Cours de Virtualisation

Jérémy BRIFFAUT

INSA-CVL/LIFO/SDS

11 novembre 2019

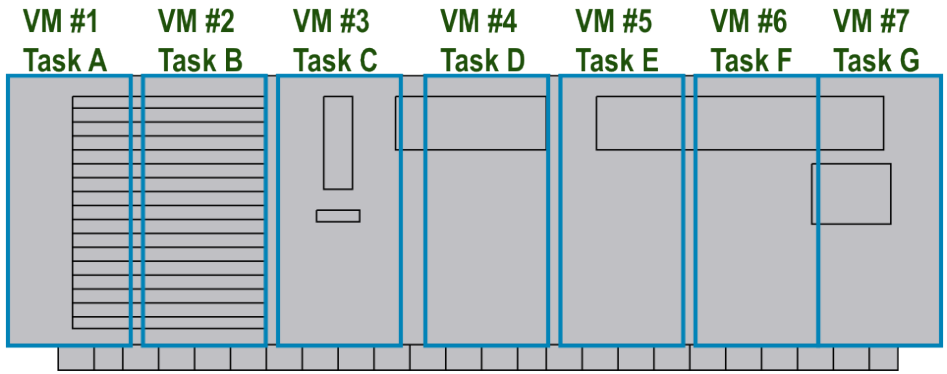
Table des matières

Sommaire

Sommaire

Introduction

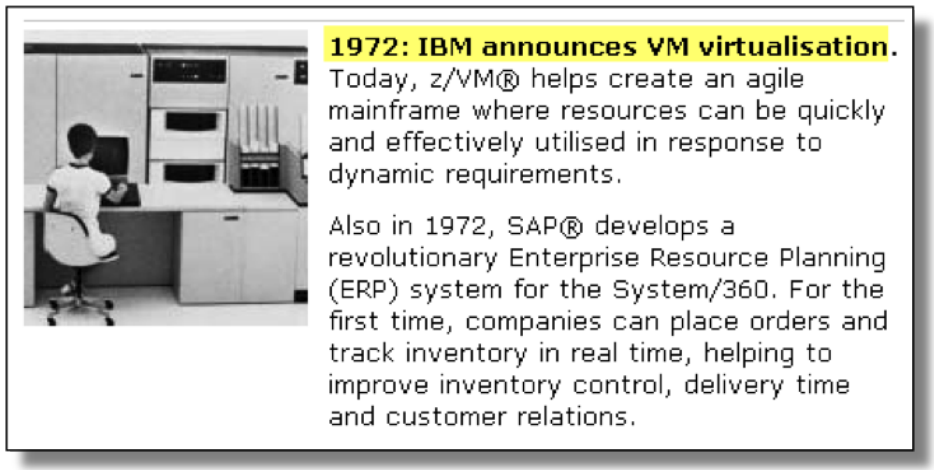
- La virtualisation n'est pas un nouveau concept
- Au début : Partage de mainframe
- Orienté partage de temps et de ressources
- Isolation des utilisateurs
- IBM System/360 (1965) et CTSS (1962)



Logical VMs on a mainframe

Virtualisation Matériel

- Toujours pas un nouveau concept
- DEC PDP-8 : 1965 (buggé)
- DEC PDP-10 et IBM VM/370 : 1970



- `http://www-07.ibm.com/systems/my/z/about/timeline/1970/`

Sommaire

Concept de base

- Utilisation d'instructions permettant de changer de contextes
- Sauvegarde de l'état du processeur
- Exécution d'un autre programme
- Restauration de l'état du processeur
- Toujours le même concept aujourd'hui

Concept de base

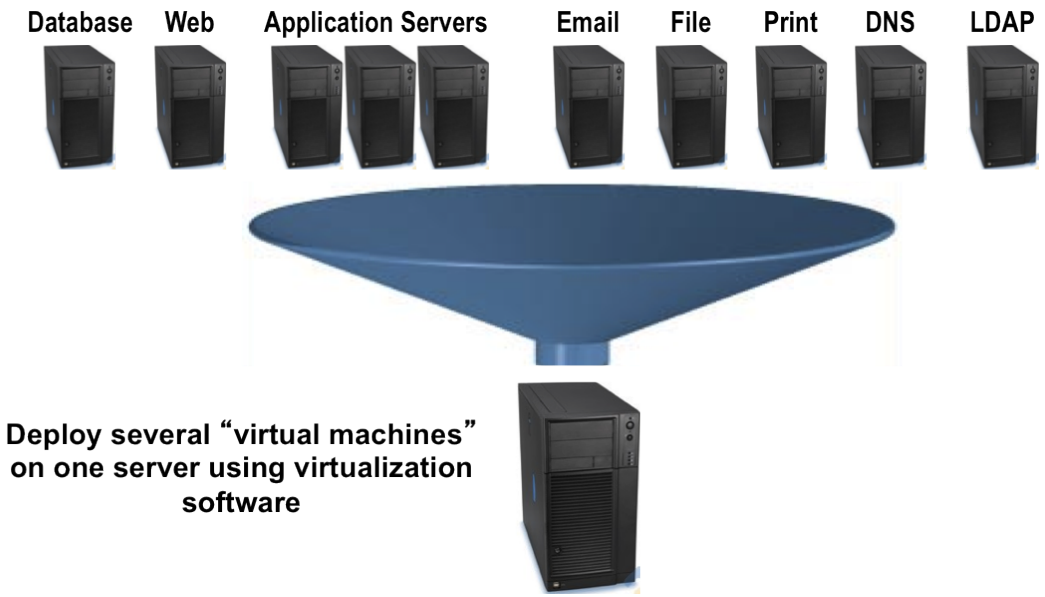
- Chaque programme a *son propre système virtuelle complet*
- Même si plusieurs programmes tournent en même temps, ils sont isolés
- Pas de parallélisme réel pour autant, une instruction à un moment
- Suite à ces travaux, oubli presque total de la virtualisation

Renaissance de la virtualisation

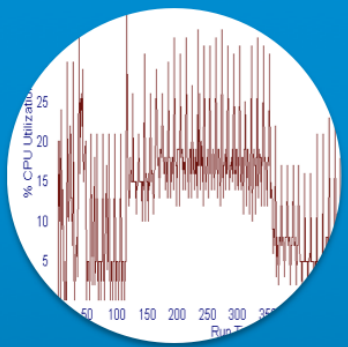
- VMWare relance la virtualisation (fin 1990, début 2000)
- Utilisation de la translation de binaire
- Tout d'abord vers le poste client
- Retour des instructions matériels pour la virtualisation (intel et amd vers 2005)

Objectifs

- Appliquer le principes des Main-Frame au serveurs X86
- Utiliser la virtualisation pour partitionner un serveur INTEL/AMD en un ensemble d'OS et d'applications



Objectifs



Hardware Resources Underutilized

- CPU utilizations ~ 10% - 25%
- One server – One Application
- Multi-core even more under-utilized

Data Centers are running out of space

- Last 10+ years of major server sprawl
- Exponential data growth
- Server consolidation projects just a start

Rising Energy Costs

- As much as 50% of the IT budget
- In the realm of the CFO and Facilities Mgr. now!

Administration Costs are Increasing

- Number of operators going up
- Number of Management Applications going up

Sommaire

Qu'est ce que la virtualisation ?

- Couche de virtualisation : Hyperviseur (Hypervisor) ou Virtual Machine Monitor (VMM)
- Conteneur de système : Machine Virtuel ou Virtual Machine (VM)
- Définition :

Virtualisation

Le processus de virtualisation est l'acte d'abstraire les couches inférieur (ex. le matériel). La virtualisation insère une couche entre les couches existantes (matériels, système d'exploitation, applicatifs) pour résoudre le problème de support de portabilité, de changements des interfaces, d'isoler les ressources/utilisateurs ou de consolider les serveurs.

Plus précisément

- Virtualisation de la couche matériel
- Virtualisation du système
- Virtualisation de l'application

Hypervisor

Hypervisor

Un hyperviseur, aussi appelé un manager de machines virtuelles (VMM), est un programme qui permet à plusieurss OS de partager le matériel d'un seul hôte. Chaque OS croit posséder le processeur de l'hôte, la mémoire et les autres ressources pour lui seul. Cependant, l'hyperviseur contrôle le processeur de l'hôte et les ressources, il alloue ce qui est nécessaire à chaque OS pour fonctionner et assure qu'un système invité (aussi appelé machine virtuelle) ne peut pas interférer avec les autres VM.

Comment mettre en place la virtualisation ?

- G. Popek et R.P. Goldberg ont défini en 1973 les besoins de la virtualisation
- Leur définition demande trois conditions qui sont suffisantes pour qu'une architecture supporte la virtualisation
- Ces conditions sont toujours valables de nos jours.

Les trois conditions de la virtualisation

- **Efficacité** : Les VM ne doivent pas souffrir de dégradation de performances qui sont observables. C'est à dire qu'un grand sous ensemble des instructions (processeurs) doivent se dérouler directement sur le vrai processeur sans avoir besoin de demander la permission à la couche de virtualisation (hyperviseur).
- **Contrôle des ressources** : L'hyperviseur doit alloué les ressources qui sont demandées par les système virtualiser. Il doit également nettoyé ces ressources quand elles ne sont plus utilisées. En pratique, cela veut dire qu'une VM ne doit pas pouvoir interférer avec les ressources des autres VM. L'hyperviseur doit donc avoir un contrôle total des ressources virtualisées.
- **Équivalence** : L'hyperviseur doit fournir une couche virtuelle d'abstraction matériel qui permet aux programmes de

Sommaire

Avantages de la virtualisation : Réduction des couts

- La virtualisation permet de regrouper plusieurs application/utilisateurs/systèmes d'exploitation sur un nombre réduit d'ordinateurs physique.
- On appelle ça la consolidation.
- But de la consolidation : Réduction des coûts en réduisant l'énergie, l'infrastructure et le personnel technique.
- La consolidation permet aussi de regrouper plusieurs services.
- Passage d'un service par serveur à plusieurs services par serveur.

Avantages de la virtualisation : Write Once, Run Everywhere

- Permet de ne pas se soucier des dépendances matériels et/ou logiciels.
- Avec un bon outil de virtualisation, tout ce qui tourne sur du matériel devrait fonctionner.
- Permet de faire tourner plusieurs systèmes d'exploitation et/ou architecture matériel.
- Économie en réduisant le temps de développement.

Avantages de la virtualisation : Efficacité

- Utilisation efficace des ressources.
- Dépend du type de virtualisation.
- Optimal quand le maximum d'utilisation de ressources matériels sont accessible (voir les théorèmes de la virtualisation).

Avantages de la virtualisation : Isolation

- Introduction de la virtualisation comme aide pour la sécurité par le MIT dans le Projet MAC en 1973.
- L'hyperviseur peut être vu comme un moniteur de référence.
- Le but de l'hyperviseur est d'isoler les VM entre elles.
- Isolation beaucoup plus forte que l'isolation par processus.
- Cette isolation est essentielle pour répartir la charge de plusieurs VM sur un ensemble d'hyperviseur.

Avantages de la virtualisation : Fiabilité

- Si un des services i.e. VM plante, les autres VM ne plantent pas.
- C'est une conséquence de l'isolation des ressources.
- Peut ne pas toujours être vrai.
- Très lier au type de virtualisation.

Sommaire

Désavantages de la virtualisation : Sur-utilisation

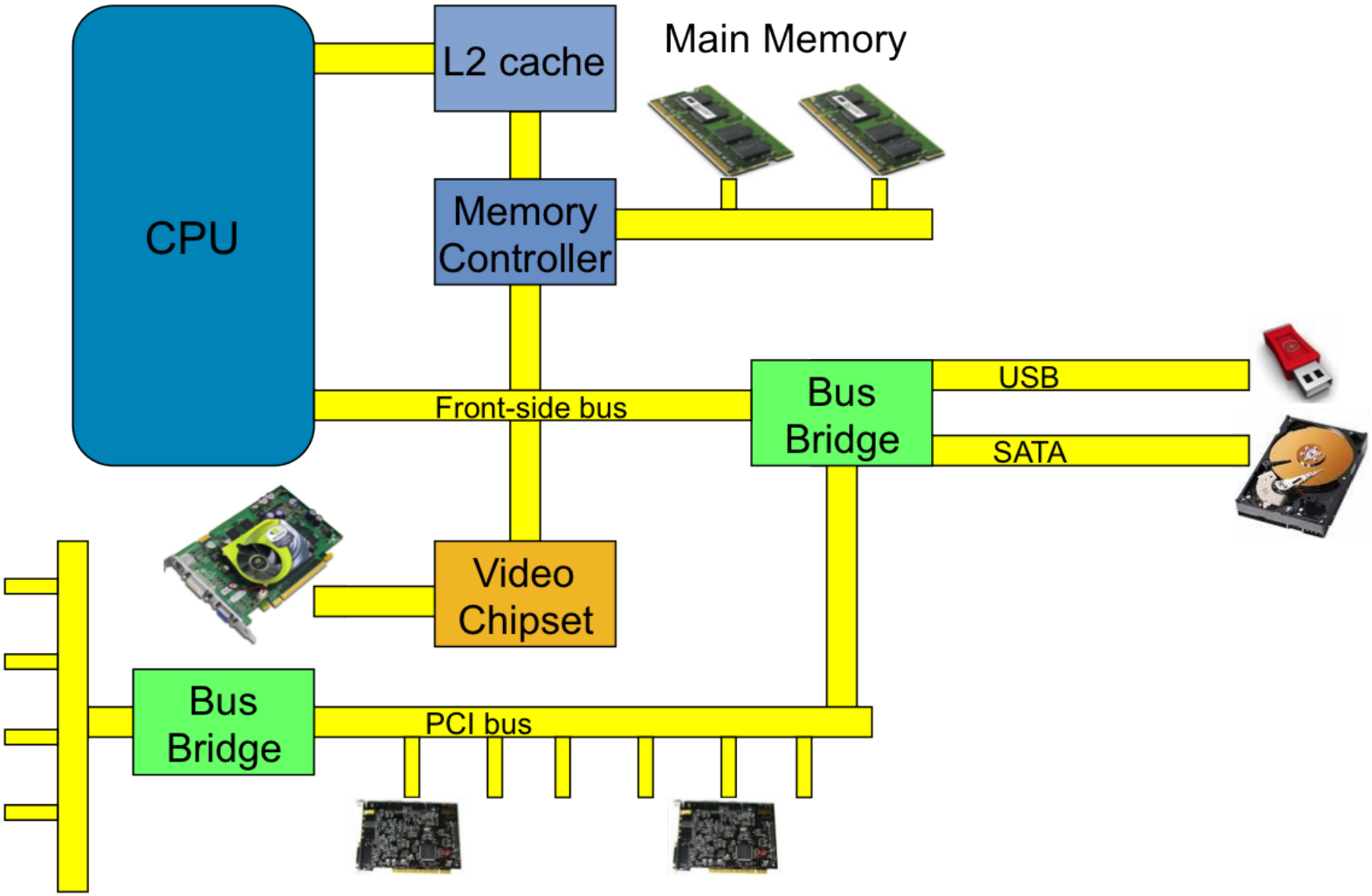
- La sur-utilisation est due à la couche de virtualisation qui est une couche supplémentaire.
- Résultat : il faut un temps de calcul supplémentaire pour faire fonctionner cette couche.
- La plupart des méthodes de virtualisation ne respecte pas les règles.
- Résultat : Sur-utilisation due à la translation d'instructions dans les VM vers les réelles sur le matériel.

Désavantages de la virtualisation : Pas si efficace

- Dans certains cas, il faut réécrire/modifier plus ou moins partiellement les applications/OS/services.
- Résultat : Pas aussi portable.
- Résultat : Peut remettre en cause la facilité de mise à jour (deux versions : une normale et une virtualisée).

Sommaire

Computer System Organization



Qu'est-ce qu'un système d'exploitation ?

