Reconfiguration dynamique et tolérance aux fautes pour les applications distribuées à grande échelle

Xavier Besseron

Équipe MOAIS
Laboratoire d'Informatique de Grenoble
Université de Grenoble

DR Franck Cappello	INRIA	Rapporteur
Pr Christophe Cérin	LIPN	Rapporteur
DR Frédéric Desprez	INRIA	Examinateur
Pr Jacques Mossière	INPG	Examinateur
Pr Denis Trystram	INPG	Directeur de thèse
CR Thierry Gautier	INRIA	CoDirecteur de thèse

28 avril 2010

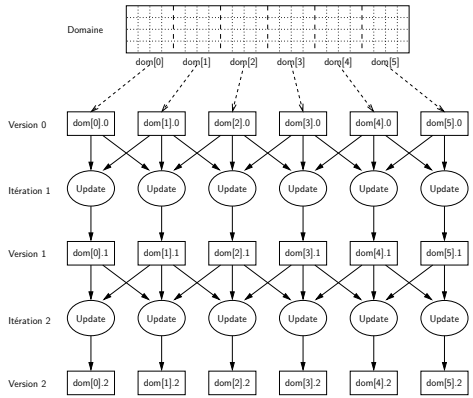
Plan

Calcul haute performance

Applications parallèles

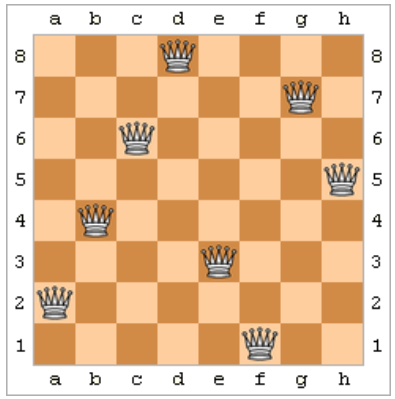
Applications numériques

- Type « décomposition de domaine »
- Régulière ⇒ ordonnancement basé sur un partitionnement de données
- Itérative ⇒ répétition du même squelette de calcul
- Exemple : Méthode de Jacobi



Applications en optimisation combinatoire

- Type recherche arborescente
- Irrégulière ⇒ ordonnancement à la volée
- Exemple : Problème des N-Reines



Plateformes d'exécution

Grilles de calcul

- Ensemble hétérogène de grappes distantes
- Réseau longue distance (*e.g.* Internet)
- Exemples :
  - Aladdin–Grid'5000 (+ de 5000 cœurs)
  - Grille EGEE

Super calculateurs

- Ensemble homogène de nœuds connectés par un réseau rapide
- Très grand nombre de cœurs
- Hétérogénéité au sein des nœuds : CPU, GPU, Processeur Cell, ...
- Exemples :
  - 2008 : RoadRunner (LANL ; 13 824 cœurs Opteron + 116 640 cœurs Cell)
  - 2011 : BlueWaters à Urbana-Champaign (UIUC, NCSA, NSF ; + de 200 000 cœurs)

Caractéristiques : dynamicité et défaillances

Réservations des machines, maintenance

$\Rightarrow$ Variations du nombre de machines disponibles

Ressources partagées

$\Rightarrow$ Variations de la charge des processeurs et de l'utilisation du réseau

Très grand nombre de composants

$\Rightarrow$ Forte probabilité de panne

Objectifs

Calcul haute performance

⇒ Exploiter efficacement ces plateformes d'exécution

⇒ Limiter le temps/calcul perdu à cause des défaillances

Solutions

- Adaptation dynamique de l'application
  - aux variations de l'environnement d'exécution
- Tolérance aux fautes

Adaptation dynamique

Objectif

Adapter l'application pour la rendre plus performante à l'environnement d'exécution et à ses changements

Observation $\longrightarrow$ Décision $\longrightarrow$ Reconfiguration

Reconfiguration

Modifier la configuration de l'application sans entraver son fonctionnement

$\Rightarrow$ Garantir la cohérence de la reconfiguration

Cohérence mutuelle de la reconfiguration dynamique

Pour faire de la reconfiguration dynamique, il faut garantir la cohérence mutuelle.

But : Garantir un état sain entre les entités participant à la reconfiguration

Cohérence mutuelle basé sur [Goudarzi99]

Des entités sont dans des états mutuellement cohérents si elles ont la même perception du résultat de leur interaction (*i.e.* réussie ou échouée).

Réalisation logicielle (dans le cadre du HPC)

- AFPAC [Buisson06] (applications SPMD) → même endroit du calcul
- ASSIST [Vanneschi07], AMPI [Huang07] → barrières globales
- PCL [Ensink04] → graphe statique de tâches

Tolérance aux fautes

Algorithmes tolérants aux fautes (ABFT)

L'algorithme corrige lui-même les erreurs en ajoutant de la redondance dans le calcul ou les données

- Corrige un taux limité d'erreurs
- Intrinséquement lié à l'algorithme (algèbre linéaire généralement)

Tolérance aux fautes par réplication

Utiliser des copies multiples de processus ou de composants

- Tolère seulement un nombre fixé de pannes
- Consomme des ressources de calcul pour la réplication

Tolérance aux fautes par mémoire stable

Mémoire stable = moyen de conserver les informations nécessaires à la reprise

1. Sauvegarder régulièrement des informations sur la mémoire stable
2. En cas de panne, reconstruire un état global cohérent de l'application

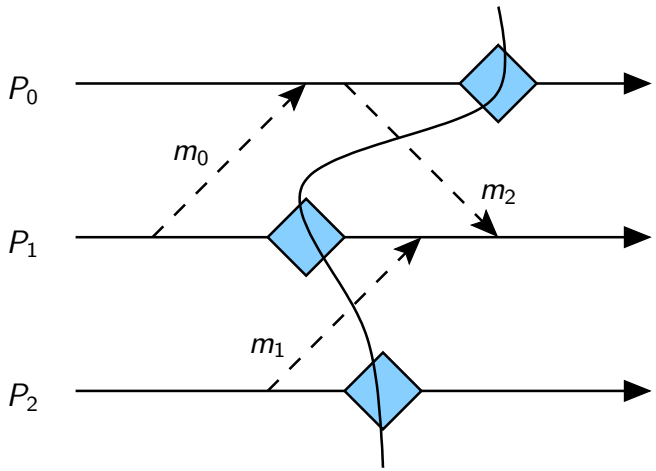
État global cohérent pour la tolérance aux fautes

État global d'une application distribuée

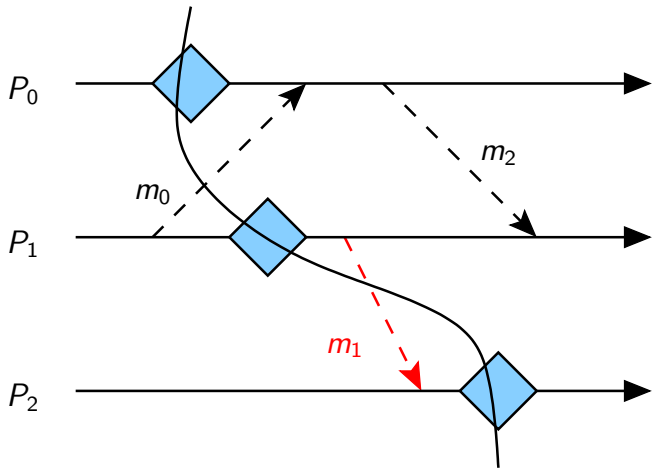
- états locaux de tous les processus
- états de tous les canaux de communication

État global cohérent [Chandy85]

Un état global cohérent est un état qui peut se produire durant une exécution correcte (*i.e.* sans panne) de l'application.



État global cohérent



État global incohérent

Techniques de tolérance aux fautes par mémoire stable

Deux techniques de tolérance aux fautes par mémoire stable :

Journalisation

- Les informations sauvegardés sont les événements non déterministes
- Logiciels : MPICH-V [Bouteiller03, Lemarinier04], Charm++ [Huang04]

Sauvegarde de l'état des processus

- Les informations sauvegardés sont les états des processus
- Lors de la reprise, il faut garantir un état global cohérent
- Trois méthodes pour garantir la cohérence de l'état :
  - À la sauvegarde → Sauvegarde coordonnée
  - À la reprise → Sauvegarde non coordonnée
  - D'après les communications → Sauvegarde induite par les communications
- Logiciels : MPICH-V [Coti06], Charm++ [Zheng04], Kaapi [ThèseJafar06, Jafar09, Besseron08]

Problématique de la thèse

But : Tolérer efficacement les variations de l'environnement et les défaillances

Solutions de 2 domaines proches, 2 types de cohérence :

Reconfiguration dynamique et tolérance aux fautes

Problématique

- Proposer et étudier un mécanisme de reconfiguration dynamique performant

Méthodologie

- Lien entre la cohérence mutuelle et la cohérence globale
- Modélisation de l'exécution d'une reconfiguration
- Voir l'ordonnancement des tâches et les opérations de tolérance aux fautes comme des reconfigurations
- Amélioration du protocole classique de sauvegarde/reprise coordonnée
- Évaluation expérimentale sur grille de calcul

dans le cadre de Kaapi et du modèle graphe de flot de données

Plan

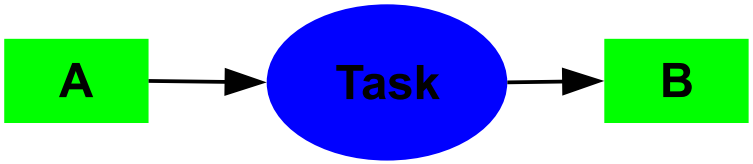
Modèle de programmation Athapascan [PACT'98]

Principe

L'application est décrite sous-forme d'un graphe de flot de données.

Graphe de flot de données

- Données en mémoire partagée
- Tâches de calcul qui accèdent aux données
- Modes d'accès : contraintes d'accès aux données (lecture, écriture, ...)



Propriétés

- Indépendant du nombre de processeurs
- Graphe dynamique déroulé à l'exécution

Moteur d'exécution Kaapi

Exécution du graphe de flot de données

- Le graphe de flot de données est distribué sur tous les processus
- L'ordre d'exécution respecte les contraintes de flot de données
- Tâches sans effet de bord $\Rightarrow$ Exécution déterministe

Graphe de flot de données

- Représentation abstraite de l'état de l'application [ThèseJafar06]
- Causalement connecté à l'exécution

Utilisation du graphe de flot de données

- Ordonnancer les tâches
- Gérer les transferts et les redistributions de données [ThèsePigeon07, Besseron et al. PASCO'07]
- Sauvegarder l'état de l'application [ThèseJafar06]

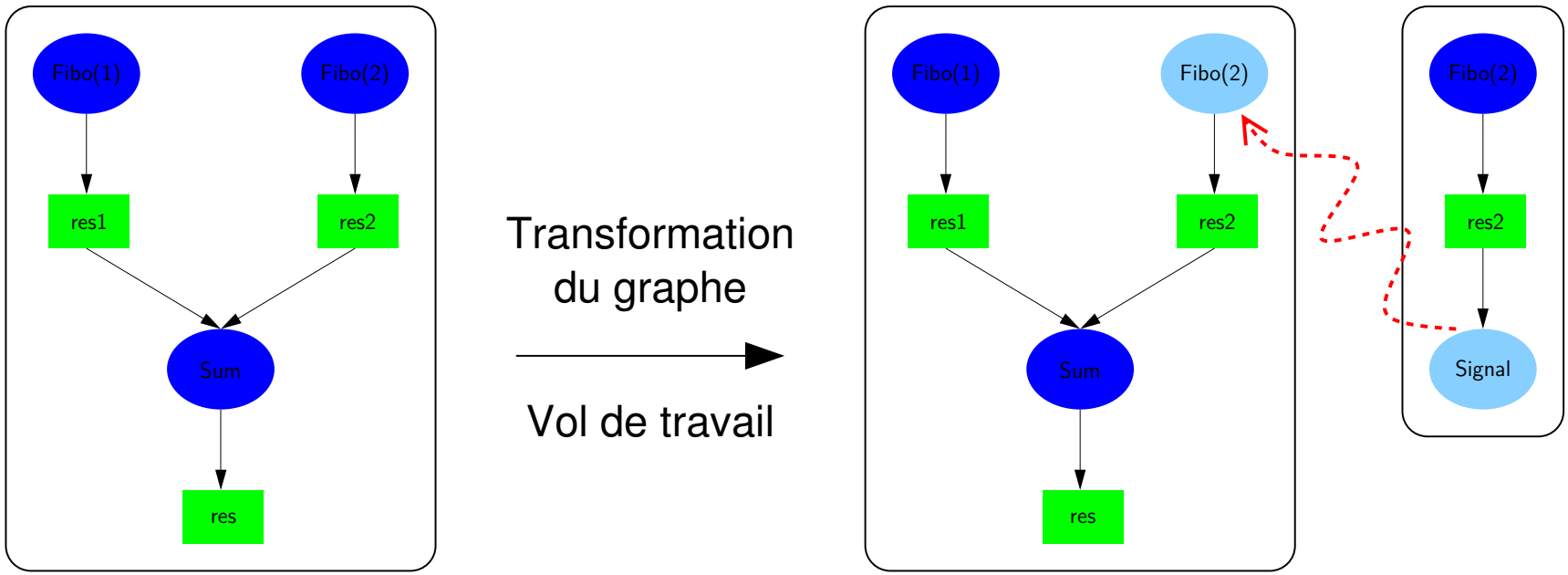
$\Rightarrow$ D'une manière générale : reconfigurer l'application

Ordonnancement par vol de travail

Principe

Lorsqu'un processeur est inactif, il vole du travail à un processeur actif.

Transformation du graphe

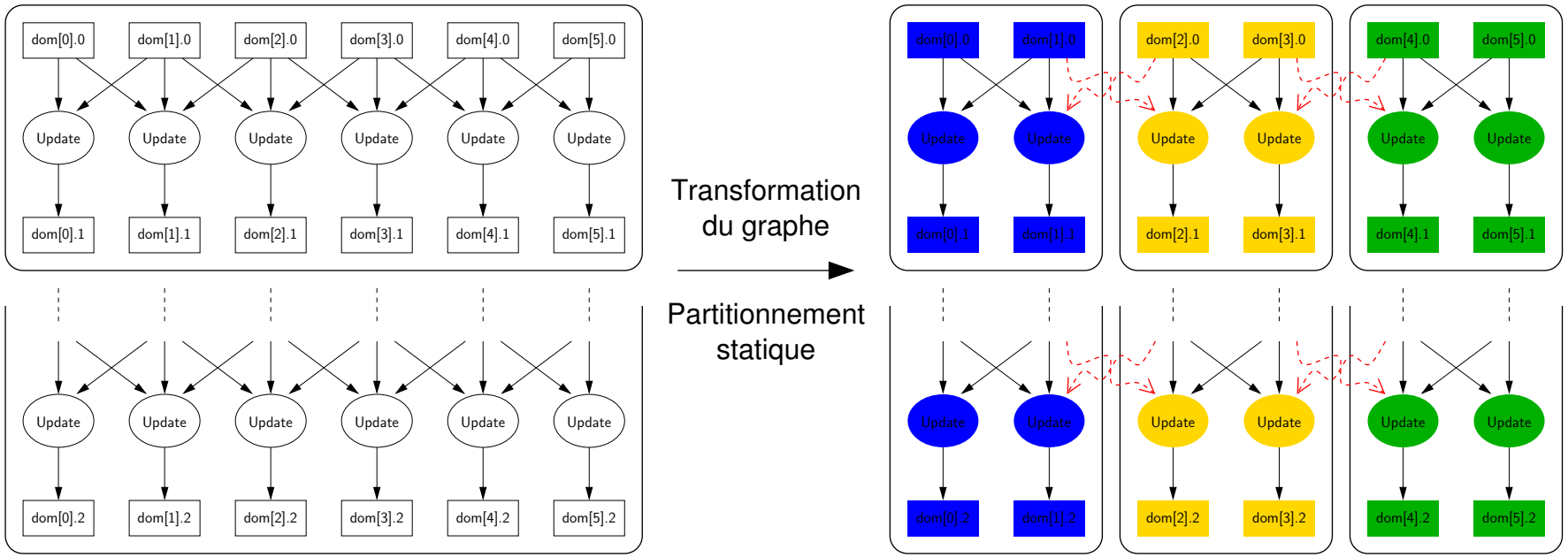


Ordonnancement par partitionnement statique

Principe

- Applications itératives
- Partitionnement du graphe d'une itération (METIS, SCOTCH, ...)
- Génération automatique des tâches de communication

Transformation du graphe



Plan

Reconfiguration dynamique pour Kaapi

Objectif principal

Fournir un mécanisme de reconfiguration dynamique générique et performant

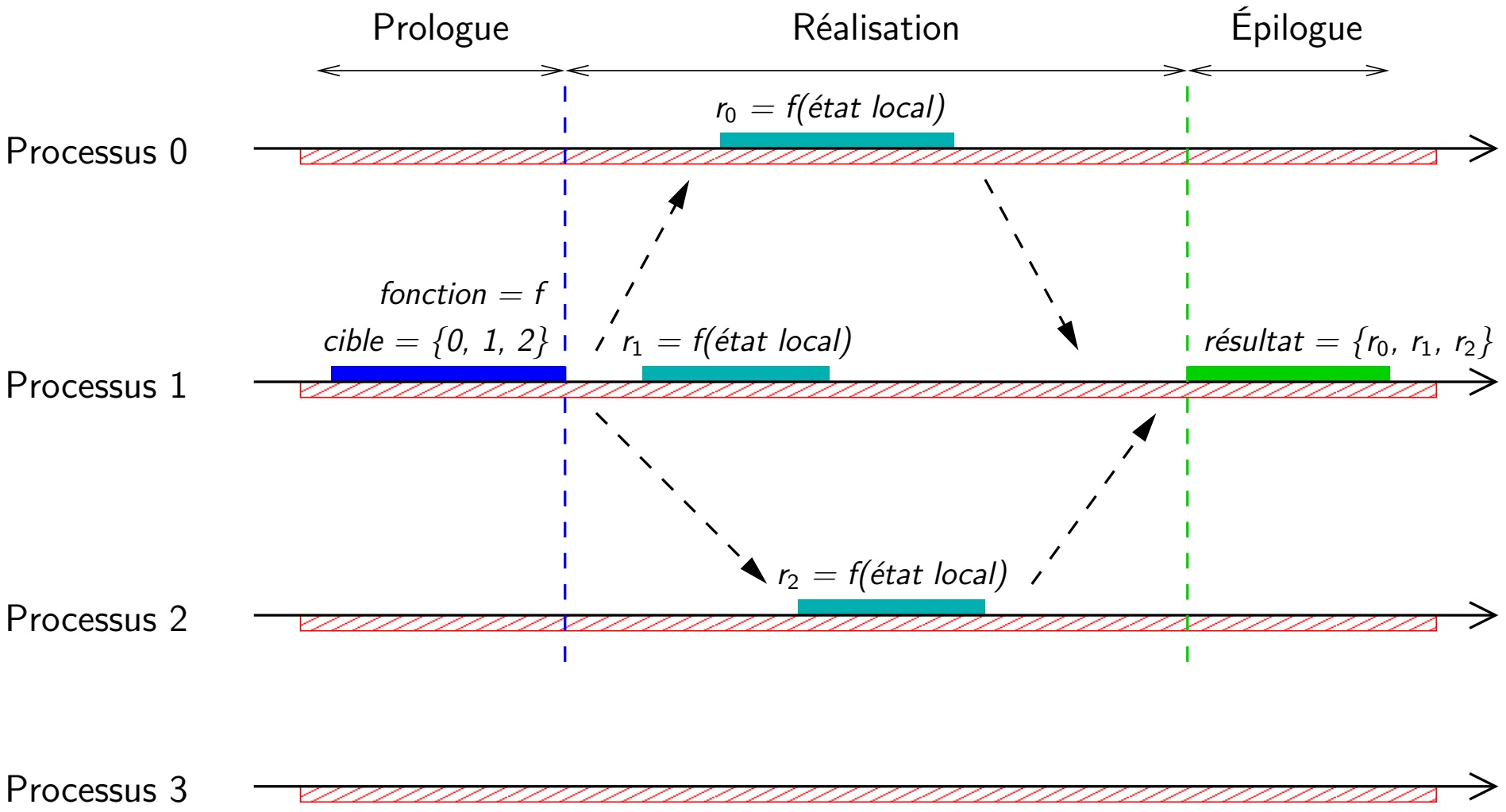
Définitions

- Application = Ensemble de processus communicants
- État d'un processus = Graphe de flot de données
- Reconfiguration = Modification de l'état de l'application
  - Fonction de reconfiguration = Opération sur le graphe de flot de données
  - Cible = Ensemble des processus visés

Exemples

- Ordonnancement : *vol de travail*, *partitionnement statique*, *migration*
- Tolérance aux fautes : *sauvegarde*, *reprise*
- Exploration de l'état : *débogage*, *visualisation*
- Modification du code : *changement de la méthode de calcul*, *du format des données*

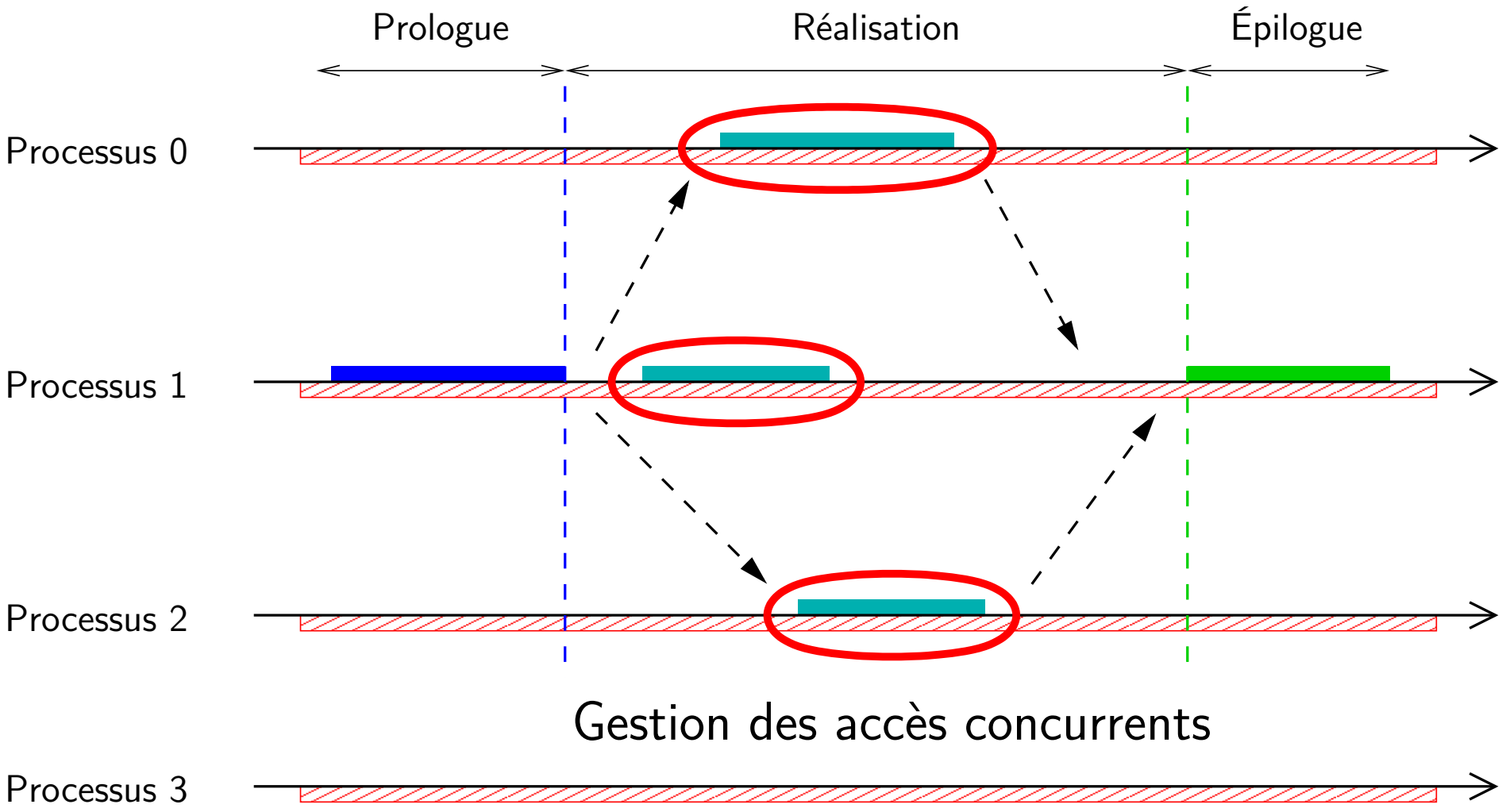
Déroulement d'une reconfiguration dans Kaapi



Problème 1 : Gestion des accès concurrents au sein d'un processus

Problème 2 : Gestion de la cohérence mutuelle entre les processus

Problème 1 : accès concurrents au sein d'un processus



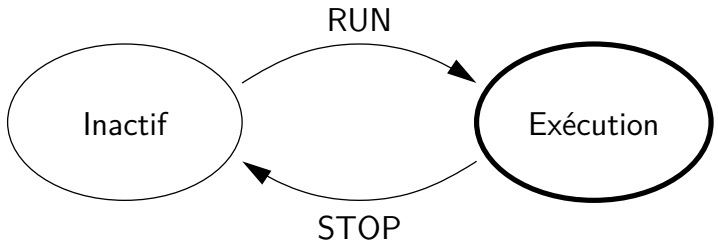
Modélisation du problème des accès concurrents

