

Virtualisation et Cloud OpenStack
Filière : Ingénierie Système d'informations et Big Data

KVM : Une Approche Professionnelle de la Virtualisation sur Linux

Encadré par :

Pr. Abdellah JAMALI

Présenté le : 08/11/2024

Préparé par :

KHALIS Samir
MALYANA Hamza

Année universitaire : 2024-2025

PLAN

**01
INTRODUCTION**

**02
VIRTUALISATION**

**03
ARCHITECTURE DE
KVM**

**04
LIBVERT**

**05
QEMU**

**06
KVM**

**07
CONCLUSION**

INTRODUCTION

KVM joue un rôle central dans l'optimisation et la flexibilité des infrastructures modernes. Il permet de maximiser l'utilisation des ressources matérielles tout en assurant une isolation fiable entre différents environnements. Au-delà de la simple virtualisation, KVM contribue à un écosystème où conteneurs, réseaux et stockage peuvent être gérés de manière flexible et sécurisée, répondant ainsi aux exigences croissantes des entreprises en matière de performance et d'évolutivité.



VIRTUALISATION

Définition :

- La virtualisation est une technologie permettant de créer plusieurs environnements virtuels en utilisant les ressources d'une machine physique unique, maximisant ainsi l'utilisation de sa capacité. Cela permet à une seule machine physique d'héberger plusieurs systèmes d'exploitation ou applications **isolées**, chacun fonctionnant comme un ordinateur distinct. (*d'après IBM*)
- En pratique, un logiciel appelé **hyperviseur** gère la distribution des ressources matérielles (comme la mémoire et le processeur) pour chaque environnement virtuel, ce qui facilite la gestion, **réduit** les coûts d'infrastructure, et **optimise** l'efficacité des systèmes en entreprise.

VIRTUALISATION - TYPES

La virtualisation dans le secteur des technologies de l'information peut être analysée sous deux axes principaux : le premier axe concerne **l'objet de la virtualisation** (par exemple, les serveurs, le réseau, le stockage, etc.), tandis que le second axe concerne **la méthode de virtualisation** employée (telles que la partition, la virtualisation complète, la paravirtualisation, la virtualisation hybride et la virtualisation basée sur des conteneurs). Chaque combinaison de ces axes répond à des besoins spécifiques, qu'ils soient matériels ou logiciels.

Voici un aperçu des principaux types de virtualisation utilisés aujourd'hui dans le secteur des technologies de l'information :

VIRTUALISATION - TYPES

1. Virtualisation des serveurs:

La virtualisation des serveurs est le premier type de virtualisation rencontré. Cette technique implique le regroupement de plusieurs serveurs virtuels en un seul serveur physique, et ce à l'aide d'une couche logicielle. Chacune des machines virtuelles créée agit ensuite de manière autonome et isolée, exécutant ses propres système d'exploitation et applications.

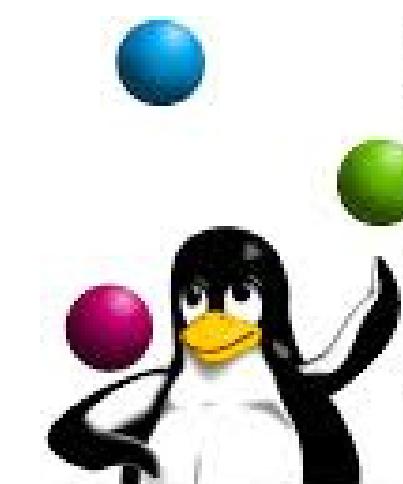
Ce type de virtualisation repose sur le rôle de l'hyperviseur, c'est-à-dire du logiciel, installé sur le serveur physique, qui assure la gestion des différents OS invités.

Il existe deux types d'hyperviseurs :

VIRTUALISATION - TYPES

- **l'hyperviseur de type 1 ou bare metal :**

il opère **directement** sur le hardware, et devient de ce fait l'outil de contrôle du système d'exploitation. Les OS invités s'exécutent alors par dessus cet hyperviseur.



VIRTUALISATION - TYPES

- **l'hyperviseur de type 2, ou host metal :**

fonctionne en tant qu'application installée sur un système d'exploitation hôte (host operating system).



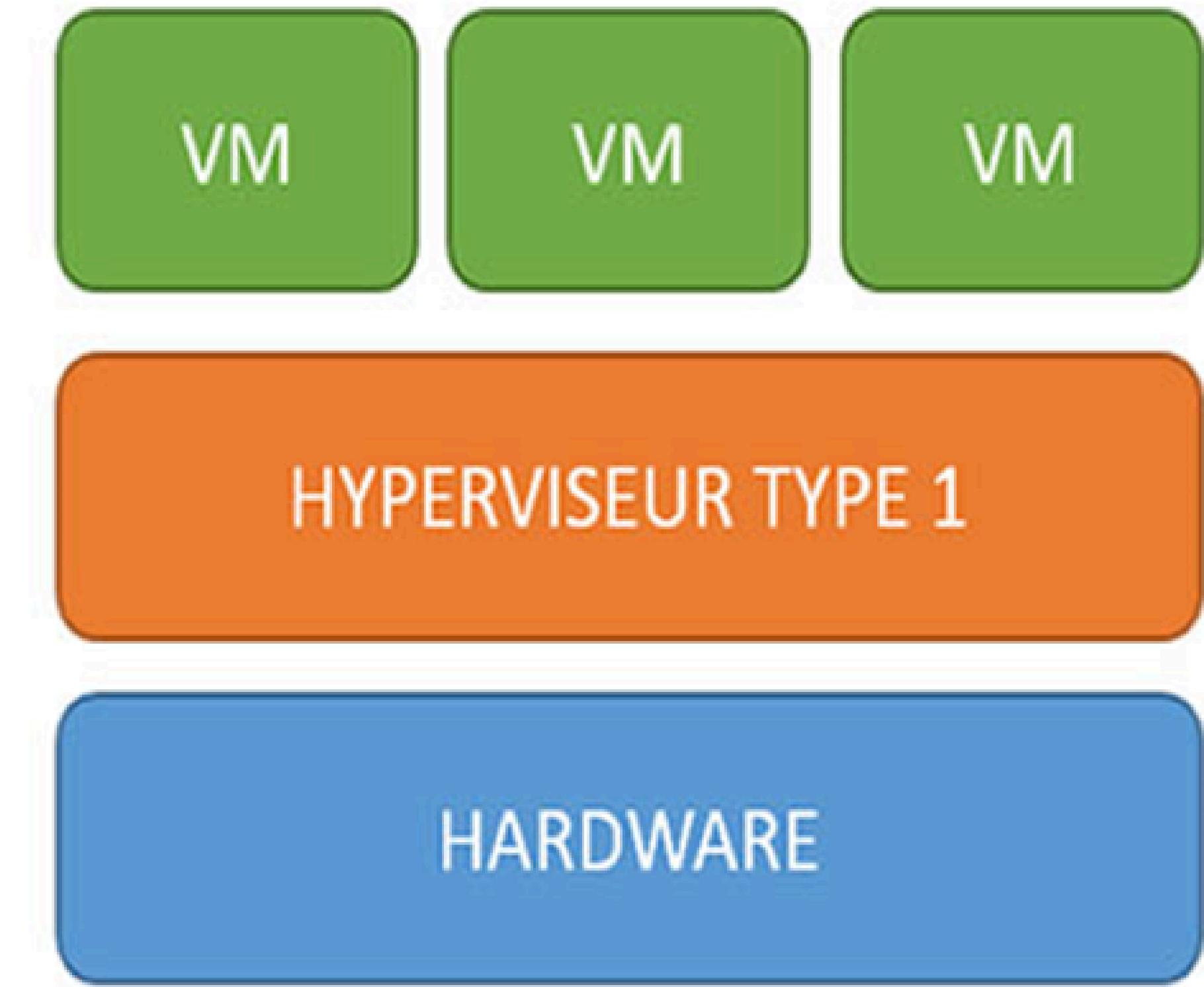
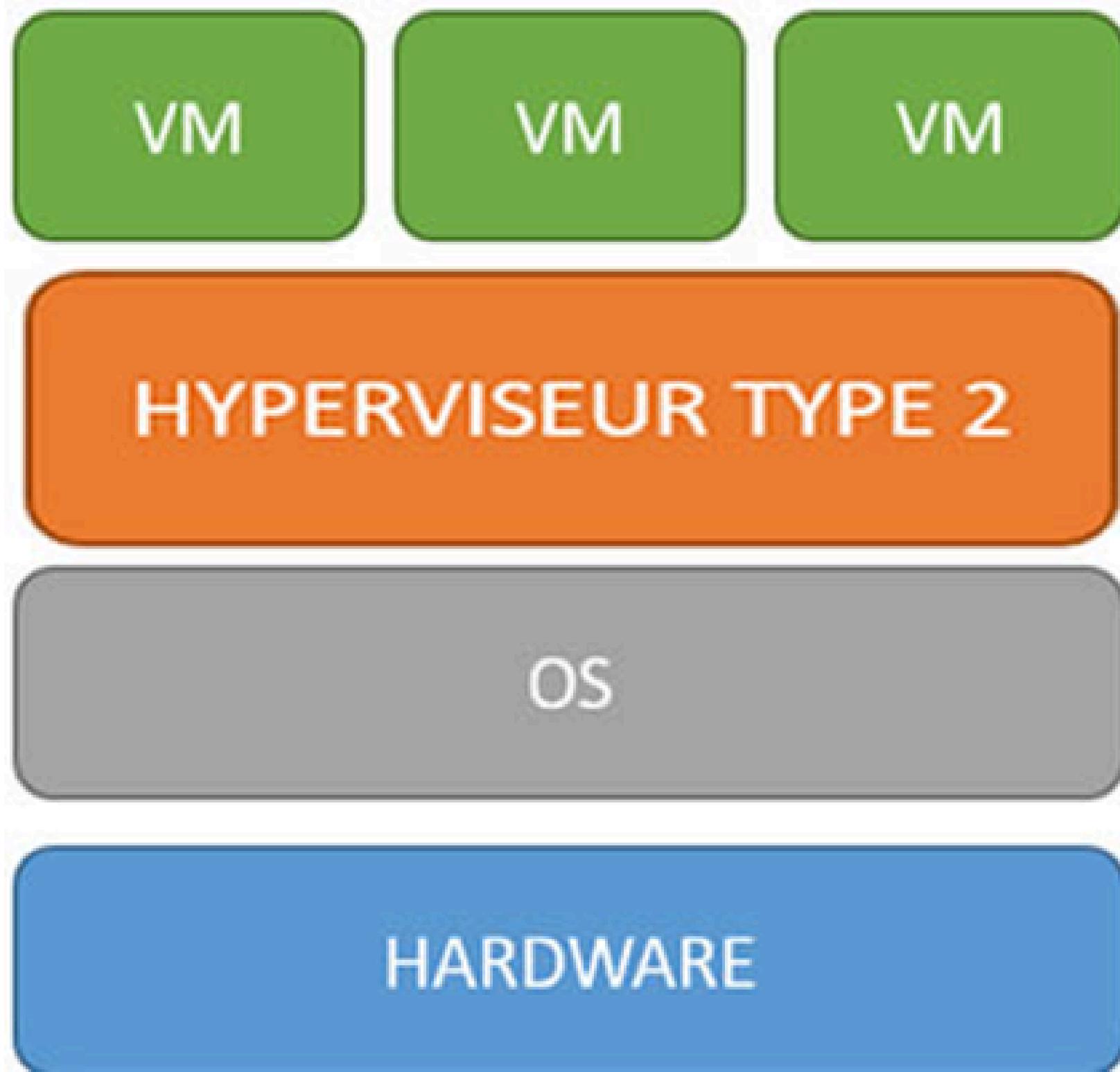
QEMU