- Un système d'exploitation :
  - Couche logicielle intercalée entre l'ordinateur et ses utilisateurs, lancée au démarrage ;
  - Intermédiaire entre les logiciels applicatifs et le matériel ;
  - Facilite l'exploitation des périphériques matériels dont il coordonne et optimise l'utilisation ;
  - Met ‡ disposition des logiciels applicatifs une interface de programmation standardisée d'utilisation des matériels ;

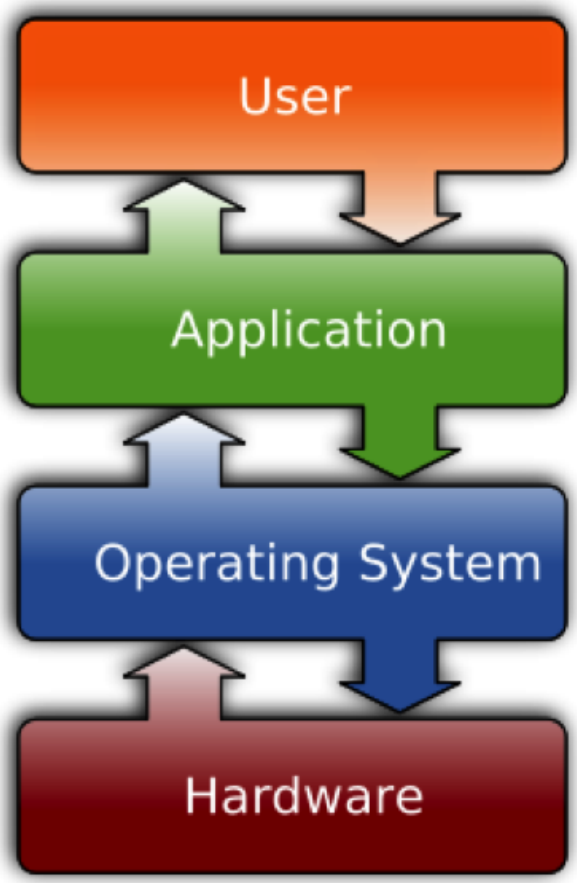
Qu'est-ce qu'un système d'exploitation ?

- Ordinateur = ensemble de ressources physiques
  - Processeur/Unité Centrale (CPU)
  - Mémoire principale
  - Mémoires secondaires
  - Périphériques d'E/S
  - Périphériques internes (horloge,...)
- Son utilisation génère des ressources logiques
  - Processus
  - Fichiers
  - Bibliothèques *Système* partagées
  - Sessions utilisateurs

Qu'est-ce qu'un système d'exploitation ?

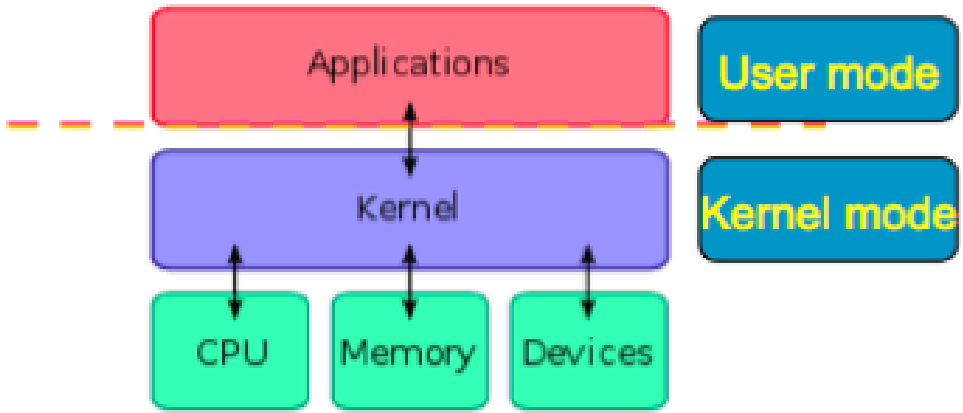
- Un système d'exploitation :
  - Alloue les ressources aux utilisateurs
  - Contrôle leur bonne utilisation
  - Problèmes : efficacité, fiabilité, sécurité, équité, etc.

Operating System



OS Architecture : the Kernel

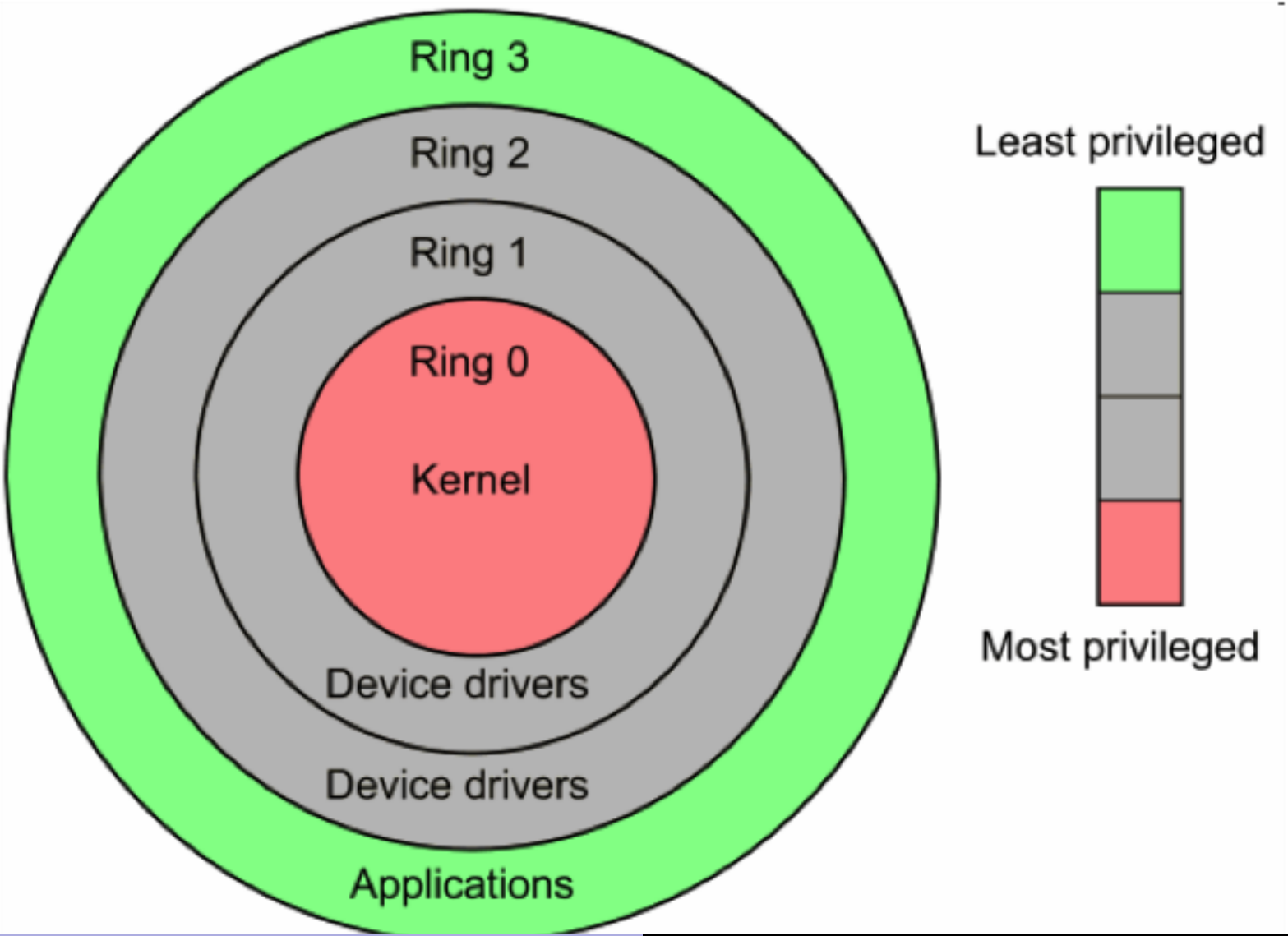
- Kernel = procure une couche d'abstraction pour les ressources nécessaires à une application
- Resources = matériel tel que processeur, mémoire, I/O
- Le Kernel démarre immédiatement après le bootloader. Il ne s'exécute pas réellement directement, ses services sont invoqués via :



Les Anneaux

- Un anneau de protection (ou ring) correspond à un niveau de privilèges sur le processeur.
- Il existe de tel structure en anneau sur la plupart des processeurs modernes.
- Introduit par Multics (8 anneaux).
- Utilisation assez faible (2 anneaux de réellement utilisées).
- Ring0 : Noyau.
- Ring3 : Programmes utilisateurs.
- Problème de passage d'un ring à l'autre : Fort coût.

Les Anneaux


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged

Les Anneaux

Traditional systems

Operating system runs in privileged mode in Ring 0 and owns the hardware

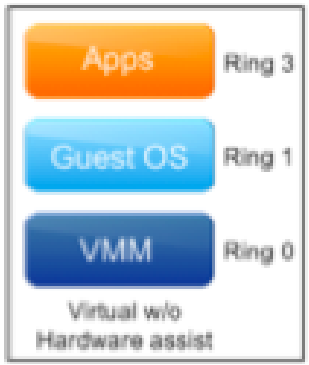
Applications run in Ring 3 with less privileges

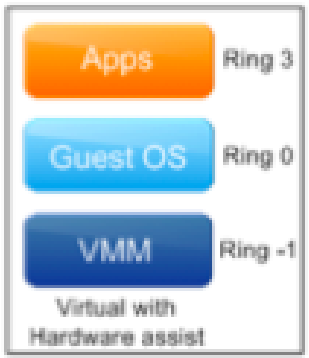
Virtualized systems

VMM runs in privileged mode in Ring 0

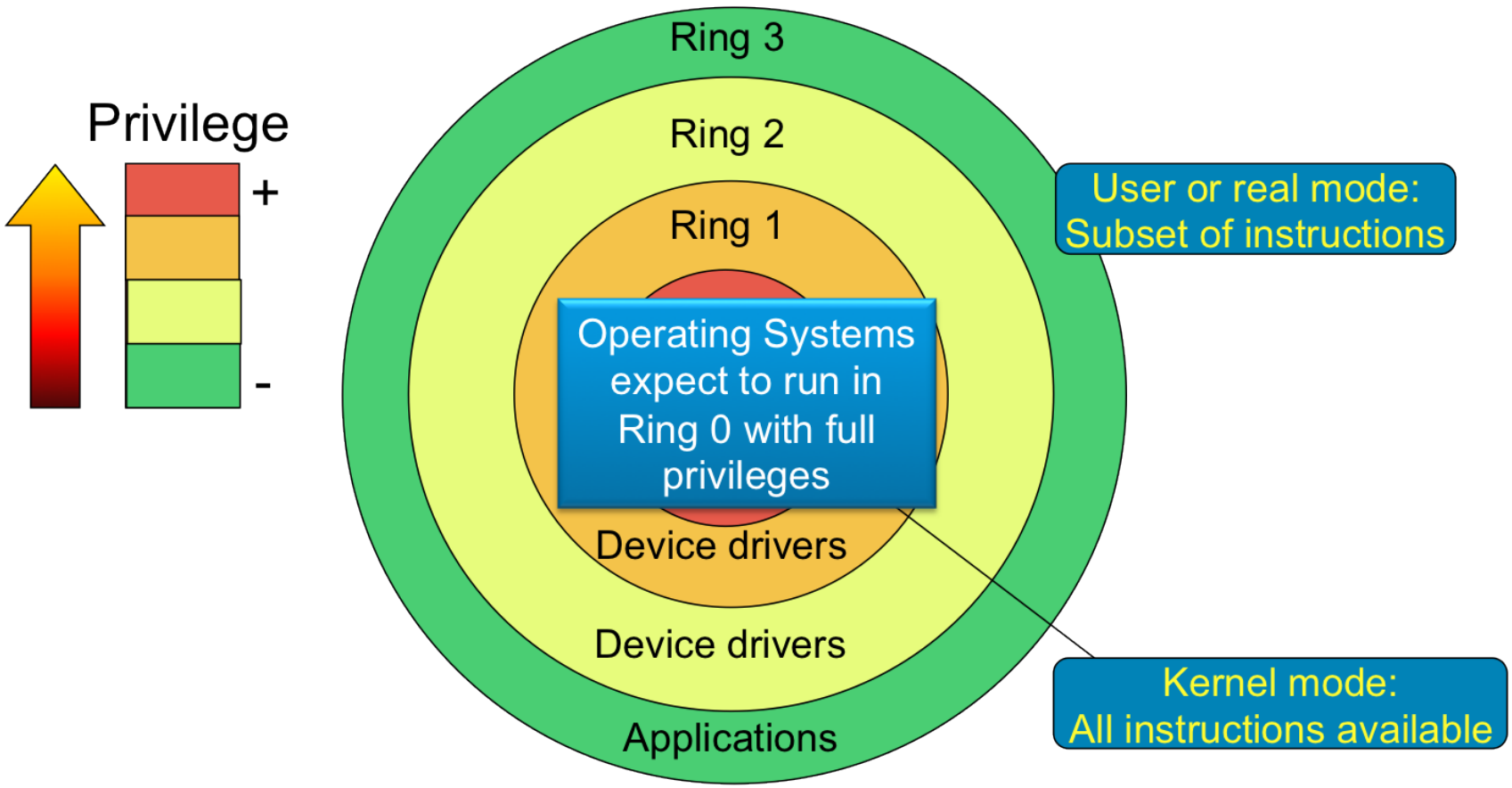
Guest OS inside VMs are fooled into thinking they are running in Ring 0, privileged instructions are trapped and emulated by the VMM

Newer CPUs (AMD-V/Intel-VT) use a new privilege level called Ring -1 for the VMM to reside allowing for better performance as the VMM no longer needs to fool the Guest OS that it is running in Ring 0.





x86 Protection/Privilege Rings



Sensitive and Privileged Instructions

- Un processeur fournit un ensemble d'instructions (instruction set)
- Sur un processeur x86, certaine instructions sont
  - Privileged
  - Sensitive
  - Normal
- Les instructions privilégiées sont piégées (trap) lorsqu'elles sont exécutées en espace utilisateur (user mode)
- Les instructions sensibles tentent de modifier la configuration des ressources du système
- Naturellement, le noyau est autorisé à exécuter ces instructions sans qu'elles soient piégées

You Said Trap ?

- Normalement un processus interagit avec le matériel/services noyau à travers des interruptions logicielles ou des appels système (instructions virtuelles ; Unix en fourni plus de 150)
- Un processus qui exécute une instruction privilégiées cause une trap qui provoque le passage du système en mode kernel
- Exemples : *invalid memory access*, *file not found*, *file system full*, etc.
- Le Kernel exécute ensuite une action (exécute une ou plusieurs instructions privilégiées)

Virtualization Requirements

- Rappel : l'article de Popek et Goldberg en 1974 : "*For any computer a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions*"
- C'est-à-dire : *virtual machine monitor needs a way of determining when a guest executes sensitive instructions.*

x86 Virtualization Challenges

- L'ensemble d'instructions de architecture IA-32 (x86) contient 17 instructions sensibles et non-privilégiées qui ne sont pas piégées
  - Instructions sensible sur les registres : *read or change sensitive registers and/or memory locations such as a clock register or interrupt registers* :
    - SGDT, SIDT, SLDT, SMSW, PUSHF, POPF, etc.
- l'architecture x86 se satisfait pas aux conditions de Popek-Golberg !

Virtualizing the x86 Processor : possible !

