Bases de données

Architecture et Systèmes

Cours 2 : Transactions

Luc Bouganim

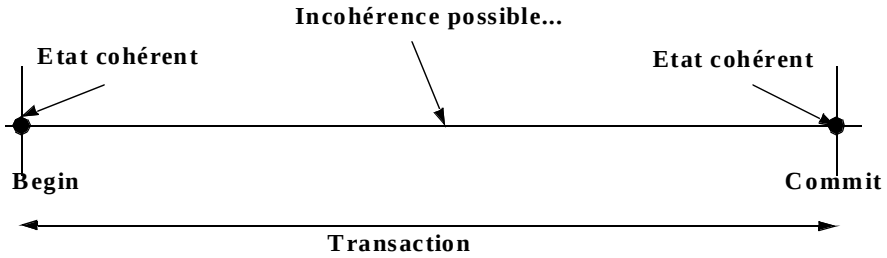
PRiSM Versailles - INRIA, Rocquencourt

Gestion de transactions

- 2 problèmes :
  - De multiples utilisateurs doivent pouvoir accéder à la base de donnée en même temps
  - De nombreuses et diverses pannes peuvent apparaître. Il ne faut pourtant pas perdre les données...
- La Gestion de Transactions répond à ces problèmes par :
  - Le concept de Transaction
  - 4 propriétés toujours respectées
  - Des mécanismes

Notion de transaction

- Unité d'échange entre le programme d'application (ou l'usager) et le SGBD.
- Une transaction se compose d'une séquence d'opérations élémentaires dont l'exécution provoque le passage d'un état cohérent de la BD à un nouvel état cohérent



Propriétés

- Une Transaction est composée d'opérations primitives encadrée par des marqueurs begin et end.

```
begin(Ti)
    ...
    read(x)
    read(y)
    write(z ,x + y)
    ...
end( Ti )
```

- Les Transactions respectent toujours les 4 propriétés suivantes :
  - Atomicité
  - Cohérence
  - Isolation
  - Durabilité

ATOMICITE ET COHERENCE

Atomicité :

Une Transaction s'effectue soit **ENTIÈREMENT** ,

BEGIN	COMMIT
Etat Cohérent Y	Etat Cohérent Y+1

soit **PAS DU TOUT** .

BEGIN	ABORT
Etat Cohérent Y	Etat Cohérent Y

Cohérence :

Les effets d'une Transaction commise ou annulée ne doivent jamais violer les contraintes d'intégrité de la BD.

ISOLATION ET DURABILITE

Isolation :

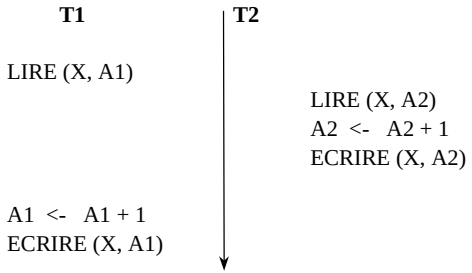
Une transaction ne doit jamais voir les résultats intermédiaires des autres transactions.
Les transactions s'exécutent comme si elles étaient seules dans la BD.
Entrelacement possible d'opérations entre plusieurs transactions ssi I est respectée.
C'est le rôle du contrôle de concurrence.

Durabilité :

Une fois une transaction commise, le système assure que ses effets ne seront jamais perdus.

Accès concurrent: les problèmes (1)

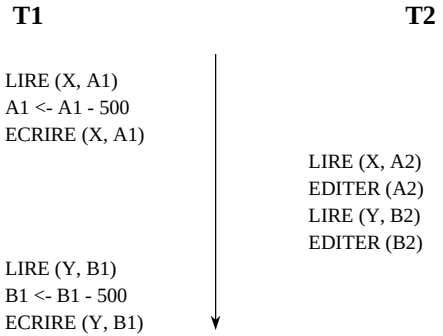
Perte d'opération



Accès concurrent: les problèmes (2)

Observation d'incohérences

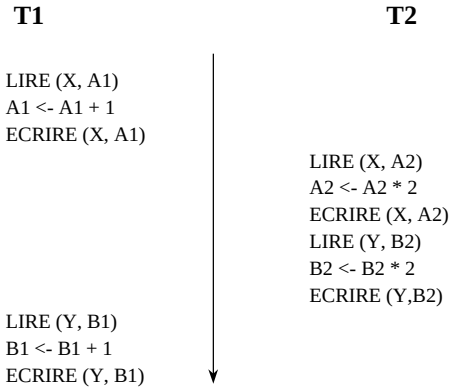
Contrainte d'Intégrité : X = Y



Accès concurrent: les problèmes (3)

