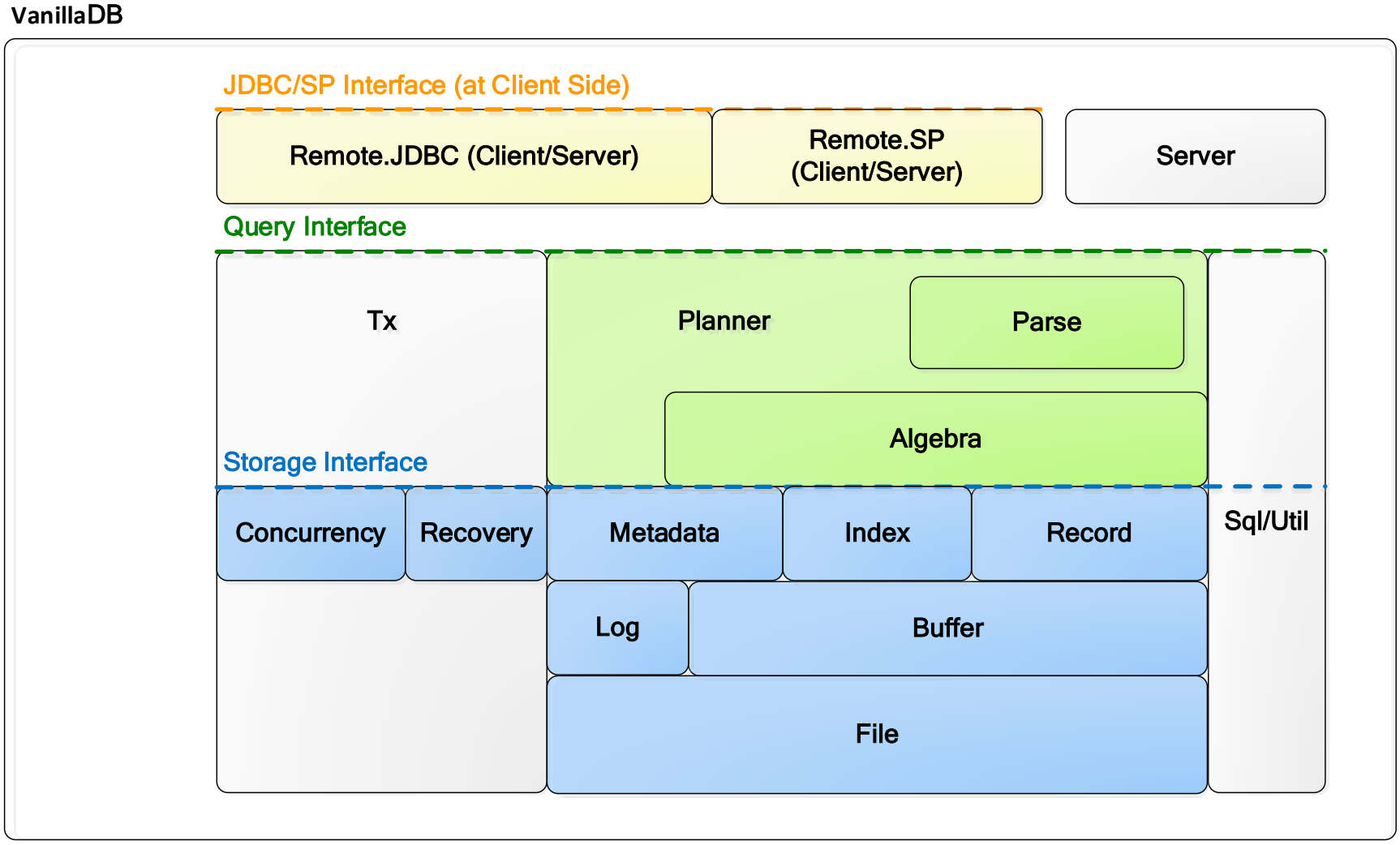
VanillaCore Walkthrough Part 1

Introduction to Database Systems

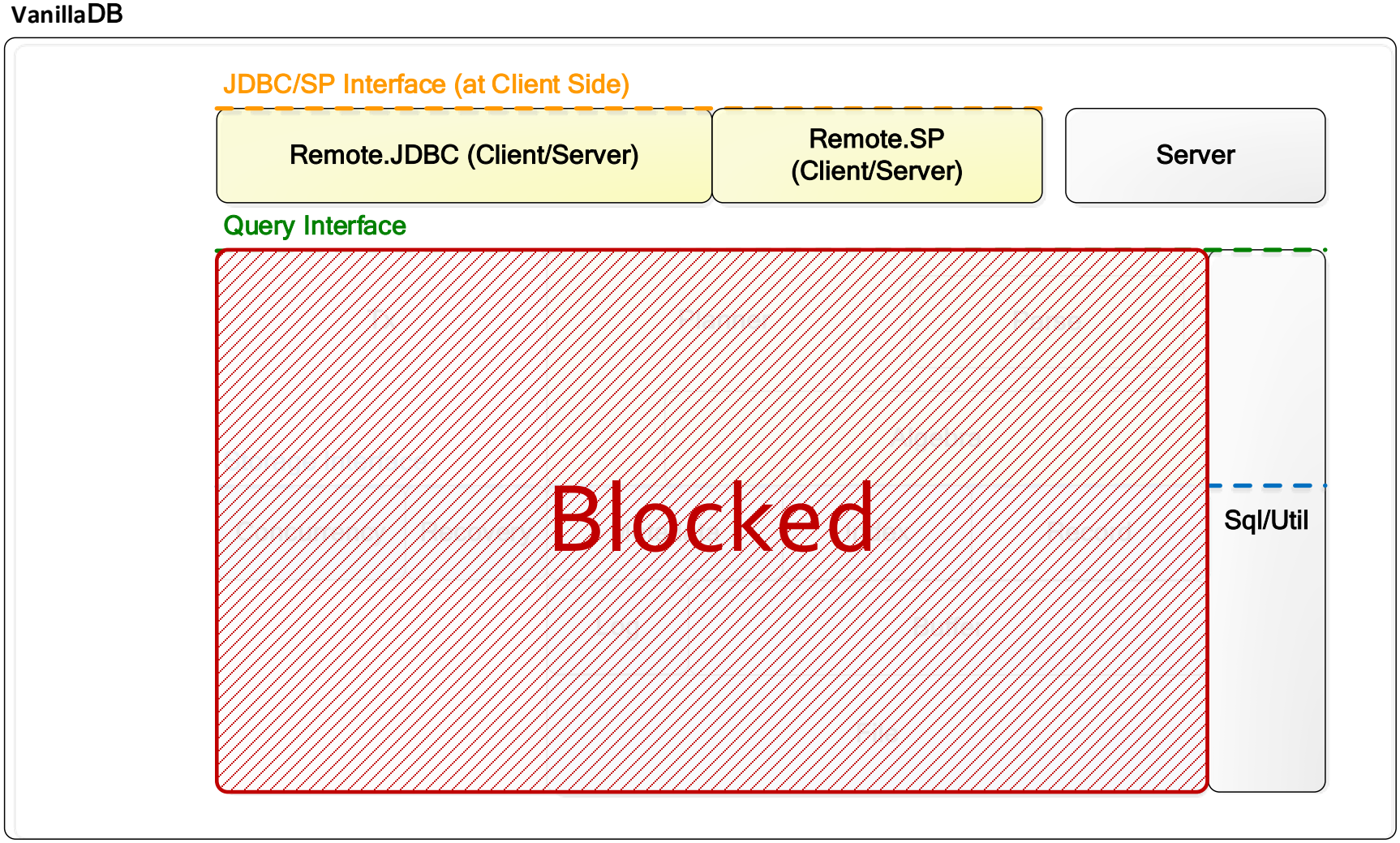
DataLab

CS, NTHU

The Architecture



The Architecture




200
200
100
200
4/5
黑暗时代
AGE OF EMPIRES
HD EDITION

Fog

- Now, you can only see a part of VanillaCore.
- As the progress of the course, we will open more packages in the future.

