

Java et les bases de données:

JDBC: Java DataBase Connectivity
SQLJ: Embedded SQL in Java

Michel Bonjour

<http://cuiwww.unige.ch/~bonjour>

Plan

- **JDBC: API bas niveau pour l'accès aux BD (SQL)**
 - Introduction
 - JDBC et : Java, ODBC, SQL
 - Interfaces, exemples
 - Types de drivers JDBC, architectures
- **SQLJ: code SQL 'embarqué' dans Java**
 - Introduction
 - Développement en SQLJ
 - Concepts, exemples
 - Traduction SQLJ - JDBC

JDBC: Introduction

- **Quoi ?**
 - API Java pour interagir avec des BD relationnelles
 - * exécuter des requêtes SQL (statiques ou dynamiques)
 - * récupérer les résultats
 - Tentative de standardiser l'accès aux BD (futur: ODMG)
 - Spécification basée sur X/Open SQL CLI (comme ODBC)
- **Pourquoi ?**
 - Réponse de SUN à la pression des développeurs
 - Java est idéal pour les applications BD
 - Alternative aux solutions propriétaires

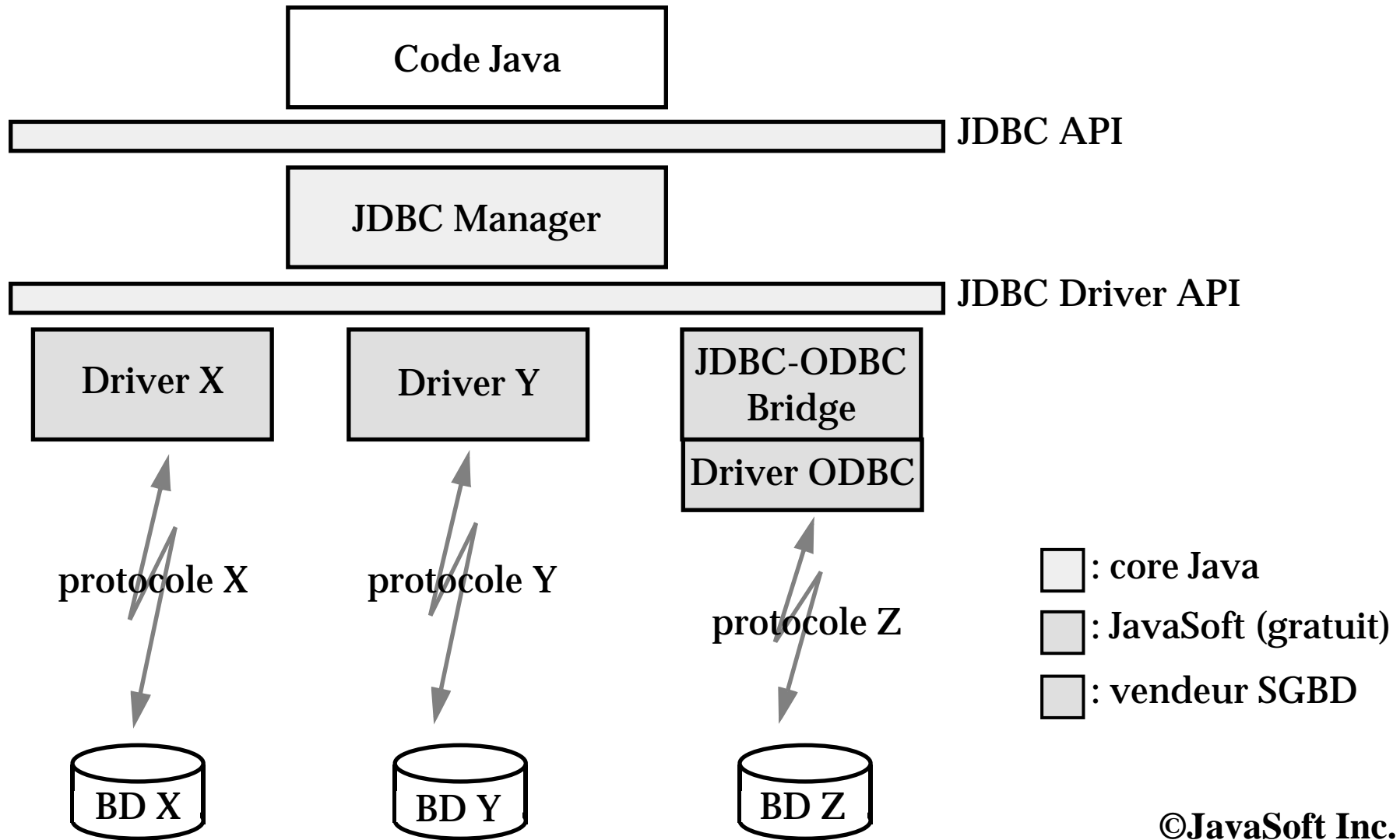
JDBC et ...

