Introduction à la concurrence d’accès

Le problème

- Nécessité d'exécuter les programmes et requêtes de façon concurrente
- Garantir que l’exécution simultanée de plusieurs programmes se fait correctement
  - Cohérence de la base de données
  - Exécution correcte des programmes
- Exemples de contextes hautement concurrentiels:
  - annuaires téléphoniques (accès en lecture)
  - systèmes de réservation de billets (lecture et mise à jour)

Transaction

- Transaction = unité de programme exécutée sur SGBD
  - Début Transaction
    - Accès à la base de données (lectures, écritures)
    - Calculs en mémoire centrale
  - Fin Transaction : **COMMIT** ou **ROLLBACK**
- Commit : exécution correcte (validation)
- Rollback : exécution incorrecte (effacement)
- Définition: Une transaction est l'ensemble des instructions séparant un commit ou un rollback du commit ou du rollback suivant.

Transaction…

- On adopte alors les règles suivantes :

1. Quand une transaction est validée (par *commit*), toutes les opérations sont validées ensemble, **et on ne peut plus en annuler aucune**. En d'autres termes les mises à jour deviennent définitives.

2. Quand une transaction est annulée par *rollback* ou par une panne, on annule toutes les opérations depuis le dernier *commit* ou *rollback*, ou depuis le premier ordre SQL s'il n'y a ni *commit* ni *rollback*.

Transaction…

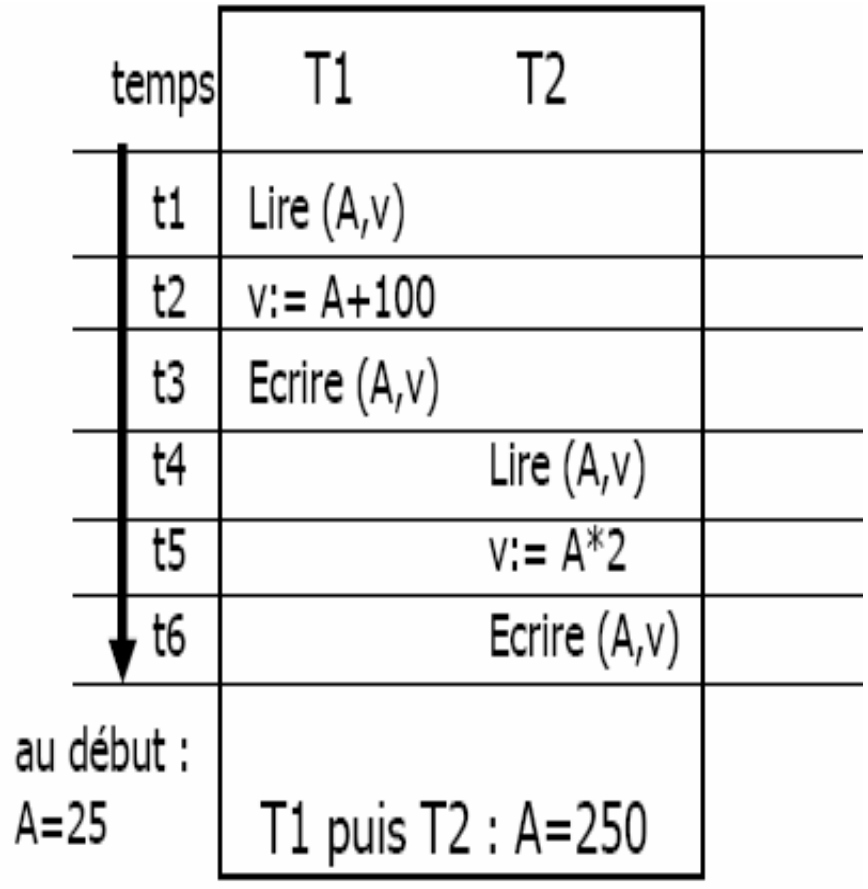
- Problèmes à éviter:
  - Perte de mise à jour
  - Données fantômes
  - Analyse incohérente

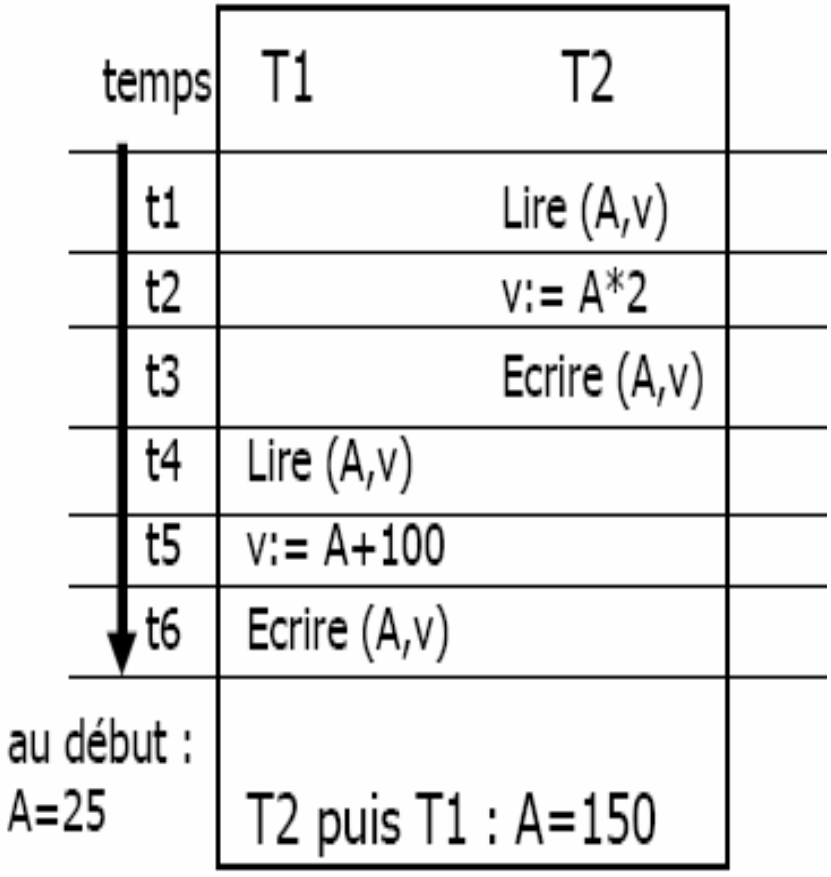
- Propriétés d'une transaction (ACID)
  - Atomicité : opérations : tout ou rien
  - Cohérence : BD cohérente avant – BD cohérente après
  - Isolation: les transactions s'ignorent i.e. chaque transaction doit s'exécuter comme si aucune autre ne s'exécutait en même temps
  - Durabilité: le résultat acquis par une transaction terminée et validée est garanti même en cas de défaillance du système

