SUNGARD

L'informatique qui réinvente la finance

Mapping Objet Relationnel (ORM)

Partie 2 : Principes & Fonctionnement en Ecriture Transactions, Versionning, Session / Cache niveau 2, Locks

Arnaud NAUWYNCK
arnaud.nauwynck@gmail.com

Plan

- Rappel partie 1
  - Architecture, API ORM, ...
- Accès en Ecriture
  - Aspects Transactionnels
  - API JTA, intégration Jdbc + ORM
  - Lock, Versionning, Lock Optimiste
  - Pattern Read-Mostly, Session / Cache Niveau 2
  - Oracle Undo/Redo, isolation no-read-lock

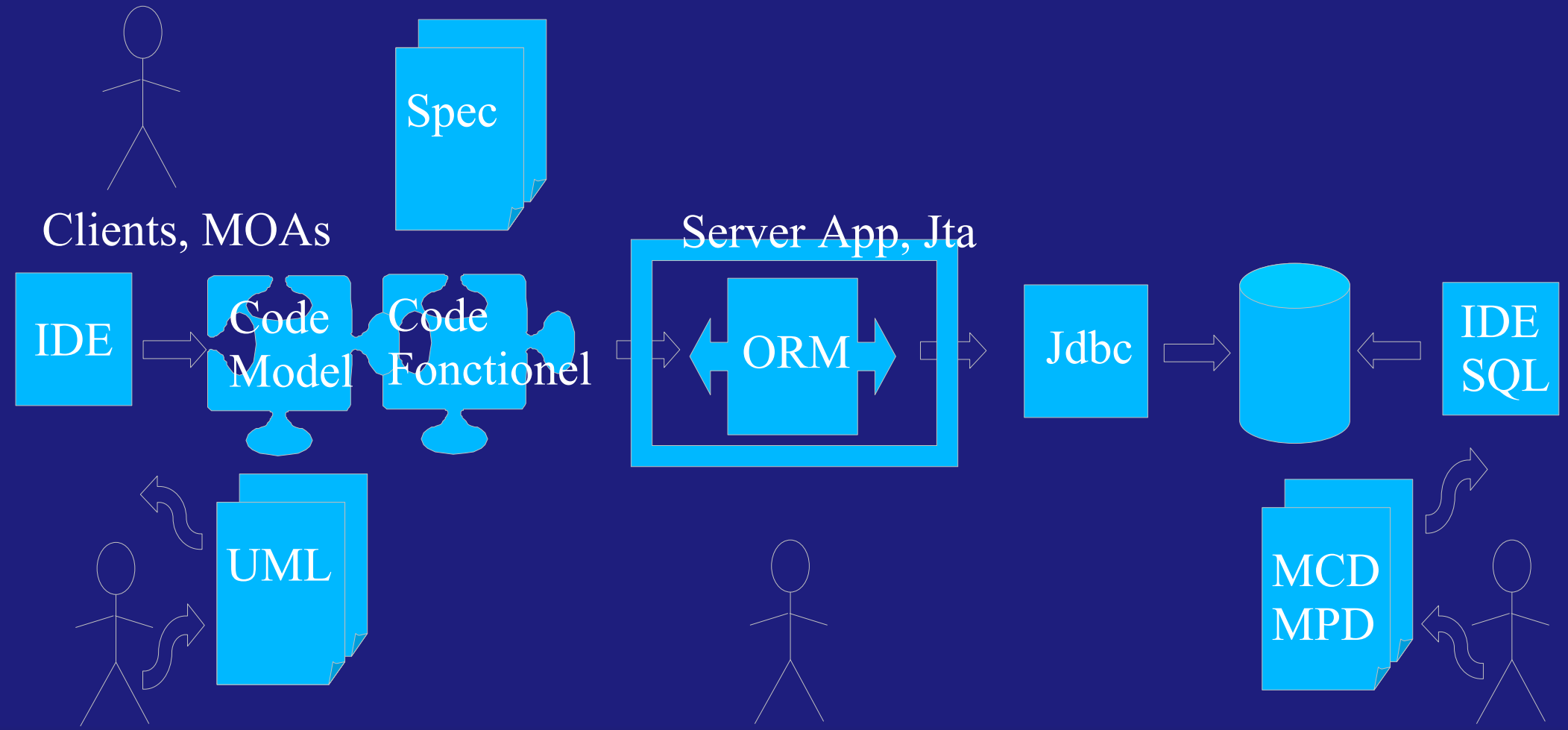
Audience - Prérequis

- PAS un tutorial hibernate/JPA/EJB...
- Pré-requis: connaissances SGBD / persistence
- But: explications avancées
  - Aspects Code => DBA
    (Java, JPA, Serveur Applicatif...)
  - Aspects SGBD => Développeurs
    (Optimisations, Index, spécificités Oracle...)

SUNGARD

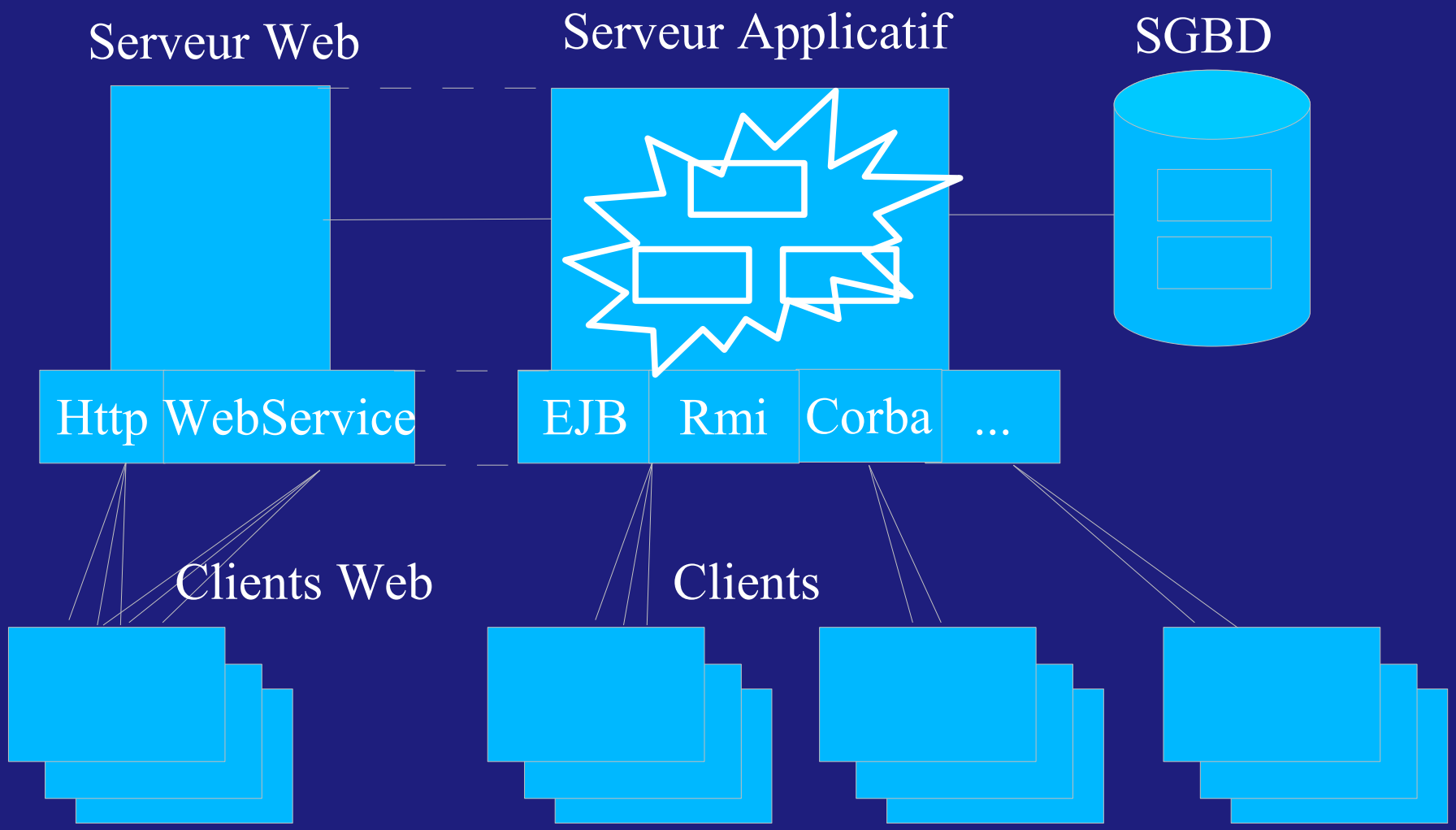
Rappel : Domaine Développeur - DBA

- Répartition des taches, outils ... OK
- Répartition des connaissances, compréhension ?



