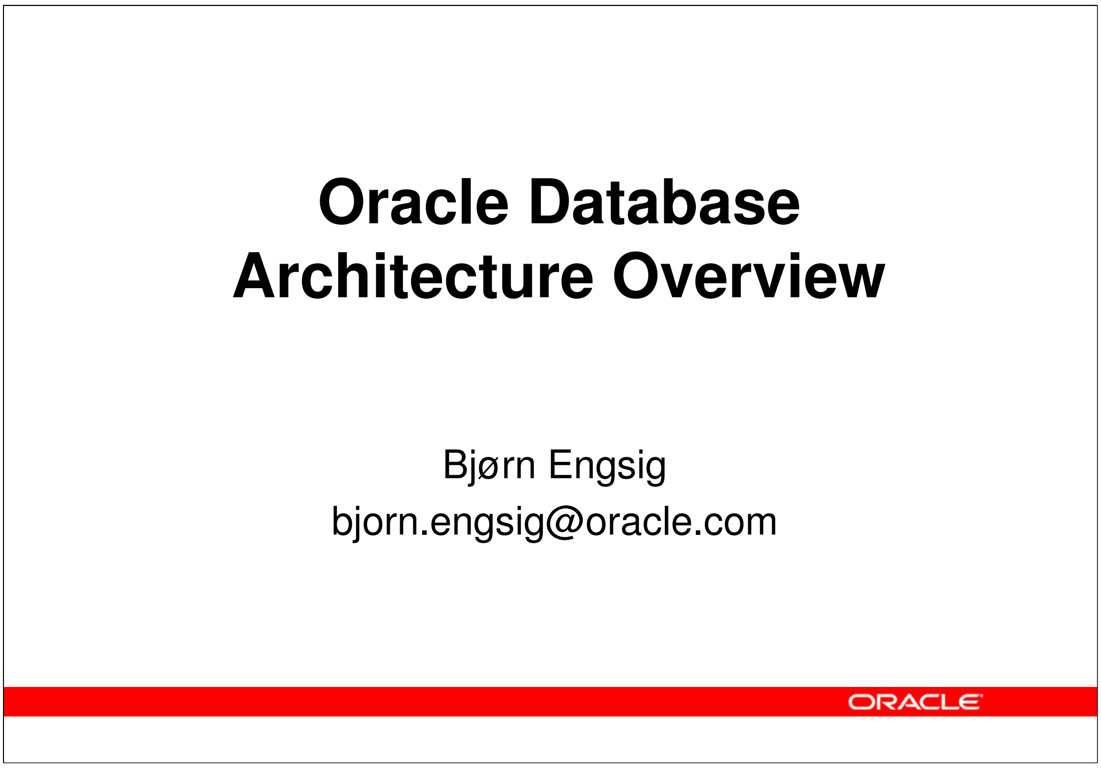


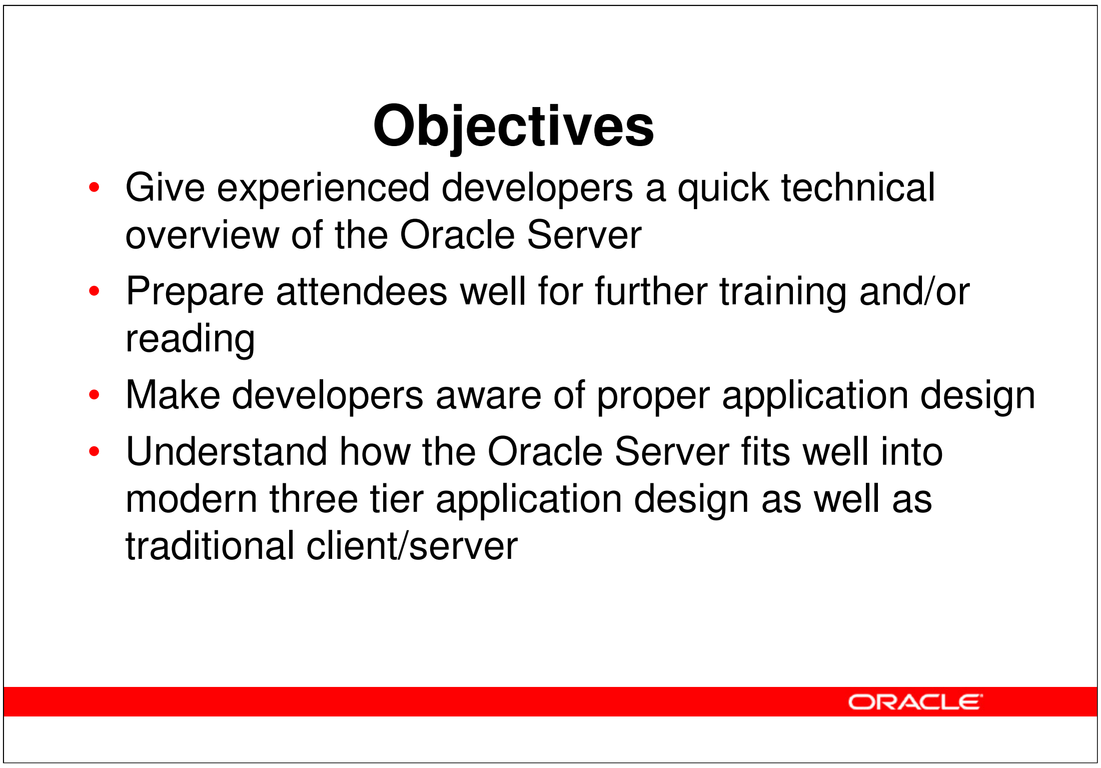
Oracle Database Architecture Overview

Bjørn Engsig
bjorn.engsig@oracle.com

This one day course is intended to give people with good technical background and preferably some knowledge of Oracle an introduction to the architecture of Oracle.

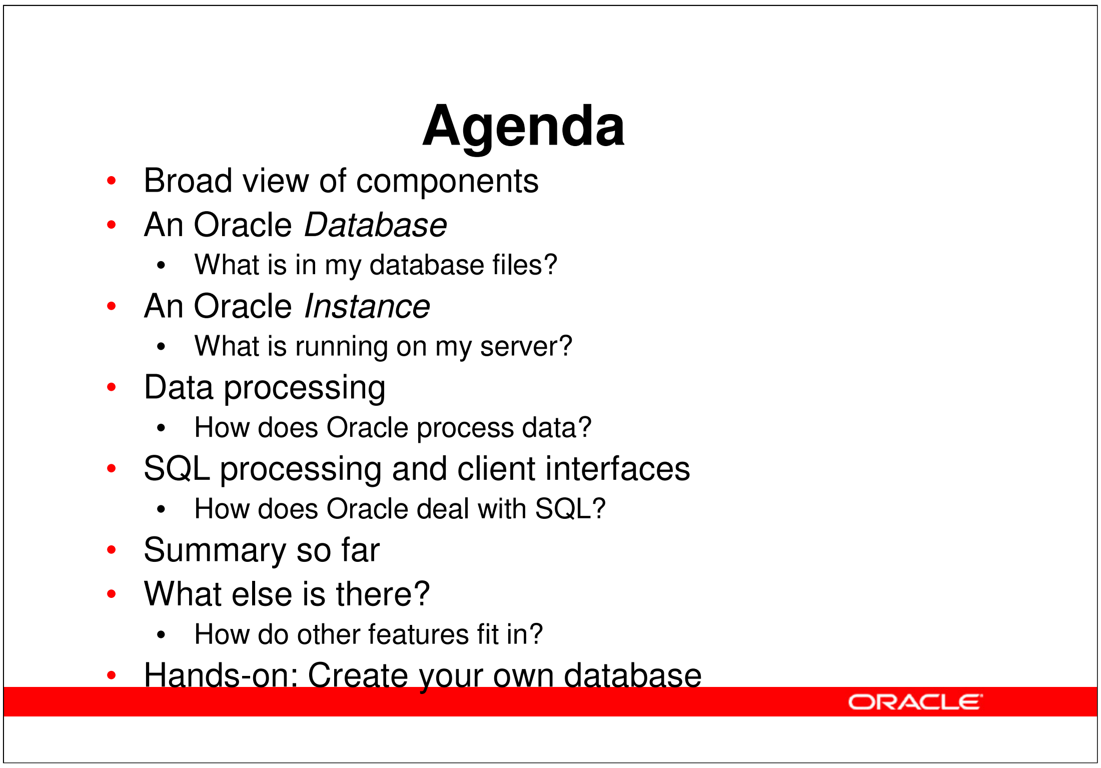
The following topics are covered:

- •Oracle components
    - •The database
    - •The instance
- •Oracledata processing
    - •Queries
    - •Data manipulation
    - •Read consistency
- •Oracle SQL processing
- •Hands on
    - •Create your own database
    - •Start/stop the instance



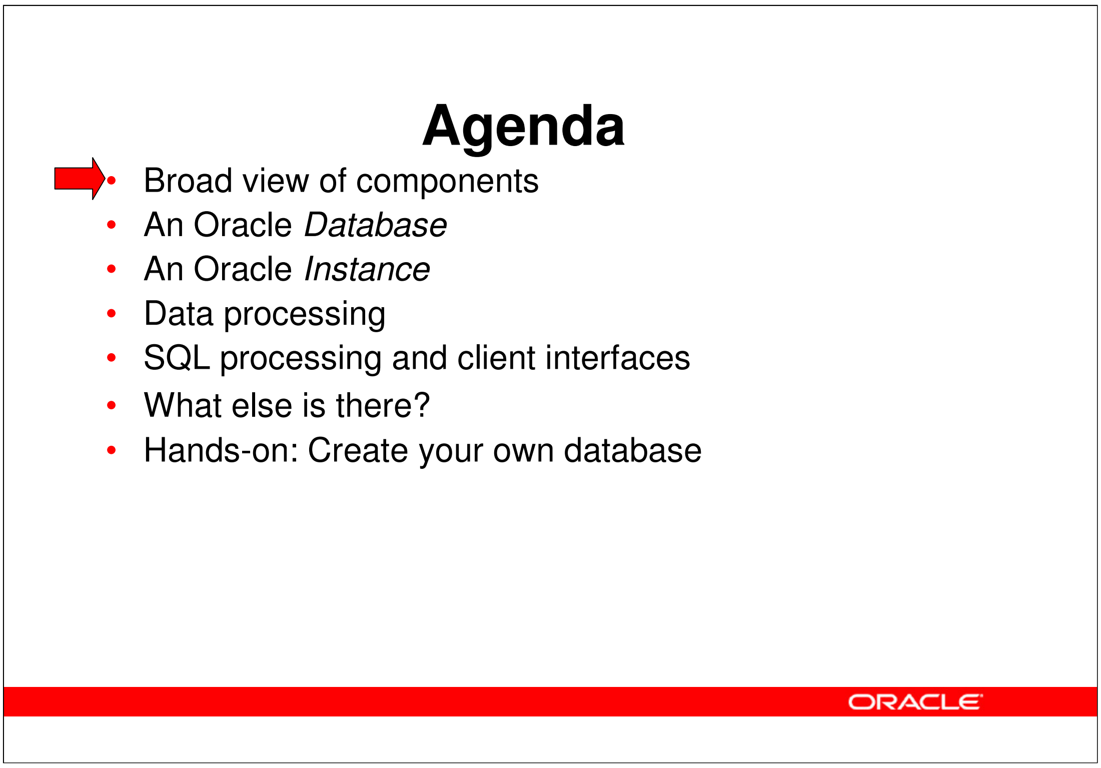
Objectives

- Give experienced developers a quick technical overview of the Oracle Server
- Prepare attendees well for further training and/or reading
- Make developers aware of proper application design
- Understand how the Oracle Server fits well into modern three tier application design as well as traditional client/server



Agenda

- Broad view of components
- An Oracle *Database*
  - What is in my database files?
- An Oracle *Instance*
  - What is running on my server?
- Data processing
  - How does Oracle process data?
- SQL processing and client interfaces
  - How does Oracle deal with SQL?
- Summary so far
- What else is there?
  - How do other features fit in?
- Hands-on: Create your own database

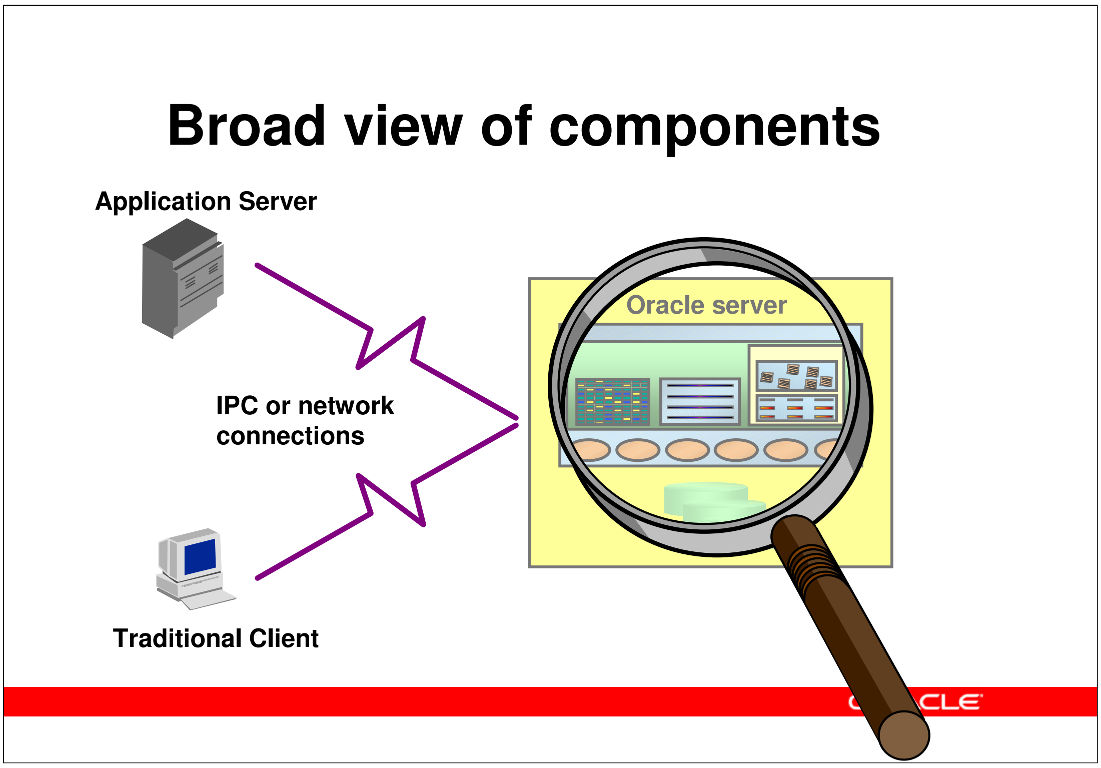


Agenda

- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
- Data processing
- SQL processing and client interfaces
- What else is there?
- Hands-on: Create your own database

ORACLE

Initially, we will take a broad view of the components with Oracle as a black box and with some connections to it.



Broad view of components

Application Server

IPC or network connections

Traditional Client

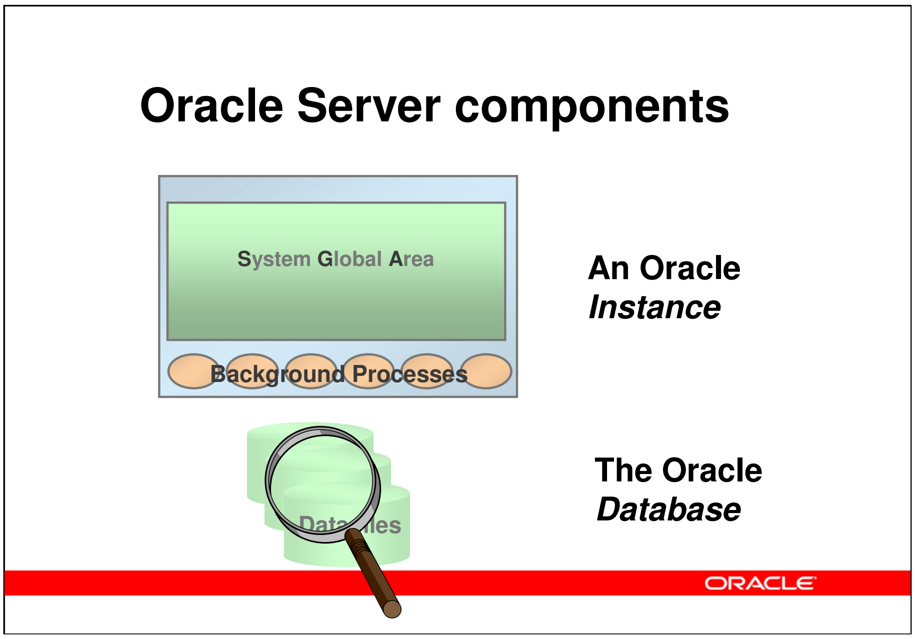
Oracle server

The broad view of an Oracle Server and the related components show:

•The Oracle Server itself, which is were the database resides. In this picture, the server is simply considered a “black box”, which most of the rest of this presentation looks inside.

•Systems connected to the Oracle Server. This can be either traditional clients, directly connected to the server, or it can be Application Servers, which run some application code, and to which clients connect (not shown). In this entire presentation, the term client is used throughout, as the behaviour of a traditional clients and an application server is the same, when seen from the database server.,

•Connections, which are some mechanism by which different processes can communicate. In practice, this is either physical network connections, often using TCP/IP, or it can be some inter process communication, e.g. using Unix pipes, when the client and the database server are actually executing on the same computer. In this presentation, there is no explicit distinction between a connection over a physical network, and one using inter process communication.



Oracle Server components

An Oracle *Instance*

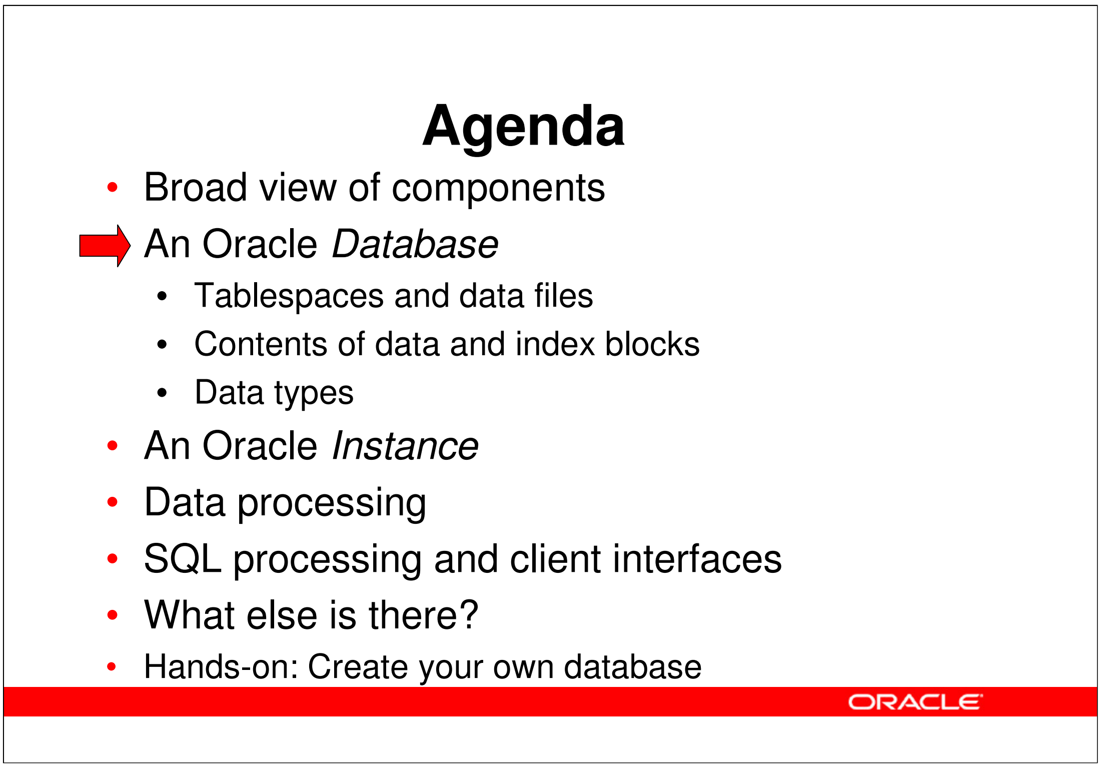
The Oracle *Database*

Looking into the Oracle server, two major components are seen:

•The Oracle *Database*, which is the set of operating system files, where the application's actual data are stored (such as customers, orders and order lines), together with Oracle's own internal information (such as information about users registered in the database, or tables found in the database). The Oracle Database as such, cannot be directly accessed in any way (except for backup) by clients. In contrast to other systems, there is no correlation between the concept of a user, an application or a file system and an Oracle database; rather, there is often only one database on each server, and you can even have a single database spanning multiple servers. This is known as Oracle Real Application Clusters.

•The Oracle *Instance*, which is a set of memory and process structures, running on a specific computer. This is the point of access for clients, and the instance is responsible for translating the SQL calls given by the client, to acual data transfers to and from the database stored in the operating system files. An Oracle instance is normally associated with an Oracle database, and those two together make up the Oracle Server. However, an Oracle instance may exist without being associated with an Oracle database; this is e.g. the case before the database is created or as an intermediate step during the startup of the Oracle server. An instance cannot be associated with more than one database, but one database (on a set of physical disks), can be associated with multiple instances (each on a separate computer), if the actual hardware configuration allow multiple computers to concurrently access a single set of disks.

•Inside the instance, two important parts are seen: The *System Global Area* (or *SGA*), which is a shared memory segment, typically of a size, which is roughly half of the physical RAM available on the computer. There is no direct access from clients to the SGA. Additionally, there is a set of *background processes*, which are working indirectly on behalf of the clients to do various specialized tasks, such as clean-up. These background processes are directly attached to the SGA, and can directly read and write to the disk files making up the database.