- **Java: JDBC c'est du Java !**
 - Interfaces, classes, multi-threading
 - Applets, applications, utilise le Security Manager
- **ODBC: JDBC est "au-dessus" de ODBC**
 - Arguments pour ODBC:
 - * existe, implanté, fonctionne
 - * accepté, diffusé
 - Arguments contre ODBC
 - * très lié au langage C (void *, pointeurs)
 - * compliqué, basé sur des paramètres locaux

JDBC et SQL

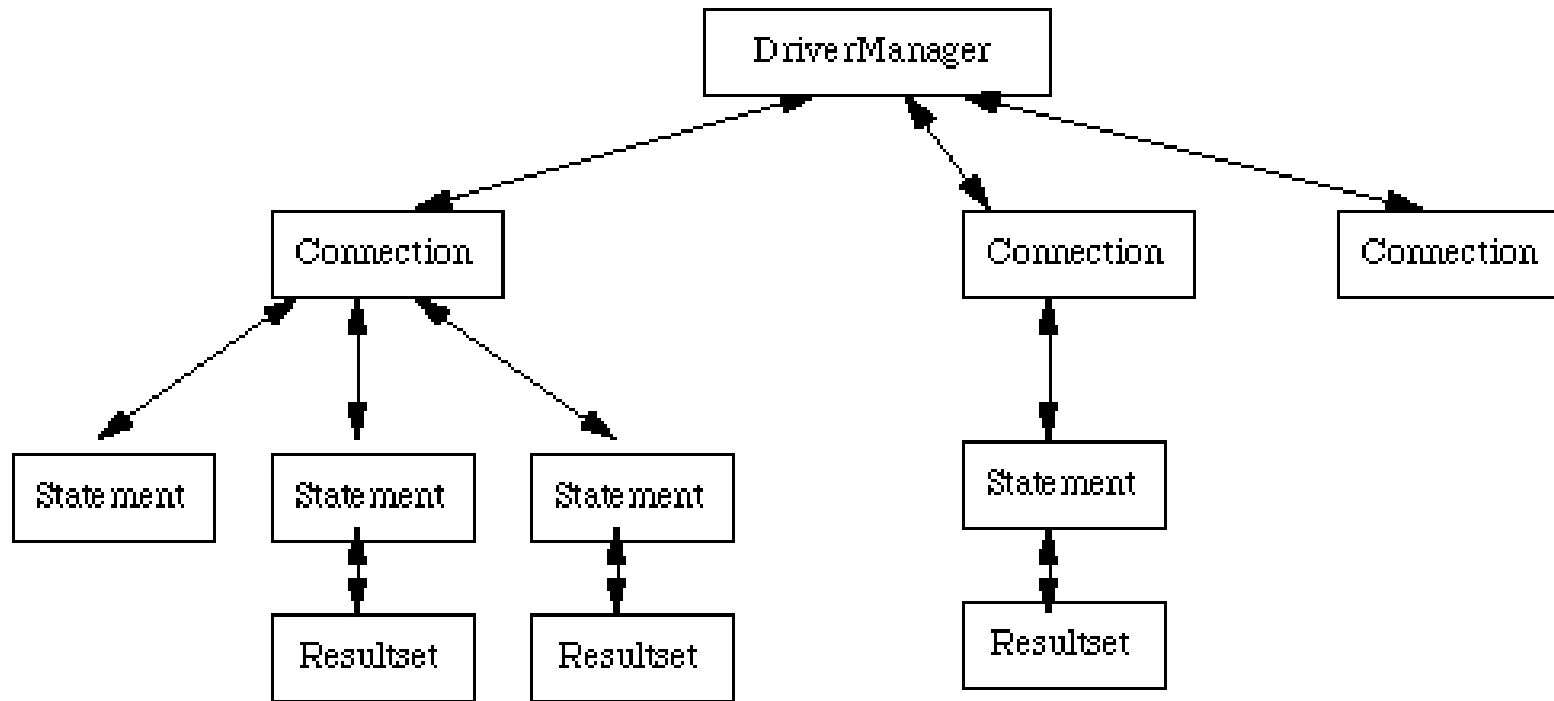
- **Support de SQL-2 Entry Level**
 - SQL dynamique, principaux types SQL
 - Transactions, curseurs simples
 - Méta-données (accès au dictionnaire de la BD)
- **Mécanismes d'extension**
 - Syntaxe inspirée d'ODBC: { mot-clé ... paramètres ... }
 - Fonctions ODBC de conversion, mathématiques, etc.
- **Long terme**
 - Support de SQL-2 complet

