JDBC

Subject : JAVA PROGRAMMING

SAVIYA VARGHESE
Dept of BCA

2020-21

1.JDBC - INTRODUCTION

- What is JDBC?

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

WHAT IS API?

- API (Application programming interface) is a document that contains a description of all the features of a product or software.
- It represents classes and interfaces that software programs can follow to communicate with each other.
- An API can be created for applications, libraries, operating systems, etc.

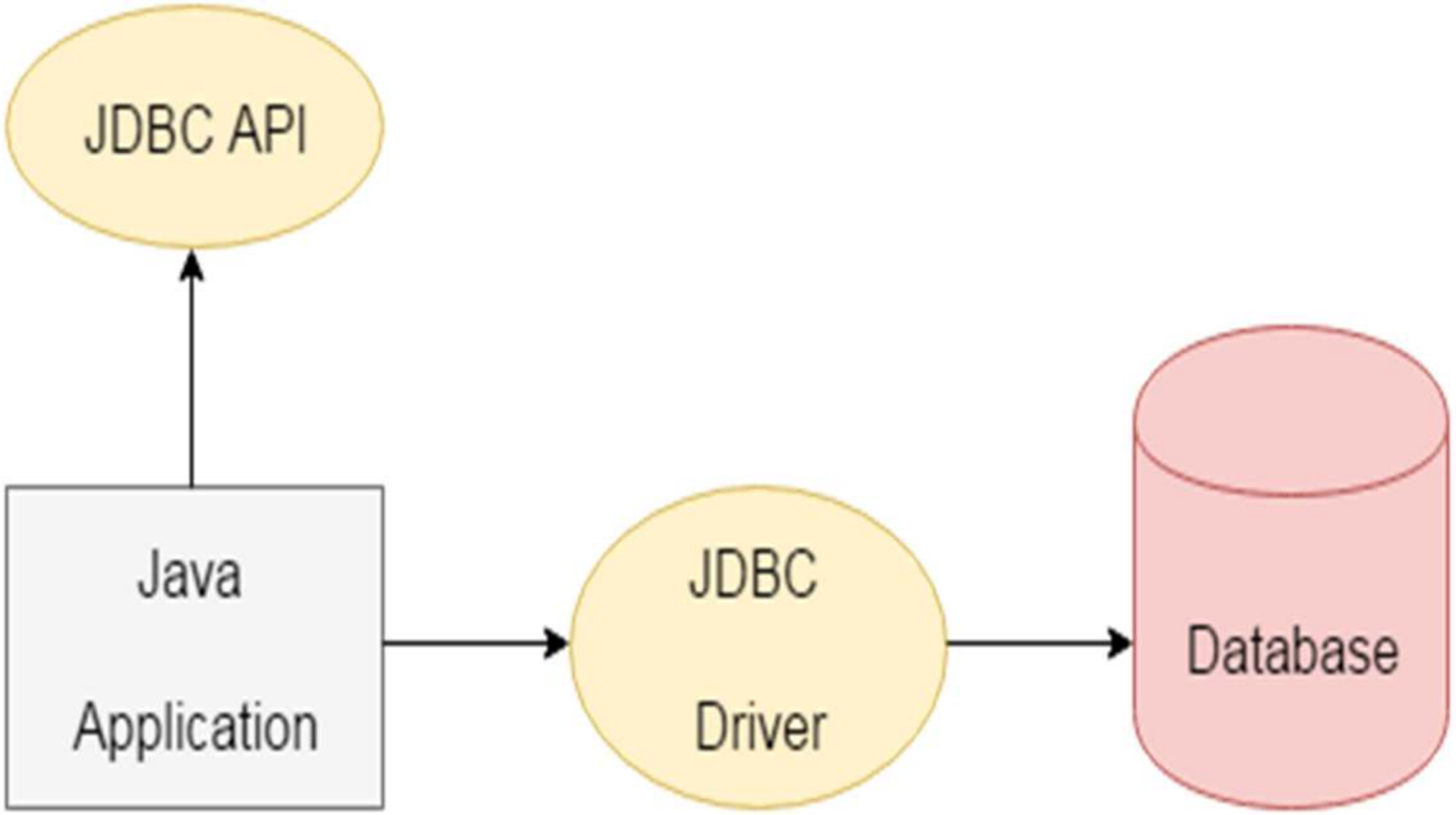
Why should we use jdbc?

