Université des sciences et de la Technologie Houari Boumediene
USTHB – Alger

Département d'Informatique

MASTER SYSTÈMES INFORMATIQUES INTELLIGENTS

MASTER INFORMATIQUE VISUELLE

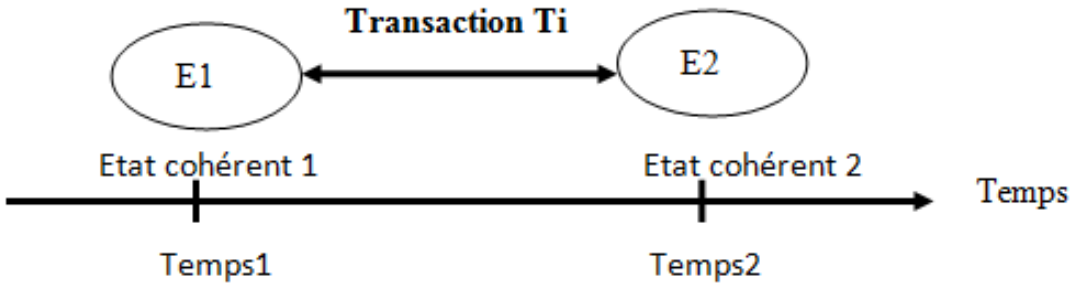
MASTER ARCHITECTURES PARALLÈLES ET CALCUL INTENSIF

ARCHITECTURE ET ADMINISTRATION DES BASES DE DONNÉES

CONTRAINTES D'INTÉGRITÉ ET TRIGGERS

TRANSACTIONS ET GESTION DE TRANSACTIONS

- **Définition** : Une transaction est une **unité de traitement séquentiel**, exécutée pour le compte d'un utilisateur, qui appliquée à une **BD cohérente**, **restitue une BD cohérente**.



- Une transaction est composée d'unités de traitement plus fines appelées **Actions**
- Une action est une commande **indivisible** exécutée pour le compte d'une transaction par le système

EXEMPLE

Soit la table Employé

Employé (numemp, nomemp, prenomemp, fonction, salaire)

Transaction qui double le salaire d'un employé

READ (numemp=x, sal)

**/ lire la donnée salaire pour le tuple dont x est la valeur de clé primaire*

*Nouv_salaire = sal*2*

WRITE (numemp=x, nouv_salaire) */* écrire la donnée salaire pour le tuple dont x est la valeur de tuple*

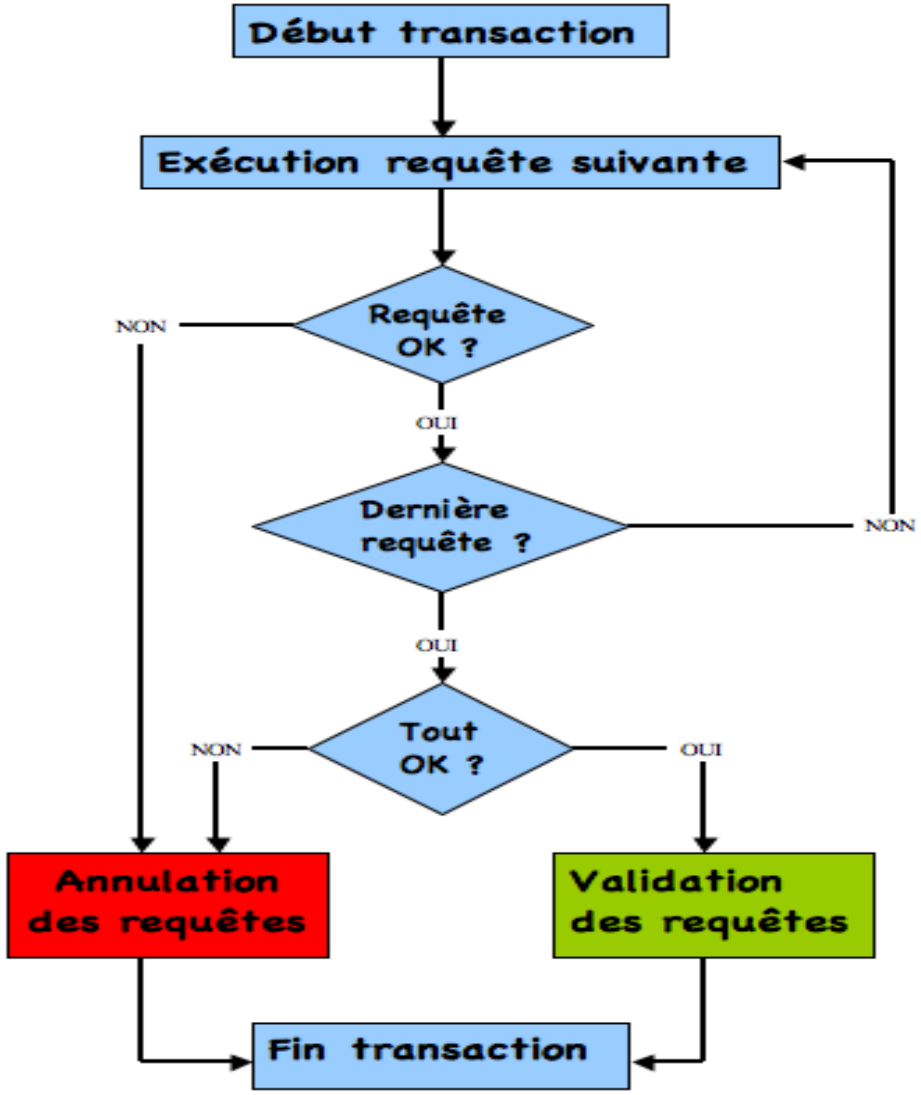
PRIMITIVES DE GESTION DES TRANSACTIONS

- Une transaction est une séquence d'actions de **lectures** et **d'écritures** suivie d'actions de **validation** (**Commit**) et ou **d'annulation** (**Rollback**).

- **Validation** : exécution en fin de transaction d'une action spéciale « Commit » qui provoque **l'intégration définitive** de toutes les mises-à-jour de la transaction courante.

- **Annulation** : exécution d'une action « **Rollback** » qui provoque **l'annulation de toutes les mises-à-jour** de la transaction courante.