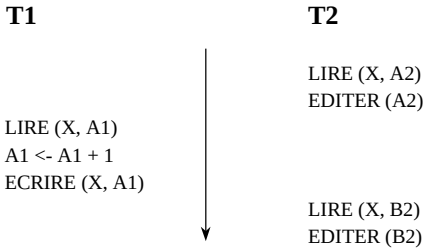
Introduction d'incohérences

Contrainte d'Intégrité : X = Y



Accès concurrent: les problèmes (4)

Lectures non reproductibles

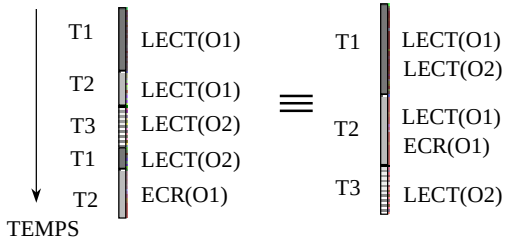


Exécution sérialisable

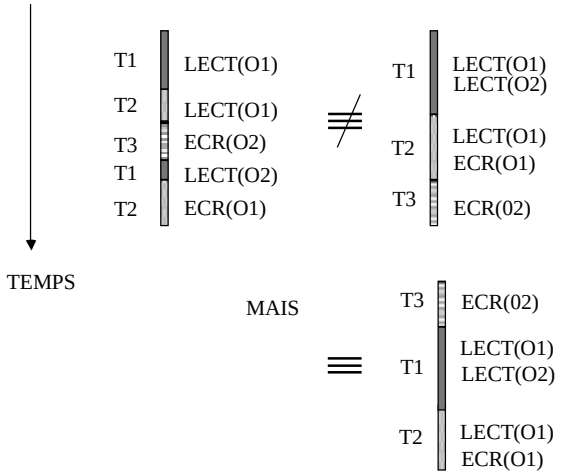
- **Exécution sérialisable :** une exécution en parallèle des transactions T1, ..., Tn est dite sérialisable si son résultat est équivalent à une exécution en série de ces mêmes transactions.
  - Les actions A et B sont commutables si toute exécution de A suivie par B donne le même résultat que l'exécution de B suivie par A.
  - Si une exécution de transactions est transformable par commutations successives en une exécution en série, alors elle est sérialisable.

Exemple

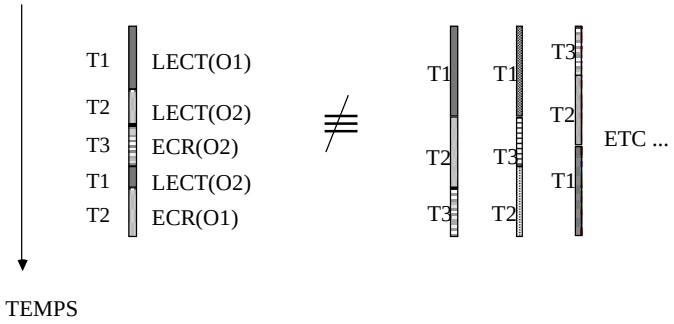
d'exécution sérialisable



Autre exemple d'exécution sérialisable

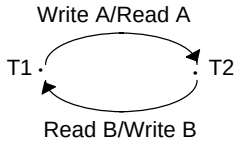


Exemple d'exécution non sérialisable



Relation de précédence

- **Ti précède Tj dans une exécution s' il existe deux actions non permutables ai et aj tel que ai soit exécutée par Ti avant que aj soit exécutée par Tj**
- **Exemple :**



- **Si le graphe de précédence d'une exécution est sans circuit, l'exécution est sérialisable.**

VERROUILLAGE

Objectif : éviter de générer des exécutions incorrectes en faisant attendre une transaction qui veut exécuter des opérations conflictuelles avec une autre transaction sur un même granule.

Protocole de verrouillage : deux actions utilisées par chaque transaction

LOCK(g,M) : avant d'exécuter une action
UNLOCK(g) : après exécution complète de la transaction

Les verrous sont en lecture (shared ou S) ou en écriture (exclusive ou X)

Les verrous en lecture et en écriture sont incompatibles

	S	X
S	oui	non
X	non	non

VERROUILLAGE DEUX PHASES

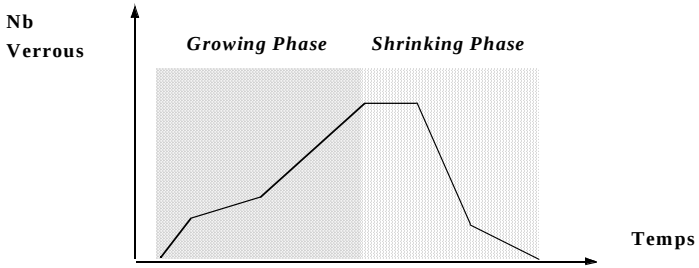
Avant de lire ou d'écrire, une transaction demande un verrou.

