

Virtualization

Luigi Libero Lucio Starace

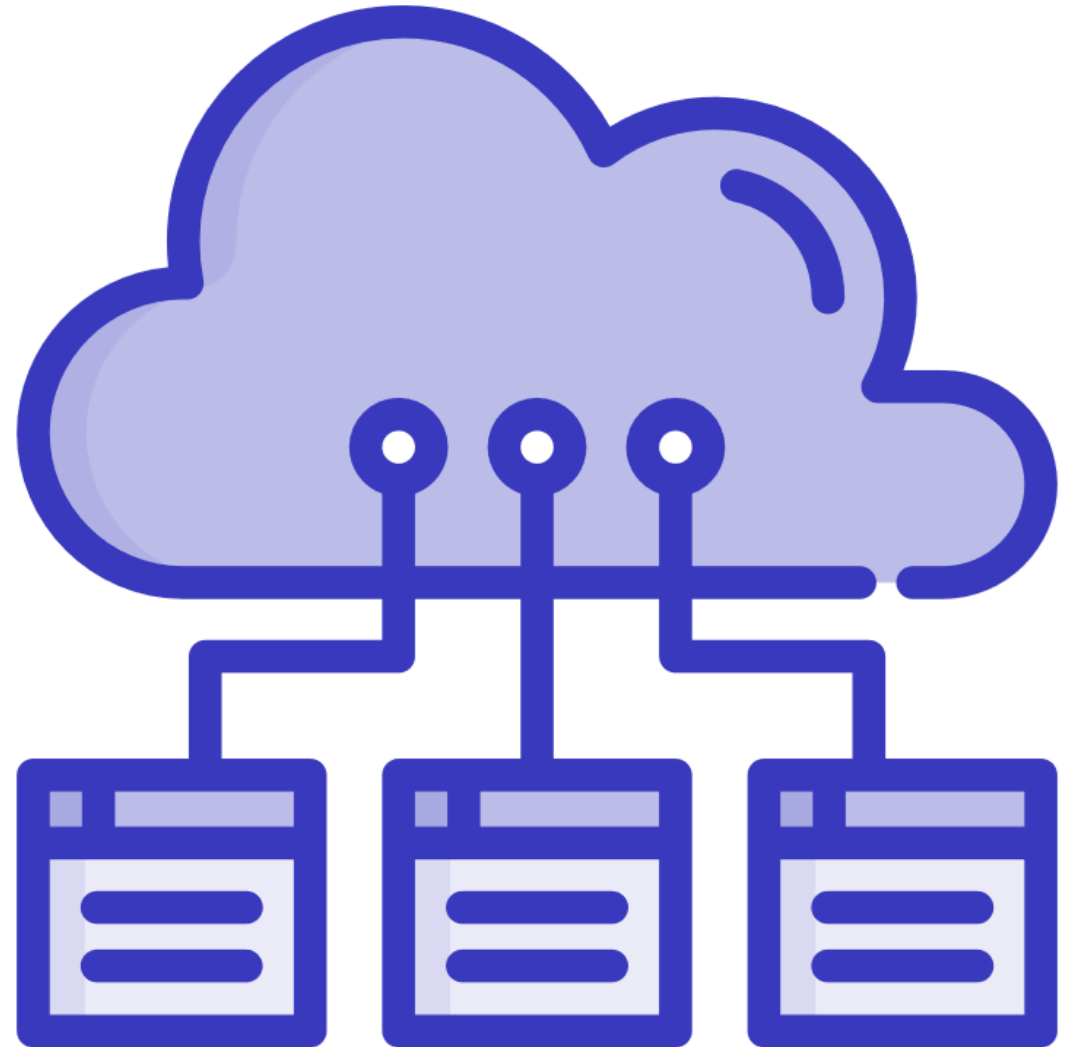
luigiliberolucio.starace@unina.it

21 March, 2022



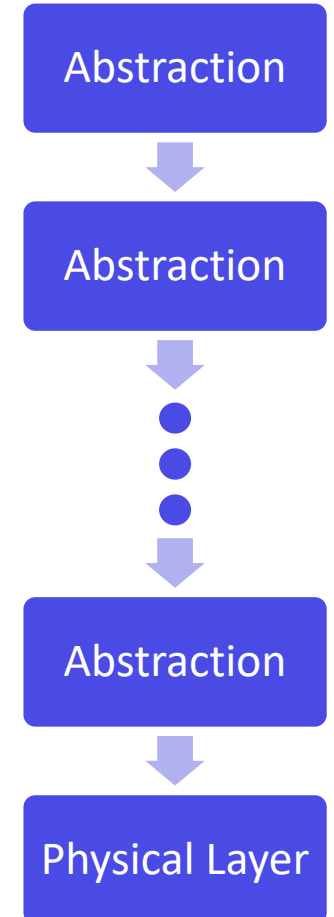
Why?

- **Key enabler for Cloud Computing**
- Virtualization can help us:
 - **Security and Privacy**
 - **Reliability**
 - **Portability**
 - **Productivity**
 - **Performance**
 - and more!



Abstraction

- Computer Systems are **incredibly** complex!
- Nevertheless, they exist, (mostly) work, and continue to evolve...
- How do we manage that kind of complexity?
- **Abstraction** makes it possible
- Computer Systems are designed as **hierarchies** of well-defined interfaces separating different levels of abstraction



Abstraction

- Each abstraction layer provides a **simplified view** of the underlying low-level implementation details
- Well-defined interfaces allow each layer to **evolve independently** from the others



Virtualization Domains

- Storage Virtualization (SAN)
- Network Virtualization (VLAN, VPN)
- **Execution Environment Virtualization**

Interfaces in Computing Systems

App. Programming Interface (API)

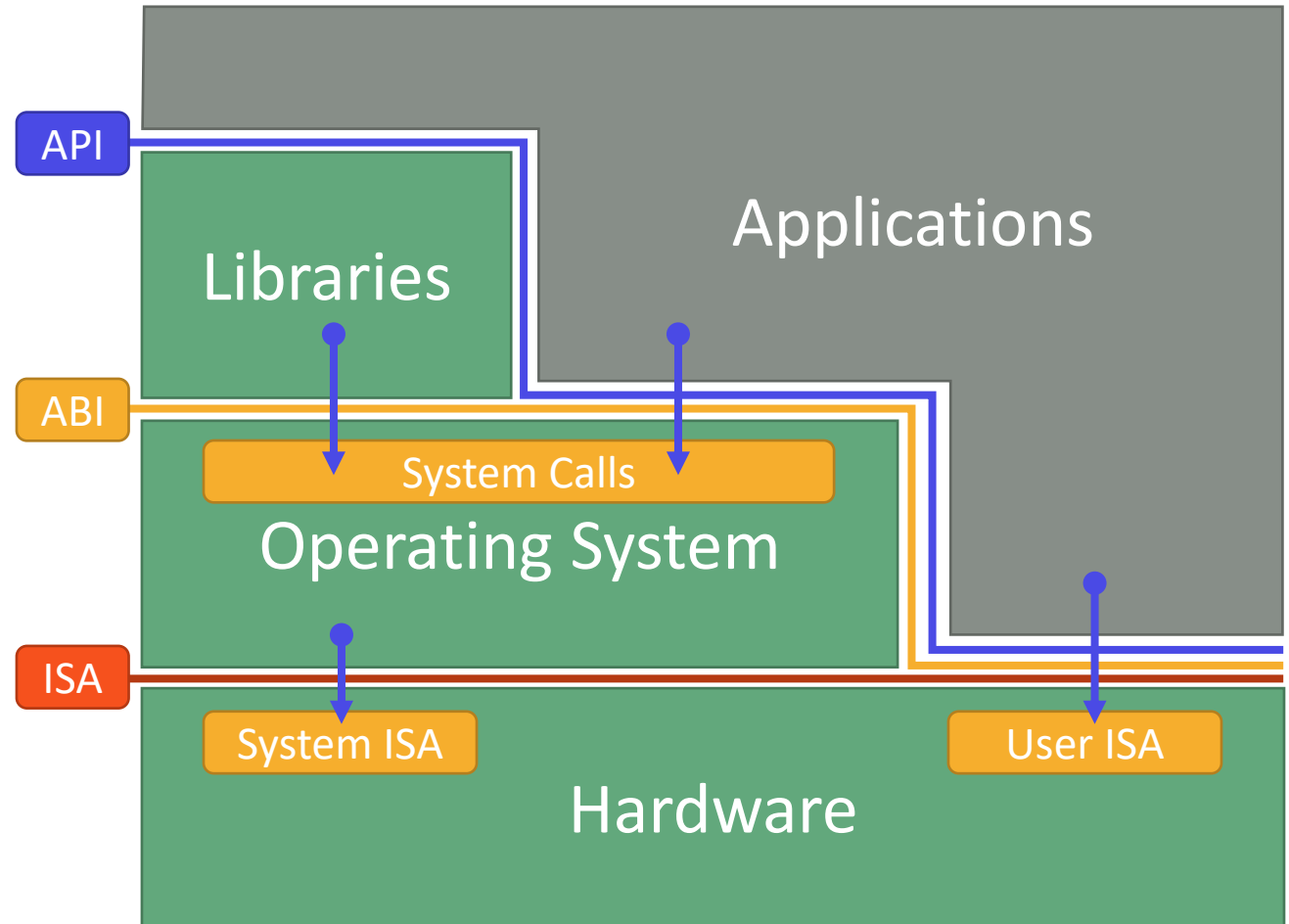
Source code level interface for devs.

Application Binary Interface (ABI)

Interface between OS and Apps, at machine code level

Instruction Set Architecture (ISA)

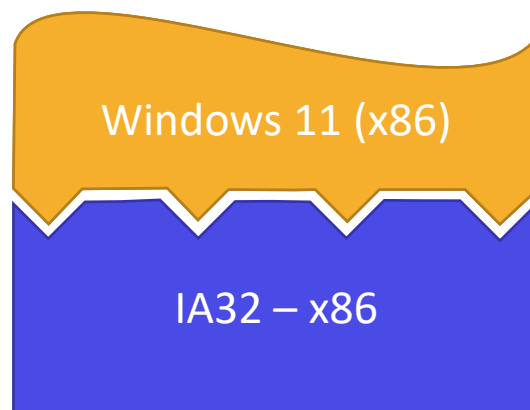
Boundary between HW and SW



Virtualization

Well-defined interfaces pose some limitations as well:

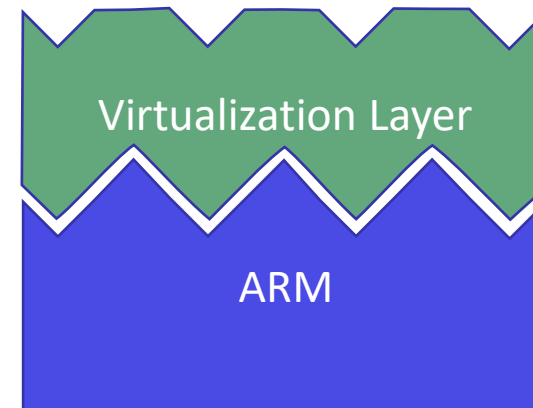
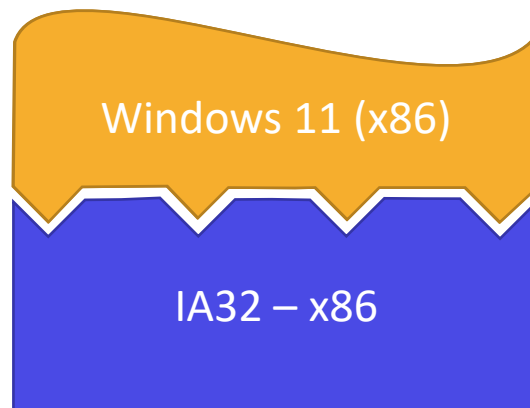
- Subsystems designed to standard with different interfaces might be incompatible
- Can't run an x86 OS on a ARM platform!



Virtualization: motivations

Virtualization helps getting around these limitations (and much more)

- **Goal: mimic** the behaviour of these interfaces



Components of a Virtualized Environment

Host

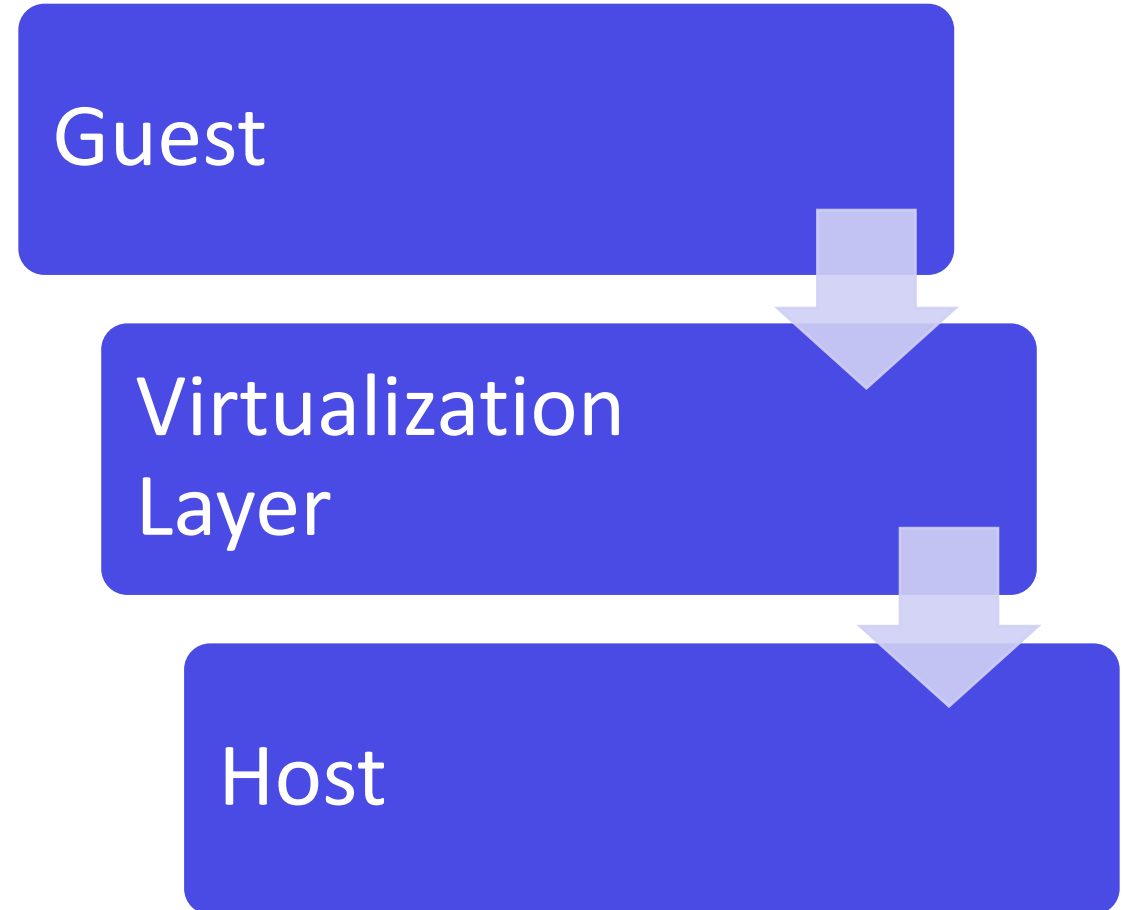
- Existing component exposing a specific interface

Virtualization Layer

- Responsible for mapping a different interface to the one exposed by the host

Guest

- Component that interacts with the virtual interface provided by the Virtualization Layer.



