CS122B: Projects in Databases and Web Applications
Spring 2018

Notes 2: Database Connectivity

Professor Chen Li

Department of Computer Science

UC Irvine

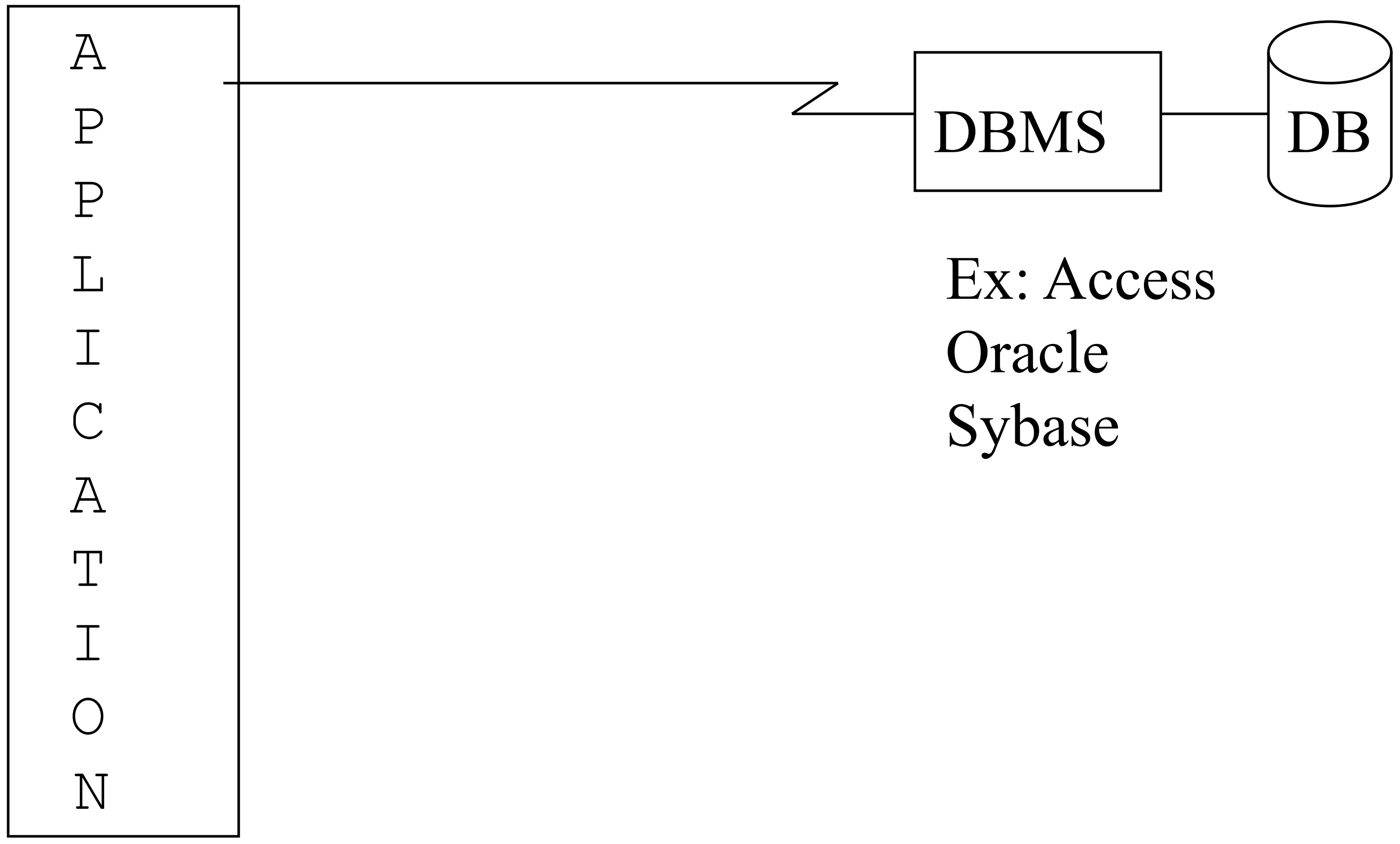
Outline

- Motivation
- Architecture
- 7 steps
- Sample program
- Metadata

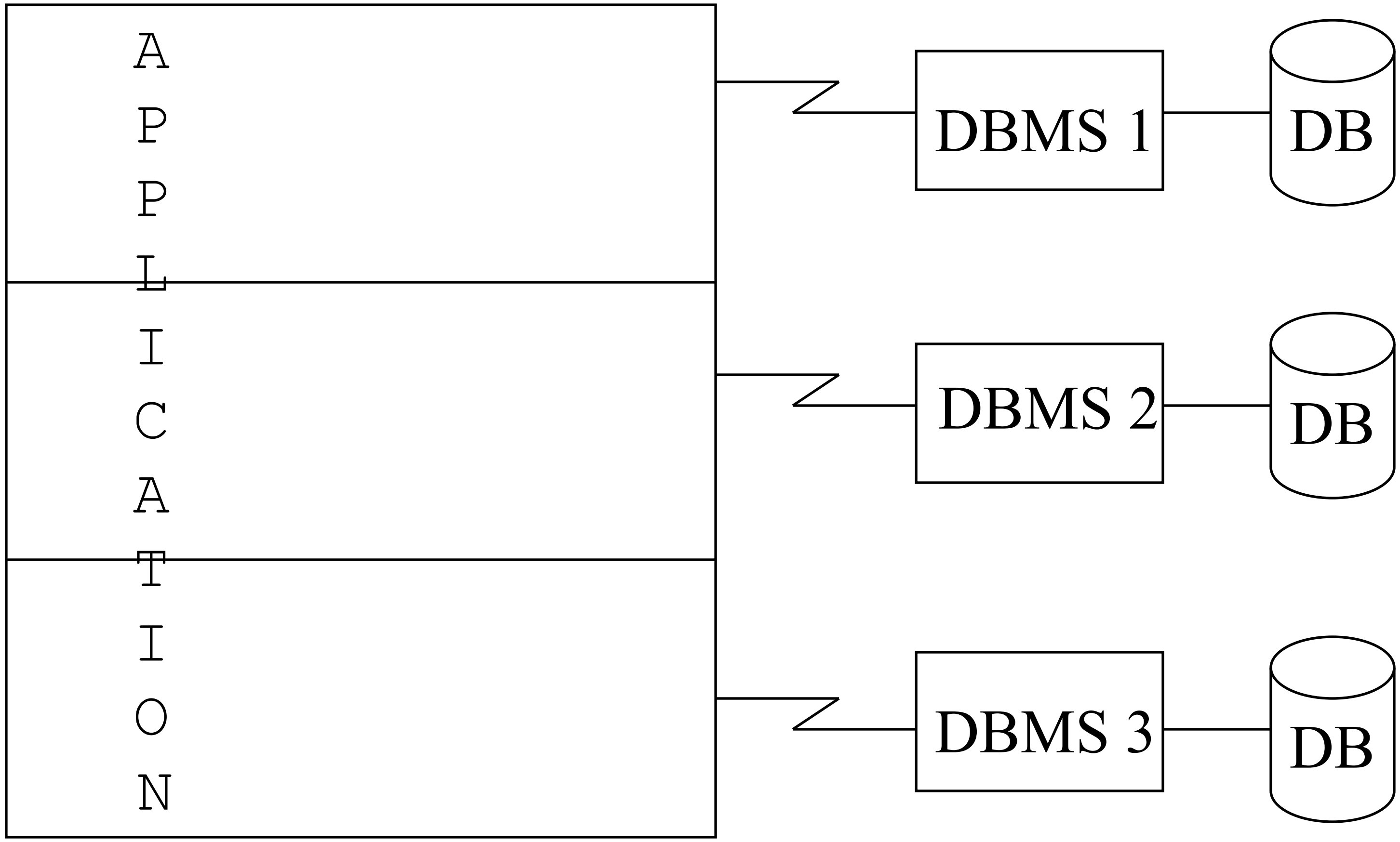
Motivation

- Most popular form of database system is the relational database system.
  - Examples: MS Access, Sybase, Oracle, MS Sequel Server, Postgres.
- Structured Query Language (SQL) is used among relational databases to construct queries.
- Applications need to query databases

