# 8

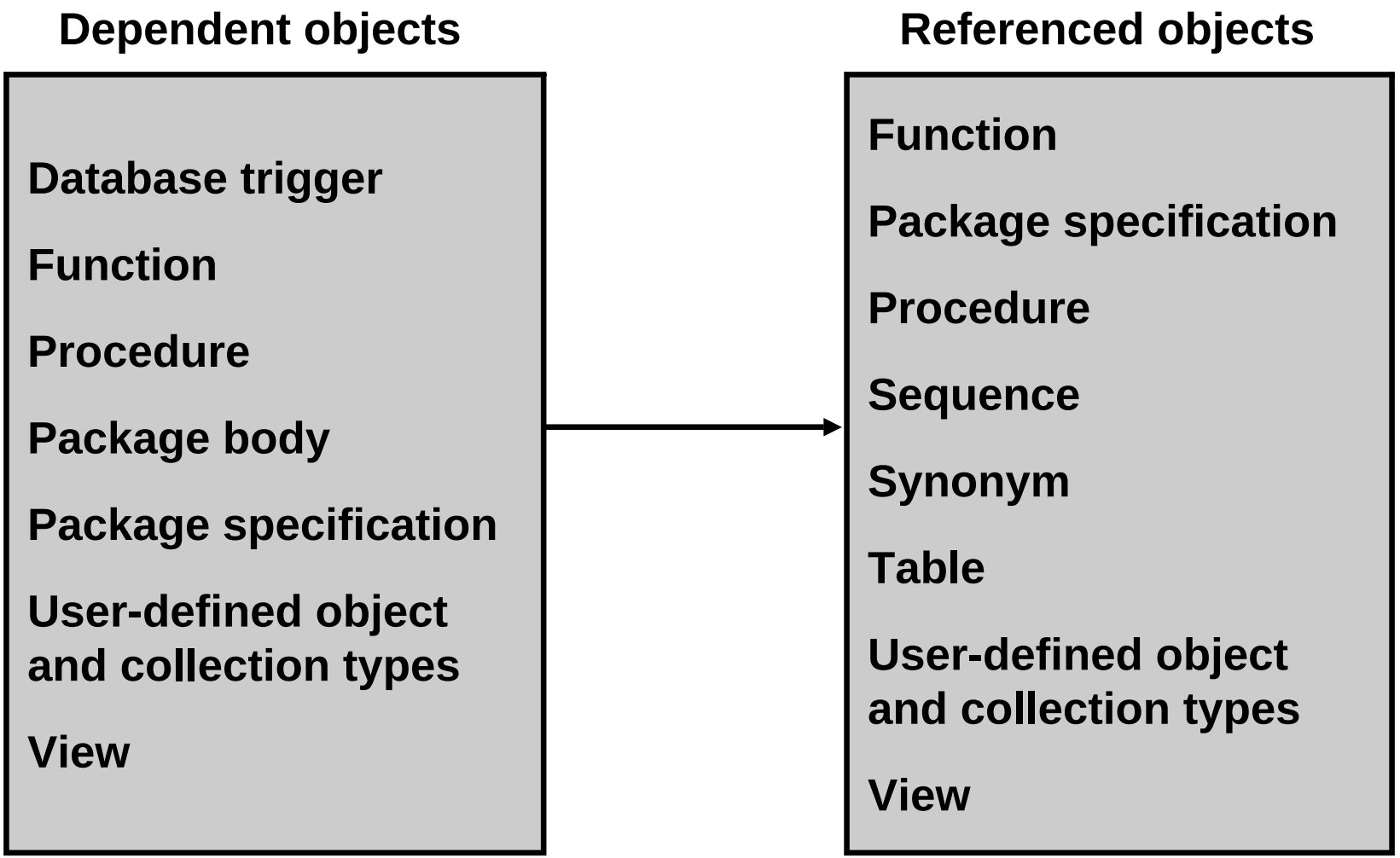
# Managing Dependencies

# Objectives

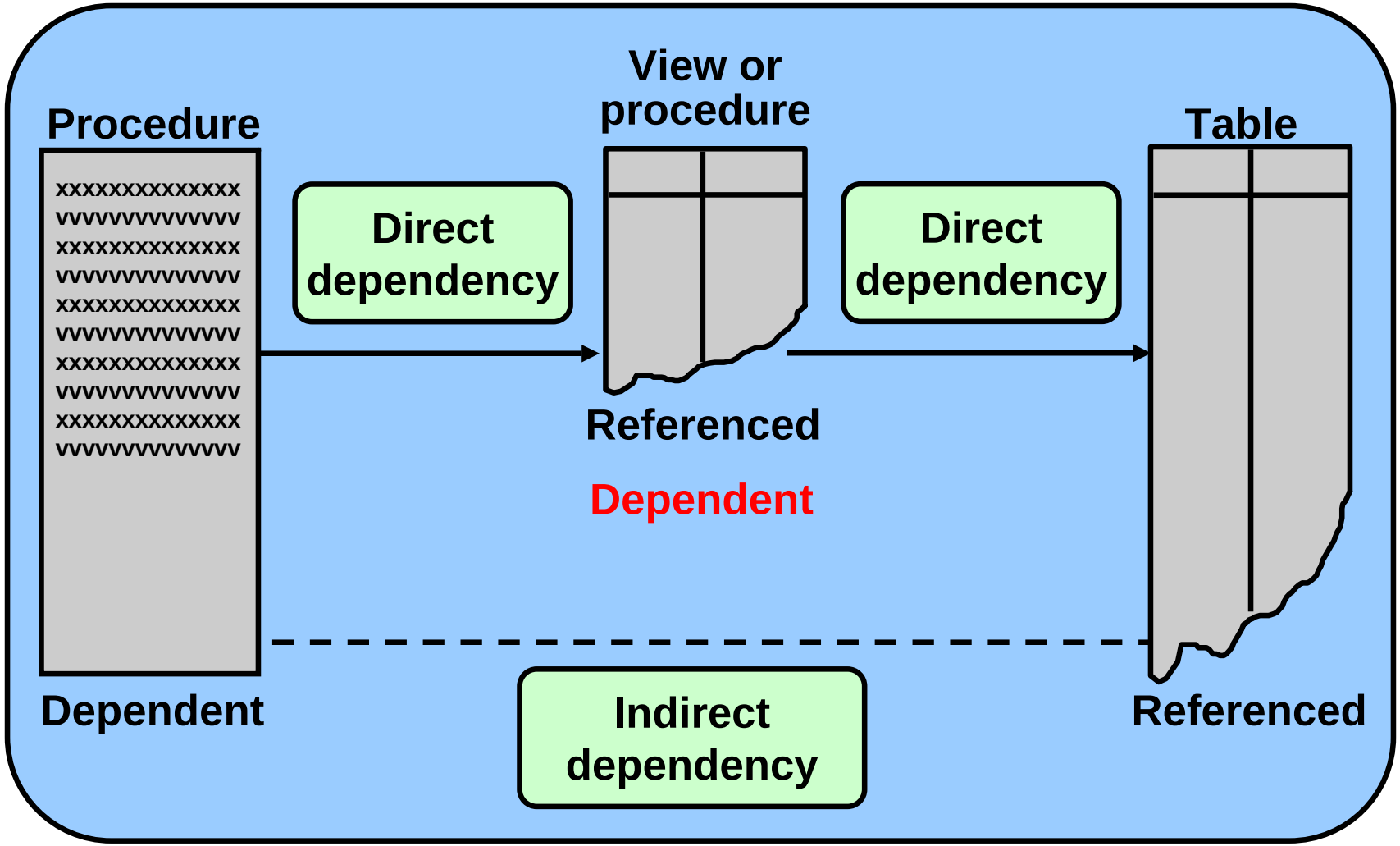
After completing this lesson, you should be able to do the following:

- Track procedural dependencies
- Predict the effect of changing a database object on stored procedures and functions
- Manage procedural dependencies

