

Gestion de transactions centralisée



Patricia Serrano Alvarado

Laboratoire d'Informatique de Nantes Atlantique (LS2N)

Patricia.Serrano-Alvarado@univ-nantes.fr

Transaction

- Un utilisateur manipule une base en écrivant des **programmes d'application** qui font des appels au SGBD. L'exécution d'un programme fait naître au niveau du SGBD l'occurrence d'une **transaction**
- Une transaction (T) est un **groupe d'actions** (lectures, écritures) sur une **base de données**
- Les **notions de cohérence** et tout ce qui concerne le support transactionnel sont **indépendantes du modèle de données** utilisé par la base de données

Cycle de vie d'une transaction

- Une transaction peut :
 - se dérouler normalement,
 - être tuée en cours d'exécution ou
 - s'arrêter par elle même avant la fin.

- Les opérations d'une transaction
 - début
 - fin (validation, annuler)
 - lire : lecture de la valeur d'un objet à partir de la BD et stockage de la valeur dans l'espace de travail de la transaction
 - écrire : à partir d'une valeur stockée dans l'espace de travail de la transaction et écrire cette valeur dans la base pour l'objet désigné
 - abandonner (rollback, abort) : défaire toute les m à j faites par la transaction depuis son début

Le travail du gestionnaire des transactions

- Initialiser chaque transaction et contrôler son exécution.
Si celle-ci se passe bien => confirmer la transaction
Sinon annuler la transaction en défaisant ses opérations.
- Contrôler les *accès concurrents en synchronisant* les transactions en conflit.
- Assurer la *reprise après panne* : refaire le travail des transactions ayant atteint leur point de confirmation (commit/validation) avant la panne et défaire celles qui n'avaient pas atteint ce point au moment de la panne.

Deux problématiques

1. Gestion des accès concurrents

- TRANSACTION T_i = séquence d'actions (LIRE, ECRIRE)
- Exécution concurrente de N TRANSACTIONS:
ORDONNANCEMENT (S) dans le temps des actions (A) de ces transactions

$S = \langle \dots\dots\dots (T1, A1, x) \dots\dots\dots (T2, A2, x) \dots\dots\dots \rangle$
-----|-----|-----> temps

