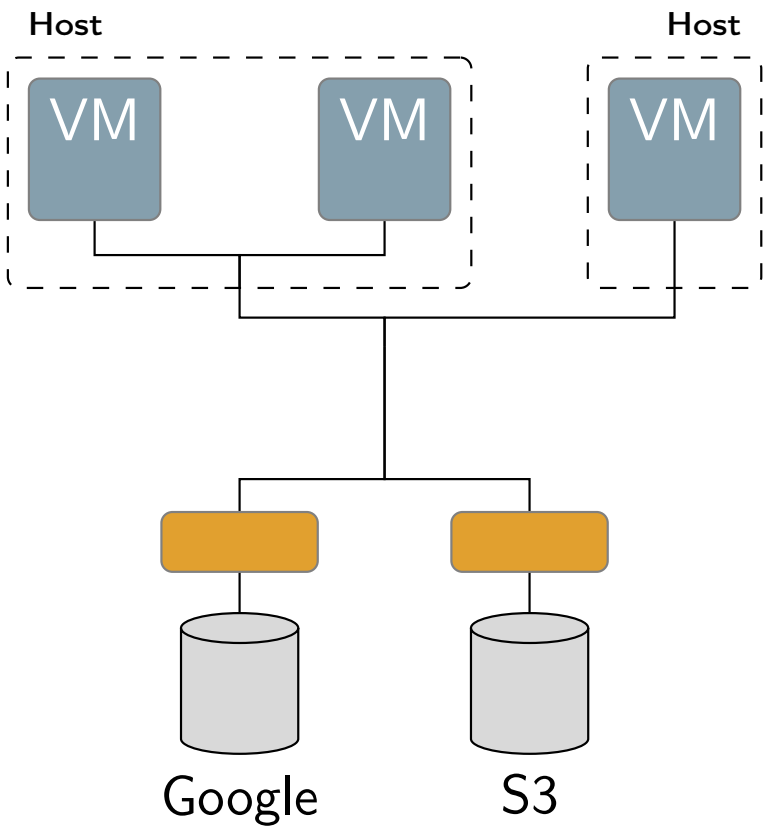
Vers un cache réparti adapté au cloud computing

Maxime Lorrillere, Julien Sopena, Sébastien Monnet et Pierre Sens

Cloud computing



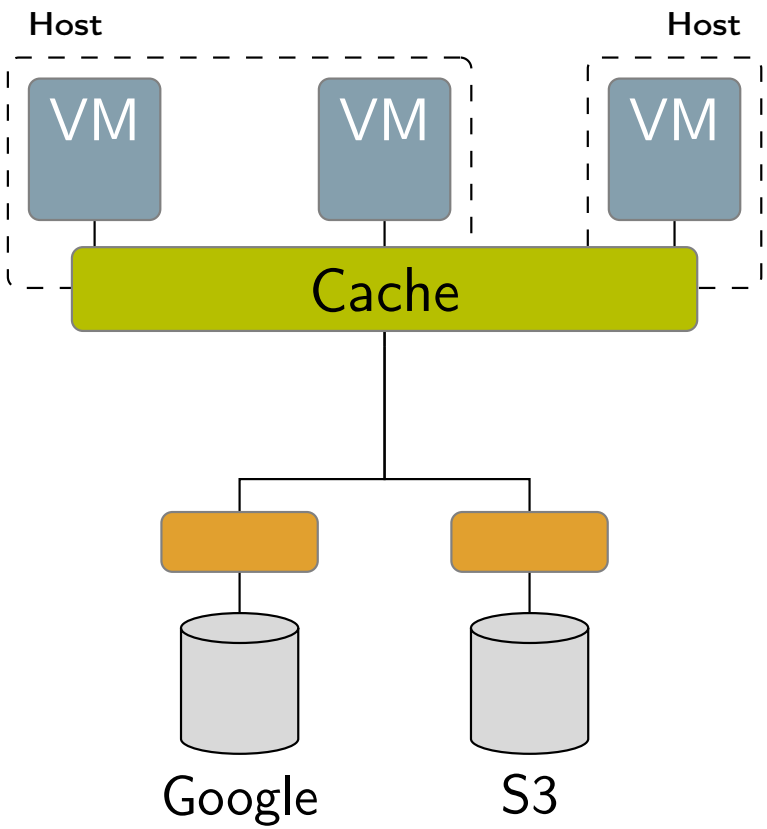
Cloud computing

- Différents niveaux de services
- Flexibilité
- Hétérogénéité

Former une plateforme virtualisée

- Composition de ressources
- Différents fournisseurs
- Différentes propriétés
- Nécessité d'une interface commune

Cloud computing



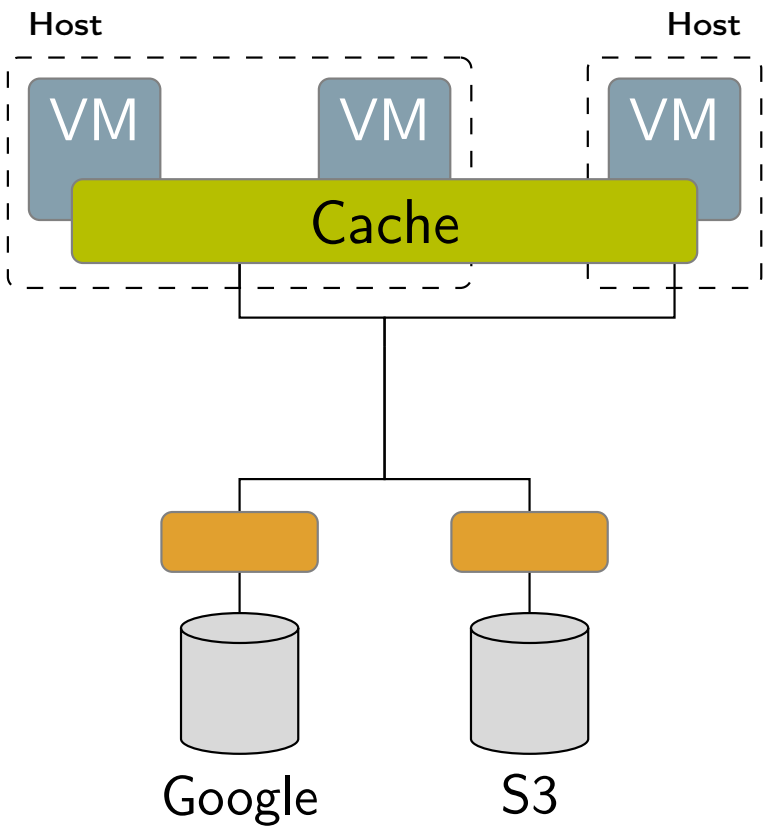
Cloud computing

- Différents niveaux de services
- Flexibilité
- Hétérogénéité

Former une plateforme virtualisée

- Composition de ressources
- Différents fournisseurs
- Différentes propriétés
- Nécessité d'une interface commune