ORGANIGRAMME D'UNE TRANSACTION



Propriétés d'une transaction

- Une transaction est caractérisée par quatre propriétés dites ACID (Atomicité, Cohérence, Isolation, Durabilité).

- **Atomicité**

La séquence d'actions d'une transaction est indivisible. Si toutes les actions sont effectuées alors validation sinon annulation.

- **Cohérence**

L'exécution d'une transaction dans un environnement sans pannes préserve la cohérence de la BD. Le SGBD a la responsabilité de la cohérence de la BD en vérifiant le respect des contraintes d'intégrité.

PROPRIÉTÉS D'UNE TRANSACTION

- **Isolation**

Les actions d'une transaction sont isolées, c'est-à-dire que les transactions sont indépendantes les unes des autres. Les résultats intermédiaires ou partiels d'une transaction incomplète (la validation par le COMMIT non encore réalisée), (état temporairement incohérent) sont masqués aux autres transactions.

- **Durabilité**

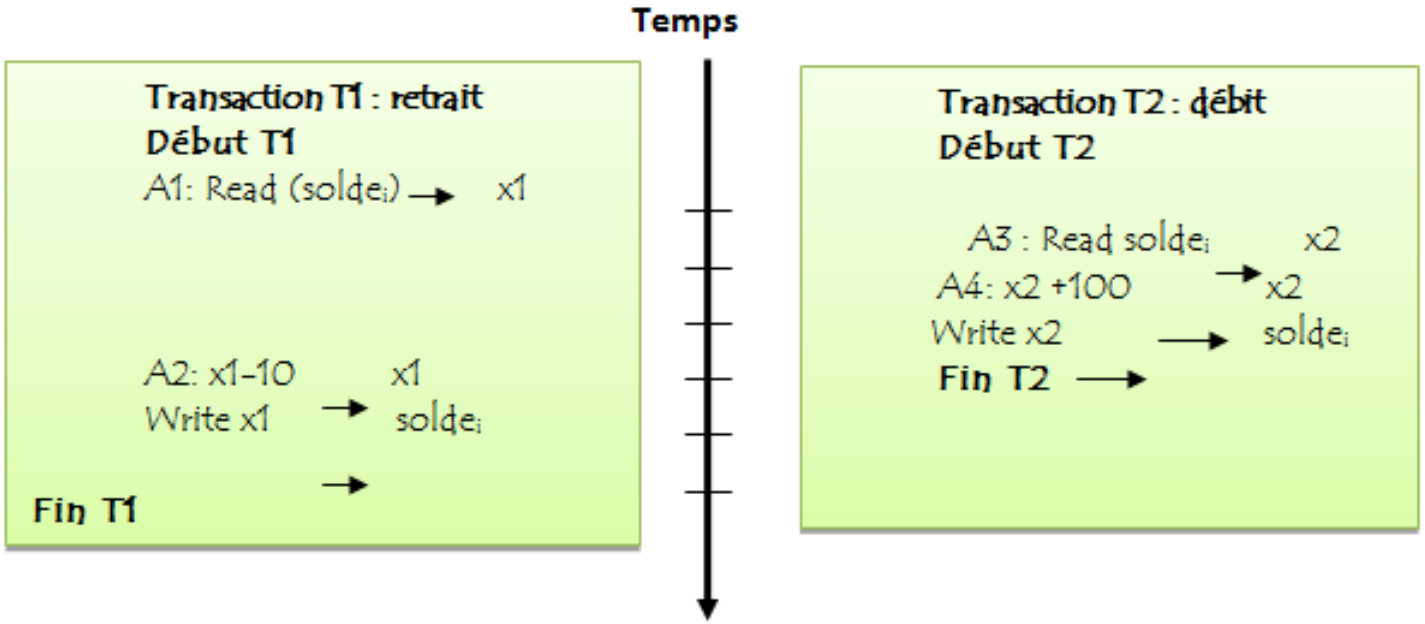
Les résultats d'une transaction complètement achevée (validée par un COMMIT) sont inscrites d'une manière durable dans la base de données. Il est impossible d'annuler les résultats d'une transaction qui s'est bien terminée.

Les problèmes de concurrence

- Si les transactions sont **parfaitement correctes** prises chacune **séparément**
- Des problèmes peuvent survenir lorsqu'elles sont lancées **en parallèle** et que leur exécution **nécessitent des entrelacements de leurs opérations** respectives.
- Trois problèmes ont été répertoriés
  - La mise à jour perdue
  - La dépendance non validée
  - L'analyse incohérente

MISE À JOUR PERDUE

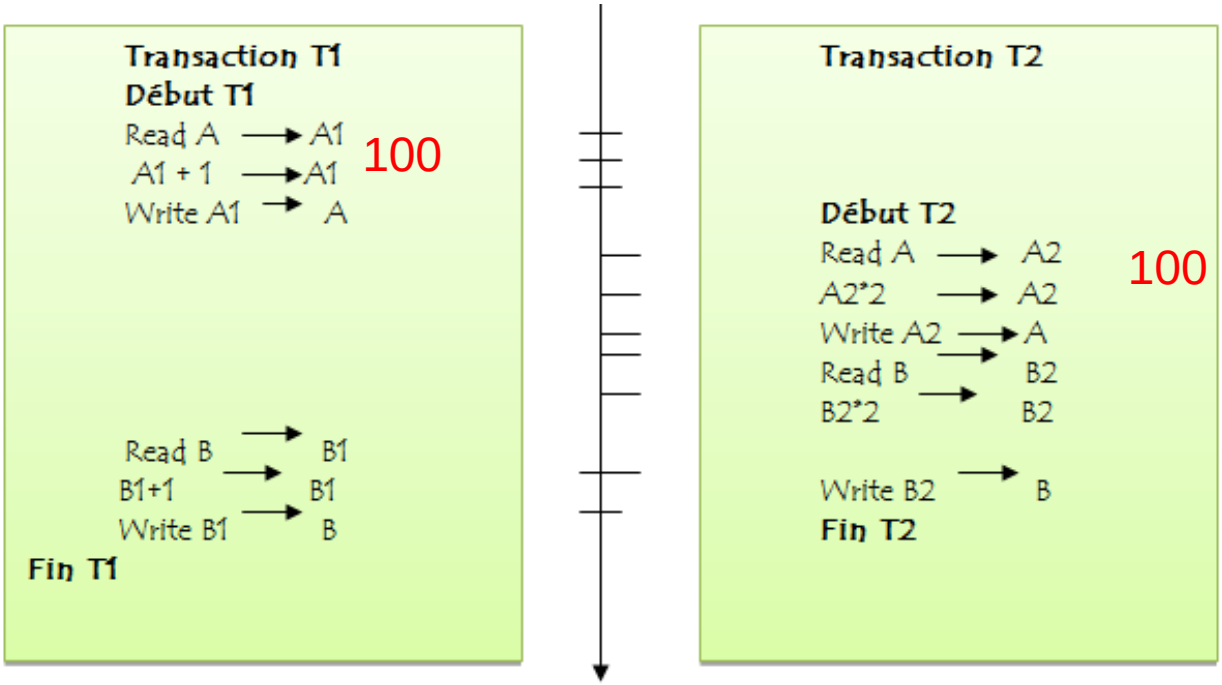
- Soit deux transactions T1 et T2 qui effectuent respectivement des débits et crédit sur le même compte i. A l'état initial le compte est à 10000 DA.