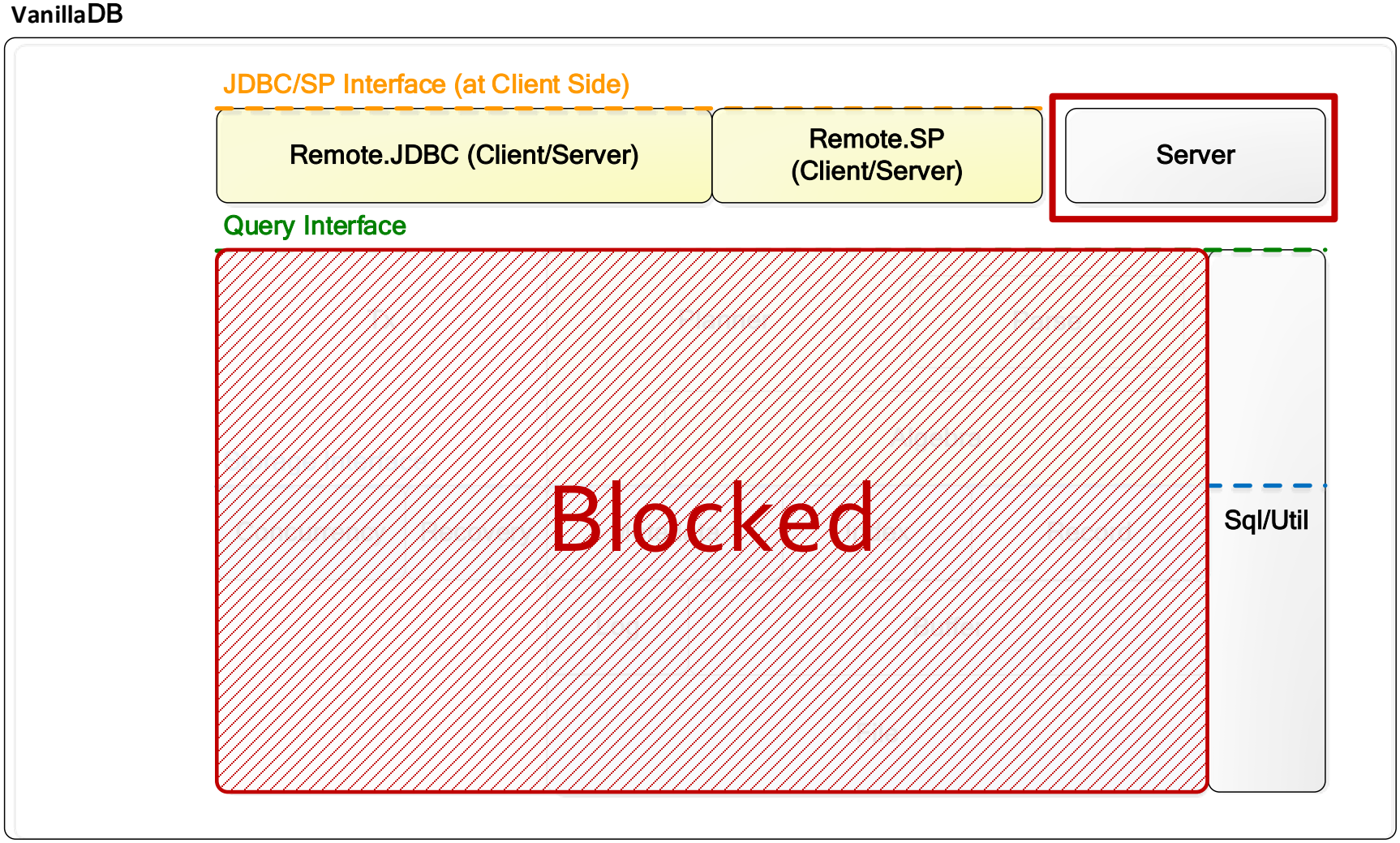
Outline

- Server package
- Remote package
- SQL package

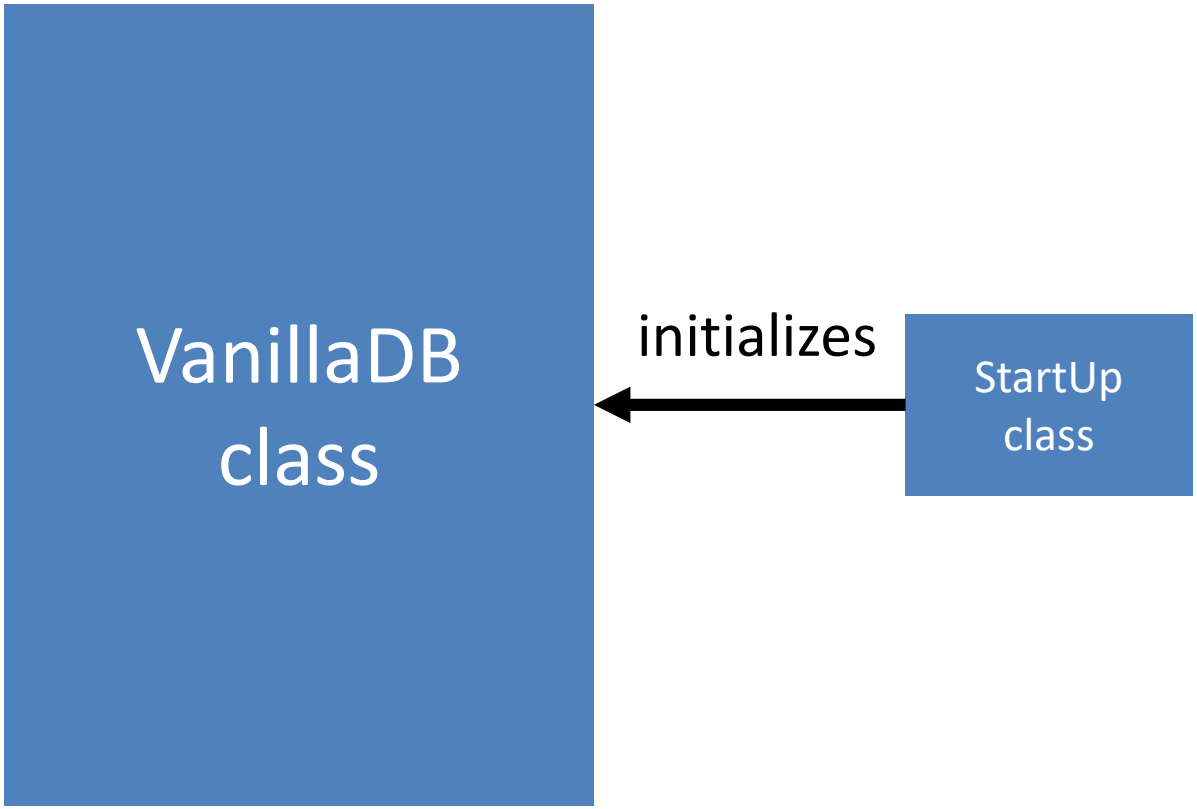
Outline

- Server package
- Remote package
- SQL package

Where are we?

