ÉCOLE POLYTECHNIQUE MONTRÉAL

Module 7:

Transactions

- **Transactions**
  - Séquence d’opérations que le serveur exécute d’une façon atomique, même en présence d'accès simultanés par plusieurs clients et de pannes.
  - Assurer le partage sécuritaire et cohérent des données des serveurs par plusieurs clients.

- **Exemple: Banque**

Chaque compte est représenté par un objet distant

Interface Account: fournit les opérations pouvant être exécutées par les clients

deposit(amount)	: deposit amount in the account
withdraw(amount)	: withdraw amount from the account
getBalance() → amount	: return the balance of the account
setBalance(amount)	: set the balance of the account to amount

Interface Branch: fournit les opérations pour les succursales de cette banque

create(name) → account	: create a new account with a given name
lookUp(name) → account	: return a reference to the account with the given name
branchTotal() → amount	: return the total of all the balances at the branch

- **Synchronisation sans transaction**

  - Opérations atomiques au niveau du serveur.

  - Utilisation de plusieurs fils au niveau du serveur.

  - Opérations des clients peuvent s'exécuter simultanément, et peuvent donc accéder simultanément le même objet.

  - Java: synchronized
    Assure qu'un seul thread accède à un objet à un instant donné

Exemple :

```
Public synchronized void deposit(int amount) {
        throws RemoteException {
                // adds amount to the balance of the account
        }
}
```

- **Exemple: transaction bancaire**

Transaction T:

a.withdraw(100);
b.deposit(100);
c.withdraw(200);
b.deposit(200);

Doit être exécutée de façon atomique :

- Pas d'interférence avec d'autres opérations simultanées déclenchées par d'autres clients

- Soit que toutes les opérations sont exécutées ou elles n'ont aucun effet si les serveurs tombent en panne

- Transaction atomique: deux propriétés à satisfaire

- Tout ou rien: soit la transaction est exécutée (ses effets sont alors permanents), ou bien elle n'a aucun effet (même en cas du panne du serveur)

- Isolation: chaque transaction est exécutée sans interférence avec d'autres transactions (effets intermédiaires invisibles aux autres transactions)

- **Propriétés des transactions (ACID) :**

Atomicité: Les effets d'une transaction ne sont rendus visibles que si toutes ses opérations peuvent être réalisées. Sinon aucun effet n'est produit par la transaction

Cohérence: les effets d'une transaction doivent satisfaire les contraintes de cohérences de l'application (la transaction transforme l'état consistant du système en un autre état consistant)

Isolation: les effets d'une transaction sont identiques à ceux qu'elle pourrait produire si elle s'exécutait seule dans le système

Durabilité: les effets d'une transaction terminée ne peuvent être détruits ultérieurement par une quelconque défaillance

- **Pannes inévitables mais objets récupérables**
  - Objets pouvant être récupérés suite à une panne de serveur. Le serveur doit stocker suffisamment d'information en mémoire stable (disque), possiblement redondante.
  - Un journal sur disque permet de lister les éléments d'une transaction avant de commettre ou d'annuler et permet en cas de panne de savoir ce qui était en train d'être fait.
  - La synchronisation des transactions pour les sérialiser est une méthode permettant l'isolation.

- **Coordonnateur de transaction :**

  - Crée et gère une transaction
  - Affecte un identificateur à la transaction (TID)

