SUNGARD

L'informatique qui réinvente la finance

Mapping Objet Relationnel (ORM)

Partie 1 : Principes & Fonctionnement en lecture
SQL Généré
Optimisations

Arnaud NAUWYNCK
arnaud.nauwynck@gmail.com

Plan

- Introduction
  - Architecture, Outils ORM
  - API Language L4G <-> SGBDR
- Accès en Lecture
  - Fonctionnement
  - Relation 1-1, 1-N, Héritage
- Problèmes de performances
  - Optimisations Modèle, Index
  - Optimisations Mapping, Caches
  - Optimisations Oracle

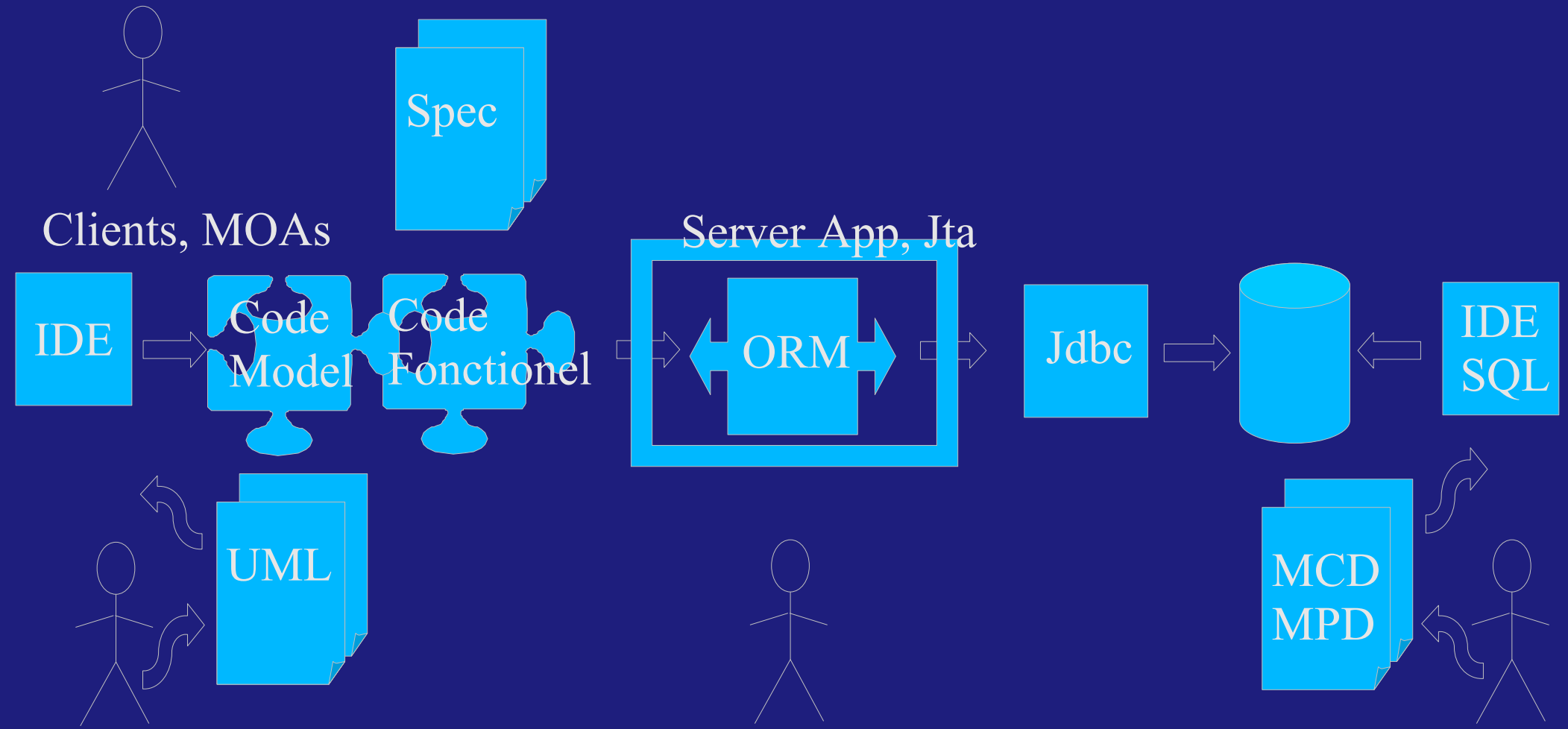
Audience - Prérequis

- PAS un tutorial hibernate/JPA/EJB...
- Pré-requis: connaissances SGBD / persistence
- But: explications avancées
  - Problèmes de mapping Object vs Relationnel
  - SQL généré
  - Problèmes de Performances, Optimisations
  - Réduire distance Développeur / DBA

