Oracle 11g – PL SQL

Exceptions Handling

About Exceptions

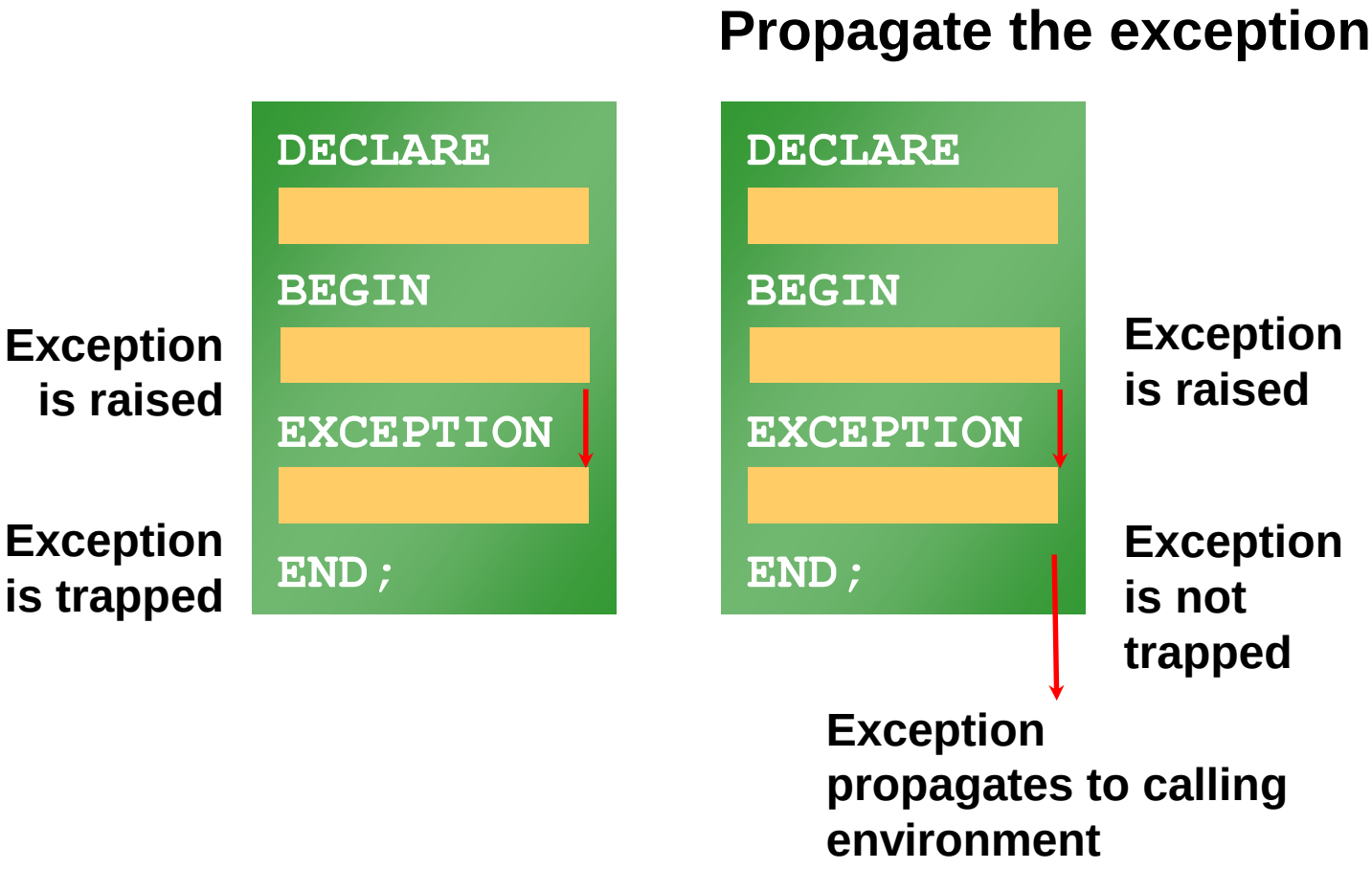
- *What is Exception*
- Handling Exceptions in PL/SQL
- Trap Predefined Oracle Server Errors
- Trap Non-Predefined Oracle Server Errors
- Trap User-Defined Exceptions
- Propagate Exceptions
- RAISE_APPLICATION_ERROR Procedure

What is Exception

- An exception is an identifier in PL/SQL that is raised during execution.
- How is it raised?
  - An Oracle error occurs.
  - You raise it explicitly.
- How do you handle it?
  - Trap it with a handler.
  - Propagate it to the calling environment.

