Gestion dynamique du cache entre machines virtuelles

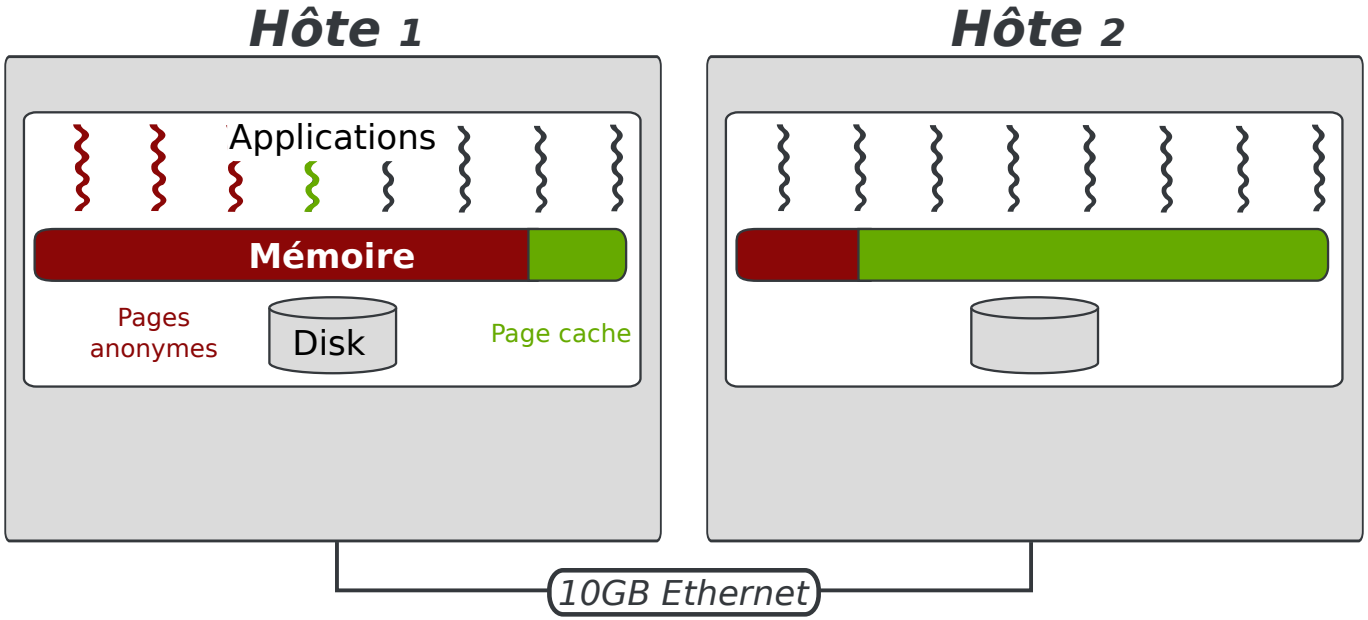
Maxime Lorrillere, Joel Poudroux, Julien Sopena et Sébastien Monnet

Conférence d'informatique en Parallélisme, Architecture et Système

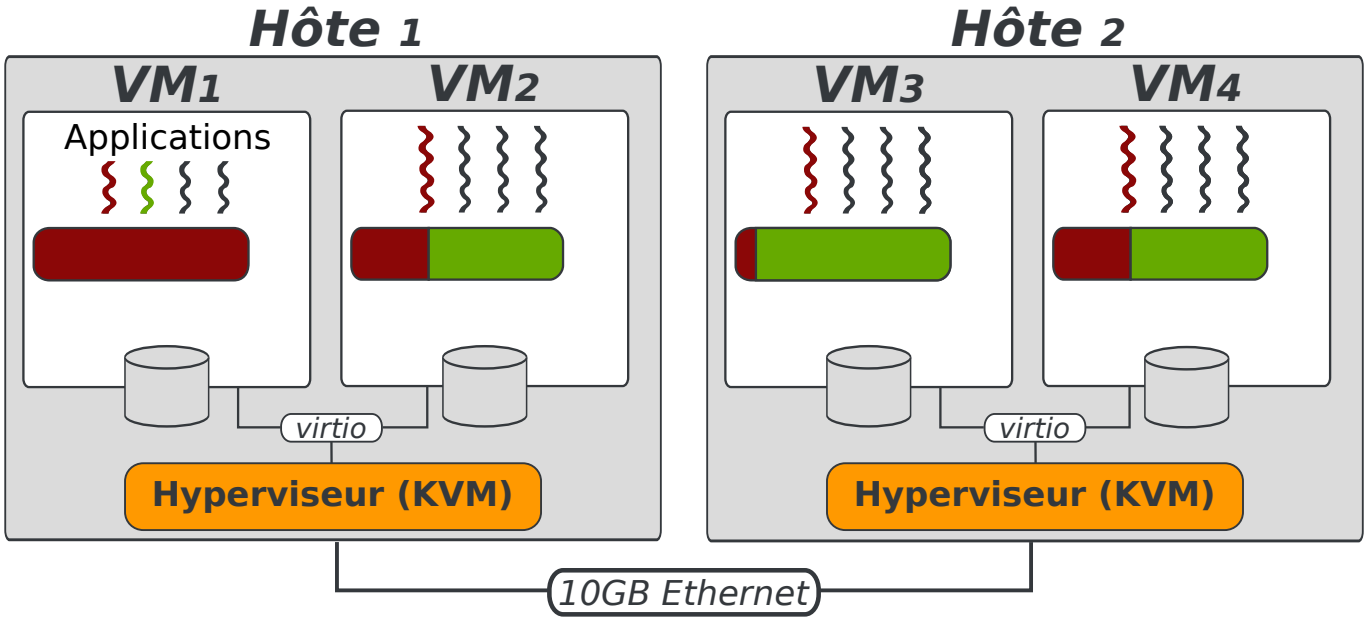
30 juin – 3 juillet 2015

Lille, France

Virtualisation et fragmentation de la mémoire

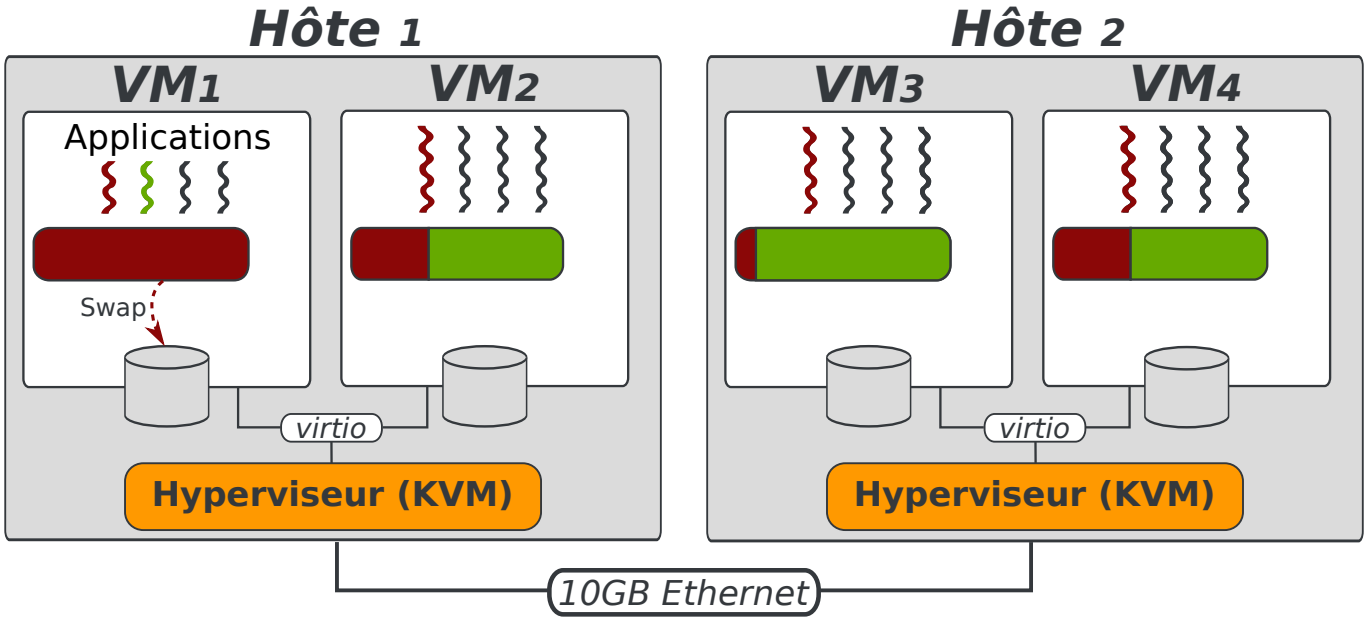


Virtualisation et fragmentation de la mémoire



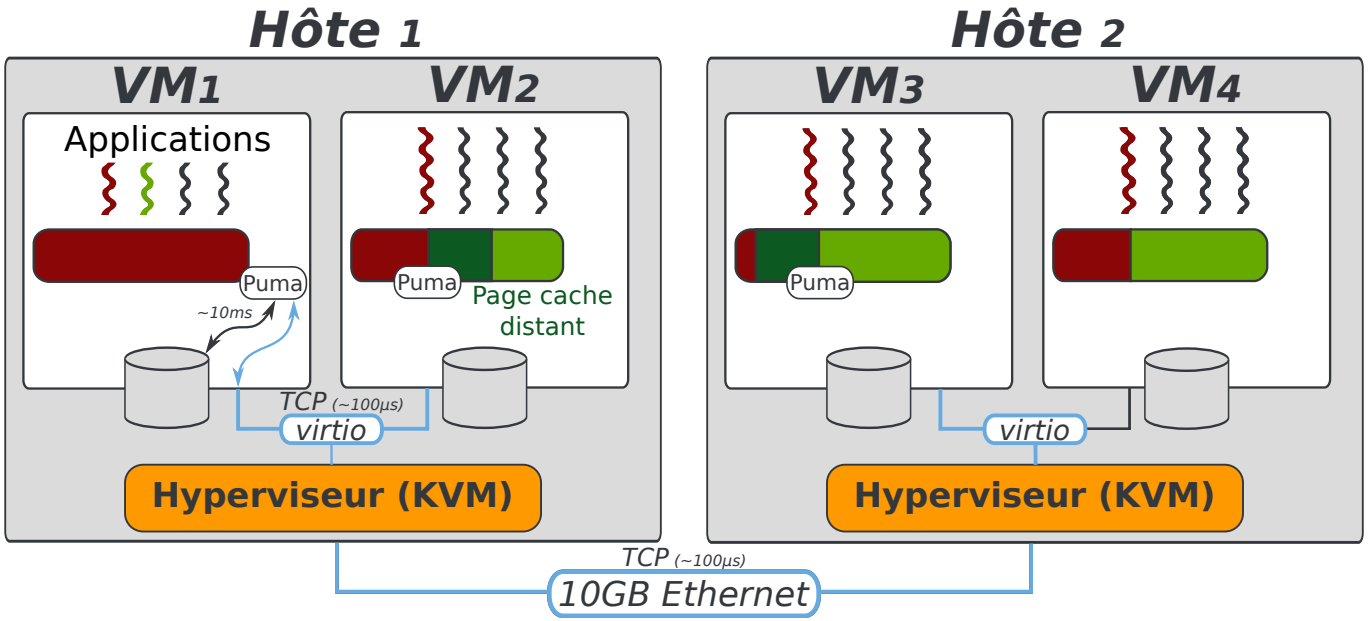
- La virtualisation apporte flexibilité et isolation

Virtualisation et fragmentation de la mémoire



- La virtualisation apporte flexibilité et isolation
  Problème : fragmentation de la mémoire disponible
  - ⇒ Partager des ressources comme le temps CPU est facile
  - ⇒ La mémoire ne peut pas être réaffectée aussi efficacement que le temps CPU (*swap*)

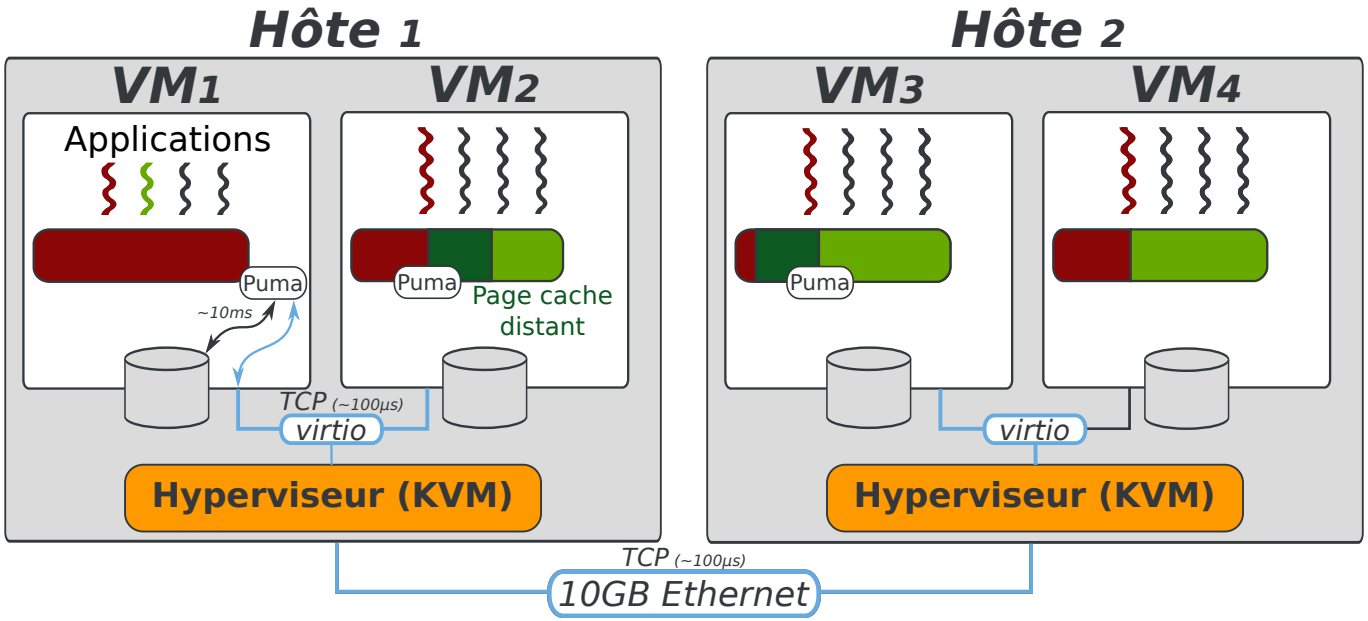
Solution : PUMA [ComPAS'14]