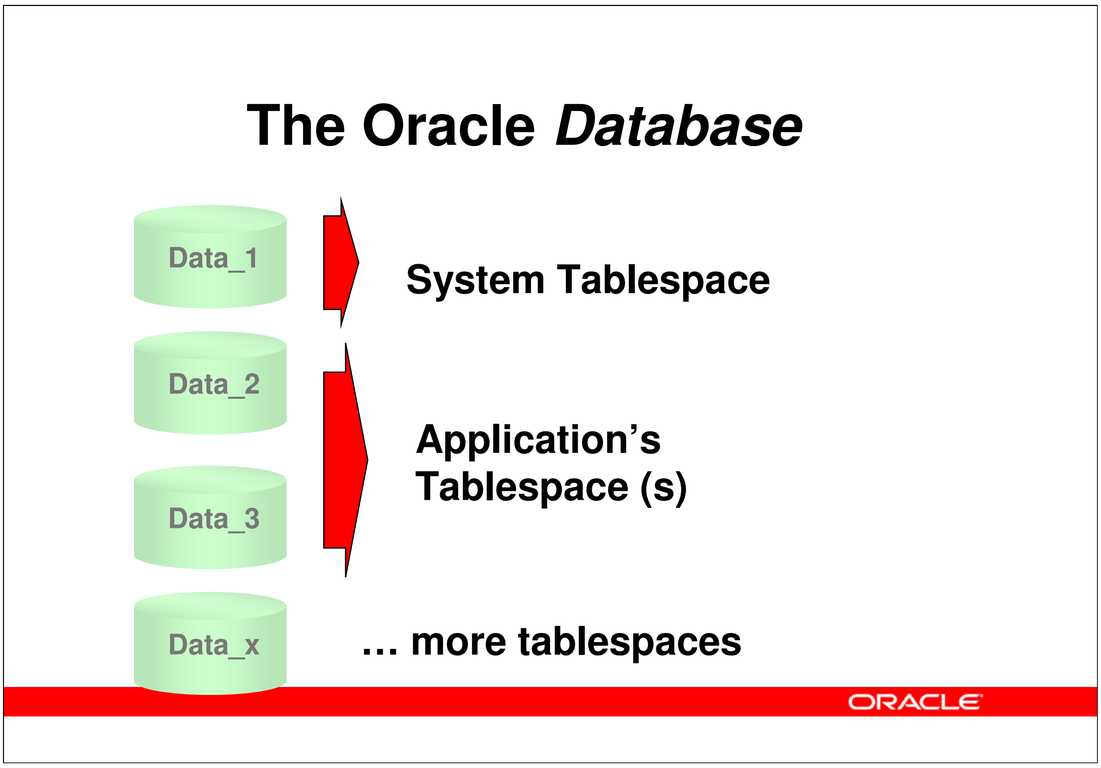


Agenda

- Broad view of components
- ➡ An Oracle *Database*
  - Tablespaces and data files
  - Contents of data and index blocks
  - Data types
- An Oracle *Instance*
- Data processing
- SQL processing and client interfaces
- What else is there?
- Hands-on: Create your own database

ORACLE

The next part takes us into the black box, to the part where your real database is stored. The database is really made up of a number of files.

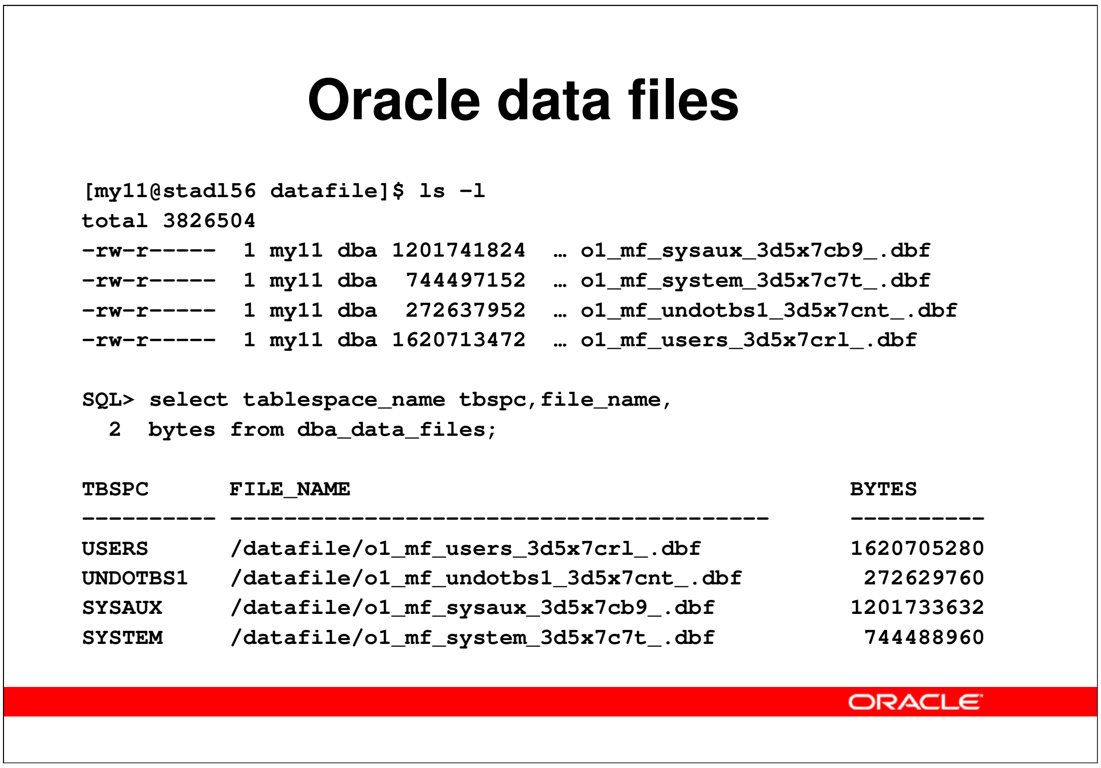


The Oracle *Database*

•All data of the database are stored in Tablespaces, which can be viewed as logical disks. Each tablespace is made of one or more physical disks, and can have sizes from megabytes to terabytes.

•Oracle uses a set of standard tablespaces, including the system tablespace, which is where Oracle stores its own internal information about e.g. users, tables and columns. Additionally, there are standard tablespaces used for e.g. temporary storage, etc. Only the tablespace with the actual name, *system*, is found right after database creation, other tablespaces, including some system related ones, are created subseequently.

•Each application (or set of related applications) define one or more tablespaces for storing that application's data. The names of these can be chosen freely, and can e.g. reflect application type such as manufactoring, finance, or you can have tablespace names reflecting use such as appl_data, and appl_index.

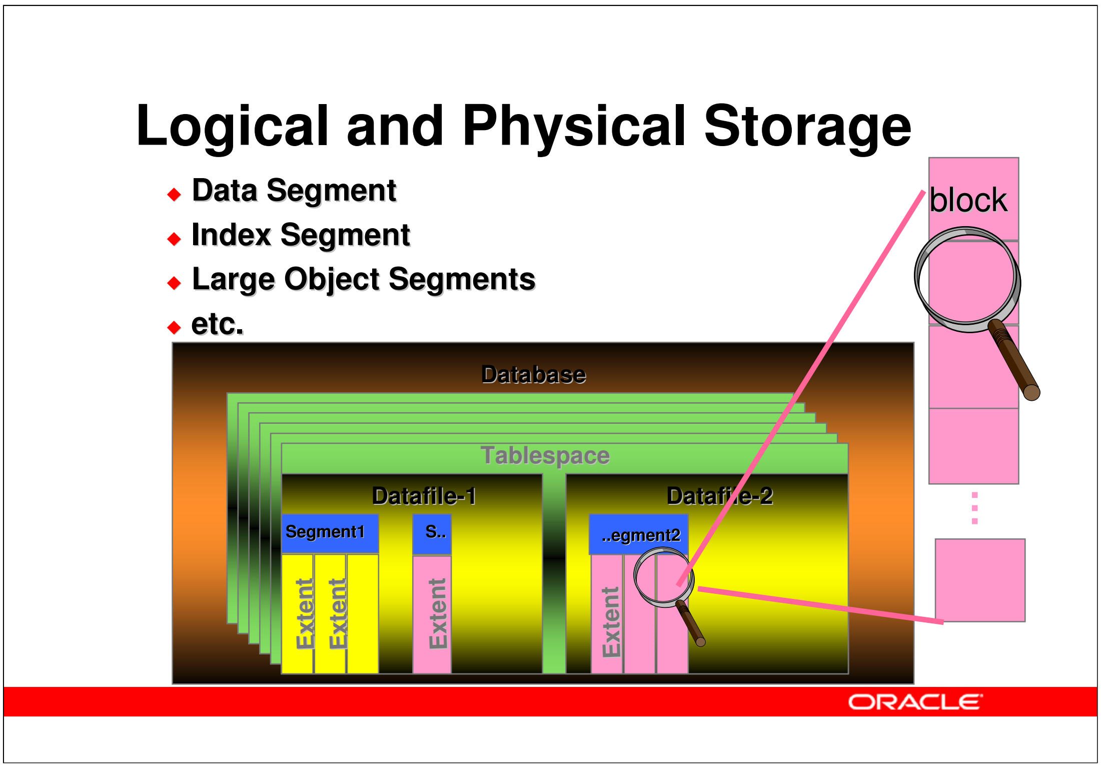


Oracle data files

```
[my11@stadl56 datafile]$ ls -l
total 3826504
-rw-r-----  1 my11 dba 1201741824  … o1_mf_sysaux_3d5x7cb9_.dbf
-rw-r-----  1 my11 dba  744497152  … o1_mf_system_3d5x7c7t_.dbf
-rw-r-----  1 my11 dba  272637952  … o1_mf_undotbs1_3d5x7cnt_.dbf
-rw-r-----  1 my11 dba 1620713472  … o1_mf_users_3d5x7crl_.dbf

SQL> select tablespace_name tbspc,file_name,
  2  bytes from dba_data_files;

TBSPC      FILE_NAME                                     BYTES
---------- ----------------------------------------      ----------
USERS      /datafile/o1_mf_users_3d5x7crl_.dbf           1620705280
UNDOTBS1   /datafile/o1_mf_undotbs1_3d5x7cnt_.dbf         272629760
SYSAUX     /datafile/o1_mf_sysaux_3d5x7cb9_.dbf          1201733632
SYSTEM     /datafile/o1_mf_system_3d5x7c7t_.dbf           744488960
```

This database is made up from four database files, that you can see at the Operating System (Unix) level using standard operating system commands, and that you cann see at the database level using SQL queries.

Data may alternatively be stored using Oracle’s Automatic Storage Management, which is a volume manager made specifically for Oracle by Oracle. If you are doing this, the concept of files still exist inside the database and tablespaces are still made from files. However, the files are only visible as such inside the database and not at the Operating System.



Logical and Physical Storage

- Data Segment
- Index Segment
- Large Object Segments
- etc.

Tablespaces are the logcal storage media of an Oracle Database. Each tablespace contains data from one or more segments, such as the rows of a table, or the index of a table, and each segment, is made up of one or more extents.

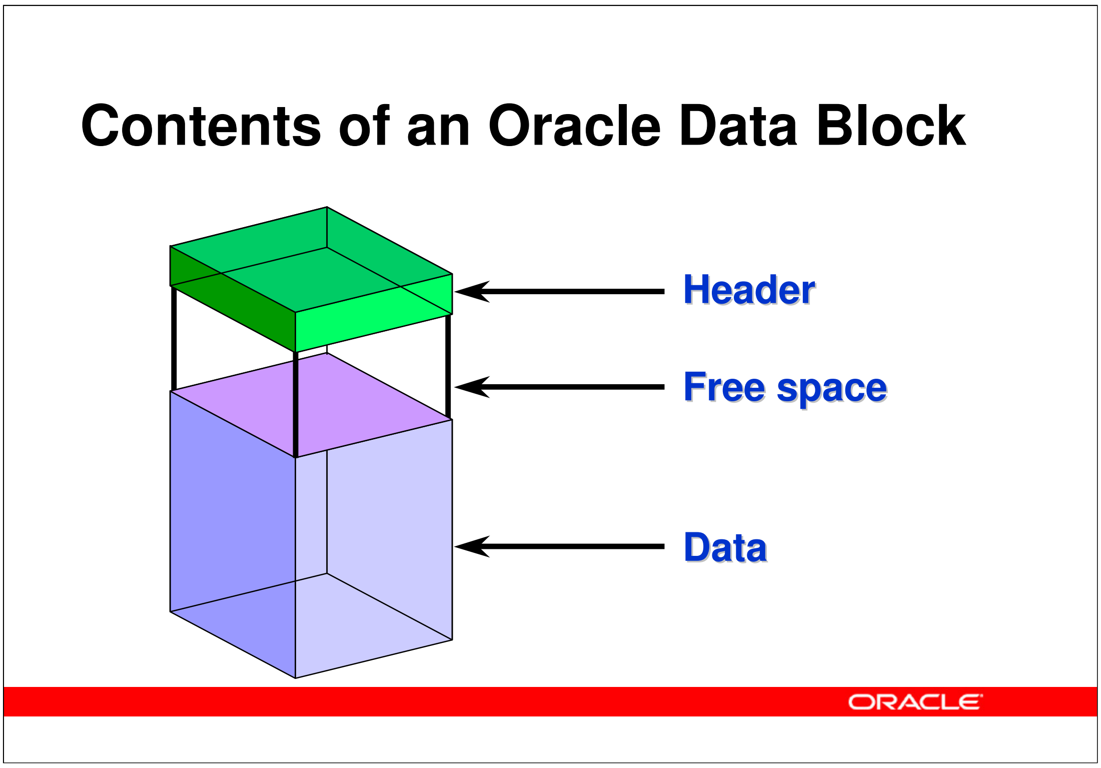
The picture shows a tablespalce, that is made up of two physical data files.

There are two segments shown, the yellow one (lightest grey) is made up of three extents, and the pink (medium grey) contains four extents. These could e.g. be a table and one of its indexes. The remaining part of the tablespace are unused blocks.

A segment, including an initial extent, is created when you create an object, such as a table or an index, and more extents are added as necessary, when data is inserted.

Extent size can be controlled at the tablespace level, for each segment individually, or by explicitly allocating an extent of a certain size. Normally, extent sizes are controlled by Oracle, using a bitmap of all used and free extents. Alternatively, extent size and allocation can be managed by the database administrator.

Note, there is not a one-to-one mapping between tables or indexes and datafiles, although you can specify, that a specific table is the only one stored in a specific tablespace.



Contents of an Oracle Data Block

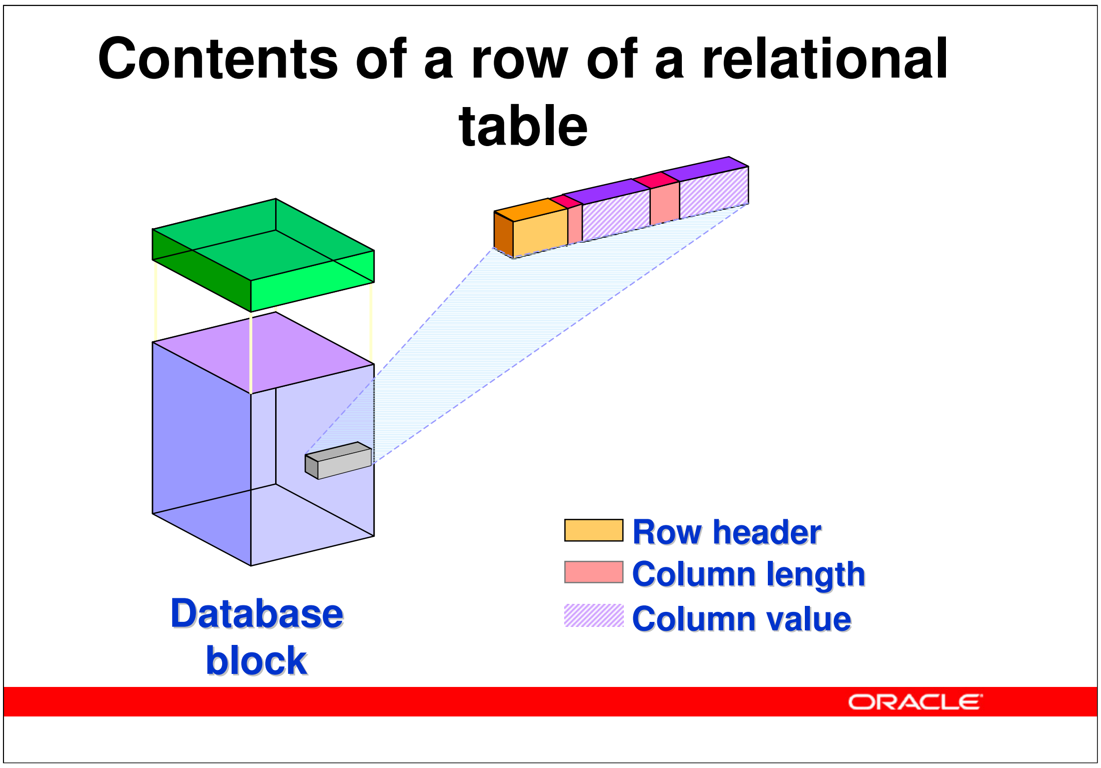
Header

Free space

Data

All data is stored in Oracle blocks, the size of which are defined when the database is created. Typical sizes are 4kb, 8kb, or 16kb.

Each block contains a header, which includes a directory of rows, and it includes the actual data. Space management can be applied by the database designer and/or database administrator to control the amount of free space in each block that is used to insert new rows or to update existing rows.



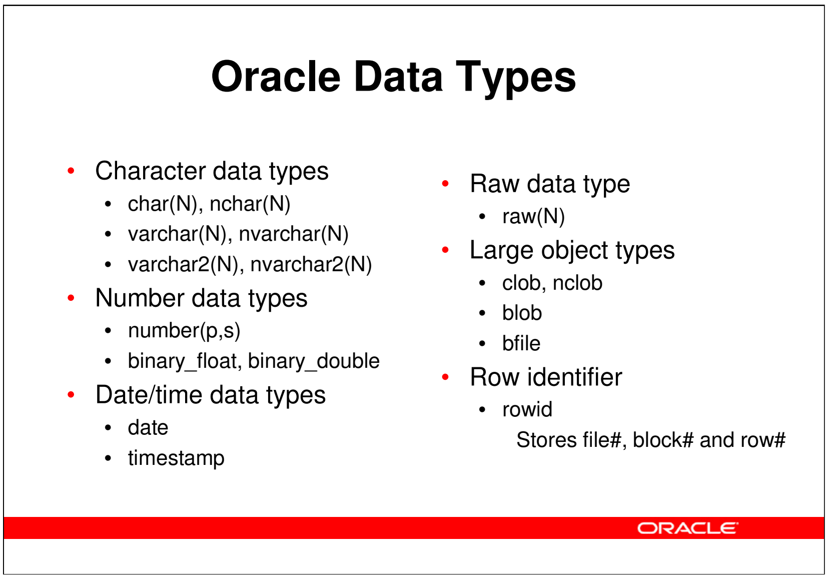
Contents of a row of a relational table

Database block

🟨 Row header
🟥 Column length
▨ Column value

Each row of a table is stored in the database block. The row storage includes a header and subsequent colun length/column data pairs. A row is completely identified by its rowid which consists of database file number (mapped to file name), block number in that file, and row number within the block.

The header of the data block contains, among other things, a directory of rows in the block.



Oracle Data Types

- Character data types
  - char(N), nchar(N)
  - varchar(N), nvarchar(N)
  - varchar2(N), nvarchar2(N)
- Number data types
  - number(p,s)
  - binary_float, binary_double
- Date/time data types
  - date
  - timestamp
- Raw data type
  - raw(N)
- Large object types
  - clob, nclob
  - blob
  - bfile
- Row identifier
  - rowid

    Stores file#, block# and row#

Oracle data types are:

char, nchar - Fixed length character string, maximum 2000 bytes, but only recommended for smaller strings.

varchar, varchar2, nvarchar, nvarchar2 - Variable length character string, maximum 4000 bytes. The “2” types are currently identical to the normal types, but the latter may change with an evolving SQL standard.

number - Variable length numbers, scale and precision can be specified. The datatype is 100% identical on all Oracle platforms, which would not be the case with *native* types such as **integer**, or **float**. When declaring tables, native types are mapped to corresponding number types. The maximum precision is 38 decimal digits, and the maximal range is at least $\pm 10^{125}$.

binary_float, binary_double – are used to directly store IEEE floating point numbers

date - Fixed length date (7 bytes) with second resolution.

timestamp – Date and time with variable resolution (up to nano-seconds) and potentially with timezone information

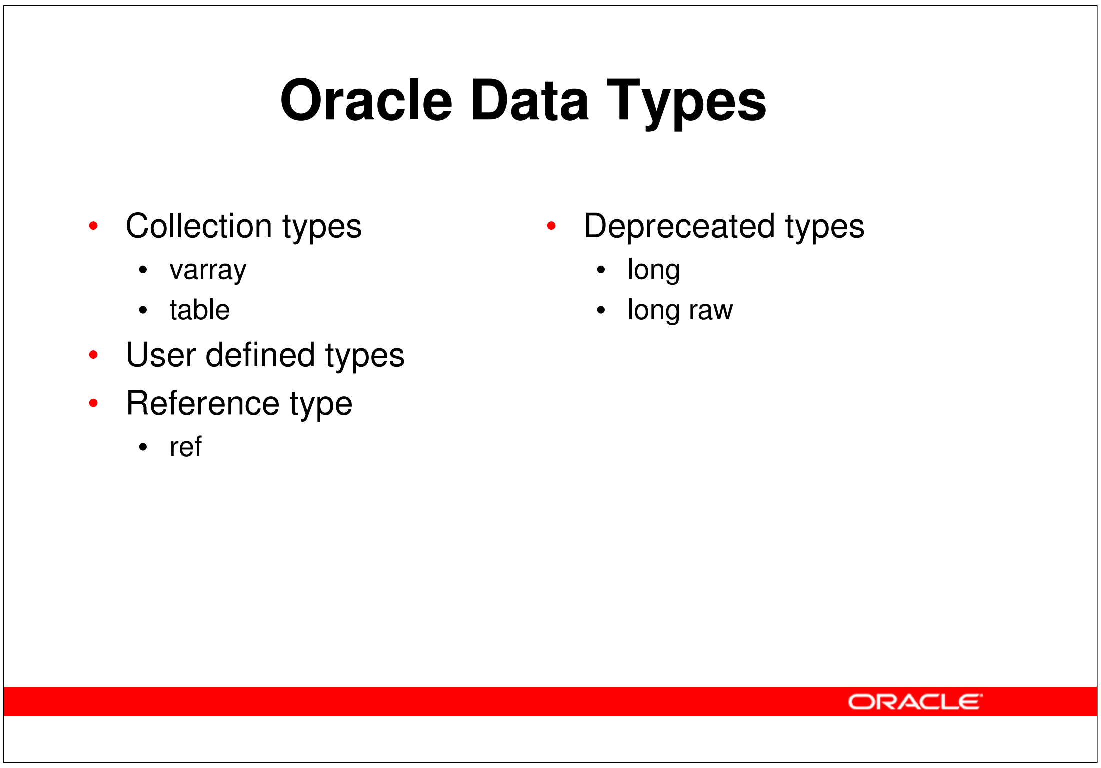
rowid - Row identifier, guaranteed to be unchanged during the lifetime of a row. Each row in each table is uniquely identified by a rowid, which contains file#, block# and row# (the details a sligthly more complicated)

raw - Raw binary data, maximum 2000 bytes, can be indexed.

clob, nclob, blob- Text or binary large objects, maximum 4Gb, cannot be indexed.