2. Reprise après pannes pour remettre la base dans un état cohérent (fiabilité)

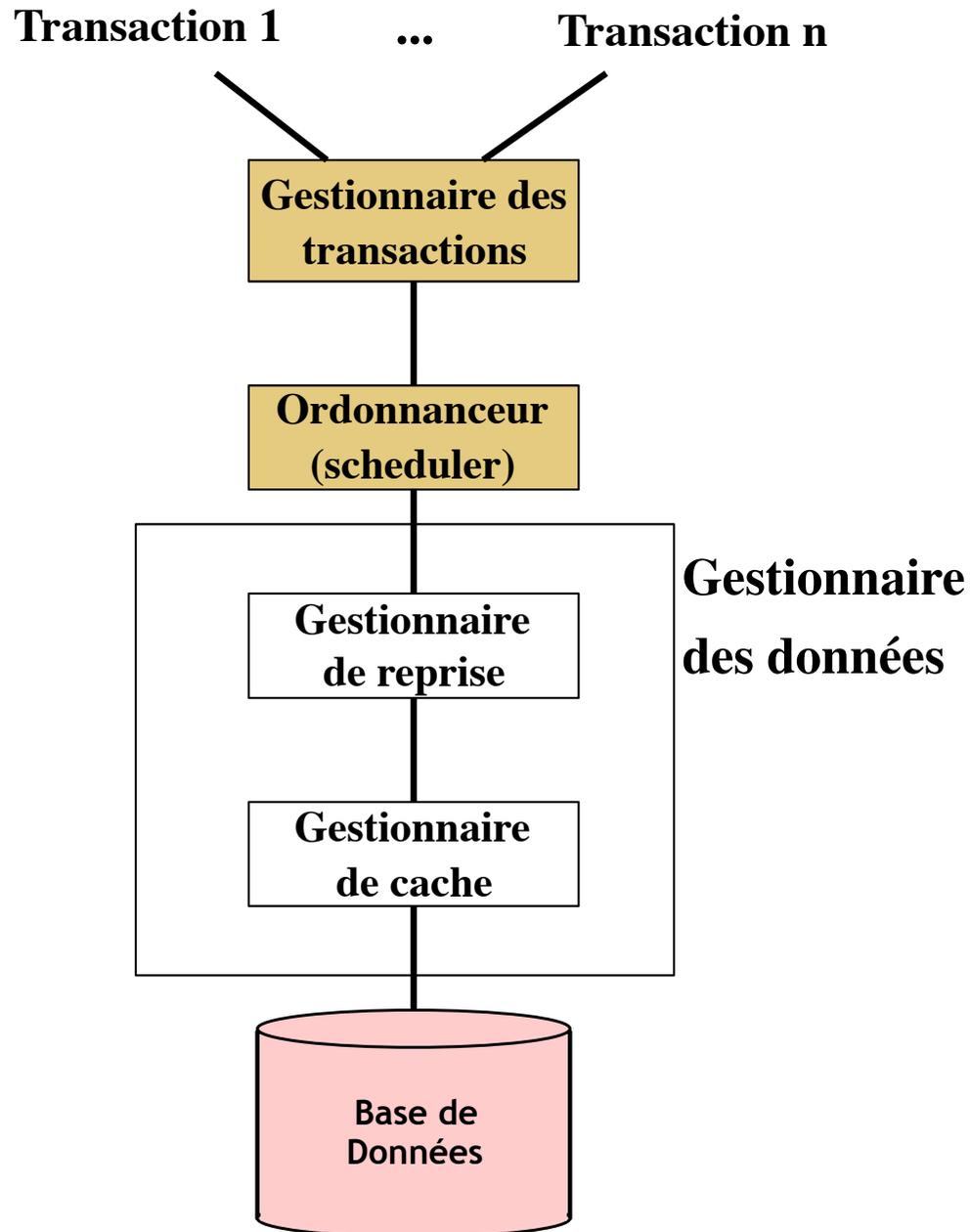
Reprise après pannes



RECOVERY

SGBD centralisé

Architecture abstraite



Les menaces

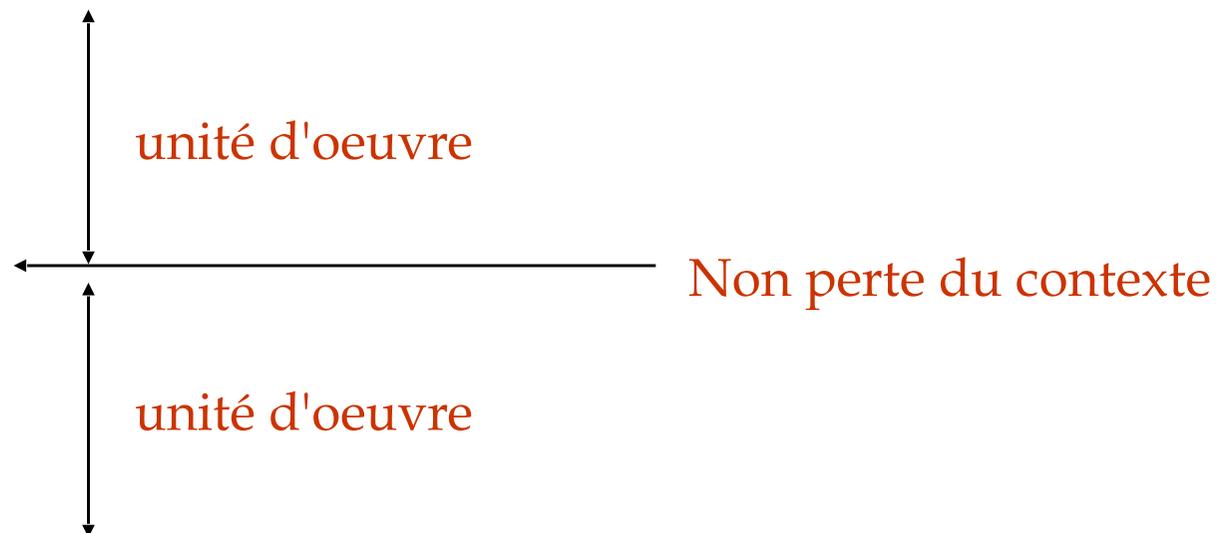
- Problèmes de concurrence
 - pertes d'opérations
 - introduction d'incohérences
 - verrous mortels (deadlock)
- Panne de transaction/opération
 - erreur en cours d'exécution du programme applicatif
 - nécessité de défaire les mises à jour effectuées
- Panne système
 - perte de la mémoire centrale
 - toutes les transactions en cours doivent être défaites
- Panne disque
 - perte de données de la base

Panne d'une transaction

- Points de reprise au sein d'une transaction
 - *Savepoint, commitpoint*
- Permet au programmeur d'avoir un contrôle sur comment la transaction va être récupérée (*rolled-back*) en cas d'erreur ou panne

- Begin_Trans
 - update
 - update
 - **Savepoint A**
 - update
 - update

- Commit
- Roll-back A



Panne système

- Utilisation de la journalisation (*logging*) et de **points de reprise** (*checkpoints*)

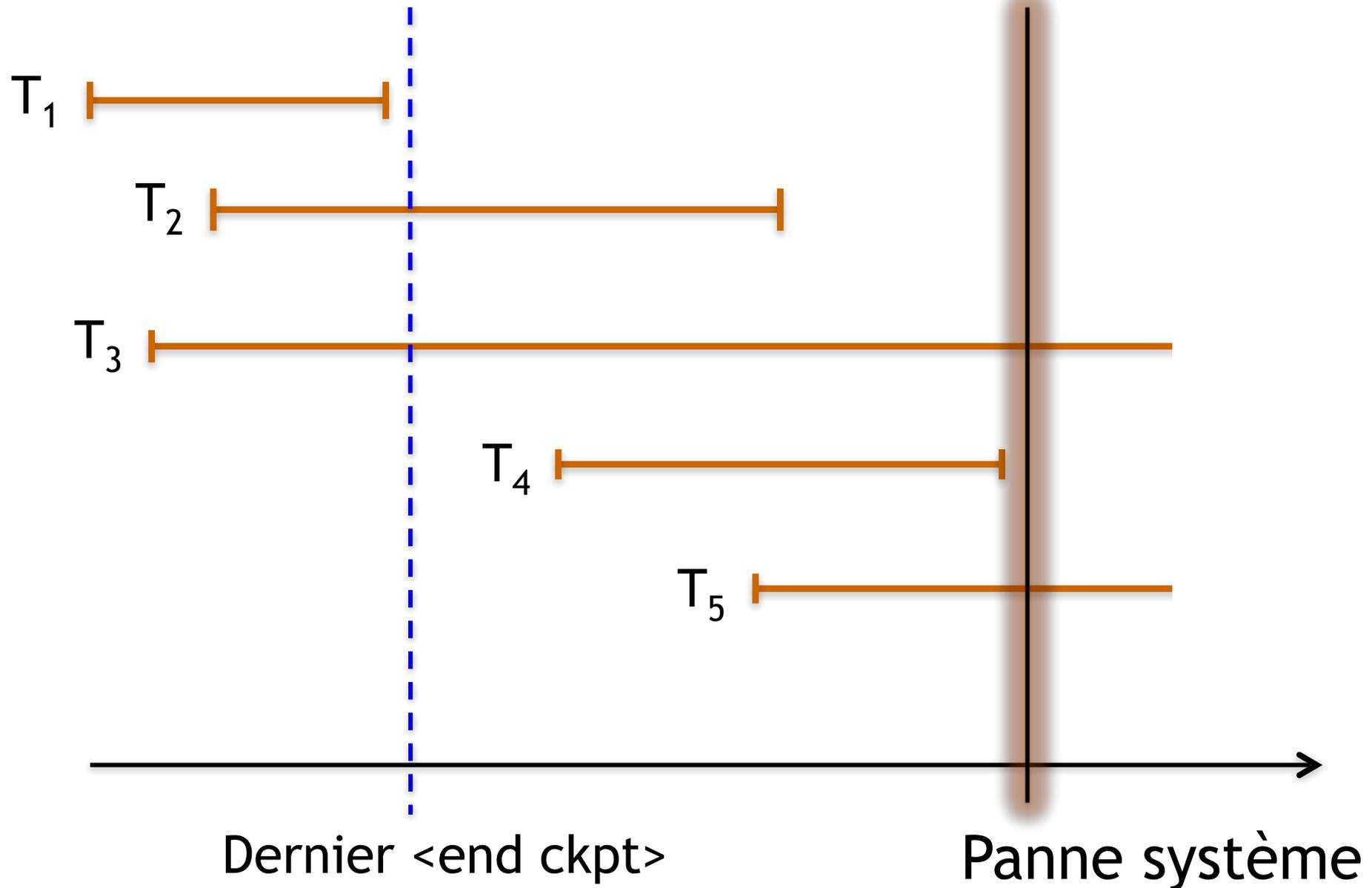
- Un journal est un fichier d'enregistrements qui dit ce qui a été fait dans une transaction

