

GESTION DE TRANSACTIONS

1. Objectifs et bases
2. Journaux et reprise
3. Scénarios de reprise
4. Modèles étendus
5. Cas des systèmes répartis

1. Le transactionnel (OLTP)

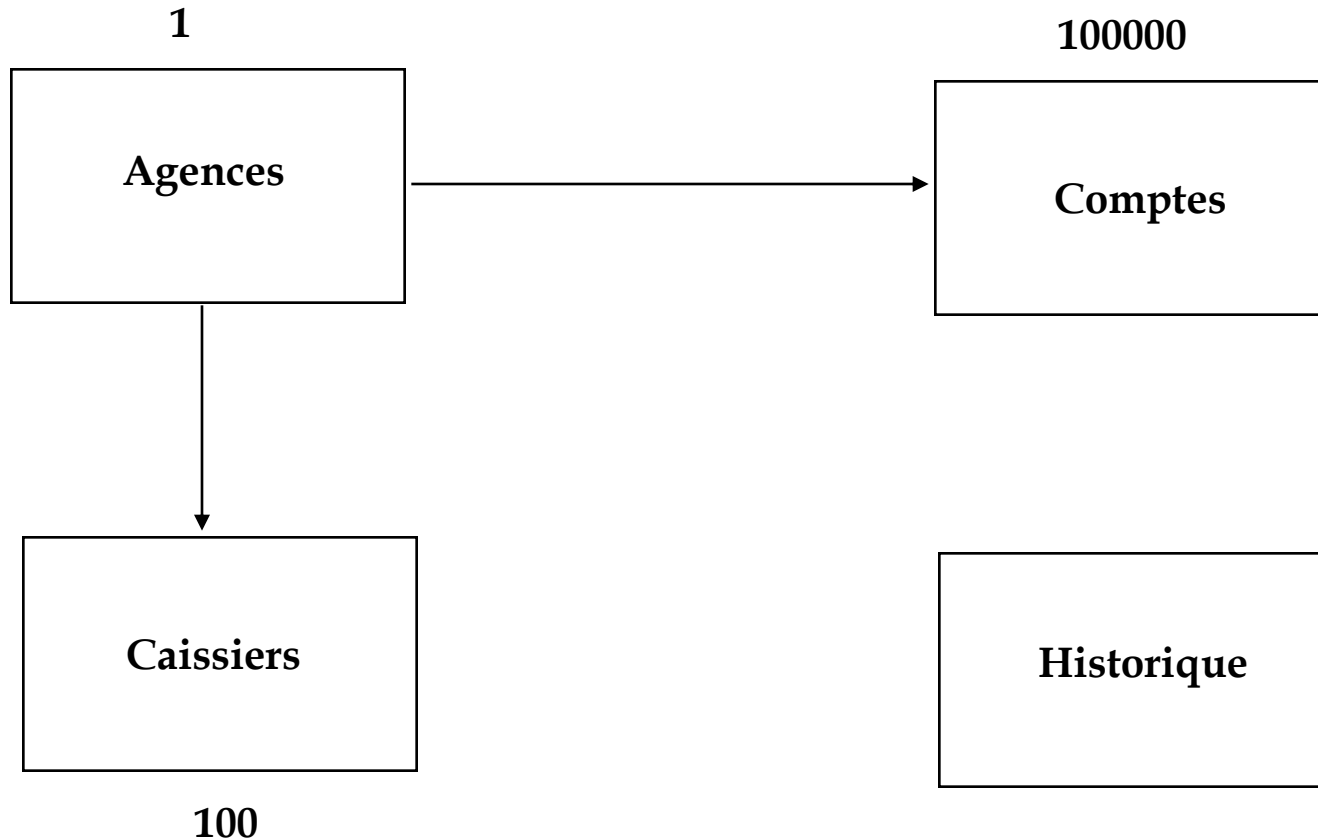
- **Opérations typiques**

- mises à jour ponctuelles de lignes par des écrans prédéfinis, souvent répétitives, sur les données les plus récentes

- **Exemple**

- Benchmark TPC-A et TPC-B : débit / crédit sur une base de données bancaire
- TPC-A transactionnel et TPC-B avec traitement par lot
- Mesure le nombre de transactions par seconde (tps) et le coût par tps

La base TPC-A/B



Taille pour 10 terminaux, avec règle d'échelle (scaling rule)

La transaction Débit - Crédit

- Begin-Transaction
 - Update Account Set Balance =
Balance + Delta
Where AccountId = Aid ;
 - Insert into History (Aid, Tid,
Bid, Delta, TimeStamp)
 - Update Teller Set Balance =
Balance + Delta
Where TellerId = Tid ;
 - Update Branch Set Balance =
Balance + Delta
Where TellerId = Tid ;
- 90 % doivent avoir un temps de réponse < 2 secondes
- Chaque terminal génère une transaction toute les 10s
- Performance = Nb transactions commises / Elapse time
- End-Transaction.

Cohabitation avec le décisionnel

- Les transactions doivent souvent cohabiter avec des requêtes décisionnelles, traitant un grand nombre de tuples en lecture
- Exemple :
 - Moyenne des avoir des comptes par agence
 - **SELECT** B.BranchId, AVG(C.Balance)
FROM Branch B, Account C
WHERE B.BrachId = C.BranchId
GROUP BY B.BranchId ;

Les menaces

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

Propriétés des transactions

- **Atomicité**
 - Unité de cohérence : toutes les mises à jour doivent être effectuées ou aucune.
- **Cohérence**
 - La transaction doit faire passer la base de donnée d'un état cohérent à un autre.
- **Isolation**
 - Les résultats d'une transaction ne sont visibles aux autres transactions qu'une fois la transaction validée.
- **Durabilité**
 - Les modifications d'une transaction validée ne seront jamais perdues

Commit et Abort

- INTRODUCTION D' ACTIONS ATOMIQUES
 - Commit (fin avec succes) et Abort (fin avec echec)
 - Ces actions s'effectuent en fin de transaction
- COMMIT
 - Validation de la transaction
 - Rend effectives toutes les mises à jour de la transaction
- ABORT
 - Annulation de la transaction
 - Défait toutes les mises à jour de la transaction

