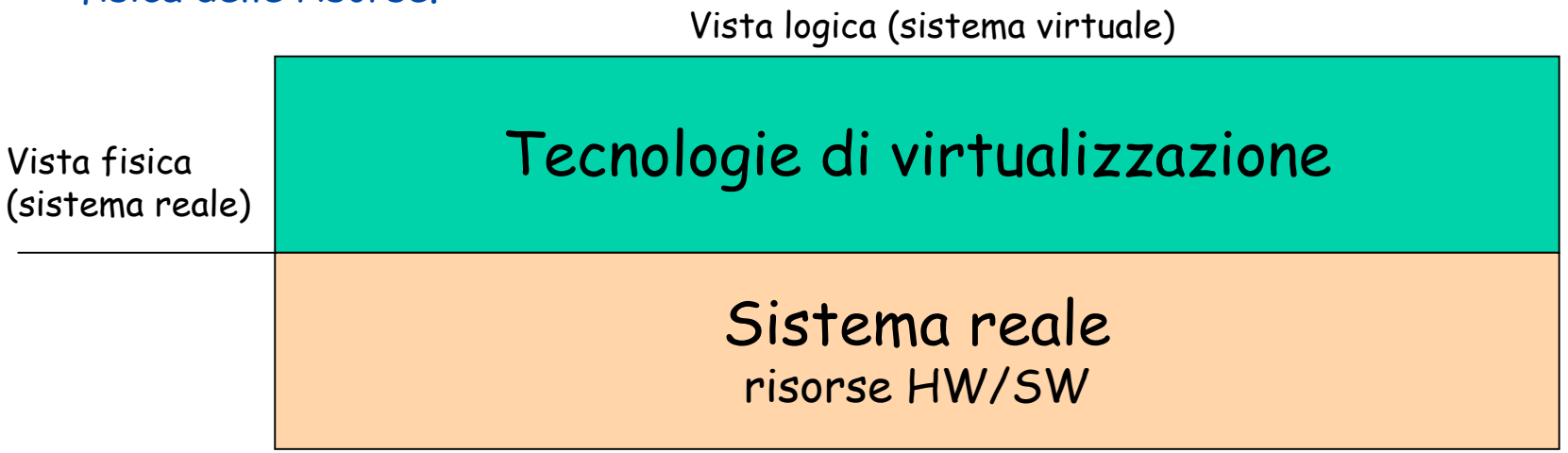
Sistemi per la Virtualizzazione

Virtualizzazione

Dato un sistema caratterizzato 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.

Tecnologie di virtualizzazione

Obiettivo: disaccoppiare il comportamento delle risorse hardware e software di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica.

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.

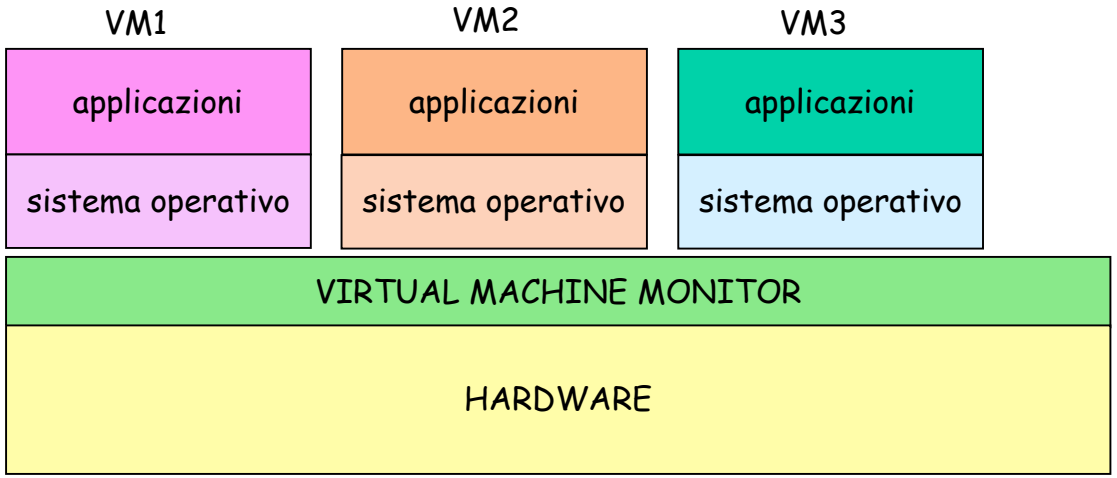
Linguaggi di Programmazione. La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura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

- Interpreti (esempio Java Virtual Machine)
- Compilatori

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 di Sistema. Una singola piattaforma hardware viene condivisa da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale.

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 macchina virtuale è costituita oltre che dall'applicazione che in essa viene eseguita, anche dal sistema operativo utilizzato.



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

Emulazione

Esecuzione di programmi compilati per un **certo 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, 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 mediamente molto 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 come **Virtual PC per MAC** e **QEMU** utilizzano questa tecnica per attuare la virtualizzazione.

Virtual PC

• Software di emulazione che consente a computer con **sistema operativo Microsoft windows** l'esecuzione di **sistemi operativi diversi**, come varie versioni di Windows 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.

• **La successiva introduzione dei processori Intel** nei computer Apple ha tolto parecchio interesse pratico all'utilizzo di Virtual PC da parte degli utenti Macintosh, poiché è divenuto possibile riavviare (*dual boot*) tali elaboratori in **Windows XP "nativo"** o utilizzare software di virtualizzazione più performanti come Paralleles Desktop for Mac.

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*. È simile a Virtual PC, ma più veloce nell'emulazione delle architetture x86.

. **Qemu**, inizialmente, era un progetto che si prefiggeva di emulare solo il microprocessore x86 su un sistema GNULinux. Tuttora, **Qemu** è in grado di emulare sistemi x86, AMD64, Power PC, MIPS e ARM

MAME

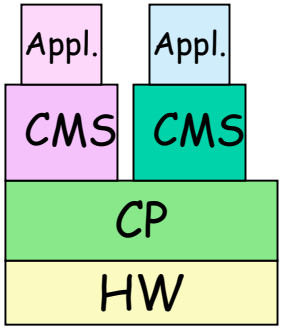
• (acronimo per Multiple Arcade Machine Emulator) è un software per personal computer sviluppato inizialmente per MS-DOS e in seguito per quasi tutte le macchine e sistemi operativi in circolazione, in grado di **caricare ed eseguire il codice binario originale delle ROM** dei videogiochi da bar (*arcade*), **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 secondo il *paradigma dell'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.