- Apporter le support de la virtualisation pour un processeur x86 :
  - Les instructions non-sensibles, non-protégées peuvent être exécutées directement
  - Les instructions Sensitive privileged doivent être piégées (trap)
  - Les instructions Sensitive non-privileged doivent être détectées

Sommaire

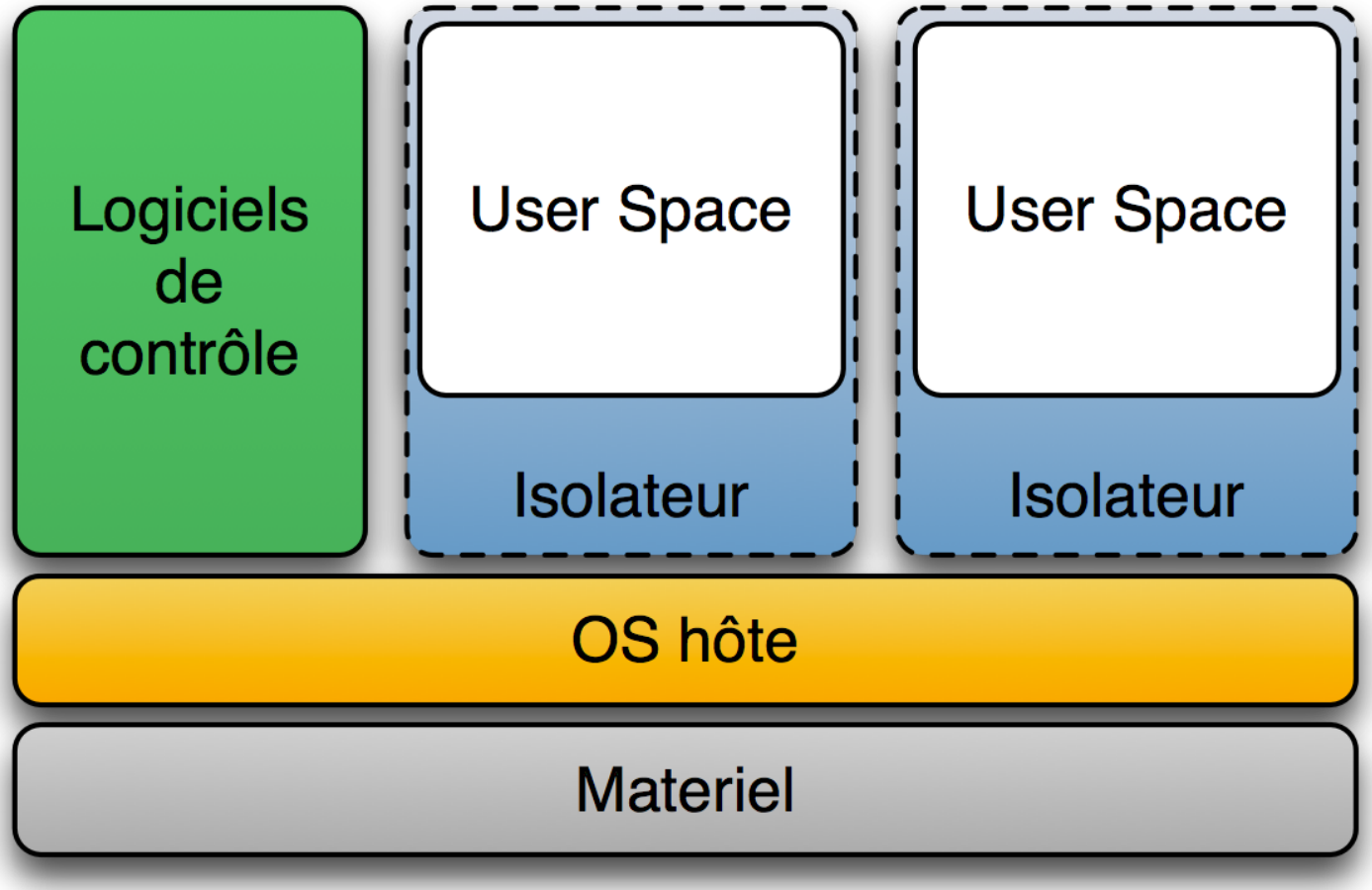
Méthodes de Virtualisation

- Virtualisation complète.
- Paravirtualisation.
- Virtualisation Système (confinement).
- Virtualisation Applicative (sandboxing).
- Hyperviseur bare-metal (type 1)
- Hyperviseur sur système d'exploitation (type 2)

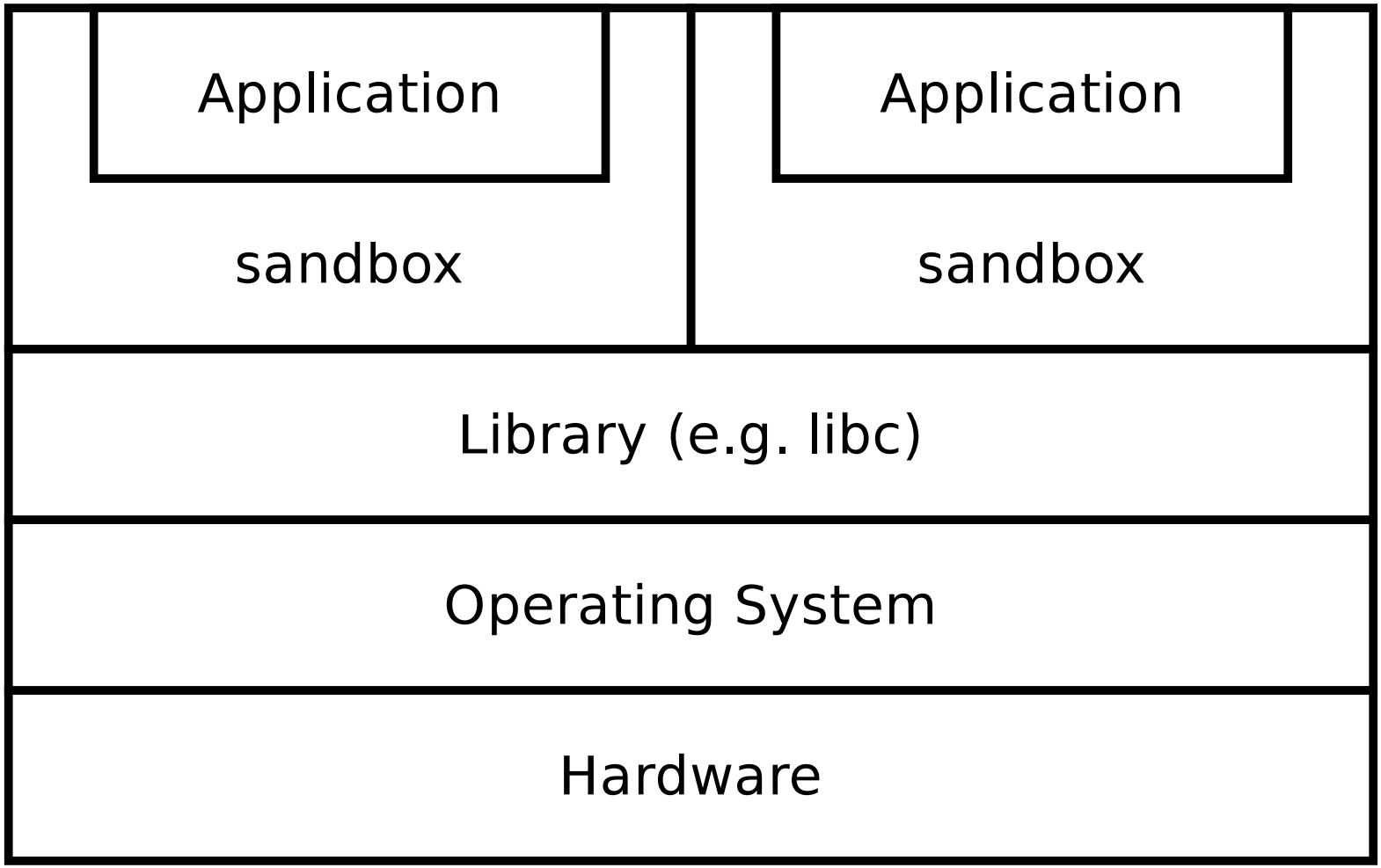
Sommaire

Isolateur

- Logiciel permettant d'isoler l'exécution des applications
- Permet ainsi de faire tourner plusieurs fois la même application même si elle n'était pas conçue pour ça.
- Solution très performante, du fait du peu d'overhead ...
- ... mais les environnements virtualisés ne sont pas complètement isolés.
- On ne peut pas vraiment parler de virtualisation de systèmes d'exploitation
- Exemples :
  - **Linux-VServer** : isolation des processus en espace utilisateur ;
  - **chroot** : isolation changement de racine ;
  - **BSD Jail** : isolation en espace utilisateur ;
  - **OpenVZ** : libre, partitionnement au niveau noyau sous Linux.


Logiciels de contrôle
User Space
Isolateur
User Space
Isolateur
OS hôte
Materiel

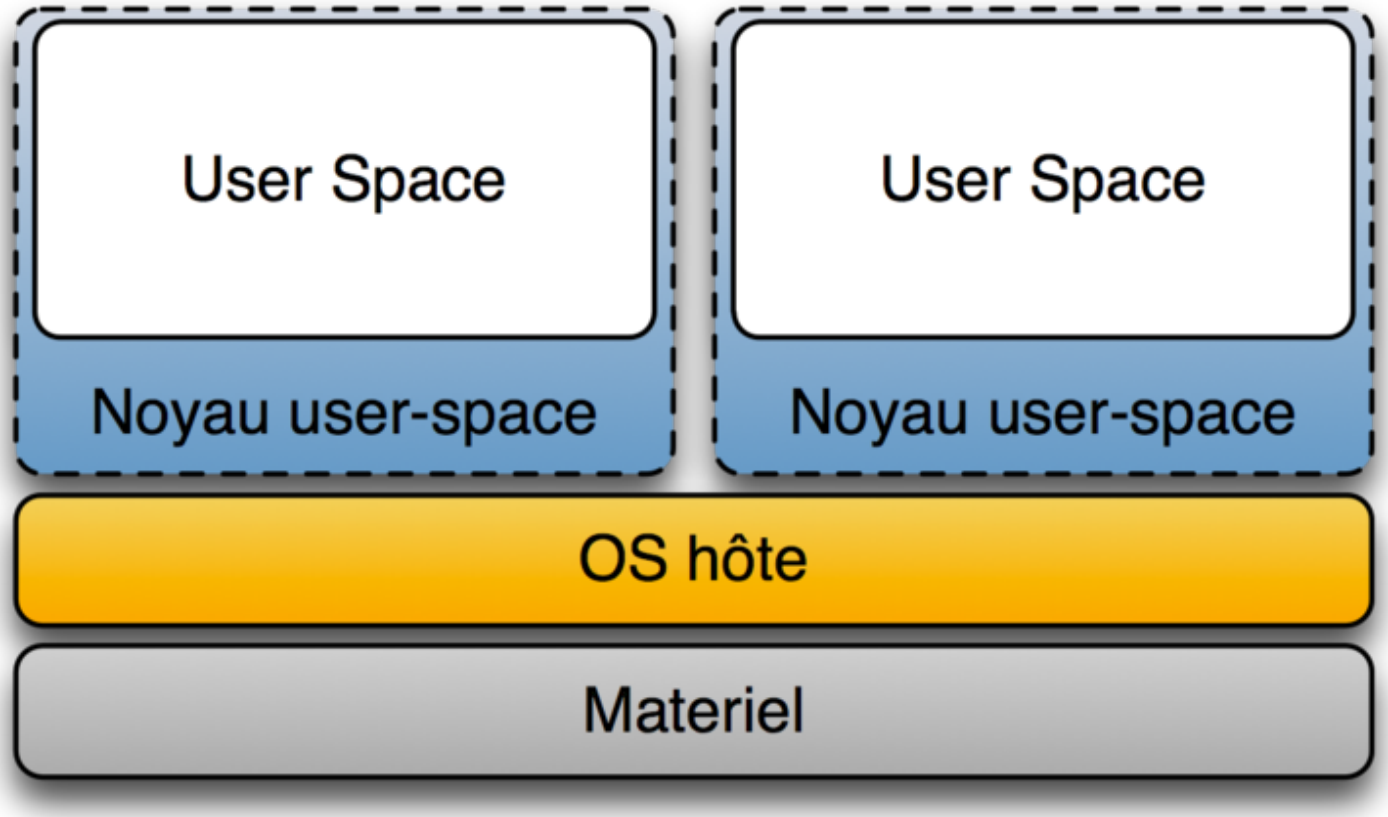
Virtualisation applicative



Sommaire

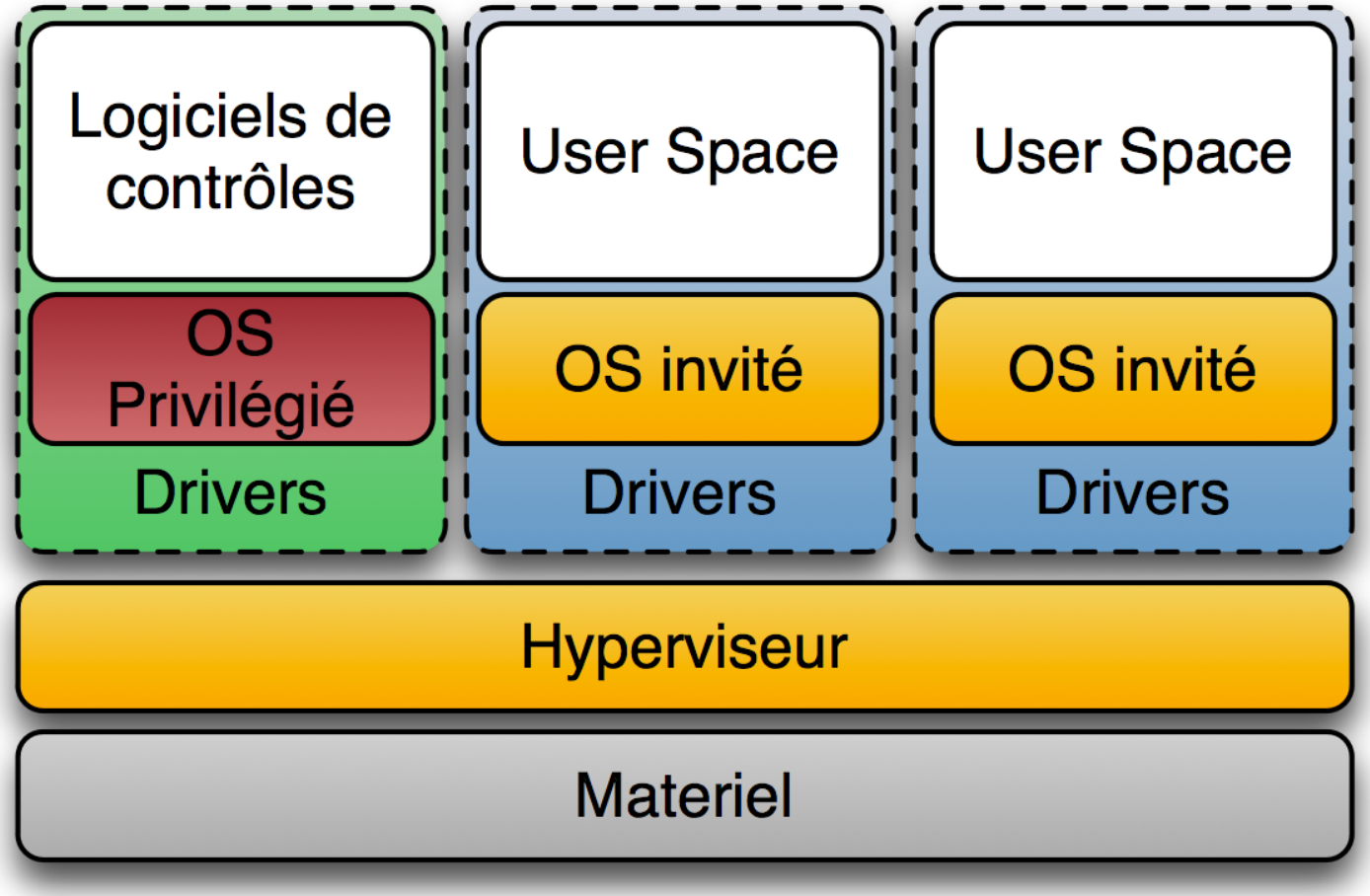
Noyau en espace utilisateur

- Un noyau en espace utilisateur (user-space) tourne comme une application en espace utilisateur de l'OS hôte.
- Le noyau user-space a donc son propre espace utilisateur dans lequel il contrôle ses applications.
- Solution est très peu performante, car deux noyaux sont empilés
- Isolation des environnements n'est pas gérée et l'indépendance par rapport au système hôte est inexistante.
- **Elle sert surtout au développement du noyau.**
- Exemples :
  - **User Mode Linux** : noyau tournant en espace utilisateur
  - **Cooperative Linux ou coLinux** : noyau coopératif avec un hôte Windows
  - **Adeos** : micro noyau RT faisant tourner Linux en kernel-space non-RT
  - **L4Linux** : micro noyau RT faisant tourner Linux en