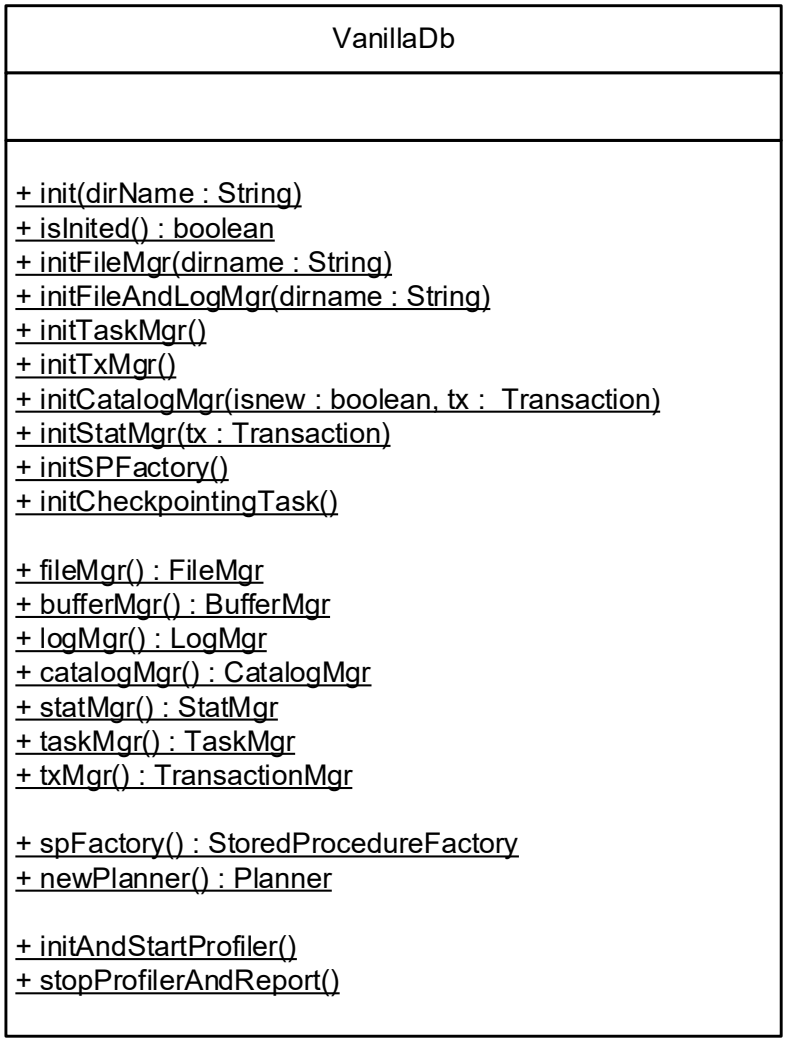

VanillaDB
JDBC/SP Interface (at Client Side)
Remote.JDBC (Client/Server)
Remote.SP (Client/Server)
Server
Query Interface
Blocked
Sql/Util

`server` Package



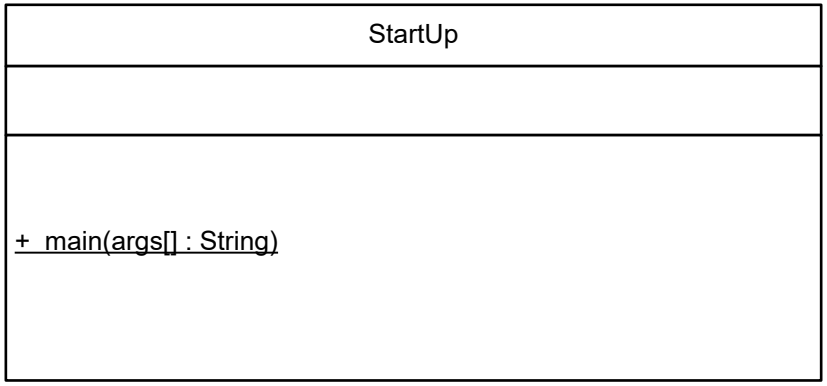
VanillaDb

- There are four types of methods
  - Initialization
  - Global getters
  - Factory methods
  - Profiler



`StartUp`

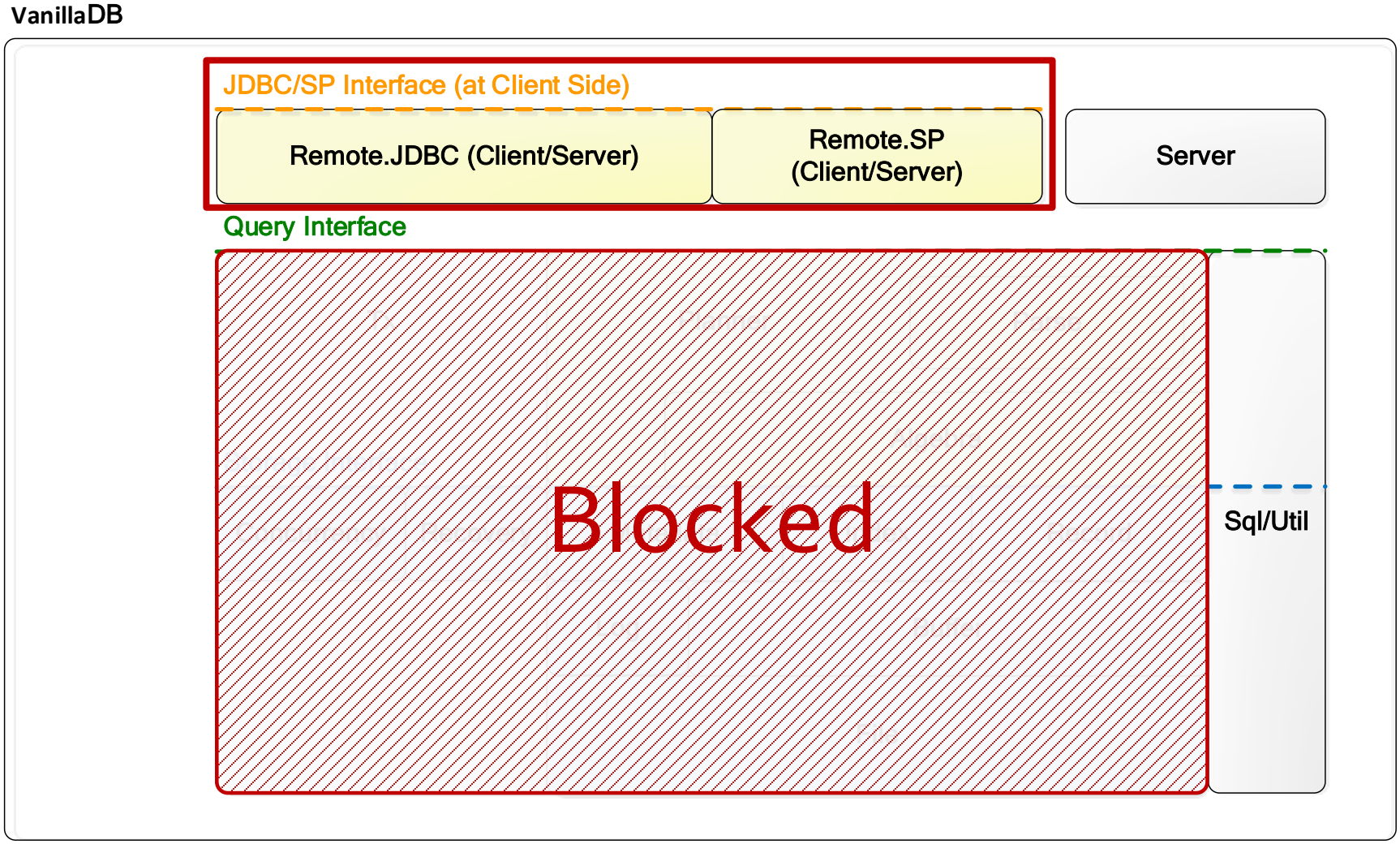
- `StartUp` provides `main()` that runs VanillaCore as a JDBC server
  - Calls `VanillaDB.init()`
    - Sharing global resources through static variables
  - Binds `RemoteDriver` to RMI registry
    - Thread per connection



