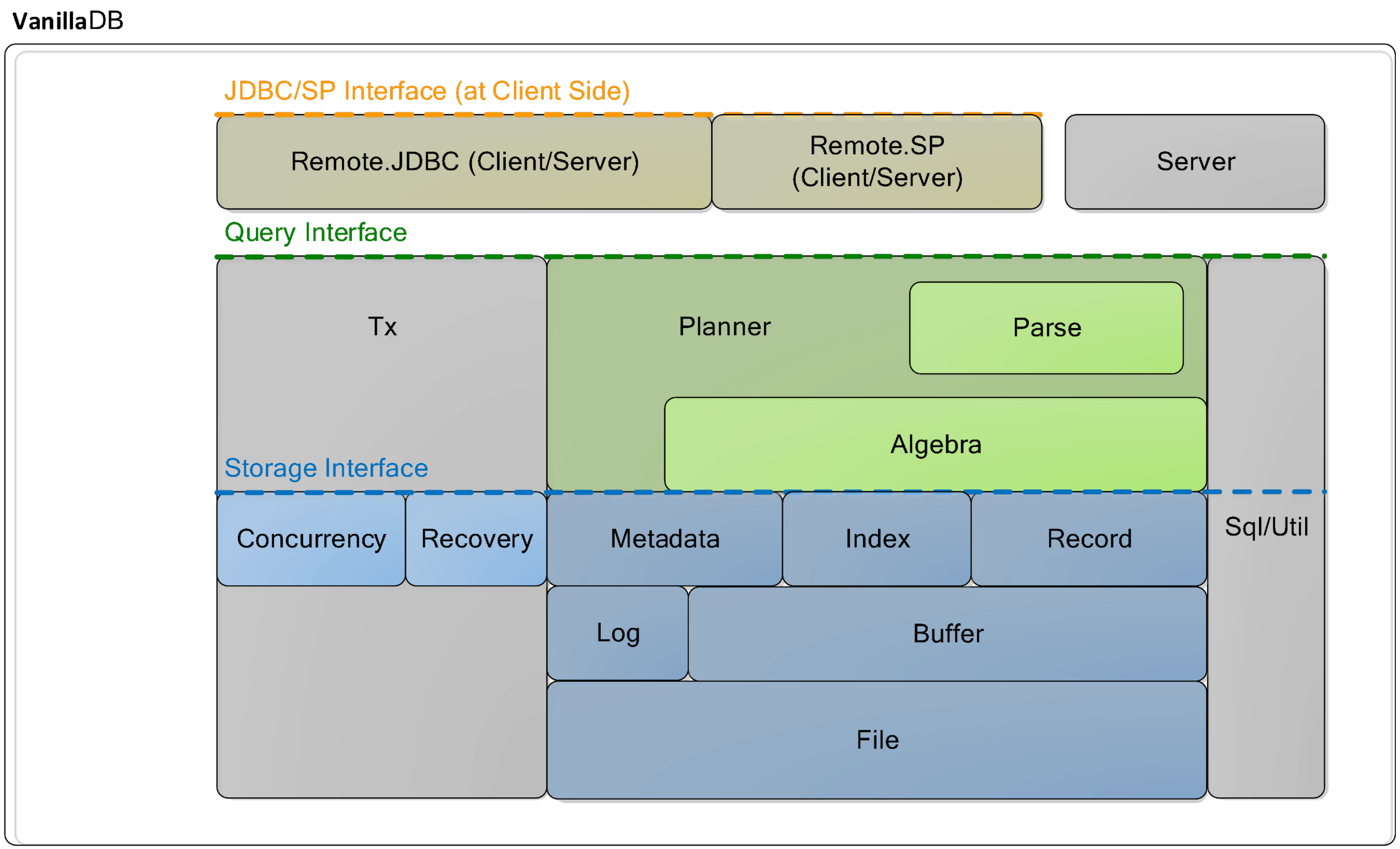
VanillaCore Walkthrough
Part 1

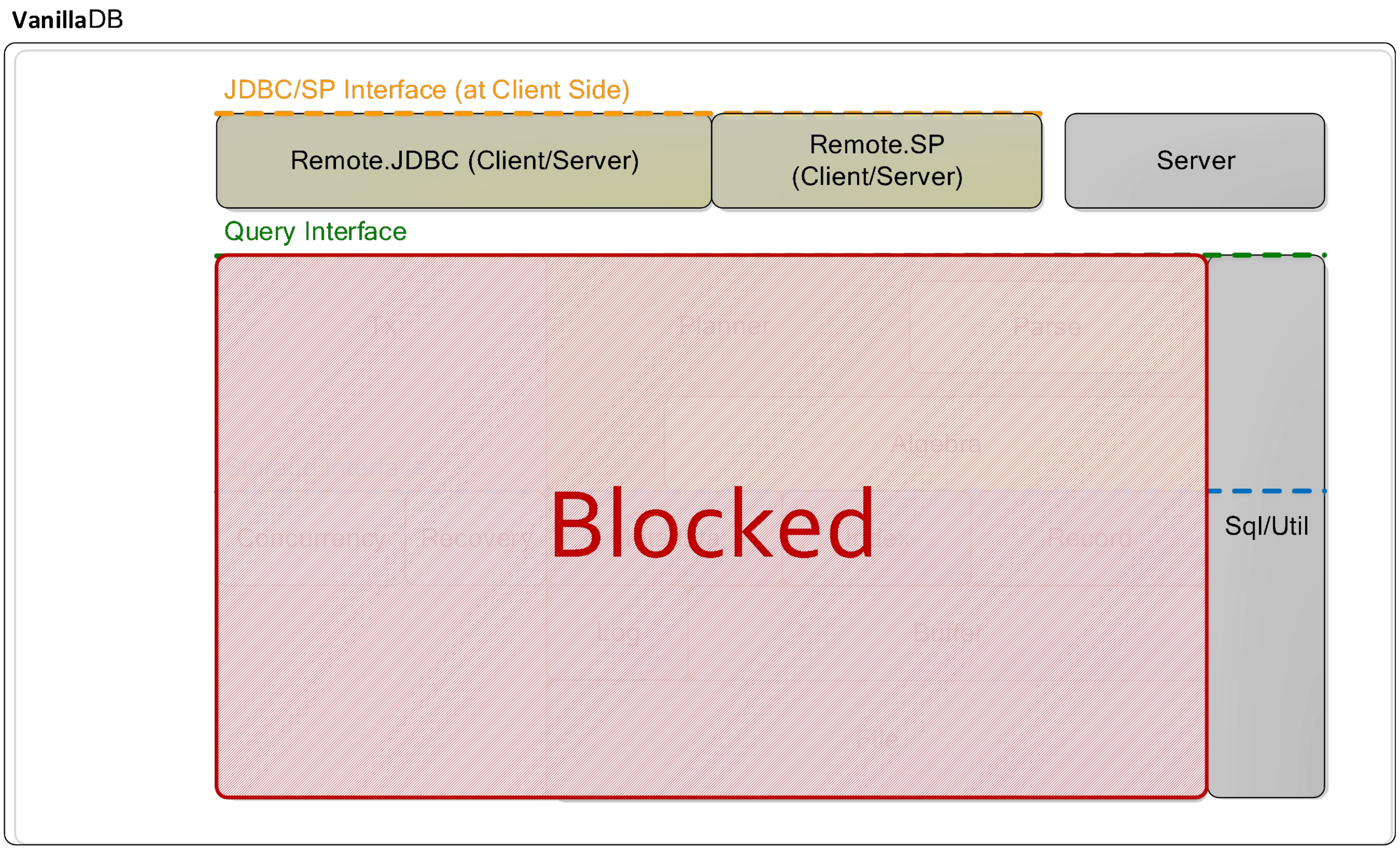
Introduction to Database Systems
2024

DataLab, CS, NTHU

The Architecture