Handling Exceptions in PLSQL

Trap the exception



Exception Types

- **Predefined** Oracle Server
- **Non-predefined** Oracle Server

} **Implicitly raised**

- **User-defined** **Explicitly raised**

Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Trapping Exceptions Guidelines

- The `EXCEPTION` keyword starts exception-handling section.
- Several exception handlers are allowed.
- Only one handler is processed before leaving the block.
- `WHEN OTHERS` is the last clause.

Trapping **Predefined** Oracle Server Errors

- Reference the standard name in the exception-handling routine.
- Sample predefined exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

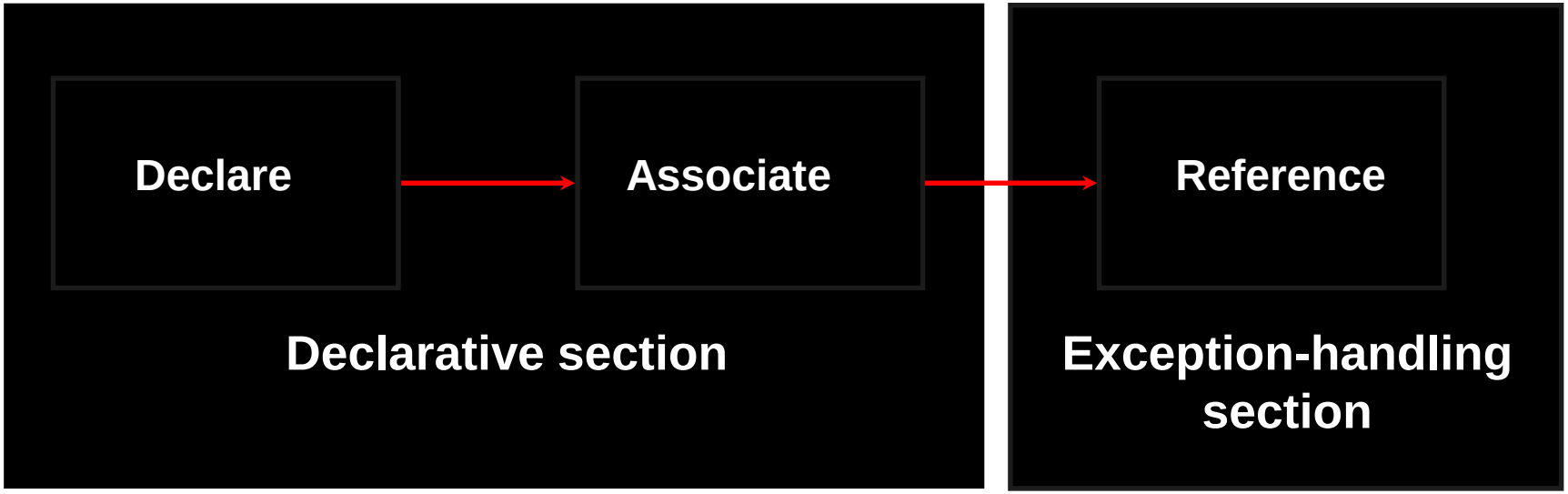
Predefined Exceptions

Syntax:

```
BEGIN
. . .
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      statement1;
      statement2;

   WHEN TOO_MANY_ROWS THEN
      statement1;
   WHEN OTHERS THEN
      statement1;
      statement2;
      statement3;
END;
```

Trapping **Non-predefined** Oracle Server Errors



Name the exception

Code the `PRAGMA EXCEPTION_INIT`

Handle the raised exception

Non-predefined Error

Trap for Oracle server error number –2292, an integrity constraint violation.

```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;
  PRAGMA EXCEPTION_INIT
     (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining   THEN
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno) || '.  Employees exist. ');
END;
```

