Virtualisation du matériel

Principes de fonctionnement et implémentation par Qemu/KVM

Comment partager un système ?

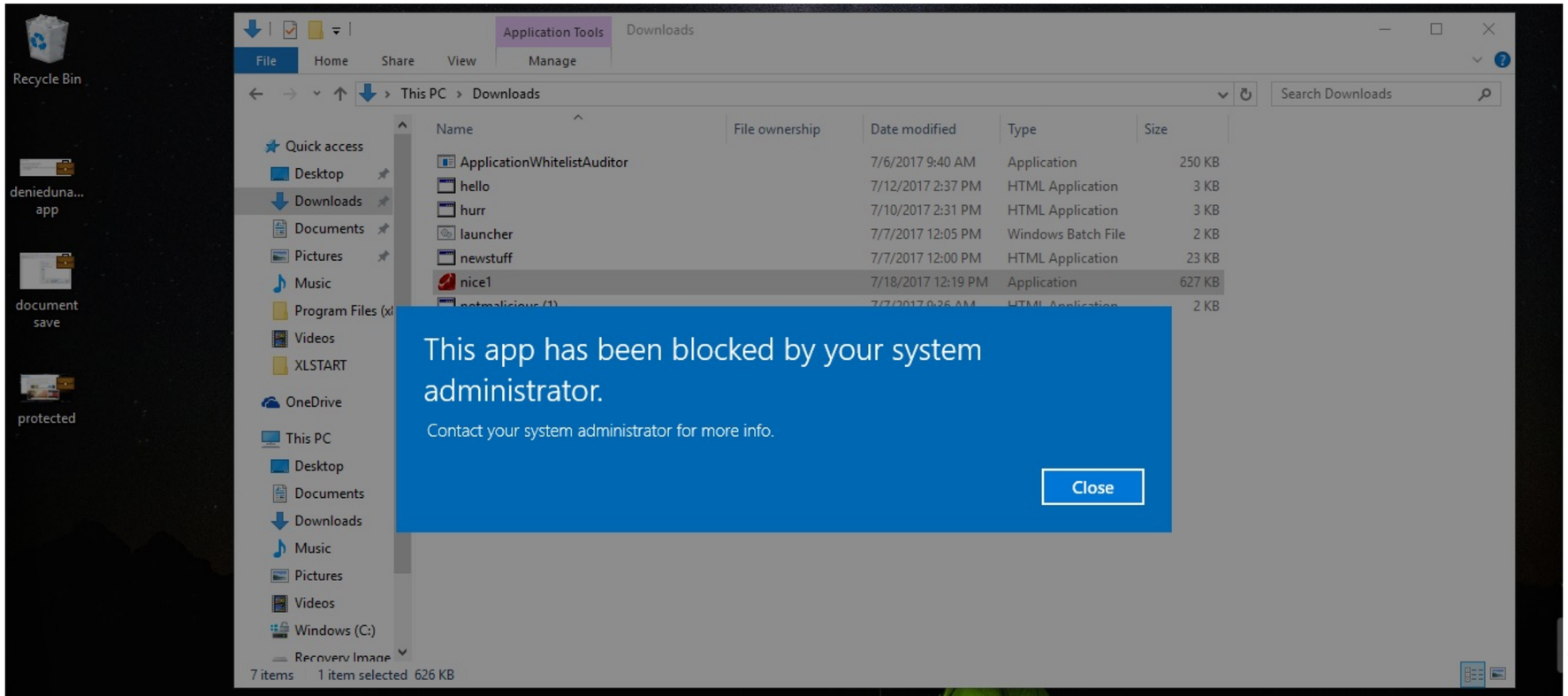
Le besoin

- Éviter d'avoir à dédier chaque serveur à une unique application
  - Gâchis de ressources
  - Manque de flexibilité
- Une machine doit pouvoir être partagée
  - Par plusieurs applications
  - Par plusieurs utilisateurs
  - Par plusieurs organisations
- Partager les ressources du système
  - Sans contraintes ou pré-requis pour les applications
  - En assurant la confidentialité des données

Comment partager un système ?

Problématique

- Interfaces classiques des systèmes d'exploitation
  - Solutions au cas par cas nécessitant
    - Un administrateur système définissant une politique d'utilisation
    - Des applications capables de prendre en compte cette politique



Comment partager un système ?

Identifiants de fichiers

- Toutes les applications voient les mêmes fichiers
  - /etc/monapp.cfg
  - /var/log/monapp.log
  - /var/run/monapp/…
- Chaque instance d'une application a besoin de ses propres données
  - => Définition d'utilisateurs, chemins et droits d'accès par l'administrateur
  - => Configuration spécifique de chaque instance application pour prendre en compte l'arborescence définie
  - => Peut nécessiter une recompilation
- Le système ne fournit pas forcément les dépendances nécessaires
  - Versions de bibliothèques et supports exécutifs
    - => Installation manuelle des dépendances dans des répertoires dédiés
    - => Sans profiter du gestionnaire de paquets
    - => Relocalisation pas toujours aisée

Comment partager un système ?

Identifiants réseaux

- Toutes les applications voient les mêmes identifiants réseaux
  - Interfaces, adresses, ports
- Comment attribuer un port réseau à chaque application ?
  - Ports standardisés pour chaque protocole applicatif (ex: HTTP=80)
    - => Configuration spécifique de l'application pour chaque serveur
- Comment filtrer les accès aux différents ports ?
  - => Définition de règles de filtrage par l'administrateur (ex: iptables)

Comment partager un système ?

Ressources matérielles

- Les ressources matérielles sont pilotées par un unique OS
- Chaque application a accès à l'ensemble des ressources
    - => Politques de partage des ressources (resource limits, cgroups)

Processus

- Chaque application voit l'ensemble de l'activité du système
    - => Politiques de sécurité avancées (ex: selinux)

Comment partager un système ?

Gestion des privilèges

- Un seul utilisateur root
- Politiques définies de façon centralisée par l'administrateur
  - Doit maitriser l'ensemble des usages de la machine
  - Trouver une solution à chaque problématique
  - Garantir la sécurité des utilisations
- Travail très complexe
- Ralentit le déploiement des machines

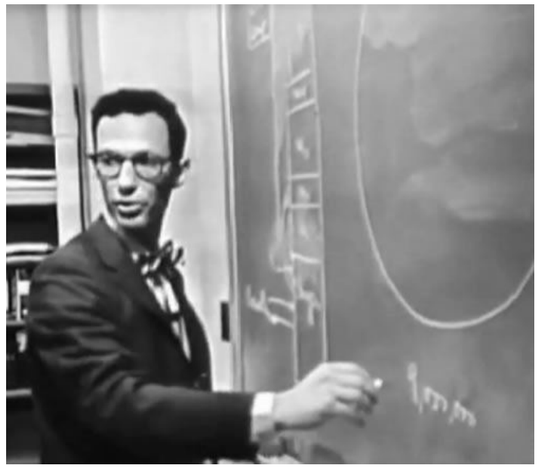
Comment partager un système ?

Une nécessité apparue dès les premiers ordinateurs

- Machines très coûteuse partagées par une ou plusieurs institutions
  - Nombreux utilisateurs
- Premiers systèmes d'exploitation sont mono-tâches
  - Exécution un par un des programmes
  - Impossibilité de travailler de façon interactive
    - Monopoliserait l'ensemble du système pour un utilisateur

Comment partager un système ?

Vers des systèmes multi-utilisateurs à temps partagé



- CTSS: Compatible Time Sharing System (1961)
  - Premier prototype développé au MIT
  - Processeur spécialement modifié par IBM
  - Démontre l'intérêt de pouvoir partager système
    - Consoles interactives
- Nombreux projets visant à développer un OS complet
  - Mettra longtemps à aboutir du fait de la difficulté
    - TSS est abandonné
    - Multics subira de nombreux retards
      - Pose les base pour un système plus simple: Unix

> "…overdesigned and overbuilt and over everything. It was close to unusable" K. Thompson

Comment partager un système ?

La virtualisation: une solution globale au partage de système

- CP/CMS (1968)
  - Un des premiers OS multi-utilisateur à temps partagé fonctionnel
  - Décomposition en deux sous-problèmes plus simples
  - CP: Control Program
    - Multiplexe la machine en plusieurs machines indépendants
    - **L'interface proposée à chaque utilisateur de la machine est l'interface de la machine complète**
    - Équivalent d'une machine virtuelle (appelé pseudo-machine)
  - CMS: Cambdrige monitor system
    - Système d'exploitation mono-utilisateur simple
    - Pilote les systèmes virtuels gérés par CP

Définitions

Terminologie

- **Virtualisation**: Permet de créer des versions virtuelles d'un matériel réel
- **Machine virtuelle (VM)** : Version virtuelle de l'ensemble composants d'une machine physique
  - Processeur, mémoire, périphériques …
- **Hyperviseur**: Logiciel en charge du fonctionnement des machines virtuelles
  - Arbitre l'accès aux ressources physique
  - Aussi appelé **VMM** (Virtual Machine Monitor)
- **Hôte**: Machine physique hébergeant des machines virtuelles
- **Invité**: Machine virtuelle hébergée sur une machine physique

Définitions

Virtualisation, émulation et simulation

- Émulation: reproduire le comportement d'un système
  - Système émulé indépendant du système physique
    - Ex: console de jeu sur un PC x86
  - Compromis performance/fidélité possibles
    - Reproduire le résultat, pas le mode de fonctionnement
    - Différence avec la simulation
- Virtualisation: multiplexage et/ou partitionnement d'un système
  - Système virtuel proche du système physique
    - Jeu d'instruction, type de périphériques …
  - Multiplexer et/ou partitionner sans perdre de performance
  - Tous les composants ne peuvent être virtualisés efficacement
    - Une partie des composants d'une VM sont émulés

Virtualisation

Définition selon Popek et Golberg

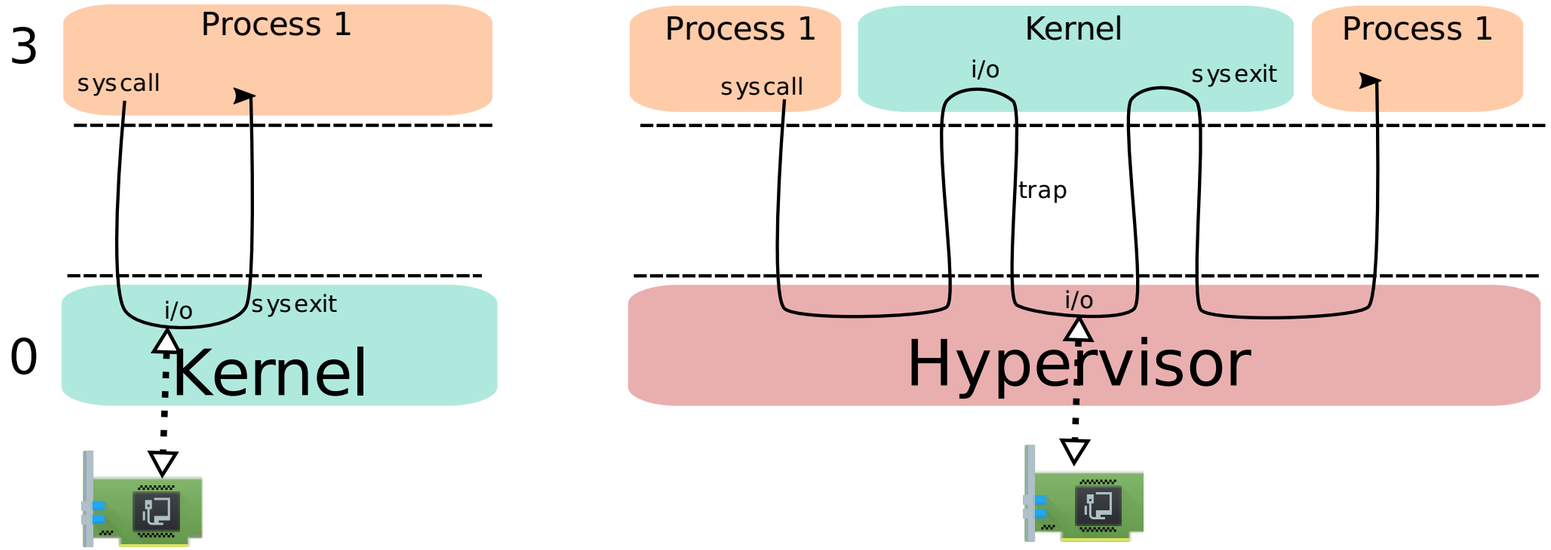
- Propriétés caractérisant un système de virtualisation
  - **Efficacité**
    - La majorité des instructions soumises dans un invité sont exécutés directement sur l'hôte
  - **Contrôle des ressources**
    - L'hyperviseur reste maître de toutes les ressources physiques
  - **Équivalence**
    - Un programme exécuté dans un invité se comporte comme sur l'hôte

Virtualisation

Défiintion de Popek et Golberg

- Méthode de déprivilégisation (*trap and emulate*)
  - Exécution directe de la quasi-totalité des instructions du processeur virtuel
  - Exécution dans le mode le moins privilégié du processeur physique
  - Interruption logicielle (*trap*) cas d'utilisation d'instruction privilégiée
  - L'hyperviseur reprend la main
  - Émulation du comportement attendu de l'instruction fautive

rings



Virtualisation

Critères de Popek et Golberg

Critères indiquant si une architecture est virtualisable (permet de construire un hyperviseur basé sur la déprivilégisation)

1. Les instructions non-privilégiées doivent avoir la même forme quel que soit le niveau de privilège courant

2. L'accès à la mémoire doit pouvoir être restreint en mode non privilégié

3. Toute instruction *sensible* doit être privilégiée (déclencher une interruption en mode non privilégié)

    - a) accès au niveau de privilège courant
    - b) accès aux registres sensibles (ex: vecteurs d'interruption)
    - c) accès aux mécanismes d'isolation mémoire
    - d) entrées/sorties

Virtualisation du jeu d'instruction x86

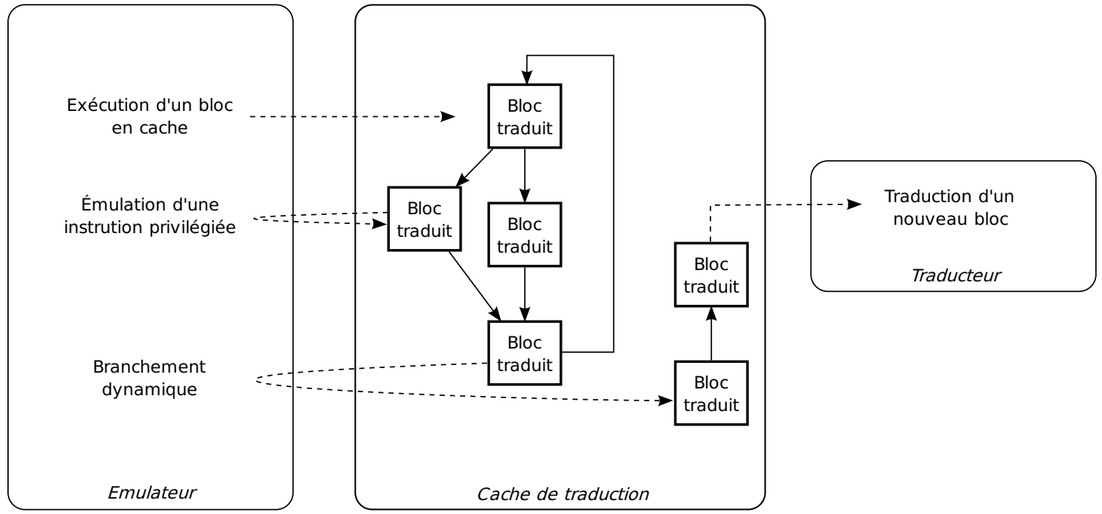
Non-conformité du x86

- Échecs silencieux
  - Exemple: POPF (échoue au critère 3b)
    - Contrôle entre autre le masquage des interruptions (EFLAGS)
    - Ignoré si le niveau de privilège est insuffisant
- Lectures d'informations incorrectes
  - Exemples: MOV (échoue au critère 3a)
    - Permet de lire le contenu du registre CS
    - Contient le niveau de privilège courantau
    - Lecture du mauvais niveau de privilèges

Virtualisation du jeu d'instruction x86

Émulation efficace

- Popularisé par VMWare
- Émulation de l'OS invité uniquement
- Traduction binaire et caches de traduction
  - Analyser chaque instruction une par une serait trop lent
  - Traduction de blocs d'instruction sans branchements
  - Instructions modifiées:
    - Références mémoire, branchement dynamiques
    - Instructions sensibles