Approche de PUMA : un cache réparti entre VMs

- Repose sur un réseau performant entre les VMs et les hôtes
- Intégré directement au sein du *page cache* du noyau Linux
  - ⇒ Agnostique à l'hyperviseur et aux systèmes de fichiers

Solution : PUMA [ComPAS'14]

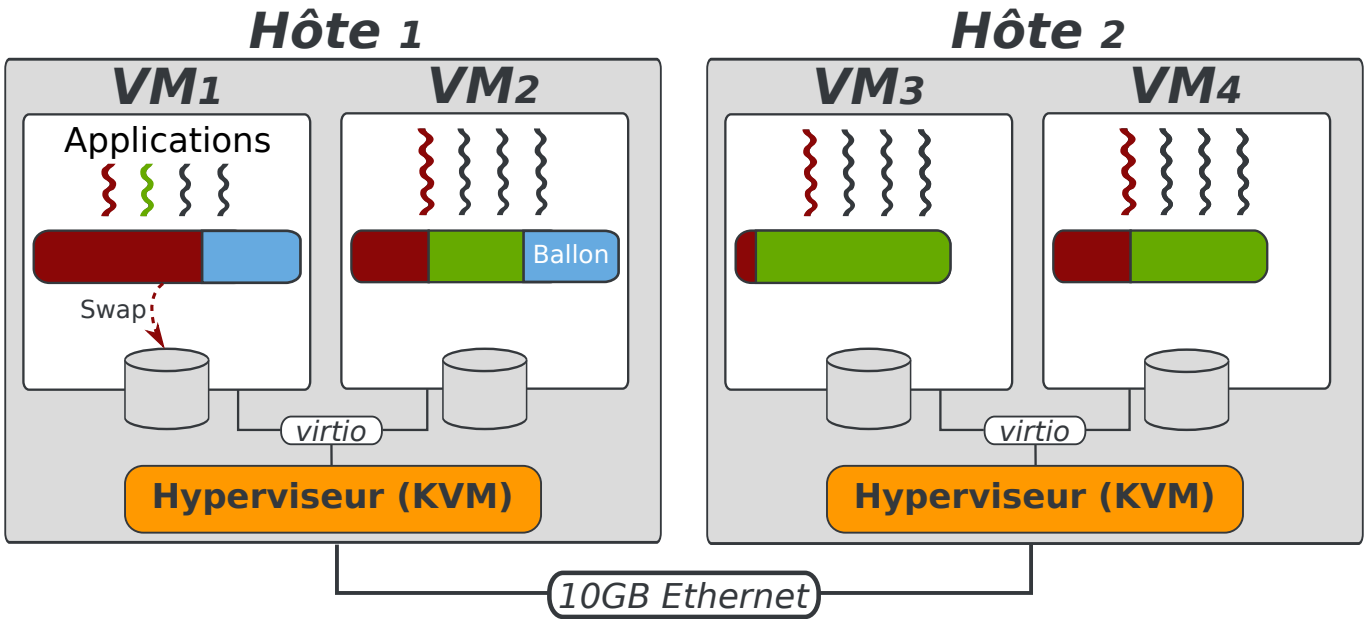


Approche de PUMA : un cache réparti entre VMs

- Repose sur un réseau performant entre les VMs et les hôtes
- Intégré directement au sein du *page cache* du noyau Linux
  - ⇒ Agnostique à l'hyperviseur et aux systèmes de fichiers

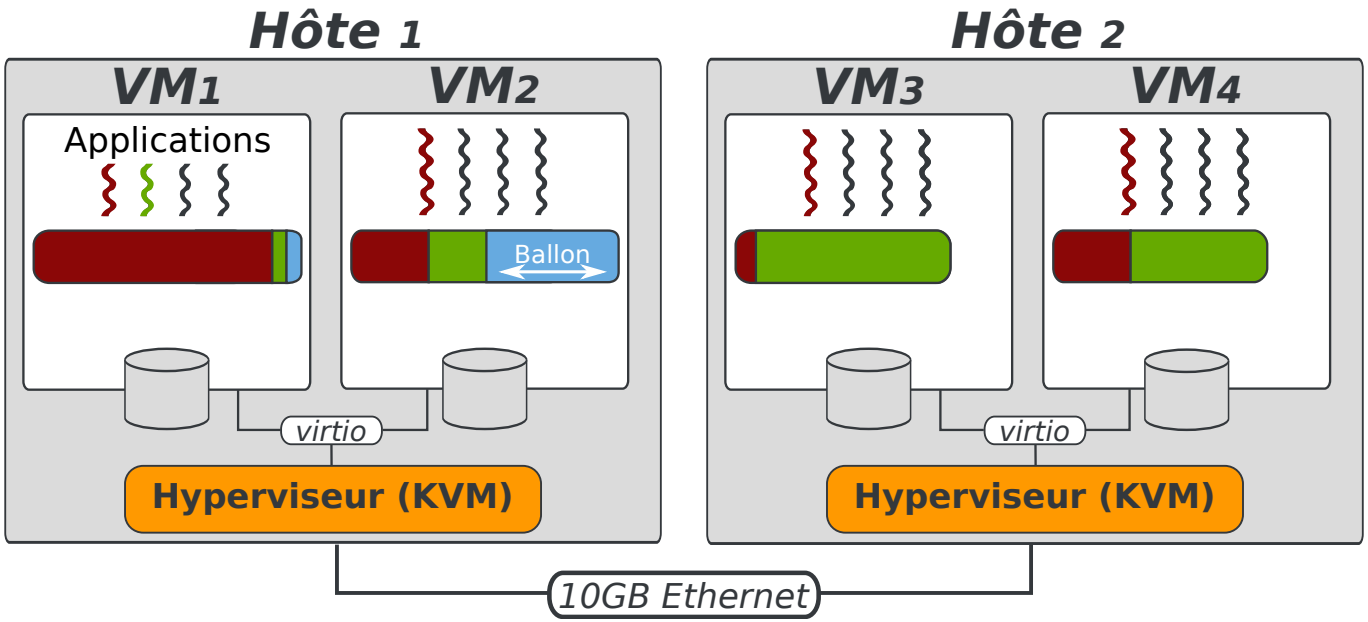
Problème : la configuration de PUMA est statique

Approche dynamique : ballooning automatique



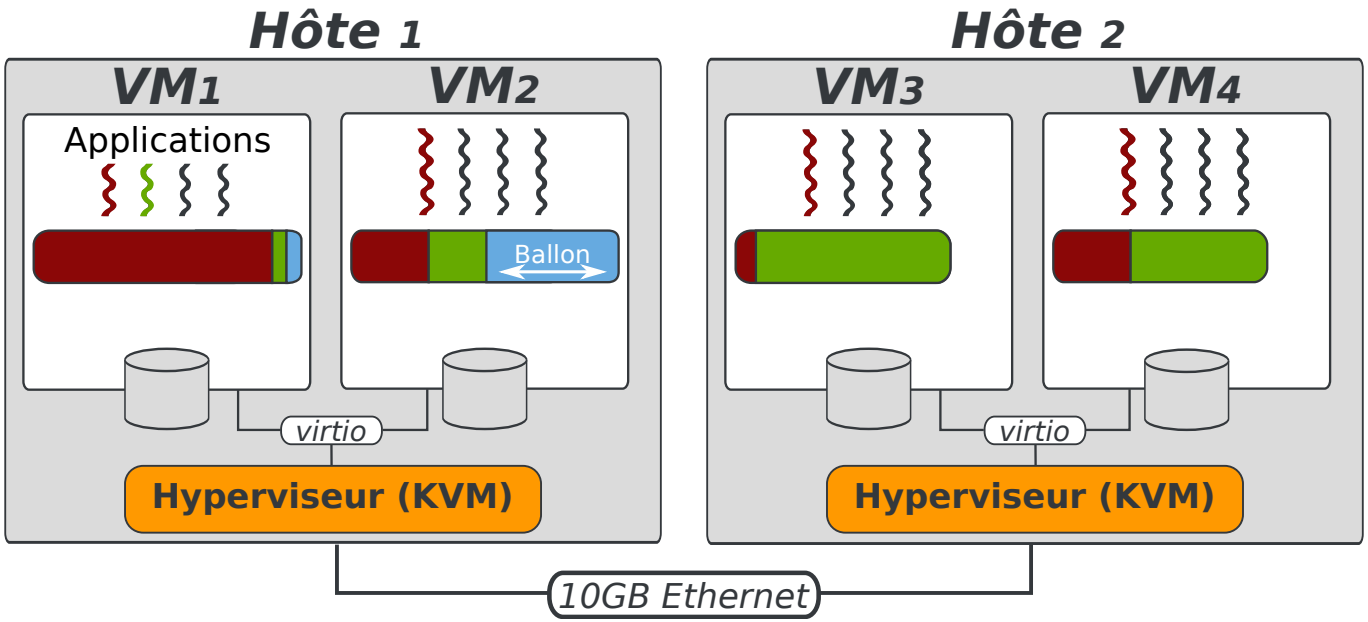
- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Approche dynamique : ballooning automatique



- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Approche dynamique : ballooning automatique

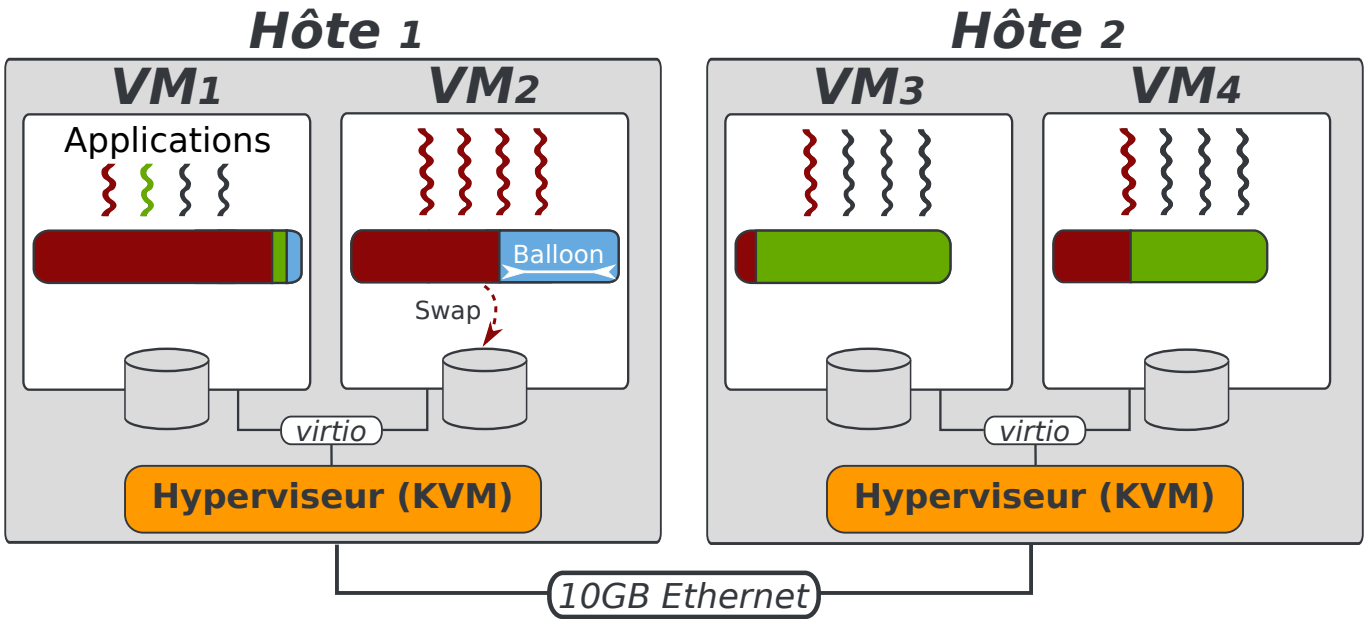


- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Limitations

⇒ *Gap sémantique* hyperviseur/VMs : difficile de tenir compte du cache

Approche dynamique : ballooning automatique



- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Limitations

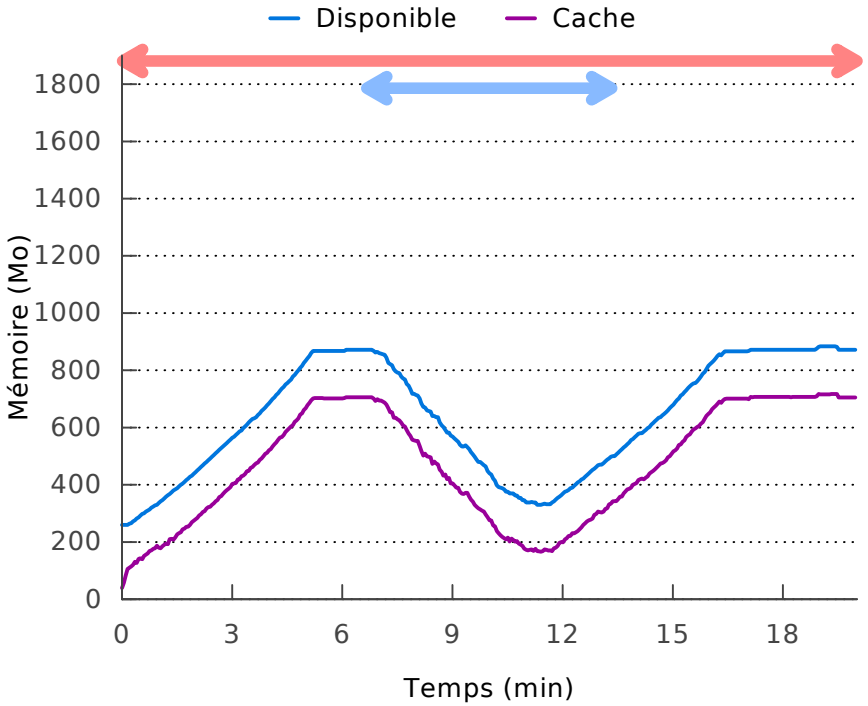
⇒ *Gap sémantique* hyperviseur/VMs : difficile de tenir compte du cache