- A3 et A4 ont été effectuées entre A1 et A2, la M-A-J effectuée par A3 et A4 a été perdue.
- Il y a recouvrement de T2 par T1

DÉPENDANCE NON VALIDE

- La dépendance non validée ou lecture incorrecte, se produit lorsqu'une transaction est autorisée à voir les résultats d'une autre transaction avant que celle-ci ait été validée.
- Considérons une BD contenant les comptes A et B avec la CI : A=B



- Si Etat initial : A = B = 10
- Etat Final : A = 22, B = 11

PROBLÈME DE L'ANALYSE INCOHÉRENTE

- Il apparaît quand une transaction lit plusieurs valeurs de la BD, mais qu'une autre transaction en modifie certaines pendant l'exécution de la première transaction.

Transaction T1 : m-a-j comptes x, y

- début T1
- READ ($solde_x$)
- $solde_x = solde_x - 10$
- WRITE $solde_x$
- READ ($solde_y$)
- $solde_y = solde_y + 10$
- WRITE $solde_y$
- Fin T1

Transaction T2 : Somme

- Debut T2
- Somme = 0
- READ ($solde_x$)
- Somme = somme + ($solde_x$)
- READ ($solde_i$)
- Somme = somme + $solde_i$
- READ ($solde_y$)
- Somme = somme + ($solde_x$)
- Fin T2

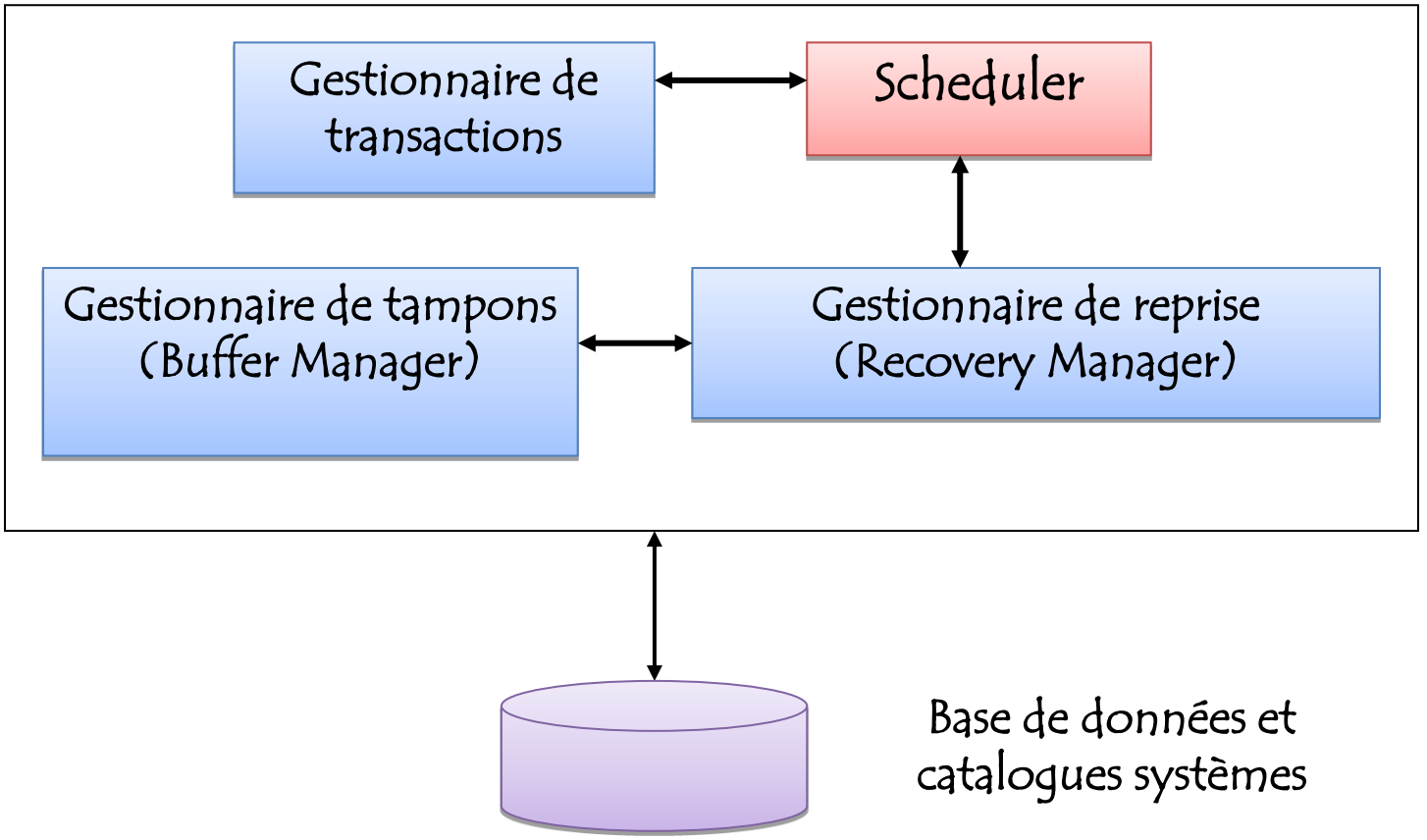
Explication : Etat initial : compte x = 100, compte y = 50, compte i = 25

T1 : transfert de 10 vers compte y donc x= 90 et y = 60

T2 : somme = 100+25+ 60 = 185 au lieu de 100+25+50= 175

RÉSOLUTION DES CONFLITS CONCURRENTS

- La résolution des conflits concurrents peut s'effectuer grâce à quatre modules de haut niveau qui gère les transactions



QUELQUES CONCEPTS

- **Granule**
  - L'unité de base de données dont l'accès est contrôlé individuellement par le contrôleur. Cela peut être une page (ensemble de table), un fichier (une table), un enregistrement (un tuple).
- **Une opération**
  - Une suite d'actions accomplissant une fonction sur un granule en respectant sa cohérence interne. Par exemple si le granule est une page, LIRE(page) et ECRIRE (page) sont les opérations de base et qui dans bien des systèmes également des actions (indivisibles).
- **Un résultat d'une opération**
  - L'état du granule concerné après application de l'opération considérée.