Execution Environment Virtualization

Virtual Machines (VMs)

- A Virtual Machine (VM) is an **isolated execution environment**
- But what's an «execution environment»?
- For a **Process** running a user program, the execution environment is:
 - logical address space; user instructions and registers;
 - I/O only accessible through the OS and its ABI.
- For an **Operating System**, the execution environment is a HW system:
 - The underlying ISA, real memory, persistent storage, and I/O resources;
 - The OS manages a number of processes, and its environment persists over time as processes come and go

Process and System VMs

As there are different perspective of what an execution environment is, there are different approaches to Virtual Machines as well:

- **Process-level VMs**

- Virtual platforms that can run a **single** process;
- **Ephmeral**: created when the process starts and destroyed on completion.

- **System-level VMs**

- Provide a **complete and persistent** virtual hardware environment
- Support an Operating System and its many processes

Execution Environment Virtualization

Execution Environment Virtualization

Process Level VMs

System Level VMs

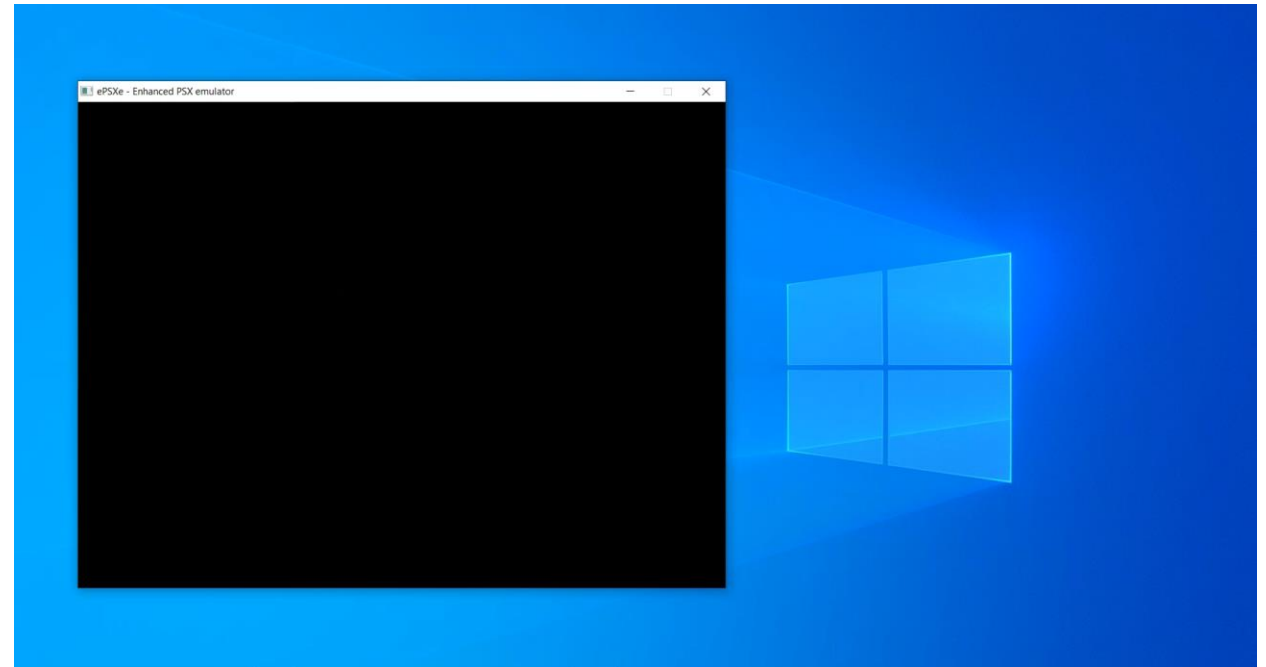
Process-level VMs: Multi-programmed OS

- A traditional multiprogrammed OS is itself a virtualization layer
- It provides abstractions of underlying physical resources
- It implements process-level VMs

Physical Resource	Operating System Abstraction
CPU	Process/Threads
Memory	Virtual Memory
Hard Disks	File System/File/Directories
Network Interfaces	Sockets

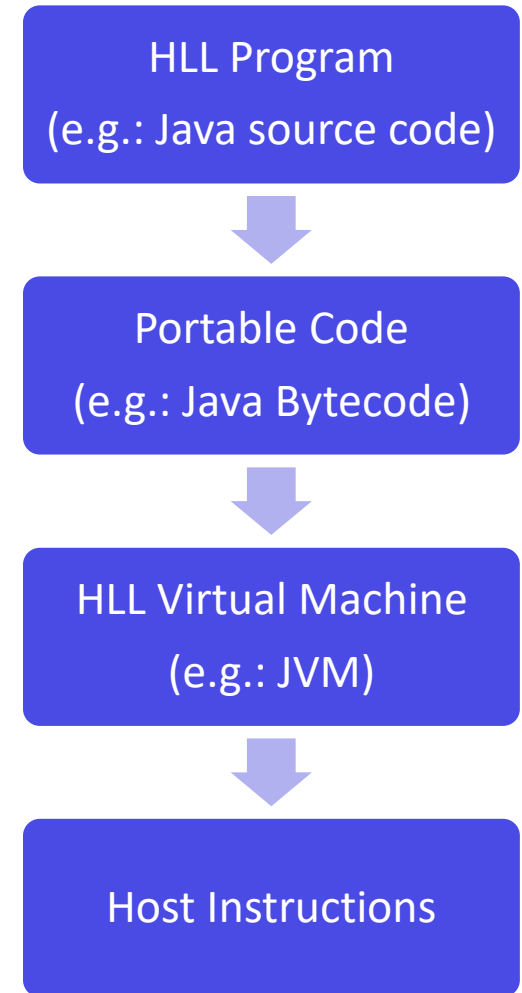
Process-level VMs: ISA-level virtualization

- Can emulate different ISAs (e.g.: MIPS on x86)
- Can be done through:
 - **Code interpretation:** slower, each instruction is translated
 - **Dynamic Binary Translation:** faster, entire blocks are converted and cached
- Same-ISA dynamic binary optimizers



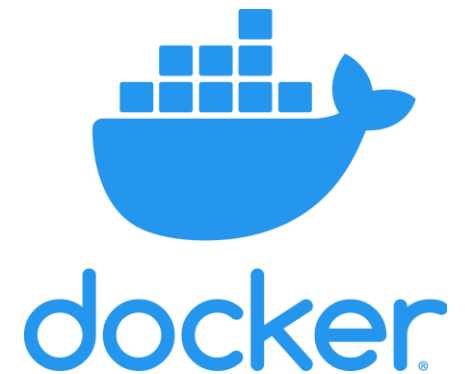
Process-level VMs: High-Level Language VMs

- **Cross-platform portability** is a key objective for Process VMs.
- Virtualizing an architecture on another needs **case-by-case** work.
- Portability is more easily achieved by designing a process level VM (not corresponding to any real platform) as part of an overall High-Level Language (HLL) execution environment.
- Anything comes to mind?



Process-level VMs: OS-level virtualization

- Processes in a multiprogrammed OS run in isolation:
 - Each has its own virtual memory space, privilege level, etc...
- OS-level virtualization (i.e.: *Containers*) leverages certain host OS capabilities to guarantee **higher levels** of isolation between processes
 - Containers are basically processes (or groups of processes), managed by a shared kernel, each running in an isolated environment with tight resource access control.
 - Examples: chroot, FreeBSD Jails, Docker, ...
- We will focus on *Containers* in a future lecture!



Execution Environment Virtualization

Execution Environment Virtualization

Process Level VMs

System Level VMs

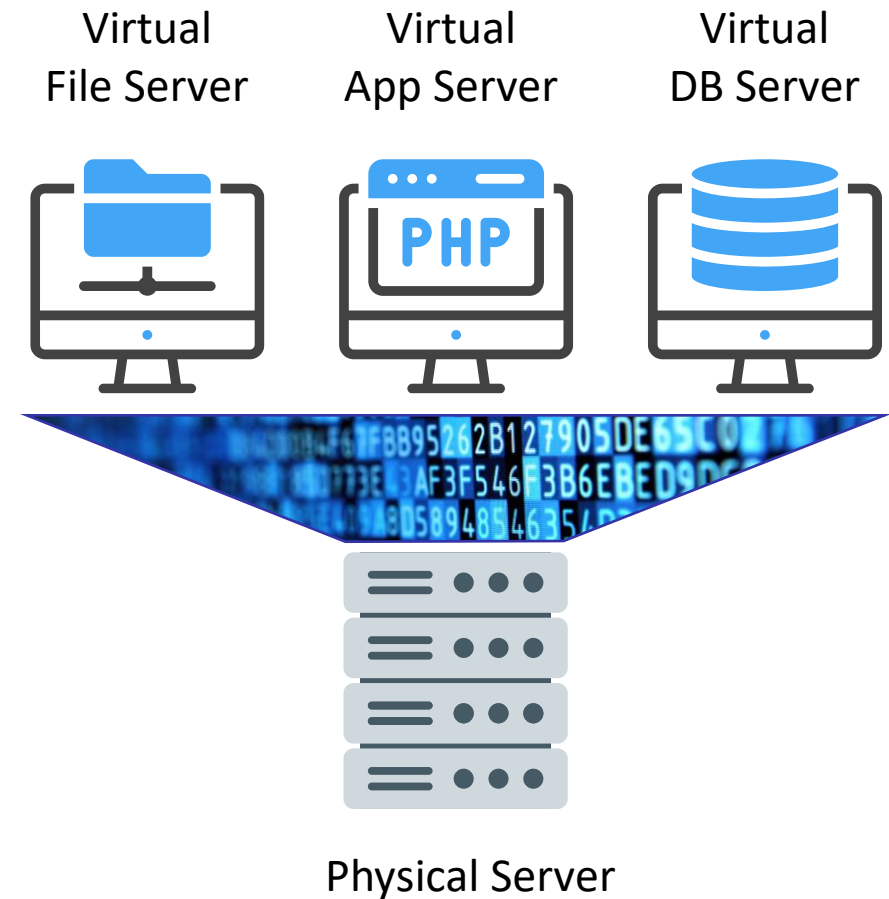
Multi-
programming

ISA-level /
High-Level
Language VM

OS-level

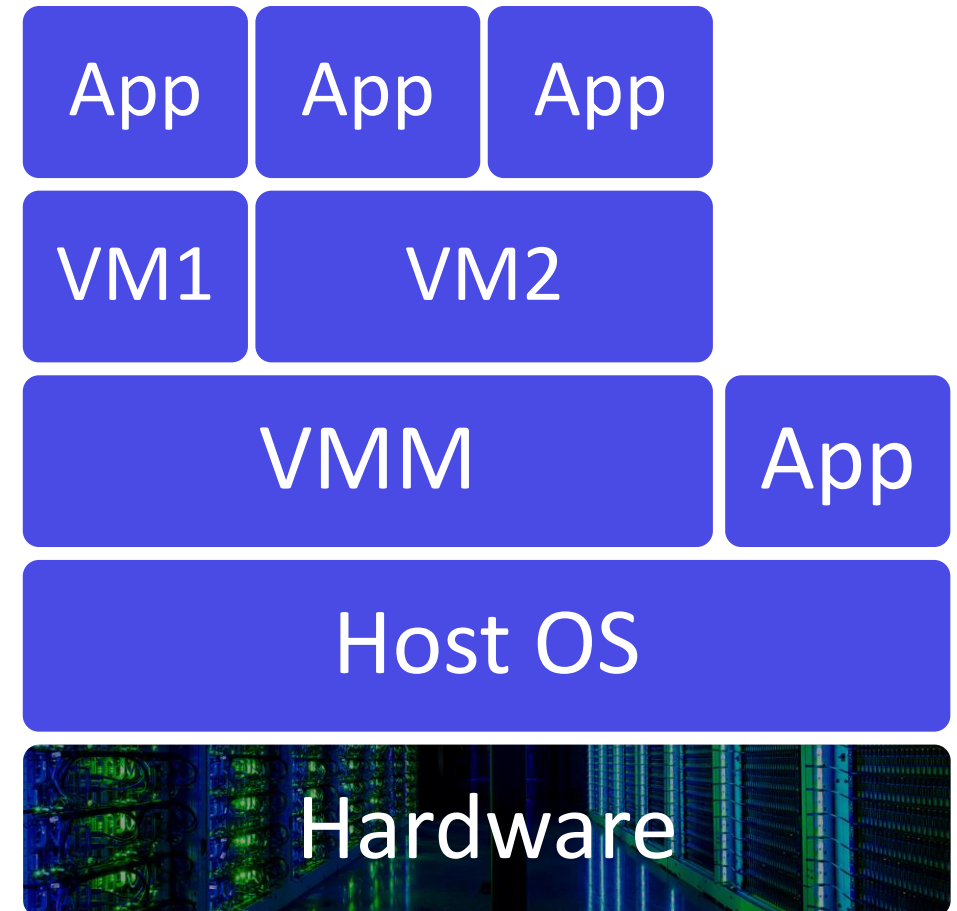
System-level Virtual Machines

- Provide a complete virtual hardware platform
- Can run their own OS, and multiple user processes
- Virtualization layer is called **Virtual Machine Monitor (VMM)** or **Hypervisor**
- Three main approaches for VMMs: **Hosted, Bare Metal, Hybrid**



Hosted VMM

- Runs on top of the Host OS
- Also called «Type 2» VMM
- VMM is simpler:
 - leverage host OS facilities for scheduling, memory management, I/O.
- Greater overhead, less attractive for enterprise solutions
- Examples: VirtualBox, VMWare Player.



ubuntu [In esecuzione] - Oracle VM VirtualBox

File Macchina Visualizza Inserimento Dispositivi Aiuto

Activities Terminal gen 18 14:29

Home

luigi@ubuntu-VirtualBox: ~

```
./+oossssoo+/-.  
:~+ssssssssssssssssss+~  
-+ssssssssssssssssssyyssss+  
.oosssssssssssssssssdMMMMNyssso.  
/sssssssssssshdmmNNmyNMMMMhssssss/  
+ssssssssshmydMMMMMMNdddysssssss+  
/ssssssshNMMMyhyyyyhNMMMNhssssssss/  
.sssssssdMMMNhssssssssshNMMMdssssssss.  
+ssshhhyNMMNyssssssssssssyNMMMyssssss+  
ossyNMMNyMMhssssssssssshmmhssssssso  
ossyNMMNyMMhssssssssssshmmhssssssso  
+ssshhhyNMMNyssssssssssssyNMMMyssssss+  
.sssssssdMMMNhssssssssshNMMMdssssssss.  
/ssssssshNMMMyhyyyyhNMMMNhssssssss/  
+sssssssdmydMMMMMMNdddysssssss+  
/sssssssshdmmNNmyNMMMMhssssss/  
.oosssssssssssssssssdMMMMNyssso.  
-+ssssssssssssssssssyyssss+  
:~+ssssssssssssssssss+~  
./+oossssoo+/-.
```

luigi@ubuntu-VirtualBox

OS: Ubuntu 21.10 x86_64
Host: VirtualBox 1.2
Kernel: 5.13.0-25-generic
Uptime: 18 mins
Packages: 1657 (dpkg), 8 (snap)
Shell: bash 5.1.8
Resolution: 1024x768
DE: GNOME 40.5
WM: Mutter
WM Theme: Adwaita
Theme: Yaru [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i7-7700HQ (3) @ 2.808GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 621MiB / 3913MiB

luigi@ubuntu-VirtualBox:~\$

Oracle VM VirtualBox Gestore

File Macchina Aiuto

Strumenti Nuova Impostazioni Scarta Mostra

64 ubuntu-14.04 Spenta

64 ubuntu In esecuzione

Generale
Nome: ubuntu
Sistema operativo: Ubuntu (64-bit)

Sistema
Memoria di base: 4096 MB
Processori: 3
Ordine di avvio: Floppy, Ottico, Disco fisso
EFI: Abilitato
Accelerazione: VT-x/AMD-V, Paginazione nidificata, PAE/NX, Paravirtualizzazione KVM

