JDBC

http://www.java2all.com

Chapter 1

JDBC Introduction

http://www.java2all.com

Introduction to JDBC

http://www.java2all.com

JDBC - Java Database Connectivity.

JDBC provides API or Protocol to interact with different databases.

With the help of **JDBC** driver we can connect with different types of databases.

Driver is must needed for connection establishment with any database.

A driver works as an interface between the client and a database server.

JDBC have so many classes and interfaces that allow a java application to send request made by user to any specific **DBMS**(Data Base Management System).

JDBC supports a wide level of portability. **JDBC** provides interfaces that are compatible with java application

components and specification of JDBC:

http://www.java2all.com

Components of JDBC:

JDBC has four main components as under and with the help of these components java application can connect with database.

The **JDBC API** - it provides various methods and interfaces for easy communication with database.

The **JDBC** DriverManager - it loads database specific drivers in an application to establish connection with database.

The **JDBC** test suite - it will be used to test an operation being performed by **JDBC** drivers.

The **JDBC-ODBC** bridge - it connects database drivers to the database.

JDBC Specification:

Different version of JDBC has different specification as under.

JDBC 1.0 - it provides basic functionality of JDBC.

JDBC 2.0 - it provides JDBC API(JDBC 2.0 Core API and JDBC 2.0 Optional Package API).

JDBC 3.0 - it provides classes and interfaces in two packages(java.sql and javax.sql).

JDBC 4.0 - it provides so many extra features like Auto loading of the driver interface.

Connection management.

ROWID data type support.

Enhanced support for large object like **BLOB**(Binary Large Object) and **CLOB**(**Character** Large Object).

JDBC Architecture:

As we all know now that driver is required to communicate with database.

JDBC API provides classes and interfaces to handle request made by user and response made by database.

Some of the important JDBC API are as under.

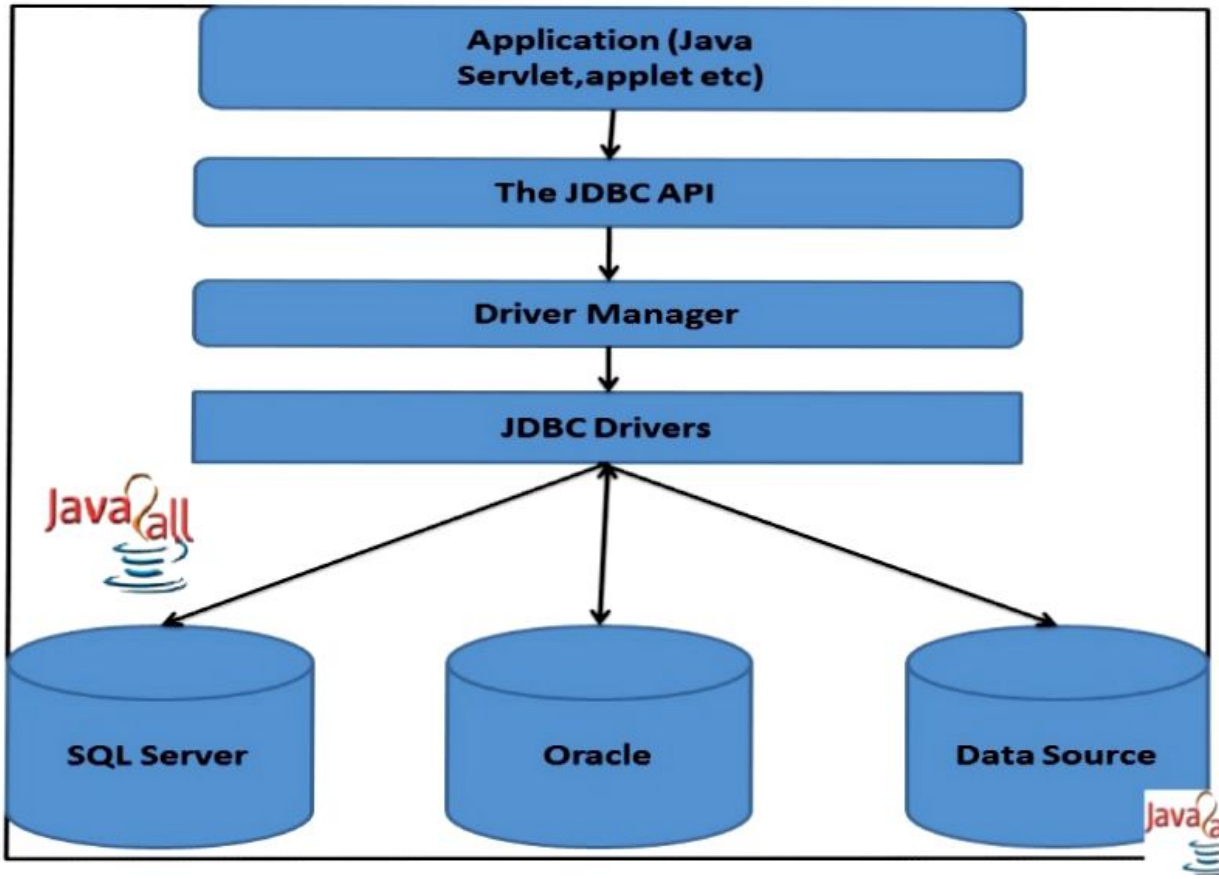
DriverManager
Connection
PreparedStatement
ResultSet
ResultSetMetaData

