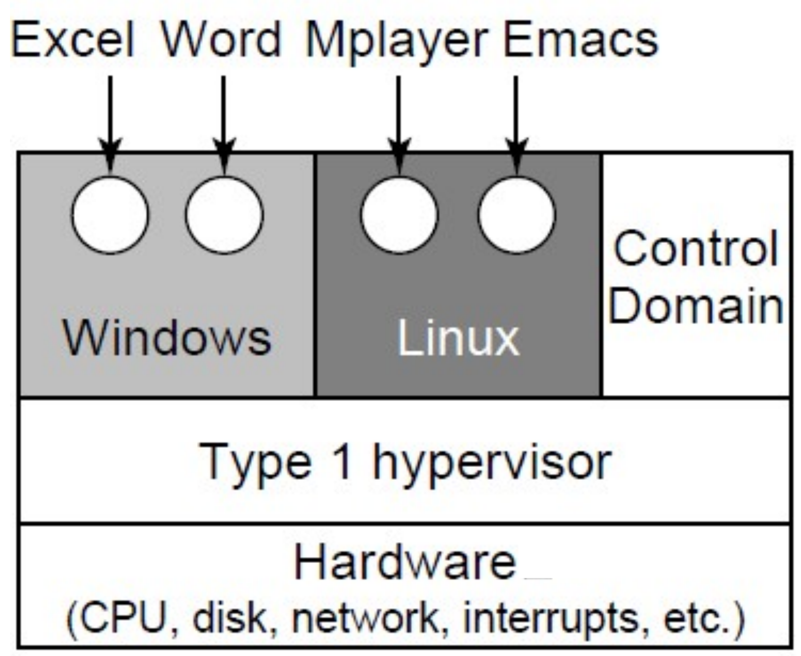
Two types of hypervisors

- Type 1: Native (bare metal)
  - Hypervisor runs on top of the bare metal machine
  - e.g., KVM
- Type 2: Hosted
  - Hypervisor is an emulator
  - e.g.,VMWare, virtualbox, QEMU



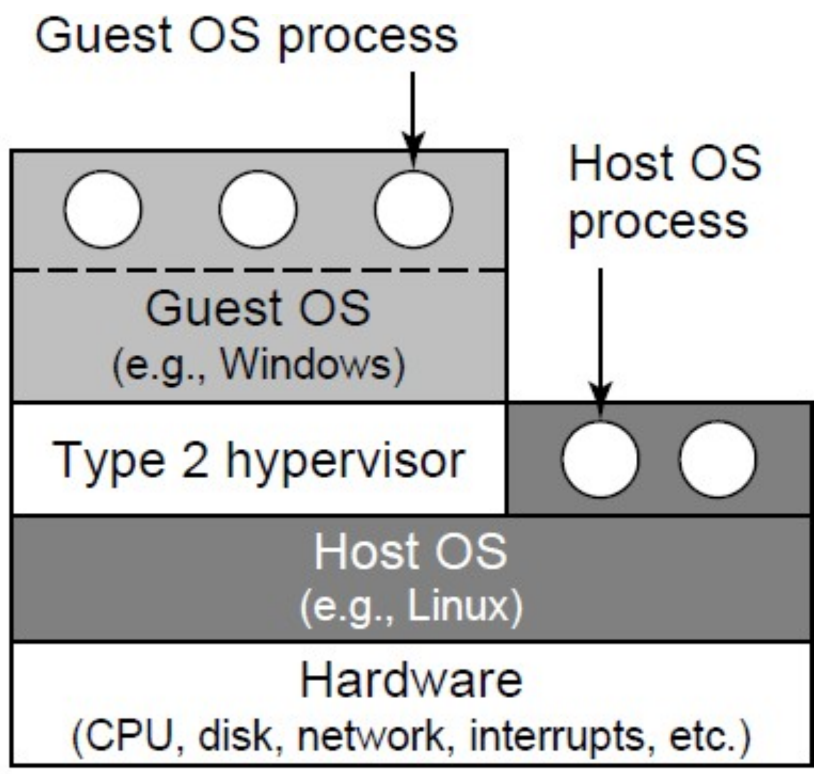


Type 1 and Type 2 Hypervisor (VMM)