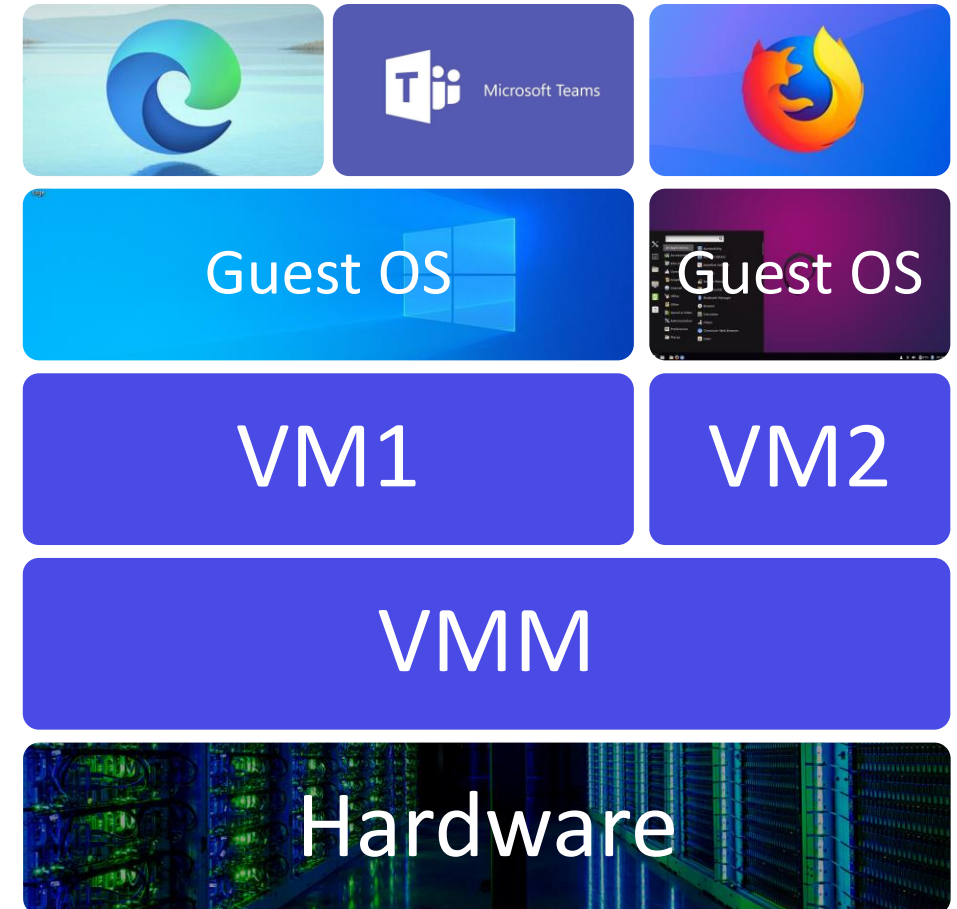
Schermo
Memoria video: 16 MB
Scheda grafica: VMSVGA

Anteprima

Hosted VMM in a Windows 10 Host

Bare Metal VMM

- Runs directly on the HW
- Also called «Type 1» or native VMM
- More complex than hosted ones
- Better performance
- Used in most enterprise solutions
- Examples: VMWare ESX/ESXi, Microsoft Hyper-V



Console di gestione di Hyper-V

File Azione Visualizza ?

Console di gestione di Hyper-V
CORSAIRSERVER


Macchine virtuali

Nome	Stato	Utilizzo CPU	Memoria assegnata	Tempo di attività	Condizione	Versione configurazione
DockerDesktopVM	Salvato					9.0
ICSE20-NDD	Spento					9.0
Ubuntu-20.04LTS-OSRM	In esecuzione	0%	8192 MB	2.04:37:15		9.0

Punti di controllo

Per la macchina virtuale selezionata non sono disponibili punti di controllo.

Ubuntu-20.04LTS-OSRM

	Data creazione: 27/07/2021 17:26:40	In cluster: No
	Versione configurazione: 9.0	Heartbeat: OK (dati applicazioni non disponibili)
	Generazione: 1	
	Note: Nessuna	

Riepilogo Memoria Rete Replica

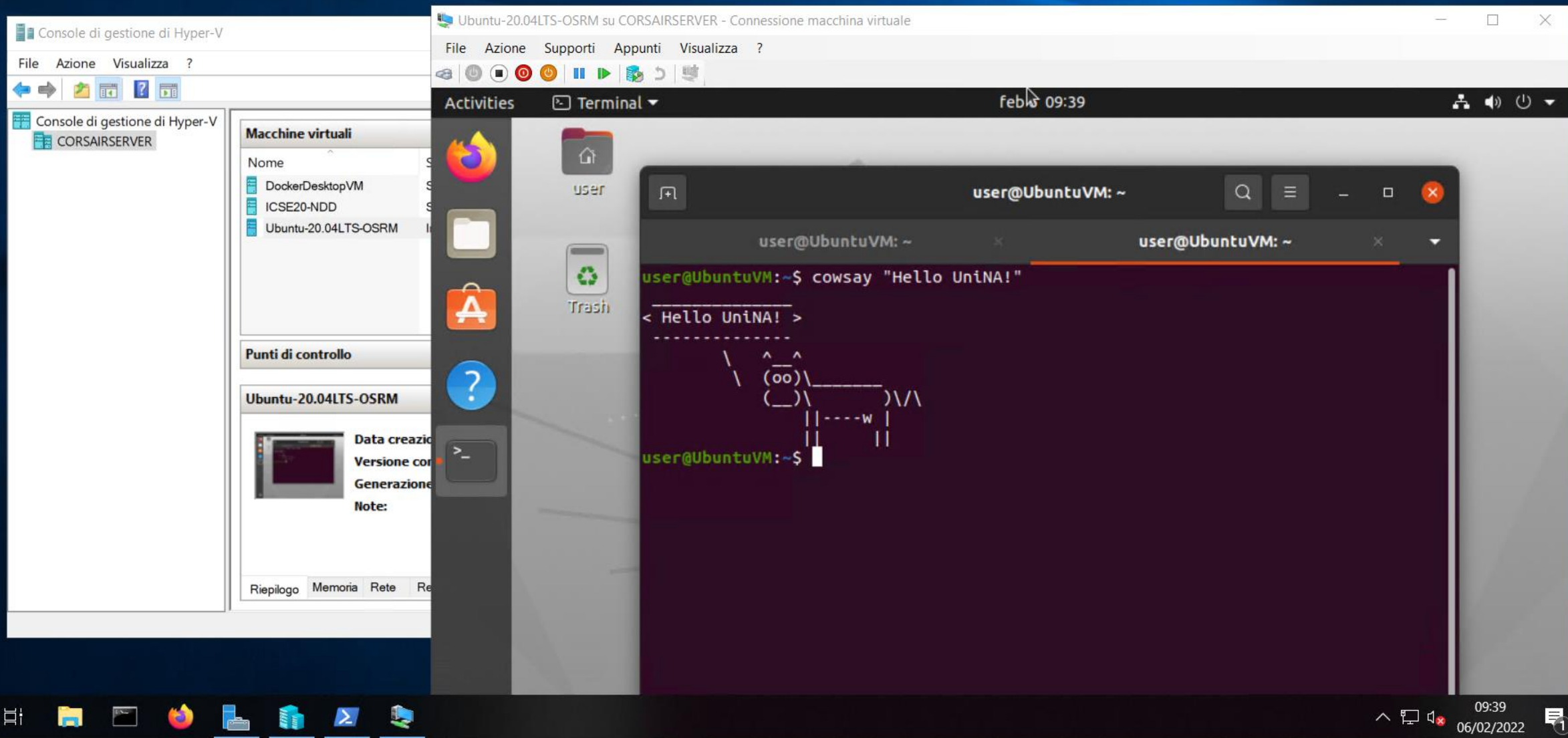
Azioni

CORSAIRSERVER

- Nuovo
- Importa macchina virtuale...
- Impostazioni Hyper-V...
- Gestione commutatori virtuali...
- Gestione SAN virtuale...
- Modifica disco...
- Controlla disco...
- Arresta servizio
- Rimuovi server
- Aggiorna
- Visualizza
- Guida

Ubuntu-20.04LTS-OSRM

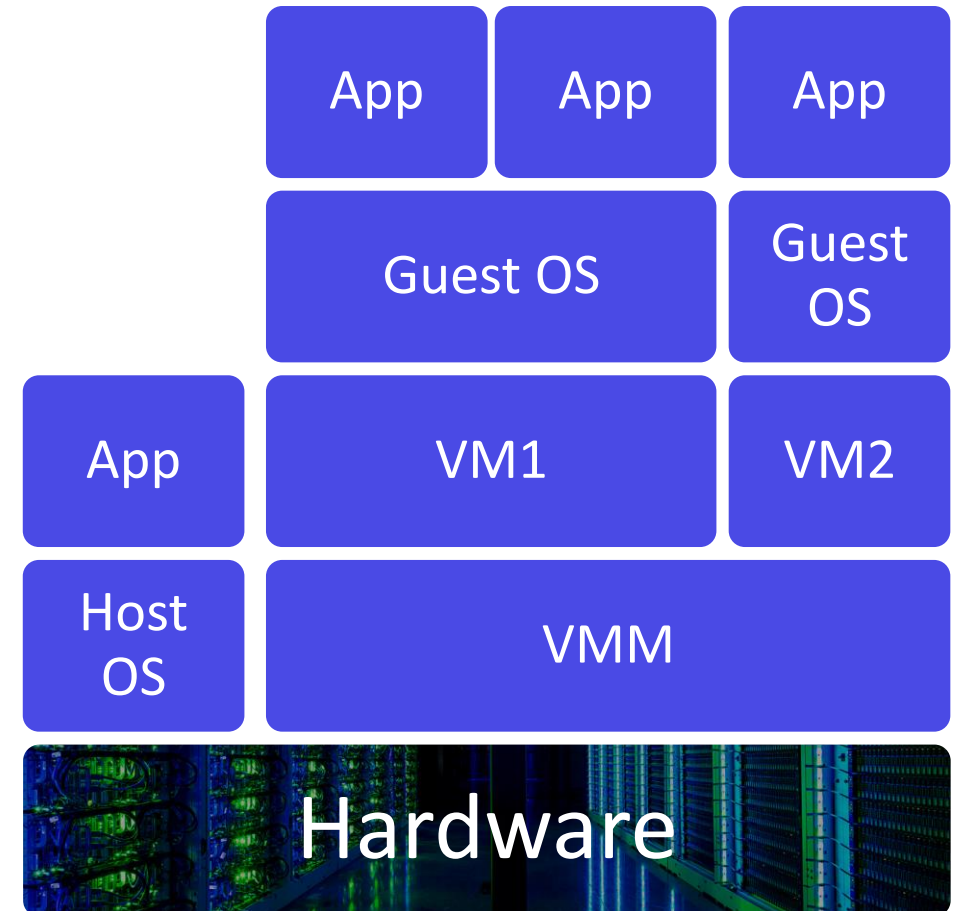
Microsoft Hyper-V Management Console

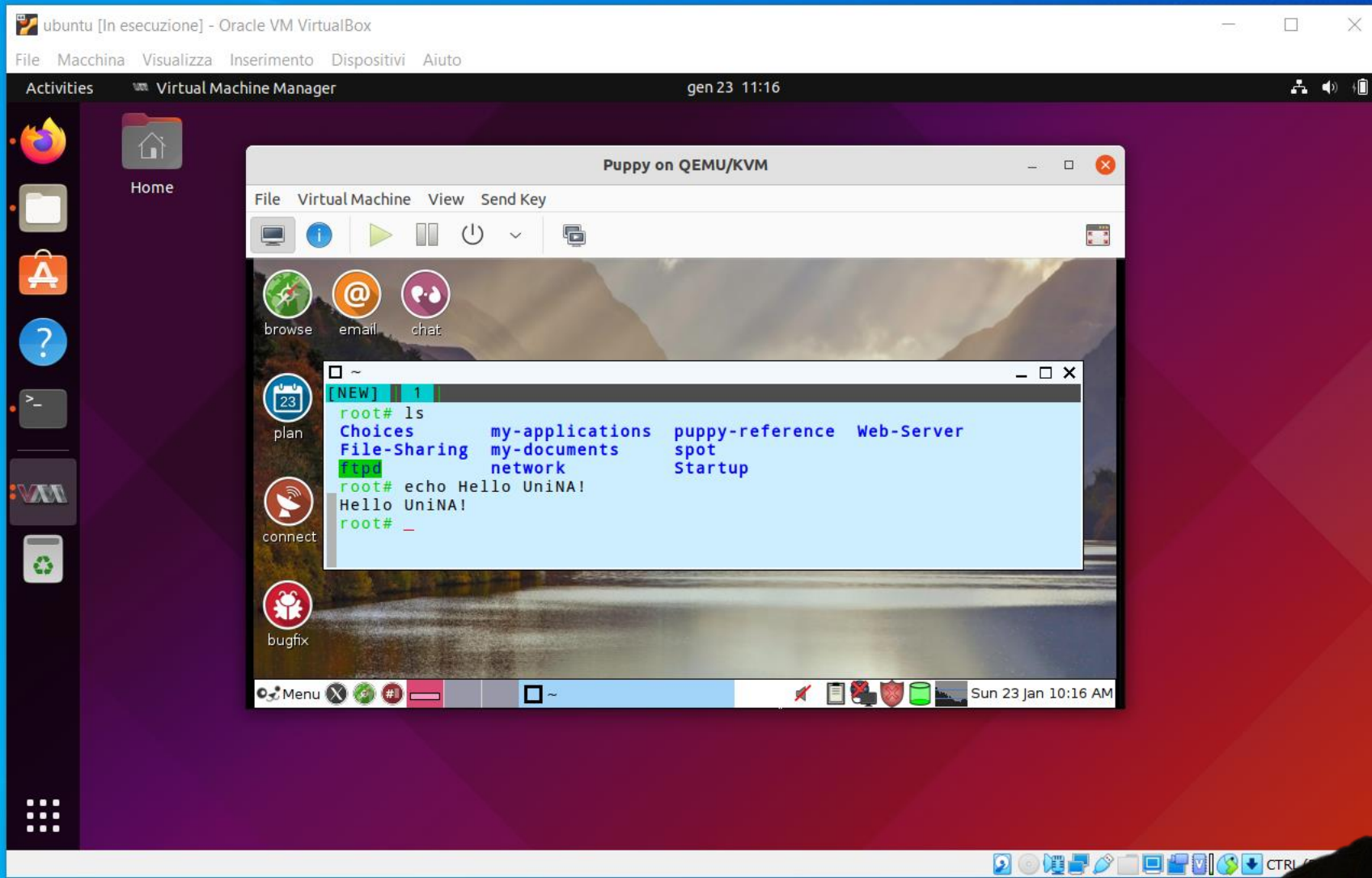


Microsoft Hyper-V Management Console + Ubuntu Guest

Hybrid VMM









- VMM shares the hardware with an existing OS
- Same performance as Bare Metal ones
- Example: Linux + KVM, Xen
- KVM (Kernel-based Virtual Machine) is a set of kernel modules that introduce Bare Metal VMM capabilities





