VanillaCore Walkthrough Part 1

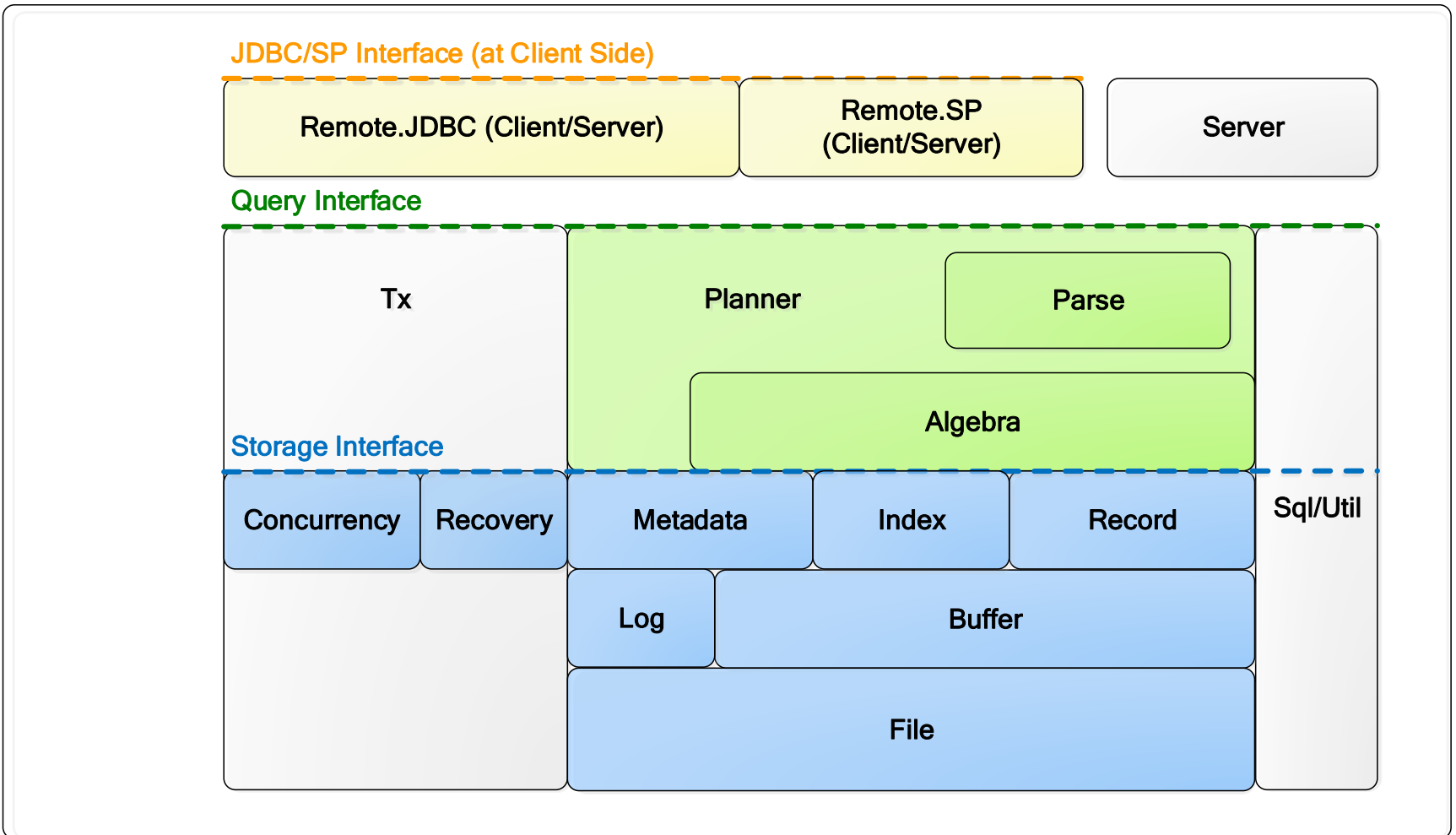## Introduction to Database Systems

DataLab

CS, NTHU
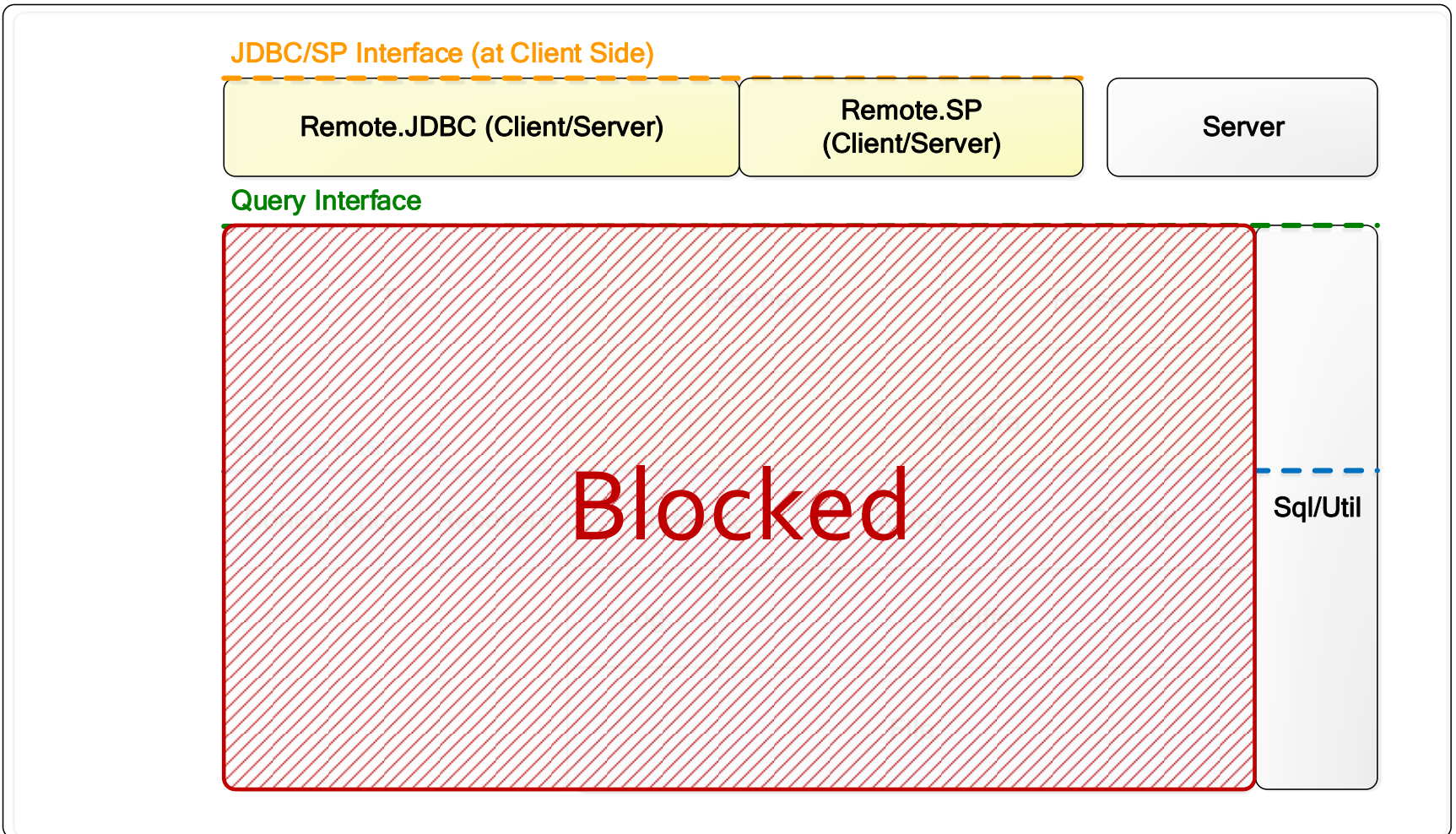
The Architecture

VanillaDB

| Remote.JDBC (Client/Server) | Remote.SP (Client/Server) | Server |

Query Interface

| Tx | Planner | Parse |

Algebra

Storage Interface

| Concurrency | Recovery | Metadata | Index | Record | Sql/Util |