Schéma de transaction simple

- Fin avec succès ou échec

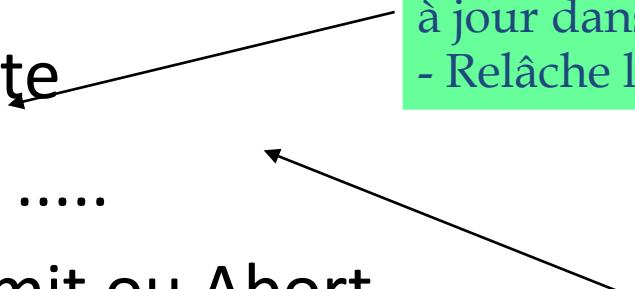
- Begin_Transaction

- update

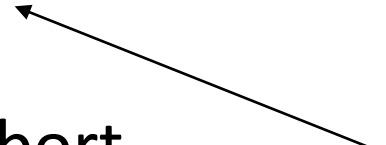
- update

-

- Commit ou Abort

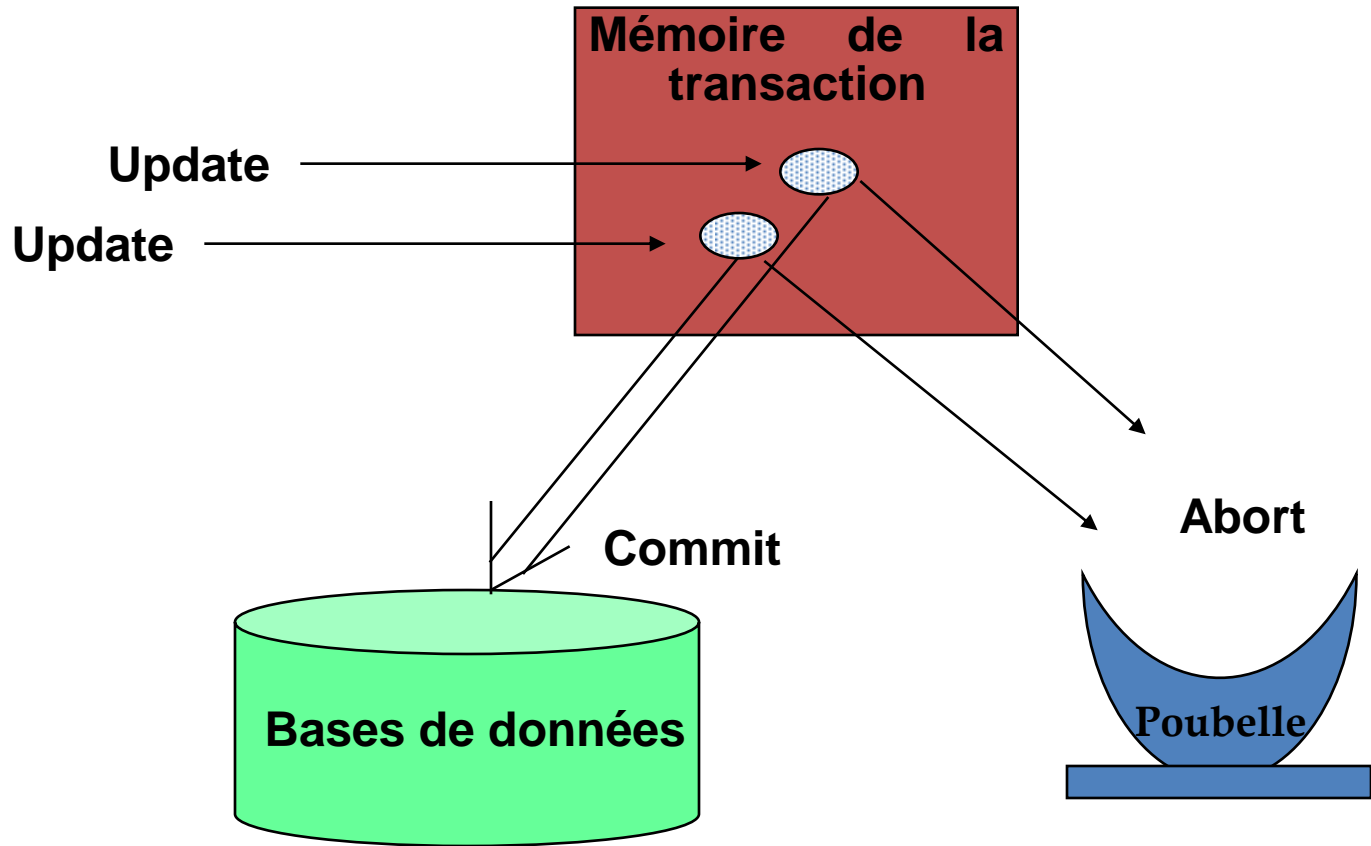


- Provoque l'intégration réelle des mises à jour dans la base
- Relâche les verrous



- Provoque l'annulation des mises à jour
- Relâche les verrous
- Reprend la transaction

Effet logique



Interface applicative

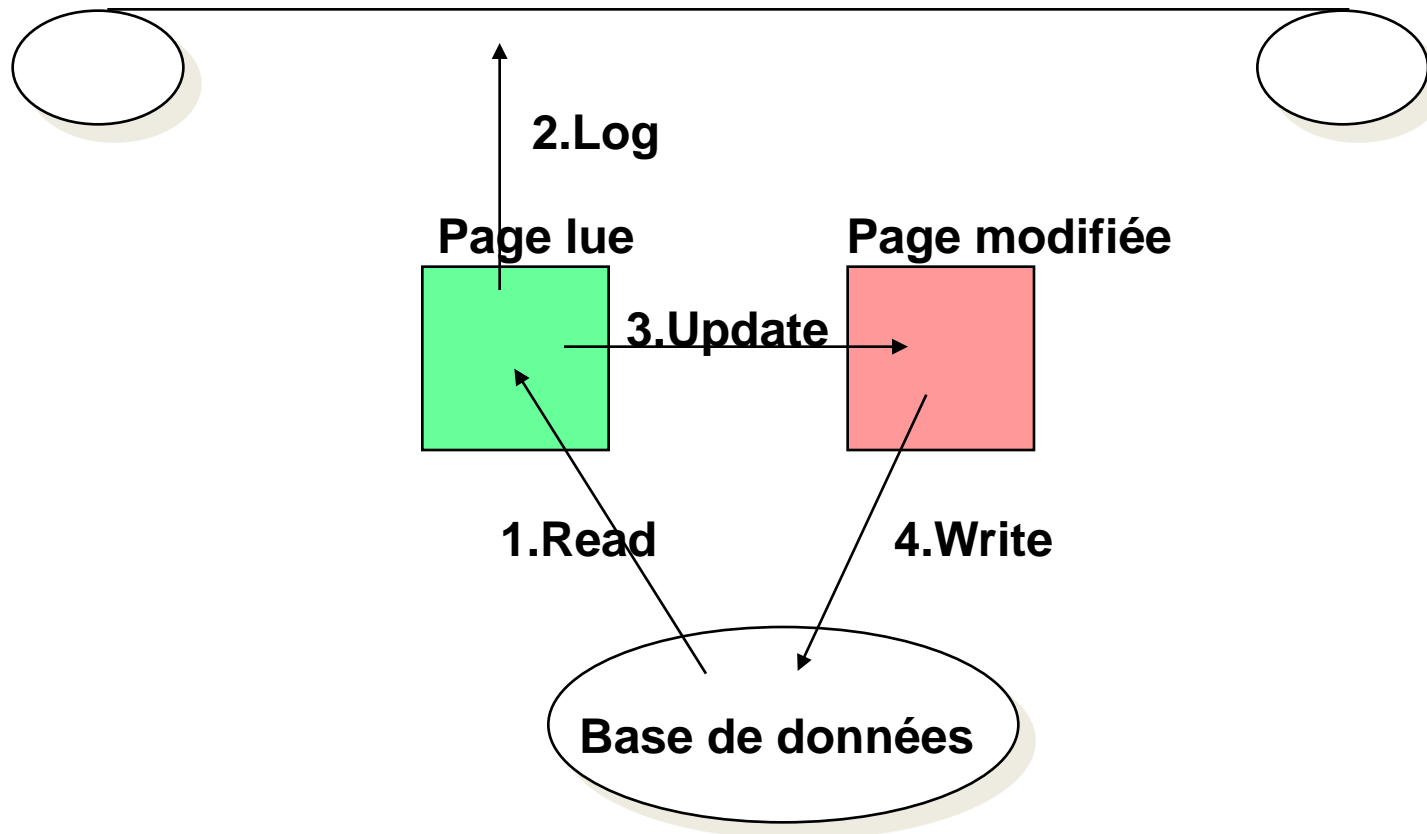
- API pour transaction simple
 - Trid Begin (context*)
 - Commit ()
 - Abort()
- Possibilité de points de sauvegarde :
 - Savepoint Save()
 - Rollback (savepoint) // savepoint = 0 ==> Abort
- Quelques interfaces supplémentaires
 - ChainWork (context*) //Commit + Begin
 - Trid Mytrid()
 - Status(Trid) // Active, Aborting, Committing, Aborted, Committed

2. Journaux et Sauvegarde

- **Journal des images avant**
 - Journal contenant les débuts de transactions, les valeurs d'enregistrement avant mises à jour, les fins de transactions (commit ou abort)
 - Il permet de défaire les mises à jour effectuées par une transaction
- **Journal des images après**
 - Journal contenant les débuts de transactions, les valeurs d'enregistrement après mises à jour, les fins de transactions (commit ou abort)
 - Il permet de refaire les mises à jour effectuées par une transaction