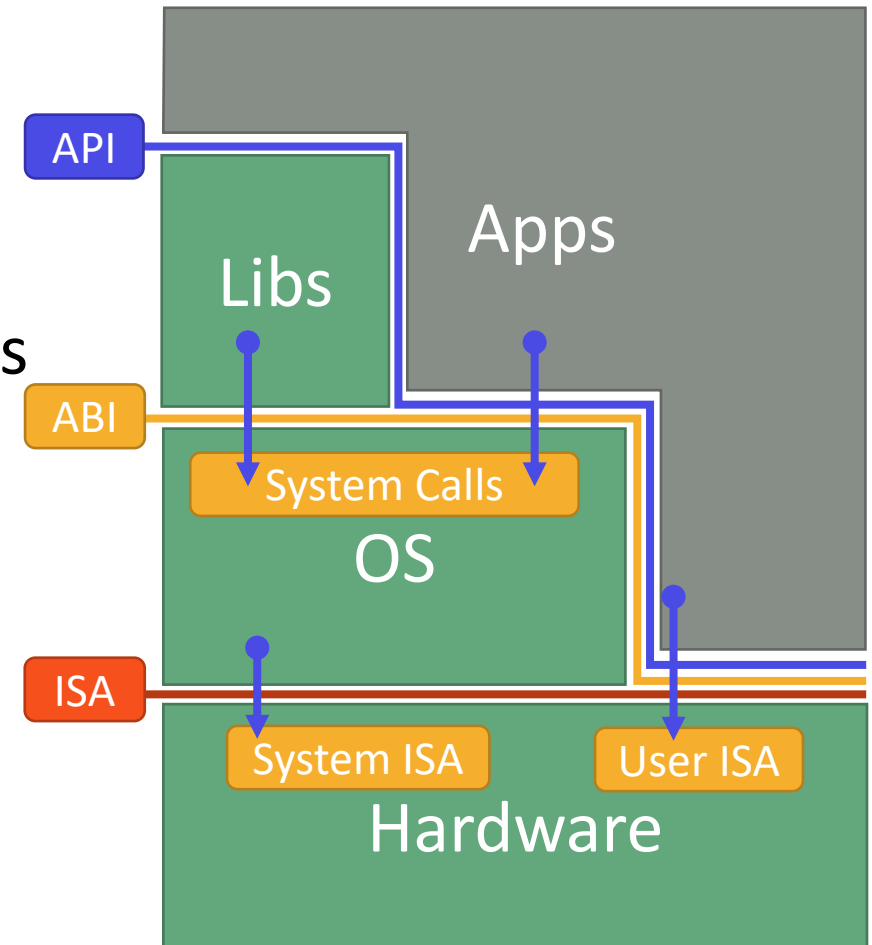
Nested Virtualization: Ubuntu + KVM running a Guest

Virtualization Solutions Overview

	Solution	Type	License
	<u>VMWare ESX/ESXi</u>	Bare Metal	Proprietary
	<u>Microsoft Hyper-V</u>	Bare Metal	Proprietary
	<u>KVM</u>	Hybrid	FOSS
	<u>Xen</u>	Hybrid	FOSS
	<u>XCP-ng</u>	Bare Metal	FOSS
	<u>VirtualBox</u>	Hosted	Mixed
	<u>QEMU</u>	Hosted	FOSS
	<u>VMWare Workstation</u>	Hosted	Proprietary

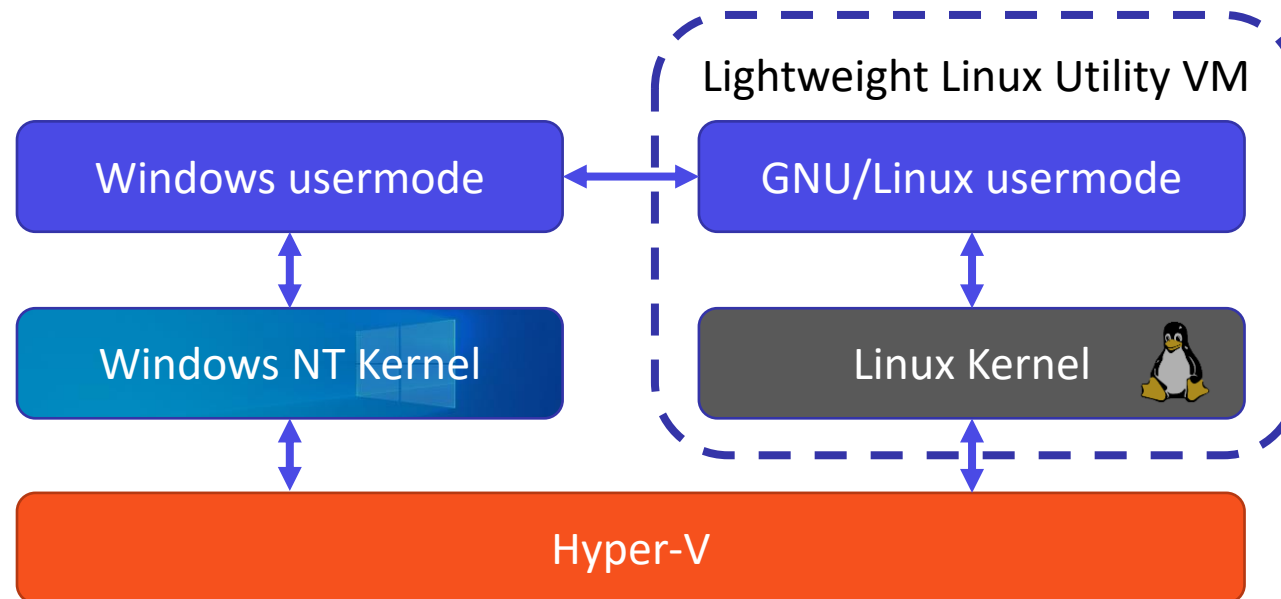
The Windows Subsystem for Linux (WSL)

- The initial release of the WSL is basically a compatibility layer
- It allows users to run Linux binaries by «translating» Linux system call to Windows kernel calls.
- No VM is involved



WSL 2

- WSL 2 features an entirely different architecture from WSL 1
- Leverages Hyper-V features and an actual VM
- Full system call compatibility



Execution Environment Virtualization

Execution Environment Virtualization

Process Level VMs

System Level VMs

Multi-
programming

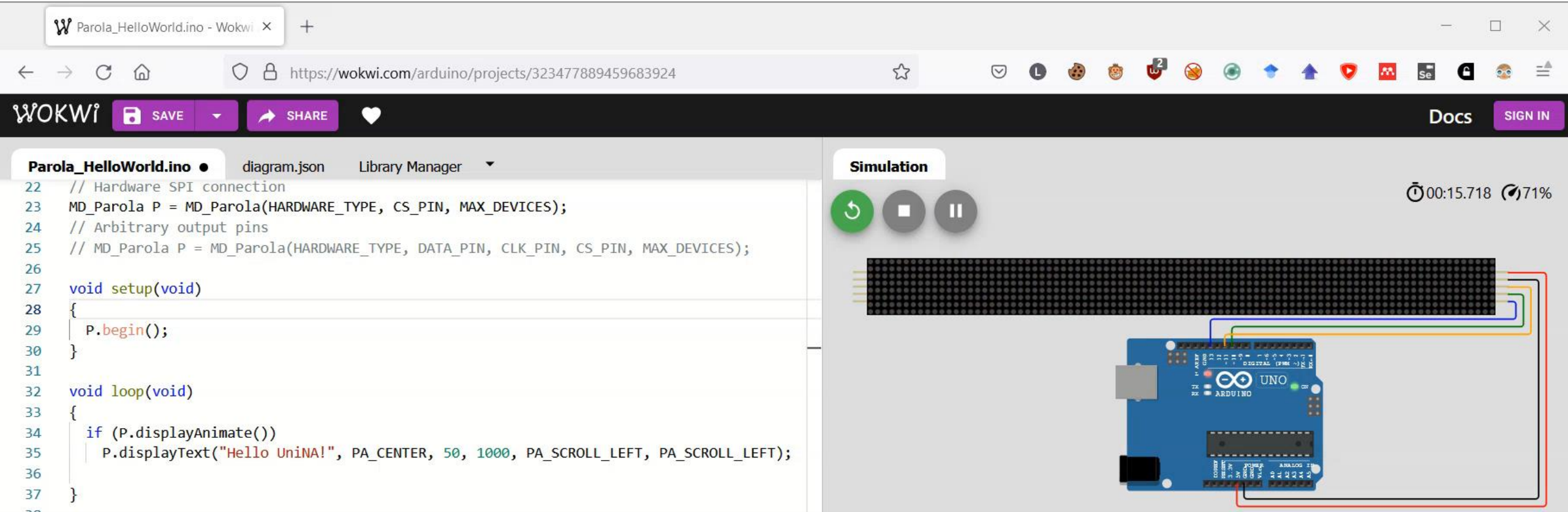
ISA-level /
High-Level
Language VM

OS-level

How do we implement System Level VMs?

Complete Machine Emulation

- VMM implements the complete hardware architecture **in software**
- Each instruction is interpreted → **very slow!** (demo [here](#))



The screenshot displays the Wokwi IDE interface. The top navigation bar includes the Wokwi logo, 'SAVE', 'SHARE', and 'SIGN IN' buttons. The browser address bar shows the URL 'https://wokwi.com/arduino/projects/323477889459683924'. The main workspace is split into two panels. The left panel, titled 'Parola_HelloWorld.ino', contains the following code:

```
22 // Hardware SPI connection
23 MD_Parola P = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);
24 // Arbitrary output pins
25 // MD_Parola P = MD_Parola(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, MAX_DEVICES);
26
27 void setup(void)
28 {
29   P.begin();
30 }
31
32 void loop(void)
33 {
34   if (P.displayAnimate())
35     P.displayText("Hello UniNA!", PA_CENTER, 50, 1000, PA_SCROLL_LEFT, PA_SCROLL_LEFT);
36 }
37 }
```

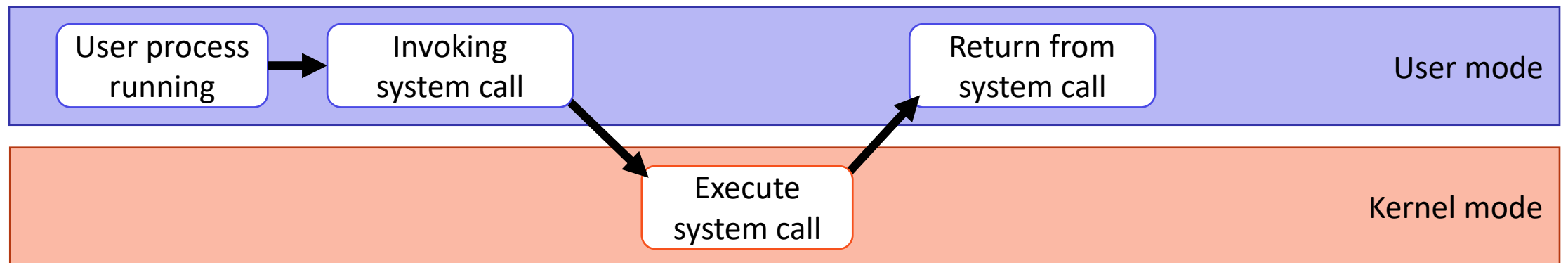
The right panel, titled 'Simulation', shows a virtual representation of an Arduino Uno board connected to a Parola display module. The simulation controls include a green play button, a grey stop button, and a grey pause button. A timer in the top right corner indicates a duration of 00:15.718 and a zoom level of 71%.

Efficient Virtualization

- Complete Machine Emulation is **unfeasible** for complex systems
 - Too slow, too much overhead
- How can we implement efficient virtualization?
- Ideally, we want each VM to access physical resources with as little overhead as possible.
- Not as easy as it may seem!

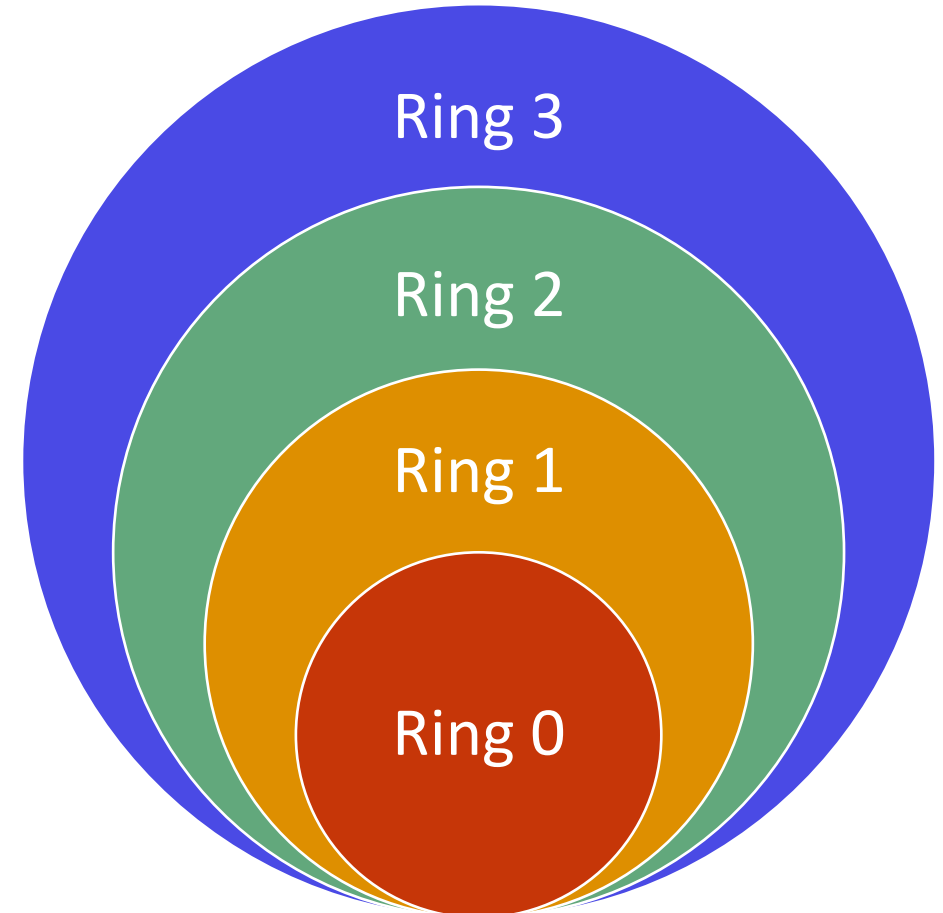
(A look back on) Dual-mode OS operation

- OSs need to protect themselves from user applications
- A hardware-implemented protection mechanism allows two CPU modes: **user mode** and **kernel (supervised) mode**
- Some **privileged** instructions can only be executed in kernel mode



(A look back on) Privilege rings

- x86 supports four privilege rings, only two of them are actually used in modern OSs
- **Ring 0** is the most privileged ring
 - Allows the execution of any instruction
 - Kernel code runs in this ring
- **Ring 3** is least privileged ring
 - User code runs in this ring
 - In this ring, it is not possible to run privileged instructions



Key challenges of System Virtualization

- OSs assume to run in Kernel mode and have complete control on the resources
- In a virtualized environment, the VMM has complete control over the resources (runs in supervisor/kernel mode), while the guest OS and its user programs run in user mode
- **Ring depriving:** Guest OS operates in a ring where it does not belong
- **Ring compression:** we need to protect Guest OS from the user programs it manages

Requirements for Efficient Virtualization

Popek and Goldberg (1974)¹ gave a set of sufficient conditions for a computer architecture to support efficient virtualization:

- 1. A program running under the hypervisor should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.*
- 2. The hypervisor should be in complete control of the virtualized resources.*
- 3. A statistically significant fraction of machine instructions must be executed without the intervention of the hypervisor.*

[1] Popek, G.J., and Goldberg, R.P. (1974). *Formal Requirements for Virtualizable Third Generation Architectures*. Communications of the ACM, vol. 17, pp. 412–421.

