Virtualisation efficace d'architectures NUMA

Thèse de doctorat de Sorbonne Université

Gauthier Voron

Directeurs de thèse :

Gaël Thomas	Professeur, Télécom SudParis
Pierre Sens	Professeur, Sorbonne Université

Membres du jury :

Emmanuelle Encrenaz	Maître de conférences HDR, Sorbonne Université
Rachid Guerraoui	Professeur, École Polytechnique Fédérale de Lausanne
Laurent Réveillère	Professeur, Université de Bordeaux
Vivien Quéma	Professeur, Grenoble INP / ENSIMAG
Pierre Sens	Professeur, Sorbonne Université
Gaël Thomas	Professeur, Télécom SudParis

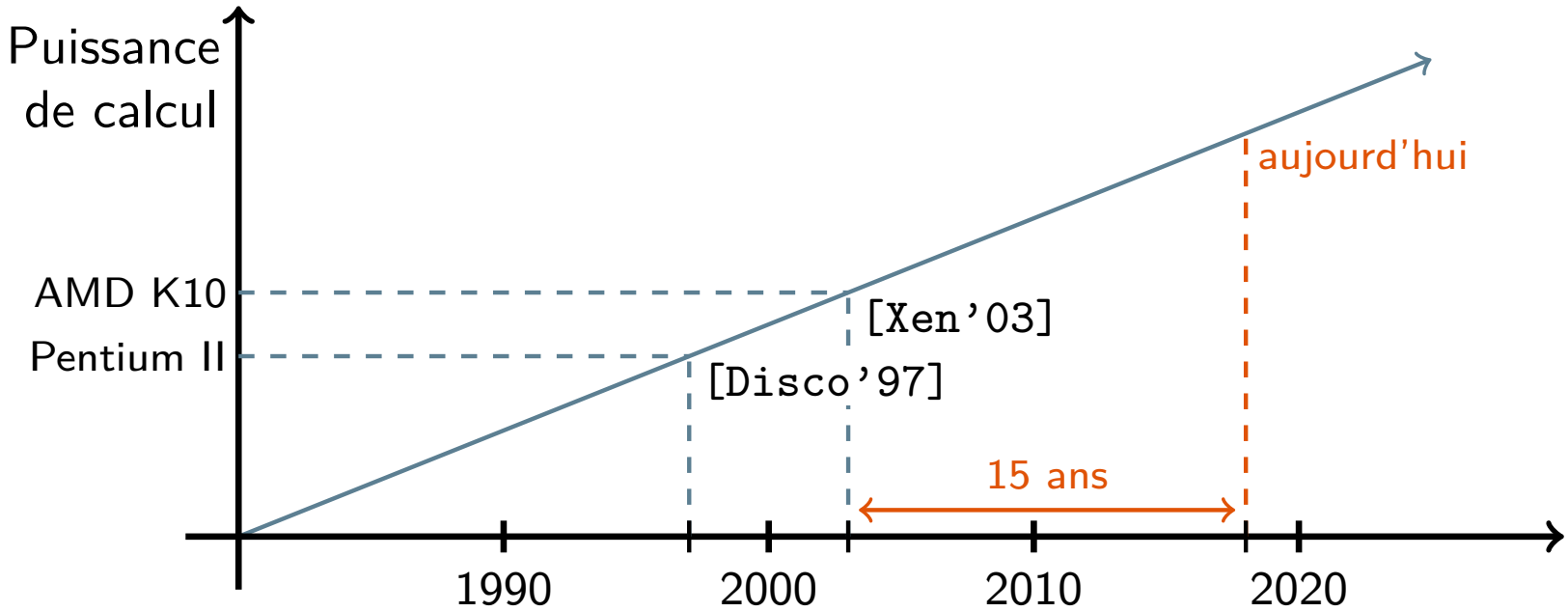
SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

LIP6

TELECOM SudParis

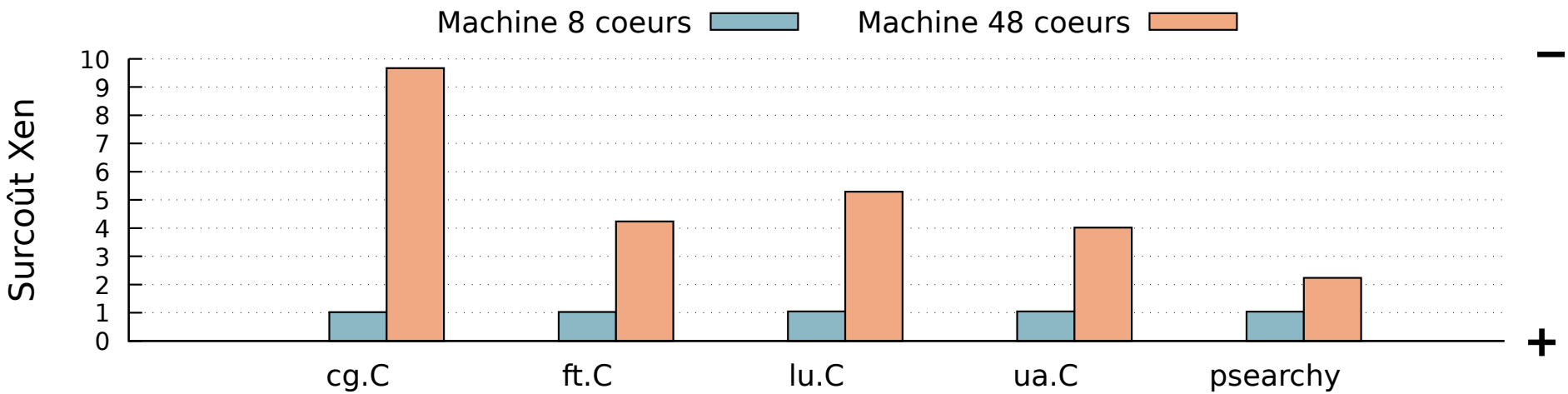
Rendre le cloud computing efficace

- Le cloud computing : effectuer des calculs sur les machines d'un tiers



- Le cloud computing repose sur des technologies vieilles de 15 ans
- La puissance des machines a augmenté

Surcoût du cloud computing aujourd'hui



- Les technologies du cloud n'arrivent pas à exploiter la puissance des machines modernes

Plan

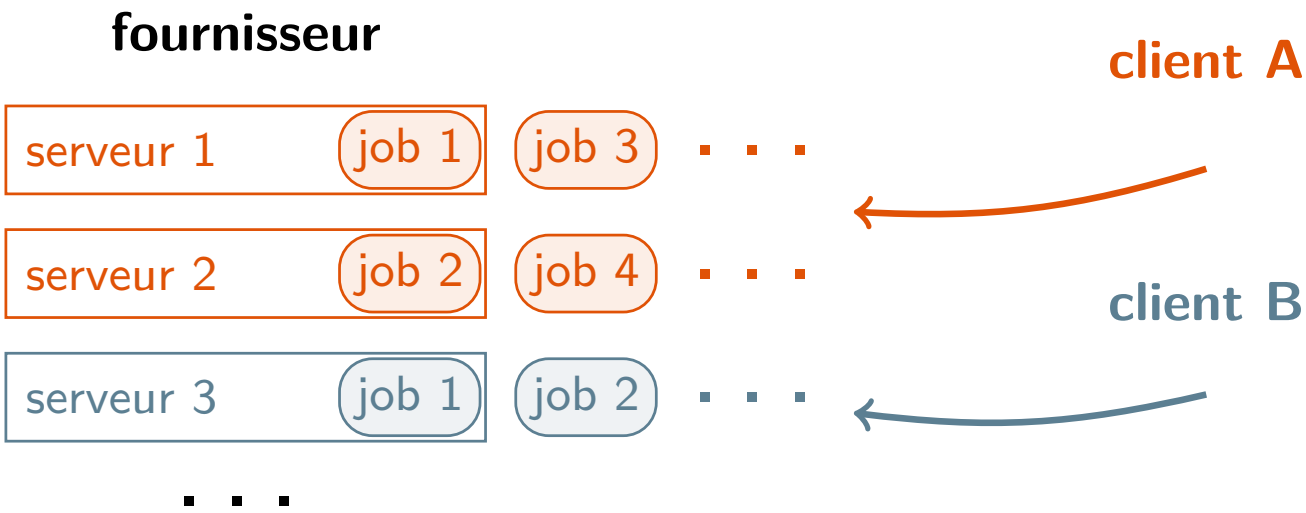
Cloud computing : exécution de jobs en série

- Cloud computing : modèle de gestion de ressources
  - Un fournisseur de ressources → des serveurs (CPU + mémoire + disque)
  - Plusieurs consommateurs de ressources

- Les consommateurs veulent exécuter des jobs

Cloud computing : exécution de jobs en série

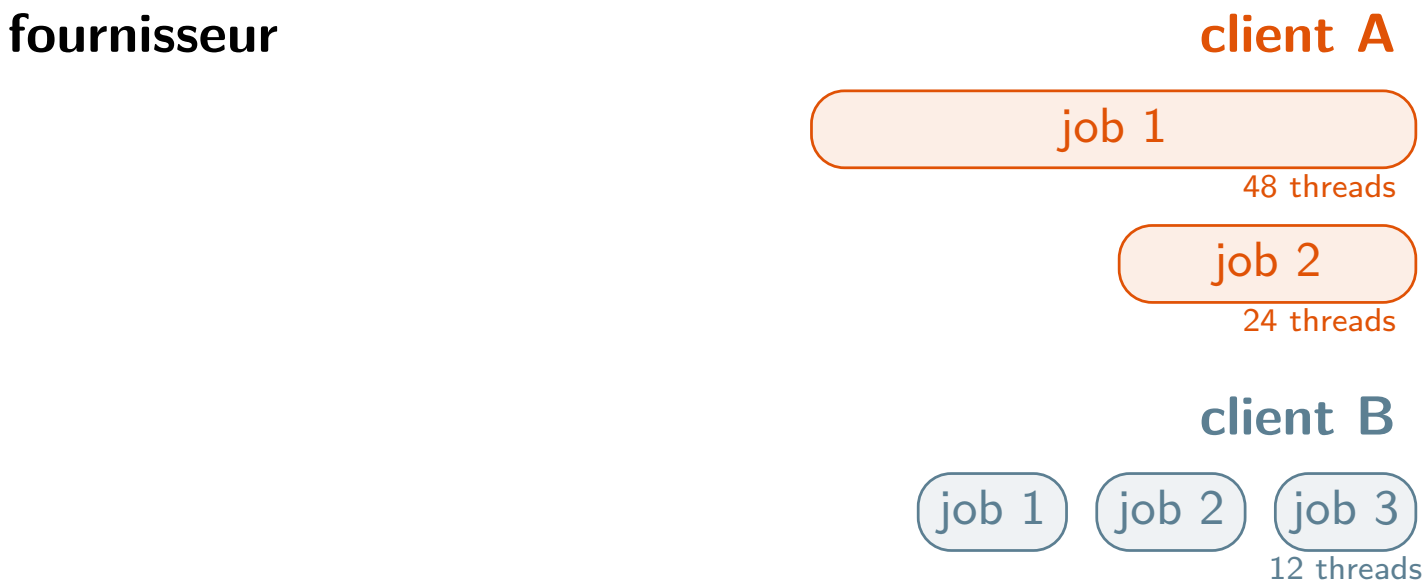
- Cloud computing : modèle de gestion de ressources
  - Un fournisseur de ressources → des serveurs (CPU + mémoire + disque)
  - Plusieurs consommateurs de ressources



- Les consommateurs veulent exécuter des jobs
  - Un consommateur peut louer des serveurs pour exécuter ses jobs
  - Un consommateur veut exécuter ses jobs rapidement

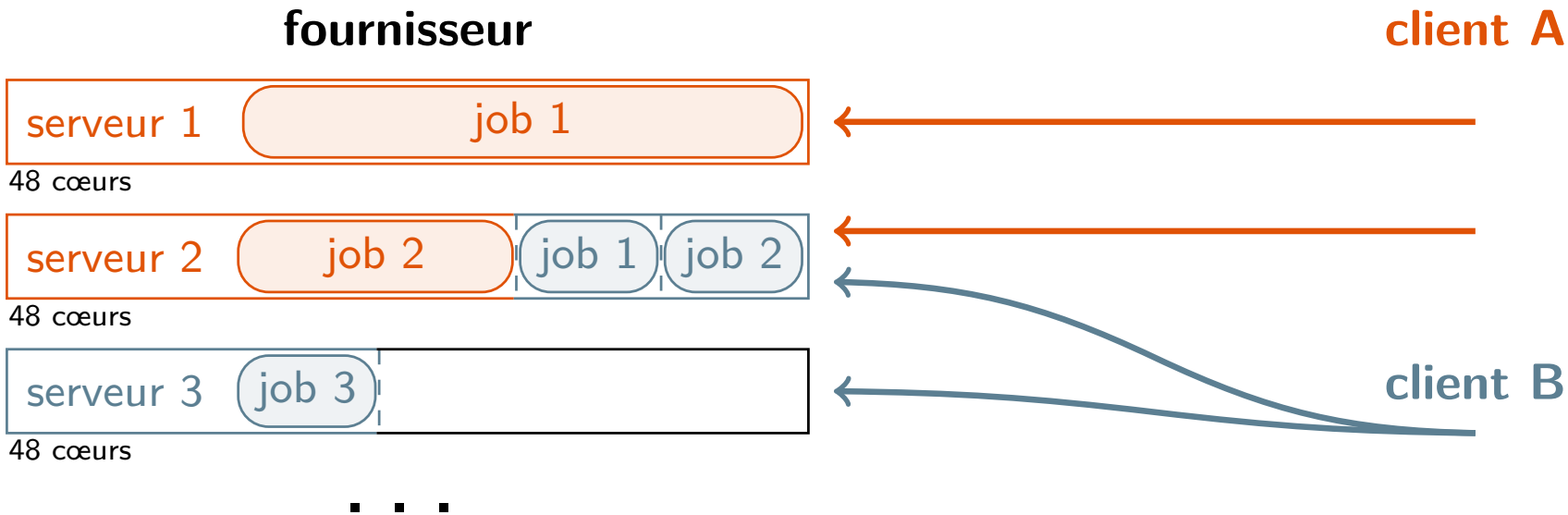
Cloud computing : allocation de ressources à la demande

- Chaque consommateur a des besoins différents
  - Nombre de cœurs pour un job
  - Quantité de mémoire pour un job
- Le fournisseur doit pouvoir satisfaire toutes ces demandes



Cloud computing : allocation de ressources à la demande

- Chaque consommateur a des besoins différents
  - Nombre de cœurs pour un job
  - Quantité de mémoire pour un job
- Le fournisseur doit pouvoir satisfaire toutes ces demandes



- Le fournisseur dispose de puissantes machines multicœurs
  - Ce qu'un consommateur peut demander de mieux pour un job
- Le fournisseur peut louer une partie de la machine

La virtualisation : un outil pour fragmenter les ressources

- Le fournisseur doit fragmenter ses serveurs entre plusieurs clients

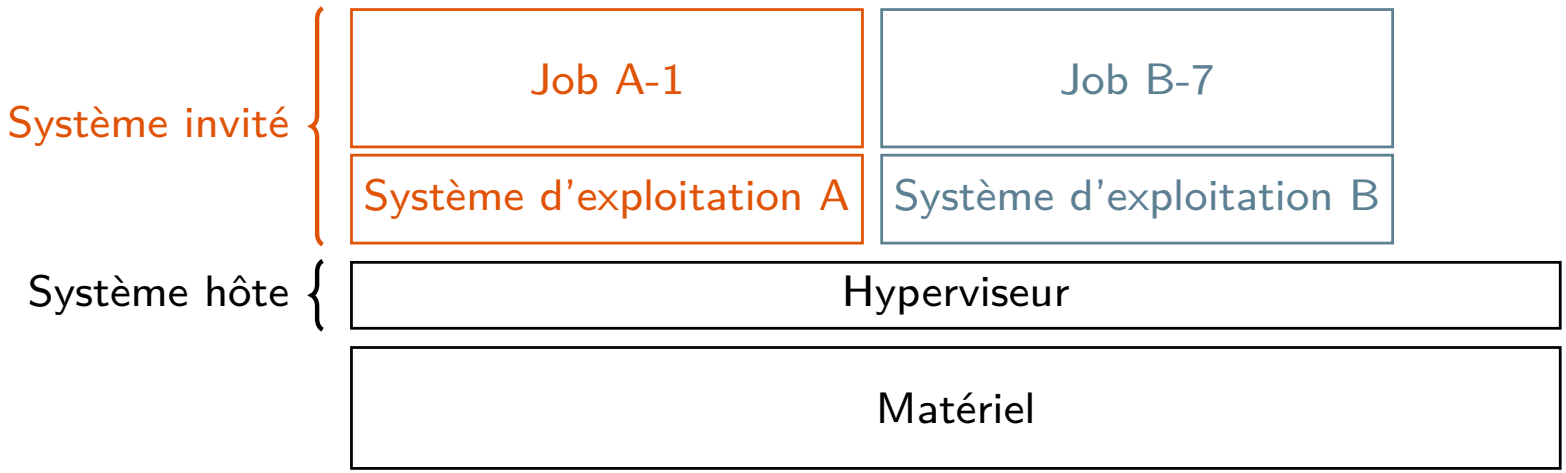
- Chaque client veut utiliser son propre environnement
  - Déployer son propre système d'exploitation
  - Déployer ses propres outils et applications