The normal text datatypes (i.e. without “n”) are stored in the database character set, which must have the first 127 ASCII characters (or all printable EBCDIC characters) at their usual location; this is e.g. the case with the variable length Unicode character set. The text datatypes with “n” (nchar, nvarchar(2), nclob) are stored in the database’s national character set, which can use any character set e.g. fixed width multi-byte character sets. If the client and the server use different character sets, the underlying interface will translate between them.

Extensive implicit or explicit data conversion is possible



Oracle Data Types

- Collection types
  - varray
  - table
- User defined types
- Reference type
  - ref

- Depreceated types
  - long
  - long raw

In the object relational model, the following extra data types are available

table, varray - Collections, i.e. table (unordered) or varying arrays (ordered) of other types.

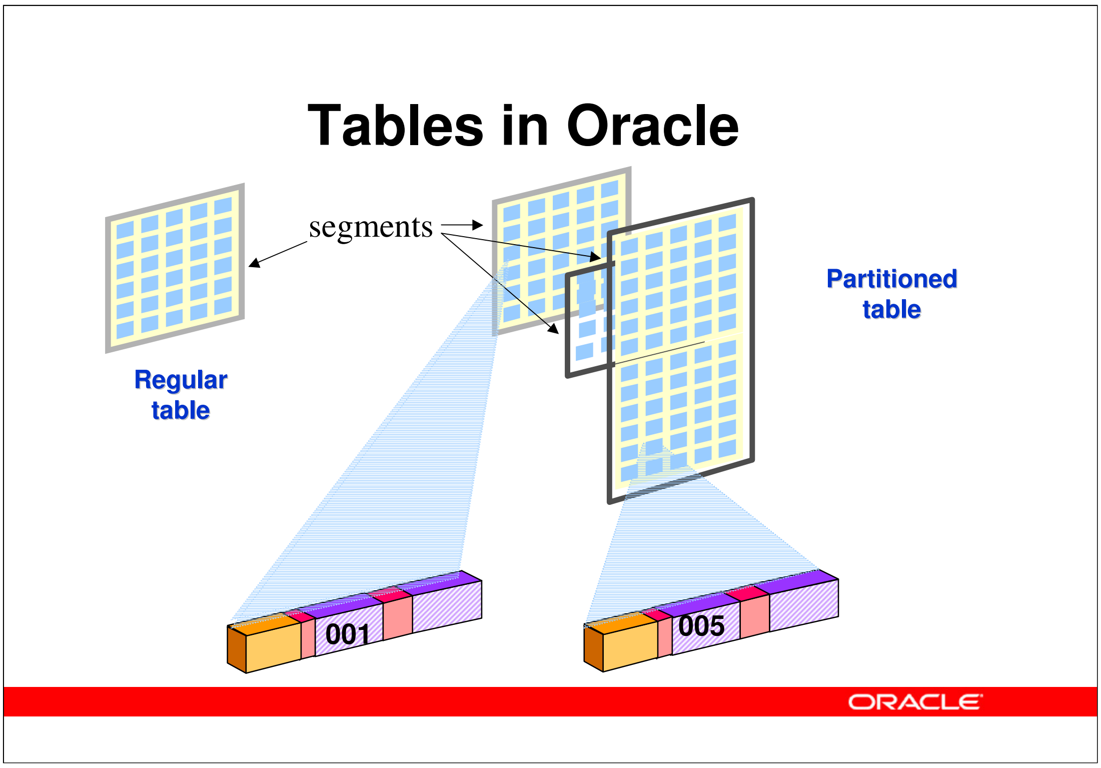
user defined types - Records of scalars or other types

ref - References to objects

For compatibility with older versions of Oracle, the following are supported but depreceated:

long – Character type with up to 2GB length

long raw – Raw binary type with up to 2GB length



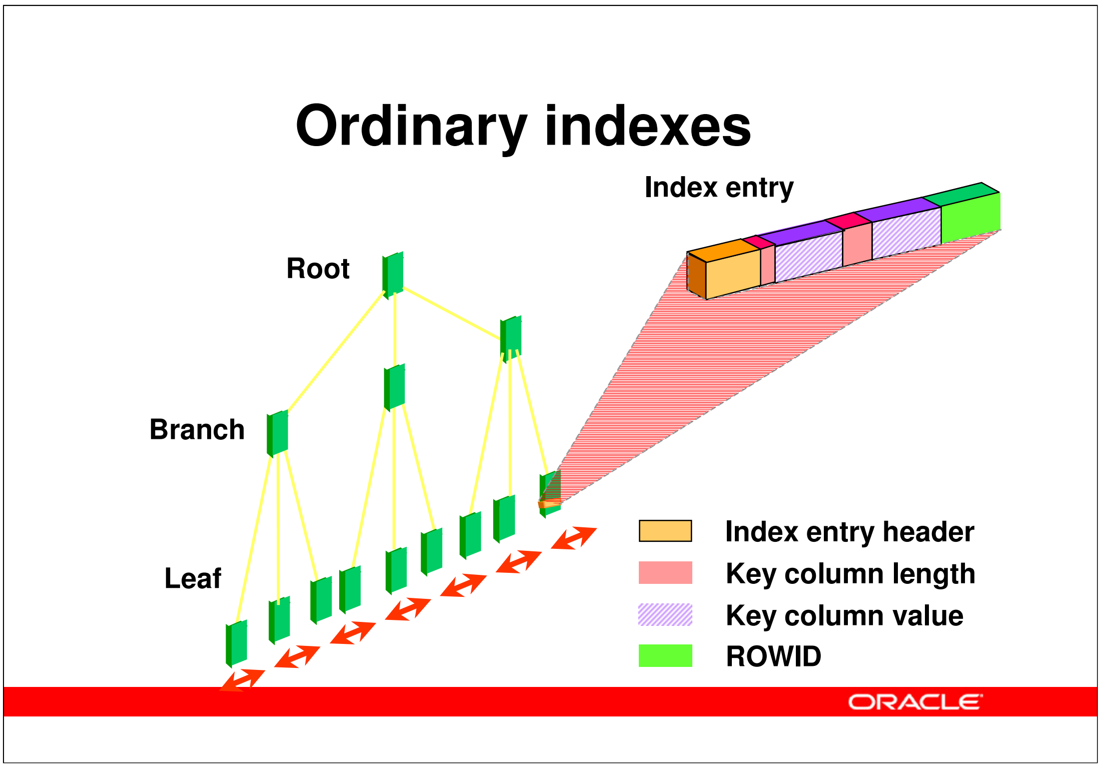
Tables in Oracle

Tables in Oracle can be stored in three different ways:

•Ordinary tables are stored with all rows in no particular order, this makes up one segment (with potentially many extents). In most cases, one or more indexes will be used as well.

•In Partitioned tables, rows of the table are stored in different segments (and typically also different tablespaces) depending on a partition key, e.g. one partition per month of data or one partition per geographical region. This is typically used with very large databases (100+ Gb), and does e.g. allow database administrators to backup (and recover) parts of a table at different times. Partitions may alternatively be based on hash values rather than ranges or lists of values, and there can be two levels of partitioning.

•Index organized tables are described in a few slides



Ordinary indexes

Index entry

Root

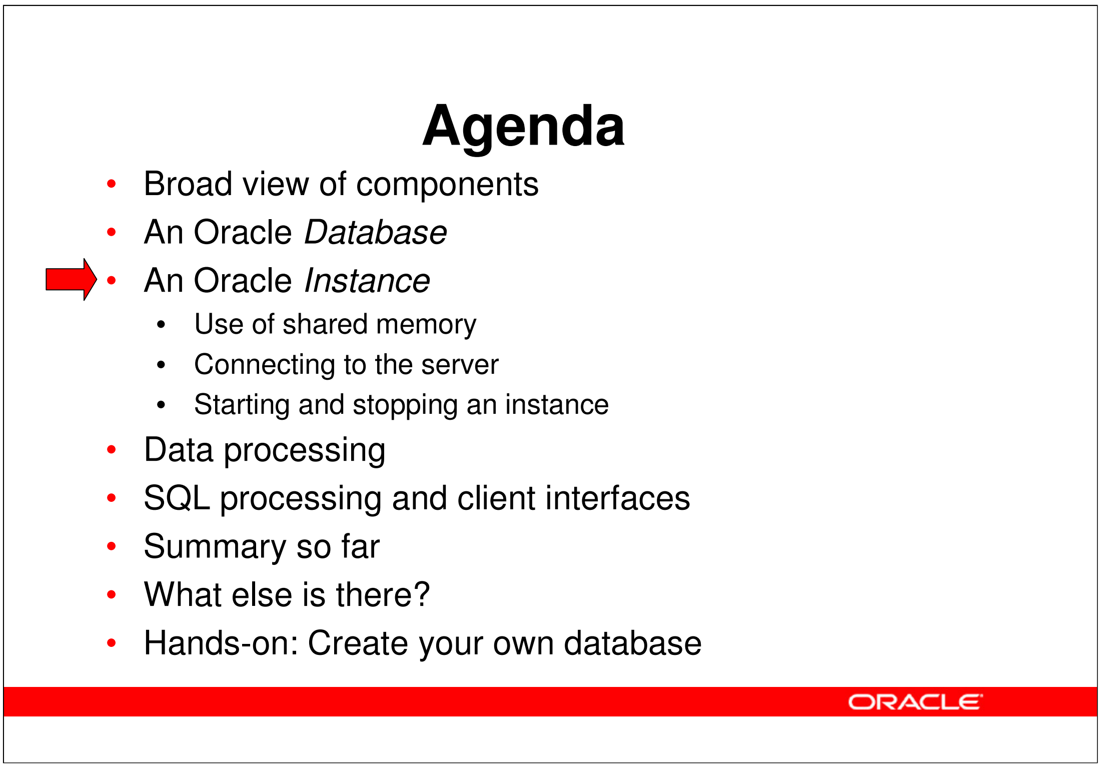
Branch

Leaf

- 🟧 Index entry header
- 🟥 Key column length
- ▨ Key column value
- 🟩 ROWID

Ordinary indexes are binary an contains one or more levels of branch blocks (the top one being the root), and one level of leaf blocks. Each index entry stores a header, the actual column values and a rowid pointing at the data block. In case of an index organized table, the rowid part is replaced by the full row of the table.

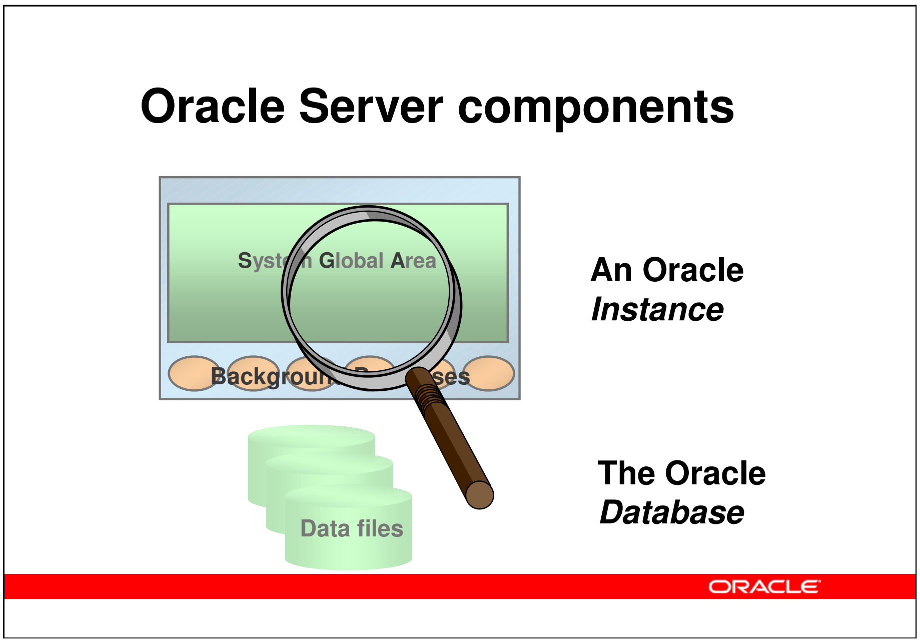
Indexes may also be partitioned, like tables



Agenda

- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
  - Use of shared memory
  - Connecting to the server
  - Starting and stopping an instance
- Data processing
- SQL processing and client interfaces
- Summary so far
- What else is there?
- Hands-on: Create your own database

The second major part of the black box is the Oracle instance, which is a set of memory and processes running on the database server.



Oracle Server components

An Oracle *Instance*

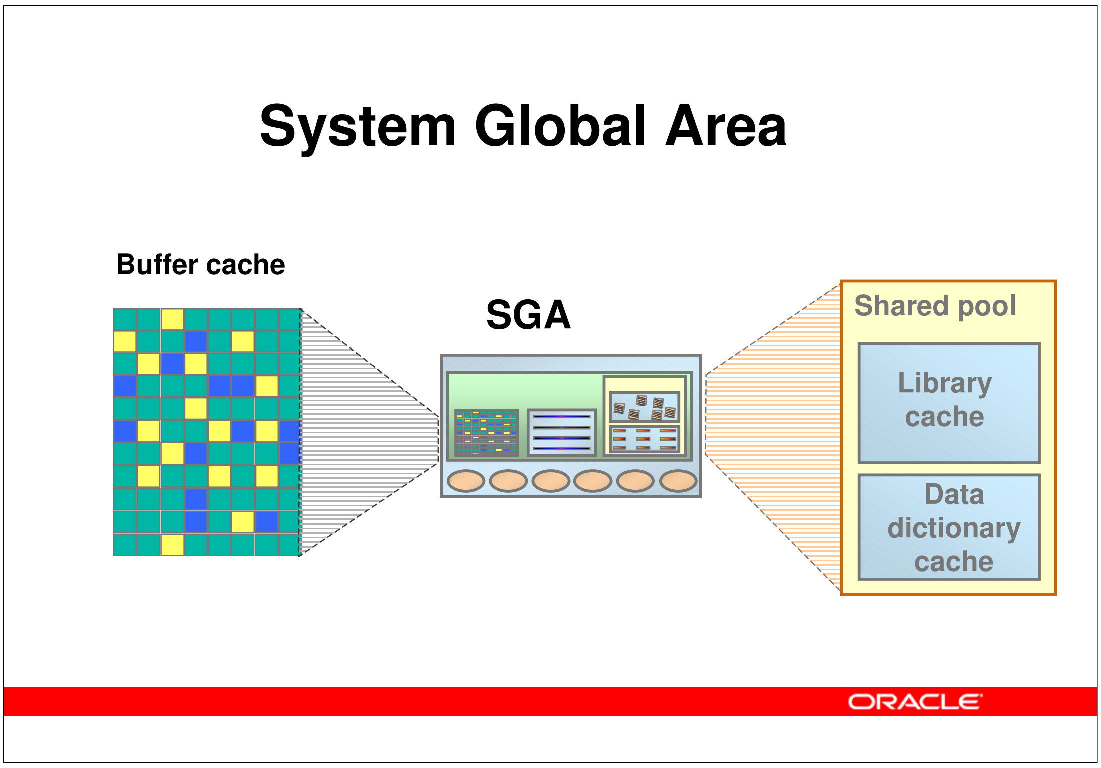
The Oracle *Database*

Looking into the Oracle server, two major components are seen:

•The Oracle *Database*, which is the set of operating system files, where the application's actual data are stored (such as customers, orders and order lines), together with Oracle's own internal information (such as information about users registered in the database, or tables found in the database). The Oracle Database as such, cannot be directly accessed in any way (except for backup) by clients. In contrast to other systems, there is no correlation between the concept of a user, an application or a file system and an Oracle database; rather, there is often only one database on each server, and you can even have a single database spanning multiple servers. This is known as Oracle Real Application Clusters.

•The Oracle *Instance*, which is a set of memory and process structures, running on a specific computer. This is the point of access for clients, and the instance is responsible for translating the SQL calls given by the client, to acual data transfers to and from the database stored in the operating system files. An Oracle instance is normally associated with an Oracle database, and those two together make up the Oracle Server. However, an Oracle instance may exist without being associated with an Oracle database; this is e.g. the case before the database is created or as an intermediate step during the startup of the Oracle server. An instance cannot be associated with more than one database, but one database (on a set of physical disks), can be associated with multiple instances (each on a separate computer), if the actual hardware configuration allow multiple computers to concurrently access a single set of disks.

•Inside the instance, two important parts are seen: The *System Global Area* (or *SGA*), which is a shared memory segment, typically of a size, which is roughly half of the physical RAM available on the computer. There is no direct access from clients to the SGA. Additionally, there is a set of *background processes*, which are working indirectly on behalf of the clients to do various specialized tasks, such as clean-up. These background processes are directly attached to the SGA, and can directly read and write to the disk files making up the database.



System Global Area

Buffer cache

SGA

Shared pool

Library cache

Data dictionary cache

The large shared memory segment, the SGA, contains the following major components:

•The buffer cache, which is a cache of disk blocks, very similar to the file system cache found in most operating systems. Blocks are always read and written in sizes of the Oracle Block Size, which is defined, when the database is created, although tablespaces can be added with a different block size.

•The shared pool, which contains two main components.

•The library cache, which is a cache of SQL statements, etc.

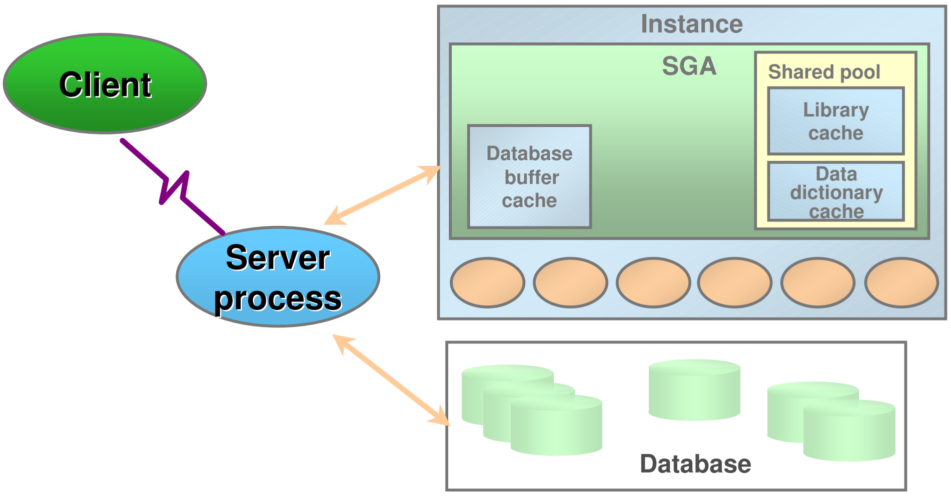
•The dictionary cache, which caches Oracle own internal information, such as information about users and tables.

As a very rough rule of thumb, the buffer cache and the shared pool, which by far are the largest of the SGA, will each make up little less than half the total SGA size.

Other parts of the SGA contain information about currently running processes, locks, etc.

The use of some of the background processes will be explained later.

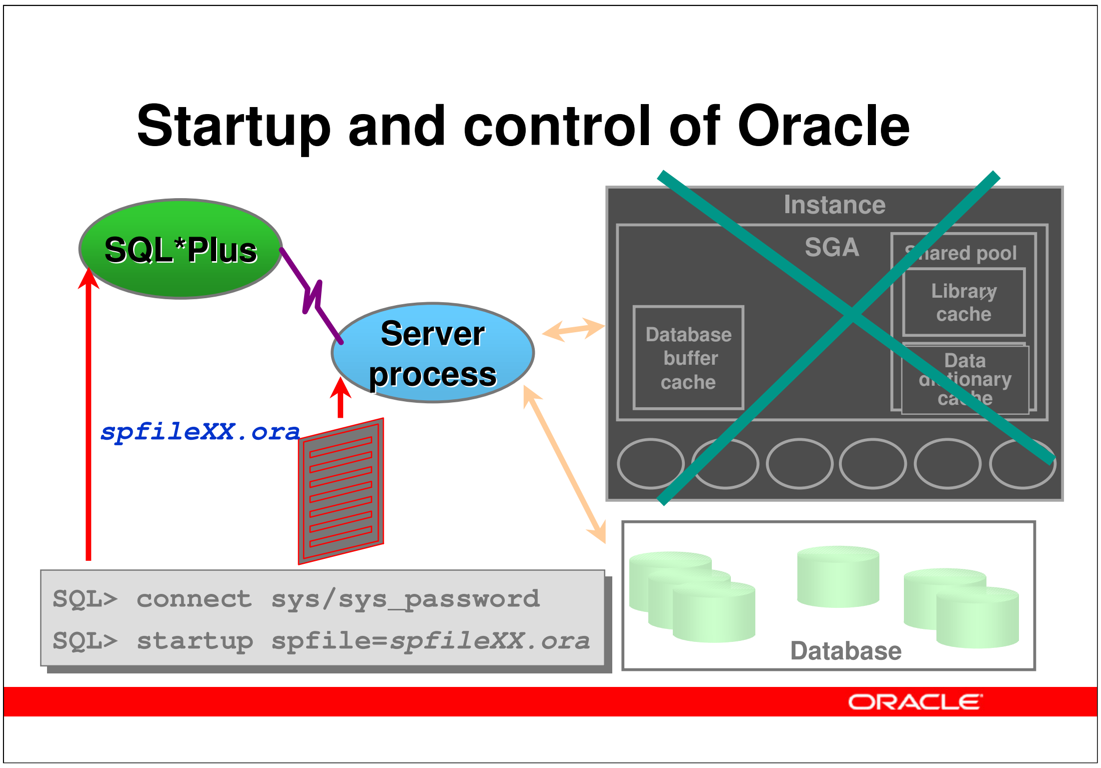
Connecting a client to the server



The client process runs in its own address space (actually as its own operating system process, quite often on a separate computer), completely separated from the Oracle Instance by the network or the inter process communication. Hence, the client cannot attach to the SGA, and cannot read or write to the database files.

