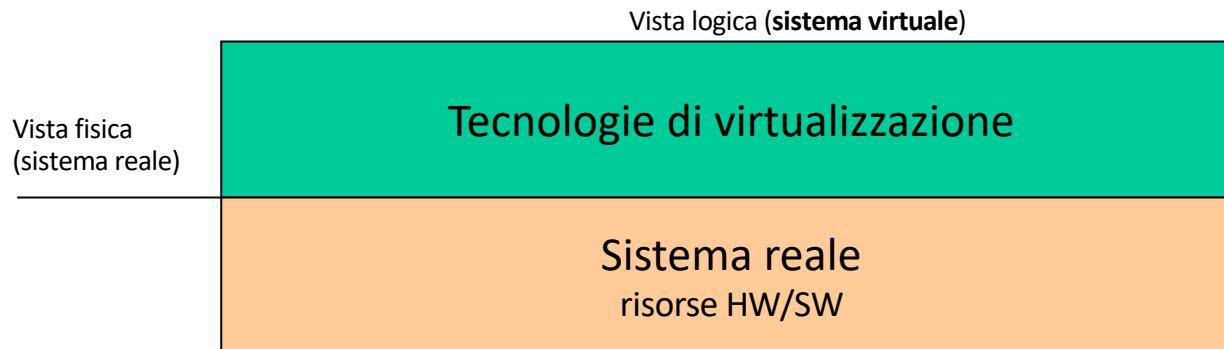


# Virtualizzazione

# Virtualizzazione

Dato un sistema costituito da un insieme di risorse (hardware e software), **virtualizzare il sistema** significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.

Ciò si ottiene introducendo **un livello di indirezione** tra la vista logica e quella fisica delle risorse.



Gli obiettivi della virtualizzazione possono essere diversi.

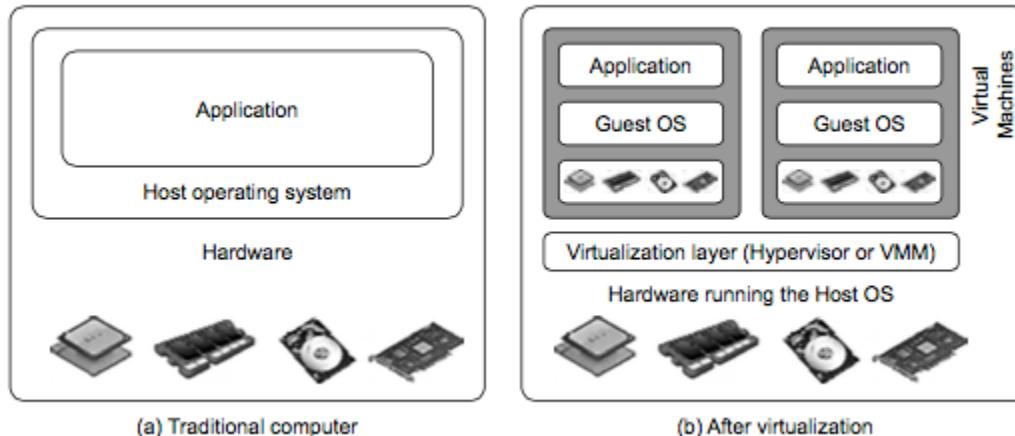
# Virtualizzazione: esempio

**Obiettivo:** disaccoppiare il comportamento delle risorse di un sistema di elaborazione offerte all'utente dalla loro realizzazione fisica.

Le risorse virtualizzate appaiono all'utente con caratteristiche in generale diverse e indipendenti da quelle "reali".

**Esempio:**

## Macchine Virtuali



# Esempi di virtualizzazione

**Virtualizzazione a livello di processo.** I sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una macchina virtuale (CPU, memoria, dispositivi) dedicata. La virtualizzazione è realizzata dal **kernel** del sistema operativo.

**Virtualizzazione della memoria.** In presenza di memoria virtuale, ogni processo vede uno spazio di indirizzamento di dimensioni indipendenti dallo spazio fisico effettivamente a disposizione. La virtualizzazione è realizzata dal **kernel** del sistema operativo.

# Altri Esempi di virtualizzazione

**Astrazione:** in generale un oggetto astratto (risorsa virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica):

- esibendo le proprietà significative per l'utilizzatore
- nascondendo i dettagli realizzativi non necessari.

Es: tipi di dato vs. rappresentazione binaria nella cella di memoria

Il **disaccoppiamento** è realizzato dalle operazioni (interfaccia) con le quali è possibile utilizzare l'oggetto.

**Linguaggi di Programmazione.** La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura del sistema (S.O. e HW):

- Interpreti (esempio Java Virtual Machine)
- Compilatori

# Emulazione

Esecuzione di programmi compilati per una particolare architettura (e quindi un **particolare insieme di istruzioni**) su un sistema di elaborazione dotato di un **diverso insieme di istruzioni**.

- Vengono emulate interamente le singole istruzioni dell'architettura ospitata permettendo a sistemi operativi o applicazioni, pensati per determinate architetture, di girare, non modificati, su architetture completamente differenti.
- **Vantaggi:** interoperabilità tra ambienti eterogenei,
- **Svantaggi:** ripercussioni sulle performances (problemi di efficienza).
- L'approccio dell'emulazione ha seguito nel tempo due strade: **l'interpretazione e la ricompilazione dinamica**.

# Interpretazione

Il modo più diretto per emulare è *interpretare*. L'interpretazione si basa sulla lettura di **ogni singola istruzione** del codice macchina che deve essere eseguito e **sulla esecuzione di più istruzioni sull'host virtualizzante** per ottenere semanticamente lo stesso risultato

E' un metodo molto generale e potente che presenta una grande flessibilità nell'esecuzione perché consente di emulare e riorganizzare i meccanismi propri delle varie architetture. Vengono, ad esempio, normalmente utilizzate parti di memoria per salvare il contenuto dei registri della CPU emulata, registri che potrebbero non essere presenti nella CPU emulante.

- Produce un sovraccarico generalmente elevato poiché possono essere necessarie **molte istruzioni dell'host** per interpretare una singola istruzione sorgente.

# Compilazione dinamica

- Invece di leggere una singola istruzione del sistema ospitato, legge **interi blocchi di codice**, li analizza, li traduce per la nuova architettura **ottimizzandoli** e infine li mette in esecuzione.
- Il vantaggio in termini prestazionali è evidente. Invece di interpretare una singola istruzione alla volta, il codice viene **tradotto e ottimizzato**, utilizzando tutte le possibilità offerte dalla nuova architettura e messo in esecuzione.
- Parti di codice utilizzati frequentemente possono essere **bufferizzate** per evitare di doverle ricompilare in seguito.
- Tutti i più noti emulatori (es: **QEMU , Virtual PC, Mame**) utilizzano questa tecnica per implementare l'emulazione.

# **QEMU**

**Qemu** è un software che implementa un particolare sistema di emulazione che permette di ottenere **un'architettura nuova e disgiunta** in un'altra che si occuperà di ospitarla(convertire, ad esempio, le istruzioni da 32 bit a 64) **permettendo quindi di eseguire programmi compilati su architetture diverse**

- Questo software è conosciuto grazie alla sua velocità di emulazione ottenuta grazie alla tecnica della *traduzione dinamica*.
- . **Qemu**, inizialmente, era un progetto che si prefiggeva di emulare solo il microprocessore x86 su un sistema GNULinux, allo scopo di poter eseguire in ambiente linux applicazioni windows (tramite Wine).
- Ora, **Qemu** è in grado di emulare sistemi x86, AMD64, Power PC, MIPS e ARM.

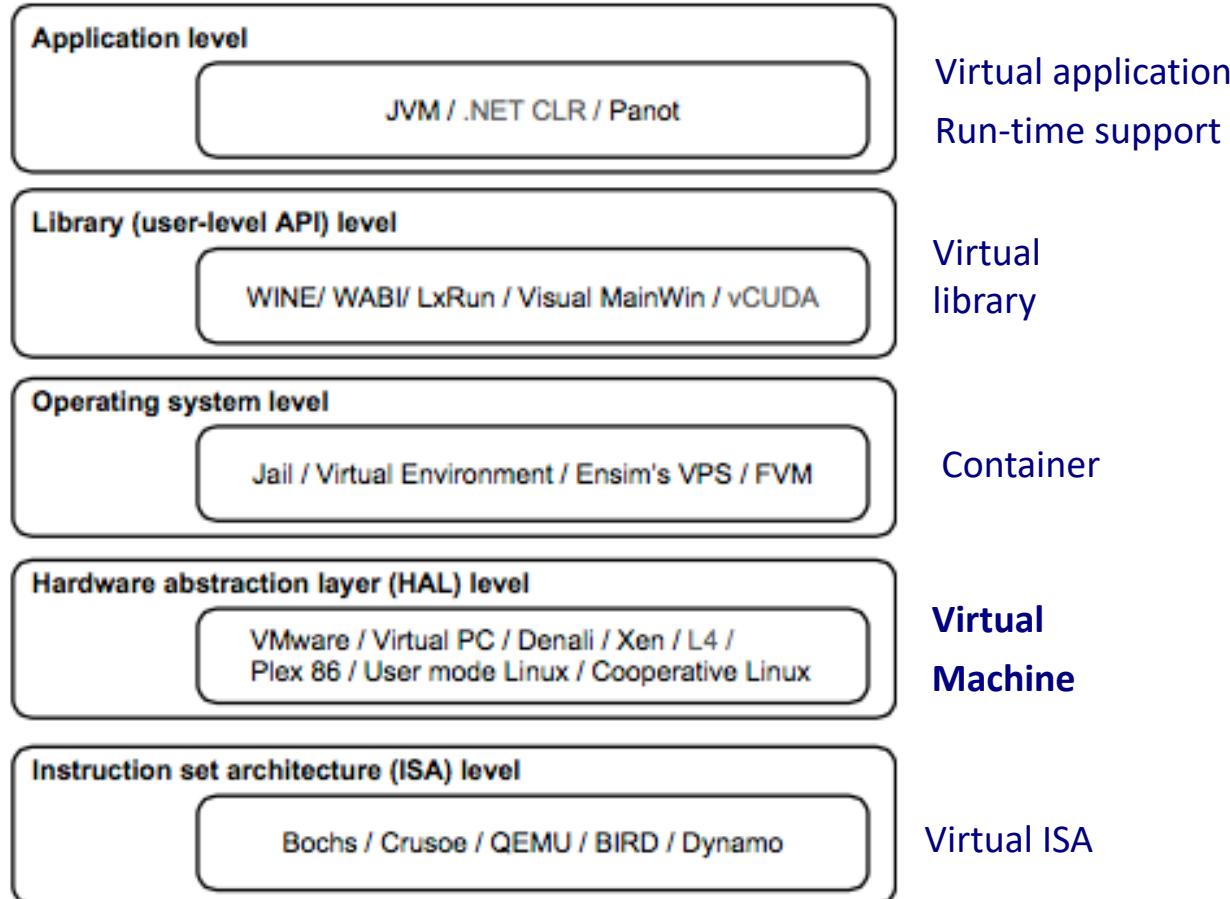
# Virtual PC

- Software di emulazione che consentiva a computer con **sistema operativo Microsoft windows o Mac OSX** l'esecuzione di **sistemi operativi diversi**, come varie versioni di Windows, MacOS o Linux, anche in contemporanea.
- L'emulatore ricrea in forma virtuale un ambiente di lavoro che riproduce quasi integralmente quello di un **PC basato su Intel**.
- è destinato soprattutto a consentire **l'uso di vecchie applicazioni** non più supportate dai moderni sistemi operativi.
- **L'introduzione dei processori Intel** nei computer Apple ha reso Virtual PC non più così interessante per gli utenti MacOS.  
(possibilità di *dual boot* o di uso di software di virtualizzazione, es. Parallels, Virtual Box, ecc. )

