DB2 Application Development overview

IBM Information Management Cloud Computing Center of Competence
IBM Canada Lab

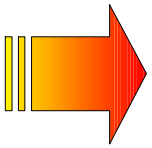
Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

Supporting reading material & videos

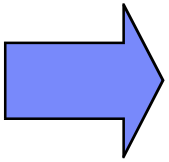
- **Reading materials**
  - Getting started with DB2 Express-C eBook
    - Chapter 14: Introduction to DB2 application development
  - Getting started with IBM Data Studio for DB2 eBook
    - Chapter 5: Developing SQL Stored Procedures
    - Chapter 7: Developing user-defined functions
  - Getting started with DB2 Application Development eBook
    - Chapter 3, section 3.6: Triggers: The big picture
- **Videos**
  - db2university.com course AA001EN
    - Lesson 12: DB2 application development

Agenda



- **DB2 Application Development overview**
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

DB2 Application Development Overview

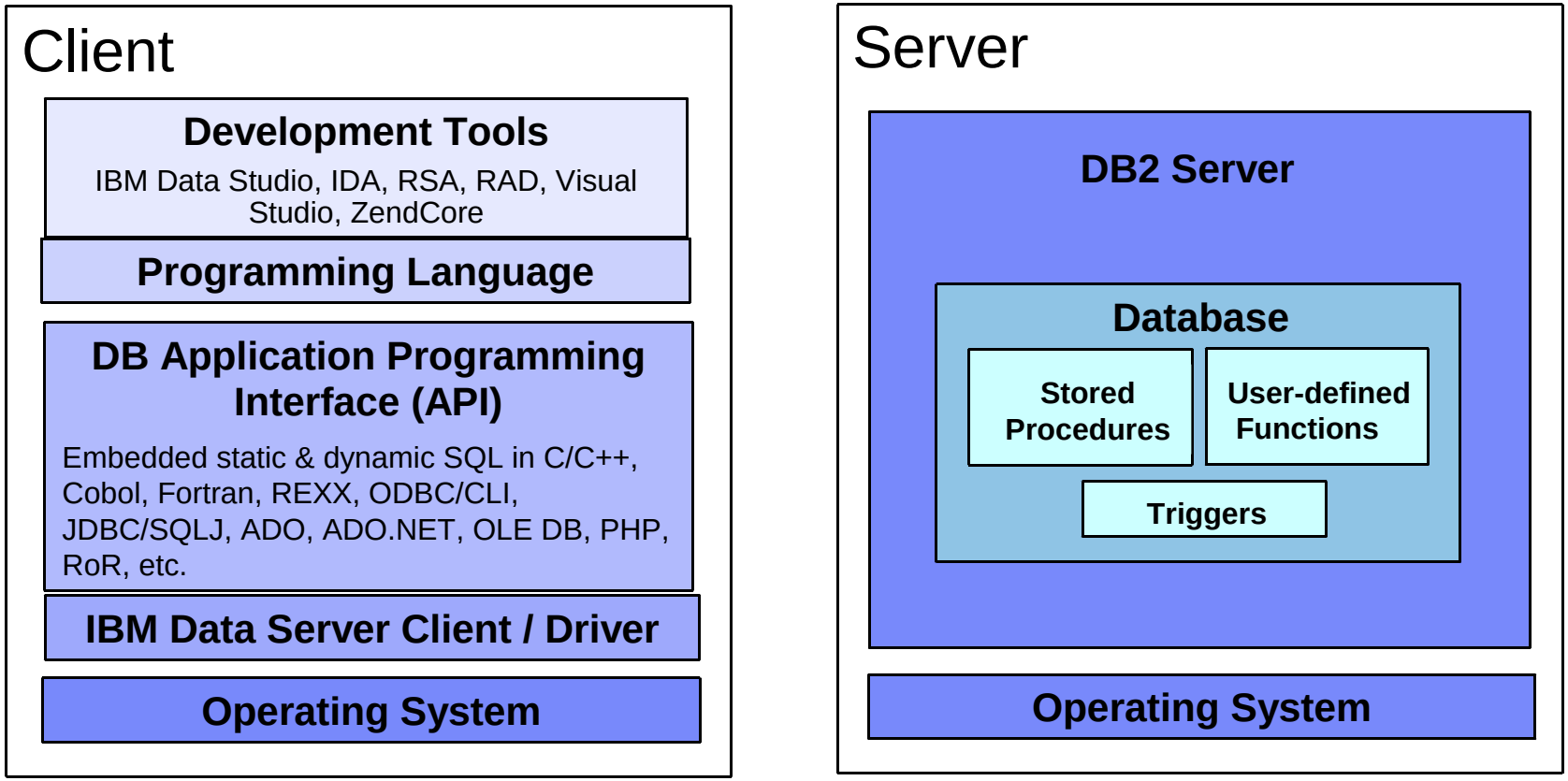


Server-side development (at the DB2 database server):

