École Doctorale 2008/2009

Systèmes de Bases de Données Avancés

4. Reprise après panne

École nationale Supérieure d'Informatique

Plan

1. Introduction

2. La journalisation

3. Cas des MMDB

4. Reprise

4.1. Introduction

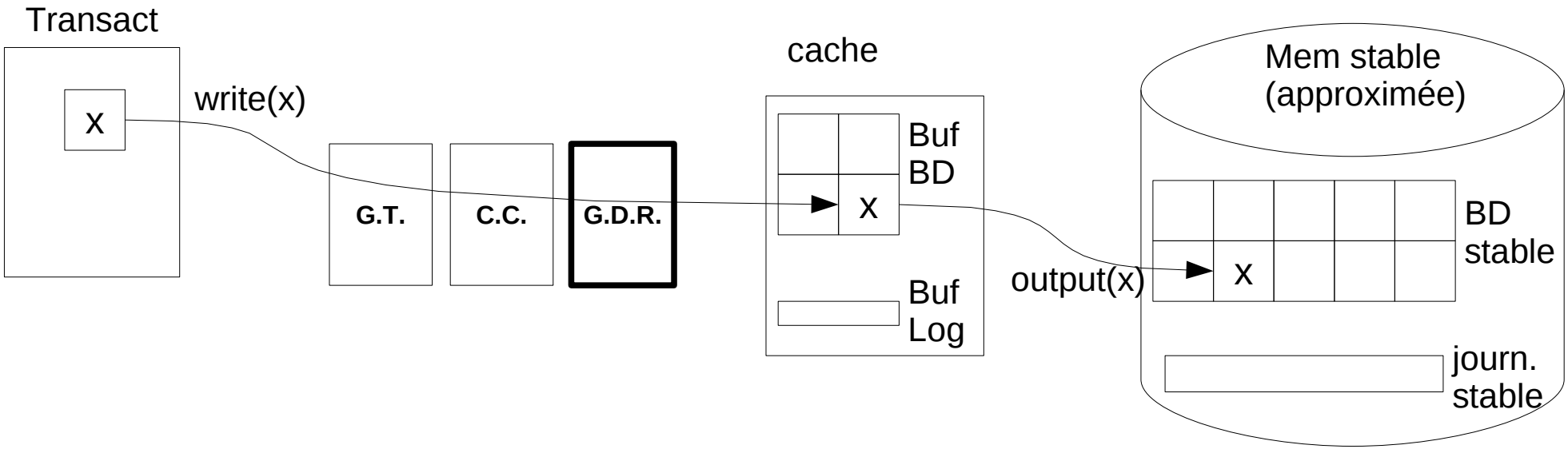
Les pannes

- Erreur de Transaction
  - Erreurs logiques (opération incorrecte, condition non satisfaite, ...)
  - Erreurs système (interblocage, ...)
- Crash système
  - Pb d'alimentation, Reboot, ...
- Erreur de disque
  - En partie ou en totalité

Supports de stockage

- Support volatile
  - Lors des crash système, son contenu est perdu
  - Ex : Mémoire centrale, Mémoire cache
- Support non volatile
  - Résiste aux crash système
  - Ex : Disque, Bande
- Support stable
  - Résiste à tout type de panne
  - Approximé par le maintient d'une certaine redondance de supports non volatiles

Accès aux données



•La BD stable et le journal stable résident sur disque

•Les accès transitent par le cache en MC

•Le Gestionnaire de recouvrement (GR) fait en sorte qu'il y ait toujours assez d'informations sur mémoire stable pour qu'une reprise correcte soit possible après l'occurrence d'une panne.

Modèle du gérant de données

•Le gestionnaire de donnée est composé des modules de recouvrement et de gestion du cache

•Les opérations du gestionnaire de recouvrement

–read, write, commit, abort et restart (en cas de reprise)

•Les opérations du gestionnaire de cache

–Fetch (pour obliger une lecture physique)

–Flush (pour obliger une écriture physique)

–Pin (pour bloquer une page dans le cache)

–Unpin (pour débloquer une page du cache)

Propriétés

•Les techniques de reprises vérifient l'**Atomicité** et la **Durabilité** des transactions

•Le CC envoie les opérations suivant un ordre sérialisable et strict

–Les **images avant** suffisent pour l'implémentation de **l'annulation**

•On ne considère que les pannes provoquant la perte de la mémoire centrale

–Le **disque** est supposé « **stable** »