⇒ Récupération lente de la mémoire (*swap*)

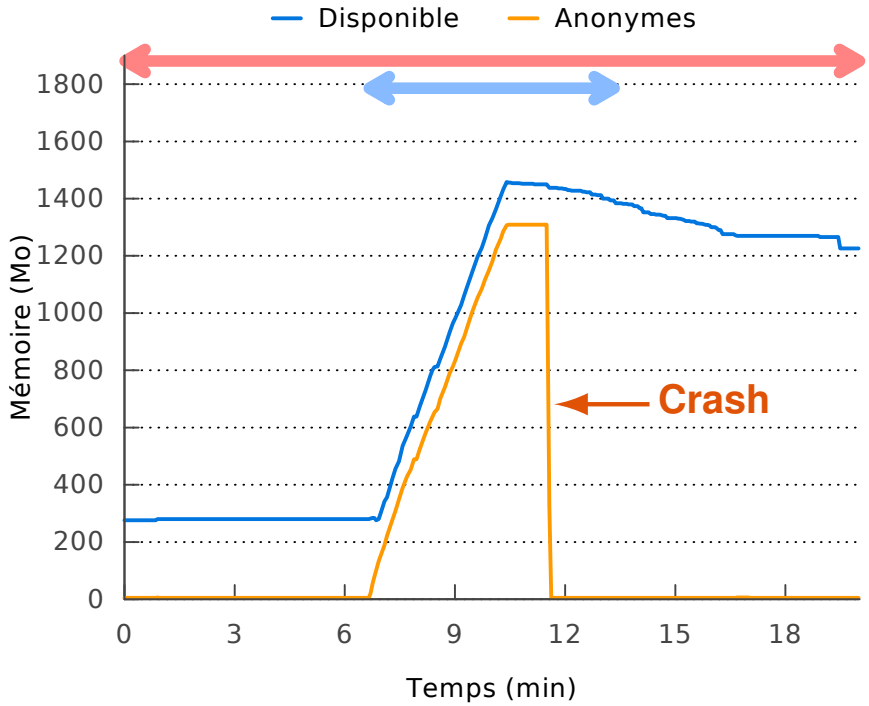
Auto-ballooning : récupération de la mémoire (KVM)

↔ Client : application I/O intensive (filebench)

↔ Serveur : malloc par bloc de 8M à partir de ~7min



Client (mémoire)



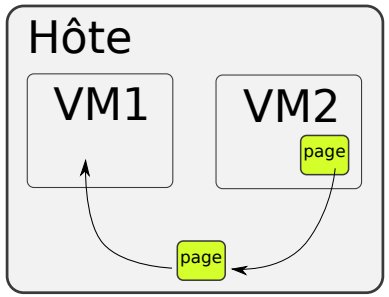
Serveur (mémoire)

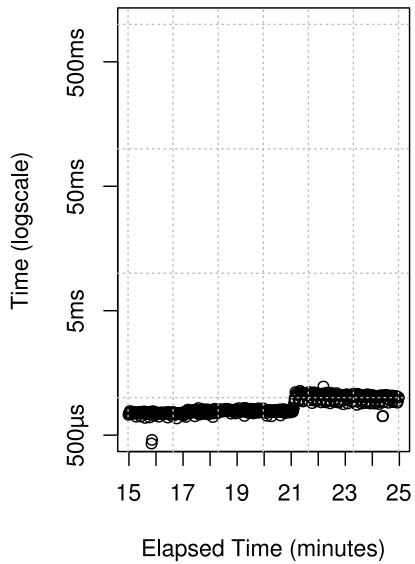
Forte latence : { malloc ralentis (3 min) + crash ; reprise très lente (5 min)

Auto-ballooning : récupération de la mémoire (KVM)

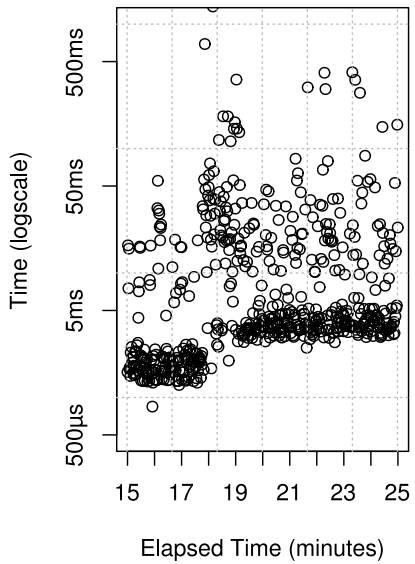
La récupération nécessite 4 étapes :

1. VM1 en saturation mémoire, demande à l'hôte
2. Si hôte saturé : gonflage du ballon de VM2
3. Dégonflage de VM1
4. Allocation de la page





malloc (référence)

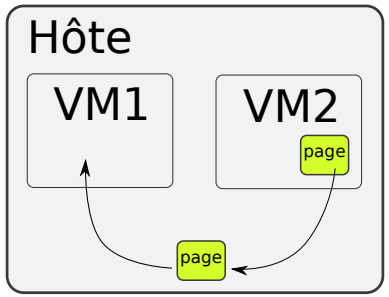


malloc (auto-ballooning)

Auto-ballooning : récupération de la mémoire (KVM)

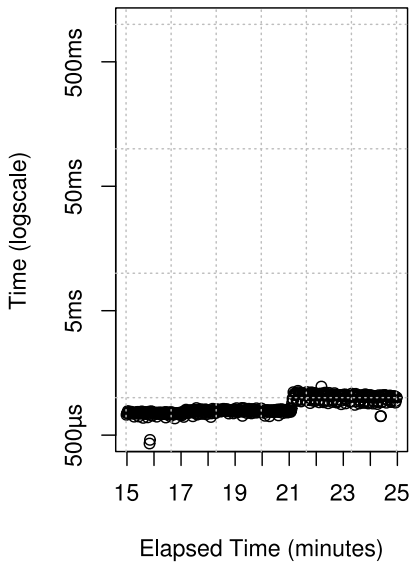
La récupération nécessite 4 étapes :

1. VM1 en saturation mémoire, demande à l'hôte
2. Si hôte saturé : gonflage du ballon de VM2
3. Dégonflage de VM1
4. Allocation de la page

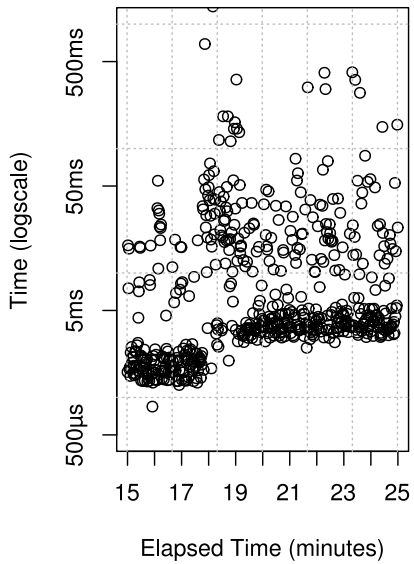


Une approche se concentrant sur le cache est nécessaire.

Solution : améliorer PUMA

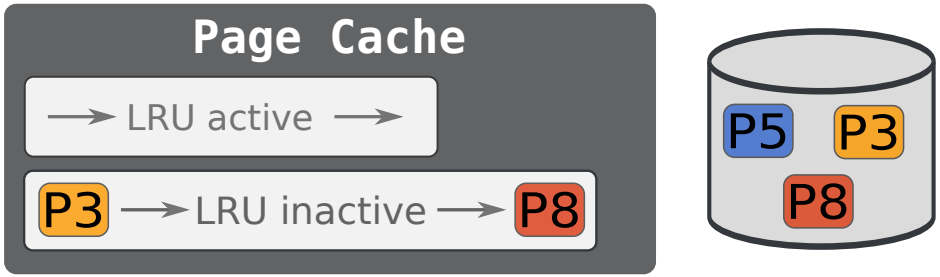


malloc (référence)



malloc (auto-ballooning)

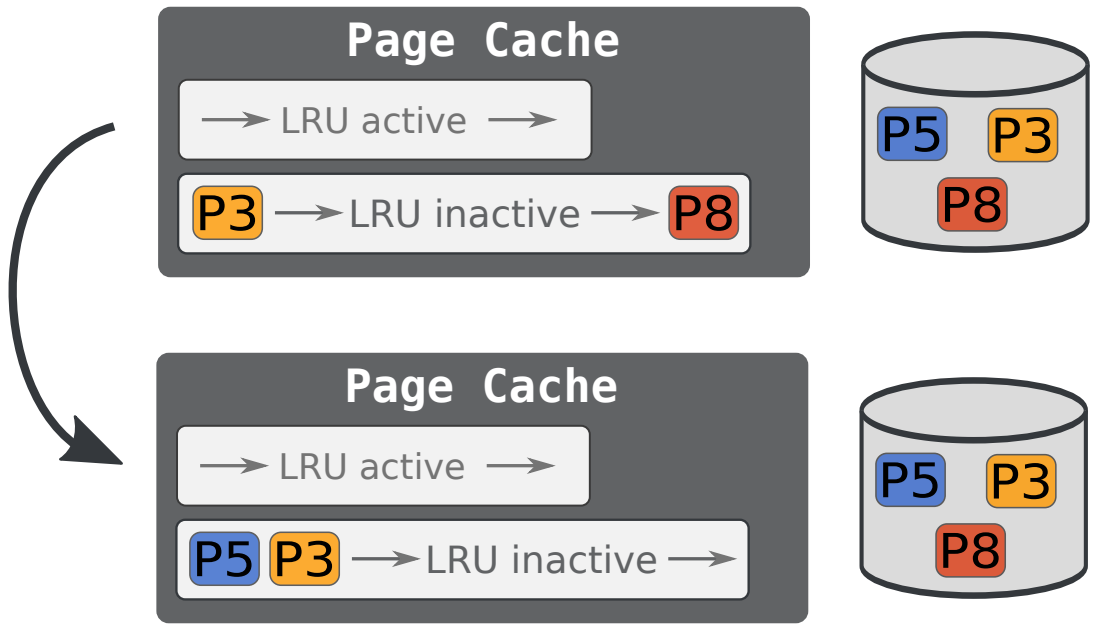
Contexte #1 : gestion du page cache dans Linux



Contexte #1 : gestion du page cache dans Linux

1ere lecture de P5 :

- Ajout de P5
- Eviction de P8



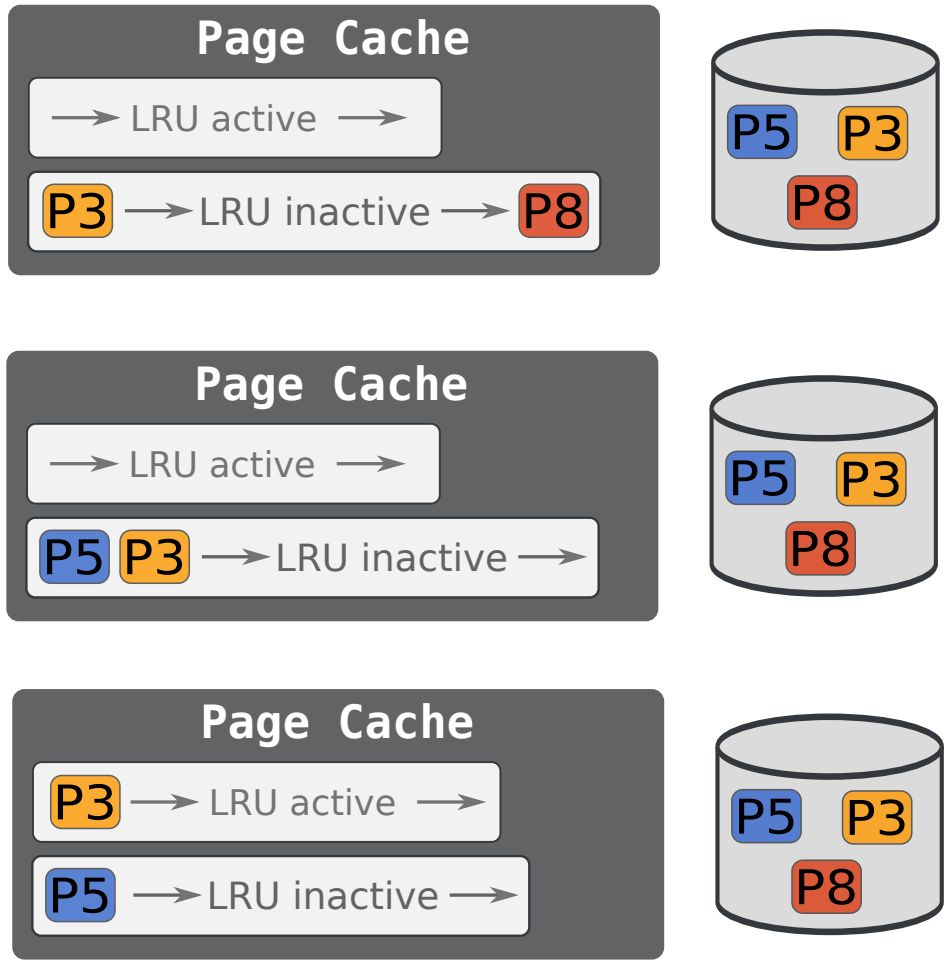
Contexte #1 : gestion du page cache dans Linux

