# Gestion autonome de flots d'exécution événementiels

Fabien GAUD, M2R S&L

Équipe SARDES (INRIA/LIG)

18 juin 2007


# Sommaire

# Problématique

## But

- Proposer un modèle de programmation et son modèle d'exécution **performant**

## Modèles actuels

- Modèle à base de threads
- Modèle à base d'événements
- Chacun est performant sous certaines conditions

## Proposition

- Un modèle mixte permettant d'obtenir le meilleur de ces deux modèles

# Modèle à base de threads

## Principes des threads

- Les threads sont une forme légère des processus
- Un processus est l'unité de base d'une exécution
- Les processus sont ordonnancés par le système d'exploitation
- Les threads peuvent partager de la mémoire
- Parallélisation de tout ou partie de l'exécution
- Permet l'utilisation des multiprocesseurs

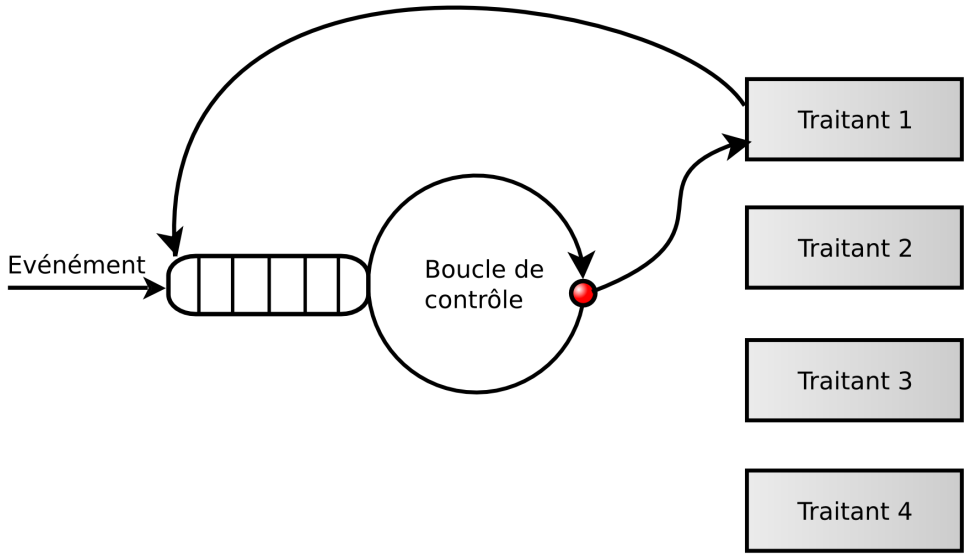
## Performances

- Bonnes performances à basse charge
- Mauvaises performances à haute charge (coût des changements de contexte)

# Modèle à base d'événements

## Principes

- Basé sur une boucle et des événements
- Un seul et unique flot d'exécution (thread)
- Les traitants doivent être courts et non-bloquants
- Pas d'utilisation possible des multiprocesseurs



