# Fiabilité d'un SGBD

Comment gérer la reprise après panne d'un SGBD pour garantir la cohérence des BDs ?

# Panne et durabilité

- Pour chaque transaction un SGBD doit assurer
  - Que les effets d'une transaction validée persiste dans la base,
  - Qu'une transaction non validée n'ait d'effet ni sur la base ni sur les autres transactions.
- Un SGBD doit inclure des mécanismes pour éviter les erreurs et restaurer un état correct après une panne.
- Nombreuses causes d'erreurs: programmes, système, humaines, hardware, externe…

# Définition de la fiabilité

- Capacité à restaurer la base de données dans un état connu comme correct après une erreur.
- Solution: la redondance d'informations
  - Backup périodique de la base,
  - Écriture dans un Journal des opérations sur la base
  - En cas de panne + base endommagée: restauration + utilisation du journal pour refaire les modifications depuis le backup
  - En cas de panne + base Ok: le contenu n'est plus fiable, on la restaure à l'aide du journal en défaisant les modifications.

# Différentes catégories de panne

- Panne locale à une transaction
  - Détectée par le code de l'application: erreurs classiques suite à des conditions non vérifiées → Rollback
  - **Imprévue → Annulation par le système**
- **Panne du système impactant les transactions sans endommager la base → mise en œuvre de protocoles de reprise**
- Arrêt volontaire du système → attente de terminaison des transactions, puis arrêt.
- Panne endommageant la base
  - Perte de donnée → restauration + journal pour refaire les modifications.
  - Perte du journal → restauration ancien backup

# Panne locale imprévue

- Le système doit défaire toutes les modifications de la transaction comme si elle n'avait jamais eu lieu.
- Utilisation du journal:
  - Écriture du début de transaction (BEGIN)
  - Écrire chaque modification, suppression, insertion avant la MAJ de la base (pourquoi ?)
  - Écriture de la fin de transaction (COMMIT/ROLLBACK)
- Procédure d'annulation: remonter le journal en arrière en replaçant les anciennes valeurs jusqu'au marqueur BEGIN !
- Défaire une transaction X fois = défaire 1 fois, sinon effet indésirable lors d'une panne pendant l'annulation.

# Panne système → point de reprise

- Panne système = redémarrage
  - Perte de la mémoire (transactions en cours, buffer d'E/S)
  - La base n'est pas endommagée
- Comment connaître
  - les transactions validées
  - Les transactions qui auraient pu valider (mais écritures des buffers incertains ?)
  - Les transactions qui n'ont pas terminé avant la panne.
- On introduit **un point de reprise** pour ordonner les différentes transactions par rapport à un état cohérent.