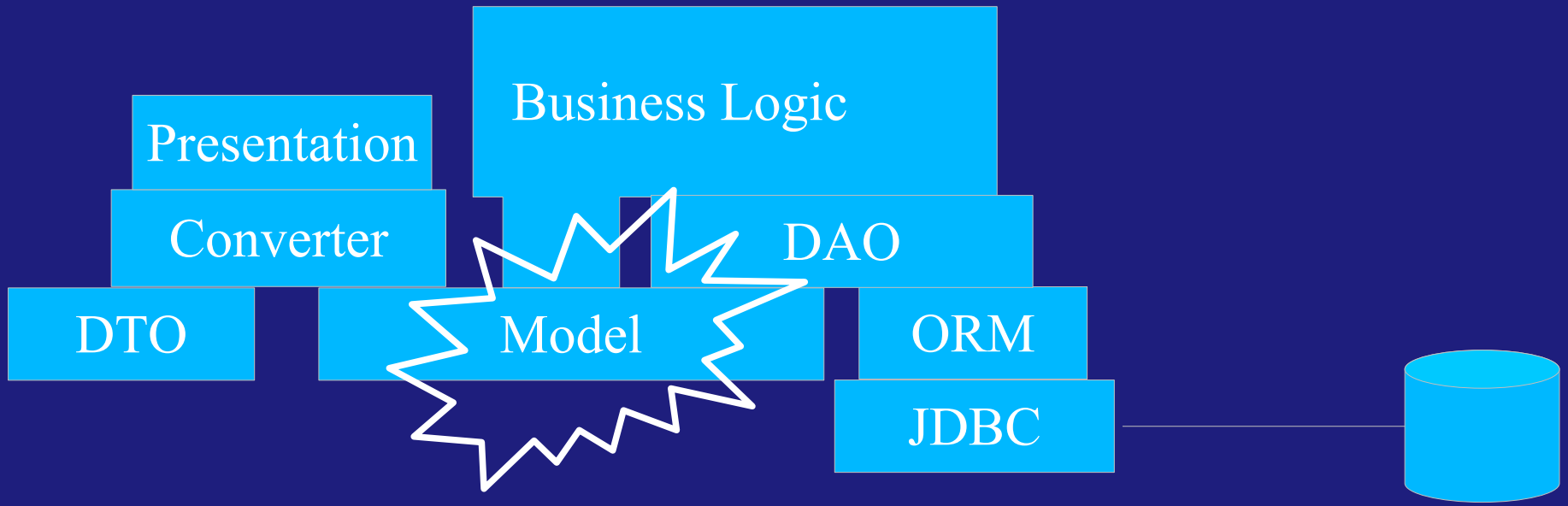
SUNGARD

Rappel : Architecture 3-Tiers



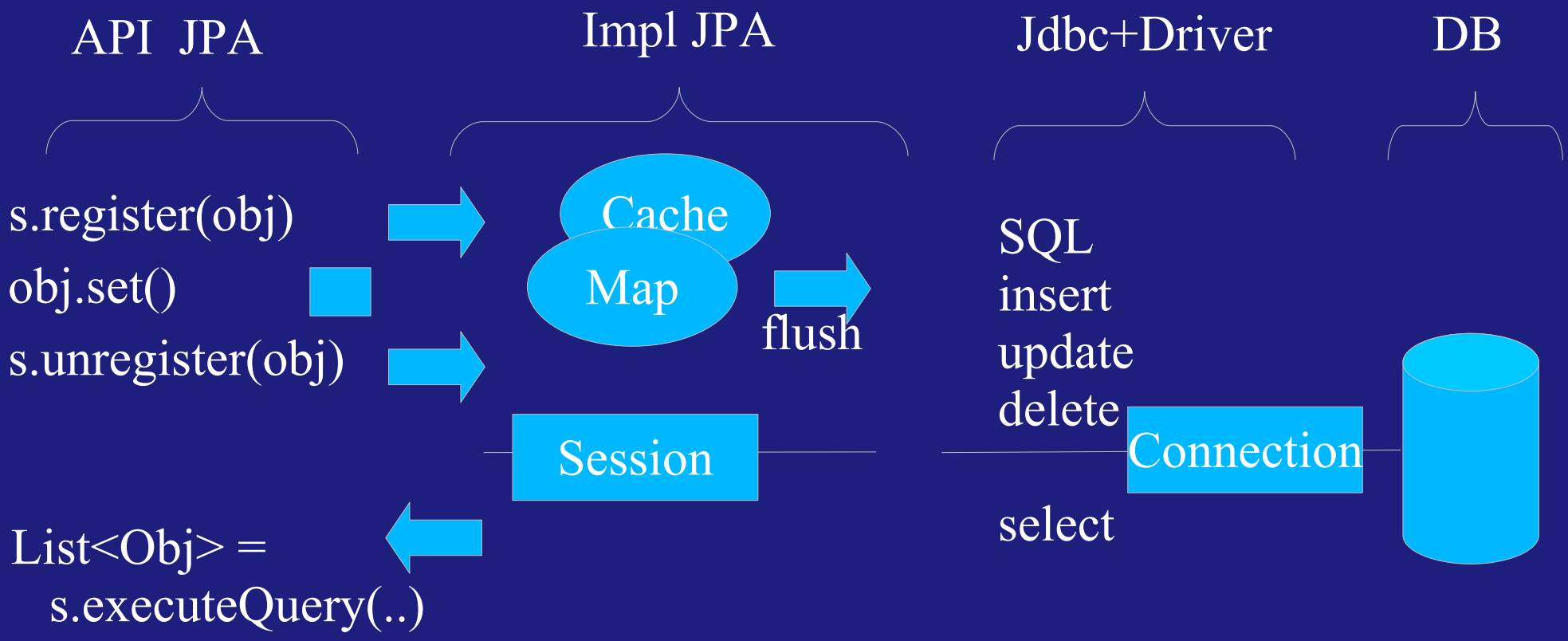
SUNGARD

Rappel : Couches App, DAO, ORM, Jdbc



SUNGARD

Rappel : API JPA, IN/OUT



SUNGARD

RAPPEL : Implémentation DAO avec JPA

```
public class EmpDAO extends DAO {
   // protected Session getSession() ...

    public Emp create(Emp p) {
C                return session.registerObject(p); }

    public Emp findById(EmpPK id) {
R                return (Emp) session.findById(Emp.class, id); }

    // update: simply call setter!
U   // ... emp.setA(..), emp.addB(..)

    public void delete(Emp p) {
D                session.remove(p); }
```

SUNGARD

Approche Naïve Updates DAO CRUD

```
Connection conn = ...


PreparedStatement pstmt = null;
try {
    String sql = “update EMP “
                 +  “set NAME=?, DEPT=?, ...“
                 + “where ID = ?”;
    pstmt = conn.prepareStatement(sql) ;
    pstmt.setInt(1, “Tom”); ... pstmt.setInt(10, id);
    int c = pstmt.executeUpdate();
    if (c != 1) throw new RuntimeException();
} finally { safeClose(pstmt); }
```

SUNGARD

Problèmatique Updates CRUD

1) Aspects Transactionnels
   ... non spécifiques au CRUD, voir suite XA = ACID

2) Update de tous les champs
   ... peu efficace, et risqué !
    => besoin d'un update “incrémental”
       via champs oldValue / dirtyFlag

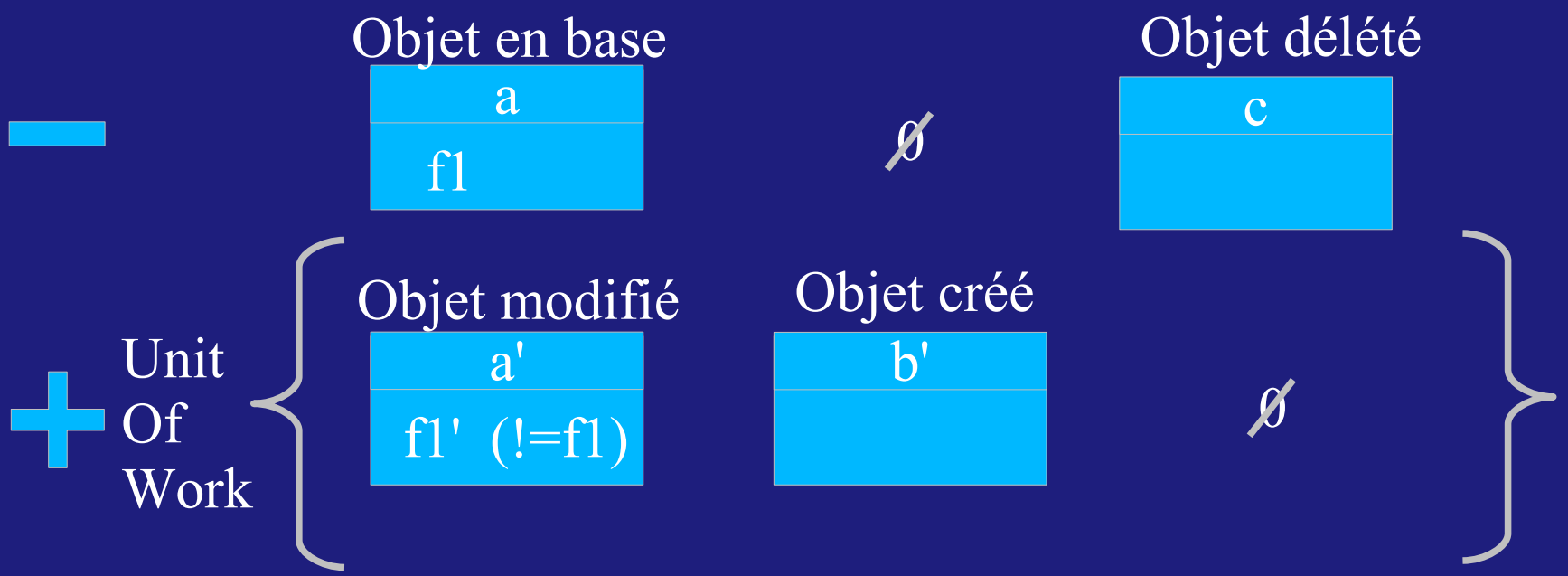
3) Déclenchement des appels
   ... juste après chaque setter ?
    => besoin de “flusher les changements 1 fois à la fin”

4) Problèmes concurrentiels, Isolation, “LostUpdate” ...
   ... besoin de locks / versionning

SUNGARD

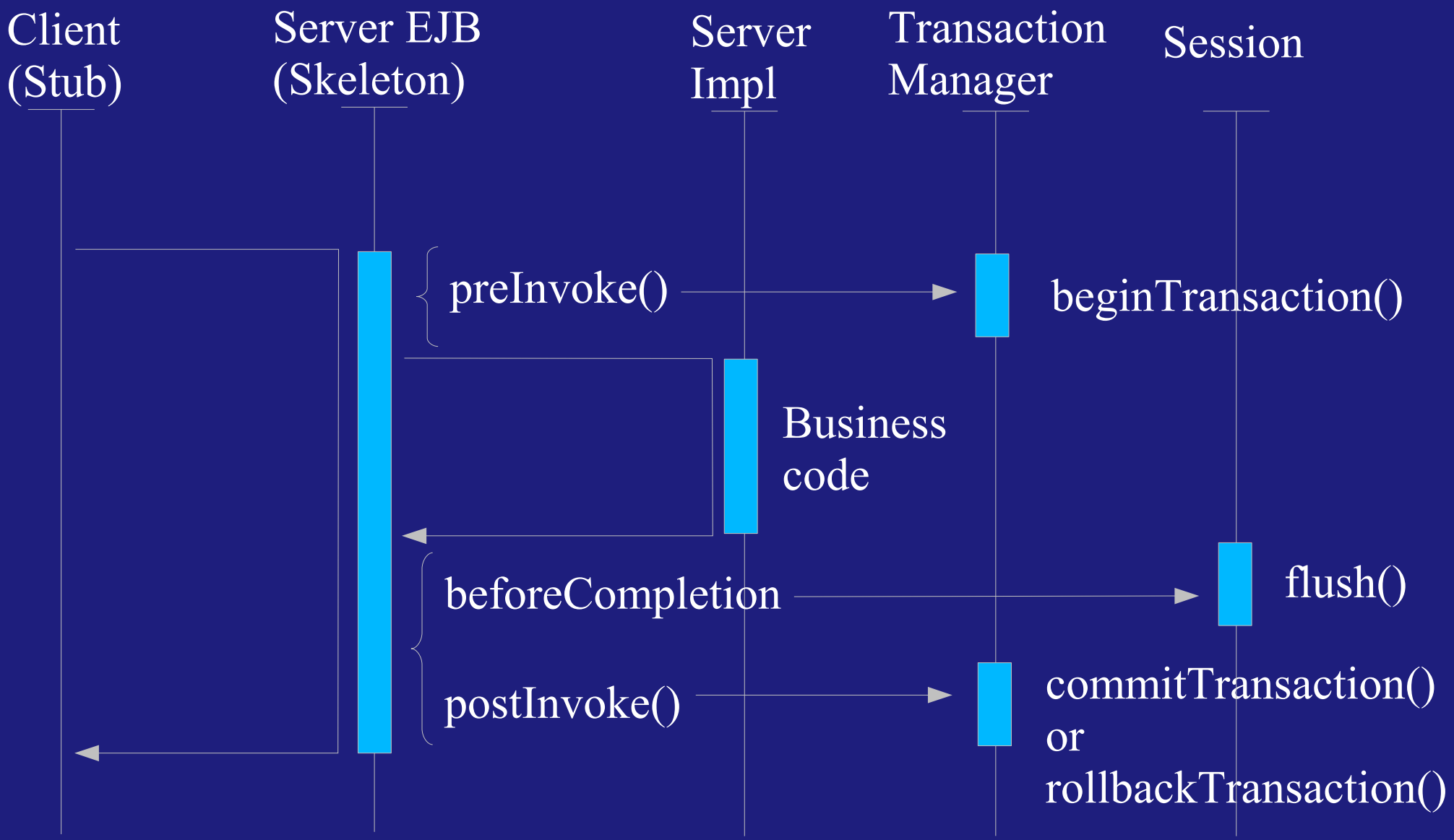
Update (Diff) Incrémental, UOW

- Diff = After – Before
  - After : in “Session”, “UnitOfWork”, “WorkingCopy”
  - Before : as Init / as in Cache / as in DB ... ??