- Before JDBC, ODBC API was the database API to connect and execute the query with the database.
- But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).
- That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

JDBC ARCHITECTURE



1.1 JDBC ARCHITECTURE

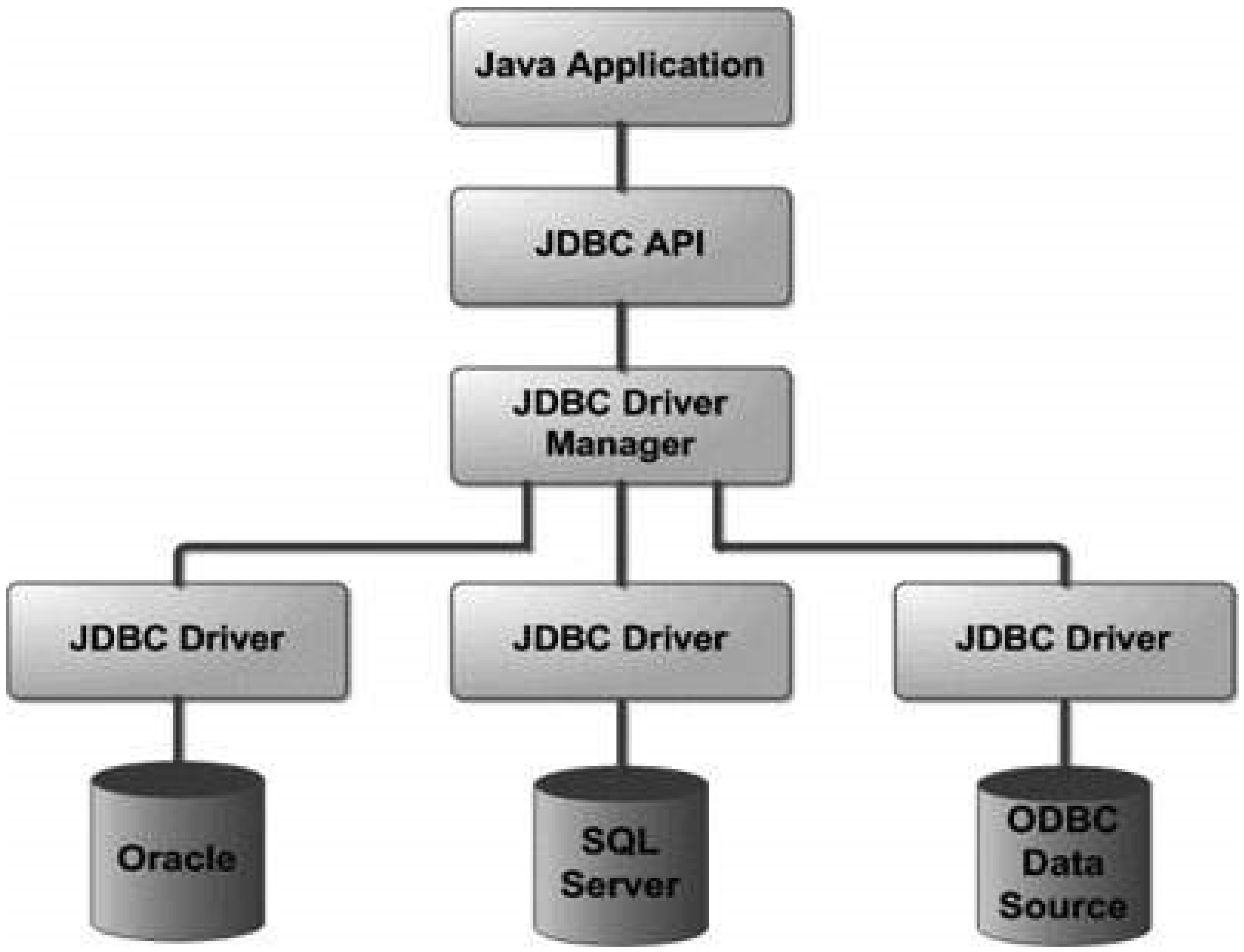
- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