- Le fournisseur doit faire cohabiter plusieurs systèmes sur une machine
- Chaque système doit être isolé des autres

- Le fournisseur a recours à la virtualisation
  - Utilisation d'un hyperviseur

Objectifs de l'hyperviseur : isolation et performance

- Hyperviseur : couche logicielle intercalée entre le système d'exploitation et le matériel



- Rôles de l'hyperviseur :
  - Isoler les systèmes invités les uns des autres
  - Isoler les systèmes invités du matériel
  - Permettre aux systèmes invités de s'exécuter rapidement

Plan

Défi des applications modernes : le parallélisme

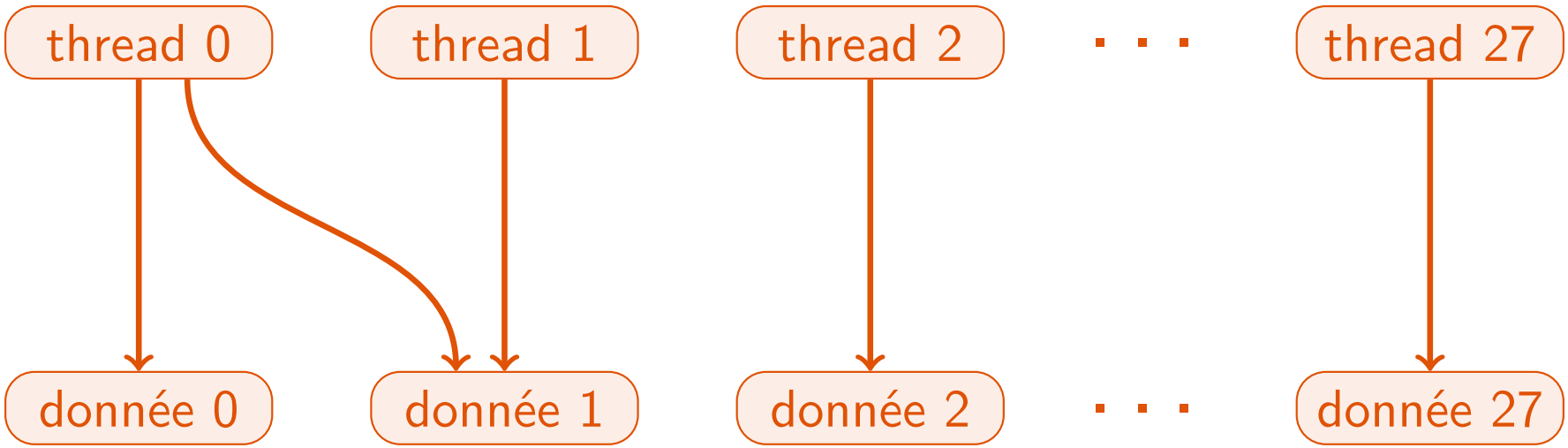
Il y a deux manières de rendre un application plus rapide

- Exécuter l'application sur un processeur plus rapide
  - La fréquence des processeurs stagne depuis les années 2000

- Exécuter l'application sur plusieurs cœurs en parallèle
  - Le nombre de cœurs par machine augmente depuis les années 2000
  - Efficace si les cœurs travaillent effectivement en parallèle
  - Efficace s'il y a peu de synchronisations

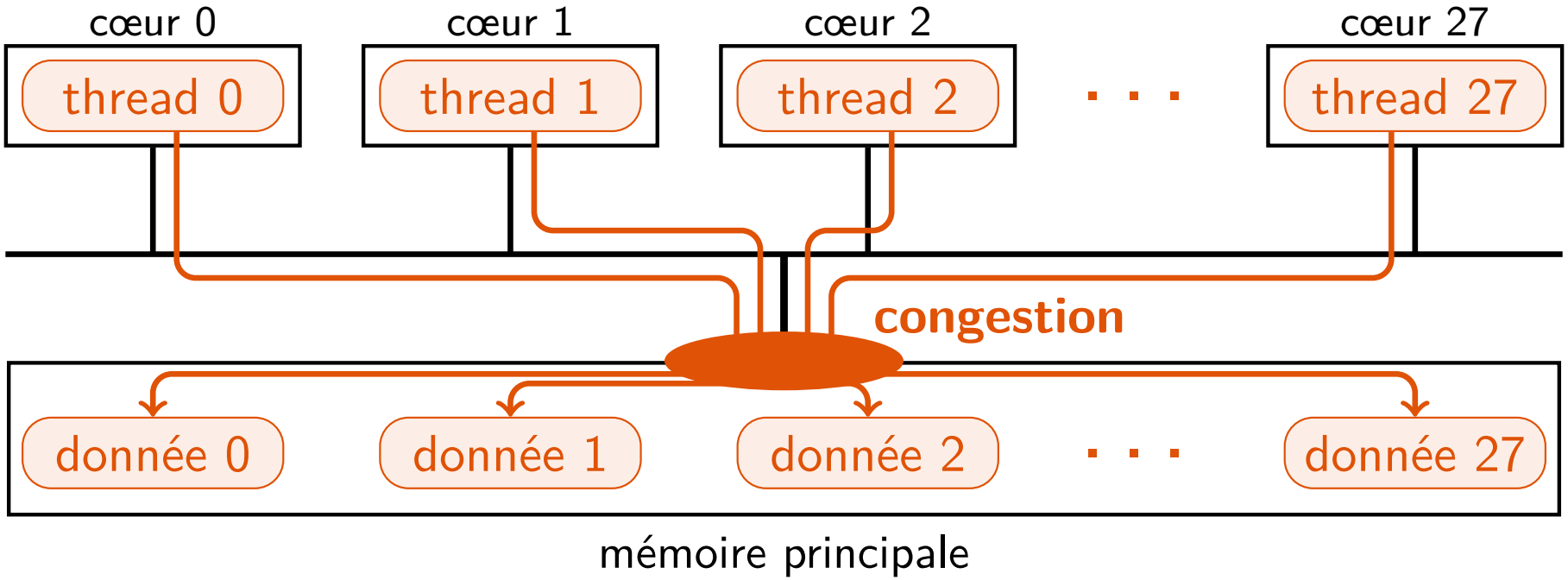
Défi des machines multicœur : la contention mémoire

- Les applications actuelles utilisent efficacement des dizaines de cœurs
  - Chaque thread accède à ses propres données → pas de synchronisation



Défi des machines multicœur : la contention mémoire

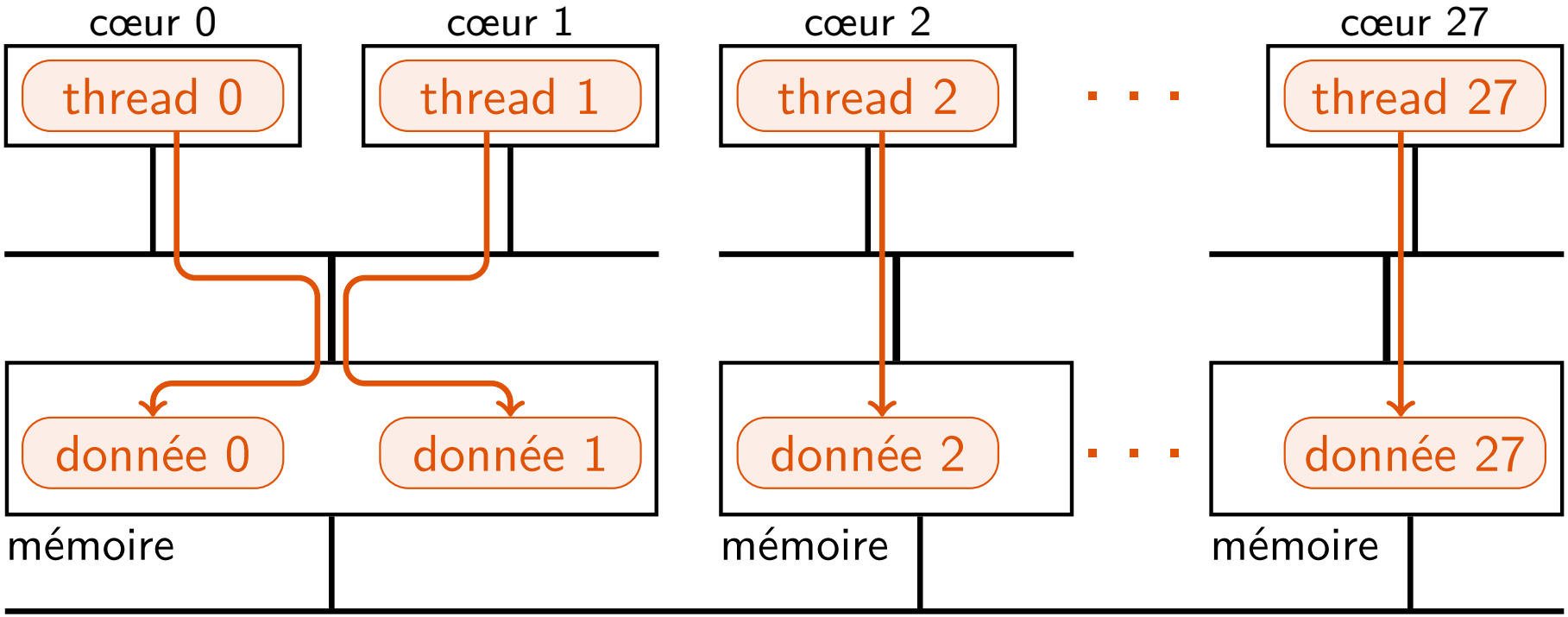
- Les applications actuelles utilisent efficacement des dizaines de cœurs
  - Chaque thread accède à ses propres données → pas de synchronisation



- Les données accédées sont toutes en mémoire principale
- Tous les cœurs accèdent à la mémoire via un même contrôleur
  - Si trop de cœurs accèdent aux données, il y a congestion du contrôleur

Défi des machines multicœur : la contention mémoire

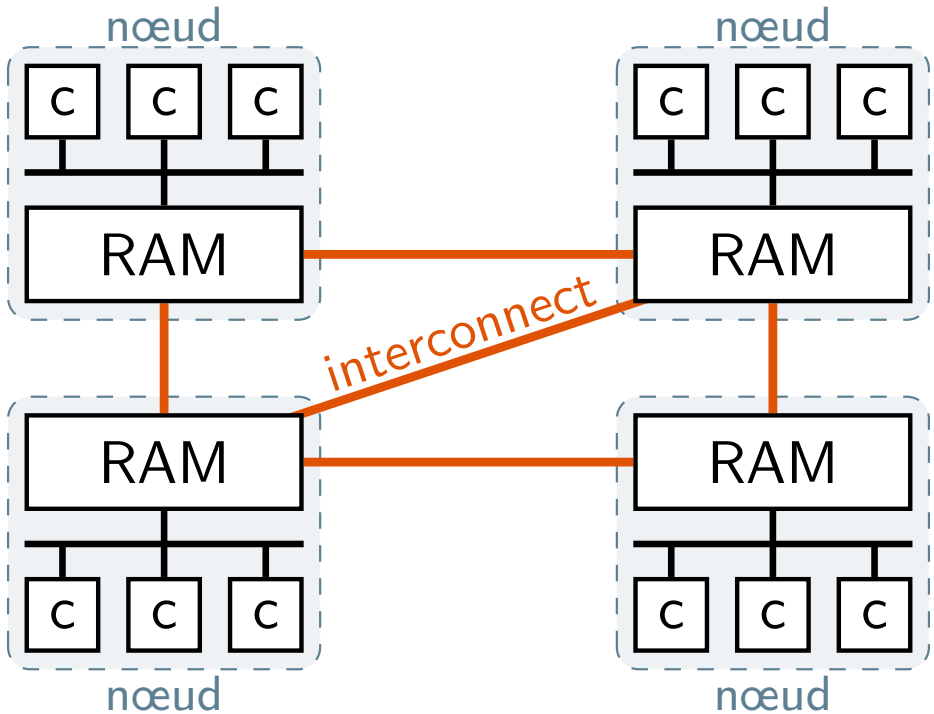
- Les applications actuelles utilisent efficacement des dizaines de cœurs
  - Chaque thread accède à ses propres données → pas de synchronisation



- Les données accédées sont toutes en mémoire principale
- Tous les cœurs accèdent à la mémoire via un même contrôleur
  - Si trop de cœurs accèdent aux données, il y a congestion du contrôleur
- Il faut distribuer la charge mémoire sur plusieurs contrôleurs

Architecture NUMA : problème du placement des données

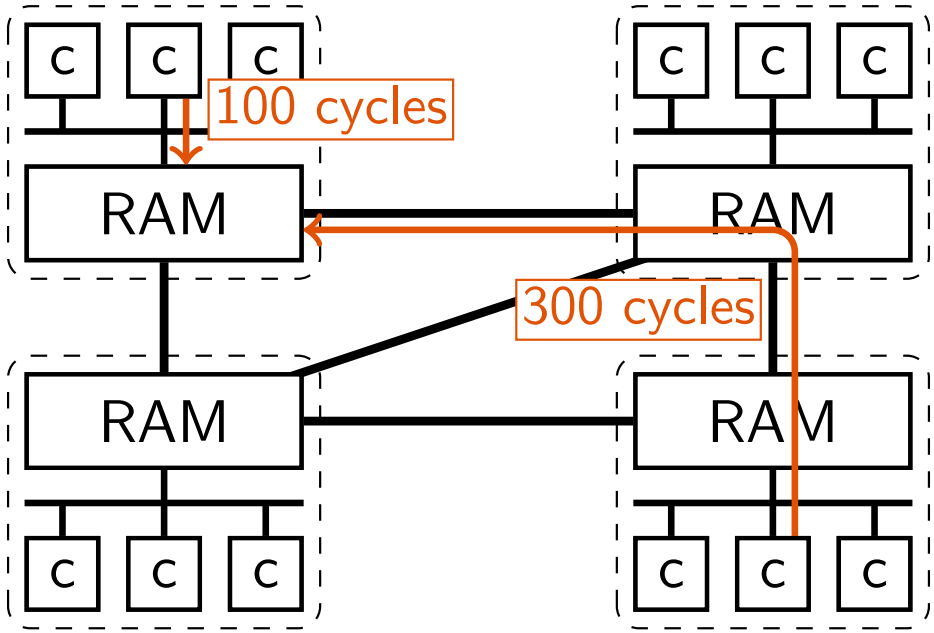
- Un multicœur NUMA est composé d'un ensemble de nœud NUMA
  - Chaque nœud est composé de cœurs et d'un contrôleur mémoire
  - Les nœuds sont reliés entre eux par l'interconnect
  - Le matériel route chaque requêtes mémoire vers le bon nœud



- Fonctionellement équivalent à un multicœur classique
  - Exécution possible de programmes *legacy*
- Besoin de traitements spécifiques pour être performant

Architecture NUMA : problème du placement des données

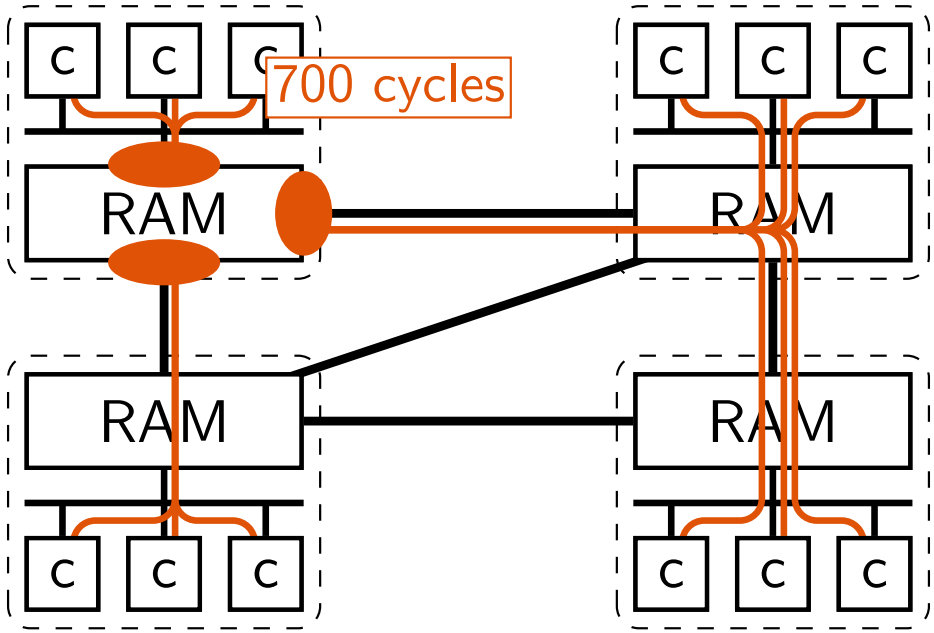
- Un multicœur NUMA est composé d'un ensemble de nœud NUMA
  - Chaque nœud est composé de cœurs et d'un contrôleur mémoire
  - Les nœuds sont reliés entre eux par l'interconnect
  - Le matériel route chaque requêtes mémoire vers le bon nœud



- La latence mémoire dépend de :
  - La localité des accès → local : 100 cycles / distant : 300 cycles

