Caches collaboratifs noyau adaptés aux environnements virtualisés

Soutenance de thèse

Maxime Lorrillere

4 février 2016

Contexte : du cloud à la virtualisation

Objectif

→ Maximiser l'utilisation des ressources disponibles

Principe de la virtualisation

→ Exécuter plusieurs *machines virtuelles* (VM) sur un seul hôte

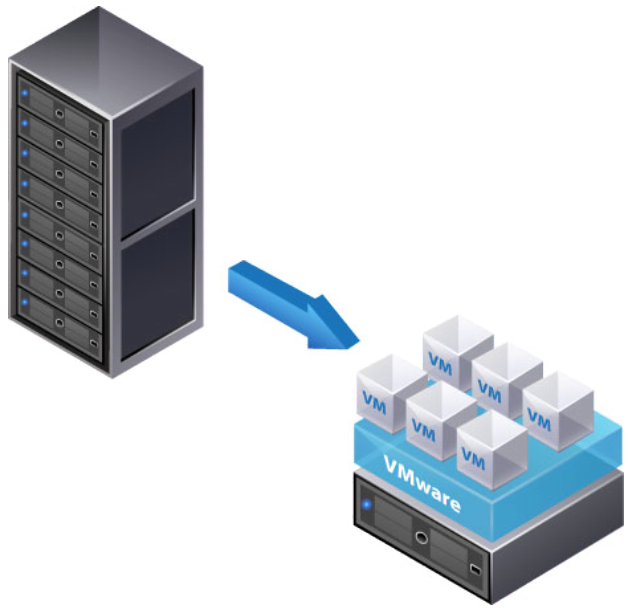
Contexte : du cloud à la virtualisation

Objectif

→ Maximiser l'utilisation des ressources disponibles

Principe de la virtualisation

→ Exécuter plusieurs *machines virtuelles* (VM) sur un seul hôte



Contexte : du cloud à la virtualisation

Objectif

→ Maximiser l'utilisation des ressources disponibles

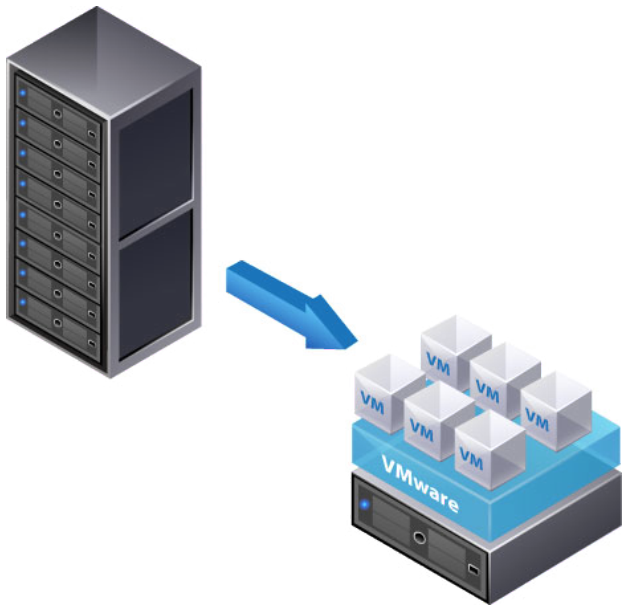
Principe de la virtualisation

→ Exécuter plusieurs *machines virtuelles* (VM) sur un seul hôte

Exemple

NextRadioTV (BFM, RMC, 01net) [OVH]

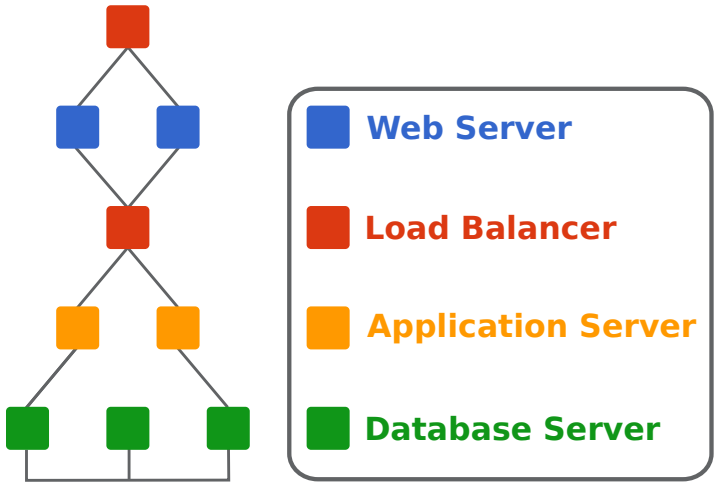
- ~~80 serveurs physiques~~
  - ⇒ 7 serveurs, 80 VMs
- 40 à 50% de réduction de coûts



Intérêts de la virtualisation

Avantages

- Consolidation
- Isolation
- Flexibilité
- Économies d'énergie

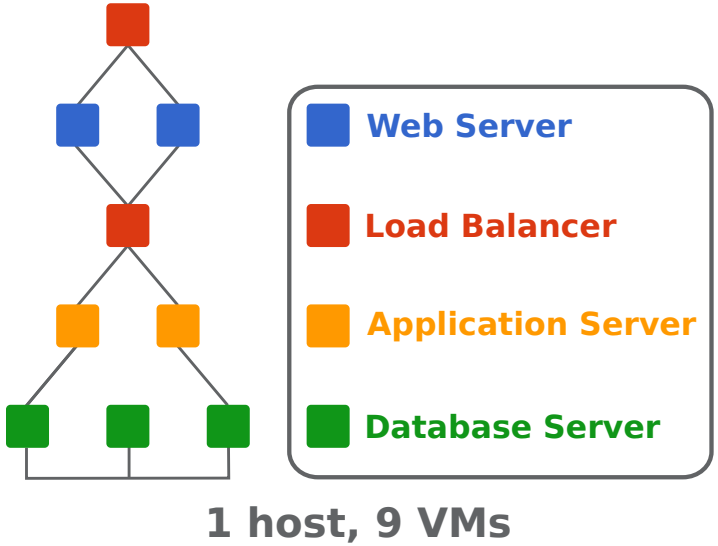


1 host, 9 VMs

Intérêts de la virtualisation

Avantages

- Consolidation
- Isolation
- Flexibilité
- Économies d'énergie



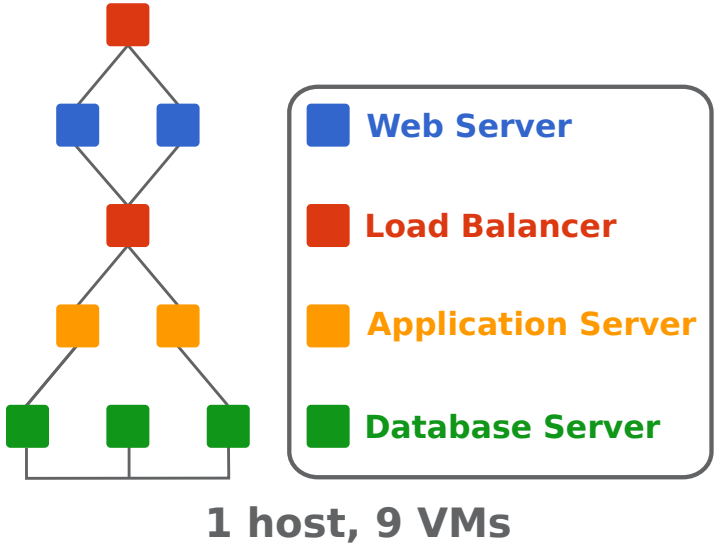
Inconvénients

- Fragmentation des ressources disponibles

Intérêts de la virtualisation

Avantages

- Consolidation
- Isolation
- Flexibilité
- Économies d'énergie



Inconvénients

- Fragmentation des ressources disponibles

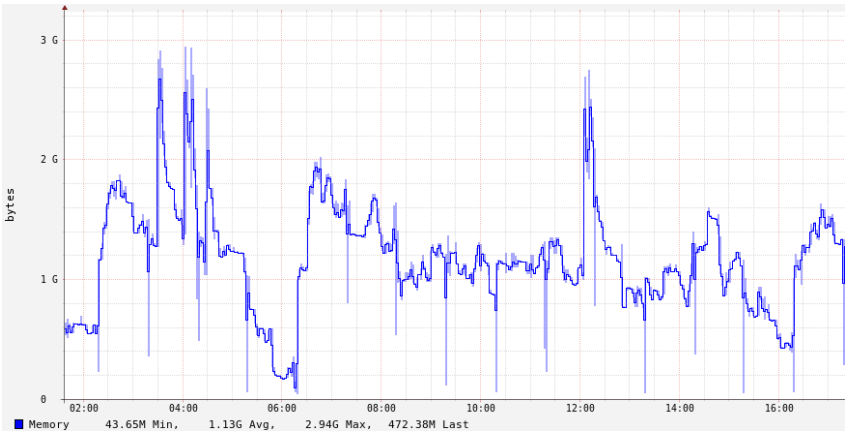
Solutions

- Fragmentation du CPU ⇒ **ordonnanceur**
- Fragmentation du stockage ⇒ **quotas**
- Fragmentation de la mémoire ⇒ ?

Fragmentation de la mémoire : difficultés

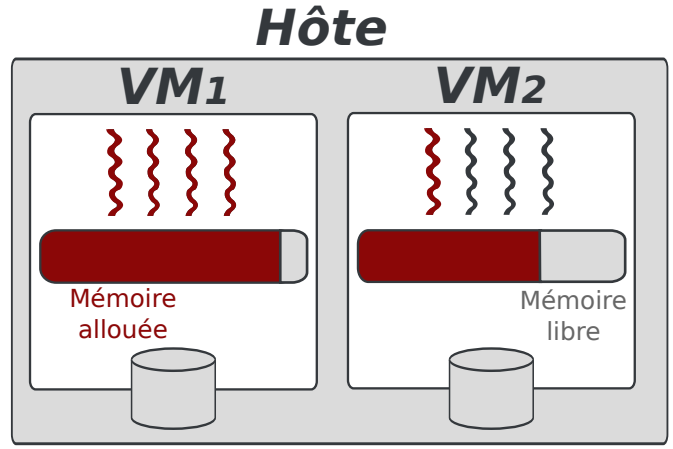
Dynamicité

- Charge variable dans le temps



Hétérogénéité

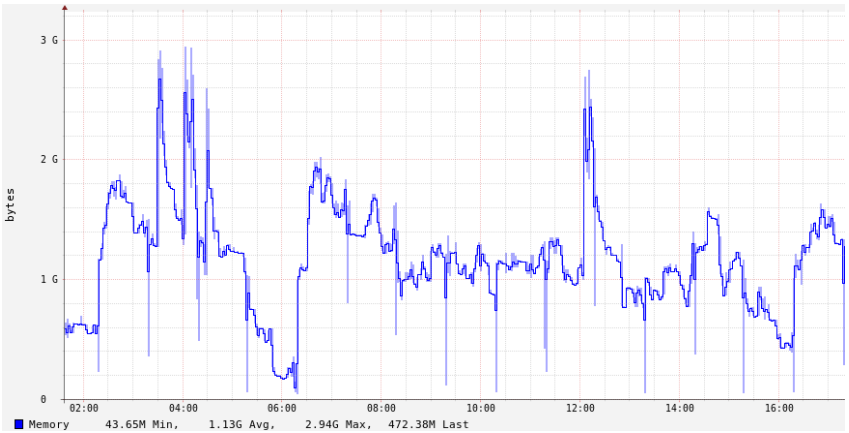
- Calcul (CPU, mémoire)
- Données (I/O)



Fragmentation de la mémoire : difficultés

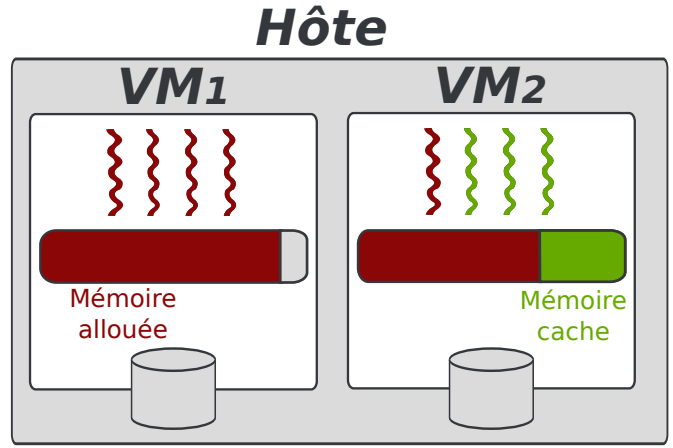
Dynamicité

- Charge variable dans le temps



Hétérogénéité

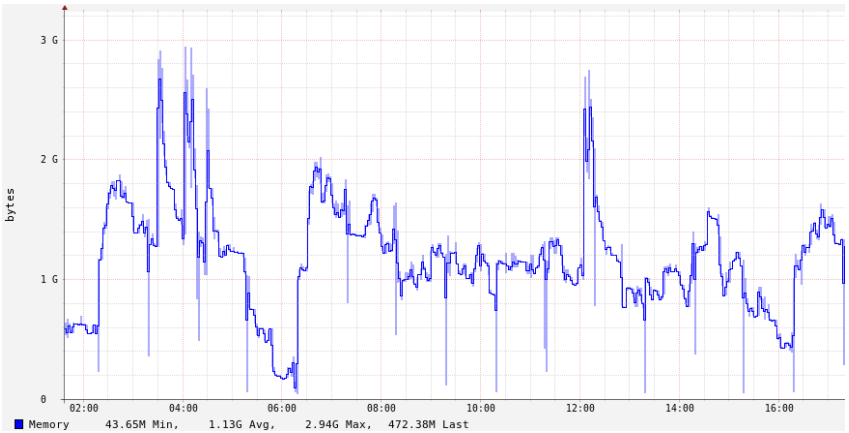
- Calcul (CPU, mémoire)
- Données (I/O)



Fragmentation de la mémoire : difficultés

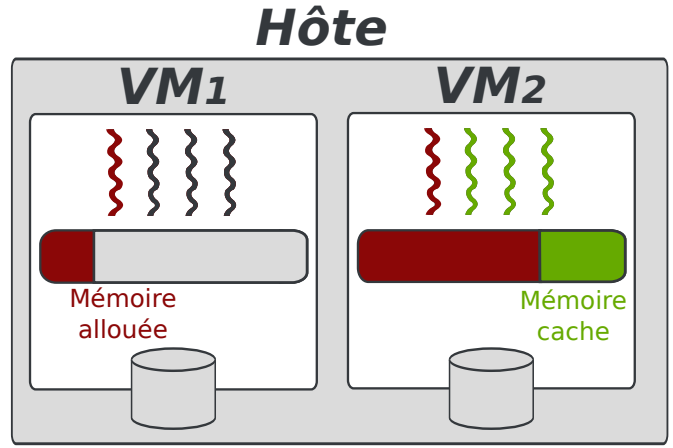
Dynamicité

- Charge variable dans le temps



Hétérogénéité

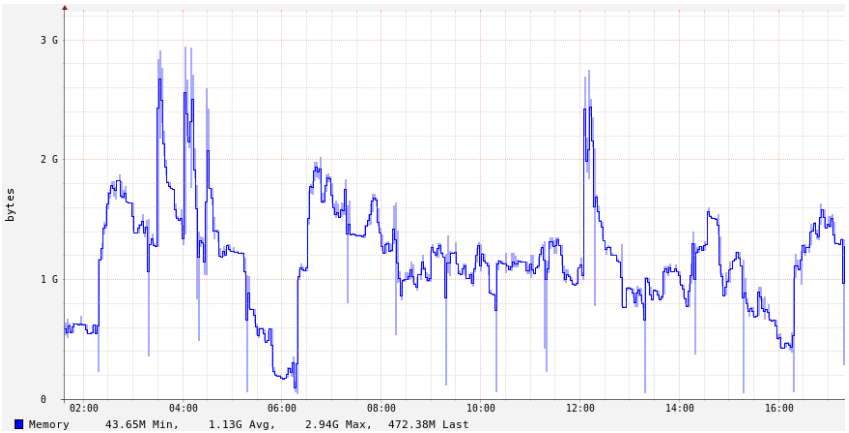
- Calcul (CPU, mémoire)
- Données (I/O)



Fragmentation de la mémoire : difficultés

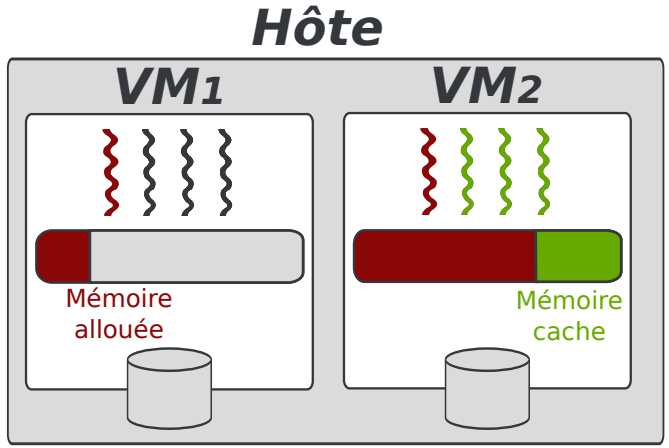
Dynamicité

- Charge variable dans le temps



Hétérogénéité

- Calcul (CPU, mémoire)
- Données (I/O)



Objectif

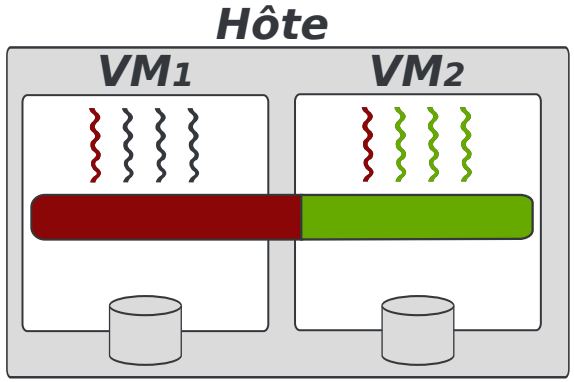
- Faire bénéficier à une VM *active* la mémoire d'une VM *inactive*

Approches classiques

Mémoire partagée distribuée

- Exemple : Kerrighed [CLUSTER '04]

⇒ Problème : perte de l'isolation entre VMs

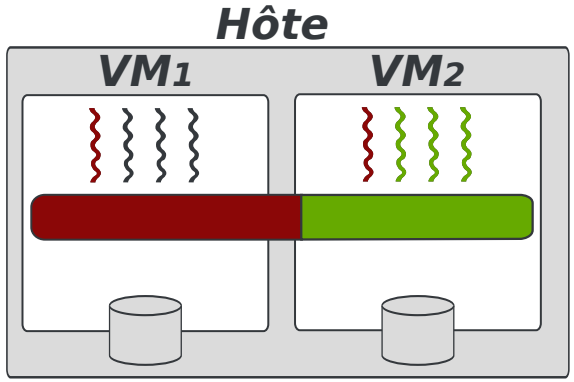


Approches classiques

Mémoire partagée distribuée

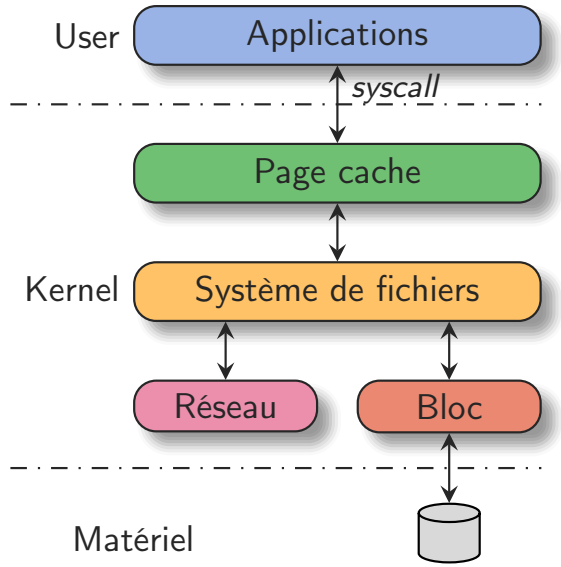
- Exemple : Kerrighed [CLUSTER '04]

⇒ Problème : perte de l'isolation entre VMs



Caches répartis

- Exemples
  - Memcached
    - → Modification des applications
  - xFS [SOSP '95], Shark [NSDI '05], NFS
    - → Système de fichiers imposé
  - dm-cache, FlashCache, bcache
    - → Limité à certains systèmes de fichiers

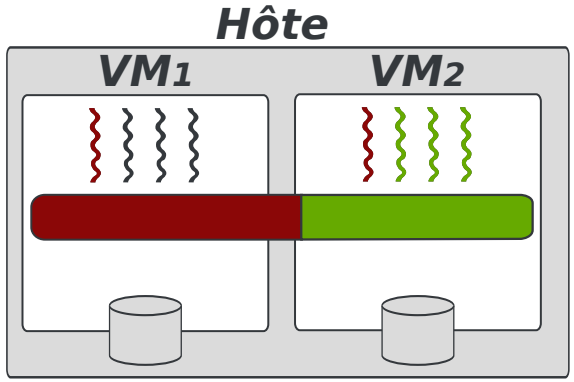


Approches classiques

Mémoire partagée distribuée

- Exemple : Kerrighed [CLUSTER '04]

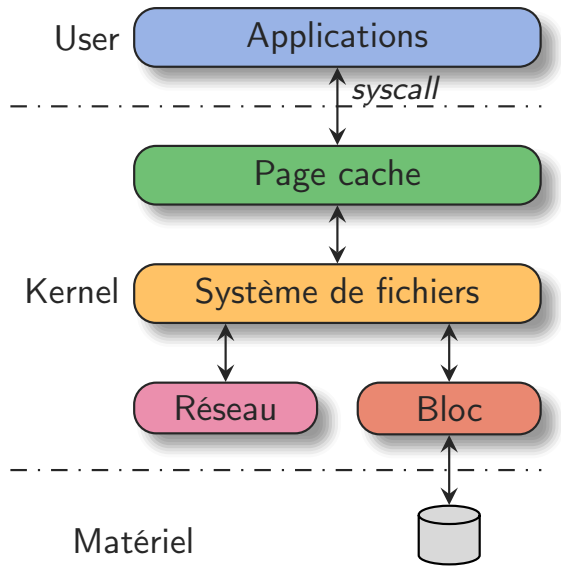
⇒ Problème : perte de l'isolation entre VMs