Les verrous respectent les règles de comptabilité.

Si les verrous sont incompatibles, la transaction attend.

Après avoir obtenu tous ses verrous une transaction déverrouille les objets dont elle n'a plus besoin.

Dès qu'un verrou à été libéré, aucun autre verrou ne peut être posé



VERROU MORTEL

Deux transactions peuvent se bloquer mutuellement

Prévention :

une transaction n'attend jamais une plus jeune

--> une des deux transactions est tuée

Détection :

on construit le graphe des attentes au fur à mesure et on détecte les circuits

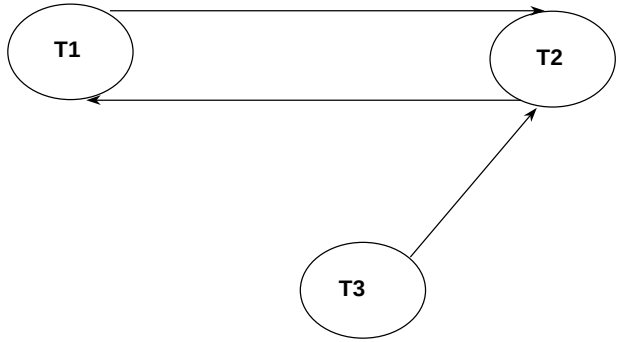
--> une transaction est tuée

Timeout :

La transaction attend trop longtemps son verrou

--> la transaction est tuée

Problème du Verrou Mortel



Performance et Contrôle de concurrence

Le strict two phase locking assure la serialisabilité mais pose de gros problèmes de performances

Problème des mises à jour

Niveau d'isolation

Granule de verrouillage (Performance et pb du fantome)

Cohabitation du décisionnel et de l'OLTP

Degrés d'isolation SQL (1)

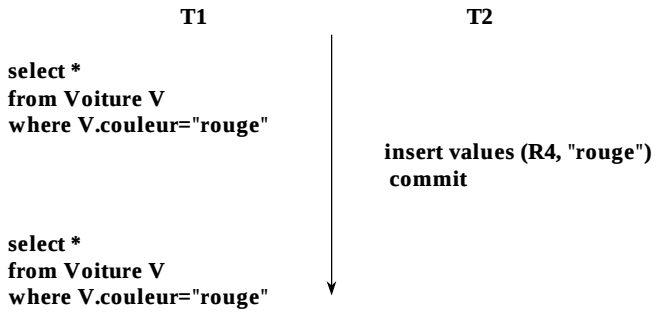
- **Objectif: accroître le parallélisme en autorisant certaines transactions à violer la règle d'isolation**
- **Standardisés par SQL2 (Set Transaction Isolation Level)**
- **Degré 0 (Read Uncommitted)**
  - **Une transaction de degré 0 est sans perte de mise à jour**
  - **Pas de pose de verrous lors des lectures**
  - **Interdiction de faire des écritures**
- **Degré 1 (Read Committed)**
  - **Une transaction de degré 1 satisfait le degré 0 et ne fait pas de lecture sale**
  - **Pose de verrous courts partagés (S) lors des lectures**
  - **Pose de verrous longs exclusifs (X) lors des écritures**

Degrés d'isolation SQL (2)

- **Degré 2 (Repeatable Read)**
  - **Une transaction de degré 2 satisfait le degré 1 et ne fait pas de lecture non reproductible**
  - **Pose des verrous longs partagés (S) lors des lectures**
- **Degré 3 (Serializable)**
  - **Une transaction de degré 3 satisfait le degré 2 et ne fait pas de requête non reproductible**
  - **Elimine les fantômes**

Le problème des fantômes

- **Hypothèse :**
- **Toute donnée accédée est verrouillée et les verrous ne sont relâchés qu'en fin de transaction**
- **Pourtant, l'apparition de fantômes peut entraîner des requêtes non reproductibles, donc une exécution non sérialisable**



Verrous hiérarchiques

Idée: On accède à tout objet par un chemin hiérarchique, par exemple

Base de données
Relation
Segment
Page

La transaction demande des verrous d'INTENTIONS le long du chemin parcouru et un verrou CLASSIQUE sur le dernier objet modifié.

