Database Development and Design (CPT201)

Lecture 9: Database Connectivity

Dr. Wei Wang

Department of Computing

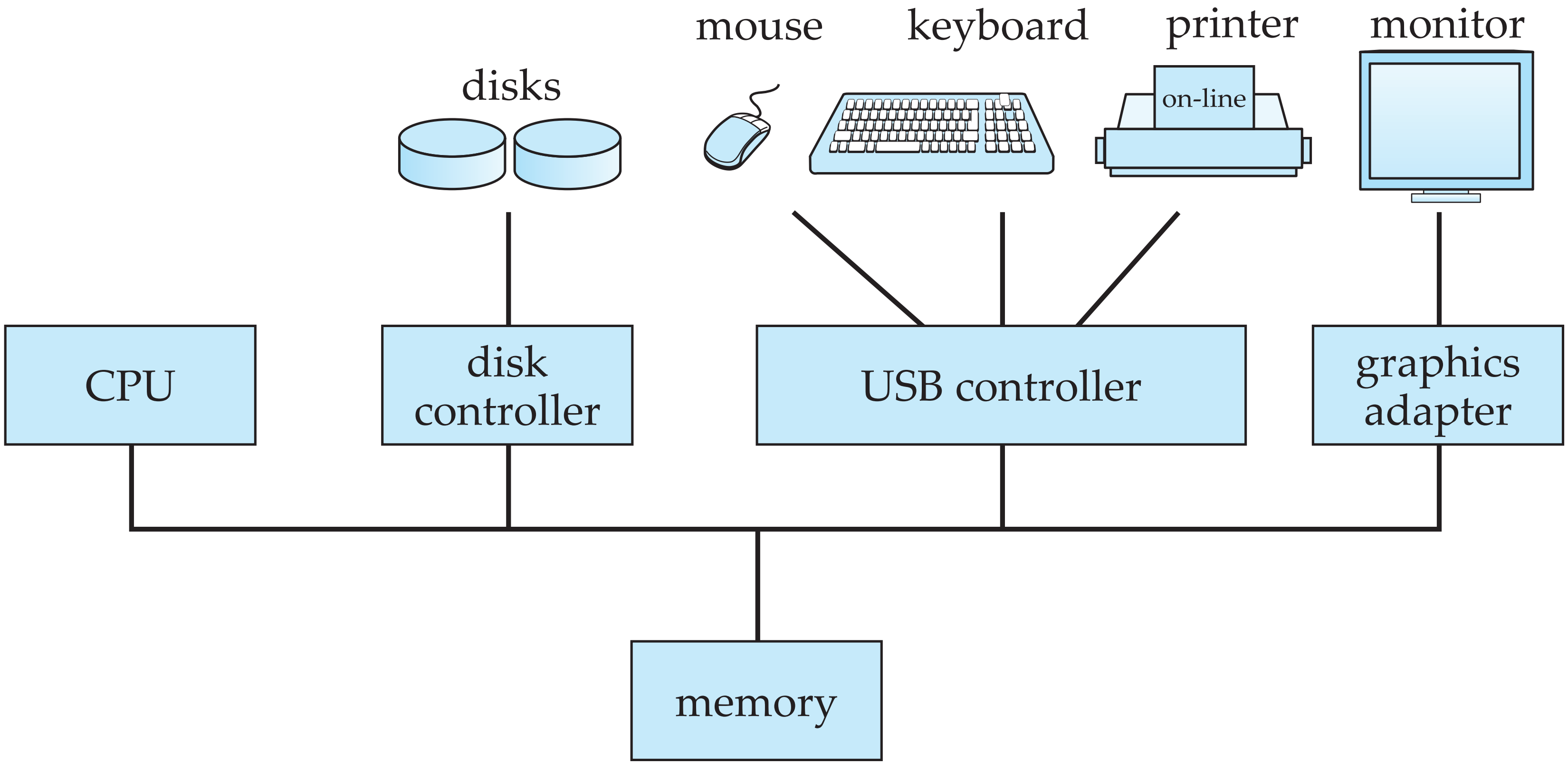
Learning Outcomes

- Centralised and Client-Server Systems
- Server System Architectures
- Database Connectivity
  - ODBC
  - JDBC
  - Java Programming wit JDBC