SUNGARD

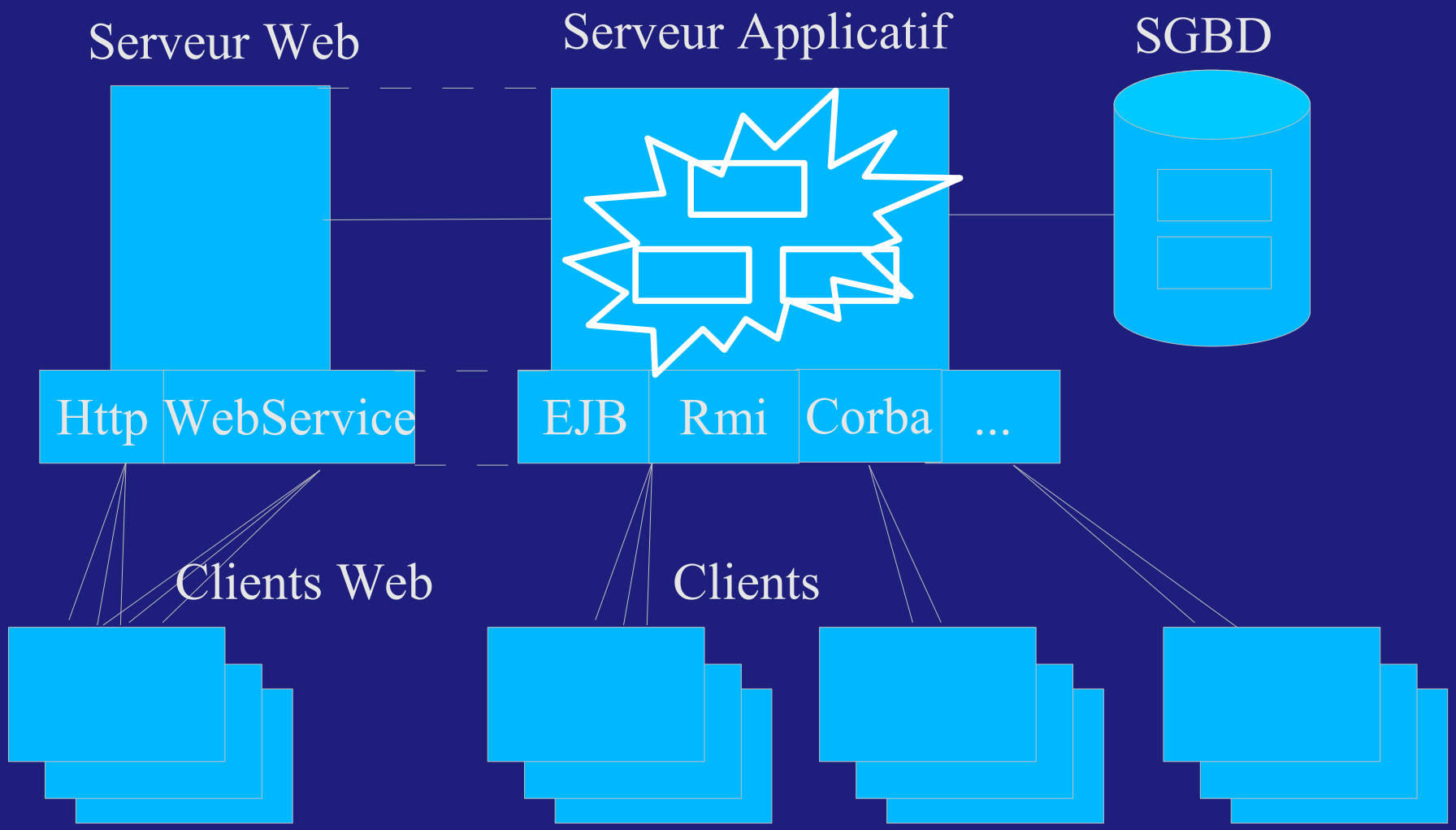
Domaine Développeur - DBA

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



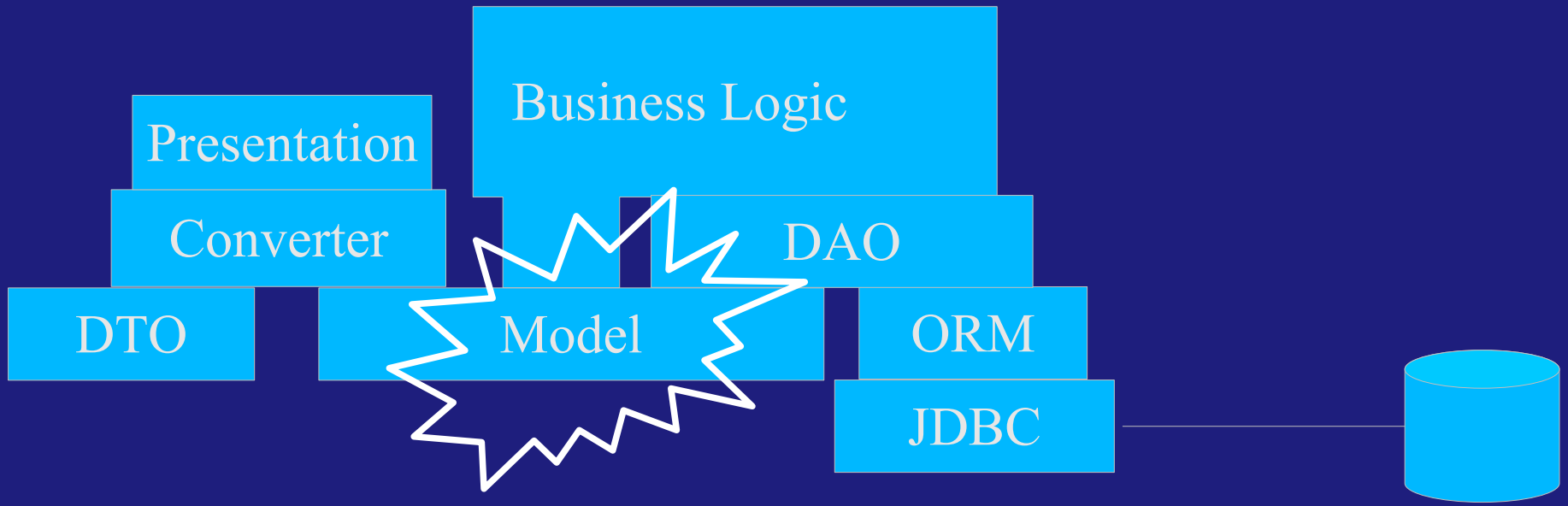
SUNGARD

Architecture 3-Tiers



SUNGARD

Couches: Applicatives, DAO, ORM, Jdbc



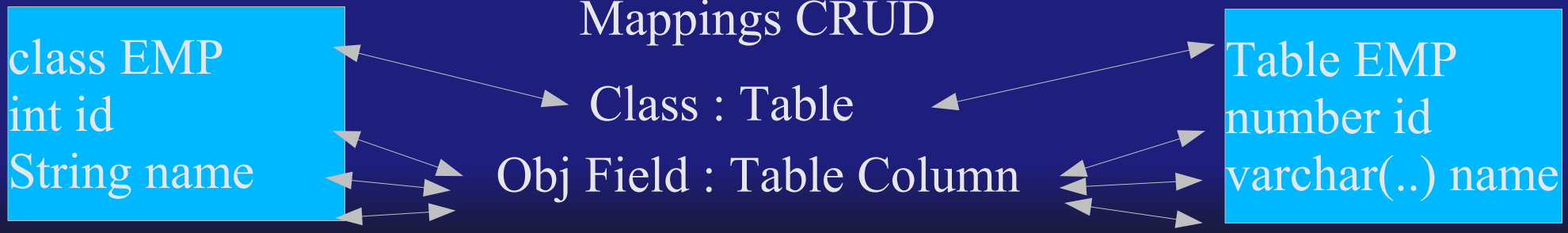
JDBC Sample Code

```
Connection conn = ...

PreparedStatement pstmt = null;
try {
    String sql = “select * from emp where id = ?”;
    pstmt = conn.preparedStatement(sql) ;
    pstmt.setInt(1, id);
    ResultSet rs = pstmt.executeQuery();
    while(rs.hasNext()) { rs.next(); int val = rs.getInt(1); ...  }
} finally { safeClose(pstmt); }
```

DAO BMP : CRUD

```
public class EmpDAO extends DAO {
```

C	**Create**	… “insert into EMP (id, name) values(?,?)”
R	**Read**	… “select * from EMP where id = ?” … “select * from where ...”
U	**Update**	… “update EMP set name=?, ...=? where id=?”
D	**Delete**	… “delete EMP where id =?”



Problèmes CRUD - BMP

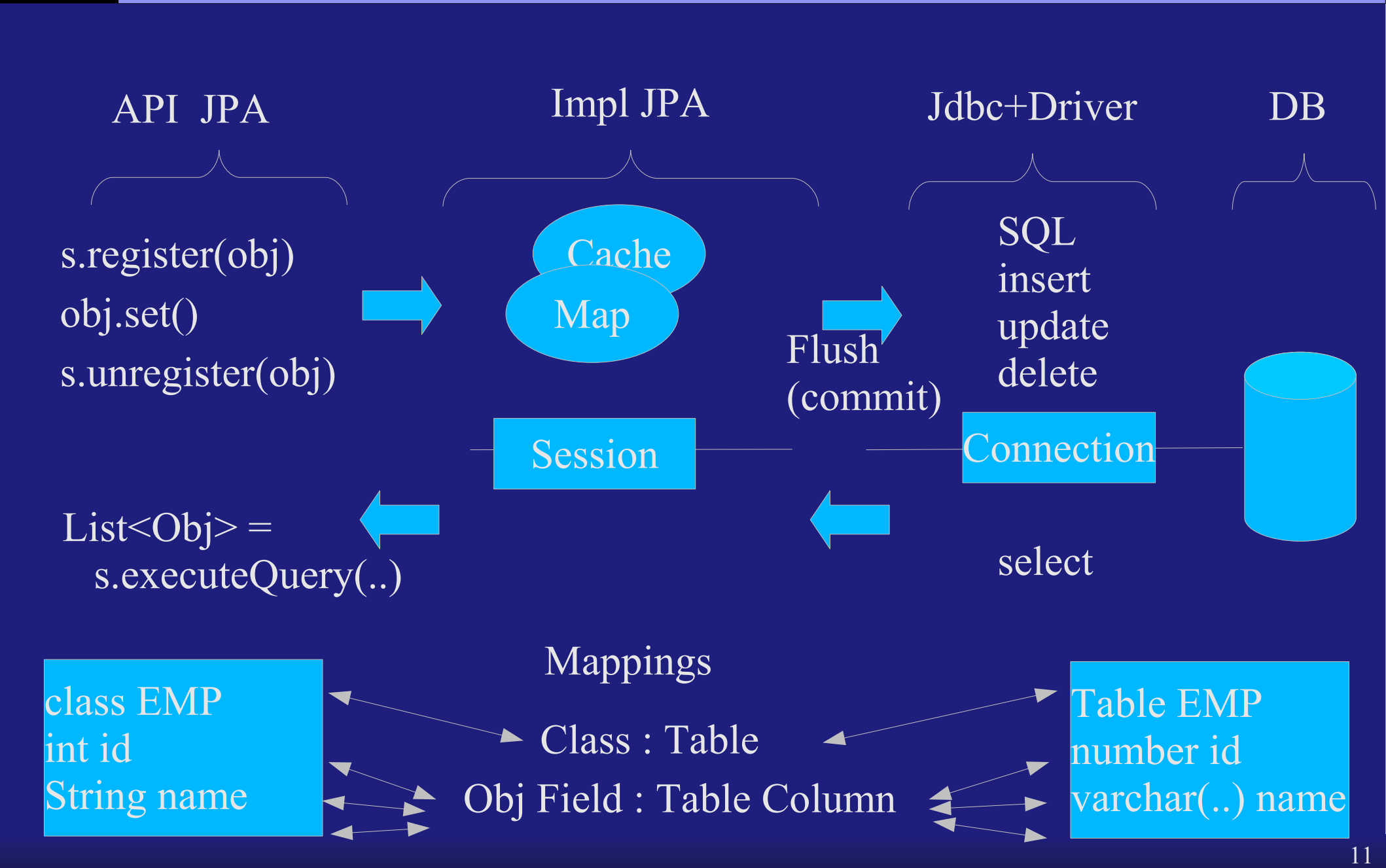
- Code simple ... mais Fastidieux
  - copier&coller *3 pour Select/Update/Create !
  - Code Query et Mapping imbriqués
- Performances
  - Avantages: 1 appel => SQL Explicit
  - Inconvénients: contrôle des appels
  - Framework nécessaire!
    - Cache pour 'Read'
    - XA pour 'Update'

ORMs : CMP/JDO/Hibernate/JPA...

- ... Solutions avec Introspection Java
- Historique:
  - Origine > 10 ans : Castor, JDO / EJB-CMP
  - Unification : JPA (EJB-3)
  - De-facto standards : Hibernate...
- Principe
  - Plus de SQL ... seulement code POJOs
  - Objets managés
  - Notion de Session (-> Connection + Cache + XA)

SUNGARD

Vue d'ensemble API ORM, IN/OUT



SUNGARD

Implémentation DAO avec JPA

```
public class EmpDAO extends DAO {

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

    Emp findById(EmpPK id) {
        return (Emp) session.findById(...); }
    List<Emp> findByXY(...) {
        return (List<Emp>) session.executeQuery(...); }

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

    void delete(Emp p{ session.unregisterObject(p); }
```

C

R

~~U~~

D

APIs de Query

- Expression
  - traduction en Orienté-Object (AST) de grammaire SQL “where clause”
- OQL / EJB-QL
  ... similar to SQL with Objects
  - Exemple:
    “select e from EMP e
    where e.name = ? and e.dept.name = ?”
- Deprecated... “Query By Example”
- Direct SQL

SUNGARD

API “where clause” : AST Expression

- API très low-level... exemple : Toplink

```
ReadAllQuery q = new ReadAllQuery(Emp.class);
ExpressionBuilder b = q.getExpressionBuilder();
Expression e = ...
Expression expr = b.get(“f”).get(“g”).equals(b.getParam(“v1”));
if (e != null) e = e.and(expr); else e = expr;
b.addArgument(“v1”);
Vector params = new Vector();
params.add(new Integer(123));
```

- => encapsuler dans API applicative high-level
  - QueryHelper b = new QueryHelper(Emp.class);
    b.andEq(“f.g”, 123);

Avantage ORM – Effets Cachés

- Manipulation Orientée Objet.... simple
  - Code = finder + getter/setter
  - Mapping externalizé (code != xml, annotation)
- ... MAIS
  - Code SQL = implicite, généré ... complexe
  - Effets non compris/prévus au dev
  - Getter =>lazy loading, relation => jointure
  - Setter => pbs XA et caches hardus
- Rien de magique... comprendre => prévoir

Sémantique Relationnel vs Object

- Inexistant en SQL
  - Hiérarchie de classe (sub-class <-> class)
  - Ordered list, array...
- Purement SQL
  - Id technique, versioning, audits, trigger...
  - Query, Relation inverse
- Remarques : généricité ultime ...
  - Table Name-Value
  - Bien pour progiciels (plugins) ... calvaires DBA

Intégration Langage 4GL - DB Relationnel

- Relation code OO <--> SGBD Relationnel...
  "The Vietnam of Computer science"
- SQL non intégré dans code:
  - Non natif dans Java
  - Sémantique non orientée-objet
- Alternatives
  - Externalisation String ... OK si SQL static
  - extensions Java-SQL (précompilateur?...)
  - Oracle : PL-SQL = langage 3GL procédural + SQL
  - SGBD Orientés-Objets (ex: O2, Versant, Db4j)

