Java et les bases de données

Michel Bonjour

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

CENTRE UNIVERSITAIRE D'INFORMATIQUE

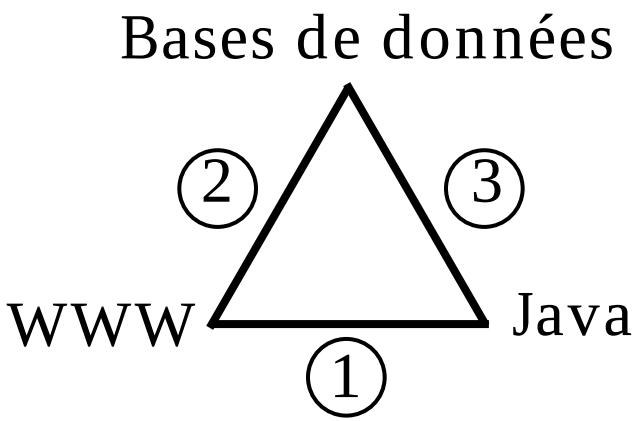
UNIVERSITE DE GENEVE

Plan

- **Introduction**
- **JDBC: API SQL pour Java**
  - JDBC, Java, ODBC, SQL
  - Architecture, interfaces, exemples
- **Java et le client-serveur**
  - Architecture “classique”
  - Architecture revisitée: Java côté client, Java côté serveur
- **Persistence d’objets Java**
  - Principes
  - Exemple de produit: JRB

Java et les BD: Quel intérêt ?

- **Idem que pour les applications Internet (1)**
  - Portabilité, distribution, accès au réseau, GUI
- **Evolution: statique/dynamique (2)**
  - Problèmes: transactions, pré-traitements, etc.
- **Le client-serveur revisité (3)**
  - Client “fin”, code mobile



source: J. Guyot

JDBC: Introduction

- **Quoi ?**
  - API Java pour interagir avec des BD relationnelles
    * exécuter des requêtes SQL
    * récupèrer 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

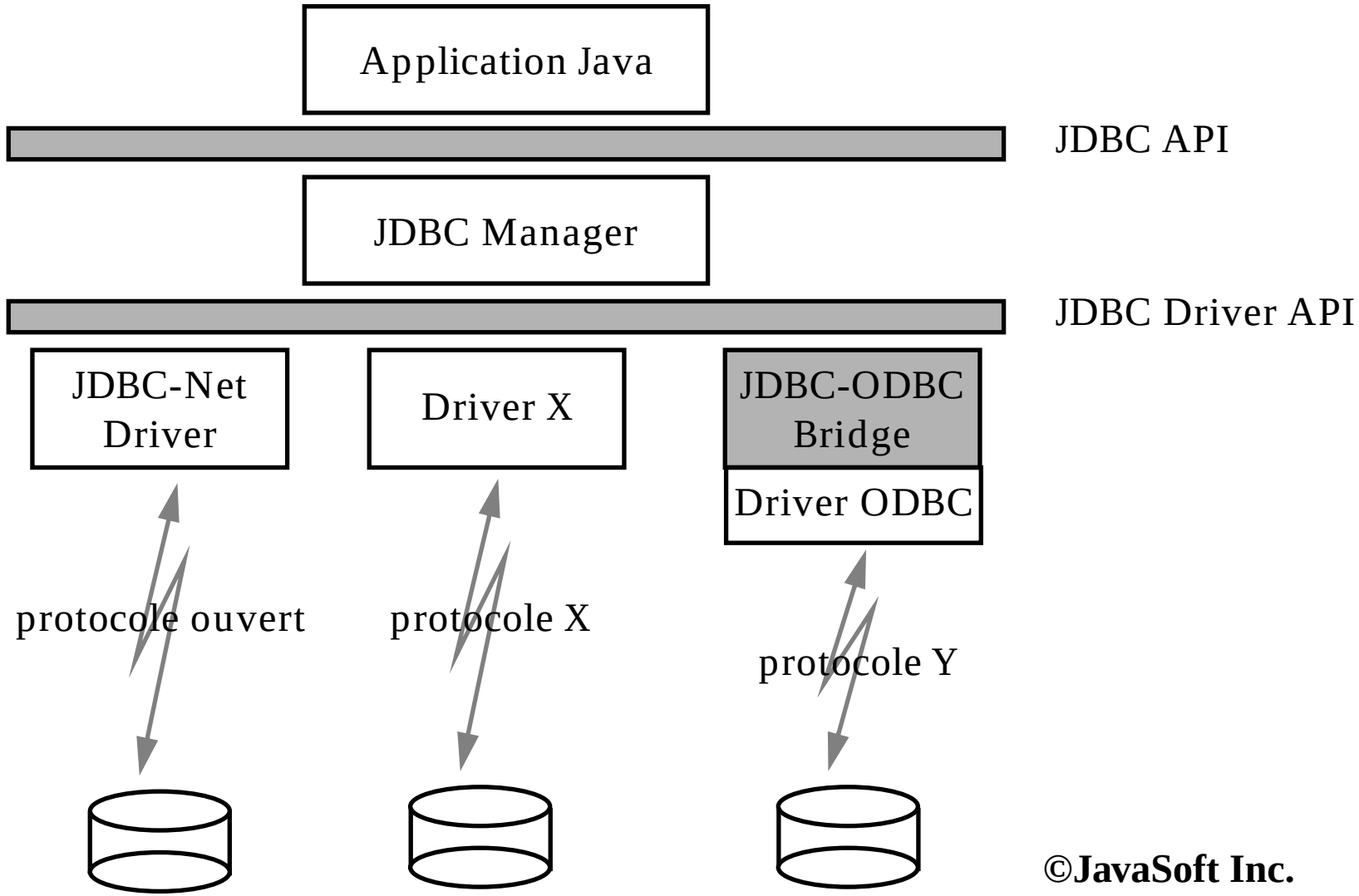
JDBC et ...

