Les mecanismes transactionnels
Du client/serveur

Définitions

Base de données répartie

Ensemble de bases de données gérées par des sites différents et apparaissant à l'utilisateur comme une base unique.

SGBD Réparti (ambiguïté de SGBDR)

Système qui gère des collections de BD logiquement reliées, distribuées sur un réseau, en fournissant un mécanisme d'accès qui rend la répartition transparente aux utilisateurs

Définitions

Client de SGBD Répartie

Application qui accède aux informations distribuées par les interfaces du SGBD Réparti.

Serveur de SGBD Répartie

SGBD gérant une base de données locale intégrée dans une base de données répartie

D'une façon générale on parlera de SITE (client ou serveur)

Pourquoi répartir les données ?

La performance d'accès aux bases est limitée

- **Par le nombre d'accès disques nécessaires**
- **Par le volume de données transmis (débit du réseau)**
- **Par le nombre d'accès concurrents**

Les performances peuvent se dégrader rapidement

- **Au-delà de 30 postes clients**
- **Pour des consultations très fréquentes ou très importantes**
- **Dans le cadre d'accès à distance (réseau étendu)**

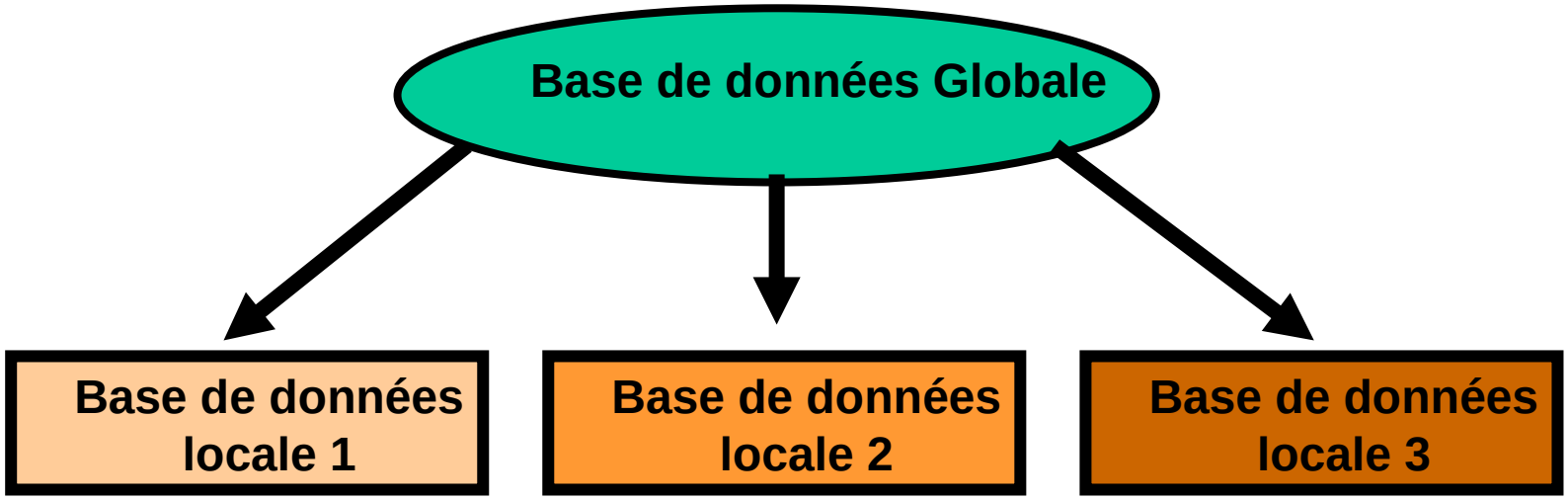
Conception des BdD Réparties

Il existe deux types de conception :