Plan

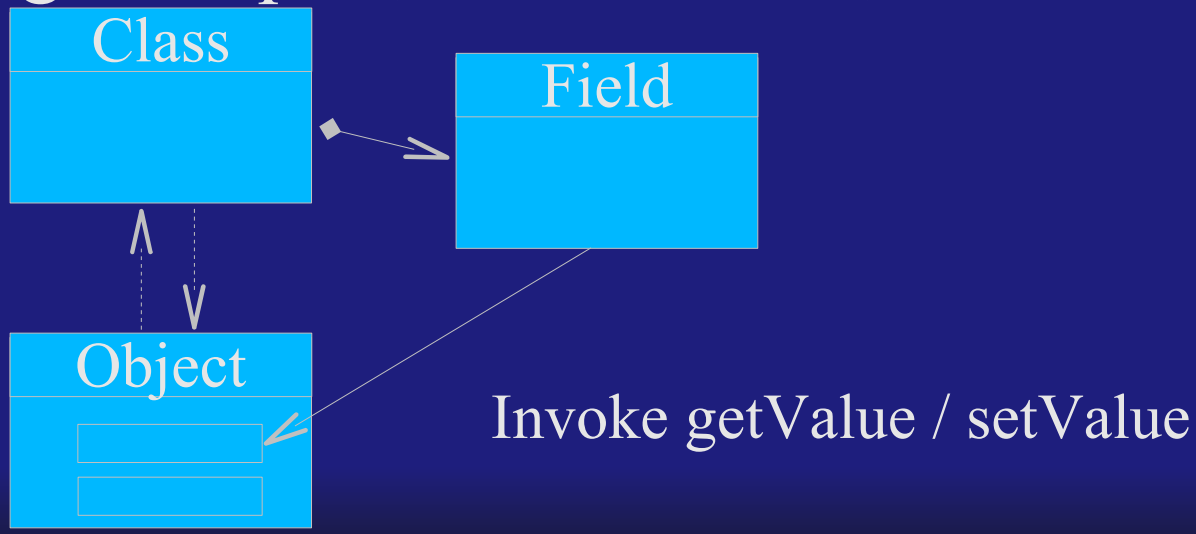
- Introduction
  - Architecture, ORM
- Accès en Lecture
  - Fonctionnement
  - Relation 1-1
  - Relation 1-N
  - Relation Héritage
- Problèmes de Performances
  - Optimisations Modèle, Mapping, Oracle

Types de Champs Mappés

- Champs simples (ValueObject)
  - Type primitif : conversion SQL – Java
  - Type user-defined: ex Date, Struct, ...
- Relations
  - Relation **1->1** "**OneToOne**" = pointer
  - Relation **1->N** "**OneToMany**" = Collection / Map
  - Relation N->M ("ManyToMany")
    => décomposée en modèle normale : 1-N + N-1
  - Relation **Héritage**

Introspection, Reflection

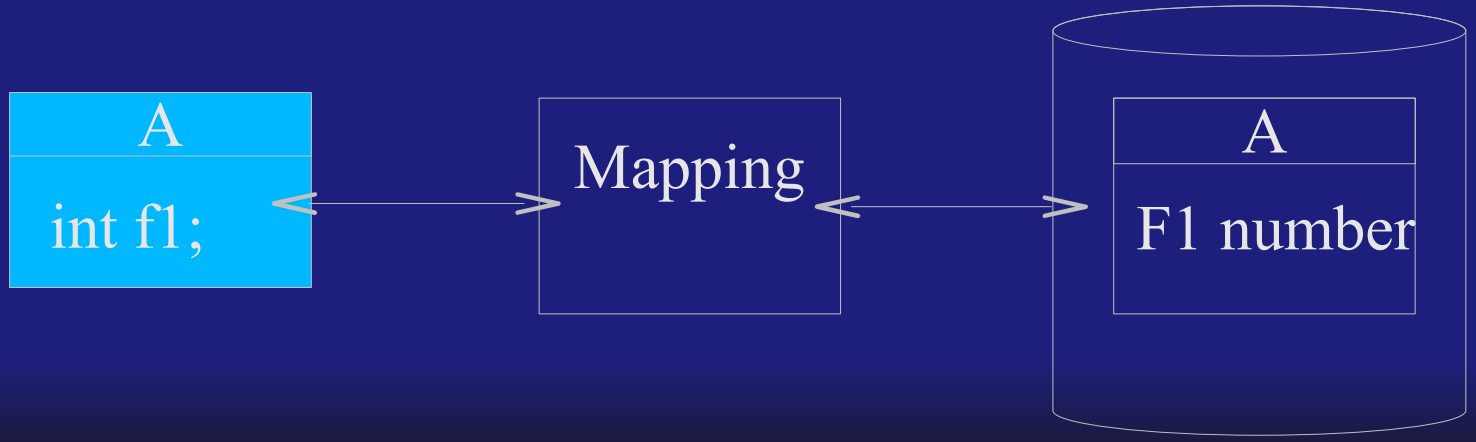
- Reflection = descripteur (meta donnée)
- Introspection = invocation “de champ/method”
- Dualité:
  obj.setXX(val) <=> xxField.setValue(obj, value)
- Conséquences:
  - outils génériques, Serialisation, ...



Mapping Champs Simple

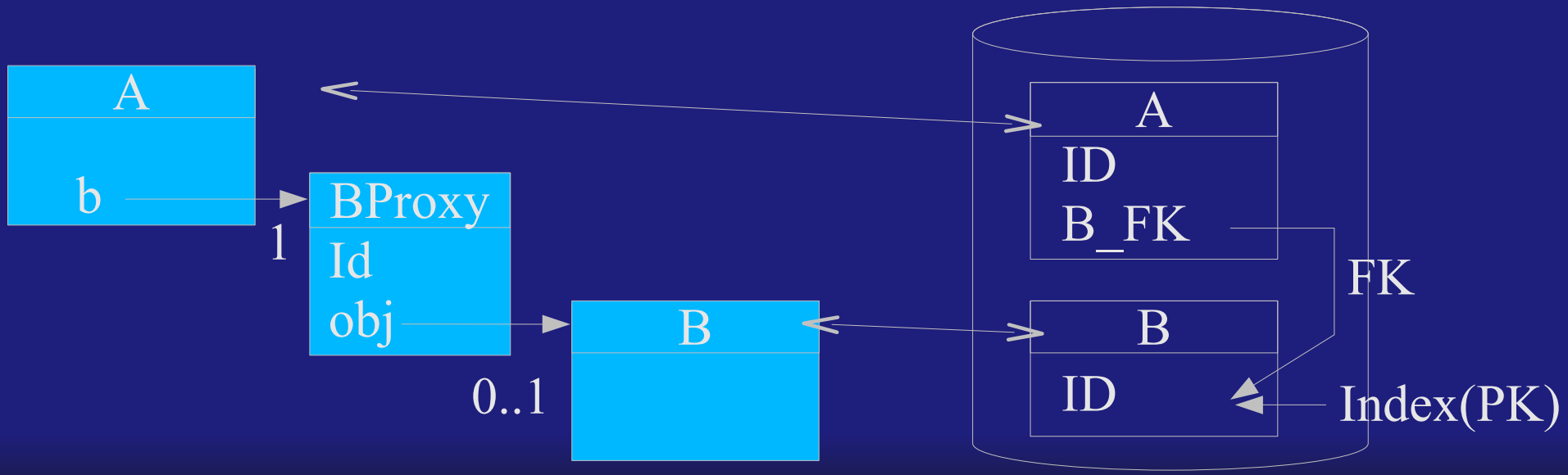
- read JDBC -> fill Obj Field

```
ResultSet rs =...   Object obj = ...
    for(FieldMapping f : ...) {
      switch(f.getMappingType()) {

        case SQL_INT:
          int val = rs.getInt(f.getSQLColIndex());
          f.setValue(obj, convert(val));
          break

      .... } /*switchType*/  ... } /*fieldMapping*/  ... } /*rs*/
```



Mapping Relation 1-1

- Relations loadées indirectement: **Proxy**
- load A
  sql="select a.B_ID ... from A a where id=?"
- **lazy load** A->B
  find B by id (in cache, then in db)
  sql="select b.* ... from B b where id=?"

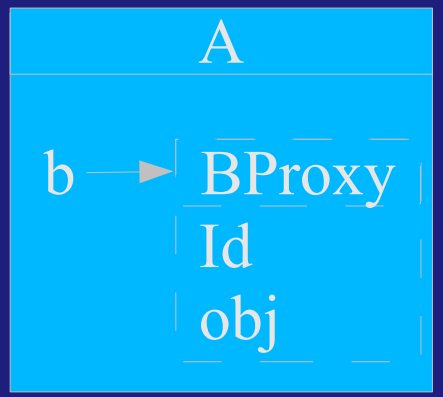


Proxy : Approche Explicite / AOP

- Approche **Intrusive** (code Explicite)
  - Ex Toplink:

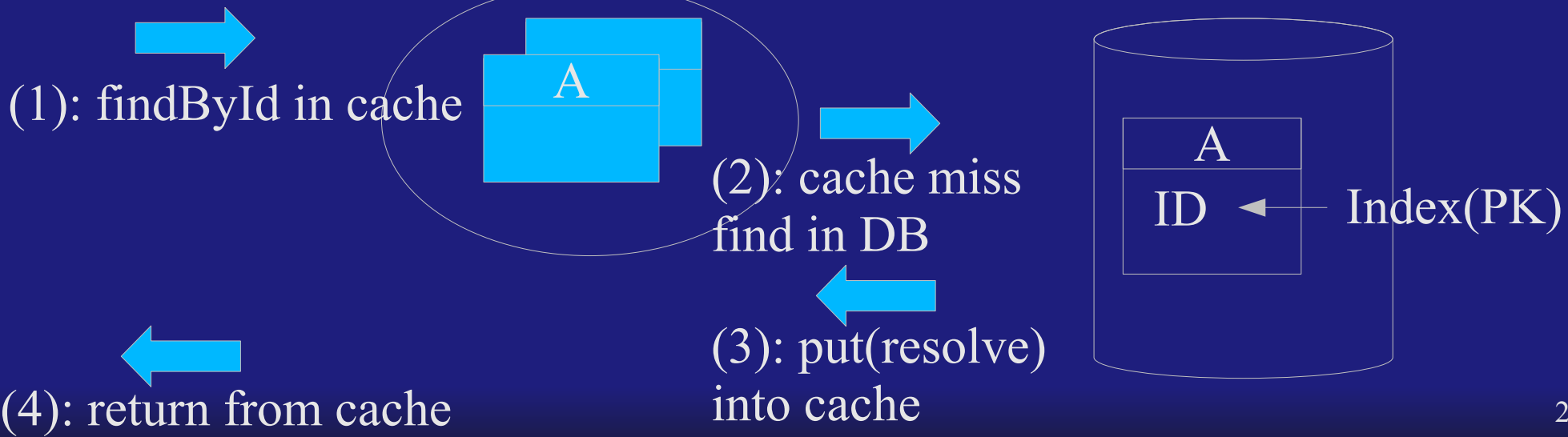
```
private ValueHolderInterface b = new ValueHolder();
public B getB() { return (B) b.getValue(); }
public void setB(B p) { b.setValue(p); }
```