Sérialisabilité

- Hypothèses:
  - Exécution d'une transaction individuelle est correcte
  - Exécution de transactions en série (les unes derrière les autres) est correcte
- Idée: se ramener à une exécution de transactions en série
- Concept de **sérialisabilité**

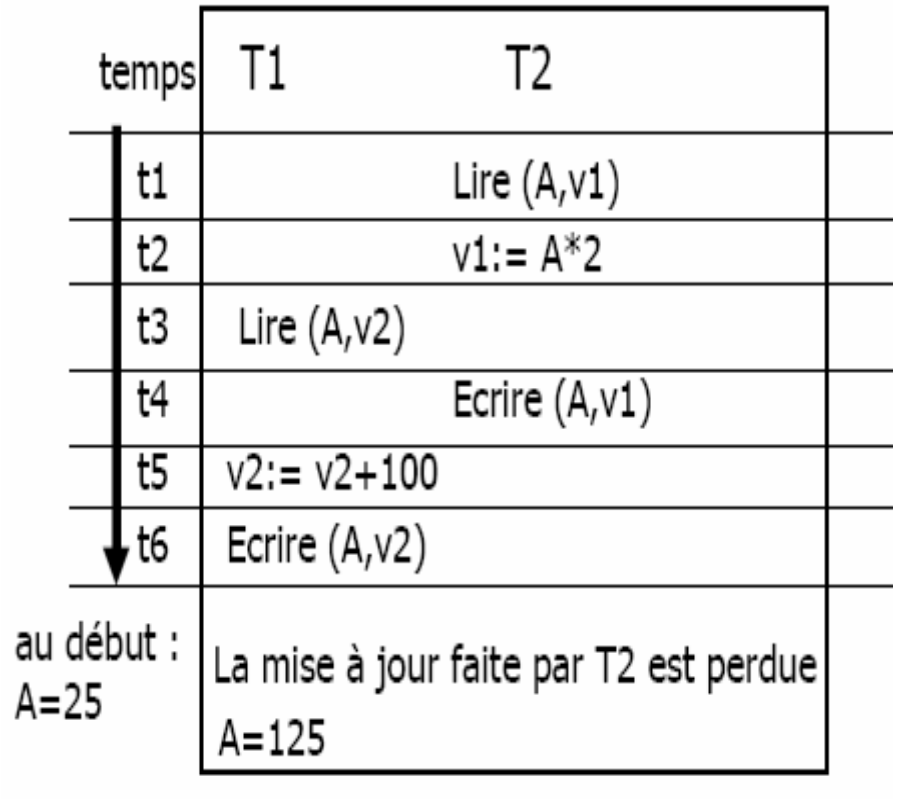
Exécution des transactions





Exécutions en série correctes

Exécution des transactions



Exécution entremêlée incorrecte

Sérialisabilité

- Sérialisabilité: Critère permettant de dire que l'exécution d'un ensemble de transactions (schedule) est correcte
- 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)
- on dit que cet ensemble de transactions est sérialisable

Exemples

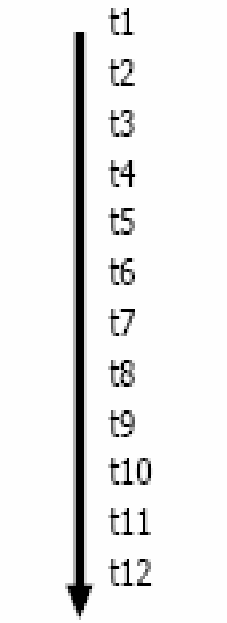
T1	T2
Lire (A,v)	Lire (A,v)
v:= A+100	v:= A*2
Ecrire (A,v)	Ecrire (A,v)
Lire (B,v)	Lire (B,v)
v:= B+100	v:= B*2
Ecrire (B,v)	Ecrire (B,v)

au début : A= B= 25

T1 puis T2 : A= B= 250

T2 puis T1 : A= B= 150

Une exécution entremêlée est correcte si elle est sérialisable



schedule 1	schedule 2
(T1,T2,T1,T2)	(T1,T2,T2,T1)
Lire (A,v)	Lire (A,v)
v:= A+100	v:= A+100
Ecrire (A,v)	Ecrire (A,v)
Lire (A,v)	Lire (A,v)
v:= A*2	v:= A*2
Ecrire (A,v)	Ecrire (A,v)
Lire (B,v)	Lire (B,v)
v:= B+100	v:= B*2
Ecrire (B,v)	Ecrire (B,v)
Lire (B,v)	Lire (B,v)
v:= B*2	v:= B+100
Ecrire (B,v)	Ecrire (B,v)
A=250, B=250	A=250, B=150
sérialisable	non sérialisable