(a)

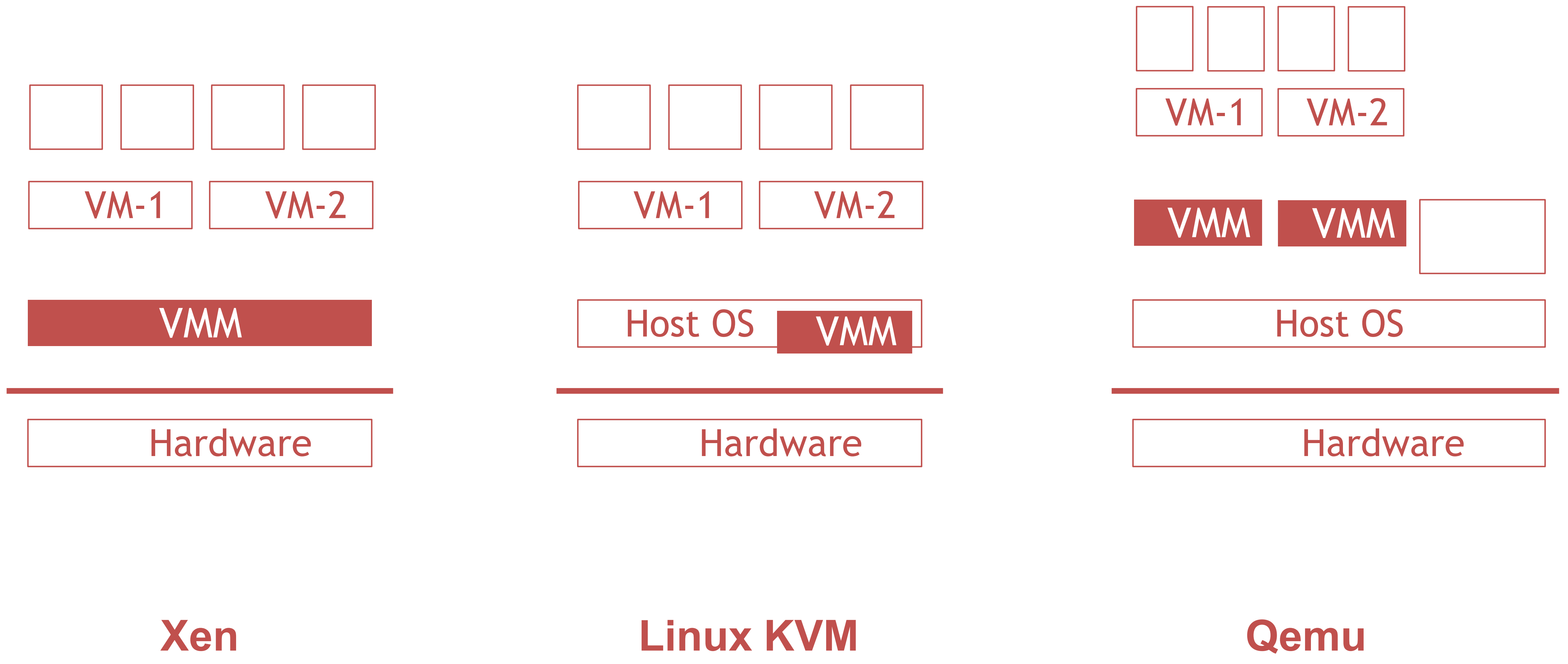
VMware ESX, Microsoft Hyper-V, Xen, KVM



(b)

VMware Workstation, Microsoft Virtual PC, VirtualBox, QEMU

Different Architectures of VMM



Virtualization Principles

- Popek and Goldberg's virtualization principles in 1974:
- **Fidelity**. Software on the VMM executes identically to its execution on hardware, barring timing effects.
- **Performance**. An overwhelming majority of guest instructions are executed by the hardware without the intervention of the VMM.
- **Safety**. The VMM manages all hardware resources.

