Algorithmique distribuée d'exclusion mutuelle : vers une gestion efficace des ressources

Jonathan Lejeune

LIP6-UPMC/CNRS, Inria

19 septembre 2014

Directeur : Pierre Sens
Encadrants : Luciana Arantes et Julien Sopena

Introduction

Les systèmes à grande échelle se caractérisent par :

- de nombreuses ressources partagées
- de nombreux d'utilisateurs
- des accès avec différentes contraintes applicatives

  $\Longrightarrow$ **mécanismes de réservation de ressources indispensables**

Exclusion mutuelle :

Assurer qu'à tout un instant, seul un processus puisse exécuter une partie d'un code concurrent, appelée section critique.

Propriétés :

- **Sûreté :** au plus un processus en section critique
- **Vivacité :** pour un temps de section critique fini, chaque demande aboutira à la section critique dans un temps fini

Contributions

Limites des mécanismes de réservation :

- approche centralisée :
  - passage à l'échelle difficile
- approche distribuée :
  - faible prise en compte de l'hétérogénéité des requêtes (QoS, priorités, ...) $\Rightarrow$ non respect des contraintes applicatives
  - gestion peu efficace des accès sur un ensemble de ressources

$\Longrightarrow$ **mécanismes inadaptés aux systèmes actuels**

Contributions :

- **Contribution 1 : Exclusion mutuelle à priorité**
- **Contribution 2 : Exclusion mutuelle à contraintes temporelles**
- **Contribution 3 : Exclusion mutuelle pour plusieurs ressources hétérogènes**

Contribution 1 : Exclusion mutuelle à priorité

Motivation :
Il existe plusieurs classes de client dans les systèmes large-échelle.

Exemples : utilisateur/administrateur, clients avec différents niveaux de service requis, ...

Principe :

- Associer chaque requête à un niveau de priorité.
- Satisfaire les requêtes en respectant l'ordre des priorités (limiter les inversions).

Inversion de priorité : Accès à la section critique d'une requête de priorité initiale p alors qu'il existe une requête pendante de priorité initiale supérieure à p.

Contribution 1 : Exclusion mutuelle à priorité

Solutions existantes

- **Priorités statiques [Goscinksi90, Mueller 99, Housni-Trehel 01]** : la priorité d'une requête reste la même jusque satisfaction
  - \+ pas d'inversion de priorité
  - \- famine potentielle : violation de la vivacité
- **Priorités dynamiques [Chang 94, Kanrar-Chaki 10]** : la priorité d'une requête augmente au cours du temps.
  - \+ famine impossible
  - \- génère beaucoup d'inversion

Contribution 1 : Exclusion mutuelle à priorité

Solutions existantes

- **Priorités statiques [Goscinksi90, Mueller 99, Housni-Trehel 01]** : la priorité d'une requête reste la même jusque satisfaction
  - \+ pas d'inversion de priorité
  - \- famine potentielle : violation de la vivacité
- **Priorités dynamiques [Chang 94, Kanrar-Chaki 10]** : la priorité d'une requête augmente au cours du temps.
  - \+ famine impossible
  - \- génère beaucoup d'inversion

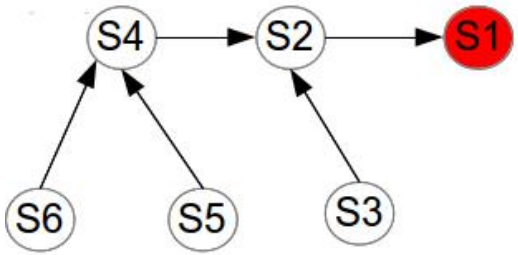
Objectifs :

1. Respecter la propriété de vivacité
2. Réduire le nombre d'inversions de priorités
3. Réduire le temps d'attente de la section critique

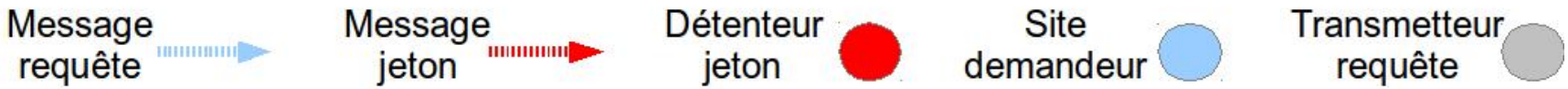
L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



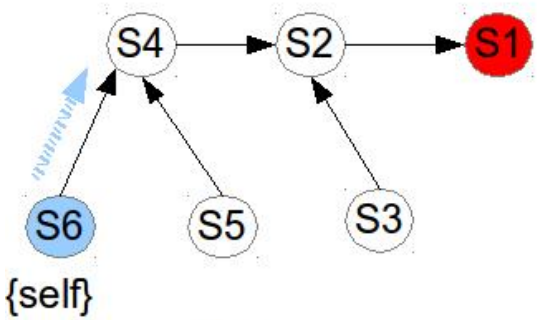
Étape 1



L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



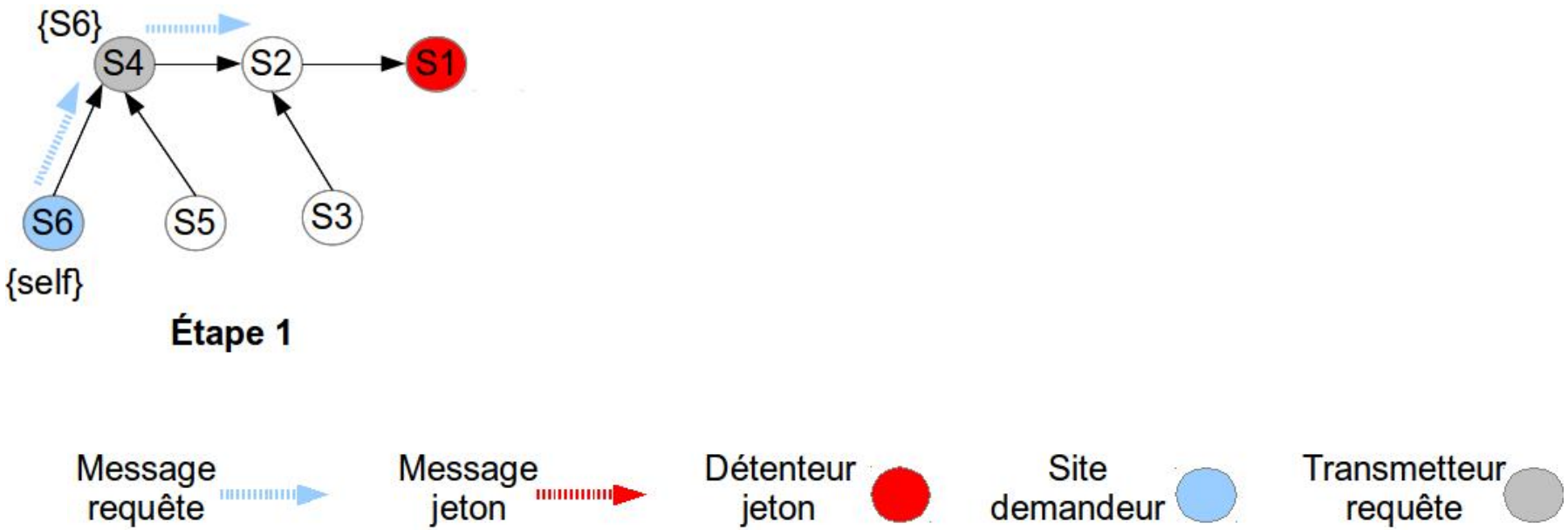
Étape 1

Message requête — Message jeton — Détenteur jeton — Site demandeur — Transmetteur requête

L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

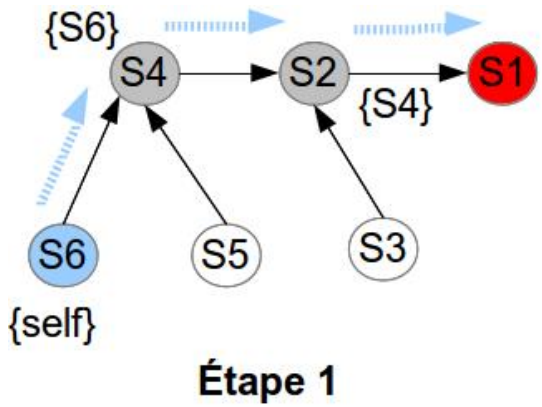
- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



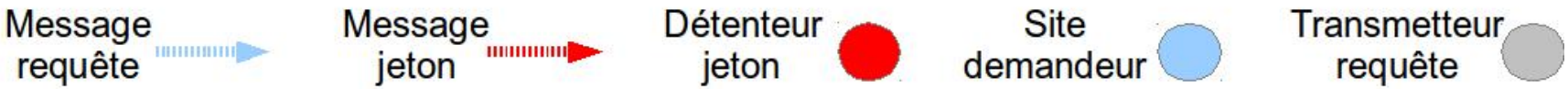
L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



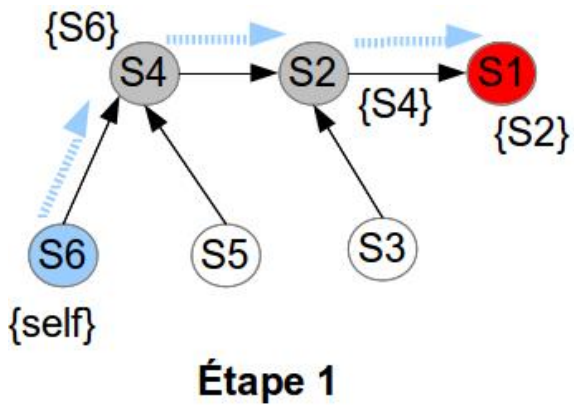
Étape 1



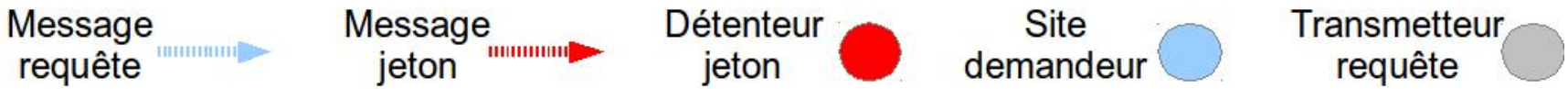
L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



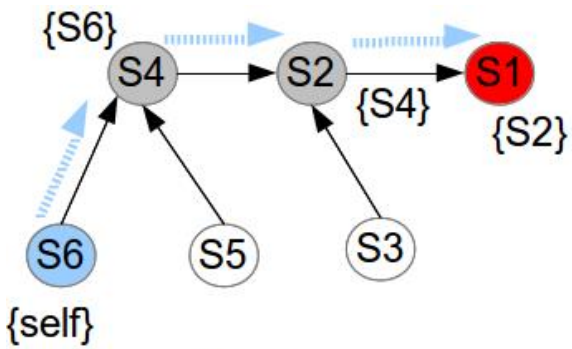
Étape 1



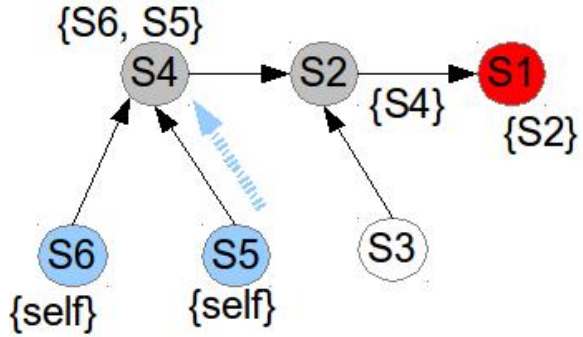
L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

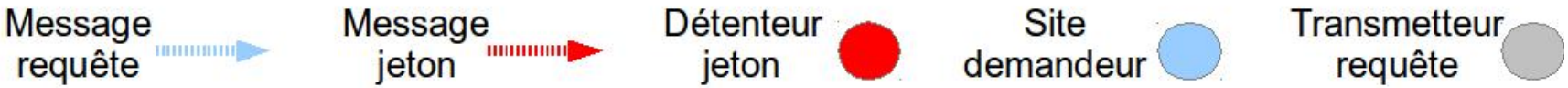
- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