# Modèle à base d'événements (2)

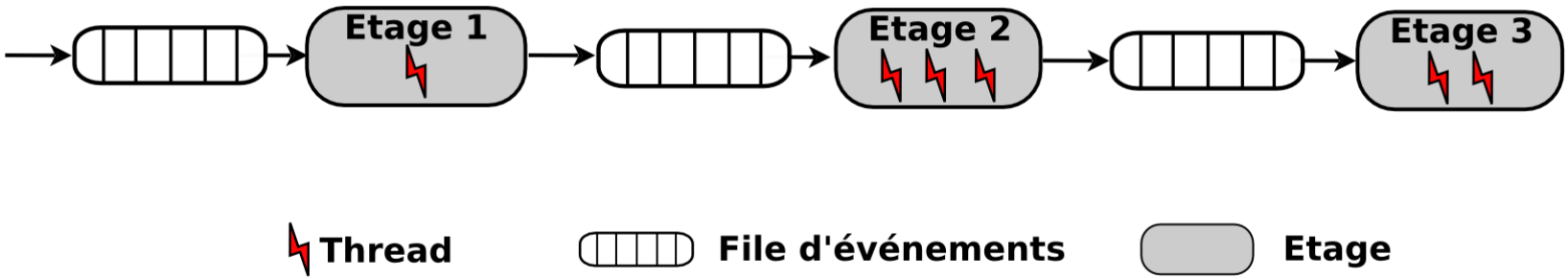
## Performances

- Mauvaises performances à basse charge
- Bonnes performances à haute charge

# Modèle hybride à base d'étages

## Principes

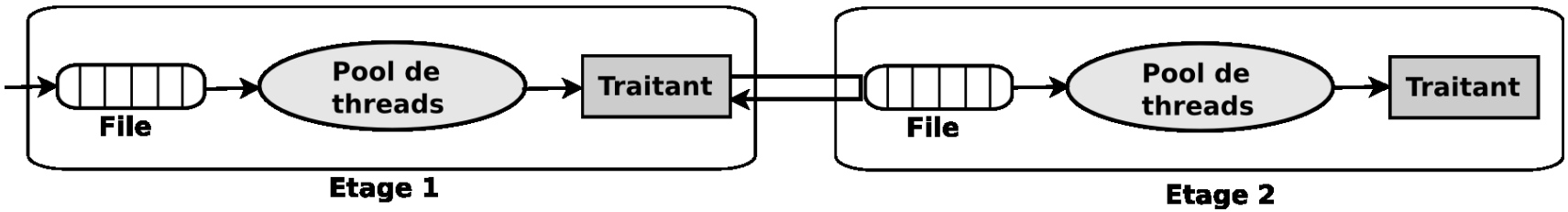
- Modèle fusionnant la programmation par threads et celle par événements
- Le programme est un ensemble d'étages
- Les étages communiquent de manière asynchrone
- Les étages disposent de leur propre pool de threads



# Modèle hybride à base d'étages (2)

## Mise en œuvre

- 3 composants : File, Pool de threads, Traitant
- L'étage connaît la file de l'étage suivant
- Possibilité d'ordonnancement par lot



# Modèle hybride à base d'étages (3)

## Caractéristiques

- Utilisation des multiprocesseurs
- Bonne utilisation des caches grâce à l'ordonnancement par lot
- Possibilité de politiques de qualité de service

## Performances

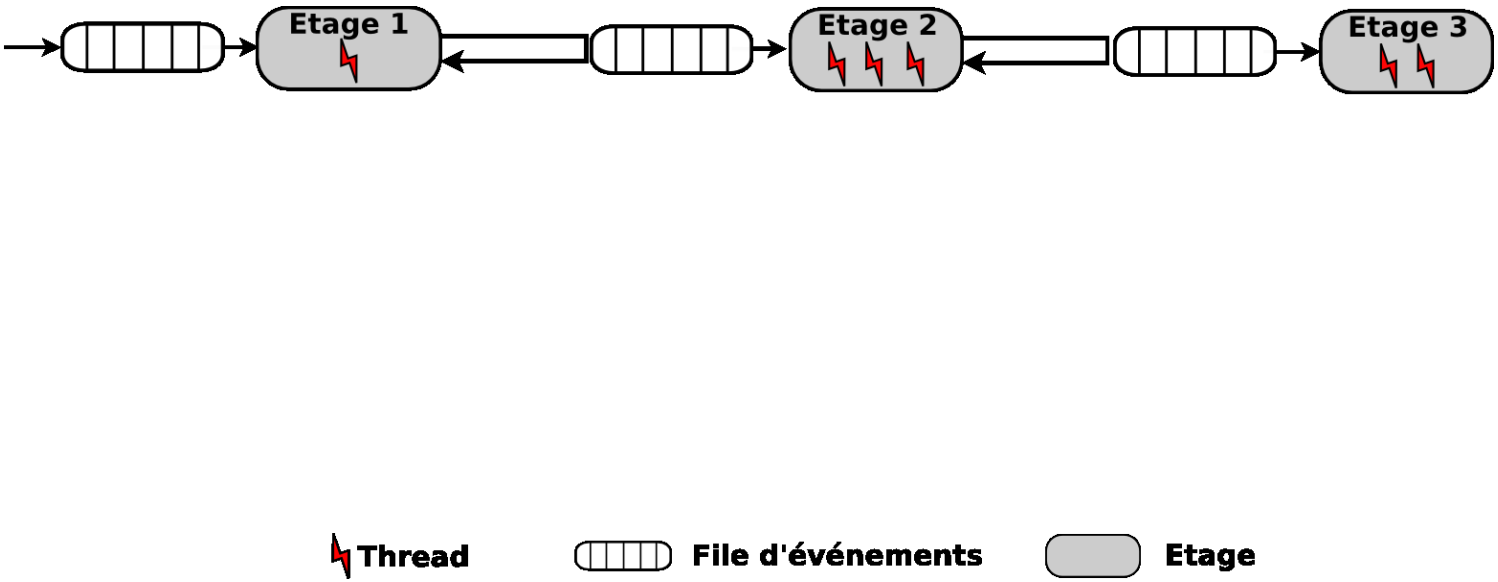
- Mauvaises performances à basse charge (modèle événementiel)
- Bonnes performances à haute charge

# Synthèse

- Modèle à base de threads
  - Bonnes performances à basse charge
  - Mauvaises performances à haute charge
- Modèle à base d'évènements
  - Mauvaises performances à basse charge
  - Bonnes performances à haute charge
- Modèle hybride
  - Identique au modèle événementiel avec plusieurs bonnes propriétés