Architecture NUMA : problème du placement des données

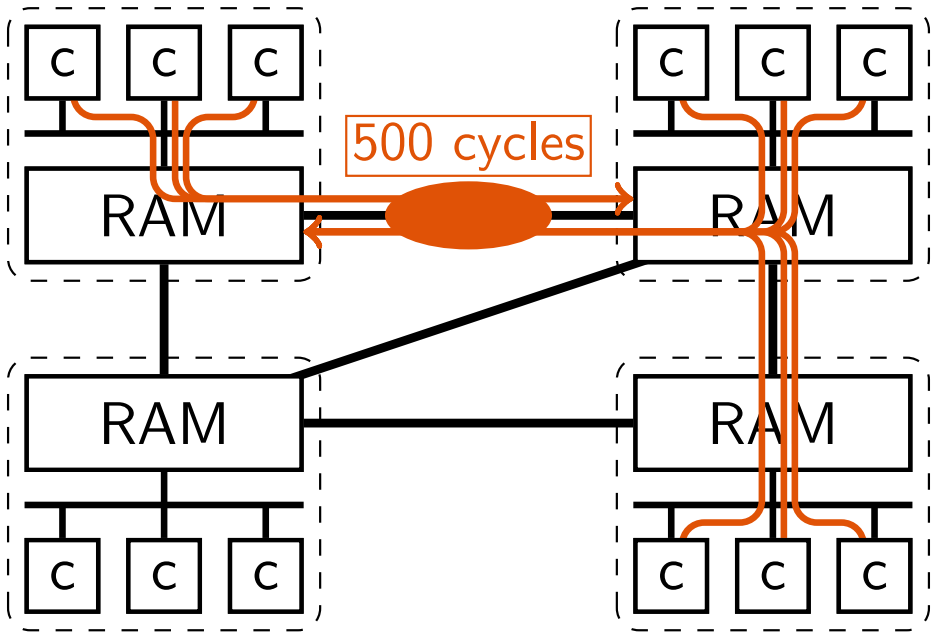
- Un multicœur NUMA est composé d'un ensemble de nœud NUMA
  - Chaque nœud est composé de cœurs et d'un contrôleur mémoire
  - Les nœuds sont reliés entre eux par l'interconnect
  - Le matériel route chaque requêtes mémoire vers le bon nœud



- La latence mémoire dépend de :
  - La localité des accès → local : 100 cycles / distant : 300 cycles
  - L'absence de congestion des contrôleurs → 700 cycles

Architecture NUMA : problème du placement des données

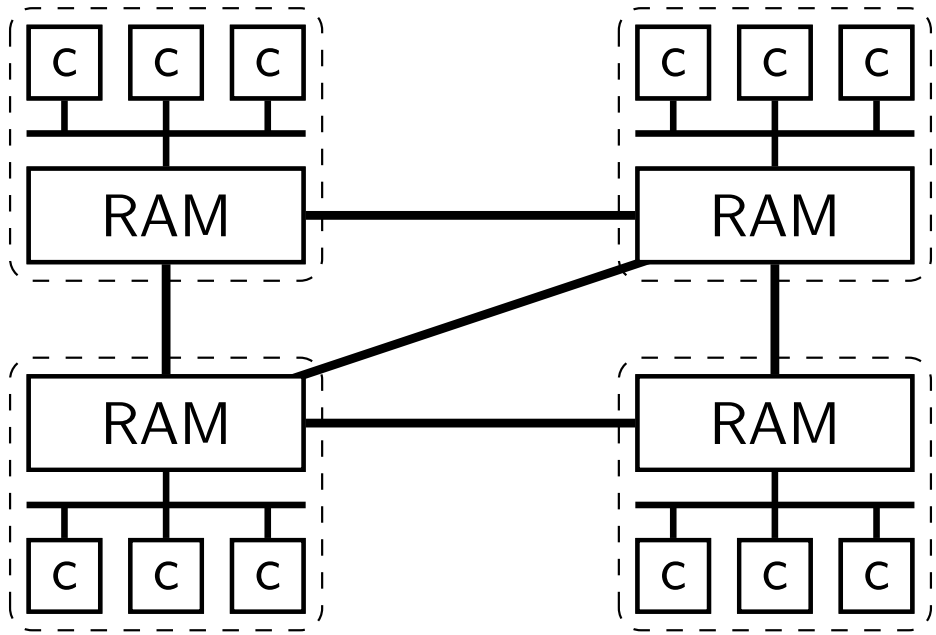
- Un multicœur NUMA est composé d'un ensemble de nœud NUMA
  - Chaque nœud est composé de cœurs et d'un contrôleur mémoire
  - Les nœuds sont reliés entre eux par l'interconnect
  - Le matériel route chaque requêtes mémoire vers le bon nœud



- La latence mémoire dépend de :
  - La localité des accès → local : 100 cycles / distant : 300 cycles
  - L'absence de congestion des contrôleurs → 700 cycles
  - L'absence de congestion de l'interconnect → 500 cycles

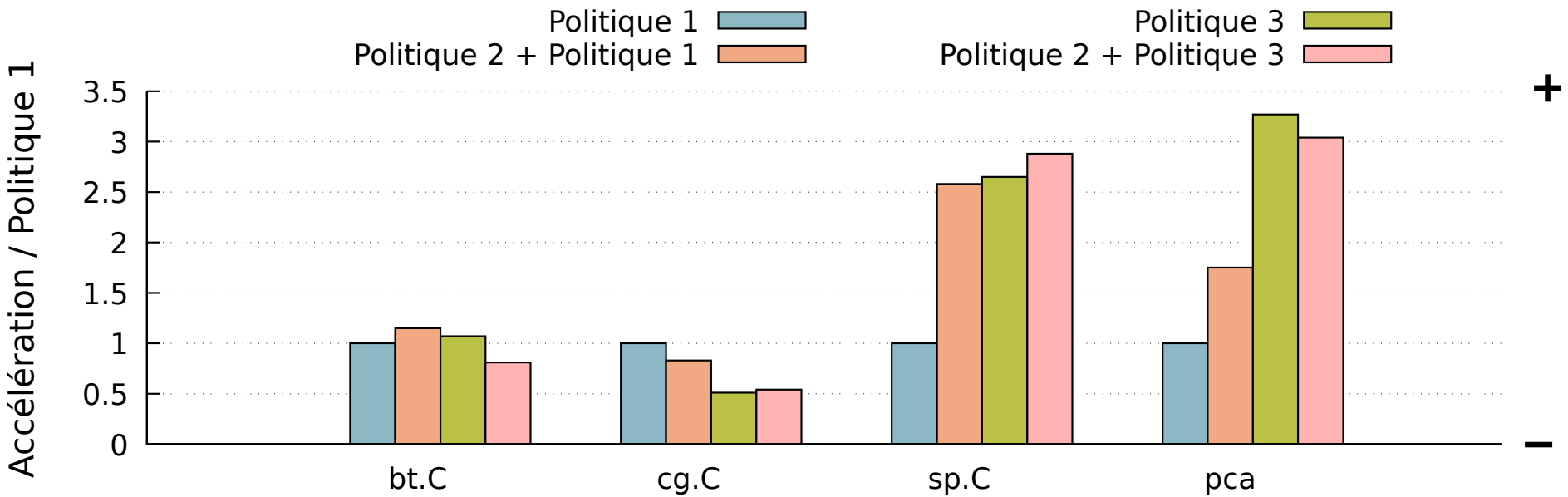
Architecture NUMA : problème du placement des données

- Un multicœur NUMA est composé d'un ensemble de nœud NUMA
  - Chaque nœud est composé de cœurs et d'un contrôleur mémoire
  - Les nœuds sont reliés entre eux par l'interconnect
  - Le matériel route chaque requêtes mémoire vers le bon nœud



- La latence mémoire dépend du placement des tâches / données
- Peu d'applications savent gérer finement ce paramètre
- Le système d'exploitation propose différentes politiques de placement

Besoin de multiples politiques mémoire



- Il n'y a pas aujourd'hui une politique meilleure que toutes les autres
- La politique doit être sélectionnée en fonction de l'application
  - Par l'application elle-même (`set_mempolicy()`)
  - Par l'utilisateur (`numactl`)

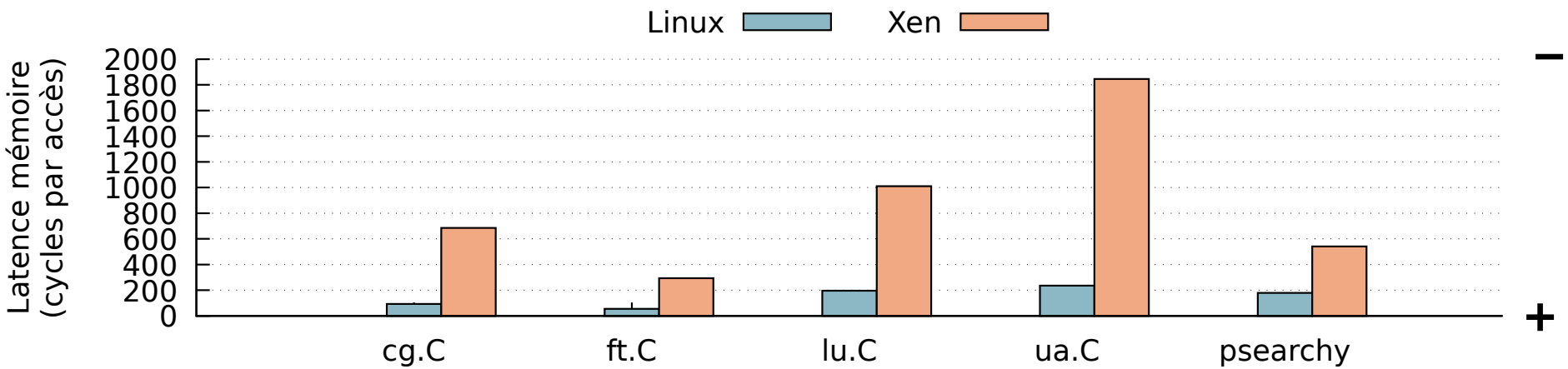
- Le système d'exploitation doit laisser le choix à l'utilisateur entre différentes politiques mémoire

Plan

L'inefficacité des machines virtuelles NUMA

- La virtualisation est peu performante sur des architectures NUMA
  - Les architectures NUMA causent des problèmes de latence mémoire

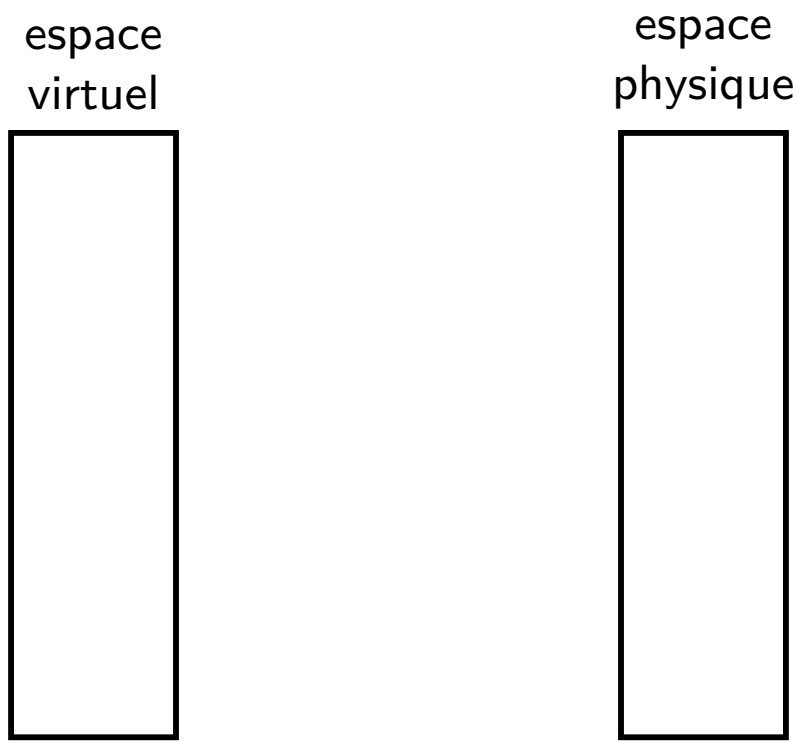
- Latence mémoire d'un système invité sur une machine NUMA :



- Hypothèse : la virtualisation affecte les politiques mémoire

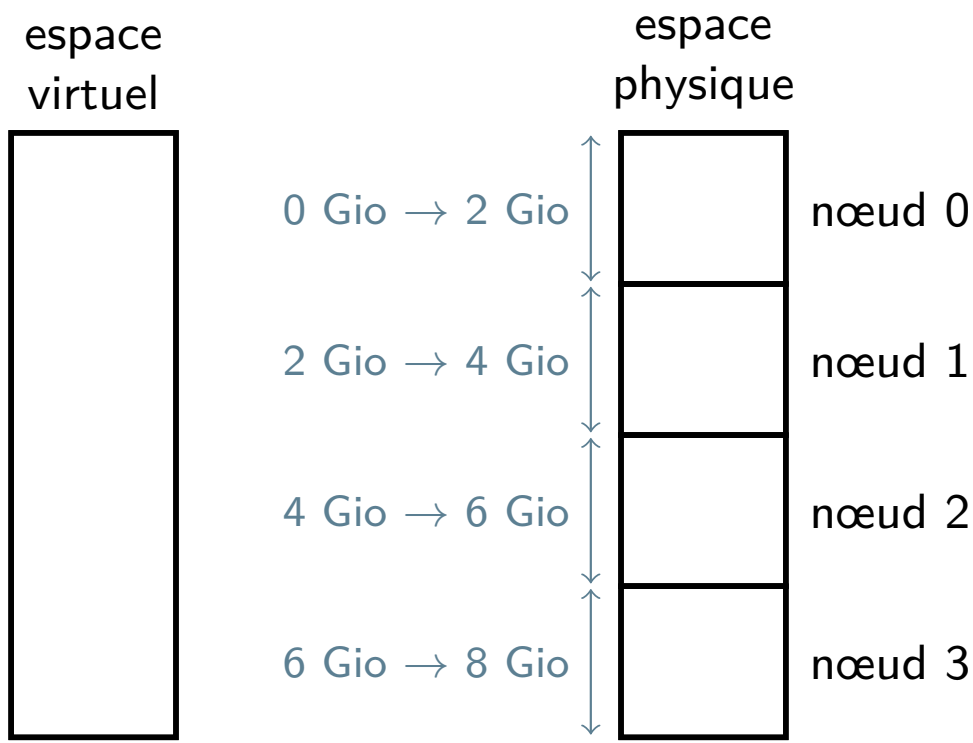
Placement NUMA des données en mode natif

- Un système travaille avec deux espaces d'adressages
  - L'espace d'adressage virtuel
  - L'espace d'adressage physique



Placement NUMA des données en mode natif

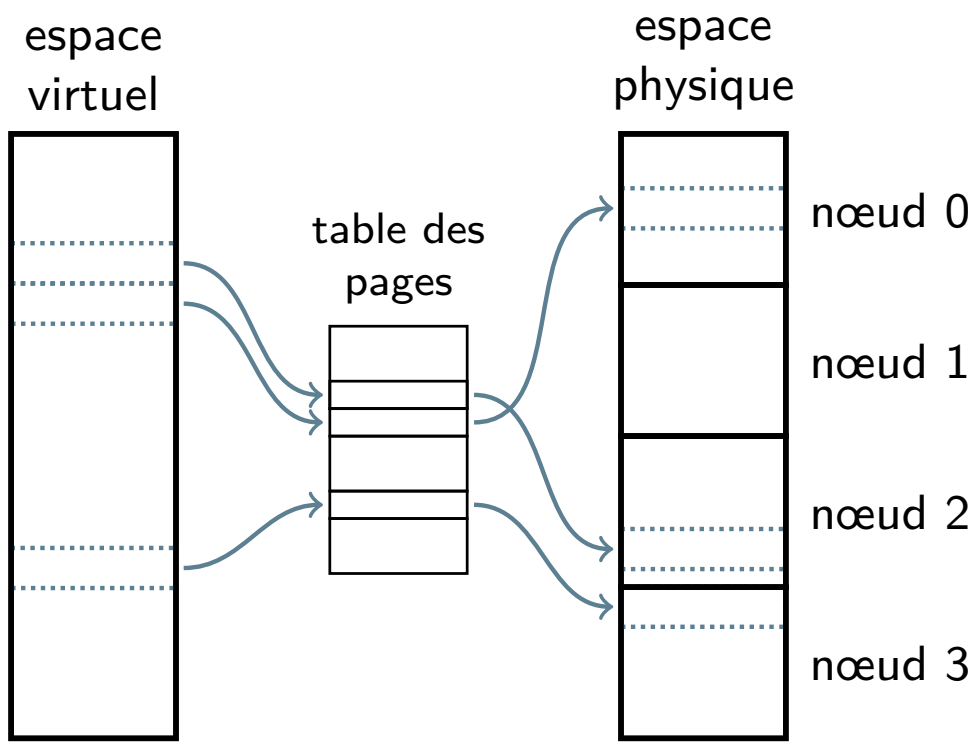
- Un système travaille avec deux espaces d'adressages
  - L'espace d'adressage virtuel
  - L'espace d'adressage physique



- Les nœuds NUMA sont une partition de l'espace physique
  - Le matériel route les requêtes mémoire en fonction de l'adresse physique