# MAME

- (Multiple Arcade Machine Emulator) è un software per personal computer sviluppato inizialmente per MS-DOS e in seguito per quasi tutte le macchine e sistemi operativi in circolazione, in grado di **caricare ed eseguire il codice binario originale delle ROM** dei videogiochi da bar (*arcade*), emulando l'hardware tipico di quelle architetture.
- Facendo leva sull'enorme potenza di calcolo degli attuali PC rispetto ai primordiali processori per video giochi dell'epoca, la VM può tranquillamente operare tramite **interpretazione** senza compromettere in modo significativo l'esperienza di gioco, almeno per i video giochi più vecchi che non facevano un uso massiccio della grafica.

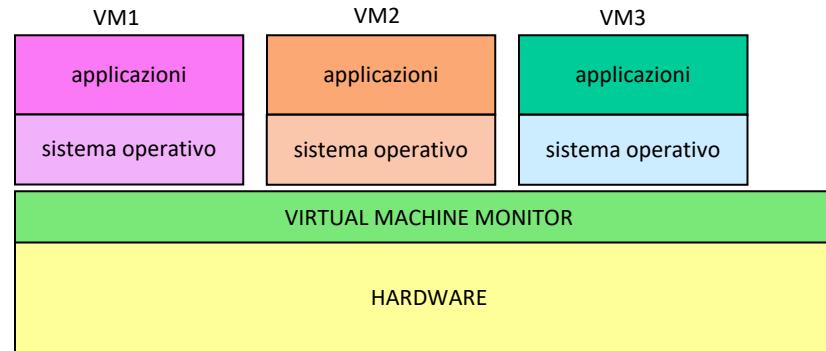
# Tipi (livelli) di virtualizzazione\*



# Macchine virtuali

Una singola piattaforma hardware viene condivisa da più elaboratori virtuali (macchine virtuali o VM) ognuno gestito da un proprio sistema operativo.

Il disaccoppiamento è realizzato da un componente chiamato *Virtual Machine Monitor* (VMM, o *hypervisor*) il cui compito è consentire la **condivisione da parte di più macchine virtuali di una singola piattaforma hardware**. Ogni VM contiene un proprio sistema operativo, che definisce un ambiente di esecuzione distinto e isolato delle altre macchine virtuali, che consente l'esecuzione di applicazioni all'interno di esso.



Il VMM è il **mediatore unico** nelle interazioni tra le macchine virtuali e l'hardware sottostante, che garantisce:

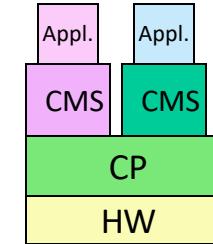
- **isolamento** tra le VM
- **stabilità** del sistema

# Qualche cenno storico

La virtualizzazione non è un concetto nuovo:

## ■ Anni 60: IBM

- CP/CMS sistema suddiviso in 2 livelli:
  - CP (control program): esegue direttamente sull'HW svolgendo il ruolo di VMM, offrendo molteplici interfacce allo strato superiore
  - CMS (conversational monitor system): sistema operativo, interattivo e monoutente, replicato per ogni macchina virtuale
- VM/370: evoluzione di CP/CMS, caratterizzata dalla possibile eterogeneità di sistemi operativi nelle diverse macchine virtuali (IBM OS/360, DOS/360).

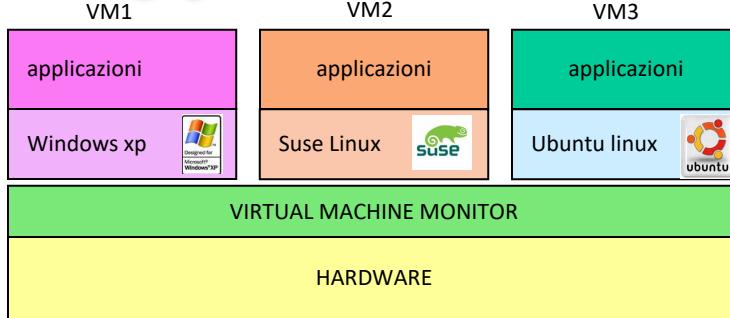


## ■ Anni 70:

- sistemi operativi multitasking

- Anni 80:
  - Evoluzione della tecnologia (microprocessori, reti)
  - Crollo del costo dell'hw
  - Migrazione da architetture basate su mainframe verso minicomputer e PC
- Anni 80-90:
  - I produttori di hardware abbandonano l'idea di supportare il concetto di virtualizzazione a livello architetturale (Es. Intel IA-32)
  - Paradigma “one application, one server”
    - ➔ esplosione del numero di server fisici da configurare, gestire, manutenere, ecc.
    - ➔ Sottoutilizzo delle risorse hardware
- Anni 2000: necessità di razionalizzazione !
  - nuovi sistemi di virtualizzazione per l'architettura Intel x86 (1999, VMware)
- Anni 2010: Cloud Computing.

# Vantaggi della virtualizzazione



- **Uso di piu` S.O. sulla stessa macchina fisica:** più ambienti di esecuzione (eterogenei) per lo stesso utente:
  - Legacy systems
  - Possibilità di esecuzione di applicazioni concepite per un particolare s.o.
- **Isolamento degli ambienti di esecuzione:** ogni macchina virtuale definisce un ambiente di esecuzione separato (sandbox) da quelli delle altre:
  - possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM.
  - Sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

- **Consolidamento HW:** possibilita` di concentrare piu` macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. server farm):
  - Abbattimento costi hw
  - Abbattimento costi amministrazione
- **Gestione facilitata delle macchine:** e` possibile effettuare in modo semplice:
  - la **creazione** di macchine virtuali (virtual appliances)
  - l'**amministrazione** di macchine virtuali (reboot, ricompilazione kernel, etc.)
  - **migrazione a caldo** di macchine virtuali tra macchine fisiche:
    - possibilita` di **manutenzione** hw senza interrompere i servizi forniti dalle macchine virtuali
    - **workload balancing:** alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte a situazioni di sbilanciamento del carico computazionale
    - **disaster recovery**

# Realizzazione del VMM

In generale, il VMM deve offrire alle diverse macchine virtuali le risorse (virtuali) che sono necessarie per il loro funzionamento:

- CPU
- Memoria
- Dispositivi di I/O

# Realizzazione del VMM

**Requisiti** (Popek e Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures" 1974):

1. **Ambiente di esecuzione** per i programmi sostanzialmente **identico** a quello della macchina reale.  
Gli stessi programmi che eseguono sull'architettura non virtualizzata possono essere eseguiti nelle VM senza modifiche.
2. **Garantire un'elevata efficienza** nell'esecuzione dei programmi.  
Quando possibile, il VMM deve permettere l'esecuzione diretta delle istruzioni impartite dalle macchine virtuali -> le istruzioni non privilegiate vengono eseguite direttamente in hardware senza coinvolgere il VMM.
3. **Garantire la stabilità e la sicurezza** dell'intero sistema.  
Il VMM deve rimanere sempre nel pieno controllo delle risorse hardware -> i programmi in esecuzione nelle macchine virtuali (applicazioni e S.O.) non possono accedere all'hardware in modo privilegiato.

# Realizzazione VMM: parametri e classificazione

- **Livello** dove è collocato il VMM:

- **VMM di sistema**: eseguono direttamente sopra l'hardware dell'elaboratore (es. vmware esx, xen, kvm)
- **VMM ospitati**: eseguiti come applicazioni sopra un S.O. esistente (es. vmware player, parallels, virtualPC, virtualbox, UserModeLinux)

- **Modalità di dialogo** per l'accesso alle risorse fisiche tra la macchina virtuale ed il VMM:

- **Virtualizzazione pura** (Vmware): le macchine virtuali usano la stessa interfaccia (istruzioni macchina) dell'architettura fisica
- **Paravirtualizzazione** (xen): il VMM presenta un'interfaccia diversa da quella dell'architettura hw.