Cenni storici

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

- **Fine anni 90:** necessità di **razionalizzazione**
  - nuovi sistemi di virtualizzazione per l'architettura Intel x86 (1999, VMware)

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, 1974):

1. **Ambiente di esecuzione per i programmi sostanzialmente identico a quello della macchina reale.**

   Uniche differenze legate alle dipendenze temporali (più macchine virtuali concorrenti)

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

- **Livello** dove è collocato il VMM:
  - VMM di sistema: eseguono direttamente sopra l'hardware dell'elaboratore (es. vmware esx, xen)
  - 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 completa** (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.

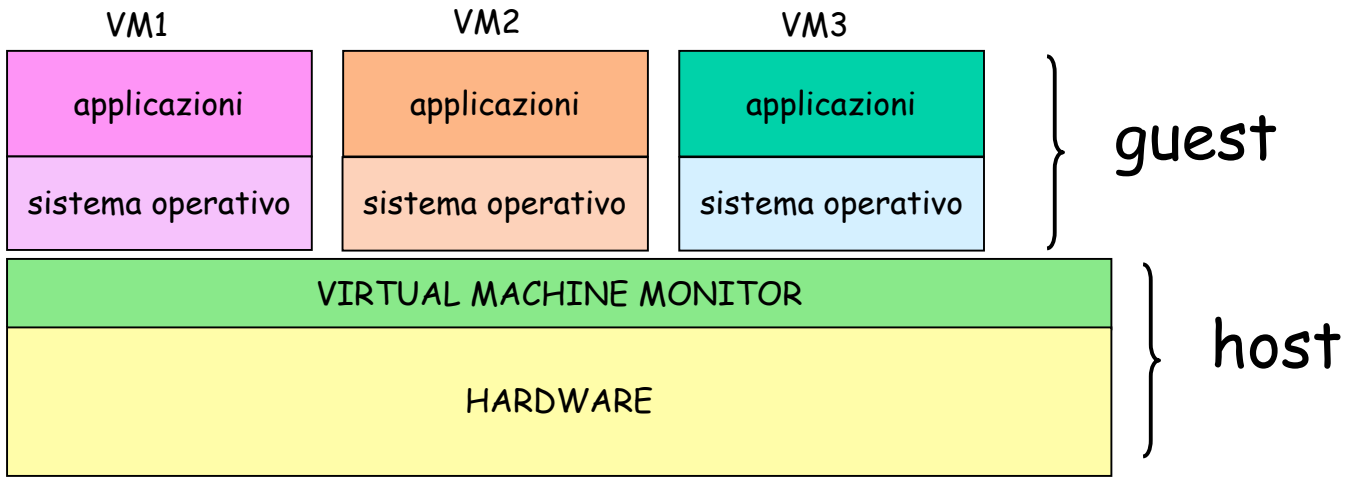
le funzionalità di virtualizzazione vengono integrate in un sistema operativo leggero, costituendo un unico sistema posto direttamente sopra l'hardware dell'elaboratore.

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

Esempi di VMM di sistema: Vmware ESX, xen, kvm VirtualIron.

Host: piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica, l'eventuale sistema operativo ed il VMM.