The server process, which executes SQL statements sent from the client process, executes within the address space of the SGA, and within the priveledge space of the database, and can hence get and modify information in the SGA, and can read and write to the database files.

Full security is guaranteed by the Oracle security mechanisms and the SQL language.



Startup and control of Oracle

SQL*Plus

Server process

spfileXX.ora

Instance

SGA

Shared pool

Library cache

Database buffer cache

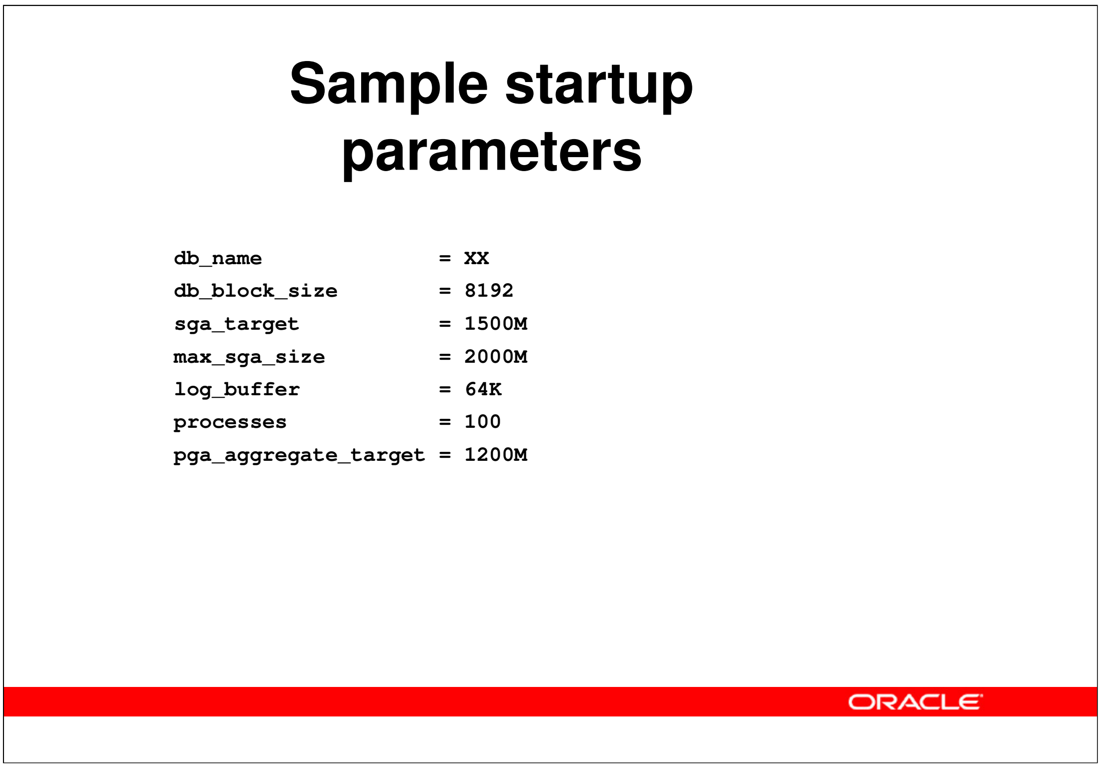
Data dictionary cache

Database

```
SQL> connect sys/sys_password
SQL> startup spfile=spfileXX.ora
```

Right after system startup, only the Oracle database will exist; the instance will not be running. To start the instance, a user with DBA authority connects to a server process using either the sqlplus utility or a GUI tool such as Oracle Enterprise Manager. The utility will connect to a server process, which will get the startup command and read a parameter file. This file has information about different things such size of the buffer cache and the shared pool, and about the maximum number of processes to start. The server process then creates the SGA and starts the necessary background processes.

On Unix systems, this is typically done in a system startup script, such as /etc/rc, and on Windows Servers, this is done as a service.

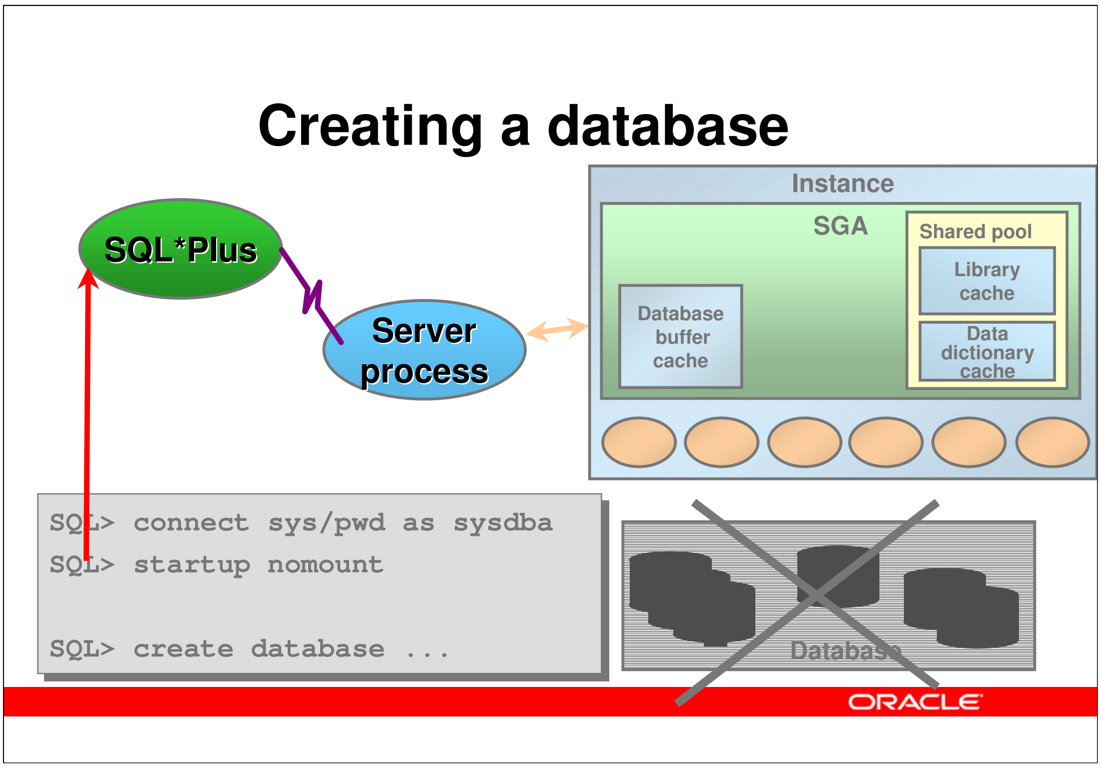


Sample startup parameters

```
db_name              = XX
db_block_size        = 8192
sga_target           = 1500M
max_sga_size         = 2000M
log_buffer           = 64K
processes            = 100
pga_aggregate_target = 1200M
```

ORACLE

During startup of the Oracle instance, a parameter file is read, specifying information on how to configure Oracle. Some, but not all, of the parameters can later be modified at runtime either at the system level for the complete instance, or at the session level for a specific session.

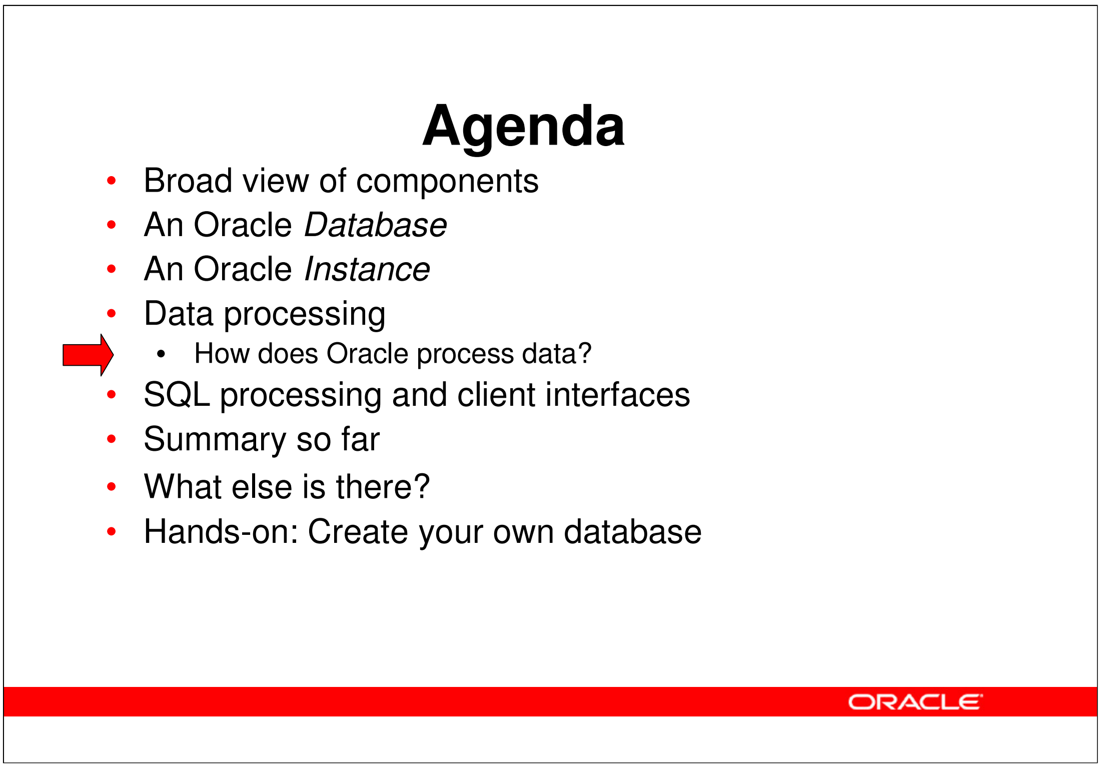


Creating a database

```
SQL> connect sys/pwd as sysdba
SQL> startup nomount

SQL> create database ...
```

On the previous slide, you saw how to start an instance when the database already exists – this slide shows the opposite: Creating a new database when only the instance exists. Initially, you start an instance that is not associated with a database, using 'startup nomount'. The nomount option tells the server process, that it should create the SGA and the background processes, but not attempt to mount an actual database. Once the instance is running, you can run the 'create database' command to actually create the initial set of database and redo log files.

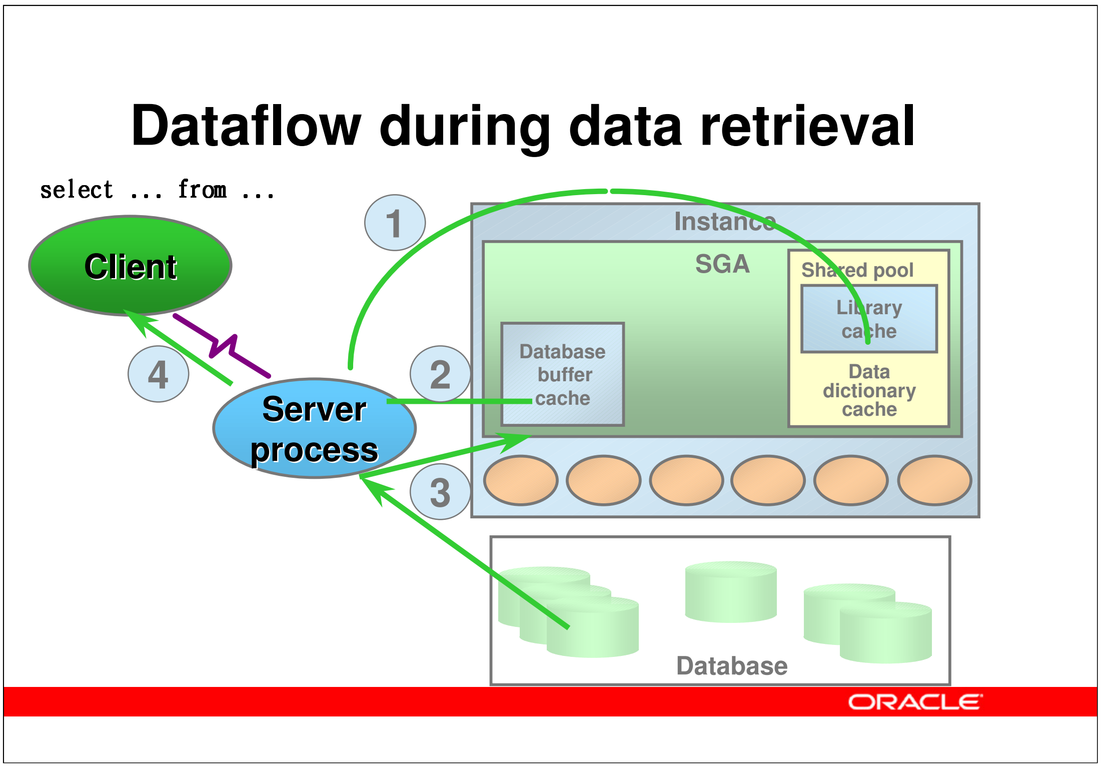
In most practical cases, however, databases are created by restoring a generic (i.e. empty) database from a distributed backup.



Agenda

- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
- Data processing
  - How does Oracle process data?
- SQL processing and client interfaces
- Summary so far
- What else is there?
- Hands-on: Create your own database

The following slides show how data flows through Oracle when you retrieve data, or when you modify data in the database.



Dataflow during data retrieval

`select ... from ...`

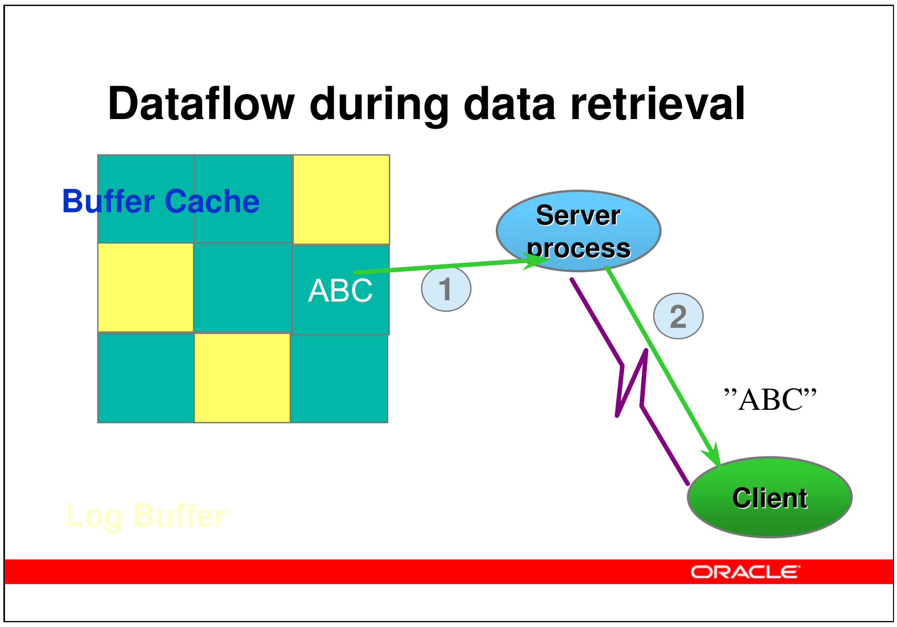
During queries (SQL select statements) data is found in the actual database, i.e. in the database files, and the server process, that is connected to the client together with the other parts of the Oracle instance is responsible for executing the SQL statement, getting data from the database and sending these to the client.

The major steps in this are:

1. The SQL statement is looked up in the library cache part of the SGA. If it is not already there, it will have to be parsed first.
2. The blocks containing the data to be retrieved, including e.g. index blocks, are identified in the buffer cache part of the SGA.
3. If the blocks do not exist, free buffers are found in the buffer cache, and the necessary blocks are read from the database files.
4. The information in the blocks are decoded into rows and columns, which are sent to the client.

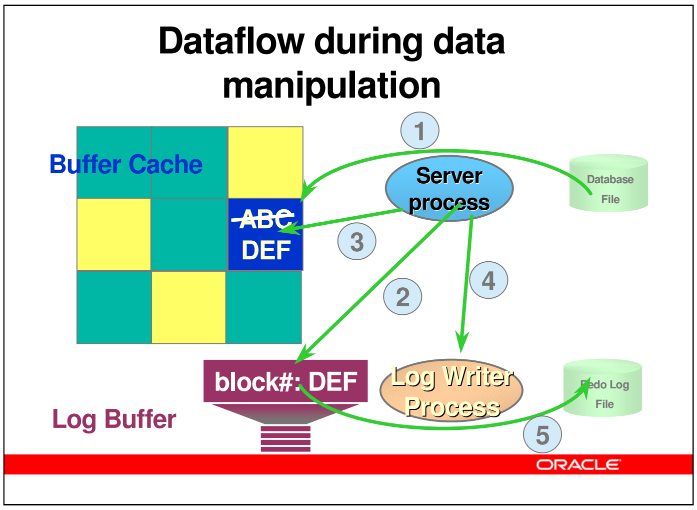
There are some important things, that should be noted:

- Oracle uses caches at different places to ensure good performance of repetitive things. For the buffer cache, this is very much like an operating system file buffer cache; the behaviour and use of the library cache will become clear later.
- The actual reading of data from the database files into the buffer cache is done by the server process. We shall see later, that writing of data is done by one of the background processes.
- The physical (on disk) storage of data is not seen by the client, which simply receives rows with columns of data as expected.



Dataflow during data retrieval

1. The buffer cache contains exact copies of blocks on disk, where data is stored in the block format shown previously, with typically several rows per block.
2. Over the network connection, the data is sent in row format, ready to be used by the client. The client purely needs to use SQL and can deal with rows of data from tables – the client is not at all concerned with actual storage.



Dataflow during data manipulation

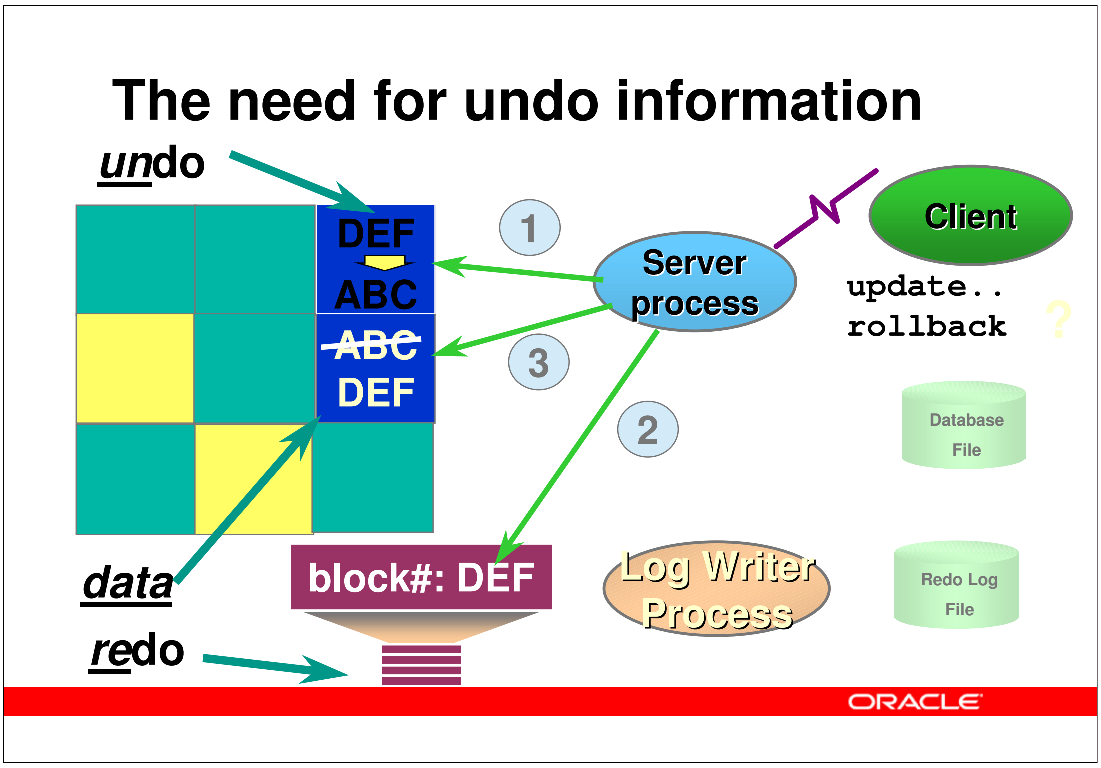
During processing of a data manipulation statement (DML, i.e. SQL insert, update or delete), the basic processing is like for queries. I.e. handling of SQL and the library cache is the same.

A new element is now identified in the SGA: the *log buffer*. The log buffer logs all changes made to the buffer cache, so that these changes can be redone in case of a recovery. The processing steps are:

1. The block, that needs to be modified is read from the disk (unless it is already found in the cache)
2. The server process (on behalf of the SQL statement executed by the client), makes an entry into the log buffer, specifying the operation to be done on the data block; in case of a later recovery, the redo information is needed. The picture shows that the redo log entry indicates which block is being updated (it is actually more than simply a block number), and it shows the actual change. No information is logged about the previous value in the block, as this information is not needed to redo the operation.
3. The server process makes the actual change in the block in the buffer cache. In the picture, the value “ABC” is replaced by the value “DEF”. At this point in time, the block in the buffer cache is different from the block in the datafile; this is called a dirty block.
4. The user issues the *commit* operation, and the server process indicates this to the *log writer* background process. This process is part of the Oracle instance, and is started when the instance starts.
5. The log writer process writes the log buffer to a *redo log file*. At this time, in case of recovery, the old block, found in the database file, and the redo log record, found in the redo log file, can be used to redo the change made by the user.