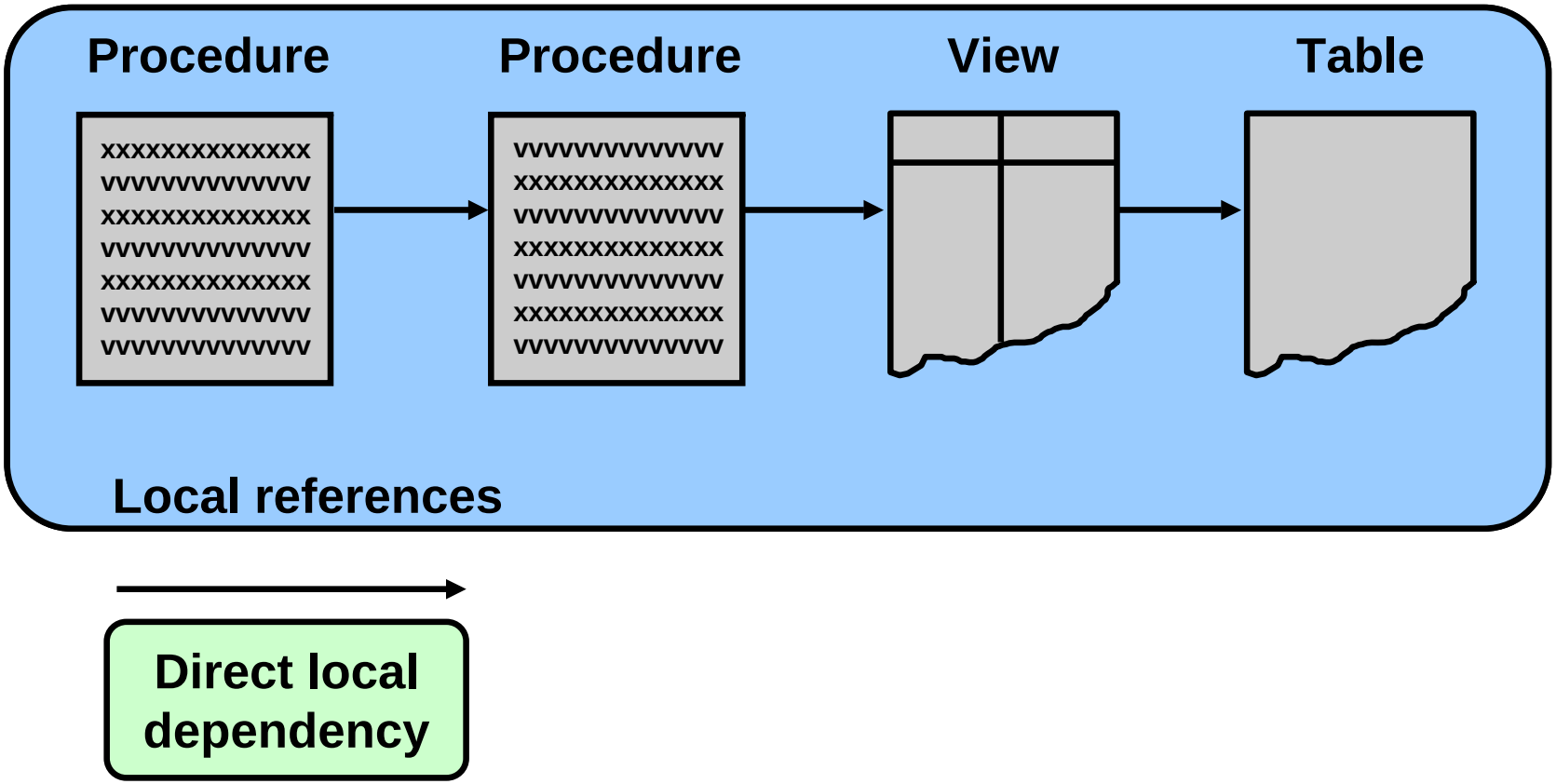
# Understanding Dependencies



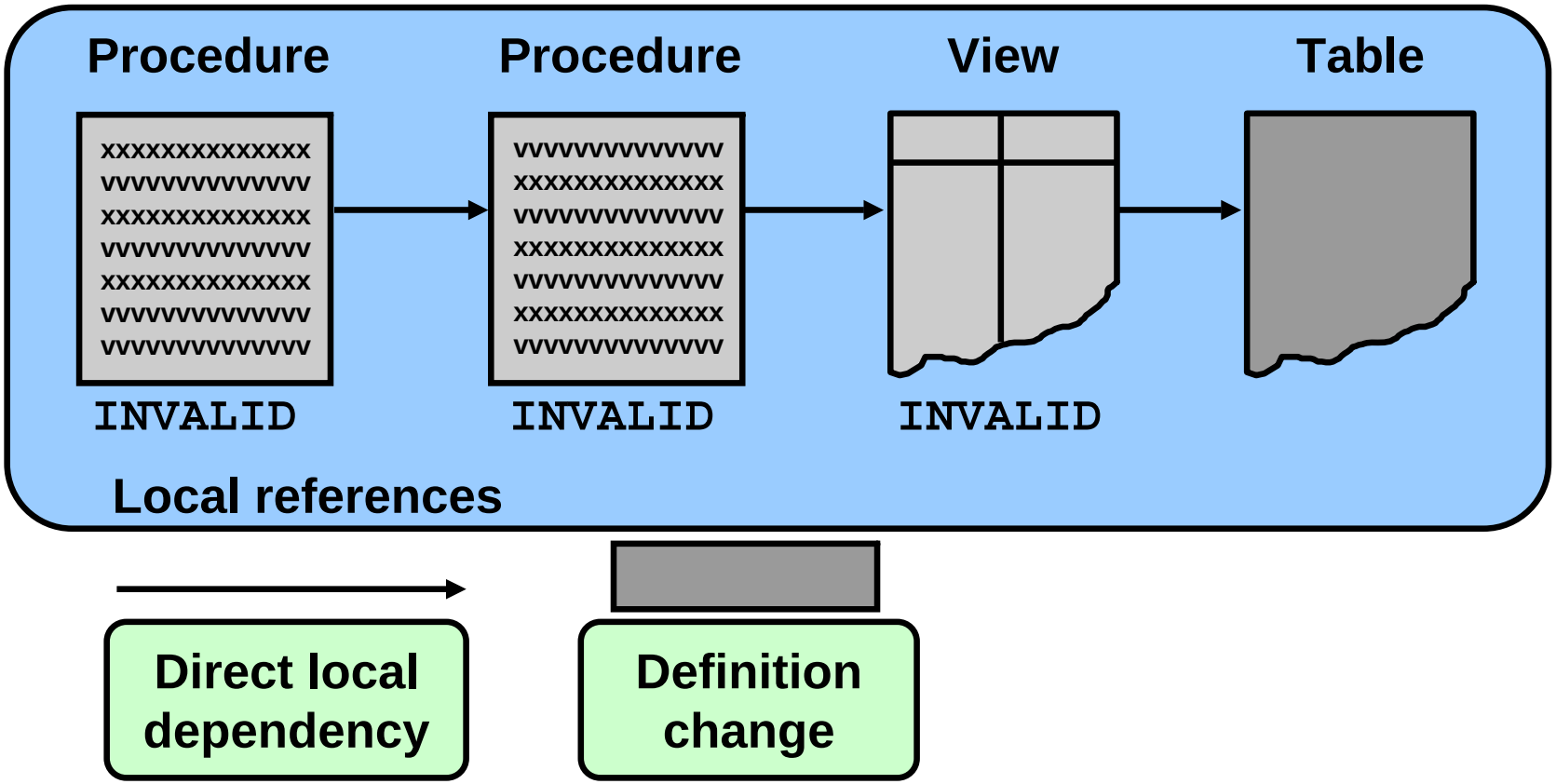
# Dependencies



# Local Dependencies



# Local Dependencies




The Oracle server implicitly recompiles any `INVALID` object when the object is next called.

# A Scenario of Local Dependencies

**ADD_EMP procedure**

xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv

**EMP_VW view**

| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME | EMAIL | DEPARTMENT_ID |
|---|---|---|---|---|---|
| 1 | 100 | King | Steven | SKING | 90 |
| 2 | 101 | Kochhar | Neena | NKOCHHAR | 90 |
| 3 | 102 | De Haan | Lex | LDEHAAN | 90 |
| 4 | 103 | Hunold | Alexander | AHUNOLD | 60 |
| 5 | 104 | Ernst | Bruce | BERNST | 60 |

...