1ere lecture de P5 :

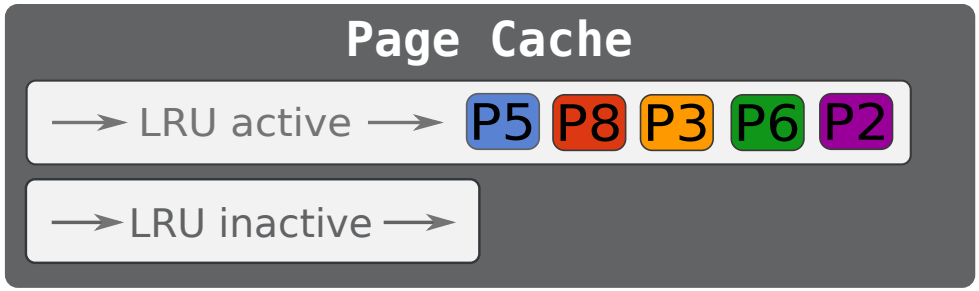
- Ajout de P5
- Eviction de P8

2eme lecture de P3 :

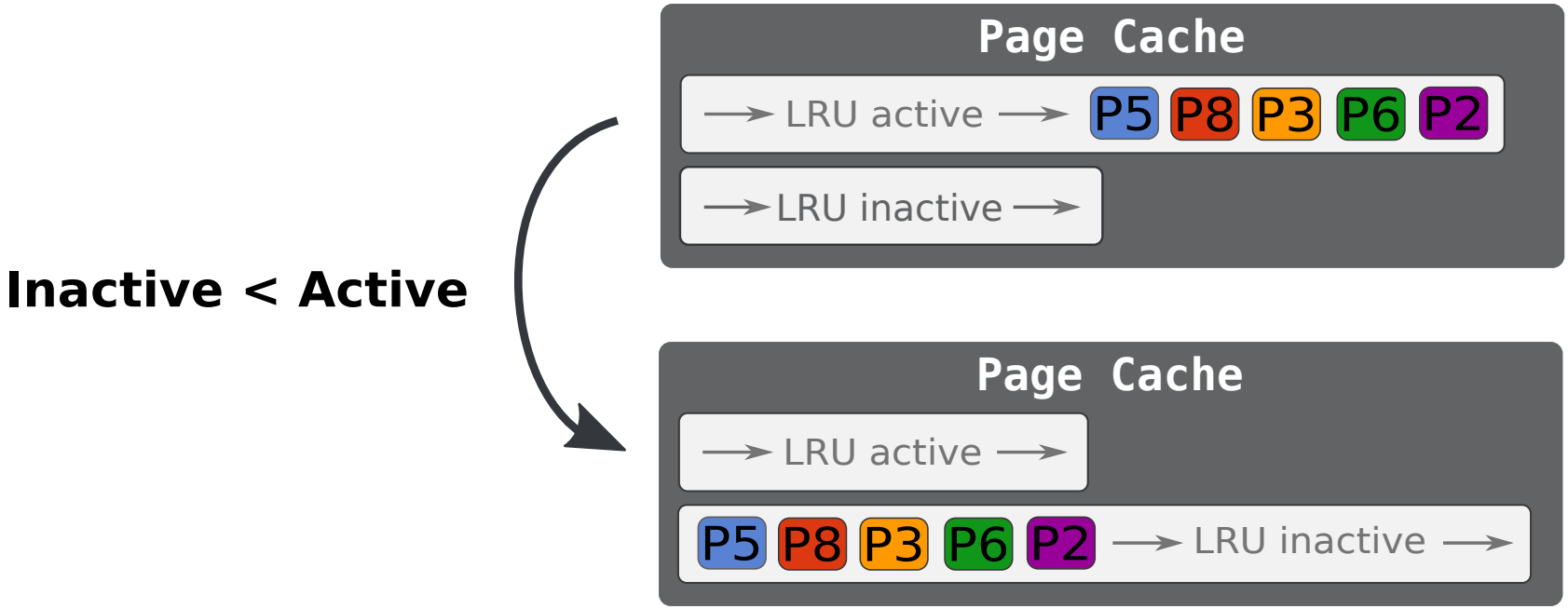
- Promotion de P3
- **Pas d'éviction**



Contexte #2 : désactivation des pages



Contexte #2 : désactivation des pages



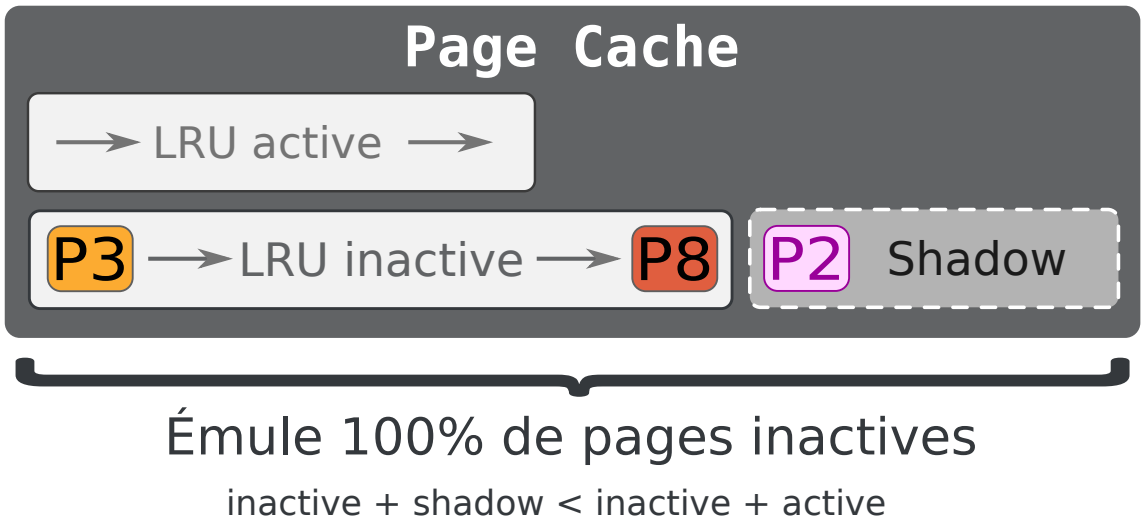
Invariant du système

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

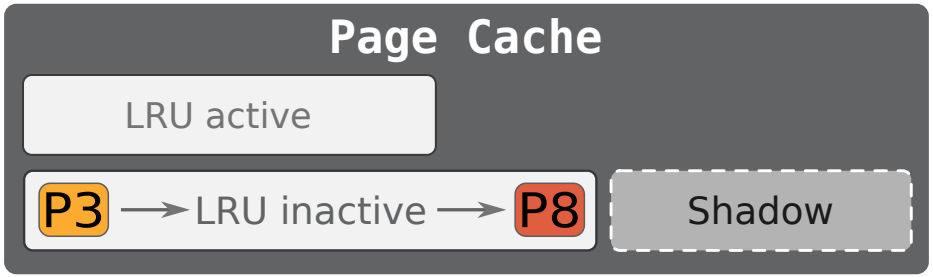
Contexte #3 : *shadow* page cache

- **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.



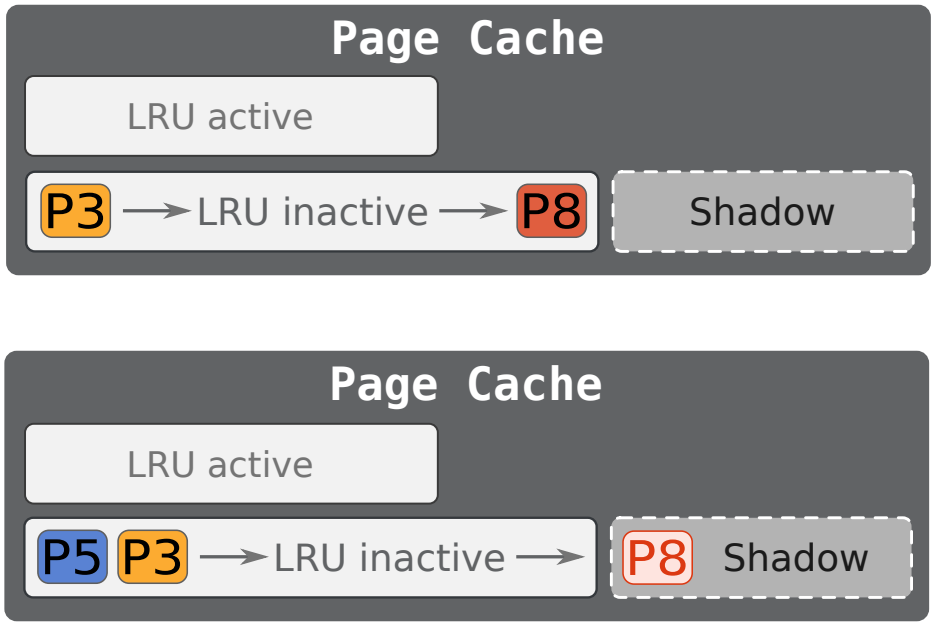
Contexte #3 : *shadow* page cache (exemple)



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

1ere lecture de P5 :

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



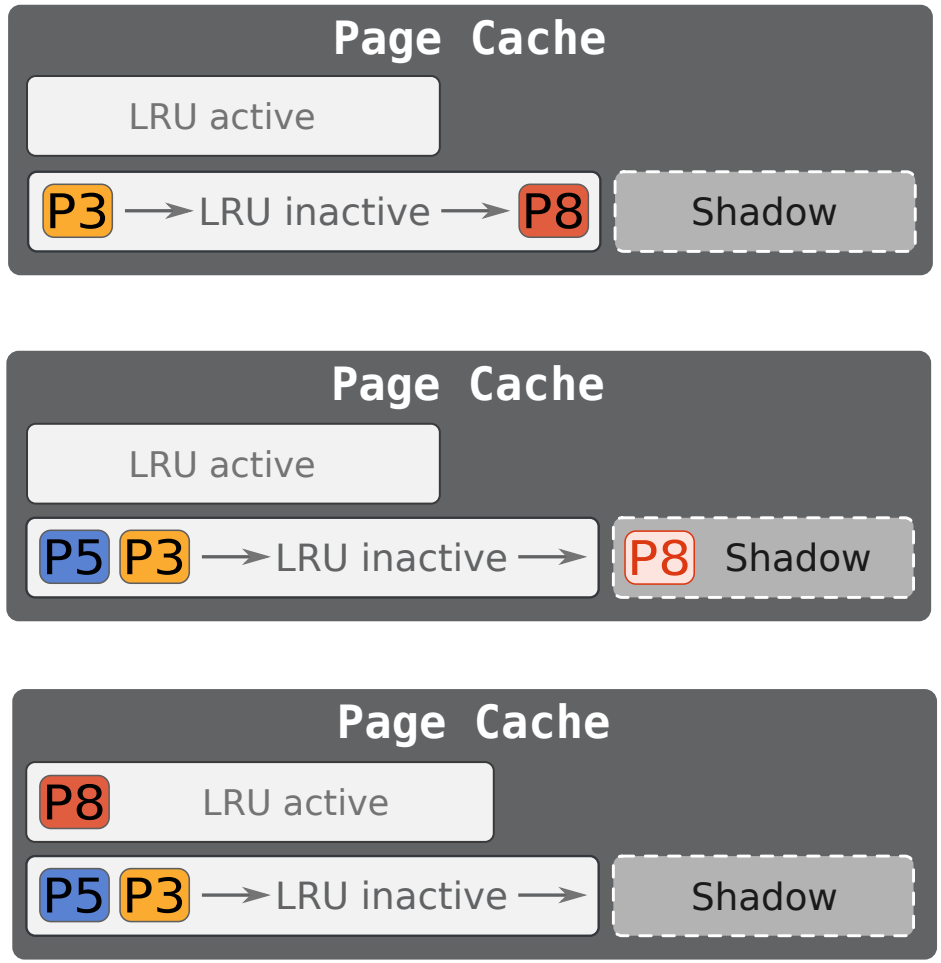
Contexte #3 : *shadow* page cache (exemple)

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



Problèmes à résoudre sur PUMA

1. Assurer le bon fonctionnement du serveur

   Scénario : Client : I/O // Serveur : malloc

2. Le serveur doit récupérer la mémoire après l'avoir prêtée au client

   Scénario : Client : I/O puis Serveur : I/O

3. Le client doit pouvoir profiter du cache quand le serveur ne l'utilise plus

   Scénario : Serveur : I/O puis Client : I/O

4. Ne pas gêner le pagecache du serveur

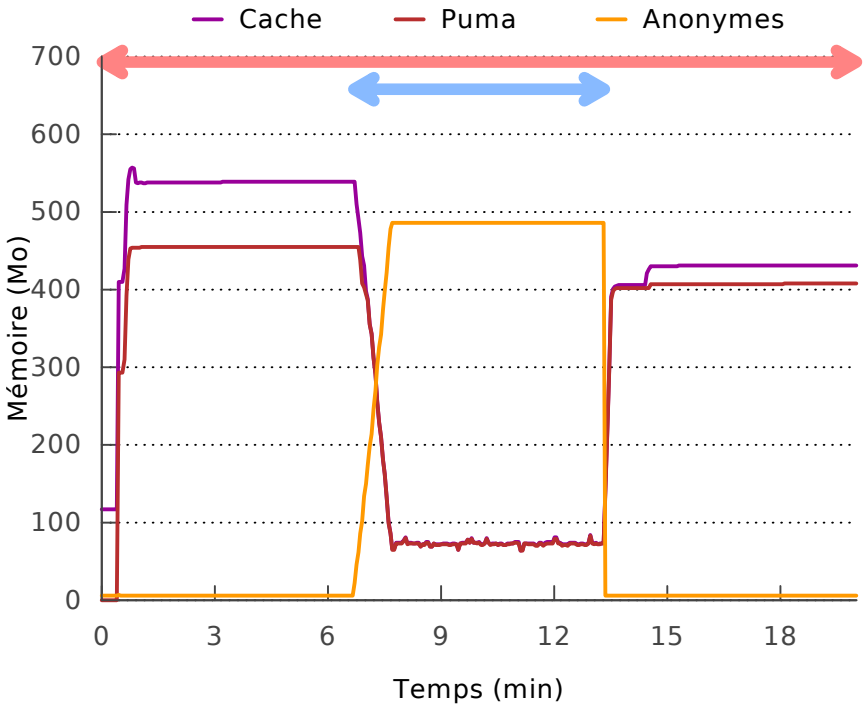
   Scénario : Client : I/O // Serveur : I/O

