Développement d'Application Client-Serveur

Chapitre 2 : Le MIDDLEWARE

Présenté par: **Abdelmajid EL ALAMI**

Définition

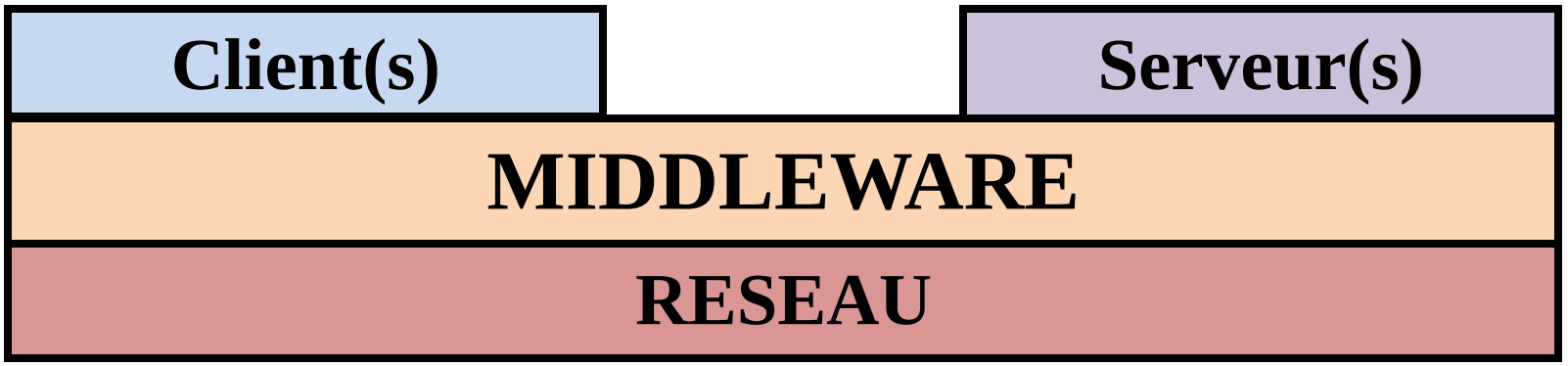
On appelle **middleware** (ou logiciel médiateur en français), littéralement "élément du milieu", l'ensemble des couches réseau et services logiciel qui permettent le dialogue entre les différents composants d'une application **répartie**. Ce dialogue se base sur un **protocole** applicatif commun, défini par **l'API** du middleware.

Autre définition

Georges GARDARIN définit le middleware comme :

"L'ensemble **des services logiciels** construits au-dessus d'un **protocole** de transport afin de permettre l'échange de requêtes et des réponses associées entre client et serveur de manière **transparente**."

Définition



- Une triple transparence :
  - **Transparence aux réseaux**. Tous les types de réseaux doivent être supportés.
  - **Transparence aux serveurs**. Tous les SGBD (avec leur SQL souvent différents) doivent être accessibles.
  - **Transparence aux langages**. Les fonctions appelées doivent être aussi indépendantes que possible des langages.

Pourquoi le Middleware ?

La **complexité** du dialogue client/serveur est à l'origine du middleware.

Complexité due à la présence :

- Des systèmes hétérogènes
- Des systèmes propriétaires
- Du dialogue à distance

Avantages

- Offre des services de «haut niveau» aux applications
- Rend portable les applications (avec certaines limites)
- Prend en charge les protocoles de conversion de caractères et d'établissement de sessions entre clients et serveurs hétérogènes

Les services des middleware

Un middleware est susceptible de rendre les services suivants :

- **Conversion :** Service utilisé pour la communication entre machines mettant en œuvre des formats de données différents.
- **Adressage :** Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès.
- **Sécurité :** Permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes d'authentification et de cryptage des informations.

Les services des middleware

- **Communication :** Permet la transmission des messages entre les deux systèmes sans altération (perte). Ce service doit gérer la **connexion** au serveur, la **préparation** de l'exécution des requêtes, la **récupération** des résultats et la **déconnexion** de l'utilisateur.

➔Le middleware **masque** la complexité des échanges inter-applications et permet ainsi **d'élever** le niveau des API utilisées par les programmes. Sans ce mécanisme, la programmation d'une application client/serveur serait complexe et difficilement évolutive.

L'architecture type du Middleware

- L'**IPC** (**I**nter **P**rocessus **C**ommunication) est l'autre nom du middleware.
- L'IPC se compose :
  - **L'interface API** (**A**pplication **P**rogramming **I**nterface) - Interface de programmation au niveau applicatif. Interface entre un programme et le système qui propose un ensemble de **fonctions standards** pour accéder à un service local ou distant.

