Oracle 11g – PL SQL

PL/SQL subprograms

About Sub Programs

To explain PL/SQL subprograms units

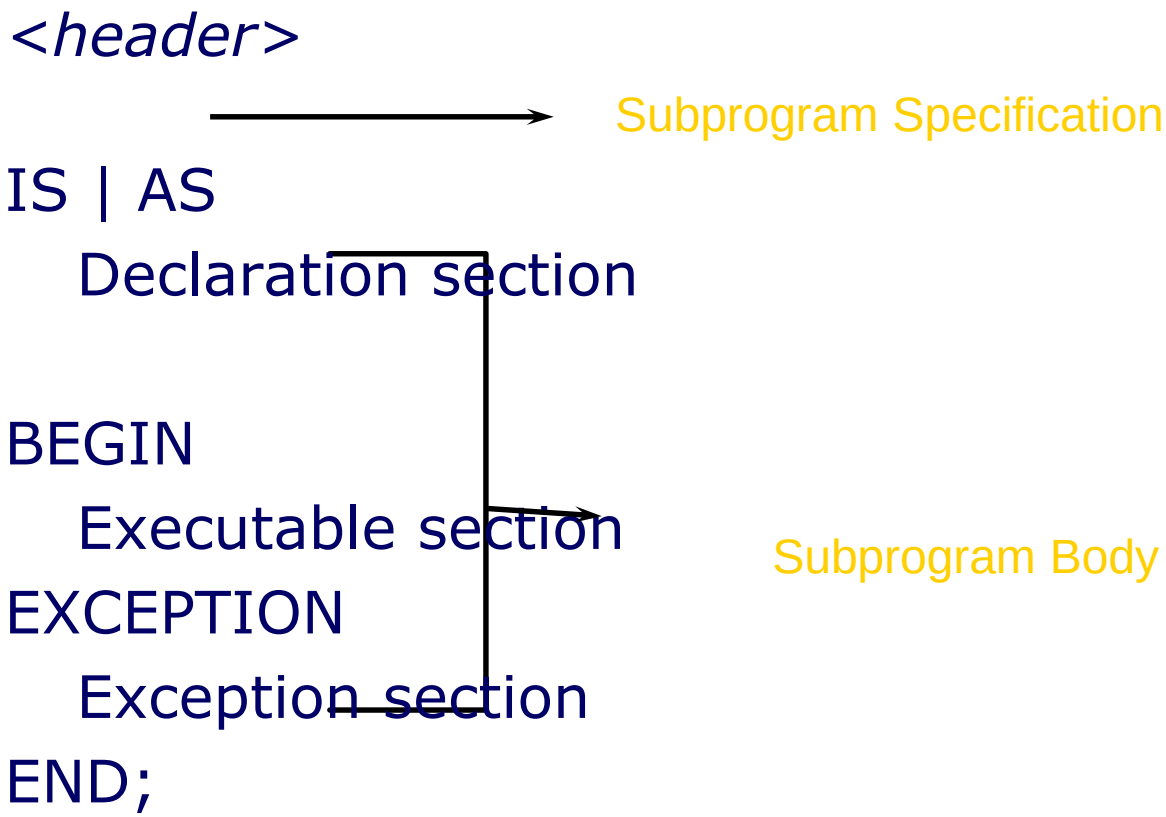
- Procedures
- Functions

Overview of Subprograms

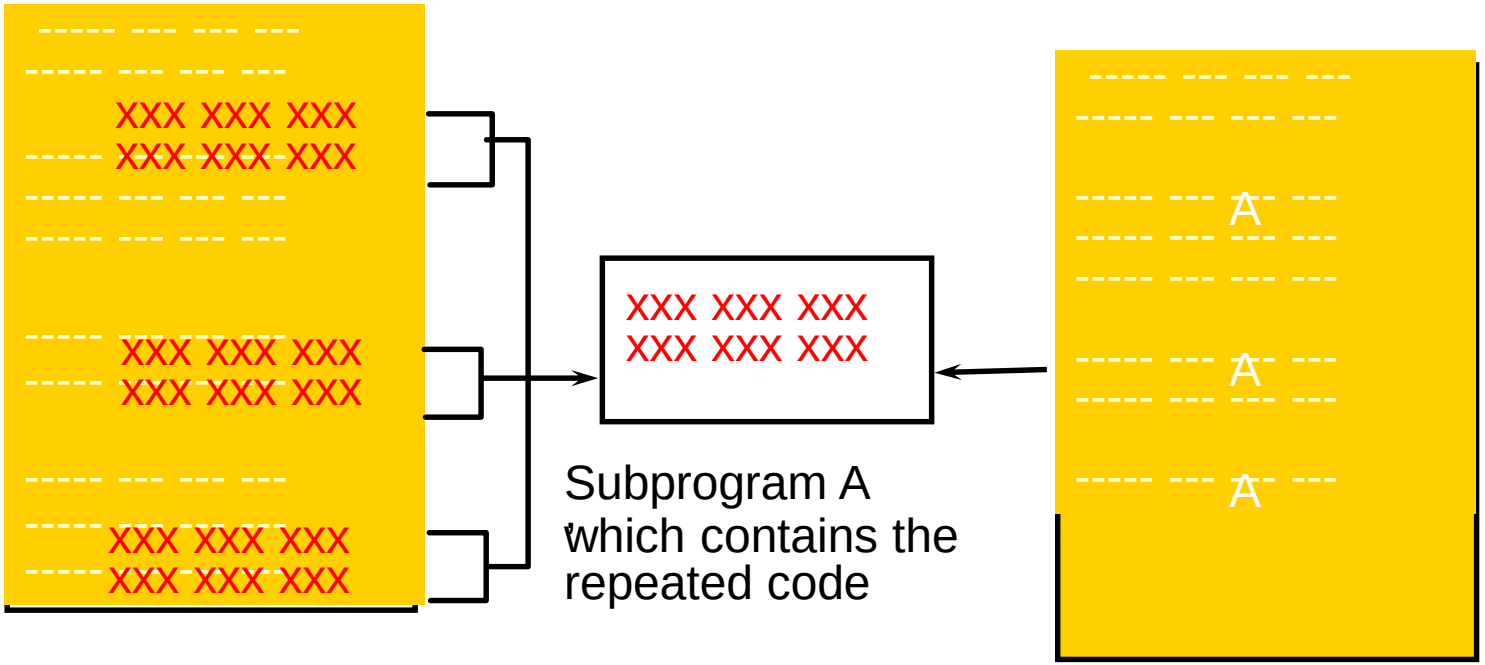
A subprogram:

- Is a named PL/SQL block that can accept parameters and be invoked from a calling environment
- Is of two types:
  - A procedure that performs an action
  - A function that computes a value
- Is based on standard PL/SQL block structure
- Provides modularity, reusability, extensibility, and maintainability
- Provides easy maintenance, improved data security and integrity, improved performance, and improved code clarity

Block Structure for PL/SQL Subprograms



PL/SQL Subprograms



Code repeated more than once in a PL/SQL program

PL/SQL program invoking the subprogram at multiple locations

Benefits of Subprograms

- Easy maintenance
- Improved data security and integrity
- Improved performance
- Improved code clarity

What Is a Procedure?