- Approche Transparente : **AOP**
  - Remplacement byte-code setter/getter
    - Au Runtime (ClassLoader)
    - Ou Post-Compilation



IdentityMap : Cache by Id

- Principe:
  - 1 instance d'objet pour 1 type + 1 ID (+ 1 session)
    "==" au lieu de ".equals()"
- Conséquences:
  - FindById => always resolve in cache
  - Cache by Id = Solution optimale (Map)



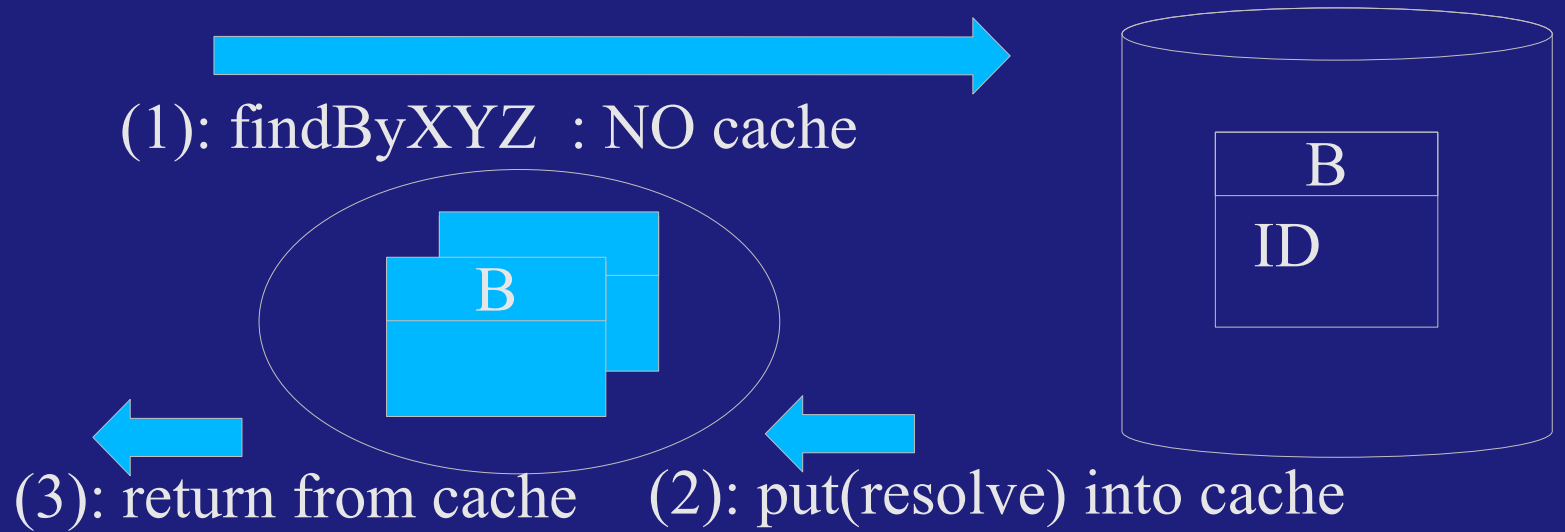
SUNGARD

Query No Cache + Resolve

- Pour TOUTES les Querys != findById

  => select in DB .... without cache

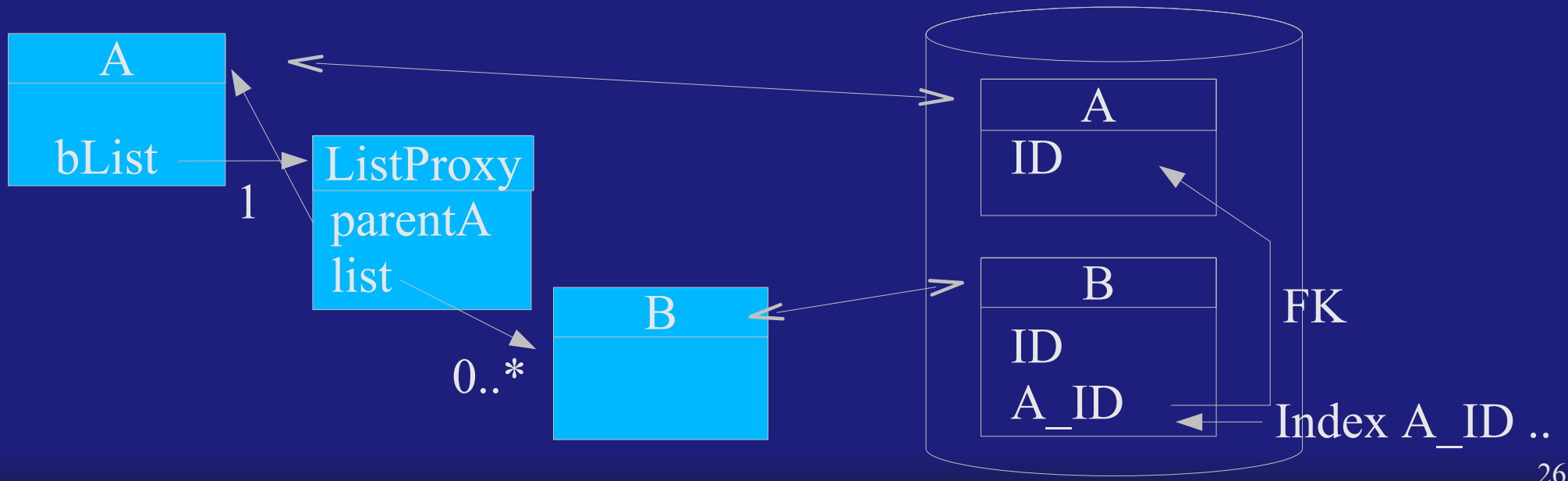
  + resolve result into cache !



- Hibernate : Cache de queries (pas Toplink)
  - Pbs: Invalidations sur updates / mémoire

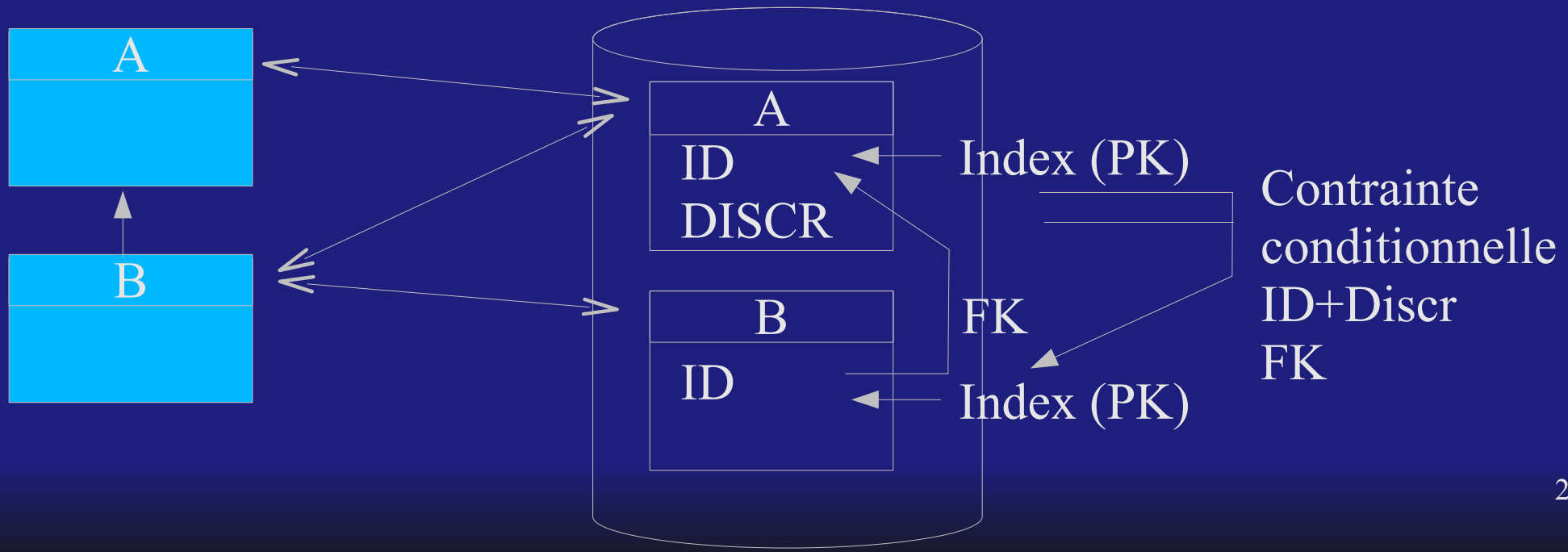
Mapping Relation 1-N

- Load A :
  sql = “select ... from A where id=?”
- Lazy load A->B :
  find all B by parentA .... in db (no cache) !
  sql = “select * from B where A_FK = ?”



Mapping Relation Héritage

- Discriminant
  ... = sous-type exact d'un objet
- Recherche A => retourne des A et des B!
  un B “est un” A
- Pb ... plusieurs tables/jointures/queries!