The Architecture



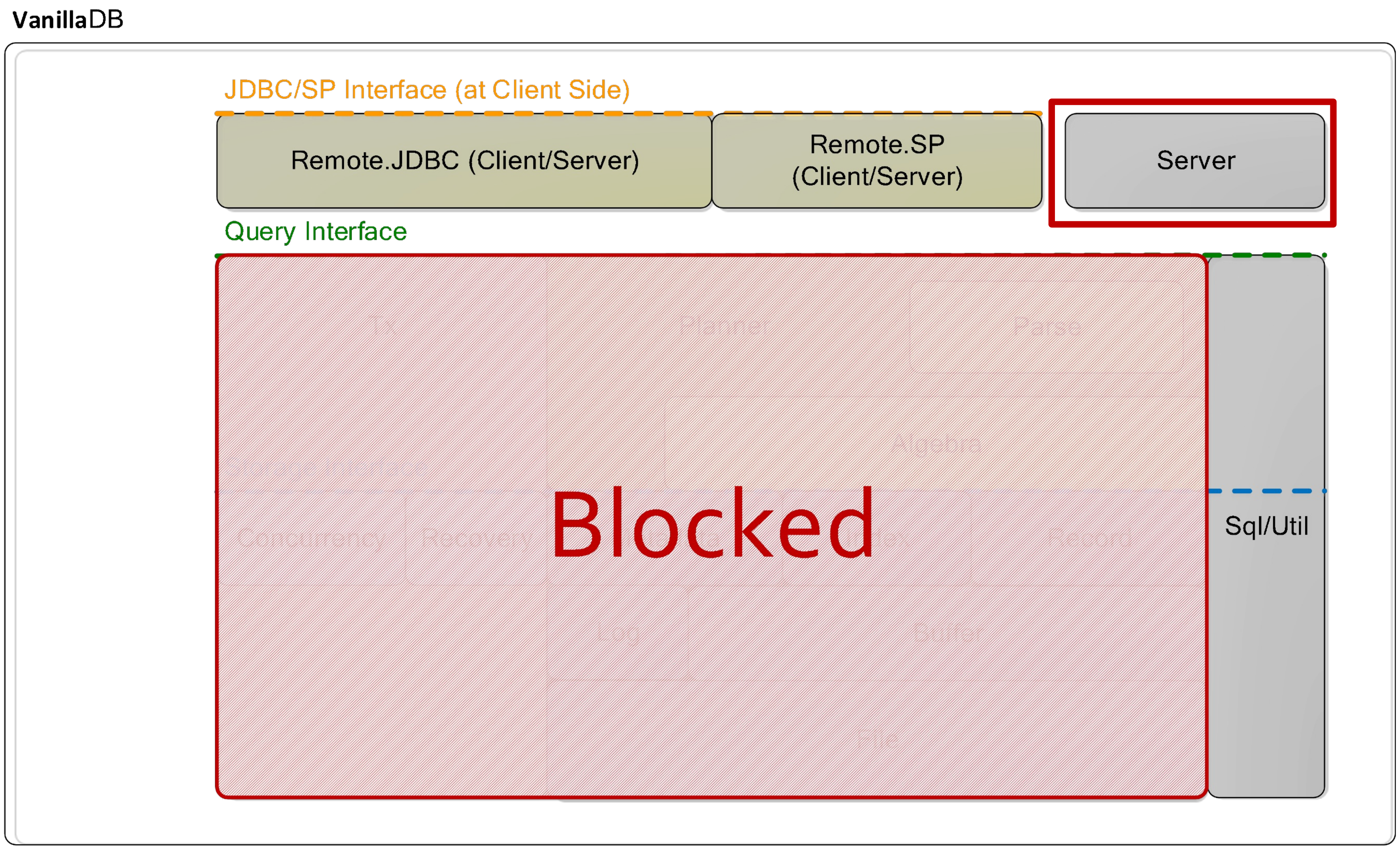
Outline

- Server package
- Remote package

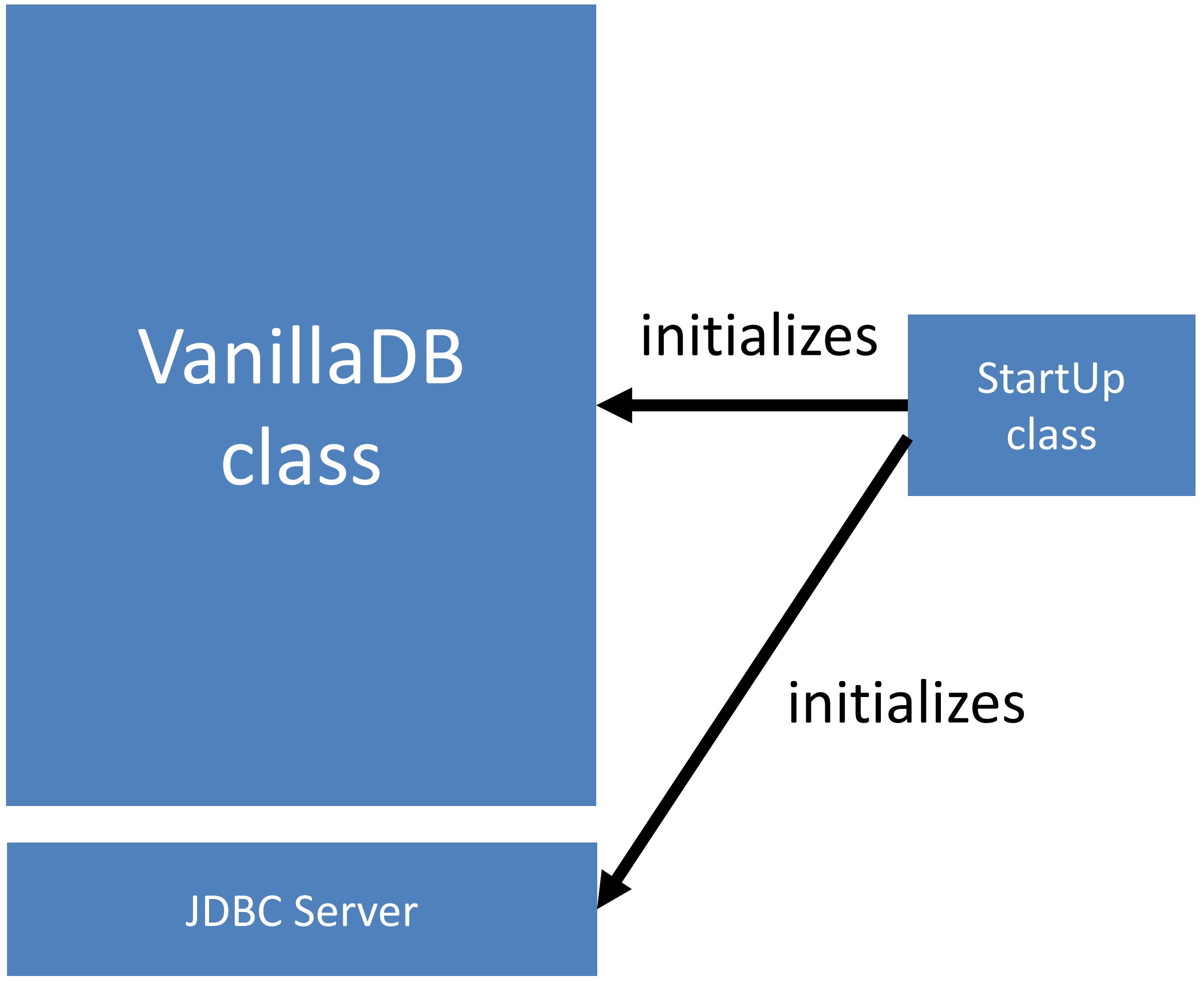
Outline

- Server package
- Remote package

Where are we?

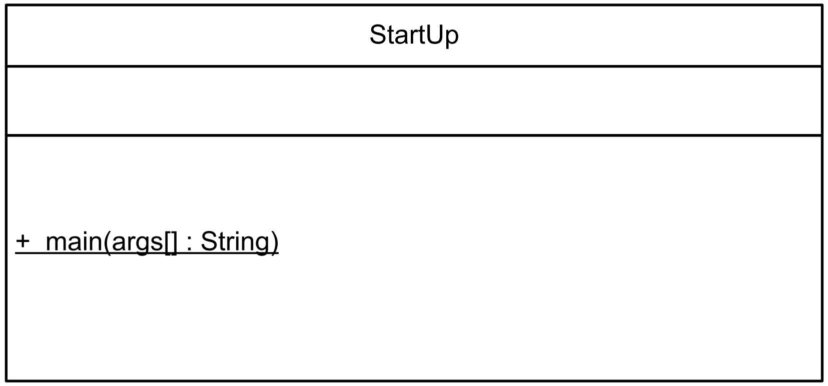

VanillaDB
JDBC/SP Interface (at Client Side)
Remote.JDBC (Client/Server)
Remote.SP (Client/Server)
Server
Query Interface
Blocked
Sql/Util