JDBC API & JDBC DRIVER API

- The **JDBC API** uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The **JDBC driver** manager ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

ARCHITECTURAL DIAGRAM, WHICH SHOWS THE LOCATION OF THE DRIVER MANAGER WITH RESPECT TO THE JDBC DRIVERS AND THE JAVA APPLICATION



1.1.1 JDBC API

Common JDBC Components

- The JDBC API provides the following interfaces and classes –

- **DriverManager**
- **Driver**
- **Connection**
- **Statement**
- **ResultSet**
- **SQLException**

DRIVERMANAGER:

- This class manages a list of database drivers.
- Matches connection requests from the java application with the proper database driver using communication sub protocol.
- The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

DRIVER

- This interface handles the communications with the database server.

- Instead of direct interaction with Driver objects ,we use Driver Manager objects, which manages objects of this type.

- It also abstracts the details associated with working with Driver objects.

CONNECTION

- This interface with all methods for contacting a database.

- The connection object represents communication context, i.e., all communication with database is through connection object only.

STATEMENT

- We use objects created from this interface to submit the SQL statements to the database.

- Some derived interfaces accept parameters in addition to executing stored procedures.

ResultSet:

- These objects hold data retrieved from a database after you execute an SQL query using Statement objects.
- It acts as an iterator to allow you to move through its data.

SQLException:

- This class handles any errors that occur in a database application

- The JDBC architecture consists of two-tier and three-tier processing models to access a database.

1.Two-tier model:

- A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source.

  When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.

  The data source can be located on a different machine on a network to which a user is connected. This is known as a **client/server configuration**, where the user's machine acts as a client and the machine having the data source running acts as the server.

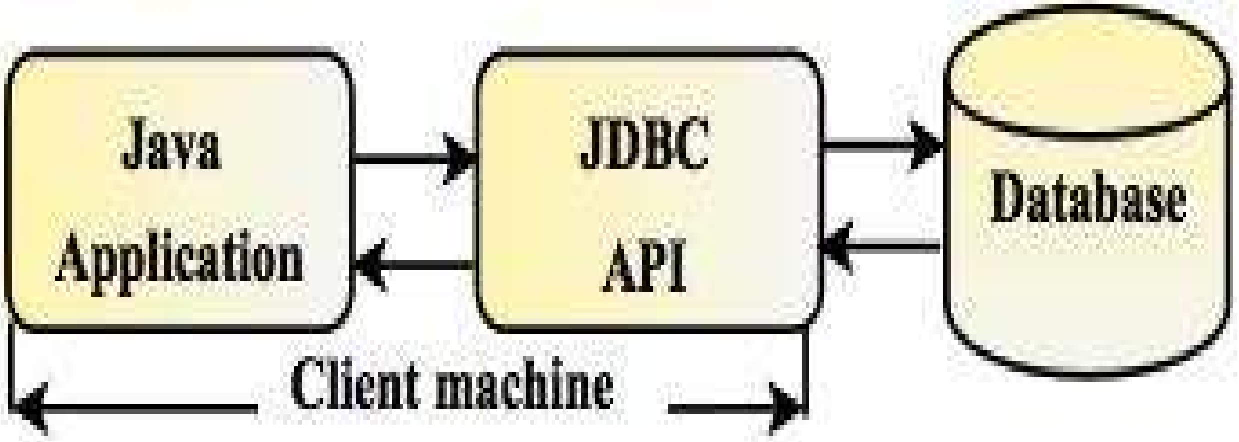


Fig: Two-tier Architecture of JDBC

- **Three-tier model:**

In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data source.