//**Parallels**



VirtualBox

VIRTUALISATION - TYPES



Pourquoi opter pour la virtualisation des serveurs ?

- **Réduction du Nombre de Serveurs** : La virtualisation permet de diminuer le nombre de serveurs utilisés. Dans un environnement non virtualisé, chaque serveur n'est exploité qu'à 15-20 % de ses capacités (d'après Dell).
- **Optimisation des Ressources** : Grâce à la virtualisation, les ressources sont mieux optimisées, ce qui entraîne une utilisation plus efficace de l'infrastructure informatique.
- **Réduction des Coûts** : La virtualisation contribue également à la réduction des coûts liés au matériel et à la maintenance, permettant ainsi des économies significatives pour les entreprises.

2. Virtualisation des systèmes d'exploitation :

La virtualisation des systèmes d'exploitation, utilisée parfois à l'échelle domestique, permet d'exécuter sur une seule et même machine plusieurs OS différents, **n'interférant pas les uns avec les autres.**

Exemple : naviguer sur un même ordinateur d'un environnement Windows à un environnement Linux.

Pourquoi opter pour la virtualisation des systèmes d'Exploitation ?

- **Exécution d'Applications Spécifiques :**

Grâce à la virtualisation, il est possible de lancer des applications nécessitant un autre système d'exploitation ou une version antérieure de celui-ci. Cela permet d'accéder à des logiciels qui ne sont pas compatibles avec le système actuel.

- **Familiarisation avec de Nouveaux Systèmes :**

La virtualisation des systèmes d'exploitation constitue également un excellent moyen de se familiariser avec un OS méconnu. Cela offre l'opportunité d'explorer et d'apprendre à utiliser un nouvel environnement sans modifier la configuration principale.

Pourquoi opter pour la virtualisation des systèmes d'Exploitation ?

- **Exécution d'Applications Spécifiques :**

Grâce à la virtualisation, il est possible de lancer des applications nécessitant un autre système d'exploitation ou une version antérieure de celui-ci. Cela permet d'accéder à des logiciels qui ne sont pas compatibles avec le système actuel.

- **Familiarisation avec de Nouveaux Systèmes :**

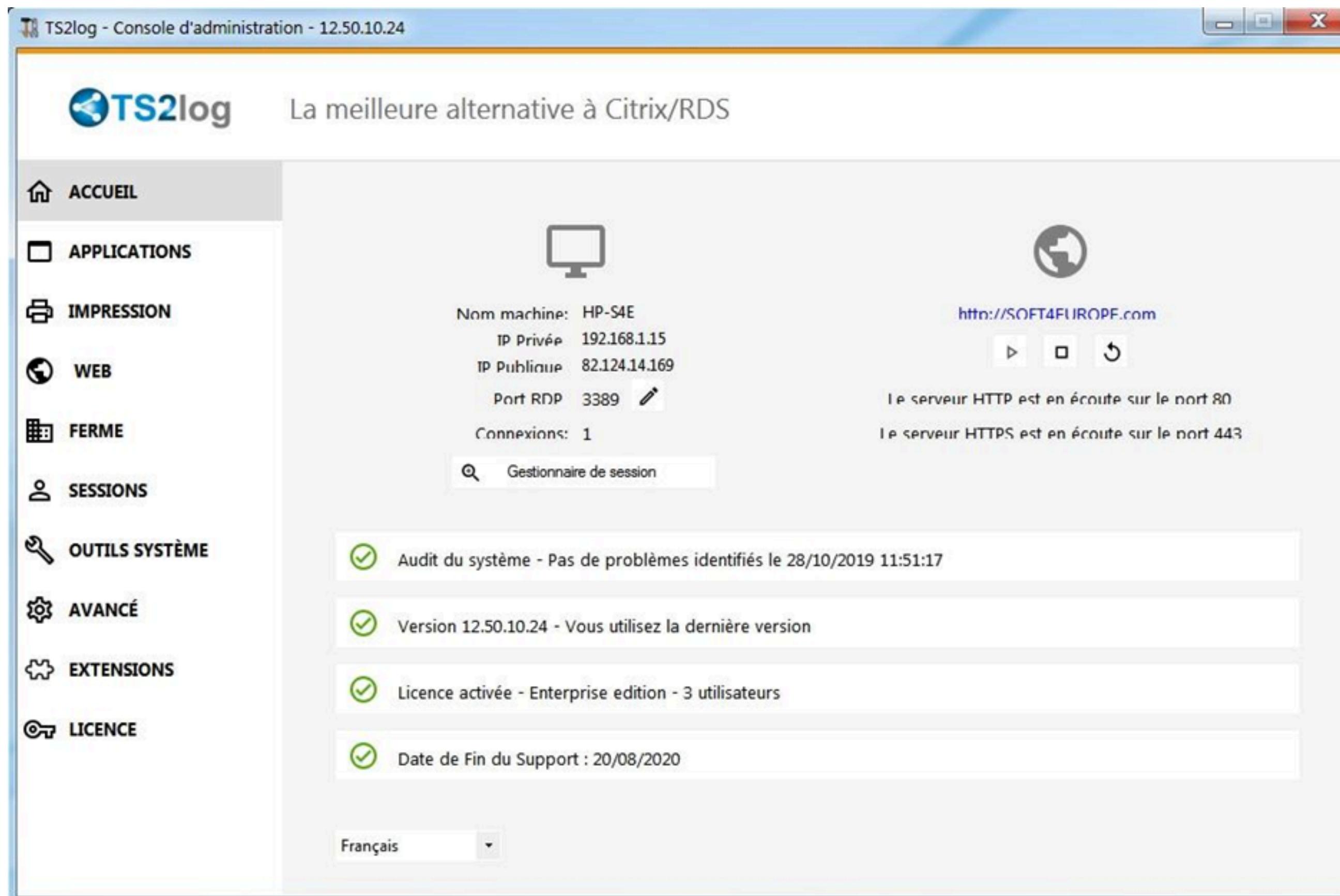
La virtualisation des systèmes d'exploitation constitue également un excellent moyen de se familiariser avec un OS méconnu. Cela offre l'opportunité d'explorer et d'apprendre à utiliser un nouvel environnement sans modifier la configuration principale.

3. Virtualisation des postes de travail :

La virtualisation des postes de travail est une technique essentielle pour les entreprises, car elle permet aux employés d'accéder à leur environnement de travail de manière flexible et sécurisée, quel que soit leur emplacement. En étant hébergés sur un serveur VDI (Virtual Desktop Infrastructure), les utilisateurs profitent d'une expérience informatique uniforme et efficace, favorisant ainsi la productivité.

VIRTUALISATION - TYPES

Exemple de console d'administration d'un desktop virtualisé :



Pourquoi opter pour la virtualisation des postes de travail ?

- **Flexibilité accrue :**

La virtualisation des postes de travail permet une grande flexibilité, en particulier dans les situations de mobilité, ce qui facilite le travail à distance.

- **Transfert simplifié des environnements :**

Cette technique facilite le transfert des environnements de travail aux équipes de sous-traitants, permettant une collaboration efficace sans interruption.

- **Réduction des coûts :**

Les coûts associés à la multiplication des postes de travail, notamment ceux liés aux licences de logiciels, sont considérablement réduits grâce à la centralisation des ressources.

4. Virtualisation des applications :

Il y a virtualisation des applications lorsque celles-ci s'exécutent sous une forme encapsulée (regroupement des données brutes) et indépendante du système d'exploitation sous-jacent.

Exemple : utiliser une application Linux sur un environnement Windows.

Pourquoi opter pour la virtualisation des applications ?

- **Accessibilité des applications** : Grâce à la virtualisation des applications, il est possible d'accéder à vos applications favorites, quel que soit le système d'exploitation, l'appareil utilisé ou la version installée.
- **Multi-version** : Cette technique permet d'exécuter plusieurs versions d'une même application sur un seul système d'exploitation, facilitant ainsi les tests et la compatibilité.
- **Flexibilité et mobilité** : Comme pour la virtualisation des postes de travail, la virtualisation des applications favorise la flexibilité et l'agilité, tout en facilitant la mobilité des utilisateurs.