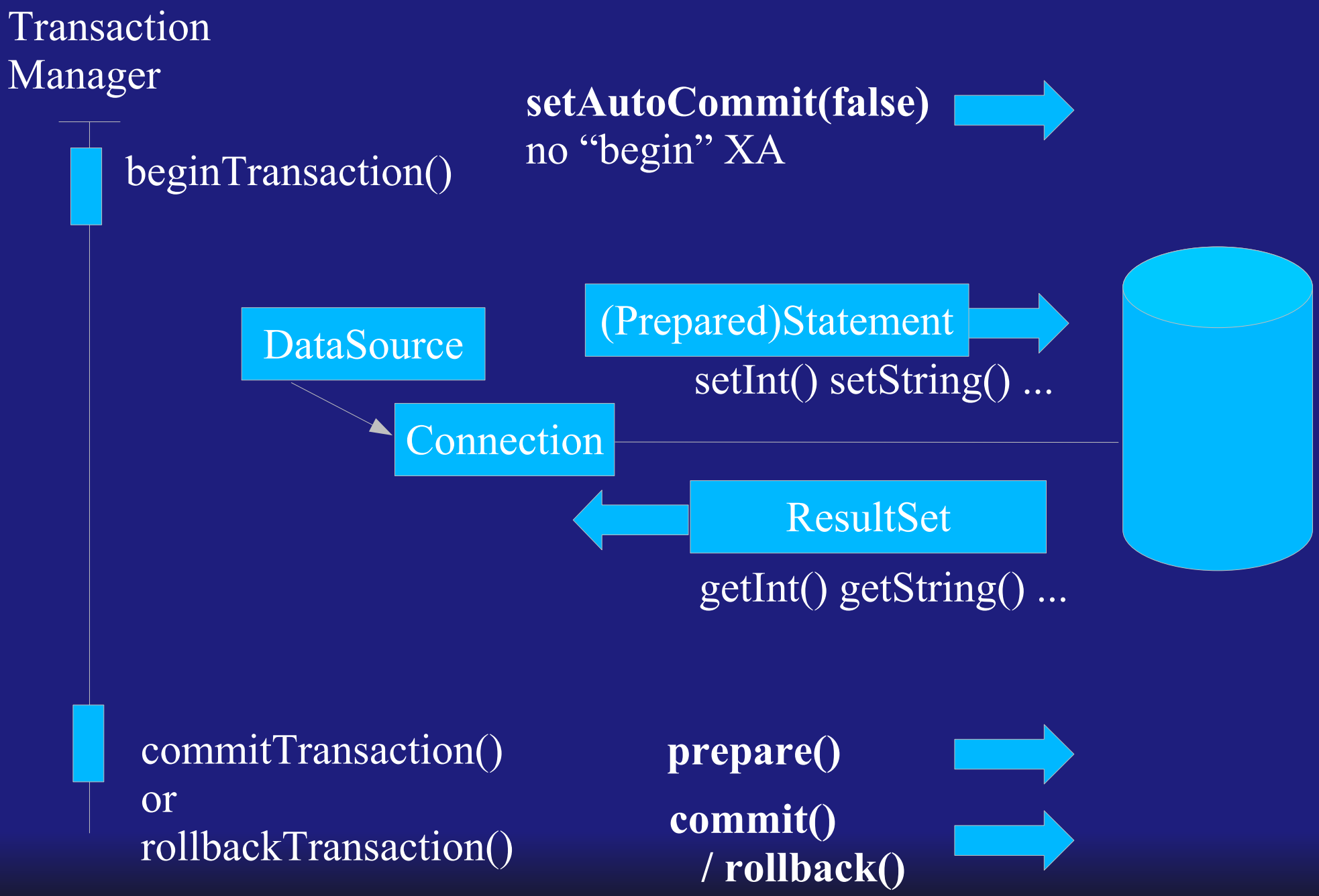
SUNGARD

Exemple Code Pre-Post pour XA



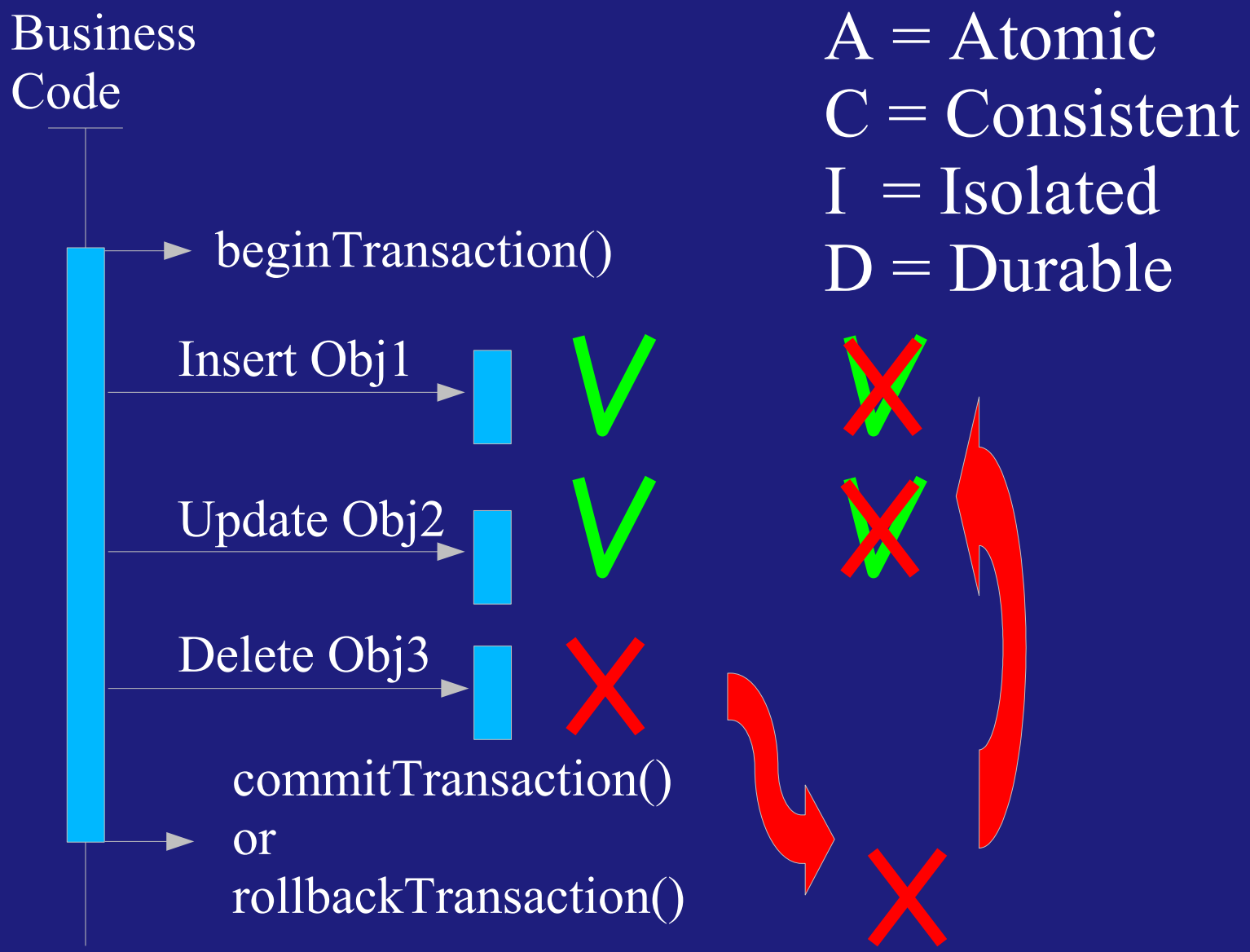
SUNGARD

Aspect Transactionnel dans JDBC



SUNGARD

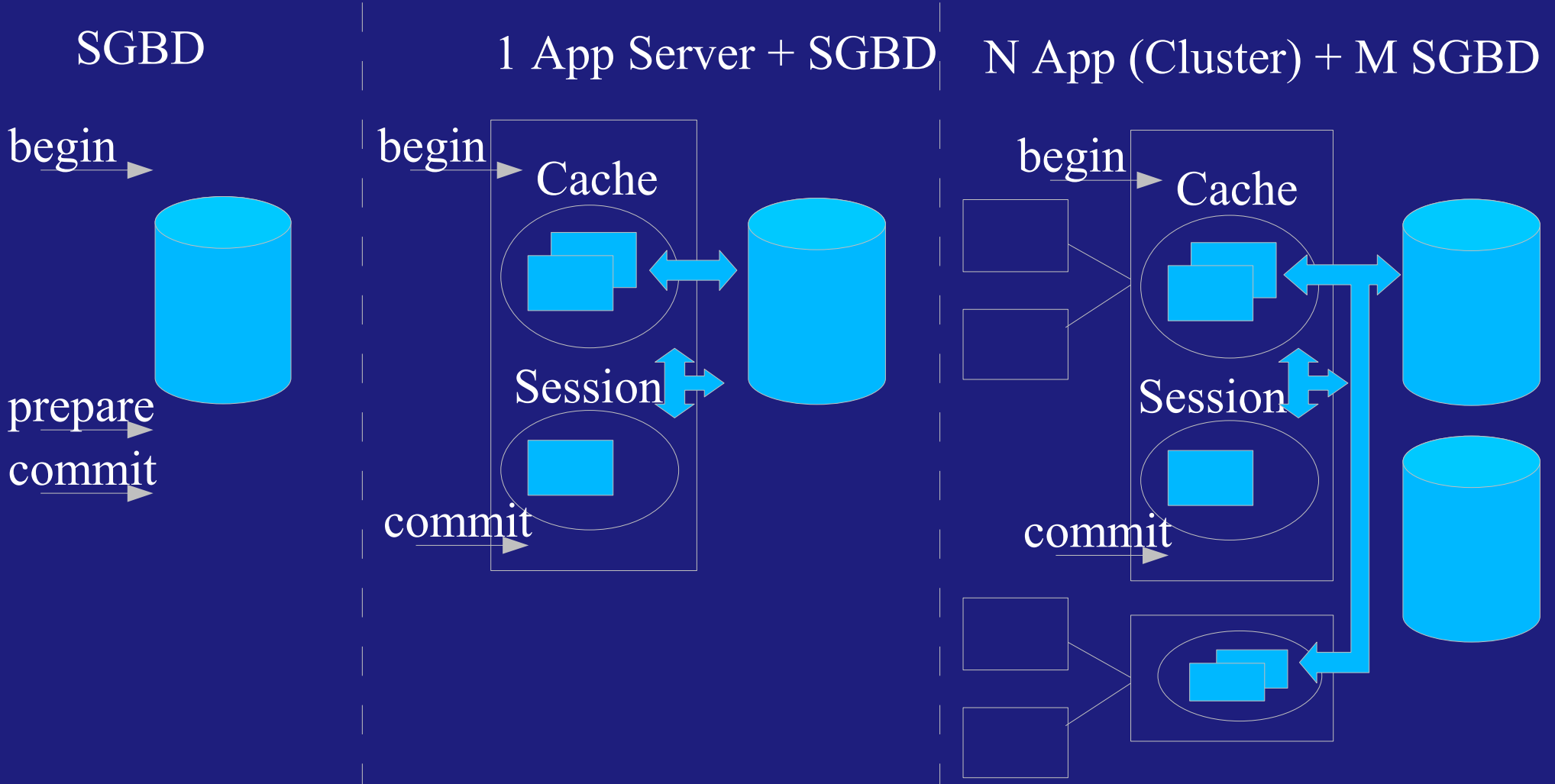
XA = ACID



A = Atomic
C = Consistent
I = Isolated
D = Durable

SUNGARD

Problématique ACID x 3 = SGBD + AppServer + Client-ClusterServer...