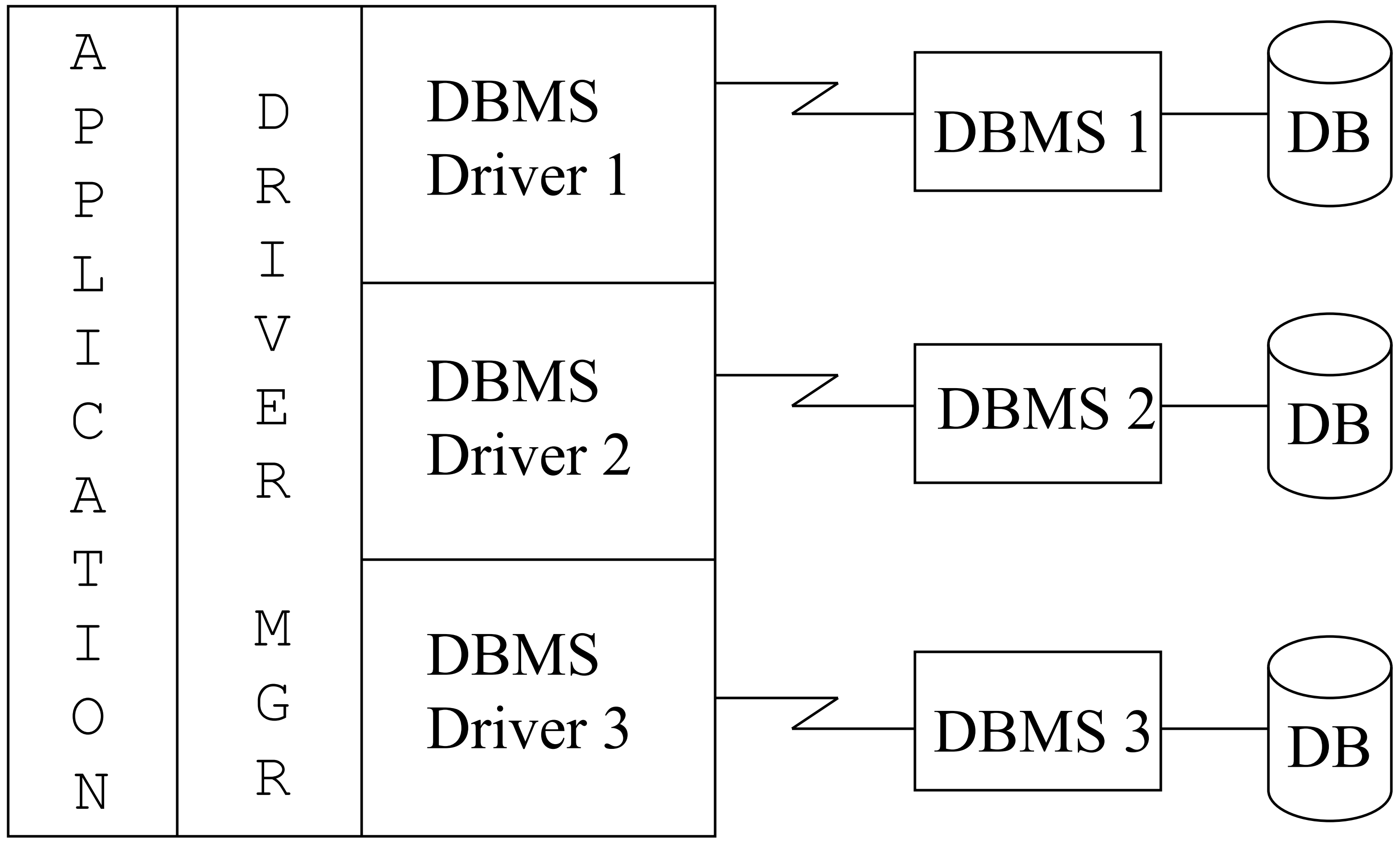
Simple Database Application



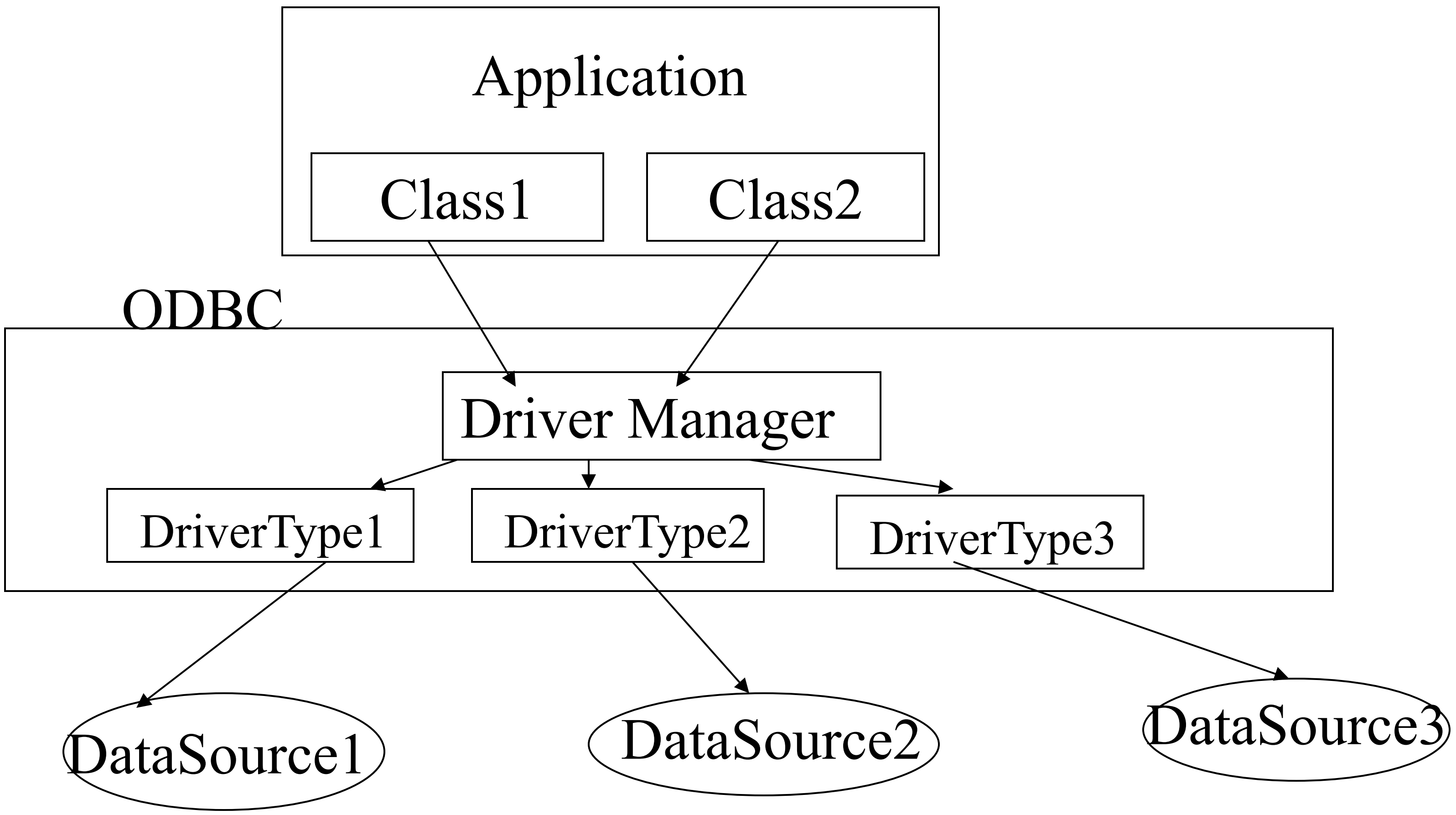
Multi-Databases



Standard Access to DB



ODBC Architecture



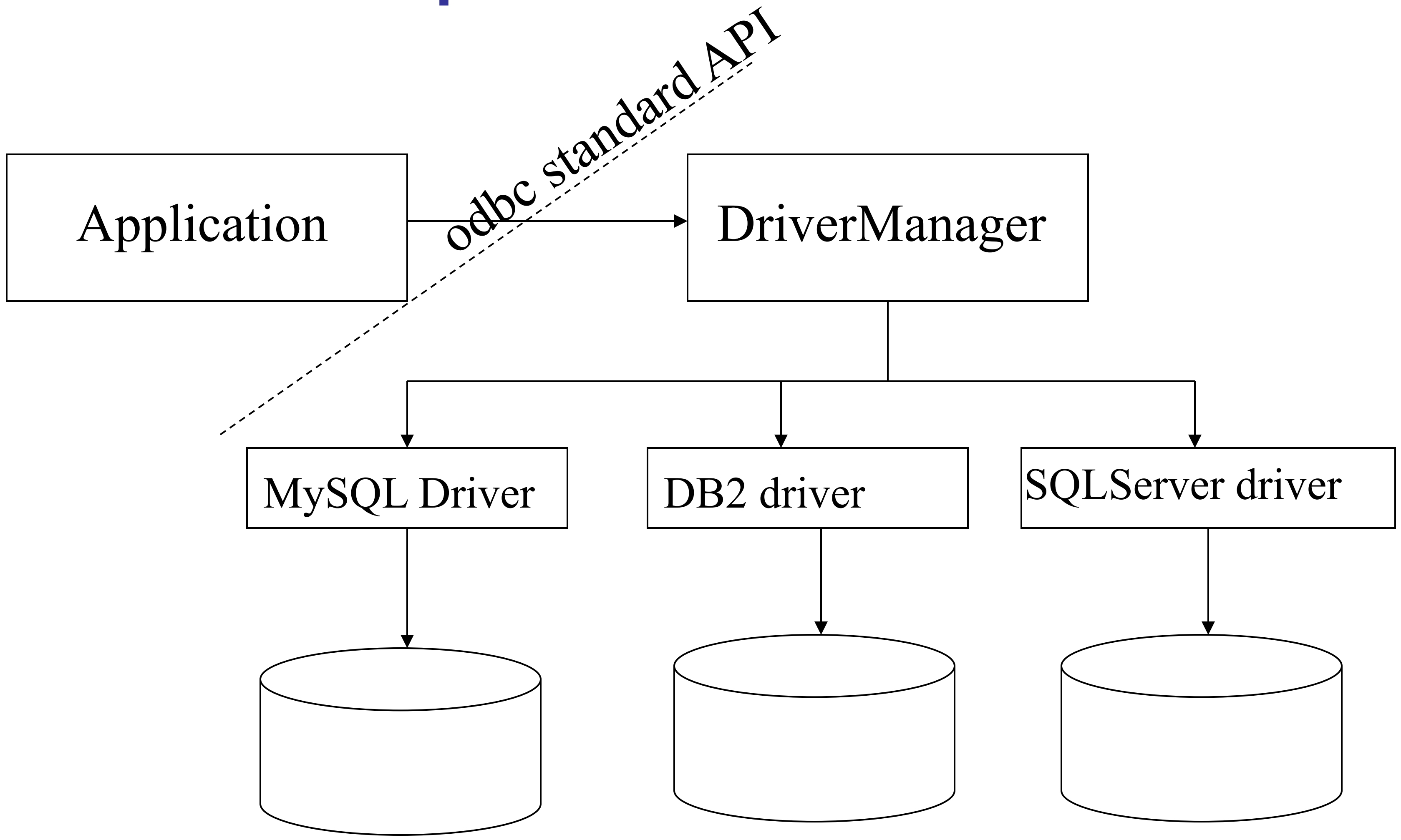
Open Database Connectivity (ODBC) Standard

- ODBC standard is an interface by which application programs can access and process SQL databases in a DBMS-independent manner. It contains:
  - A **Data Source** that is the database, its associated DBMS, operating system and network platform
  - A **DBMS Driver** that is supplied by the DBMS vendor (e.g., Oracle, IBM, MS) or independent software companies
  - A **Driver Manager** that is supplied by the vendor of the O/S platform (e.g., Windows/UNIX/Mac) where the application is running
  - E.g Control Panel->Administrative Tools->Data Sources (ODBC) in Windows. You can use ODBC to access MS Access or even Excel documents using the corresponding drivers.