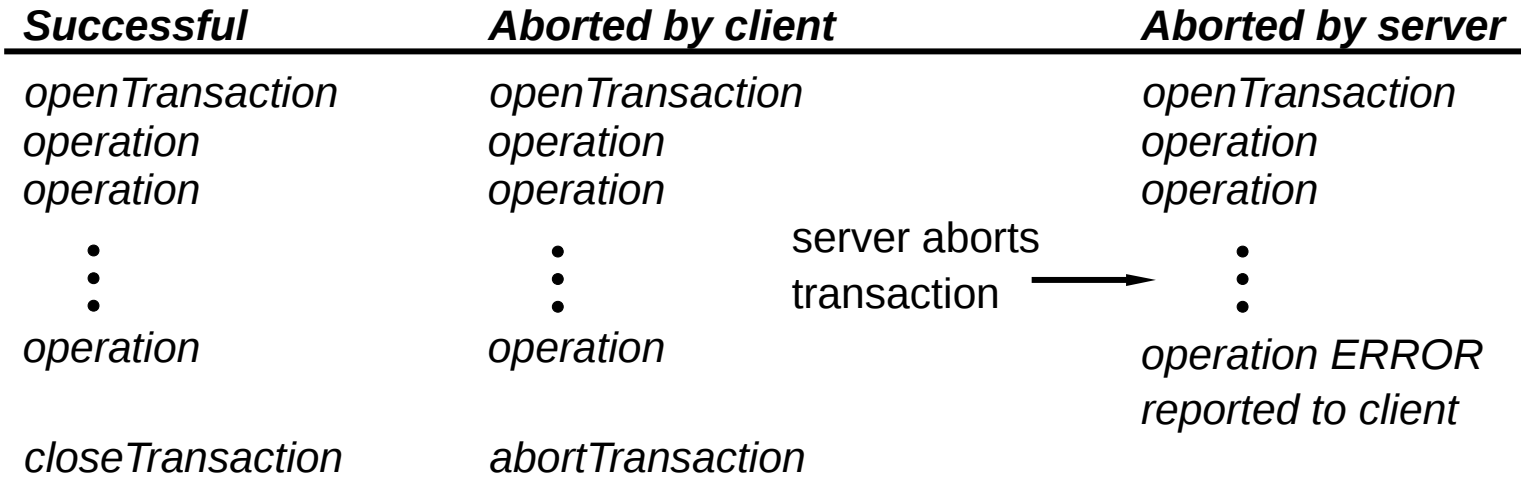
  - Opérations de l'interface Coordonnateur :

    openTransaction() → trans;
    starts a new transaction and delivers a unique TID *trans*. This identifier will be used in the other operations in the transaction.
    closeTransaction(trans) → (commit, abort);
    ends a transaction: a *commit* return value indicates that the transaction has committed; an *abort* return value indicates that it has aborted.
    abortTransaction(trans); aborts the transaction.

  - Une transaction est complétée suite à la coopération entre le programme du client, les objets récupérables et le coordonnateur

    deposit(trans, amount)
    // deposit amount in the account for transaction with TID trans

- **Cycle de vie d'une transaction :**



- **Actions possibles suite à une panne :**

- Actions du serveur :

- Abandonne toute transaction non complétée.
- Utilise une procédure de recouvrement afin de revenir à un état cohérent des objets (valeurs produites par les plus récentes transactions complétées).
- Cas de panne d'un client : le serveur accorde un délai d'expiration (timeout) au client pour son redémarrage et annule la transaction non complétée si le délai est expiré.

- Actions du client :

- Cas de panne du serveur durant une transaction: le client est averti qu'une erreur s'est produite après l'expiration d'un délai.
- Cas d'un serveur qui tombe en panne durant une transaction et redémarre avant l'expiration du délai: Il abandonne la transaction.

- **Problème d'interrogations incohérentes si les transactions sont exécutées simultanément**

Exemple :

Solde initial des comptes : A=200, B=200

Transactions : V (transfère 100$ de A vers B),
W (calcule la somme de tous les comptes de la banque)

- **Solution: sérialisation des transactions**
  - Une transaction exécutée seule ne présente pas de problème.
  - L'exécution sérialisée des transactions, une par une, permet aussi d'assurer les propriétés requises, T_1, T_2, …, T_n .
  - Entrelacement des transactions équivalent à une exécution en série, pour un ordre des transactions quelconque, est aussi acceptable.

- **Sérialisation des transactions**
  - Deux opérations sont en conflit si leur effet combiné dépend de l'ordre dans lequel elles sont exécutées.
  - Lecture-Lecture, pas de conflit.
  - Lecture-Ecriture, conflit, le résultat de la lecture est affecté par l'écriture de la variable à lire.
  - Ecriture-Ecriture, conflit, le résultat des lectures subséquentes dépend de l'ordre des deux écritures sur une même variable.