Cloud computing



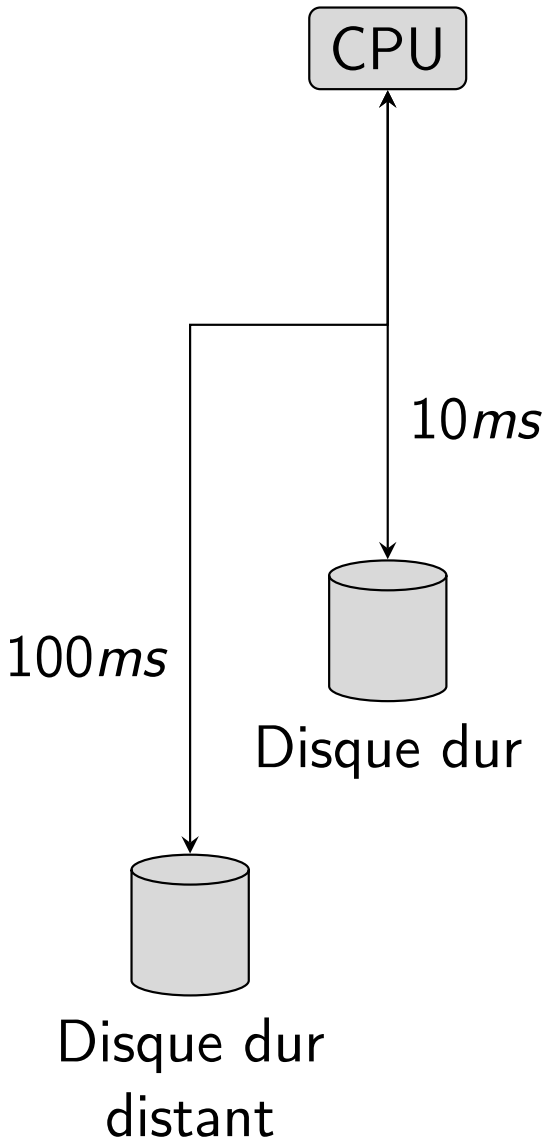
Cloud computing

- Différents niveaux de services
- Flexibilité
- Hétérogénéité

Former une plateforme virtualisée

- Composition de ressources
- Différents fournisseurs
- Différentes propriétés
- Nécessité d'une interface commune

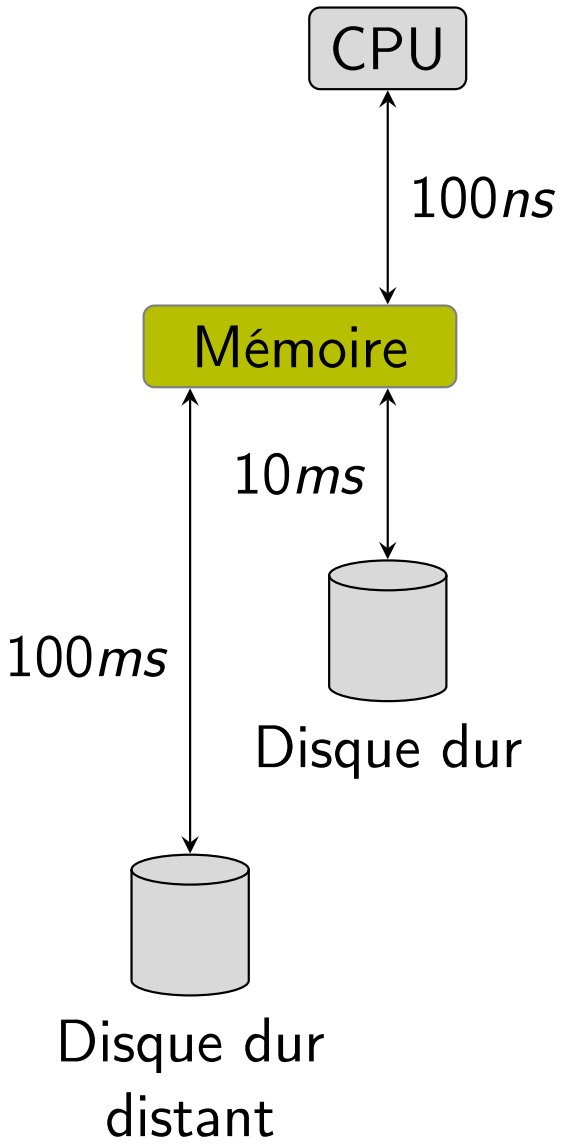
Principe des caches



Disque dur

- Latence disque élevée
- Goulot d'étranglement

Principe des caches



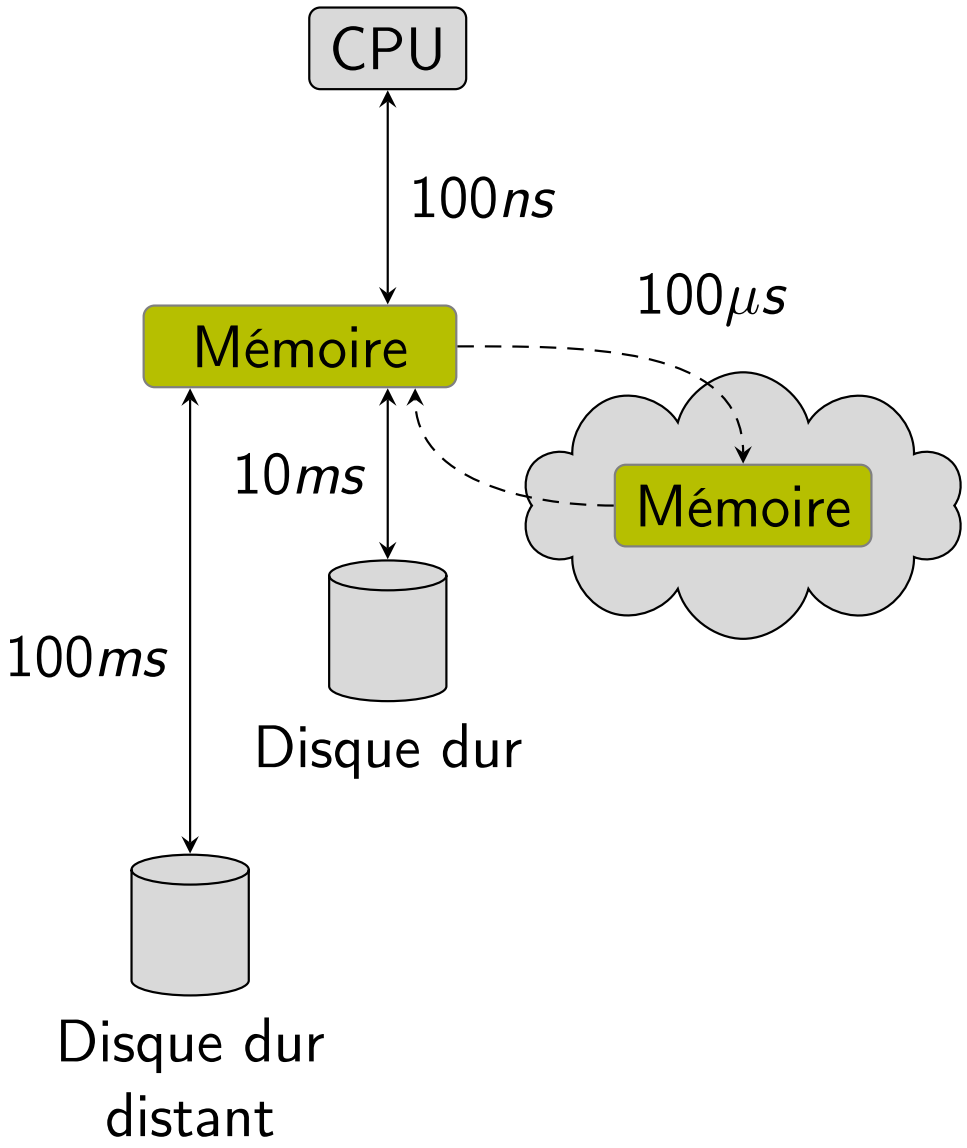
Disque dur

- Latence disque élevée
- Goulot d'étranglement

Accès à la mémoire

- Latence très faible
- Faible capacité de stockage
- Principe de localité

Principe des caches répartis



Disque dur

- Latence disque élevée
- Goulot d'étranglement

Réseau

- Latence faible
- Capacité de stockage ?