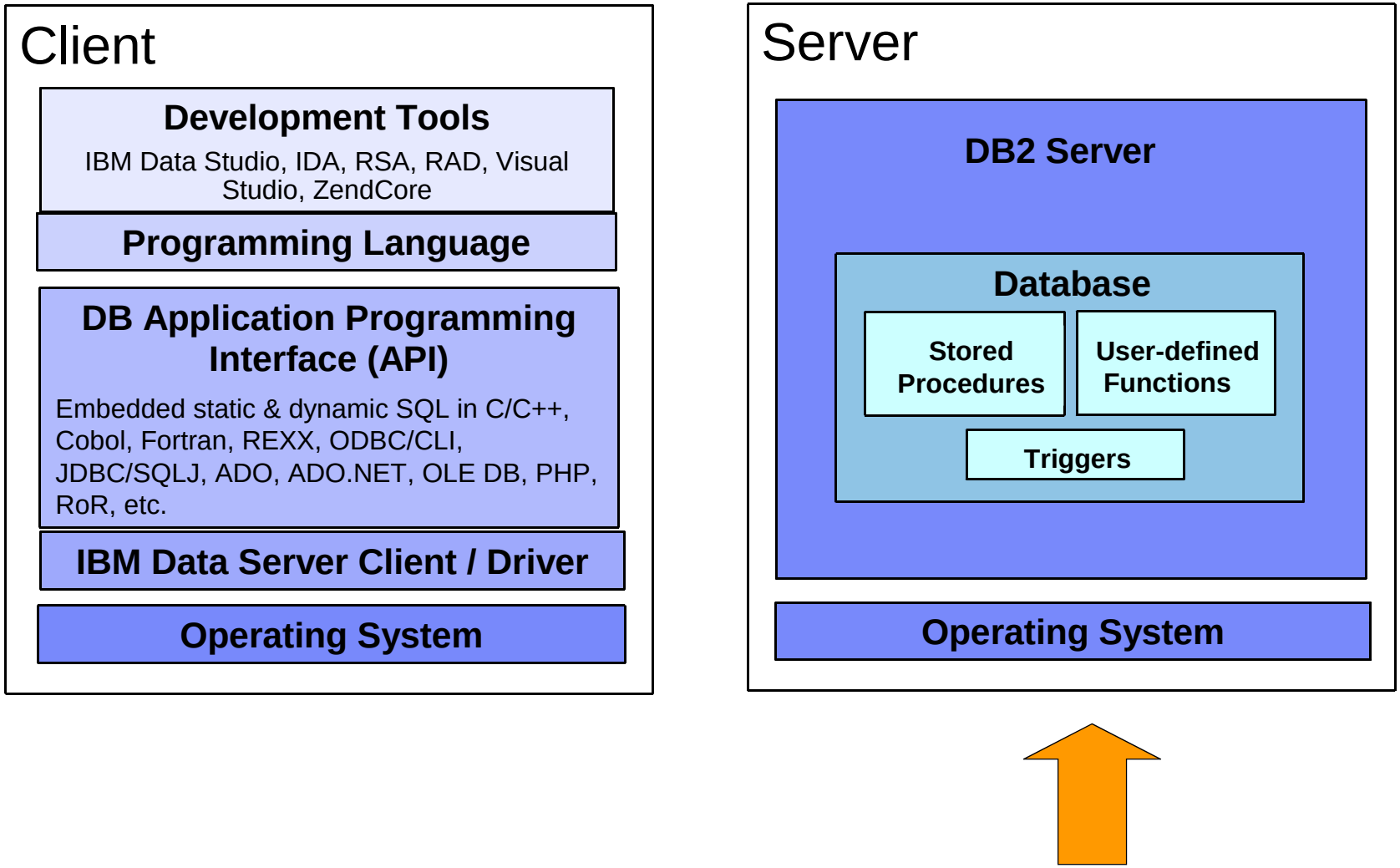
- Routines (Stored Procedures, UDFs)
- Database objects (Triggers)

- **Client-side development (at the client):**
  - May require a DB2 client or driver to be installed
  - Database applications (in C/C++, .NET, Cobol, Java, etc)