Virtualisation du jeu d'instruction x86

Para-virtualisation du processeur

- Popularisé par l'hyperviseur Xen
  - Processeur virtuel légèrement différent d'un processeur x86
    - Non-support des instructions problématiques
  - Le sytème d'exploitation de la VM sait qu'il est virtualisé
  - Coopération avec l'hypervieur
    - Fonctionnalités exposées via *hypercall*
- Ne fonctionne qu'avec des systèmes d'exploitation modifiés
- Nécessite un support de la segmentation pour être vraiment performant
  - Pas disponible en x86_64

Virtualisation du jeu d'instruction x86

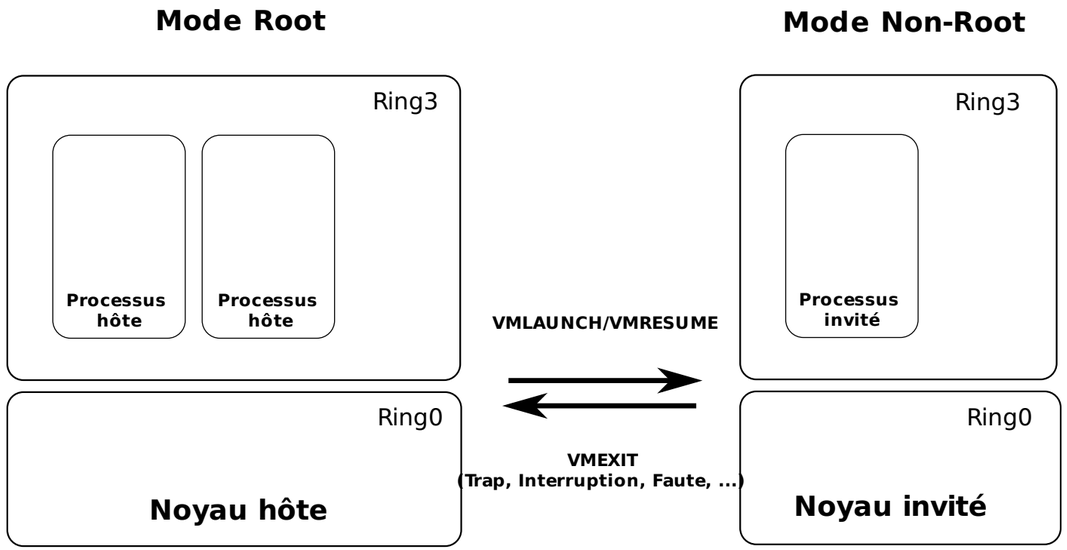
Extensions du jeu d'instruction x86

- Introduites par AMD et Intel face à la demande croissante de virtualisation
  - AMD: SVM / AMD-V
  - Intel: VT-x
- Extensions aux principes similaires:
  - Permettre l'utilisation du *trap and emulate*
  - Modes d'exécution orthogonaux aux niveaux de privilèges du x86
    - **root**: mode privilegié pour l'hôte
    - **non-root**: mode déprivilégié pour les invités
  - Tous les niveaux de privileges x86 sont accessibles dans chaque mode
    - Limite le nombre de transitions VM / VMM

Virtualisation du jeu d'instruction x86

Fonctionnement de VT-x

- Transitions root / non-root
  - Appel de VMLAUNCH/VMRESUME par l'hyperviseur
    - Déclenche une VMENTRY: bascule en mode non-root
    - Restauration de l'état du processeur virtuel
  - Conditions déclenchant un VMEXIT
    - Retour en mode root
    - Restauration de l'état du processeur hôte



Virtualisation du jeu d'instruction x86

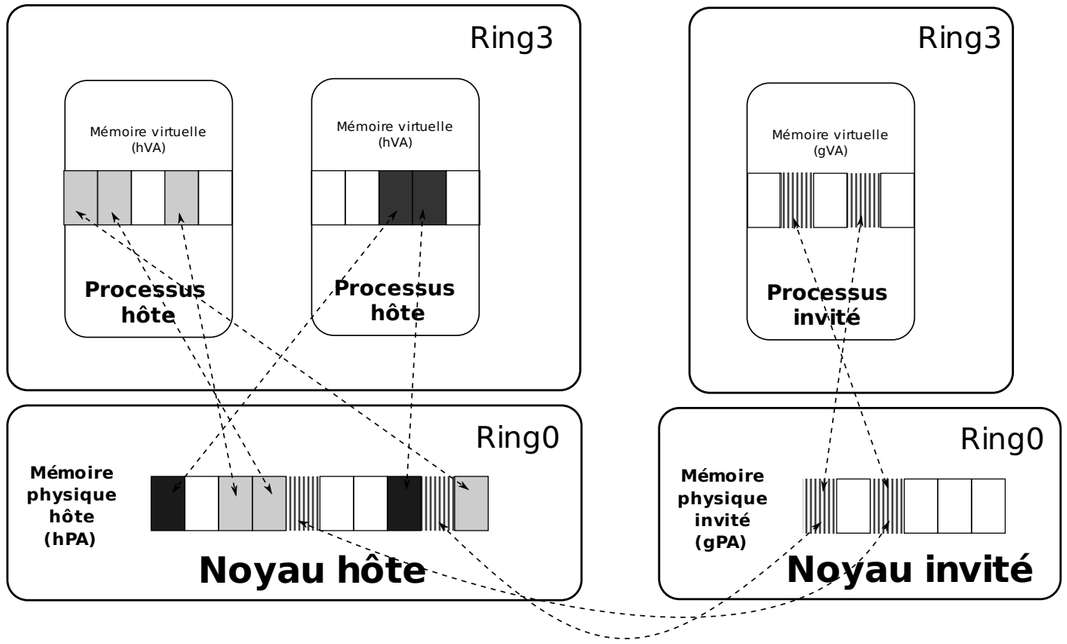
Fonctionnement de VT-x

- Structure VMCS (VM Control Structure)
  - Contrôle des transistions root / non-root
- Contient notamment:
  - État du processeur à restaurer lors de la prochaine transition
    - VMENTRY: sauvegarde de l'état hôte et restauration de l'état invité
    - VMEXIT: sauvegarde de l'état invité et restauration de l'état hôte
  - Configuration des conditions déclenchant un VMEXIT
    - Instructions à intercepter
    - Défaut de pages
    - Interruptions matérielles …
  - Cause de la dernière sortie du mode invité
    - Simplifie le travail de l'hyperviseur

Virtualisation de la mémoire

Unité de gestion de mémoire (MMU) x86

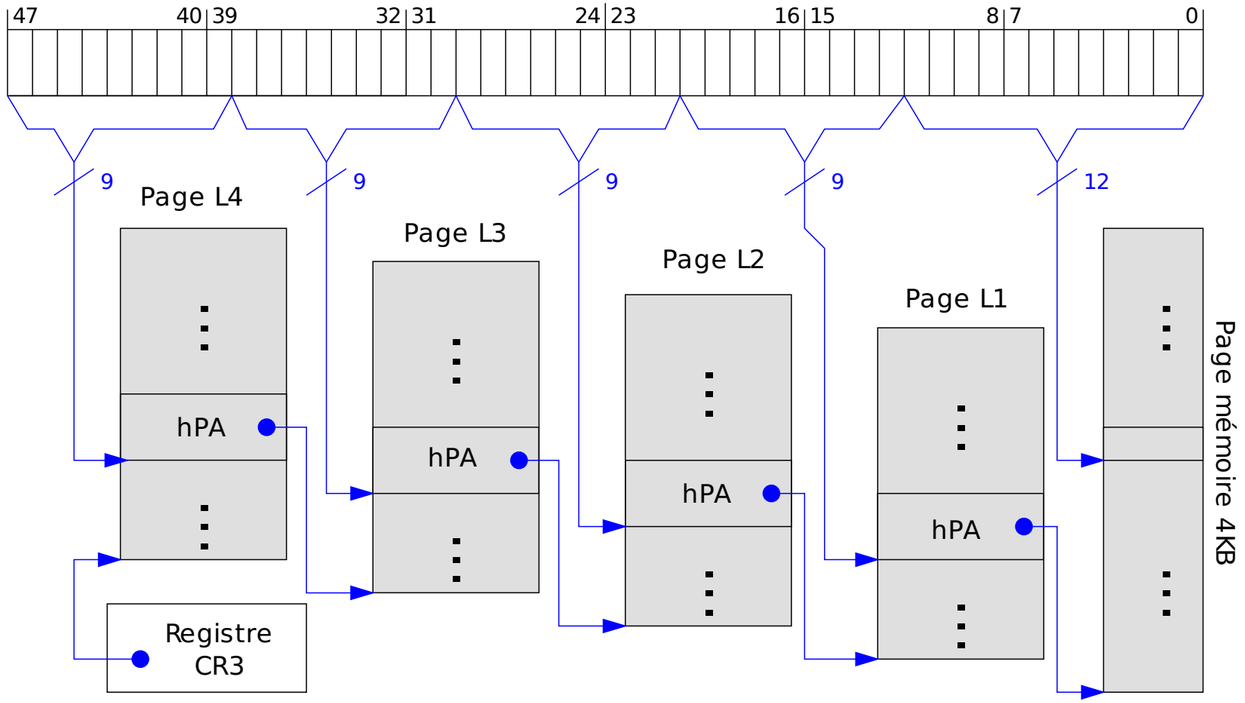
- Hors des phases d'intialisation: mode protégé / long
- Les instructions assembleur manipulent des adresses logiques
  - Aussi appelé adresses virtuelles
    - **hVA**: host virtual address, **gVA**: guest virtual address
  - Pointe vers des zones mémoires physiques
    - **hPA**: host physical address, **gPA**: guest physical address



Virtualisation de la mémoire

Unité de gestion de mémoire (MMU) x86

- Table des pages (TDP)
  - Correspondance avec les adresses physiques
- TLB: Translation Lookaside buffer
  - Coût du parcours hiérarchique de la TDP (4 accès mémoire)
  - Cache du résultat des précedents parcours

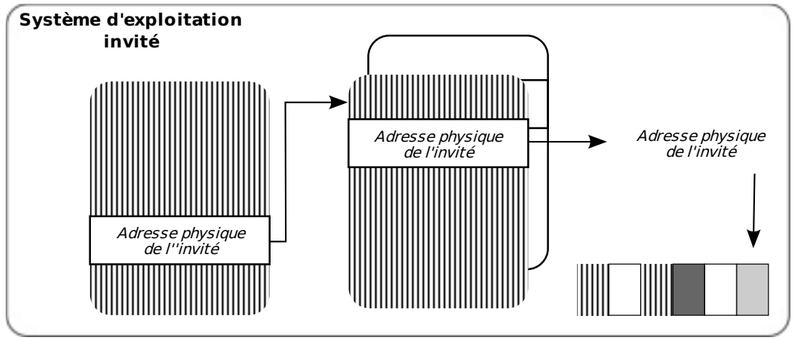


Virtualisation de la mémoire

Table des pages fantômes

- La VM dispose de mémoire physique virtuelle (gPA)
- Les tables des pages qu'elle met en place sont virtuelles
  - Traduction gVA -> gPA
  - Jamais utilisée directement par la MMU physique

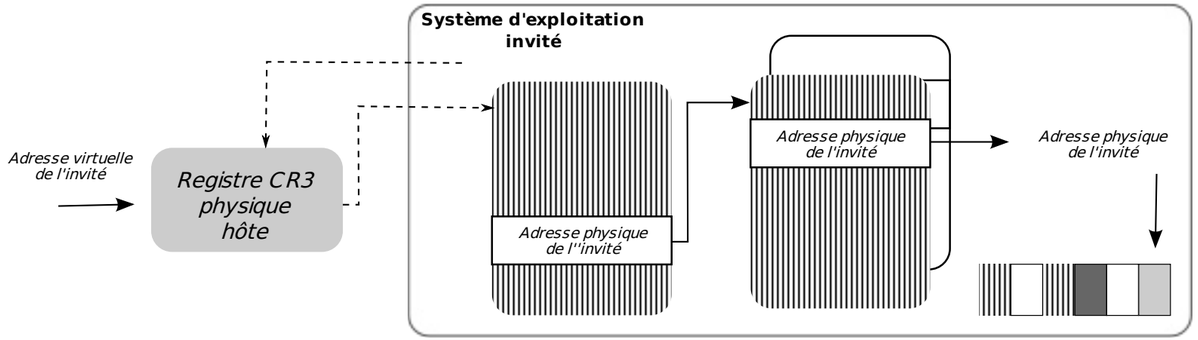
Registre CR3 physique hôte



Virtualisation de la mémoire

Table des pages fantômes

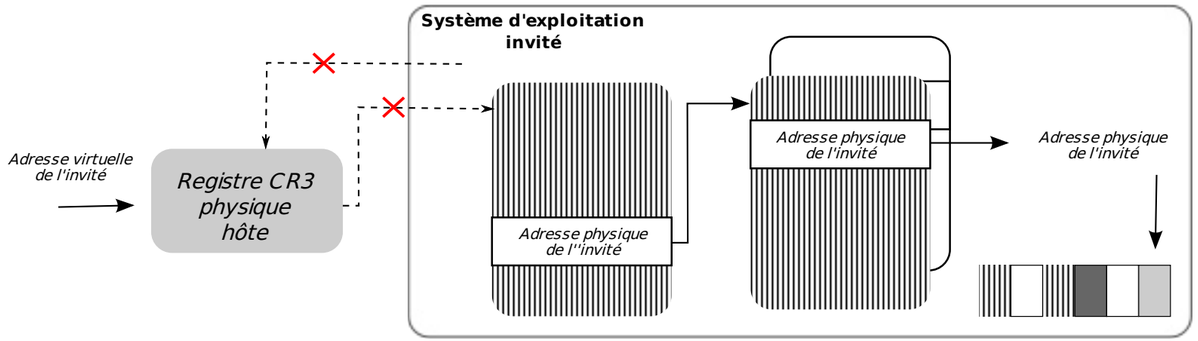
- Le VMM maintient en cohérence une table des pages fantôme
  - Installée sur le processeur quand la VM s'exécute
  - Contient la traduction gVA -> hPA
  - Calculé par la composition gVA -> gPA et gPA -> hPA



Virtualisation de la mémoire

Table des pages fantômes

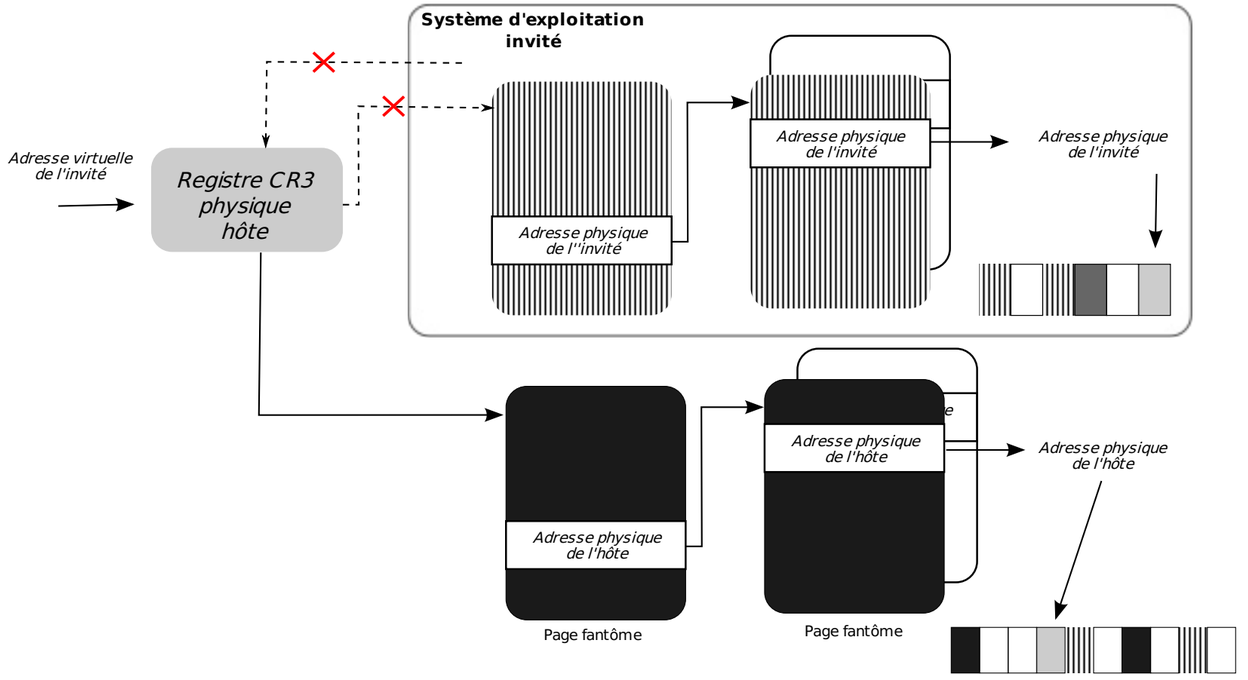
- Le VMM maintient en cohérence une table des pages fantôme
  - Installée sur le processeur quand la VM s'exécute
  - Contient la traduction gVA -> hPA
  - Calculé par la composition gVA -> gPA et gPA -> hPA