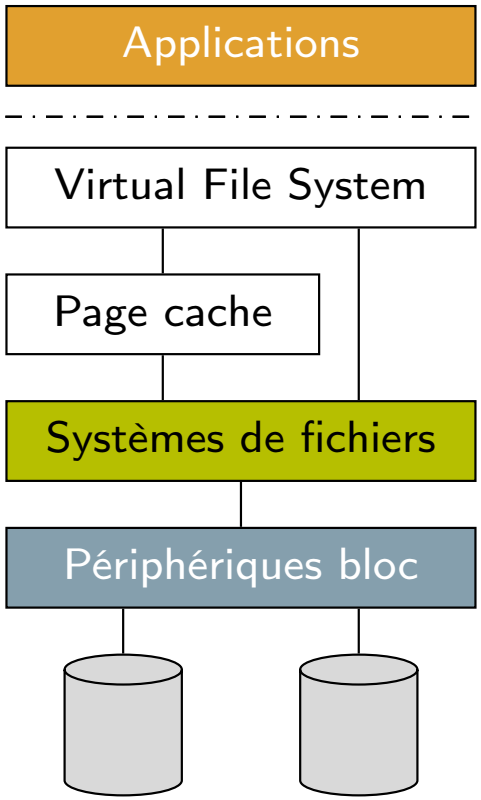
Accès à la mémoire

- Latence très faible
- Faible capacité de stockage
- Principe de localité

Limites des solutions existantes

Niveau de l'implémentation :

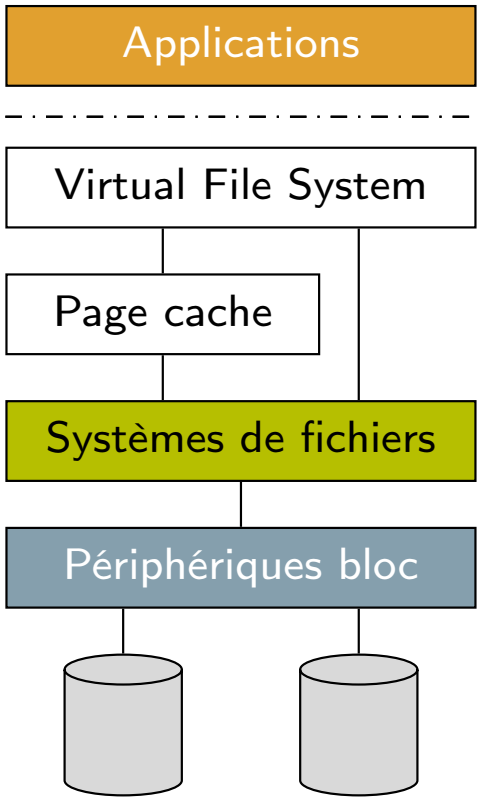
- Niveau applicatif [Memcached]
  - Nécessite d'adapter les applications existantes
- Associés à un système de fichiers [xFS, PAFS, Ceph]
  - Contraint les invités à utiliser un système de fichiers spécifique
- Niveau bloc [XHive, dm-cache]
  - Inadapté aux systèmes de fichiers répartis
- Incompatibles avec l'esprit du cloud



Limites des solutions existantes

Niveau de l'implémentation :

- Niveau applicatif [Memcached]
  - Nécessite d'adapter les applications existantes
- Associés à un système de fichiers [xFS, PAFS, Ceph]
  - Contraint les invités à utiliser un système de fichiers spécifique
- Niveau bloc [XHive, dm-cache]
  - Inadapté aux systèmes de fichiers répartis
- Incompatibles avec l'esprit du cloud



Notre contribution : une approche générique pour développer des caches répartis adaptés aux contraintes du cloud computing

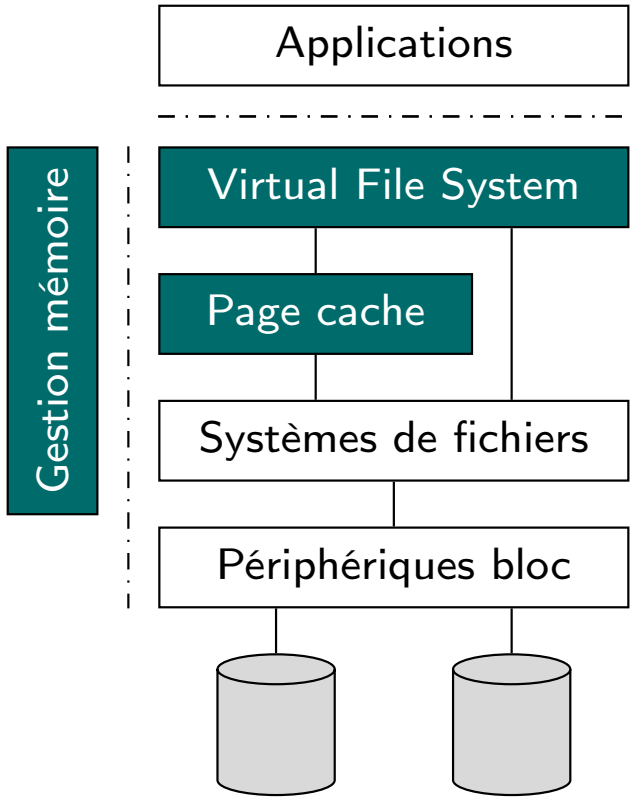
Réalisation d'un cache réparti

Contraintes de développement

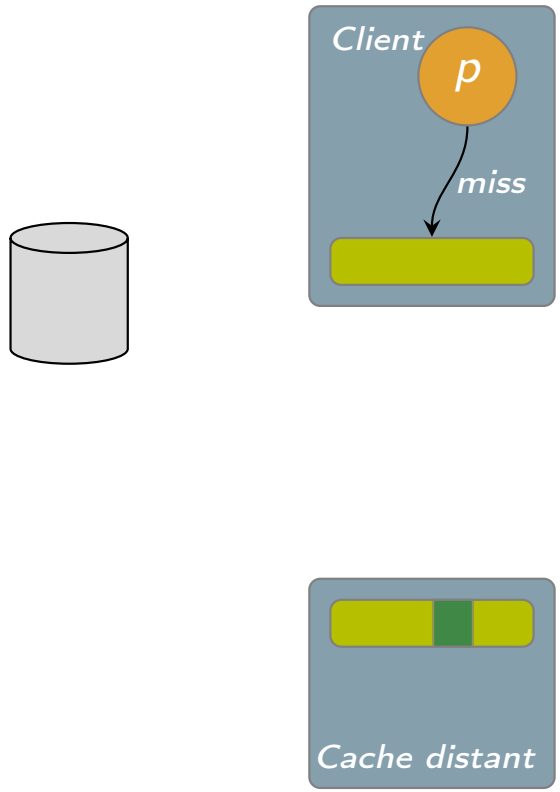
- Assurer la généricité
  - S'intégrer au sein du noyau Linux
- Limiter l'intrusion dans le noyau
- Passer des tests de régression