**QUERY_EMP procedure**

xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv

**EMPLOYEES table**

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER |
|---|---|---|---|---|---|
| 1 | 100 | Steven | King | SKING | 515.123.4567 |
| 2 | 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 |
| 3 | 102 | Lex | De Haan | LDEHAAN | 515.123.4569 |
| 4 | 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 |
| 5 | 104 | Bruce | Ernst | BERNST | 590.423.4568 |

...

# Displaying Direct Dependencies by Using USER_DEPENDENCIES

```
SELECT name, type, referenced_name, referenced_type
FROM   user_dependencies
WHERE  referenced_name IN ('EMPLOYEES');
```

| | NAME | TYPE | REFERENCED_NAME | REFERENCED_TYPE |
|---|---|---|---|---|
| 1 | SAL_STATUS | PROCEDURE | EMPLOYEES | TABLE |
| 2 | WEB_EMP | PROCEDURE | EMPLOYEES | TABLE |
| 3 | EMPLOYEE_SAL | PROCEDURE | EMPLOYEES | TABLE |
| 4 | UPDATE_SALARY | PROCEDURE | EMPLOYEES | TABLE |
| 5 | RAISE_SALARY | PROCEDURE | EMPLOYEES | TABLE |
| 6 | EMP_DETAILS_VIEW | VIEW | EMPLOYEES | TABLE |
| 7 | SECURE_EMPLOYEES | TRIGGER | EMPLOYEES | TABLE |
| 8 | UPDATE_JOB_HISTORY | TRIGGER | EMPLOYEES | TABLE |
| 9 | EMP_PKG | PACKAGE | EMPLOYEES | TABLE |

# Displaying Direct and Indirect Dependencies

1. Run the `utldtree.sql` script that creates the objects that enable you to display the direct and indirect dependencies.
2. Execute the `DEPTREE_FILL` procedure.

```
EXECUTE deptree_fill('TABLE','SCOTT','EMPLOYEES')
```

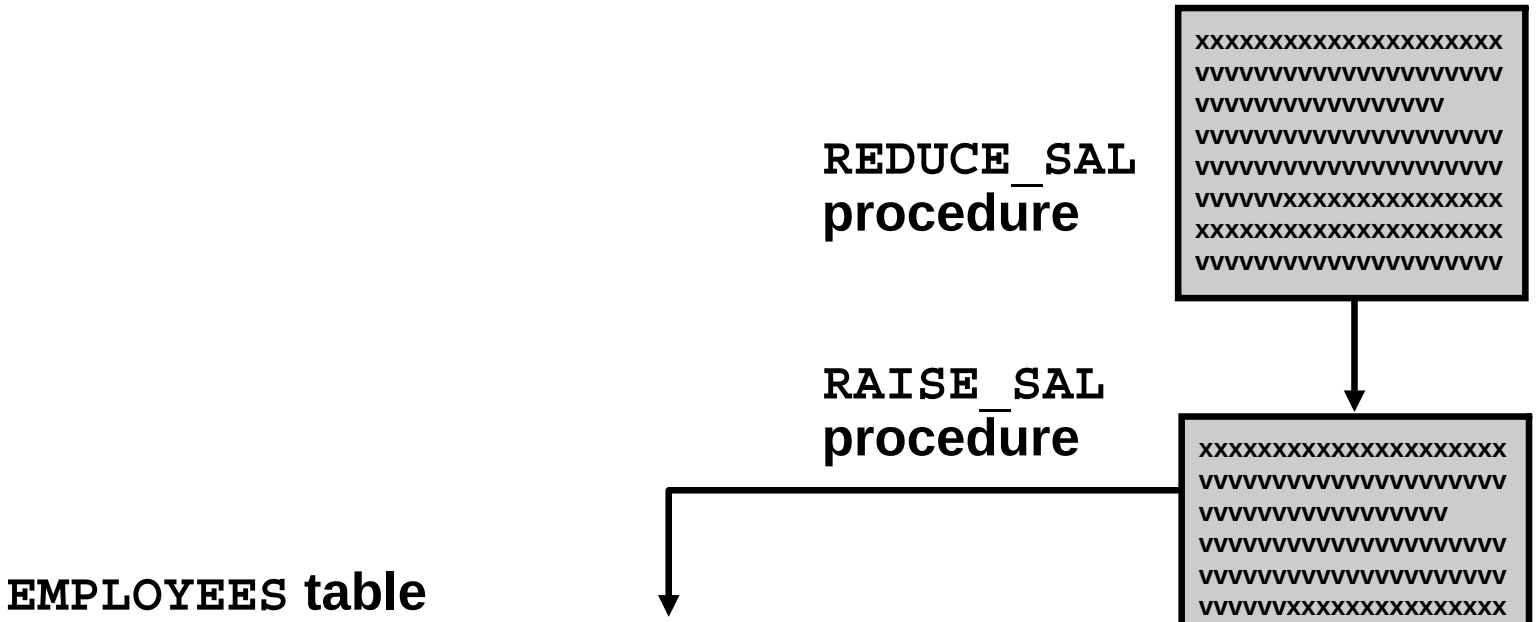
# Displaying Dependencies

The DEPTREE view:

```
SELECT   nested_level, type, name
FROM     deptree
ORDER BY seq#;
```

| | NESTED_LEVEL | TYPE | NAME |
|---|---|---|---|
| 1 | 0 | TABLE | EMPLOYEES |
| 2 | 1 | PROCEDURE | SAL_STATUS |
| 3 | 1 | PROCEDURE | WEB_EMP |
| 4 | 1 | PROCEDURE | EMPLOYEE_SAL |
| 5 | 1 | PROCEDURE | UPDATE_SALARY |
| 6 | 1 | PROCEDURE | RAISE_SALARY |
| 7 | 1 | VIEW | EMP_DETAILS_VIEW |
| 8 | 1 | TRIGGER | SECURE_EMPLOYEES |
| 9 | 1 | TRIGGER | UPDATE_JOB_HISTORY |
| 10 | 1 | PACKAGE | EMP_PKG |

# Another Scenario of Local Dependencies




| | EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 24000 |
| 2 | 101 | Kochhar | AD_VP | 18700 |
| 3 | 102 | De Haan | AD_VP | 18700 |
| 4 | 103 | Hunold | IT_PROG | 9900 |
| 5 | 104 | Ernst | IT_PROG | 6000 |

# A Scenario of Local Naming Dependencies

**QUERY_EMP procedure**

```
xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvvvvvvv
vvvvvvxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvvvv
```

**EMPLOYEES public synonym**

| | EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 24000 |
| 2 | 101 | Kochhar | AD_VP | 18700 |
| 3 | 102 | De Haan | AD_VP | 18700 |
| 4 | 103 | Hunold | IT_PROG | 9900 |
| 5 | 104 | Ernst | IT_PROG | 6000 |