Code SQL pour Héritage

- Step 1 : “select distinct(A.DISCR) from A where <expr>”
- Step 2.. N : query + jointures par type
  - “select * from A a, B b where a.ID=b.ID and <expr>”
  - “select * from A,B,C where ... <expr>”
  - etc

- Adieu Performances...
- Alternative possible
  - 1 select “ids, type”
  - + N findByIds : id in (?, ?, ? ...) cf next

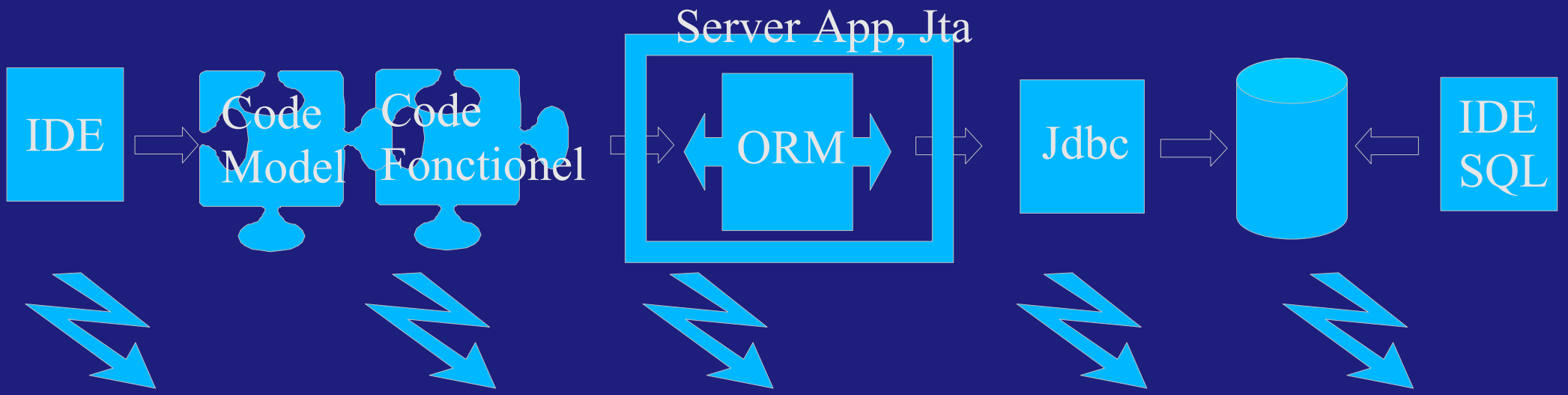
Plan

- Introduction, ORM
- Accès en Lecture
  - Fonctionnement
  - Relation 1-1, 1-N, Héritage
- Problèmes de performances
  - Optimisations Indexes, Modèle de données,
  - Optimisations Mapping, Caches
  - Optimisations Oracle

SUNGARD

Outils de Diagnostic Performance

- Nombreux Outils de Profiling... choisir cibles
- Meilleur rapport efficacité/coût: jdbc profiling



Jprobe(TM)
OptimizeIt(TM)
...

App metrics,
App log

J2EE Profiling
Wily(TM)
...

Sp6
Jdbc-logger
...

Oracle Stats
...

Développeurs

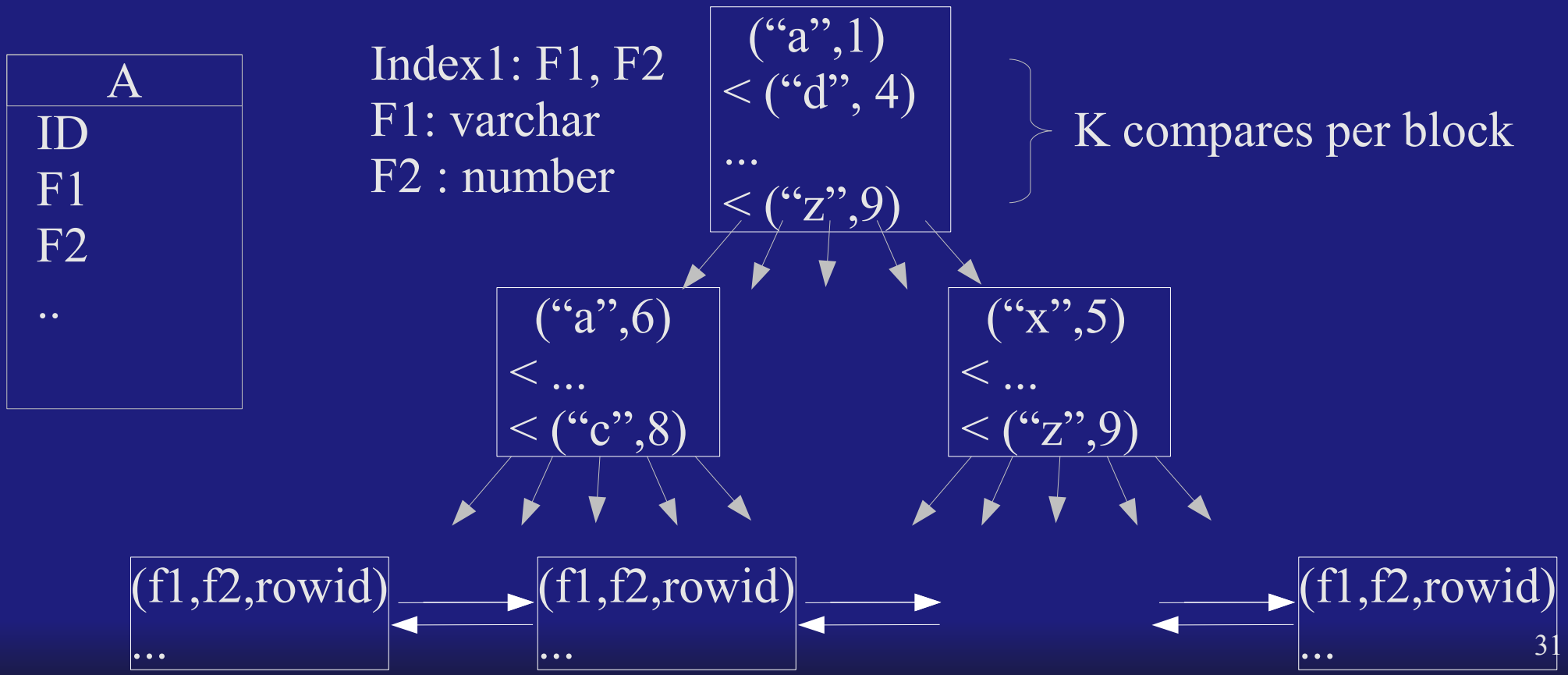
Architectes

DBAs

SUNGARD

Rappel : B*-Tree Index

- Index = B*-Tree (B=Balanced) / hash / bitmap...
  - Recherche en log(N/K) => ~2-3 logical Ios
  - Leaf doublement chainées => “index scan”



Index Eligible

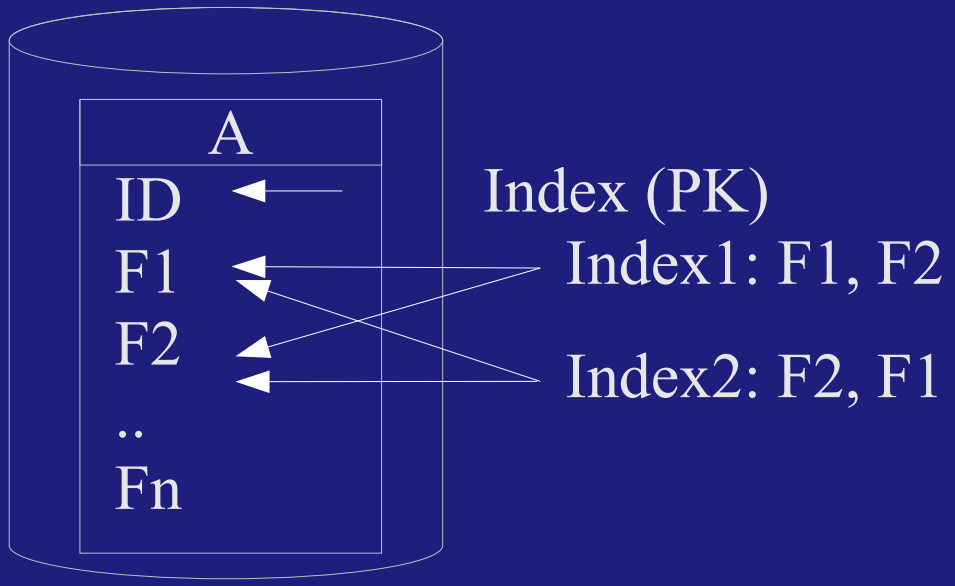
- Query “where “F1=? And F2=? ... and Fn=?”
- Utilisation de k (<=N) 1$^{\text{ères}}$ colonnes de l'index
  - Index sur (F1,F2...Fn ...) => Eligible
  - Si Manque “Fi=?” => (reste de l') index inutilisable!!

Select * from A where ...

where F1=? ==> index1 only

where F1=? And F2=?
==> index1 or index2

where F4=? ==> no index

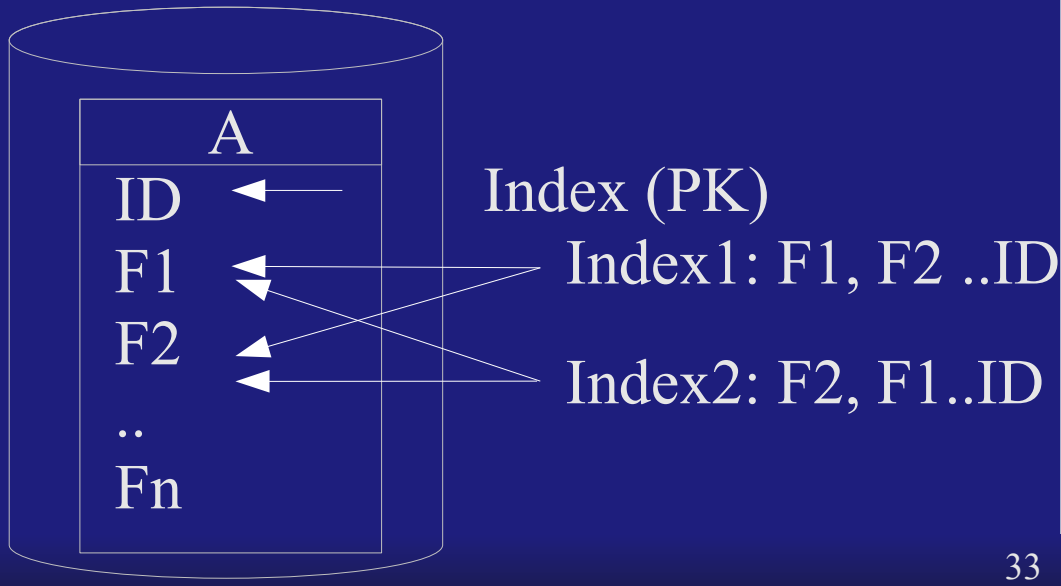


SUNGARD

Couverture d'Index

- “select Fn+1, ... Fp where F1..Fn”
  - Colonnes utilisées en “lecture” au lieu de “recherche”
  - Ordre indexé indifférent
  - Bonne idée: rajouter “ID” en dernier