Driver
Statement
CallableStatement
DatabaseMetaData

http://www.java2all.com

Here The DriverManager plays an important role in **JDBC** architecture.

It uses some database specific drivers to communicate our **J2EE** application to database.



As per the diagram first of all we have to program our application with **JDBC API**.

With the help of **DriverManager** class than we connect to a specific database with the help of spcific database driver.

Java drivers require some library to communicate with the database.

We have four different types of java drivers. We will learn all that four drivers with architecture in next chapter.

Some drivers are pure java drivers and some are partial.

So with this kind of **JDBC** architecture we can communicate with specific database.

We will learn programmatically all this thing in further chapter.

JDBC Driver Types:

http://www.java2all.com

There are four categories of drivers by which developer can apply a connection between Client (The JAVA application or an applet) to a DBMS.

(1) Type 1 Driver : JDBC-ODBC Bridge.

(2) Type 2 Driver : Native-API Driver (Partly Java driver).

(3) Type 3 Driver : Network-Protocol Driver (Pure Java driver for database Middleware).

(4) Type 4 Driver : Native-Protocol Driver (Pure Java driver directly connected to database).

(1) Type 1 Driver: JDBC-ODBC Bridge :-

The JDBC type 1 driver which is also known as a JDBC-ODBC Bridge is a convert JDBC methods into ODBC function calls.

