Abyme : un voyage au cœur des hyperviseurs récursifs

Benoît Morgan, **Guillaume Averlant**,
Vincent Nicomette, Éric Alata

LAAS-CNRS, INSA Toulouse, Université Toulouse III

2 juin 2015

Plan

Contexte des travaux

Projet SVC – *Secured Virtual Cloud*

- Projet Investissement d'Avenir
  Itrust, *Bull*, *Eneed*, *Secludit*, *Eurogiciel*, *Val Informatique*, *Blue Mind*, *LAAS-CNRS*, *IRIT*
- Coordinateur du projet : Bull

Contributions du LAAS – 3 thèses

- Évaluation de la sûreté de fonctionnement dans le *cloud*
- Évaluation et analyse de la sécurité dans le *cloud*
- **Sécurisation de gestionnaires de machines virtuelles**

Principe de séparation des privilèges

- Tendance générales des noyaux monolithiques
  - Xen, KVM, VM Ware, Linux, NT
  - Gestion des périphériques, scheduling, gestion mémoire au même niveau de privilèges
  - Compromission d'un des composants du noyau $\Rightarrow$ Contrôle du système
- Nécessité de séparation des fonctionnalités par niveau de criticité
- Nombre de niveau de privilège limité par le matériel
  - Limitation des rings : seul le ring 0 contrôle les niveaux supérieurs

  $\Rightarrow$ Les anneaux étendus par la virtualisation ?
- Hyperviseurs existants pas forcément adaptés à être "empilés"
  - Implémentation d'un hyperviseur récursif

La virtualisation récursive

Intérêts scientifiques et stratégiques

- Une fonction de sécurité par niveau de virtualisation
- Fonction de sécurité spécifique à un composant directement intégré dans sa ROM d'extension
- Développement d'une solution globale par des acteurs séparés

Défis techniques

- Peut-on utiliser le même code pour les différents niveaux ?
- Quel impact sur les performances (combien de niveaux peut-on enchaîner) ?
- Les structures de données associées à la virtualisation récursive : quelle complexité ?

Plan

Machines virtuelles

Définition

A virtual machine is taken to be an efficient, isolated duplicate of the real machine. We explain these notions through the idea of a virtual machine monitor (VMM) [...] the virtual machine monitor provides an environment for programs which is essentially identical with the original machine ; second, programs run in this environment show at worst only minor decreases in speed ; and last, the VMM is in complete control of system resources.

"Formal Requirements for Virtualizable Third Generation Architectures" G. Popek, R. Goldberg

Choix technologiques pour l'hyperviseur

- Full virtualization : généricité et faible dépendance inter-niveaux
- Bare metal : meilleures performances
- Extensions d'assistance matérielle à la virtualisation

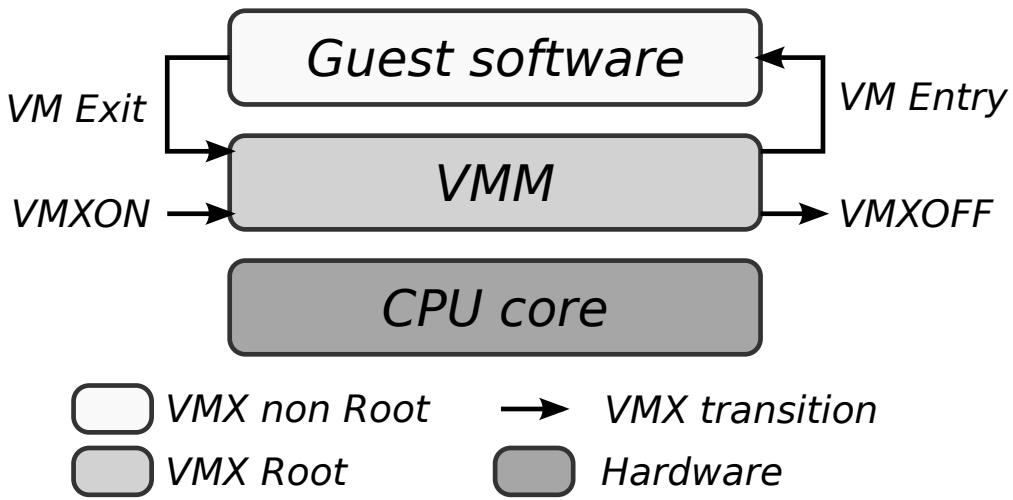
- Choix d'Intel : VT-x, VT-d (DMA/IRQ Remapping)

$\Rightarrow$ représentatif de la réalité du marché

- Autres technologies de fondeurs
  - ARM : Virtualization extensions
  - AMD : AMD-V

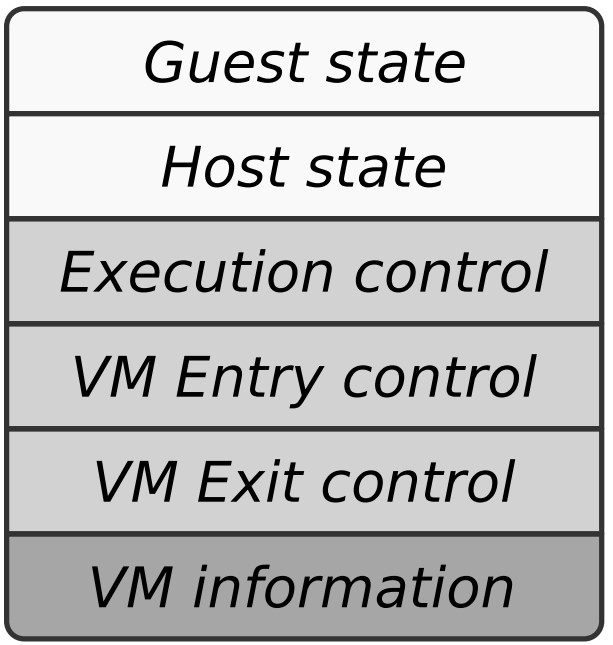
Intel VT-x

- Terminologie Intel
  - Virtual Machine Monitor (VMM) : contrôle
  - Guest software : exécuté dans un cœur virtuel
- Modèle de contrôle identique à celui des OS
- 2 modes d'exécution supplémentaires



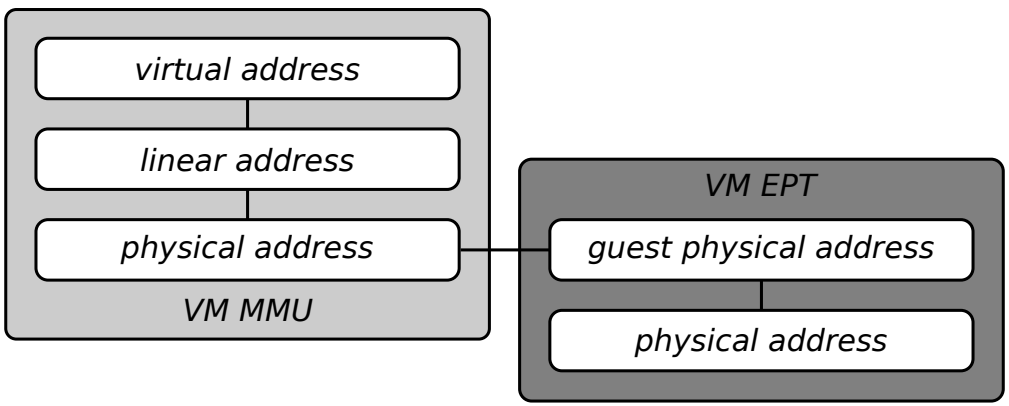
VMCS

- Virtual Machine Control Structure
- Contrôle des cœurs virtuels
- 1 par cœur physique
- VMCS Region : cache mémoire de VMCS