Possible Implementation: Complete Machine Emulation (Hosted Interpretation)

- VMM implements the complete hardware architecture in software
- VMM steps through VM’s instructions and update emulated hardware as needed

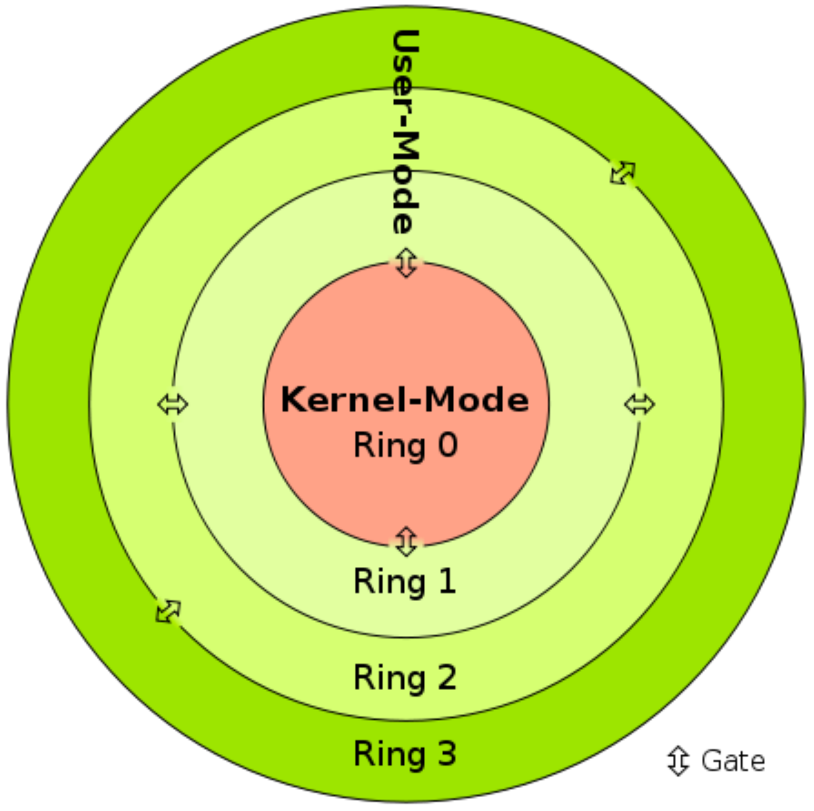
```
while(true) {
  Instruction instr = fetch();

  // emulate behavior in software
  instr.emulate();
}
```

Complete Machine Emulation (Hosted Interpretation)

- Pros
  - Easy to handle all types of instructions (can enforce policy when doing so)
  - Provides complete isolation (no guest instructions runs directly on hardware)
  - Can debug low-level code in the guest
- Cons
  - Emulate a modern processor is difficult
  - Violates performance requirement (it is really slow!)

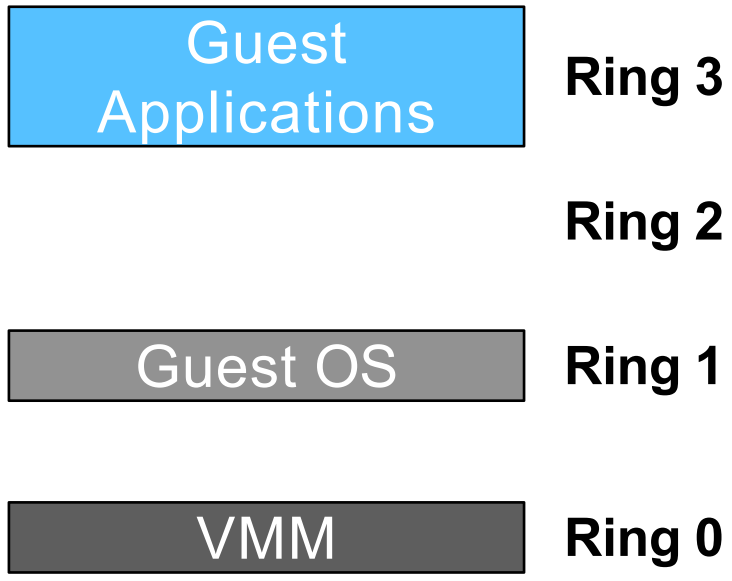
Protection Rings

- More privileged rings can access memory of less privileged ones
- Calling across rings can only happen with hardware enforcement
- Only Ring 0 can execute privileged instructions
- Rings 1, 2, and 3 trap when executing privileged instructions
- Usually, the OS executes in Ring 0 and applications execute in Ring 3