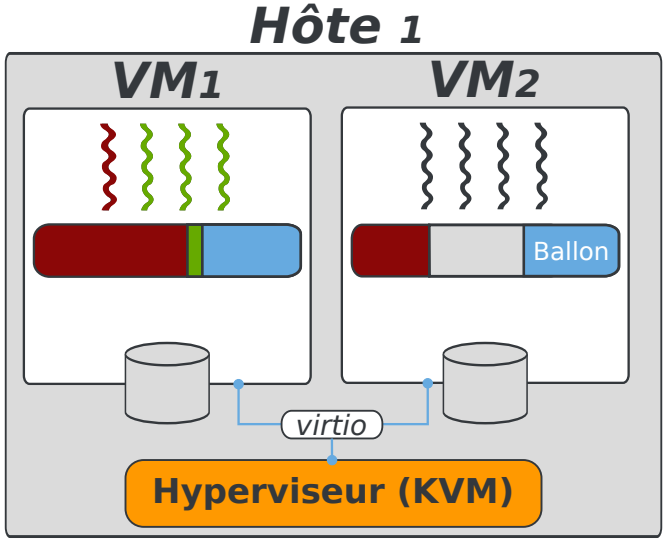
Caches répartis

- Exemples
  - Memcached
    - → Modification des applications
  - xFS [SOSP '95], Shark [NSDI '05], NFS
    - → Système de fichiers imposé
  - dm-cache, FlashCache, bcache
    - → Limité à certains systèmes de fichiers

⇒ Problème : manque de transparence



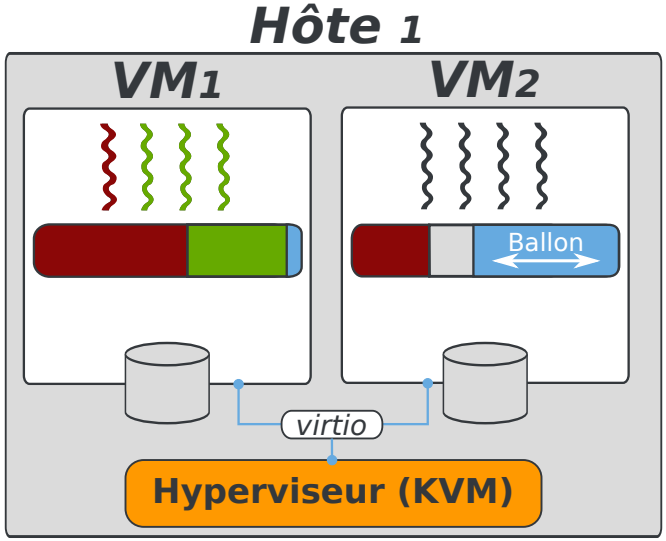
Approches virtualisées – *Memory ballooning* [OSDI '02]



Principe

- Une VM *gonfle* son ballon pour rendre de la mémoire
- Une VM *dégonfle* son ballon pour obtenir plus de mémoire

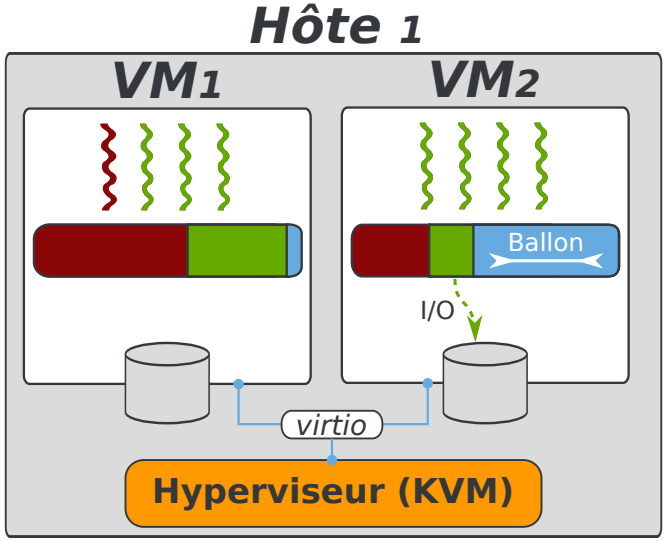
Approches virtualisées – *Memory ballooning* [OSDI '02]



Principe

- Une VM *gonfle* son ballon pour rendre de la mémoire
- Une VM *dégonfle* son ballon pour obtenir plus de mémoire

Approches virtualisées – *Memory ballooning* [OSDI '02]



Principe

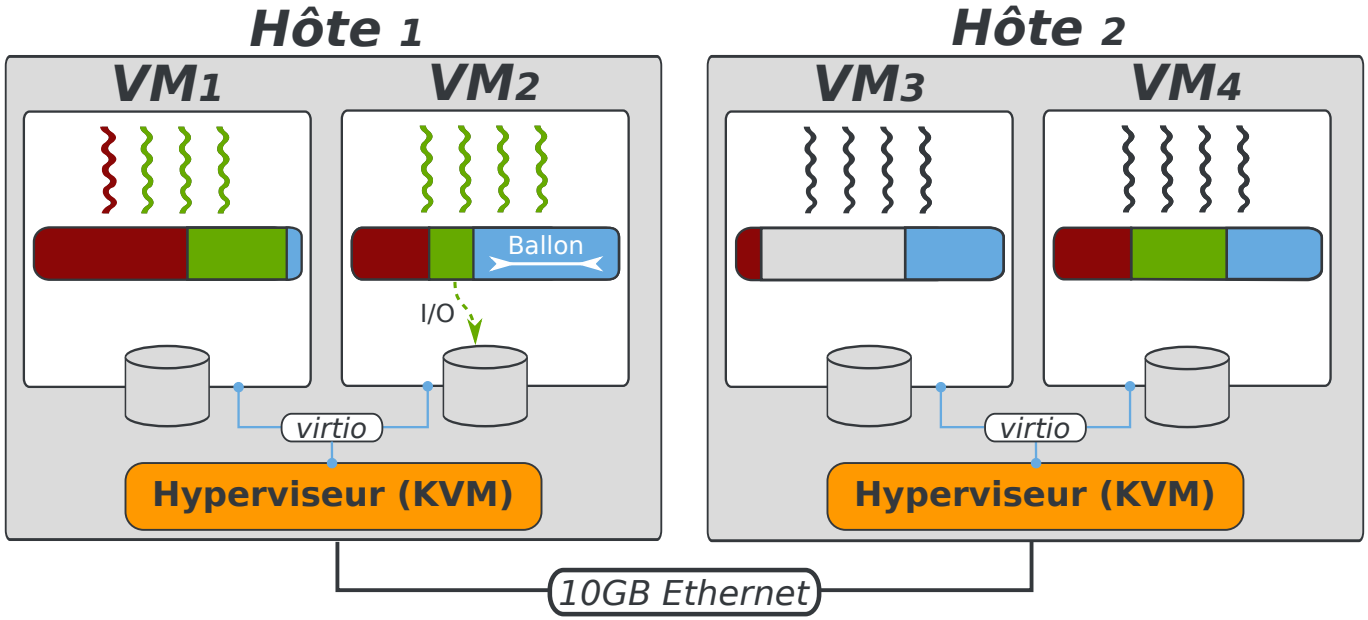
- Une VM *gonfle* son ballon pour rendre de la mémoire
- Une VM *dégonfle* son ballon pour obtenir plus de mémoire

Limites

1. Performances

   $\Rightarrow$ Difficilement automatisable

Approches virtualisées – *Memory ballooning* [OSDI '02]



Principe

- Une VM *gonfle* son ballon pour rendre de la mémoire
- Une VM *dégonfle* son ballon pour obtenir plus de mémoire

Limites

1. Performances
   ⇒ Difficilement automatisable
2. Fonctionnelles
   ⇒ Limitation à un seul hôte

Contributions de cette thèse

Objectifs : mutualiser la mémoire inutilisée entre les VMs à l'échelle d'un centre de données

⇒ Caches répartis transparents et efficaces entre VMs

⇒ Automatisation et dynamicité des caches répartis

Contribution 1

Puma : Pooling Unused memory in virtual MAchines

Contribution 2

Gestion dynamique du cache entre machines virtuelles

Première partie

Puma : mutualisation de la mémoire inutilisée entre machines virtuelles

Principes de Puma

Principe 1 : intégration au cœur du noyau des VMs

$\Rightarrow$ Indépendant de l'hyperviseur

Principe 2 : interaction avec le *page cache* du noyau Linux

$\Rightarrow$ Indépendant des applications

Principe 3 : manipulation de pages propres

$\Rightarrow$ Écritures déjà performantes

$\Rightarrow$ Plus de flexibilité $\rightarrow$ disque local conserve une copie

Principes de Puma

Principe 1 : intégration au cœur du noyau des VMs

$\Rightarrow$ Indépendant de l'hyperviseur

Principe 2 : interaction avec le *page cache* du noyau Linux

$\Rightarrow$ Indépendant des applications

Principe 3 : manipulation de pages propres

$\Rightarrow$ Écritures déjà performantes

$\Rightarrow$ Plus de flexibilité $\rightarrow$ disque local conserve une copie

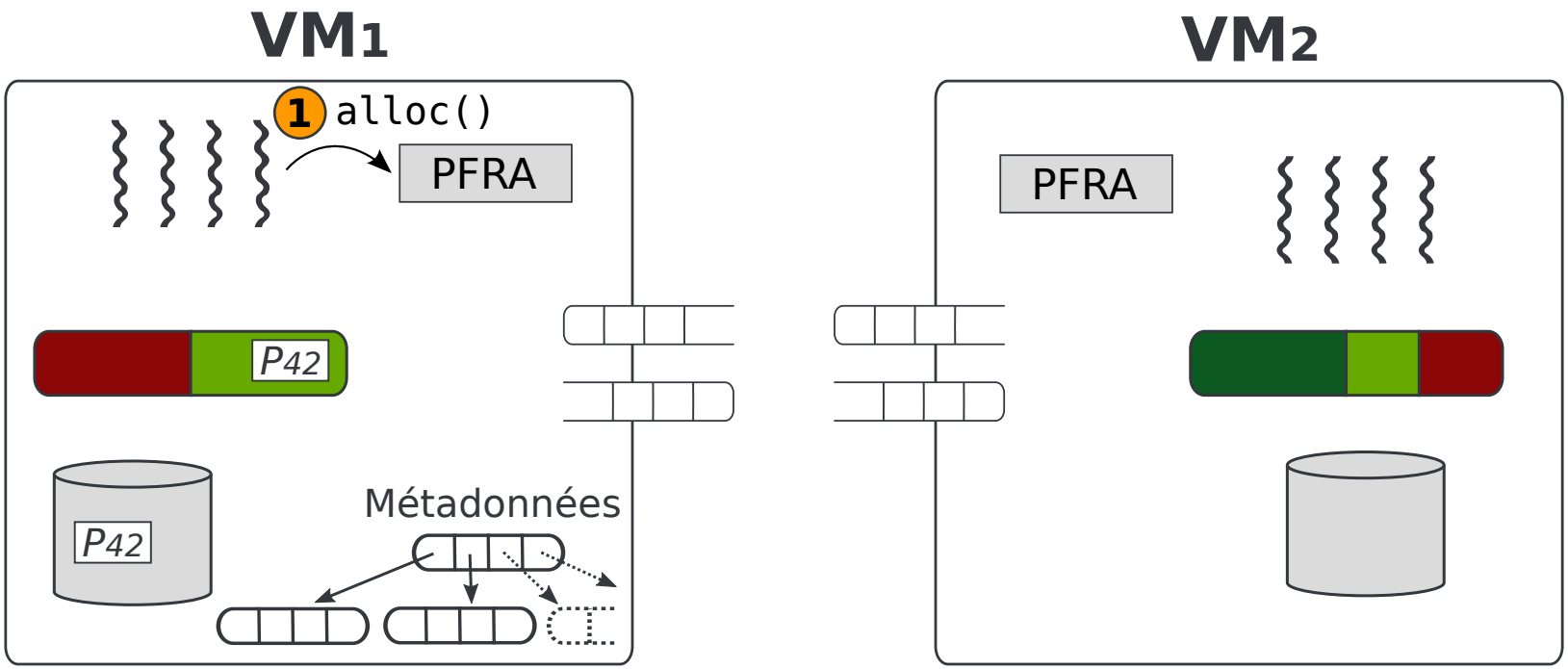
Interaction avec le noyau

$\Rightarrow$ On intercepte les éviction du cache (*put*)

$\Rightarrow$ On intercepte les défauts du cache (*get*)