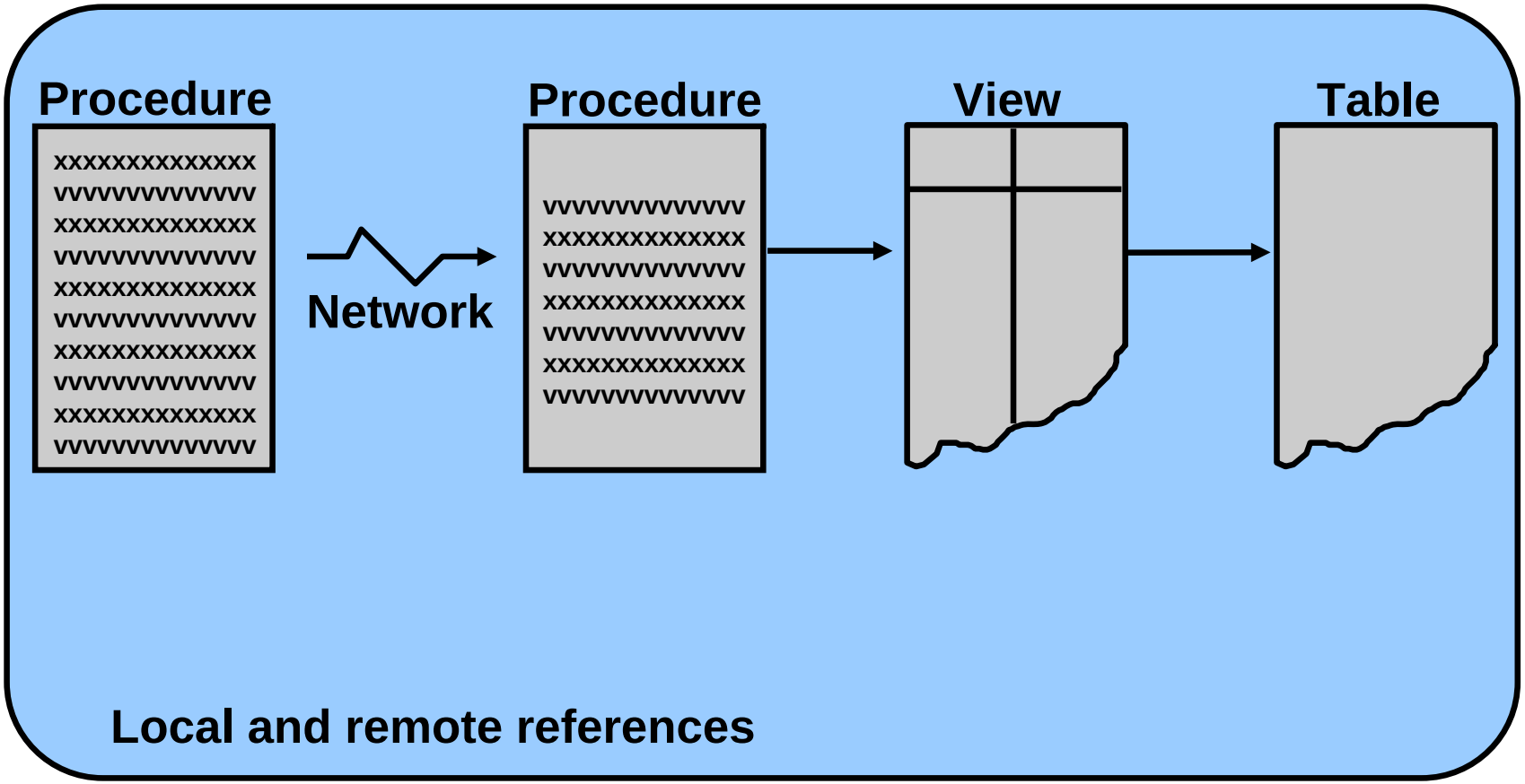
...

**EMPLOYEES table**

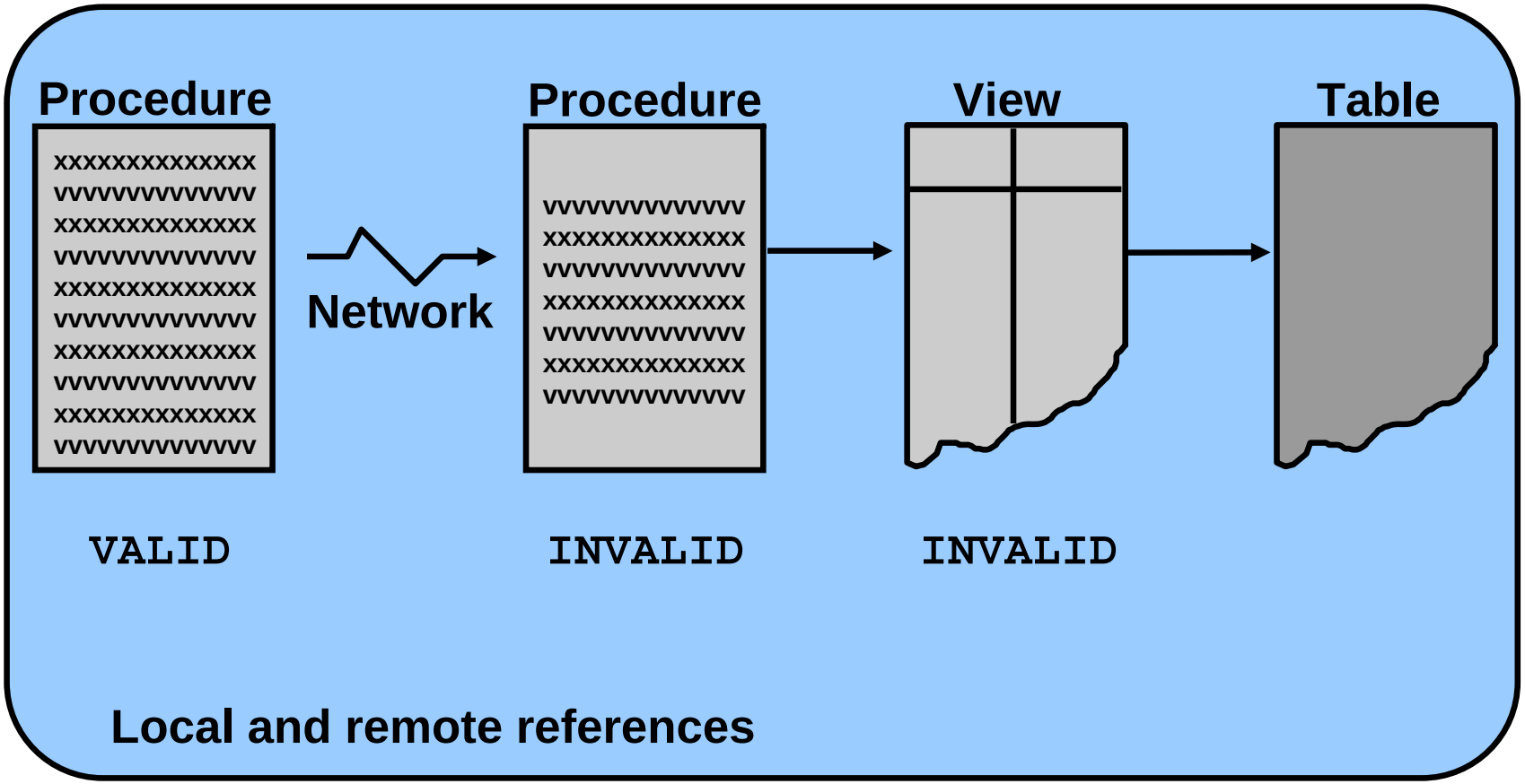
| | EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 24000 |
| 2 | 101 | Kochhar | AD_VP | 18700 |
| 3 | 102 | De Haan | AD_VP | 18700 |
| 4 | 103 | Hunold | IT_PROG | 9900 |
| 5 | 104 | Ernst | IT_PROG | 6000 |

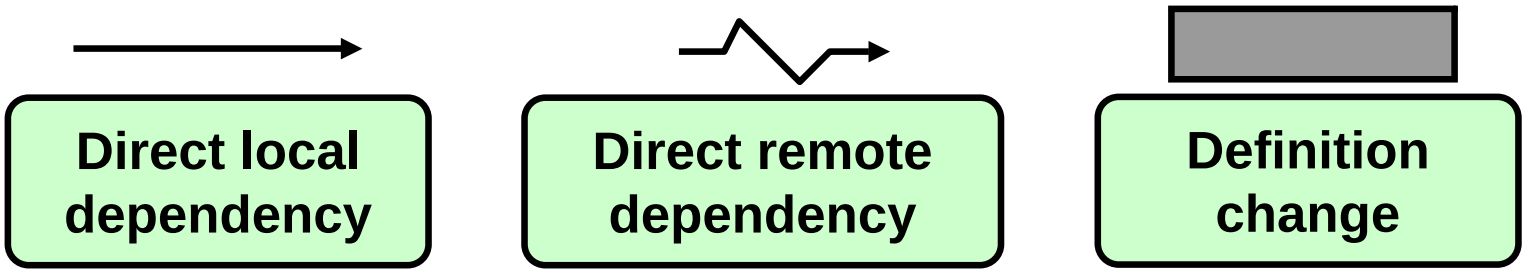
...