# Définition du point de reprise

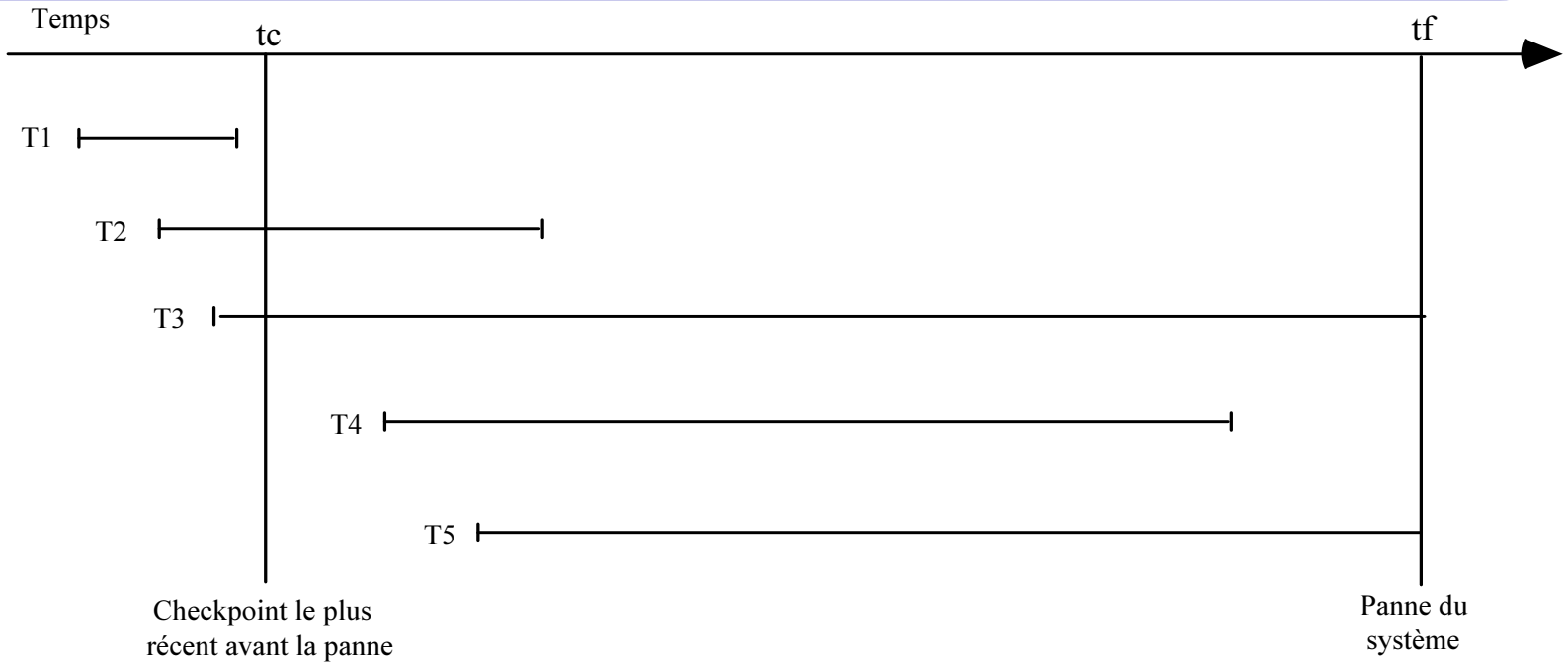
- Au point de reprise:
  - Les transactions sont suspendues
  - Écriture des buffers du journal
  - Écriture d'un point de reprise dans le journal (CHECKPOINT)
  - Écriture des buffers de données → Force-writing (les données en mémoire sont forcées sur le support persistant)
  - Écriture du point de reprise dans le fichier de redémarrage.

- Intervalle entre point de reprise
  - Soit périodique (tous les m minutes),
  - Soit en fonction du nombre de transactions validées (depuis le dernier CHECKPOINT)

# Redémarrage avec point de reprise

- Le fichier de redémarrage indique le point de reprise le plus récent,
- Le journal est lu à partir du checkpoint
  - Les transactions à défaire → en remontant le journal jusqu'au checkpoint
  - Les transactions à refaire → en parcourant le journal à partir du checkpoint
- 5 cas de transactions sont considérés par le gestionnaire de reprise.

# Différents cas de transactions




- T1 commencées et finies avant le checkpoint;
- T2 commencées avant le checkpoint et finies après (mais avant la panne);
- T3 commencées avant le checkpoint et toujours en cours lors de la panne;
- T4 commencées et finies après le checkpoint;
- T5 commencées après le checkpoint et toujours en cours lors de la panne.

# Méthode générale de reprise

- Transaction non considérée: T1
  - Les modifications de T1 ont déjà été forcées sur la base.
- Transaction à défaire: T3 et T5
  - Les transactions ne sont pas terminées et la fin n'est pas connue
  - Elles ne doivent jamais avoir existé
- Transaction à refaire: T2 et T4
  - Elles ont validé après le checkpoint
  - Aucune assurance que les buffers ont été écrits sur la base
  - Pour T2, seuls les changements après le checkpoint sont à refaire.
  - Refaire une transaction X fois = refaire 1 fois !
- Mode d'écriture du journal *Write-ahead*:l'écriture sur la base se fait après l'écriture sur le journal !