Objet reconfigurable ou R-objet

- Représentation abstraite = Ensemble de structures de données
- R-objet = Modélisation des changements d'état de ces structures
- Exemple : une tâche du graphe de flot de données Kaapi

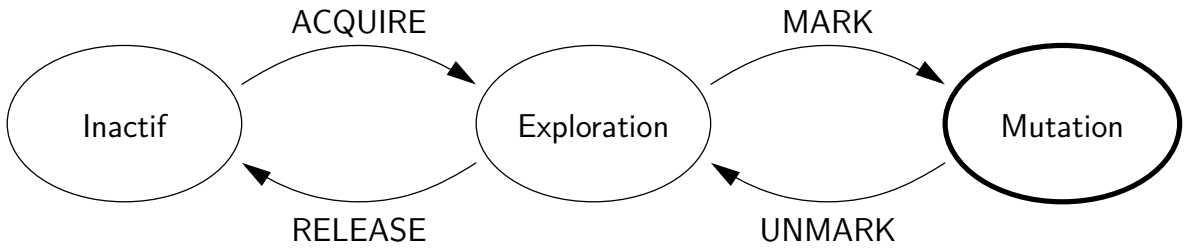
Flot d'exécution (exécute l'application)

États du R-objet :



Flot de reconfiguration (exécute la reconfiguration)

États du R-objet :

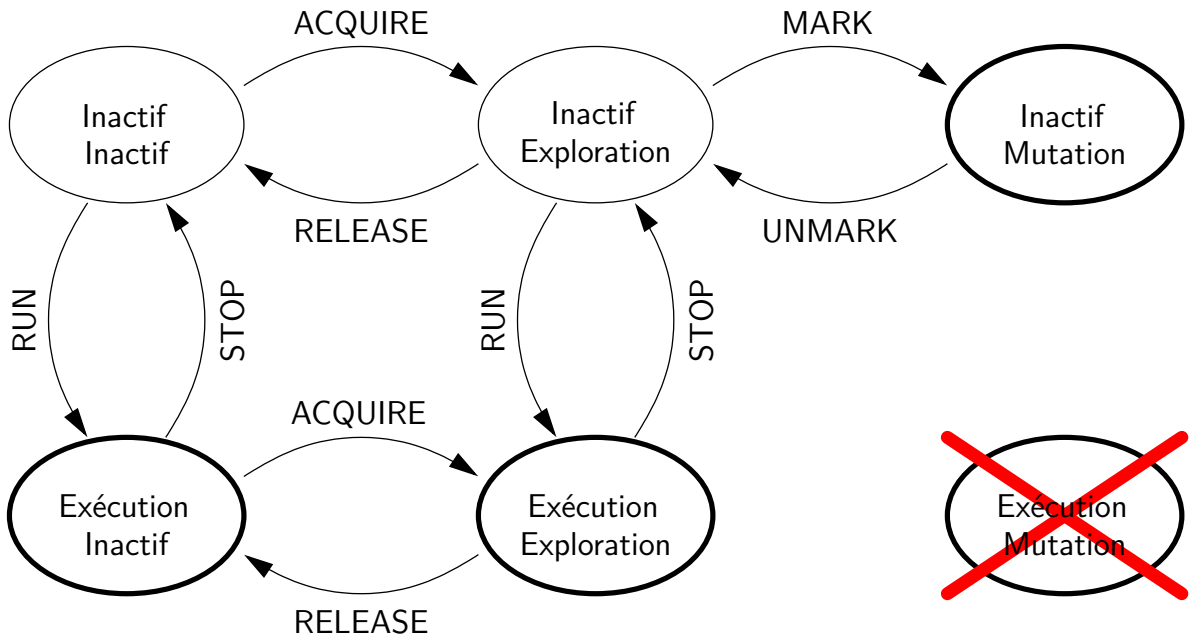


⇒ Garantir l'exclusion mutuelle des états *Exécution* et *Mutation* pour ces deux flots

1re solution : l'exécution concurrente

Principe

Le flot de reconfiguration s'exécute en concurrence du flot d'exécution.



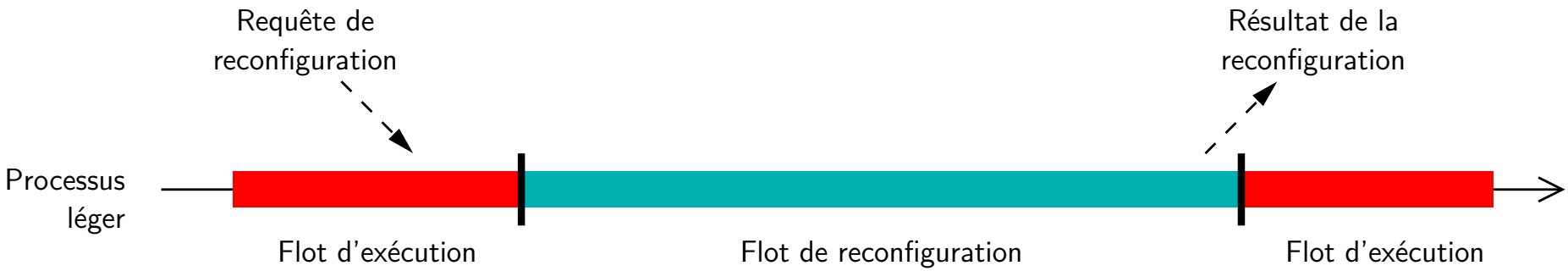
Avantages et inconvénients

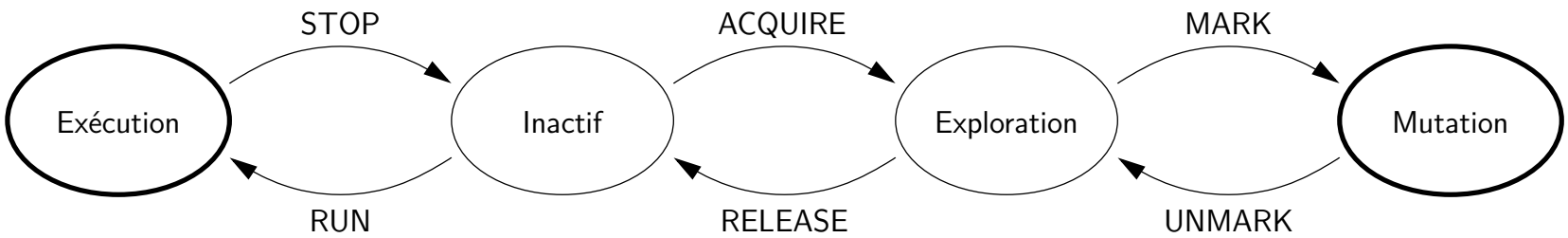
- Permet la reconfiguration sans arrêter tout le calcul
- Nécessite l'utilisation de primitives de synchronisation

2e solution : l'exécution coopérative

Principe

Le même processus léger est utilisé pour exécuter les deux flots.

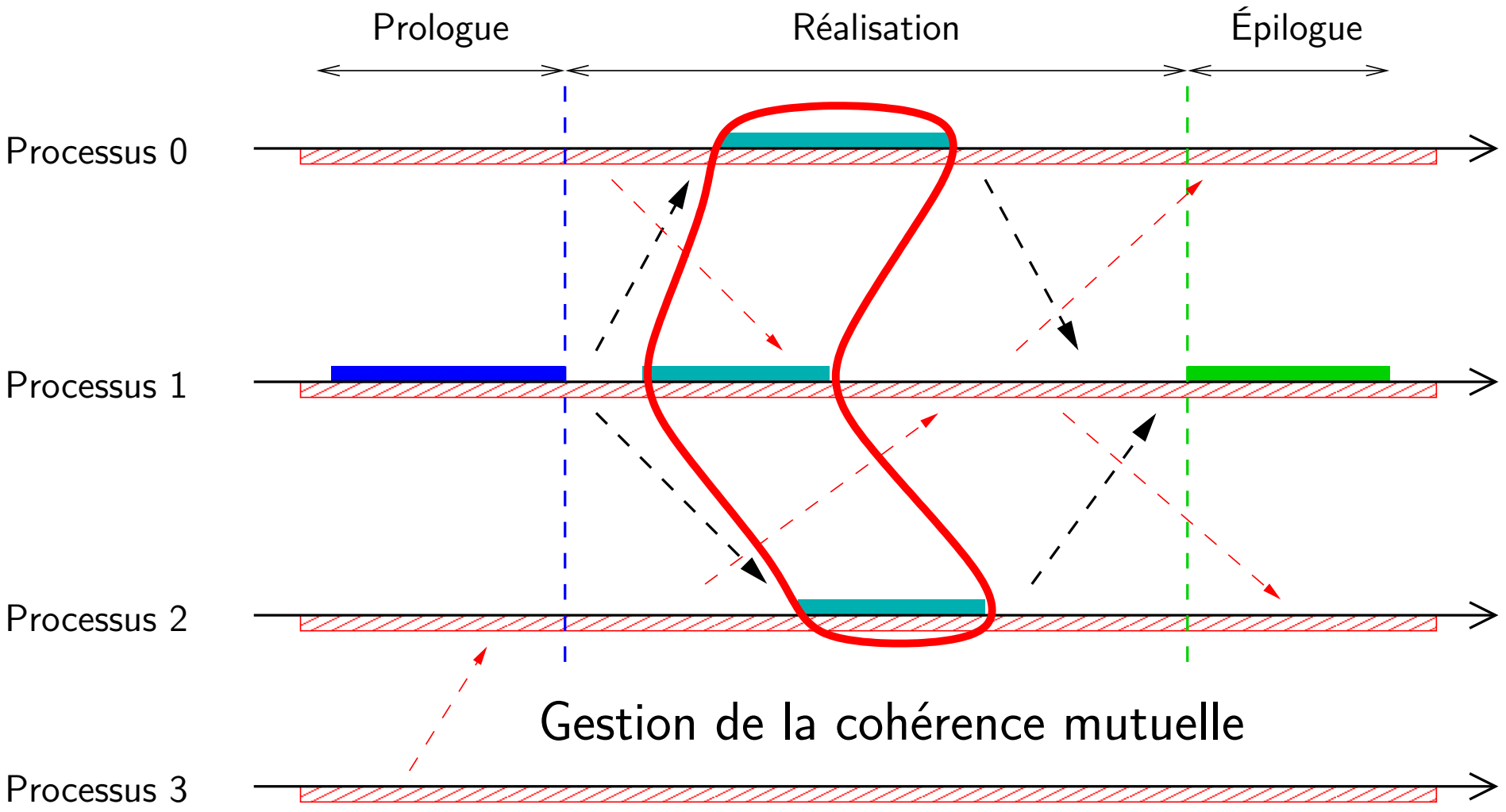




Avantages et inconvénients

- Pas de primitives de synchronisation, lectures/écritures atomiques
- La reconfiguration est retardée jusqu'à la fin de l'exécution d'une tâche

Problème 2 : cohérence mutuelle entre les processus



Lien entre la cohérence mutuelle et la cohérence globale

Rappel : cohérence mutuelle [Goudarzi99]

Un état mutuellement cohérent est un état dans lequel tous les processus ont la même vision des messages qu'ils ont échangés (émis ou non, reçu ou non).

Rappel : cohérence globale (selon Chandy et Lamport [Chandy85])

Un état globalement cohérent est un état dans lequel, si un processus reflète la réception d'un message, alors l'état du processus émetteur reflète l'émission du message.

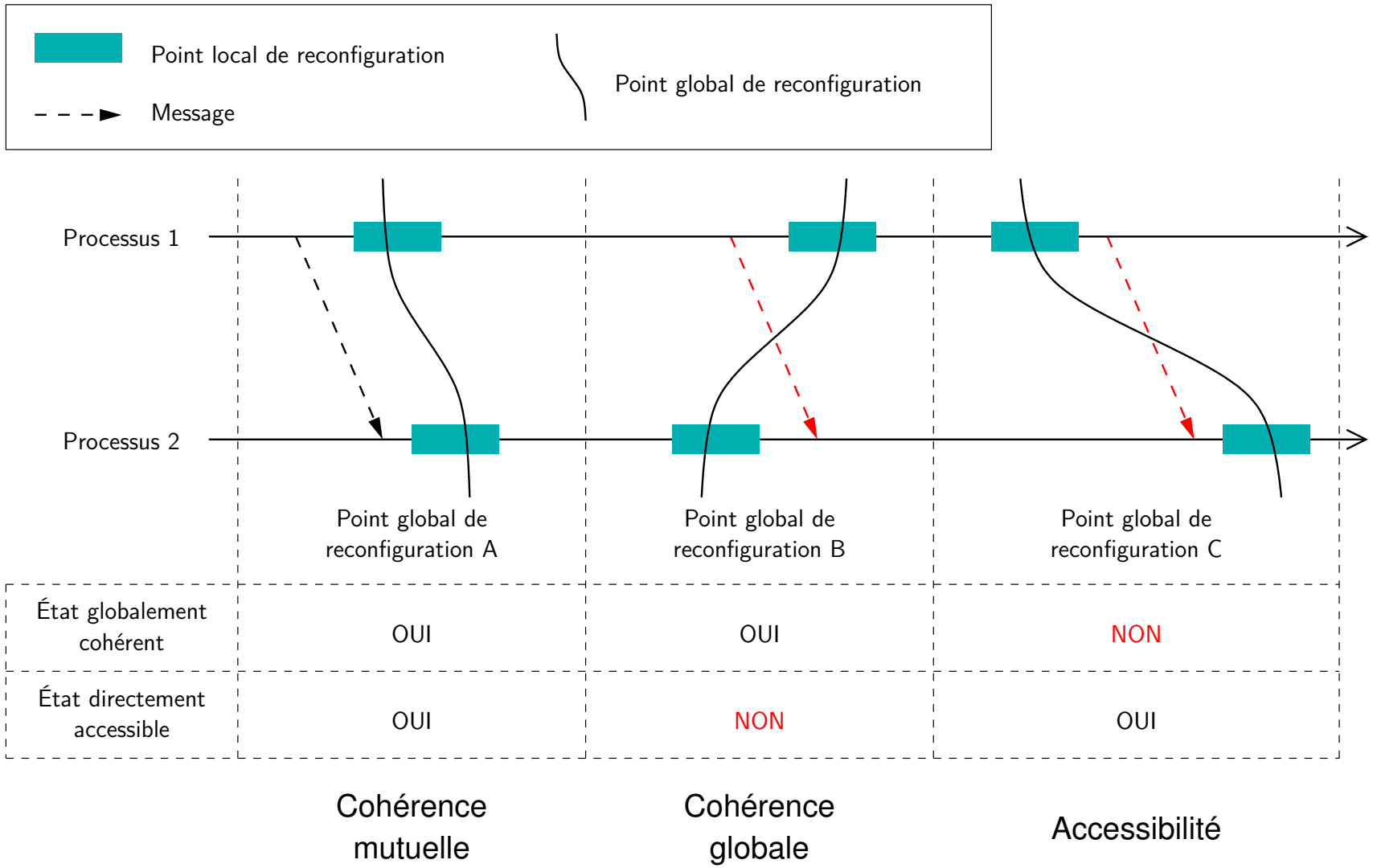
Notion proposée : accessibilité

Un état accessible est un état dans lequel, si un processus reflète l'émission d'un message, alors l'état du processus récepteur reflète la réception du message.

Propriété (démontrée dans le manuscrit)

Accessibilité et Cohérence globale $\Leftrightarrow$ Cohérence mutuelle

Cohérence mutuelle, cohérence globale et accessibilité



Réalisation de la cohérence mutuelle dans Kaapi

Protocole classique

⇒ Nécessite de vider tous les canaux de communication

Chacun des P processus envoie $O(P)$ messages, soit $O(P^2)$ messages au total

Protocole optimisé pour Kaapi

- Le graphe de flot de données indique les communications futures
- On identifie les processus voisins (*i.e.* qui vont potentiellement communiquer)

⇒ Vidage des canaux de communication seulement entre les processus voisins

$O(kP)$ messages échangés au total, avec k = nombre moyen de voisins

Nombre moyen de voisins k

- Dépend de l'application et de l'ordonnancement
- Pour l'application N-Reines avec vol de travail : $k < 2$, $P \approx 1000$
- Pour l'application Jacobi3D avec partitionnement statique : $k \approx 7, P \approx 1000$

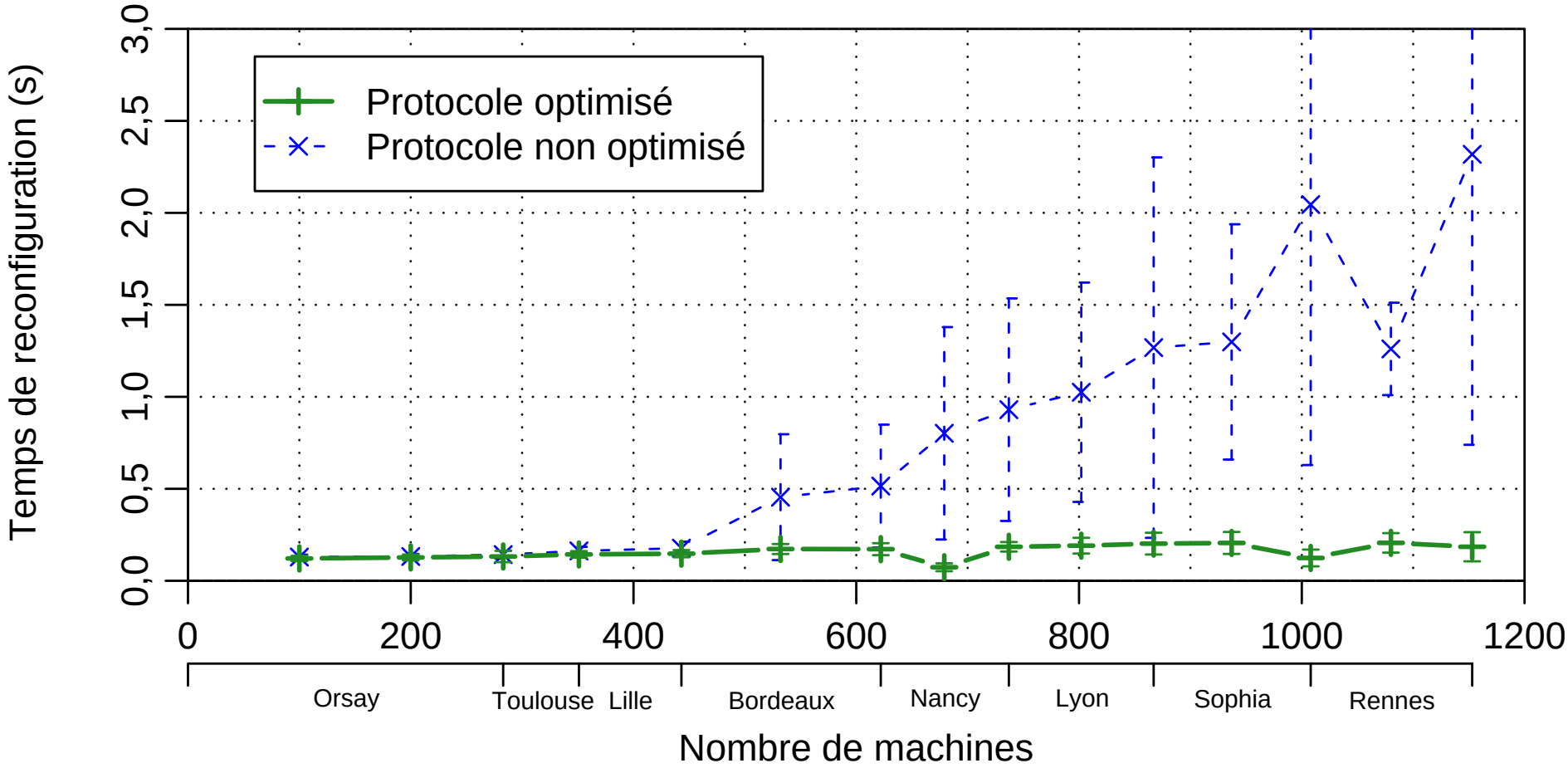
Temps de gestion de la cohérence mutuelle

Application des N-Reines ordonnancée par vol de travail

Fonction de reconfiguration nulle

Jusqu'à 1153 machines de Grid'5000

Temps mesuré : du début du prologue à la fin de l'épilogue sur le processus maitre



Plan

Objectif

Objectif principal

- Améliorer le protocole classique de sauvegarde/reprise coordonnée

Sauvegarde/reprise coordonnée classique

- Régulièrement, les processus sauvegardent leur état de manière coordonnée
  $\Rightarrow$ L'état sauvegardé est un état global cohérent
- En cas de panne, tous les processus redémarrent à partir de la dernière sauvegarde

Sauvegarde et reprise coordonnée pour Kaapi

Hypothèses

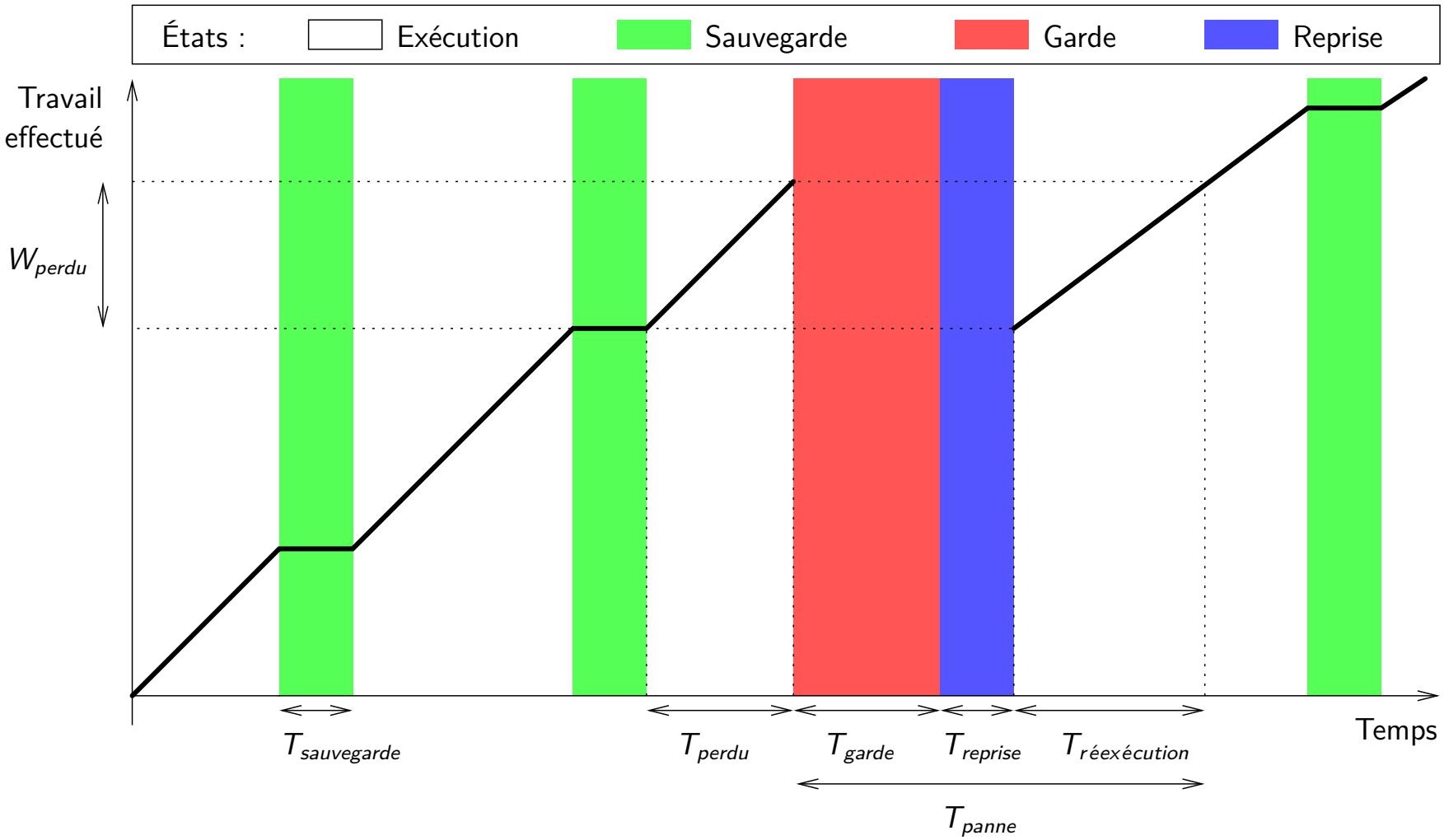
- Pannes franches, ou pannes pouvant être ramenées à des pannes franches
- Pas de machine de rechange

Approche

La sauvegarde et la reprise sont considérées comme des reconfigurations de l'application.