- Un **checkpoint** (*quiescent*) est un enregistrement particulier sur le *log* demandant :
 1. Arrêter d'accepter de transactions
 2. Attendre la fin des transactions actives
 3. Flush du log
 4. Écrire <ckpt> sur le log
 5. Flush du nouveau
 6. Accepter à nouveau des transactions

Nonquiescient checkpoint

- Dans la plupart de systèmes impossible d'arrêter d'accepter de nouvelles transactions
- Pour un nonquiescent checkpoint:
 1. Écrire un enregistrement `<start ckpt (T1,...,TK)>` avec toutes les transactions actives
 2. Flush du log
 3. Attendre la fin des transactions actives
 4. Ne pas interdire de nouvelles transactions
 5. Lorsque les transactions `T1,...,TK` ont fini, écrire `<end ckpt>`
 6. Flush du log

Exécution de transactions concurrentes



Récupération après pannes

- Transaction T1 n'a pas besoin de récupération
- Toute transaction **en exécution** au moment de la panne doit être **défaite et relancée**
 - C'est le cas de T3 et T5
- Toute transaction **validée** après le dernier point de reprise doit être **refaite**
 - C'est le cas de T2 et T4

Comment choisir les transactions à faire ou refaire ?

□ Algorithme simple

UNDO = all transactions running at the last checkpoint

REDO = empty

For each entry in the log, starting at the last checkpoint

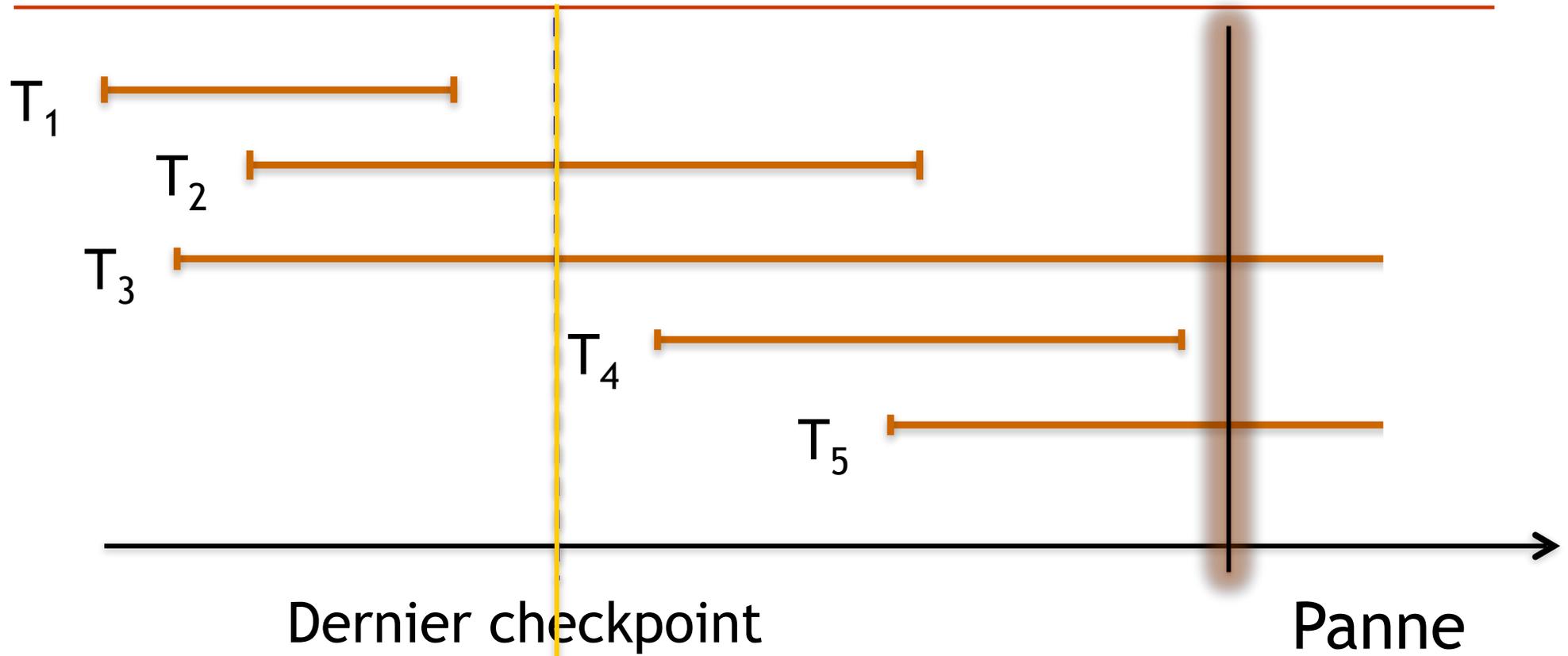
 If a BEGIN TRANSACTION entry is found for T