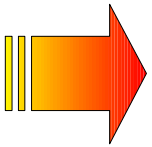
DB2 Application Development Overview



DB2 Application Development Overview

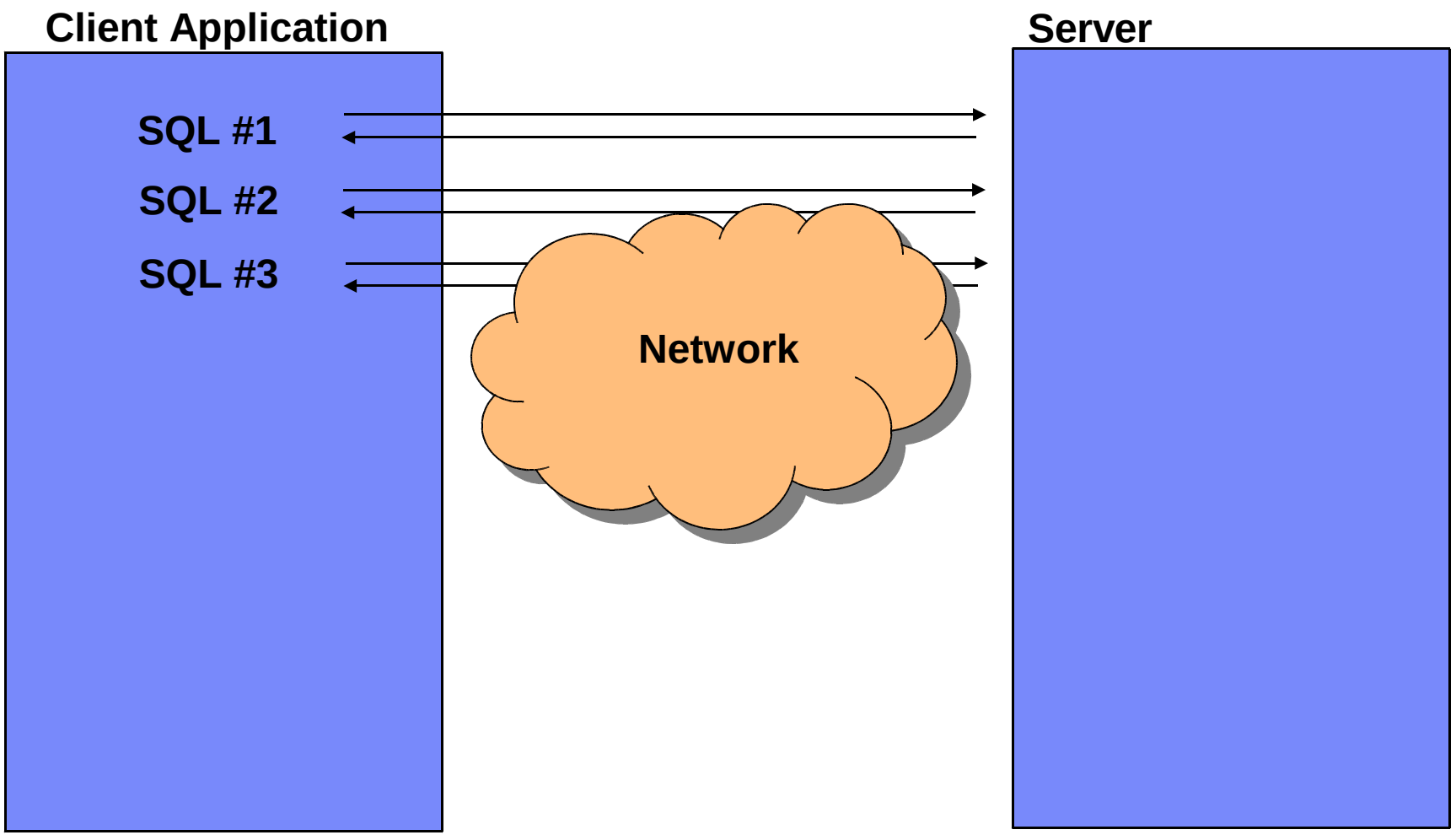


Agenda

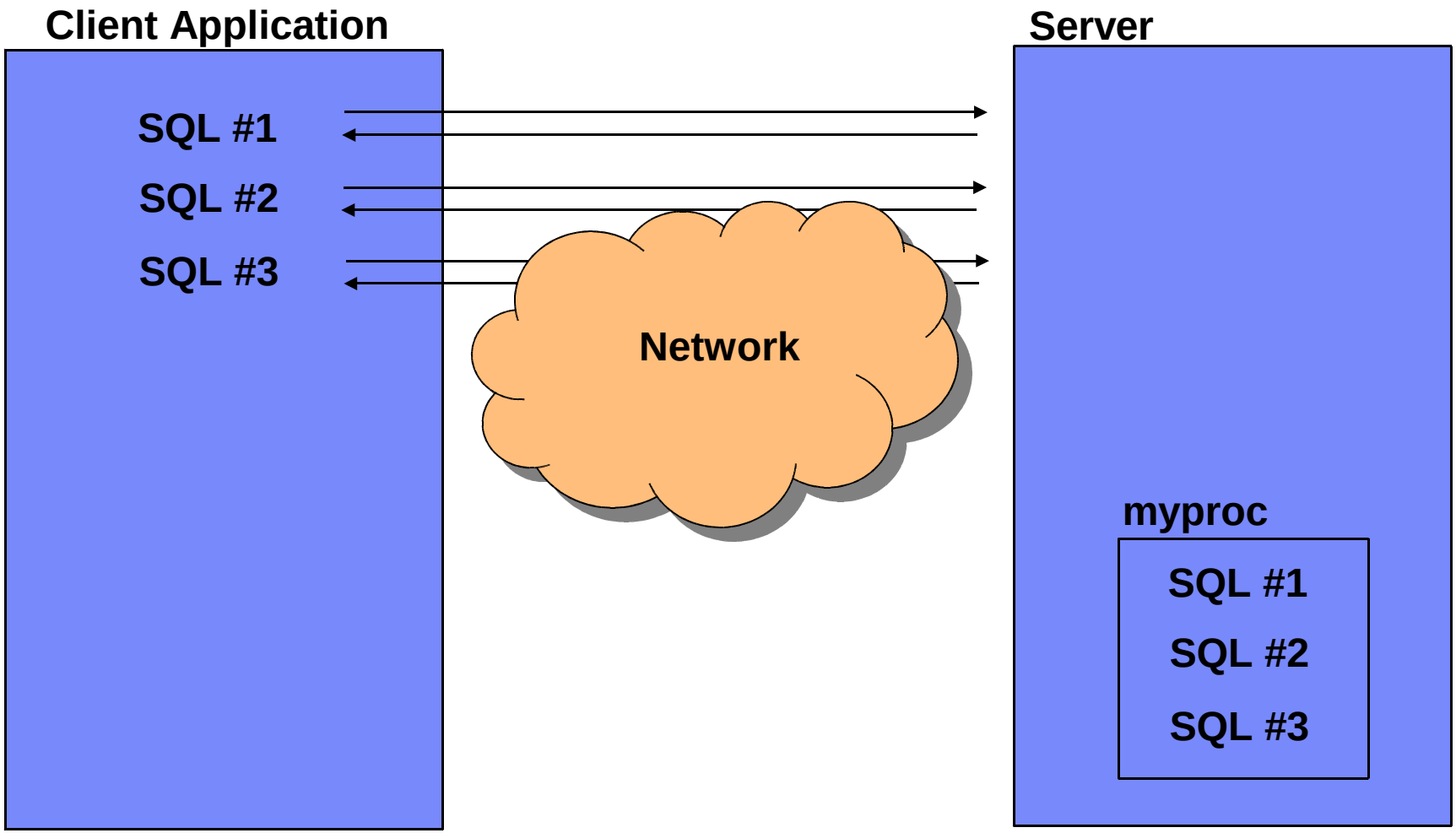


- DB2 Application Development overview
- **Server-side development**
  - **Stored Procedures**
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

