La virtualizzazione

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Indice e riferimenti

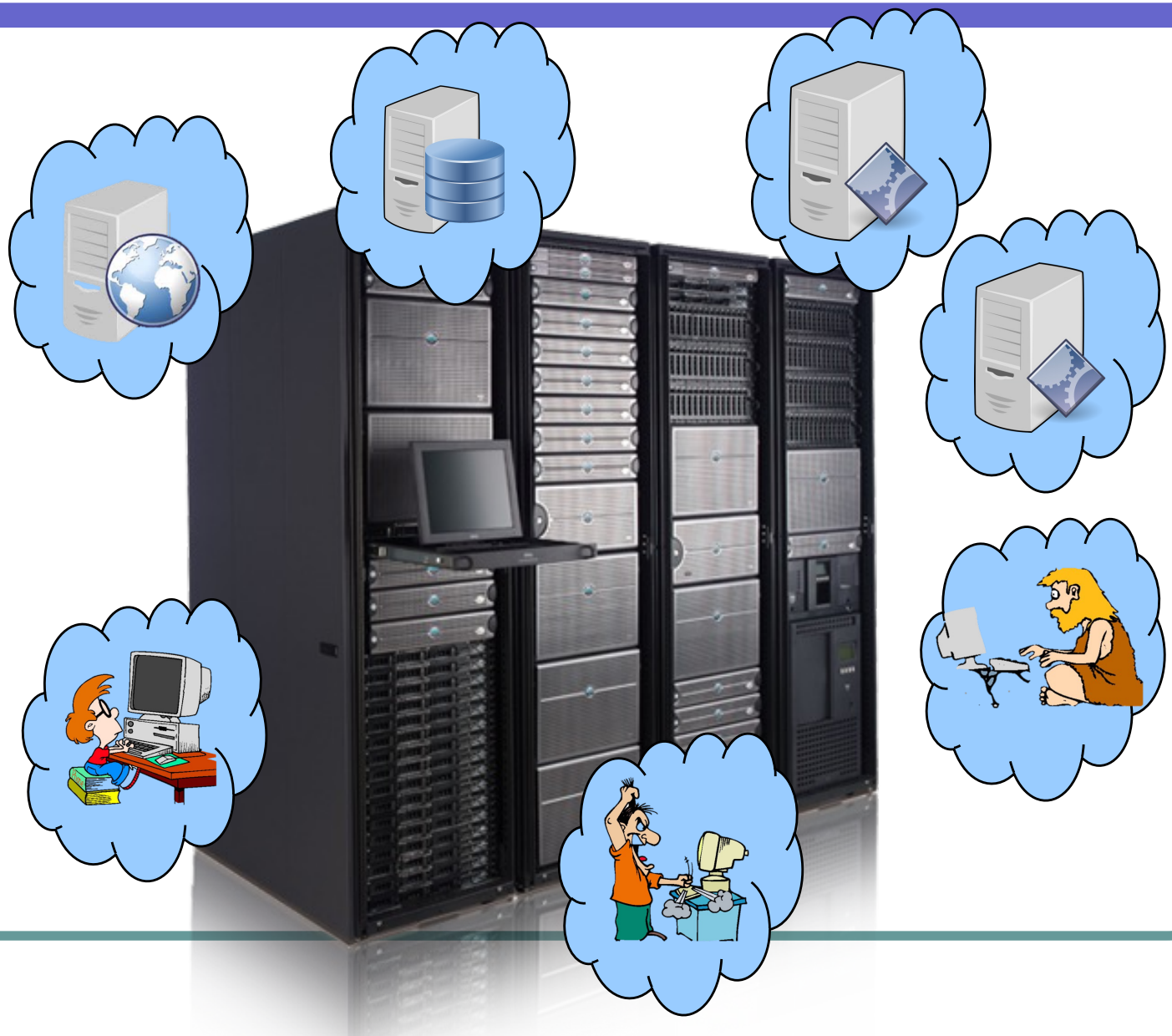
- Argomenti:
  - Utilizzi e benefici della virtualizzazione
  - Architetture e tecniche di virtualizzazione (CPU, memoria, I/O)
  - Approfondimento su VMware Workstation
- Riferimenti:
  - A. Tanenbaum, H. Bos, "*Modern Operating Systems*," Pearson ed., 4ª edizione, 2015 (**Capitolo 7: Virtualization and the Cloud**)
  - Neiger et al., "*Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization*", Intel Technology Journal, 10(3), 2006

Macchine virtuali

- Una **macchina virtuale** (**VM**) è una emulazione (mediante tecniche sw/hw) di un macchina reale
- Ogni macchina virtuale esegue il proprio **sistema operativo** ed applicazioni
- Le VM sono gestite da un **virtual machine monitor** (**VMM**), o **hypervisor**
- Più macchine virtuali condividono le risorse fisiche della macchina su cui eseguono

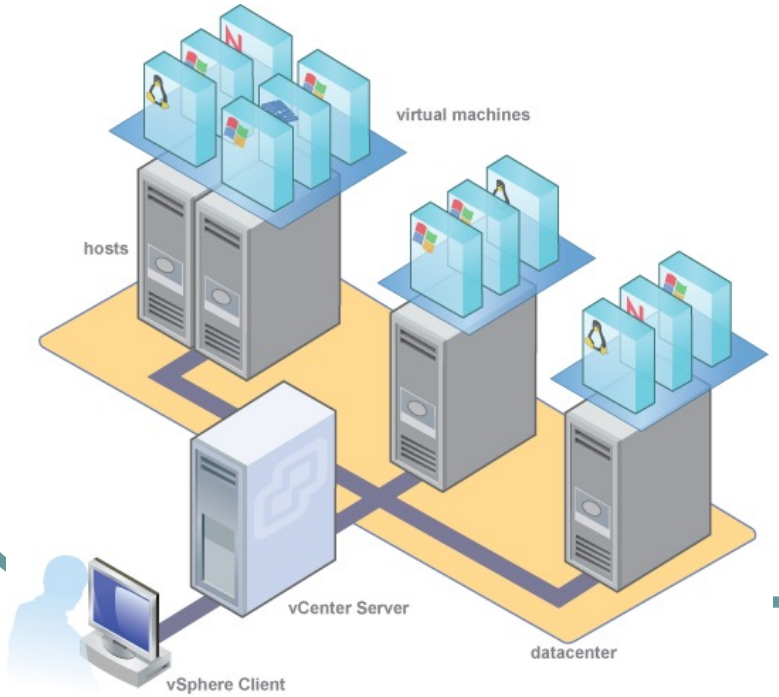
Macchine fisiche e virtuali

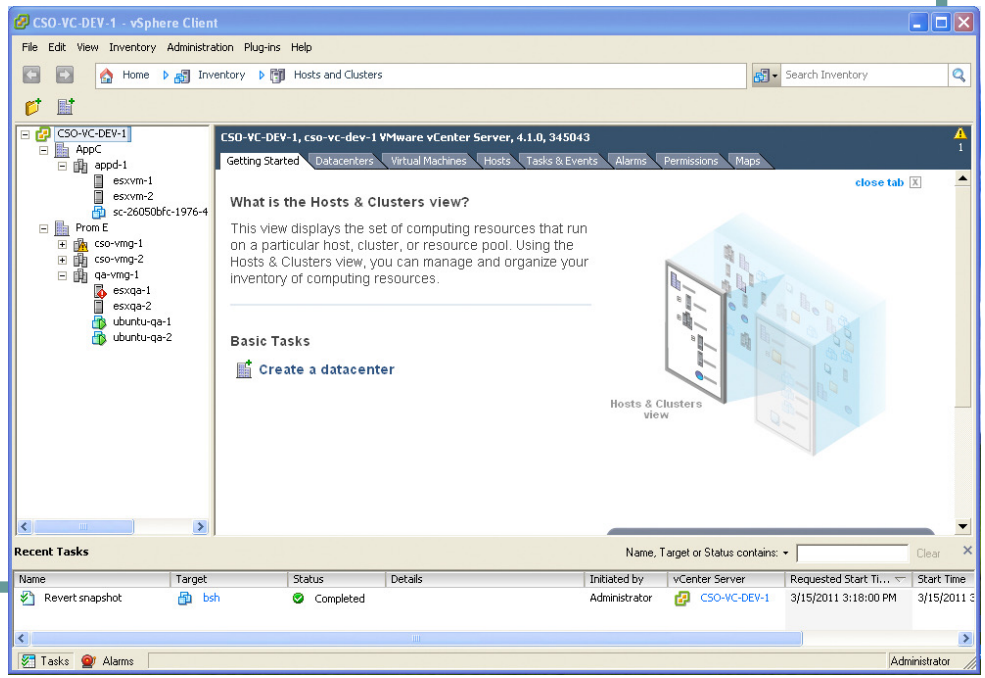
Sistemi virtualizzati



Gestione di sistemi virtualizzati

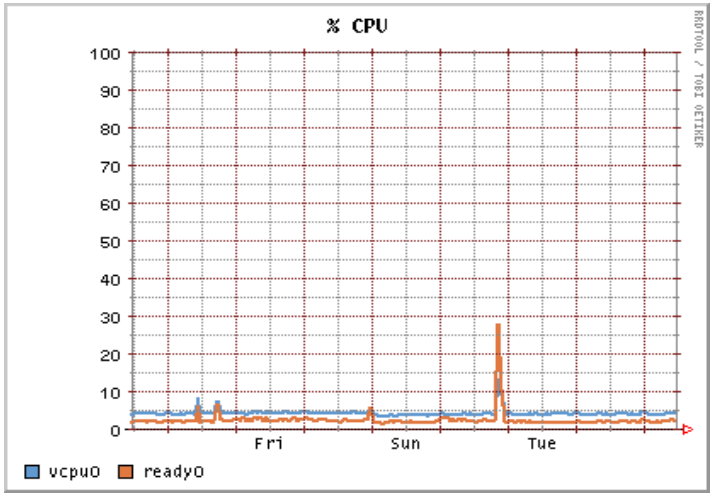
- Le macchine virtuali possono essere velocemente **create, configurate, monitorate, migrate**, ..., attraverso strumenti software centralizzati
  - Costi ridotti, manutenzione più semplice



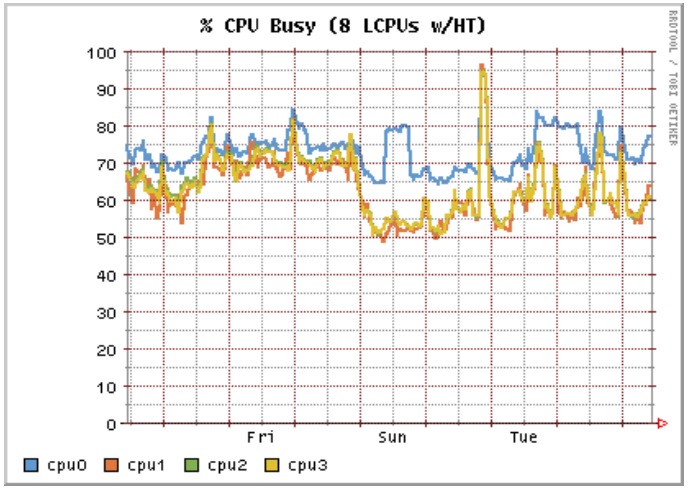


Efficienza

- Avere più VM su una stessa macchina fisica (**workload consolidation**) permette di sfruttare appieno la capacità dell'hardware e di ridurre i consumi energetici



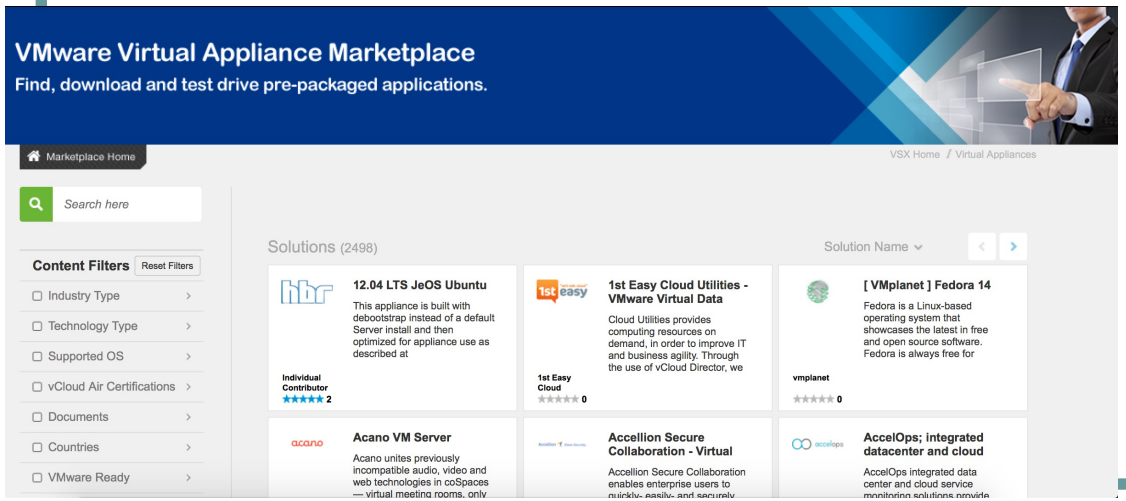
Un singolo server

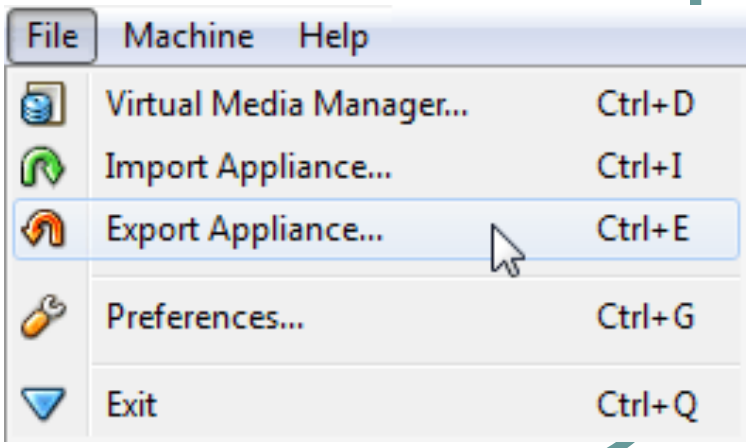


Server consolidati

Flessibilità

- Applicazioni **legacy**, basate su SO obsoleti e non più supportati, possono essere eseguite su macchine moderne
  - Una applicazione (**virtual appliance**) può “portarsi dietro” il suo ambiente di esecuzione
  - versione di SO, librerie, file di configurazione, ...



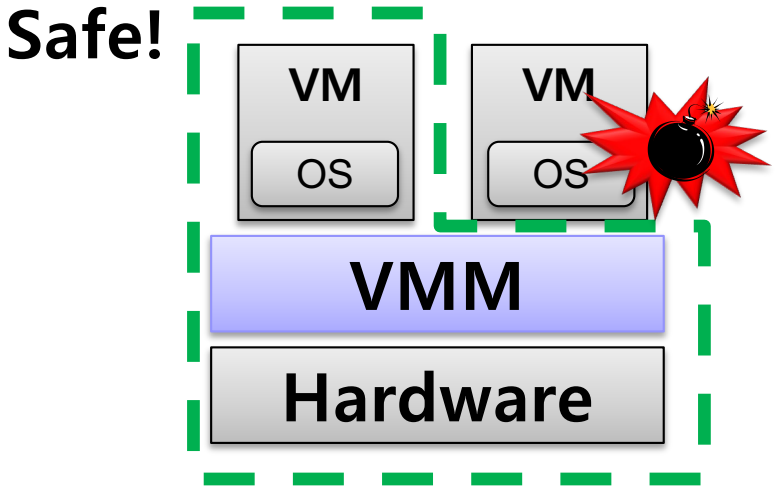


Virtualizzazione e cloud computing

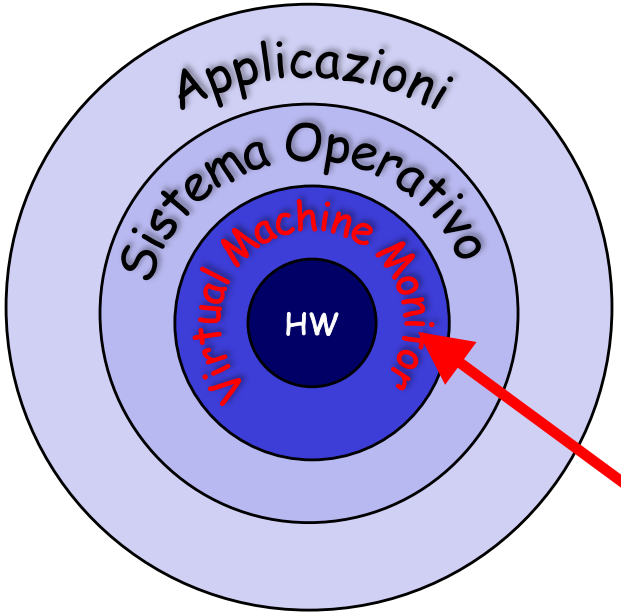
- Il **cloud computing** permette lo outsourcing di VM in centri di calcolo privati o di terze parti (***pay-per-use***)
- I fornitori hanno ampie risorse e personale specializzato, e possono condividere le risorse tra più clienti (**multi-tenancy**), realizzando economie di scala
- I clienti sono sollevati dai costi di acquisizione e manutenzione di un centro di calcolo

Affidabilità e sicurezza

- Le macchine virtuali permette di isolare meglio le applicazioni (affidabilità e sicurezza)
  - Una applicazione difettosa/compromessa “vede” una **VM isolata**, e non può accedere alle altre VM o il VMM
  - Il VMM è l’unico componente **privilegiato** che può gestire l’hardware fisico e le macchine virtuali



VMM e SO a confronto

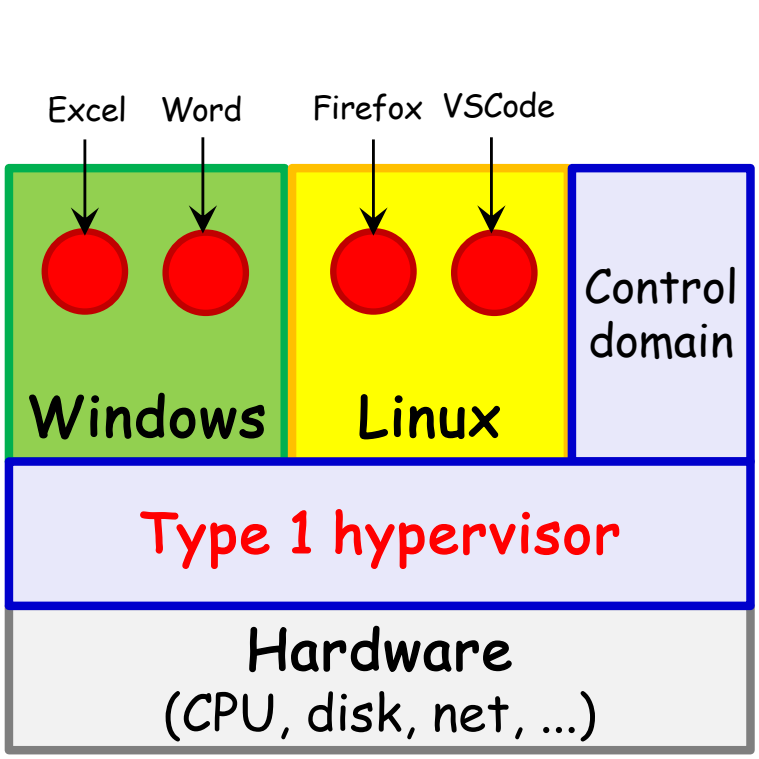


Il VMM (come il SO) fornisce una **astrazione** della macchina fisica su cui esegue

- Emulazione e scheduling di CPU virtuali
- Allocazione di pagine fisiche alle VM
- Emulazione di dischi ed interfacce di rete

Risorse	Sistema Operativo	VMM
CPU	Processi, Thread	CPU virtuale
Memoria	Memoria virtuale	Memoria virtuale
Disco	File, Directory	Disco virtuale
Rete	Socket	Rete virtuale

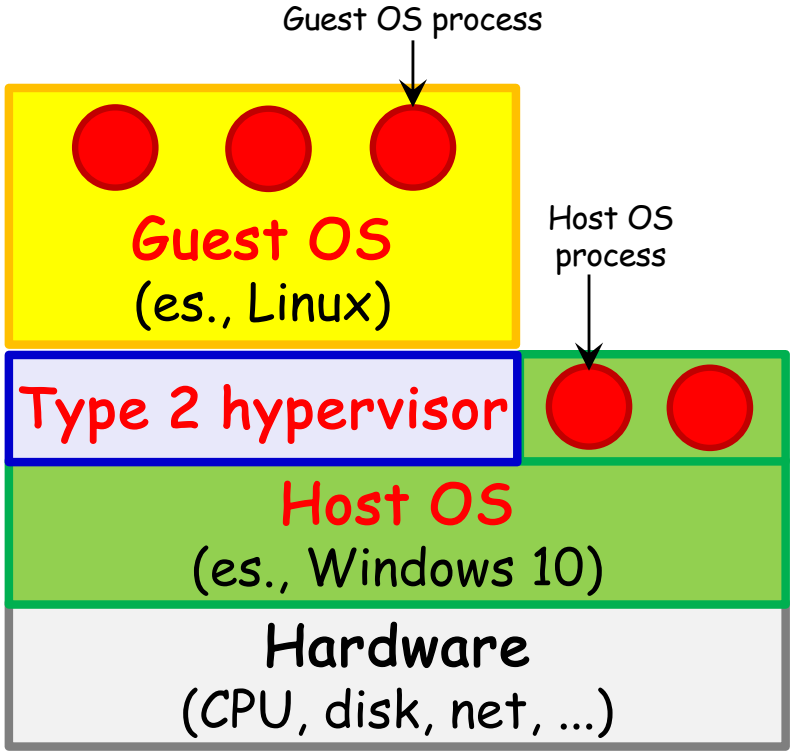
Architetture di virtualizzazione



Hypervisor Tipo 1:

il VMM esegue su "hardware nudo"

(**bare-metal virtualization**)



Hypervisor Tipo 2:

il VMM esegue su un SO tradizionale (es. Windows)

(**hosted hypervisor**)

Architetture di virtualizzazione

- La architettura di tipo 1 ha migliori prestazioni
  - È la soluzione più utilizzata in ambito server
  - VMware ESXi, Xen, Microsoft Hyper-V
- La architettura di tipo 2 facilita la integrazione tra i SO nelle VM (**guest OS**) con il SO di partenza dell'utente (**host OS**)
  - È la soluzione più utilizzata in ambito desktop
  - VMware Workstation/Fusion, Oracle VM VirtualBox

VIRTUALIZZAZIONE DELLA CPU

La “virtualizzabilità” della CPU

- Il SO non è un programma "normale"
- Esso accede in modo **esclusivo e privilegiato** a:
  - Interrupts e interrupt masking
  - Eccezioni (page faults, machine checks, ...)
  - MMU e tabelle delle pagine
  - I/O ports e memory-mapped IO
  - Lo stato del processore (registri di controllo, etc.)
  - ...