VIRTUALISATION - MÉTHODES

Types de virtualisation contribuant à la technologie de virtualisation

Virtualisation basée sur des conteneurs

Utilise des conteneurs pour le déploiement d'applications

Virtualisation hybride

Combine les avantages de la virtualisation complète et de la paravirtualisation

Paravirtualisation

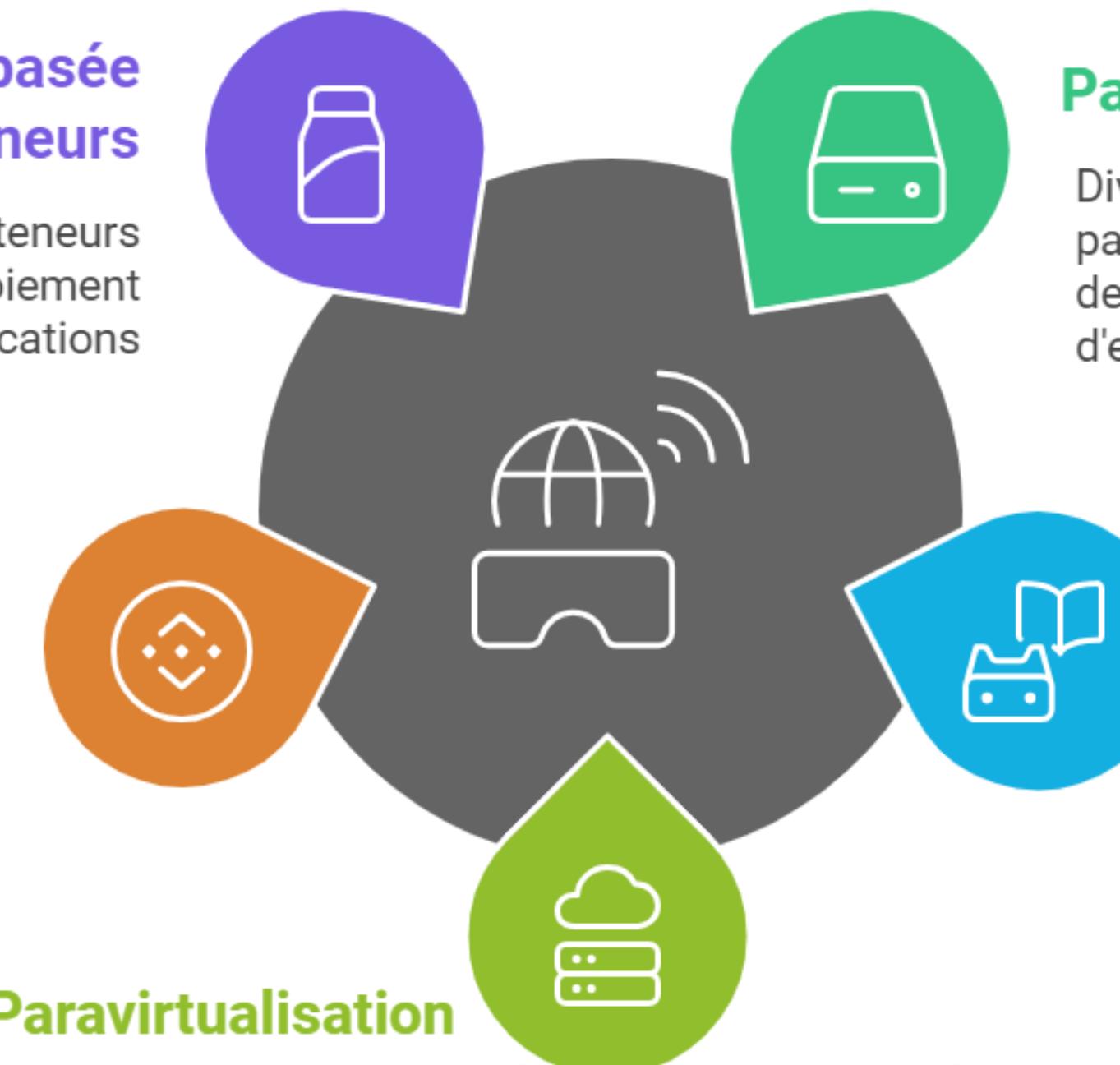
Système d'exploitation conscient de la virtualisation, nécessite une modification

Partitionnement

Division du CPU en partitions isolées pour des systèmes d'exploitation distincts

Virtualisation complète

Simulation du matériel sans modification du système d'exploitation



VIRTUALISATION - EXIGENCES

Exigences matérielles de virtualisation

Architecture 64 bits

Permet d'adresser plus de 4 Go de RAM pour de meilleures performances.

Compatibilité du processeur

Prend en charge la virtualisation avec une technologie assistée par le matériel pour l'efficacité.

RAM suffisante

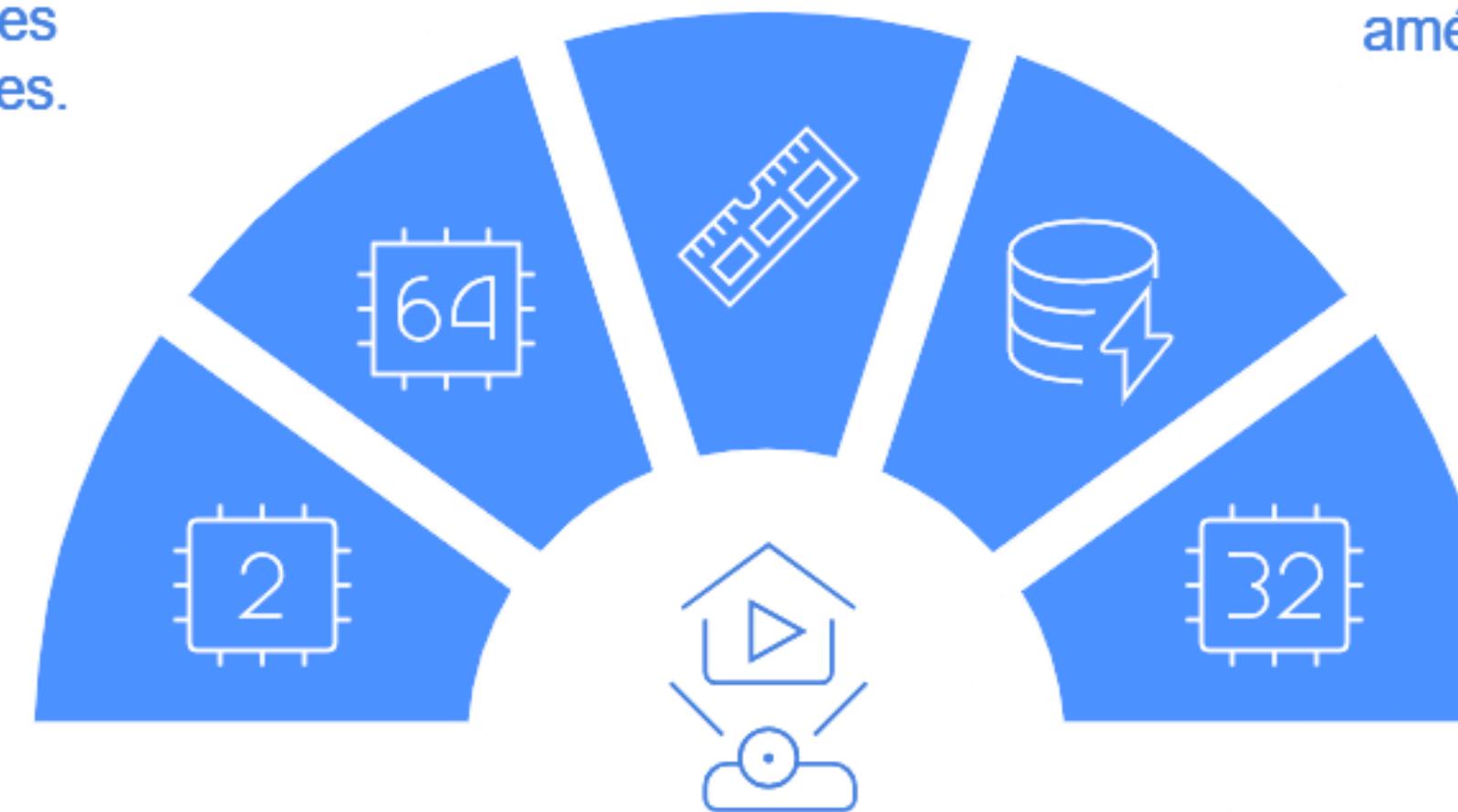
Fournit la mémoire nécessaire pour plusieurs machines virtuelles.

Stockage rapide

Réduit les temps de démarrage des VM et améliore la réactivité.