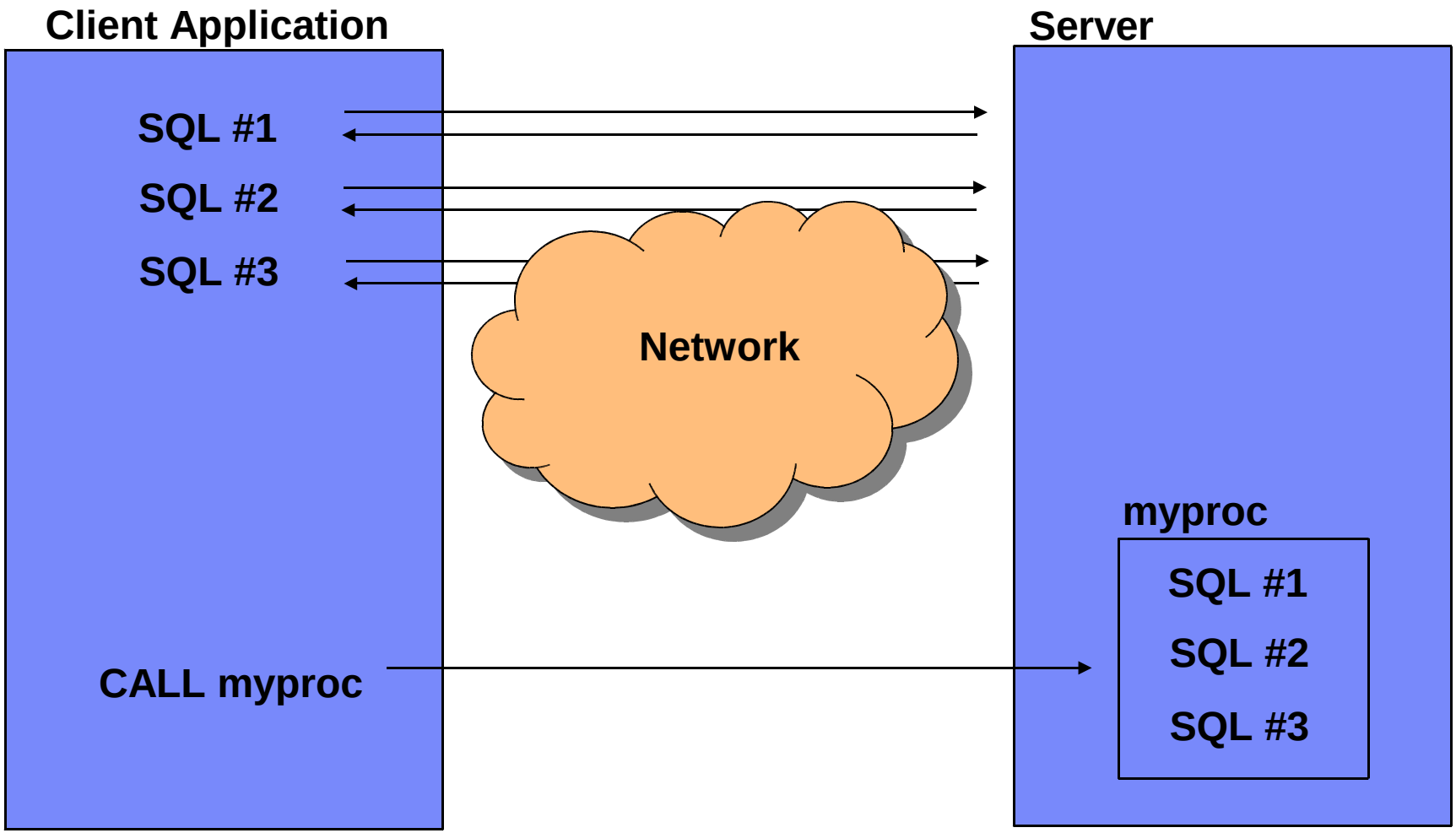
Stored procedures overview



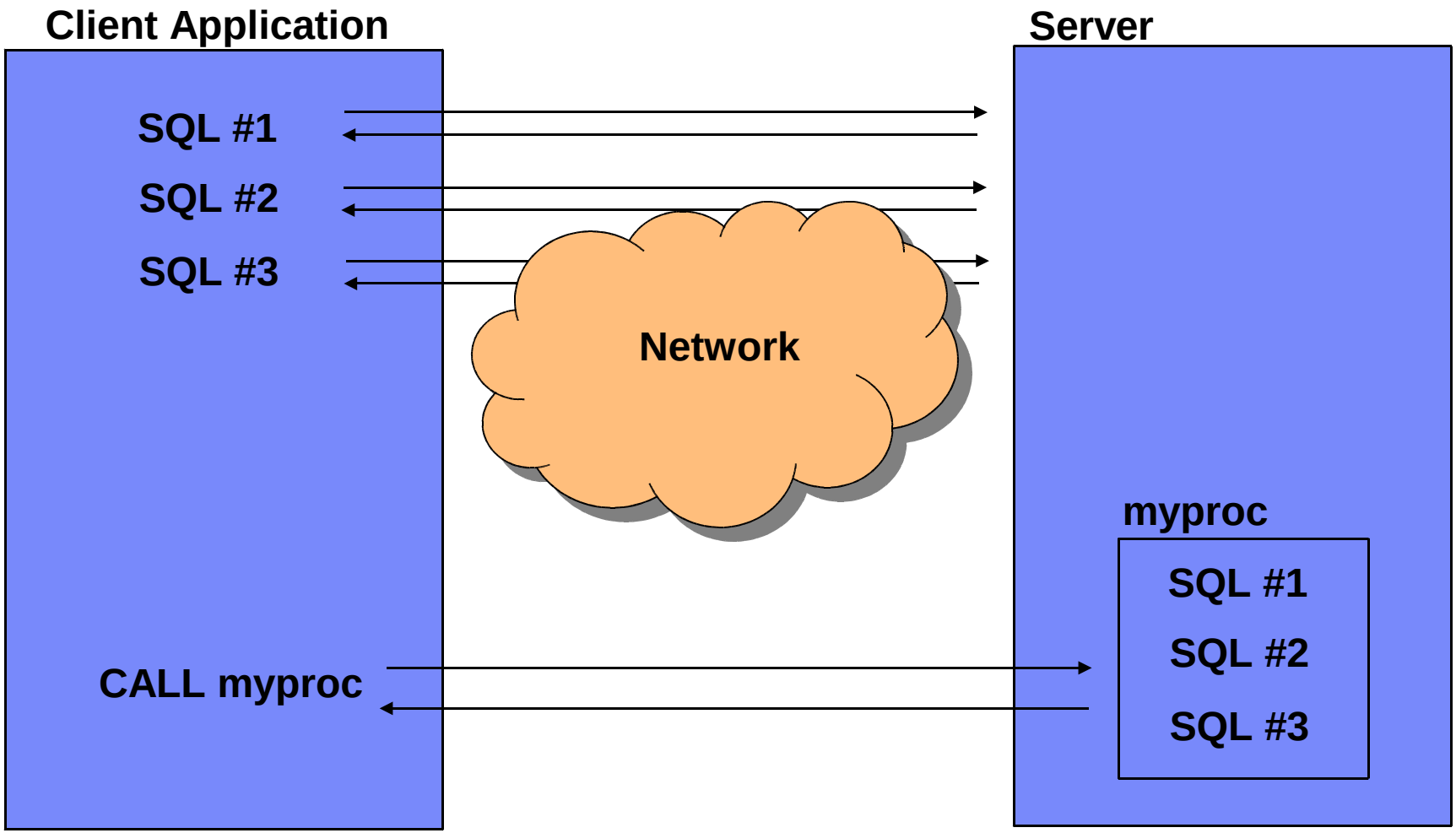
Stored procedures overview



Stored procedures overview



Stored procedures overview



Stored procedures overview

- Usually contain one or more SQL statements as well as procedural (business) logic
- Executed and managed by DB2 (server-side objects)
- Can be written using SQL PL, C/C++, Java, Cobol, CLR supported languages, OLE, PL SQL, etc.
- Benefits for using stored procedures include:
  - Centralized business logic that promotes code re-use
  - Improved security
  - Improved performance
- This workshop focuses on SQL PL procedures because of their popularity, good performance and simplicity

Creating your first stored procedure

- Using the Command Line Processor:

```
db2=> connect to sample
db2=> create procedure p1 begin end
```

- Using the IBM Data Studio
  (Demo)

Basic stored procedure structure

CREATE PROCEDURE proc_name *[({optional parameters})]*
[optional procedure attributes]
<statement>

[optional parameters]

IN	**Input parameter**
OUT	**Output parameter**
INOUT	**Input and Output parameter**

Example:

```
CREATE PROCEDURE proc(IN p1 INT, OUT p2 INT, INOUT p3 INT)
    …
```

