Oracle8i: Managing Data

Electronic Presentation

30062GC10
Production 1.0
February 2000
M09980

ORACLE

Authors

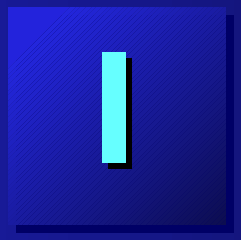
Bruce A. Ernst
Tigger Newman
Nagavalli Pataballa

Technical Contributors and Reviewers

David Austin
David J. Bridges
E. Leta Davis
Ruth Delaney
Ric Van Dyke
Dan Gabel
Usha George
Scott Gossett
Rosita Hanoman
Craig Hollister
Taj-ul Islam
Tamas Kereps
Stefan Lindblad
Bram van der Vos
Jean-Francois Verrier

Publisher

Nita Brozowski



Introduction

Course Objectives

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

- Identify major components of Oracle architecture
- Design a schema
- Implement data storage structures
- Enforce business rules using constraints
- Load and extract data

Course Overview

Oracle Architecture

- **Overview of Oracle Application Architecture**
- **SQL Statement Processing**
- **Transactions**

Course Overview

Designing a Schema

- **Designing the Physical Structure**
- **Organizing Storage**
- **Controlling Use of Block Space**
- **Selecting an Index Type**
- **Implementing Index-Organized Tables**
- **Implementing Clusters**

Course Overview

Enforcing Business Rules

- **Enforcing Business Rules with Constraints**

Loading and Extracting Data

- **Transporting Data**
- **Reorganizing Data**

Developer Tasks

- Design the physical structure of the application
- Design and develop the database application
- Write SQL code
- Enforce security in the application
- Tune the application
- Maintain and update the application

1

Overview of Oracle Application Architecture

ORACLE

Objectives

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

- Differentiate between database architectures
- Differentiate between Oracle database configurations
- List Oracle application components

Benefits of Relational Databases

- Independence of physical data storage and logical database structure
- Variable and easy access to all data
- Flexibility in database design
- Reduced data storage and redundancy

Object Technology

- Object technology is another step forward in the software engineering life cycle that enables you to:
  - Model real things
  - Represent real things as objects
- Object technology is a method of managing complexity so that you can develop solutions that simulate real-world problems.

What Is an Object-Relational Database?

- Allows creation of user-defined data types
- Supports multimedia and large data objects
- Includes compatibility with object-oriented features of the ANSI SQL:1999 standard
- Includes the benefits of relational databases

Benefits of Object Technology

- Easy maintenance of applications
- Faster product development
- Higher quality of code at a lower cost
- Reusability of code

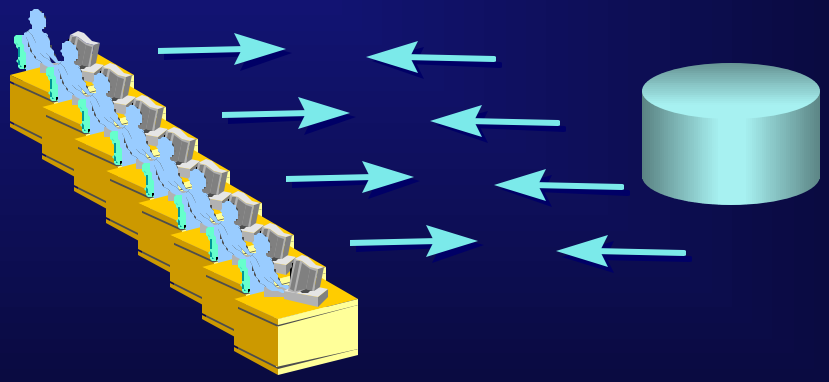
Application Architectures

Data design and logical database structure should cater to the following application types:

- Online transaction processing (OLTP)
- Decision support systems (DSS)
- Hybrid systems that involve both types of applications

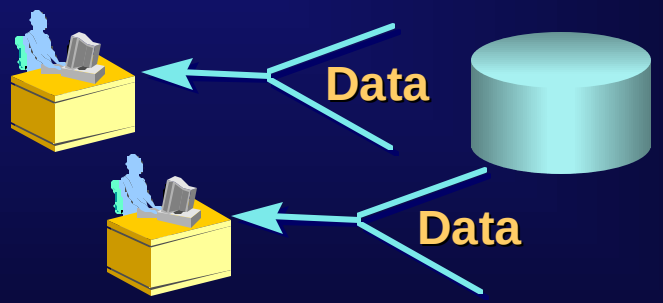
OLTP Applications

- Are high-throughput, I/O-intensive systems
- Contain large volumes of data that:
  - Grow continuously
  - Are accessed concurrently by hundreds of users
- Tuning goals:
  - Speed
  - Concurrency
  - Availability
  - Recoverability



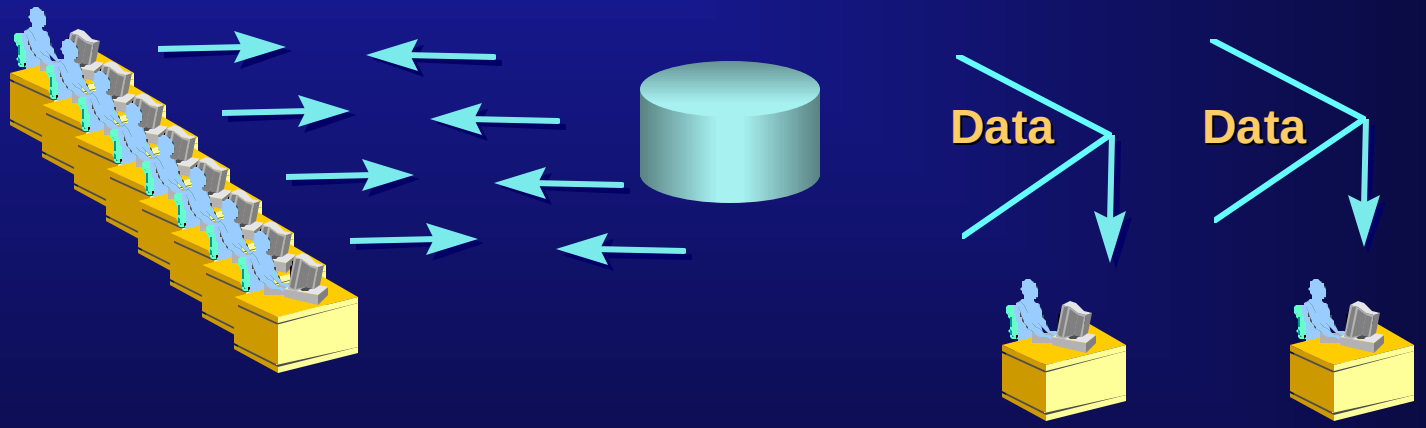
Decision Support Systems

- DSS performs queries on large amounts of data.
- DSS makes heavy use of full table scans.
- The tuning goals of DSS are:
  - Response time
  - Accuracy
  - Availability
- Parallel query is designed particularly for DSS.



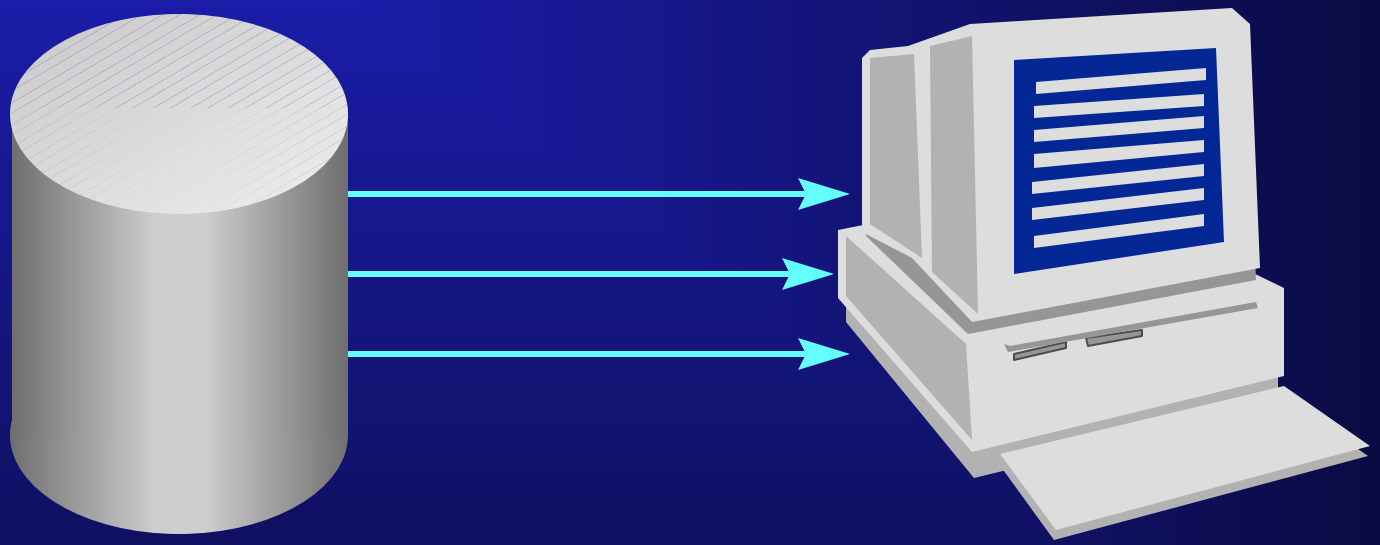
Hybrid Applications

- Are a combination of OLTP and DSS
- Use both online and batch processing



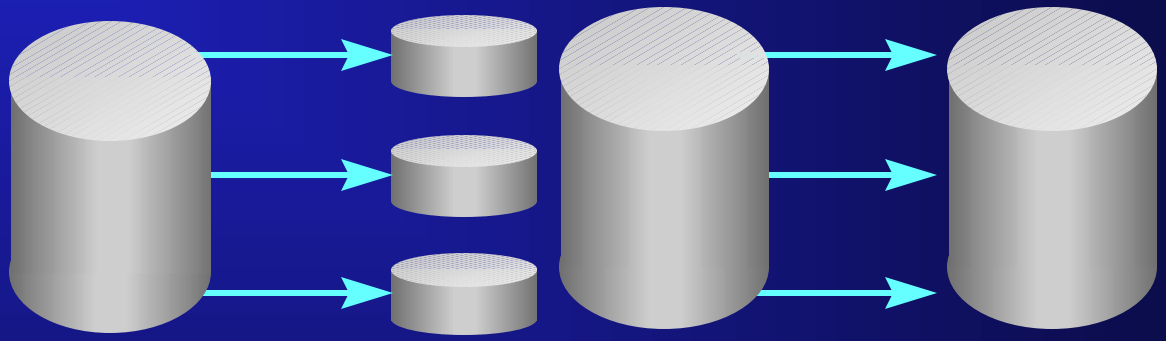
Parallel Query

SELECT ...

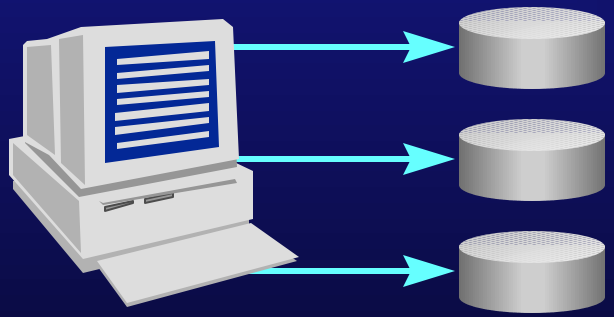


Parallel DML

INSERT ... SELECT



Parallel UPDATE and DELETE



Database Configurations

Different database configurations can be used, depending on the requirements:

- Dedicated server
- Multithreaded server
- Distributed databases
- Oracle Parallel Server

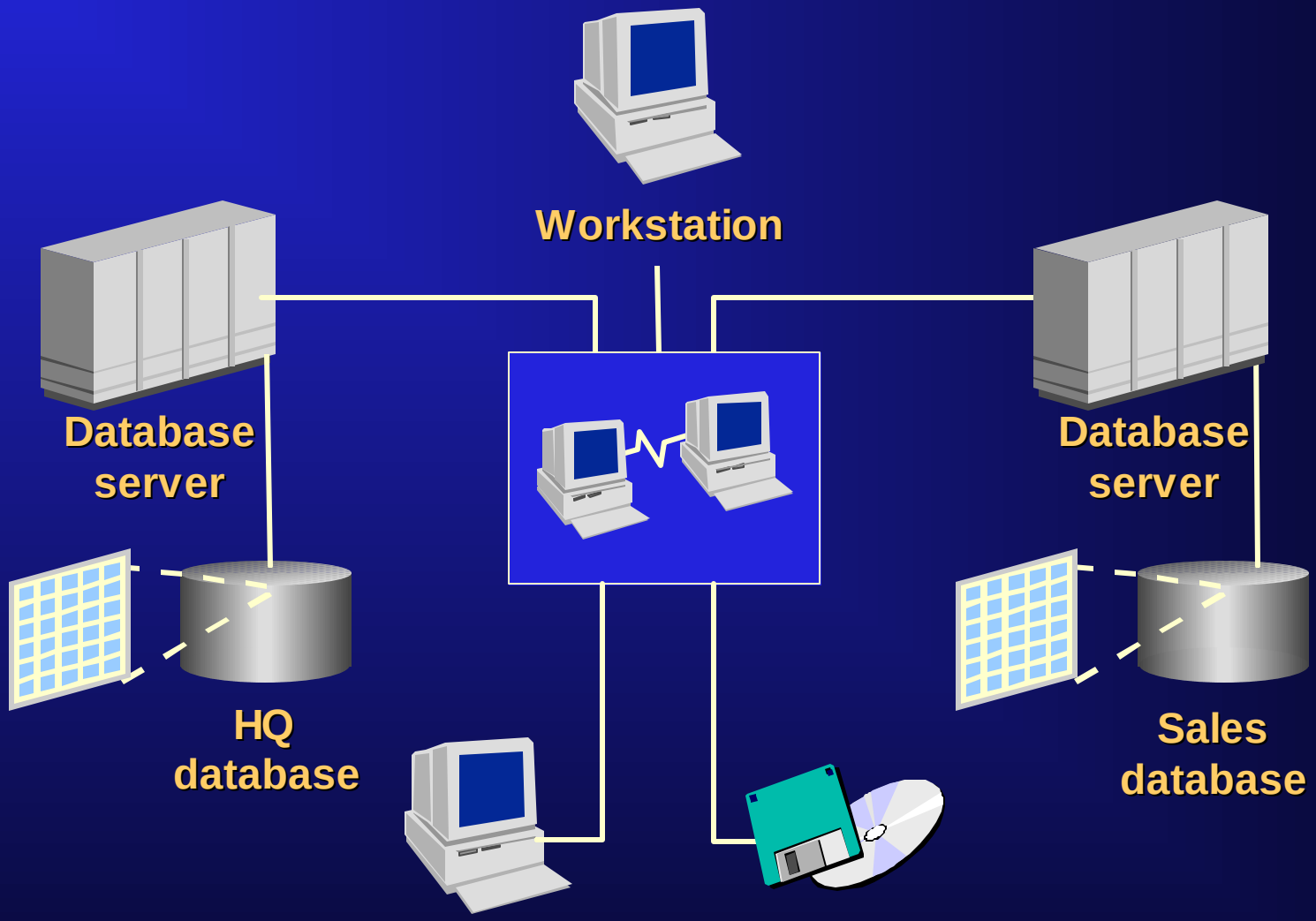
Dedicated Server

- Each dedicated server process handles requests for a single user process.
- The dedicated server process remains idle when the user is not actively making a database request.

Multithreaded Server

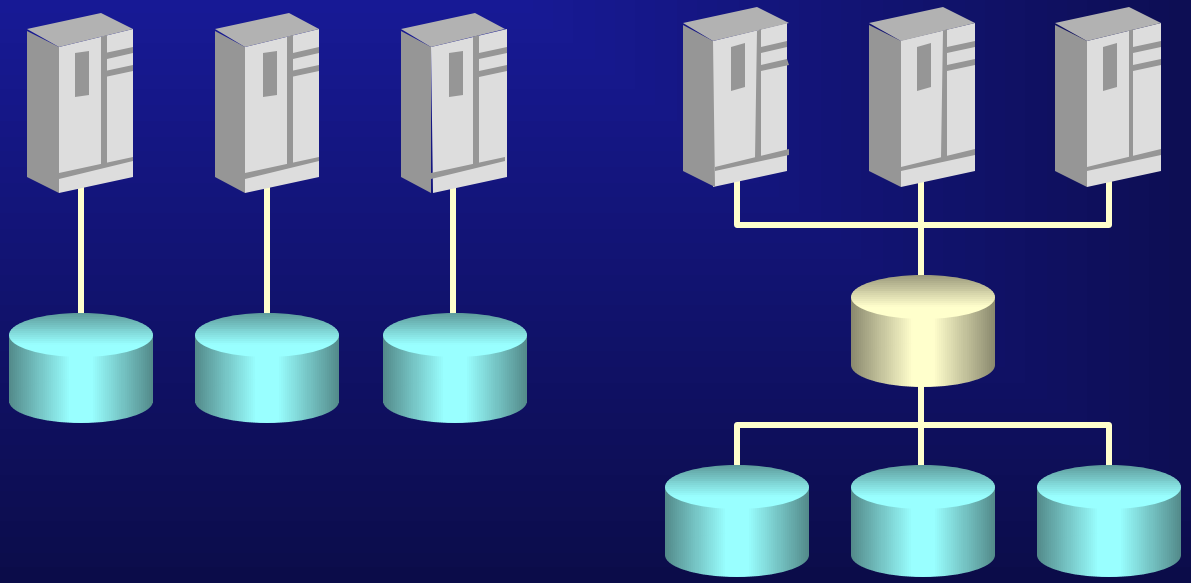
- Increases the number of possible users
- Achieves load balancing
- Reduces the number of processes against instance
- Reduces the number of idle server processes
- Reduces memory usage and system overheadwith a large number of connections

Distributed Databases



Oracle Parallel Server Technology

You can consolidate multiple databases into a single database when you use Oracle8*i* Parallel Server technology.

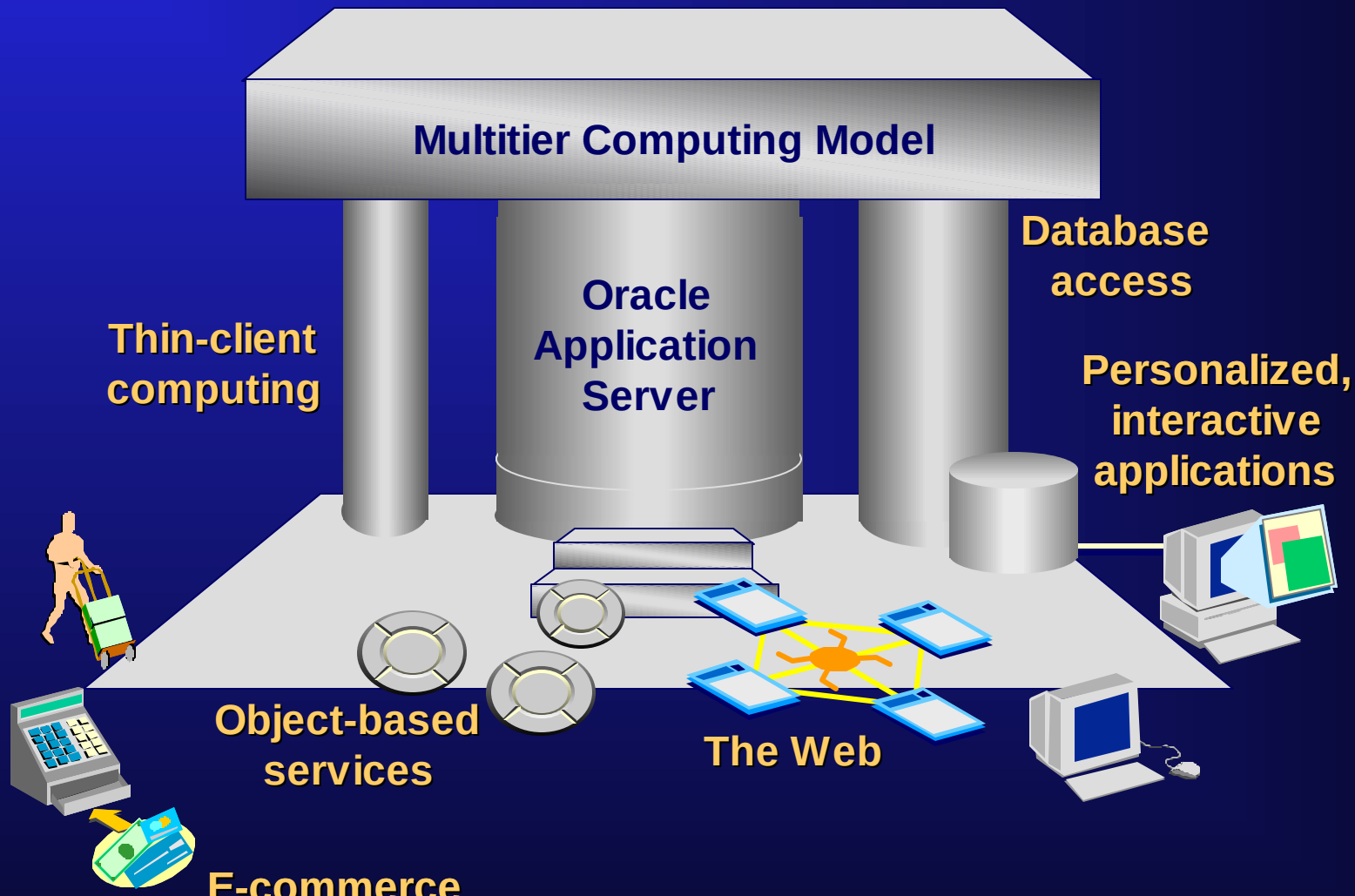


Oracle Application Components

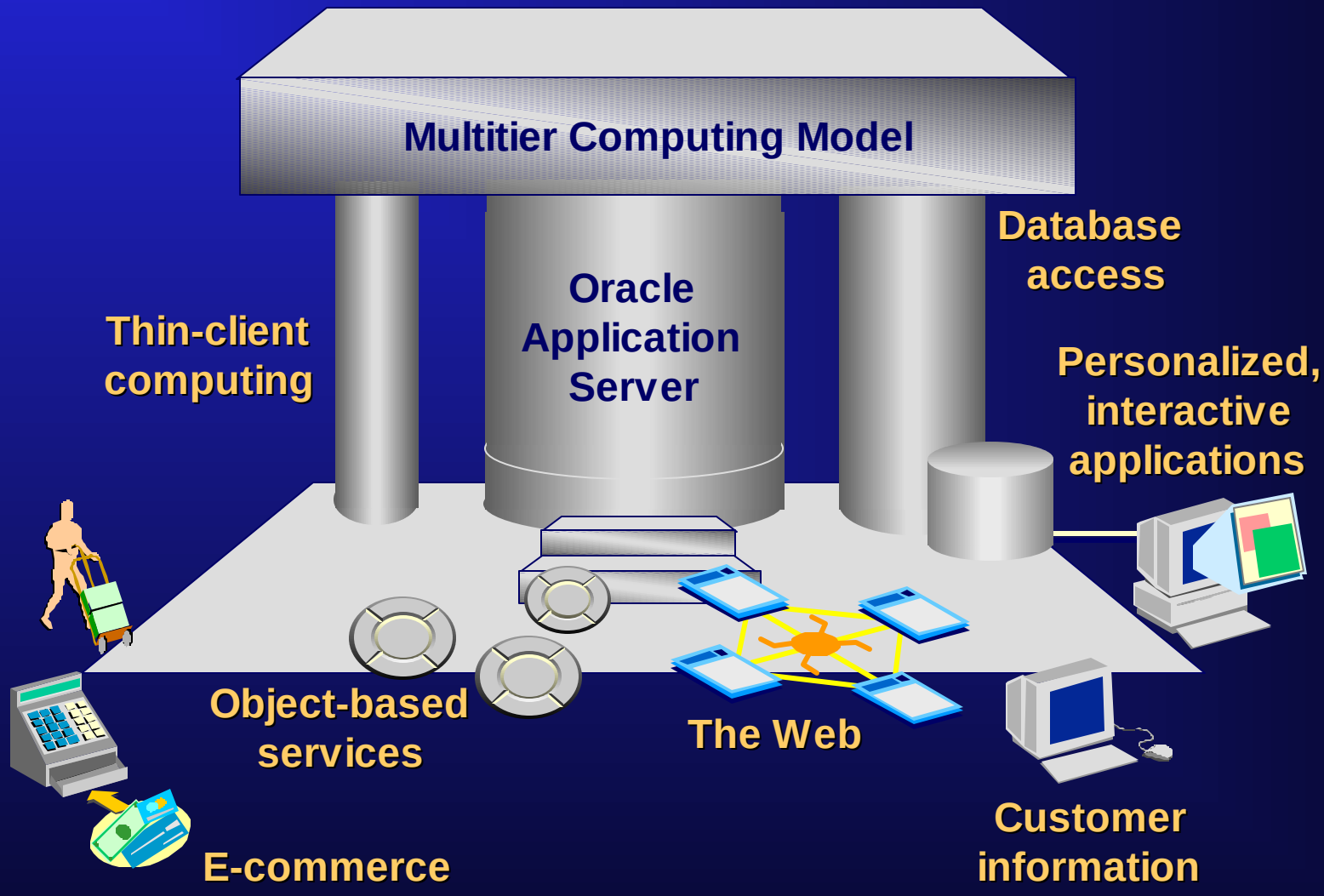
Oracle application components include:

- Oracle Application Server (OAS)
- Programmatic environments:
  - Application programming interfaces (API)
  - Tools

Multitier Computing Model



Multitier Computing Model



Role of the Application Server

OAS provides:

- A robust and scaleable platform for applications
- Web-enabled databases
- Web-based administration tools
- Services, such as security and transactions

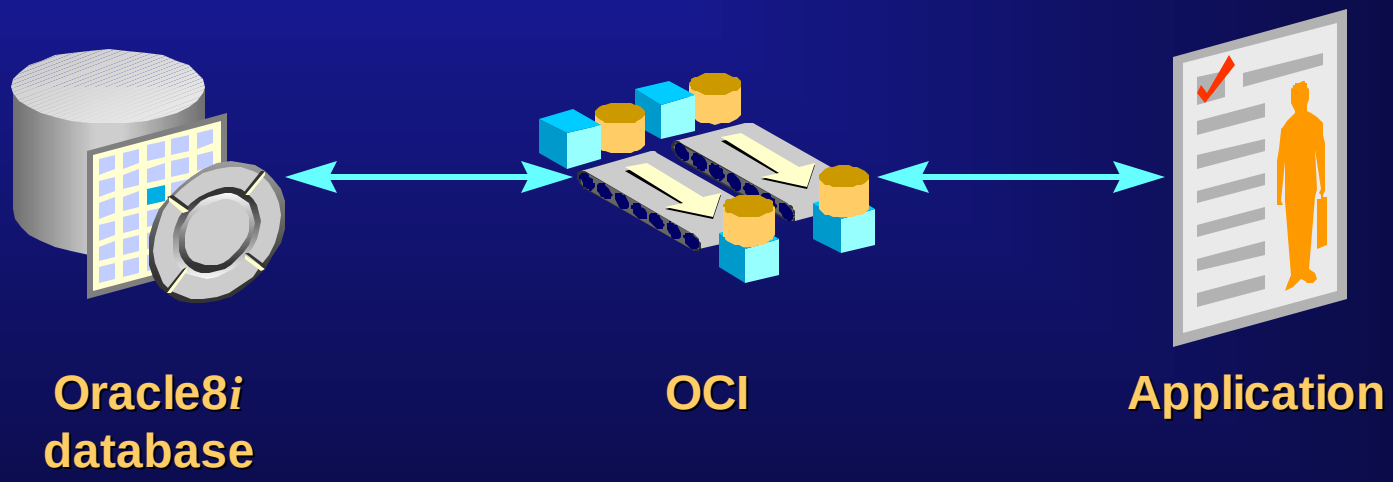
Oracle APIs

Application programming interfaces:

- ODBC
- JDBC
- 3GL precompiler: Pro*C/C++, Pro*COBOL
- Oracle Call Interface (OCI)
- SQLJ
- Oracle Objects for OLE (OO4O)

What Is OCI?

A call-level interface to the Oracle database allowing application development with 3GLs



Oracle Tools

- Oracle Designer
- Oracle Developer
  - Oracle Forms
  - Oracle Reports
- JDeveloper

Summary

In this lesson, you should have learned that an Oracle8*i* application developer can:

- Develop applications to be deployed in a variety of database architectures and configurations
- Create applications with a variety of Oracle application components
- Use several APIs and tools for application development

2

SQL Statement Processing

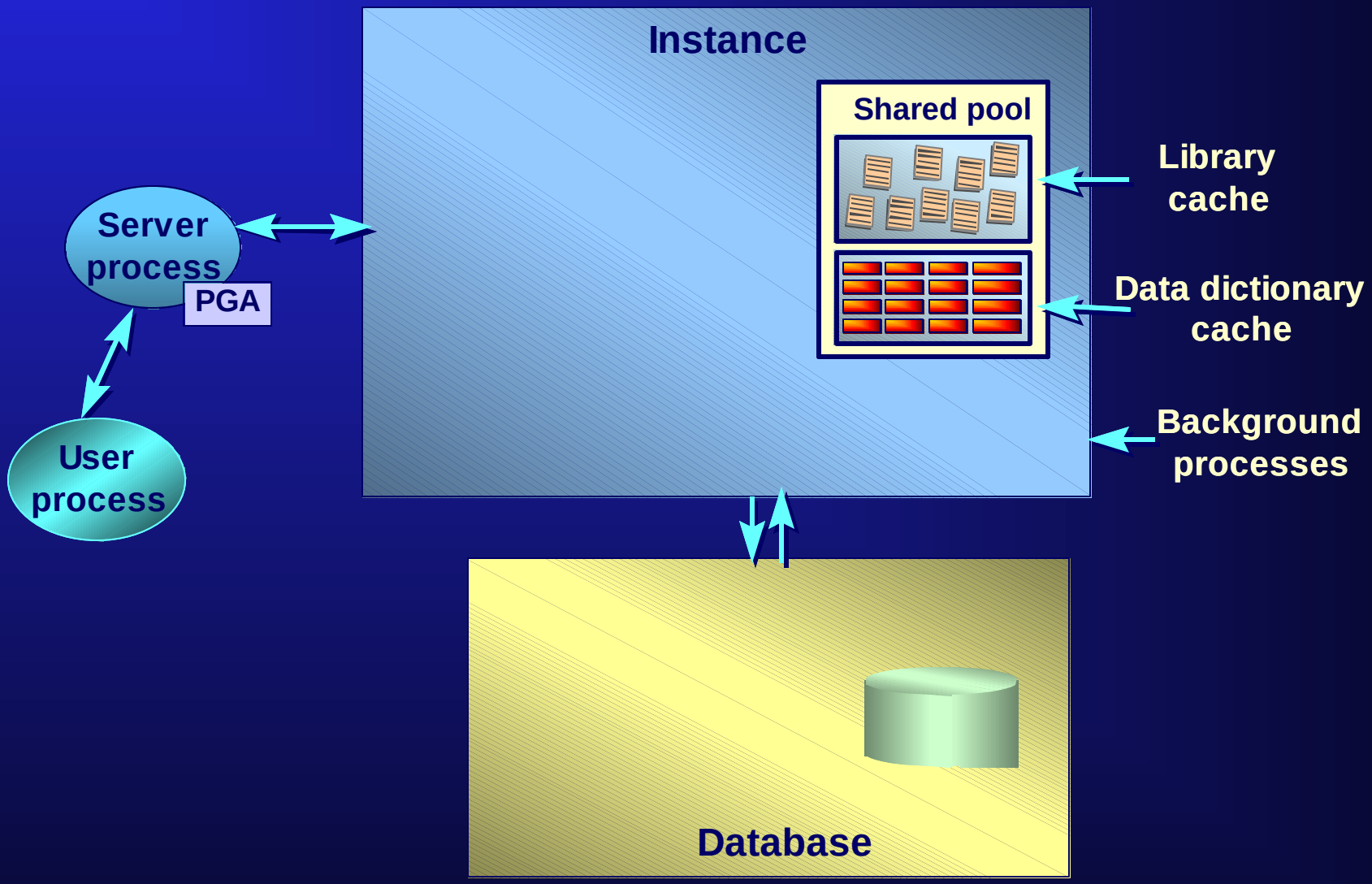
ORACLE

Objectives

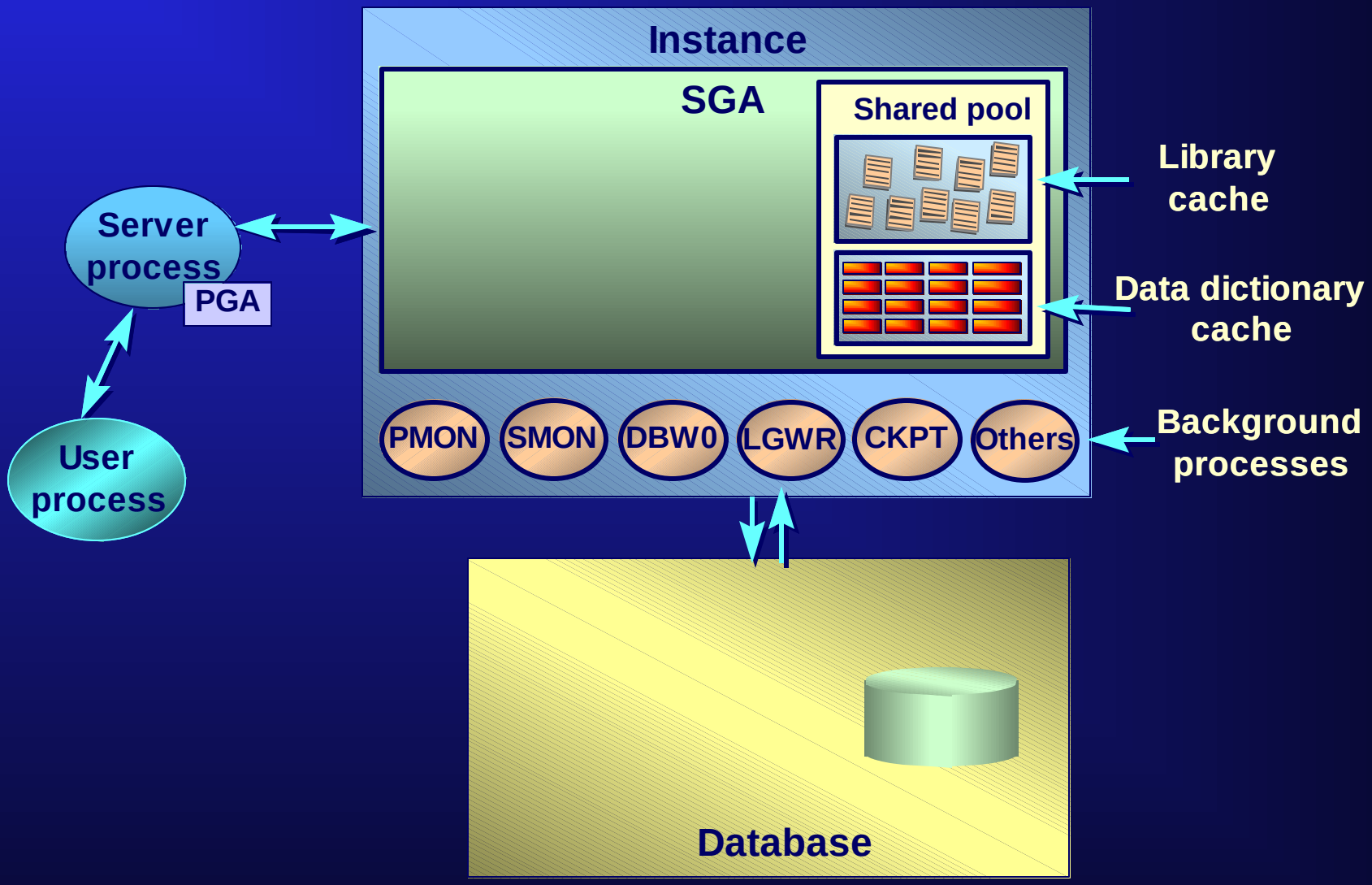
After completing this lesson, you should be able to do the following:

- Describe the Oracle server architecture and its main components
- List the structures involved in connecting a user to an Oracle instance
- List the stages in processing:
  - Queries
  - DML statements
  - COMMITs

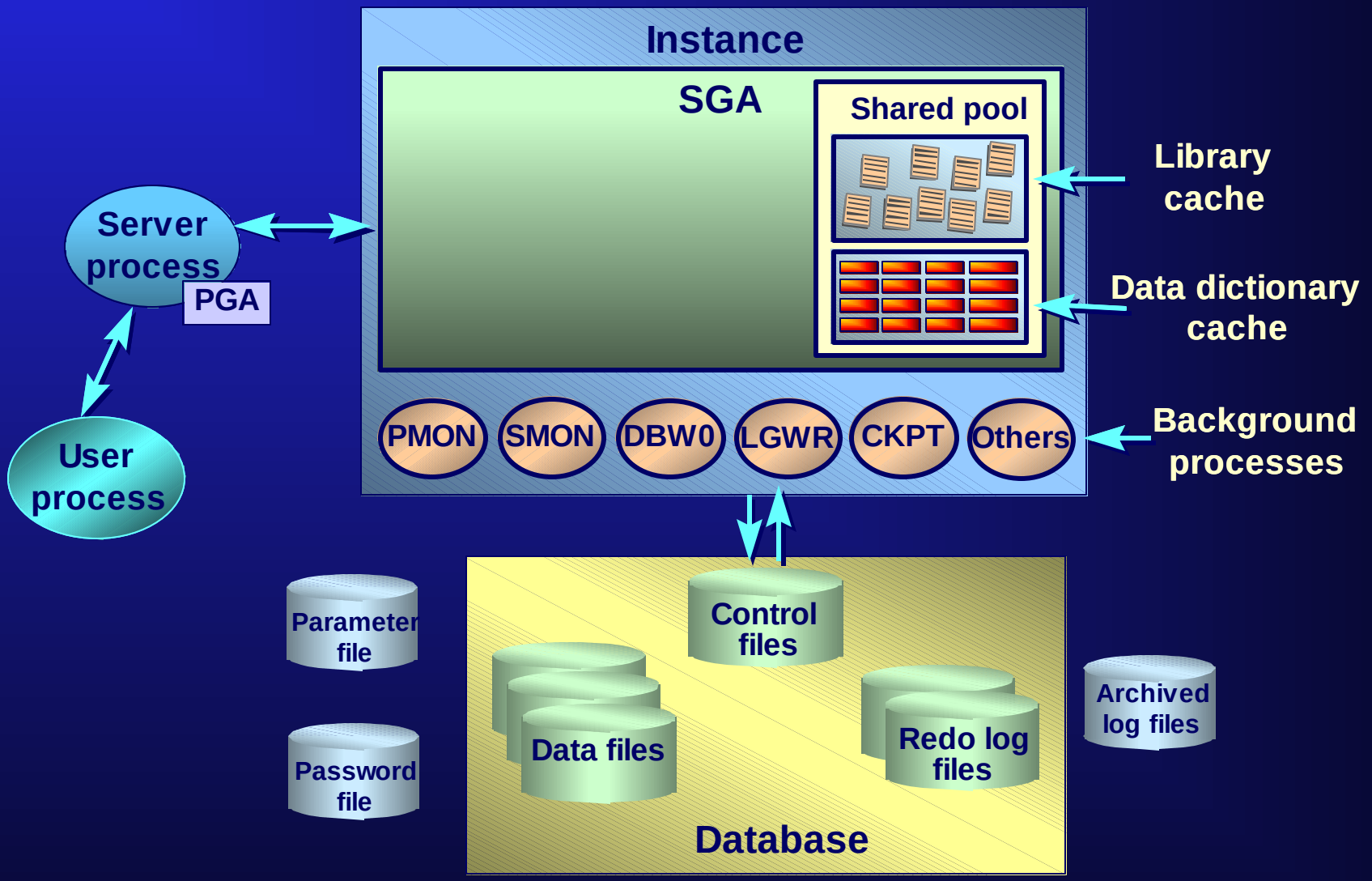
Overview



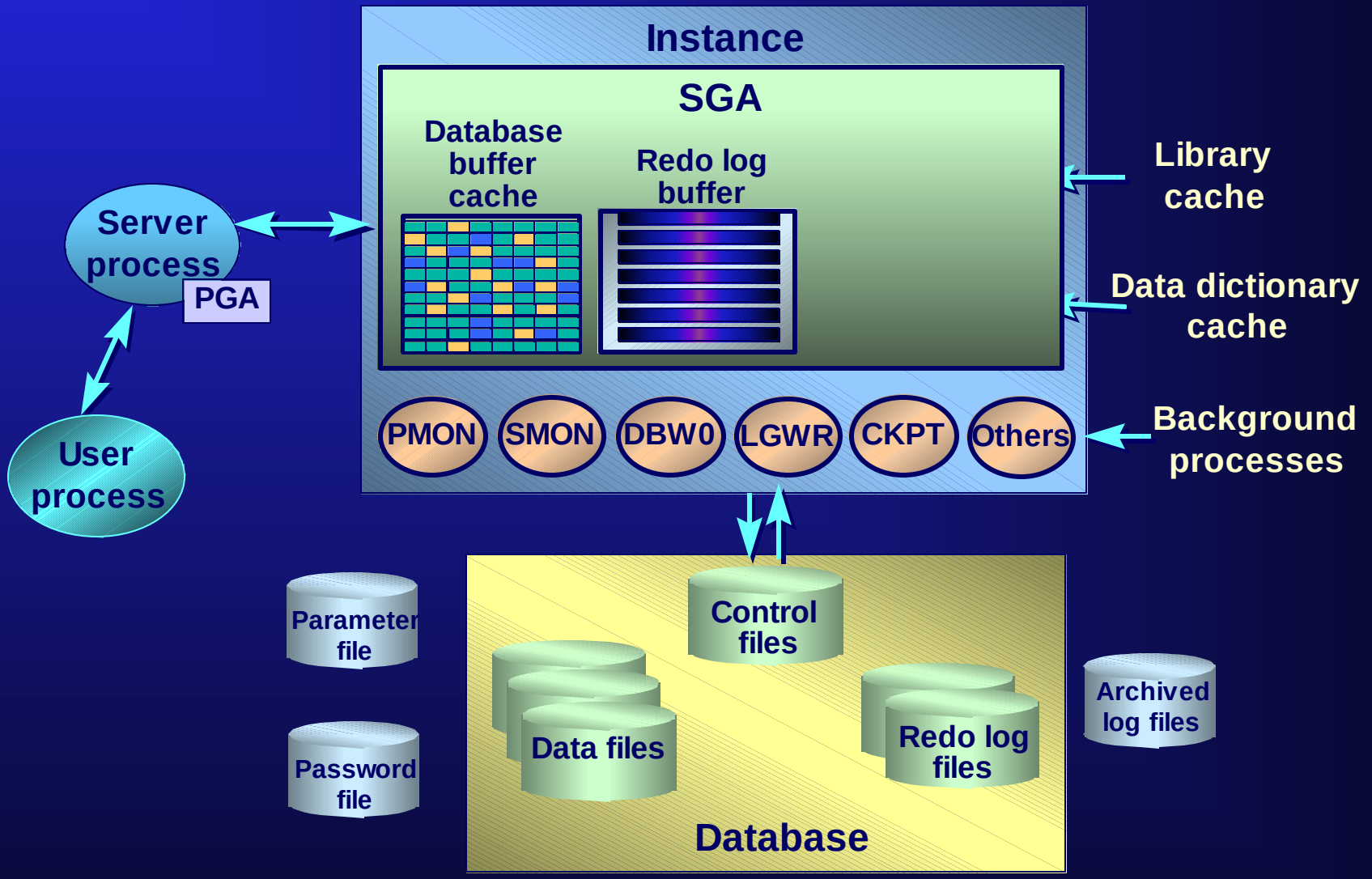
Overview



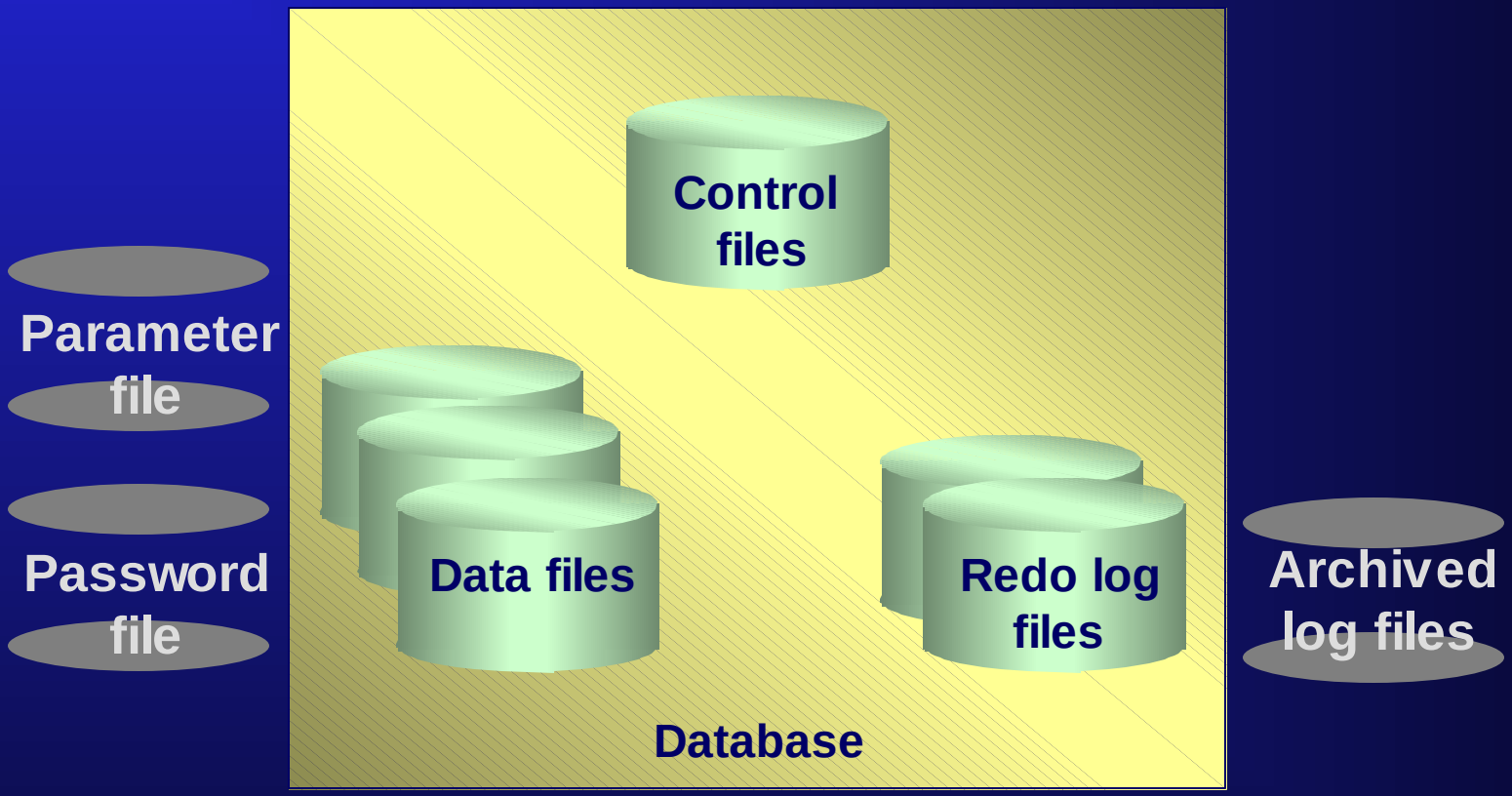
Overview



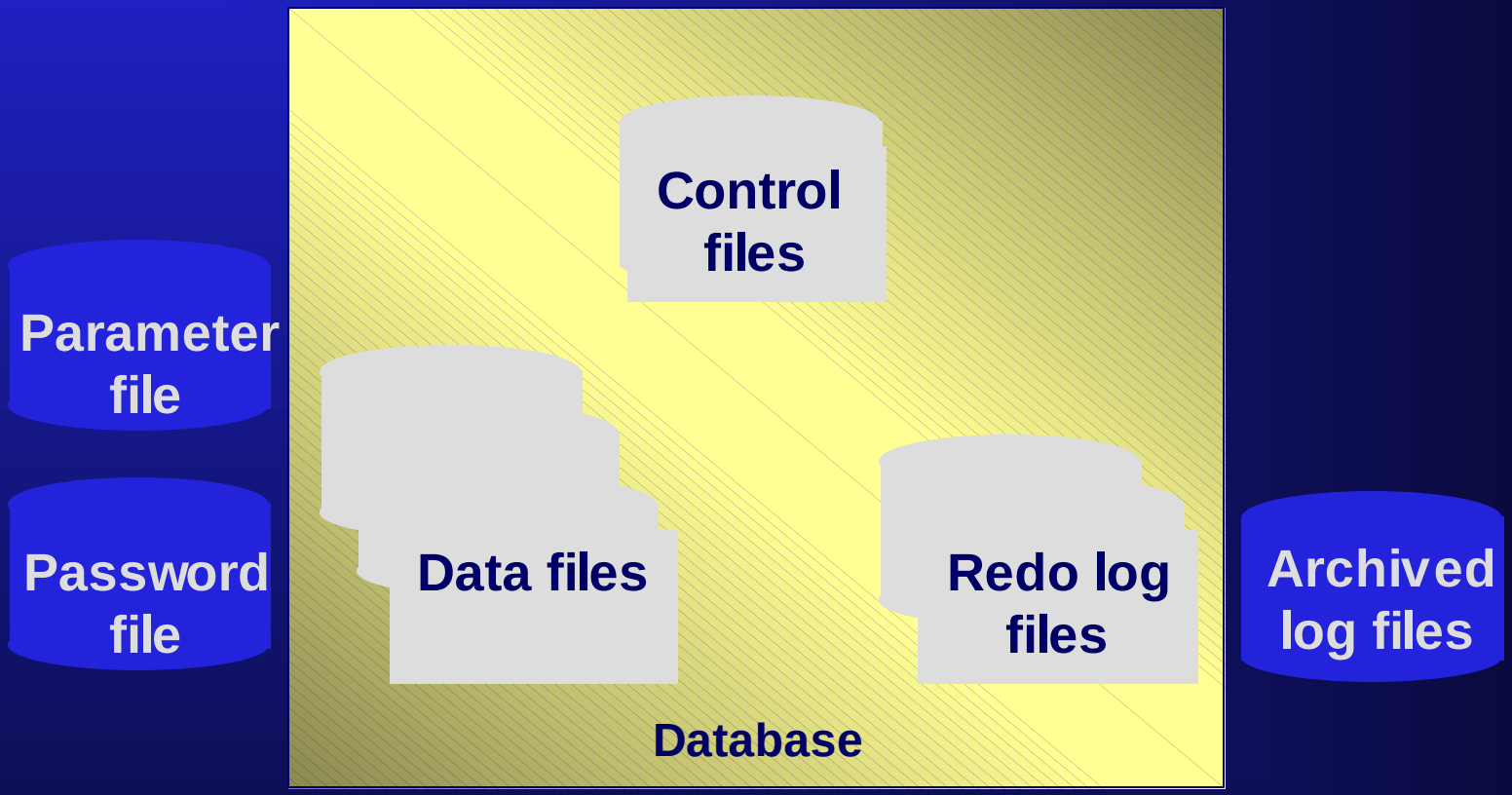
Overview



Oracle Database Files



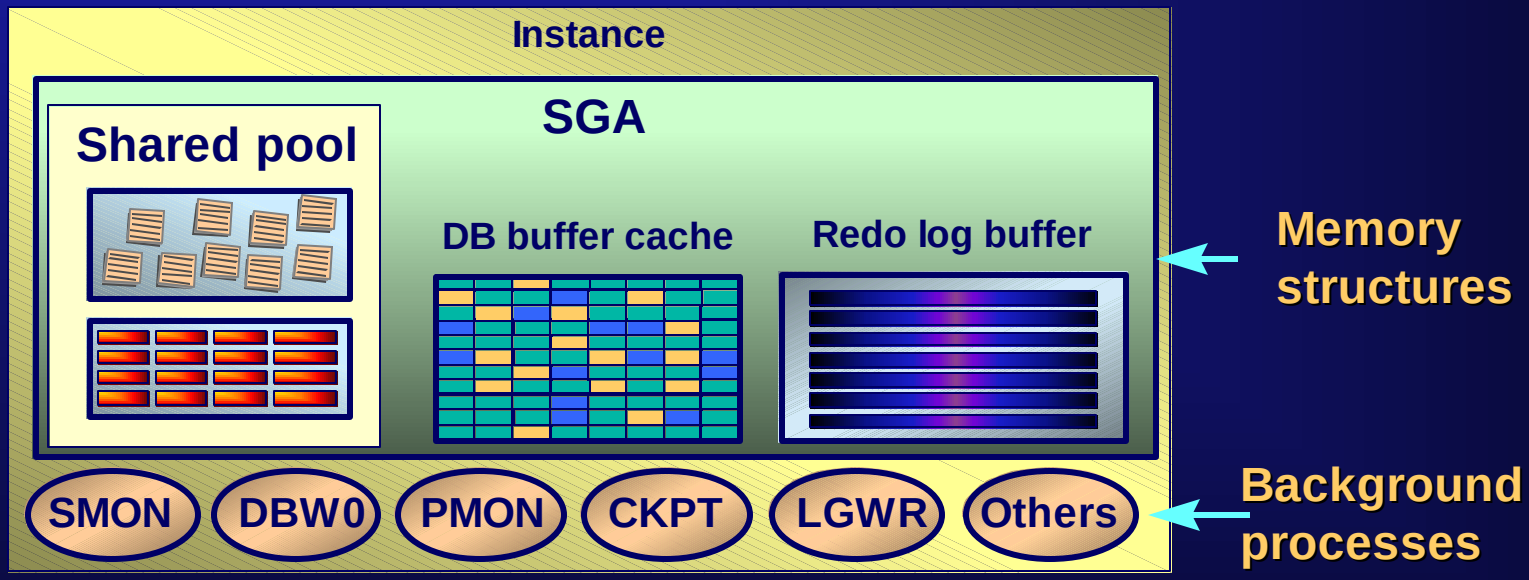
Nondatabase Files



Oracle Instance

An Oracle instance:

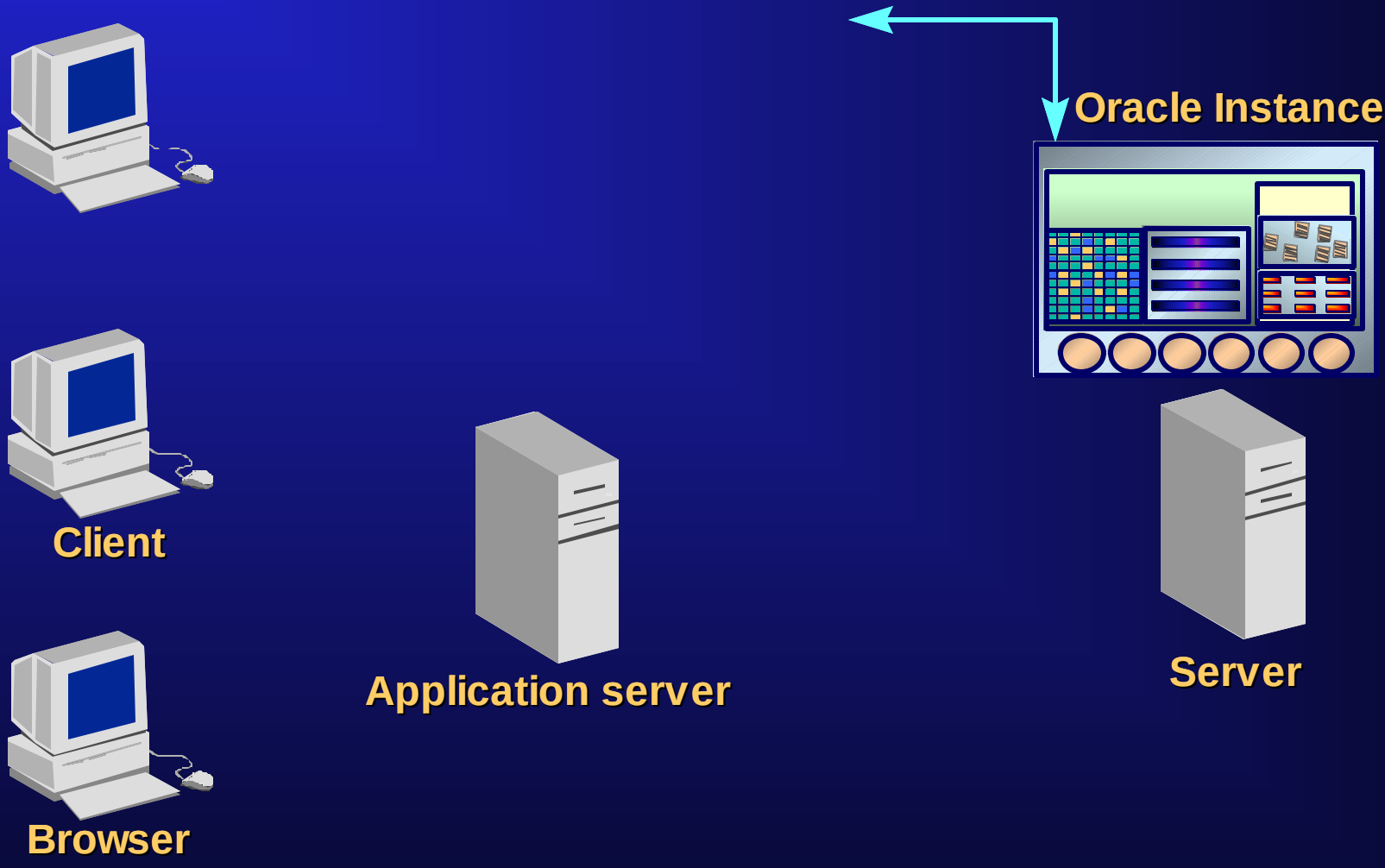
- Is a means to access an Oracle database
- Always opens only one database



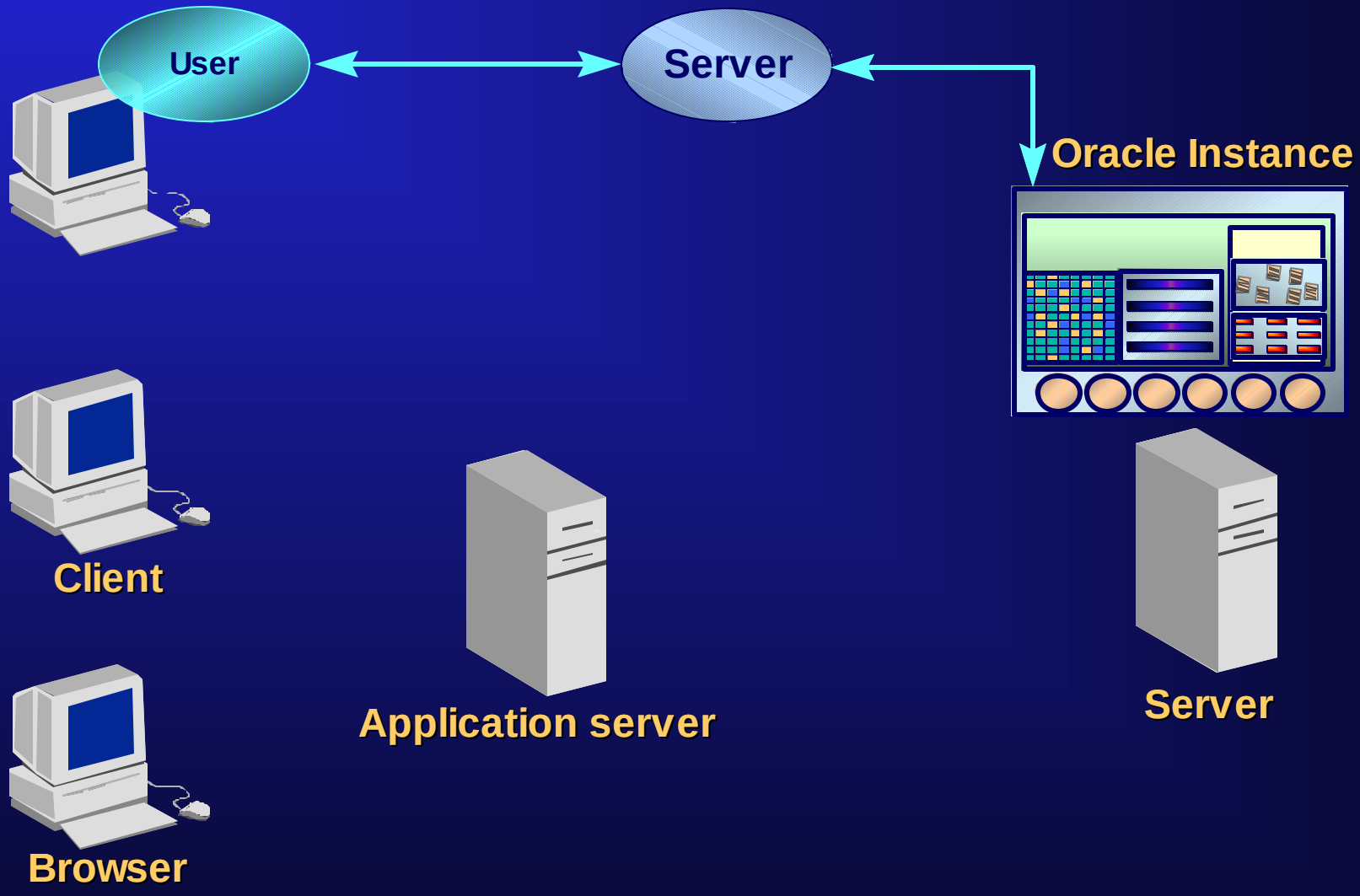
Processing a SQL Statement

- Connect to an instance using:
  - The user process
  - The server process
- The Oracle server components that are used depend on the type of SQL statement:
  - Queries return rows.
  - DML statements log changes.
  - COMMIT ensures transaction recovery.
- Some Oracle server components do not participate in SQL statement processing.