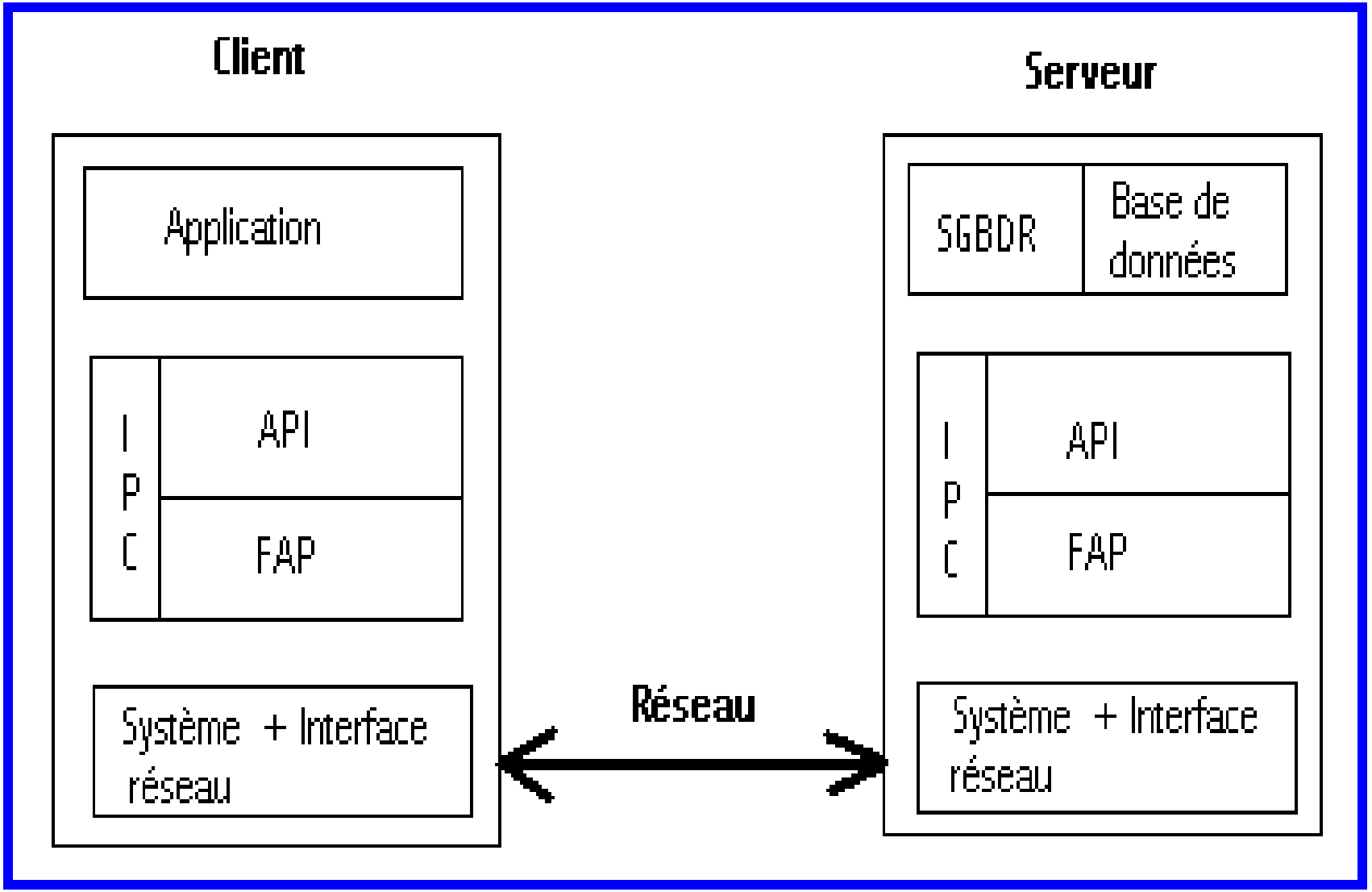
L'architecture type du Middleware

- **L'interface FAP** (**F**ormat **A**nd **P**rotocols) - Protocoles de communication et format des données,

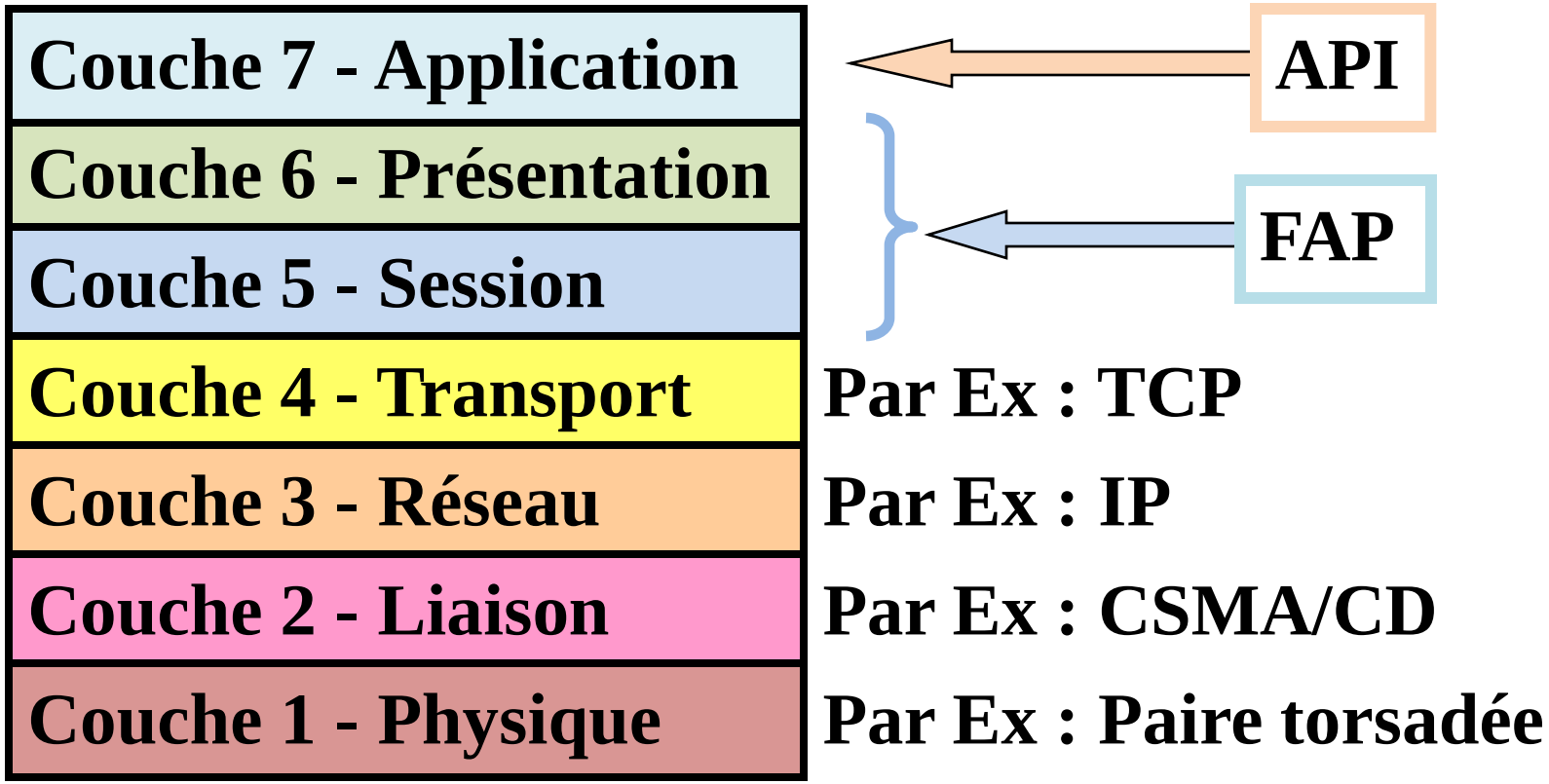
Ce module (IPC) assure :