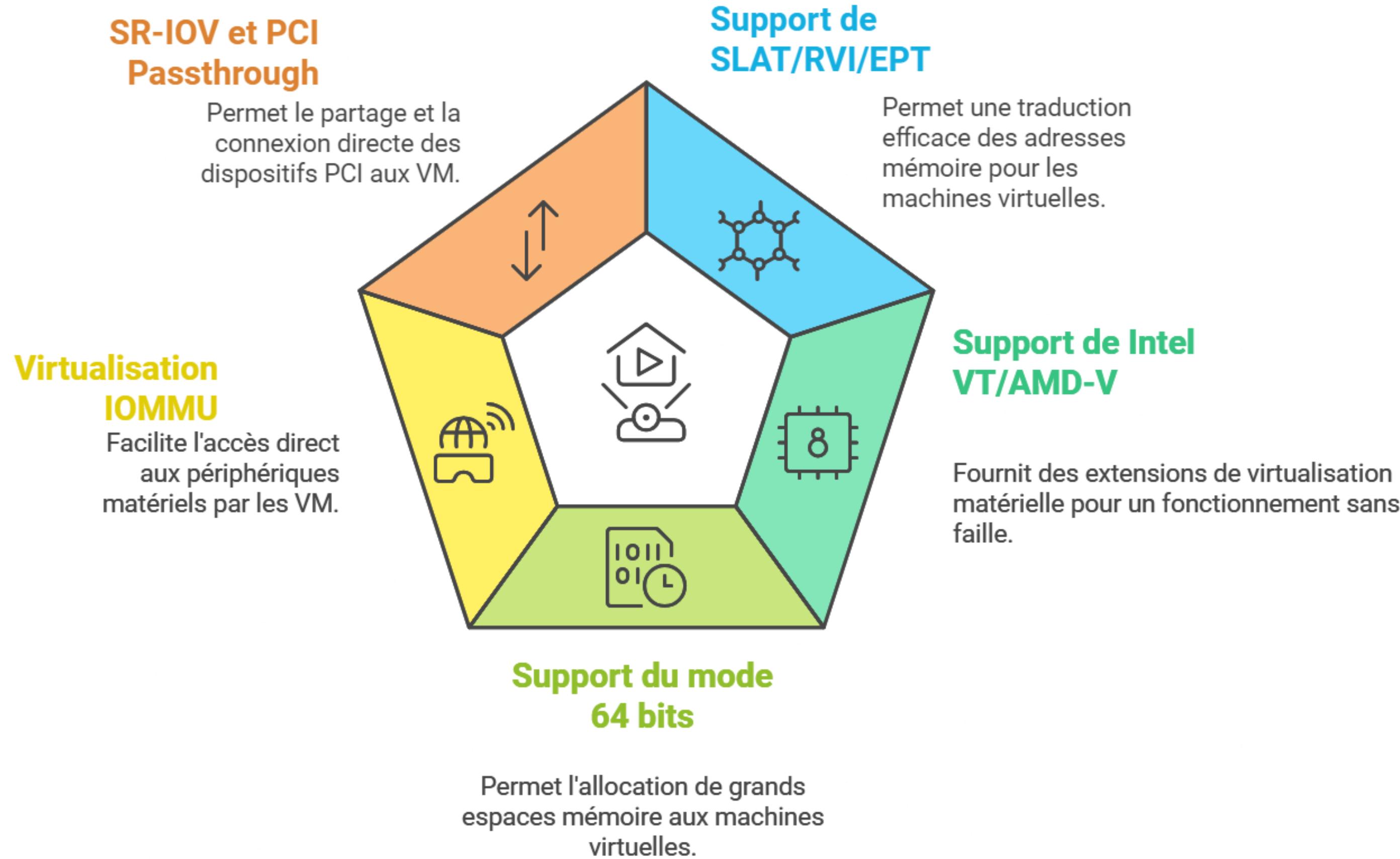
Compatibilité de la carte mère

Assure le support BIOS/UEFI pour l'activation de la virtualisation.

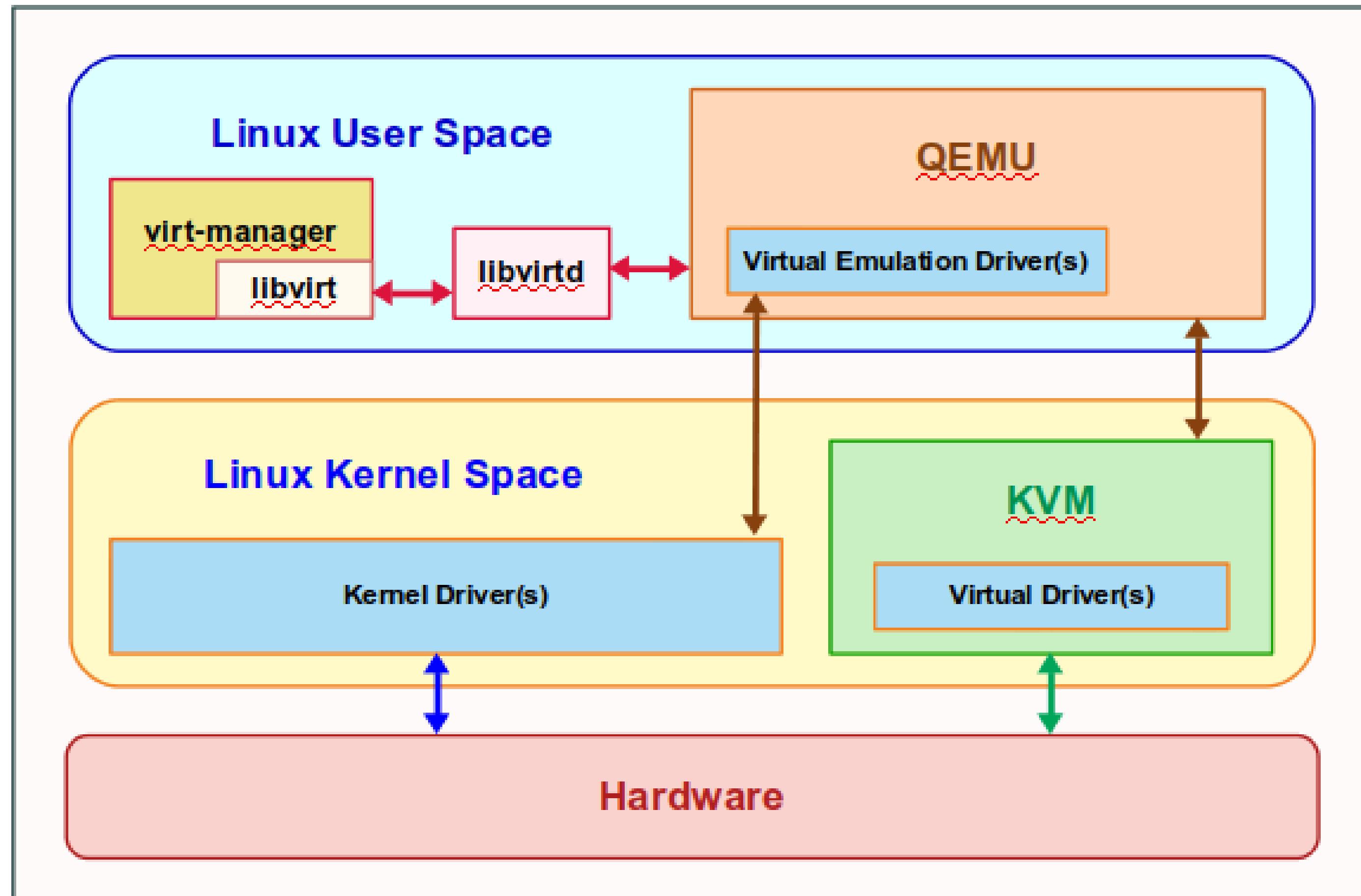


VIRTUALISATION - EXIGENCES

Exigences modernes pour les hyperviseurs



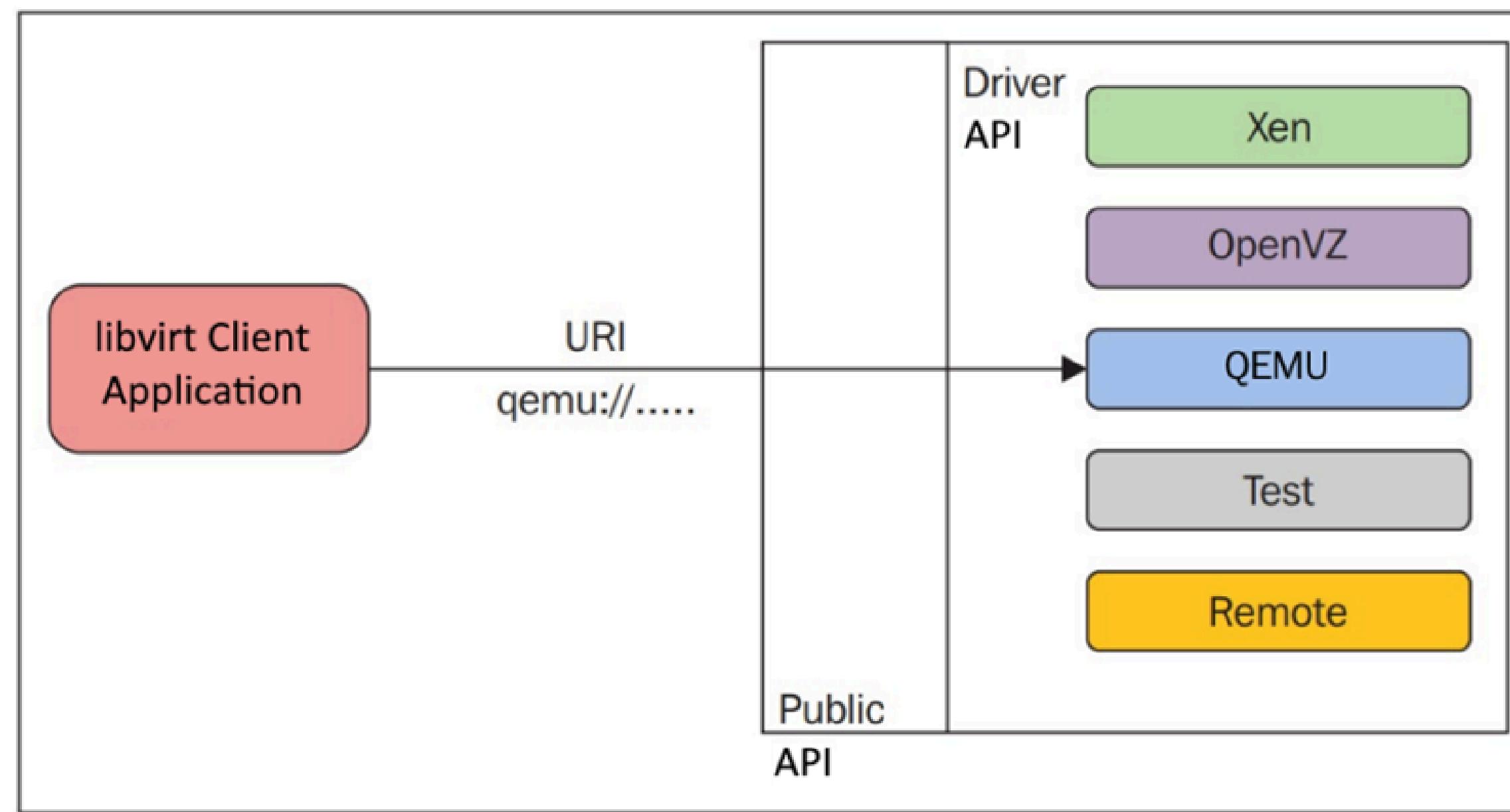
ARCHITECTURE KVM



libvirt est une API majeure pour la gestion des machines virtuelles, spécialement sous KVM. Elle agit aussi comme un démon et un outil de gestion pour divers hyperviseurs, permettant une gestion locale et distante des machines virtuelles via des outils comme virt-manager (interface graphique) et virsh (outil en ligne de commande). libvirt offre une couche de gestion stable pour la création, la surveillance, le contrôle, et la migration de machines virtuelles. Le démon libvirtd se charge de communiquer avec l'hyperviseur (QEMU/KVM dans ce cas) en fonction des requêtes faites par des clients comme virt-manager ou virsh.