Centralised Systems

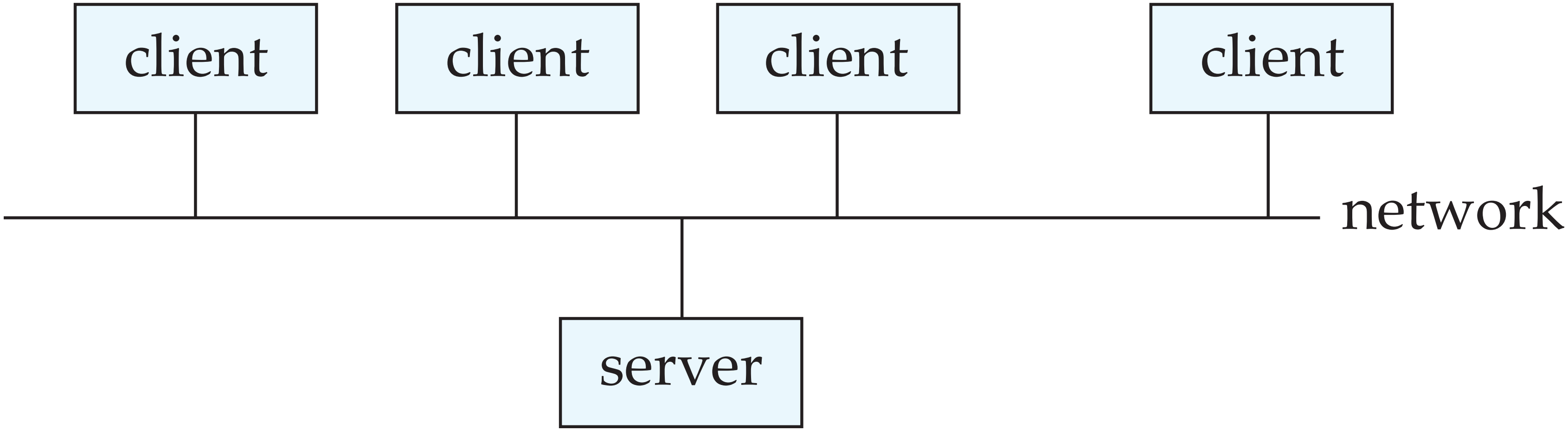
- Run on a single computer system and do not interact with other computer systems.
  - General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
  - Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
  - Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called *server* systems.