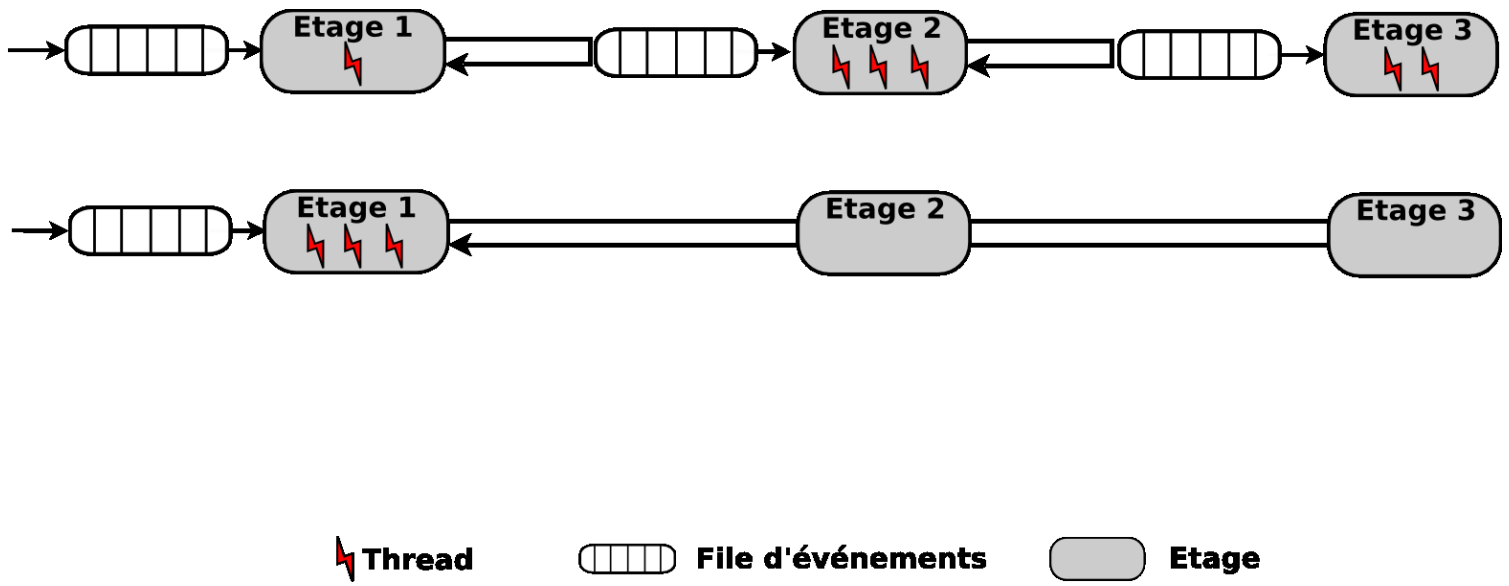
# Contribution

- Un modèle de programmation et une infrastructure d'exécution qui permette un modèle d'exécution :
  - Asynchrone (événementielle)