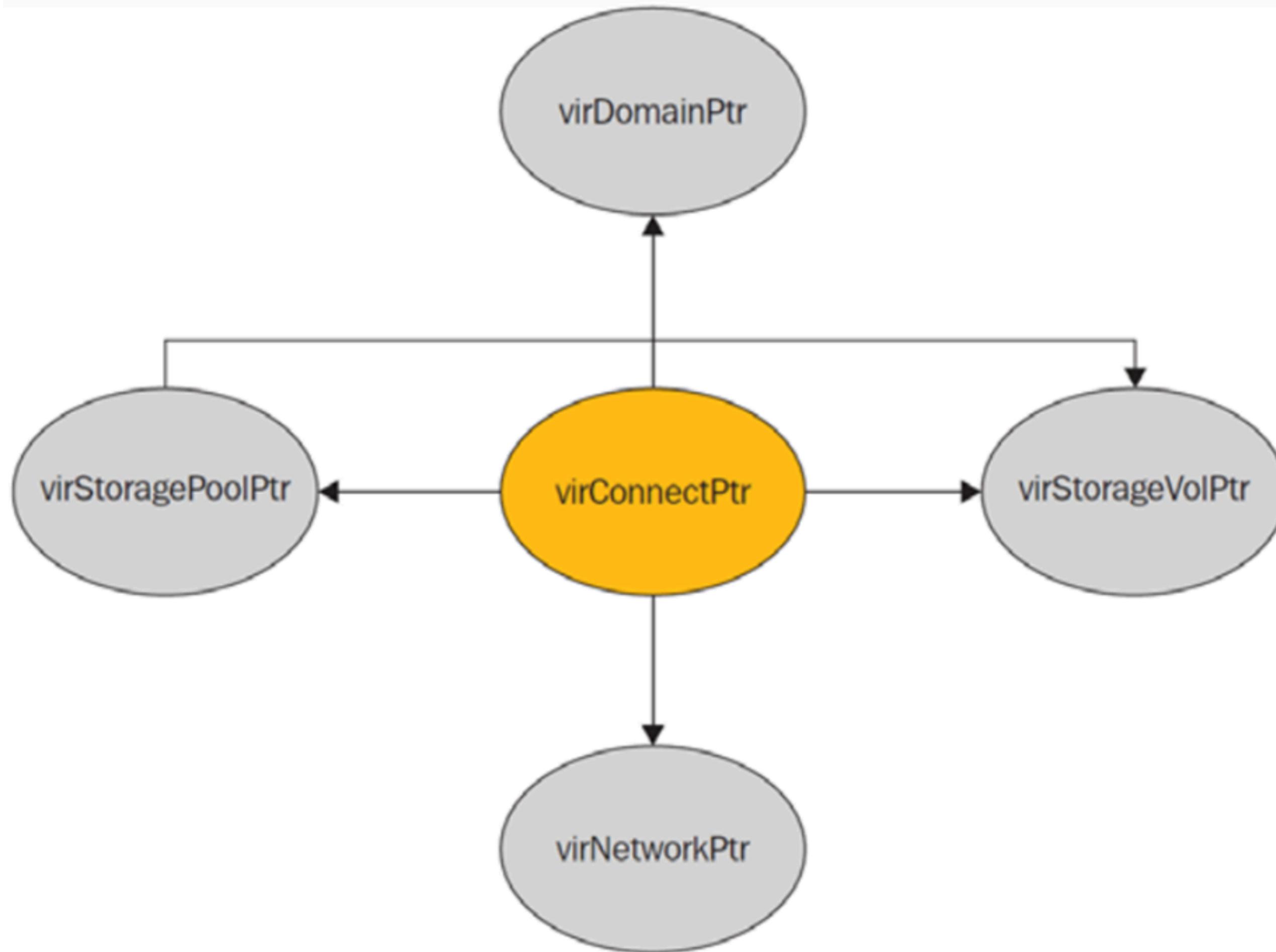


libvirt utilise une architecture basée sur **des pilotes**, facilitant la communication avec différents hyperviseurs externes. Cela inclut des pilotes pour des hyperviseurs comme LXC, Xen, QEMU, VirtualBox, Microsoft Hyper-V, bhyve (BSD), IBM PowerVM, et OpenVZ , comme ce diagramme montre :



Libvirt est largement utilisé pour gérer les opérations courantes liées aux machines virtuelles, comme leur création et leur gestion. Pour ces tâches, des pilotes secondaires sont employés pour des fonctions telles que la configuration des interfaces, la gestion des règles de pare-feu et du stockage. Selon *libvirt.org*, une application peut établir une connexion avec l'hyperviseur via un `virConnectPtr`, ce qui lui permet de gérer les domaines et les ressources de virtualisation associées, exposées comme des objets de premier plan. La figure suivante illustre les cinq principaux objets de l'API et leurs connexions:

LIBVIRT - OBJETS PRINCIPAUX



LIBVERT - OBJETS PRINCIPAUX

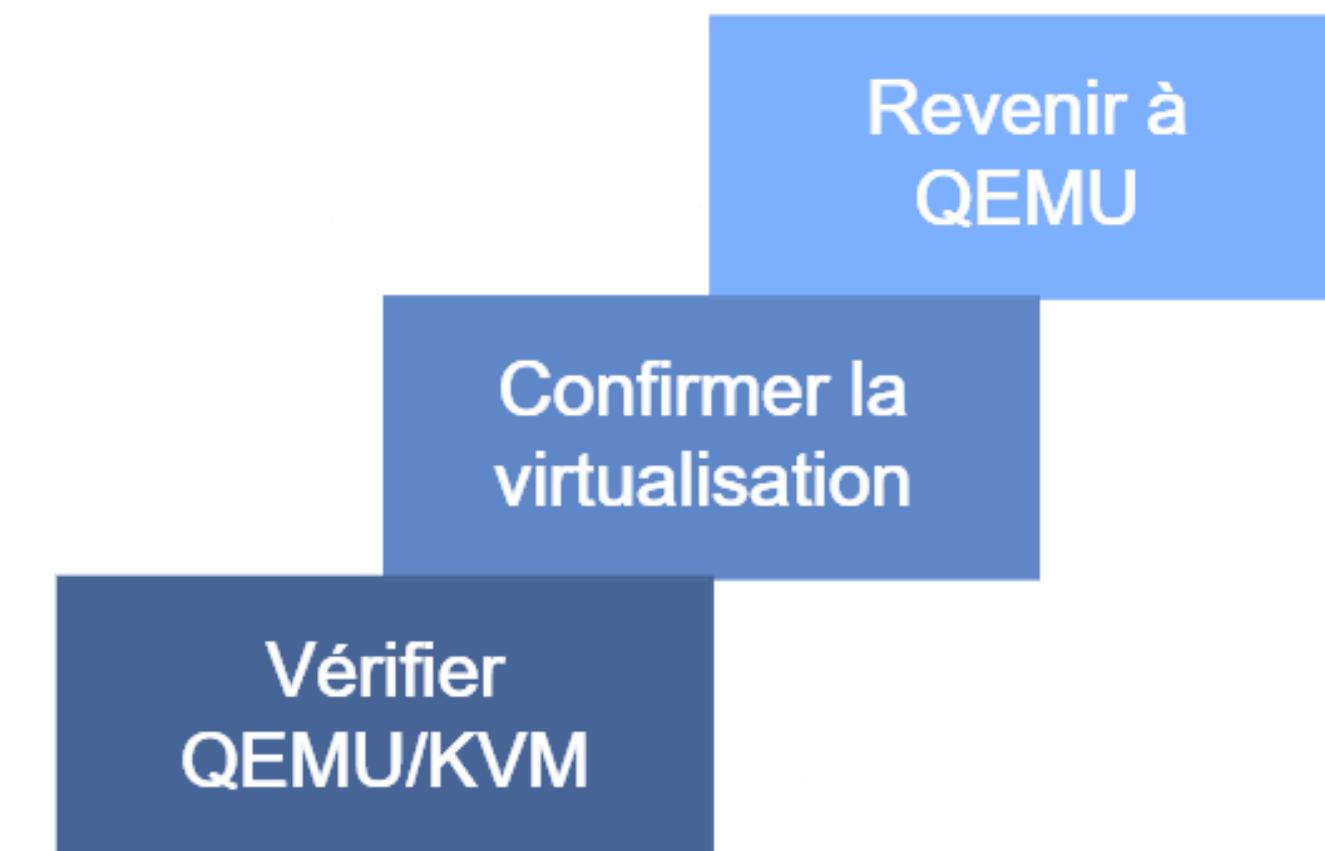
Donnons quelques détails sur les principaux objets disponibles dans le code de libvirt. La plupart des fonctions de libvirt utilisent ces objets pour leurs opérations :

- **virConnectPtr** : Comme nous l'avons mentionné précédemment, libvirt doit se connecter à un hyperviseur et agir. La connexion à l'hyperviseur est représentée par cet objet, qui est l'un des objets fondamentaux de l'API de libvirt.
- **virDomainPtr** : Les machines virtuelles ou systèmes invités sont généralement appelés domaines dans le code de libvirt. virDomainPtr représente un objet pour un domaine/machine virtuelle actif ou défini.
- **virStorageVolPtr** : Il existe différents volumes de stockage, exposés aux domaines/systèmes invités. virStorageVolPtr représente généralement l'un de ces volumes.
- **virStoragePoolPtr** : Les volumes de stockage exportés font partie de l'un des pools de stockage. Cet objet représente l'un des pools de stockage.
- **virNetworkPtr** : Dans libvirt, nous pouvons définir différents réseaux. Un seul réseau virtuel (statut actif/défini) est représenté par l'objet virNetworkPtr.