Virtualisation de la mémoire

Table des pages fantômes

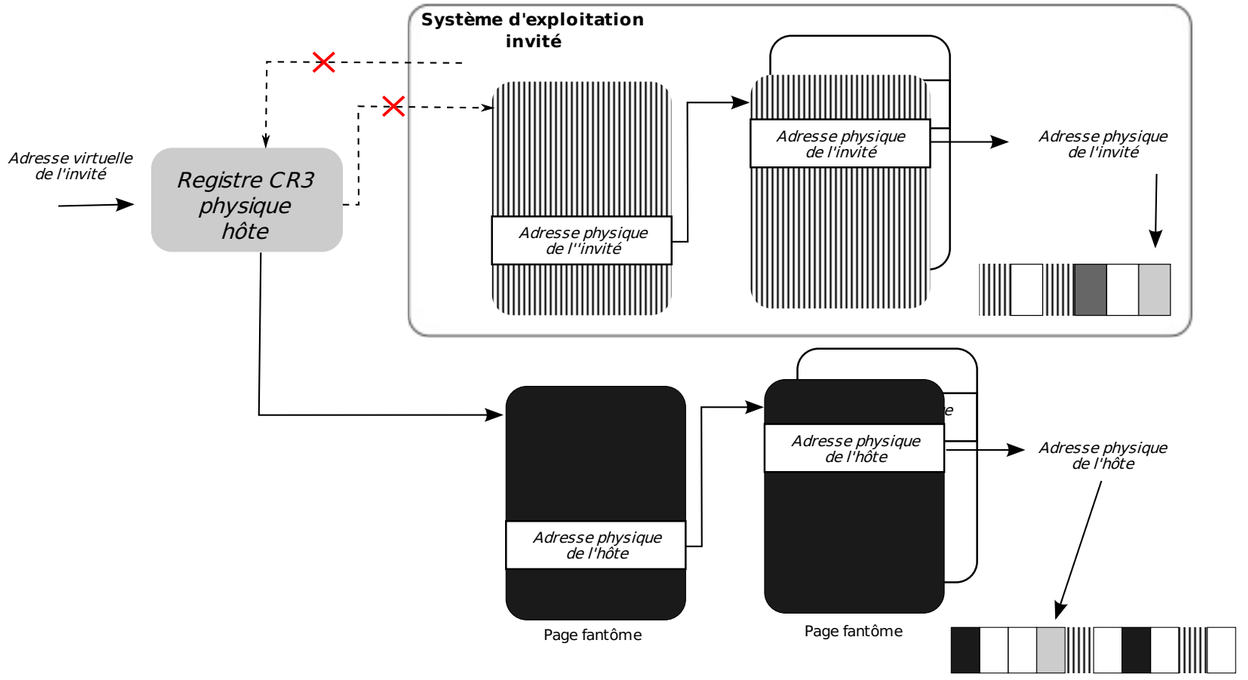
- Le VMM maintient en cohérence une table des pages fantôme
  - Installée sur le processeur quand la VM s'exécute
  - Contient la traduction gVA -> hPA
  - Calculé par la composition gVA -> gPA et gPA -> hPA



Virtualisation de la mémoire

Table des pages fantômes

- Mise en cache des table des pages fantômes
- Interception des modifications de la TDP virtuelle
  - Protection en écriture des pages de la TDP virtuelle
    - Complexe: trouver les entrées permettant d'écrire dans ces pages
    - Heuristiques de détection de changement d'utilisation des pages
    - Évaluation paresseuse en cas de modification multiples



Virtualisation de la mémoire

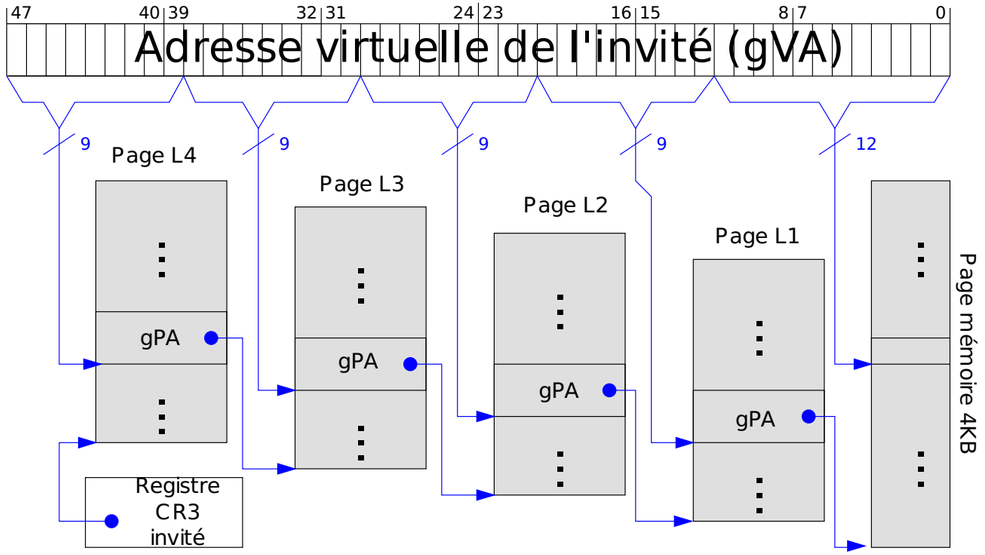
Table des pages imbriquées

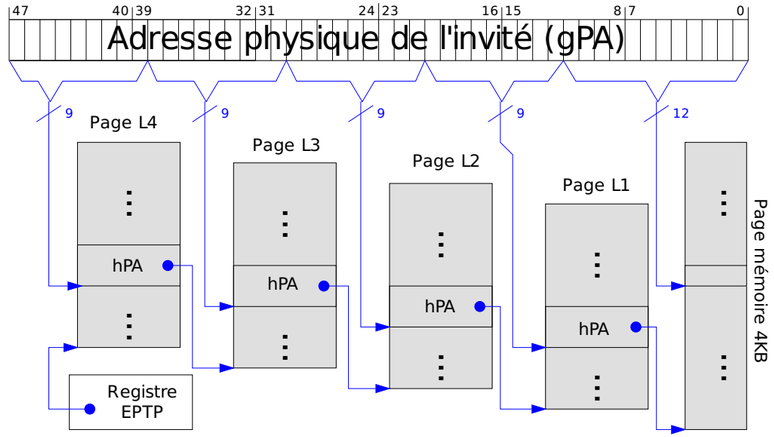
- Extensions matérielles introduites par AMD et Intel
  - Améliorer les performances
  - Passage à l’échelle (nombre de CPUs virtuels)
  - Simplifier les hyperviseurs
    - Sécurité

Virtualisation de la mémoire

Table des pages imbriquées

- Le processeur parcourt deux niveaux de table des pages
  - CR3 invité: table des pages gVA -> gPA
    - Utilisable par l'OS invité sans *trap*
  - EPTP: table des pages gPA -> hPA (spécifié via VMCS)

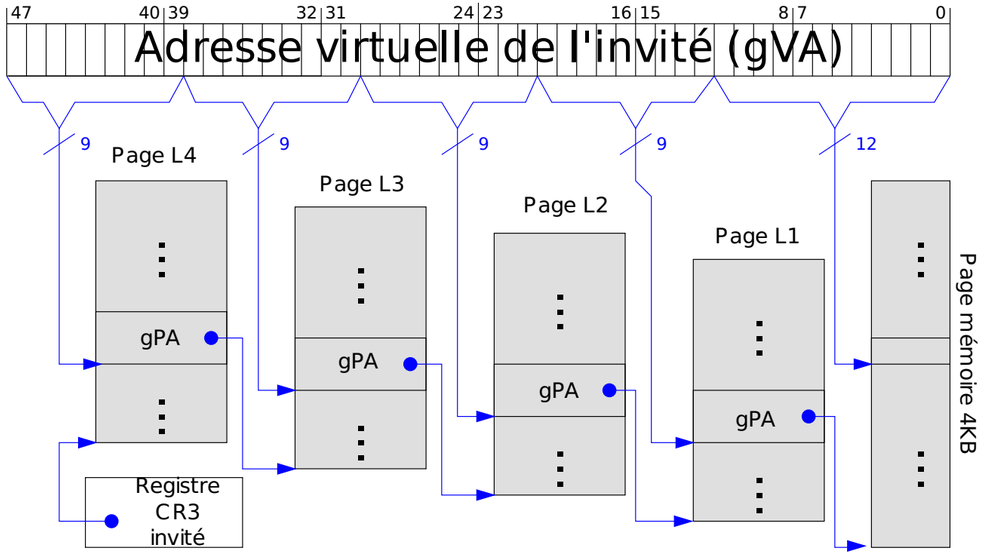


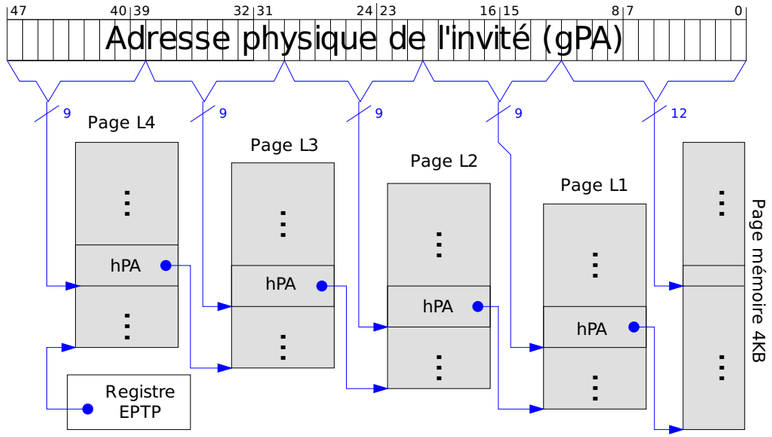


Virtualisation de la mémoire

Table des pages imbriquées

- La TLB cache les traductions gVA -> hPA
- Problème: coût des défauts de TLB
- Calculer le nombre d'accès mémoires nécessaires à une traduction

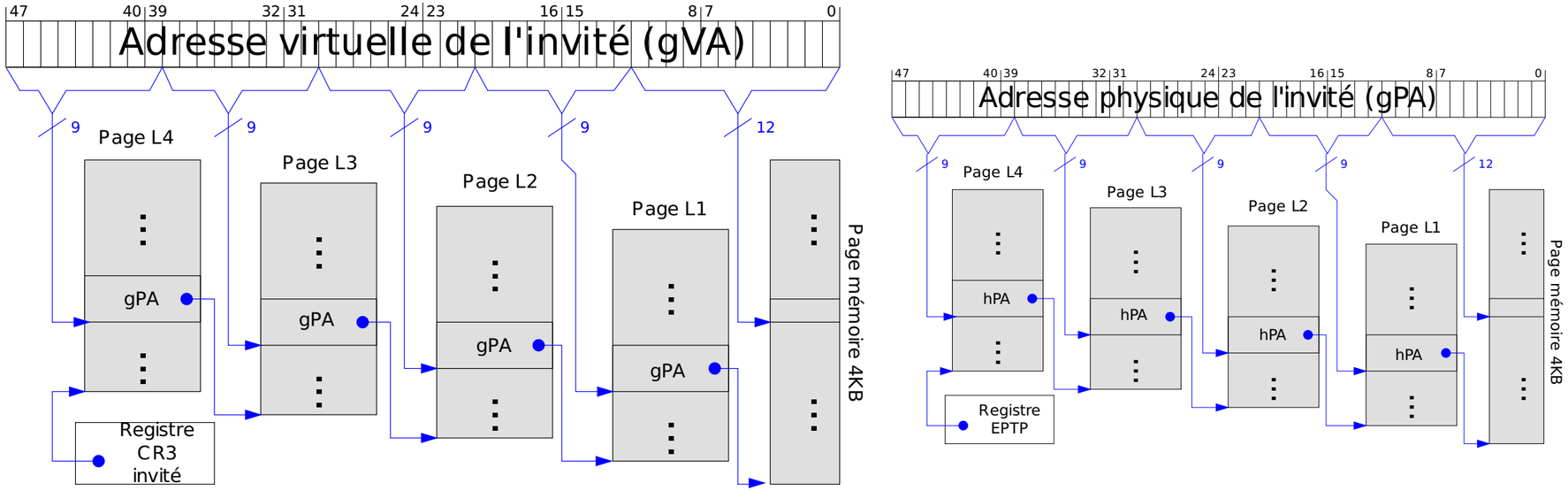




Virtualisation de la mémoire

Table des pages imbriquées

- 24 accès mémoire au total !
  - Si la TDP est assez statique les TDP fantômes sont plus efficaces
  - Les TDP imbriquées sont plus performantes dans la plupart des cas
    - Utilisées en standard sur les processeurs récents

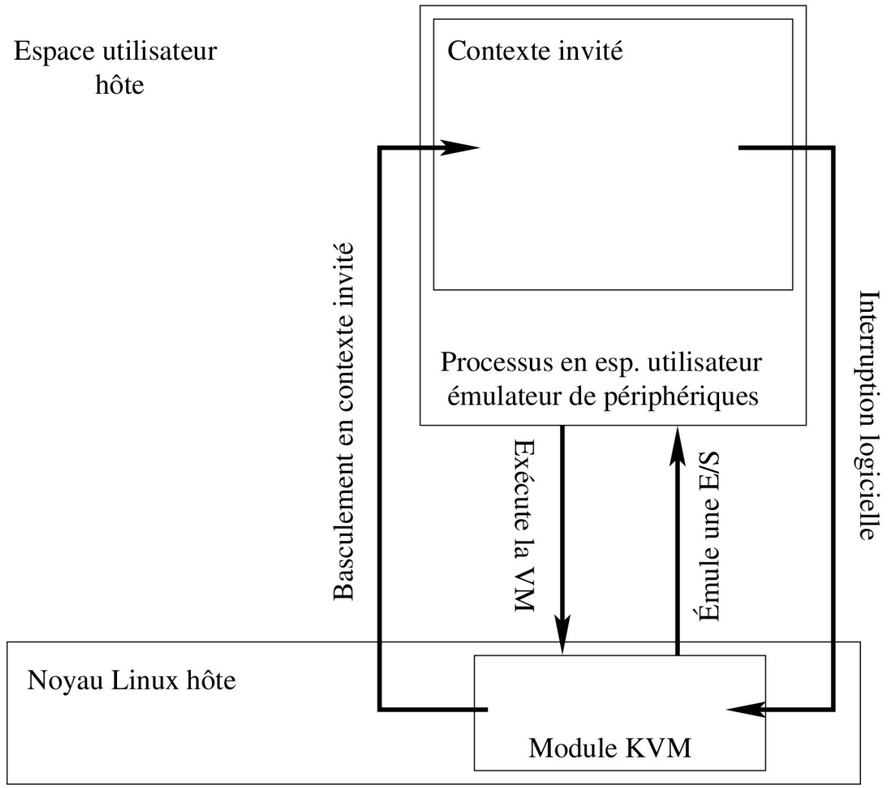


Le module KVM

- Module noyau Linux permettant d'implémenter un hyperviseur
  - Expose élégamment les fonctionnalités de virtualisation
    - Processeur et mémoire
    - Contrôleur d'interruption
  - Les VMs sont des processus Unix standards
    - Mémoire allouée par le processus
    - Un thread par CPU virtuel (vCPU)
    - E/S via les interfaces standard fournies par le noyau
  - Intégration parfaite dans l'ensemble du système
    - Ordonnancement standard
    - Outils Unix ps, top, nice, kill …
    - Cgroups

Le module KVM

Vue d'ensemble du fonctionnement



Le module KVM

Mise en place de la mémoire d'une VM

```
    /*Ouverture de l'interface kvm et création d'une VM*/
    kvm = open("/dev/kvm", O_RDWR | O_CLOEXEC);
    vmfd = ioctl(kvm, KVM_CREATE_VM, (unsigned long)0);
    /*Alloue de la mémoire avec l'appel système mmap*/
    mem = mmap(NULL, 0x1000000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS
    /*Positionne cette mémoire à l'adresse physique virtuelle 0x1000*/
    struct kvm_userspace_memory_region region = {
    .slot = 0,
    .guest_phys_addr = 0x1000,
    .memory_size = 0x1000000,
    .userspace_addr = (uint64_t)mem,
    };
    ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);
```