`server` Package



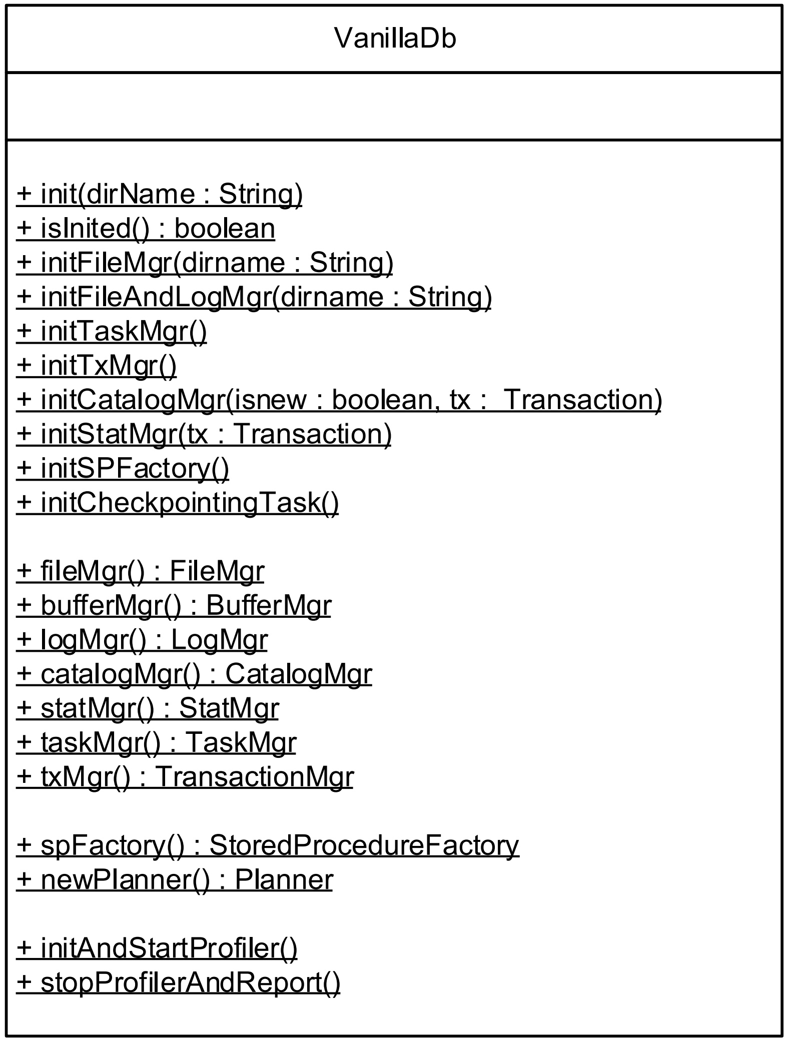
`StartUp`

- `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



VanillaDb

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



VanillaDb

+ init(dirName : String)
+ isInited() : boolean
+ initFileMgr(dirname : String)
+ initFileAndLogMgr(dirname : String)
+ initTaskMgr()
+ initTxMgr()
+ initCatalogMgr(isnew : boolean, tx : Transaction)
+ initStatMgr(tx : Transaction)
+ initSPFactory()
+ initCheckpointingTask()

+ fileMgr() : FileMgr
+ bufferMgr() : BufferMgr
+ logMgr() : LogMgr
+ catalogMgr() : CatalogMgr
+ statMgr() : StatMgr
+ taskMgr() : TaskMgr
+ txMgr() : TransactionMgr

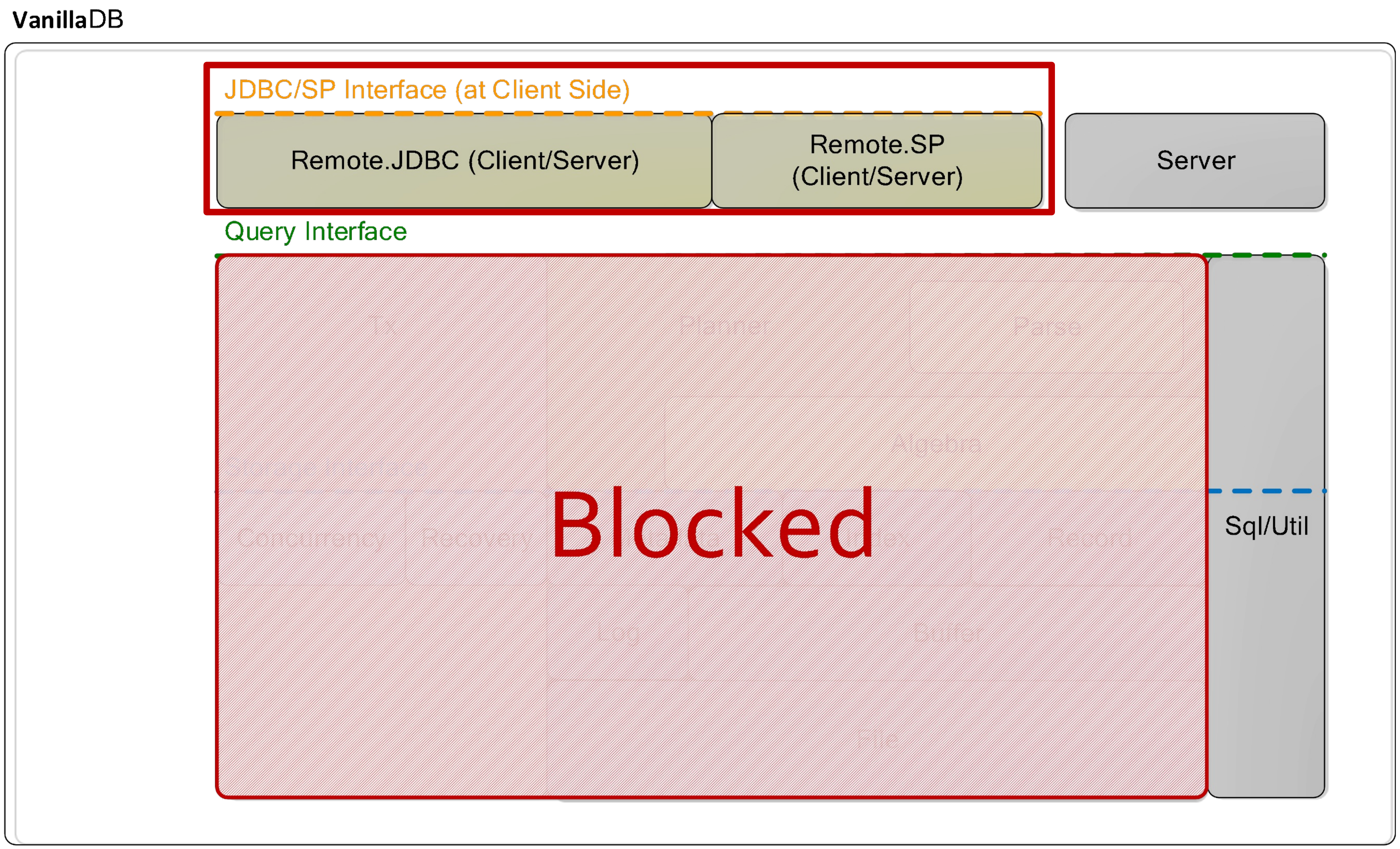
+ spFactory() : StoredProcedureFactory
+ newPlanner() : Planner

+ initAndStartProfiler()
+ stopProfilerAndReport()

Outline

- Server package
- Remote package

Where are we?


VanillaDB
JDBC/SP Interface (at Client Side)
Remote.JDBC (Client/Server)
Remote.SP (Client/Server)
Server
Query Interface
Tx
Planner
Parse
Algebra
Concurrency
Recovery
Index
Record
Log
Buffer
File
Blocked
Sql/Util

