Quand des erreurs se produisent dans les supercalculateurs...

Anne Benoit

LIP, Ecole Normale Supérieure de Lyon, France
Equipe ROMA

Anne.Benoit@ens-lyon.fr
http://graal.ens-lyon.fr/~abenoit/

Café (gourmand et) scientifique du LIP, 10 décembre 2020

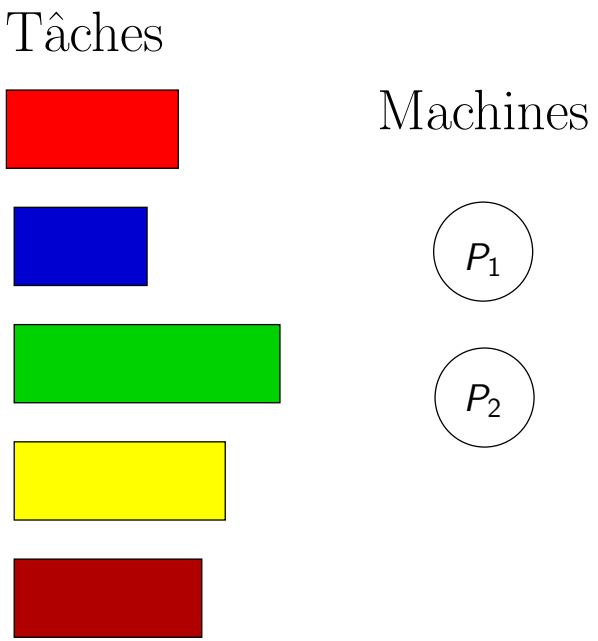
Motivation: Ordonnancement

Mais que fait-on sur ces super-calculateurs?

Ordonnancement: Allouer des ressources à des applications afin d'optimiser des critères de performance

- Ressources: Systèmes à grande échelle, avec des millions de composants
- Applications: Applications parallèles, constituées d'un ensemble de tâches, ou alors une grosse application avec une certaine quantité de travail à compléter
- Critères de performance: Bien sûr on veut que l'exécution se termine vite, mais aussi que le résultat soit fiable, et que l'exécution consomme peu d'énergie

Problèmes d'ordonnancement classiques

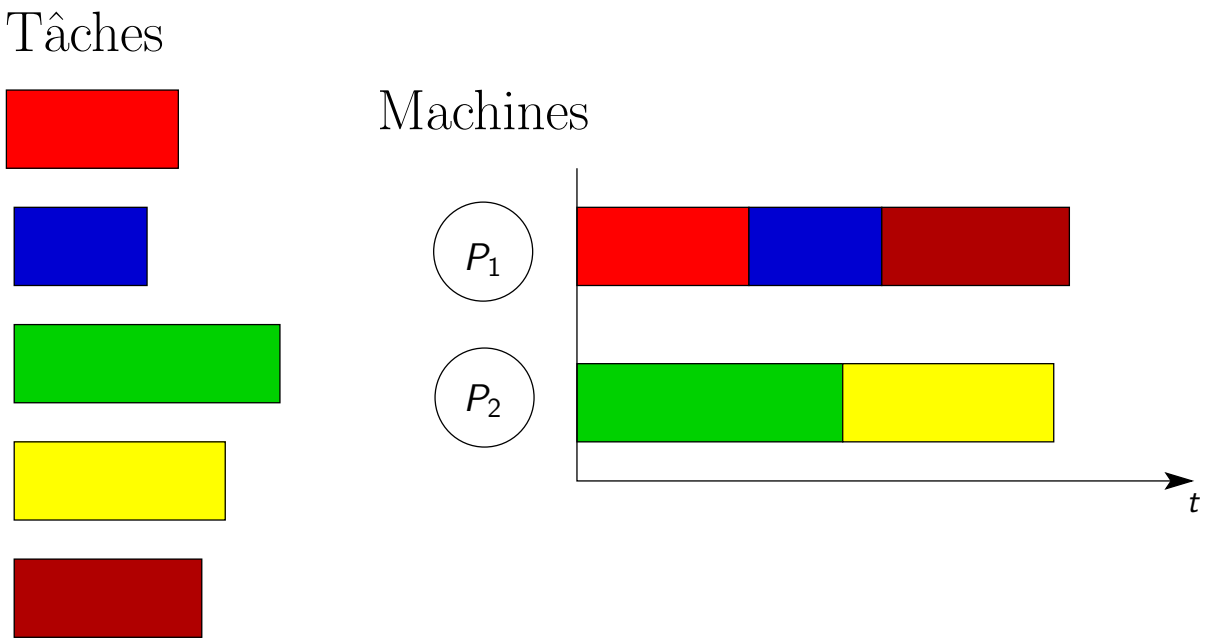


Objectifs:

- Minimiser le temps total d'exécution (*makespan*, C_{max})
- Minimiser une somme pondérée des temps d'exécution $\sum_i w_i C_i$

Résultats: NP-complétude (complexité du problème), algorithmes, bornes et garanties sur les algorithmes

Problèmes d'ordonnancement classiques

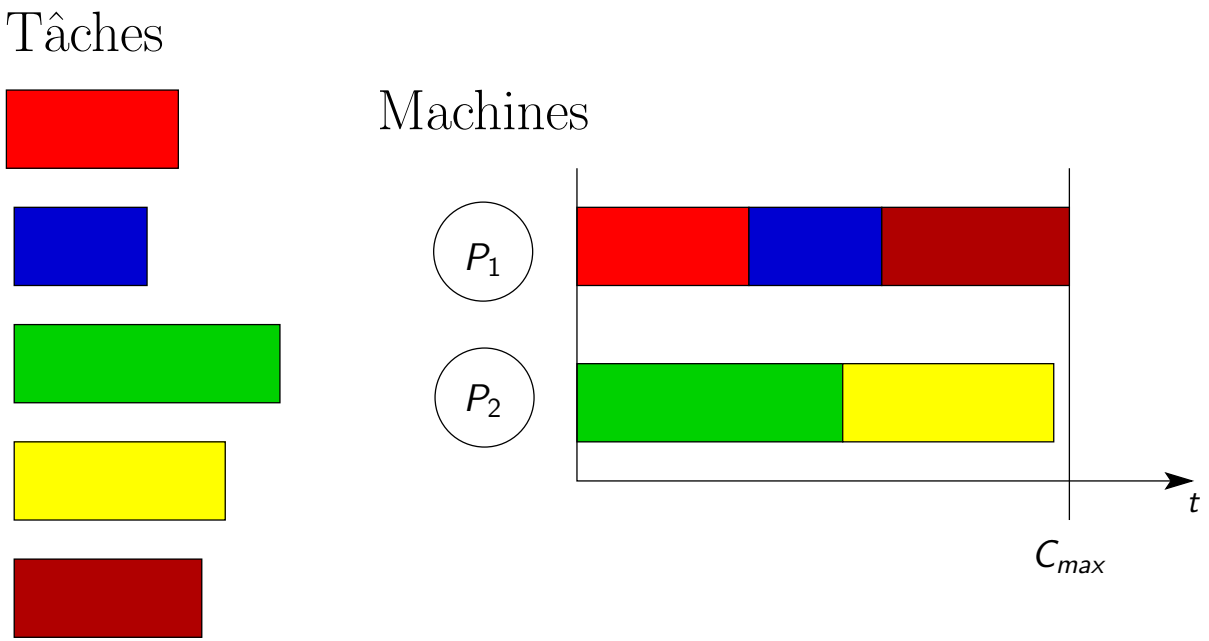


Objectifs:

- Minimiser le temps total d'exécution (*makespan*, C_{max})
- Minimiser une somme pondérée des temps d'exécution $\sum_i w_i C_i$

Résultats: NP-complétude (complexité du problème), algorithmes, bornes et garanties sur les algorithmes