Conception descendante

Conception d'un schéma global

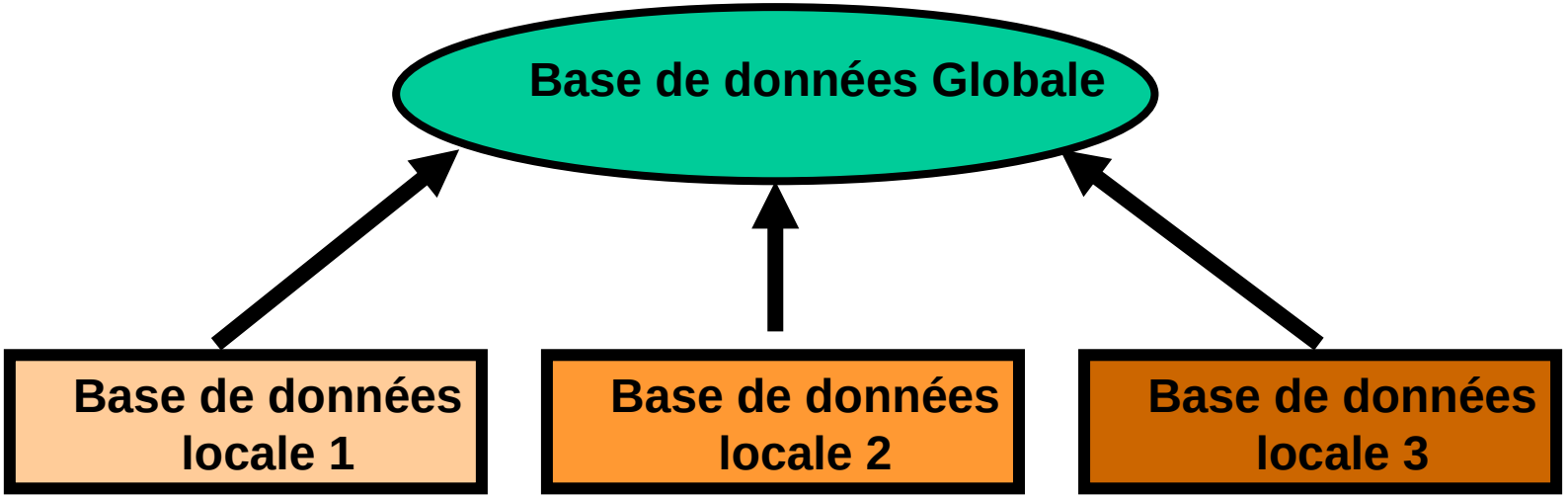
Distribution des objets de ce schéma sur les différents sites pour obtenir des schéma locaux



Conception des BdD Réparties

Conception ascendante

Dans ce cas une base de données globale fédère des base de données locales afin de créer un ou plusieurs schémas globaux. (Le plus souvent il y refonte des schémas locaux)



Conception des BdD Réparties

Les deux cas reviennent à partager,

fragmenter la base de données globale entre plusieurs sites.

Fragment

Un fragment est une sous-table obtenue par sélection de lignes et de colonnes à partir d'une table globale, localisée sur un site unique.

(peut correspondre également à la table entière)

Conception des BdD Réparties

2 Types de fragmentation :

Fragmentation Horizontale

Découpage d'une table en sélectionnant des lignes (Il s'agit d'une sélection SQL).

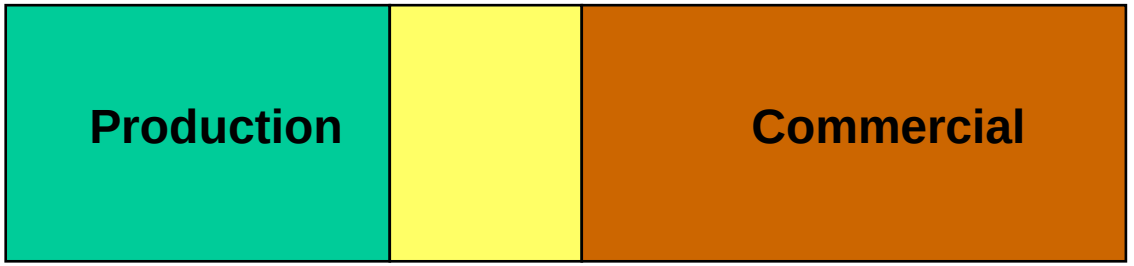
Exemple : Table VENDEUR fragmentée selon les régions d'affectation des représentants

Conception des BdD Réparties

Découpage d'une table en sélectionnant des colonnes (Il s'agit d'une projection SQL).

Fragmentation Verticale

Exemple : Table PRODUIT fragmentée selon les fonctions commerciale et production (Pour la production projection sur : Ref, Desig et cout (Pour le commercial projection sur : Ref, Desig, Prix et Conditionnement)



Conception des BdD Réparties

Fragmentation Mixte

Résultat d'un fragmentation horizontale et verticale.

La recomposition de la table originale doit toujours être possible par :

- **L'union des fragments horizontaux,**
- **La jointure des fragments verticaux.**

Conception des BdD Réparties

Allocation des fragments (*)

Les fragments peuvent être :

Dupliqués sur les sites

Les fragments apparaissent plusieurs fois.