Architecture de JDBC



©JavaSoft Inc.

Interfaces de JDBC (1)



©JavaSoft Inc.

Interfaces de JDBC (2)

- **DriverManager**

- Gère la liste des Drivers chargés
- Crée les connexions TCP (**Connection**)
- 'Mappe' des URLs vers des connexions

- **Connection**

- Canal de communication TCP vers une BD
- Format d'URL: `jdbc:odbc:cuibd.unige.ch:9876/mabd`
- Propriétés de la Connection: `username`, `password`
- Crée les **Statements** (requêtes SQL)
- Gère les transactions (au sens de SQL)

Interfaces de JDBC (3)

- **Statement**

- Gère les requêtes SQL simples
- Sous-types:
 - * **PreparedStatement**: requêtes paramétrées (IN)
 - * **CallableStatement**: procédures stockées (OUT)
- Passage de paramètres:
 - * méthodes set...
 - * accès par l'index
- Crée les **ResultSet**

Interfaces de JDBC (4)

- **ResultSet**

- Gère l'accès aux tuples d'un résultat (SELECT)
- Accès aux colonnes: par nom, par index
- Conversions de types entre SQL et Java
- Possibilité d'utiliser un stream pour récupérer les données "brutes" (bytes, ASCII, Unicode)
- Gère des curseurs "simples"

Exemple de SELECT

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@host:port:sid", "user", "passwd");

java.sql.Statement st = conn.createStatement();

ResultSet r=st.executeQuery("SELECT nom, age FROM T1");

while (r.next()) {
    // imprime les éléments du tuple
    String lenom = r.getString("nom");
    int lage = r.getInt ("age"); // ou bien r.getInt (2)