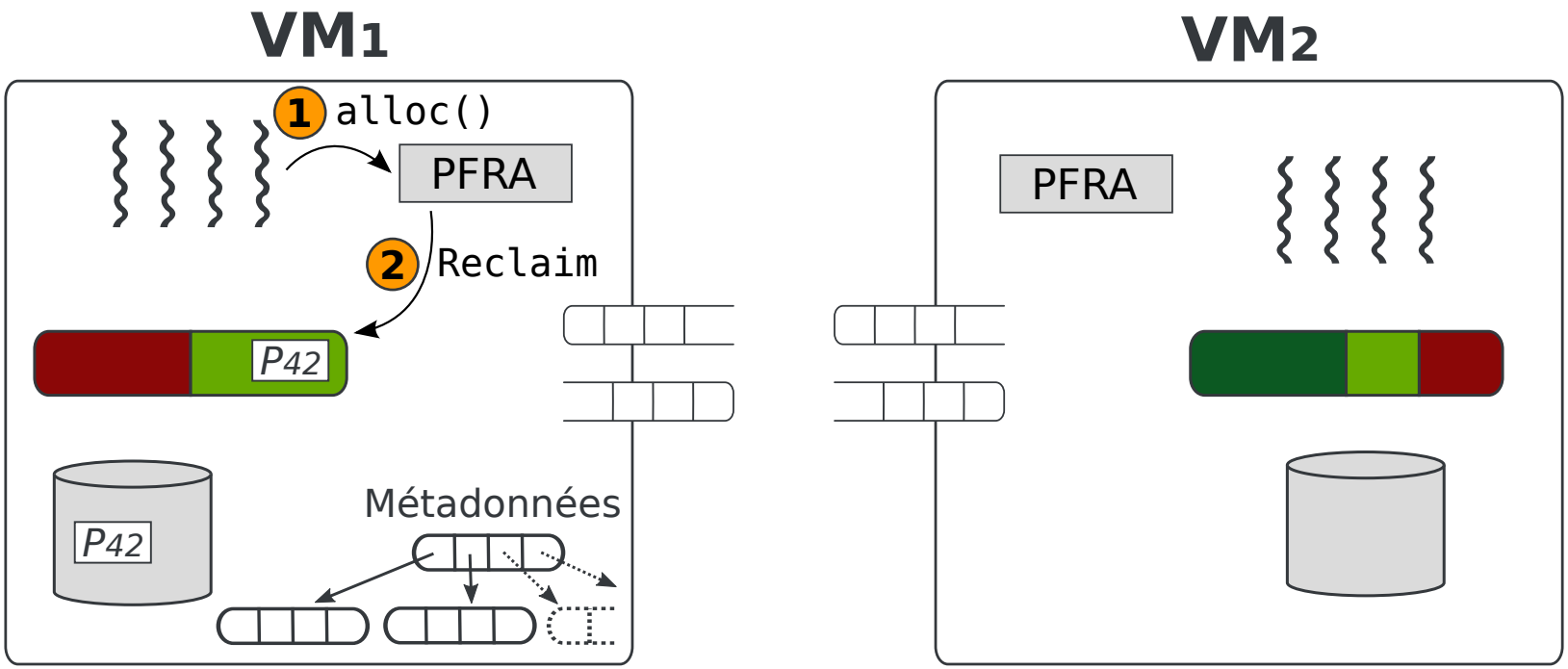
Opération *put*

Placement d'une page dans le cache



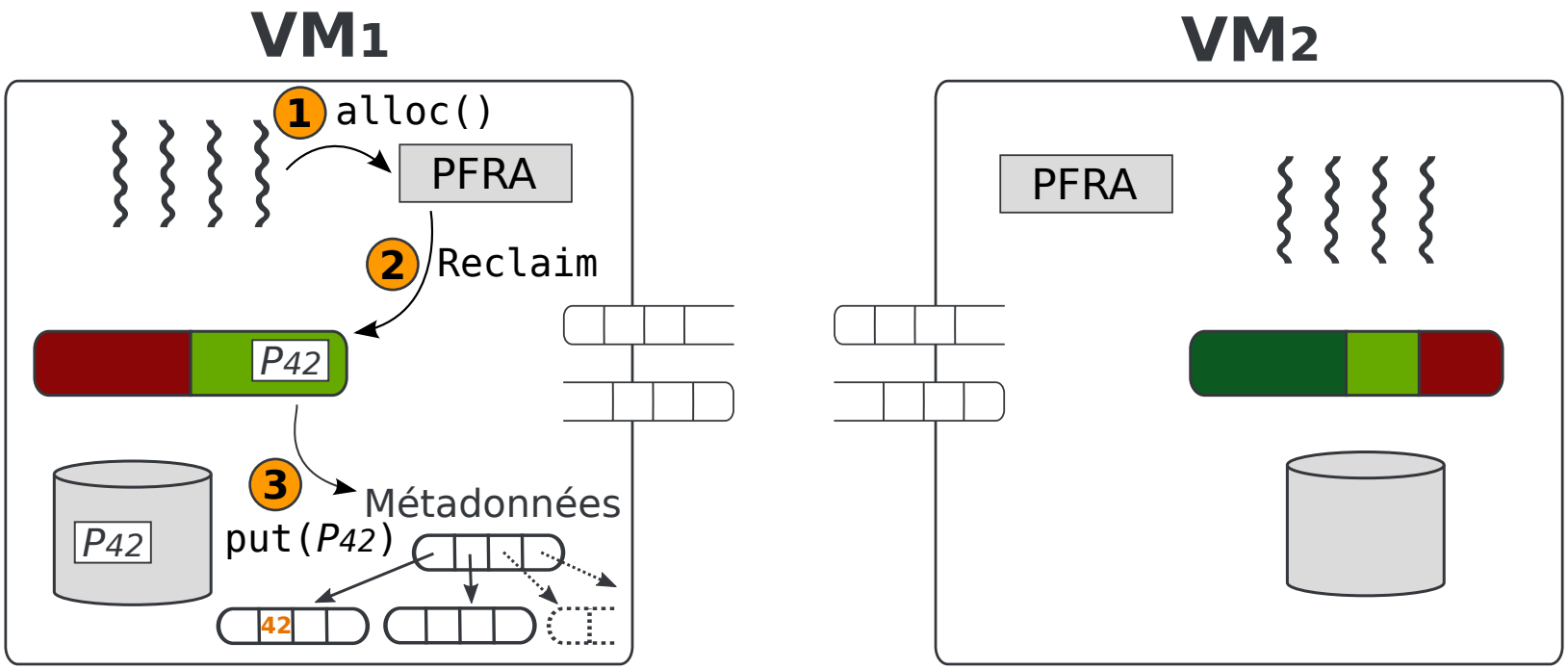
Opération *put*

Placement d'une page dans le cache



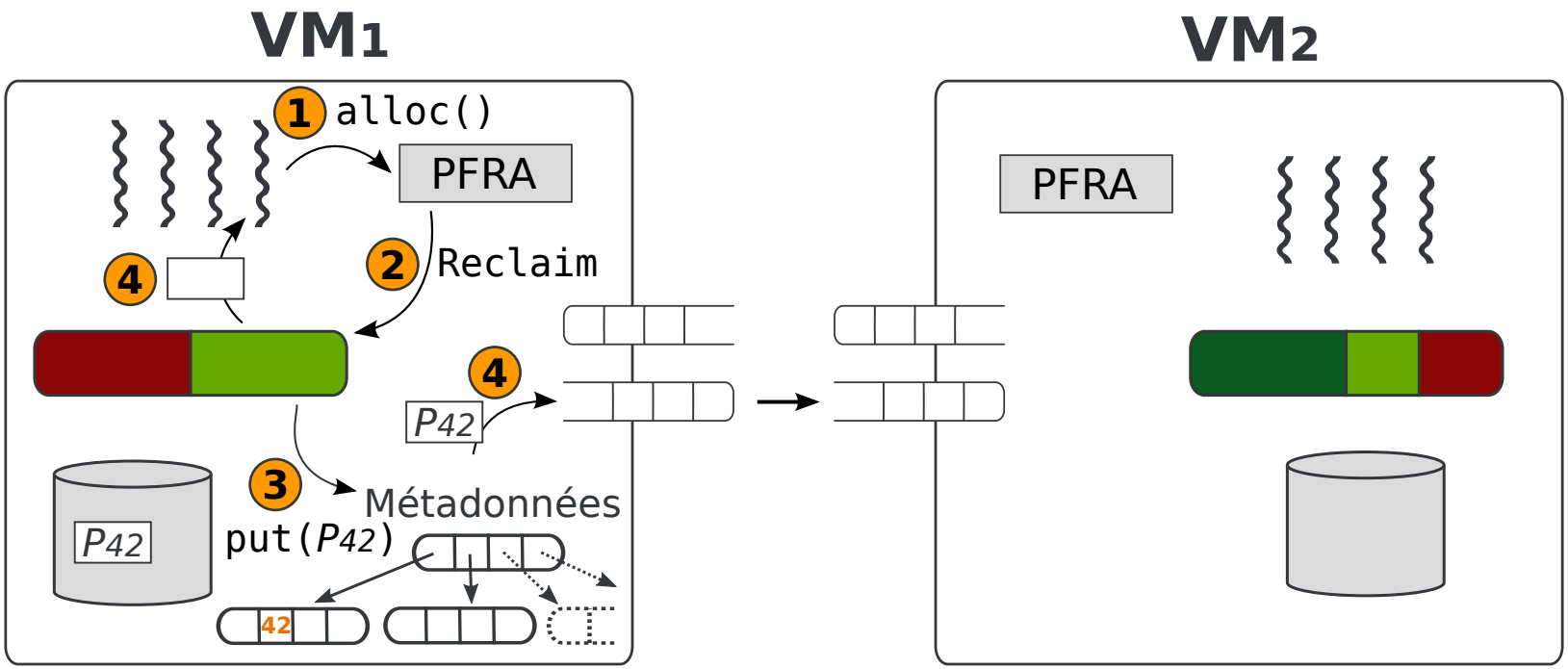
Opération *put*

Placement d'une page dans le cache



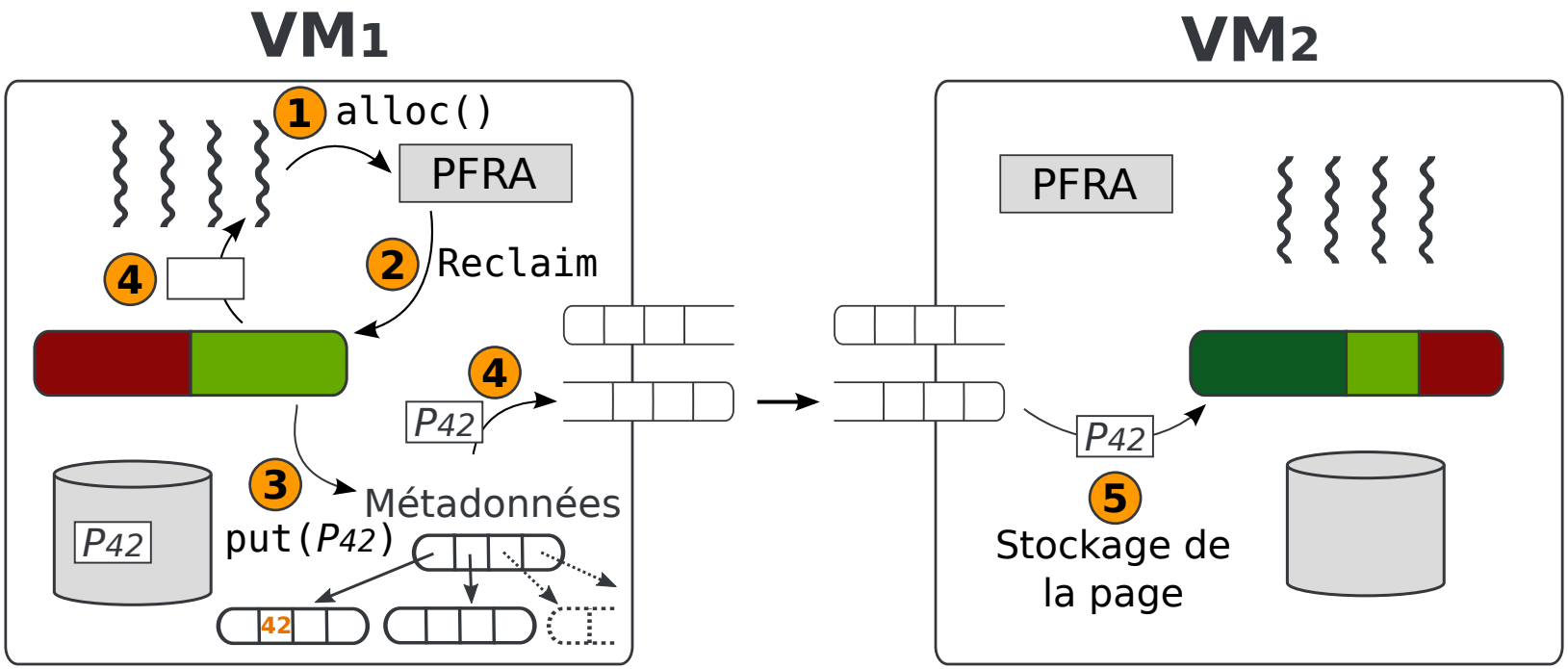
Opération *put*

Placement d'une page dans le cache



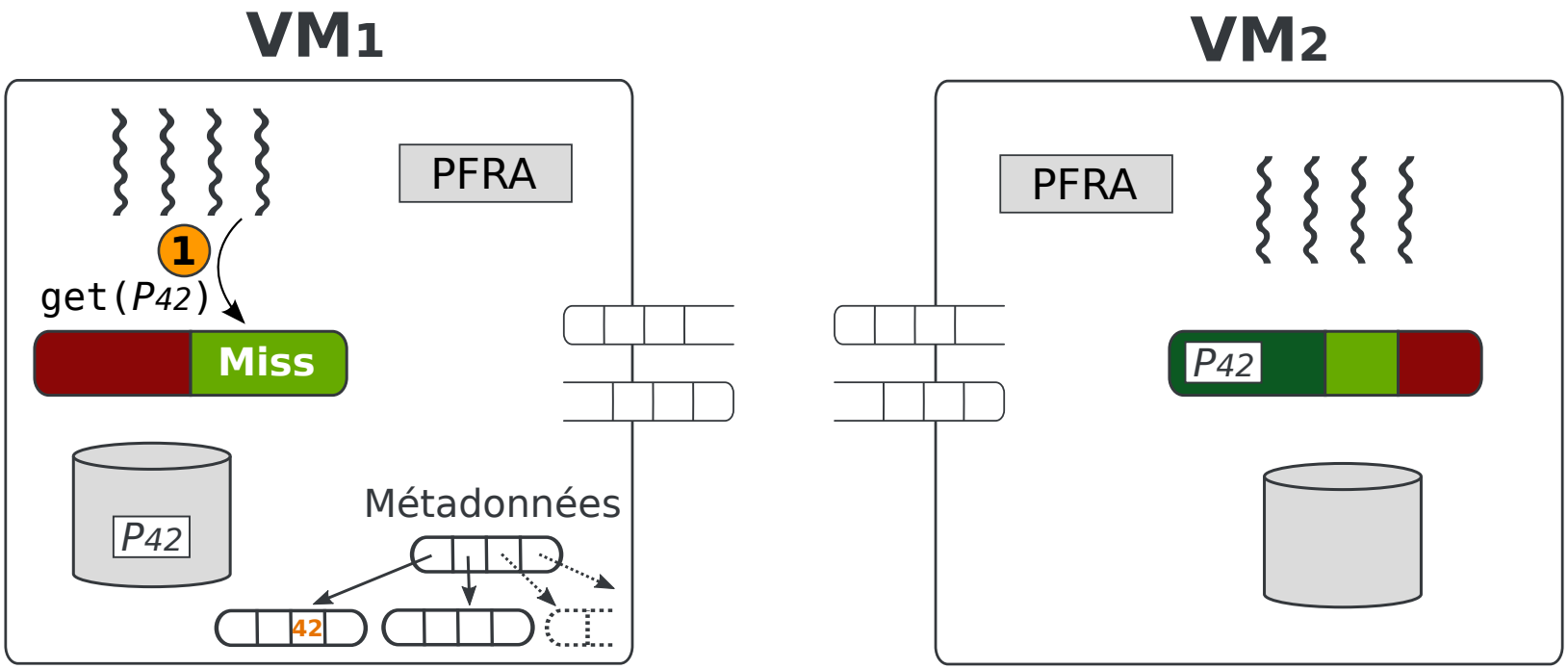
Opération *put*

Placement d'une page dans le cache



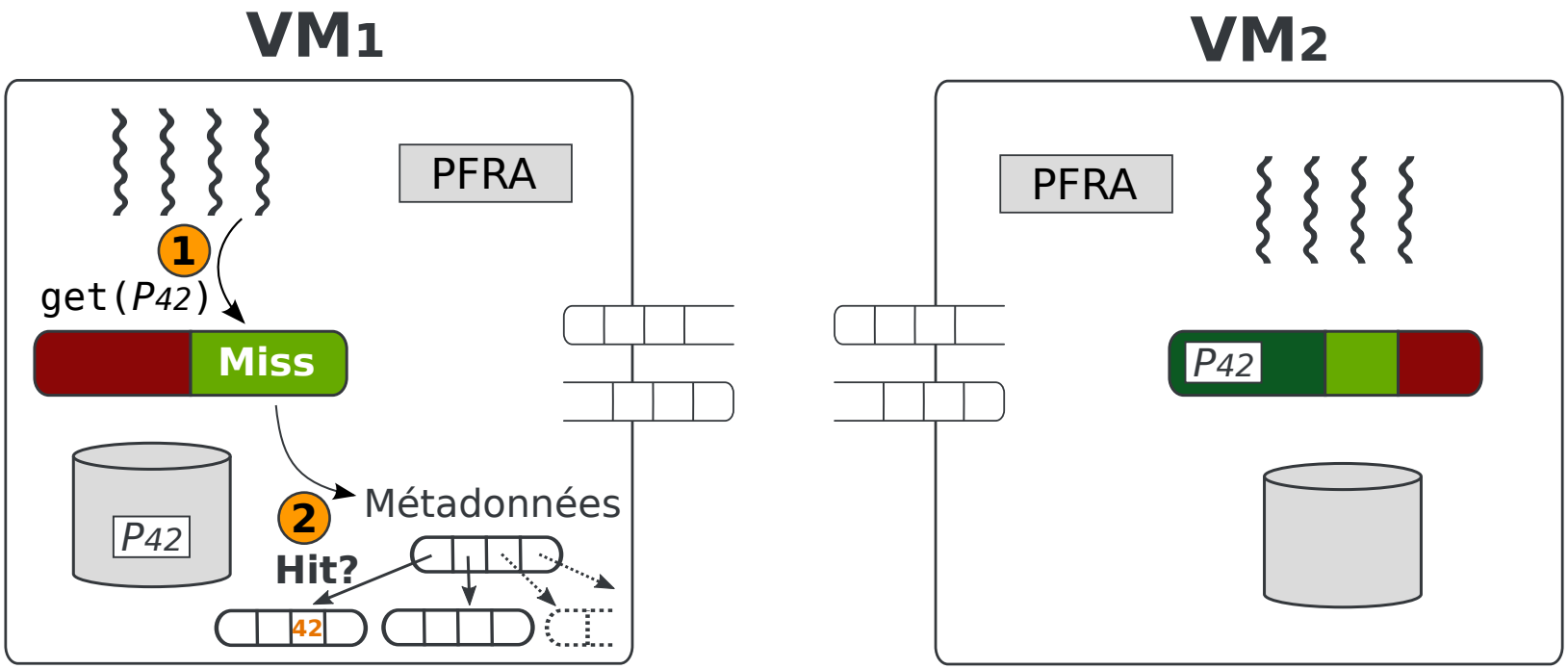
Opération *get*

Récupération d'une page du cache



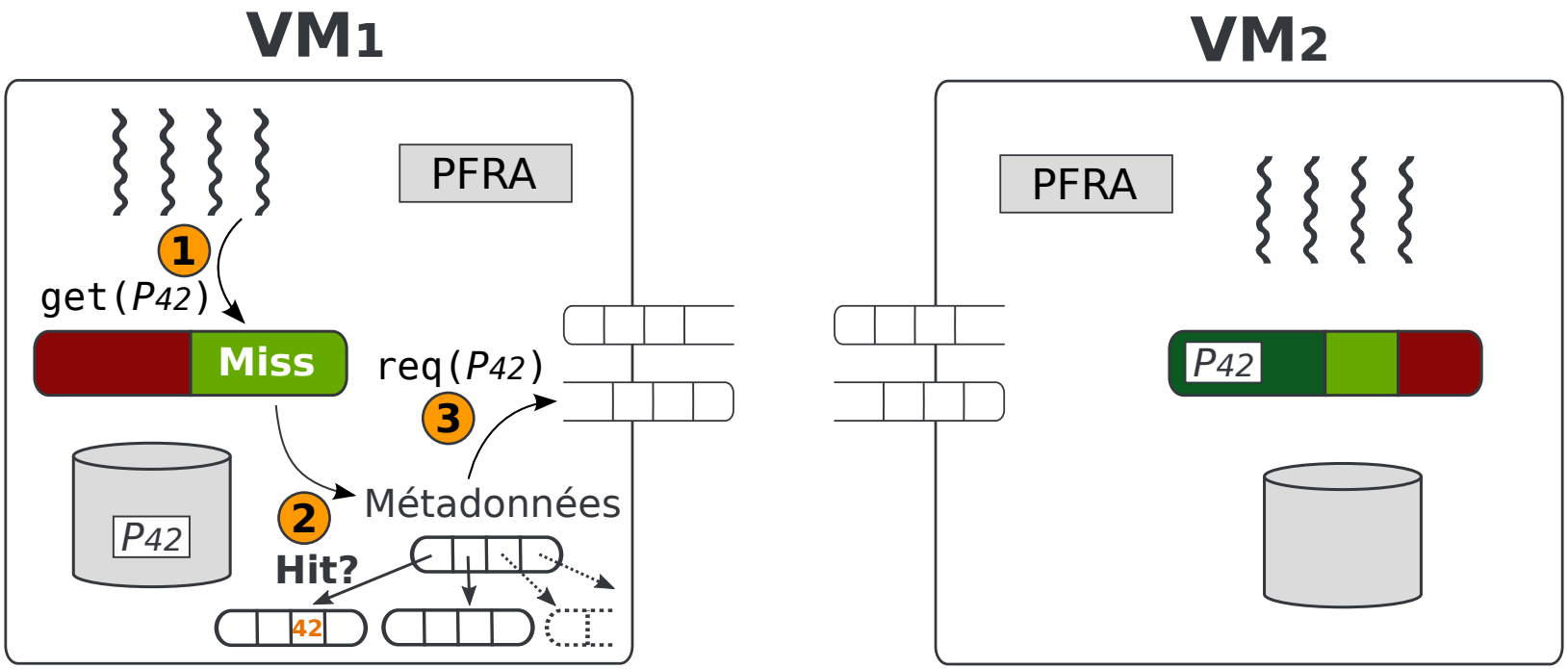
Opération *get*

Récupération d'une page du cache



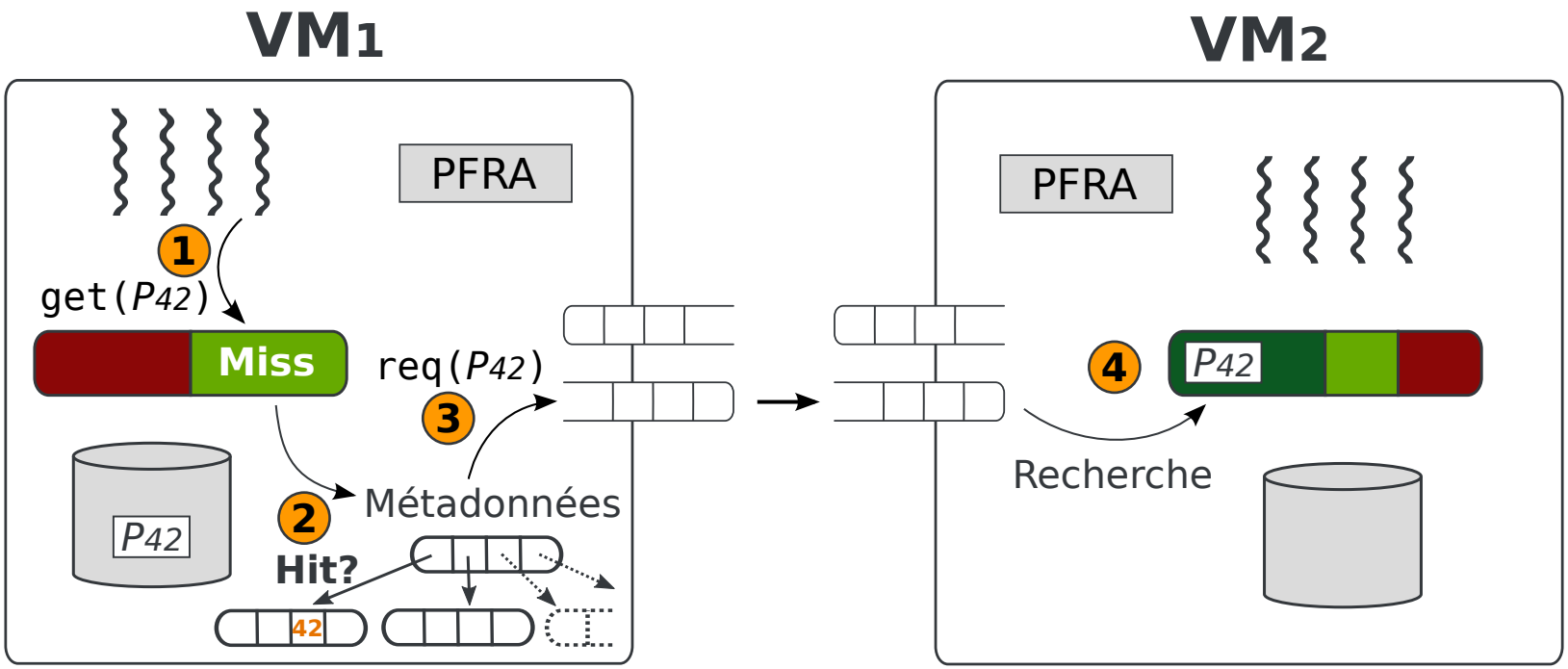
Opération *get*

Récupération d'une page du cache



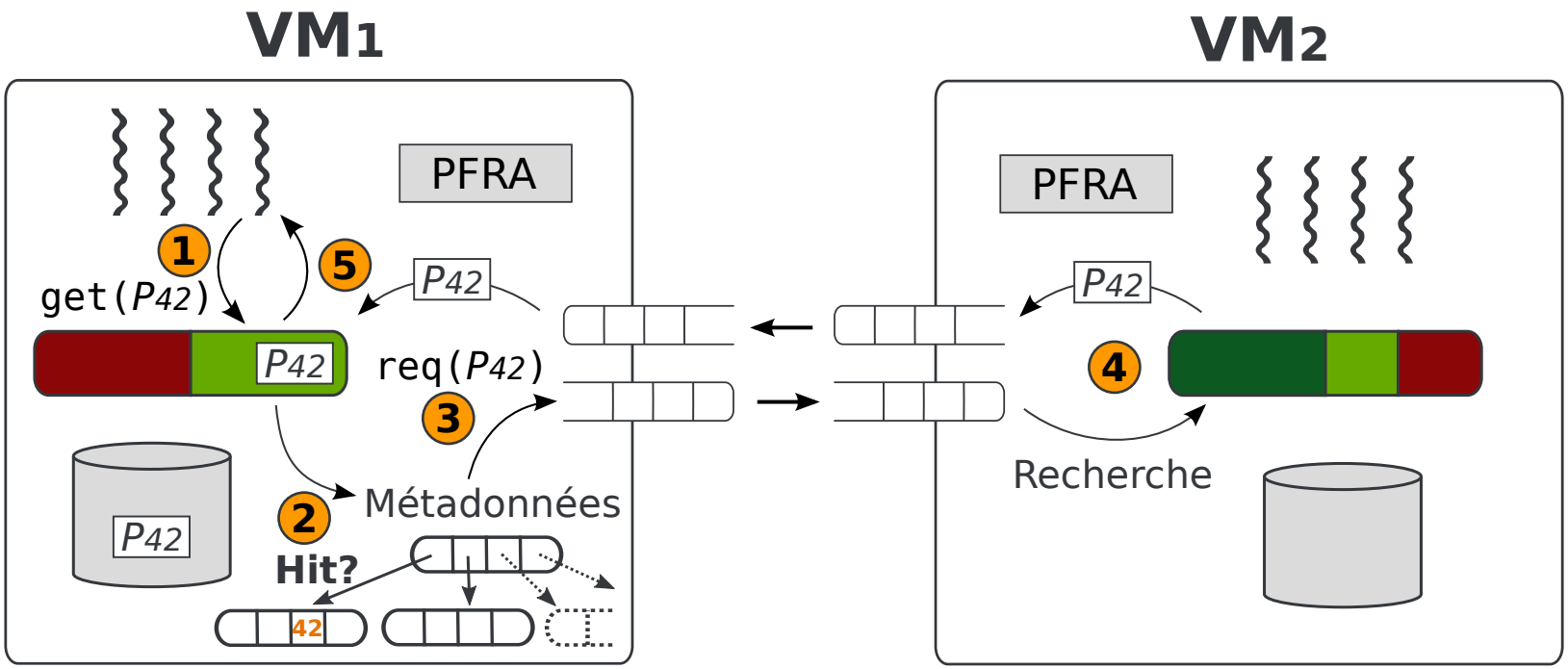
Opération *get*

Récupération d'une page du cache

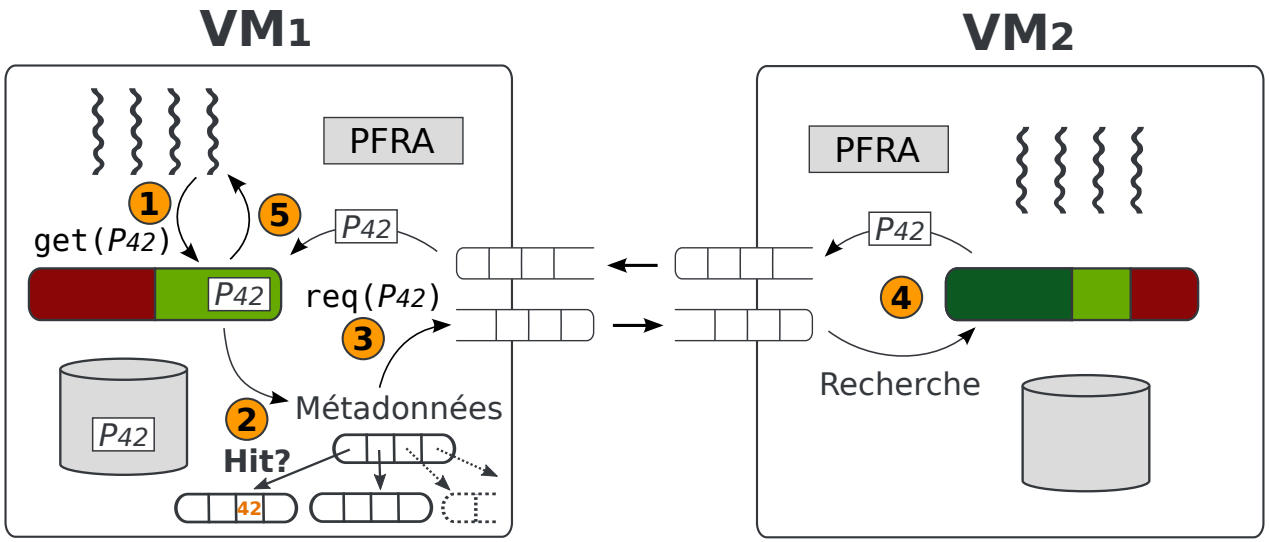


Opération *get*

Récupération d'une page du cache



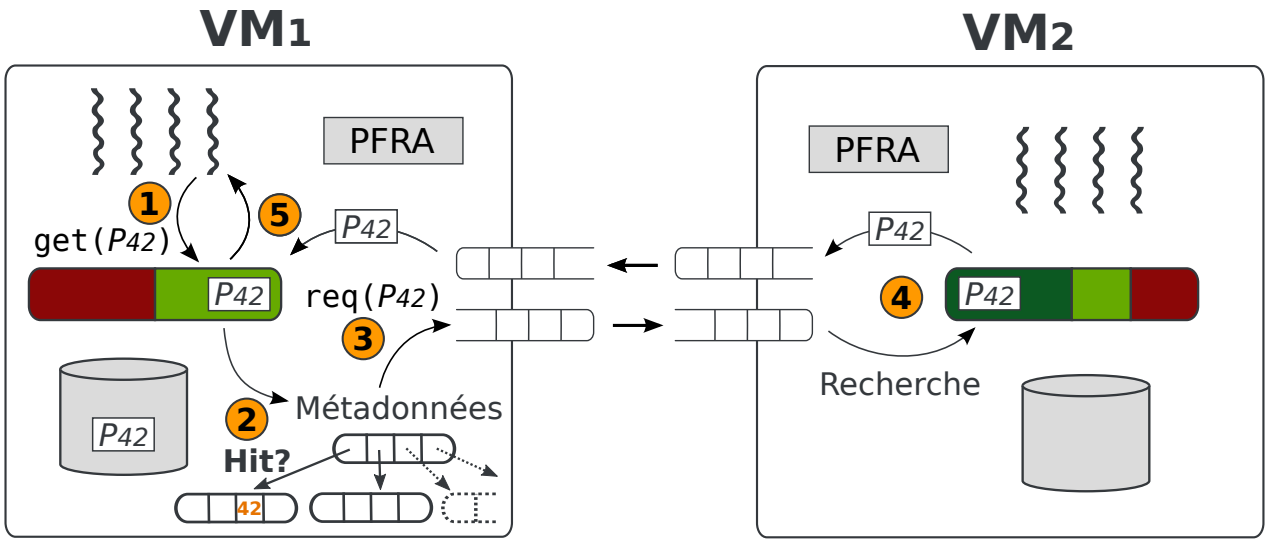
Stratégies de placement dans le cache



Inclusivité du cache

- Exclusive
- Non-inclusive

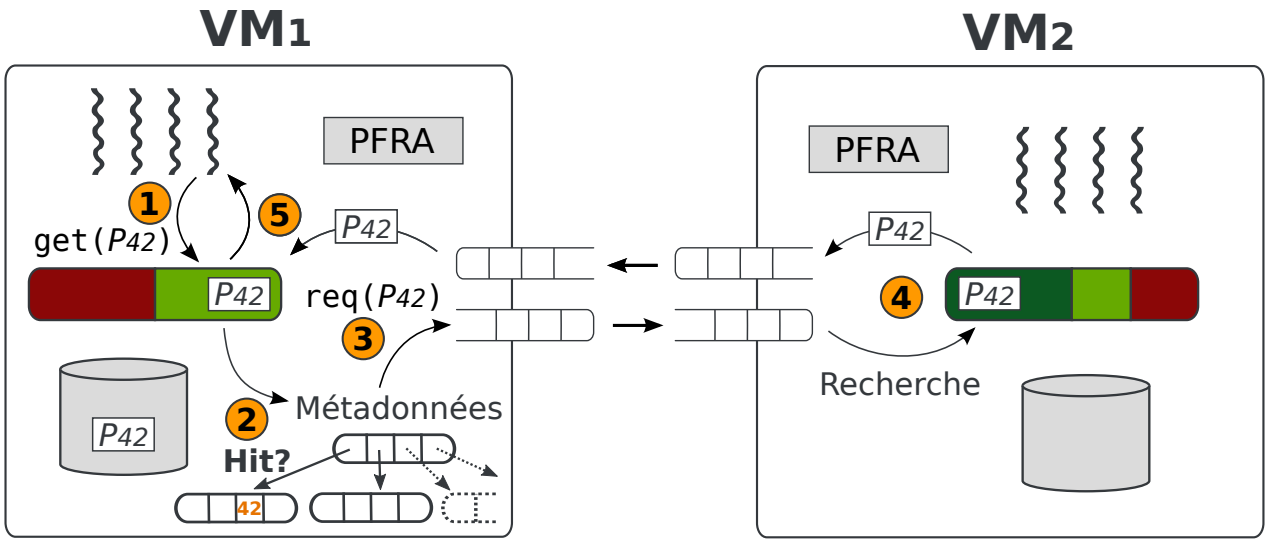
Stratégies de placement dans le cache



Inclusivité du cache

- Exclusive
- Non-inclusive

Stratégies de placement dans le cache



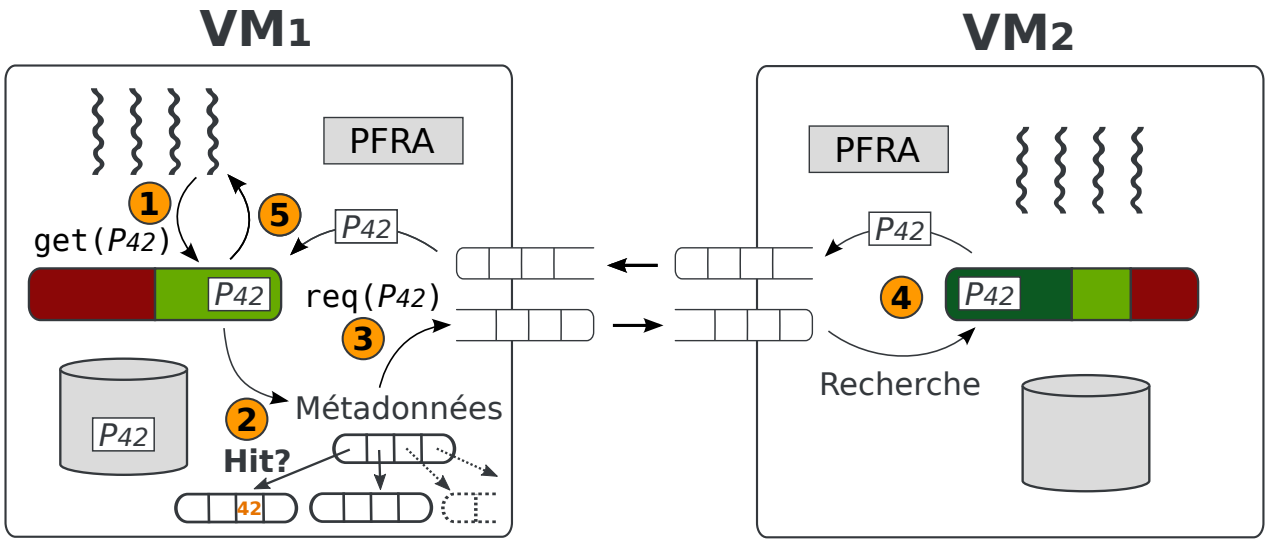
Inclusivité du cache