# Understanding Remote Dependencies



# Understanding Remote Dependencies



# Concepts of Remote Dependencies

Remote dependencies are governed by the mode that is chosen by the user:

- `TIMESTAMP` checking
- `SIGNATURE` checking

# REMOTE_DEPENDENCIES_MODE Parameter

Setting REMOTE_DEPENDENCIES_MODE:

- As an init.ora parameter

```
REMOTE_DEPENDENCIES_MODE = value
```

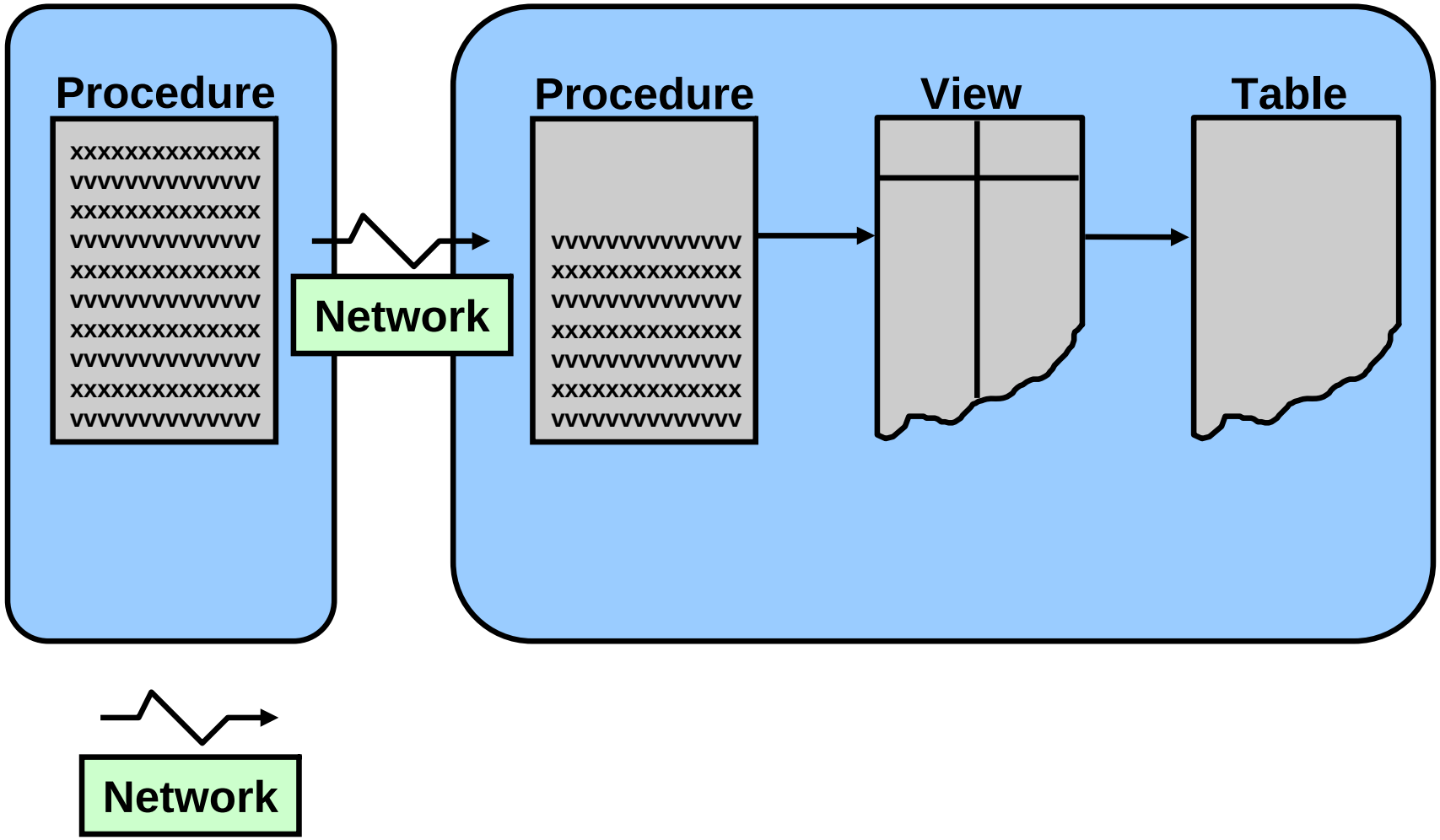
- At the system level

```
ALTER SYSTEM SET
REMOTE_DEPENDENCIES_MODE = value
```

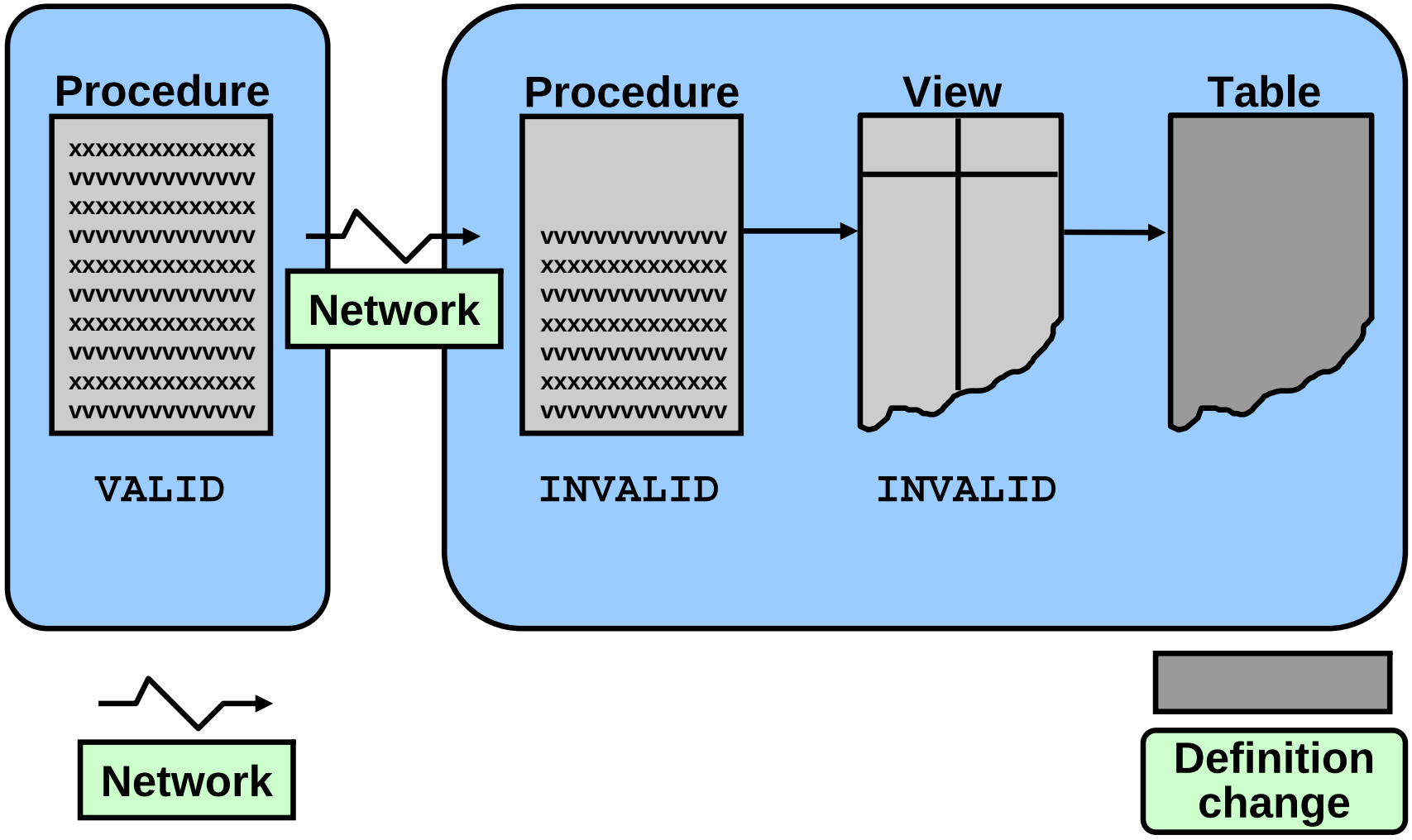
- At the session level

```
ALTER SESSION SET
REMOTE_DEPENDENCIES_MODE = value
```