L'ORDONNANCEMENT

- L'ordre dans lequel les opérations des transactions dans un environnement concurrent sont exécutées.
- **Ordonnancement série** de deux transactions T1 et T2 consiste à exécuter en séquence toutes les opérations de T1, puis celles de T2 (pas de parallélisme).
- **Une exécution concurrente** sera correcte si elle produit les mêmes résultats et a les mêmes effets sur la BD qu'une exécution série de mêmes transactions.
- **Un ordonnancement série est correct**.

PRINCIPE DE SÉRIALISABILITÉ

- Critère permettant de dire que l'exécution d'un ensemble de transactions est correct.
- **L'exécution de transactions entremêlées est correcte** si elle est équivalente à une exécution des mêmes transactions en série (mêmes valeurs de départ et mêmes valeurs finales)
- **Un ordonnancement est sérialisable** s'il est correct et équivalent à un ordonnancement série formé des mêmes transactions
- **Un plan d'exécution de transactions (T1, T2, …, Tn)** (Schedule) est une séquence d'actions obtenue en intercalant les diverses actions des transactions T1, T2, …, Tn tout en respectant l'ordre interne des actions de chaque transaction.

OPÉRATIONS PERMUTABLES

Opérations Permutables

- Deux opérations Oi et Oj sont permutables si et seulement si toute exécution de Oi suivie de Oj donne le même résultat que celle de Oj suivi de Oi.
- Deux opérations travaillant sur 2 granules différents sont toujours compatibles et toujours permutables.
- O1, O3 ne sont pas permutables
- O1, O4 sont permutables

```
O1 {
Lire A -> a1
a1+1->a1
Ecrire a1->A
}
```

```
O3 {
Lire A -> a2
a2*2->a2
Ecrire a2->A
}
```

```
O4 {
Lire A -> a1
a1+10->a1
Ecrire a1-> A
}
```

Théorème

- Une condition suffisante pour qu'une exécution soit sérialisable est qu'elle puisse être transformée par permutation des opérations permutables

OPÉRATIONS CONFLICTUELLES

- Deux opérations issues de deux transactions différentes sont en conflit si elles opèrent sur le même élément de données et au moins l'une d'entre elles est une action d'écriture.

Ti :READ(X) Tj : WRITE(X)

Ti : WRITE(X) Tj : WRITE(X)

Ti : WRITE(X) Tj : READ(X)

- **Notion de précédence**

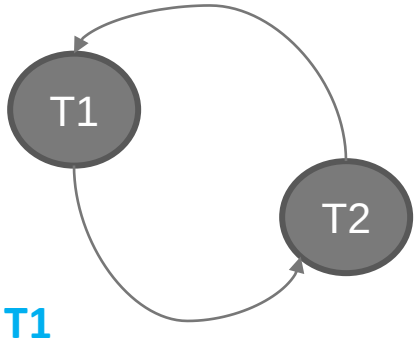
Il y a précédence de Ti par Tj (Ti précède Tj, notée Ti <Tj) dans une exécution E si et seulement si, il existe deux opérations non permutables Oi et Oj telle que Oi est exécutée par Ti avant Oj par Tj.

GRAPHE DE PRÉCÉDENCE

- Un graphe de précédence est un graphe dirigé G=(N,F) tel que : N ensemble de nœuds, F ensemble de flèches :
- créer un nœud pour chaque transaction
- créer une flèche dirigée $T_i \rightarrow T_j$ si T_j lit la valeur d'un élément écrit par T_i
- créer une flèche dirigée $T_i \rightarrow T_j$ si T_j écrit une valeur d'un élément après qu'il ait été lu par T_i
- créer une flèche dirigée $T_i \rightarrow T_j$ si T_j écrit une valeur d'un élément après qu'il ait été écrit par T_i
- Si une flèche existe de Ti vers Tj pour un ordonnancement O, alors dans tout ordonnancement sérialisable de O, Ti doit apparaître avant Tj.
- Si le graphe de précédence contient un cycle, l'ordonnancement n'est pas sérialisable.

R1(S) R2(S) R1(M) R1(C) W1(M) R2(M) R2(C) W2(M) W1(C) Commit1 W2(C) Commit2

2 Transactions T1 et T2 et 3 variables S, M et C



Pour S : R1(S)R2(S) Aucune precedence

Pour M : R1(M)W1(M)R2(M)W2(M) : T1 precede T2

Pour C : R1(C)R2(C)W1(C)W2(C) : T1 precede T2, T2 Precede T1

R6(X) R8(Y) W8(Y) W6(X)R5(X) W5(X) R7(X) R5(Y) W7(X) W5(Y)R8(Z) R7(Y) W7(Y)W8(Z)