Virtualisation de la MMU

- Extended Page Tables (EPT) : Niveau supplémentaire de virtualisation de la mémoire contrôlé par le VMM

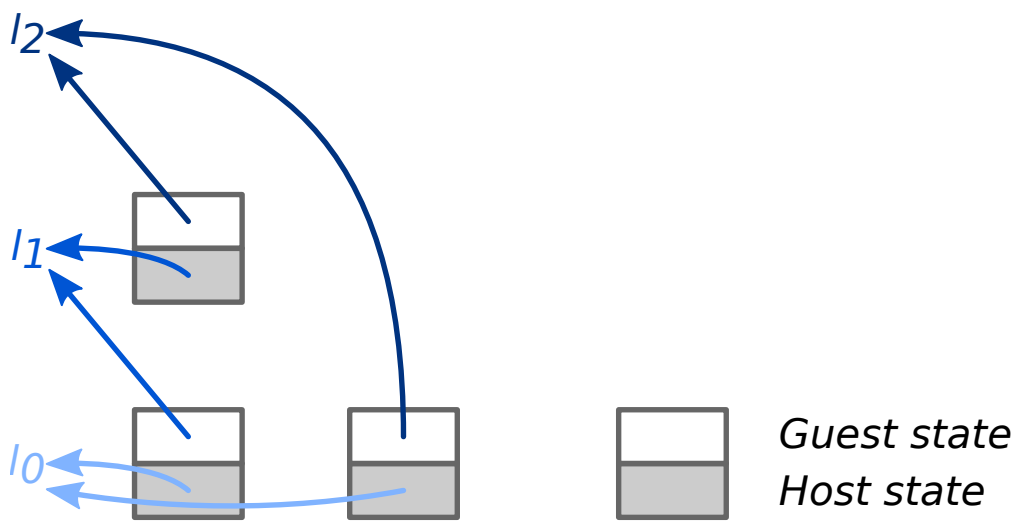


Intel VMX

- 13 instructions de gestion de VT-x
- Entrée et sortie de VMX Operation
  - `VMXON`, `VMXOFF`
- Gestion de la VMCS
  - `VMCLEAR`, `VMPTRLD`, `VMPTRST`, `VMREAD`, `VMWRITE`
- Ordonnancement des VMs
  - `VMLAUNCH`, `VMRESUME`
- Paravirtualisation
  - `VMFUNC`, `VMCALL`
- Gestion des caches MMU
  - `INVVPID`, `INVEPT`

Virtualisation d'un hyperviseur

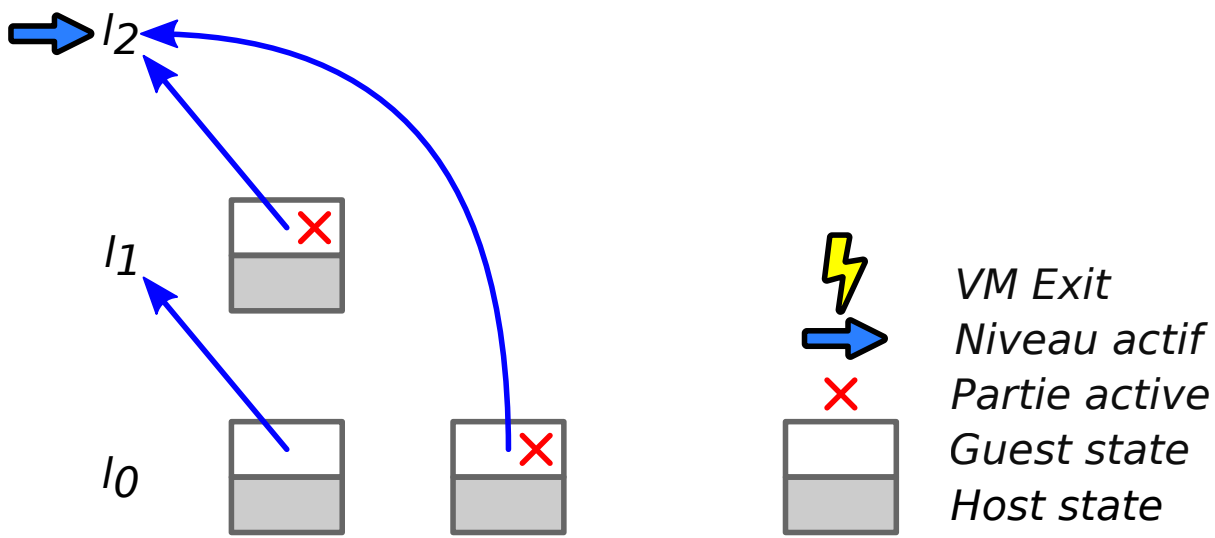
- Réduction des privilèges de l'hyperviseur virtualisé
  - 2 VMCS : pour l'hyperviseur virtualisé (l_1) et sa VM (l_2)
  - Guest state : l_1 ou l_2
  - Host state : Hyperviseur hôte (l_0)