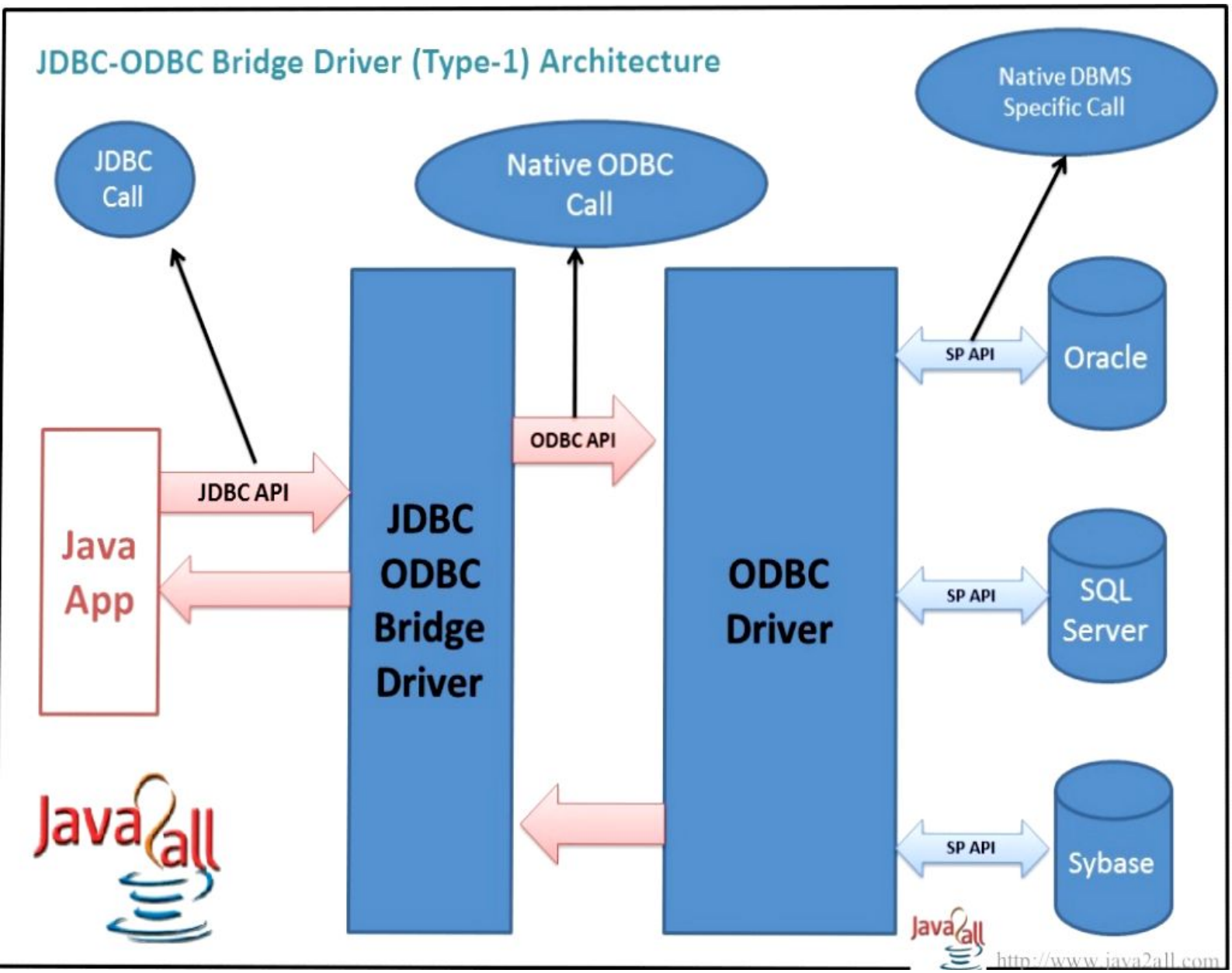
Sun provides a JDBC-ODBC Bridge driver by "**sun.jdbc.odbc.JdbcOdbcDriver**".

The driver is a platform dependent because it uses ODBC which is depends on native libraries of the operating system and also the driver needs other installation for example, ODBC must be installed on the computer and the database must support ODBC driver.

Type 1 is the simplest compare to all other driver but it's a platform specific i.e. only on Microsoft platform.

The JDBC-ODBC Bridge is use only when there is no PURE-JAVA driver available for a particular database.

Architecture Diagram:



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 1 Driver → ODBC Driver → Database library APIs → Database

Advantage:

(1) Connect to almost any database on any system, for which ODBC driver is installed.

(2) It's an easy for installation as well as easy(simplest) to use as compare the all other driver.

Disadvantage:

(1) The ODBC Driver needs to be installed on the client machine.