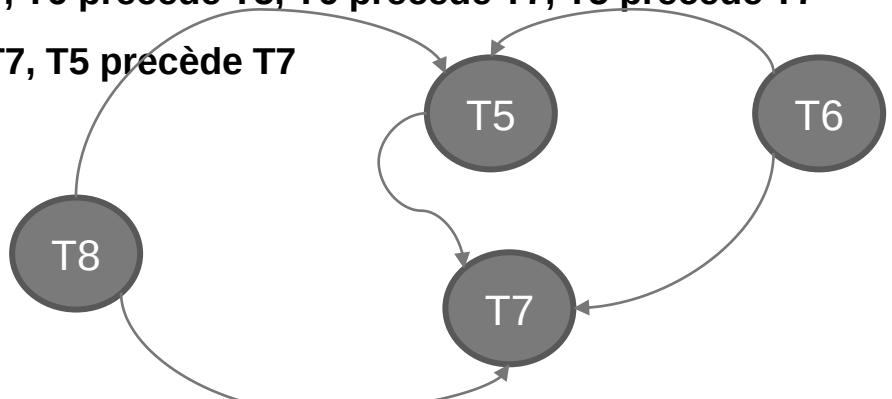
T5, T6, T7, T8

X, Y, Z

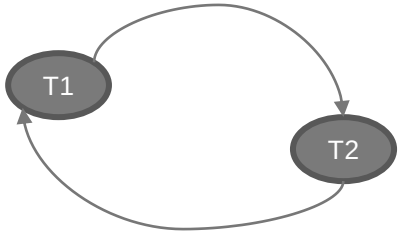
Pour X : T6 precède T5, T6 prcède T7, T6 precède T5, T6 precède T7, T5 precède T7

Pour Y : T8 precède T5, T8 Precède T7, T5 precède T7

Pour Z : aucune



R1(S) R2(S) R1(M) R1(C) W1(M) R2(M) R2(C) W2(M) W1(C) Commit1 W2(C) Commit2



2 transactions T1, T2, Granules : S, M et C

Granule S : Aucune

Granule M : T1 precede T2

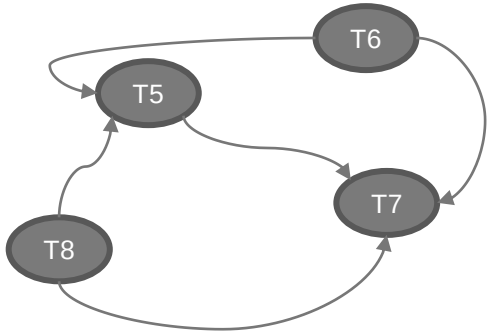
Granule C : T1 precede T2, T2 Precède T1

R6(X) R8(Y) W8(Y) W6(X) R5(X) W5(X) R7(X) R5(Y) W7(X) W5(Y)R8(Z) R7(Y) W7(Y)W8(Z)

T5,T6,T7,T8, Granules : X, Y, Z

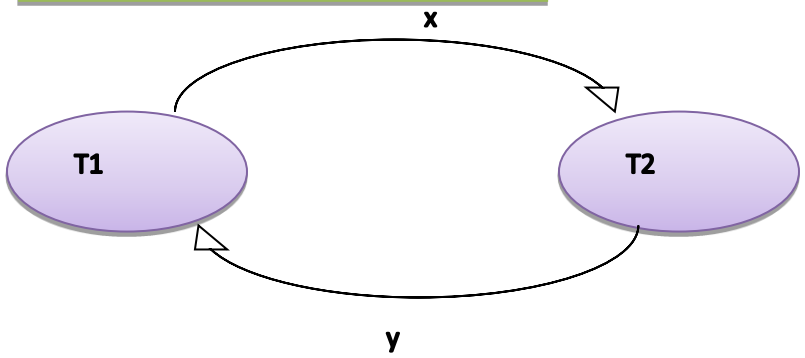
Pour X : T6 precede T5 et T7, T5 precede T7

Pour Y : T8 precede T5 et T7, T5 precede T7



EXEMPLE

	Temps	
Début T1		
Read (solde(x))		
Solde(x)= solde(x) + 100		
Write (solde(x))		
		Début T2
		Read solde(x))
		Solde(x))= solde(x))*1.1
		Write solde(x))
		Read solde(y))
		Solde(y))=solde(y))*1.1
		Write solde(y))
Read (solde(y))		End T2
Solde(y)= solde(y) + 100		
Write solde(y)		
End T1		



T1 précède T2 car T2 lit x écrit par T1

T2 précède T1 car T1 lit y écrit par T2

MÉTHODES DE SÉRIALISATION

1. Sérialisation par estampillage
2. Sérialisation par Verrouillage

L'ESTAMPILLAGE

- **Concept d'estampille**
  - Une estampille est une valeur numérique unique créée par le SGBD indiquant l'heure de démarrage relative d'une transaction
- Pour chaque transaction on lui associe une estampille. Deux manières possibles
  - L'horloge système
  - Utiliser un compteur.
- **Protocole d'estampillage**
  - un protocole de contrôle de concurrence qui classe les transactions dans un ordre tel que les transactions les plus anciennes (qui portent l'estampille la plus petite) obtiennent la plus grande priorité en cas de conflits.

ALGORITHME BASÉ SUR LES ESTAMPILLES

- L'algorithme consiste à vérifier que les accès aux données par les transactions s'effectuent bien dans l'ordre affecté au lancement repéré par les estampilles.
- Principe : si une transaction tente de lire ou d'écrire une donnée, alors la lecture ou l'écriture n'est autorisée que si la dernière mise à jour de la donnée a été effectuée par une transaction plus ancienne.
- Sinon la transaction qui a demandée la lecture ou l'écriture est reprise avec une nouvelle estampille. (pour éviter qu'elles soient continuellement annulées ou redémarrées)

Si Ti émet un Read (A)

Si E(A) =< i alors exécution de la lecture et E(A) :=i
Sinon, l'action Read est rejetée, la transaction aussi.

Si Ti émet un Write (A)

Si E(A) =< i alors exécution de l'écriture et E(A) := i
Sinon refuser l 'écriture et rejeter la transaction.

VERSION 2 DE L'ALGORITHME

- Amélioration de l'algorithme en utilisant deux estampilles pour la donnée : estampille de lecture (EL(A)) et estampille d'écriture (EE(A)).

La transaction T émet un Read (A)

si E(T) < EE(A) : {*une transaction plus ancienne essaie de lire une donnée qui a été mise à jour par une transaction plus récente*} alors l'action Read est rejetée et T est reprise.

Sinon, E(T) >= EE(A), exécution de Read (A) et EL(A) = max (E(T), EL(A))

La transaction T effectue un Write (A)

Si E(T) < EL(A) : {*une transaction plus récente utilise déjà la valeur courante de la donnée, ce qui serait une erreur de la modifier maintenant}*, alors l'action Write est rejetée et reprise de la transaction.

Si E(T) < EE(A) : {*la transaction T essaie d'écrire une valeur obsolète de la donnée}, alors rejeter l'action et reprise de la transaction.*

Sinon, {E(T) >= EL(A) et E(T) >= EE(A)}, exécution de l'opération d'écriture et EE(A)= E(T).