- Cohérence mutuelle pour la sauvegarde et la reprise
  - état de l'application = graphe de flot de données
- Fonctions de reconfiguration considérées :
  - sauvegarde de l'état local (pour la sauvegarde)
  - rechargement de l'état sauvegardé (pour la reprise globale)
  - réordonnancement du graphe (après la reprise)

Modélisation de l'exécution



Reprise globale

Problème du déséquilibre de la charge après la reprise

Hypothèse : pas de machine de rechange

$\Rightarrow$ Un processus non défaillant recharge l'état d'un processus défaillant

$\Rightarrow$ Déséquilibre de la charge (généralement)

Proposition

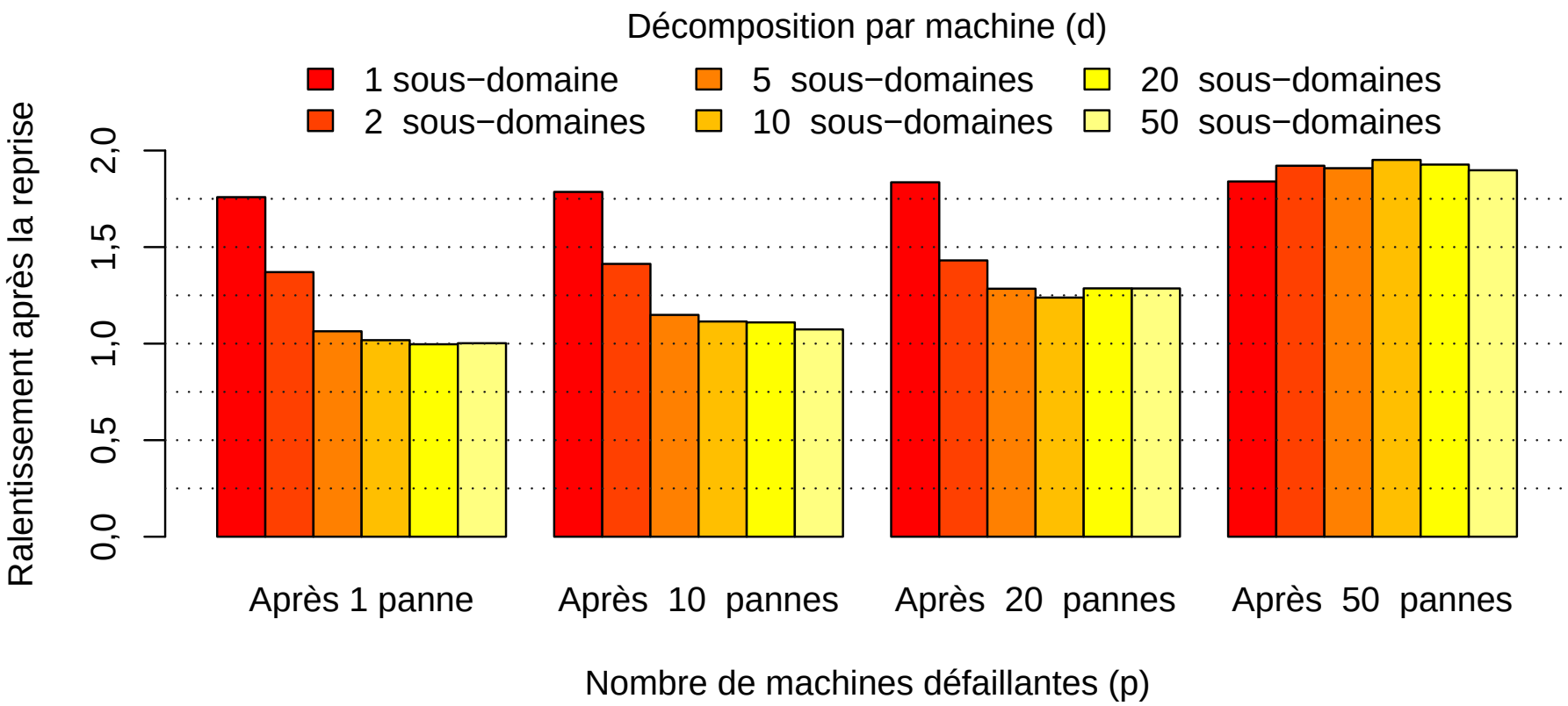
- Réordonnancer le graphe de flot de données à la reprise
- Nécessite d'augmenter le degré de parallèlisme de chacun processus
  - sur-décomposition : découper le calcul en plus de tâches que le nombre de processeurs

Expérimentations

- Méthode de Jacobi sur un problème de Poisson 3D (ANR DiscoGrid)
- Décomposition de domaine, application itérative
- Étude du ralentissement = Temps après panne(s) / Temps avant panne(s)

Ralentissement après la reprise

- 100 machines de la grappe de Nancy de Grid'5000
- Décomposition en d sous-domaines par machine, grand nombre d'itérations
- Exécution sur 100 machines ; p défaillances ; exécution sur $100 - p$ machines
- Temps d'une itération avant la panne $\approx$ 0,4 s
- Basé sur les temps d'exécution moyen de 200 itérations



Proposition : nouveau protocole de reprise partielle

Hypothèses

- La dernière sauvegarde est un état global cohérent
- Exécution déterministe (Kaapi)
- Communications identifiées (Kaapi)

Principe

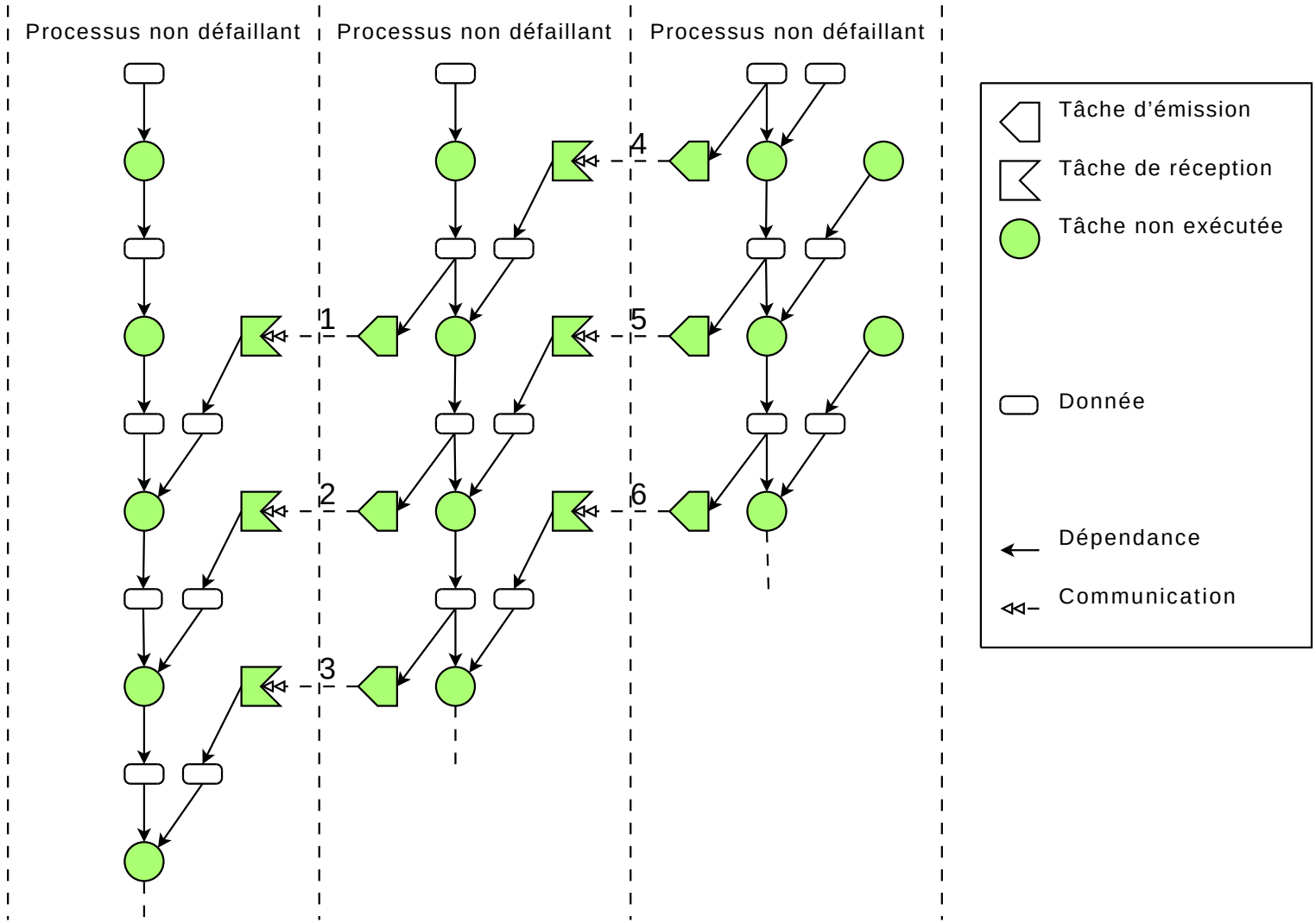
- Les processus défaillants redémarrent à partir de la dernière sauvegarde
- Les communications perdues sont réjouées vers les processus redémarrés
  - pas de journalisation des messages
  - mais réexécution des tâches qui produisent les communications

Déroulement en deux étapes

- Déterminer l'ensemble des tâches nécessaires à la reprise
  - appelé le travail perdu
- Réordonnancer le travail perdu
  - de manière à réduire son temps de réexécution

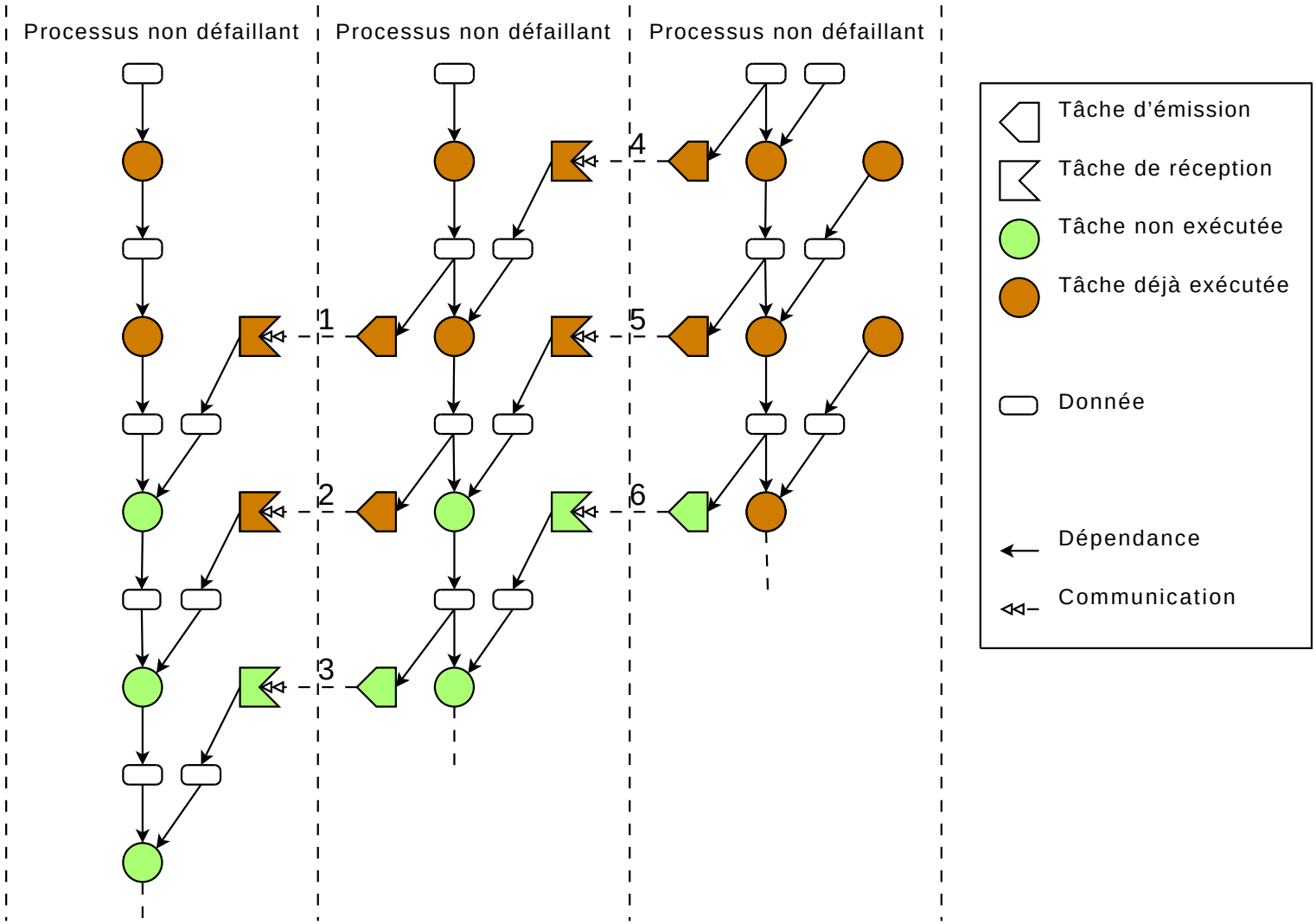
1re étape : identification des tâches à réexécuter

Après la sauvegarde



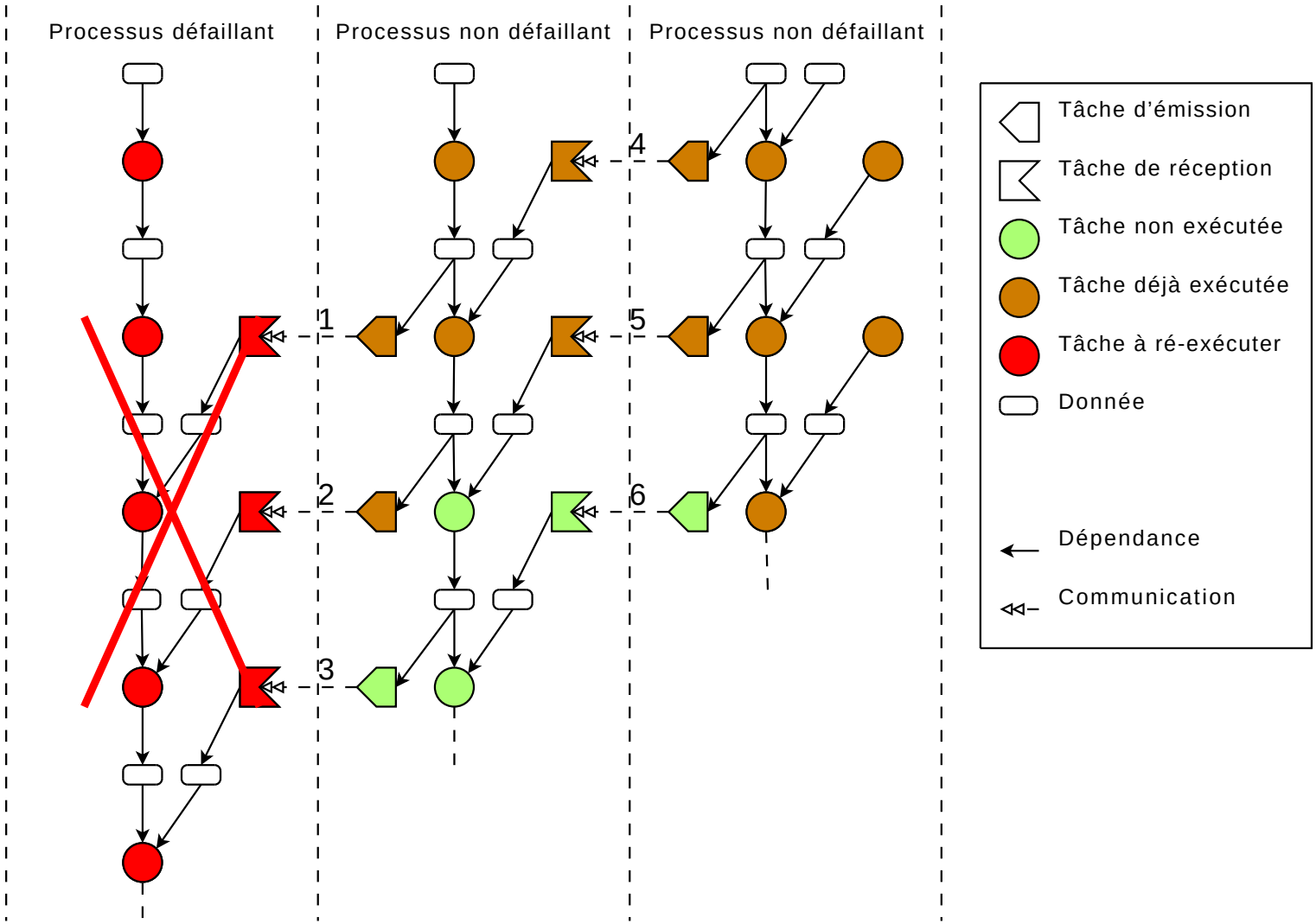
1re étape : identification des tâches à réexécuter

Exécution



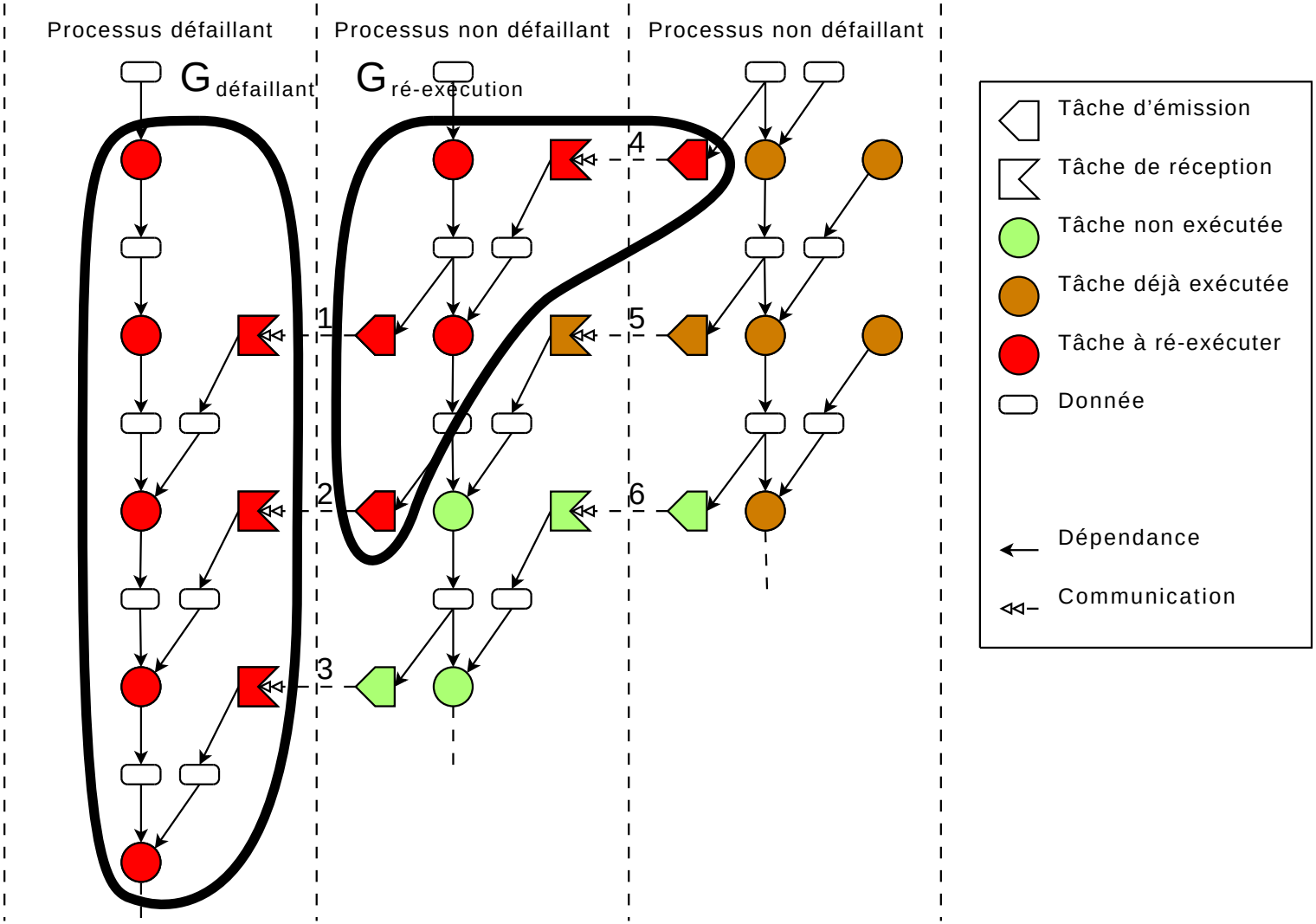
1re étape : identification des tâches à réexécuter

Panne



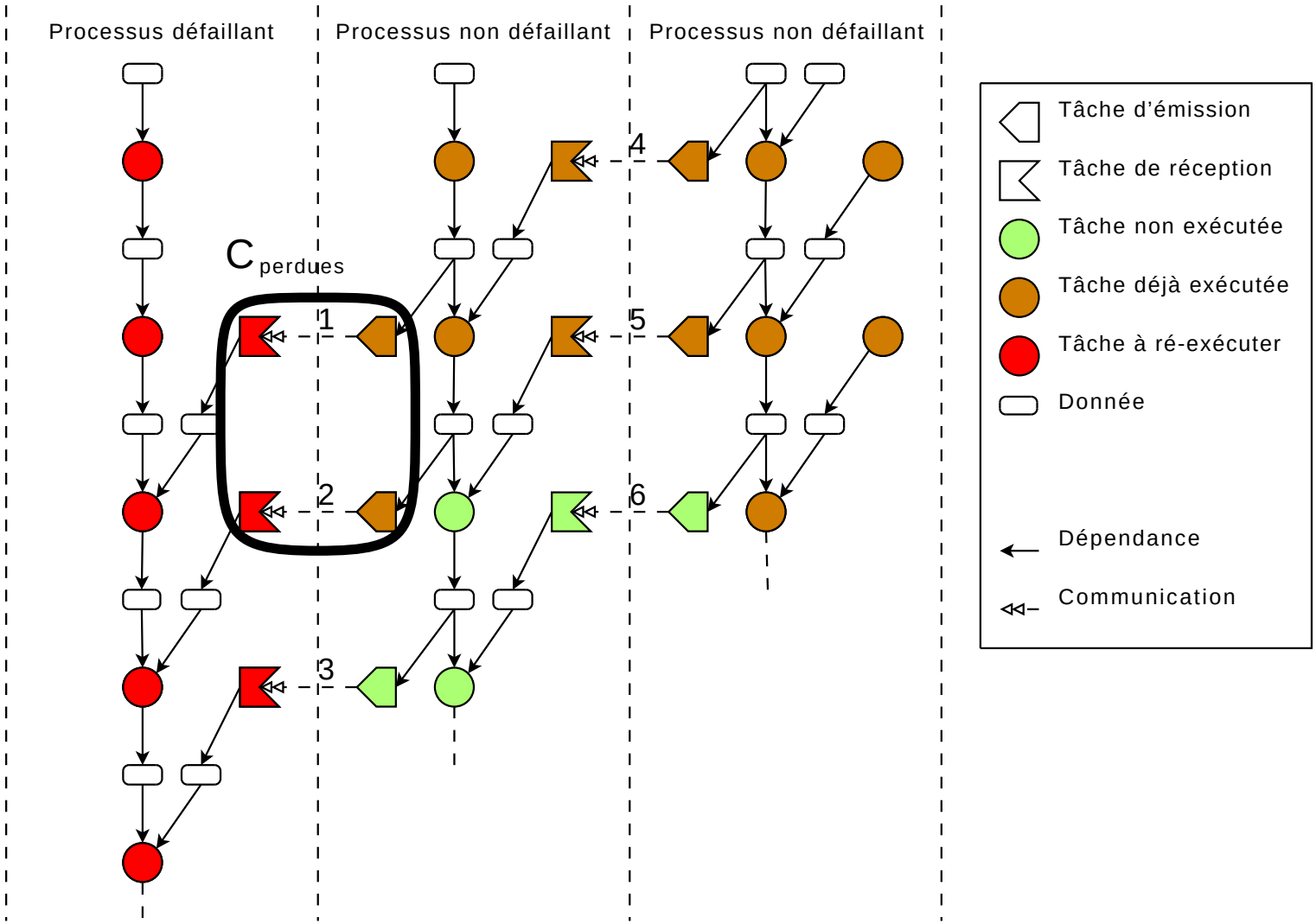
1re étape : identification des tâches à réexécuter