LIBVERT - GESTION DES HYPERVISEURS

Dans le cas de QEMU/KVM, libvirt suit 3 étapes (selon le livre *Mastering KVM Virtualisation*) pour vérifier la présence et la compatibilité de QEMU et KVM :

- Vérification de QEMU/KVM** : libvirt vérifie si le binaire *qemu-kvm* et le nœud de périphérique */dev/kvm* sont présents pour s'assurer de la compatibilité avec KVM.
- Confirmation de la Virtualisation** : La présence du binaire et du nœud de périphérique confirme que KVM peut être utilisé pour la virtualisation.
- Fallback vers QEMU** : Si KVM n'est pas disponible, libvirt vérifie si QEMU seul peut être utilisé en cherchant des binaires QEMU sans KVM.



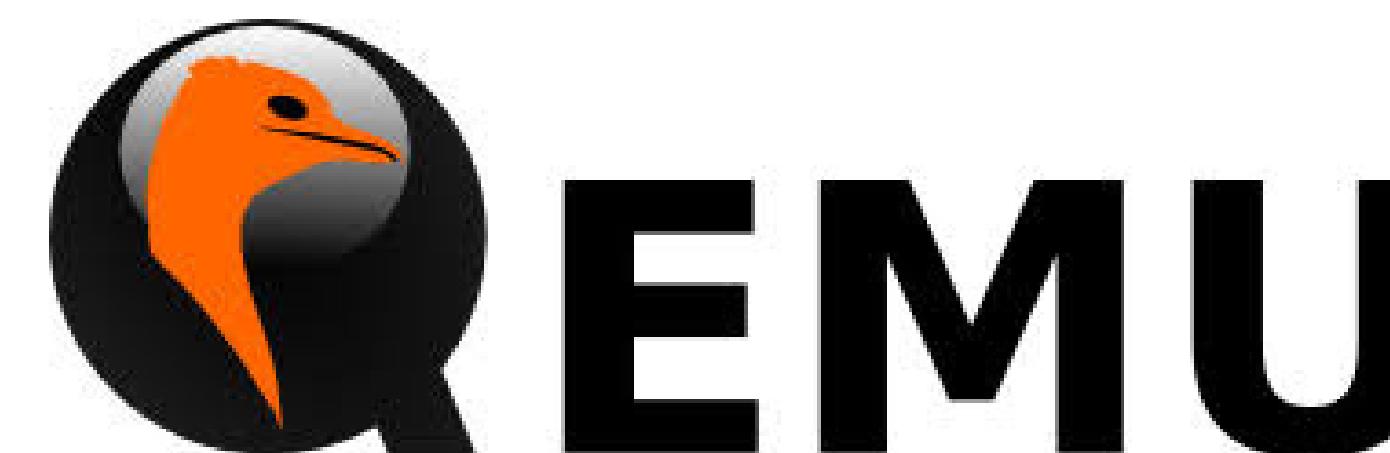
LIBVERT - FONCTIONNEMENT

libvirt gère plusieurs processus pour la virtualisation efficace avec QEMU/KVM :

- **Recherche de Binaries** : Le pilote QEMU dans libvirt cherche des exécutables pour QEMU-KVM en fonction de la distribution Linux.
- **Création de Processus** : Après la formation des arguments, libvirt utilise exec() pour lancer un processus QEMU-KVM.
- **Interaction avec KVM** : libvirt déclenche QEMU-KVM, mais c'est QEMU qui interagit avec les modules du noyau via /dev/kvm.
- **Création de Machines Virtuelles** : Chaque machine virtuelle lancée est un processus QEMU-KVM distinct, géré par libvirtd.
- **Configuration via XML** : Les propriétés des machines virtuelles sont définies dans des fichiers XML situés dans /etc/libvirt/qemu.

QEMU - DÉFINITION

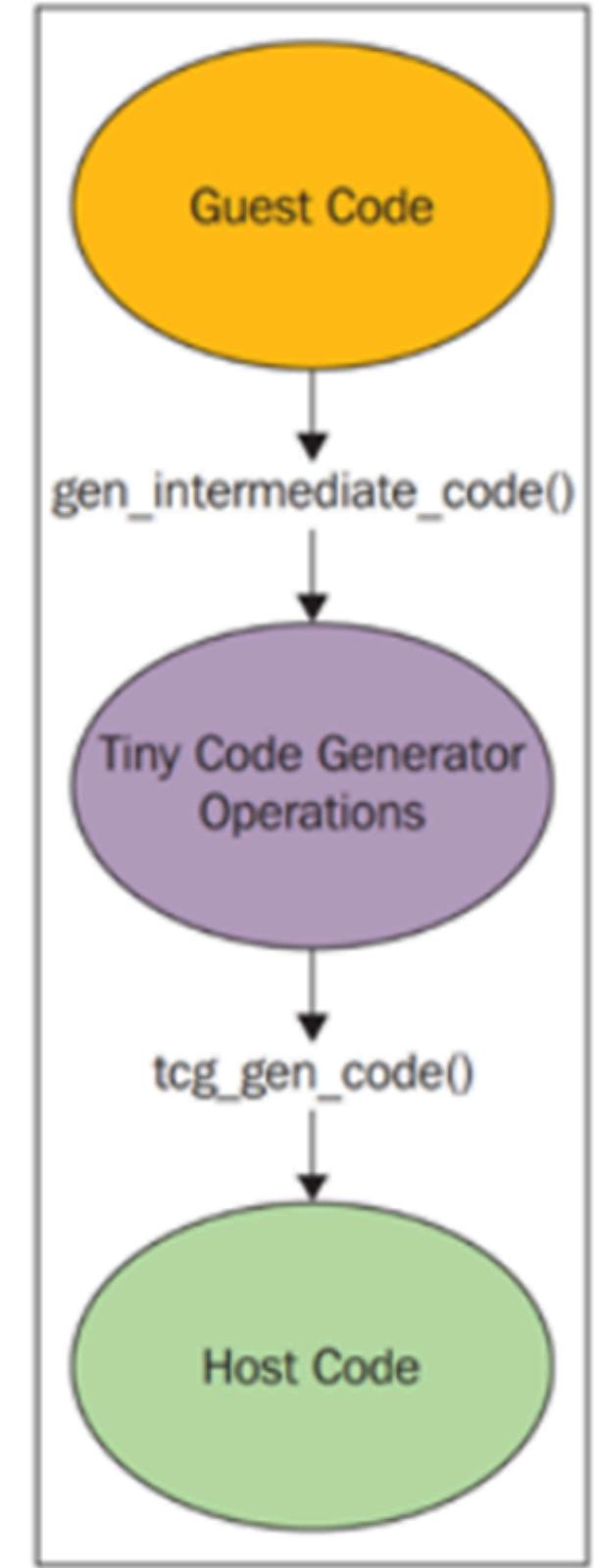
QEMU est un émulateur de machine et un virtualiseur open source, développé par *Fabrice Bellard*. Il permet d'exécuter des systèmes d'exploitation et des programmes conçus pour une architecture matérielle spécifique sur une autre architecture. Par exemple, il est possible d'exécuter un système d'exploitation conçu pour une carte ARM sur un PC x86. Cette capacité d'émulation est particulièrement utile pour les développeurs qui souhaitent tester des applications sur différentes plateformes sans avoir besoin de matériel physique pour chaque type de machine.



QEMU - MÉCANISMES

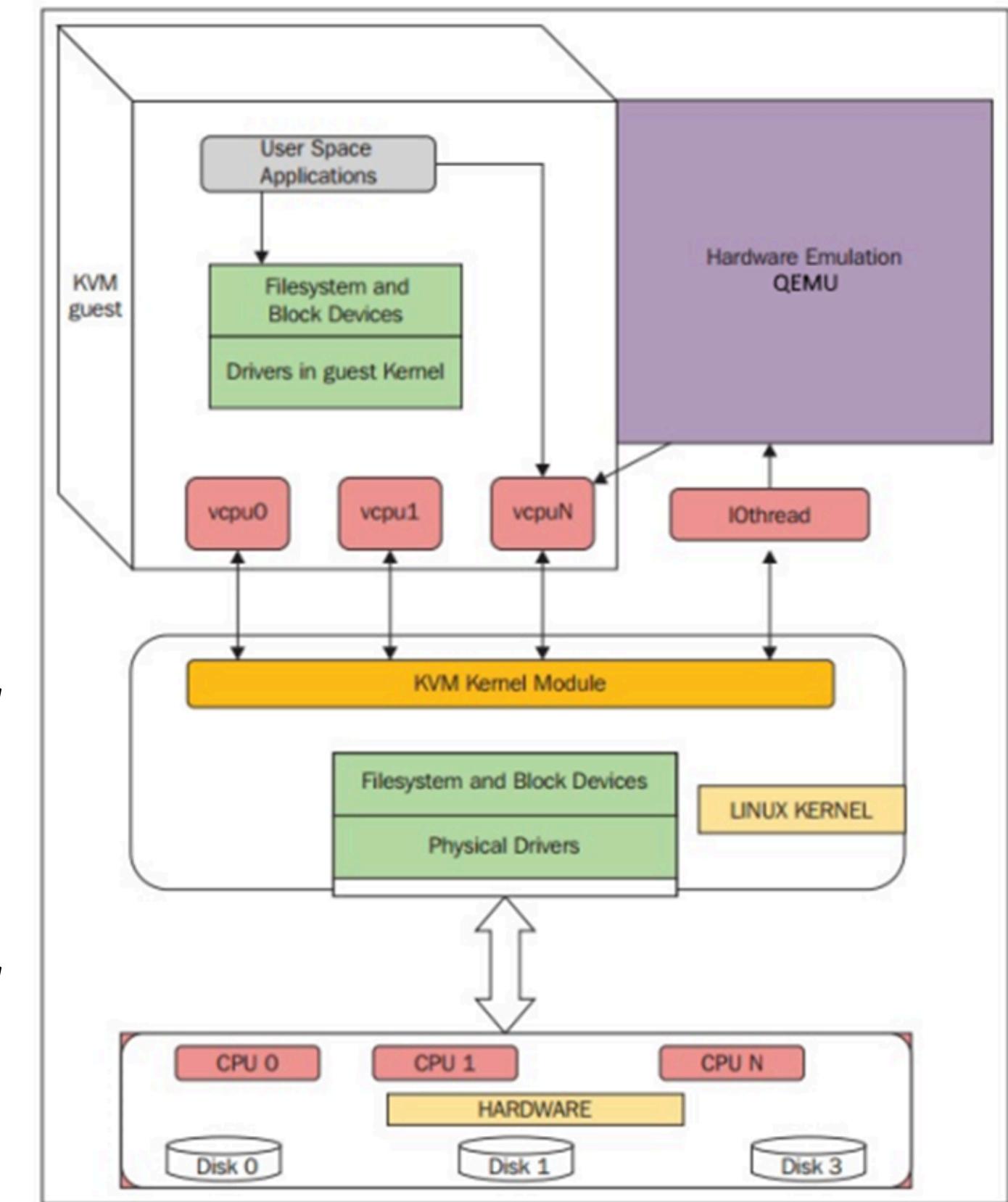
L'un des principaux mécanismes utilisés par QEMU pour atteindre cette émulation est la traduction dynamique. Cela signifie que QEMU traduit les instructions du code binaire d'une architecture cible en instructions compréhensibles par l'architecture de l'hôte au moment de l'exécution. Ce processus est géré par un composant appelé **Tiny Code Generator (TCG)**, qui agit comme un compilateur Just-In-Time (JIT). Le TCG permet à QEMU d'exécuter du code invité en temps réel, en optimisant les performances tout en maintenant la compatibilité entre différentes architectures.