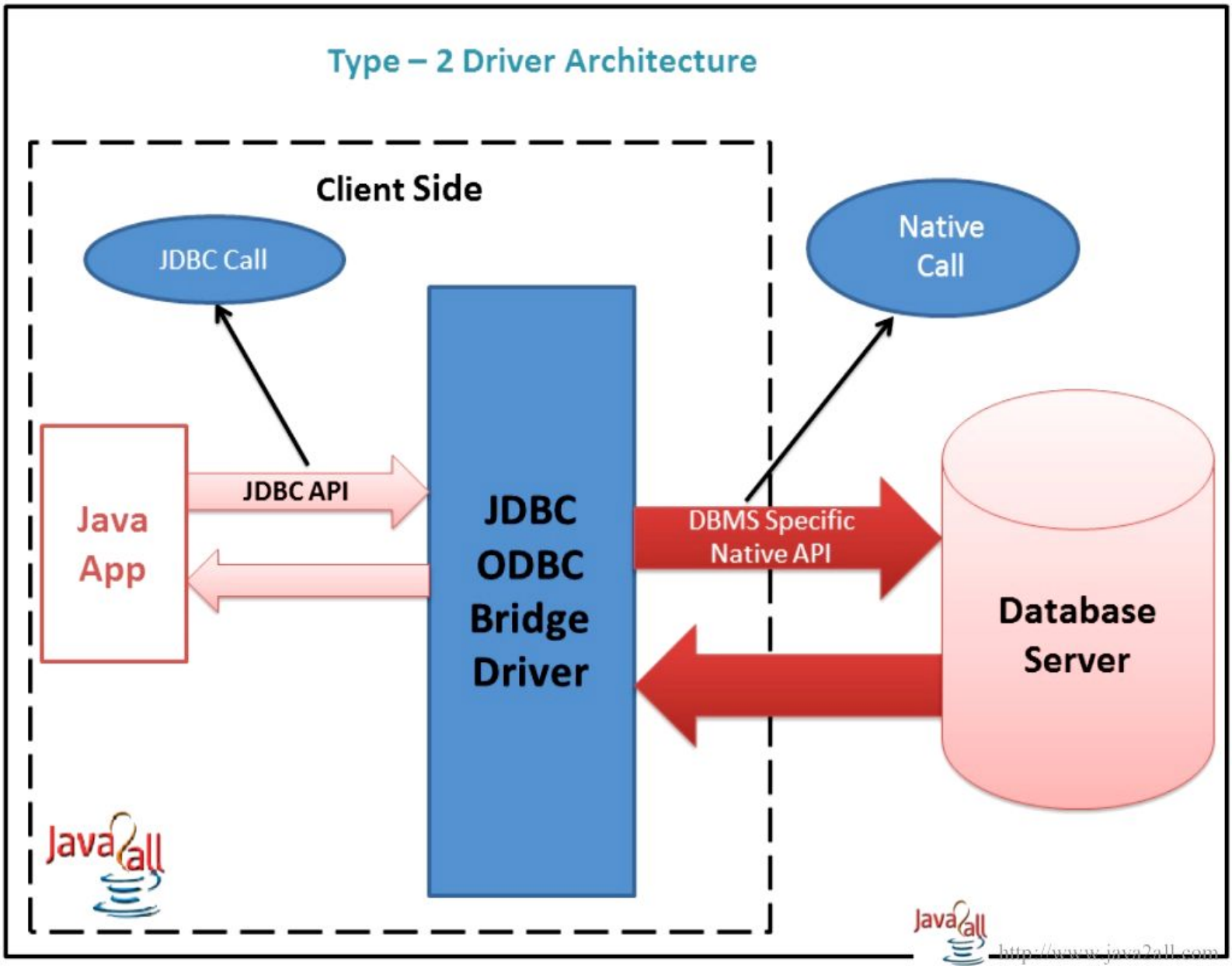
(2) It's a not a purely platform independent because its use ODBC which is depends on native libraries of the operating system on client machine.

(3) Not suitable for applets because the ODBC driver needs to be installed on the client machine.

(2) Type 2 Driver: Native-API Driver (Partly Java driver) :-

The JDBC type 2 driver is uses the libraries of the database which is available at client side and this driver converts the JDBC method calls into native calls of the database so this driver is also known as a Native-API driver.

Architecture Diagram :



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 2 Driver → Vendor Client Database library APIs → Database

Advantage:

(1) There is no implantation of **JDBC-ODBC** Bridge so it's faster than a type 1 driver; hence the performance is better as compare the type 1 driver (**JDBC-ODBC Bridge**).

Disadvantage:

(1) On the client machine require the extra installation because this driver uses the vendor client libraries.

(2) The Client side software needed so cannot use such type of driver in the web-based application.