- Instruction VMX exécutée par l'hyperviseur Guest ⇒ VM Exit

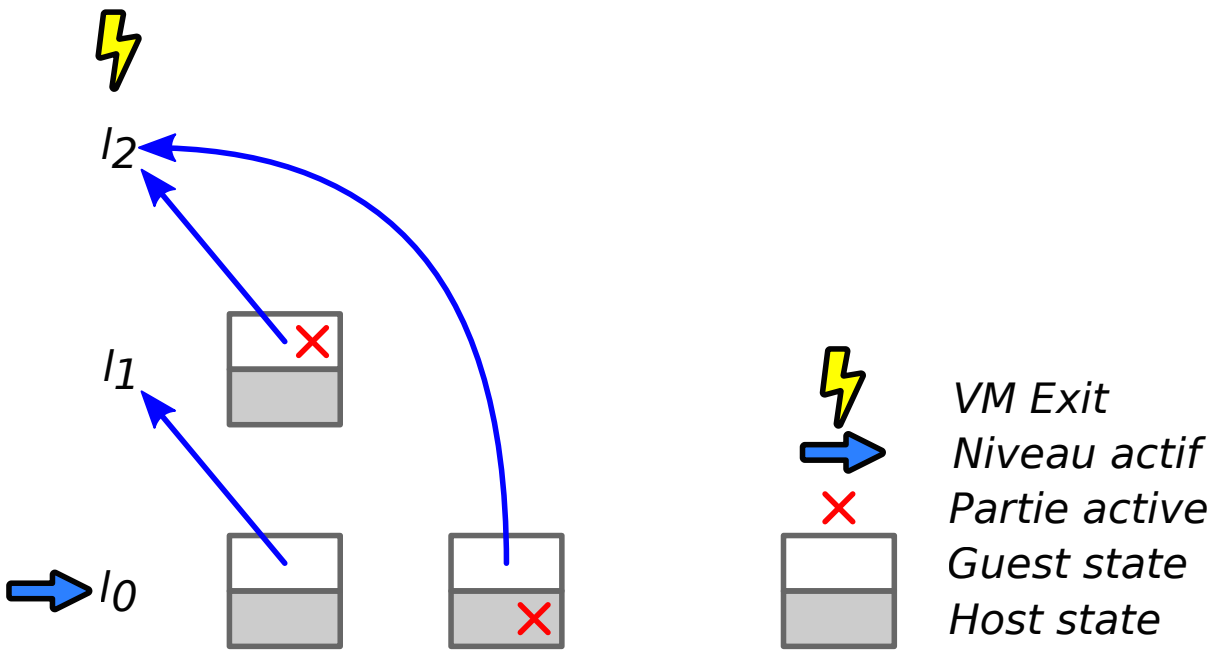
Virtualisation d'un hyperviseur

- VM Exit durant l'exécution de l_2



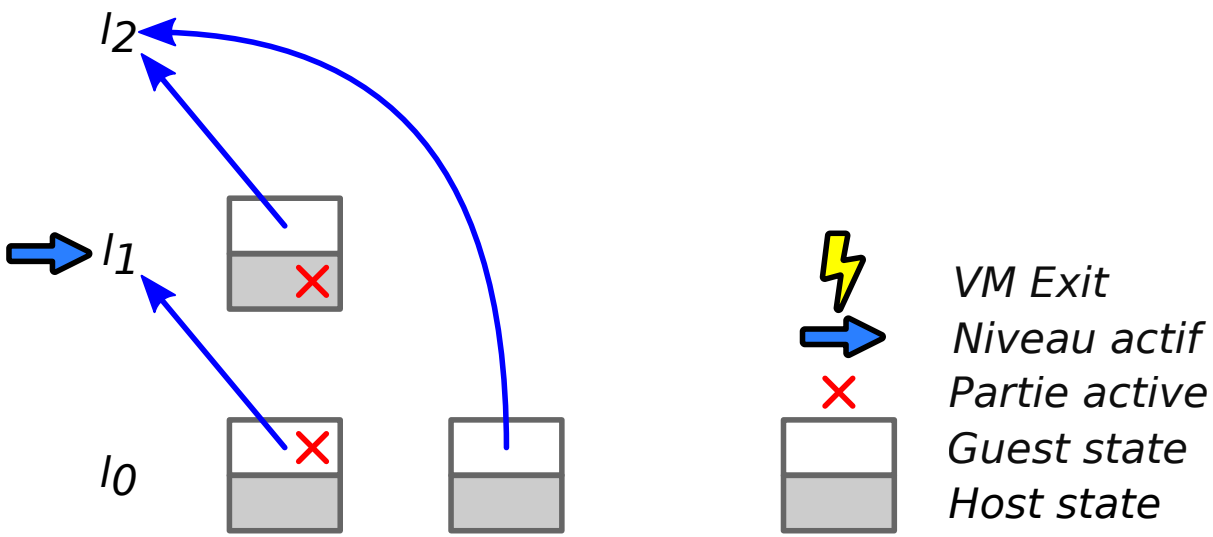
Virtualisation d'un hyperviseur

- VM Exit au niveau l_2



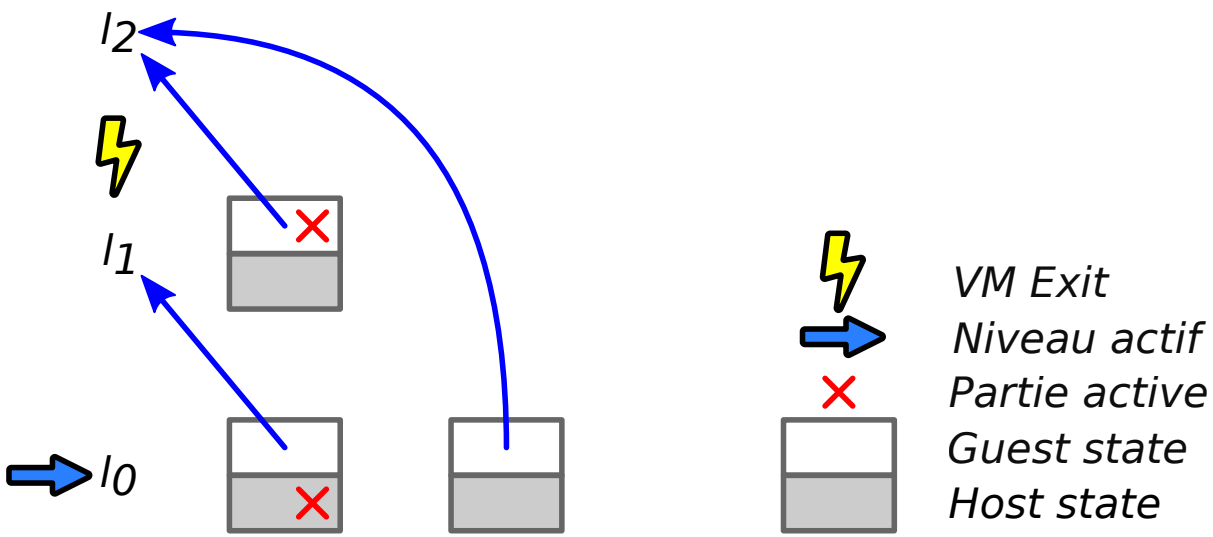
Virtualisation d'un hyperviseur

- Notification du VM Exit au niveau supérieur l_1



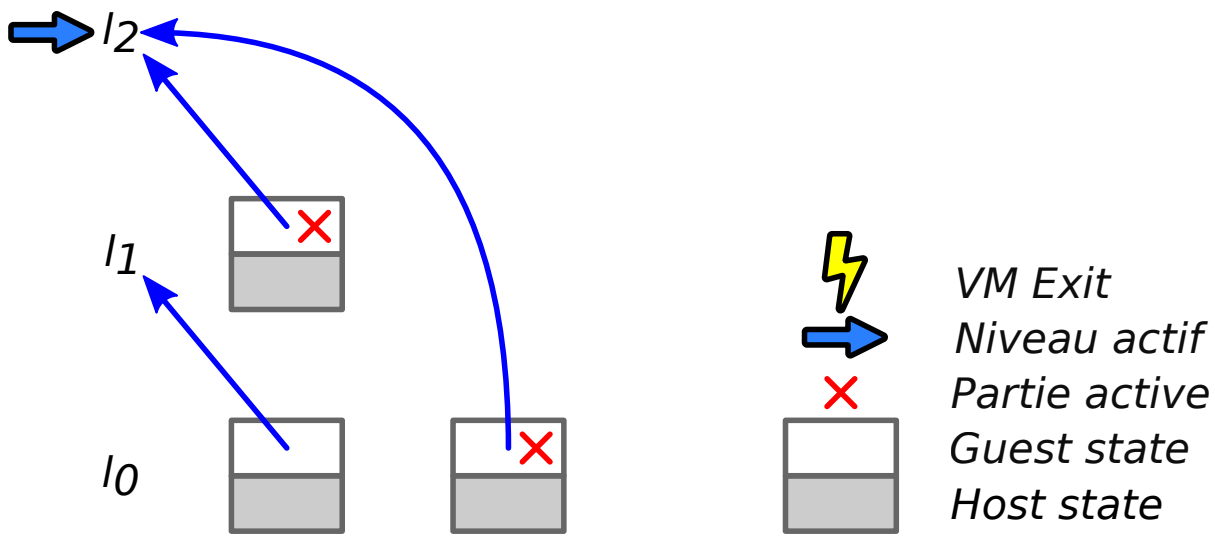
Virtualisation d'un hyperviseur

- Exécution de `VMRESUME` par $l_1 \Rightarrow$ VM Exit



Virtualisation d'un hyperviseur

- Exécution du `VMRESUME` de l_2 demandé par l_1