En outre, QEMU peut fonctionner en tant qu'hyperviseur hébergé, permettant à plusieurs systèmes d'exploitation de s'exécuter simultanément sur le même matériel. Cela est essentiel pour les environnements de virtualisation où l'isolation et la gestion des ressources sont cruciales. QEMU est capable de gérer les ressources matérielles de manière efficace, offrant ainsi une plateforme robuste pour le développement et les tests.



QEMU - MÉCANISMES

QEMU, en mode **virtualiseur**, exécute directement le code des systèmes d'exploitation invités sur le CPU de l'hôte, atteignant des performances proches du natif. Associé à KVM, il virtualise divers systèmes d'exploitation, notamment sur des architectures comme PowerPC, S390 et x86. En plus du CPU, QEMU émule des périphériques essentiels (disques, réseau, VGA, PCI, USB) et utilise des threads POSIX pour gérer chaque vCPU. Cette combinaison de virtualisation et d'émulation permet une solution flexible et performante pour déployer et gérer différents systèmes d'exploitation sur des architectures matérielles variées.



QEMU utilise des threads POSIX pour exécuter le code invité sur le CPU physique, ce qui permet d'exécuter les vCPUs invités en tant que processus dans le noyau hôte, offrant ainsi plusieurs avantages. Il joue un rôle clé en tant que composant utilisateur de l'hyperviseur KVM, gérant l'exécution du code invité et les tâches telles que l'émulation I/O, la configuration des périphériques et la migration en direct. En interagissant avec le fichier de périphérique (`/dev/kvm`) via des appels `ioctl()`, QEMU permet à KVM de devenir un hyperviseur complet. KVM, en tant qu'accélérateur des extensions de virtualisation matérielle, nécessite que les systèmes virtuels utilisent la même architecture pour bénéficier des performances supérieures par rapport à des méthodes comme la traduction binaire.

1. Clonage du Dépôt QEMU :

Avant de commencer l'examen des détails internes de QEMU, commencez par cloner le dépôt Git de QEMU.

```
# git clone git://git.qemu-project.org/qemu.git
```

2. Exploration de la Hiérarchie de Fichiers :

Une fois le clonage terminé, explorons la hiérarchie de fichiers à l'intérieur du dépôt pour mieux comprendre l'architecture et les composants principaux.

```
[root@kvmsource qemu]# ls
accel           dump          meson_options.txt      qga
audio           exec.c        migration            qobject
authz           exec-vary.c   module-common.c    qom
backends        fpu          monitor             README.rst
block           fsdev         nbd                 replay
block.c         gdbstub.c    net                 replication.c
blockdev.c      gdb-xml       os-posix.c        replication.h
blockdev-nbd.c gitdm.config os-win32.c        roms
blockjob.c      hmp-commands.hx pc-bios            rules.mak
bootdevice.c   hmp-commands-info.hx plugins           scripts
bsd-user        hw           po                  scsi
capstone        include       python             slirp
Changelog       io           qapi               softmmu
chardev         iothread.c   qdev-monitor.c   storage-daemon
CODING_STYLE.rst job.c       qemu-bridge-helper.c stubs
configure       job-qmp.c    qemu-edid.c     target
contrib         Kconfig      qemu-img.c      tcg
COPYING         Kconfig.host qemu-img-cmds.hx tests
COPYING.LIB      libdecnumber LICENSE           thunk.c
cpus-common.c   libdecreader  linux-headers   tools
crypto          libqemu       linux-user      tpm.c
default-configs MAINTAINERS  MAINTAINERS      trace
device_tree.c   Makefile     qemu-nbd.c     trace-events
disas           Makefile.objs  qemu-nsi        ui
disas.c         memory_ldst.c.inc  qemu-options.h util
dma-helpers.c   meson        qemu-options.hx  VERSION
docs            meson        qemu-options-wrapper.h version.rc
dtc             meson.build   qemu-sasl        version.texi.in |
```

QEMU - INTERNES DE KVM

3. Structures de Données Principales :

Dans QEMU, plusieurs structures de données et appels de fonction ioctl() assurent la communication entre l'espace utilisateur de QEMU et l'espace noyau de KVM :

Structures Importantes :

KVMState : contient des descripteurs de fichiers clés pour les machines virtuelles.

int fd; : descripteur de fichier

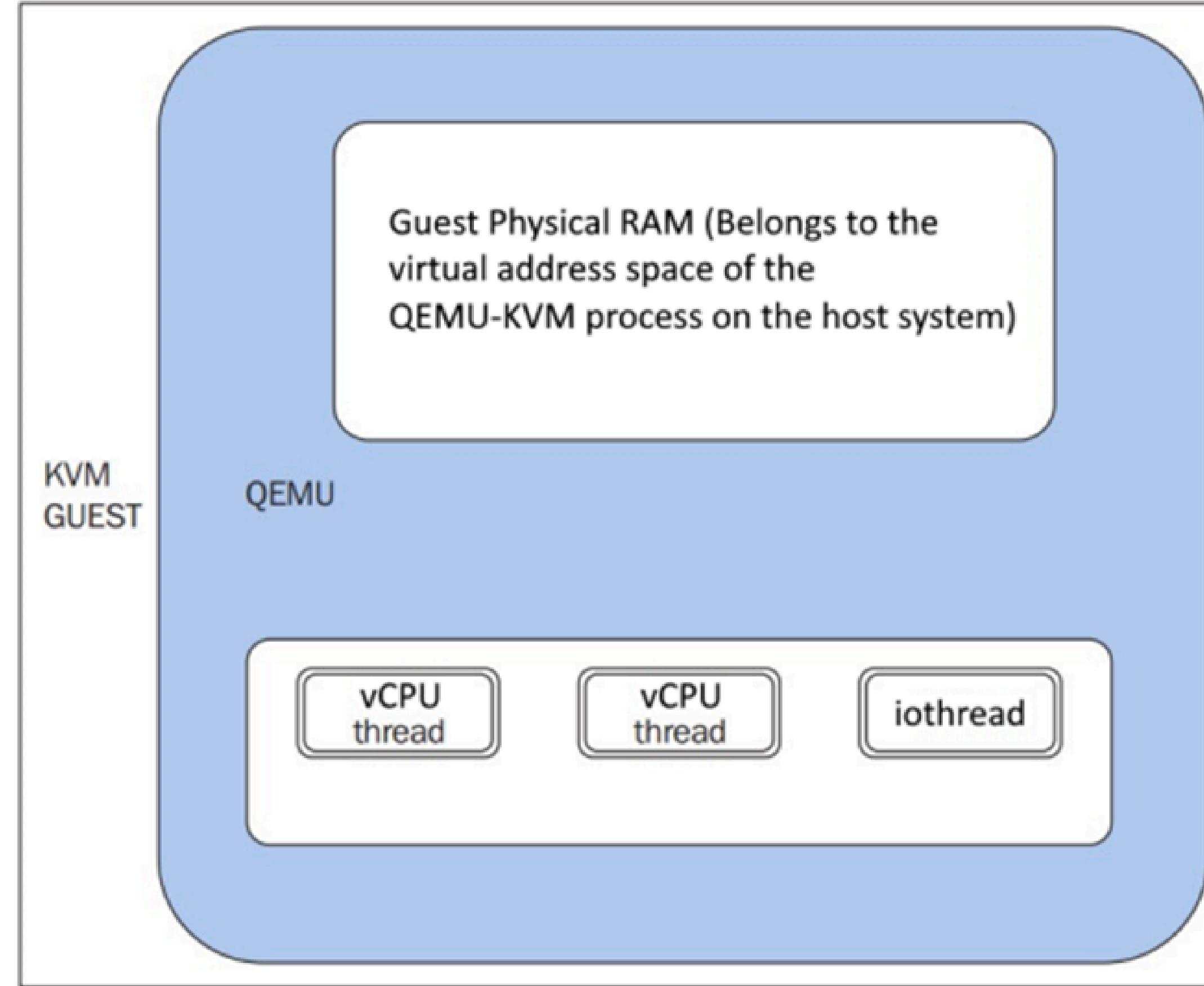
int vmfd; : descripteur de machine virtuelle

*struct kvm_coalesced_mmio_ring *coalesced_mmio_ring;*

CPU{X86}State : chaque structure de ce type stocke le contenu des registres d'un vCPU.

MachineState et CPUState : contiennent des informations essentielles, comme le nombre de cœurs et de threads.