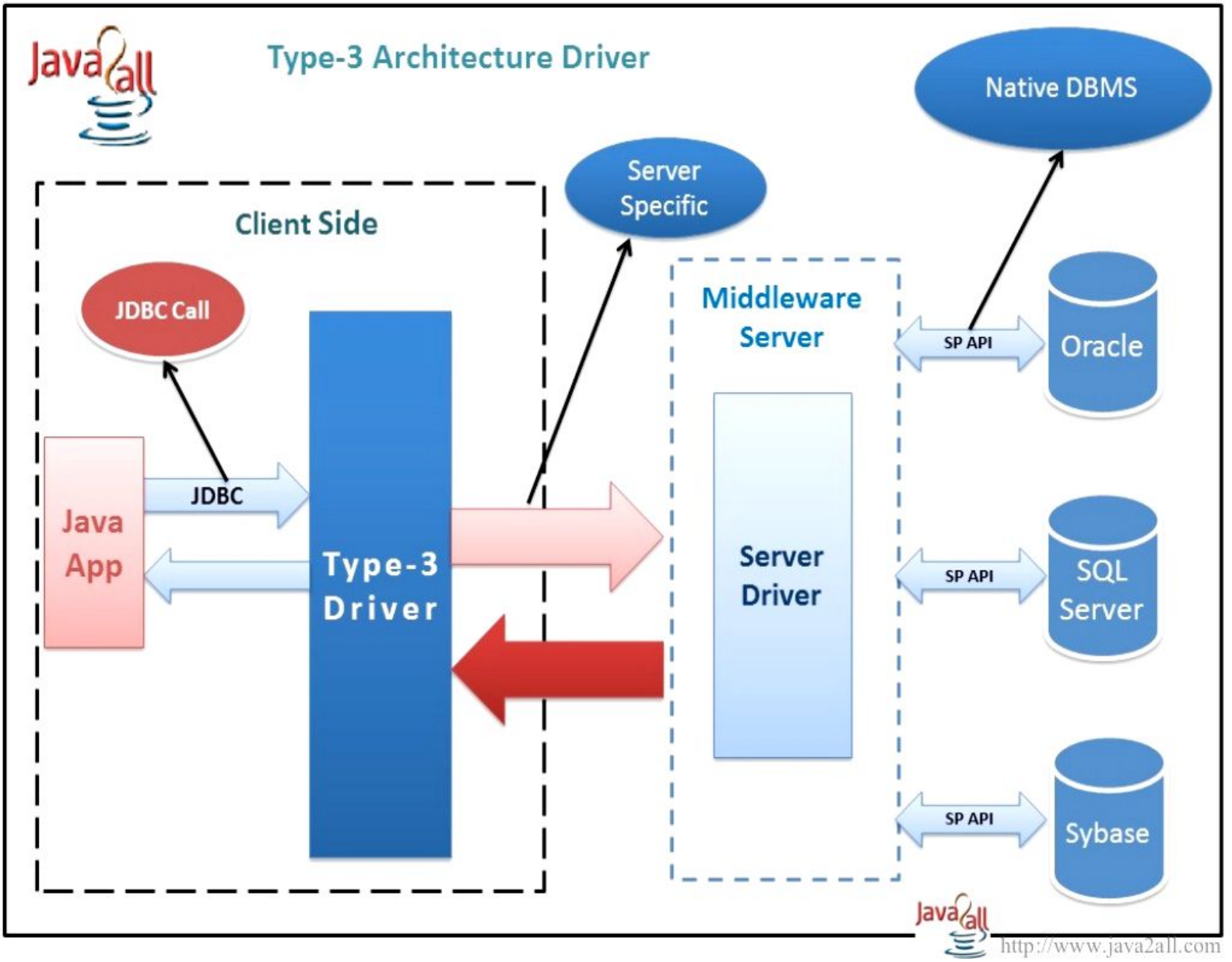
(3) Not all databases have the client side library.

(4) This driver supports all **JAVA** applications except applets.

(3) Type 3 Driver: Network-Protocol Driver (Pure Java driver for database Middleware) :-

The JDBC type 3 driver uses the middle tier(application server) between the calling program and the database and this middle tier converts JDBC method calls into the vendor specific database protocol and the same driver can be used for multiple databases also so it's also known as a Network-Protocol driver as well as a **JAVA** driver for database middleware.

Architecture Diagram:



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 3 Driver → Middleware (Server)→ any Database

Advantage:

(1) There is no need for the vendor database library on the client machine because the middleware is database independent and it communicates with client.

(2) Type 3 driver can be used in any web application as well as on internet also because there is no any software require at client side.

(3) A single driver can handle any database at client side so there is no need a separate driver for each database.

(4) The middleware server can also provide the typical services such as connections, auditing, load balancing, logging etc.