New conecpts introduced:

- The *redo log buffer*, which is found in the SGA is used to store log information between an actual update and the coresponding commit. The typical size is up to a few hundred kilobytes.
- The *log writer* background process, often abbreviated to *LGWR*, which writes the redo log buffer to the redo log files.
- The *redo log* files, which is where Oracle stores log records necessary for recovery. Typical sizes are up to a few hundred megabytes or in very high-transaction systems, a few gigabytes.



The need for undo information

undo

DEF

ABC

~~ABC~~

DEF

1

2

3

Server process

Client

update..

rollback

?

Database File

data

block#: DEF

Log Writer Process

Redo Log File

redo

ORACLE

The previous slide showed the processing during normal DML operations, i.e. where the client eventually commits the operation. However, clients may want to undo the operation by performing a rollback in stead of the commit, and the simple picture does not show this. On this slide, the details of undo processing is shown.

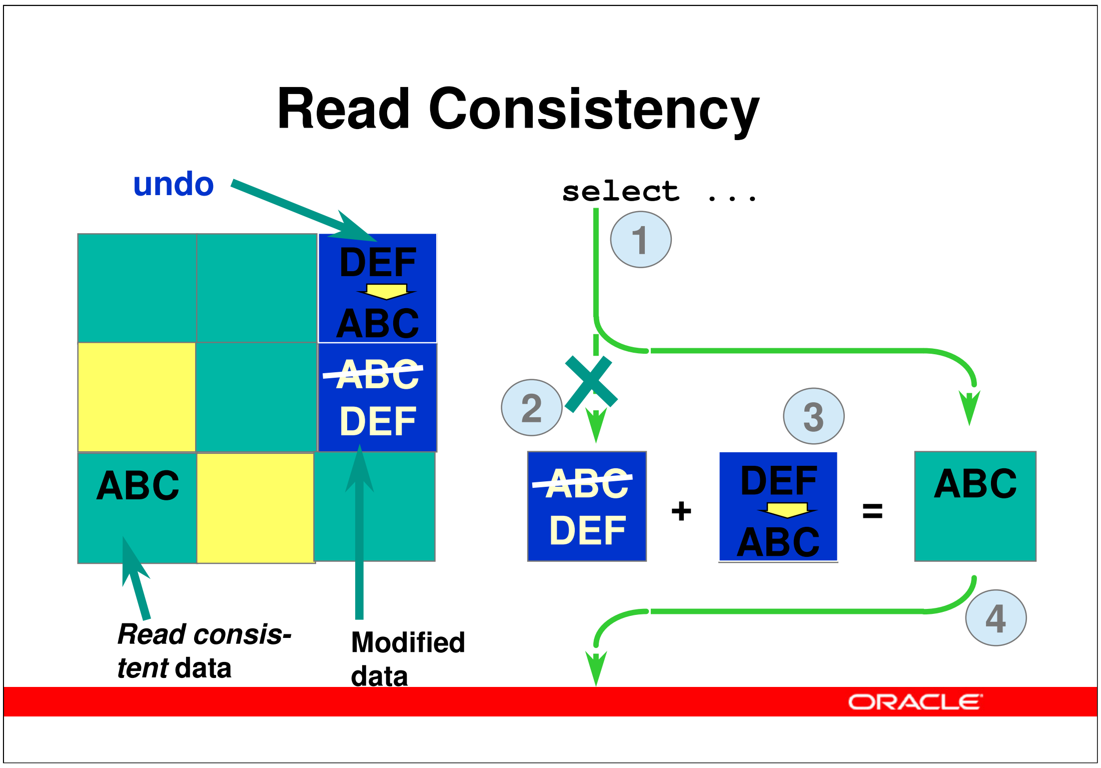
1. Before doing the actual modification of the data block, information necessary to undo the operation is written into an undo block, which is also found in the buffer cache.
2. The redo log information is written to the redo log buffer.
3. The actual data block is modified. In order to reconstruct the data block to its state before the modification, the information in the rollback block can be applied to the modfied block.

Important points

Note the duality of the two words “redo” and “undo”. Rollback, which is the normally used word, is synonymous with undo, which is frequently used in Oracle internal documents.

The undo blocks are actually part of an undo tablespace, which are created and managed by the database administrator. The undo tablespace is system controlled; but they the actual undo blocks are stored in database files just like ordinary table and index blocks, and the undo blocks are cached in the buffer cache of the SGA.

The modification of the undo block (step 1 above) is actually logged in the redo log buffer, just as the modification of the actual data block.

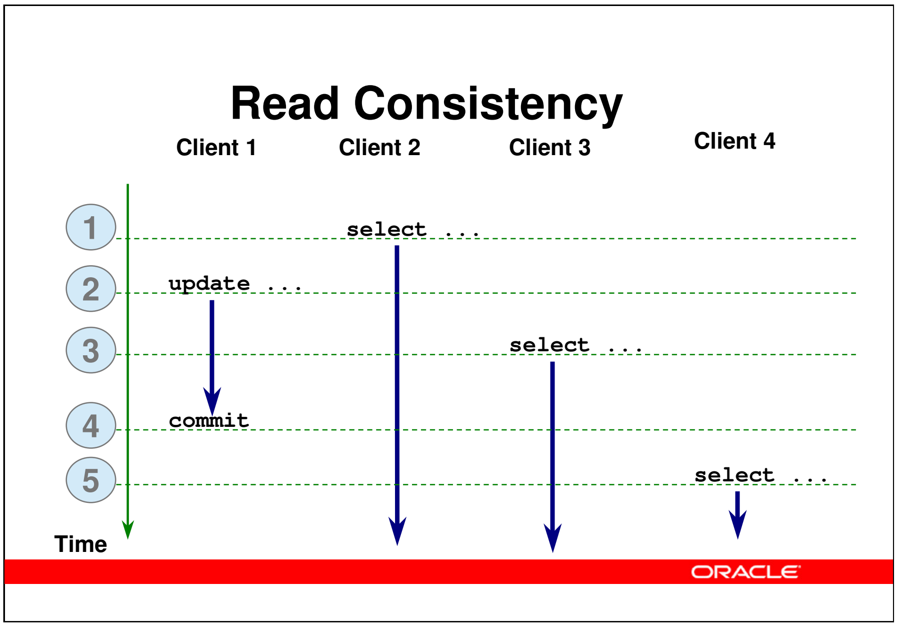


Read Consistency

The undo (rollback) information, that was discussed on the previous slide, is used in other cases, besides an actual rollback operation from the client. In the scenario above, some user has updated the block with “ABC” into “DEF”, which is actually changed in the data block and registered in the undo block as rollback information. Concurrently, some other user is performing a query.

1. A user is performing a query, which reads blocks in the buffer cache.
2. A block is read, which is too new, i.e. it is modified after the start of the query. Hence, to get a consistent read for the query, this block cannot be used.
3. The information in the undo-block is applied to a copy of the modified block, with the effect that a block with the original block contents is constructed.
4. The query uses this read consistent copy of the block and continues.

This behaviour is fully automatic and cannot be switched off, so Oracle guarantees consistency for all queries.



Read Consistency

Client 1 Client 2 Client 3 Client 4

1 select ...

2 update ...

3 select ...

4 commit

5 select ...

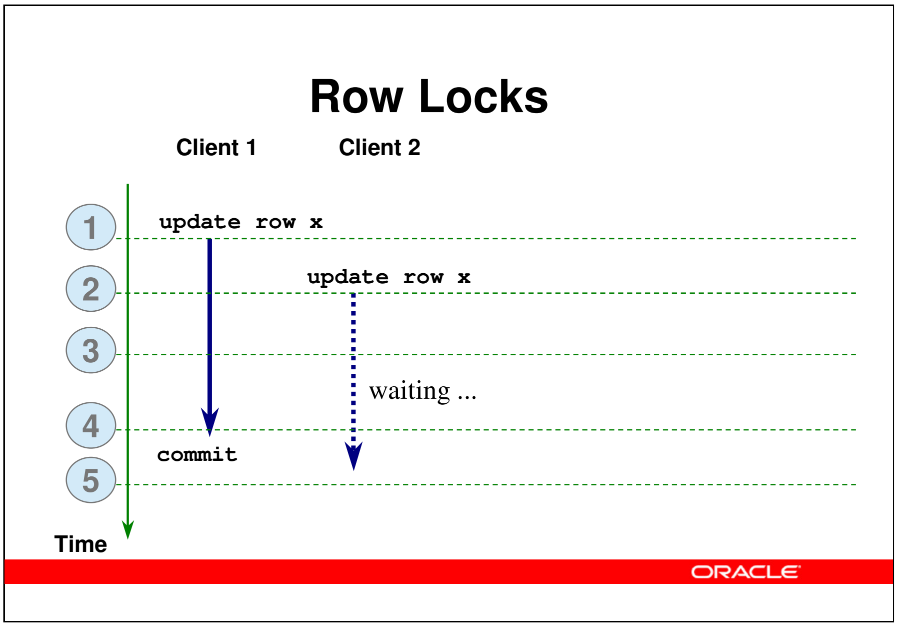
Time

The behaviour of the *read consistency* model is described further in this slide:

1. A user, client-2 is starting a query at time-1. The read consistency model ensures a consistent view of data during the entire run of the query, which is a snapshot of the database taken at the time the query starts.
2. The user, client-1 executes an update operation (which still may be either committed or rolled back).
3. The user, client-3, starts a query. Since the start of the query is before the commit of client-1's update operation, client-3 does not see the update done by client-1.
4. At this time, client-1 decides to commit the update. However, since both queries of client-2 and client-3 started earlier than this time, none of the queries will see the update.
5. Client-4 starts a query after the commit of client-1, hence the query started at time-5 does see the change made by client-1.

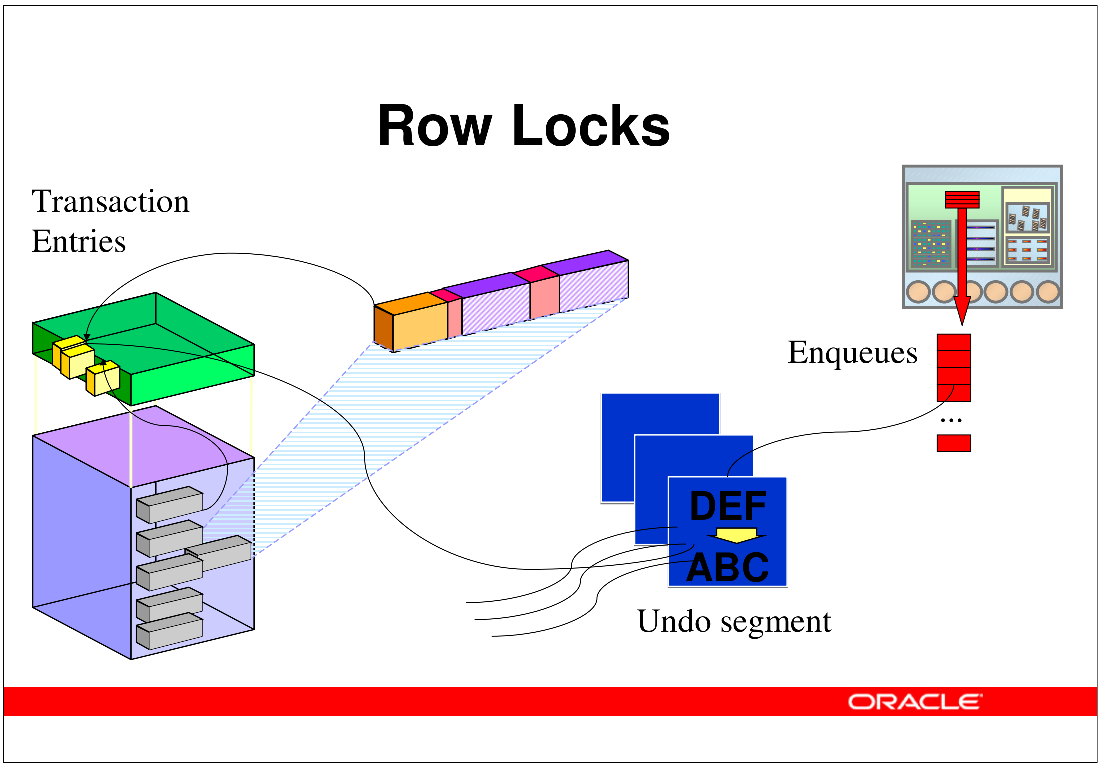
Important points:

- The snapshot time of a query is the start time of the query, independent on the time taken to complete the query.
- Effective timestamp of DML operations (insert, update, delete) is the commit time, and only the client actually performing the DML operation, is able to see the changes made until commit time.
- The undo blocks are used by all other clients to reconstruct a read consistent view of data, taken at the start time of the query.
- Due to the fact that undo blocks may be needed by queries after the commit of an operation, undo is not released immediately after commit.
- Undo blocks are in fact managed automatically in circular buffers, which means that undo information is being overwritten after a certain period of time. As of Oracle9i, this is managed automatically by the database administrator specifying the time to keep undo information available.
- If very long running queries (several minutes to hours) are executing at the same time there is a risk of ageing out necessary undo information for the query. In this case, an error will be returned to the client. Note, however, that this is far better than using other database systems, where concurrently running updates and queries are either impossible due to locking, or erroneous due to queries returning in-consistent results.



Row Locks

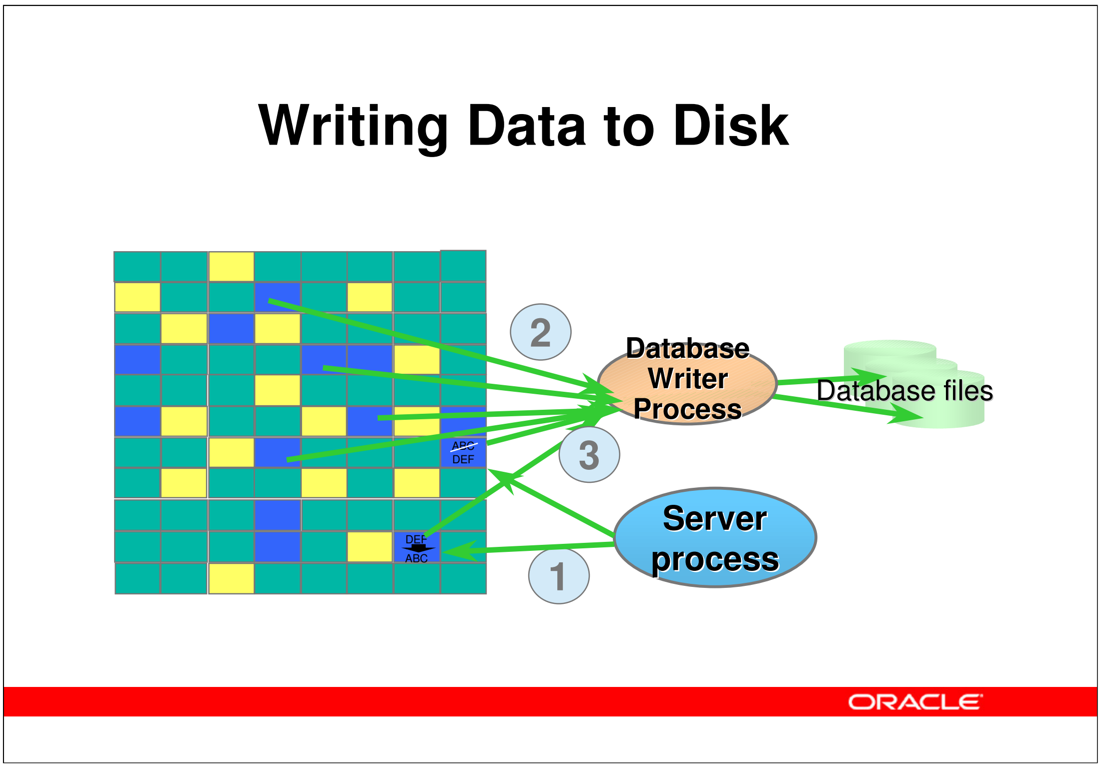
After issuing a SQL statement that need to lock a resource – typically a row – another SQL statement attempting the same need to wait until the first client has either committed or rolled back.



Row Locks

Row locks – which can e.g. come from DML statements or from ‘select for update’ statements – are not directly associated with any lock resources. In stead, *enqueues* that can be considered as advanced locking structures are stored in the System Global Area; they can be held shared or exclusive, and there can be a set of holders and waiters for them.

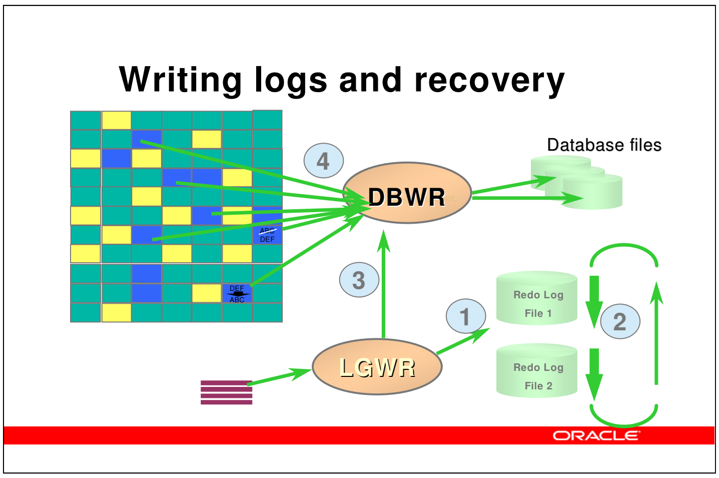
At the block level, the header includes a list of current row locks – transaction entries – referring to rows locked in that particular block. Row locks can therefore best be characterized as attributes of the row, as they are really stored in the block. Whenever a change to a block – including locking a row – is needed, that change is under transaction control and is associated with an undo segment. The result is that the number of enqueue resources required for locking depends on the number of objects (such as tables) and transactions but not on the number of concurrent row locks. Hence, locks never need to escalate to the page or table level.



Writing Data to Disk

As it has been seen earlier, whenever data is changed by a server process on behalf of a client's SQL request, the server process makes modifications to the blocks in the buffer cache, including undo blocks, and writes redo information to the redo log buffer, which is written to disk by the log writer process at commit time. This implies that the buffer cache over time will be filled with modified blocks, so called dirty blocks. These blocks really need to be written to the database files. This is the mechanism for doing this:

1. Blocks are modfied in the buffer cache, and marked as dirty.
2. At regular intervals, or if no free buffers are found, the database writer process, abbreviated DBWR, will write dirty blocks to disk. For performance reasons, writing is done in batches and it will use asynchronous I/O.
3. Note specifically, that undo blocks are also written to disk – we will see later why this may be useful.



Writing logs and recovery

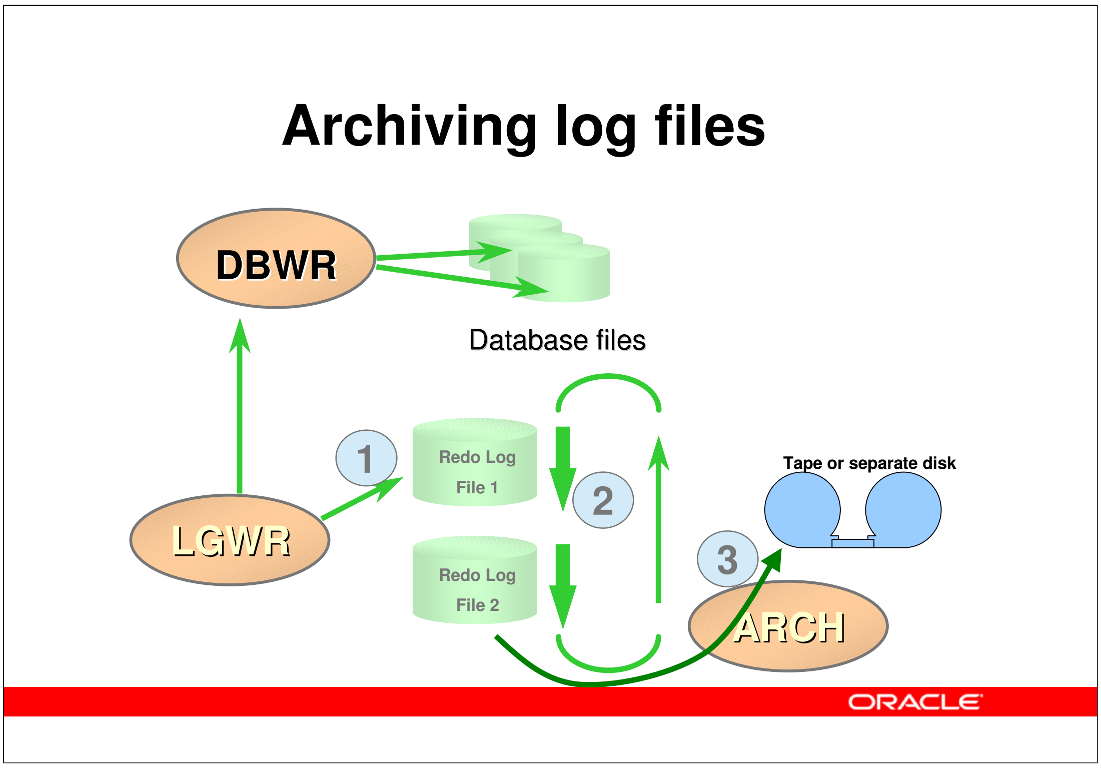
This slide shows some details of the way the log writer and the database writer work together. In general, there are at least two redo log files, and the set of files are used sequentially; when one fills up, the next log buffer is written to the next redo log file.