•Si une panne système se produit en cours d'exécution d'un ensemble de transaction H, alors **C(H)** (projection validée) représente l'historique à appliquer pour retrouver, le dernier **état cohérent de la BD**

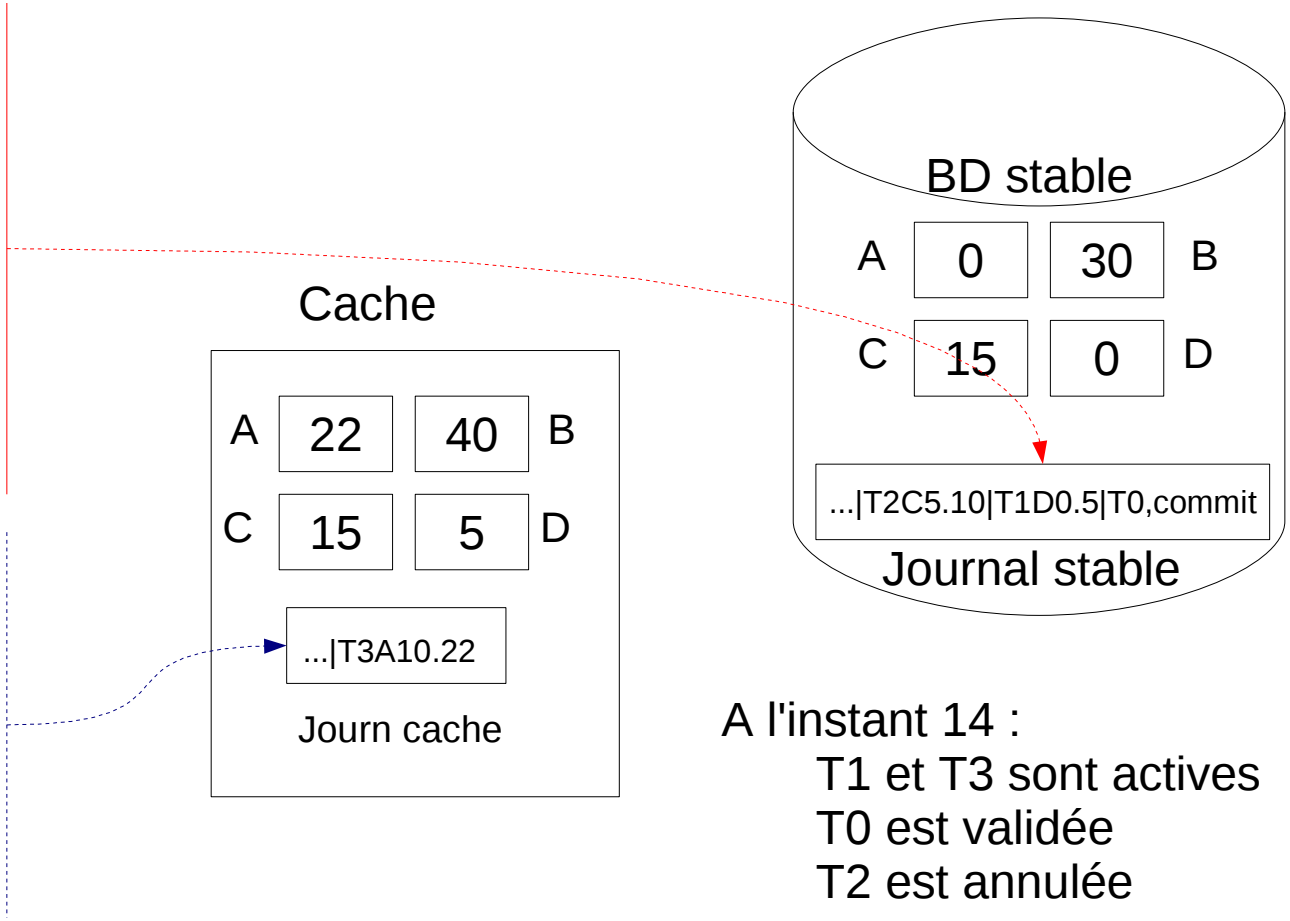
4.2. La journalisation

Définitions

- Un journal est une séquence d'enregistrements décrivant les mises à jours effectuées par les transactions
  - C'est l'historique d'une exécution sur fichier séquentiel
- Un journal est dit « physique » s'il garde la trace des modifications au niveau octet à l'intérieur des pages
  - Ex: <Ti, numPg, Depl, Long, img_avant, img_après>
- Un journal est dit « logique » s'il garde la trace de la description de haut niveau des opérations de mise à jour
  - Ex: « insérer le tuple x dans la table T et mettre à jour les index »