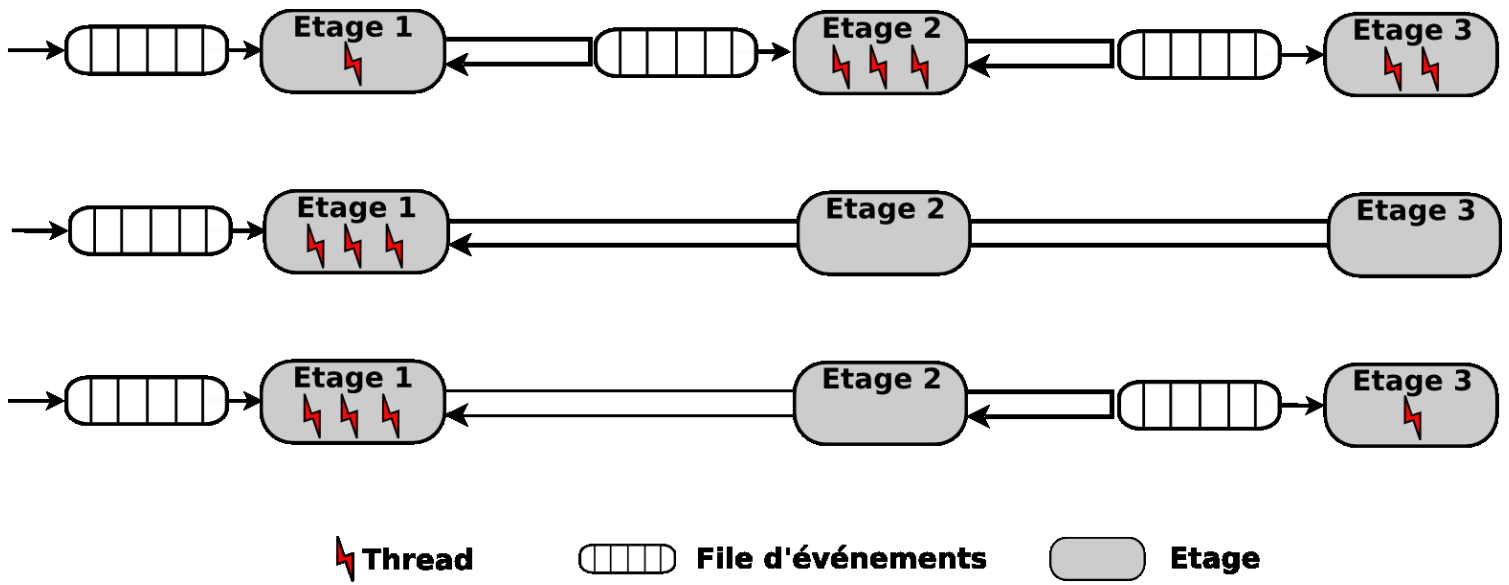
# Contribution

- Un modèle de programmation et une infrastructure d'exécution qui permette un modèle d'exécution :
  - Asynchrone (événementielle)
  - Synchrone (threads)



# Contribution

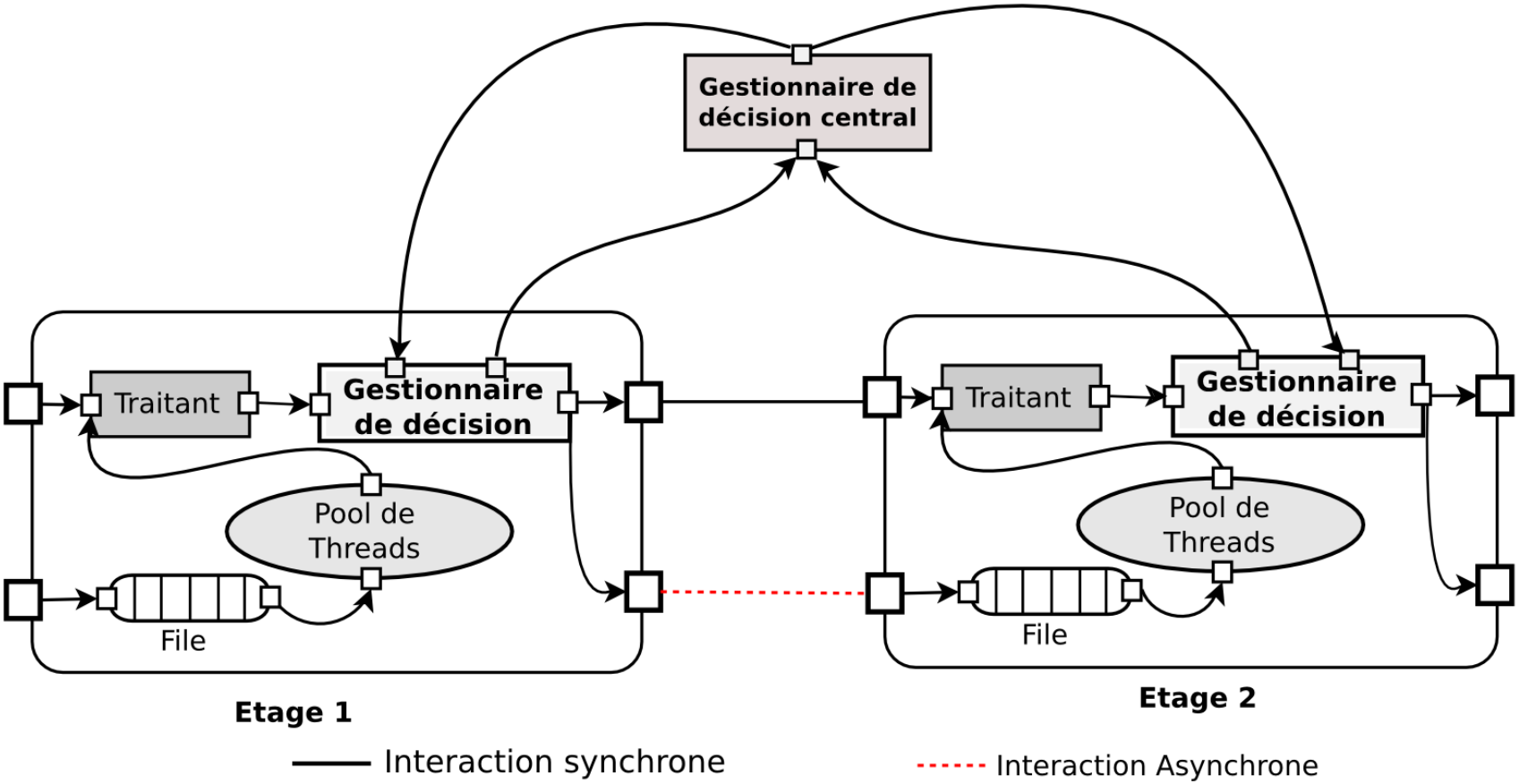
- Un modèle de programmation et une infrastructure d'exécution qui permette un modèle d'exécution :
  - Asynchrone (événementielle)
  - Synchrone (threads)
  - Partiellement synchrone