Definitions

- **Sensitive** instructions can **change** or **expose** privileged state
 - E.g.: I/O instructions, CLI, POPF
- **Privileged** instructions can only be executed in Kernel mode
 - When executed in user mode, a privileged instruction must **trap** (i.e.: generate a software interrupt and transfer control to the Kernel)

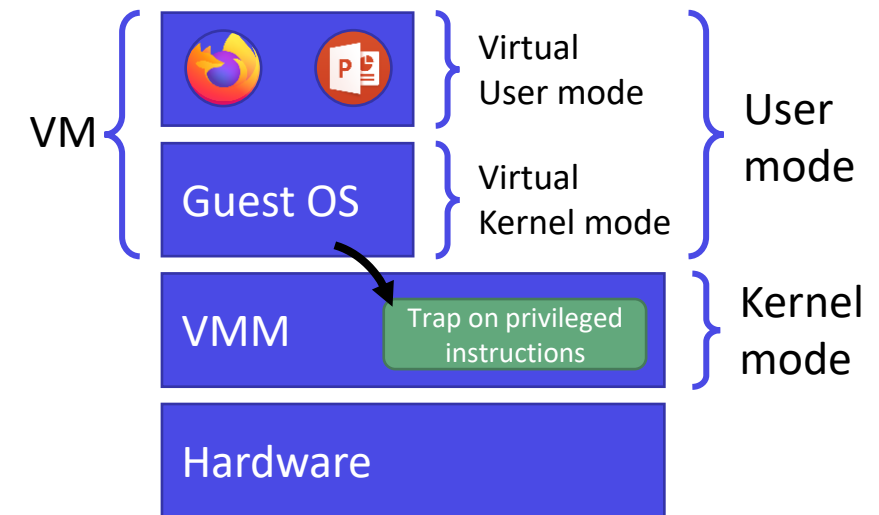
Popek and Goldberg's Theorem

- An efficient VMM may be constructed if the set of sensitive instructions is a subset of the set of privileged instructions.
- In other words, a computer architecture is efficiently virtualizable if every sensitive instruction is privileged
- How? **Trap-and-emulate!**

[1] Popek, G.J., and Goldberg, R.P. (1974). *Formal Requirements for Virtualizable Third Generation Architectures*. Communications of the ACM, vol. 17, pp. 412–421.

Trap-and-emulate

- **Trap:** All the sensitive instruction that could affect the correct functioning of the VMM need to trap when executed in user mode, passing control to the VMM
- **Emulate:** the VMM emulates the effect of the sensitive instruction in software, and returns control to the VM
- Non-sensitive instructions are executed directly without the need of VMM intervention.



Trap-and-emulate: example

- The Guest OS tries to disable interrupts by running **CLI**
- The Guest OS **must not be able to disable interrupts** for the VMM, or for other VMs!
- **CLI** raises a trap when executed in user mode, VMM takes control
- VMM sets the Interrupt Flag (IF) for the specific VM to 0, using a dedicated data structure, and won't send interrupts to the VM
- The IF in the state register of the physical CPU is unchanged

Virtualization Unfriendly Architectures

- Implementing trap-and-emulate is challenging!
- Common architectures are non-virtualizable according to Popek and Goldberg
 - ARM: some instructions are undefined in user mode
 - x86: some instructions are non-virtualizable
 - PUSHF is sensitive but not privileged
 - LTR fails silently when executed in user mode
 - ... and others!

Virtualizing the Unvirtualizable

So, how do we manage to **efficiently** virtualize x86?


- **Binary Translation**
- **Paravirtualization**
- **Hardware-assisted virtualization** (since 2005)

Binary Translation

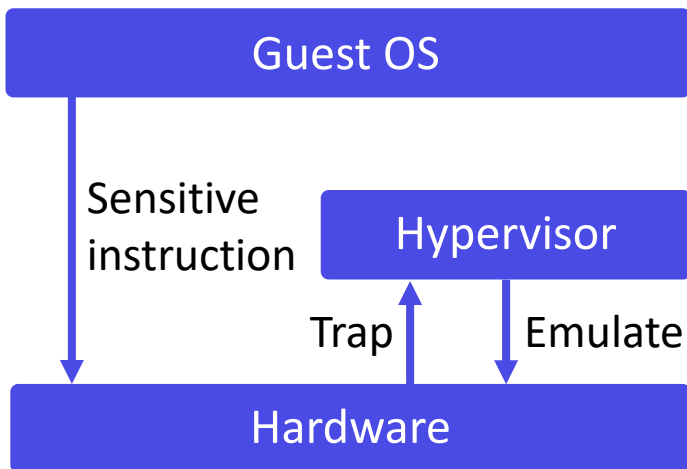
- Employed by VMWare Engineers¹ in first x86 virtualization solutions
- The VMM scans the binary blocks a VM executes, translating instructions ***on-the-fly***. Results are cached for future reuse.
 - The goal is to even avoid trap-and-emulate, when possible
 - E.g.: CLI can be rewritten to invoke the VMM handler, without Trap-and-Emulate!
- Binary Translation is a **full virtualization solution**.
 - It can be used to run any **unmodified** OSs transparently.

[1] Adams, K., & Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. ACM Sigplan Notices, 41(11), 2-13.

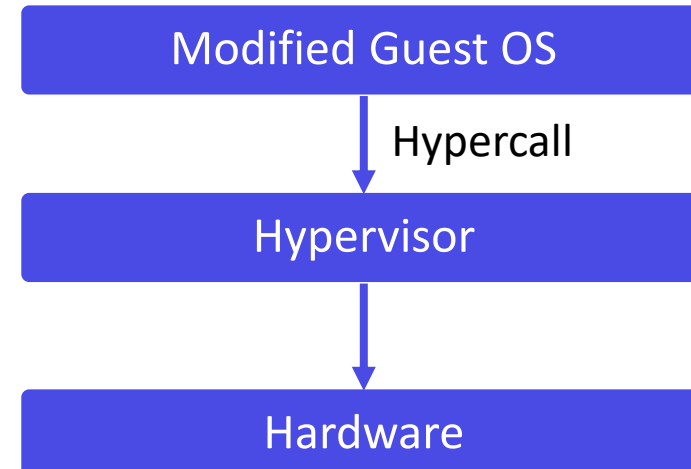
Paravirtualization

- **Idea** : **Enlighten** the Guest OS that it is running in a VM
- The Guest OS Kernel must be modified to invoke virtual APIs (**hypercalls**) exposed by the VMM, instead of sensitive instructions

Full virtualization with Trap-and-Emulate



Paravirtualization

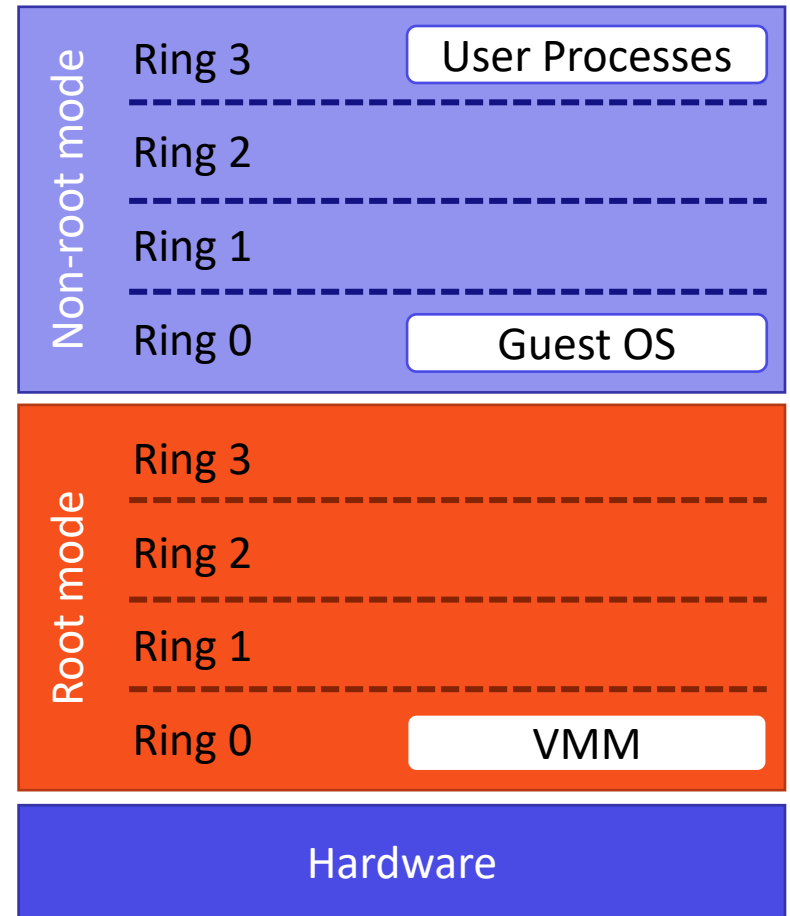


Paravirtualization: Pros and Cons

- Paravirtualization can be efficient (some EC2 instances on AWS use it)
- It is relatively easy to implement
- Requires Guest OS source code to be available
 - OS that cannot be ported (e.g.: Windows) can use ad-hoc device drivers
- Maintenance cost of the paravirtualized OSs
 - Original OS can't run in a paravirtualized environment
 - Each update needs to be ported to the paravirtualized version
- Each VMM has its own hypercalls: a specifically tailored OS version might be required for each paravirtualization VMM

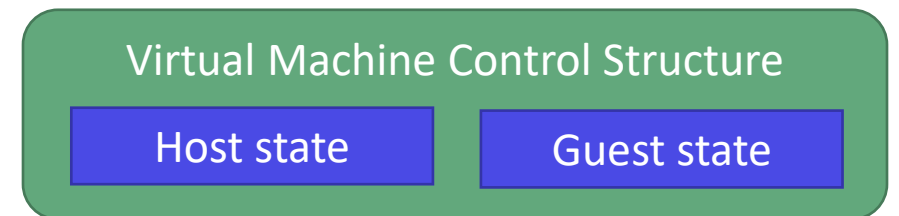
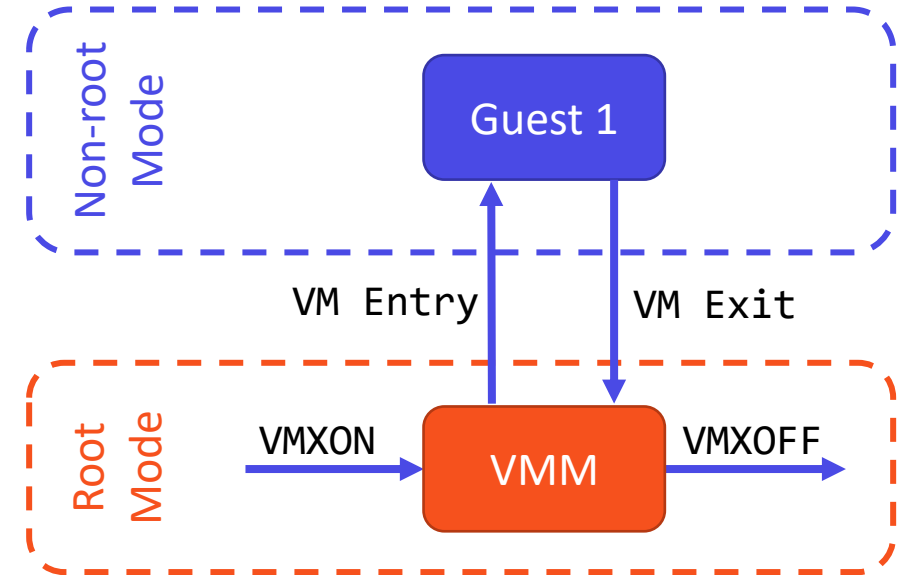
Hardware-assisted Virtualization

- Intel VT-x / AMD SVM introduced in 2005
- Intel's Virtual Machine Extension (VMX) extended x86:
 - New privilege modes: root / non root (guest) mode, each supporting all four x86 protection rings
 - New instructions (VMXON, VMXOFF, ...)
 - New data structure: Virtual Machine Control Structure (VMCS)

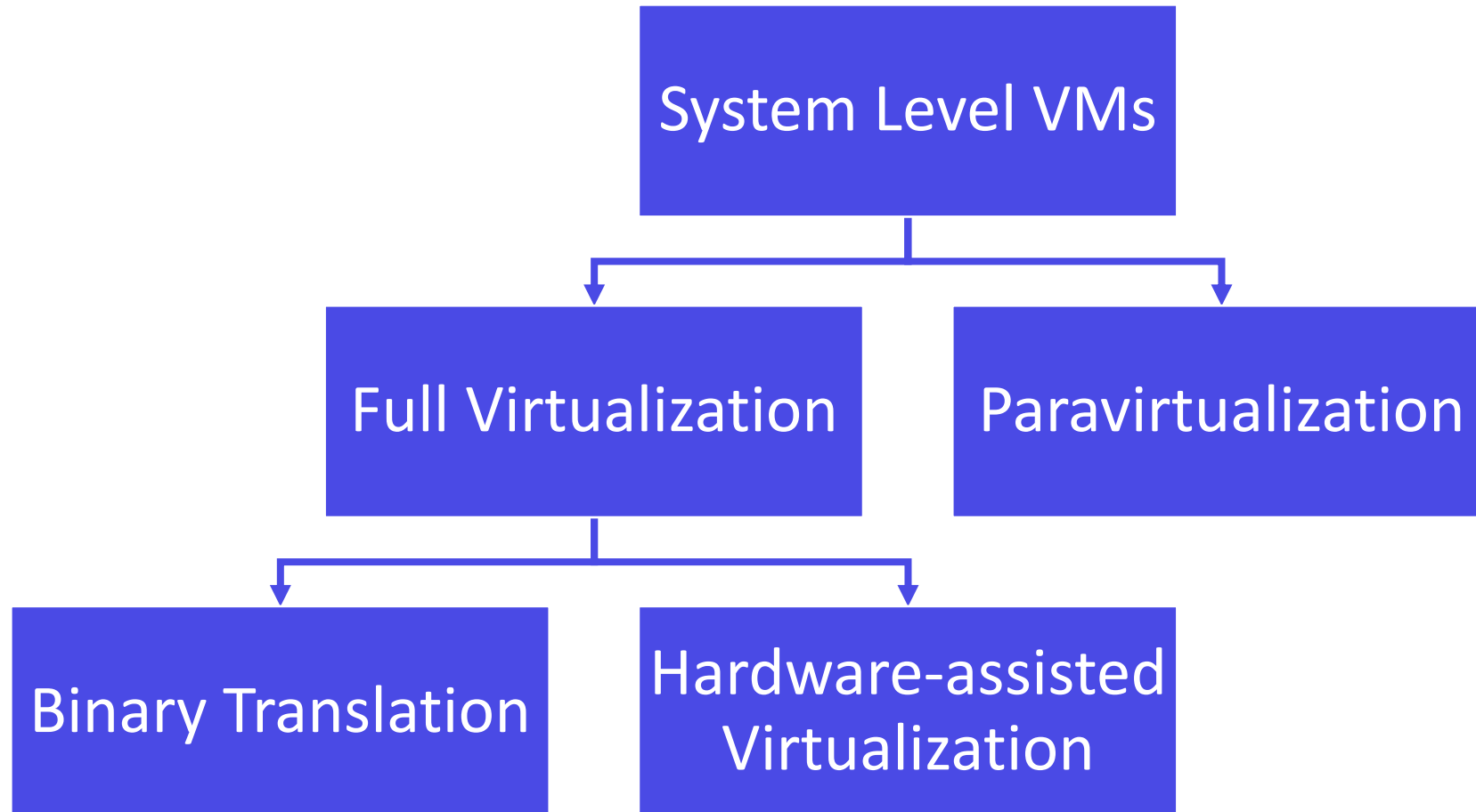


Intel's Virtual Machine Extensions (VMX)

- VMM enters VMX operation with VMXON
- VMM can perform VM Entries (e.g.: VMLAUNCH, VMRESUME)
- Upon VM Entry, processor state is loaded from Guest state in VMCS, control is transferred from VMM to VM
- Upon VM Exit, processor state is saved in Guest state in VMCS, processor state from Host state is restored, control returns to the VMM