The results are sent back to the middle tier, and from there to the user.

This type of model is found very useful by management information system directors

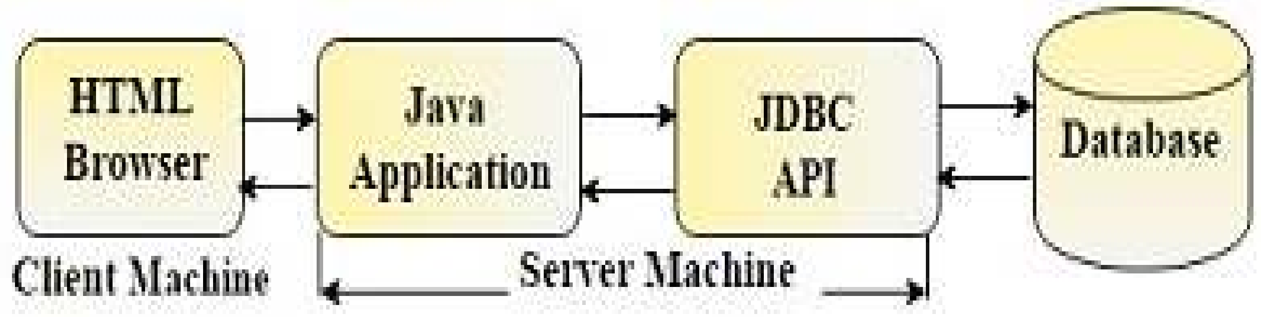


Fig: Three-tier Architecture of JDBC

1.1.2 JDBC DRIVER

- What is JDBC Driver?

  JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

- The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers.

- Third party vendors implements the *java.sql.Driver* interface in their database driver.