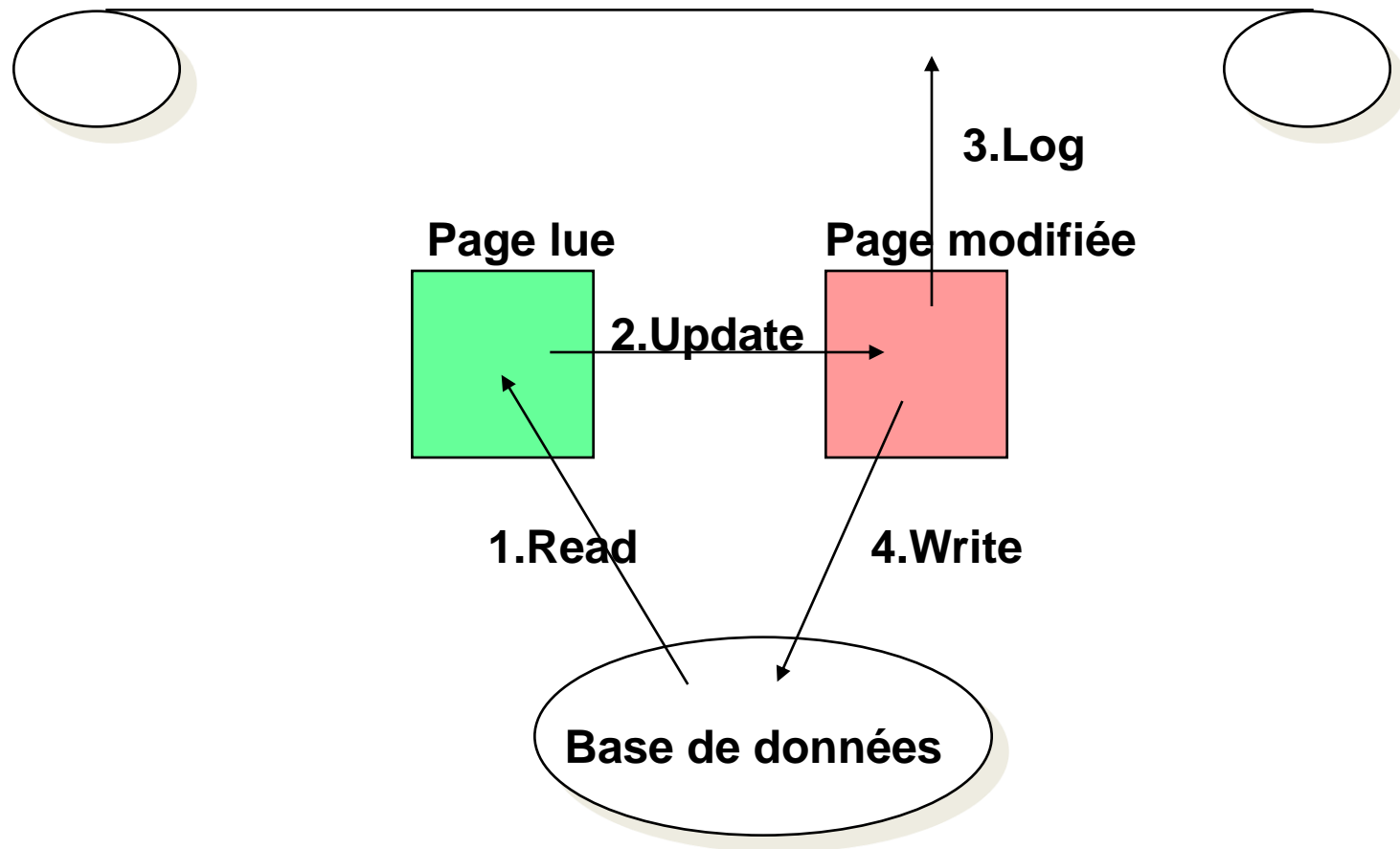
Journal des images avant

- Utilisé pour défaire les mises à jour : Undo



Journal des images après

- Utilisé pour refaire les mises à jour : Redo



Gestion du journal

- Journal avant et après sont unifiés
- Écrits dans un tampon en mémoire et vider sur disque en début de commit
- Structure d'un enregistrement :
 - N° transaction (Trid)
 - Type enregistrement :
 - {début, update, insert, commit, abort}
 - TupleId
 - [Attribut modifié, Ancienne valeur, Nouvelle valeur] ...
- Problème de taille
 - on tourne sur N fichiers de taille fixe
 - possibilité d'utiliser un fichier haché sur Trid/Tid

Sauvegarde

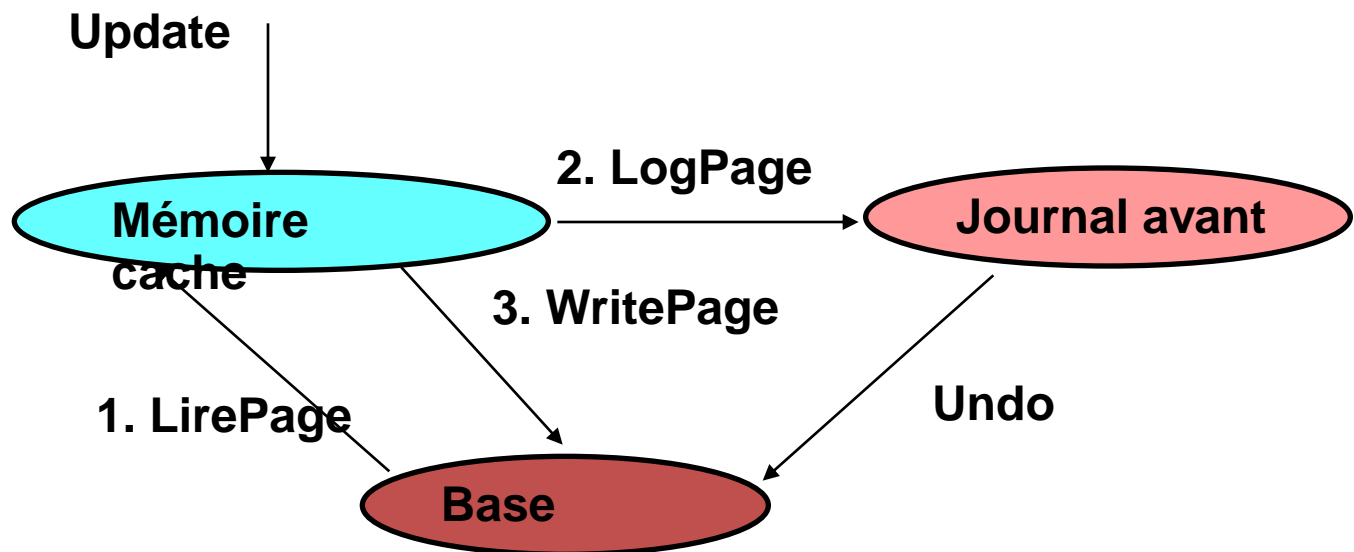
- Sauvegarde périodique de la base
 - toutes les heures, jours, ...
 - Doit être effectuée en parallèle aux mises à jour
- Un Point de Reprise (checkpoint) est écrit dans le journal pour le synchroniser par rapport à la sauvegarde
 - permet de situer les transactions effectuées après la sauvegarde
- Pose d'un point de reprise :
 - écrire les buffers de journalisation (Log)
 - écrire les buffers de pages (DB)
 - écrire un record spécial "checkpoint" dans le journal

3. Scénarios de Reprise

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

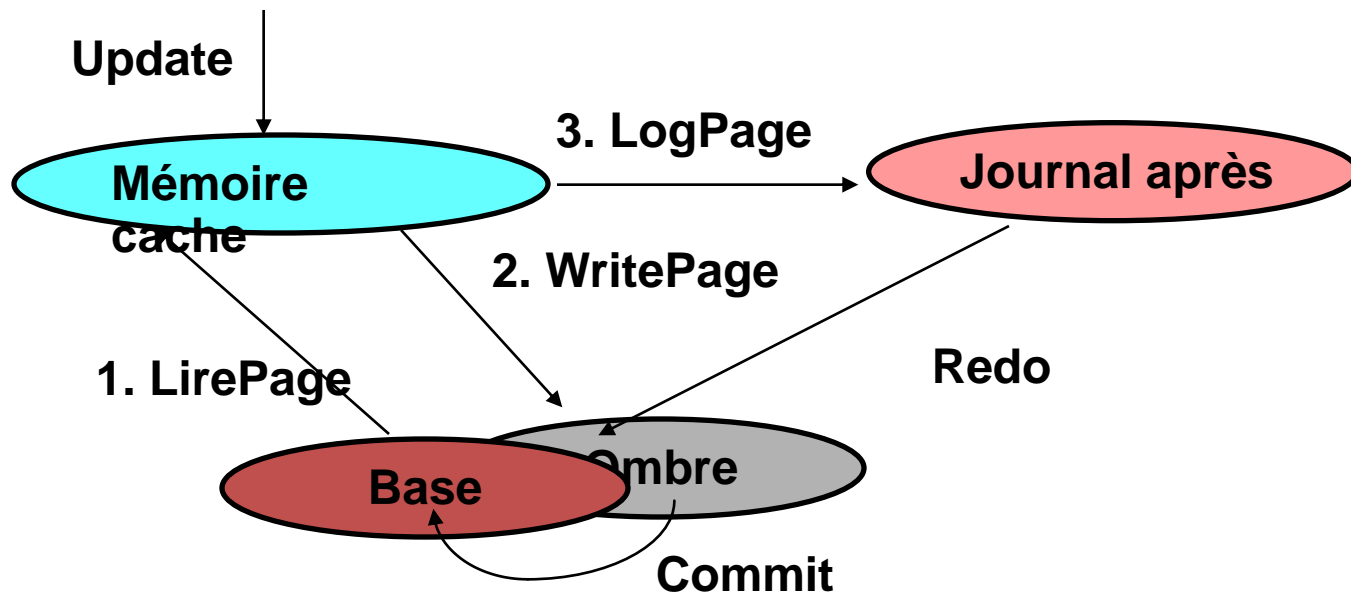
Stratégie do-undo

- Mises à jour en place
 - l'objet est modifié dans la base
- Utilisation des images avant
 - copie de l'objet avant mise à jour
 - utilisée pour défaire en cas de panne