# Remote Dependencies and Time Stamp Mode

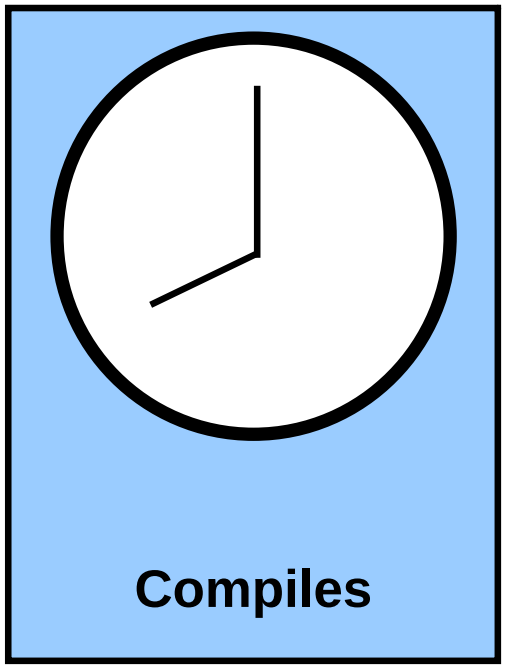


# Remote Dependencies and Time Stamp Mode



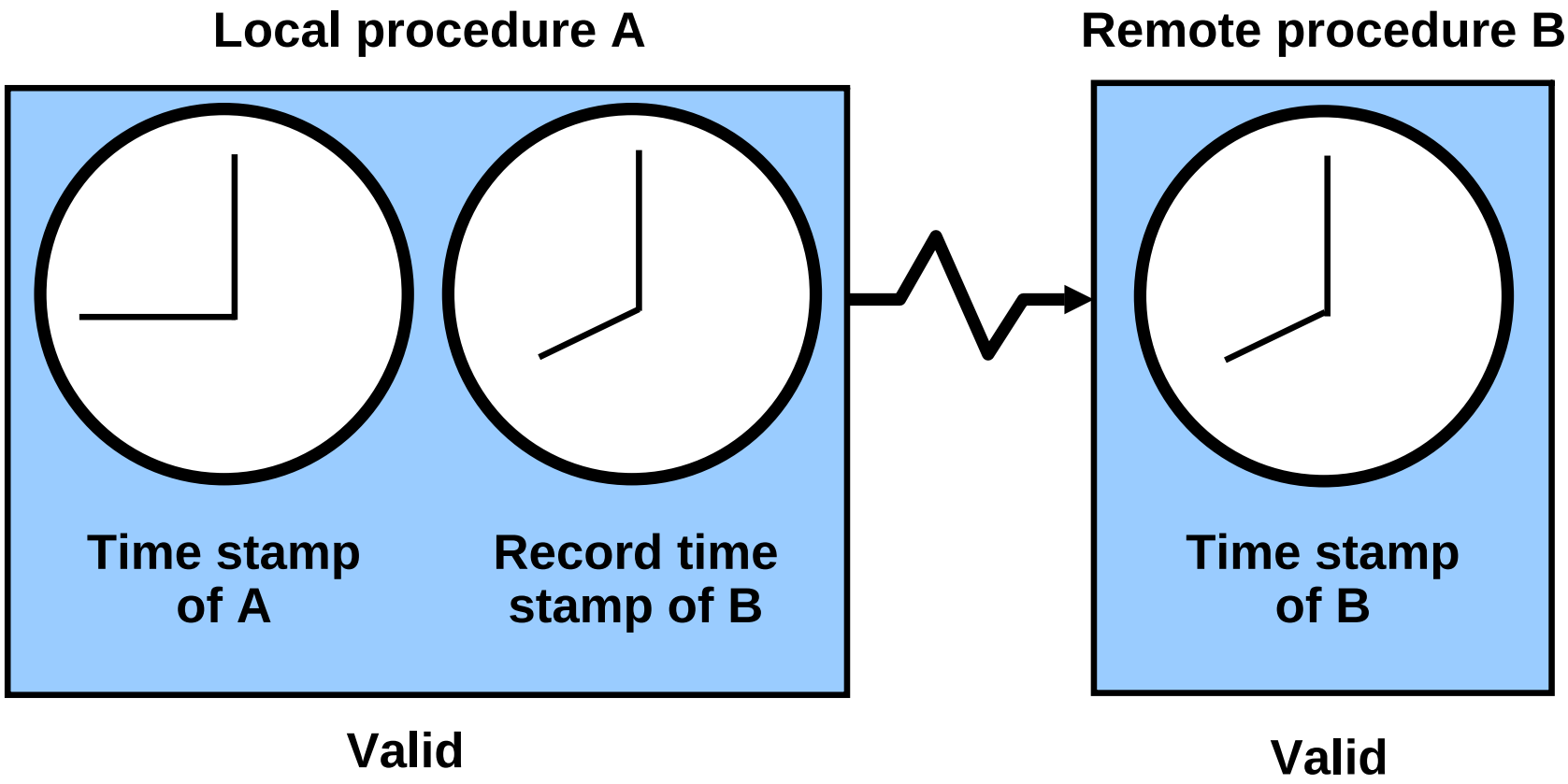
# Remote Procedure B Compiles at 8:00 AM