Placement NUMA des données en mode natif

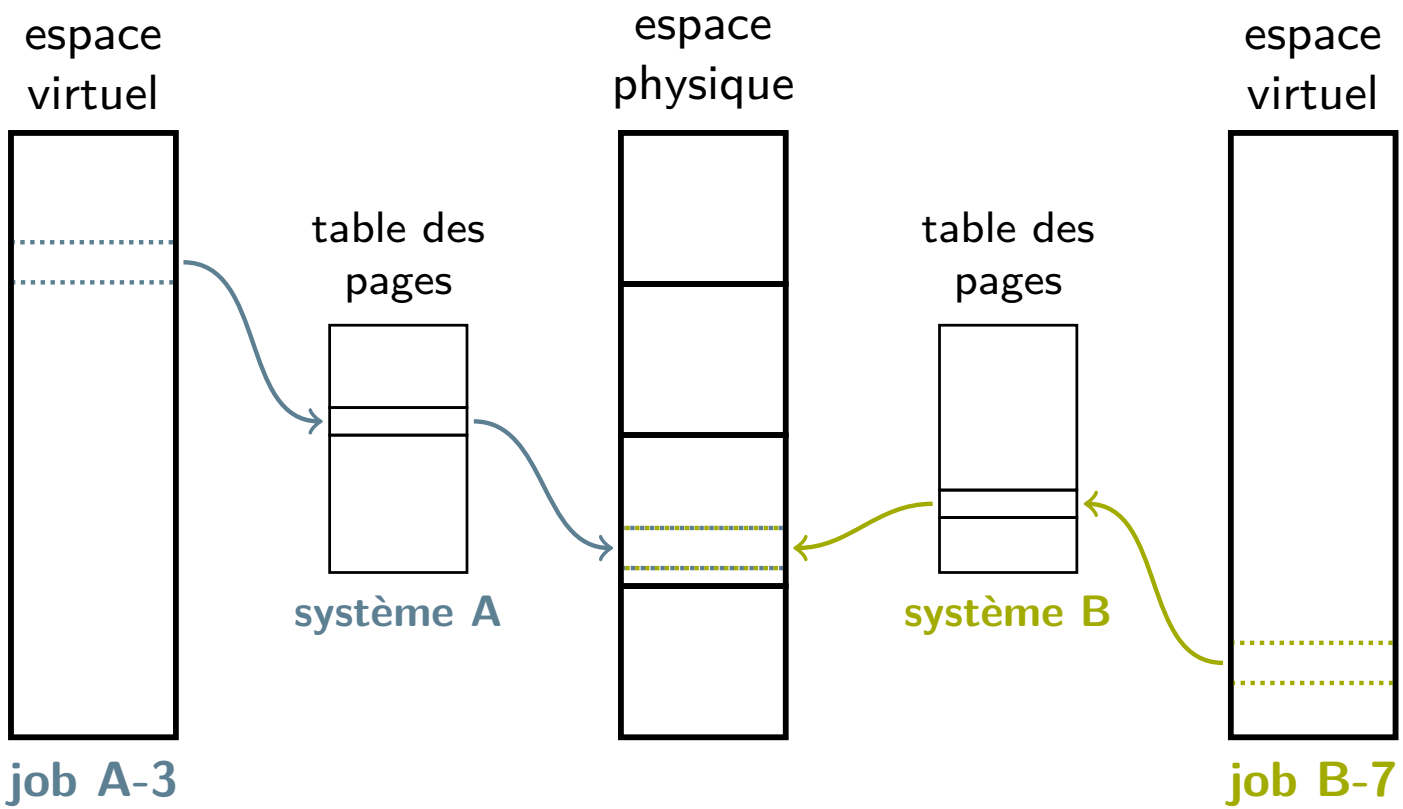
- Un système travaille avec deux espaces d'adressages
  - L'espace d'adressage virtuel
  - L'espace d'adressage physique



- Les nœuds NUMA sont une partition de l'espace physique
  - Le matériel route les requêtes mémoire en fonction de l'adresse physique
- Le système d'exploitation utilise la traduction d'adresses pour choisir le placement NUMA des données

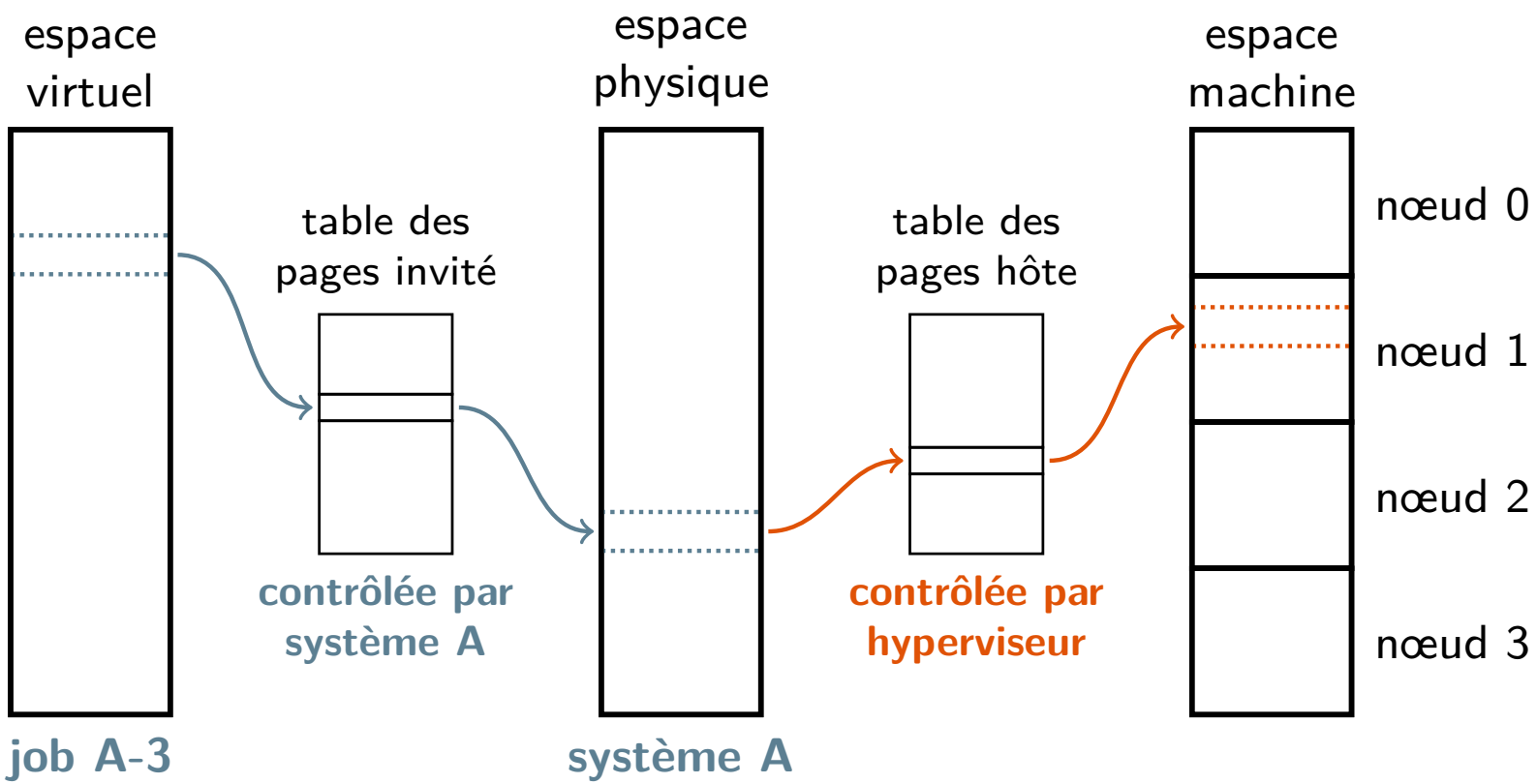
Placement NUMA à plusieurs systèmes : collisions d'adresse

- Un système invité croit s'exécuter seul sur une machine



- Plusieurs invités peuvent utiliser les mêmes adresses physiques
- L'hyperviseur doit éviter les collisions d'adresses physiques

Placement NUMA des données en mode virtualisé



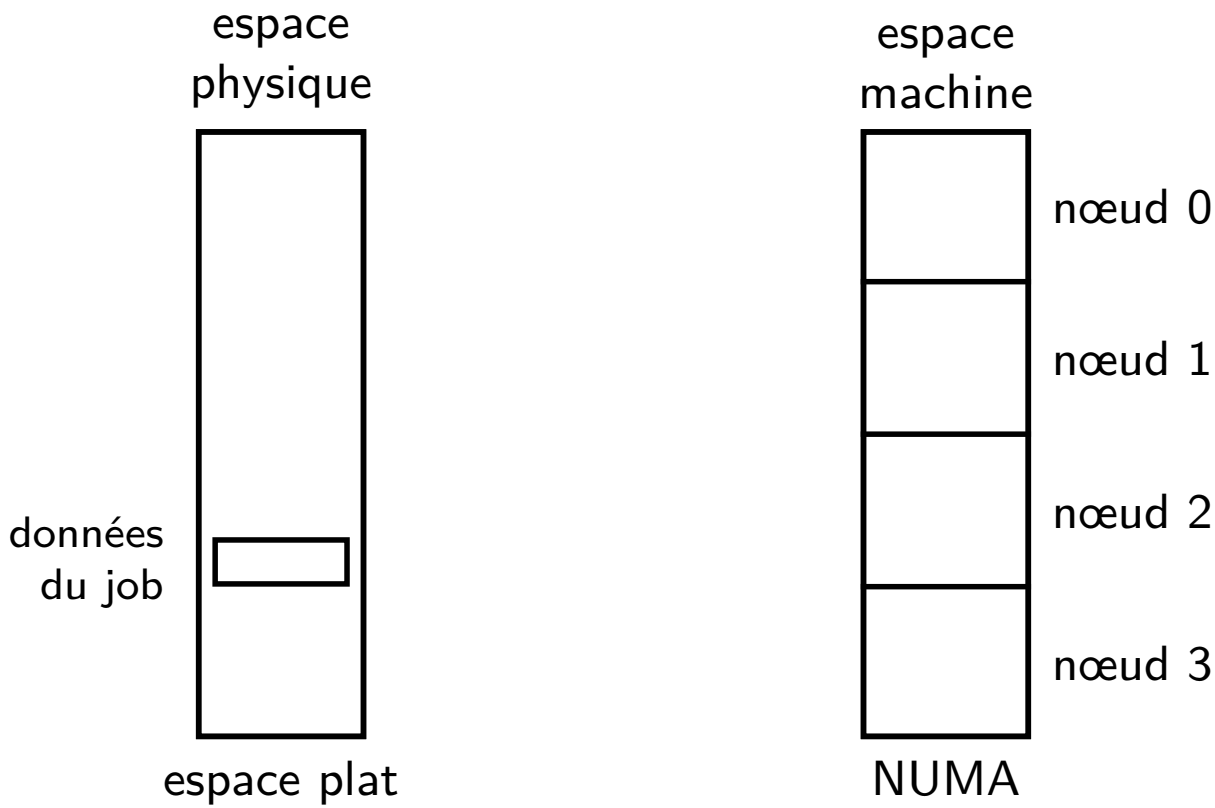
- Les processeurs modernes effectuent deux traductions d'adresses
  - La première traduction est contrôlée exclusivement par le système invité
  - La seconde traduction est contrôlée exclusivement par l'hyperviseur
- Le placement des données est déterminé conjointement par le système invité et l'hyperviseur

Deux approches existantes face au problème du placement

- Le problème du placement NUMA virtualisé est connu

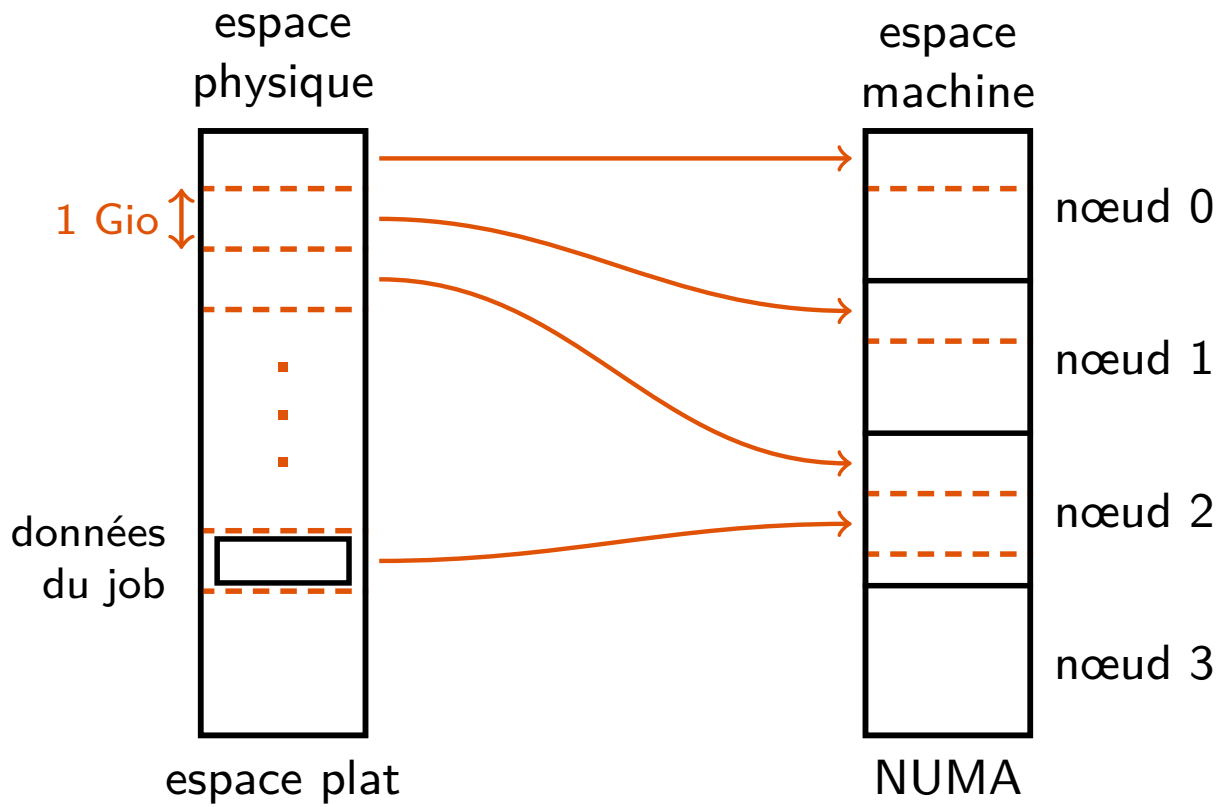
- Deux approches déjà utilisées :
  - L'hyperviseur décide seul car c'est le plus privilégié
  - L'invité décide seul car c'est le plus proche de l'application

L'hyperviseur décide seul car c'est le plus privilégié



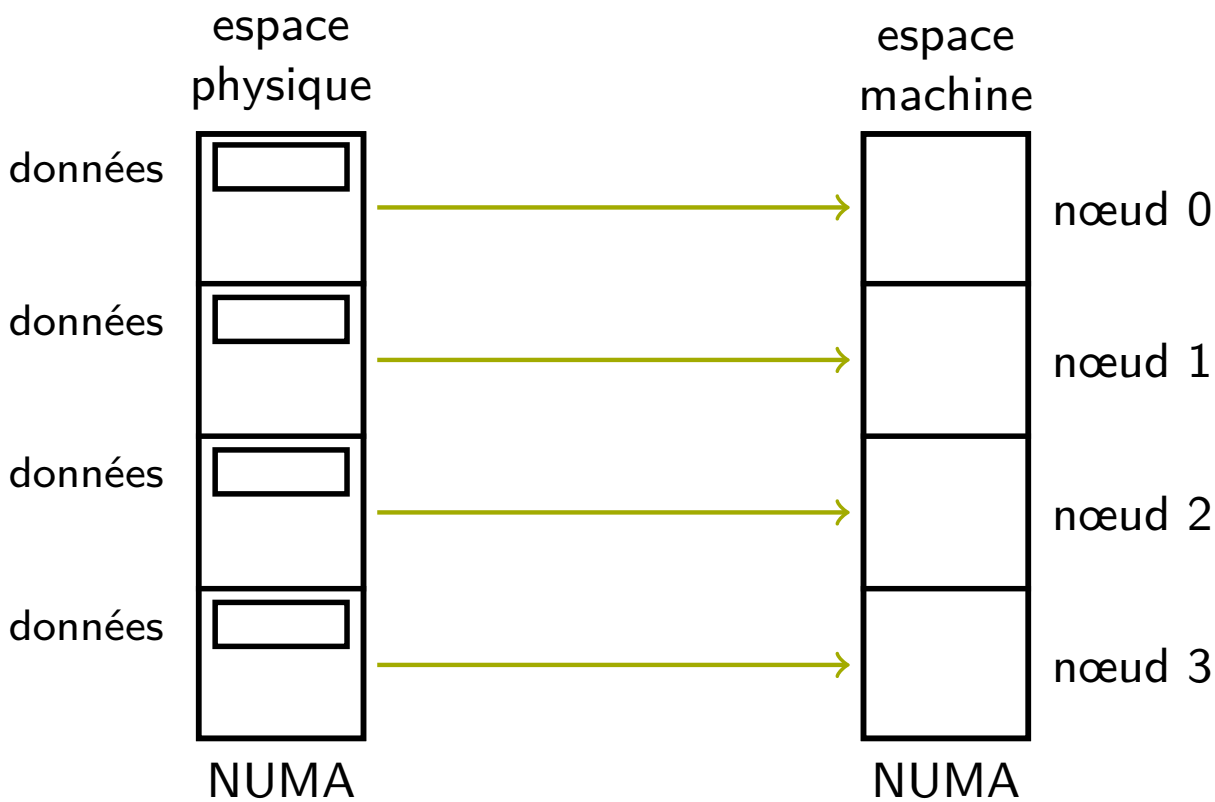
- Utilisée par l'hyperviseur Xen
- Expose une topologie plate au système invité
- Le système invité alloue son espace physique de façon continue