- **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

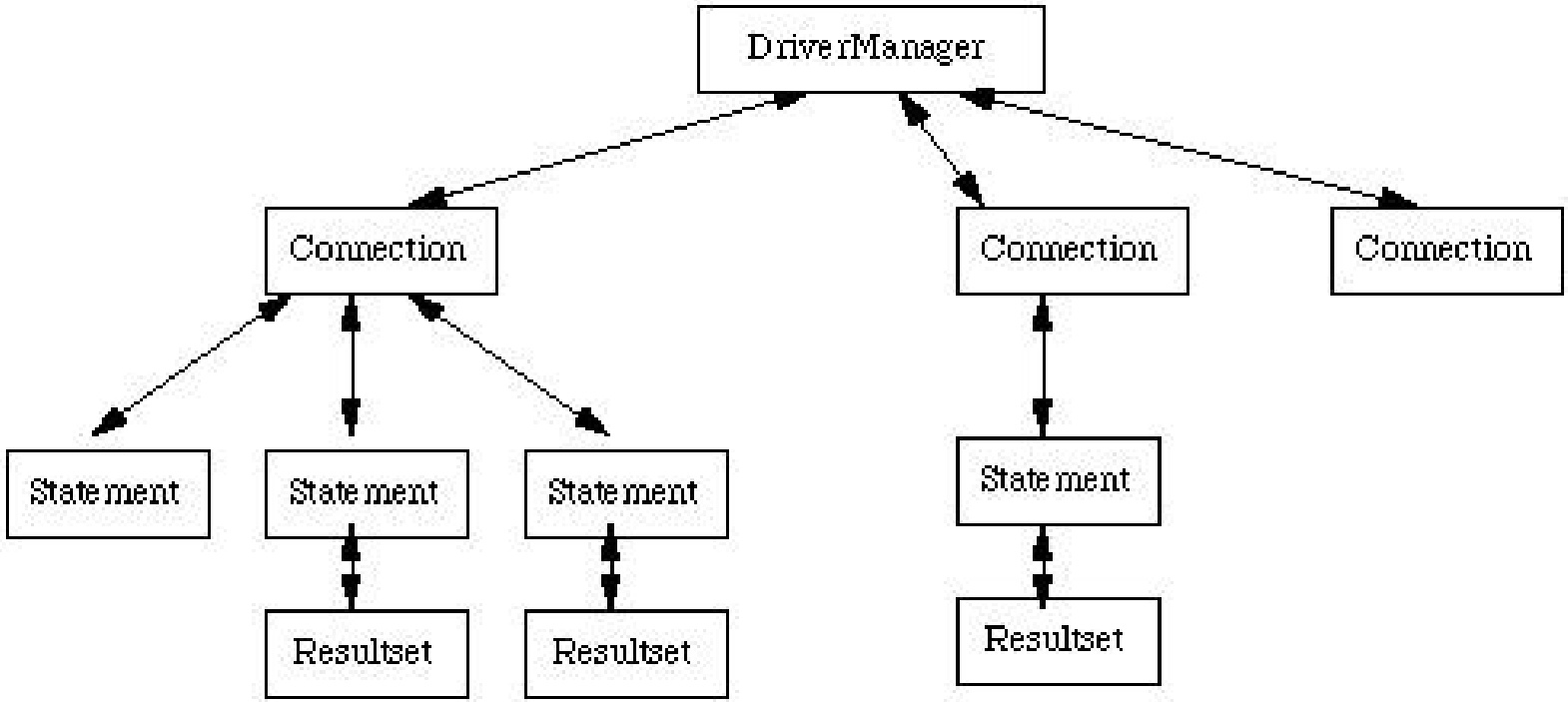
- **Support de SQL-2 Entry Level**
  - SQL dynamique, principaux types SQL
  - Transactions, curseurs simples
  - Méta-données
- **Mécanismes d'extension**
  - Syntaxe inspirée d'ODBC: `{ mot-clé ... paramètres ... }`
  - Fonctions ODBC de conversion, mathématiques, etc.
- **Long terme**
  - Support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