Placés (répartis) sur les sites

Les fragments n'apparaissent que sur un seul site.

La Gestion des Transactions

Transaction : suite atomique d'actions

• **Atomicité**: Lorsqu'une transaction est exécutée, toutes les opérations qu'elle comprend sont exécutées. Si une des
opérations n'est pas exécutée, aucune des opérations n'est exécutée.

• **Consistence**: Une transaction transforme un système d'un état consistent à un autre état consistent.

• **Isolation:** Les résultats d'une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée (commit).

• **Durabilité**: Une fois qu'une transaction est complétée, le système doit garantir que les résultats de ses opérations sont durables
même en cas de pannes subséquentes du système.

Atomicité

Introduction de deux instructions :

– **Commit :** validation des modifications (écriture sur disque)

– **Abort :** annulation des modifications (retour en arrière) (Rollback sous ORACLE)

Une transaction peut donc :

– Avoir une « vie sans histoire » (son commit a été exécuté)

  – Se suicider (son code contient un abort)

  – Etre assassinée (le système a exécuté un abort)

Cohérence

● Perte de mise à jour lorsque deux transactions modifient le même objet

● Deux transactions T1 et T2 ajoutent x au même compte comme suit

– **T1 : lire C.valeur**

– **T1 : C.valeur = C.valeur+ x**

– **T2 : lire C.valeur**

– **T2 : C.valeur= C.valeur + x**

– **T2 : Ecrire C.valeur puis Commit**

– **T1 : Ecrire C.valeur puis Commit**

T2 est perdue

Isolation

- **Lectures impropres : Une transaction modifie un objet lu par une autre transaction**
- **T1 ajoute x mais est annulée et T2 lit valeur**

– T1 : lire C.valeur
– T1 : C.valeur = C.valeur + x
– T1 : Ecrire C.valeur
– T2 : lire C.valeur
– T1 : Abort
– T2 : ...

T2 lit une valeur fausse

Durabilité

- **Une panne peut « annuler » les effets d'une transaction**
- **Exemple**
  - **T1 : lire C.Balance**
  - **T1 : C.valeur = C.valeur + x**
  - **T1 : Ecrire C.valeur**
  - **.....PANNE.....**
  - **Commit**

T1 n'est pas sauvegardée

Transaction répartie

- **Transaction:** peut faire des requêtes à plusieurs serveurs
- **Serveur:** peut faire appel à d'autres serveurs pour traiter une requête

Transaction répartie: une transaction impliquent plus d'un serveur Dont les opérations

- Directement
- Indirectement (transactions imbriquées)

Propriétés à satisfaire:

- Commit ou abort: à la fin de la transaction, tous les serveurs doivent soit compléter, soit abandonner (prennent conjointement la même décision)

Transaction répartie

Contrôle de la concurrence :

- **Transactions doivent être sérialisées globalement sachant que chaque serveur sérialise localement les transactions.**
- **Éliminer les interblocages répartis qui peuvent survenir suite à des cycles de dépendances entre différents serveurs.**

Recouvrement d'abandons de transactions:

- **Chaque serveur doit assurer que ses objets soient récupérables**
- **Garantir que le contenu des objets reflète bien tous les effets des transactions complétées et aucun de celles abandonnées**