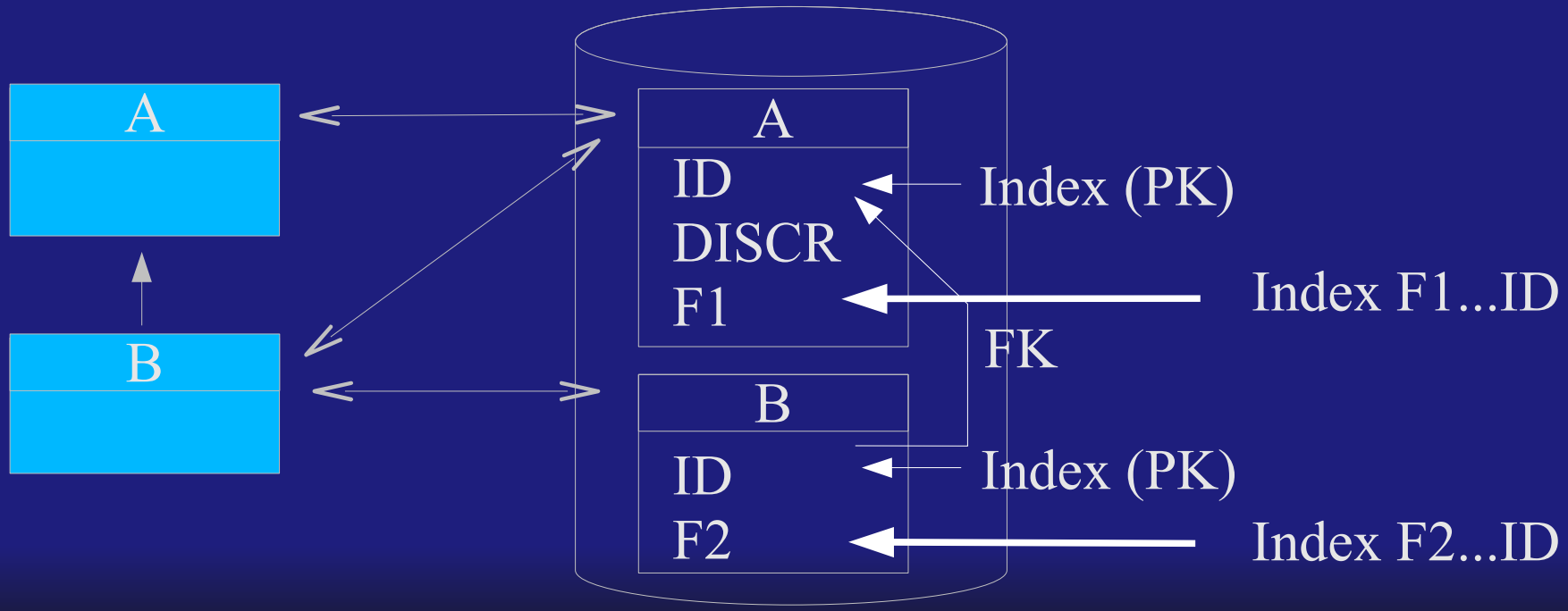
Select F2 from A where F1=?
== > index 1 ... avec couverture d'index
(no table access A)



SUNGARD

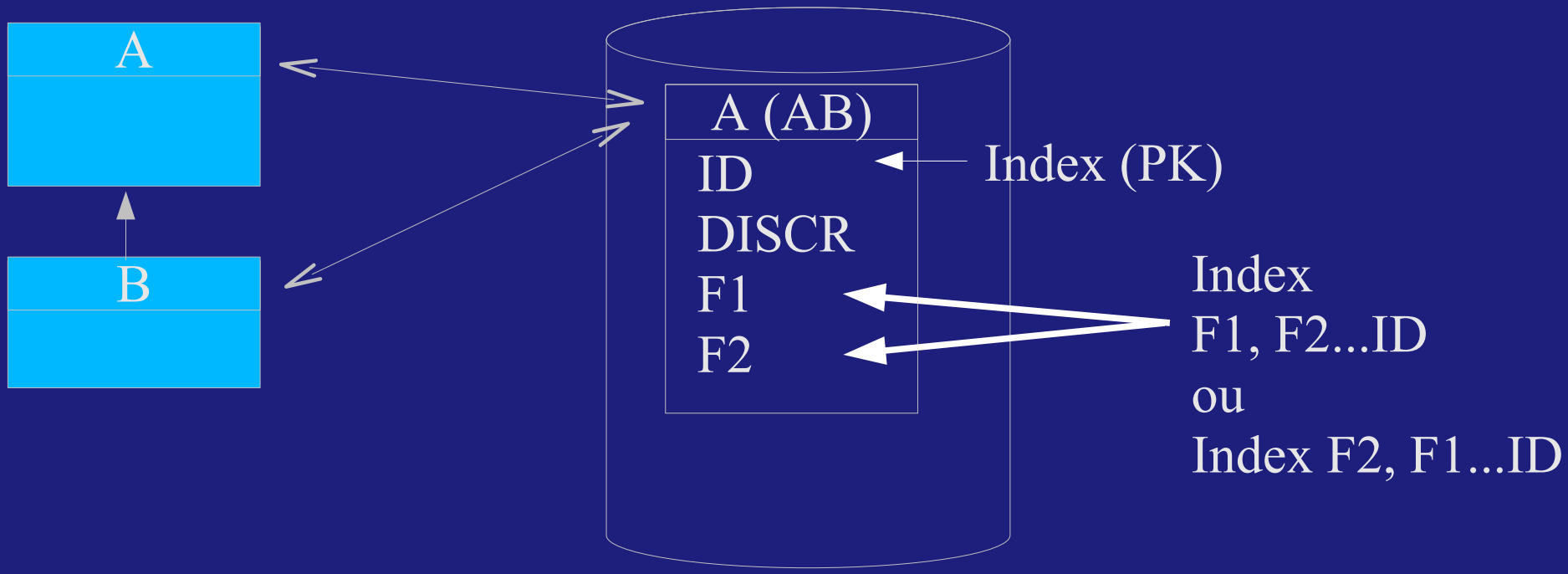
Pb de Performance avec Heritage

- lsB = findBy(A_F1, B_F2)
- sql: “select * from A a, B b
  where a.ID =b.ID
  and a.F1 = ? and b.F2 = ?”
- Pb: sous-indexation séparée de 2 tables = HashJoin



Dénormalisation Mapping

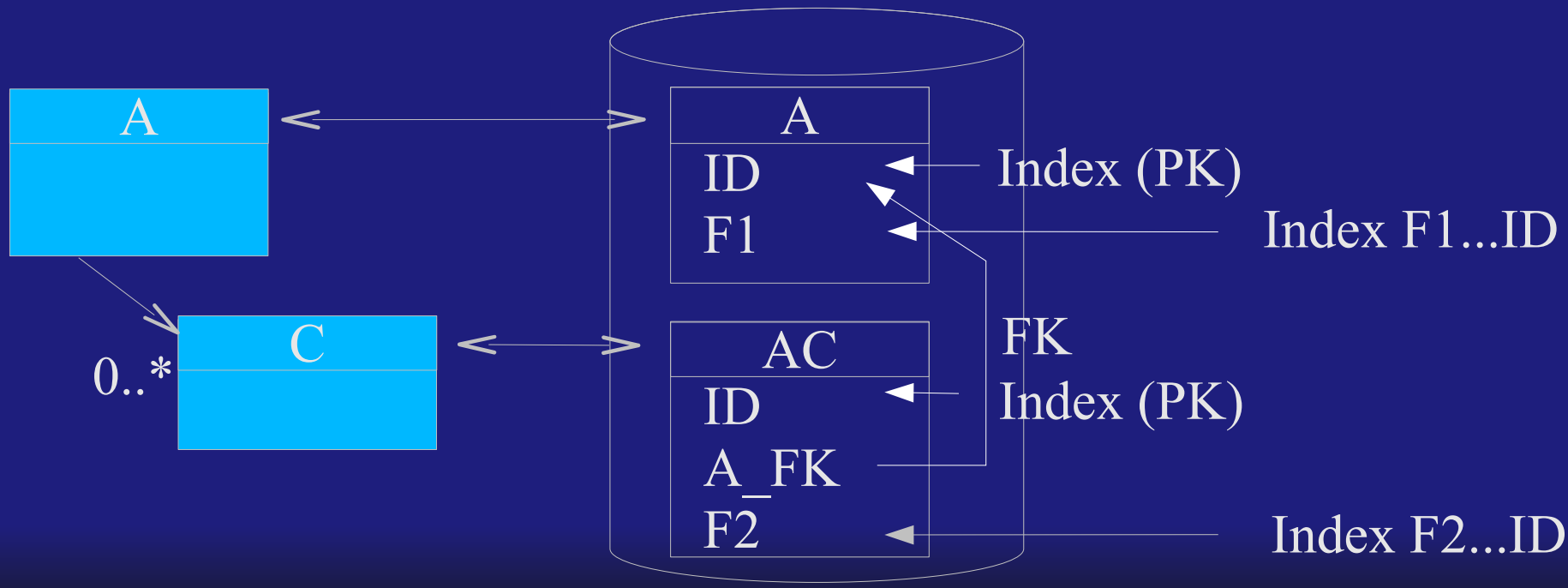
- 2 Class <-> 1 seule table + columns nullables



SUNGARD

Pb Similaire avec Relation OneToMany

- lsC = findBy(A_F1, C_F3)
- Sql: “select * from A a, C c
  where c.A_FK = a.ID
  and a.F1 = ? and c.F3 = ?”
- Pb: sous-indexation séparée de 2 tables