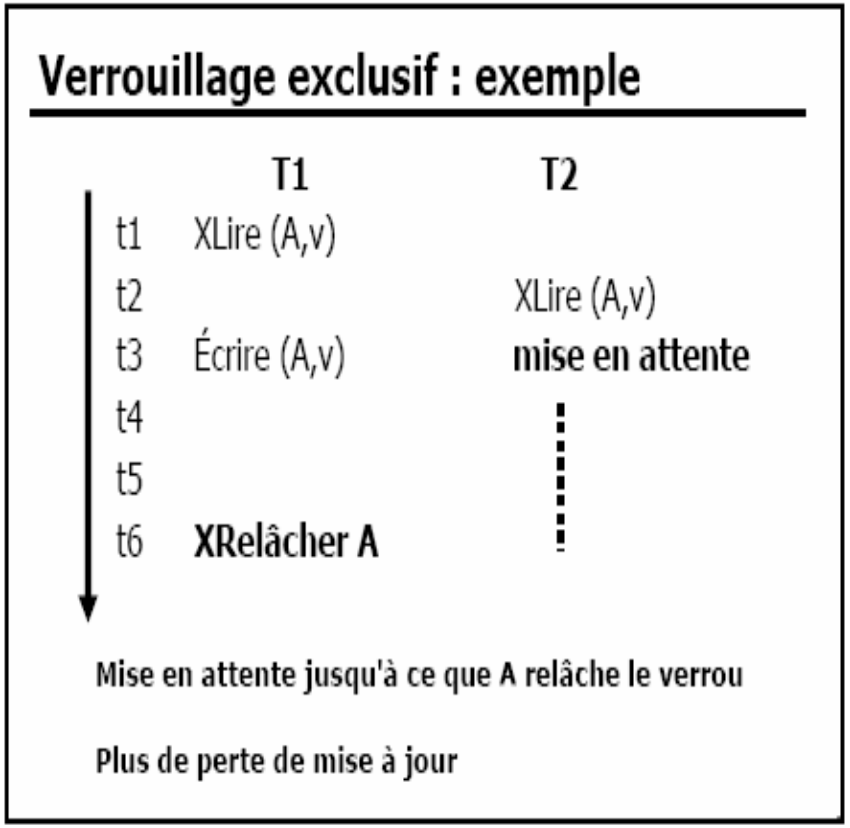
Approches de gestion de la concurrence

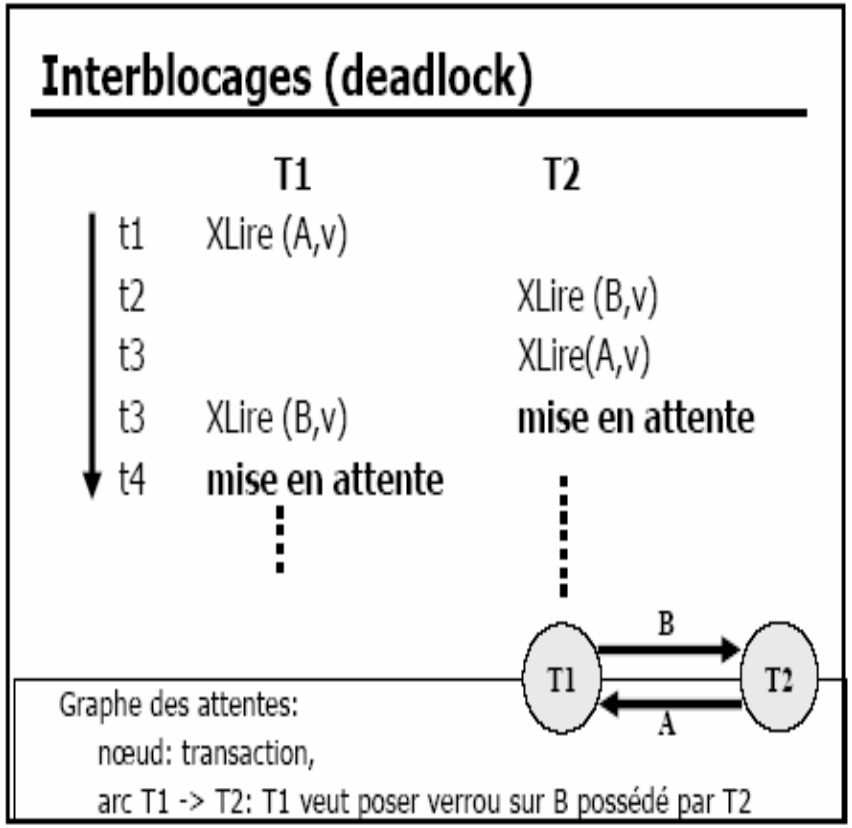
- Objectif: forcer la sérialisabilité
- Deux méthodes:
  - Verrouillage
  - Estampillage
- Verrouillage: L'idée est simple : on bloque l'accès à une donnée dès qu'elle est lue ou écrite par une transaction (« pose de verrou ») et on libère cet accès quand la transaction termine par *commit* ou *rollback* (« libération du verrou »)
  - Le verrou **partagé (SLocks)** est typiquement utilisé pour permettre à plusieurs transactions concurrentes de **lire** la même ressource.
  - Le verrou **exclusif (XLocks)** réserve la ressource en écriture à la transaction qui a posé le verrou.