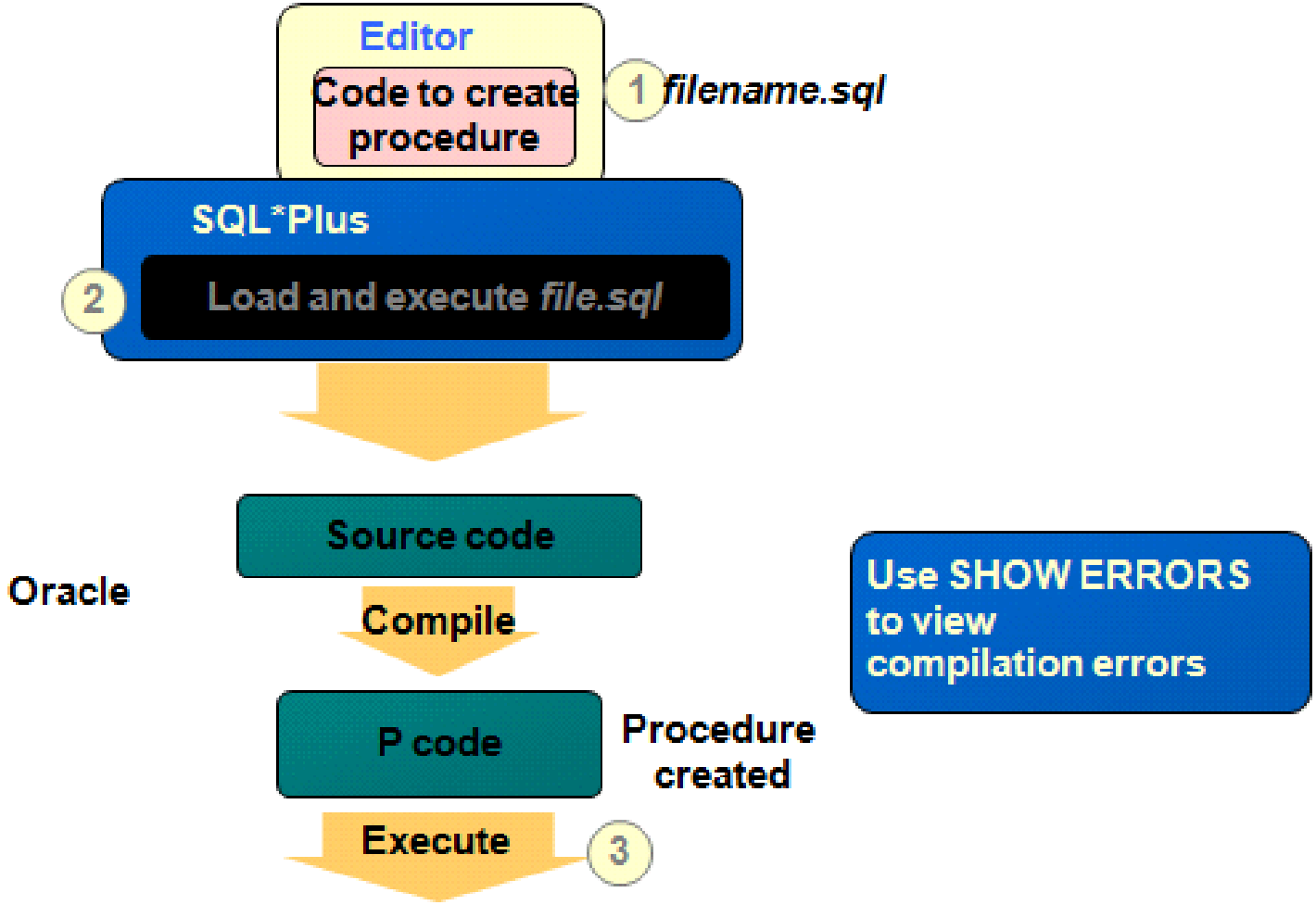
- A procedure is a type of subprogram that performs an action.
- A procedure can be stored in the database, as a schema object, for repeated execution.

Syntax for Creating Procedures

```
CREATE [OR REPLACE] PROCEDURE procedurename
 (parameter1 [mode] datatype1,
  parameter2 [mode] datatype2,
  . . .)
IS|AS
PL/SQL Block;
```

- The REPLACE option indicates that if the procedure exists, it will be dropped and replaced with the new version created by the statement.
- PL/SQL block starts with either BEGIN or the declaration of local variables and ends with either END or END *procedurename.*

Developing Procedures



Formal Versus Actual Parameters

•Formal parameters: variables declared in the parameter list of a subprogram specification

Example:

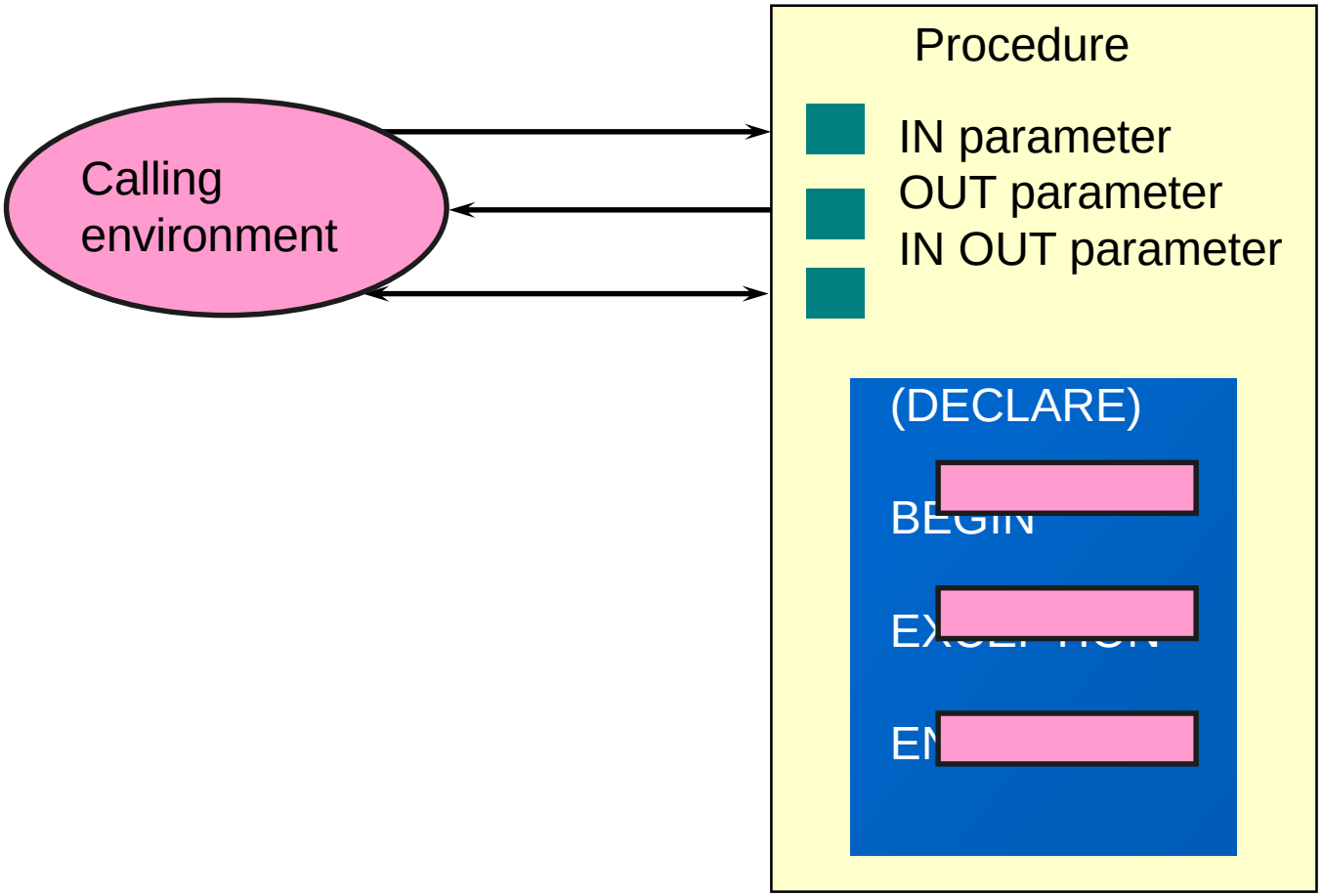
```
CREATE PROCEDURE in_salary(p_id NUMBER, p_amount
NUMBER)
...
END inc_salary;
```

•Actual parameters: variables or expressions referenced in the parameter list of a subprogram call