| Log | Buffer |

| File |

The Architecture

VanillaDB

JDBC/SP Interface (at Client Side)

| Remote.JDBC (Client/Server) | Remote.SP (Client/Server) | Server |

Query Interface

Blocked

Sql/Util

200 200 100 200 4/5

Fog

- Now, you can only see a part of VanillaCore.

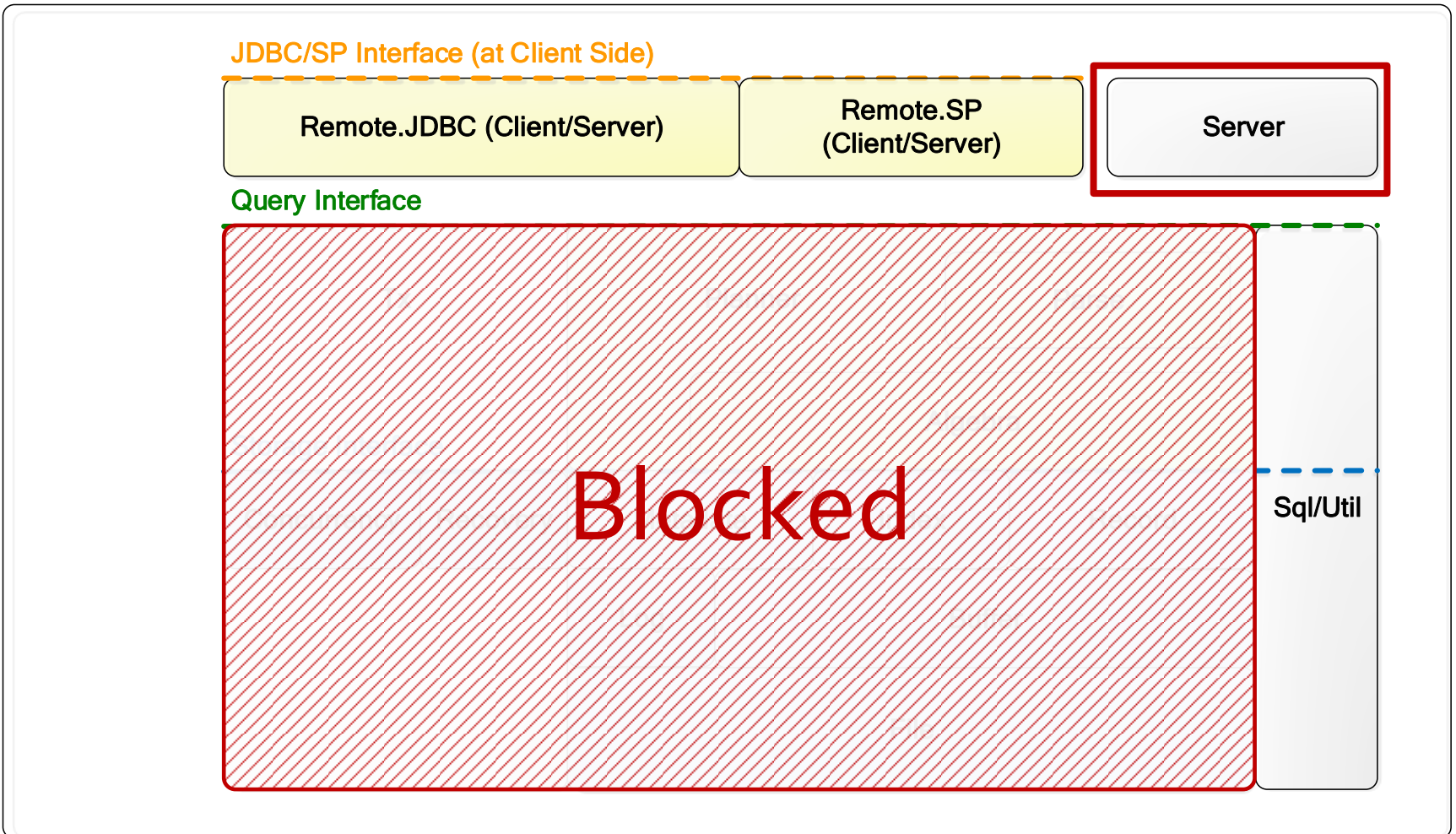- As the progress of the course, we will open more packages in the future.