# VMM di sistema vs. VMM ospitati

## VMM di Sistema.

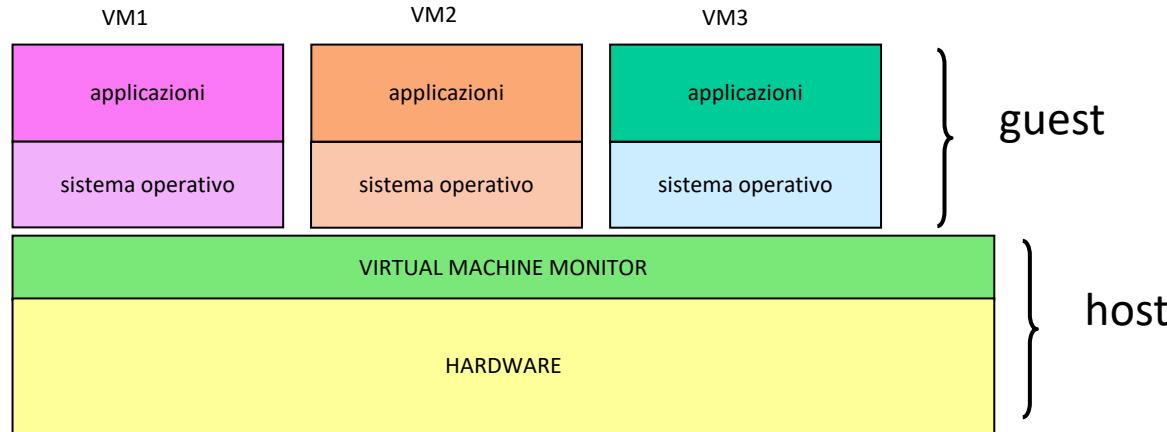
le funzionalità di virtualizzazione vengono integrate in un sistema operativo *leggero* (VMM) posto direttamente sopra l'hardware dell'elaboratore.

- E' necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche.

Esempi di VMM di sistema: kvm, xen, Vmware vsphere, Microsoft HyperV.

**Host**: piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica ed il VMM.

**Guest**: la macchina virtuale. Comprende applicazioni e sistema operativo



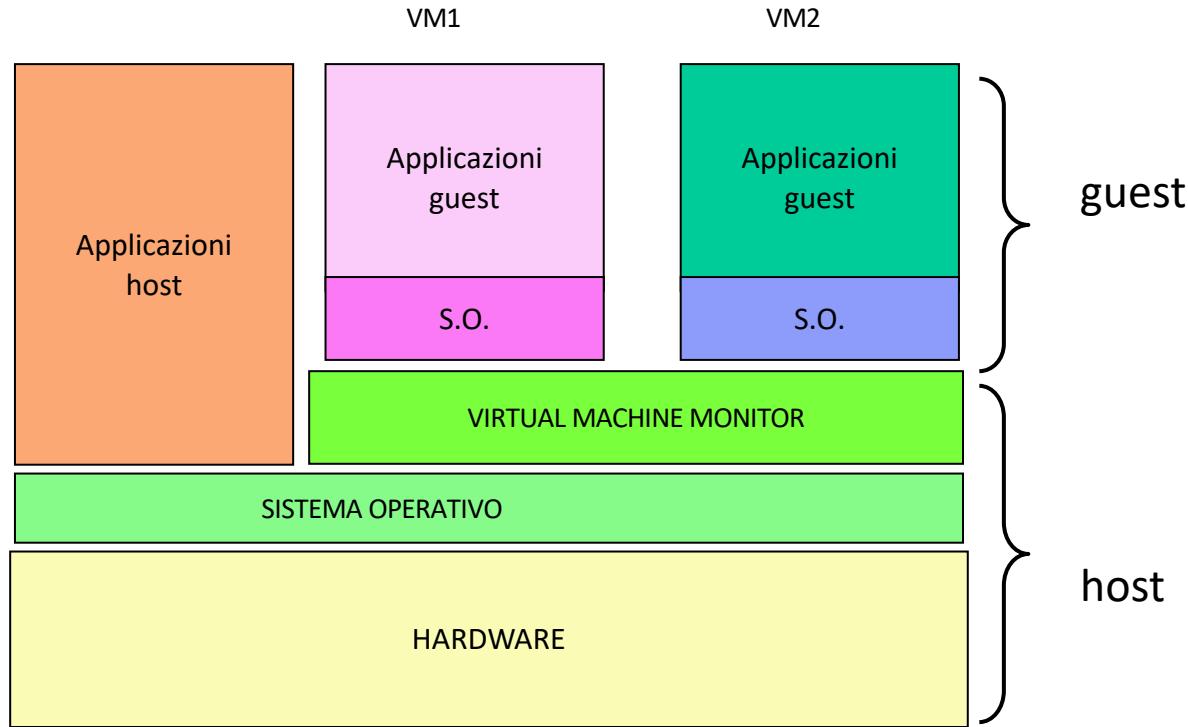
**VMM di Sistema**

## VMM ospitato

il VMM viene installato come **un'applicazione sopra un sistema operativo** esistente; il VMM opera nello spazio utente e accede all'hardware tramite le system call del S.O. su cui viene installato.

- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O. (es. scheduling, gestione dei dispositivi, ecc.).
- Peggiore la performance (rispetto al VMM di sistema).

**Prodotti:** Virtualbox, User Mode Linux, VMware Fusion/player, Parallels, Microsoft Virtual Server , ..



**VMM ospitato**

# Ring di protezione

L'Architettura della CPU prevede, in generale, almeno due livelli di protezione (**ring**): **supervisore o kernel(0)** e **utente(>0)**.

Ogni ring corrisponde a una **diversa modalità di funzionamento del processore**:

- al livello 0 è possibile eseguire le **istruzioni privilegiate** della CPU
- nei ring di livello superiore a 0 le istruzioni privilegiate non possono essere eseguite

Vi sono programmi progettati per eseguire nel ring 0; ad esempio il **kernel del SO** (che deve avere pieno controllo dell'HW).

# VMM\* : realizzazione

In un sistema virtualizzato il VMM deve essere l'unica componente in grado di mantenere il controllo completo dell'hardware:

- solo il VMM opera nello stato supervisore, mentre il sistema operativo e le applicazioni (la macchina virtuale) eseguono in un ring di livello superiore.

## Problemi:

- **ring deprivilegging**: il s.o. della macchina virtuale esegue in un ring che non gli e` proprio (esecuzione di system call)
- **ring compression**: se i ring utilizzati sono solo 2, applicazioni e s.o. della macchina virtuale eseguono allo stesso livello: scarsa protezione tra spazio del s.o. e delle applicazioni.

\* d'ora in poi faremo implicitamente riferimento a VMM di sistema

# Ring deprivilegging

Le istruzioni privilegiate richieste dal sistema operativo nell'ambiente guest **non possono essere eseguite** (perchè richiederebbero il ring 0, e il kernel della VM esegue in un ring di livello superiore).

## Possibile Soluzione: *trap & emulate*

Se il guest tenta di eseguire un'istruzione privilegiata:

- la CPU **notifica** un'eccezione al VMM (**trap**) e gli trasferisce il controllo
- il VMM controlla la correttezza dell'operazione richiesta e ne emula il comportamento (**emulate**).

**NB:** Le istruzioni non privilegiate possono essere eseguite direttamente dalle VM senza alcun intervento da parte del VMM (esecuzione diretta).

**Esempio:** tentativo di esecuzione da parte del guest dell'istruzione privilegiata per la disabilitazione delle interruzioni (**popf**).

👉 Se la richiesta della macchina virtuale fosse eseguita direttamente sul processore, sarebbero disabilitati gli interrupt per tutto il sistema. Il VMM non potrebbe riguadagnare il controllo della CPU.

**Comportamento desiderato:** la consegna degli interrupt va sospesa solamente per la macchina virtuale richiedente.

### **Con trap&emulate:**

Il VMM riceve la notifica di tale richiesta (effettuata da una particolare VM) e ne emula il comportamento atteso sospendendo le interruzioni solo per la VM richiedente.

# Supporto HW alla virtualizzazione

L'architettura della CPU si dice **naturalmente virtualizzabile** (o con supporto nativo alla virtualizzazione) se prevede l'invio di trap allo stato supervisore per ogni istruzione privilegiata invocata da un livello di protezione diverso dal Supervisore.

■ Se l'architettura della CPU e` naturalmente virtualizzabile:

- la realizzazione del VMM e` semplificata: per ogni trap generato dal tentativo di esecuzione di istruzione privilegiata dal guest viene eseguita una routine di emulazione. (Approccio “trap-and-emulate”)
- supporto nativo all'esecuzione diretta.

Es. Intel VT, AMD-V

**PROBLEMA:** Non tutte le architetture sono naturalmente virtualizzabili !

**Esempio: Intel IA32**

Alcune istruzioni privilegiate di questa architettura invocate a livello user non provocano una trap, ma:

- vengono ignorate non consentendo quindi l' intervento trasparente del VMM,
- in alcun casi provocano il crash del sistema.

Ulteriore problema (*Ring Aliasing*) :

Alcune istruzioni non privilegiate, eseguite in modo user, permettono di accedere in lettura ad alcuni registri la cui gestione dovrebbe essere riservata al VMM -> possibili inconsistenze.

**Esempio:** registro CS, che contiene il livello di privilegio corrente (CPL).

# Realizzazione del VMM in architetture non virtualizzabili

Se il processore non fornisce alcun supporto alla virtualizzazione, è necessario ricorrere a **soluzioni software**.

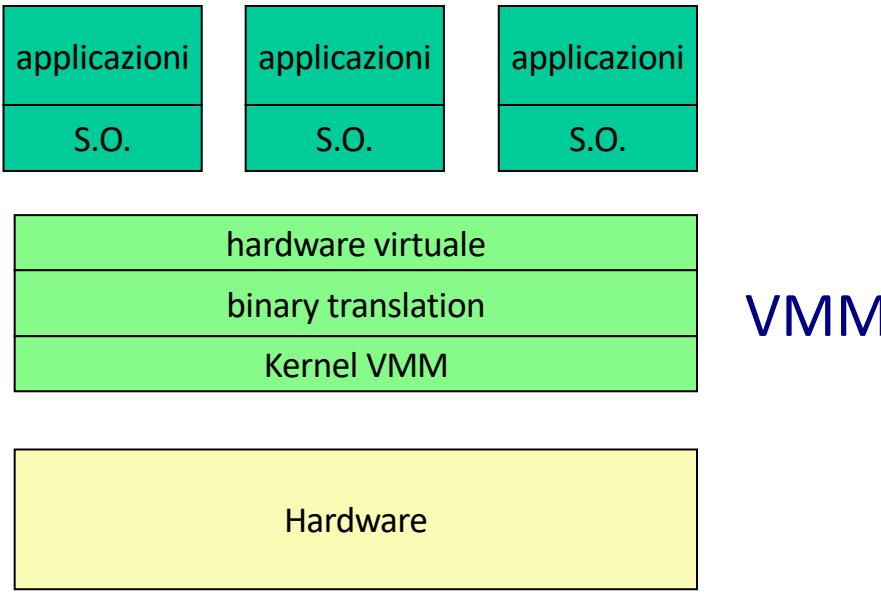
## Soluzioni possibili:

- **Fast binary translation**
- **Paravirtualizzazione.**

## *Fast binary translation*

Il VMM scansiona dinamicamente il codice dei so guest prima dell'esecuzione per **sostituire a run time** i blocchi contenenti istruzioni privilegiate in blocchi equivalenti dal punto di vista funzionale e contenenti chiamate al VMM.