La “virtualizzabilità” della CPU

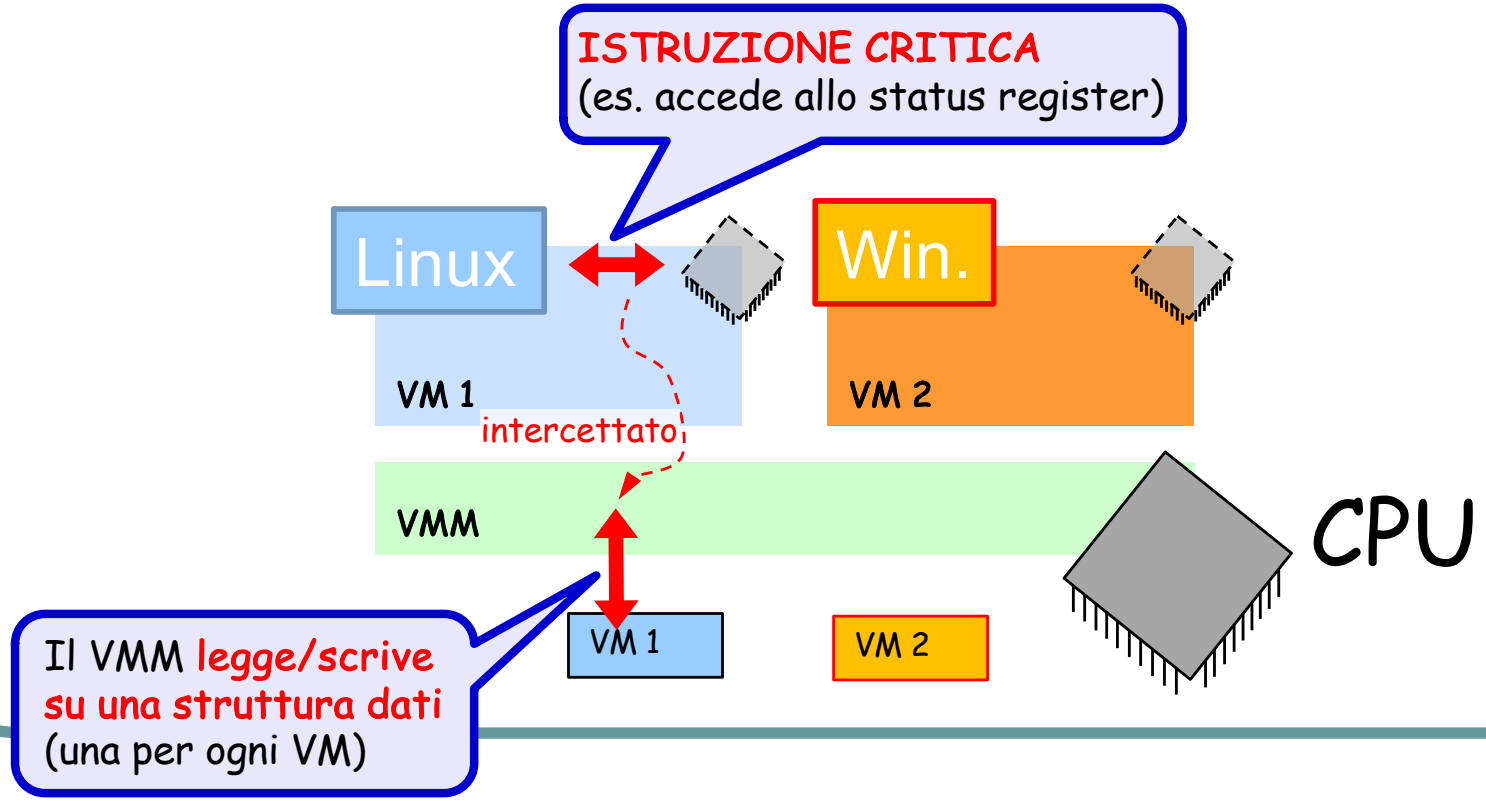
```
$ ./vmlinux
```

MMU registers ISR table I/O devices …

Se lanciassimo il binario del kernel come un programma user-space, non riuscirebbe ad eseguire!

La “virtualizzabilità” della CPU

- Il VMM espone al guest OS una **emulazione** della CPU
- Intercetta l'esecuzione delle "istruzioni critiche” (**sensitive instructions**)

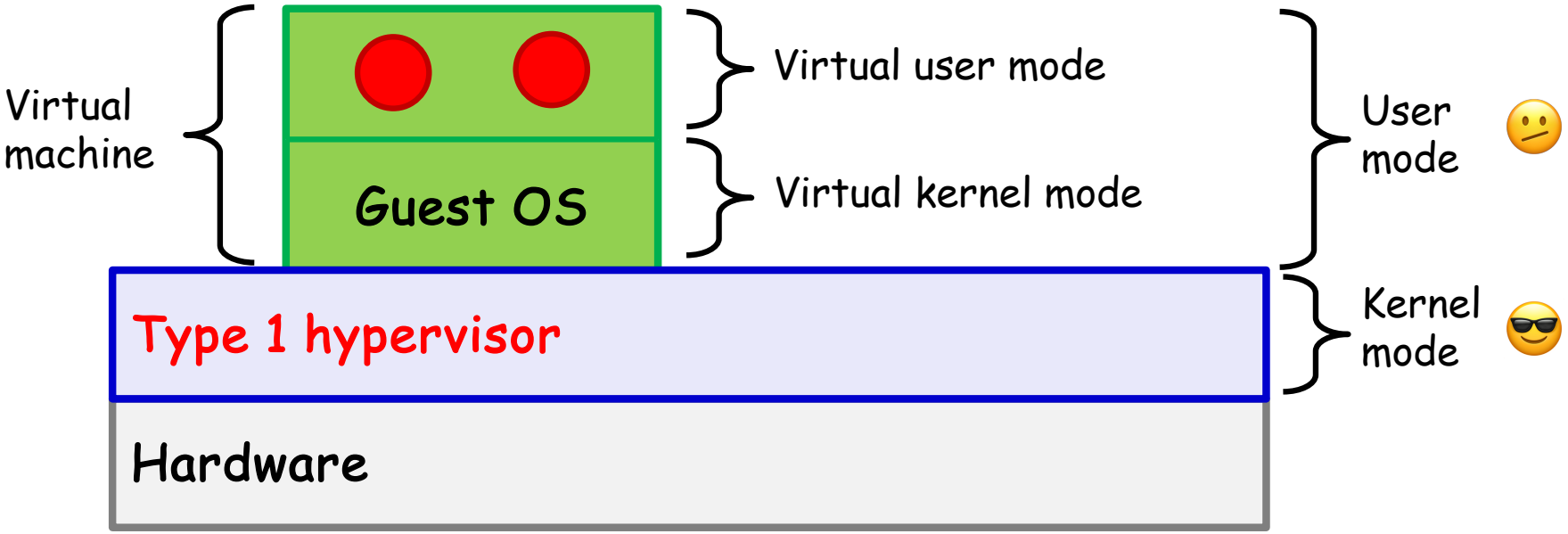


De-privileging

- Il VMM e guest OS coesistono, a diversi **livelli di privilegio della CPU (de-privileging)**

- Il VMM è in **kernel mode** (ruolo normalmente ricoperto dal SO)
- La VM è in **user mode**, dentro cui:
  - Il Guest OS è in **virtual kernel mode** e
  - Le applicazioni utente sono in **virtual user mode**

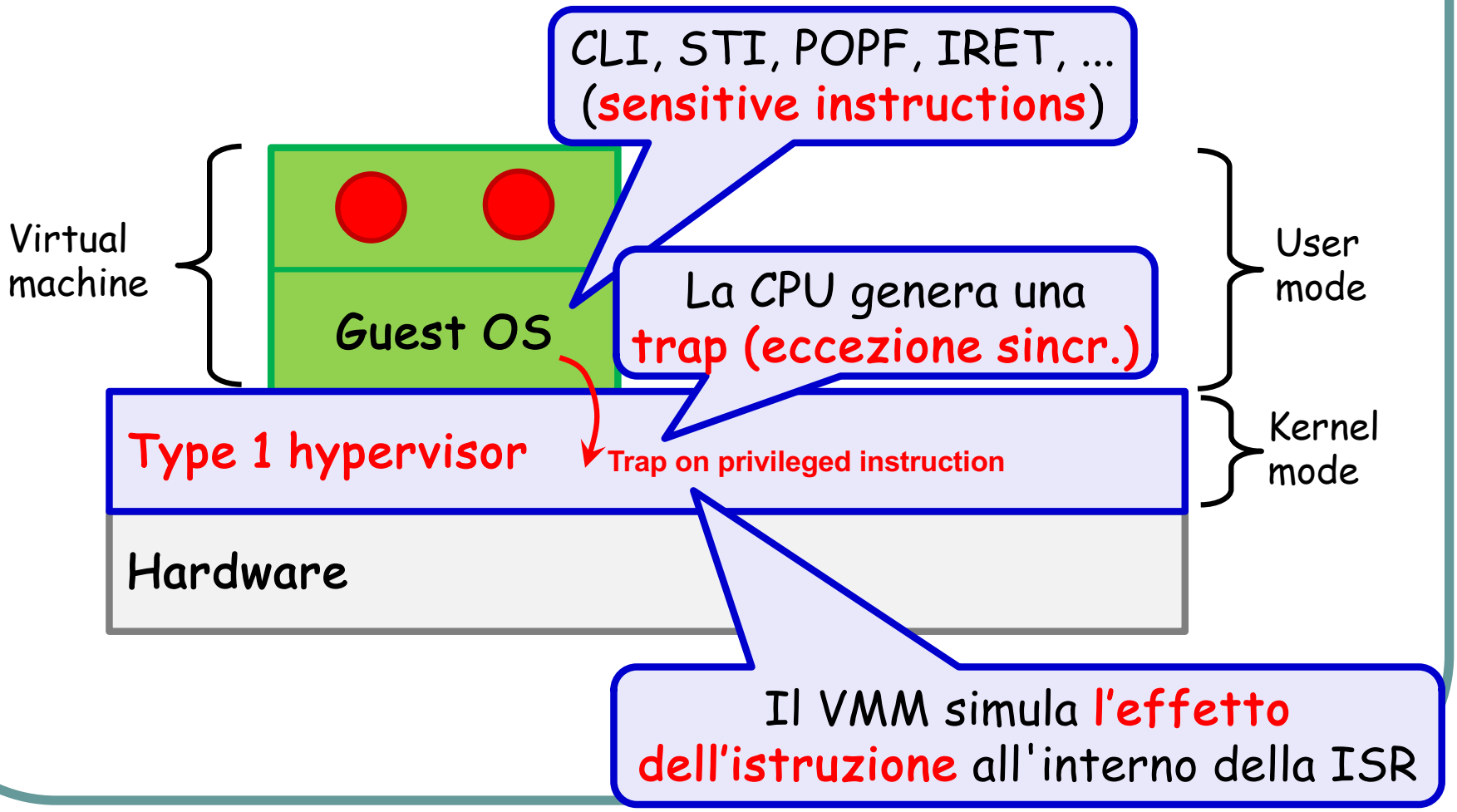
De-privileging



Trap-and-emulate

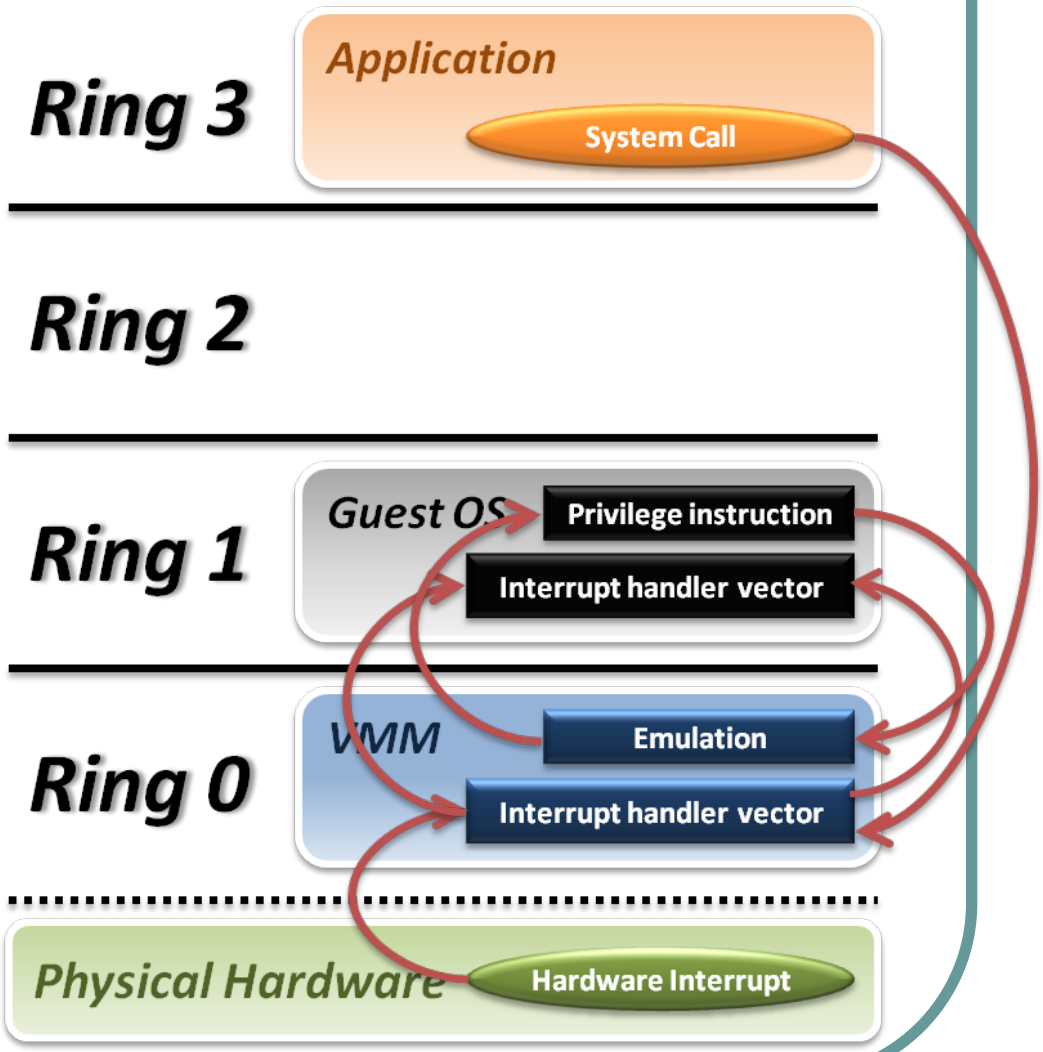
- Se il guest OS (de-privilegiato) esegue istruzioni privilegiate, la CPU genera una **trap**
- Il VMM intercetta la trap, ed emula in software gli effetti della istruzione privilegiata (**trap-and-emulate**)

Trap-and-emulate



Esempio nel caso di CPU Intel (4 livelli di privilegio, "ring")

- **System call**
  - La CPU genera una trap
  - Gestita dal VMM
  - Il VMM "salta" al guest OS
- **Interrupt hardware**
  - L'interrupt hardware innesca una ISR del VMM
  - Il VMM "salta" alla ISR del guest OS
- **Istruzioni privilegiate**
  - Le istruzioni privilegiate nel guest OS causano trap
  - Il VMM gestisce la trap ed emula l'istruzione



Esempio di trap-and-emulate: istruzione CLI ("clear interrupts")

1. Il VMM simula CLI **ponendo a 0** lo ***Interrupt Flag*** all'interno di una **struttura dati** dedicata alla VM
   - Lo *Interrupt Flag* nel **registro fisico** della CPU è inalterato!
   - Le interrupt fisiche sono ancora ricevute dallo VMM
2. Da questo momento, quando si verifica una interrupt della CPU fisica, il VMM **non invia** una interrupt al guest OS

La (non) “virtualizzabilità” in CPU x86

- L’architettura **x86** tradizionale **non è “virtualizzabile”** con solo il trap-and-emulate
- Molte delle istruzioni sensibili **non generano alcuna trap**!

  - Se il guest tenta di eseguire una istruzione sensitive, la CPU ignora l’istruzione
  - Tali istruzioni sono dette “sensibili ma non-privilegiate”

Tecniche di virtualizzazione CPU

- **Full virtualization**, senza supporto hardware
  - Utilizza tecniche software (dynamic binary translation, shadow page tables, ...)
- **Para-virtualizzazione**
  - Il guest SO è sviluppato appositamente per cooperare con il VMM
- **Full virtualization**, con **supporto hardware** (Intel VT, AMD SVM)
  - Migliori prestazioni e VMM più semplice

Virtualization & Cloud Computing

Virtualization Solutions	Cloud Providers
vmware Full Virtualization (hw assisted, sw)	terremark, CONTEGIX, bluemile, TIER 3, bluelock., DATAPIPE
Xen™ Para-Virtualization	amazon web services™, rackspace HOSTING, SOFTLAYER, CARPATHIA HOSTING, INC., GOGRID
KVM Full Virtualization (hw assisted)	IBM, DATAPIPE, Joyent, ElasticHosts Flexible servers in the cloud