- La synchronisation entre client et serveur,
- La reconnaissance du format des données échangées
- L'appel aux fonctions de transport du réseau.

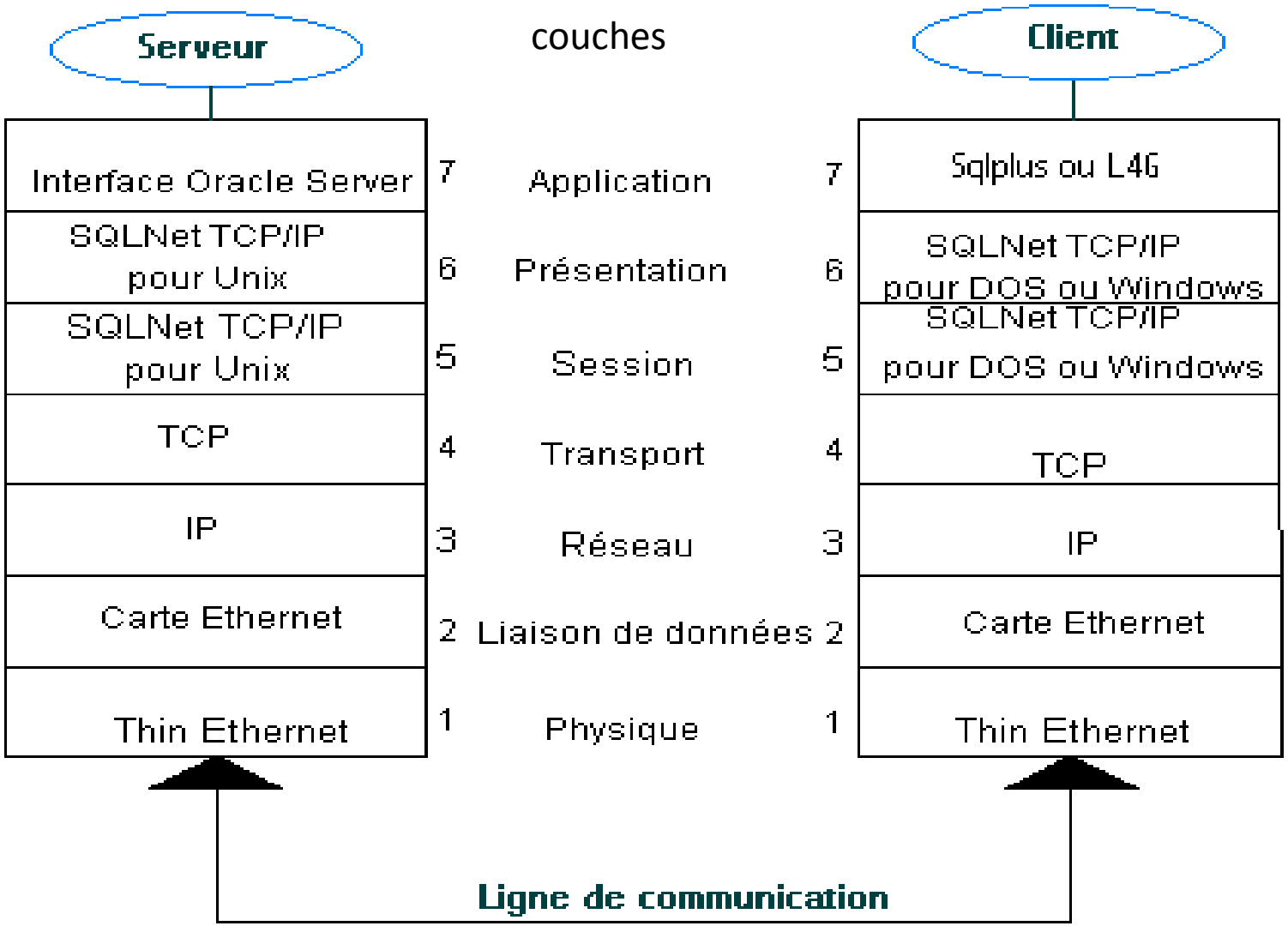
L'architecture type du Middleware



Client-serveur et le modèle OSI



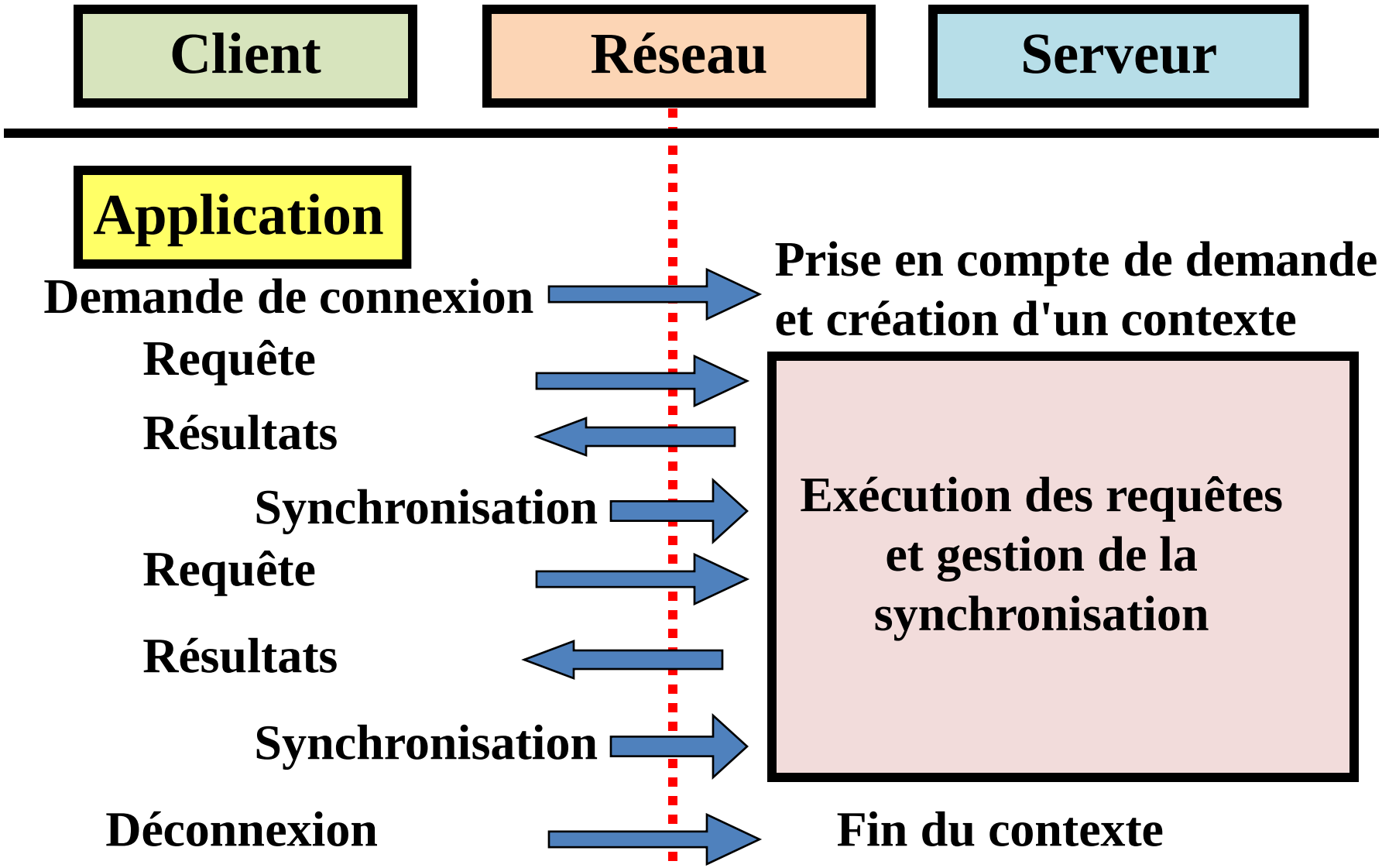
Client-serveur et le modèle OSI



Types de Middleware

- TPM: Transaction Processing Monitor
  - CICS, IMS, TUXEDO BEA,…
- RPC : Remote Call Procedure
- OOM : Object Oriented Middlware
  - RMI, CORBA,…
- MOM : Message Oriented Middleware
  - MQ Series IBM, JMS,…
- SOA : Service Oriented Architecture
  - Web services (jaxws, jaxrs), ESB,…

Le dialogue avec session



Le dialogue avec session

- Dans les **dialogues avec session** (ou avec connexion). Les échanges d'informations sont subordonnés à l'ouverture d'une «session» par le client **vers** le serveur.
- **Exemples** d'IPC (Middleware) avec connexion :
  - Protocole **APPC** (Advanced Program-to-Program Communications) de l'architecture réseau **SNA** (Systems Network Architecture) d'IBM
  - Protocole **RDA** (Remote Data Access), basé sur SQL défini par l'ISO pour l'accès distant aux BDD