Verrous exclusif : protocole PXO

- Verrous exclusifs avec libération explicite
- Toute demande d'accès contient implicitement une demande de verrouillage (Lire →XLire). Toute transaction qui a besoin d'une ressource qui est actuellement verrouillée est mise en attente (FIFO).
- L'obtention du verrou autorise lectures et mises à jour
- Un verrou peut être relâché explicitement (XRelâcher) à n'importe quel moment. Au plus tard, il est relâché implicitement à la fin de la transaction.

Verrouillage exclusif: exemple et problèmes





Solutions des Interblocages

- Deux approches:
- Détection: laisser les transactions se mettre en interblocage, le détecter puis le résoudre. Grâce au graphe des attentes: un interblocage: un cycle dans le graphe
  - Quand faire la détection ?
    - lors des demandes de verrouillage
    - périodiquement
    - après un certain temps d'attente pour une transaction
  - Action: tuer l'une des transactions
    - celle qui a fait le moins de maj
    - la plus jeune
    - celle qui a le moins de ressources allouées …
  - Tuer: défaire (annuler ses maj, ses verrous) puis relancer automatiquement plus tard
- Prévention: prévenir les interblocages.
- En pratique, solution détection la plus souvent utilisée car prévention plus coûteuse.

Prévention des interblocages

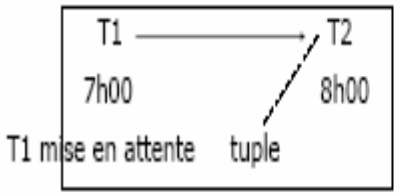
- Prévention par estampillage:
  - Estampillage de chaque transaction avec l'heure de lancement
  - Si une transaction demande un verrou sur une ressource déjà verrouillée alors résolution dépend de l'estampille

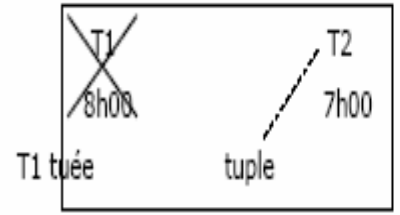
Exemple:

T2 a la ressource

T1 demande l'accès à la ressource

Wait-die (la plus jeune continue)





Si T1 est plus vieille que T2, alors T1 est mise en attente et laisse T2 finir (COMMIT ou ROLLBACK),
sinon T1 est "tuée" (ROLLBACK)

Une transaction tuée est relancée avec la même estampille qu'avant

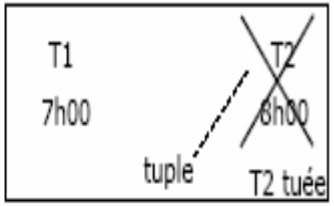
Exemple:

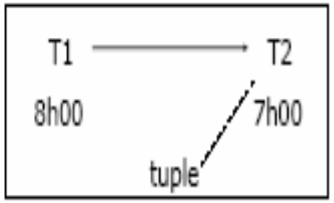
T2 a la ressource

T1 demande l'accès à la ressource

Wound-wait (la plus jeune est tuée)

Si T1 est plus vieille que T2, alors la transaction T1 ne laisse pas continuer la + jeune qui la gène. Elle tue T2 (ROLLBACK de B) et s'empare de sa ressource. Dans le cas contraire, T1 est mise en attente.



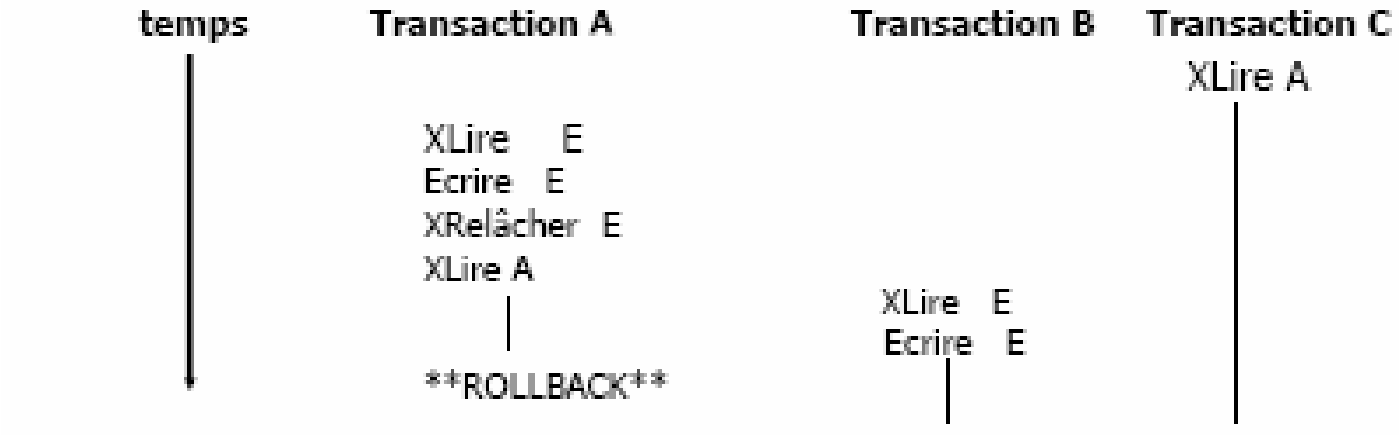


T1 mise en attente

Une transaction tuée est relancée avec la même estampille qu'avant

Libération des verrous après écriture

- Prévention et détection des interblocages →défaire des transactions → problèmes d'incohérence