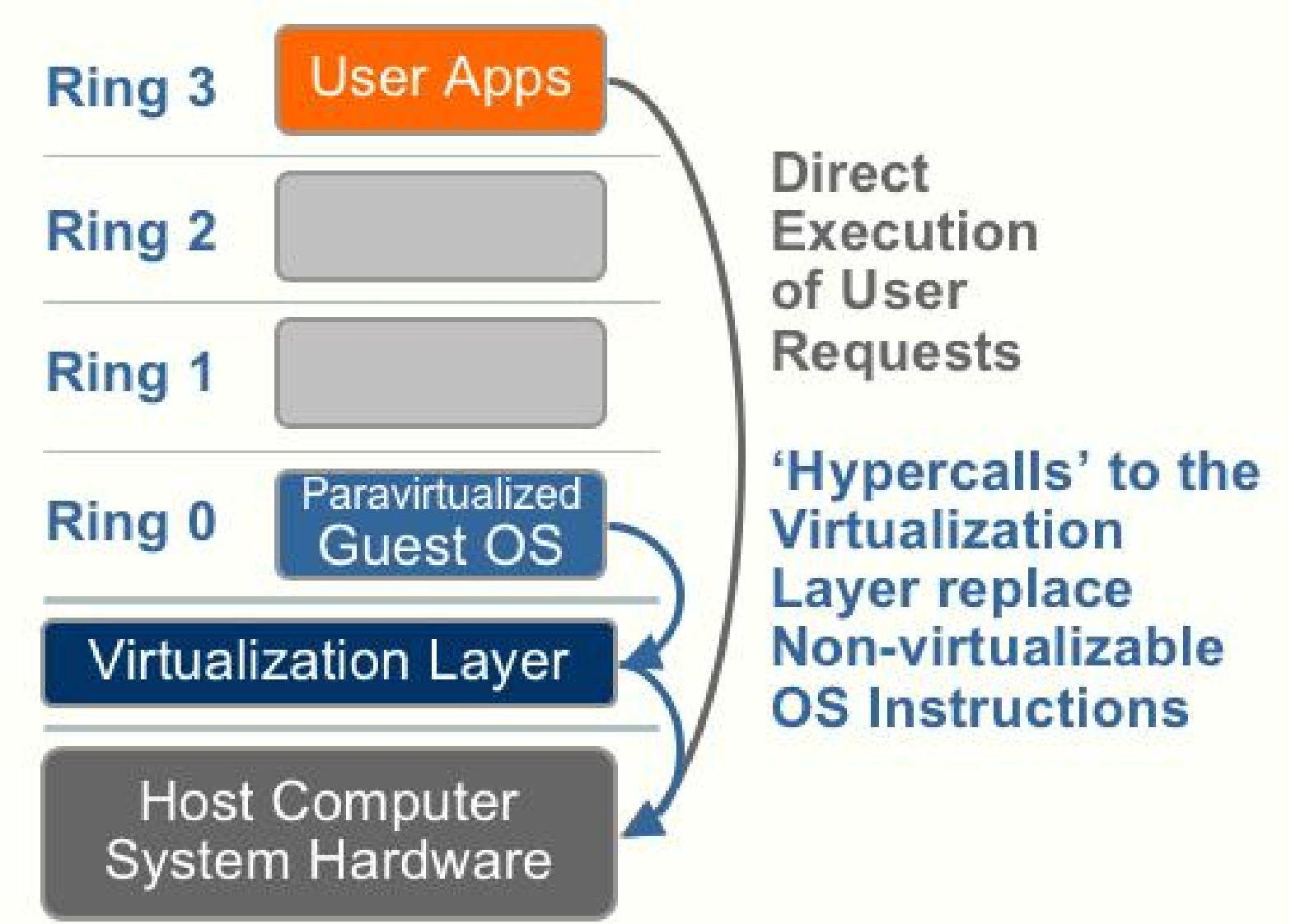

User Space
User Space
Noyau user-space
Noyau user-space
OS hôte
Materiel

Sommaire

- fonctionne comme un noyau système très léger et optimisé pour gérer les accès des noyaux d'OS invités à l'architecture matérielle sous-jacente.
- Si les OS invités fonctionnent en ayant conscience d'être virtualisés et sont optimisés pour ce fait, on parle alors de **para-virtualisation** (méthode indispensable sur Hyper-V de Microsoft et qui augmente les performances sur ESX de VMware par exemple).
- Actuellement l'hyperviseur est la méthode de virtualisation d'infrastructure la plus performante mais (onéreuse ?)
- Exemples :
  - Xen (libre)
  - VMware ESX, VMware ESXi
  - Microsoft Hyper-V Server
  - Parallels Server Bare Metal
  - KVM (libre)
  - Oracle VM (gratuit)


Logiciels de contrôles
OS Privilégié
Drivers
User Space
OS invité
Drivers
User Space
OS invité
Drivers
Hyperviseur
Materiel

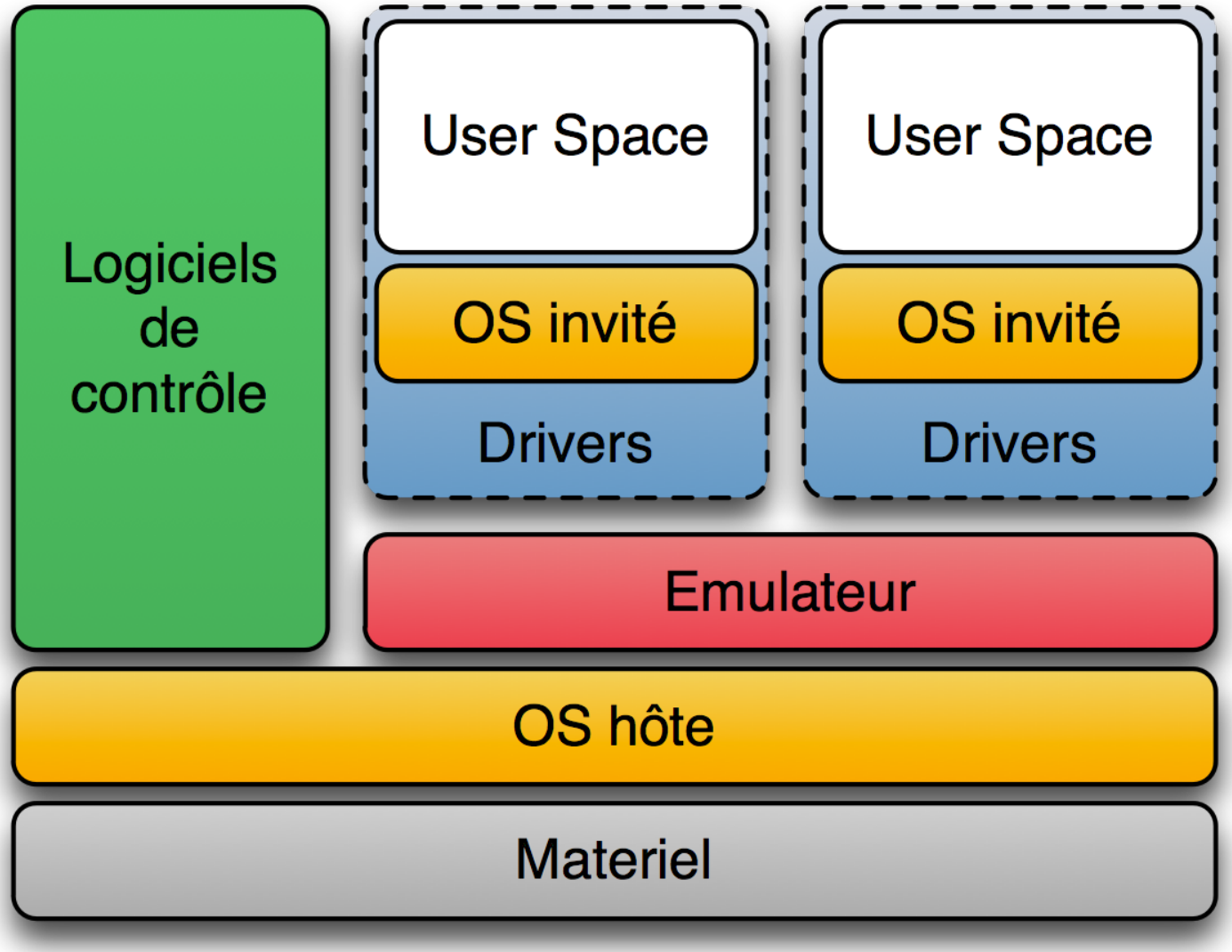
Paravirtualisation



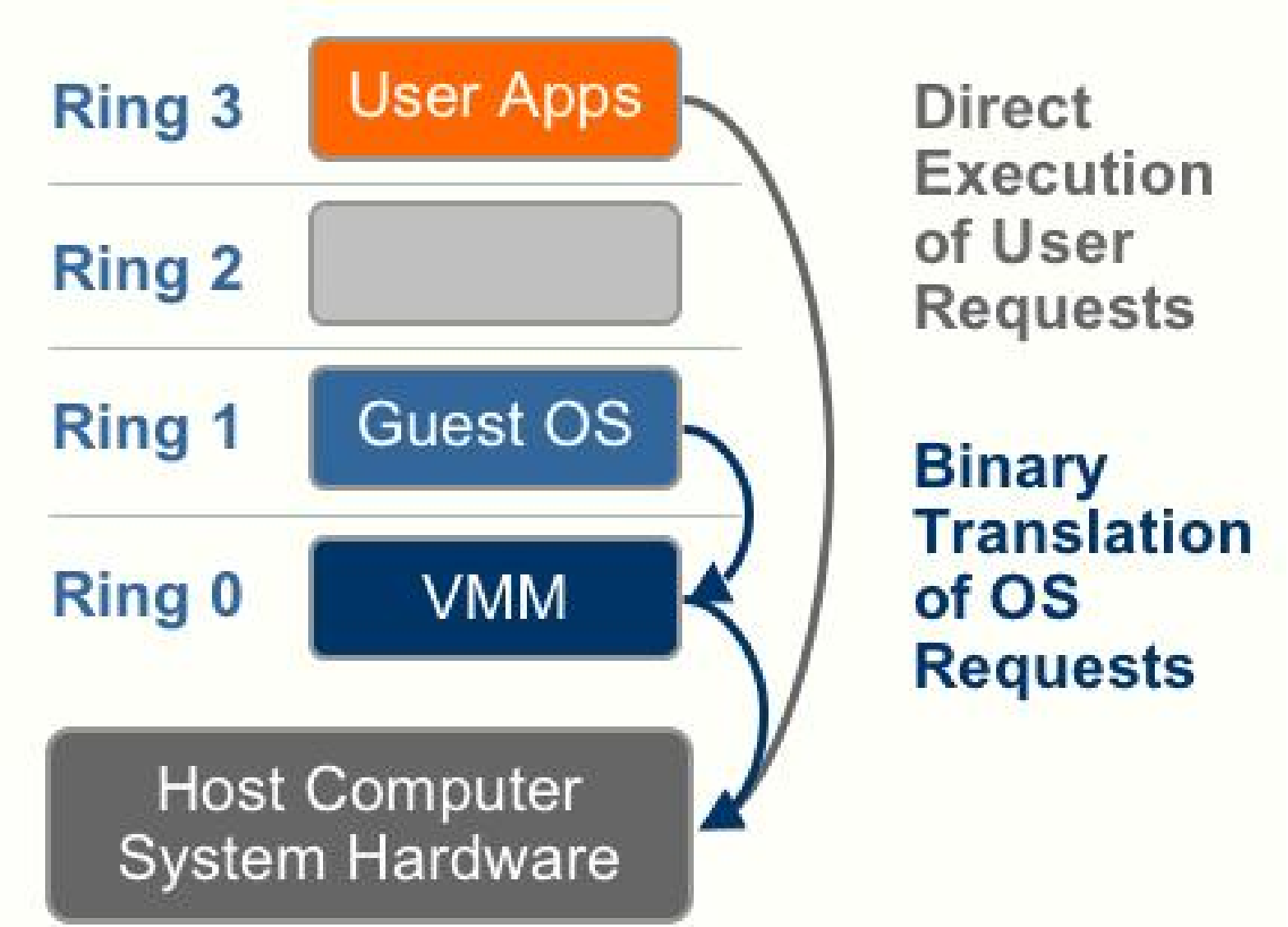
Sommaire

Hyperviseur (Type 2)

- Logiciel (généralement assez lourd) qui tourne sur l'OS hôte.
- Ce logiciel permet de lancer un ou plusieurs OS invités.
- La machine virtualise ou/et émule le matériel pour les OS invités, ces derniers croient dialoguer directement avec ledit matériel.
- Cette solution est très comparable à un émulateur, et parfois même confondue.
- Isole bien les OS invités, mais elle a un coût en performance.
- Exemples :
  - Logiciels Microsoft : Microsoft VirtualPC, Microsoft VirtualServer
  - Logiciels Parallels :Parallels Desktop, Parallels Server
  - Oracle VM VirtualBox (libre)
  - Logiciels VMware : VMware Fusion, VMware Player, VMware Server, VMware Workstation


Logiciels de contrôle
User Space
OS invité
Drivers
User Space
OS invité
Drivers
Emulateur
OS hôte
Materiel