T1 pose un verrou IX sur R
T2 pose un verrou S sur R

Matrice de compatibilité

- **Matrice de compatibilité**

	IS	IX	S	SIX	X
IS	1	1	1	1	0
IX	1	1	0	0	0
S	1	0	1	0	0
SIX	1	0	0	0	0
X	0	0	0	0	0

IS : J'ai l'intention de lire un objet de plus bas niveau
IX : J'ai l'intention d'écrire un objet de plus bas niveau
S : Je vais lire cet objet
SIX : Je lit cet objet avec l'intention d'écrire un objet de plus bas niveau
X : Je vais modifier cet objet

Cohabitation décisionnel OLTP

- **Requêtes décisionnelles**
  - **Beaucoup de lectures**
  - **Peu ou pas d'écritures**
  - **Transaction longues**
  - **Peu de transactions**
- **Requêtes transactionnelles**
  - **Peu de lectures**
  - **Peu d'écritures**
  - **Transaction courtes**
  - **Beaucoup de transactions**
- **Les requêtes décisionnelles vont bloquer les requêtes transactionnelles ou le contraire !!!**

Protocoles Multiversions

- **Les transactions en lecture seul accèdent à une version de la base correspondant à ce qu'elles voyaient en début d'exécution**
  - **Permet d'exécuter les transactions Read Only sans prendre de verrous**
  - **Consomme de la mémoire (ou génère des I/O)**
  - **Approche proposée dans de nombreux produits**
  - **Ne résoud pas les conflits de transactions longues avec écriture (même peu)**
  - **Utilisation d'estampille pour marquer les versions (timestamp)**

Tolerance aux pannes

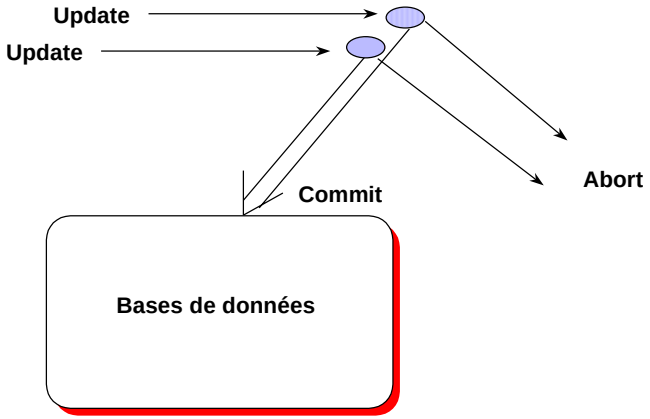
- **Motivations :**
  - **Environnements non fiables**
  - **Pannes matérielles**
    - » **Crash Disk,**
    - » **Panne de Courant, ...**
  - **Pannes logicielles**
    - » **Erreurs dues aux contrôle de concurrence,**
    - » **Violations de contraintes d'intégrité, Crtl C, ...**
- **Objectifs :**
  - **Assurer l'atomicité des transactions**
  - **Garantir la durabilité des effets des transactions commises**
- **Moyens :**
  - **Journalisation**
  - **Mécanismes de reprise après panne**

Commit et Abort

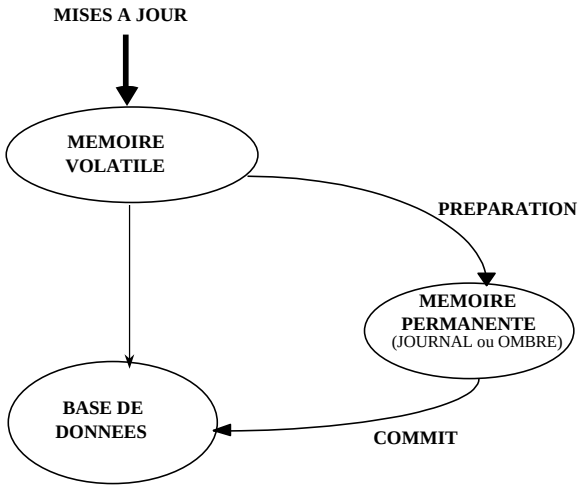
- **NECESSITE D'INTRODUIRE DES ACTIONS ATOMIQUES**
  - **COMMIT (FIN AVEC SUCCES)**
  - **ABORT (FIN AVEC ECHEC)**
- **CES ACTIONS S'EFFECTUENT EN FIN DE TRANSACTION**
- **COMMIT :**
  - **REND EFFECTIVES TOUTES LES MISES A JOUR DE LA TRANSACTION**
- **ABORT :**
  - **ANNULE TOUTES LES MISES A JOUR DE LA TRANSACTION**