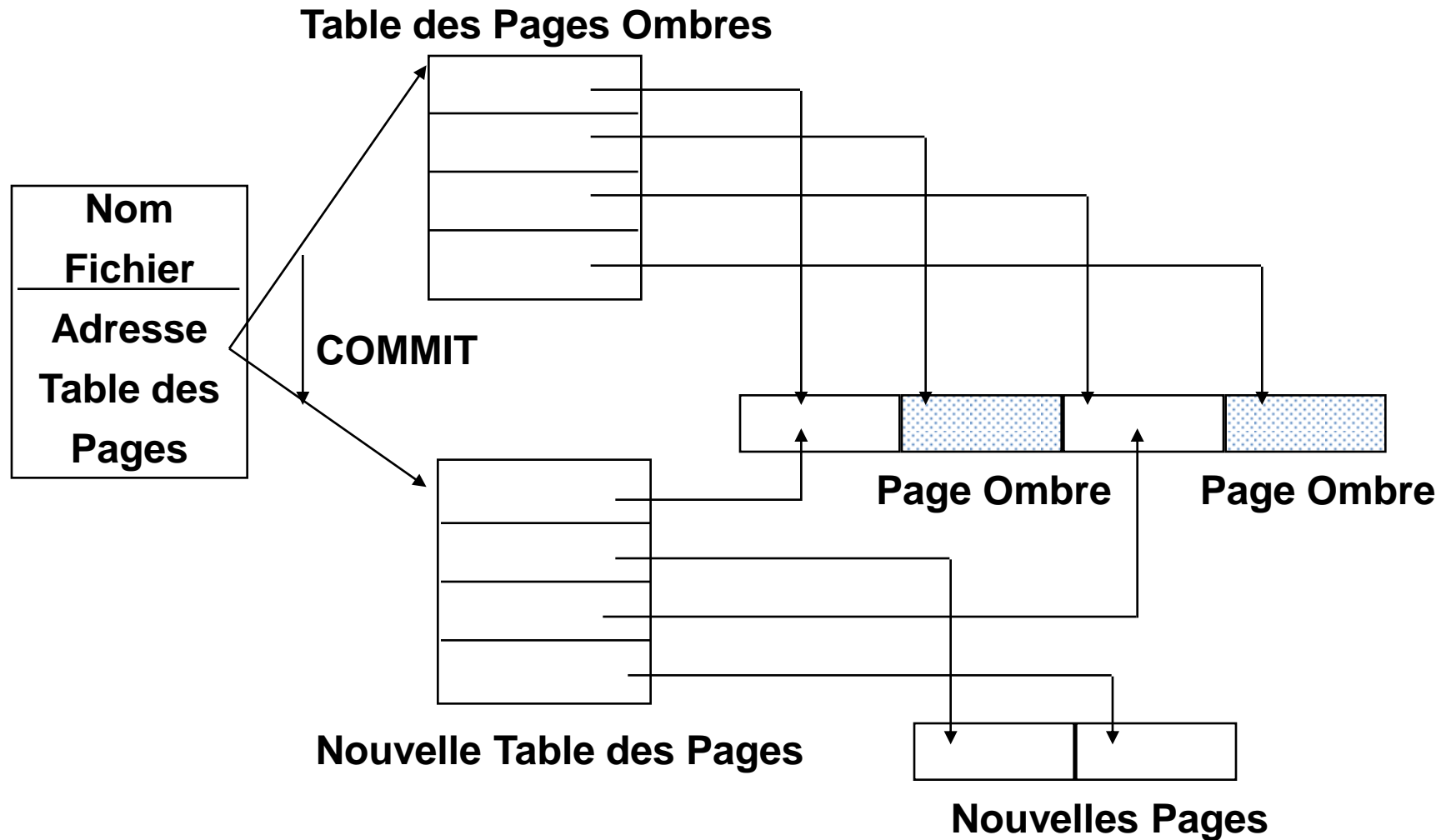


Stratégie do-redo

- Mises à jour en différentiel
 - l'objet est modifié en page différentielle (non en place=ombre)
- Utilisation des images après
 - copie de l'objet en journal après mise à jour (do)
 - utilisée pour refaire en cas de panne (redo)



Pages ombres



La gestion des buffers

- Bufferisation des journaux
 - on écrit le journal lorsqu'un buffer est plein
 - ou lorsqu'une transaction commet
- Bufferisation des bases
 - on modifie la page en mémoire
 - le vidage sur disque s'effectue en différé (processus E/S)
- Synchronisation journaux / base
 - le journal doit toujours être écrit avant modification de la base !

Commits bloqués

- AFIN D'EVITER 3 E/S POUR 1:
 - Le système reporte l'enregistrement des journaux au commit
 - Il force plusieurs transactions à commettre ensemble
 - Il fait attendre les transactions au commit afin de bloquer un buffer d'écriture dans le journal
- RESULTAT
 - La technique des "commits" bloques permet d'améliorer les performances lors des pointes sans faire attendre trop sensiblement les transactions

Reprise à froid

- En cas de perte d'une partie de la base, on repart de la dernière sauvegarde
- Le système retrouve le checkpoint associé
- Il ré-applique toutes les transactions commises depuis ce point
 - (for each committed T_i : redo (T_i))

5. Modèles étendus

- Applications longues composées de plusieurs transactions coopérantes
- Seules les mises-à-jour sont journalisées
- Si nécessité de défaire une suite de transactions:
 - contexte ad-hoc dans une table temporaire
 - nécessité d'exécuter des compensations

Points de Sauvegardes

- Introduction de points de sauvegarde intermédiaires

- (savepoint, commitpoint)