Full virtualization and Paravirtualization Recap

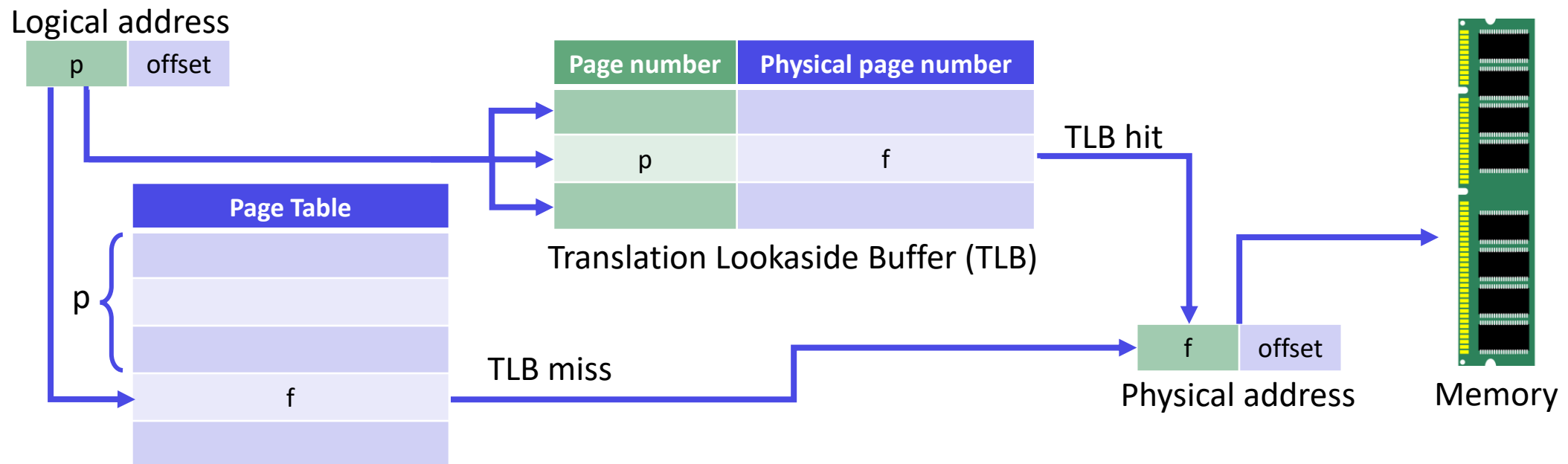


The other piece of the puzzle

- We've talked about how trap-and-emulate can be used to run privileged instructions in user mode in the Guest OS...
- The other big piece of the puzzle is **memory!**
- The OS Kernel thinks it has access to a separate region of memory from the application it manages, and thinks that it can switch context by changing the page table pointer

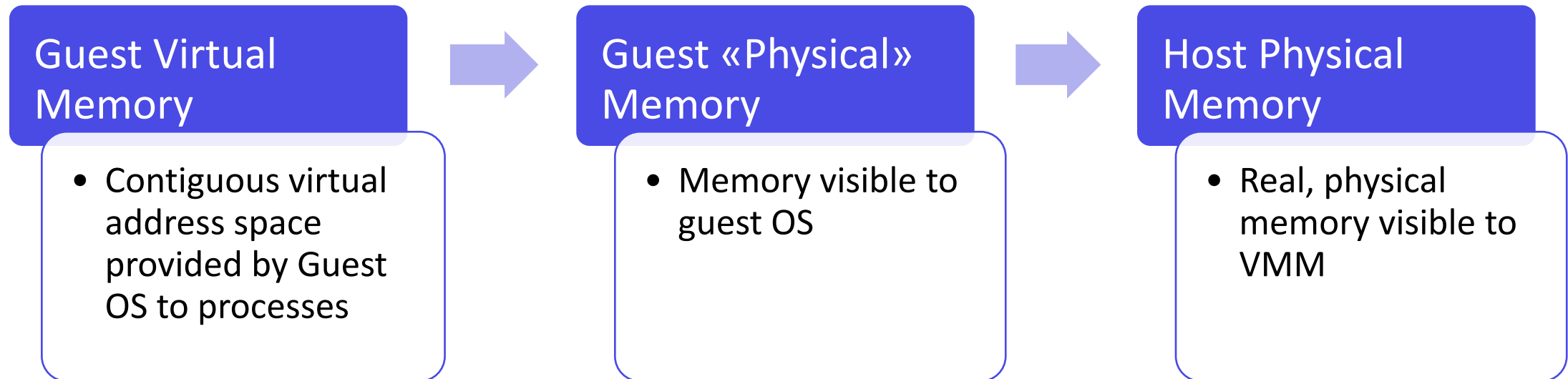
Memory Virtualization in Non-virtualized env.

- In a non-virtualized environment:
 - **One-level memory mapping**: from virtual memory to physical memory
 - **MMU / TLB** are used to efficiently map virtual address to physical ones

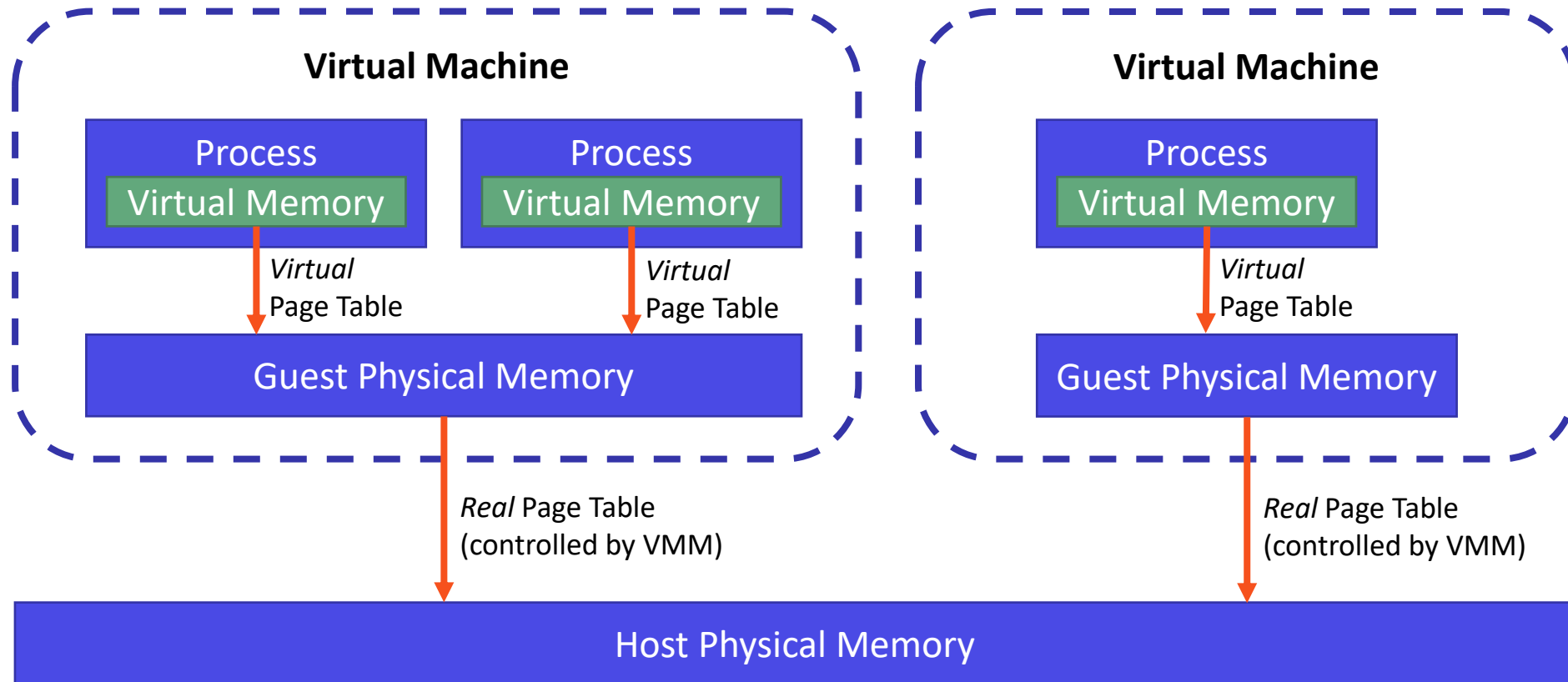


Memory Virtualization in Virtualized envs.

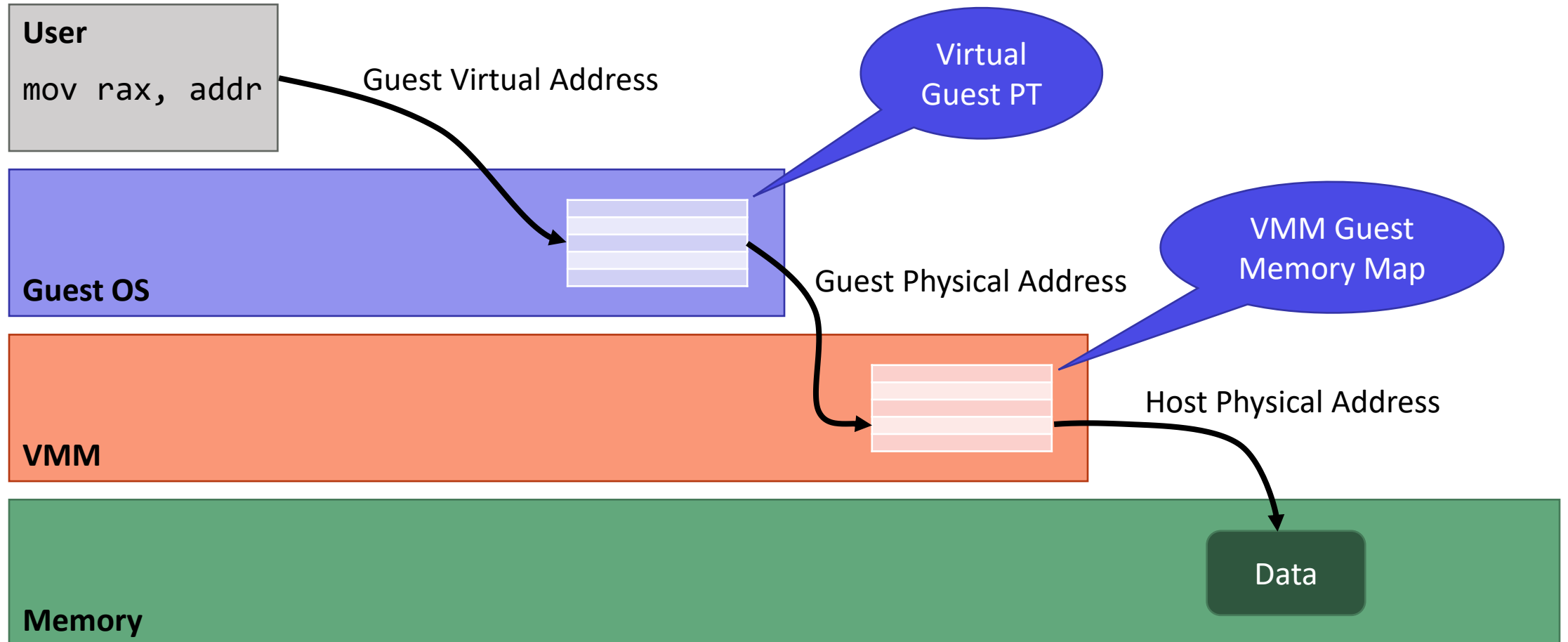
- In a virtualized environment, possibly many VMs, each with its own OS, and the VMM share the same physical memory!
- A two-level memory mapping is needed:



Memory Virtualization in Virtualized envs.



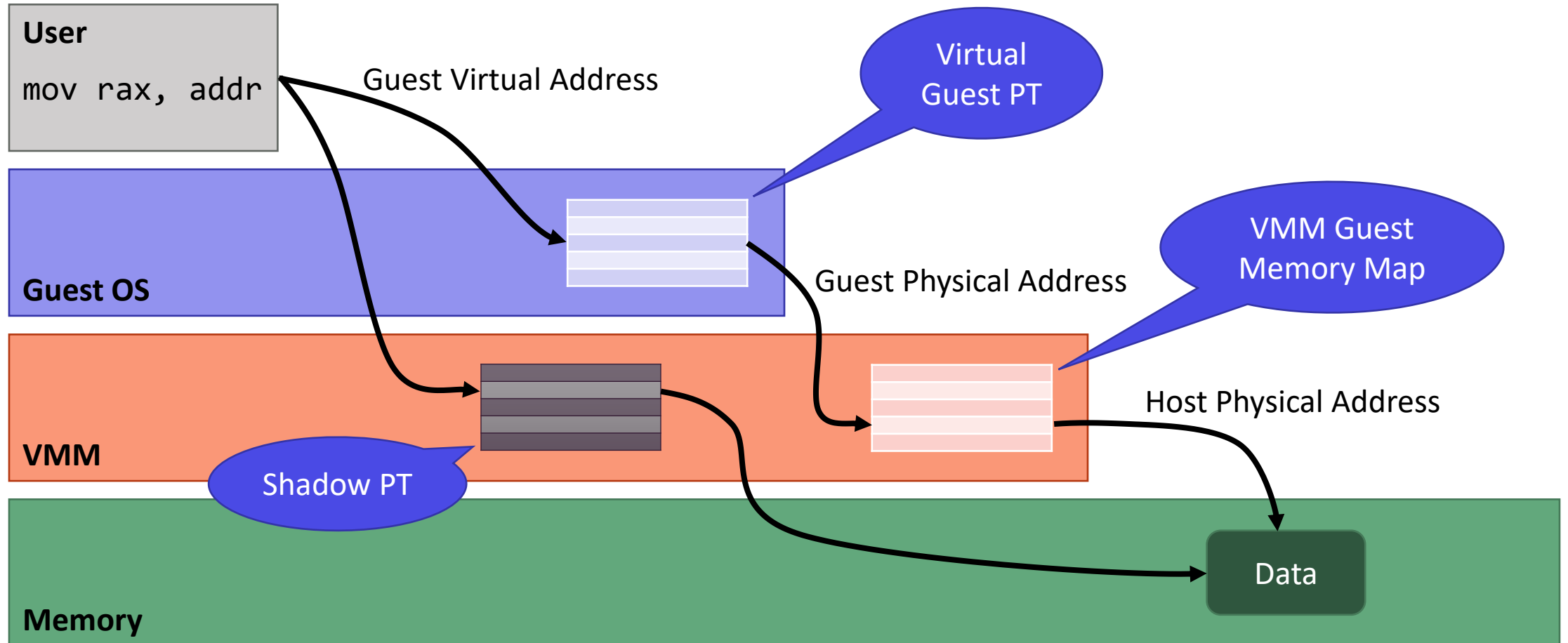
Memory Virtualization in Virtualized envs.



Shadow Page Tables

- To avoid an unbearable performance drop due to the double memory mapping, VMMs maintain **Shadow Page Tables (SPTs)**
- SPTs maps Guest Virtual Addresses to Host Physical Addresses
- Guest OSs creates and manages PTs for its processes as usual
 - But these PTs are not used in the MMU anymore!
- VMM creates and manages SPTs, mapping virtual pages to real machine pages
 - These are loaded in the MMU!

Shadow Page Tables



Challenges with SPTs

- VMM needs to maintain consistency
- When Guest OS changes its PTs, VMM needs to update its SPTs. How?
 - VMM marks Guest OS PTs as read-only
 - When Guest OS writes to its PTs, trap to VMM
 - VMM writes to both SPTs and Guest OS PTs
- Maintaining SPTs still adds an overhead
- Lots of complexity in the VMM
- Solution? **More hardware!**

Second Level Address Translation (SLAT)

- SPTs are a purely software-managed solution
- **SLAT** (aka **Nested Paging**) is a hardware-assisted solution for memory virtualization
 - Intel's Extended Page Table (EPT), AMD's Rapid Virtualization Indexing (RVI)
 - Guest OS still maintains its PTs
 - VMM maintains an additional level of Nested Page Tables (NPTs)
 - Both Guest OS PTs and NPTs are exposed to (and walked by) the hardware
 - Can significantly improve performances¹

[1] Performance Evaluation of Intel EPT Hardware Assist on VMWare ESX.