- I blocchi tradotti sono eseguiti e conservati in cache per eventuali riusi futuri.  
(es.Vmware):



Virtualizzazione mediante Fast Binary Translation (architettura non virtualizzabile)

**Pro:** ogni macchina virtuale è una esatta replica della macchina fisica  
 ➔ possibilità di installare gli stessi s.o. di architetture non virtualizzate (**Virtualizzazione pura**)

**Contro:** la traduzione dinamica è costosa !

# Paravirtualizzazione

Il VMM (**hypervisor**) offre al sistema operativo guest un'interfaccia virtuale (**hypercall API**) alla quale i S.O. guest devono riferirsi per aver accesso alle risorse:

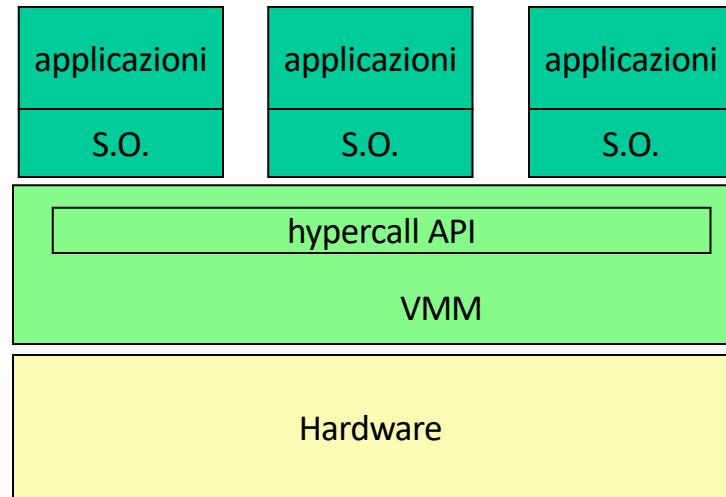
- per ottenere un servizio che richiede l'esecuzione di istruzioni privilegiate (es. accesso a dispositivi) non vengono generate interruzioni al VMM, ma viene direttamente invocata la hypercall corrispondente;
- i **kernel** dei S.O. **guest** devono quindi essere **modificati** per aver accesso all'interfaccia del particolare VMM
- la struttura del VMM è semplificata perché non deve più preoccuparsi di tradurre dinamicamente i tentativi di operazioni privilegiate dei S.O. guest.

(esempio: **xen**):

# Paravirtualizzazione

**Pro:** prestazioni migliori rispetto a fast binary translation

**Contro:** necessità di **porting** dei S.O. guest (le applicazioni rimangono invariate): soluzione preclusa a molti sistemi operativi proprietari non open source.



# Architetture Virtualizzabili

- L'uscita sul mercato di **processori con supporto nativo** alla virtualizzazione (Intel VT, AMD-V) ha dato l'impulso allo sviluppo di VMM semplificati, basati su virtualizzazione pura:
  - **No Ring Compression/Aliasing:** il s.o. guest esegue in un ring separato (livello di protezione intermedio) da quello delle applicazioni
  - **Ring Deprivilegierung:** ogni istruzione privilegiata richiesta dal s.o. guest genera un trap gestito dal VMM.

## Pro:

- **Efficienza:** non c'è bisogno di binary translation;
- **Trasparenza:** l'API presentata dall'hypervisor è la stessa offerta dal processore.

# Architetture Virtualizzabili

## Prodotti:

**Xen e Vmware:** hanno rilasciato versioni compatibili con architetture con supporto nativo alla virtualizzazione.

**Kvm:** modulo di virtualizzazione integrato nel kernel linux

## Protezione nell' Architettura x86

**Prima generazione:** Fino al 80186 la famiglia x86 non aveva alcuna capacità di protezione, non faceva cioè distinzione tra sistema operativo e applicazioni, girando ambedue con i medesimi privilegi.

➔ **Problemi:** possibilità per le applicazioni di accedere direttamente ai sottosistemi IO, di allocare memoria senza alcun intervento del sistema operativo. Problemi di sicurezza e stabilità (es. MS-DOS).

## Seconda generazione (80286 e seguenti)

Viene introdotta la **protezione** -> distinzione tra sistema operativo, che possiede controllo assoluto sulla macchina fisica sottostante e le applicazioni, che possono interagire con le risorse fisiche solo facendone richiesta al S.O. (concetto di **ring di protezione**).

**Registro CS:** i due bit meno significativi vengono riservati per rappresentare il **livello corrente di privilegio (CPL)**. -> 4 possibili ring:

Ring 0 dotato dei maggiori privilegi e quindi destinato al kernel del sistema operativo.

...

Ring 3, quello dotato dei minori privilegi e quindi destinato alle applicazioni utente.

➔ **Protezione della CPU:** non è permesso a ring diversi dallo 0 di eseguire le istruzioni privilegiate, che sono destinate solo al kernel del sistema operativo, in quanto considerate critiche e potenzialmente pericolose.

Una qualsiasi violazione di questo comportamento può provocare un'eccezione, con l'immediato passaggio al sistema operativo, il quale, catturandola, potrà correttamente gestirla, terminando ad esempio l'applicazione in esecuzione.

**Segmentazione:** ogni segmento è rappresentato da un descrittore in una tabella (GDT o LDT); nel descrittore sono indicati il livello di protezione richiesto (**PL**) e i permessi di accesso (r,w,x).

➔ **Protezione della Memoria:** una violazione dei vincoli di protezione provoca una eccezione.

Cio' accade, ad esempio, se il valore di CPL è maggiore del PL del segmento di codice contenente l'istruzione invocata.

**Uso dei Ring:** Nonostante la famiglia x86 possiede 4 ring di sicurezza, ne sono comunemente utilizzati soltanto 2:

Il ring 0 (quello a più alto privilegio) destinato al sistema operativo;

Il ring 3 nel quale sono eseguite le applicazioni.

- Gli altri due ring (cioè 1 e 2), pensati inizialmente per parti di sistema operativo che non necessitano del privilegio massimo, non sono stati utilizzati, se non in rari casi (es., IBM OS2), per mantenere la massima portabilità dei sistemi operativi verso processori con solo 2 ring di protezione.

# Funzionamento dei VMM nell'architettura x86 classica

**ring deprivilegging.** Viene dedicato il ring 0 al VMM e conseguentemente i sistemi operativi guest vengono collocati in ring a privilegi ridotti.

Vengono comunemente utilizzate due tecniche:

0/1/3: Consiste nello spostare il sistema operativo dal ring 0, dove nativamente dovrebbe trovarsi, al ring 1 a privilegio ridotto, lasciando le applicazioni nel ring 3 e installando il VMM sul ring 0. Questa tecnica non è però compatibile con sistemi operativi a 64 bit.

0/3/3: Consiste nello spostare il sistema operativo direttamente al ring applicativo, e cioè il 3, insieme alle applicazioni stesse, installando sul ring 0, come nella tecnica precedente, il VMM (ring compression)

**0/1/3:**

è ad oggi largamente la più usata nei VMM software operanti in architetture x86. Il livello applicativo (livello 3) non può danneggiare il sistema operativo virtuale sul quale è in esecuzione (livello1) andando a scrivere su porzioni di memoria ad esso dedicate.

- Le eccezioni generate sono catturate dal VMM sul ring 0 e passate da esso, praticamente invariate, al livello 1 dove è presente il sistema operativo virtualizzato.

**0/3/3:**

non essendo possibile generare alcuna eccezione (sia le applicazioni, che il sistema operativo condividono lo stesso privilegio), devono essere intrapresi meccanismi molto sofisticati con un controllo continuo da parte del VMM, funzionamento che molto si avvicina all'emulazione.

Usermode Linux (UML), prodotto open source che utilizza la tecnica 0/3/3, mostra performances ridotte.

# Problemi: Ring Aliasing

Alcune istruzioni non privilegiate, eseguite in stato utente, permettono di accedere in lettura alcuni registri del sistema la cui gestione dovrebbe essere riservata al VMM: possibilità di rilevare il proprio livello di protezione-> possibili inconsistenze.

**Esempio:** Nella famiglia x86 l'utilizzo dell'istruzione **PUSH** applicata al registro CS, che permette di salvarne il contenuto nello stack, può essere eseguita in qualunque ring. Ma il CS contiene il CPL (Current Privilege Level). Il sistema operativo virtualizzato si trova pertanto nelle condizioni di rilevare direttamente su quale ring di protezione esso stia girando.

Per esempio: in un ambiente virtualizzato 0/1/3, il SO guest sarebbe in grado di capire che il ring di protezione che lo ospita non è lo 0 (dove si aspetta di essere), bensì l'1, dove il VMM lo ha confinato.

# Problemi: mancate eccezioni

Nell'architettura x86 esistono istruzioni privilegiate che, se eseguite in  $\text{ring} > 0$ , non provocano un'eccezione, ma vengono ignorate non consentendo quindi l'intervento trasparente del VMM, o in alcun casi provocano il crash del sistema.

Esempio: istruzione **popf**: trasferisce un insieme di flag dallo stack nel registro flag.

Se eseguita in modo kernel, popf assegna a tutti i flag (sia ALU che flag di sistema) i valori dati.

Se viene eseguita in user mode, invece, non ha effetto e nessun trap viene generato.

Quindi, se un kernel guest (in  $\text{ring} > 0$ ) tenta di disabilitare le interruzioni (resettando il flag di abilitazione con popf), l'architettura non notifica alcuna eccezione e il VMM non recepisce che non deve trasmettere interruzioni al guest.