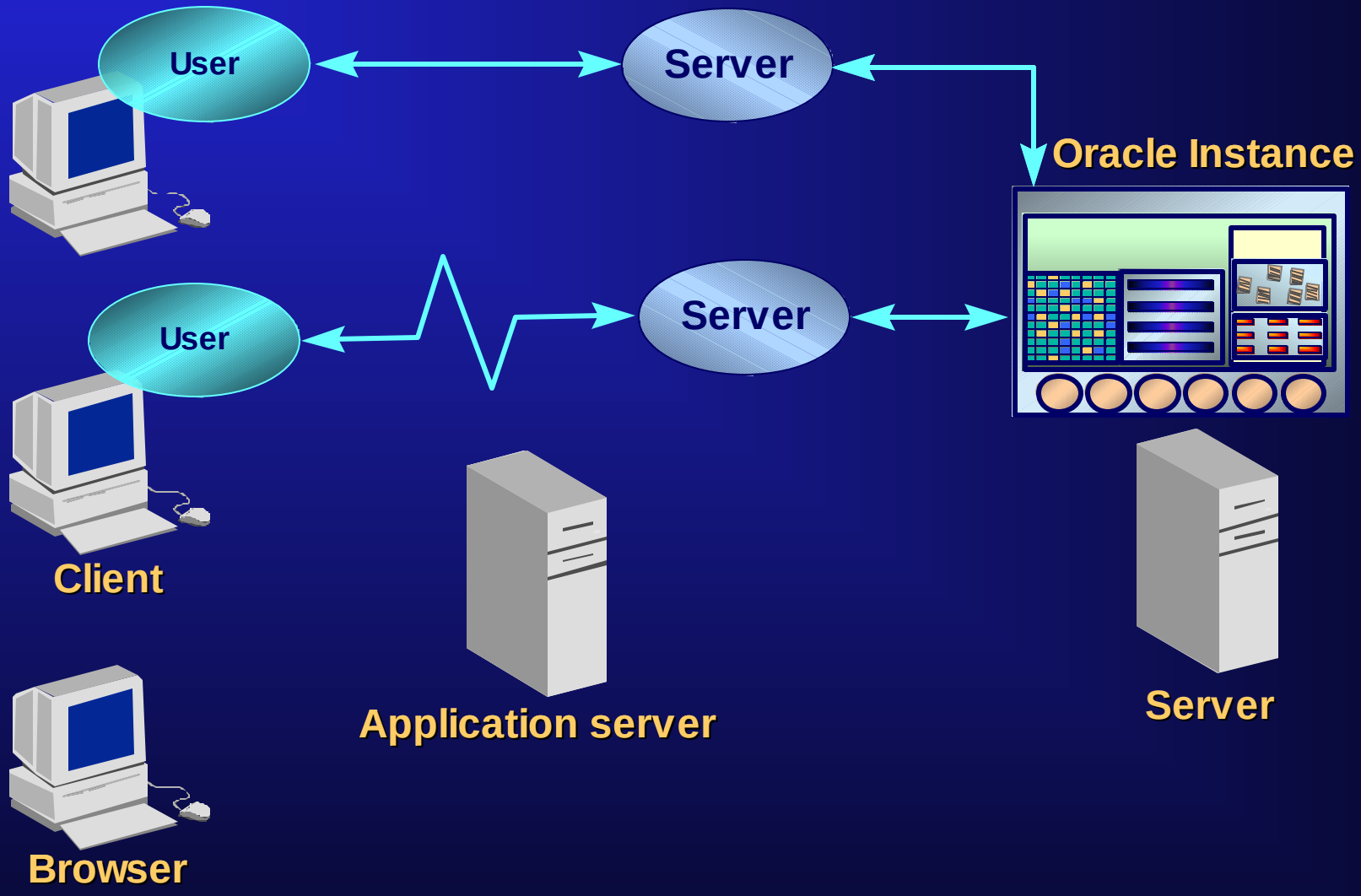
Connecting to an Instance



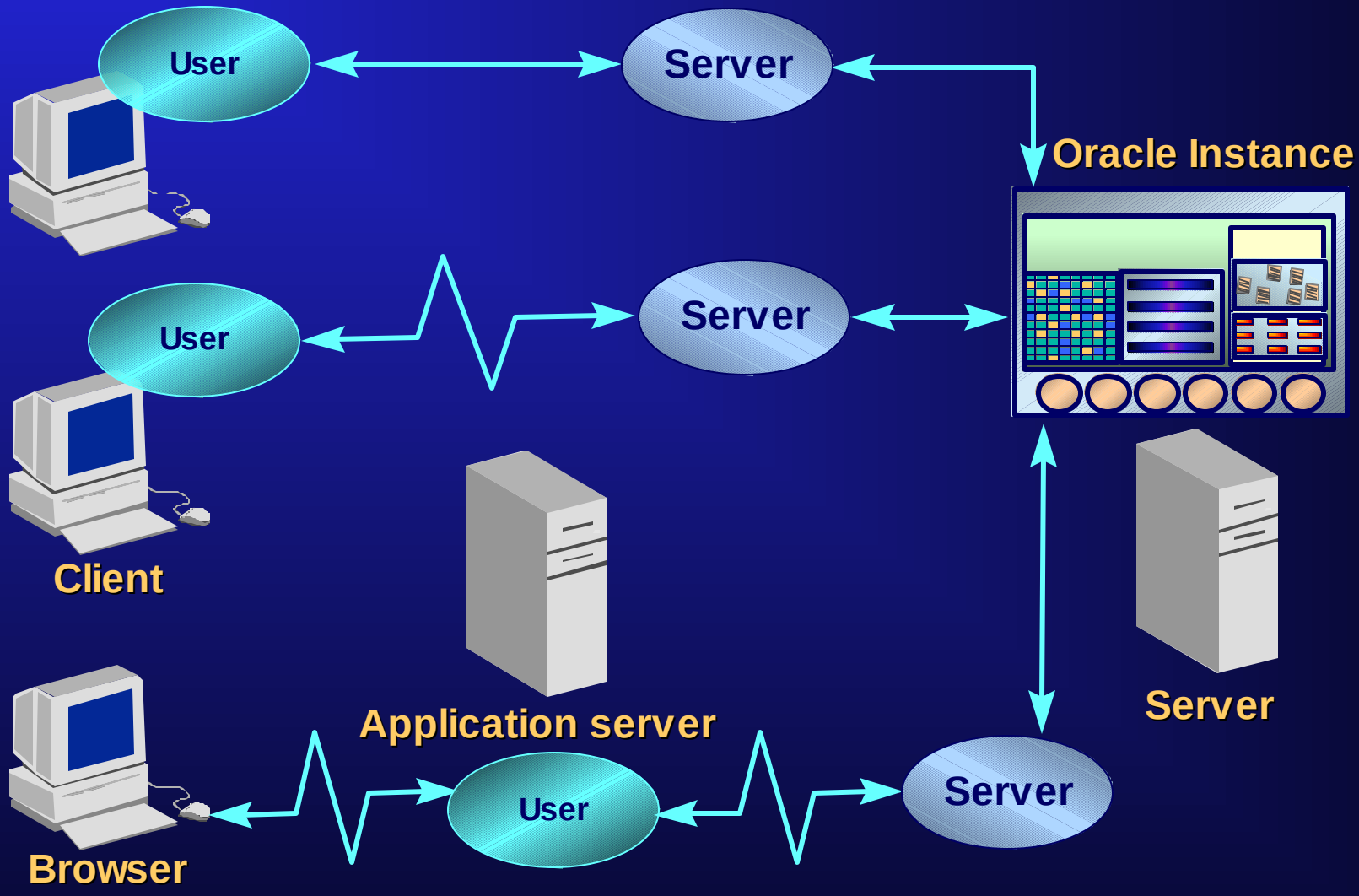
Connecting to an Instance



Connecting to an Instance



Connecting to an Instance

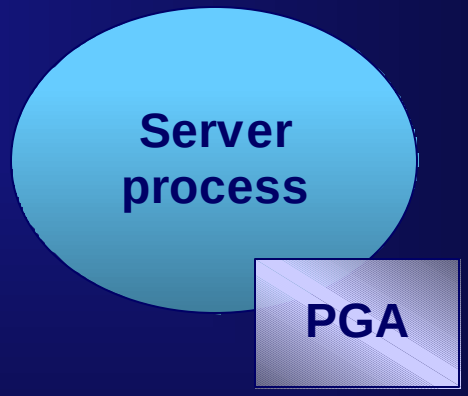


Processing a Query

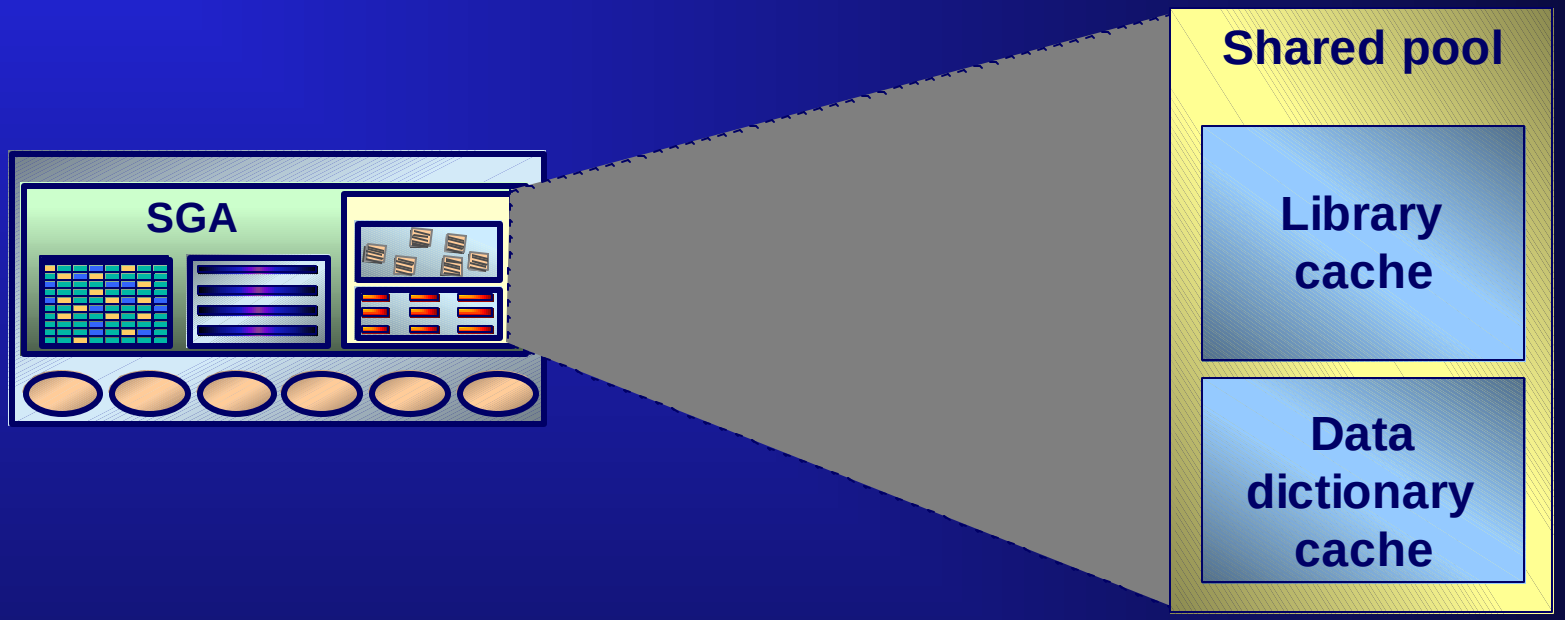
- Parse:
  - Search for identical statement
  - Check syntax, object names, and privileges
  - Lock objects used during parse
  - Create and store execution plan
- Execute: Identify rows selected
- Fetch: Return rows to user process

Program Global Area

- Not shared
- Writable only by the server process
- Contains:
  - Sort area
  - Session information
  - Cursor state
  - Stack space



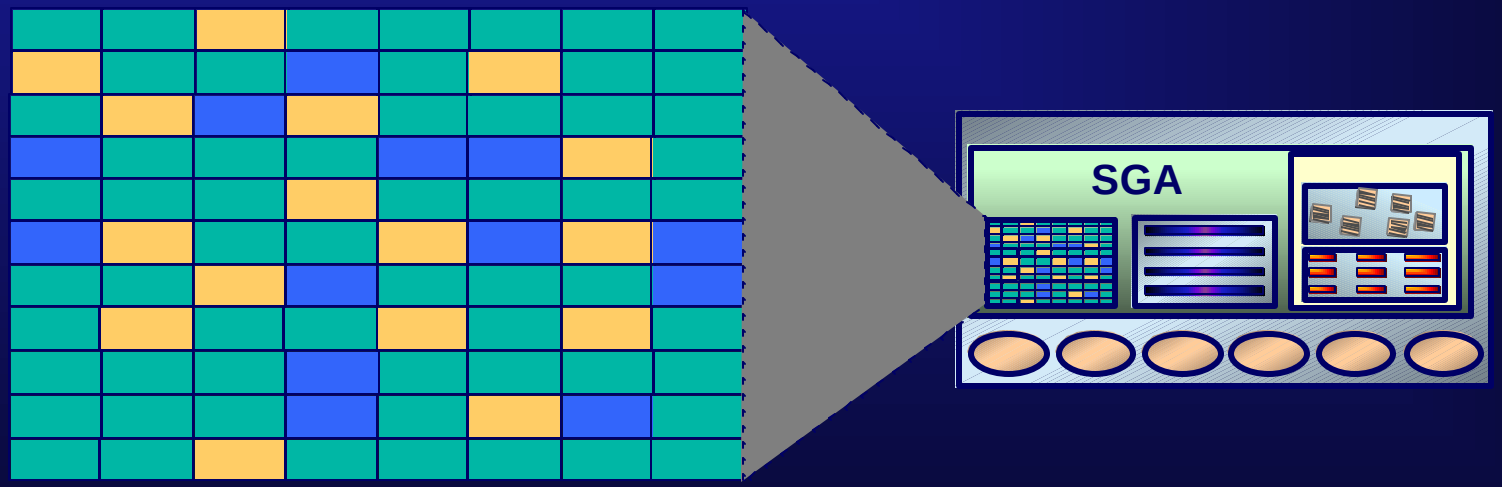
The Shared Pool



- The library cache contains the SQL statement text, parsed code, and execution plan.
- The data dictionary cache contains table, column, and other object definitions and privileges.
- The shared pool is sized by SHARED_POOL_SIZE.

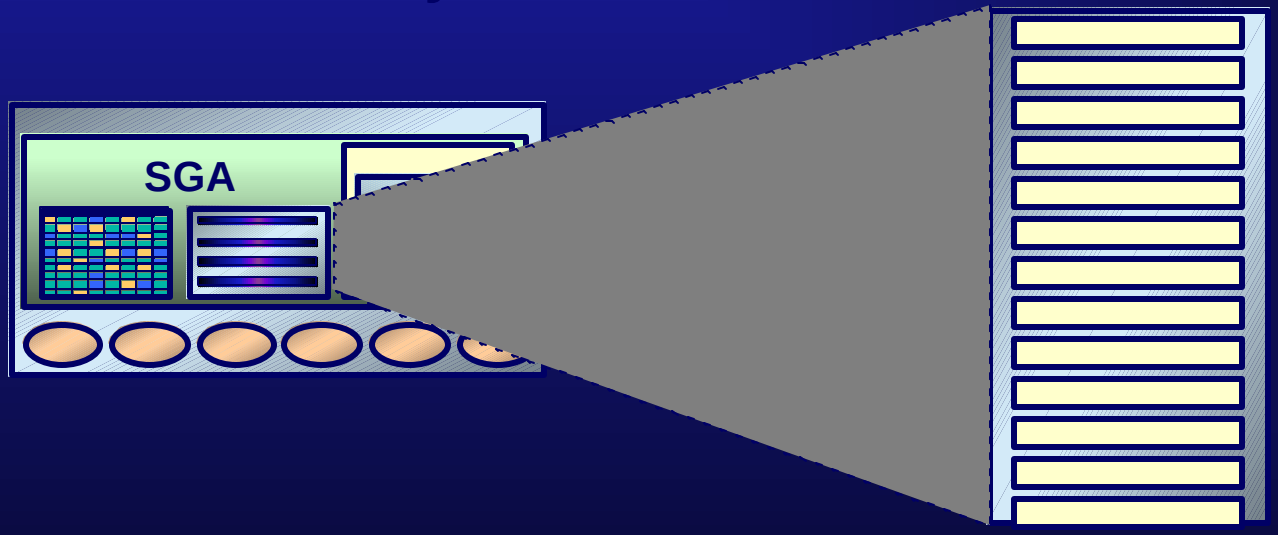
Database Buffer Cache

- Buffer stores the most recently used blocks
- Size of a buffer based on DB_BLOCK_SIZE
- Number of buffers defined by DB_BLOCK_BUFFERS

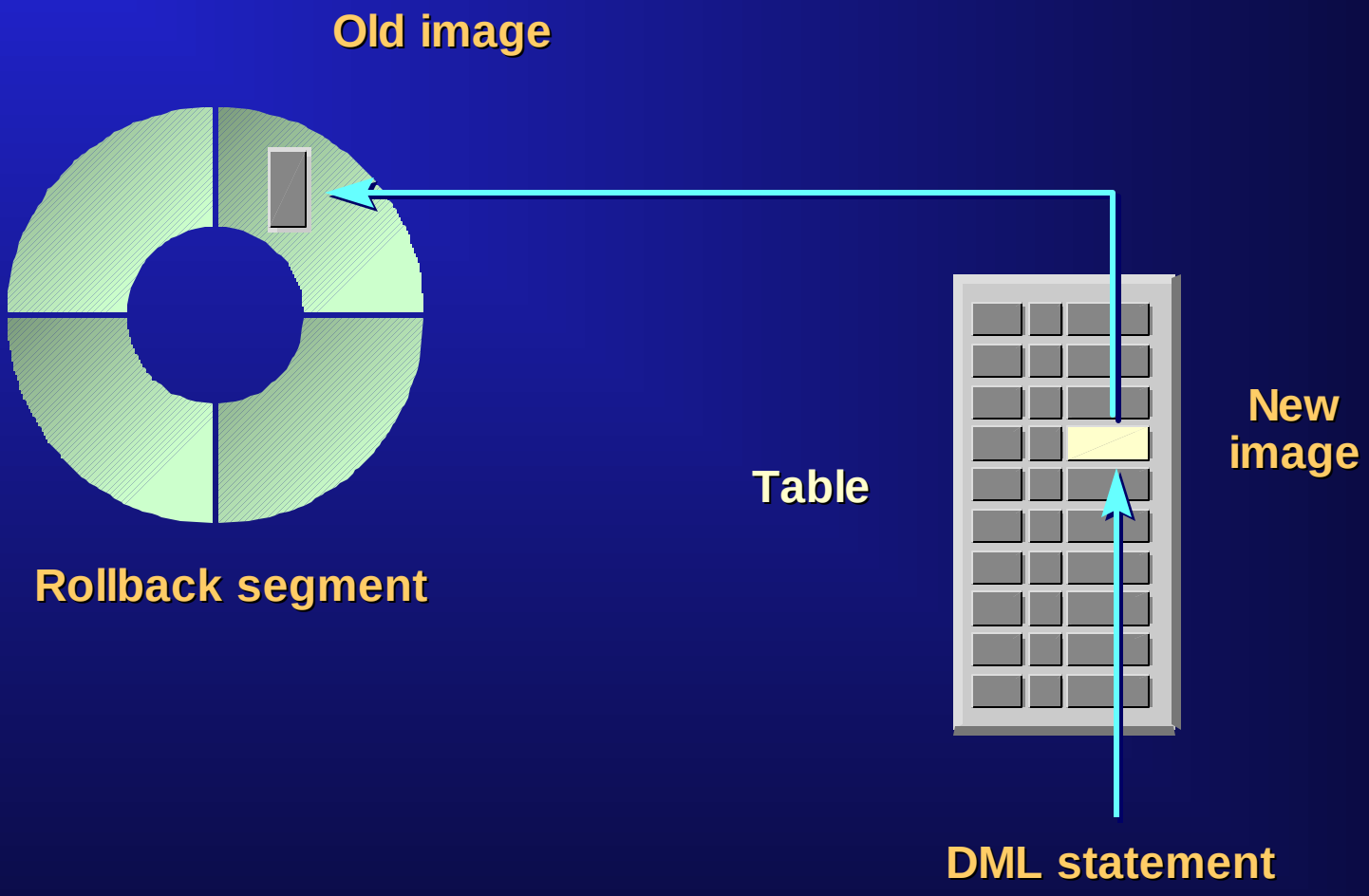


Redo Log Buffer

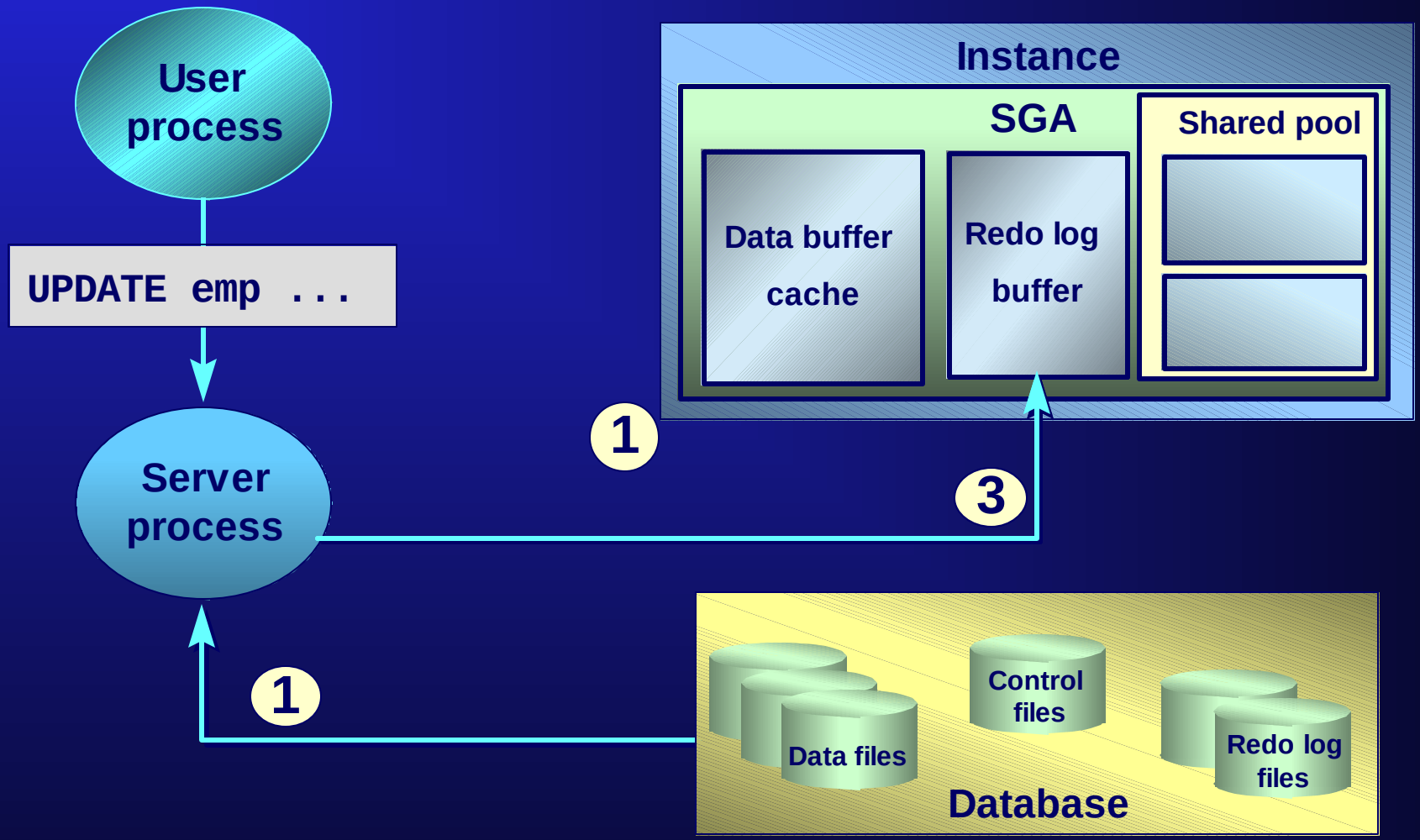
- Records changes made through the instance
- Used sequentially
- Is a circular buffer
- Size defined by LOG_BUFFER



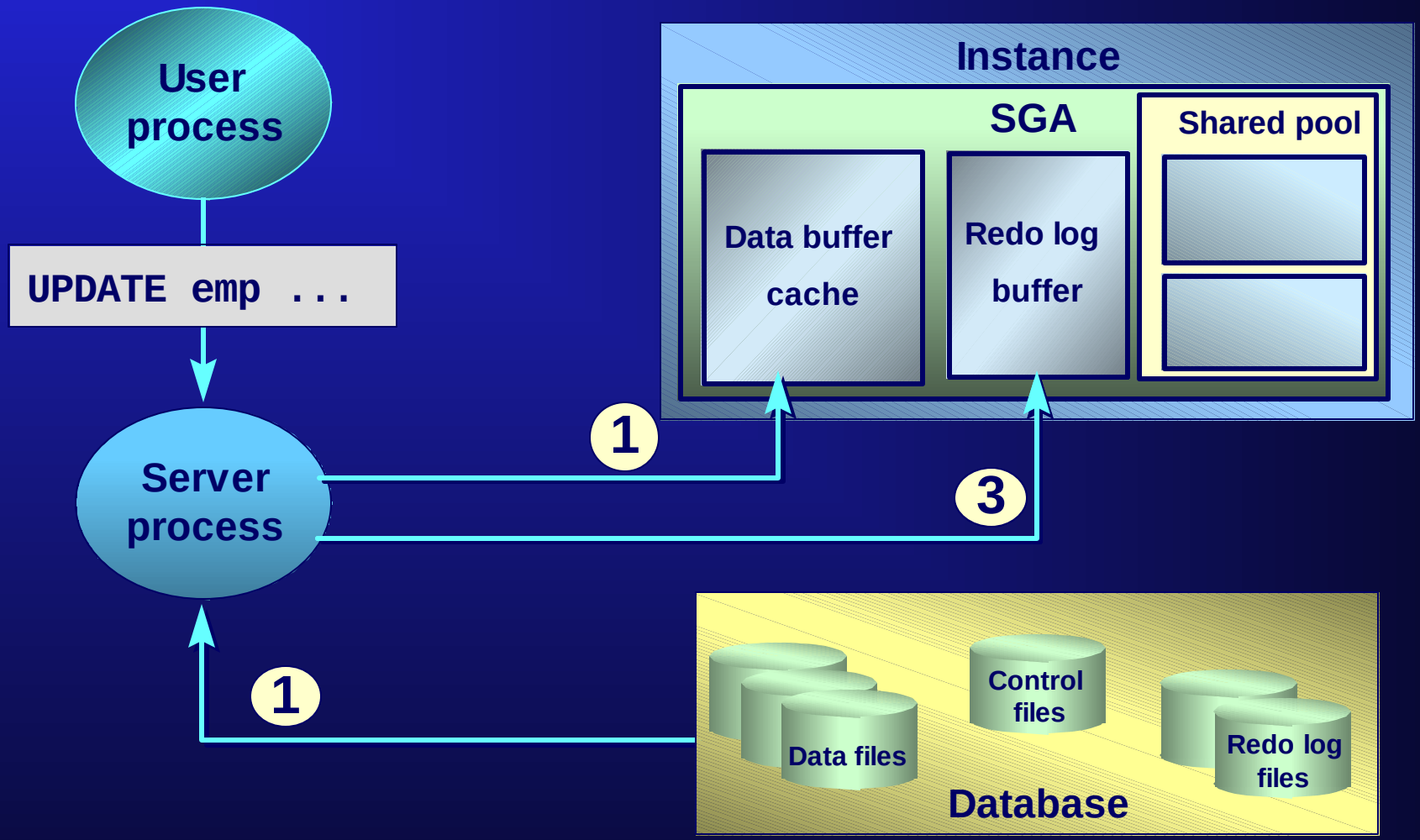
Rollback Segment



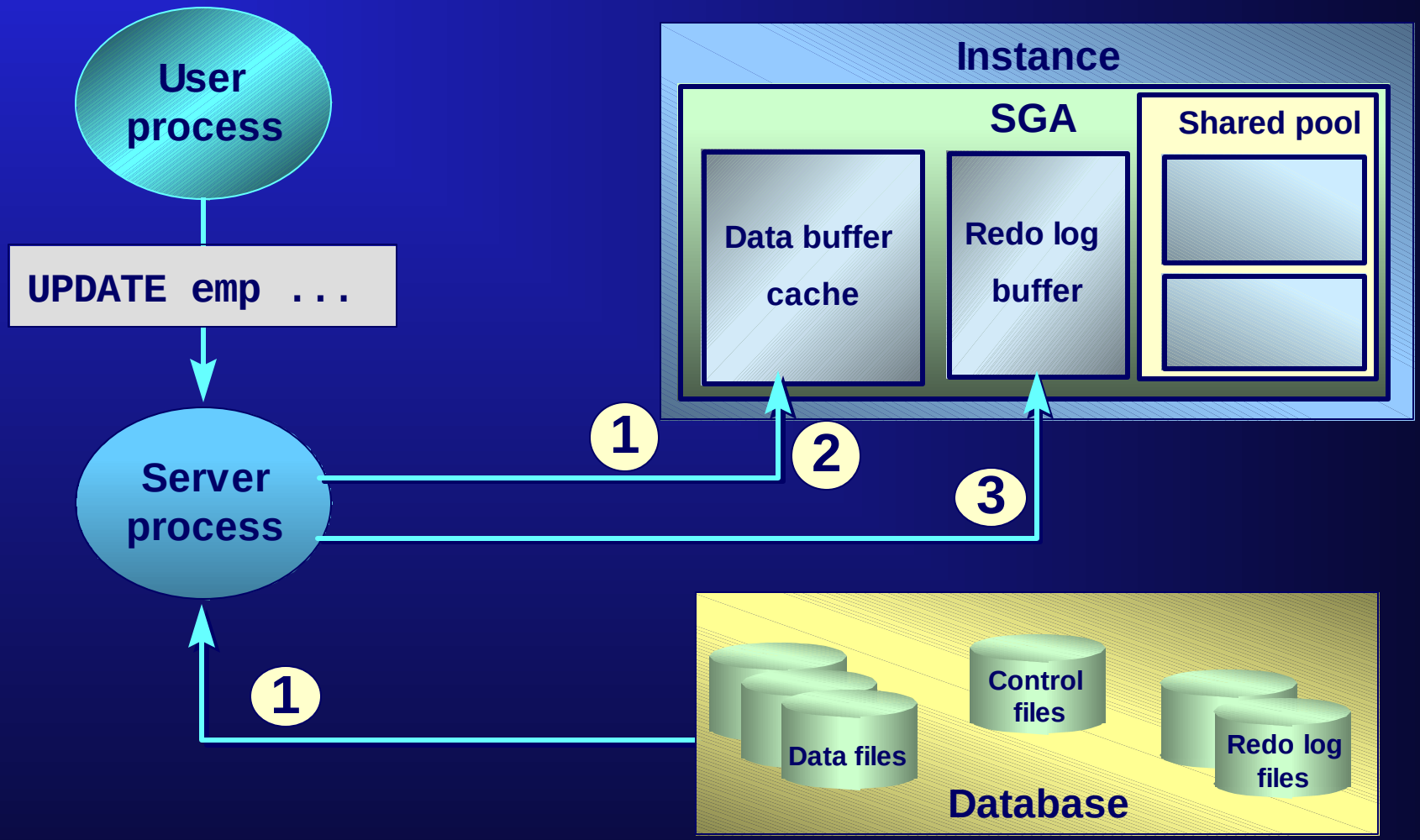
Processing a DML Statement



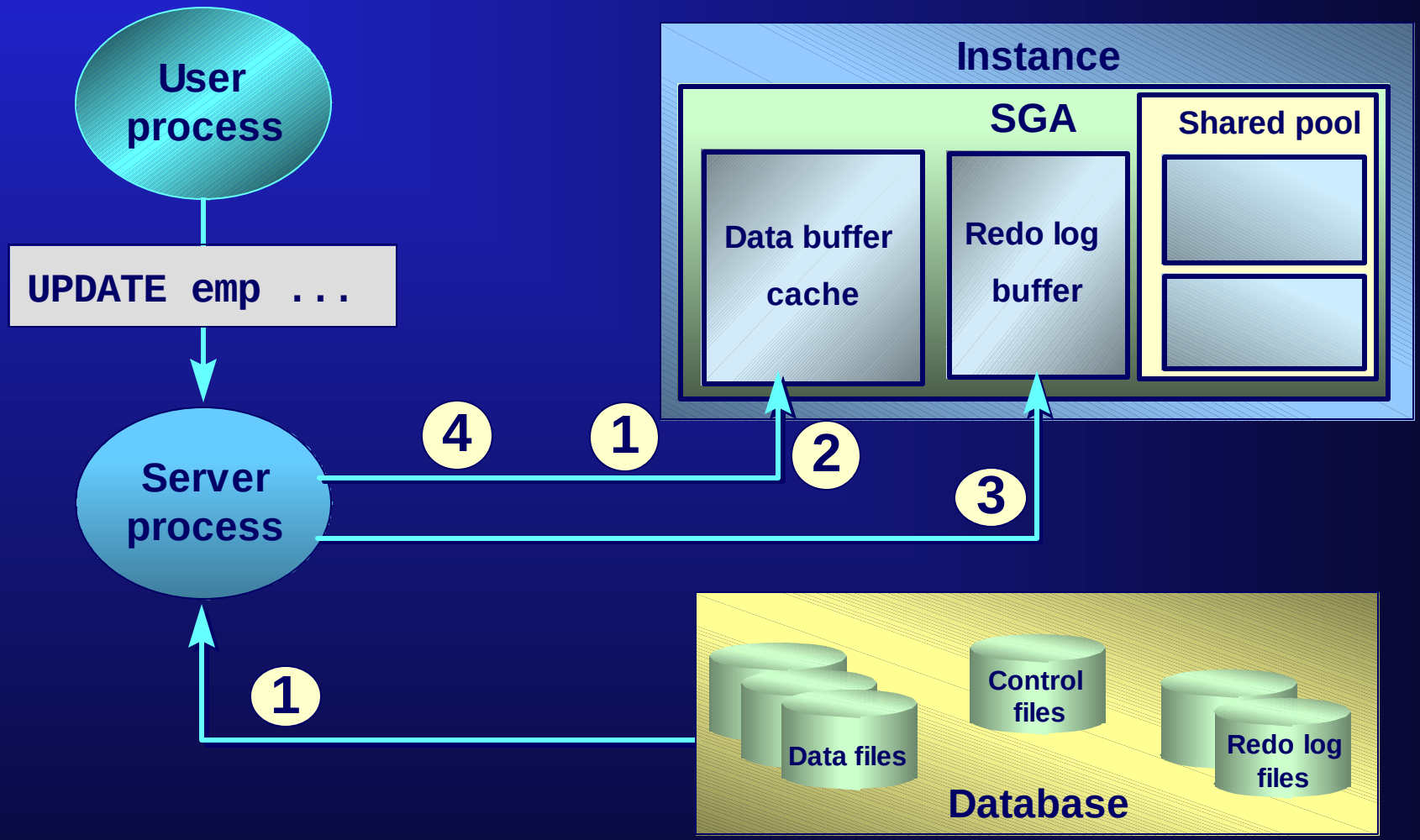
Processing a DML Statement



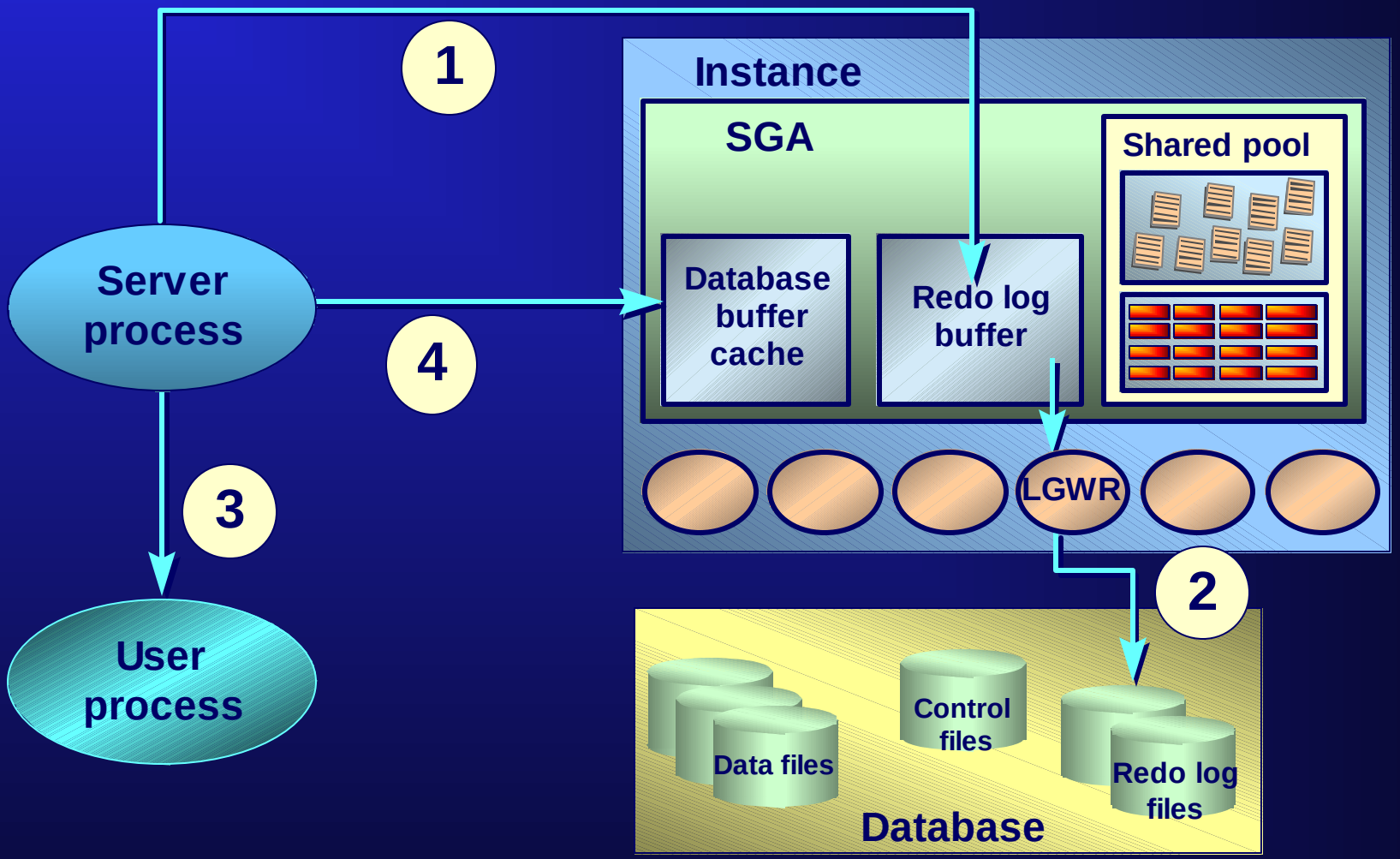
Processing a DML Statement



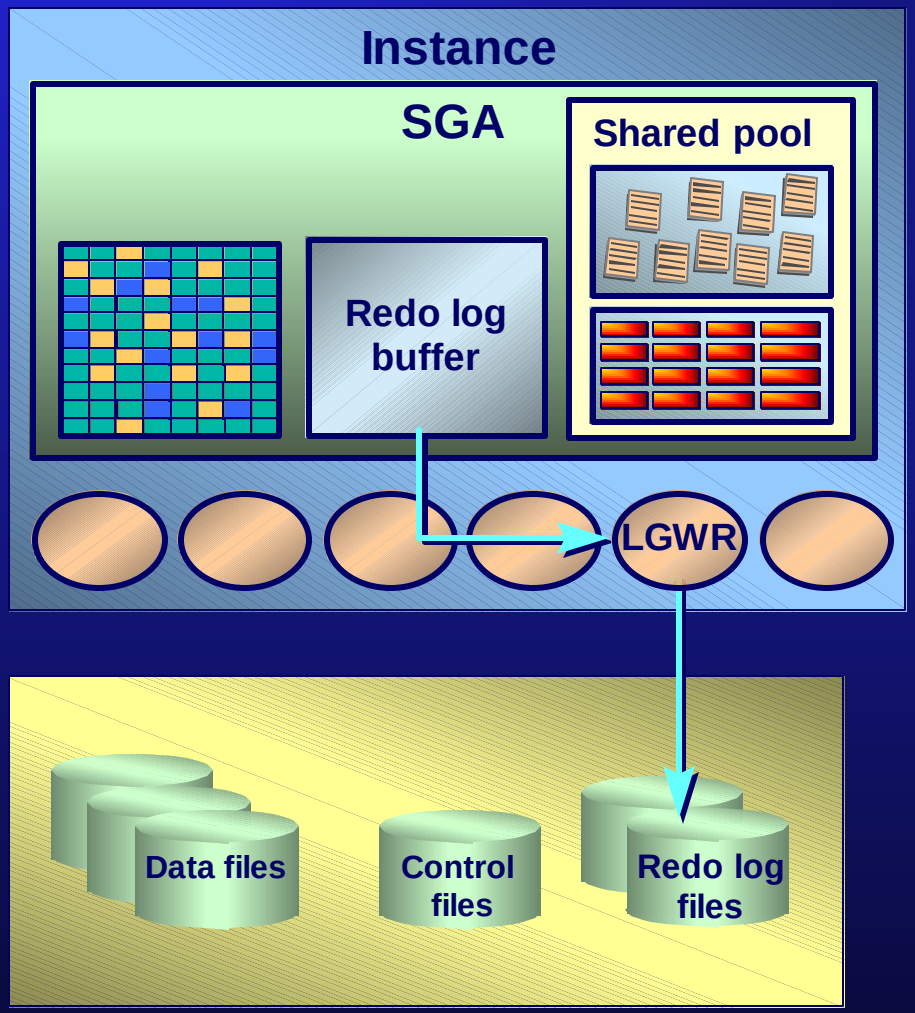
Processing a DML Statement



COMMIT Processing



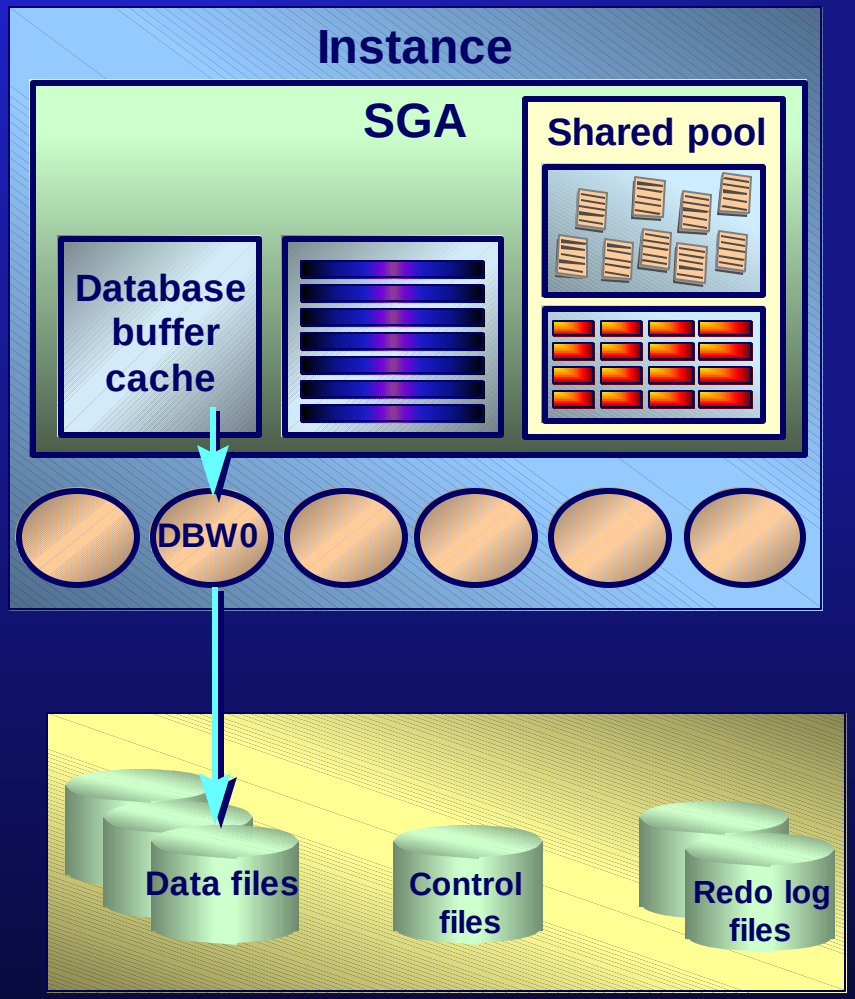
Log Writer



LGWR writes when:

- There is a commit
- The redo log buffer is one-third full
- There is more than 1 MB of redo
- Before DBW0 writes

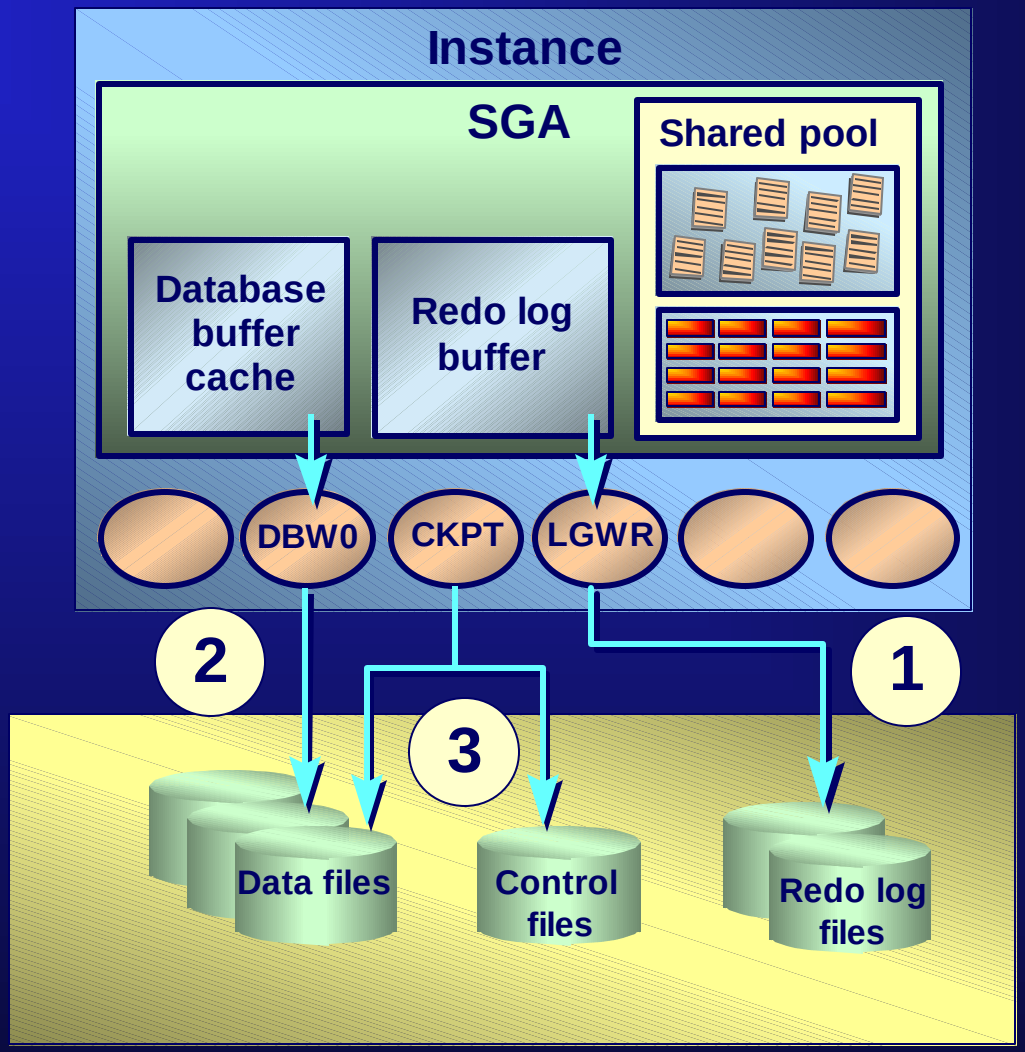
Database Writer



DBW0 writes when:

- There are many dirty buffers
- There are few free buffers
- Timeout occurs
- Checkpoint occurs

Checkpoints



System Monitor

- Automatically recovers the instance
  - Rolls forward changes in the redo logs
  - Opens the database for user access
  - Rolls back uncommitted transactions
- Coalesces free space
- Deallocates temporary segments

Process Monitor

Cleans up after failed processes by:

- Rolling back the transaction
- Releasing locks
- Releasing other resources

Archiver Process

- Database archive mode
  - NOARCHIVELOG for databases that do not require recovery after a disk failure
  - ARCHIVELOG mode for production databases
- ARC0 process
  - Automatically archives online redo logs
  - Preserves the record of all changes made to the database

Summary

In this lesson, you should have learned how to explain:

- Database files: data files, control files, and online redo logs
- SGA memory structures: DB buffer cache, shared pool, and redo log buffer
- Primary background processes: DBW0, LGWR, CKPT, PMON, SMON, and ARC0
- SQL processing steps: parse, execute, and fetch

Practice Overview

This practice covers the following topics:

- Identifying the processes and files with their descriptions
- Querying the data dictionary views to display details of the data file and control file

3

Transactions

Objectives

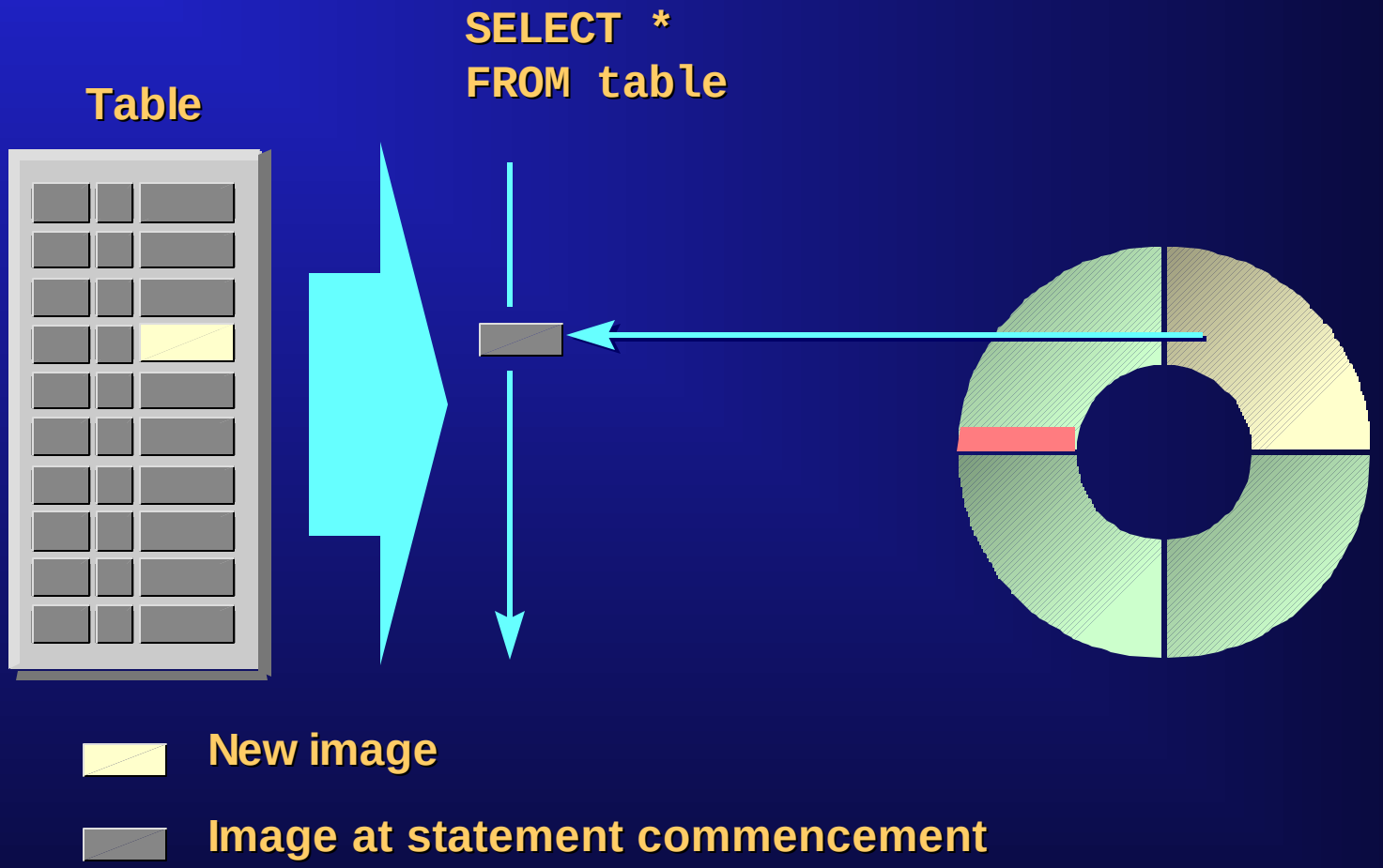
After completing this lesson, you should be able to do the following:

- Choose the appropriate table locks
- Recognize a deadlock
- Determine when to use a regular, read-only, or serializable transaction

Transactions

- Start with an executable SQL statement
- End with a commit or rollback
- For online transactions:
  - Recognize that each transaction is an autonomous unit of work
  - Commit often to avoid lock contention
- For batch transactions:
  - Group units of work into batches
  - Use savepoints to roll back a unit of work

Data Consistency



Data Concurrency

- Two lock modes:
  - Exclusive: Locks out other users
  - Share: Allows other users to access
- High level of data concurrency:
  - DML: Table share, row exclusive
  - Queries: No locks required
  - DDL: Protects object definitions
- Locks held until commit or rollback

Row-Level DML Locks

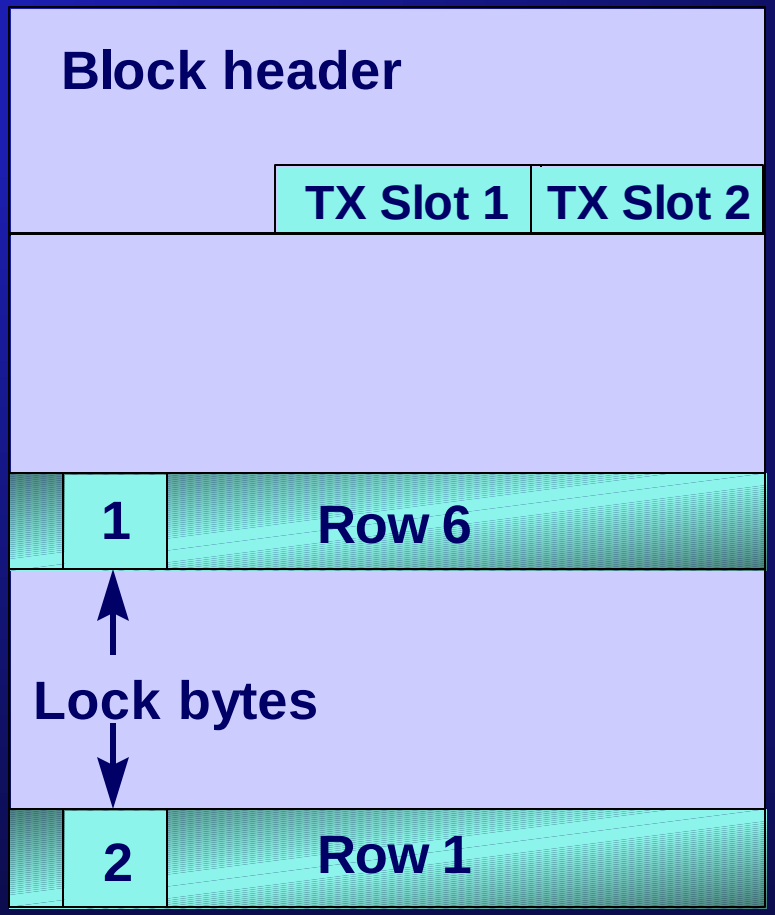


Table-Level Lock Modes

Lock Mode	Description	How Acquired
RS	Row share	SELECT … FOR UPDATE, LOCK TABLE
RX	Row exclusive	DML, LOCK TABLE
S	Share	DML FK parent, LOCK TABLE
SRX	Share row exclusive	DML FK parent with CASCADE, LOCK TABLE
X	Exclusive	LOCK TABLE