- Begin_Trans

- update

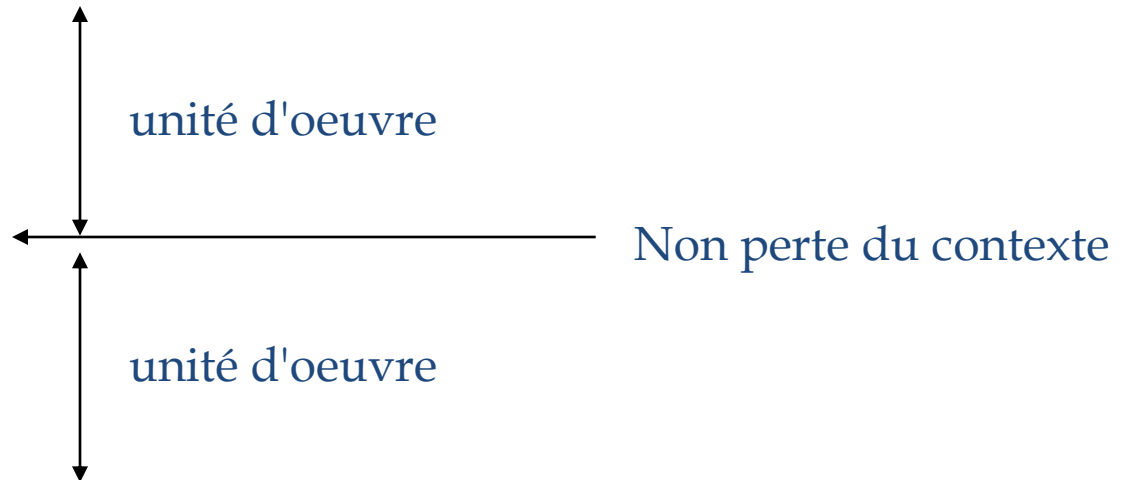
- update

- **savepoint**

- update

- update

- Commit



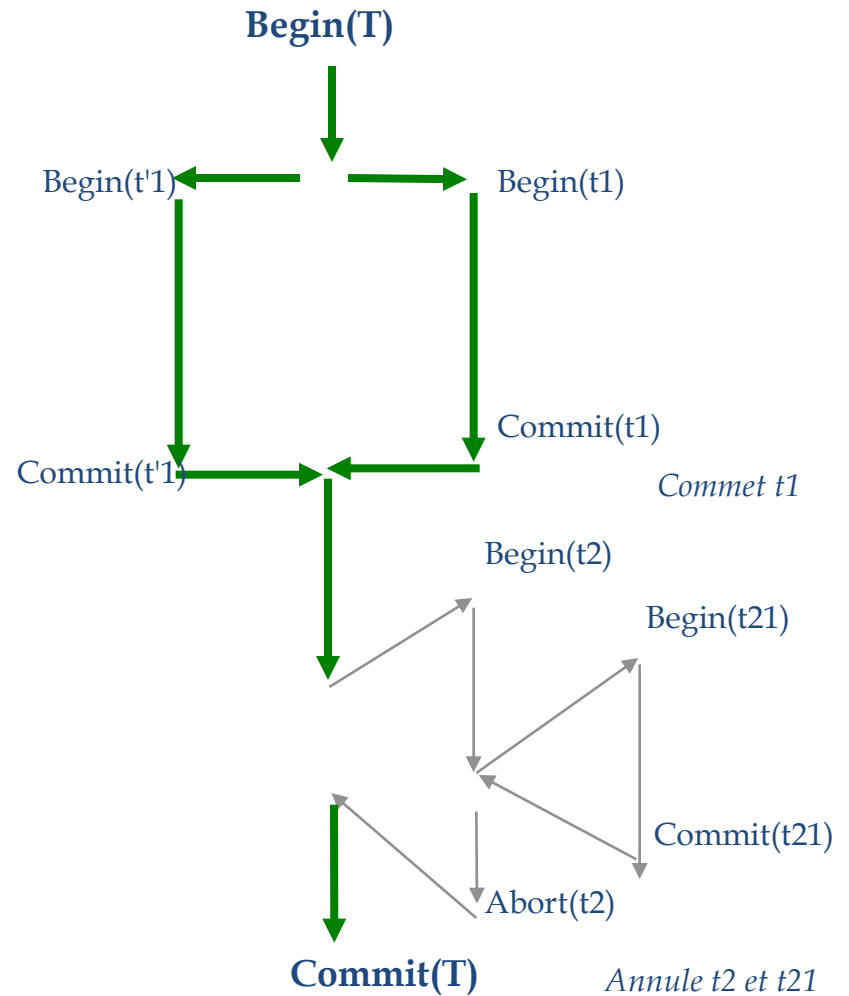
Transactions Imbriquées

- OBJECTIFS

- Obtenir un mécanisme de reprise multi-niveaux
- Permettre de reprendre des parties logiques de transactions
- Faciliter l'exécution parallèle de sous-transactions

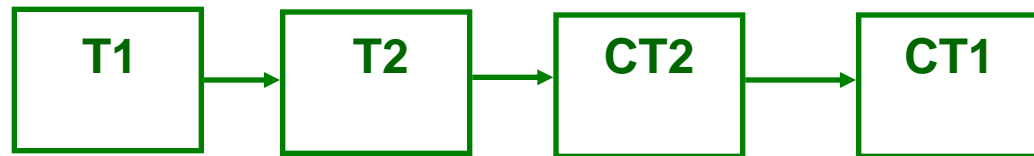
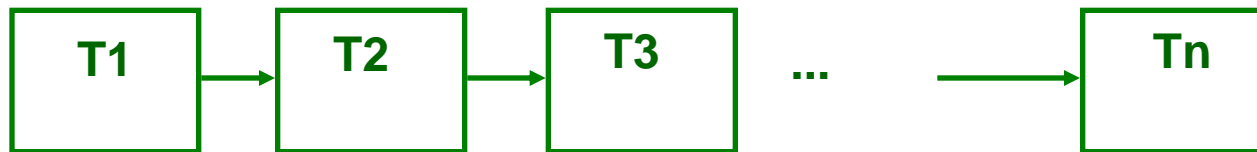
- SCHEMA

- Reprises et abandons partiels
- Possibilité d'ordonner ou non les sous-transactions



Sagas

- Groupe de transactions avec transactions compensatrices
- En cas de panne du groupe, on exécute les compensations



Activités : Propriétés Souhaitées

- contexte persistant
- rollforward, rollback avec compensations
- flot de contrôle dépendant des succès et échecs
 - différencier les échecs systèmes des échecs de programmes
- monitoring d'activités:
 - état d'une activité, arrêt, ...

Langage de Contrôle d'Activités

- Exemple: réservation de vacances
 - T1 : réservation avion alternative : location voiture
 - T2 : réservation hôtel
 - T3 : location voiture
- Activité
 - Ensemble d'exécution de transactions avec alternative ou compensation
- Langage de contrôle d'activités
 - Possibilité de transactions vitales (ex: réservation hôtel)
 - Langage du type :
 - If abort, If commit, Run alternative, Run compensation, ...

6. Transactions réparties

- OBJECTIF
 - Garantir que toutes les mises à jour d'une transaction sont exécutées sur tous les sites ou qu'aucune ne l'est.
- EXEMPLE
 - Transfert de la somme X du compte A vers le compte B
 - DEBUT
 - site 1: $A = A - X$
 - site 2: $B = B + X$ PANNE --> INCOHERENCE DONNEES
 - FIN
- PROBLEME
 - Le contrôle est réparti : chaque site peut décider de valider ou d'annuler ...

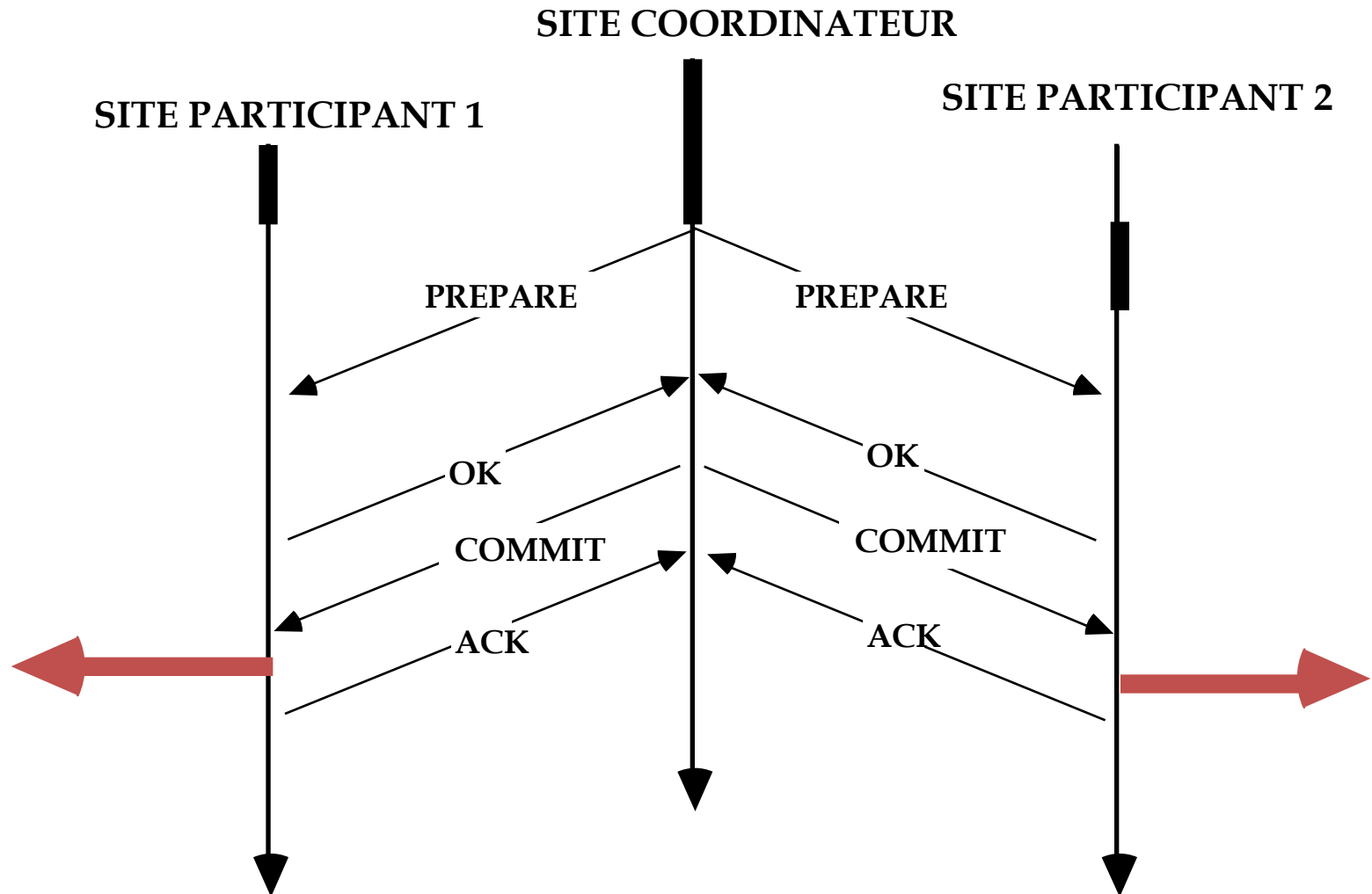
Commit en 2 Phases

- Principe
 - Diviser la commande COMMIT en deux phases
 - *Phase 1* :
 - Préparer à écrire les résultats des mises à jour dans la BD
 - Centralisation du contrôle
 - *Phase 2* :
 - Écrire ces résultats dans la BD
- Coordinateur :
 - Le composant système d'un site qui applique le protocole
- Participant :
 - Le composant système d'un autre site qui participe dans l'exécution de la transaction