Exemple

```
 1 <T0, start>
 2 <T0, A, 0, 10>
 3 <T1, start>
 4 <T0, B, 20,30>
 5 <T2, start>
 6 <T2, C,5 ,10>
 7 <T1, D, 0, 5>
 8 <T0, commit>
 9 <T2, A, 10, 15>
10 <T1, B, 30, 40>
11 <T2, abort>
12 <T1, C, 5, 15>
13 <T3, start>
14 <T3, A, 10, 22>
15 ...
```



A l'instant 14 :

T1 et T3 sont actives
T0 est validée
T2 est annulée

Règles du undo et du redo

•Les deux règles suivantes doivent toujours être vérifiées par le module du recouvrement afin d'assurer l'atomicité et la durabilité des transactions: « WAL » et « Force log at commit »

•**Règle du Undo** (ou « Write-Ahead Log Protocol »)
Avant d'écraser une ancienne valeur par une nouvelle, dans la BD stable, sauvegarder l'ancienne (l'image avant) dans un autre emplacement stable (dans le journal stable)

•**Règle du Redo** (ou « Force Log at commit »)
Avant d'autoriser une transaction à validée, toutes les valeurs générées (les images-après de son journal) doivent d'abord être sauvegardées en mémoire stable (journal stable)

Utilisations du journal

- **Lors de l'annulation**

=> remettre les images avant (Undo) en le parcourant en arrière

- **Lors de la validation**

=> sauvegarder toutes les images après en mémoire stable

=> optimisation « **group commit + precommit** »

- **Lors d'une reprise après panne**

=> Refaire l'historique en exécutant les opérations du journal du début jusqu'au moment de la panne (dernier enregistrement)

=> très coûteux et beaucoup de travail inutile

Checkpoints

C'est la sauvegarde (périodique) de certaines informations en mémoire stable durant le déroulement normal des transactions, afin de réduire la quantité de travail que doit faire la procédure 'Restart' après une panne.

•**Checkpoint consistant**
- Stopper l'acceptation de nouvelles transactions
- Attendre que les transactions actives se terminent
- Sauvegarder toutes les pages modifiées du cache
- Marquer le journal stable par un enreg <ckpt>

•**Lors de la reprise (Restart)**
- Parcourir le journal en arrière jusqu'au <ckpt> le plus récent et défaire les opérations des transactions annulées
- Parcourir le journal en avant depuis la marque du <ckpt> et refaire les opérations des transactions validées