# Gestione di VM

- Compito fondamentale del VMM è la gestione delle macchine virtuali:
  - **Creazione**
  - **Spegnimento/accensione**
  - **Eliminazione**
  - **Migrazione live**

# Stati di una VM

Una Macchina virtuale può trovarsi nei seguenti stati:

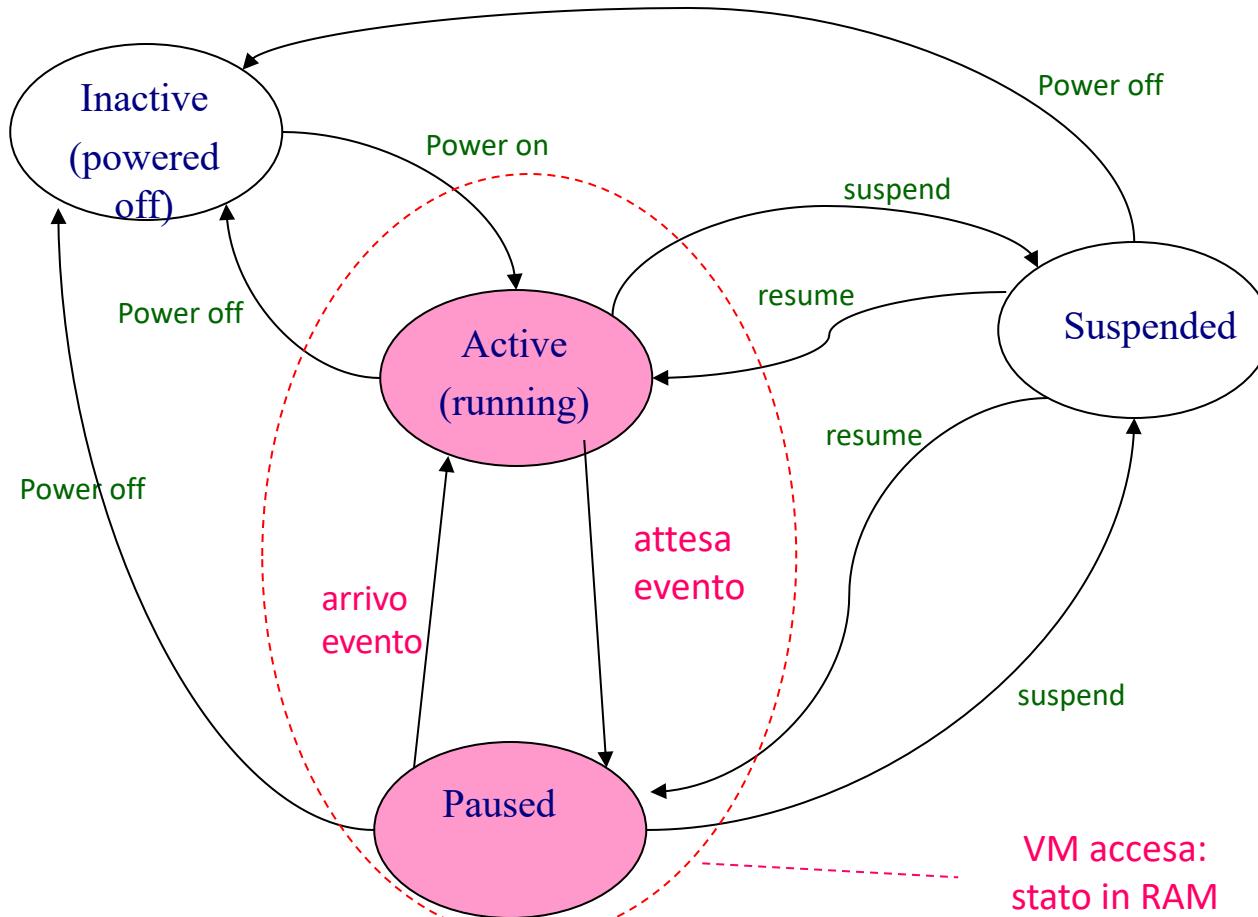
**Running** (o attiva): la macchina è accesa e occupa memoria nella ram del server sul quale è allocata.

**Inactive** (Powered Off): la macchina è spenta ed è rappresentata nel file system tramite un file immagine.

**Paused**: la macchina virtuale è in attesa di un evento (es. I/O richiesto da un processo nell'ambiente guest).

**Suspended**: la macchina virtuale è stata sospesa dal VMM; il suo stato e le risorse utilizzate sono salvate nel file system (file immagine). L'uscita dallo stato di sospensione avviene tramite l'operazione resume da parte del VMM.

# VM: Diagramma degli stati



# Migrazione di VM

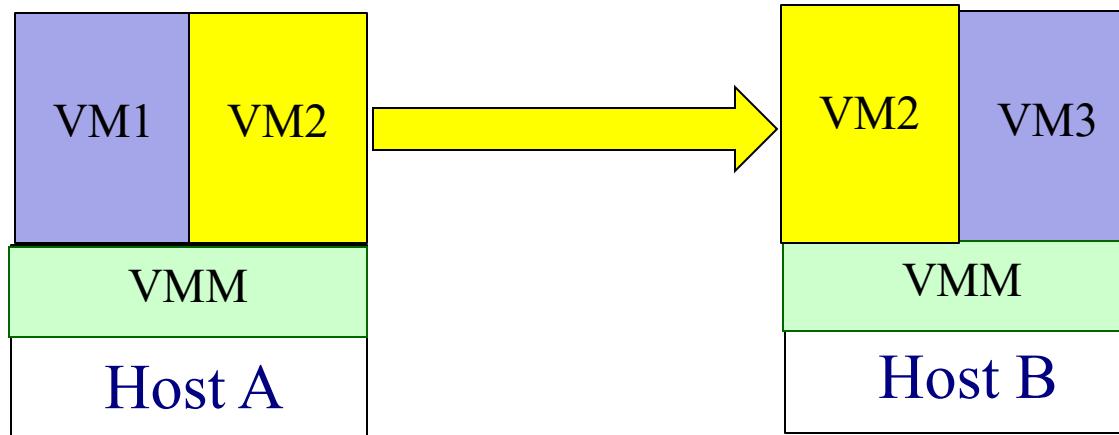
In datacenter di server virtualizzati è sempre più sentita la necessità di una gestione agile delle VM per fare fronte a:

- Variazioni dinamiche del carico: load balancing, consolidamento
- Manutenzione “online” dei server
- Gestione finalizzata al risparmio energetico
- Tolleranza ai guasti/disaster recovery

In tutti questi casi la possibilità di muovere VM tra server è un meccanismo fondamentale per la soluzione.

# Migrazione live

- Le macchine virtuali possono essere spostate da un server fisico a un altro senza essere spente.

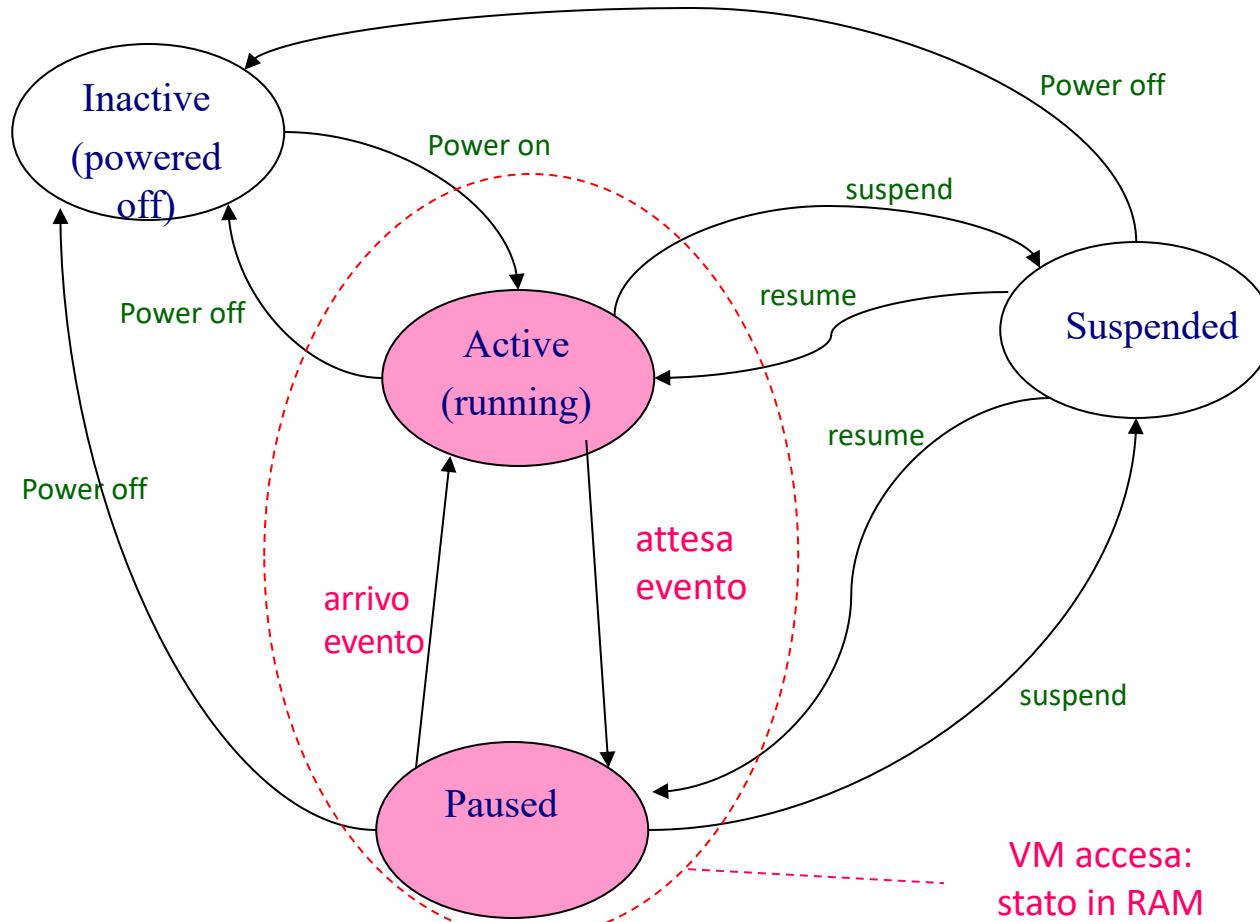


# Suspend/resume

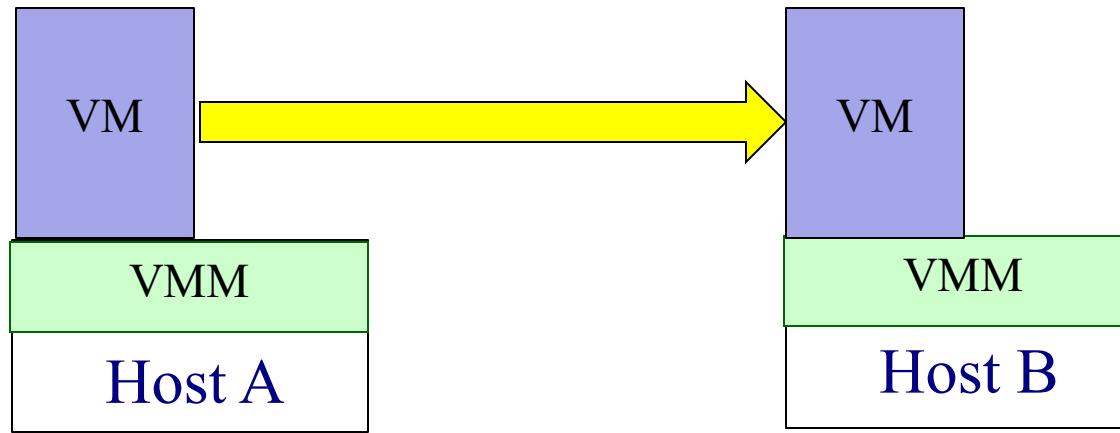
- Il VMM può mettere in stand-by una VM tramite l'operazione **suspend**: lo stato della macchina viene salvato in memoria secondaria.
- Una VM suspended può riprendere l'esecuzione, a partire dallo stato in cui si trovava quando è stata sospesa tramite l'operazione **resume**. Lo stato salvato viene ripristinato in memoria centrale