Le module KVM

Lancement de la VM

```
/*Création d'un vCPU*/
vcpu_fd = ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0);
vcpu_ctx = mmap(NULL, vcpu_size, PROT_READ | PROT_WRITE, MAP_SHARED, vcpufd, 0);
[...] /* Configuration de l'etat du vCPU (registres...) */
while (1) {
    /*Exécution de la machine virtuelle*/
  ioctl(vcpufd, KVM_RUN, NULL);
  /*Analyse de la cause du VMEXIT*/
  switch (vcpu_ctx->exit_reason) {
         case KVM_EXIT_IO:
         /* Traitement de l'E/S */
  }
}
```

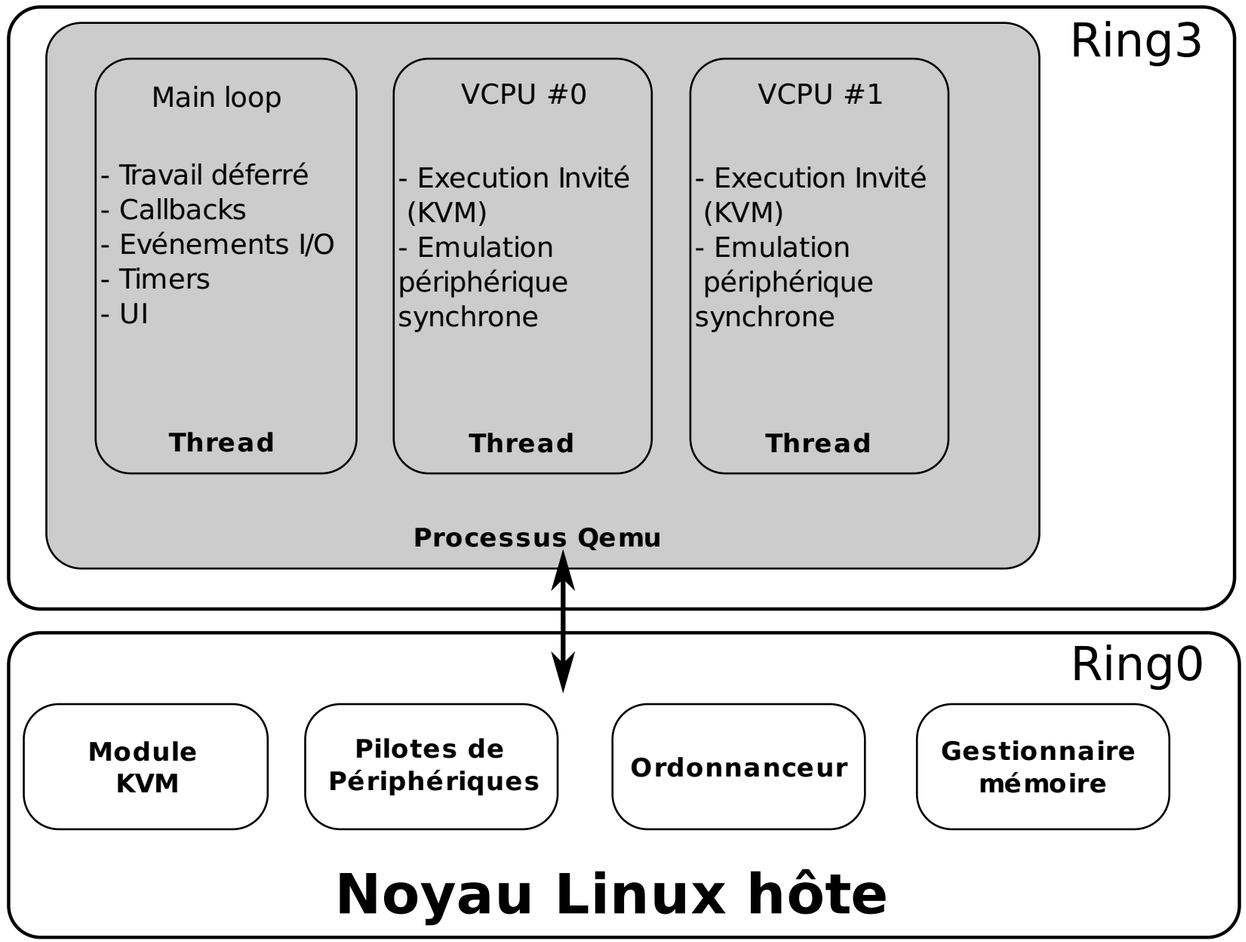
Qemu

Présentation

- Émulateur open source très versatile
  - Principaux jeux d’instructions
    - Traduction binaire
  - Très grand nombres de périphériques
- Exploite KVM pour former des machines virtuelles
  - KVM virtualise le CPU et la mémoire
  - Qemu émule les autres composants

Qemu

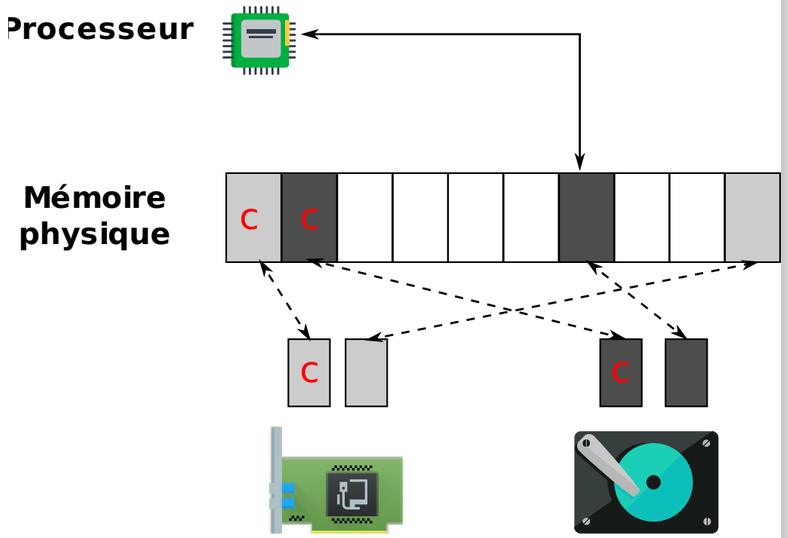
Architecture



Émulation d'un périphérique (frontend)

Programmation d'un périphérique

- Périphériques PCI(-Express), plateforme x86
- Un périphérique expose son interface via des registres
- Deux mécanisemes permettent d'y accèder
  - I/O ports
    - Mécanisme historique
    - Accès via 65536 adresses de ports
    - Instructions IN/OUT
  - Memory Mapped I/O (MMIO)
    - Accès via des adresses mémoire
    - Instructions MOV
  - Comment savoir:
  - Quels périphériques sont présents ?
  - A quelle adresse sont-ils accessibles ?

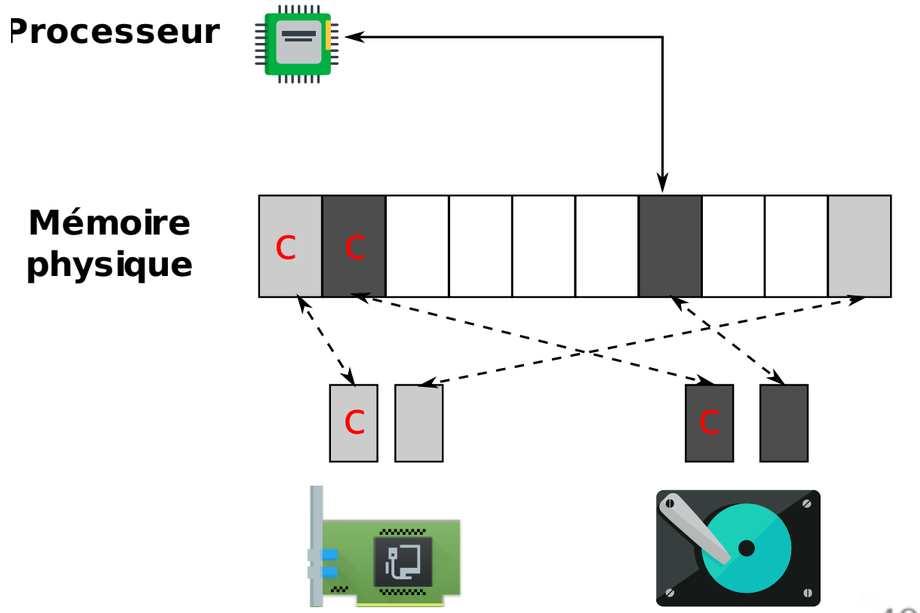


Émulation d'un périphérique (frontend)

Espace de configuration PCI

- Registres de configuration exposés par chaque périphérique
  - Structure standardisée
    - Vendor et Device ID
  - Base Address Registers (BARs)
    - Configuration des adresses associées à une ressource
    - Adresses mémoire ou I/O ports

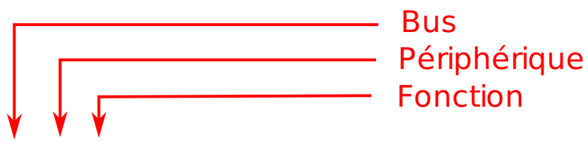
31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	0Ch
Base Address Registers				10h–24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

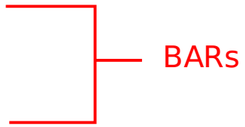


Émulation d'un périphérique (frontend)

Découverte des périphériques

- Chaque périphérique PCI est identifié par
  - Numéro de bus: 8 bits
  - Numéro de périphérique (dipositif physique): 5 bits
  - Numéro de fonction (dispositif logique): 3 bits

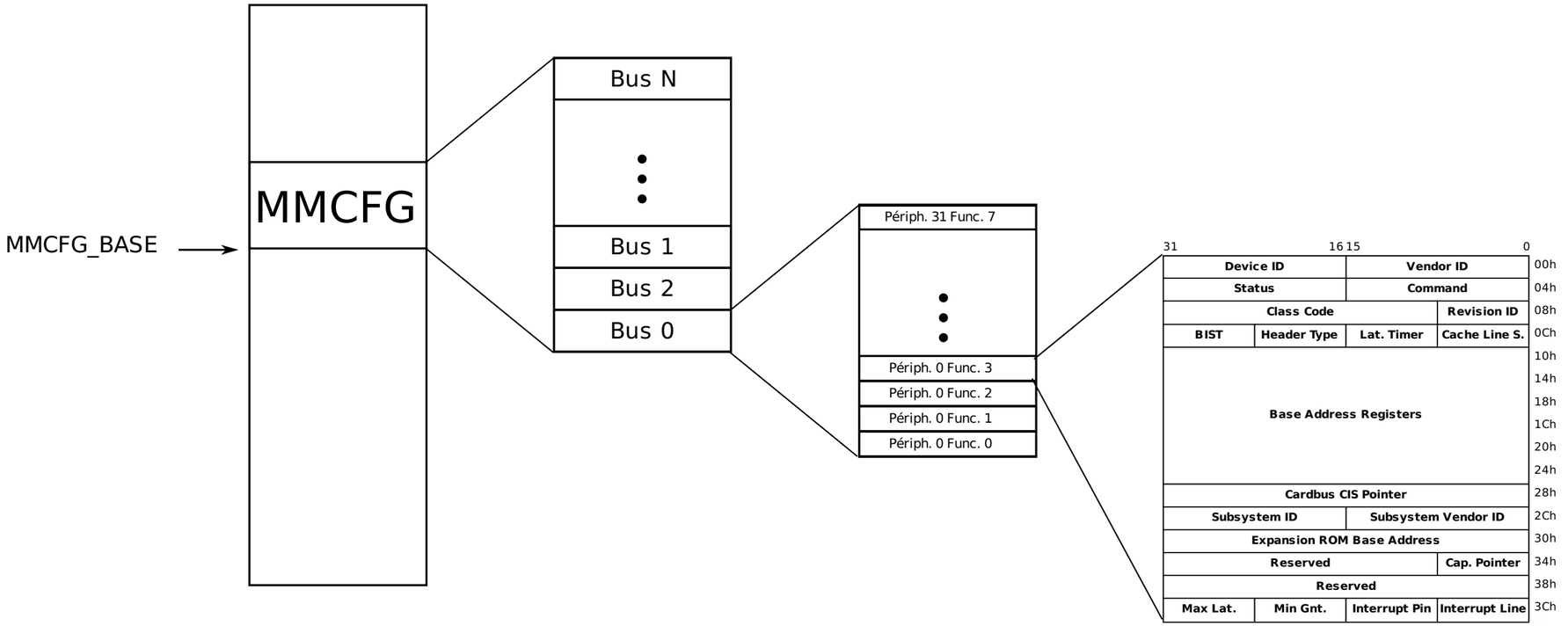




Émulation d'un périphérique (frontend)

Découverte des périphériques

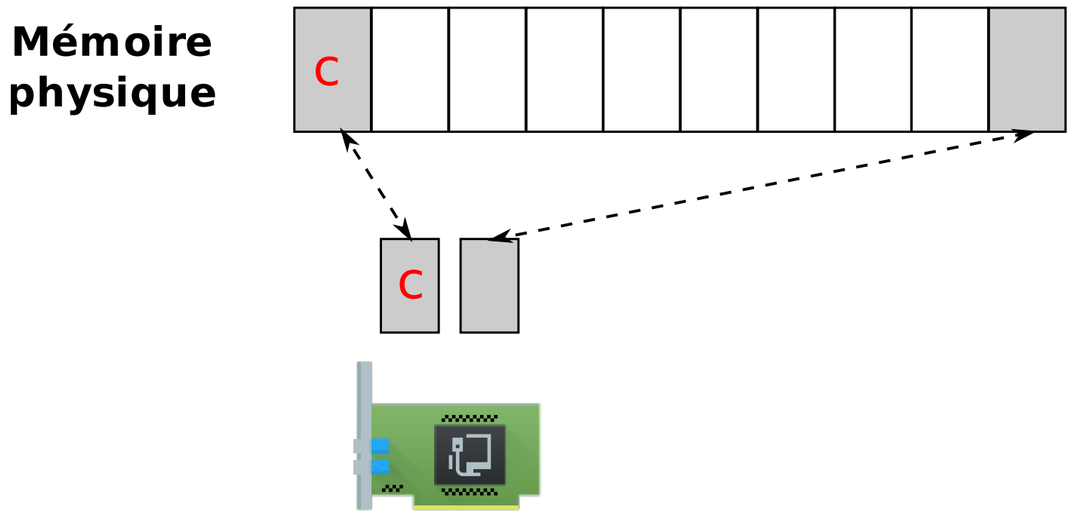
- Accès à l'espace de configuration d'un périphérique via son identifiant
  - Accès MMIO à une adresse fournie par le firmware
  - Accès historique par deux I/O ports spécifiques
    - Un port d'adresses et un port de données



Émulation d'un périphérique (frontend)

Transferts de données

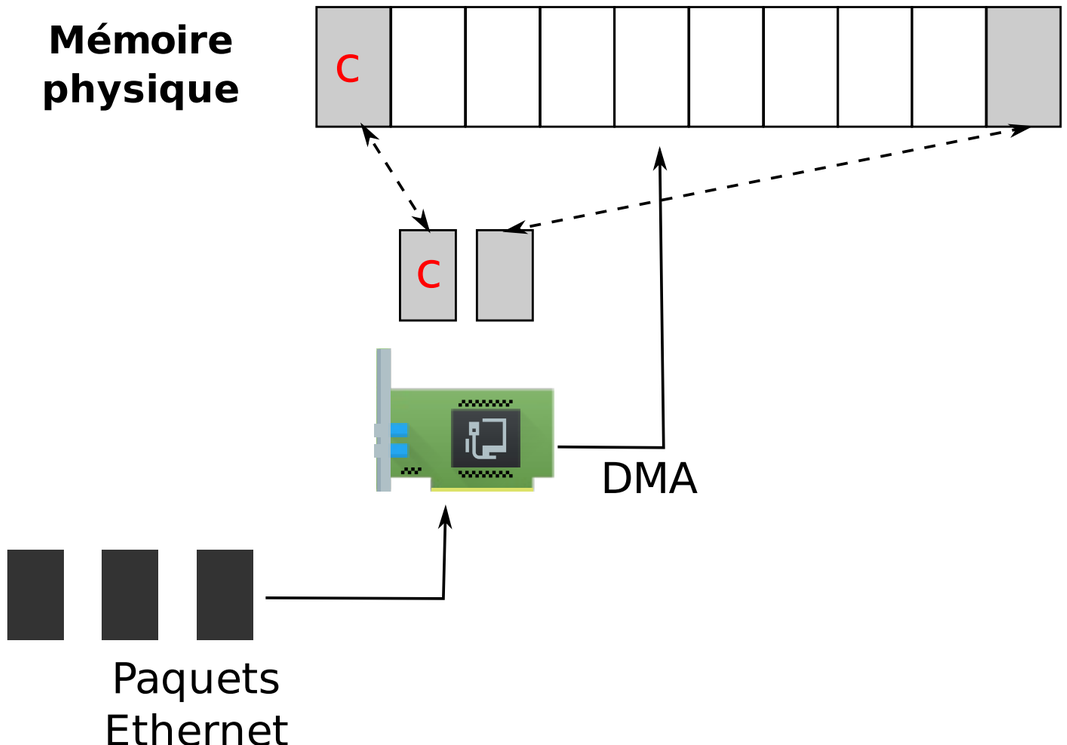
- Direct Memory Access
  - Le périphérique lit/ecrit directement dans la mémoire du système.
  - Exemple: paquets réseaux, blocs disques