Using Manual Locks

- Manually acquired row locks:

```
SQL> SELECT … FROM table_name FOR UPDATE NOWAIT;
```

- Manually acquired table locks:

```
SQL> LOCK TABLE table_name IN mode_name MODE NOWAIT;
```

- SHARE to prevent “snapshot too old”
- EXCLUSIVE for long-running updates

Deadlocks

Transaction 1		Transaction 2

`UPDATE emp`
`  SET sal = sal * 1.1`
`  WHERE empno = 7521;`
	9:00	`UPDATE emp`
		`  SET mgr  = 7698`
		`  WHERE empno = 7782;`

`UPDATE emp`
`  SET sal = sal * 1.1`
`  WHERE empno = 7782;`
	9:01	`UPDATE emp`
		`  SET mgr  = 7698`
		`  WHERE empno = 7521;`

`ORA-0060: deadlock detected while waiting for resource`	9:02	

Isolation Levels

- Data consistency:
  - Always at the statement level
  - Optionally at the transaction level
- Preventable phenomena:
  - Dirty read: Uncommitted data
  - Nonrepeatable read: Requery returns changed data
  - Phantom read: Requery returns new rows

Oracle Isolation Levels

- Read committed is the default level

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SQL> ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
```

- Serializable provides transaction-level consistency

```
SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
```

- Read-only provides transaction level without DML

```
SQL> SET TRANSACTION READ ONLY;
```

Choosing an Isolation Level

- Choose based on application requirements
- Trade-off between:
  - Consistency
  - Concurrency

Isolation Level	Non-Repeat Read	Phantom Read	DML	Cannot Serialize Error
Read only	No	No	No	No
Read commit	Yes	Yes	Yes	No
Serializable	No	No	Yes	Yes

Read Committed Isolation

- Default isolation level
- Suitable for:
  - High-performance environments
  - Few users with low transaction rates
  - Most applications
- Programming logic:
  - None for trapping “cannot serialize” errors
  - Rows stored in program to prevent phantom and nonrepeatable read

Serializable Isolation

- Suitable where:
  - Large databases and short transactions
  - Little concurrent access of rows
  - Long-running transactions are read-only
- Programming logic:
  - Trap “cannot serialize access” errors
  - Overhead to rollback after error
  - Use SELECT … FOR UPDATE to lock rows

Summary

In this lesson, you should have learned that:

- Queries do not lock data, except SELECT FOR UPDATE
- DML statements use exclusive row-level locks and share table-level locks
- The Oracle server always provides statement-level read consistency
- The Oracle server can also provide transaction-level read consistency

Practice Overview

This practice covers the following topics:

- Detecting the table locks by updating a table
- Detecting dead locks

4

Designing the Physical Structure

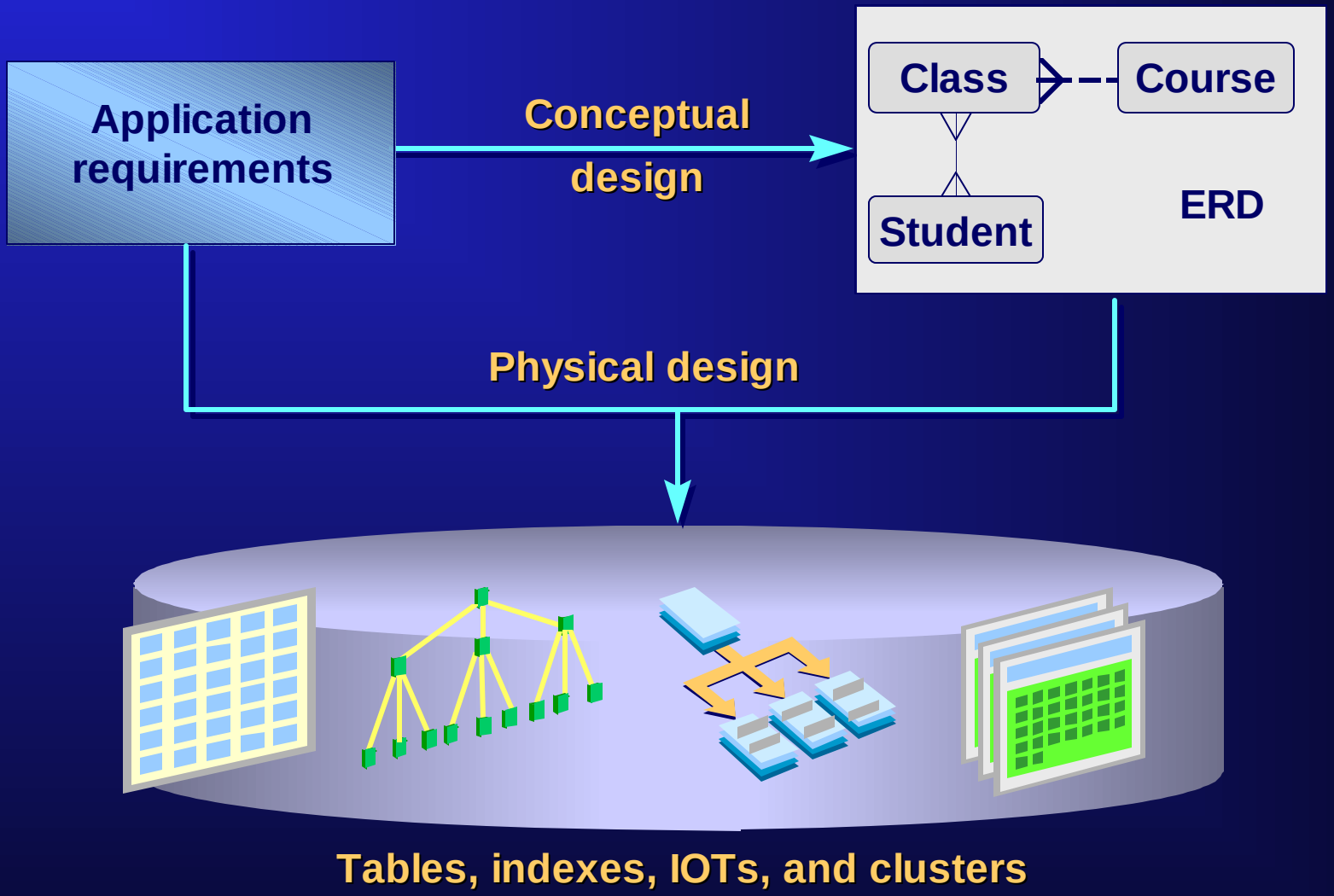
ORACLE

Objectives

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

- Identify the two phases in the data design process
- List the factors influencing the schema design process
- List the physical data storage structures

Overview



Oracle Schema Design

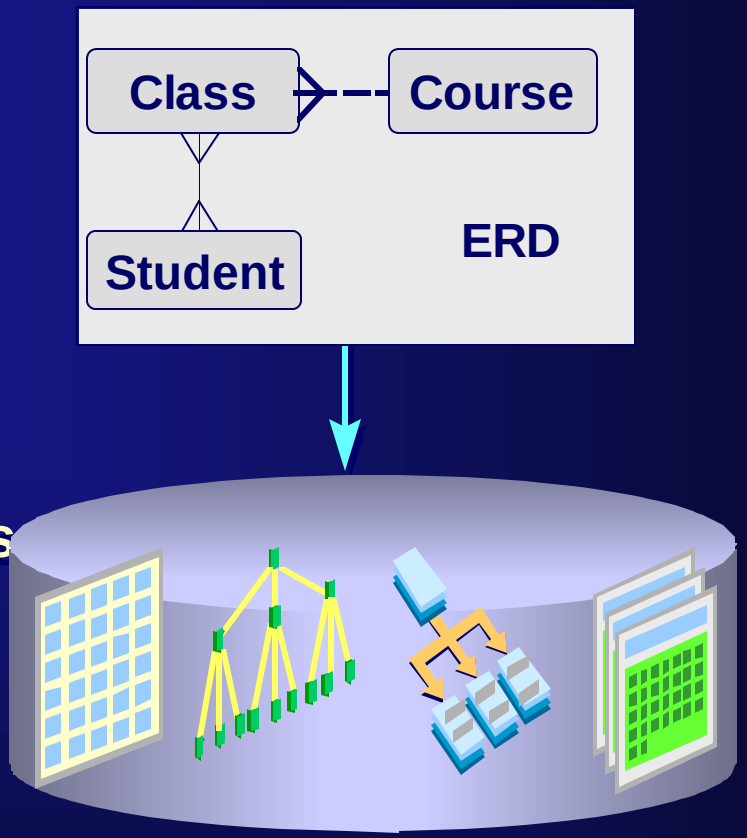
1. Convert ERD to tables, columns, and foreign keys:
   - Consider derived data.
   - Prevent contention.
   - Combine entities.
2. Select data types.
3. Determine how to enforce data integrity.
4. Select physical structures.
5. Select indexes.
6. Decide whether to use other Oracle8*i* features.

Schema Design Factors

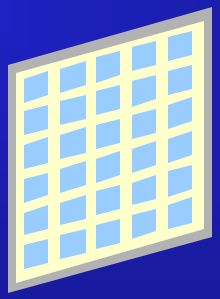
- Type of application:
  - Online transaction processing (OLTP)
  - Decision support system
  - Hybrid system
- Application requirements:
  - Row access
  - Table size
  - Other considerations

Selecting the Physical Data Structure

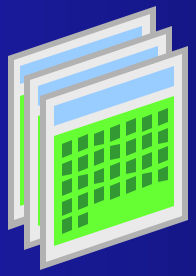
- Rows read in groups
- SELECT or DML
- Table size
- Row or block size
- Row group or block size
- Small or large transactions
- Parallel query to load or SELECT



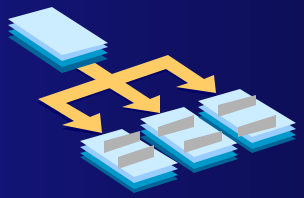
Data Storage Structures



Regular table



Cluster



Index-organized table

Organization by value

Heap — Clustered — Sorted

Partitioned table

ORACLE

Summary

Summary

In this lesson, you should have learned the following

Summary

In this lesson, you should have learned the following topics:

Summary

In this lesson, you should have learned the following topics:

- The two phases in the data design process
  - Conceptual design
  - Schema design

Summary

In this lesson, you should have learned the following topics:

- The two phases in the data design process
  - Conceptual design
  - Schema design
- Schema design factors

Summary

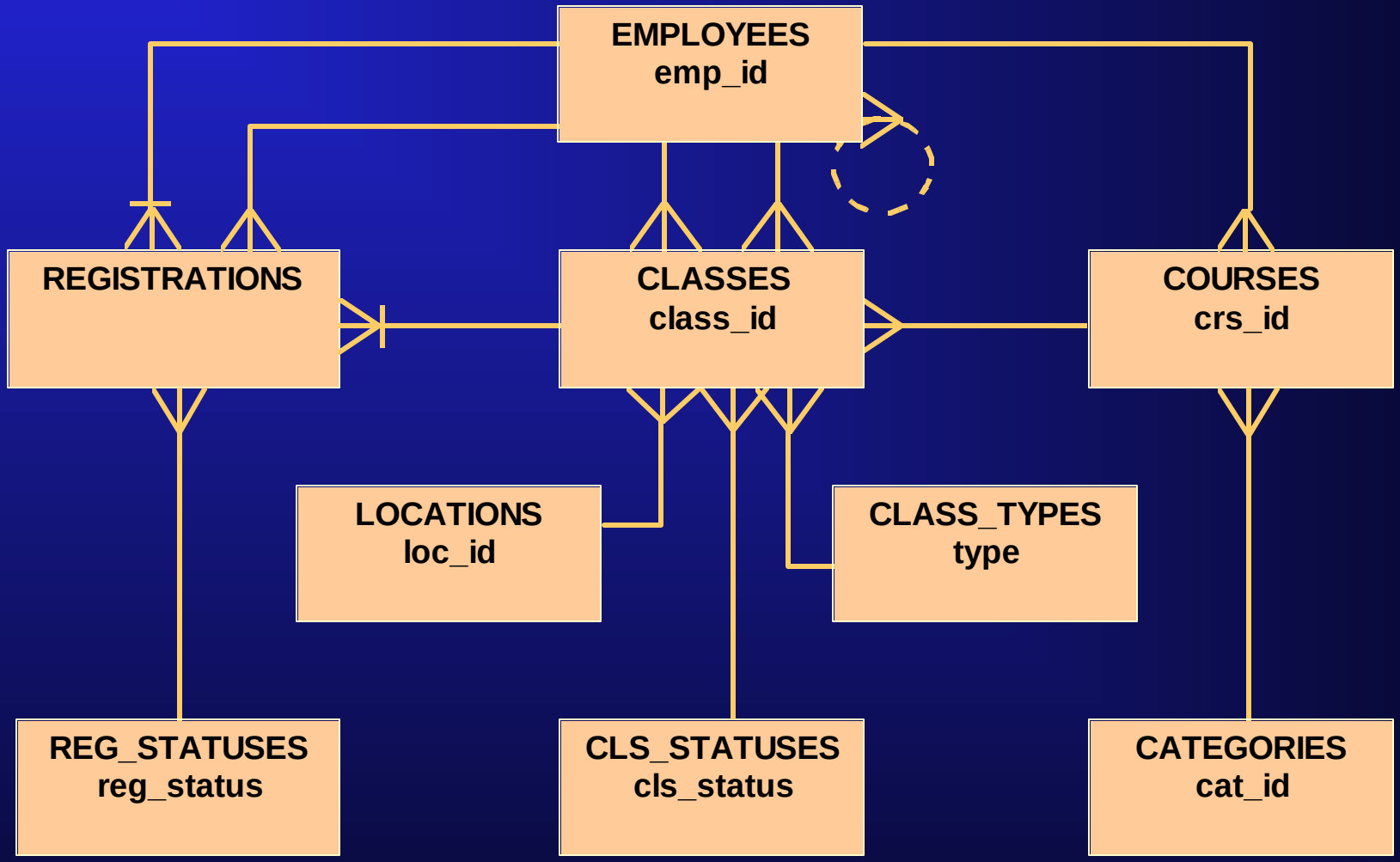
In this lesson, you should have learned the following topics:

- The two phases in the data design process
  - Conceptual design
  - Schema design
- Schema design factors
- Selecting the physical data storage structure

Practice Overview

This practice covers observing the case study that will be implemented in the following lessons.

Case Study



Entity Description

- Entity name
- Examples
- Entity description
  - Data characteristics
  - Access characteristics
  - Column description

Data Characteristics

- Primarily for sizing segments
- May be used to determine the data structure
- Includes:
  - Current number of rows
  - Average bytes added by update
  - Annual % increase in the number of rows
  - Other characteristics that may be useful

Access Characteristics

- Used to choose the physical structure
- Access is for an online program
- Includes:
  - Primary data access methods
  - Secondary data access methods
  - Other types of Select or DML activity
  - Entities joined to this entity

Column Description

- Similar to the output of SQL*Plus DESCRIBE command
- Also includes:
  - Average number of bytes used to store the column value
  - Constraints:
    - PK - primary key constraints
    - NN - not null constraints
    - FK - foreign key, including the parent table

Other Notes

- All but one of these structures used:
  - Table
  - IOT
  - Hash cluster
  - Index cluster
- Information on following slides will be used in subsequent labs

Storage Guidelines

- Extents >= 40K
  - Occur on a 40K boundary
  - Install in tablespace CASE_LARGE_DATA or CASE_LARGE_INDEX
- Extents < 40K
  - Occur on a 4K boundary
  - Install in tablespace CASE_SMALL_DATA or CASE_SMALL_INDEX

Storage Guidelines

- Next extents are set to the greater of:
  - Space for one year's growth or
  - The extent boundary (as outlined in the previous slide)
- Next extents follow the 4k or 40k boundary rules
- PCTUSED
  - Put on the freelist when room for 5 rows
  - Maximum value of 90%.

5

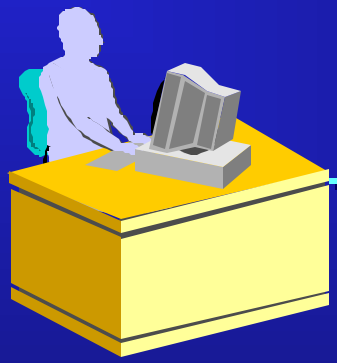
Organizing Storage

Objectives

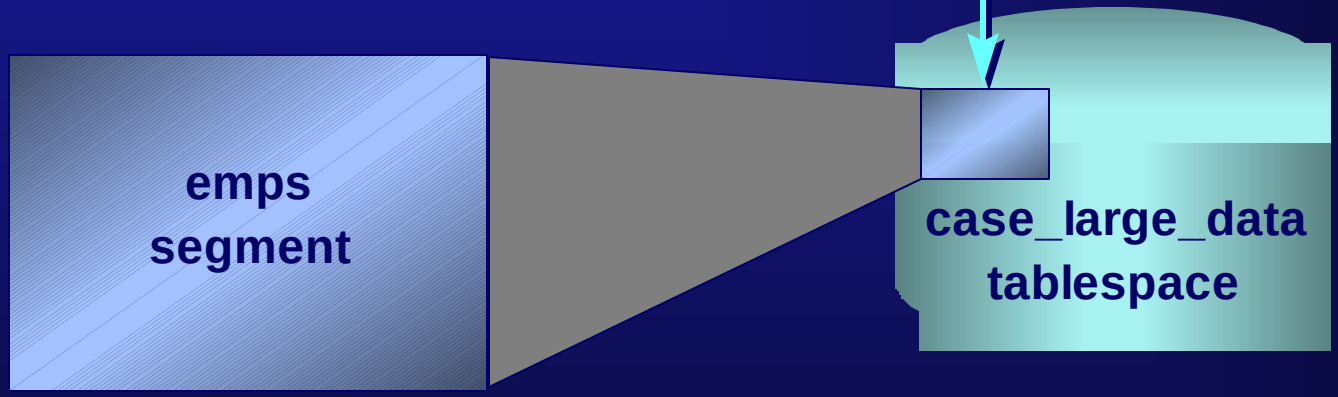
After completing this lesson, you should be able to do the following:

- Identify Oracle storage structures
- Use the STORAGE clause
- List methods used to size tables
- Select storage information from the data dictionary

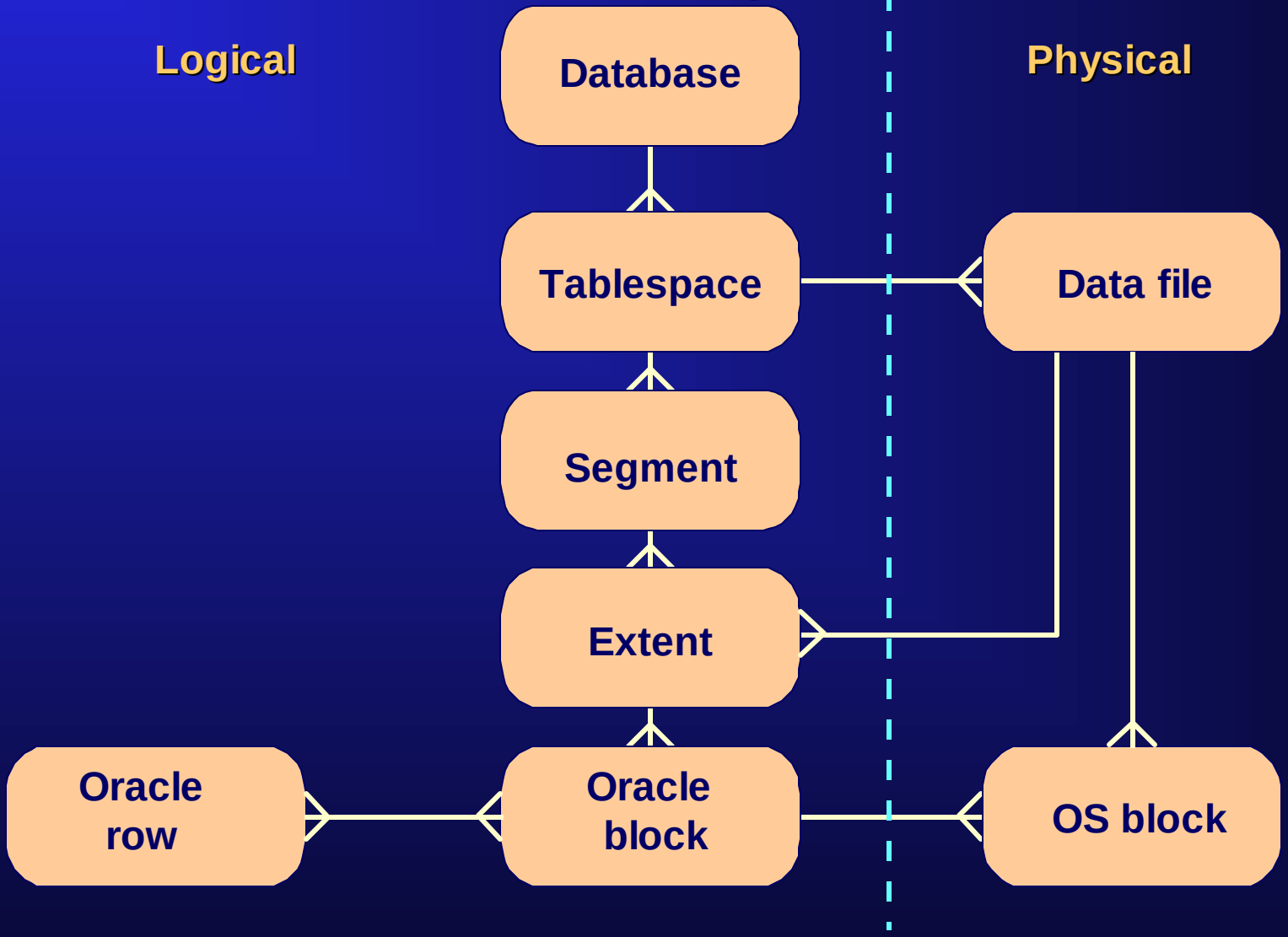
Organizing Storage



```
CREATE TABLE emps
 . . .
 TABLESPACE case_large_data;
```



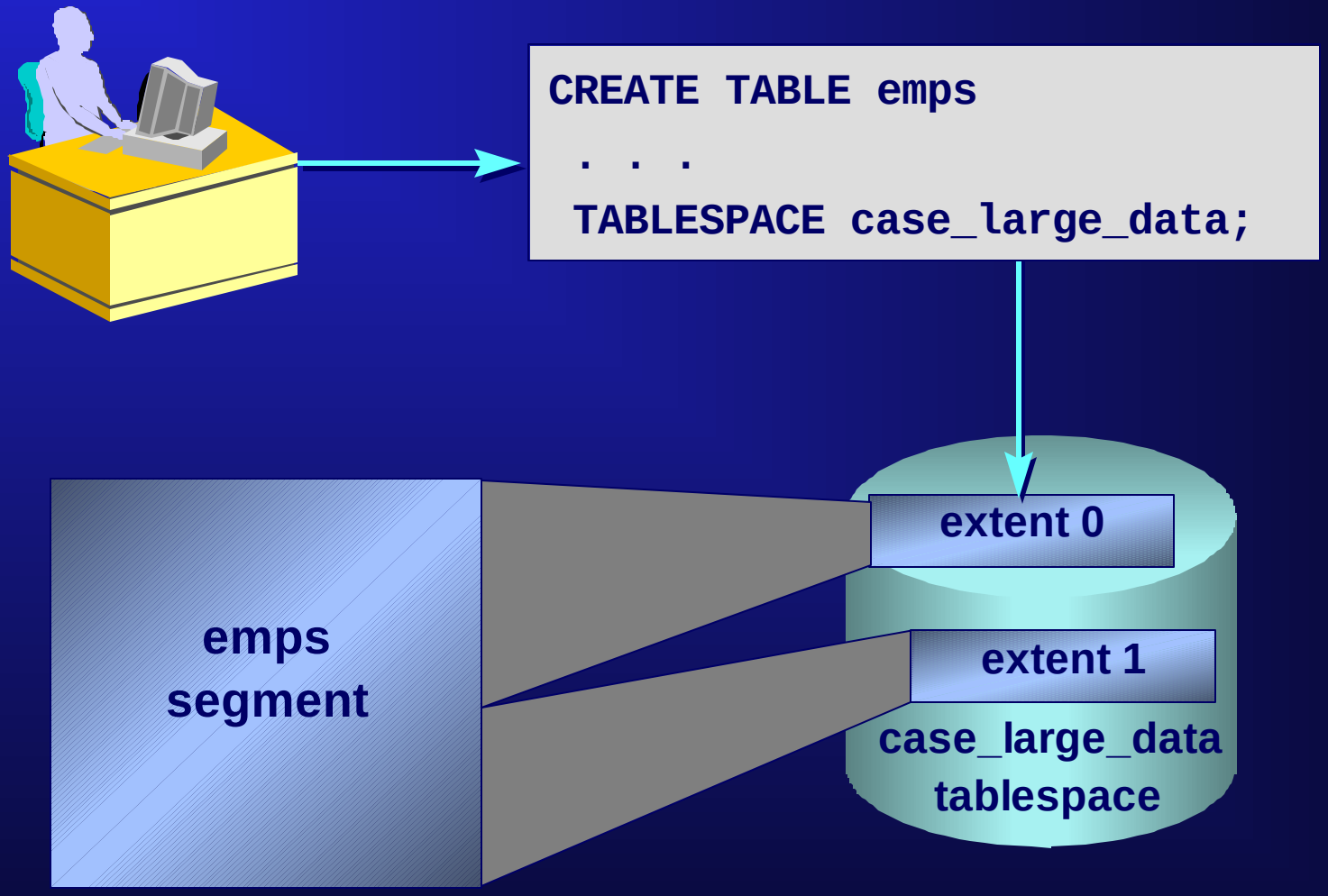
Database Storage Hierarchy



Segment Types

- Application segments
  - With data: Table, cluster, table partition, nested table, LOB, IOT, IOT overflow
  - Used to access data: Index, index partition, IOT, LOB index
- System segments: Rollback, temporary, bootstrap

Creating Segments and Extents



Storage Parameters

The following parameters influence the segment storage allocation:

- INITIAL: Size of the first extent in bytes
- NEXT: Starting size of subsequent extents
- PCTINCREASE: Percent increase of third and subsequent extents' size
- MINEXTENTS: Extents initially allocated
- MAXEXTENTS: Maximum extents allowed for a segment

Creating a Table

```
CREATE TABLE emps (
     empno NUMBER(4), name VARCHAR2(30),
     deptno NUMBER(3), hire_date DATE )
  STORAGE
    ( INITIAL         2M
      NEXT            500K
      PCTINCREASE 0
      MINEXTENTS  1
      MAXEXTENTS  50 )
  TABLESPACE case_large_data;
```

Temporary Tables

- The rows are private to the session.

```
CREATE GLOBAL TEMPORARY TABLE employee_temp
AS SELECT * FROM employee;
```

- Tables retain data only for the duration of a transaction or session.

```
ON COMMIT PRESERVE ROWS;
```

- DML locks are not acquired on the data.
- DMLs do not generate redo logs.

Creating a Table: Guidelines

- Separate segments by type, size, and update frequency.
- Use standard extent sizes for tables to reduce tablespace fragmentation.
- Make extent sizes a multiple of:
  - 5 * DB_BLOCK_SIZE
  - DB_FILE_MULTIBLOCK_READ_COUNT
- Use the CACHE clause for frequently used, small tables.

Estimating Table Size

- Case tools
- Sample data
  - Load a sample of rows.
  - Use DBMS_SPACE.UNUSED_SPACE to determine the number of blocks used.
  - Extrapolate to the actual number of rows.

Changing Storage Parameters

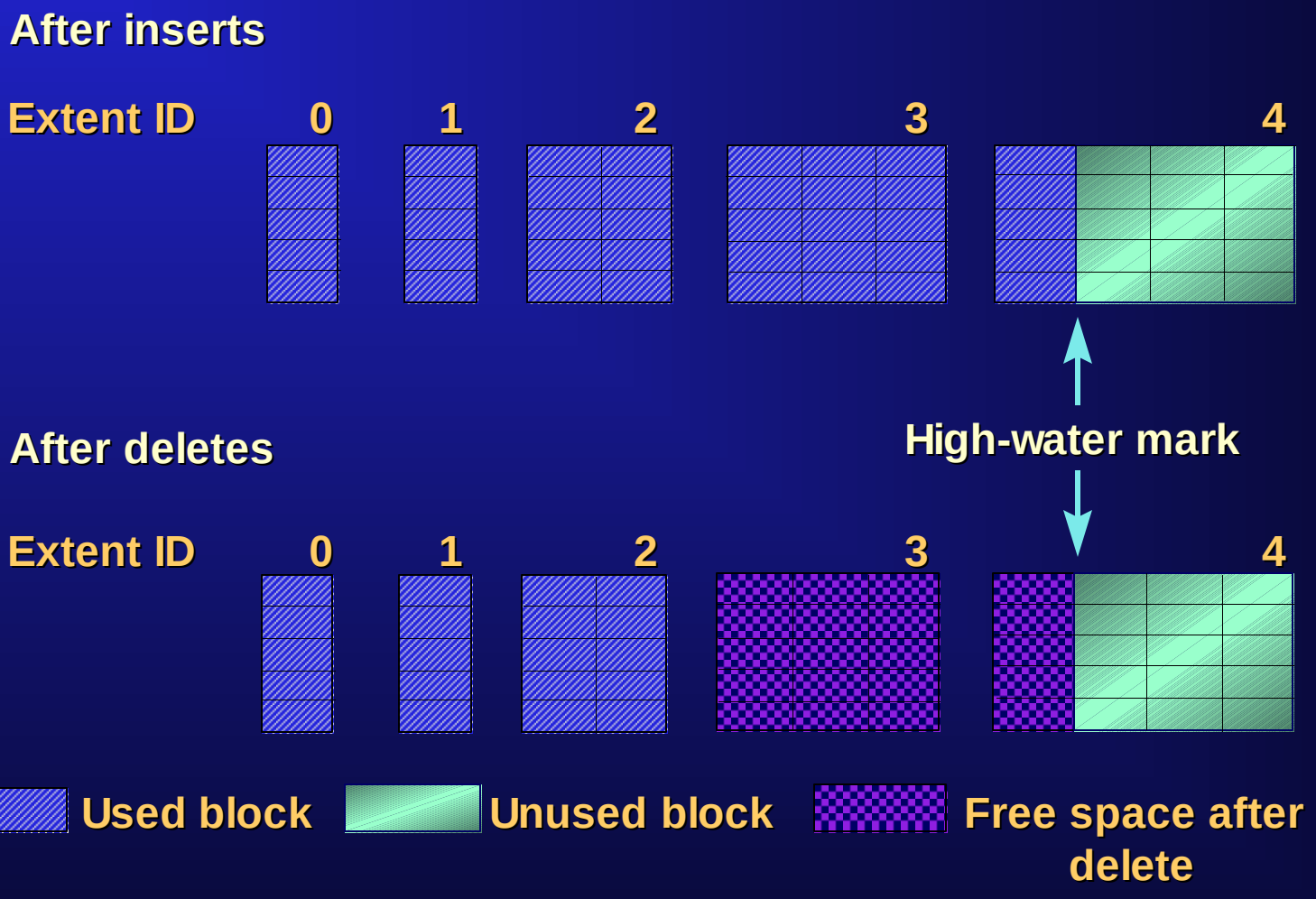
```
ALTER TABLE emps
  STORAGE (
     NEXT          1M
     PCTINCREASE   0
     MINEXTENTS    1
     MAXEXTENTS    100 );
```

Manually Allocating Extents

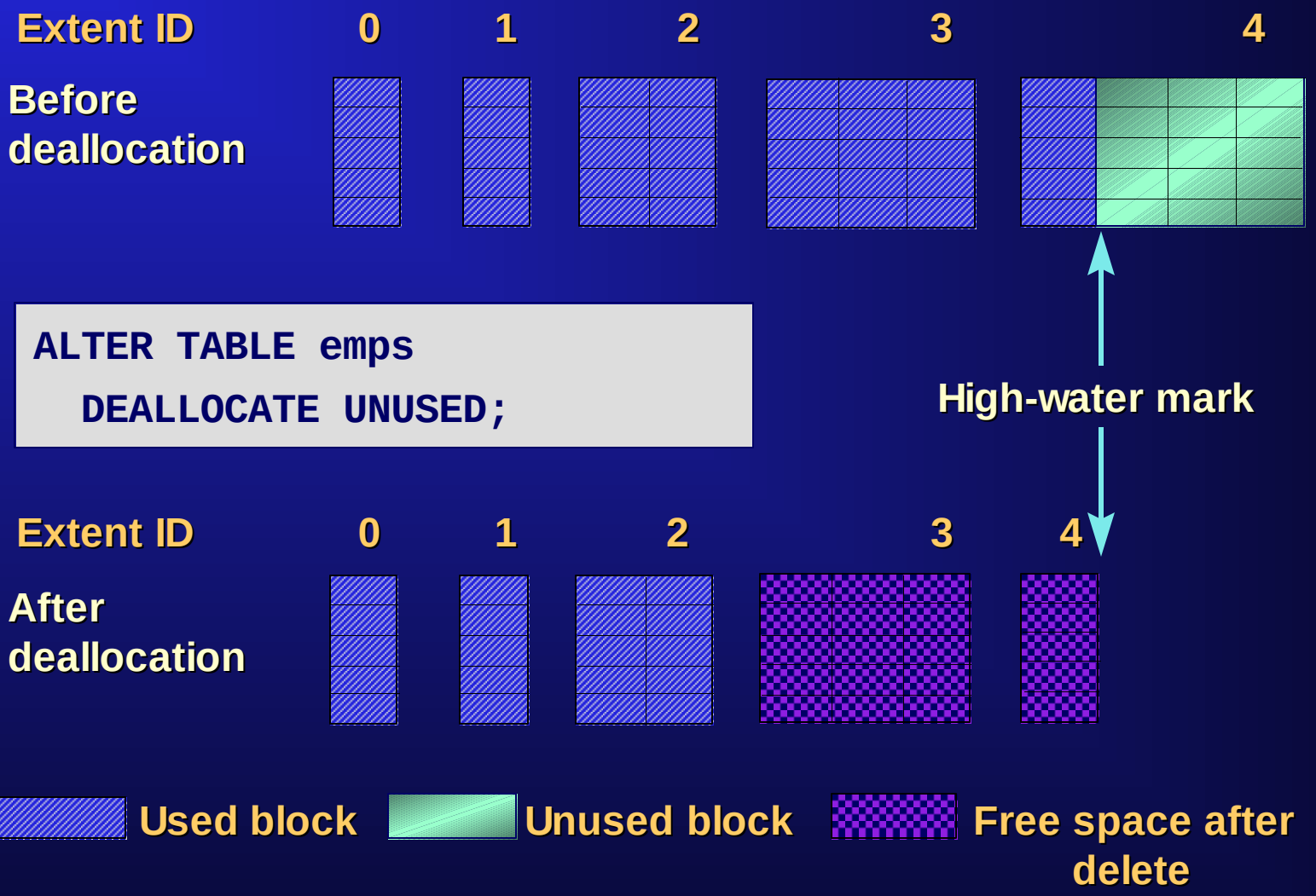
```
ALTER TABLE emps
   ALLOCATE EXTENT
      (  SIZE 500K
         DATAFILE '/DISK3/DATA01.DBF' );
```

- To control the distribution of extents of a table across files
- Before loading data in bulk to avoid dynamic extension of tables

High-Water Mark



Deallocating Unused Space

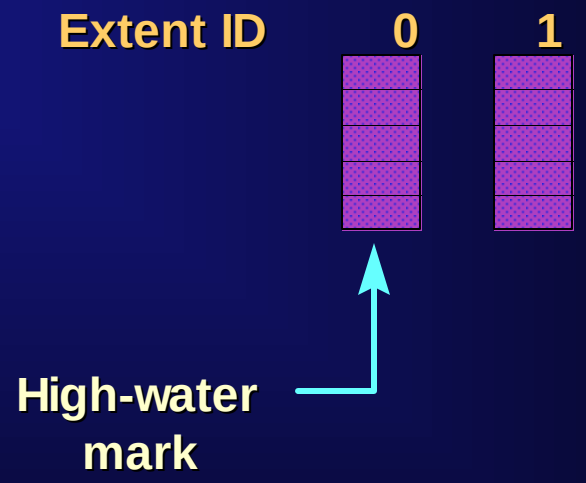


Truncating a Table

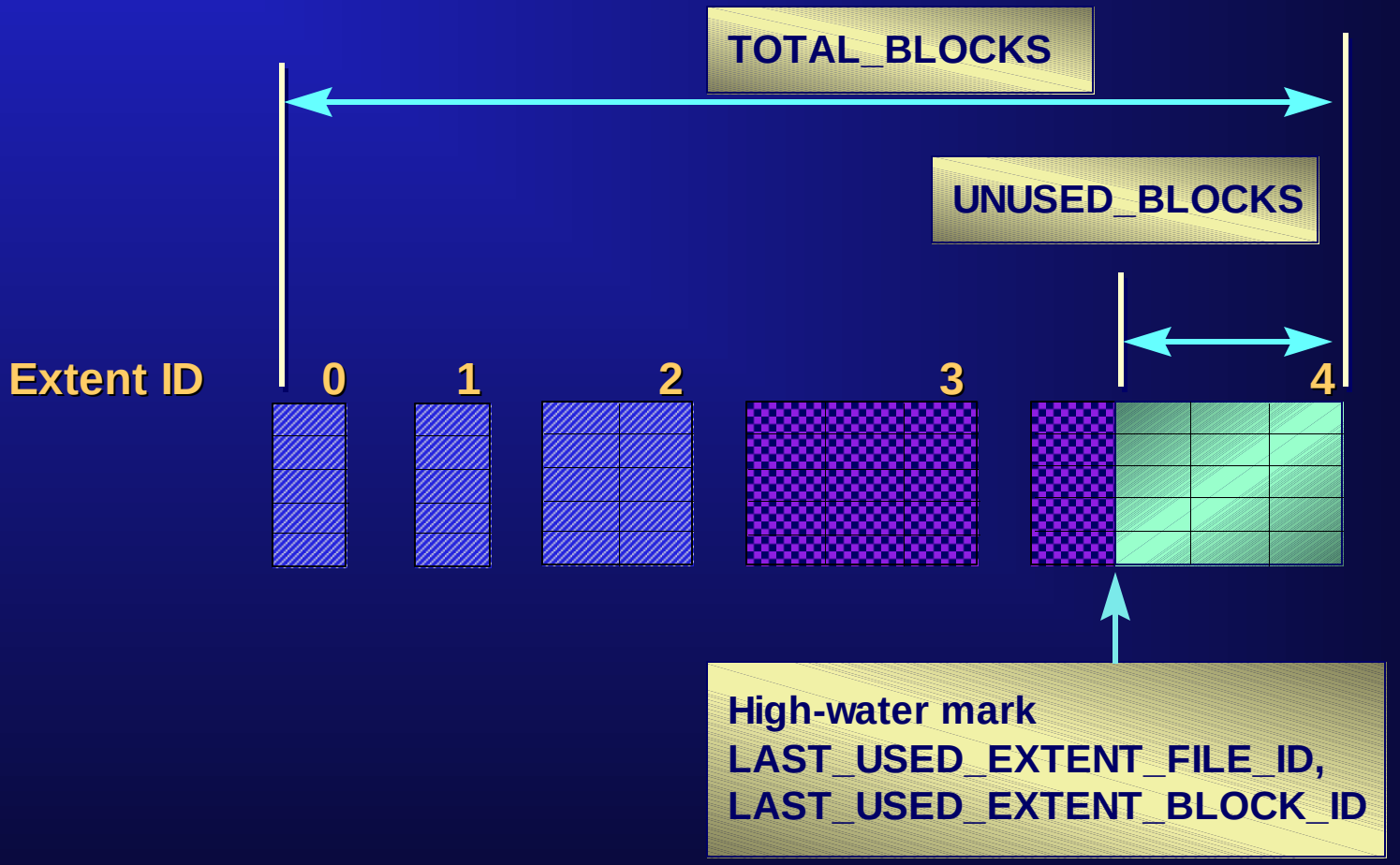
Syntax:

```
TRUNCATE TABLE [schema.] table
[{DROP | REUSE} STORAGE]
```

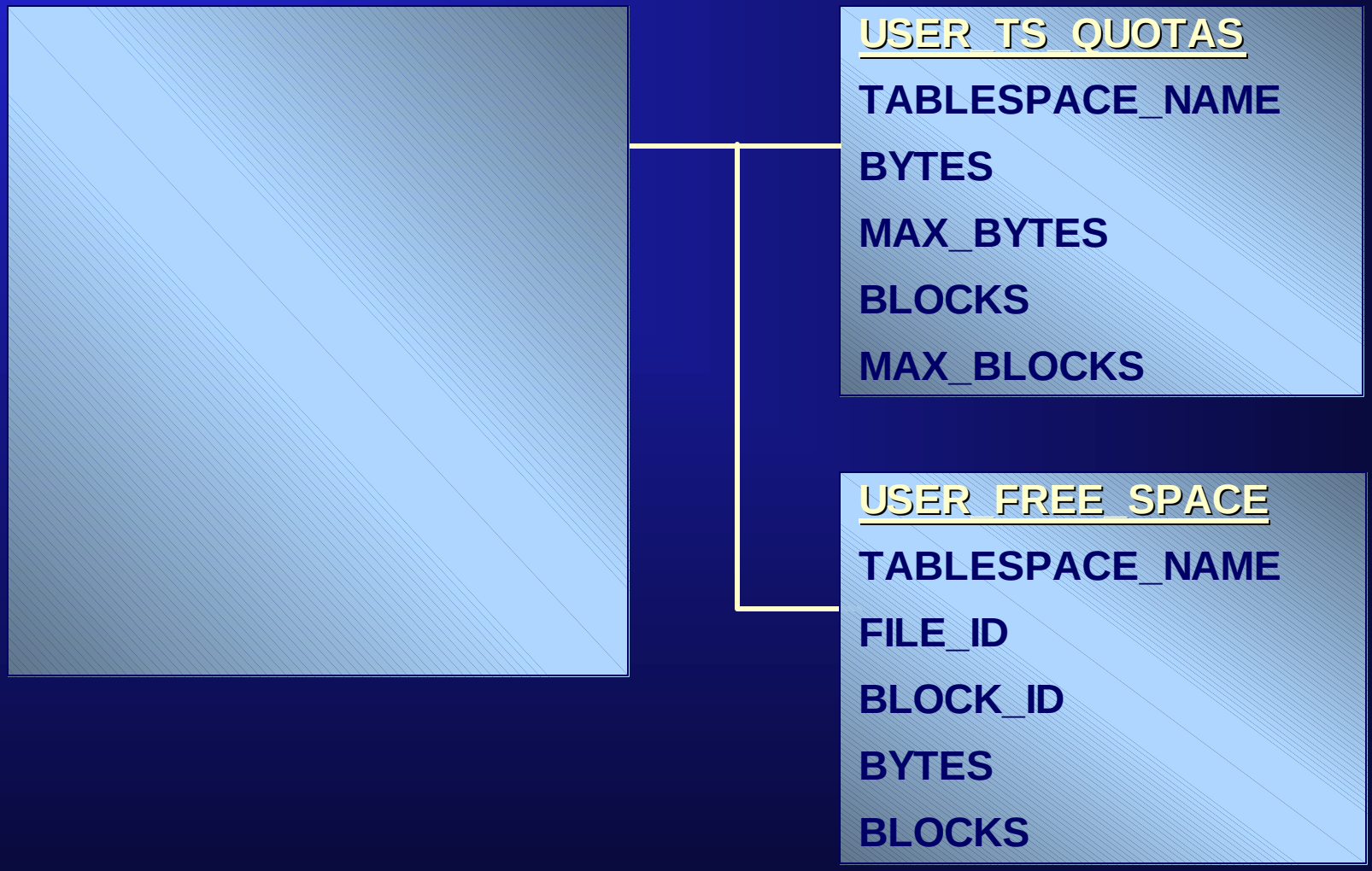
```
TRUNCATE TABLE emps;
```



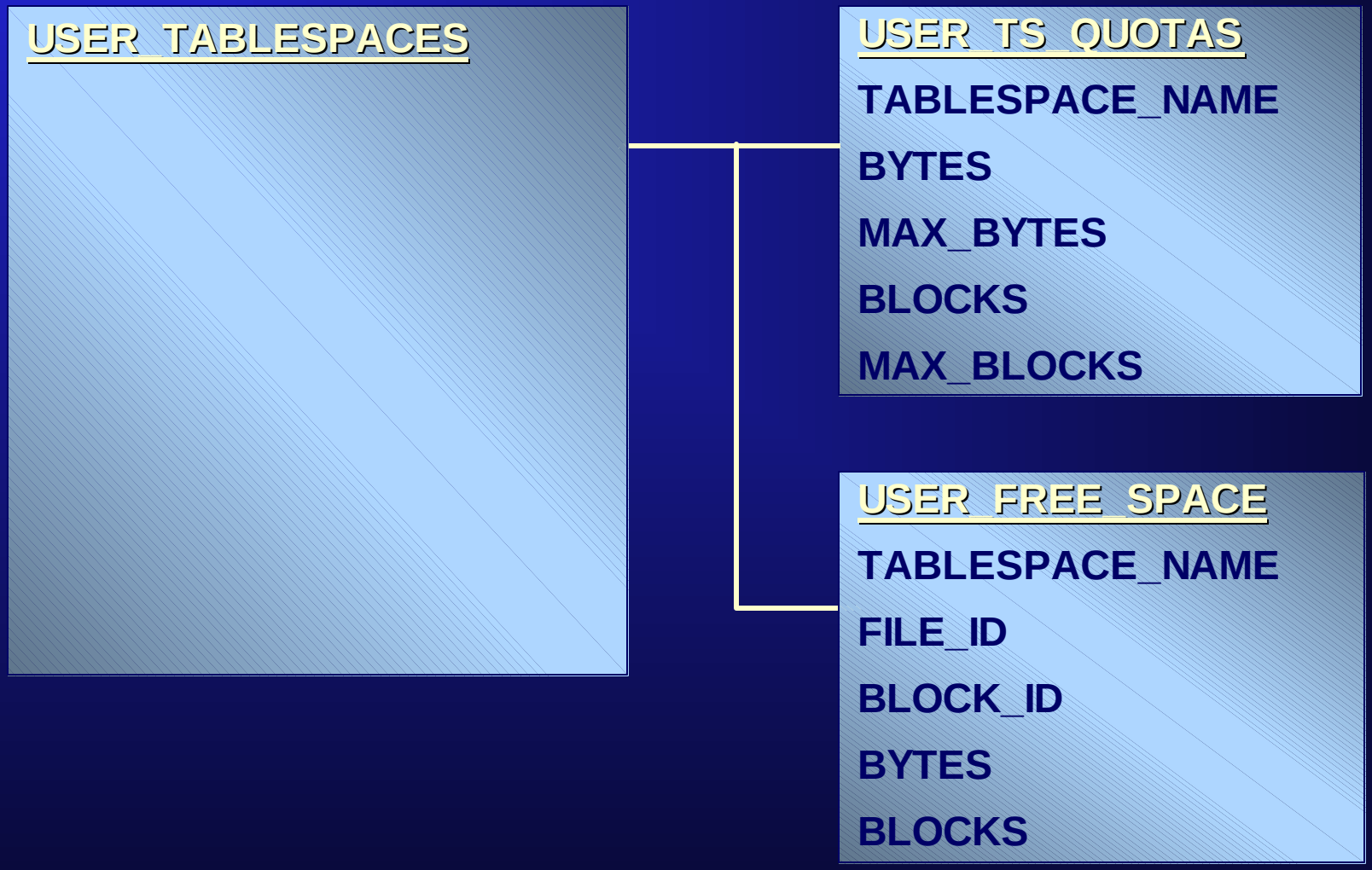
Finding the High-Water Mark: DBMS_SPACE.UNUSED_SPACE



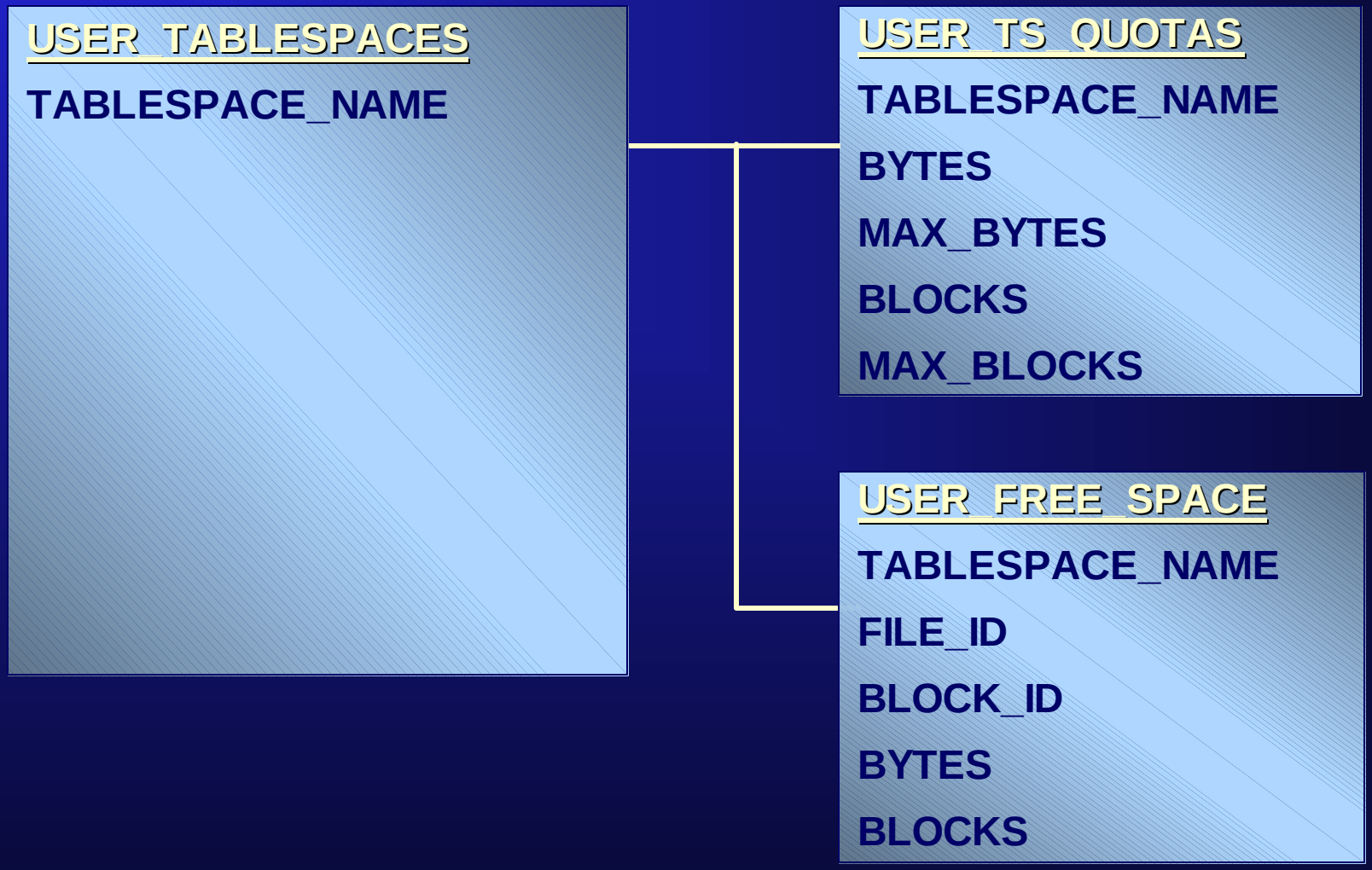
Retrieving Tablespace Information



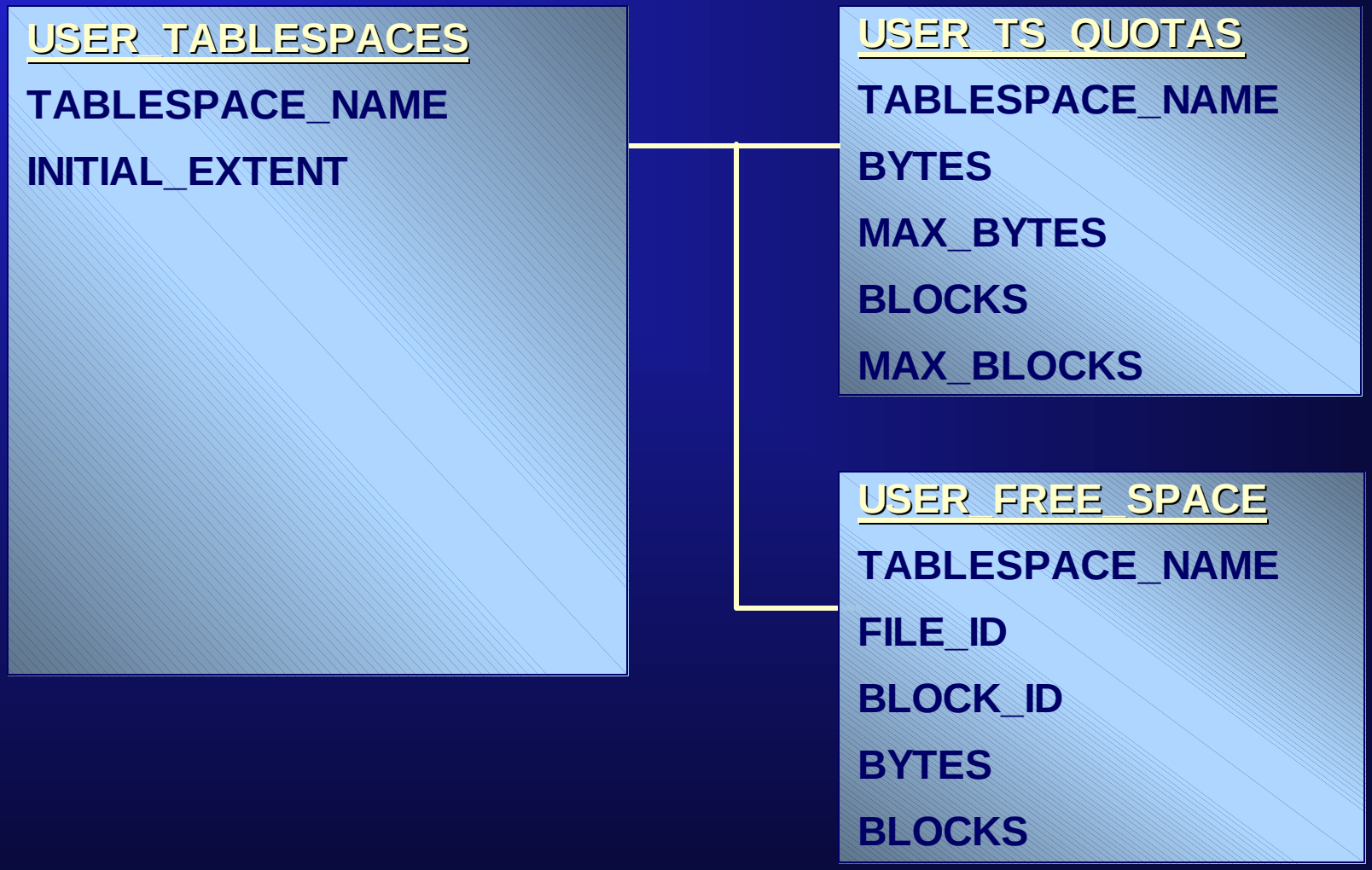
Retrieving Tablespace Information



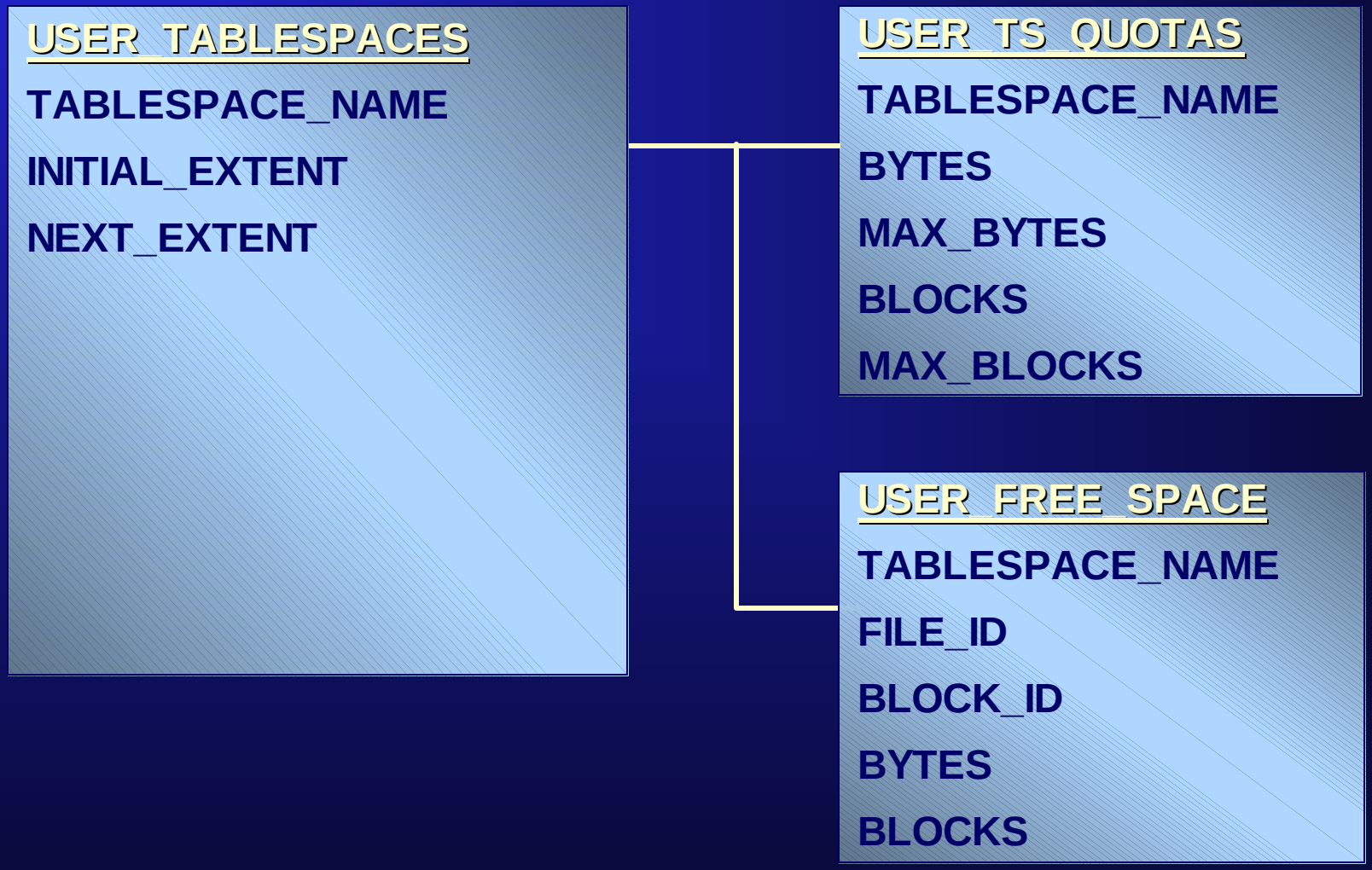
Retrieving Tablespace Information



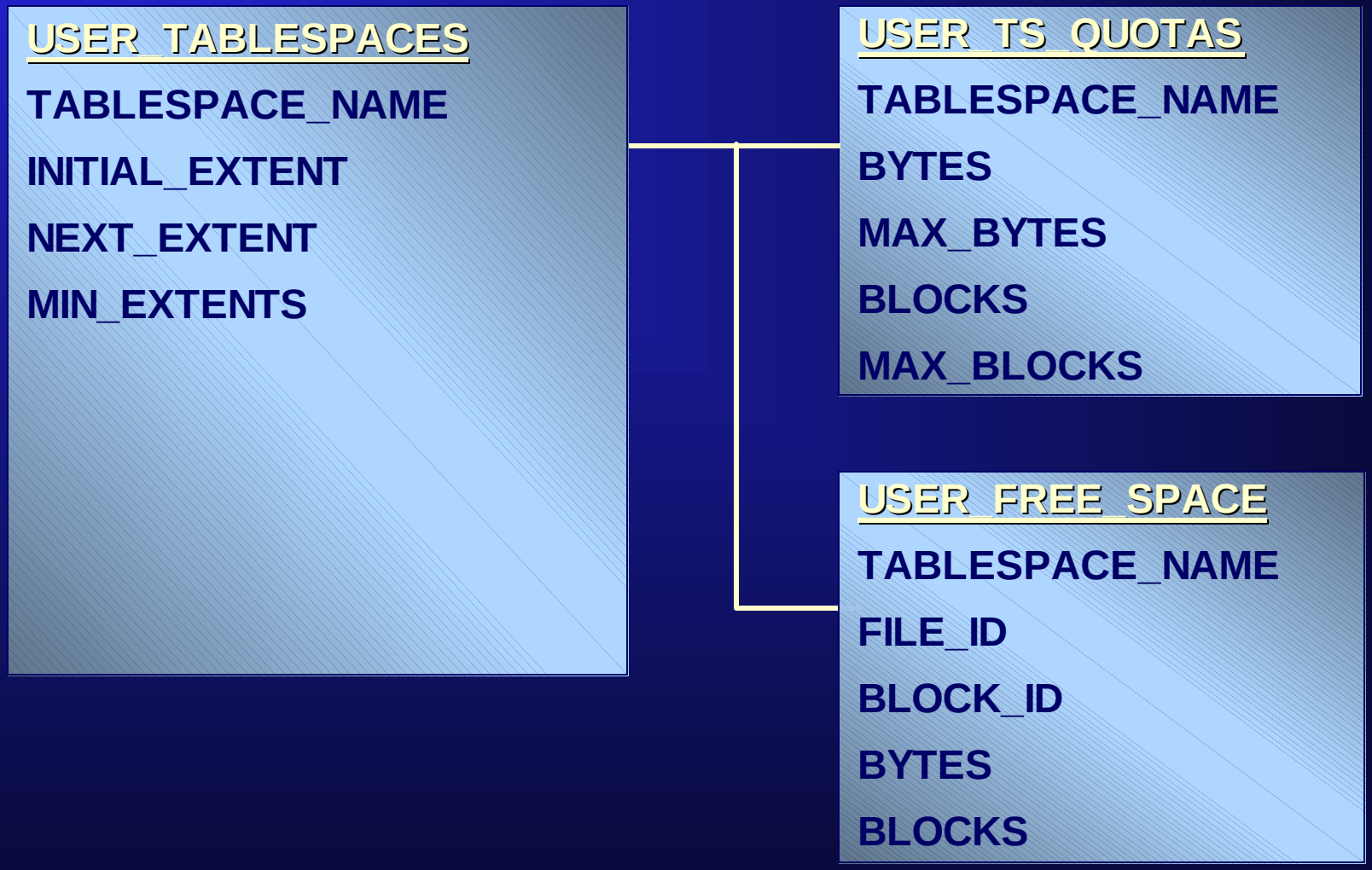
Retrieving Tablespace Information



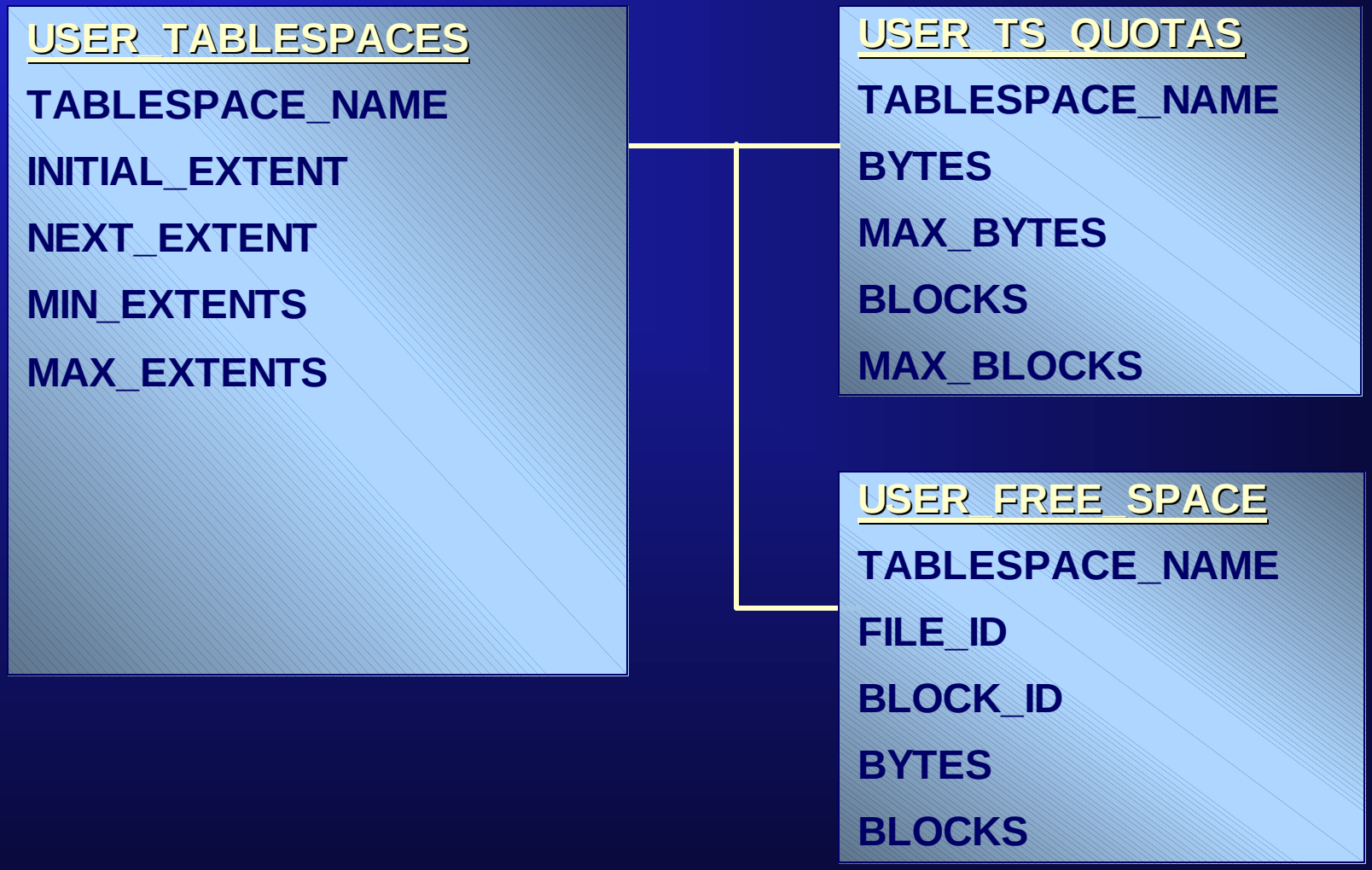
Retrieving Tablespace Information



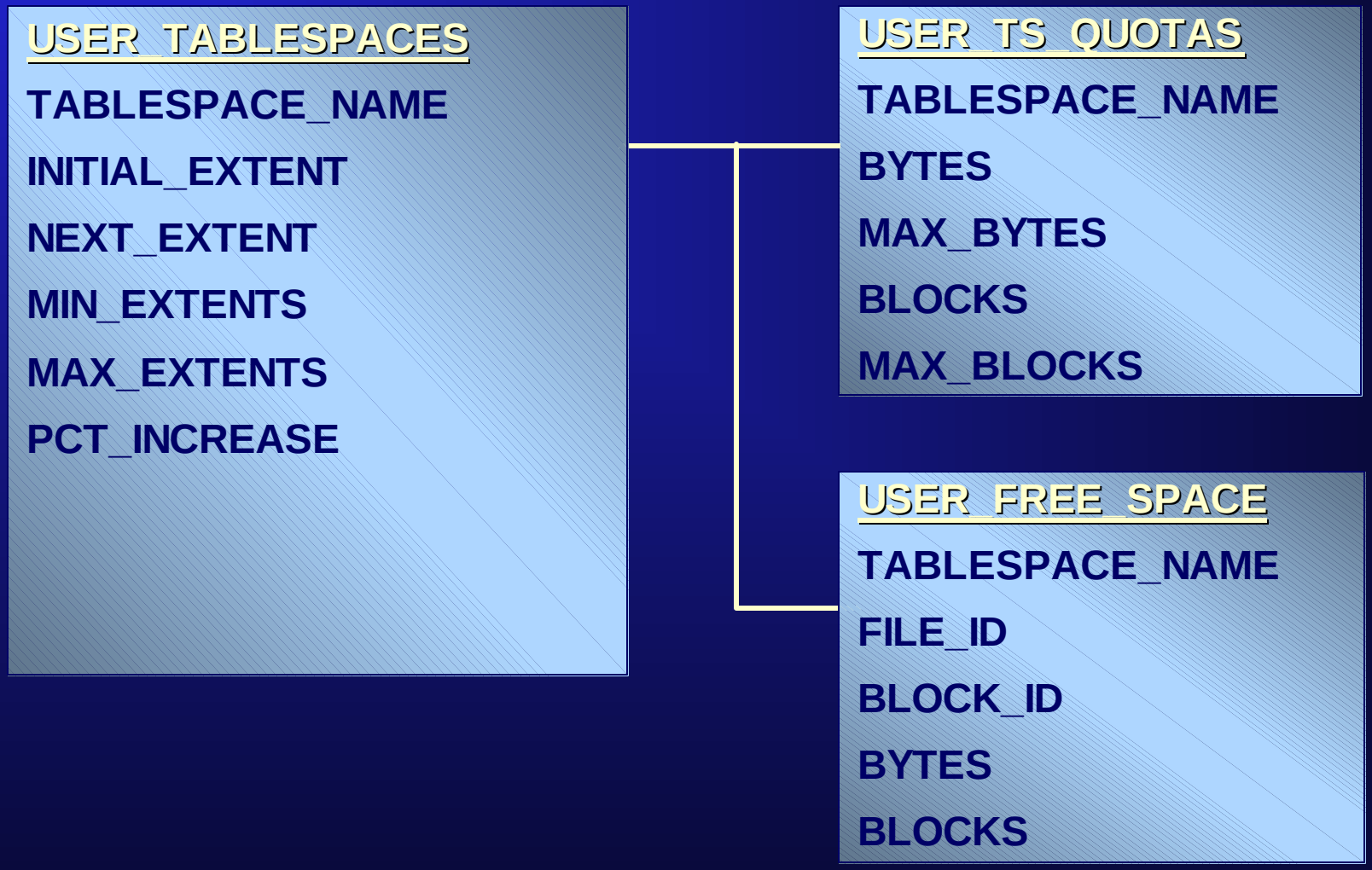
Retrieving Tablespace Information



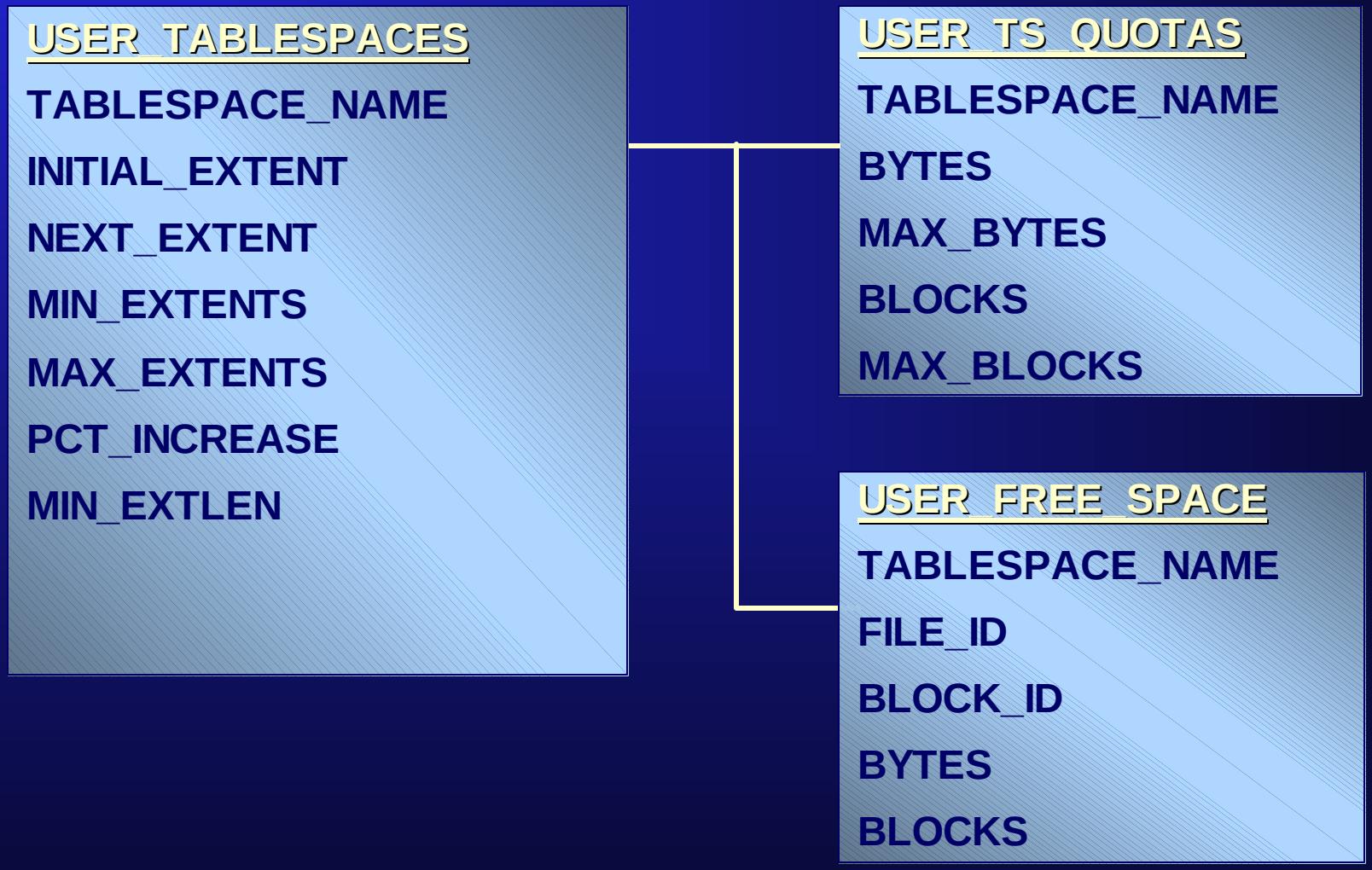
Retrieving Tablespace Information



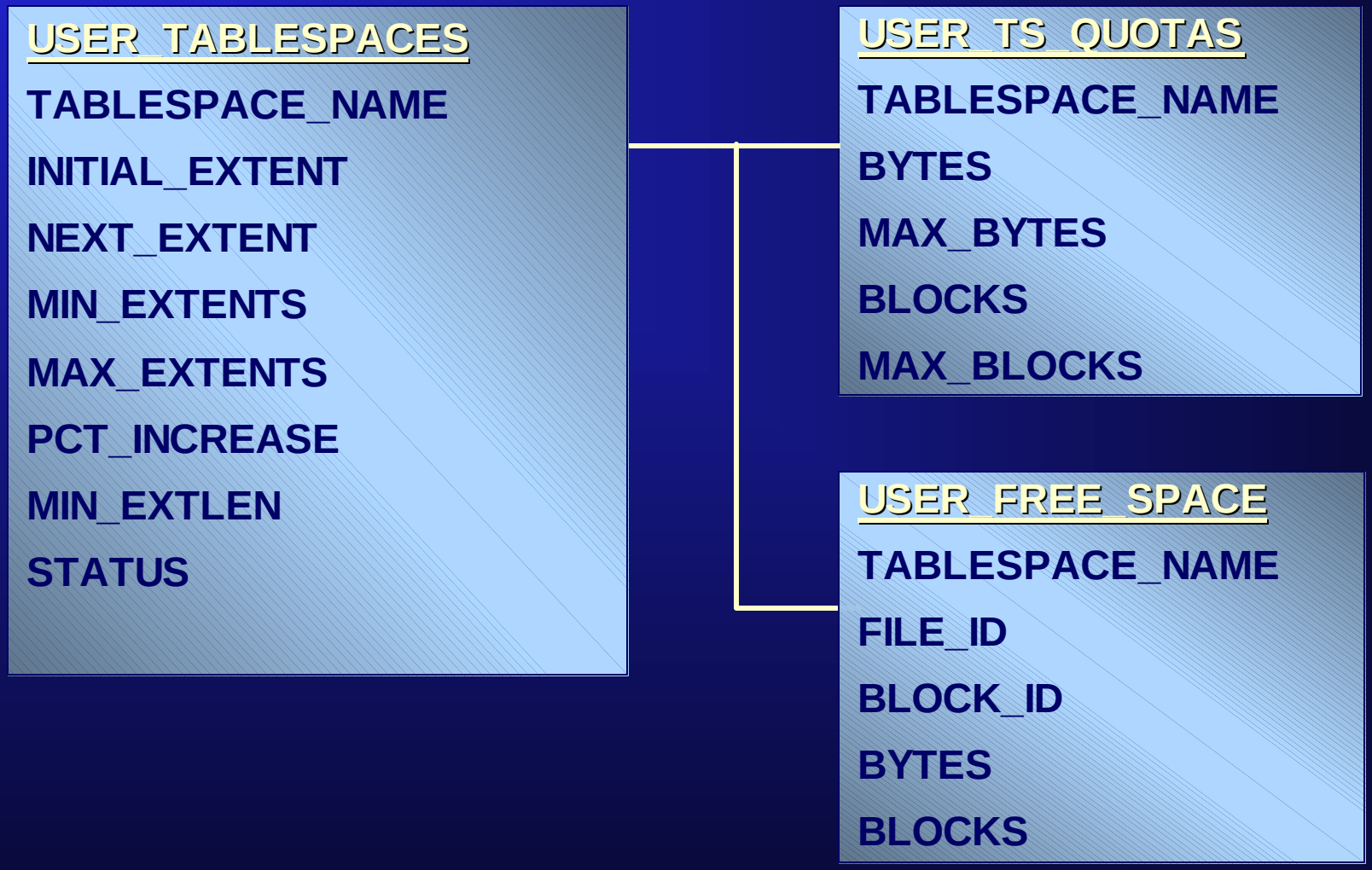
Retrieving Tablespace Information



Retrieving Tablespace Information



Retrieving Tablespace Information



Retrieving Tablespace Information

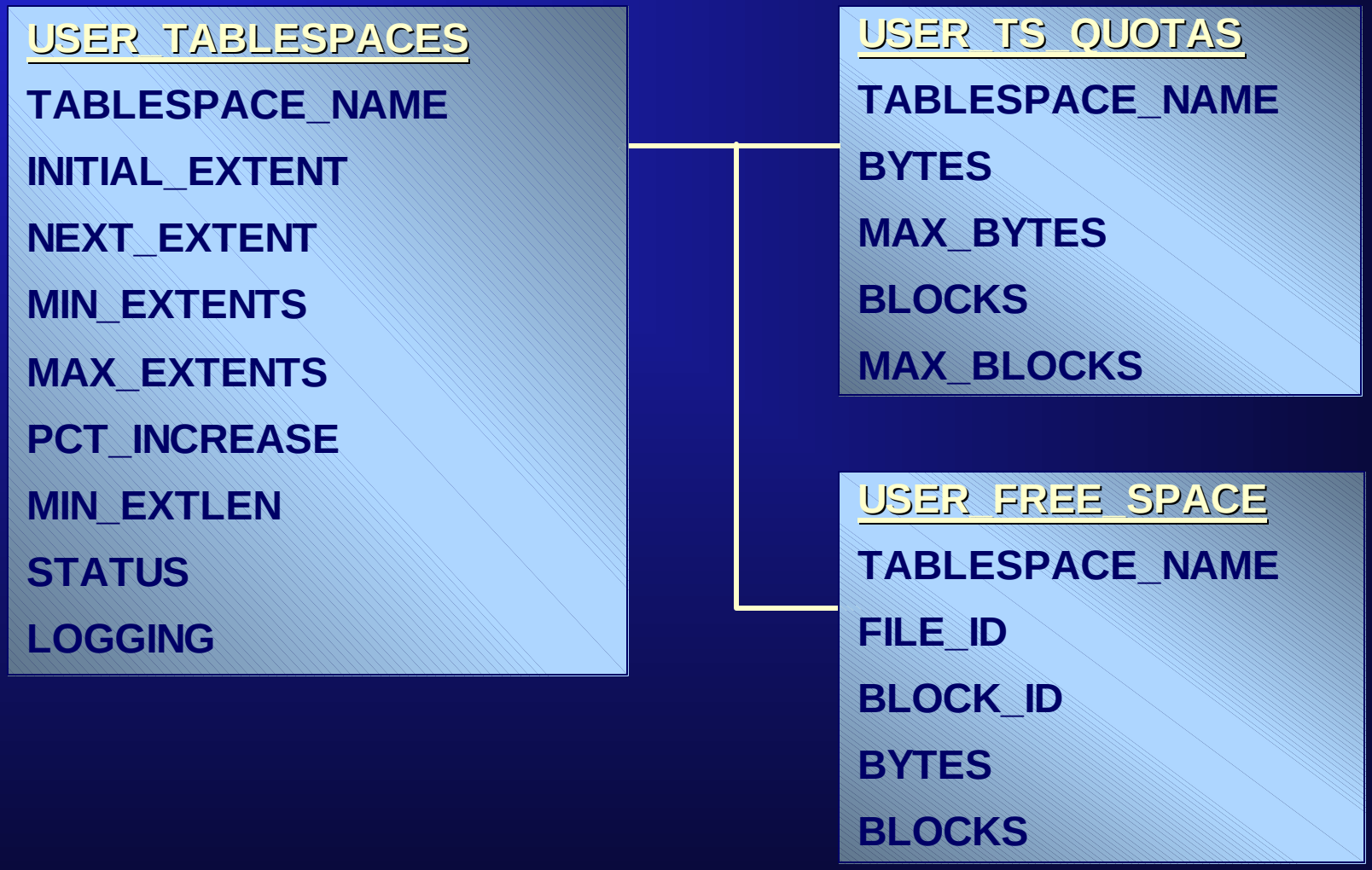
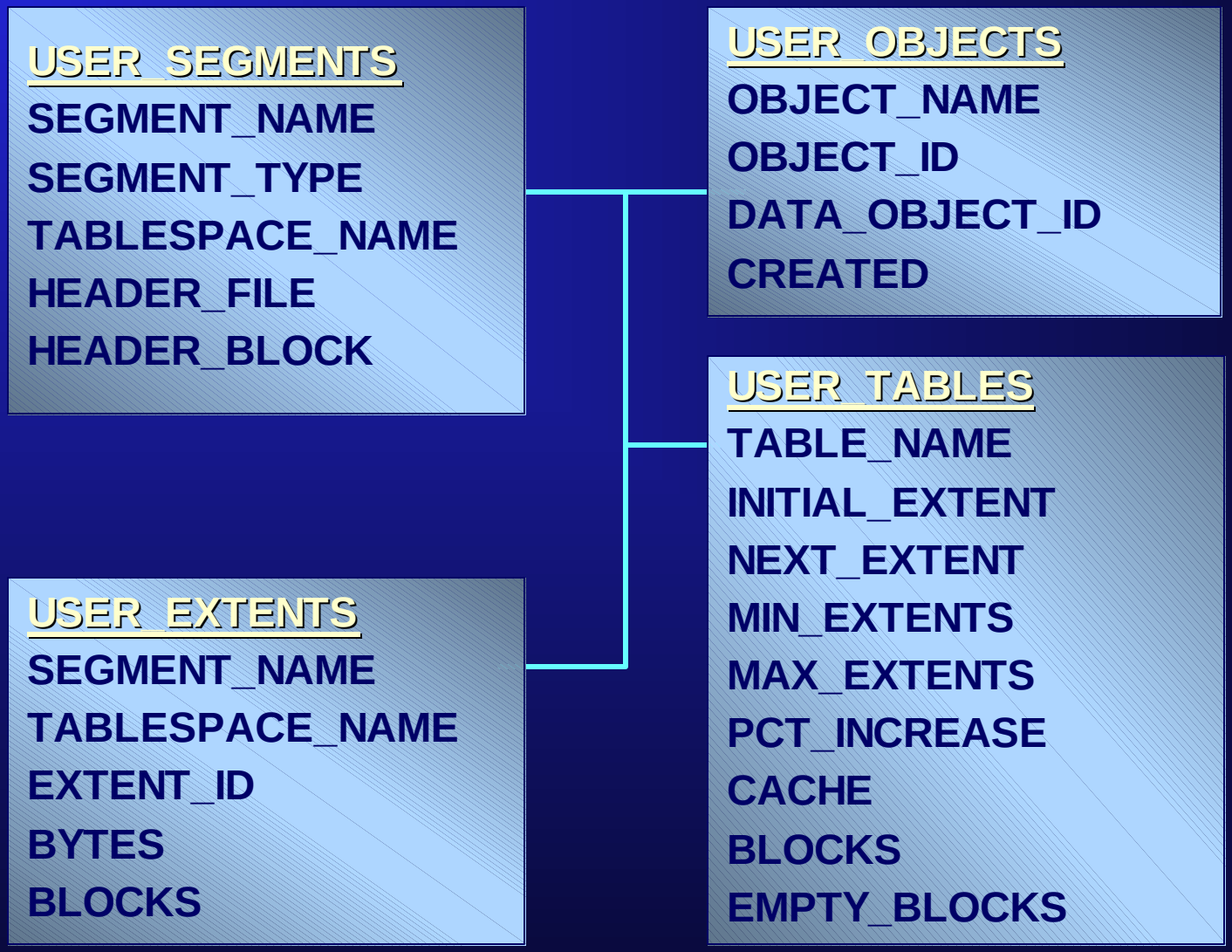


Table and Segment Information



Summary

In this lesson, you should have learned how to:

- Create a table with appropriate storage parameters
- Control table storage
- Use the DBMS_SPACE package

Practice Overview

This practice covers the following topics:

- Creating tables with the specified storage parameters
- Altering tables with the specified storage parameters
- Querying the data dictionary to view storage parameters

6

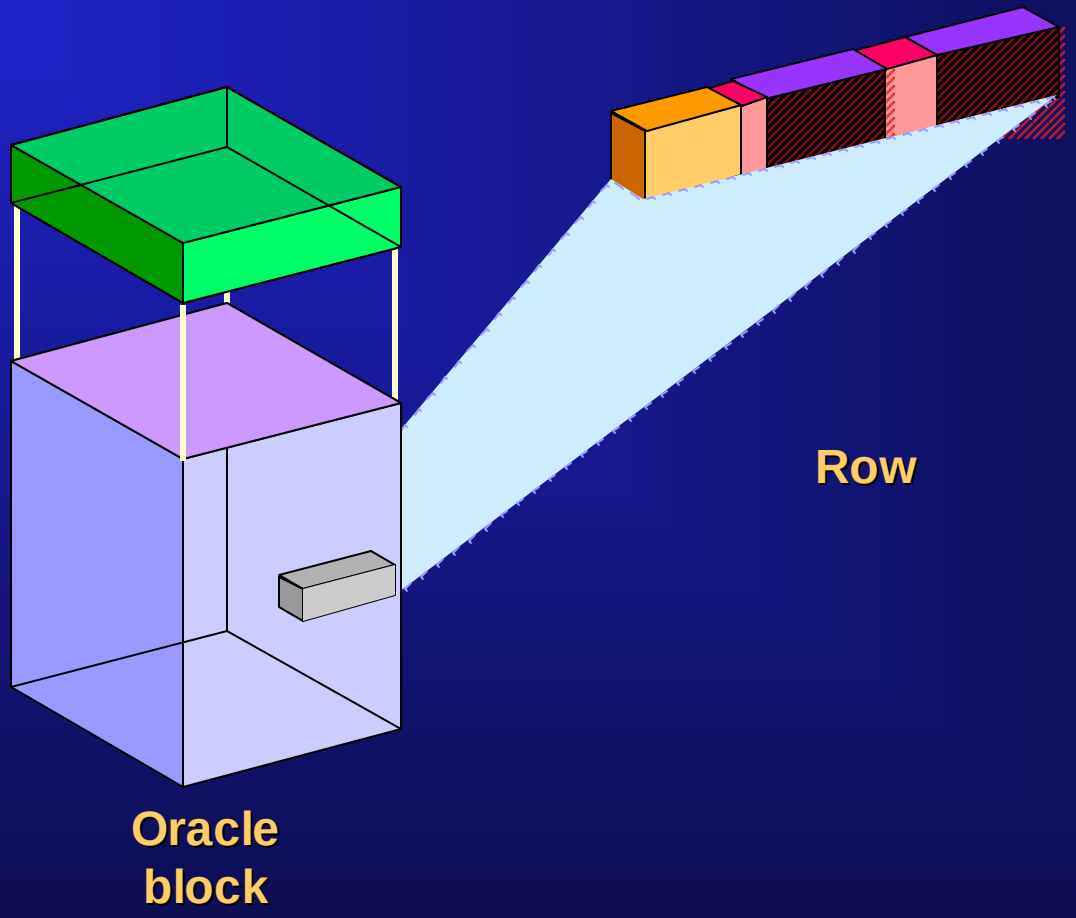
Controlling Use of Block Space

Objectives

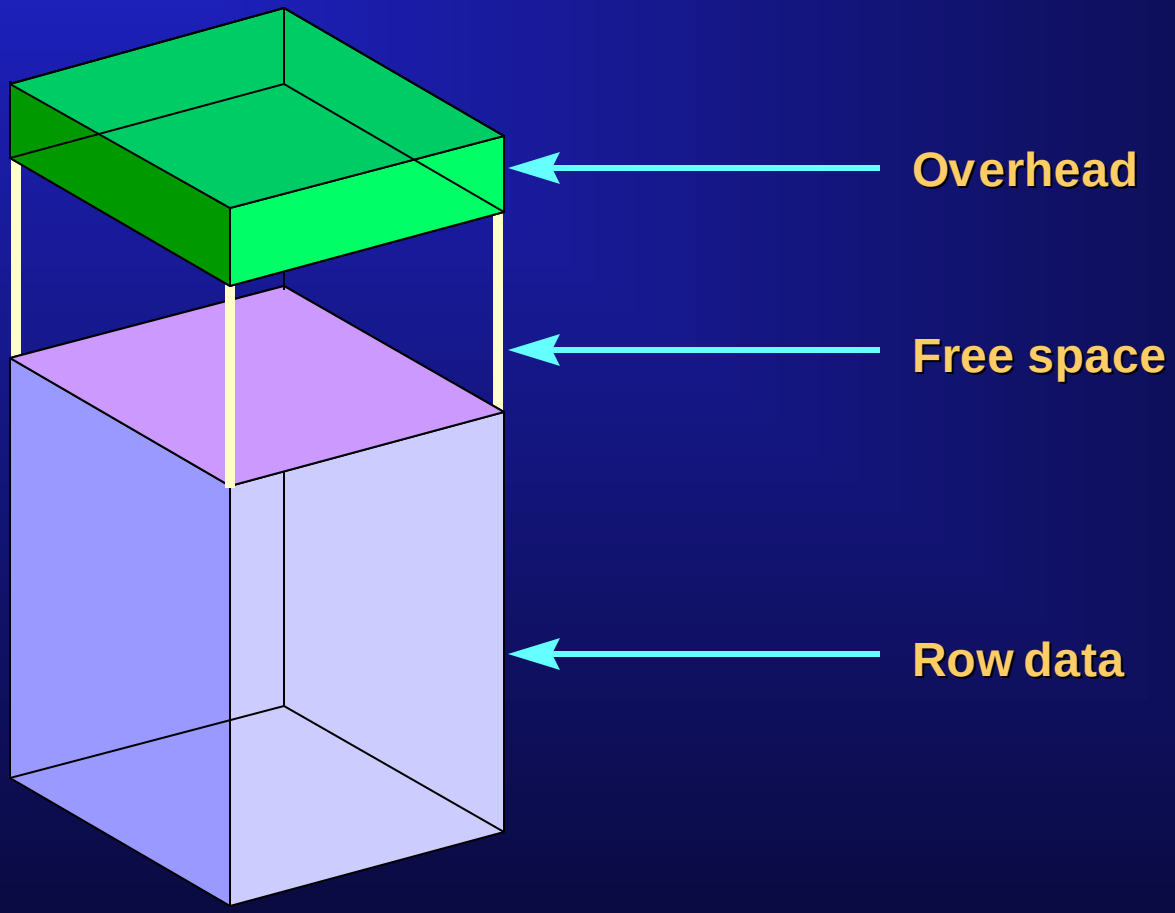
After completing this lesson, you should be able to do the following:

- Set block utilization parameters
- Calculate appropriate values for block utilization parameters
- Correct migrated rows
- List the components of rowids

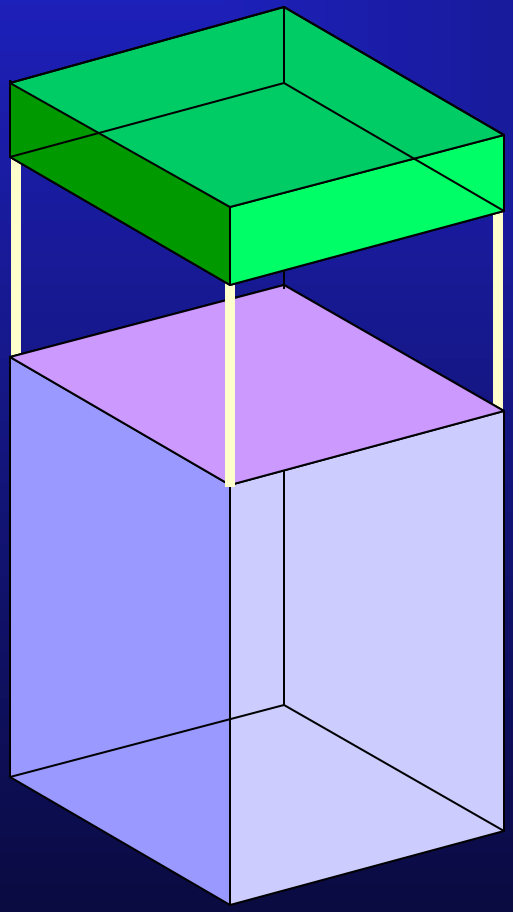
Overview



Database Block Contents



Block Space Utilization Parameters



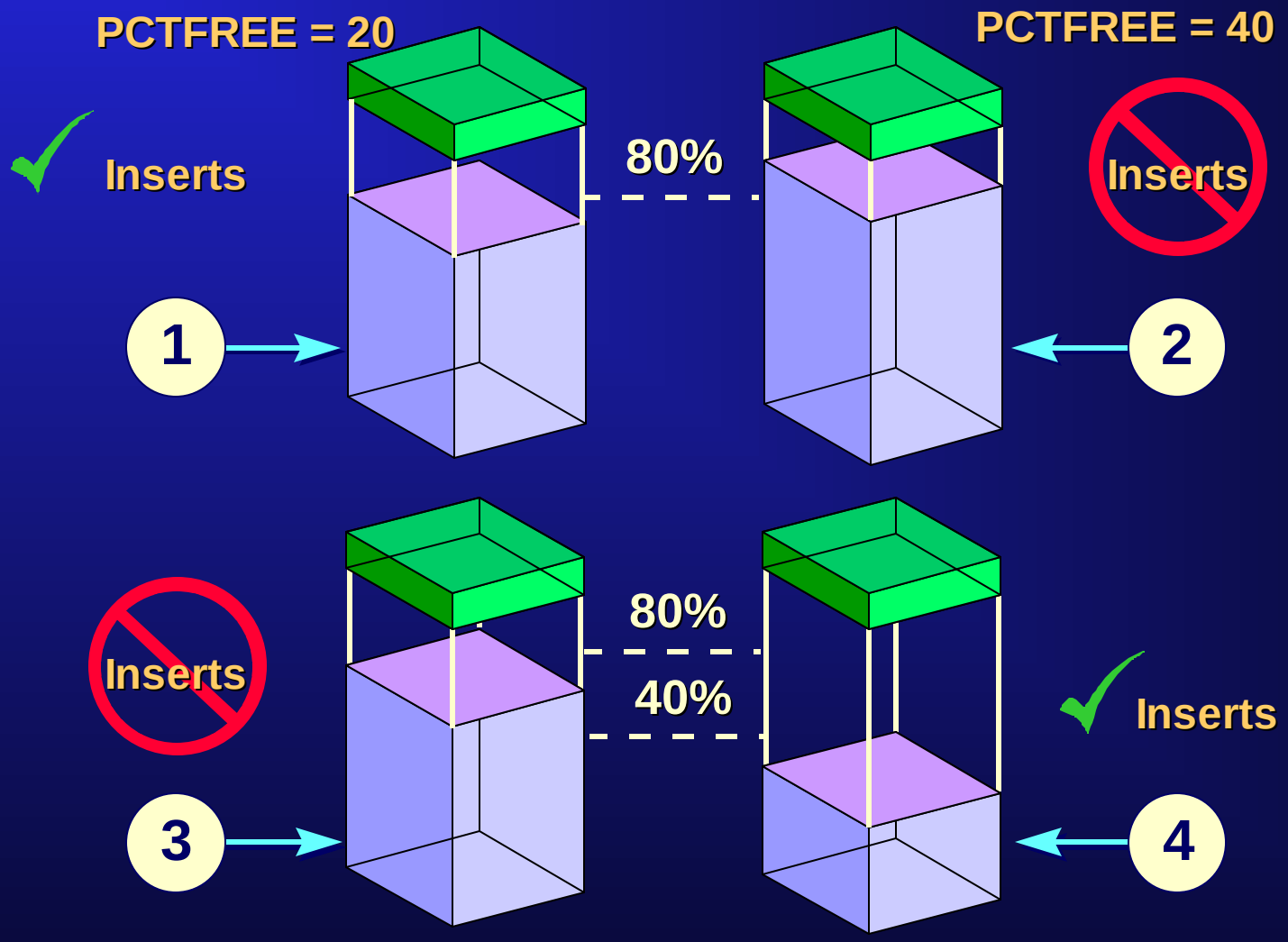
INITRANS

MAXTRANS

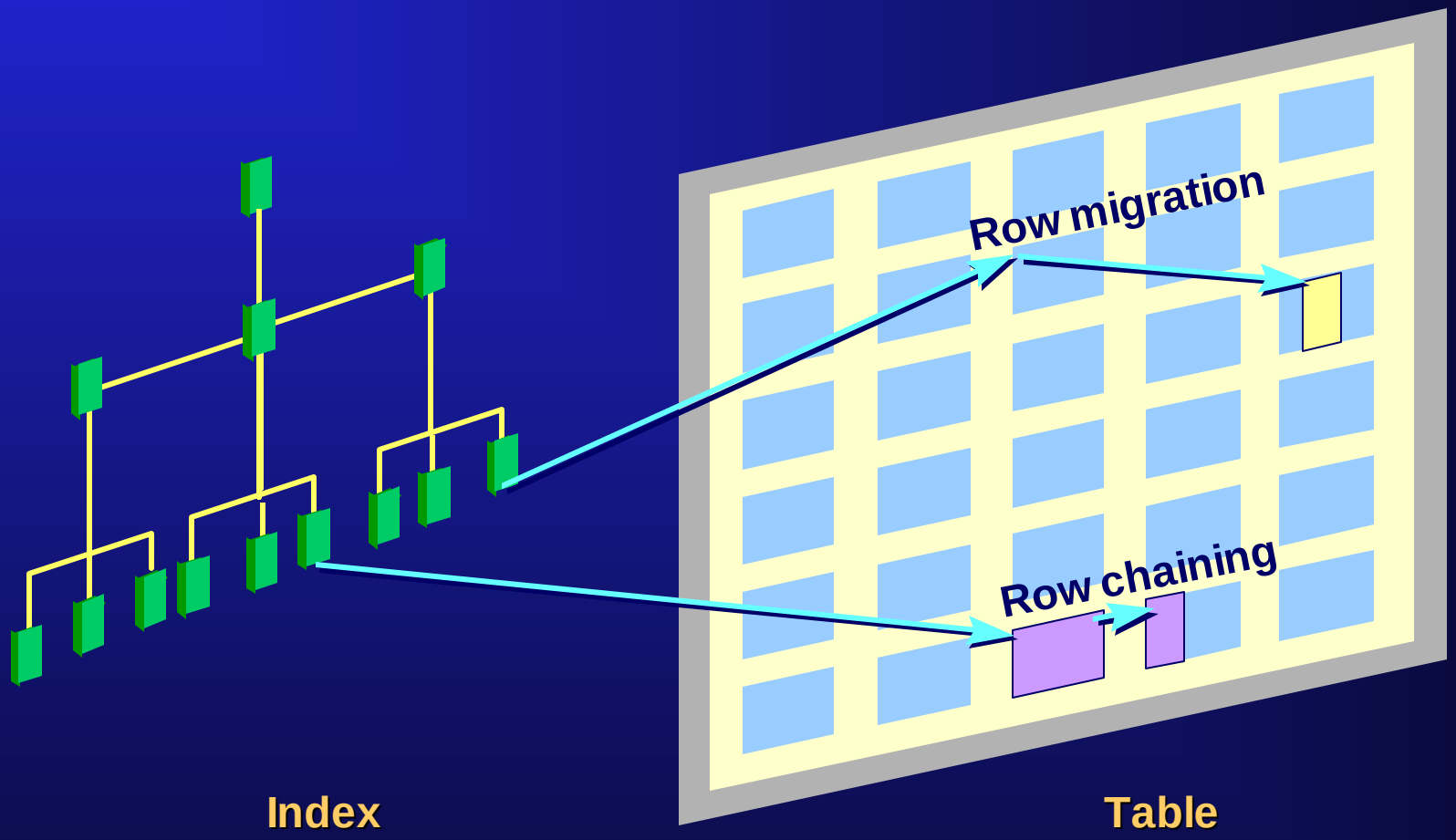
PCTFREE

PCTUSED

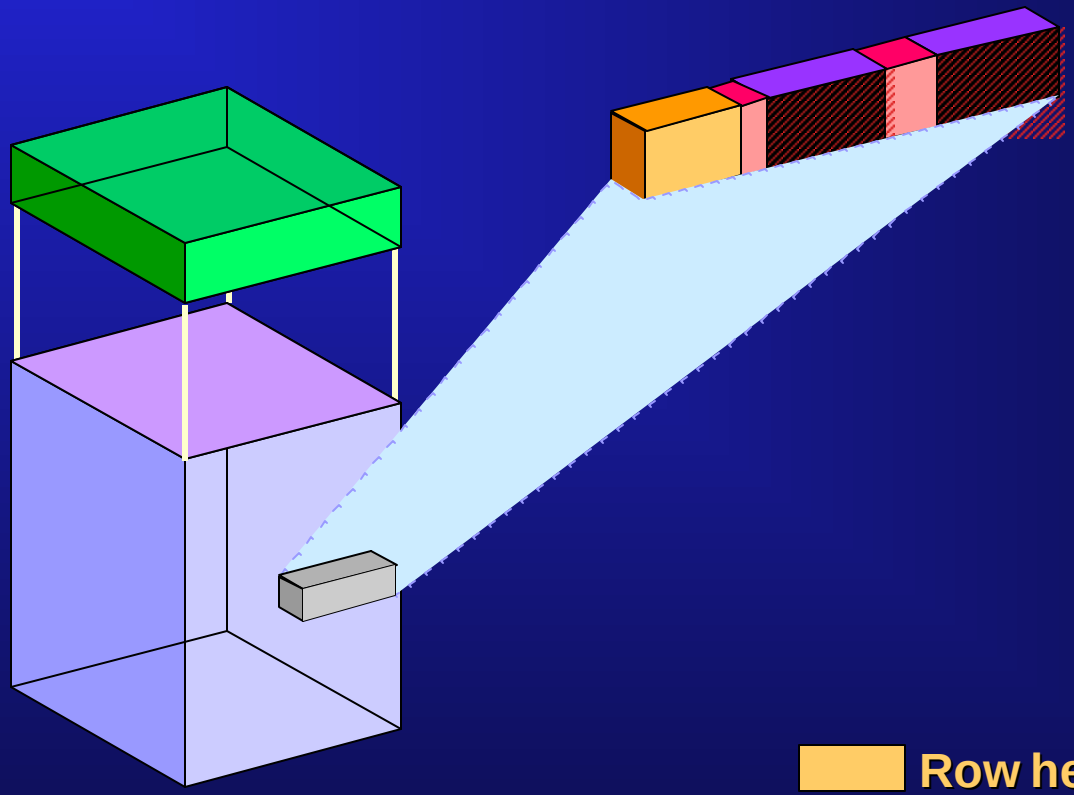
PCTFREE and PCTUSED



Migration and Chaining



Structure of a Row



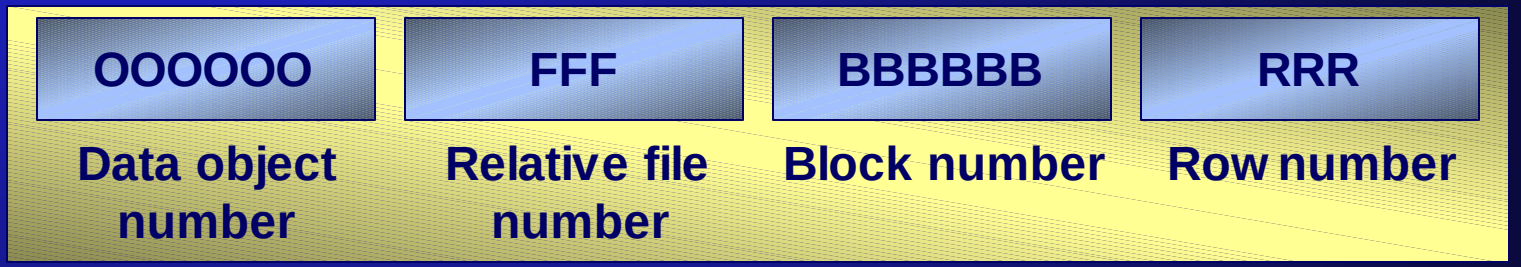
ROWID

- Pseudocolumn that uniquely identifies a row
- Most efficient way to locate a row

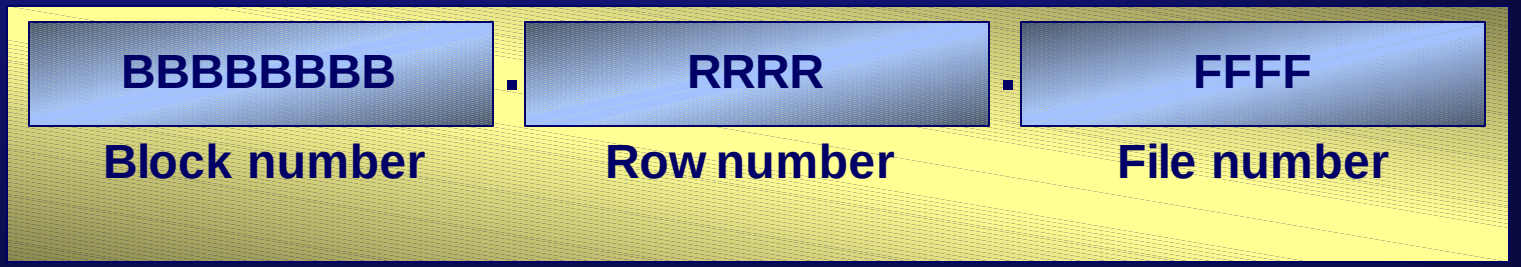
```
 5  SELECT  ename,   rowid
 6    INTO  v_ename, v_rowid
. . .
11  UPDATE  emp
12    SET   ename = 'WAYNE'
13    WHERE rowid = v_rowid;
```

ROWID Formats

Extended ROWID Format



Restricted ROWID Format



Setting Block Utilization Parameters

- To improve space utilization
- To minimize the possibility of row migration

```
CREATE TABLE emps
  . . .
  INITRANS 2
  MAXTRANS 50
  PCTFREE 20
  PCTUSED 50;
```

```
ALTER TABLE emps
  INITRANS 2
  MAXTRANS 50
  PCTFREE 30
  PCTUSED 60;
```

PCTFREE and PCTUSED Guidelines

- **PCTFREE**
  - Default 10
  - Zero if no UPDATE activity
  - $$\frac{\text{(Average row size - Initial row size)}}{\text{Average row size}} * 100$$
- **PCTUSED**
  - Default 40
  - Set if rows deleted
  - $$100 - \text{PCTFREE} - \frac{\text{(rows * Avg. row size * 100)}}{\text{Available data space}}$$

DBMS_ROWID Package

Commonly used functions:

Function Name	Description
ROWID_CREATE	Creates a ROWID from individual components
ROWID_OBJECT	Returns the object identifier for a ROWID
ROWID_RELATIVE_FNO	Returns the relative file number for a ROWID
ROWID_BLOCK_NUMBER	Returns the block number for a ROWID
ROWID_ROW_NUMBER	Returns the row number for a ROWID
ROWID_TO_ABSOLUTE_FNO	Returns the absolute file number for a ROWID
ROWID_TO_EXTENDED	Converts a ROWID from restricted to extended
ROWID_TO_RESTRICTED	Converts a ROWID from extended to restricted

DBMS_ROWID

```
SQL> SELECT  id, ROWID,
  2    DBMS_ROWID.ROWID_OBJECT(ROWID) AS OBJECT,
  3    DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) AS “RELATIVE FILE”,
  4    DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) AS BLOCK
  5  FROM  dept;

DEPTNO ROWID                  OBJECT RELATIVE FILE     BLOCK
------ ------------------ --------- ------------- ---------
    10 AAAAupAABAAABpjAAA      2985             1      6755
    20 AAAAupAABAAABpjAAB      2985             1      6755
    30 AAAAupAABAAABpjAAC      2985             1      6755
    40 AAAAupAABAAABpjAAD      2985             1      6755
```

Summary

Summary

In this lesson, you should have learned how to:

Summary

In this lesson, you should have learned how to:

- Create a table with appropriate block utilization parameters

Summary

In this lesson, you should have learned how to:

- Create a table with appropriate block utilization parameters
- Use DBMS_ROWID to retrieve the components of the rowid

Practice Overview

This practice covers the following topics:

- Computing the values of PCTFREE and PCTUSED and altering the tables
- Displaying the PCTFREE and PCTUSED values for a table from data dictionary views

7

Selecting an Index Type

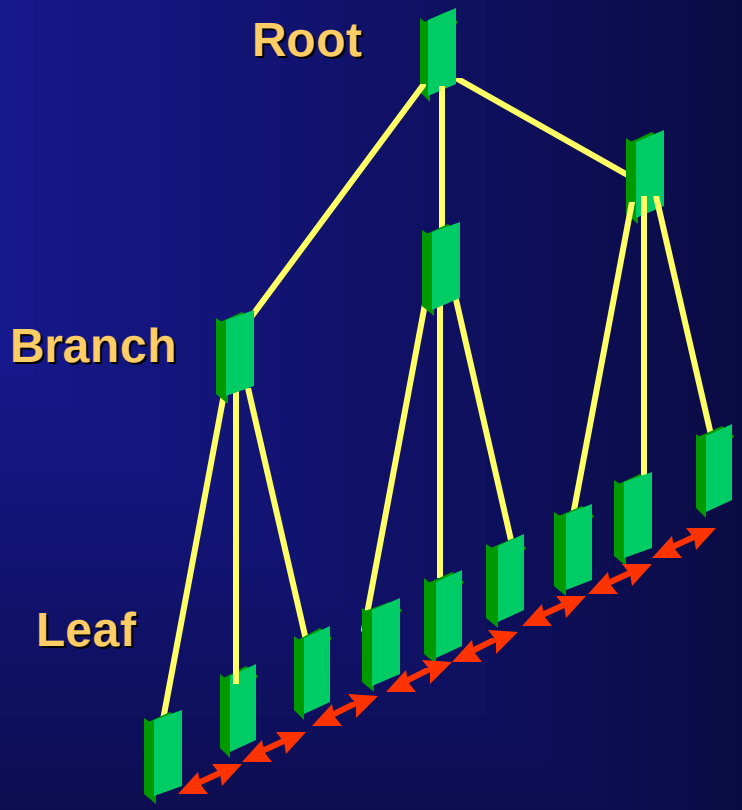
Objectives

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

- List the different types of indexes and their uses
- Choose an appropriate index type
- Create B-tree and bitmap indexes
- Drop indexes
- Get index information from the data dictionary

Overview

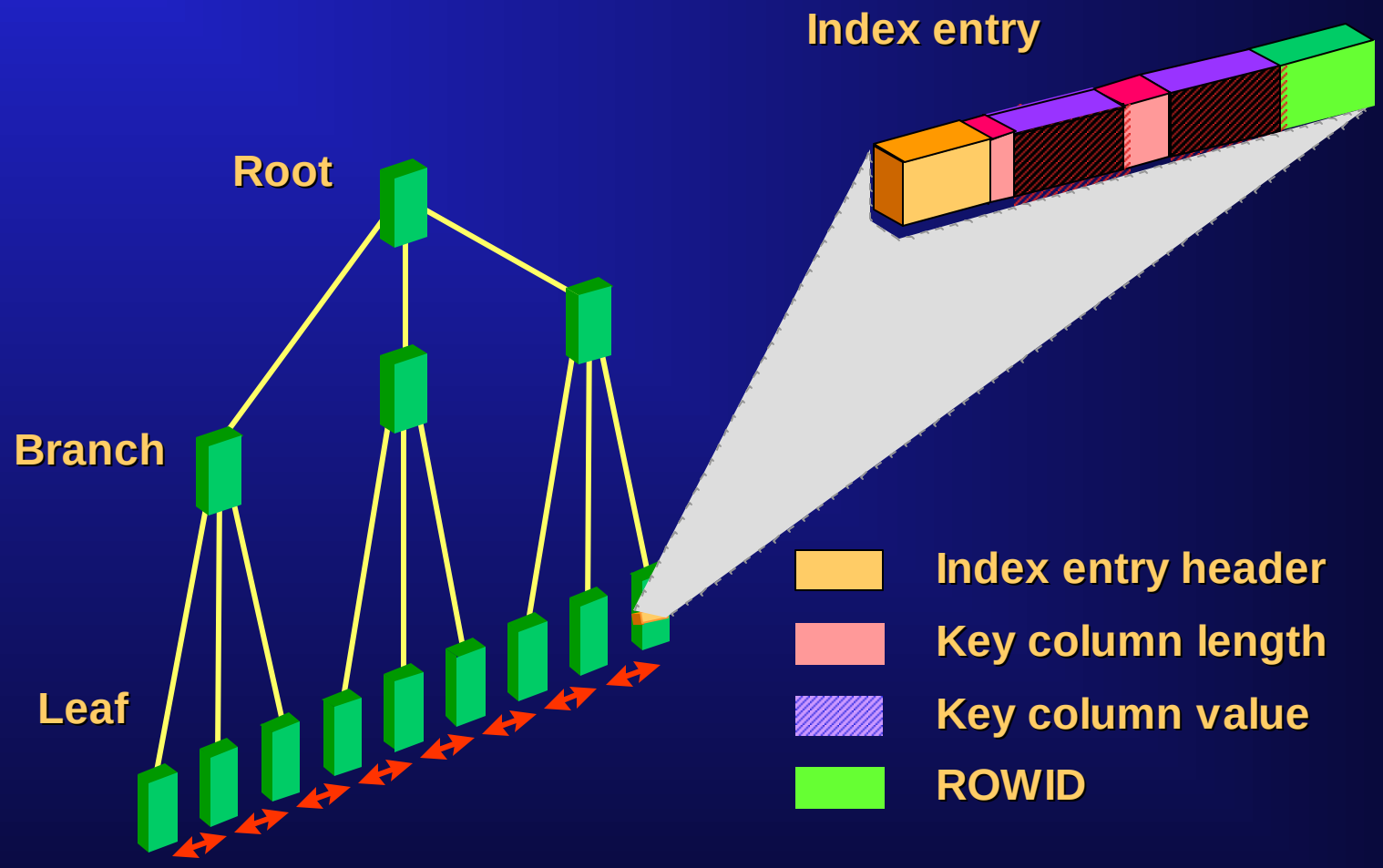
- Important for OLTP
- Overhead to maintain
- Indexing foreign key
- Bitmap for DSS



Classification of Indexes

- Logical classification
  - Single-column and composite indexes
  - Unique and nonunique indexes
  - Function-based indexes
  - Domain indexes
- Physical classification
  - Partitioned and nonpartitioned indexes
  - B-tree indexes
  - Normal and reverse key indexes
  - Bitmap indexes

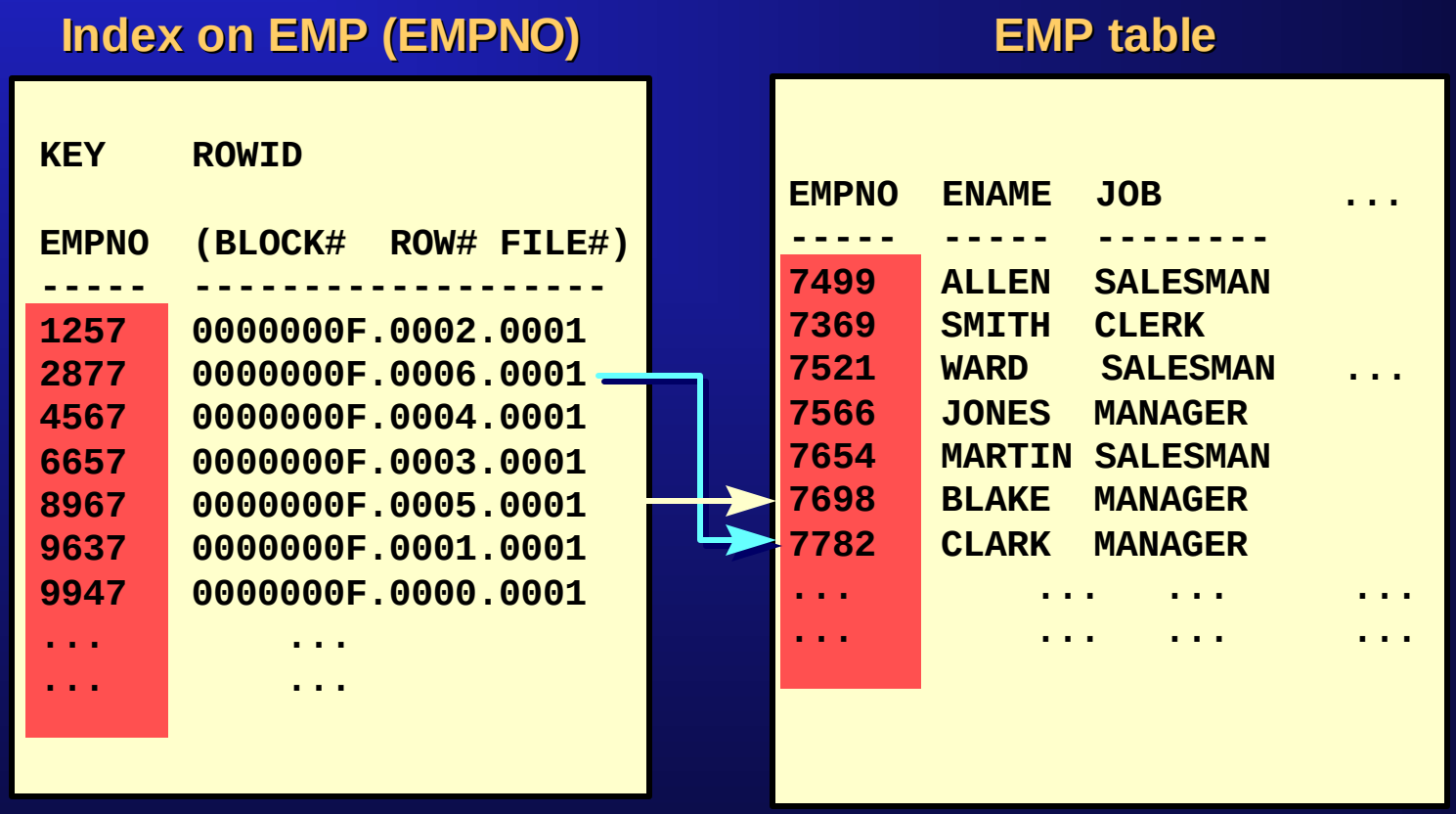
B-Tree Indexes



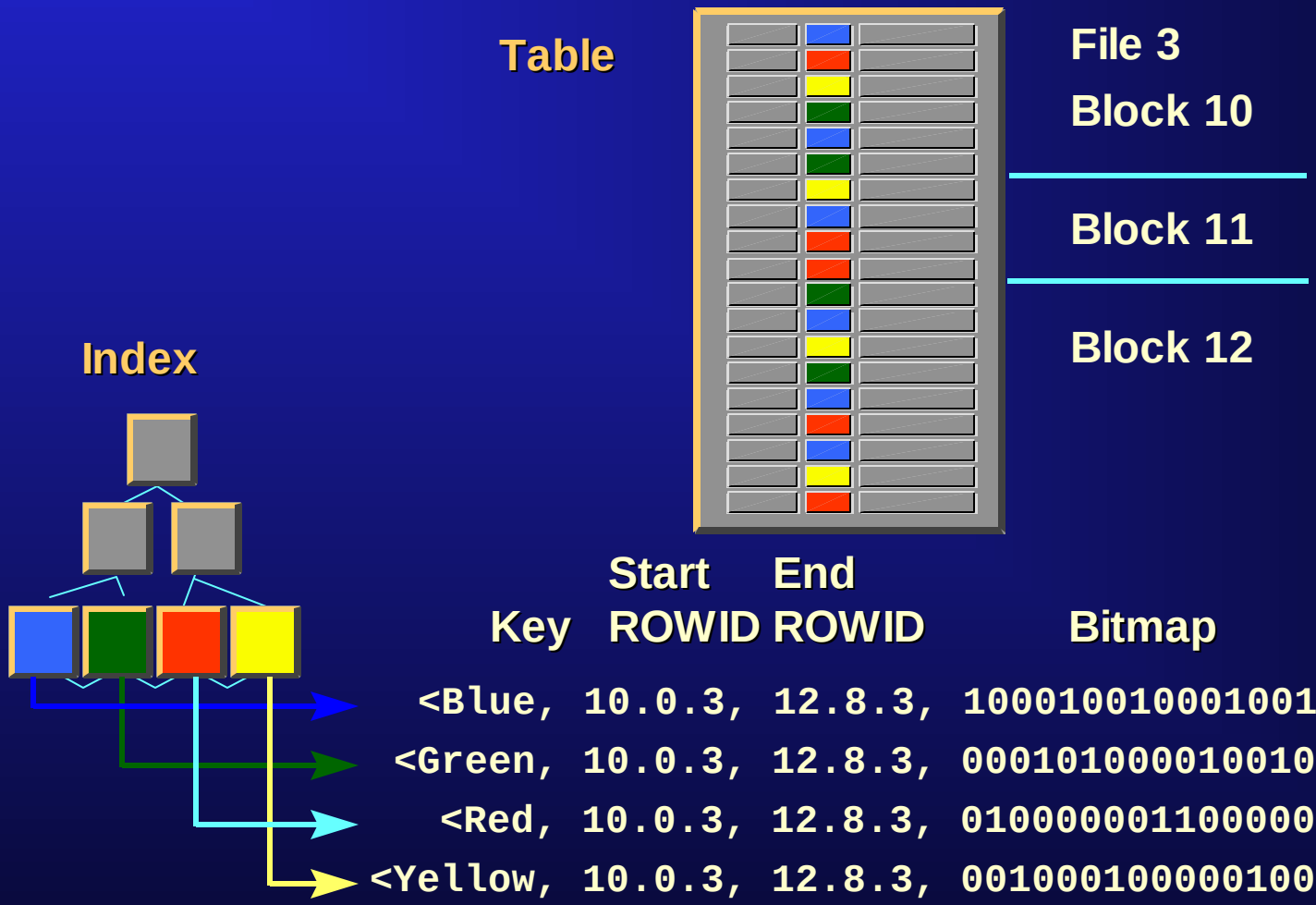
Function-Based Indexes

- Indexes are based on functions or expressions.
- Expressions can be built from columns of table, constants, SQL functions, and user-defined functions.
- Queries using expressions can use the function-based index.

Reverse Key Index



Bitmap Index



Comparing B-Tree and Bitmap Indexes

B-TREE	BITMAP
Suitable for high-cardinality columns	Suitable for low-cardinality columns
Updates on keys relatively inexpensive	Updates to key columns very expensive
Inefficient for queries using OR predicates	Efficient for queries using OR predicates
Useful for OLTP	Useful for DSS

Creating Normal B-Tree Indexes