Outline

- Server package
- Remote package
- SQL package

Outline

- **Server package**
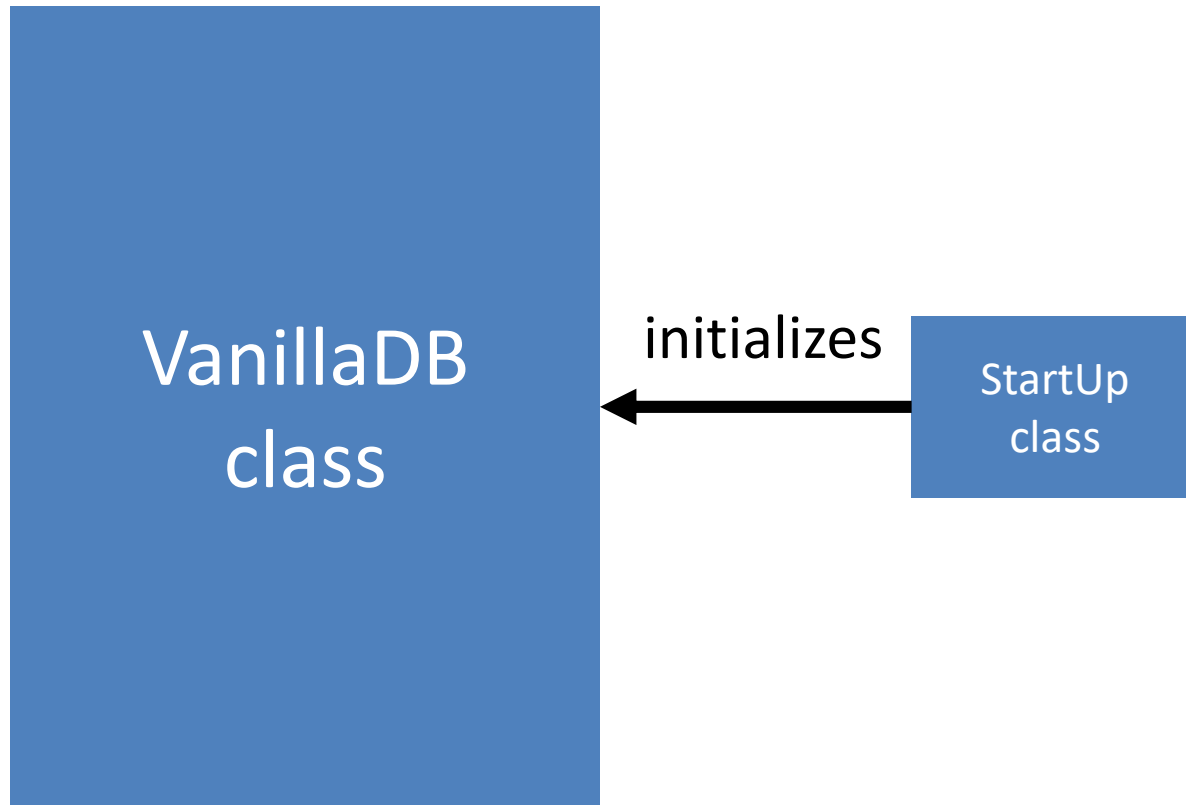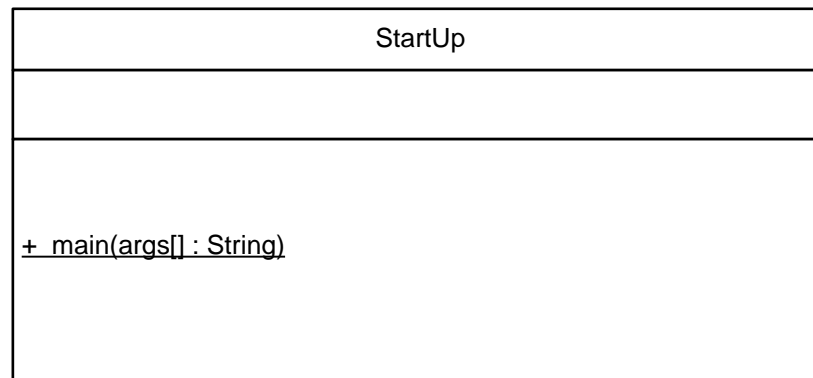- Remote package
- SQL package

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

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()

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 connction

StartUp
+ main(args[] : String)

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
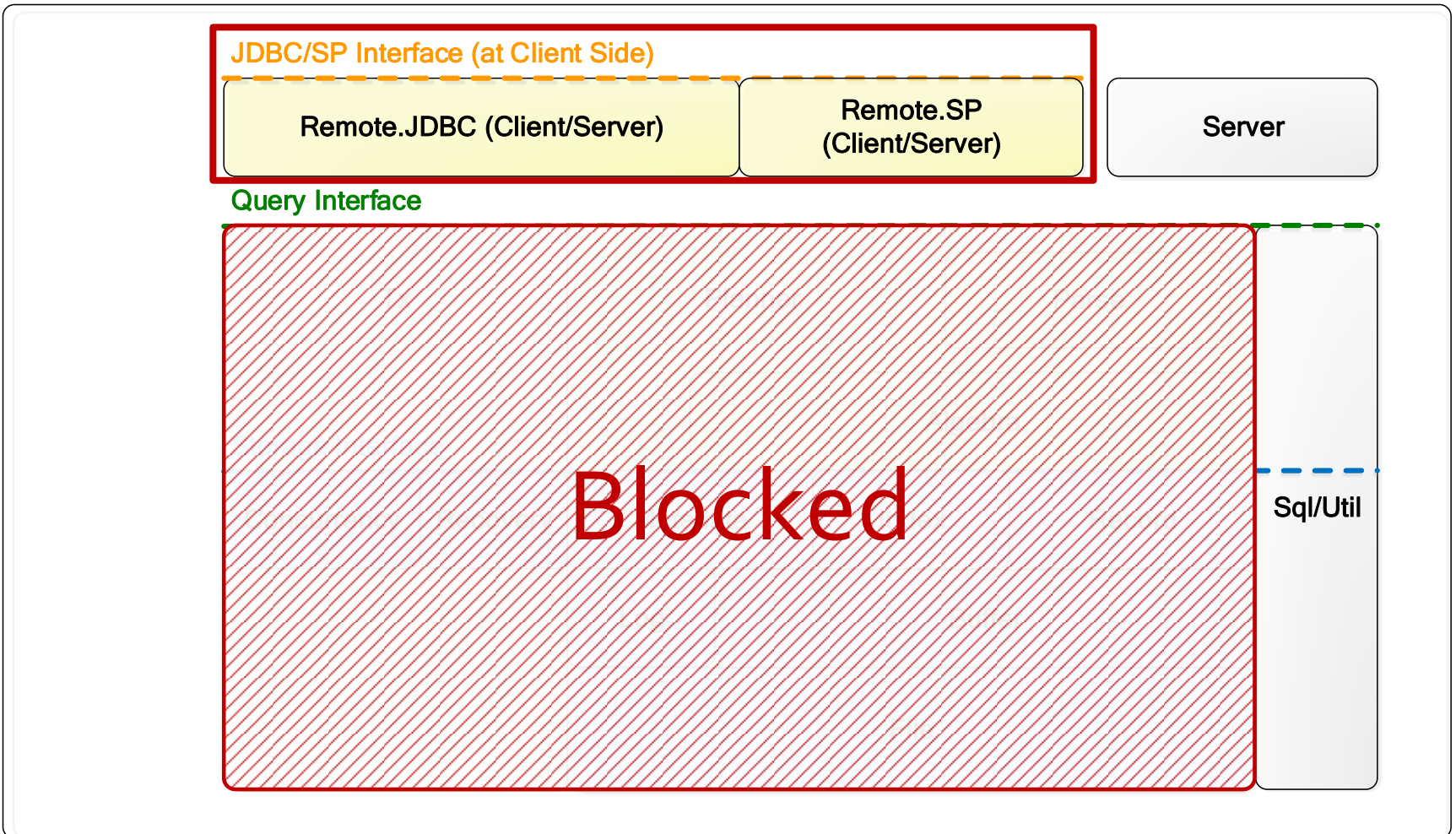
`remote` **Package**