# Contribution : détail

- Architecture permettant d'avoir les différents modèles d'exécution (synchrone et asynchrone)
- Heuristiques de décision pour le changement de modèle
- Validation de l'architecture et de certaines de ces heuristiques sur un exemple simple

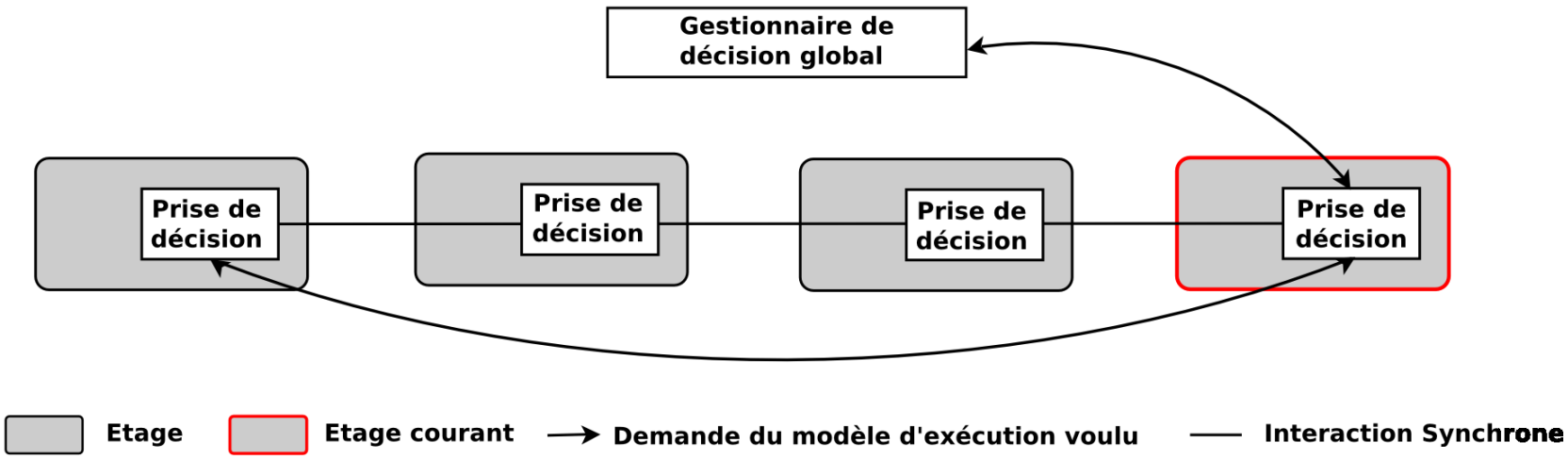
# Architecture



# Prise de décision

## Principes

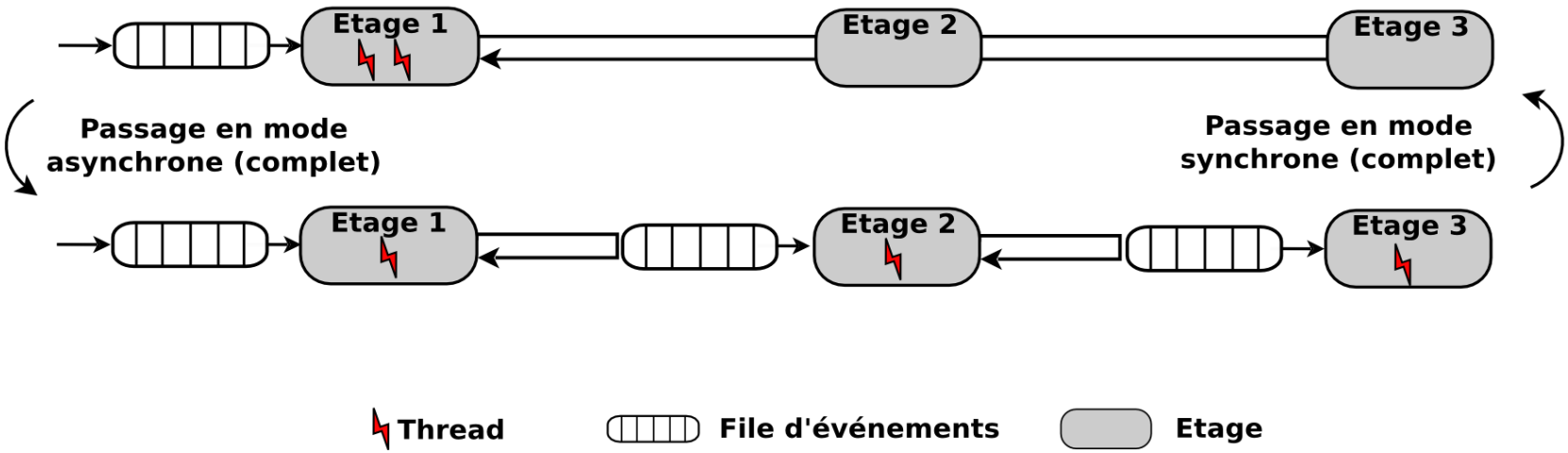
- Un gestionnaire de décision local à chaque étage
- Un gestionnaire de décision global
- Prise en compte de l'avis du possesseur du thread ainsi que de l'étage courant
- Le choix d'une suite d'exécution asynchrone est prioritaire sur un choix synchrone



# Prise de décision (2)

## Problèmes à résoudre

- Répartition des threads dans les étages lors d'un changement de modèle
- Gestion des opérations bloquantes



# Prise de décision (3)

## Heuristiques étudiées

- Débit (nombre de requêtes traitées par seconde) entrant / sortant
- Nombre d'éléments dans les files d'attente des étages

## Heuristiques envisageables

- Nombre de changements de contexte
- Nombre de défauts de cache
- Présence d'opérations bloquantes