Example:

```
in_salary(v_id, 2000)
```

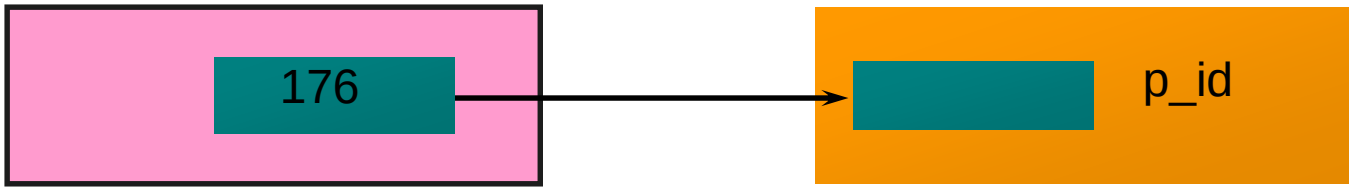
Procedural Parameter Modes



Creating Procedures with Parameters

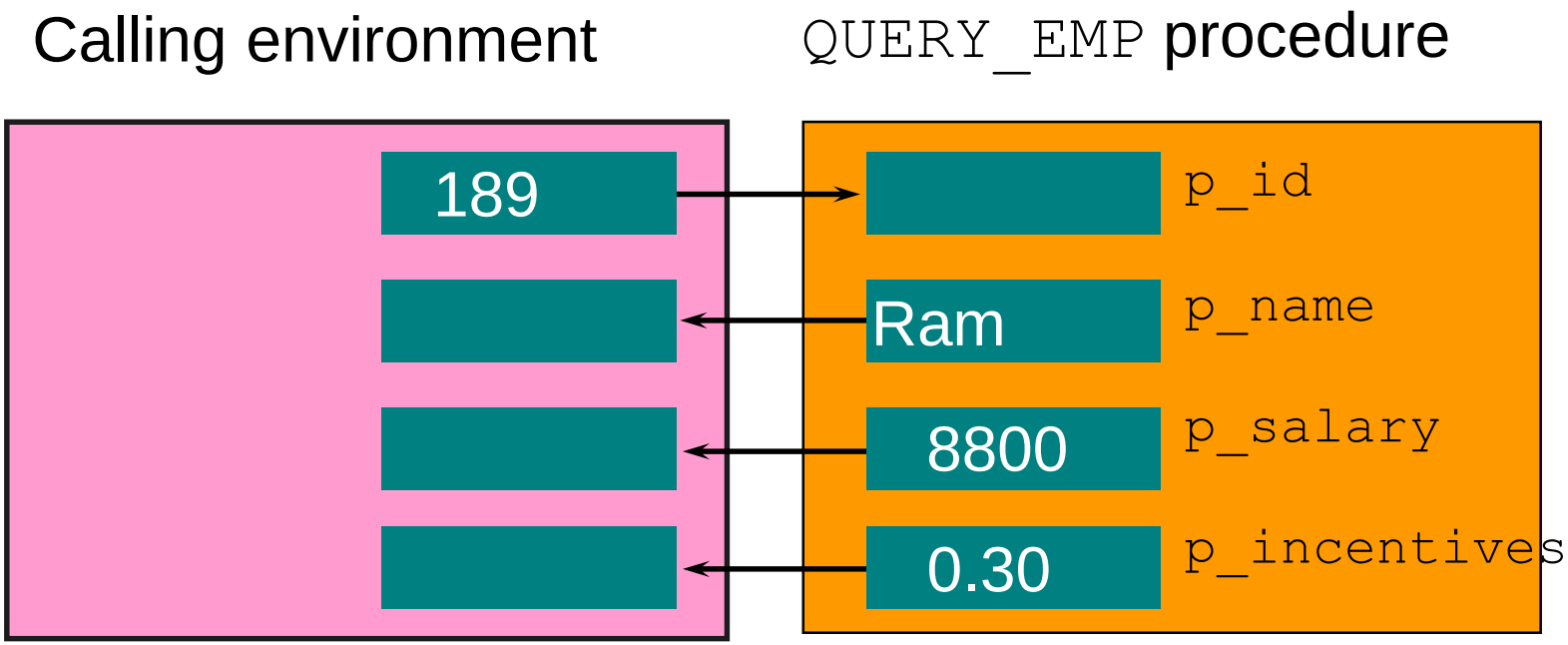
IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

IN Parameters: Example




```
CREATE OR REPLACE PROCEDURE inc_salary
  (p_id IN empl.emp_id%TYPE)
IS
BEGIN
  UPDATE empl
  SET    salary = salary * 1.10
  WHERE  emp_id = p_id;
END inc_salary;
/
```

OUT Parameters: Example



OUT Parameters: Example

```
CREATE OR REPLACE PROCEDURE query1
  (p_id      IN   empl.emp_id%TYPE,
   p_name    OUT  empl.name%TYPE,
   p_salary  OUT  empl.salary%TYPE,
   p_comm    OUT  empl.commission%TYPE)
IS
BEGIN
  SELECT   name, salary, commission
   INTO    p_name, p_salary, p_comm
   FROM    empl
   WHERE   emp_id = p_id;
END query1;
/
```

Viewing OUT Parameters

- Load and run the emp_query.sql script file to create the QUERY1 procedure.
- Declare host variables, execute the QUERY1 procedure, and print the value of the global G_NAME variable.

```
VARIABLE g_name        VARCHAR2(25)
VARIABLE g_salary      NUMBER
VARIABLE g_comm        NUMBER

EXECUTE query1(189, :g_name, :g_sal, :g_comm)

PRINT g_name
```

Viewing IN OUT Parameters

```
VARIABLE g_phone VARCHAR2(15)
BEGIN
  :g_phone := ‘919885260358';
END;
/
PRINT g_phone
EXECUTE Phone_proc (:g_phone)
PRINT g_phone
```

Methods for Passing Parameters

- Positional: List actual parameters in the same order as formal parameters.
- Named: List actual parameters in arbitrary order by associating each with its corresponding formal parameter.
- Combination: List some of the actual parameters as positional and some as named.