JDBC
Package

Stored Procedure
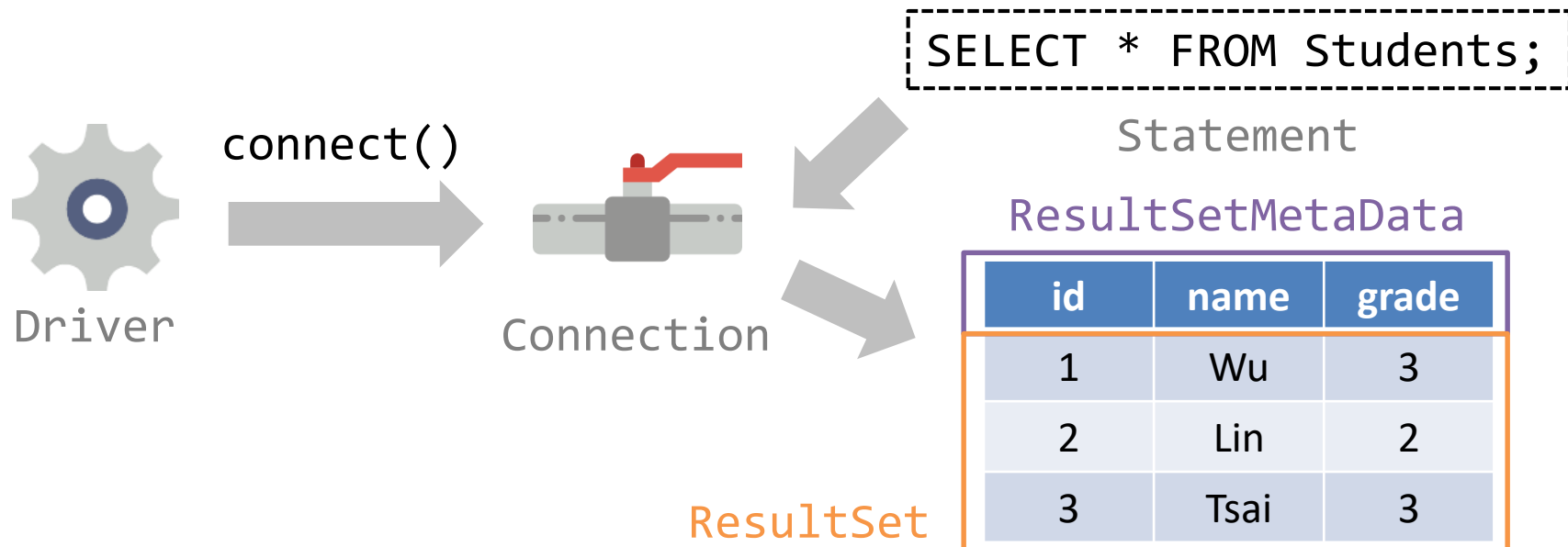Package

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.
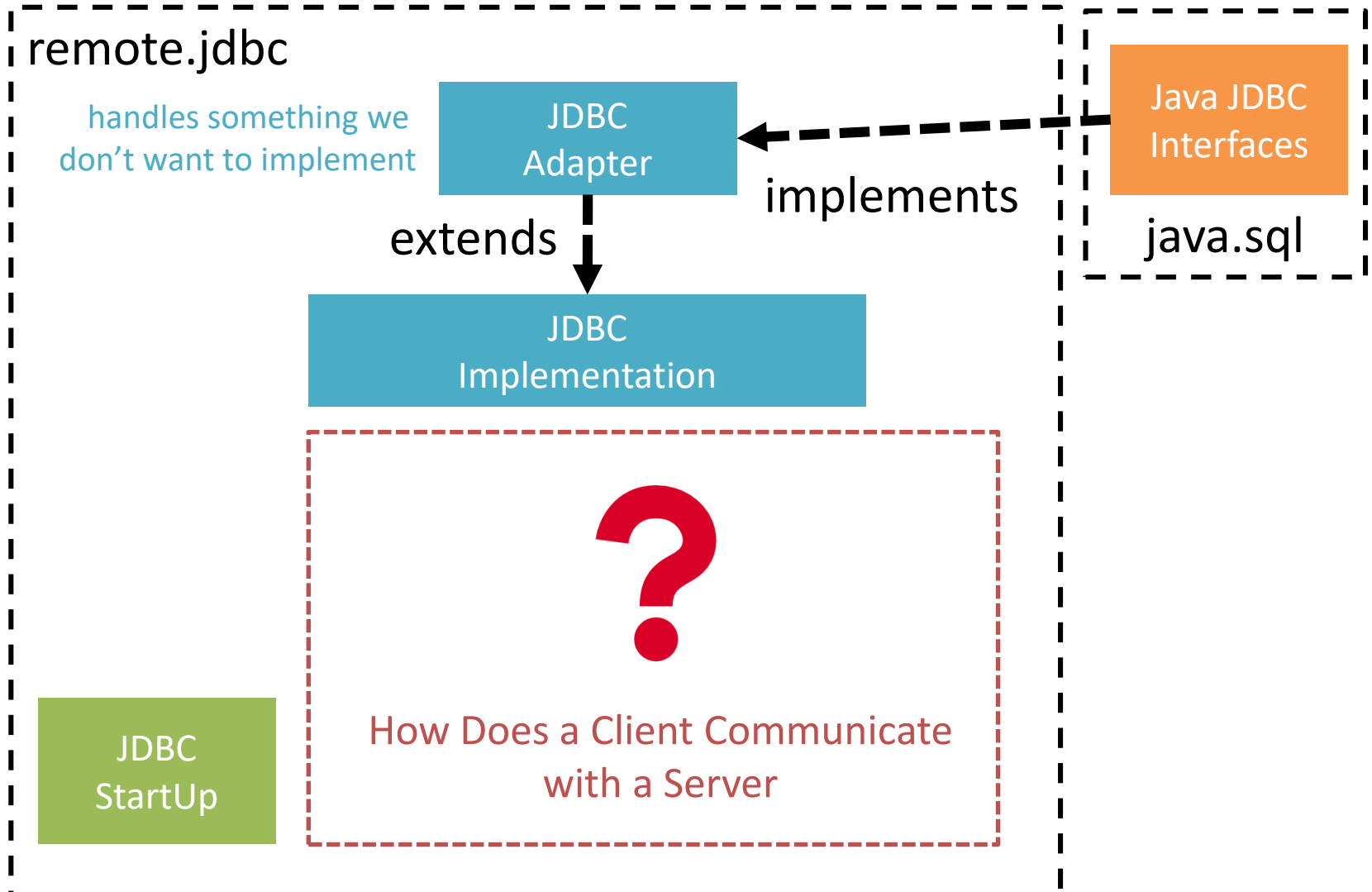
```
SELECT * FROM Students;
```