Problèmes d'ordonnancement classiques

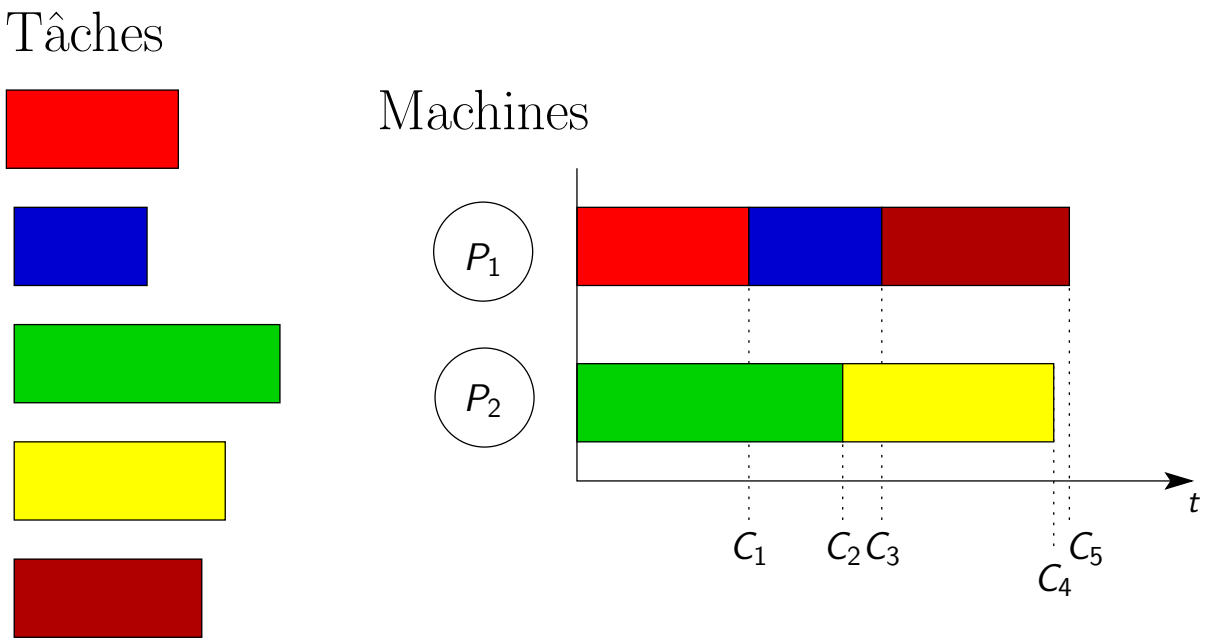


Objectifs:

- Minimiser le temps total d'exécution (*makespan*, C_{max})
- Minimiser une somme pondérée des temps d'exécution $\sum_i w_i C_i$

Résultats: NP-complétude (complexité du problème), algorithmes, bornes et garanties sur les algorithmes

Problèmes d'ordonnancement classiques

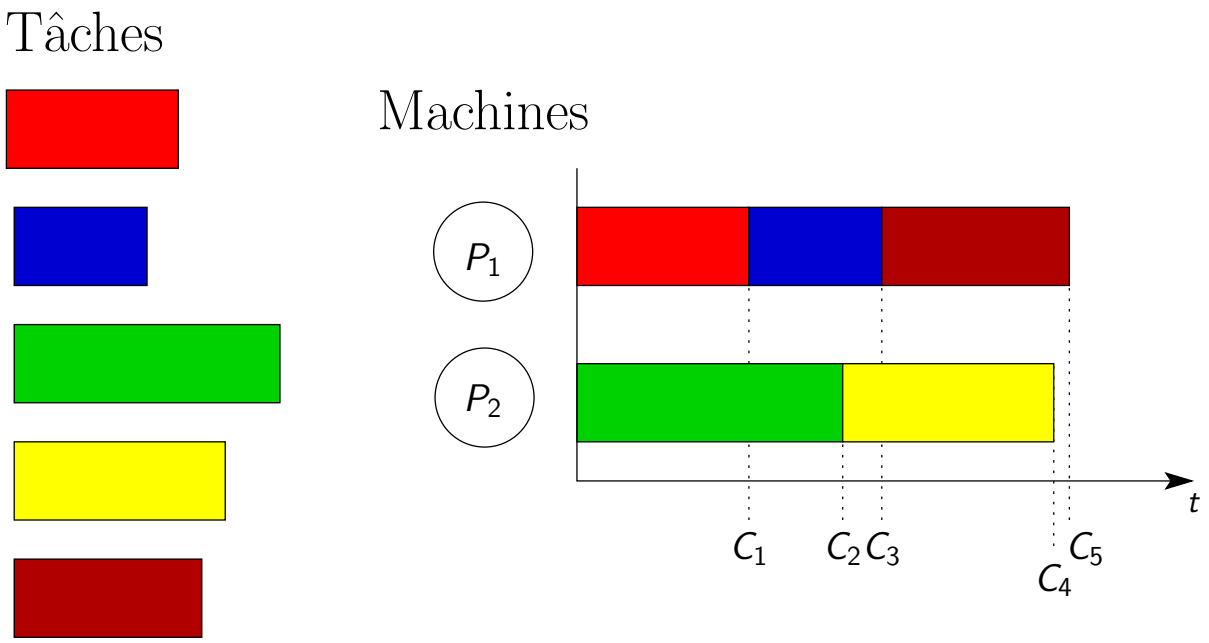


Objectifs:

- Minimiser le temps total d'exécution (*makespan*, C_{max})
- Minimiser une somme pondérée des temps d'exécution $\sum_i w_i C_i$

Résultats: NP-complétude (complexité du problème), algorithmes, bornes et garanties sur les algorithmes

Problèmes d'ordonnancement classiques



Objectifs:

- Minimiser le temps total d'exécution (*makespan*, C_{max})
- Minimiser une somme pondérée des temps d'exécution $\sum_i w_i C_i$

Résultats: NP-complétude (complexité du problème), algorithmes, bornes et garanties sur les algorithmes

Sommaire

1 Pourquoi des erreurs?

2 Points de sauvegarde

3 Réplication

4 Conclusion

Gérer les erreurs/fautes

- Prenons un processeur (par exemple votre ordinateur personnel)
  - Temps moyen entre deux fautes: Mean Time Between Failures (MTBF) = 100 ans (en étant optimiste!)
  - (Presque) pas de fautes en pratique ☺

  Pourquoi se soucier des erreurs?

- **Théorème:** Le MTBF décroît linéairement avec le nombre de processeurs! Avec 36500 processeurs:
  - MTBF $\mu = 1$ jour
  - Une faute par jour en moyenne!

Une grande simulation peut tourner pendant plusieurs semaines, ainsi elle sera confrontée à des erreurs ☹

Gérer les erreurs/fautes

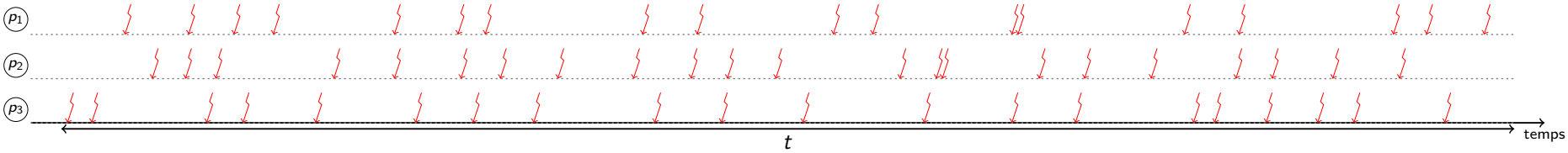
- Prenons un processeur (par exemple votre ordinateur personnel)
  - Temps moyen entre deux fautes: Mean Time Between Failures (MTBF) = 100 ans (en étant optimiste!)
  - (Presque) pas de fautes en pratique ☺

  Pourquoi se soucier des erreurs?

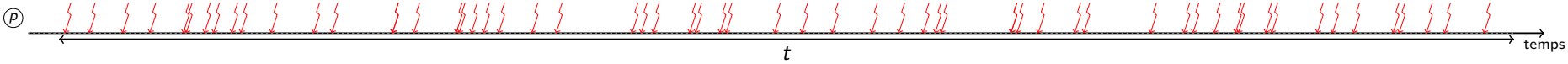
- **Théorème:** Le MTBF décroît linéairement avec le nombre de processeurs! Avec 36500 processeurs:
  - MTBF $\mu = 1$ jour
  - Une faute par jour en moyenne!

Une grande simulation peut tourner pendant plusieurs semaines, ainsi elle sera confrontée à des erreurs ☹

Intuition



Si 3 processeurs ont 20 fautes pendant un temps t ($\mu_{ind} = \frac{t}{20}$)...