Étape 1



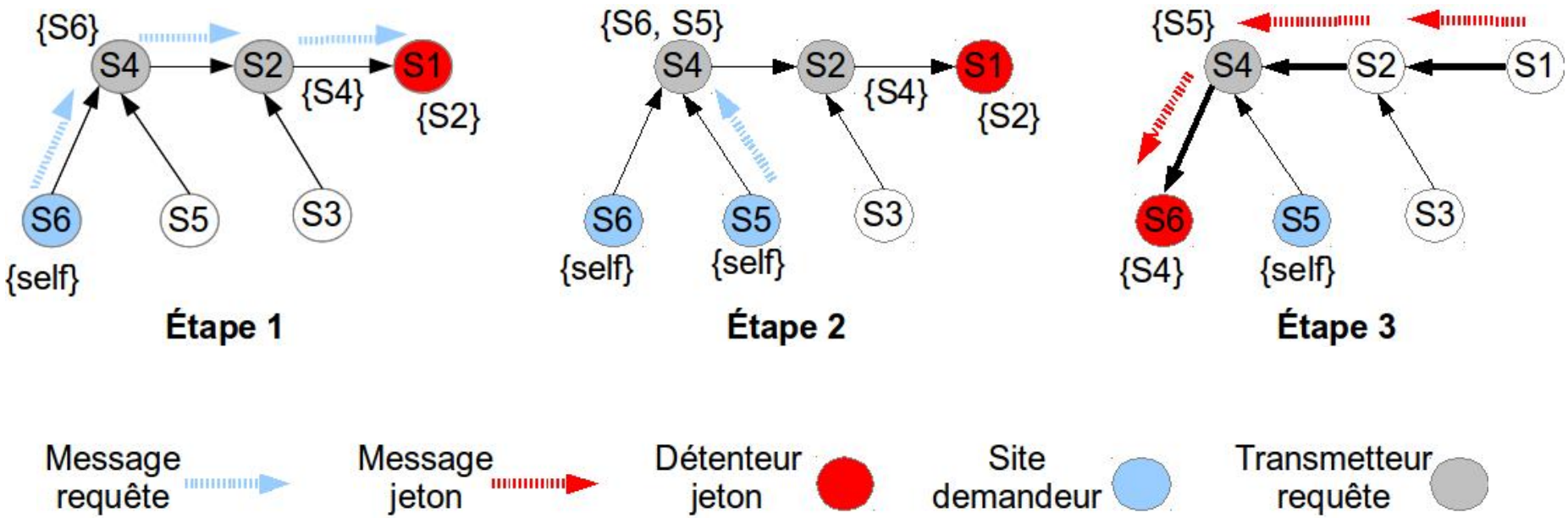
Étape 2



L'algorithme de base de Kanrar-Chaki

L'algorithme de Raymond (1989)

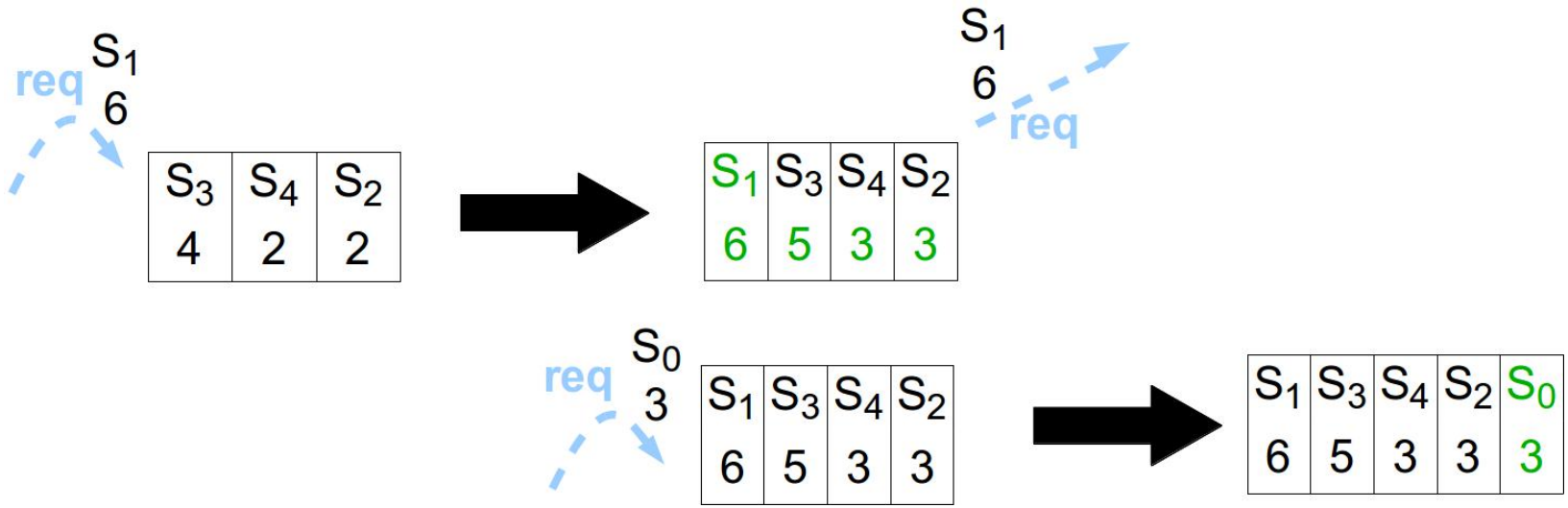
- Circulation de jeton dans une topologie logique en arbre statique
- Complexité moyenne en messages : $\mathcal{O}(LogN)$
- site racine = détenteur du jeton = seul site qui peut entrer en CS
- Stockage des requêtes pendantes dans des files locales
- Transmission d'une requête au père si la file locale est vide



Objectif 1 : propriété de vivacité

Utiliser l'algorithme de Kanrar-Chaki (2010)

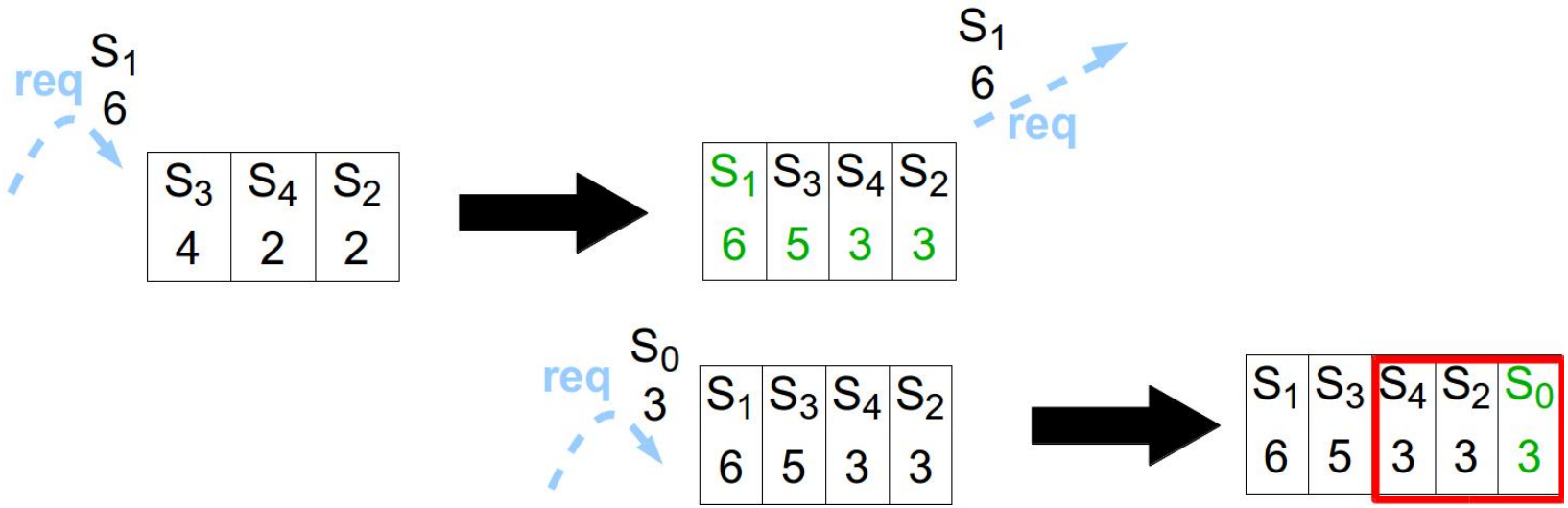
- Basé sur l'algorithme de Raymond
- une priorité = un entier : priorité haute = valeur haute
- retransmission si insertion de la requête reçue en tête de file locale (plus prioritaire).
- **Mécanisme dynamique** : à l'insertion d'un requête de priorité p dans une file locale, les priorités inférieures à p sont incrémentées de 1.



Objectif 1 : propriété de vivacité

Utiliser l'algorithme de Kanrar-Chaki (2010)

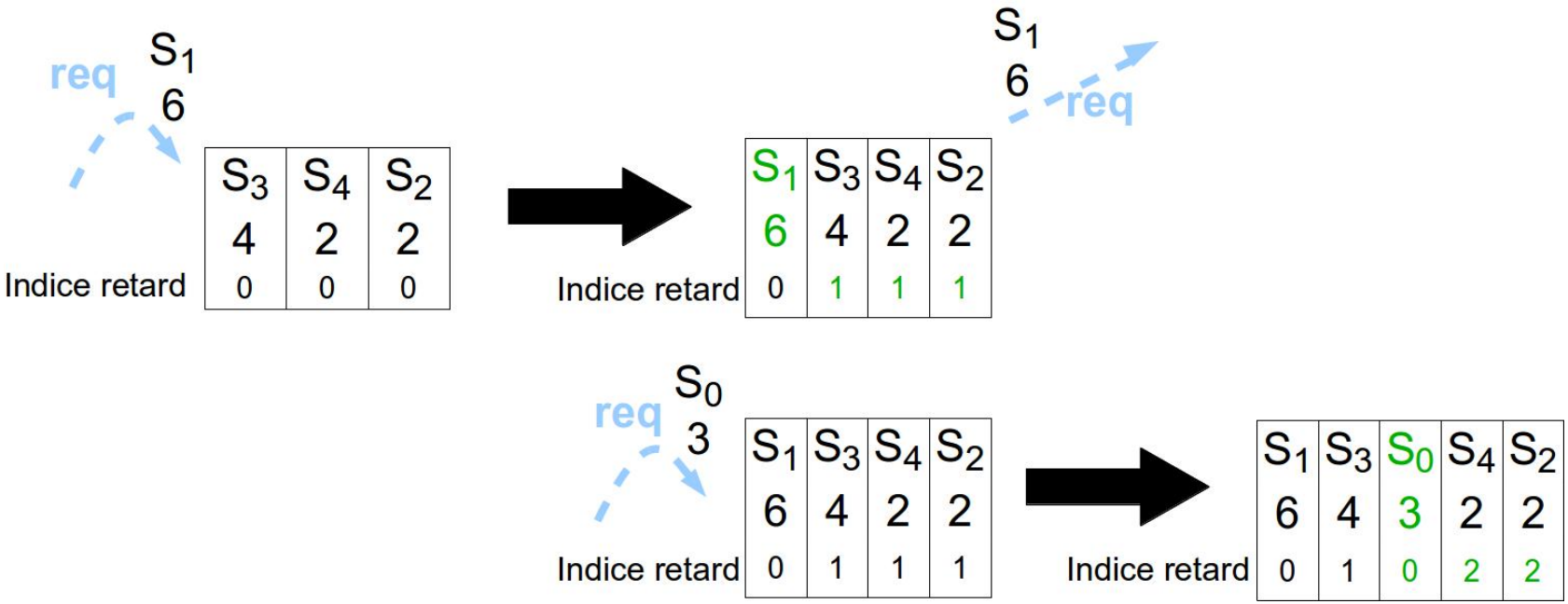
- Basé sur l'algorithme de Raymond
- une priorité = un entier : priorité haute = valeur haute
- retransmission si insertion de la requête reçue en tête de file locale (plus prioritaire).
- **Mécanisme dynamique** : à l'insertion d'un requête de priorité p dans une file locale, les priorités inférieures à p sont incrémentées de 1.



Objectif 2 : Réduction des inversions de priorité

Mécanisme de retard [CCGrid2012] [Compas2013]

- retarder l'incrémentation de priorité avec une fonction de palier $F(p)$
  - $\Rightarrow$ Pour augmenter sa priorité à p, une requête de priorité $p-1$ doit attendre $F(p)$ insertions de requêtes de plus haute priorité.
- pour retarder un maximum, une fonction exponentiellement croissante est considérée ($F(p) = 2^p$)



Objectif 2 : Réduction des inversions de priorité

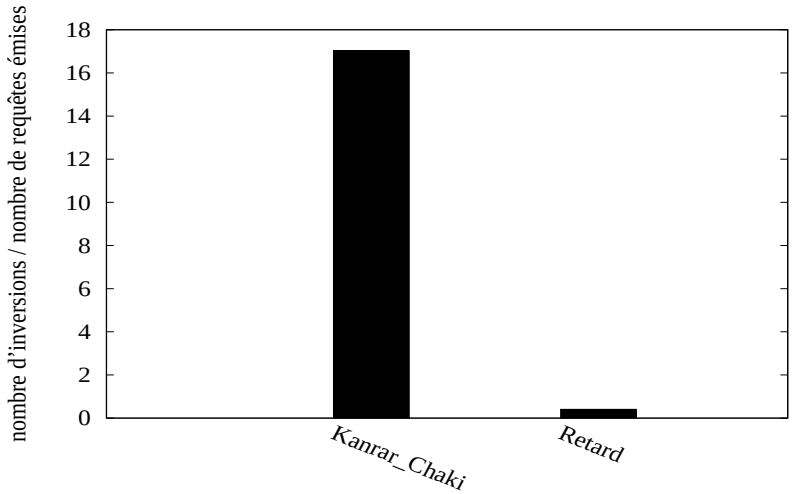
Paramètres expérimentaux

- Grid 5000 Nancy : 64 processus
- charge constante haute
- un processus garde la même priorité à chaque nouvelle requête