 Add T to UNDO

 If a COMMIT entry is found for T

 Move T from UNDO to REDO

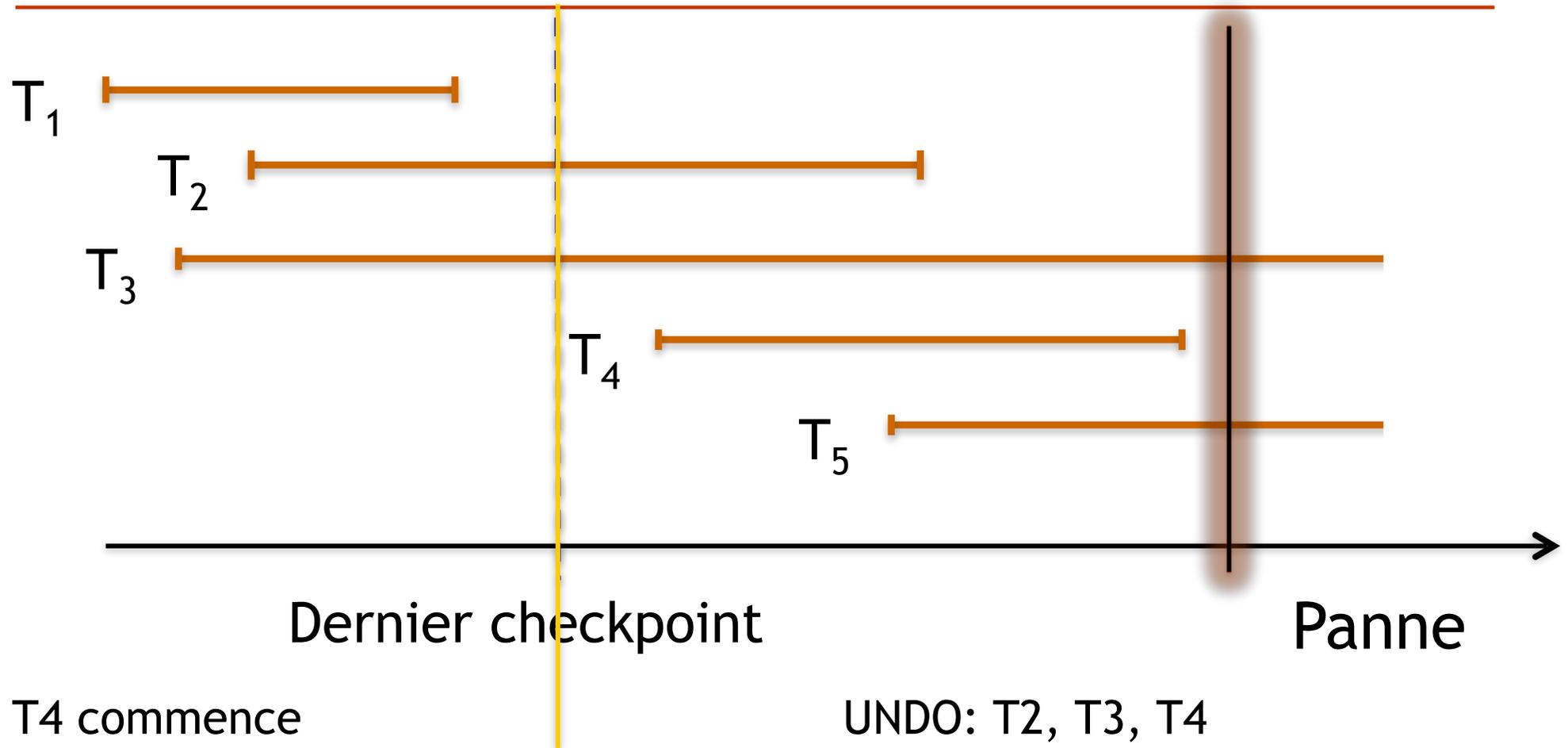
Cont.



Au dernier checkpoint
transactions actives : T2 et T3

UNDO: T2, T3
REDO:

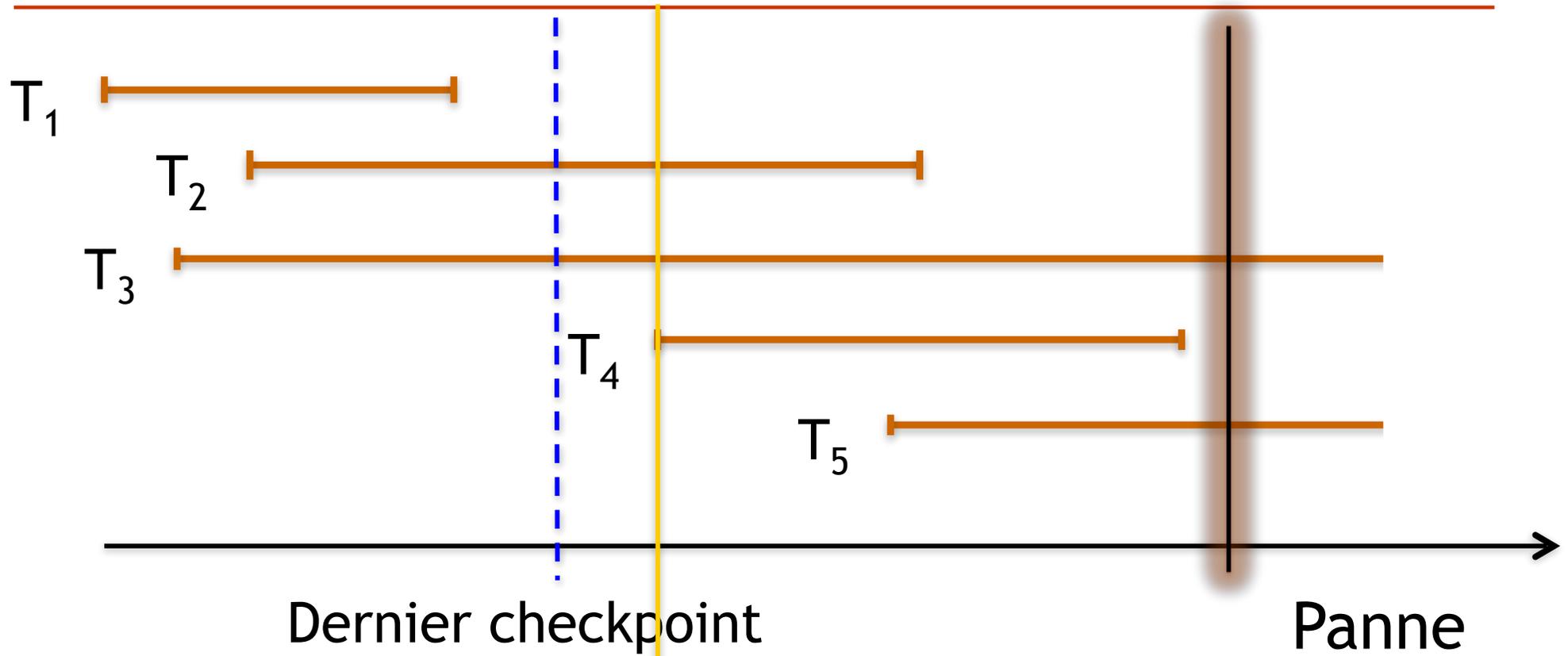
Cont.