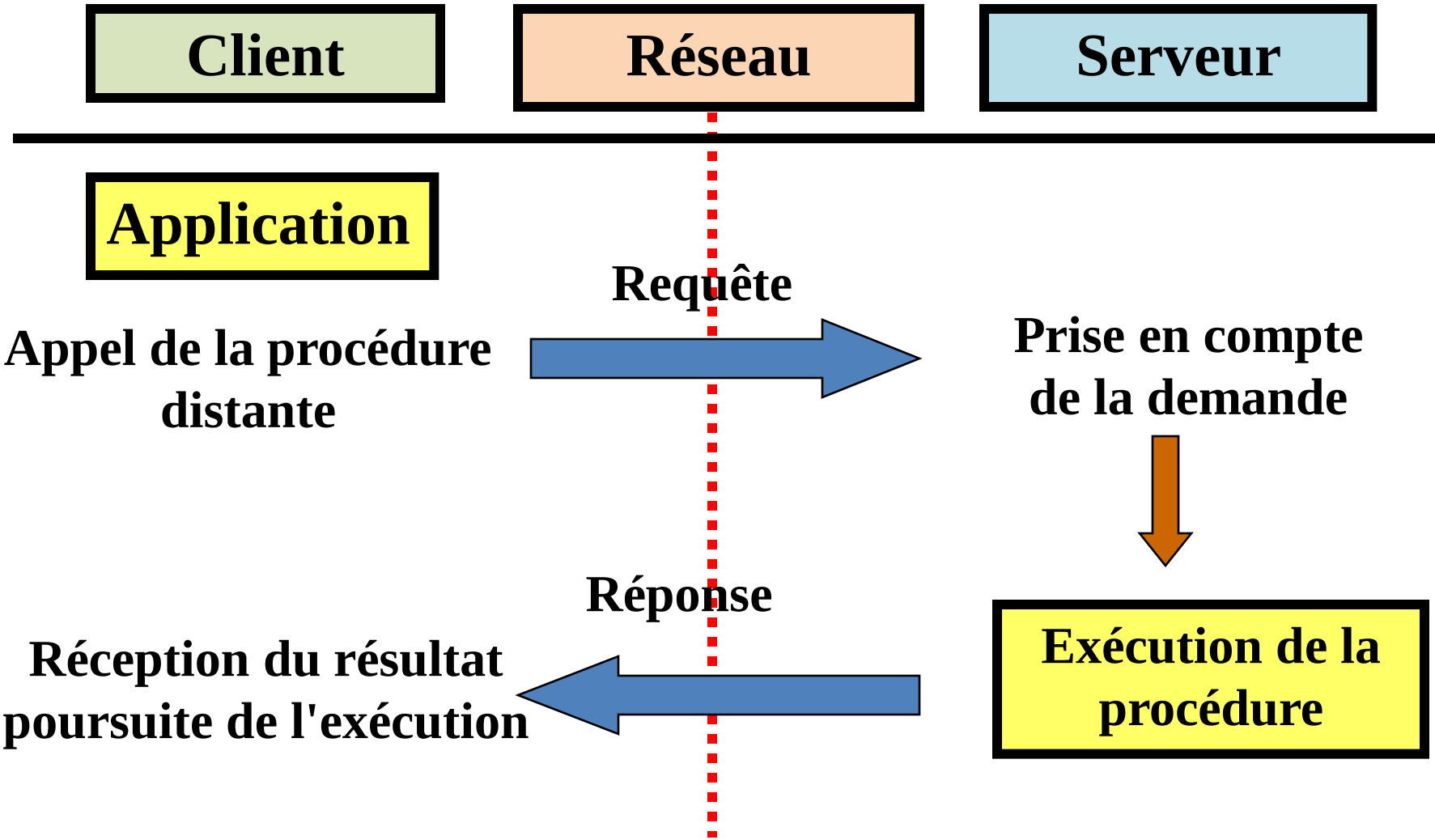
Le dialogue avec session

- Si le serveur accepte la connexion, il crée un **contexte** propre à chaque application cliente connectée.
- Client et serveur s'échangent des **requêtes,** des **réponses** et des **points de synchronisation**.
- Le client a la responsabilité de conduire les phases successives de l'échange
- Le serveur a la responsabilité de garantir le contexte perçu par le client.

Le dialogue avec session

- Les ordres SQL "**COMMIT**" ou "**ROLL BACK**" sont des exemples de points de synchronisation.
- A la suite d'une requête le :
  - COMMIT confirmera la transaction,
  - ROLL BACK l'annulera.
- Le serveur mettra réellement à jour la base de données qu'à la suite de ces ordres de synchronisation (avant cela les transactions s'appliquent dans le "contexte")

Le dialogue sans connexion : les RPC



Le dialogue sans connexion : les RPC

Les dialogues sans connexion avec **appels de procédures distantes** (RPC - Remote Procedure Call).

- ✓ Le processus client **invoque** une procédure distante située sur le serveur.
- ✓ La requête contient tous les éléments nécessaires au serveur (nom de la procédure, paramètres, identité du processus).
- ✓ Le message en retour contient toute la réponse.

L’offre Middleware

Les offres Middleware sont variées :

- Offres propriétaires,
- Offres pour des accès multibases,
- Offres d'accès universel aux bases

- Les offres propriétaires aux SGBDR :
  - ORACLE avec Sql*Net
  - SYBASE avec Db-lib

L'offre Middleware

- Les offres multi-clients, multi-serveurs:

Elles permettent aux clients d'accéder **en toute transparence** à plusieurs bases hétérogènes, situées éventuellement sur des serveurs différents.

- SEQUELINK : Technologie propose une API sur presque toutes les architectures clientes ou serveurs
- EDA/SQL : Information Builders propose d'accéder à tout type de bases de données à partir de plates-formes hétérogènes

L’offre Middleware

- DRDA (Distributed Relational Database Architecture) d'IBM pour fédérer les bases IBM (DB2) et non IBM.
- IDAPI (Integrated Database Application Programming Interface) de Borland en collaboration avec Novell et IBM.

Remarque: Évidemment l'accès multibases permet également l'accès monobase.

L'offre Middleware

- L'accès universel aux données pour les clients:
  - ODBC (**O**pen **D**ata **B**ase **C**onnectivity) de Microsoft : accès standardisé aux principales bases de données du marché (drivers)
  - IDAPI (Interactive Database Application Programming Interface) de Borland et Novell