Émulation d'un périphérique (frontend)

Transferts de données

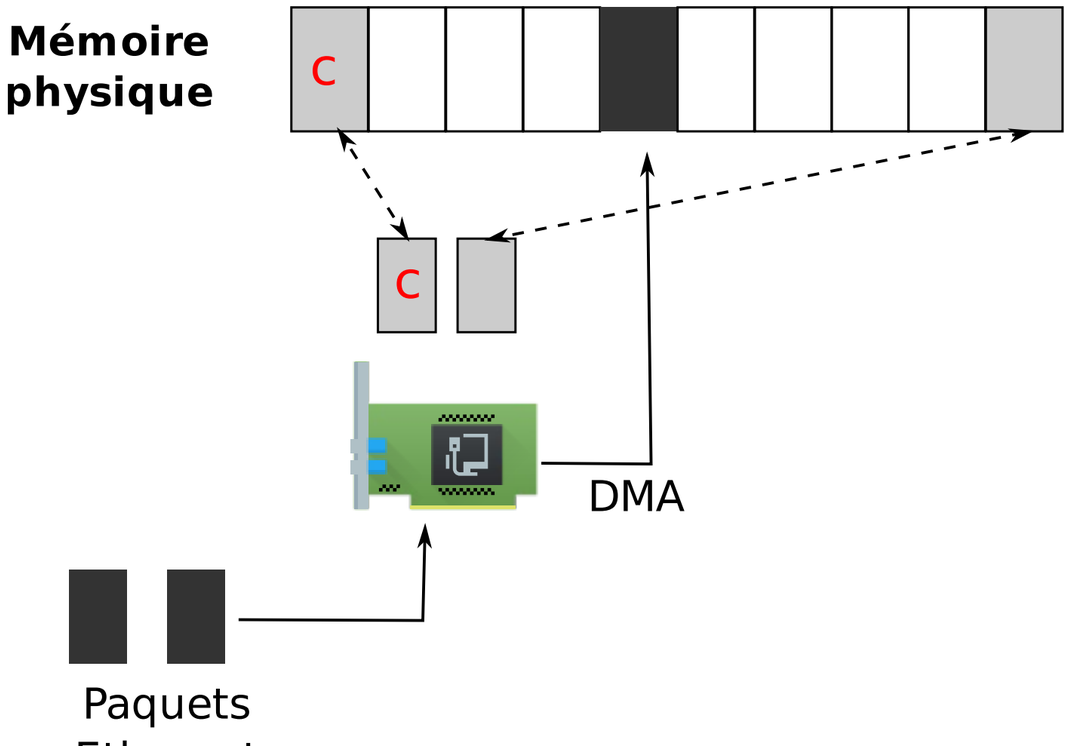
- Direct Memory Access
  - Le périphérique lit/ecrit directement dans la mémoire du système.
  - Exemple: paquets réseaux, blocs disques



Émulation d'un périphérique (frontend)

Transferts de données

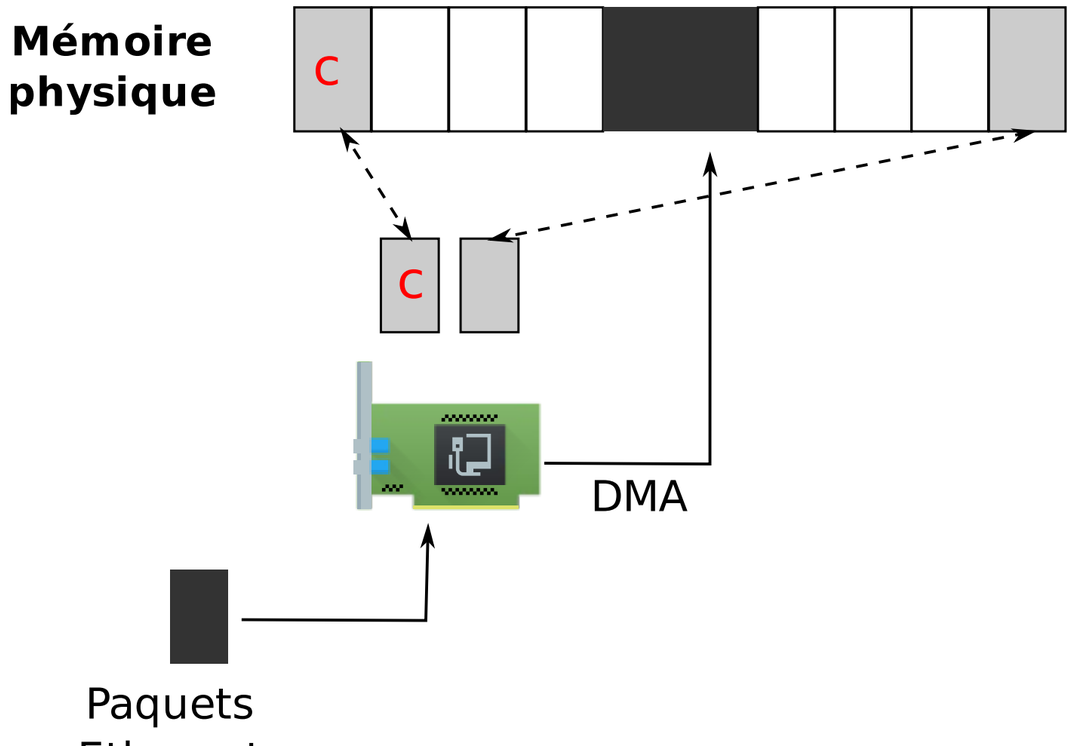
- Direct Memory Access
  - Le périphérique lit/ecrit directement dans la mémoire du système.
  - Exemple: paquets réseaux, blocs disques



Émulation d'un périphérique (frontend)

Transferts de données

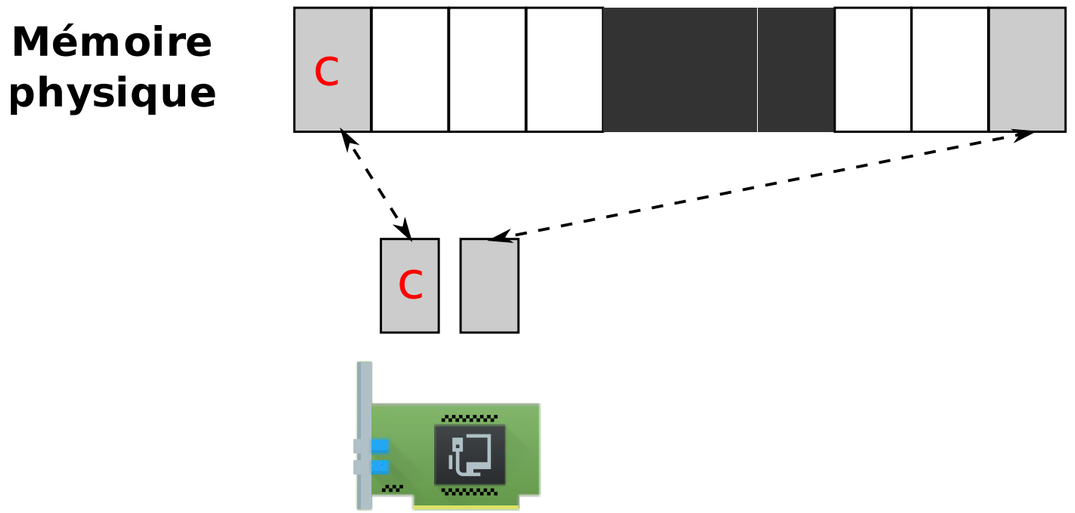
- Direct Memory Access
  - Le périphérique lit/ecrit directement dans la mémoire du système.
  - Exemple: paquets réseaux, blocs disques



Émulation d'un périphérique (frontend)

Transferts de données

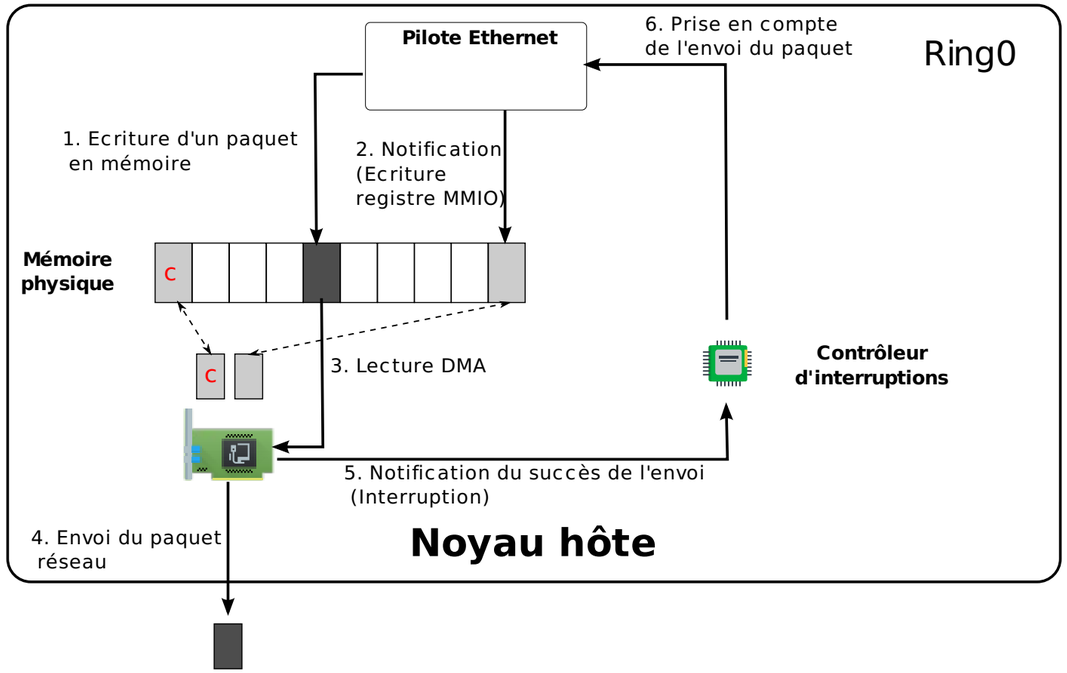
- Direct Memory Access
  - Le périphérique lit/ecrit directement dans la mémoire du système.
  - Exemple: paquets réseaux, blocs disques



Émulation d'un périphérique (frontend)

Notifications

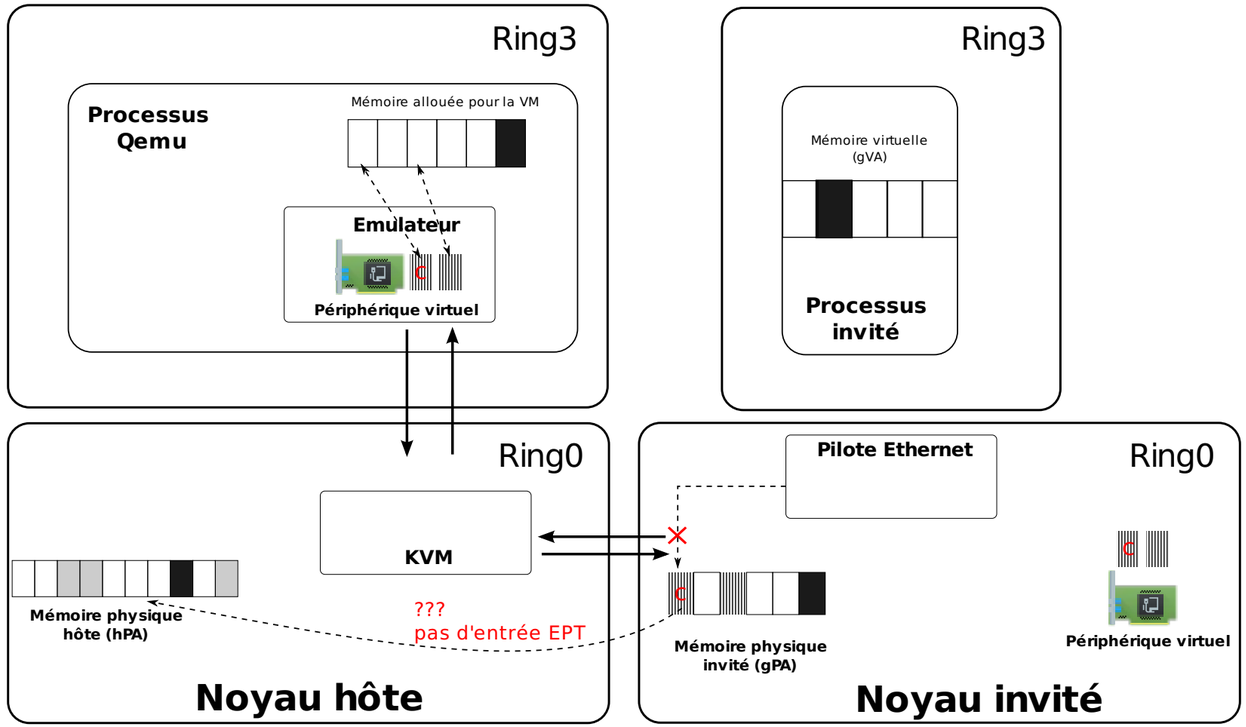
- Processeur -> Périphérique : modification de registres (accès MMIO)
- Périphérique -> Processeur: interruptions
  - Éviter l'attente active (polling)
  - Vecteurs d'interruption (256 sur plateforme intel) associés à des fontcions
    - Choix du vecteur utilisé via le config space du périphérique
    - Interruption du processeur pour exécuter la fonction



Émulation d'un périphérique (frontend)

Émulation des accès aux registres (MMIO)

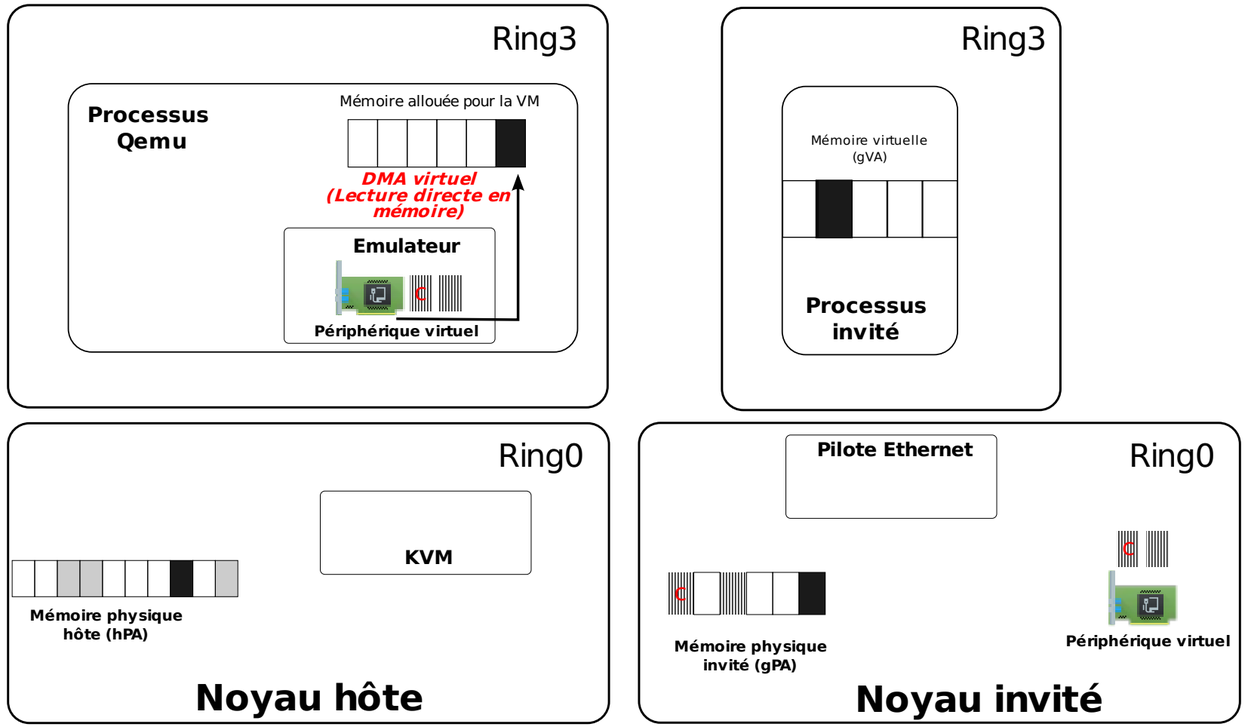
- KVM
  - VMEXIT (pas d'adresse hPA correspondante)
  - Analyse de l'exception, émulation de l'instruction (adresse lue/écrite, EIP)
- Qemu
  - Émulation du périphérique, renvoie la valeur lue