• Sérialisation des transactions (suite)

Deux transactions sont sérialisables si toutes les paires d'opérations en conflit des deux transactions sont exécutées dans le même ordre sur tous les objets qu'elles accèdent

Exemple : T : x = read(i); write(i, 10); write(j, 20);
U : y = read(j); write(j, 30); z = read(i);

Transaction T :	**Transaction U :**
x = read(i)	
write(i, 10)	
	y = read(j)
	write(j, 30)
write(j, 20)	
	z = read (i)

L'ordonnancement n'est pas équivalent à une exécution en série : paires d'opérations en conflits ne sont pas exécutées dans le même ordre sur i et j

Ordonnancement équivalent à une exécution en série : nécessite l'une des conditions suivantes :

1) T accède à i avant U et T accède à j avant U
2) U accède à i avant T et U accède à j avant T

- **Approches de contrôle de la concurrence**

Ordonnancement équivalent à une exécution en série est retenu comme critère pour la dérivation de protocoles de contrôle de la concurrence

Quelques approches :

Verrouillage (Locking) :

- Utilisé par la plupart des systèmes
- Chaque objet est verrouillé par la première transaction qui y accède
- L'objet est déverrouillé au *commit* ou *abort*

- **Approches de Contrôle de la Concurrence (suite)**

Contrôle optimiste de la concurrence :

- On suppose l'absence de conflit.

- Chaque transaction est exécutée jusqu'à la fin sans bloquer.

- Avant d'être complétée, le serveur vérifie si elle n'a pas exécuté des opérations sur des objets qui sont en conflit avec les autres opérations des autres transactions simultanées.

- Si de tels conflits existent, le serveur abandonne la transaction et le client peut la redémarrer.

- **Approches de Contrôle de la Concurrence (suite)**

Ordonnancement par estampilles:

- Les transactions et les objets ont des estampilles de temps.

- Le serveur enregistre le temps d'accès le plus récent de la lecture et de l'écriture de chaque objet et celui de chaque opération.

- L'estampille de la transaction est comparée à celle de l'objet pour déterminer si elle peut être exécutée immédiatement, différée ou abandonnée.

- **Recouvrement après l'abandon d'une Transaction**

Le serveur doit :
- Sauvegarder les effets des transactions complétées
- Assurer qu'aucun effet des transactions abandonnées ne soit stocké

Problèmes associés à l'abandon de transactions, si mal géré:

- Problème de lectures contaminées (*dirty reads*)
- Problème d'écritures prématurées (*premature writes*)

- **Recouvrement après l'abandon d'une Transaction**

Problème d'écritures prématurées:

Transaction T :		**Transaction U :**	
a.setBalance(105)		*a.setBalance(110)*	
	$100		
a.setBalance(105)	**$105**		
		a.setBalance(110)	**$110**
abort transaction (restore $100)			
		abort transaction (restore $105)	

- **Recouvrement après l'abandon d'une transaction (suite)**

Solution aux problèmes de lectures contaminées et d'écritures prématurées (suite) :

- Transaction : possède ses propres versions provisoires (tentative) des objets qu'elle met à jour.
- Écritures : faites sur les versions provisoires.
- Lectures : faites soit à partir des versions provisoires, soit à partir des versions permanentes.
- **Versions provisoires** : transférées dans les versions permanentes des objets seulement lorsque la transaction est complétée en une seule étape.
- Pendant le transfert, toutes les autres transactions n'ont pas accès aux objets modifiés.

- **Transactions Imbriquées**

Transaction imbriquée = transaction qui contient des transactions qui peuvent elles-mêmes être imbriquées

Transaction imbriquée à un niveau de la hiérarchie: n'est complétée que si son parent complète son exécution

Avantages :