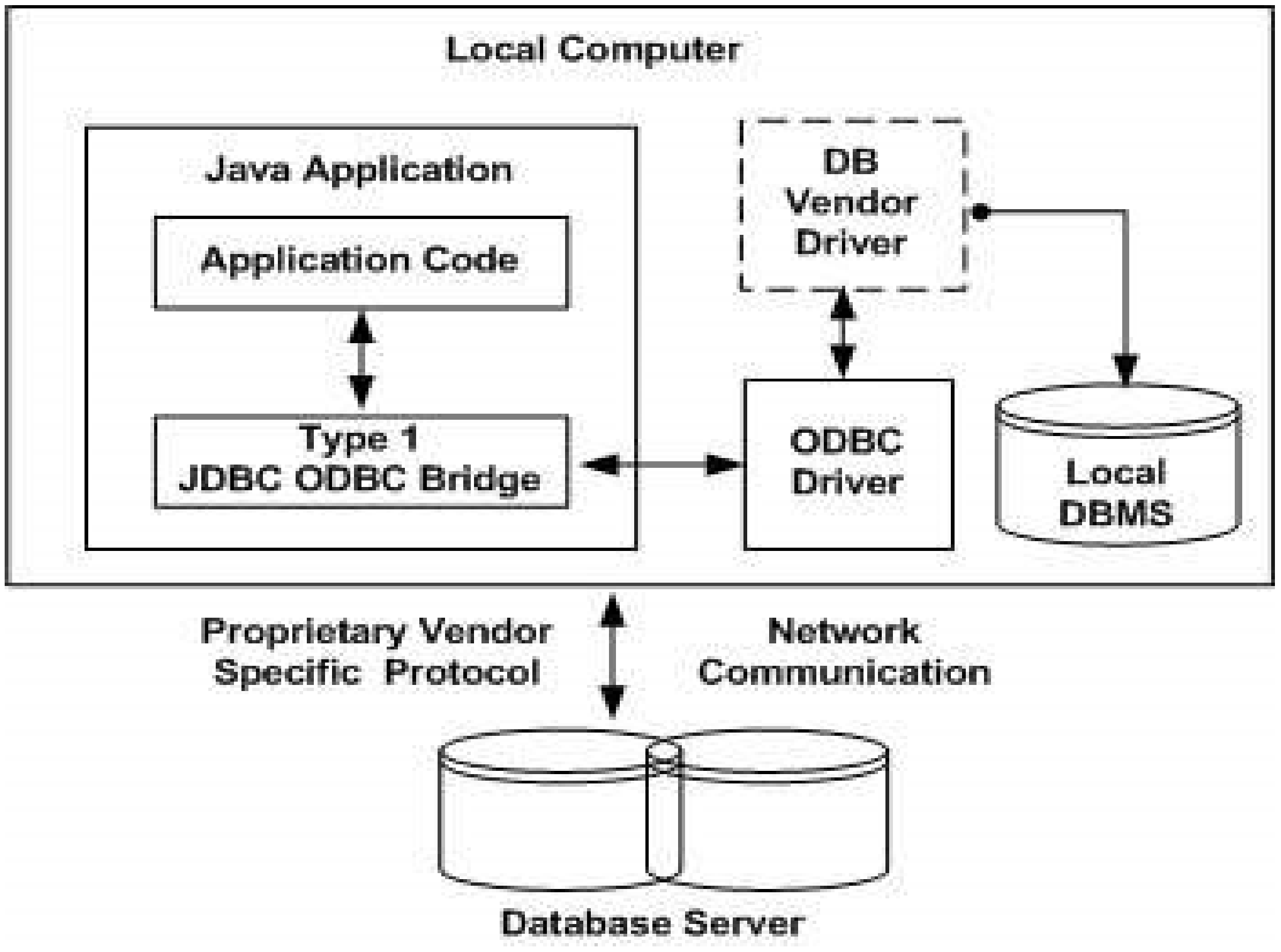
JDBC DRIVERS TYPES

- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates.
- Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4
- Type 1: JDBC-ODBC Bridge Driver
- Type 2: JDBC-Native API
- Type 3: JDBC-Net pure Java
- Type 4: Direct to Database Pure Java Driver

TYPE 1: JDBC-ODBC BRIDGE DRIVER

- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.
- Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
- When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
- The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver

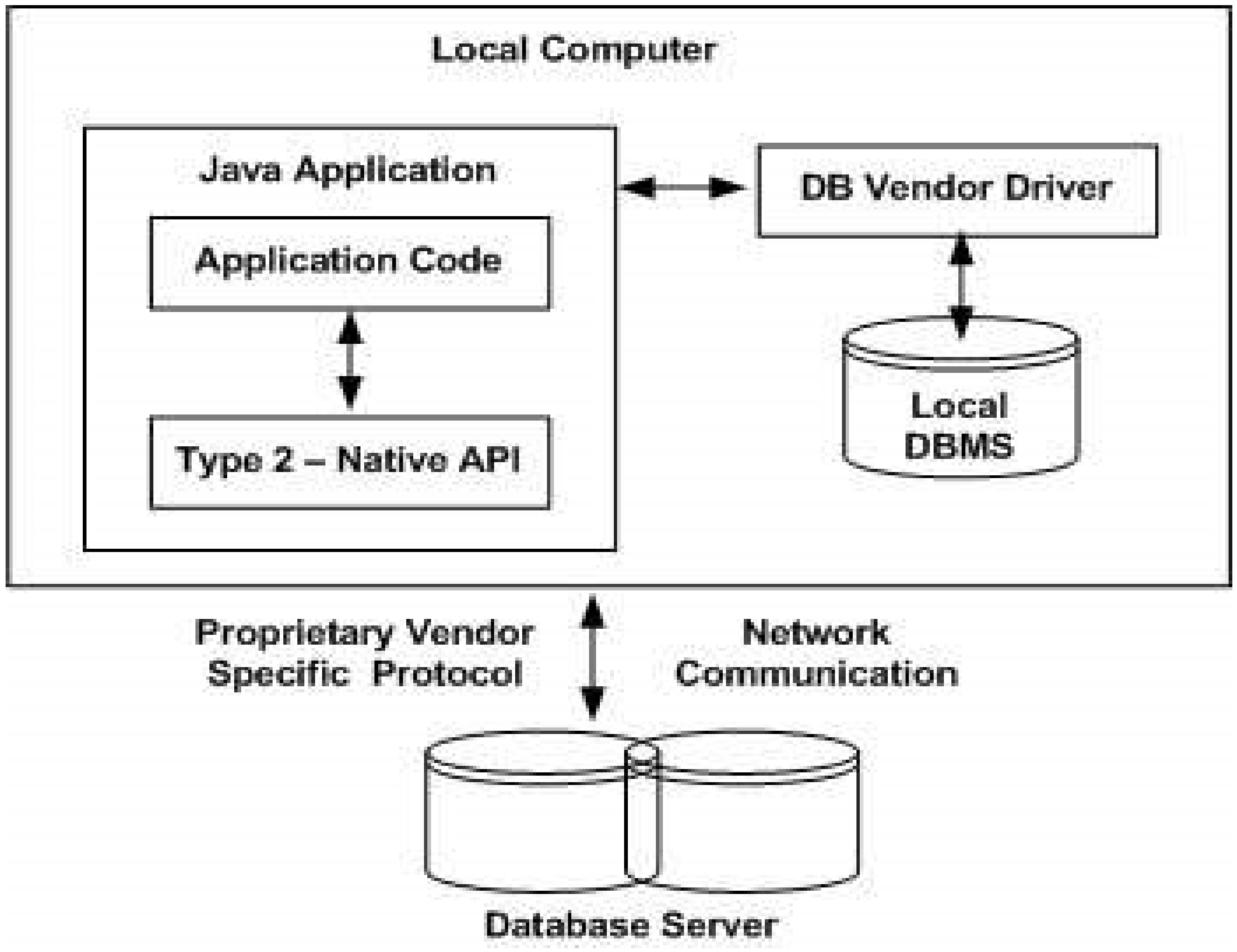
TYPE 1: JDBC-ODBC BRIDGE DRIVER



TYPE 2: JDBC-NATIVE API

- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
- The vendor-specific driver must be installed on each client machine.
- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.
- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