Hypercall e DBT a confronto

OS source code

```
…
…
…
val = store_idt()
…
…
…
```

OS binary

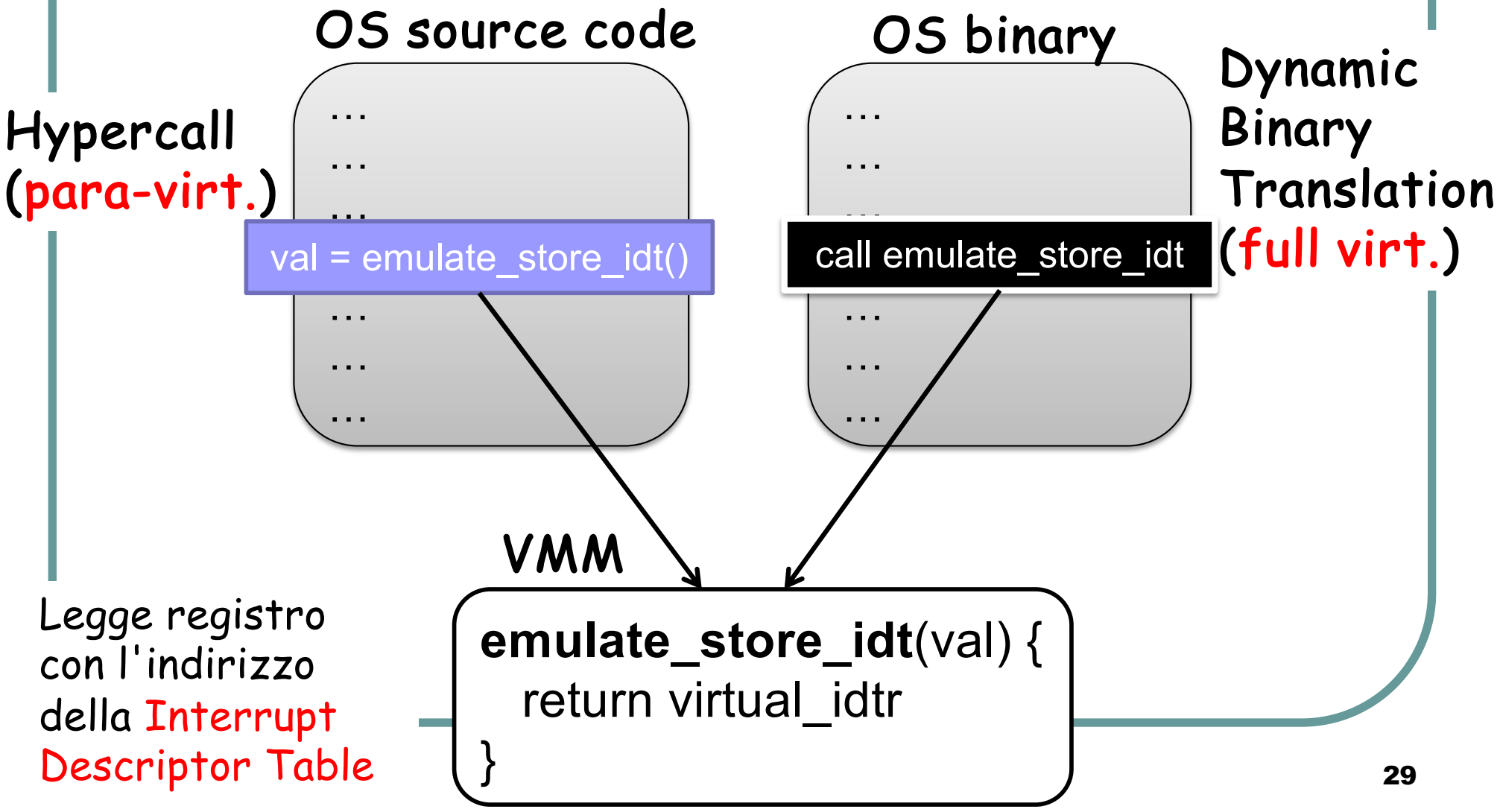
```
…
…
…
mov val, idtr
…
…
…
```

VMM

```
emulate_store_idt(val) {
   return virtual_idtr
}
```

Legge registro con l'indirizzo della Interrupt Descriptor Table

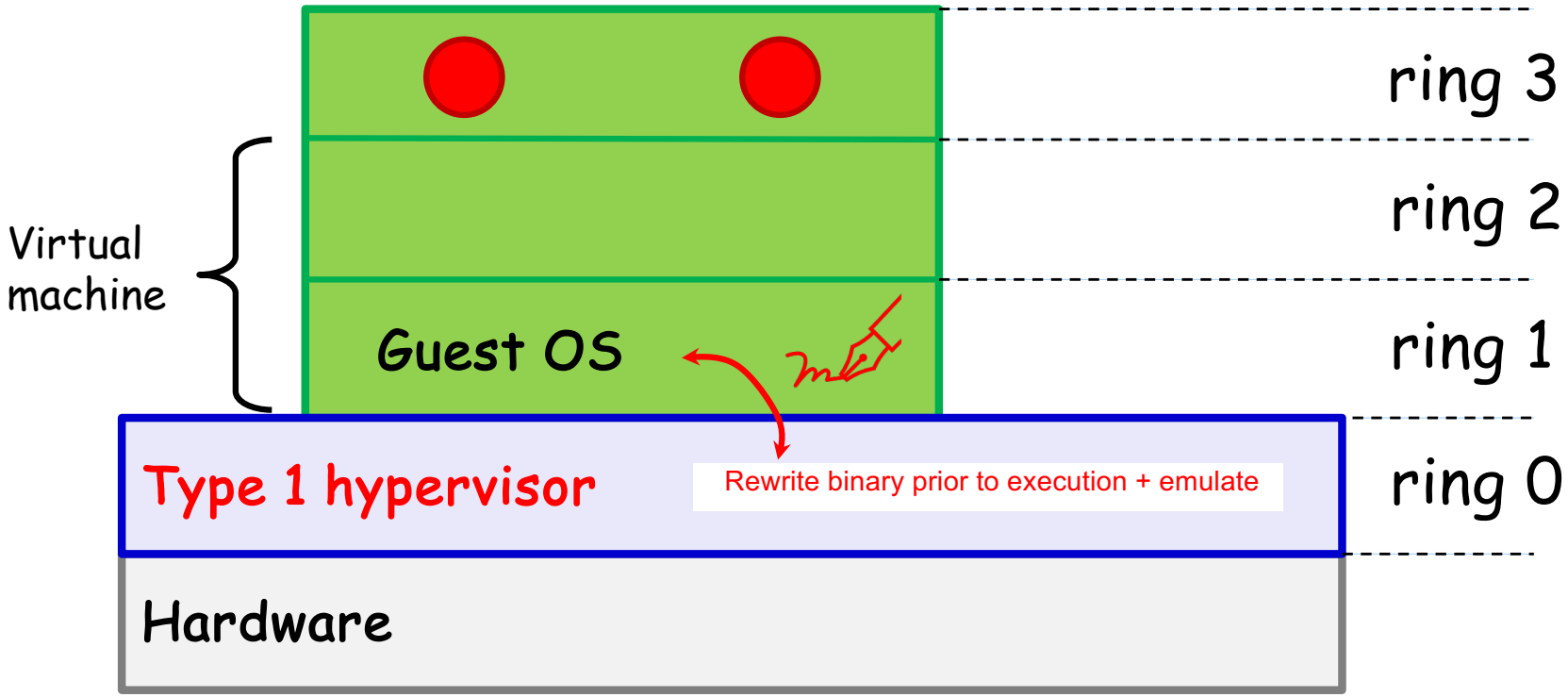
Hypercall e DBT a confronto



Full virtualization, no supporto hw

- Nel 1999, l'hypervisor di **VMware** introdusse tecniche efficienti di full-virtualization per Intel x86
- Il **codice binario** del guest OS viene "**riscritto al volo**" dal VMM, prima di venire eseguito
- Sostituisce le istruzioni sensibili con codice di emulazione

Full virtualization, no supporto hw



Dynamic binary translation

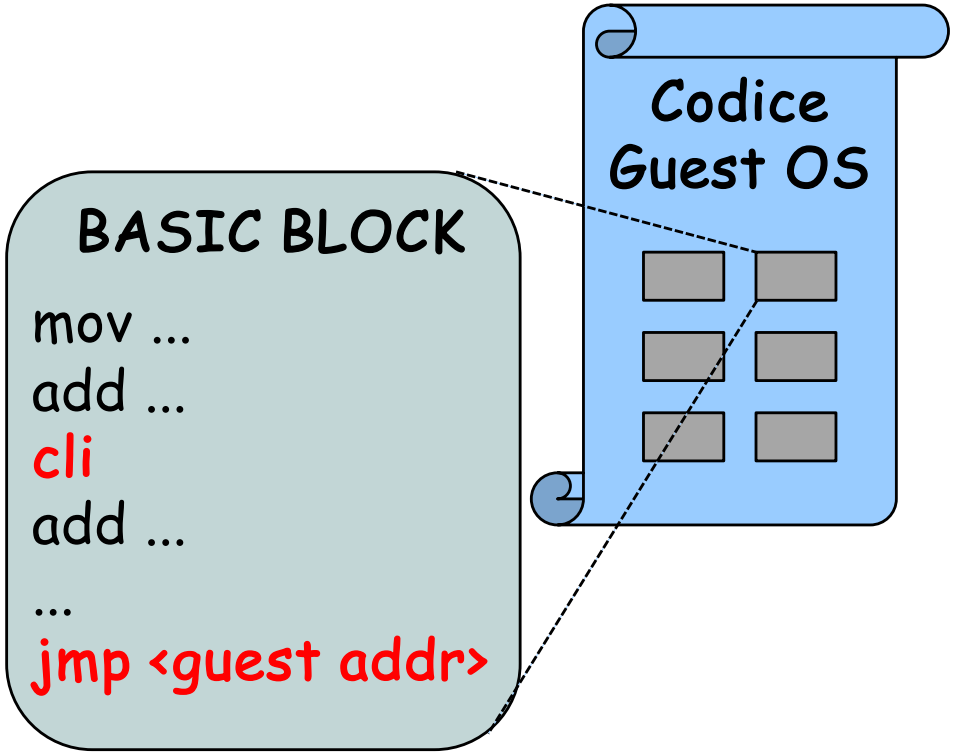
- Sin dall'avvio della VM, il VMM analizza **a blocchi** il codice eseguito dal guest OS
- Il VMM **riscrive** le eventuali istruzioni privilegiate con codice di emulazione
- Inoltre, il **salto finale** viene sostituito con una chiamata al VMM, in modo che **possa avanzare il processo di traduzione** al prossimo blocco

BASIC BLOCK

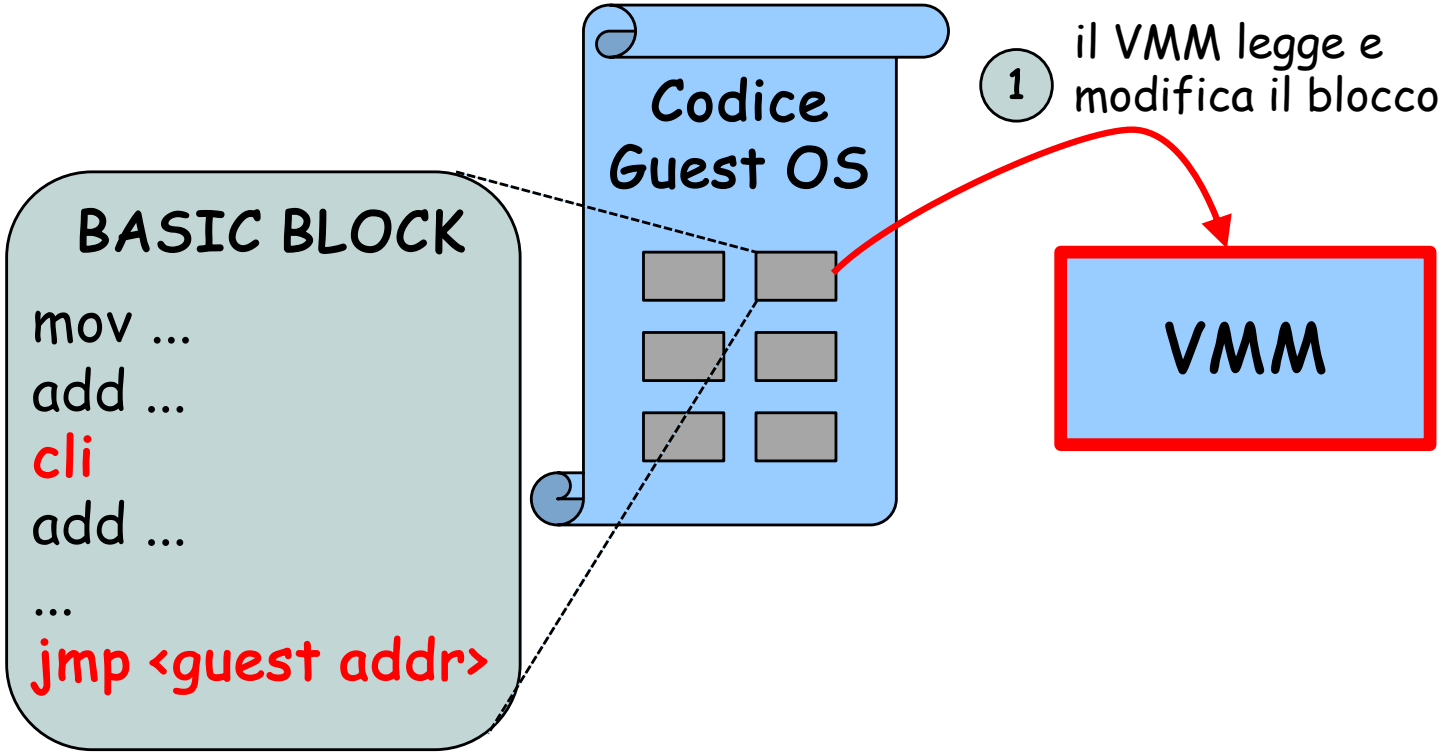
```
mov ...
add ...
cli
add ...
...
jmp <guest>
```

*Un blocco è una breve **sequenza di istruzioni sequenziali** che termina con una **istruzione di salto***