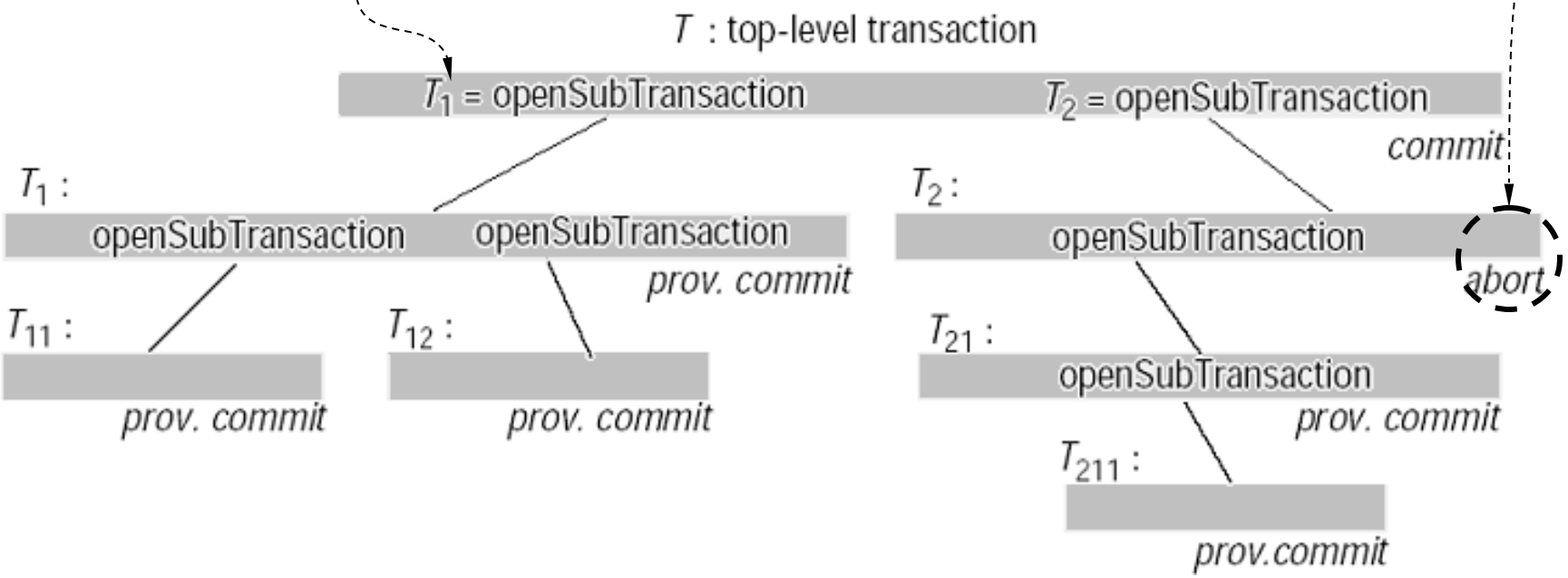
- Accroître le niveau de concurrence en permettant aux transactions imbriquées d'un même niveau dans la hiérarchie d'être exécutées simultanément

- Permet aux transactions imbriquées d'être complétées ou abandonnées indépendamment les unes des autres

• Transactions Imbriquées (suite)

Sous-transactions du même niveau, e.g. T_1 et T_2, peuvent s'exécuter simultanément, mais leurs accès aux objets partagés sont sérialisés (verrouillage)

Sous-transaction est abandonnée: le parent peut choisir une autre sous-transaction pour compléter sa tâche



• Verrouillage

Utilisation de verrous exclusifs:

- Un verrou permet de réserver l'accès unique à un objet (ressource)
- Si un client demande l'accès à un objet déjà verrouillé par une autre transaction, alors sa requête doit être suspendue et le client doit attendre jusqu'à que l'objet soit déverrouillé

Types de verrou:

- Verrou de lecture: avant une opération de lecture d'un objet, le serveur pose un verrou (non exclusif) de lecture sur l'objet.
- Verrou d'écriture: avant une opération d'écriture d'un objet, le serveur pose un verrou (exclusif) d'écriture sur l'objet.

• Exemple de verrouillage (suite)

Transaction T	Transaction U
bal = b.getBalance()	*bal = b.getBalance()*
*b.setBalance(bal*1.1)*	*b.setBalance(bal*1.1)*
a.withdraw(bal/10)	*c.withdraw(bal/10)*

Operations	Locks	Operations	Locks
openTransaction			
bal = b.getBalance()	**Lock B**		
*b.setBalance(bal*1.1)*			
		openTransaction	
a.withdraw(bal/10)	**Lock A**		
		bal = b.getBalance()	**waits for *T*'s lock on *B***
		⋮	
closeTransaction	**Unlock A, B**		
		*b.setBalance(bal*1.1)*	**Lock *B***
		c.withdraw(bal/10)	**Lock *C***
		closeTransaction	**Unlock *B* , *C***

- **Verrouillage (suite)**

Granularité :

- Verrou posé sur un objet.
- Verrou posé sur un groupe d’objets.
- Verrou global (revient à sérialiser complètement les transactions).