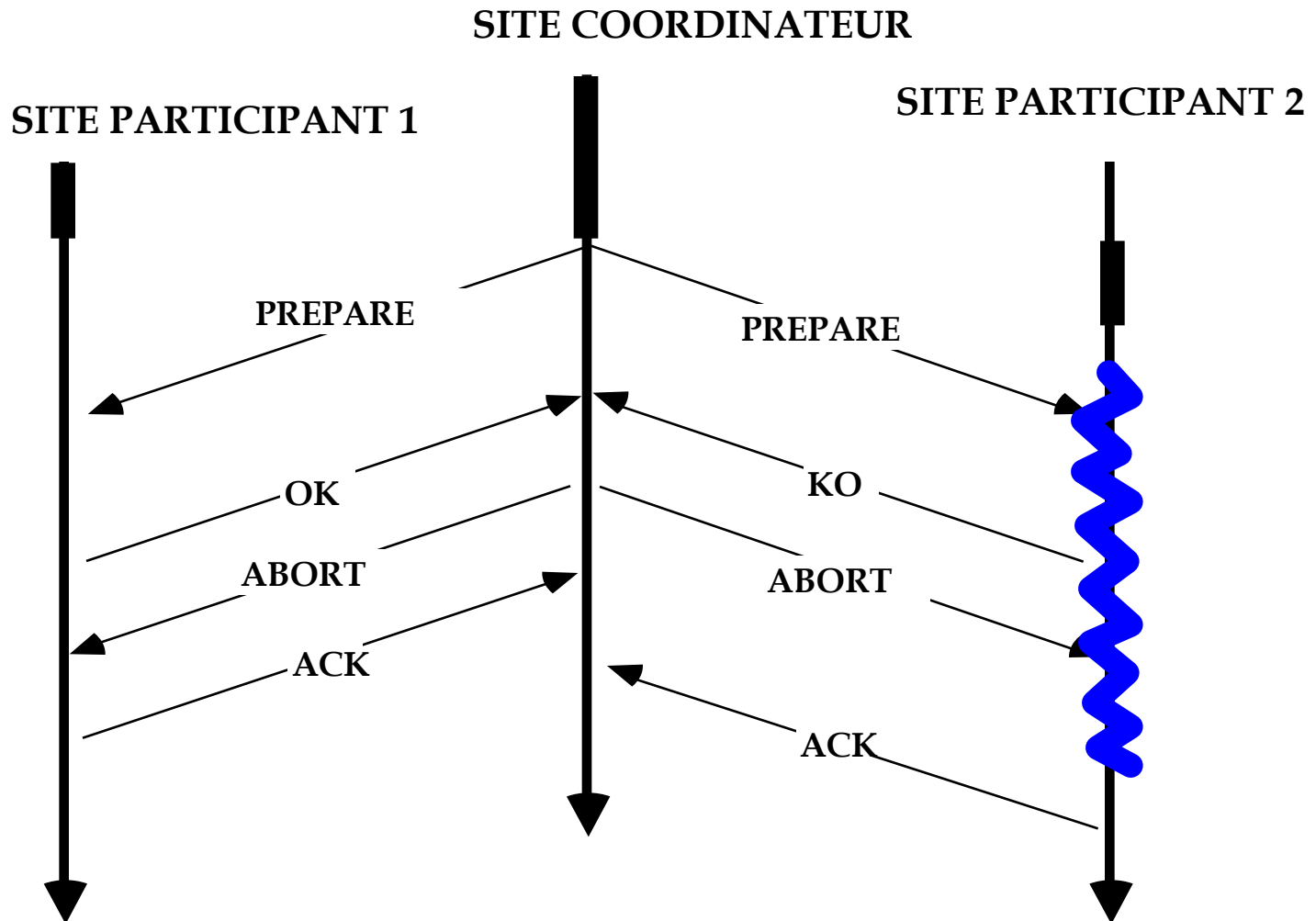
Protocole C/S

- 1. PREPARER
 - Le coordinateur demande aux autres sites s'ils sont prêts à commettre leurs mises à jour.
- 2a. SUCCES : COMMITTRE
 - Tous les participants effectuent leur validation sur ordre du client.
- 2b. ECHEC : ABORT
 - Si un participant n'est pas prêt, le coordinateur demande à tout les autres sites de défaire la transaction.
- REMARQUE
 - Le protocole nécessite la journalisation des mises à jour préparées et des états des transactions dans un journal local à chaque participant.

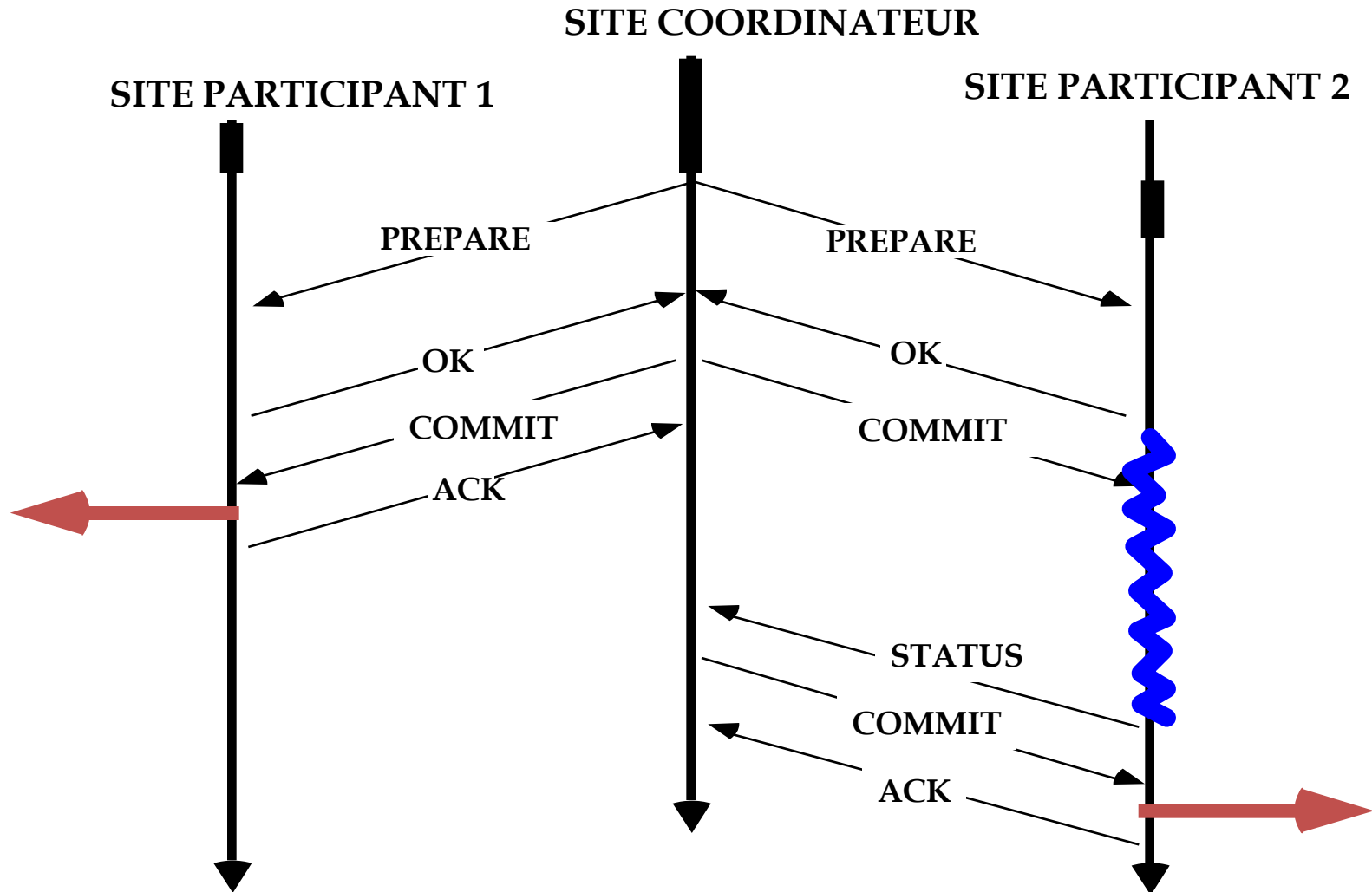
Cas Favorable



Cas Défavorable (1)