L'hyperviseur décide seul car c'est le plus privilégié



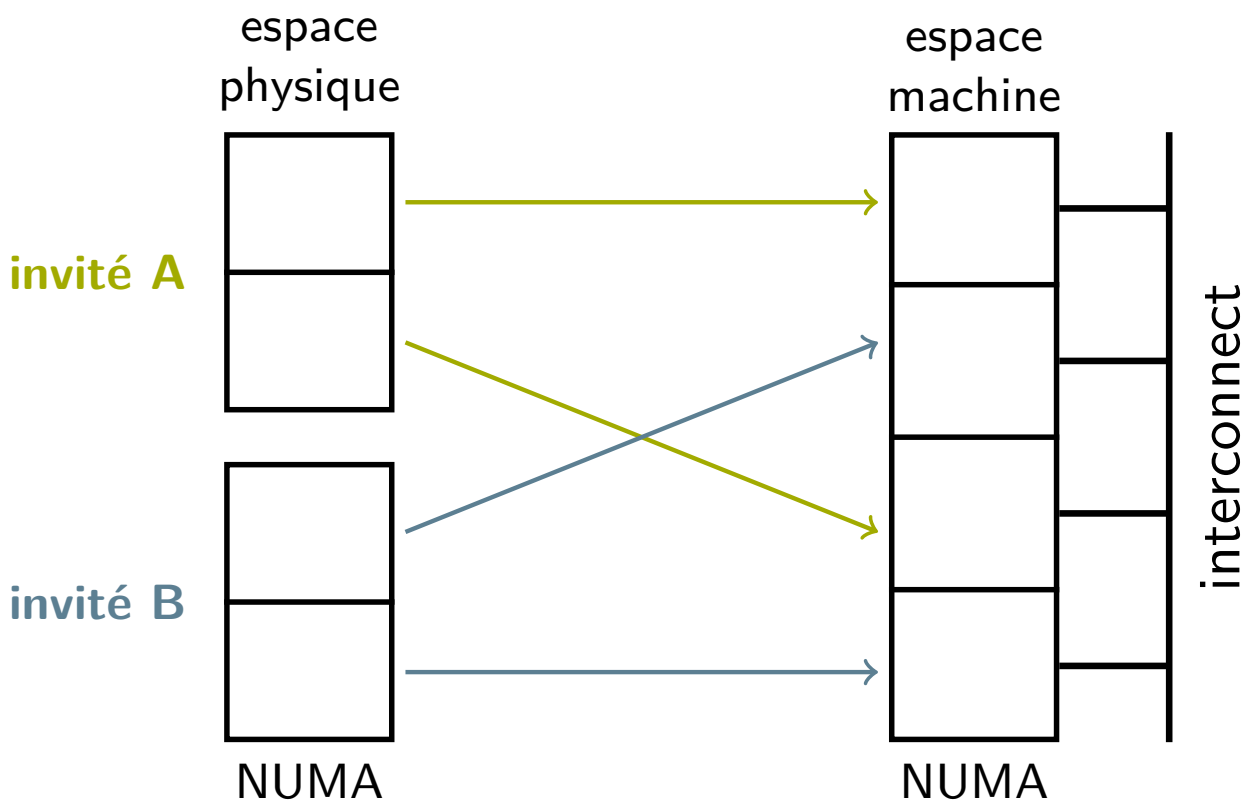
- Utilisée par l'hyperviseur Xen
- Expose une topologie plate au système invité
- Le système invité alloue son espace physique de façon continue
- Xen utilise une unique stratégie de placement NUMA → round-1G
  - Provoque une congestion d'un nœud NUMA

L'invité décide seul car c'est le plus proche de l'application



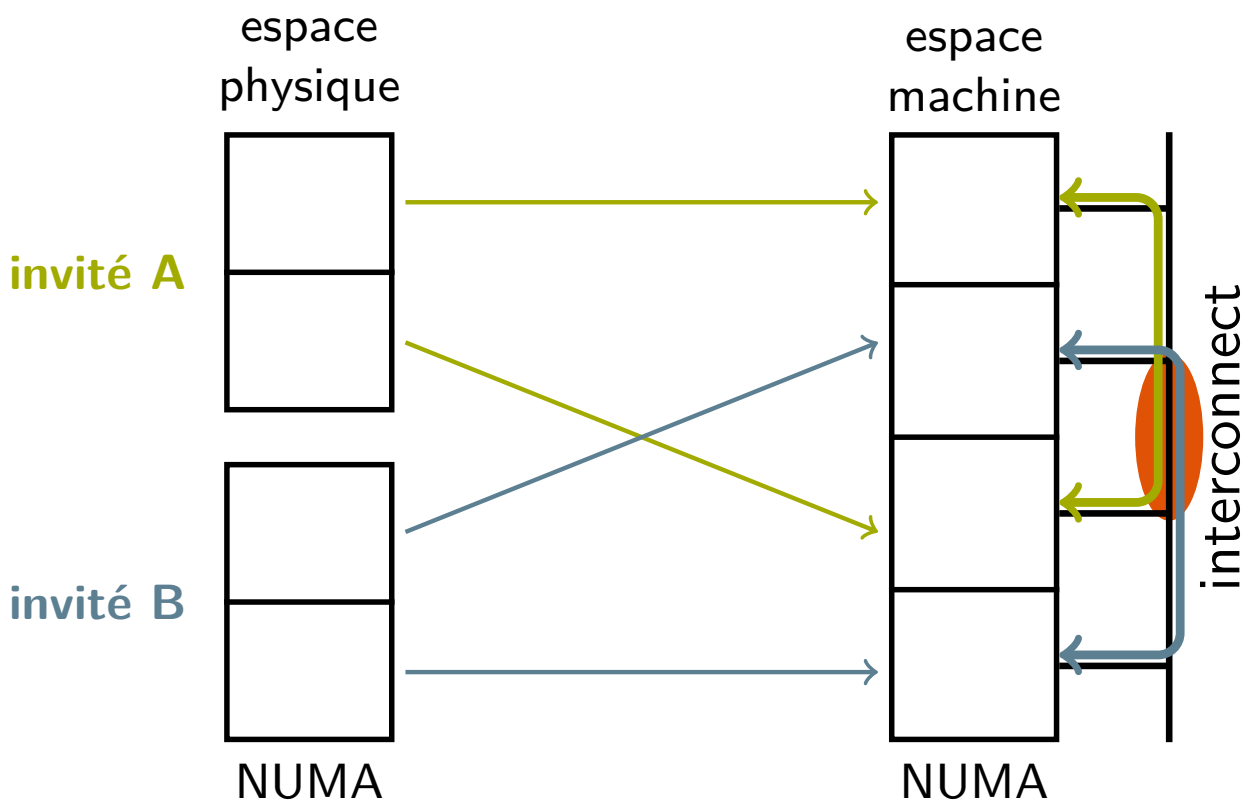
- Utilisée dans Amazon EC2
- Expose une topologie NUMA au système invité
- Le système invité applique les politiques NUMA classiques

L'invité décide seul car c'est le plus proche de l'application



- Utilisée dans Amazon EC2
- Expose une topologie NUMA au système invité
- Le système invité applique les politiques NUMA classiques
- Amazon EC2 garantit qu'il n'y a qu'un système invité par machine
  - Deux systèmes invités peuvent se géner mutuellement

L'invité décide seul car c'est le plus proche de l'application



- Utilisée dans Amazon EC2
- Expose une topologie NUMA au système invité
- Le système invité applique les politiques NUMA classiques

- Amazon EC2 garantit qu'il n'y a qu'un système invité par machine
  - Deux systèmes invités peuvent se géner mutuellement

Contribution

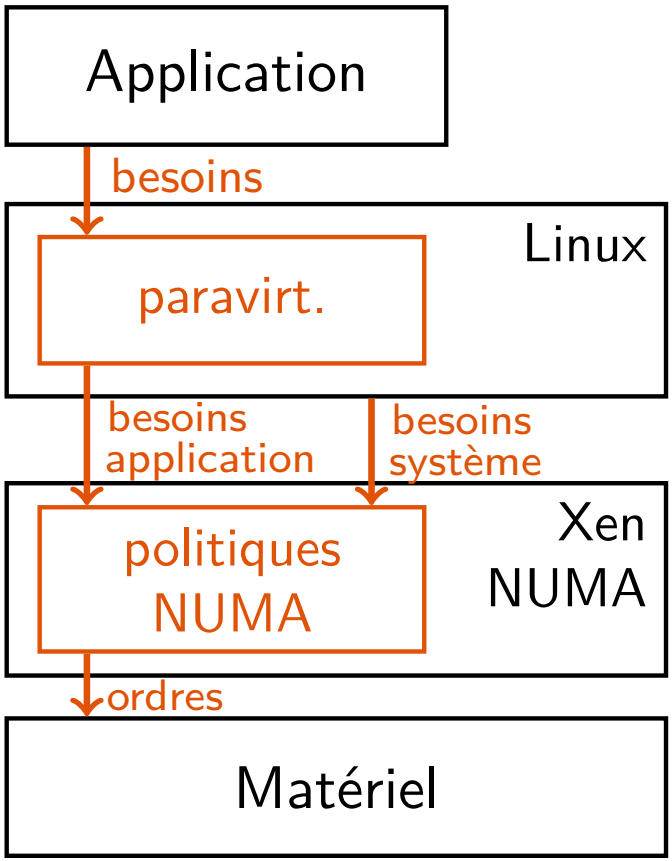
	Prise en compte des besoins de l'application	Prise en compte des contraintes globales
L'hyperviseur décide du placement	**non**	**oui**
Les invités décident du placement	**oui**	**non**

Contribution de la Thèse

L'hyperviseur décide les invités indiquent leurs besoins	**oui**	**oui**

- Centraliser la gestion NUMA dans l'hyperviseur
  - Prise en compte des contraintes globales
- Permettre au système invité d'indiquer les besoins de l'application
  - Prise en compte des besoins de l'application

Contribution : aperçu



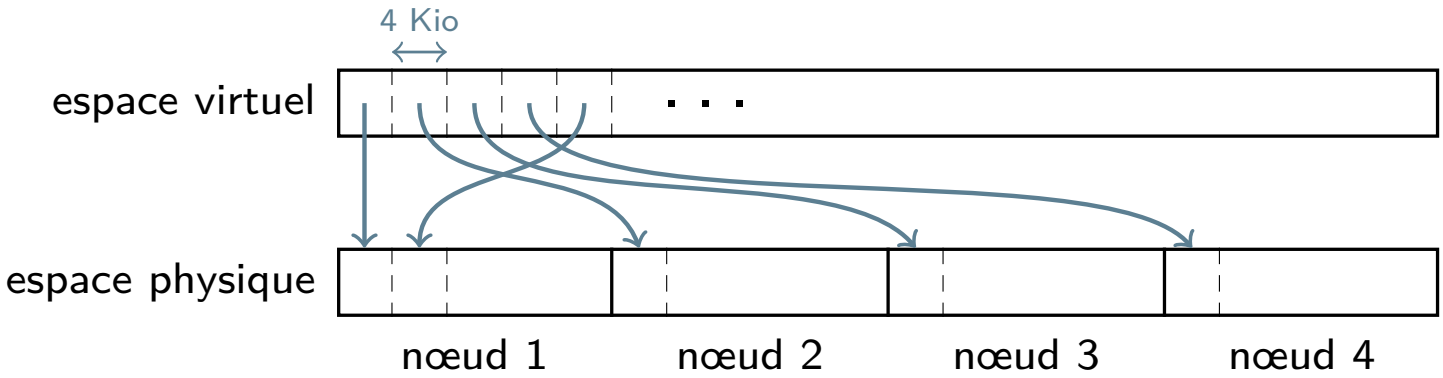
- On met en œuvre trois politiques NUMA dans l'hyperviseur Xen
- Ces politiques observent le système invité comme une boîte noire
- Le système invité transmet les besoins de l'applications à l'hyperviseur

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++
Carrefour	+	+	+

Plan

Round-4k : éviter la congestion mémoire d'un nœud

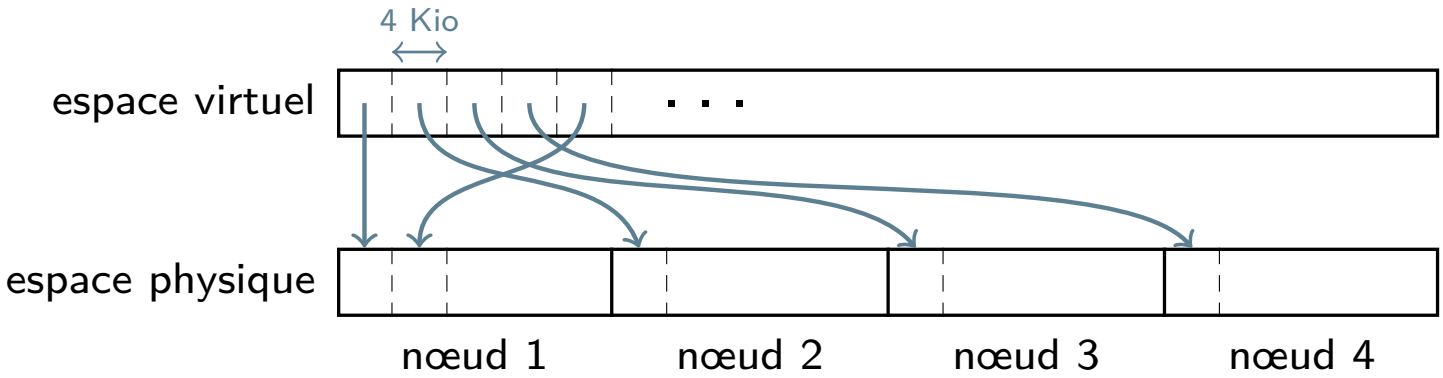
- Facteur de performance : la saturation des contrôleurs mémoire
- Stratégie round-4k : répartir l'espace virtuel sur les différents nœuds

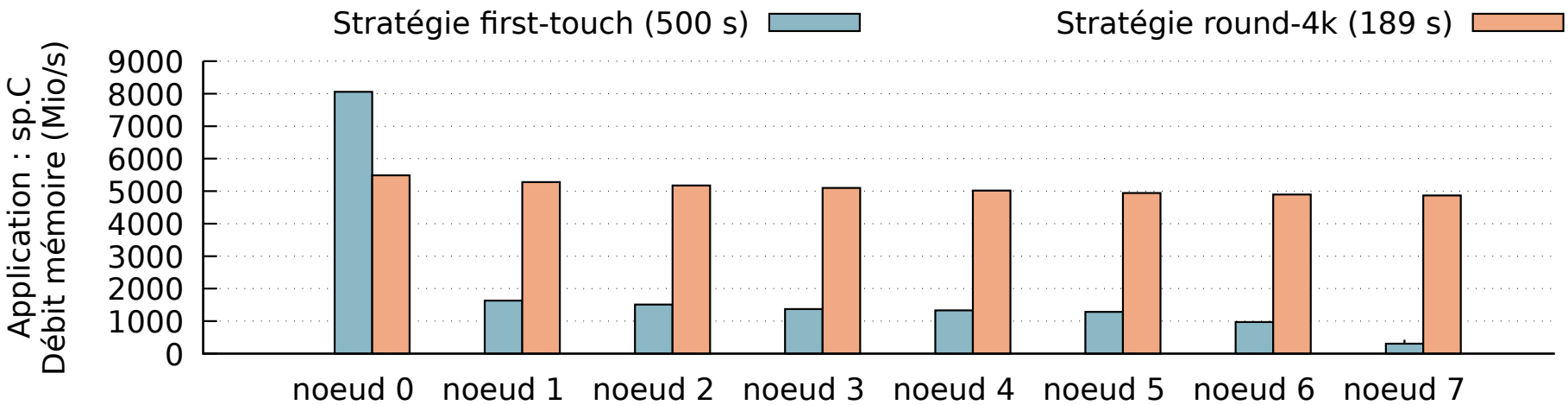


	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−

Round-4k : éviter la congestion mémoire d'un nœud

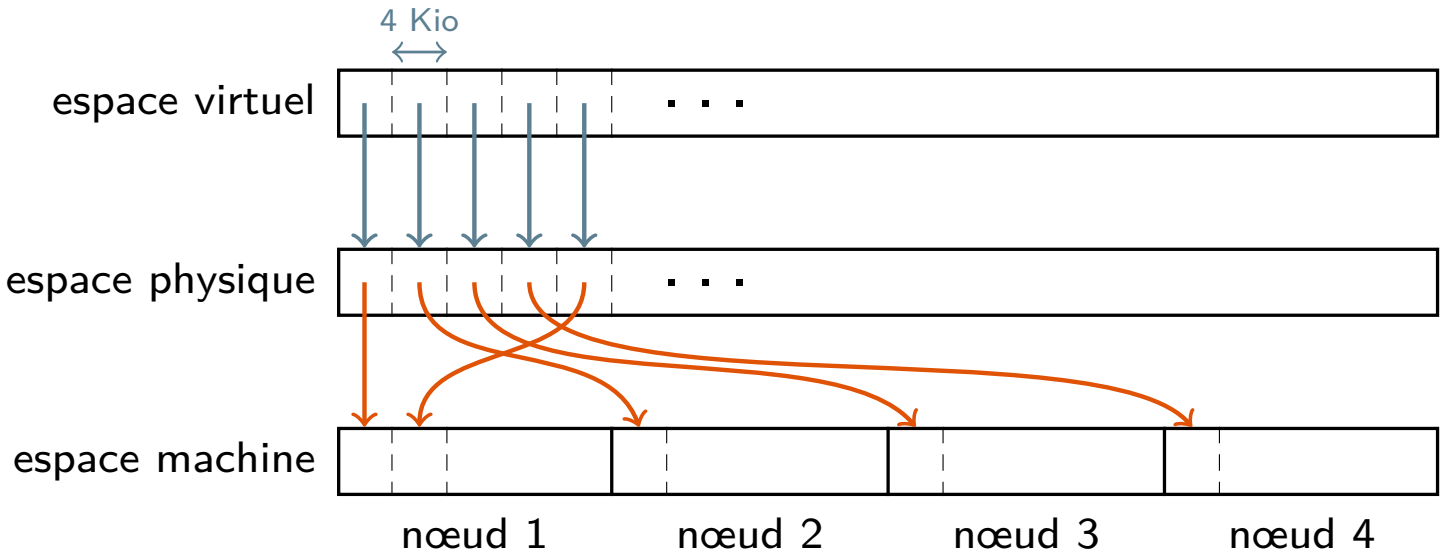
- Facteur de performance : la saturation des contrôleurs mémoire
- Stratégie round-4k : répartir l'espace virtuel sur les différents nœuds