Virtualisation complète



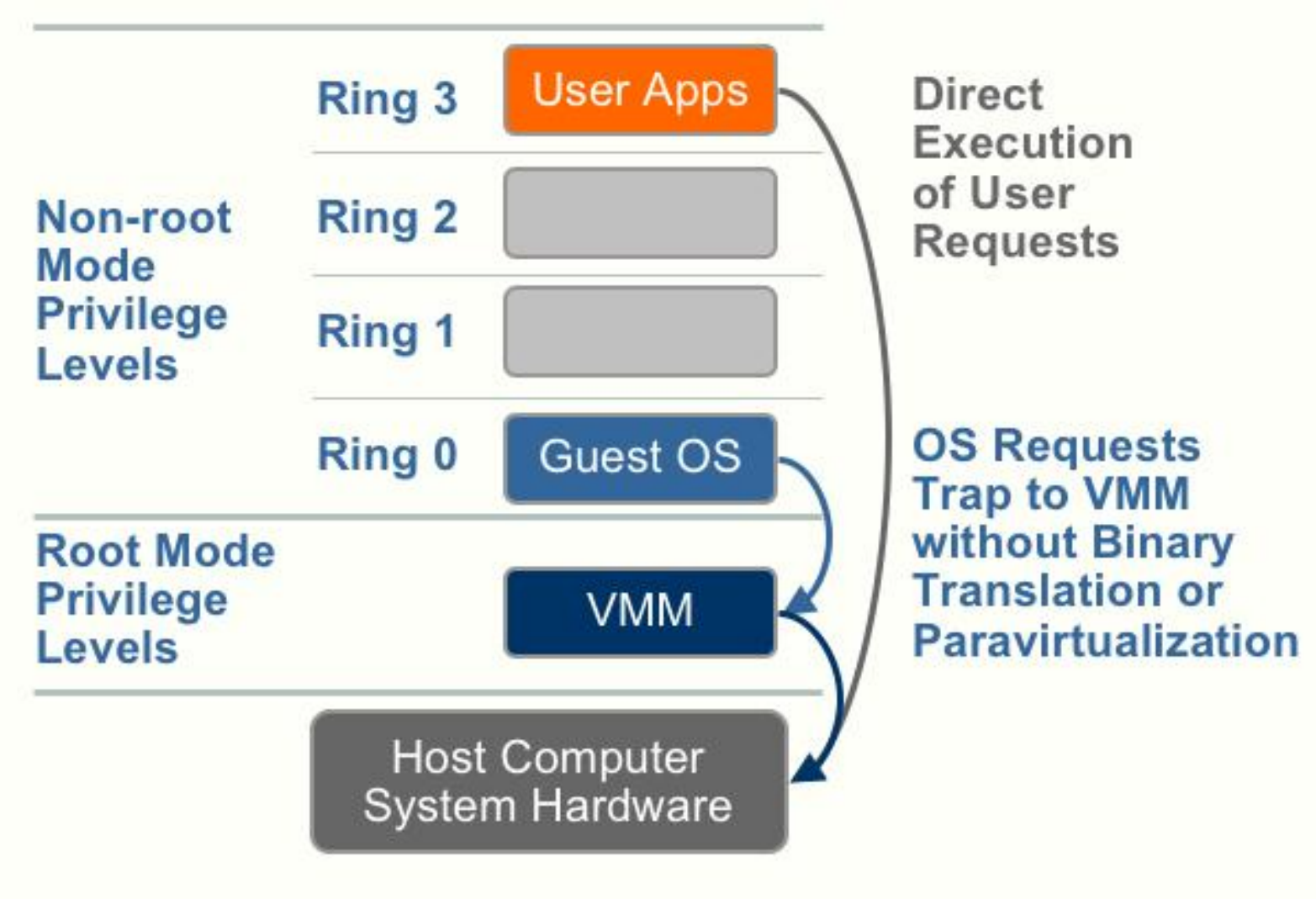
Sommaire

Virtualisation Matériel

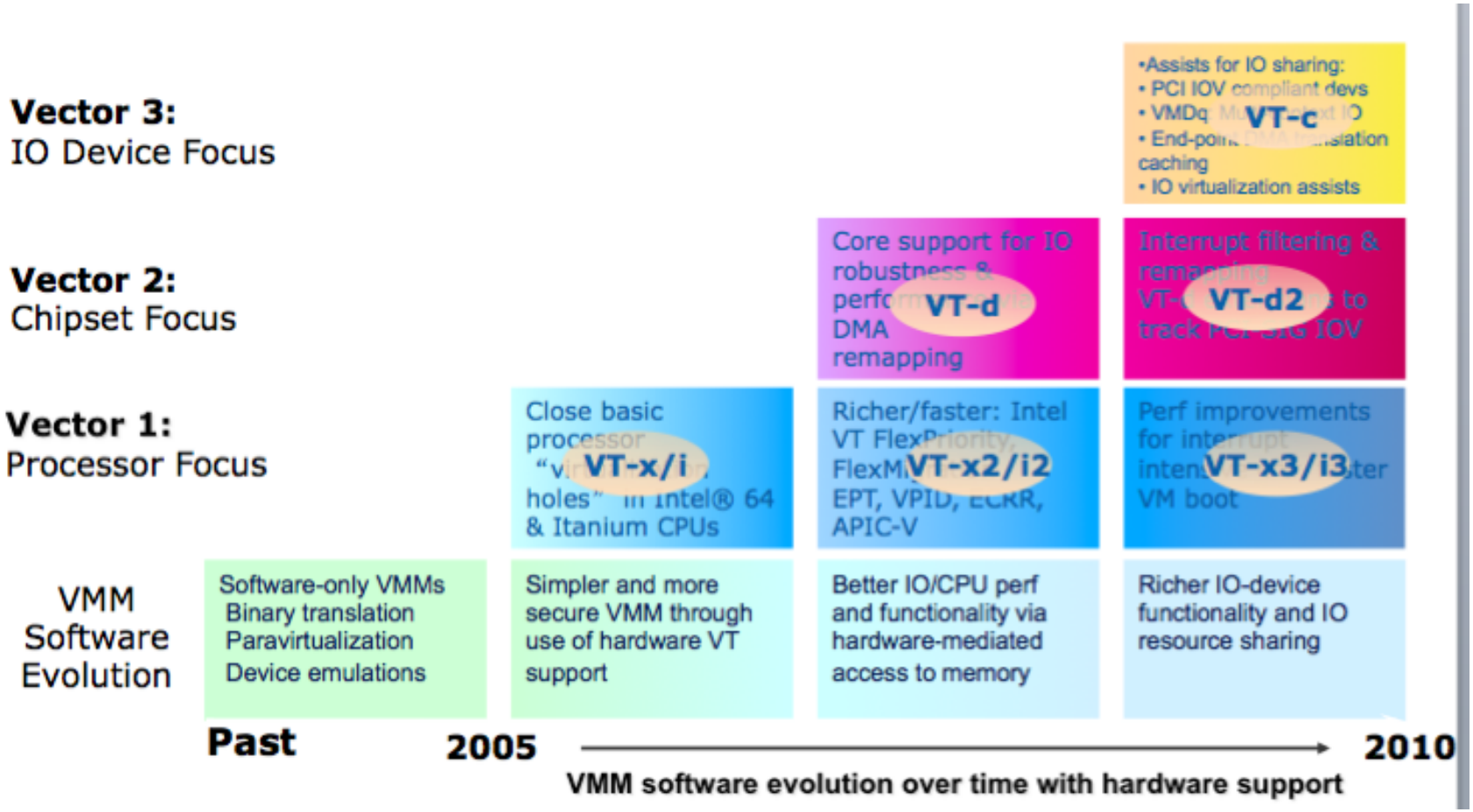
- Le support de la virtualisation peut être intégré au processeur ou assisté par celui-ci, le matériel se chargeant, par exemple, de virtualiser les accès mémoire ou de protéger le processeur physique des accès les plus bas niveau.
- Cela permet de simplifier la virtualisation logicielle et de réduire la dégradation de performances.
- Exemples :
  - Hyperviseur IBM Power4 & Micro-partitionnement AIX
  - Mainframes : VM/CMS
  - Sun LDOM (hyperviseur pour la gestion de "logical domains")
  - Sun E10k/E15k
  - HP Superdome
  - AMD-V (Assistance à la virtualisation de AMD, anciennement Pacifica)
  - Intel VT (Assistance à la virtualisation de Intel, anciennement Vanderpool)

Hardware support for full virtualization and paravirtualization

- Intel (et AMD) utilise des nouvelles technologies de virtualisation afin de supporter les hyperviseurs sur les architectures x86 (VT-x) ou sur les Itanium (VT-i)
- VT-x supporte deux nouveaux ensemble d'instructions :
  - un pour les VMM (root)
  - un pour les OS dans des VM (non-root).
- Les instructions root sont des opérations privilégiées, alors que les instructions non-root sont non-privilégiées (même en ring 0)
- L'architecture permet aussi de d'utiliser des instructions permettant à une VM de sortir vers le VMM tout en sauvegardant l'état du processeur.


Ring 3
User Apps
Ring 2
Ring 1
Ring 0
Guest OS
Non-root Mode Privilege Levels
Root Mode Privilege Levels
VMM
Host Computer System Hardware
Direct Execution of User Requests
OS Requests Trap to VMM without Binary Translation or Paravirtualization

Intel Virtualization Technology Evolution



Sommaire

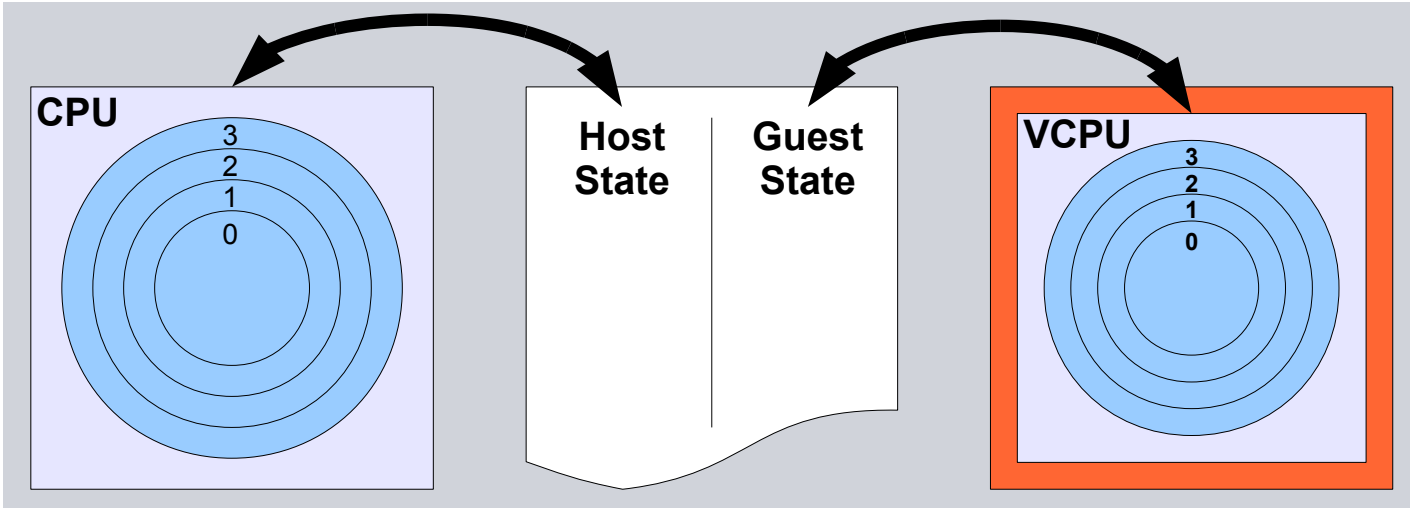
Sommaire

Virtualizing the x86 Instruction Set Architecture

- x86 originally virtualization "unfriendly"
  - No hardware provisions
  - Instructions behave differently depending on privilege context
  - Performance suffered on trap-and-emulate
  - CISC nature complicates instruction replacements
- Early approaches to x86 virtualization
  - Binary translation (e.g. VMware)
    - Execute substitution code for privileged guest code
    - May require substantial replacements to preserve illusion
  - CPU paravirtualization (e.g Xen)
    - Guest is aware of instruction restrictions
    - Hypervisor provides replacement services (hypercalls)
    - Raised abstraction levels for better performance

Hardware-assisted x86 CPU Virtualization

- Two variants
  - Intel's Virtualization Technology, VT-x
  - AMD-V (aka Secure Virtual Machine)
- Identical core concept



QEMU

- Processor emulator
- CPU emulator (e.g. x86, ARM)
- Emulated devices (e.g. VGA,IDE HD)
- Generic devices (e.g. network devices)
- Machine descriptions (e.g. PC)
- Debugger
- User Interface
- QEMU Accelerator (kernel module)

Kernel-based Virtual Machine (KVM)

- Full Virtualization Solution
- Requires Hardware Support (Intel VT or AMD-V)
- Two main components :
  - Kernel module as VMM
  - User land based on QEMU
- A virtual machine is scheduled like any other process
- The kernel component of KVM is included in mainline Linux, as of 2.6.20
- Windows/Linux/Unix guests (32-bit and 64-bit)
- Live Migration of guests from one host to another

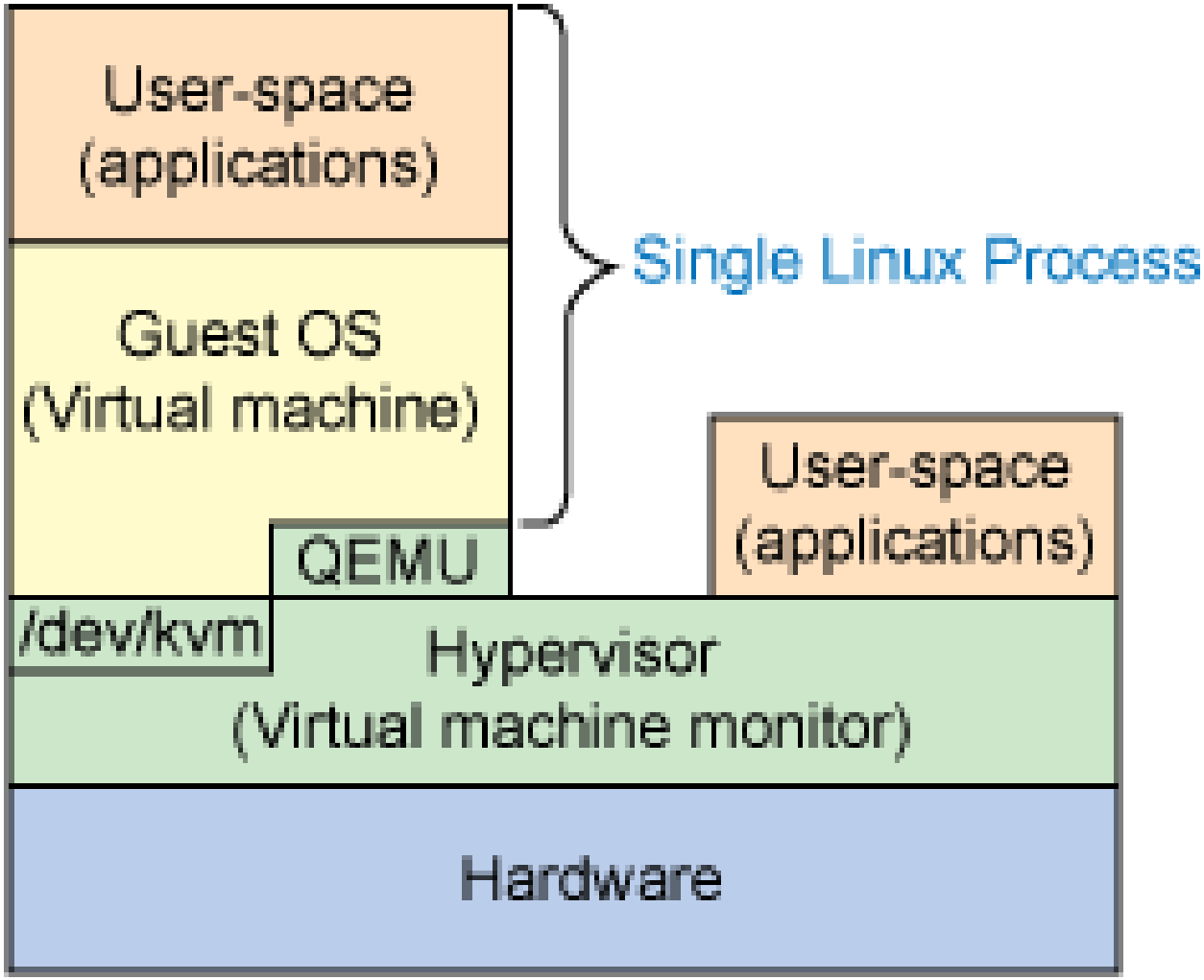
Linux KVM (Kernel Virtual Machine)

- The KVM module introduces a new execution mode into the kernel. Where vanilla kernels support kernel mode and user mode, the KVM introduces a guest mode. The guest mode is used to execute all non-I/O guest code, where normal user mode supports I/O for guests.
- The introduction of the KVM is an interesting evolution of Linux, as it represents the first virtualization technology that is part of the mainline Linux kernel. It exists in the 2.6.20 tree, but can be used as a kernel module for the 2.6.19 kernel. When run on hardware that supports virtualization, Linux (32-and 64-bit) and Windows (32-bit) guests are supported.

Advent and Evolution of KVM

- Introduced to make VT-x/AMD-V available to user space
  - Exposes virtualization features securely
  - Interface : /dev/kvm
- Merged quickly
  - Available since 2.6.20 (2006)
  - From first LKML posting to merge : 3 months
  - One reason : originally 100% orthogonal to core kernel
- Evolved significantly since then
  - Ported to further architectures (s390, PowerPC, IA64)
  - Always with latest x86 virtualization features
  - Became recognized & driving part of Linux

KVM/Qemu

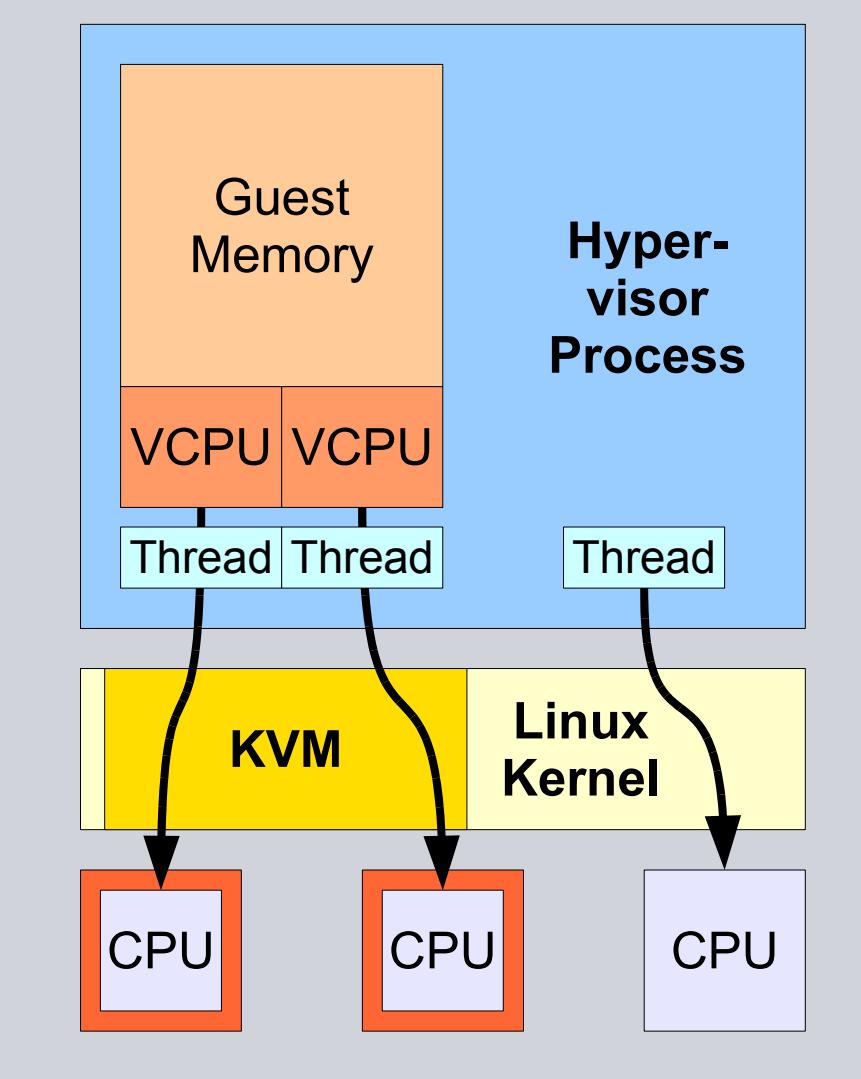

User-space
(applications)
Guest OS
(Virtual machine)
QEMU
Single Linux Process
User-space
(applications)
/dev/kvm
Hypervisor
(Virtual machine monitor)
Hardware