Tâches à réexécuter



1re étape : identification des tâches à réexécuter

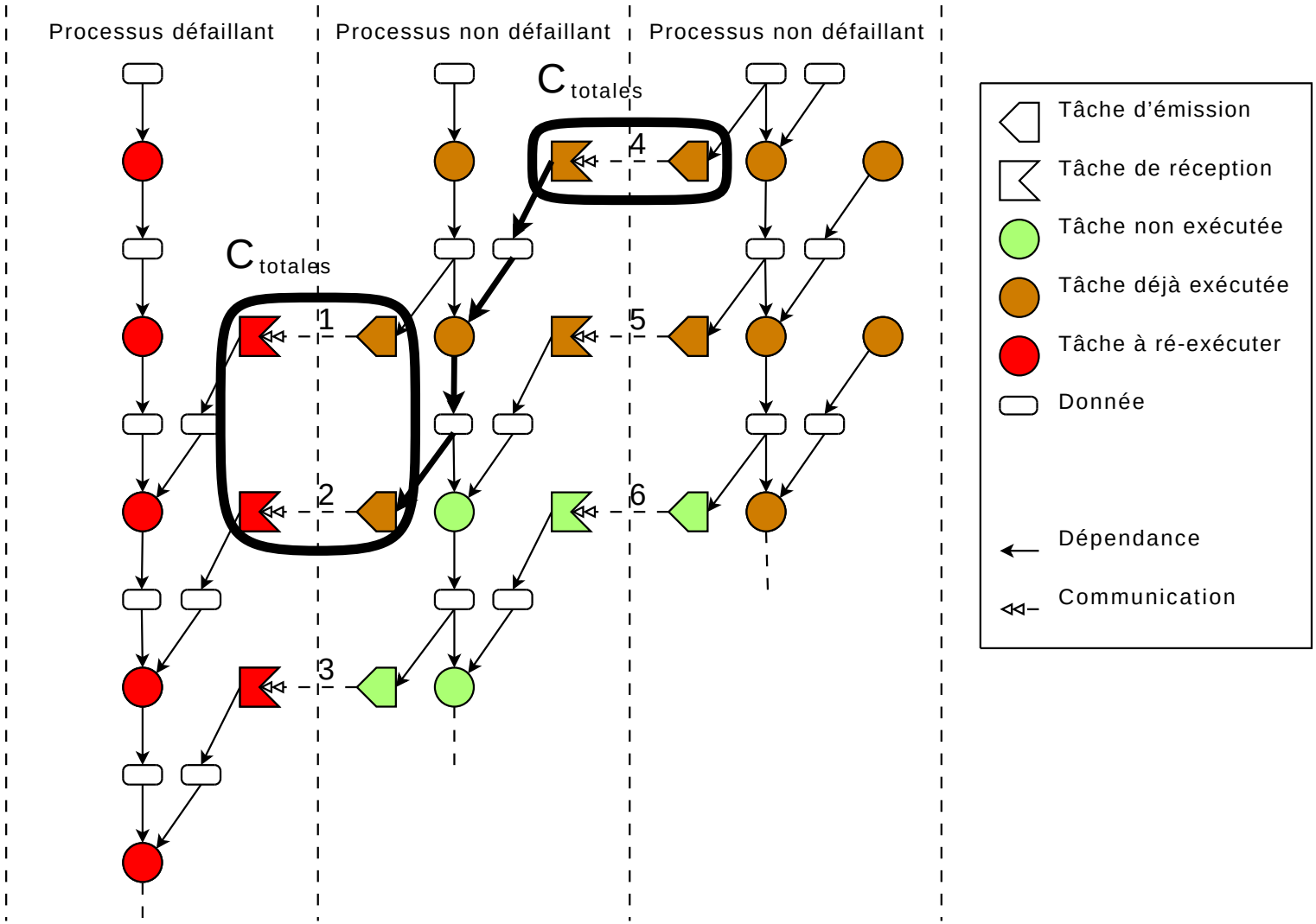
Communications perdues



Identification des communications

1re étape : identification des tâches à réexécuter

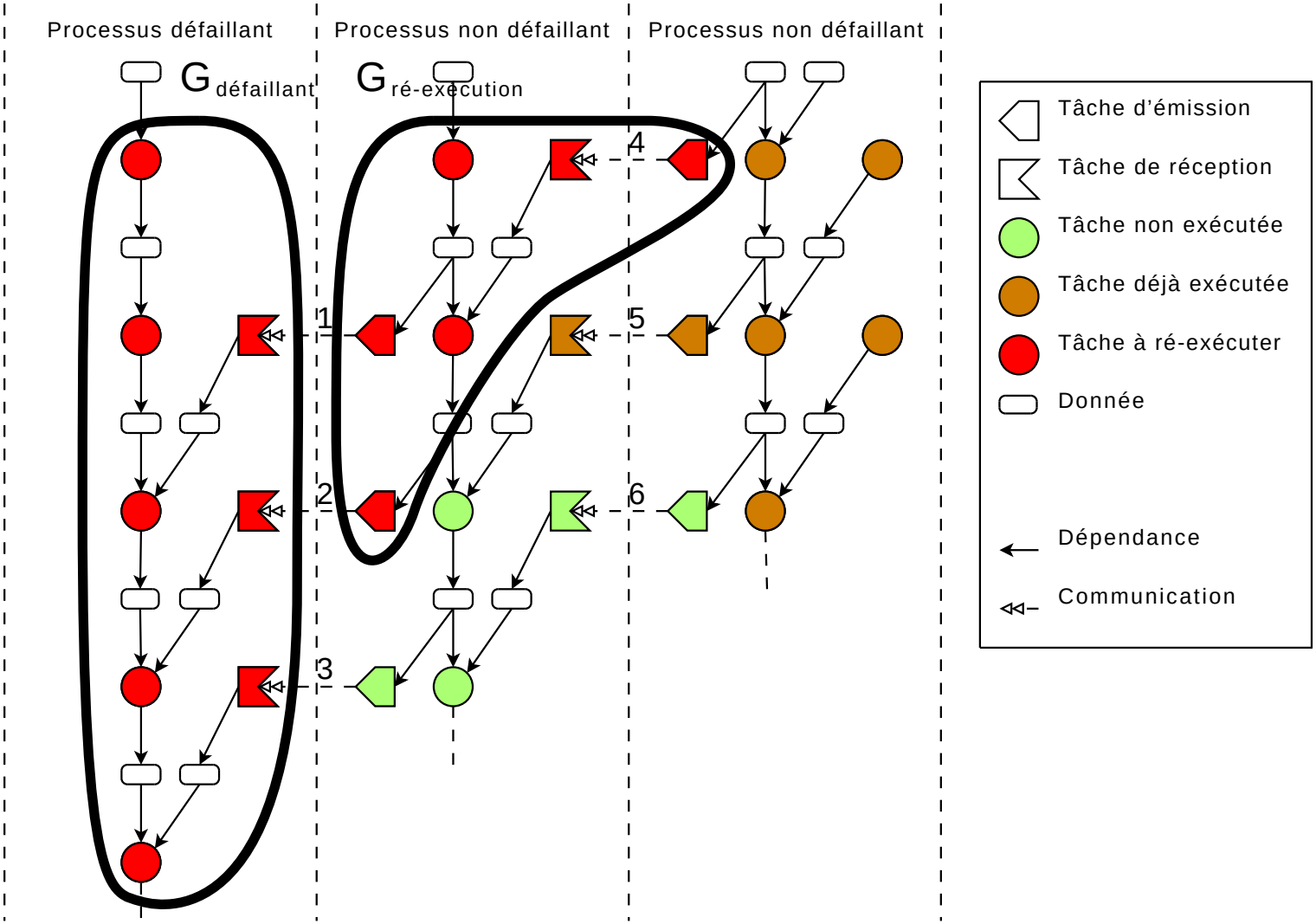
Communications à rejouer



Fermeture transitive du graphe

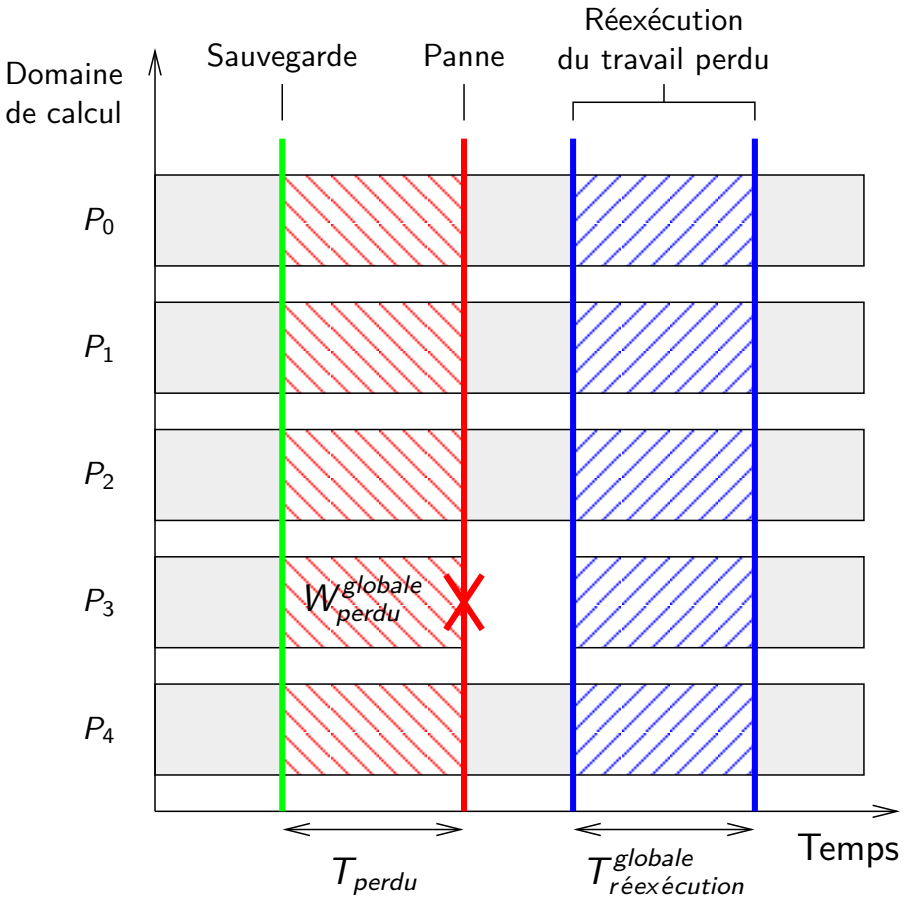
1re étape : identification des tâches à réexécuter

Tâches à réexécuter

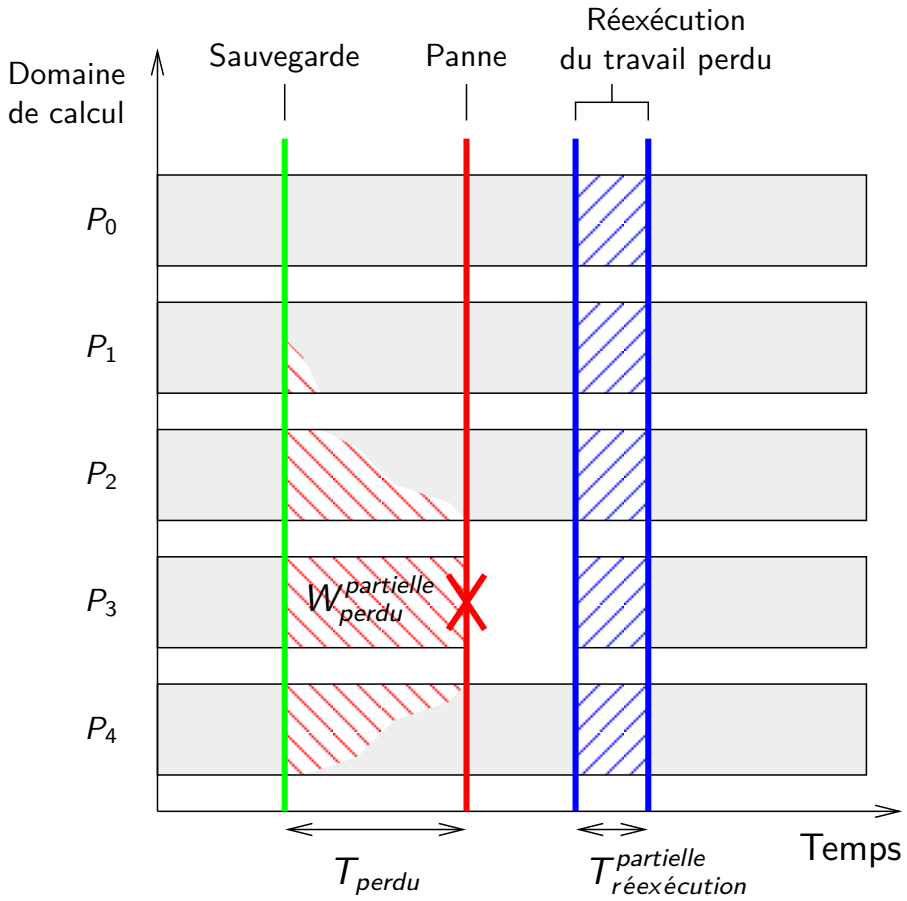


2e étape : réexécution du travail perdu

Reprise globale



Reprise partielle



Grâce à la sur-décomposition
le travail perdu peut être parallélisé !

Mesure du temps de réexécution du travail perdu

Remarque

Le temps de réexécution du travail perdu prend en compte :

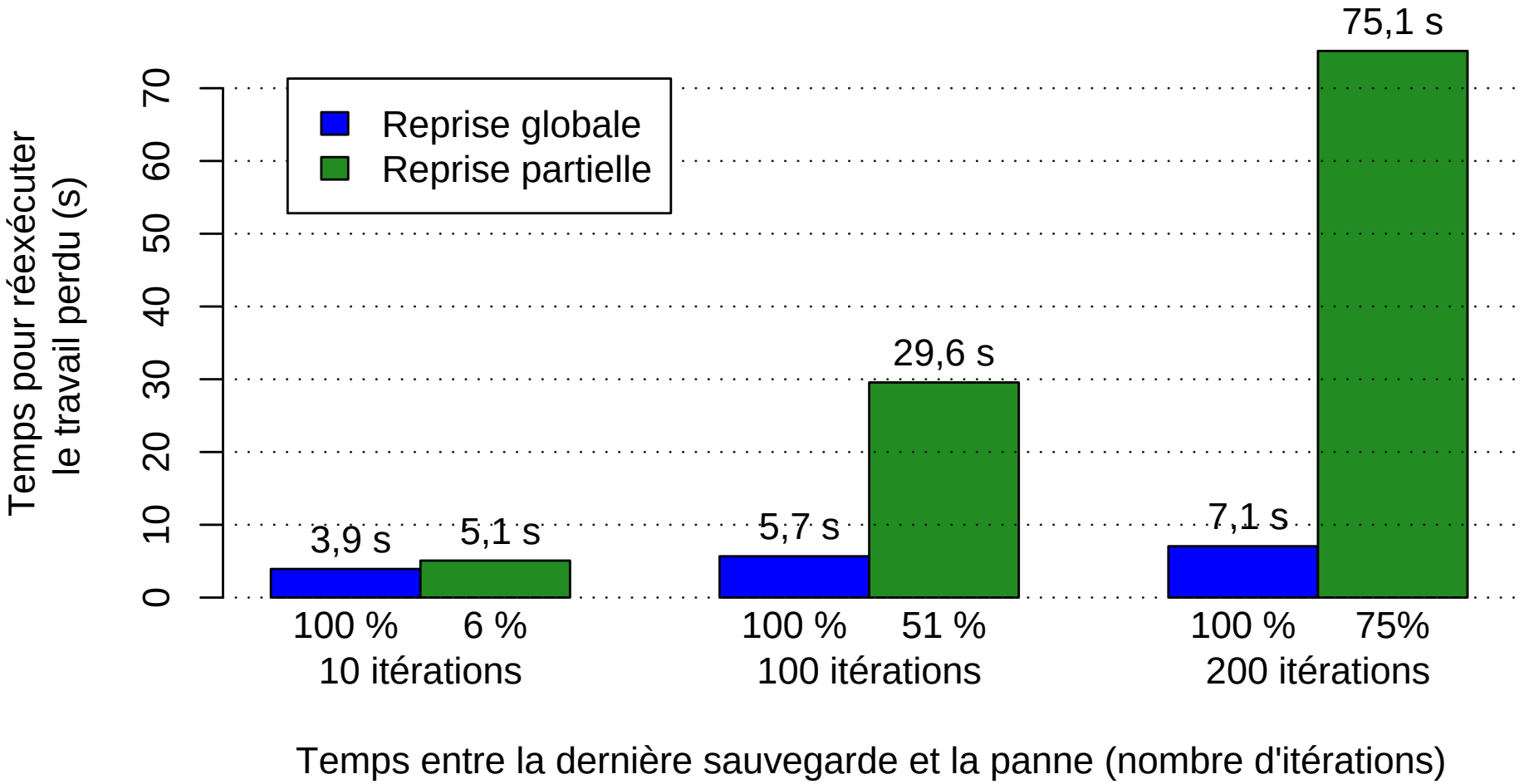
- le temps de réexécution des tâches
- le temps de redistribution des données

Expérience

- 100 machines de calcul, 10 serveurs de sauvegarde (grappe de Bordeaux)
- Taille du domaine = 76 Mo, découpé en $100 \times 10 \times 1$ sous-domaines
- Panne d'une machine fixée
- 2 grains de calcul considérées :
  - 2 ms pour la mise à jour d'un sous-domaine
  - 50 ms pour la mise à jour d'un sous-domaine

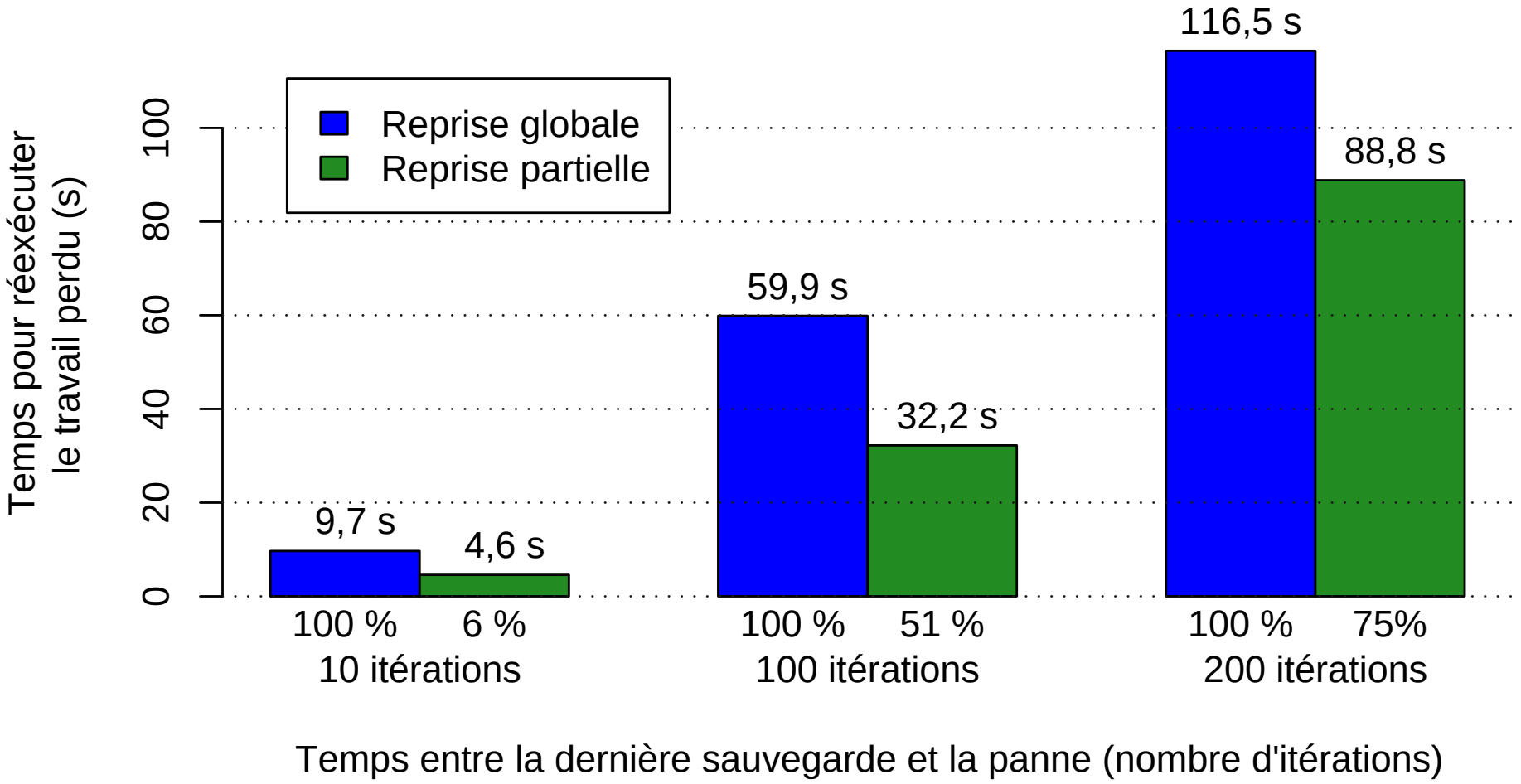
Temps de réexécution du travail perdu

Mise à jour d'un sous-domaine $\approx$ 2 ms



Temps de réexécution du travail perdu

Mise à jour d'un sous-domaine $\approx$ 50 ms



Les ordonnanceurs de Kaapi ne prennent pas en compte les communications

Sauvegarde dans Kaapi et Charm++

Jacobi sur un domaine en 3 dimensions

- Grappe Griffon de Nancy : 80 machines avec 8 cœurs, soit 640 cœurs
- Domaine de calcul : 10^6 réels de type `double` par cœur

Protocoles de sauvegarde

- Charm++ : Double sauvegarde coordonnée bloquante
- Kaapi : Sauvegarde coordonnée bloquante

Sauvegarde sur disque

	Taille totale	Temps de sauvegarde
Charm++	35,5 Go	27,1 s
Kaapi	15,8 Go	16,0 s

Reprise globale dans Kaapi et Charm++

Scénario

- 1 panne ⇒ Reprise sur 79 machines, soit 632 cœurs
- Domaine de calcul : 10^6 réels de type `double` par cœur

Protocoles de reprise et d'équilibrage de charge

- Charm++ : Reprise globale + algorithme OrbLB
- Kaapi : Reprise globale + partitionnement statique

Mesures des temps

	Reprise globale	Équilibrage de charge
Charm++	2,2 s	4,6 s
Kaapi	2,1 s	5,0 s

Plan

Autres contributions

Développement logiciel dans Kaapi ($\approx$ 100 000 lignes de code)

- Partitionnement statique ($\approx$ 10 000 lignes)
- Tolérance aux fautes ($\approx$ 10 000 lignes)
- Outils de déploiement sur grille et multi-grille (basés sur TakTuk)

Participation aux Plugtests 2007 et 2008

Victoires aux concours de déploiement et de calcul sur grille et multi-grille organisé par l'ETSI

- 2007 : application des N-Reines (3654 cœurs de Grid'5000)
- 2008 : application de finance (3609 cœurs sur Grid'5000 et Intrigger au Japon)

Visite au *Parallel Programming Laboratory* à Urbana-Champaign

- Dans le cadre du laboratoire commun entre l'INRIA et Urbana-Champaign pour le *Petascale Computing*
- Comparaison qualitative et expérimentale de Charm++ et Kaapi

Rappel des contributions

Reconfiguration dynamique

Conception d'un mécanisme de reconfiguration dynamique générique et performant

- Gestion des accès concurrents
  - → appliquée au vol de travail dans X-Kaapi
- Gestion de la cohérence mutuelle
  - → appliquée à l'ordonnancement et à la tolérance aux fautes dans Kaapi

Tolérance aux fautes

- Étude des paramètres qui influencent le cout de la sauvegarde
  - → serveurs de sauvegarde, taille de la sauvegarde, période de sauvegarde
- Amélioration de la vitesse d'exécution après la reprise
  - → grâce à un rééquilibrage de charge et à la sur-décomposition
- Conception et implémentation d'un protocole original de reprise partielle
  - → utilisant les informations offertes par la représentation abstraite de l'application

qui ont fait l'objet de publications et communications orales :
7 publications, dont 3 conférences internationales et 1 chapitre de livre

Enseignements tirés de ces études

Reconfiguration dynamique

- Exécution concurrente et exécution coopérative
  - → Exécution coopérative réduit le surcout à l'exécution
  - → Mais possède une forte latence à gros grain
  - → Bonnes performances dans X-Kaapi comparées à Cilk et TBB
- Cohérence mutuelle optimisée dans Kaapi
  - → Réduit et stabilise le cout de la coordination
  - → Gain faible pour la sauvegarde (transfert des données prépondérant)

Tolérance aux fautes

- Sur-décomposition et rééquilibrage de charge à la reprise
  - → Gain important pour l'exécution après reprise
  - → Bénéficie à tout le reste de l'exécution
- Reprise partielle
  - → Quantité de travail à réexécuter plus faible
  - → Gain en temps lié au ratio calcul/communication
  - → L'ordonnancement doit prendre en compte les communications

Perspectives 1/2

Compromis pour la gestion des accès concurrents

- Méthode hybride entre l'exécution concurrente et l'exécution coopérative

Exécution avec garanties de performances pour Kaapi

- Adaptation dynamique
- Ajout et retrait de machines pour adapter l'efficacité
- Réaliser les fonctions d'*Observation* et de *Décision*

Algorithmes d'ordonnancement avec contraintes de placement

- Prendre en compte le placement des données et le cout des communications
- Problème NP-dur
- $\Rightarrow$ Chercher des heuristiques