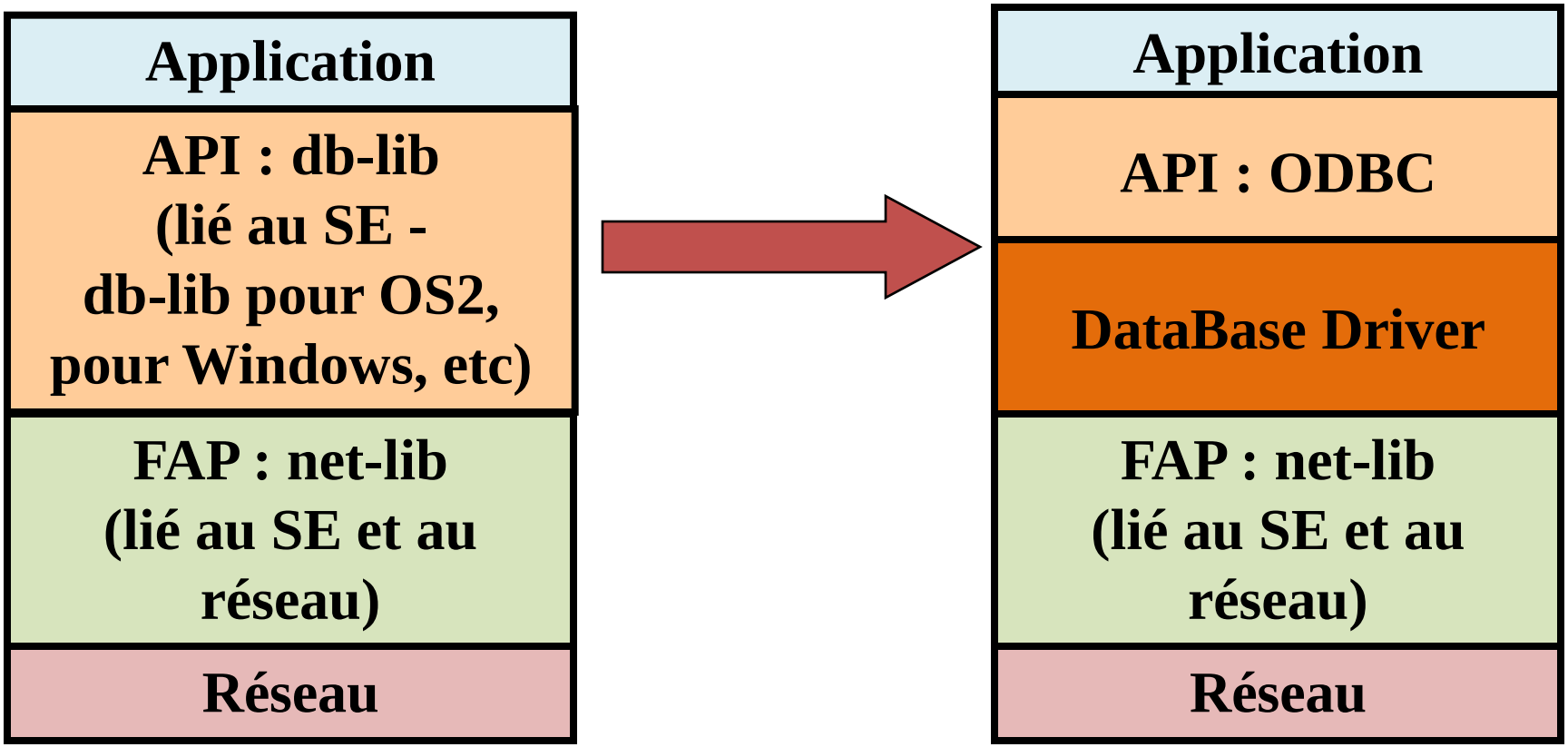
Le Standard ODBC

ODBC (Open DataBase Connectectivity) est présenté en 1992 par Microsoft comme une interface universelle aux bases de données.

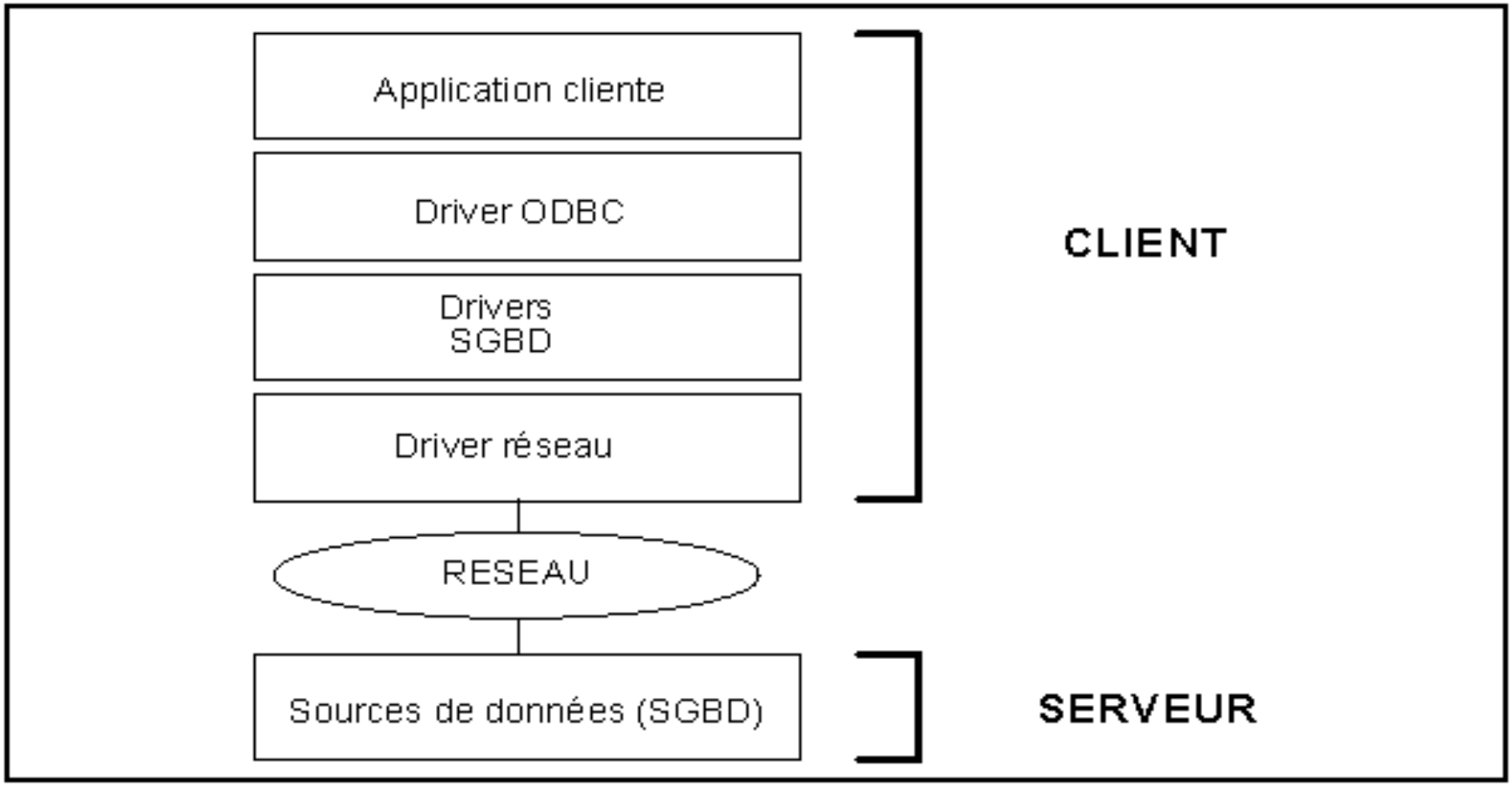
Il permet à une application, par un procédé unique, de manipuler plusieurs bases de données qui sont mises à disposition par des systèmes de gestion de bases de données (SGBD) ayant chacun un procédé propre.