Effet Logique

- Mémoire de la transaction



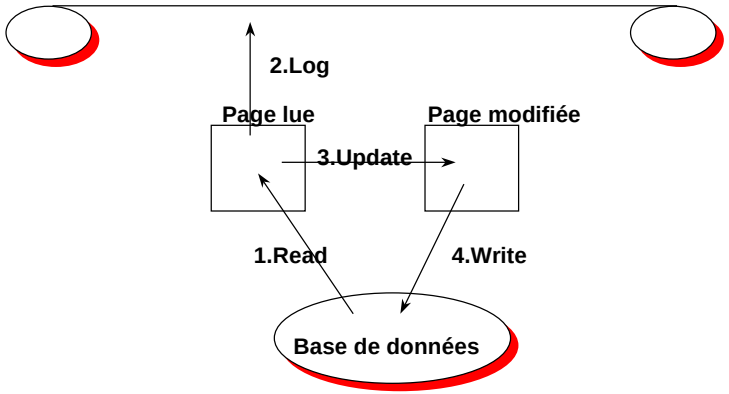
Réalisation Physique



- Deux outils:
  - Journal d'Images Avant
    Afin de pouvoir défaire une mise à jour
  - Journal d'Images Après
    Afin de pouvoir refaire une mise à jour

Journal des Images Avant

■ Utilisé pour défaire les mises à jour

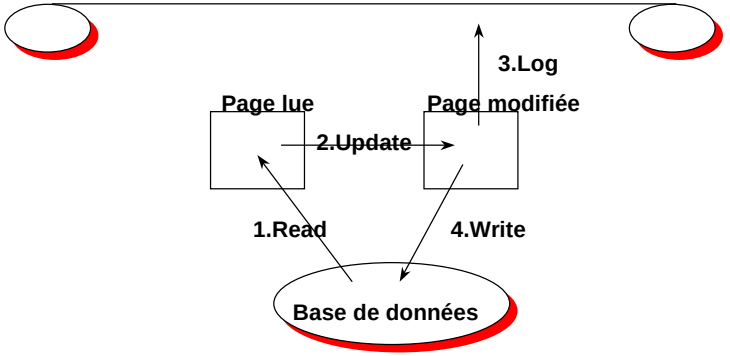


Règle d'or: Write Ahead Logging (WAL)

Toute mise à jour doit être précédée d'une écriture dans le journal des images avant

Journal des Images Après

■ Utilisé pour refaire les mises à jour

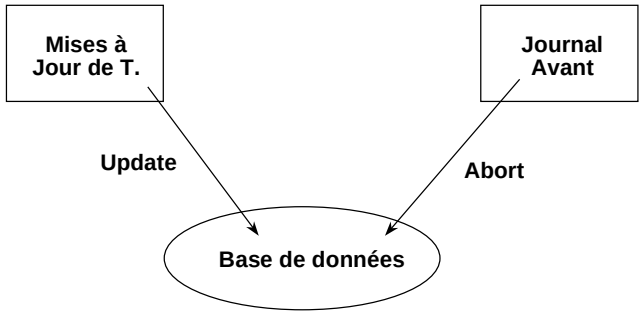


Règle d'or: Force Log at Commit

Toute validation de transaction doit être précédée de l'écriture de son journal d'images après sur un disque différent de celui contenant la base de données

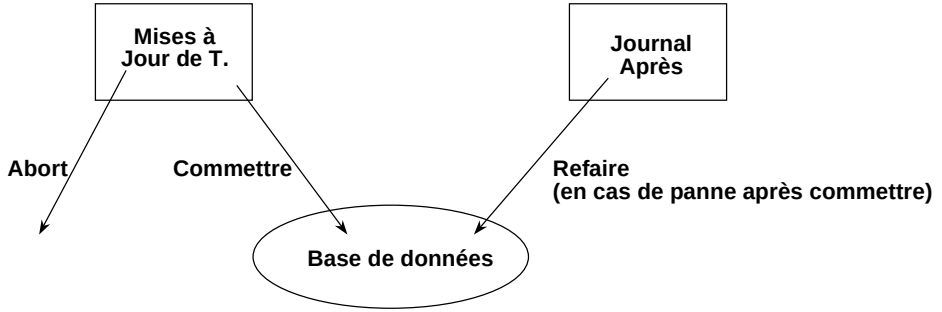
Défaire sans Refaire

- Les mises à jour sont effectuées en place dans la base avant de commettre la transaction
- En cas de panne, on applique les images avant pour annuler les mises à jour déjà effectuées