XA enabled?
Vraiment ACID ?
... voir suite (1)

XA via Cache ?
... voir suite (2)

XA via ClusterCache ?
... voir suite (3)

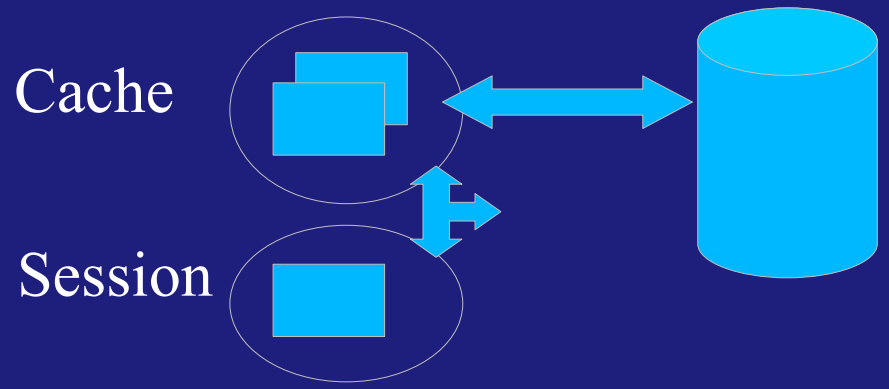
Anomalies ACID d'un SGBD

- A et I .... impossible sauf si “Serialized”
  ... = mono cpu = non performant, non scalable!
- En pratique : classification des “défauts”
  - Dirty read
  - Read committed
  - Read Phantom
  - Read consistent
  - Lost Update
  - Read lock , write lock
  - Dead-lock watchdog... DeadLockVictimException

SUNGARD

Anomalies ACID d' 1 ORM + SGBD

- Updates Incrémentales via Session.. via Cache
  - Si Cache FAUX => update FAUX !

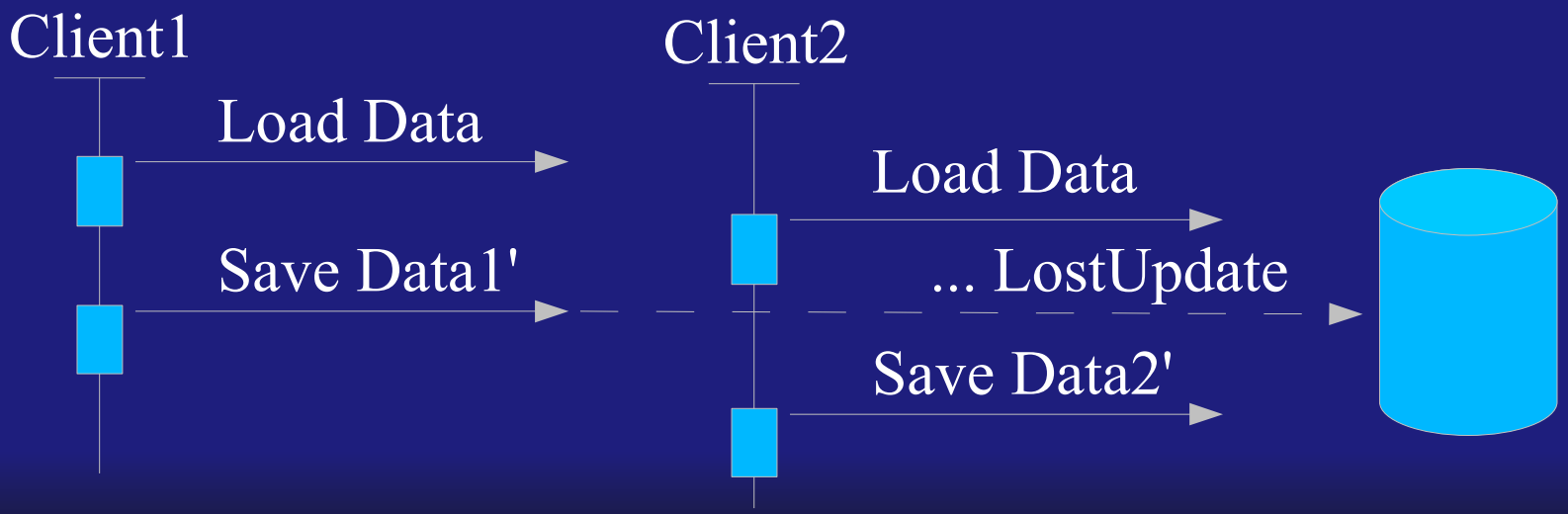


- Problèmes Synchronization Cache vs SGBD
  - Setter sur objets du cache => CORRUPTION!
  - Updates directes db => DE-SYNCHRONIZATION!
  - Read XA / Dirty / phantom / Rollback ?
  - Commit => commit “putObjectsInCache” XA

SUNGARD

Anomalies ACID de N Apps + SGBD

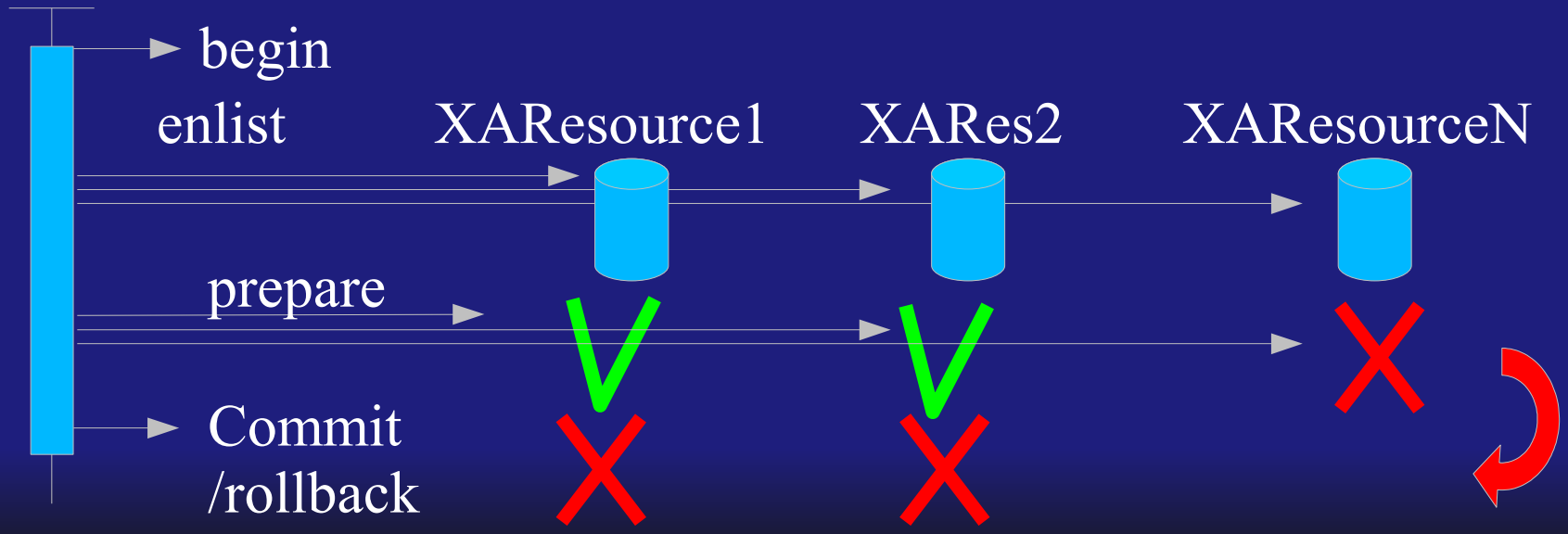
- Problèmes de réplication de N caches
  - Synchrones et XA ?
- Problèmes N Clients-Server(s)
  - LostUpdate, OptimisticLockException ...
  - Effet cache via fenètres d'éditions :
    Server->Client -> ... edit ... -> Client->Server



SUNGARD

Anomalies ACID de 1 App + N SGBDs

- N Resources à commiter dans 1 transaction...
  - Ex: 2 bases (transfert argent...), JMS, Mail...
  - => utilisation 2-phases-commit
  - Au plus 1 resource non XA
    ... en général SGBD non XA (optionnel) !!!
  - HeuristicLockException si pb...

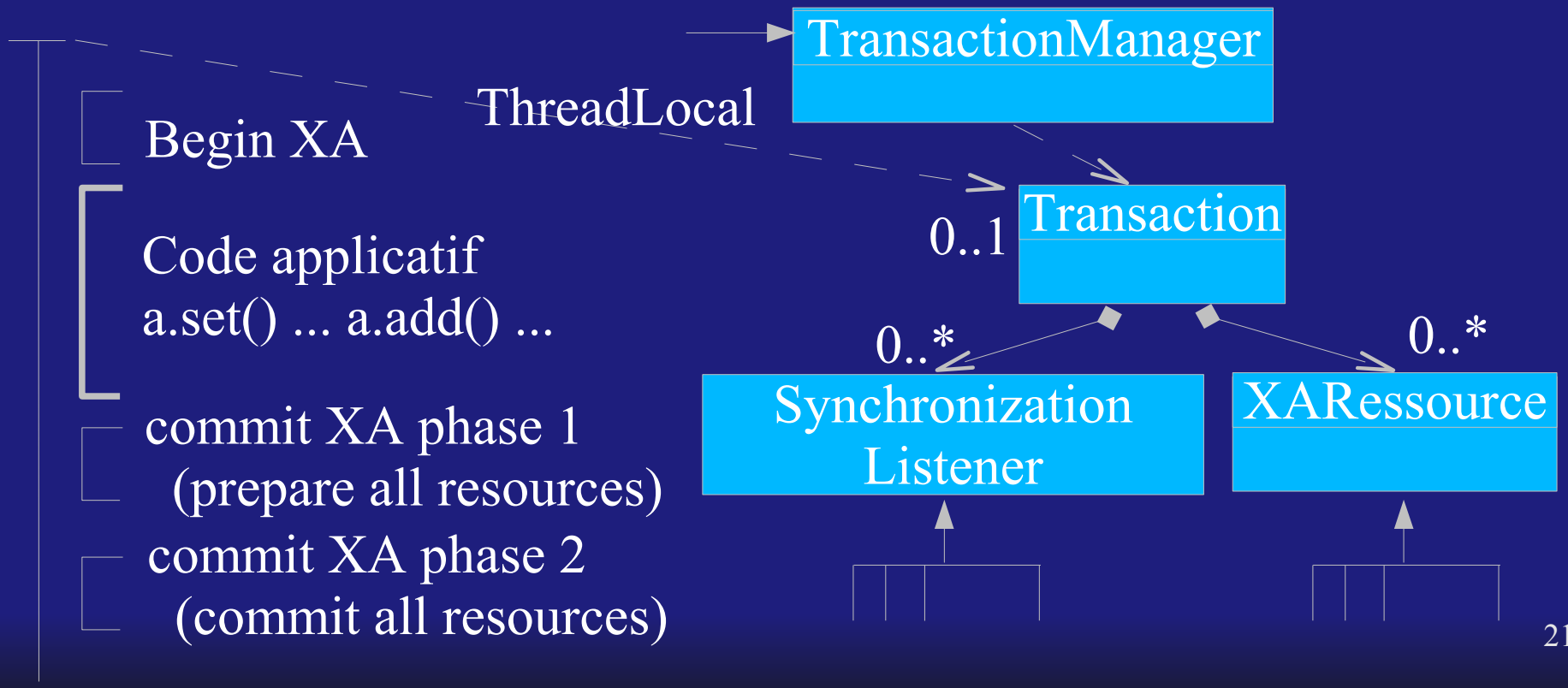


Plan

- Introduction, Rappel
- Aspects Transactionnels
  - JTA, intégration JDBC + ORM
- Versionning, Lock Optimiste
- Pattern Read-Mostly, Cache Niveau 1 & 2
- Oracle Undo/Redo, isolation no-read-lock
- Lock Base de Données / Oracle

JTA : Transaction API

- Singleton TransactionManager
  - XA rattaché au Thread courant
  - Thread -> XA -> XAResource -> resource
- Spring : “Not invented here” syndrome !!