ODBC Interface

- System-independent interface to database environment
  - requires an ODBC driver to be provided for each database system from which you want to manipulate data.
- The ODBC driver manager bridges the differences between
  - The ODBC DBMS-independent interface and the DBMS-dependent interface of the DBMS driver

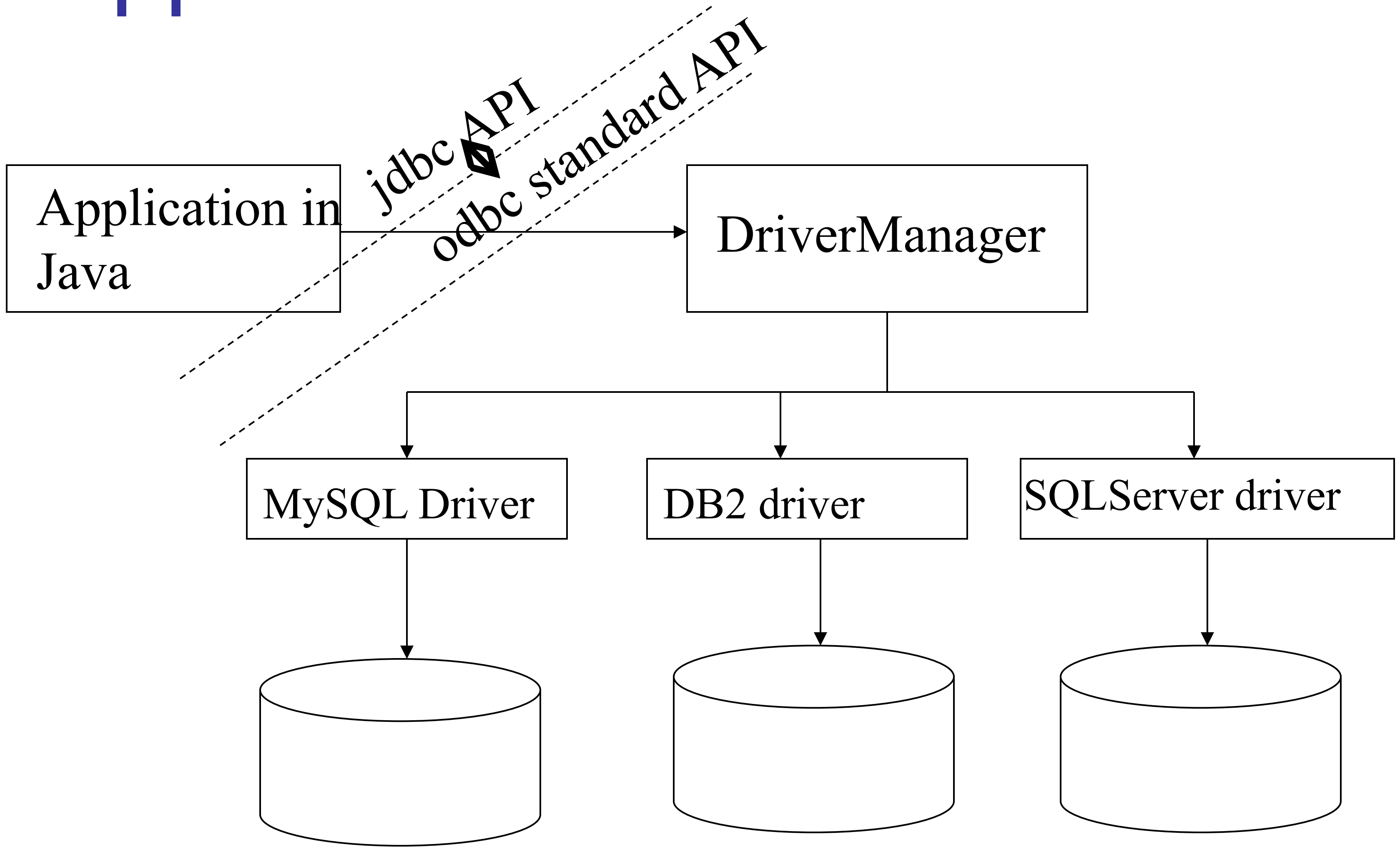
An Example


Application
odbc standard API
DriverManager
MySQL Driver
DB2 driver
SQLServer driver