Transaction répartie

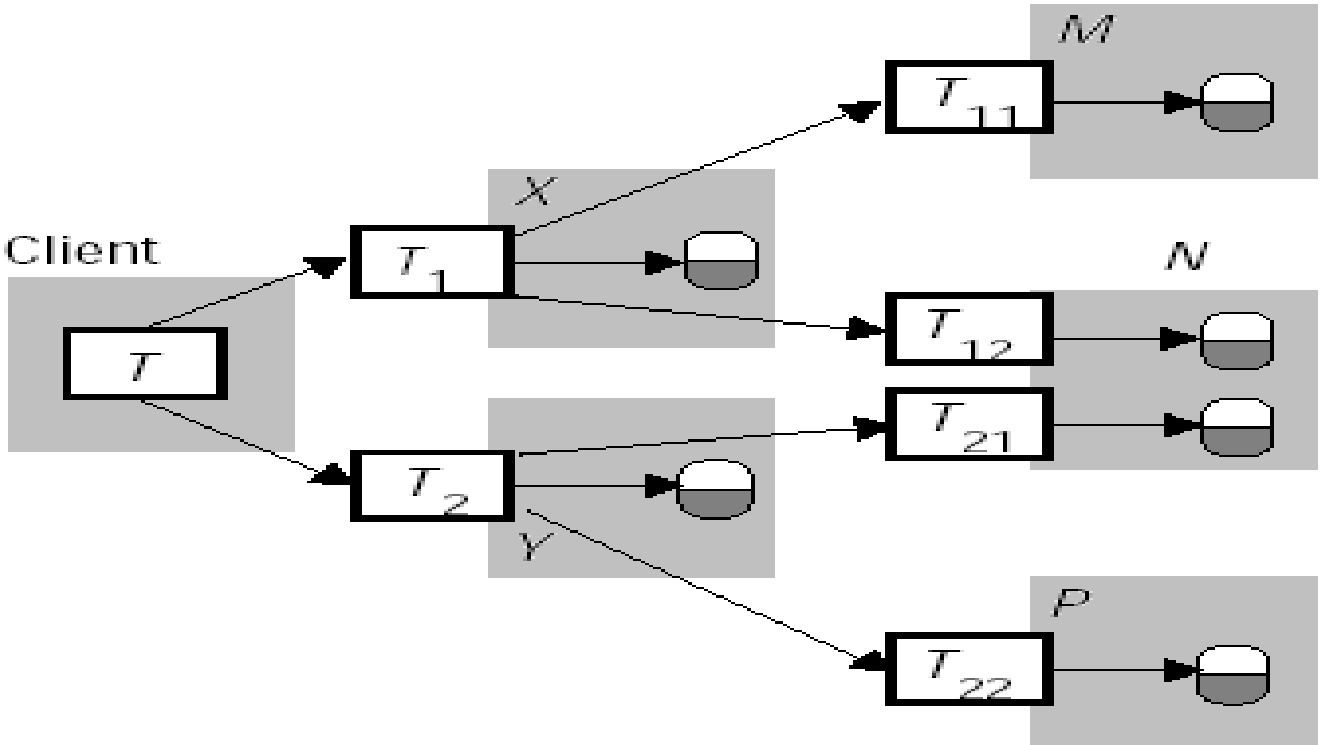
Recouvrement de panne:

Un serveur est désigné comme coordonnateur, il détermine si la transaction est approuvée ou non. Chaque serveur doit finaliser les transactions approuvées même en cas de panne.

Transactions Réparties Imbriquées

Transaction répartie imbriquée :

- **Consiste en une hiérarchie de sous-transactions**
- **Les sous-transactions du même niveau s'exécutent simultanément, s'adressent à plusieurs serveurs et sont elles-mêmes des transactions imbriquées**



Transactions Réparties

- Traitement d'une transaction

Serveurs (exécutant des requêtes appartenant à une transaction imbriquée):

- Doivent communiquer entre eux afin de coordonner leurs actions quand la transaction se termine correctement.
- Réception d'une opération openTransaction par un serveur :

⊢ retourne un identificateur unique pour la nouvelle transaction (identificateur du serveur, e.g., adresse IP, concaténé avec un numéro unique généré localement)

⊢ devient le coordonnateur, responsable de compléter (commit) ou d'abandonner (abort) la transaction

Transactions Réparties

- Traitement d'une transaction

Participants = autres serveurs impliqués dans la transaction

- maintiennent les objets récupérables impliqués dans la transaction
- Coopèrent avec le coordonnateur pour exécuter un protocole de fin de la transaction (commit protocol)

Protocoles de fin de transaction

But = assurer l'atomicité d'une transaction répartie

- Toutes les opérations sont exécutées correctement, ou
- aucune opération n'est exécutée (elle est abandonnée)

Protocole de fin de transaction atomique **à une phase :**

Coordonnateur :

• Envoie d'une façon répétitive un message de fin de transaction aux participants impliqués dans la transaction

• Jusqu'à ce que tous les participants répondent par un accusé de réception

La Gestion des Transactions

Validation en deux phases

Cette validation est basée sur un principe centralisé.

L'exécution de la transaction est contrôlée par un site coordinateur, rôle joué par le client.

Les autres sites intéressés par la transaction sont des participants, rôle joué par les sites serveurs.