SUNGARD

Container Managed XA, Démarcation

- Attributs Transactionnels: annotés sur EJB
- 1+4 possibilités:
  - **Required** (le plus courant)
  - Supports .... si parent non XA => non XA !
  - Mandatory ... couche interne seulement
  - NotSupported ... pas XA !
  - RequiresNew ... pour log, audit
- Supports/NotSupported est suspecte
  ... souvent: workaround pour Timeout JTA!

SUNGARD

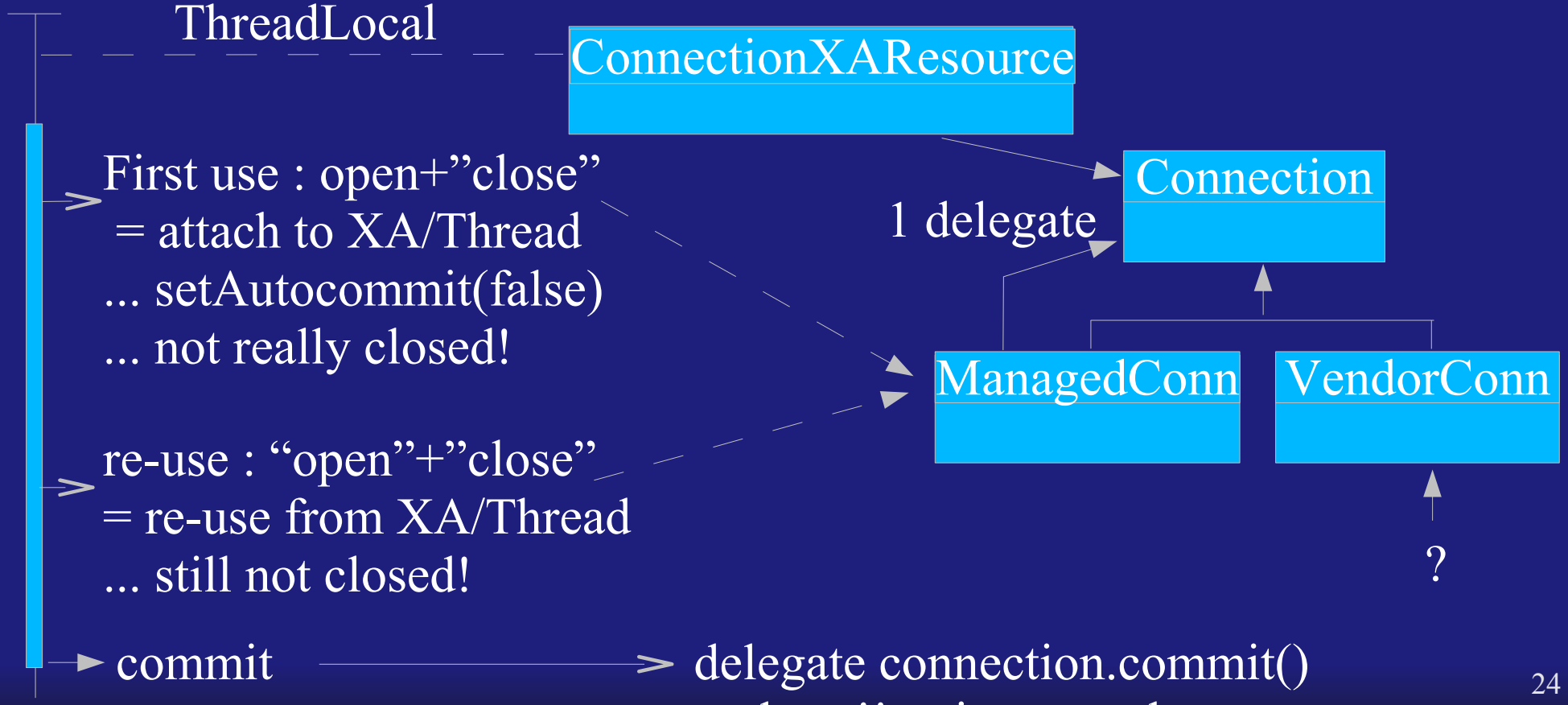
Intégration JDBC dans JTA

- Connection JDBC = XARessource
  - "begin tran" .... implicit!
  - **"setAutocommit(false)"** ... 1 fois init pool
  - "**setAutocommit(**true)" ... suspect
    - Supports / NotSupported au lieu de Required!
  - "prepare()"
    - ... seulement si mode 2-Phases commit (plusieurs bases)!
  - **"commit()"**
- Confusion nommage:
  - Session Oracle = Connection... XA Oracle != JTA

SUNGARD

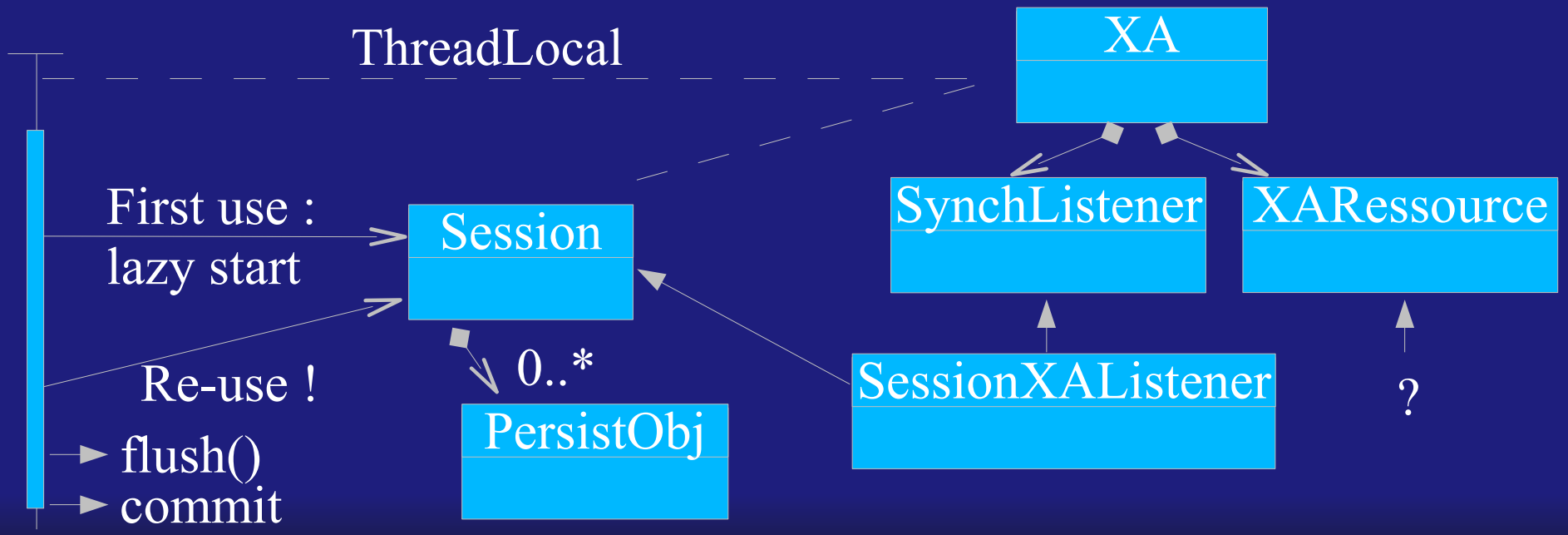
Connection.close() ? .... commit()

- code de connection jdbc: toujours
  try { ... } finally { close(); }
- Connection wrappées !! close()= noop



Intégration ORM dans JTA - XAResource

- Session = XAResource ?
  - Automatiquement démarré + rattaché au XA
  - ordre resource important :
    flush() jdbc before jdbc commit !
  - Use SynchronizationListener.beforeCompletion()



Plan

- Introduction, Rappel
- Aspects Transactionnels
- Versionning, Lock Optimiste
- Pattern Read-Mostly, Cache Niveau 1 & 2
- Oracle Undo/Redo, isolation no-read-lock
- Lock Base de Données / Oracle

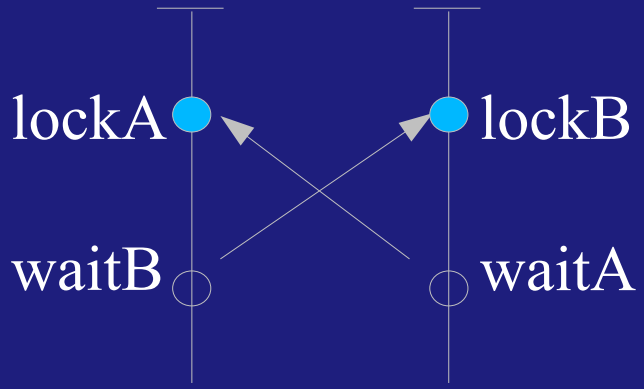
Méchanismes de Locks

- But:
  - locké les objets avants les updates
  - Maintenir la cohérence des groupes d'objets
  - ACID : Atomic, Consistent, Isolated, Durable
  - Code multi-thread safe
- Ex:

```
lock()
try {
   obj1.set().. obj2.set() ...
} finally {
   unlock();
}
```

Lock ... Dead-Locks

- Use Lock => get Dead-Locks...