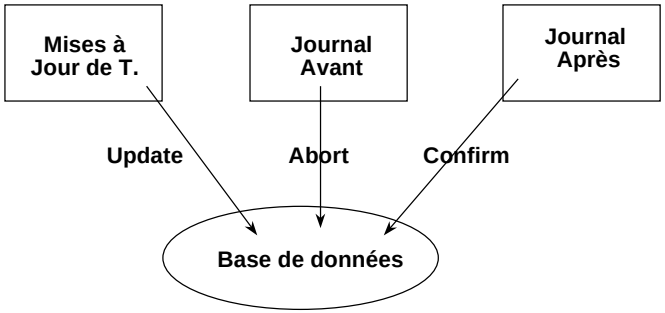
Refaire sans Défaire

- Les mises à jour sont effectuées en mémoire et enregistrées dans le journal des images après puis dans la base après le commit d'une transaction
- En cas de panne, on applique les images après pour refaire les mises à jour



Défaire et Refaire

- Les mises à jour sont effectuées en place dans la base quand le gestionnaire de cache le décide
- En cas de panne, on applique les images avant pour annuler les mises à jour des transactions non commises et celui des images après pour refaire les transactions commises



Journaux Physiques/Logiques

- **Journal physique:**
  - On enregistre la valeur avant et après modification
  - la structure peut être : IdTrans, NumPageDisk, Adresse, Valeur
  - Pour défaire, on écrase la valeur actuelle avec les valeurs avant
  - Pour refaire, on écrase la valeur actuelle avec les valeurs aprés
- **Journal Logique**
  - On enregistre l'action logique qui a entraîné la modification
  - La structure peut être : IdTrans, IdObjet, IdActions, Arguments
  - Exemple: Ajouter 500 au tuple xxx
  - pour refaire, on ré-applique l'action logique
  - et pour défaire ?

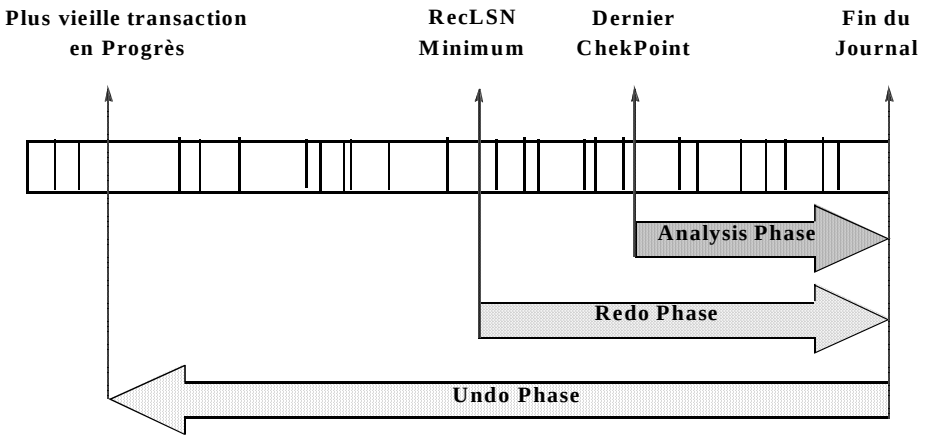
Utilisation des journaux

- **Journaux : Qu'y trouve t-on ?**
  - **Identifiant de la transaction**
  - **Identifiant de l'enregistrement de la BD**
  - **Valeur avant**
  - **Valeur après**
  - **Encadrés par des marqueurs**
- **Défaire:**
  - **Lecture de la fin vers le début du journal des images avant. On défait les transactions non commises.**
- **Refaire**
  - **Lecture du début vers la fin du journal des images après. On refait toutes les transactions validées.**

Optimisations

- **Écriture anticipée du journal**
  - **Contexte : transactions avec beaucoup de modifications (ou beaucoup de transactions)**
  - **Principe : un buffer stocke les enregistrement destinés au journaux, dés que ce buffer est plein, il est écrit sur disque**
  - **But : permet de diminuer le temps de commit**
- **Group commit**
  - **Contexte : Pour une charge de travail avec beaucoup de transactions courtes (avec peu de modifications)**
  - **Principe : Attendre quelques dixièmes de secondes lors du commit pour effectuer l'écriture en une I/O des enregistrements de journaux pour plusieurs transactions**
  - **But : Augmenter le débit transactionnel**

