LA CONCURRENCE

Objectifs
Les bases
Le verrouillage deux phases
L’ordonnancement par estampilles
Les applications avancées

1. Objectifs

- Permettre l’exécution simultanée d’un grand nombre de transactions
- Régler les conflits lecture / écriture
- Garder de très bonne performance
- Eviter les blocages

Les problèmes de concurrence

- Perte d'opérations
  - { T1 : Read A->a; T2 : Read A->b; T2 : b+1 -> b; T2 : Write b->A; T1: a*2 ->a; T1: Write a -> A }
- Introduction d'incohérence
  - A = B { T1 : A*2->A; T2 : A+1->A; T2 : B+1 -> B; T1 : B*2 -> B }
- Non reproductibilité des lectures
  - { T1 : Read A->a; T2 : Read A->b; T2 : b+1 -> b; T2 : Write b->A; T1: Read A -> a }

2. Les bases

- Chaque transaction Ti est composée d'une séquence d'actions <a11, a12, ..., a1ni>
- Une exécution simultanée (Histoire) des transactions {T1, T2, ... Tn} est une séquence d'actions
  - H = < ai1j1, ai2j2 .... aikjk > telle que aij < aij+1 pour tout i et tout j et quel que soit aij de T1 ,... Tn, aij est dans H
  - C'est une séquence d'actions complète respectant l'ordre des actions des transactions
- Une exécution est sérielle si toutes les actions des transactions ne sont pas entrelacées
  - elle est donc de la forme
  - <Tp(1), Tp(2), ...Tp(n)> ou p est une permutation de 1, 2, ... n.

Sérialisabilité

- Exécution sérialisable
  - Une exécution est dite sérialisable si elle est équivalente à une exécution sérielle
- Plusieurs critères d'équivalence possibles
  - Equivalence de vue : tous les résultats visibles sont identiques
  - Equivalence du conflit : toutes les actions conflictuelles sont effectuées dans le même ordre sur les objets de la base

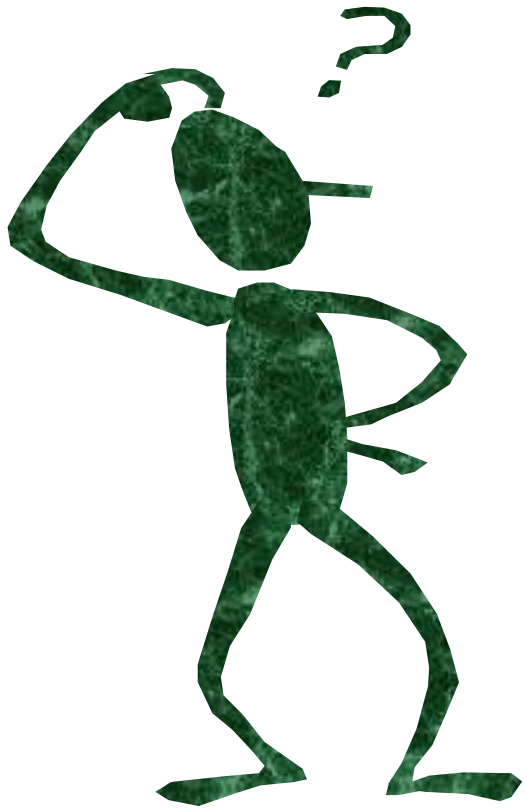
Graphe de précédence

- Précédences
  - Techniques basées sur la seule sémantique des opérations de lecture / écriture
  - Ti lit O avant Tj écrit => Ti précède Tj
  - Ti écrit O avant Tj écrit => Ti précède Tj

- Condition de sérialisabilité
  - Le graphe de précédence doit rester sans circuit

Bilan Problèmatique

- La sérialisabilité est une condition suffisante de correction



- Exercice
  - Démontrer que les cas de perte d'opérations et d'incohérences sont non sérialisables

3. Le Verrouillage 2 phases

- PRINCIPES
  - verrouillage des objets en lecture/écriture
  - opérations Lock(g,M) et Unlock(g)
  - compatibilité:

	L	E
L	V	F
E	F	F

  - toute transaction attend la fin des transactions incompatibles
  - garantie un graphe de précédence sans circuit
  - les circuits sont transformés en verrous mortels