Perspectives 2/2

Comparaisons expérimentales

- Kaapi, Charm++/AMPI, Open MPI
- À grande échelle

Modèles analytiques des protocoles de tolérance aux fautes

- Choix du nombre de serveurs de sauvegarde
- Choix des intervalles de sauvegardes
- Basé sur des lois de probabilité d'apparition des pannes
- $\Rightarrow$ Algorithmes en ligne

Réalisation d'une mémoire stable distribuée

- Persistance par la redondance d'informations ($\Rightarrow$ codes correcteurs)
- Choix/placement automatique des machines de sauvegarde
- Transparence référentielle

Merci de votre attention

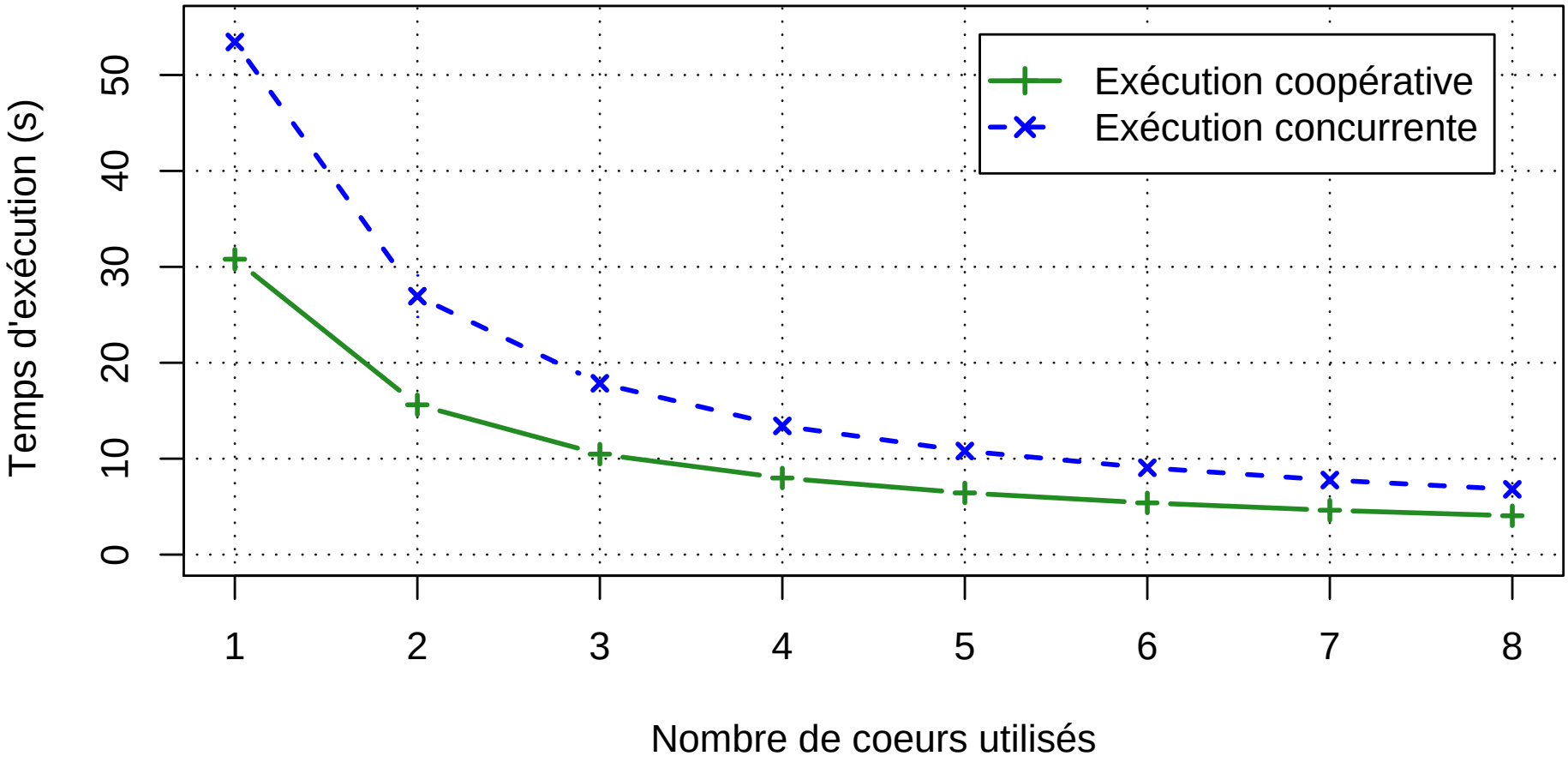
Questions ?

Annexes

◂ Plan de la présentation

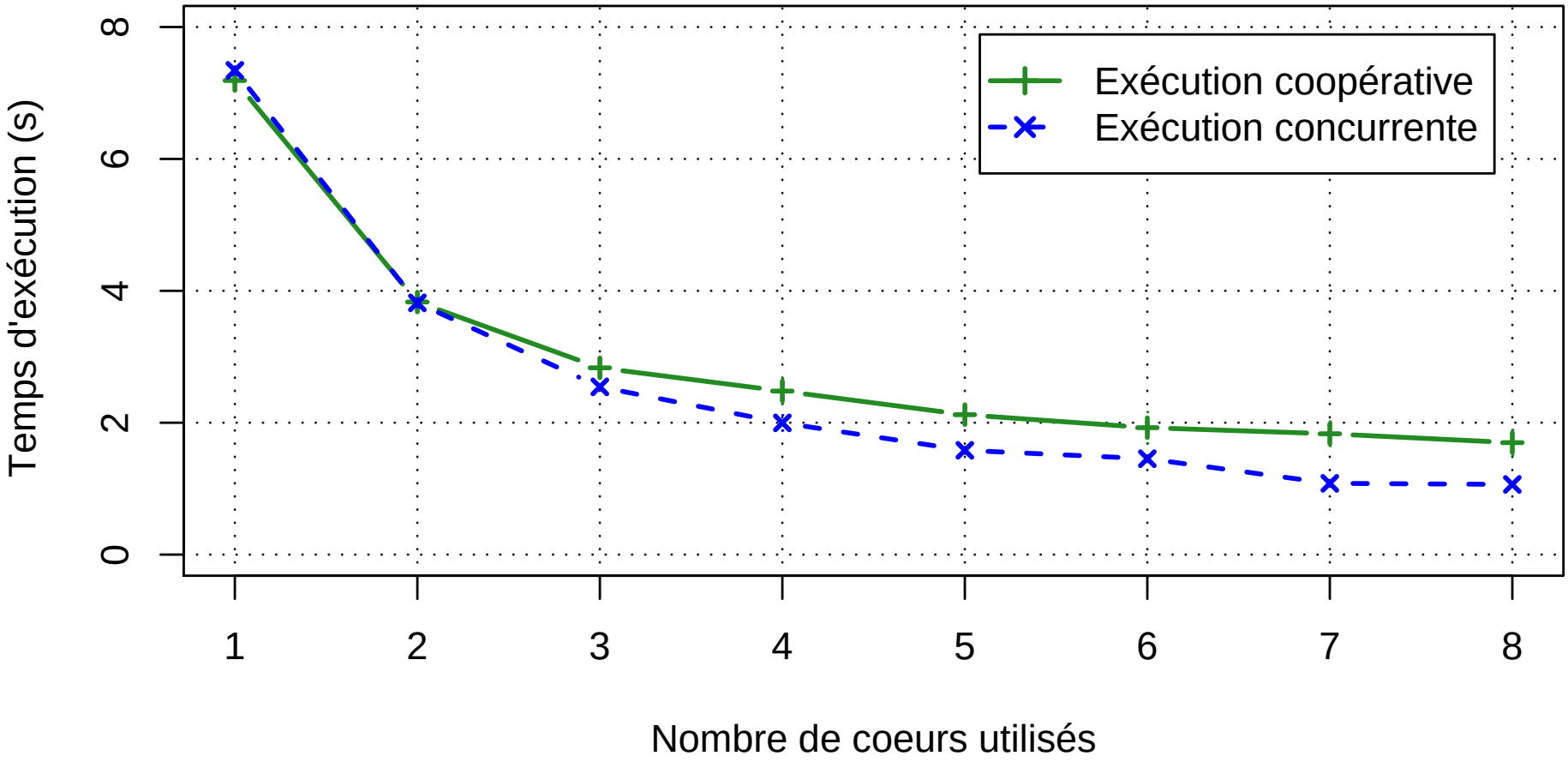
Exécution concurrente et exécution coopérative

- Calcul du N^e terme de Fibonacci (approche naïve récursive)
- Pour N = 45, temps séquentielle = 7,2 s
- N = 45, seuil = 5 (grain fin)



Exécution concurrente et exécution coopérative

- Calcul du N^e terme de Fibonacci (approche naïve récursive)
- Pour N = 45, temps séquentielle = 7,2 s
- N = 45, seuil = 40 (gros grain)

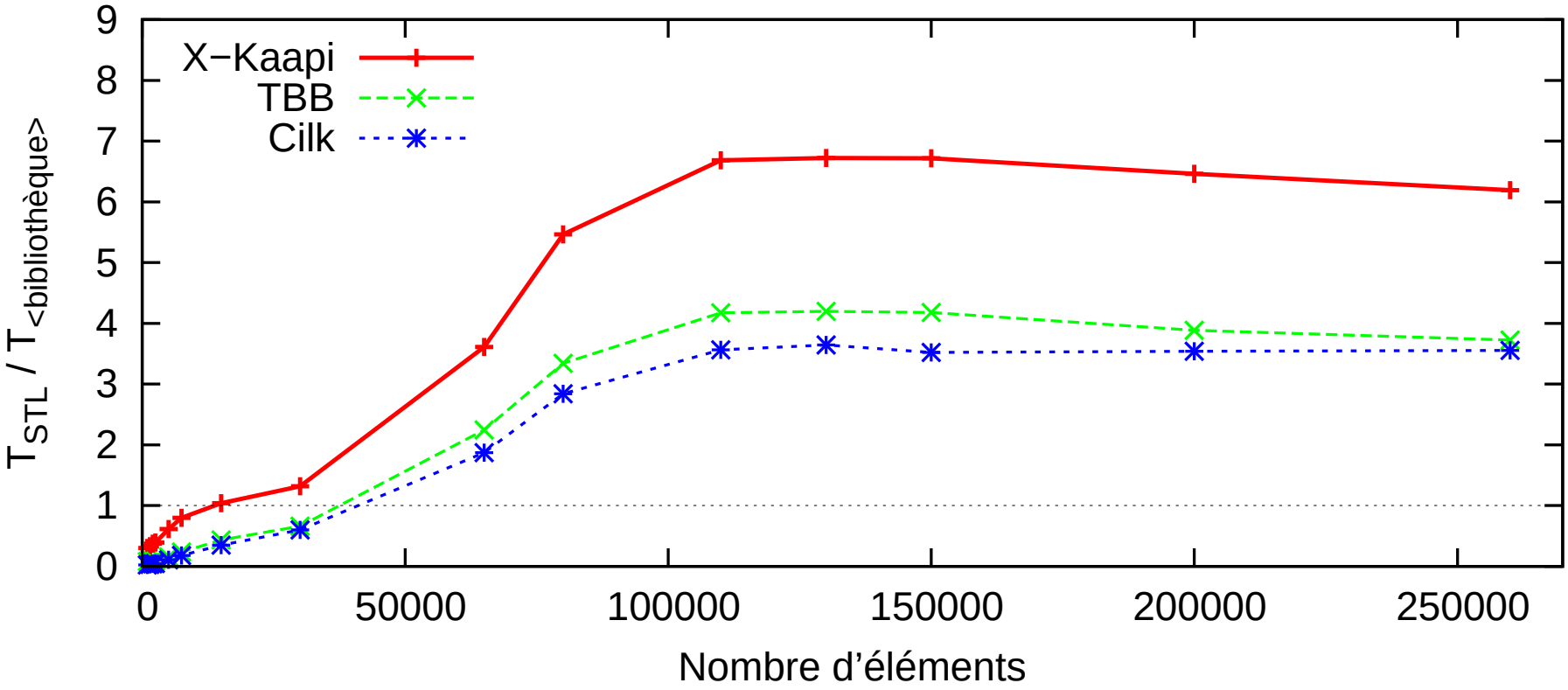


Annexes

◂ Plan de la présentation

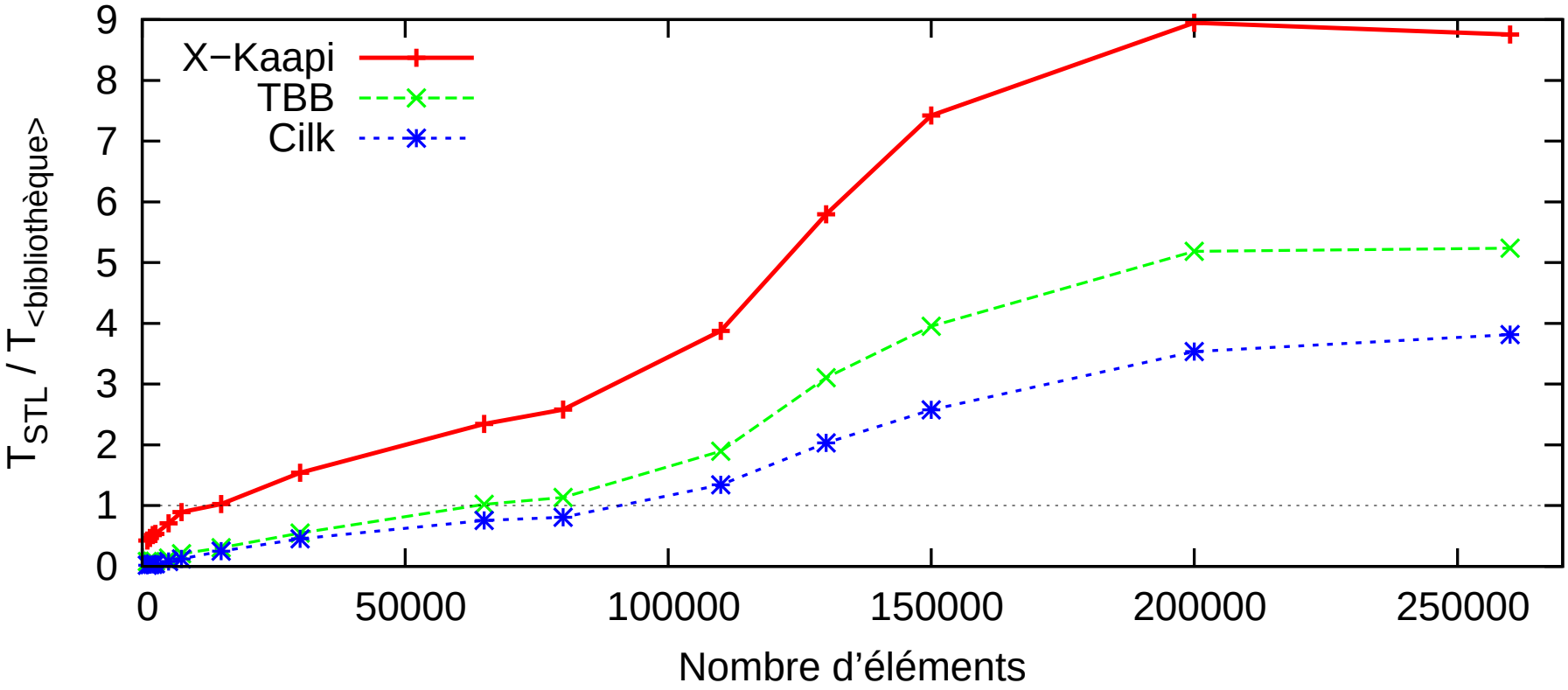
Comparaison X-Kaapi, Cilk et TBB

- X-Kaapi : exécution coopérative
- Accélération par rapport à la STL
- Algorithme `transform` de la STL



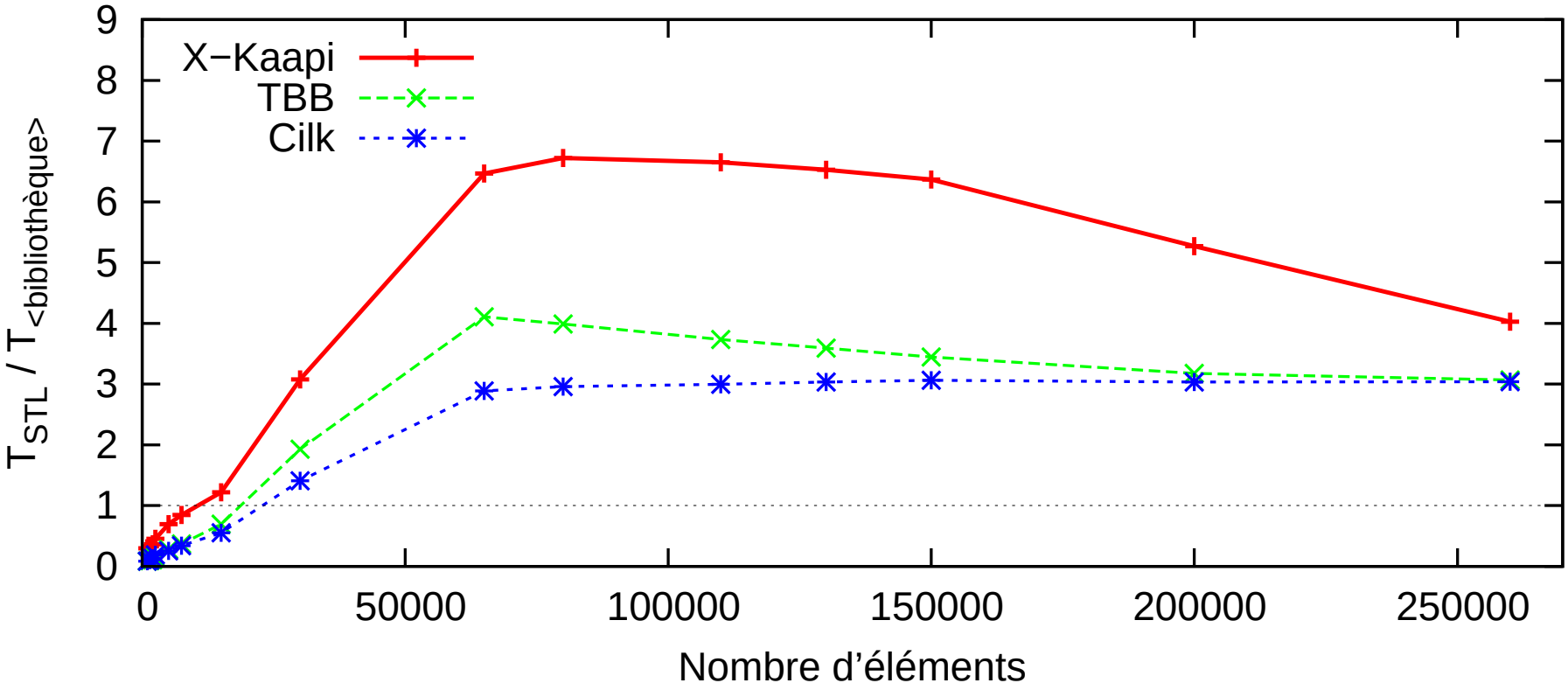
Comparaison X-Kaapi, Cilk et TBB

- X-Kaapi : exécution coopérative
- Accélération par rapport à la STL
- Algorithme `min_element` de la STL



Comparaison X-Kaapi, Cilk et TBB

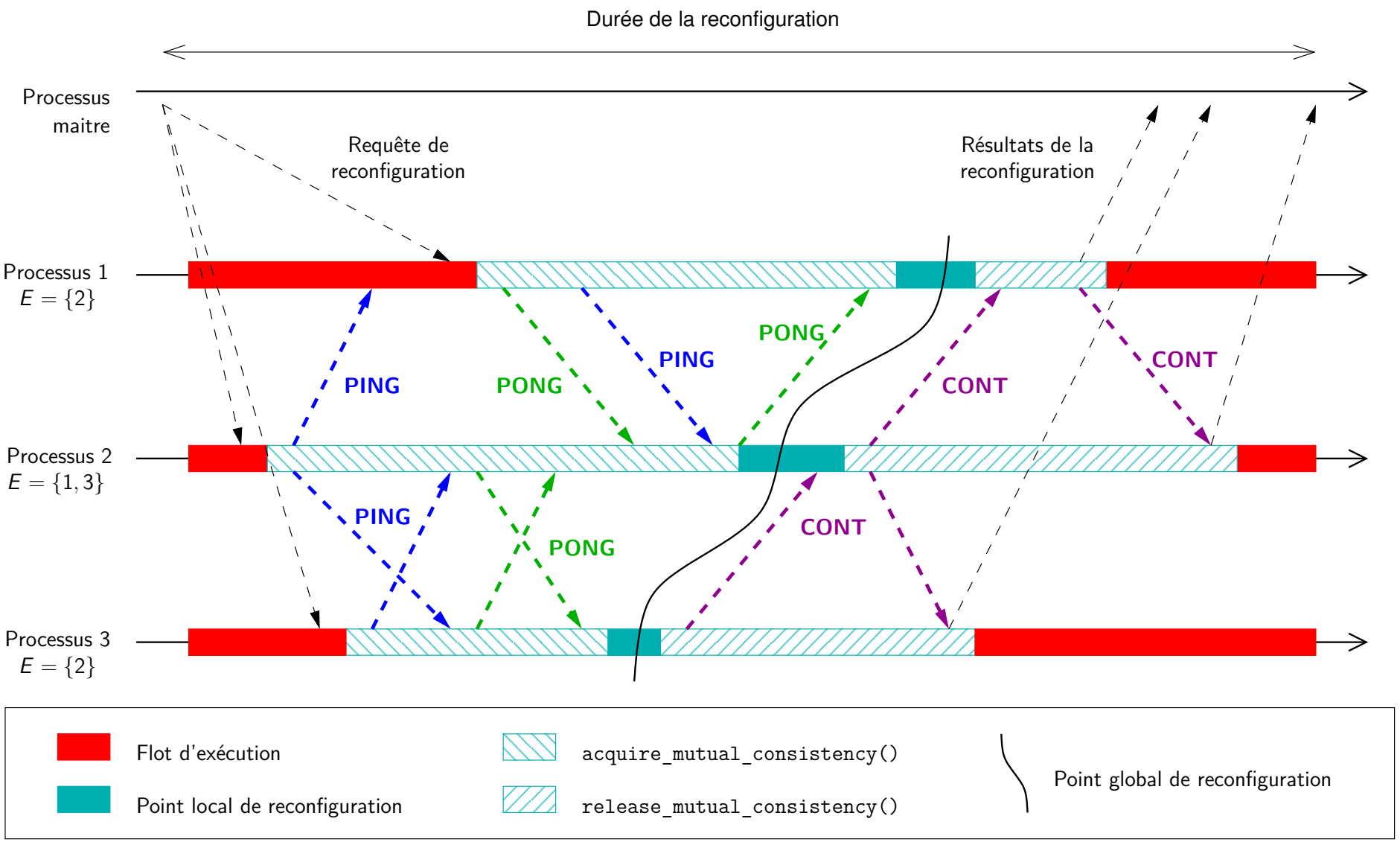
- X-Kaapi : exécution coopérative
- Accélération par rapport à la STL
- Algorithme `merge` de la STL