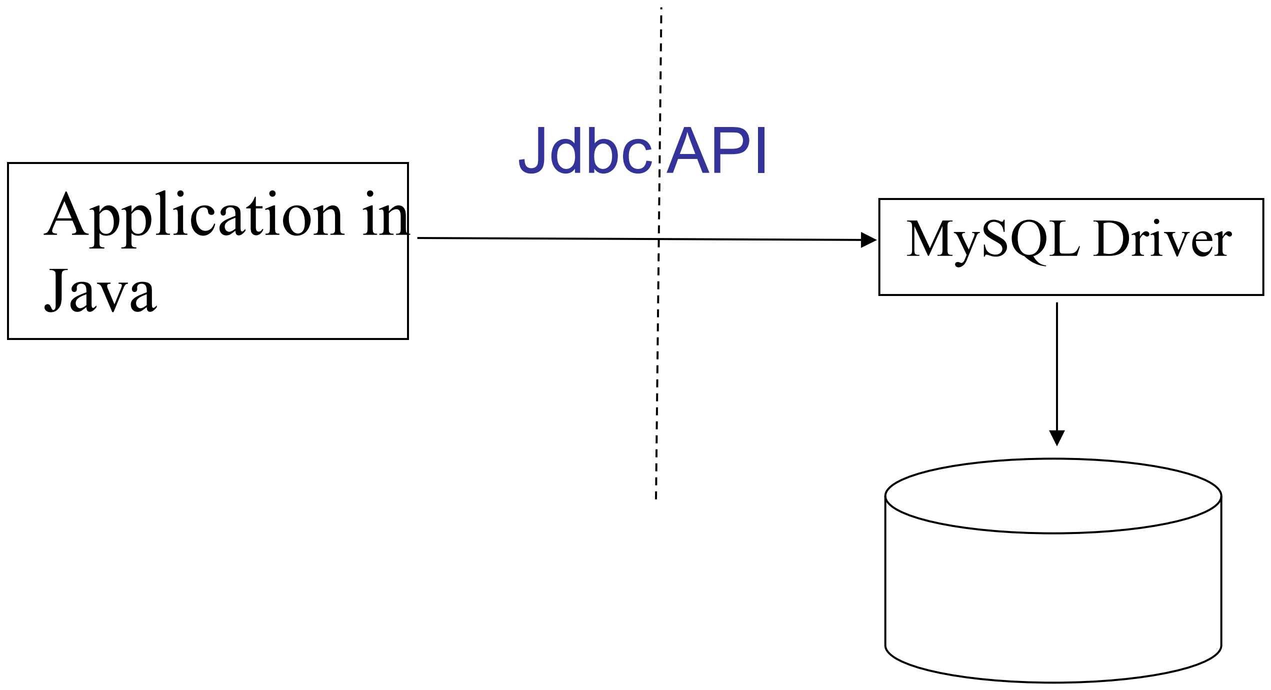
Java Support for Database Connectivity

- When applications written in Java want to access data sources, they use classes and associated methods provided by Java DBC (JDBC) API.
- JDBC is specified an "interface."
- An interface in Java can have many "implementations."
  - it provides a convenient way to realize many "drivers"
- JDBC can be an interface to an ODBC driver manager.
- Also it provides a JDBC API as a standard way to directly connect to common relational databases.

Application in Java



Application in Java



Java Support for SQL

- Java supports embedded SQL.
  - Embedded SQL in Java (SQLJ is one version of it) provides a standardized way for Java programs to interact with multiple databases, but at a higher level than the existing JDBC API.
  - Embedded SQL allows connecting to a database by including SQL code right in the program.
  - An Embedded SQL preprocessor converts the SQL statements to Java code at pre-compile time. The preprocessor generates code that includes the driver functionality for direct connection to the DBMS via JDBC.
- java.sql package and an extensive exception hierarchy.
- We will examine a direct pure Java JDBC driver using sample code.

An JDBC example

- JdbcExample.java

Data Source and Driver

- Data source: database created using any of the common database applications.
- Your system should have the driver for the database you will be using.
  - E.g., MS SQL Server on a Windows system.
- There are a number of JDBC drivers available.

JDBC Components

- **Driver Manager**: Loads database drivers, and manages the connection between application & driver.
- **Driver**: Translates API calls to operations for a specific data source.
- **Connection**: A session between an application and a driver.
- **Statement**: A SQL statement to perform a query or an update operation.
- **Metadata**: Information about the returned data, driver and the database.
- **Result Set** : Logical set of columns and rows returned by executing a statement.

JDBC Classes

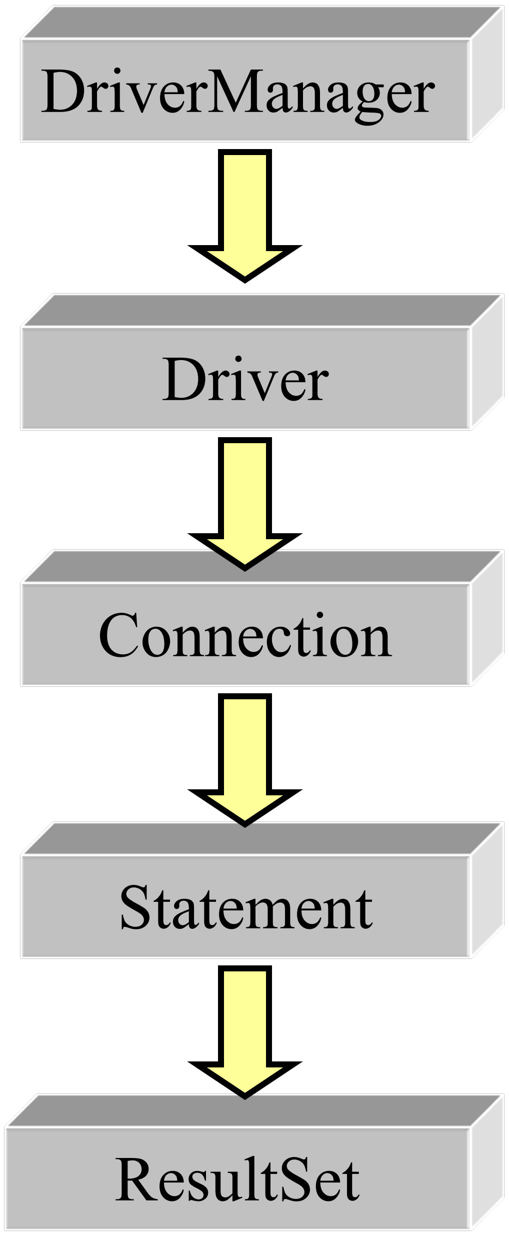
- Java supports DB facilities by providing classes and interfaces for its components
- **DriverManager** class
- **Connection** interface (abstract class)
- **Statement** interface (to be instantiated with values from the actual SQL statement)
- **ResultSet** interface

java.sql

- JDBC is implemented via classes in the java.sql package
  - Supports SQL-2 entry level
- Define objects for:
  - Remote connection to DB
  - Executing query
- 8 interfaces to define objects:
  - Statement, CallableStatement, PreparedStatement, DatabaseMetaData, ResultSetMetaData, ResultSet, Connection, Driver