Improving Performance over Full Emulation

- Idea: execute most guest instructions natively on hardware (assuming guest OS runs on the same architecture as real hardware)
- Applications run in ring 3
- Cannot allow guest OS to run sensitive instructions directly!
- Guest OS runs in ring 1
- When guest OS executes a privileged instruction, will trap into VMM
- When guest applications generates a software interrupt, will trap into VMM

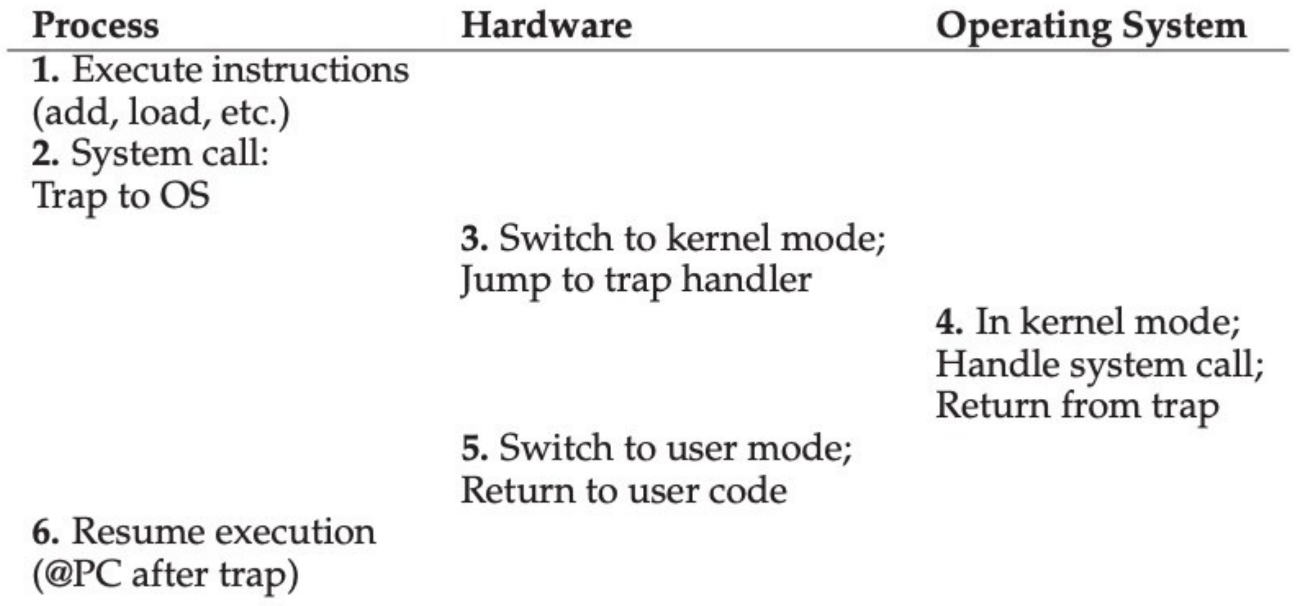


Trap-and-emulate

- Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as trap-and-emulate
  - Most virtualization products use this at least in part