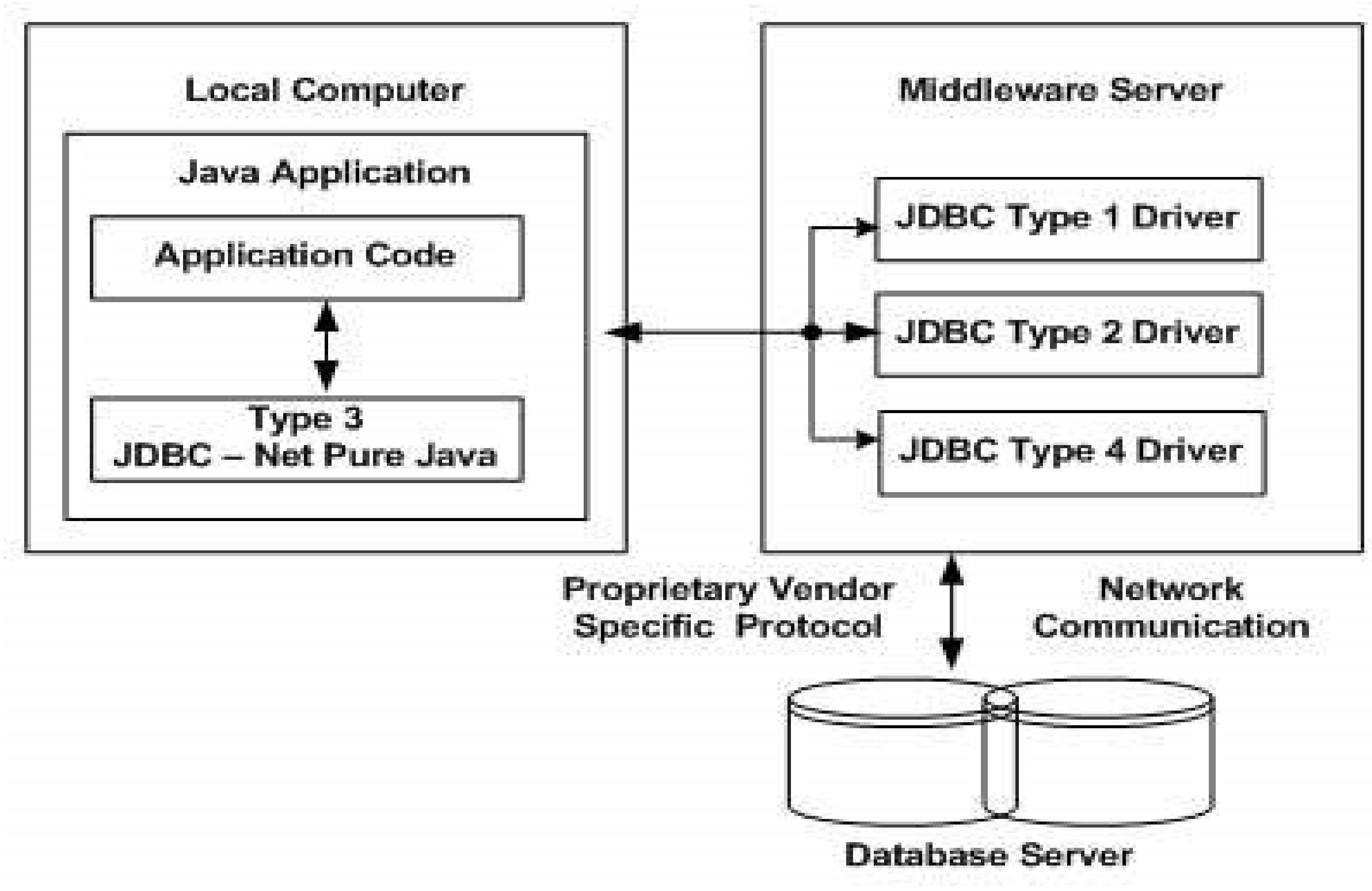
TYPE 2: JDBC-NATIVE API



TYPE 3: JDBC-NET PURE JAVA

- In a Type 3 driver, a three-tier approach is used to access databases.
- The JDBC clients use standard network sockets to communicate with a middleware application server.
- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases

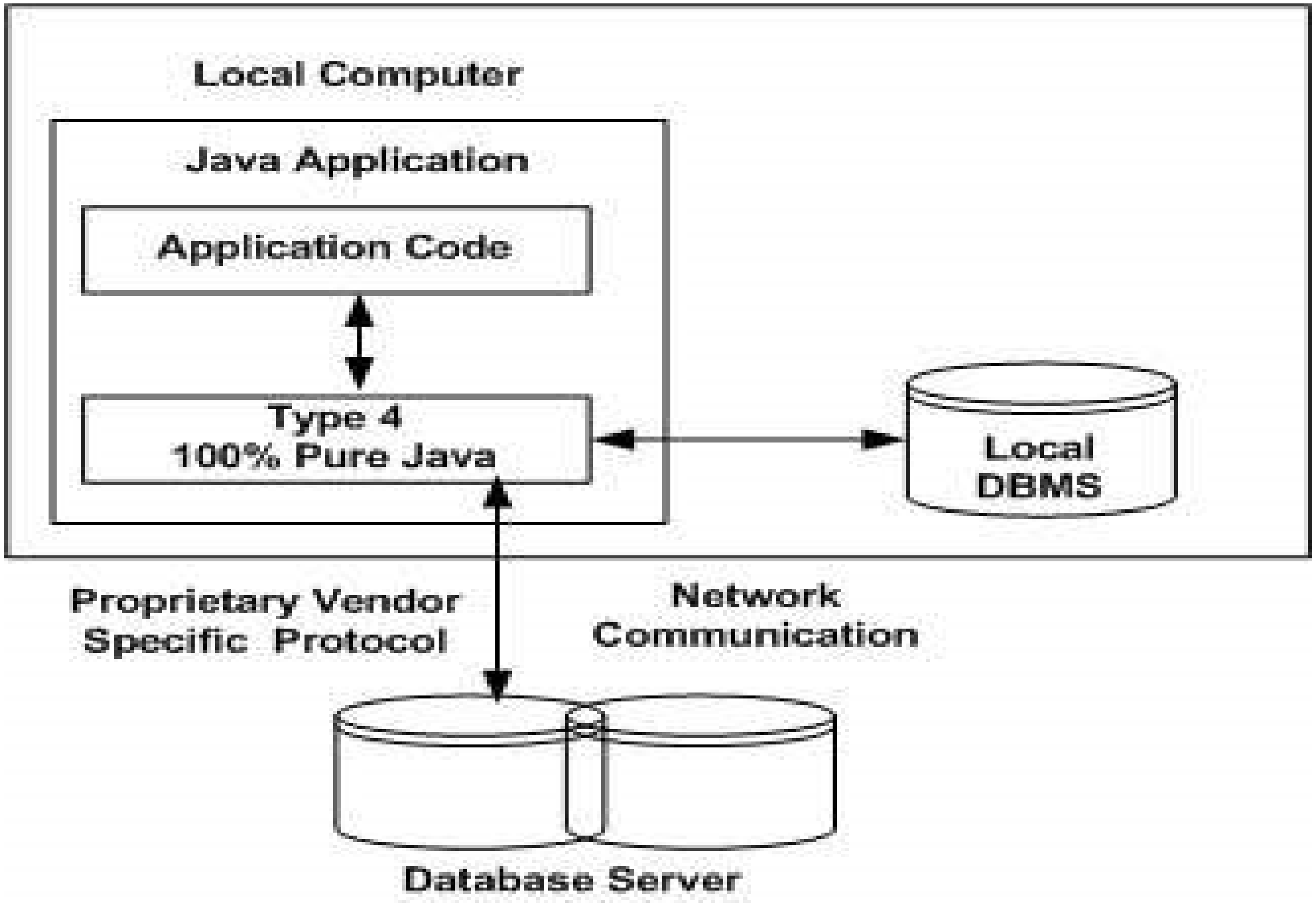
TYPE 3: JDBC-NET PURE JAVA



TYPE 4: DIRECT TO DATABASE PURE JAVA DRIVER

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
- This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server.Further, these drivers can be downloaded dynamically.
- MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers

TYPE 4: DIRECT TO DATABASE PURE JAVA DRIVER



WHICH DRIVER SHOULD BE USED?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

ADVANTAGES &DISADVANTAGES OF TYPE 1: JDBC-ODBC BRIDGE DRIVER

ADVANTAGES

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

TYPE 2: JDBC-NATIVE API

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

TYPE 3: JDBC-NET PURE JAVA

- Advantage:
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Disadvantages:
- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
-

TYPE 4: DIRECT TO DATABASE PURE JAVA DRIVER

- Advantage:
- Better performance than all other drivers.
- No software is required at client side or server side.

- Disadvantage:
- Drivers depend on the Database.