Basic stored procedure structure

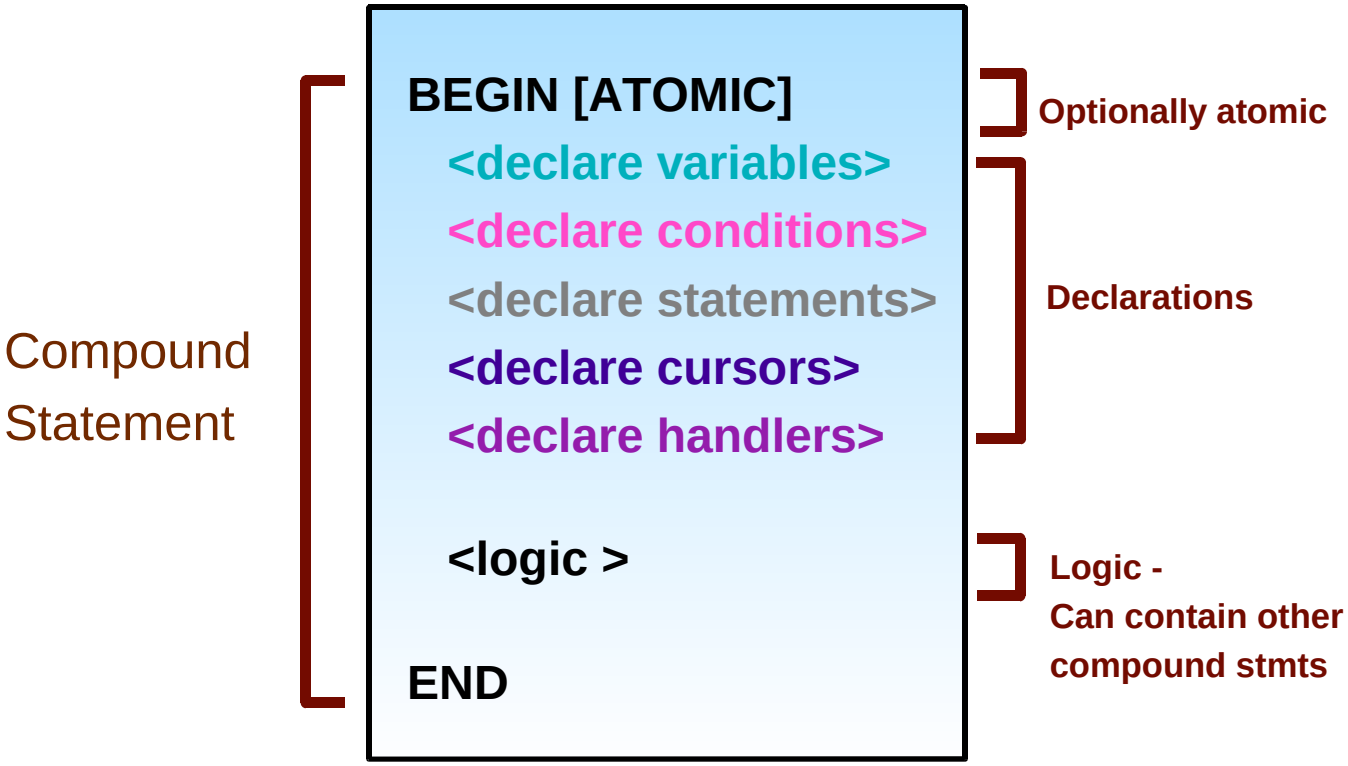
[optional procedure attributes]

LANGUAGE SQL

RESULT SETS <n> (required if returning result sets)

<*statement*> is a single statement, or a set of statements grouped by BEGIN [ATOMIC] ... END

Basic stored procedure structure: Compound statements



Variable declaration & assignments

DECLARE var_name <data type> [DEFAULT value]

Examples:

```
DECLARE temp1 SMALLINT DEFAULT 0;
DECLARE temp2 VARCHAR(10) DEFAULT 'hello';
DECLARE temp3 DATE DEFAULT '1998-12-25';
```

SET var_name = value

Examples:

- **SET total = 100;**
  - **Same as VALUES(100) INTO total;**
- **SET total = NULL;**
  - **any variable can be set to NULL**
- **SET total = (select sum(c1) from T1);**
  - **Condition is raised if more than one row**
- **SET first_val = (select c1 from T1 fetch first 1 row only)**
  - **fetch only the first row from a table**
- **SET sch = CURRENT SCHEMA;**

Example: Stored procedure with parameters

```
CREATE PROCEDURE P2 ( IN    v_p1 INT,
                      INOUT v_p2 INT,
                      OUT   v_p3 INT)
LANGUAGE SQL
SPECIFIC myP2
BEGIN
    -- my second SQL procedure
    SET v_p2 = v_p2 + v_p1;
    SET v_p3 = v_p1;
END
```

To call the procedure from the Command Line Processor:

```
db2=> call P2 (3, 4, ?)
```

Example: Stored procedure processing a cursor

```
CREATE PROCEDURE sum_salaries(OUT sum INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE p_sum INTEGER;
  DECLARE p_sal INTEGER;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE c CURSOR FOR
            SELECT SALARY FROM EMPLOYEE;
  SET p_sum = 0;
  OPEN c;
  FETCH FROM c INTO p_sal;
  WHILE(SQLSTATE = '00000') DO
     SET p_sum = p_sum + p_sal;
     FETCH FROM c INTO p_sal;
  END WHILE;
  CLOSE c;
  SET sum = p_sum;
END
```

SQLCODE and SQLSTATE

- Access requires explicit declaration:
  - **DECLARE SQLSTATE CHAR(5);**
  - **DECLARE SQLCODE INT;**
- Can be declared ONLY at outermost scope and automatically set by DB2 after each operation
- SQLCODE
  - **= 0, successful.**
  - **> 0, successful with warning**
  - **< 0, unsuccessful**
  - **= 100, no data was found.**
    - **i.e. FETCH statement returned no data**
- SQLSTATE
  - **SQLSTATE '00000' = Success**
  - **SQLSTATE '02000' = Not found**
  - **SQLSTATE '01XXX' = Warning**
  - **Everything else = Exception**

Example: Calling a stored procedure from Java application

```
try
{
  // Connect to sample database
  String url = "jdbc:db2:sample";
  con = DriverManager.getConnection(url);
  CallableStatement cs = con.prepareCall("CALL
  trunc_demo(?, ?)");
  // register the output parameters
  callStmt.registerOutParameter(1, Types.VARCHAR);
  callStmt.registerOutParameter(2, Types.VARCHAR);
  cs.execute();
  con.close();
}
catch (Exception e)
{
  /* exception handling logic goes here */
}
```

More examples at:

C:\Program Files\IBM\SQLLIB\samples

Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - **User-defined functions**
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