DEFAULT Option for Parameters

```
CREATE OR REPLACE PROCEDURE add_department
  (p_name  IN dep.dep_name%TYPE DEFAULT 'abc',
   p_loc   IN dep.loc_id%TYPE         DEFAULT 1900)
IS
BEGIN
  INSERT INTO dep(dep_id,
            dep_name, loc_id)
  VALUES (seq1.NEXTVAL, p_name, p_loc);
END add_department;
/
```

Examples of Passing Parameters

```
BEGIN
 add_department;
 add_department ('TRNG', 3500);
 add_department ( p_loc => 3400, p_name =>'EDU');
 add_department ( p_loc => 3200) ;
END;
/
SELECT dep_id, dep_name, loc_id
FROM dep;
```

Declaring Subprograms

```
CREATE OR REPLACE PROCEDURE empl_leave
  (p_id  IN  empl.emp_id%TYPE)
IS
  PROCEDURE log_exec1
  IS
 BEGIN
    INSERT INTO log_table1 (user_id, log_date)
    VALUES (USER, SYSDATE);
 END log_exec1;
BEGIN
  DELETE FROM empl
  WHERE emp_id = p_id;
  log_exec1;
END emp_leave;
/
```

Invoking a Procedure from an Anonymous PL/SQL Block

```
DECLARE
  v_id NUMBER := 123;
BEGIN
  inc_salary(v_id);    --invoke procedure
  COMMIT;
...
END;
```

Invoking a Procedure from Another Procedure

```
CREATE OR REPLACE PROCEDURE emp_process
IS
   CURSOR c1 IS
    SELECT emp_id
    FROM   empl;
BEGIN
   FOR i IN c1
   LOOP
    inc_salary(emp_rec.employee_id);
   END LOOP;
   COMMIT;
END emp_process;
/
```

Removing Procedures

Drop a procedure stored in the database.

Syntax:

```
DROP PROCEDURE procedurename
```

Example:

```
DROP PROCEDURE inc_salary;
```

Stored Functions

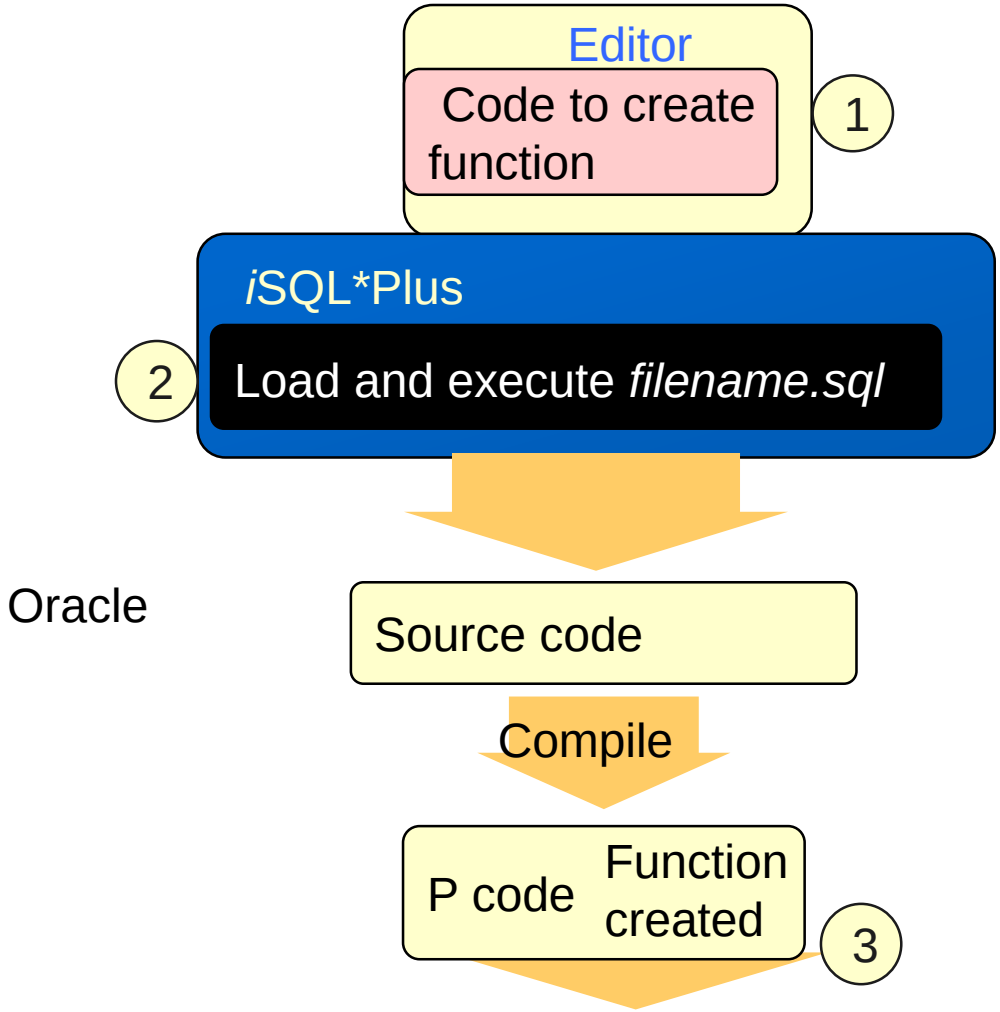
- A function is a named PL/SQL block that returns a value.
- A function can be stored in the database as a schema object for repeated execution.
- A function is called as part of an expression.