Algorithmes Lock

- Bool Function Lock(Transaction t, Objet O, Mode M){
  Cverrou := 0 ;
  Pour chaque transaction i ≠ t ayant verrouillé l'objet O faire {
  Cverrou := Cverrou U t.verrou(O) } ; // cumuler les verrous sur O
  si Compatible (Mode, Cverrou) alors {
  t.verrou(O) = t.verrou(O) U M; // marquer l'objet verrouillé
  Lock := true ; }
  sinon {
  insérer (t, Mode) dans la queue de O ; // mise en attente de t
  bloquer la transaction t ;
  Lock := false ; } ;
  }

Algorithme Unlock

- Procédure Unlock(Transaction t, Objet O){

  t.verrou(O) := 0 ;

  Pour chaque transaction i dans la queue de O {

  si Lock(i, O,M) alors {

  enlever (i,M) de la queue de O ;

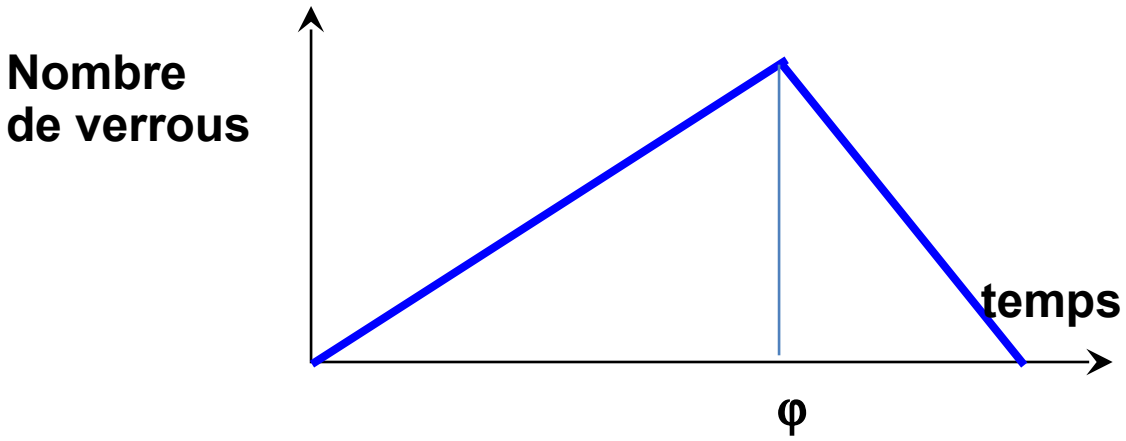
  débloquer i ; } ;

  }

  }

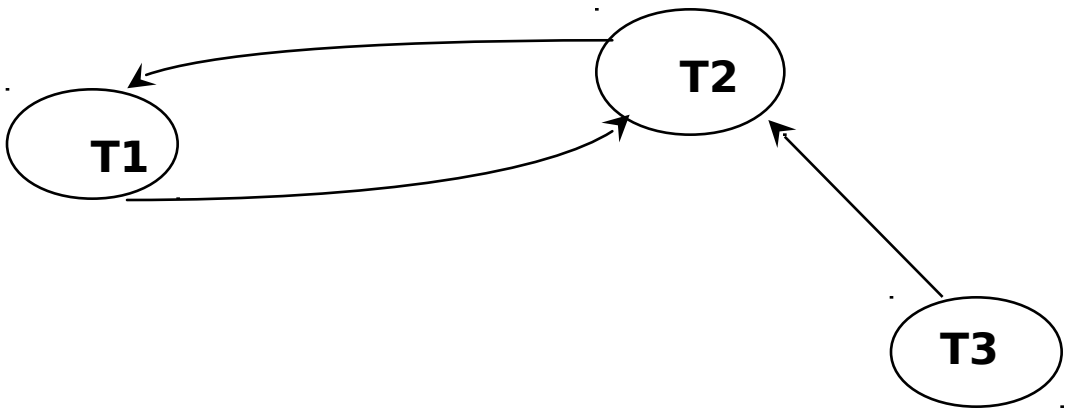
Condition de corrections

- Transactions deux phases
  - une transaction ne peut relâcher de verrous avant de les avoir tous acquis