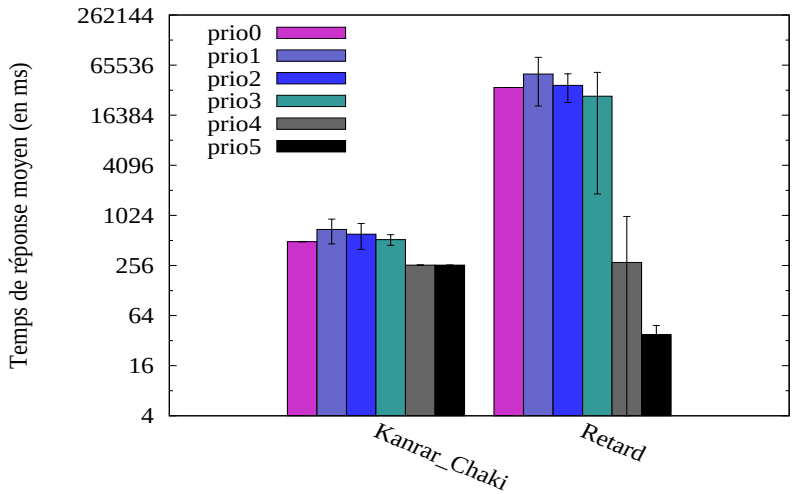
Objectif 2 : Réduction des inversions de priorité

Paramètres expérimentaux

- Grid 5000 Nancy : 64 processus
- charge constante haute
- un processus garde la même priorité à chaque nouvelle requête

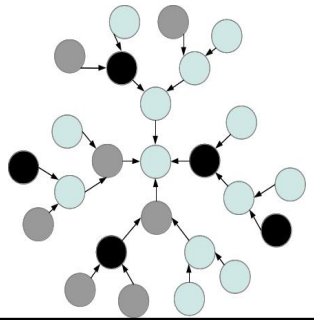
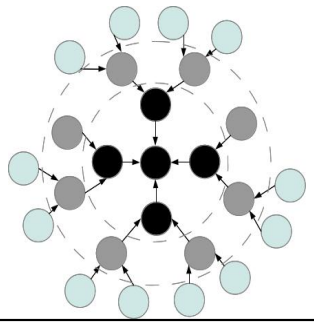


(j) nombre d'inversions / nombre de requêtes émises

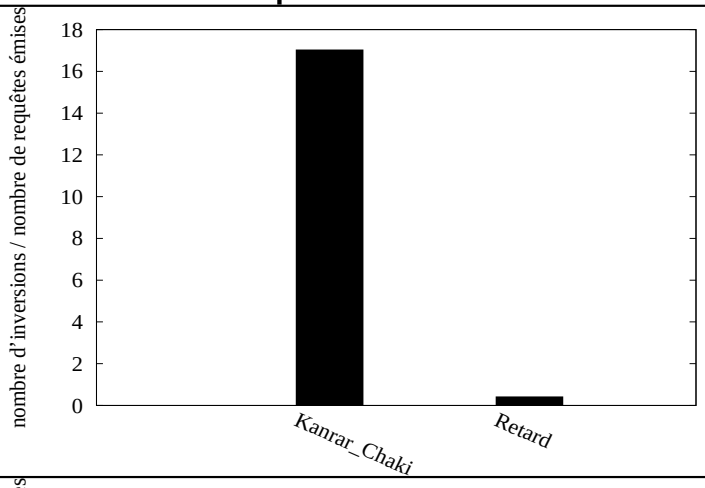
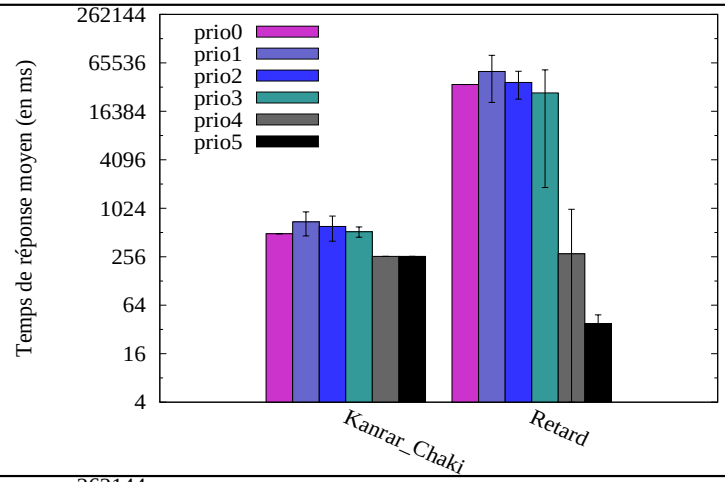
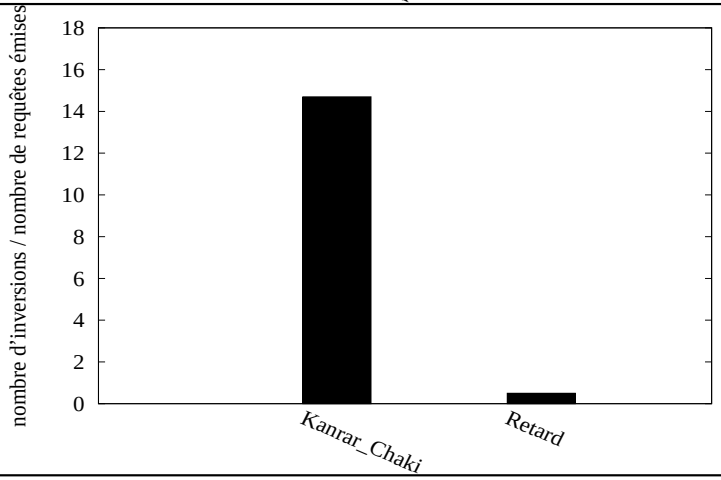
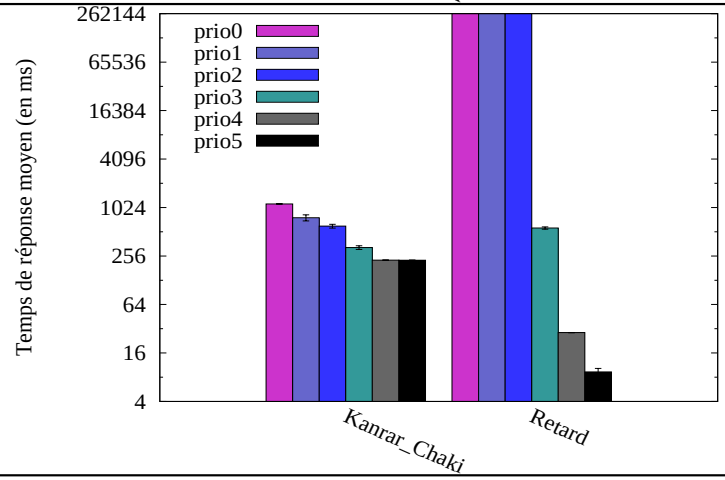


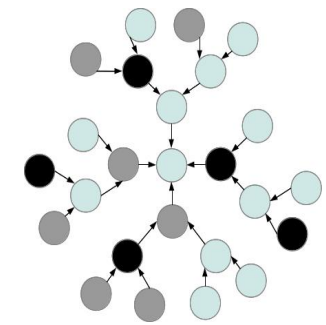
(k) Temps de réponse moyen (ms)

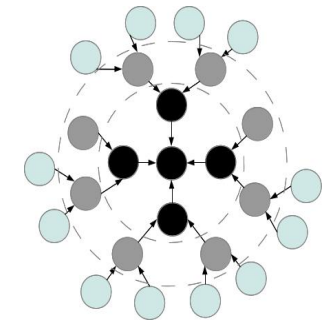
Impact de la localisation des priorités dans la topologie

Topologie	Nombre d'inversions / nombre de requêtes émises	Temps de réponse moyen (ms)
Aléatoire 		
Hiérarchique 		

Impact de la localisation des priorités dans la topologie

Topologie	Nombre d'inversions / nombre de requêtes émises	Temps de réponse moyen (ms)
Aléatoire		
Hiérarchique		





Priorité 2 Priorité 1 Priorité 0

Objectif 3 : Réduire le temps d'attente

Algorithme "Awareness" [ICPP2013]

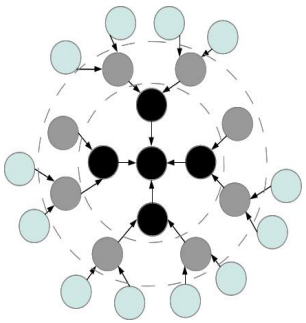
- **But : être indépendant de la topologie**
- **Idée principale : permettre aux nœuds de connaître le nombre de requêtes émises dans le système**
- Utilisation du jeton pour propager les informations

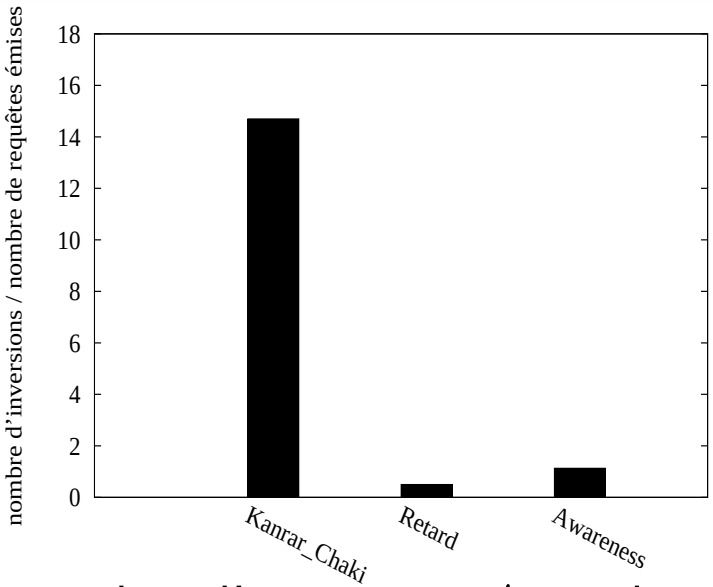
Objectif 3 : Réduire le temps d'attente

Algorithme "Awareness" [ICPP2013]

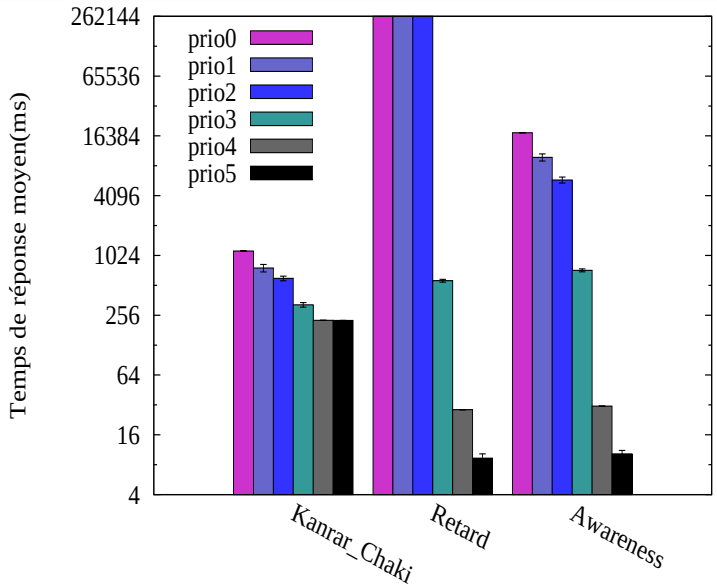
- **But : être indépendant de la topologie**
- **Idée principale : permettre aux nœuds de connaître le nombre de requêtes émises dans le système**
- Utilisation du jeton pour propager les informations

Hiérarchique





Nombre d'inversions / nombre de requêtes émises



Temps de réponse moyen (ms)

Conclusion contribution 1

Récapitulatif :

- priorités dynamiques $\Rightarrow$ **Vivacité**
- retard d'incrémentation $\Rightarrow$ **réduction des inversions de priorités** mais un temps d'attente énorme
- Algorithme Awareness $\Rightarrow$ **réduction conséquente du temps d'attente** pour un nombre d'inversions équivalent

Minimisation du temps de réponse vs. pas d'inversion de priorité = deux objectifs contradictoires :

- Un compromis qui dépend des besoins de l'application
- Ce compromis peut s'ajuster grâce à la fonction de palier.

Contribution 2 : Requêtes à contraintes temporelles

Motivation : Réserver des ressources pour une date précise.