User-defined functions

- Functions always return a value
- Some built-in functions already exist out-of-the-box
  - Eg: SUM(), AVG(), DIGITS(), etc.
- Can create UDFs in:
  - SQL PL, C/C++, Java, CLR, OLE, etc.
  - In this workshop, we focus on SQL PL functions because of their simplicity and popularity

Type of functions

- **Scalar functions**
  - Return a single value
  - Cannot change database state (i.e. no INSERT, UPDATE, DELETE statements allowed)
    - Example: COALESCE( ), SUBSTR( )
- **Table functions**
  - Return values in a table format
  - Called in the FROM clause of a query
  - Can change database state (i.e. allow INSERT, UPDATE, DELETE statements)
    - Example: SNAPSHOT_DYN_SQL( ), MQREADALL( )
- **Others type of functions (not covered in this course):**
  - Row functions
  - Column functions

Scalar functions

- Scalar functions take input values and return a single value
- They cannot be used to modify table data

```
CREATE FUNCTION deptname(p_empid VARCHAR(6))
RETURNS VARCHAR(30)
SPECIFIC deptname
BEGIN ATOMIC
    DECLARE v_department_name VARCHAR(30);
    DECLARE v_err VARCHAR(70);
    SET v_department_name = (
        SELECT d.deptname FROM department d, employee e
            WHERE e.workdept=d.deptno AND e.empno= p_empid);
    SET v_err = 'Error: employee ' || p_empid || ' was not found';
    IF v_department_name IS NULL THEN
        SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT=v_err;
    END IF;
  RETURN v_department_name;
END
```

Invoking a scalar function

- Scalar UDFs can be invoked in SQL statements wherever a scalar value is expected, or in a VALUES clause

```
SELECT DEPTNAME('000010') FROM SYSIBM.SYSDUMMY1

VALUES DEPTNAME('000010')
```

Table UDFs

- Returns a table
- Used in the FROM clause of a query
- Typically used to return a table and keep an audit record

Example: A function that enumerates a set of employees of a department

```
CREATE FUNCTION getEnumEmployee(p_dept VARCHAR(3))
RETURNS TABLE
    (empno CHAR(6),
     lastname VARCHAR(15),
     firstnme VARCHAR(12))
SPECIFIC getEnumEmployee
RETURN
  SELECT e.empno, e.lastname, e.firstnme
    FROM employee e
    WHERE e.workdept=p_dept
```

Calling a table UDFs

- Used in the FROM clause of an SQL statement
- The TABLE() function must be applied and must be aliased.

```
SELECT * FROM
    TABLE (getEnumEmployee('E01')) T
```

Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - **Triggers**
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

Triggers

- A trigger is a database object defined on a table and fired when an INSERT, UPDATE, or DELETE operation is performed.
- Activate (“fire”) automatically
- Operations that cause triggers to fire are called *triggering* SQL statements

Types of triggers

- **BEFORE**
  - Activation before row is inserted, updated or deleted
- **AFTER**
  - Activated after the triggering SQL statement has executed to successful completion
- **INSTEAD OF**
  - Defined on views
  - Logic defined in the trigger is executed *instead of* the triggering SQL statement

Example of a BEFORE trigger

define qualifier for new values

```
CREATE TRIGGER default_class_end
NO CASCADE BEFORE INSERT ON cl_sched
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
WHEN (n.ending IS NULL)
    SET n.ending = n.starting + 1 HOUR
```

optional WHEN

if no value provided on insert, column is NULL

Example of an AFTER trigger

- Similar to BEFORE triggers, except that INSERT, UPDATE and DELETE are supported
- Prereq:
  - **CREATE TABLE audit (mytimestamp timestamp, comment varchar (1000))**

```
CREATE TRIGGER audit_emp_sal
AFTER UPDATE OF salary ON employee
REFERENCING OLD AS o NEW AS n
FOR EACH ROW
MODE DB2SQL
  INSERT INTO audit VALUES (
  CURRENT TIMESTAMP, ' Employee ' || o.empno || '
    salary changed from ' || CHAR(o.salary) || ' to
    ' || CHAR(n.salary) || ' by ' || USER)
```

Example of an INSTEAD OF trigger

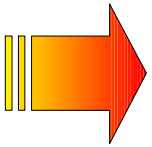
- It is activated when performing changes to a view
- Prereq:

```
CREATE TABLE countries (id int, country varchar(50),
                        region varchar (50), average_temp int)
CREATE VIEW view1 (id, continent, temperature) as
       SELECT id, region, average_temp from countries
```

```
CREATE TRIGGER update_view1
       INSTEAD OF UPDATE ON view1
       REFERENCING OLD AS o NEW AS n
       FOR EACH ROW
       MODE DB2SQL
       BEGIN ATOMIC
          UPDATE countries
          SET region = n.region
          WHERE region = o.region;
       END
```

Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers

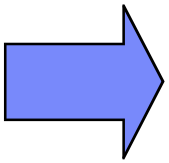


- **Client-side development**
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