...pendant le même temps, la plate-forme a 60 fautes ($\mu_p = \frac{t}{60}$)

Différents types de fautes

- Erreurs fatales:
  - Composant fautif (noeud de calcul, réseau, alimentation, ...)
  - L'application plante, les données sont perdues
- Erreurs silencieuses:
  - Flip de bit (Disque, RAM, Cache, Bus, ...)
  - La détection n'est pas instantanée, le résultat peut être faux

Alors, comment gérer les erreurs?

En général, on rajoute de la redondance à l'exécution:

- S'il y a une faute, on ré-exécute la tâche fautive (*encore faut-il savoir qu'il y a eu une faute, dans le cas d'erreurs silencieuses!*)
- *Pour une grosse application, c'est trop coûteux de repartir de zéro!* Idée: prendre des points de sauvegarde de l'application (ce qu'on appelle un *checkpoint*). Sauvegarder périodiquement l'état de l'application sur un support fiable, ainsi on peut repartir de là où on en était sans avoir tout perdu. C'est ce que vous faites (normalement ☺) avec votre ordinateur!
- *Pour être encore plus fiable?* Répliquer le travail (on travaille plus pour assurer, par exemple on utilise seulement la moitié de nos processeurs, l'autre moitié effectue le même travail)

Alors, comment gérer les erreurs?

En général, on rajoute de la redondance à l'exécution:

- S'il y a une faute, on ré-exécute la tâche fautive (*encore faut-il savoir qu'il y a eu une faute, dans le cas d'erreurs silencieuses!*)
- *Pour une grosse application, c'est trop coûteux de repartir de zéro!* Idée: prendre des points de sauvegarde de l'application (ce qu'on appelle un *checkpoint*). Sauvegarder périodiquement l'état de l'application sur un support fiable, ainsi on peut repartir de là où on en était sans avoir tout perdu. C'est ce que vous faites (normalement ☺) avec votre ordinateur!
- *Pour être encore plus fiable?* Répliquer le travail (on travaille plus pour assurer, par exemple on utilise seulement la moitié de nos processeurs, l'autre moitié effectue le même travail)

Alors, comment gérer les erreurs?

En général, on rajoute de la redondance à l'exécution:

- S'il y a une faute, on ré-exécute la tâche fautive (*encore faut-il savoir qu'il y a eu une faute, dans le cas d'erreurs silencieuses!*)
- *Pour une grosse application, c'est trop coûteux de repartir de zéro!* Idée: prendre des points de sauvegarde de l'application (ce qu'on appelle un *checkpoint*). Sauvegarder périodiquement l'état de l'application sur un support fiable, ainsi on peut repartir de là où on en était sans avoir tout perdu. C'est ce que vous faites (normalement ☺) avec votre ordinateur!
- *Pour être encore plus fiable?* Répliquer le travail (on travaille plus pour assurer, par exemple on utilise seulement la moitié de nos processeurs, l'autre moitié effectue le même travail)

Sommaire

1 Pourquoi des erreurs?

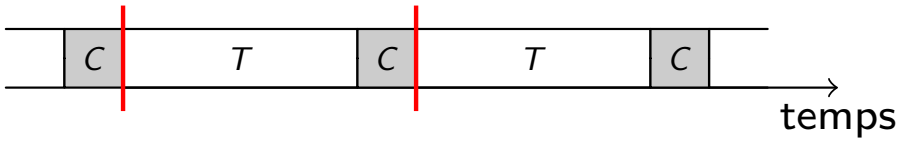
2 Points de sauvegarde

3 Réplication

4 Conclusion

Gérer les erreurs fatales

Points de sauvegarde périodique, retour en arrière et reprise du travail:



(pas d'erreur)

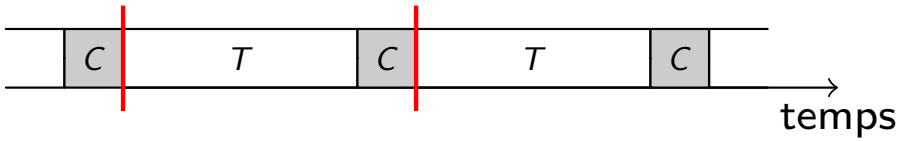
(erreur)

(erreur)

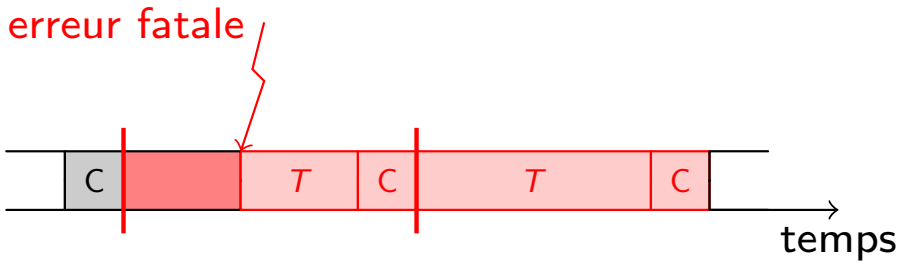
- Checkpoint coordonné (plate-forme: gros processeur)
- Interruption de l'application et détection instantanées
- Retour au dernier checkpoint et ré-exécution

Gérer les erreurs fatales

Points de sauvegarde périodique, retour en arrière et reprise du travail:



(pas d'erreur)



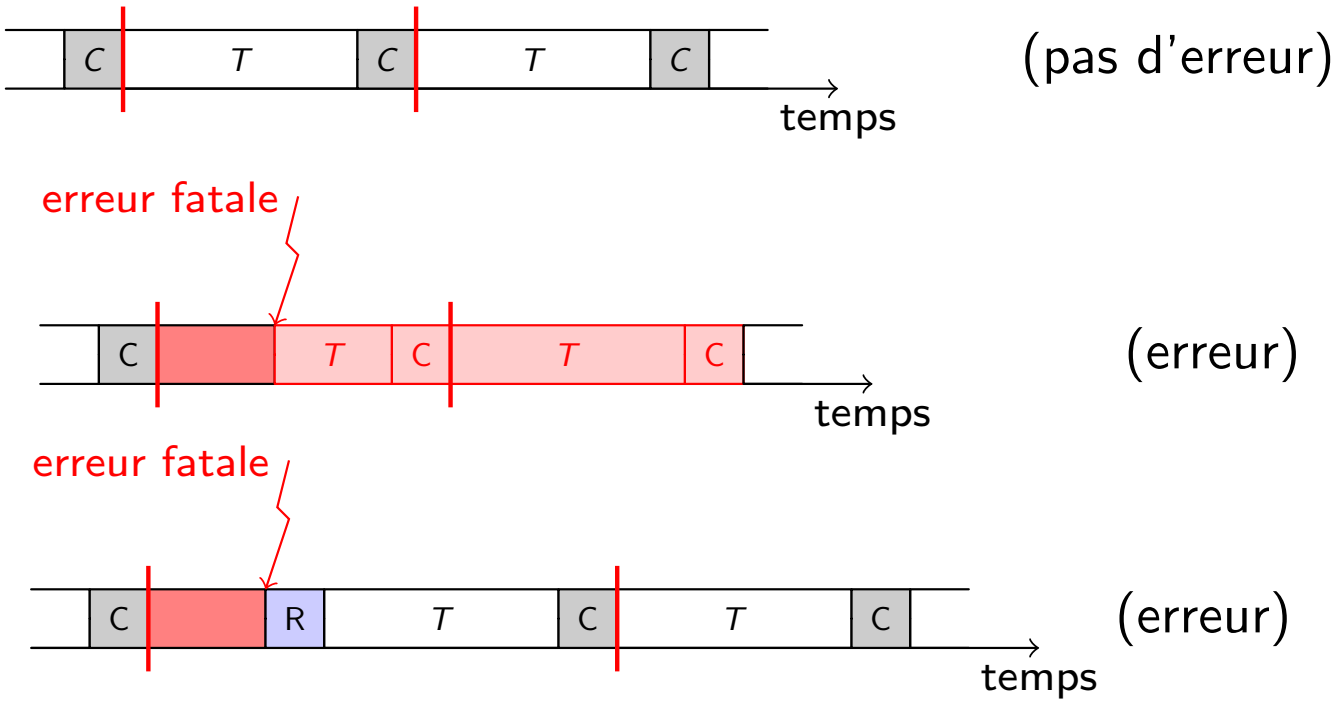
(erreur)

(erreur)

- Checkpoint coordonné (plate-forme: gros processeur)
- Interruption de l'application et détection instantanées
- Retour au dernier checkpoint et ré-exécution