Émulation des instructions VMX

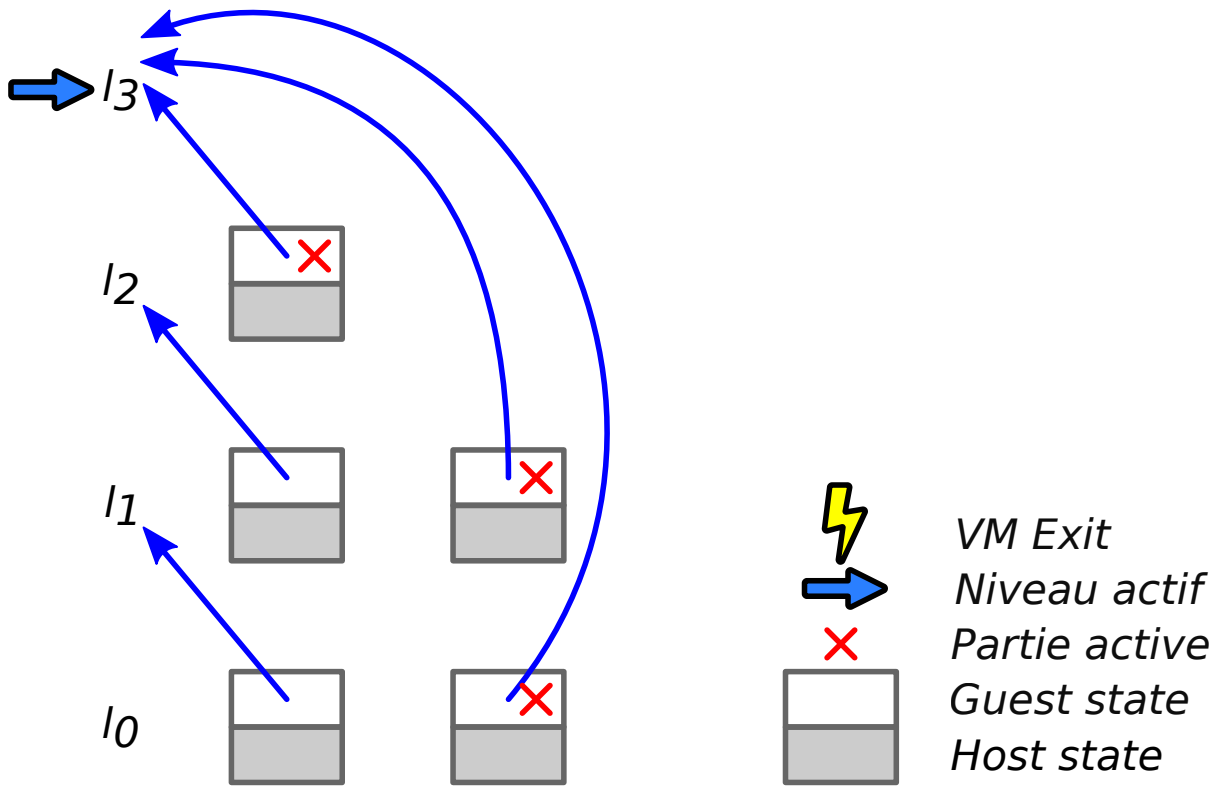
Principales instructions

- `VMXON` : Rien à faire.
- `VMPTRLD` : Sauvegarde du pointeur de VMCS region $\Leftrightarrow$ shadow VMCS (S)
- `VMWRITE`/`VMREAD` : Chargement de S, `VMWRITE`/`VMREAD` et rechargement de la VMCS en cours
- `VMRESUME`, `VMLAUNCH` : Copie de la partie Guest de S dans une VMCS interne et lancement ou reprise de celle-ci

Stratégie de virtualisation récursive

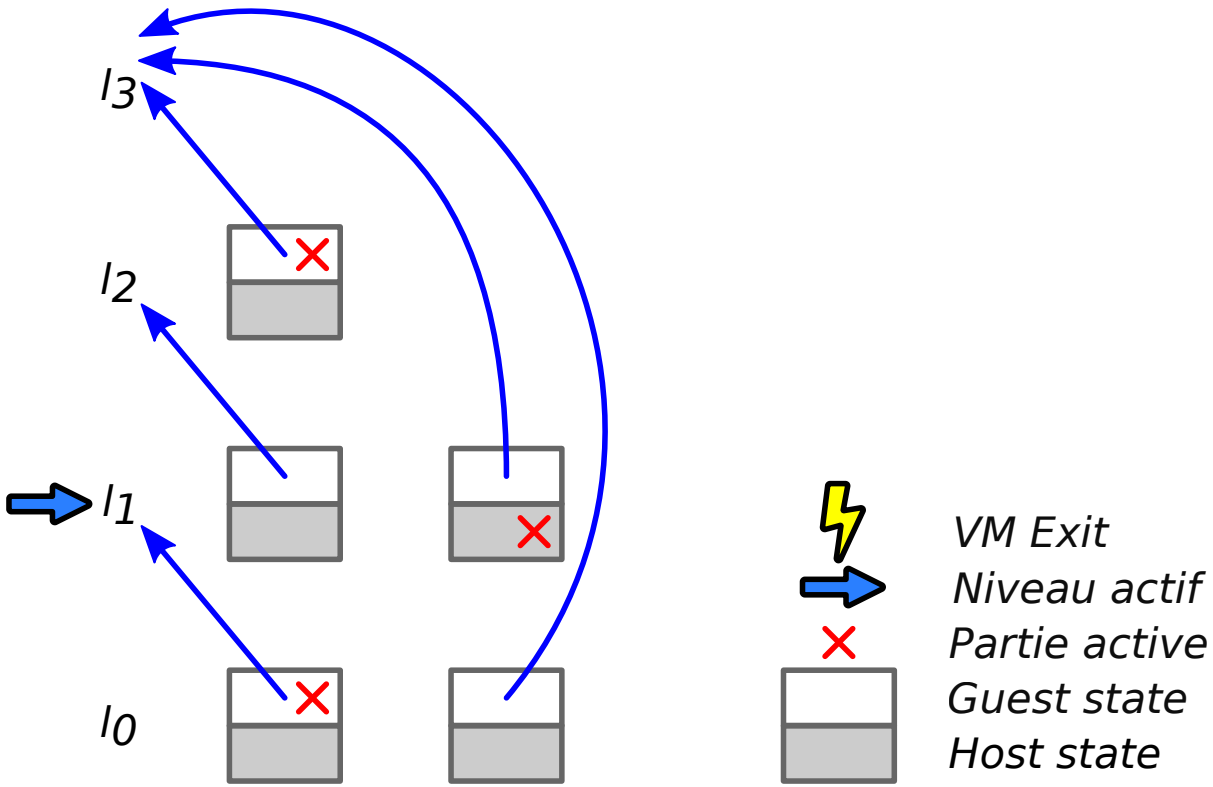
- Tout VM Exit généré par l_x doit être traité par l_{x-1}
- Les hyperviseurs $l_k | k < x$ délègueront au niveau supérieur