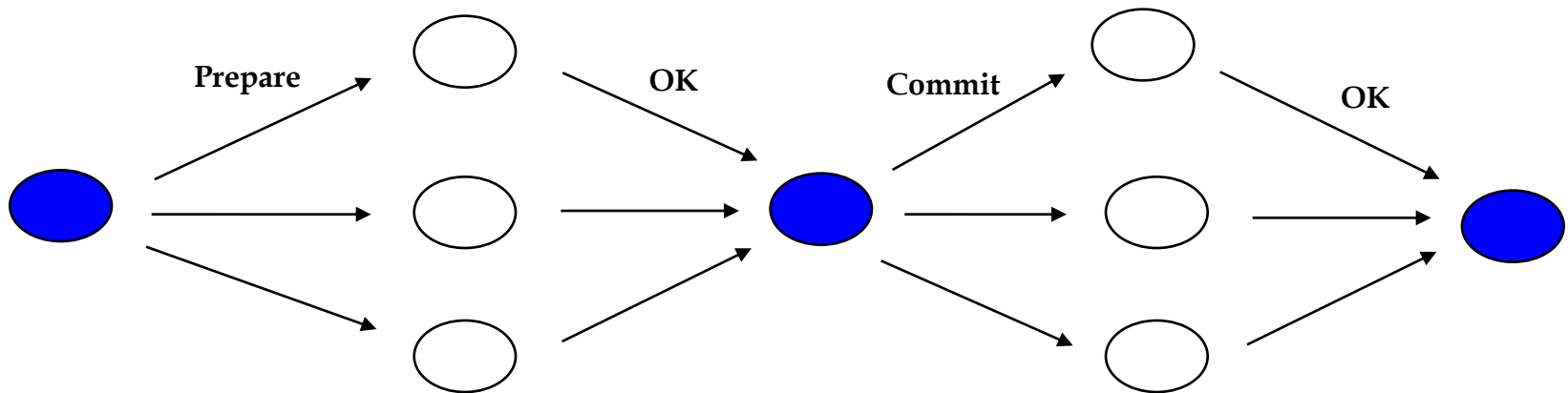


Cas Défavorable (2)

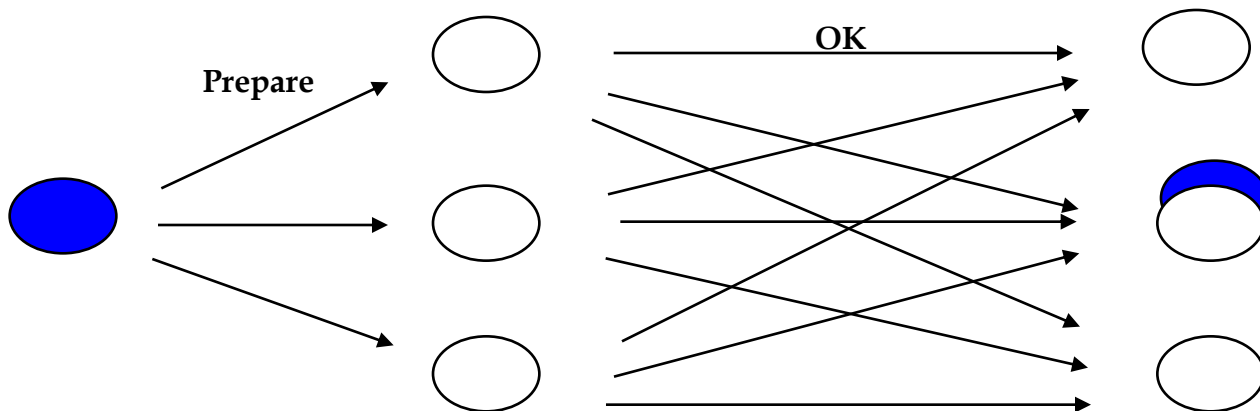


Commit distribué ou centralisé

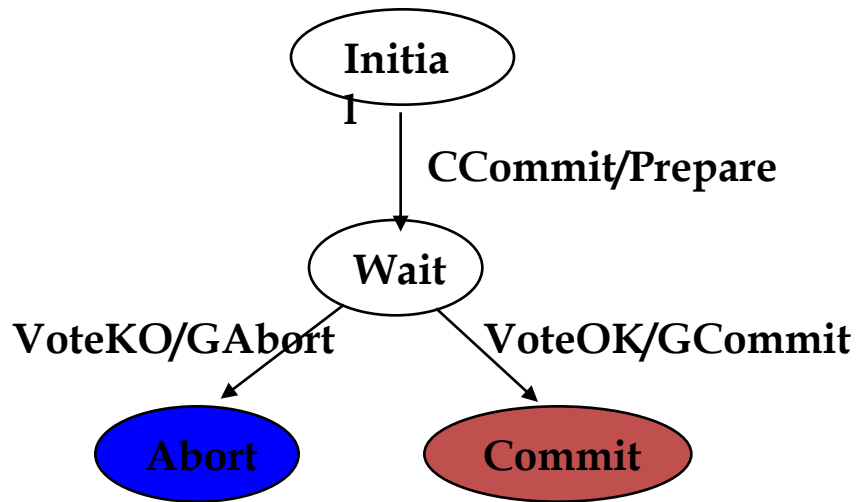
- Validation à deux phases centralisée



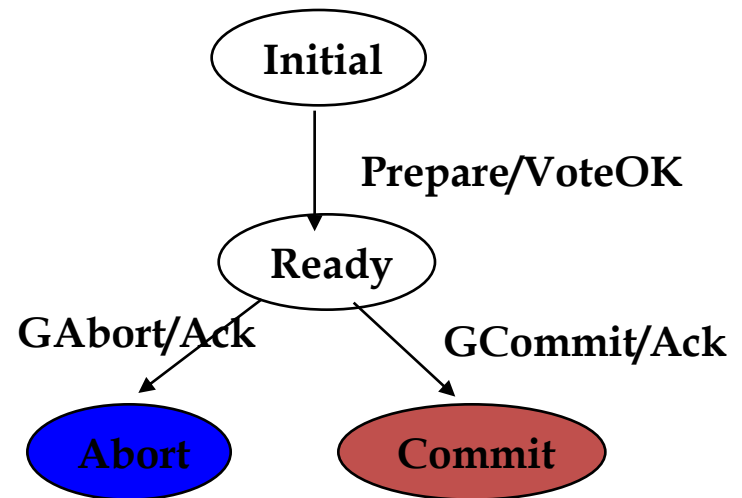
- Possibilité de diffuser la réponse au PREPARE
 - chaque site peut décider localement dans un réseau sans perte



Transitions d'Etats



COORDINATEUR



PARTICIPANT

Transactions bloquées

- Que faire en cas de doute ?
 - Demander l'état aux autres transactions : STATUS
 - conservation des états nécessaires
 - message supplémentaire
 - Forcer la transaction locale : ABORT
 - toute transaction annulée peut être ignorée
 - cohérence garantie avec un réseau sans perte de message
 - Forcer la transaction locale : COMMIT
 - toute transaction commise peut être ignorée
 - non garantie de cohérence avec le coordinateur

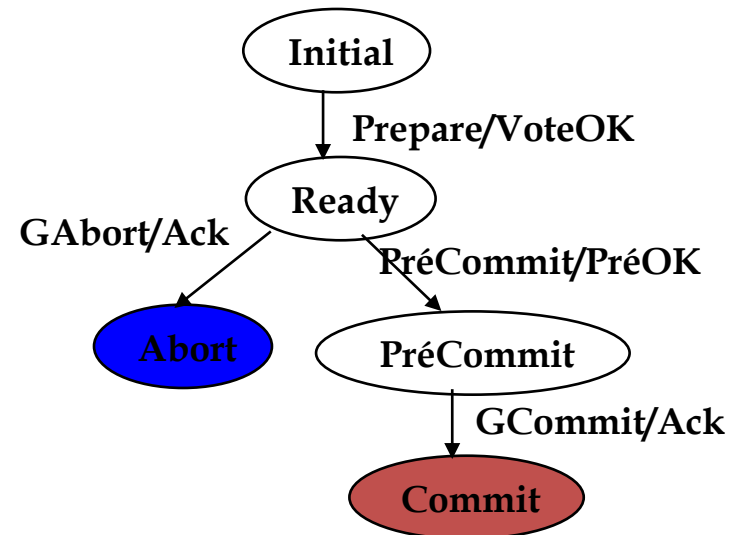
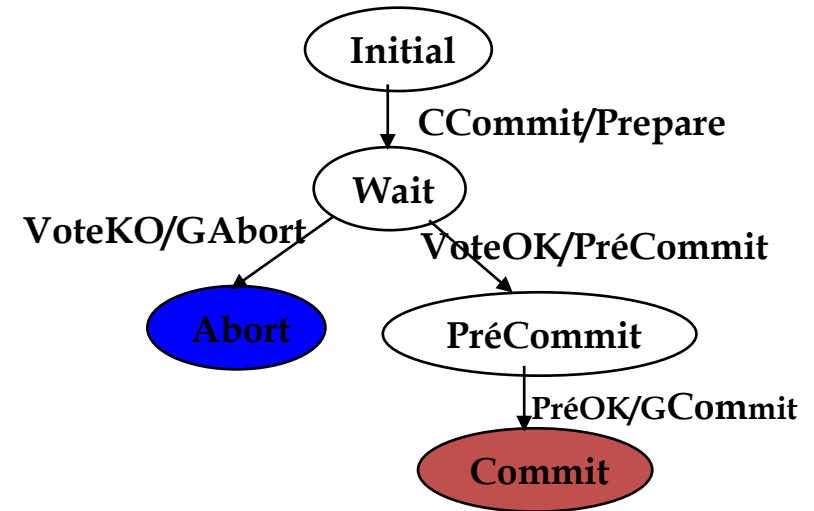
Commit en 3 Phases

