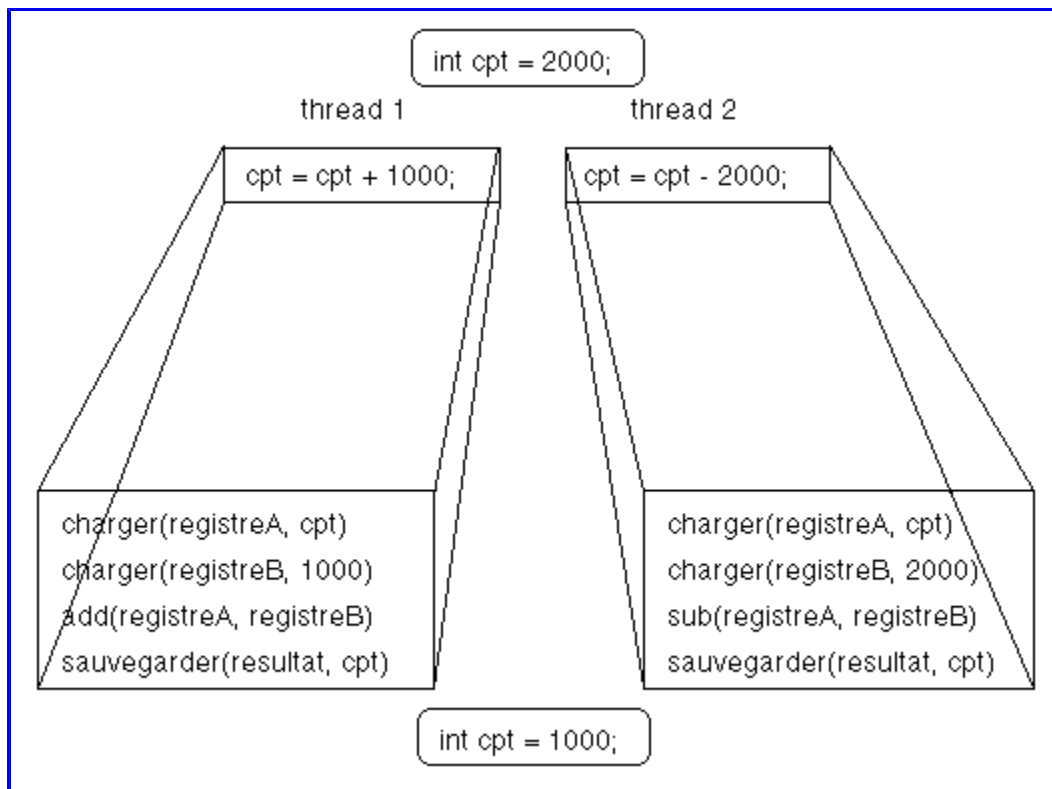


Les problèmes liés à la concurrence

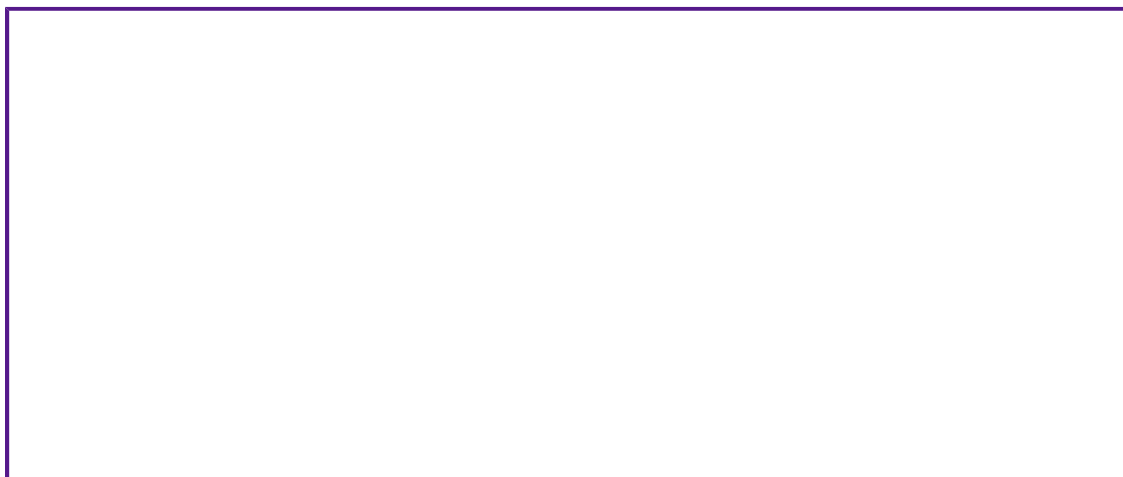
- [Le maintien de la cohérence](#)
- [Section Critique & Exclusion mutuelle](#)
- [Solutions logicielles à l'exclusion mutuelle](#)
- [Le problème majeur des solutions logicielles : l'attente active](#)
- [Solutions matérielles à l'exclusion mutuelle](#)