Problèmes du Verrouillage

- Verrou mortel
  - risques de circuit d'attentes entre transactions



- Granularité des verrous
  - page : en cas de petits objets, trop d'objets verrouillés
  - objet : trop de verrous, gestion difficile

Résolution du verrou mortel

- Prévention
  - définir des critères de priorité de sorte à ce que le problème ne se pose pas
  - exemple : priorité aux transactions les plus anciennes

- Détection
  - gérer le graphe des attentes
  - lancer un algorithme de détection de circuits dès qu'une transaction attend trop longtemps
  - choisir une victime qui brise le circuit

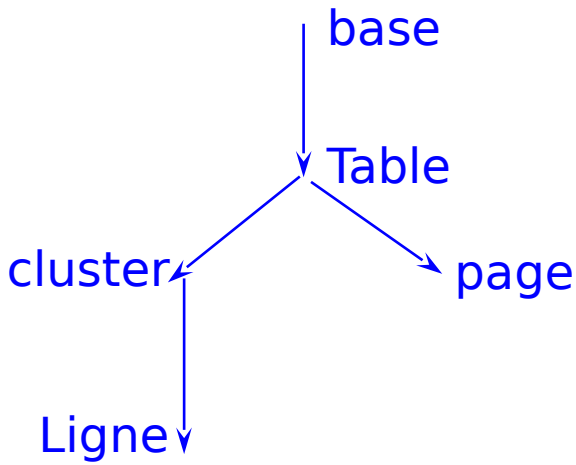
Améliorations du verrouillage

- Relâchement des verrous en lecture après opération
  - - non garantie de la reproductibilité des lectures
  - + verrous conservés moins longtemps

- Accès à la version précédente lors d'une lecture bloquante
  - - nécessité de conserver une version (journaux)
  - + une lecture n'est jamais bloquante