Exemples : requêtes temps-réels

Principe :

- Associer chaque requête à une date d'échéance requise.
- Satisfaire les requêtes dans leur délai requis

Contribution 2 : Requêtes à contraintes temporelles

Objectifs

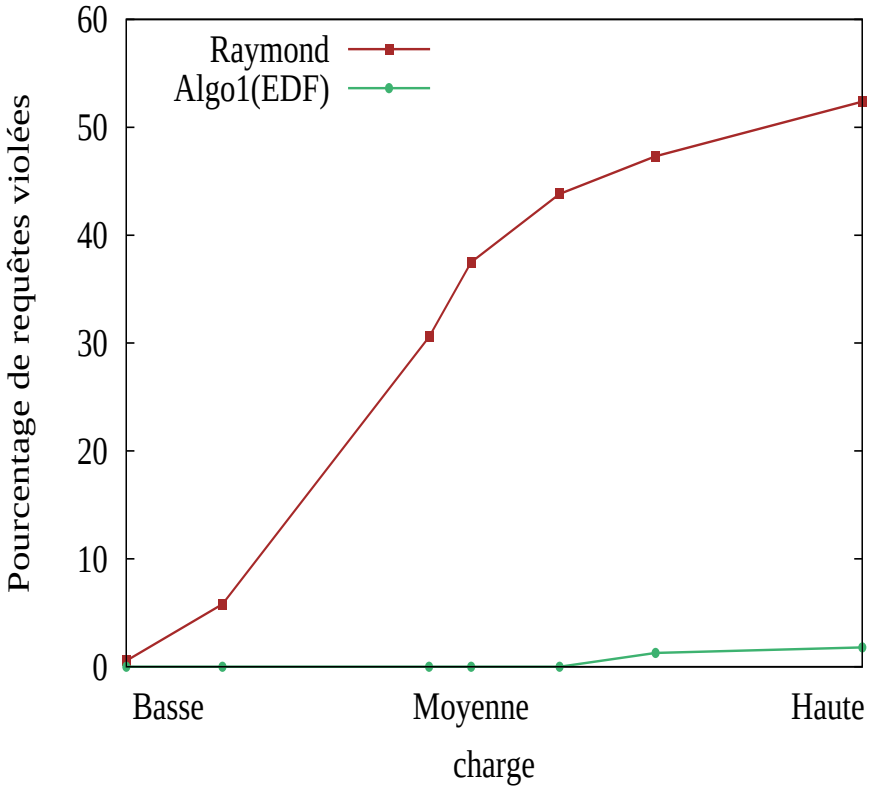
1. Garder une faible complexité en messages
2. Respecter les dates d'échéance des requêtes
3. Filtrer les requêtes en amont en fonction de l'état du système
4. Ne pas dégrader le taux d'utilisation de la ressource critique

Idées de l'algorithme [CCGrid2012] [CCGrid2013]

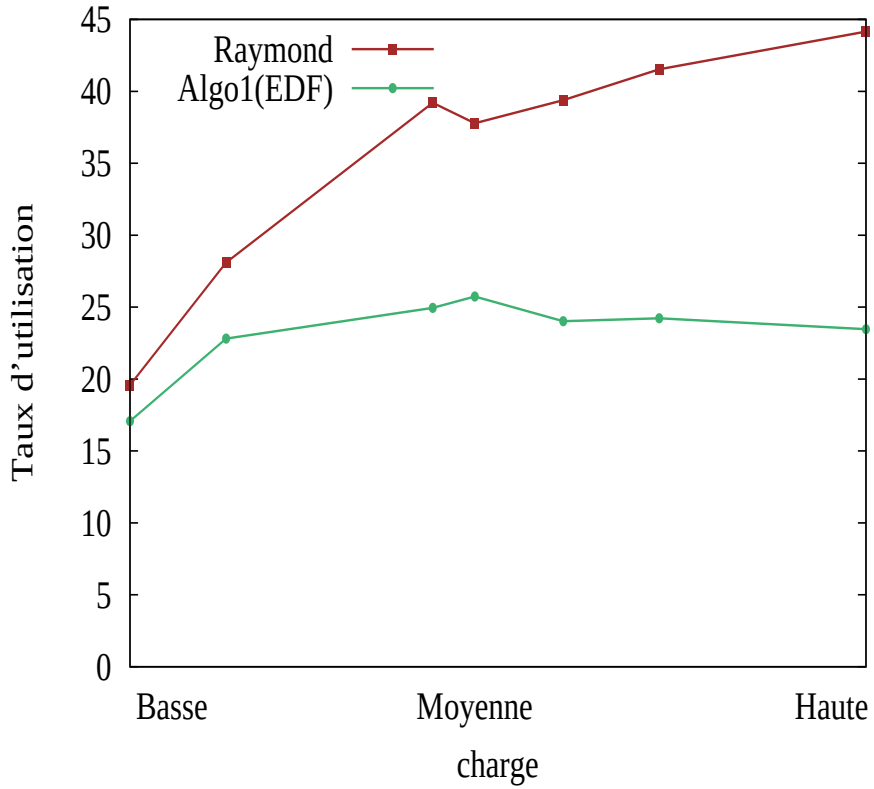
1. Basé sur l'algorithme de Raymond
2. Les files sont triées par date d'échéance croissante (ordre EDF)
3. Un contrôle d'admission distribué
4. Un mécanisme de préemption du jeton

Contrôle d'admission

Introduction d'un contrôle d'admission distribué :



(l) Pourcentage de requêtes violées

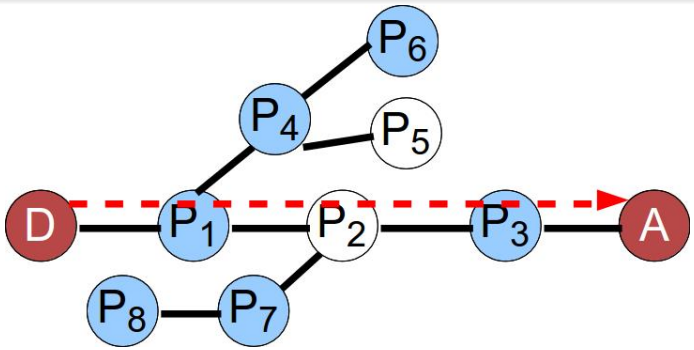


(m) Taux d'utilisation (%)

Mécanisme de préemption

Principe

- Prise en compte de la localité des requêtes
- Le jeton peut être détourné de son chemin initial.

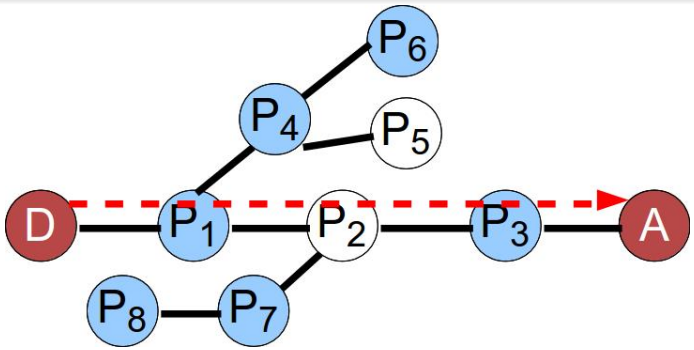


Sans préemption

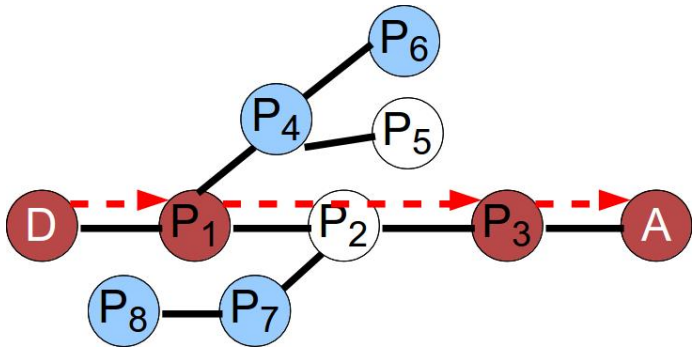
Mécanisme de préemption

Principe

- Prise en compte de la localité des requêtes
- Le jeton peut être détourné de son chemin initial.



Sans préemption

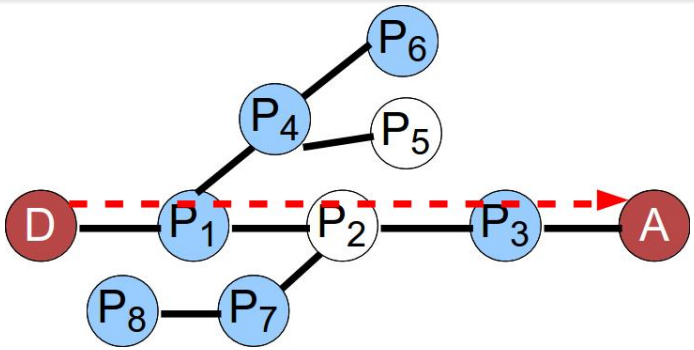


Préemption par interception

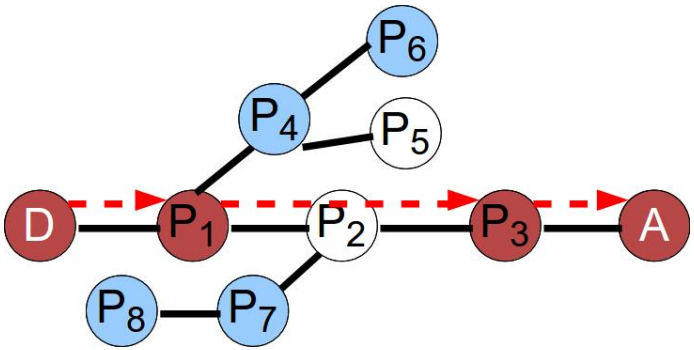
Mécanisme de préemption

Principe

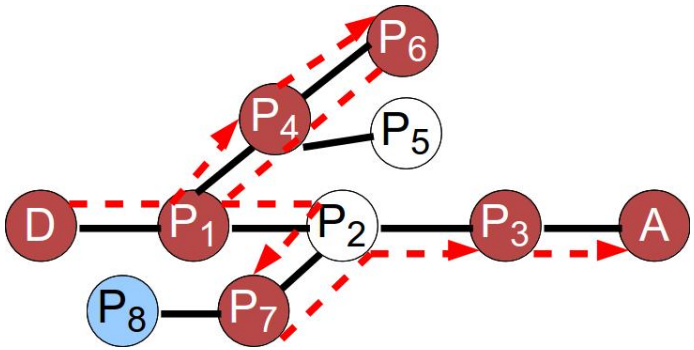
- Prise en compte de la localité des requêtes
- Le jeton peut être détourné de son chemin initial.



Sans préemption



Préemption par interception

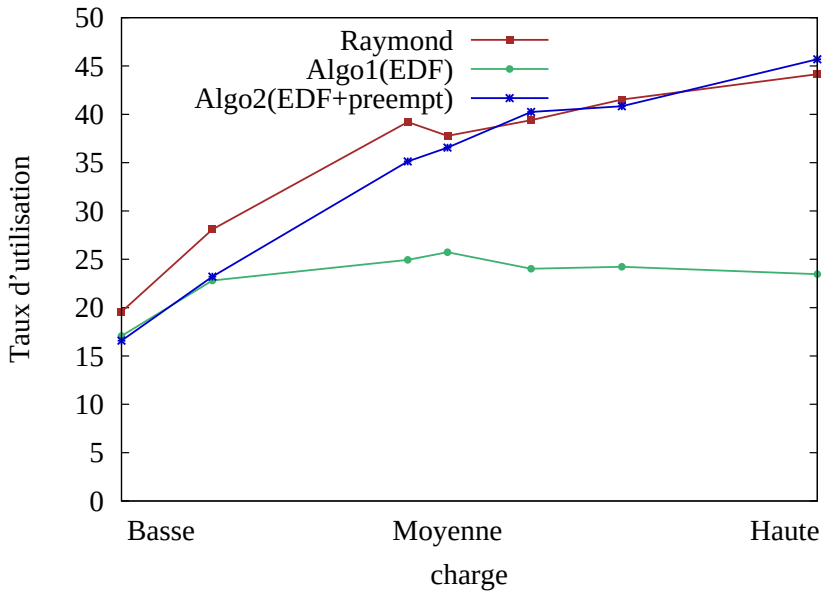


Préemption par détournement

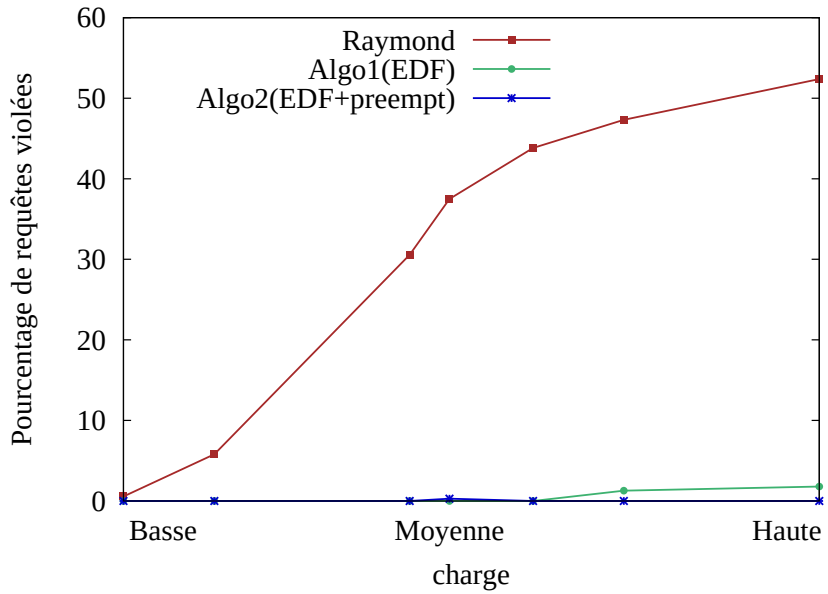
Mécanisme de préemption

Principe

- Prise en compte de la localité des requêtes
- Le jeton peut être détourné de son chemin initial.