T4 commence
Ajouter T4 to UNDO

UNDO: T2, T3, T4
REDO:

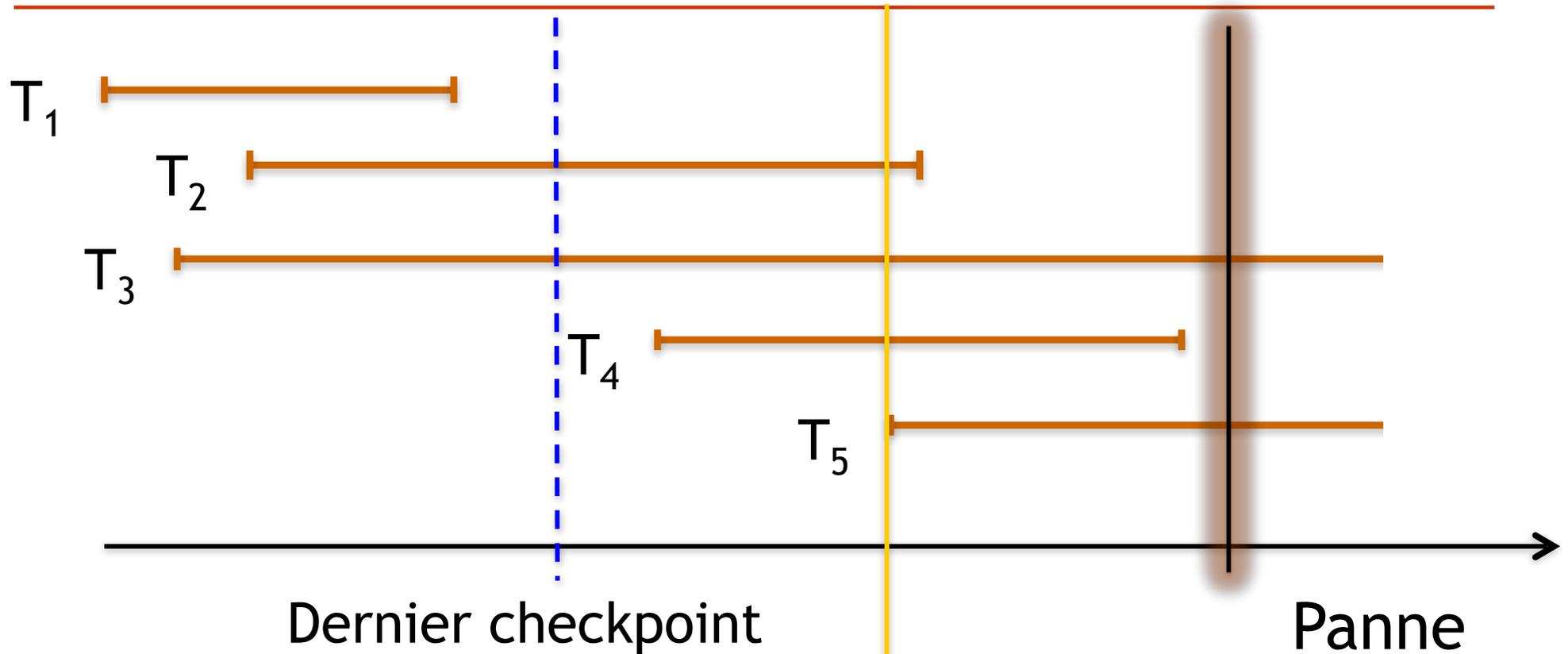
Cont.



T5 commence
Ajouter T5 à UNDO

UNDO: T2, T3, T4, T5
REDO:

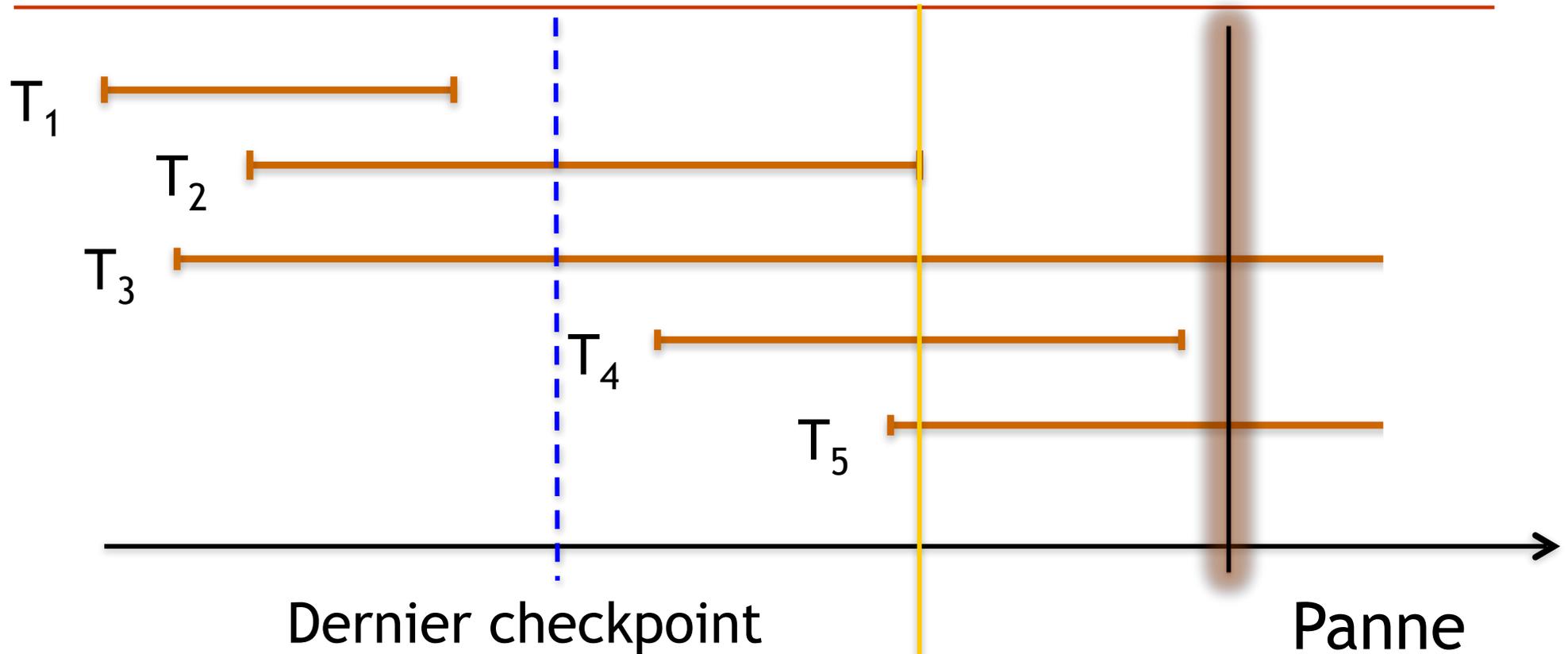
Cont.