$\Rightarrow$ Aucun évènement n'est manqué par chaque niveau d'hyperviseur



Stratégie de virtualisation récursive

- Un VM Exit de l_3 à été notifié à l_1

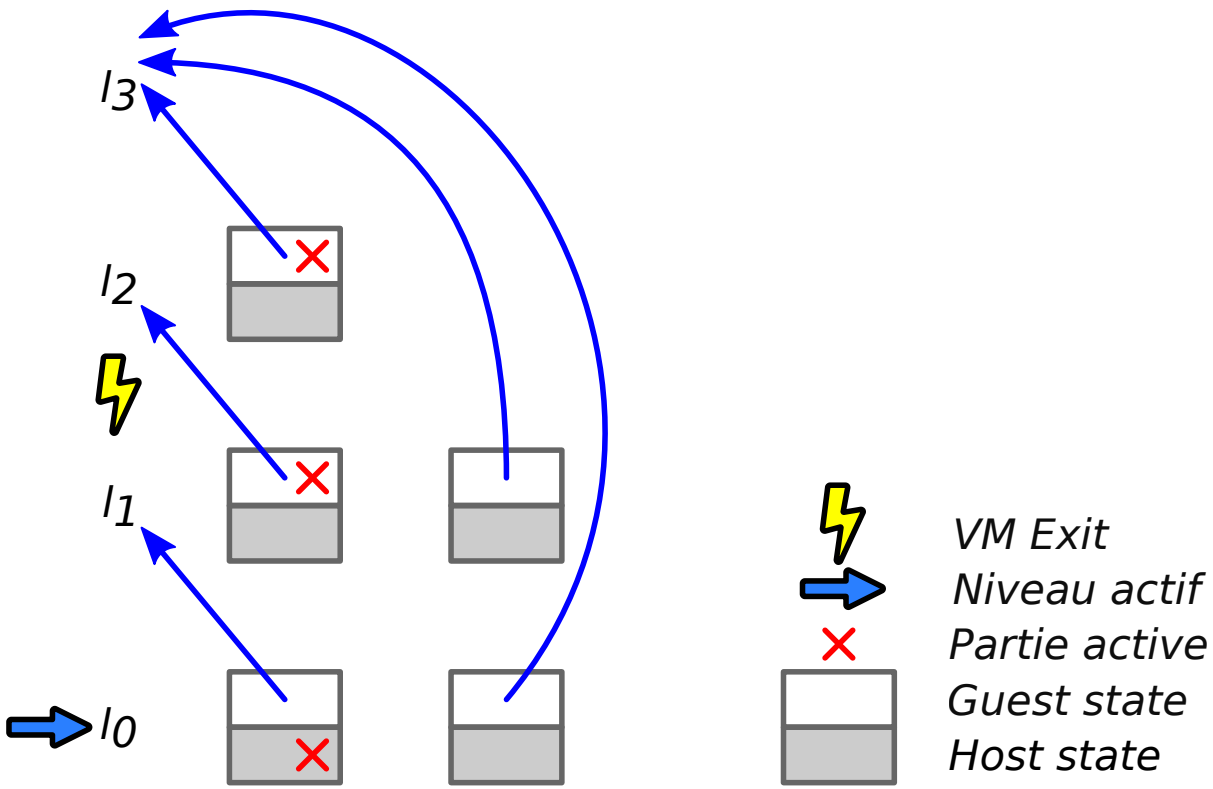


Stratégie de virtualisation récursive

- l_1 n'est pas responsable du VM Exit de l_3

$\Rightarrow$ l_1 donne la main à l_2 via un `VMRESUME`

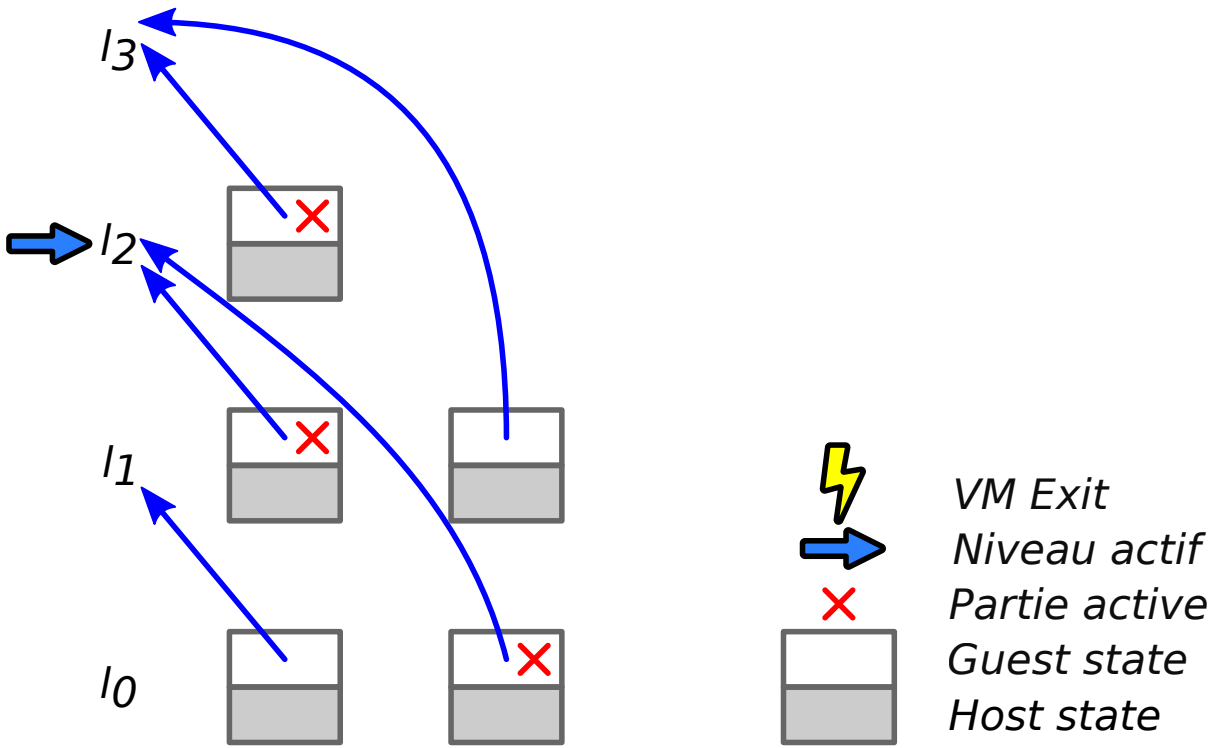
$\Rightarrow$ Génération d'un VM Exit



Stratégie de virtualisation récursive

- Le VM Exit associé au VMRESUME de l_2 est traité par l_0

$\Rightarrow$ l_0 donne la main à l_2 en reconfigurant la partie Guest de sa VMCS