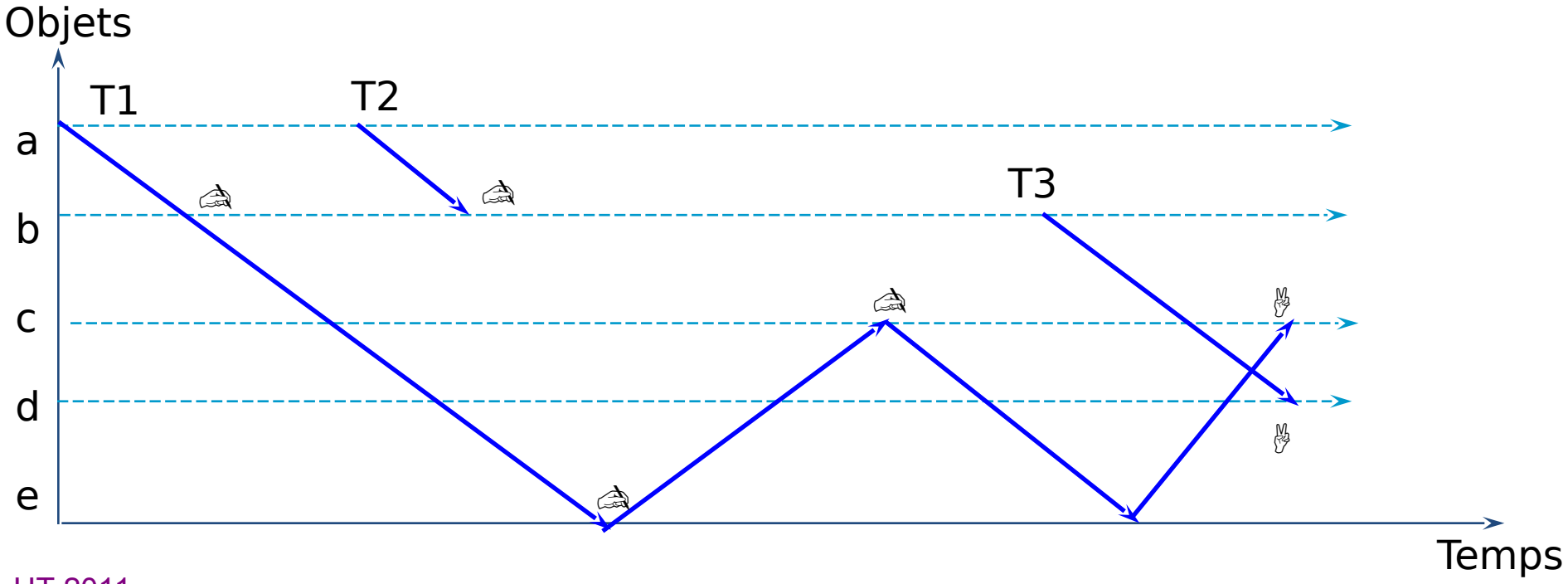
Granularité Variable

- Plusieurs granules de verrouillage sont définis, inclus l'un dans l'autre
- Le verrouillage s'effectue en mode intention sur les granules supérieurs et en mode effectif sur les granules choisis
  - les modes intentions sont compatibles
  - les modes effectifs et intentions obéissent aux compatibilités classiques



Verrouillage Altruiste

- Restitution des verrous sur les données qui ne seront plus utilisées
  - L'abandon d'une transaction provoque des cascades d'abandons
  - Nécessité d'introduire la portée d'une transaction (transactions ouvertes)

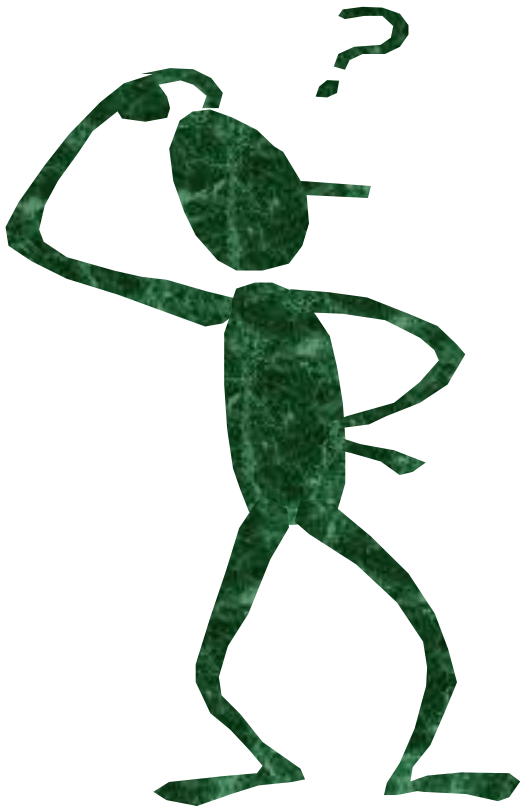


Degré d'isolation en SQL2

- Définition de degrés d'isolation emboîtés
  - Degré 0
    - garantit les non perte des mises à jour
    - pose de verrous courts exclusifs lors des écritures
  - Degré 1
    - garantit la cohérence des mises à jour
    - pose de verrous longs exclusifs en écriture
  - Degré 2
    - garantit la cohérence des lectures individuelles
    - pose de verrous courts partagés en lecture
  - Degré 3
    - garantit la reproductibilité des lectures
    - pose de verrous longs partagés en lecture

Bilan Verrouillage

- Approche pessimiste
  - prévient les conflits
  - assez coûteuse
  - assez complexe
- Approche retenue
  - dans tous les SGBD industriels
- Difficile de faire mieux !



4. Ordonnancement par estampillage

- Estampille (TimeStamp) associée à chaque transaction
  - date de lancement
  - garantie d'ordre total (unicité)
- Conservation des estampilles
  - dernier écrivain Writer
  - plus jeune lecteur Reader
- Contrôle d'ordonnancement
  - en écriture: estampille écrivain > Writer et > Reader
  - en lecture: estampille lecteur > Writer
- Problèmes
  - reprise de transaction en cas d'accès non sérialisé
  - risque d'effet domino en cas de reprise de transaction

Algorithme d'ordonnancement

- Fonction READ (Transaction t, Objet O) {

  si O.Writer ≤ t alors {

  Read := Get(O) ; // effectuer la lecture

  O.Reader := MAX (O.Reader,t) ; // mettre à jour dernier lecteur }

  sinon Abort(t); **}** .