**Remote procedure B**

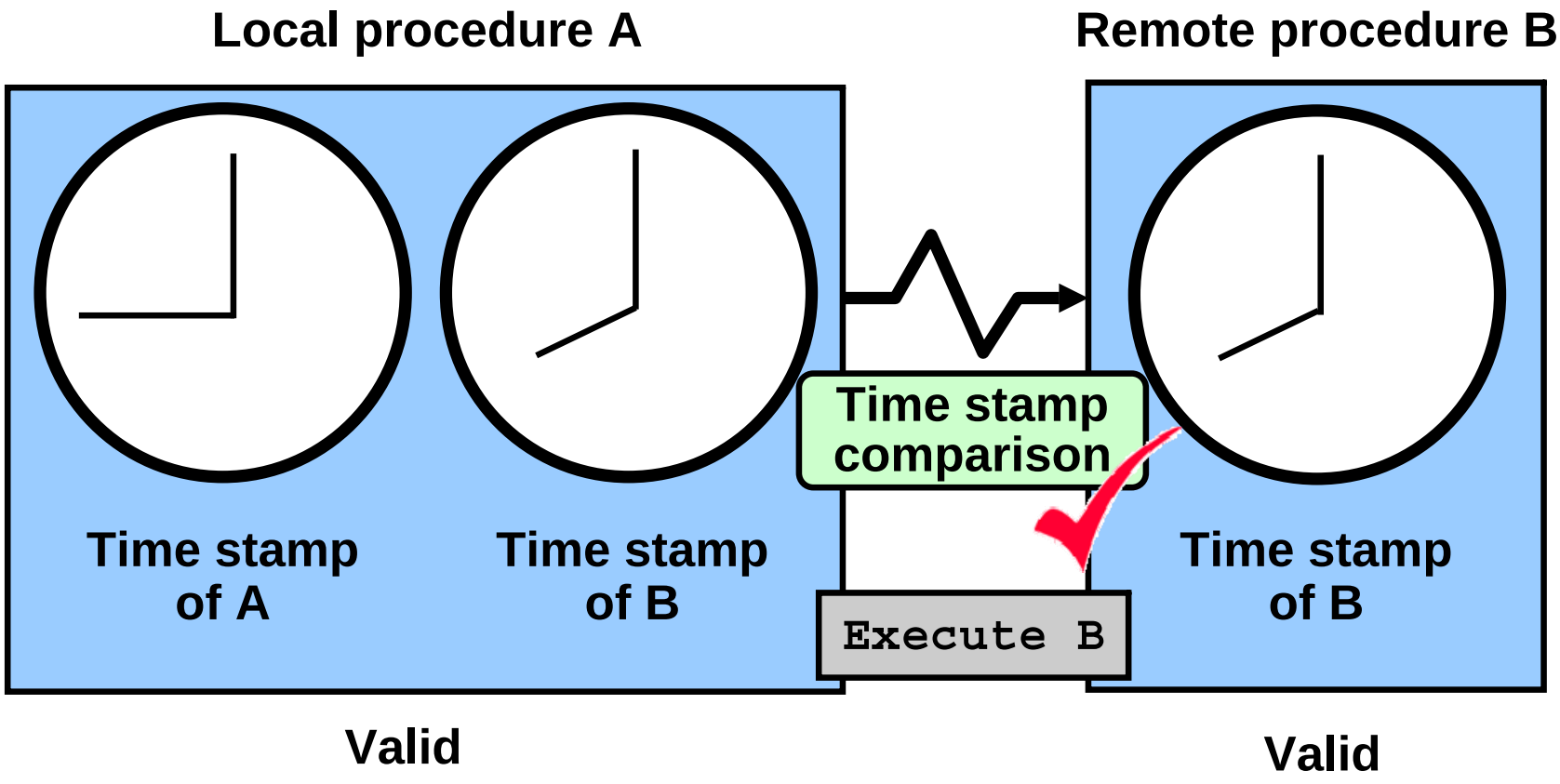



**Valid**

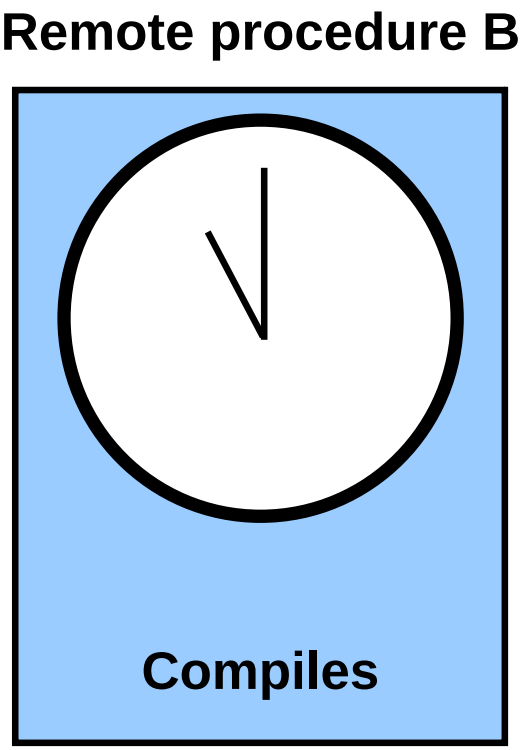
# Local Procedure A Compiles at 9:00 AM



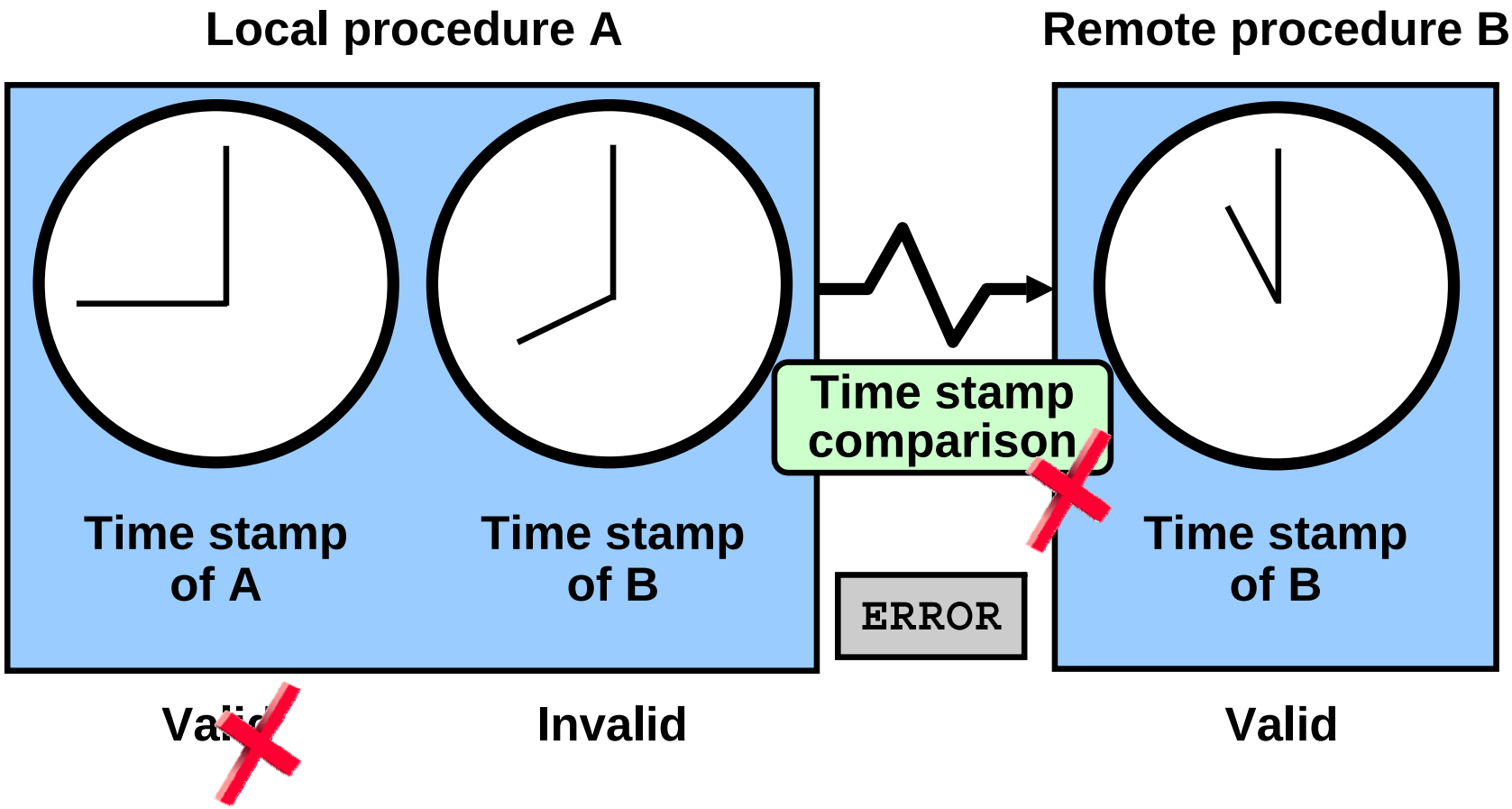
# Execute Procedure A



# Remote Procedure B Recompiled at 11:00 AM



# Execute Procedure A



# Signature Mode

- The signature of a procedure is the:
  - Name of the procedure
  - Data types of the parameters
  - Modes of the parameters
- The signature of the remote procedure is saved in the local procedure.
- When executing a dependent procedure, the signature of the referenced remote procedure is compared.