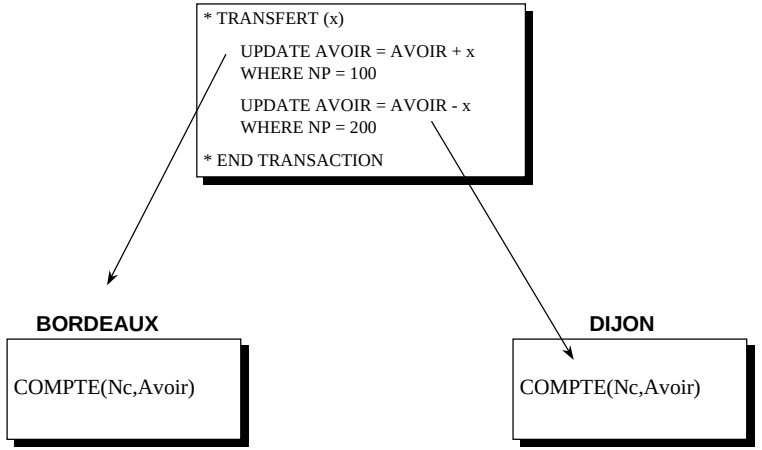
Journalisation centralisée : ARIES



Journalisation répartie

- **ATOMICITE :**

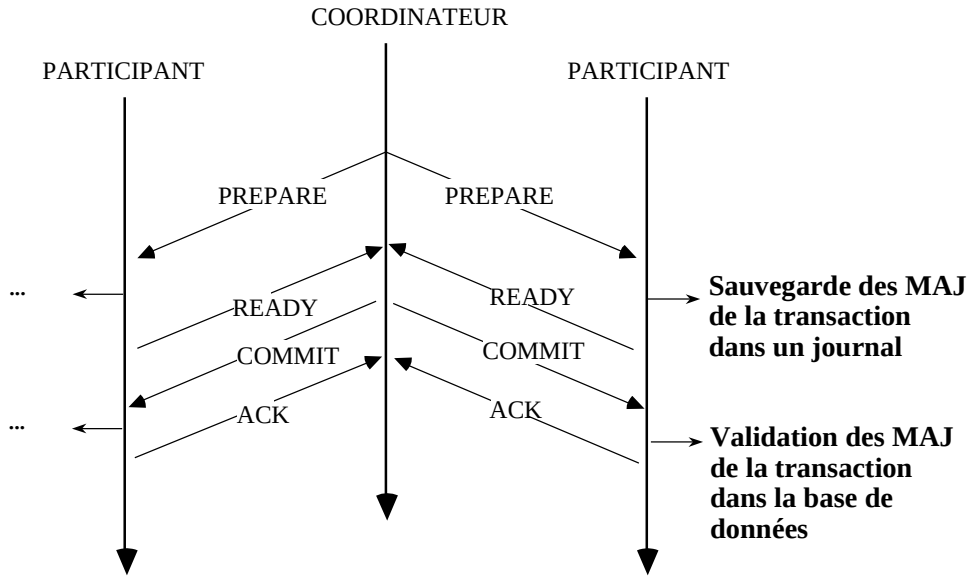
  Garantir qu'un ensemble de mises à jour sur plusieurs sites est totalement exécuté ou pas du tout



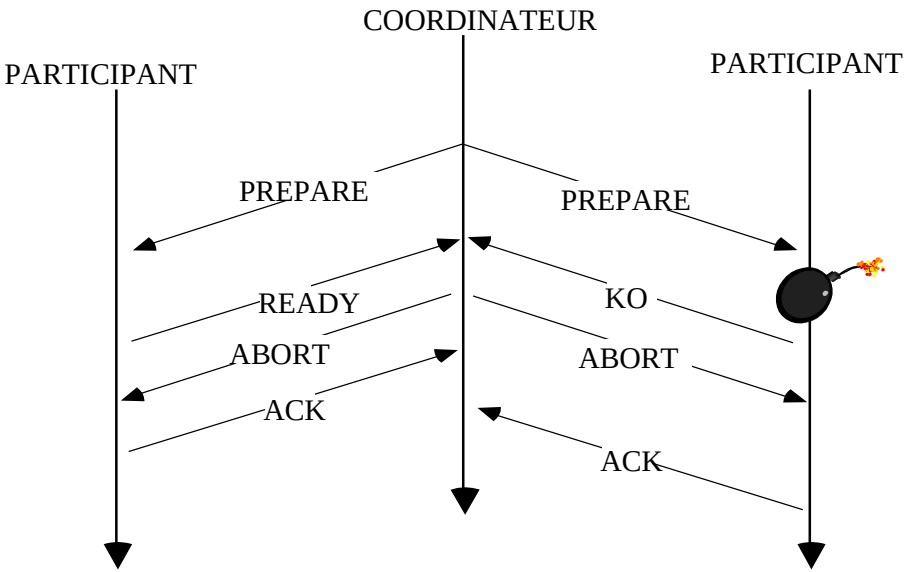
Validation à 2 Phases (2PC)