EXEMPLE

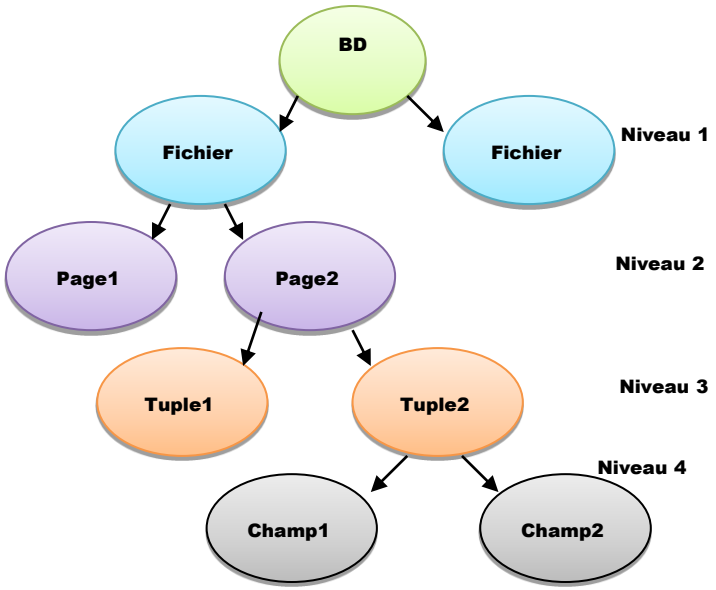
R5(X) R6(X) R7(X) R8(Y) W8(Y) W5(X) W7(X) R7(Y) W7(Y) R5(Y) W5(Y) W6(X) R8(Z) W8(Z)

SÉRIALISATION PAR VERROUILLAGE

- Procédure employée pour contrôler les accès concurrents aux données en utilisant des verrous.
  - **Verrou partagé :** si une transaction dispose d'un verrou partagé SLOCK sur une donnée, elle peut lire la donnée mais pas la modifier
  - **Verrou exclusif :** si une transaction dispose d'un verrou exclusif XLOCK sur une donnée, elle peut lire et modifier cette donnée.
- A chaque ressource est donc associée
  - Une action de verrouillage qui peut être deux types :
    - **XLOCK(A) (verrou exclusif)** qui permet à une transaction d'acquérir un contrôle exclusif de l'objet X
    - **SLOCK(A) (verrou partagé)** qui permet à une transaction de partager la lecture sur A. Ils sont aussi appelés verrou d'écriture (XLOCK) et verrou de lecture (SLOCK)
  - Une action de déverrouillage **UNLOCK(X)** qui permet la libération de l'objet X pour lequel on disposait d'un contrôle exclusif ou partagé.

GRANULARITÉ DES DONNÉES

- La granularité est la taille des données choisies comme l'unité de protection par un protocole de contrôle de concurrence.
- Elle peut être
  - La BD dans sa totalité
  - Un fichier
  - Une page
  - Un enregistrement (un tuple)
  - La valeur de champ d'un enregistrement

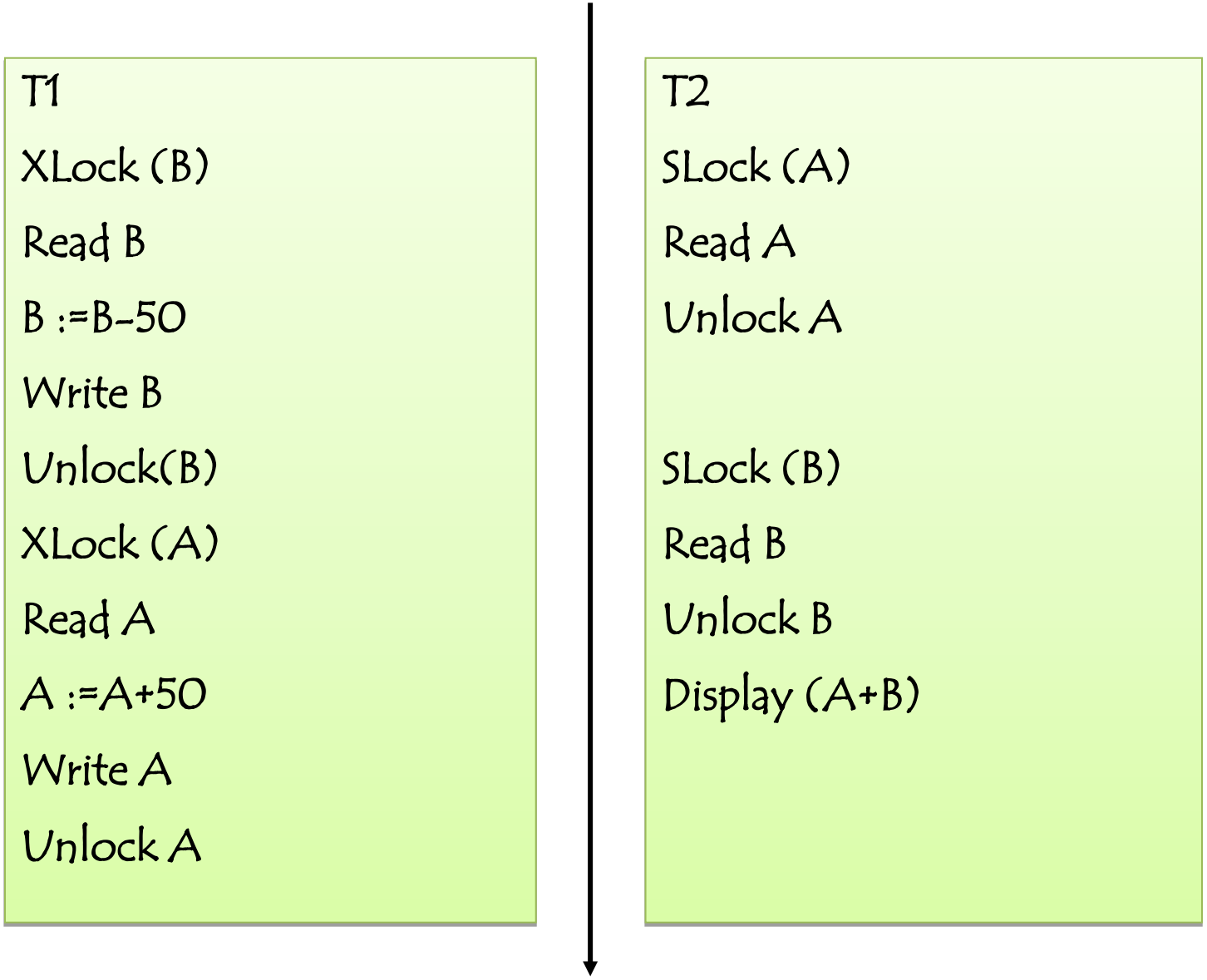


- Chaque fois qu'un nœud est verrouillé ses descendants le sont aussi.
- Si une transaction demande un verrou sur un granule d'un niveau i, alors les SGBD vérifie s'il n'existe pas de verrous sur les nœuds i-1 jusqu'à la racine.