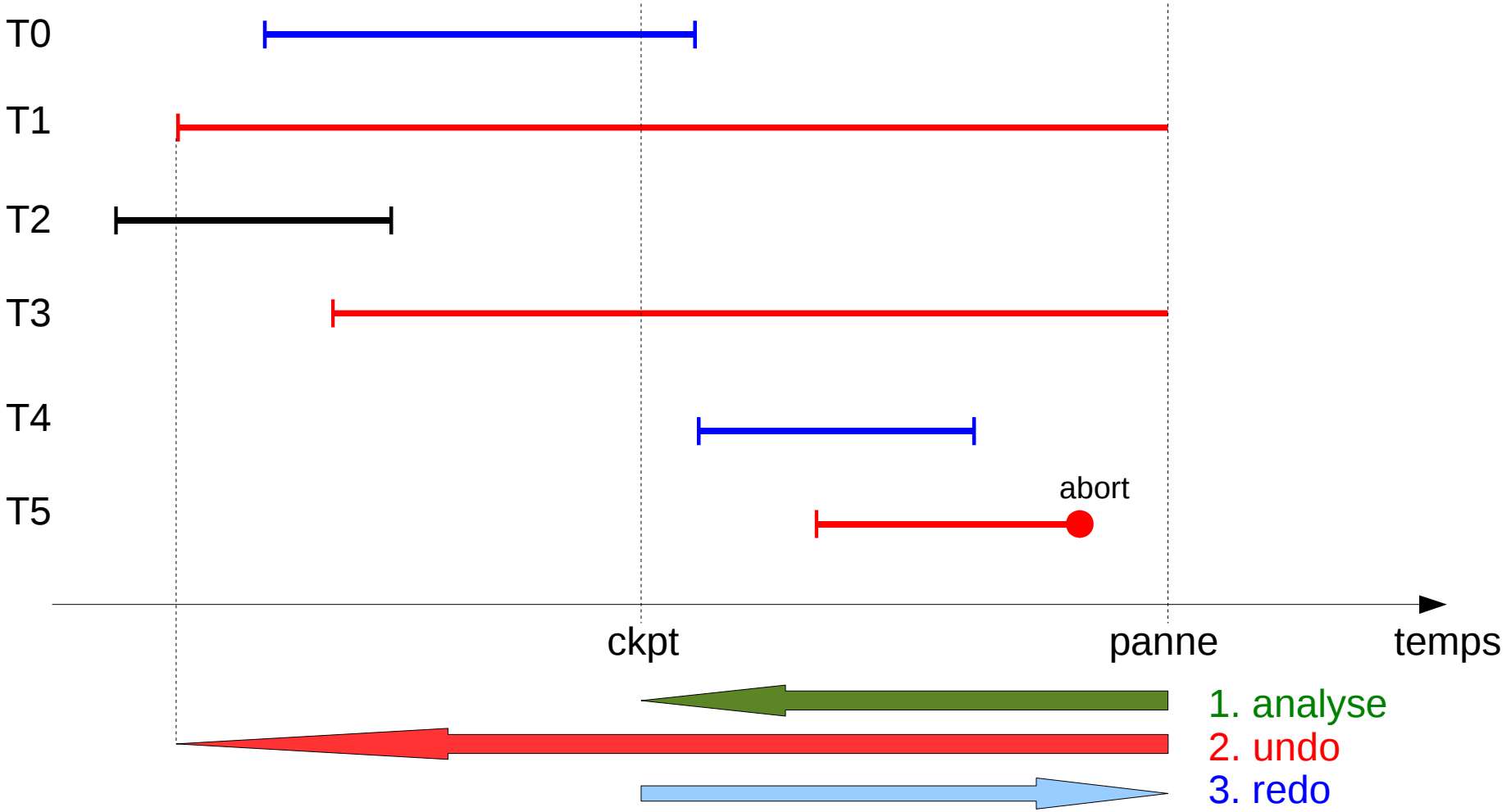
Checkpoint inconsistant

•**But:** réduire le temps de blocage du système transactionnel durant l'opération du checkpoint

•**Checkpoint inconsistant:**
- Stopper l'acceptation de nouvelles opérations (en laissant donc les transactions actives dans un état bloqué)
- Sauvegarder toutes les pages modifiées du cache (en appliquant WAL)
- Marquer le journal stable par un enreg <ckpt, L> où L est la liste des transactions actives durant le checkpoint

•**Reprise en 3 phases:**
- analyse : détermine les transactions validées des autres
- undo : défaire les transactions annulées et actives
- redo : refaire les transactions validées