```
SQL> CREATE INDEX emps_name_idx
  2    ON emps (last_name, first_name)
  3    PCTFREE    30
  4    STORAGE (  INITIAL        2M
  5               NEXT           600K
  6               PCTINCREASE    0
  7               MAXEXTENTS     50 )
  8    TABLESPACE case_large_index;
```

Creating Function-Based Indexes

- Dramatically improves query performance

```
CREATE INDEX uppercase_fbidx
ON     emp(UPPER(ename));
```

- Function-based indexes must be enabled using:

```
ALTER SESSION SET query_rewrite_enabled = TRUE;
```

- Queries using expressions can use the index

```
SELECT empno, ename, deptno, job, sal
FROM   emp
WHERE  UPPER(ename) = 'SMITH';
```

Creating Reverse Key Indexes

```
SQL> CREATE INDEX emps_hire_date_idx
  2    ON emps (hire_date)
  3    REVERSE
  4    PCTFREE 30
  5    STORAGE ( INITIAL     100K
  6              NEXT        100K
  7              PCTINCREASE 0
  8              MAXEXTENTS  50 )
  9    TABLESPACE case_large_index;
```

Creating Bitmap Indexes

```
SQL> CREATE BITMAP INDEX bonus_comm_idx
     2   ON bonus(comm)
     3   PCTFREE 30
     4   STORAGE (   INITIAL        40K
     5               NEXT           40K
     6               PCTINCREASE    0
     7               MAXEXTENTS     50)
     8   TABLESPACE case_large_index;
```

Creating Indexes: Guidelines

- Balance query and DML needs
- Place in separate tablespace
- Use uniform extent sizes: multiples of five blocks or MINIMUM EXTENT size
- Consider NOLOGGING for large indexes
- Set PCTFREE = 0 for increasing key values

  If new key values within the current range
  $$\text{PCTFREE} = 100 * \frac{(\text{Max\_entries} - \text{Current\_entries})}{\text{Max\_entries}}$$
- Calculate size similar to a table
- Make bitmap indexes generally smaller than B-tree indexes

Changing Storage Parameters for Indexes

```
SQL> ALTER INDEX emps_name_idx
  2    STORAGE ( NEXT         400K
  3              MAXEXTENTS   100 )
  4    INITRANS 4;
```

Allocating and Deallocating Index Space

```
SQL> ALTER INDEX emps_name_idx
     2   ALLOCATE EXTENT (
     3      SIZE 200K
     4      DATAFILE
     5         '/DISK6/indx01.dbf' );
```

```
SQL> ALTER INDEX emps_name_idx
     2   DEALLOCATE UNUSED;
```

Rebuilding Indexes

Use the ALTER INDEX command to:

- Move an index to a different tablespace
- Improve space utilization by removing deleted entries
- Change a reverse key index to a normal B-tree index and vice versa

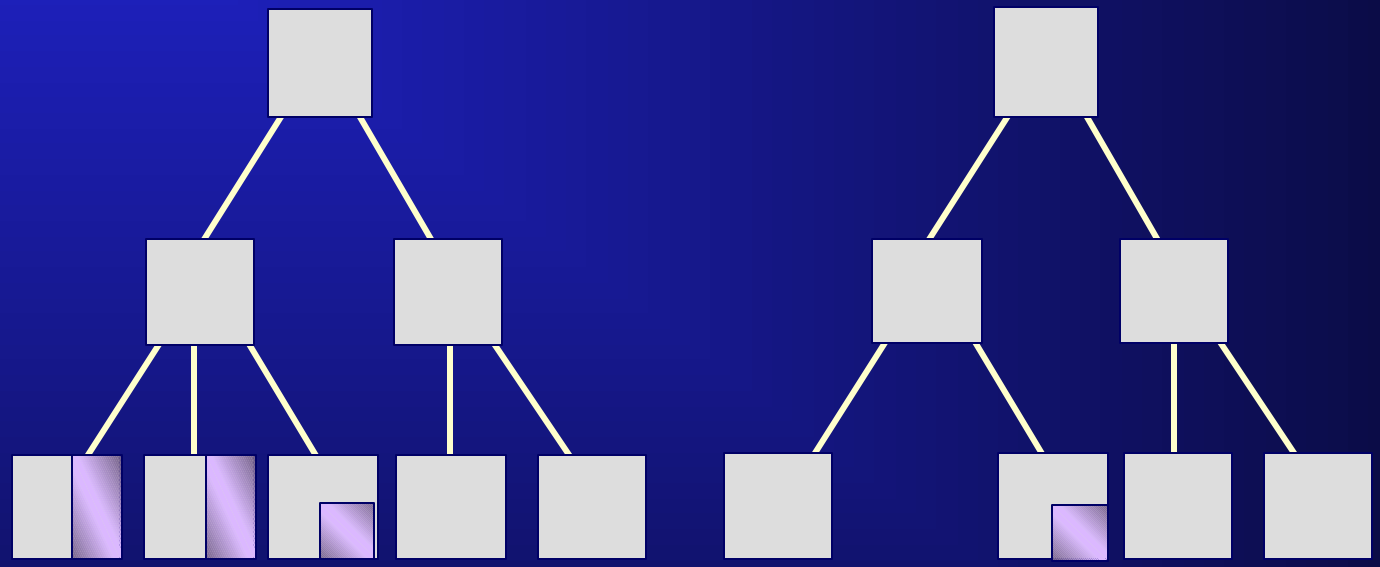
```
ALTER INDEX summit.orders_region_id_idx REBUILD
TABLESPACE indx02;
```

Rebuilding Indexes Online

Rebuilding indexes can be done with minimal table locking.

```
ALTER INDEX summit.orders_id_idx REBUILD ONLINE;
```

Coalescing Indexes



Before coalescing

After coalescing

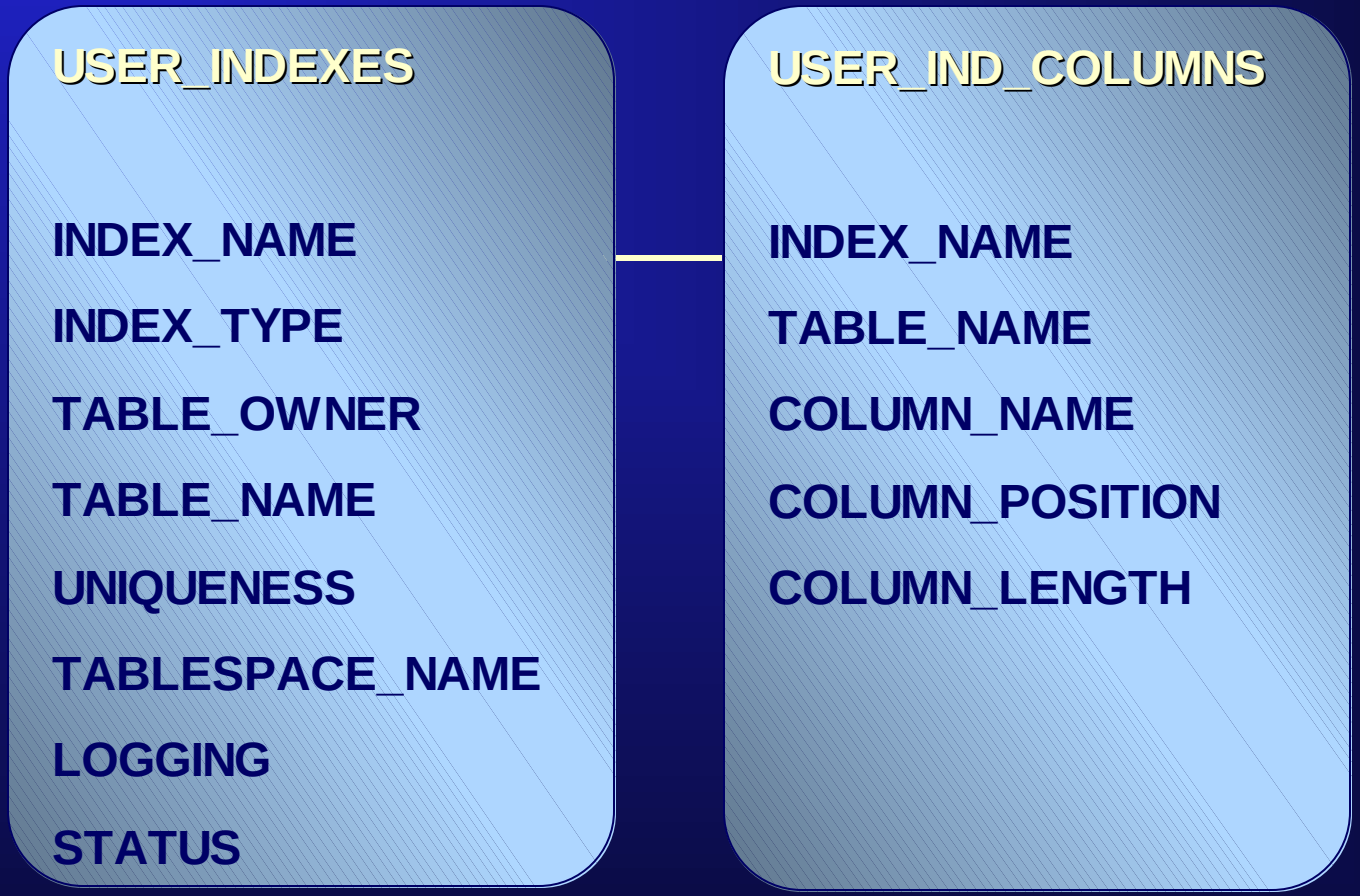
```
ALTER INDEX summit.orders_id_idx COALESCE;
```

Dropping Indexes

- Drop an index before a bulk load and re-create after.
- Drop indexes that are needed infrequently and build them when necessary.
- Drop and recreate invalid indexes.

```
DROP INDEX emp_name_idx;
```

Obtaining Index Information



Summary

In this lesson, you should have learned how to:

- Create different types of indexes
- Determine what type of index to use
- Reorganize indexes
- Select index information from the data dictionary

Practice Overview

This practice covers the following topics:

- Identifying the columns that need to be indexed in a table
- Creating indexes on a table
- Setting the storage parameters for indexes

8

Implementing Index-Organized Tables

ORACLE

Objectives

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

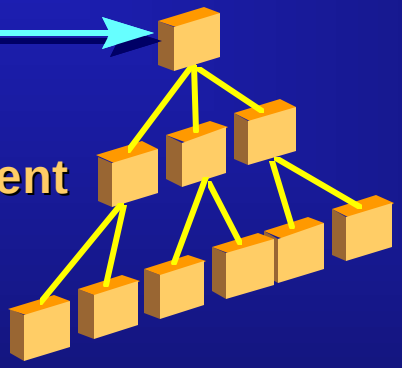
- Create and maintain index-organized tables
- Retrieve information about IOTs from the data dictionary

Index-Organized Tables

Indexed access on table

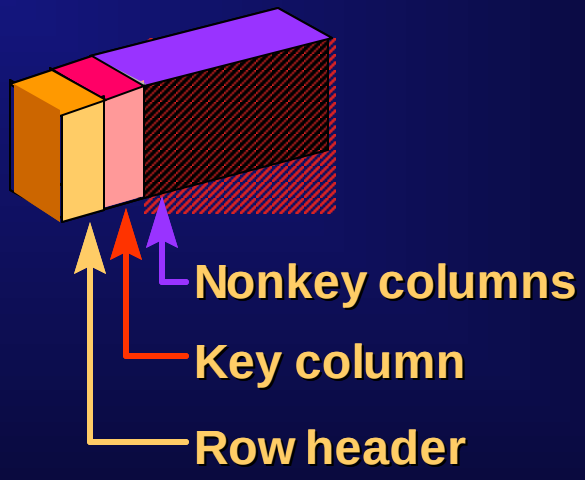
Indexed column value

Index segment

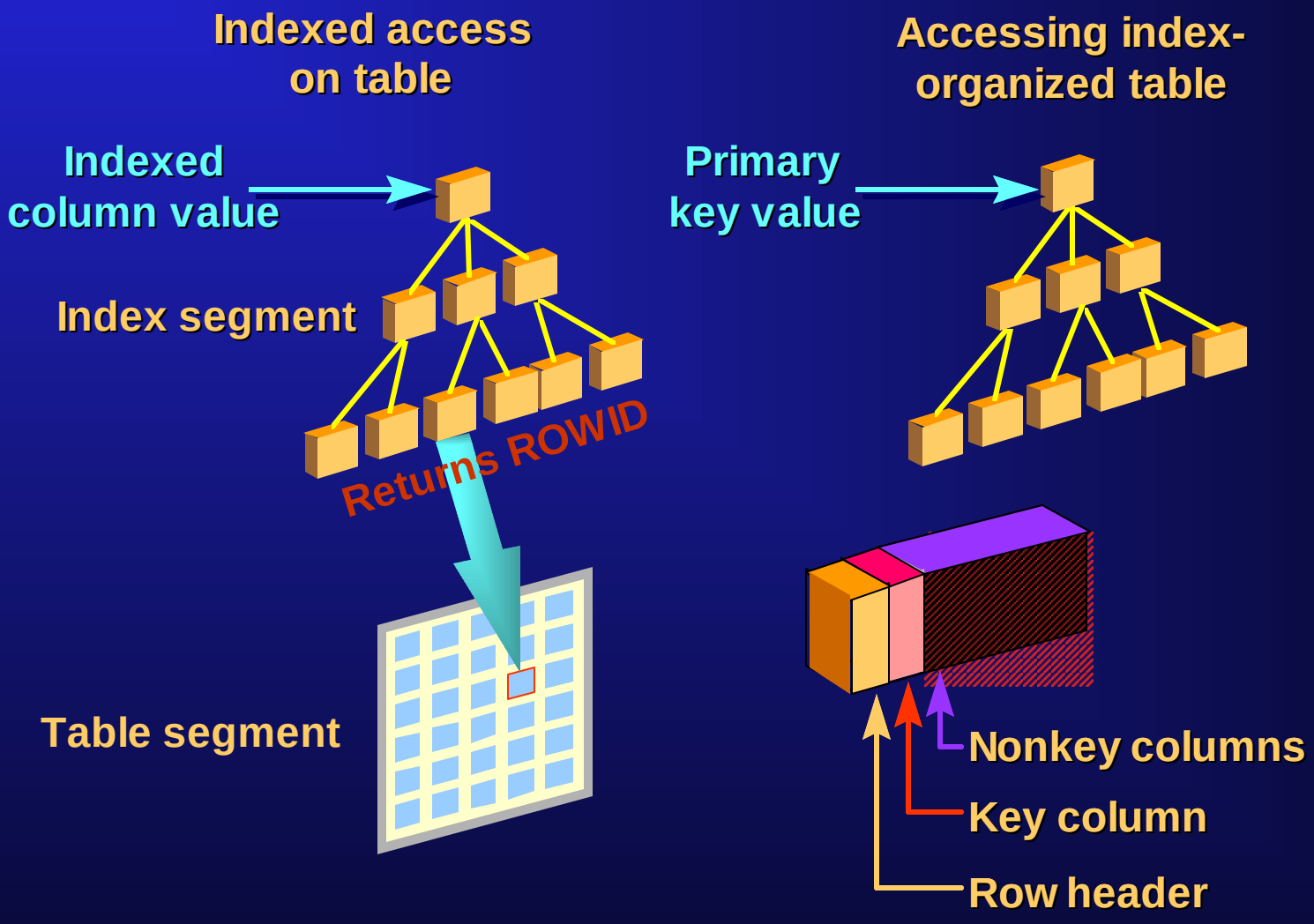


Accessing index-organized table

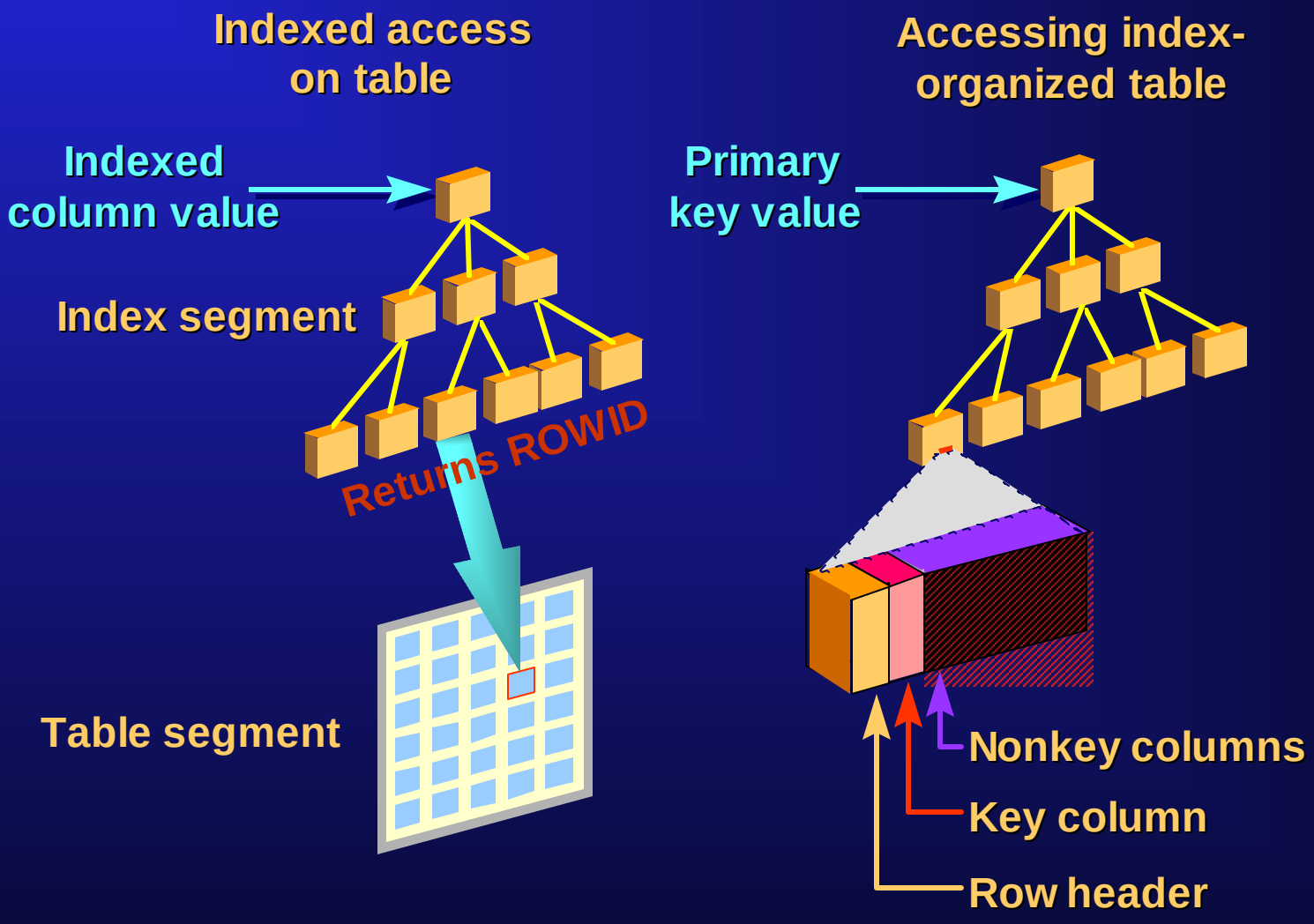
Primary key value



Index-Organized Tables



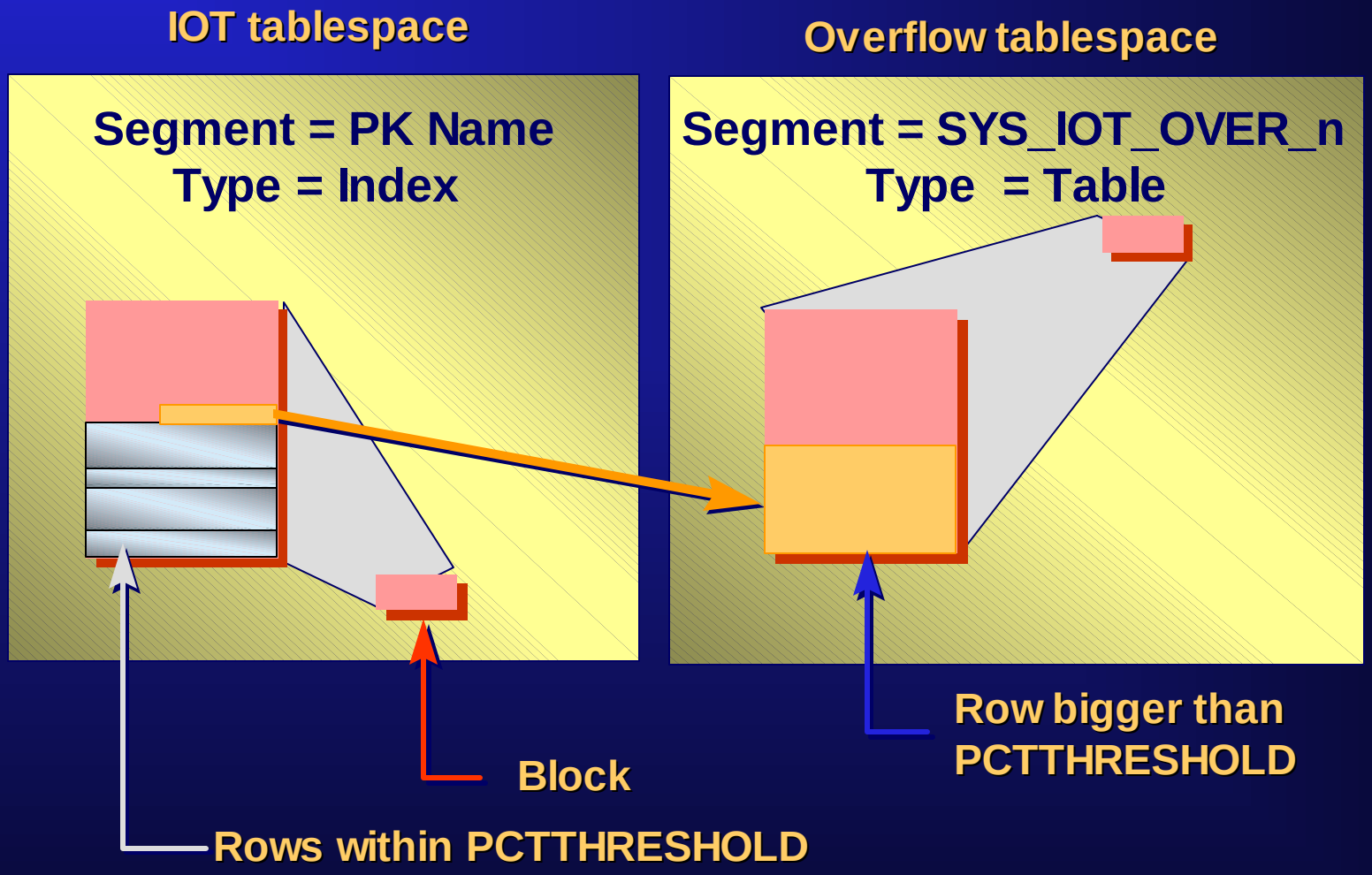
Index-Organized Tables



Index-Organized Tables

ORDINARY TABLE	INDEX-ORGANIZED TABLE
Primary key is optional	Primary key must be specified
ROWID-based access	Primary key-based access
Physical rowid in ROWID	Logical rowid in ROWID
Full table scan returns unordered rows	Full-index scan returns all rows in primary key order
Duplicate storage of primary key	Single storage of primary key
Can contain LONG columns	Cannot contain LONG columns

Row Overflow



Creating Index-Organized Tables

```
SQL> CREATE TABLE codes . . .
  6    CONSTRAINT code_pk PRIMARY KEY...
  7    ORGANIZATION INDEX
  8      INCLUDING             short_desc
  9      PCTTHRESHOLD          0
 10    TABLESPACE case_large_index
 11    STORAGE . . .
 17    OVERFLOW
 18      STORAGE . . .
 24      TABLESPACE case_large_data;
```

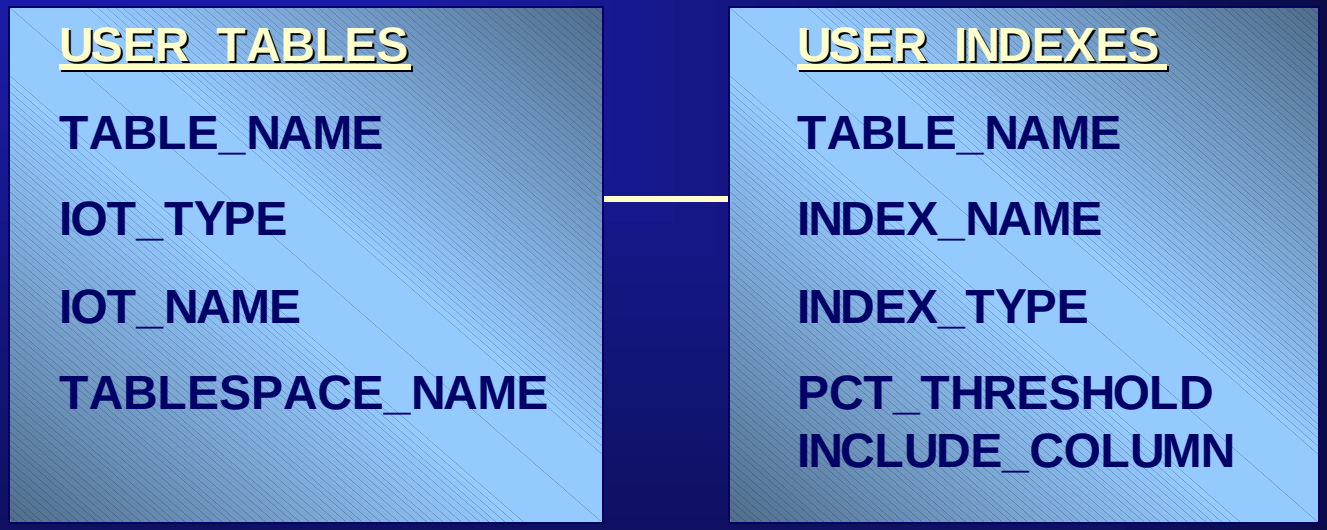
Secondary Indexes on IOTs

- Provide efficient access to IOT using nonkey columns
- Use logical rowids for their construction

IOT: Guidelines

- Primary key must be specified
- Overflow segment must be specified with PCTTHRESHOLD
- Lookup or reference tables are good IOT candidates
- Index segment:
  - Contains frequently used columns
  - Is sized like an index
- If overflow, then row segment:
  - Contains infrequently used columns
  - Is sized like a table

Retrieving IOT Information from the Data Dictionary



Summary

In this lesson, you should have learned:

- When to use index-organized tables
- How to create an index-organized table
- How to retrieve IOT information from the data dictionary

Practice Overview

This practice covers the following topics:

- Computing the STORAGE clause parameters for an IOT
- Creating an index-organized table
- Retrieving index information for an IOT from the data dictionary

9

Implementing Clusters

ORACLE

Objectives

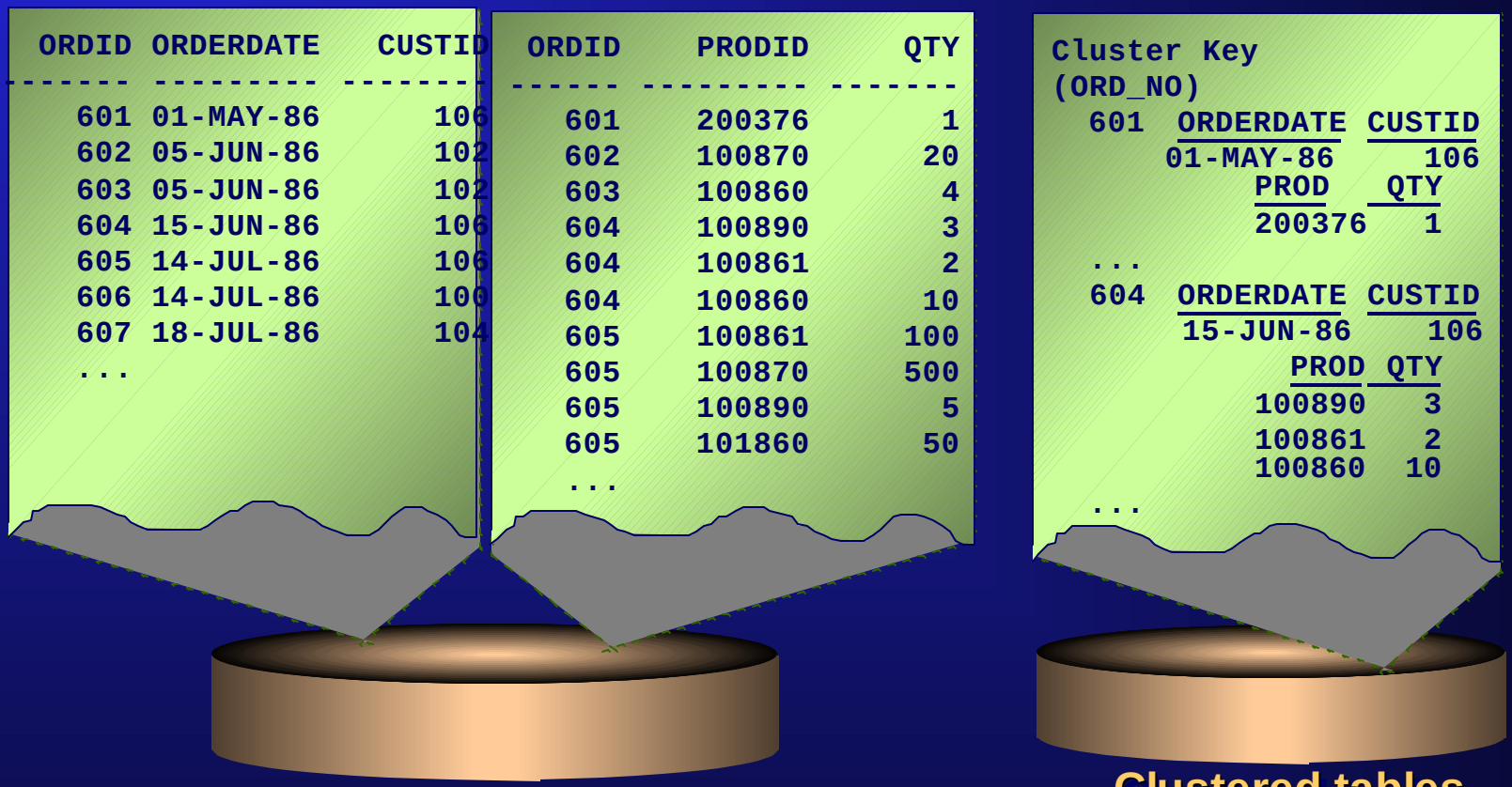
After completing this lesson, you should be able to do the following:

- Identify the characteristics of clusters
- Size a cluster record
- Create an index cluster
- Create a hash cluster
- Select cluster information from the data dictionary

Clusters

- One or more tables may form a cluster
- Rows from tables grouped on common column (cluster key) values
- Each cluster key value stored only once
- Choose good cluster key
- Extra overhead for DML
- No direct loads
- Full tables scans expensive

Clusters



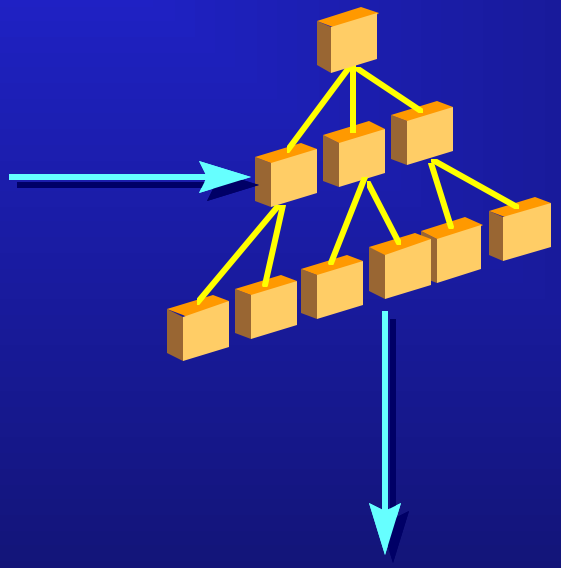
Unclustered tables stored separately

Clustered tables stored together

Index and Hash Clusters

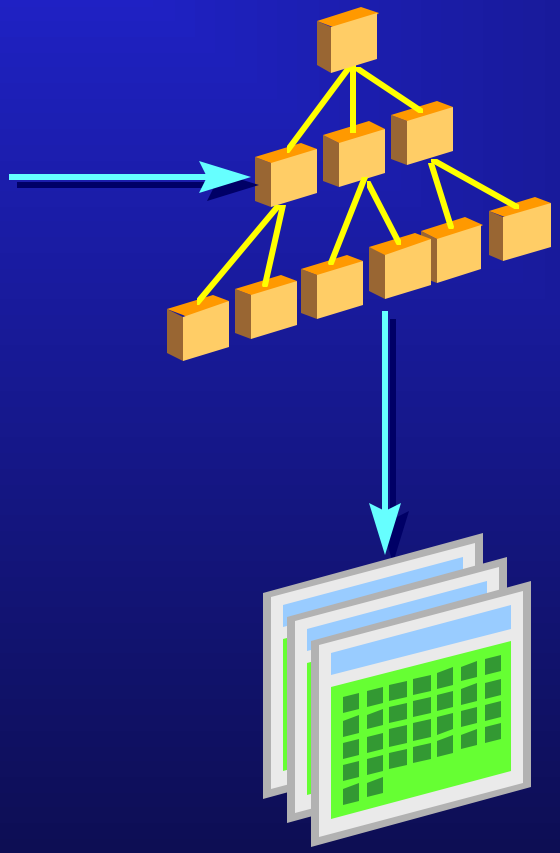
- Index cluster
  - Requires cluster index to access rows
  - Clusters rows
- Hash cluster
  - Uses hashing function
  - Can bypass index I/O
  - Optimizes selects with = operator
  - Requires even row distribution

Index Cluster



- Cluster index:
  - Clusters rows
  - Required to access rows
  - Includes null entries
  - Cannot index LONG
- Cluster key stored once
- Use index cluster when hash cluster not suitable

Index Cluster



- Cluster index:
  - Clusters rows
  - Required to access rows
  - Includes null entries
  - Cannot index LONG
- Cluster key stored once
- Use index cluster when hash cluster not suitable

Creating an Index Cluster

1. Create a cluster.

```
SQL> CREATE CLUSTER ord_clu
  2      ( ord_no NUMBER(3) )
  3    SIZE         161
  4    PCTFREE      10
  5    PCTUSED      48
  6    TABLESPACE   case_large_data
  7    STORAGE (    INITIAL      5M
  8                 NEXT         1M
  9                 PCTINCREASE  0 )
 10   INDEX;
```

Creating an Index Cluster

2. Create a cluster index.

```
SQL> CREATE INDEX ord_clu_idx
  2    ON CLUSTER ord_clu
  3    TABLESPACE case_large_index
  4    STORAGE (
  5      INITIAL     400K
  6      NEXT        100K
  7      PCTINCREASE 0 );
```

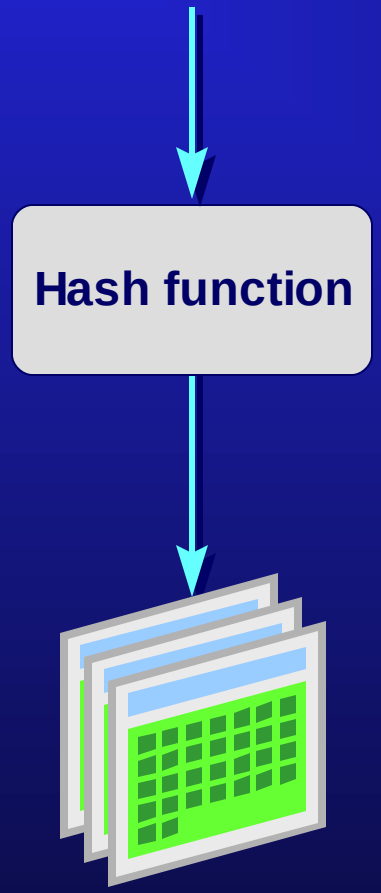
Creating an Index Cluster

3. Create tables in the cluster.

```
SQL> CREATE TABLE ord
  2    ( ord_no  NUMBER(3)
. . .
  6    CLUSTER ord_clu(ord_no);
```

```
SQL> CREATE TABLE item
  2    ( ord_no NUMBER(3)
. . .
  9    CLUSTER ord_clu(ord_no);
```

Hash Cluster



- Uses hashing function
- Can bypass index I/O
- Range or full table scans may not perform well
- Use when:
  - Optimizing SELECTs
  - Querying with the = operator
  - Table is large
  - Even row distribution

Creating a Hash Cluster

1. Create a cluster.

```
SQL> CREATE CLUSTER off_clu
  2    ( country   VARCHAR2(2),
  3      postcode  VARCHAR2(10) )
  4    SIZE        350
  5    PCTFREE     0
  6    TABLESPACE  case_large_data
  7    STORAGE (INITIAL     3M
  8             NEXT        1M
  9             PCTINCREASE 0 )
 10    HASHKEYS    6000;
```

Creating a Hash Cluster

2. Create tables in the cluster.

```
SQL> CREATE TABLE office
  2    ( office_cd   NUMBER(3),
  3      cost_ctr    NUMBER(3),
  4      country     VARCHAR2(2),
  5      postcode    VARCHAR2(10) )
  6    CLUSTER off_clu(country, postcode);
```

Parameters Specific to Hash Clusters

- HASHKEYS: Number of key values
- HASH IS: Optional user-defined hash function
- SIZE: A larger value may improve SELECT performance by reducing collisions.

Altering Clusters

- Change parameters for storage and block space usage
- Change SIZE for index clusters
- Allocate and deallocate space

```
ALTER CLUSTER ord_clu
  SIZE 300
  STORAGE (NEXT 2M);
```

- Cannot alter SIZE, HASH IS, or HASHKEYS for hash clusters

Dropping Clusters

- Use INCLUDING TABLES clause to drop the tables with the cluster.

```
DROP CLUSTER ord_clu
  INCLUDING TABLES;
```

or

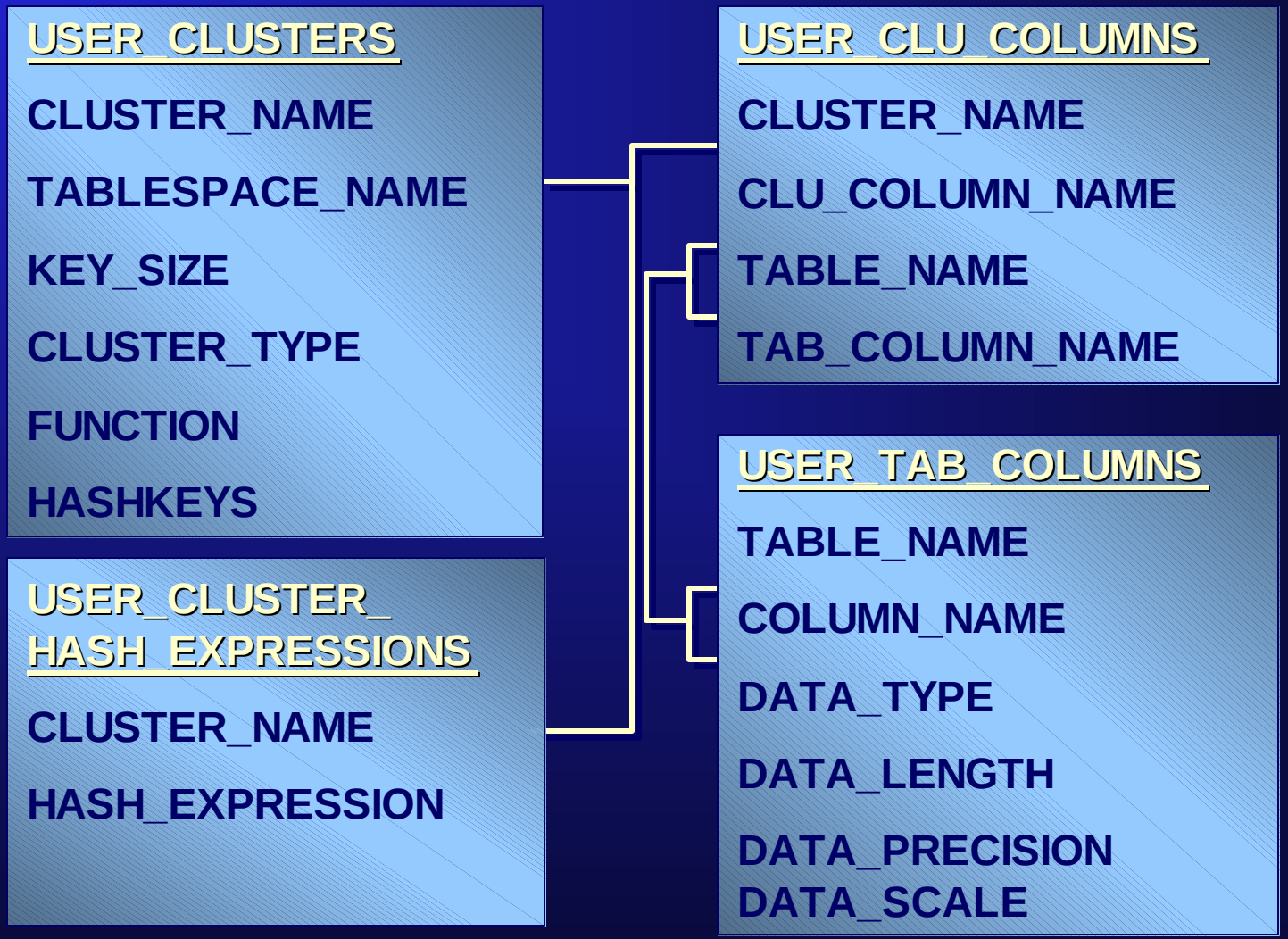
- Drop the tables before dropping the cluster.

```
DROP TABLE ord;
DROP TABLE item;
DROP CLUSTER ord_clu;
```

Using Clusters: Guidelines

- Use clusters
  - For tables that are primarily queried or joined on the cluster key
  - If rows for a cluster key can fit into one or two blocks
- Tables can be converted into clustered tables without rewriting the application.
- Clusters can reduce the performance of:
  - DML statements
  - Full table scans
- Clusters are not suitable for tables that are frequently accessed individually.

Retrieving Cluster Information



Summary

In this lesson, you should have learned how to implement:

- Index clusters
- Hash clusters

Practice Overview

This practice covers the following topics:

- Creating index clusters
- Creating hash clusters
- Displaying the storage information, size, and hashing characteristics for the clusters
- Displaying the table and cluster names from the data dictionary

10

Enforcing Business Rules with Constraints

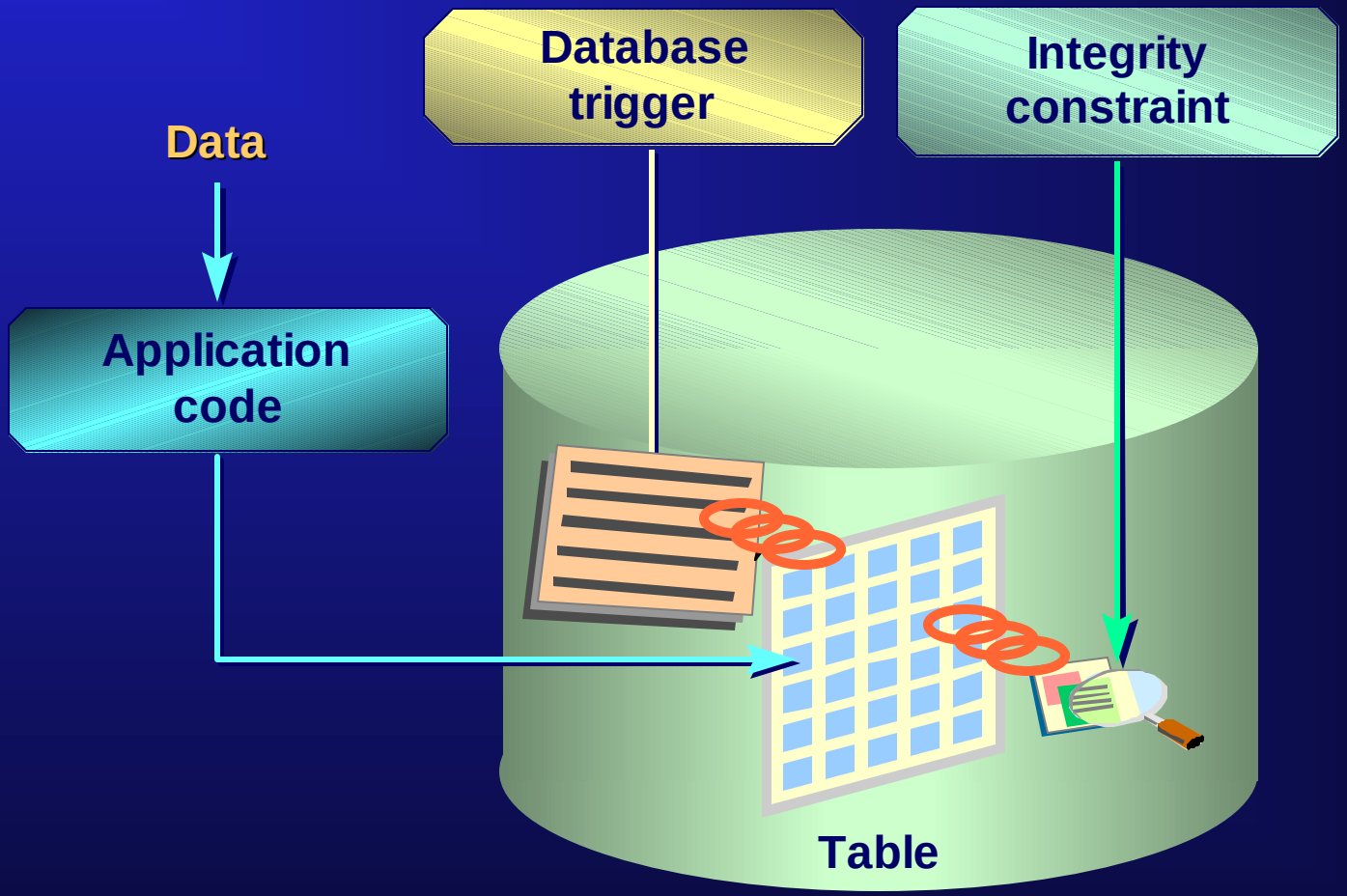
ORACLE

Objectives

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

- Differentiate between constraint types
- Disable constraints
- Enable constraints with or without validation
- Defer the enforcement of constraints
- Describe how the Oracle server enforces primary key and unique constraints
- Decide when to use a foreign key index
- Handle constraint exceptions

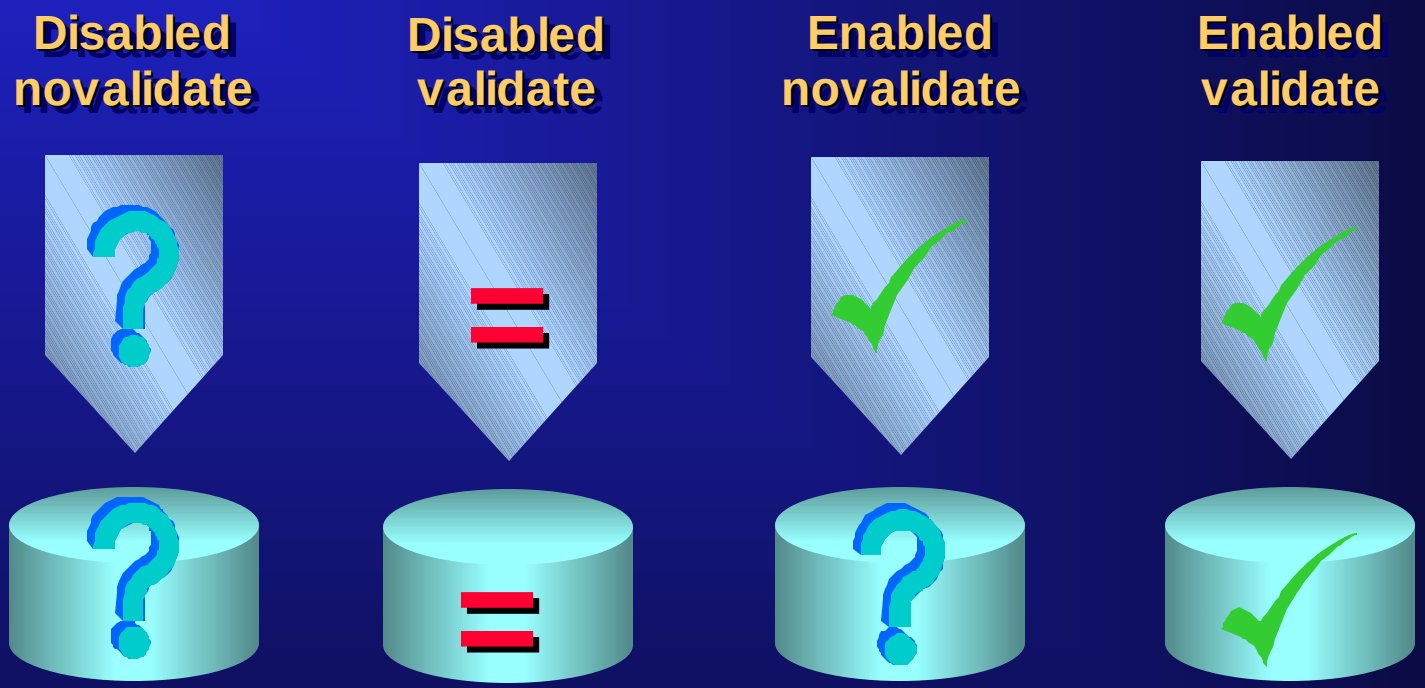
Data Integrity



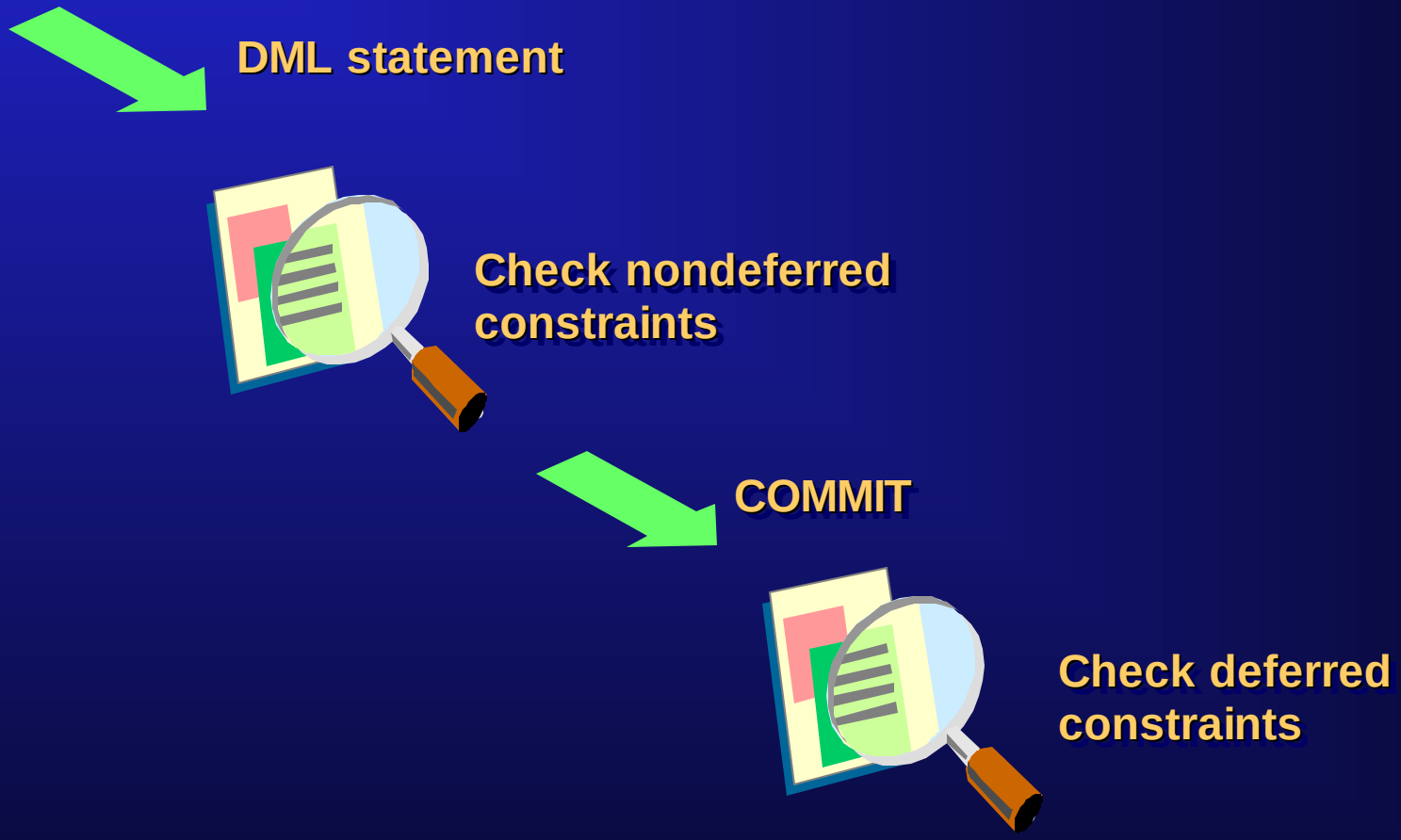
Types of Constraints

CONSTRAINT	DESCRIPTION
NOT NULL	Specifies that a column cannot contain null values
UNIQUE	Designates a column or combination of columns as unique
PRIMARY KEY	Designates a column or combination of columns as the table' s primary key
FOREIGN KEY	Designates a column or combination of columns as the foreign key in a referential integrity constraint
CHECK	Specifies a condition that each row of the table must satisfy

Constraint States



Deferred Constraints

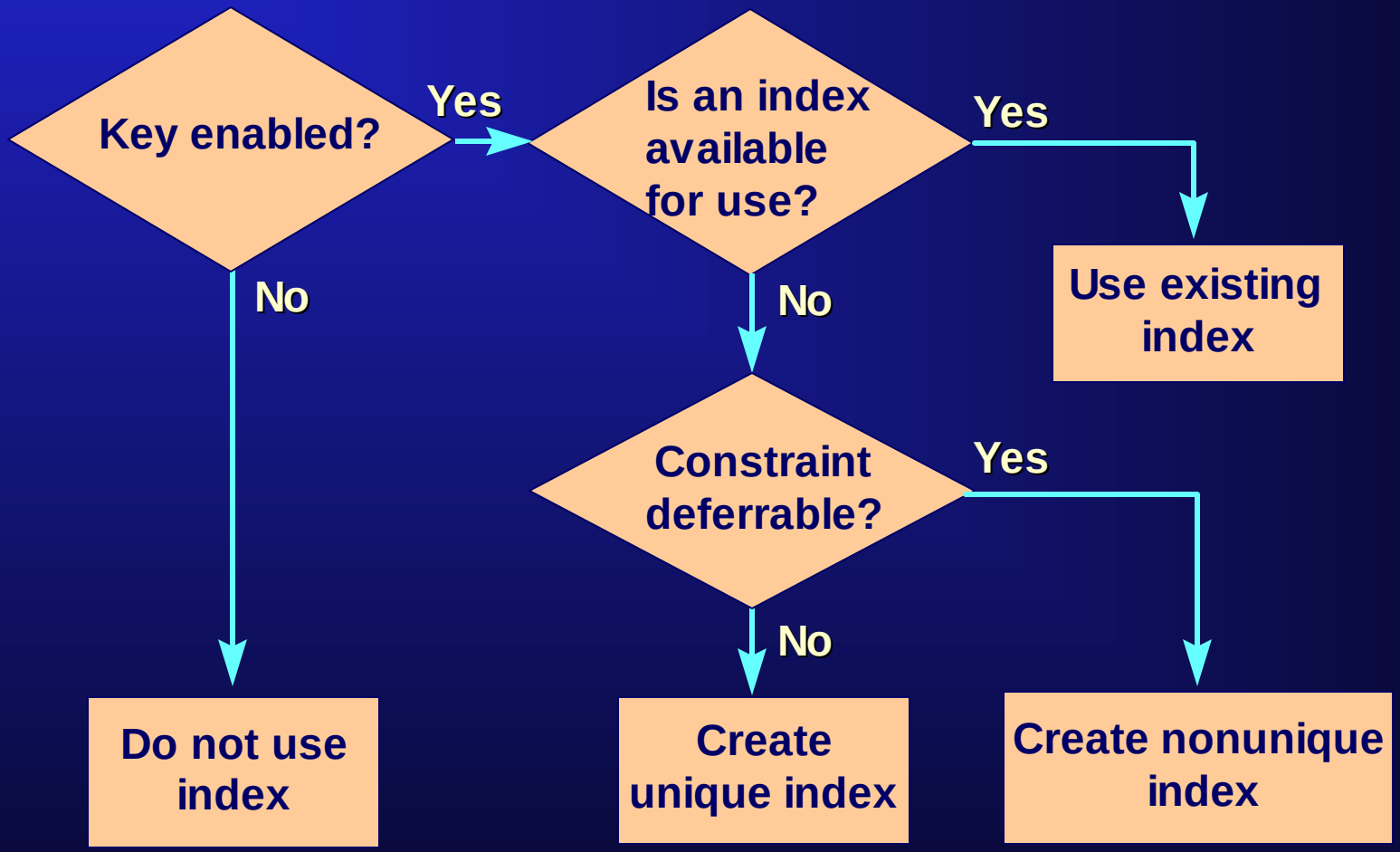


Defining Constraints as Immediate or Deferred

Use the ALTER SESSION or SET CONSTRAINTS command to set the mode for your constraint.