Contraintes de performances

- Limiter le surcoût de notre code
- Minimiser l'empreinte mémoire

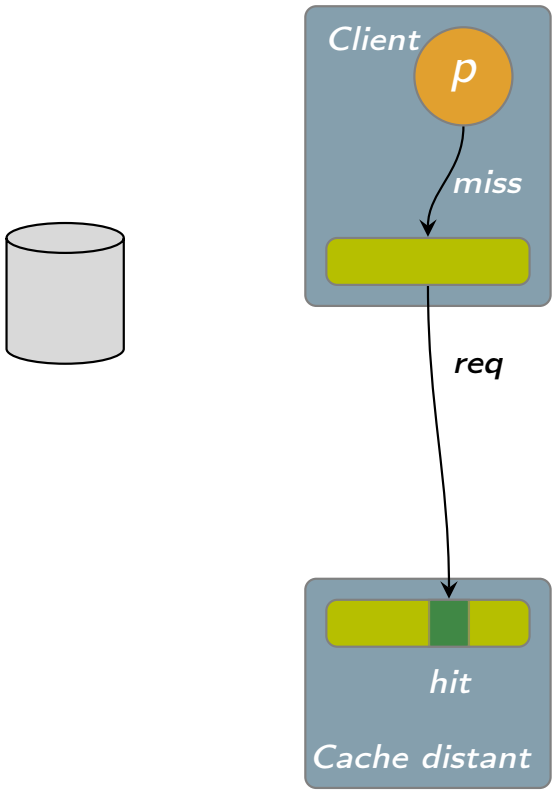


Caches distants



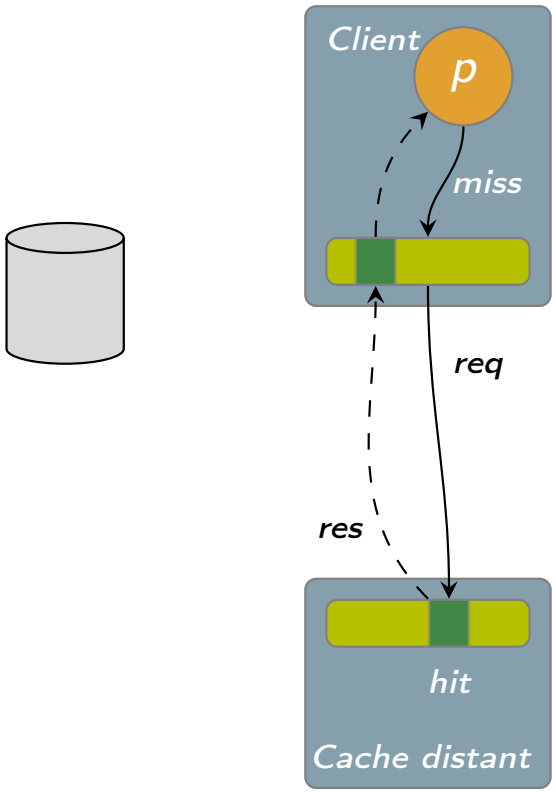
- Pas de cache global
- Localisation simple
- Pas de partage de données

Caches distants



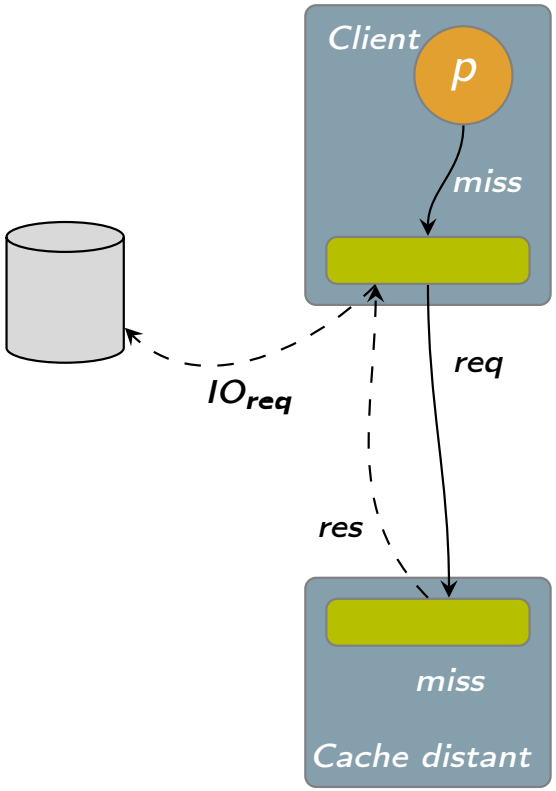
- Pas de cache global
- Localisation simple
- Pas de partage de données

Caches distants



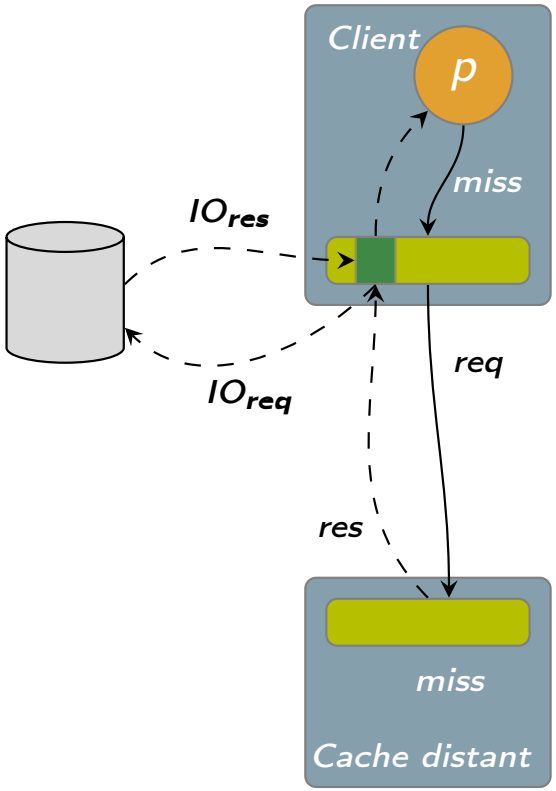
- Pas de cache global
- Localisation simple
- Pas de partage de données

Caches distants



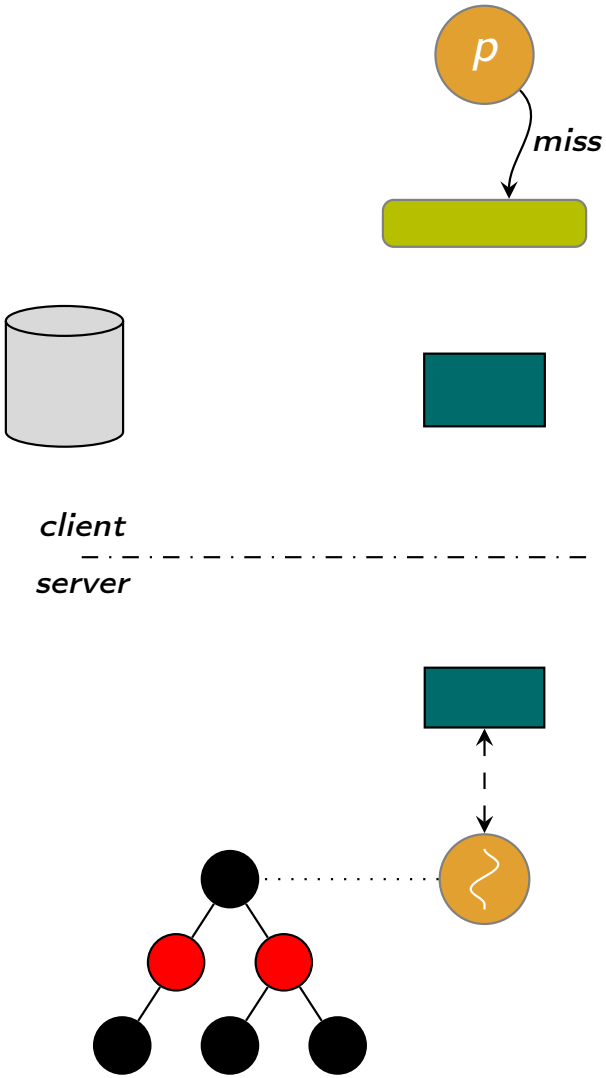
- Pas de cache global
- Localisation simple
- Pas de partage de données