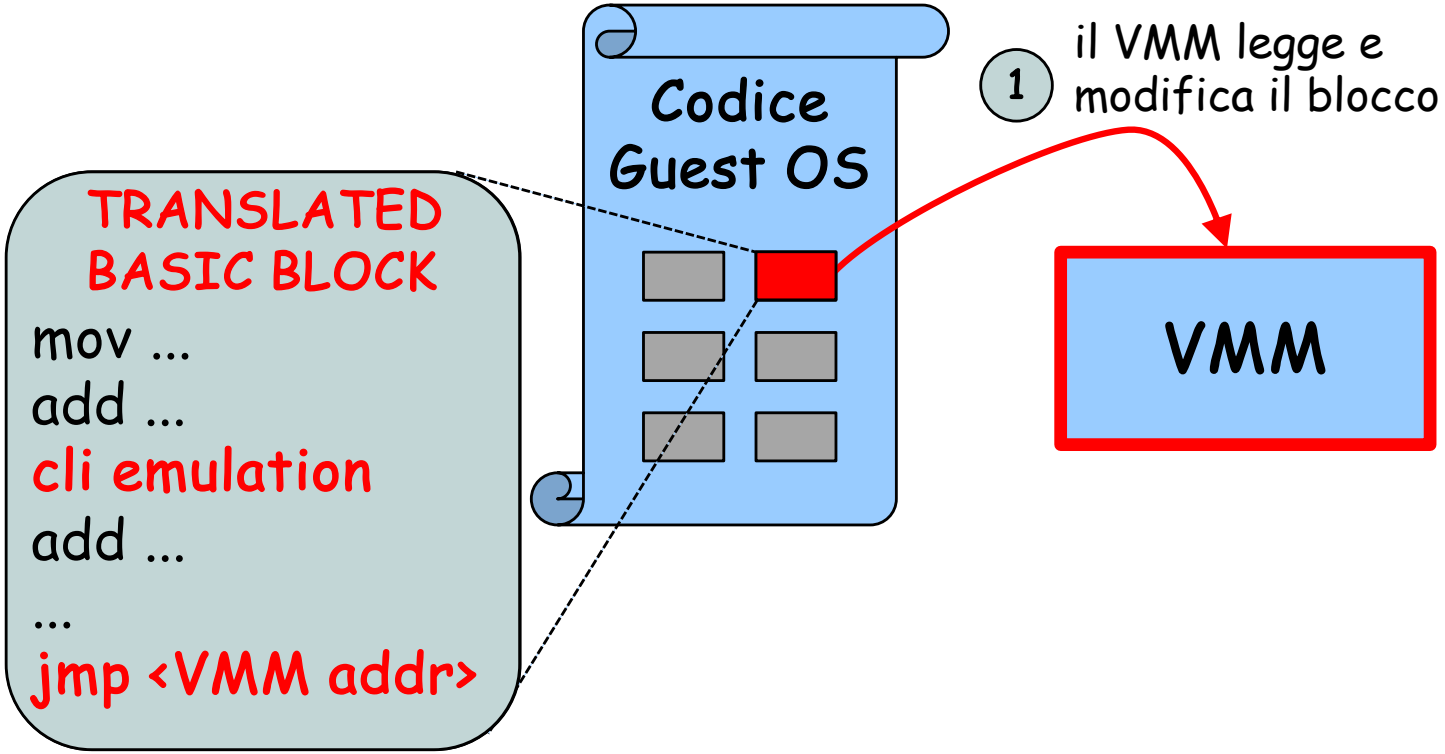
Dynamic binary translation



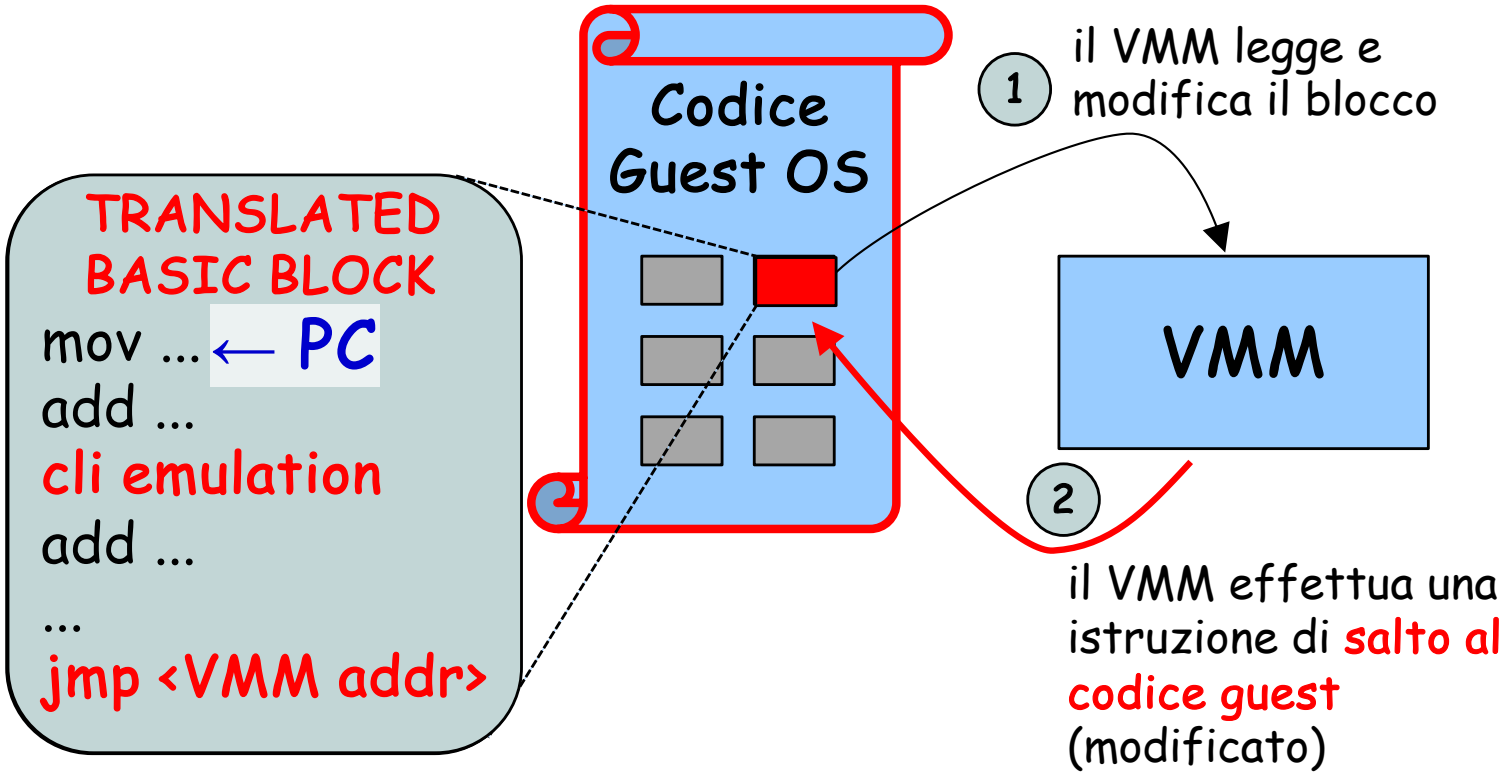
Dynamic binary translation



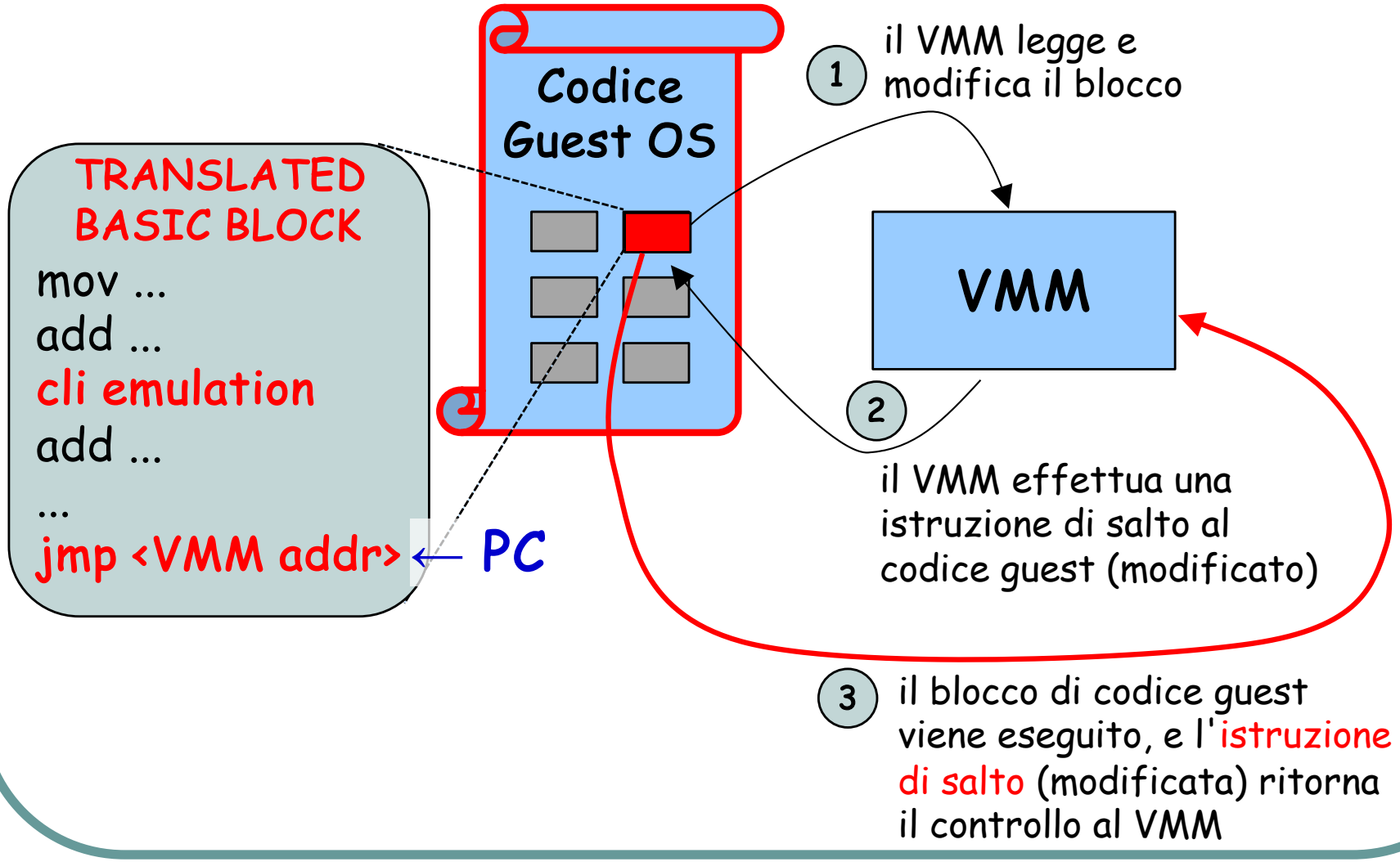
Dynamic binary translation



Dynamic binary translation



Dynamic binary translation



Para-virtualizzazione

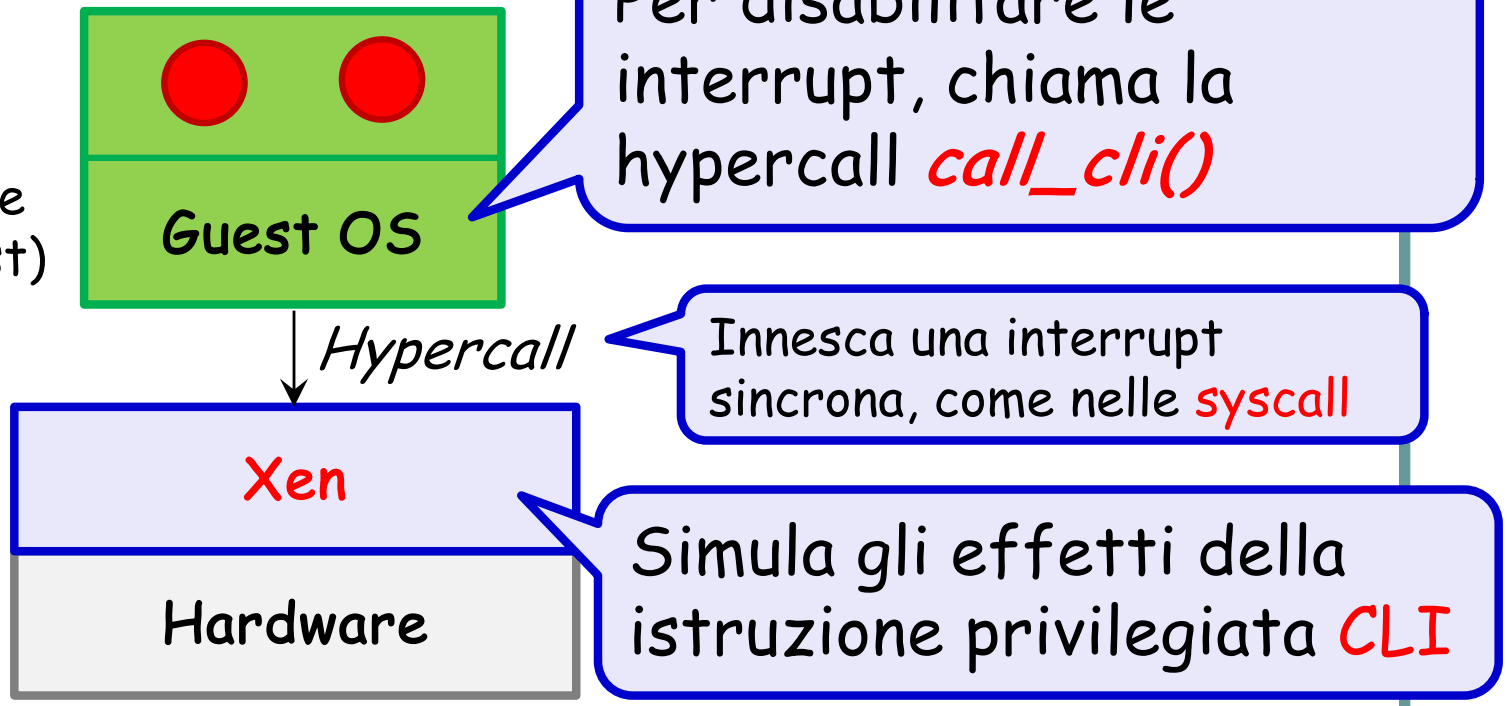
- Le istruzioni sensibili nel guest OS sono sostituite da hypercalls
- Le hypercalls sono parte dallo VMM
- Simili a system calls, ma chiamate dal guest OS!

A differenza della full virtualization, si modifica il codice sorgente del guest OS, non il codice binario.

La modifica è fatta dal **programmatore**, non dal VMM.

Para-virtualizzazione

Guest OS
"riscritto" per para-virtualizzazione (es. Ubuntu PV Guest)



Para-virtualizzazione

Il guest OS così modificato **non può eseguire sull'hardware fisico**

Può eseguire solo in combinazione con il **VMM**

- es. "Ubuntu per Xen hypervisor"
- Approccio non applicabile ai sistemi **legacy** oppure **proprietari**, come Windows

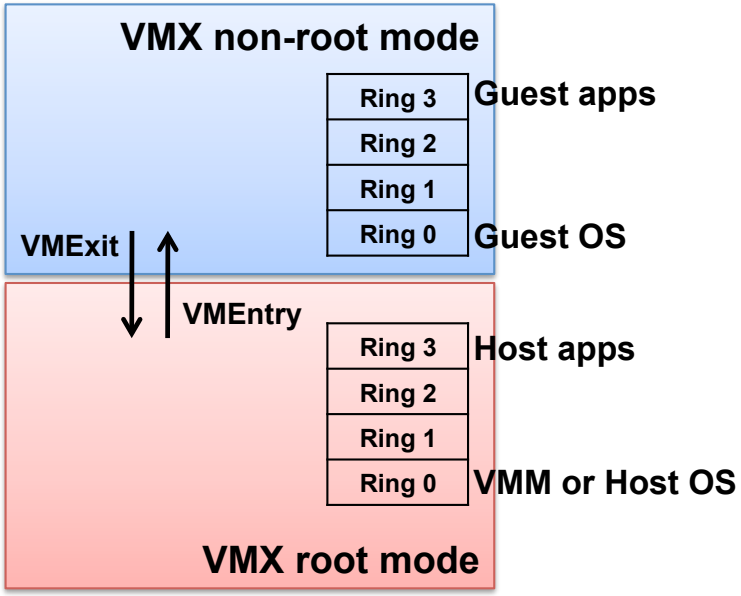
es. https://help.ubuntu.com/community/Xen#Creating_vms

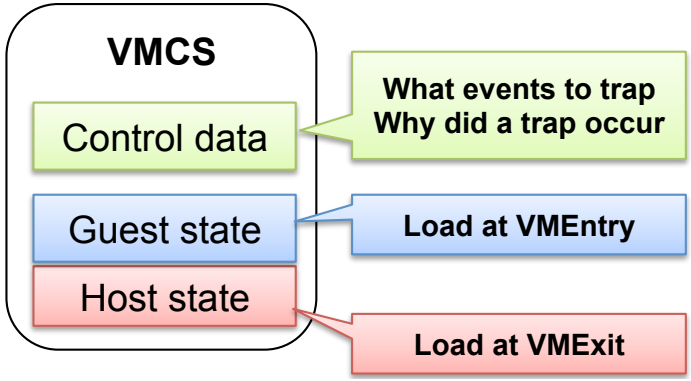
Supporto hardware per la virtualizzazione della CPU

- Le CPU Intel VT introducono:
  - Due modalità di esecuzione **(VMX root/non-root)**
  - **VMCS** (VM Control Structure)

- **Semplificano** il codice del VMM, buona parte della virtualizzazione è fatta in hardware
- Evita il **ring de-privileging** del guest OS
- Maggiore **efficienza** del cambio di contesto tra VM e VMM
- Garantisce il **meccanismo di trap** per tutte le istruzioni critiche

Supporto hardware per la virtualizzazione della CPU



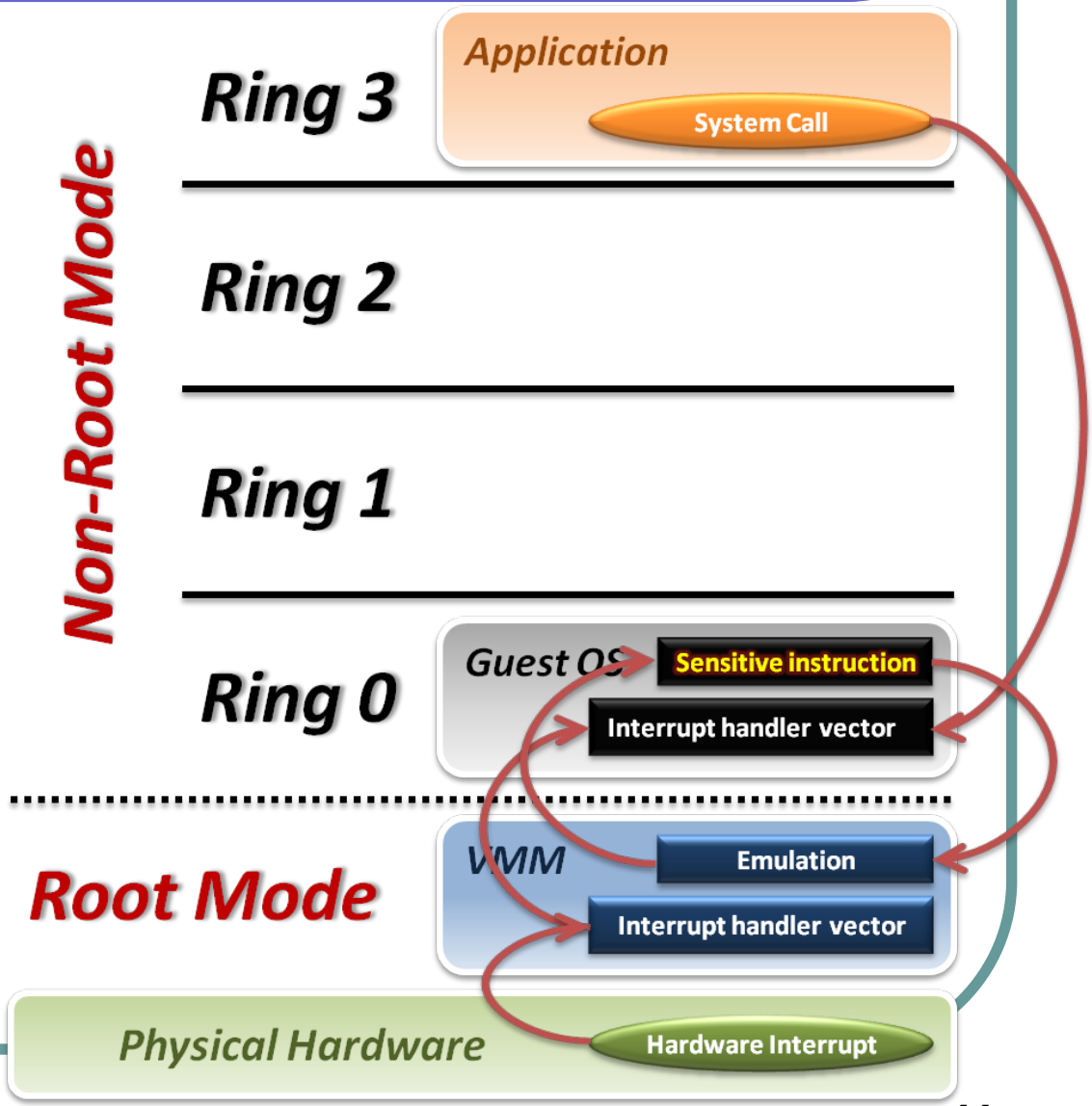


Struttura dati
(una per VM/CPU virtuale)

- **VMX root** mode: esecuzione del **VMM**
- **VMX non-root** mode: esecuzione del **guest OS**
  - Il comportamento delle istruzioni sensibili è **configurato dal VMM**
- **VM-exit**, **VM-enter**: eventi che causano il passaggio root/non-root (es., timer, page fault, ...)

Supporto hardware per la virtualizzazione della CPU

- **System call**
  - Trap gestita direttamente dal guest OS
- **Interrupt hardware**
  - Gli eventi hardware sono sempre gestiti dal VMM, ma con ottimizzazioni
- **Istruzioni privilegiate**
  - Nessun bisogno di riscrittura del guest OS
  - Tutte le istruzioni causano trap (in modo configurabile)



VM Control Structure (VMCS)

- Il VMM usa la VMCS per indicare alla CPU le condizioni e le azioni di **VM-entry** e **VM-exit**
- Lo **stato della CPU** è salvato/caricato in **hw** nella VMCS

VM-execution controls	Determines what operations cause VM exits	CR0, CR3, CR4, Exceptions, IO Ports, Interrupts, Pin Events, etc.
Guest-state area	Saved on VM exits Reloaded on VM entry	EIP, ESP, EFLAGS, IDTR, Segment Regs, Exit info, etc.
Host-state area	Loaded on VM exits	CR3, EIP set to monitor entry point, EFLAGS hardcoded, etc.
VM-exit controls	Determines which state to save, load, how to transition	Example: MSR save -load list
VM-entry controls	Determines which state to load, how to transition	Incl uding injecting events (interrupts, exceptions) on entry

VM-exit

- Esempi di cause di VM-exit:
  - Istruzioni sensibili
    - **CPUID**: riporta le CPU capabilities
    - **RDMSR**, **WRMSR**: legge/scrive i “model-specific registers”
    - **INVLPG**: invalida entry TLB
    - **RDPMC**, **RDTSC**: legge i registri di perf. monitoring e di timestamp
    - **HLT**, **MWAIT**, **PAUSE**: disattivazione del guest OS
    - **VMCALL**: nuova istruzione per invocare il VMM
  - Accessi a stato sensibile
    - **MOV DRx**: accessi ai debug register
    - **MOV CRx**: accessi ai control register
    - **Task switch**: accessi al CR3 (puntatore alla tabella della pagine)
  - Eccezioni ed eventi asincroni
    - Page fault, debug exceptions, interrupts, etc.

VM-entry

- Il VMCS consente di “iniettare” eventi (interrupt, eccezioni) nella VM, al momento del VM-entry
- L’iniezione è fatta dalla CPU in hardware, semplificando il VMM e migliorando le prestazioni

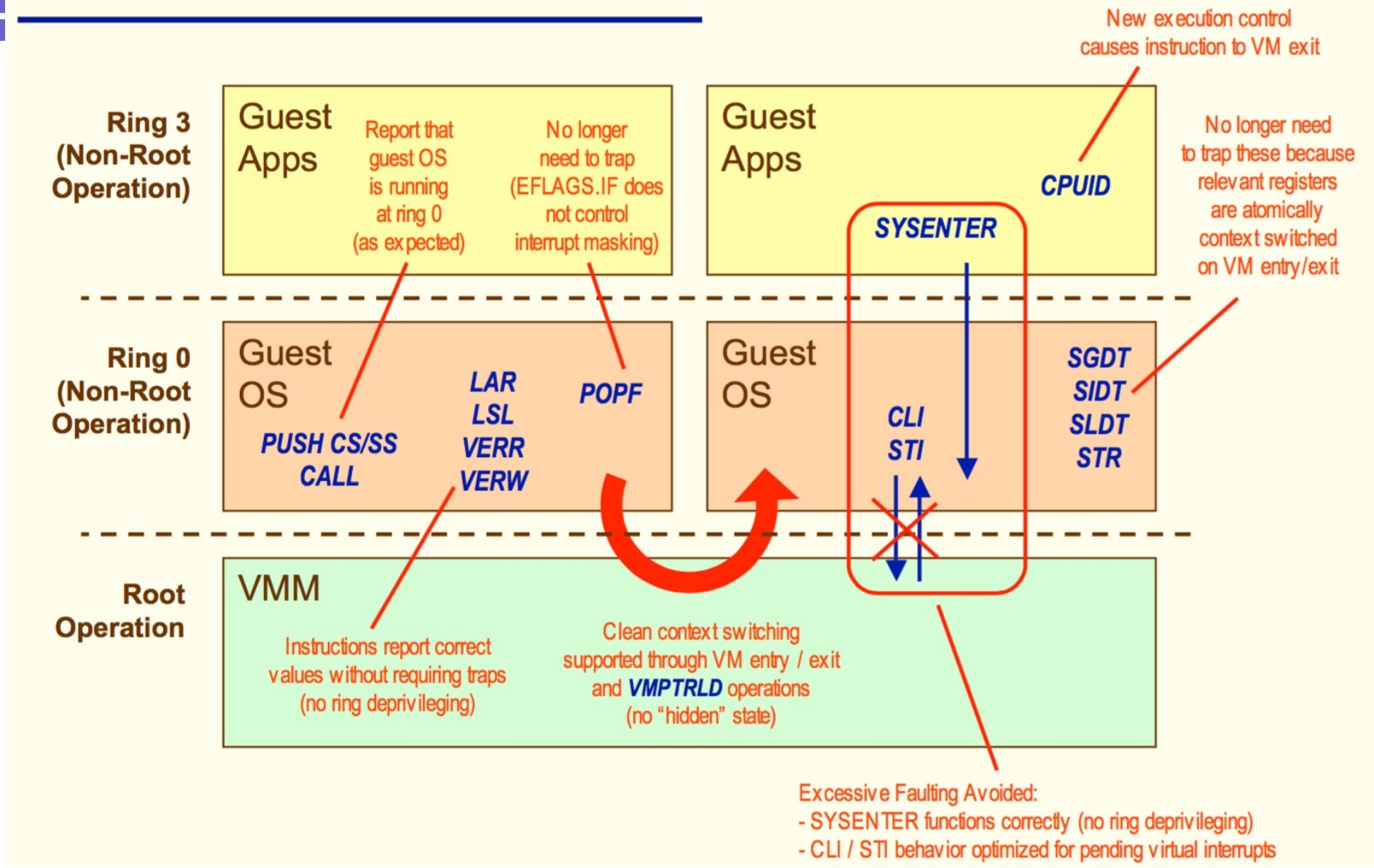
Esempio: Page faults

1. La CPU scrive nel VMCS le cause dell’eccezione
2. Il VMM sceglie se **gestire il page fault**, in modo trasparente al guest OS (es., paginazione a domanda), ...
3. ... oppure iniettare l’eccezione nel guest OS, utilizzando il campo VM-entry controls nel VMCS

Esempio: Interrupt masking

1. Il guest OS usa istruzioni CLI/STI (caso frequente)
2. La CPU apre/chiude la “interrupt window” della VM, **senza innescare** il VMM con delle VM-exit
3. Quando si verifica un interrupt, il VMM “accoda” l’interrupt alla VM
4. La CPU inietterà automaticamente l’interrupt alla apertura della interrupt window