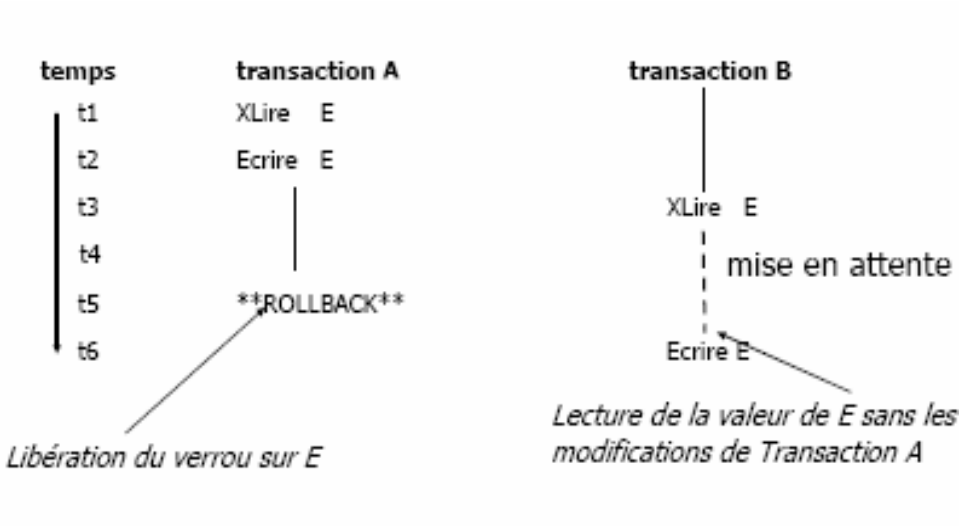


- B a lu une valeur de E qui logiquement n'a jamais existé (fantôme), Suite au "Rollback" de A, B doit être défaite, ainsi que toutes les transactions qui ont lu une valeur écrite par B (ROLLBACK en cascade)

- Solution: ne relâcher un verrou exclusif qu'en fin de transaction (COMMIT ou ROLLBACK) → Protocole PX (avec libération implicite)

Protocole PX

- Verrous exclusifs avec libération implicite
- Toute demande d'accès contient implicitement une demande de verrouillage (Lire →XLire). Toute transaction qui a besoin d'une ressource qui est actuellement verrouillée est mise en attente (FIFO).
- L'obtention du verrou autorise lectures et mises à jour
- Le verrou n'est relâché qu'en fin de transaction (plus de XRelâcher)



Verrous Partagés (S Lock)

- Les verrous exclusifs réduisent la simultanéité d'exécution
- Des transactions effectuant uniquement des lectures pourraient s'exécuter simultanément
- Il faut s'assurer que pendant les lectures les éléments lus ne sont pas modifiés → besoin d'un verrou partagé
- Plusieurs transactions peuvent obtenir un verrou partagé sur un même élément
- Aucune transaction ne peut obtenir un verrou exclusif sur un élément tant que tous les verrous partagés sur cet élément ne sont pas libérés

Verrous partagés:

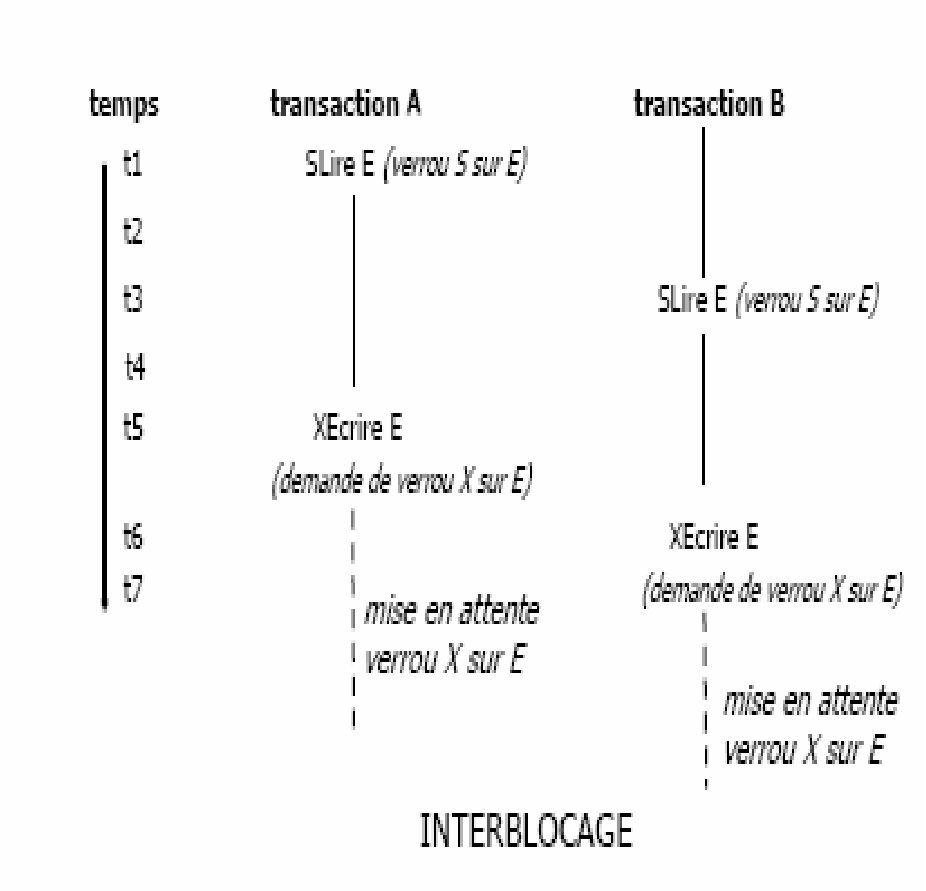
- SLire E:
  - si le verrou partagé ne peut être obtenu → mise en attente
  - sinon un verrou partagé est mis sur E et la lecture est effectuée
- SRelâcher E: libérer le verrou partagé détenu par la transaction sur E
  - plus de nécessité de libérer le verrou à la fin de transaction car pas de mise à jour