Exemple



reprise: T0 et T4 seront refaites. T1, T3 et T5 seront défaites

Checkpoint inconsistant optimisé

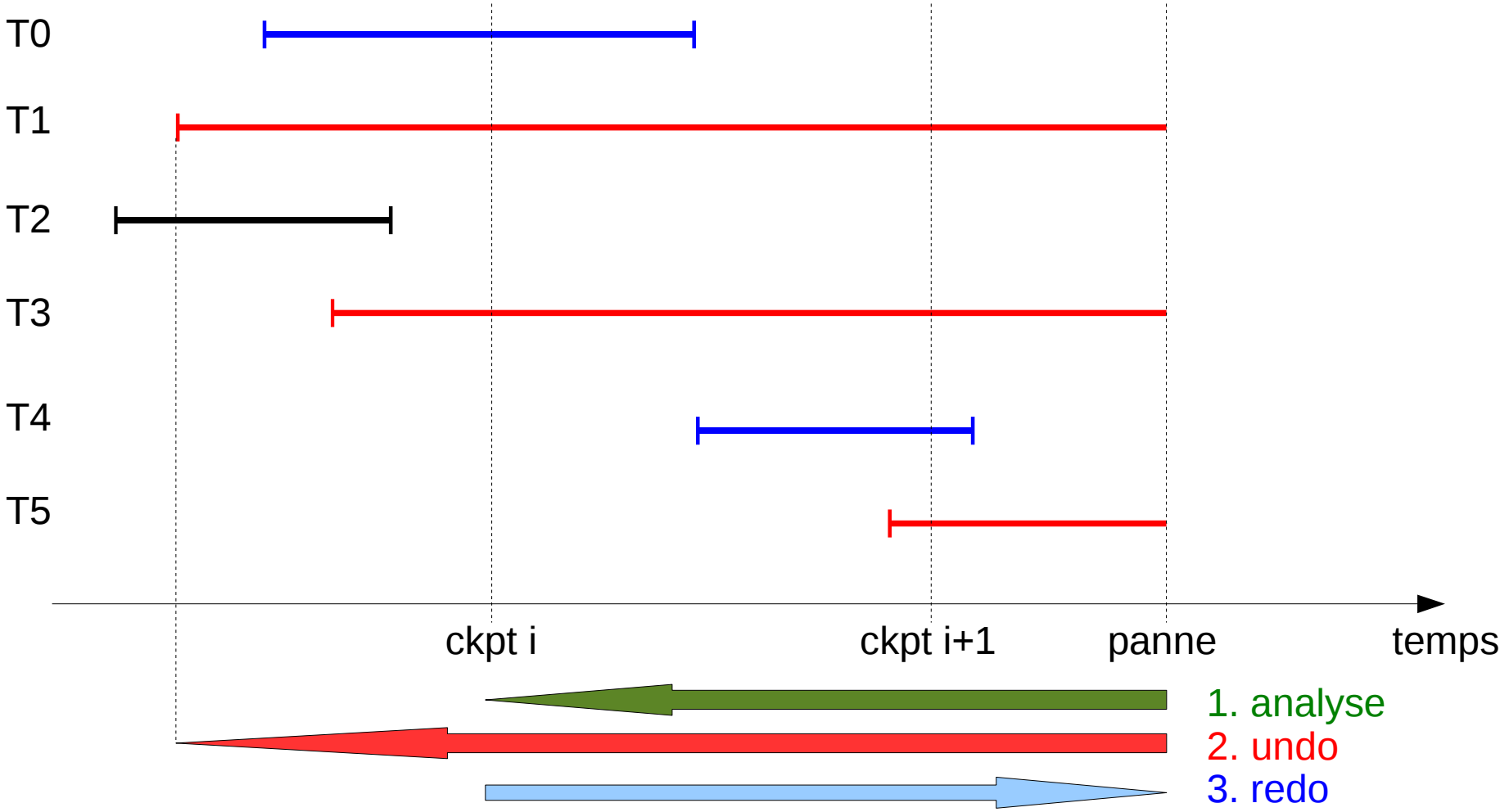
•**But:** réduire encore plus le temps de blocage du système transactionnel durant l'opération du checkpoint

•**Checkpoint inconsistant optimisé:**
- Stopper l'acceptation de nouvelles opérations (en laissant donc les transactions actives dans un état bloqué)
- Sauvegarder toutes les pages modifiées avant le dernier checkpoint (en appliquant WAL)
- Marquer le journal stable par un enreg <ckpt, L> où L est la liste des transactions actives durant le checkpoint

Remarque: le nombre de page à sauvegarder devrait alors être petit

•**Reprise en 3 phases:** (en allant jusqu'à l'avant dernier checkpoint)
- analyse : détermine les transactions validées des autres
- undo : défaire les transactions annulées et actives
- redo : refaire les transactions validées