Disadvantage:

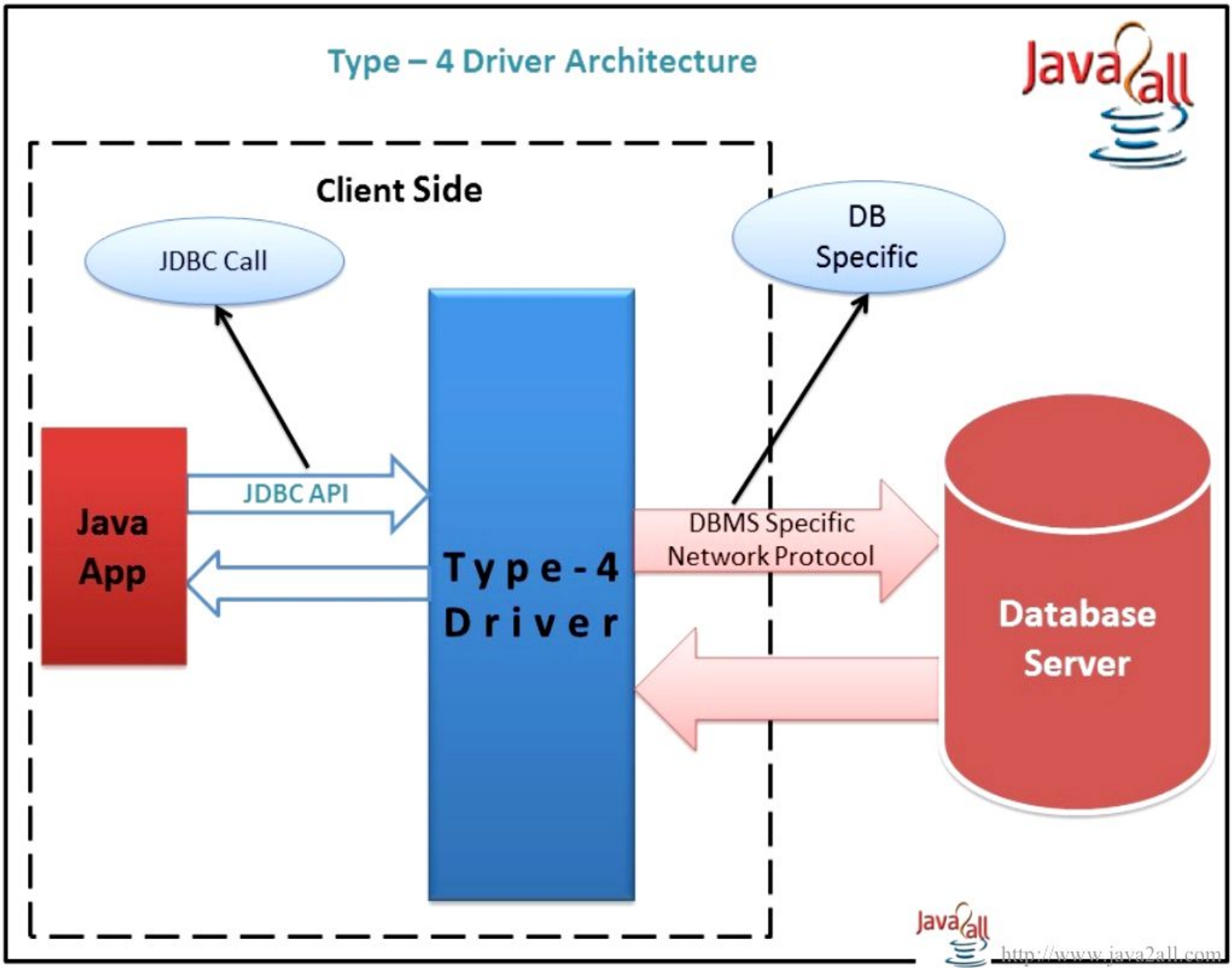
(1) An Extra layer added, may be time consuming.

(2) At the middleware develop the database specific coding, may be increase complexity.

(4) Type 4 Driver: Native-Protocol Driver (Pure Java driver directly connected to database) :-

The JDBC type 4 driver converts JDBC method calls directly into the vendor specific database protocol and in between do not need to be converted any other formatted system so this is the fastest way to communicate quires to DBMS and it is completely written in JAVA because of that this is also known as the "direct to database Pure JAVA driver".

Architecture Diagram:



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 4 Driver (Pure JAVA Driver) → Database Server

Advantage:

(1) It's a 100% pure JAVA Driver so it's a platform independence.

(2) No translation or middleware layers are used so consider as a faster than other drivers.

(3) The all process of the application-to-database connection can manage by JVM so the debugging is also managed easily.

Disadvantage:

(1) There is a separate driver needed for each database at the client side.

(2) Drivers are Database dependent, as different database vendors use different network protocols.