`remote` Package

JDBC
Package

Stored Procedure
Package

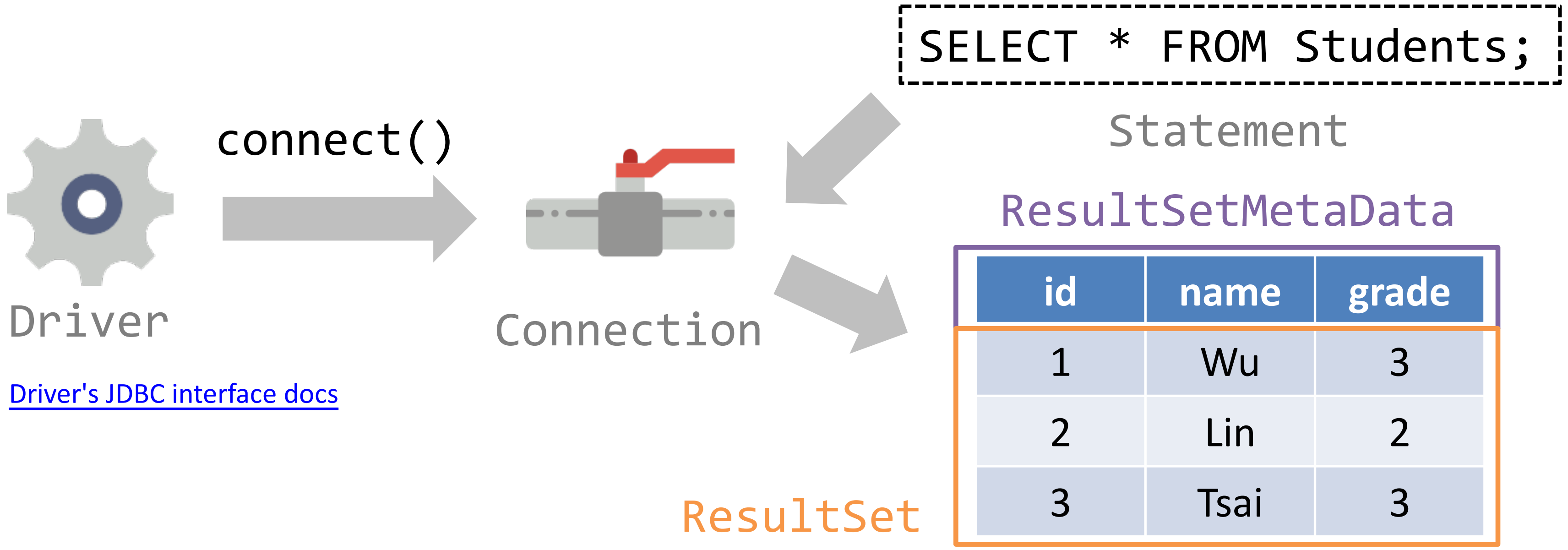
`remote` Package

JDBC
Package

Stored Procedure
Package

JDBC

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



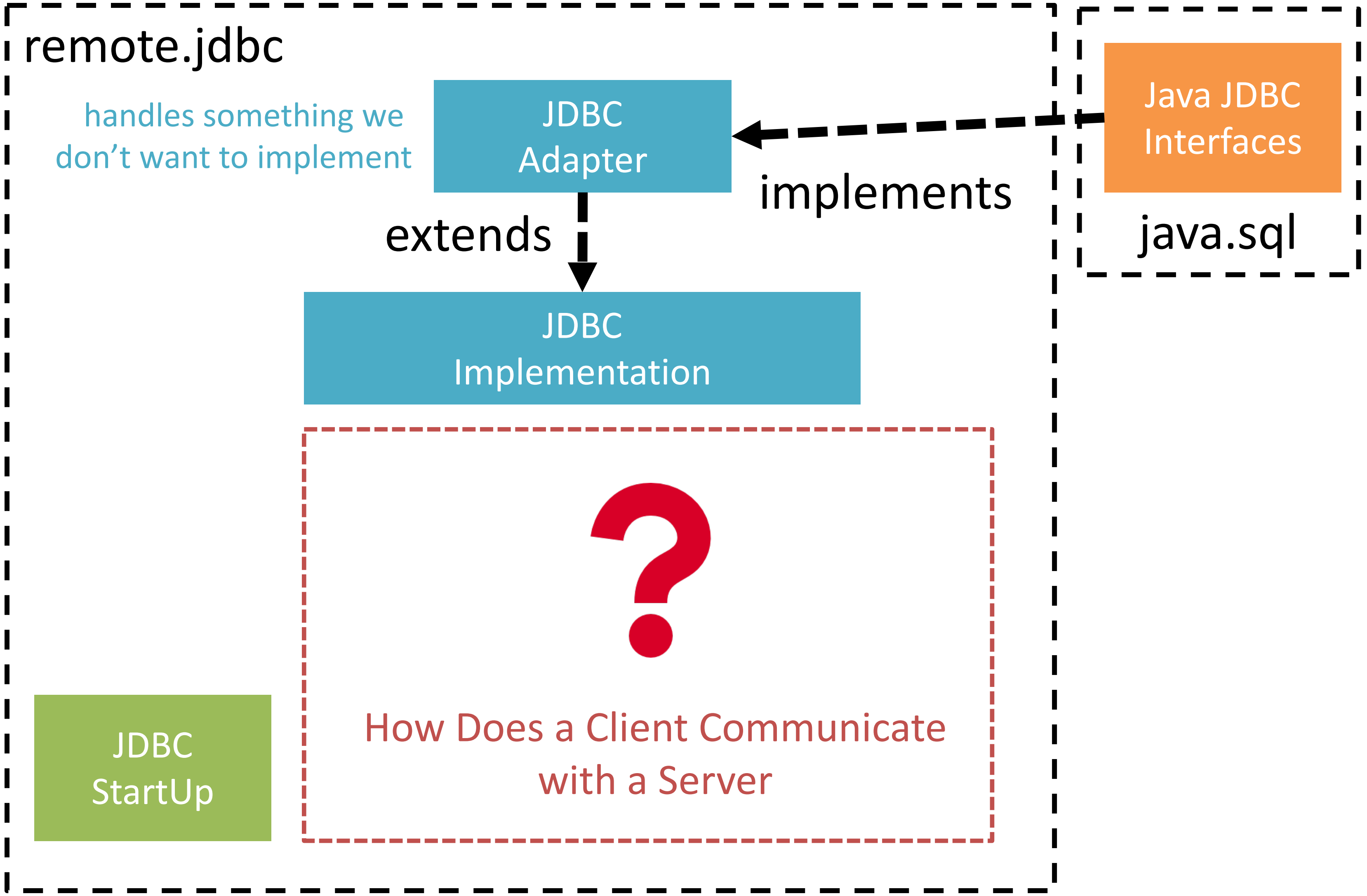
id	name	grade
1	Wu	3
2	Lin	2
3	Tsai	3

Driver's JDBC interface docs

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

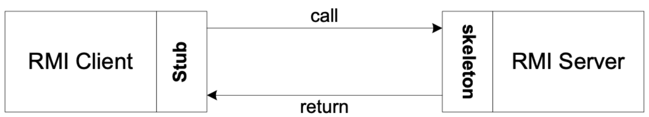
```
public class Server {
    public int[] sort(int[] numbers) {
        int[] array = Arrays.copyOf(numbers, numbers.length);
        Arrays.sort(array);
        return array;
    }
}
```