Outline

- Server package
- Remote package
- SQL package

Where are we?


VanillaDB
JDBC/SP Interface (at Client Side)
Remote.JDBC (Client/Server)
Remote.SP (Client/Server)
Server
Query Interface
Blocked
Sql/Util

`remote` Package

JDBC Package

Stored Procedure Package

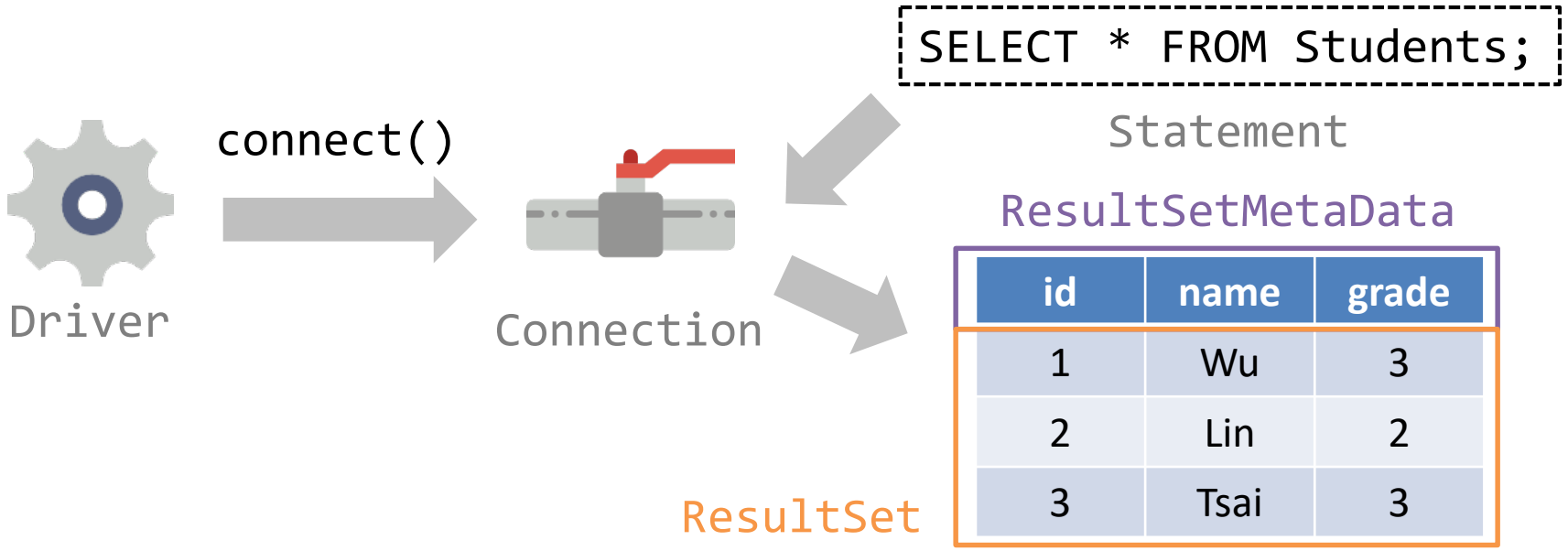
`remote` Package

JDBC Package

Stored Procedure Package

JDBC

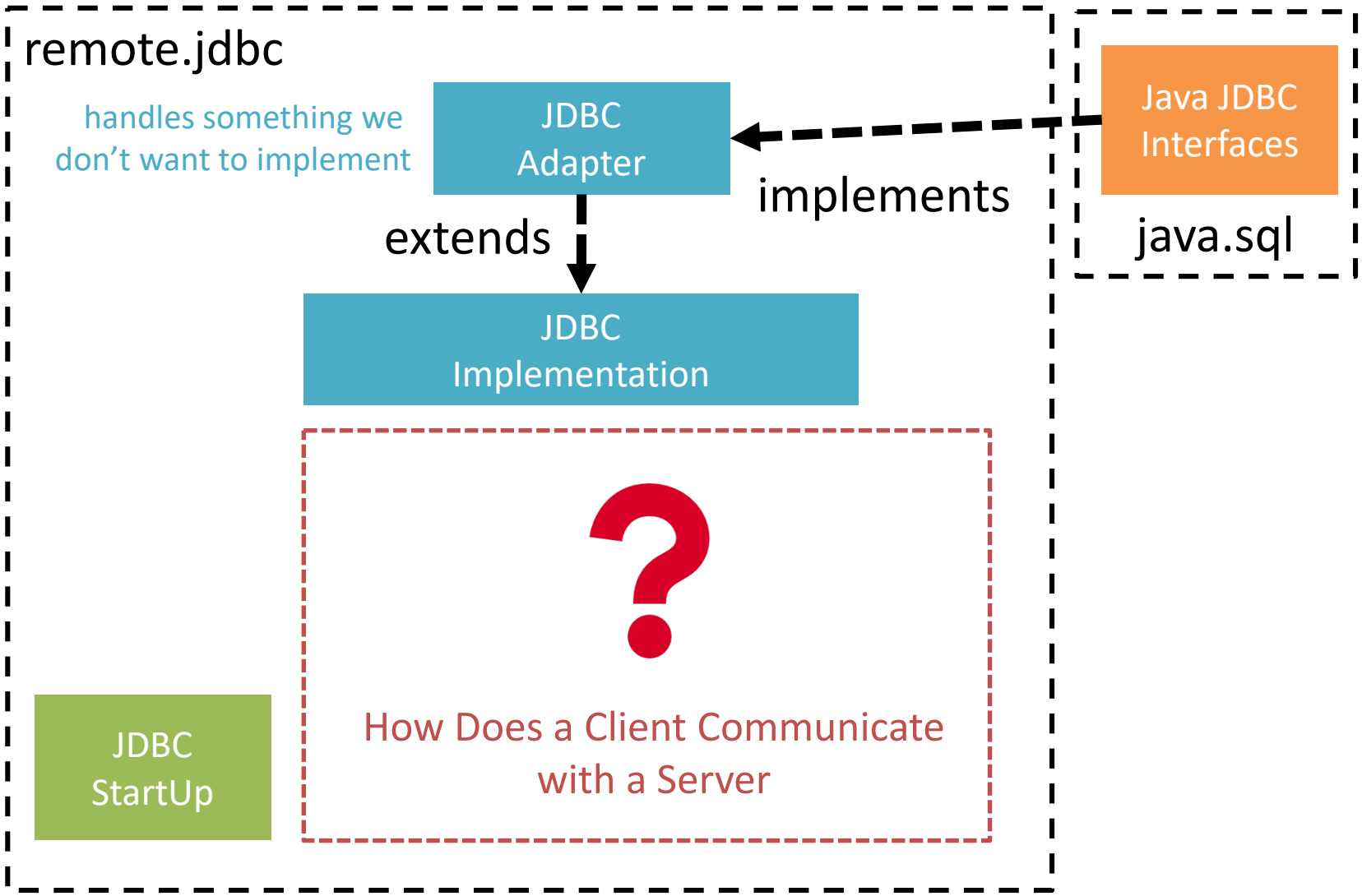
- Java Database Connectivity (JDBC) is an API for Java, that defines how a client may access a database.



JDBC Program: Finding Major

```
Connection conn = null;
try {
	// Step 1: connect to database server
	Driver d = new JdbcDriver();
	conn = d.connect("jdbc:vanilladb://localhost", null);
	conn.setAutoCommit(false);
	conn.setReadOnly(true);
	// Step 2: execute the query
	Statement stmt = conn.createStatement();
	String qry = "SELECT s-name, d-name FROM departments, "
	+ "students WHERE major-id = d-id";
	ResultSet rs = stmt.executeQuery(qry);
	// Step 3: loop through the result set
	rs.beforeFirst();
	System.out.println("name\tmajor");
	System.out.println("-------\t-------");
	while (rs.next()) {