A Centralised Computer System



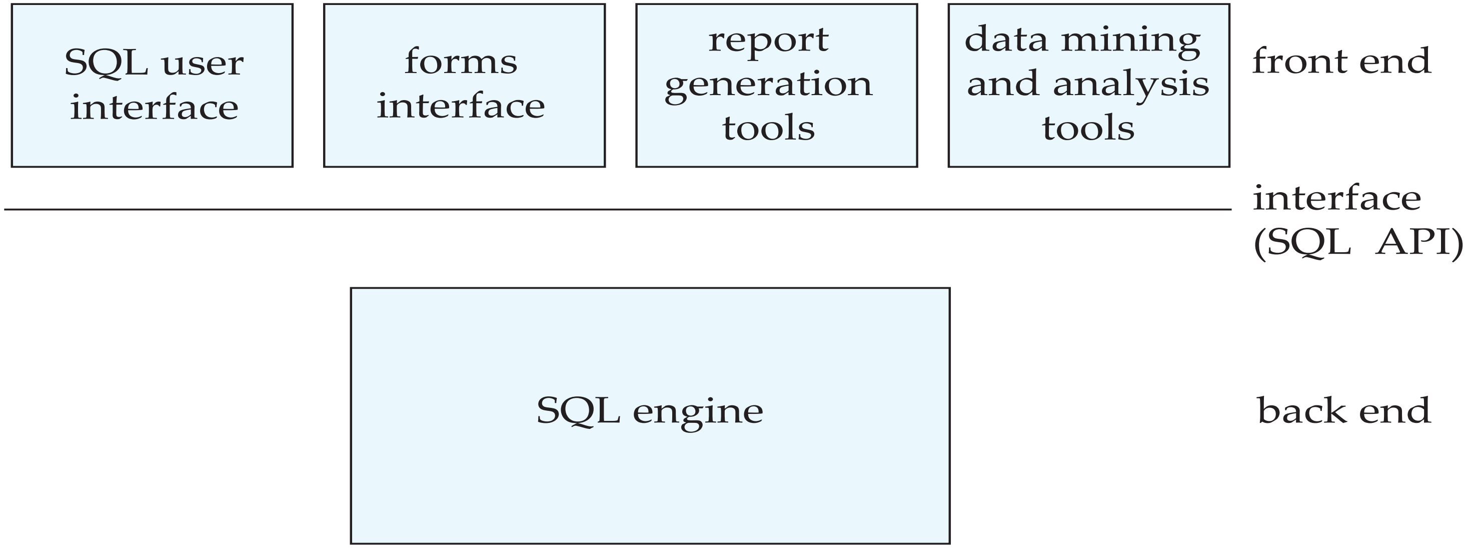
Client-Server Systems

- Server systems satisfy requests generated at client systems.



Client-Server Systems cont'd

- Database functionality can be divided into:
  - **Back-end**: manages access structures, query evaluation and optimisation, concurrency control and recovery, etc.
  - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities, etc.
- The interface between the front-end and the back-end is through SQL or through an application program interface.



Client-Server Systems cont’d

- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance

Server System Architecture

- Server systems can be broadly categorised into two kinds:
  - **transaction servers** widely used in relational database systems
  - **data servers** used in object-oriented database systems

Transaction Servers

- Also called **query server** systems or SQL *server* systems
  - Clients send requests to the server,
  - Transactions are executed at the server,
  - Results are shipped back to the client.
- Requests are specified in SQL, and communicated to the server through a *remote procedure call* (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- *Open Database Connectivity* (ODBC) is a C language application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.
- JDBC standard is similar to ODBC, but for Java.

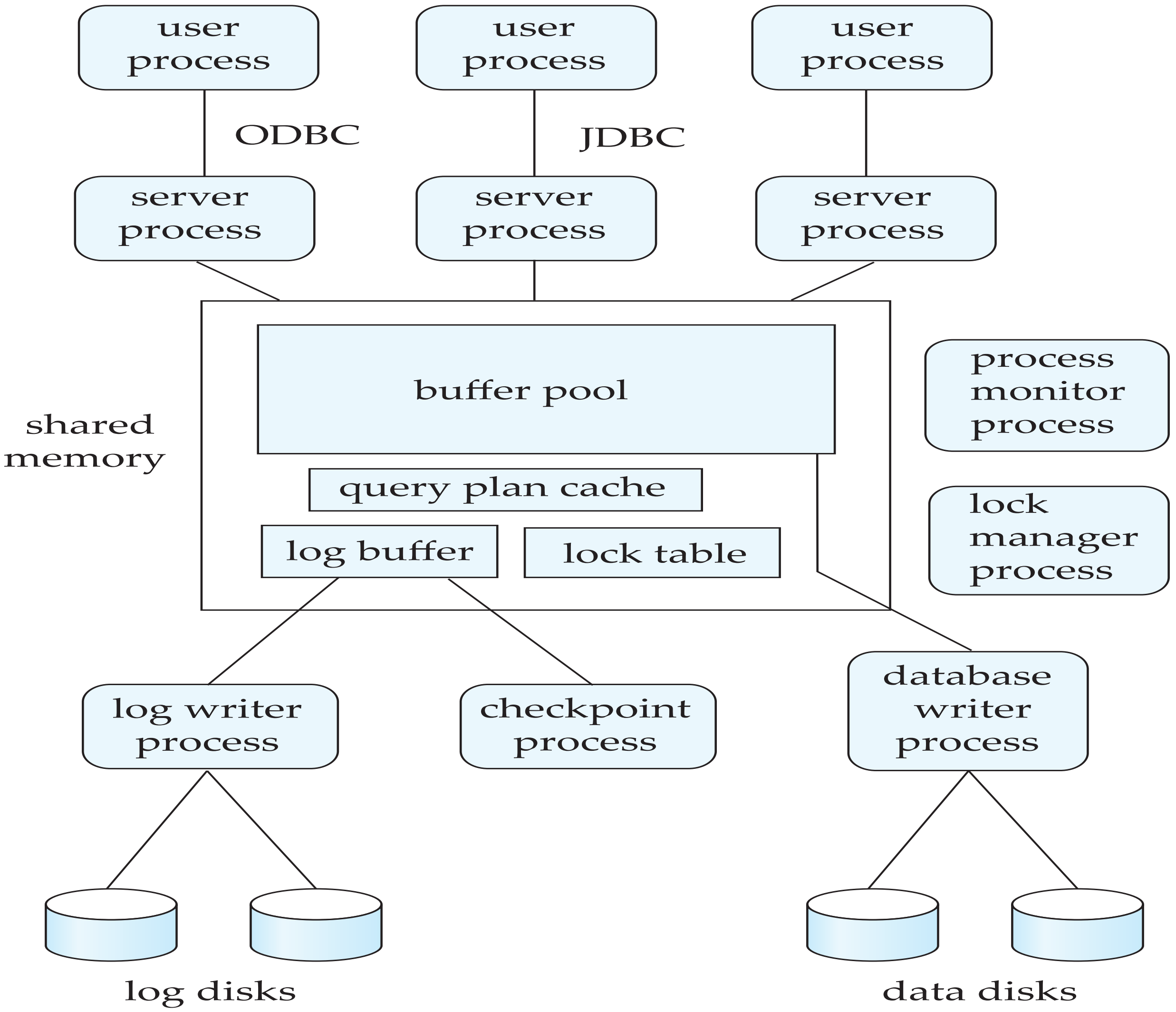
Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory.
- Server processes
  - receive user queries (transactions), execute them and send results back
  - may be **multithreaded** to support multiple user queries concurrently
  - may be multiple multithreaded server processes
- Lock manager process
- Database writer process
  - Output modified buffer blocks to disks continually