1. The LGWR process writes log buffers to one of the redo log files. The dirty buffers in the buffer cache are not yet written to the database file, but in case of a recovery, the old blocks in the database file plus the information in the redo log file can be used to reconstruct the modified blocks.
2. The redo log file fills up, and should now be made to ready for the next cycle of redo log writing. This means, that before the current redo log files is being overwritten, all dirty buffers in the buffer cache must be flushed, so that recovery of the blocks are not necessary.
3. The LGWR instructs the DBWR, that a flush is necessay, this is called a checkpoint. The checkpoint must complete before the current redo logfile is being overwritten.
4. The DBWR executes the checkpoint, which means that all dirty buffers are being written to the database file.

Important points

A checkpoint, which is a complete flush of all dirty buffers in the buffer cache, occurs at least whenever a redo log file becomes full. The database administrator my choose to make this happen more often. If a checkpoint does not finish before the complete cycle of redo log files, all operations in the Oracle Server will have to wait for the checkpoint to complete.

If a database crash should occur, the recovery time will be long, if there are many log records in the log file(s) since the last checkpoint. And conversely: If there a few log records, recovery time will be fast. The database administrator can specify that maximum recovery time accepted, and Oracle will make sure checkpoints occur sufficiently frequently to keep the recover time under the limit.



Archiving log files

DBWR

Database files

1

Redo Log File 1

2

Redo Log File 2

LGWR

3

Tape or separate disk

ARCH

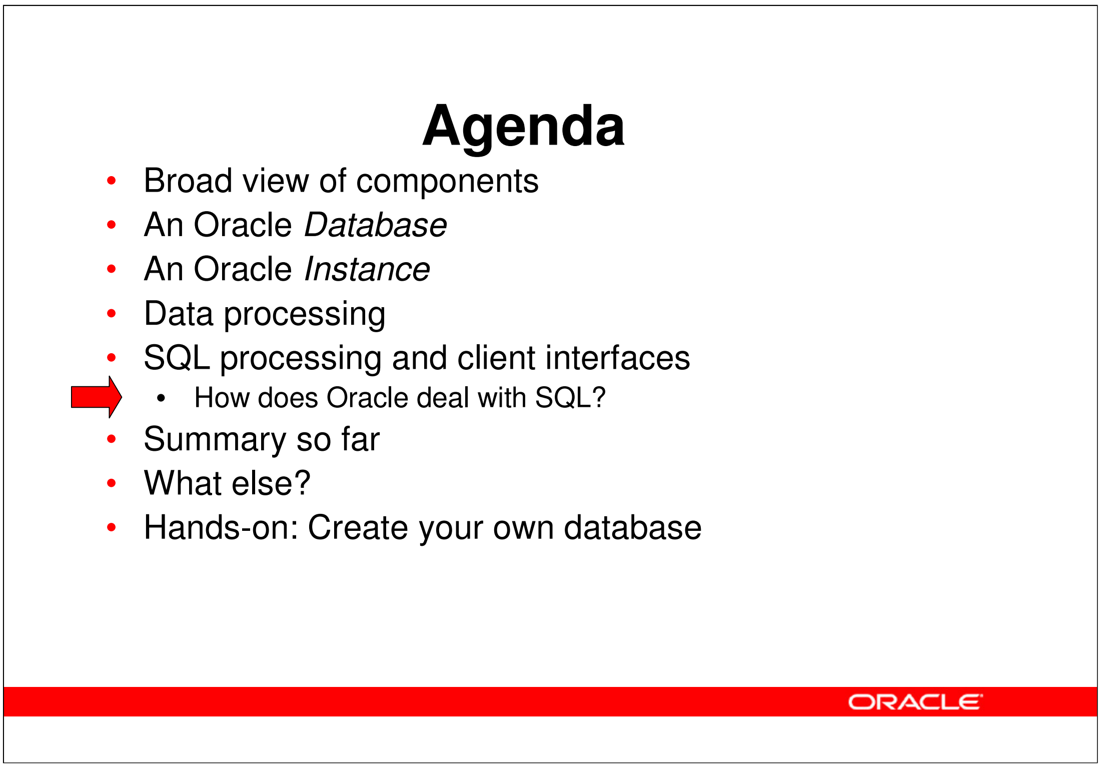
ORACLE

This slide shows the work of the archiver. If you need to be able to recover from e.g. a disk crash by using an old copy of the database files, all redo logs created since the old backup, must be available.

1. The LGWR process writes log buffers to one of the redo log files.
2. The redo log file fills up, and should now be made to ready for the next cycle of redo log writing.
3. The archiver process will take a filled redo log file and copy it to a different device, either a separate disk (from where it may later be written to a tape) or directly to a tape.

Important points

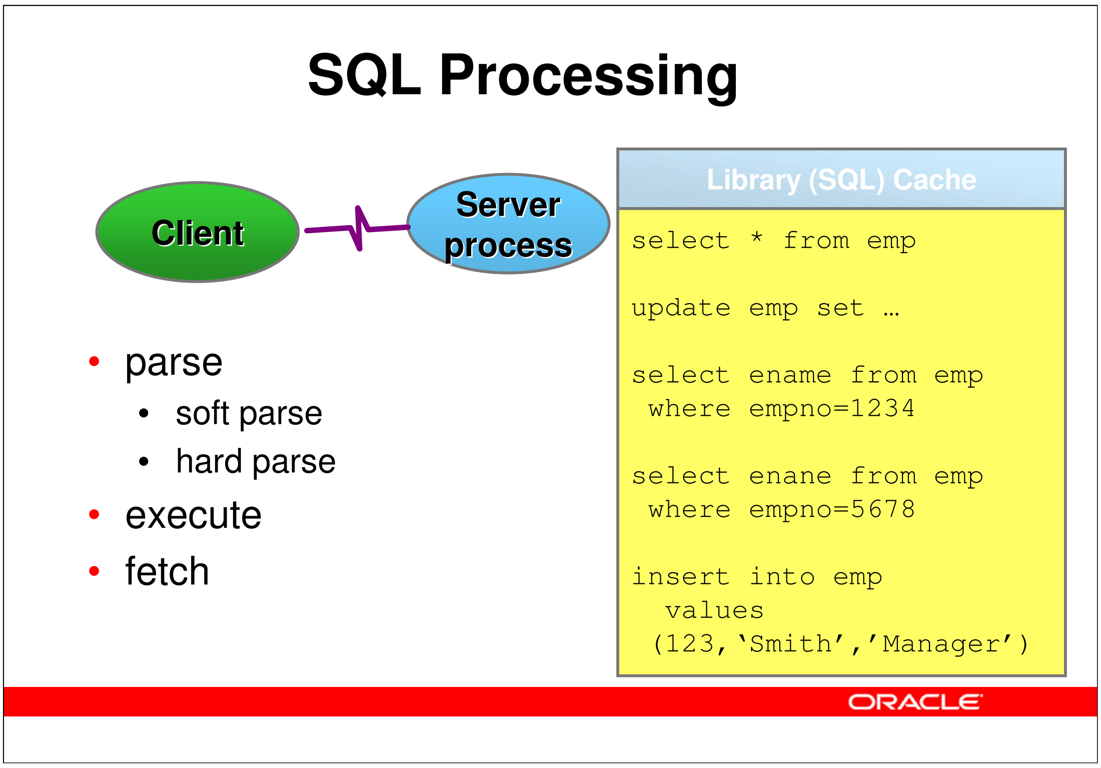
The ARCH process is only started when the database is in archive log mode. When archive log mode is in effect, you can recover from old copies of database files by applying the archived redo log files. Archive log mode also allows you to backup the database while it is running.



Agenda

- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
- Data processing
- SQL processing and client interfaces
  - How does Oracle deal with SQL?
- Summary so far
- What else?
- Hands-on: Create your own database

The SQL language is used to ”talk” to relational databases like Oracle, and the following slides show how the Oracle server processes SQL statements.



SQL Processing

```
select * from emp

update emp set …

select ename from emp
 where empno=1234

select enane from emp
 where empno=5678

insert into emp
  values
 (123,'Smith','Manager')
```

- parse
  - soft parse
  - hard parse
- execute
- fetch

Oracle SQL processing is done in generally three steps via the client/server interface; SQL statements are sent to the server, are being executed at the server and results are sent back to the client. For optimization, the number of roundtrips between the server and the client should be limited, as each roundtrip incurs a network message or a process switch if the client and the server runs on the same computer.

The parse step

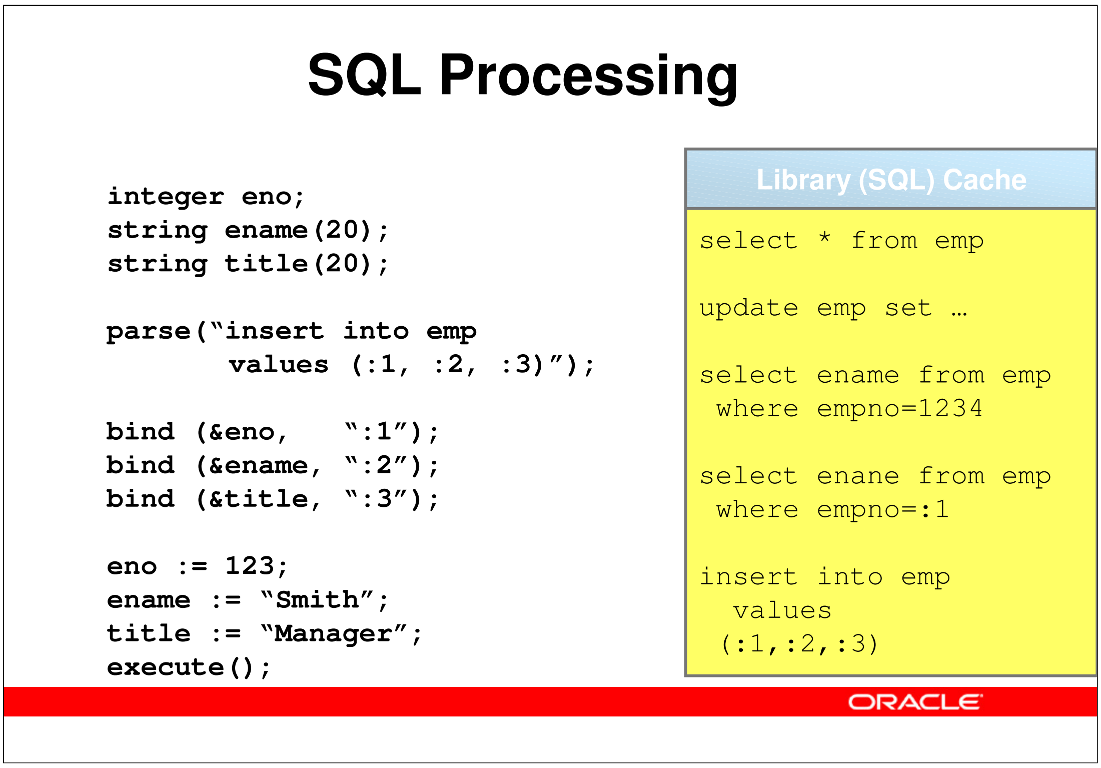
In order for a SQL statement to be executed, is must be found in the Library Cache of the SGA, and must be prepared for exeuction. This is called the parse step. The SQL statement is sent to the server, which first tries to identify an identical statement in the cache; if one is found, a soft parse is executed, which mainly does a verification of access rights. If the SQL statement is not found, a hard parse is done, which includes actual parsing and optimization of the statement. The hard parse can be time consuming, in particular for complex SQL statement, but even relatively simple SQL statements have a parse overhead, that can (and should) be avoided.

The execute step

After parsing, the client instructs the server to execute the SQL statement. This step can be repeated, which means that a SQL statement can be parsed once and executed several times, which may heavily reduce the overhead by parsing.

The fetch step

If the SQL statement is a query, the rows that are the result are sent to the client during this step.



SQL Processing

```
integer eno;
string ename(20);
string title(20);

parse(“insert into emp
       values (:1, :2, :3)”);

bind (&eno,   “:1”);
bind (&ename, “:2”);
bind (&title, “:3”);

eno := 123;
ename := “Smith”;
title := “Manager”;
execute();
```

Library (SQL) Cache

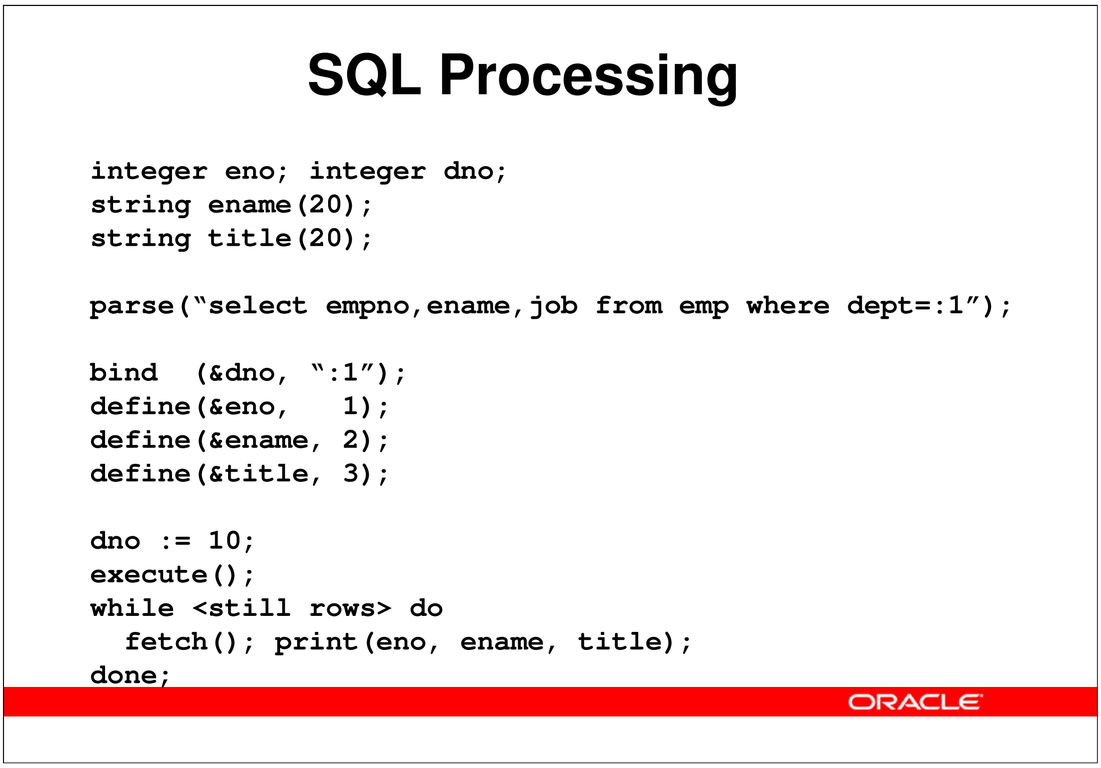
```
select * from emp

update emp set …

select ename from emp
 where empno=1234

select enane from emp
 where empno=:1

insert into emp
  values
 (:1,:2,:3)
```

ORACLE

In order to reduce the number of parse operations, all SQL statements that are being used repetitively should use placeholders, also called bind-variables, as the “:1” in the select statement in the program example above. The example shows an extra processing step:

The bind step

Any constant in SQL statements, e.g. the ‘1234’ in ‘select * from emp where empno=1234’ can, and generally should, be replaced by bind-variables, which are identified by colon followed by a number or a string, e.g. :1, :2, :abc, etc. In the client program, an actual variable is bound to this bind-variable, and the actual value of the program variable is then used at execute time. This means that the execute can be repeated with different values, but with the same SQL statement; hence, no new parse is necessary.



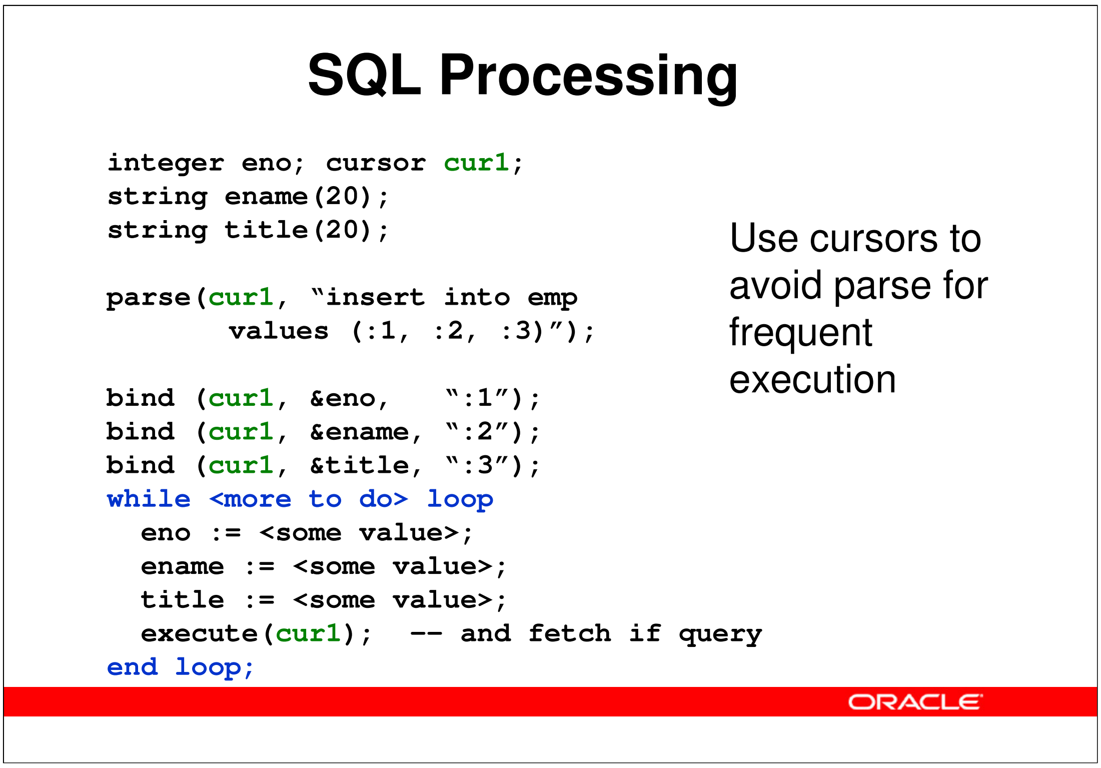
SQL Processing

```
integer eno; integer dno;
string ename(20);
string title(20);

parse(“select empno,ename,job from emp where dept=:1”);

bind  (&dno, “:1”);
define(&eno,   1);
define(&ename, 2);
define(&title, 3);

dno := 10;
execute();
while <still rows> do
  fetch(); print(eno, ename, title);
done;
```

Bind-variable can (and should) also be used with queries as in this example. Additionally, this example shows the define step:

The define step

For queries, program variables must be available to retrieve the results of the query during the fetch step. This is done by the define call, which takes program variable and the number in the select-list as arguments.



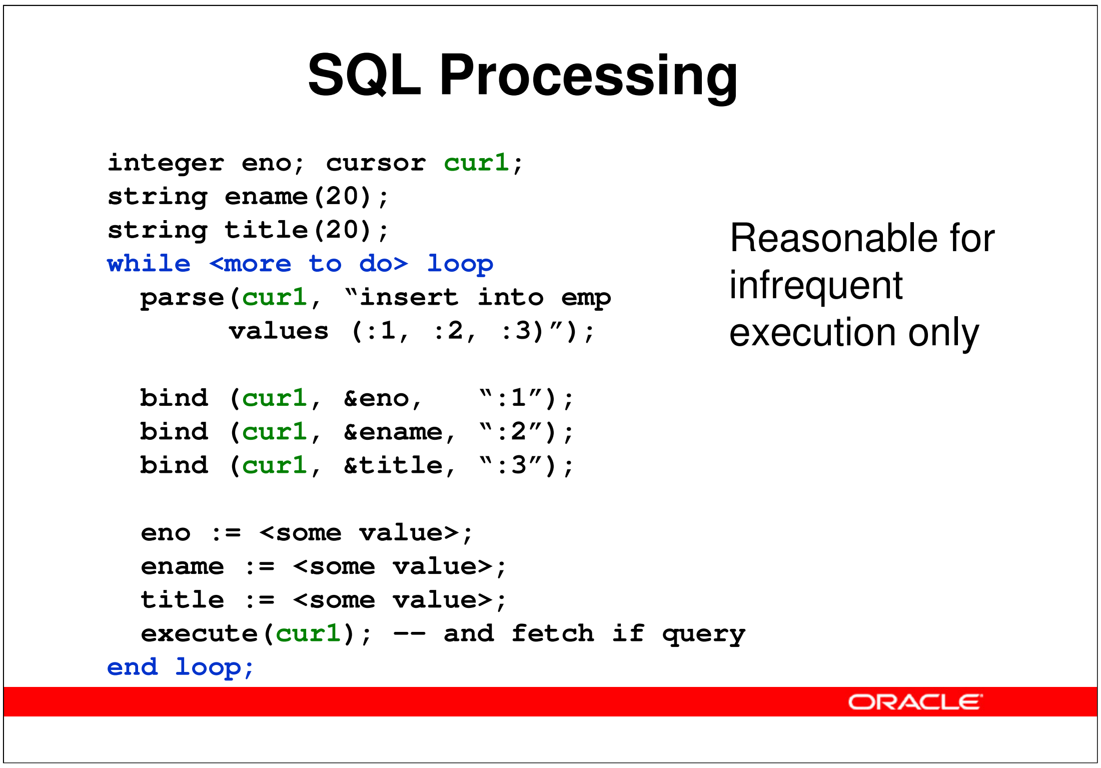
SQL Processing

```
integer eno; cursor cur1;
string ename(20);
string title(20);

parse(cur1, “insert into emp
       values (:1, :2, :3)”);

bind (cur1, &eno,   “:1”);
bind (cur1, &ename, “:2”);
bind (cur1, &title, “:3”);
while <more to do> loop
  eno := <some value>;
  ename := <some value>;
  title := <some value>;
  execute(cur1);  -- and fetch if query
end loop;
```

Use cursors to avoid parse for frequent execution

Typically, applications need to keep track of more than one SQL statement at a time. This is done using *cursors*, which can be seen as pointers to SQL statements.