La Gestion des Transactions

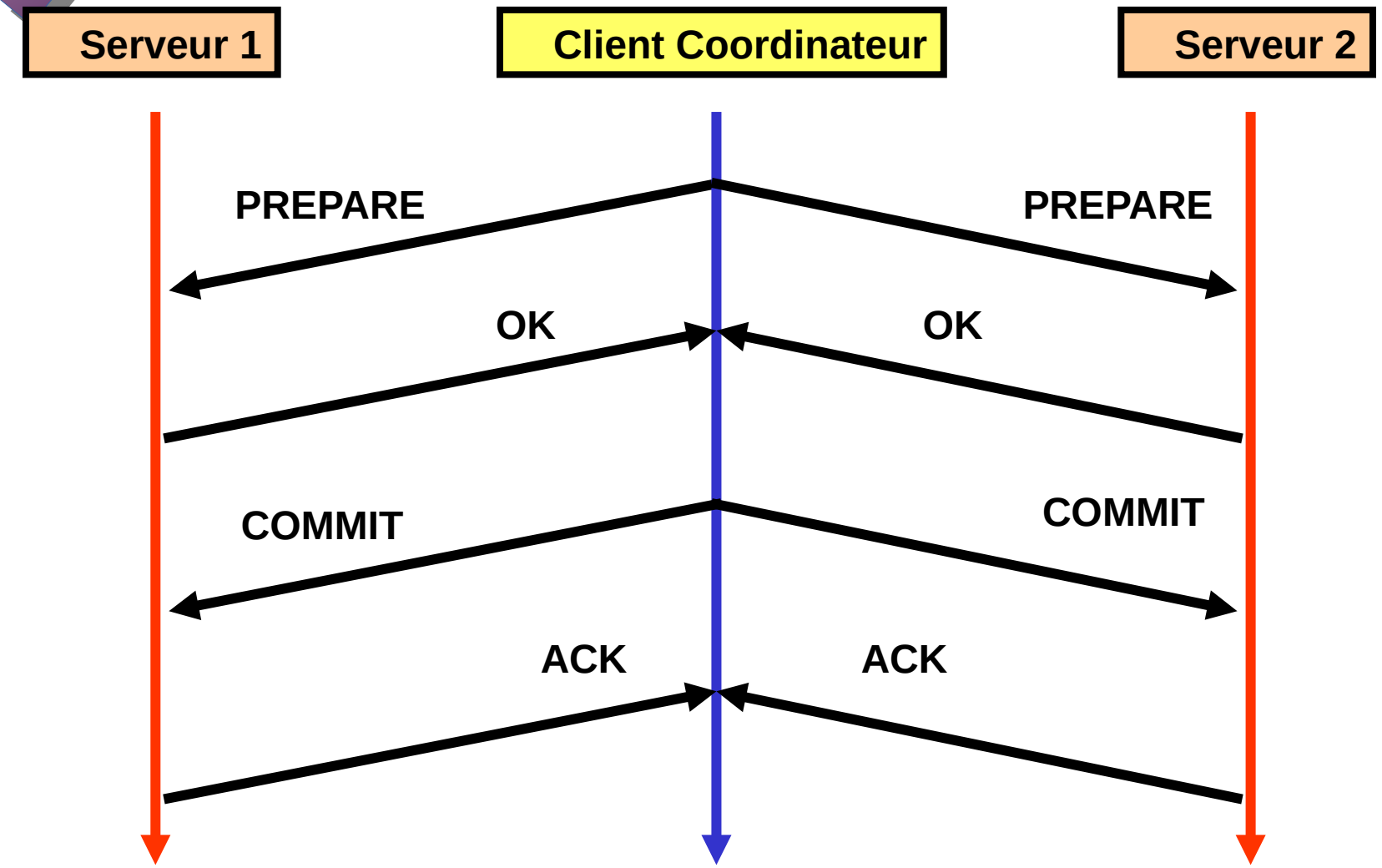
Validation en deux phases

Le client coordinateur demande aux autres sites (serveurs) s'ils sont prêts à mettre à jour leur base (ordre PREPARE).

Si tous les participants répondent positivement (ordre OK) alors le site coordinateur envoie l'ordre COMMIT. Les serveurs envoient un acquittement au coordinateur (ordre ACK).