    System.out.println ("nom: " + lenom + " age: " + lage);
}
```

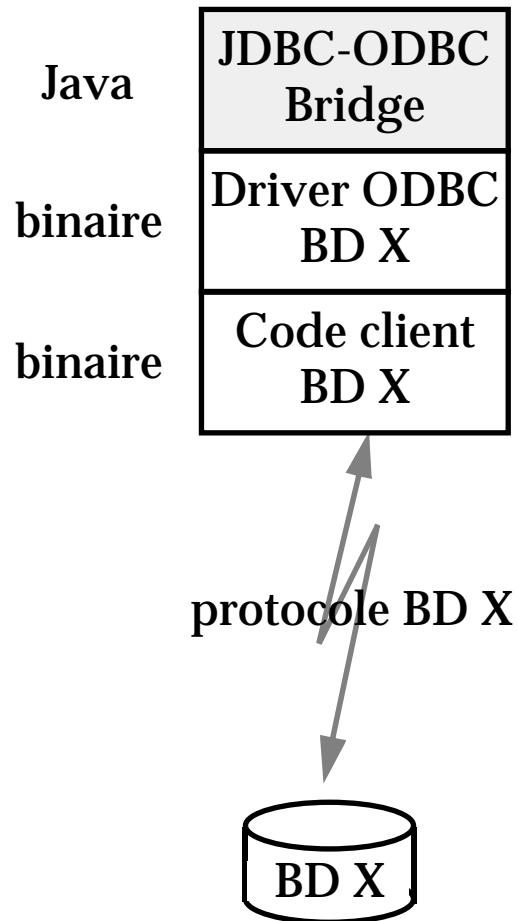
Exemple de UPDATE

“afficher le nb de personnes ayant 14,15..18 ans et modifier leur catégorie (Ado) “

```
java.sql.PreparedStatement ps =  
conn.createStatement("UPDATE T1 SET cat=? WHERE age=?");  
  
ps.setString(1, "Ado");  
  