Guest: la macchina virtuale. Comprende applicazioni e sistema operativo



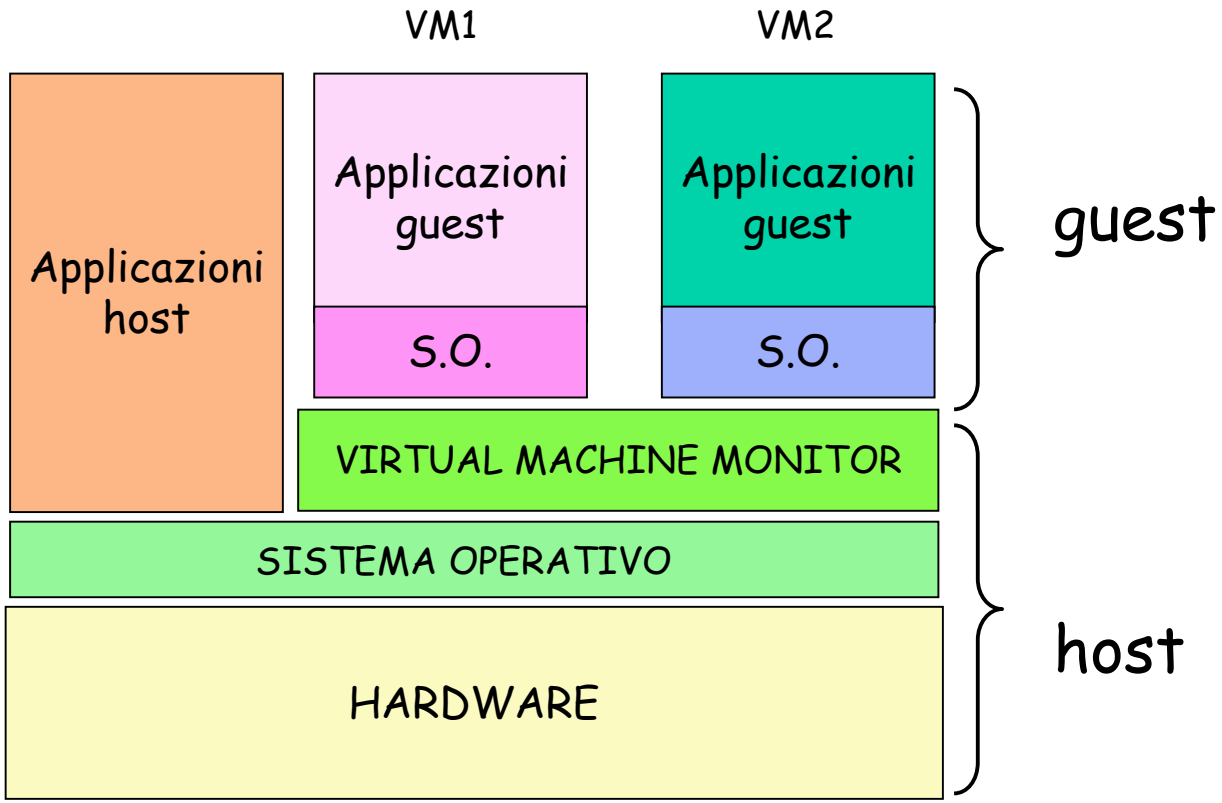
VMM di Sistema

VMM ospitato

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

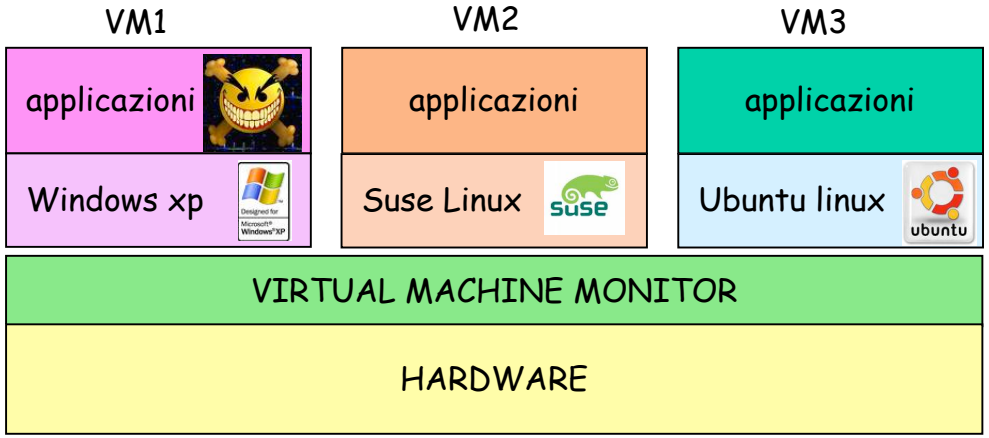
- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O.(es. scheduling, gestione delle risorse.).
- Peggiore la performance.

Prodotti: User Mode Linux, VMware Server/Player, Microsoft Virtual Server , Parallels



VMM ospitato

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:
  - possibilita` di effettuare testing di applicazioni preservando l'integrita` degli altri ambienti e del VMM.
  - Sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

Vantaggi della virtualizzazione

- **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
    - disaster recovery
    - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo "autonomico" a situazioni di sbilanciamento

Vantaggi della virtualizzazione

- **In ambito didattico:** invece di assegnare ad ogni studente un account su una macchina fisica, si assegna una macchina virtuale.

DEIS Virtual Lab. La Facoltà di Ingegneria (DEIS) sta realizzando un laboratorio di macchine virtuali che offrirà ad ogni studente una macchina virtuale personale da amministrare autonomamente:

- possibilita` di esercitarsi senza limitazioni nelle tecniche di amministrazione e configurazione del sistema;
- possibilita` di installazione e testing di nuovi sistemi operativi, anche prototipali, senza il rischio di compromettere la funzionalita` del sistema.
- possibilita` di testing di applicazioni potenzialmente pericolose senza il rischio di interferire con altri utenti/macchine;
- possibilita` di trasferire le proprie macchine virtuali in supporti mobili (es: penne USB, per continuare le esercitazioni sul computer di casa).

Dotazione hw: 5 server Intel-VT xeon (2 processori quadcore), storage unit CORAID 12TB

Software: xen, linux CentOS.

VMM di sistema: realizzazione

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

➡ solo il VMM opera nello stato supervisore, mentre il sistema operativo e le applicazioni (la macchina virtuale) operano nello stato utente.

Problemi:

- ***ring deprivileging***: il s.o. della macchina virtuale esegue in uno stato che non gli e` proprio (esecuzione di system call)
- ***ring compression:*** applicazioni e s.o. della macchina virtuale eseguono allo stesso livello: necessita` di protezione tra spazio del s.o. e delle applicazioni.

Ring deprivileging:

Le istruzioni **privilegiate** richieste dal sistema operativo nell'ambiente guest non possono essere eseguite (richiederebbero lo stato supervisore).

Possibile Soluzione:

- Se il guest tenta di eseguire un'istruzione privilegiata, la CPU **notifica un'eccezione al VMM** e gli trasferisce il controllo (trap): il VMM controlla la correttezza della operazione richiesta e **ne emula** il comportamento.

- Le istruzioni non privilegiate possono essere eseguite direttamente dall'hardware senza alcun intervento da parte della CPU (**esecuzione diretta**).

Esempio: tentativo di esecuzione dell'istruzione privilegiata che disabilita le interruzioni da parte del S.O. guest.

- Il VMM riceve la notifica di tale richiesta e ne emula il comportamento atteso sospendendo la consegna degli interrupt solamente per la macchina virtuale (emulazione).
- Se la richiesta della macchina virtuale fosse eseguita direttamente sul processore sarebbero disabilitati gli interrupt per tutti i sistemi ed il VMM non potrebbe riguadagnare il controllo della CPU.

➡ Un'architettura CPU si dice **naturalmente virtualizzabile** se prevede l'invio di notifica allo stato supervisore **per ogni istruzione privilegiata** eseguita 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.
- supporto nativo all'esecuzione diretta.

Non tutte le architetture sono naturalmente virtualizzabili ! IA32:

- **Alcune istruzioni privilegiate** di questa architettura eseguite in **stato utente** non provocano una trap, ma vengono ignorate non consentendo quindi l' intervento trasparente del VMM, o in alcun casi provocano il crash del sistema.
- ***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 -> possibili inconsistenze.