(n) Taux d'utilisation (%)



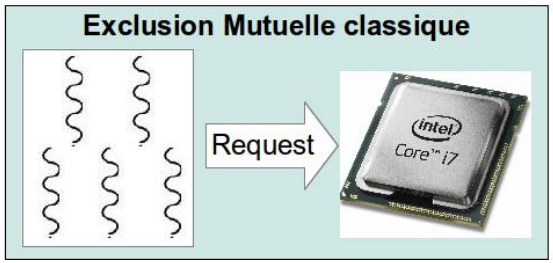
(o) Pourcentage de requêtes violées

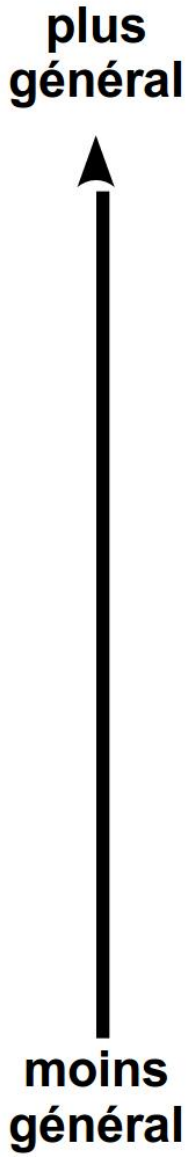
Conclusion contribution 2

Élaboration d'un algorithme pour des requêtes à contraintes temporelles :

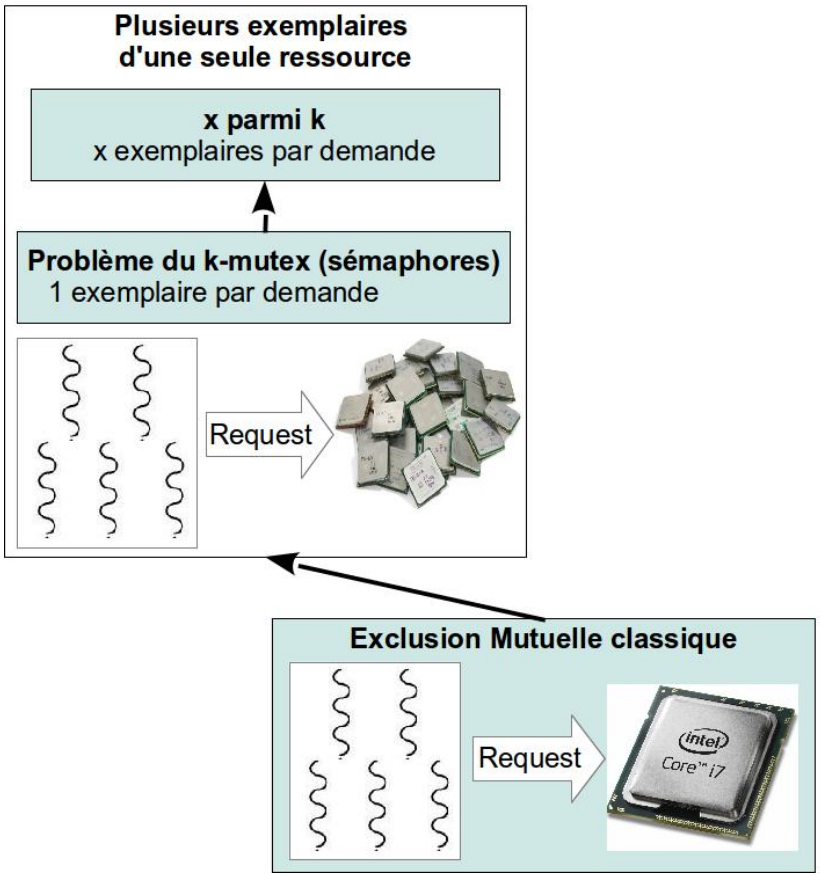
- Contrôle d'admission et ordonnancement EDF
  $\Rightarrow$ **minimisation des violations**
- Mécanisme de préemption
  $\Rightarrow$ **amélioration du taux d'utilisation de la section critique**

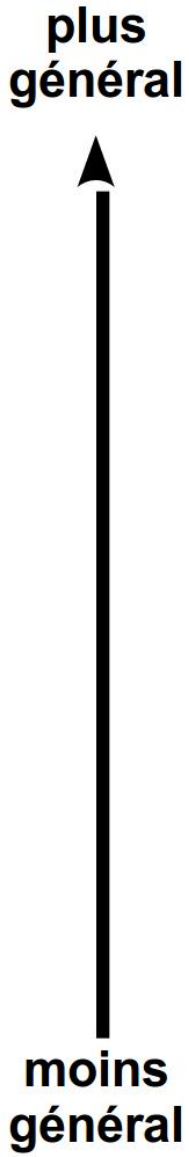
Contribution 3 : Requêtes à plusieurs ressources