for (int i = 14; i < 19; i++) {  
    ps.setInt (2, i);  
    int nbTuples = ps.executeUpdate();  
    System.out.println ("age: " + i + " nb: " + nbTuples);  
}
```

Driver JDBC Type 1

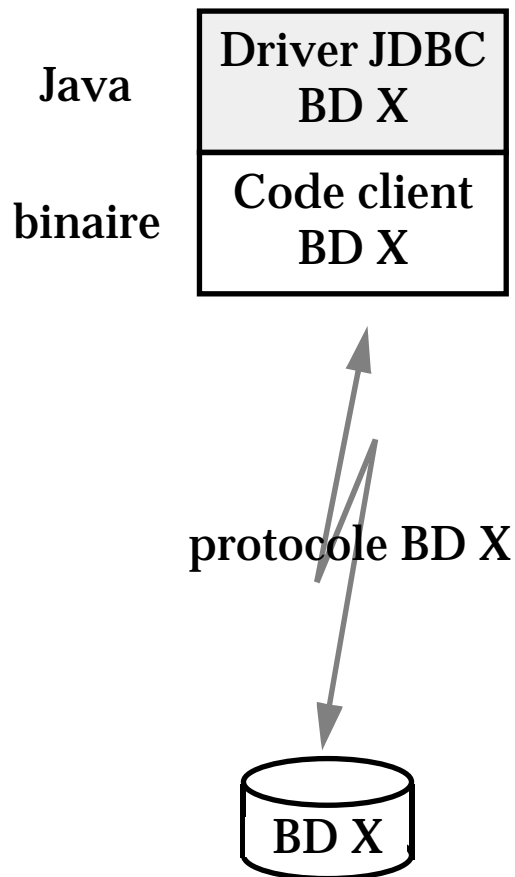
JDBC-ODBC Bridge



- **Client 'épais'**
 - ODBC + Driver ODBC du SGBD
 - Bibliothèques client du SGBD
- **Application Java, mais pas applet**
 - Fichiers de config de ODBC
 - Binaire non portable
- **Utilisation**
 - Test, 1ère implantation de JDBC
 - Middle-tier (Accès à pls. BD)

Driver JDBC Type 2

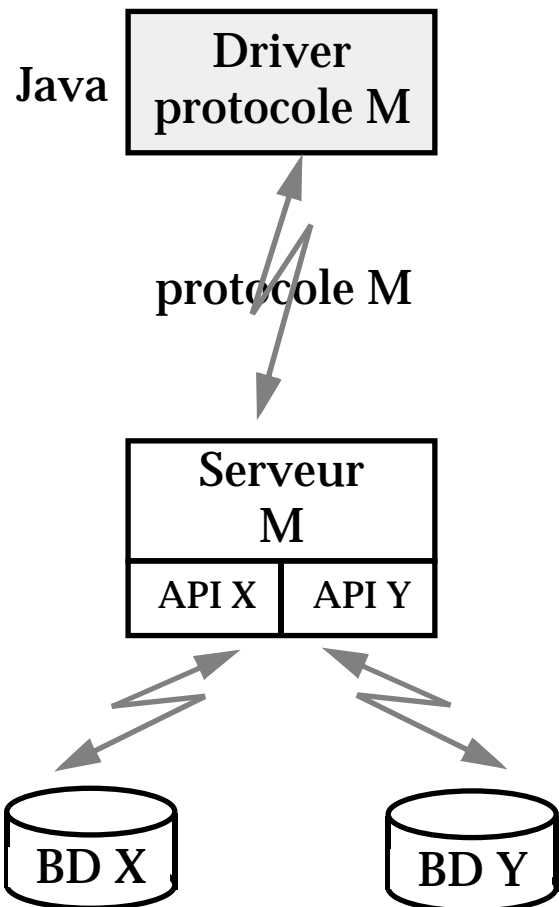
Native-API Partly-Java Driver



- **Client 'épais'**
 - Driver traduit les appels JDBC en appels à l'API du SGBD
 - Bibliothèques client du SGBD
- **Avantage**
 - Protocole BD X optimisé
- **Utilisation**
 - Applications Java 2-tier
 - Middle-tier (Accès efficace)

Driver JDBC Type 3

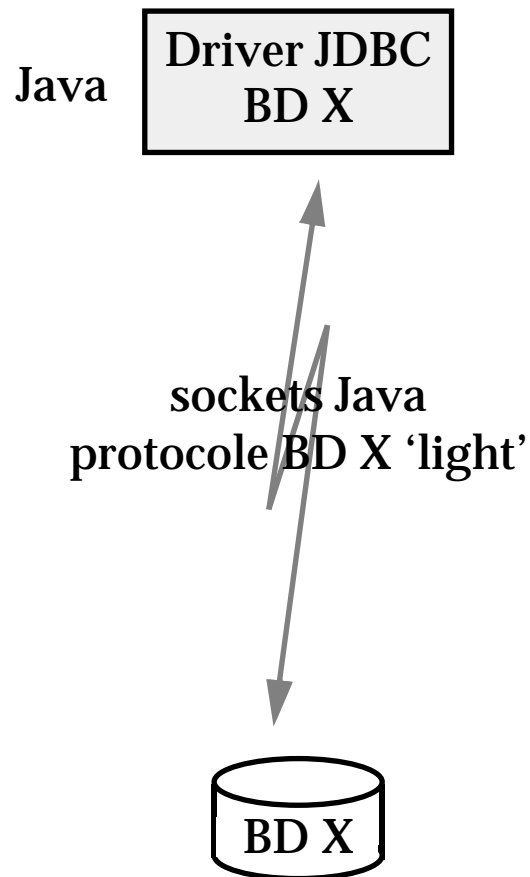
Net Protocol All-Java Driver



- **Client 'fin'**
 - Driver traduit les appels JDBC en appels au middleware