How VT-x Closes Virtualization Holes

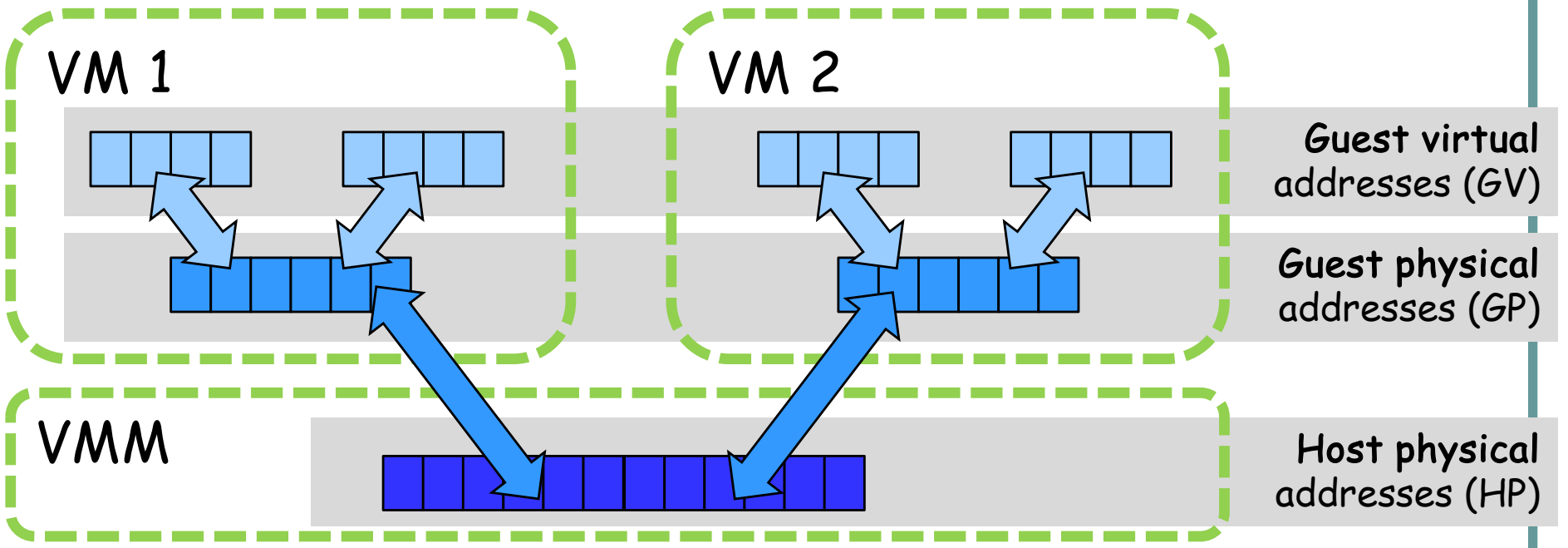


VIRTUALIZZAZIONE DELLA MEMORIA

Virtualizzazione della memoria

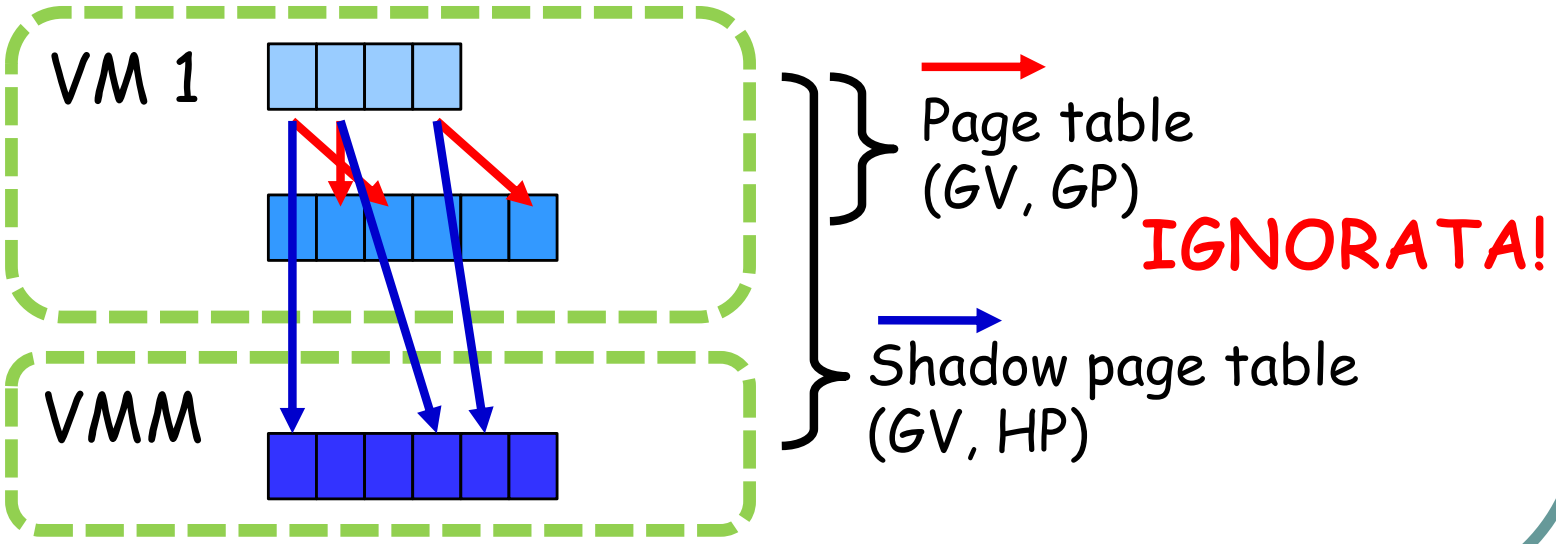
- All'interno di una VM, il guest OS gestisce la **"finta" memoria fisica della VM**
- Il VMM amministra la **"vera" memoria fisica**, anche esso con paginazione a domanda, swapping, algoritmi di prelazione, ...
- Le VM sono **isolate** e non possono accedere alla memoria di altre VM (come per i processi nei SO)

Indirizzi virtuali e fisici



Shadow page tables

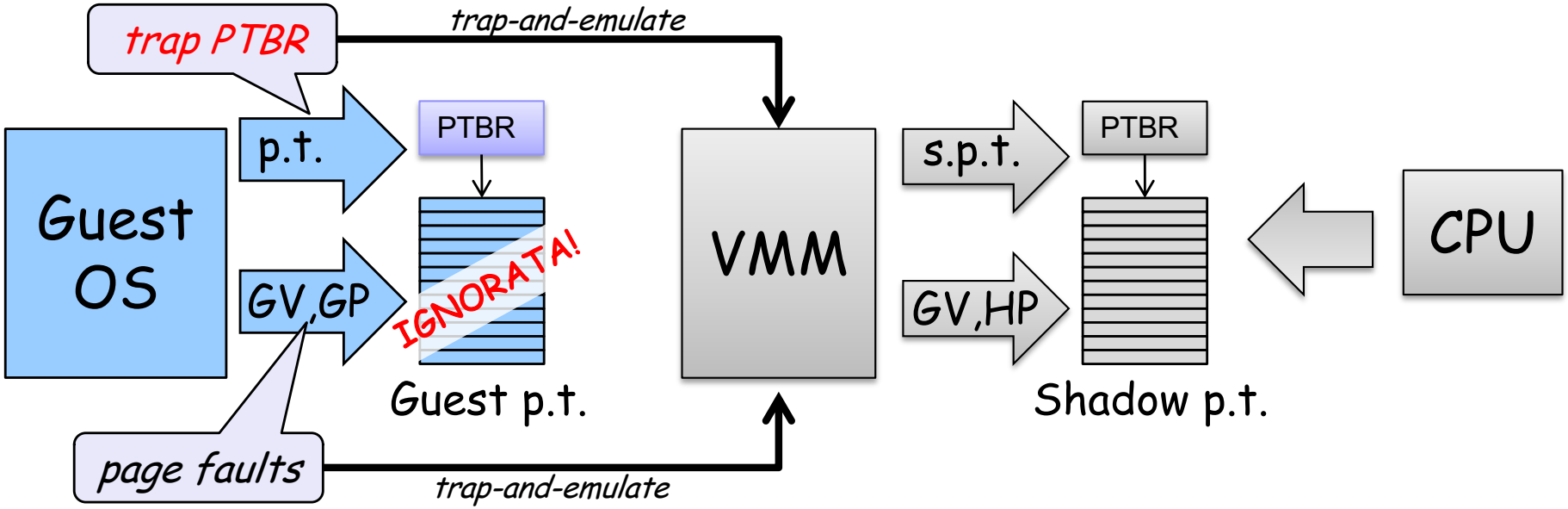
- **Tecnica software** per la virtualizzazione della memoria
- Il VMM intercetta le modifiche alle page table nel guest, creando una loro controparte detta **shadow page table**
- La CPU usa la shadow page table, **ignorando la page table del guest OS**



Shadow page tables

1. Il guest OS scrive, con una **istruzione sensitive**, nel registro (virtuale) PTBR l'indirizzo della p.t.
2. Con trap-and-emulate, il VMM carica nel registro (fisico) PTBR l'indirizzo della **shadow page table**
3. Il VMM intercetta le modifiche alla p.t. (le pagine della p.t. sono **read-only**, in modo da causare **page fault**)
4. La **CPU usa la shadow page table** per tradurre gli indirizzi virtuali della VM (la guest p.t. è **ignorata**)

Shadow page tables

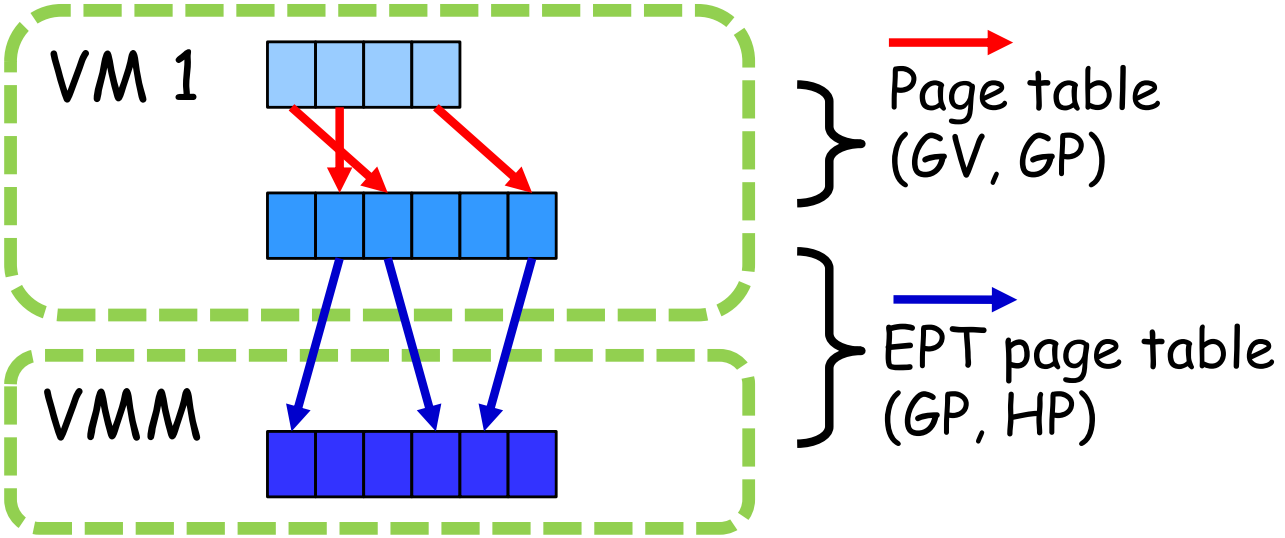


Shadow page tables

- Le shadow page tables sono un meccanismo di virtualizzazione oneroso, poiché aumentano **il volume e l'overhead dei page fault**
  - **Hypervisor-induced page faults**: Page faults introdotti dall'hypervisor per aggiornare le shadow p.t.
  - **Guest-induced page faults**: Page faults "genuini" prodotti dalla VM. Sono intercettati dal VMM e "re-iniettati" nella VM.

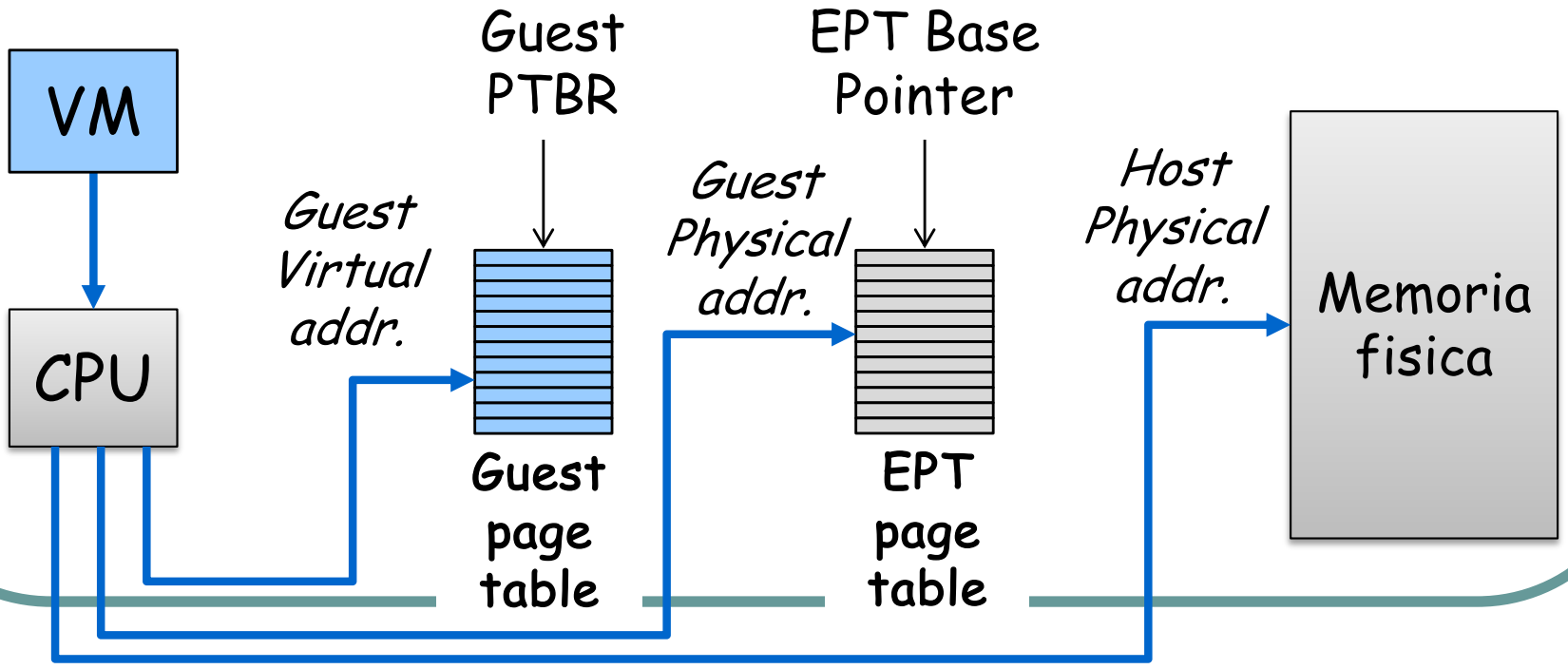
Supporto hardware per virtualizzazione della memoria

- Intel VT introduce un livello aggiuntivo di traduzione degli indirizzi mediante **Extended Page Tables (EPT)**, gestite dal VMM
- La MMU **accede in hardware** ad entrambe le tabelle
- Si evitano hypervisor-induced page faults



Traduzione degli indirizzi

1. La CPU accede le guest page table del guest OS (da **guest virtual addr.** a **guest physical addr.**)
2. La CPU accede le extended page table del VMM (da **guest physical addr.** a **host physical addr.**)



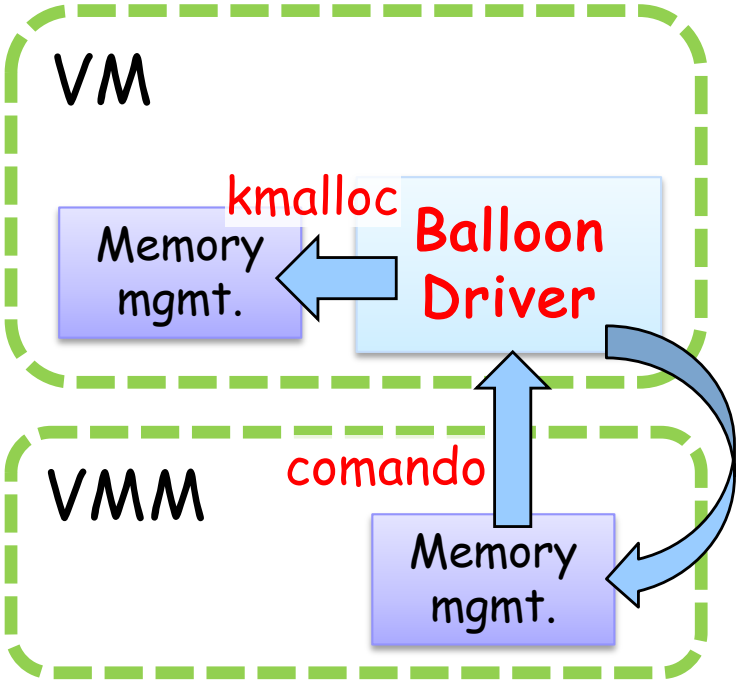
Prelazione delle pagine (ballooning)

- L'**algoritmo di sostituzione** del VMM può entrare in conflitto con l'algoritmo di sostituzione del guest OS
  - non conosce quali pagine il guest OS ritiene "superflue"
- La **tecnica del balooning** permette la cooperazione tra guest e VMM

```
so@Ubuntu-SO: ~
so@Ubuntu-SO:~$ lsmod | grep -i vm
vmw_balloon            24576  0
vmw_vsock_virtio_transport_common    40960  1 vsock_loopback
vmw_vsock_vmci_transport    32768  2
vsock                  49152  7 vmw_vsock_virtio_transport_common,vsoc
vmw_vmci               90112  2 vmw_balloon,vmw_vsock_vmci_transport
```

Prelazione delle pagine (ballooning)

1. Il guest OS esegue un **modulo kernel** (**baloon driver**), che riceve comandi dal VMM
2. Quando il VMM ha bisogno di allocare nuove pagine, invia al baloon un comando di "espansione"
3. Il baloon **alloca memoria** all'interno della VM, forzando il guest OS a fare **swap-out delle pagine superflue**
4. Il VMM utilizza le pagine allocate dal baloon



Virtualizzazione dell’I/O

Virtualizzazione dell'I/O

- Le **operazioni di I/O** sono "virtualizzabili" con *trap-and-emulate*
- Si **simula un dispositivo virtuale**

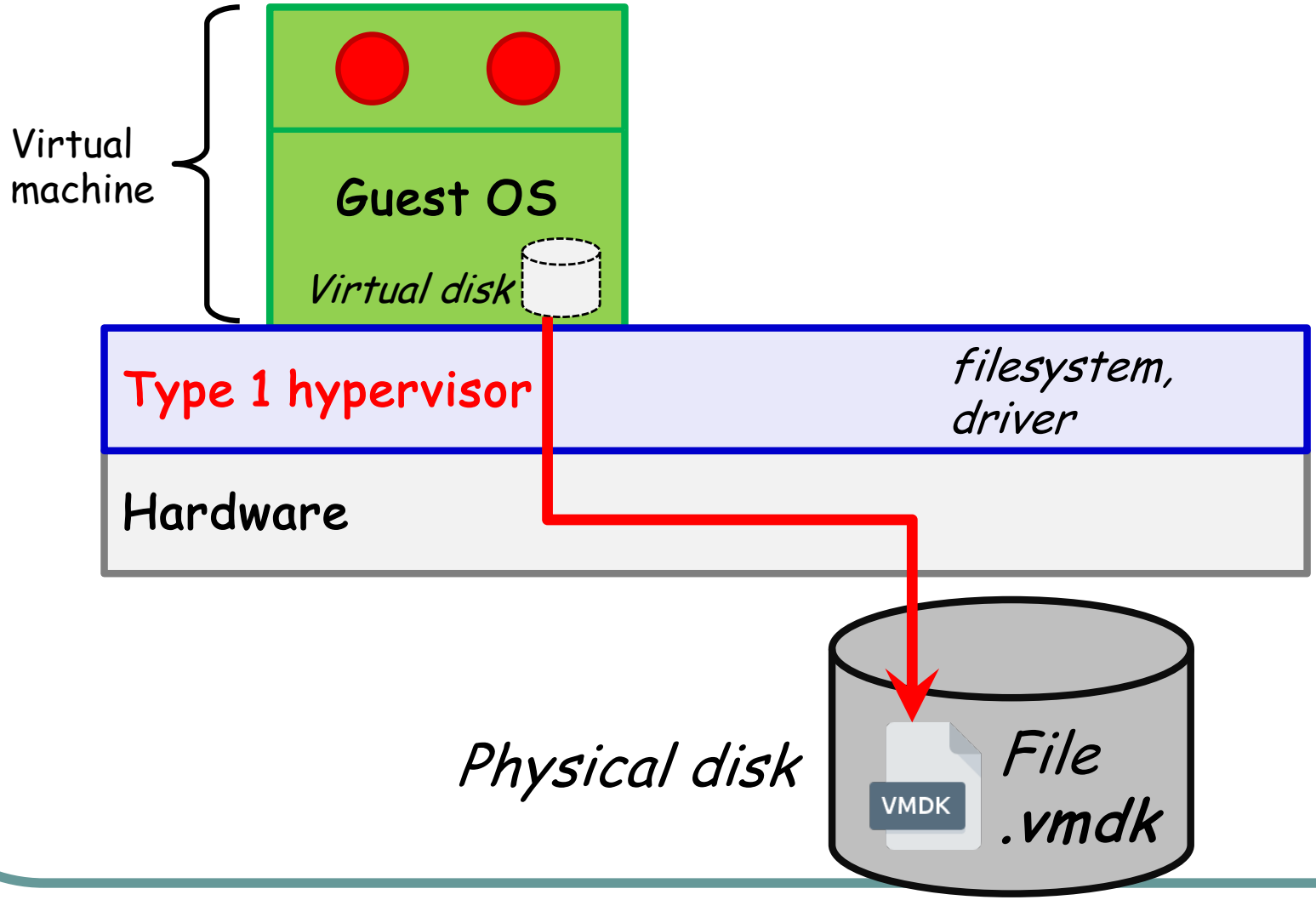
- **memory-mapped I/O**: intercetta ***accessi agli indirizzi del dispositivo*** tramite page fault

- **port-based I/O**: intercetta le ***istruzioni sensibili*** di I/O

Dispositivi virtuali

- **Disco virtuale**
  - il VMM riceve il comando di lettura/scrittura
  - Opera su un file dedicato (es. **.VMDK**) che conterrà i dati del disco virtuale
- **NIC virtuale**
  - il VMM riceve un pacchetto
  - Lo inoltra alla rete fisica o virtuale (**virtual switch**)