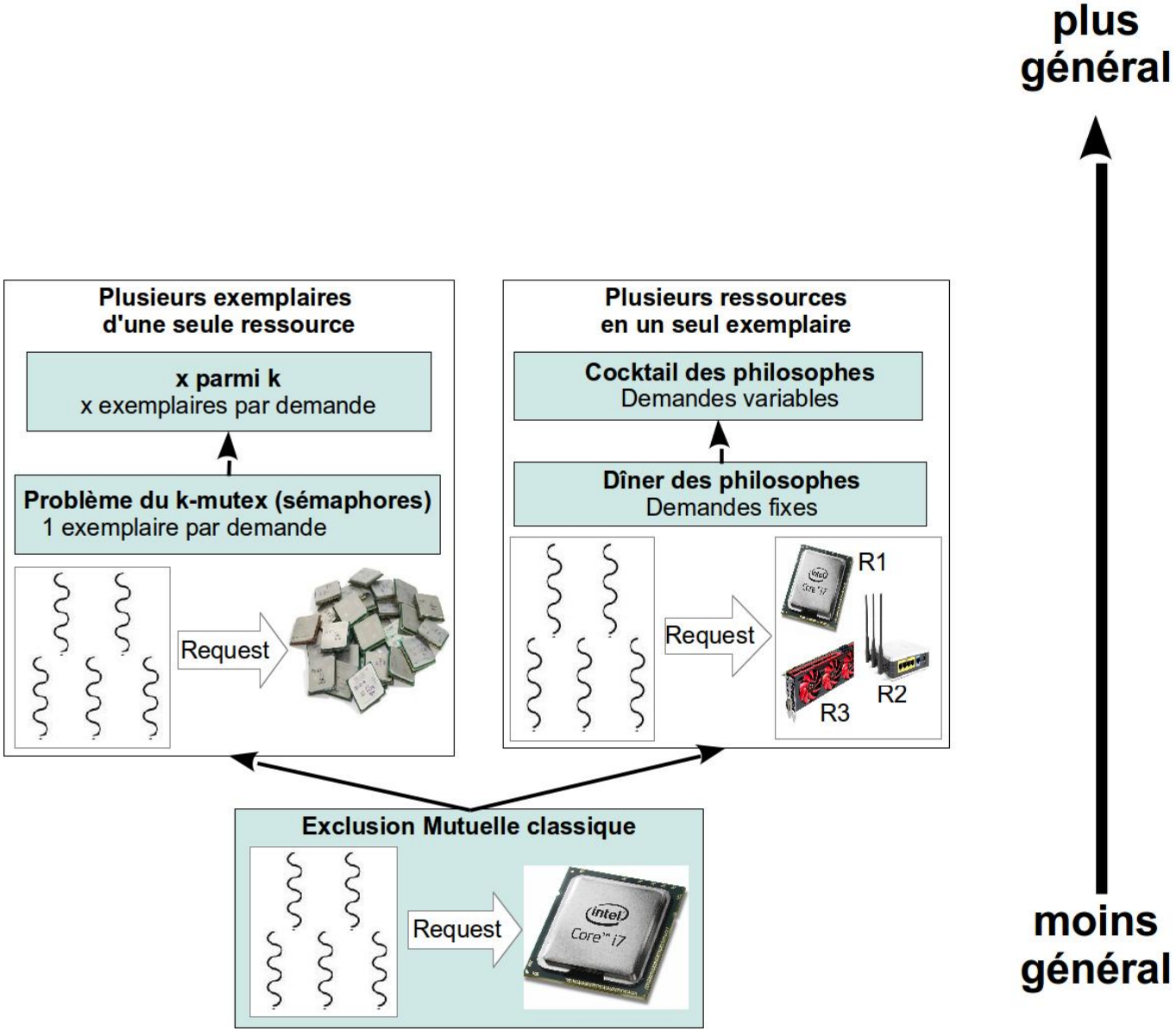


Contribution 3 : Requêtes à plusieurs ressources

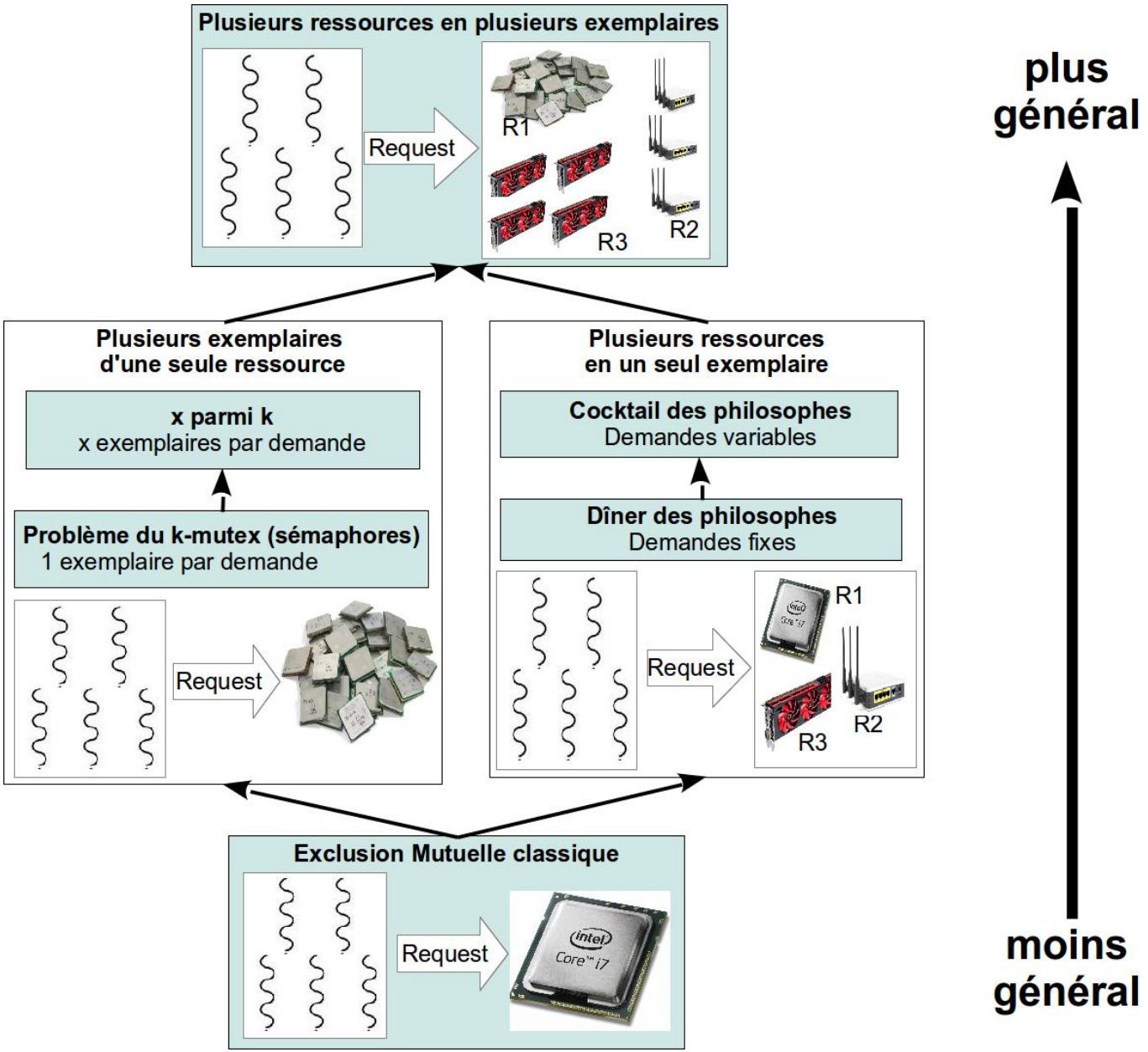




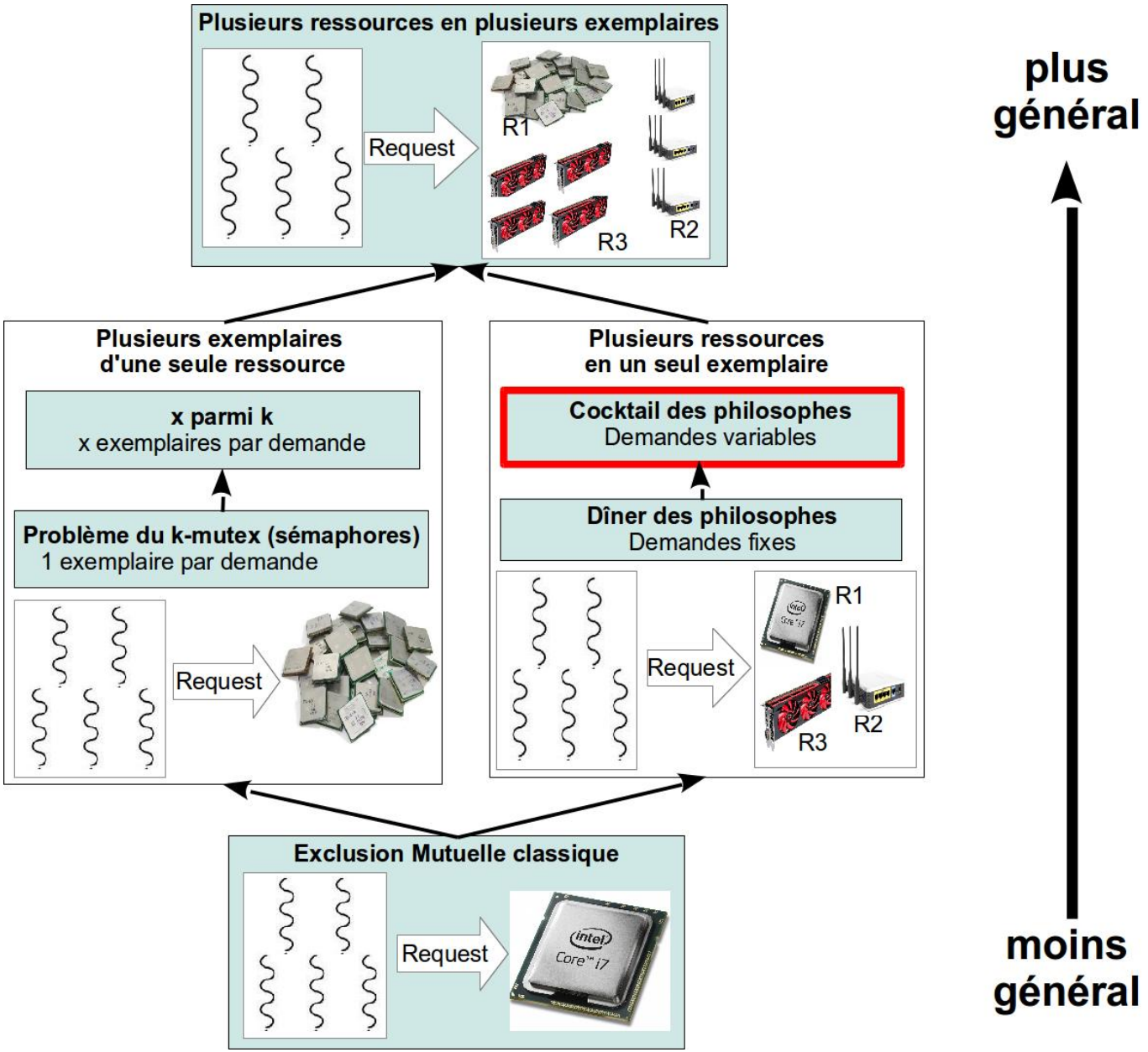
Contribution 3 : Requêtes à plusieurs ressources



Contribution 3 : Requêtes à plusieurs ressources



Contribution 3 : Requêtes à plusieurs ressources



Cocktail des philosophes : solutions existantes

Avec connaissance du graphe de conflits [Chandy-Misra84] :

- ✓ communication entre processus seulement si conflit
- ✗ connaissance du graphe conflit

  $\Longrightarrow$ contrainte forte qui limite le cadre applicatif

Sans connaissance du graphe de conflits [Bouabdallah-Laforest00] :

- ✗ communication entre processus non-conflictuels

  $\Longrightarrow$ ralentit l'accès aux ressources

Objectifs

Proposer un Algorithme qui :

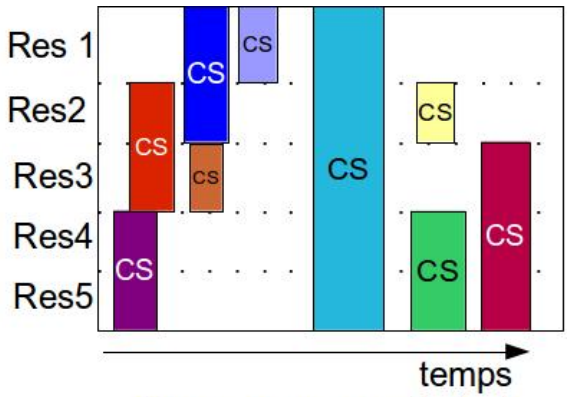
- ne nécessite pas de connaître a priori le graphe de conflits
- limite la communication entre les processus qui n'ont pas de conflit

Limites de l'algorithme de Bouabdallah-Laforest

L'algorithme de Bouabdallah-Laforest a deux limites :

1. **Section critique de contrôle (verrou global) :**
   $\Longrightarrow$ synchronisation entre processus non-conflictuels

2. **Ordonnancement statique :**
   $\Longrightarrow$ pas de préemption : on ne revient pas sur un verrouillage

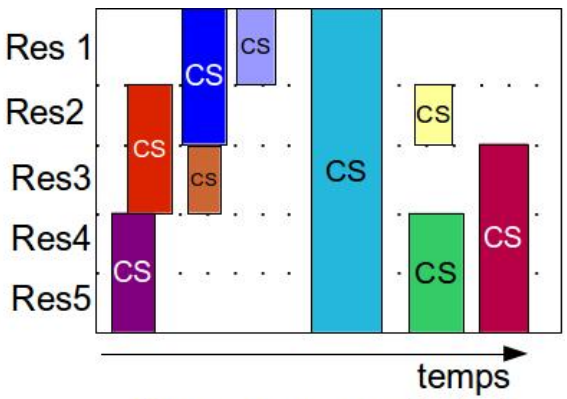


Avec verrou global

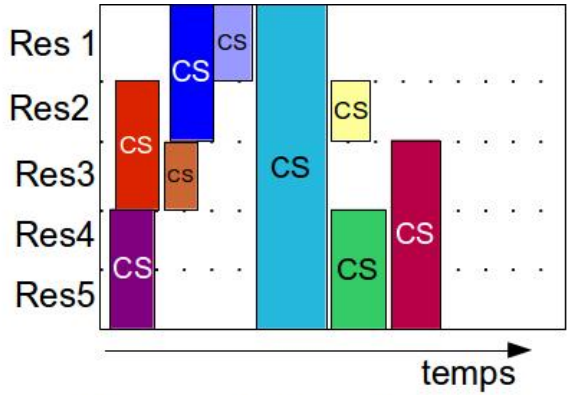
Limites de l'algorithme de Bouabdallah-Laforest

L'algorithme de Bouabdallah-Laforest a deux limites :

1. **Section critique de contrôle (verrou global) :**
   $\Longrightarrow$ synchronisation entre processus non-conflictuels
   Solution proposée : utilisation de compteurs
2. **Ordonnancement statique :**
   $\Longrightarrow$ pas de préemption : on ne revient pas sur un verrouillage



Avec verrou global



Sans verrou global

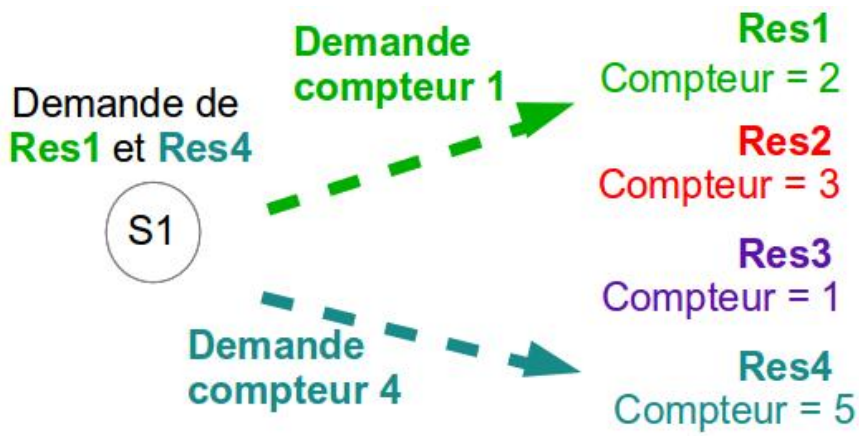
Principe de l'algorithme : Suppression du verrou global

Principe de l'algorithme : Suppression du verrou global

- M jetons et M compteurs incrémentaux à accès exclusif

Principe de l'algorithme : Suppression du verrou global

- M jetons et M compteurs incrémentaux à accès exclusif
- **Première étape** :
  - demander pour chaque ressource requise la valeur du compteur

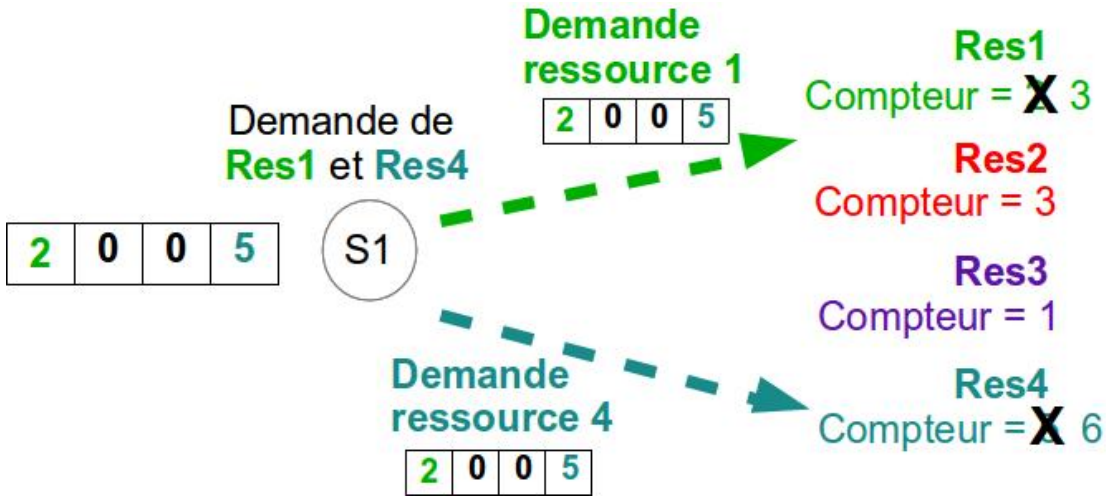


Principe de l'algorithme : Suppression du verrou global

- M jetons et M compteurs incrémentaux à accès exclusif
- **Première étape** :
  - demander pour chaque ressource requise la valeur du compteur
  - obtention d'un unique vecteur de M entiers
  - compteur nul pour une ressource non demandée

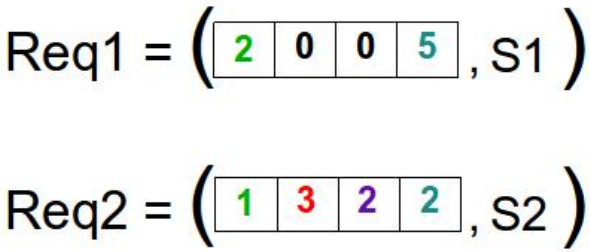
Principe de l'algorithme : Suppression du verrou global

- *M* jetons et *M* compteurs incrémentaux à accès exclusif
- **Première étape** :
  - demander pour chaque ressource requise la valeur du compteur
  - obtention d'un unique vecteur de *M* entiers
  - compteur nul pour une ressource non demandée
- **Deuxième étape** : demander **indépendamment** chaque ressource requise en indiquant le vecteur



Principe de l'algorithme : éviter les interblocages

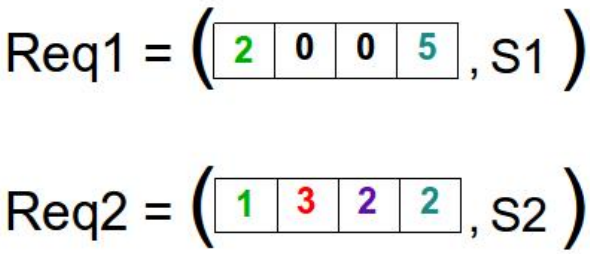
Les interblocages sont évités s'il existe un ordre total sur les requêtes.



Principe de l'algorithme : éviter les interblocages

Les interblocages sont évités s'il existe un ordre total sur les requêtes.

Calculer la moyenne des compteurs non nuls :

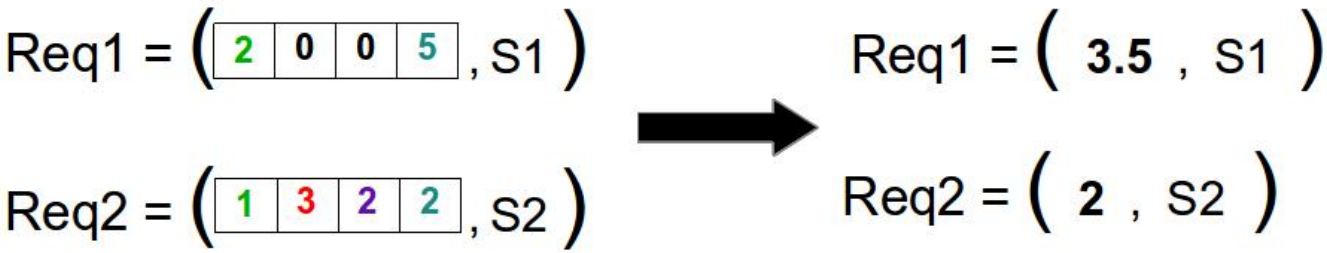


Principe de l'algorithme : éviter les interblocages

Les interblocages sont évités s'il existe un ordre total sur les requêtes.