Le Standard ODBC

Exemple : De Sybase à ODBC



Le Standard ODBC



Architecture d'ODBC; schéma de principe

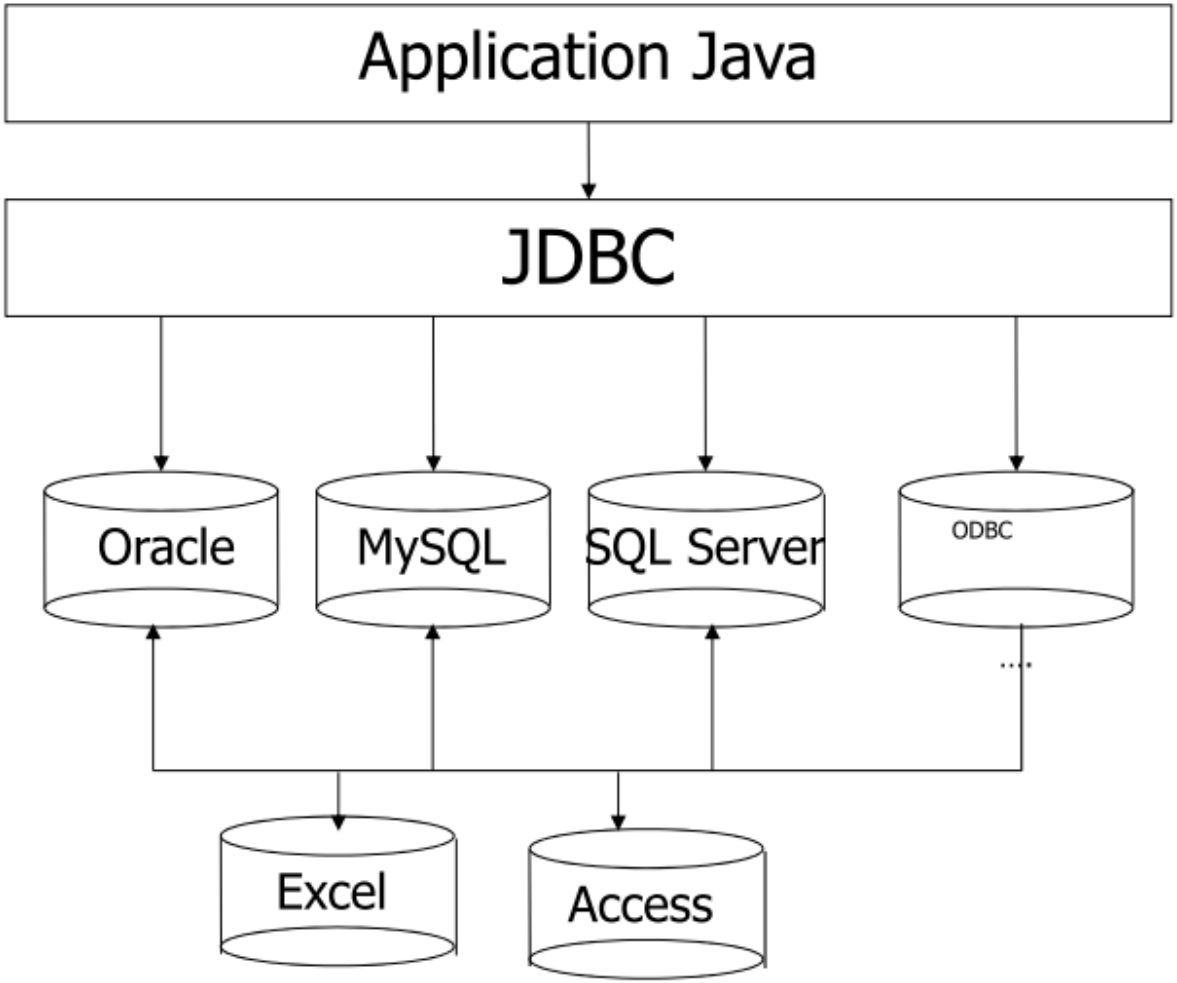
API-JDBC

Pilotes JDBC

- Pour qu'une application java puisse communiquer avec un serveur de bases de données, elle a besoin d'utiliser les pilotes JDBC (Java Data Base Connectivity)
- Les Pilotes JDBC est une bibliothèque de classes java qui permet, à une application java, de communiquer avec un SGBD via le réseau en utilisant le protocole TCP/IP
- Chaque SGBD possède ses propres pilotes JDBC.
- Il existe un pilote particulier « JdbcOdbcDriver » qui permet à une application java communiquer avec n'importe quelle source de données via les pilotes ODBC (Open Data Base Connectivity)
- Les pilotes ODBC permettent à une application Windows de communiquer une base de données quelconque (Access, Excel, MySQL, Oracle, SQL SERVER etc…)
- La bibliothèque JDBC a été conçu comme interface pour l'exécution de requêtes SQL.Une application JDBC est isolée des caractéristiques particulières du système de base de données utilisé.