QEMU - INTERNES DE KVM



QEMU - INTERNES DE KVM

4. Appels de Fonction ioctl() :

QEMU utilise plusieurs appels ioctl() pour gérer les machines virtuelles et les vCPUs :

kvm_ioctl(), kvm_vm_ioctl(), kvm_vcpu_ioctl() : interagissent avec le système KVM pour gérer les vCPUs et l'infrastructure de la VM.

kvm_init() : ouvre le fichier du périphérique KVM, initialise les descripteurs fd et vmfd, et crée des vCPUs.

5. Crédit et Exécution des vCPUs :

Les vCPUs, créés sous forme de threads POSIX, exécutent le code de l'invité via KVM_RUN. La fonction kvm_cpu_exec() définit les actions en cas de retour à l'espace utilisateur de QEMU.

6. Utilisation des Extensions Matérielles :

KVM utilise les extensions matérielles d'Intel et d'AMD pour exécuter directement le code de l'invité sur le CPU hôte. Cependant, lors de l'accès à des registres de périphériques, KVM retourne à QEMU pour émuler les opérations nécessaires.

KVM - DÉFINITION

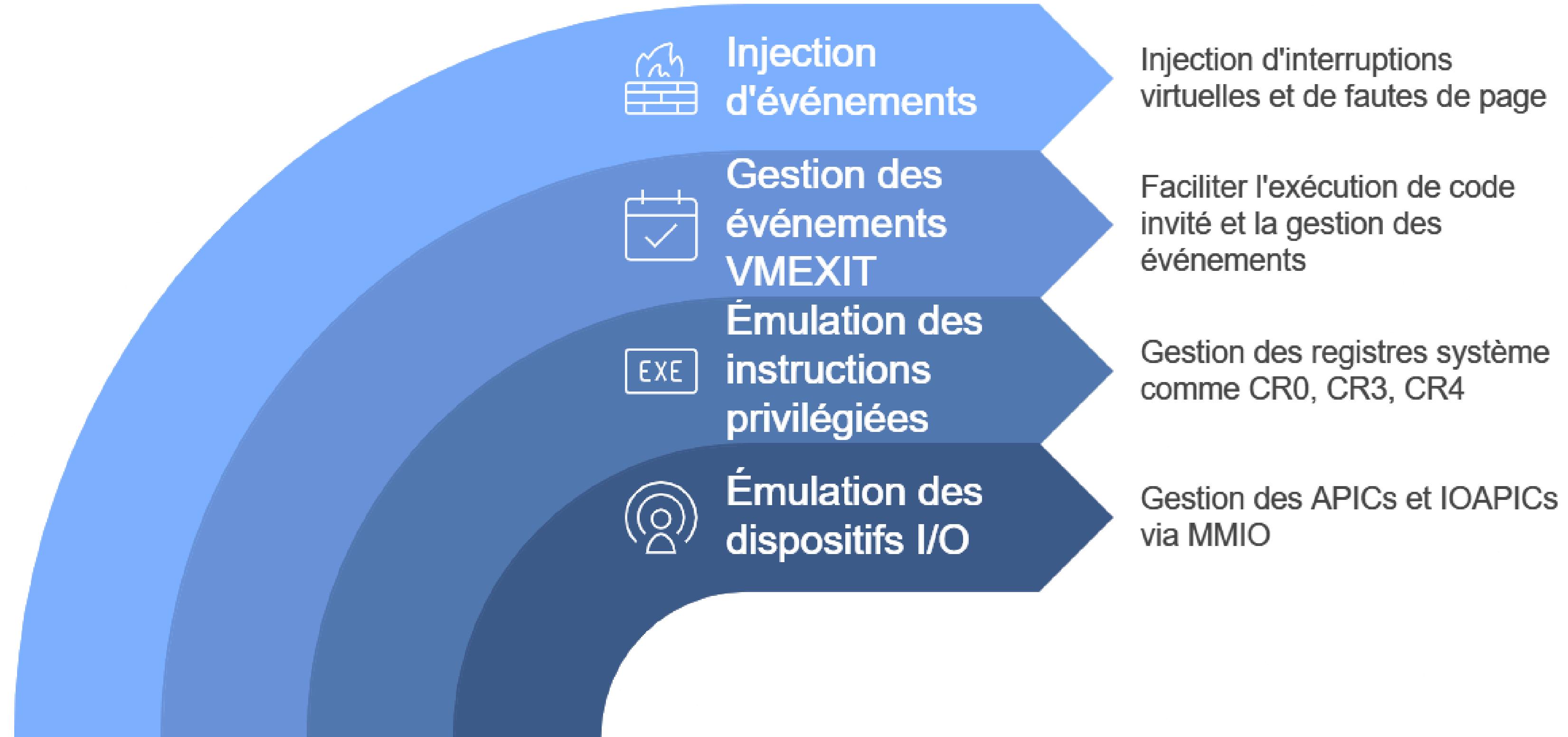
KVM (Kernel-based Virtual Machine) est un hyperviseur intégré au noyau Linux, qui permet à un programme comme QEMU d'exécuter du code invité directement sur le processeur hôte. KVM fonctionne en transformant le noyau Linux standard en hyperviseur, et il nécessite un processeur compatible avec la virtualisation matérielle pour fonctionner efficacement. Ce processus repose sur l'utilisation de modules noyau spécifiques, tels que [kvm.ko](#), [kvm-intel.ko](#) pour les processeurs Intel, et [kvm-amd.ko](#) pour les processeurs AMD. En fonction de la présence de certains drapeaux matériels comme vmx ou svm, KVM charge le module approprié pour permettre la virtualisation.



KVM - FONCTIONNEMENT

KVM expose un fichier de périphérique `/dev/kvm` aux applications, permettant ainsi l'utilisation des appels système via `ioctl()`. Ce fichier sert de point de communication entre QEMU et KVM, permettant à QEMU de créer, initialiser et gérer les machines virtuelles (VM). L'espace d'adressage physique des VM est mappé dans l'espace utilisateur de QEMU/KVM, ce qui permet d'exécuter les machines virtuelles de manière isolée tout en exploitant la puissance du processeur hôte.

KVM - CAPACITÉS



KVM - INTERACTION AVEC QEMU

KVM et QEMU travaillent ensemble pour créer une solution de virtualisation complète. KVM gère la virtualisation matérielle, tandis que QEMU s'occupe de l'émulation des périphériques et des fonctions non couvertes par KVM. QEMU permet également une émulation CPU au niveau des instructions, ce qui lui permet de faire fonctionner des machines virtuelles avec des architectures CPU différentes de celles de l'hôte.

QEMU peut également être utilisé en tant qu'hyperviseur autonome, émettant des événements d'horloge, des interruptions, et gérant les appels système des invités. QEMU permet aussi d'accéder à des périphériques virtuels et de gérer des périphériques d'entrée/sortie, rendant ainsi l'environnement virtuel beaucoup plus souple et adaptable.

KVM - VIRTUALISATION DE CPU

La virtualisation du CPU avec KVM repose sur des mécanismes de virtualisation matérielle, utilisant des structures comme la Structure de Contrôle de Machine Virtuelle (VMCS) et le Bloc de Contrôle de Machine Virtuelle (VMCB), qui gèrent les transitions entre le mode hôte et le mode invité. Ces structures facilitent la gestion des interruptions, des exceptions et de la mémoire virtuelle.

Les processeurs modernes, comme ceux d'Intel avec Intel VT-X et AMD avec AMD-V (SVM), fournissent des instructions supplémentaires pour la gestion de la virtualisation, telles que VMXON, VMXOFF, VMLAUNCH, VMRESUME, et VMEXIT. Ces instructions permettent la transition entre les modes d'opération de l'hôte et de l'invité, en gérant l'état de la machine virtuelle pendant la transition.

KVM - LES APIs

Les APIs de KVM reposent principalement sur les appels `ioctl()`, qui permettent de gérer divers aspects des machines virtuelles. Les appels `ioctl()` sont divisés en trois catégories :

- **System ioctls** : Ils permettent de gérer les attributs globaux de KVM et sont utilisés pour créer des machines virtuelles.
- **Device ioctls** : Utilisés pour le contrôle des périphériques au sein des machines virtuelles.
- **Autres types d'ioctls** : Concernent des opérations plus spécifiques pour gérer la virtualisation.

Ces appels permettent aux applications de configurer et d'interagir avec les machines virtuelles de manière flexible et contrôlée.

KVM - VIRTUALISATION DE CPU

Exemples d'appels ioctl() KVM :

- Pour les descripteurs /dev/kvm :

```
c

#define KVM_GET_API_VERSION _IO(KVMIO, 0x00)
#define KVM_CREATE_VM _IO(KVMIO, 0x01) /* renvoie un descripteur VM */
```

- Pour les descripteurs de VM :

```
c

#define KVM_SET_MEMORY_REGION _IOW(KVMIO, 0x40, struct kvm_memory_regi
#define KVM_CREATE_VCPU _IO(KVMIO, 0x41)
```

- Pour les descripteurs de vCPU :

```
c

#define KVM_RUN _IO(KVMIO, 0x80)
#define KVM_GET_REGS _IOR(KVMIO, 0x81, struct kvm_regs)
#define KVM_SET_REGS _IOW(KVMIO, 0x82, struct kvm_regs)
```

CONCLUSION

KVM (Kernel-based Virtual Machine) est une solution de virtualisation complète et open-source intégrée au noyau Linux, qui transforme un serveur Linux en hyperviseur capable d'exécuter plusieurs machines virtuelles (VM). En utilisant les extensions de virtualisation matérielle comme Intel VT-x et AMD-V, KVM offre des performances proches du matériel natif, ce qui en fait un choix privilégié pour les environnements de virtualisation haute performance.

RÉFÉRENCES

- Mastering KVM Virtualization par *Prasad Mukhedkar et Suresh K. Shah*
- <http://www.linux-kvm.org>
- <https://www.libvirt.org>
- <https://www.qemu.org>
- <https://www.packtpub.com/product/masteringkvmvirtualization>
- <https://wiki.debian.org/KVM>
- https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/8/html/using_virtualization/index
- <https://ubuntu.com/server/docs/virtualization>
- https://fedoraproject.org/wiki/Getting_started_with_virtualization