Caches distants



- Pas de cache global
- Localisation simple
- Pas de partage de données

Architecture



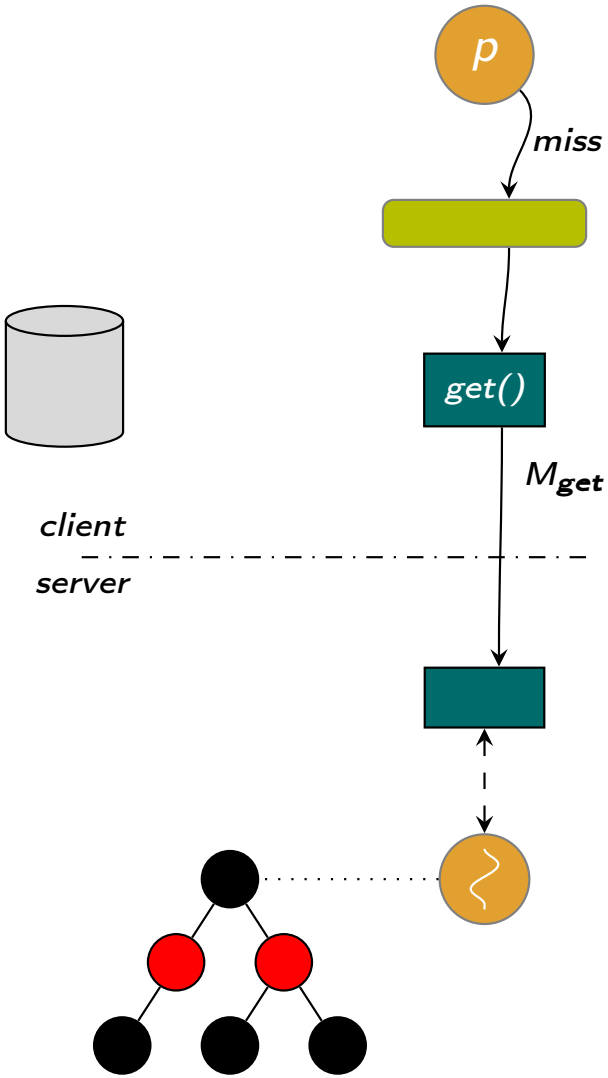
Client

- Opérations de base : *get* et *put*
- *get* bloquant
- Exécutée par le processus

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



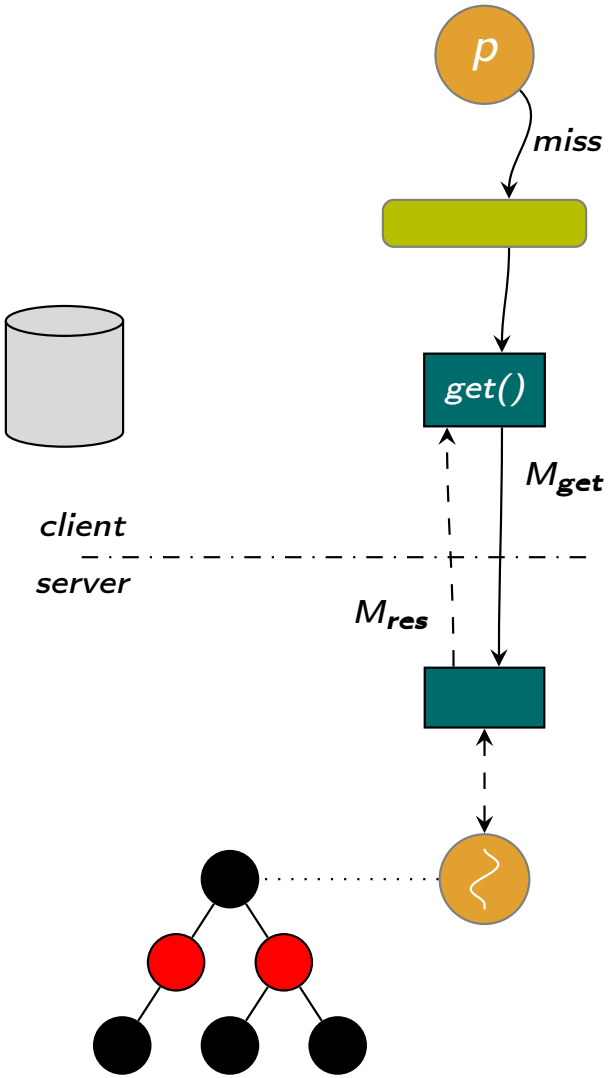
Client

- Opérations de base : *get* et *put*
- *get* bloquant
- Exécutée par le processus

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



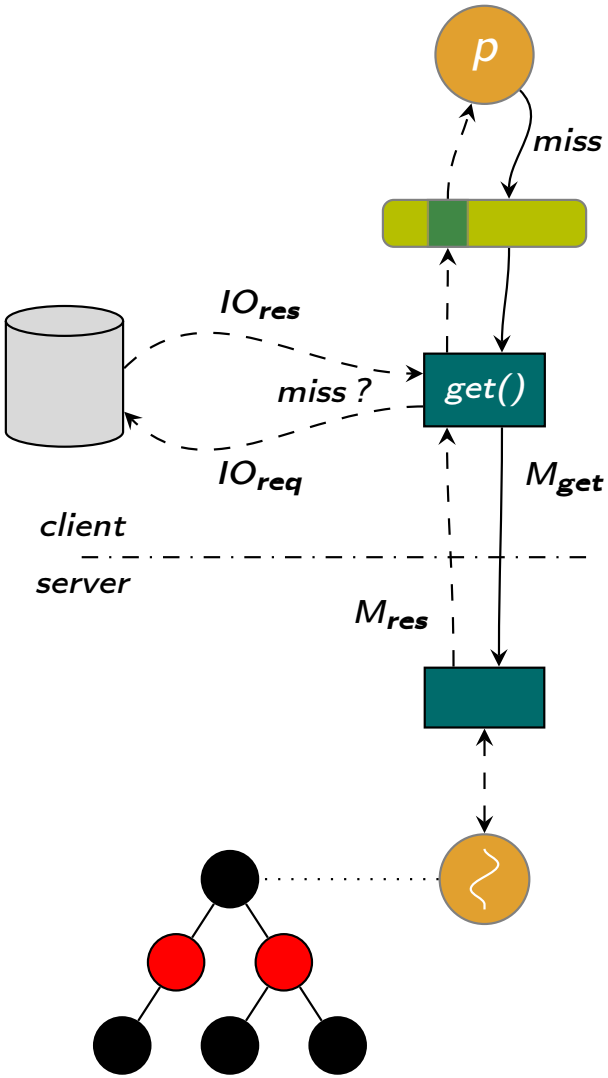
Client

- Opérations de base : *get* et *put*
- *get* bloquant
- Exécutée par le processus

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



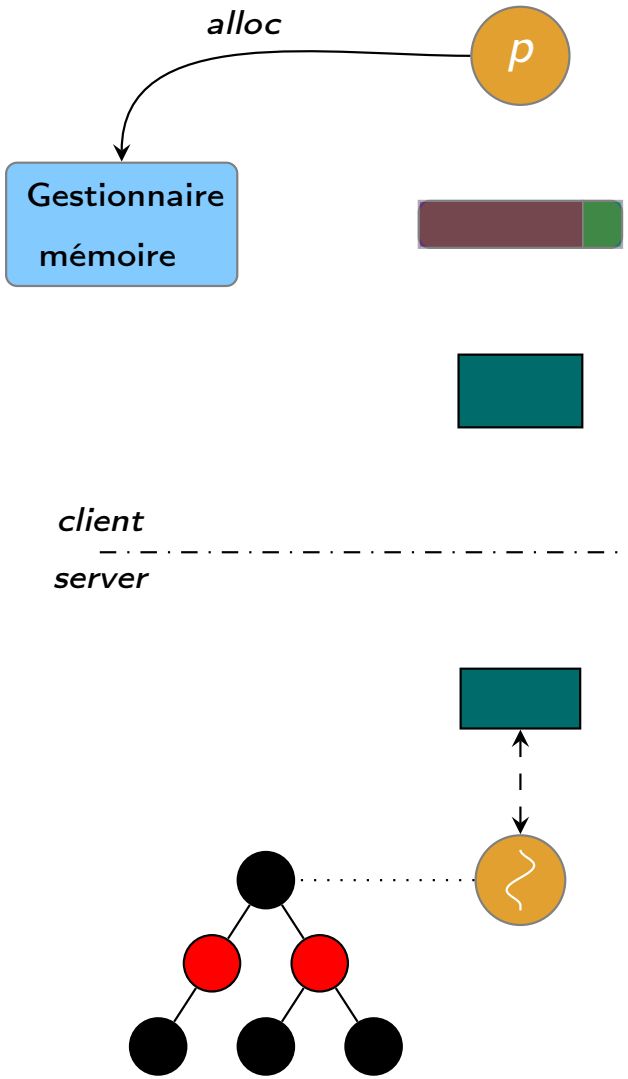
Client

- Opérations de base : *get* et *put*
- *get* bloquant
- Exécutée par le processus