Émulation d'un périphérique (frontend)

Émulation des accès DMA

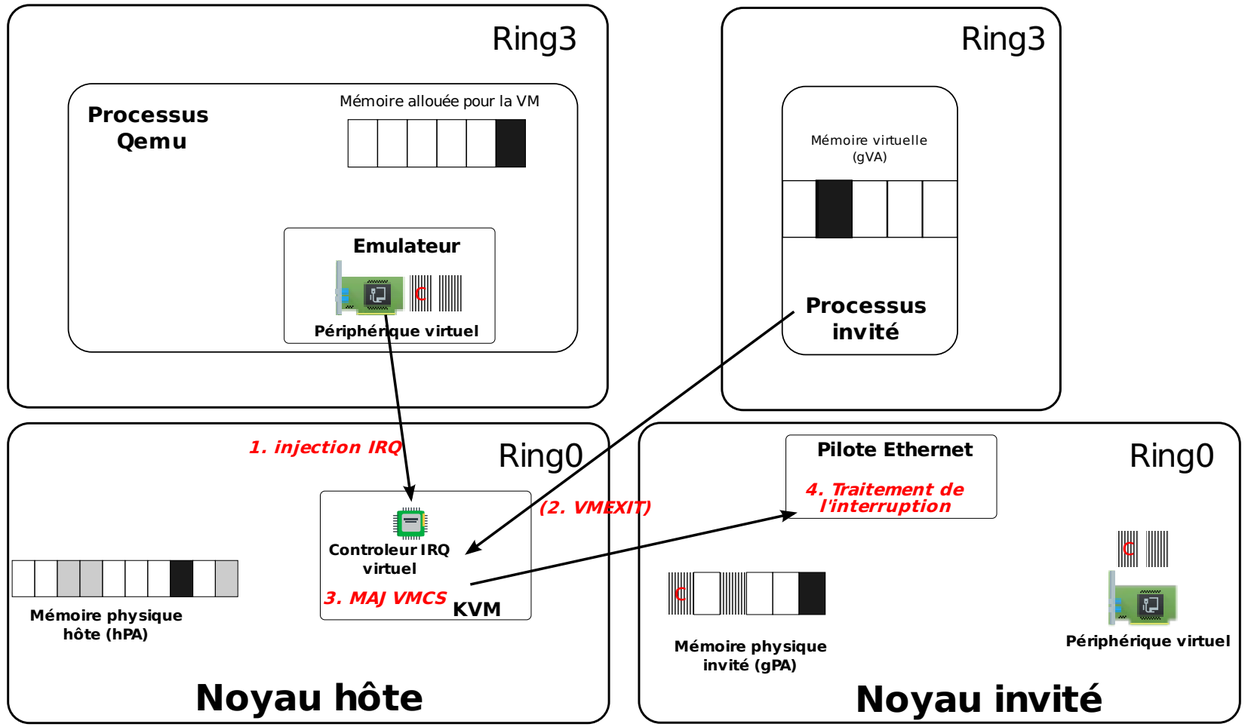
- Qemu accède directement à la mémoire physique de la VM
  - Mémoire allouée en espace utilisateur hôte



Émulation d'un périphérique (frontend)

Émulation des interruptions

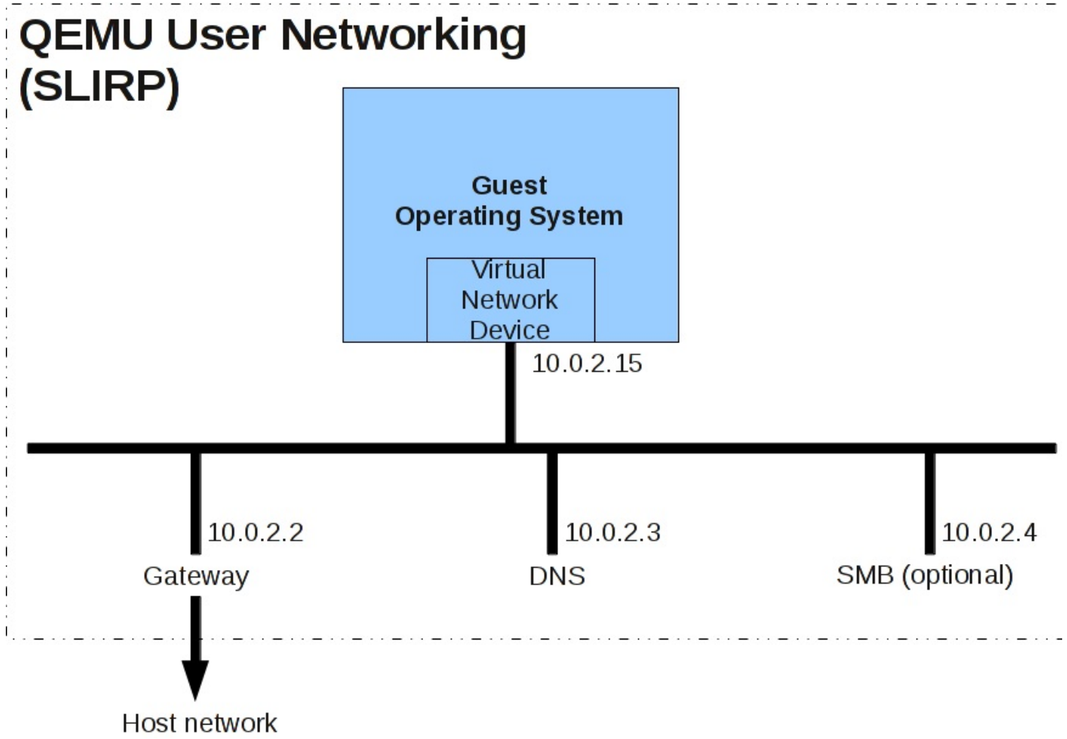
- L'API KVM permet d'injecter des iterruptions (via un ioctl ou irqfd)
- Modifie le contrôleur d'interruption virtuel du vCPU cible
  - Si besoin, envoi IPI physique pour sortir le CPU du mode invité
- Modification VMCS
  - Sélection d'un vecteur d'interruption: exécuté au VMENTRY suivant



Émulation d'un périphérique (backend)

Périphériques réseau Ethernet

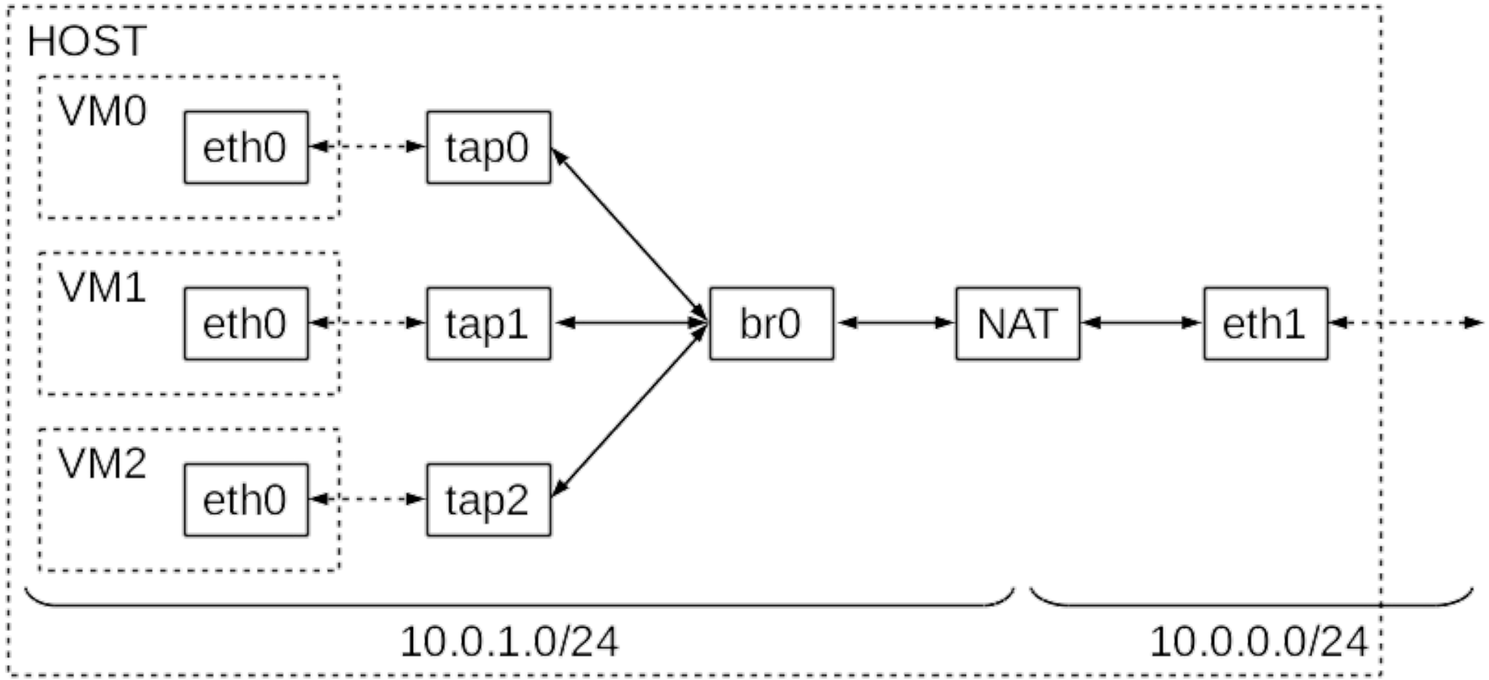
- Backend réseau utilisateur
  - Pile TCP/IP en espace utilisateur
  - Interprète les paquets Ethernet dans le processus Qemu
  - Connexion aux machines distantes via sockets udp/tcp



Émulation d'un périphérique (backend)

Périphériques réseau Ethernet

- Backend TUN/TAP
  - Relai les paquets sur un switch virtuel
  - Connexion du switch à un réseau phyisque ou un routeur NAT



Émulation d'un périphérique (backend)

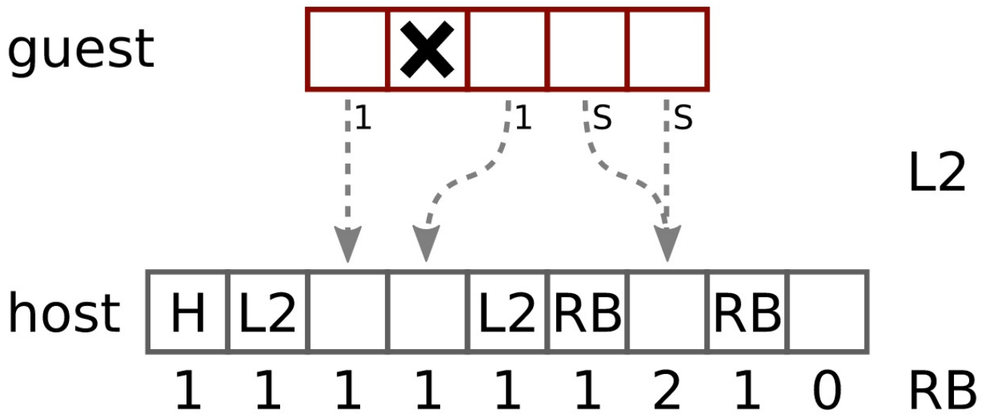
Périphériques de stockage bloc (disques virtuels)

- Nombreux backends disponibles
  - Système de fichiers local (ext4, xfs, …)
  - Système de distant (nfs, glusterfs, lustre, …)
  - Périphérique bloc local (disque physique, lvm, …)
  - Périphérique bloc distant (ceph, iscsi, …)

Émulation d'un périphérique (backend)

Format de fichiers d'image bloc

- raw: stockage des données brutes
- qcow2: Qemu Copy on Write
  - Fichiers de références
  - Snapshots internes
  - Allocation granulaire (thin provisionning)
  - Compression



Émulation d'un périphérique (backend)

Politique de gestion du cache

- cache=writeback:
  - Qemu ouvre et écrit dans les fichiers de destination normalement
    - Les données passent par le cache d'I/O hôte
  - Grande quantité de données en vol
    - Dans le cache d'I/O mais pas encore sur disque
  - Le système de fichier invité doit gérer correctement les commandes *flush*
    - Corruption assurée en cas de crash dans le cas contraire
  - Doublon des caches d'I/O hôte et invité
    - Consommation et copies mémoires souvent inutile
    - Peut permettre un partage entre plusieurs VMs

Émulation d'un périphérique (backend)

Politique de gestion du cache

- cache=none:
  - Qemu utilise le flag O_DIRECT pour l'écriture des données
    - N'utilise pas le cache d'I/O hôte
  - La ressource sous-jacente peut avoir un cache interne
    - L'invité doit correctement envoyer des commandes *flush*
  - Pas compatible avec tous systèmes de fichiers hôte
  - Fonctionnement le plus proche d'un périphérique physique
  - Généralement plus performant avec du matériel haut de gamme

Émulation d'un périphérique (backend)

Politique de gestion du cache

- cache=writethrough:
  - Qemu utilise le flag O_DSYNC pour l'écriture des données
    - Une écriture rend la main quand les données sont sur disque
    - Le cache de page est peuplé en lecture
  - Évite au maximum les possibilités de corruption de données
    - L'invité n'a pas besoin d'envoyer de commande *flush*
  - Pénalise les performances

Émulation d'un périphérique (backend)

Politique de gestion du cache

- cache=unsafe:
  - Qemu ouvre et écrit dans les fichiers de destination normalement
  - Toutes les demandes de flush sont ignorées
  - Maximise les performances si on ne craint pas les pertes de données
    - VM éphémère
    - Phase d'installation d'une VM

Paravirtualisation d'un périphérique

Problématique des interfaces de périphériques

- Interfaces non conçues pour une émulation efficace
- Caractéristiques de performances des mécanismes virtuels et physiques
  - Registres
    - Physique: Rapide et simple
    - Émulation: Accès MMIO très lent (vmexit, décodage, …)
  - Transfert DMA
    - Physique: Plus complexe
    - Émulation: Simple et rapide (accès mémoire direct)
  - Interruptions:
    - Physique: Coûteux
    - Émulation: Très coûteux (multiples vmexit)
- Exemple: interface réseau Ethernet e1000
  - 6 accès registres par paquet envoyé
  - 1 interruption

Paravirtualisation d'un périphérique

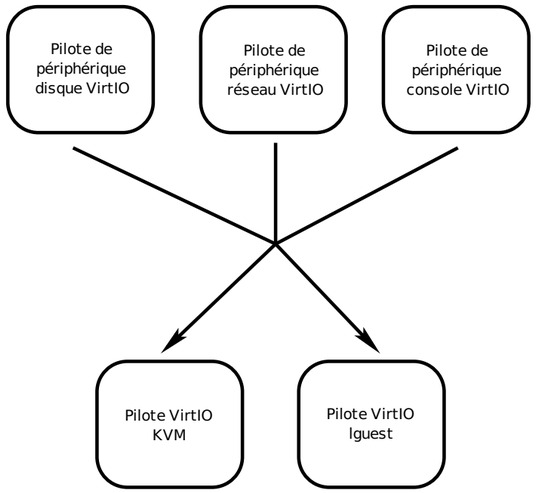
Interfaces paravirtualisées

- Périphériques purement virtuels
  - Aucune version physique des périphériques n'existe
- Conçu pour une virtualisation efficace
  - Prise en compte des performances des mécanismes de communication
- Inconvénient
  - Nécessite des drivers spécifiques dans l'OS invité
    - Pour chaque classe de périphériques (réseau, stockage bloc, …)
    - Pour chaque hyperviseur