matrice de compatibilité:

PSX	S	X
S	O	N
X	N	N

Verrous partagés: Protocole

- Protocole PSX: pour mettre à jour un élément E de la base
  - demande verrou partagé sur E (SLire E)
  - Si E est déjà verrouillé en X →mise en attente
  - mise à jour → XEcrire E: demande de changement du verrou S sur E en verrou X sur E
  - comme en PX, le verrou exclusif n'est libéré qu'en fin de transaction
- Inconvénient de PSX : augmentation des interblocages
- deux transactions voulant mettre à jour le même élément vont obtenir deux verrous partagés qu'ensuite elles ne pourront transformer en verrous exclusifs (Les verrous de mise à jour remédient à ce problème, voir PUX)

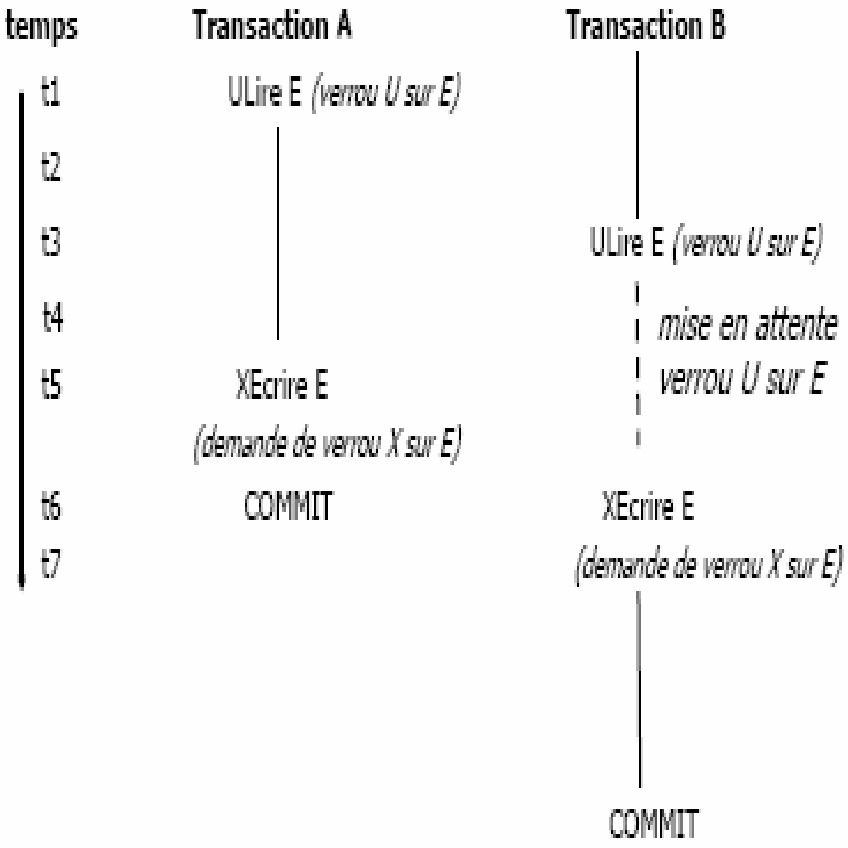


Verrous de mise à jour (U)

- Protocole PUX: Toute transaction qui veut mettre à jour un élément E de la base doit:
- demander un verrou de mise à jour sur E → ULire E
- Si E est déjà verrouillé en X ou U → mise en attente
- pour la mise à jour sur E → XEcrire E (demande de changement du verrou U en X)
- le verrou exclusif n'est libéré qu'en fin de transaction

matrice de compatibilité:

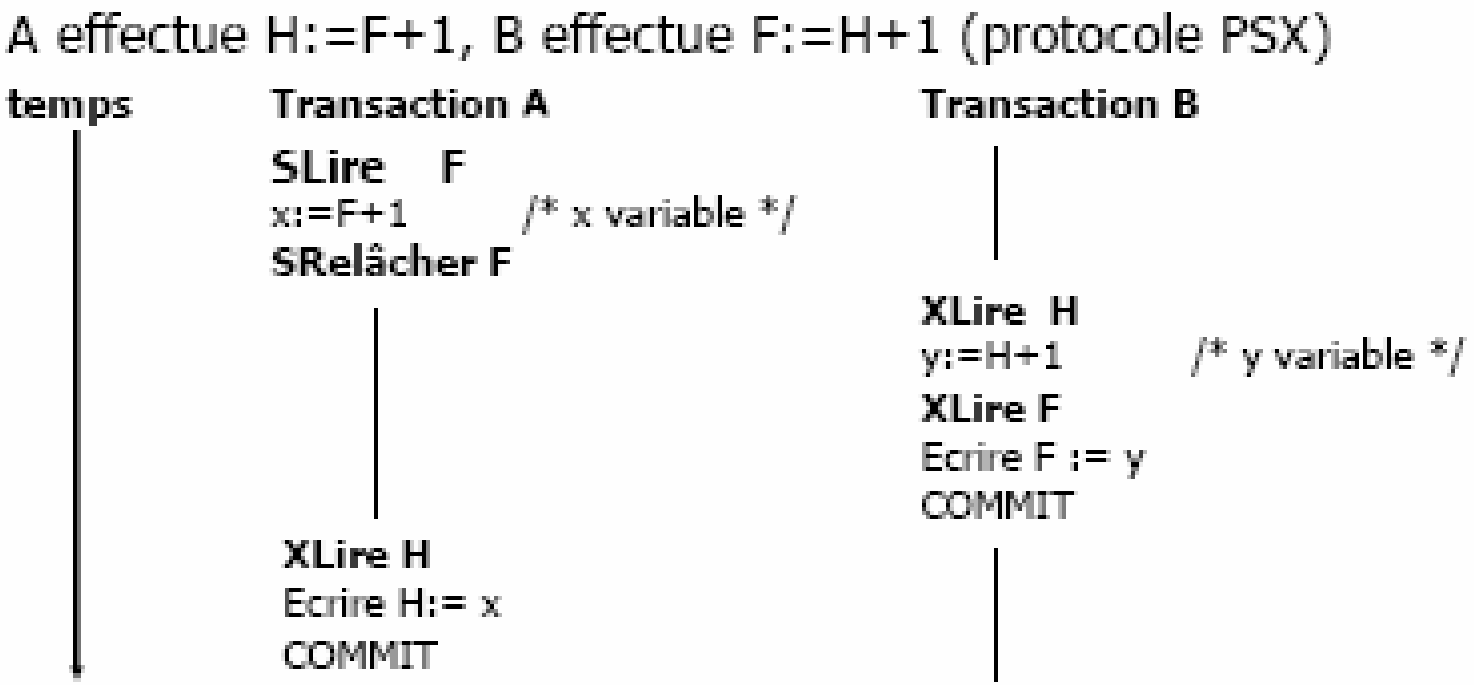
PUX	S	U	X
S	O	O	N
U	O	N	N
X	N	N	N



Verrous simples (X, S, U)

- Le protocole PUX et les verrous de mise à jour
  - réduisent les cas d'interblocage
  - réduisent aussi la concurrence
- Conclusion: les SGBD choisissent l'un des protocoles PX, PX0, PSX ou PUX, en fonction du degré de concurrence voulu et du pourcentage d'interblocages toléré

Libération des verrous après lecture (verouillage à deux phases)



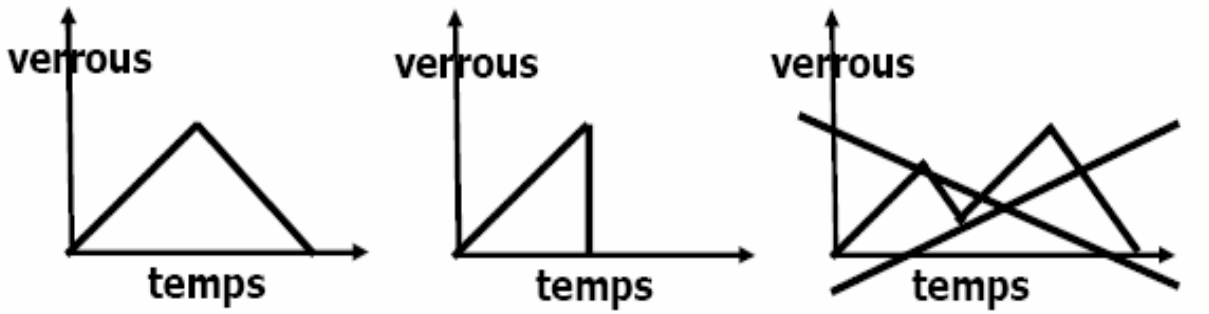
Exécution pas sérialisable: si F et H sont nuls

- A puis B : F = 2, H = 1
- B puis A: F = 1, H = 2
- exemple: F = 1, H = 1

…. le problème vient du 1er verrou

Verrouillage à deux phases

- Une transaction est à verrouillage à deux phases si :
  - avant toute opération sur un élément, la transaction demande un verrou sur cet élément pour cette opération, et
  - après avoir rendu un verrou, la transaction n'en demande plus aucun
- Durant une première phase la transaction acquiert des verrous et durant une seconde phase elle les libère
- La deuxième phase peut être réduite à un instant si tous les verrous sont relâchés à la fin de la transaction



Verrouillage à deux phases...