Seven Steps

- Load the driver
- Define the Connection URL
- Establish the Connection
- Create a Statement object
- Execute a query
- Process the result
- Close the connection



Loading the Driver

- Registering the driver directly automatically:

```
Class.forName("com.mysql.jdbc.Driver");
```

- Calling **Class.forName**, which automatically
  - creates an instance of the driver
  - registers the driver with the **DriverManager**

Identifying Data Sources

- Gives the required information for making the connection to the database
- Specified using the URL format.

    <scheme>: <sub_scheme>:<scheme-specific-part>

- Example:

```
jdbc:postgresql://foo.ics.uci.edu/mydatabase
jdbc:postgresql://localhost/testdb
jdbc:mysql:///mydb
```

Connection

- A Connection represents a session with a specific database
- Within the context of a Connection, SQL statements are executed and results are returned

Connections

- There can be multiple connections to a database
- A connection provides "metadata", i.e., information about the database, tables, and fields
- Connection object has methods to deal with transactions

Creating a Connection

- Use getConnection on the Driver

```
Connection getConnection
           (String url,
            String user,
            String password)
```

- Connects to given JDBC URL with given user name and password
- Throws java.sql.SQLException
- returns a Connection object

Creating a Connection

```
Connection connection =
  DriverManager.getConnection("jdbc:my
  sql:///mydb", "testuser",
  "mypassword");
```

Statements

- **Statement createStatement()**
  - returns a new Statement object
  - Used for general queries
- **PreparedStatement prepareStatement(String sql)**
  - returns a new PreparedStatement object
  - For a statement called multiple times with different values (precompiled to reduce parsing time)
- **CallableStatement prepareCall(String sql)**
  - returns a new CallableStatement object
  - For stored procedures

Statements

- A Statement object is used for executing a static SQL statement and obtaining the results produced by it
- executeQuery is used for statements that return an output result
- executeUpdate is used for statements that need not return an output

Executing Queries and Updates

- ResultSet executeQuery(String)
  - Execute a SQL statement that returns a single ResultSet
- int executeUpdate(String)
  - Execute a SQL INSERT, UPDATE or DELETE statement
  - Used for CREATE TABLE, DROP TABLE and ALTER TABLE
  - Returns the number of rows changed

Timeout

- Use setQueryTimeOut to set a timeout for the driver to wait for a statement to be completed
- If the operation is not completed in the given time, an SQLException is thrown
- What is it good for?

Cursor

- What is the result of a query?
- How can a database send the result of a query through communication lines?

- The answer: using a cursor

Result Set

- A ResultSet provides access to a table of data generated by executing a Statement
- Only one ResultSet per Statement can be open at once
- The table rows are retrieved in sequence
- A ResultSet maintains a cursor pointing to its current row of data
- The 'next' method moves the cursor to the next row
  - you can't rewind

Working with ResultSet

- boolean next()
  - activates the next row
  - the first call to next() activates the first row
  - returns false if there are no more rows
- void close()
  - disposes of the ResultSet
  - automatically called by most Statement methods

Getting Values from Rows

- *Type* get *Type*(int columnIndex)
  - returns the given field as the given type
  - E.g., int getInt(5); string getString(3);
  - fields indexed starting at 1 (not 0)
- *Type* get *Type*(String columnName)
  - same, but uses name of field
  - less efficient
- int findColumn(String columnName)
  - looks up column index given column name

isNull

- In SQL, NULL means the field is empty
- Not the same as 0 or “”
- In JDBC, you must explicitly ask if a field is null by calling ResultSet.isNull(column)

Mapping Java Types to SQL Types

SQL type	**Java Type**
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Database Time

- Times in SQL are nonstandard
- Java defines three classes to help
- java.sql.Date
  - year, month, day
- java.sql.Time
  - hours, minutes, seconds
- java.sql.Timestamp
  - year, month, day, hours, minutes, seconds, nanoseconds
  - usually use this one