- Verrou régulier versus verrous de lecture et d'écriture.

• Verrouillage (suite)

Types de verrouillage :

Verrouillage à deux phases:

- 1ère phase (growing phase): la transaction acquiert de nouveaux verrous.
- 2ème phase (shrinking phase): la transaction libère ses verrous.
- Aucun verrou ne peut être posé si un verrou a été levé.

Verrouillage strict à deux phases:

- Verrouillage à deux phases dans lequel les verrous sont libérés lorsque la transaction est complétée ou abandonnée
- Prévient les lectures contaminées et les écritures prématurées
- Il n'est pas toujours nécessaire d'attendre la fin de la transaction pour libérer les verrous de lecture

- **Verrouillage (suite)**
  - Gestionnaire de verrous, maintient la table de verrous pour les objets d'un serveur.
  - Entrée pour chaque verrou:
    - objet associé au verrou;
    - type de verrou (écriture ou lecture);
    - identificateurs des transactions détenant le verrou (un seul pour écriture);
    - liste des transactions en attente du verrou.

• Verrouillage: interblocage

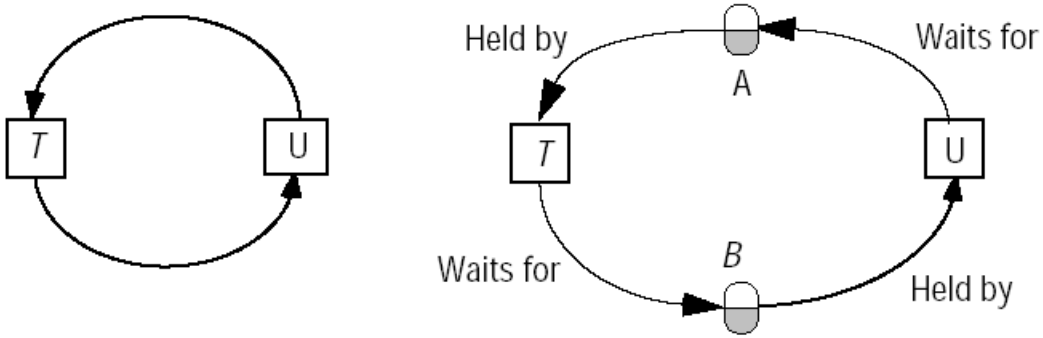
Interblocage:

- Implique au moins deux transactions.
- Caractérisé par le fait que chaque transaction est bloquée et ne peut être débloquée que par une autre transaction de l'ensemble.

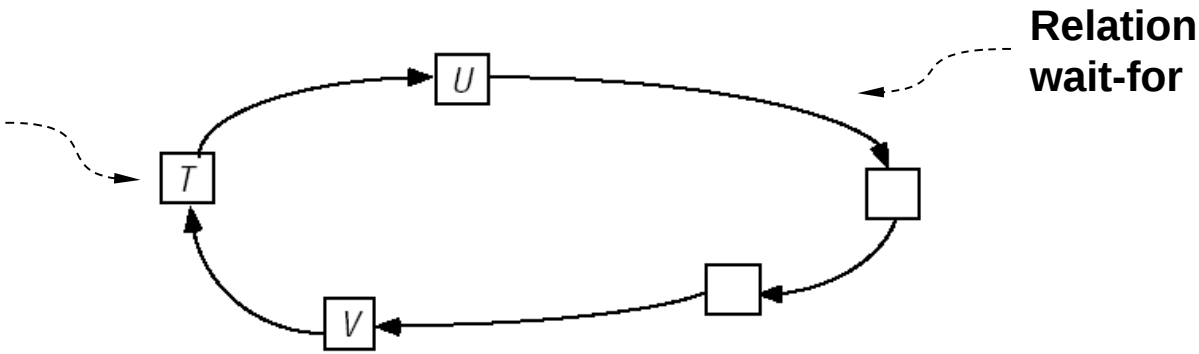
Transaction T		Transaction U	
Operations	**Locks**	**Operations**	**Locks**
a.deposit(100);	**write lock A**		
		b.deposit(200)	**write lock B**
b.withdraw(100)			
		a.withdraw(200);	
⋮	**waits for *U*'s**	⋮	**waits for *T*'s**
⋮	**lock on *B***	⋮	**lock on *A***