```
ALTER SESSION
SET CONSTRAINT[S] =
{IMMEDIATE|DEFERRED|DEFAULT};
```

Primary/Unique Key Enforcement



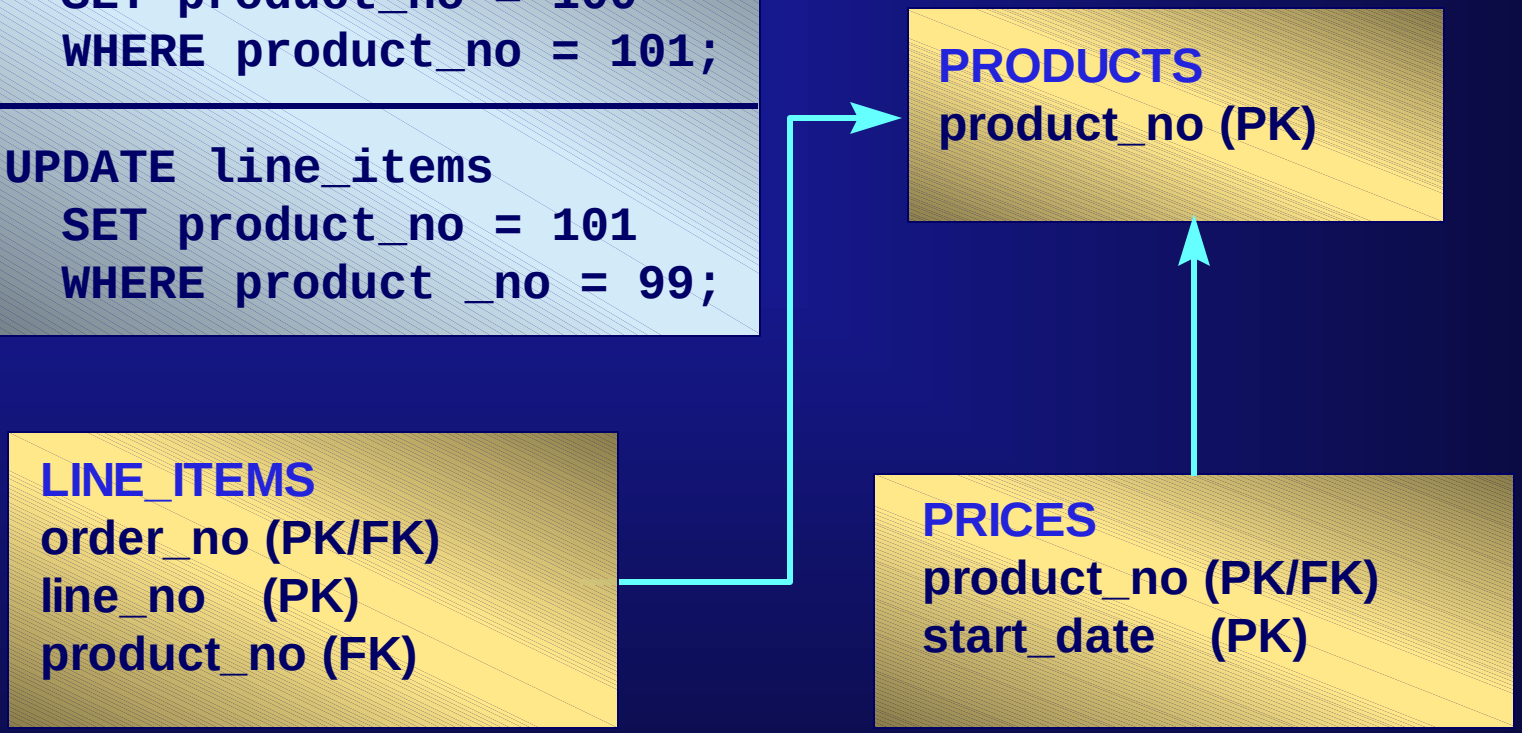
Foreign Key Considerations

DESIRED ACTION	APPROPRIATE SOLUTION
Drop parent table	CASCADE CONSTRAINTS
Truncate parent table	Disable/drop foreign key
Drop tablespace containing parent table	Use CASCADE CONSTRAINTS clause
Avoid locks on child table while performing DML on parent table	Create index on foreign key
Perform DML on child table	Ensure tablespace containing parent key index online

Foreign Key Table Locks

```
UPDATE products
  SET product_no = 100
  WHERE product_no = 101;
```

```
UPDATE line_items
  SET product_no = 101
  WHERE product _no = 99;
```



Defining Constraints While Creating a Table

```
CREATE TABLE scott.employees(
  empno NUMBER(4)
        CONSTRAINT emp_pk PRIMARY KEY
        DEFERRABLE
        USING INDEX
          STORAGE(INITIAL 100K NEXT 100K)
          TABLESPACE indx01,
  last_name VARCHAR2(30)
        CONSTRAINT emp_ln_nn NOT NULL,
  deptno NUMBER(2))
  TABLESPACE data01;
```

Defining Constraints: Guidelines

- Primary and unique constraints:
  - Place indexes in a separate tablespace.
  - Use nonunique indexes if bulk loads are frequent.
- Self-referencing foreign keys:
  - Define or enable foreign keys after initial load.
  - Defer constraint checking.

Disabling Constraints

- Disable before bulk load, especially self-referencing foreign keys
- Disable referencing foreign keys before disabling parent keys

```
ALTER TABLE scott.departments
DISABLE CONSTRAINT dept_pk CASCADE;
```

- Disable when exporting or importing one table at a time
- Drop unique indexes, but retain nonunique indexes

Enabling Constraints

- No locks on table
- Primary/unique keys must use nonunique indexes if duplicates exist

```
ALTER TABLE scott.departments
ENABLE NOVALIDATE CONSTRAINT dept_pk;
```

Enabling Constraints

- Can use unique or nonunique indexes
- Needs valid table data

```
ALTER TABLE scott.employees
ENABLE VALIDATE CONSTRAINT emp_dept_fk;
```

Using the EXCEPTIONS Table

1. Create EXCEPTIONS table (`utlexcpt.sql`).
2. Execute ALTER TABLE with EXCEPTIONS clause.
3. Use subquery on EXCEPTIONS to locate rows with invalid data.
4. Rectify the errors.
5. Reexecute ALTER TABLE to enable the constraint.

Dropping Constraints

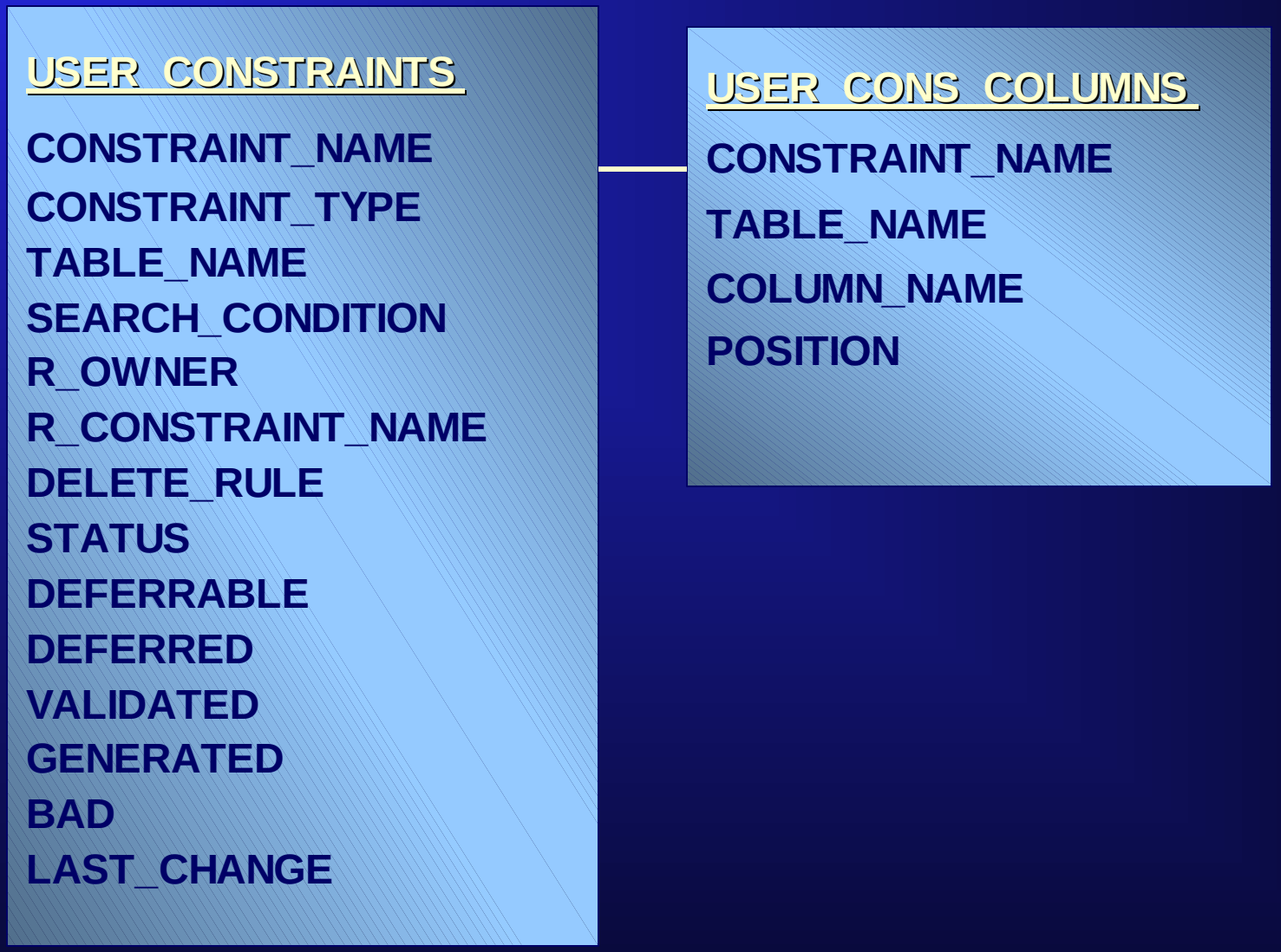
- Drop constraints using this command:

```
ALTER TABLE scott.employees
DROP CONSTRAINT emp_ln_uk;
```

- Drop a table and any referencing foreign key using this command:

```
DROP TABLE departments
CASCADE CONSTRAINTS;
```

Getting Constraint Information



Summary

In this lesson, you should have learned how to:

- Implement constraints
- Use an appropriate strategy for creating and maintaining constraints

Practice Overview

This practice covers the following topics:

- Enabling and disabling constraints
- Identifying constraint violations in a table
- Displaying constraint information by querying from the data dictionary

11

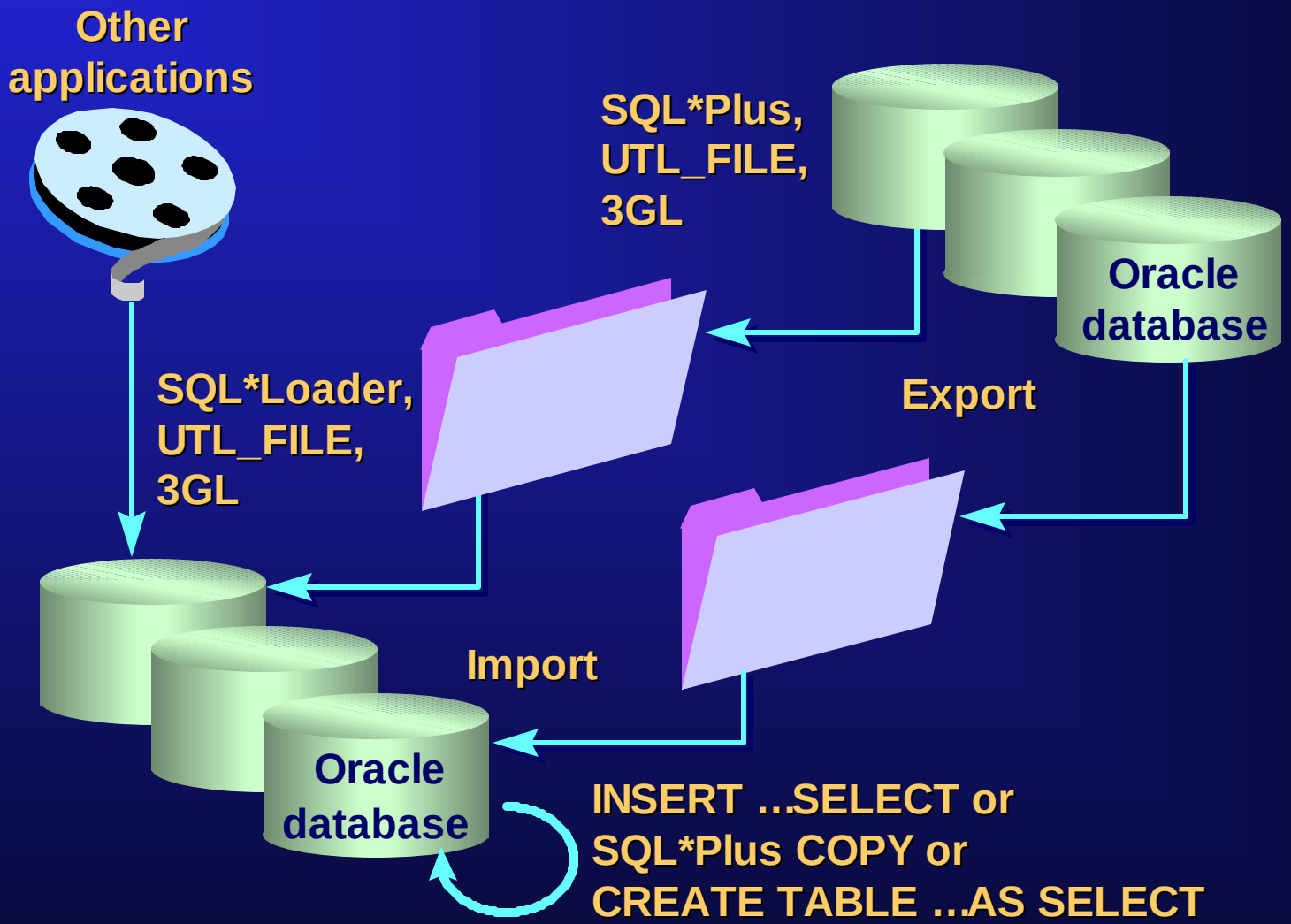
Transporting Data

Objectives

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

- Select the appropriate transfer technique
- Use SQL*Plus to extract data
- Use the UTL_FILE package
- Use SQL*Loader to load data
- Use direct mode in SQL*Loader
- Use direct path load API
- Use direct-load inserts

Overview



Selecting the Correct Method: DB to DB or Schema to Schema

- Export and import utilities
  - Schema backup/recovery
  - Separate definitions and data
  - Uses operating system file
- CREATE TABLE …AS SELECT
- ALTER TABLE …MOVE
- INSERT …SELECT
- SQL*Plus COPY

Selecting the Correct Method: Oracle to Non-Oracle Data Stores

- SQL*Loader
- SQL*Plus
- Oracle Heterogeneous Services
- Direct path load API
- UTL_FILE package
- Third-generation languages (3GLs)

Methods Used in This Course

- SQL*Plus
- UTL_FILE package
- SQL*Loader
- Direct path load API
- Export and import utilities

SQL*Plus COPY Command

- To copy data between databases and between tables on the same database:

```
SQL> COPY FROM <source_database> -
>       TO <destination_database> action -
>          <destination_table>(column_name,…)-
>    USING query
```

- Where:
  - *action*: REPLACE / CREATE / INSERT / APPEND
  - *query*: Specifies source table and data to be copied to destination table

Example: Extracting Data with SQL*Plus

```
SET FEEDBACK OFF ECHO OFF TERMOUT OFF PAGESIZE 0
ACCEPT p_stud_id NUMBER -
       PROMPT “Enter the student ID: “
SPOOL std_crs.1st

SELECT  class_id, stud_id, reg_date, status
        cancel_date, appr_by, attended
FROM    registrations
WHERE stud_id = &p_stud_id;

SPOOL OFF
```

PL/SQL File I/O

- UTL_FILE package
  - Similar to operating system I/O
  - Input and output capabilities
- DBA sets parameter to indicate valid directories
- utl_file.file_type to DECLARE files

Example: UTL_FILE

```
CREATE PROCEDURE msg_log
  ( p_msg IN VARCHAR2 )
IS
  v_log_file   utl_file.file_type;
BEGIN
  v_log_file := utl_file.fopen
    ('\data','msg.log','a');
  utl_file.put_line(v_log_file,p_msg);
  utl_file.fclose(v_log_file);
END;
```

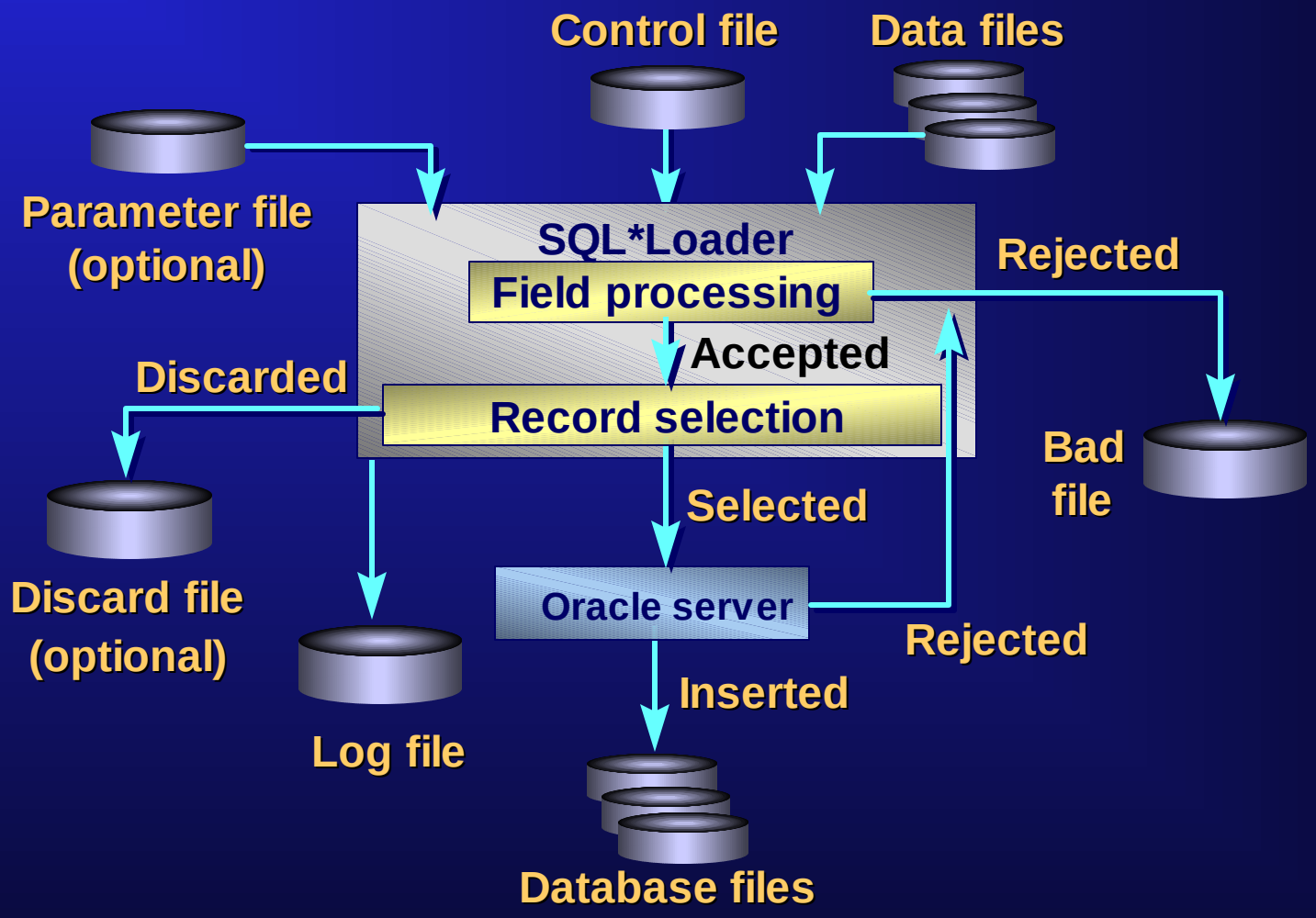
UTL_FILE Functions and Procedures

FOPEN	Opens a file for input or output
IS_OPEN	Tests if a file handle identifies an open file
FCLOSE	Closes an open file identified by a file handle
FCLOSE_ALL	Closes all open file handles for the session
GET_LINE	Reads a line of text
PUT	Writes the text string
NEW_LINE	Writes one or more line terminators
PUT_LINE	Writes the text string with end-of-line
PUTF	Formatted PUT procedure
FFLUSH	Physically writes all pending data

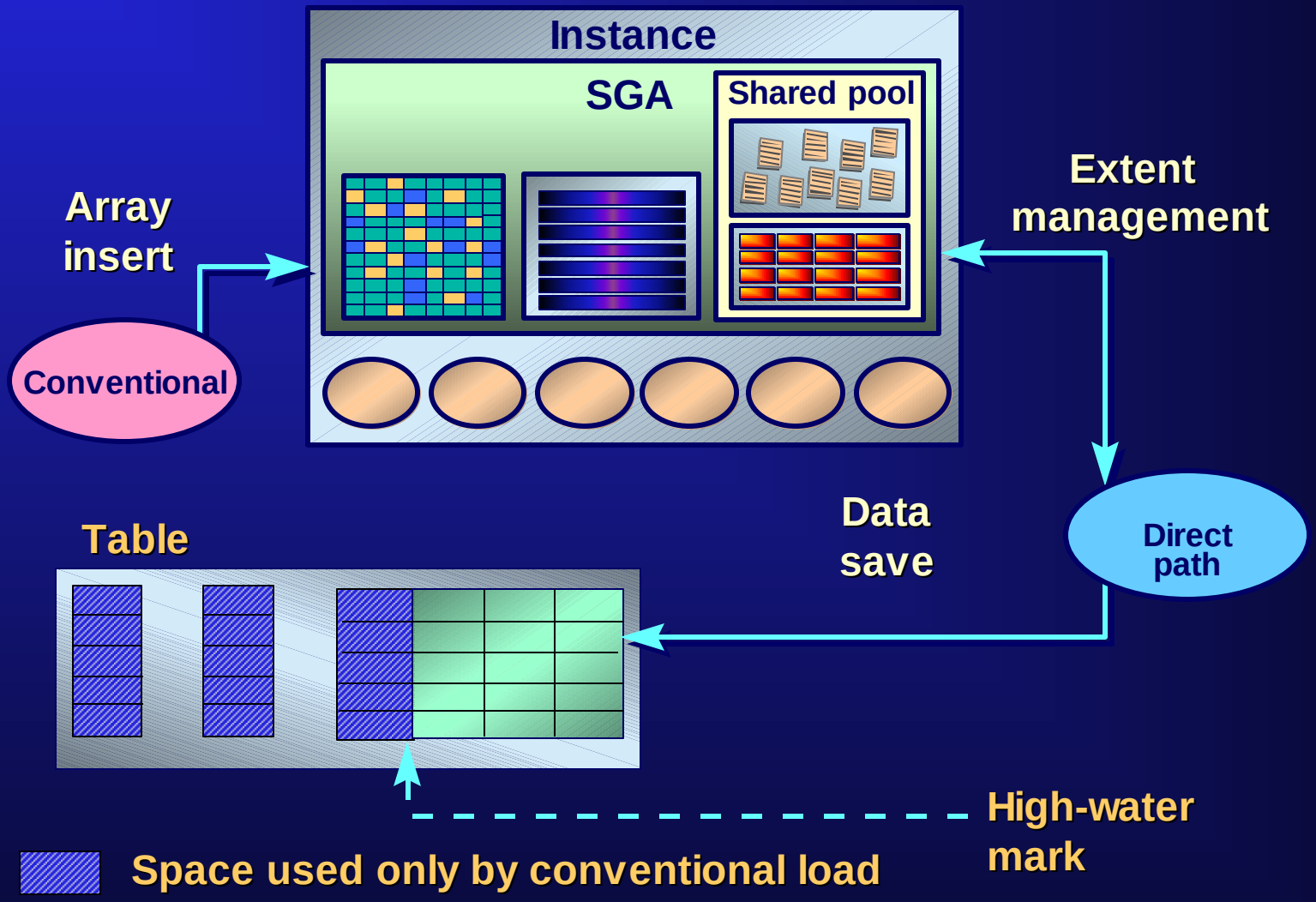
Exceptions

• INVALID_PATH	Invalid location or filename
• INVALID_MODE	Invalid open_mode in FOPEN
• INVALID_ FILEHANDLE	Invalid file handle
• INVALID_ OPERATION	Could not be opened or operated on
• READ_ERROR	OS error during a read
• WRITE_ERROR	OS error during a write
• INTERNAL_ERROR	Unspecified error in PL/SQL

SQL*Loader



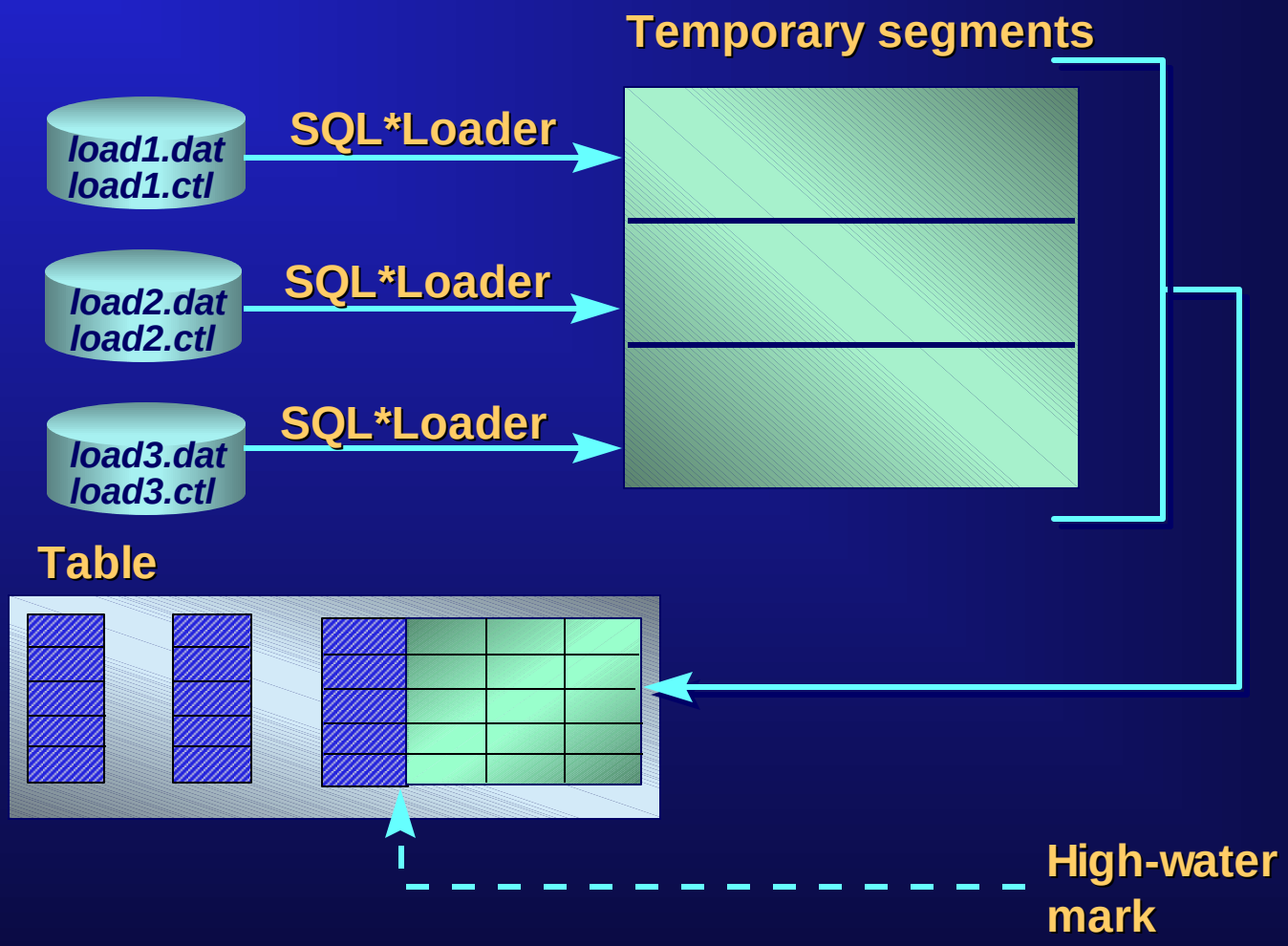
Conventional and Direct Loads



Comparing Direct and Conventional Path Loads

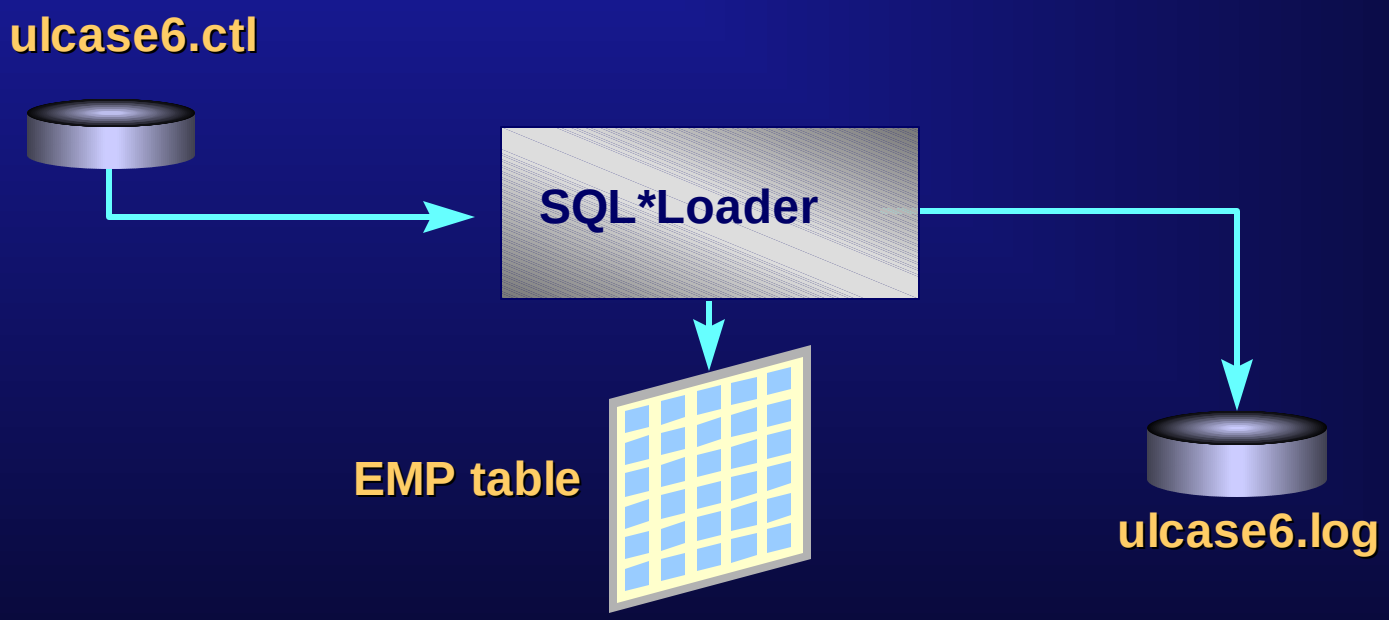
CONVENTIONAL LOAD	DIRECT PATH LOAD
Uses COMMITs to make changes permanent	Uses data saves
Redo log entries always generated	Generates redo only under specific conditions
Enforces all constraints	Enforces only primary key, unique, and not null constraints
INSERT triggers fire	INSERT triggers do not fire
Can load into clustered tables	Cannot load into clustered tables
Other users can make changes to tables	Other users cannot make changes to tables

Parallel Direct Loads



Using SQL*Loader

```
$sqlldr scott/tiger \
> control=ulcase6.ctl \
> log=ulcase6.log direct=true
```



SQL*Loader: Input Files

- Parameter file:
  Load options
- Control file:
  Load instructions
- Data file:
  Input records

```
LOAD DATA
INFILE 'ulcase6.dat'
INSERT
  INTO TABLE emp
(empno POSITION(01:04) INTEGER
  EXTERNAL NULLIF empno=BLANKS,
...
)
```

Log File Contents

- Header information
- Global information: Parameters and filenames
- Table information: Table and column specifications
- Data file information: Records processed
- Table load information: Errors and discards
- Summary statistics

SQL*Loader: Other Output Files

- Bad file
  - Rejected records
  - Same format as data files
- Discard file
  - Records not satisfying conditions
  - Same format as data files

SQL*Loader: Usage Guidelines

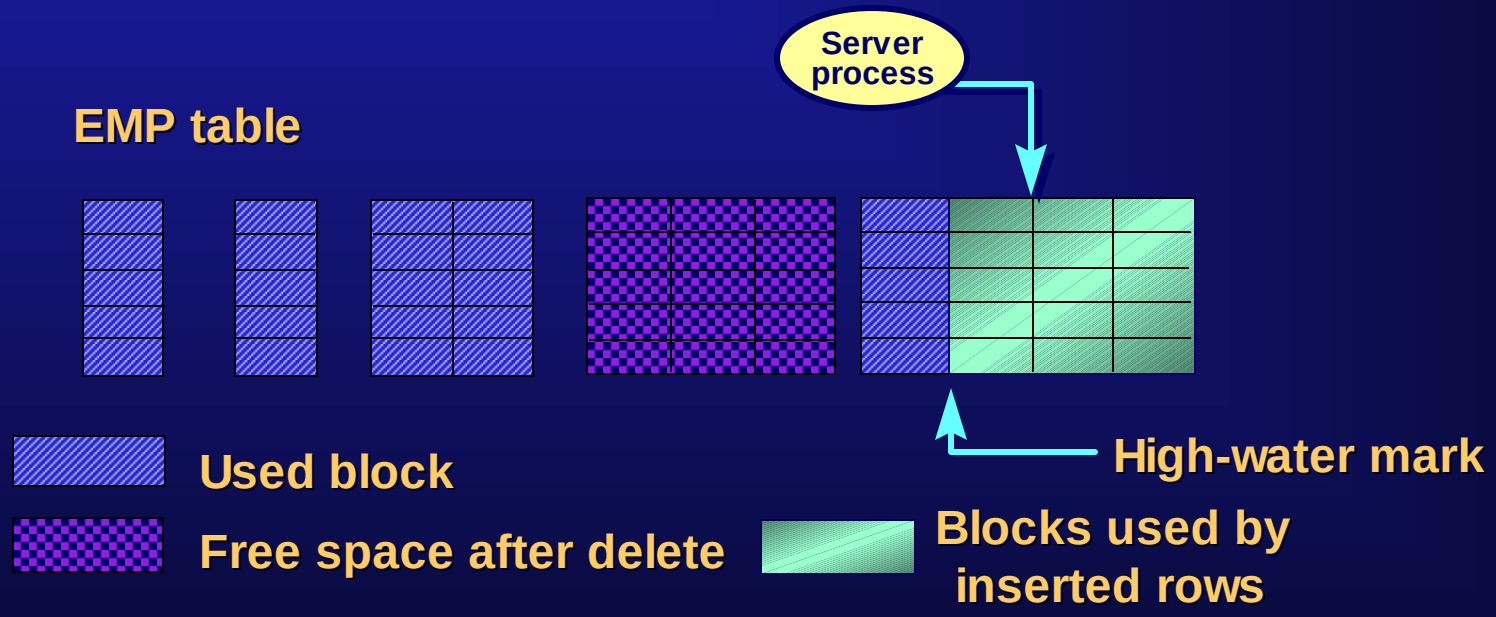
- Use a parameter file to specify commonly used command-line options.
- Place data in the control file only for a small, one-time load.
- Improve performance by:
  - Allocating sufficient space
  - Sorting the data on the largest index
  - Specifying different files for temporary segments for parallel loads

SQL*Loader: Troubleshooting

- Insufficient space for table or index
- Instance failure during load
- SORTED INDEXES clause is used and data is not in the order specified
- Duplicate keys found in a unique index, unique or primary key during a direct load
- BINDSIZE for conventional load cannot fit one row
- Errors or discards exceed specified limit

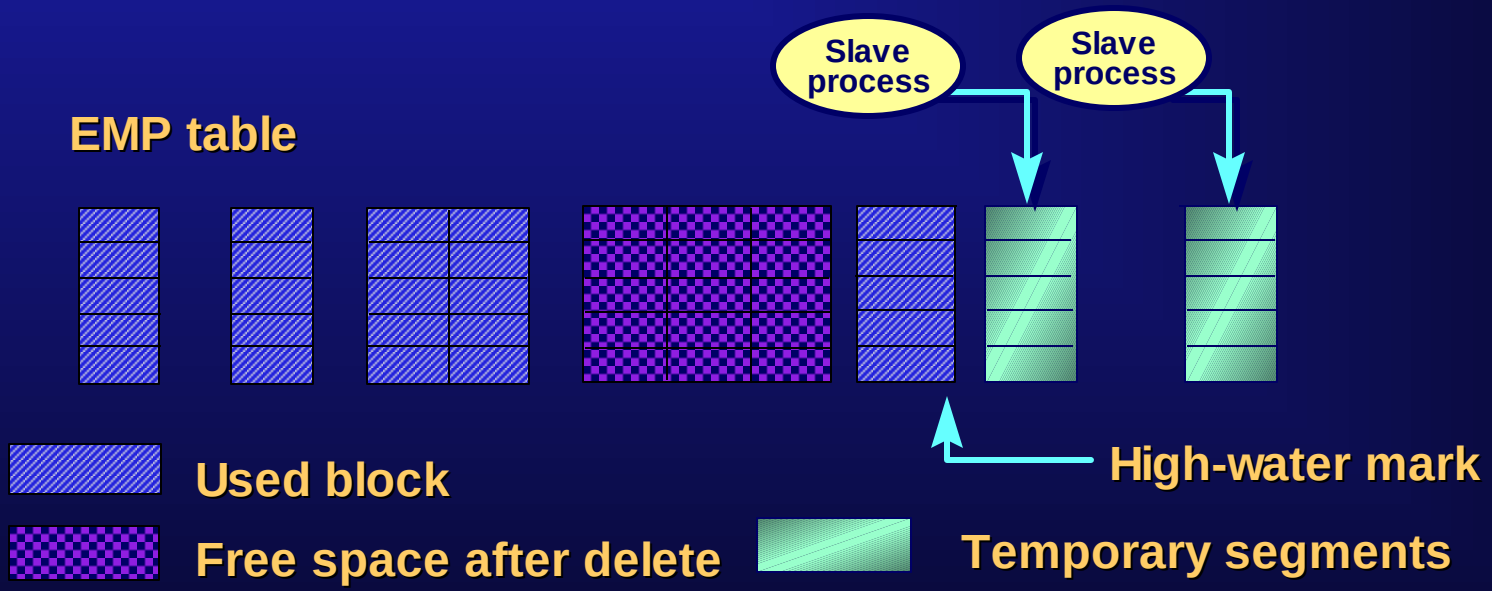
Using Direct-Load Inserts

```
INSERT /*+APPEND */ INTO scott.emp
NOLOGGING
SELECT * FROM scott.old_emp;
```

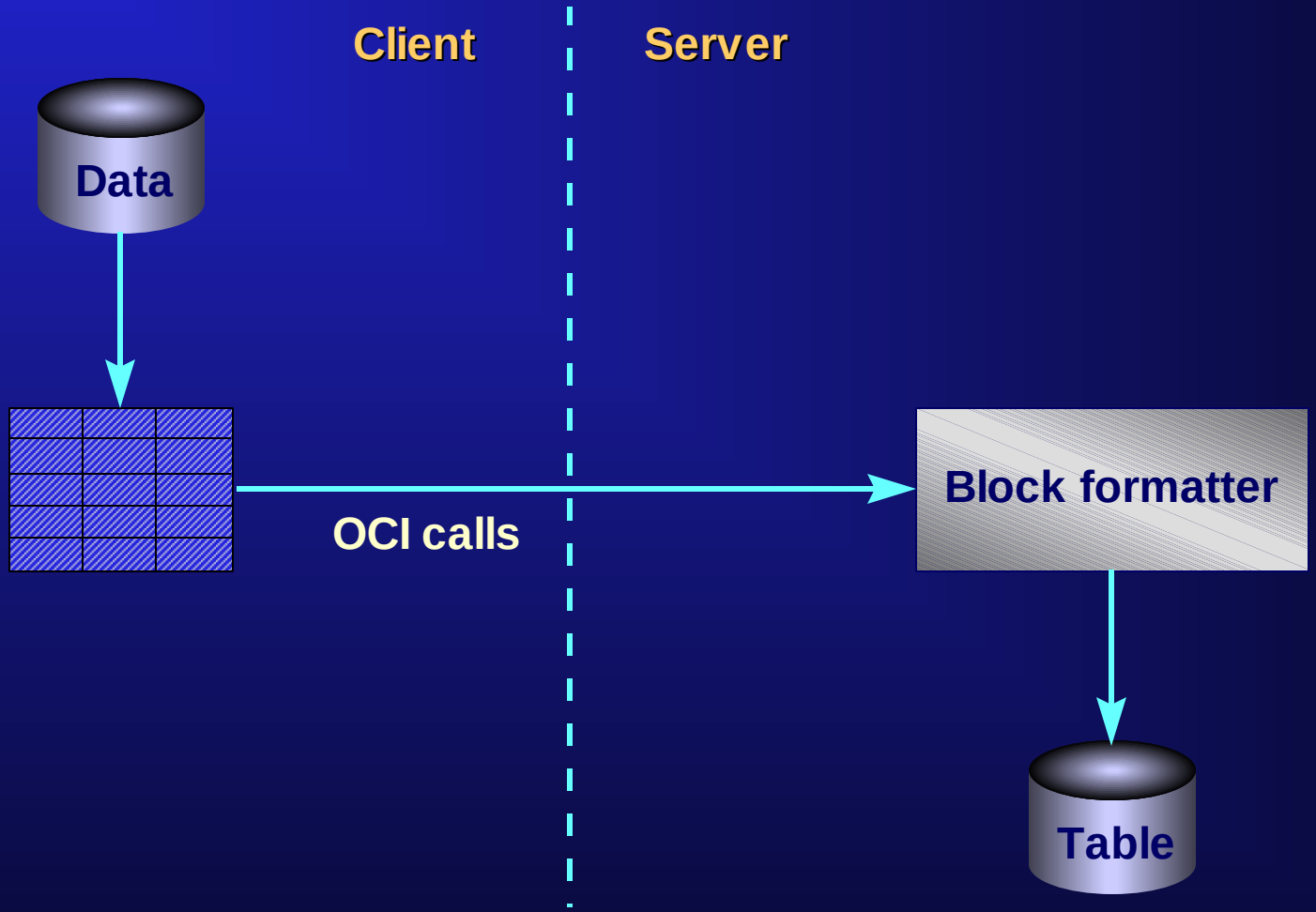


Parallel Direct-Load Insert

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+PARALLEL(scott.emp,2) */
    INTO scott.emp NOLOGGING
 SELECT * FROM scott.old_emp;
```



Direct Path Load API



Summary

Summary

In this lesson, you should have learned how to:

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method
- Use SQL*Plus to extract data

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method
- Use SQL*Plus to extract data
- Use UTL_FILE to read or write character data

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method
- Use SQL*Plus to extract data
- Use UTL_FILE to read or write character data
- Use direct-load insert to copy tables

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method
- Use SQL*Plus to extract data
- Use UTL_FILE to read or write character data
- Use direct-load insert to copy tables
- Use SQL*Loader to load data

Summary

In this lesson, you should have learned how to:

- Select the appropriate load or unload method
- Use SQL*Plus to extract data
- Use UTL_FILE to read or write character data
- Use direct-load insert to copy tables
- Use SQL*Loader to load data
- Use direct path mode when applicable

Practice Overview

This practice covers the following topics:

- Using SQL*Plus script file to unload the data from a table
- Using SQL*Loader to load the data into a table

12

Reorganizing Data

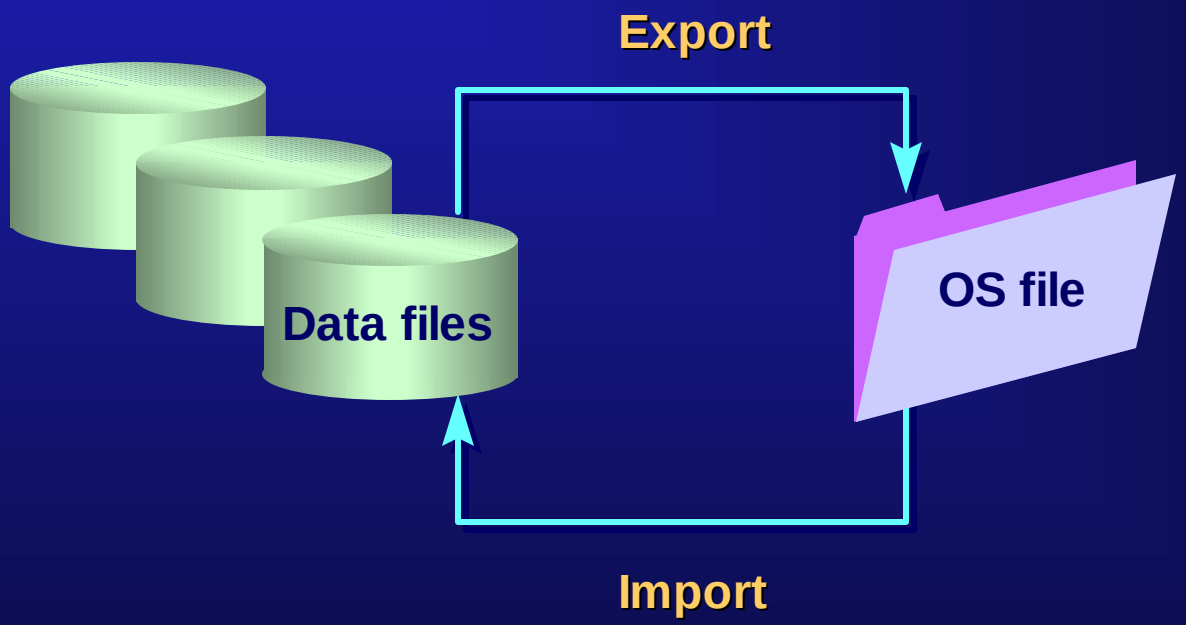
ORACLE

Objectives

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

- Describe how direct export works
- Export a schema or table
- Import data only
- Import objects
- Transport tablespaces

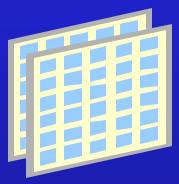
Moving Data Using Export and Import



Uses of Export and Import

- Reorganize tables
- Move data owned by one user to another user
- Move data between databases:
  - Development to production
  - OLTP system to a data warehouse
- Migrate the database to a different:
  - OS platform
  - Release of the Oracle database
- Repeat test runs during development or upgrade
- Perform a logical backup

Export Modes



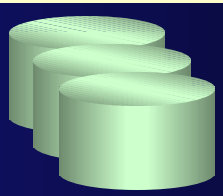
Table

- Table, indexes, constraints, and triggers
- Grants
- Analyze method

- All objects owned by user, except indexes on tables owned by other users

User

- All objects in the database (except objects owned by SYS)



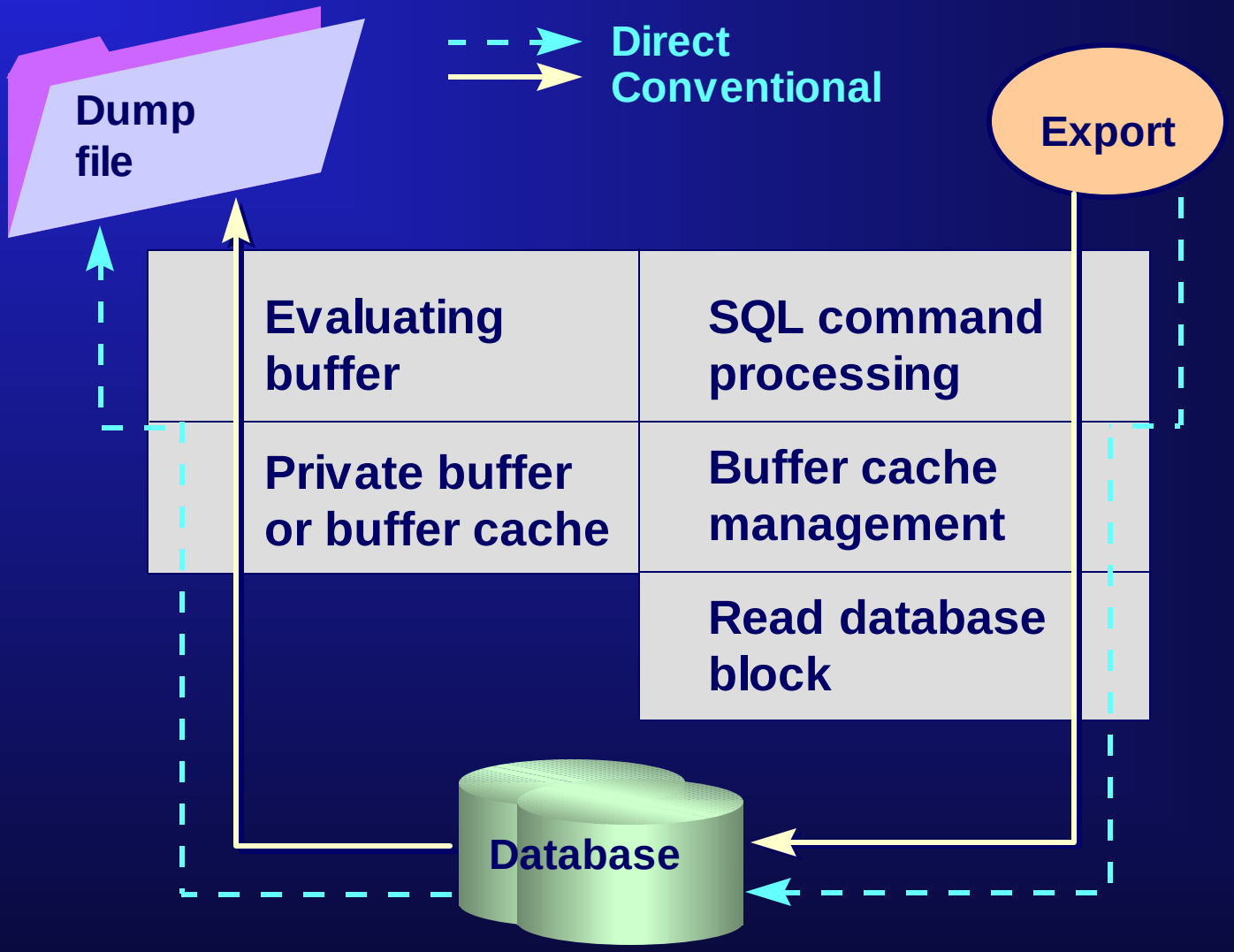
Database

Tablespace

- Table, cluster, and object type definitions
- Constraints, triggers, and bitmap indexes