Calculer la moyenne des compteurs non nuls :

- plus la moyenne est petite, plus la requête est prioritaire (ordre partiel)
- en cas de réel égal, le site au plus petit identifiant (ordre total)

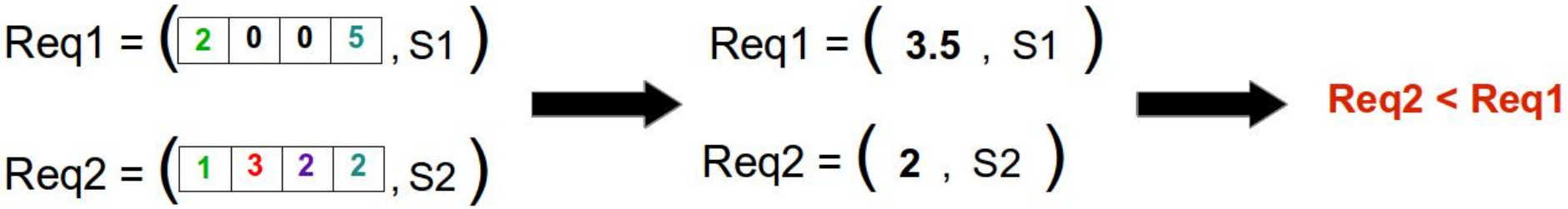


Principe de l'algorithme : éviter les interblocages

Les interblocages sont évités s'il existe un ordre total sur les requêtes.

Calculer la moyenne des compteurs non nuls :

- plus la moyenne est petite, plus la requête est prioritaire (ordre partiel)
- en cas de réel égal, le site au plus petit identifiant (ordre total)

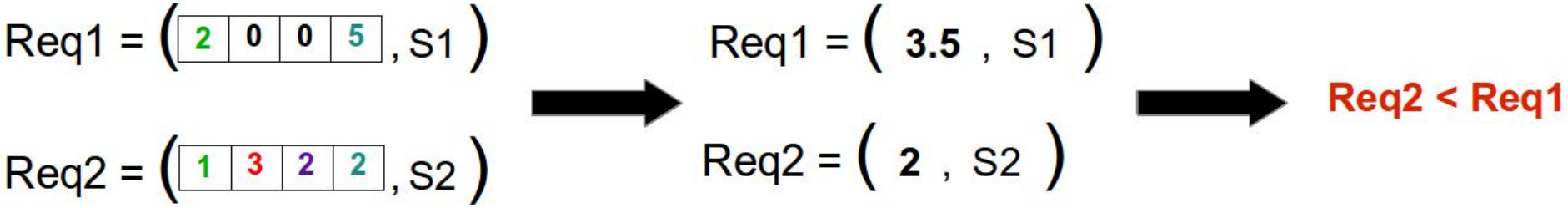


Principe de l'algorithme : éviter les interblocages

Les interblocages sont évités s'il existe un ordre total sur les requêtes.

Calculer la moyenne des compteurs non nuls :

- plus la moyenne est petite, plus la requête est prioritaire (ordre partiel)
- en cas de réel égal, le site au plus petit identifiant (ordre total)



Vivacité assurée : la moyenne minimale augmente à chaque nouvelle requête

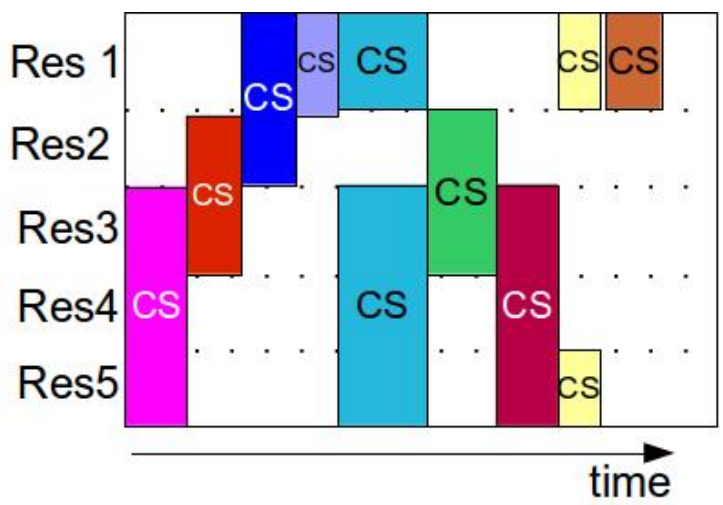
Expérimentation

- 32 processus Grid 5000 Lyon
- 80 ressources
- **Charge constante** : régime de stationnaire de requêtes
- ϕ : nombre maximum de ressources qu'un site peut demander.
- A chaque nouvelle requête :
  - choix d'une taille de requête x entre 1 et ϕ
  - choix de x ressources dans l'ensemble des ressources

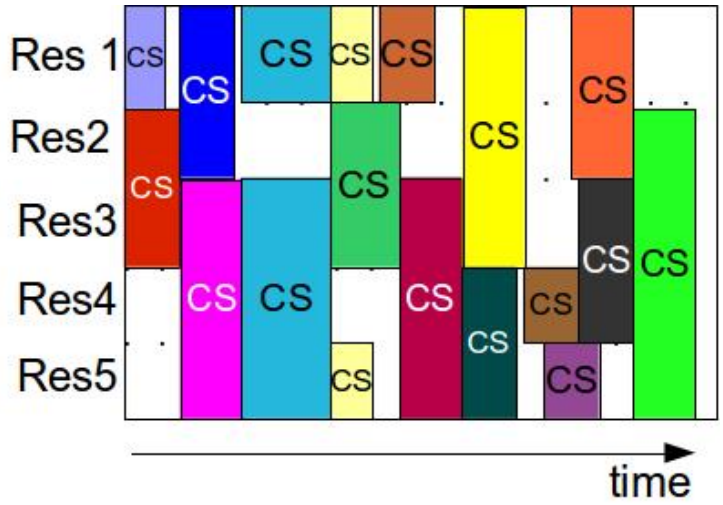
Expérimentation : métriques considérées

- **Taux d'utilisation des ressources** :

  pourcentage de temps où les ressources sont utilisées



mauvais



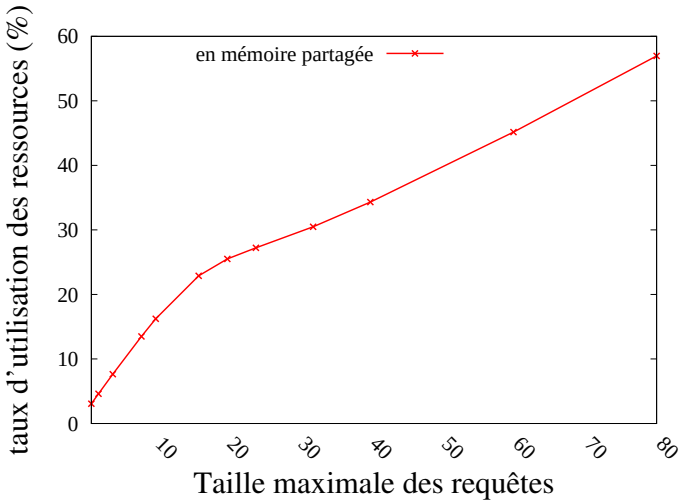
bon

- **Temps d'attente moyen** :

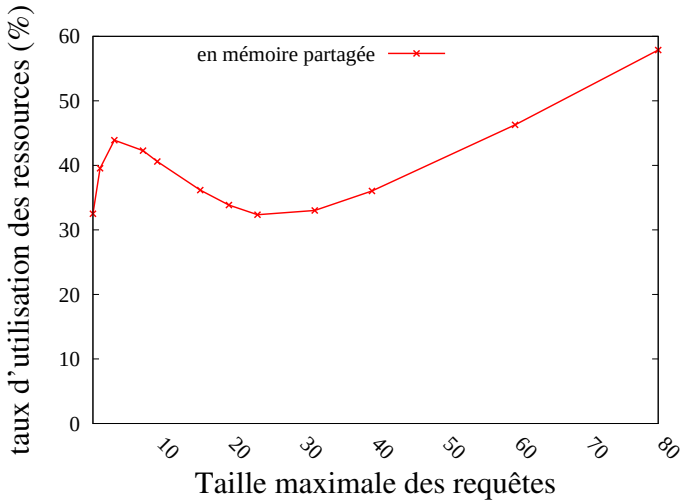
  temps entre le moment où un processus émet une requête et le moment où il entre en section critique