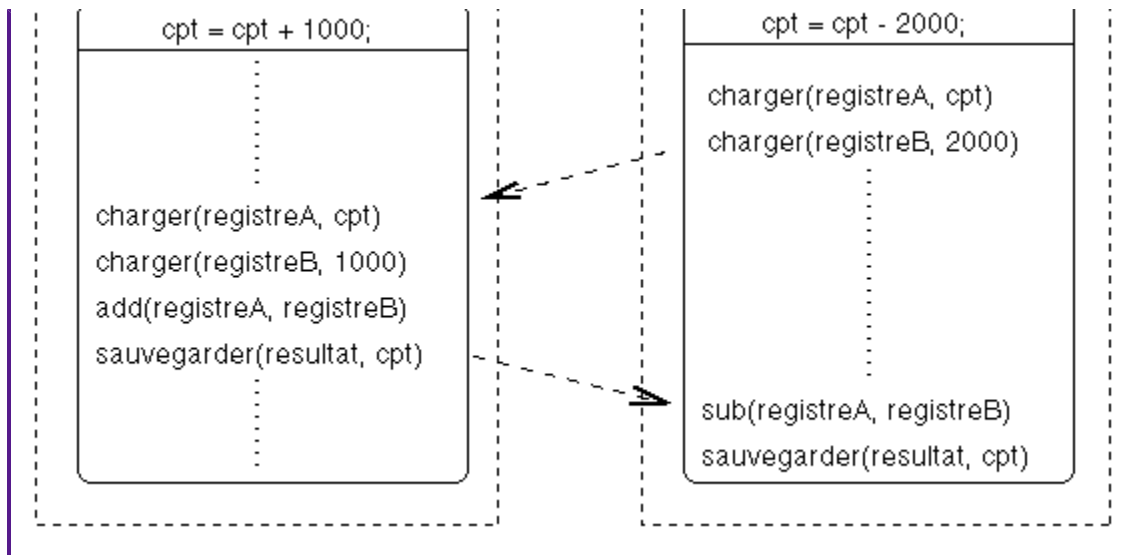
Le maintien de la cohérence

La mise à jour **concurrente** des données peut se dérouler sans problème, comme dans la figure suivante, où la variable `cpt`, initialement à **2000**, a comme valeur finale **1000** ($2000 + 1000 - 2000$).



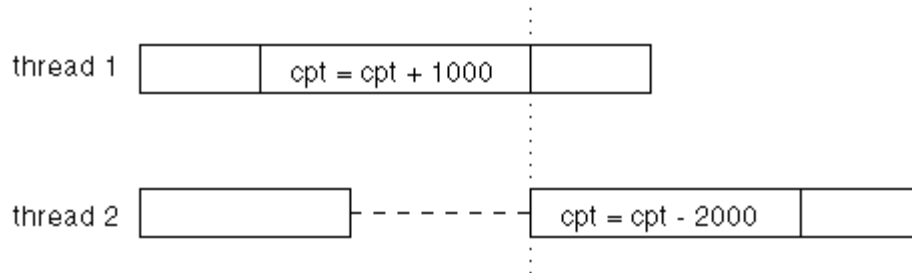
Malheureusement, la mise à jour **concurrente** peut aussi générer des **incohérences**, comme dans la figure suivante, où la variable `cpt`, initialement à **2000**, a comme valeur finale **0** après l'exécution du même code.





Section Critique & Exclusion mutuelle

La mise-à-jour de **cpt** doit s'effectuer en **exclusion mutuelle** (e.g. seule à l'exclusion de toute autre action) car il s'agit d'une **section critique**. Une seule thread à la fois modifie **cpt**. L'autre doit **attendre** que la



thread en cours ait fini.

Solution logicielle à l'exclusion mutuelle

Première solution :

Utiliser un booléen *c* initialisé à **vrai**.

```

tant que (non c) faire rien;
c := faux;
< Début Section Critique (DSC) >
< SC >
< Fin Section Critique (FSC) >
c := vrai;

```

Solution logicielle (fausse) à l'exclusion mutuelle

Solution logicielle (juste) à l'exclusion mutuelle

L'algorithme de **Deekker**. Soit *i* le numéro (0 ou 1) de la thread courante et *j* le numéro de l'autre thread.

initialisations :

```

int c[2];
int tour;
c[0] = FALSE;
c[1] = FALSE;
tour = 0;

```

algorithme :

```
c[i] = TRUE;
tour = j;
while ((c[j]) && (tour == j)) {};
< SC >
c[i] = FALSE;
```

L'inconvénient majeur des solution logicielle : l'attente active

Même si une thread ne fait rien (*i.e.* elle attend pour la section critique), elle **monopolise** le processeur !

L'**attente active** ne devrait pas être permise !

Solutions matérielles (justes) à l'exclusion mutuelle

Le masquage des interruptions.

Une thread qui veut rentrer en section critique **inhibe** les interruptions pour conserver le processeur jusqu'à ce qu'elle les **rétablisse**.

L'instruction Test&Set.

```
procédure Test&Set(var a,b : entier);
début
  a := b;
  b := 1;
fin;

Test&Set(test_i, verrou);
tant que test_i = 1 faire
  Test&Set(test_i, verrou);
< SC >
verrou := 0;
```

Problème = attente active !



[Retour au sommaire.](#)