- Inconvénient du commit à 2 phases

- en cas de time-out en état “Prêt”, le participant est bloqué
- le commit à 3 phases permet d’éviter les blocages

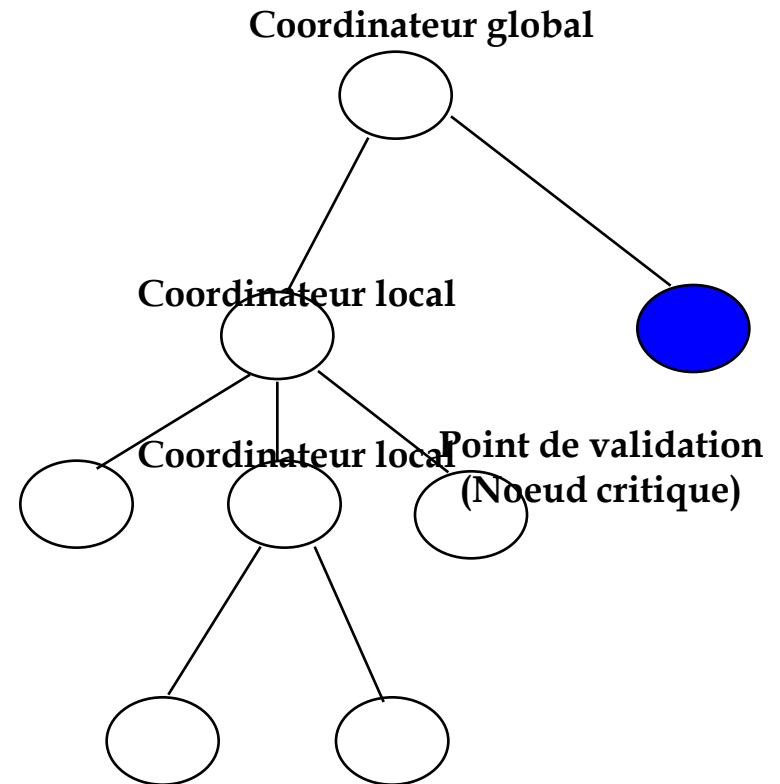
- Messages du commit à 3 phases

- Prepare,
- Prepare to Commit,
- Global-Commit,
- Global-Abort.



Protocole arborescent TP

- TP est le standard proposé par l'ISO dans le cadre OSI
- Protocole arborescent
 - Tout participant peut déclencher une sous-transaction
 - Un responsable de la validation est choisi
 - Un coordinateur est responsable de ses participants pour la phase 1
 - collecte les PREPARE
 - demande la validation
 - Le point de validation est responsable de la phase 2
 - envoie les COMMIT

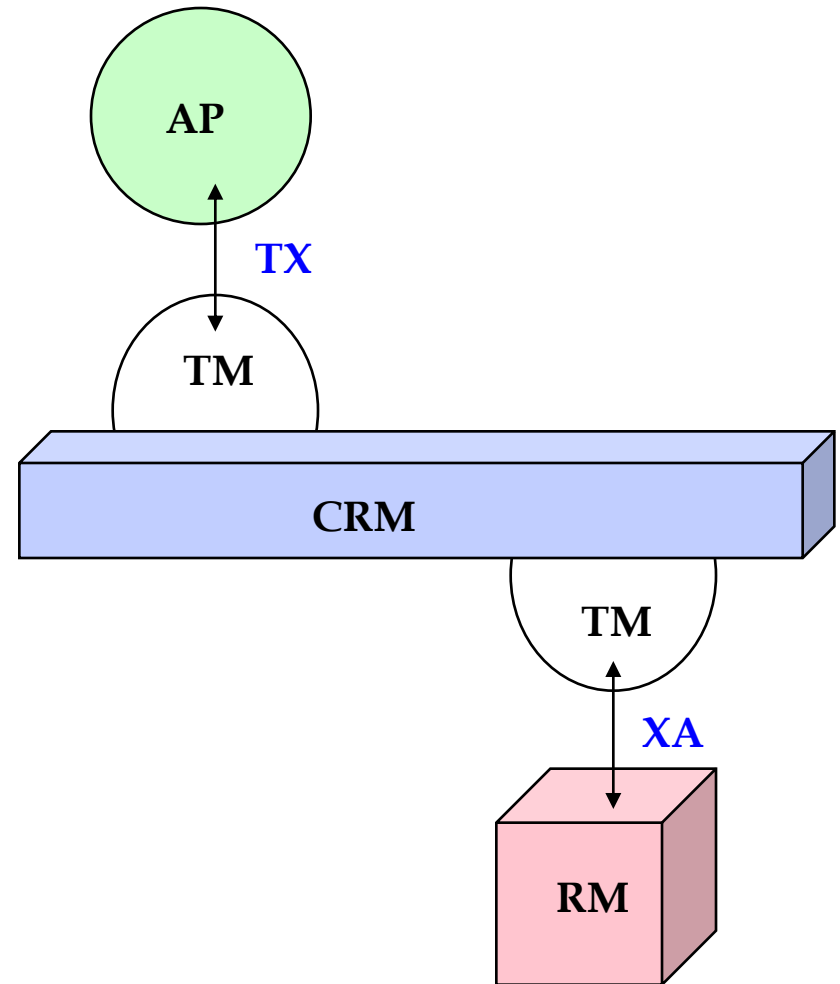


7. Moniteurs transactionnels

- Support de transactions ACID
- Accès continu aux données
- Reprise rapide du système en cas de panne
- Sécurité d'accès
- Performances optimisées
 - Partage de connexions
 - Réutilisation de transactions
- Partage de charge
 - Distribution de transactions
- Support de bases hétérogènes
- Respect des normes et standards

Moniteur transactionnel : Modèle

- Modèle DTP de l'X/OPEN
 - Programme d'application AP
 - Gestionnaire de transactions TM
 - Gestionnaire de communication CRM
 - Gestionnaire de ressources RM
- Interfaces standards
 - TX = interface du TM
 - XA = interface du RM
 - intégration de TP
- Types de RM
 - gestionnaire de fichiers
 - SGBD
 - périphérique



Interface applicative TX

- `tx_open`
 - ordonne au TM d'initialiser la communication avec tous les RM dont les librairies d'accès ont été liées à l'application.
- `tx_begin`
 - ordonne au TM de demander aux RM de débiter une transaction.
- `tx_commit` ou `tx_rollback`
 - ordonne au TM de coordonner soit la validation soit l'abandon de la transaction sur tous les RM impliqués.
- `tx_set_transaction_timeout`
 - positionne un " timeout " sur les transactions
- `tx_info`
 - permet d'obtenir des informations sur le statut de la transaction.

Interface ressource XA

- `xa_open`
 - ouvre un contexte pour l'application.
- `xa_start`
 - débute une transaction.
- `xa_end`
 - indique au RM qu'il n'y aura plus de requêtes pour le compte de la transaction courante.
- `xa_prepare`
 - lance l'étape de préparation du commit à deux phases.
- `xa_commit`
 - valide la transaction.
- `xa_rollback`
 - abandonne la transaction.

Principaux moniteurs (1)

- Encina de Transarc
 - issu de CMU (1992), racheté par IBM
 - construit sur DCE (OSF) pour la portabilité et la sécurité
 - transactions imbriquées
 - conformité DTP : Xa, CPI-C, TxRPC
- Open CICS de IBM
 - construit sur Encina (et DCE)
 - reprise de l'existant CICS (API et outils)
 - conformité DTP : Xa, CPI-C

Principaux moniteurs (2)

- Tuxedo de USL
 - éprouvé (depuis 1984), à la base de DTP
 - supporte l'asynchronisme, les priorités et le routage dépendant des données
 - conformité DTP: Xa, Tx, XaTMI, CPI-C, TxRPC
- Top End de NCR
 - produit stratégique d'AT&T
 - respecte le modèle des composants DTP (AP, RM, TM, CRM)
 - haute disponibilité
 - conformité DTP: Xa, Xa+, Xap-Tp, Tx
- Autres : UTM de Siemens, Unikix

MTS de Microsoft

- Microsoft Transaction Server
- Intégré à DCOM
- Partage de grappes de NT (cluster)
- Les disques sont supposés partagés
- Allocation des ressources en pool aux requêtes :
 - pool de connexion aux ressources (SQL Server)
 - pool de transactions (support)
 - pool de machines
- Ne suit pas les standards !

8. Conclusion

- Des techniques complexes
- Un problème bien maîtrisé dans les SGBDR
- La concurrence complique la gestion de transactions
- Les transactions longues restent problématiques
- Enjeu essentiel pour le commerce électronique
 - validation fiable
 - reprise et copies
 - partage de connections
 - partage de charge