Transaction Server Processes cont'd

- Log writer process
  - Server processes simply add log records to log record buffer
  - Log writer process outputs log records to stable storage.
- Checkpoint process
  - Performs periodic checkpoints
- Process monitor process
  - Monitors other processes, and takes recovery actions if any of the other processes fail
    - e.g., aborting any transactions being executed by a server process and restarting it

Transaction System Processes (Cont.)



Transaction System Processes cont’d

- Shared memory contains shared data
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)
- All database processes can access shared memory
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement **mutual exclusion** using either
  - Operating system semaphores
  - Atomic instructions such as test-and-set
- To avoid overhead of inter-process communication for lock request/grant, each database process operates directly on the lock table
  - instead of sending requests to lock manager process
- Lock manager process still used for deadlock detection

Data Servers

- Used in high-speed LANs, in cases where
  - The clients are comparable in processing power to the server
  - The tasks to be executed are computationally intensive.
- Data is shipped to clients where processing is performed, and then shipped results back to the server.
- This architecture requires full back-end functionality at the clients.
- Used in many object-oriented database systems
- Issues:
  - Page-Shipping versus Item-Shipping
  - Locking
  - Data Caching
  - Lock Caching

Data Servers cont'd

- Page-shipping versus item-shipping
  - Smaller unit of shipping VS more messages
  - Worth prefetching related items along with requested item
  - Page shipping can be thought of as a form of prefetching
- Locking
  - Overhead of requesting and getting locks from server is high due to message delays.
  - Can grant locks on requested and prefetched items; with page shipping, transaction is granted lock on whole page.
  - Locks on a prefetched item can be called back by the server, and returned by client transaction if the prefetched item has not been used.
  - Locks on the page can be deescalated to locks on items in the page when there are lock conflicts. Locks on unused items can then be returned to server.

Data Servers cont'd

- **Data Caching**
  - Data can be cached at client even in between transactions
  - But check that data is up-to-date before it is used (**cache coherency**)
  - Check can be done when requesting lock on data item
- **Lock Caching**
  - Locks can be retained by client system even in between transactions
  - Transactions can acquire cached locks locally, without contacting server
  - Server **calls back** locks from clients when it receives conflicting lock request.
  - Client returns lock once no local transaction is using it.