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

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

Preview / Review

The Stub and Skeleton

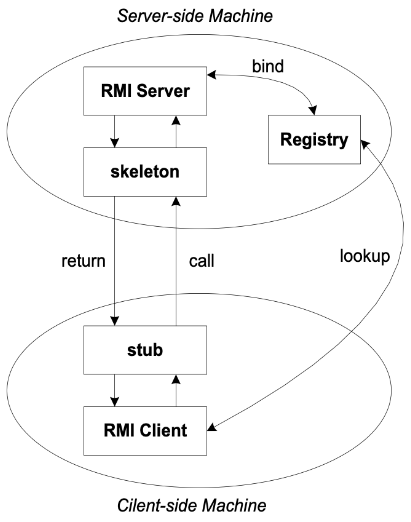


1. The ***skeleton*** (run by a server thread) binds the interface of the remote object
2. A client thread looks up and obtain a ***stub*** of the skeleton
3. When a client thread invokes a method, it is blocked and the call is first forwarded to the stub
4. The stub marshals the parameters and sends the call to the skeleton through the network
5. The skeleton receives the call, unmarshals the parameters, allocates from pool a worker thread that runs the remote object's method on behalf of the client
6. When the method returns, the worker thread returns the result to skeleton and returns to pool
7. The skeleton marshals the results and send it to stub
8. The stub unmarshals the results and continues the client thread

42

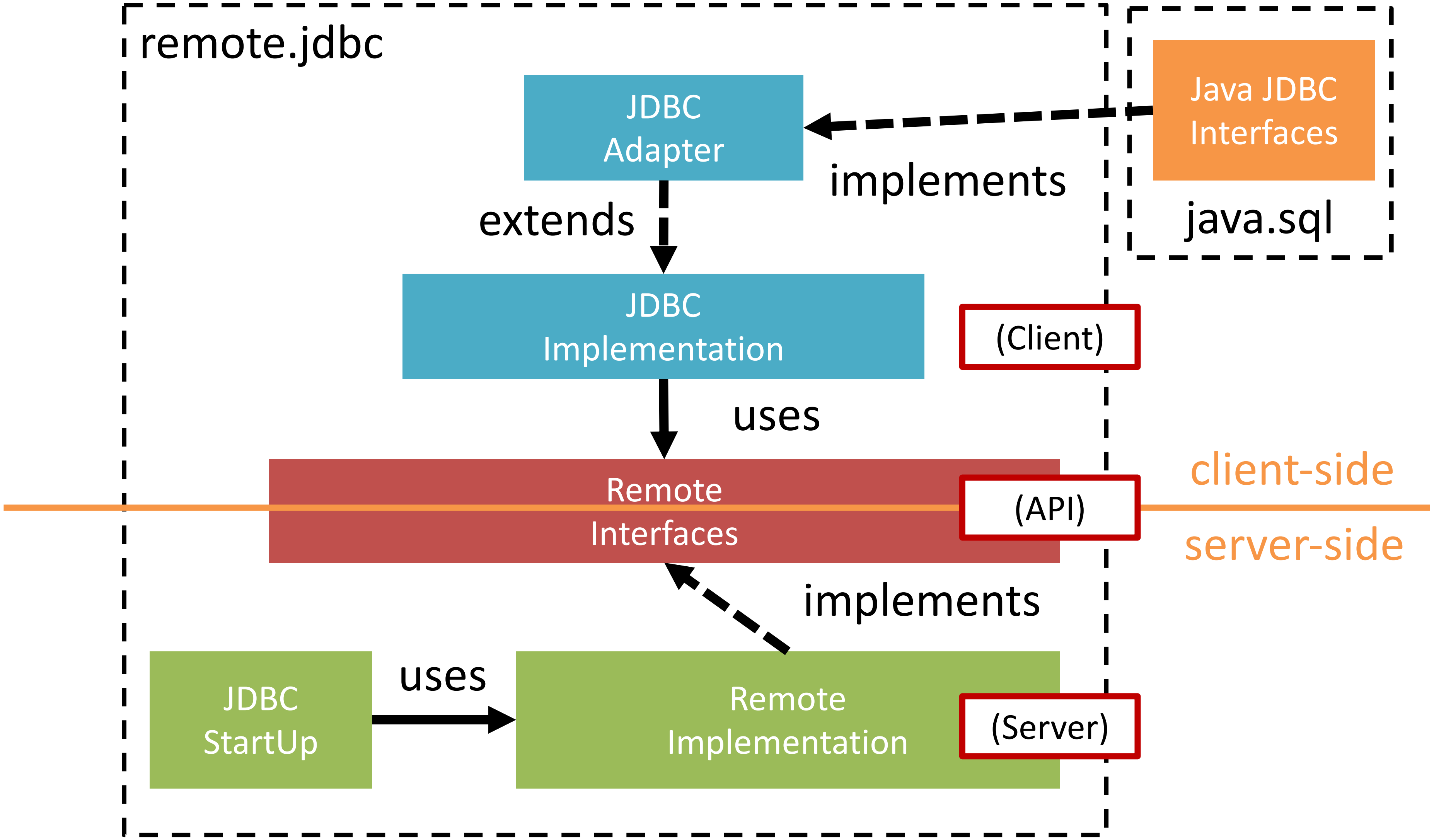
RMI registry

- The server must first bind the remote obj's interface to the registry with a name
  - The interface must extend the `java.rml.Remote` interface
- The client lookup the name in the registry to obtain a stub