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

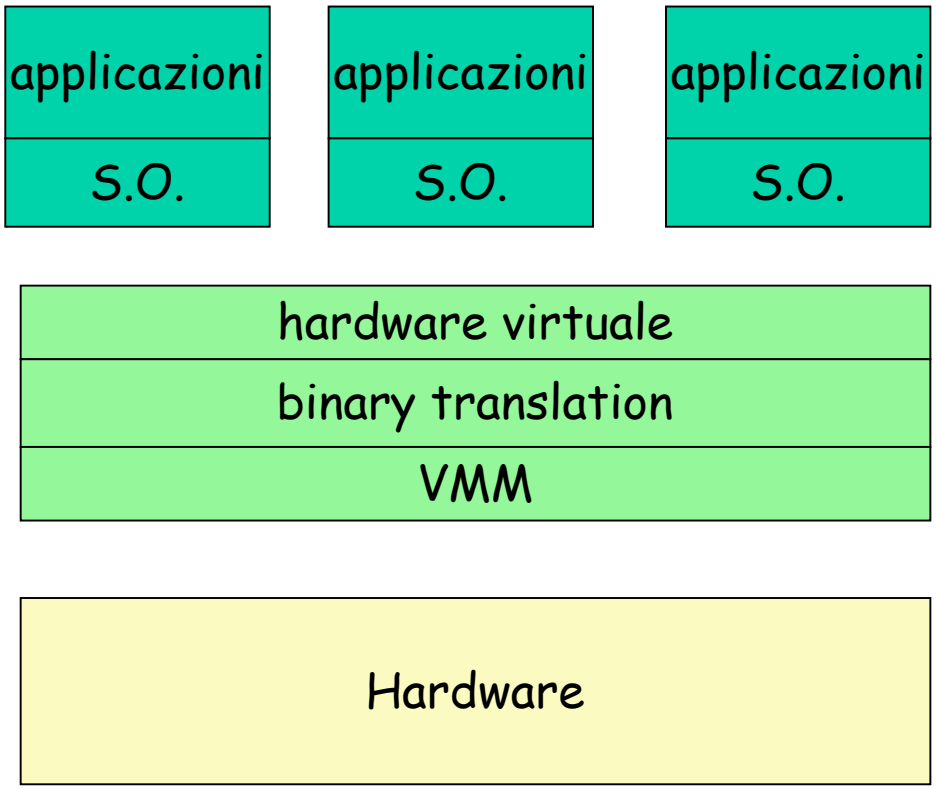
Architetture non virtualizzabili

Soluzioni:

- Virtualizzazione pura: ***fast binary translation***
- Paravirtualizzazione

Fast binary translation. (es.Vmware): il VMM **scansiona dinamicamente il codice prima della sua esecuzione** per sostituire a **run time** blocchi contenenti **istruzioni problematiche** in blocchi equivalenti dal punto di vista funzionale e contenenti istruzioni per la notifica di eccezioni al VMM.

- I blocchi tradotti sono eseguiti direttamente sull'hw e conservati in una cache apposita per riusi futuri

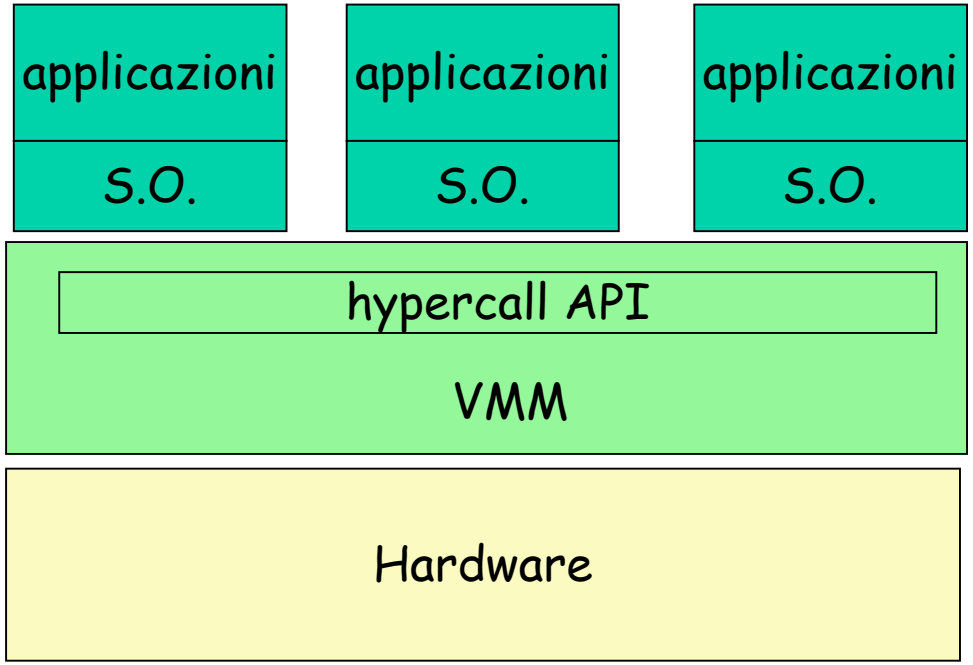


Virtualizzazione pura mediante Fast Binary Translation (architettura non virtualizzabile)

Pro: ogni macchina virtuale è una esatta replica della macchina fisica
➡ possibilità di installare gli stessi s.o. di architetture non virtualizzate
Contro: la traduzione dinamica è costosa !

Paravirtualizzazione (es. xen): 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:

- **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.
- per ottenere un servizio che richiede l'esecuzione di istruzioni privilegiate (es. accesso a dispositivi) non vengono generate interruzioni al VMM, **ma viene invocata la hypercall corrispondente**.



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 commerciali 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 Deprivileging:** 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 già rilasciato nuove versioni compatibili con architetture virtualizzabili.

Kvm: modulo di virtualizzazione integrato nel kernel linux

Virtual Iron: prodotto derivato da xen, esplicitamente sviluppato per architetture Intel VT & AMD-V.

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 introdotto il concetto di **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 singificativi 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.

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

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.

Protezione della Memoria: ogni segmento di memoria viene anch'esso protetto mediante:

permessi di scrittura, lettura, esecuzione;
livello di privilegio (PL)

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 possieda 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 deprivileging. Viene dedicato il ring 0 al VMM e conseguentemente i sistemi operativi guest vengolo 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.

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 si 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.

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.

Il livello di privilegio è scritto nel registro CS, il registro di segmento, del microprocessore x86. Più precisamente il CPL (Current Privilege Level) è scritto nei due bit meno significativi di questo registro. Accedere a questa informazione permette quindi di capire a quale livello di privilegio il codice stia effettivamente eseguendo.