In the example, a cursor is declared and associated with a SQL statement using the parse call. After being parsed, the cursor can be executed as many times as necessary, which avoids any overhead of the parse call.

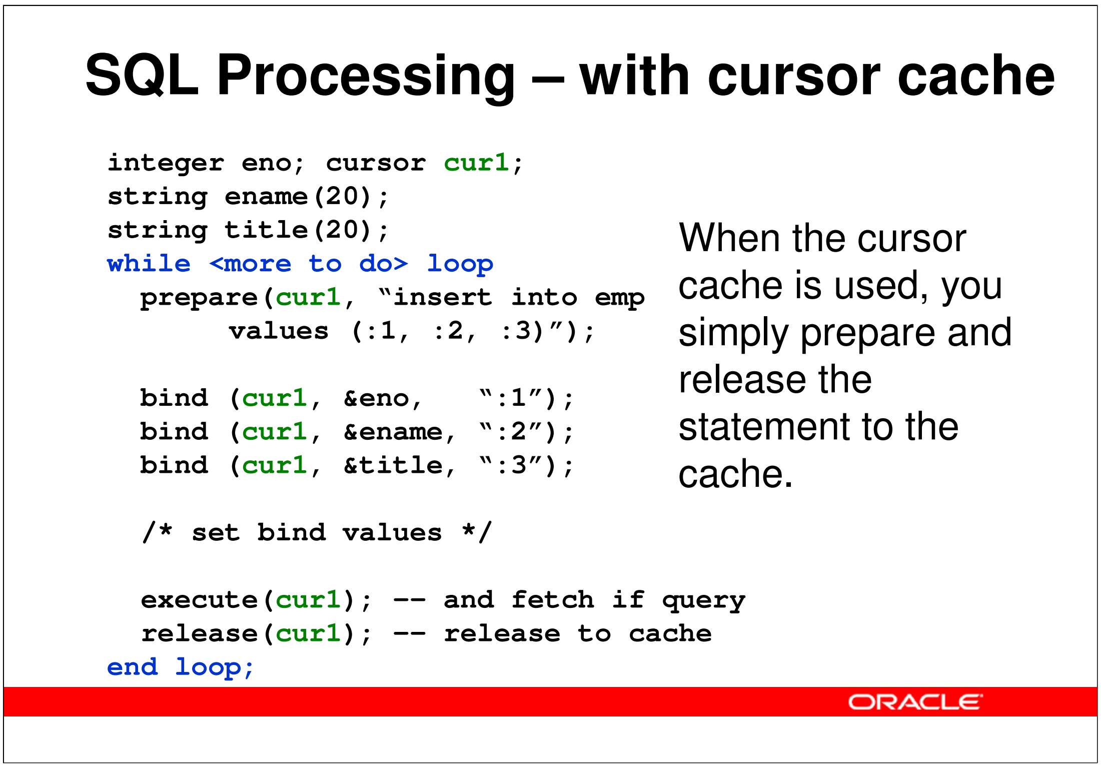


SQL Processing

```
integer eno; cursor cur1;
string ename(20);
string title(20);
while <more to do> loop
  parse(cur1, “insert into emp
        values (:1, :2, :3)”);

  bind (cur1, &eno,   “:1”);
  bind (cur1, &ename, “:2”);
  bind (cur1, &title, “:3”);

  eno := <some value>;
  ename := <some value>;
  title := <some value>;
  execute(cur1); -- and fetch if query
end loop;
```

Reasonable for infrequent execution only

In this case, only the first parse will be a *hard* parse, i.e. the Oracle server will have to do a full parse for syntax and data dictionary information. Each subsequent execution only does a *soft* parse, which primarily performs a verification of access rights. However, even the soft parse has an overhead, so the above code should only be used if the SQL statement is infrequently executed.



SQL Processing – with cursor cache

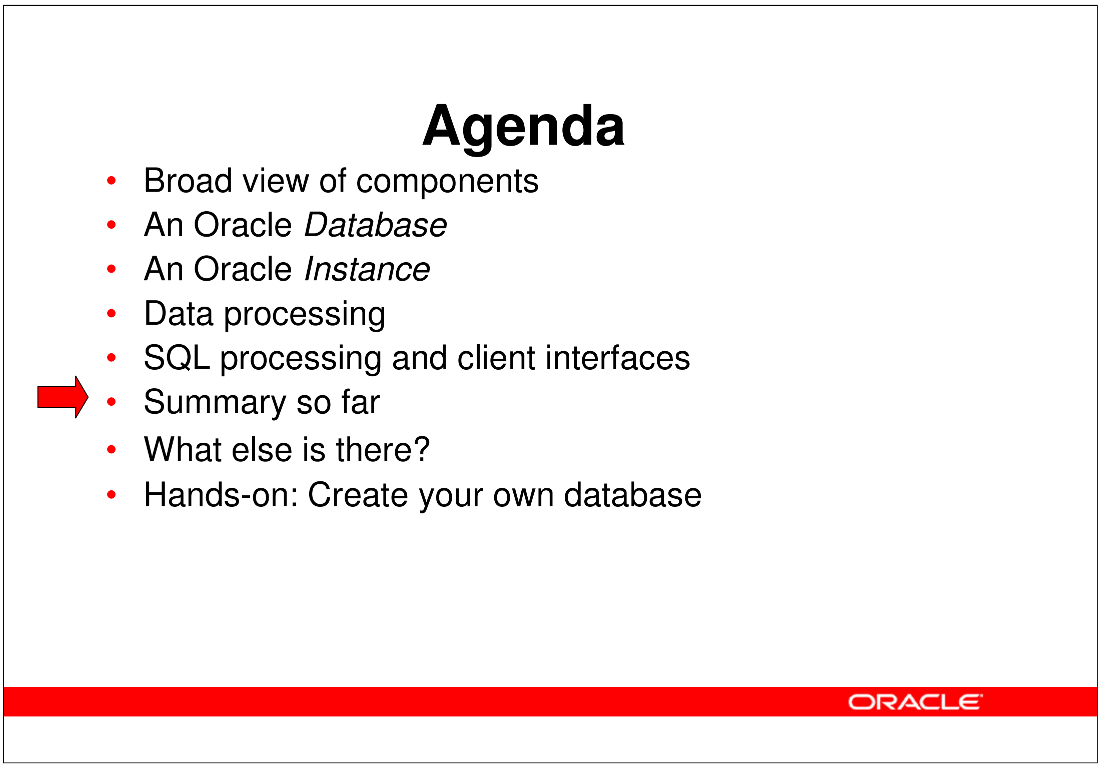
```
integer eno; cursor cur1;
string ename(20);
string title(20);
while <more to do> loop
  prepare(cur1, “insert into emp
        values (:1, :2, :3)”);

  bind (cur1, &eno,   “:1”);
  bind (cur1, &ename, “:2”);
  bind (cur1, &title, “:3”);

  /* set bind values */

  execute(cur1); -- and fetch if query
  release(cur1); -- release to cache
end loop;
```

When the cursor cache is used, you simply prepare and release the statement to the cache.

In all modern API’s, including Oracle’s own, you can use a prepare/release set of calls in stead of explicitly performing the parse.

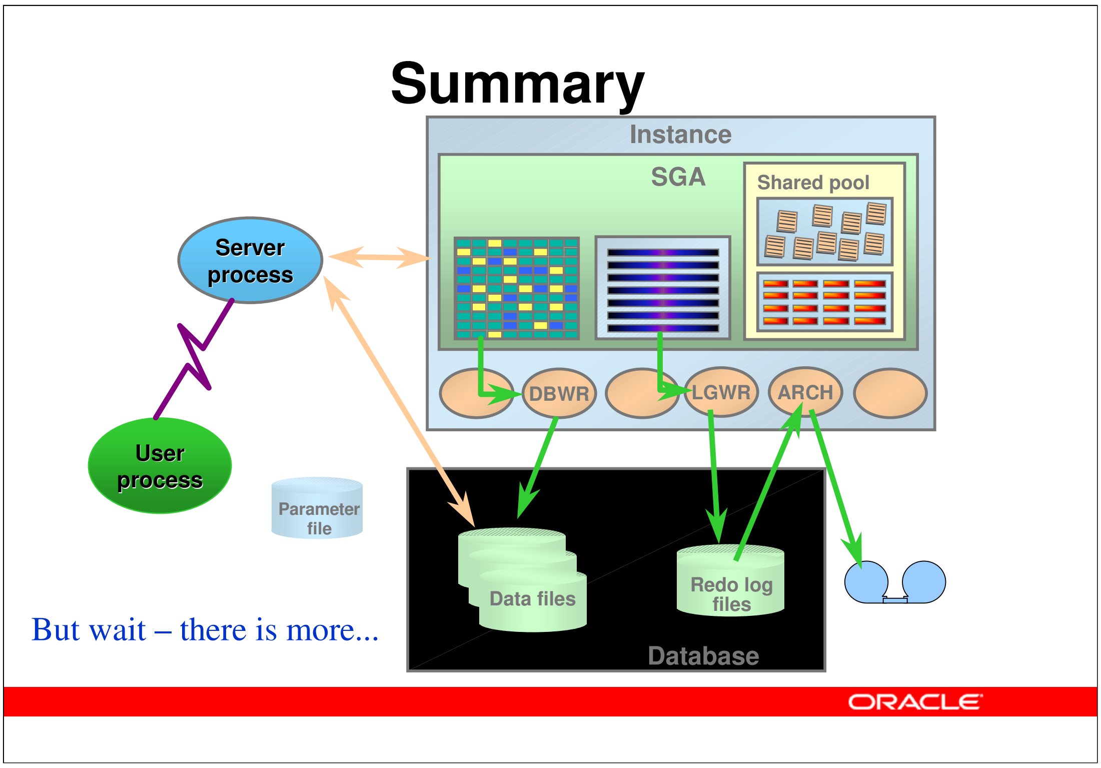
The prepare call may do a parse – and will do a parse first time it is executed – but after having used the release call, the API will simply put the cursor into a local cache, ready to be picked up during the next prepare of the identically same SQL statement.

This approach should always be used for new development.



Agenda

- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
- Data processing
- SQL processing and client interfaces
- Summary so far
- What else is there?
- Hands-on: Create your own database



Summary

But wait – there is more...

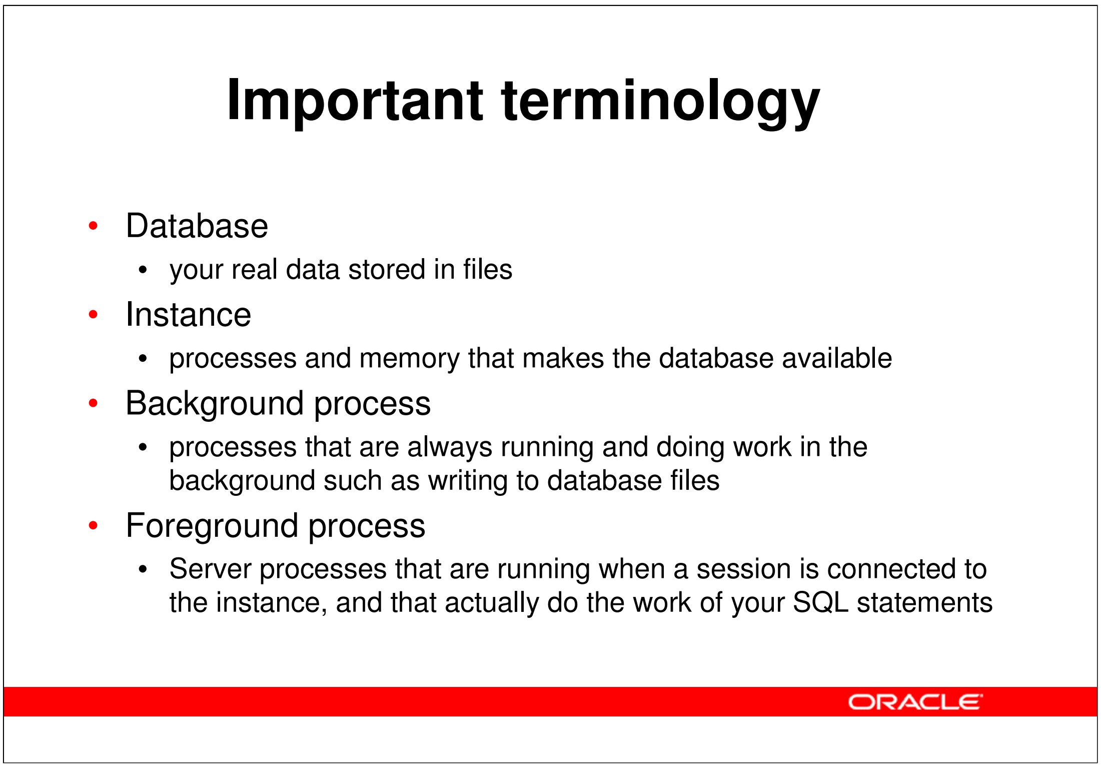
Understanding of the slides so far will assist in understanding many other Oracle concepts. This is a brief overview:

•An Oracle server is centered around the shared memory area, SGA (System Global Area), that among other things contains a cache of database blocks, a cache of SQL statements, and a cache of information about tables, columns, etc.

•The Oracle database is your real data, stored on disk. Important extra information, such as the redo log used for recovery and other purposes is also stored on disk.

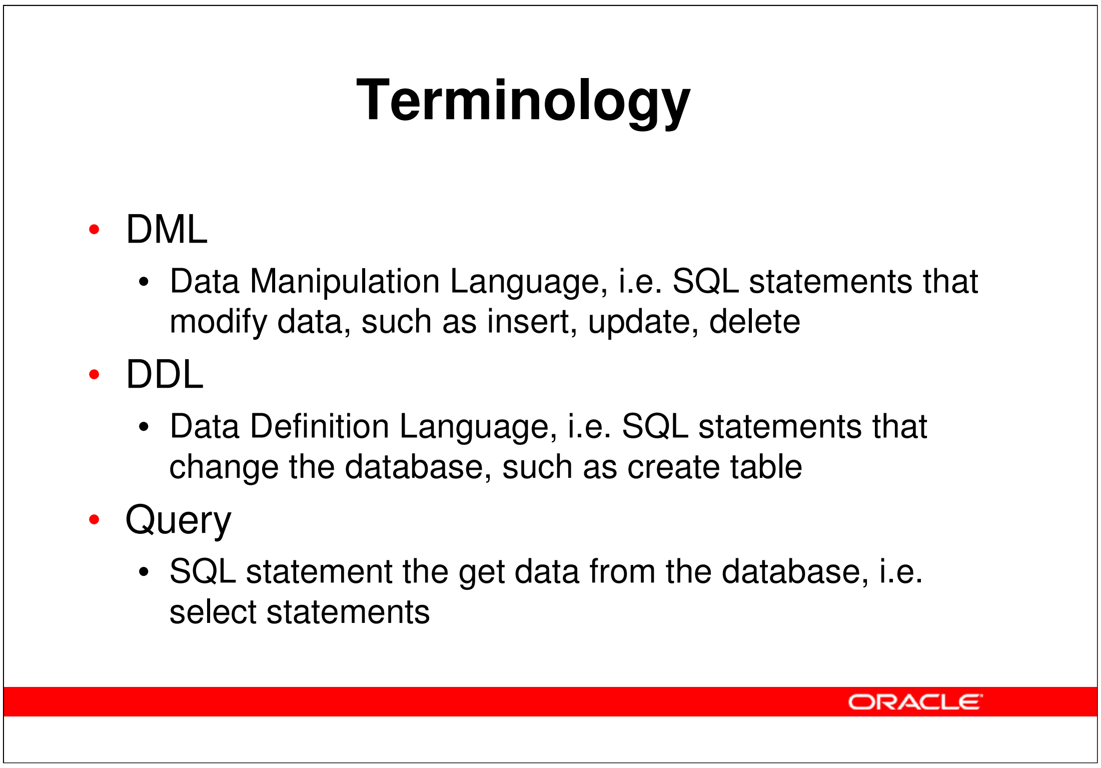
•A number of background processes, each with a specific purpose, are running. Examples are processes responsible for writing data to the database files or the redo log files. The actual set of background processes depends on various configuration settings, and newer Oracle versions typically have more background processes.

•Each client or user process connected to the instance is served by a server process; these are occasionally called foreground processes to distinguish them from background processes.



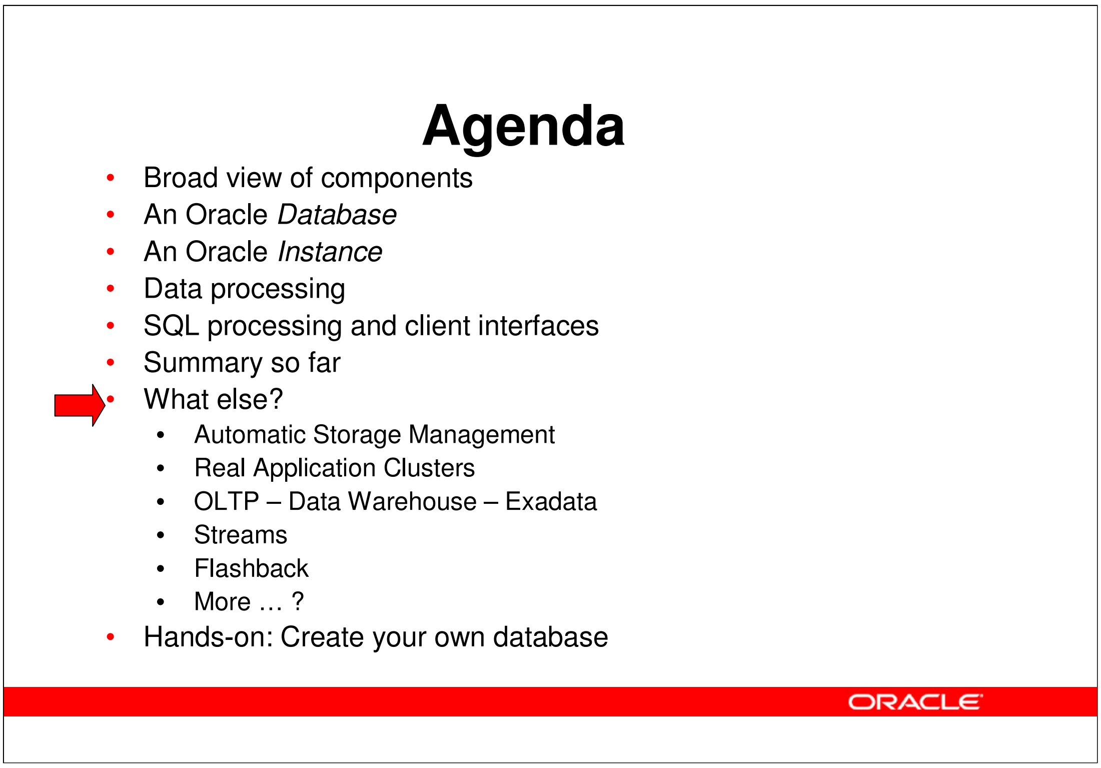
Important terminology

- Database
  - your real data stored in files
- Instance
  - processes and memory that makes the database available
- Background process
  - processes that are always running and doing work in the background such as writing to database files
- Foreground process
  - Server processes that are running when a session is connected to the instance, and that actually do the work of your SQL statements



Terminology

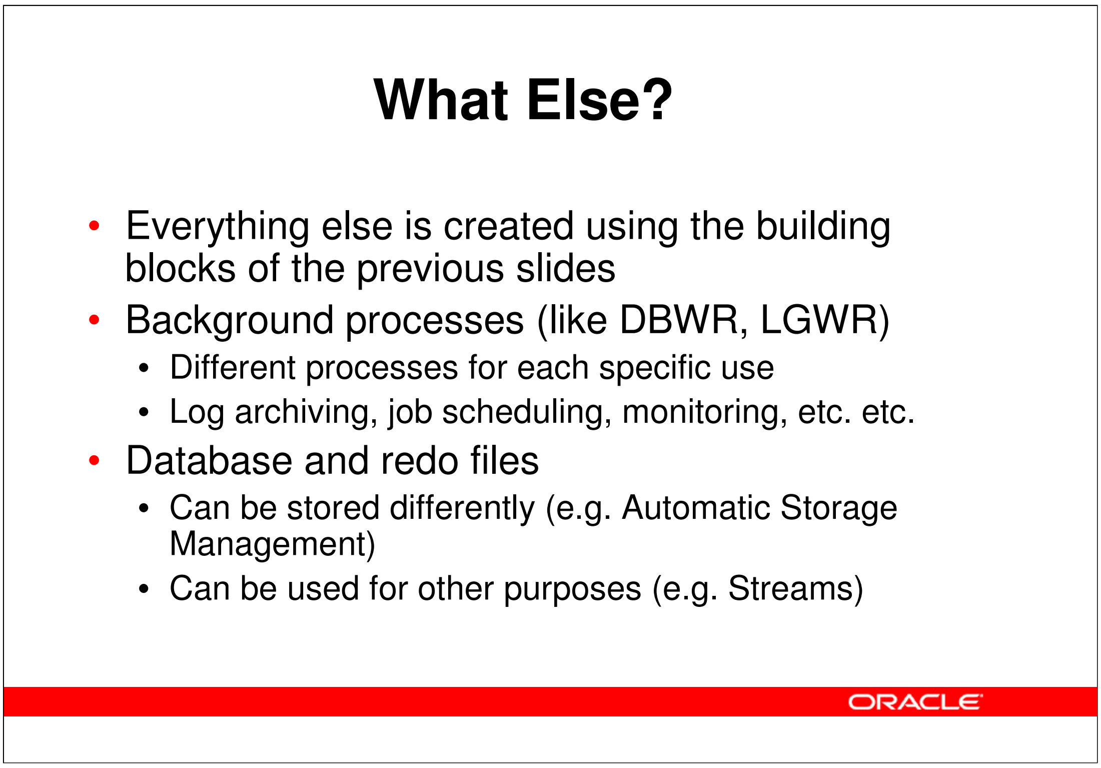
- DML
  - Data Manipulation Language, i.e. SQL statements that modify data, such as insert, update, delete
- DDL
  - Data Definition Language, i.e. SQL statements that change the database, such as create table
- Query
  - SQL statement the get data from the database, i.e. select statements



Agenda

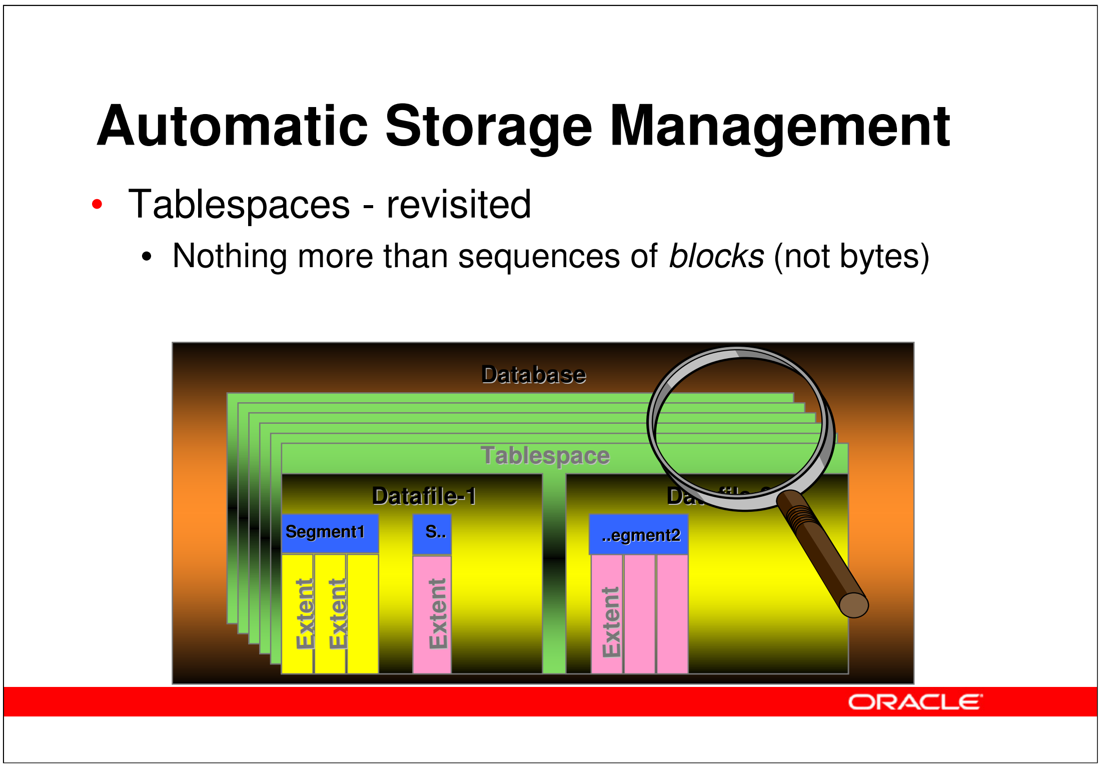
- Broad view of components
- An Oracle *Database*
- An Oracle *Instance*
- Data processing
- SQL processing and client interfaces
- Summary so far
- What else?
  - Automatic Storage Management
  - Real Application Clusters
  - OLTP – Data Warehouse – Exadata
  - Streams
  - Flashback
  - More … ?
- Hands-on: Create your own database

The following slides will list some further Oracle features, and explain how they are built



What Else?