Gérer les erreurs fatales

Points de sauvegarde périodique, retour en arrière et reprise du travail:



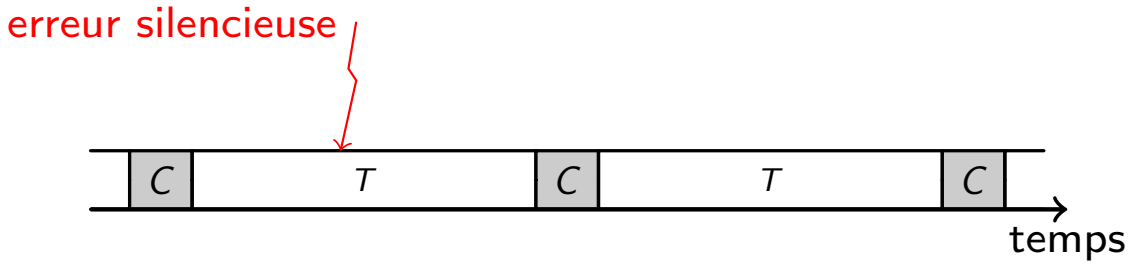
- Checkpoint coordonné (plate-forme: gros processeur)
- Interruption de l'application et détection instantanées
- Retour au dernier checkpoint et ré-exécution

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

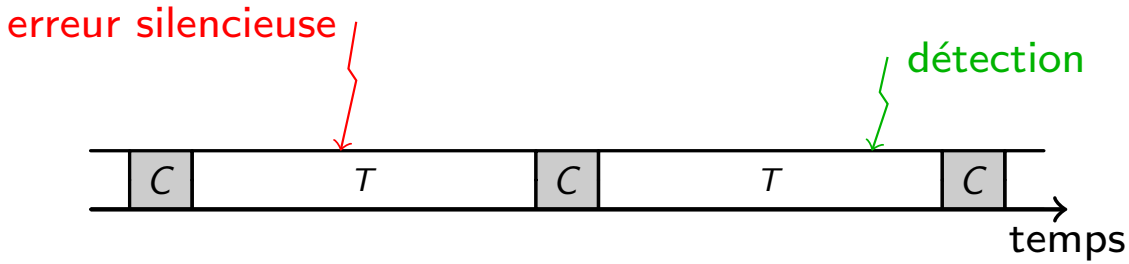
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

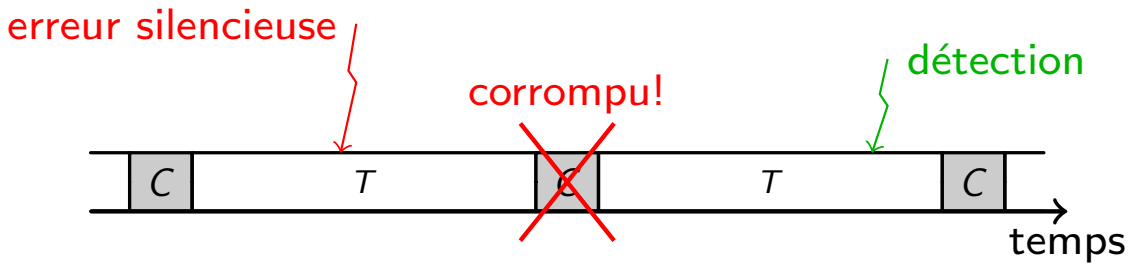
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

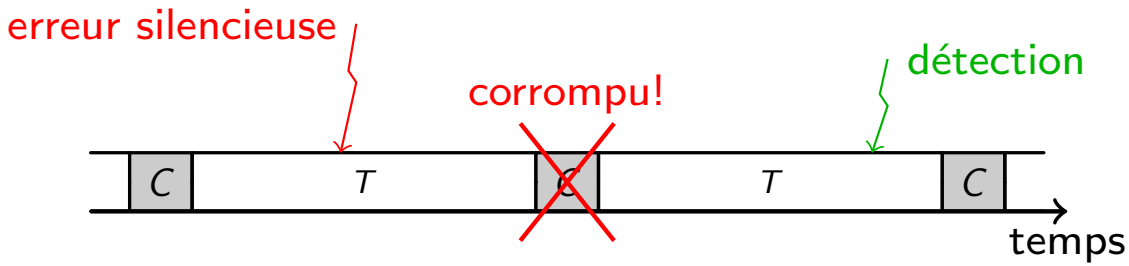
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

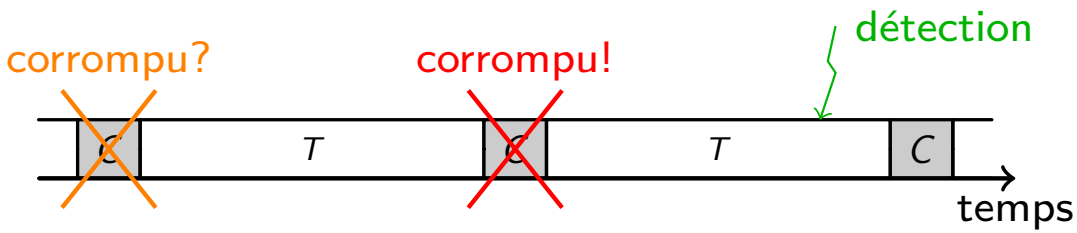
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

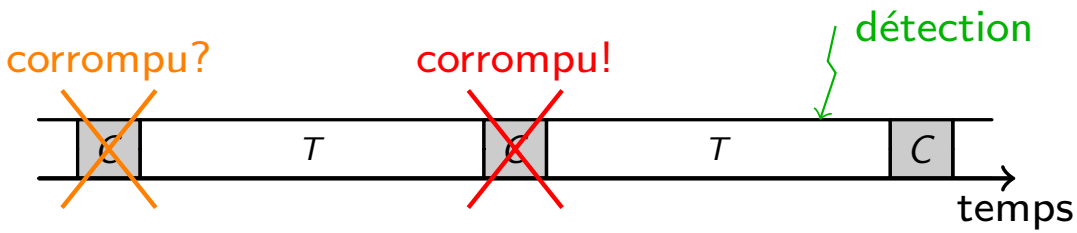
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?



Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

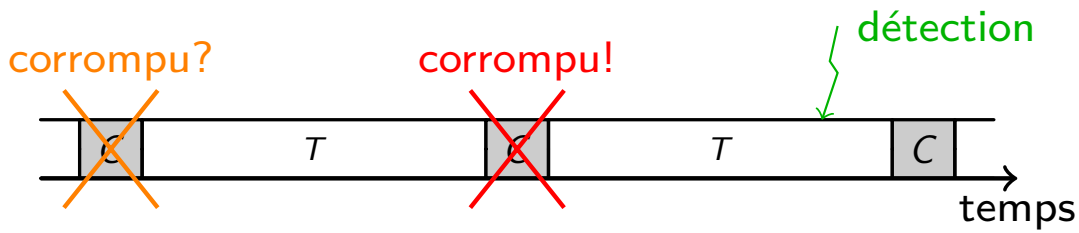
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec les erreurs silencieuses?

Erreur silencieuse = on ne la détecte pas instantanément

Détectée lorsque la donnée corrompue est activée

Même approche?