T2 valide
Déplacer T2 à REDO

UNDO : T3, T4, T5
REDO : T2

Cont.



T4 valide
Déplacer T4 à REDO

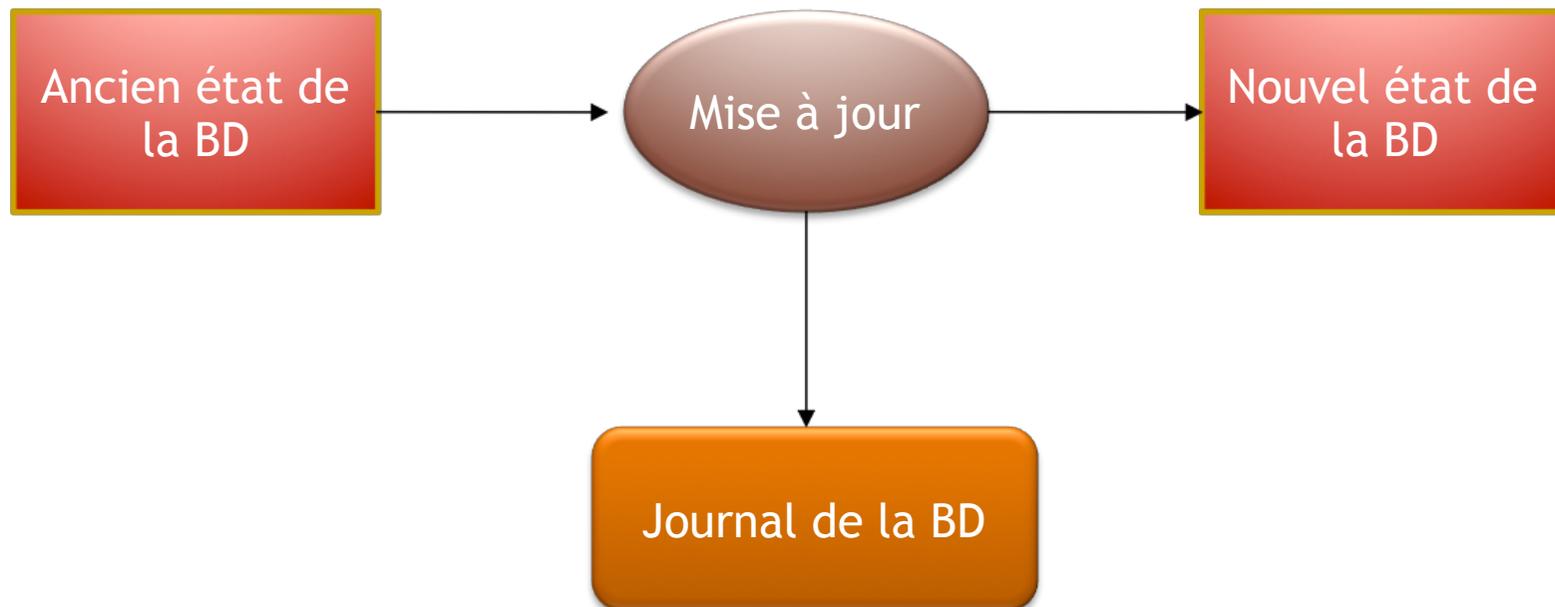
UNDO : T3, T5
REDO : T2, T4

Comment faire et refaire ?

- Utilisation de journaux (*logs*)
- Récupération en arrière (*backward recovery*)
 - Parcourir les journaux en arrière
 - Permet de défaire les transactions
 - Rends la BD cohérente
- Récupération en avant (*forward recovery*)
 - Parcourir les journaux en avant, à partir du dernier *checkpoint*,
 - Permet de refaire les transactions
 - Rends la BD à jour

Journalisation (*logging*)

- ❑ Stocker sur un support de mémorisation fiable, différent de celui de la base de données, les informations nécessaires pour défaire (*undo*) et refaire (*redo*) les transactions
- ❑ Règle *write ahead log* -> les entrées sur le journal doivent être faites avant que les opérations soient écrites sur la BD