# Recompiling a PL/SQL Program Unit

Recompilation is handled:

- Automatically through implicit run-time recompilation
- Through explicit recompilation with the `ALTER` statement

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION  [SCHEMA.]function_name  COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name
   COMPILE [PACKAGE | SPECIFICATION | BODY];
```

```
ALTER TRIGGER trigger_name [COMPILE[DEBUG]];
```

# Unsuccessful Recompilation

Recompiling dependent procedures and functions is unsuccessful when:

- The referenced object is dropped or renamed
- The data type of the referenced column is changed
- The referenced column is dropped
- A referenced view is replaced by a view with different columns
- The parameter list of a referenced procedure is modified

# Successful Recompilation

Recompiling dependent procedures and functions is successful if:

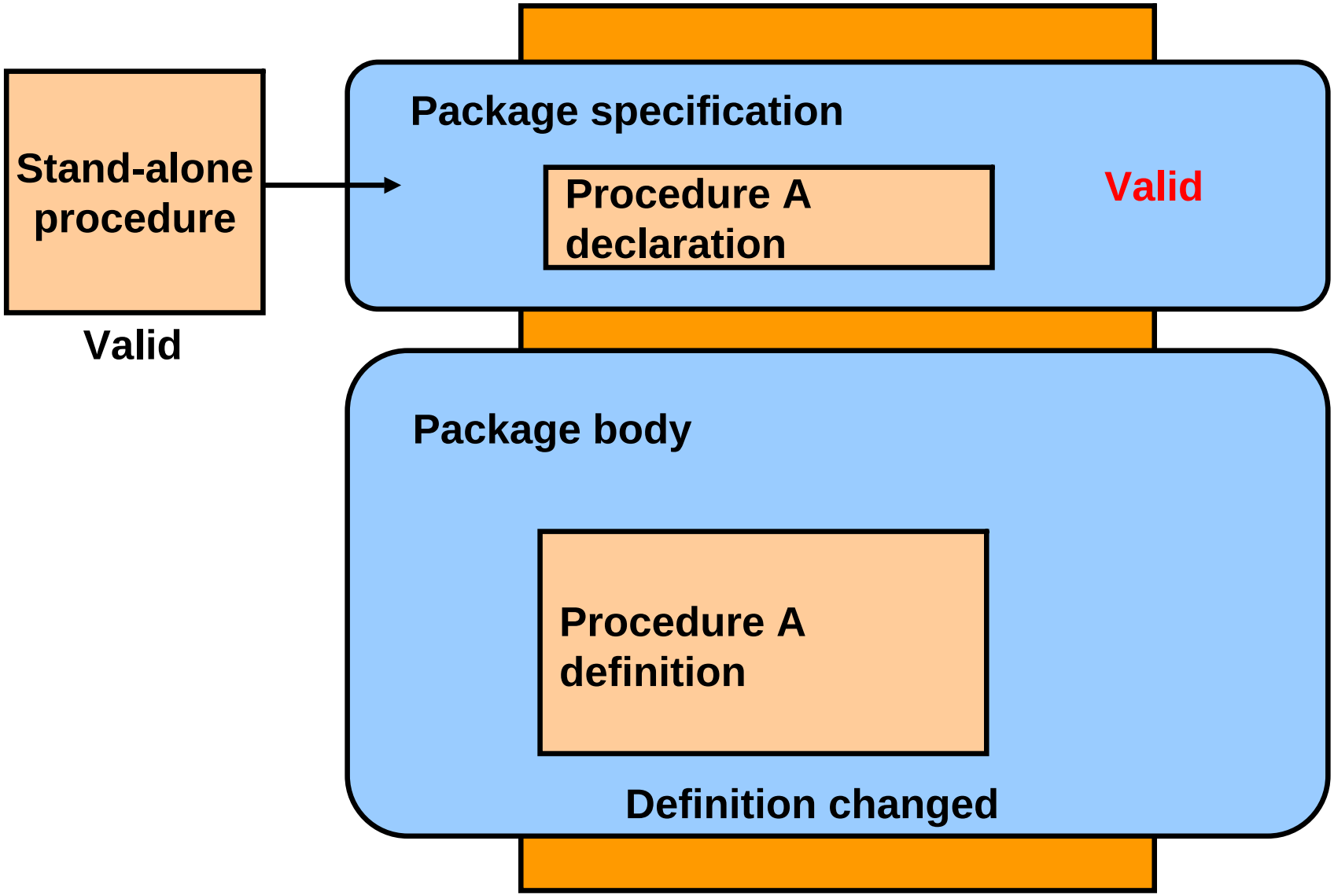
- The referenced table has new columns
- The data type of referenced columns has not changed
- A private table is dropped, but a public table that has the same name and structure exists
- The PL/SQL body of a referenced procedure has been modified and recompiled successfully

# Recompilation of Procedures

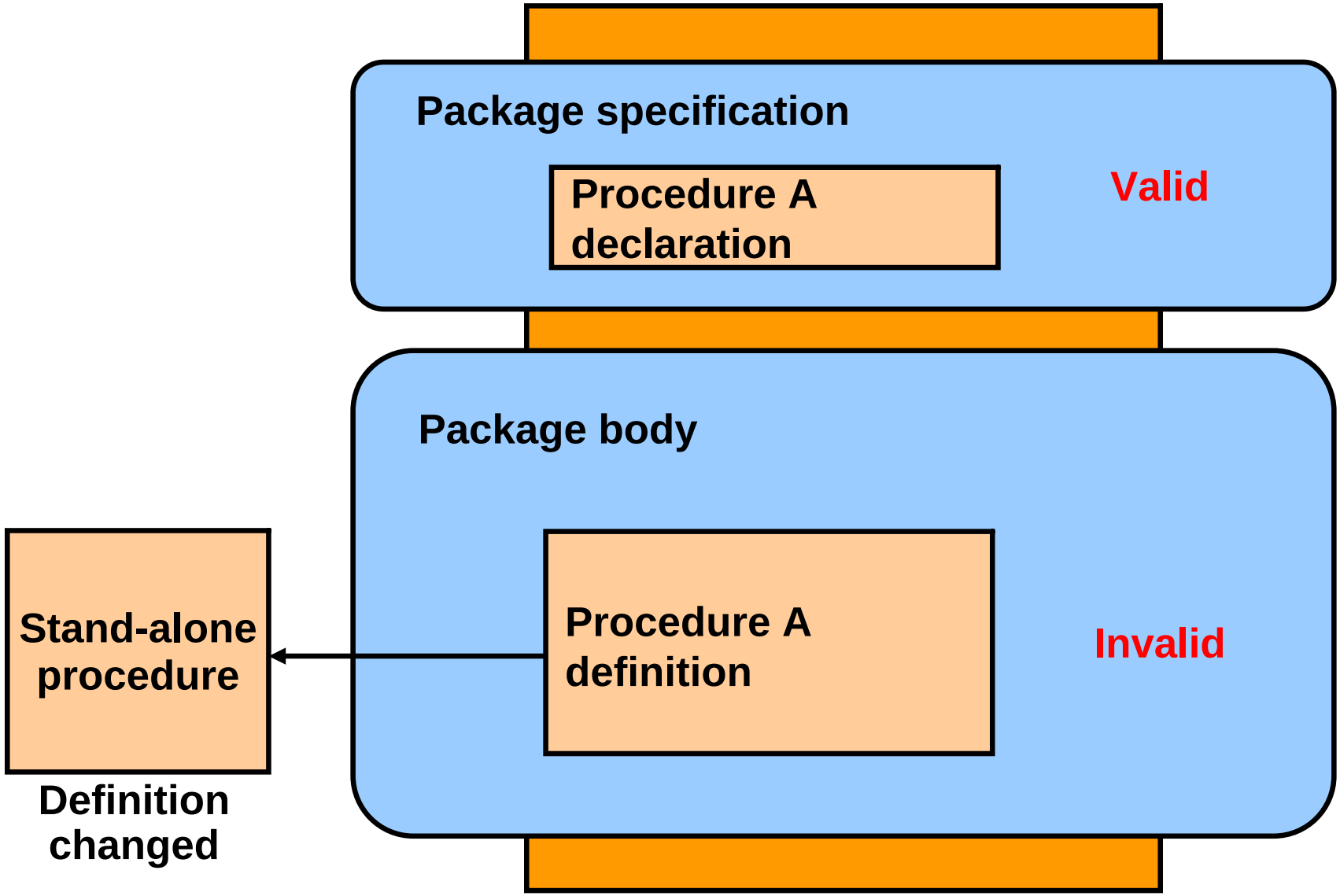
Minimize dependency failures by:

- Declaring records with the `%ROWTYPE` attribute
- Declaring variables with the `%TYPE` attribute
- Querying with the `SELECT *` notation
- Including a column list with `INSERT` statements

# Packages and Dependencies



# Packages and Dependencies



# Summary

In this lesson, you should have learned how to:

- Keep track of dependent procedures
- Recompile procedures manually as soon as possible after the definition of a database object changes

# Practice 8: Overview

This practice covers the following topics:

- Using `DEPTREE_FILL` and `IDEPTREE` to view dependencies
- Recompiling procedures, functions, and packages