- Exclusive
- Non-inclusive

Filtrage des accès séquentiels

- Filtrés : ils ne vont pas dans le cache
- Non-filtrés : ils sont envoyés dans le cache

Stratégies de placement dans le cache

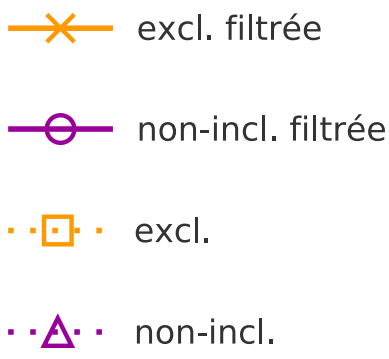


Inclusivité du cache

- Exclusive
- Non-inclusive

Filtrage des accès séquentiels

- Filtrés : ils ne vont pas dans le cache
- Non-filtrés : ils sont envoyés dans le cache



Évaluation

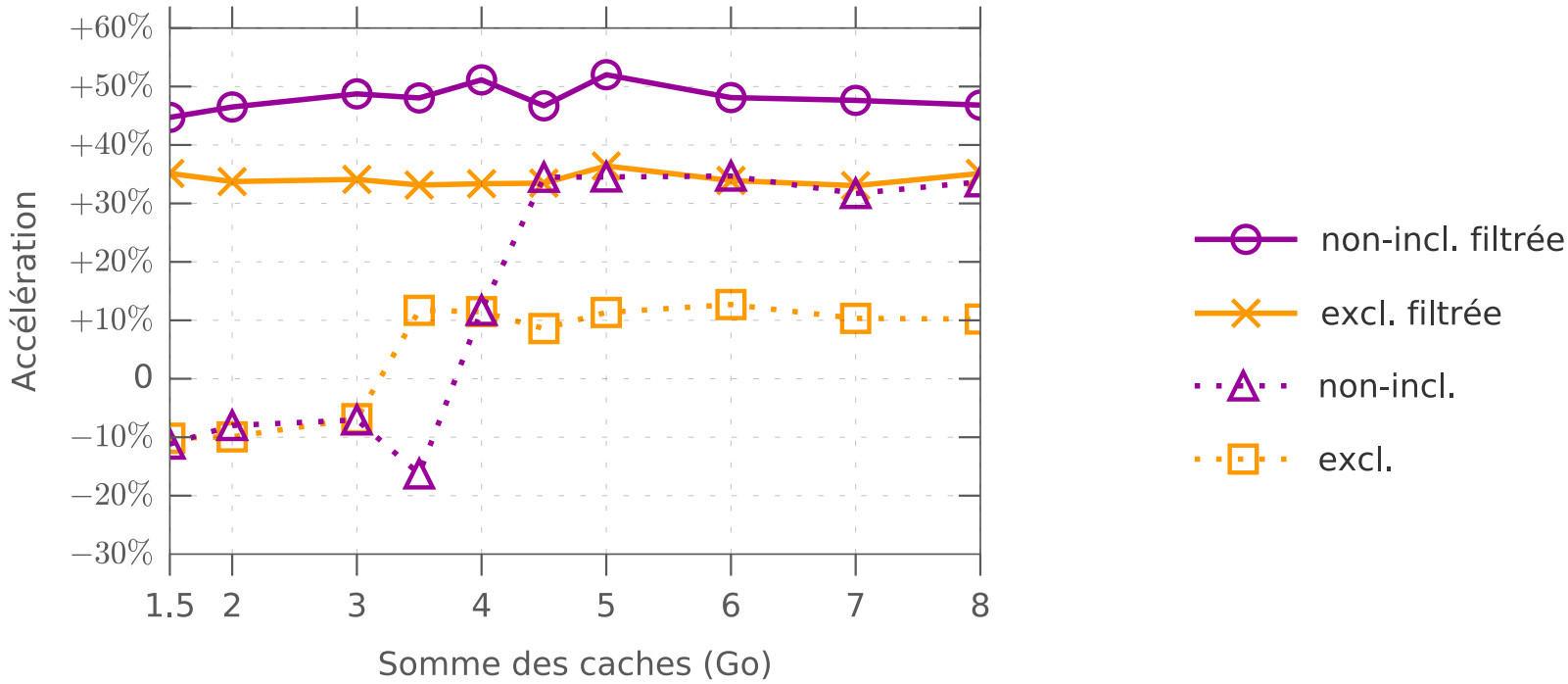
Plateforme expérimentale sur QEMU/KVM

- VM_1, active : 1 Go de mémoire
- VM_2, inactive : 512 Mo à 12 Go de mémoire
- Référence : une seule VM sans cache additionnel
- Machine utilisée : Intel Xeon E5-2660v2, $5 \times 600 GB\ SAS$ en RAID-0

Expériences

- Avec/sans filtre d'accès séquentiels
- Stratégies de cache exclusive, non-inclusive

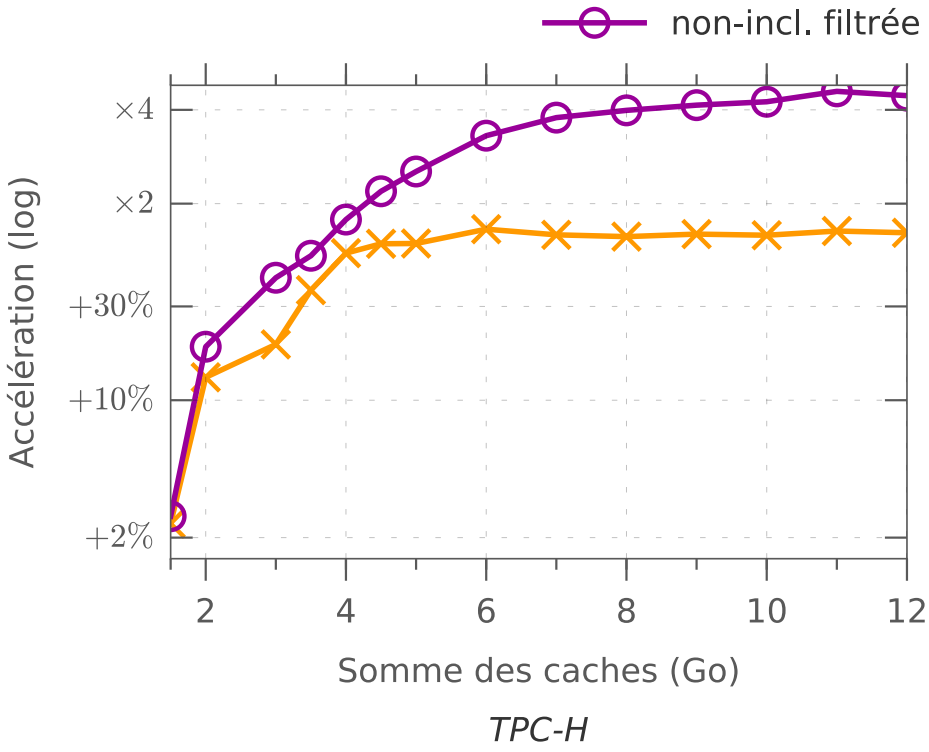
Accès séquentiels



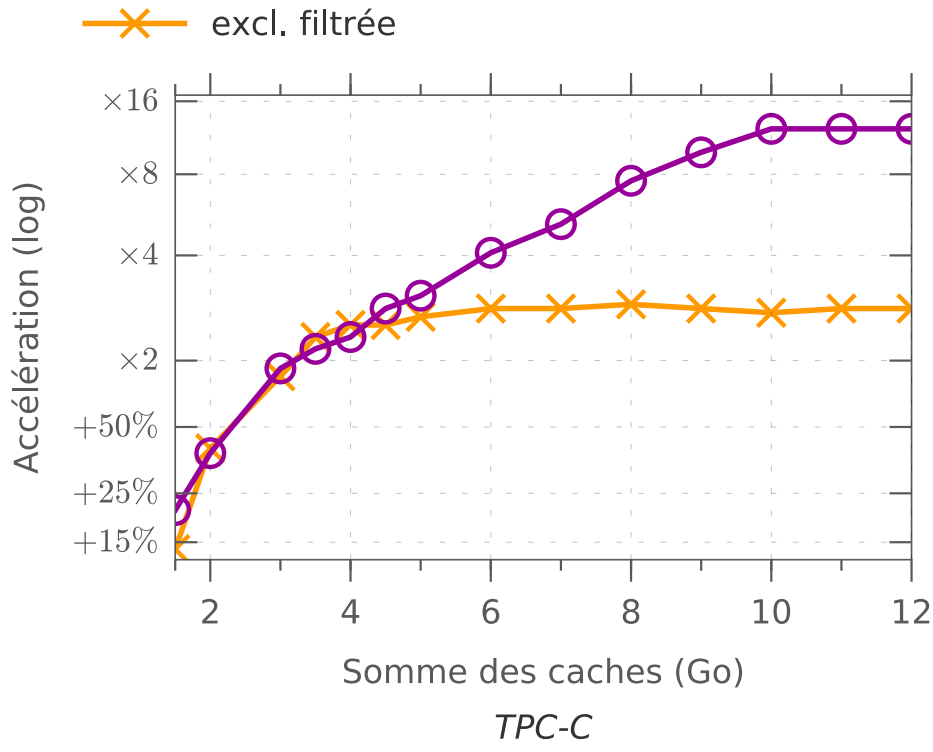
BLAST

Absence de filtre ⇒ dégrade les performances
Non-inclusive ⇒ meilleures performances globales

Performances avec des bases de données



TPC-H

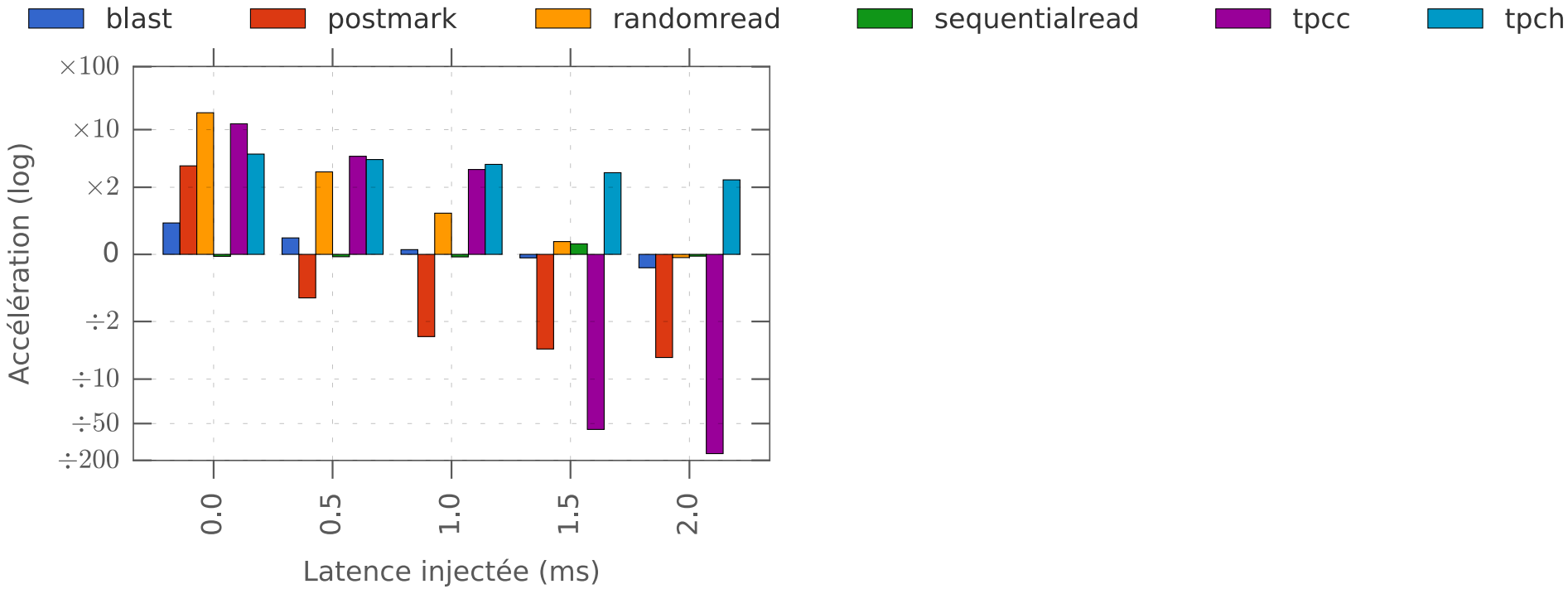


TPC-C

Accès aléatoires ⇒ importante amélioration

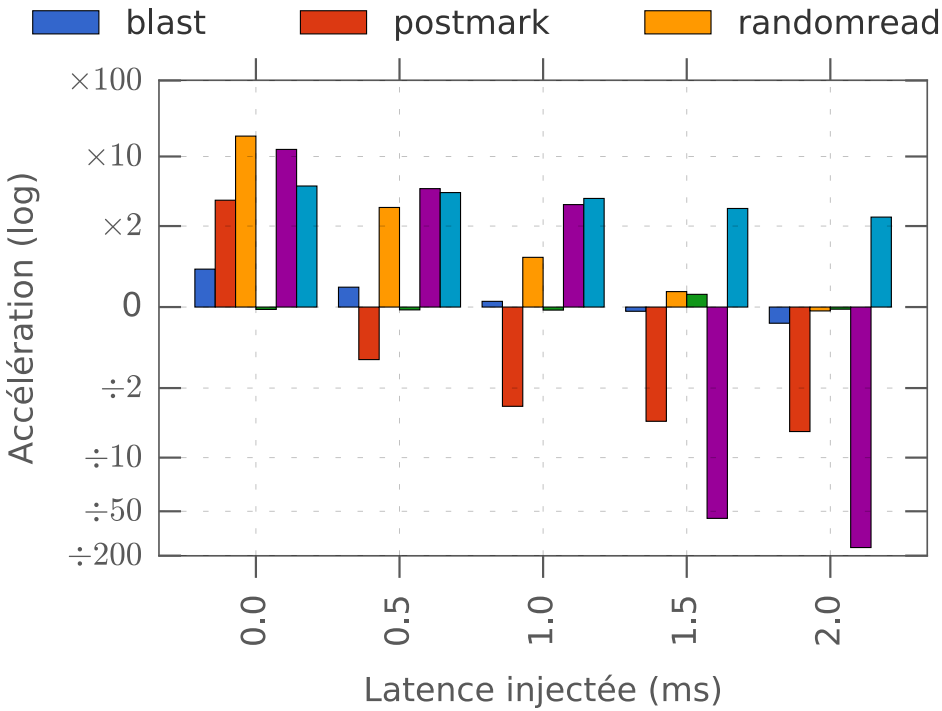
Gestion de la latence réseau

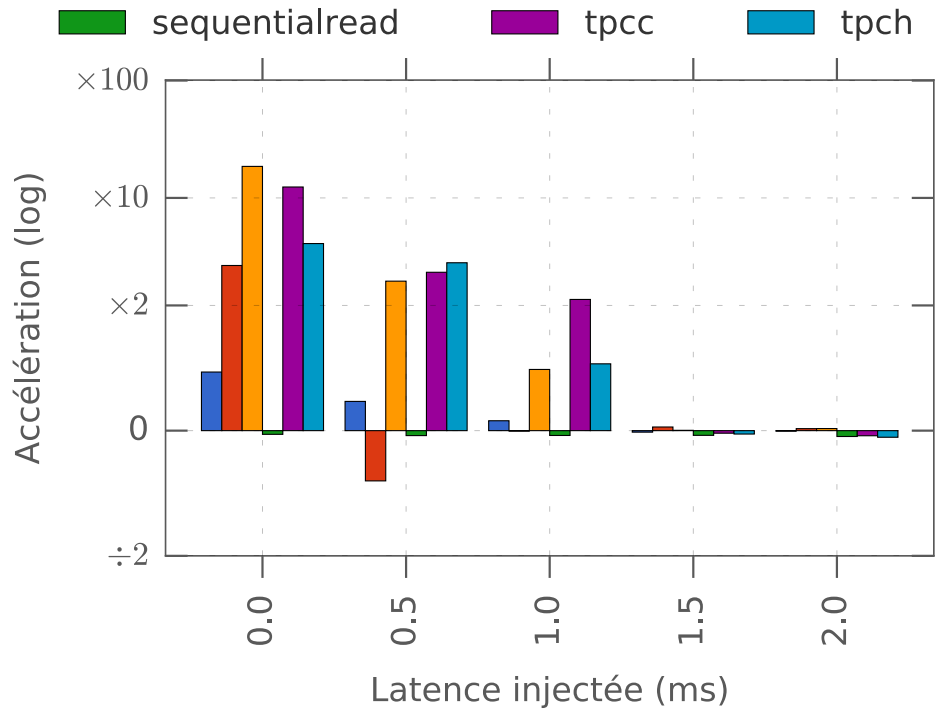
Injection de latence avec Netem [LCA'05]