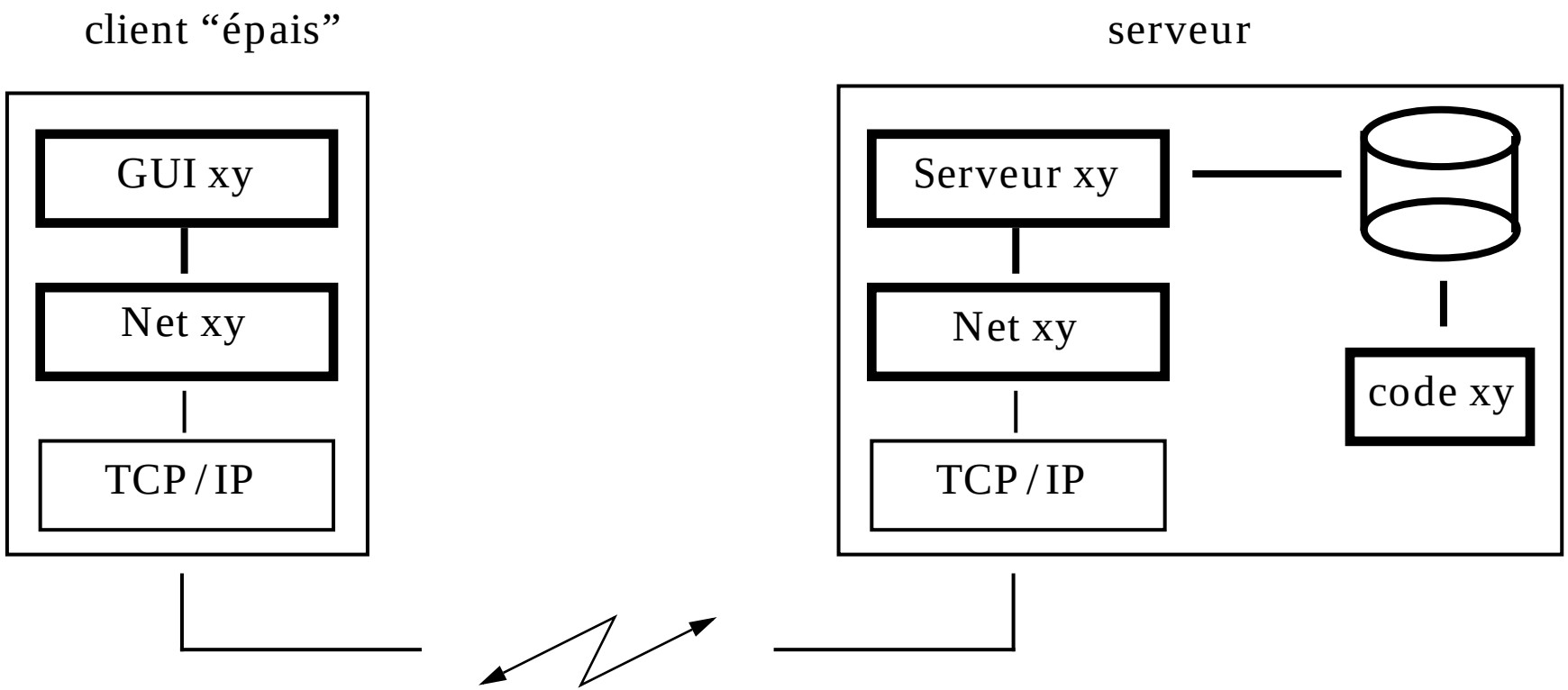
```
java.sql.Statement st = conn.createStatement();
ResultSet r=st.executeQuery(“SELECT nom, age FROM T1”);
while (r.next()) {
  // imprime les éléments du tuple
  String s = r.getString(“nom”);
  int i = r.getInt (“age”); // ou bien r.getInt (2)

  System.out.println (“nom: “ + s + “ age: ” + i);
}
```