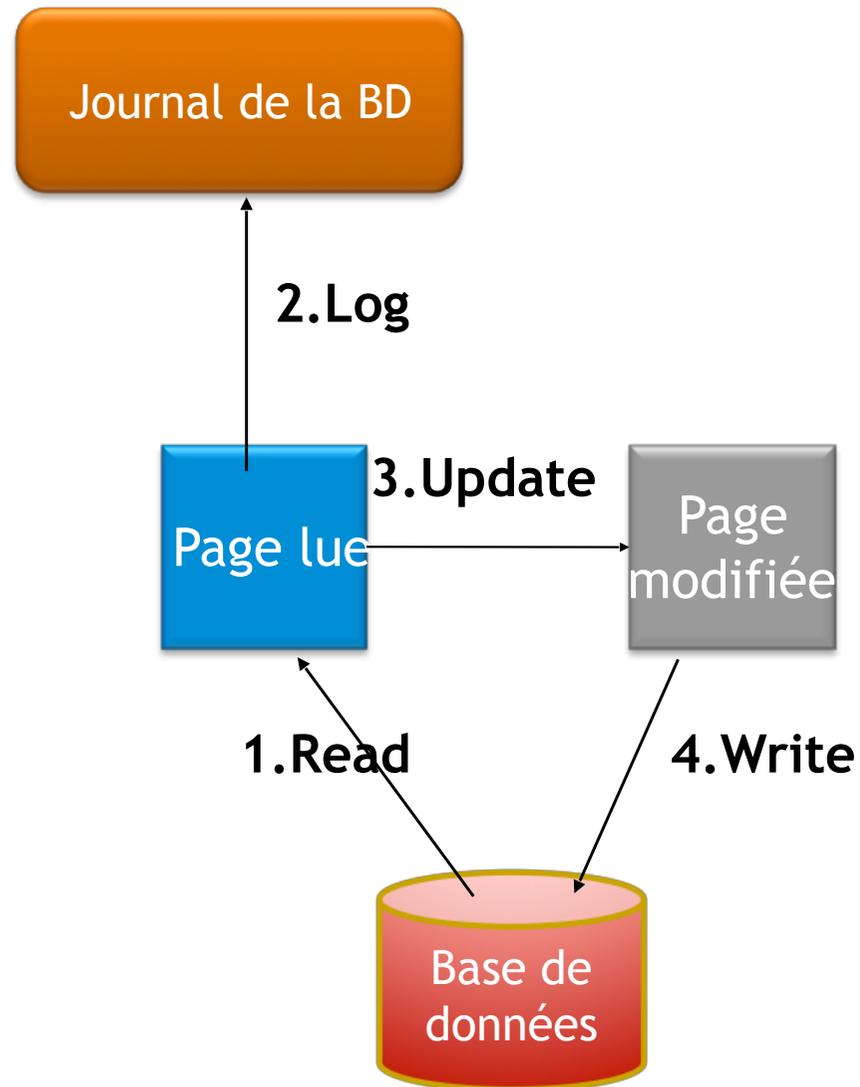


Type de journaux

- Journal des images avant
 - contient les débuts de transactions, les valeurs d'enregistrement **avant mises à jour**, les fins de transactions (commit ou abort)
 - permet de **défaire** les mises à jour effectuées par une transaction **no validé**
 - lecture à partir de la fin du journal
- Journal des images après
 - contient les débuts de transactions, les valeurs d'enregistrement **après mises à jour**, les fins de transactions (commit ou abort)
 - permet de **refaire** les mises à jour effectuées par une transaction **validé**
 - lecture à partir du début du journal

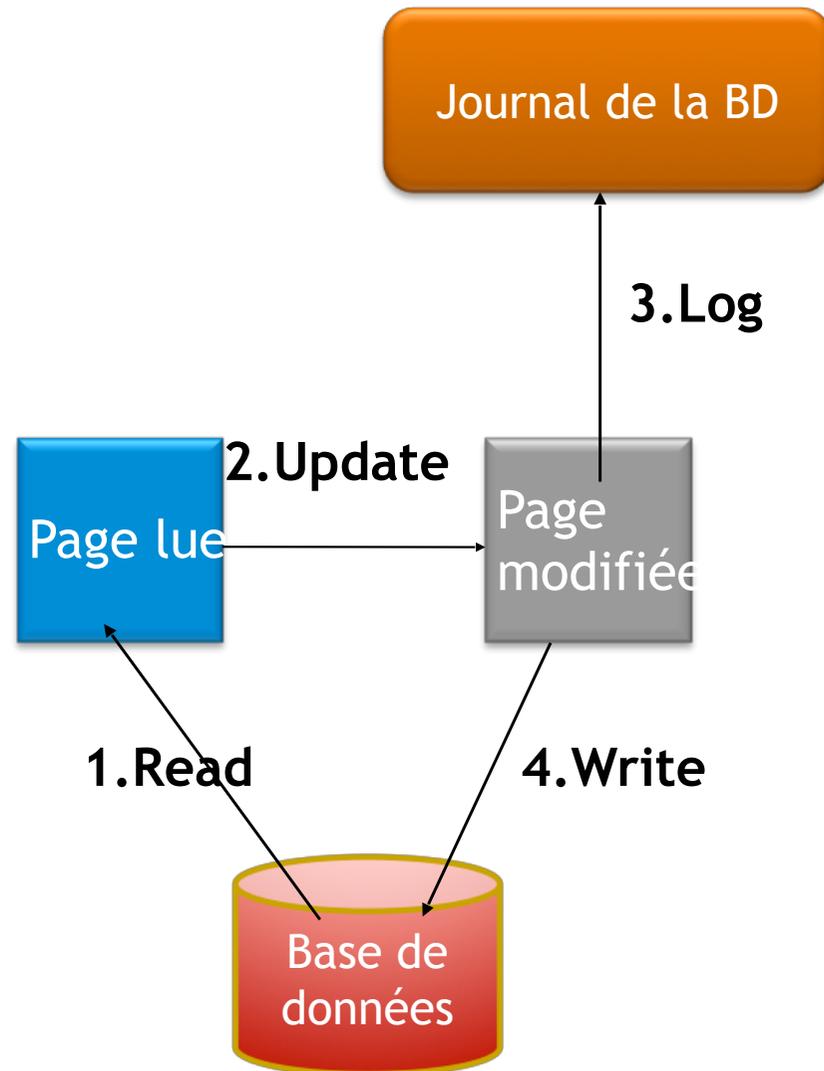
Journal des images avant

- Utilisé pour défaire les mises à jour : Undo



Journal des images après

- Utilisé pour refaire les mises à jour : Redo



Gestion du journal

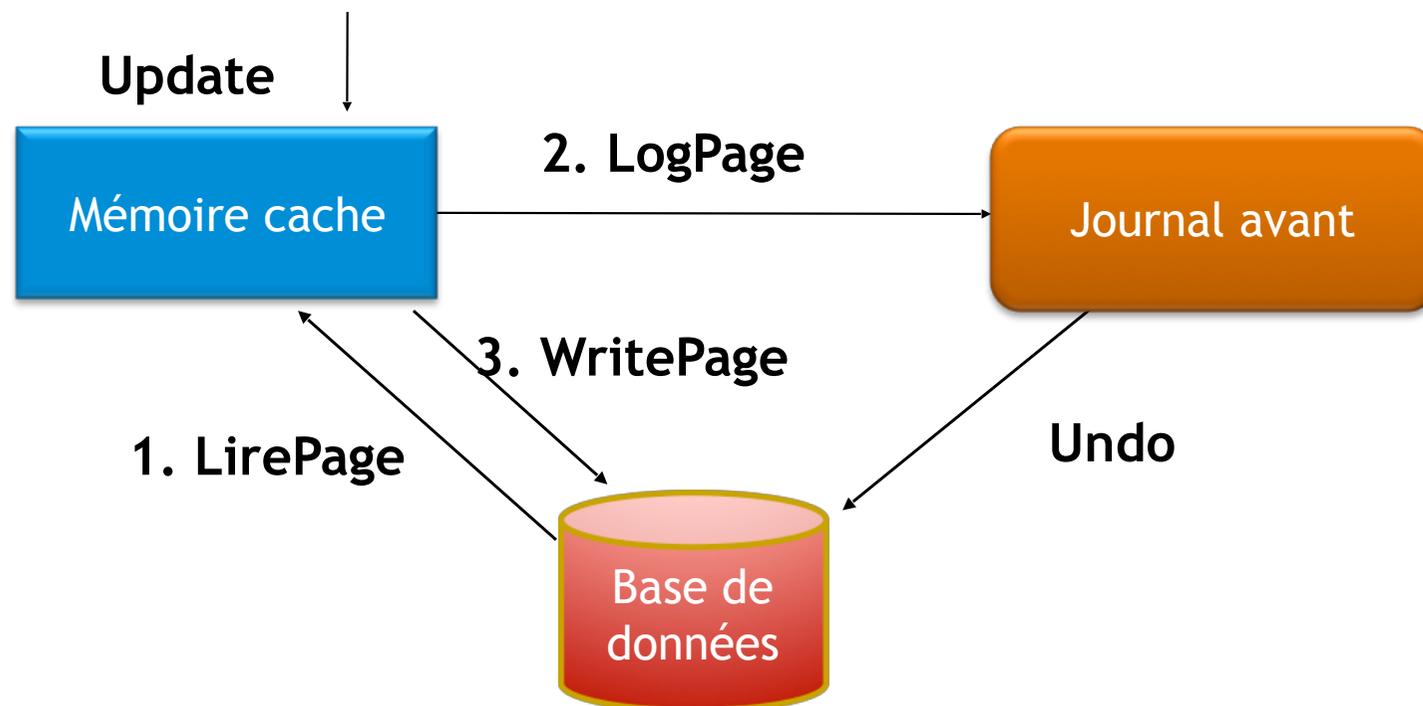
- Journaux avant et après sont unifiés
- Écrits dans un tampon en mémoire et vidés (*flush*) sur disque en début de commit
- Structure d'un enregistrement :
 - N° transaction (Tid)
 - Type enregistrement: début|update|insert|commit|abort
 - Donnée (TupleId, attribut, ...)
 - Ancienne valeur (image avant)
 - Nouvelle valeur (image après)