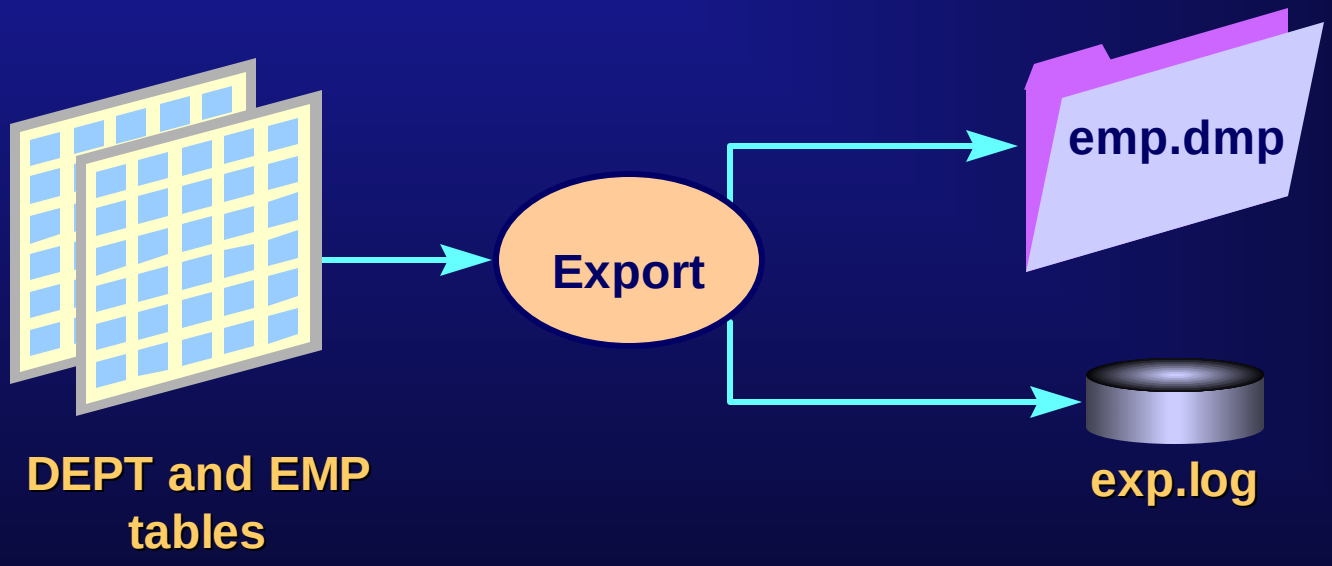
ORACLE

Conventional and Direct Path Export



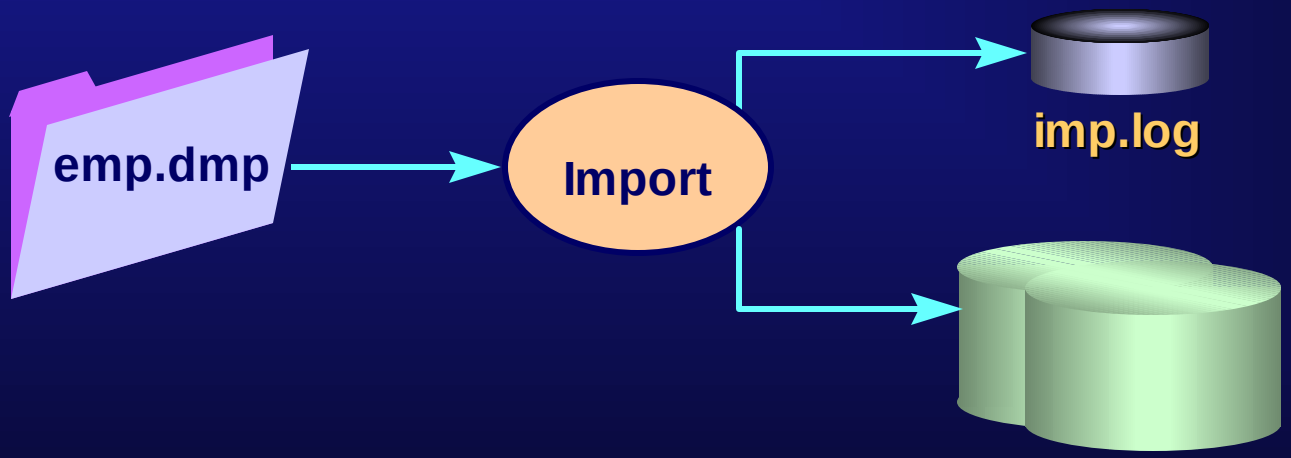
Using Export

```
$ exp scott/tiger TABLES=(dept,emp) \
>  FILE=emp.dmp LOG=exp.log COMPRESS=N \
>  DIRECT=Y RECORDLENGTH=32768
```



Using Import

```
$imp scott/tiger TABLES=(dept,emp) \
> FILE=emp.dmp LOG=imp.log IGNORE=Y
```



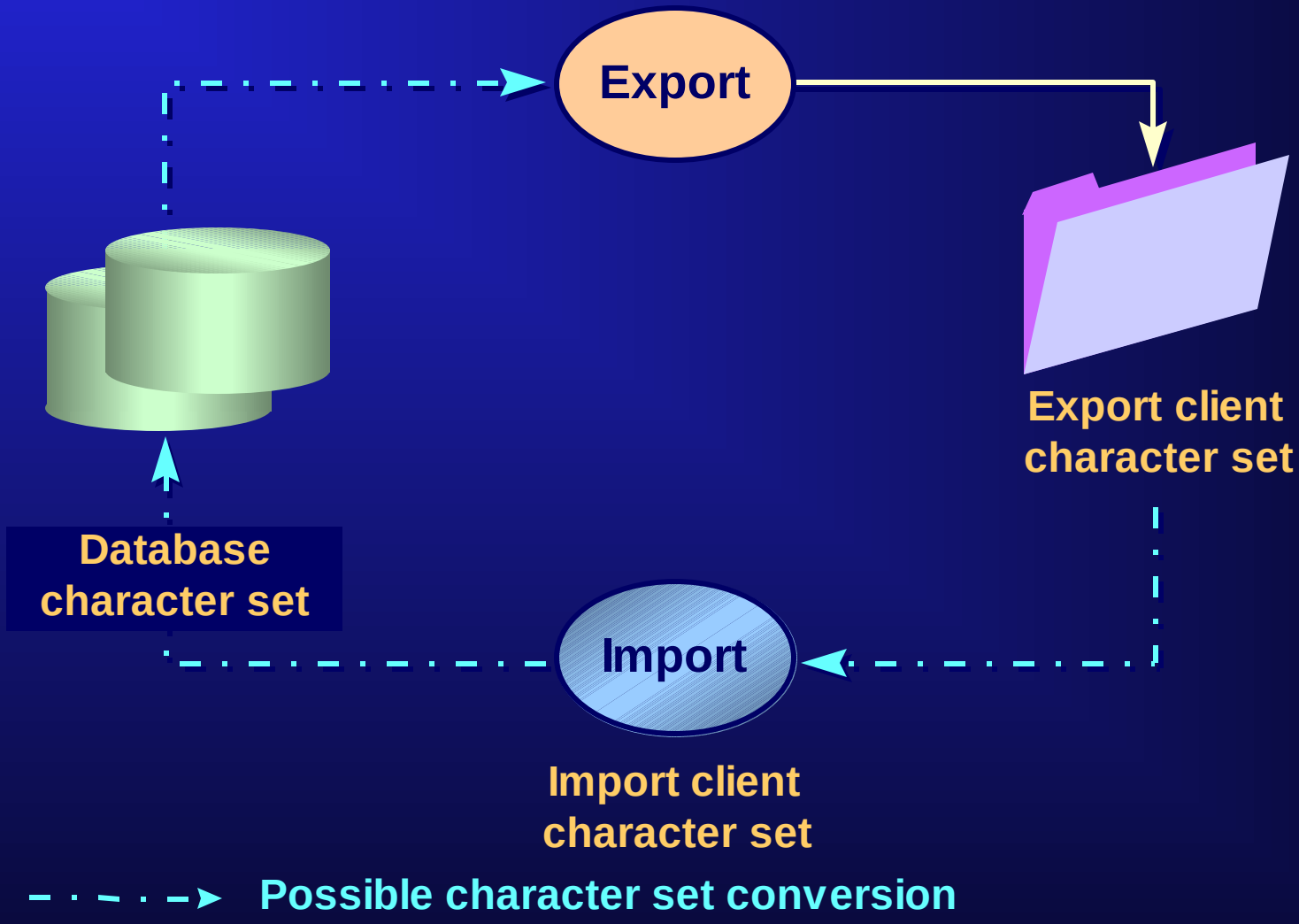
Import Behavior

- Order of import
  - Table → data → B-tree indexes → constraints, triggers, bitmap indexes
- Tablespace used for the object
  - Same tablespace as in the source database, if possible
  - User' s default tablespace

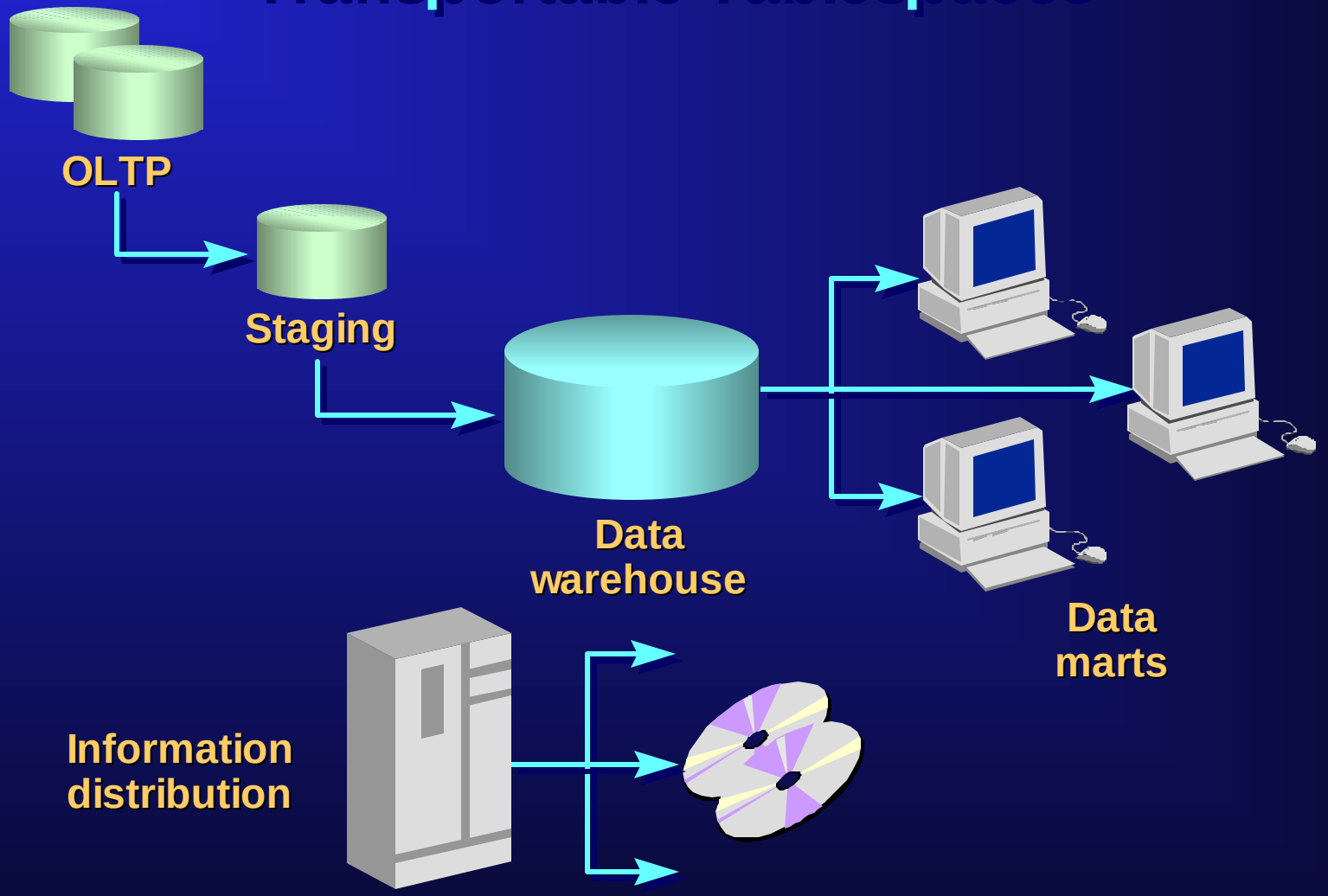
Using Export and Import: Guidelines

- Use a parameter file to specify commonly used command-line options
- Use CONSISTENT=Y only if exporting a small volume of data
- Do not use COMPRESS=Y if there are many deleted rows
- Improve performance by:
  - Allocating large buffer size
  - Using direct path if using only 7.3.3 or later

NLS Considerations in Export and Import



Data Transportation: Transportable Tablespaces



Transporting Tablespaces

1. Make tablespace read-only
2. Export metadata from source
3. Copy data files to target system
4. Transfer export file
5. Import metadata into target
6. Alter tablespace to read-write, if necessary

Uses and Requirements of Transportable Tablespaces

- Uses
  - Move entire tablespace data
  - Support media recovery
- Requirements: Source and target databases must have the following in common:
  - Be on the same operating system
  - Run Oracle8*i*, release 8.1 or later
  - Have the same block size
  - Use the same character set

Summary

Summary

In this lesson, you should have learned how to:

Summary

In this lesson, you should have learned how to:

- Choose between conventional and direct export

Summary

In this lesson, you should have learned how to:

- Choose between conventional and direct export
- Use the Export utility

Summary

In this lesson, you should have learned how to:

- Choose between conventional and direct export
- Use the Export utility
- Use the Import utility to reorganize data

Practice Overview

Practice Overview

This practice covers the following topics:

Practice Overview

This practice covers the following topics:

- Exporting a table definition

Practice Overview

This practice covers the following topics:

- Exporting a table definition
- Importing a table definition

Practice Overview

This practice covers the following topics:

- Exporting a table definition
- Importing a table definition
- Creating a CREATE TABLE command using an export file

Implementing Large Objects

ORACLE

Objectives

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

- Differentiate between LOB types
- Choose the appropriate LOB type
- Implement Oracle directory objects
- Create storage for LOB columns
- Appropriately set PCTVERSION and CHUNK
- Explain LOB read-consistency issues

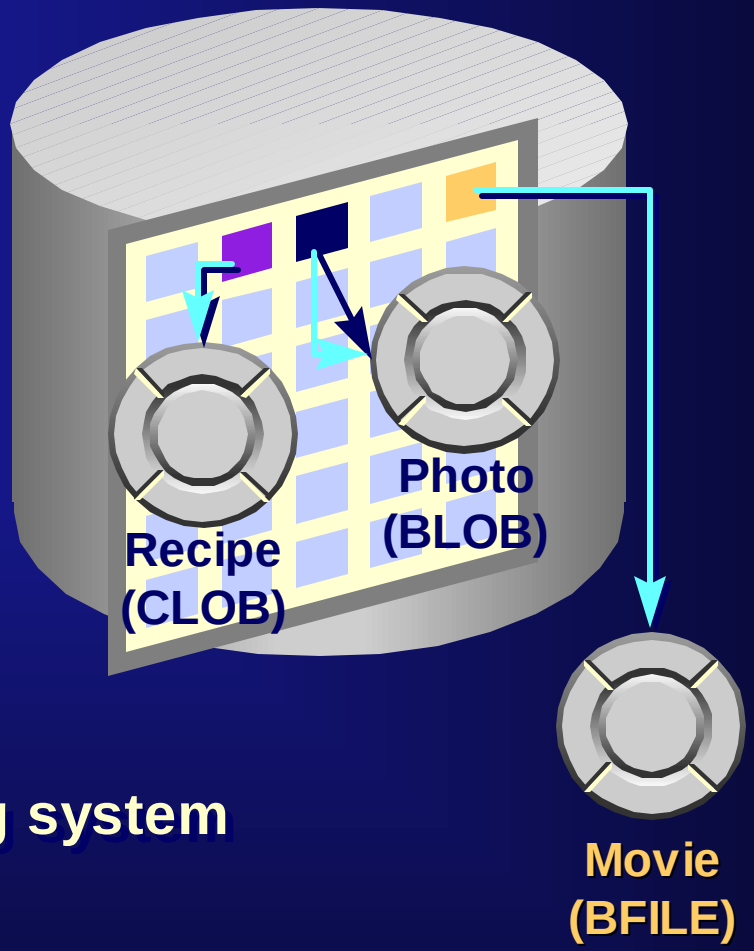
Overview

LOB storage:

- Unstructured data
- Size to 4 GB
- Special storage
- Special concurrency

Storage method:

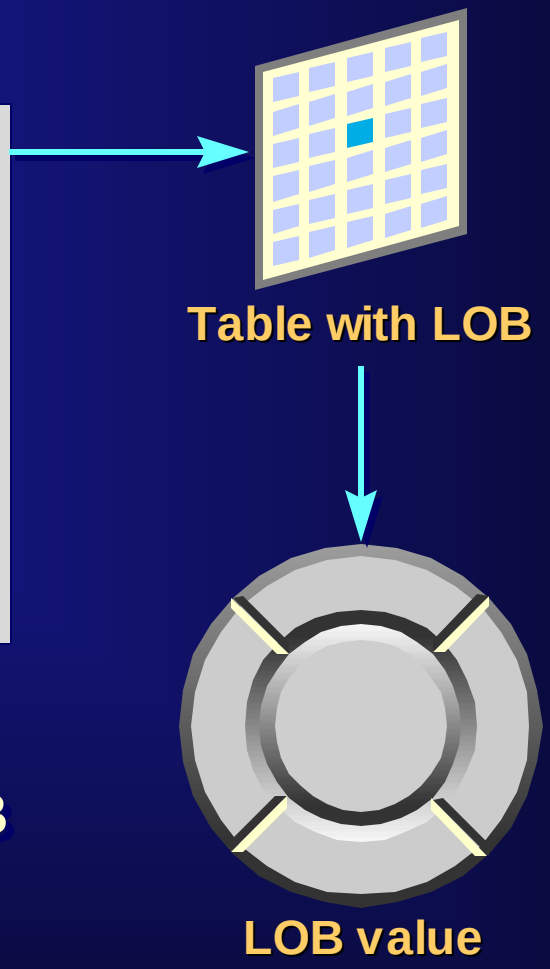
- Internal, in the database
- External, in the operating system



Characteristics of a LOB

Program with LOB locator

```
DECLARE
  jobloc BLOB;
BEGIN
...
  SELECT description INTO jobloc
   FROM job_table
   WHERE title='Analyst';
...
END;
```

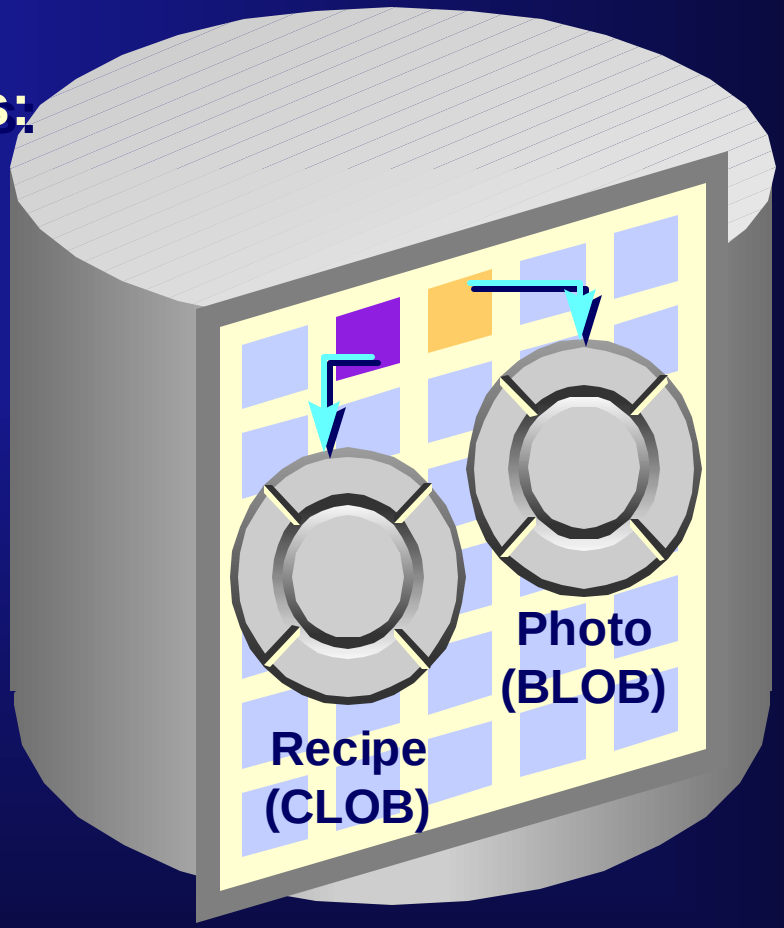


Two distinct parts of a LOB:

- Locator is a pointer to the LOB
- Value is the actual data

Internal LOBs

- Uses database features:
  - Concurrency
  - Recovery
  - Transactions
- LOB locator in row
- LOB value in:
  - Row
  - Another segment

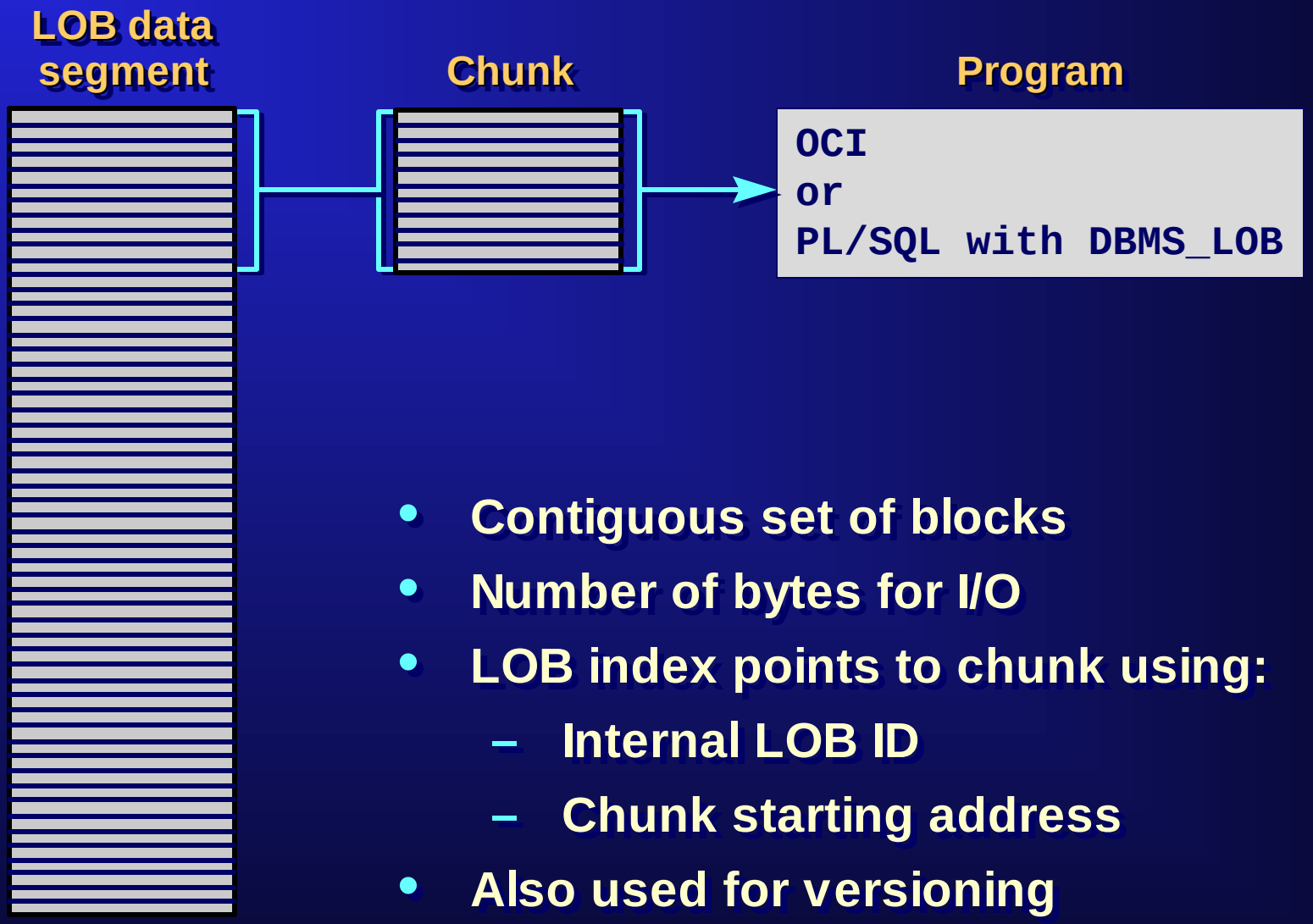


Internal LOB Storage

LOB value can be stored:

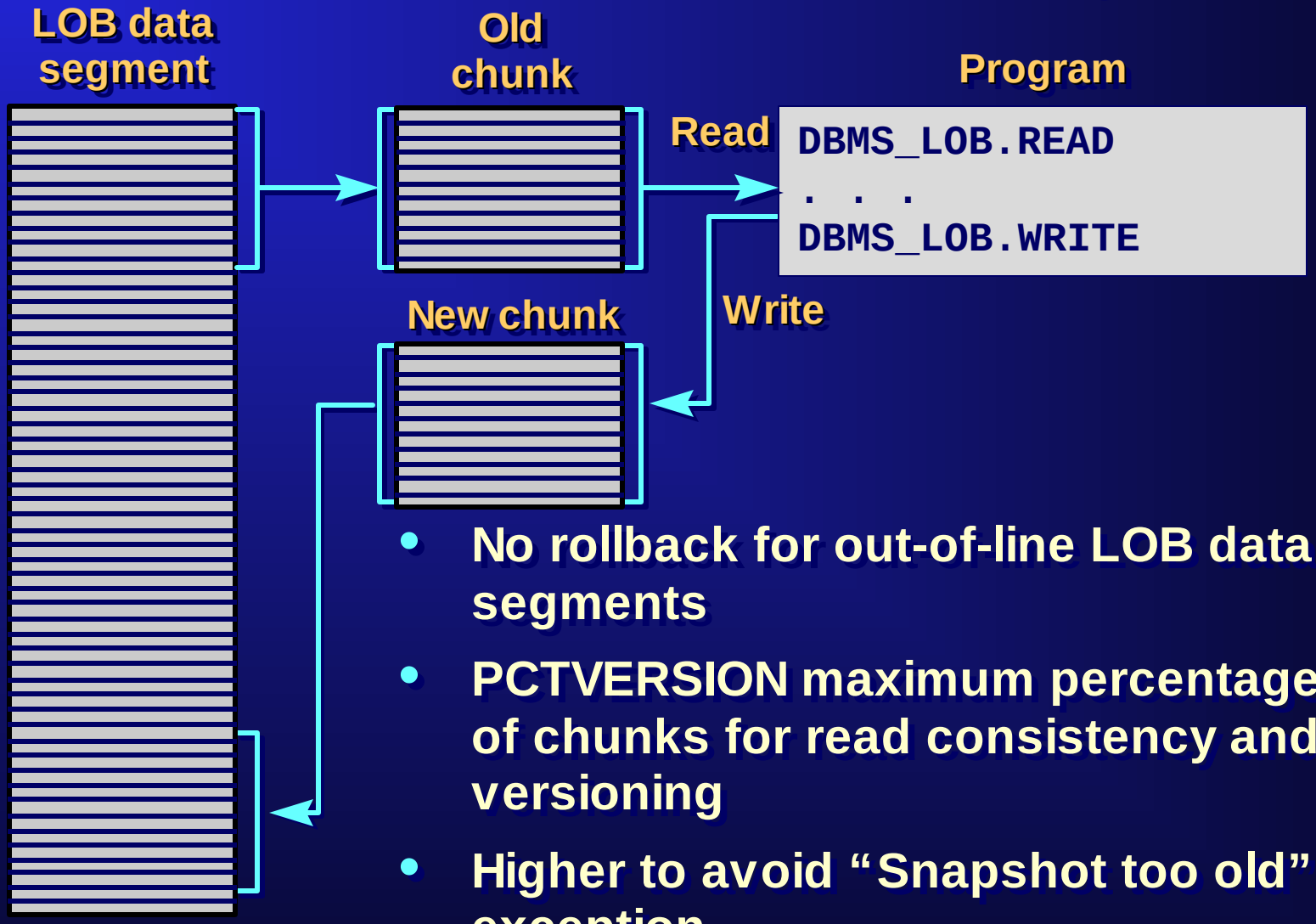
- In-line
  - Stored with the other row data
  - Only if 4,000 bytes or less
- Out-of-line
  - Stored in a separate segment
  - LOB index segment accesses LOB values
  - LOB data segment stores LOB values

Chunks



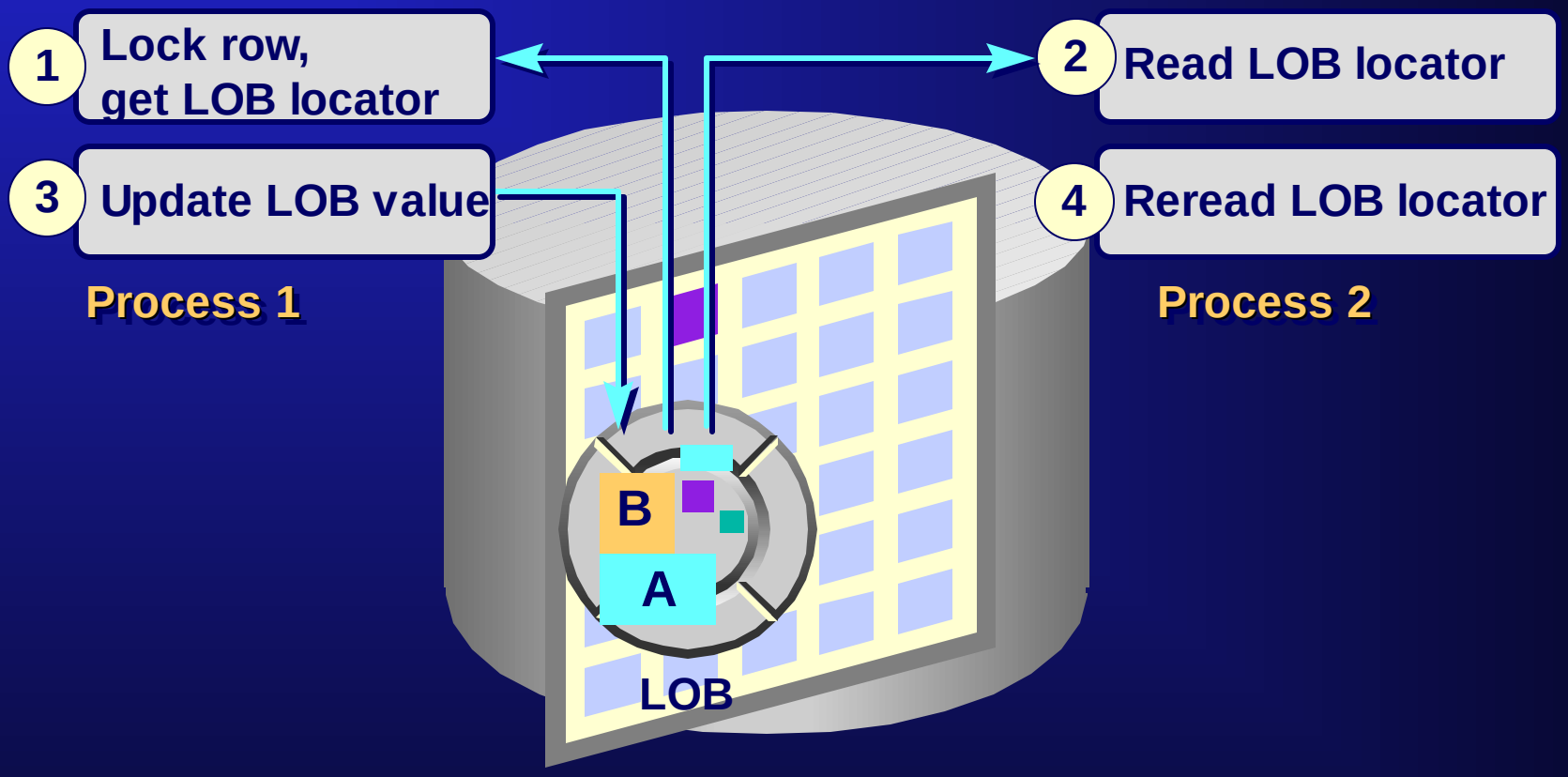
- Contiguous set of blocks
- Number of bytes for I/O
- LOB index points to chunk using:
  - Internal LOB ID
  - Chunk starting address
- Also used for versioning

Internal LOB Read Consistency



- No rollback for out-of-line LOB data segments
- PCTVERSION maximum percentage of chunks for read consistency and versioning
- Higher to avoid “Snapshot too old” exception

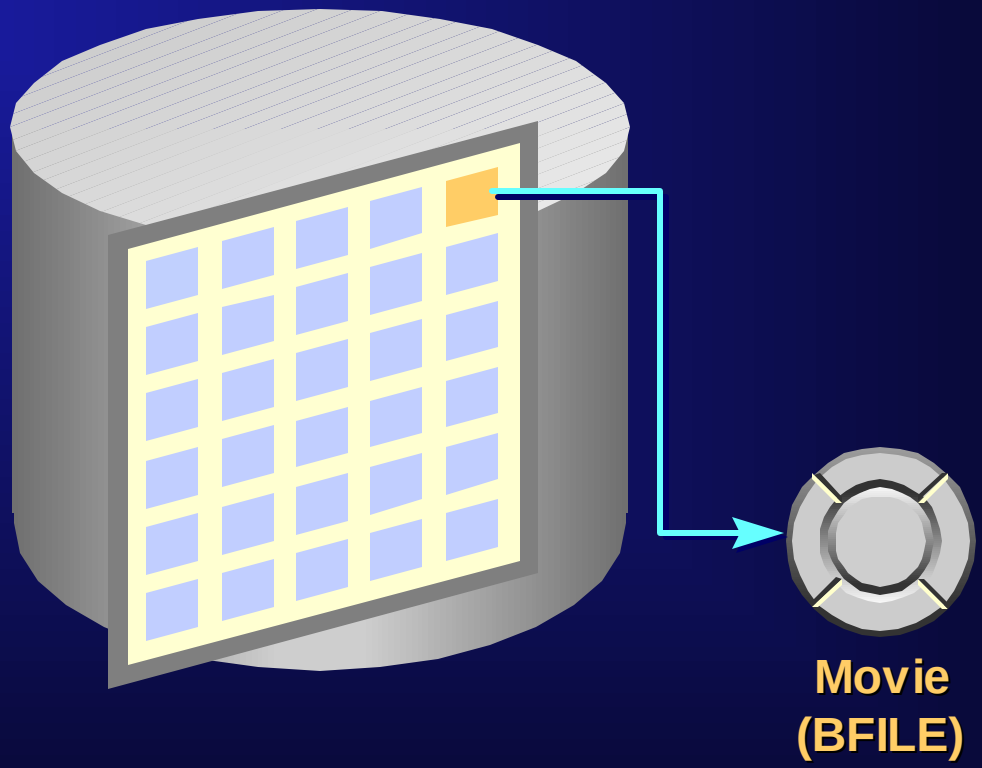
Concurrency and Internal LOBs



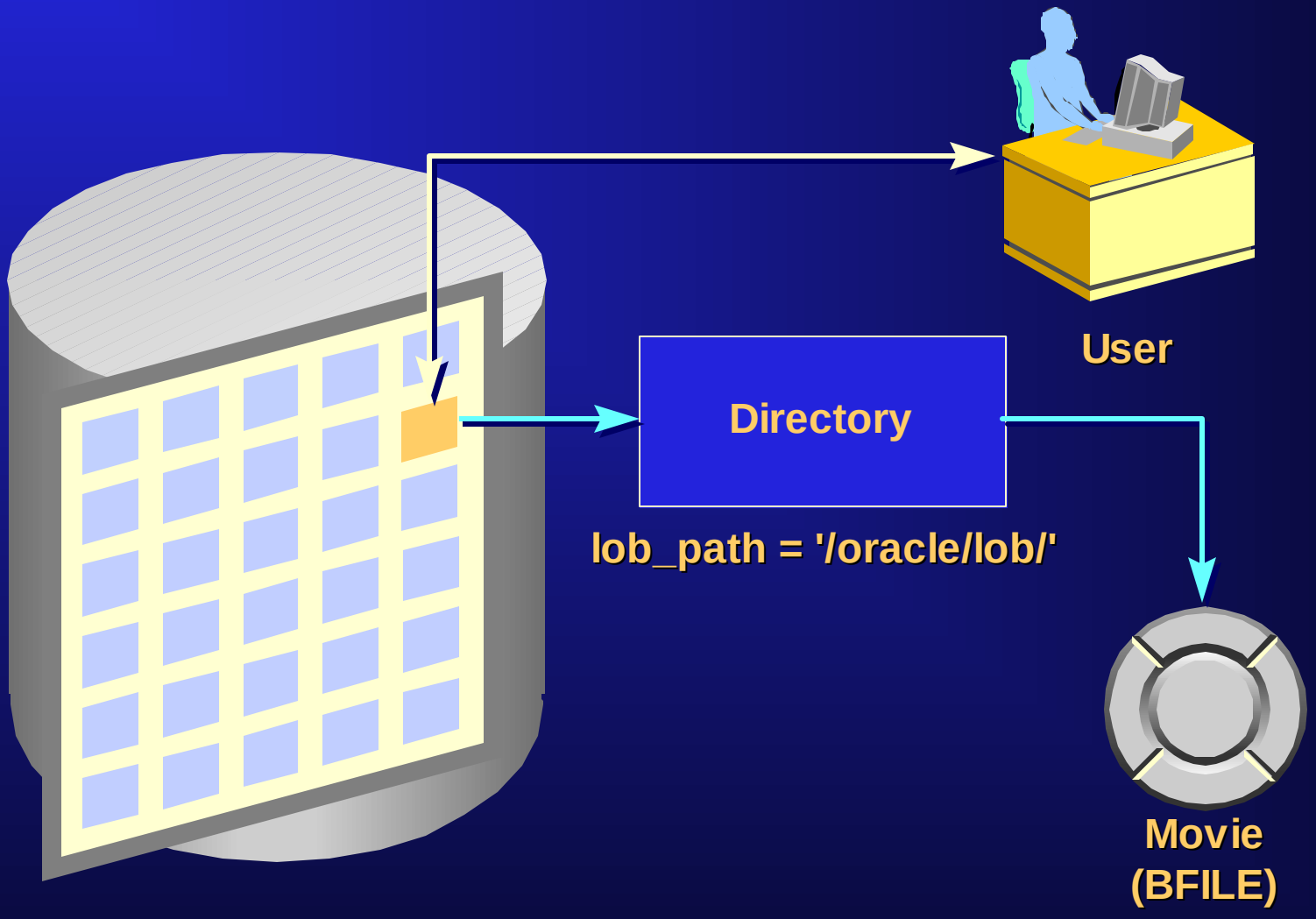
External LOBs

BFILE data type

- File-based LOB
- Read-only access
- Example: CDROM



Database Directory Objects



Creating an Internal LOB

```
CREATE TABLE apartments (
  apt_id      NUMBER,
  floor_plan  BLOB,
  contract    CLOB, ... )
  LOB (floor_plan, contract ) STORE AS (
    STORAGE ( INITIAL 100K          -- chunk multiple
              NEXT    100K
              PCTINCREASE 0 )
    CHUNK   20480             -- 10 blocks per chunk
    PCTVERSION 20             -- 20% for versioning
    NOCACHE
    NOLOGGING
    INDEX ( STORAGE ( INITIAL 100K NEXT 100K ) )
  );
```

Creating an External LOB

- Create the directory item:

```
SQL> CREATE DIRECTORY lob_path AS '/oracle/LOB';
```

- Grant access privileges to users:

```
SQL> GRANT READ ON DIRECTORY lob_path TO scott;
```

- Create the table with the external LOB:

```
SQL> CREATE TABLE home_page (
  2   emp_id         NUMBER(6),
  3   last_update    DATE,
  4   homepage       BFILE );
```

Guidelines for Internal LOBs

- BFILE exports only directory and filename
- Not analyzed in ANALYZE
- Not allowed in:
  - Partitioned tables
  - Clustered tables
  - Index-only tables
  - GROUP BY, ORDER BY, SELECT DISTINCT, aggregates, and joins

Guidelines for Internal LOBs

- Examples of default segment names:
  - LOB segment: SYS_LOB0000002032C00002$
  - LOB index segment: SYS_IL0000002032C00002$
- Separate tablespaces for table and LOB
- Default tablespaces:
  - Table segment default: Table’ s tablespace
  - LOB segment and LOB index default: LOB segment’ s tablespace

Guidelines for Internal LOBS

- **CHUNK**
  Use a multiple when:
  - **Setting INITIAL and NEXT extent sizes**
  - **Reading or writing LOBs with OCI or DBMS_LOB**
- **PCTVERSION**
  Depends on accesses:
  - **Low if updates and reads done at different times or read-only; low if written once and then read-only**
  - **High if large queries; high if heavy write and read activity**

Guidelines for Internal LOBs

- CACHE/NOCACHE
  - CACHE if read frequently by multiple users
  - NOCACHE to avoid using DB buffer cache blocks
- LOGGING/NO LOGGING
  Use NOLOGGING when:
  - Recovery not required
  - Bulk loading, backup tablespace after load

Guidelines for Internal LOBS

- DISABLE STORAGE IN ROW
  - Improves performance when other columns are frequently read without the LOB value
- ENABLE STORAGE IN ROW
  - Moves LOB out-of-line when size >4,000 bytes
  - Improves performance when small LOBs frequently read with rows
  - Hurts performance of full table scans that do not need LOB value

Guidelines for External LOBS

- Useful for read-only LOBs
- Avoid duplicate storage of data in operating system and database
- Administered in operating system:
  - File for external LOB is created, updated, or deleted without using SQL
  - Recovery provided by operating system
  - Operating system restrictions can prevent LOB access

Temporary LOBs

- Provide an interface to support creation of LOBs that act like local variables
- Faster than using persistent LOBs
- Reside in temporary tablespaces
- Can be created using DBMS_LOB package and CREATETEMPORARY procedures

Creating a Temporary LOB

PL/SQL procedure to create and test a temporary LOB

```
SQL> CREATE OR REPLACE PROCEDURE
  2    IsTempLOBOpen(Lob_loc IN OUT BLOB,
  3                  Retval OUT INTEGER)
  4  IS
  5  BEGIN
  6   /* Create the temporary LOB: */
  7    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
  8  /* See If the LOB is open: returns 1 if open*/
  9    Retval := DBMS_LOB.ISOPEN(Lob_loc);
 10  /* Free the temporary LOB: */
 11    DBMS_LOB.FREETEMPORARY(Lob_loc);
 12  END;
```

LOB Data Dictionary Views

```
SQL> CREATE TABLE persons (
  2    pname          VARCHAR2(50),
  3    resume         CLOB,
  4    picture        BFILE );
. . .
SQL> SELECT   segment_type, segment_name
  2    FROM   user_lobs      l,
  3           user_segments  s
  4    WHERE  l.segment_name = 'PERSONS'
  5      AND  s.segment_name IN (l.segment_name,
  6                  l.table_name, l.index_name);

SEGMENT_TYPE   SEGMENT_NAME
------------   ------------------------
TABLE          PERSONS
LOBINDEX       SYS_IL0000002032C00003$
LOBSEGMENT     SYS_LOB0000002032C00003$
```

LOB Data Dictionary Views

```
SQL> COLUMN table_name  FORMAT A10
SQL> COLUMN column_name FORMAT A11
SQL> SELECT  table_name, column_name, chunk,
  2          pctversion, cache, logging, in_row
  3    FROM user_lobs;

TABLE_NAME COLUMN_NAME   CHUNK PCTVERSION CAC LOG IN_
---------- ---------- ------- ---------- --- --- ---
APARTMENTS FLOOR_PLAN     2048         10 NO  YES YES
APARTMENTS CONTRACT       2048         10 NO  YES YES
```

Summary

- Data types: BLOB, CLOB, NCLOB, BFILE
- Two storage options for LOBs
  - Internal LOBs in tables
    - In-line stored with row
    - Out-of-line stored in segment
    - CHUNK determines I/O and storage size
    - PCTVERSION for read consistency
  - External LOBs (BFILE) in OS files
- LOB data in USER_LOBS and ALL_LOBS data dictionary views

Practice Overview

This practice covers the following topics:

- Creating a table that contains a LOB column
- Querying the data dictionary view to display the LOB segment details

B

Using National Language Support

ORACLE

Objectives

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

- Describe National Language Support (NLS)
- Explain parameters affecting the NLS environment
- Obtain information about NLS usage
- Write a multilanguage application

NLS Features

- Language support
- Territory support
- Character set support
- Linguistic sorting
- Message support
- Date and time formats
- Numeric formats
- Monetary formats

Types of Encoding Schemes

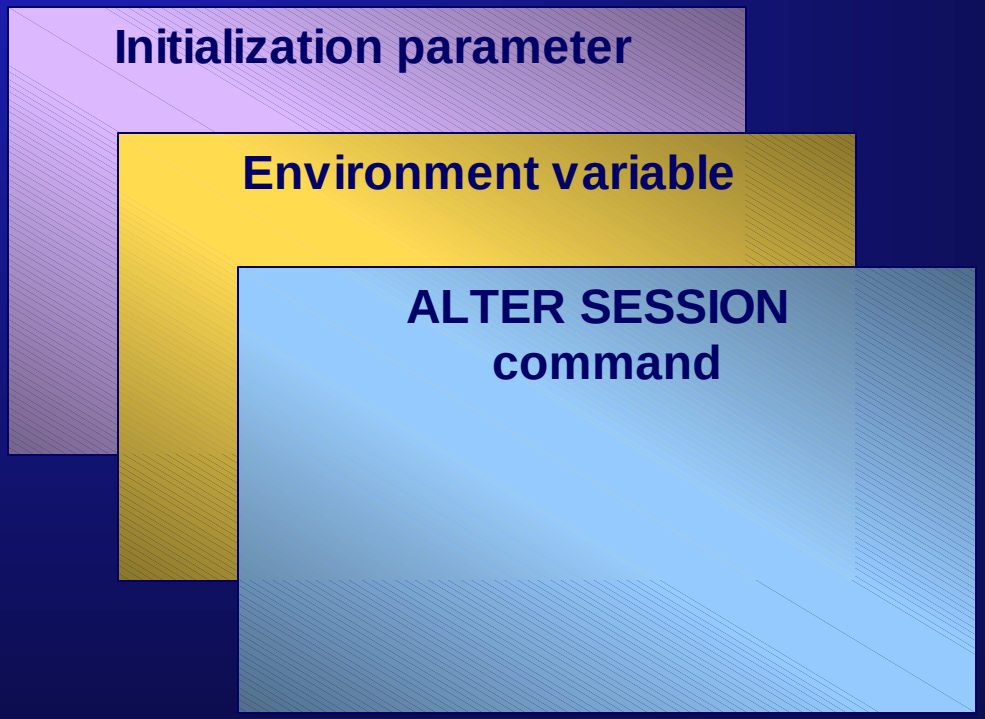
The Oracle server supports different classes of character encoding schemes:

- Single-byte character sets
  - 7 bit
  - 8 bit
- Varying-width multibyte character set
- Fixed-width multibyte character set
- Unicode (UTF8, AL24UTFFSS)

Character Sets and National Character Sets of a Database

DATABASE CHARACTER SETS	NATIONAL CHARACTER SETS
Defined at creation time	Defined at creation time
Cannot be changed without re-creation	Cannot be changed without re-creation
Store data columns of type CHAR, VARCHAR2, CLOB, and LONG	Store data columns of type NCHAR, NVARCHAR2, and NCLOB
Can store varying-width character sets	Can store fixed-width and varying-width multibyte character sets

Specifying Language-Dependent Behavior



Language-Dependent Behavior on the Server

- NLS_LANGUAGE specifies:
  - The language for messages
  - Day and month names
  - Symbols for a.m., p.m., A.D., and B.C.
  - The default sorting mechanism
- NLS_TERRITORY specifies:
  - Day and week numbering
  - Default date format, decimal character, group separator, default ISO, and local currency symbols

Dependent Language and Territory Default Values

PARAMETER	VALUES
NLS_LANGUAGE	AMERICAN
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
NLS_TERRITORY	AMERICA
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA
NLS_DATE_FORMAT	DD-MON-YY
NLS_NUMERIC_CHARACTERS	,.

Specifying Language-Dependent Behavior for the Session

- Environment variable:
  - NLS_LANG=<language>_<territory>.<charset>
- Additional environment variables:
  - NLS_DATE_FORMAT
  - NLS_DATE_LANGUAGE
  - NLS_SORT
  - NLS_NUMERIC_CHARACTERS
  - NLS_CURRENCY
  - NLS_ISO_CURRENCY
  - NLS_CALENDAR

Specifying Language-Dependent Behavior for the Session

```
ALTER SESSION SET NLS_DATE_FORMAT='DD.MM.YYYY';
```

```
DBMS_SESSION.SET_NLS('NLS_DATE_FORMAT',
'''DD.MM.YYYY''') ;
```

Sorting

- The Oracle server provides a linguistic sort.
- NLS_SORT specifies types of sort.
- The NLSSORT function reflects linguistic comparison.

```
ALTER SESSION SET NLS_SORT=GERMAN;
SELECT letter FROM letters ORDER BY letter;
LETTER
------
ä
z
```

Using NLS Parameters in SQL Functions

```
SELECT TO_CHAR(hiredate,'DD.MON.YYYY',
'NLS_DATE_LANGUAGE=GERMAN') FROM emp;
```

```
SELECT ename, TO_CHAR(sal,'9G999D99',
'NLS_NUMERIC_CHARACTERS='',. ''')
FROM emp;
```

NLS Linguistic Index Support

- Linguistic indexing
- High performance with local sorting

```
SQL> CREATE INDEX nls_ename ON emp
  2    (NLSSORT(ename,'NLS_SORT=German'));
```

- NLS_COMP parameter for linguistic comparisons

Importing and Loading Data Using NLS

- Data will be converted from NLS_LANG to the database character set during the import.
- Loader:
  - Conventional: Data is converted into the session character set specified by NLS_LANG.
  - Direct: Data is converted directly into the database character set.

Multilanguage Applications

- Different sets of programs
- Language-dependent text in tables

```
SQL> SELECT  text
  2  FROM    lang_text           t,
  3          v$nls_parameters    n
  4  WHERE n.parameter = 'NLS_LANGUAGE'    -- session language
  5    AND t.language = n.value            -- message language
  6    AND t.type = 'M'                    -- M for message
  7    AND t.id = 'APPL-18402';            -- message code

TEXT
--------------------------------------------------
The credit rating must be "Excellent", "Good", or "Fair".
```

Multilanguage Applications

- Client location
  - Change directories depending on NLS_LANG
  - Query V$NLS_PARAMETERS
- User selection
  - Change directories depending on user selection
  - Issue ALTER SESSION depending on user selection
- Multiple languages at once
  - Use NLS parameters in SQL functions

Obtaining Information About Character Sets

NLS_DATABASE_PARAMETERS

- **PARAMETER (NLS_CHARACTERSET, NLS_NCHAR_CHARACTERSET)**
- **VALUE**

Obtaining Information About NLS Settings

- NLS_INSTANCE_PARAMETERS
  - PARAMETER (NLS initialization parameters that have been explicitly set)
  - VALUE
- NLS_SESSION_PARAMETERS
  - PARAMETER (NLS session parameters)
  - VALUE

Obtaining Information About NLS Settings

- V$NLS_VALID_VALUES
  - PARAMETER (LANGUAGE, SORT, TERRITORY, CHARACTERSET)
  - VALUE
- V$NLS_PARAMETERS
  - PARAMETER (NLS session parameters, NLS_CHARACTERSET)
  - VALUE

VLDB Performance Enhancements: Working with Partitions

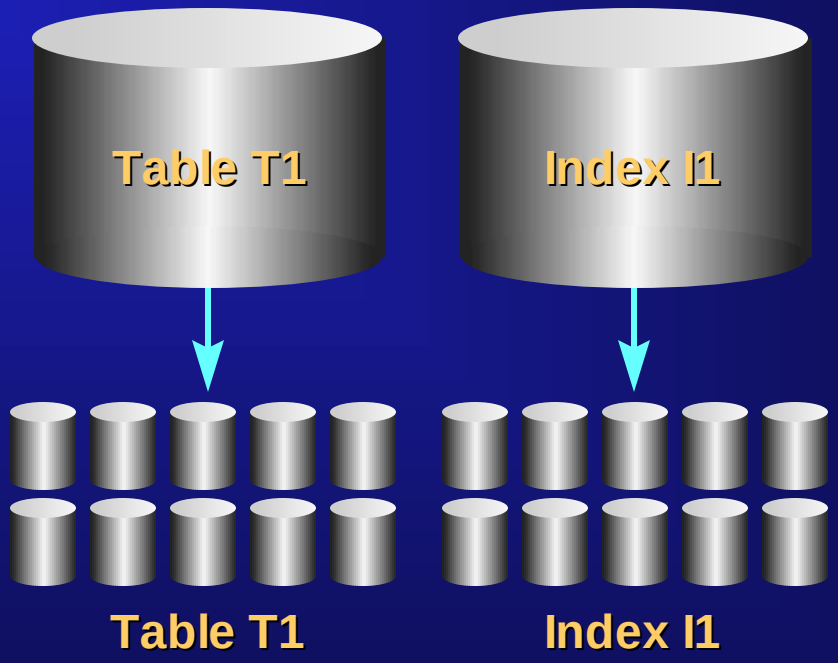
ORACLE

Objectives

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

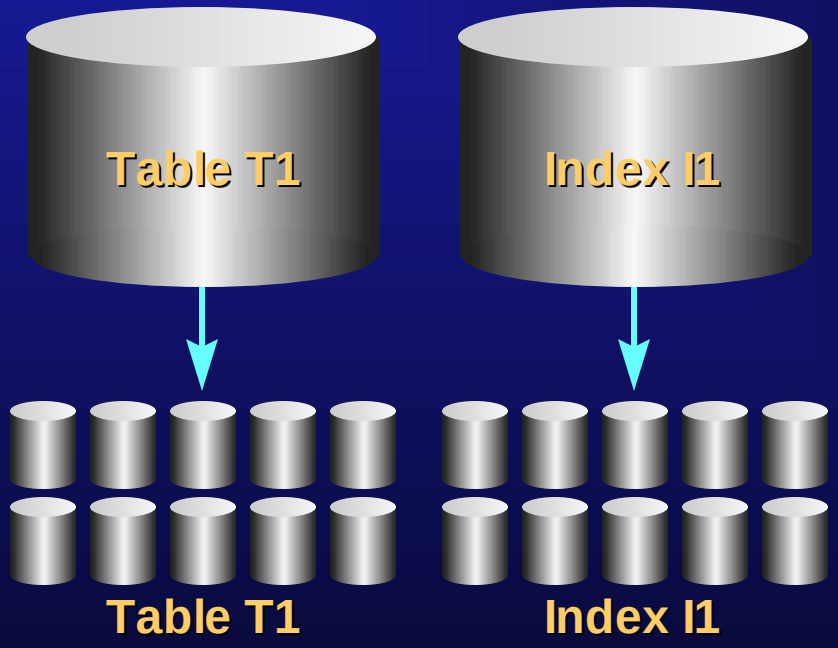
- Create partitioned tables and indexes
- Load partitioned tables and indexes using parallel DML
- Convert from partitioned views to partitioned tables

Partitioned Tables and Indexes



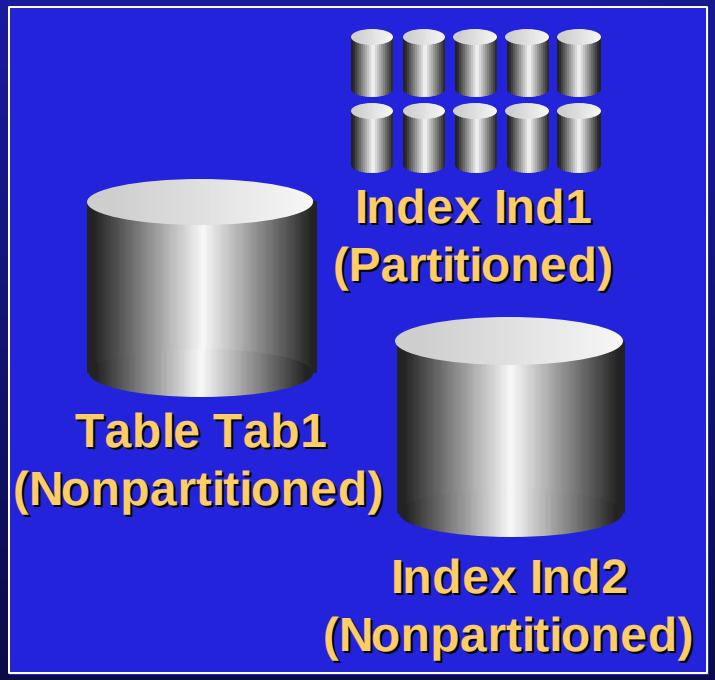
Partitioning Tables and Indexes

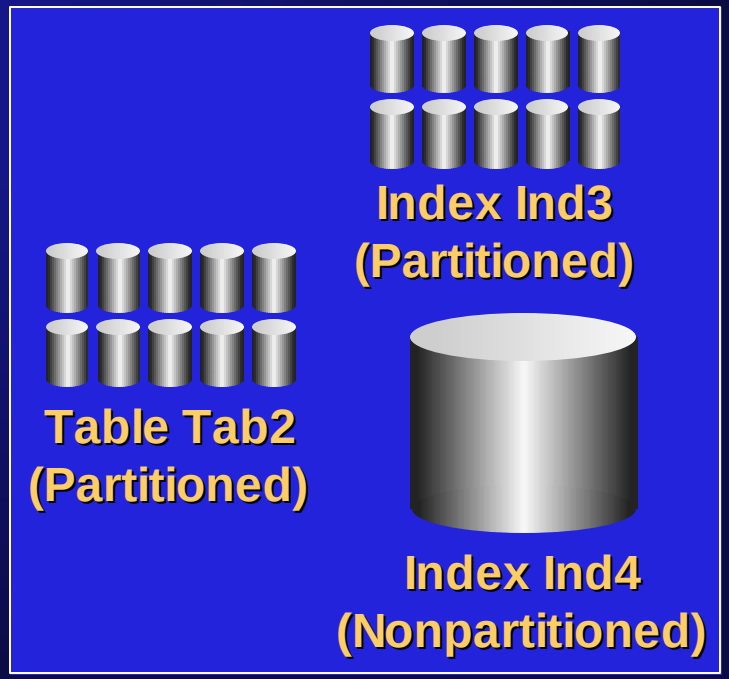
- Same logical makeup
- Distinct physical construction
- Partitioned by values



Indexing Partitioned Tables

Both partitioned and nonpartitioned tables can have partitioned and/or nonpartitioned indexes.





Managing Partitions

- Manage partitions independently.
- Back up and restore partitions individually.
- Make some partitions unavailable without affecting queries or DML operations on other partitions.

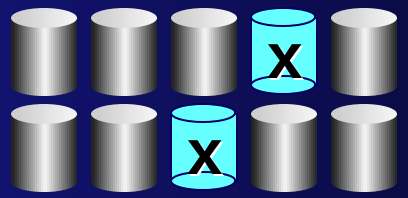
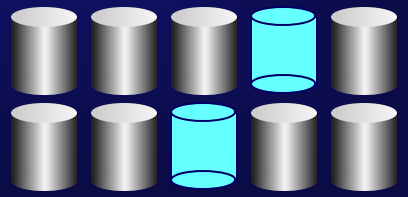


Table T1



Index I1

What Can Be Partitioned?

- Tables
  - Range (most common technique)
  - Hash (8.1)
  - Composite (8.1)
- Index-organized tables (8.1)
  - Range
- Indexes
  - Global (for range-partitioned tables)
  - Local (for all partitioned tables)

Specifying Partitioning for Tables

- Choose the partitioning method
- Choose the partitioning key (the columns to partition on)
  - Up to 16 columns can be specified as partitioning key
- Specify for each partition:
  - Range bound (range only)
  - Storage clause or tablespace
- Storage defaults inherited from table or tablespace if not specified

```
CREATE TABLE emp
   (empno    NUMBER(4),
    ename    VARCHAR2(10),
    job      VARCHAR2(9),
    sal      NUMBER(7,2),
    deptno   NUMBER(2),
    pname    varchar2(25))
  PARTITION BY RANGE (deptno)
 (PARTITION emp_p1 VALUES
    LESS THAN (10)
    tablespace tbs_1,
  PARTITION emp_p2 VALUES
    LESS THAN (20)
    tablespace tbs_2,
 PARTITION emp_p3 VALUES
    LESS THAN (30)
    tablespace tbs_3,
 PARTITION emp_p4 VALUES
    LESS THAN (MAXVALUE)
    tablespace tbs_4);
```

Range Partitioning

- Oracle8 release 8.0 introduced key-range partitioning:

- Ideal for chronological data

Partitioning Components

```
SVRMGR> CREATE TABLE sales
     2>   ( acct_no  NUMBER(5),
     3>     person   VARCHAR2(30),
     4>     sales_amount NUMBER(8),
     5>     week_no  NUMBER(2) )
     6>   PARTITION BY RANGE (week_no)                    1
     7>    ( PARTITION P1 VALUES LESS THAN  (4)
     8>        TABLESPACE data0,
     9>      PARTITION P2 VALUES LESS THAN  (8)           2
    10>        TABLESPACE data1,
           ......
    31>      PARTITION P10 VALUES LESS THAN (40)
    32>        TABLESPACE data10);                        3
```

Using MAXVALUE in a Partition

- Can be specified for one or more columns in VALUES LESS THAN
- Places data having a higher binary value than VALUES LESS THAN literal in the partition below where MAXVALUE is specified

```
SQL> CREATE TABLE sales
     . . .
 10>  PARTITION P11 VALUES LESS THAN (MAXVALUE)
 11>  TABLESPACE data11);
```

Partitioned Table Characteristics

- Partition key:
  - Can contain a concatenated key up to 16 columns
  - Cannot be updated if the change would cause moving the row to another partition (8.0 only)
- Clustered tables cannot be partitioned.
- LONG, LONG RAW, LOB, and ROWID data types are not supported in a partitioned table.