DB2 Application Development Overview

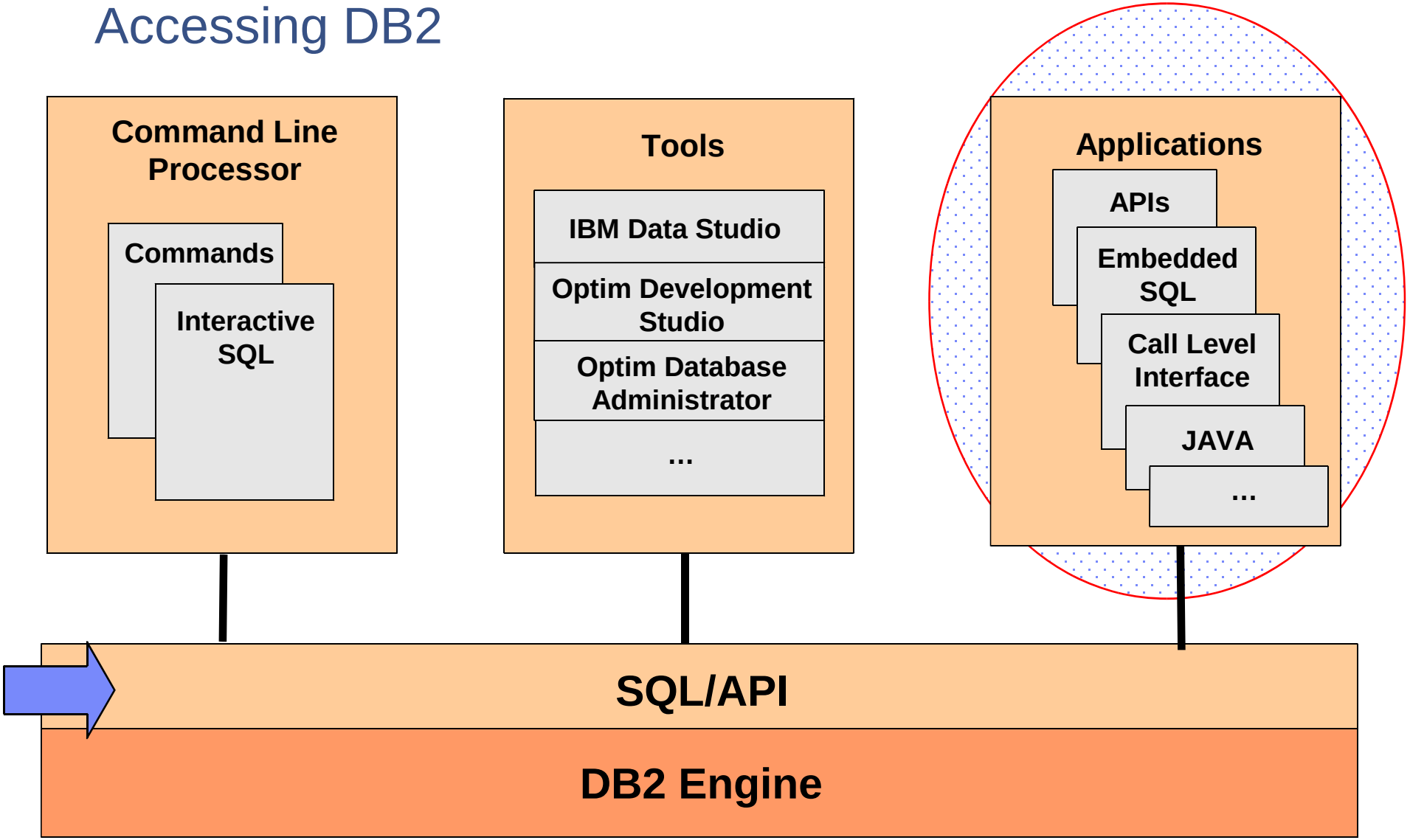
Server-side development (at the DB2 database server):

- Routines (Stored Procedures, UDFs)
- Database objects (Triggers)



- **Client-side development (at the client):**
  - May require a DB2 client or driver to be installed
  - Database applications (in C/C++, .NET, Cobol, Java, etc)

Accessing DB2



Application development freedom

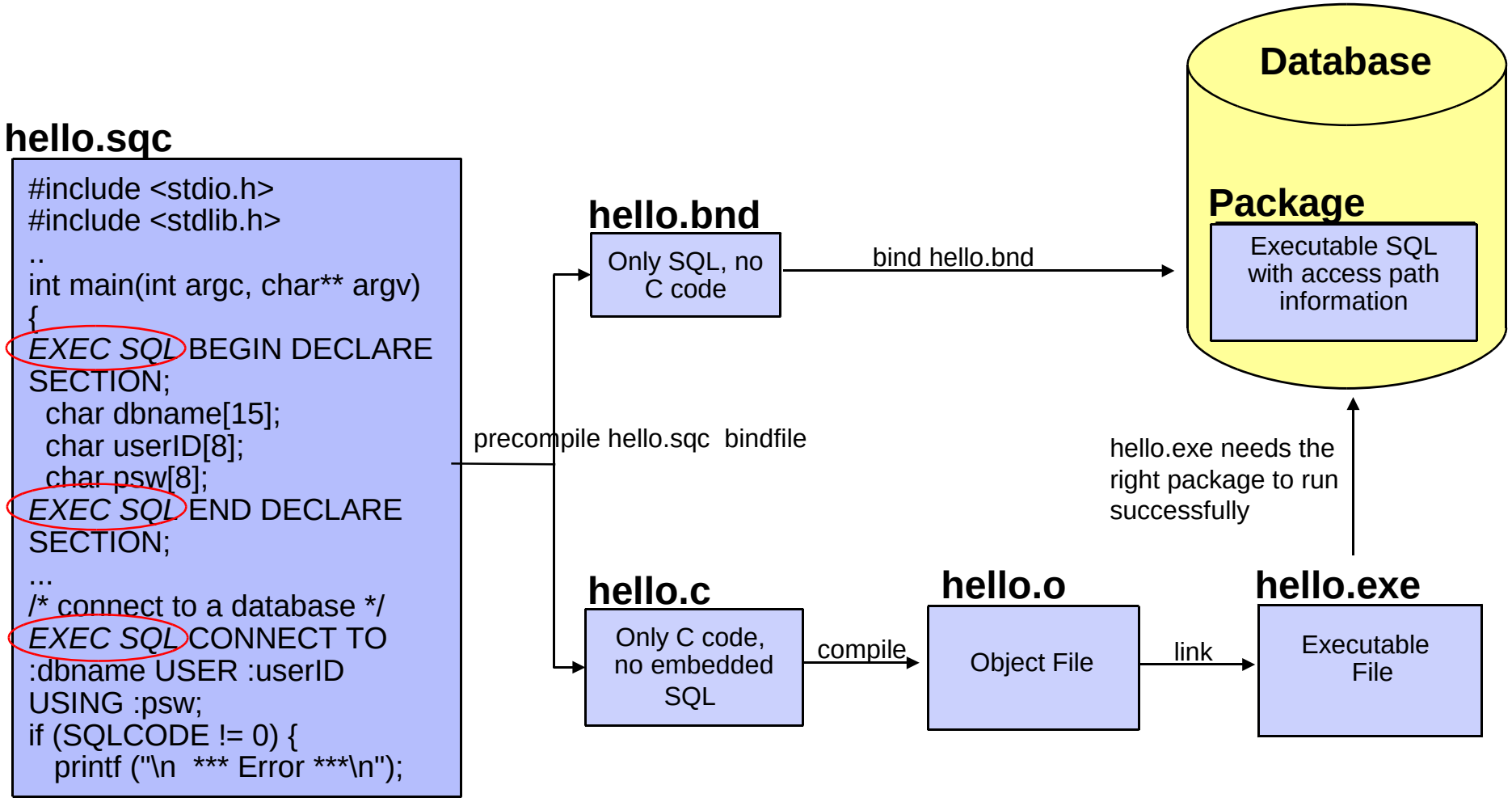
- Ruby on Rails
- C/C++ (ODBC and Static SQL)
- JDBC and SQLJ
- Borland
- Python
- PHP
- Perl
- .NET languages
- OLE-DB
- ADO
- Web Services
- SQL
- MS Office: Excel, Access, Word



Agenda

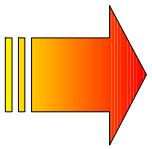
- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - **Embedded SQL**
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery

Embedded SQL



Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - **Static vs. Dynamic SQL**
  - CLI/ODBC
  - JDBC / SQLJ / pureQuery



Static SQL

- The SQL statement structure is fully known at precompile time.

```
SELECT lastname, salary FROM employee
```

  - The names for the columns (lastname, firstname) and tables (employee) referenced in a statement are fully known at precompile time.