Statement

connect()

ResultSetMetaData

Driver

Connection

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

ResultSet

JDBC Program: Finding Major

```java
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

remote.jdbc

handles something we don't want to implement

JDBC Adapter

Java JDBC Interfaces

java.sql

implements

extends

JDBC Implementation

?

How Does a Client Communicate with a Server

JDBC StartUp

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.
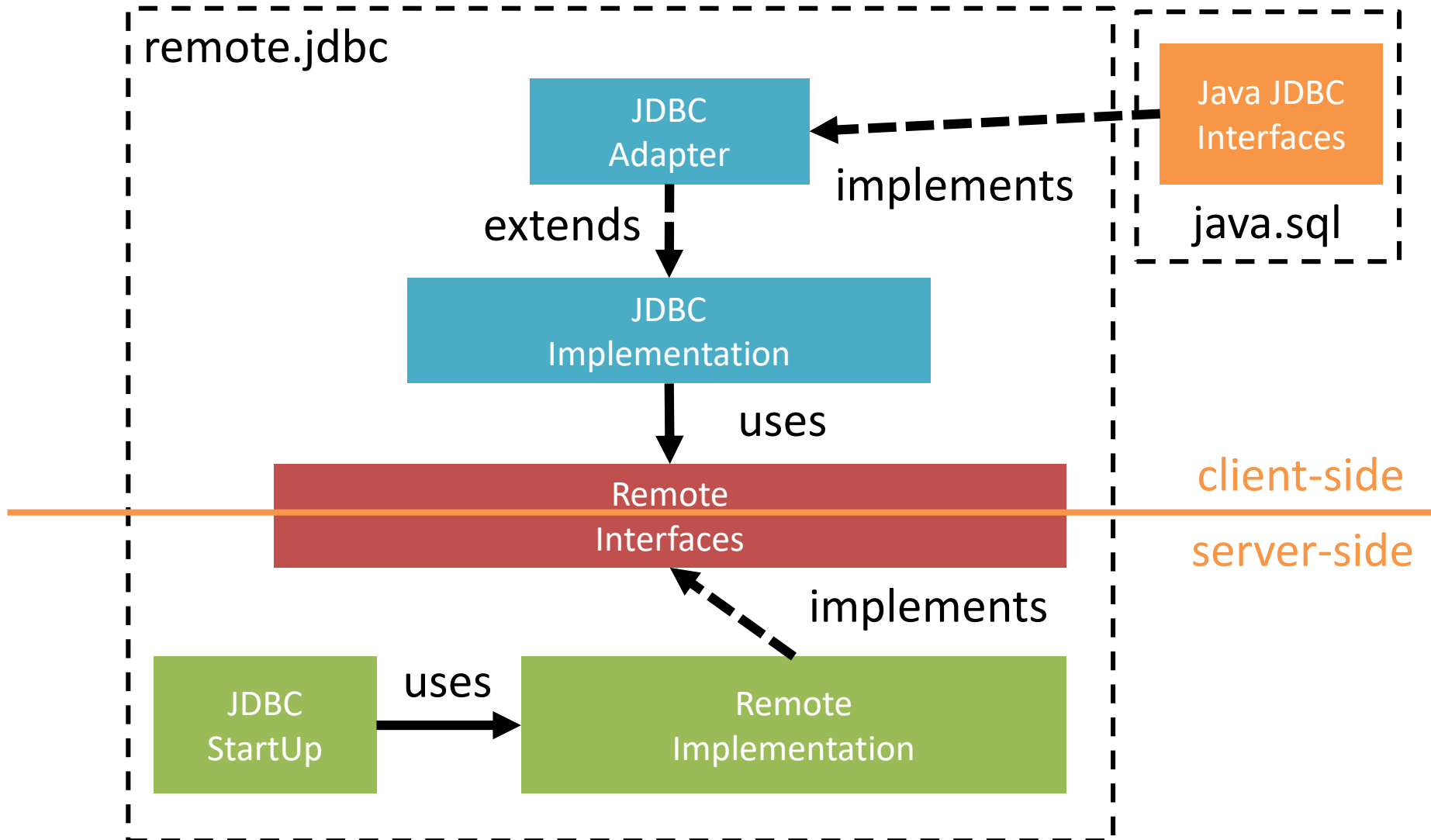
RMI Example

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

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