Accessing Database from Applications

- SQL commands can be called from within a host language (e.g., C++ or Java) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include statement to *connect* to right database.
- Two main integration approaches:
  - Embed SQL in the host language (e.g., Pro*C, Embedded SQL, SQLJ)
  - Create special API (Call Level Interface) to call SQL commands (eg: JDBC, ODBC, PHP)

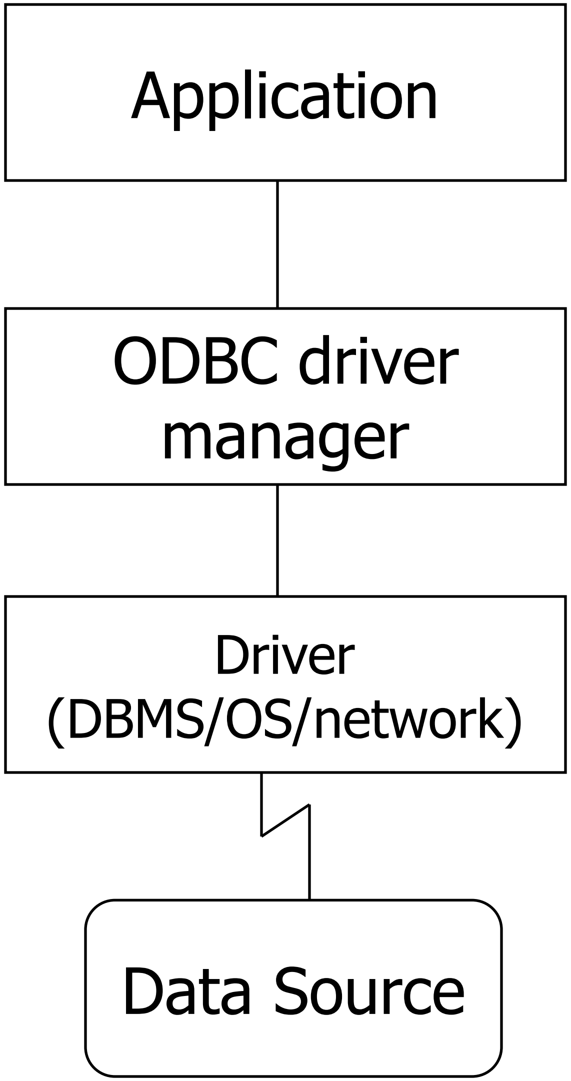
What is ODBC?

- ODBC is short for Open Database Connectivity
  - A standard open application programming interface (API) for accessing a database.
  - SQL Access Group, chiefly Microsoft, in 1992
  - By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel, and Text.
  - It allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases.
  - ODBC handles the SQL request and converts it into a request the individual database system understands.

More on ODBC

- You need:
  - the ODBC software, and
  - a separate module or *driver* for each database to be accessed (library that is dynamically connected to the application).
- Driver masks the heterogeneity of DBMS operating system and network protocol.
  - e.g. (Sybase, Windows/NT, Novell driver)