Rendre et prendre de la mémoire efficacement

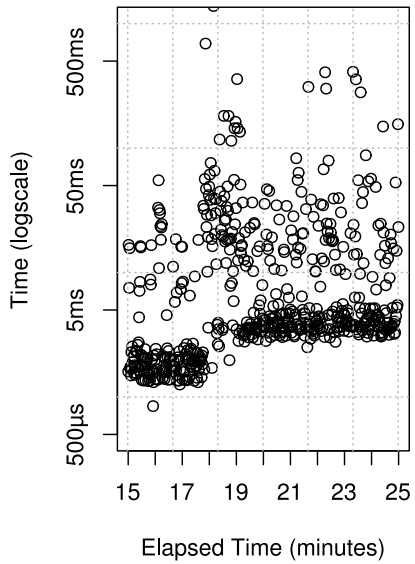
- Client : I/O // Serveur : malloc
- Problème : il ne faut pas perturber les processus du serveur
  $\Rightarrow$ intégrer PUMA dans les LRUs du pagecache

Rendre et prendre de la mémoire efficacement

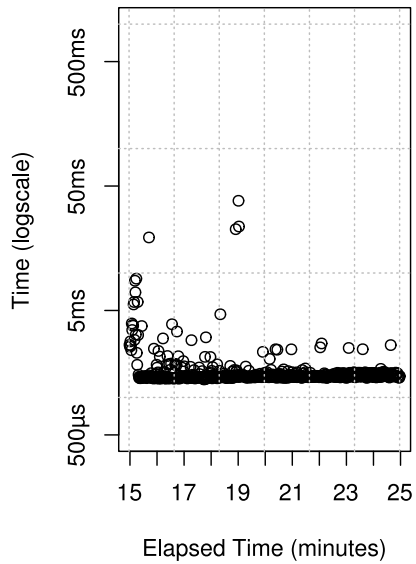
- Client : I/O // Serveur : malloc
- Problème : il ne faut pas perturber les processus du serveur
  ⇒ intégrer PUMA dans les LRUs du pagecache



PUMA *(mémoire serveur)*



malloc (ballooning)



malloc (PUMA)

Problèmes à résoudre sur PUMA

1. Assurer le bon fonctionnement du serveur

   Scénario : Client : I/O // Serveur : malloc

   Solution : intégration au pagecache

2. Le serveur doit récupérer la mémoire après l'avoir prêtée au client

   Scénario : Client : I/O puis Serveur : I/O

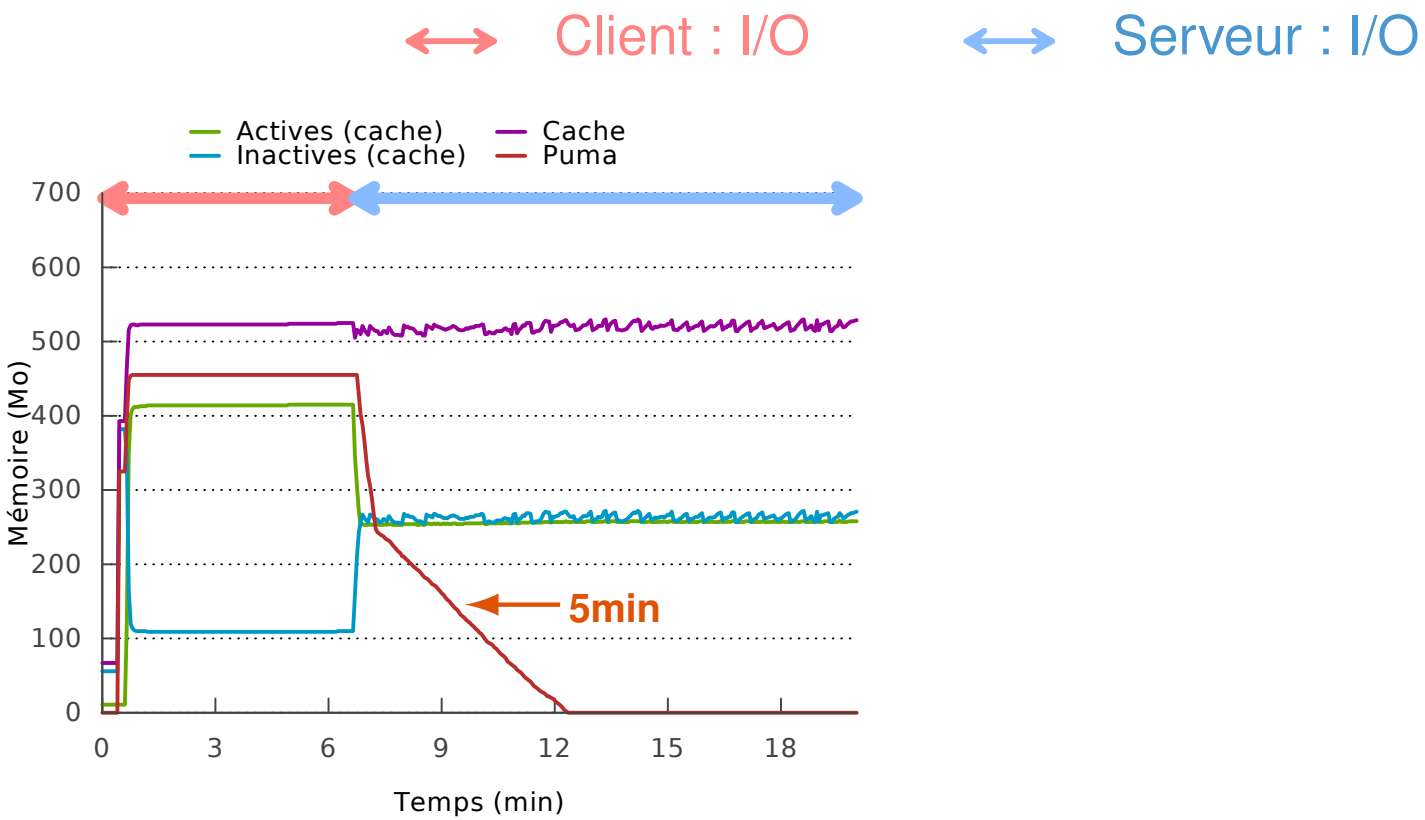
3. Le client doit pouvoir profiter du cache quand le serveur ne l'utilise plus

   Scénario : Serveur : I/O puis Client : I/O

4. Ne pas gêner le pagecache du serveur

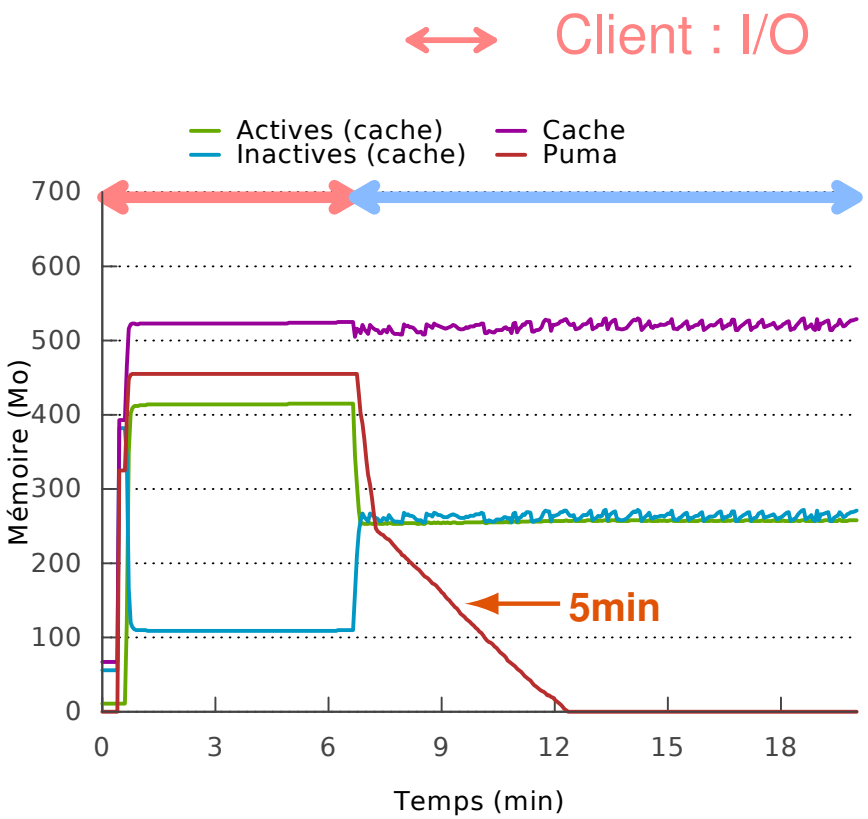
   Scénario : Client : I/O // Serveur : I/O