Poichè una VM è quasi completamente indipendente dal server fisico su cui è allocata, la resume può avvenire su un nodo diverso da quello in cui era prima della sospensione.

# VM: Diagramma degli stati



# Realizzazione della live migration



E' desiderabile minimizzare:

- **Downtime** (tempo in cui la macchina non risponde alle richieste degli utenti)
- **Tempo di migrazione**
- **Consumo di banda**

Se il file system è condiviso tra A e B, non c'è bisogno di copiare il file immagine.

# Soluzione: Precopy

- La migrazione viene realizzata in 6 passi:
  1. **Pre-migrazione:** individuazione della VM da migrare e dell'host (B) di destinazione
  2. **Reservation:** viene inizializzata una VM (container) sul server di destinazione
  3. **Pre-copia iterativa delle pagine:** viene eseguita una copia nell'host B di tutte le pagine allocate in memoria sull'host A per la VM da migrare; successivamente vengono iterativamente copiate da A a B tutte le pagine modificate (dirty pages) fino a quando il numero di dirty pages è inferiore a una soglia data.
  4. **Sospensione della VM e copia dello stato + dirty pages** da A a B.
  5. **Commit:** la VM viene eliminata dal server A
  6. **Resume:** la VM viene attivata nel server B

*VM running normally on Host A*

**Stage 0: Pre-Migration**

Active VM on Host A

Alternate physical host may be preselected for migration

Block devices mirrored and free resources maintained

**Stage 1: Reservation**

Initialize a container on the target host

*Overhead due to copying*

**Stage 2: Iterative pre-copy**

Enable shadow paging

Copy dirty pages in successive rounds.

*Downtime  
(VM out of service)*

**Stage 3: Stop and copy**

Suspend VM on host A

Generate ARP to redirect traffic to Host B

Synchronize all remaining VM state to Host B

*VM running normally on Host B*

**Stage 4: Commitment**

VM state on Host A is released

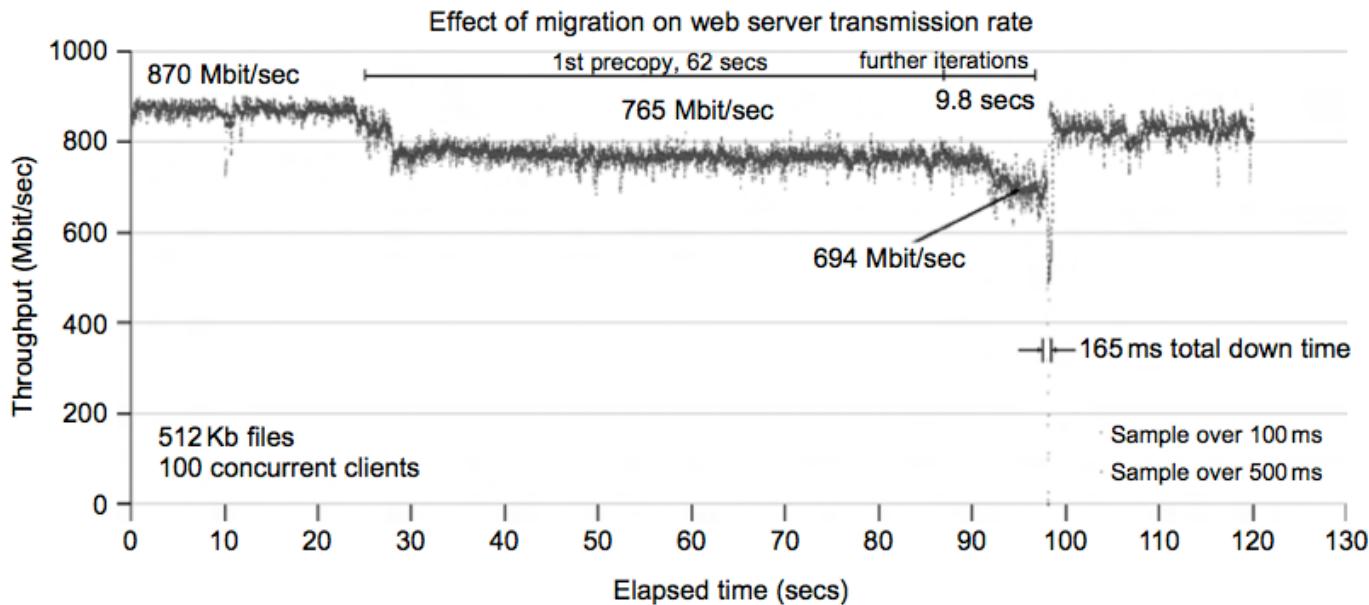
**Stage 5: Activation**

VM starts on Host B

Connects to local devices

Resumes normal operation

# Precopy: risultati sperimentali



# Realizzazione migrazione

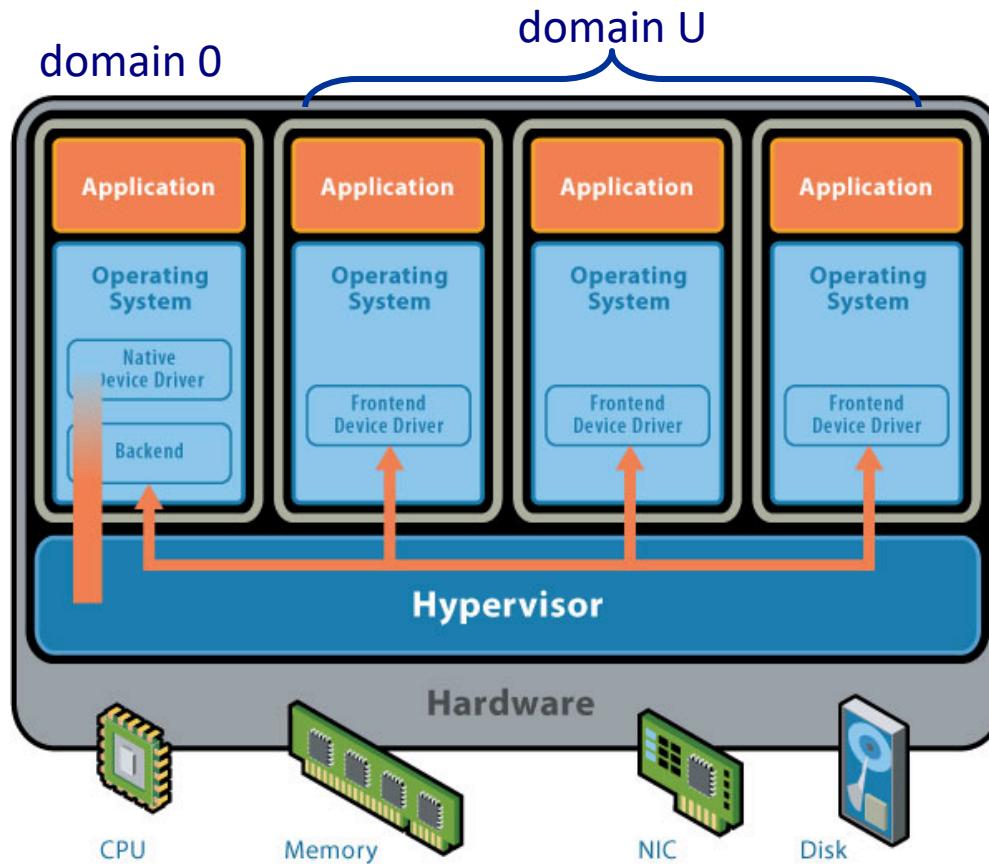
- In alternativa a precopy:

👉 **Post-copy**: la macchina viene sospesa e vengono copiate (non iterativamente) pagine e stato. Tempo di migrazione più basso, ma downtime molto più elevato.

# xen

- VMM open source che opera secondo i principi della paravirtualizzazione (Università di Cambridge, 2003).
- Porting di Linux su XEN (XenoLinux, Suse, CentOs, ...). Modifica del Kernel di Linux per dialogare con le API di XEN pari a circa 3000 linee di codice (1,36% del totale).
- Porting di Windows XP (XenoXP) in collaborazione con Microsoft. Lavoro non completato.
- Molte distribuzioni di Linux (Suse, Red Hat, CentOs, Debian....) offrono pacchetti precompilati per installare XEN.
- Acquisito da Citrix nel 2007, che lo usa come piattaforma a supporto dei propri prodotti di virtualizzazione e cloud computing

# Architettura di xen



# Organizzazione

- il vmm (hypervisor) si occupa della virtualizzazione della CPU, della memoria e dei dispositivi per ogni macchina virtuale (domain<sub>i</sub>)
- Xen dispone di un'interfaccia di controllo in grado di gestire la divisione di queste risorse tra i vari domini.
- L'accesso a questa interfaccia di controllo è ristretta; può essere controllata solamente utilizzando una VM dedicata a questo compito: **domain 0**. In questo dominio viene eseguita l'applicazione software che gestisce il controllo di tutto il sistema.
- Il software di controllo nel domain 0 (dom0) è separato dallo stesso hypervisor -> separazione dei meccanismi dalle politiche.