# Implantation

## Basée sur Sandstorm

- Implantation en Java du modèle défini par SEDA
- Environ 25000 lignes de code
- Environ 200 classes

## Modifications apportées à Sandstorm

- Ajout des fonctions de changement de modèle d'exécution
- Ajout des gestionnaires de décision
- Modification de la gestion des E/S réseau
- Modification de la gestion des threads

$\Rightarrow$ Portent sur environ 1/3 du code de Sandstorm

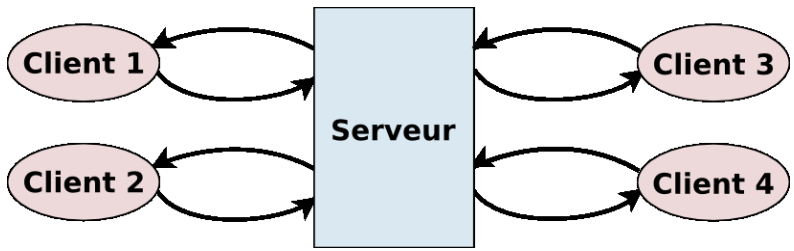
# Évaluation de performances

## Objectifs

- Vérifier les différences de performances entre le mode synchrone et le mode asynchrone
- Valider une première solution d'adaptation dynamique

## Architecture

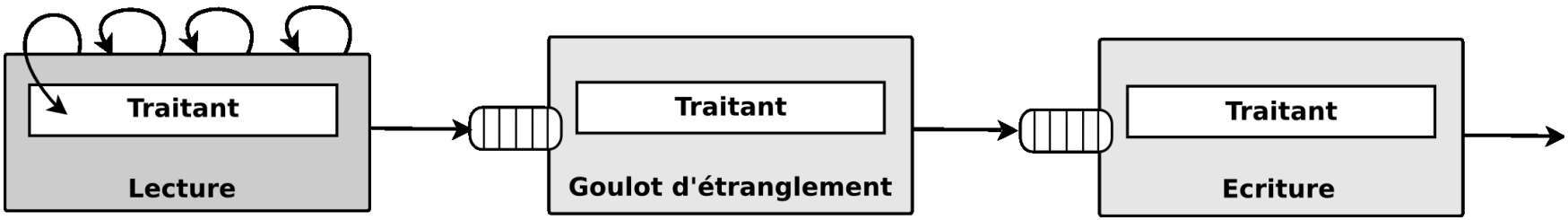
- Architecture Clients / Serveur
- Injection de charge en boucle fermée
- Variation du nombre de clients simultanés
- Calcul du débit et de la latence pour chaque expérience



# Architecture du serveur

## Caractéristiques

- Reçoit une requête et renvoi une réponse de même taille
- Serveur simplifié composé de 3 étages : Lecture, Écriture, Goulot d'étranglement
- Utilisation d'E/S réseau synchrones
- Configuré en mono-processeur