- Host variables values can be specified at run time (but their data type must still be precompiled).

```
SELECT lastname, salary
   FROM employee
  WHERE firstname = :fname
```

- You precompile, bind, and compile statically executed SQL statements before you run your application.

- Static SQL is best used on databases whose statistics do not change a great deal.

Dynamic SQL

- The SQL is built and executed at run-time.

```
SELECT ?, ? FROM ?
```

  - The names for the columns and tables referenced in a statement are not known until runtime.

- The access plan is determined at runtime.
  - Normally static SQL performs better than dynamic SQL since the access plan is calculated ahead of time
  - For tables whose statistics change often, dynamic SQL may provide a more accurate access plan.

- When working with dynamic SQL, use parameter markers (?) to reduce the amount of times an access plan is calculated. (see following example)

Dynamic SQL

- Example:

Case 1:

```
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 1
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 2
```

Case 2:

```
strcpy(hVStmtDyn, “SELECT name FROM emp WHERE dept = ?");
PREPARE StmtDyn FROM :hVStmtDyn;
EXECUTE StmtDyn USING 1;
EXECUTE StmtDyn USING 2;
```

- In case 1, each statement is treated as different SQL, therefore DB2 will calculate the access plan for each.
- In case 2, there is only one SQL statement:

  “SELECT name FROM emp WHERE dept = ?“

- Therefore, the access plan will only be calculated once, and cached in memory.

Static vs. Dynamic SQL

- Embedded SQL applications support static & dynamic SQL
- Example of a static SQL in an embedded SQL C program

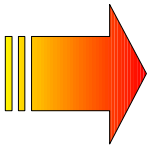
```
EXEC SQL SELECT name, dept
           INTO :name, :dept
           FROM staff WHERE id = 310;
printf( …)
```

- Example of a dynamic SQL in an embedded SQL C program

```
...
strcpy(hostVarStmtDyn,
  "UPDATE staff SET salary = salary + 1000 WHERE dept = ?");
EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
EXEC SQL EXECUTE StmtDyn USING :dept;
```

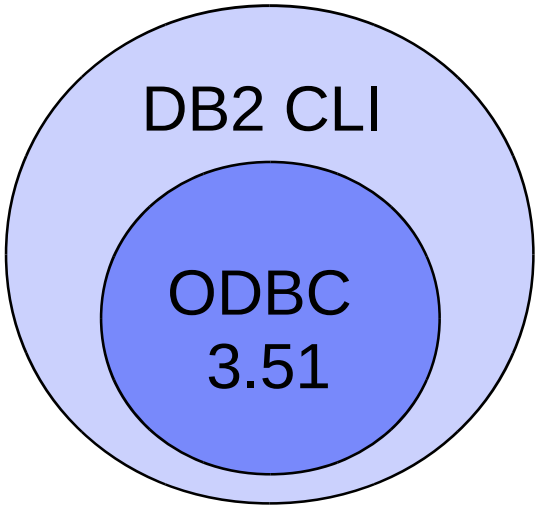
Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - **CLI/ODBC**
  - JDBC / SQLJ / pureQuery



CLI / ODBC

- CLI = Call Level Interface
- DB2 CLI can be used as the ODBC Driver when loaded by an ODBC Driver Manager
- DB2 CLI conforms to ODBC 3.51



CLI / ODBC

- To ***run*** a CLI/ODBC application all you need is the DB2 CLI driver. This driver is installed from either of these:
  - IBM Data Server Client
  - IBM Data Server Runtime Client
  - IBM Data Server Driver for ODBC and CLI

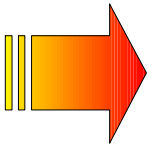
- To ***develop*** a CLI/ODBC application you need the DB2 CLI driver and also the appropriate libraries. These can be found only on:
  - IBM Data Server Client

CLI / ODBC

- CLI/ODBC characteristics:
  - The code is easily portable between several RDBMS vendors
  - Unlike embedded SQL, there is no need for a precompiler or host variables
  - It runs dynamic SQL
  - It is very popular

Agenda

- DB2 Application Development overview
- Server-side development
  - Stored Procedures
  - User-defined functions
  - Triggers
- Client-side development
  - Embedded SQL
  - Static vs. Dynamic SQL
  - CLI/ODBC
  - **JDBC / SQLJ / pureQuery**



JDBC / SQL / pureQuery

- **JDBC characteristics:**
  - Like in ODBC, the code is easily portable between several RDBMS vendors
  - Dynamic SQL
  - It is very popular
- **SQLJ**
  - Embedded SQL in Java
  - Static SQL
  - Not that popular
- **pureQuery**
  - Eclipse-based plug-in to manage relational data as objects
  - IBM's paradigm to develop Java database applications
  - New since mid-2007, available with Optim Development Studio

JDBC / SQLJ – Supported drivers

Driver Type	Driver Name	Packaged as	Supports	Minimum level of SDK for Java required
Type 2	DB2 JDBC Type 2 Driver for Linux, UNIX and Windows **(Deprecated)**	db2java.zip	JDBC 1.2 and JDBC 2.0	1.4.2
Type 2 and Type 4	IBM Data Server Driver for JDBC and SQLJ	db2jcc.jar and sqlj.zip	JDBC 3.0 compliant	1.4.2
		db2jcc4.jar and sqlj4.zip	JDBC 4.0 and earlier	6

- **Type 2 drivers need to have a DB2 client installed**
- **Deprecated means it is still supported, but no longer enhanced**
- **Note that the same file (for example db2jcc.jar) supports type 2 and 4**

JDBC / SQLJ – Supported drivers

- **db2java.zip, db2jcc.jar, sqlj.zip, db2jcc4.jar and sqlj4.zip are included with:**
  - IBM DB2 for Linux, UNIX and Windows servers
  - IBM Data Server Client
  - IBM Data Server Runtime Client
  - IBM Data Server Driver for JDBC and SQLJ

Thank you!