# Xen: realizzazione

- Virtualizzazione della CPU
- Virtualizzazione della Memoria
- Gestione I/O

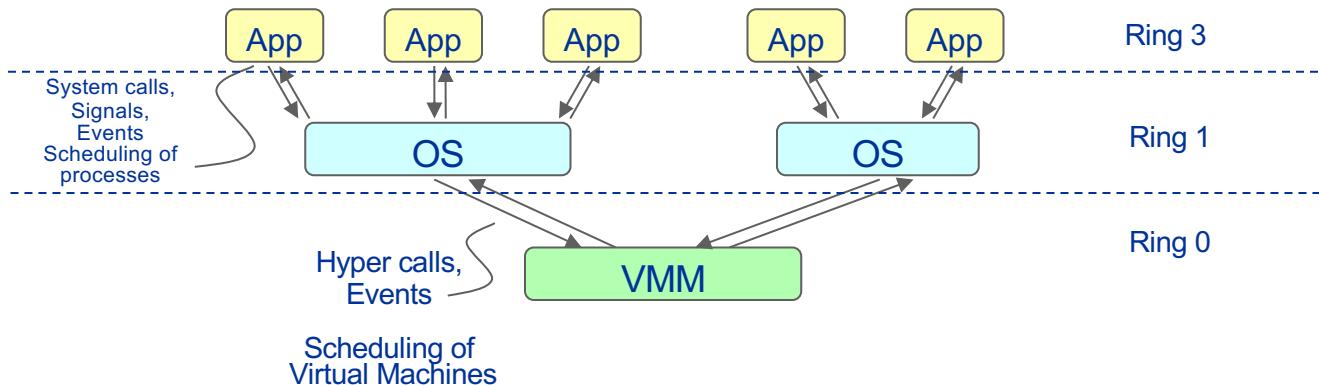
# Xen – caratteristiche

## ■ Paravirtualizzazione:

- Le macchine virtuali eseguono direttamente le istruzioni non privilegiate
- L' esecuzione di istruzioni privilegiate viene delegata al VMM tramite chiamate al VMM (hypercalls).

## ■ Protezione (x86):

- I sistemi operativi guest OS sono collocati nel ring 1
- VMM collocato nel ring 0

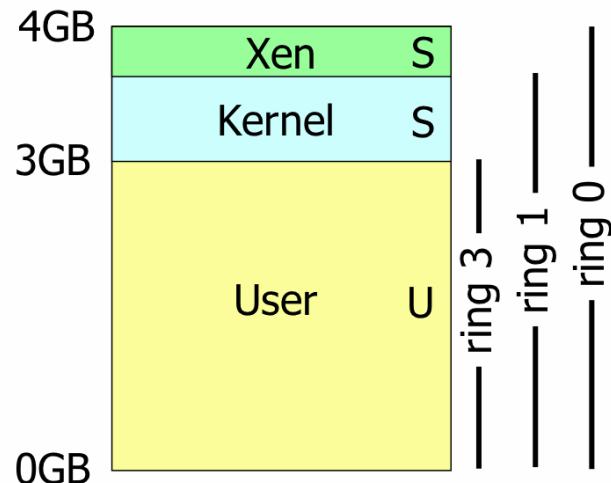


# Xen: gestione della memoria e paginazione

- Gestione della Memoria:
  - i SO guest gestiscono la memoria virtuale, mediante i tradizionali meccanismi/politiche di paginazione
  - X86: page faults gestiti direttamente a livello HW (TLB)
- Soluzione adottata: le tabelle delle pagine delle VM:
  - vengono mappate nella memoria fisica dal VMM (shadow page tables);
  - non possono essere accedute in scrittura dai kernel guest, ma solo dal VMM;
  - sono accessibili in modalità read-only anche dai guest;  
in caso di necessità di update, interviene il VMM che valida le richieste di update dei guest e le esegue.
- Memory split:
  - Xen risiede nei primi 64 MB del virtual address space di ogni VM:
  - In questo modo non è necessario effettuare un TLB “flush” per ogni hypercall -> maggiore efficienza

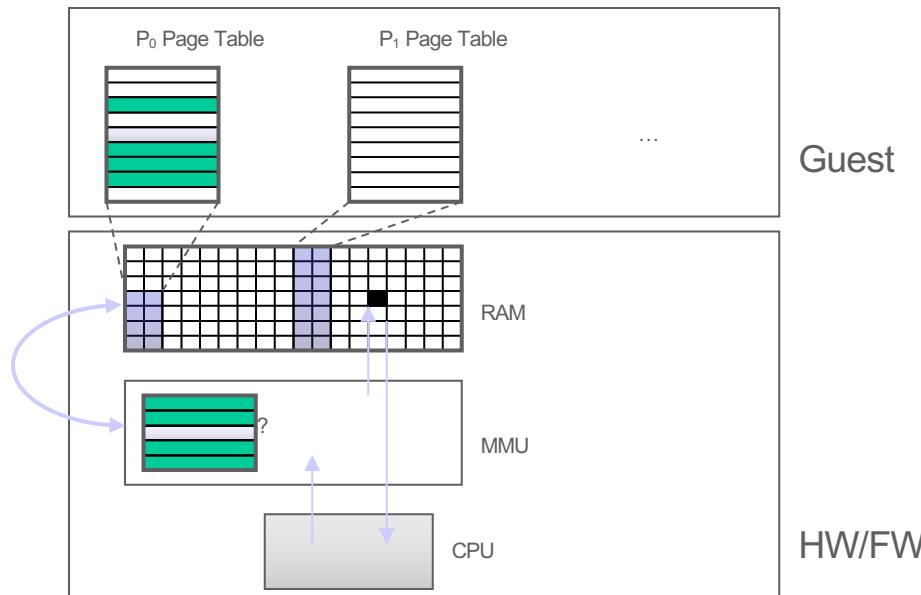
# Protezione

- **Memory split:** lo spazio di indirizzamento virtuale per ogni VM è strutturato in modo da contenere xen e il kernel in segmenti separati.



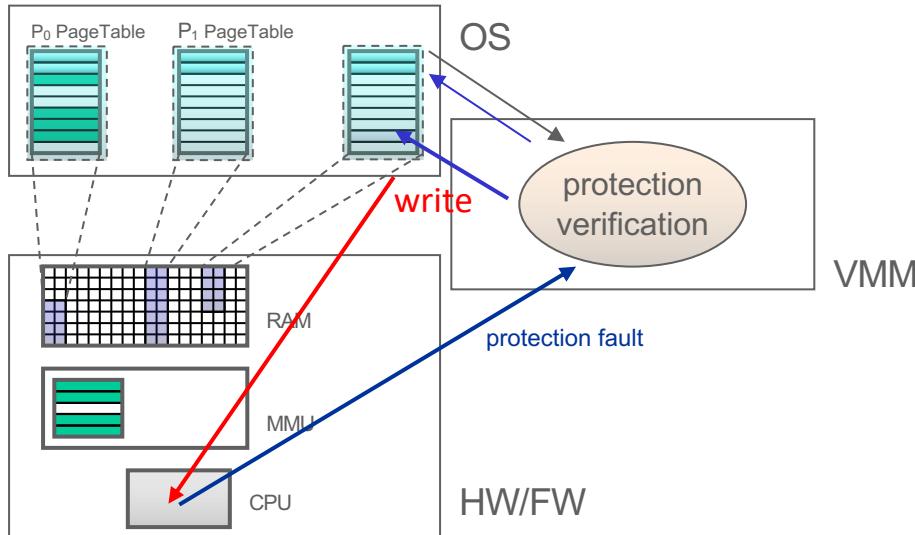
# Xen: gestione della memoria

- I guest OS si occupano della paginazione, delegando al VMM la scrittura delle page table entries.
- Le tabella della pagine devono essere create e verificate dal VMM su richiesta dei guest: una volta create, le tabelle rimangono Read-only per il guest.



# Xen – Creazione di un processo

- Il SO guest richiede una nuova tabella delle pagine al VMM:
  - Alla tabella vengono aggiunte le pagine appartenenti al segmento di xen.
  - Xen registra la nuova tabella delle pagine e acquisisce il diritto di scrittura esclusiva.
  - ogni successiva update da parte del guest provocherà un **protection-fault**, la cui gestione comporterà la verifica e l'effettivo aggiornamento della PT.



## Gestione della memoria: balloon process

La paginazione è a carico dei guest.

👉 Occorre un meccanismo efficiente che consenta al VMM di reclamare ed ottenere, in caso di necessità, dalle altre macchine virtuali pagine di memoria meno utilizzate.

**Soluzione:** su ogni macchina virtuale è in esecuzione un processo (**balloon process**) che comunica direttamente con il VMM, e che viene interpellato ogni volta che il VMM ha bisogno di ottenere nuove pagine ad esempio per l'attivazione di una VM.

## Gestione della memoria: balloon process

In caso di necessità di pagine per una nuova VM, interviene il VMM per chiedere ad altre VM di liberare memoria.

Il VMM, comunicando direttamente con il balloon process di altre VM, chiede ad essi di “gonfiarsi”, cioè di richiedere al proprio s.o. (guest) altre pagine.

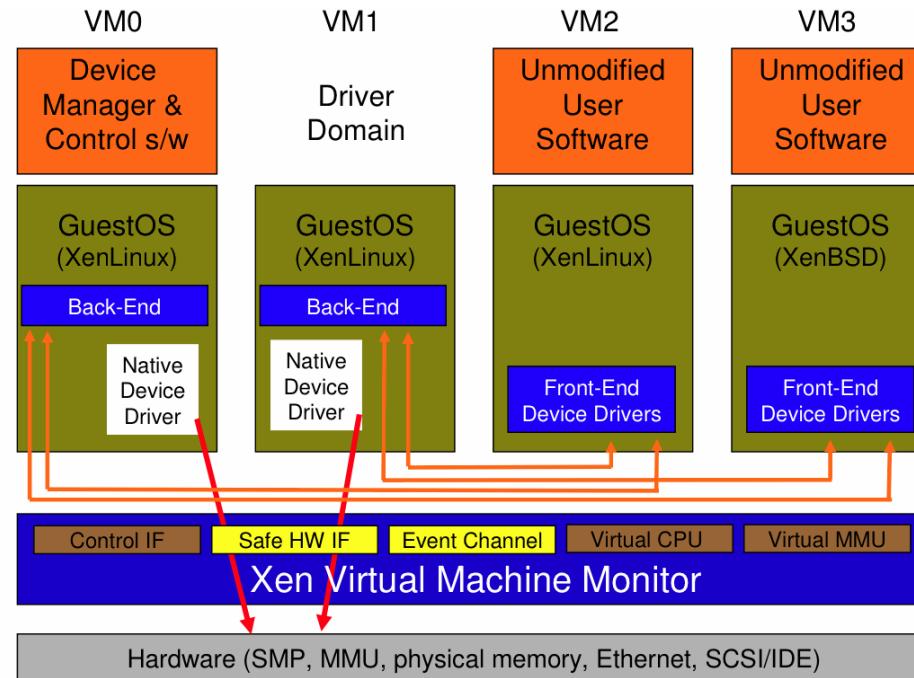
La richiesta del balloon process provoca da parte del S.O. guest **l'allocazione di nuove pagine al balloon process**, che, una volta ottenute, le cede quindi al VMM.