Syntax for Creating Functions

```
CREATE [OR REPLACE] FUNCTION functionname
 (parameter1 [mode] datatype1,
  parameter2 [mode] datatype2,
  . . .)
RETURN datatype
IS|AS
PL/SQL Block;
```

The PL/SQL block must have at least one RETURN statement.

Creating a Function



Executing Functions

- Invoke a function as part of a PL/SQL expression.
- Create a variable to hold the returned value.
- Execute the function. The variable will be populated by the value returned through a `RETURN` statement.

Advantages of User-Defined Functions in SQL Expressions

- Extend SQL where activities are too complex, too awkward, or unavailable with SQL
- Can increase efficiency when used in the `WHERE` clause to filter data, as opposed to filtering the data in the application
- Can manipulate character strings

Invoking Functions in SQL Expressions: Example

```
CREATE OR REPLACE FUNCTION taxcalc(p_val IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_val * 0.15);
END tax;
/
SELECT emp_id, name, salary, taxcalc(salary)
FROM   empl
WHERE  dep_id = 111;
```

Locations to Call User-Defined Functions

- Select list of a `SELECT` command
- Condition of the `WHERE` and `HAVING` clauses
- `CONNECT BY`, `START WITH`, `ORDER BY`, and `GROUP BY` clauses
- `VALUES` clause of the `INSERT` command
- `SET` clause of the `UPDATE` command

Restrictions on Calling Functions from SQL Expressions

.....

To be callable from SQL expressions, a user-defined function must:

- Be a stored function
- Accept only `IN` parameters
- Accept only valid SQL data types, not PL/SQL specific types, as parameters
- Return data types that are valid SQL data types, not PL/SQL specific types

Restrictions on Calling Functions from SQL Expressions

- Functions called from SQL expressions cannot contain DML statements.
- Functions called from `UPDATE`/`DELETE` statements on a table T cannot contain DML on the same table T.
- Functions called from an `UPDATE` or a `DELETE` statement on a table T cannot query the same table.
- Functions called from SQL statements cannot contain statements that end the transactions.
- Calls to subprograms that break the previous restriction are not allowed in the function.

Restrictions on Calling from SQL

```
CREATE OR REPLACE FUNCTION f1 (p_salary NUMBER)
   RETURN NUMBER IS
BEGIN
   INSERT INTO employees(emp_id, name, mail,
                         join_date, job_id, salary)
                 VALUES(1, 'Lisa', 'Lisa@INFY.com',
                                   SYSDATE, 'EDUCATOR', 9000);
   RETURN (p_salary + 1100);
END;
/
```

```
UPDATE empl SET salary = f1(2000)
 WHERE emp_id = 121;
```

Removing Functions

Drop a stored function.