Gestion de la latence réseau

Injection de latence avec Netem [LCA'05]





Monitoring de la latence de réseau

$\Rightarrow$ Puma se désactive si elle est trop élevée

Comparaison avec un cache SSD

Expérience

dm-cache *VM* seule, 1 Go, avec 5 Go de cache (SSD)

Puma $VM_1 \rightarrow 1$ Go, $VM_2 \rightarrow 5$ Go

	Lectures aléatoires	Postmark	BLAST	TPC-H
Puma	**×38**	+55%	**+267%**	**+218%**
dm-cache	×19	**×10**	+98%	+128%

- Puma plus efficace avec peu d'écritures
- *dm-cache* impacté autant que Puma par la virtualisation

Résumé de la contribution

Puma : Pooling Unused memory in virtual MAchines

- Cache réparti efficace et transparent car au niveau noyau
- Fonctionne avec des VMs colocalisées ou distantes
- Peu intrusif : la majorité est sous forme de module du noyau Linux
- Évalué avec de vraies applications

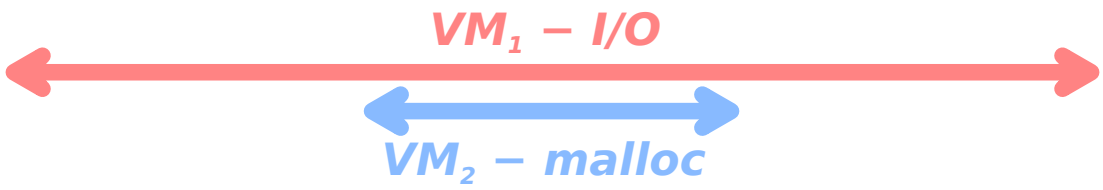
Approche prometteuse

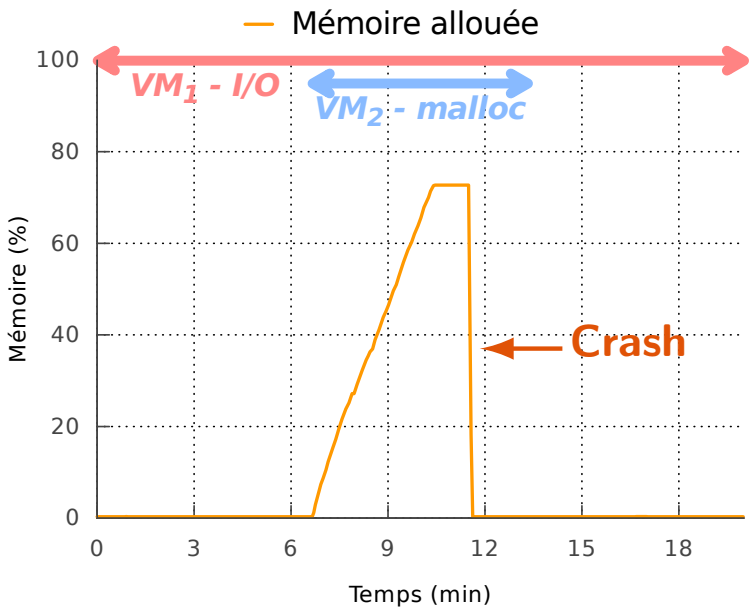
- Surcout négligeable
- Potentiels gains importants ($\times$4 avec TPC-H, $\times$16 avec TPC-C)

Deuxième partie

Gestion dynamique du cache entre machines virtuelles

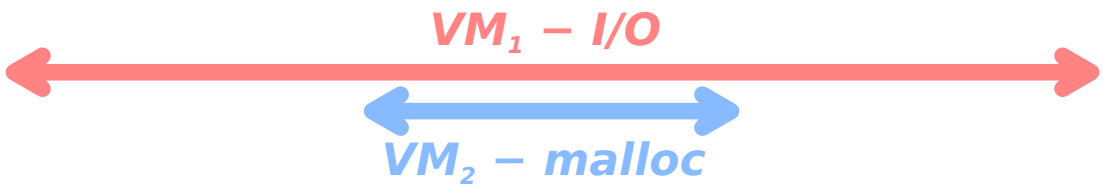
Approche existante : ballooning automatique (KVM)

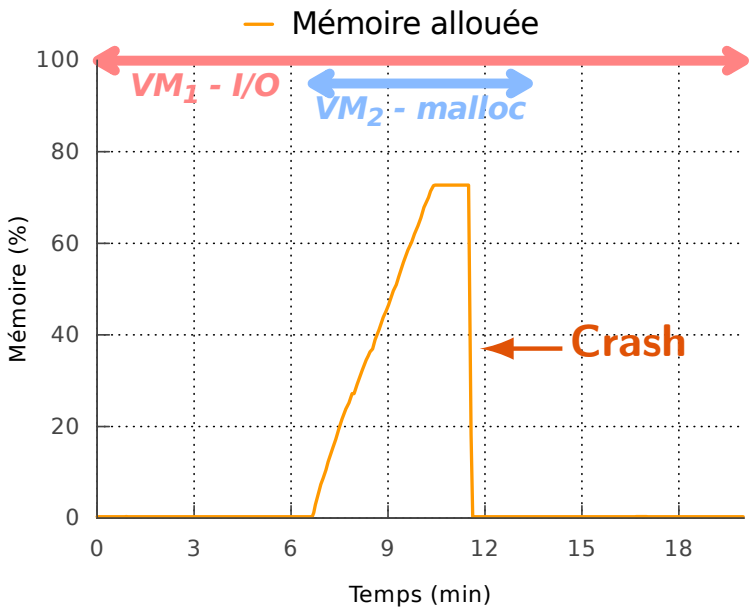




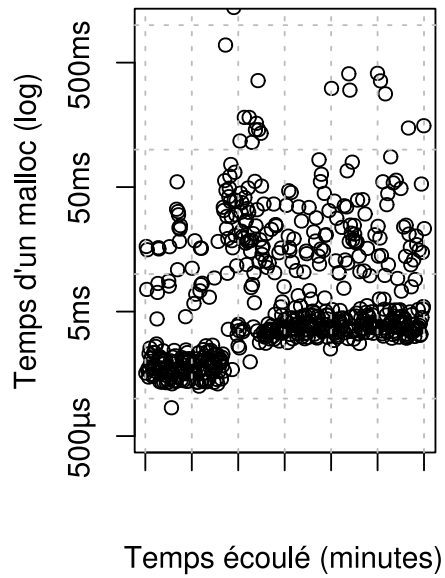
Ballooning (mémoire VM_2)

Approche existante : ballooning automatique (KVM)

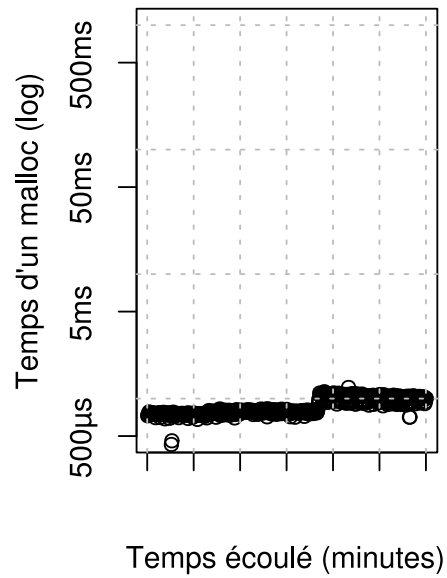




Ballooning (mémoire VM_2)



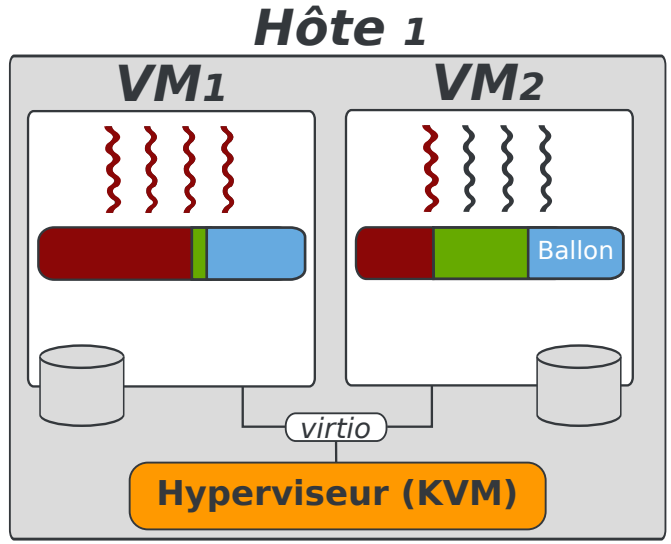
Ballooning (VM_2)



Référence (VM seule)

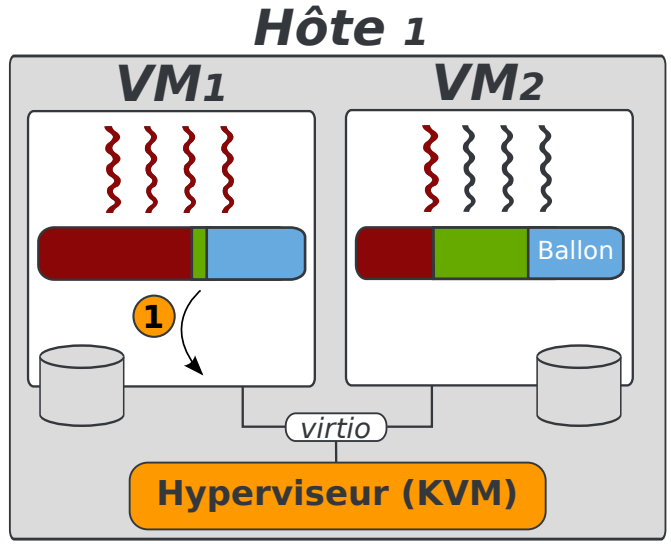
Allocations de mémoire 20× plus lentes (en moyenne)

Limites du ballooning



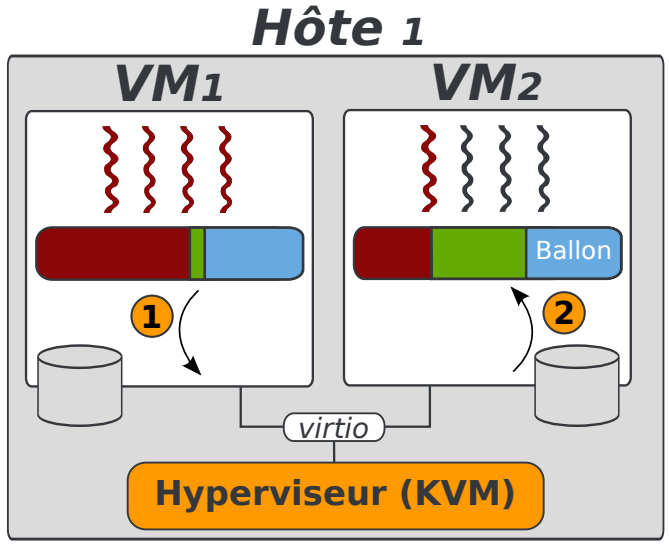
Limites du ballooning

1. VM_1 demande de la mémoire à l'hôte



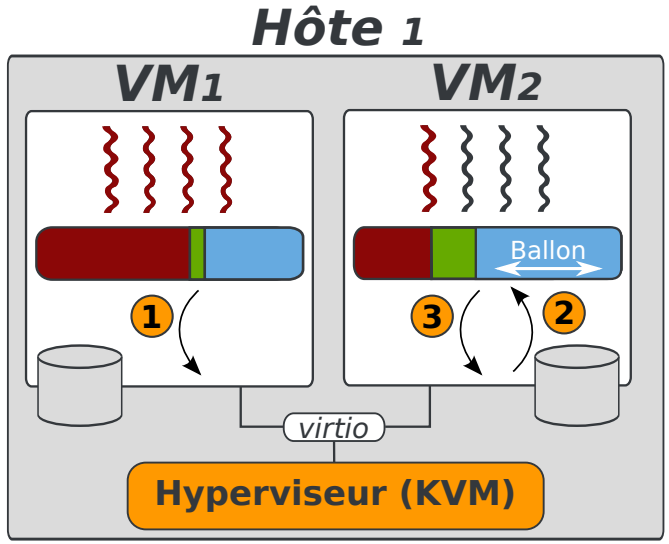
Limites du ballooning

1. VM_1 demande de la mémoire à l'hôte
2. L'hôte demande à VM_2 de gonfler son ballon



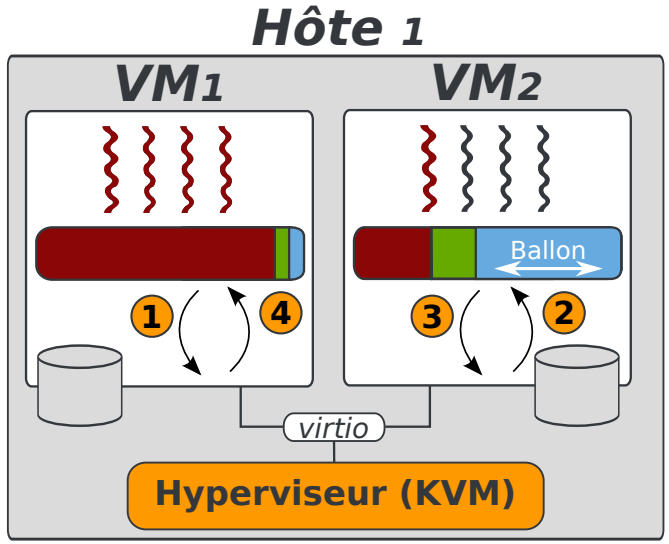
Limites du ballooning

1. VM_1 demande de la mémoire à l'hôte
2. L'hôte demande à VM_2 de gonfler son ballon
3. VM_2 rend de la mémoire à l'hôte



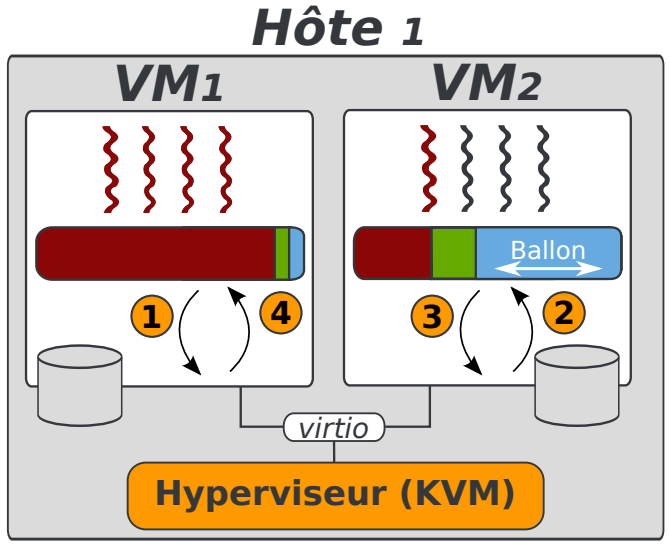
Limites du ballooning

1. VM_1 demande de la mémoire à l'hôte
2. L'hôte demande à VM_2 de gonfler son ballon
3. VM_2 rend de la mémoire à l'hôte
4. L'hôte donne cette mémoire à VM_1



Limites du ballooning

1. VM_1 demande de la mémoire à l'hôte
2. L'hôte demande à VM_2 de gonfler son ballon
3. VM_2 rend de la mémoire à l'hôte
4. L'hôte donne cette mémoire à VM_1



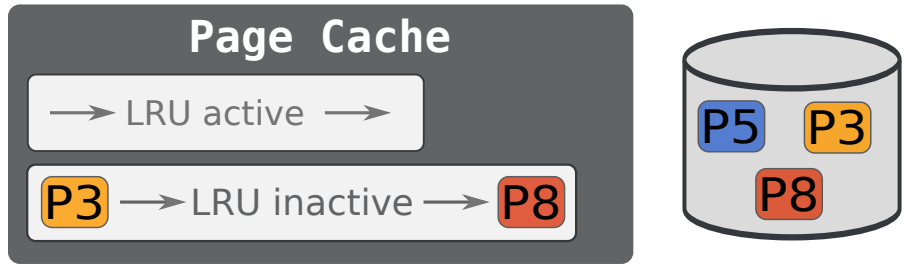
Ballooning

- Mémoire « donnée »
- Synchronisations $VM_1 \leftrightarrow$ hôte $\leftrightarrow VM_2$

Puma

- Mémoire « prêtée »
- Pages de cache *propres*

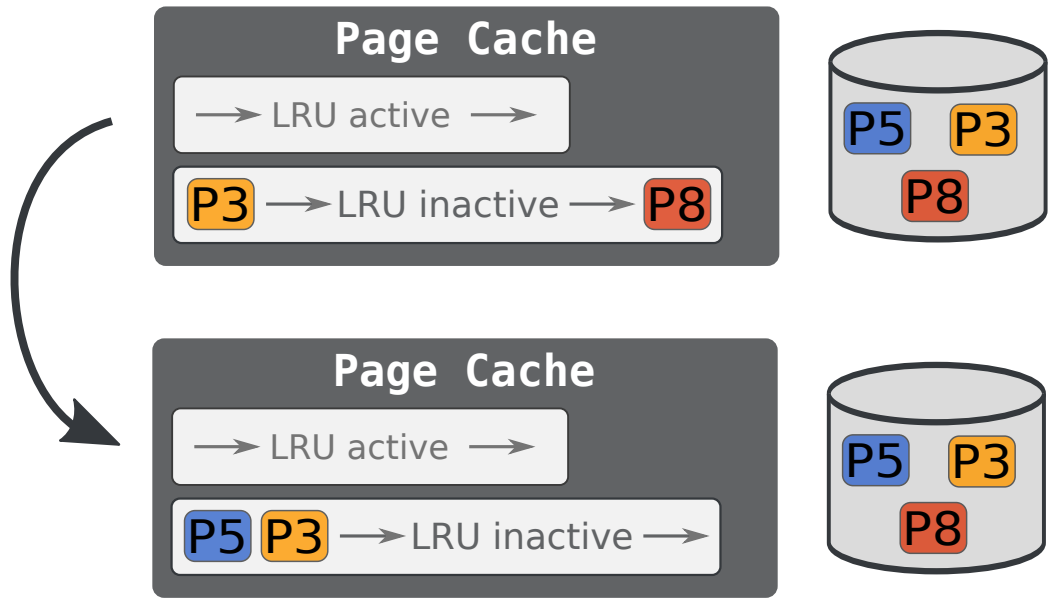
Prérequis #1 : double LRU du noyau Linux



Prérequis #1 : double LRU du noyau Linux

1ere lecture de P5 :