# Conditions de test

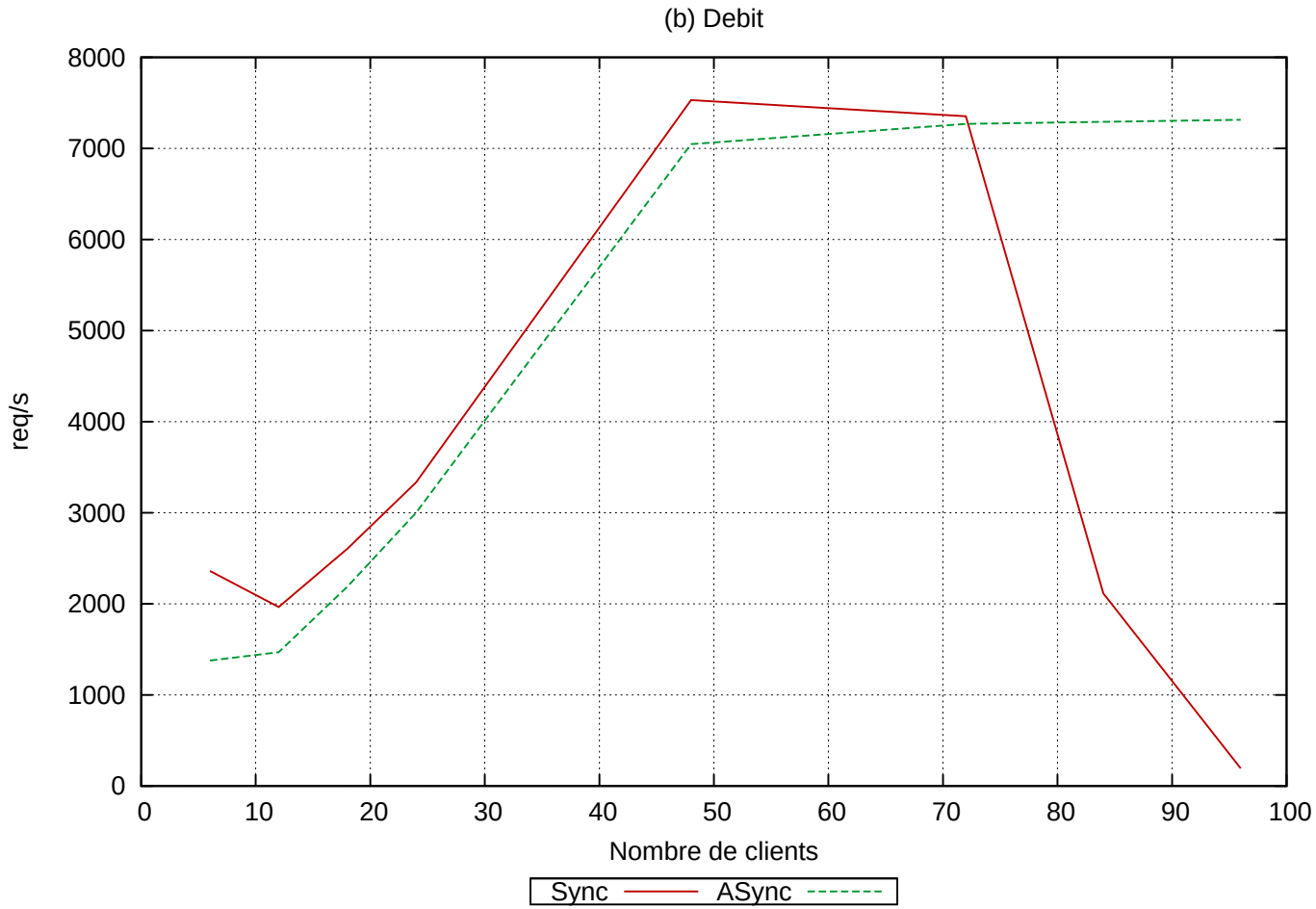
## Conditions de tests

- 7 machines de tests (1 serveur, 6 clients)
- Plusieurs clients par machine

## Configuration des machines

- 2 processeurs Intel Xeon Hyperthreading (1.8 GHz)
- Carte réseau 1Gb/s
- Noyau Linux 2.6.20