1

2

3

Functions for Trapping Exceptions

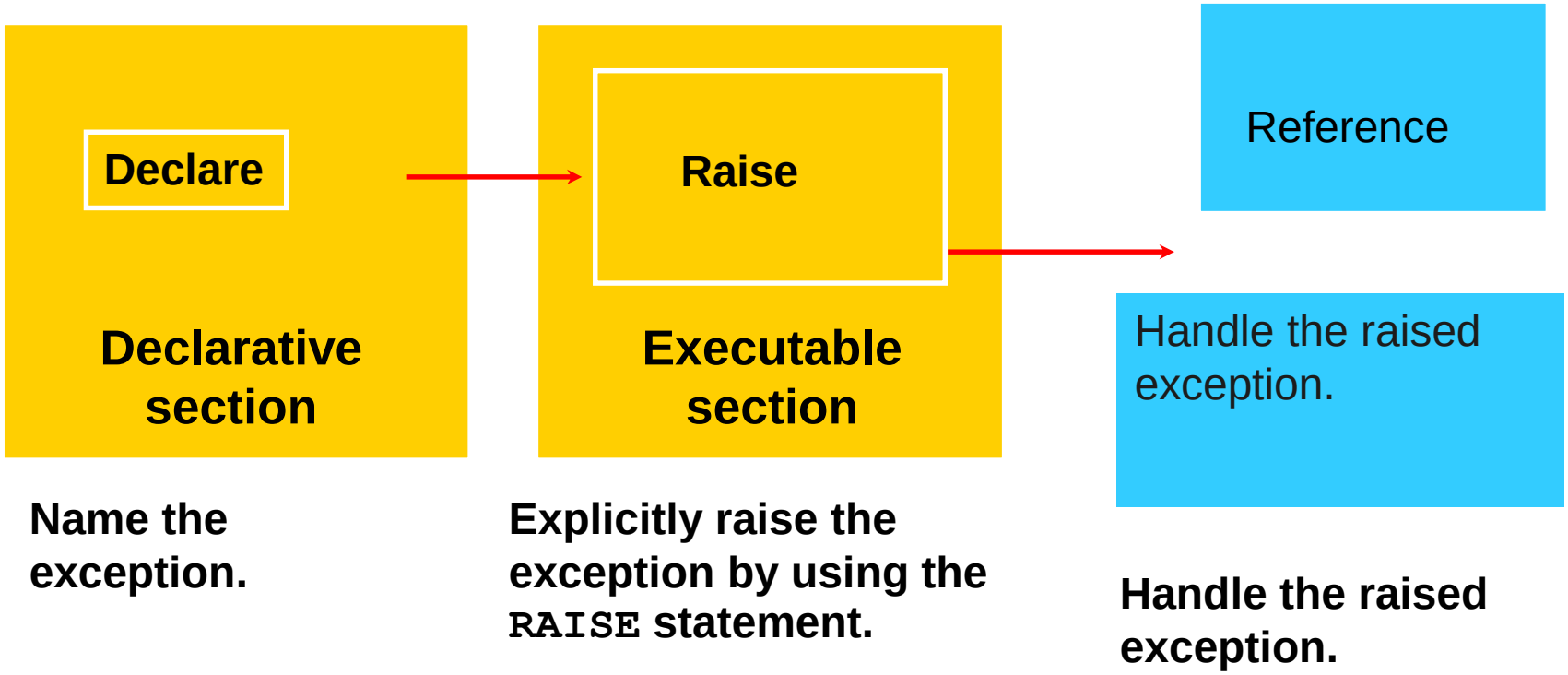
- `SQLCODE`: Returns the numeric value for the error code
- `SQLERRM`: Returns the message associated with the error number

Functions for Trapping Exceptions

Example:

```
DECLARE
   v_error_code       NUMBER;
   v_error_message    VARCHAR2(255);
BEGIN
...
EXCEPTION
...
   WHEN OTHERS THEN
      ROLLBACK;
      v_error_code := SQLCODE ;
      v_error_message := SQLERRM ;
      INSERT INTO errors
      VALUES(v_error_code, v_error_message);
END;
```

Trapping **User-Defined** Exceptions



User-Defined Exceptions

Example:

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300
```

```
DECLARE
   e_invalid_department EXCEPTION;
BEGIN
   UPDATE      departments
   SET         department_name = '&p_department_desc'
   WHERE       department_id = &p_department_number;
   IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
   END IF;
   COMMIT;
EXCEPTION
   WHEN e_invalid_department THEN
     DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

1

2

3

Calling Environments

*i*SQL*Plus	Displays error number and message to screen
Procedure Builder	Displays error number and message to screen
Oracle Developer Forms	Accesses error number and message in a trigger by means of the `ERROR_CODE` and `ERROR_TEXT` packaged functions
Precompiler application	Accesses exception number through the `SQLCA` data structure
An enclosing PL/SQL block	Traps exception in exception-handling routine of enclosing block

Propagating Exceptions

Subblocks can handle an exception or pass the exception to the enclosing block.

```
DECLARE
   . . .
   e_no_rows       exception;
   e_integrity     exception;
   PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
   FOR c_record IN emp_cursor LOOP
     BEGIN
      SELECT ...
      UPDATE ...
      IF SQL%NOTFOUND THEN
        RAISE e_no_rows;
      END IF;
     END;
   END LOOP;
   EXCEPTION
   WHEN e_integrity THEN ...
   WHEN e_no_rows THEN ...
END;
```

The RAISE_APPLICATION_ERROR Procedure

Syntax:

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

- You can use this procedure to issue user-defined error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

The RAISE_APPLICATION_ERROR Procedure

- Used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner consistent with other Oracle server errors

RAISE_APPLICATION_ERROR

Executable section:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
    ...
```

Exception section:

```
...
EXCEPTION
     WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR (-20201,
         'Manager is not a valid employee.');
END;
```