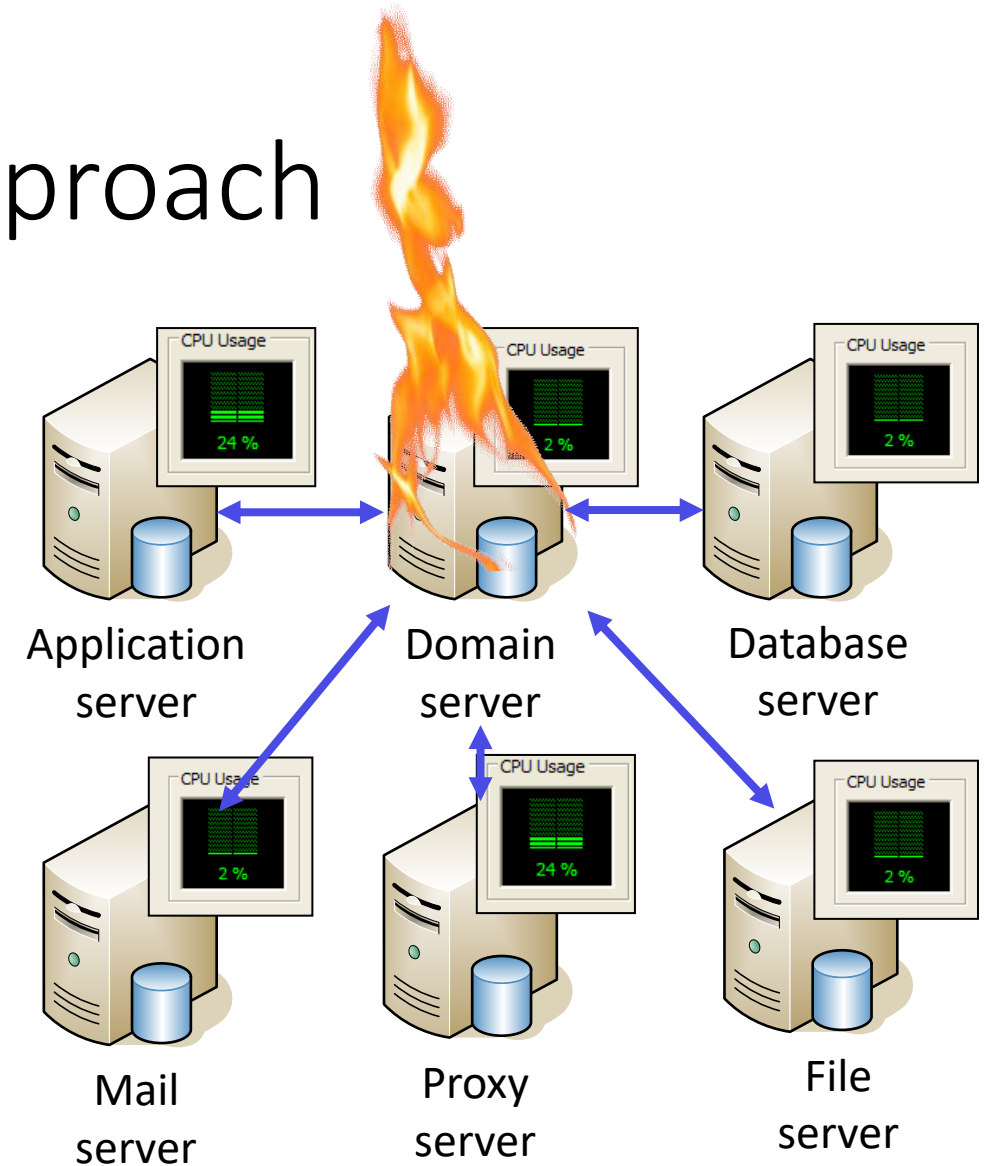
https://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf

Virtualization in practice

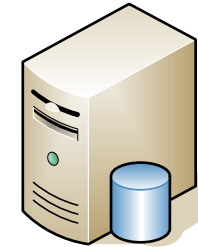
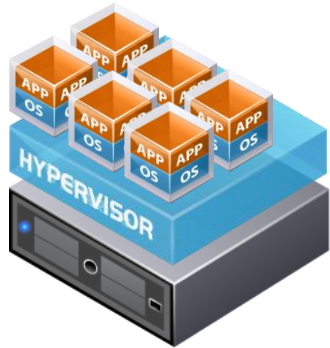
Some virtualization use cases from the real world

The ancient data center approach

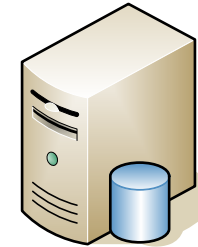
- **One workload, one box.**
- New service? Add a new box!
- Most of the times, underused.
- Hard to manage
 - Backups, rolling updates, ...
- **Single Points of Failure**



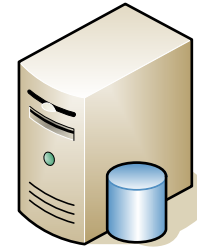
Data center consolidation



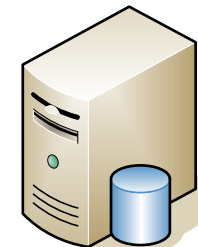
Application server



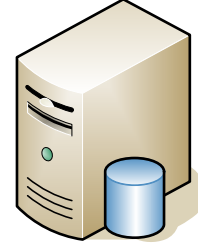
Domain server



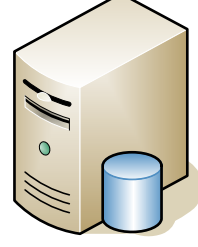
Database server



Mail server

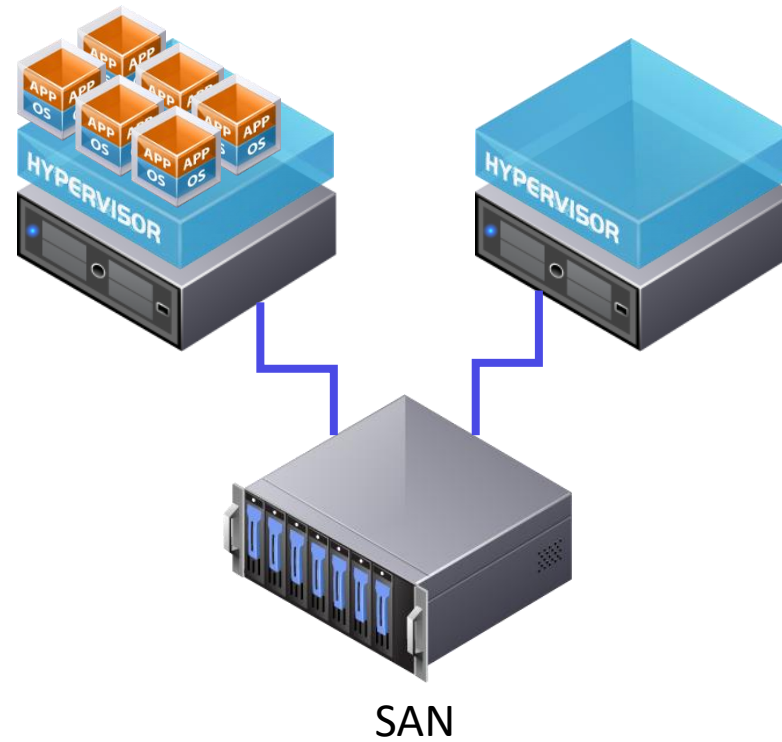


Proxy server



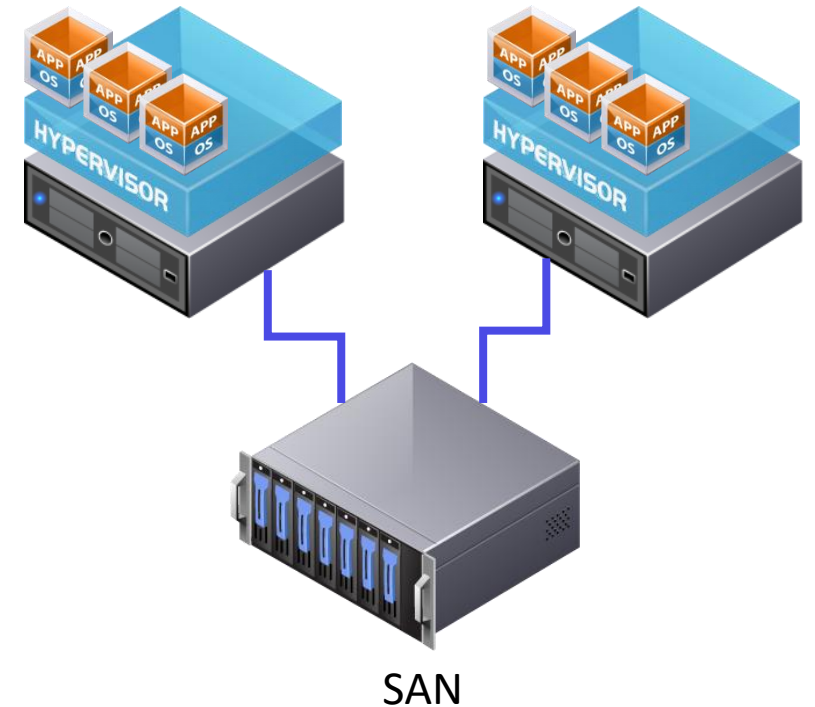
File server

High-availability data centers



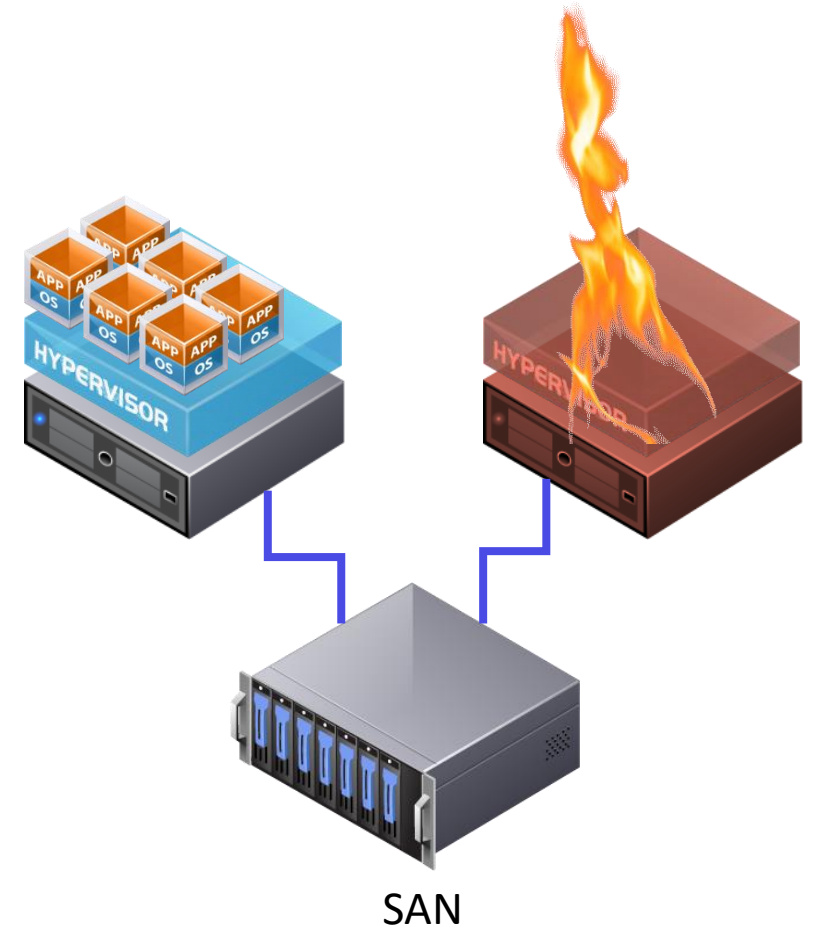
High-availability data centers

- Enterprise data centers typically feature many servers.
- Each physical server is connected to a Storage Area Network (SAN), where VMs are persisted.
- A VM can be migrated from one server to another to balance the load
- ... or grant high availability in case some hardware issue arises



High-availability data centers

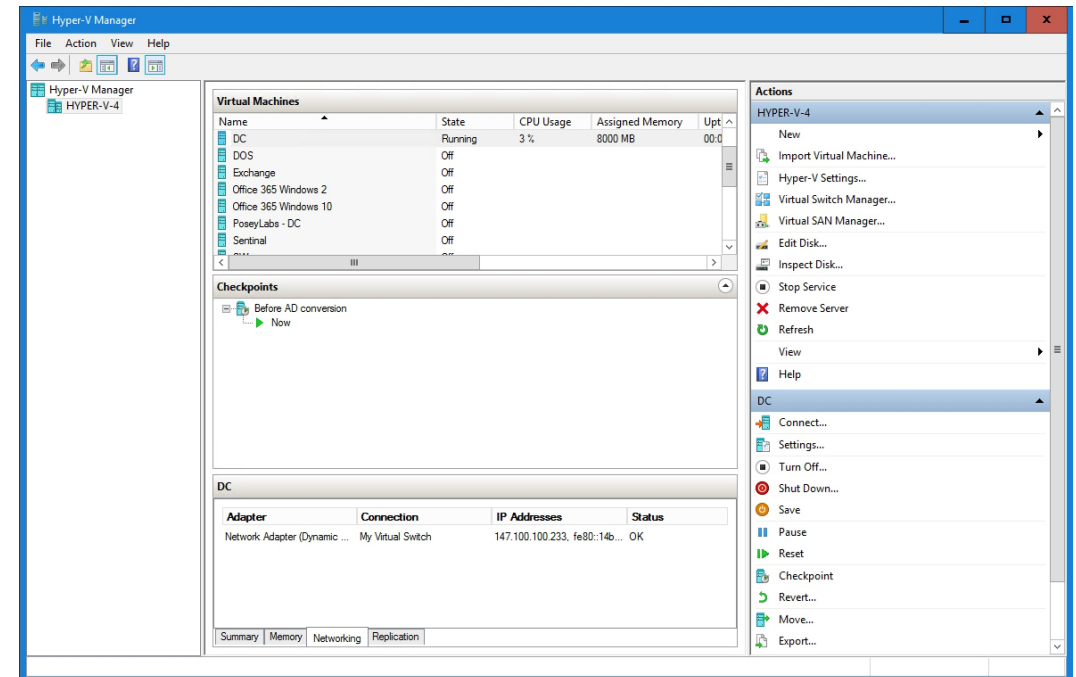
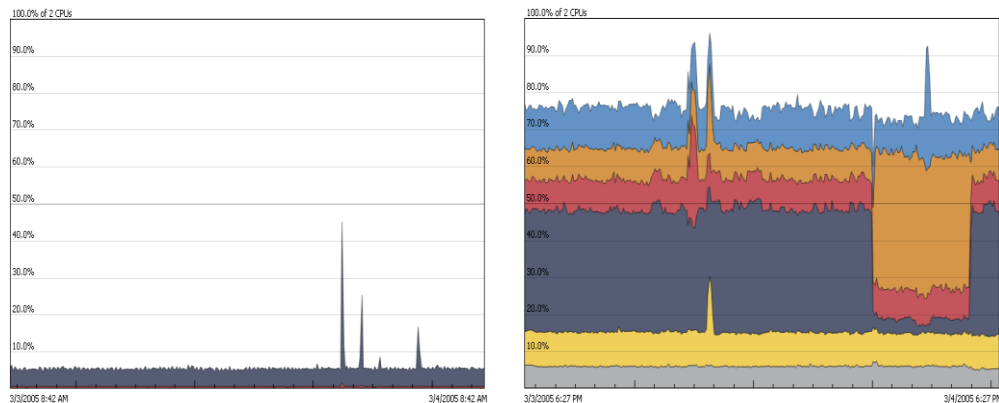
- Enterprise data centers typically feature many servers.
- Each physical server is connected to a Storage Area Network (SAN), where VMs are persisted.
- A VM can be migrated from one server to another to balance the load
- ... or grant high availability in case some hardware issue arises