Exemple de UPDATE

```
 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);
 }
```

Client-serveur: Architecture “classique”

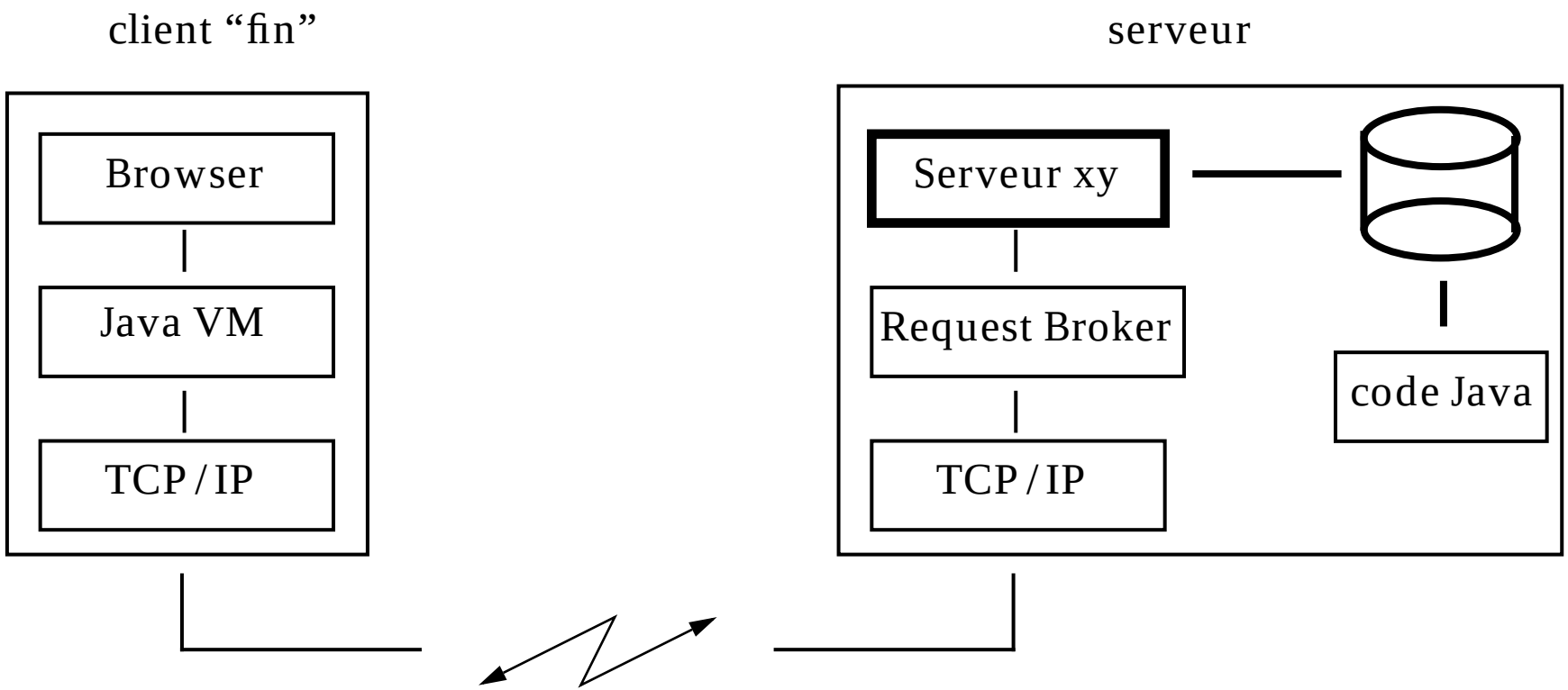


xy: Oracle, Sybase, etc.