Review: Regular System Call

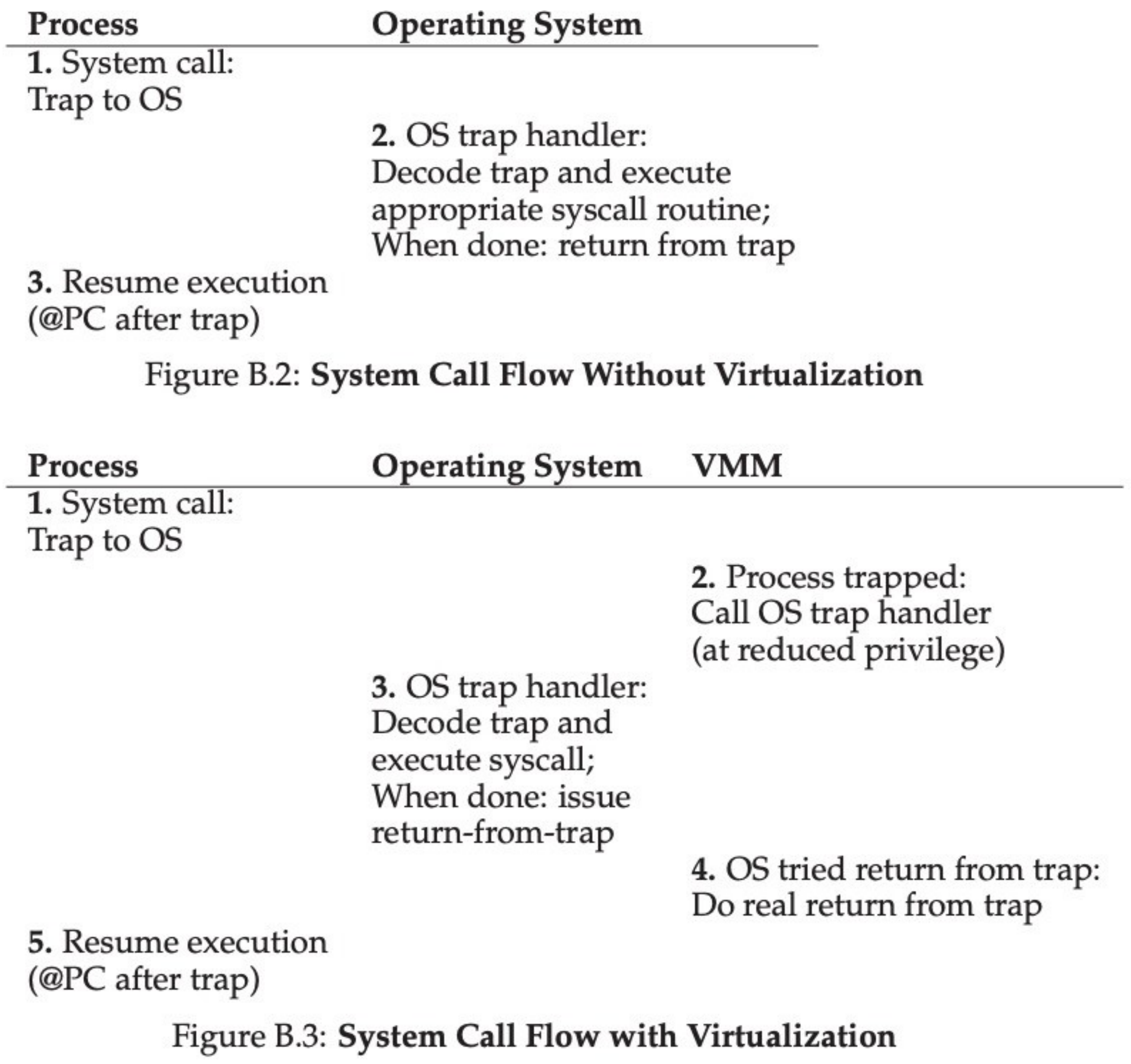
```
open:
    push    dword mode
    push    dword flags
    push    dword path
    mov     eax, 5
    push    eax
    int     80h
```



Process	Hardware	Operating System
1. Execute instructions (add, load, etc.)		
2. System call: Trap to OS		
	3. Switch to kernel mode; Jump to trap handler	
		4. In kernel mode; Handle system call; Return from trap
	5. Switch to user mode; Return to user code	
6. Resume execution (@PC after trap)		

Figure B.1: **Executing a System Call**

System Calls with Virtualization



Process	Operating System
1. System call: Trap to OS	
	2. OS trap handler: Decode trap and execute appropriate syscall routine; When done: return from trap
3. Resume execution (@PC after trap)	

Figure B.2: **System Call Flow Without Virtualization**

Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after trap)		

Figure B.3: **System Call Flow with Virtualization**

Trap-and-emulate

- User mode code in guest runs at same speed as if not a guest
- But kernel mode privilege mode code runs slower due to trap-and-emulate
  - Especially a problem when multiple guests running, each needing trap-and-emulate