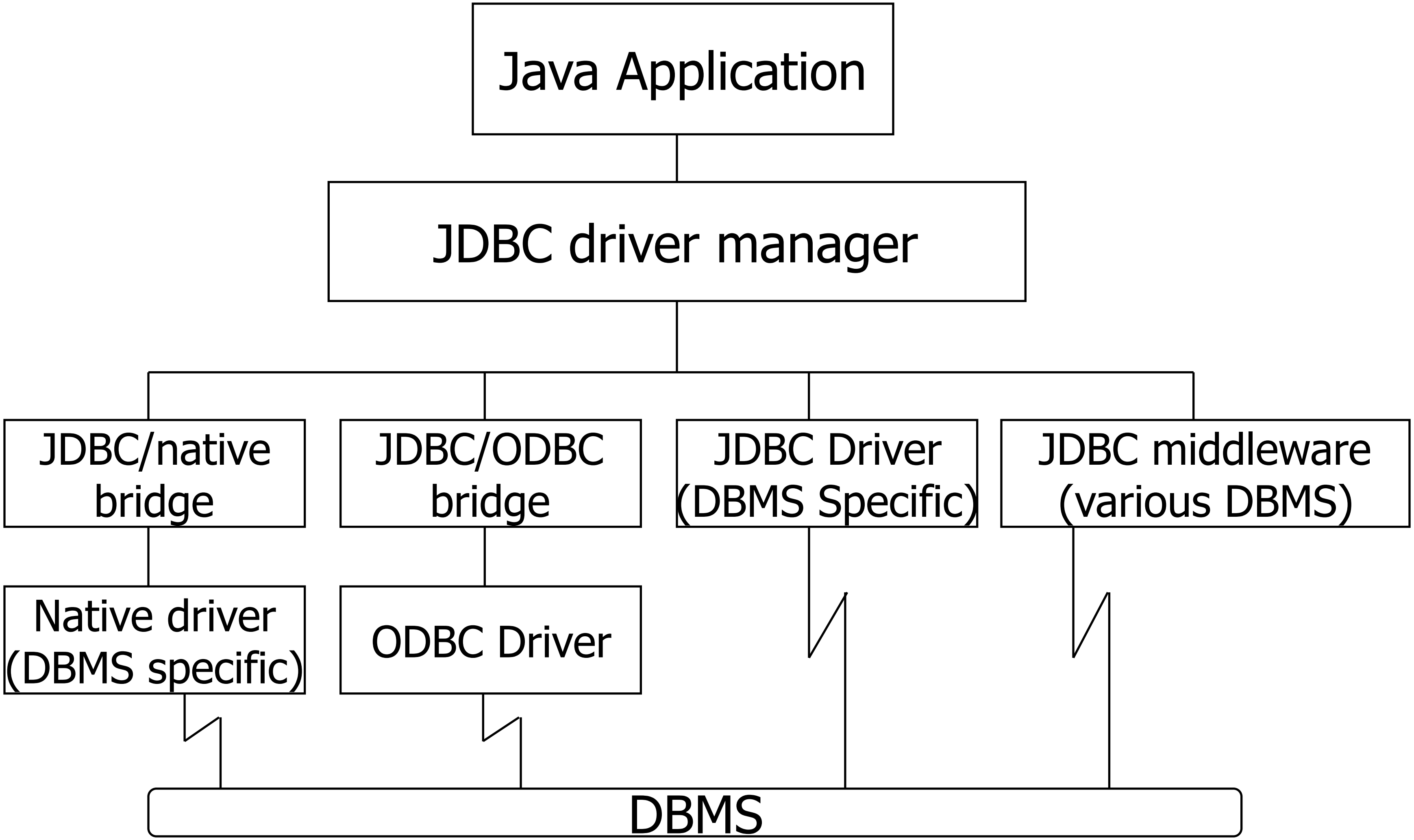
ODBC Architecture



What is JDBC?

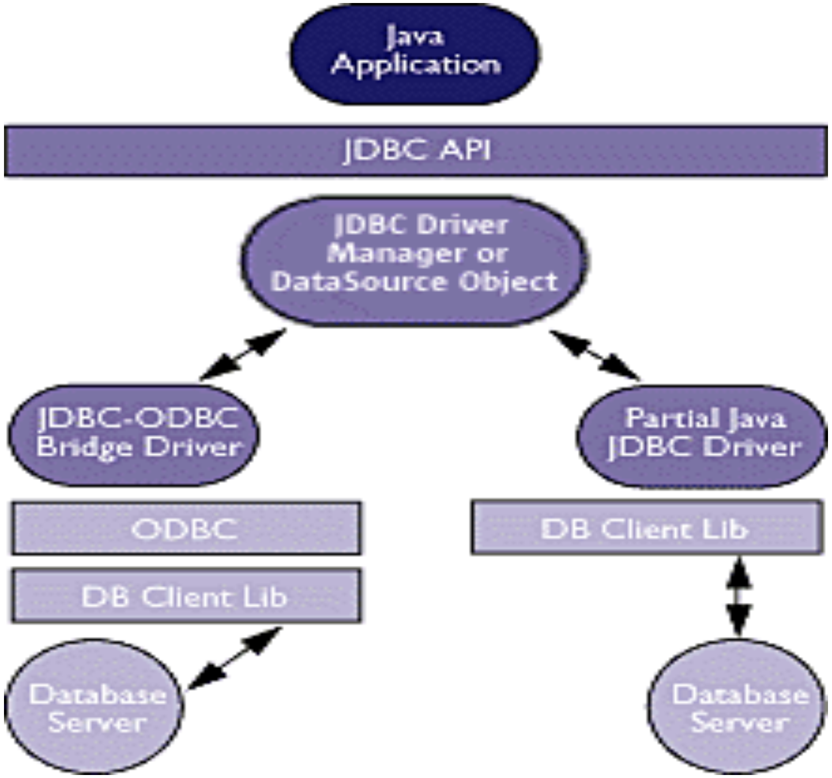
- JDBC is short for Java Database Connectivity
  - is a Java API for connecting programs written in Java to the data in relational databases.
  - consists of a set of classes and interfaces written in the Java programming language.
  - provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
  - The standard defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers.
- JDBC:
  - establishes a connection with a database
  - sends SQL statements
  - processes the results