Limites du client-serveur “classique”

- **Côté client:**
  - GUI: “moche”, non-standard, runtime
  - Net: protocole fermé, mais: sécurité, performances
  - “Epaisseur”: installation, configuration “câblée”
  - Développement: langage “client”, peu portable
- **Côté serveur**
  - Net: protocole fermé, mais: répartition, homogénéité
  - Développement: langage “serveur”, pas portable

Client-serveur: Architecture “avec Java”



xy: Oracle, Sybase, etc.

Avantages du client-serveur “avec Java”

- **Côté client:**
  - Browser: interface uniforme, bookmarks, plug-ins
  - Java VM: sécurité
  - “Finesse”: TOUT se télécharge, pas de config locale
  - Développement: Java, HTML (gare à JavaScript !)
- **Côté serveur**
  - Request Broker: centralise les appels aux services
  - Développement: Java
    * local: code exécuté par le serveur
    * mobile: code stocké dans la BD et envoyé au client

Persistence d'objets Java: Principes

- **Stocker des objets Java dans un SGBD**
  - Performance, sécurité, concurrence, etc.
- **Développement**
  - Création des classes Java
  - Génération +/- automatique:
    * Java: méthodes (binding)
    * SGBD: tables
- **Différence de paradigme: OO vs Relationnel**
  - Java: classes, types simples1, références, arrays
  - Relationnel: tables, types simples2, jointures

JRB: Java Relational Binding

O_2 Technology, Oct. 1996

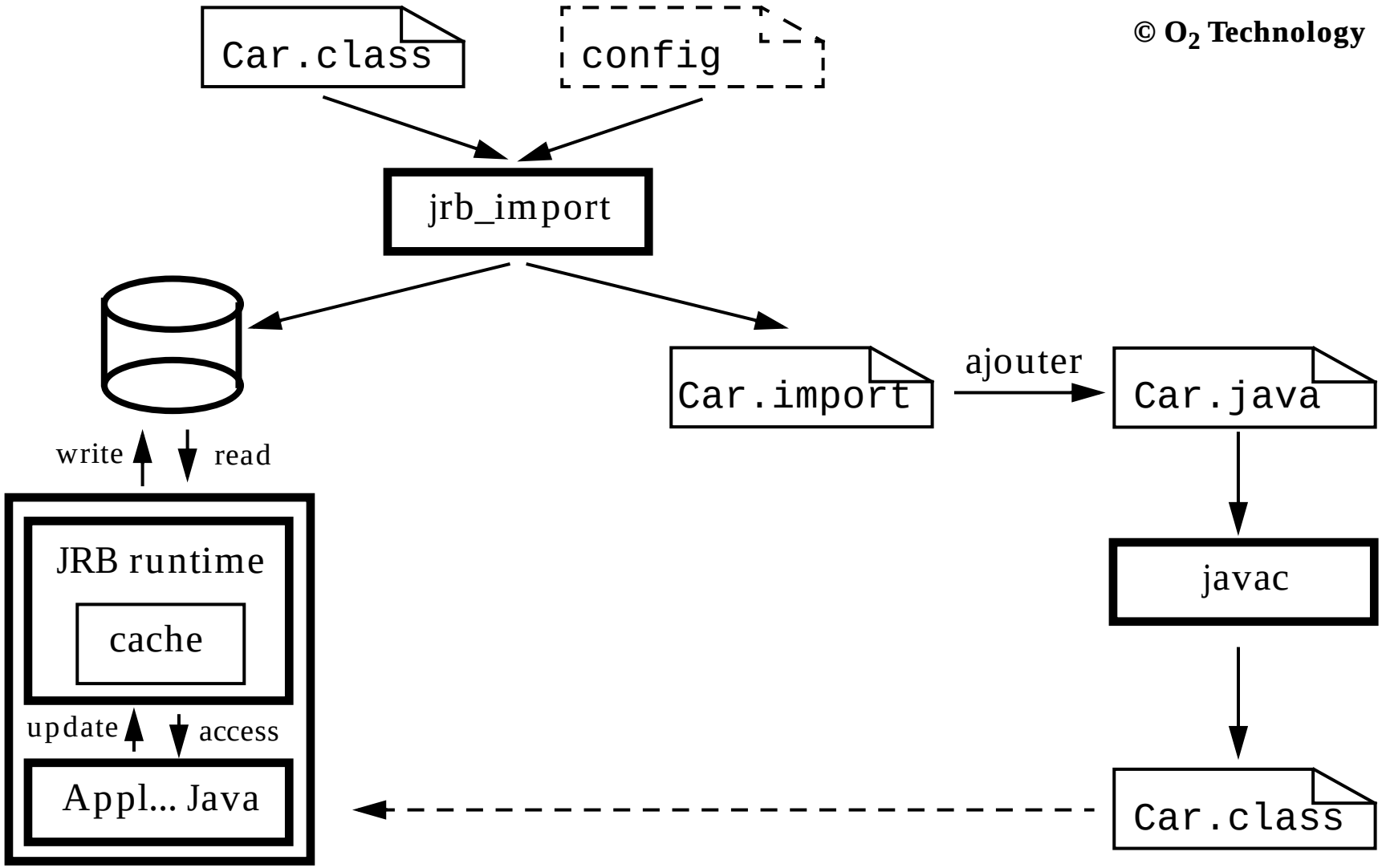
Stockage d'objets Java dans un SGBD relationnel