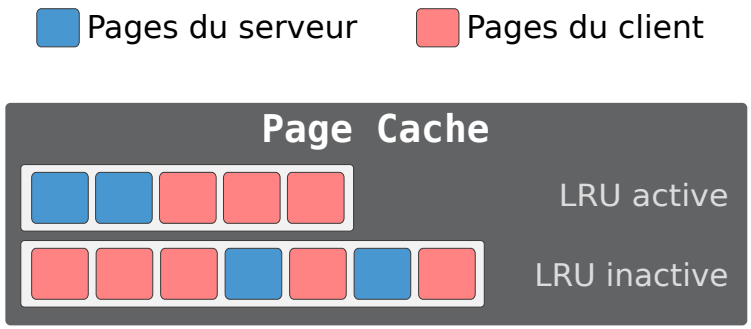
Récupération rapide du serveur



PUMA *intégré au pagecache*

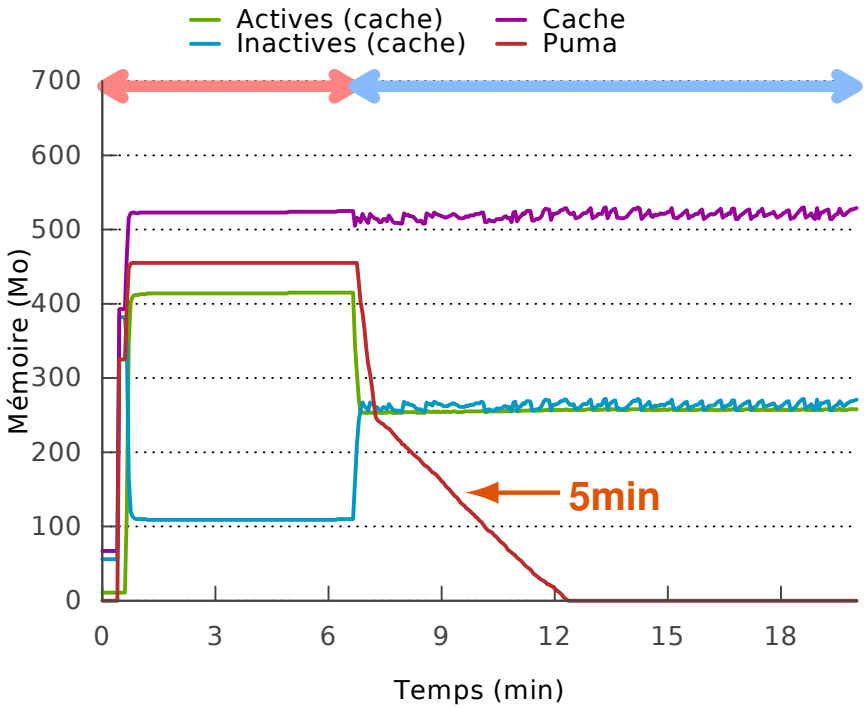
Récupération rapide du serveur



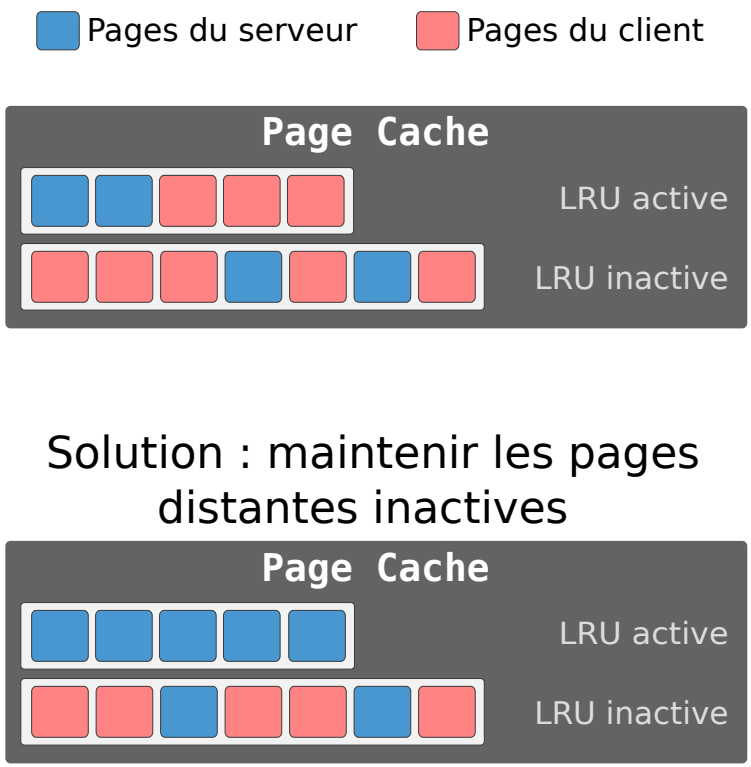


PUMA *intégré au pagecache*

Récupération rapide du serveur

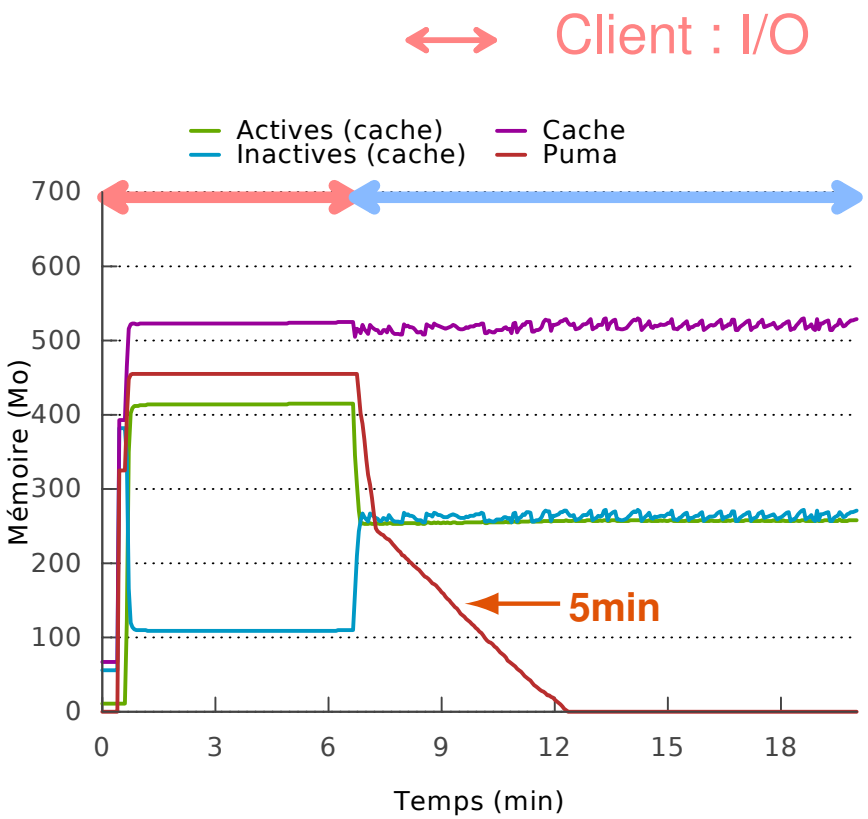


PUMA *intégré au pagecache*

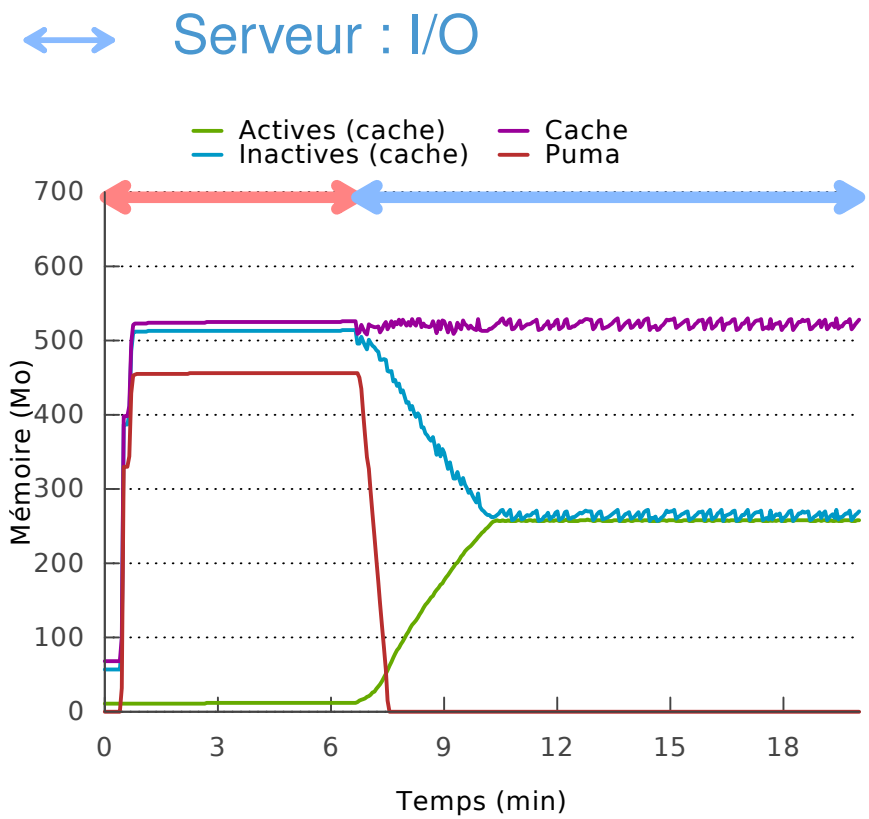


Solution : maintenir les pages distantes inactives

Récupération rapide du serveur



PUMA *intégré au pagecache*



PUMA *limité aux pages inactives*

Problèmes à résoudre sur PUMA

1. Assurer le bon fonctionnement du serveur

   Scénario : Client : I/O // Serveur : malloc

   Solution : intégration au pagecache

2. Le serveur doit récupérer la mémoire après l'avoir prêtée au client

   Scénario : Client : I/O puis Serveur : I/O

   Solution : pages distantes en inactives

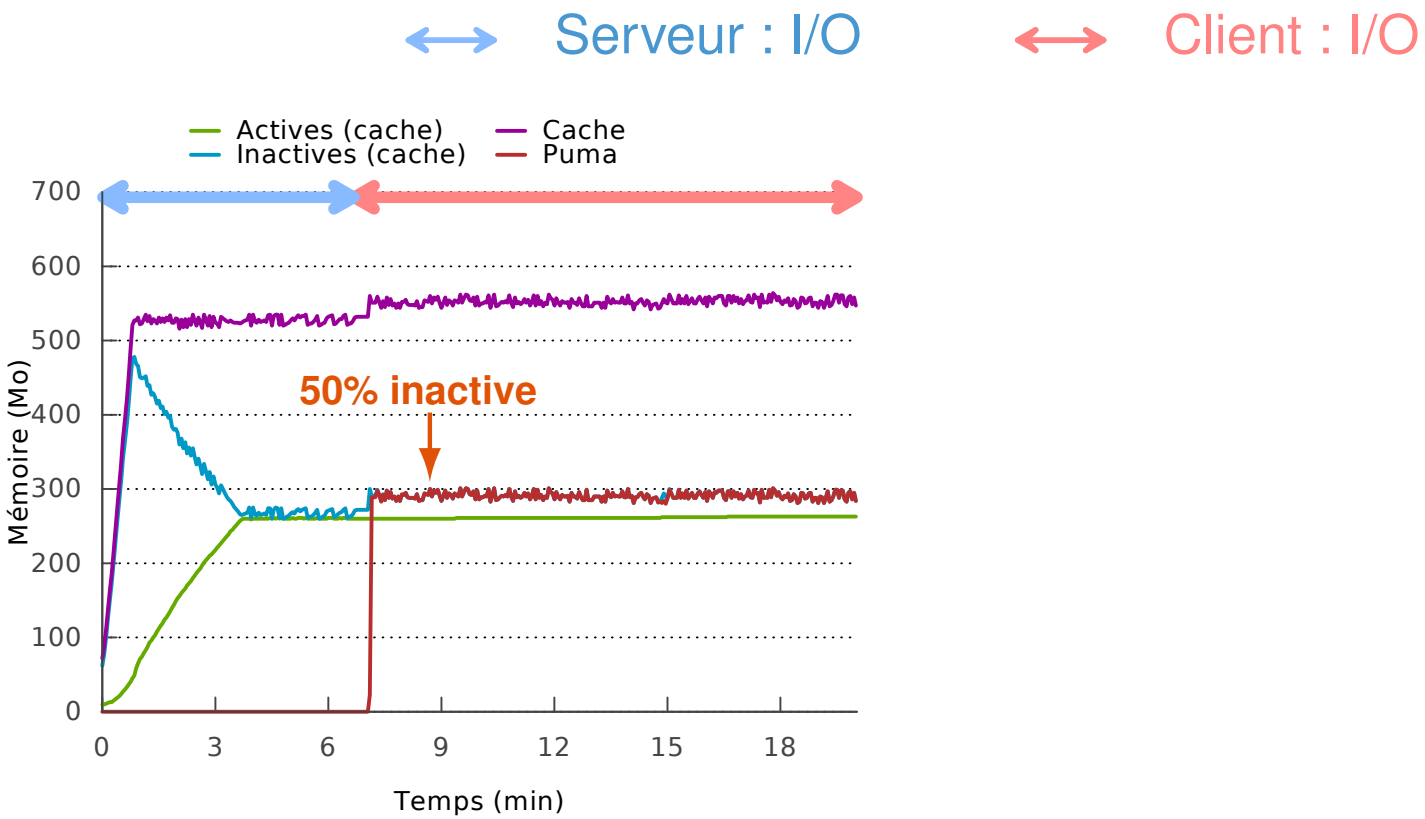
3. Le client doit pouvoir profiter du cache quand le serveur ne l'utilise plus

   Scénario : Serveur : I/O puis Client : I/O

4. Ne pas gêner le pagecache du serveur

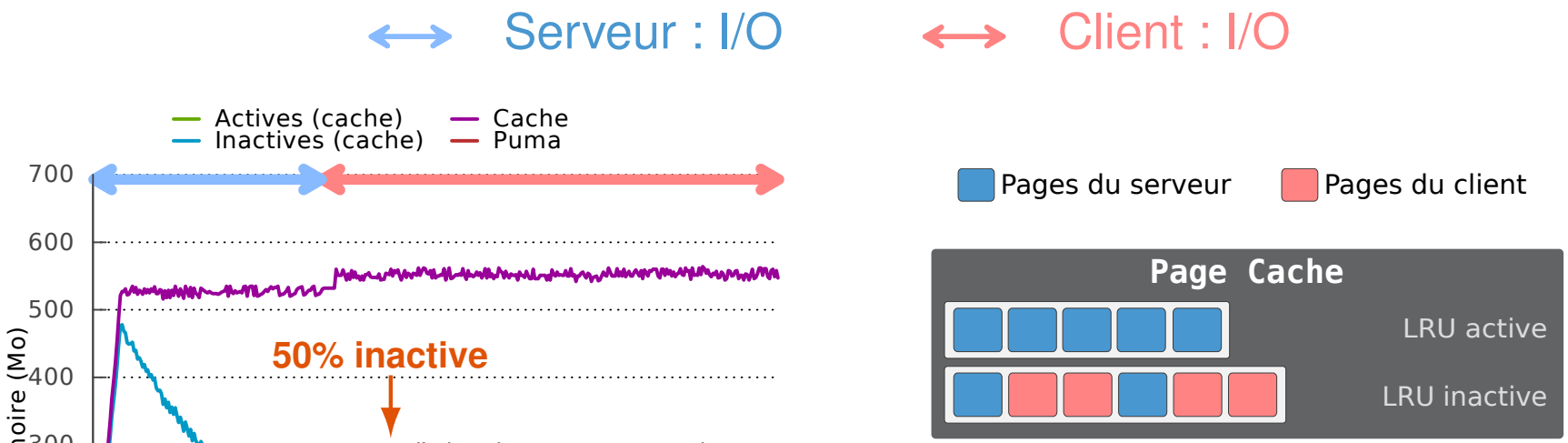
   Scénario : Client : I/O // Serveur : I/O