43

`remote.jdbc` Package

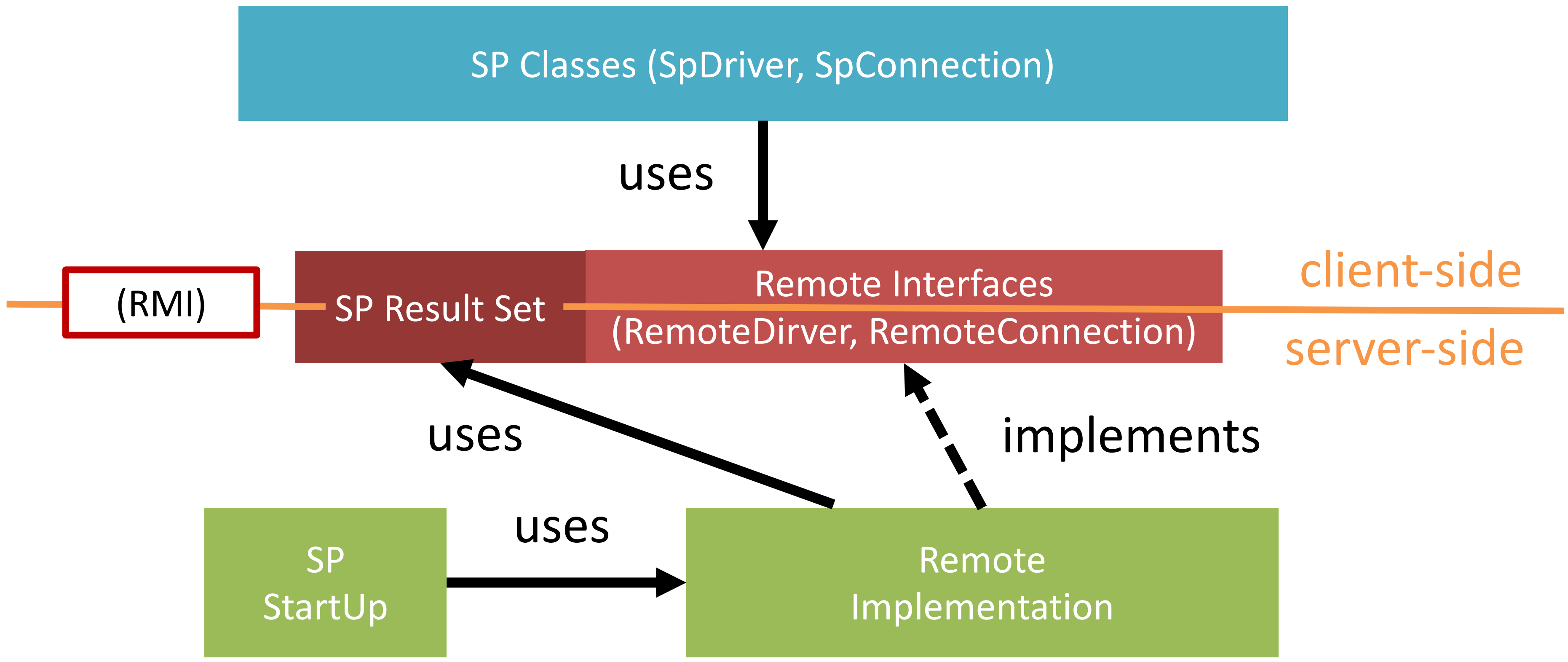


`remote` Package

JDBC Package

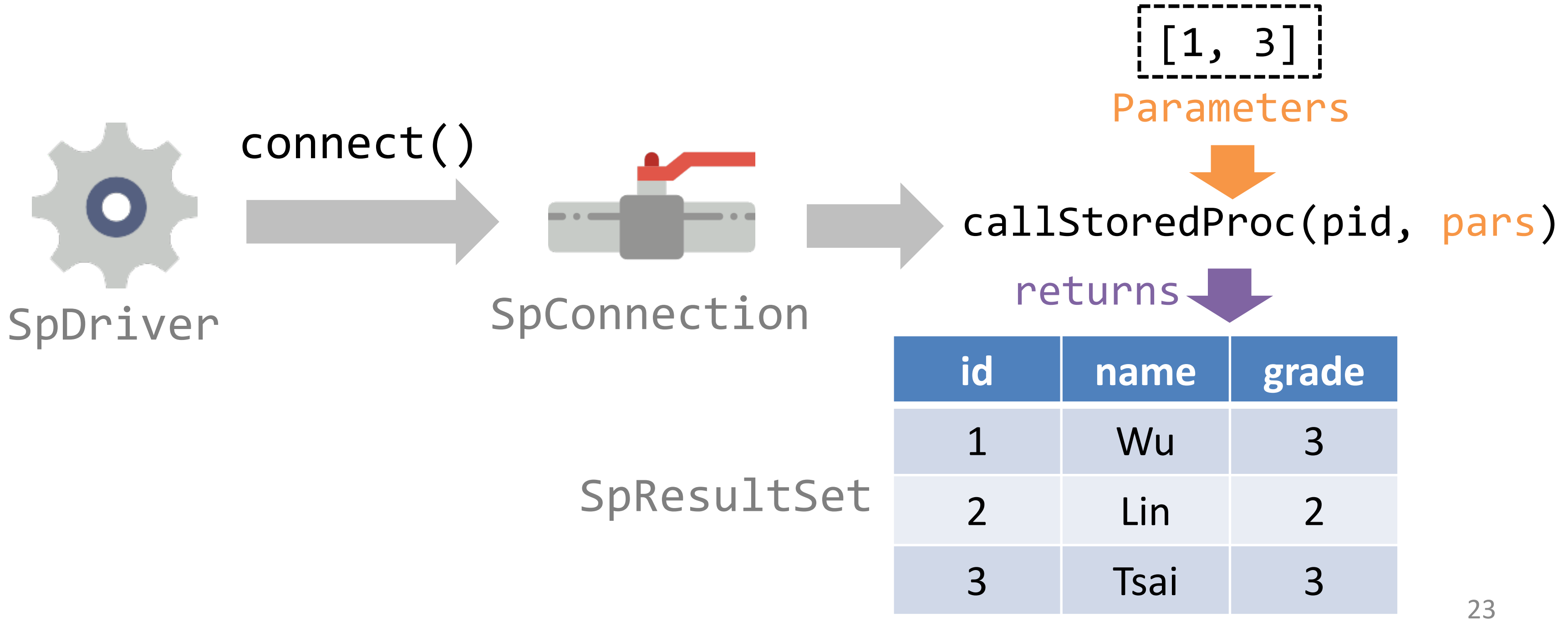
Stored Procedure Package

`remote.storedprocedure` Package



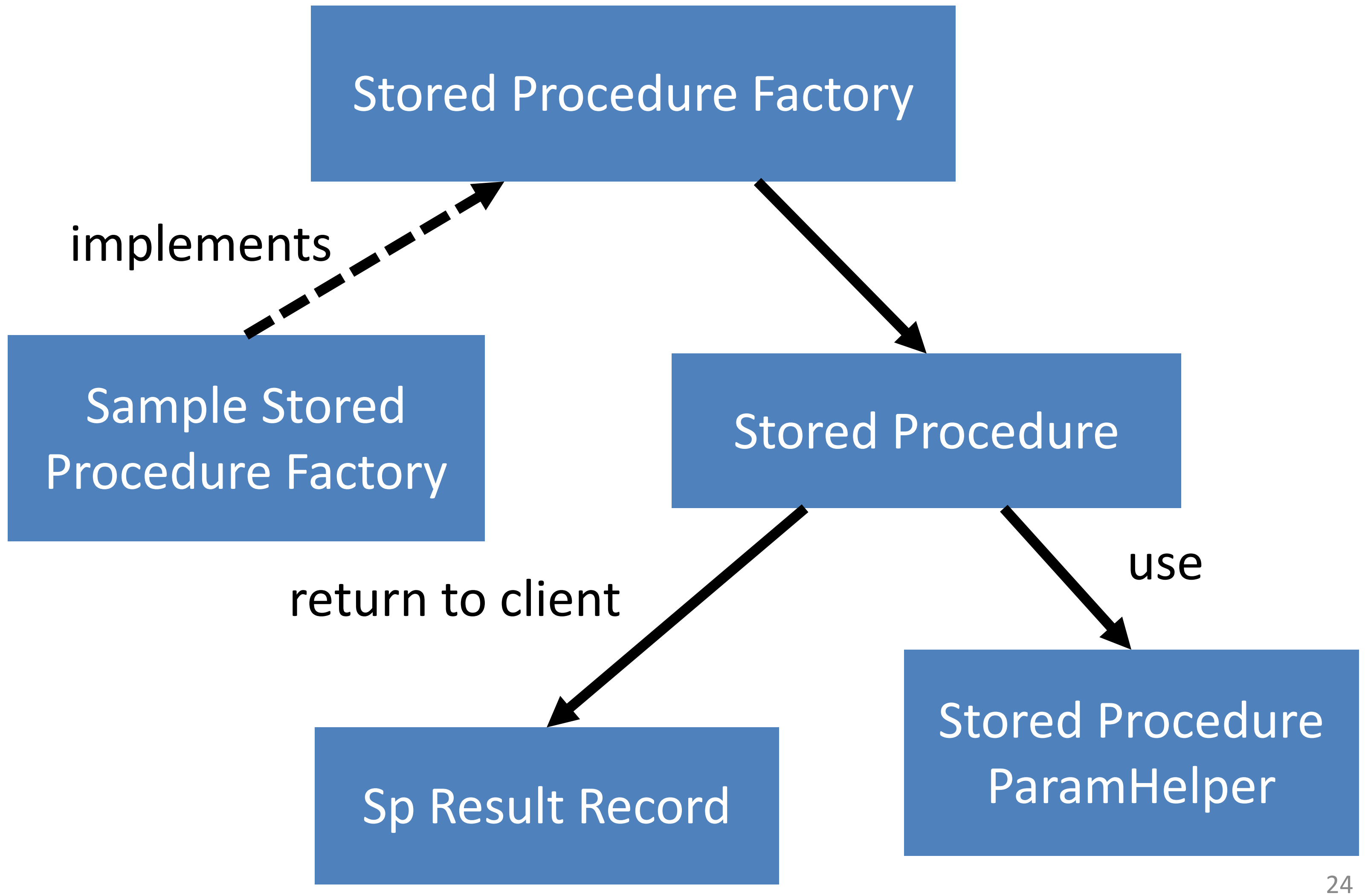