# Xen – Virtualizzazione della CPU

- Il VMM definisce un' architettura virtuale simile a quella del processore, nella quale, però, le istruzioni privilegiate sono sostituite da opportune hypercalls:
  - L' invocazione di una hypercall determina il passaggio da guest a xen (ring1 -> ring 0)
  - I kernel dei sistemi guest devono essere modificati di conseguenza
- Il VMM si occupa dello scheduling delle macchine virtuali: **Borrowed Virtual Time scheduling algorithm**
  - Si basa sulla nozione di virtual-time
  - algoritmo general-purpose, che consente, in caso di vincoli temporali stringenti (es. applicazioni time dependent, TCP/IP, servizi RT..) di ottenere schedulazioni efficienti
  - Due clock:
    - real-time (tempo del processore, inizia al boot)
    - virtual-time (associato alla VM, avanza solo quando la VM esegue)
    - I tempi vengono comunicati ai guest tramite eventi.

Disponibilità di altri scheduler ([wiki.xenproject.org](http://wiki.xenproject.org))

# Xen -Virtualizzazione dell'I/O



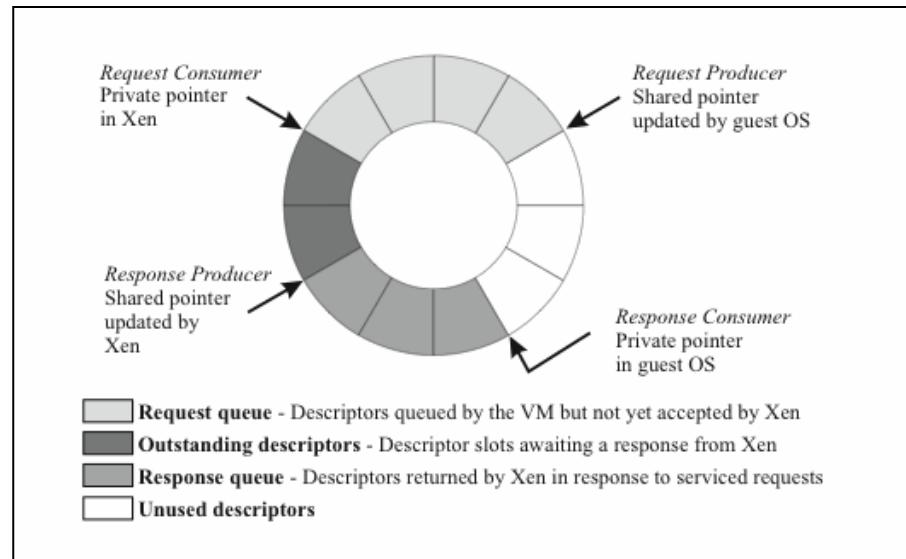
# Xen – Virtualizzazione dell' I/O

## ■ Soluzione adottata:

- Back-end driver: per ogni dispositivo, il suo driver è isolato all'interno di una particolare macchina virtuale (tipicamente Dom0). Accesso diretto all'HW.
- Front-end driver: ogni guest prevede un driver virtuale semplificato che consente l'accesso al device tramite il backend:

**Pro:** portabilità (v. migrazione), isolamento, semplificazione del VMM.

**Contro:** necessità di comunicazione con il back-end  
-> asynchronous I/O rings.

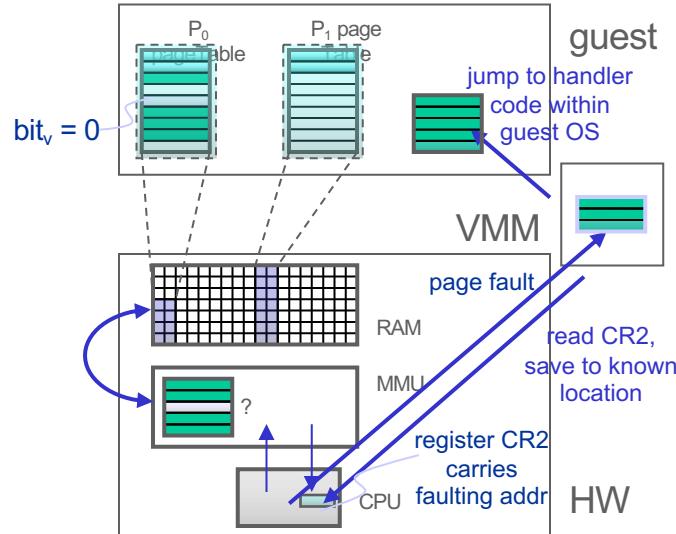


## Xen – gestione interruzioni e eccezioni

- La gestione delle interruzioni viene virtualizzata in modo molto semplice:
  - il vettore delle interruzioni punta direttamente alle routine del kernel guest: ogni interruzione viene gestita direttamente dal guest.
  - Un caso particolare riguarda il page-fault:
    - La gestione non può essere delegata completamente al guest, perché richiede l'accesso al registro **CR2** (contenente l'indirizzo che ha provocato il page fault)
    - Ma CR2 è accessibile solo nel ring 0! -> la gestione del page fault deve coinvolgere il VMM.

# Xen – gestione del page-fault

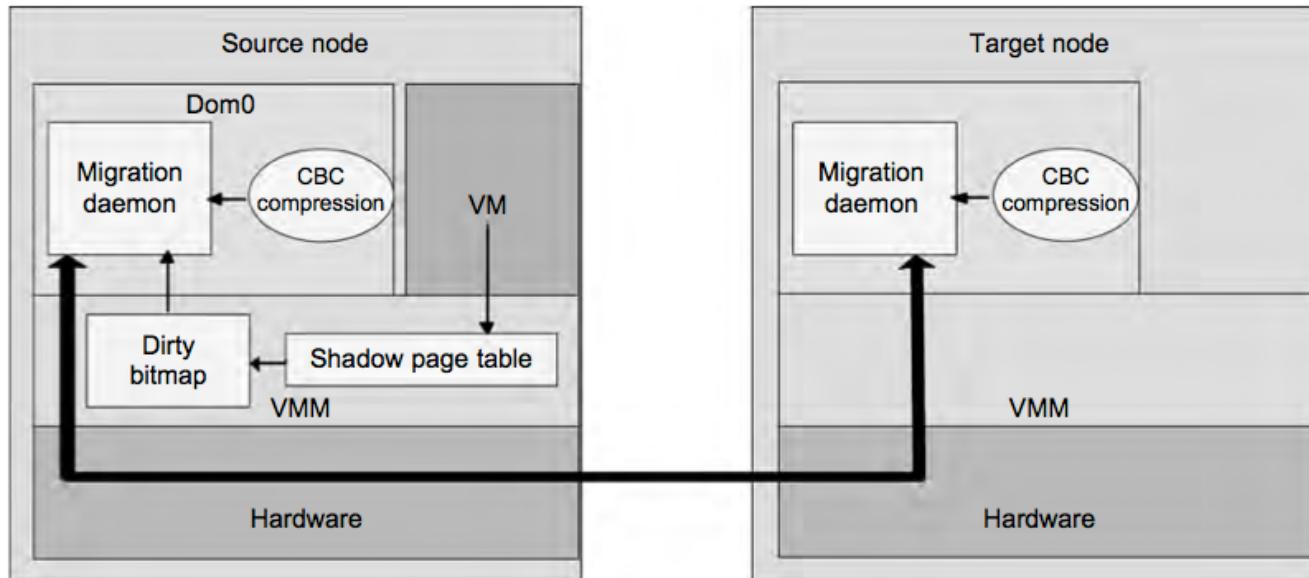
- In questo caso l' handler punta a codice xen (esecuzione nel ring 0)
- La routine di gestione eseguita da xen legge il contenuto di CR2 e lo copia in una variabile dello spazio del guest; successivamente viene trasferito (jump) il controllo al guest, che andrà finalmente a gestire il page fault.



## Live migration in xen

- Migrazione **guest-based**. Il comando di migrazione viene eseguito da un demone di migrazione nel domain0 del server di origine della macchina da migrare.
- Realizzazione basata su **pre-copy**. Le pagine da migrare vengono compresse per ridurre l'occupazione di banda.

# Live migration in xen



[Hwang, Fox, Dongarra: Distributed and Cloud Computing, Morgan Kaufmann, 2012]

# Riferimenti Bibliografici

- P. Barham, B. Dragovic, K. Fraser, S. Hand, T, Harris, A. Ho, R. Neugebauer , I. Pratt, A. Warfield  
Xen and the Art of Virtualization  
In Proceedings of the ACM Symposium on Operating Systems Principles, 2003
- J. N. Matthews; E. M. Dow et. Al.  
Running Xen: A Hands-On Guide to the Art of Virtualization, Prentice Hall, 2008.
- K. J. Duda, D.R. Cheriton  
Borrowed Virtual Time (BVT) Scheduling: Supporting latency-sensitive Threads in a general-purpose Scheduler  
In Proceedings of the 17th ACM SIGOPS Symposium on Operating System Principles, pages 261-276, 1999.
- <https://wiki.xenproject.org>
- G. J. Popek and R.P. Goldberg  
Formal Requirements for Virtualizable Third Generation Architectures. Communications of the ACM 17 (7): 412 –421, 1974.
- Hwang, Fox, Dongarra: Distributed and Cloud Computing, Morgan Kaufmann, 2012