Stratégie de virtualisation récursive

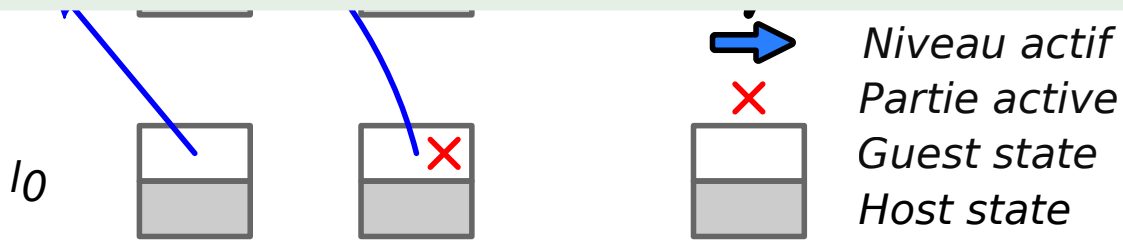
- Le VM Exit associé au VMRESUME de l_2 est traité par l_0

$\Rightarrow$ l_0 donne la main à l_2 en reconfigurant la partie Guest de sa VMCS

Hypothèse

- Plus le nombre de niveaux augmente, plus le nombre de transitions augmente
- Peut-on évaluer la complexité de cette approche ?

$\Rightarrow$ Modélisation du comportement de l'hyperviseur



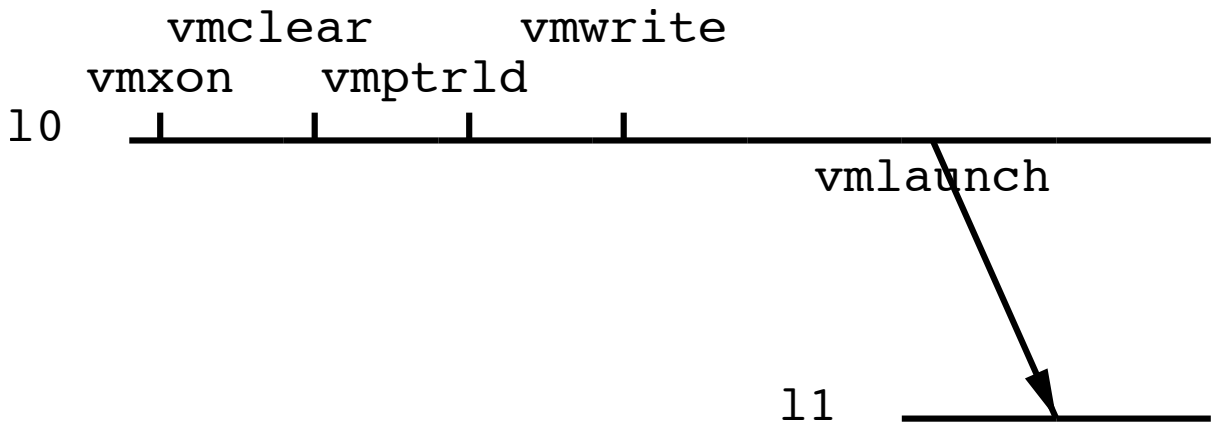
Modélisation simplifiée de l'hyperviseur

- Modèle simple proche du matériel
- Implémentation en Prolog
- Génération de graphes de traces d'exécution

$\Rightarrow$ Évaluation de la complexité du nombre de transitions en fonction du nombre d'hyperviseurs

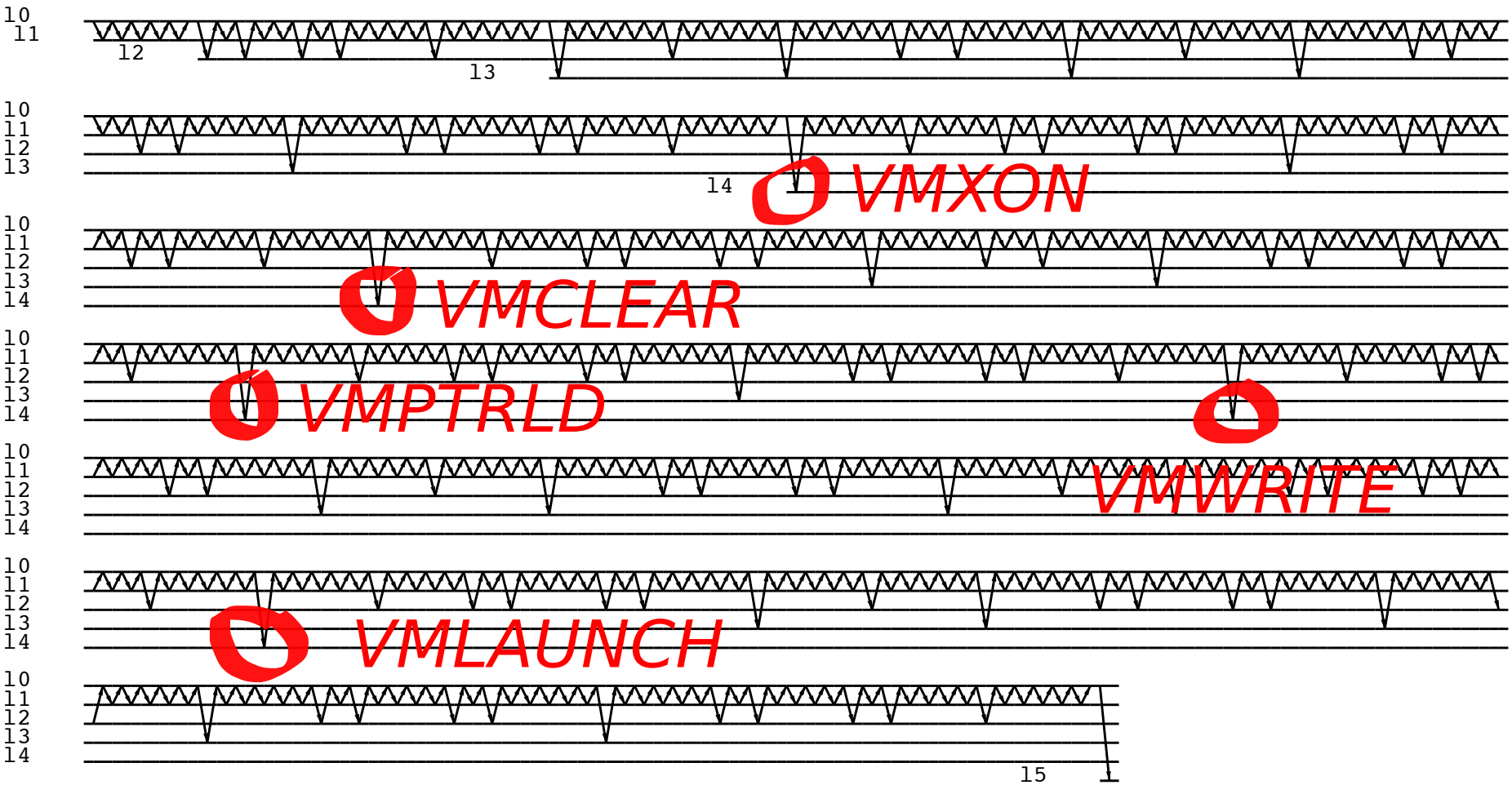
$\Rightarrow$ Validation de la stratégie théorique de virtualisation