- **Applets ou applications Java**
 - Driver chargé avec le code Java
- **Utilisation**
 - Nécessite un middle-tier (accès BD)
 - Solution 'tout Java' pour architecture 3-tier

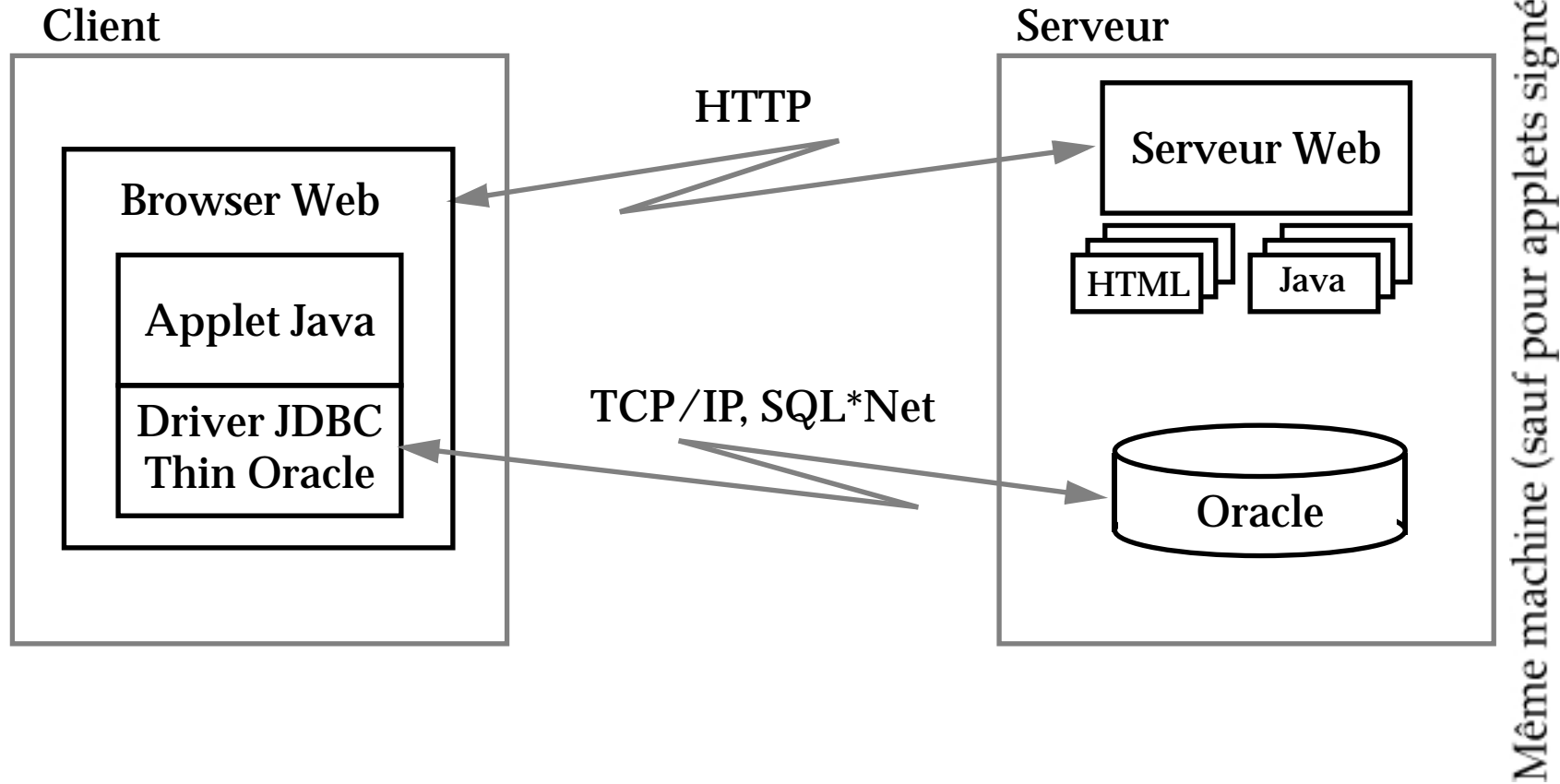
Driver JDBC Type 4

Native Protocol All-Java Driver



- **Client 'fin'**
 - Driver traduit les appels JDBC en appels à l'API du SGBD
 - Driver crée des sockets pour émuler le protocole BD X 'light'
- **Avantage / Désavantage**
 - Tout Java = portable
 - Tout Java = charger Driver (~300K)
- **Utilisation**
 - Applets en architecture 2-tier

Applet et Driver Type 4: exemple



Source: M.P. Mesaros, Oracle

JDBC: avantages - désavantages

- **Avantages**

- Fait partie de core Java
- API complet pour SQL dynamique
- Nombreuses fonctions de conversions
- Support des types 'BLOB': streams Java
- Multiples drivers: configurations 2-tier, 3-tier, etc.

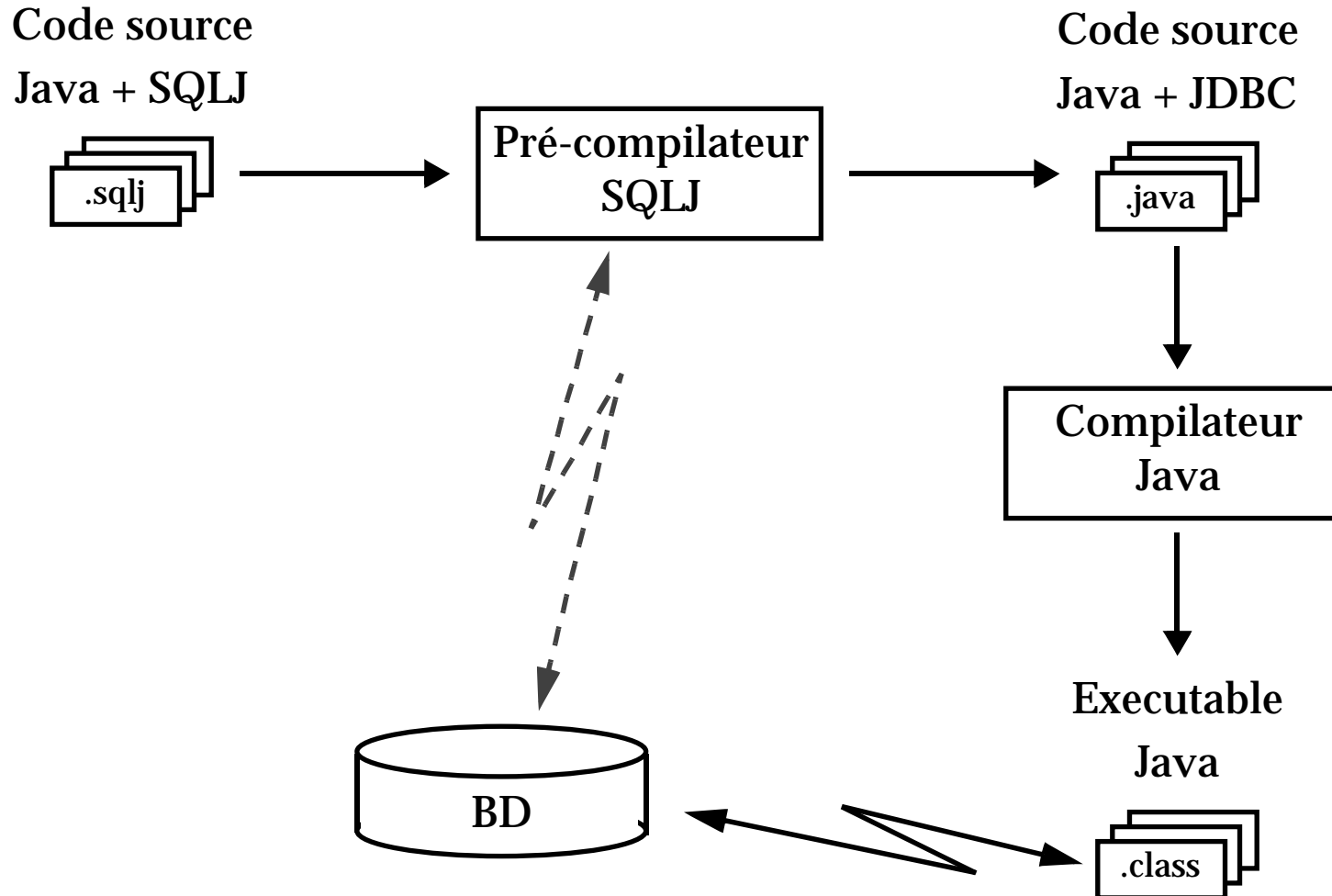
- **Désavantages**

- Pas de vérification de type vis à vis de la BD
- Ecriture fastidieuse, code peu lisible
- Résultats ne sont pas des objets Java

SQLJ: Introduction

- **SQL 'embarqué' dans Java (embedded SQL)**
- **Proposition de norme par IBM, Oracle, Sybase, Tandem**
- **Principe**
 - **Clauses #sql dans le code source Java**
 - **Etape de pré-compilation avec accès au schéma de la BD**
 - * **vérification, optimisation du SQL**
 - * **génération de classes Java (connexions, résultats)**
 - * **génération du code JDBC détaillé**
- **Pré-processeur est écrit en Java**
- **Complémentaire à JDBC**

Développement en SQLJ



Buts de SQLJ

- **Faciliter l'accès aux BDs depuis Java**
 - SQL statique (requêtes formées avant l'exécution)
 - Code simplifié, plus concis
- **Vérifier le code SQL avant l'exécution**
 - Syntaxe des ordres SQL
 - Compatibilité des types Java et SQL
 - Conformité avec le schéma de la BD (option)
- **Créer du code 'mobile': exécution sur chaque tier**
- **Supporter la composition (résultats sont des objets)**

Principaux concepts de SQLJ

- **Clauses SQLJ: expressions SQL**