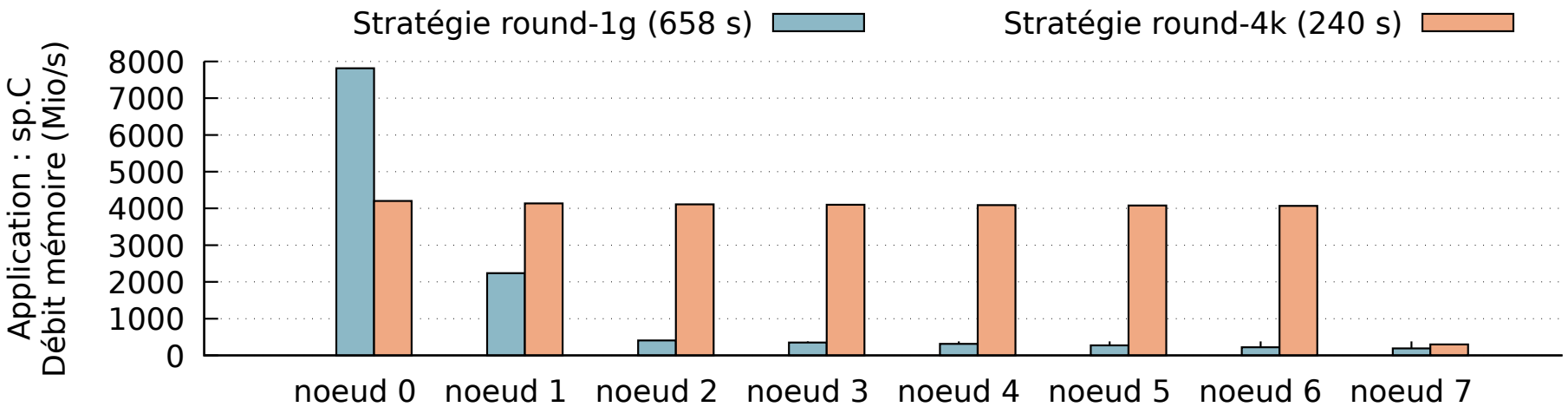




Round-4k sous Xen : répartition de l'espace physique

- Mise en œuvre triviale dans un hyperviseur
- Répartir l'espace physique (plat) sur les différents nœuds



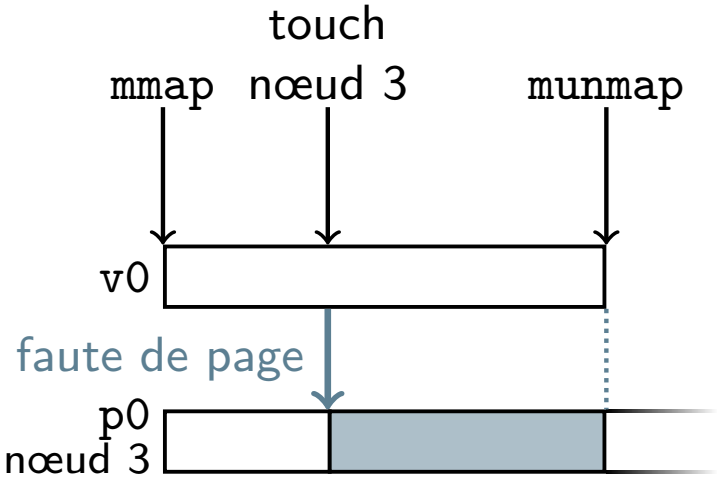


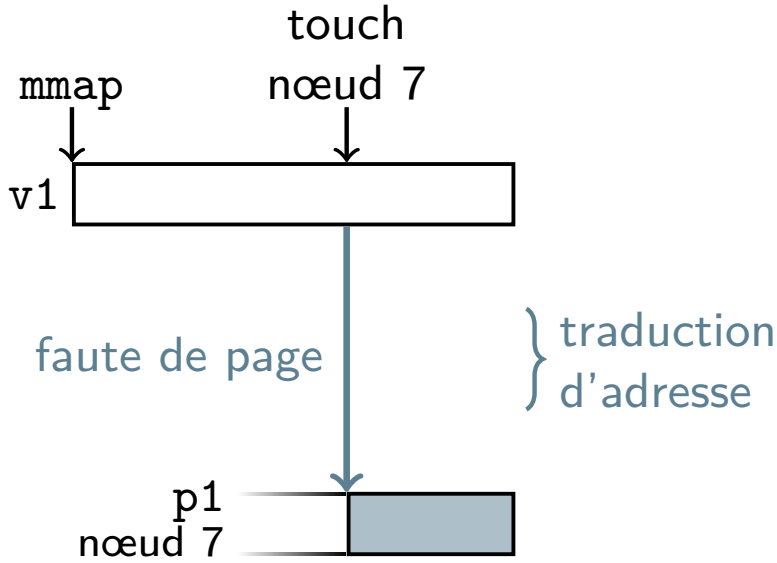
First-touch : maximiser la localité d'accès

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

- Stratégie First-touch : allouer la mémoire sur le nœud qui l'utilise
- Heuristique : un thread utilise généralement la mémoire qu'il alloue
  - Exemple : la pile d'un thread
- Stratégie : allouer la mémoire sur le nœud du thread qui alloue
- Stratégie par défault sous Linux

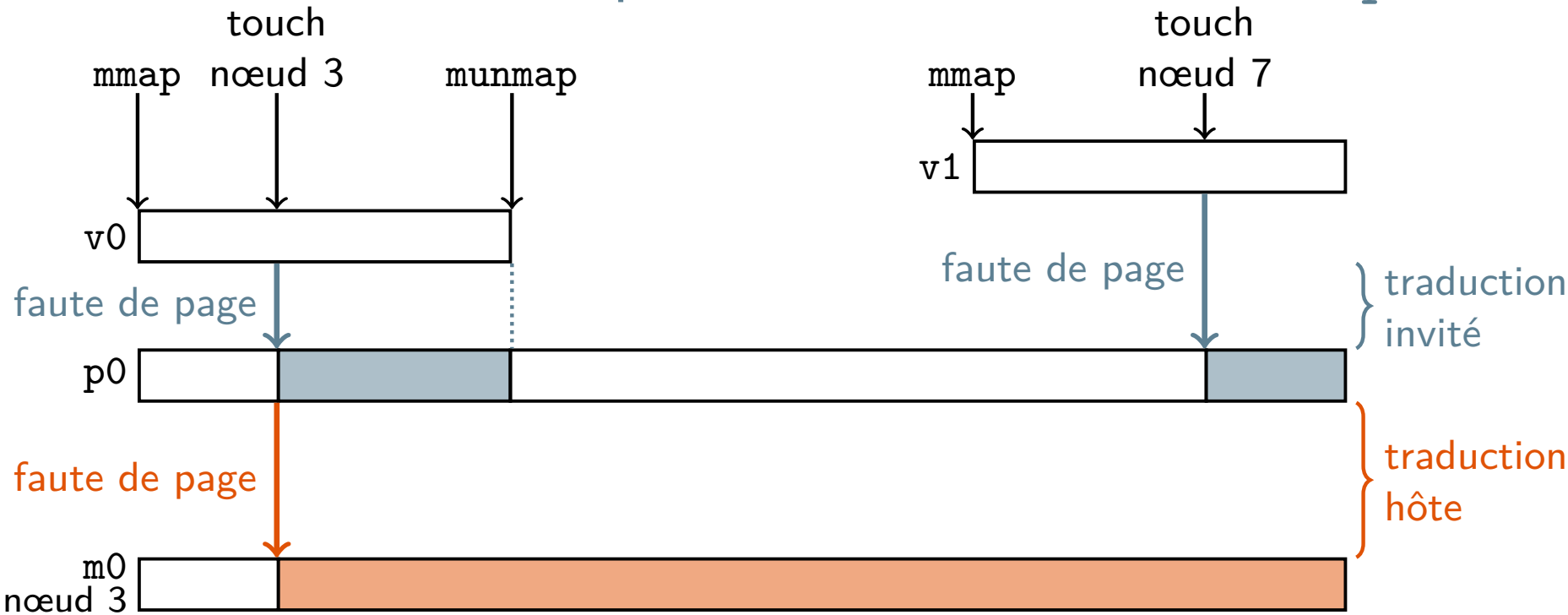
First-touch : fonctionnement sous Linux





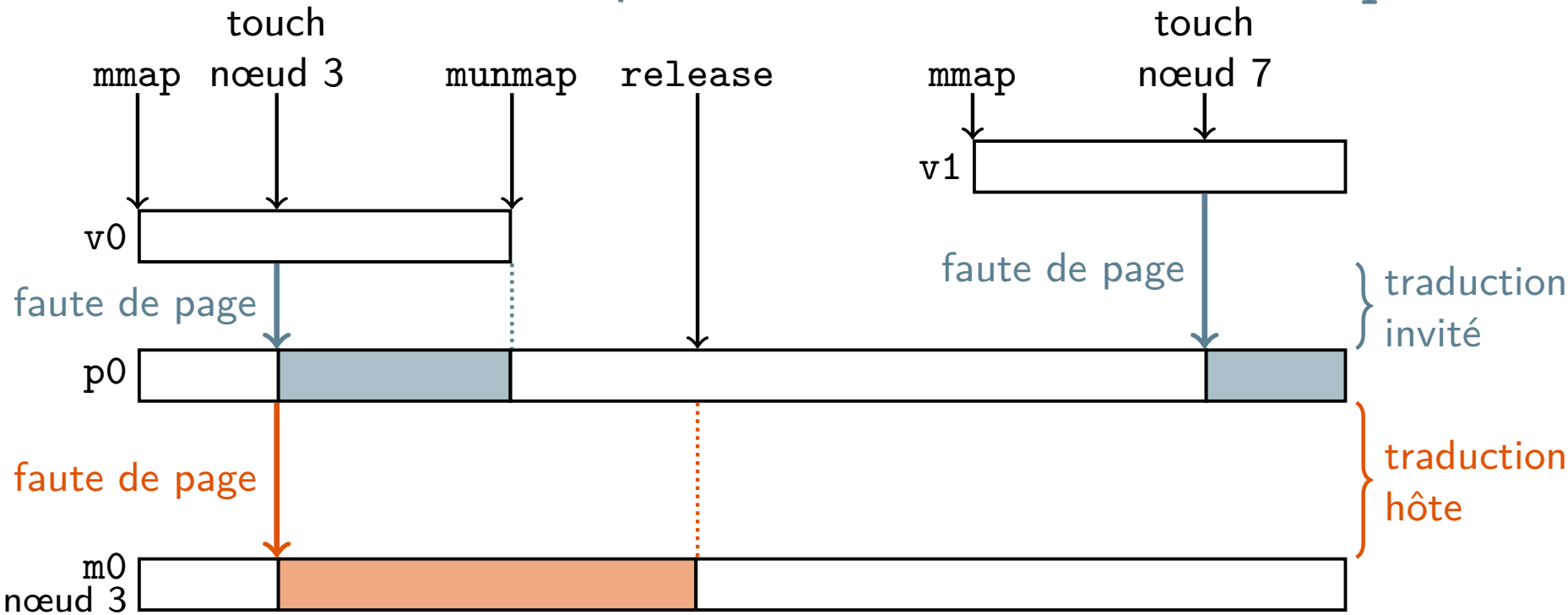
- L'allocation de mémoire virtuelle retourne des adresses non mappées
- L'utilisation d'une adresse virtuelle non mappée provoque une faute
  - Allocation de mémoire physique sur le nœud fautif
- La libération de mémoire virtuelle provoque la libération de la mémoire physique correspondante

First-touch sous Xen : paravirtualisation du `munmap()`



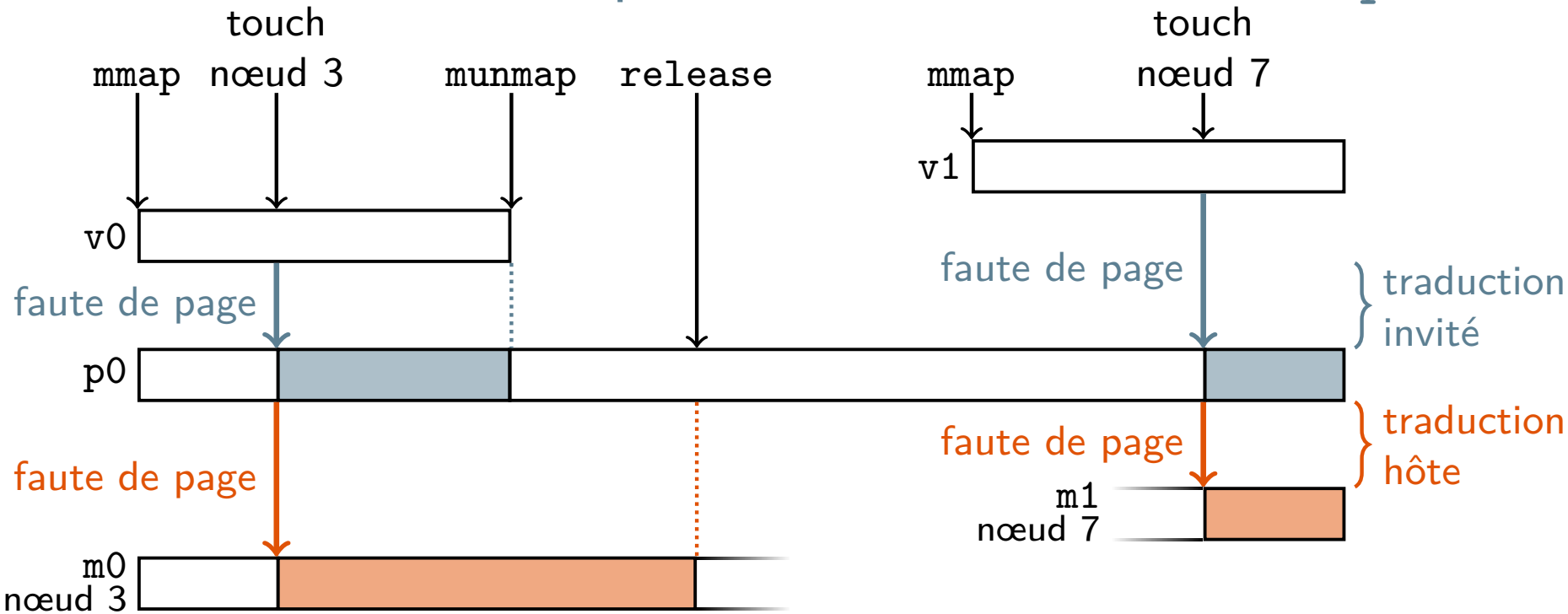
- L'utilisation d'une adresse physique non mappée provoque une faute traitée par l'hyperviseur
- La libération de mémoire virtuelle est invisible pour l'hyperviseur
- L'hyperviseur n'est plus averti des *first-touch* suivants

First-touch sous Xen : paravirtualisation du `munmap()`



- L'utilisation d'une adresse physique non mappée provoque une faute traitée par l'hyperviseur
- La libération de mémoire virtuelle est invisible pour l'hyperviseur
- L'hyperviseur n'est plus averti des *first-touch* suivants
- L'invité signale la libération d'adresses physiques par un hypercall