Démarrage d'un hyperviseur



Démarrage de cinq hyperviseurs

Démarrage de cinq hyperviseurs



Démarrage de cinq hyperviseurs

Constat

- Stratégie de virtualisation donnant le comportement attendu
- Caractère exponentiel du nombre de transitions

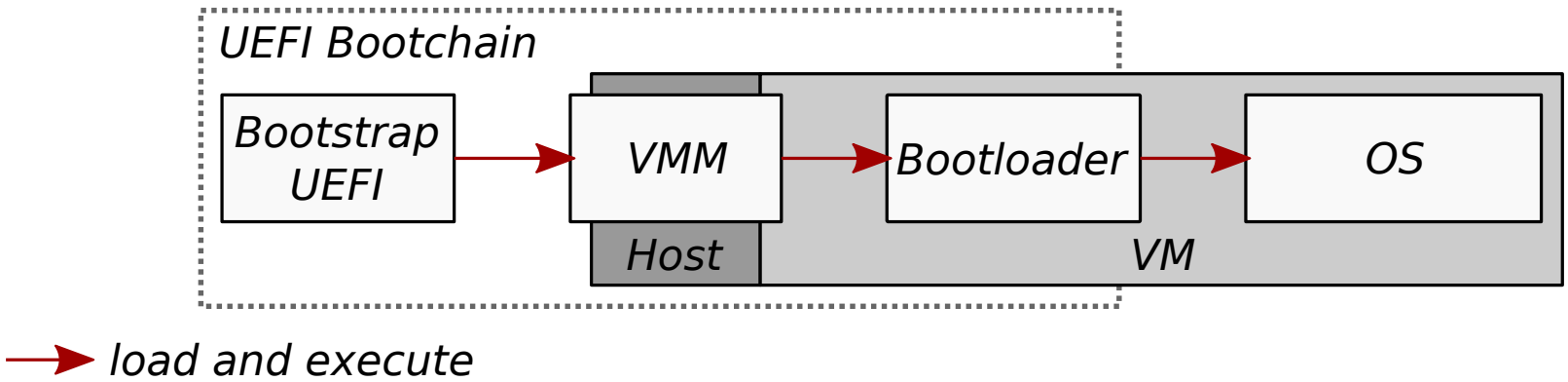
Plan

Chargement et démarrage

- Chargement au plus tôt pour ne pas perdre le contrôle du matériel

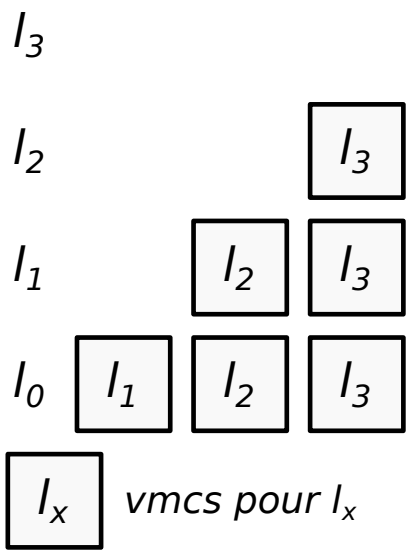
⇒ Driver UEFI

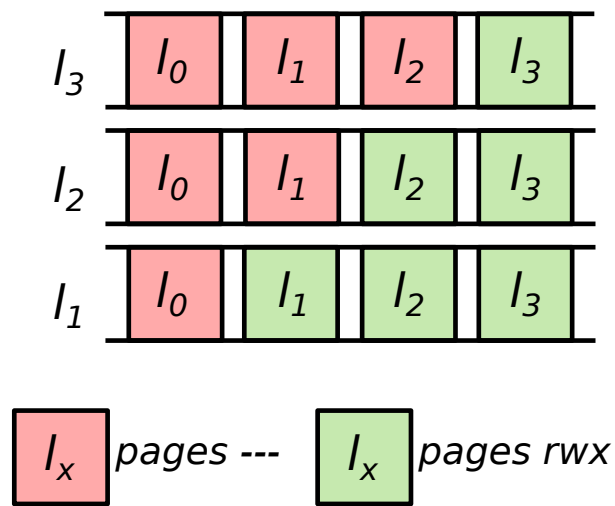
- Chargement de drivers pre boot
- Chargement d'images complexes (PE32+ modifiée)
- Script shell de chargement successifs des hyperviseurs



Traitement de la création de VMs

- Une VMCS par niveau virtualisé
  - ⇒ Évite la copie systématique de certains champs spéficiques et la remise à zéro de la VM entre chaque ordonnancement
- Démarrage d'une VM : `VMLAUNCH` par l_x
  - Association à une VMCS locale
  - Création d'un contexte EPT protégeant par transitivité l_0 à l_x
  - Exemple avec `VMLAUNCH` de l_2