Exemple



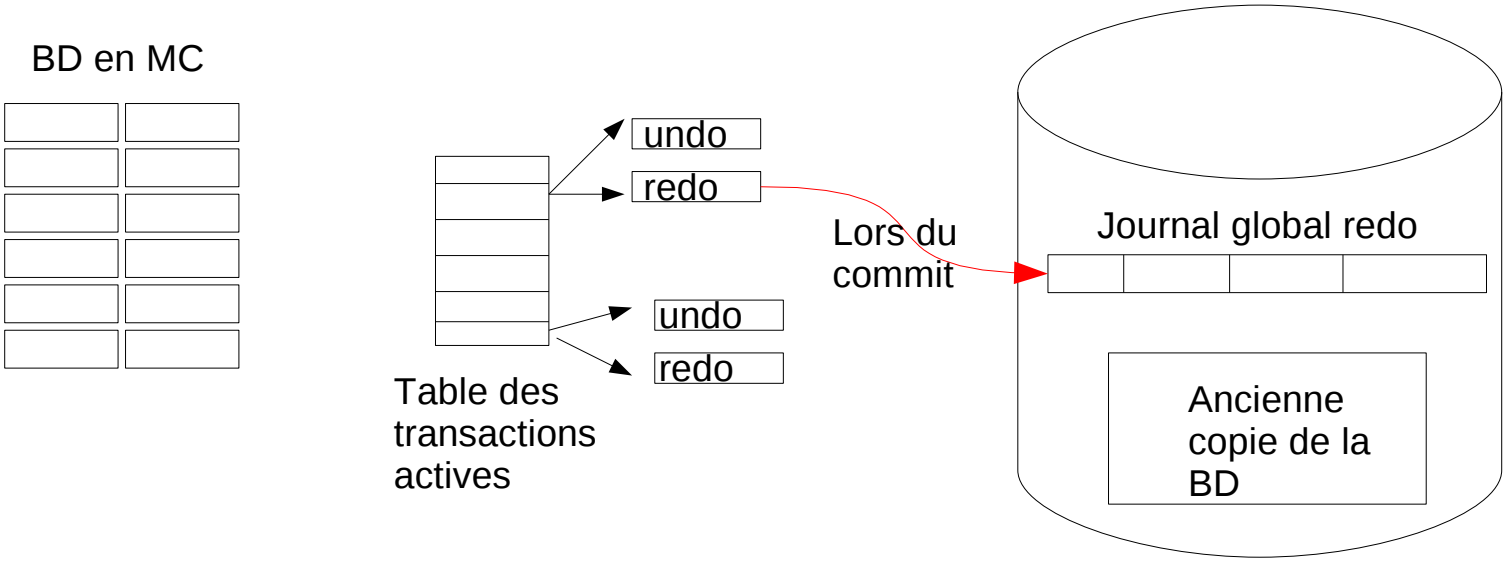
reprise: T0 et T4 seront refaites. T1, T3 et T5 seront défaites

4.3. Cas des MMDB

« Main Memory DataBases »

•Lorsque la taille de mémoire centrale peut emmagasiner toute la BD, l'utilisation des disques ne se résume qu'à la journalisation pour assurer la propriété de durabilité des transactions.

•Lors de la validation d'une transaction T, ses enreg redo sont écrits sur le journal stable. Son journal undo est alors supprimé.



« Fuzzy Checkpoints »

Périodiquement, sauvegarder les pages modifiées sur disque sans bloquer les transactions active => copie inconsistante de la BD

Rajouter à la fin les journaux undo des transactions actives au moment de la sauvegarde => copie inconsistante + les undo = copie consistante

Checkpoint de type « ping-pong »

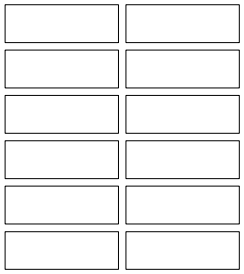
- On maintient 2 copies (A et B) de la base sur disque, utilisées en alternance
- S'il y a une panne durant l'opération de checkpoint, la 2e copie sera utilisée

- Pour la reprise, il suffit de :
  charger la dernière copie « complète »
  appliquer les undo (pour « nettoyer » les inconsistances)
  appliquer les redo (pour retrouver tous les effets des transactions validées avant la panne)

Illustration

En MC

BD en MC



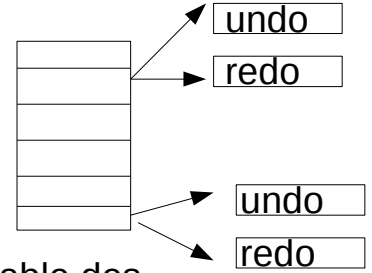
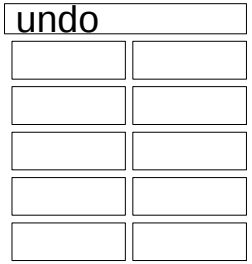


Table des transactions actives

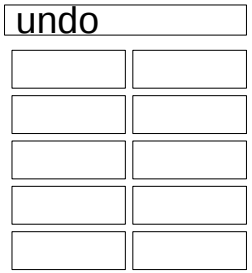
Sur disque

Copie A



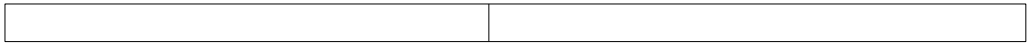
Master record

Copie B



Fin du journal

Point de reprise



Redo stable (ceux des transactions validées uniquement)