Annexes

◂ Plan de la présentation

Protocole de cohérence mutuelle dans Kaapi

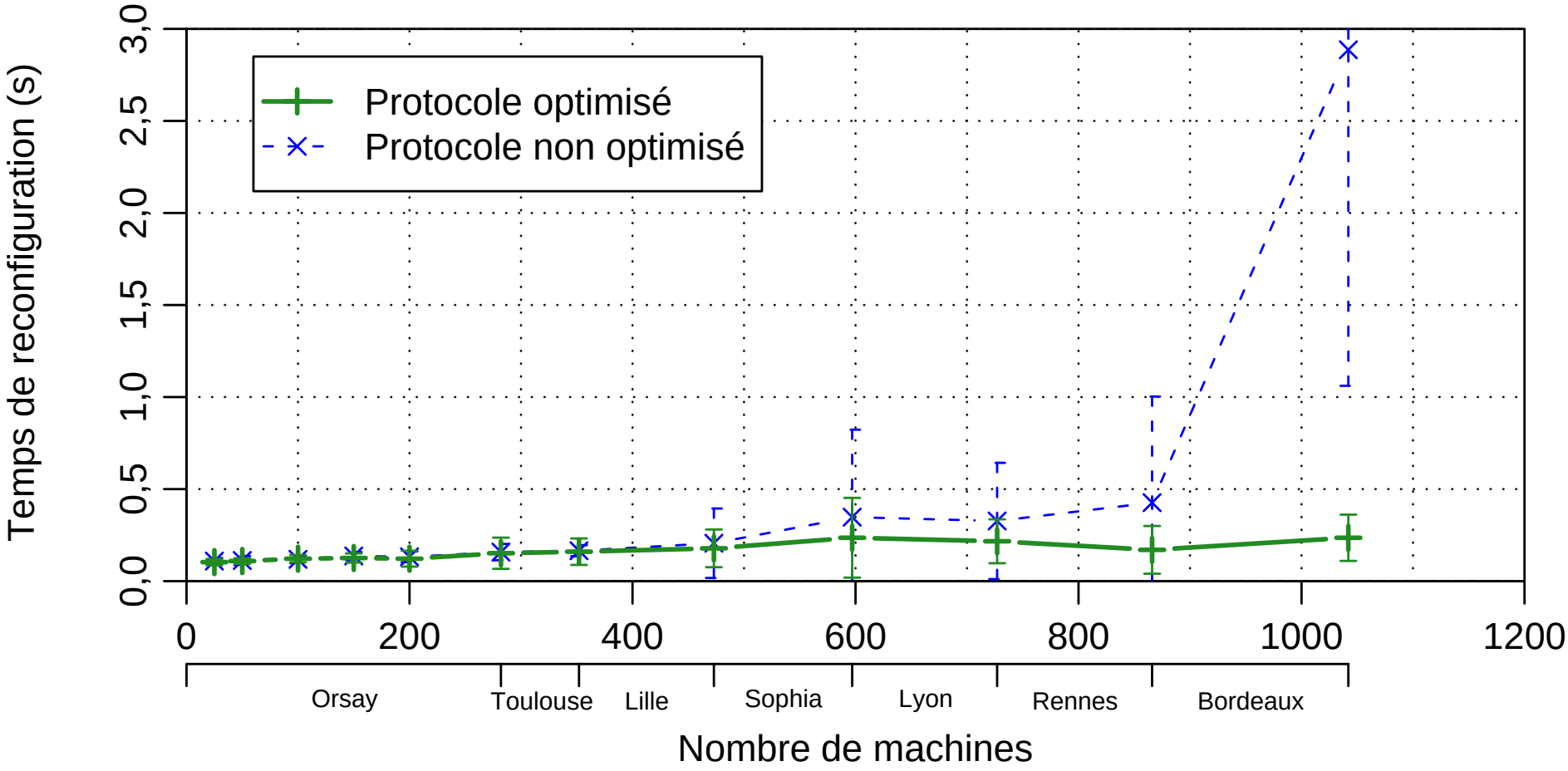


Annexes

◂ Plan de la présentation

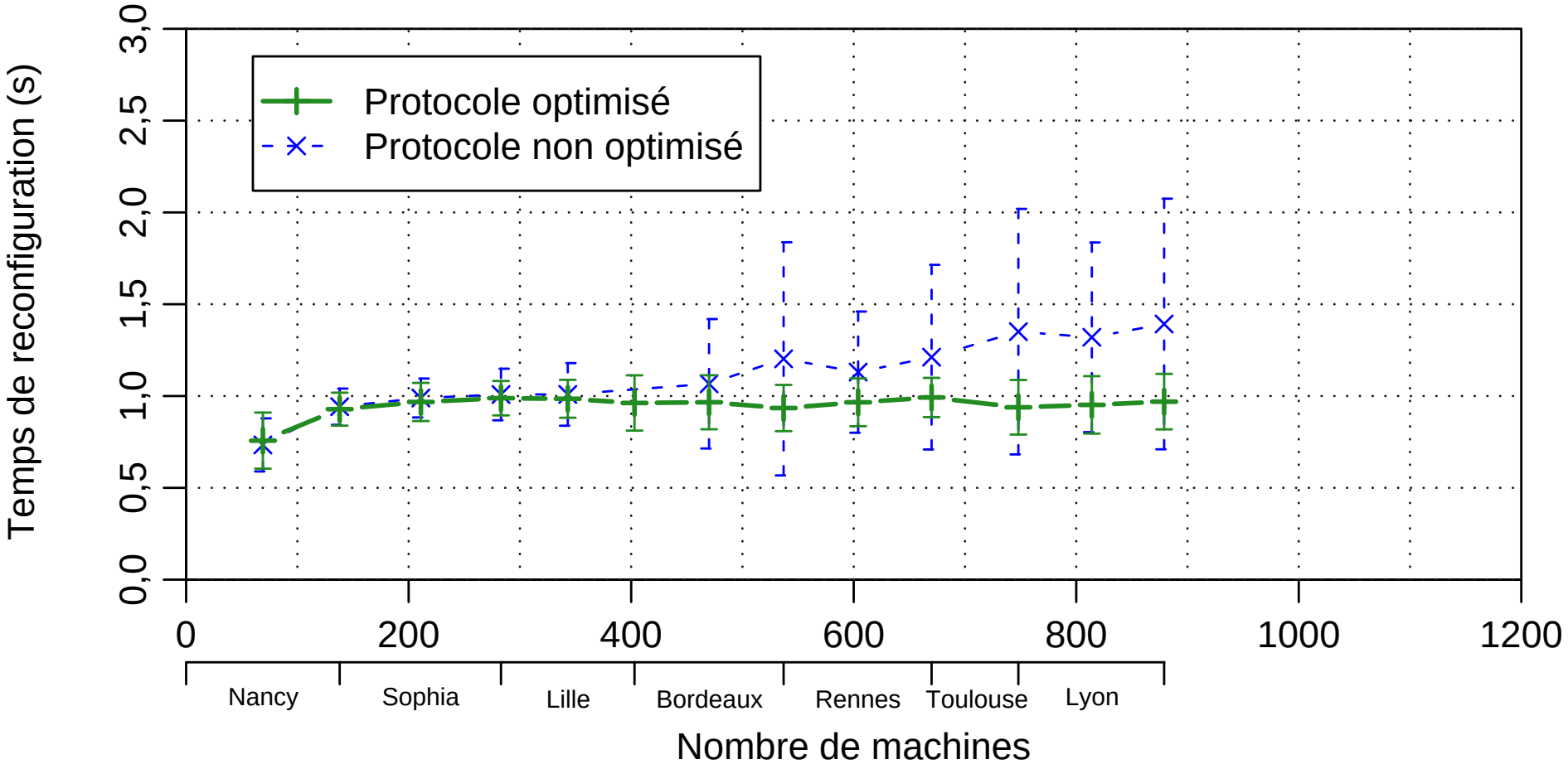
Temps de gestion de la cohérence mutuelle

Application des -Reines ordonnancée vol de travail
Reconfiguration « vide »



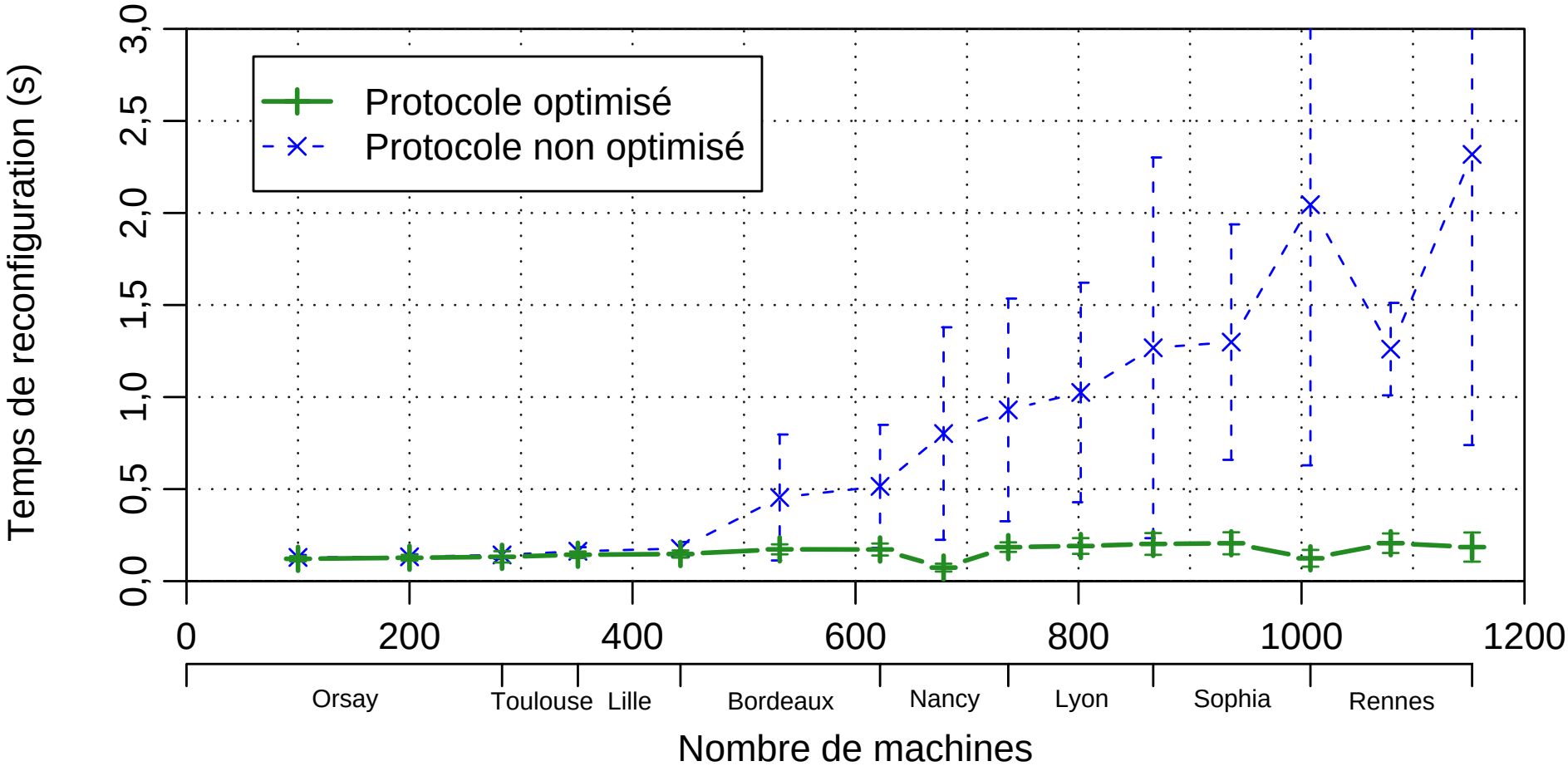
Temps de gestion de la cohérence mutuelle

Application des -Reines ordonnancée vol de travail
Reconfiguration « vide »



Temps de gestion de la cohérence mutuelle

Application des -Reines ordonnancée vol de travail
Reconfiguration « vide »



Annexes

◂ Plan de la présentation

Placement des serveurs de sauvegarde

Idée

Réduire le temps de sauvegarde en plaçant les serveurs de sauvegarde près des machines de calcul

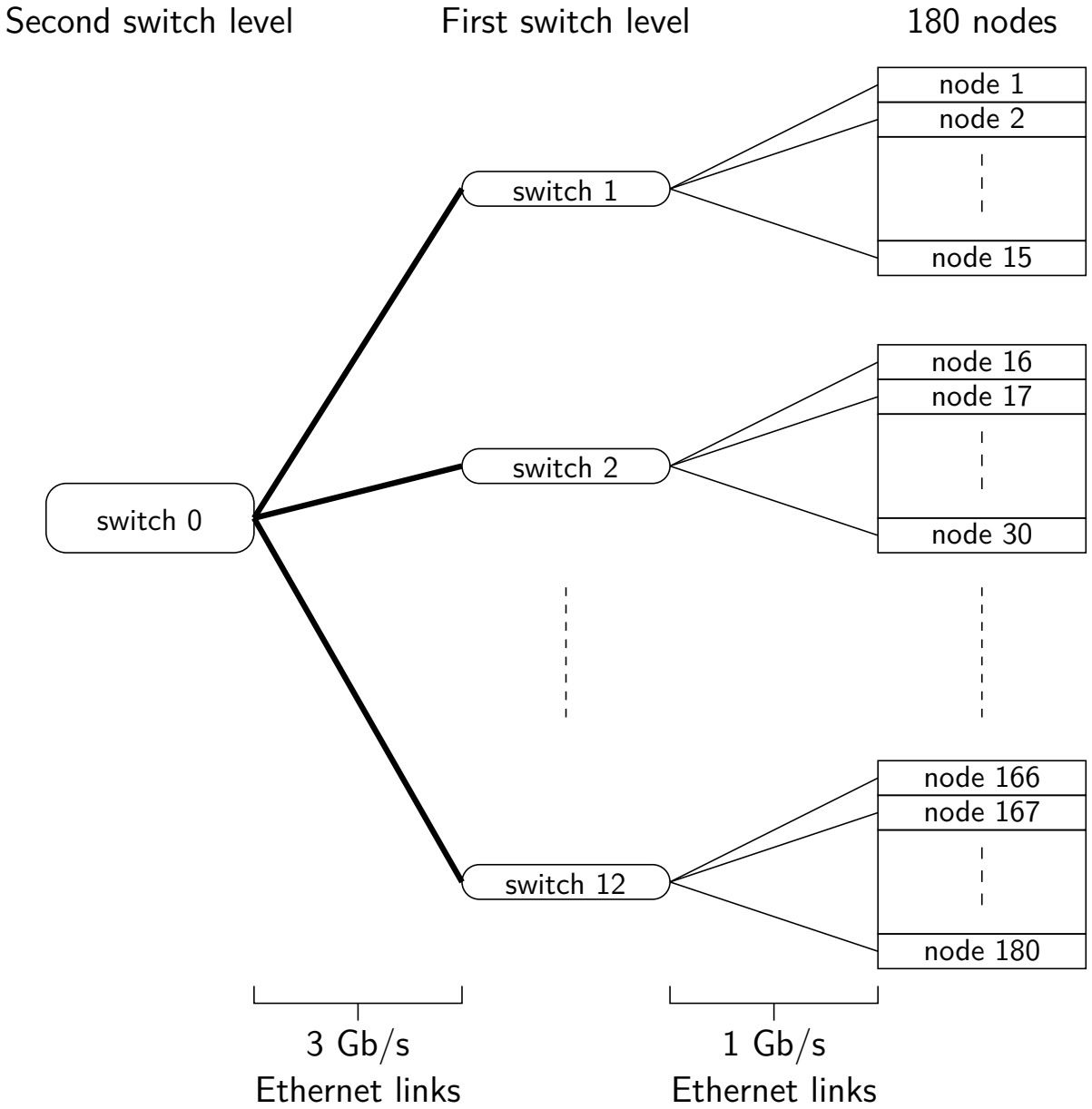
En pratique, les serveurs de sauvegarde peuvent être :

- une machine dédiée d'une grappe
- un autre processus de calcul (Charm++)

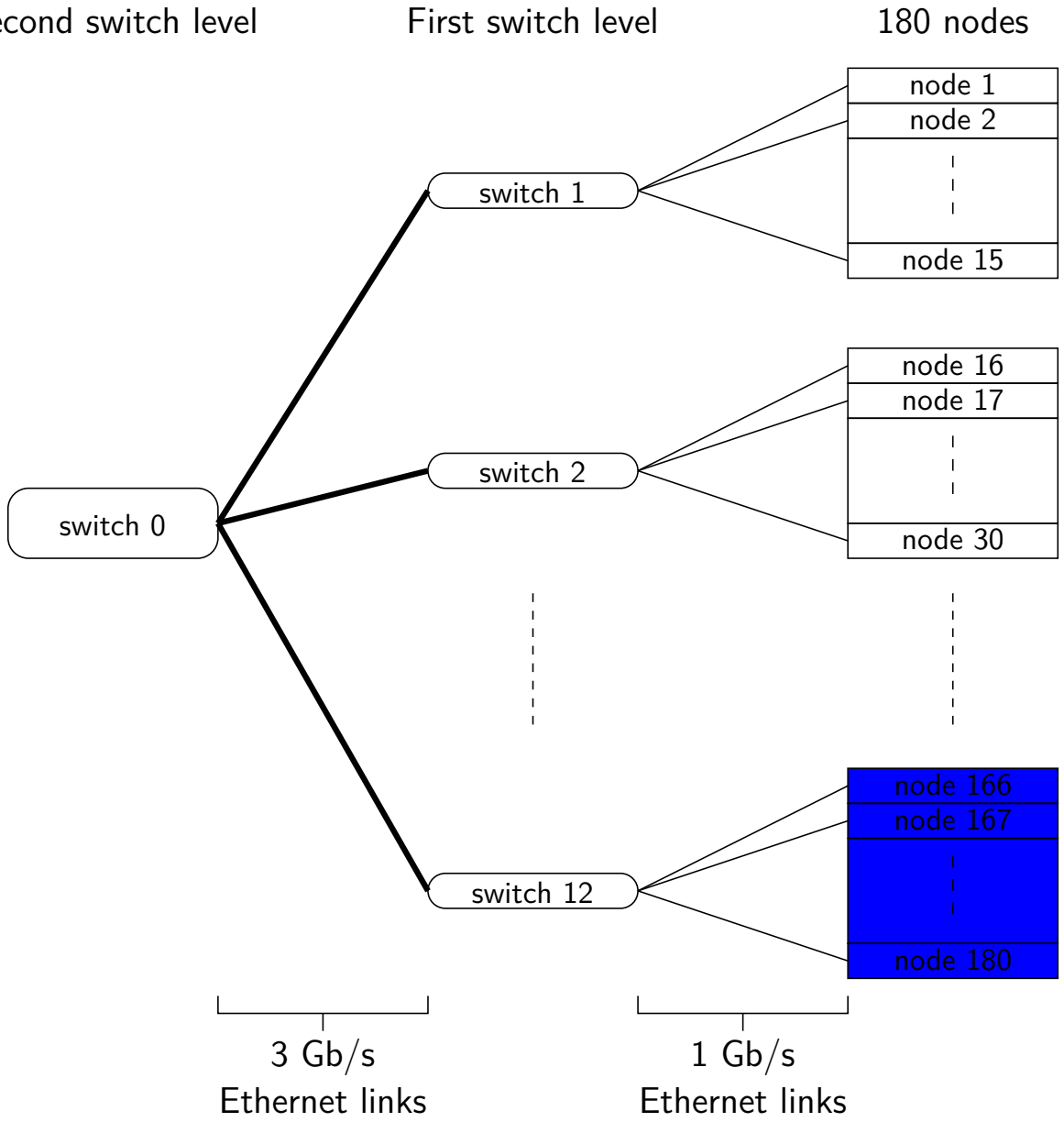
Étude expérimentale

- 180 machines de la grappe d'Orsay
  - 120 machines de calcul
  - 12, 24 ou 60 machines pour les serveurs de sauvegarde
- État de l'application $\approx$ 20 Go, *i.e.* 169 Mo par machines
- Évaluation de 3 placements : ordonné, par switch et aléatoire

Topologie réseau de la grappe d'Orsay

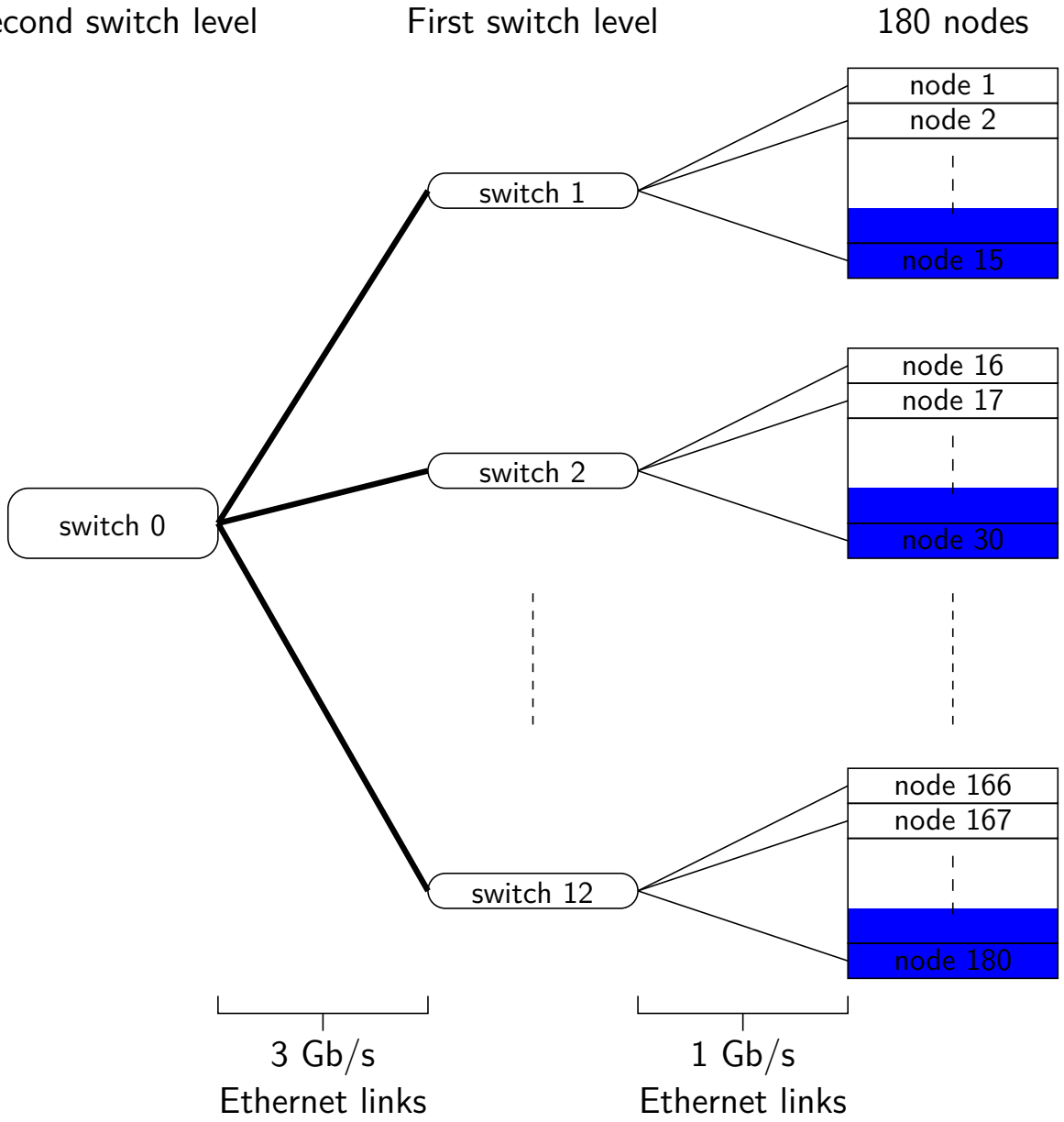


Topologie réseau de la grappe d'Orsay



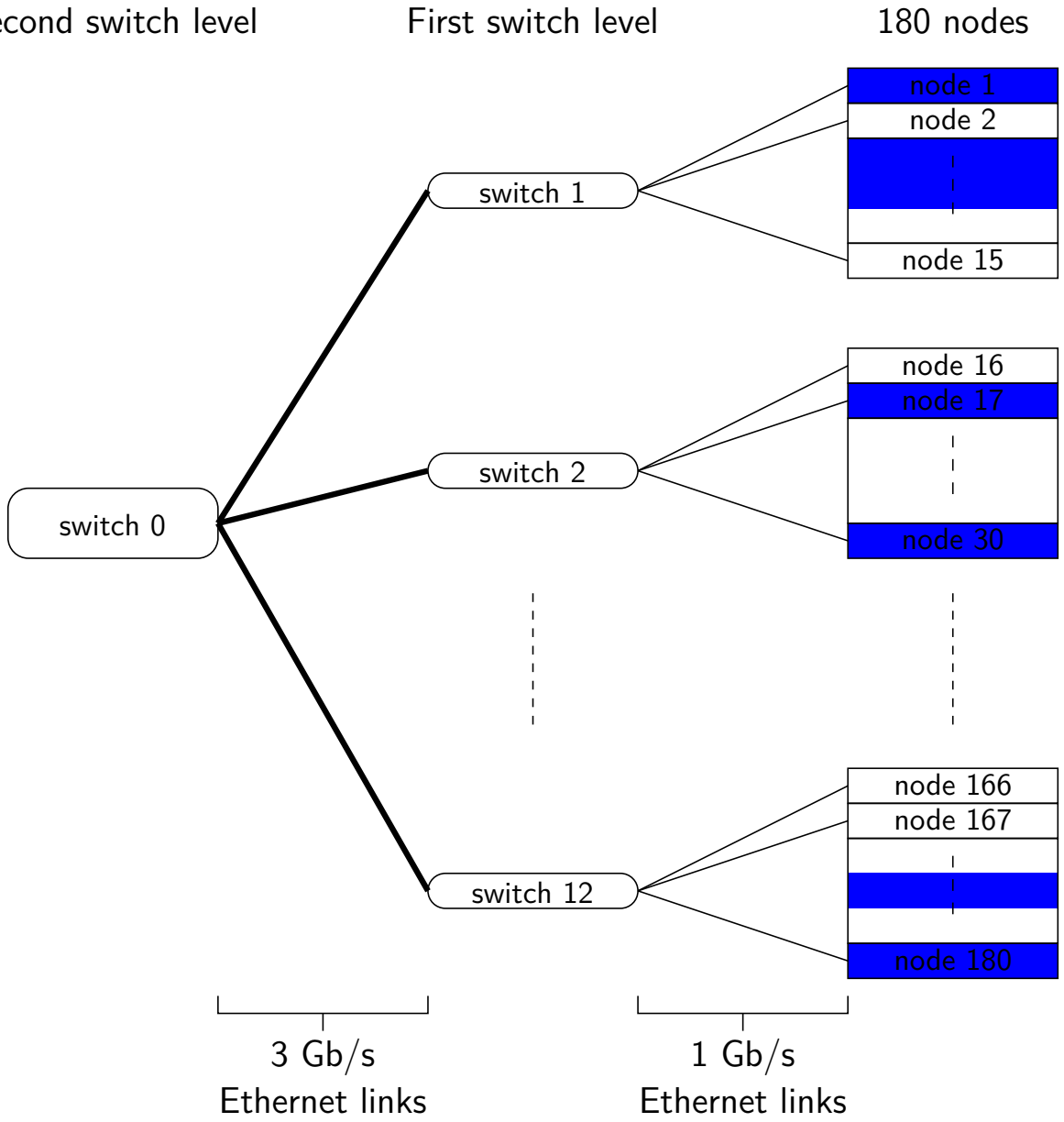
Les serveurs de sauvegarde peuvent être placés dans l'ordre