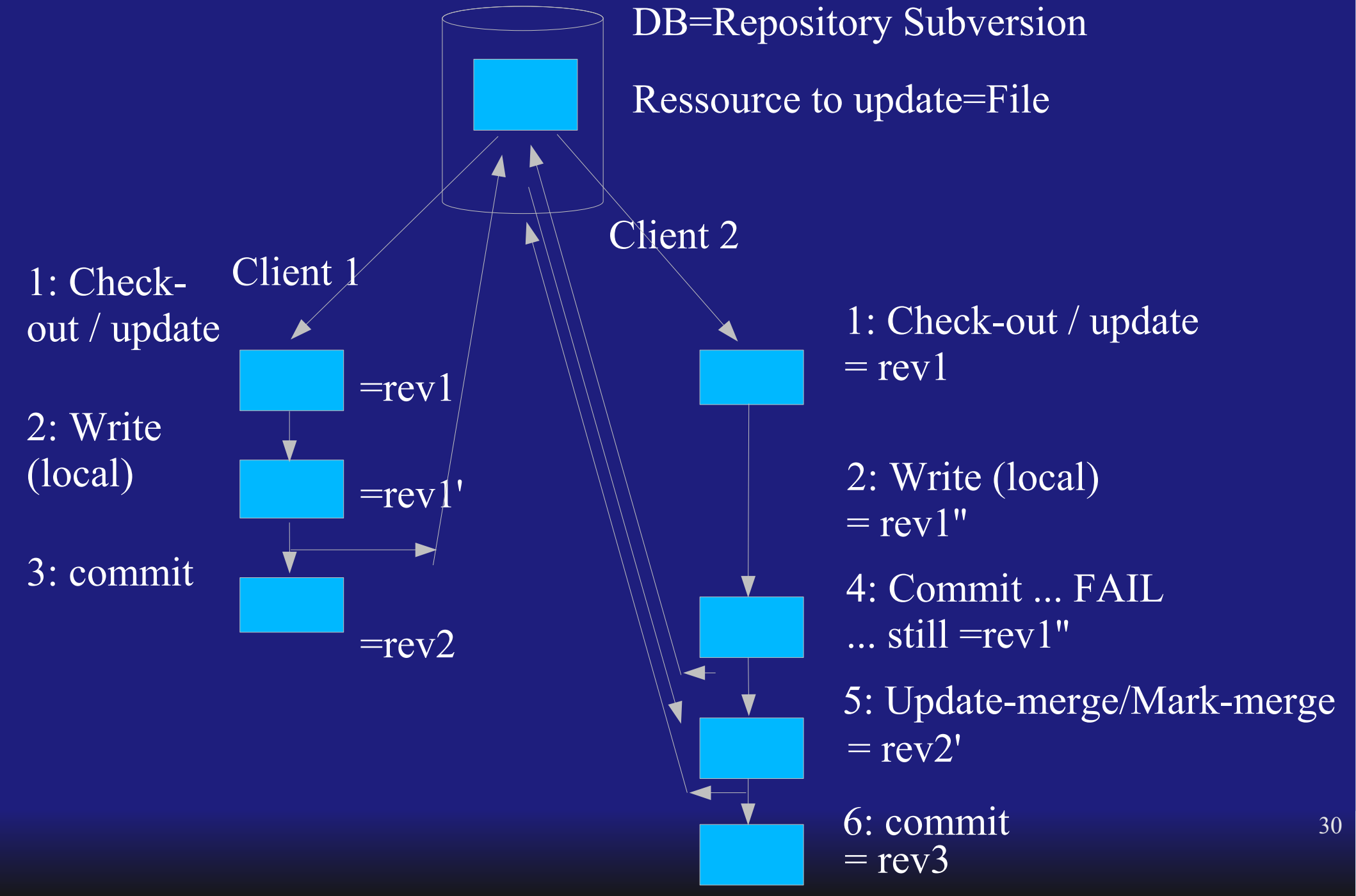
- Code J2E correct: aucun “synchronized”
  - Code client: inutile
    - Swing = “Single Threaded Appartment Model”
  - Code Serveur:
    - pool, queue, jms, ... déjà codés, EJB/JPA sans locks
    - Pattern Read-Mostly + Lock Optimiste ... cf next

Lock Optimiste

- "**Lock Optimiste**"
  = FAUX-AMIS : version pour merge (!= lock)
  - On laisse faire, jusqu'à la base qui gèrera
  - D'où le nom "optimiste"
- Versus "**Lock Pessimiste**"
  = lock (classique) exclusif
  - ... pour empécher l'avancement
- Analogie : Subversion (versus SourceSafe)
  - Svn update / svn merge / svn commit

SUNGARD

Lock Optimiste : Update/Merge/Commit



LockOptimiste en JDBC

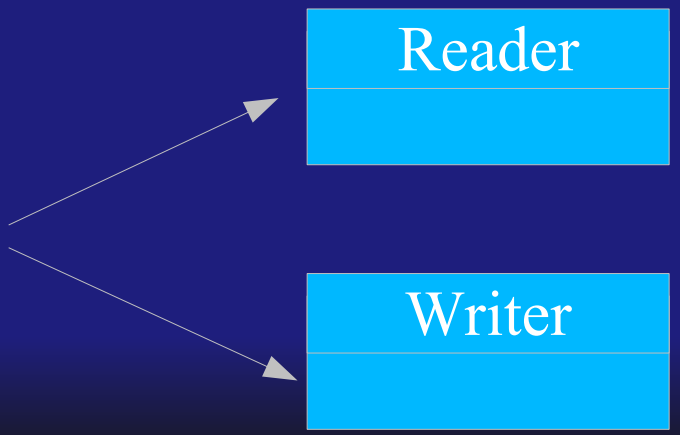
- 1) “select ID,**VERSION**, ... from T where ...”
- 2) “update T
  set VERSION=VERSION+1, A=?, B=?....
  where ID=?
  and **VERSION=?**”
- 3) check update
  - **int rowCount = stmt.executeUpdate(...)**
    // ou stmt.getUpdateCount())
    // en Sql: %rowcount
  - **If (1 != rowCount) {**
    throw new OptimisticLockException();

Plan

- Introduction, Rappel
- Aspects Transactionnels
- API JTA, Intégration Jdbc + ORM
- Lock, Versionning, Lock Optimiste
- Pattern Read-Mostly, Cache Niveau 1 & 2
- Oracle Undo/Redo, isolation no-read-lock

Design Pattern Read-Mostly

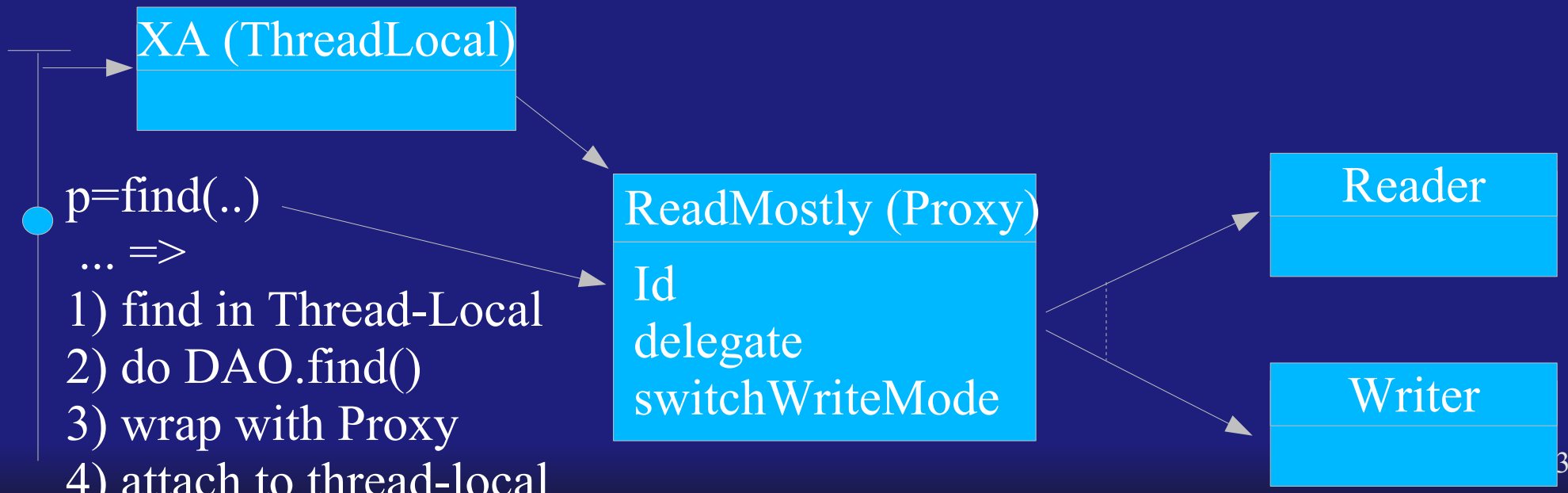
- But :
  - **Lecture SANS lock**, sur objet multi-threadé
  - **Ecriture**, en thread-local (XA) “WorkingCopy”
    ... **reader != writer**
- OK pour lectures... mais
  - Code dupliqué avec signatures différentes!
  - après “writer.set()” ...
    - writer.get() : correct
    - reader.get() : FAUX



SUNGARD

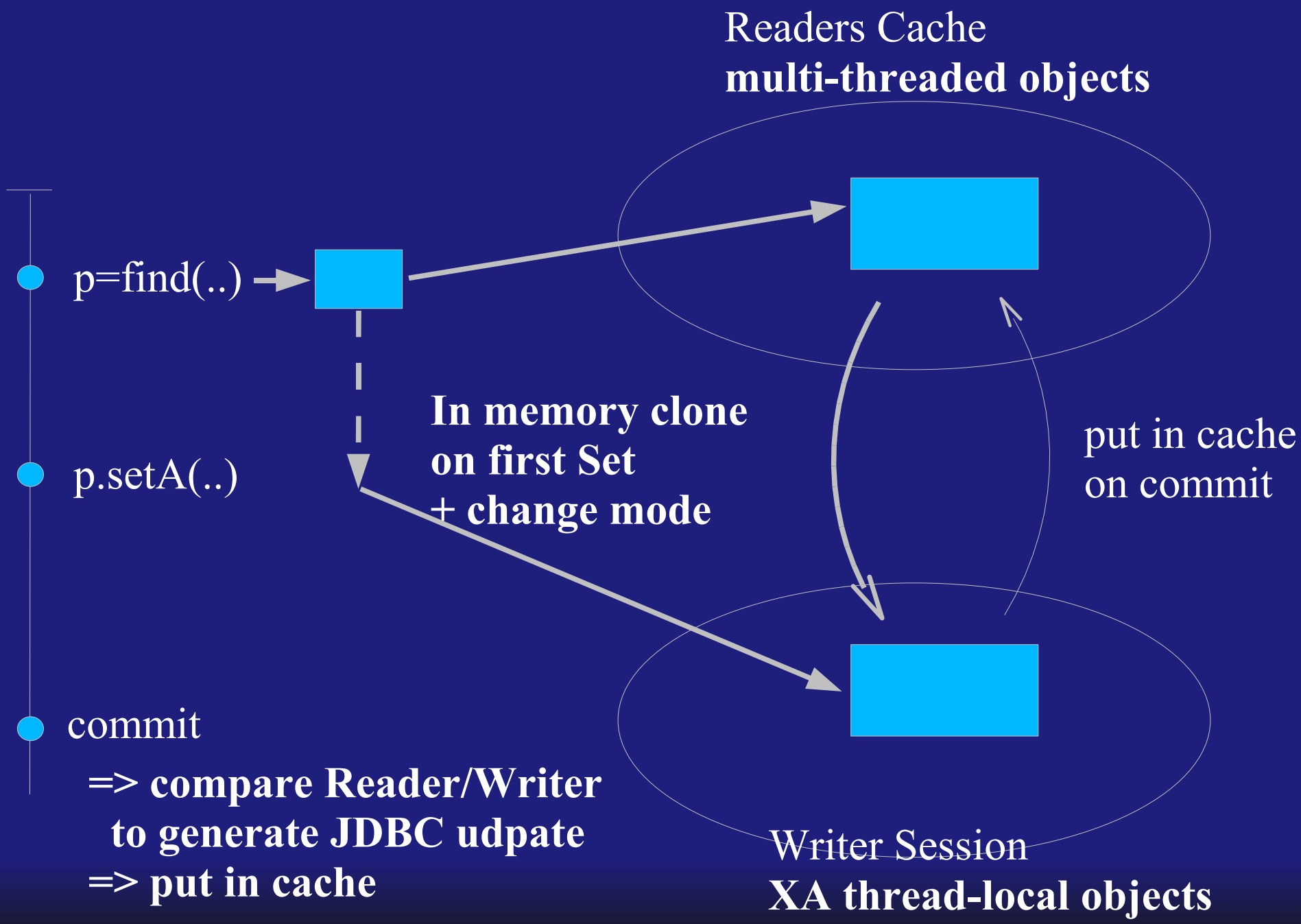
Read-Mostly Proxy (Transparent)