Reprise du client



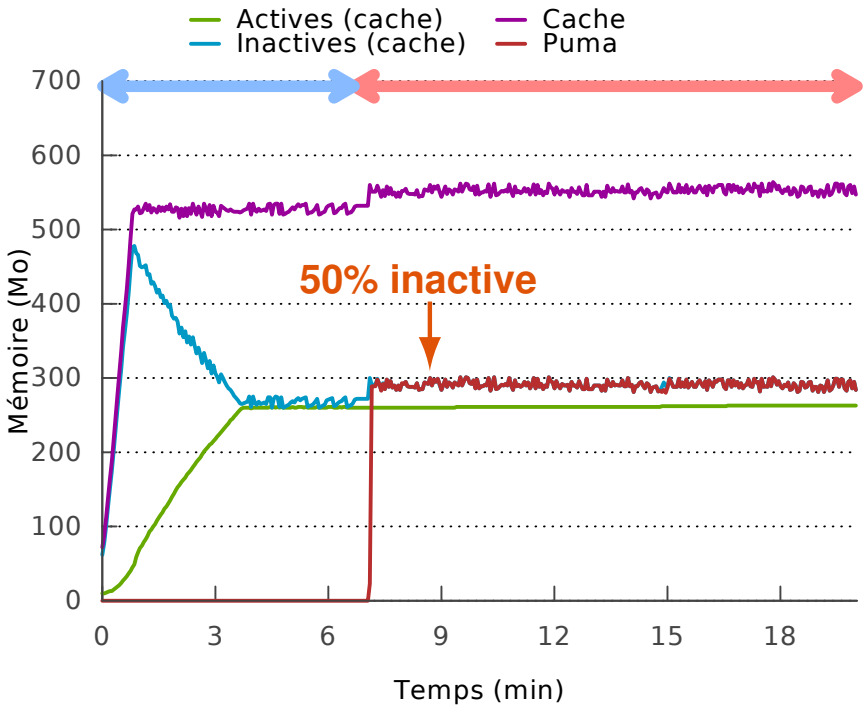
PUMA *limité aux pages inactives*

Reprise du client

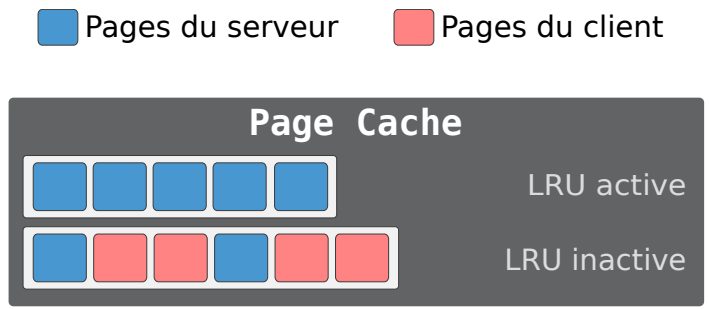


PUMA *limité aux pages inactives*

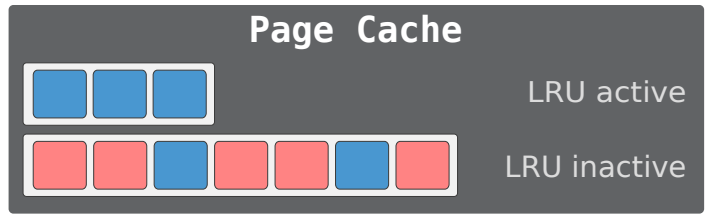
Reprise du client



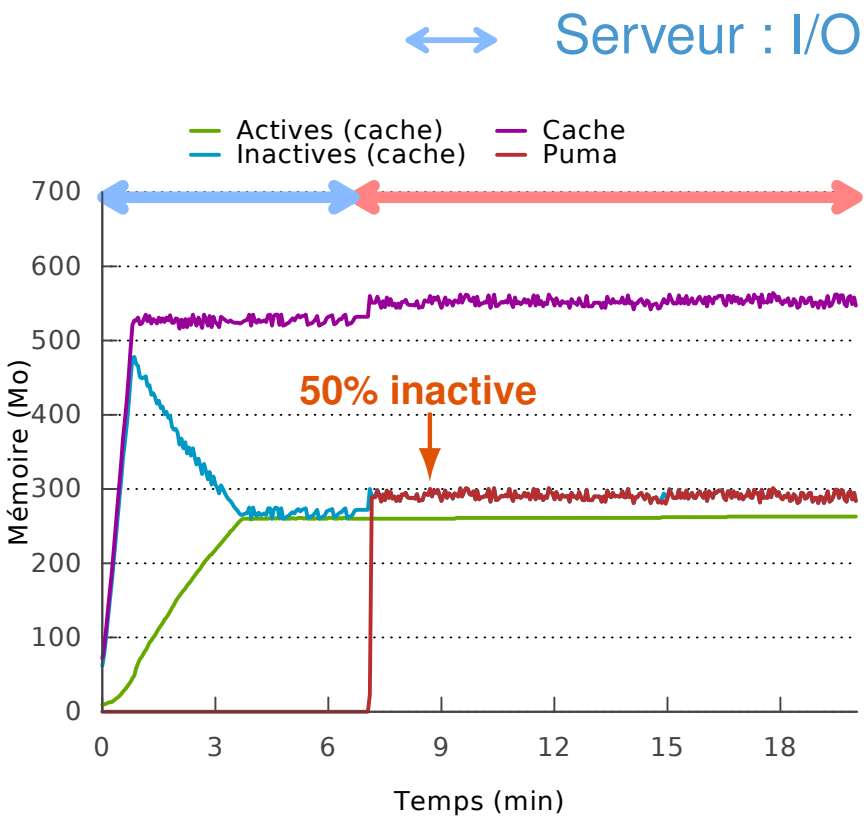
PUMA *limité aux pages inactives*



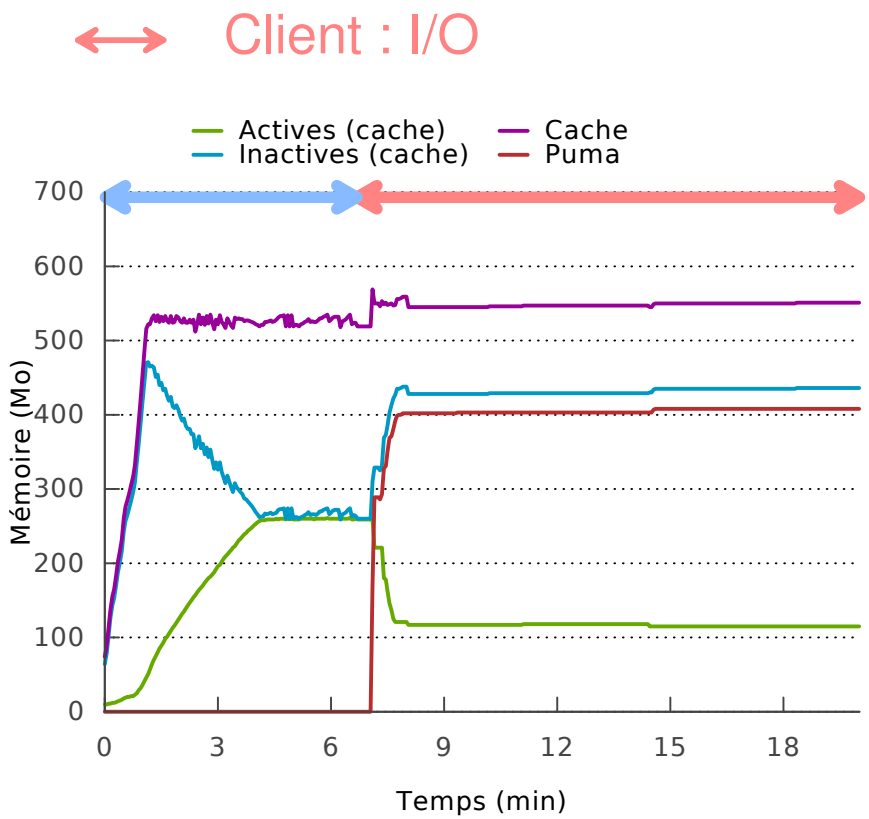
Solution : forcer la désactivation



Reprise du client



PUMA *limité aux pages inactives*



Désactivation des pages actives forcée

Problèmes à résoudre sur PUMA

1. Assurer le bon fonctionnement du serveur

   Scénario : Client : I/O // Serveur : malloc

   Solution : intégration au pagecache

2. Le serveur doit récupérer la mémoire après l'avoir prêtée au client

   Scénario : Client : I/O puis Serveur : I/O

   Solution : pages distantes en inactives

3. Le client doit pouvoir profiter du cache quand le serveur ne l'utilise plus

   Scénario : Serveur : I/O puis Client : I/O

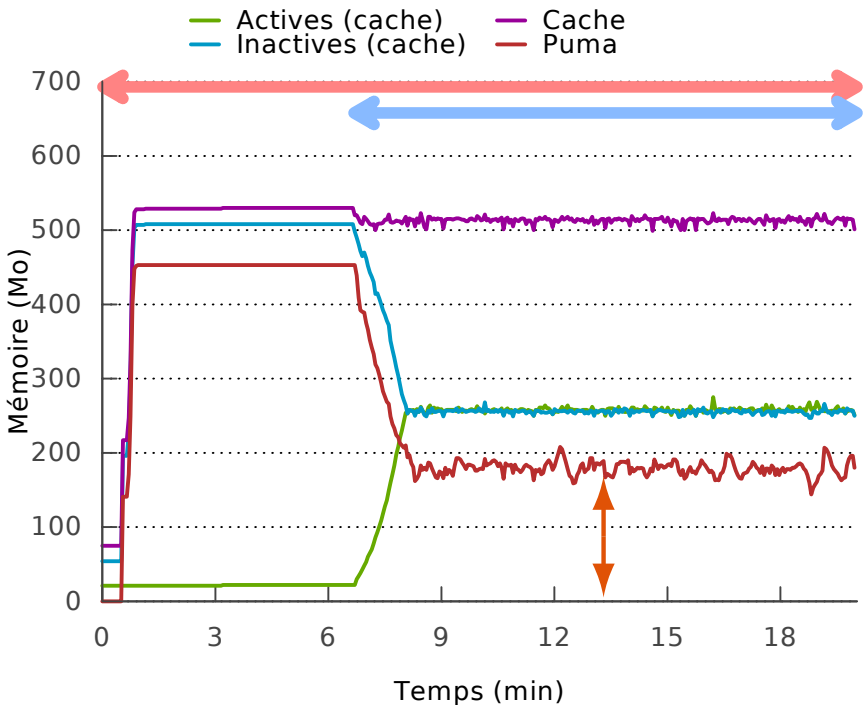
   Solution : forcer la désactivation des pages

4. Ne pas gêner le pagecache du serveur

   Scénario : Client : I/O // Serveur : I/O

Accès concurrents sur le cache ?

Client : I/O // Serveur : I/O



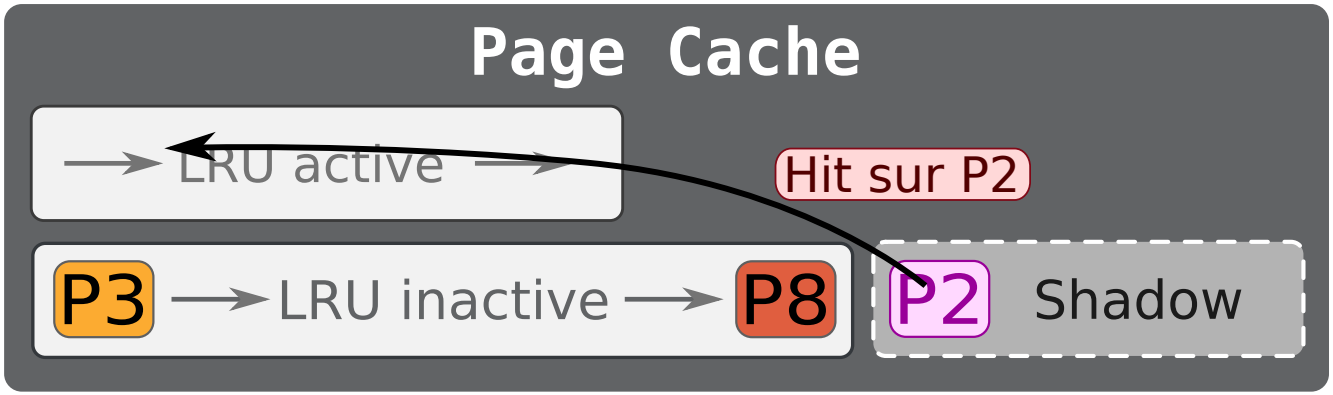
PUMA *amélioré*

I/O par secondes

Le serveur doit avoir la priorité ⇒ Comment détecter l'(in)activité du serveur ?

Détecteur #1 : *shadow* page cache

Idée : détecter l'activité quand une page inactive est évincée trop rapidement

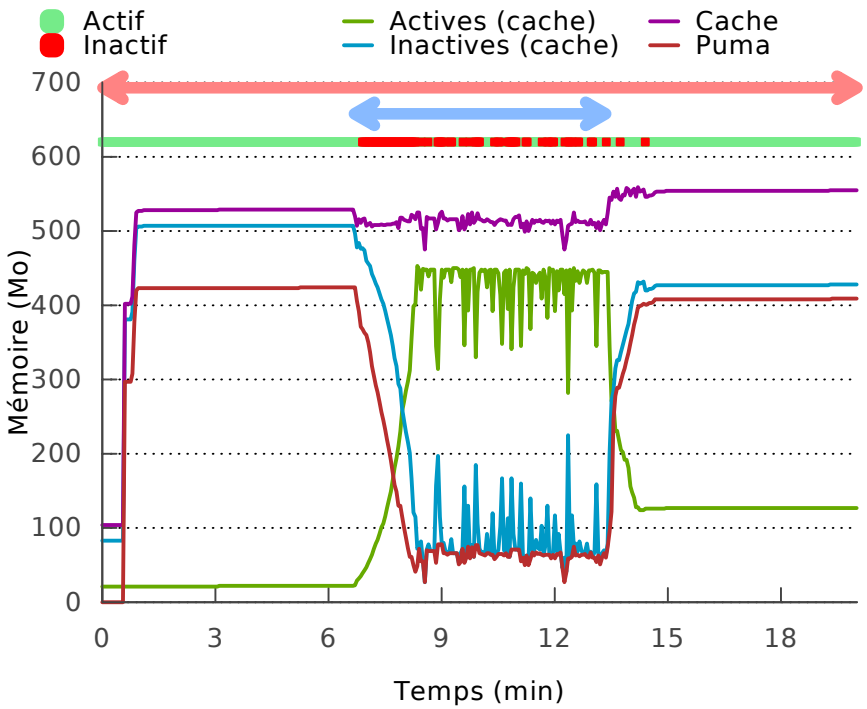


Heuristique

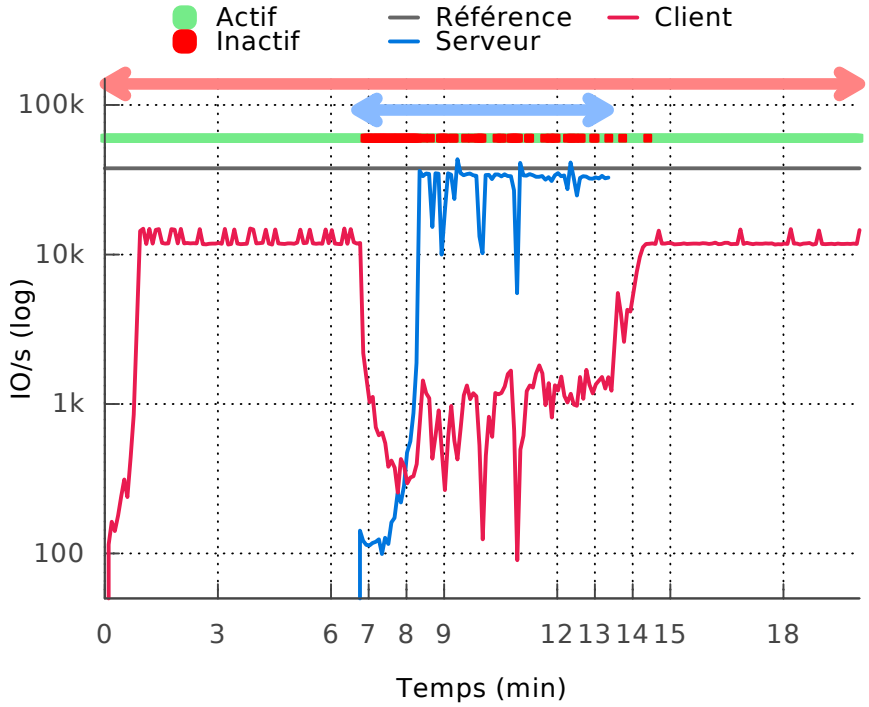
- Hit shadow → arrêt de PUMA
- Si pas de hit au bout de 3s → réactiver le service