• Verrouillage: interblocage (suite)

Graphe de dépendance (waiting graph)



Dépendance indirecte :
T attend que U libère le verrou



- **Verrouillage : interblocage (suite)**

Exemple :

Transaction T

read(C);
write(C); (WAIT)

Transaction W

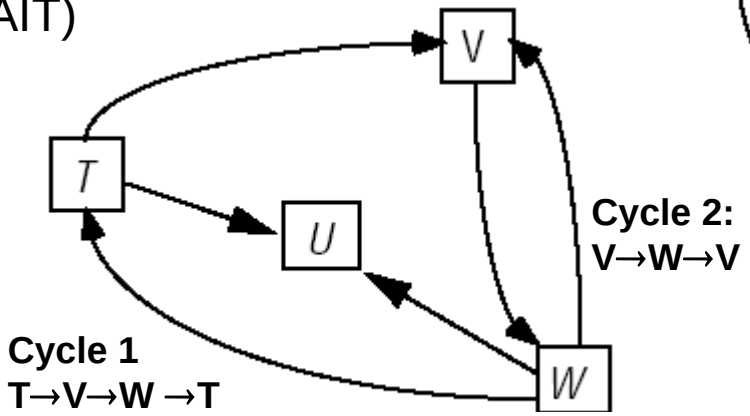
write(B);
write(C); (WAIT)

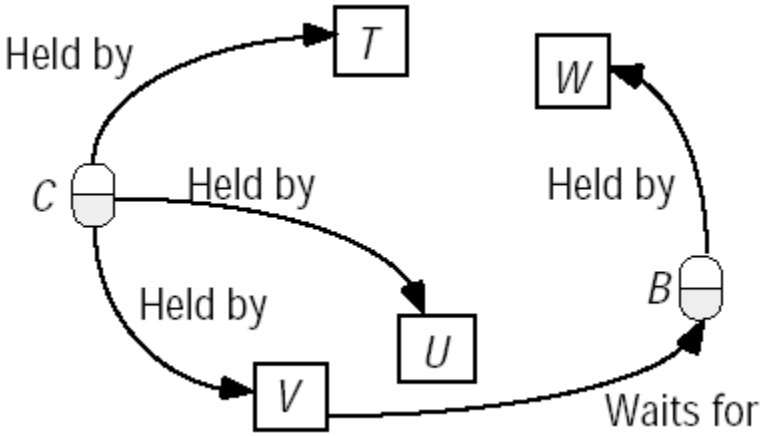
Transaction U

read(C);

Transaction V

read(C);
write(B); (WAIT)





- **Verrouillage: prévention d'interblocage**

Solution 1 : verrouiller tous les objets utilisés par la transaction dès le début.

Inconvénients :

Génère un verrouillage prématuré, réduit la concurrence

Parfois, il est impossible de prédire dès le début quels objets seront utilisés (Cas d'applications interactives)

Solution 2 : verrouiller les objets dans un ordre prédéfini.

• Verrouillage: détection d'interblocage

Solution :

• Détecter les cycles dans le graphe de dépendance.

• Abandonner une transaction appartenant à chacun des cycles.

Module responsable de la détection (peut faire partie du gestionnaire de verrous) :

• Maintient une représentation du graphe de dépendance.

• Arcs ajoutés ou supprimés dans le graphe lors des opérations sur les verrous.

• Vérification périodique de l'existence de cycle.

• Lorsqu'un cycle est détecté, une des transactions est abandonnée (choisie selon son âge, le nombre de cycles dans lesquels elle apparaît).

- **Verrouillage : résolution d'interblocage**

Délais:

- Chaque verrou pris a une période durant laquelle il est non vulnérable.
- A la fin de cette période, si une autre transaction attend après le verrou, la transaction qui le possède est annulée et le verrou libéré.
- La transaction qui attendait peut maintenant acquérir le verrou et poursuivre son travail.

• Contrôle optimiste de la concurrence COC

- Approche développée par Kung et Robinson (1981) pour palier aux inconvénients du verrouillage.

- Chaque transaction a une version provisoire des objets (tentative version) qu'elle met à jour (Il est aisé d'abandonner une transaction).