```java
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.jdbc

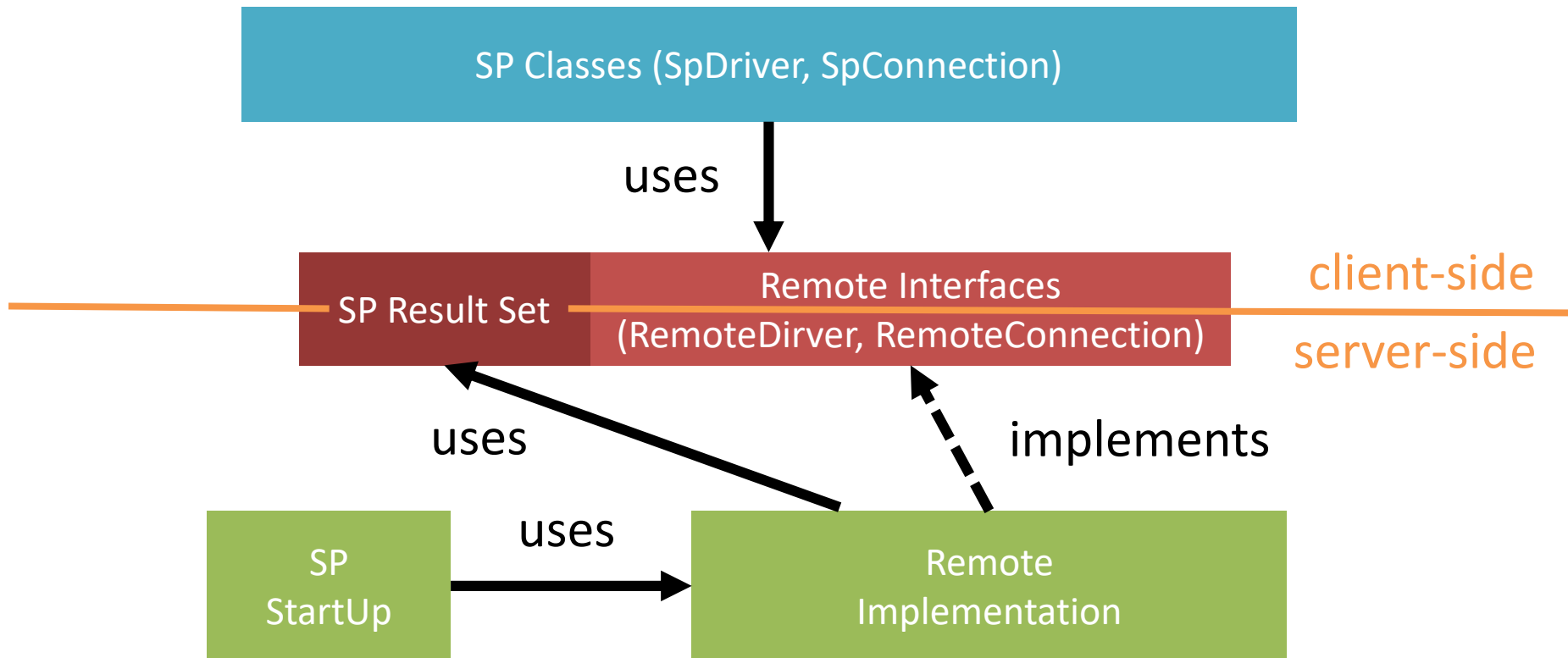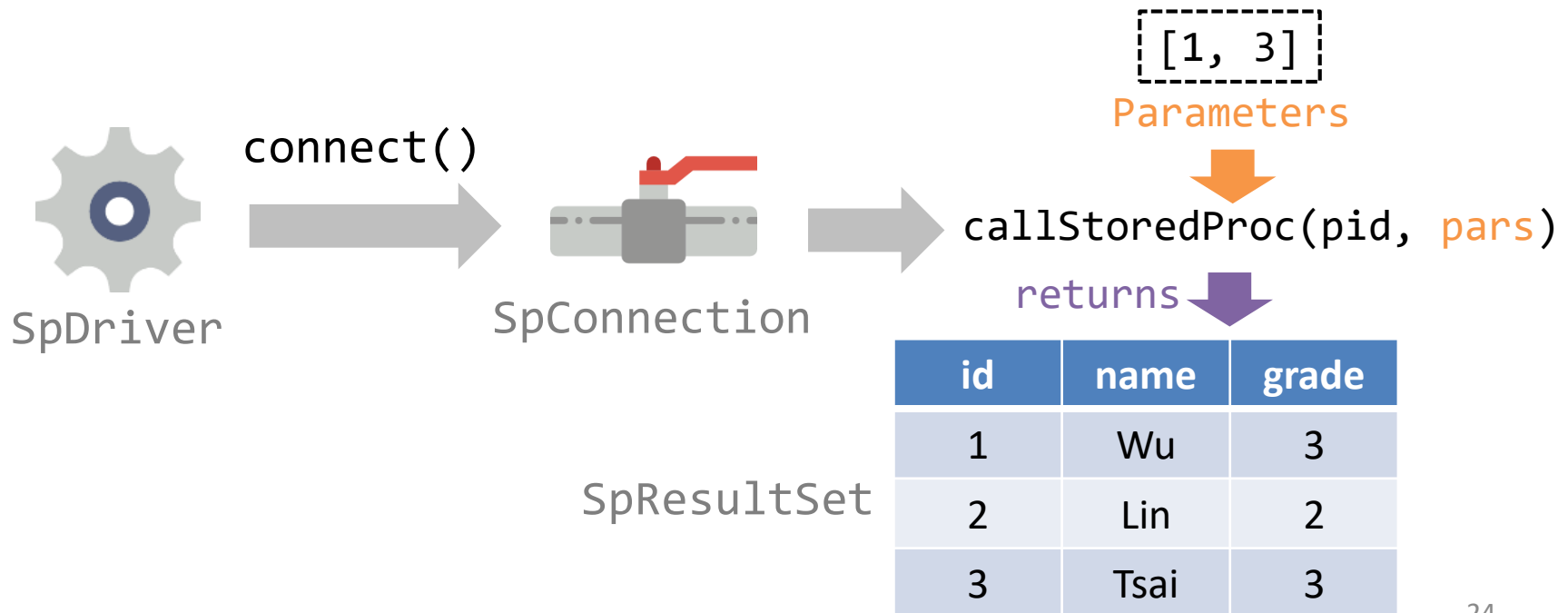JDBC Adapter

implements

Java JDBC Interfaces

java.sql

extends

JDBC Implementation

uses

Remote Interfaces

client-side

server-side

implements

JDBC StartUp

uses

Remote Implementation

remote Package

JDBC
Package

Stored Procedure
Package

remote.storedprocedure Package

SP Classes (SpDriver, SpConnection)

uses

SP Result Set | Remote Interfaces (RemoteDirver, RemoteConnection)