Sommaire

The KVM Model

- Processes can create virtual machines
- VMs can contain
  - Memory
  - Virtual CPUs
  - In-kernel device models
- Guest physical memory part of creating process' address space
- VCPUs run in process execution contexts
  - Process usually maps VCPUs on threads

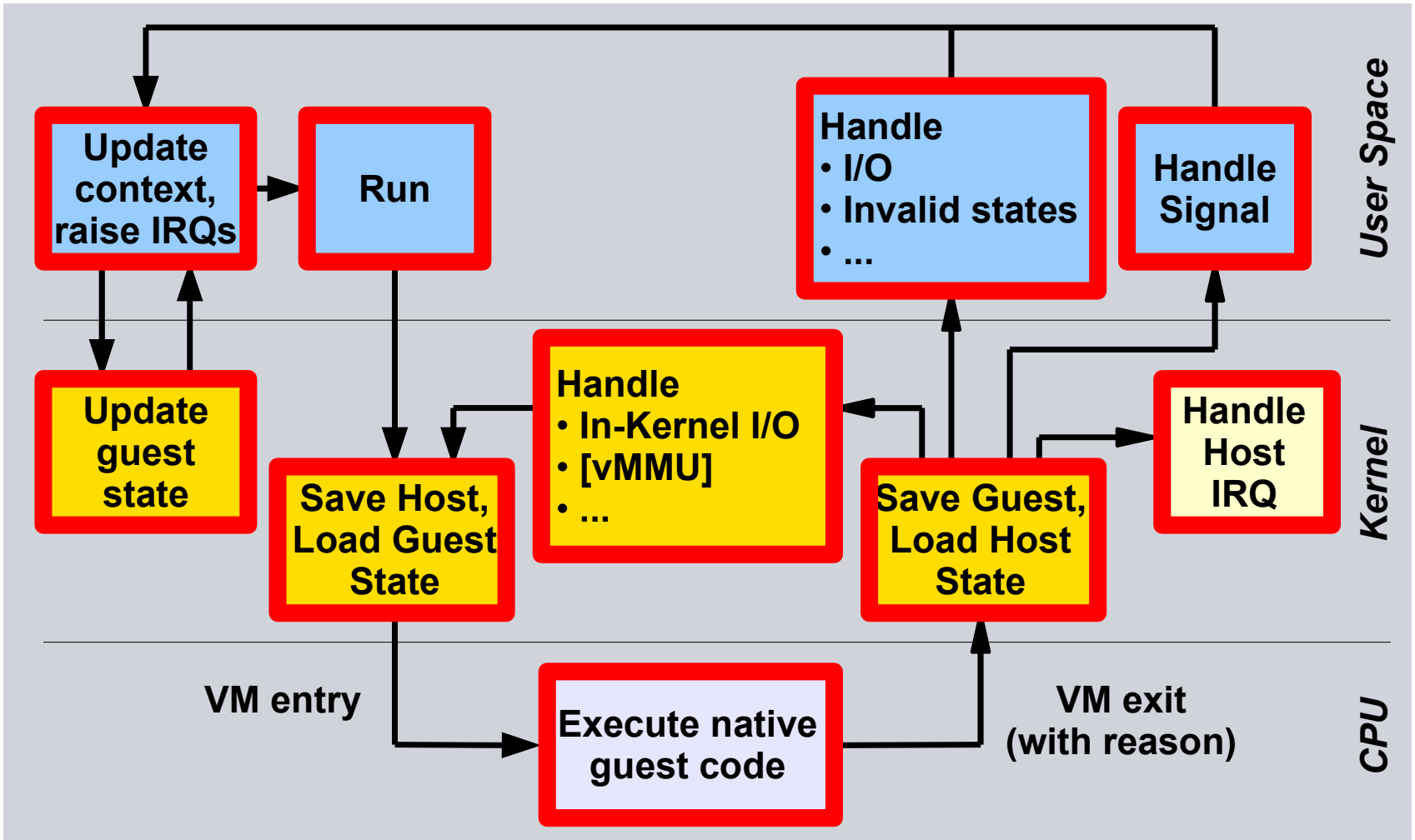
KVM/Qemu


Guest Memory
Hyper-visor Process
VCPU
VCPU
Thread
Thread
Thread
KVM
Linux Kernel
CPU
CPU
CPU

Architectural Advantages of the KVM Model

- Proximity of guest and user space hypervisor
  - Only one address space switch : guest $<->$ host
  - Less rescheduling
- Massive Linux kernel reuse
  - Scheduler
  - Memory management with swapping (though you don't what this)
  - I/O stacks
  - Power management
  - Host CPU hot-plugging
  - ...
- Massive Linux user land reuse
  - Network configuration
  - Handling VM images
  - Logging, tracing, debugging
  - ...

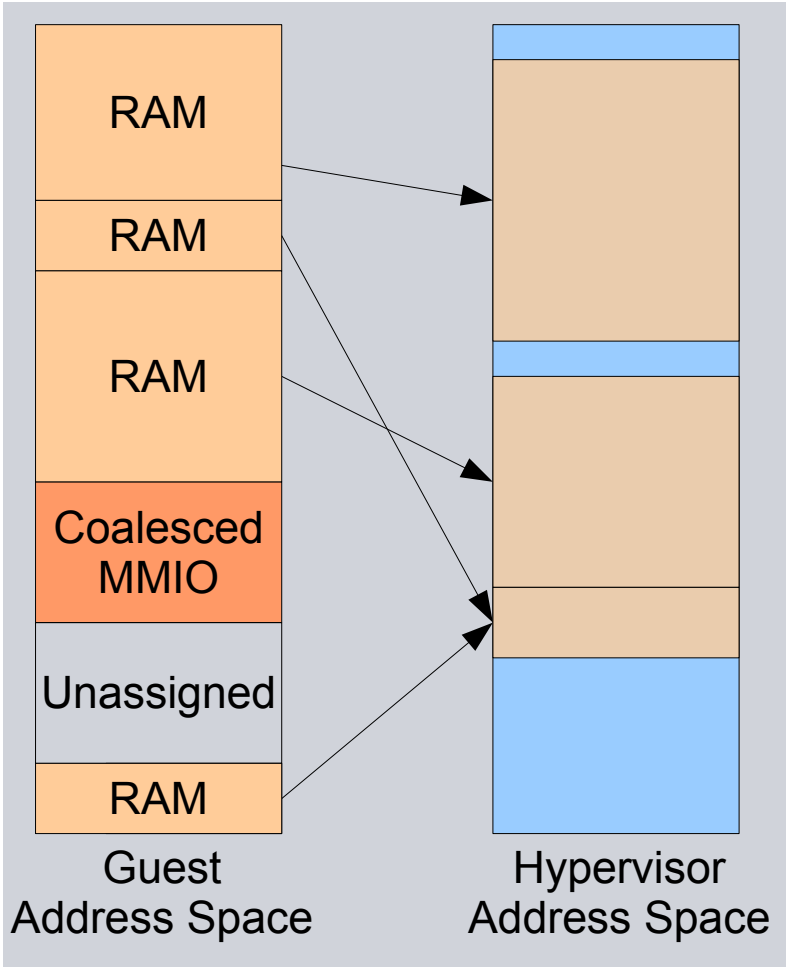
VCPU Execution Flow (KVM View)



KVM Memory Model

- Slot-Based guest memory
  - Maps guest physical to host virtual memory
  - Reconfigurable
  - Supports dirty tracking
- In-Kernel Virtual MMU
- Coalesced MMIO
  - Optimizes guest access to RAM-like virtual MMIO regions
- Out of scope
  - Memory ballooning (guest $<->$ user space hypervisor)
  - Kernel Same-page Merging (not KVM-specific)

KVM Memory Model



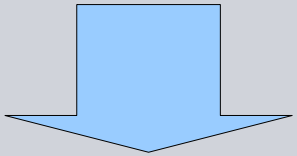
Sommaire

KVM API Overview

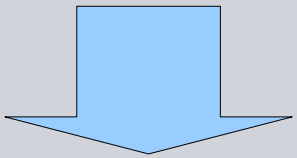
- Step 1 : open /dev/kvm
- Three groups of IOCTLs
  - System-level requests
  - VM-level requests
  - VCPU-level requests
- Per-group file descriptors
  - /dev/kvm fd for system level
  - Creating a VM or VCPU returns new fd
- mmap on file descriptors
  - VCPU : fast kernel-user communication segment
  - Frequently read/modified part of VCPU state
  - Includes coalesced MMIO backlog
  - VM : map guest physical memory (deprecated)

Basic KVM IOCTLs

KVM_CREATE_VM



KVM_SET_USER_MEMORY_REGION
KVM_CREATE_IRQCHIP / ...PIT (x86)
KVM_CREATE_VCPU



KVM_SET_REGS / ...SREGS / ...FPU / ...
KVM_SET_CPUID / ...MSRS / ...VCPU_EVENTS / ... (x86)
KVM_SET_LAPIC (x86)
KVM_RUN

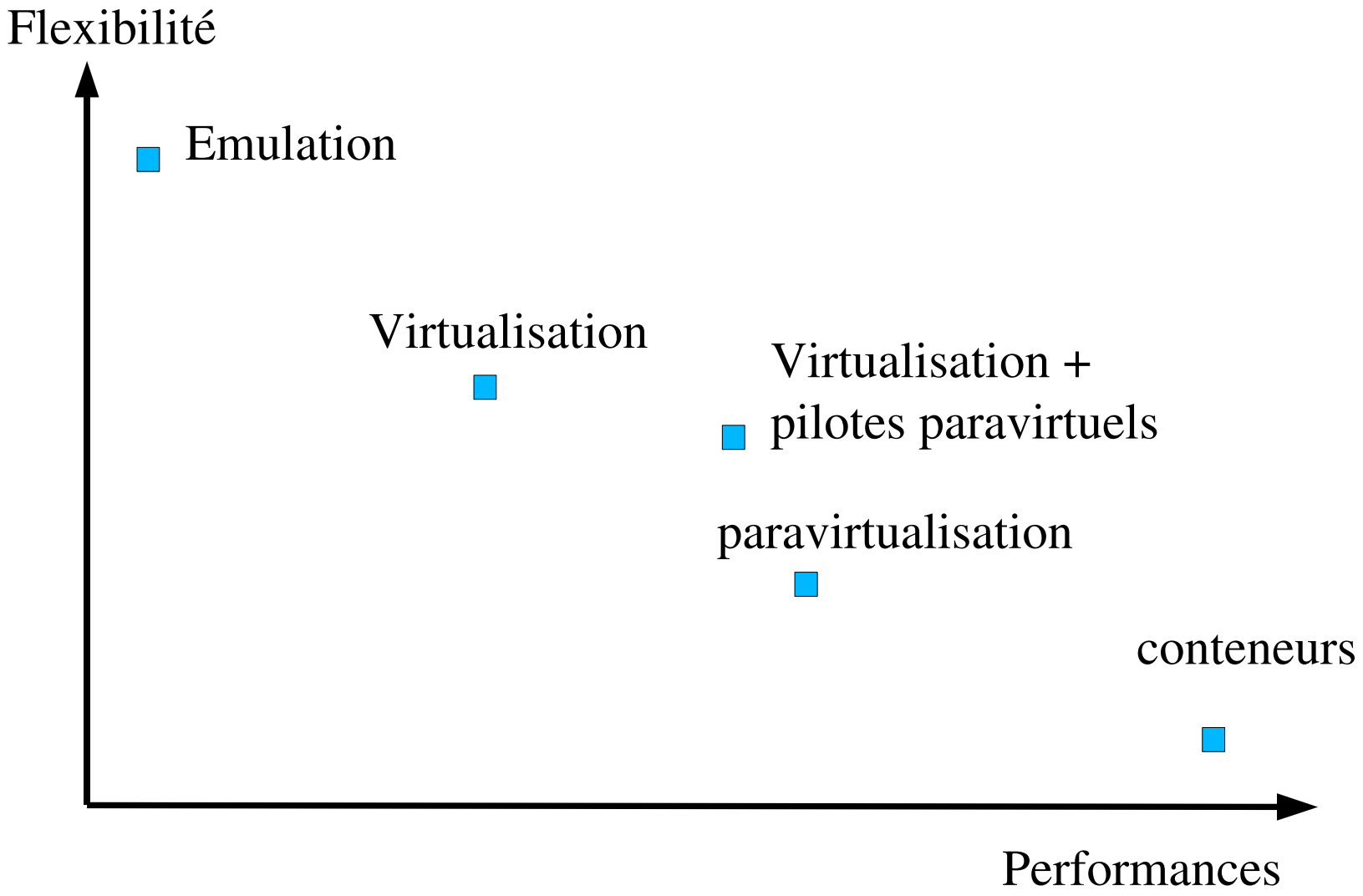
Sommaire

Paravirtual Devices

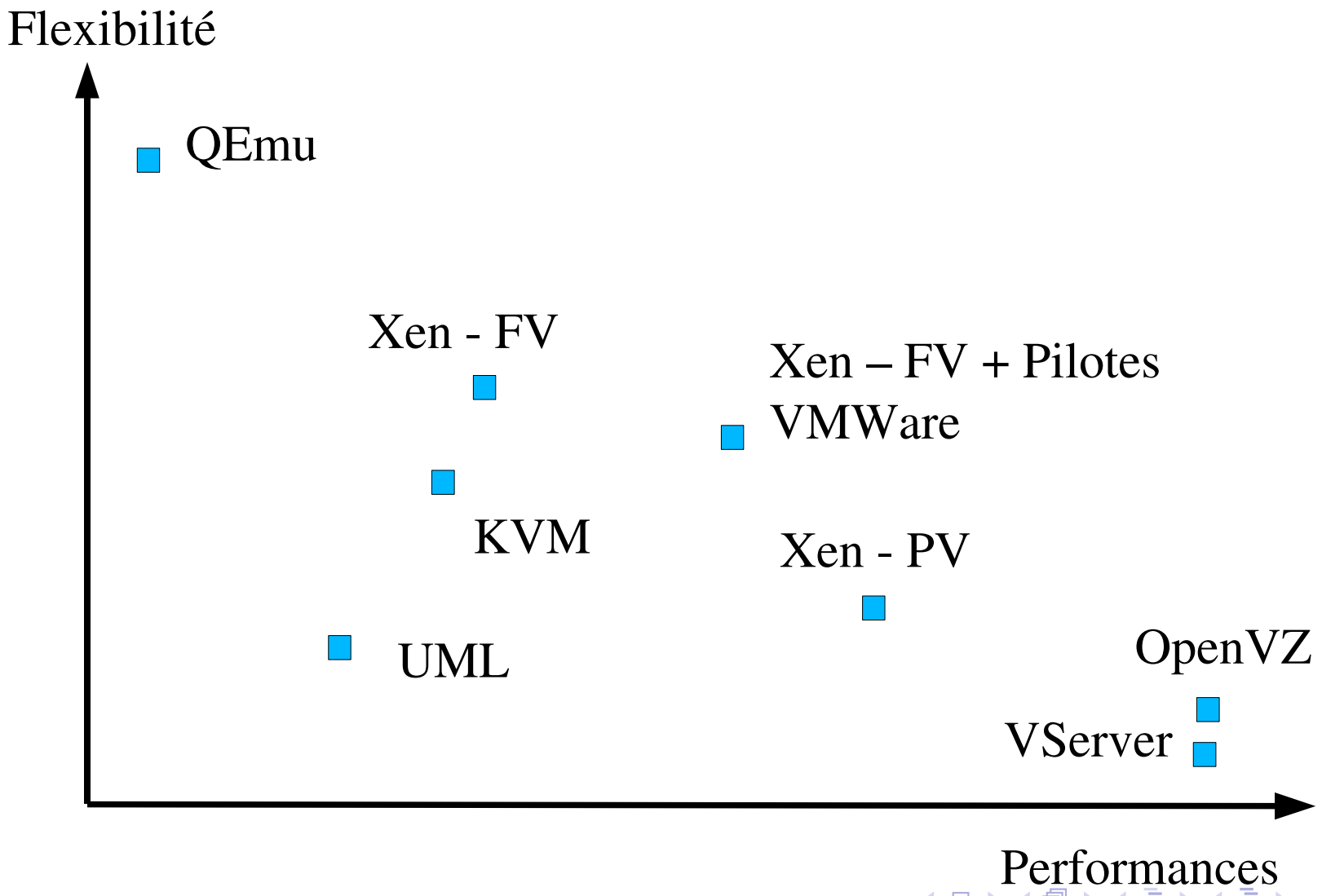
- Advantages
- Reduce VM exits or make them lightweight
- Improve I/O throughput & latency (less emulation)
- Compensates virtualization effects
- Enable direct host-guest interaction
- Available interfaces & implementions
  - virtio (PCI or alternative transports)
    - Network
    - Block
    - Serial I/O (console, host-guest channel, ...)
    - Memory balloon
    - File system (9P)
  - Clock (x86 only)
    - Via shared page + MSRs
    - Enables safeTM TSC guest usage