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



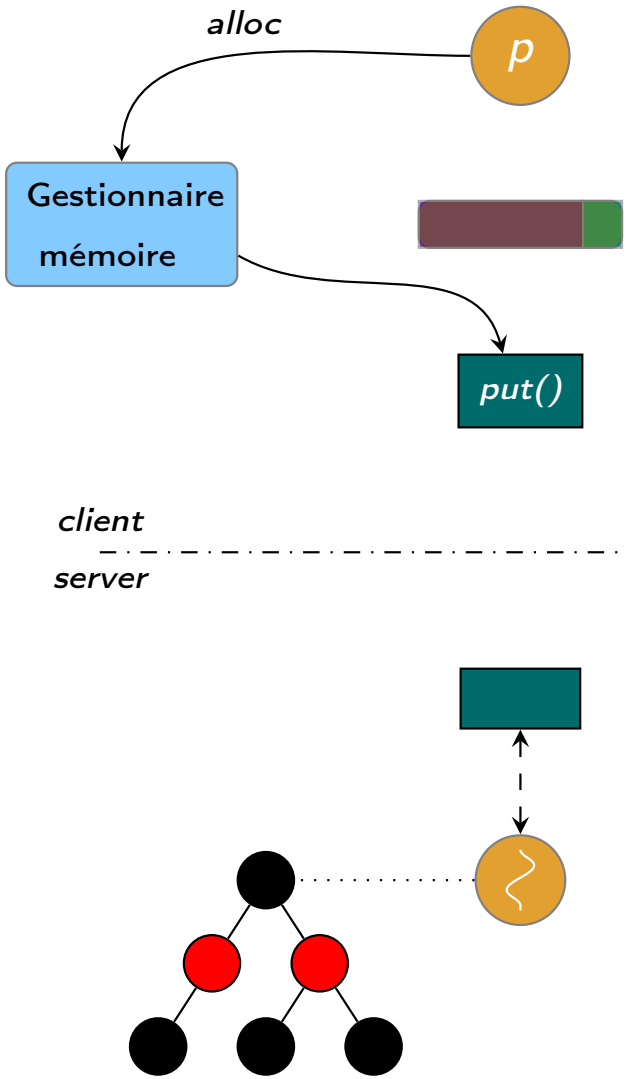
Client

- Opérations de base : *get* et *put*
- *put* appelée dans une section critique
- Exécutée par un thread dédié

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



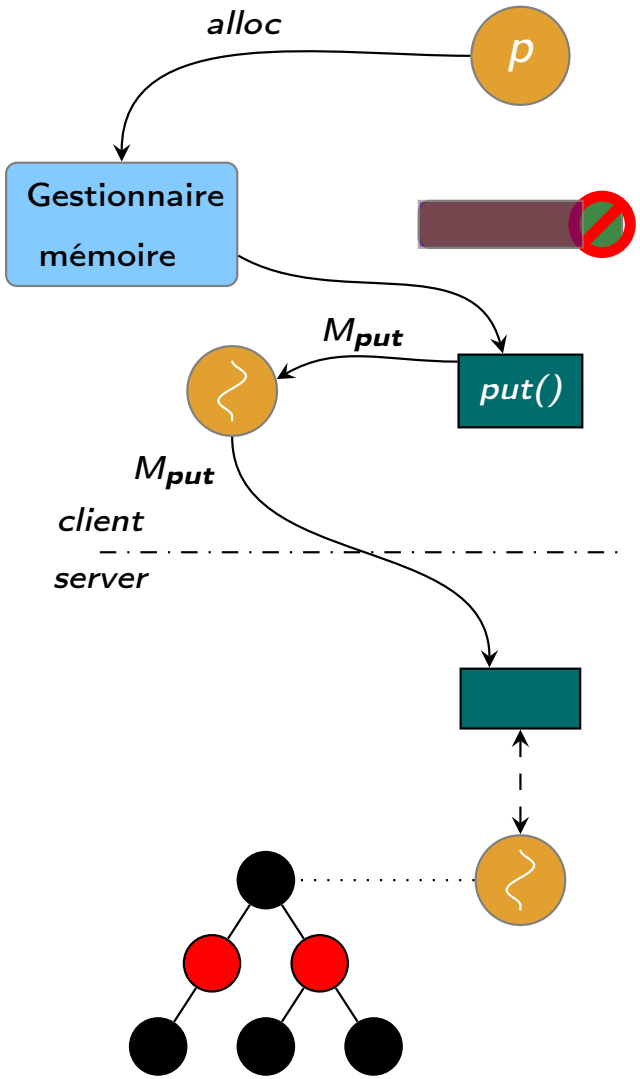
Client

- Opérations de base : *get* et *put*
- *put* appelée dans une section critique
- Exécutée par un thread dédié

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Architecture



Client

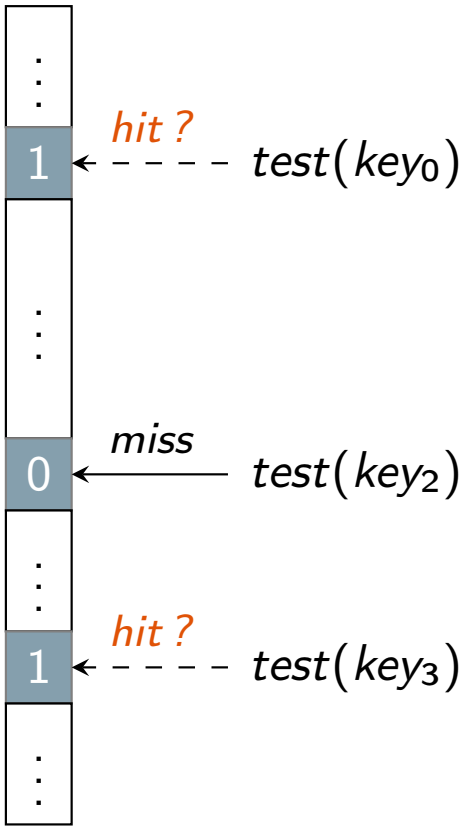
- Opérations de base : *get* et *put*
- *put* appelée dans une section critique
- Exécutée par un thread dédié