client-side

server-side

uses

implements

SP StartUp

uses

Remote Implementation

Calling Stored Procedure

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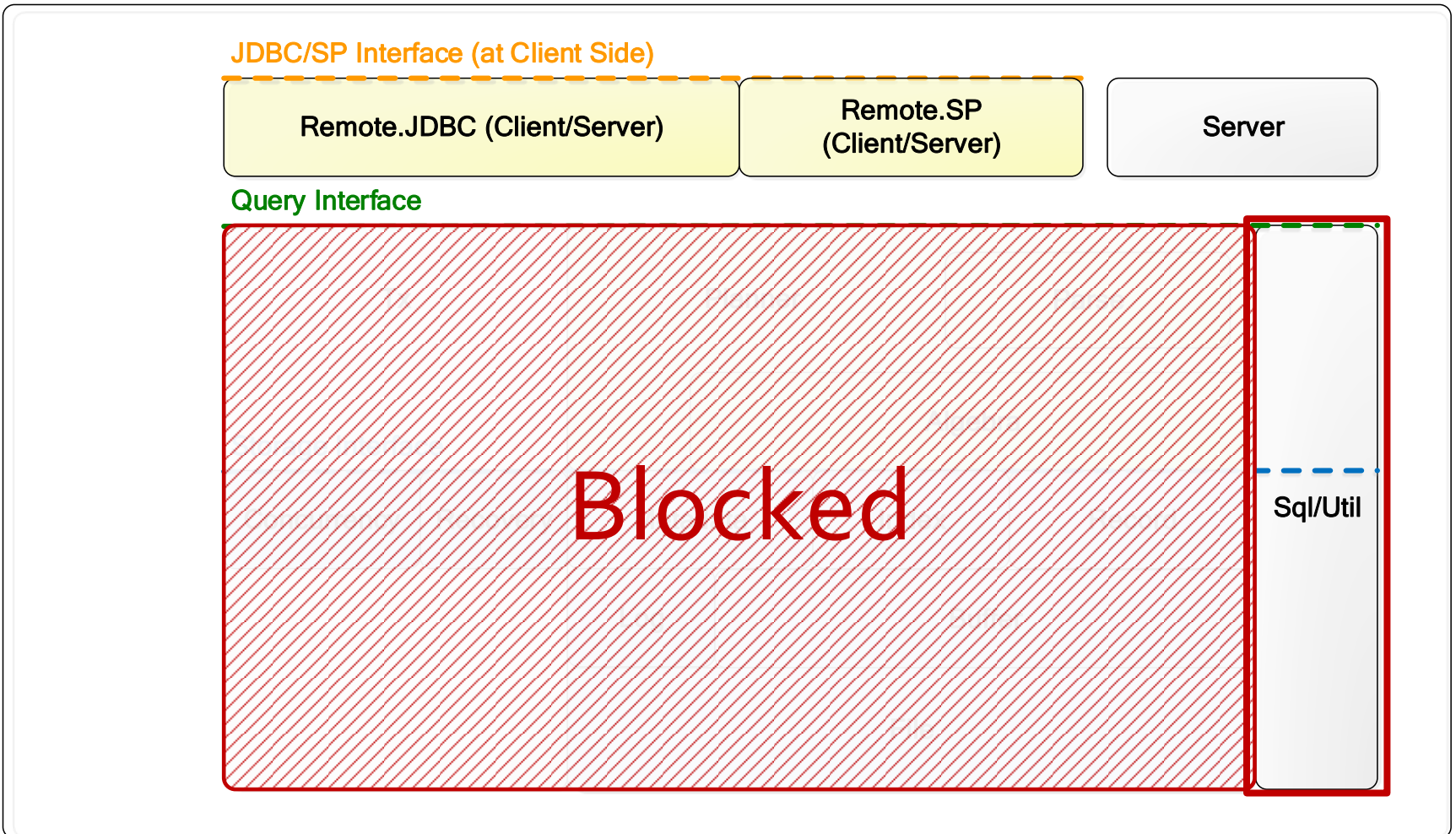 - Then send the parameters via the connection

`[1, 3]`

Parameters

`connect()`

`callStoredProc(pid, pars)`

`SpDriver`

`SpConnection`

returns

`SpResultSet`

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

blog_id	url	created	author_id
33981	ms.com/...	2012/10/31	729
33982	apache.org/...	2012/11/15	4412