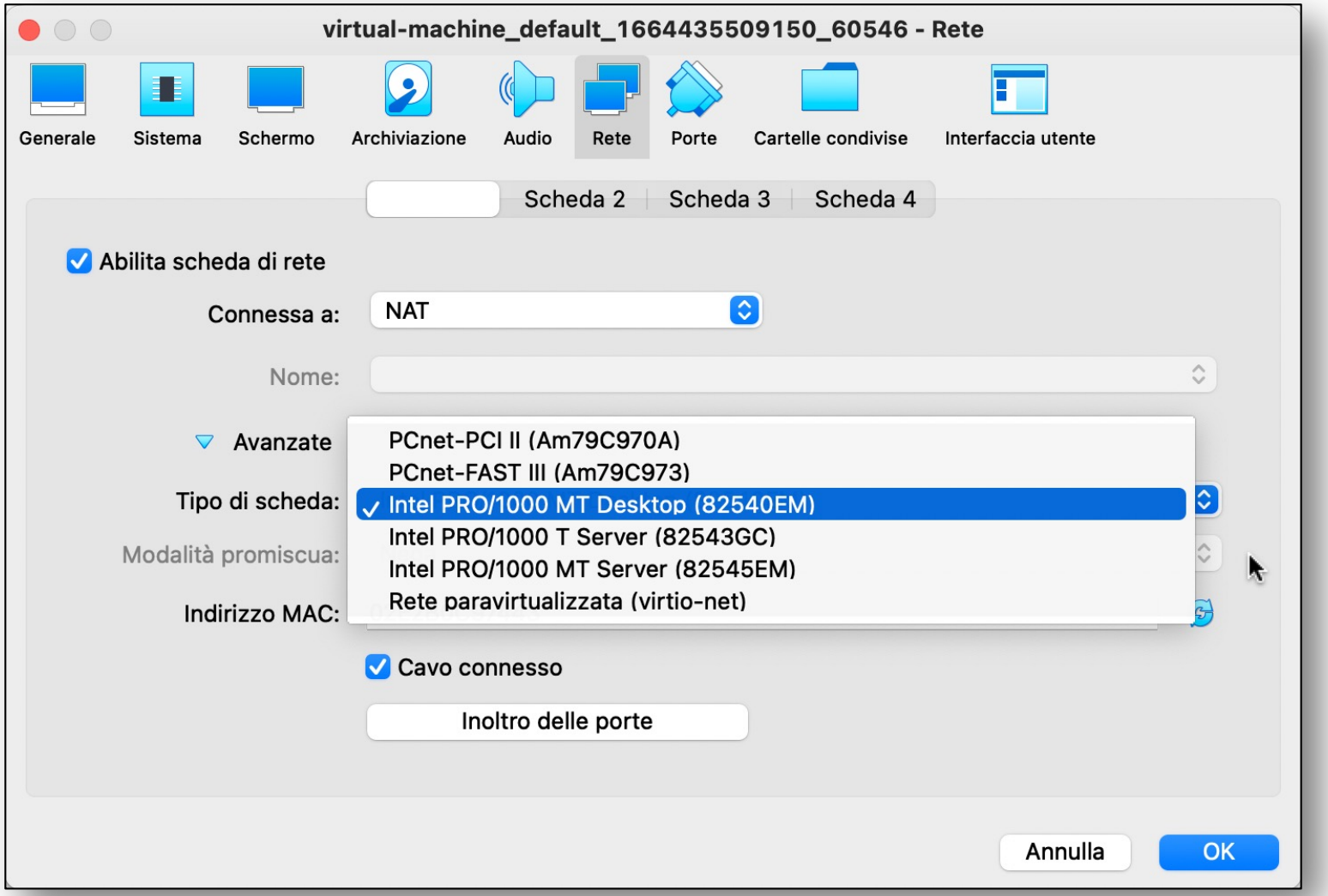
Virtualizzazione dell'I/O



Virtualizzazione dell'I/O

- Varie tecniche:
  - I/O full virtualization
  - I/O para-virtualization
  - I/O passthrough
  - SR-IOV

Virtualizzazione dell'I/O



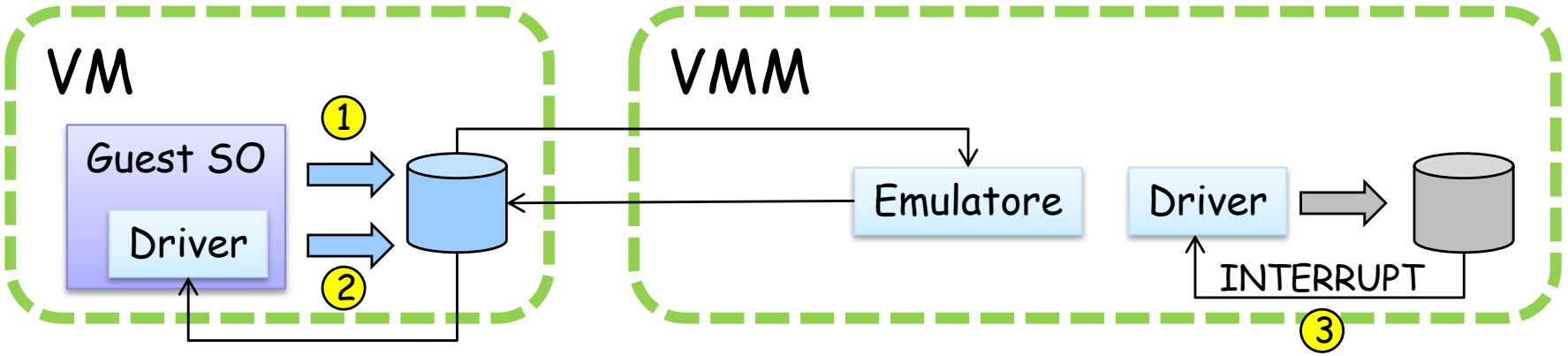
I/O full virtualization

- La **I/O full virtualization** simula in software un dispositivo reale
- Sono spesso simulati **dispositivi semplici e ben noti** (es. Ethernet *E1000*)
- Le **prestazioni sono penalizzate** dalla emulazione e dal passaggio attraverso il VMM

I/O full virtualization

1. Il guest OS **interroga il bus** per trovare i dispositivi connessi; il VMM intercetta ed indica il modello di dispositivo virtuale
2. Il guest OS tenta di **leggere/scrivere** i registri del dispositivo virtuale; il VMM intercetta e copia i valori da/verso i registri hardware
3. Il dispositivo hardware genera una **interruzione**; il VMM la serve simulando una interruzione nel contesto della VM

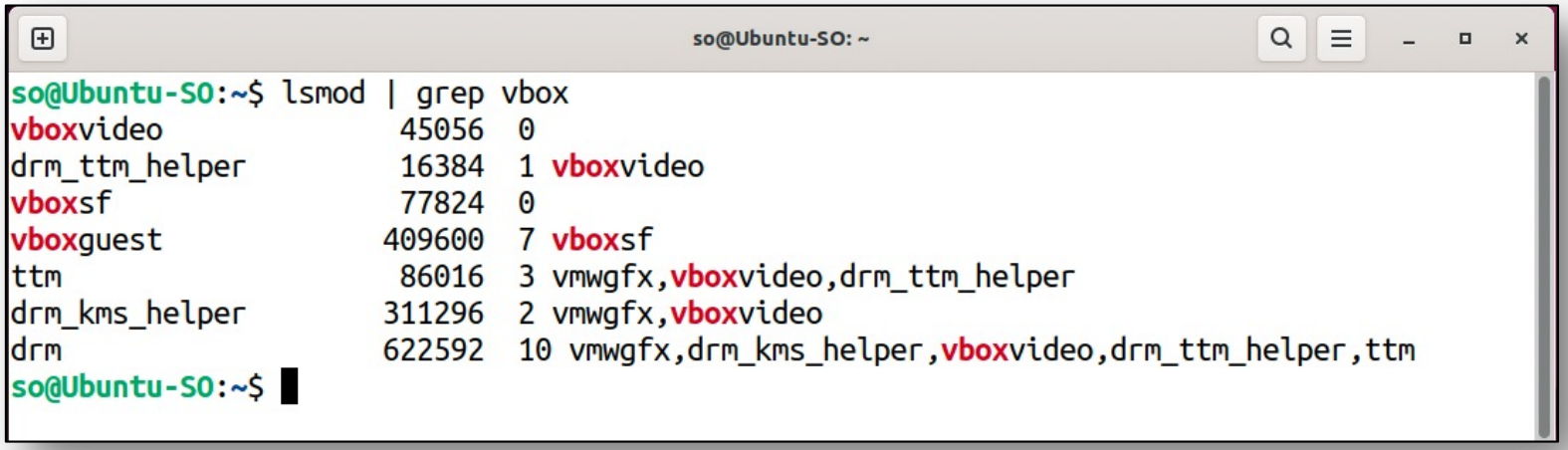
I/O full virtualization

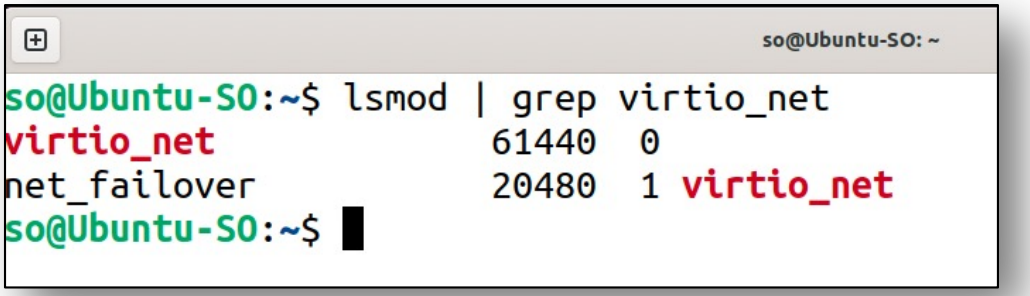


I/O paravirtualization

- La **I/O paravirtualization** è l'analogo della paravirtualization della CPU
- Il guest OS dialoga con il VMM tramite **hypercall**
- Evita la complessità e il rallentamento dovuti alla simulazione di un dispositivo

I/O paravirtualization

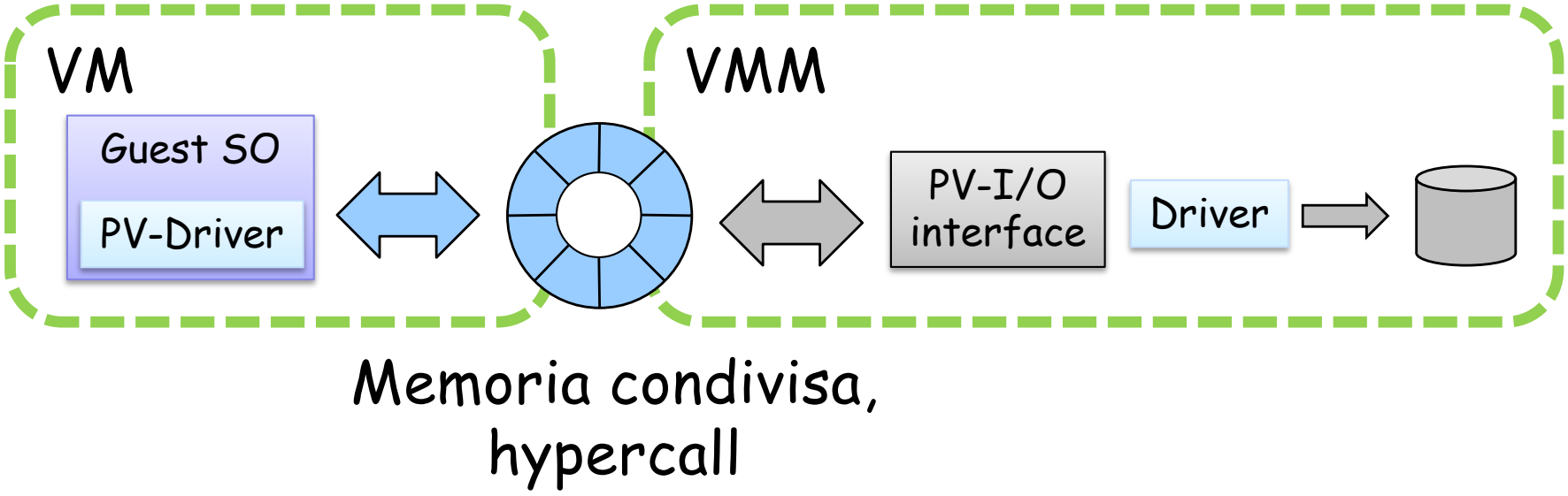




I/O paravirtualization

- Il VMM esporta al guest OS delle aree di **memoria condivisa** e **hypercall** per l'I/O
- Il guest OS è "consapevole" della virtualizzazione; esso carica un driver apposito per l'I/O paravirtualizzato

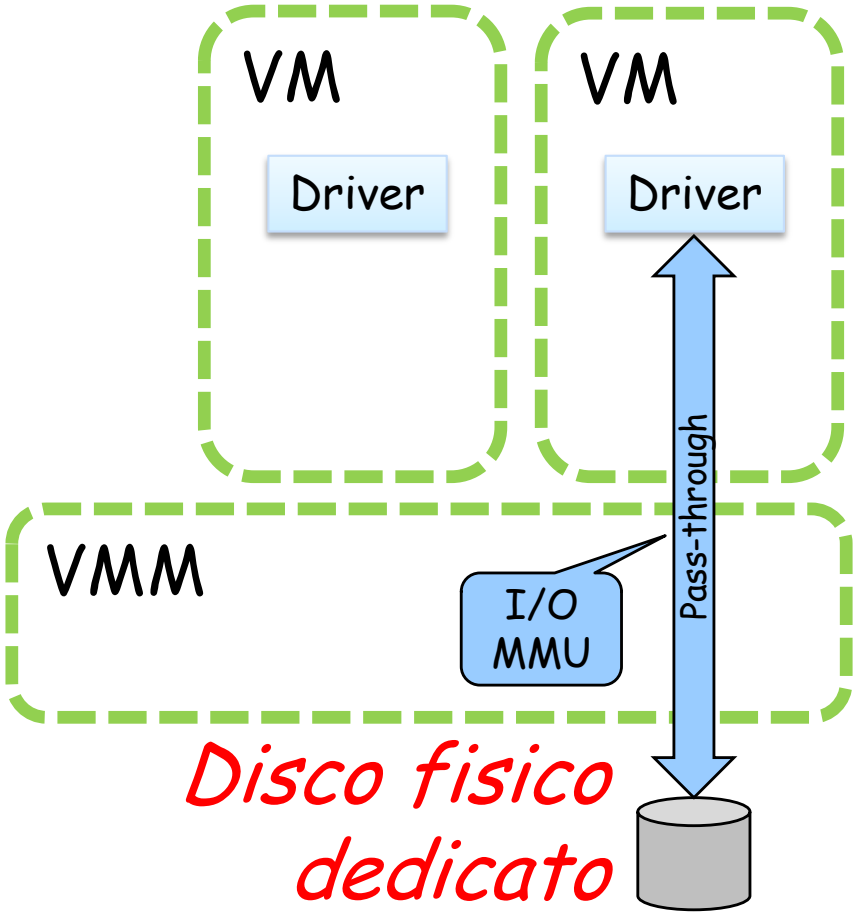
I/O paravirtualization



Device pass-through e I/O-MMU

- Il **device pass-through** assegna un dispositivo di I/O in **modo esclusivo** ad una VM, **senza virtualizzazione**
- Il guest OS accede direttamente al dispositivo tramite memory-mapped I/O

Device pass-through e I/O-MMU



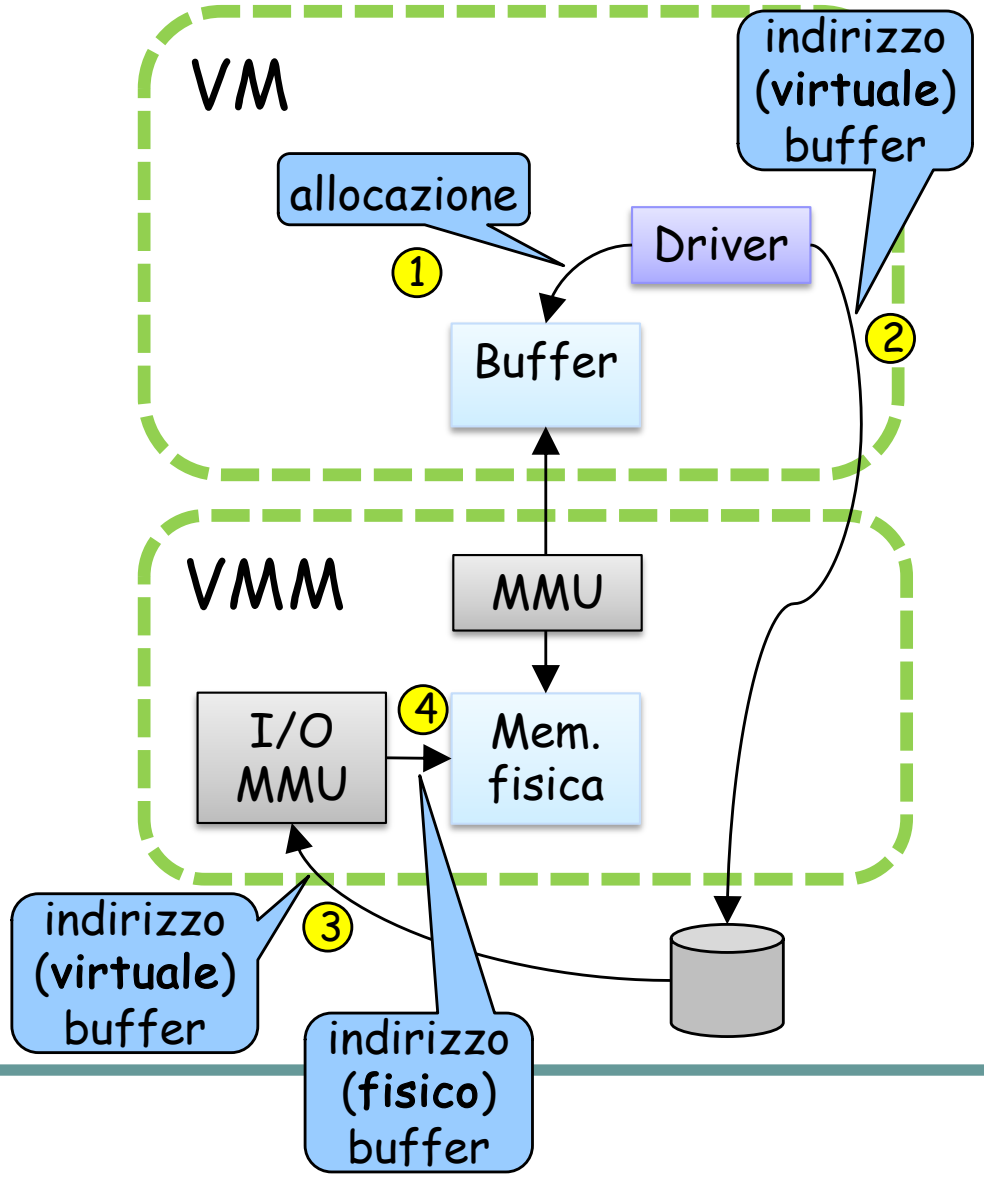
I/O-MMU

- La I/O-MMU è un componente per utilizzare in modo sicuro device pass-through e altre forme di virtualizzazione avanzate
  - Senza I/O-MMU, i dispositivi (es. DMA) possono leggere/scrivere su qualunque indirizzo fisico della RAM
  - Con I/O-MMU, i dispositivi accedono indirettamente alla memoria, tramite indirizzi di memoria virtuali configurati dal VMM

I/O-MMU

- Il guest OS effettua operazioni di memory-mapped I/O su indirizzi **"guest physical" (virtuali)**
- Il dispositivo tenta di accedere all'indirizzo di memoria virtuale
- La **I/O-MMU** traduce l'accesso in un **indirizzo fisico**, controllato dal VMM
- La I/O-MMU inoltra anche le interruzioni del dispositivo al guest OS (**interrupt remapping**)

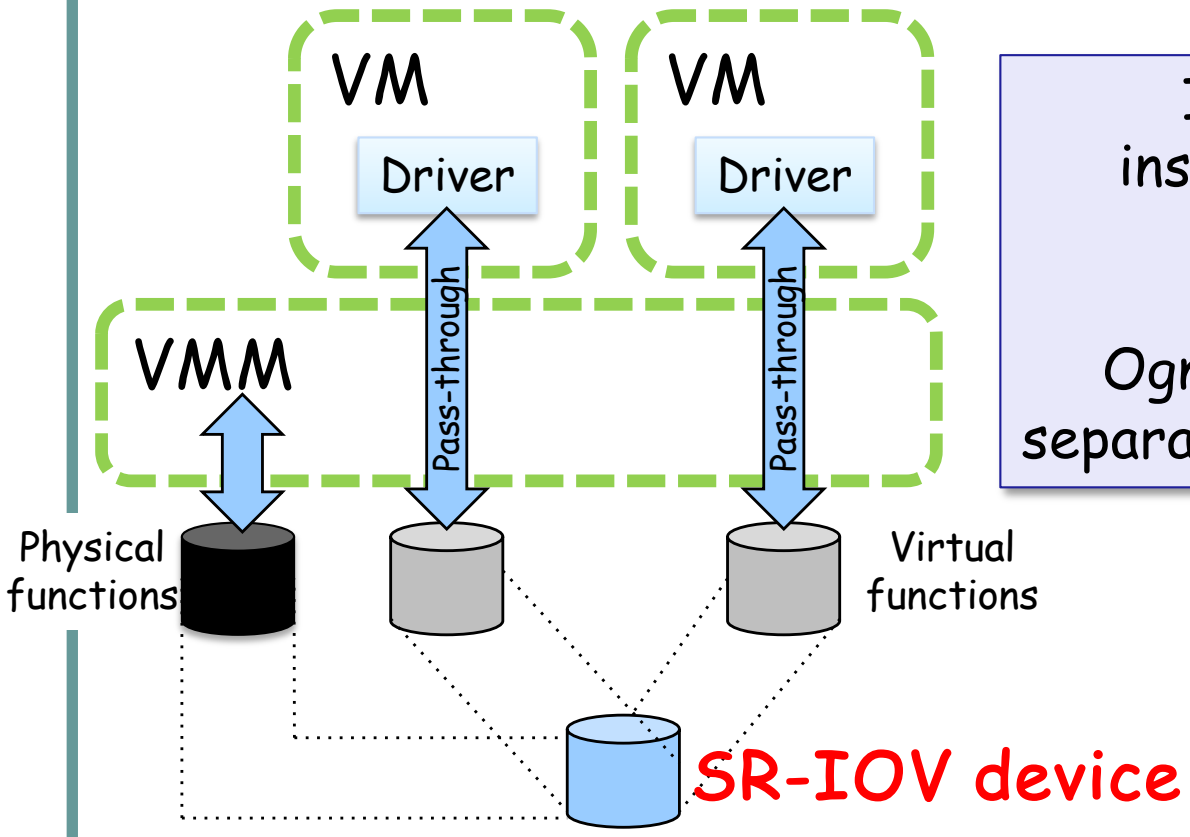
I/O-MMU


VM
indirizzo (virtuale) buffer
allocazione
1
Driver
2
Buffer
VMM
MMU
I/O MMU
4
Mem. fisica
indirizzo (virtuale) buffer
3
indirizzo (fisico) buffer

Single Root I/O Virtualization (SR-IOV)

- In caso di molte VM, non è fattibile assegnare in modo esclusivo un dispositivo ad ogni VM
- La **SR-IOV** è una funzionalità dei moderni dispositivi **PCIe (PCI Express)**
- Permette la condivisione dei device fra più VM

Single Root I/O Virtualization (SR-IOV)



Il controller fornisce un insieme di **interfacce virtuali** (virtual functions).