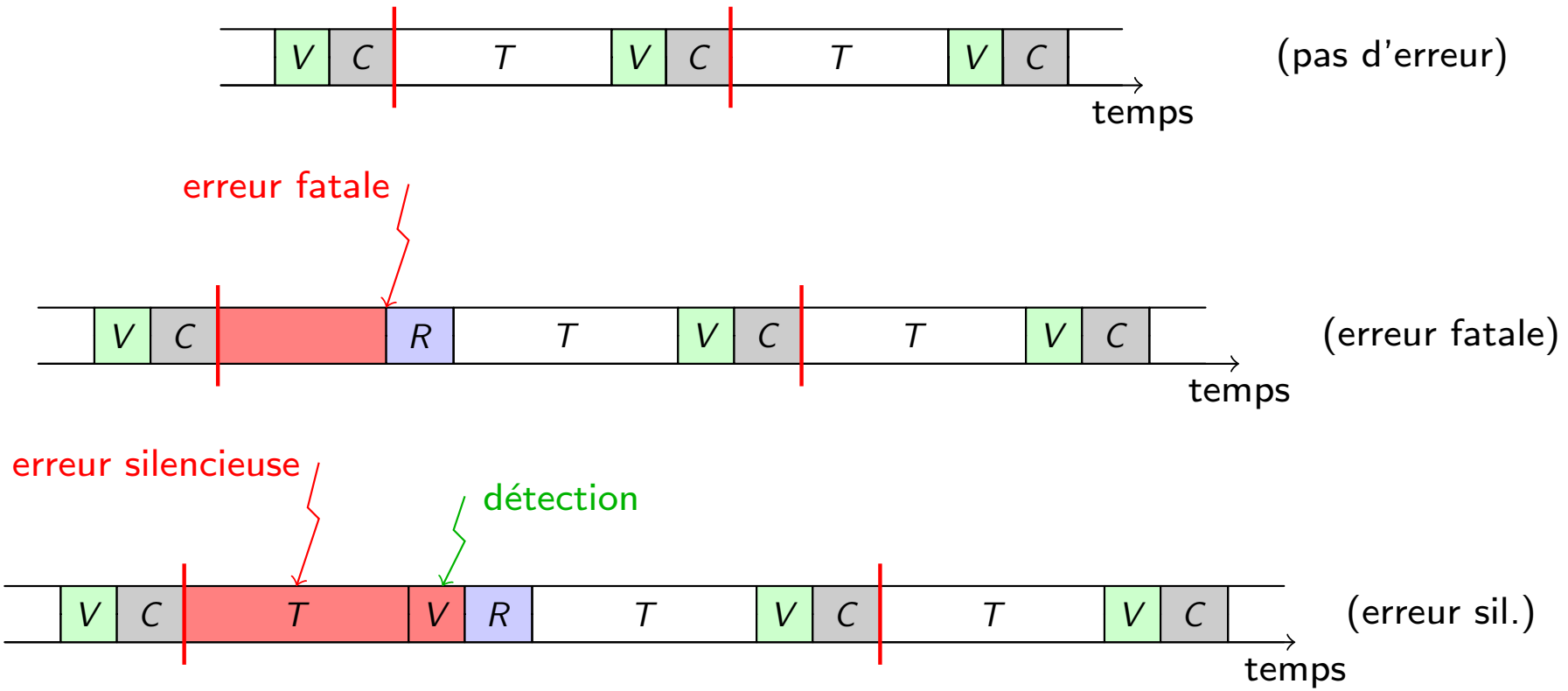


Garder plusieurs checkpoints?

A partir de quel checkpoint repartir?

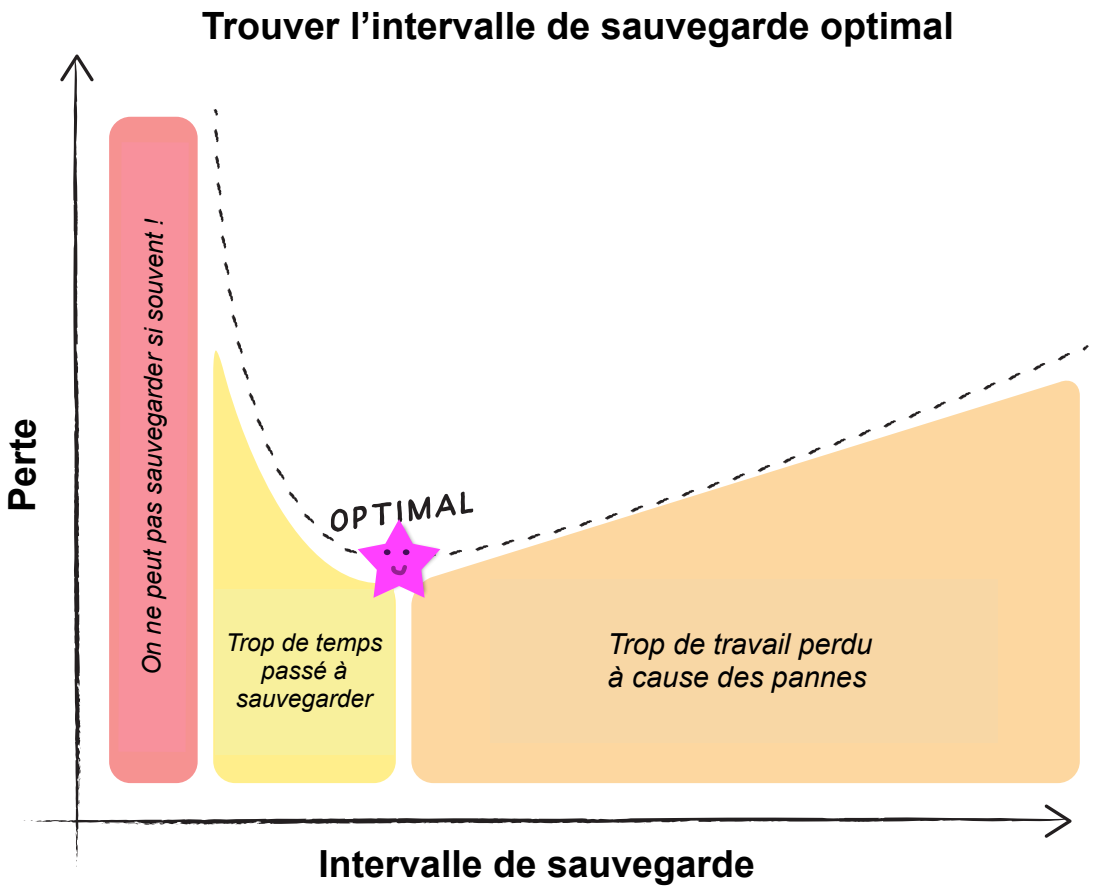
Besoin d'une méthode pour détecter les erreurs silencieuses!

Avec des erreurs fatales et silencieuses

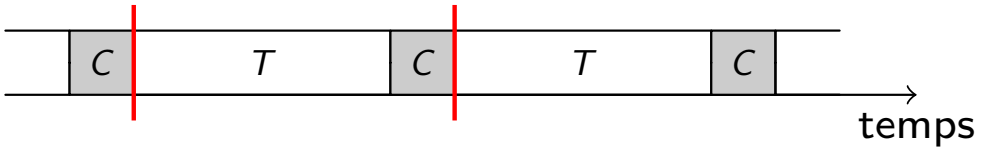


Quelle est la période optimale de checkpoint?

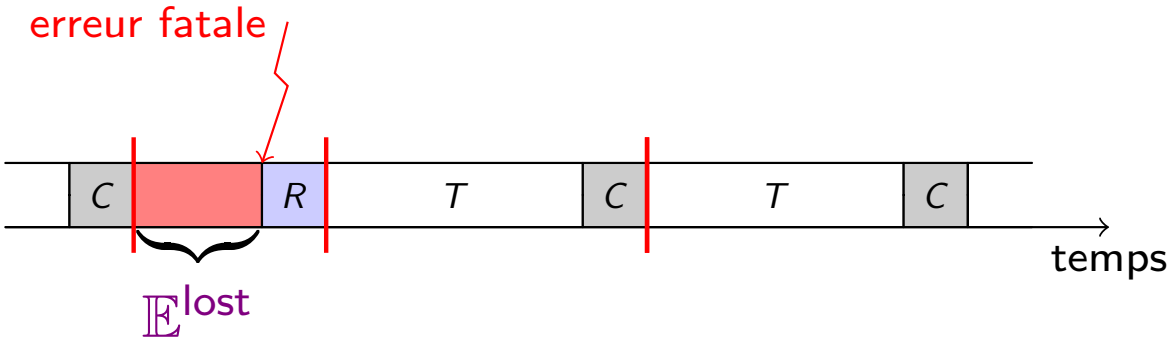
Période optimale de checkpoint



Période optimale de checkpoint



(pas d'erreur)



(reprise)

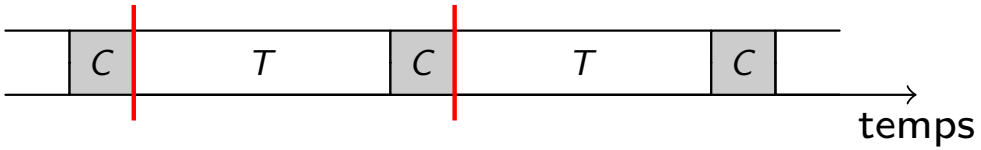
$$\mathbb{E}(T) = \mathbb{P}_{no-error}\,(T + C) + \mathbb{P}_{error}\left(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(T)\right)$$

$$\mathbb{E}^{\text{lost}} = \frac{T}{2} + o(\lambda^f T) \text{ (on perd } \frac{T}{2} \text{ en moyenne)}$$