- Ajout de P5
- Eviction de P8



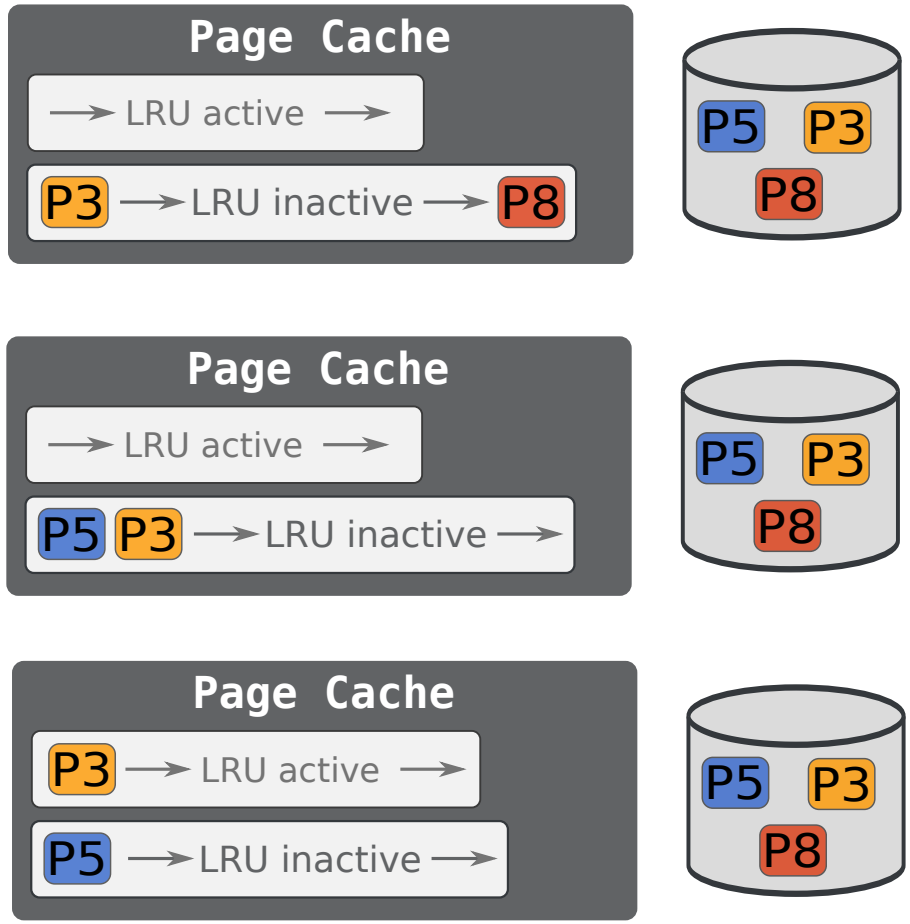
Prérequis #1 : double LRU du noyau Linux

1ere lecture de P5 :

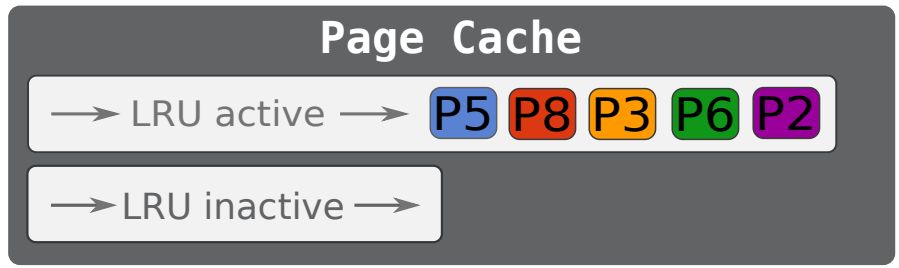
- Ajout de P5
- Eviction de P8

2eme lecture de P3 :

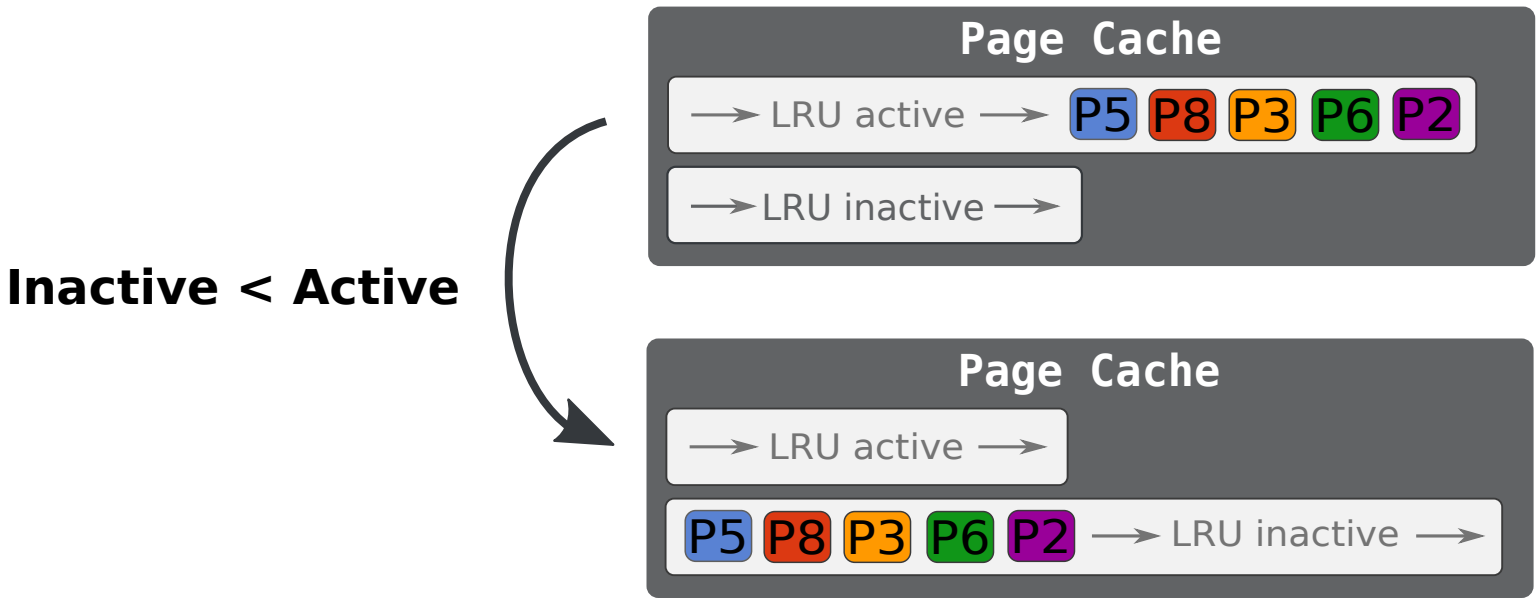
- Promotion de P3
- Pas d'éviction



Prérequis #2 : désactivation des pages



Prérequis #2 : désactivation des pages

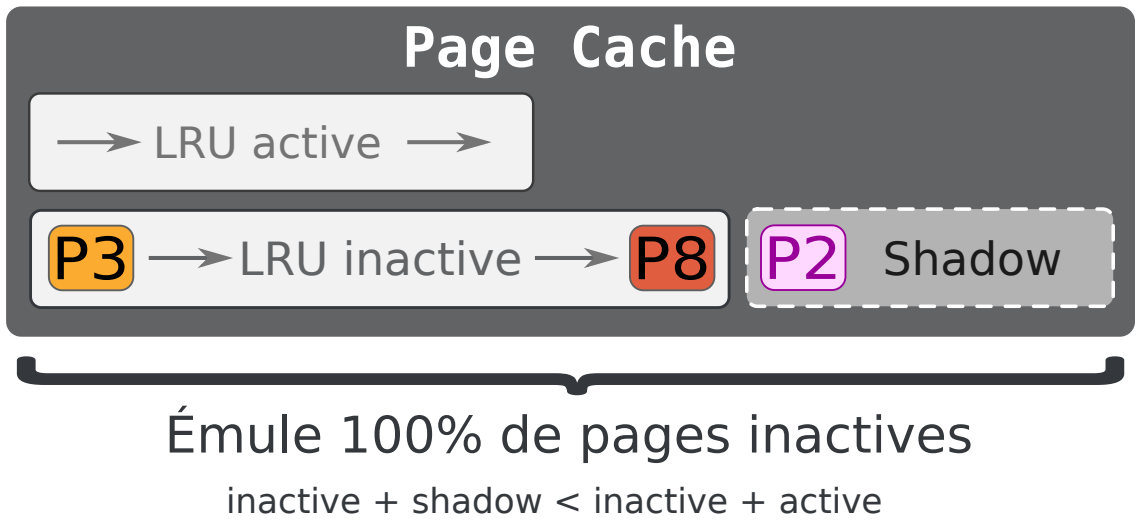


La LRU active ne représente pas plus de 50% du pagecache.

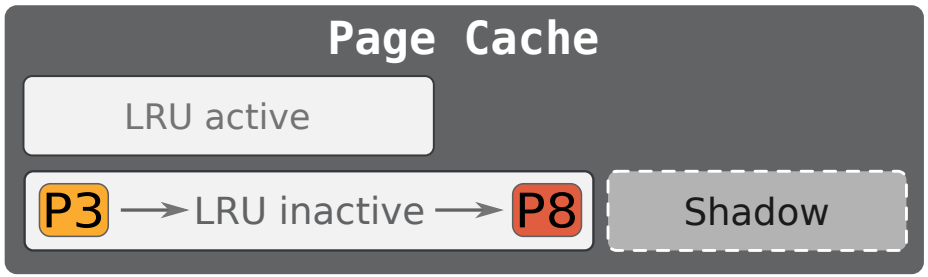
Prérequis #3 : *shadow* page cache

- **Problème :** en cas de pression mémoire, les pages inactives sont évincées trop rapidement pour être activées.

- **Solution dans le noyau 3.15 (juin 2014) :** des méta-données (shadow) sont gardées en mémoire lorsque les pages sont évincées.



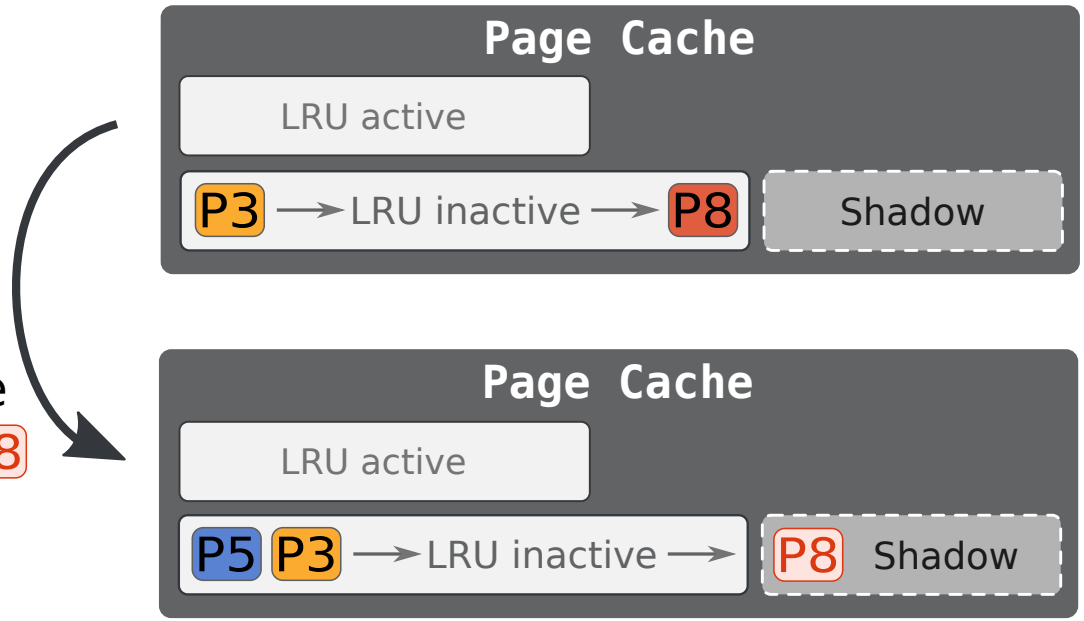
Prérequis #3 : *shadow* page cache (exemple)



Prérequis #3 : *shadow* page cache (exemple)

1ere lecture de P5 :

- Ajout de P5
- Eviction de P8
- Le shadow conserve les méta-données P8



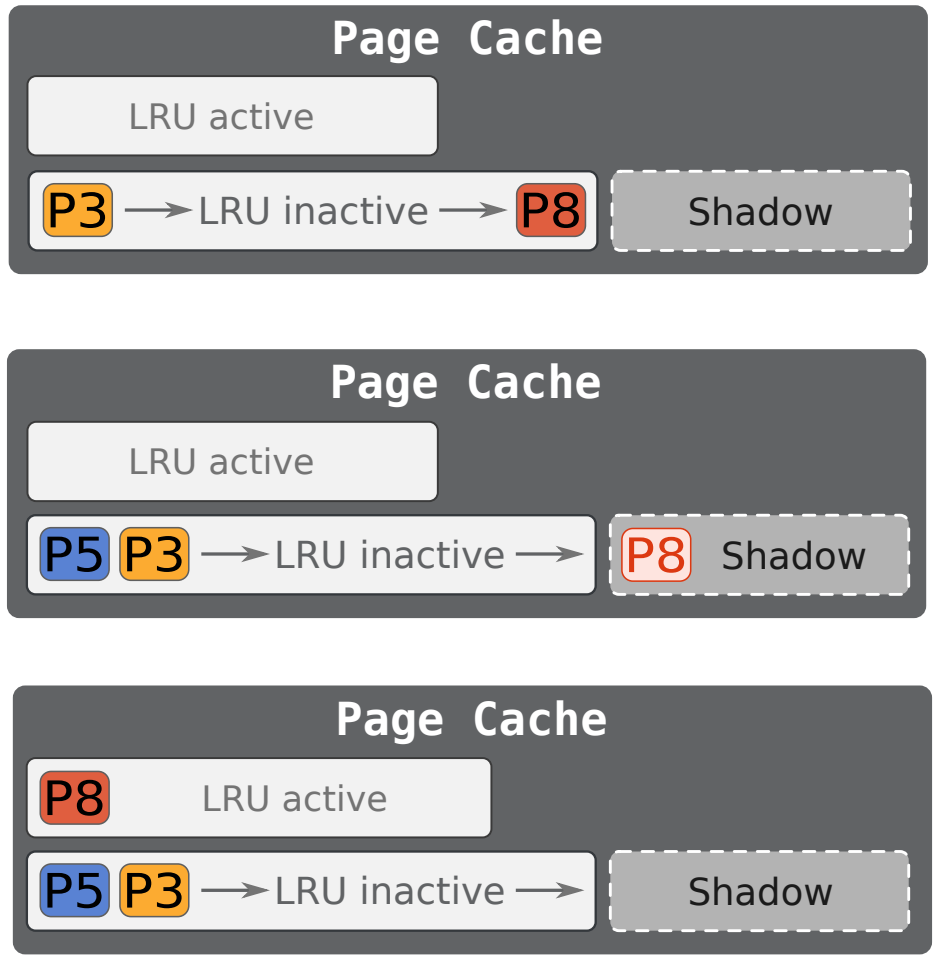
Prérequis #3 : *shadow* page cache (exemple)

1ere lecture de P5 :

- Ajout de P5
- Eviction de P8
- Le shadow conserve les méta-données P8

2eme lecture de P8 :

- Hit dans Shadow
- Activation de P8



Automatisation de Puma

Pour automatiser Puma, nous devons résoudre 3 problèmes

Problème 1

- Récupération du cache pour de la mémoire allouée

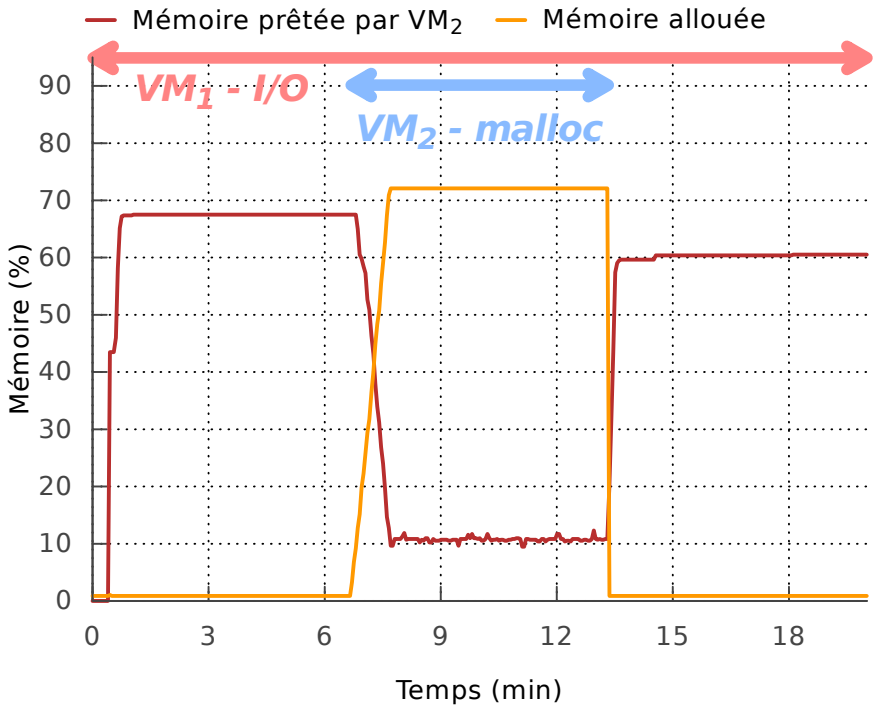
Problème 2

- Récupération de la mémoire pour du cache

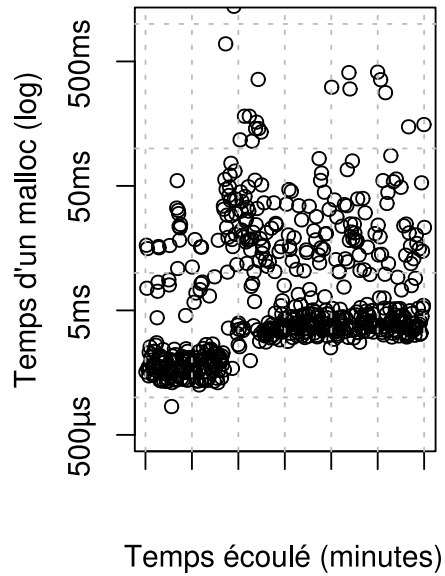
Problème 3

- Activité concurrente des VMs

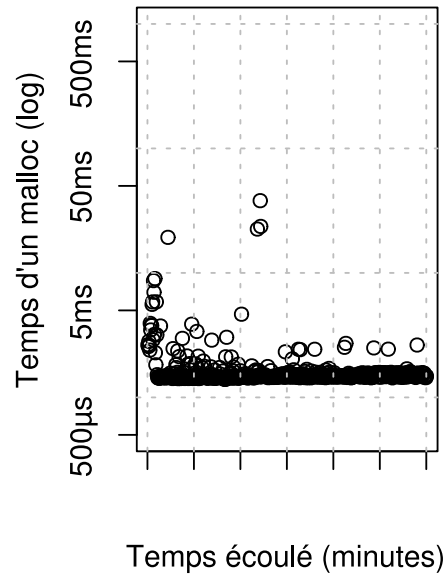
Problème 1 – Récupération du cache pour des allocations



Puma *(mémoire VM_2)*



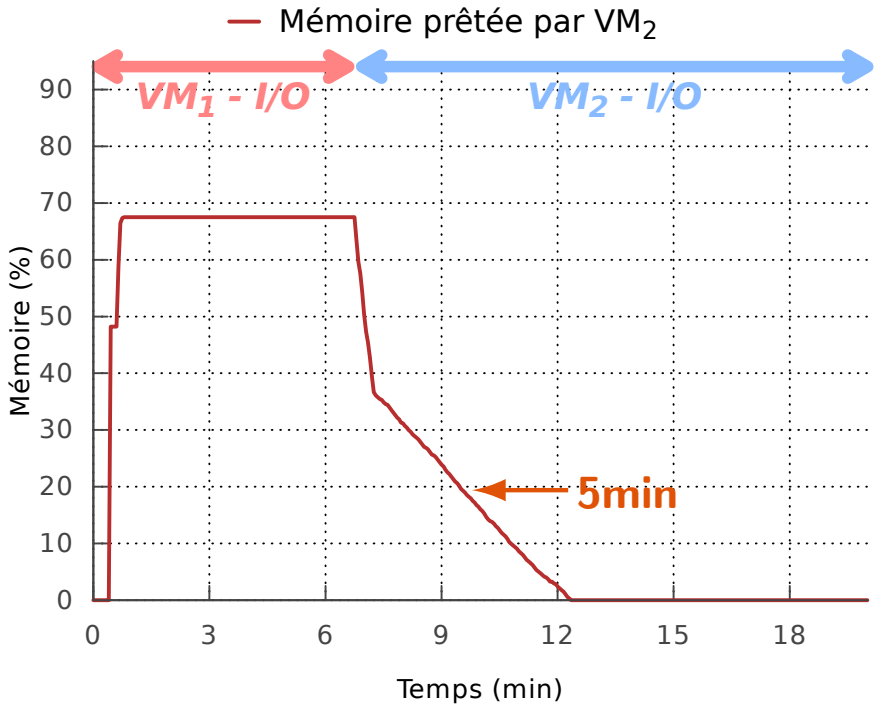
Ballooning (VM_2)