Sommaire

Types


Flexibilité
Emulation
Virtualisation
Virtualisation +
pilotes paravirtuels
paravirtualisation
conteneurs
Performances

Solutions


Flexibilité
QEmu
Xen - FV
Xen – FV + Pilotes
VMWare
KVM
Xen - PV
UML
OpenVZ
VServer
Performances

Sommaire

Sommaire

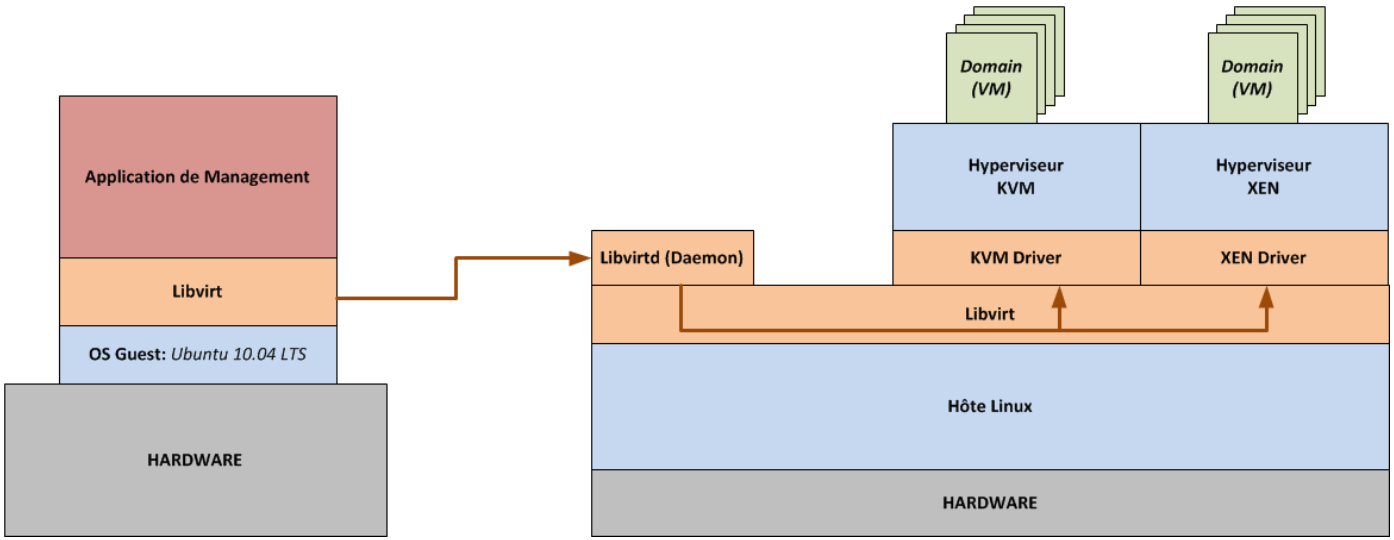
LibVirt

- API stable pour la virtualisation
  - LGPL `http://www.libvirt.org`
  - Relativement portable : Solaris, Windows, OS-X, Linux, ...
- Opérations de base
  - Créer/Détruire/suspendre/sauver/migrer
  - Description XML des machine virtuelles
  - Statistiques, support NUMA
  - Gestion du stockage
- Ecrit en C
  - Interfaces python, perl, Java, Ocaml, Ruby

Libvirt et compagnie

- Virsh : script de contrôle de la virtualisation
- Virt-manager : interface graphique de management
  - Cycle de vie complet
  - Création, migration
  - Concole locale et distantes
- Libvirt-CIM
  - implémentation CIM par IBM (LGPL)
  - Pegasus et SFCB
  - Xen et KVM
  - Cobbler
    - Environnement d'installations automatiques
    - koan : agent de réinstallation

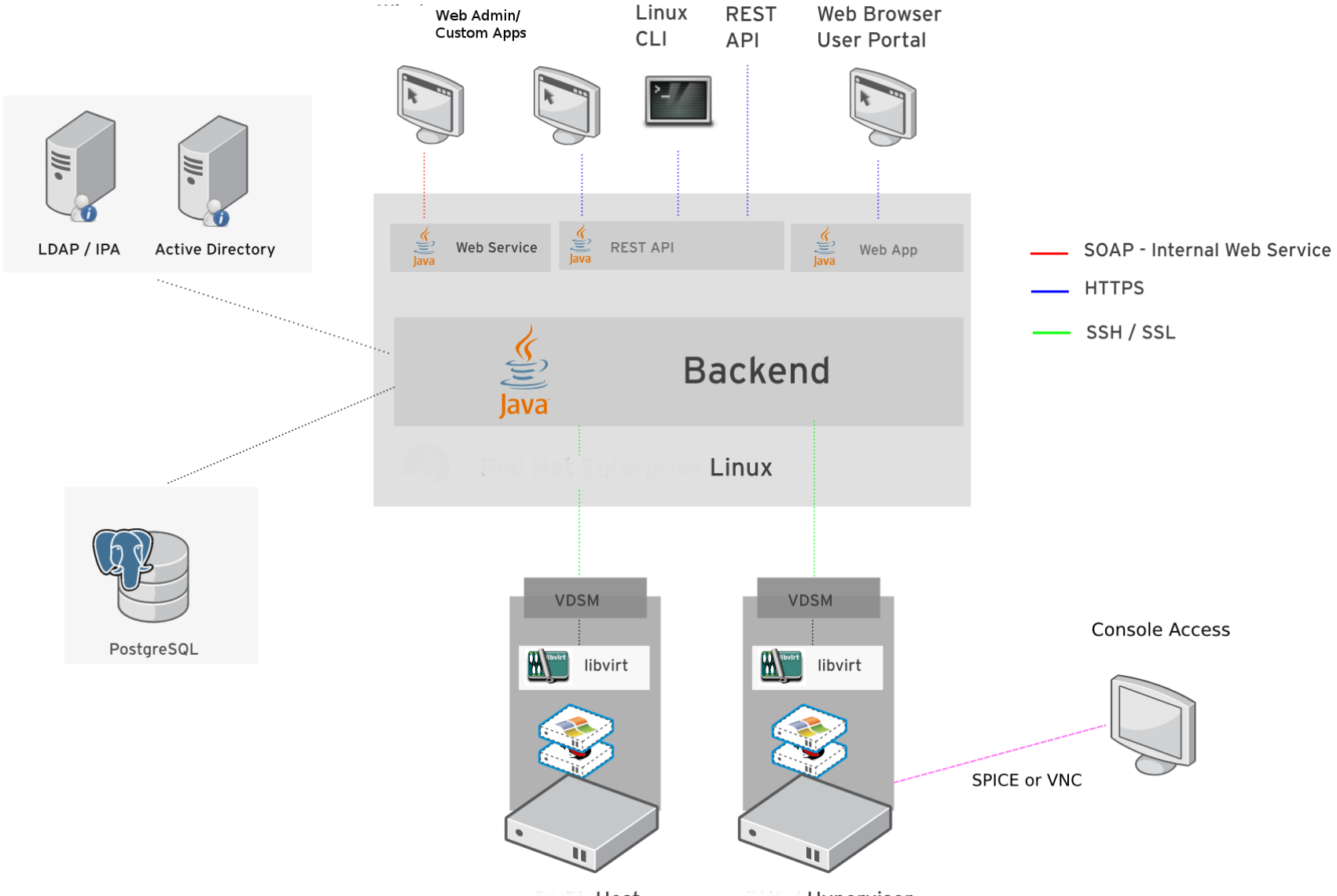
Solutions


Application de Management
Libvirt
OS Guest: Ubuntu 10.04 LTS
HARDWARE
Libvirtd (Daemon)
Domain (VM)
Domain (VM)
Hyperviseur KVM
Hyperviseur XEN
KVM Driver
XEN Driver
Libvirt
Hôte Linux
HARDWARE

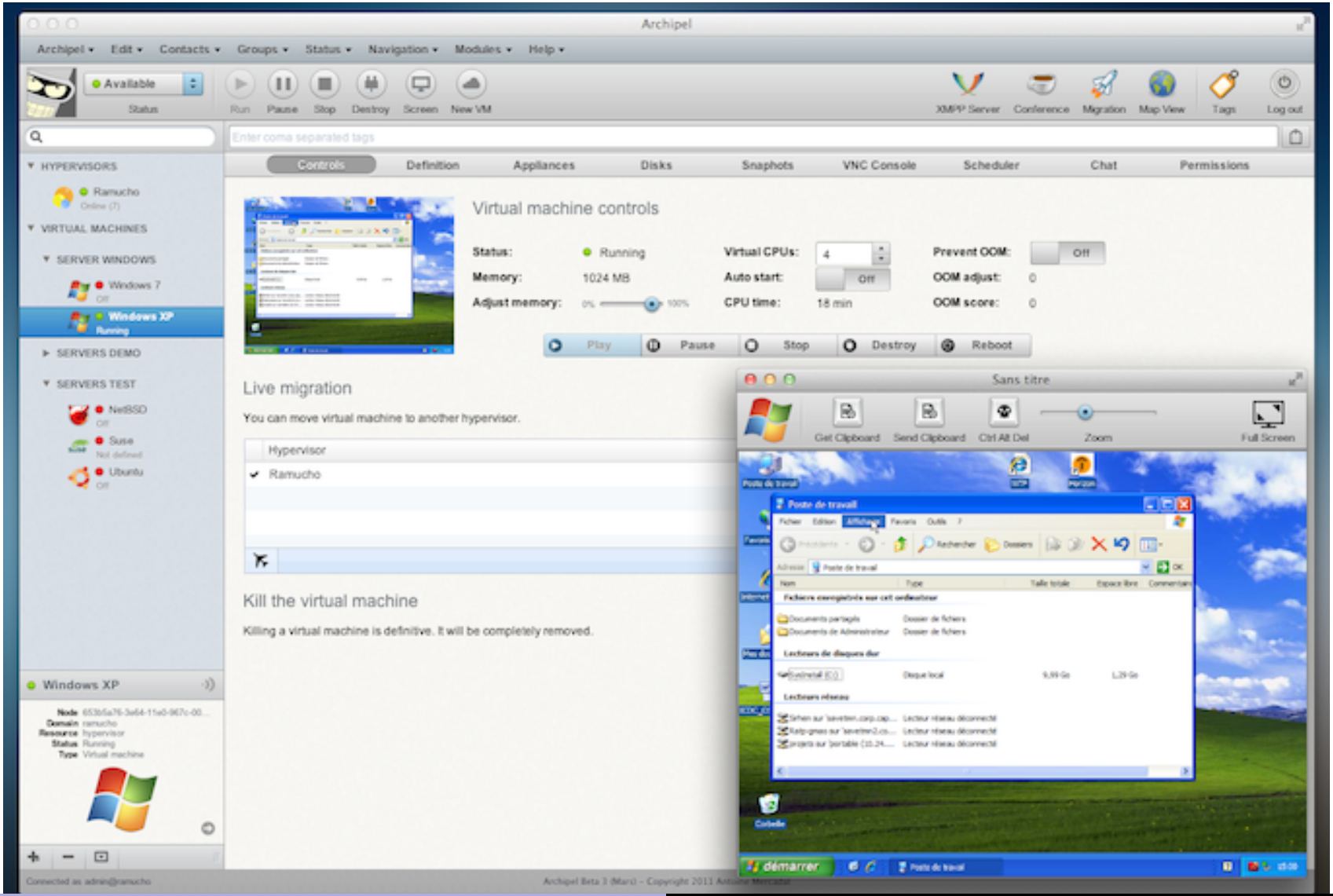
Libvirt et compagnie

- Virt-p2v
  - Outil de migration automatique
  - CD Boot, accès distant
  - `http://et.redhat.com/~rjones/virt-p2v/`
- Ovirt
  - Interface de gestion d'un ensemble de serveurs
  - Intégre le stockage, kerberos, DHCP, PXE
  - `http://ovirt.org`
- Archipel
  - Interface web de gestion de serveur KVM
  - Gestion des droits et des ACL
  - utilise le protocole XMPP

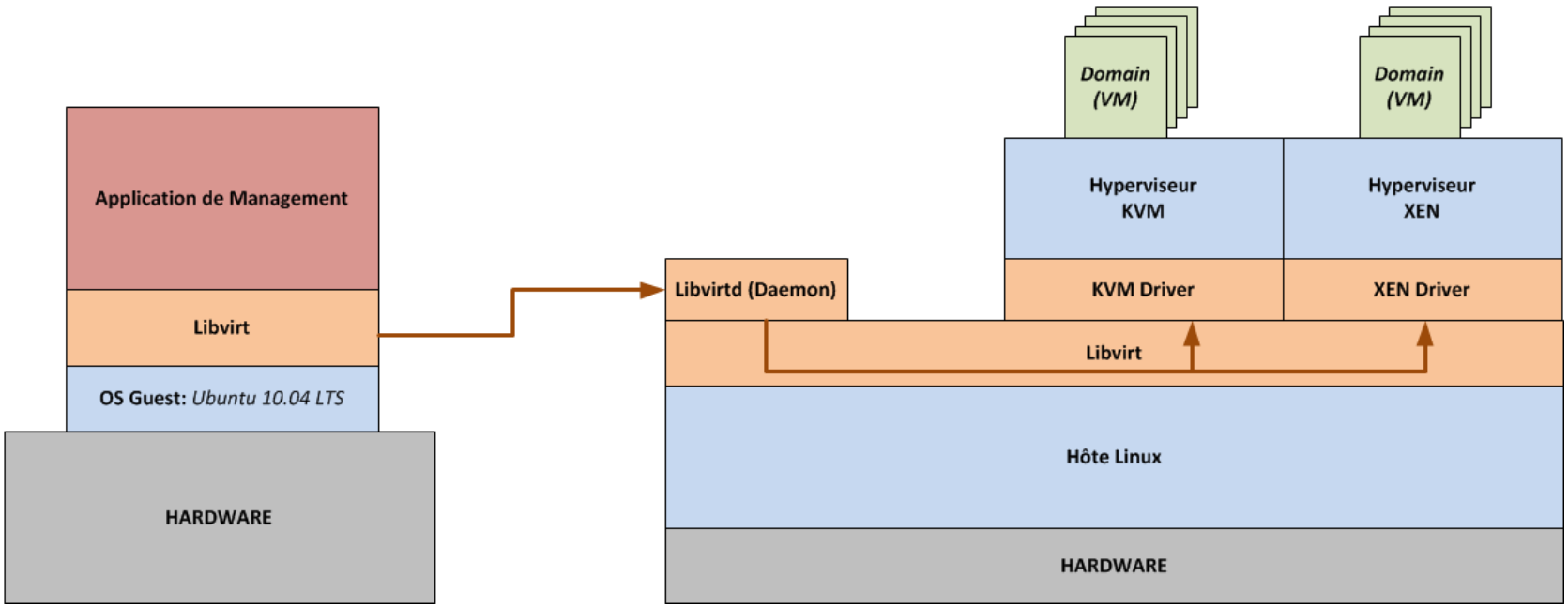
Ovirt


Web Admin/
Custom Apps
Linux
CLI
REST
API
Web Browser
User Portal
LDAP / IPA
Active Directory
Web Service
REST API
Web App
Java
Backend
Linux
PostgreSQL
VDSM
libvirt
VDSM
libvirt
Console Access
SPICE or VNC
SOAP - Internal Web Service
HTTPS
SSH / SSL