Détecteur #1 : *shadow* page cache

Client : I/O // Serveur : I/O



PUMA *avec détecteur #1*

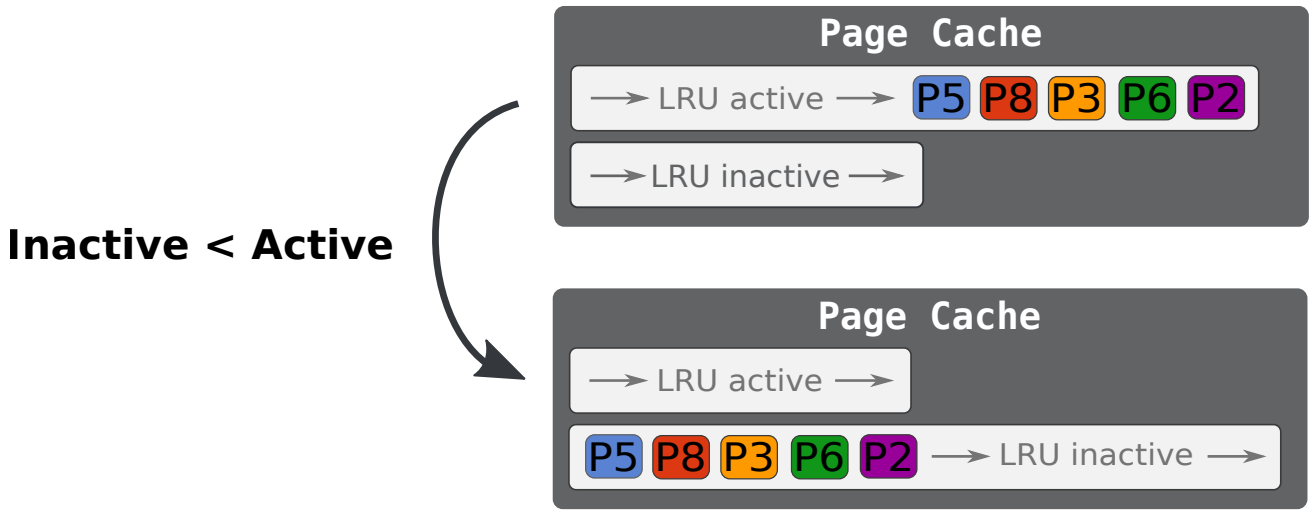


I/O par secondes

Problème : précision → PUMA se réactive trop facilement

Détecteur #2 : pression mémoire

Idée : détecter l'activité quand des pages sont désactivées

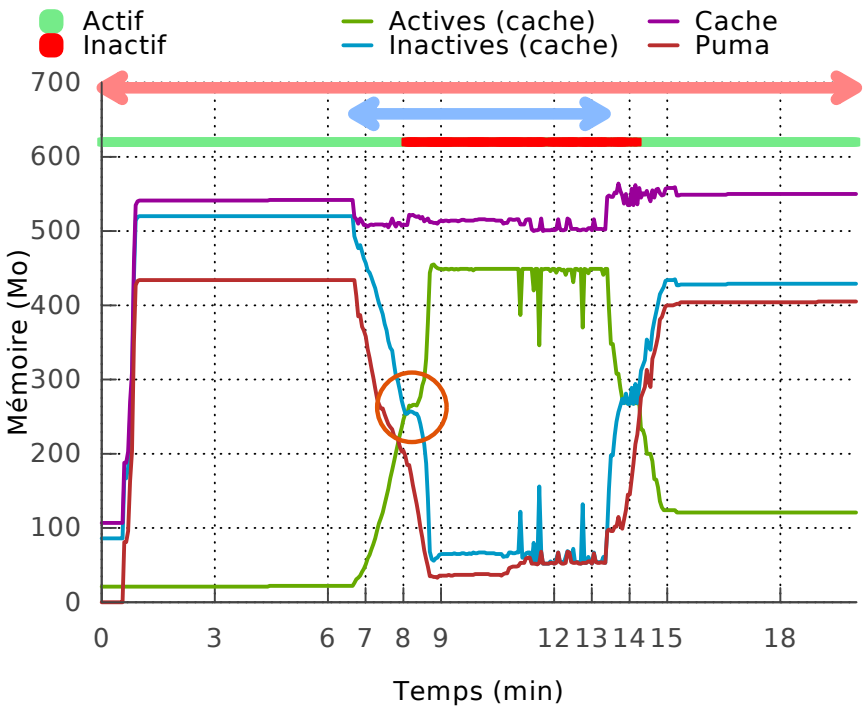


Heuristique

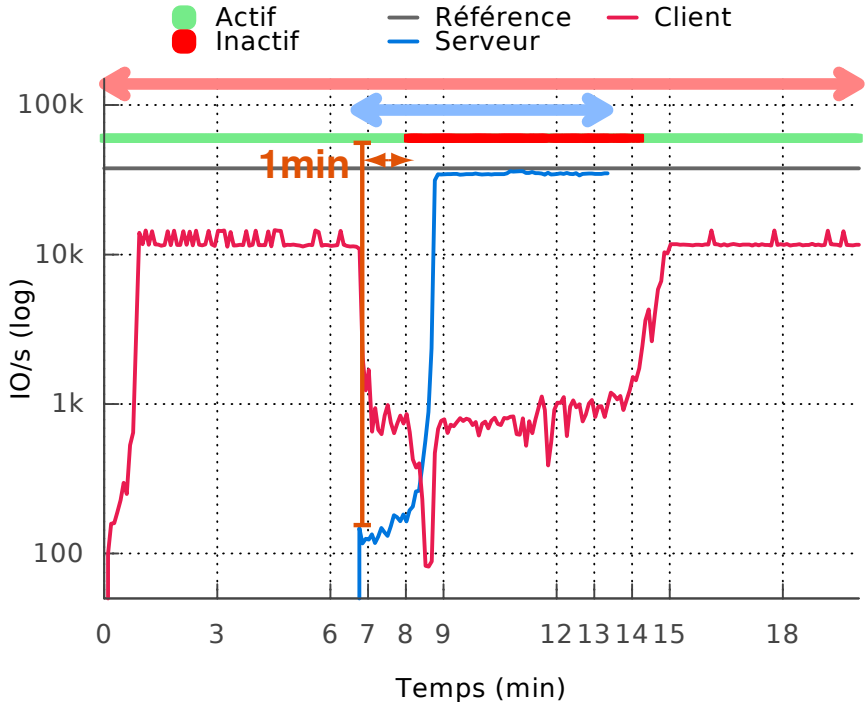
- Si désactivation → arrêt de PUMA
- Pas de désactivation au bout de 3s → réactiver le service

Détecteur #2 : pression mémoire

Client : I/O // Serveur : I/O



PUMA *avec détecteur #2*

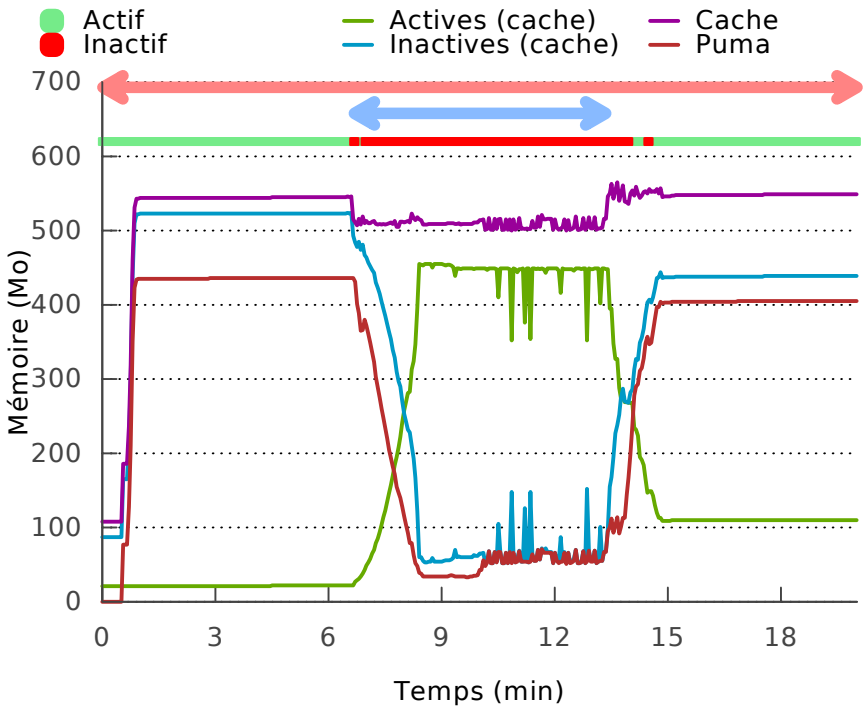


I/O par secondes

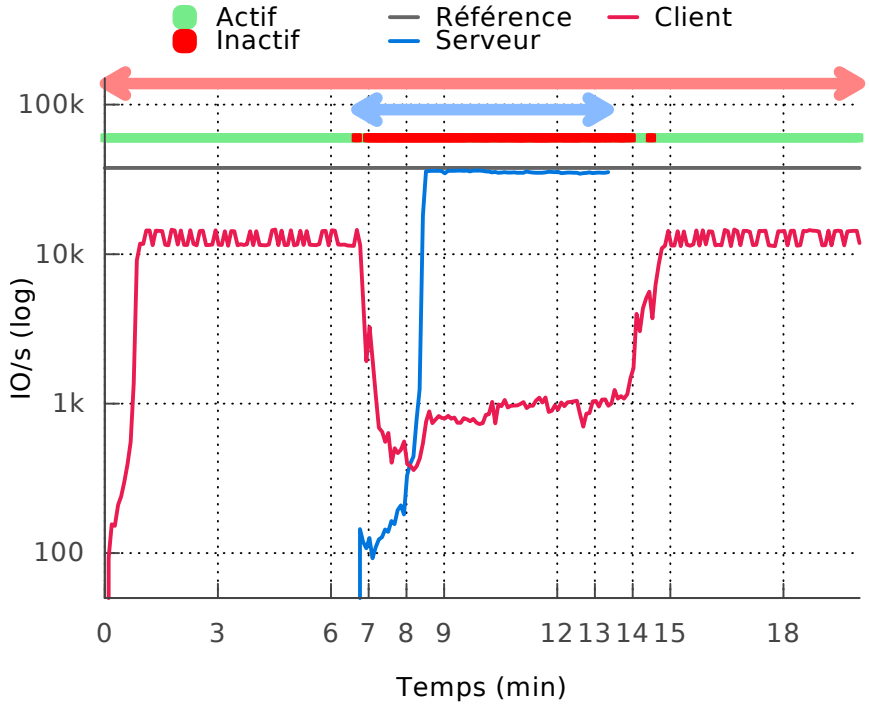
Problème : délais → il faut attendre que 50% des pages soient actives

Détecteur #3 : *shadow* page cache (#1) + pression mémoire (#2)

- Combiner les 2 techniques
  - récupération plus rapide (#1)
  - plus de précision (#2)



PUMA *avec détecteur #3*



I/O par secondes

Problèmes à résoudre sur PUMA

1. Assurer le bon fonctionnement du serveur
   - Scénario : Client : I/O // Serveur : malloc
   - Solution : intégration au pagecache
2. Le serveur doit récupérer la mémoire après l'avoir prêtée au client
   - Scénario : Client : I/O puis Serveur : I/O
   - Solution : pages distantes en inactives
3. Le client doit pouvoir profiter du cache quand le serveur ne l'utilise plus
   - Scénario : Serveur : I/O puis Client : I/O
   - Solution : forcer la désactivation des pages
4. Ne pas gêner le pagecache du serveur
   - Scénario : Client : I/O // Serveur : I/O
   - Solution : Détection d'activité basée sur le *shadow* page cache et la pression mémoire

Conclusion

Résumé

- PUMA apporte une solution au problème de fragmentation du page cache
  - $\Rightarrow$ mais sa configuration était *statique*
- Les approches à base d'auto-ballooning ne sont pas efficace
  - $\Rightarrow$ récupérer de la mémoire prêtée est 20$\times$ plus lent qu'en temps normal

Contribution

- Automatisation du dimensionnement de PUMA
  - $\Rightarrow$ mécanismes de récupération efficace de la mémoire prêtée
    - $\rightarrow$ divise par 10 le surcout comparé à l'auto-ballooning
  - $\Rightarrow$ détection de l'(in)activité mémoire d'une VM
    - $\rightarrow$ désactivation de PUMA en cas d'activité
    - $\rightarrow$ réactivation de PUMA lors d'une baisse d'activité