Topologie réseau de la grappe d'Orsay



Les serveurs de sauvegarde peuvent être placés par switch

Topologie réseau de la grappe d'Orsay

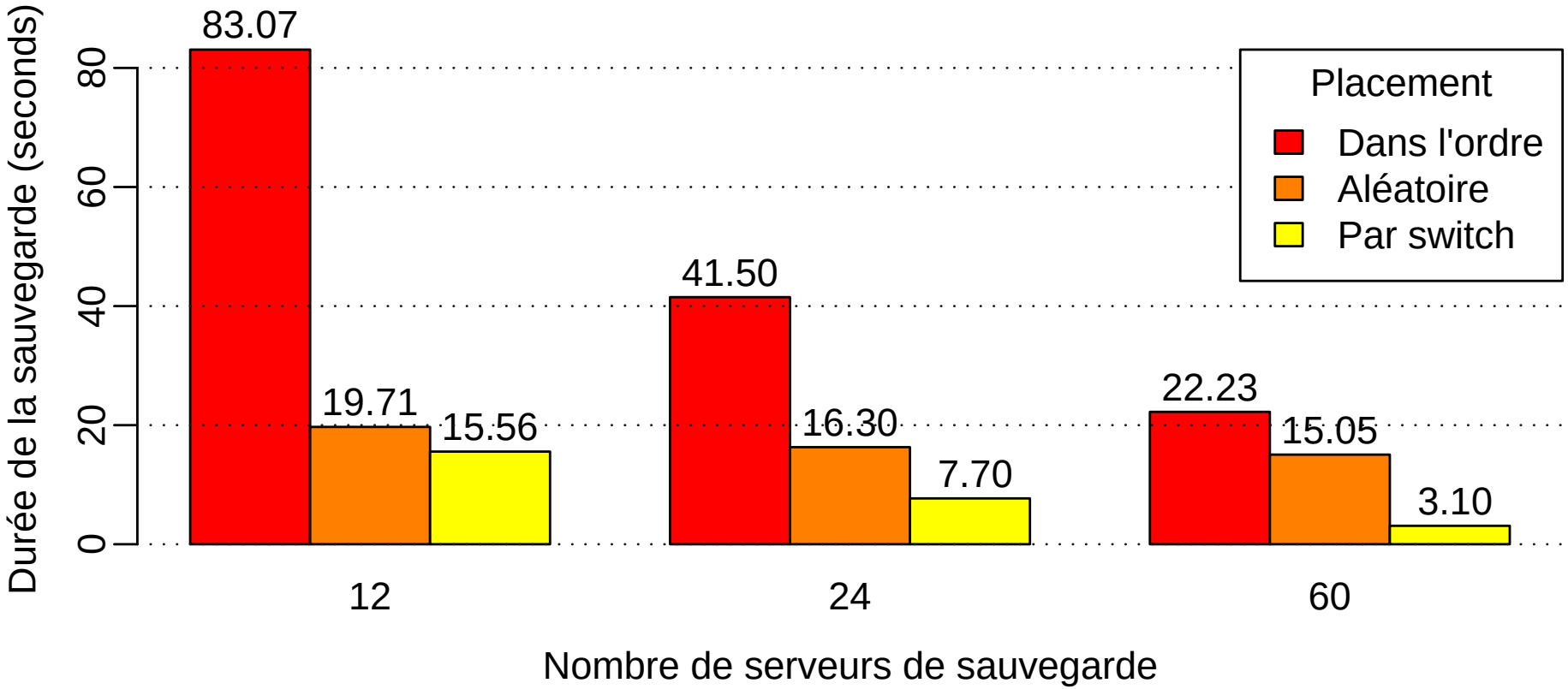


Les serveurs de sauvegarde peuvent être placés aléatoirement

Placement des serveurs de sauvegarde

120 machines de calcul

État de l'application ≈ 20 Go, *i.e.* 169 Mo par machine



⇒ Nécessaire de prendre en compte la topologie du réseau

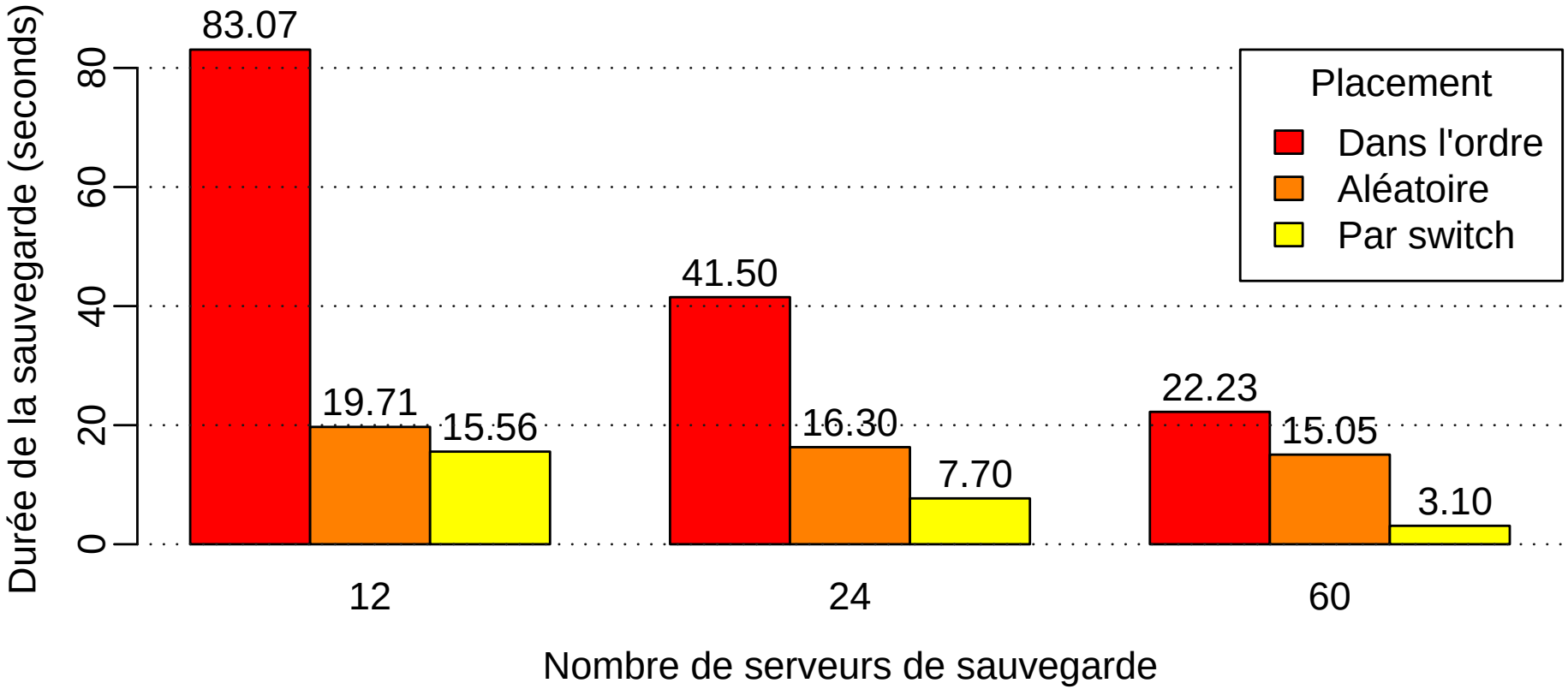
Même phénomène au niveau grille

Peut être automatisé (avec Network Weather Service par exemple)

Placement des serveurs de sauvegarde

120 machines de calcul

État de l'application ≈ 20 Go, *i.e.* 169 Mo par machine



⇒ Nécessaire de prendre en compte la topologie du réseau

Même phénomène au niveau grille

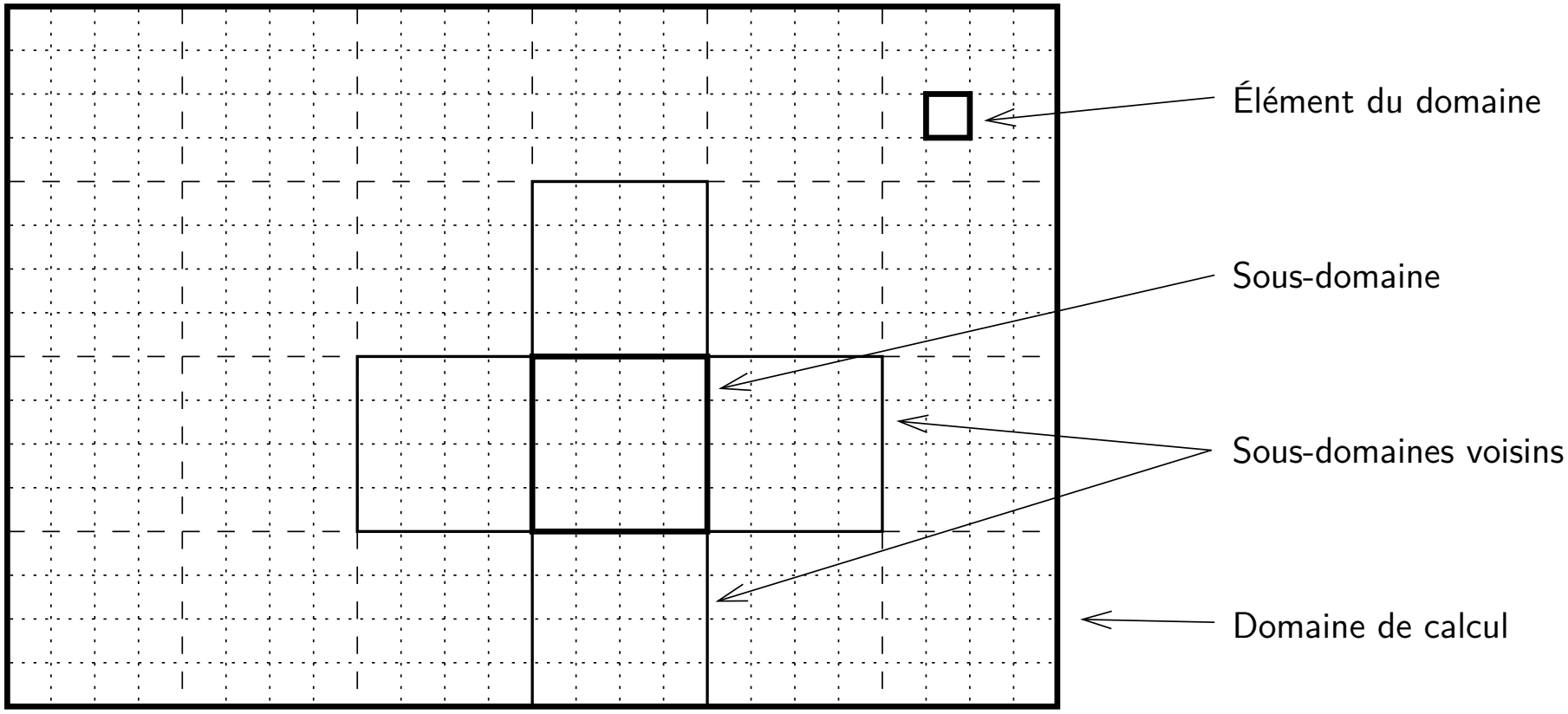
Peut être automatisé (avec Network Weather Service par exemple)

Annexes

◂ Plan de la présentation

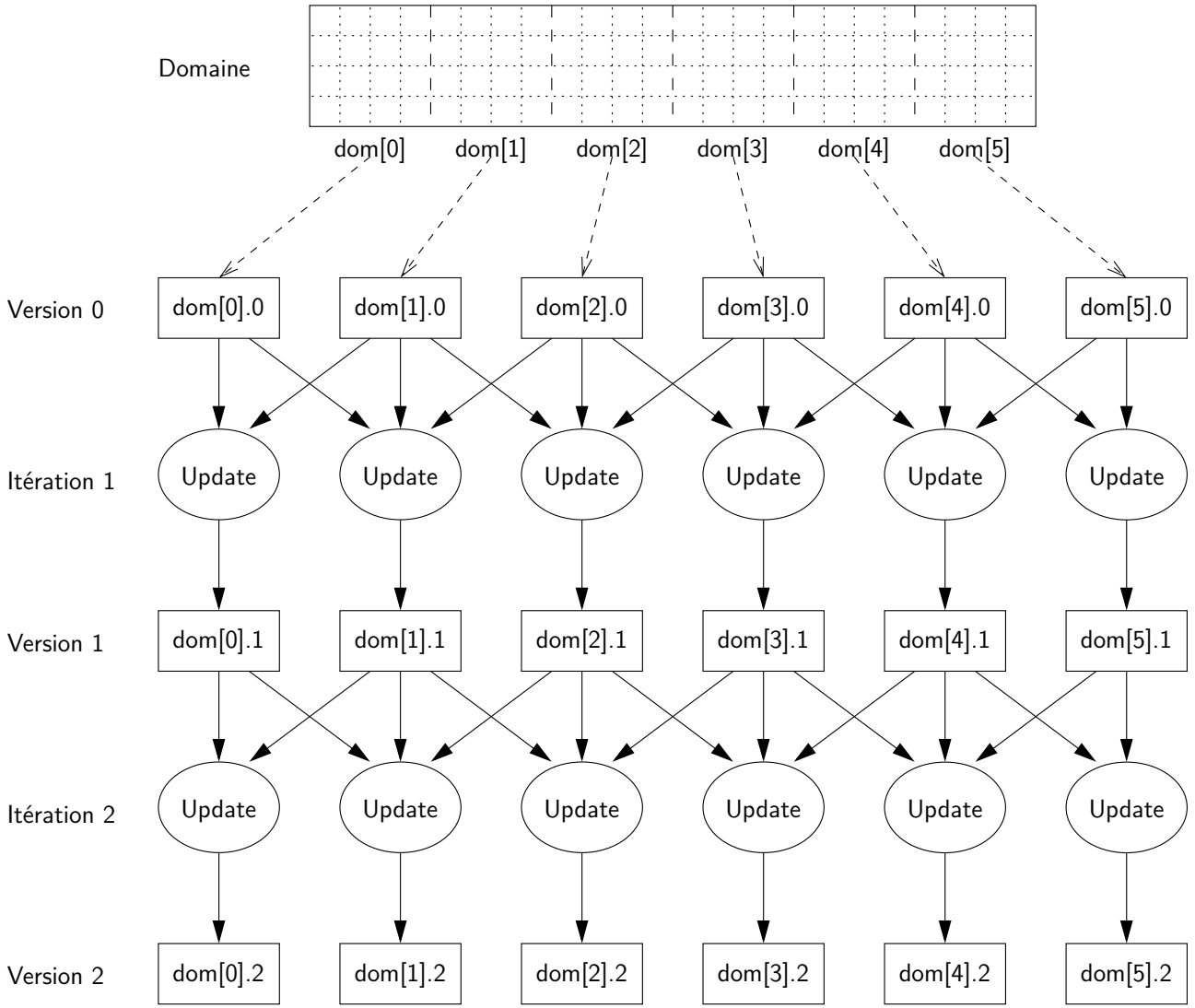
Exemple : Jacobi3D par décomposition de domaine

Example avec un domaine en 2 dimensions



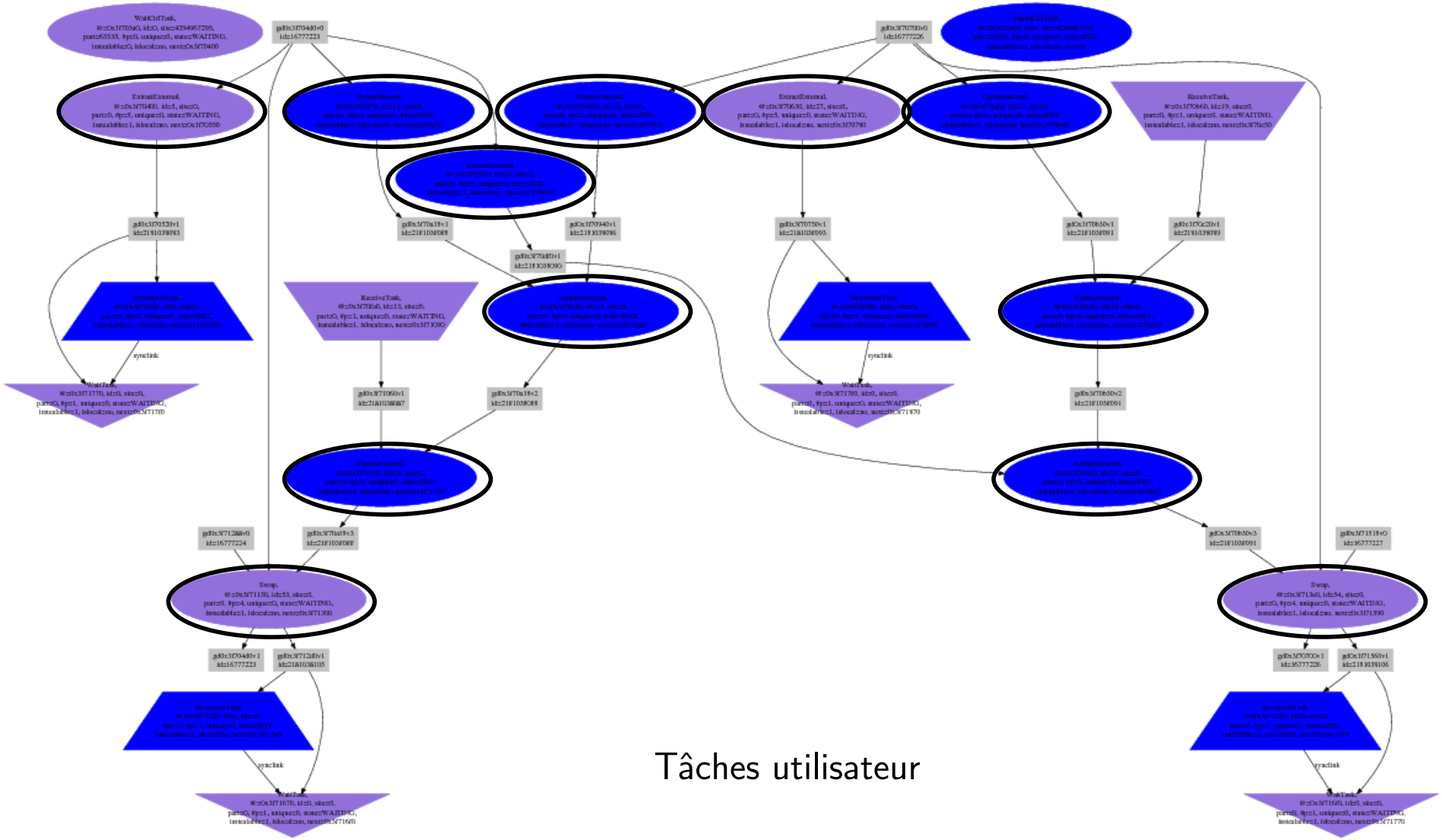
Un sous-domaine $\longleftrightarrow$ Une donnée partagée dans le graphe de flot de données

Jacobi3D : Décomposition de domaine & itérations



Application Jacobi3D : Graphe de flot de données réel

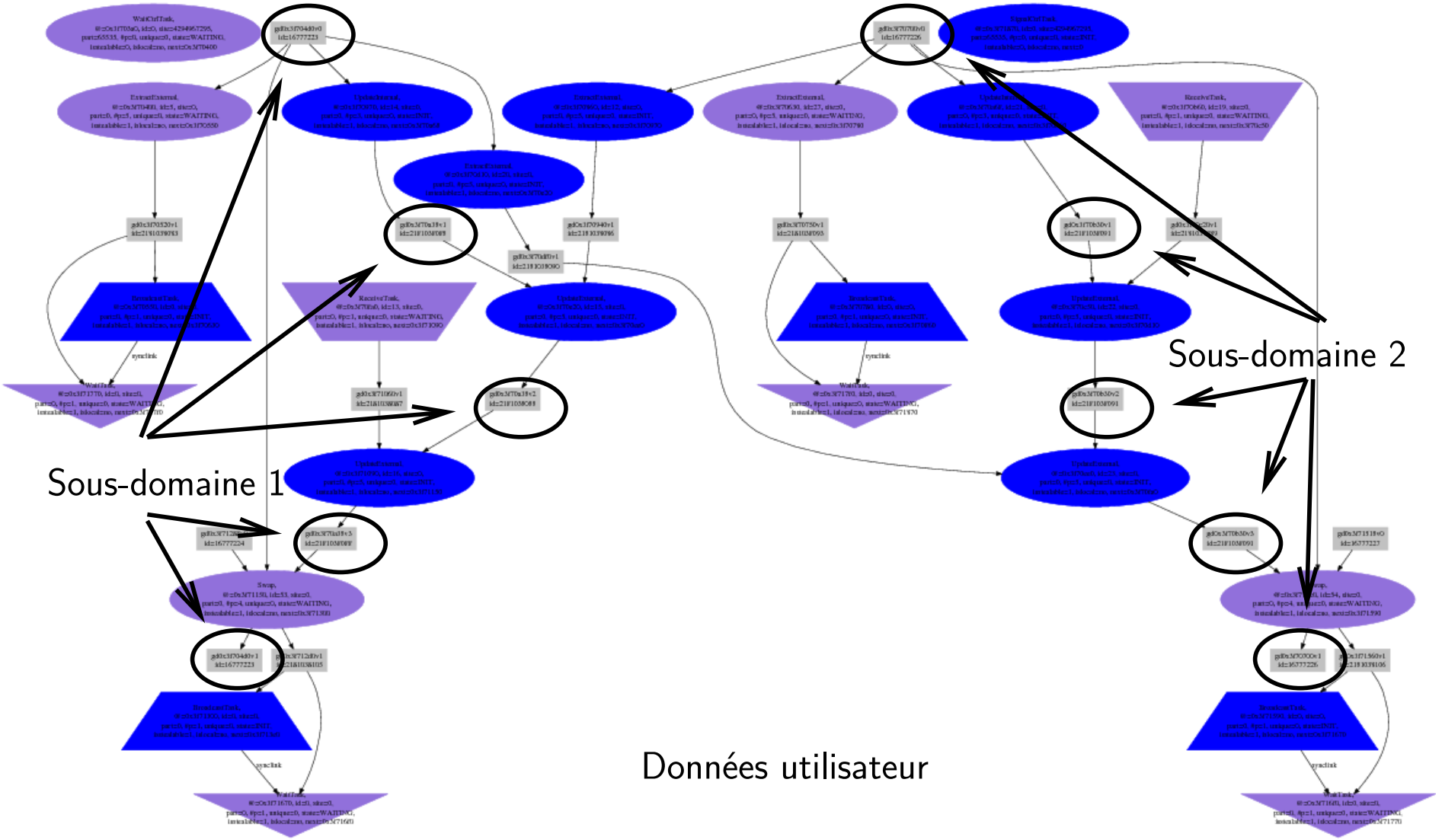
Graphe de flot de données généré pour le processus *N*â