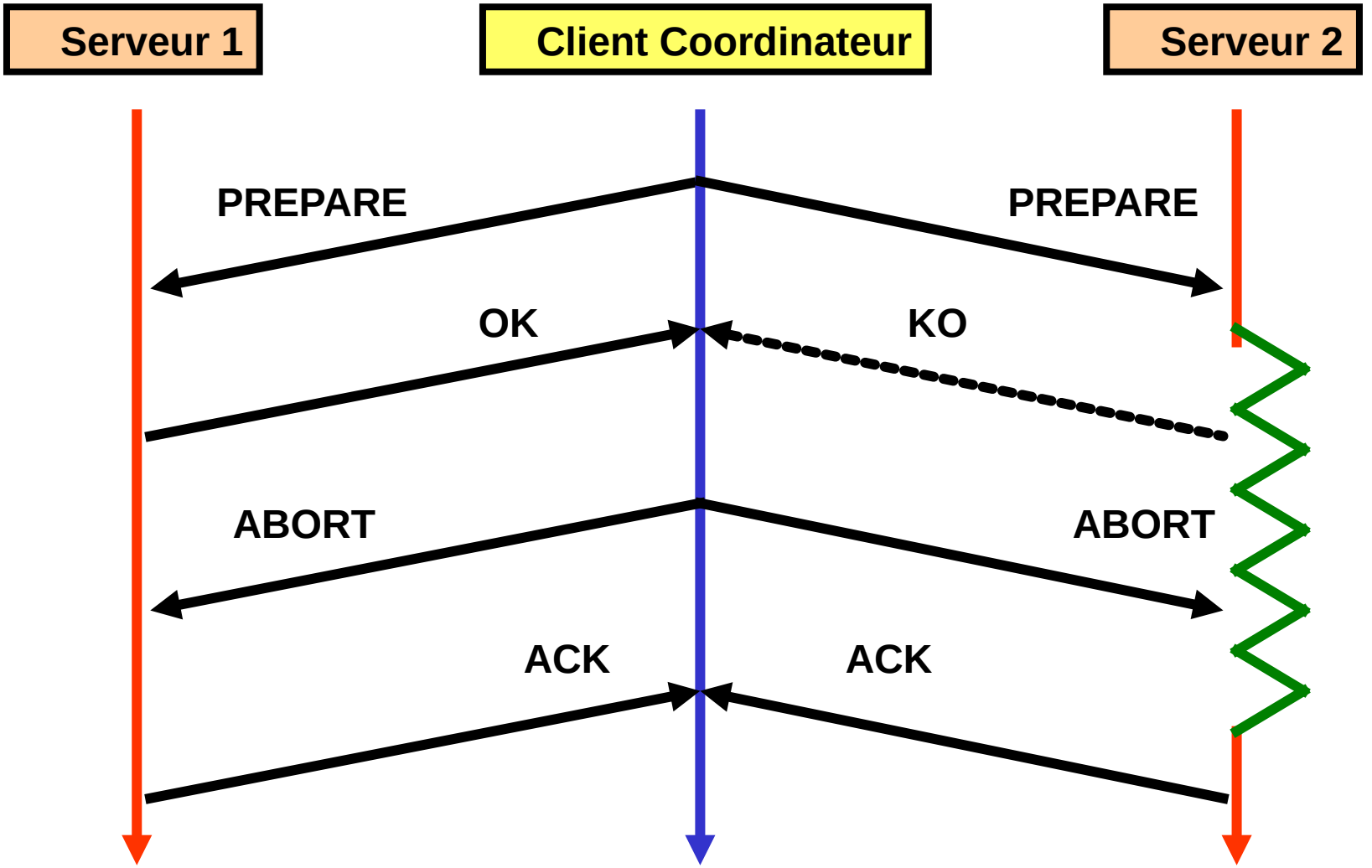
Si l'un des participant répond négativement (ordre KO) alors le site coordinateur envoie l'ordre d'annulation (ordre ABORT).