Serveur

- Thread dédié
- *Request-response*
- Arbre *rouge-noir*

Gestion des meta-données

Problème : efficacité de la gestion des meta-données

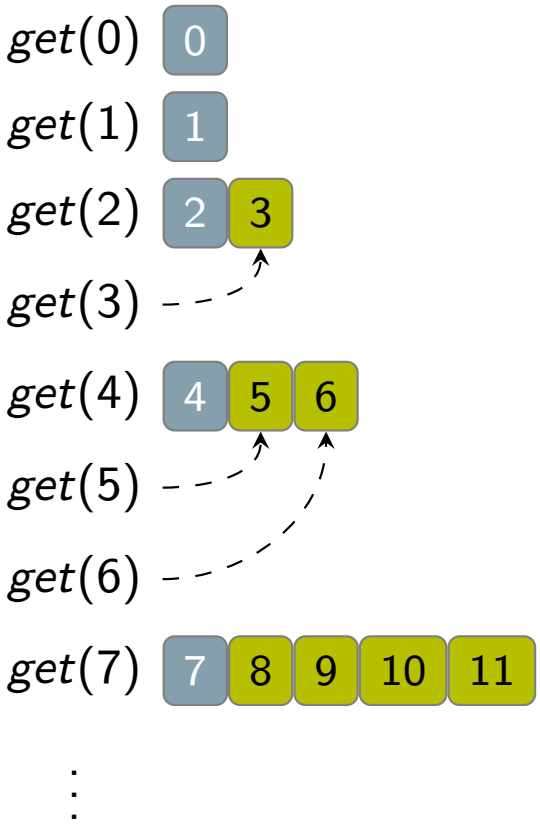


Solution : filtre de Bloom

- Structure de données probabiliste
- Compact
- Pas de faux négatifs
- Faux positifs

Gestion des accès au cache

Problème : détecter les accès séquentiels

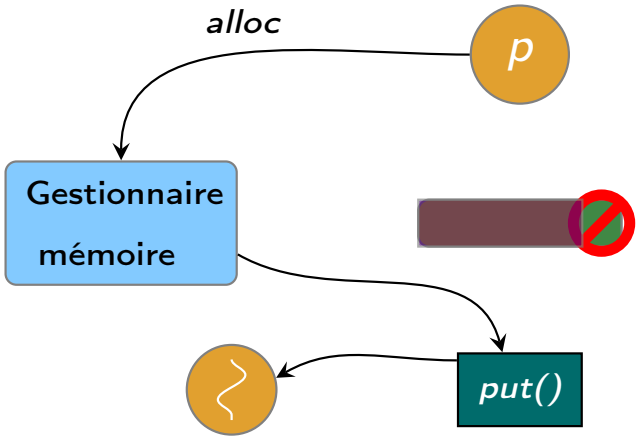


Solution : préchargement de données

- Détecte les lectures séquentielles
- Prédit les prochaines lectures
- Lit les données en avance
  - Permet d'amortir la latence réseau

Gestion des communications

Problème : empreinte mémoire des buffers réseau

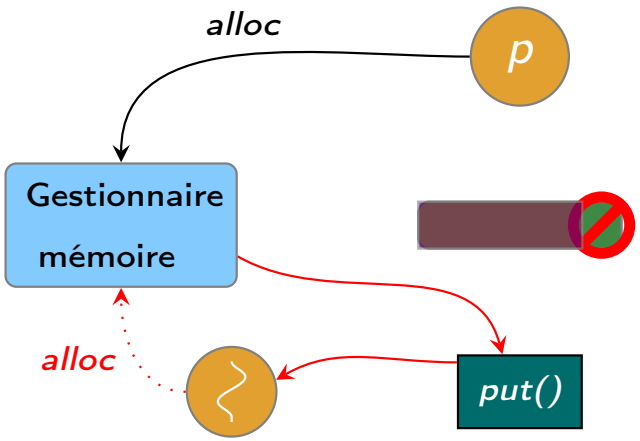


Solution : zero-copy

- Évite les recopies de TCP
- Réduit les allocations de mémoire
- Empêche les *deadlocks*

Gestion des communications

Problème : empreinte mémoire des buffers réseau



Solution : zero-copy

- Évite les recopies de TCP
- Réduit les allocations de mémoire
- Empêche les *deadlocks*

Évaluation du cache réparti

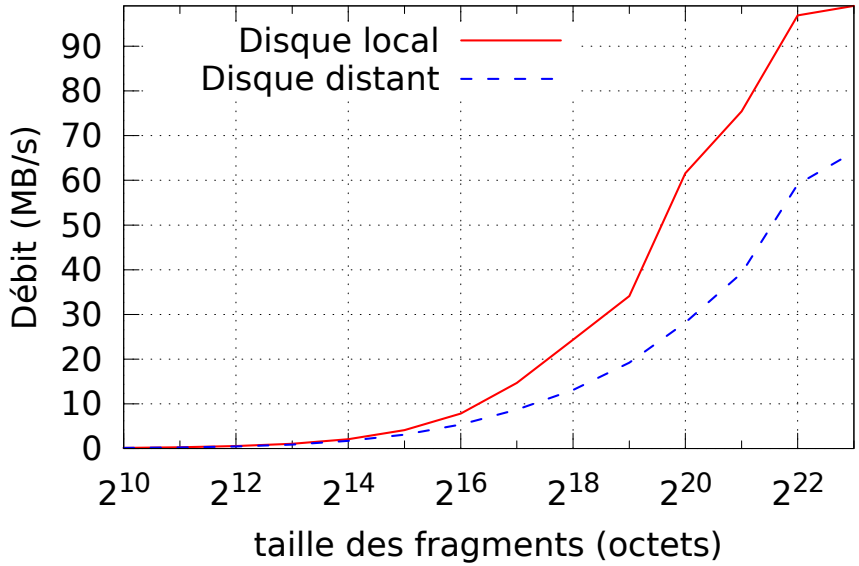
Plan d'expérience

- Surcoût des *miss* distants
  - Cache vide, *miss* systématiques
- Performances maximales
  - Cache pré-rempli, *hits* systématiques
- Performances avec mémoire saturée
  - Cache pré-rempli, mémoire locale saturée, *hits* et *puts* systématiques

Protocole expérimental

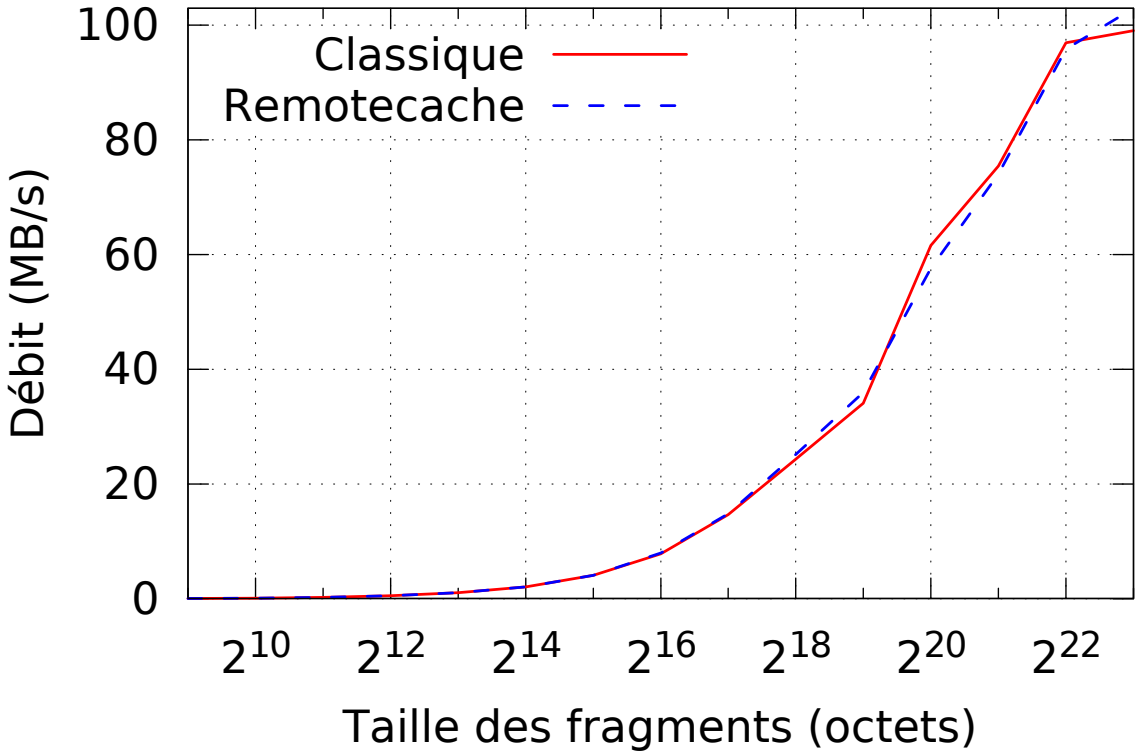
- Plateforme
  - Intel Core i7-2600 (4 cœurs hyperthreadés), 8Go de ram
  - VM « serveur de cache » (2 cœurs, 4Go de ram)
  - VM « serveur *iSCSI* » (2 cœurs, 512Mo de ram)
  - VM cliente (2 cœurs, 512Mo de ram)
    - Lectures depuis un disque virtuel local ou distant (*iSCSI*)
  - Réseau virtuel 1Gbps (RTT $\sim 600\mu$s)