Résultats

moyenne charge



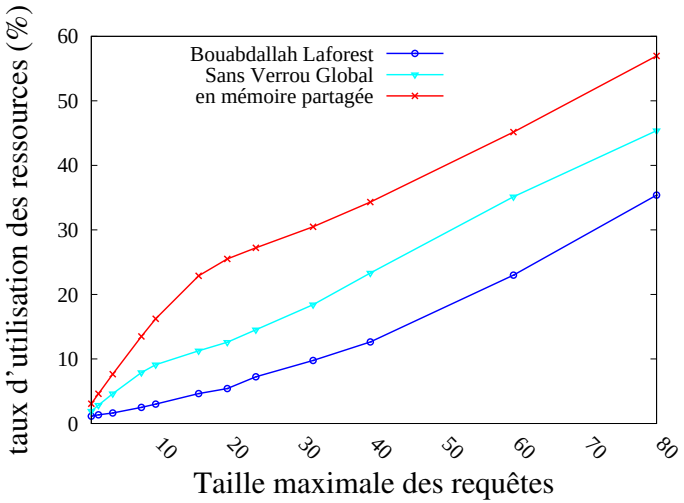
haute charge



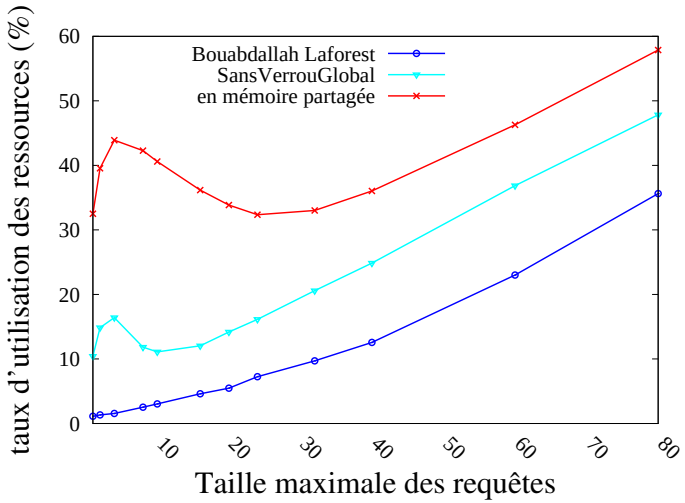
Taux d'utilisation

Résultats

moyenne charge



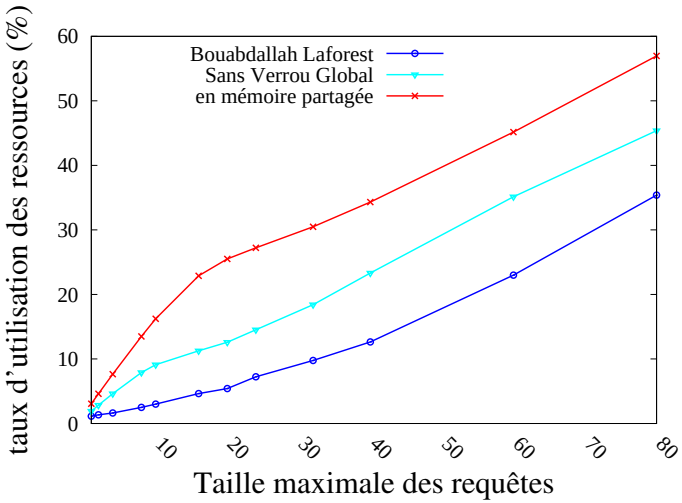
haute charge



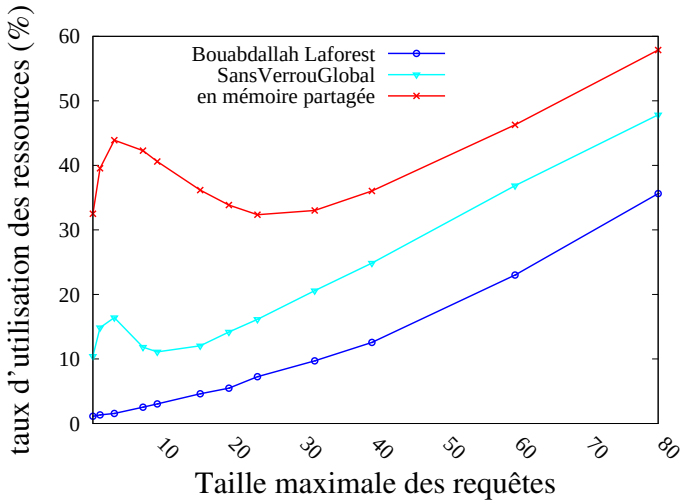
Taux d'utilisation

Résultats

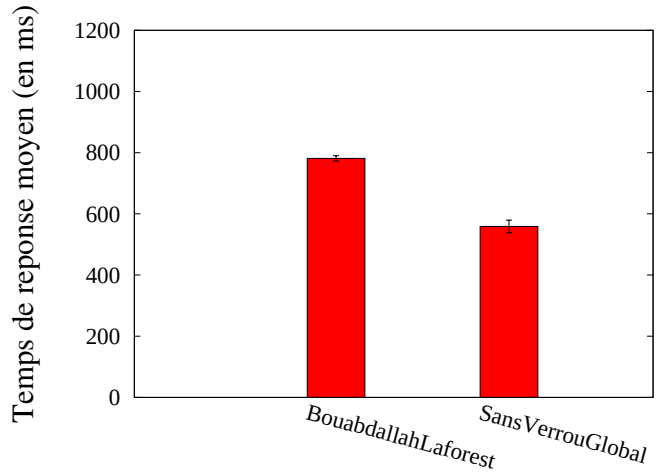
moyenne charge

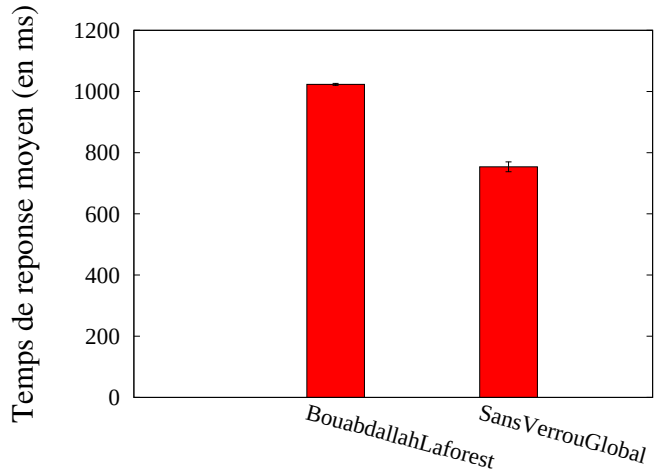


haute charge



Taux d'utilisation



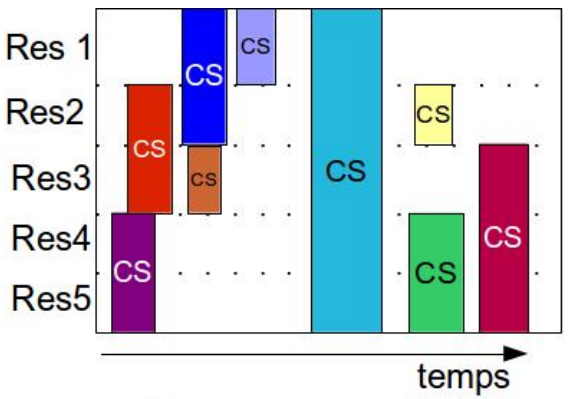


Temps de réponse moyen en ms (abscisse = 80)

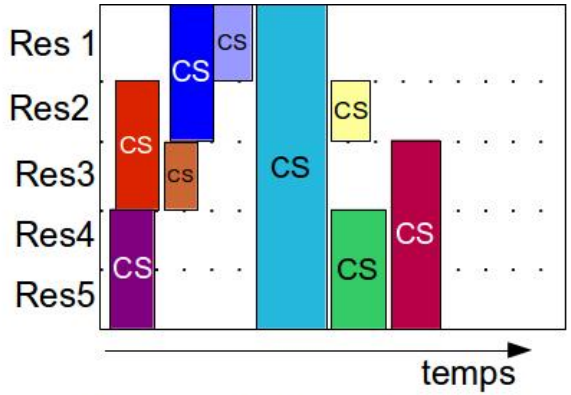
Limites de l'algorithme de Bouabdallah-Laforest

L'algorithme de Bouabdallah-Laforest a deux limites :

1. **Section critique de contrôle (verrou global) :**
   $\Longrightarrow$ synchronisation entre processus non-conflictuels
   Solution proposée : utilisation de compteurs
2. **Ordonnancement statique :**
   $\Longrightarrow$ pas de préemption : on ne revient pas sur un verrouillage



Avec verrou global

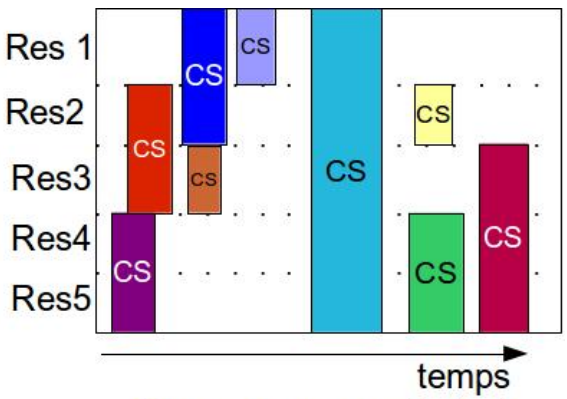


Sans verrou global

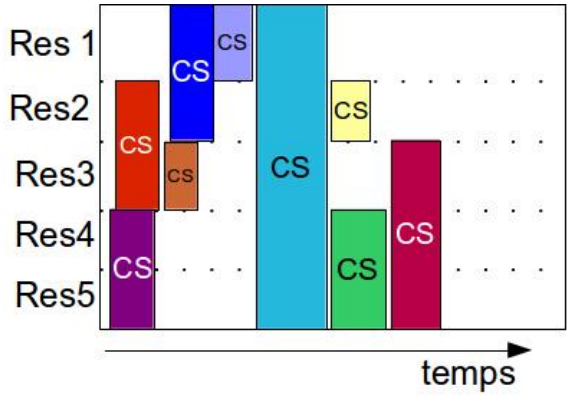
Limites de l'algorithme de Bouabdallah-Laforest

L'algorithme de Bouabdallah-Laforest a deux limites :

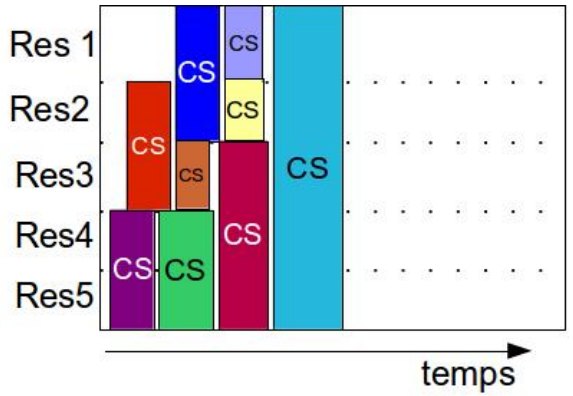
1. **Section critique de contrôle (verrou global) :**
   $\Longrightarrow$ synchronisation entre processus non-conflictuels
   Solution proposée : utilisation de compteurs
2. **Ordonnancement statique :**
   $\Longrightarrow$ pas de préemption : on ne revient pas sur un verrouillage
   Solution proposée : utilisation d'un mécanisme de prêt



Avec verrou global



Sans verrou global



Sans verrou global
Avec mécanisme de prêt

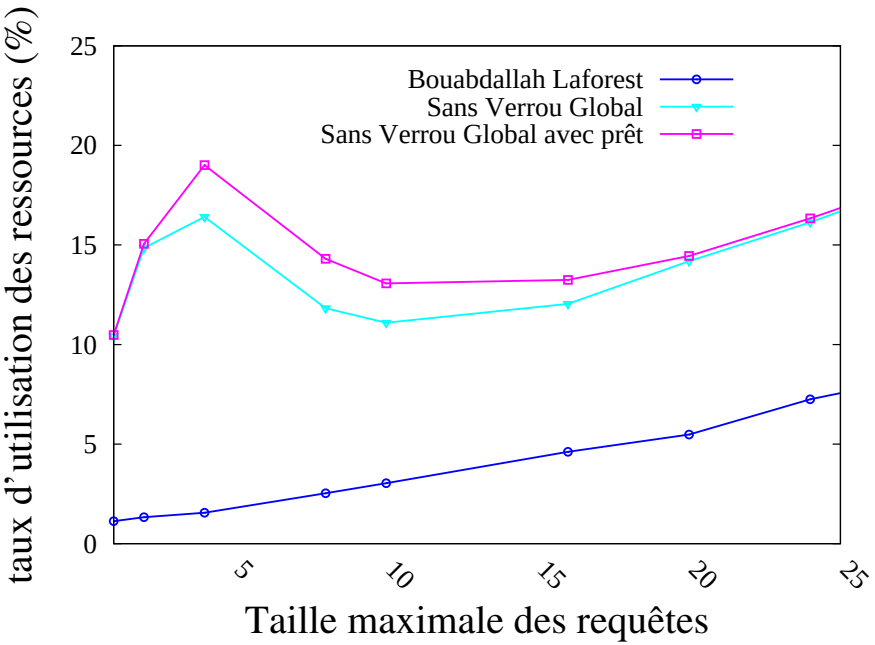
Mécanisme de prêt : permettre la préemption

Ajout d'un mécanisme de prêt :

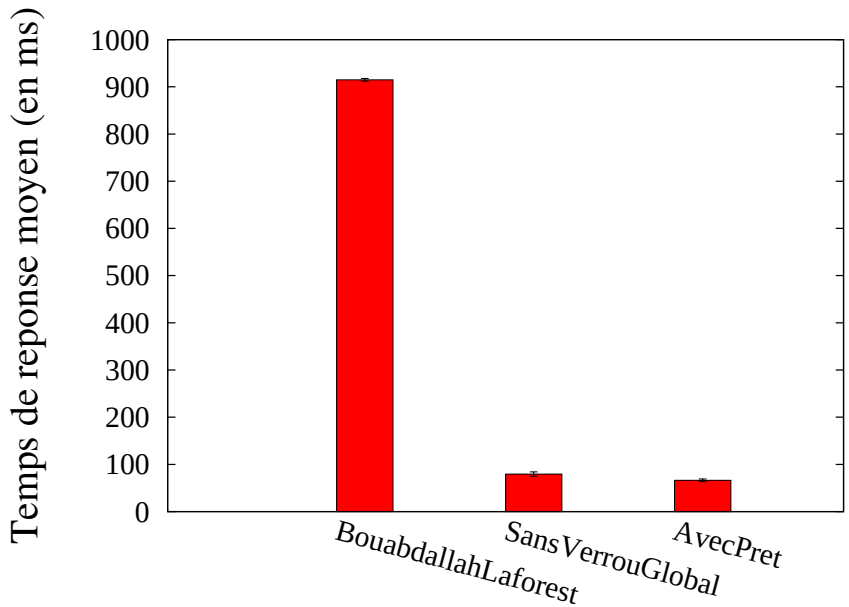
- sans ajouter de synchronisation globale
- en conservant la propriété de vivacité

Proposition d'un mécanisme basé sur deux principes :

- **Principe 1 :** emprunt qu'à un seul processus $\Longrightarrow$ pas d'interblocages
- **Principe 2 :** prêt qu'à un seul processus $\Longrightarrow$ pas de famine



Taux d'utilisation



Temps de réponse moyen en ms (abscisse = 4)

Conclusion contribution 3

Un nouveau schéma distribué d'allocation de ressources qui :

- ne nécessite pas de connaître a priori le graphe de conflits
- limite la communication entre les processus qui n'ont pas de conflit
  $\Longrightarrow$ supprime le verrou global en utilisant des compteurs
- permet d'ordonnancer les requêtes dynamiquement
  $\Longrightarrow$ introduction d'un mécanisme de prêt
- améliore le taux d'utilisation des ressources et réduit le temps d'attente moyen

Conclusion générale

Conceptions de nouveaux algorithmes d'exclusion mutuelle :

- **Exclusion mutuelle à priorité** :
  $\Longrightarrow$ équilibre entre inversions et temps d'attente
- **Exclusion mutuelle à contraintes temporelles** :
  $\Longrightarrow$ Contrôle d'admission pour limiter les violations
  $\Longrightarrow$ Mécanisme de préemption pour augmenter le taux d'utilisation
- **Exclusion mutuelle généralisée** :
  $\Longrightarrow$ pas de connaissance préalable du graphe de conflits
  $\Longrightarrow$ amélioration du taux d'utilisation en limitant la communication entre processus non-conflictuels et en ordonnançant dynamiquement les requêtes.

Perspectives

Requêtes à contraintes temporelles :

- Étudier l'impact d'une marge d'erreur dans le contrôle d'admission

Requêtes à plusieurs ressources :

- Améliorer le mécanisme de prêt
- Considérer plusieurs exemplaires par ressource
- Étude de notre solution sur une topologie hiérarchique pour prendre en compte la localité physique des processus

Globales :

- Revalidation des algorithmes avec des traces réelles d'applications
- Combinaison des algorithmes
- Adaptation aux systèmes dynamiques (élasticité)

Publications

Conférences internationales :

- J. Lejeune, L. Arantes, J. Sopena, P. Sens, A prioritized distributed mutual exclusion algorithm balancing priority inversions and response time In ICPP 2013
- D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens. Towards QoS- Oriented SLA Guarantees for Online Cloud Services. In CC- GRID'13.
- J. Lejeune, L. Arantes, J. Sopena, P. Sens, Service level agreement for distributed mutual exclusion in cloud computing. In CC-GRID'12

Conférence francophone :

- J. Lejeune, L. Arantes, J. Sopena, P. Sens. Un algorithme distribué efficace d'exclusion mutuelle généralisée sans connaissance préalable des conflits. In ComPAS 2014 (best paper track parallélisme).
- J. Lejeune, L. Arantes, J. Sopena, P. Sens. Un algorithme équitable d'exclusion mutuelle distribuée avec priorité. In ComPAS 2013 (track CFSE)

En cours de soumission :

- JPDC : J. Lejeune, L. Arantes, J. Sopena, P. Sens, A Fair Starvation-free Prioritized Mutual exclusion Algorithm for distributed systems
- IPDPS 2015 : J. Lejeune, L. Arantes, J. Sopena, P. Sens, An efficient decentralized resources allocator without prior knowledge about conflicts graph