API-JDBC

JAVA et JDBC



API-JDBC

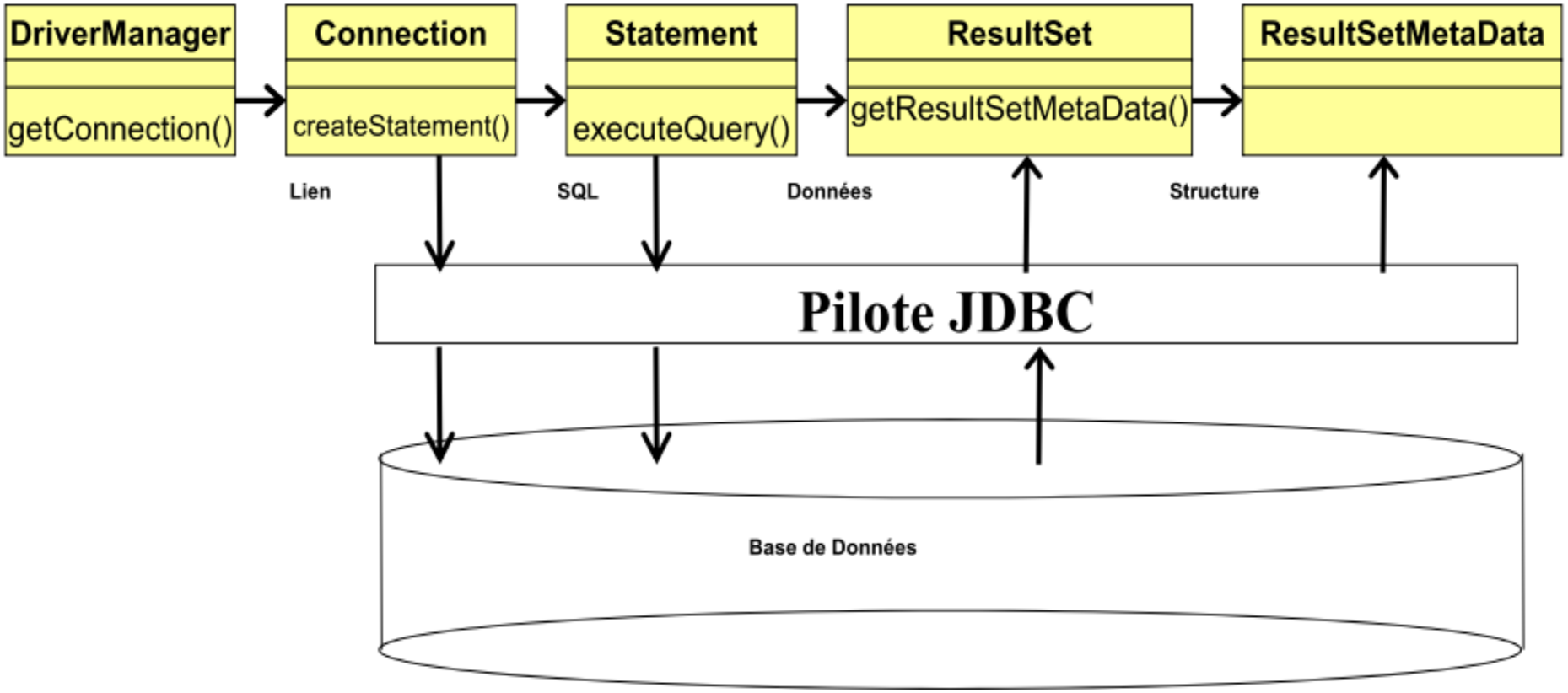
Créer une application JDBC

Pour créer une application élémentaire de manipulation d'une base de données il faut suivre les étapes suivantes :

- Chargement du Pilote JDBC ;
- Identification de la source de données ;
- Allocation d'un objet **Connection**
- Allocation d'un objet Instruction **Statement** (ou **PreparedStatement**);
- Exécution d'une requête à l'aide de l'objet Statement ;
- Récupération de données à partir de l'objet renvoyé **ResultSet** ;
- Fermeture de l'objet ResultSet ;
- Fermeture de l'objet Statement ;
- Fermeture de l'objet Connection.

API-JDBC

Créer une application JDBC



JDO (Java Data Object)

Les principaux buts de JDO sont:

- ❑ **La facilité d'utilisation.**
  - ✓ Modélisation objet sans contrainte : supporte l'héritage, les collections,...
  - ✓ Gestion automatique du mapping des données.
- ❑ **La persistance universelle:**
  - ✓ Persistance vers tout type de systèmes de gestion de ressources (bases de données relationnelles, fichiers, ...). C'est au déploiement de l'application que la liaison avec la cible sera faite.
  - ✓ JDBC est limité au SGBDR, JDO non (en théorie, dépend de l'implémentation)
- ❑ **La transparence vis à vis du système de gestion de ressources utilisé:**
  - ✓ Ce n'est plus le développeur mais JDO qui dialogue avec le système de gestion de ressources (ne nécessite pas la connaissance du schéma de la base, pas de requêtes SQL,...)
- ❑ **La standardisation des accès aux données**
- ❑ **La prise en compte des transactions**

JPA (Java Persistance API)

La Java Persistence API repose essentiellement sur l'utilisation des annotations, introduites dans Java 5. Elles permettent de définir très facilement, et précisément des objets métier, qui pourront servir d'interface entre la base de données et l'application.

La persistance dans ce contexte recouvre 3 zones :

- ✓ L'API elle-même, définie dans le paquetage *javax.persistence*.
- ✓ Le langage Java Persistence Query (JPQL) .
- ✓ L'objet/les métadonnées relationnelles.

JPA (Java Persistance API)

L'API de persistance de Java, JPA, a deux aspects :

- Le premier est la possibilité **d'associer** des objets à une base de données relationnelle. La configuration par exception permet aux fournisseurs de persistance de faire l'essentiel du travail sans devoir ajouter beaucoup de code, mais la richesse de JPA tient également à la possibilité d'adapter ces associations à l'aide d'annotations ou de descriptions XML.
- Le second aspect concerne **l'interrogation** de ces objets une fois qu'ils ont été associés à une base. Élément central de JPA, le gestionnaire d'entités permet de manipuler de façon standard les instances des entités. Il fournit une API pour créer, rechercher, supprimer et synchroniser les objets avec la base de données.

Merci pour votre attention