- Fonction Write (Transaction t, Objet O, Contenu C){

  si O.Writer ≤ t et O. Reader ≤ t alors{

  Set(O,C) ; // effectuer l'écriture

  O.Writer := t ; // mettre à jour dernier écrivain }

  sinon Abort(t); **}** .

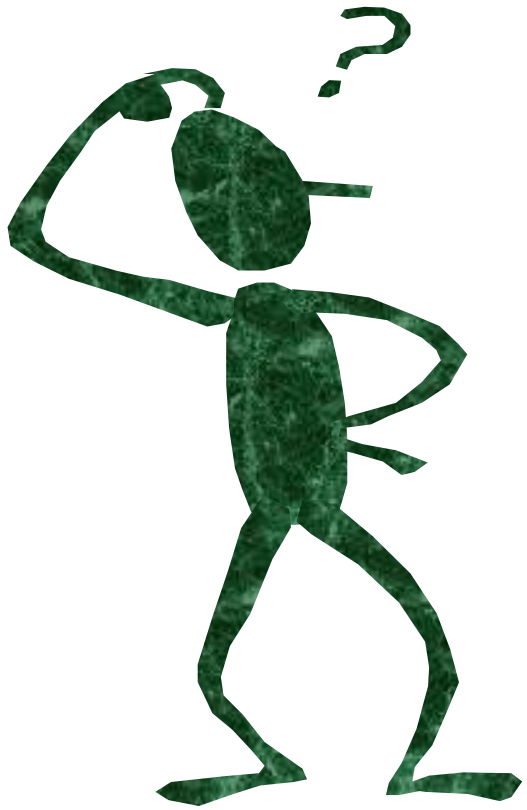
La Certification Optimiste

- Les contrôles s'effectuent seulement en fin de transaction
  - Phase d'accès: on garde les OID des objets lus/écrits
  - Phase de certification: on vérifie l'absence de conflits (L/E ou E/E même objet) avec les transactions certifiée pendant la phase d'accès
  - Phase d'écriture (commit) pour les transactions certifiées

- Avantages et inconvénients:
  - + test simple d'intersection d'ensembles d'OID en fin de transaction
  - - tendance à trop de reprises en cas de conflits fréquents (effondrement)

Bilan Estampillage

- Approche optimiste
  - coût assez faible
  - détecte et guérit les problèmes
- Guérison difficile
  - catastrophique en cas de nombreux conflits
  - absorbe mal les pointes
- Sophistication
  - ordonnancement multiversions



5. Techniques avancées

- Transactions longues
  - mise à jour d'objets complexes
  - sessions de conception
- Prise en compte de la sémantique des application
  - opérations commutatives (e.g., ajouts d'informations)
  - essais concurrents
- Travail coopératif
  - modèles concurrents plutôt que séquentiels

Commutativité d'Opérations

- Possibilité de distinguer d'autres opérations que Lire/Ecrire
  - chaque objet possède sa liste d' opérations (méthodes)
  - les opérations commutatives n'entraînent pas de conflits
  - la commutativité peut dépendre du résultat
- Cas des ensembles

	[Ins,ok]	[Del,ok]	[In, true]	[In, False]
[Ins,ok]	1	0	0	0
[Del,ok]	0	1	0	1
[In, true]	0	0	1	1
[In, False]	0	1	1	1