La Gestion des Transactions



Validation en deux étapes avec succès

La Gestion des Transactions



Validation en deux étapes avec panne totale d'un participant

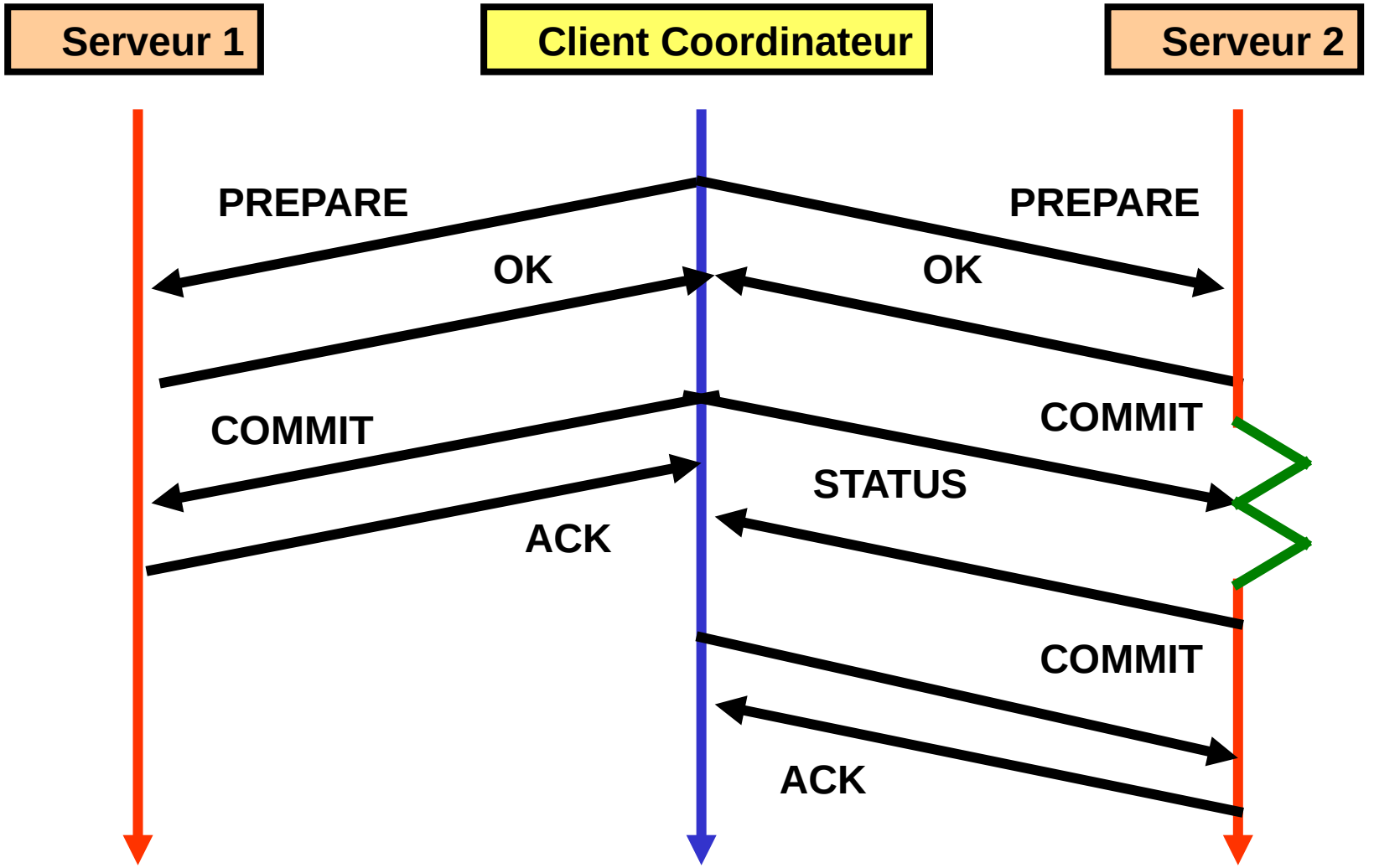
La Gestion des Transactions

Commentaires :

Une non-réponse est assimilée à un refus (time out).

Le serveur 2 annule la transaction car il ne l'a pas accepté (PREPARE mais pas de OK).

La Gestion des Transactions



Validation en deux étapes avec panne partielle d'un

La Gestion des Transactions

Commentaires :

Le serveur 2 a accepté la transaction (OK) puis tombe en panne. COMMIT n'est pas reçu.