Ogni interfaccia ha un insieme separato di registri, DMA, interrupt

LE TECNOLOGIE VMWARE

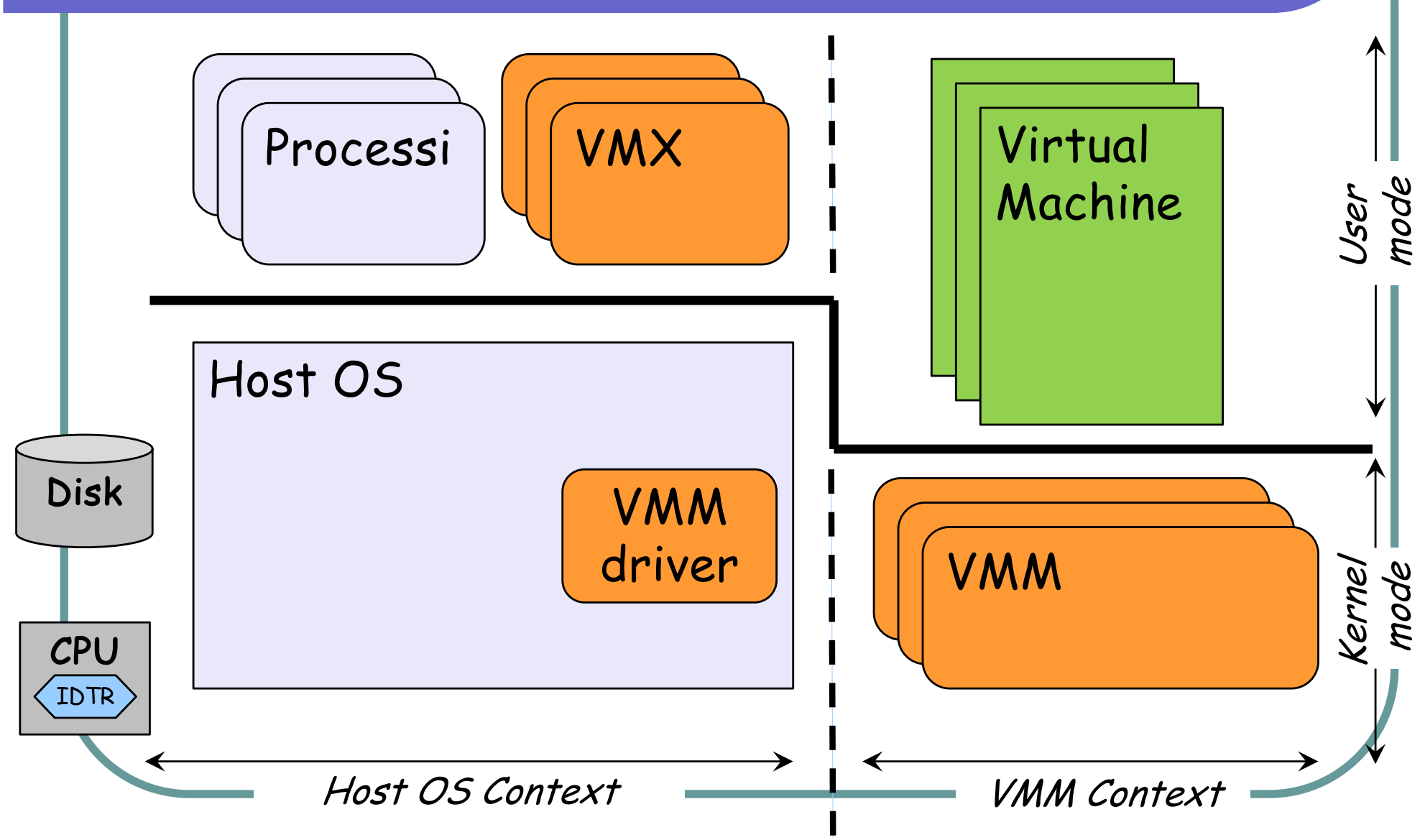
VMware Workstation

- VMware Workstation è stata la prima soluzione per piattaforme Intel x86
  - Un **hypervisor type-2**, che esegue come fosse una applicazione su **host OS** Windows o Linux
  - Full CPU virtualization con dynamic binary translation
  - Full memory virtualization con shadow page tables
  - Full I/O virtualization, emulando semplici dispositivi legacy

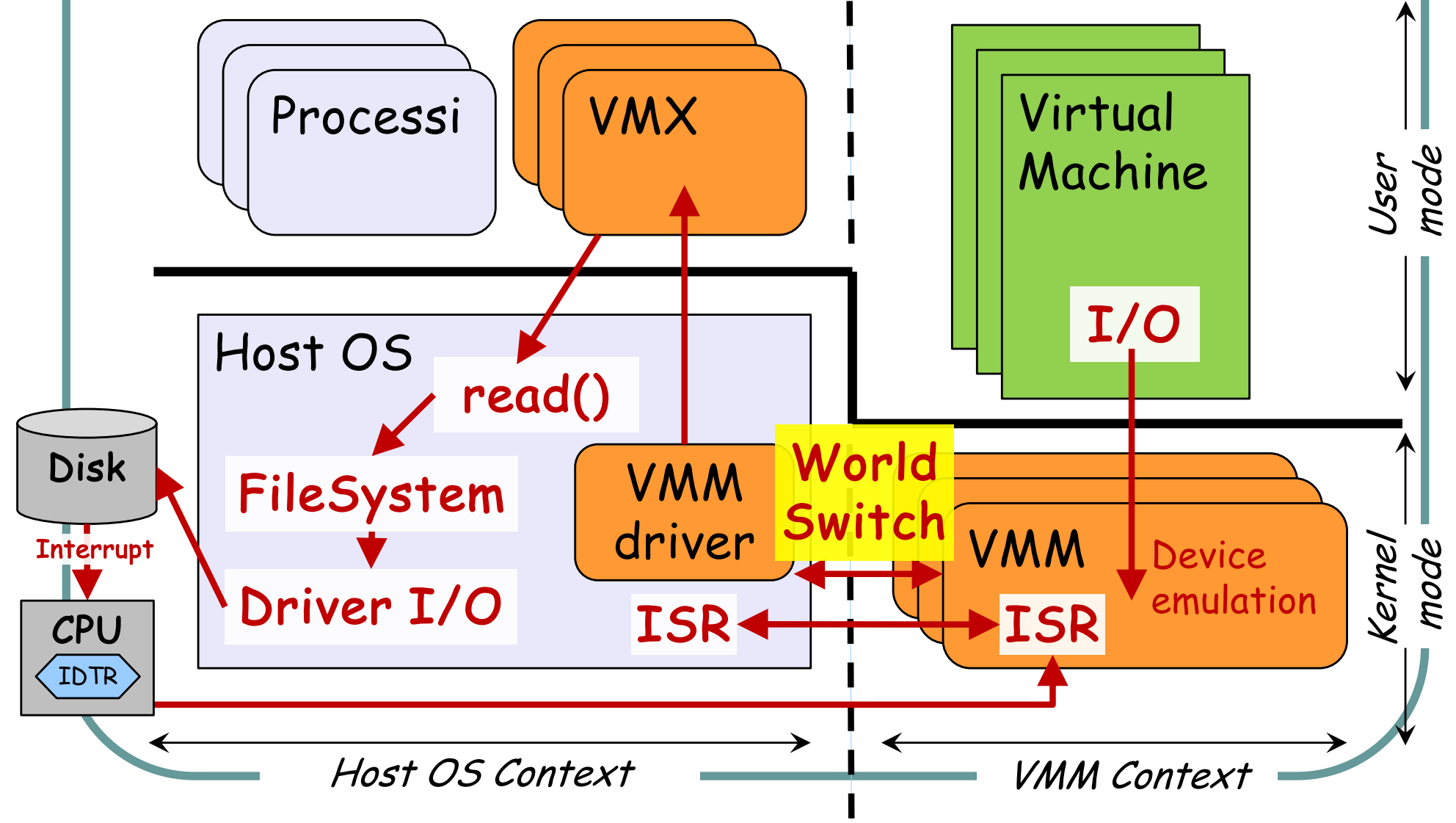
VMware Hosted Architecture

- Per "convivere" con lo host OS, VMware Workstation introduce i seguenti componenti:
  - **Processo VMX**: programma user-space “percepito” dall’utente (un VMX per ogni VM)
  - **VMM**: esegue in modo kernel, fuori dal contesto dello host OS; implementa i meccanismi di full virtualization (un VMM per ogni VM)
  - **VMM driver**: modulo kernel che esegue nel contesto dello host OS, e permette la cooperazione con il VMM
  - **World Switch**: cambio di contesto tra host OS e VMM (tutto lo stato della CPU, incluso registri di controllo)

VMware Hosted Architecture



VMware Hosted Architecture



VMware Hosted Architecture

- Esempio di lettura di I/O nella VM:
  1. Il guest OS nella VM effettua I/O sul dispositivo virtuale
  2. Il VMM intercetta l'operazione, ed innesca un world switch verso lo host OS
  3. Il VMM driver notifica il processo VMX
  4. Il processo VMX effettua read() sul file .VMDK
  5. Il disco genera una interrupt, intercettata dal VMM
  6. Il VMM innesca un altro world switch verso lo host OS
  7. Il VMM driver innesca la ISR nello host OS
  8. I dati sono letti dal processo VMX e raggiungono la macchina virtuale, seguendo il percorso opposto

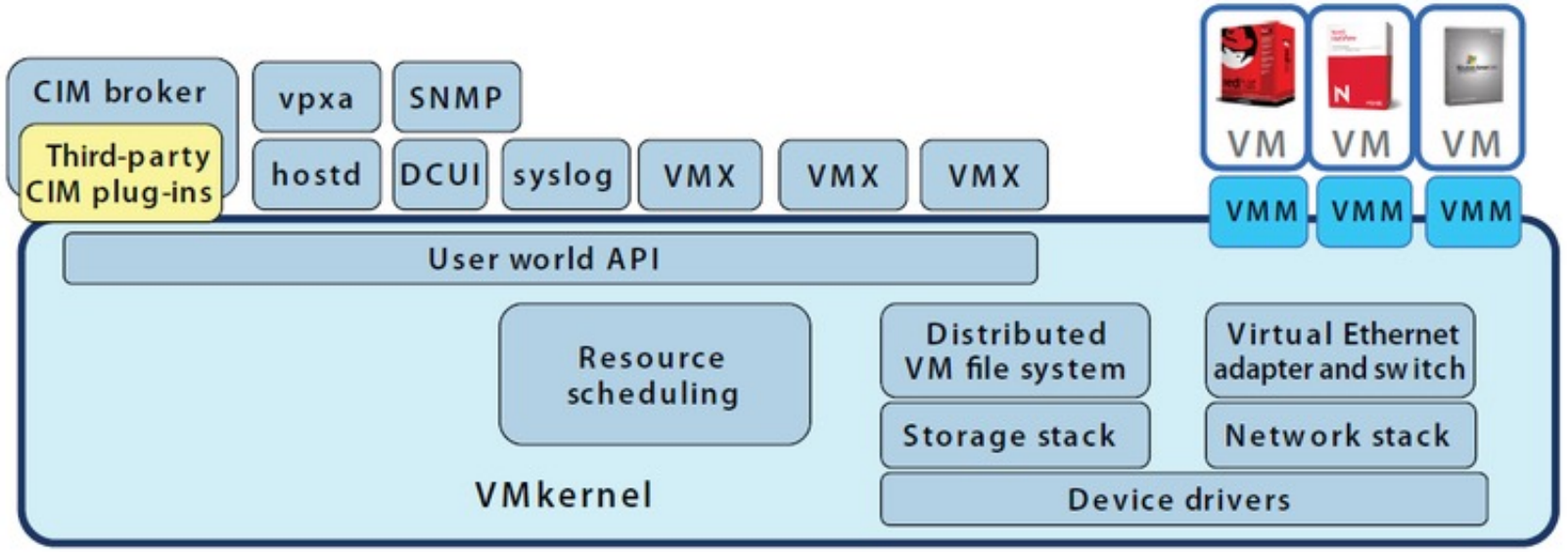
Virtual Hardware Platform

	Virtual Hardware (front end)	*Back end*
Multiplexed	1 virtual x86 CPU, with the same instruction set extensions as the underlying hardware CUP	Scheduled by the host operating system on either a uniprocessor or multiprocessor host
	Up to 512 MB of contiguous DRAM	Allocated and managed by the host OS (page-by-page)

Emulated	PCI Bus	Fully emulated compliant PCI bus
	4x IDE disks 7x Buslogic SCSI Disks	Virtual disks (stored as files) or direct access to a given raw device
	1x IDE CD-ROM	ISO image or emulated access to the real CD-ROM
	2x 1.44 MB floppy drives	Physical floppy or floppy image
	1x VMware graphics card with VGA and SVGA support	Ran in a window and in full-screen mode. SVGA required VMware SVGA guest driver
	2x serial ports COM1 and COM2	Connect to host serial port or a file
	1x printer (LPT)	Can connect to host LPT port
	1x keyboard (104-key)	Fully emulated; keycode events are generated when they are received by the VMware application
	1x PS-2 mouse	Same as keyboard
	3x AMD Lance Ethernet cards	Bridge mode and host-only modes
	1x Soundblaster	Fully emulated

VMware ESXi

- VMware ESXi è un **hypervisor bare metal**
- Riusa il VMM da Workstation, l'host OS è sostituito dal **VMkernel**
- CPU e memory mgmt. ottimizzati per scalabilità e basso overhead
- Funzioni avanzate di migrazione, network/storage virtualization, ...



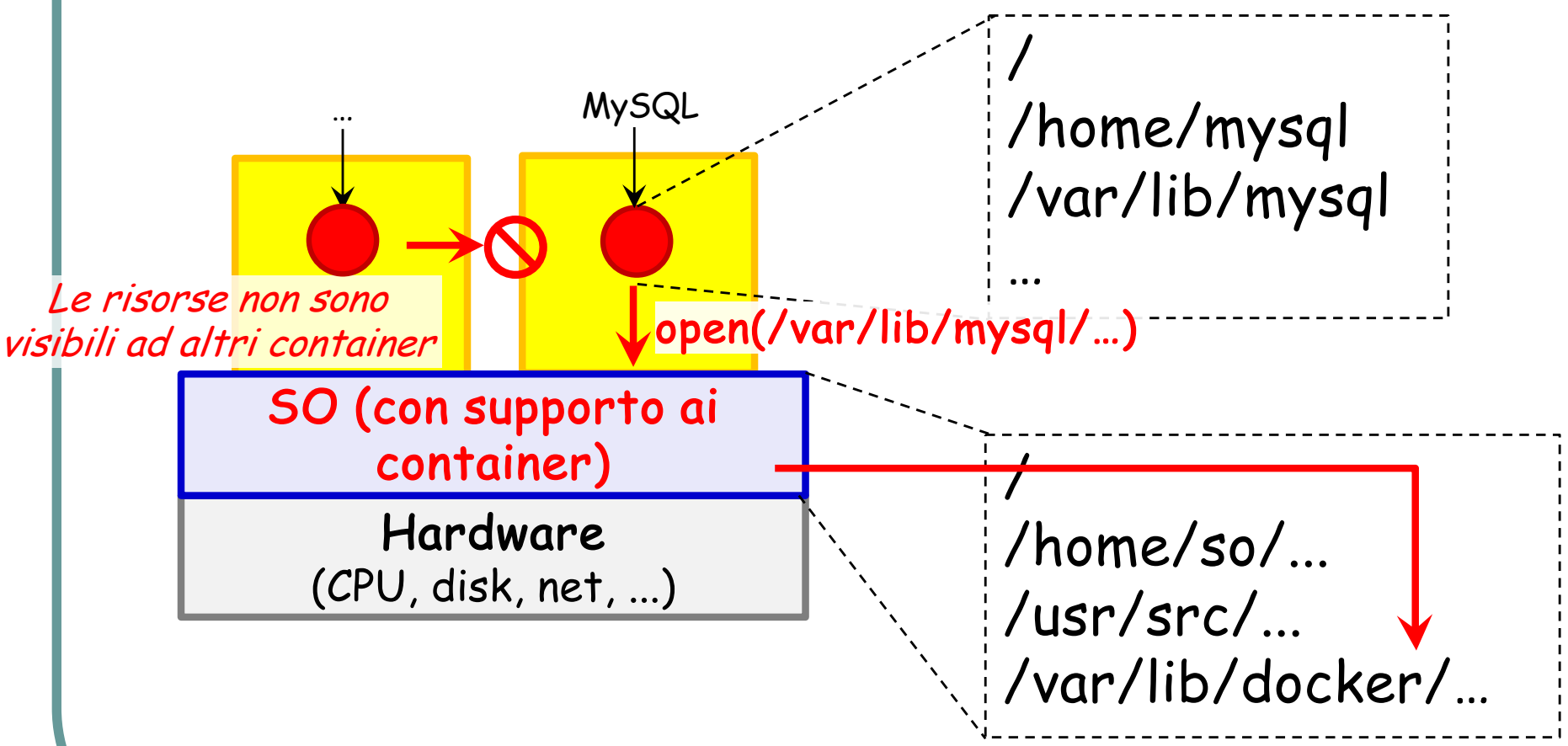
CONTAINER-BASED "VIRTUALIZATION"

Containers

Containers

- I **containers** sono una estensione del concetto di processo
- Un containers è dotato di
  - un proprio file-system namespace
  - propri indirizzi IP
  - propri processi figli
  - …