First-touch sous Xen : paravirtualisation du `munmap()`



- L'utilisation d'une adresse physique non mappée provoque une faute traitée par l'hyperviseur
- La libération de mémoire virtuelle est invisible pour l'hyperviseur
- L'hyperviseur n'est plus averti des *first-touch* suivants

- L'invité signale la libération d'adresses physiques par un hypercall
- L'utilisation de l'adresse physique libérée provoque une nouvelle faute

Carrefour : corriger les défauts des autres stratégies

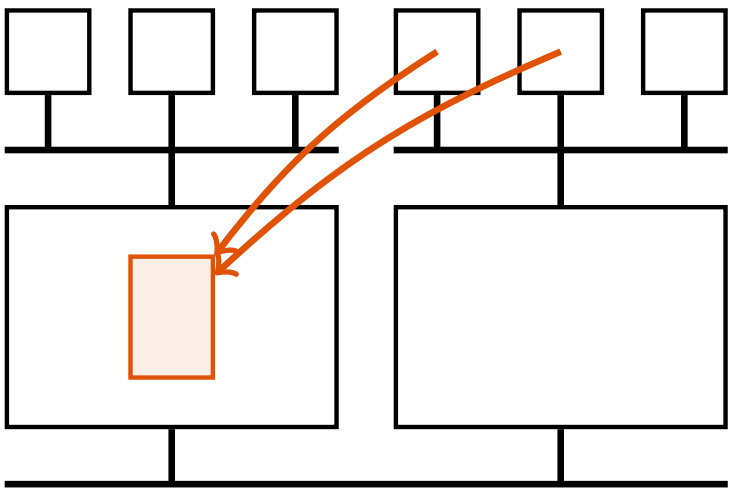
	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

- Stratégie Carrefour → corriger les défauts des autres stratégies

Carrefour : corriger les défauts des autres stratégies

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

- Stratégie Carrefour → corriger les défauts des autres stratégies

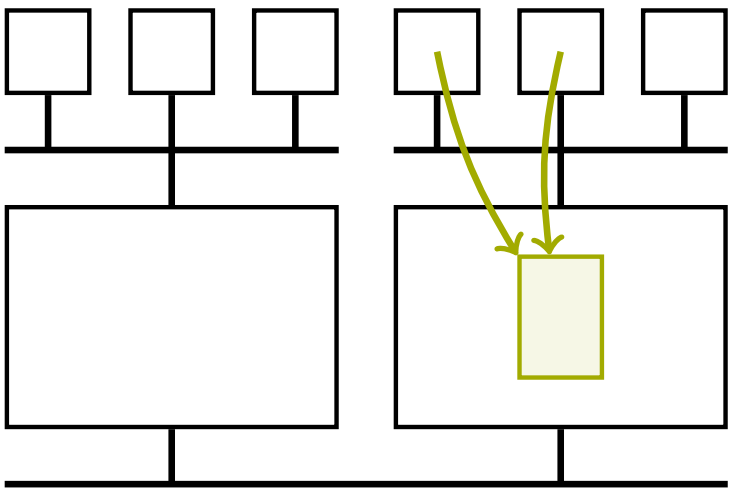


- Faible localité

Carrefour : corriger les défauts des autres stratégies

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

- Stratégie Carrefour → corriger les défauts des autres stratégies

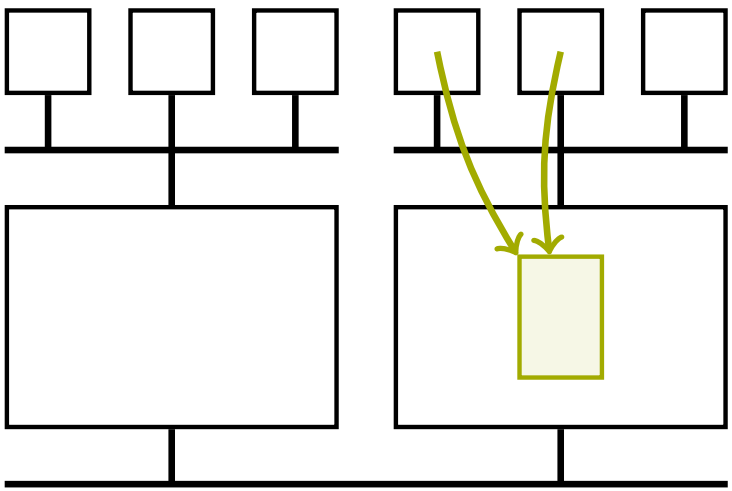


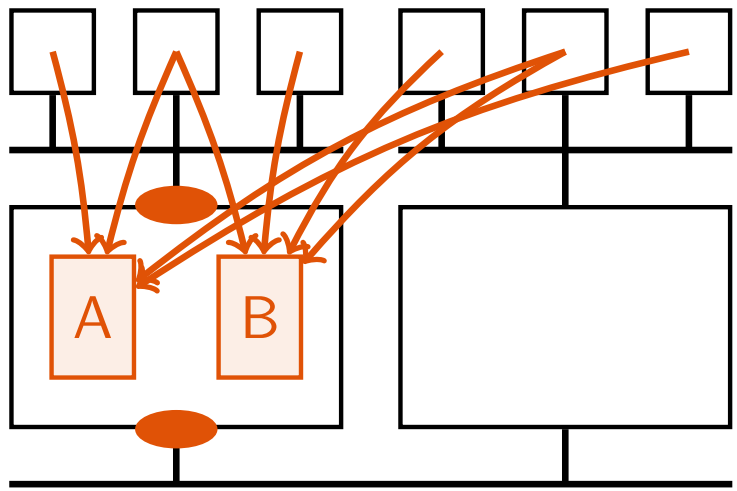
- Faible localité → migration

Carrefour : corriger les défauts des autres stratégies

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

- Stratégie Carrefour → corriger les défauts des autres stratégies



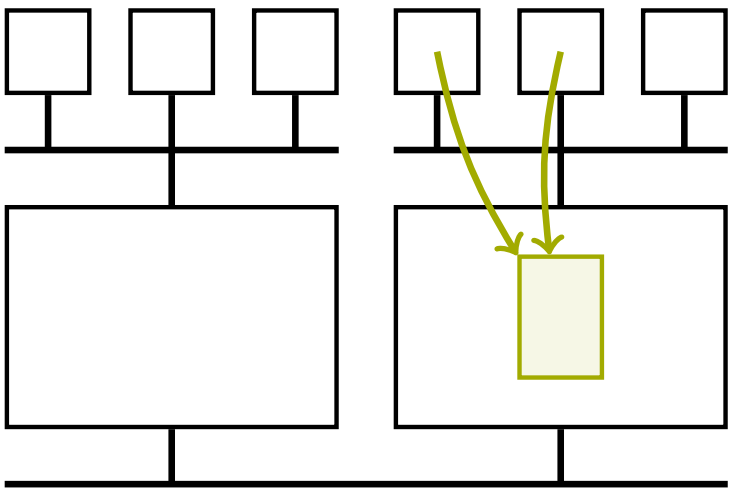


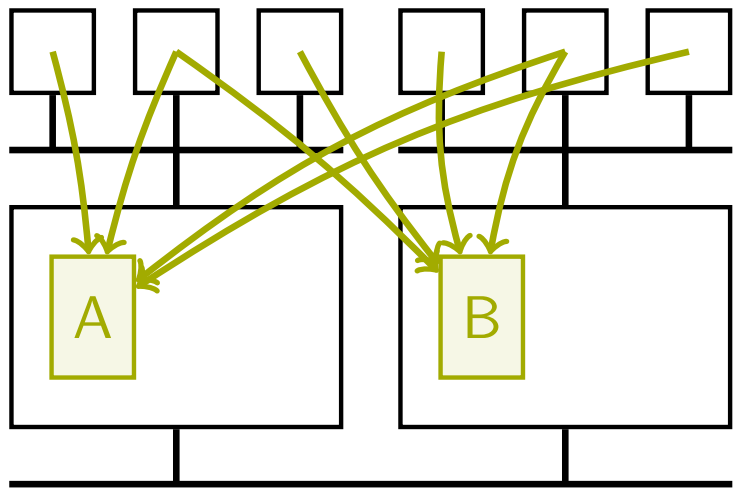
- Faible localité → migration
- Saturation contrôleur

Carrefour : corriger les défauts des autres stratégies

	saturation contrôleurs	saturation interconnect	localité d'accès
Round-4k	+++	−−−	−−−
First-touch	−−−	+++	+++

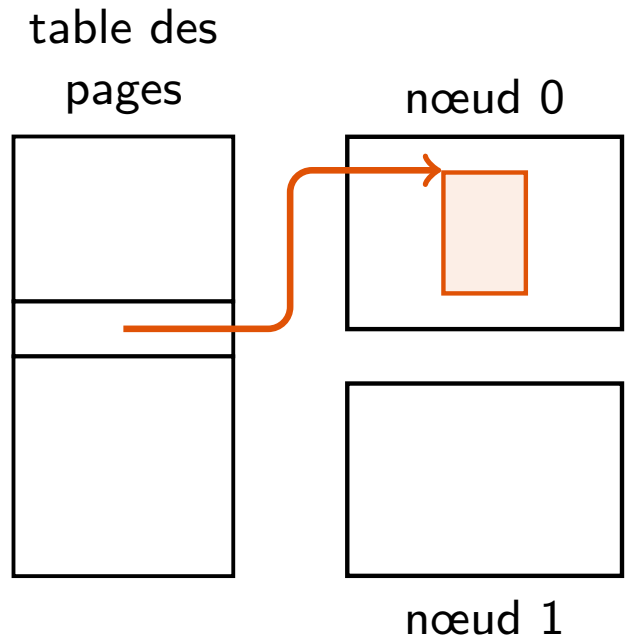
- Stratégie Carrefour → corriger les défauts des autres stratégies



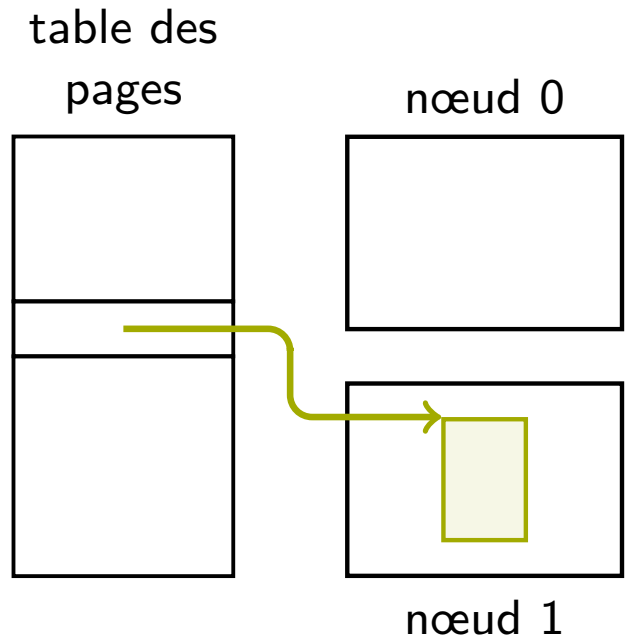


- Faible localité → migration
- Saturation contrôleur → répartition

Carrefour sous Linux : carte des accès mémoire

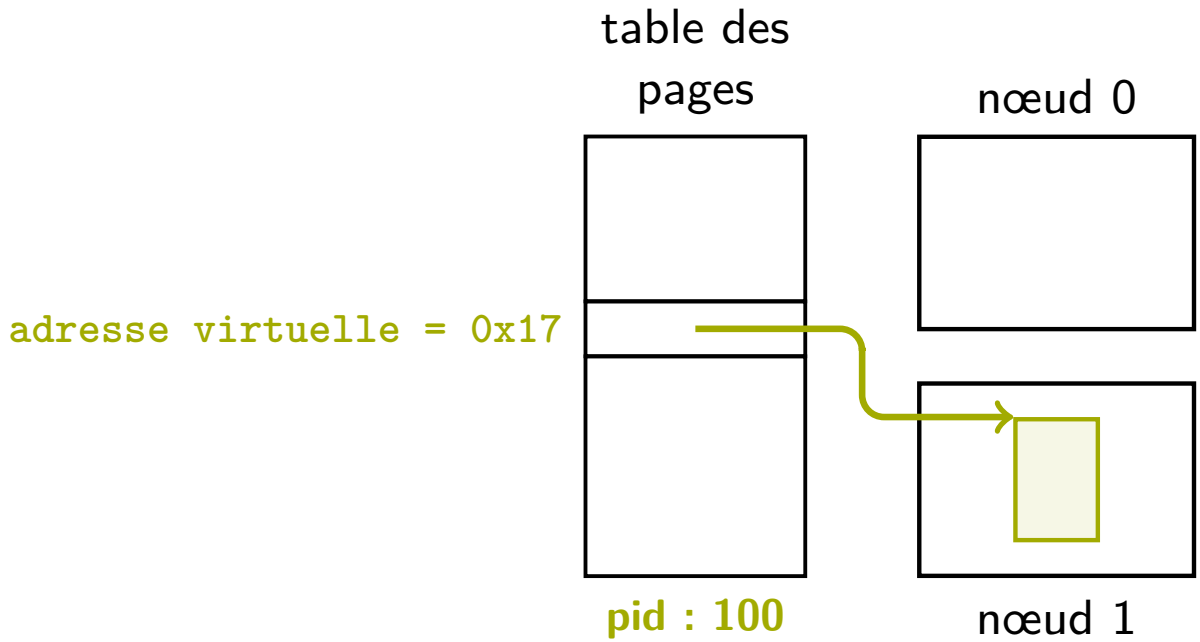


Carrefour sous Linux : carte des accès mémoire



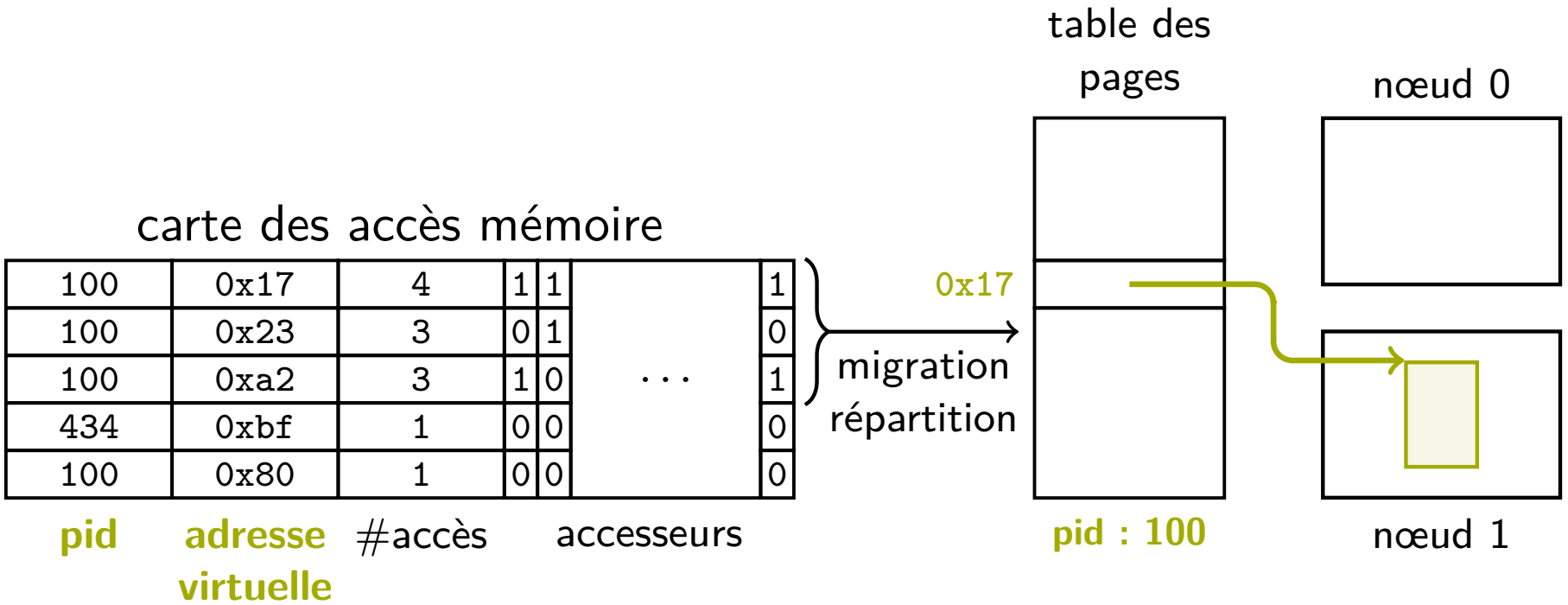
- La migration de page se fait en modifiant la table des pages