		String sName = rs.getString("s-name");
		String dName = rs.getString("d-name");
		System.out.println(sName + "\t" + dName);
	}
	rs.close();
} catch (SQLException e) {
	e.printStackTrace();
} finally {
	try {
		// Step 4: close the connection
		if (conn != null)
		conn.close();
	} catch (SQLException e) {
		e.printStackTrace();
	}
}
```

`remote.jdbc` Package



RMI

- VanillaCore uses Java Remote Method Invocation (RMI) for communication.
  - It makes a program able to call a method on other program without knowing the implementation of the method.

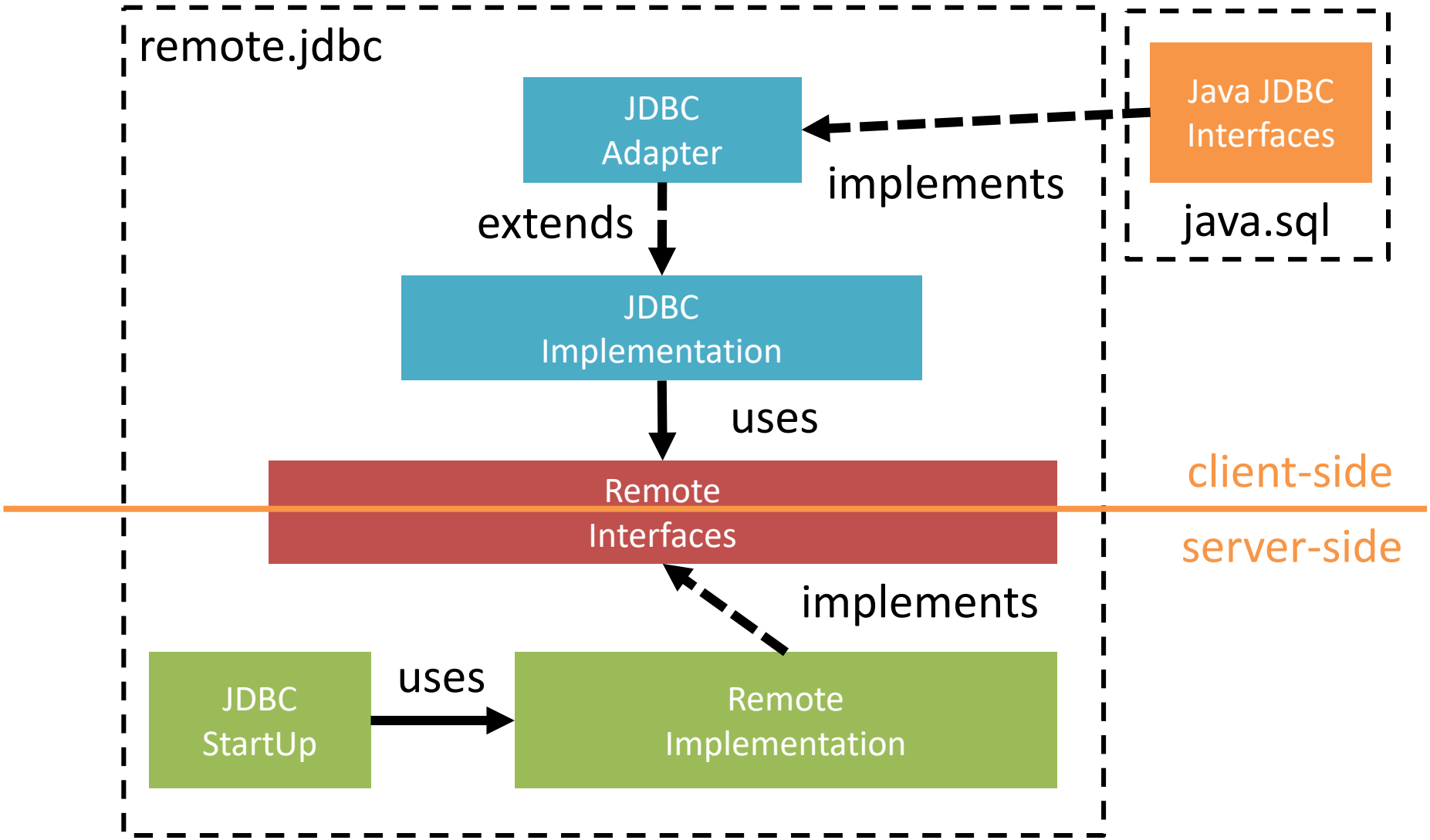
RMI Example

```
public interface API {
        int[] sort(int[] numbers);
}
```

```
public class Server implements API {
     @Override
     public int[] sort(int[] numbers) {
          int[] array = Arrays.copyOf(numbers, numbers.length);
          Arrays.sort(array);
          return array;
     }
}
```

```
public class Client {
     public static void main(String[] args) {
          ...
          Registry reg = LocateRegistry.getRegistry(host);
          API api = (API) reg.lookup(regName);
          array = api.sort(array);
     }
}
```