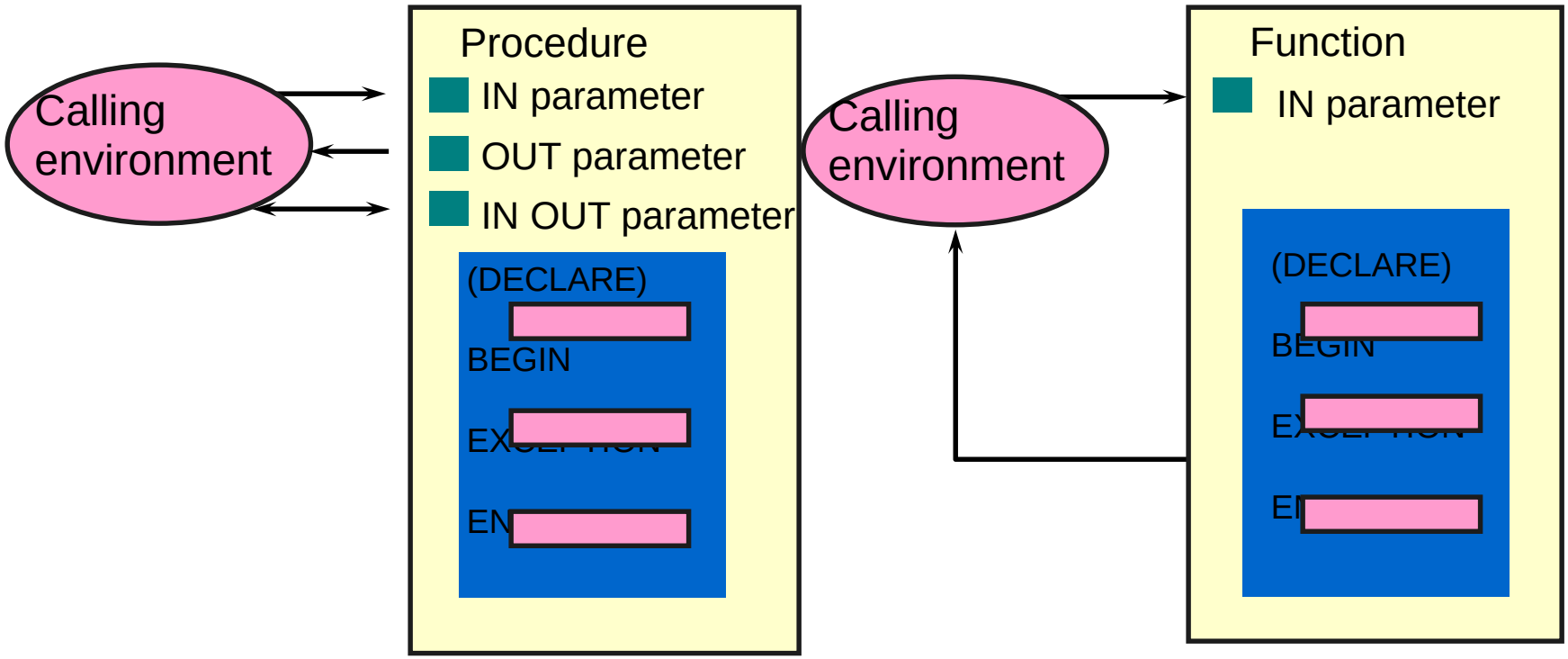
Syntax:

```
DROP FUNCTION f1;
```

Example:

- All the privileges granted on a function are revoked when the function is dropped.
- The CREATE OR REPLACE syntax is equivalent to dropping a function and recreating it. Privileges granted on the function remain the same when this syntax is used.

How Procedures and Functions Differ



Comparing Procedures and Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression Must contain a RETURN
Do not contain RETURN clause in the header	clause in the header
Can return none, one, or many values	Must return a single value
Can contain a RETURN statement	Must contain at least one RETURN statement

Benefits of Stored Procedures and Functions

- Improved performance
- Easy maintenance
- Improved data security and integrity
- Improved code clarity

List All Procedures and Functions

```
SELECT object_name, object_type
FROM  user_objects
WHERE object_type in ('PROCEDURE','FUNCTION')
ORDER BY object_name;
```

USER_SOURCE Data Dictionary View

Column	Column Description
NAME	Name of the object
TYPE	Type of object, for example, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY
LINE	Line number of the source code
TEXT	Text of the source code line

List the Code of Procedures and Functions

```
SELECT  text
FROM  user_source
WHERE  name = 'P1'
ORDER BY  line;
```

List Compilation Errors by Using SHOW ERRORS

SHOW ERRORS PROCEDURE p1

Debugging PL/SQL Program Units

- The `DBMS_OUTPUT` package:
  - Accumulates information into a buffer
  - Allows retrieval of the information from the buffer
- Autonomous procedure calls (for example, writing the output to a log table)
- Software that uses `DBMS_DEBUG`
  - Procedure Builder
  - Third-party debugging software

Invoking a User-Defined Package Function from a SQL Statement

```
SELECT taxes_package.tax(salary), salary, name
FROM   empl;
```