Carrefour sous Linux : carte des accès mémoire



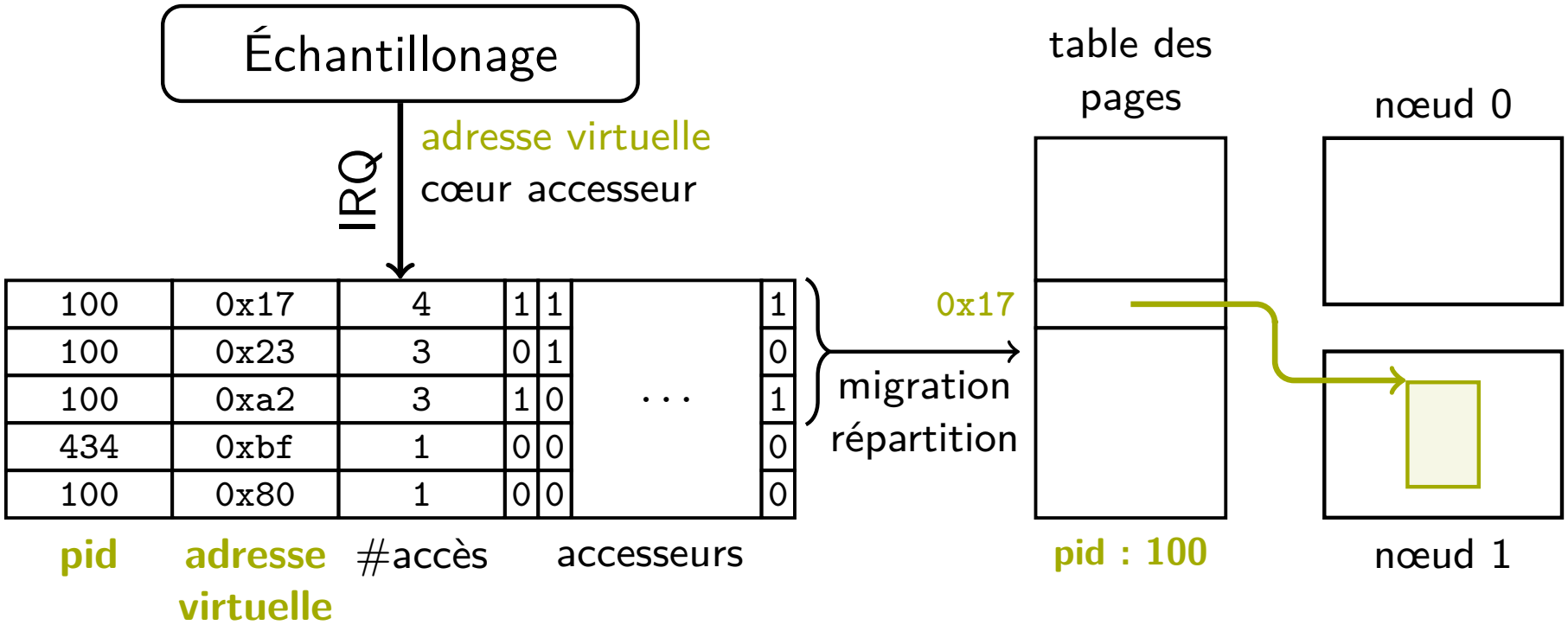
- La migration de page se fait en modifiant la table des pages
  - Chaque table des pages est associée à un processus
  - Les tables des pages sont indexées par adresse virtuelle

Carrefour sous Linux : carte des accès mémoire



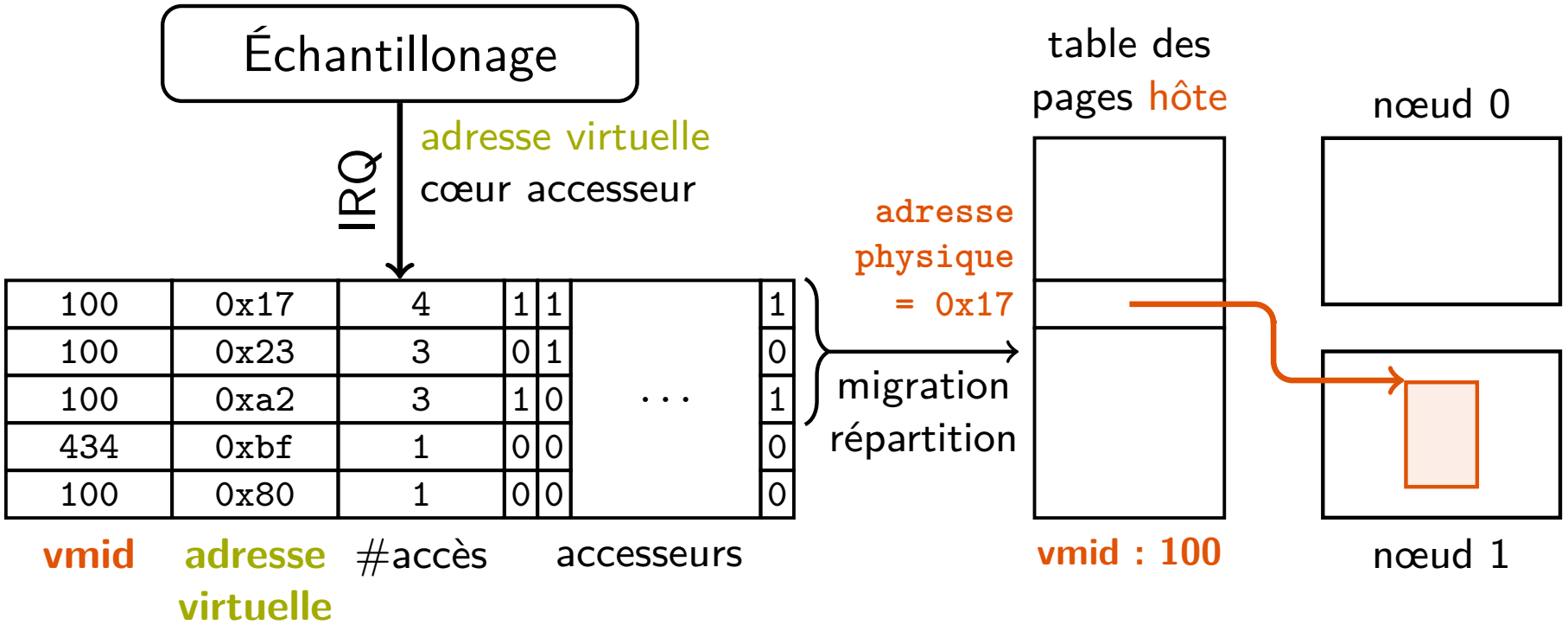
- La migration de page se fait en modifiant la table des pages
  - Chaque table des pages est associée à un processus
  - Les tables des pages sont indexées par adresse virtuelle
- Carrefour utilise une carte des accès pour savoir quelle page migrer

Carrefour sous Linux : carte des accès mémoire



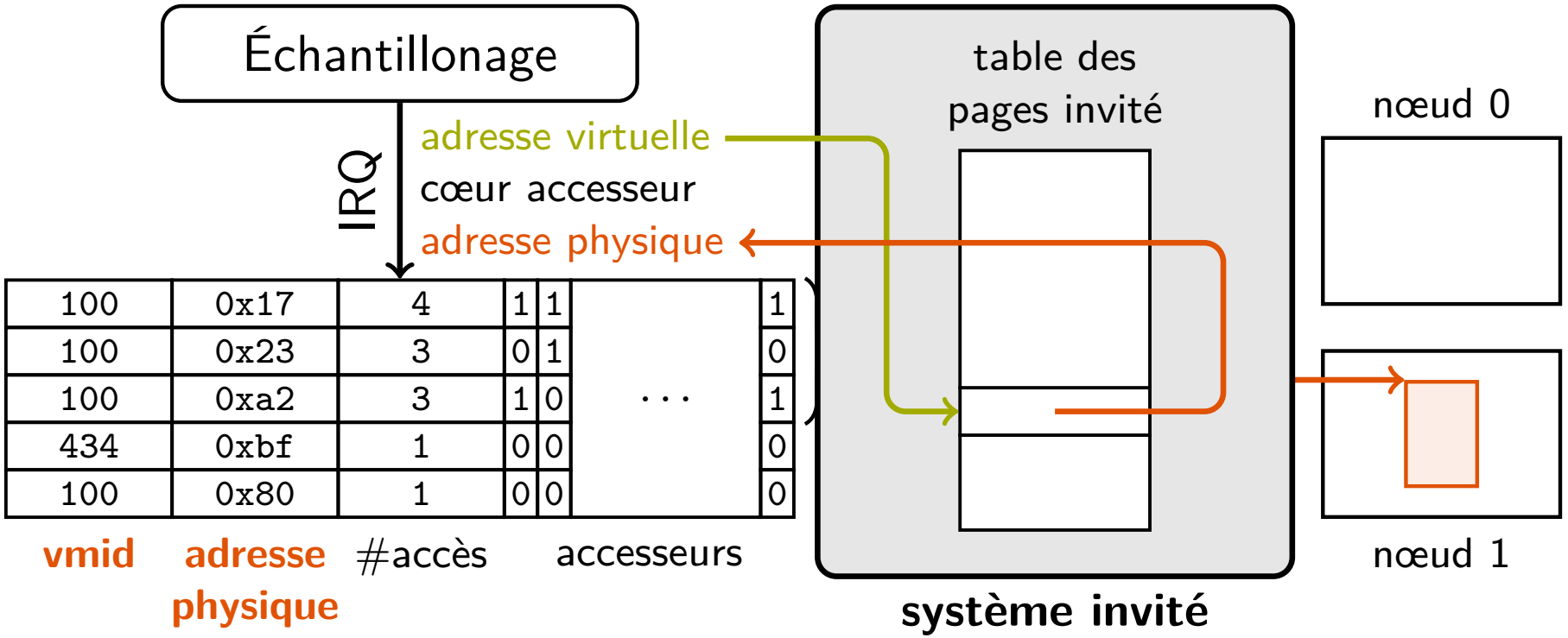
- La migration de page se fait en modifiant la table des pages
  - Chaque table des pages est associée à un processus
  - Les tables des pages sont indexées par adresse virtuelle
- Carrefour utilise une carte des accès pour savoir quelle page migrer
- Carrefour construit cette carte par échantillonage des accès mémoire

Carrefour sous Xen : traduction intermédiaire d'adresses



- L'hyperviseur migre les pages via la table des pages hôte
  - Chaque table des pages hôte est associée à un système invité
  - Les tables des pages hôte sont indexées par adresse physique
- Les compteurs d'échantillonage n'indiquent que des adresses virtuelles

Carrefour sous Xen : traduction intermédiaire d'adresses



100	0x17	4	1	1	...	1
100	0x23	3	0	1		0
100	0xa2	3	1	0		1
434	0xbf	1	0	0		0
100	0x80	1	0	0		0

- L'hyperviseur migre les pages via la table des pages hôte
  - Chaque table des pages hôte est associée à un système invité
  - Les tables des pages hôte sont indexées par adresse physique
- Les compteurs d'échantillonage n'indiquent que des adresses virtuelles
- Carrefour-Xen traduit les adresses virtuelles pendant l'IRQ

Évaluation : configuration logicielle et matérielle

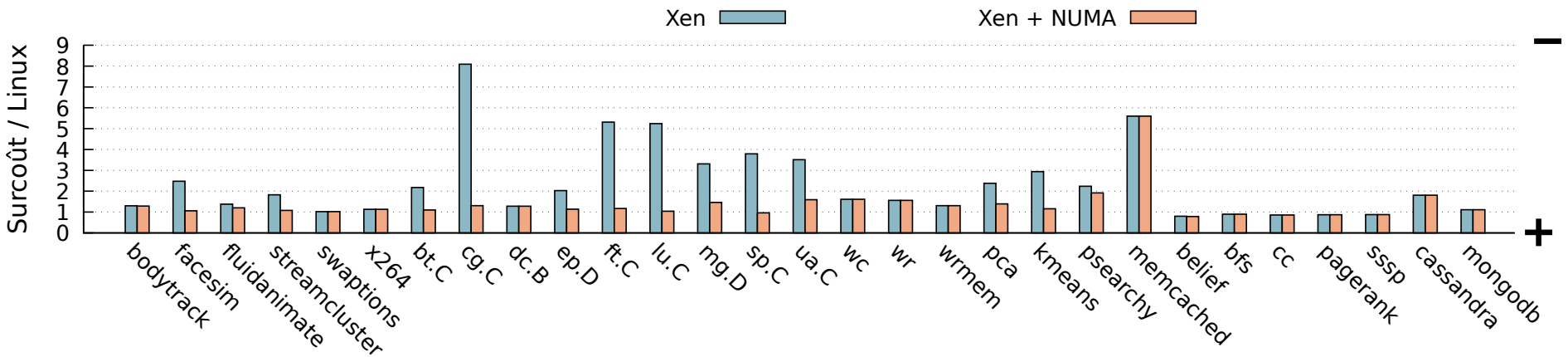
- Mise en œuvre sur Linux 3.9 / Xen 4.5
- Mesure du temps de complétion sur 29 applications de 5 benchmarks

Benchmarks	Type d'applications
Parsec	Traitement de graphes / traitement d'image
NPB	Calcul scientifique
Mosbench	MapReduce multicœur / recherche de données
X-Stream	Traitement de graphes sur disque
YCSB	Bases de données NoSQL

- Exécution sur une machine AMD de 8 nœuds
  - Par nœud : 6 cœurs (2.2 GHz) / 16 Gio de mémoire
  - Bande passante contrôleur mémoire : 13 Gio/s
  - Bande passante interconnect : 6 Gio/s asymétriques

Évaluation : surcoût de la virtualisation

- Comparaison des temps de complétion Xen / Linux
  - Linux utilise la meilleure politique mémoire
  - Xen utilise la politique round-1g
  - Xen + NUMA utilise la meilleure politique mémoire

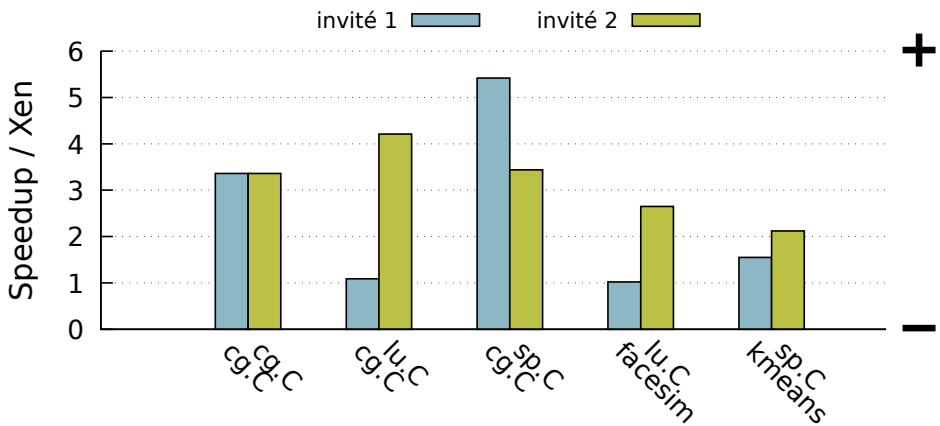


- Gain de performance jusqu'à 700% avec Xen + NUMA
- Surcoût inférieur à 50% : 12/29 → 23/29 applications

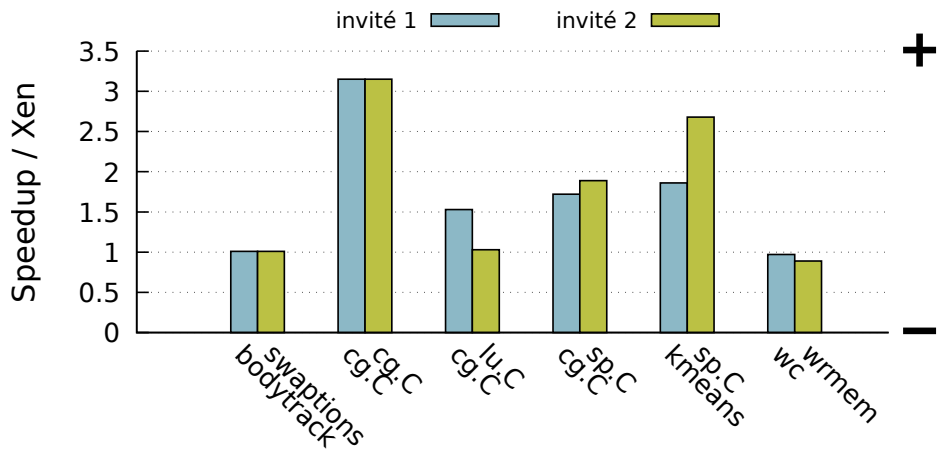
Évaluation : machines virtuelles multiples

- Comparaison des temps de complétion Xen + NUMA / Xen
- Deux systèmes invités exécutés simultanément

- exécutés sur des nœuds distincts



- exécutés sur les mêmes nœuds



- Pas de dégradation des performances
- Jusqu'à 400% de speedup par rapport à Xen

Conclusion

- Le placement NUMA est un facteur de performance majeur pour les machines virtuelles multinœud
- Une coopération entre les systèmes invités et l'hyperviseur suffit à diminuer le surcoût de la virtualisation à un niveau tolérable
- Il est possible d'utiliser dans un hyperviseur les politiques mises en œuvre dans les systèmes d'exploitation

Perspectives

- Driver NUMA $\rightarrow$ un hyperviseur dédié à la gestion du placement mémoire d'un seul système invité
- Approche inverse $\rightarrow$ mise en œuvre des politiques dans les systèmes invités / découverte de la topologie NUMA par la mesure

Perspectives

- Driver NUMA $\rightarrow$ un hyperviseur dédié à la gestion du placement mémoire d'un seul système invité
- Approche inverse $\rightarrow$ mise en œuvre des politiques dans les systèmes invités / découverte de la topologie NUMA par la mesure
- Merci pour votre attention