# Résultats sans adaptation dynamique

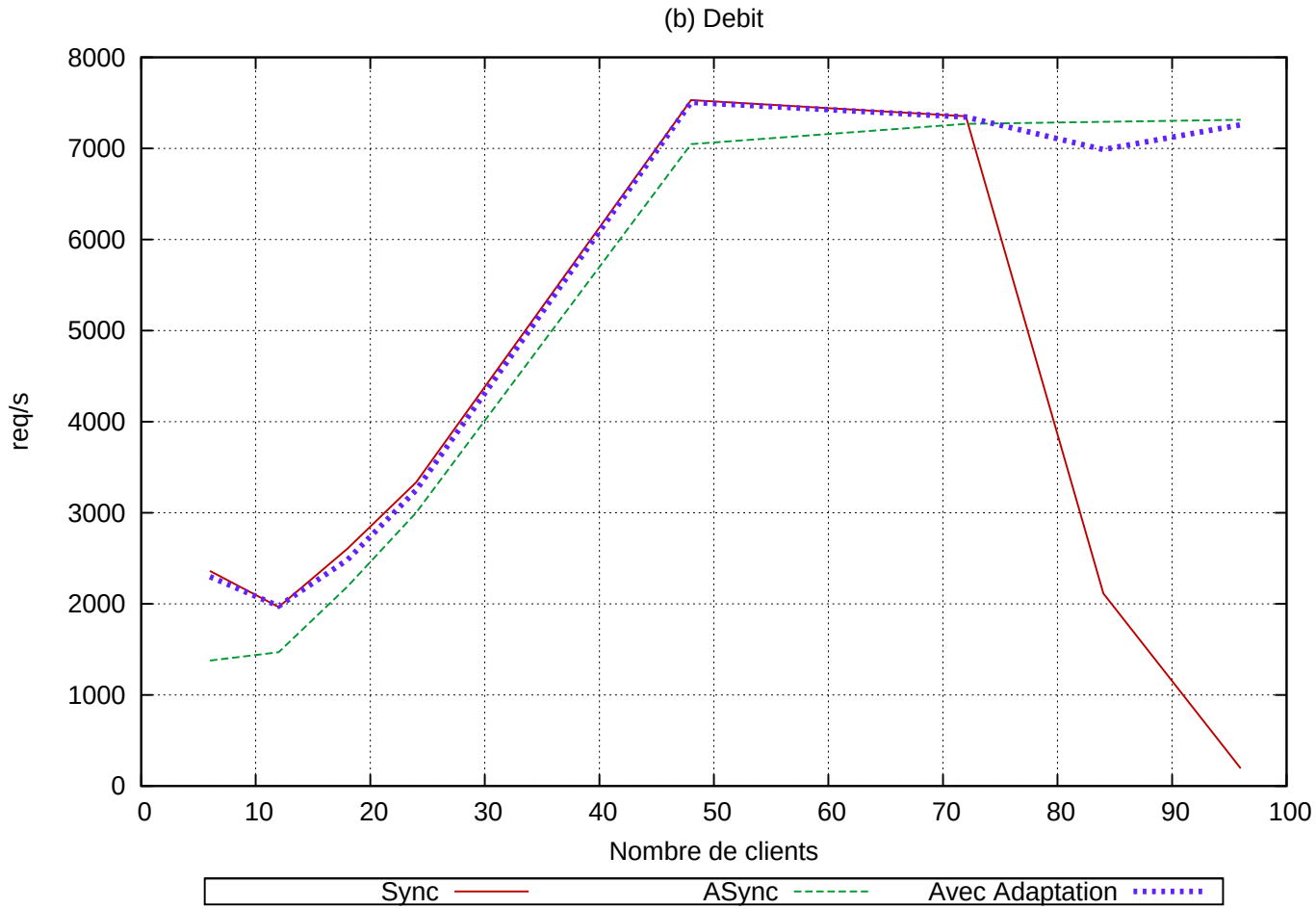


# Adaptation dynamique du modèle d'exécution

## Mise en œuvre

- Choix entre les modes d'exécution totalement synchrone et totalement asynchrone (initialement synchrone)
- Heuristiques utilisées
  - Débit (réseau) entrant / sortant
  - Taille de la file des messages entrants
- Seuils de basculement réglés en fonction des observations précédentes
- Décision prise en fonction de tendances (N valeurs en dessous/dessus du seuil)
- Calcul périodique des heuristiques

# Résultats avec adaptation dynamique



# Conclusion

## Problématique

- Modèle de programmation et modèle d'exécution performant

## Travaux

- Proposition d'une architecture permettant un changement dynamique de modèle d'exécution
- Vérification de l'intérêt d'un changement dynamique de modèle d'exécution
- Proposition d'une solution pour avoir un changement autonome de modèle d'exécution

## Résultats

- Tests menés dans le domaine des serveurs de données
- Résultats encourageants : gains en débit jusqu'à 70% à basse charge et 700% à haute charge

# Perspectives

## Études futures

- Effectuer un implantation en C de l'architecture proposée
- Améliorer l'évaluation de performances (taille des requêtes, différentes injections de charge, ...)
- Étudier les heuristiques proposées mais non traitées
- Étudier la stratégie d'exécution partiellement synchrone
- Étudier l'influence des architectures parallèles

# Questions ?