`remote.jdbc` Package

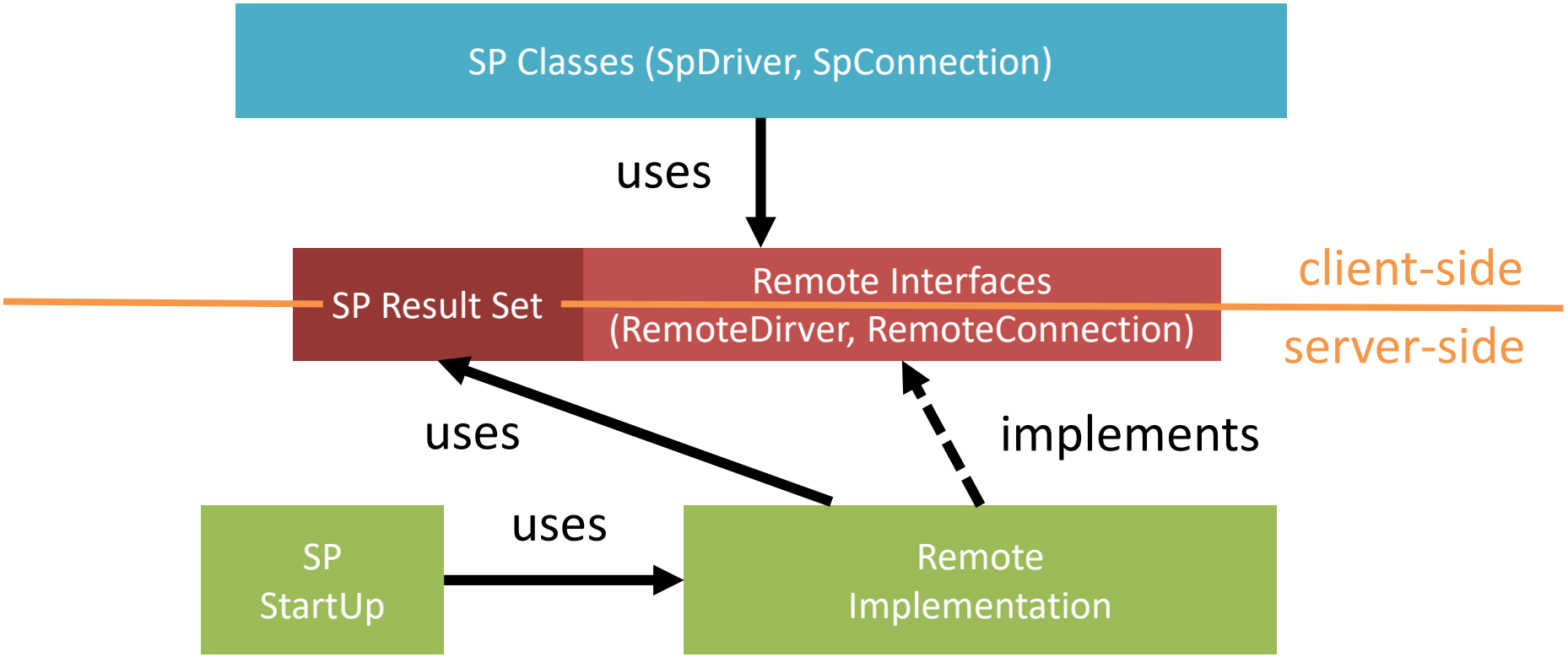


`remote` Package

JDBC Package

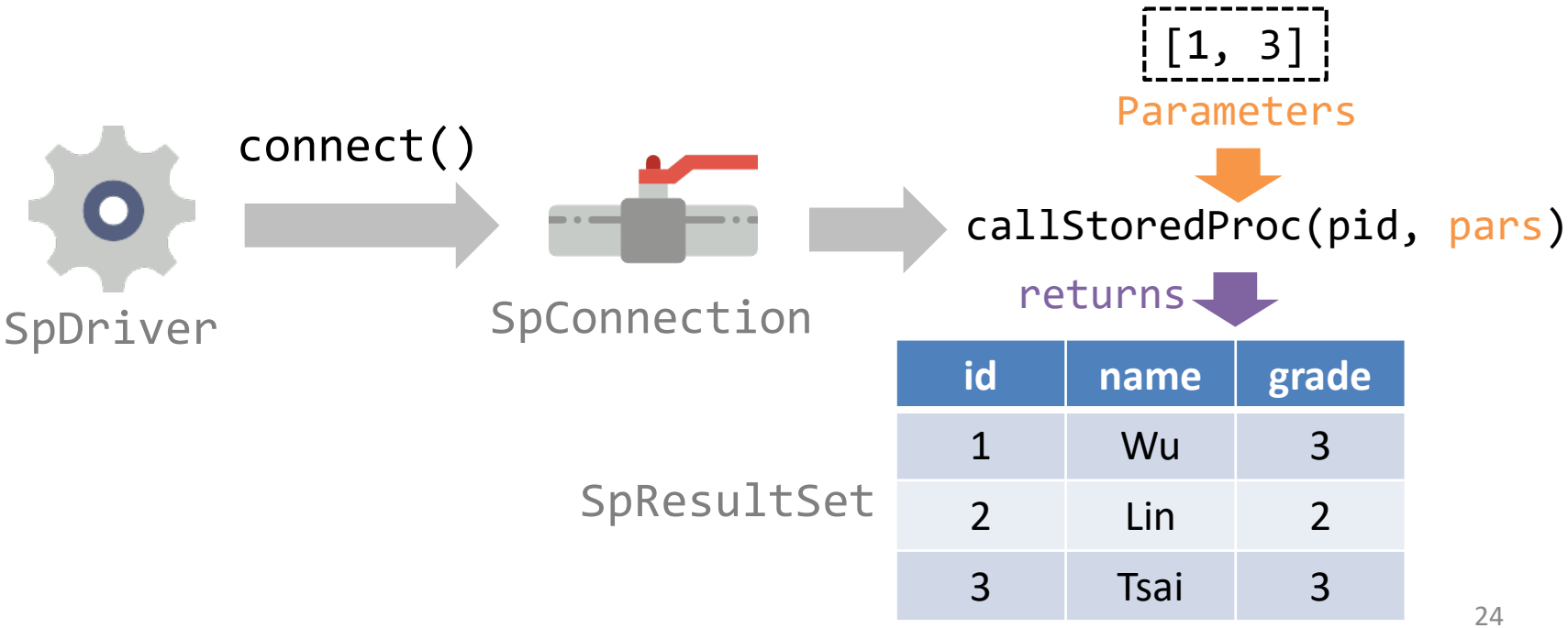
Stored Procedure Package

`remote.storedprocedure` Package



Calling Stored Procedure

- To call a stored procedure from clients, it first establishes a connection from the driver.
  - Then send the parameters via the connection