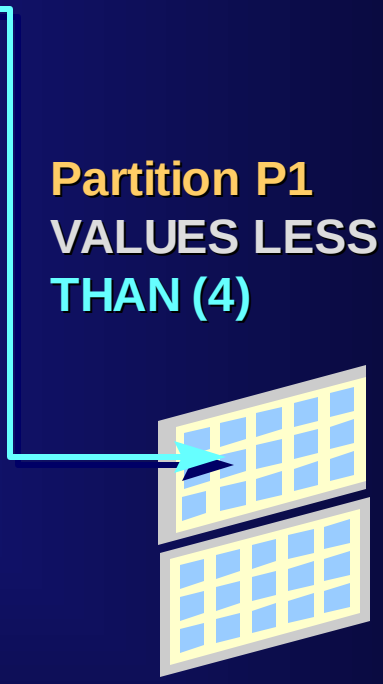
Manipulating Data in Partitions

```
SQL> INSERT INTO sales VALUES
  2> (241, 'Devin',2000,3);
```

```
SQL> INSERT INTO sales VALUES
  2>  (242, 'Pollock',1100,55)
```

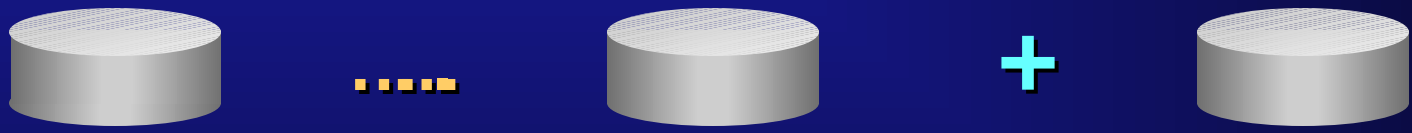
```
SQL> UPDATE sales SET week_no=1
  2> WHERE week_no=3;
```

```
SQL> UPDATE sales SET week_no=15
  2> WHERE week_no=1;
```



Adding New Partitions

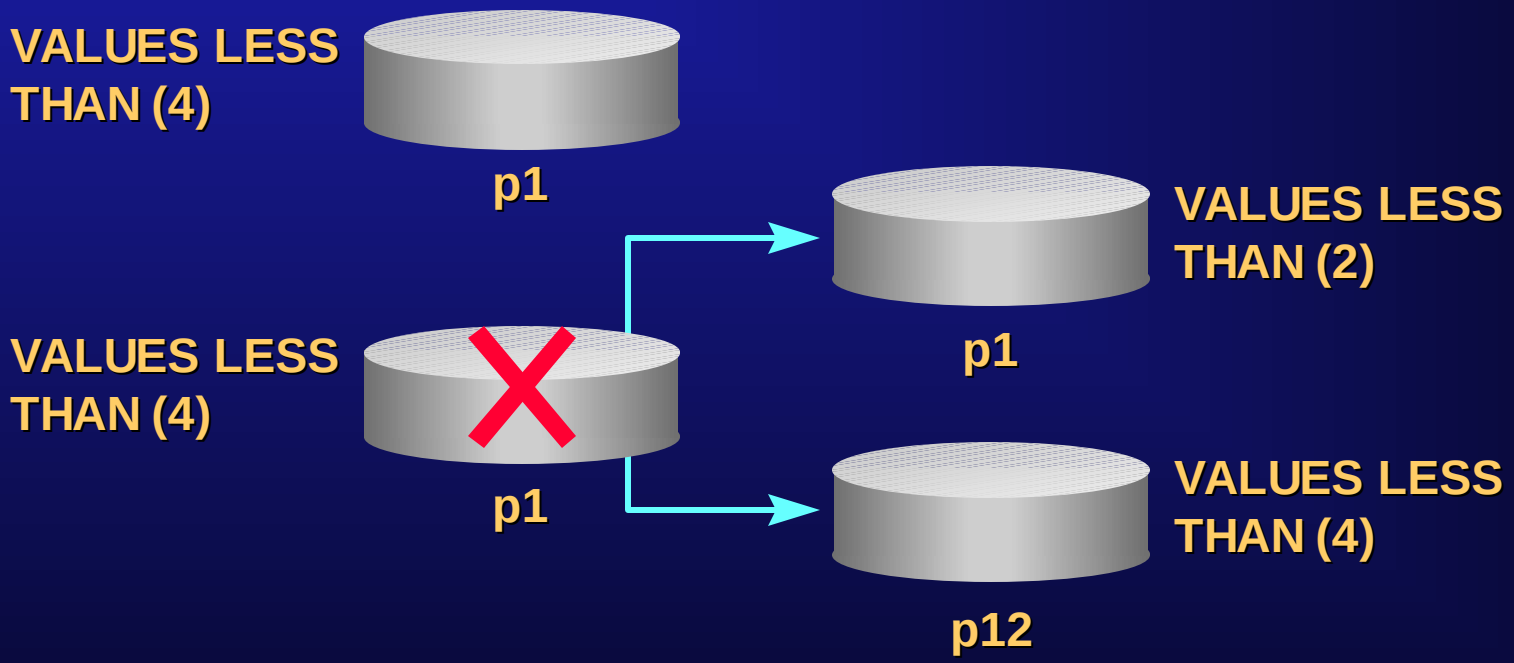
```
SQL> ALTER TABLE sales
  2>  ADD PARTITION p11 VALUES LESS THAN (44)
  3>  TABLESPACE data11;
```



Weeks 1-3 Weeks 36-39 Weeks 40-43

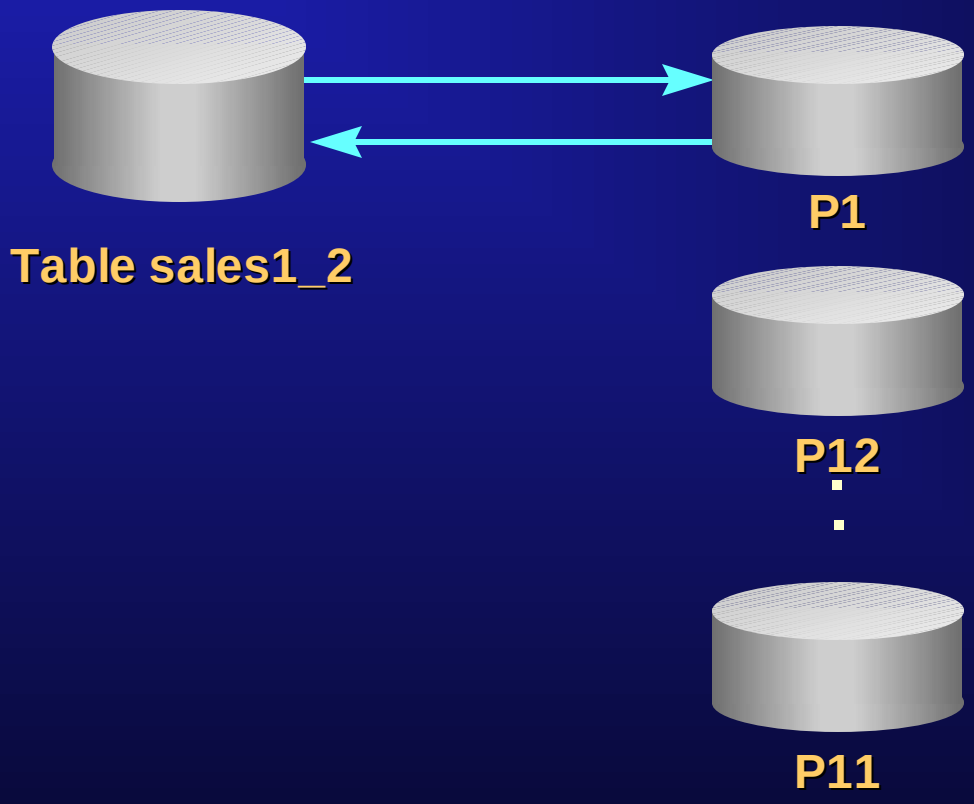
Splitting Partitions

```
SQL> ALTER TABLE sales SPLIT PARTITION p1 AT (2)
  2>  INTO (PARTITION p1 TABLESPACE data1,
  3>            PARTITION p12 TABLESPACE data12);
```



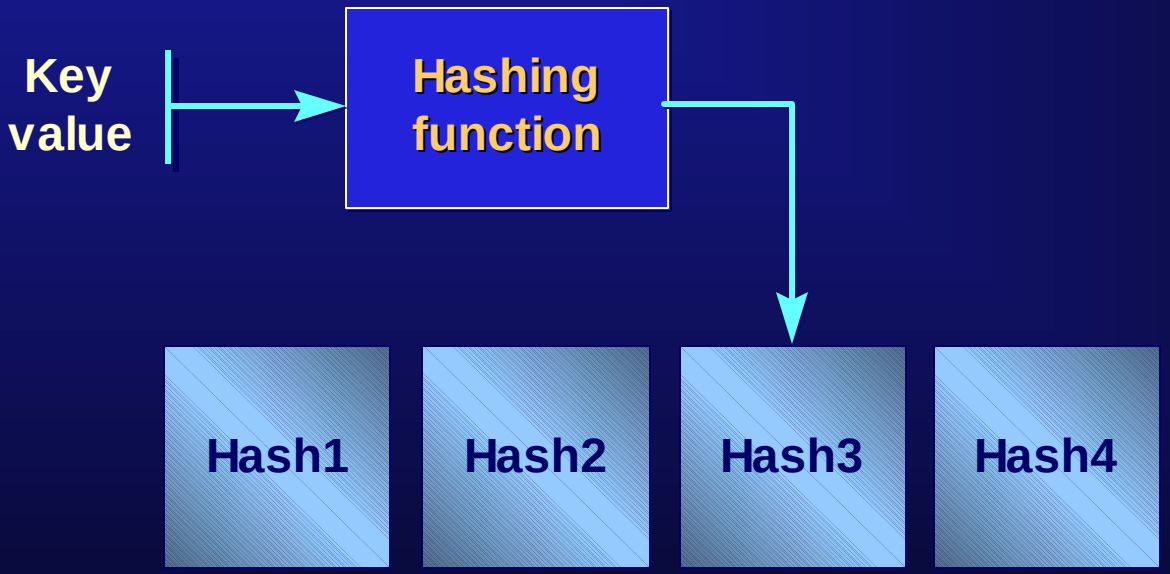
Exchanging a Partition and a Table

```
SQL>  ALTER TABLE sales EXCHANGE PARTITION p1
  2>  WITH TABLE sales1_2;
```



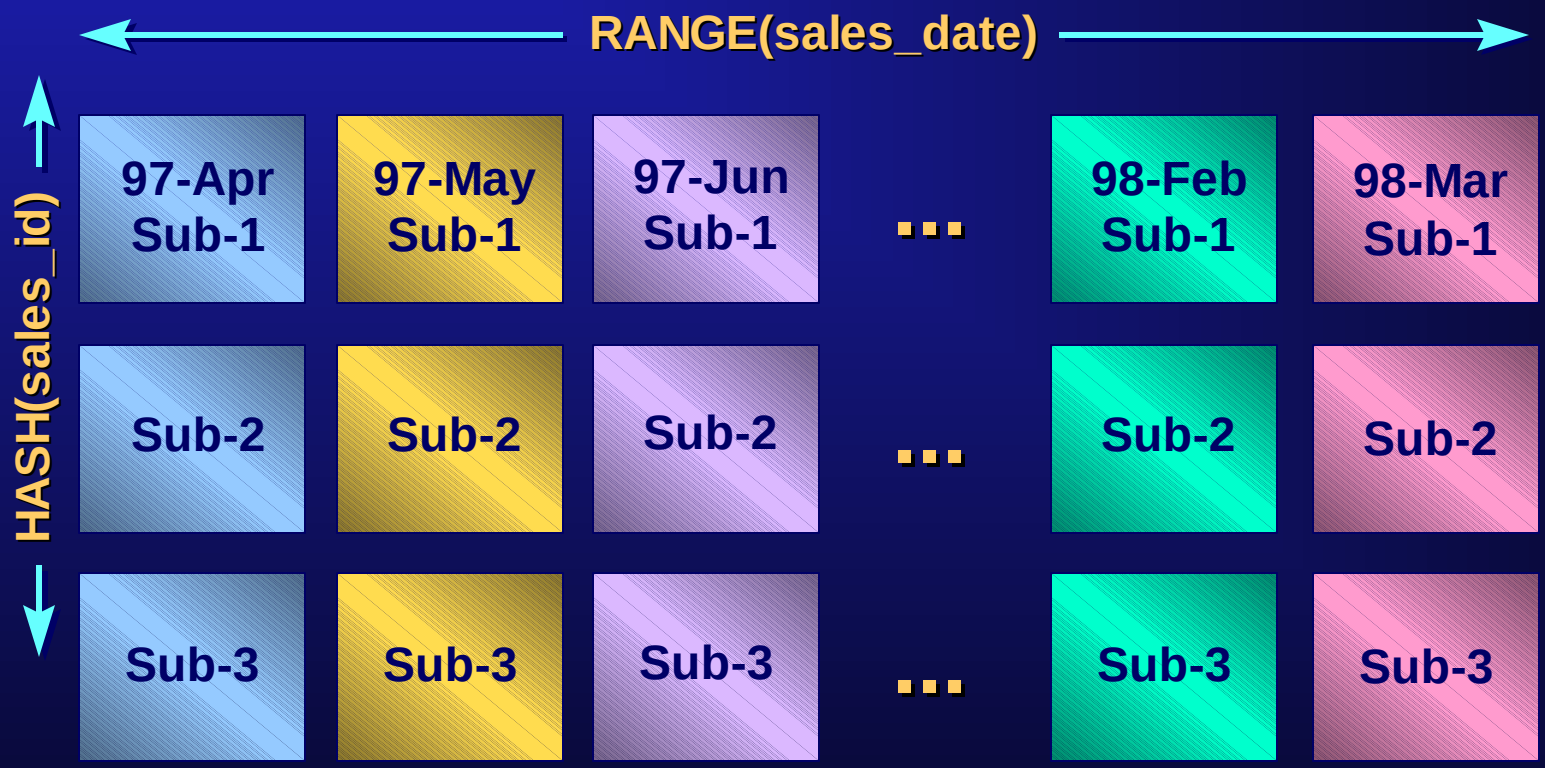
Hash Partitioning (8*i*)

- Inserts rows into partitions automatically based on hash of partition key
- Provides highly tunable data placement
- Easy-to-implement, simple syntax



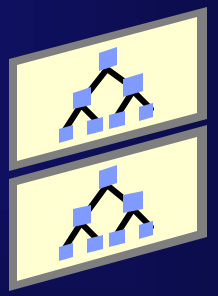
Composite Partitioning (8*i*): Partitioning by Range and Hash

```
CREATE TABLE … PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(sales_id) SUBPARTITIONS 3 …
```



Index Partitions

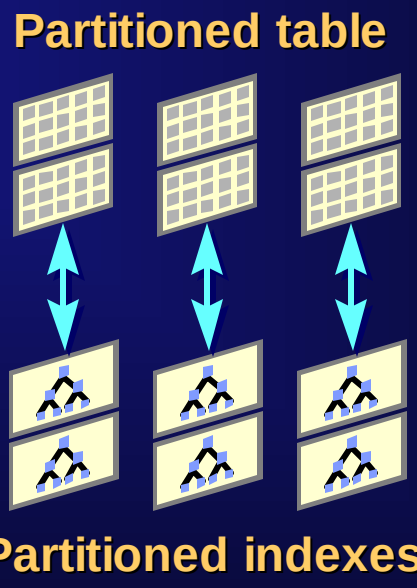
- Indexes can be either partitioned or nonpartitioned.
- Choice of indexing strategy allows greater flexibility to suit database and application requirements.



Partitioned index

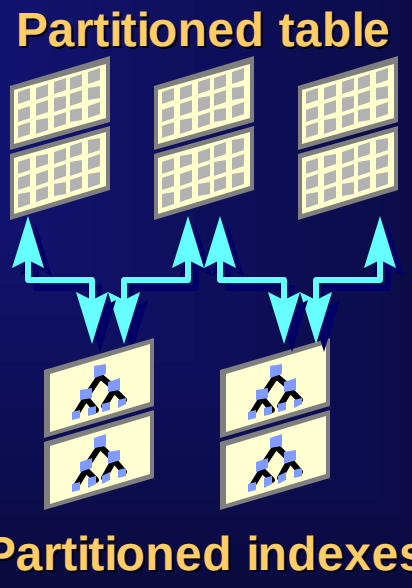
Index Partitioning Strategy: Equipartitions

- One index partition is created for each table partition.
- Indexes are maintained by the server.



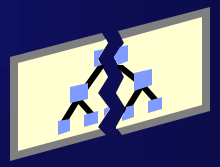
Index Partitioning Strategy: Global Partitions

- Multiple table partitions can be pointed to multiple indexes.
- Indexes are maintained manually.
- The GLOBAL keyword is used to create the partitions.



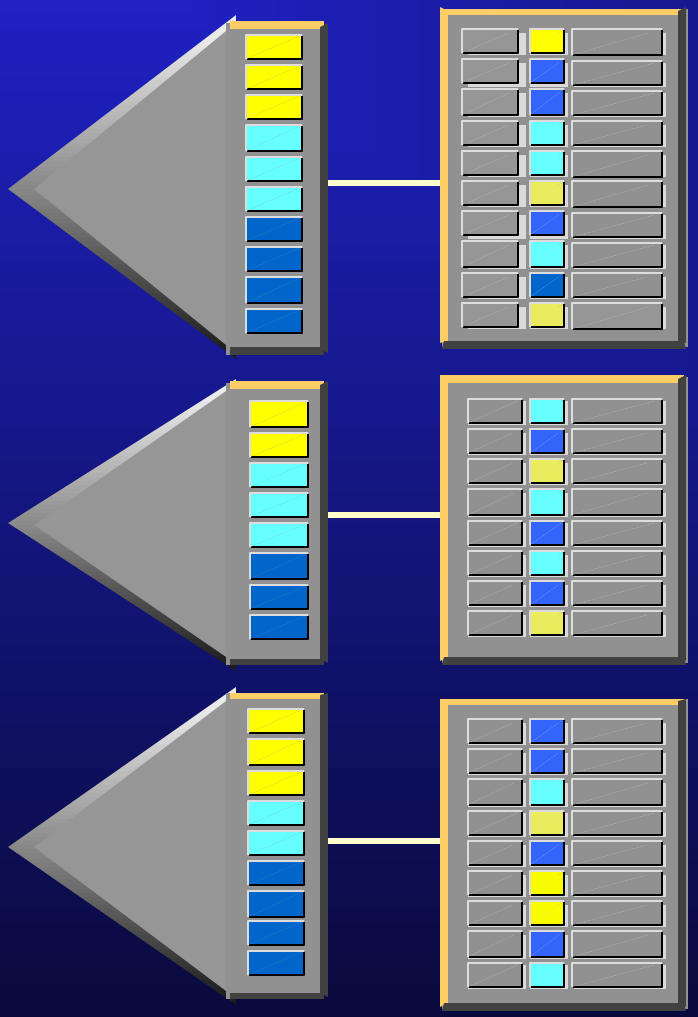
Unusable Indexes

- Nonpartitioned and partitioned indexes can be marked as index unusable (IU).
- Accessing an IU index or partition results in errors.
  - Rebuild IU partitions
  - Drop and re-create IU nonpartitioned indexes



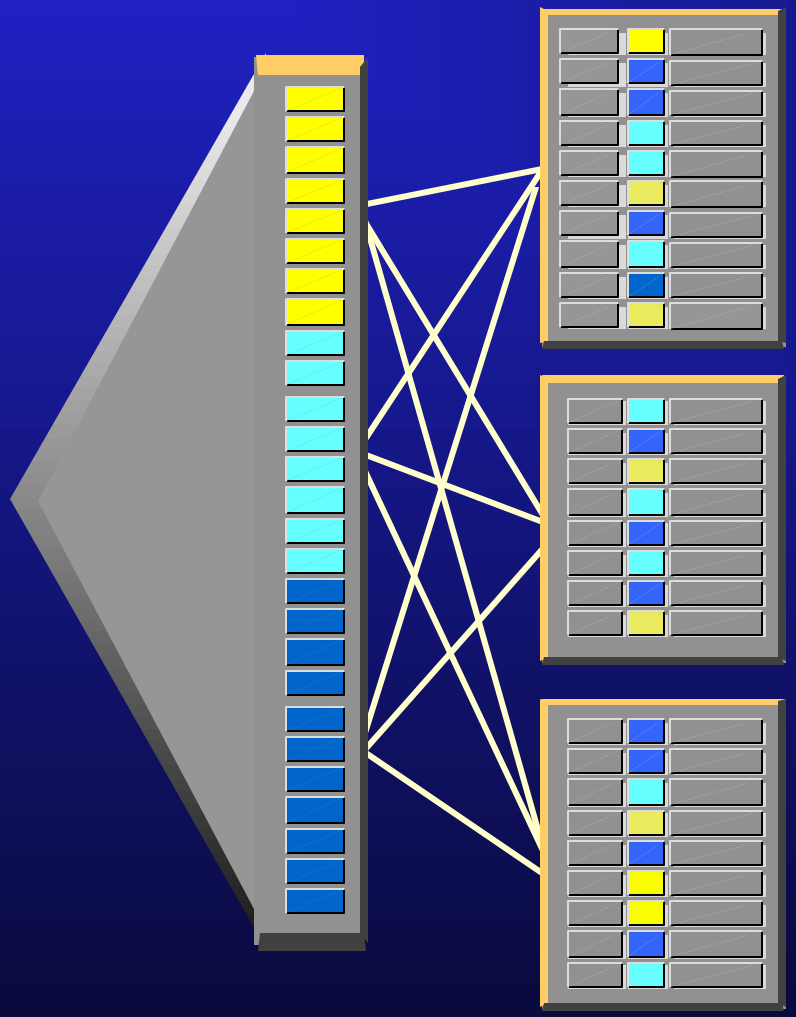
Index Unusable

Partitioning Indexes: Local Index



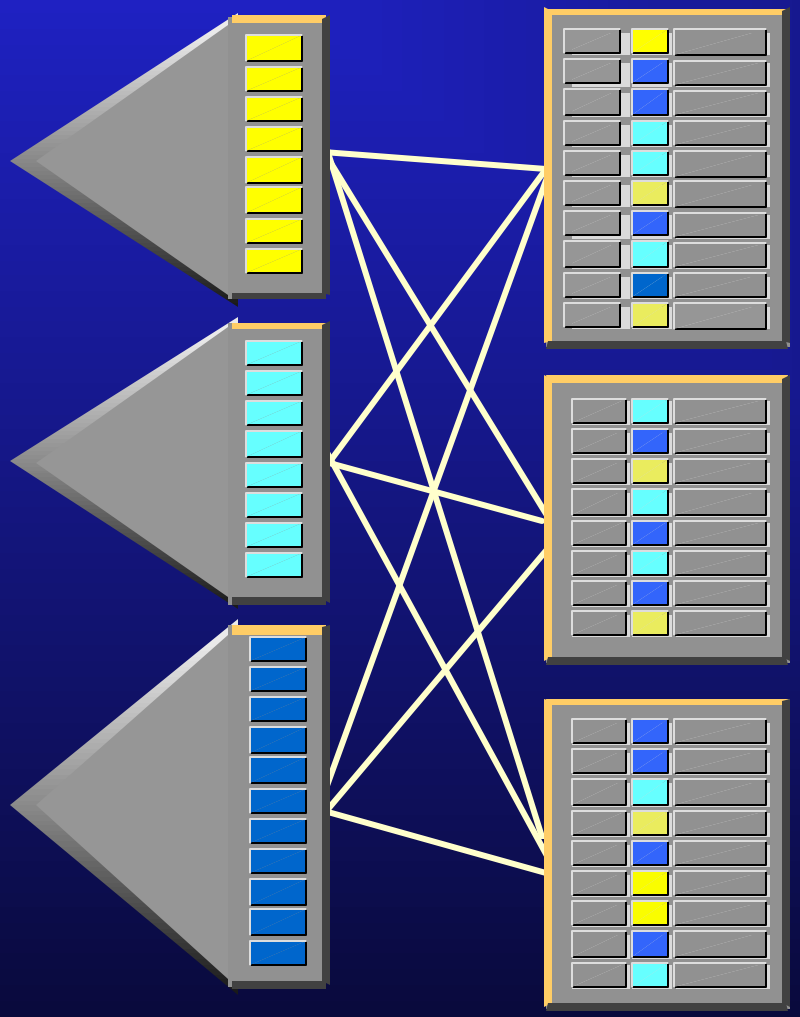
- Index (B-tree or bitmap) is partitioned along same boundaries as data
- Pros: Easy to manage; parallel index scans
- Cons: Less efficient for retrieving small amount of data
- Recommended for DW

Partitioning Indexes: Global Nonpartitioned Index



- One index B-tree structure that spans all partitions
- Pros: Efficient access to any individual record
- Cons: Lacks manageability aspects of partitioned index
- Recommended for OLTP

Partitioning Indexes: Global Partitioned Index



- Index partitioned independently of data
- Each index structure can reference any and all partitions
- Pros: Availability and manageability
- Cons: Less efficient for loading new data
- Recommended for OLTP

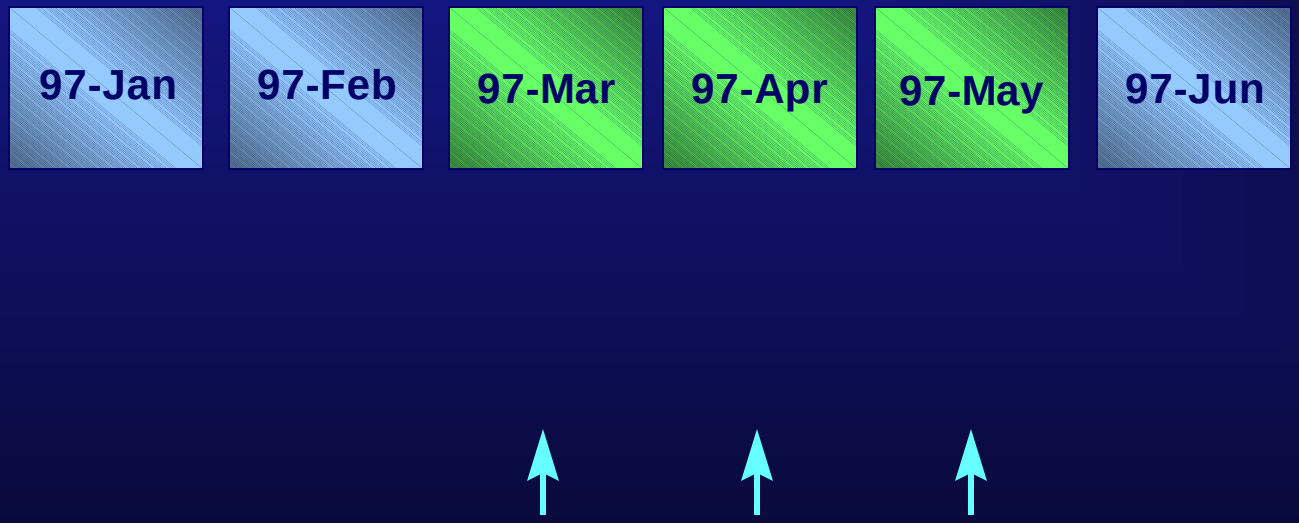
Gaining Performance with Partitions

- Eliminates data that is not needed for the current processing
- Parallelizes operations whenever applicable
- Reduces the amount of sorting required
- Balances I/O load across physical devices
- Maintains disk affinity control in Parallel Server environments

Performance: Partition Pruning

```
SELECT SUM(sales_amount) FROM sales
WHERE sales_date BETWEEN '01-MAR-97' AND '31-MAY-97';
```

Partition pruning: Only the relevant partitions are accessed

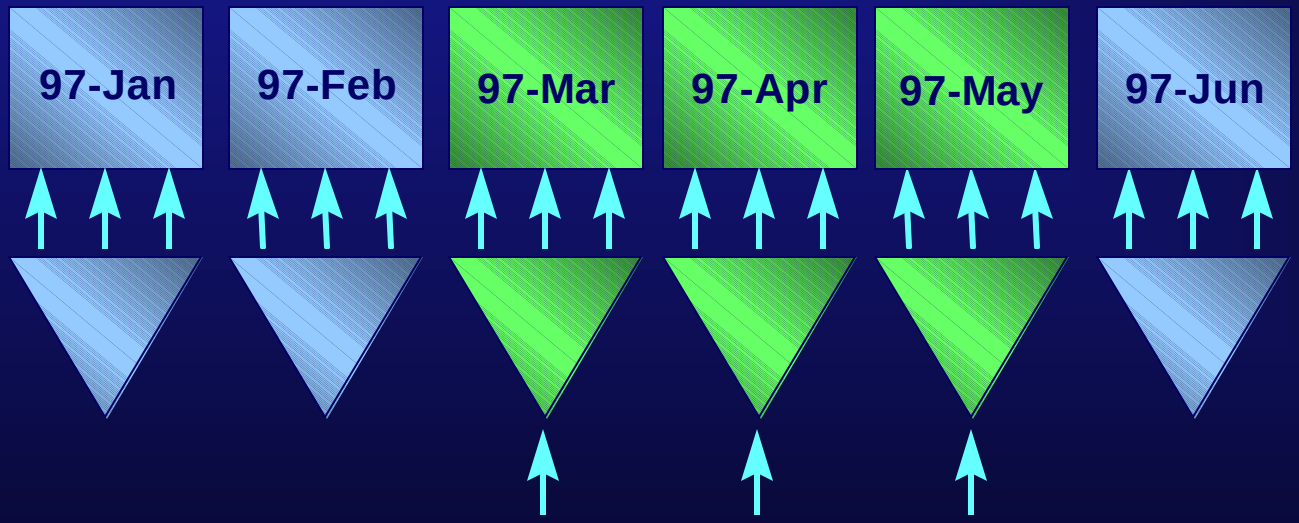


Performance: Partition Pruning

- Partition pruning works with all other techniques

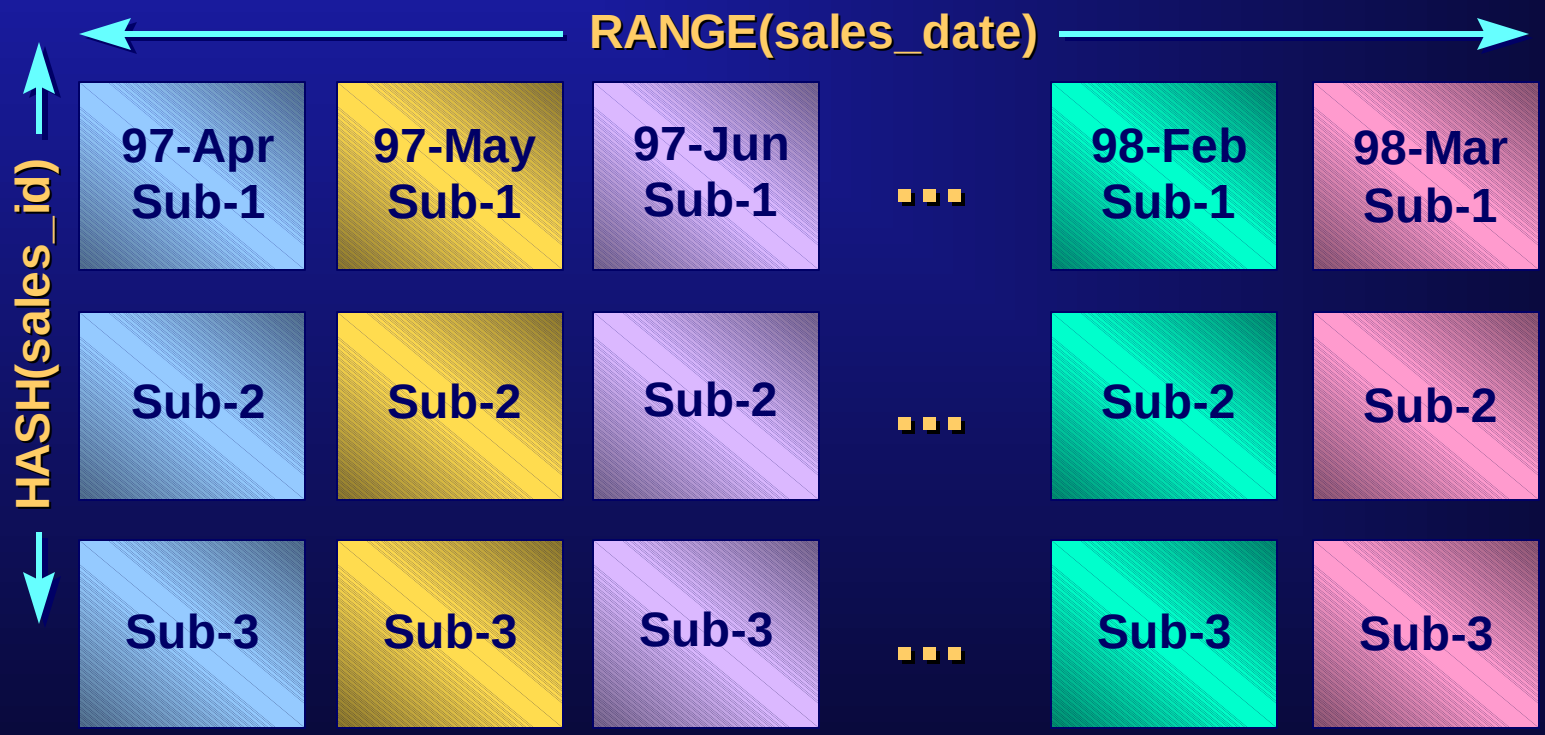
```
SELECT SUM(sales_amount) FROM sales
WHERE sales_date BETWEEN '01-MAR-97' AND '31-MAY-97'
AND sales_customer_city IN 'Los Angeles';
```

- Partition pruning combined with index access



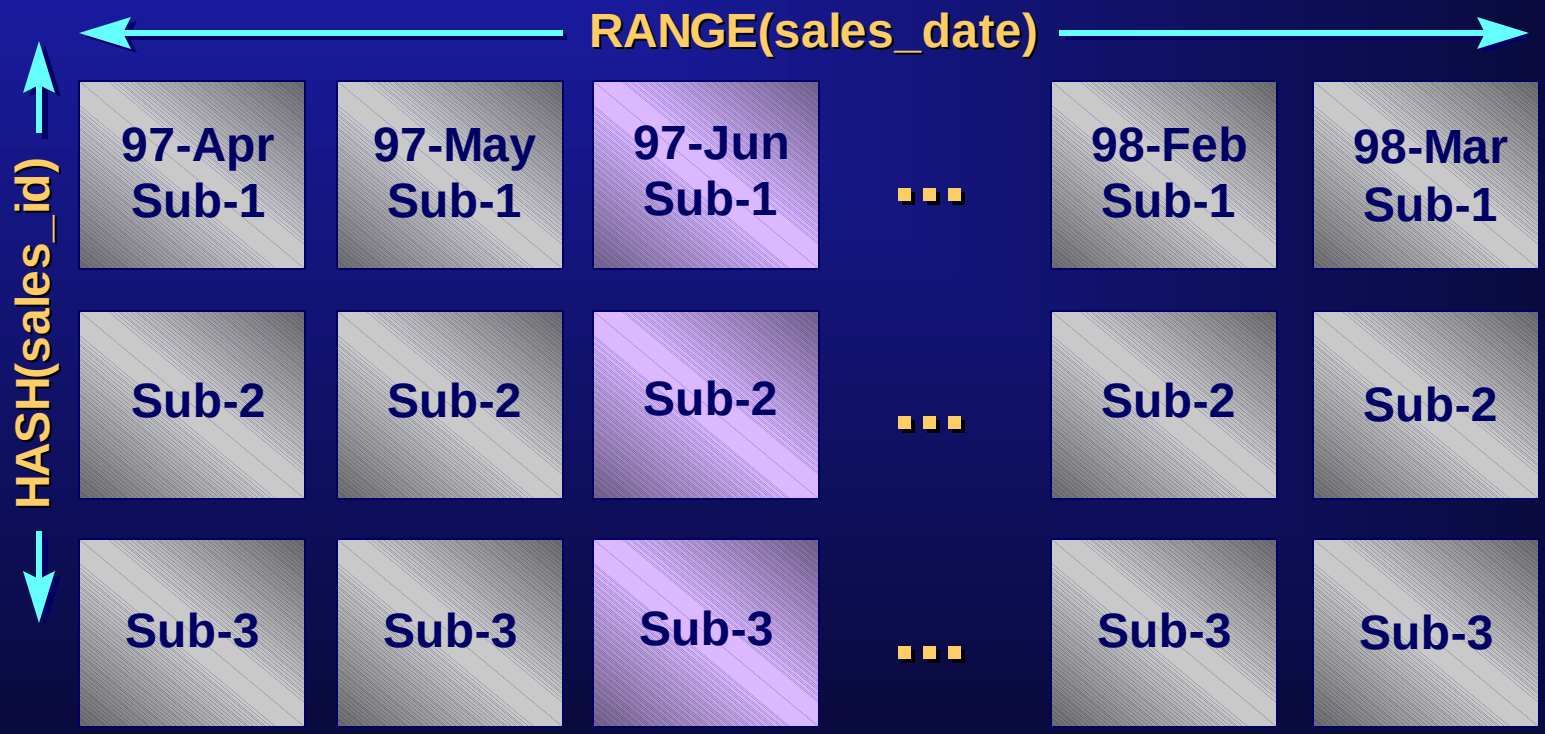
Composite Partitioning (8*i*)

```
CREATE TABLE …
PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(sales_id) SUBPARTITIONS 3
…
```



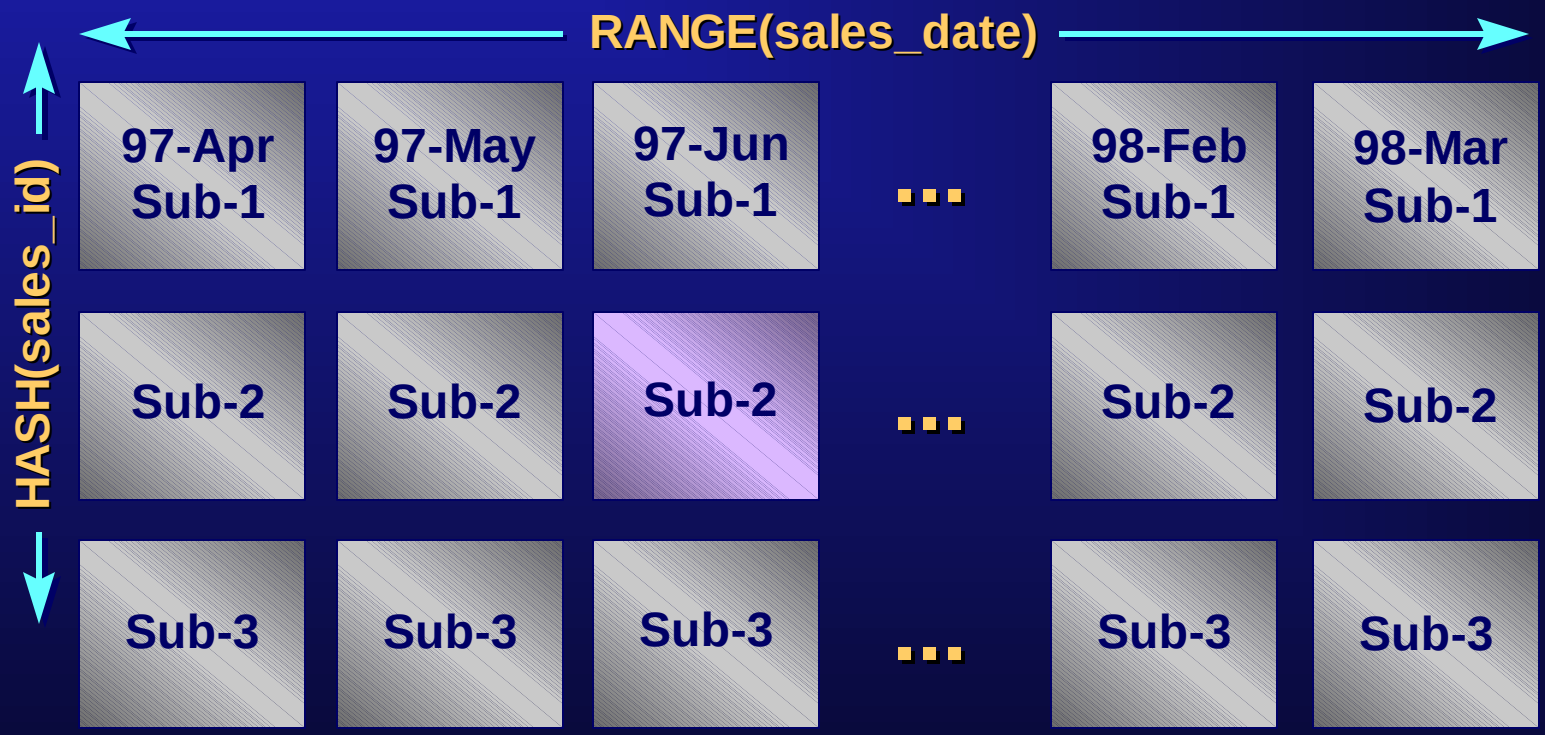
Composite Partitioning: Partition Pruning (8*i*)

```
SELECT … FROM sales
WHERE sales_date BETWEEN '05-JUN-97' AND '12-JUN-97';
```



Composite Partitioning: Partition Pruning (8*i*)

```
SELECT … FROM sales
WHERE sales_date BETWEEN '05-JUN-97' AND '12-JUN-97'
  AND sales_id = 45352;
```



Partitioning in a Data Warehouse

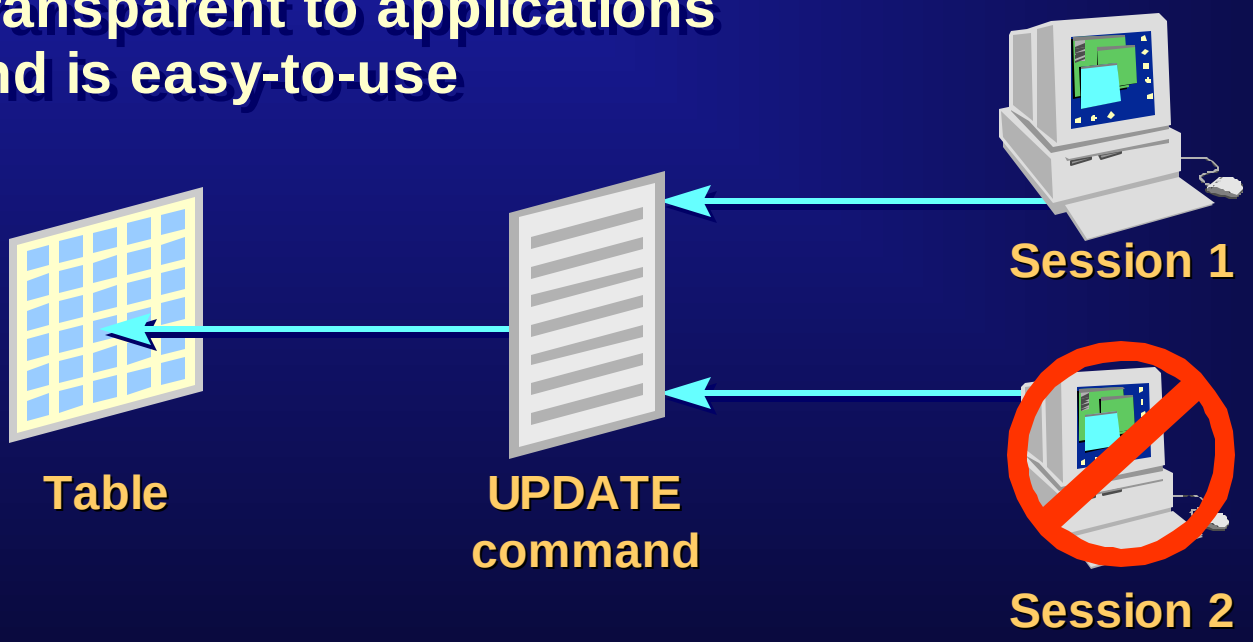
- The partitioning scheme should be designed for manageability, not query performance or parallelism.
- Oracle query operations (including parallelism) are enhanced by partitioning but are not dependent on partitioning.
- Query parallelism does not depend on partitioning.

Introducing Parallel DML

- Used for DML operations against large databases such as DSS and data warehouses
- Used for long-running DML jobs in an OLTP database while maintaining high data availability
- Complements the existing parallel query architecture

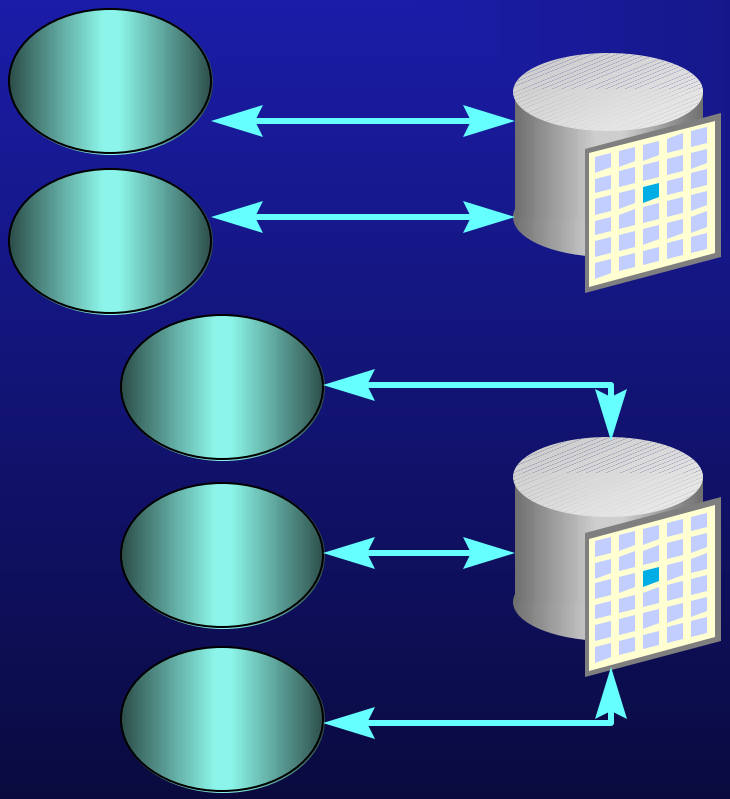
Advantages of PDML

- Can dramatically improve performance, especially for data warehouses
- Does not need multiple sessions
- Transparent to applications and is easy-to-use



Parallelization by ROWID Ranges

Used for parallel queries only



ROWID ranges

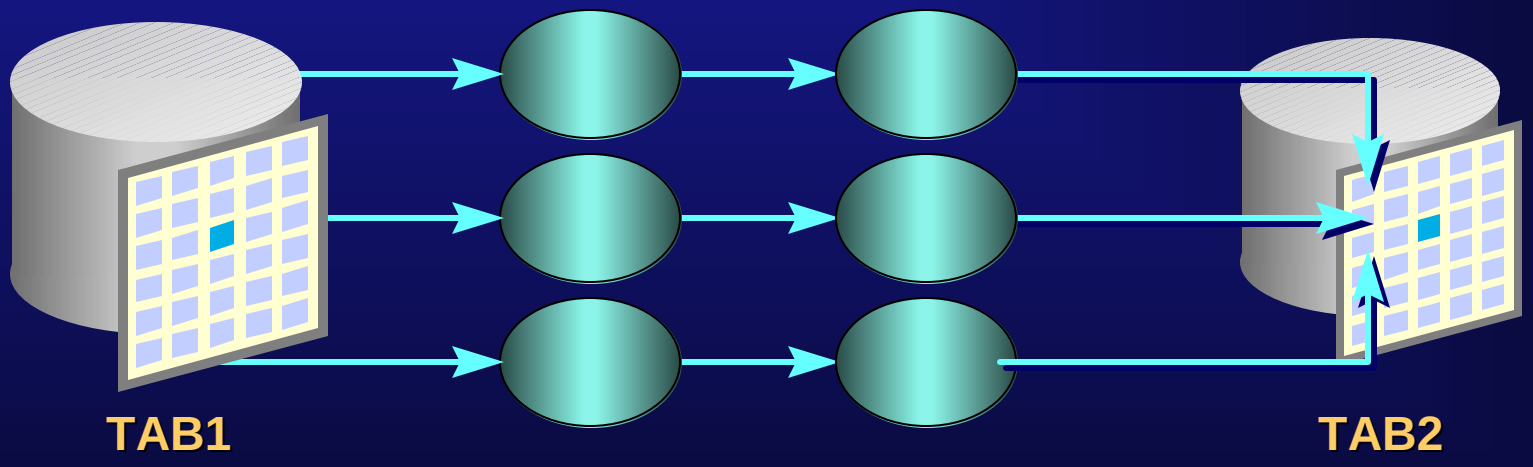
AAAAT9AABAAABaLAAY
AAAAT9AABAAABalAAv
...

FFAAT9AABAAABaLAAY
FFAAT9AABAAABalAAv
...

Parallelization by Slave Processes

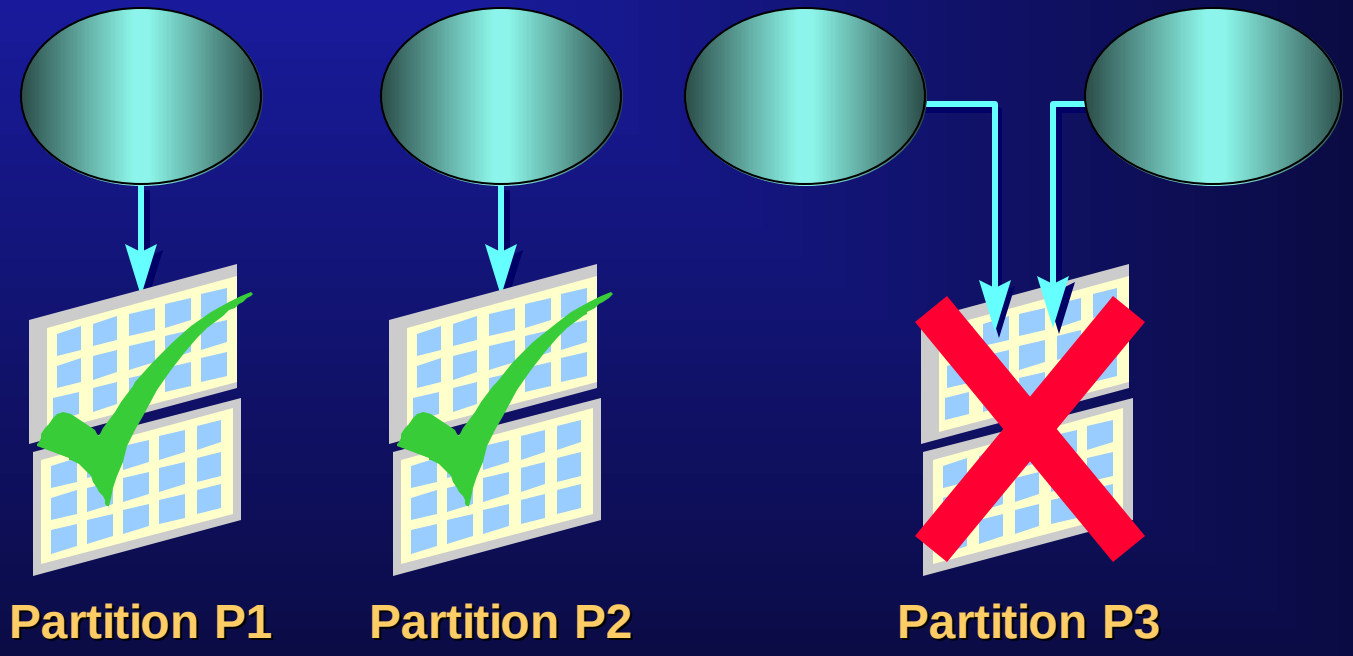
Used for parallel INSERT . . . SELECT nonpartitioned tables

```
SQL> INSERT /*+PARALLEL(tab1,3)*/ INTO tab2
  2>   SELECT * FROM tab1;
```



Parallelization by Partitions

Used for parallel inserts, updates, and deletes on partitioned tables



Establishing Degree of Parallelism

- Instance parameters in the `init.ora` file
- PARALLEL clause for the table
- Explicit hint in the statement

```
SQL> CREATE TABLE docindex
  2>   ( token CHAR(20),
  3>     doc_oid NUMBER,
  4>     token_frequency NUMBER,
  5>     token_occurrence_data VARCHAR2(512) )
  6>   PARALLEL (DEGREE 4);
Statement processed
```

Manipulating Data with Parallel Hints

- Specified after INSERT, UPDATE, or DELETE keyword
- Explicit hints override parallel clause settings

```
SQL> ALTER SESSION ENABLE PARALLEL DML;
SQL> UPDATE /*+PARALLEL(emp,5)*/ emp
  2>   SET sal = sal * 1.1
  3>   WHERE job = 'CLERK' and
  4>     deptno IN (SELECT deptno FROM dept
  5>     WHERE loc = 'DALLAS');
SQL> COMMIT;
```

Session Considerations

- You execute ALTER SESSION only between transactions.
- You can commit or roll back only after a PDML statement.
- All DML portions of statements are considered for parallel execution.
- Parallelism of a SELECT statement is not influenced.
- PDML can prevent “skewed” indexes.

Data Dictionary

- {USER|ALL|DBA}_PART_TABLES
- {USER|ALL|DBA}_PART_INDEXES
- {USER|ALL|DBA}_PART_KEY_COLUMNS
- {USER|ALL|DBA}_TAB_PARTITIONS
- {USER|ALL|DBA}_IND_PARTITIONS
- {USER|ALL|DBA}_PART_COL_STATISTICS
- {USER|ALL|DBA}_TAB_COL_STATISTICS
- {USER|ALL|DBA}_PART_HISTOGRAMS
- {USER|ALL|DBA}_TAB_HISTOGRAMS
- {USER|ALL|DBA}_OBJECTS
- {USER|ALL|DBA}_TABLES
- {USER|ALL|DBA}_INDEXES
- {USER|ALL|DBA}_TAB_COLUMNS

New Columns of the PLAN Table

- There are three new columns of the PLAN table:
  - PARTITION_START
  - PARTITION_STOP
  - PARTITION_ID
- There is also a new type of step:
  - PARTITION
- In addition, there are enhancements to TABLE ACCESS and INDEX steps when such steps refer to partitioned objects.

PARTITION_START and PARTITION_STOP

The PARTITION_START and PARTITION_STOP columns describe the start and stop partition of a range of accessed partitions. They can take these values:

- NUMBER(*n*)
- KEY
- ROW LOCATION
- INVALID

PARTITION_ID

The PARTITION_ID column identifies the step that has computed a pair of values of the PARTITION_START and PARTITION_STOP columns.

OPTIONS Column of PARTITION Step

The OPTIONS column of a PARTITION step can take these values:

- CONCATENATED
- SINGLE
- EMPTY

OPTIONS Column of TABLE ACCESS Step

The OPTIONS column of a TABLE ACCESS step describing access by ROWID to a table can contain the following values:

- BY USER ROWID
- BY INDEX ROWID
- BY GLOBAL INDEX ROWID
- BY LOCAL INDEX ROWID

Example: Explain Plan

```
SQL> EXPLAIN PLAN FOR SELECT * FROM sales
  2>   where week_no between 38 and :a;
```

```
SQL> SELECT id, operation, options, object_name,
  2>  partition_start as "START",
  4>  partition_stop as "STOP",
  3>  partition_id as pid  from plan_table;
```

```
 ID OPERATION         OPTIONS      OBJECT_NAME START    STOP    PID
--- ----------------- ------------ ----------- -------- ------- ---
  0 SELECT STATEMENT
  1 PARTITION         CONCATENATED             NUMBER(4) KEY      1
  2 TABLE ACCESS      FULL         SALES       NUMBER(4) KEY      1
```

Partition Maintenance Operations

Table Partition Maintenance Operations

```
ALTER TABLE ADD PARTITION
ALTER TABLE DROP PARTITION
ALTER TABLE EXCHANGE PARTITION
ALTER TABLE MODIFY PARTITION
ALTER TABLE MOVE PARTITION [PARALLEL]
ALTER TABLE RENAME PARTITION
ALTER TABLE SPLIT PARTITION [PARALLEL]
ALTER TABLE MERGE PARTITION [PARALLEL]
ALTER TABLE COALESCE PARTITION [PARALLEL]
ALTER TABLE ANALYZE PARTITION
ALTER TABLE TRUNCATE PARTITION
Export/Import [by partition]
SQL*Loader (direct and parallel)[by
    partition]
Parallel Table ANALYZE package
```

Unaffected partitions remain available to applications

Index Maintenance Operations

```
ALTER INDEX MODIFY PARTITION
ALTER INDEX DROP PARTITION
ALTER INDEX REBUILD PARTITION
ALTER INDEX RENAME PARTITION
ALTER INDEX RENAME
ALTER INDEX SPLIT PARTITION
ALTER INDEX ANALYZE PARTITION
```

Summary

- You can better manage data by creating partitioned tables and indexes.
- You can use ALTER TABLE . . . EXCHANGE PARTITION to move from Oracle7 partitioned views to Oracle8*i* partitioned tables.
- Parallel DML operations can speed up DML operations against large database objects.
- EXPLAIN PLAN has been enhanced to output information about the use of partitions.
- SQL*Loader, Export/Import, and Analyze have been enhanced to support table partitions.

Summary: Oracle8*i* Enhancements

- New partitioning methods:
  - Hash
  - Composite (range partition + hash subpartition)
- New data type support:
  - Object types
  - LOB columns (including partitioning of LOB spaces)
  - Index-organized tables (range only)

Summary: Oracle8*i* Enhancements

- Management tools:
  - MERGE partition
  - COALESCE partition (for hash)
  - Selectable behavior for update of partition key
- Performance optimizations:
  - Partitionwise join
  - Partition elimination optimizations, including disjunction (ORs) in WHERE clauses