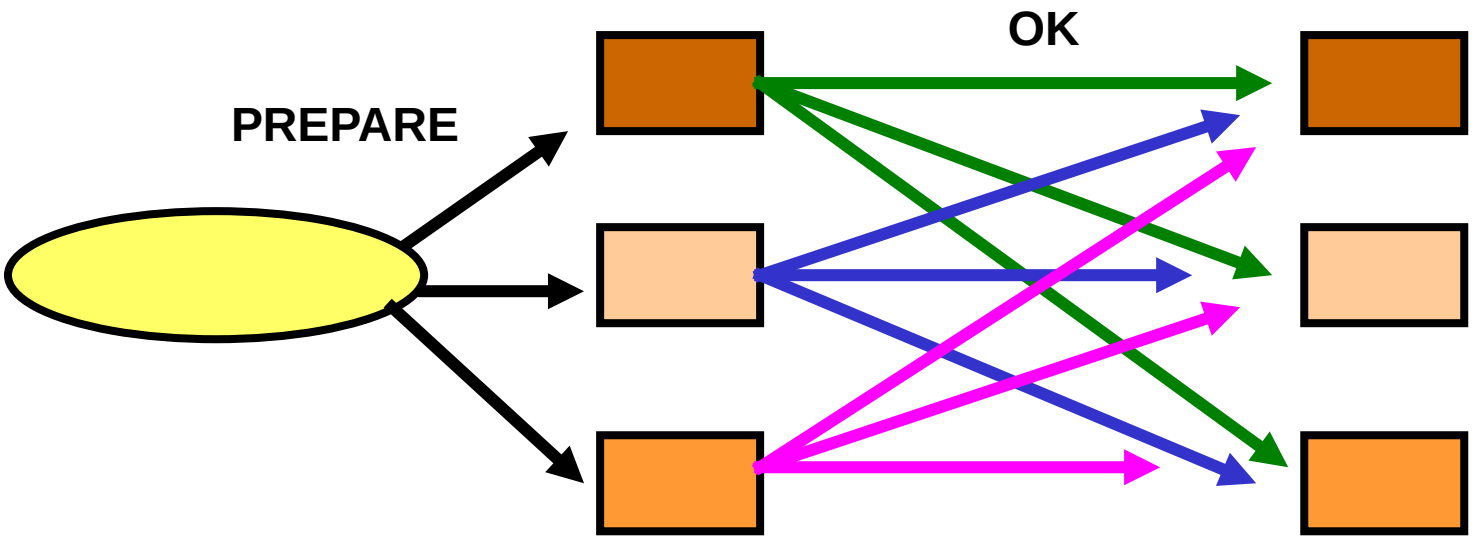
A la reprise, le serveur qui a effectué la sauvegarde sur disque analyse son journal et demande l'état de la transaction qui entre temps a pu être annulée (ordre STATUS).

Dans cet exemple la reprise est faite avec un ordre COMMIT.

La Gestion des Transactions

Validation en deux phases distribué

Dans le cadre d'un réseau local, le message OK est en fait reçu par toutes les stations. Chacune peut donc compter le nombre de OK et valider la transaction



Concurrence

- Types d'accès concurrents

– Lecture – Lecture : ne pose pas de problème

– Ecriture – Ecriture : perte de mise à jour

– Ecriture – Lecture : lectures impropres

Règles pour éviter ces problèmes

1.Toute transaction ne doit pas lire ou écrire un objet modifié par une autre transaction non terminée

2.Toute transaction ne doit pas modifier un objet lu par une autre transaction non terminée

Exemple : perte de mise à jour

Deux transactions T1 et T2 ajoutent x au même compte comme suit

– T1 : lire C.Balance

– T1 : C.Balance = C.valeur + x

– T2 : lire C.valeur

– T2 : C.valeur = C.valeur + x

– T2 : Ecrire C.valeur puis Commit

– T1 : Ecrire C.valeurpuis Commit

Règle 1 non satisfaite

Exemple : lecture impropre

T1 ajoute Delta mais est annulée et T2 lit valeur

– T1 : lire C.valeur

– T1 : C.valeur = C.valeur + x

– T1 : Ecrire Cvaleur

– T2 : lire C.valeur

– T1 : Abort

– T2 : ...

Règle 1 non satisfaite

Exemple : lecture impropre

T1 ajoute puis retire x et T2 lit valeur

– T1 : lire C.valeur

– T1 : C.valeur = C.valeur + x

– T1 : Ecrire C.valeur

– T2 : lire C.valeur

– T1 : C.valeur = C.valeur - x

– T1 : Ecrire C.valeur

– T1 : Commit

Règle 2 non satisfaite

Contrôle de la concurrence pour les transactions réparties

Serveur:

- **Gère un ensemble d'objets.**
- **Assure leur cohérence quand ils sont accédés par des transactions concurrentes.**

  Chaque serveur contrôle la concurrence de ses propres objets.

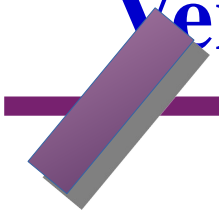
Serveurs qui traitent des transactions réparties: doivent s'assurer que leur exécution est équivalente à une exécution en série.

Algorithme:

Si transaction T est avant U selon un des serveurs

Alors elle doit être dans cet ordre pour tous les serveurs dont les objets sont accédés à la fois par T et U.

Verrouillage des transactions réparties



Verrouillage des transactions réparties

Cas de transaction qui doit être abandonnée suite à un interblocage :

- Le coordonnateur est averti et abandonne la transaction

Cas de transactions imbriquées:

- Transaction parent: ne peut s'exécuter simultanément avec ses transactions enfants
- Héritent des verrous de leurs ancêtres