← Schema

← Record

`sql` Package

Schema

Schema

Type
Classes

Aggregation
Function
Package

Record

Record

Constant
Classes

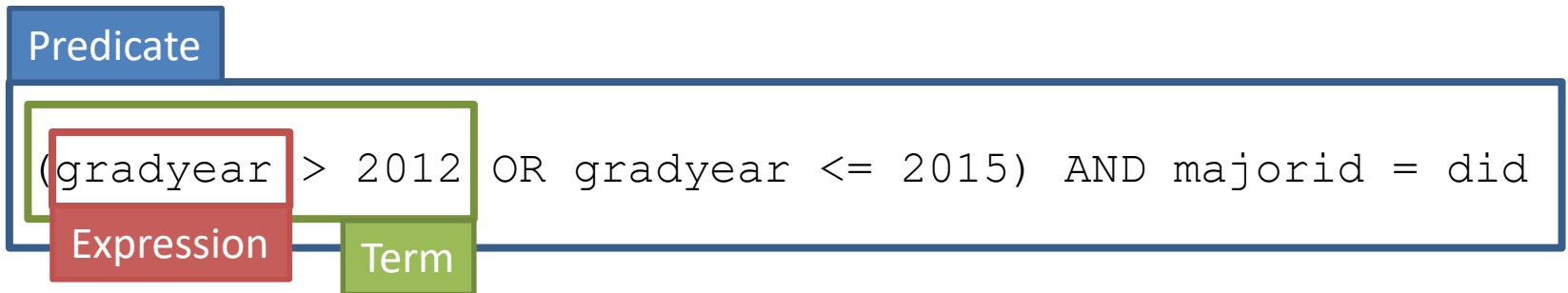Predicate
Package

Record
Comparator

Constant Range
Classes

Stored Procedure
Package
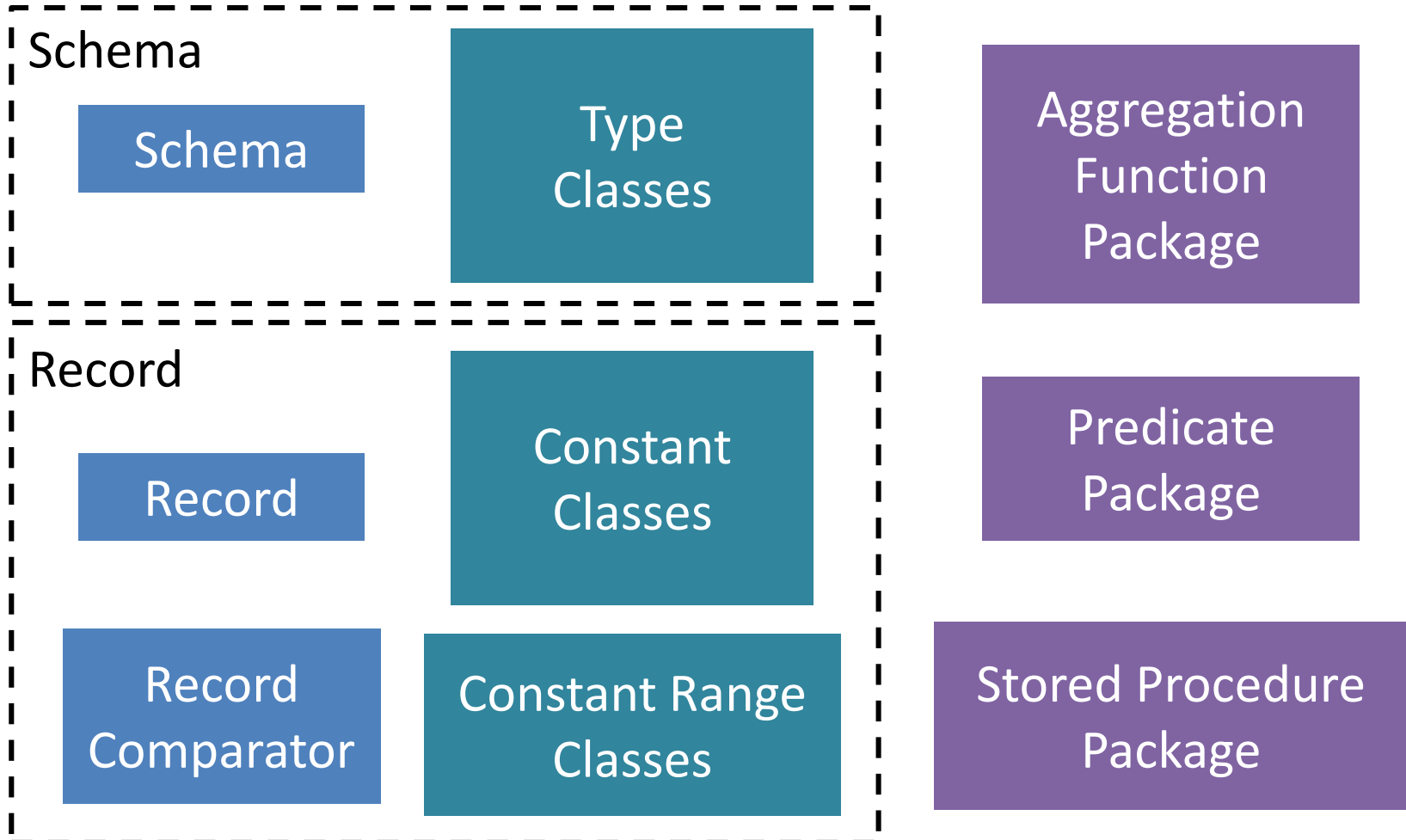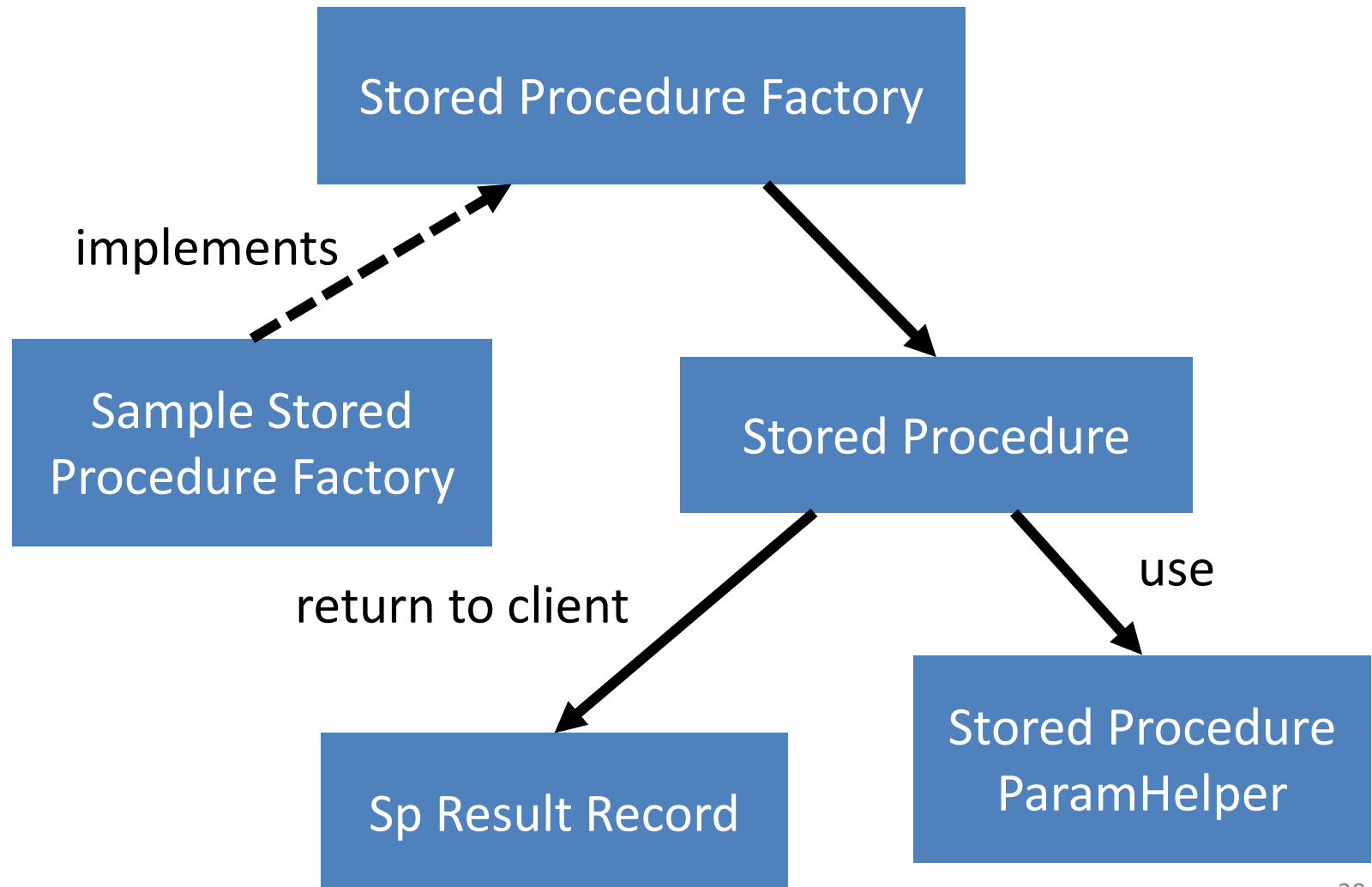
Predicates

- 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

Predicate

Expression

Term

`(gradyear > 2012 OR gradyear <= 2015) AND majorid = did`

`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