Les trois phases:

Phase de lecture:

- Lectures : exécutées d'abord sur la version provisoire, ou sur la version réelle (pas de lectures contaminées).
- Écritures : exécutées sur la version provisoire.
- Deux ensembles sont associés avec chaque transaction : ensemble d'objets lus et celui d'objets mis à jour.

Phase de validation:

- établir si les opérations ont produit des conflits ou pas.

Phase de mise à jour:

- si la transaction est validée, tous les changements seront rendus permanents.

- **COC : Validation des Transactions**
  - Un numéro d'ordre est affecté à chaque transaction au moment de l'entrée en phase de validation. Ti précède Tj si i < j.
  - Si la transaction est valide, elle complète.
  - Sinon, elle est annulée.

- **COC : Validation des Transactions**
  - Méthode vers l'arrière, en vérifiant les transactions concurrentes terminées: aucune écriture d'une variable lue par la transaction à valider.
  - Méthode vers l'avant, en vérifiant les transactions concurrentes non terminées: la transaction courante n'écrit pas de variable déjà lue par une transaction concurrente non terminée.
  - Les phases de validation et d'écriture doivent être atomiques par rapport aux phases de lecture, validation et écriture des autres transactions. Tout le reste est bloqué lorsqu'on valide et termine une transaction.

- **Validation des transactions par estampille de temps**
  - Un numéro d'ordre (temps) est affecté à chaque transaction au moment du début de la transaction. Ti précède Tj si i < j.
  - A chaque lecture, ou écriture d'une variable, on note le temps de la transaction.
  - Pour une écriture d'une variable, si la transaction est avant la dernière lecture ou avant la dernière écriture commise, la transaction est annulée, autrement l'écriture est acceptée.
  - Pour la lecture d'une variable, si la transaction est avant la dernière écriture commise, la transaction est annulée, autrement, la lecture est acceptée.
  - Des versions plus élaborées de l'algorithms peuvent lire des anciennes valeurs ou des nouvelles valeurs tentatives et ainsi éviter plusieurs conflits et augmenter la concurrence possible.

- **Gestionnaire de récupération en cas de panne**
  - Sauver les items de données sur support de stockage permanent.
  - Récupérer les données après une panne du processeur.
  - Optimiser la performance de la récupération.
  - Libérer l'espace de stockage des données intermédiaires.

- **Stockage permanent**
  - Mémoire non volatile avec écriture atomique.
  - Disque.
  - Disque et cache + pile.
  - Disques redondants.
  - Disque + ruban.
  - Rubans ou disques redondants à distance.

- **Principe de la récupération en cas de panne**
  - Toutes les nouvelles valeurs à commettre doivent être mémorisées pour les appliquer si la transaction est acceptée.
  - Toutes les anciennes valeurs doivent être conservées au cas où la transaction sera annulée.
  - Il faut éventuellement se débarrasser des nouvelles valeurs de transactions annulées, ou des anciennes valeurs de transactions acceptées.

- **Méthode du journal**
  - Ajouter à la fin du journal les nouvelles valeurs pour une transaction, et les marques pour le début, la fin ou l'annulation d'une transaction.
  - Lorsqu'une transaction est acceptée, les nouvelles valeurs sont propagées éventuellement à la base de donnée.
  - Pour récupérer, initialiser les données, lire à l'envers le journal, et ajouter toutes les valeurs provenant de transactions acceptées pour les éléments de données qui n'ont pas encore été lus du journal. Compléter avec le contenu de la base de donnée.
  - Si la récupération est arrêtée et recommencée, le résultat n'est pas modifié.
  - Lorsque le journal devient trop long, une copie de toutes les valeurs acceptées mais non propagées est écrite dans un nouveau journal, puis les entrées de transactions en cours sont ajoutées, et le nouveau journal remplace l'ancien.

- **Méthode des doubles versions**
  - Un espace de donnée suffisant pour les valeurs acceptées et les valeurs tentatives de toutes les transactions en cours est utilisé.
  - Une carte dit pour chaque variable où se trouve la valeur acceptée.
  - Pour préparer une transaction, les nouvelles valeurs sont stockées et une carte qui y réfère est créée.
  - Pour accepter une transaction, la nouvelle carte remplace l'ancienne.