Paravirtualisation d'un périphérique

Bus VirtIO

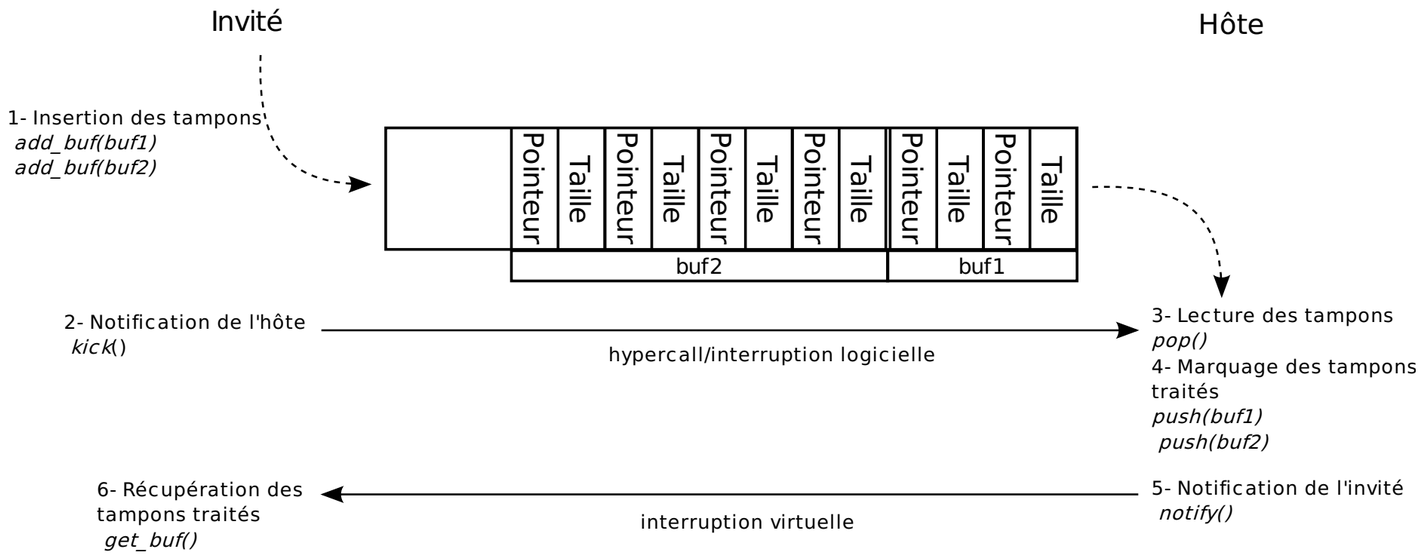
- Bus de communication standardisé pour interfaces para-virtualisées
- Découple les problématiques
  - Implémentation de chaque périphérique à l'aide de ce bus
  - Implémentation du bus par chaque hyperviseur
- Bénéfices
  - M + N drivers au lieu de M * N
  - Mécanismes communs à toutes les classes de périphériques



Paravirtualisation d'un périphérique

Bus VirtIO

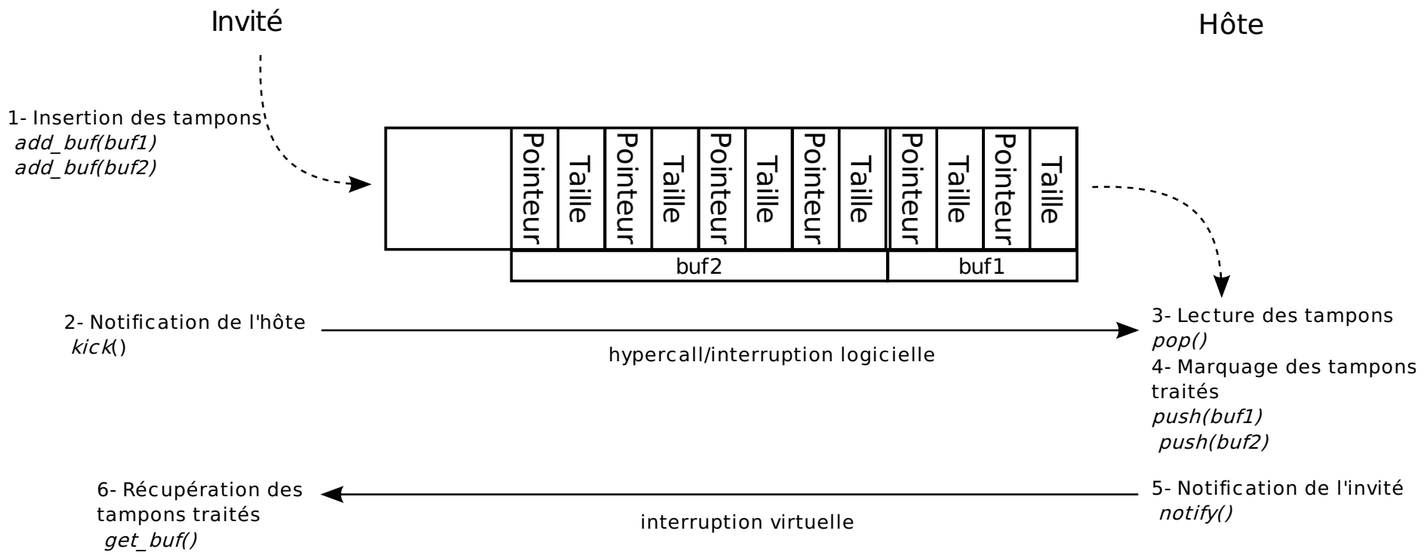
- Virtqueues: files de descripteur de messages
  - Allouées dans la mémoire de l'invité
  - Accès en DMA virtuel par l'hyperviseur



Paravirtualisation d'un périphérique

Bus VirtIO

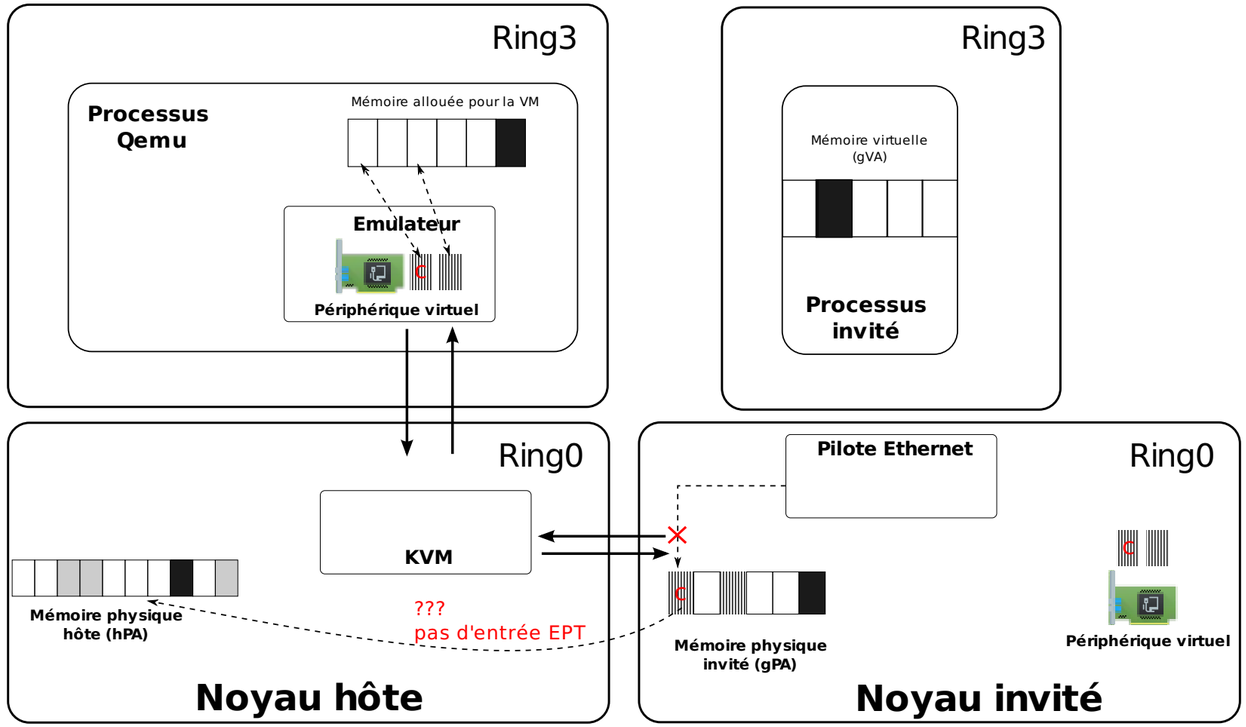
- Minimisation des interruptions et *trap* au strict nécessaire
  - Hyperviseur et invité travaillent en parallèle
  - Accès registre uniquement pour *reveiller* un thread hyperviseur
    - Consomme tous les descripteurs de la file
    - Plus d'accès registre tant que l'hyperviseur est actif
  - Symétriquement, interruption uniquement pour *reveiller* un thread invité



Minimisation des changements de contexte

Problématique

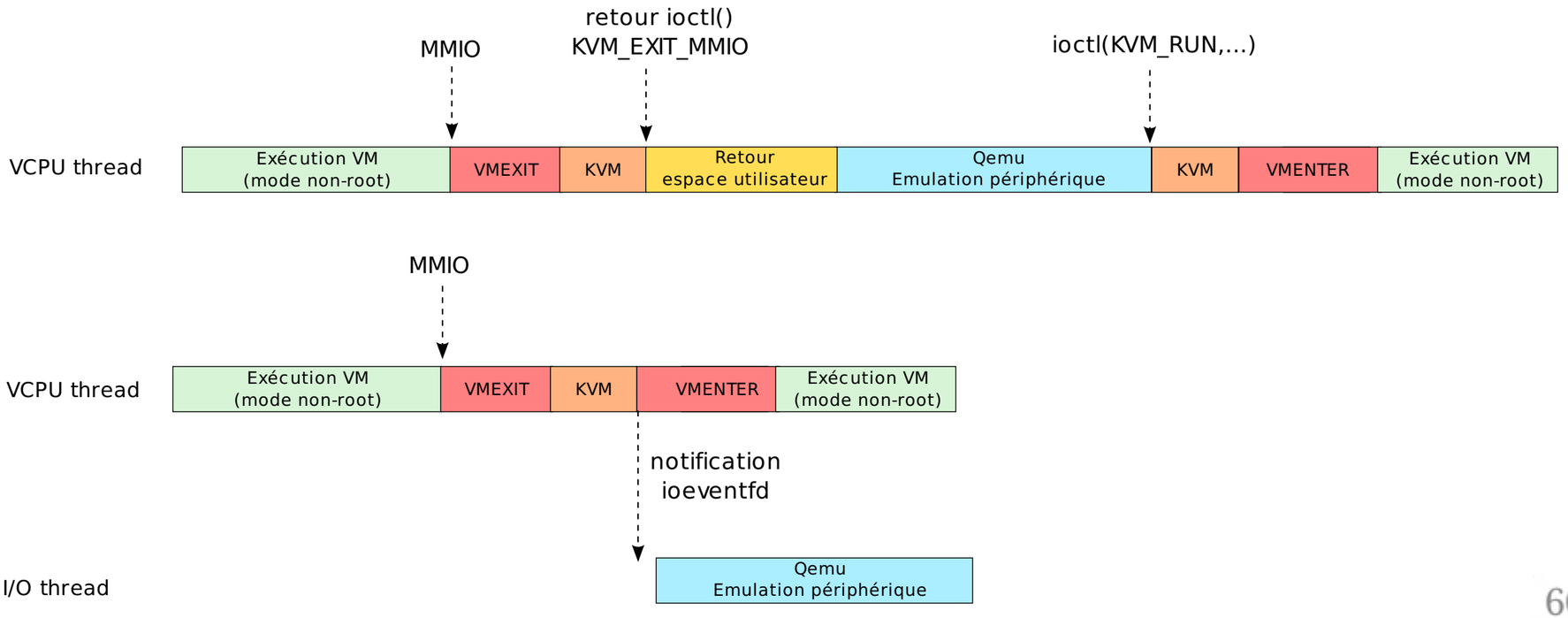
- Bascule invité vers espace utilisateur hôte très coûteuse
- Émulation synchrone dans le thread du vCPU



Minimisation des changements de contexte

Interfaces KVM irqfd et ioeventfd

- **ioeventfd**: descripteur de fichier lié à un registre MMIO
  - Evite au thread VCPU de retourner en espace utilisateur hôte
  - Permet un traitement parallele dans I/O thread dédié à chaque périphérique
- **irqfd**: descripteur de fichier lié à un vecteur d'interruption
  - Permet de déclencher une interruption depuis différents contextes

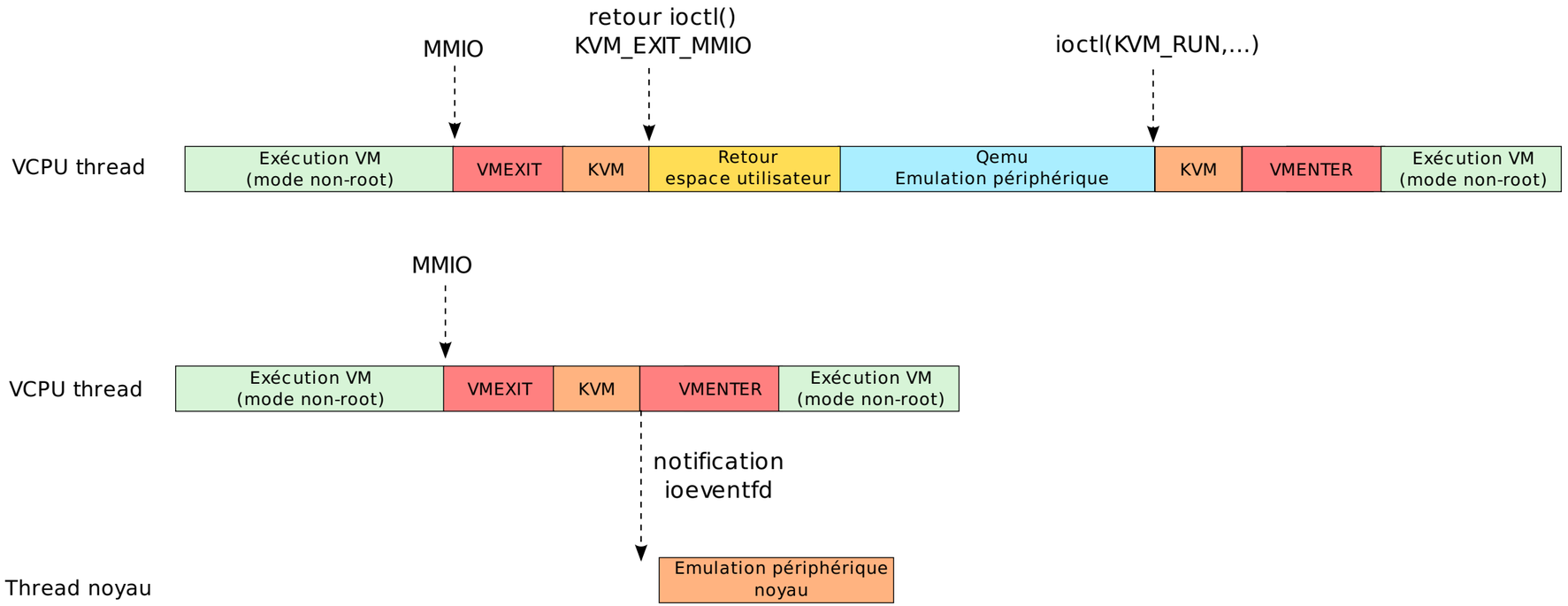


Minimisation des changements de contexte

vhost

Émulation du périphérique en espace noyau hôte

- Equivalent d'un I/O thread en mode noyau
- Adapté à la simplicité d'un périphérique para-virtualisé



PCI Passthrough

Principes

- Accès direct d'une VM à un périphérique physique
  - Pas réellement de la virtualisation
  - Assurer l'isolation de la VM
  - Permettre au périphérique physique de comprendre les gPA
- Nécessite une IOMMU
  - Redirection DMA
    - Redirection des adresses DMA accèdées par un périphérique
    - Mise en place d'une table gPA -> hPA
    - Nécessite de punaiser la mémoire de la VM
  - Redirection d'interruption
    - Redirection des vecteurs d'interruption déclenchés