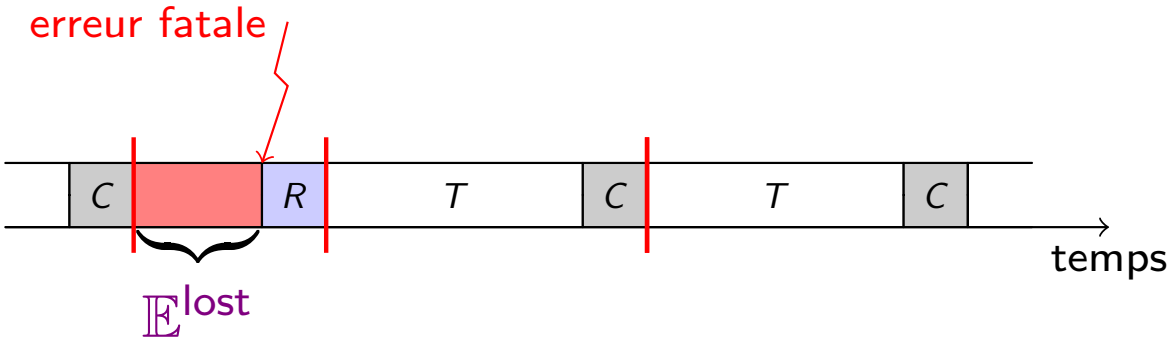
On obtient finalement:

$$T^* = \sqrt{\frac{2C}{\lambda^f}} = \sqrt{2\mu^f C}$$

Période optimale de checkpoint



(pas d'erreur)



(reprise)

$$\mathbb{E}(T) = \mathbb{P}_{no-error}(T + C) + \mathbb{P}_{error}\left(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(T)\right)$$

$$\mathbb{E}^{\text{lost}} = \frac{T}{2} + o(\lambda^f T) \text{ (on perd } \frac{T}{2} \text{ en moyenne)}$$

On obtient finalement:

$$T^* = \sqrt{\frac{2C}{\lambda^f}} = \sqrt{2\mu^f C}$$

Période optimale avec deux types d'erreurs

$$\text{Surcoût } \mathbb{H}(T) = \frac{C+V}{T} + \underbrace{\lambda^f \frac{T}{2}}_{\textit{fatales}} + \underbrace{\lambda^s T}_{\textit{silencieuses}} + o(\lambda T)$$

Approximations du premier ordre [Young 1974, Daly 2006, AB et al. 2016]

	Erreurs fatales	Erreurs silencieuses	Deux types d'erreurs
Schéma	$T + C$	$T + V + C$	$T + V + C$
Période optimale T^*	$\sqrt{\frac{C}{\frac{\lambda^f}{2}}}$	$\sqrt{\frac{V+C}{\lambda^s}}$	$\sqrt{\frac{V+C}{\lambda^s + \frac{\lambda^f}{2}}}$

Période optimale avec deux types d'erreurs

$$\text{Surcoût } \mathbb{H}(T) = \frac{C+V}{T} + \underbrace{\lambda^f \frac{T}{2}}_{\textit{fatales}} + \underbrace{\lambda^s T}_{\textit{silencieuses}} + o(\lambda T)$$

Approximations du premier ordre [Young 1974, Daly 2006, AB et al. 2016]

	Erreurs fatales	Erreurs silencieuses	Deux types d'erreurs
Schéma	$T + C$	$T + V + C$	$T + V + C$
Période optimale T^*	$\sqrt{\frac{C}{\frac{\lambda^f}{2}}}$	$\sqrt{\frac{V+C}{\lambda^s}}$	$\sqrt{\frac{V+C}{\lambda^s + \frac{\lambda^f}{2}}}$

Sommaire

1 Pourquoi des erreurs?

2 Points de sauvegarde

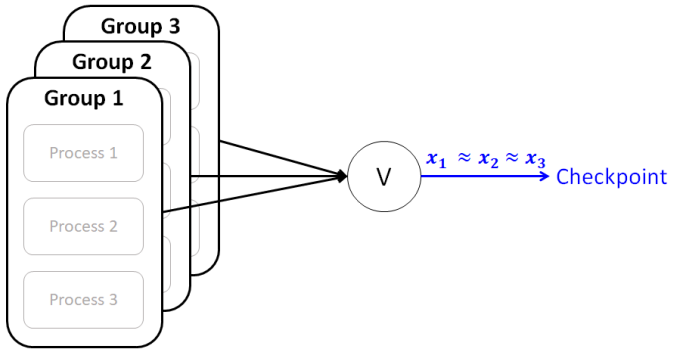
3 Réplication

4 Conclusion

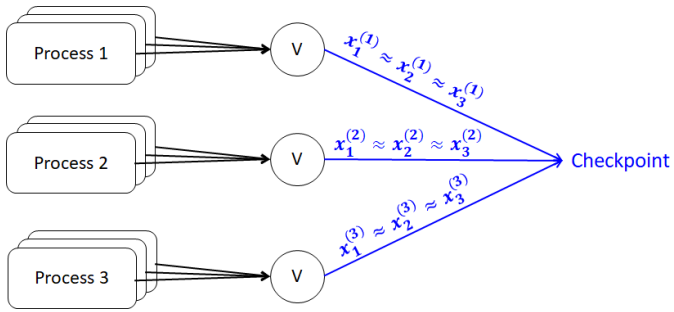
Comment procéder?

On veut effectuer le travail plusieurs fois sur des processeurs distincts: si une panne frappe sur un processeur, l'autre peut poursuivre.
Erreurs silencieuses: on peut les détecter/corriger!

- **Réplication de groupe**:



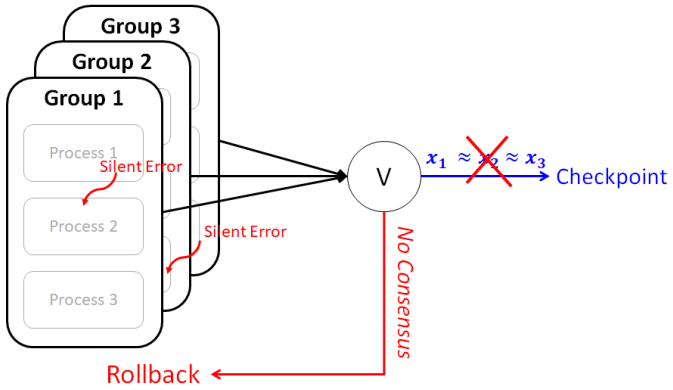
- **Réplication de processus**:



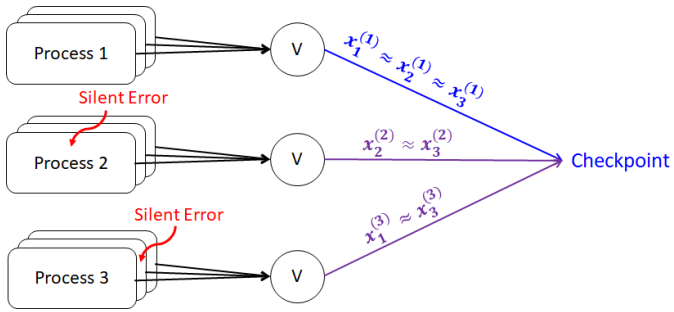
Comment procéder?

On veut effectuer le travail plusieurs fois sur des processeurs distincts: si une panne frappe sur un processeur, l'autre peut poursuivre.
Erreurs silencieuses: on peut les détecter/corriger!

- **Réplication de groupe**:



- **Réplication de processus**:



Alors quoi faire?