JDBC Architectures


Java Application
JDBC driver manager
JDBC/native bridge
JDBC/ODBC bridge
JDBC Driver (DBMS Specific)
JDBC middleware (various DBMS)
Native driver (DBMS specific)
ODBC Driver
DBMS

JDBC connectivity using ODBC drivers

- With a small "bridge" program, you can use the JDBC interface to access ODBC-accessible databases.



JDBC vs ODBC

- ODBC is used between applications
- JDBC is used by Java programmers to connect to databases
- With a small "bridge" program, you can use the JDBC interface to access ODBC-accessible databases.
- JDBC allows SQL-based database access for EJB (Enterprise JavaBeans) persistence and for direct manipulation from CORBA and other server objects.

JDBC API

- The JDBC API supports both two-tier and three-tier models for database access.
- Two-tier model - a Java applet or application interacts directly with the database.
- Three-tier model - introduces a middle-level server for execution of business logic:
  - the middle tier to maintain control over data access.
  - user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls.

The JDBC Steps

- Importing Packages
- Registering the JDBC Drivers
- Opening a Connection to a Database
- Creating a Statement Object
- Executing a Query and Returning a Result Set Object
- Processing the Result Set
- Closing the Result Set, Statement and Connection Objects

1: Importing Packages

```
// Program name: JDBCExample.java
// Purpose: Basic selection using prepared statement
//
//Import packages

    import java.sql.*;  //JDBC packages
    import java.math.*;
    import java.io.*;
    import oracle.jdbc.driver.*;
```

2: Registering JDBC Drivers

```
class JDBCExample {

public static void main (String args []) throws SQLException
    {

//Load Oracle driver
DriverManager.registerDriver (new
    oracle.jdbc.driver.OracleDriver());
```

3: Opening connection to a Database

```
//Prompt user for username and password

  String user;
  String password;

  user = readEntry("username: ");
  password = readEntry("password: ");

//Connect to the local database
  Connection conn = DriverManager.getConnection
   ("jdbc:oracle:thin:@aardvark:1526:teach",
          user,
          password);
```

4. Creating a Statement Object

```
// Query the hotels table for resort = 'palma nova’

PreparedStatement pstmt = conn.prepareStatement
    ("SELECT hotelname, rating FROM hotels WHERE
    trim(resort) = ?");
pstmt.setString(1, "palma nova");
```

5. Executing a Query, Returning a Result Set Object & 6. Processing the Result Set

```
ResultSet rset = pstmt.executeQuery();

//Print query results
while (rset.next ())
    System.out.println (rset.getString(1)+" "+
                   rset.getString(2));
```

7. Closing the Result Set and Statement Objects & 8. Closing the Connection

```
// close the result set, statement, and the connection
        rset.close();
        pstmt.close();
        conn.close();
    }
```

Mapping Data Types

- There are data types specified to SQL that need to be mapped to Java data types if the user expects Java to be able to handle them.
- Conversion falls into three categories:
  - SQL type to Java direct equivalents
    - SQL INTEGER direct equivalent of Java int data type
  - SQL type can be converted to a Java equivalent.
    - SQL CHAR, VARCHAR, and LONGVARCHAR can all be converted to the Java String data type
  - SQL data type is unique and requires a special Java data class object to be created specifically for their SQL equivalent
    - SQL DATE converted to the Java Date object that is defined in java.Date especially for this purpose

End of Lecture

- Summary
  - Centralised and Client-Server Systems
  - Server System Architectures
  - Database Connectivity: ODBC, JDBC, Java Programming wit JDBC

- Reading
  - Textbook 6th edition, chapters 5.1, 9.1, 9.2, 17.1, and 17.2
  - Textbook 7th edition, chapters 5.1, 9.1, 9.2, 20.2, and 20.3