- **API + outils**
- **Code Java**
  - Voit un système persistant
  - Fait appel à l'API JRB
- **Runtime**
  - Gère un cache d'objets
- **Communication avec le SGBD**
  - Interface compatible JDBC

Etapes du développement

- **Programmation + configuration**
  - Ecriture des classes Java, compilation
  - Choix des classes / variables à stocker
- ***Importation des classes***
  - *Génération de méthodes Java (access, update)*
  - *Création de la BD (tables, clés, index)*
- **Mise à jour du code source**
  - Inclusion des méthodes générées
  - Ecriture des méthodes d'accès 'profond'
- **Compilation, etc.**

Cycle de développement



Gestion des objets

- **Création d'un objet**
  - Méthode *new()* habituelle
- **Ecriture d'un objet**
  - Invocation de *update()* ou de VOTRE *deepUdate()*
- **Lecture d'un objet**
  - Invocation de *access()* ou de VOTRE *deepAccess()*
  - A partir de l'extension de la classe (~ curseur SQL)
  - A l'aide d'un prédicat de recherche
  - En suivant un lien depuis un autre objet stocké

Exemple de création/écriture

```
Person john = new Person("John", 36);
Person mary = new Person("Mary", 32);
Person bob = new Person ("Bob", 6);

john.spouse = mary; // references
mary.spouse = john;

john.children[0] = bob; // 1 child
mary.children[0] = bob;

Database database = new Database(Database.SYBASE,"java_store", "mb","mb_password");

database.connect(); // open a connection to the database server
database.open("base1"); // open the database ìbase1î

Transaction transaction = new Transaction();
transaction.begin() ; // start a new transaction

john.update(); // indicates to the system that these objects must be written in
mary.update(); // the database at commit time
bob.update();

transaction.commit(); // commit the transaction
database.close(); // close the database
database.disconnect();
```

Exemple d’accès

```
Database database = new Database(Databse.SYBASE, "java_store", "mb", "mb_password");
database.connect(); // open a connection to the database server

database.open("base1");

// find John in the database

Person john = (Person) Extent.all("Person").where("name = 'John'").element();

Person mary = (Person) john.spouse.access(); // load his spouse in memory

Transaction transaction = new Transaction();
transaction.begin() ; // start a new transaction

Person bob = (Person) Database.access(john.children,0); // load the 1st child in memory

bob.age = 7; // change his age
bob.update(); // mark it to be written to the database at commit time

transaction.commit(); // commit the transaction

database.close(); // close the database
database.disconnect();
```

Conclusion

- **Avantages**
  - Optimisation: utilisation des possibilités du SGBD
  - Transparence: choix des classes à stocker, appel à l'API
  - Notion de transaction
- **Inconvénients, limites**
  - Processus 'batch'
  - Ecriture des méthodes *deep...()*
  - Support limité de l'évolution des classes (ajout variable)
- **Avenir**
  - Stockage dans O_2, support de l'évolution