```
#sql { UPDATE EMP SET SAL = 9999 WHERE DEPT = :d } ;
```

- **Variables ‘hôtes’: variables Java utilisées dans des clauses**

```
String d = "ventes" ;
```

- **Connection Context: paramètres de connexion à une BD (ex: utilisateur, session, transaction)**

```
int maxi;
```

```
#sql context bdParis ;
```

```
#sql [bdparis] (SELECT MAX(SAL) INTO :maxi FROM EMP);
```

- **Itérateurs: ‘encapsulation’ des résultats de requêtes**

Itérateurs en SQLJ

“transformer les résultats de requêtes en objets Java”

- **A chaque ‘format’ de résultat on associe un itérateur**

```
#sql iterator NomSalaire (String nom, int salaire);
```

- **A chaque requête on associe une instance d’itérateur**

```
NomSalaire salventes; // salaires du dépt. ventes
```

```
#sql salventes = { SELECT ENAME, SAL FROM EMP WHERE ... };
```

- **Une fois la requête exécutée, on accède aux méthodes de cet objet itérateur**

```
while (salventes.next()) {  
    System.out.println("Nom: " + salventes.nom() +  
        "Salaire: " + salventes.salaire()); }  
}
```

Traduction SQLJ-JDBC

- **Clauses SQLJ: traduites en JDBC**
 - Accès aux variables hôtes
 - Conversion entre types SQL et Java
- **Connection Context: traduites en Connections JDBC**
- **Itérateurs: traduits en classes Java**
 - Génération des méthodes d'accès aux 'colonnes'
 - Génération de la méthode next()
 - Code Java d'instanciation de l'itérateur
- **Vérifications de types**

Avantages de SQLJ

- **Clauses de ‘haut niveau’, code lisible**
- **SQLJ c’est du Java: portabilité, etc.**
- **Qualité du code: génération ‘correcte’, optimisation**
- **Vérifications avant l’exécution (y compris avec la BD)**
- **Résultats (itérateurs) sont des objets, ils peuvent être échangés entre composants Java**

SQLJ: présent et futur

- **Promoteurs leaders du marché**
- **Proposition de norme soumise à ANSI / ISO**
- **Implémentation de référence disponible (cf. Oracle)**
- **Langage d'accès aux BD depuis chaque tier**
 - Côté client: génération depuis IDE
 - Côté serveur: procédures stockées compilées pour la BD
ex: futur langage 'natif' d'Oracle (comme PL/SQL)
- **Versions 'visual' de SQLJ dans les futurs IDE**