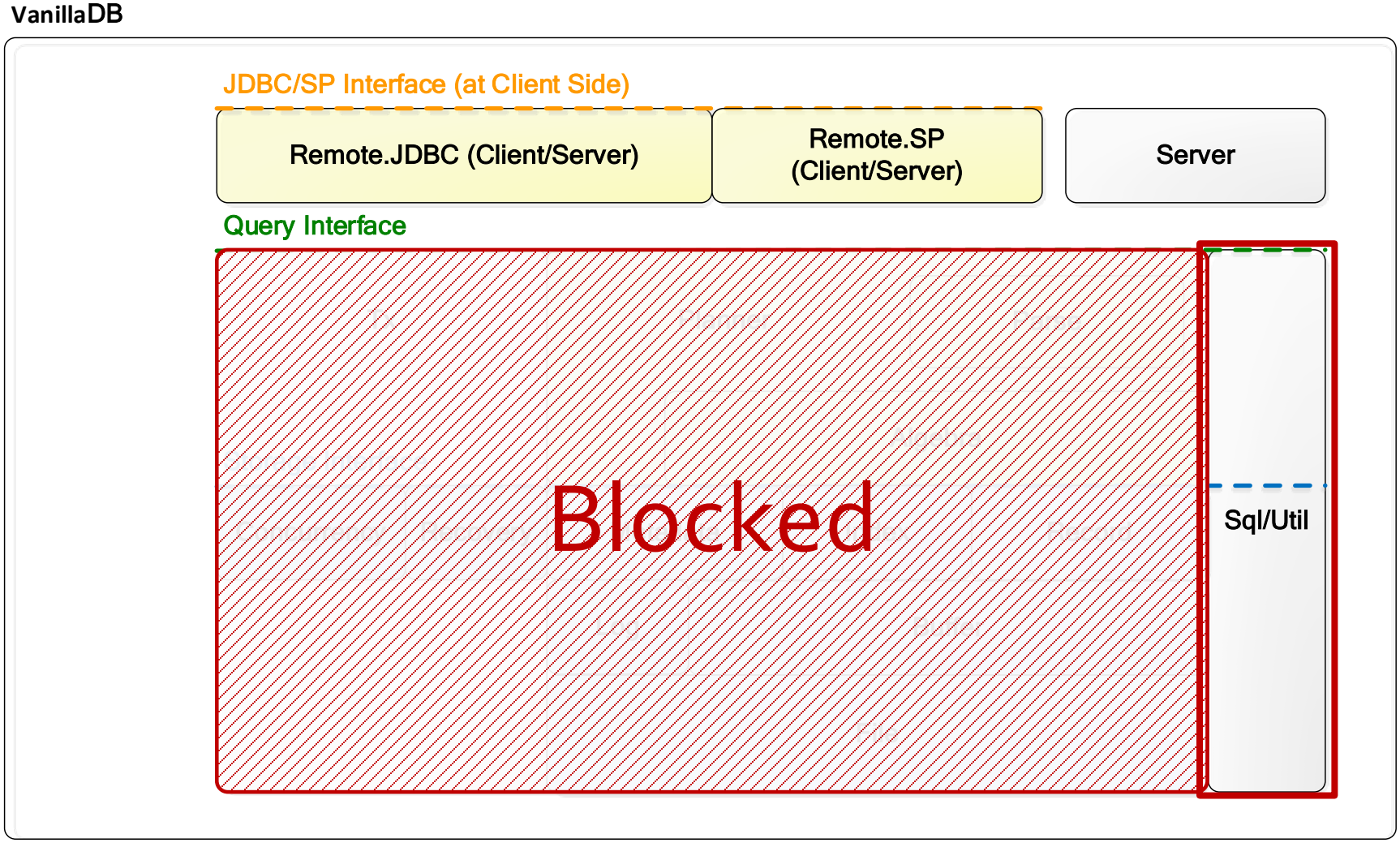


id	name	grade
1	Wu	3
2	Lin	2
3	Tsai	3

Outline

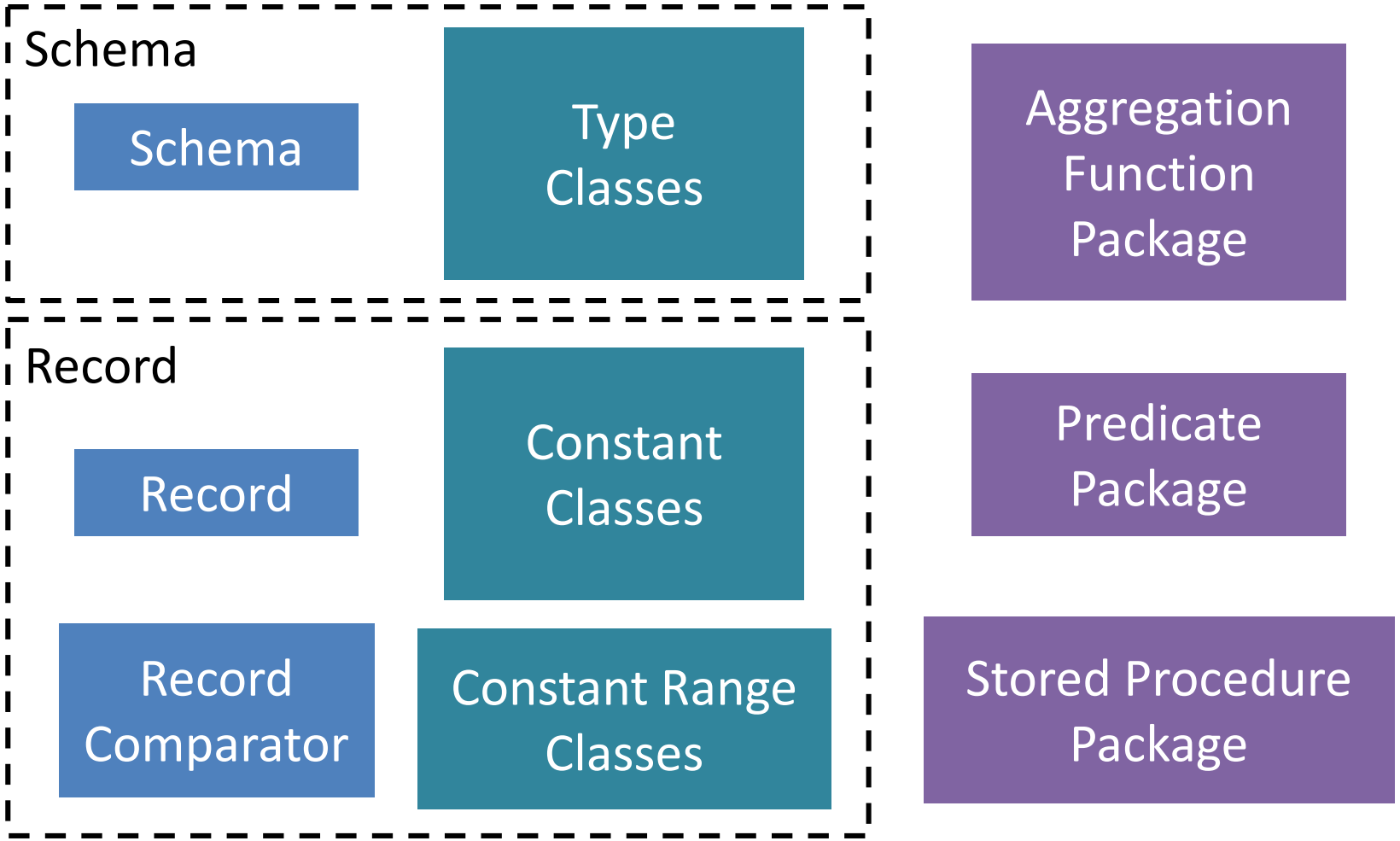
- Server package
- Remote package
- SQL package

Where are we?