Archipel


Archipel
Archipel
Edit
Contacts
Groups
Status
Navigation
Modules
Help
Available
Status
Run
Pause
Stop
Destroy
Screen
New VM
XMPP Server
Conference
Migration
Map View
Tags
Log out
Enter coma separated tags
HYPERVISORS
Ramucho
VIRTUAL MACHINES
SERVER WINDOWS
Windows 7
Windows XP
SERVERS DEMO
SERVERS TEST
NetBSD
Suse
Ubuntu
Controls
Definition
Appliances
Disks
Snaphots
VNC Console
Scheduler
Chat
Permissions
Virtual machine controls
Status:
Running
Memory:
1024 MB
Adjust memory:
Virtual CPUs:
4
Auto start:
Off
CPU time:
18 min
Prevent OOM:
Off
OOM adjust:
0
OOM score:
0
Play
Pause
Stop
Destroy
Reboot
Live migration
You can move virtual machine to another hypervisor.
Hypervisor
Ramucho
Kill the virtual machine
Killing a virtual machine is definitive. It will be completely removed.
Windows XP
Sans titre
Get Clipboard
Send Clipboard
Ctrl Alt Del
Zoom
Full Screen
Poste de travail
démarrer

Archipel

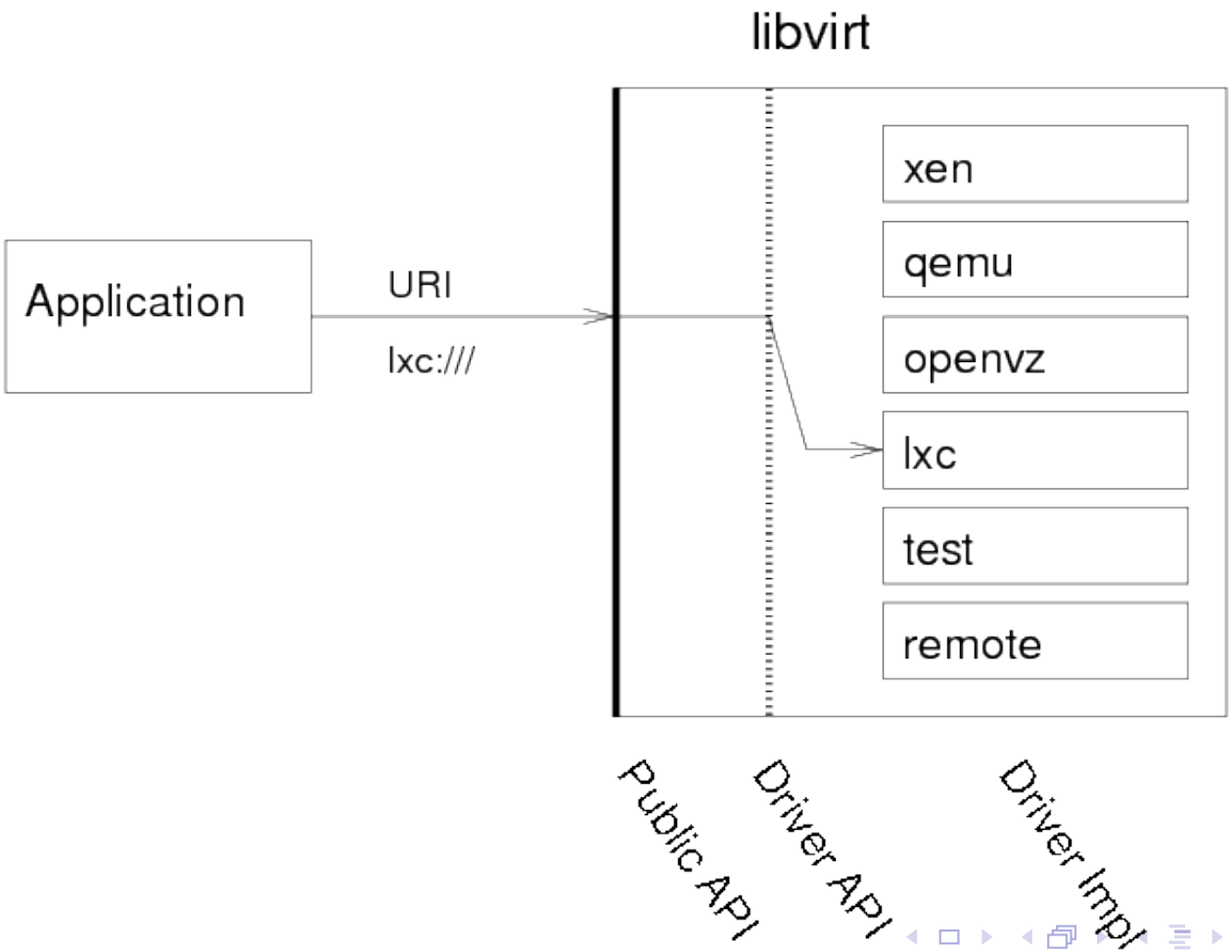

Application de Management
Libvirt
OS Guest: Ubuntu 10.04 LTS
HARDWARE
Libvirtd (Daemon)
Domain (VM)
Domain (VM)
Hyperviseur KVM
Hyperviseur XEN
KVM Driver
XEN Driver
Libvirt
Hôte Linux
HARDWARE

Sommaire

Driver model

- LibVirt fourni une API permettant l'accès/gestion aux hyperviseurs via des drivers
- la fonction **virConnectPtr** permet d'obtenir une instance d'un driver en fonction d'une URI
- Hypervisor drivers URI :
  - xen : **xen :///**
  - QEMU : **qemu :///session**
  - UML : **uml :///session**
  - LXC : **lsc :///**
  - accès a distance à libvirtd : **qemu+tcp :///**
  - ...

Driver model



Driver model

- Accès distant supportés :
  - tls via libvirtd
  - tcp via libvirtd
  - socket unix
  - ssh
- Authentifications
  - sasl
  - polkit
  - x509 : possibilité de mettre en place une PKI

Connections

- Utilisation des fonctions virConnect*

```
virConnectPtr virConnectOpen(const char *name)
virConnectPtr virConnectOpenReadOnly(const char *name)
virConnectPtr virConnectOpenAuth(const char *name,
    virConnectAuthPtr auth, int flags)
```

Connections : exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <libvirt/libvirt.h>
int main(int argc, char *argv[])
{
    virConnectPtr conn;
    conn = virConnectOpen("qemu:///system");
    // conn = virConnectOpenReadOnly("qemu:///system");
    // conn = virConnectOpenAuth("qemu+tcp://localhost/
        system", virConnectAuthPtrDefault, 0);
    if (conn == NULL) {
        fprintf(stderr, "Failed to open connection to
            qemu:///system\n");
        return 1;
     }
    virConnectClose(conn);
    return 0;
}
```

Capability information

- la fonction **virConnectGetCapabilities** permet d'obtenir des informations sur les capacités de virtualisation de l'hôte
- résultat : sur forme d'un string contenant une description XML

Capability information : exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <libvirt/libvirt.h>
int main(int argc, char *argv[])
{
    virConnectPtr conn;
    char *caps;
    conn = virConnectOpen("qemu:///system");
    if (conn == NULL) {
        fprintf(stderr, "Failed to open connection to
            qemu:///system\n");
        return 1;
     }
    caps = virConnectGetCapabilities(conn);
    fprintf(stdout, "Capabilities:\n%s\n", caps);
    free(caps);
    virConnectClose(conn);
    return 0;
}
```

Capability information : résultat I

```
<capabilities>
 <host>
   <cpu>
     <arch>x86_64</arch>
   </cpu>
   <migration_features>
     <live/>
     <uri_transports>
       <uri_transport>tcp</uri_transport>
     </uri_transports>
   </migration_features>
   <topology>
     <cells num='1'>
       <cell id='0'>
         <cpus num='2'>
<cpu id='0'/>
           <cpu id='1'/>
```

Capability information : résultat II

```
        </cpus>
      </cell>
    </cells>
  </topology>
</host>
<guest>
  <os_type>hvm</os_type>
  <arch name='i686'>
    <wordsize>32</wordsize>
    <emulator>/usr/bin/qemu</emulator>
    <machine>pc</machine>
    <machine>isapc</machine>
    <domain type='qemu'>
    </domain>
    <domain type='kvm'>
      <emulator>/usr/bin/qemu-kvm</emulator>
    </domain>
```

Capability information : résultat III

```
    </arch>
    <features>
      <pae/>
      <nonpae/>
      <acpi default='on' toggle='yes'/>
      <apic default='on' toggle='no'/>
    </features>
  </guest>
  <guest>
    <os_type>hvm</os_type>
    <arch name='x86_64'>
      <wordsize>64</wordsize>
      <emulator>/usr/bin/qemu-system-x86_64</emulator>
      <machine>pc</machine>
      <machine>isapc</machine>
      <domain type='qemu'>
      </domain>
```

Capability information : résultat IV

```
      <domain type='kvm'>
        <emulator>/usr/bin/qemu-kvm</emulator>
      </domain>
    </arch>
    <features>
      <acpi default='on' toggle='yes'/>
      <apic default='on' toggle='no'/>
    </features>
  </guest>
</capabilities>
```

Host information

- **virConnectGetHostname** : retourne le hostname de l'hôte
- **virConnectGetMaxVcpus** :
- **virNodeGetFreeMemory** :
- **virNodeGetInfo** : retourne une structure contenant les informations de l'hôte
- **virNodeGetCellsFreeMemory**, **virConnectGetType**, **virConnectGetVersion**, **virConnectGetLibVersion**, **virConnectGetURI**, ...

Host information : exemple I

```
int main(int argc, char *argv[])
{
    virConnectPtr conn;
    virNodeInfo nodeinfo;
    conn = virConnectOpen("qemu:///system");
    if (conn == NULL) {
        fprintf(stderr, "Failed to open connection to
            qemu:///system\n");
        return 1; }
    virNodeGetInfo(conn, &nodeinfo);
    fprintf(stdout, "Model: %s\n", nodeinfo.model);
    fprintf(stdout, "Memory size: %lukb\n", nodeinfo.
        memory);
    fprintf(stdout, "Number of CPUs: %u\n", nodeinfo.
        cpus);
    fprintf(stdout, "MHz of CPUs: %u\n", nodeinfo.mhz);
```

Host information : exemple II

```
    fprintf(stdout, "Number of NUMA nodes: %u\n",
        nodeinfo.nodes);
    fprintf(stdout, "Number of CPU sockets: %u\n",
        nodeinfo.sockets);
    fprintf(stdout, "Number of CPU cores per socket: %u
        \n", nodeinfo.cores);
    fprintf(stdout, "Number of CPU threads per core: %u
        \n", nodeinfo.threads);
    virConnectClose(conn);
    return 0;
}
```

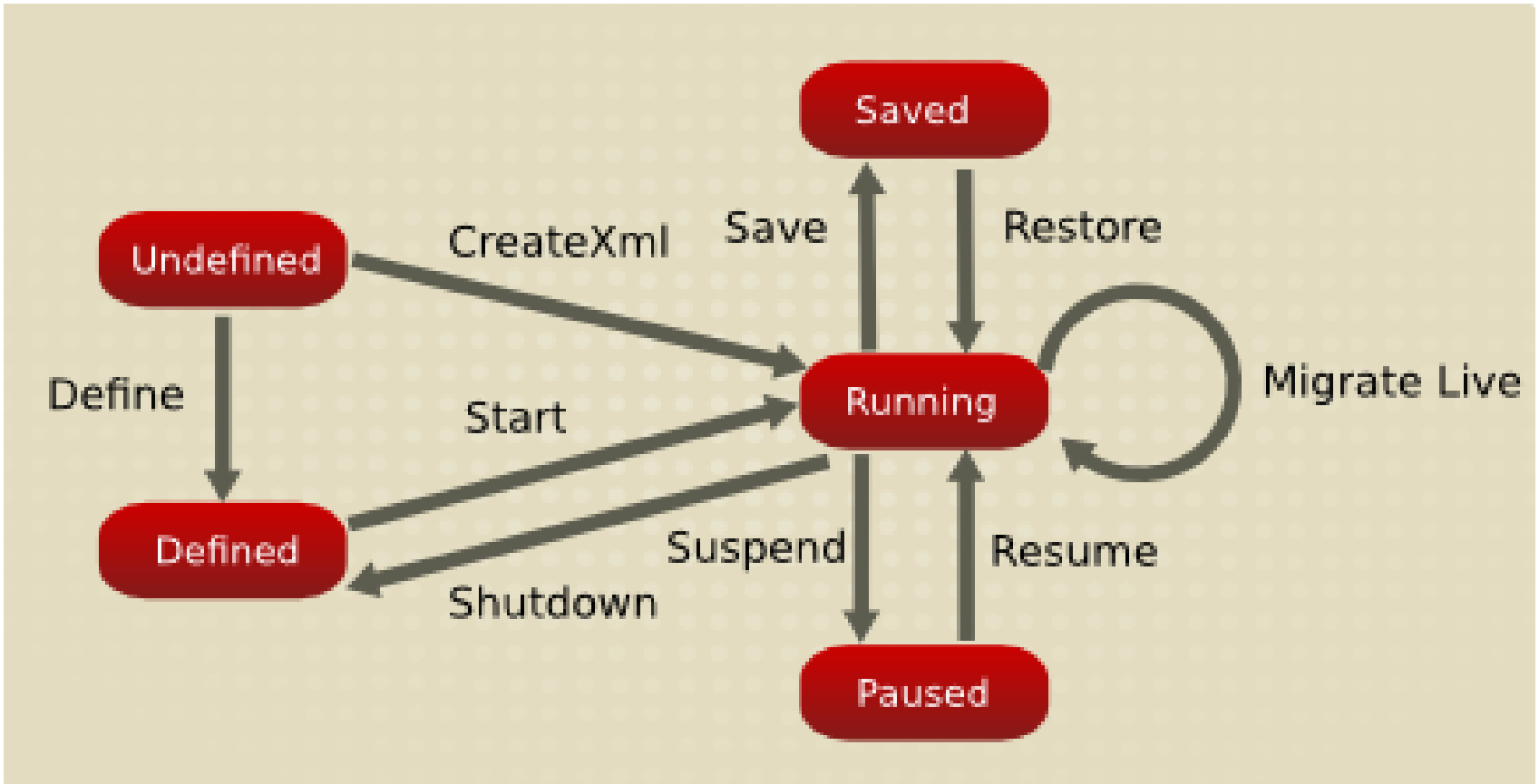
Gestions des domaines

- Un domaine est une VM
- représenté par **virDomainPtr**
  - ID : un entier
  - name : un nom
  - UUID : 16 bytes non-signés
- fonction pour rechercher un domaine :
  - **virDomainLookupByID**, **virDomainLookupByName**, **virDomainLookupByUUIDString**
  - **virConnectListDomains**, **virConnectNumOfDomains**, **virConnectNumOfDefinedDomains**

Gestions des domaines : exemple I

```
int i;
int numDomains;
int *activeDomains;
numDomains = virConnectNumOfDomains(conn);
activeDomains = malloc(sizeof(int) * numDomains);
numDomains = virConnectListDomains(conn, activeDomains,
     numDomains);
printf("Active domain IDs:\n");
for (i = 0 ; i < numDomains ; i++) {
     printf("  %d\n", activeDomains[i]);
}
free(activeDomains);
```

Lifecycle control



Lifecycle control API

- **virConnectCreateXML** : charger la représentation d'un domaine à partir de sa description XML
- **virDomainCreate** : démarrer un domaine
- **virDomainFree** : arrêter un domaine
- **(virDomainSave** : Suspendre un domaine
- **virDomainRestore** : restaurer un domaine (à partir d'un fichier)
- ...

Lifecycle control : exemple I

```
const char *xml = "<domain>....</domain>";
virDomainPtr dom;
dom = virDomainDefineXML(conn, xml);
if (!dom) {
    fprintf(stderr, "Unable to define persistent guest
        configuration");
return; }
if (virDomainCreate(dom) < 0) {
    fprintf(stderr, "Unable to boot guest configuration
        ");
}
```

Autres fonctions disponibles :

- Informations sur le modèle de sécurité utilisé
- Gestion des erreurs (Error handling)
- Gestion des périphériques
- Monitoring
- Gestion du réseau
- Debuging / Logging ...