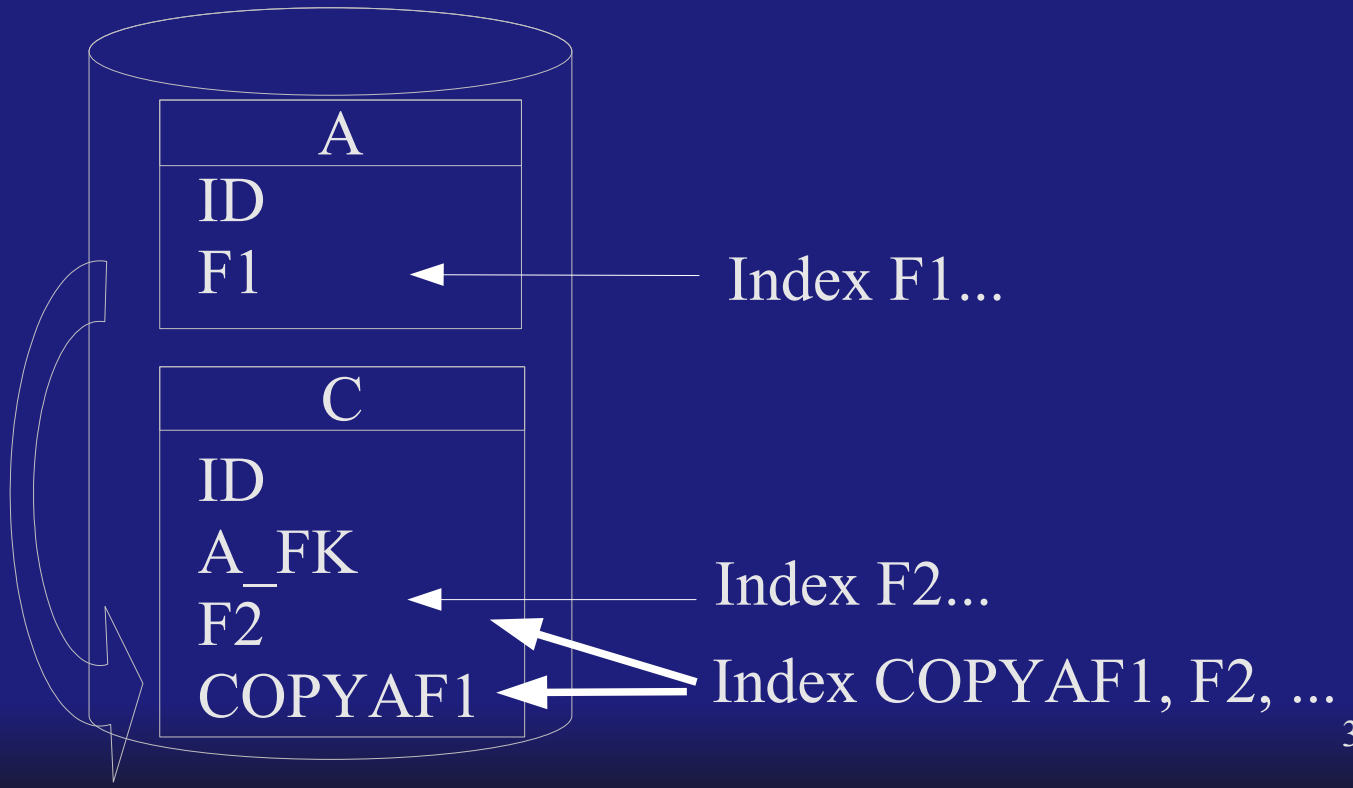


SUNGARD

Denormalisation: Copy Trigger

- Trigger to copy A.F1 -> C.COPYAF1, then index
  - on update A.F1 / on Insert C / on Update C.A_FK
- Pb: changer query,
  utiliser C.COPYAF1 au lieu de =A.F1

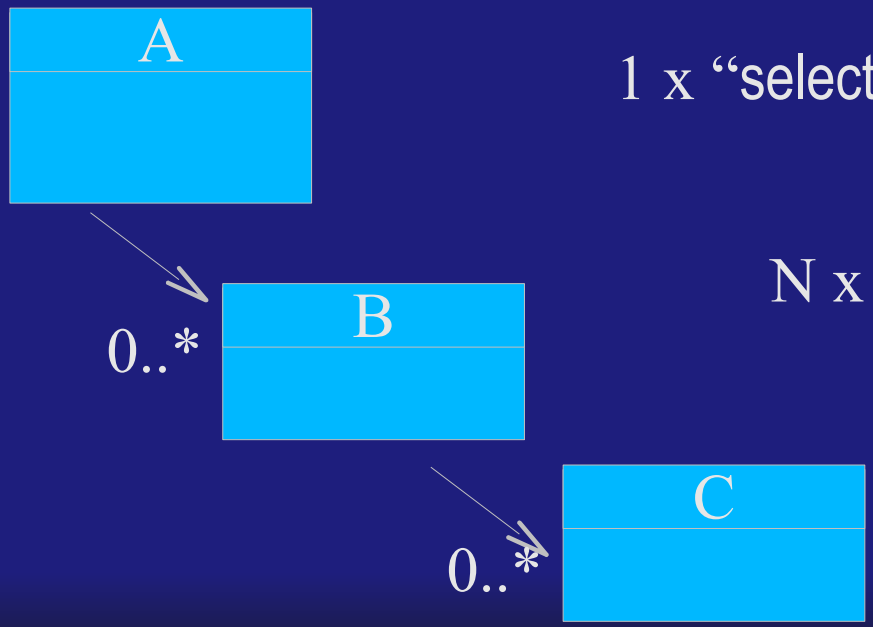
solution avancée
Materialized Views
+ Query Rewrite



SUNGARD

Problème Applicatif : “N+1” Queries

- lsA = findBy(...); // 1 Query
  for (A a : aLs) {
  b = process(a); // N Queries! 1 per A
  ... c = process(b);
  }



1 x “select * from A a where ...”

N x “select * from B where ...”

NxP x “select * from C where ...”

Problème des “N+1” Querys

- Extrèmement fréquent, voir Omni-présent !
- Complexité linéaire
  ... quadratique si 2 pbs “N+1” en cascade !
- 1 Traitement => N querys => Timeout
  - query = appel réseau + parse + execute + fetch
  - 15 ms min (~70 ms avg)
- Stats en Prod
  - ~10M de selects par jours = 100 querys / seconde

Solutions au Problème “N+1”

- Refactorer Modèle : Query->Relation
  - => Relations loadées (lazily) 1 seule fois
    ... puis rattachées au cache
- Mapper les relations OneToOne
  - Ne pas mapper des “int” mais des pointeurs!
- Caches + preload (hot data)
- Batch Reading / Join (?)

- Machine 4/8 CPUS... use it (don't save it!)

Solution au “N+1” : Cache By Id

- lsA = findBy(...); // 1 Query

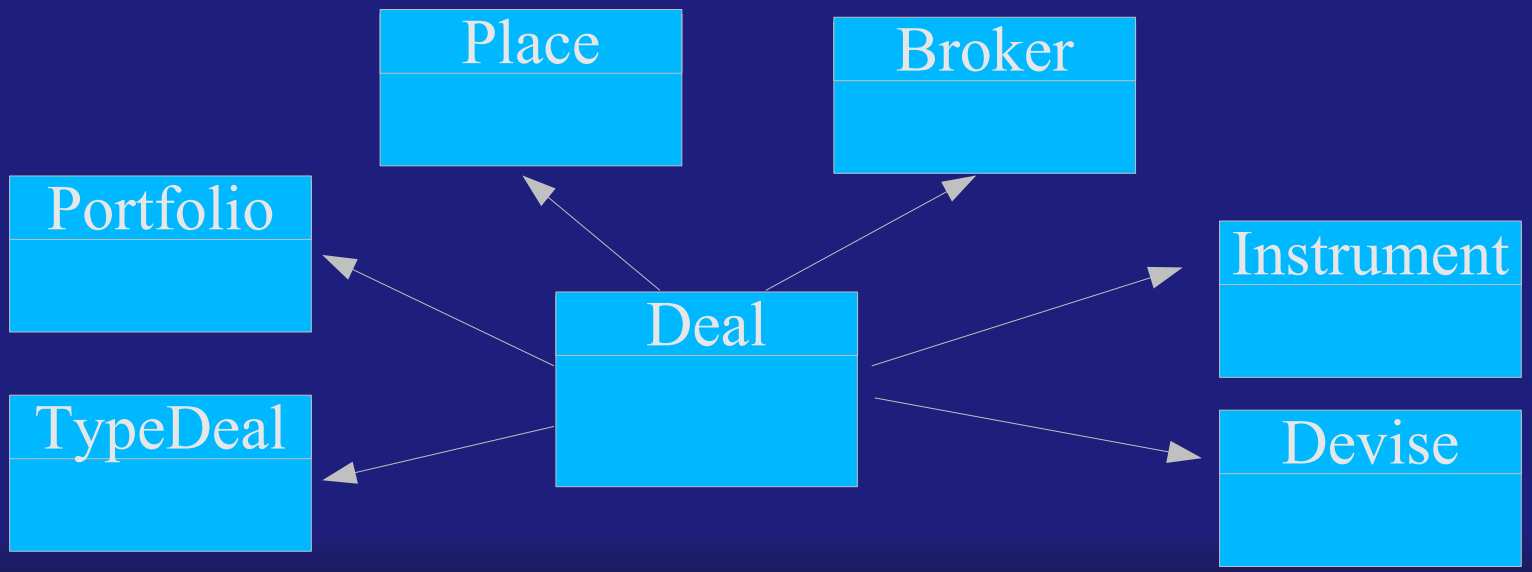
```
// preload corresponding B at startup/on demand
// “select * from B where ...”
// => put in cache: cacheB.put(bId, b)

for (A : aLs) {
    B b = a.getB();
    // = lazy load B by id = hit cache, in memory...
    // = cacheB.get(bId)
}
```

SUNGARD

Relation 1-1 de Référentiel

- Relations 1-1 très nombreuses
  - Partie “référentiel”
  - Schéma en “Etoile”, typique des Datawarehouses