- **Réplication de groupe:** plus facile à implémenter en considérant que l'application est une boite noire
- **Réplication de processus:** plus résilient que la réplication de groupe (pour un même surcoût)

Modèle de Ferreira et al. [SC' 2011]

- Plate-forme avec $N = 2b$ processeurs arrangés en b paires
- Application parallèle avec b processus, chacun est répliqué
- Lorsqu'une faute touche un réplica, il n'est pas redémarré
- L'application plante lorsque les deux réplicas d'une même paire ont eu une erreur

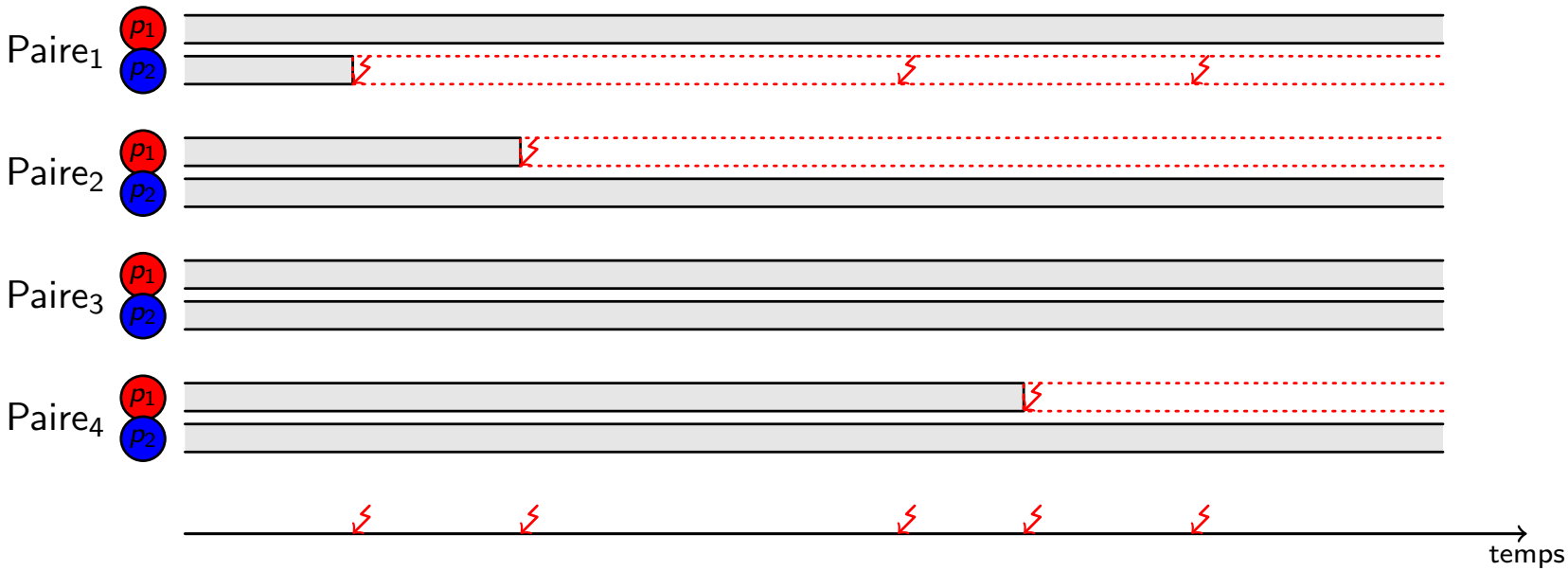
Alors quoi faire?

- **Réplication de groupe:** plus facile à implémenter en considérant que l'application est une boite noire
- **Réplication de processus:** plus résilient que la réplication de groupe (pour un même surcoût)

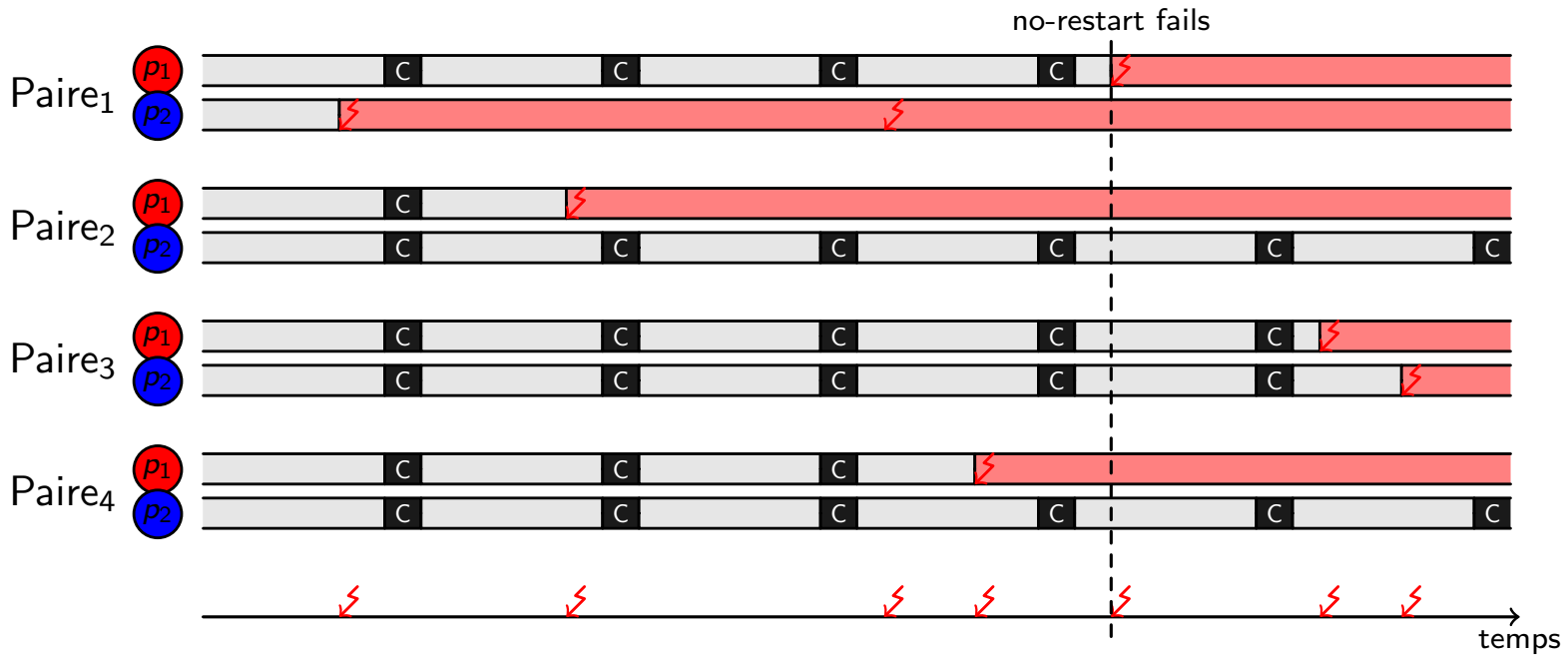
Modèle de Ferreira et al. [SC' 2011]

- Plate-forme avec $N = 2b$ processeurs arrangés en b paires
- Application parallèle avec b processus, chacun est répliqué
- Lorsqu'une faute touche un réplica, il n'est pas redémarré
- L'application plante lorsque les deux réplicas d'une même paire ont eu une erreur