Nella famiglia x86 l'utilizzo dell'istruzione PUSH applicata a questo registro, che permette di salvarne il contenuto nello stack, non porta alla generazione di alcuna eccezione indipendentemente dal livello di privilegio a cui viene eseguita. Il sistema operativo virtualizzato, si trova nelle condizioni di capire effettivamente, senza intermediazione alcuna, su quale ring di protezione esso stia girando.

In un ambiente virtualizzato, quindi, sarebbe in grado di capire che il ring di protezione che lo ospita non è lo 0, dove si aspetta di trovarsi, bensì l'1, dove il VMM lo ha confinato.

Mancate eccezioni nell'esecuzione di specifiche istruzioni

Nell'architettura x86 esistono **istruzioni privilegiate che, se eseguite in 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: Istruzioni che permettono di accedere indirettamente a strutture da cui risulta possibile ricavare lo stato virtualizzato del sistema.

Tra queste la SGDT, che permette di leggere e salvare in memoria la Global Descriptor Table (GDT) che è una struttura gestita a livello di ring 0 ed è unica a livello di microprocessore, nella quale sono contenuti, per tutti i segmenti di memoria, il corrispondente livello di protezione in scrittura, lettura ed esecuzione.

Poiché l'istruzione SGDT riporta la posizione in memoria della vera GDT, che nel caso di un ambiente virtuale coincide con quella del VMM, è evidente che la locazione non potrà coincidere con quella scelta ed impostata dal sistema operativo guest in fase di boot.

• Altro esempio è LDT (Local Descriptor Table). Rappresenta per un singolo processo la tabella in cui sono specificati tutti i privilegi di segmento. Le LDT sono quindi molteplici all'interno di un sistema, ed è il sistema operativo, passando il controllo ai singoli processi su di esso in esecuzione, a caricare la relativa LDT.

• Questa struttura possiede un istruzione dedicata, la SLDT, usabile a qualunque livello di privilegio, per verificare in quale posizione di memoria essa è mantenuta. Come nel caso della GDT il sistema operativo si aspetta di conoscere l'esatta posizione di una qualunque delle LDT che esso gestisce, e in un ambiente virtualizzato è evidente che queste locazioni possono non coincidere con la reale posizione che il VMM ha scelto per loro.

xen

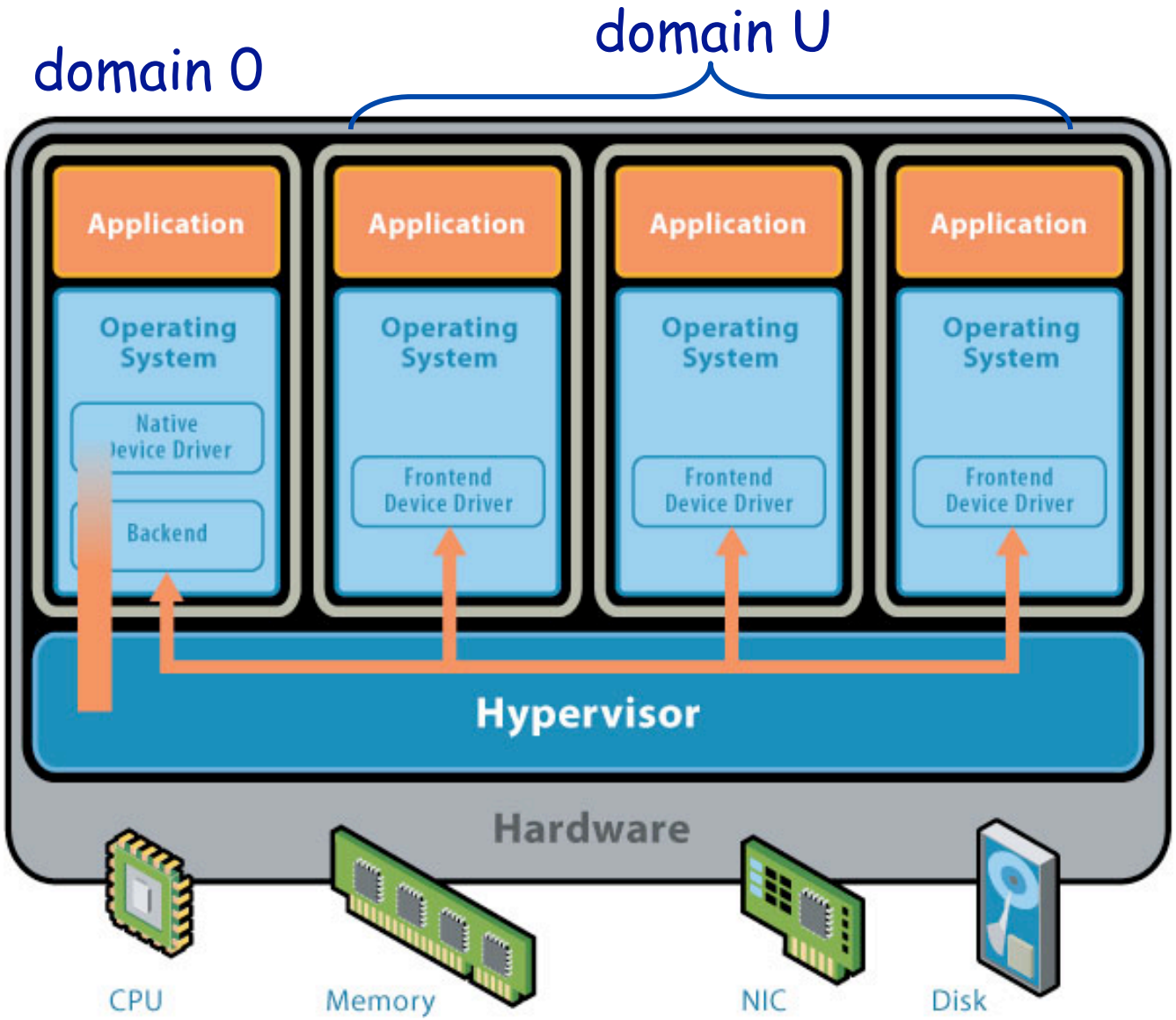
- VMM **open source** che opera secondo i principi della ***paravirtualizzazione*** (Università di Cambridge, 2003).

- 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….) offrono pacchetti precompilati per installare XEN.