Contrôleur de Commutativité

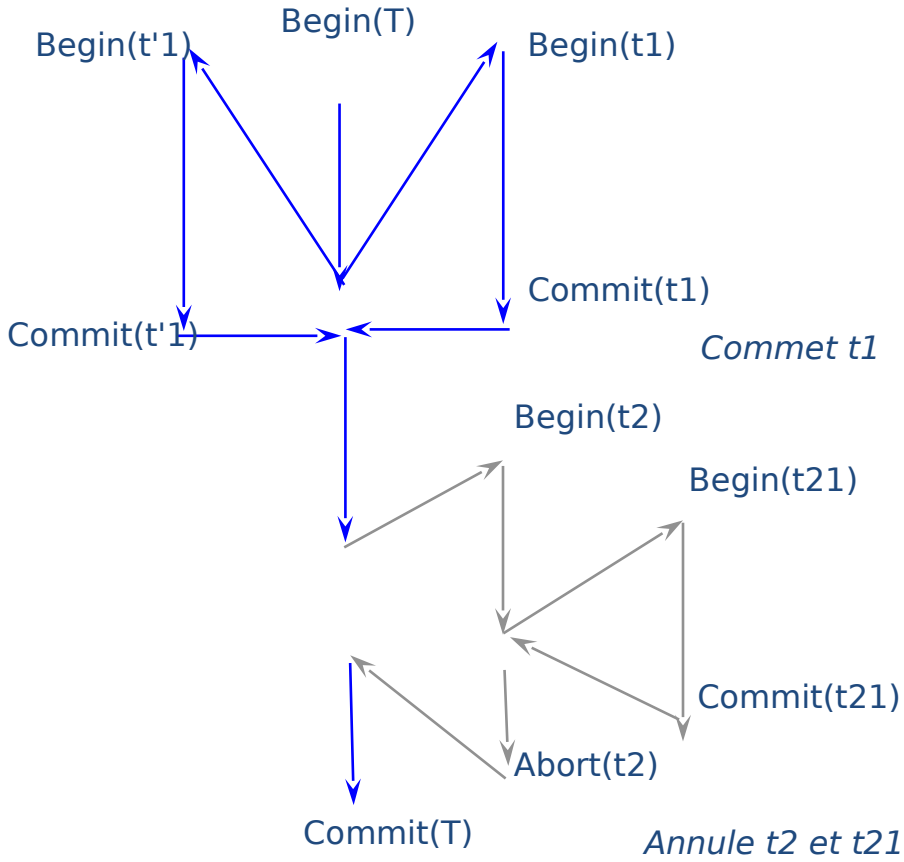
- Chaque objet possède un contrôle de concurrence défini au niveau de la classe
  - laisse passer les opérations commutatives
  - bloque les opérations non commutatives (ordonnancement)

- Le modèle est ouvert et nécessite
  - soit des transactions de compensations
  - soit la gestion de listes de transactions dépendantes

- Un potentiel pour les SGBDO non encore implémenté (complexe)