PCI Passthrough

VFIO

- Module noyau permettant d'appliquer une IOMMU à un périphérique
  - Utilisé par Qemu
  - Configuration d'un adressage cohérent avec celui défini pour KVM
- Accès aux registres via mmap
  - Virtualisation du PCI config space
  - Accès direct aux autres régions
  - Integration à l'espace d'adressage de KVM
- Redirection des interruption dans un eventfd
  - Injection dans un irqfd KVM

PCI Passthrough

Posted interrupts

- Supporté par les dernières générations de CPU
- IOMMU capable de rediriger des interruption vers des CPU virtuels
    - Extension du VMCS permettant l’injection d’interruption
    - Redirection d’interruption pour intéragir avec ce champ
    - Injection de l’interruption sans VMEXIT si le vCPU est actif

PCI Passthrough

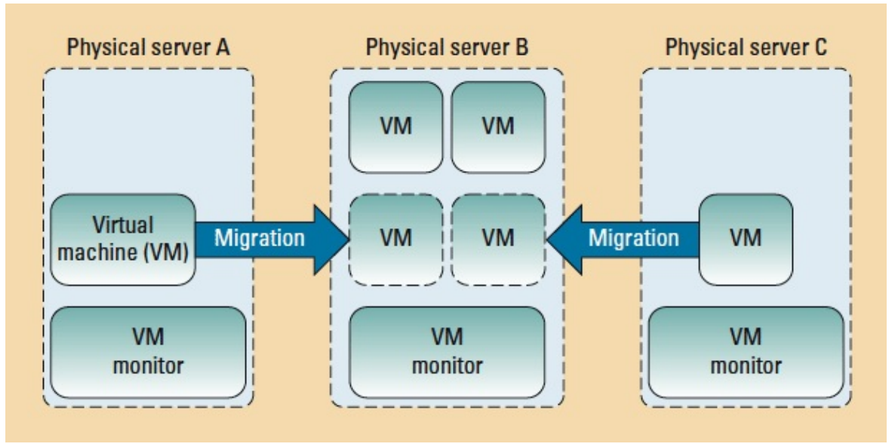
Single Root I/O Virtualization (SR-IOV)

- Multiplexage matériel d'un périphérique en plusieurs fonctions virtuelles (VF)
  - Assignation de chaque VF à une VM via PCI Passthrough
- Une fonction physique reste en charge de la configuration de la carte
  - Configurations communes
  - Gestion des VFs
    - Creation / destruction
    - Configuration
    - Mécanismes d'isolation (exemple: VLANs autorisés)
  - Gestion énergétique…
- Exemple: Mellanox Infiniband
  - Jusqu'à 95 VFs par carte physique sur les cartes récentes
  - Possibilité d'assigner de restreindre chaque VF à une liste de PKey
  - Performance équivalentes aux performances natives

Migration de VM à chaud

Description

- Déplacer une VM d'un hôte à l'autre sans disruption de service
  - Les applications restent fonctionnelles
  - Les connexions réseaux restent établies
- Applications
  - Equilibrage de charge entre hyperviseurs
  - Maintenance sur un hyperviseur



Migration de VM à chaud

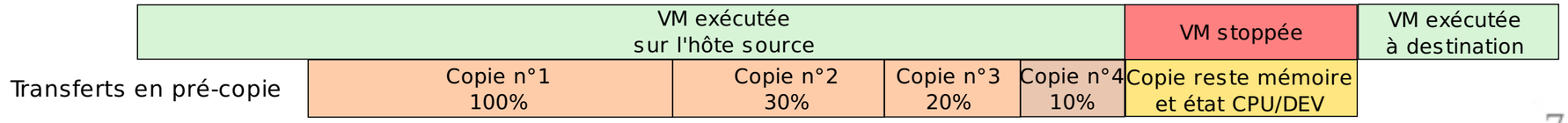
Etats à transférer

- État des périphériques de stockage
  - Potentiellement plusieurs TB
  - Utilisation d'un stockage réseau accessible depuis tous les hyperviseurs
  - Pas de migration nécessaire
- État des vCPUs et autres périphériques
  - Quelques MB: transférés en quelques millisecondes
  - Une indisponibilité pendant la durée du transfert est acceptable
    - Suffisament court pour être invisible
  - Certains périphériques peuvent être impossibles à migrer
    - PCI-Passthrough
- État de la mémoire
  - Quelques (dizaines) de GBs: transférés en quelques (dizaines) de secondes
  - Coupure trop longue pour assurer une continuité de service
  - Nombreux échecs de type *timeout* à prevoir
  - Doît être transférée à chaud

Migration de VM à chaud

Pré-copie de la mémoire

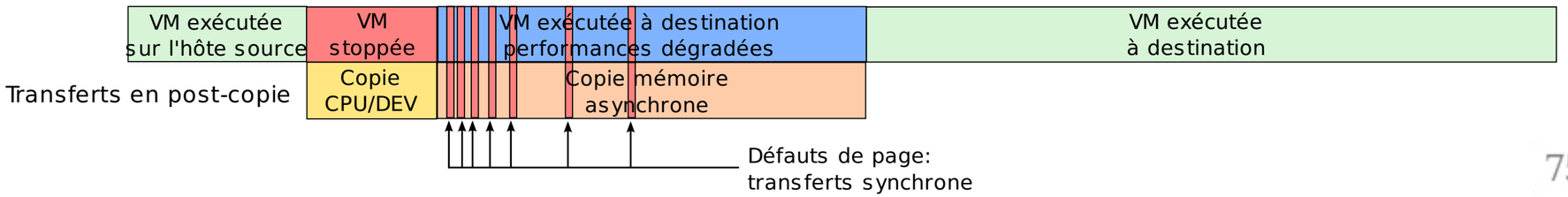
- Transfert préalable de la mémoire vers la destination
  - Pendant que la VM est active sur la source
    - Continue de modifier sa mémoire en parallèle
  - A la fin du transfert, les données transférées ne sont plus à jour
- Multiples passes de transfert
  - Copie des données modifiées depuis le début de la denière passe
  - Arrêt selon critères de convergence (qté de données restante)
- Fin de la migration
  - Suspension de la VM sur l'hôte source
  - Transfert du reste de la mémoire, état CPU et périphériques
  - Démarrage de la VM sur l'hôte de destination



Migration de VM à chaud

Post-copie de la mémoire

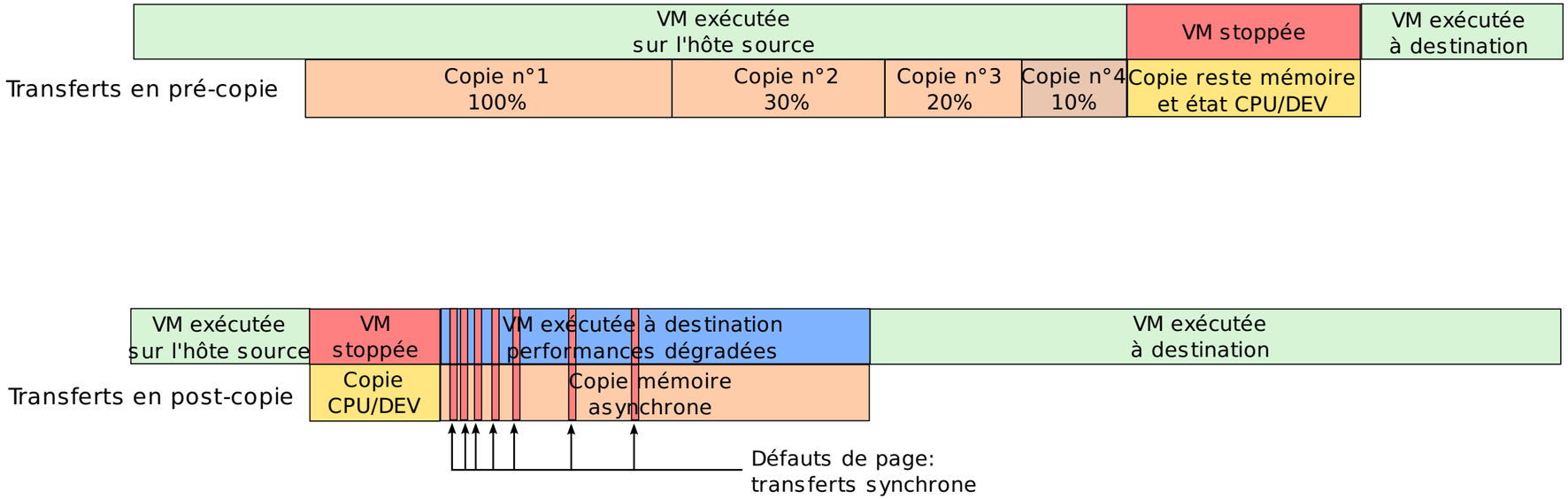
- Dès le début de la migration
  - Suspension de la VM sur l'hôte source
  - Transfert de l'état du CPU et périphériques
  - Démarrage de la VM sur l'hôte de destination
- La VM n'a pas accès à sa mémoire intialement
  - Accès à une page manquante: défaut de page déclenchant un VMEXIT
    - Transfert synchrone de la page manquante depuis l'hôte source
  - Transfert parallèle en tâche de fond de toute la mémoire



Migration de VM à chaud

Comparaison des méthodes de migration

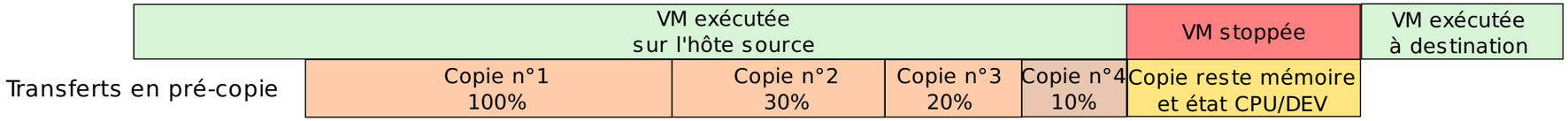
- Durée d'indisponibilité
  - Pré-copie: dépend du taux d'accès mémoire
  - Post-copie: durée courte et prévisible

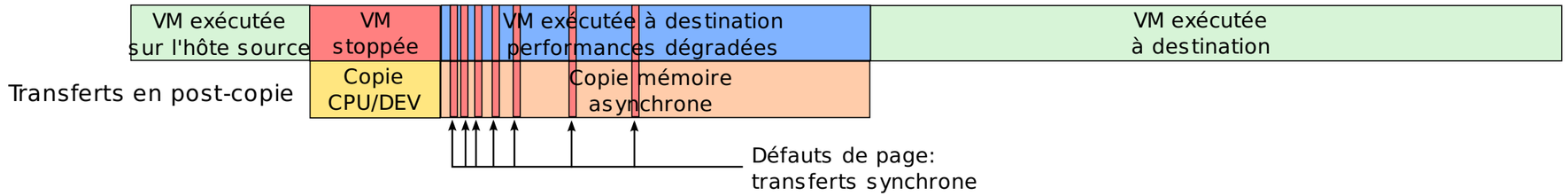


Migration de VM à chaud

Comparaison des méthodes de migration

- Délai de migration
  - Pré-copie: dépend du taux d'accès mémoire
  - Post-copie: rapide

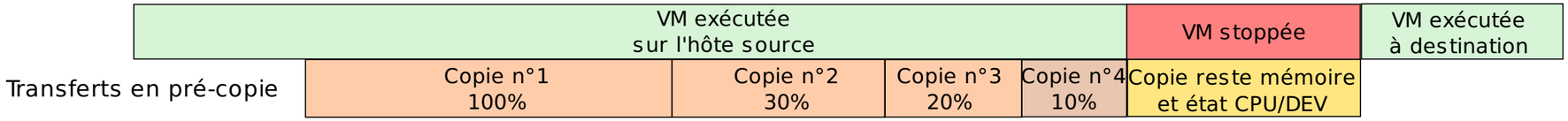


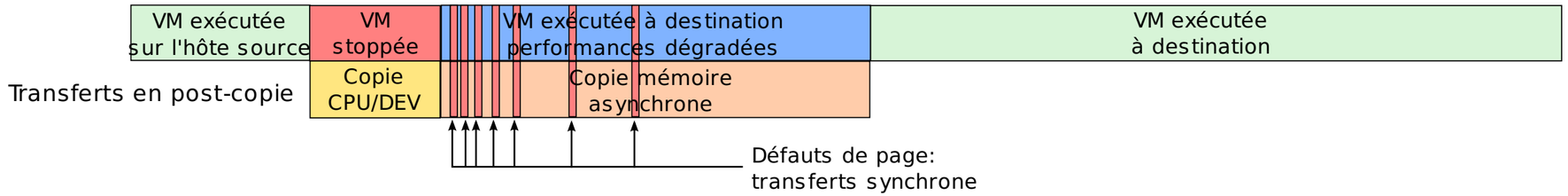


Migration de VM à chaud

Comparaison des méthodes de migration

- Quantité de données transférée
  - Pré-copie: dépend du taux d'accès mémoire
  - Post-copie: minimal

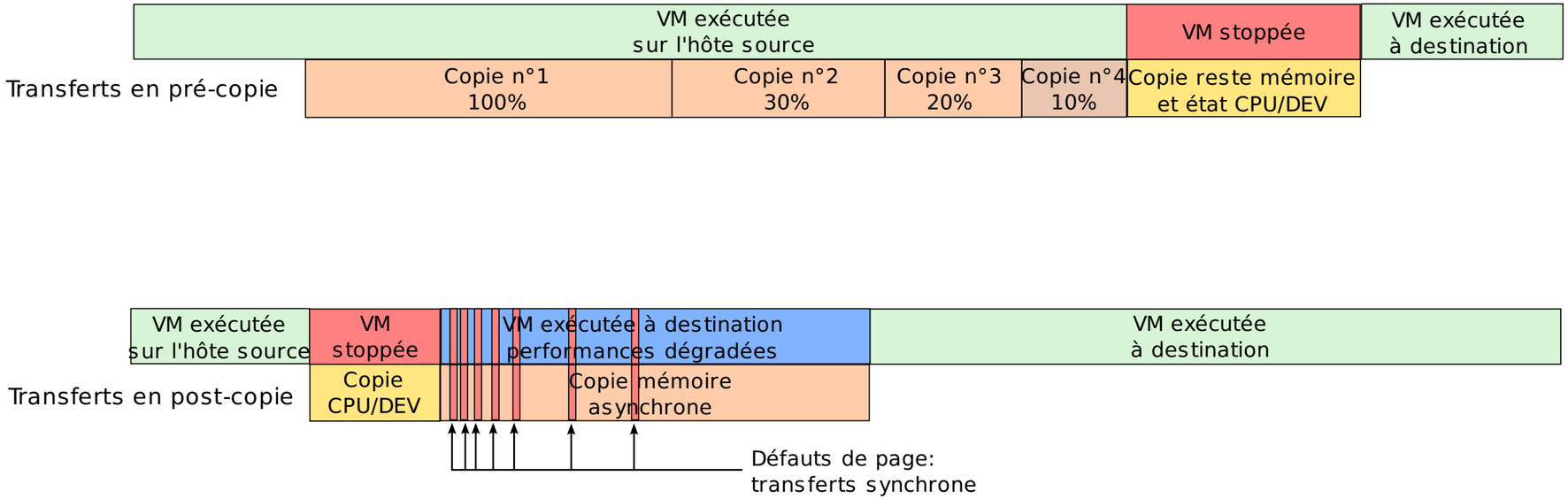




Migration de VM à chaud

Comparaison des méthodes de migration

- Performances pendant la migration
  - Pré-copie: impact modéré
  - Post-copie: possibilité de fort ralentissement, nécessite un réseau performant



Libvirt

- Bibliothèque de gestion de la virtualisation
  - Supporte les principaux hyperviseurs
    - Qemu, Xen, HyperV, VMware ESX, …
  - Permet aussi de gérer des conteneurs
    - LXC, OpenVZ
  - API stable et commune aux hyperviseurs
  - Permet d'écrire des applications portables
  - Gestion d'un seul noeud hôte
    - Pas de notion de clustering
- Exemples de fonctionnalités supportées
  - Définition, lancement, arrêt de domaines (VMs, conteneurs)
  - Sauvegarde, Migration
  - Limitation et suivi des ressources (mémoire, cpu …) utilisées
  - Gestion des périphériques, ajout/suppresion à chaud
  - Réseau (réseaux virtuels, connexion à un réseau physique, filtrage …)

Libvirt

Schéma d'architecture

- Les applications clientes se lient à la bibliothèque
  - virsh: outil ligne de commande
  - virt-manager: outil graphique
  - libguestfs: outil de gestion d'image de VMs
- Communication locale ou distante vers un demon libvirtd