Architettura di xen



Organizzazione

- il vmm (hypervisor) si occupa della virtualizzazione della CPU, della memoria e dei dispositivi per ogni macchina virtuale $domain_i$
- 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 privilegiata, conosciuta come *domain 0*.
- Questo dominio utilizza l'applicazione software che gestisce il controllo di tutta la piattaforma.
- Il software di controllo e` eseguito nel domain 0 (dom0), separato dallo stesso hypervisor : separazione dei meccanismi dalle politiche, all'interno del sistema.

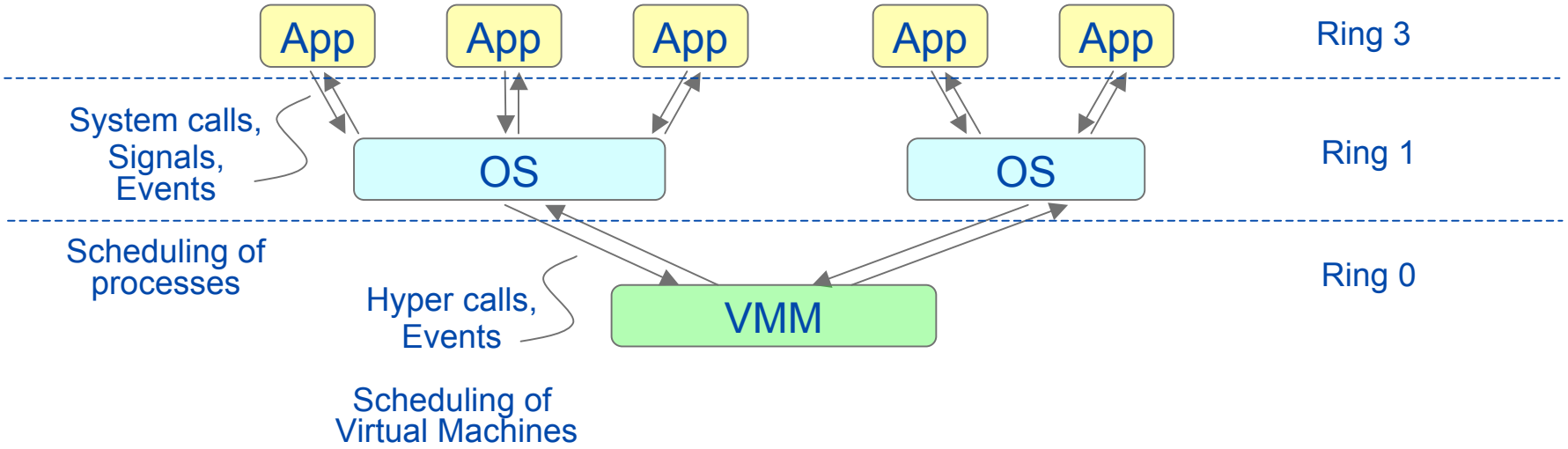
Xen: realizzazione

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

Xen – caratteristiche

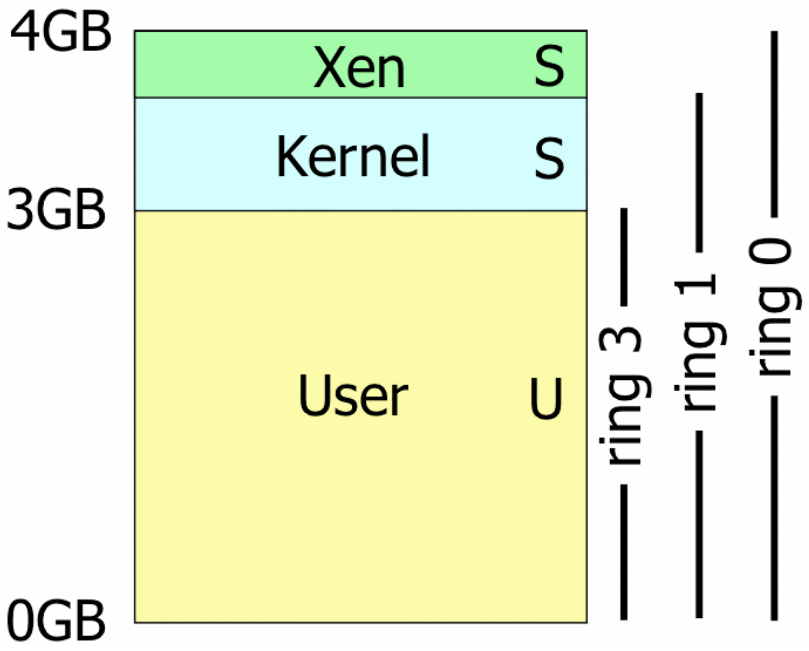
- Paravirtualizzazione:
  - Le macchine virtuali eseguono direttamente le istruzioni non privilegiate
  - L'esecuzione di istruzioni privilegiate viene delegata al VMM tramite chiamate al VMM (hypercalls).

- Protezione (x86):
  - I sistemi operativi guest OS sono collocati nel ring 1
  - VMM collocato nel ring 0



Protezione

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



Xen: paginazione

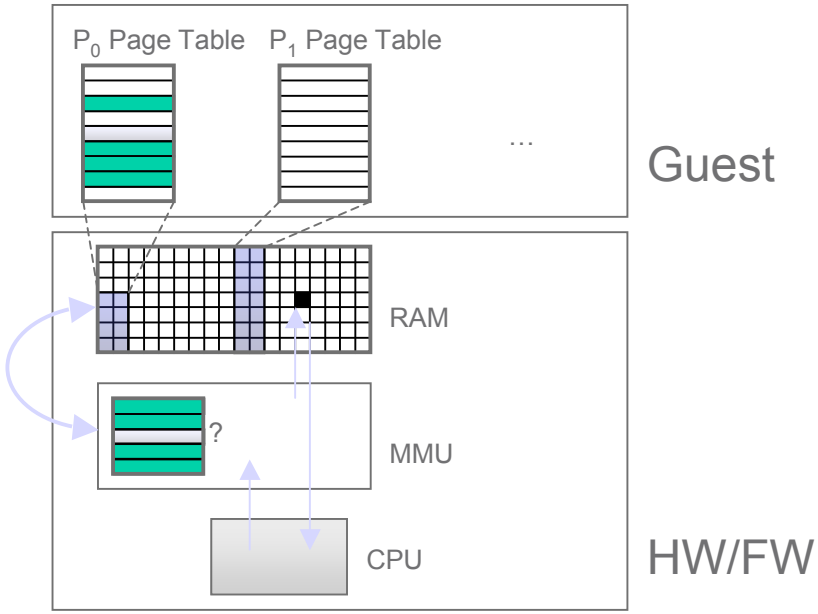
- **Gestione della Memoria:**
  - i SO guest accedono alla memoria virtuale, mediante i tradizionali meccanismi di paginazione
  - X86: page faults gestiti direttamente a livello HW (TLB)