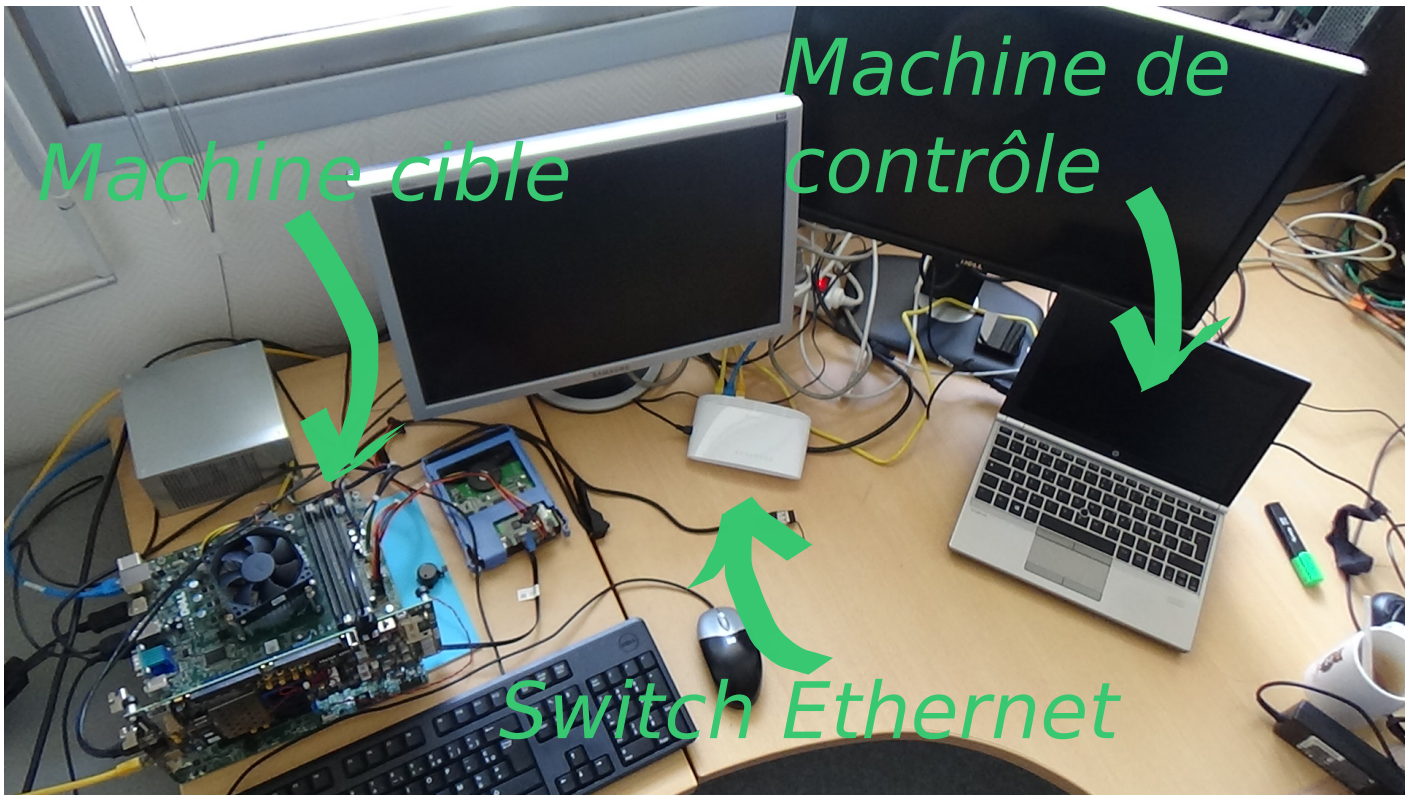
Optimisation du nombre de transitions

- Gestion des interruptions ✗
  - Masquage des interruptions lors des VM Exits
  - IDT configurée pour capturer et ré-injecter les interruptions non masquables
- Utilisation du VMCS shadowing
  - Intéret : Diminution du nombre de transitions lors de `VMREAD` et `VMWRITE` des hyperviseurs virtualisés
  - Bitmaps de configuration des champs non générateurs de VM Exits

$\Rightarrow$ Amélioration des performances

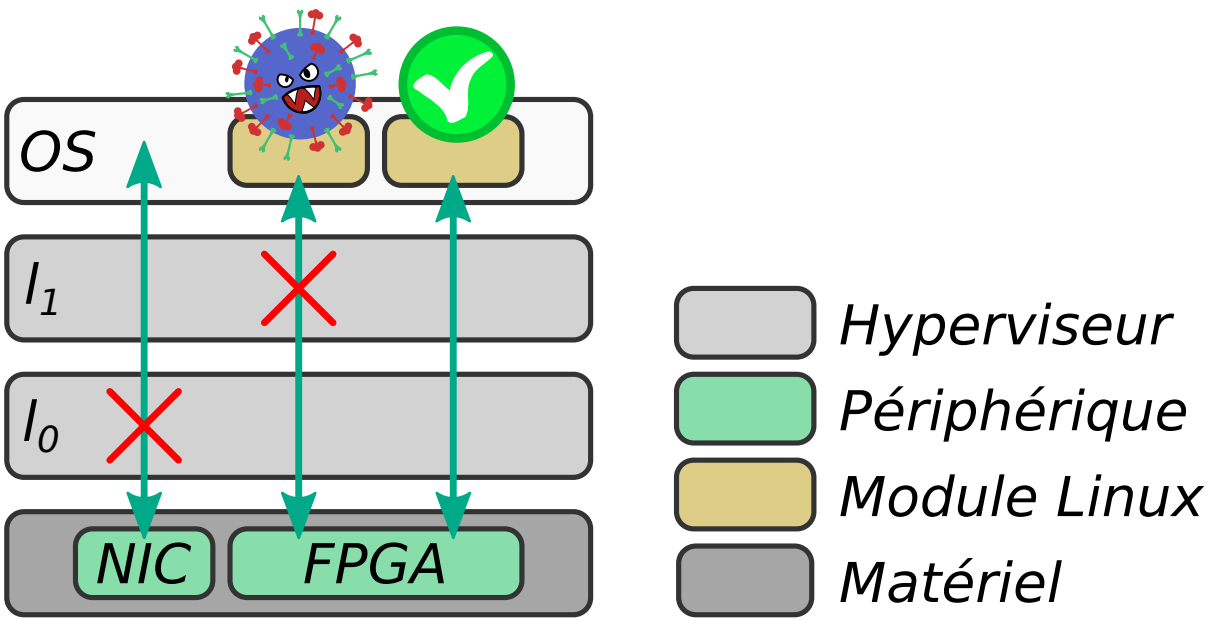
Démo 1

- 3 Hyperviseurs virtualisant une distribution GNU/Linux
- Exécution d'instruction `CPUID` prouvant leur maîtrise du matériel



Démo 2

- Hyperviseur l_0 : protection de la carte Ethernet de debug
- Hyperviseur l_1 : vérification du comportement d'un driver
  - Vérification de la séquence d'initialisation d'un périphérique
  - Si OK autorisation des lectures/écritures dans le BAR0
  - Si NOK blocage des lectures/écritures dans le BAR0



Plan

Conclusion/perspectives

Conclusion

- Implémentation réussie d'un hyperviseur récursif simple
- Pas de dégradation significatif pour un nombre raisonnable de niveaux
- MAIS performances largement en baisse pour beaucoup de niveaux

Perspectives

- Optimisation du nombre de transitions :
  - Pas de propagation des `VMRESUME`

  $\Rightarrow$ Possibilité d'enchaîner plus de niveaux sans dégradation de performances
- Enrichir le modèle avec les informations sur les données des VMCS

Abyme : un voyage au cœur des hyperviseurs récursifs

Benoît Morgan, **Guillaume Averlant**,
Vincent Nicomette, Éric Alata

LAAS-CNRS, INSA Toulouse, Université Toulouse III

2 juin 2015

LAAS-CNRS

Les sources !

- Abyme : `https://homepages.laas.fr/bmorgan/abyme.tgz`
- Démo 1 : `https://youtu.be/NaxOSHUx9GQ`
- Démo 2 : `https://youtu.be/1yz_ZUA2KGM`