PRINCIPE DE MANIPULATION DES VERROUS

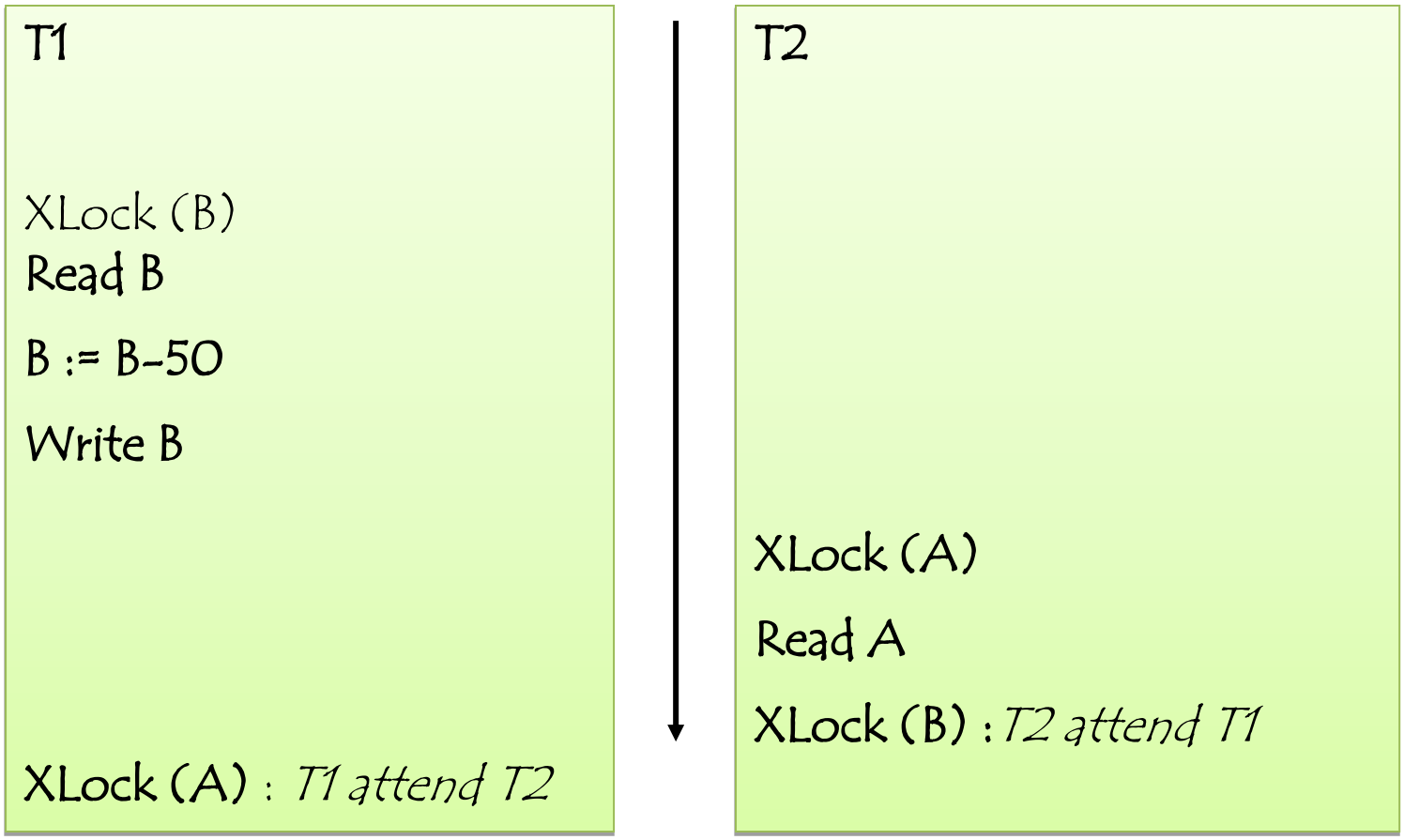
- Toute transaction devant accéder à une donnée doit la verrouiller par **SLOCK** ou **XLOCK**
- Si la donnée n'est pas déjà verrouillée, celui-ci est accordé.
- Si la donnée est déjà verrouillée, le SGBD vérifie la compatibilité avec le verrou actuel
  - Si c'est un SLOCK que la transaction demande et que le verrou déjà placé est lui aussi un SLOCK, alors la demande peut être satisfaite
  - Sinon (verrou exclusif), la transaction doit attendre que le verrou se libère.
- Une transaction qui détient un verrou le conserve tant qu'elle ne le libère pas explicitement pendant l'exécution ou implicitement lorsqu'elle se termine.

EXEMPLE


T1
XLock (B)
Read B
B :=B-50
Write B
Unlock(B)
XLock (A)
Read A
A :=A+50
Write A
Unlock A
T2
SLock (A)
Read A
Unlock A
SLock (B)
Read B
Unlock B
Display (A+B)

PROBLÈME DU VERROUILLAGE MORTEL (DEADLOCK)

- Le verrouillage mortel est l'impasse générée par deux (ou plusieurs) transactions qui sont en situation d'attente mutuelle que des verrous se libèrent.



PROTOCOLE DE VERROUILLAGE À DEUX PHASES

- Une transaction suit le protocole de verrouillage en deux phases si toutes les opérations de verrouillage **précèdent** la première opération de déverrouillage dans la transaction

- Chacune des transactions de la séquence émet des demandes de verrouillage et déverrouillage en deux phases :

  - **Verrouillage croissant** : une transaction peut obtenir des verrouillages mais pas de nouveau déverrouillage.