Scénarios de reprise

- En place : Les mises à jour sont effectuées directement dans la base
 - la base est mise à jour immédiatement, ou au moins dès que possible pendant que la transaction est active
- Différé : Les mises à jour sont effectuées en mémoire et installées dans la base à la validation (*commit*)
 - le journal est écrit avant d'écrire les mises à jour

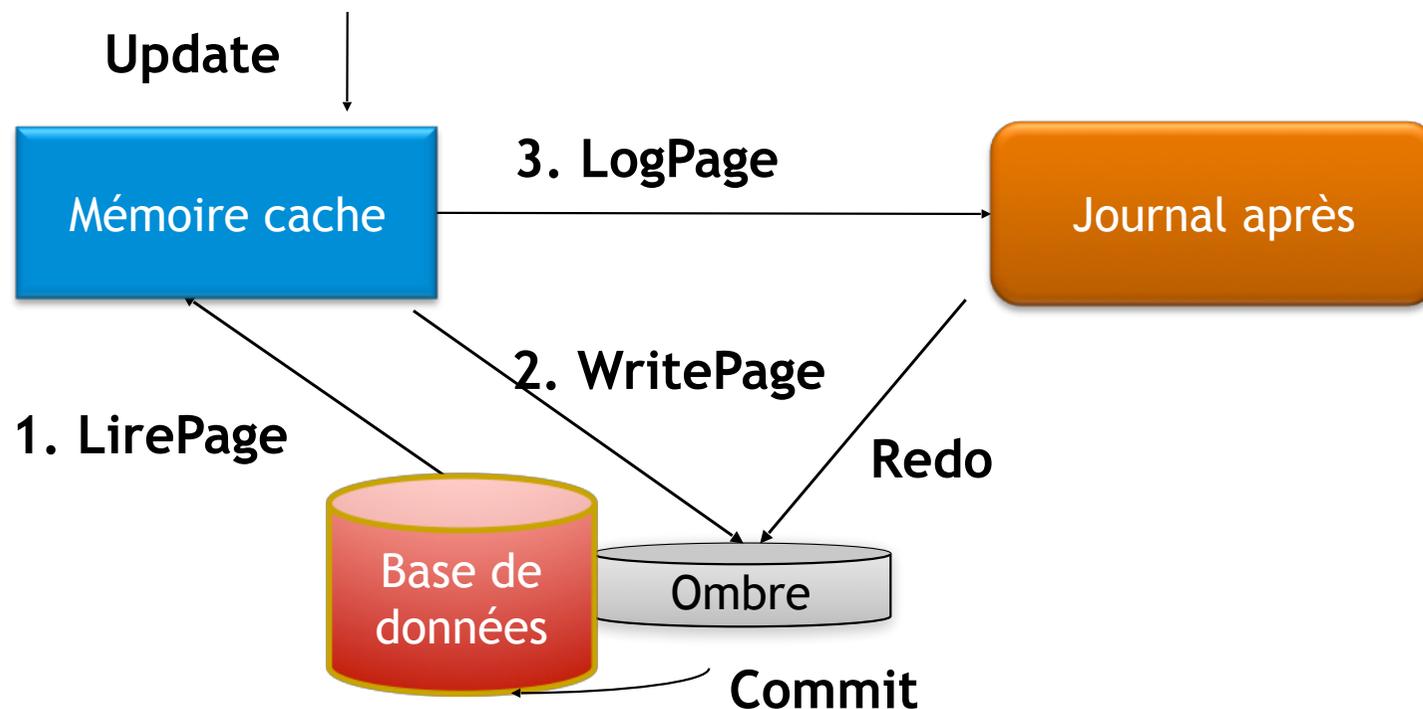
Stratégie *do-undo*

- Mises à jour en place
 - l'objet est modifié dans la base
- Utilisation des images avant
 - copie de l'objet avant mise à jour (*do*)
 - utilisée pour défaire en cas de panne (*undo*) à partir de la fin du journal



Stratégie *do-redo*

- Mises à jour en différentiel
 - l'objet est modifié en page différentielle (dans une ombre)
- Utilisation des images après
 - copie de l'objet en journal après mise à jour (*do*)
 - utilisée pour refaire en cas de panne (*redo*) à partir du début du journal



Conclusions

- ❑ Les transactions garantissent la gestion cohérente des données
- ❑ L'ordonnancement de transactions concurrentes est garanti par la **sérialisabilité**
- ❑ La reprise après panne garantit la fiabilité de la base de données
- ❑ Méthode de reprise
 - Undo, redo, points de sauvegarde, points de reprise

Méthodes de reprise

- Journalisation concept vital
- Journal des images avant permet de défaire (undo)
- Journal des images après permet de refaire (redo)
- Points de sauvegarde : se récupérer d'une panne physique, coût très cher
- Points de reprise : permettent de espacer la durée entre deux points de sauvegarde

Bibliographie

- Hector Garcia Molina, Jeffrey D. Ullman and Jennifer Widom. Database Systems. Second Edition, Pearson Prentice all, International Edition. 2009.
- Chris J. Date. Introduction aux bases de données. 8e édition. Vuibert. 2004.