Calling Stored Procedure

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



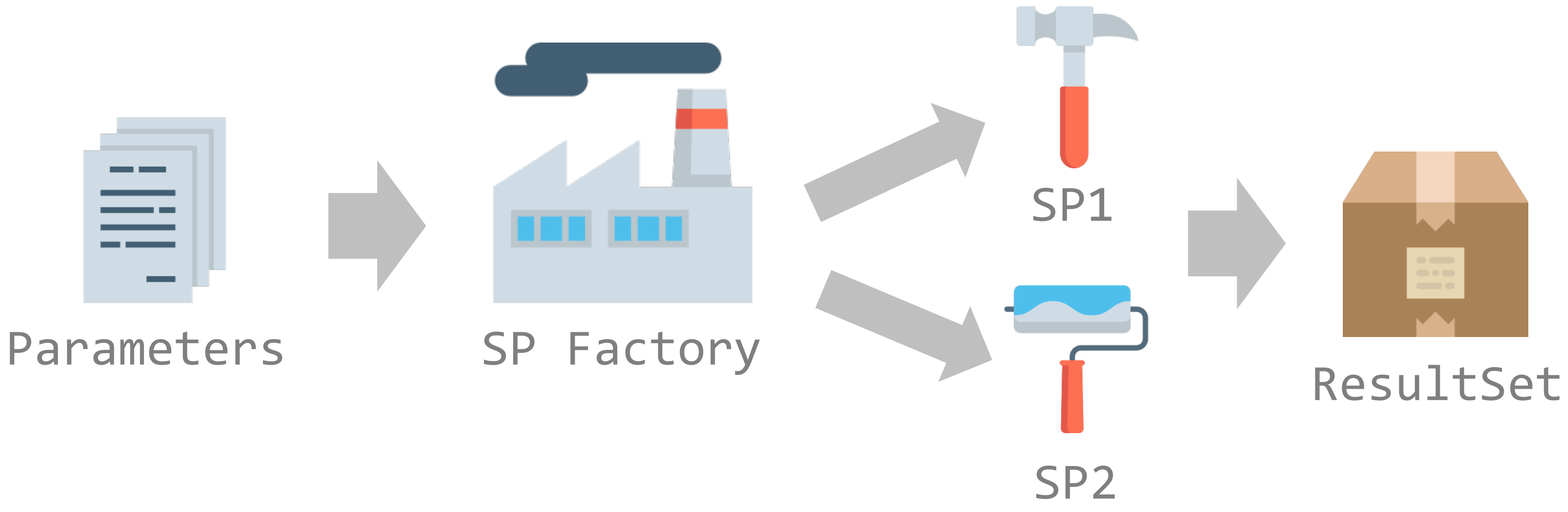
id	name	grade
1	Wu	3
2	Lin	2
3	Tsai	3

`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.


Parameters
SP Factory
SP1
SP2
ResultSet