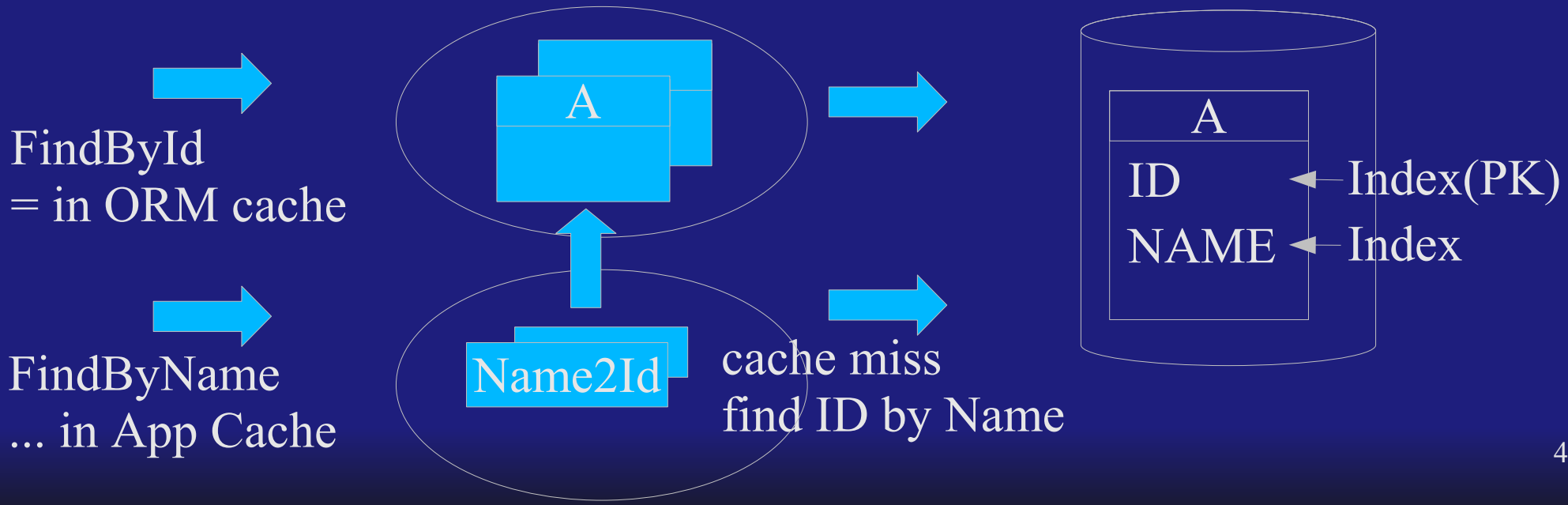
- Exemple



SUNGARD

Cache Applicatif – By Code != By Id

- 2 exemples typiques:
  - Table “User” => findByName
  - Table translation “CodeExtern” => findExtBySysCode / findCodeBySysExt
- Solution Cache Applicatif: “Name->Id”



SUNGARD

Solutions au “N+1” : Batch/Join Reading

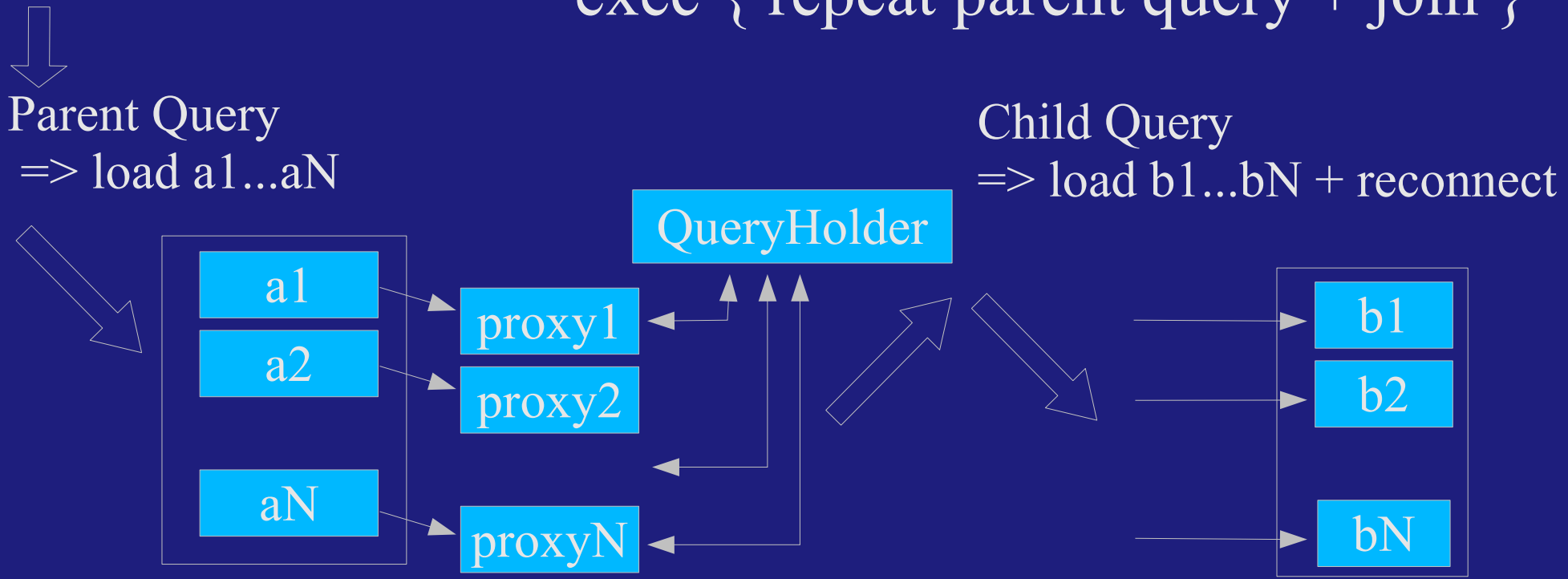
- FAUSSE Bonne Idée :
  “**Rajouter une jointure pour éviter une query ?**”

- Problèmes: parfois pire que mieux
  - execution (repeat “where clause”)
  - volume de données (repeat “data”) / resolve cache!
- Où ?
  - Jamais pour données de référentiel
  - Exclusivement pour aggrégations fortes
    - Ex: load Curve(s) => load Child Points

SUNGARD

Explication Batch Reading

- "Proxy" remplacé par "QueryBatchProxy"
  - Remember parent query + group children
  - Lazy-load 1 child = load All Children
    = exec { repeat parent query + join }

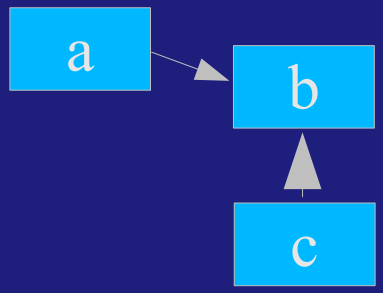


SUNGARD

Fonctionnement QueryBatchProxy

- lsA = findBy(...); // “select * from A where ...”
  lsA[0].getB(); // use one b ... load all B
  // “select B.* from A a, B b where ... and a.ID=b.A_FK”

- Queries répétées N fois avec 1 + 2 + ... + N jointures
- Cascader BatchReading + Héritage = désastre !!
- ex:



=> 3 requètes SQL :
1: select * from A ...
2: select A,B ...
3: select A,B,C ...

Plan

- Introduction, ORM
- Accès en Lecture
  - Fonctionnement
  - Relation 1-1, 1-N, Héritage
- Problèmes de performances
  - Optimisations Indexes, Modèle de données
  - Optimisations Mapping, Caches
  - Optimisations Oracle

Find By Ids Multiple

"select * from A where **ID in (.., .., ..,)**"

- Besoin fréquent: pour "N+1", héritage...
- Problèmes
  - pas de bind variables, hard parse
- Solution Oracle:

  "select * from A where ID in (select id from table(cast ? As T))"

  OracleArray = ...

  pstmt.setObject(1, oracleArray);

SUNGARD

Reports: FIRST_ROWS / Rownum

- Ecrans de recherche
  - Critères facultatifs... => rapide / full table scan
- Solution 1 : Query Timeout
- Solution 2 : “select /*+FIRST_ROWS*/ ...”
- Solution 3 : “where ... rownum < 500”
  - execute query != consume resultset.next() ... cf pb Toplink
  - Solution 3bis: Use rownum paging [first,last]

“select .. from (select rowum r, .. from ... where rownum < last) where r > first”

SUNGARD

Query Reports... Combining Solutions

- Trop de jointures ?
  a) Select Object + Héritage ... jointures
  b) Select “a.name” (référentiel?)... jointure
  c) Critère “a.b.c.d = ?” ... jointures

- Pour a) : “select ID from....” + findByIds
- Pour b) : “select a.id” + cache applicatif
- Pour c) : split queries... c.d=? / a.b in (...)

BindVariable, Soft/Hard Parse

- Utiliser **PreparedStatement** + **Bind Variables**
- 1ère exécution = **Hard-Parse**
  - CPU pour Parser + optimizer Plan
  - Stocké dans V$SQLAREA
- Re-exécution = **Soft-Parse**
  - Lookup (par text sql) + check + execute

- Sans BindVariable... performance impossible
  - Spin-locks (=CPU + lock), Cache miss / Mémoire

SUNGARD

Hard Parse – Soft Parse : JDBC

- Mauvais Exemple:

```
for (int i = 0; i < 1000; i++) {
   conn.executeQuery("select * from A where id=" + i);
}
```

- Correct, 2x plus rapide:

```
String sql = "select * from A where id = ?";
for (int i = 0; i < 1000; i++) {
   stmt = conn.prepareStatement(sql);
   stmt.setInt(1, i);
   stmt.executeQuery();
   stmt.close();
}
```

Softer-Soft-Parse

- **Softer-Soft-Parse**
  - Reuse PreparedStatement, no close
    => skip lookup, check
    => only execute + fetch

- => 2x2 fois plus rapide:

```
stmt = conn.prepareStatement("select * from A where id = ?");
for (int i = 0; i < 1000; i++) {
   pstmt.setInt(1, i);
   pstmt.executeQuery();
}
pstmt.close(); // repool in LRU pstmt cache
```

Tuning Soft(er) Parse

- Si code sans PreparedStmt... Work Around System
  - "alter system set CURSOR_SHARING similar"
  - ... pb induit: pseudo variables pour constantes!
    => plan d'exécution moins précis

- Config Serveur Applicatif
  - Pool JDBC > LRU cache Statement Size = ...
- Dans PL-SQL => implicit !!! open+cache cursors...

Conclusion

- Mapping Objet-Relationnels
  - Outils matures, fiables et performants
  - Standards
- Sémantique Objet vs Base Relationnel
  - Diff (in)évitable ... pb performance
- Principes universels
  - à connaître et maîtriser
  - Pourtant... méconnaissance développeur / DBA

Conclusion... A Suivre

- 1ere partie = accès en lecture, SQL généré, perf
- ... cf 2eme partie = accès en écriture

Plan

- Aspects Transactionnels : JTA, XAResource
- Fonctionnement ORM - XA
- Lock Optimiste, Versionning
- Pattern Read-Mostly (cache niveau 1 / 2)
- Oracle : Undo/Redo, isolation no-read-lock

Questions

Questions ??

arnaud.nauwynck@gmail.com