VanillaDB
JDBC/SP Interface (at Client Side)
Remote.JDBC (Client/Server)
Remote.SP (Client/Server)
Server
Query Interface
Blocked
Sql/Util

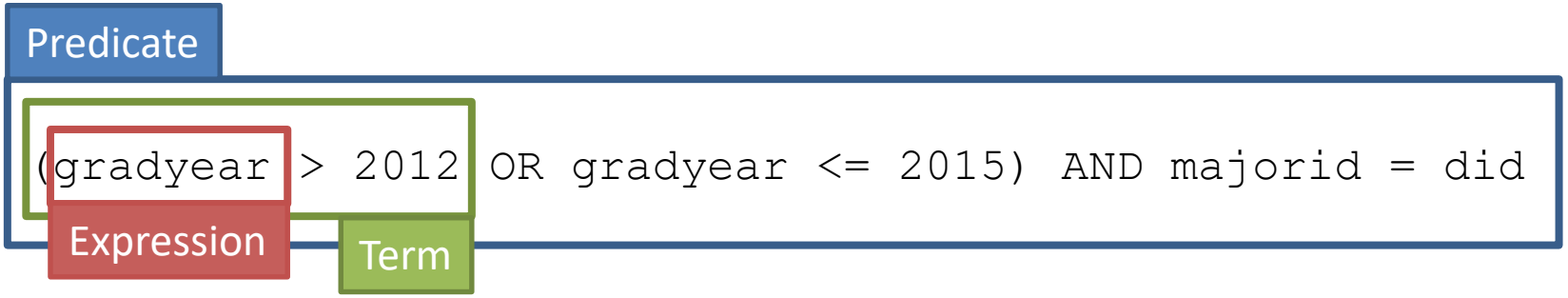
Schema & Records

`sql` Package

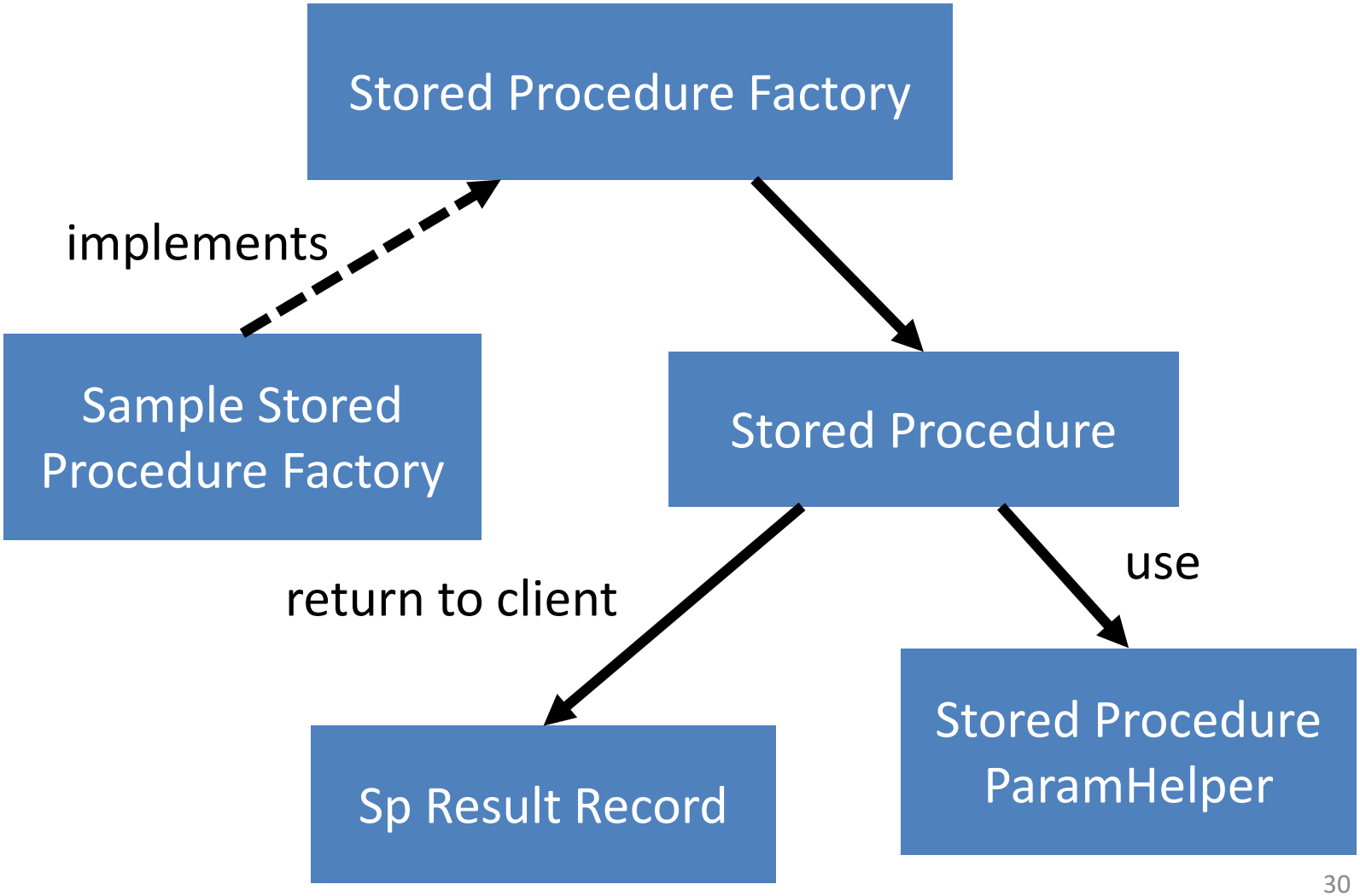

Schema
Schema
Type Classes
Record
Record
Constant Classes
Record Comparator
Constant Range Classes
Aggregation Function Package
Predicate Package
Stored Procedure Package

Predicates

- An ***expression*** consists of constants, field names, or their operations
- A ***term*** is a comparison between two expressions
- A ***predicate*** is a Boolean combination of terms



`sql.storedprocedure` Package



Factory Pattern

- A factory takes care of which implementation should be used.
- The clients only need to pass the parameters to it and wait the results.