Container namespaces



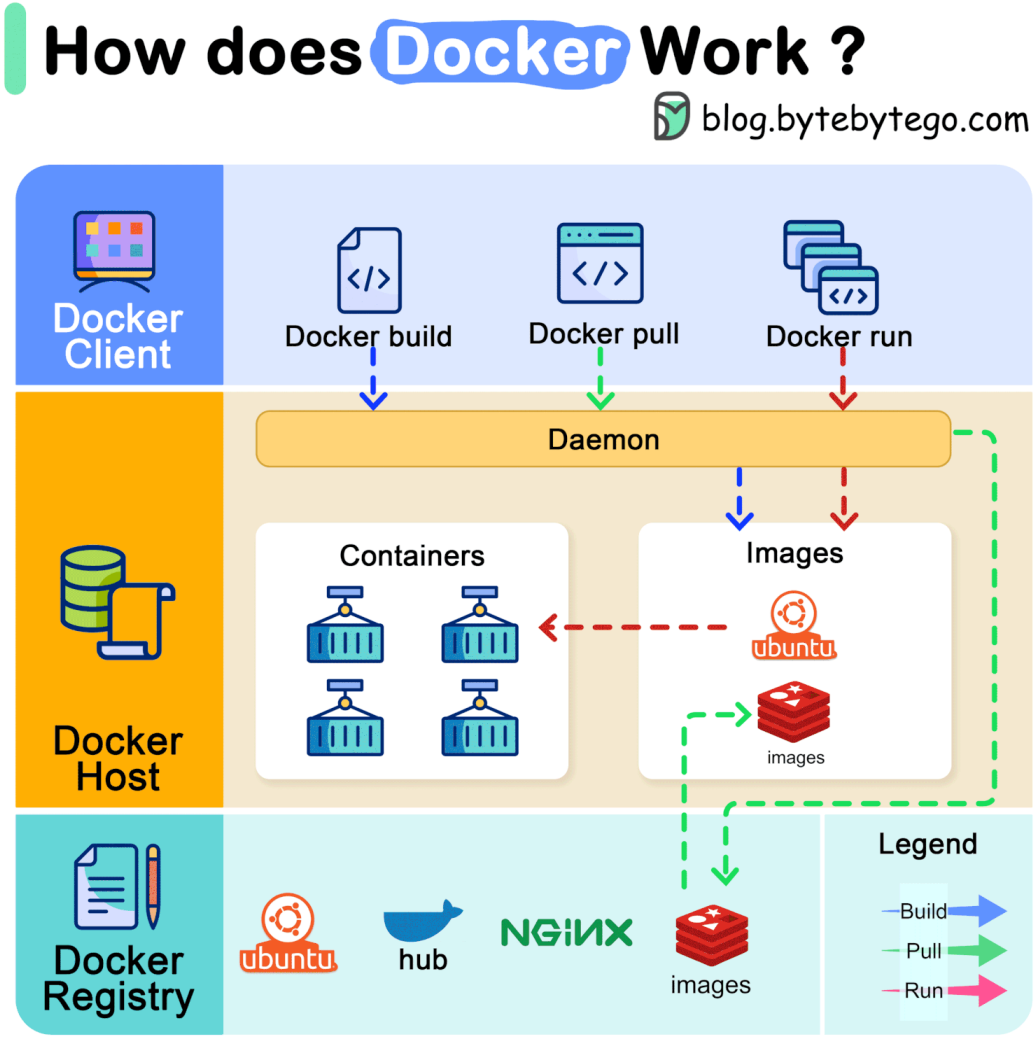
Virtualization vs. containerization

- "**Virtualization**" si riferisce alle tecnologie basate su hypervisor
- "**Containerization**" (OS-level virtualization) non richiede un hypervisor

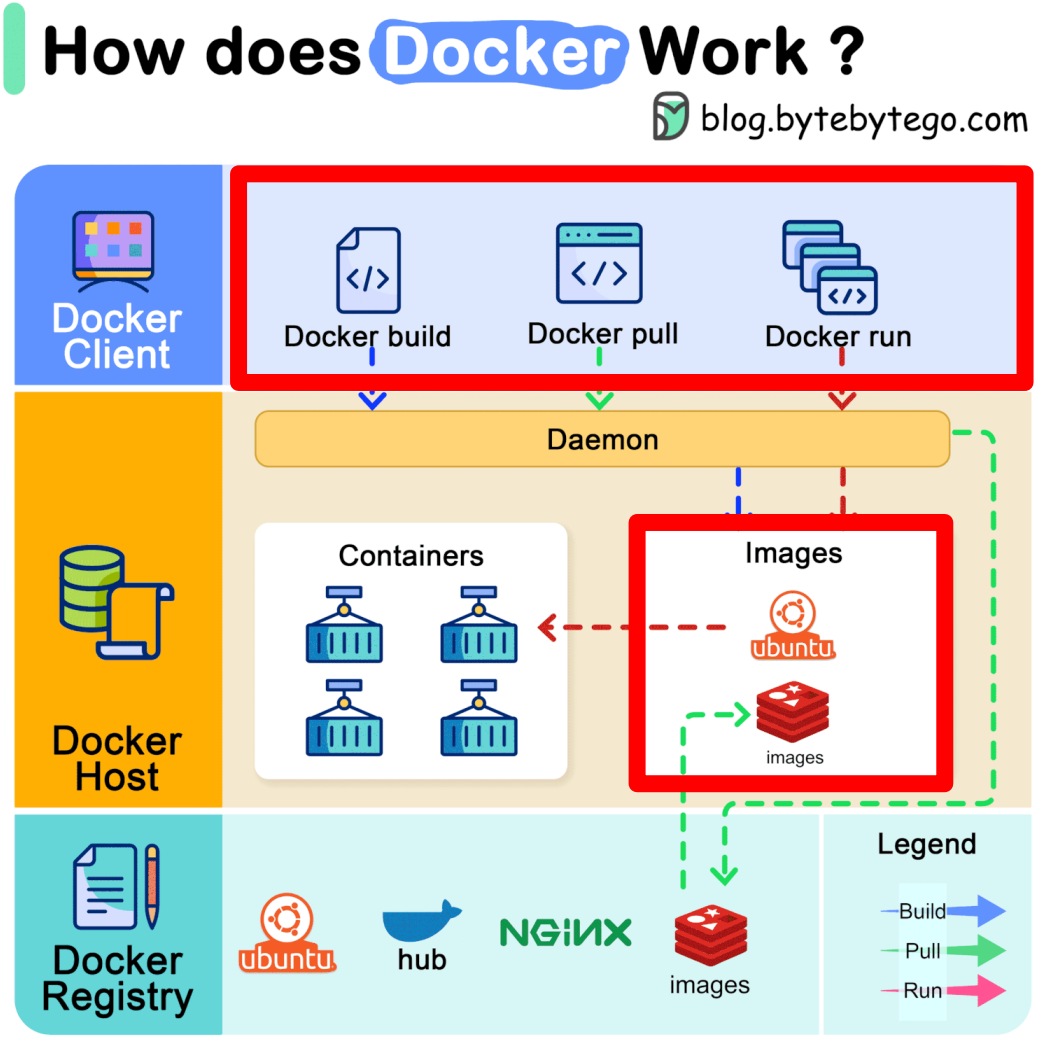
Tecnologie di containerization

- **Linux Containers (LXC)**: estensioni del kernel Linux per partizionare le risorse
  - Namespaces
  - Cgroups
- **Docker**: software di gestione dei container (di tipo LXC e altri)

Docker


How does Docker Work ?
blog.bytebytego.com
Docker Client
Docker build
Docker pull
Docker run
Daemon
Containers
Images
ubuntu
images
Docker Host
Docker Registry
ubuntu
hub
NGINX
images
Legend
Build
Pull
Run

Docker



Configurazione del container (es. il suo filesystem)

Immagine del container

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Initial Docker image

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Runs "apt", updates the package database

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

More "apt" commands

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

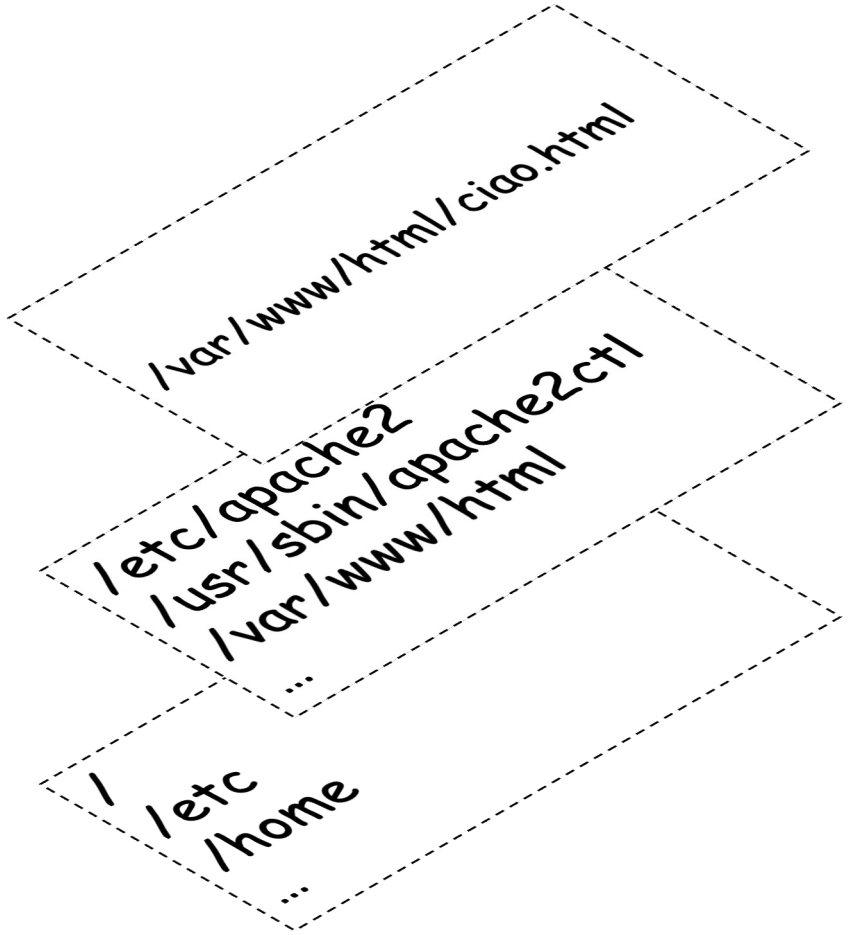
Creates a file with a simple web page

Dockerfile

```
FROM ubuntu:20.04
RUN apt update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt install -y apache2
RUN apt install -y apache2-utils
RUN apt clean
RUN echo "CIAO MONDO" > /var/www/html/ciao.html
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

When the container runs, it will launch Apache HTTPd

Docker build

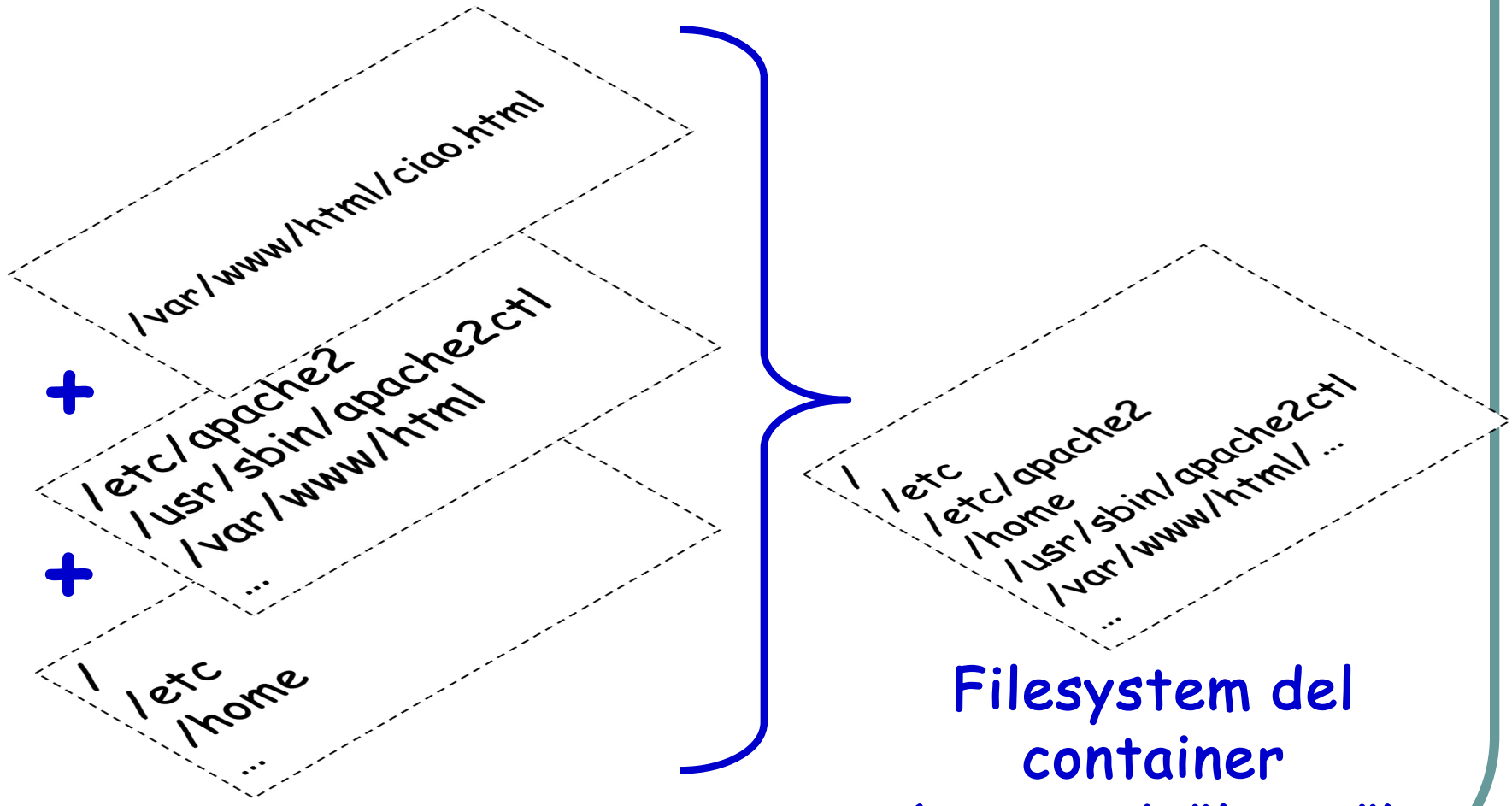


RUN echo …

RUN apt install …

FROM ubuntu22:04

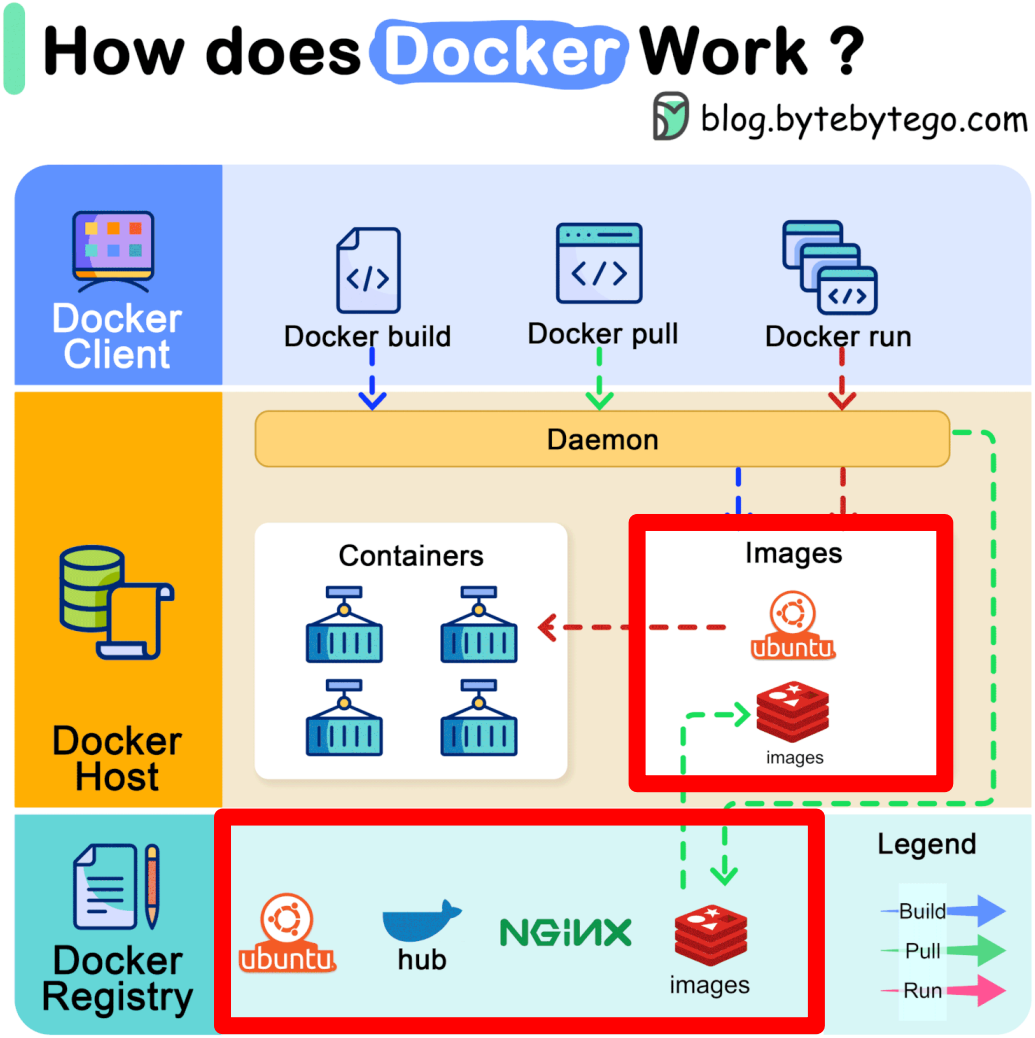
Docker build



Comando docker

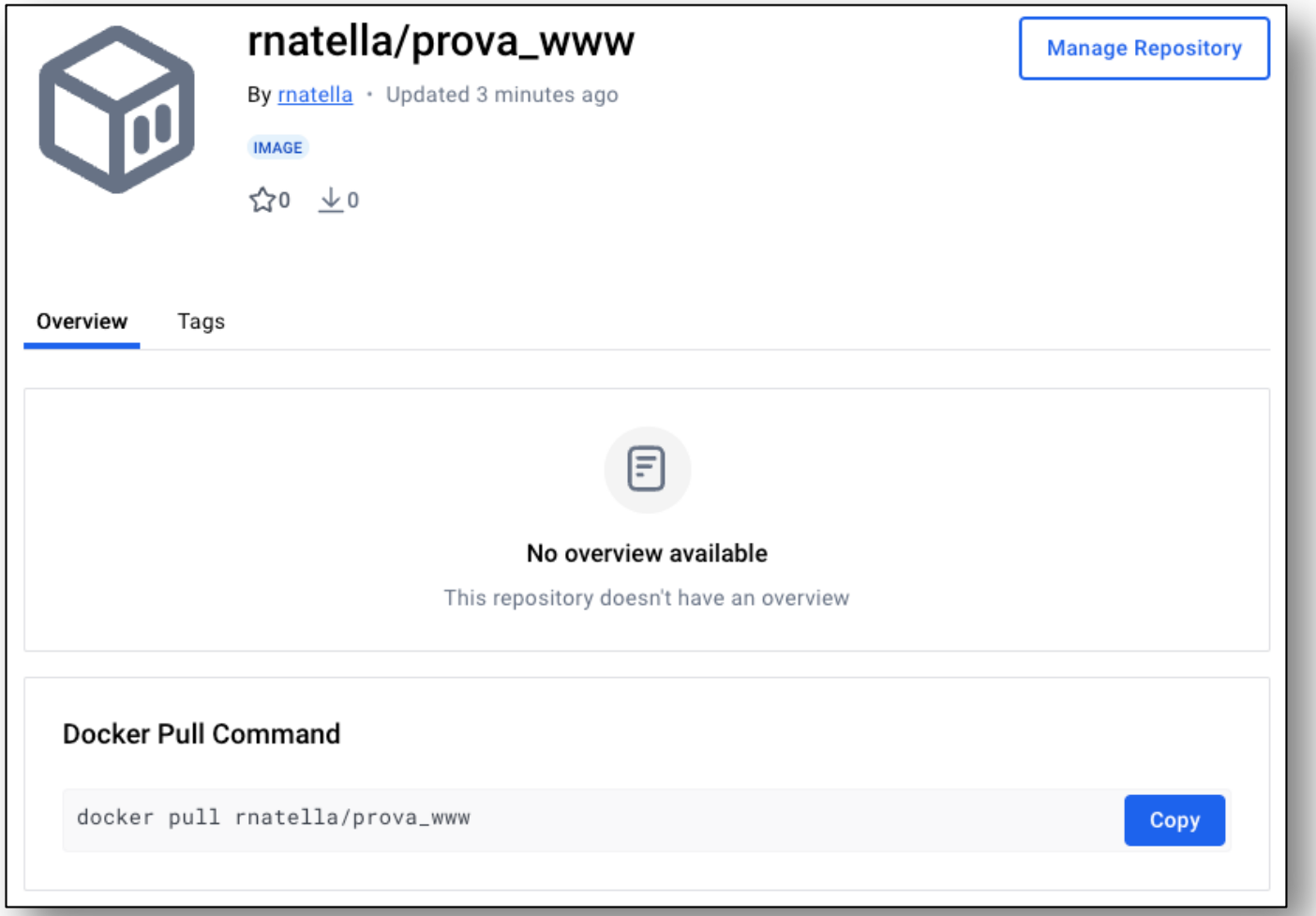
```
$ docker build -t web_server_image:1.0 .
…

$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED                  SIZE
web_server_image    1.0       437539fe2115   About a minute ago   234MB

$ docker run --name mioserver -d -p 80:80
                              web_server_image:1.0
d3d6fc174b36...

$ docker ps
CONTAINER ID  IMAGE                  COMMAND                  …  NAMES
d3d6fc174b36  web_server_image:1.0   "apache2ctl -D FOREG…"   …  mioserver
```

Docker Hub



Il Docker Hub consente di condividere immagini Docker con altri

Docker Hub



https://hub.docker.com

Docker Hub

```
$ docker login

# assegna un "tag" da usare come nome pubblico
$ docker tag prova_www rnatella/prova_www

# carica l'immagine su Docker Hub
$ docker push rnatella/prova_www
```

Docker Hub

```
# scarica l'immagine da Docker Hub
$ docker pull rnatella/prova_www

# esegue l'immagine
$ docker run -it -p 80:80 rnatella/prova_www
```