Optimized Statements

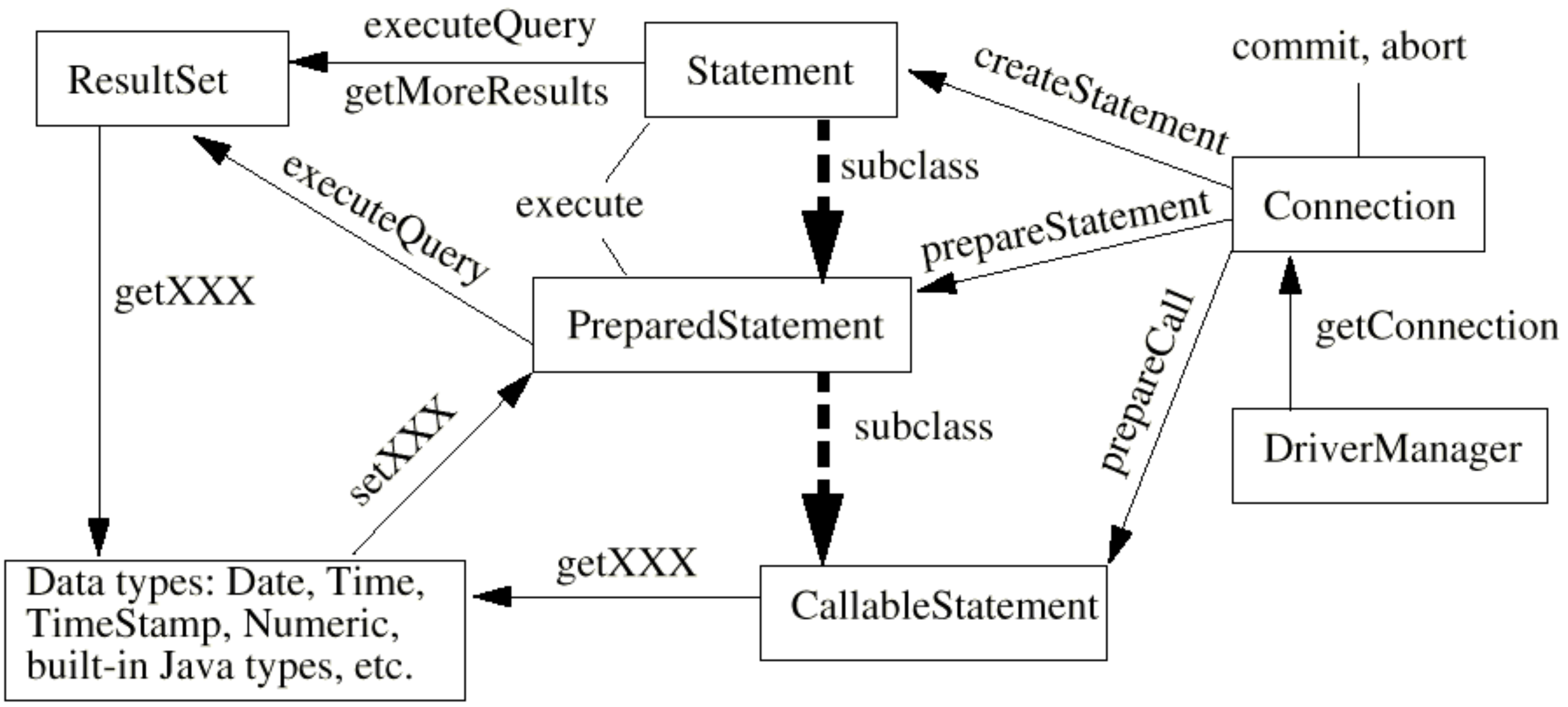
- Prepared Statements
  - SQL calls that you make again and again
  - allows driver to optimize (compile) queries
  - created with `Connection.prepareStatement()`
- Stored Procedures
  - written in DB-specific language
  - stored inside database
  - accessed with `Connection.prepareCall()`

Prepared Statement Example

```
PreparedStatement updateSales;
String updateString = "update COFFEES " +
            "set SALES = ? where COF_NAME like ?";
updateSales = con.prepareStatement(updateString);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast",
  "Espresso","Colombian_Decaf","French_Roast_Decaf"};

int len = coffees.length;
for(int i = 0; i < len; i++) {
  updateSales.setInt(1, salesForWeek[i]);
  updateSales.setString(2, coffees[i]);
  updateSales.executeUpdate();
}
```

JDBC Class Diagram



Metadata

- Connection:
  - DatabaseMetaData getMetaData()
- ResultSet:
  - ResultSetMetaData getMetaData()

ResultSetMetaData

- What's the number of columns in the ResultSet?
- What's a column's name?
- What's a column's SQL type?
- What's the column's normal max width in chars?
- What's the suggested column title for use in printouts and displays?
- What's a column's number of decimal digits?
- Does a column's case matter?
- and so on...

DatabaseMetaData

- What tables are available?
- What's our user name as known to the database?
- Is the database in read-only mode?
- If table correlation names are supported (association of a column with the table it comes from, when multiple columns of the same name appear in the same query - multi-table queries) , are they restricted to be different from the names of the tables?
- and so on…

Useful Methods of Metadata

- getColumnCount
- getColumnDisplaySize
- getColumnName
- getColumnType
- isNullabale

Imagine the case where you want to print the result

Transaction Management

- A transaction: a sequence of SQL statements
- Transactions are <u>not</u> explicitly opened and closed
- Instead, the connection has a state called AutoCommit mode
- if AutoCommit is true, then every statement is automatically committed
- default case: true

AutoCommit

Connection.setAutoCommit(boolean)

- if AutoCommit is false, then every statement is added to an ongoing transaction
- you must explicitly commit or rollback the transaction using Connection.commit() and Connection.rollback()

AutoCommit (cont)

Check:

- http://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html
- http://www.devx.com/tips/Tip/15015
- There's a slight difference between doing them in SQL and JDBC: auto-commit control and commit call in JDBC are Java calls (not SQL).