- Théorème (important): si toutes les transactions sont à verrouillage à deux phases, alors toutes les exécutions entremêlées de ces transactions sont sérialisables

Unité de verrouillage multiple

- Unités de verrouillage possibles: la base de données, un ensemble de relations, une relation, un tuple, une valeur → (Plus unité fine, plus concurrence forte, plus overhead est grand)
- Unité élémentaire: unité la plus fine proposée par le système (c'est souvent le tuple)
- Unité composée: tout autre unité proposée par le système (c'est souvent la relation et la base de donnée)
- L'utilisateur la possibilité de choisir l'unité de verrouillage. Mais : comment le système peut il savoir quand il a une demande de verrou sur une table (par ex) qu'il n'y a pas de verrous sur un de ses tuples?
- Pas désirable de parcourir tous les verrous de la table…

Deux types de verrous

- Deux types de verrous:
- Verrous d'intention (sur unité composée): intention de poser plus tard des verrous effectifs (S, X) sur les unités composantes
- Verrous effectifs (S, X vus précédemment). S et X peuvent être posés sur toute unité composée ou élémentaire
- Pour placer un verrou sur un élément, on doit partir de l'élément le plus haut (table/tuple/valeur).
- Si cet élément est celui que l'on souhaite verrouiller, on pose un verrou S ou X sur cet élément.
- Si l'élément que l'on souhaite verrouiller est plus bas alors poser un verrou d'intention sur l'élément qui le contient puis un verrou S ou X sur l'élément à verrouiller.

Verrous d’intention

- **Verrous d’intention partagée** (IS) : posé sur une unité composée C → la transaction va demander des verrous partagés (S) sur certains éléments composants de C
- **Verrou d’intention exclusive** (IX) : la transaction va demander un ou des verrous exclusifs (X) sur certains éléments composants de C, pour les mettre à jour
- **verrou partagé avec intention exclusive (**SIX): un verrou SIX est un verrou S posé sur une unité composée C et un verrou IX posé sur C. Il signifie que la transaction permet que d'autres lisent les éléments de C, mais qu'elle n'autorise aucune mise à jour, et de plus qu'elle annonce son intention de demander des verrous exclusifs sur des éléments de C pour les mettre à jour.

Protocole: PI

- Protocole d'emploi
- obtention d'un verrou X sur une unité:
  - obtention implicite d'un verrou X sur tous les composants de l'unité
- obtention d'un verrou S sur une unité
  - obtention implicite d'un verrou S sur tous les composants de l'unité
- pour demander un verrou S ou IS sur l'unité E
  - il faut avoir acquis un verrou IS (au moins) sur l'unité contenant E
- pour demander un verrou X, IX sur l'unité E
  - il faut avoir acquis un verrou IX (au moins) sur l'unité composée contenant E
- avant de libérer un verrou sur une unité E
  - avoir auparavant libéré tous les verrous sur des composants de E
- Si plus de deux unités de verrouillage, les règles s'appliquent récursivement

La matrice de compatibilité des verrous pour une unité composée

	X	S	IS	SIX	IX	aucun
X	N	N	N	N	N	O
S	N	O	O	N	N	O
IS	N	O	O	O	O	O
SIX	N	N	O	N	N	O
IX	N	N	O	N	O	O
aucun	O	O	O	O	O	O

La matrice de comptabilité des verrous pour une unité élémentaire reste inchangée

Techniques d'estampillage

- SGBD centralisé → concurrence avec verrous
- Verrous: explicitement demandés ou acquis automatiquement lors des instructions de L/E
- Bases de données réparties → peu compatibles avec un gestionnaire de verrous centralisé → technique d'estampillage
- Transactions estampillées avec l'heure de lancement
- Verrous: assurent que l'exécution simultanée est équivalente à une exécution séquentielle quelconque
- Estampillage: assure que l'exécution simultanée est équivalente à l'exécution séquentielle correspondant à l'ordre chronologique des estampilles des transactions

Estampillage: Règles

- Toutes les transactions s'exécutent simultanément → Conflits: Une transaction demande à lire un élément déjà mis à jour par une transaction plus récente, Une transaction demande à mettre à jour un élément déjà lu ou mis à jour par une transaction plus récente

- Ces demandes doivent être rejetées: la transaction trop vieille est tuée et relancée avec une nouvelle estampille

Estampillage: résolution de conflits

- A chaque élément de la base sont associées deux valeurs:
  - LMAX: estampille de la transaction la plus jeune qui a lu cet élément (sans avoir été tuée lors de cette lecture)
  - EMAX: estampille de la transaction la plus jeune qui a écrit avec succès l'élément (sans avoir été tuée lors du COMMIT)
- Si la transaction T demande à lire E (Lire E)
  - si estampille(T) ≥ EMAX(E) /* estampille (T) plus jeune que EMAX(E)
  - alors ***/*OK*/*** LMAX(E):=Max(LMAX(E), estampille(T))
  - sinon ***/*conflit*/*** Tuer T et relancer T avec une nouvelle estampille
- Si la transaction T demande à mettre à jour E (Ecrire E)
  - si estampille(T) ≥ LMAX(E) et estampille(T) ≥ EMAX(E)
  - alors ***/*OK*/*** EMAX(E):=estampille(T)
  - sinon ***/*conflit*/*** Tuer T et relancer T avec une nouvelle estampille