- Micro-benchmark
  - Lecture de 32Mo de données
  - Fragments de 512 octets à 8Mo
  - Chaque fragment est lu à une position aléatoire



Surcoût des *miss* distants

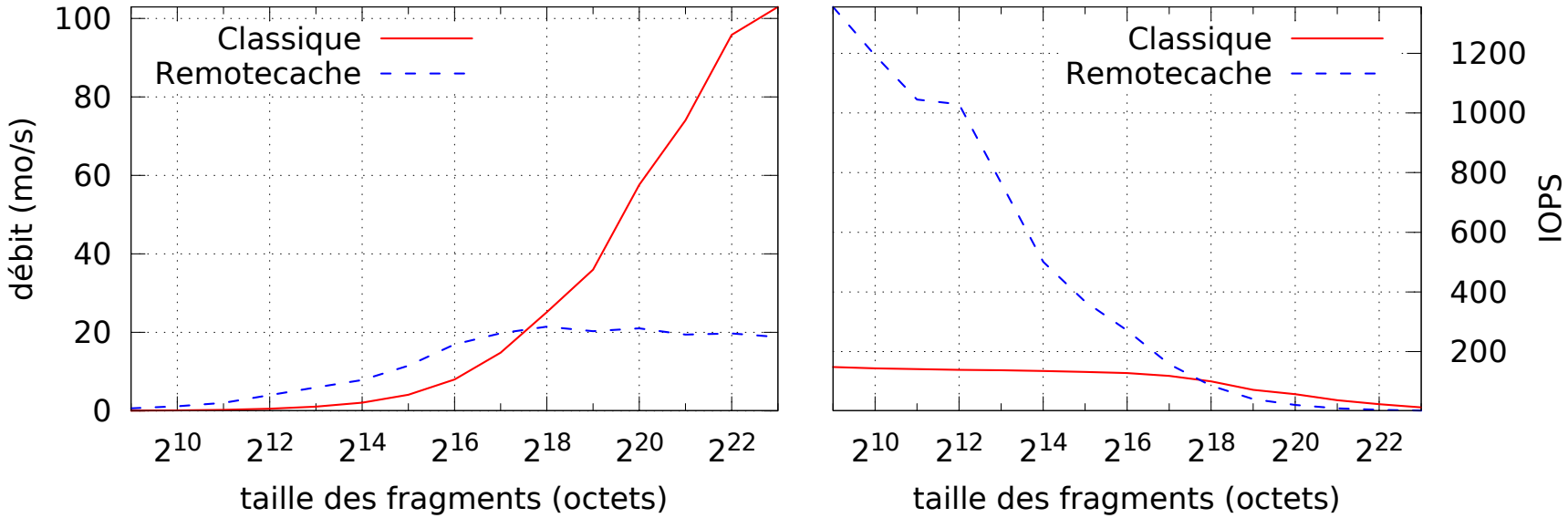
Cache distant vide



- Le filtre de Bloom évite les *miss* distants
- L'exécution du code a un faible coût sur de grands fragments

Performances maximales

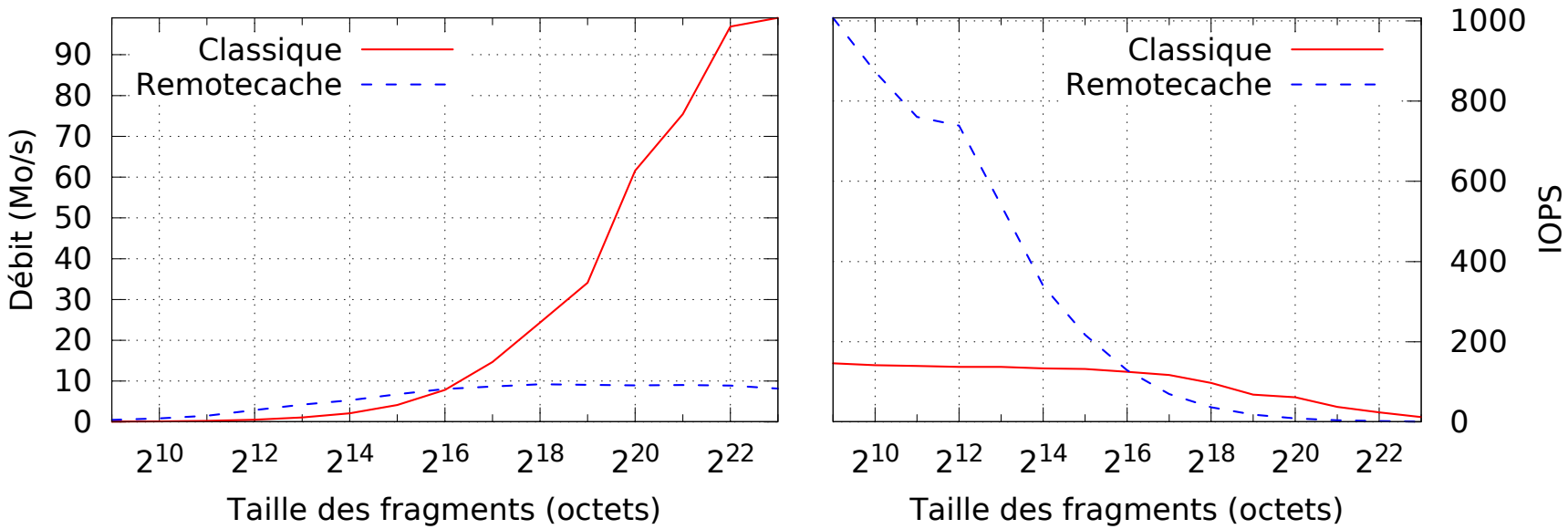
Cache distant pré-rempli



- Jusqu'à x8 avec des fragments de 1K
- Au delà de 128K, les performances sont dégradées
- Notre couche réseau plafonne à 20Mo/s

Performances maximales avec mémoire saturée

Cache distant pré-rempli, mémoire locale saturée



- Jusqu'à x6 avec des fragments de 1K
- Au delà de 64K, les performances sont dégradées

Conclusion

Synthèse

- Réalisation et évaluation d'un cache distant générique
- Peu intrusif
  - La majorité du cache est développé en tant que modules
  - $\pm$ 100 lignes modifiées, 3000 lignes de modules
  - Passe les tests de la Linux Test Suite
- Performant en lectures aléatoires ($<$ 64Ko)

Perspectives

- Benchmarks : Memcached, dm-cache, bcache,...
- Performances des lectures séquentielles
  - Optimisations de la couche réseau
  - Proposer de nouveaux algorithmes de récupération de mémoire
- Cache coopératif
  - Qualité de service