- **Soluzione adottata:** le tabelle delle pagine sono accessibili in modalità read-only anche ai 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:
  - In questo modo non è necessario effettuare un TLB "flush" per ogni hypercall -> maggiore efficienza

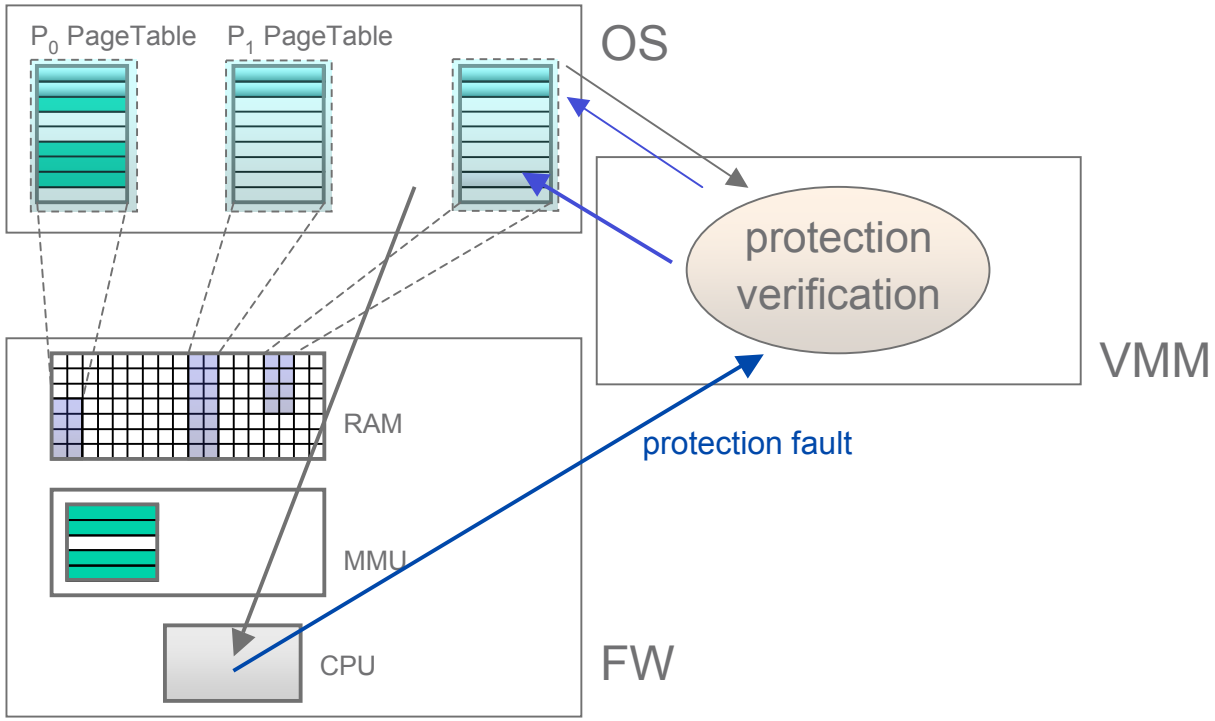
Xen: gestione della memoria

- I guest OS si occupano della paginazione, delegando al VMM la scrittura delle PTE.
- Le tabella della pagine devono essere create e verificate dal VMM su richiesta dei guest:
  - Una volta create, 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 diverse macchine virtuali pagine di memoria meno utilizzate.

• Soluzione: su ogni macchina virtuale è in esecuzione un processo (balloon process) che comunica con il VMM.

•Su richiesta del VMM, il balloon si gonfia, richiedendo al so guest altre pagine.

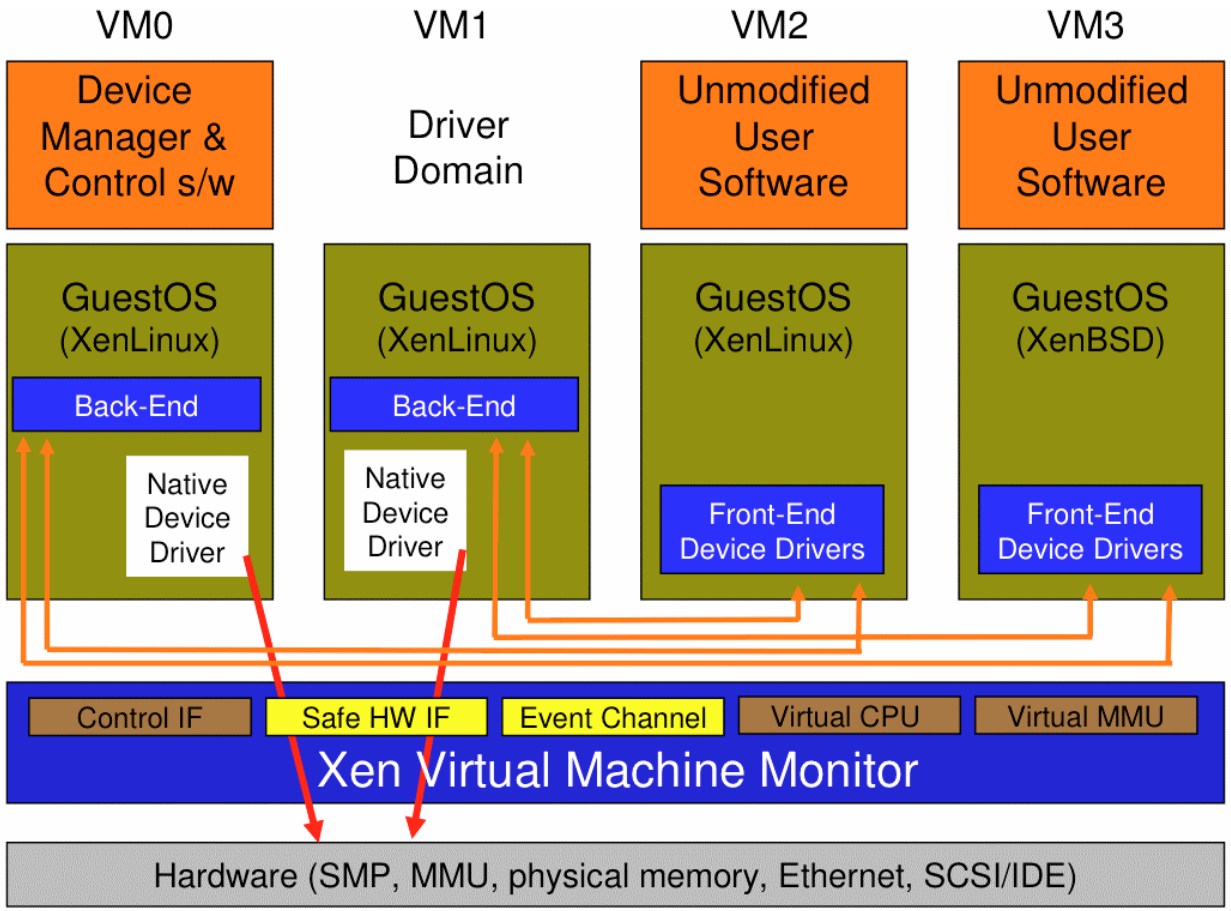
• La richiesta del balloon process provoca da parte del S.O. guest l'allocazione di nuove pagine al balloon process che le cede quindi al VMM.