- Proxy = API facade simple
  - Implémentation via reader + writer + writeMode
  - WriteMode = état du switch reader/writer
  - Read consistent: re-find => retourne même proxy
  - rattaché au thread (XA)



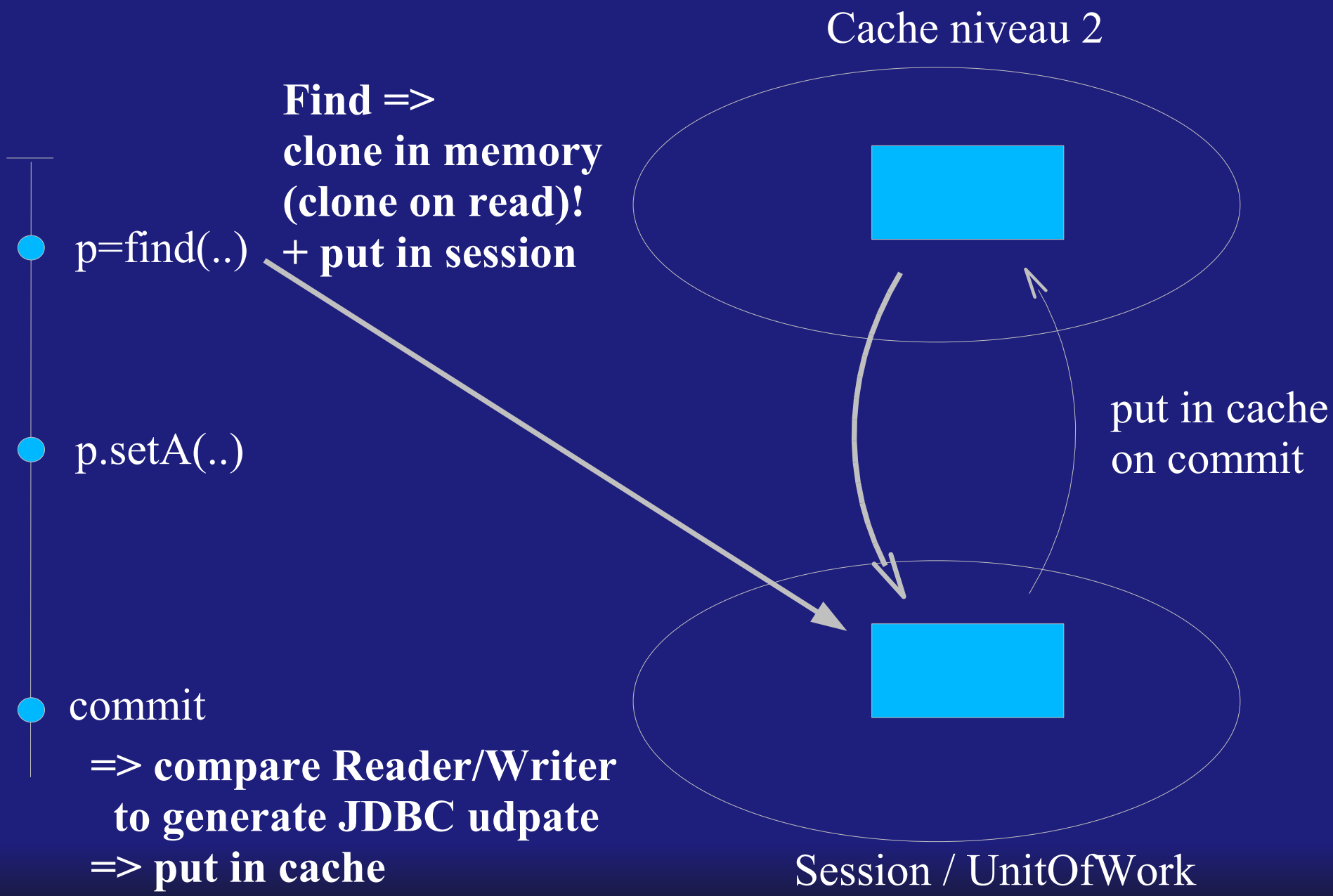
SUNGARD

Pattern Read-Mostly



SUNGARD

Sans Proxy ... Clone on Read!

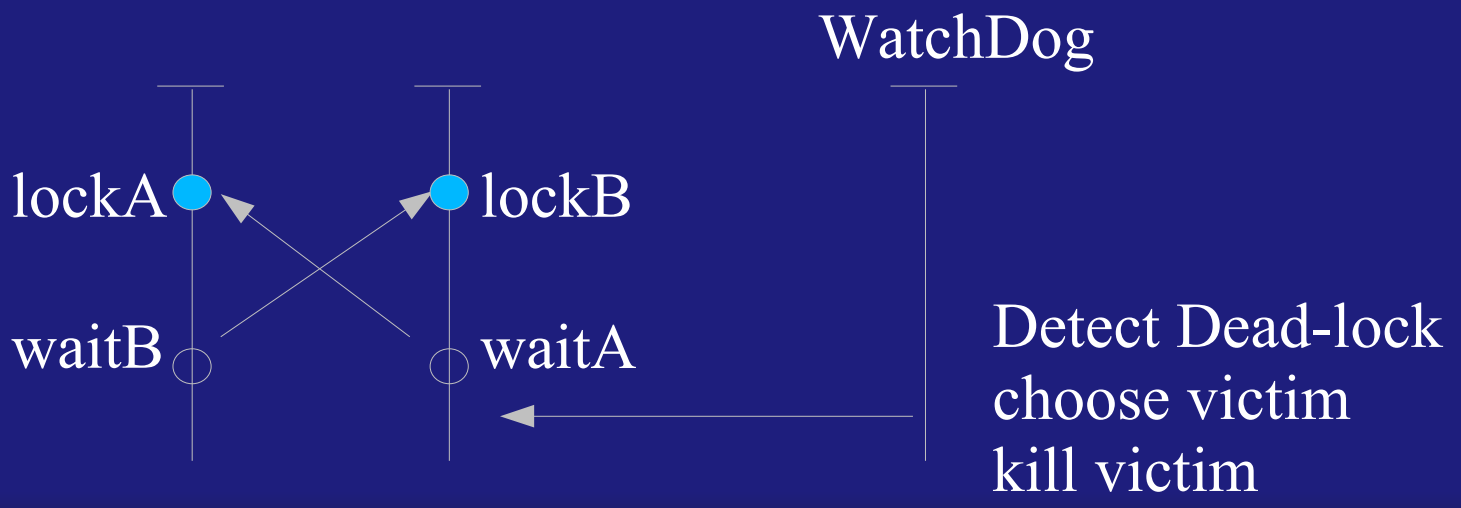


Plan

- Introduction, Rappel
- Aspects Transactionnels
- API JTA, Intégration Jdbc + ORM
- Lock, Versionning, Lock Optimiste
- Pattern Read-Mostly, Cache Niveau 1 & 2
- Oracle Undo/Redo, isolation no-read-lock

Lock SGBD Classiques

- **Read-locks** (read consistent) : lock ré-entrant
- **Write-Locks** (on update)
  - exclusif, efficace en mémoire, mais problématique...
- **WatchDog** = détecte les dead-locks
- DeadLockVictimException... ok : réessayer 3x !!



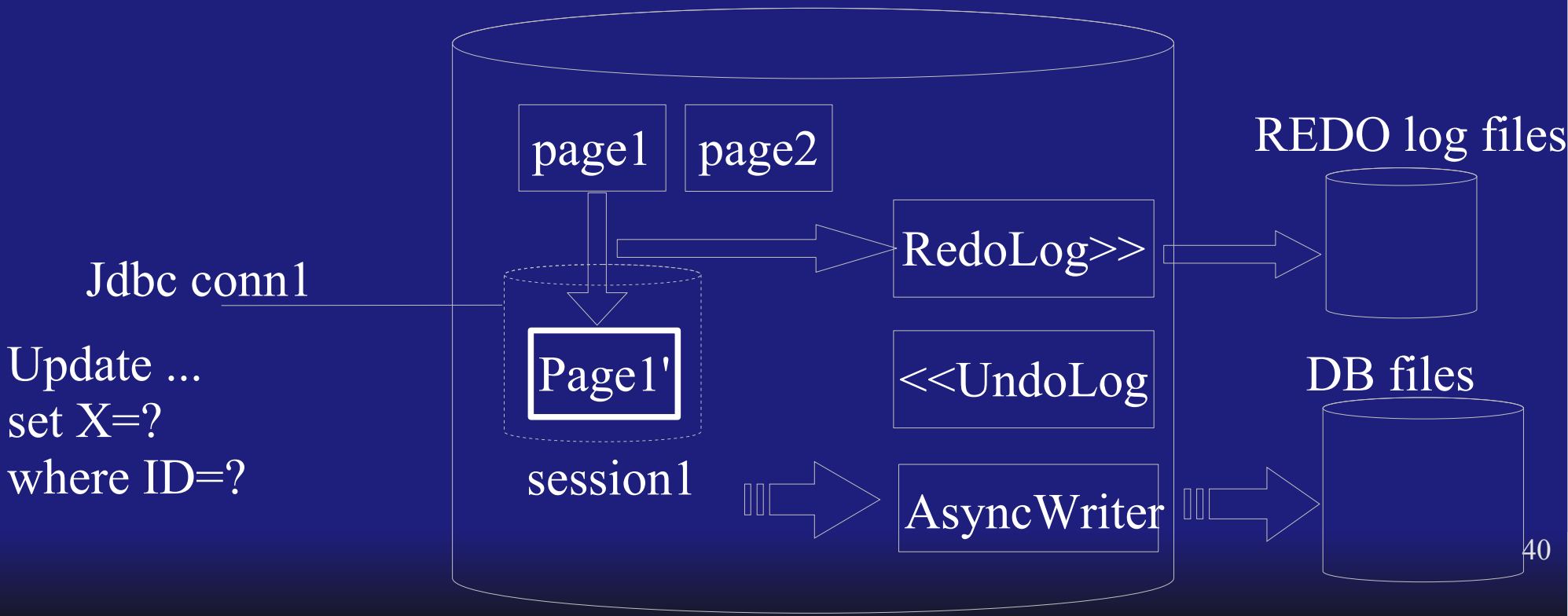
Oracle ... NEVER Read-Lock

- Oracle est différent de la plupart des SGBD !
  Jamais, Jamais de read-lock ....
- En interne:
  - pattern read-mostly : “clone on write” pages
  - historique pour “regénérer” les pages non modifiées
  - relire sans être locké par des writes!
    - “Read consistent” = read as begin tran
    - “Read flashback” = “as of date” = read dans le passé
  - voir page suivante : “UNDO Log”
- ...symmetric “REDO log” = journal XA classique