Puma *(VM_2)*

Puma est intégré au *page cache* ⇒ récupération rapide

Problème 2 – Récupération de la mémoire pour du cache

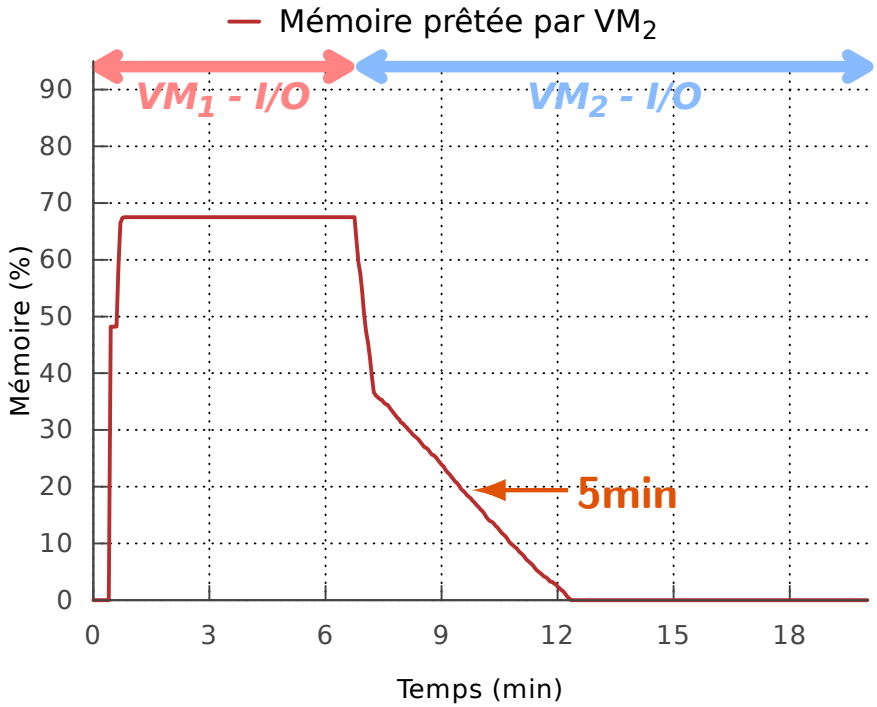


Puma *intégré au pagecache*

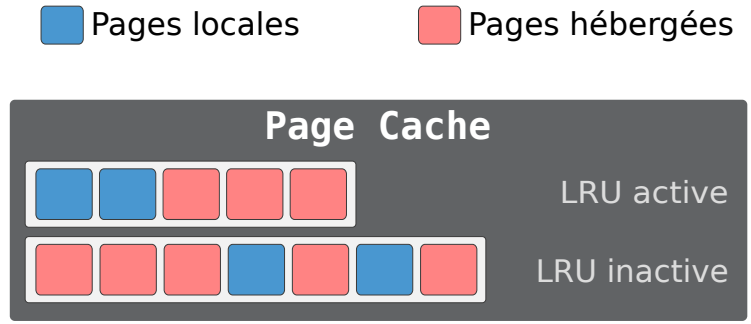
Problème

⇒ Vitesse de récupération

Problème 2 – Récupération de la mémoire pour du cache



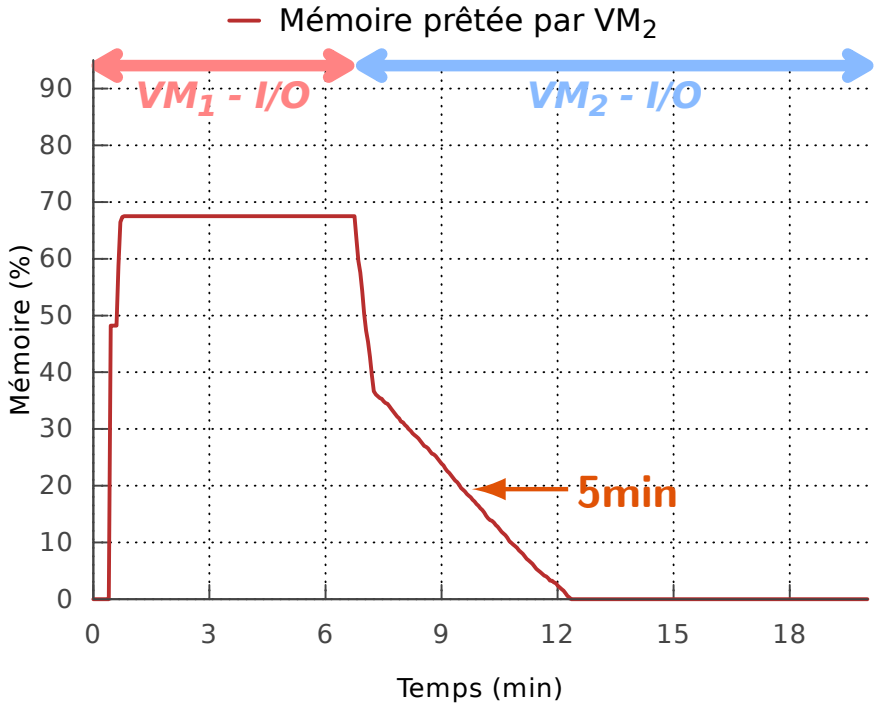
Puma *intégré au pagecache*



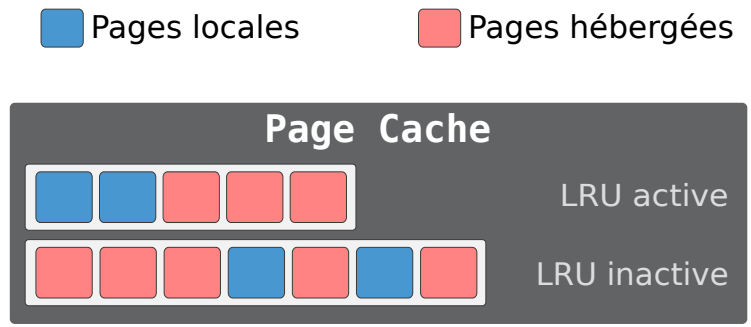
Problème

⇒ Vitesse de récupération

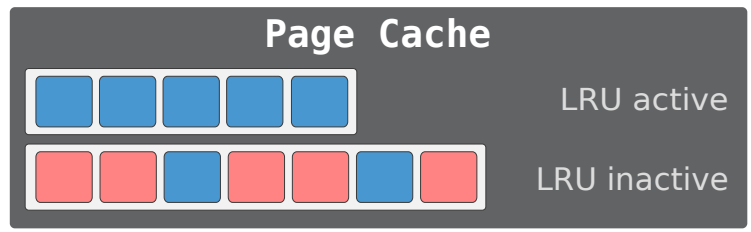
Problème 2 – Récupération de la mémoire pour du cache



Puma *intégré au pagecache*



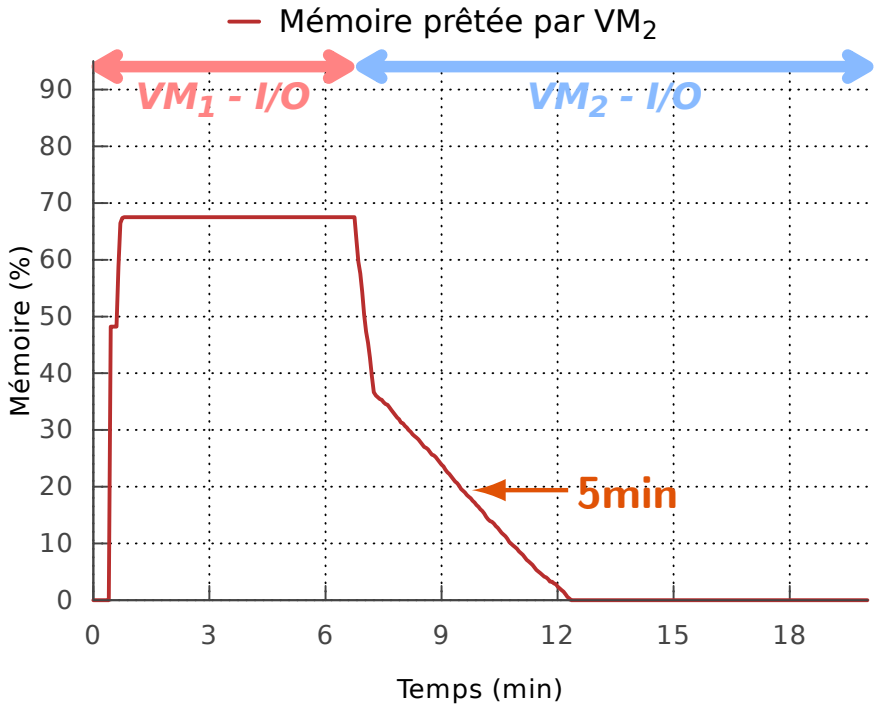
Solution : maintenir les pages distantes inactives



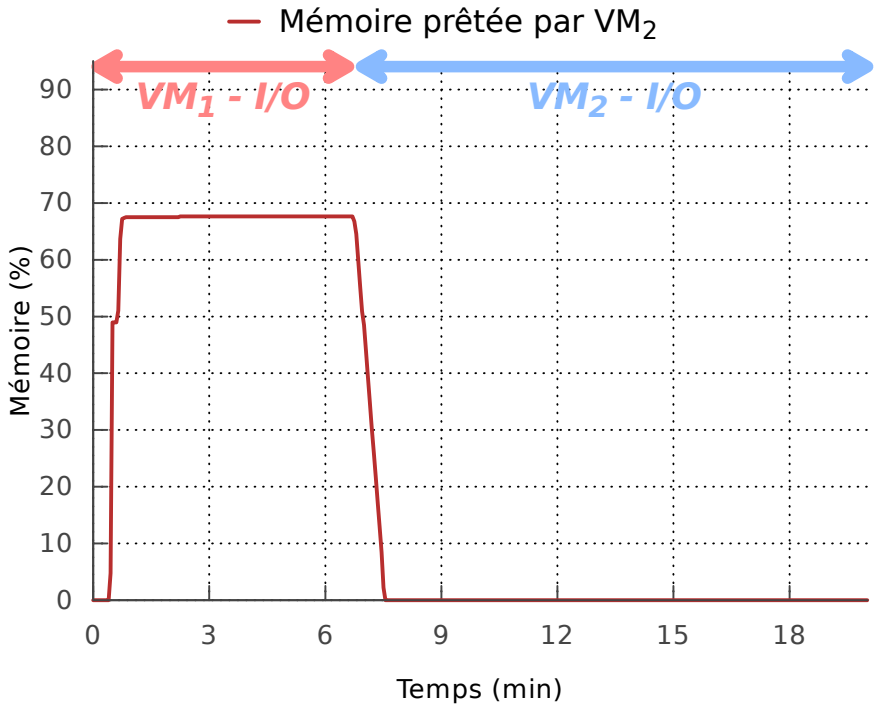
Problème

⇒ Vitesse de récupération

Problème 2 – Récupération de la mémoire pour du cache



Puma *intégré au pagecache*



Puma *limité aux pages inactives*

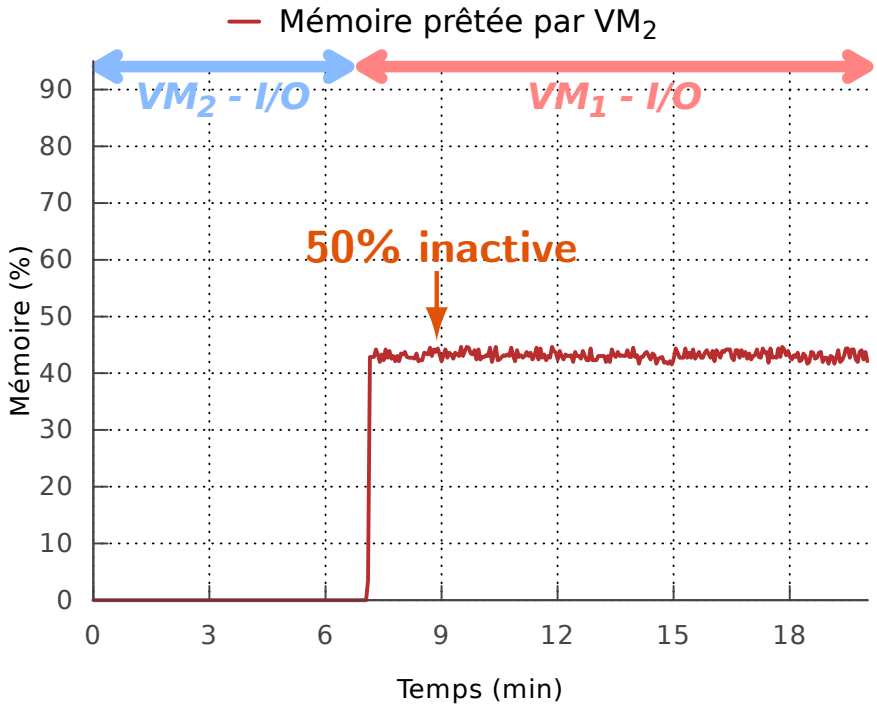
Problème

⇒ Vitesse de récupération

Solution

⇒ Donner plus de priorité aux pages locales

Problème 2 – Récupération de la mémoire pour du cache

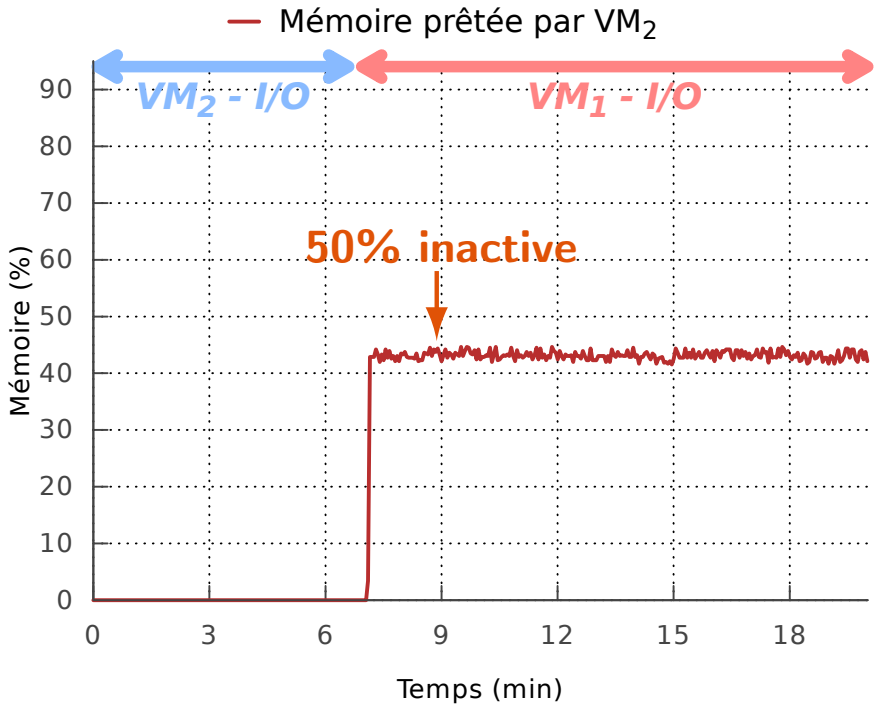


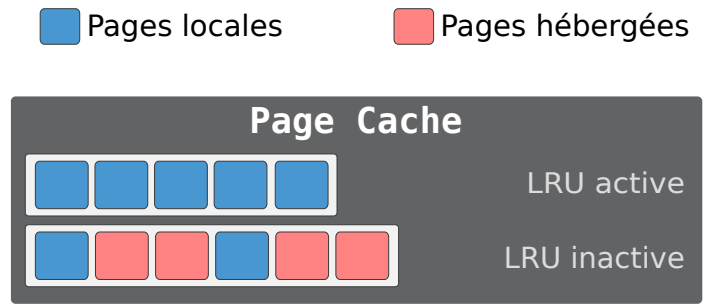
Puma *limité aux pages inactives*

Problème

⇒ Limitation à 50% du cache

Problème 2 – Récupération de la mémoire pour du cache



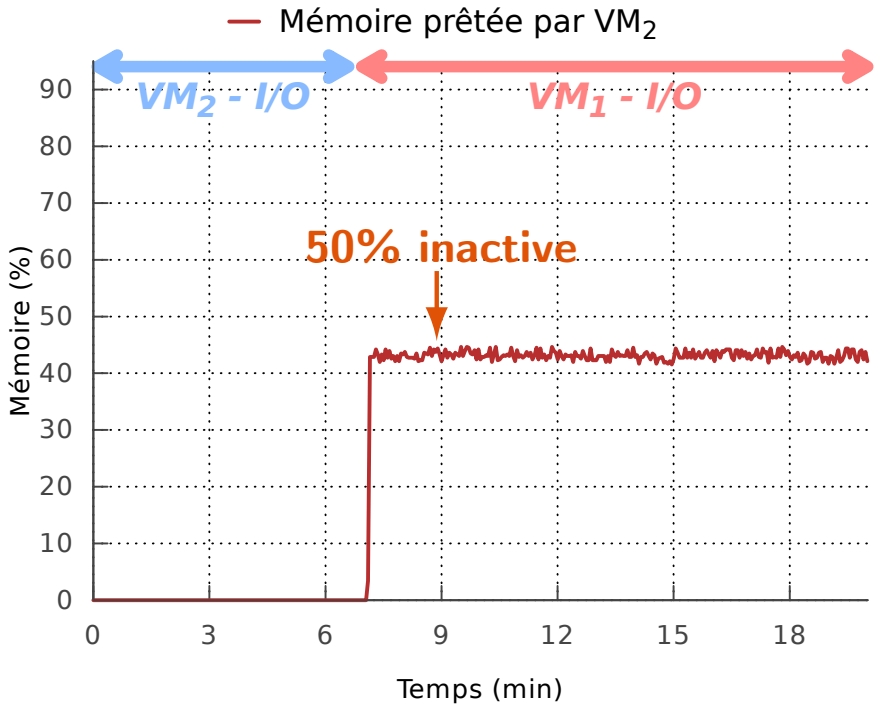


Puma *limité aux pages inactives*

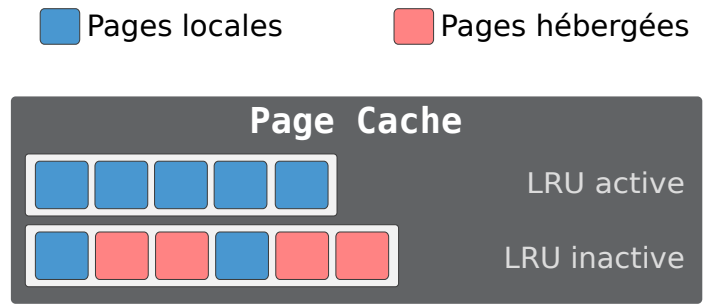
Problème

⇒ Limitation à 50% du cache

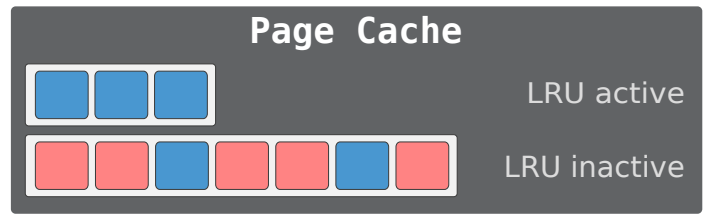
Problème 2 – Récupération de la mémoire pour du cache