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 sceduling 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.

Xen -Virtualizzazione dell'I/O

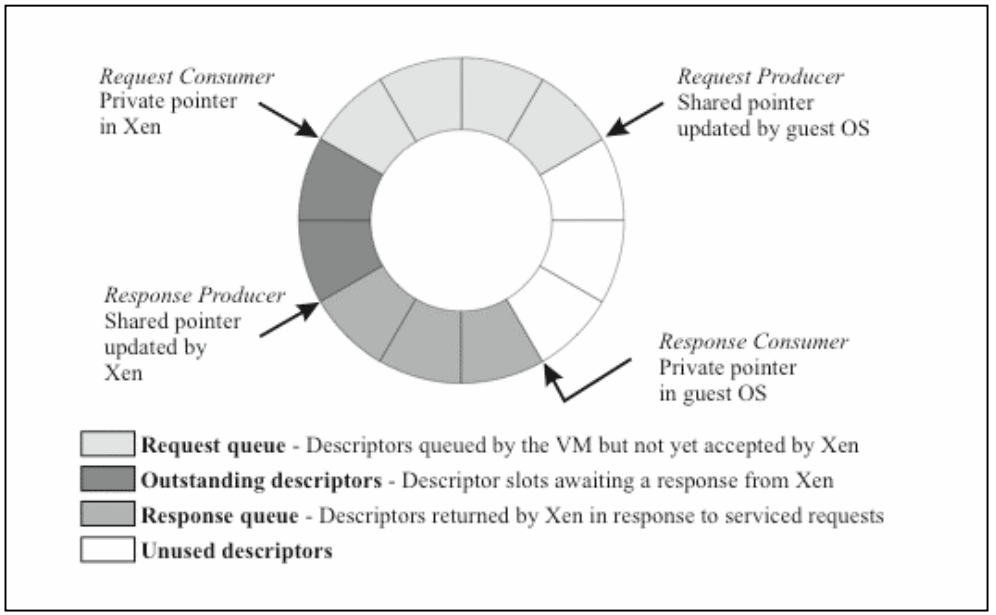


Xen – Virtualizzazione dell'I/O

- 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.

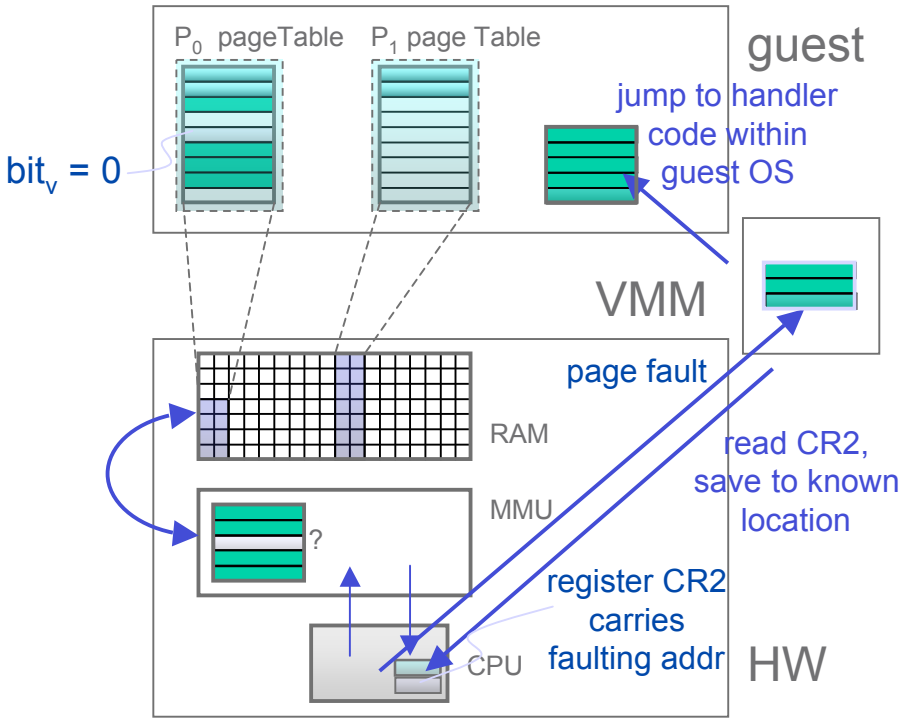


Xen – gestione interruzioni e eccezioni

- 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.



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.

- *G. J. Popek and R.P. Goldberg*
  Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM 17 (7): 412 –421, 1974.*