Exemple

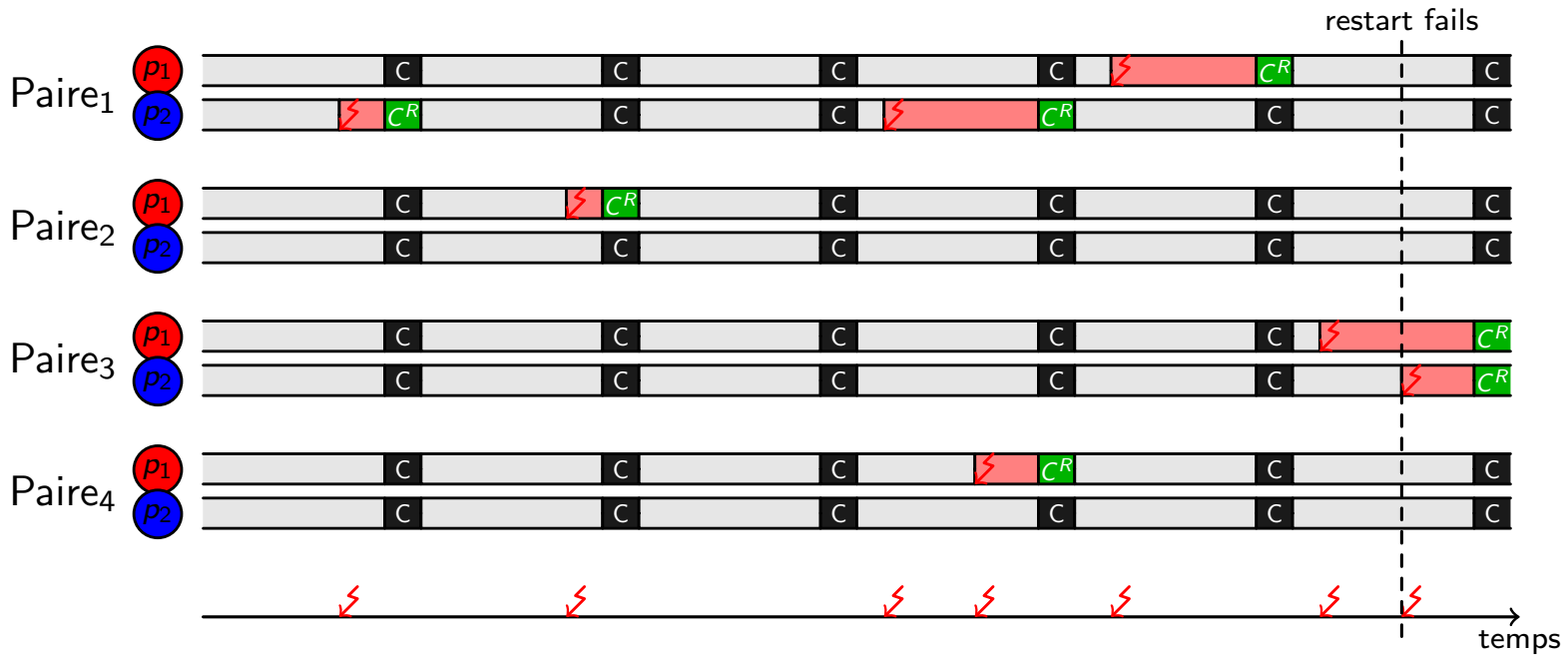


Avec des checkpoints: *no-restart* vs. *restart*



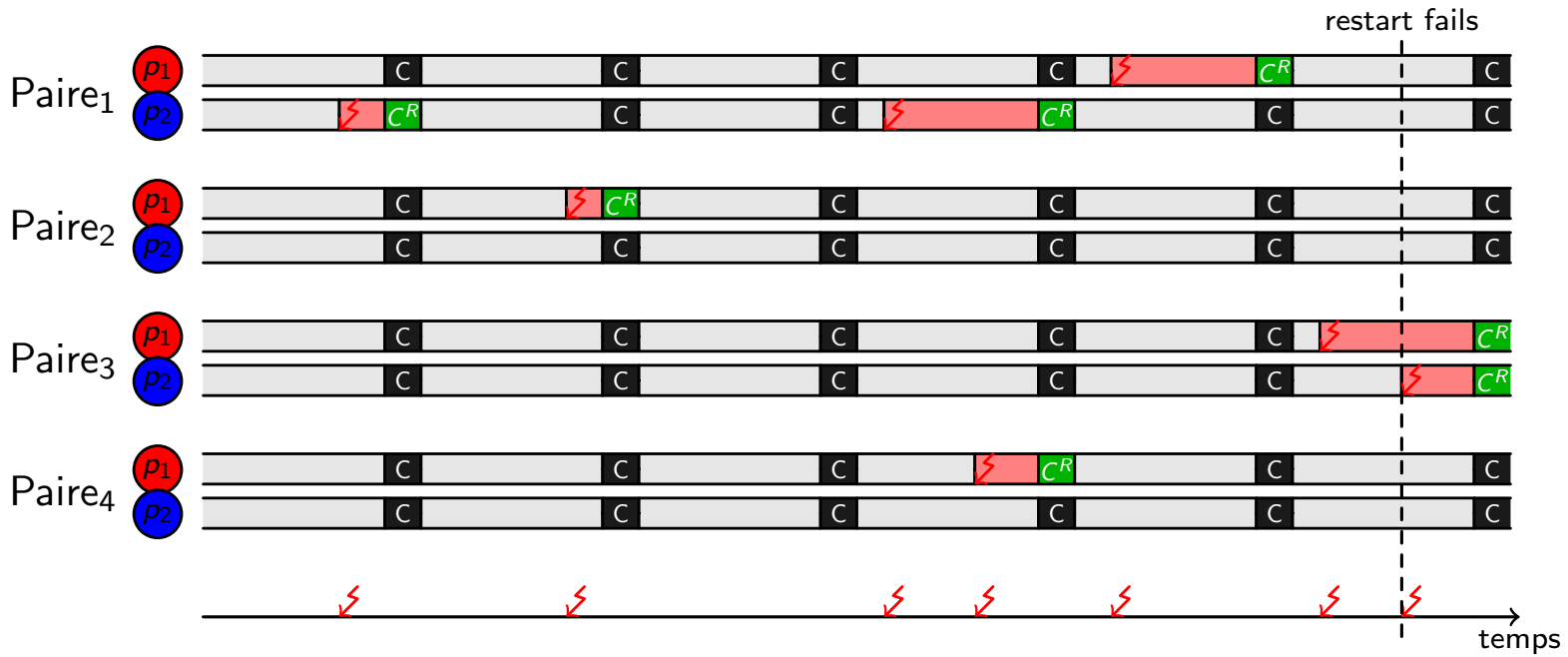
Peut-on faire mieux que ça?

Avec des checkpoints: *no-restart* vs. *restart*



Est-ce mieux de redémarrer à chaque checkpoint les proc. fautifs?

Avec des checkpoints: *no-restart* vs. *restart*



A quelle période faut-il checkpointer dans ce cas?

Sommaire

1 Pourquoi des erreurs?

2 Points de sauvegarde

3 Réplication

4 Conclusion

Conclusion

- Principaux défis des systèmes Exascale:
  - Résilience: besoin de gérer les fautes
  - Energie: réduire la consommation énergétique
- Objectif principal: souvent la performance (temps d'exécution)
- On a des réponses pour plein de modèles:
  - Période optimale de checkpoint, avec erreurs fatales/silencieuses
  - Utilisation de réplication pour détecter et corriger les erreurs silencieuses, et mélange avec le checkpoint (*restart*)
  - Période optimale de checkpoint pour optimiser la consommation énergétique?
  - Vitesse de ré-exécution différente après une faute?
- Encore plein de défis à adresser, et des techniques à développer pour plein d'applications haute performance, en faisant des compromis entre performance, fiabilité, et consommation énergétique ☺

Conclusion

- Principaux défis des systèmes Exascale:
  - Résilience: besoin de gérer les fautes
  - Energie: réduire la consommation énergétique
- Objectif principal: souvent la performance (temps d'exécution)
- On a des réponses pour plein de modèles:
  - Période optimale de checkpoint, avec erreurs fatales/silencieuses
  - Utilisation de réplication pour détecter et corriger les erreurs silencieuses, et mélange avec le checkpoint (*restart*)
  - Période optimale de checkpoint pour optimiser la consommation énergétique?
  - Vitesse de ré-exécution différente après une faute?
- Encore plein de défis à adresser, et des techniques à développer pour plein d'applications haute performance, en faisant des compromis entre performance, fiabilité, et consommation énergétique ☺

Conclusion

- Principaux défis des systèmes Exascale:
  - Résilience: besoin de gérer les fautes
  - Energie: réduire la consommation énergétique
- Objectif principal: souvent la performance (temps d'exécution)
- On a des réponses pour plein de modèles:
  - Période optimale de checkpoint, avec erreurs fatales/silencieuses
  - Utilisation de réplication pour détecter et corriger les erreurs silencieuses, et mélange avec le checkpoint (*restart*)
  - Période optimale de checkpoint pour optimiser la consommation énergétique?
  - Vitesse de ré-exécution différente après une faute?
- Encore plein de défis à adresser, et des techniques à développer pour plein d'applications haute performance, en faisant des compromis entre performance, fiabilité, et consommation énergétique ☺

Conclusion

- Principaux défis des systèmes Exascale:
  - Résilience: besoin de gérer les fautes
  - Energie: réduire la consommation énergétique
- Objectif principal: souvent la performance (temps d'exécution)
- On a des réponses pour plein de modèles:
  - Période optimale de checkpoint, avec erreurs fatales/silencieuses
  - Utilisation de réplication pour détecter et corriger les erreurs silencieuses, et mélange avec le checkpoint (*restart*)
  - Période optimale de checkpoint pour optimiser la consommation énergétique?
  - Vitesse de ré-exécution différente après une faute?
- Encore plein de défis à adresser, et des techniques à développer pour plein d'applications haute performance, en faisant des compromis entre performance, fiabilité, et consommation énergétique ☺

Conclusion

Bon appétit !!!