Puma *limité aux pages inactives*



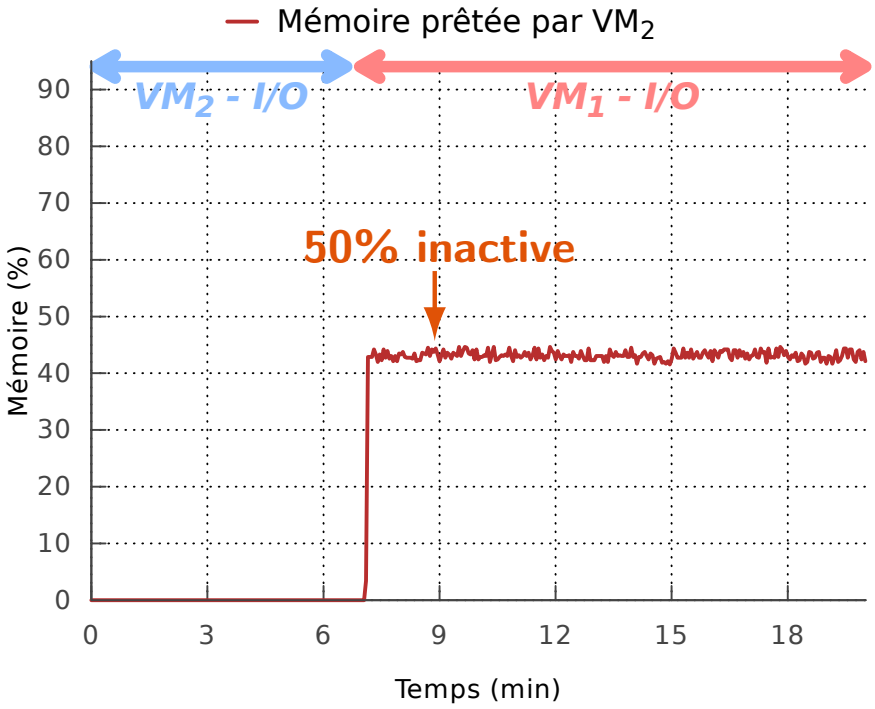
Solution : forcer la désactivation



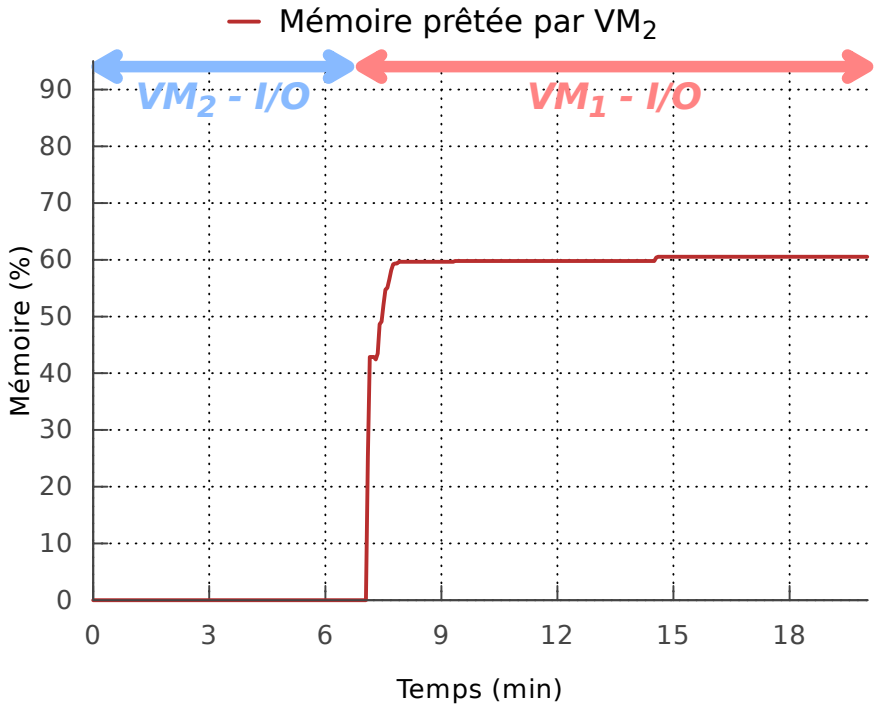
Problème

⇒ Limitation à 50% du cache

Problème 2 – Récupération de la mémoire pour du cache



Puma *limité aux pages inactives*



Désactivation des pages actives forcée

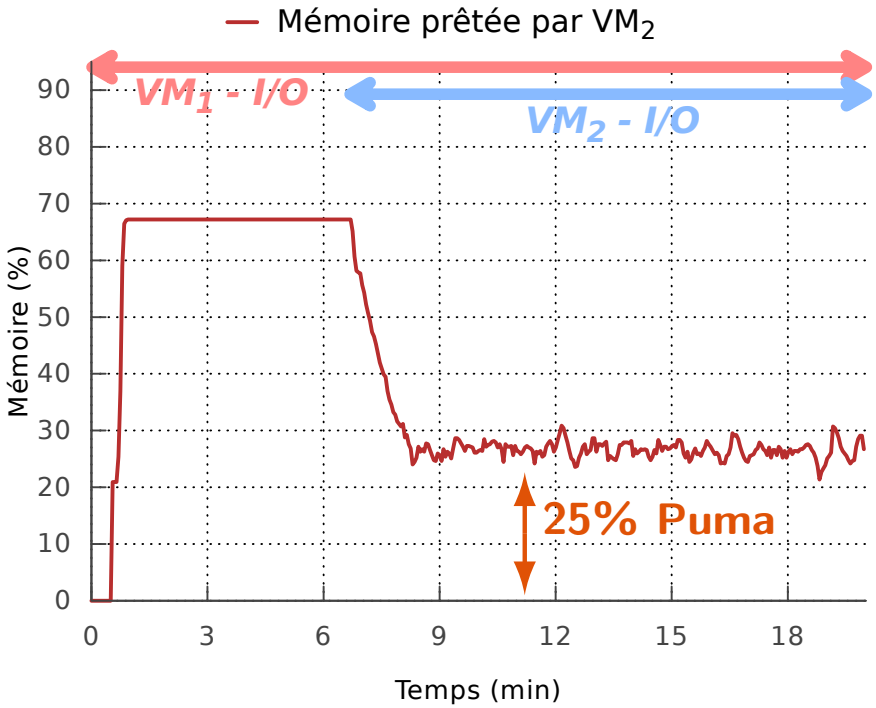
Problème

⇒ Limitation à 50% du cache

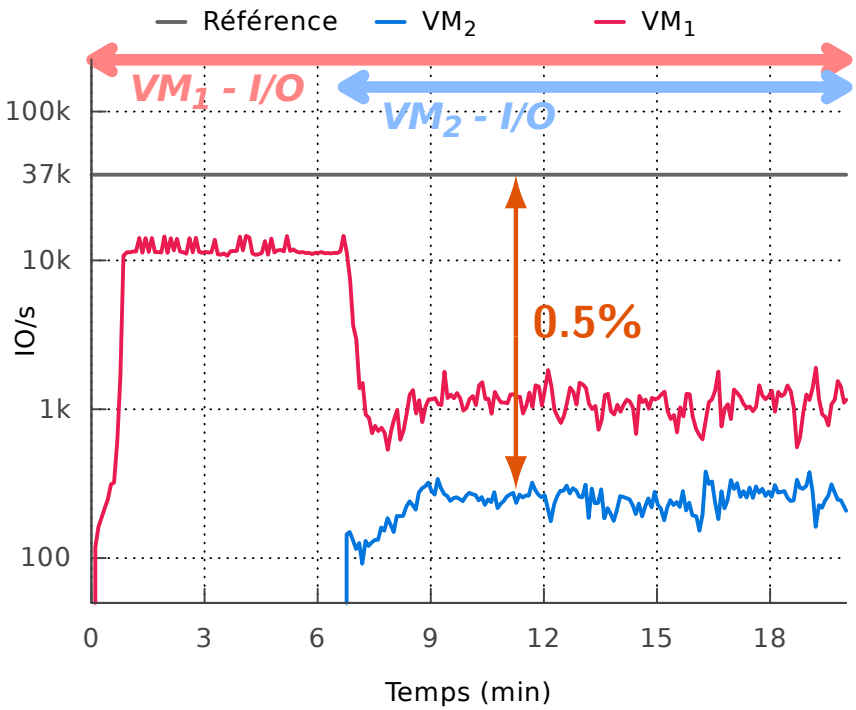
Solution

⇒ Forcer la désactivation des pages

Problème 3 – Activité concurrente des VMs



Puma *(mémoire VM_2)*



I/O par secondes

Une VM doit conserver la priorité pour ses besoins

⇒ Comment détecter l'(in)activité d'une VM ?

Détection de l'activité

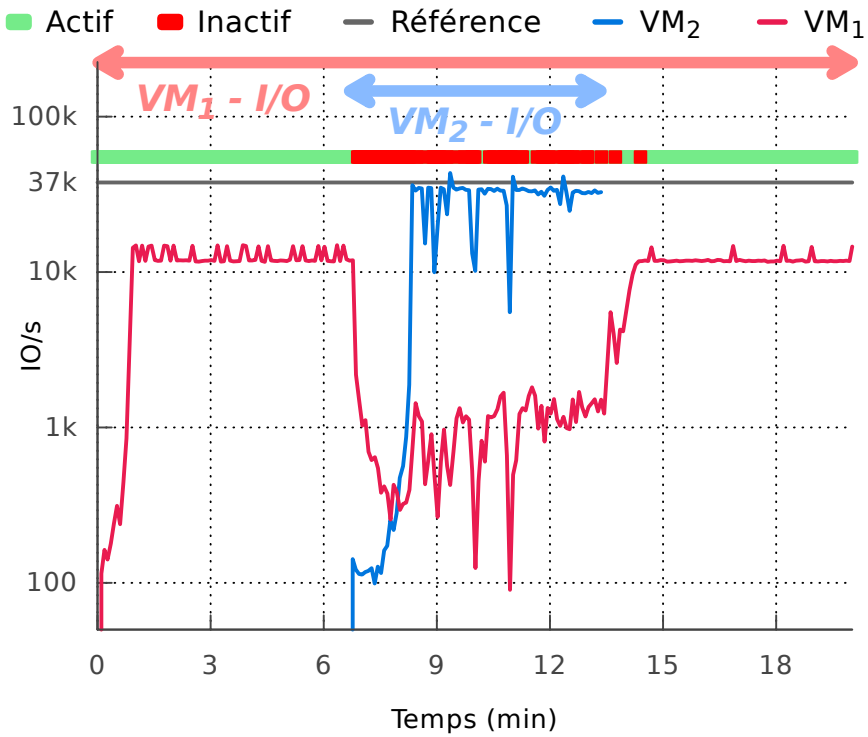
Détecteur #1 : *shadow*

- Hit shadow → arrêt de Puma

Détection de l'activité

Détecteur #1 : *shadow*

- Hit shadow → arrêt de Puma

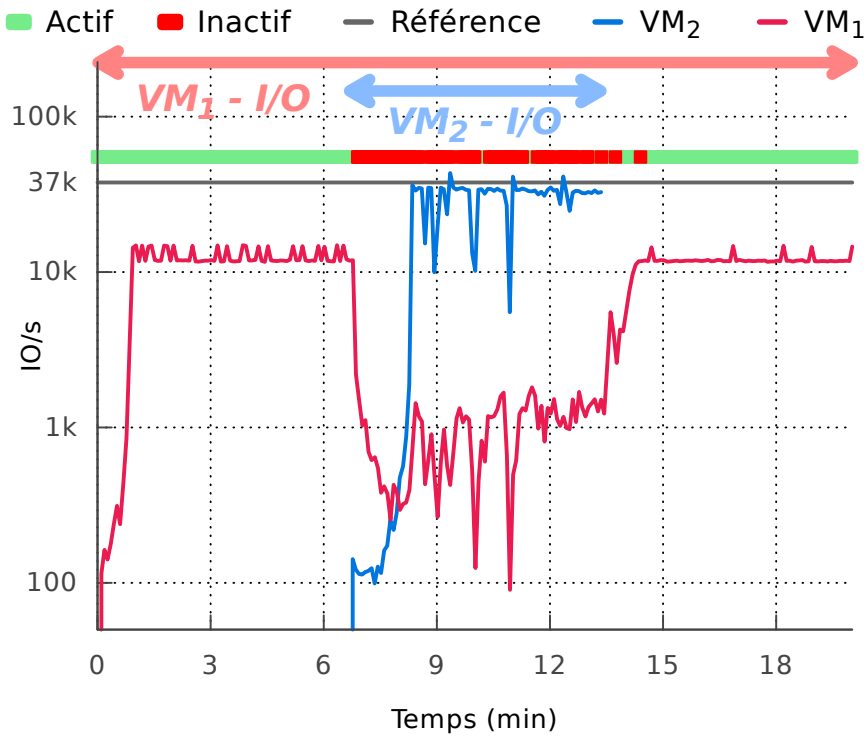


- Détection rapide
- Instable

Détection de l'activité

Détecteur #1 : *shadow*

- Hit shadow → arrêt de Puma



- Détection rapide
- Instable

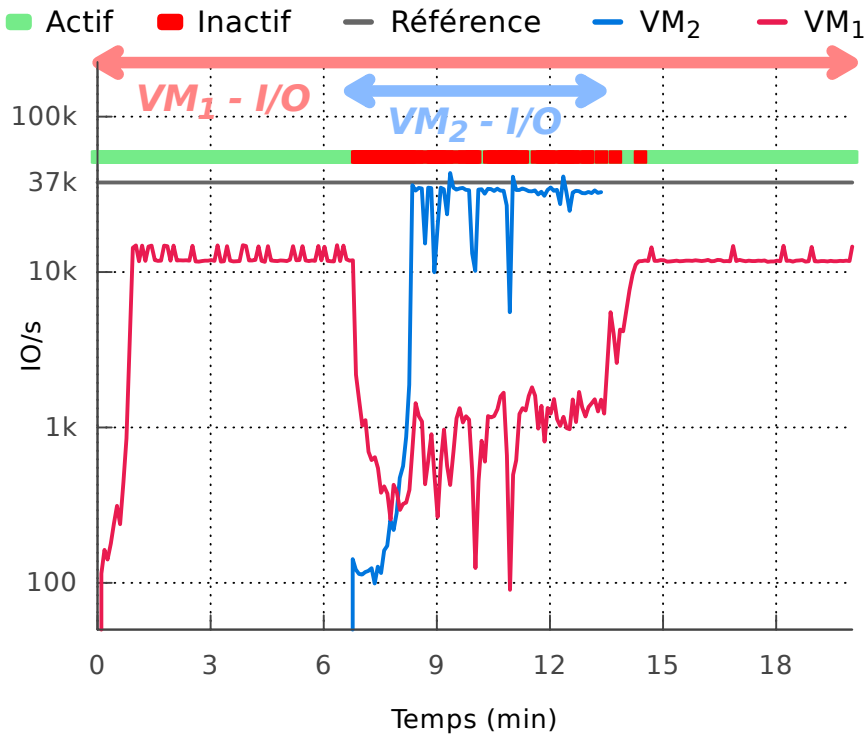
Détecteur #2 : *pression mémoire*

- Page désactivée → arrêt de Puma

Détection de l'activité

Détecteur #1 : *shadow*

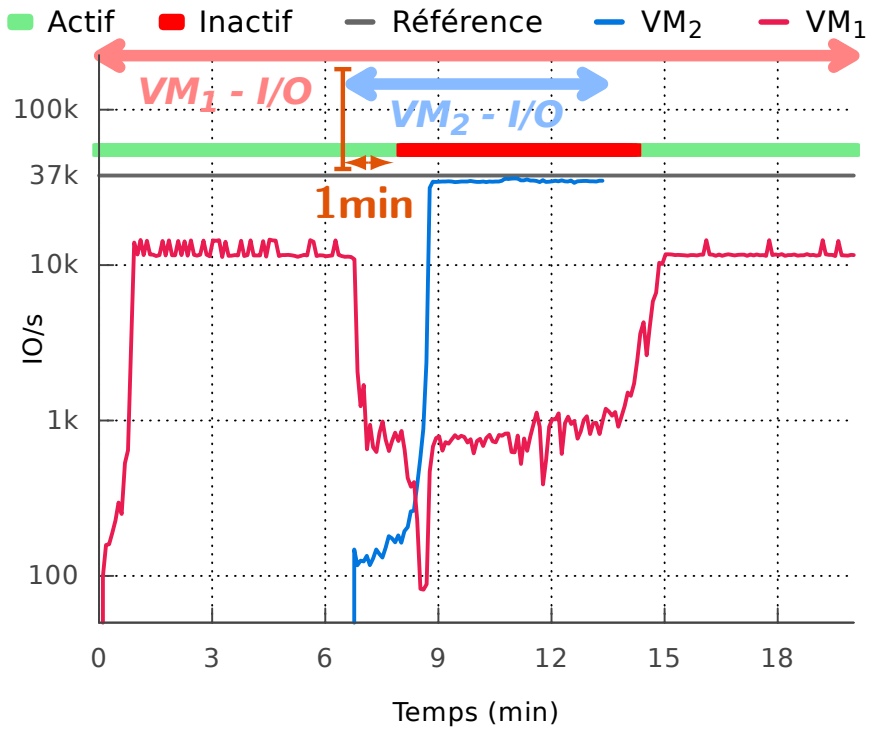
- Hit shadow → arrêt de Puma



- Détection rapide
- Instable

Détecteur #2 : *pression mémoire*

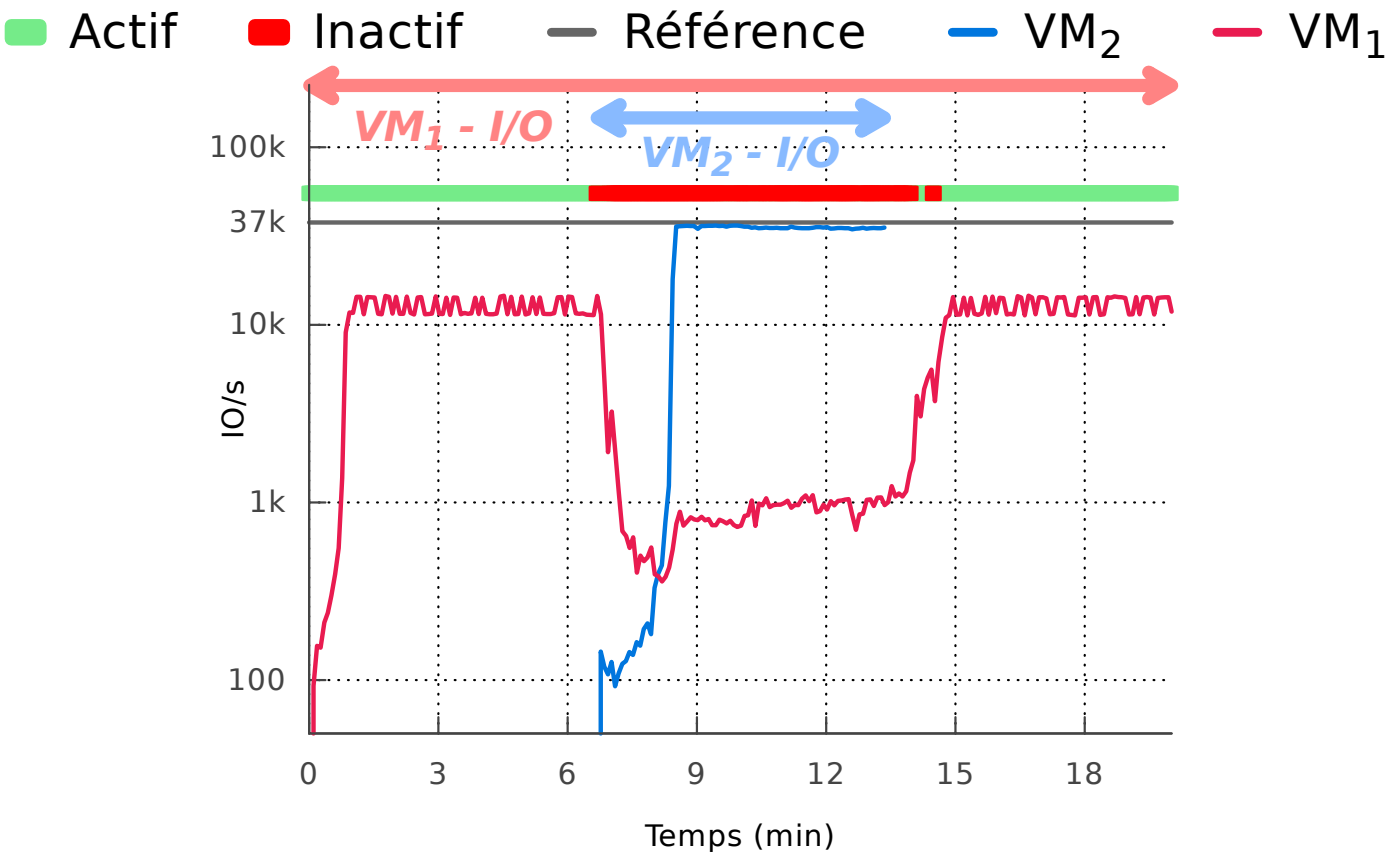
- Page désactivée → arrêt de Puma



- Vitesse de détection
- Stable

Détecteur #3 : approche combinée

Proposition : combiner les deux approches



- Détection rapide (#1)
- Stable (#2)

Résumé de la contribution

Automatisation de Puma

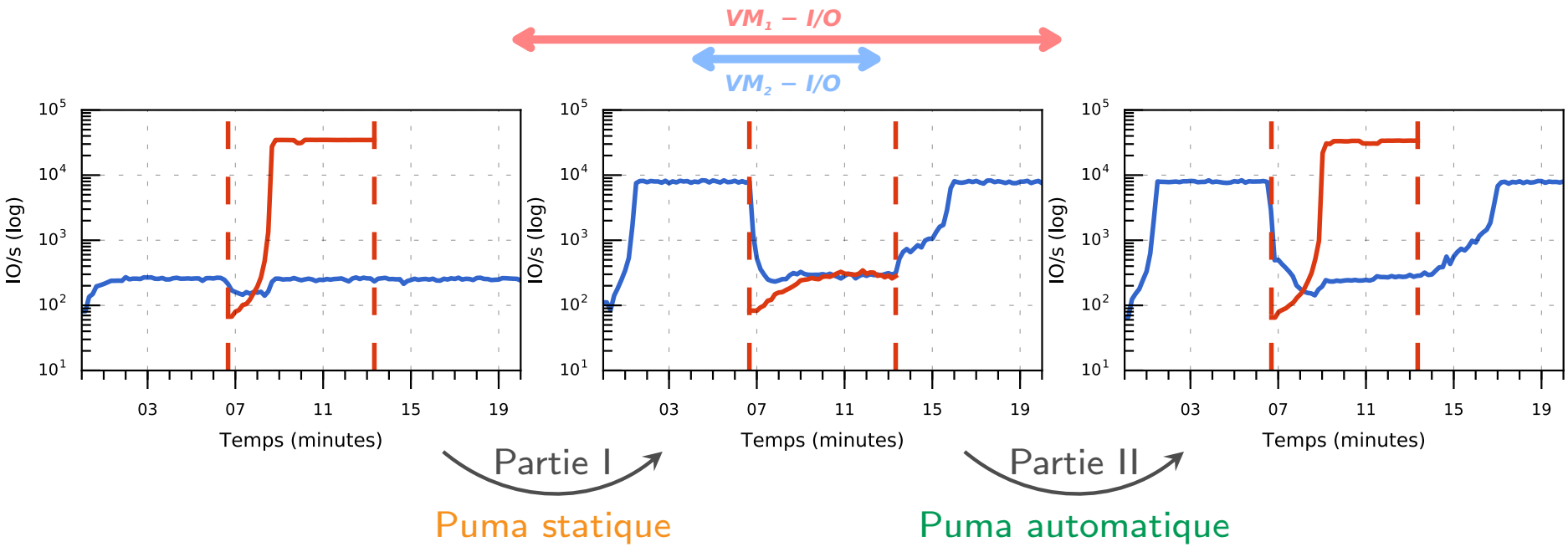
- Peu intrusif
- Récupération efficace de la mémoire prêtée pour des allocations
  - → Divise par 10 le surcout comparé au ballooning automatique
- Activité concurrente
  - → Détection de l'activité mémoire d'une VM
  - → Désactivation de Puma en cas d'activité
  - → Réactivation de Puma lors d'une baisse d'activité

Conclusion

Résumé de la thèse

Puma : Pooling Unused memory in virtual MAchines

1. Réduit de la fragmentation mémoire entre VMs
2. Conserve l'isolation des VMs
3. Approche noyau, peu intrusive, transparente
4. Automatique



Perspectives

Court terme

- Paravirtualisation de Puma pour se passer du coût réseau

Extension de Puma

- Extension de Puma aux écritures synchrones

Ouvertures

- Adaptation du ballooning au cache
- Fragmentation de la mémoire entre conteneurs