Data center consolidation: benefits

- Better exploitation of the hardware (less wasted space, electricity)
- Ease of management (thousands of VMs can be managed from a single console)
- High availability

CPU load %: One workload vs many with Virtualization

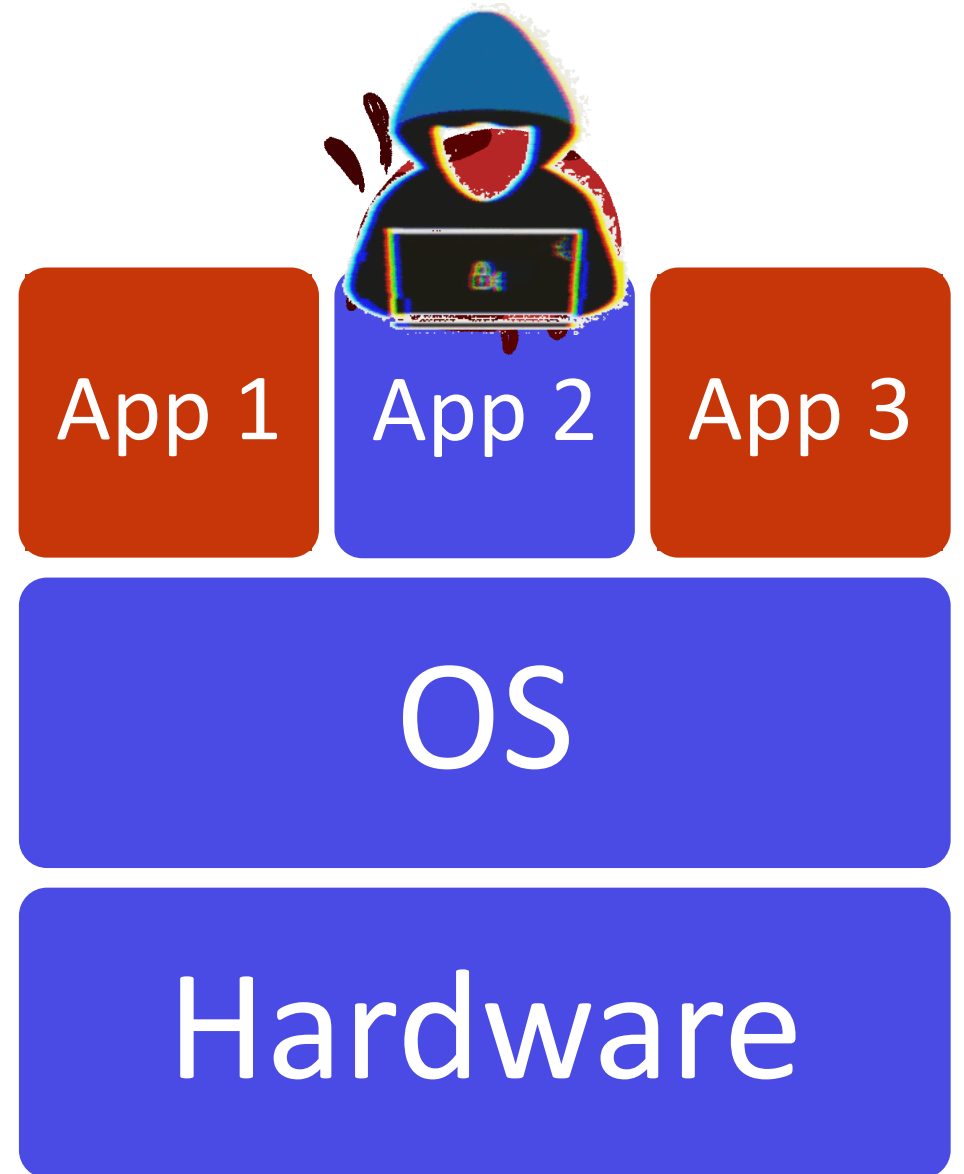


Security

Suppose you need to run different services on the same machine.

- What if one of the services is compromised?
- What if one of the services causes a memory leak or gets stuck and starts spawning new processes in a loop?

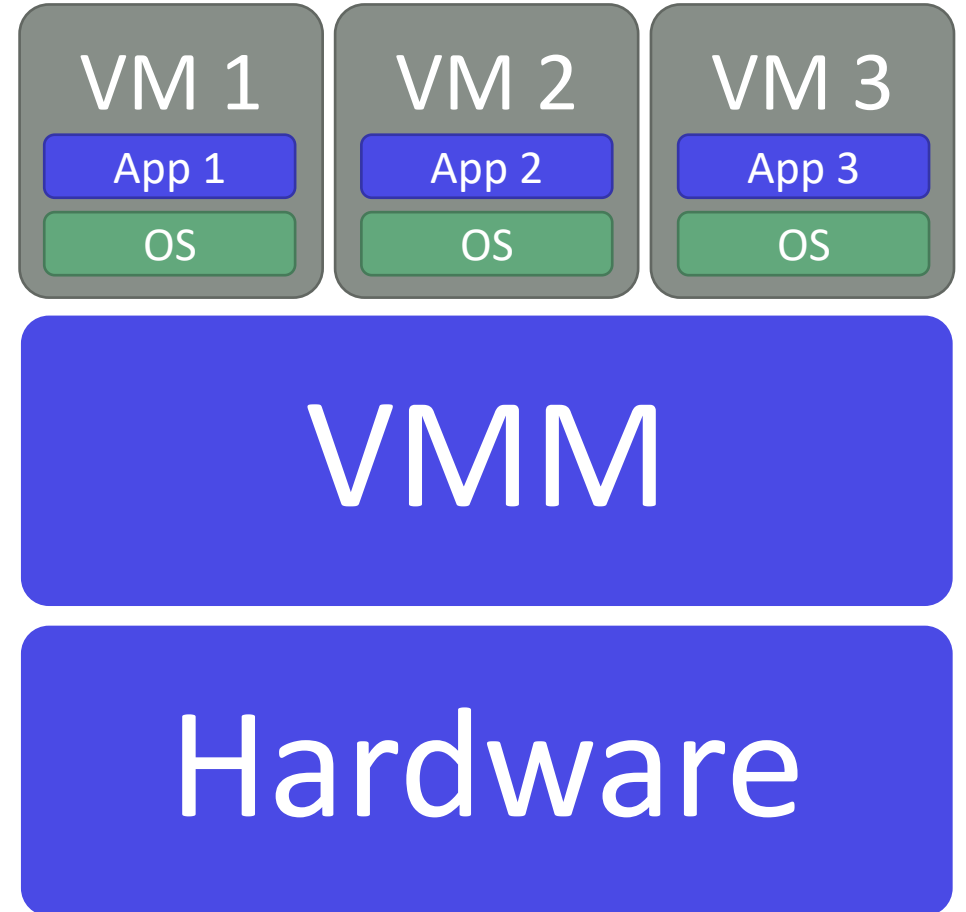
The remaining services could be impacted as well.



Security and Virtualization

With virtualization, each service can be run in its dedicated execution environment, isolated from others

- Security risks of attacks starting from a vulnerable component are largely mitigated
- Each service can at most consume the resources it has been given, with no impact on other services in case of bugs



Cloud Computing

- Virtualization is, along with networking, the key enabling technology for **Cloud Computing**
- Cloud computing is the **on-demand** delivery of **computing resources** through a cloud services platform via the internet with **pay-as-you-go pricing**.
- Thanks to Virtualization, Cloud providers can leverage **massive** data centres and economies of scale to sell computing resources at very interesting prices.
- Cloud Computing has radically changed the way we think about computing infrastructure!

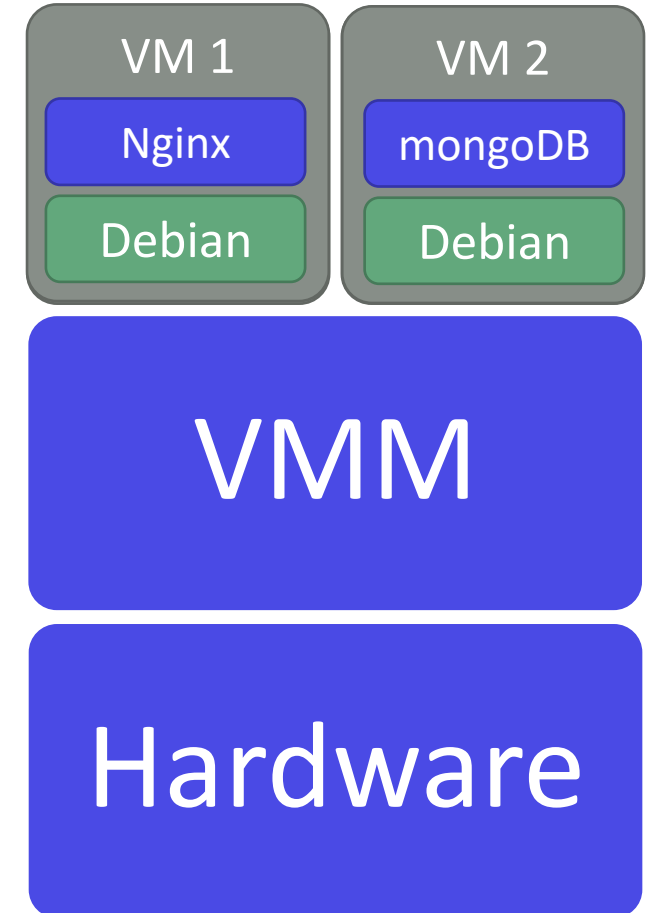
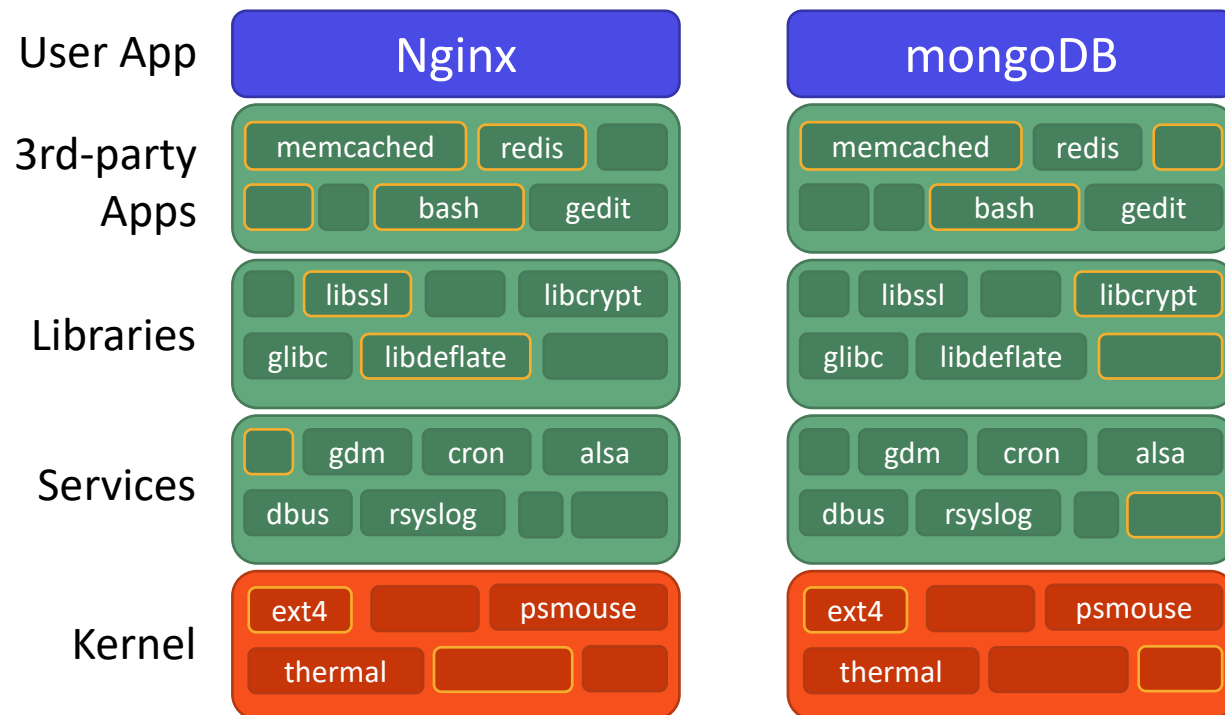
Ease of deployment

- With Virtualization, we moved from «one workload, one box» to «one workload, one VM»
- Often, software is distributed as a VM
 - No more «It works on my machine! $\backslash(_)_/\$ » and configuration/dependency hell!
- VMs can be moved around almost as freely as data

The Unikernel approach

Single-purpose VMs

- VMs are often used to deploy **single-purpose** applications



Single-purpose VMs

Using a general purpose OS in a VM is a straightforward solution...

... but it is quite inefficient!

Do we need an entire, general purpose OS just for one application?

Issues with the classic VM model:

- Lots of duplication
 - Wasted CPU cycles on unnecessary stuff
 - Needlessly large image sizes (wasting space and requiring longer boot times)
- Negative implications on Security
 - Why? **Larger attack surface!**

Can we do better?

- Break up OS functionality into modular libraries
- Link only OS functionality that our app actually needs
- Let the VMM provide memory security mechanisms and manage physical resources

Unikernel¹

- Uni-kernel: a **single-purpose** kernel
- A customized, app-specific OS, providing just enough functionality to run the app it is designed for, and nothing more.
- Fast boot time (milliseconds instead of minutes)
- Small binaries (few MegaBytes instead of few GigaBytes)
- More Security (smaller attack surface, security and isolation pushed down towards the VMM)
- Want to learn more? Check out MirageOS: <https://mirage.io/>

[1] Madhavapeddy, A., & Scott, D. J. (2013). Unikernels: Rise of the Virtual Library Operating System. *Queue*, 11(11), 30-44.

Self-evaluation time!

A few (non-exhaustive) self-assessment questions

A few (non-exhaustive) self-assessment Qs

1. What's the difference between Process- and System-level VMs?
2. What's a Virtual Machine Monitor?
3. What are the three main approaches to implement VMMs?
4. Describe the trap-and-emulate mechanism. Why do we need it?
5. What's the difference between full virtualization and paravirtualization? Discuss pros and cons of both.
6. Why is a two-level memory mapping needed in virtualized environments?
7. Describe the Shadow Page Table mechanism. Why do we need it?

A few (non-exhaustive) self-assessment Qs

8. What are the implications of Virtualization on Security?
9. Describe the Unikernel approach. What are its pros and cons?
10. What's Cloud Computing?

References (1/3)

1. Smith, J. E., & Nair, R. (2005). *The architecture of virtual machines. Computer, 38(5)*, 32-38. <https://doi.org/10.1109/MC.2005.173>
2. Popek, G.J., and Goldberg, R.P. (1974). *Formal Requirements for Virtualizable Third Generation Architectures*. *Communications of the ACM*, vol. 17, pp. 412–421.
<http://www.cs.cornell.edu/courses/cs6411/2018sp/papers/popek-goldberg.pdf>
3. Marinescu, D. C. (2018). *Cloud computing: theory and practice*. Chapter 10: “Cloud Resource Virtualization”.
<https://doi.org/10.1016/B978-0-12-812810-7.00013-3>

References (2/3)

4. Adams, K., & Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, 41(11), 2-13. https://www.vmware.com/pdf/asplos235_adams.pdf
5. Intel® 64 and IA-32 Architectures Software Developer's Manual: Introduction to Virtual Machine Extensions
<https://www.intel.it/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf>
6. Performance Evaluation of Intel EPT Hardware Assist on VMWare ESX. https://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf

References (3/3)

7. Madhavapeddy, A., & Scott, D. J. (2013). Unikernels: Rise of the Virtual Library Operating System: What if all the software layers in a virtual appliance were compiled within the same safe, high-level language framework?. *Queue*, 11(11), 30-44.
<https://dl.acm.org/doi/pdf/10.1145/2557963.2566628>