# Protocole Immediate Update

- C'est le protocole d'écriture considéré précédemment: le buffer d'une transaction est écrit dés qu'il est plein… sans attendre COMMIT.
- La méthode Write-ahead protège des MAJs impossible à défaire
- Problème : risque de défaire en cascade
  - On doit défaire T,
  - S lit une valeur X écrite par T → on doit aussi défaire S
  - R lit une valeur Y écrite par S → on doit aussi défaire R

# Gestion du journal en mise à jour immédiate

1. Commencer à partir du dernier élément dans le journal et lire en arrière. Faire deux listes de transactions
   - LR celles qui ont fait COMMIT après le dernier checkpoint
   - LD1 celles qui n'ont pas fait COMMIT.
2. Faire une liste LD2 des transactions qui ont lu des valeurs écrites par les transactions de LD1.
   - Appliquer cette étape récursivement à toute transaction de LR
   - pour faire une liste LD2 des transactions qui doivent être défaites.
3. Défaire toutes les opérations WRITE des transactions de LD1 et LD2 dans l'ordre inverse du journal.
4. Refaire les opérations WRITE des transactions de LR dans l'ordre d'écriture dans le journal.

# Application Immediate Update

| T0: A=1000, B=2000 | T1: C=700 |
|---|---|
| Read(A) | Read(C) |
| A=A-50 | C=C-100 |
| Write(A) | Write(C) |
| Read(B) | |
| B=B+50 | |
| Write(B) | |

Exécution sérielle cohérent (T0, T1) sans panne

| | |
|---|---|
| <T0, begin> | |
| <T0, A, 1000, 950> | |
| <checkpoint> | A=950 |
| <T0, B, 2000, 2050> | |
| <T0, commit> | B=2050 |
| <T1, begin> | |
| <T1, C, 700, 600> | |
| <T1, commit> | C=600 |

- Panne 1 après write(B)
  - A=1000, B=2000, C=700
- Panne 2 après write(C)
  - A=950, B=2050, C=700
- Panne 3 après <T1,commit>
  - A=950, B=2050, C=600

# Protocole Deffered Update

- Les mises à jour sur la base de données sont effectuées physiquement seulement quand la transaction arrive au COMMIT.
- Les mises à jour sont mémorisées dans
  - le journal
  - l'espace de travail de la transaction.
- La transaction arrive au COMMIT
  - Le journal est forcé sur disque,
  - Les mises à jour sont mémorisées dans la base
- Avantage: si une transaction a une panne avant d'arriver au COMMIT → pas besoin de défaire les opérations, le journal est plus simple !

# Gestion du journal en mise à jour différée

1. Remonter le journal jusqu‘au checkpoint le plus récent.  Faire deux listes de transactions:
   - LR celles qui ont fait COMMIT après le dernier checkpoint
   - LD celles qui n'ont pas fait COMMIT.
2. Refaire toutes les opérations WRITE des transactions LR dans l'ordre d'écriture dans le journal.
3. Les transactions de LD sont ignorées (rien à défaire).

- Inconvénient: cette méthode limite l'exécution concurrente car tous les éléments sont verrouillés jusqu'au COMMIT.

# Application Deferred Update 1/2

| T0: A=1000, B=2000 | T1: C=700 |
|---|---|
| Read(A) | Read(C) |
| A=A-50 | C=C-100 |
| Write(A) | Write(C) |
| Read(B) | |
| B=B+50 | |
| Write(B) | |

Exécution sérielle cohérent (T0, T1) sans panne

| | |
|---|---|
| <T0, begin> | |
| <T0, A, 950> | |
| checkpoint | |
| <T0, B, 2050> | |
| <T0, commit> | A=950, B=2050 |
| <T1, begin> | |
| <T1, C, 600> | |
| <T1, commit> | C=600 |

- Panne 1 après write(B)
  - A=1000, B=2000, C=700
- Panne 2 après write(C)
  - A=950, B=2050, C=700
- Panne 3 après <T1,commit>
  - A=950, B=2050, C=600
- Que se passe-t'il si un crache intervient pendant la reprise de la panne 2 ?
  - Rien, on peut refaire plusieurs fois = refaire 1 fois.