  - **Verrouillage décroissant (appelée aussi phase de résorption):** une transaction peut obtenir des déverrouillages mais pas de nouveau verrouillage.

PROTOCOLE DE VERROUILLAGE À DEUX PHASES

- Si une transaction T veut lire (respectivement modifier) un objet, elle demande un *verrou partagé* (respectivement un *verrou exclusif)* sur l'objet.
- Une transaction peut relâcher un verrou avant de se terminer, mais elle ne peut plus redemander de verrou une fois qu'elle en a relâché un.

THÉORÈME DU V2P

- Si toutes les transactions satisfont le protocole de verrouillage à deux phases, tous les ordonnancements entrelacés sont sérialisables.

- Techniques de gestion des verrous indéfinis
  - Imposer un délai
  - Prévenir les verrous indéfinis
  - Les détecter et les récupérer.

PRÉVENTION DES VERROUS INDÉFINIS

- **Délais impartis :** le SGBD impose des délais pour la libération de verrous. Si une transaction atteint ce délai, il est fort probable que soit la transaction est en deadlock, auquel cas, le SGBD va l'annuler puis va la reprendre.
- Prévention des verrous indéfinis :
  - **Algorithme Wait-Die (attendre-mourir) :** il permet à une transaction plus ancienne d'attendre une transaction plus récente, sinon la transaction est annulée (meurt) puis reprise avec la même estampille (elle devient la plus ancienne et ne meurt pas).
  - **Algorithme Wound-Wait (blessé attend) :** seule une transaction plus récente peut attendre une transaction plus ancienne.
  - **V2P conservateur :** au moment ou la transaction démarre, elle demande tous ses verrous et ne démarre réellement que si elle les obtient tous. Ainsi les transactions ne sont jamais bloquées.

DÉTECTION DES VERROUS INDÉFINIS GRAPHE D'ATTENTE

- La détection des verrous indéfinis se fait par l'utilisation **des graphes d'attentes**
- Une transaction Ti dépend d'une transaction Tj quand la transaction Tj détient un verrou sur la donnée que Ti attend.
- Le graphe des attentes est un graphe dirigé **G= (N,F)** constitué d'un ensemble de nœuds N et d'un ensemble de flèches F construit comme suit
  - Créer un nœud pour chaque transaction
  - Créer une flèche de Ti vers Tj quand la transaction Ti attend de verrouiller une donnée actuellement verrouillé pat Tj.
- **Un verrou indéfini** existe si et seulement si le graphe des attentes contient un **cycle**.

EXEMPLE

O1 : R1(X) R2(Y) W1(X) R3(Y) R2(X) W3(Y) R2(Z) R3(Z) W2(Z) W3(Z)

- **Précédence**
- **Estampillage**

Transaction	Action	EL(X)	EE(X)	EL(Y)	EE(Y)	EL(Z)	EE(Z)
		0	0	0	0	0	0
T1	R1(X)						
T2	R2(Y)						
T1	W1(X)						
T3	R3(Y)						
T2	R2(X)						
T3	W3(Y)						
T2	R2(Z)						
T3	R3(Z)						
T2	W2(Z)						

Verrouillage

	X	Y	Z
S	~~T1 T2~~	~~T2,~~ T3	~~T2~~
X	~~T1~~		
Attente	~~T3~~		

R1(X) R2(Y) W1(X) Fin T1 R3(Y) R2(X) W3(Y) R2(Z) R3(Z) W2(Z) Fin T2 W3(Z) Fin T3

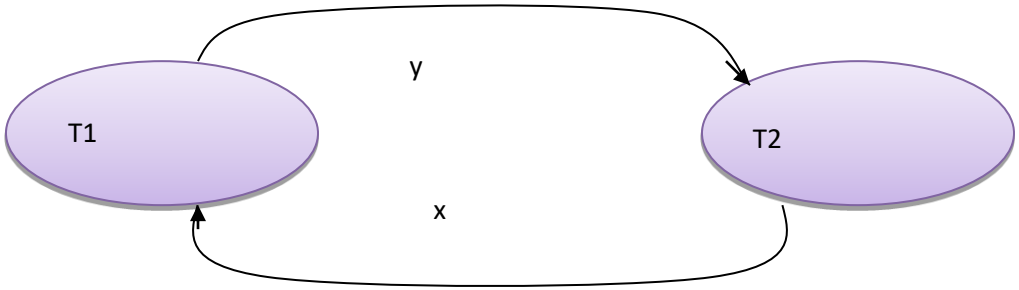
Transaction	Action	Demande Verrou	Réponse	Etape 2
T1	R1(X)	SLOck(X)	OUI	
T2	R2(Y)	SLOCK(Y)	OUI	
T1	W1(X) Fin T1	XLOCK(X)	OUI, U(X)	
T3	R3(Y)	SLOCK(Y)	OUI	
T2	R2(X)	SLOck(X)	OUI	
T3	W3(Y)	XLOCK(Y)	NON, T3 Attend T2	OUI
T2	R2(Z)	SLOCK(Z)	OUI	
T3	R3(Z)	En attente		OUI
T2	W2(Z) Fin T2	Xlock(Z)	OUI, U(X,Y,Z)	
T3	W3(Y)	Réveiller T3		OUI
T3	R3(Z)			OUI
T3	W3(Z) Fin T3			OUI

R4(X) W4(X)R6(X)R5(Y)R5(X)R5(Z)R6(Y)W5(Y)W6(X)W6(Z)W5(X)

Transaction	Action	Demande Verrou	Réponse
T4	R4(X)		
T4	W4(X)		
T6	R6(X)		
T5	R5(Y)		
T5	R5(X)		
T5	R5(Z)		
T6	R6(Y)		
T5	W5(Y)		
T6	W6(X)		
T6	W6(Z)		
T5	W5(X)		

EXEMPLE

T1	T2
Début T1	
XLOCK solde(x)	Début T2
Read solde(x)	XLOCK (solde(y))
Solde(x)=solde(x)610	Read (solde(y))
Write solde(x)	Solde(y)=solde(y)+100
XLOCK solde(y)	Write (solde(y))
Attente	XLOCK(solde(x))
Attente	Attente
Attente	Attente
…	…



REPRISE APRÈS PANNES