SUNGARD

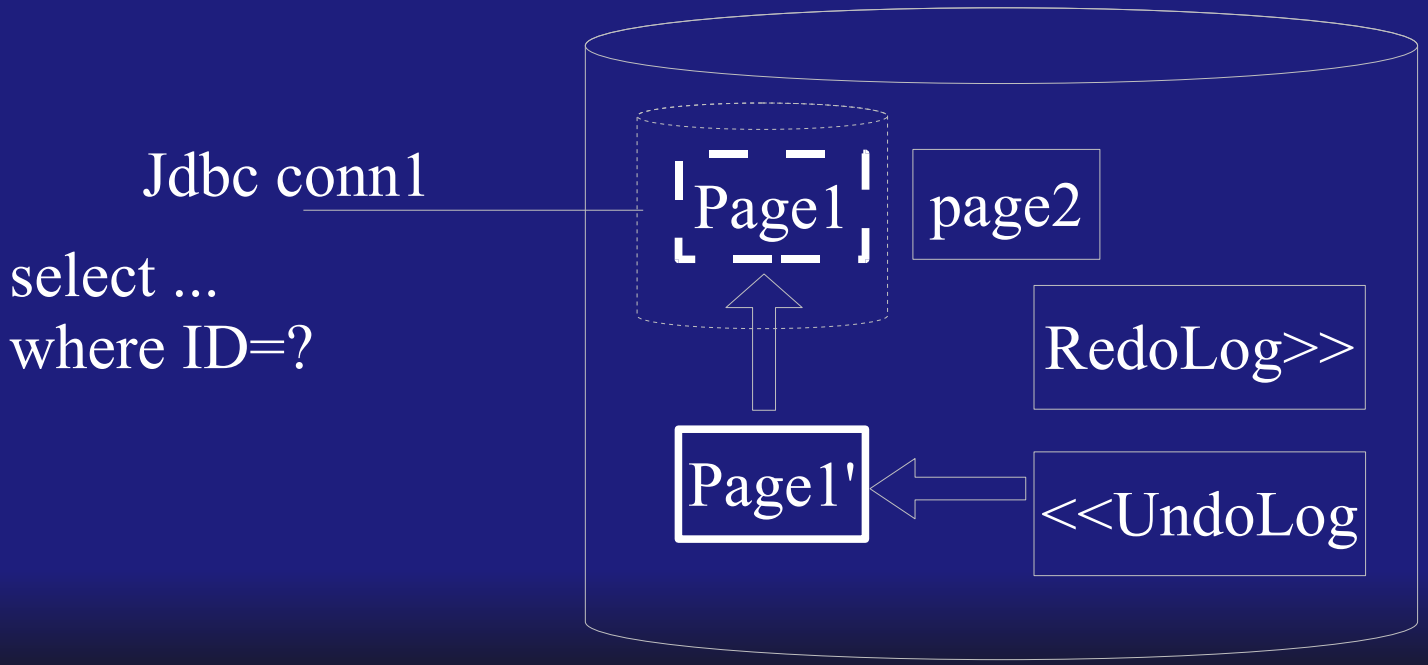
Oracle REDO Log

- Chaque Session ne voit que ses Updates
  - Principe ACID: I=Isolé! ... **Copy-on-write blocks**
- "**Redo log**" = journal xa: Flush on commit
- DBWriter: sauvegarde pages asynchrone



Oracle Reads : UNDO Log

- **Read Consistent =**
  - **Lecture répétable, sans lock**
  - **Mieux encore: comme au début de la transaction**
- **"UNDO Log" : pour** re-générer les pages
  - Read Flashback (read in past)



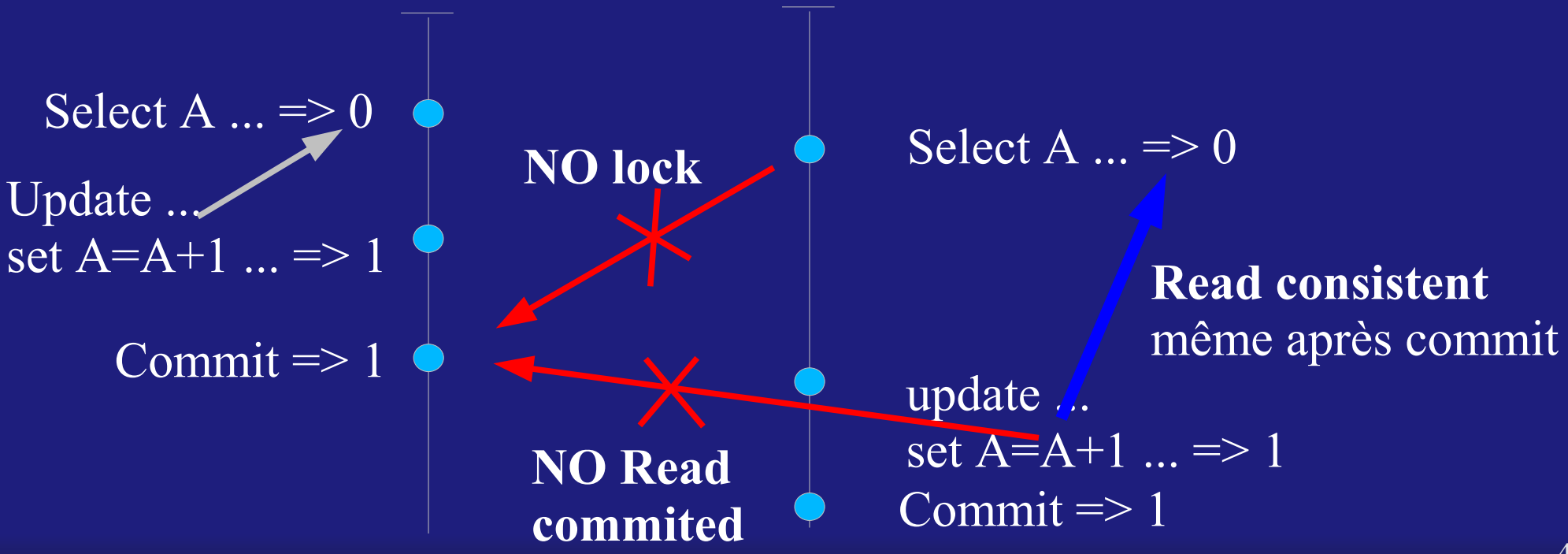
Conséquence des Read Consistent

- Attention

  SQL “correct” en Sybase ... dead-lock

  = “buggué” dans Oracle ... sans lock

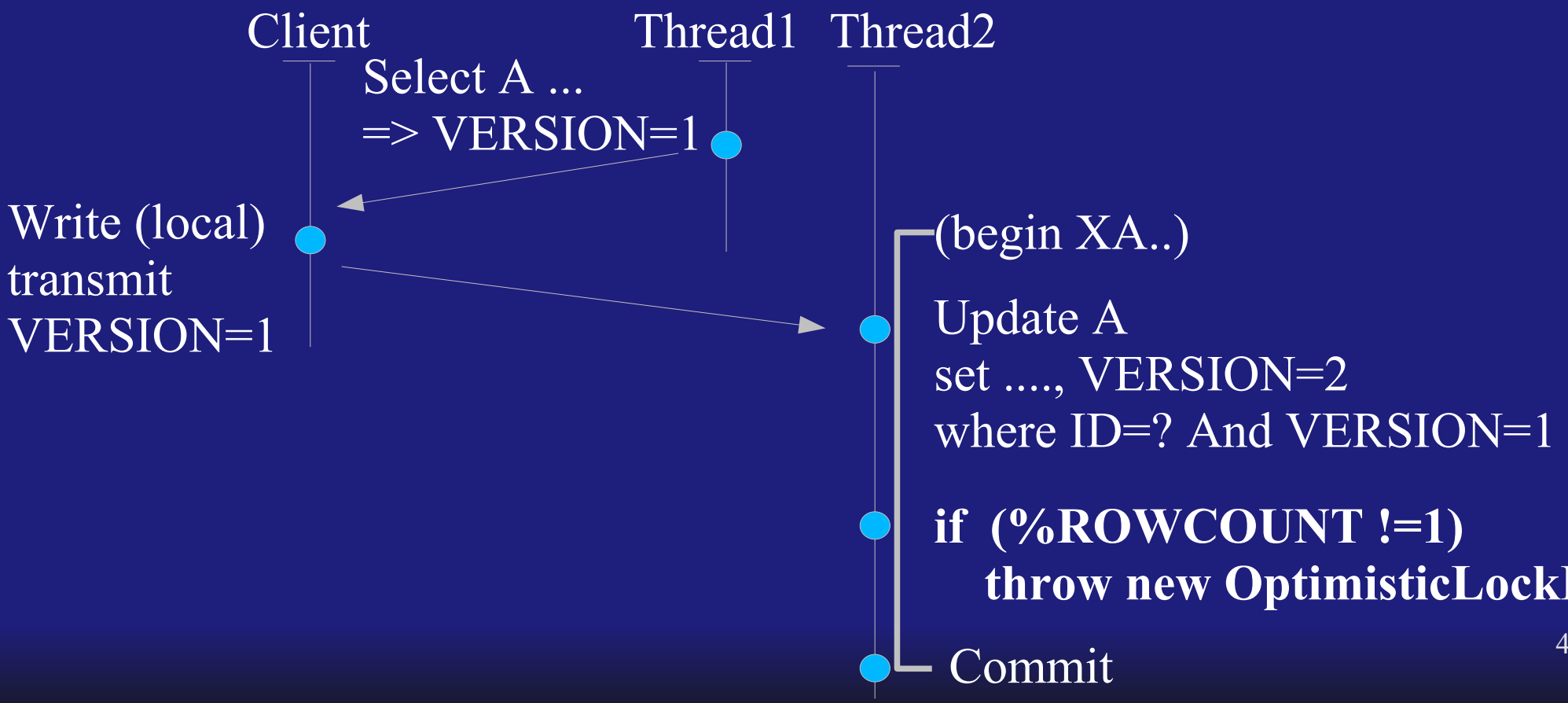
- Ex: calcul de débit/crédit ... **0 +1 +1 = 1 !!**



SUNGARD

Pseudo Lock-Optimiste en Base

- Ni Oracle, Ni ReadMostly ne lockent vraiment
  ... check version minimaliste
- OK pour les saisies utilisateurs
  ...pas suffisant pour le pb 0+1+1 = 1 !!



Conclusion

- Principes universels à connaître et maîtriser
  - Transaction ACID, JTA, XAResource
  - ReadMostly : Session : Niveau 1 / Cache : Niveau 2
  - OptimisticLock ... Niveau 3
- Combinaison des 3 solutions =
  - Outils matures, fiables et performants
  - Standards
- Pourtant... méconnaissance développeurs / DBAs

Questions

Questions ??

arnaud.nauwynck@gmail.com