Tâches utilisateur

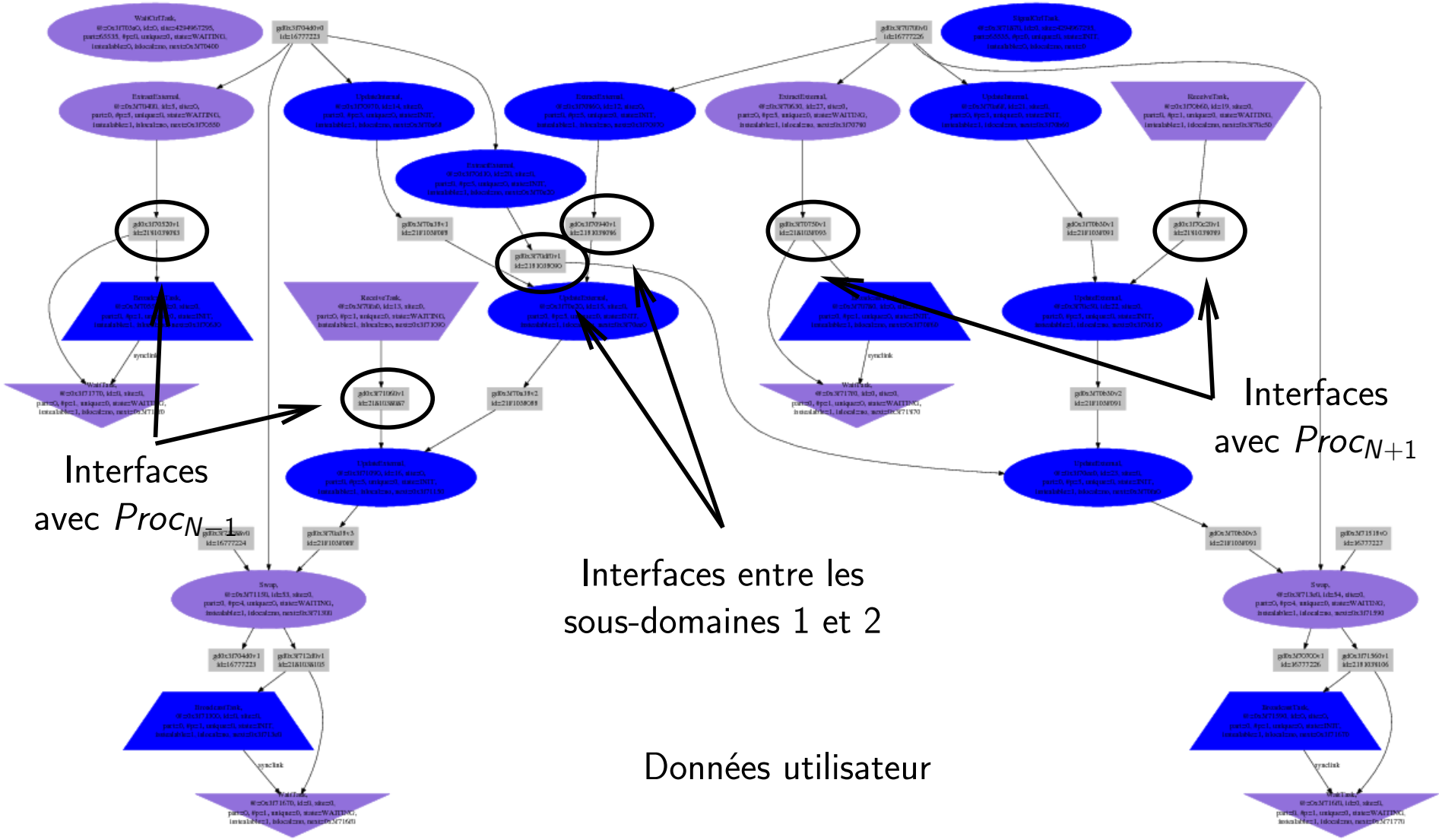
Application Jacobi3D : Graphe de flot de données réel

Graphe de flot de données généré pour le processus *N*â



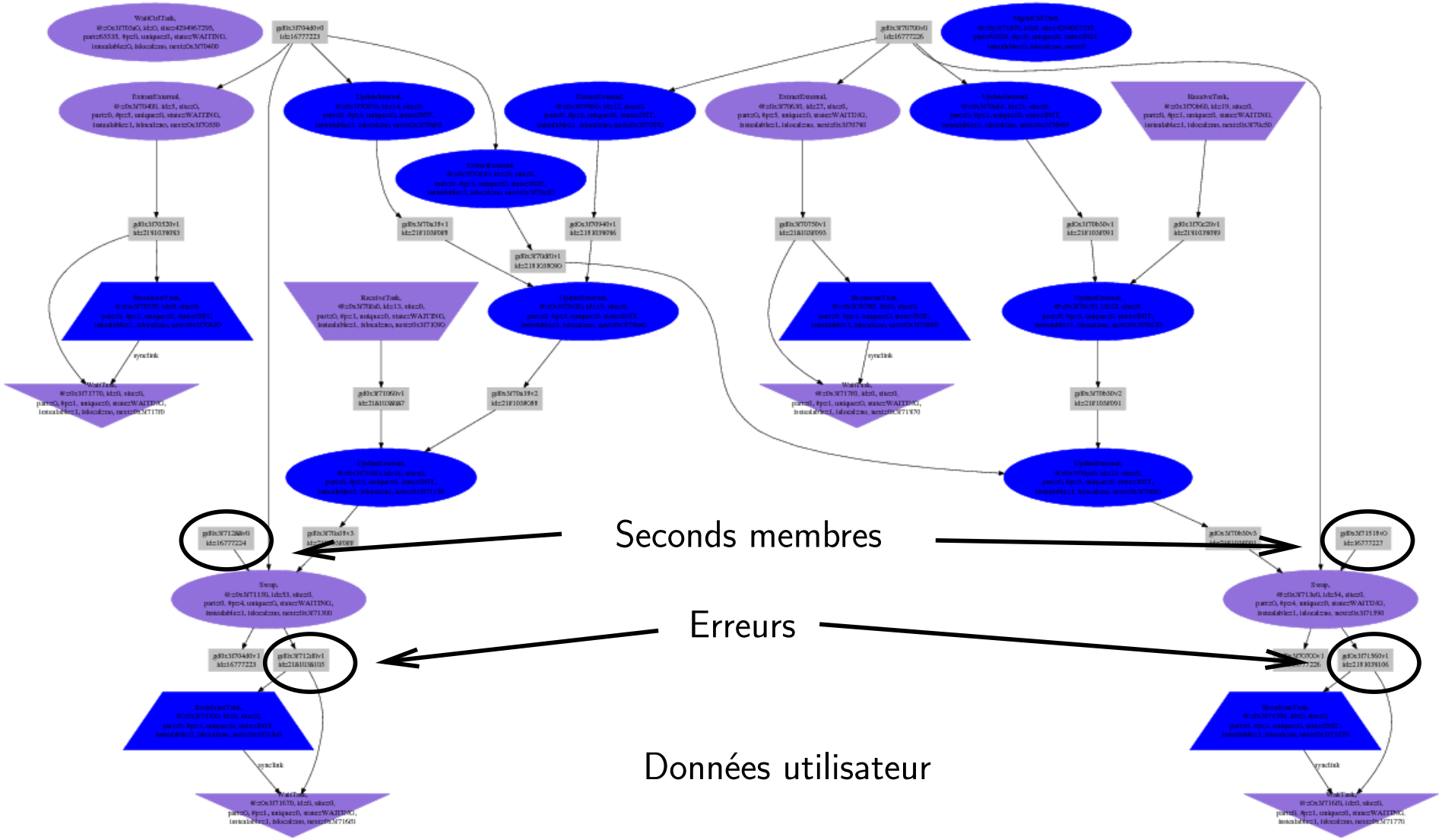
Application Jacobi3D : Graphe de flot de données réel

Graphe de flot de données généré pour le processus *N*â



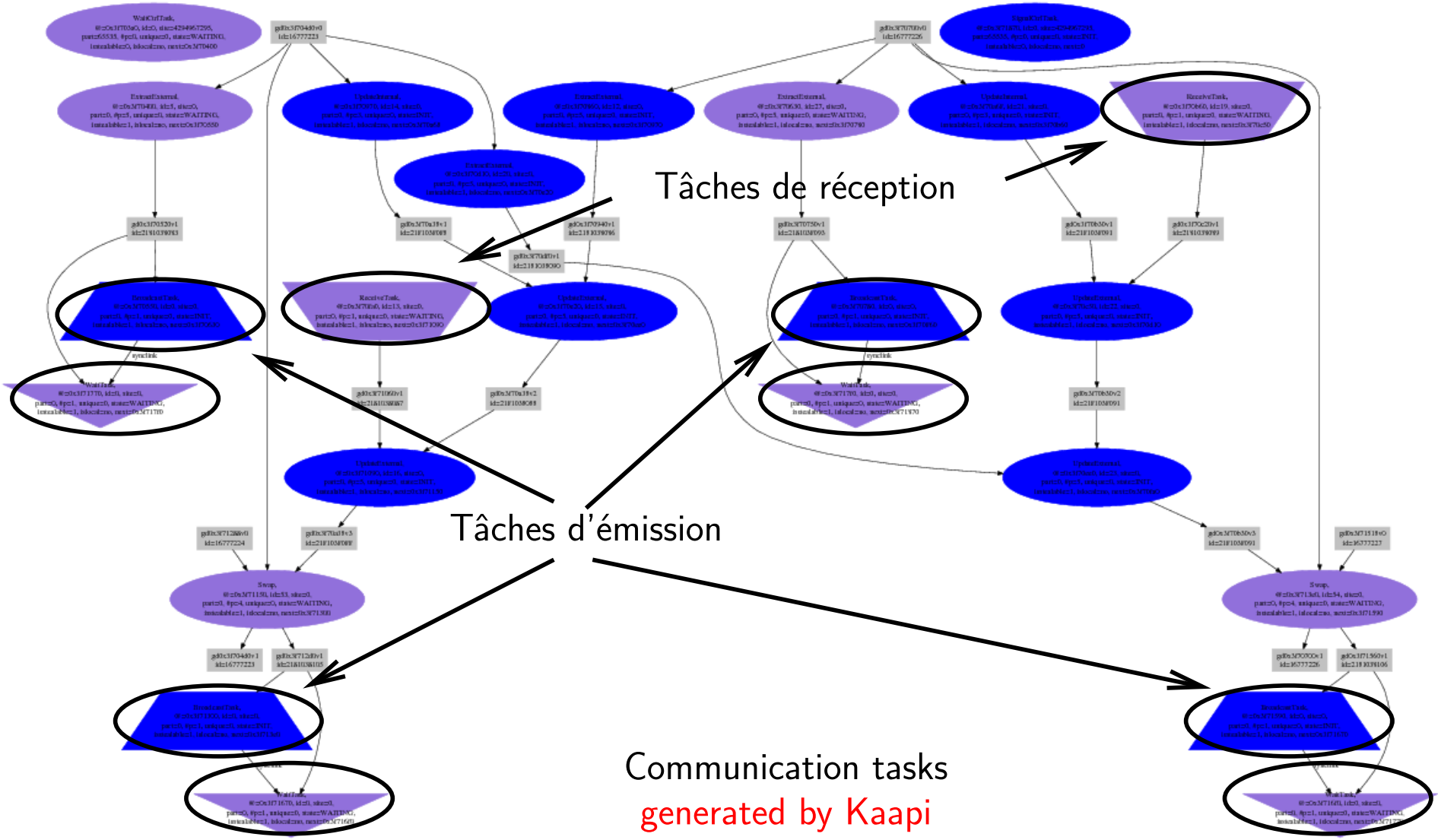
Application Jacobi3D : Graphe de flot de données réel

Graphe de flot de données généré pour le processus *N*â



Application Jacobi3D : Graphe de flot de données réel

Graphe de flot de données généré pour le processus *N*â



Annexes

◂ Plan de la présentation

Sur-décomposition pour Jacobi3D

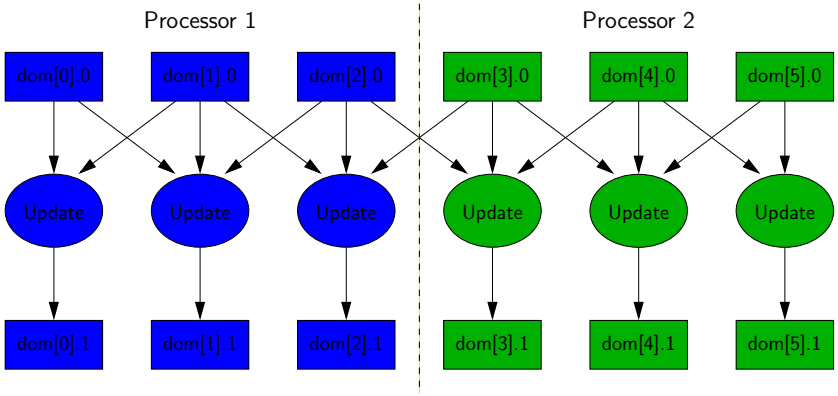
Nombre de machines : n, nombre de sous-domaines : d

- Décomposition classique (MPI) : $n = d$
- Sur-décomposition : $d \gg n$

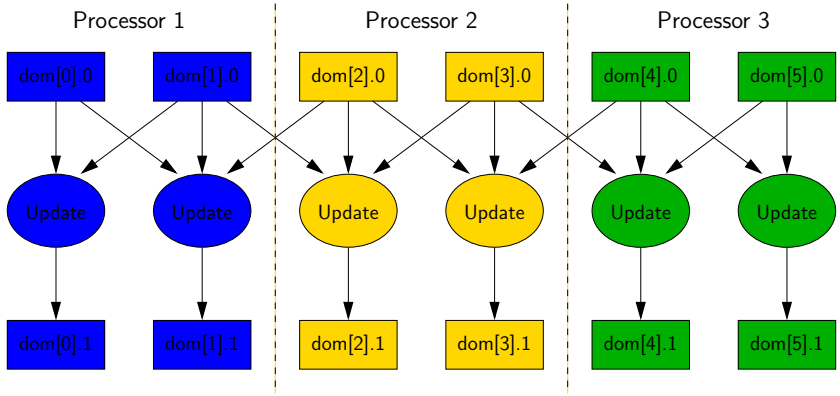
$\Rightarrow$ La sur-décomposition permet d'être indépendant du nombre de processeurs

Exemple : "Sur"-décomposition en 6 sous-domaines

Distribution sur 2 processeurs

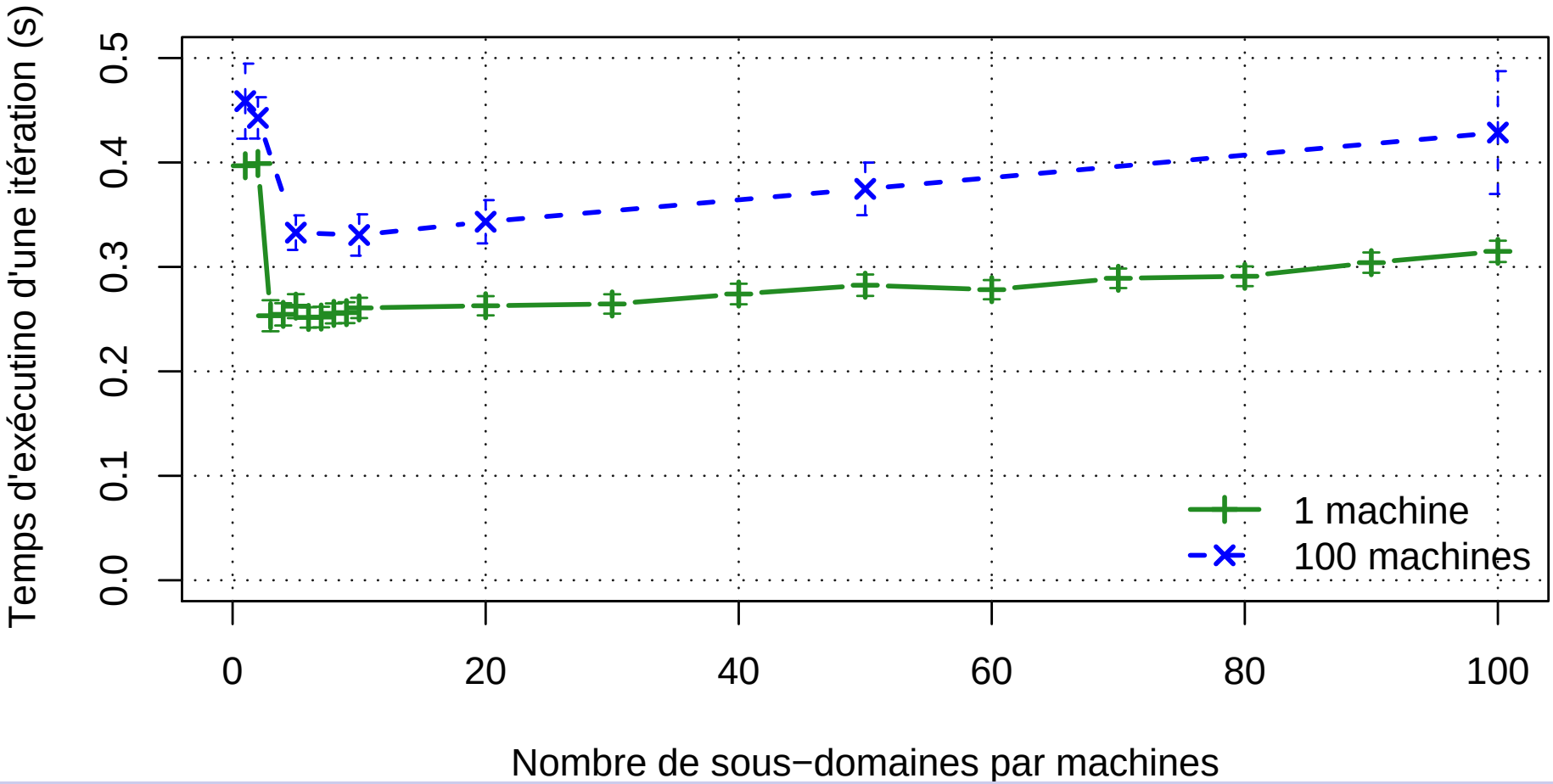


Distribution sur 3 processeurs



Expérience : Influence de la sur-décomposition

- Temps d'exécution en fonction de la décomposition d par machine
- Domaine en 3D, taille constante par machine : 10^7 réels de type `double`
  - Sur 1 machine : 10^7 réels, *i.e.* $\approx$ 76 Mo
  - Sur 100 machines : 100×10^7 réels, *i.e.* $\approx$ 7,6 Go
- Grappe Griffon de Nancy



L'influence de la sur-décomposition : Modélisation

Soit T_n^d le temps d'exécution d'une itération pour

- une décomposition en d sous-domaines
- en utilisant n processeurs

- Temps d'exécution $T_n^d = \lceil \frac{d}{n} \rceil \times \frac{T_1^1}{d}$
- Temps optimal T_n^n pour $d = n$
- Le surcout de la sur-décomposition est

$$T_n^d / T_n^n = \left\lceil \frac{d}{n} \right\rceil \times \frac{n}{d} \leq 1 + \frac{n}{d}$$

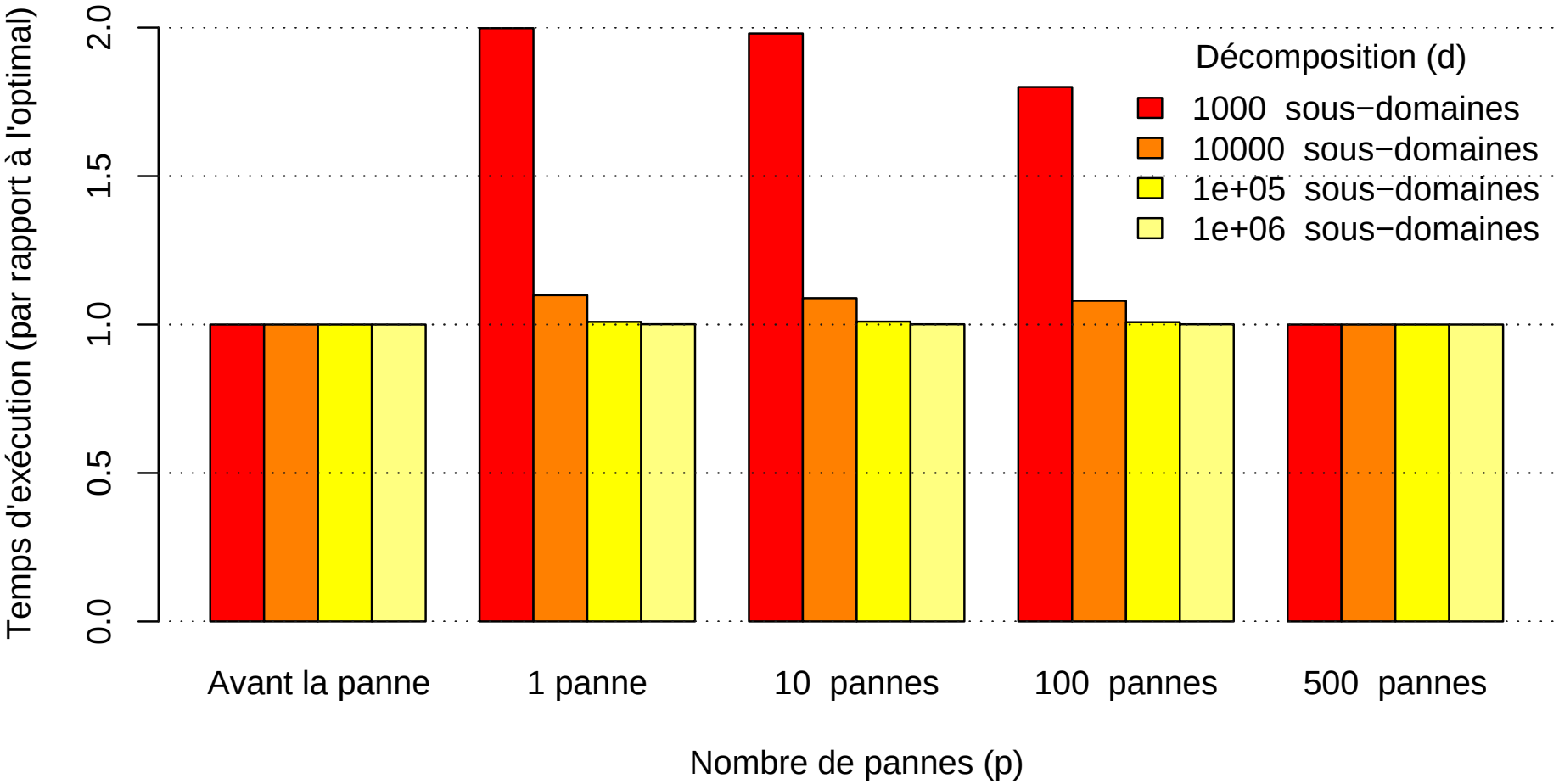
Après la reprise globale et le rééquilibrage de charge

- p est le nombre de machines défaillantes
- Après la panne, le surcout de la sur-décomposition est

$$T_{n-p}^d / T_{n-p}^{n-p} = \left\lceil \frac{d}{n-p} \right\rceil \times \frac{n-p}{d} \leq 1 + \frac{n}{d}$$

L'influence de la sur-décomposition : Simulation

Simulation sur $1000 - p$ processeurs

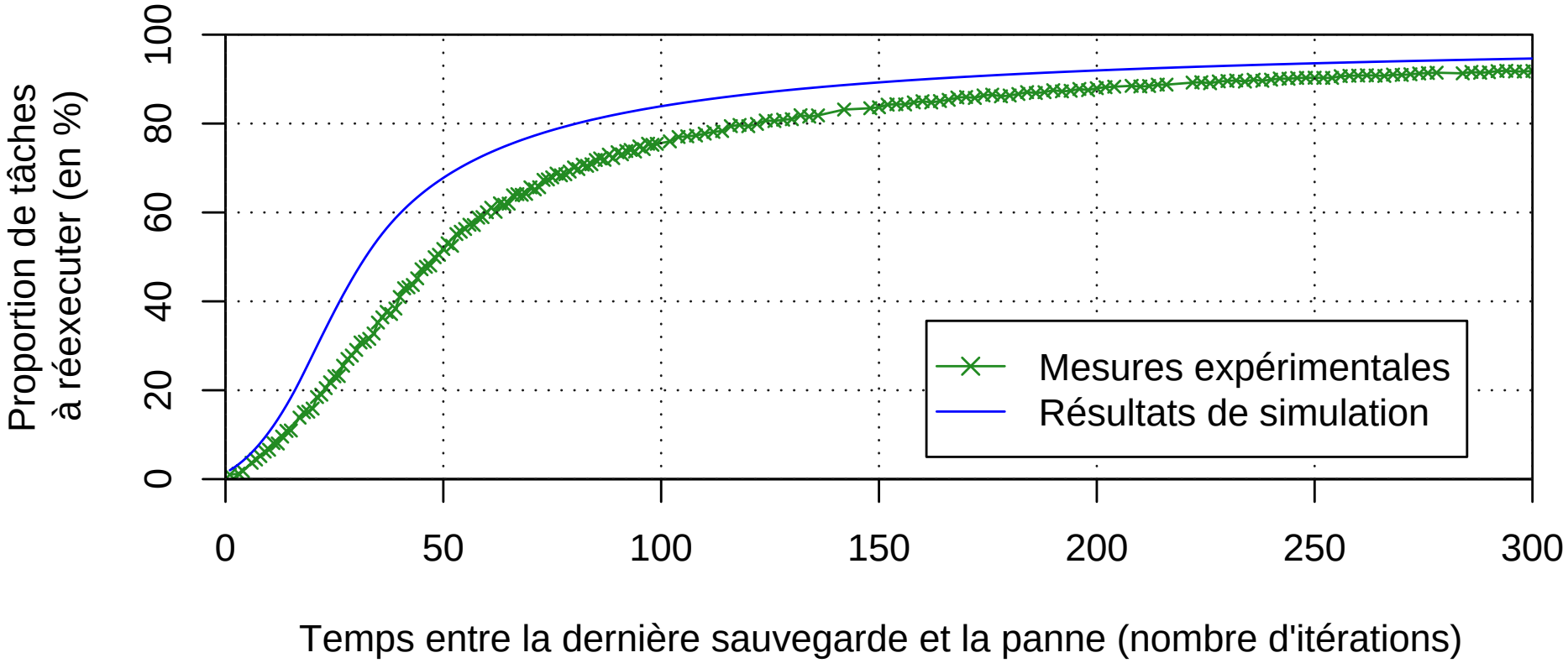


Annexes

◂ Plan de la présentation

Proportion des tâches à réexécuter

- Jacobi3D exécuté sur 100 machines
- $40 \times 40 \times 1$ sous-domaines, *i.e.* 16 sous-domaines par machine
- Panne d'une machine fixée



Dépend de la « forme » du graphe ($\sim$ taille du domaine)

Annexes

◂ Plan de la présentation