Transactions Imbriquées

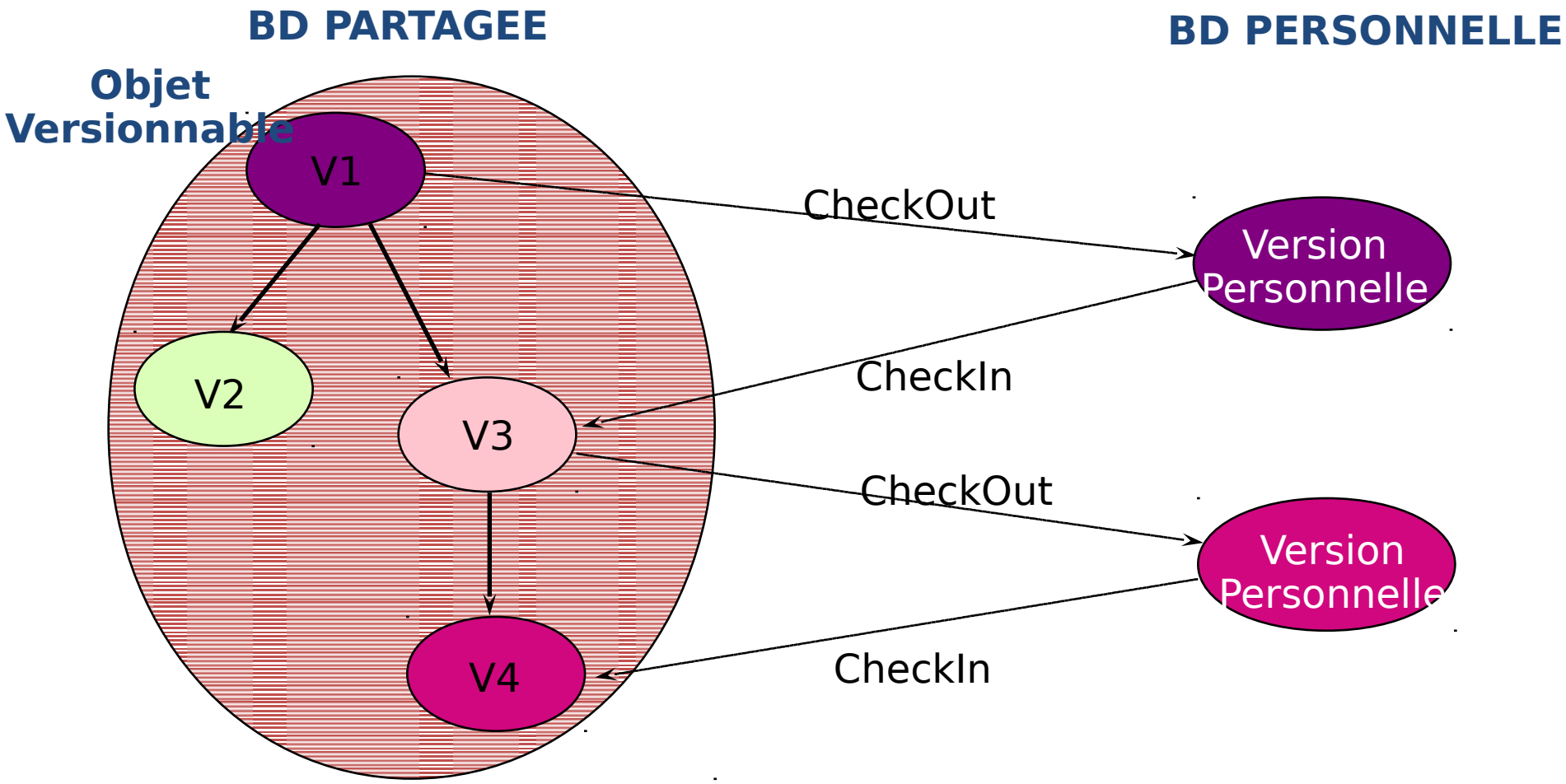
- OBJECTIFS
  - Obtenir un mécanisme de reprise multi-niveaux
  - Permettre de reprendre des parties logiques de transactions
  - Faciliter l'exécution parallèle de sous-transactions
- SCHEMA
  - Reprises et abandons partiels
  - Possibilité d'ordonner ou non les sous-transactions



Verrouillage et Transactions Imbriquées

- Chaque transaction peut acquérir ou relâcher des verrous
- Un verrou est accepté si l'objet est libre ou verrouillé par un ancêtre
- Les verrous non relâchés sont rendus à la transaction mère
- Problèmes de conflits entre sous-transactions parallèles
  - risque de verrous mortels
  - l'ancêtre commun peut trancher
- Gestion de journaux multiniveaux
  - organisation sous forme de piles
  - nécessité de journaux multiples en cas de parallélisme

Versions


BD PARTAGEE
BD PERSONNELLE
Objet Versionnable
V1
V2
V3
V4
CheckOut
CheckIn
CheckOut
CheckIn
Version Personnelle
Version Personnelle

CheckOut / CheckIn

- ChechOut
  - extrait un objet de la BD afin d'en dériver une nouvelle version
- CheckIn
  - réinstalle une nouvelle version de l'objet dans la BD

	Lecture	Ecriture	COut P	COut E
Lecture	1	0	1	0
Ecriture	0	0	0	0
COut Partagée	1	0	1	0
COut Exclusif	0	0	0	0

Fusion des Versions

- Maintient différentiel
  - seuls les objets/pages modifiés sont maintenus
- Pas d'objets communs modifiés
  - la fusion est une union des deltas
- Des objets communs modifiés
  - nécessité d'intervention manuelle (choix)

6. Conclusion

- Amélioration du verrouillage
  - Transactions ouvertes
  - Granularité variable
  - Commutativité des opérations
  - Transactions imbriquées
  - Versions
- Amélioration des modèles transactionnels
  - Transactions imbriquées
  - Sagas, Activités, Versions
- Beaucoup d'idées, peu d' implémentations originales
  - la plupart des systèmes utilise le verrouillage type SQL