- **Principe : Diviser la commande de validation en deux phases**
  - **Phase 1 : Phase de vote pour savoir si tous les sites sont capables de valider ou non la transaction**
  - **Phase 2 : Phase de décision. La transaction n'est validée que si TOUS les sites votent OK lors de la phase 1.**
- **Coordinateur :**
  - **Le composant système du site qui centralise et pilote le protocole**
- **Participant :**
  - **Le composant système d'un site qui participe à l'exécution de la transaction**
- **Ce protocole doit être résistant aux pannes**

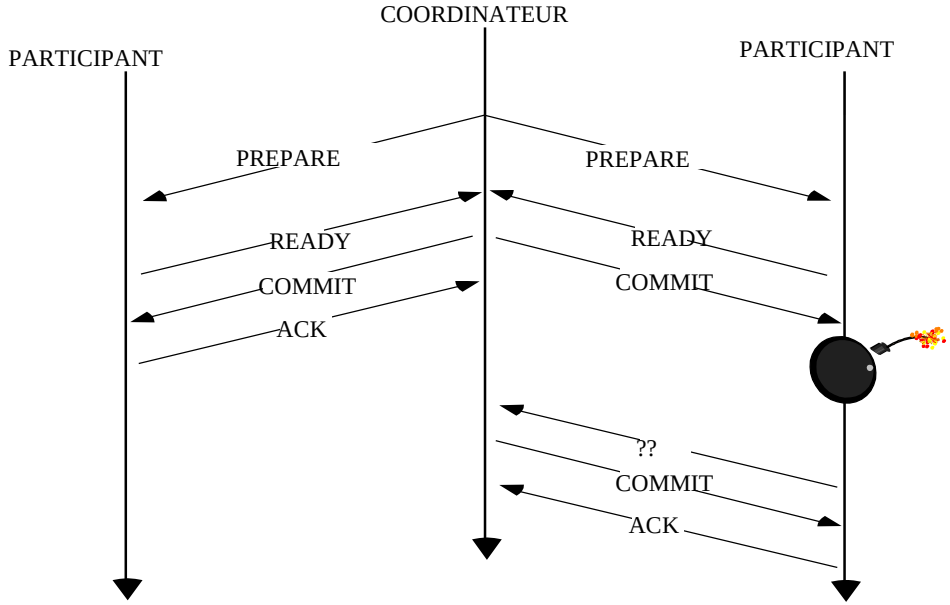
Cas Favorable



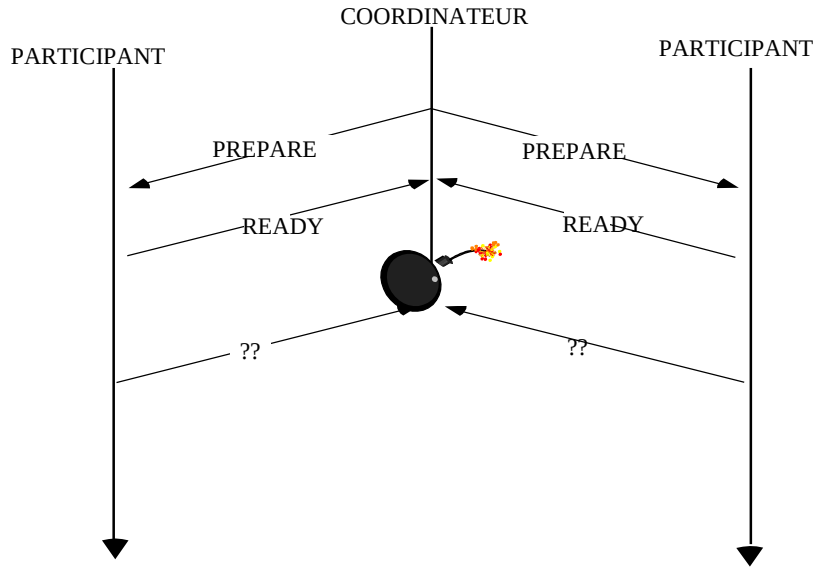
Cas défavorable (1)



Cas défavorable (2)



Cas très défavorable (3)



Phase d'incertitude pour le participant:
entre l'envoi du Ready et l'attente de la réponse

Le modèle X/OPEN DTP

- **Modèle de référence pour Distributed Transaction Processing [1993]**
  - **Objectif: étendre une sphère transactionnelle à tout type de serveur (SGBD ou non) et assurer la portabilité de tous les composants d'un environnement transactionnel**