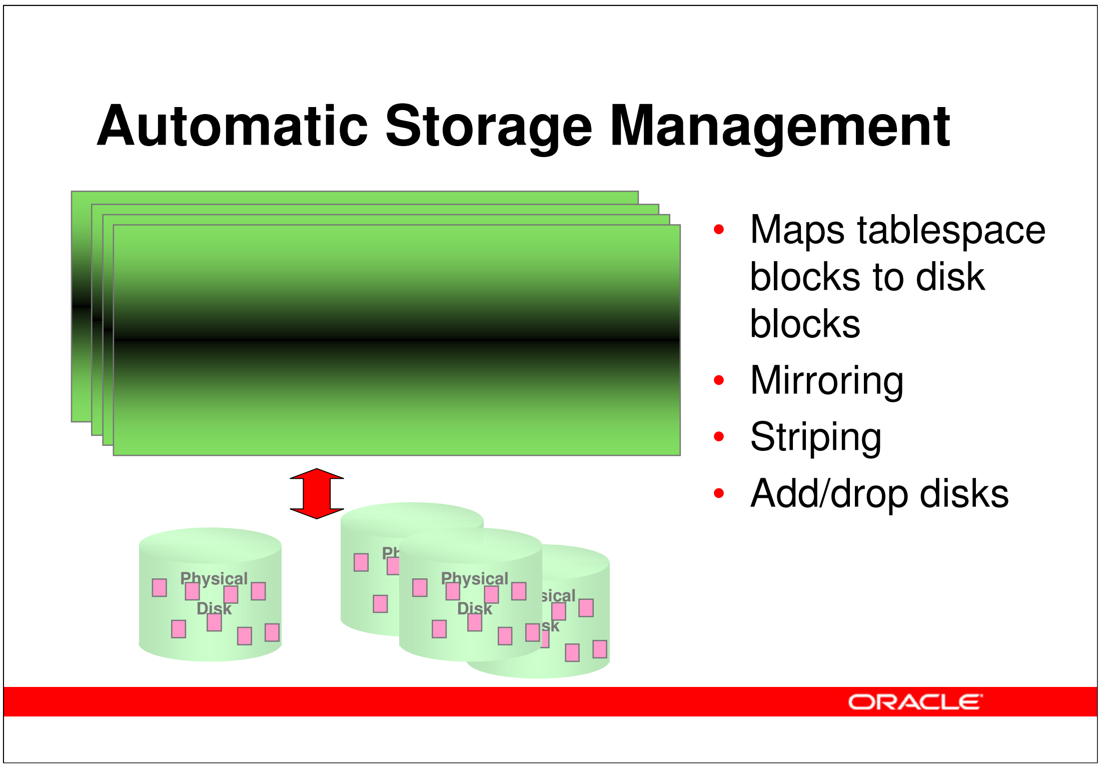
- Everything else is created using the building blocks of the previous slides
- Background processes (like DBWR, LGWR)
  - Different processes for each specific use
  - Log archiving, job scheduling, monitoring, etc. etc.
- Database and redo files
  - Can be stored differently (e.g. Automatic Storage Management)
  - Can be used for other purposes (e.g. Streams)



Automatic Storage Management

- Tablespaces - revisited
  - Nothing more than sequences of *blocks* (not bytes)

Tablespaces are nothing more than sequences of blocks and do therefore not require all the advances features of modern file systems. In particular Oracle does not need the file system to support files of any size or to support locking or access control.



Automatic Storage Management

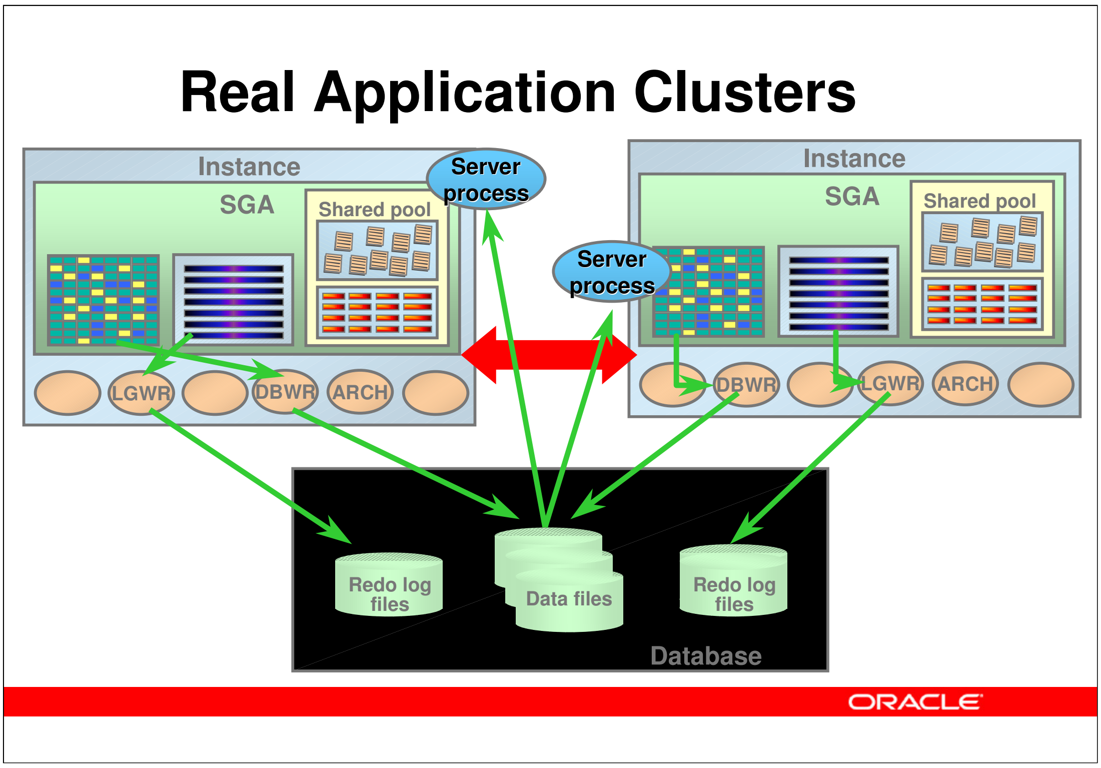
- Maps tablespace blocks to disk blocks
- Mirroring
- Striping
- Add/drop disks

Automatic Storage Management (ASM) is a volume manager for Oracle and it can also be viewed as a simplified file system.

•Blocks in tablespaces are mapped to blocks on disk

•As a volume manager, ASM does everything you would expect from a volume manager, including mirroring and striping.

•As it also knows about Oracle, it can transparently support physical modifications, such as add (or dropping) disks, and will re-distribute actual database blocks.



Real Application Clusters

In Real Application Clusters, there are multiple instances (normally one per database server node) serving a single database. The single database is stored on a set of disks, that are *shared* between all instances.

The slide shows some of the characteristics of Real Application Clusters:

•All files are shared between all instances, this includes database, redo log, parameter files, etc. There is only one set of database files, i.e. there is a single database, that just can be seen from multiple nodes/instances. Each instance actually has it's own set of redo log files, but they are shared and visible to all instances. This is e.g. used in case of recovery after loss of an instance.

•The instances are on separate nodes and do therefore not share any memory. All coordination between the instances take place of the *interconnect* (thick, red arrow).

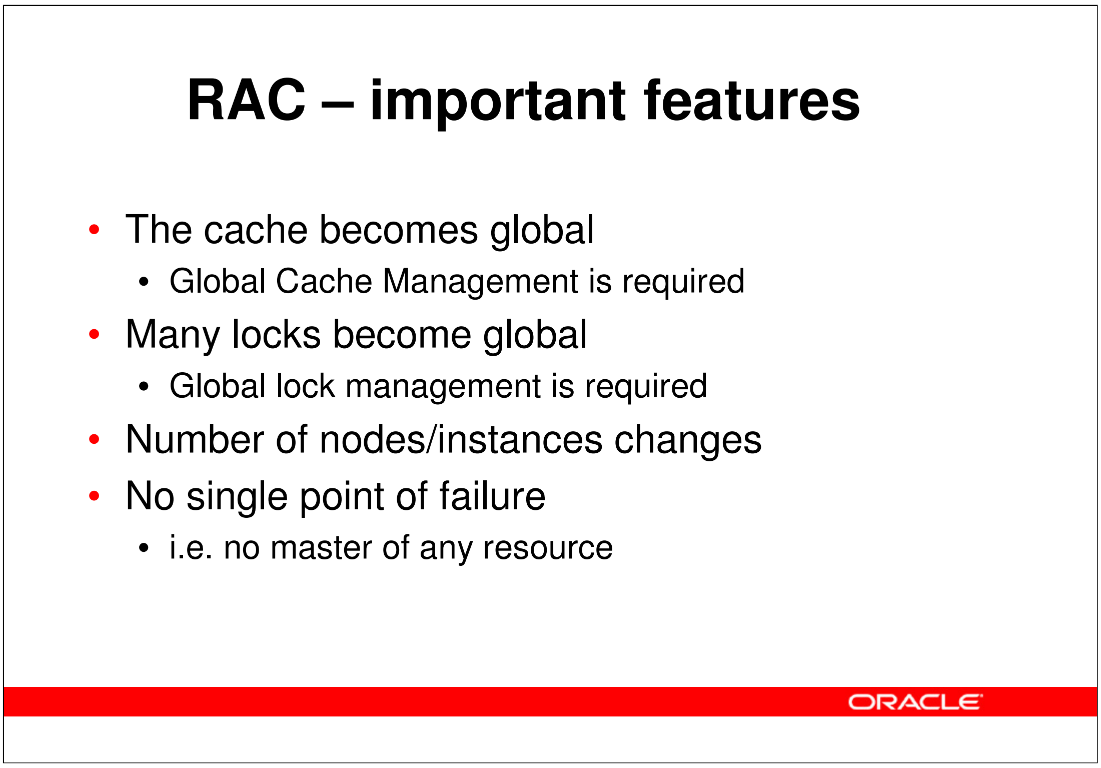
•A client and its associated server process can connect to any instance to get access to the database.

•The picture only shows two instances; many more can be configured.

There are some requirements to the hardware to support Real Application Clusters:

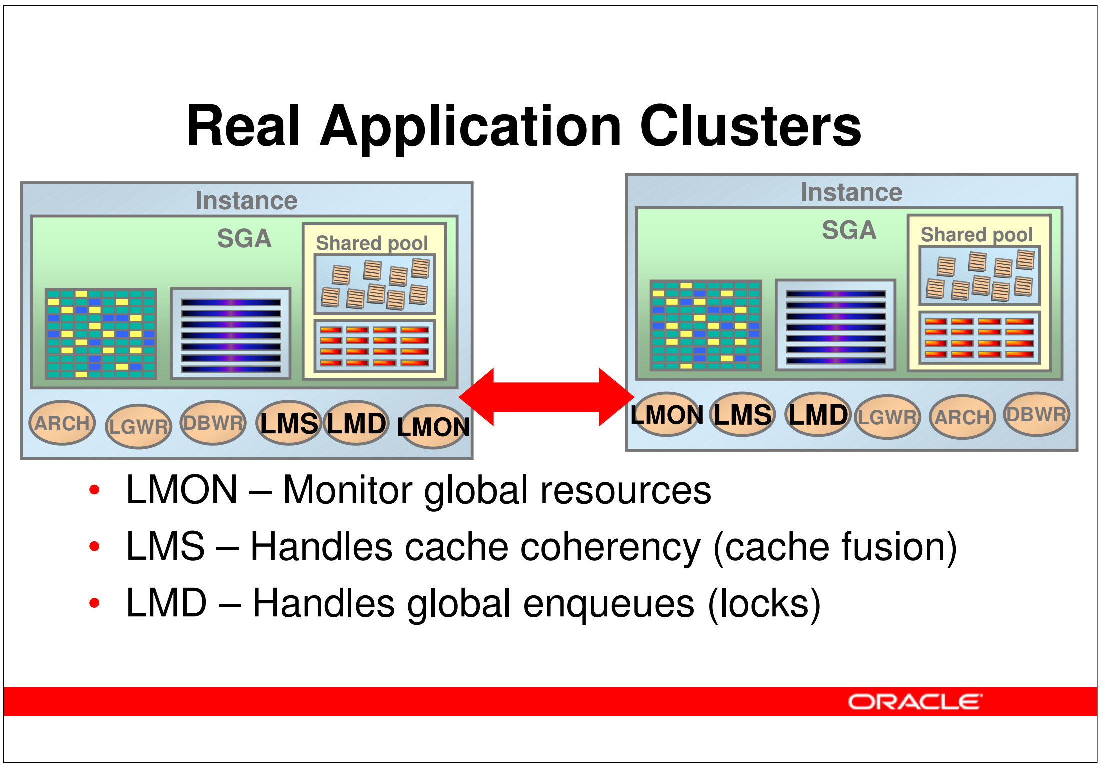
•A shared disk storage must be available; typically this involves using a SAN.

•A high speed interconnect must be available solely for the inter-node communication. This could e.g. be a Gigabit Ethernet.



RAC – important features

- The cache becomes global
  - Global Cache Management is required
- Many locks become global
  - Global lock management is required
- Number of nodes/instances changes
- No single point of failure
  - i.e. no master of any resource



Real Application Clusters

- LMON – Monitor global resources
- LMS – Handles cache coherency (cache fusion)
- LMD – Handles global enqueues (locks)

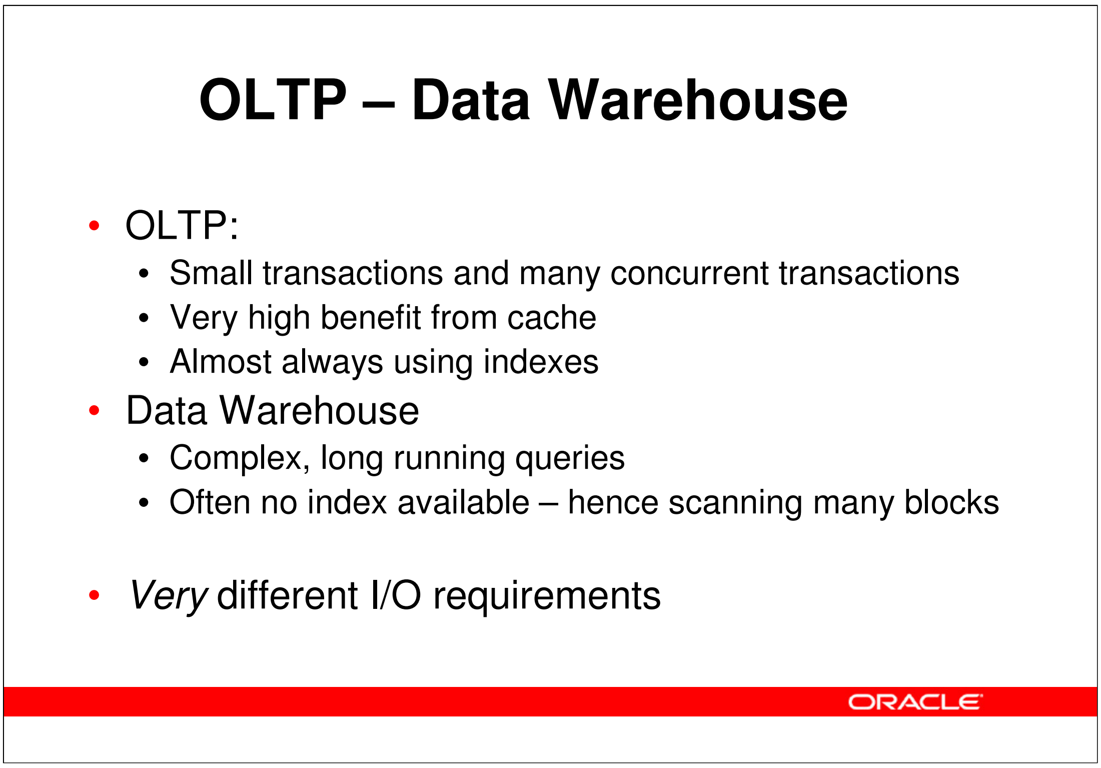
In Real Application Clusters, extra background processes are started. Some of the important ones of these are shown on this slide:

•LMON is responsible for monitoring of all global resources, i.e. global cache resources and global enqueue (lock) resources. Whenever a node is added or removed from the cluster, the LMON process will distribute the global resources on all nodes. In case of a loss of a node, this involves recovery of global resources from the surviving nodes.

•The LMSn processes actually implement the Oracle cache fusion mechanism. This makes sure only one node is modifying a block at any one time, it will send dirty blocks between instances if multiple instances have block modifications to perform, and it will construct and ship read-consistent blocks if one instance need to read a block, that has been modified by another instance.

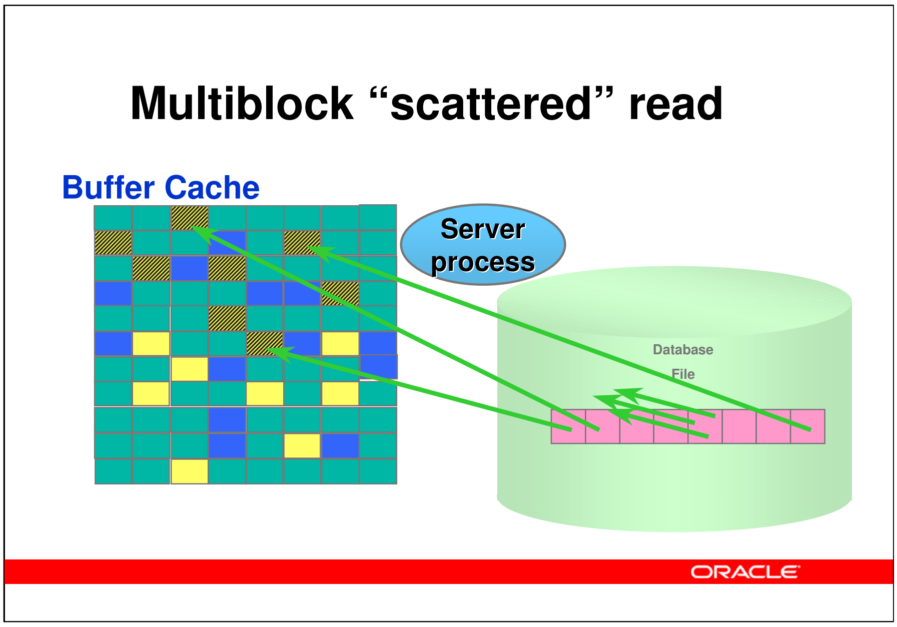
•The LMDn processes make the enqueues (e.g. row locks) – that in the non-RAC case is handled internally on one instance – global, so that they can be seen by all instances.

There are more processes in addition to these.



OLTP – Data Warehouse

- OLTP:
  - Small transactions and many concurrent transactions
  - Very high benefit from cache
  - Almost always using indexes
- Data Warehouse
  - Complex, long running queries
  - Often no index available – hence scanning many blocks
- *Very* different I/O requirements



Multiblock “scattered” read

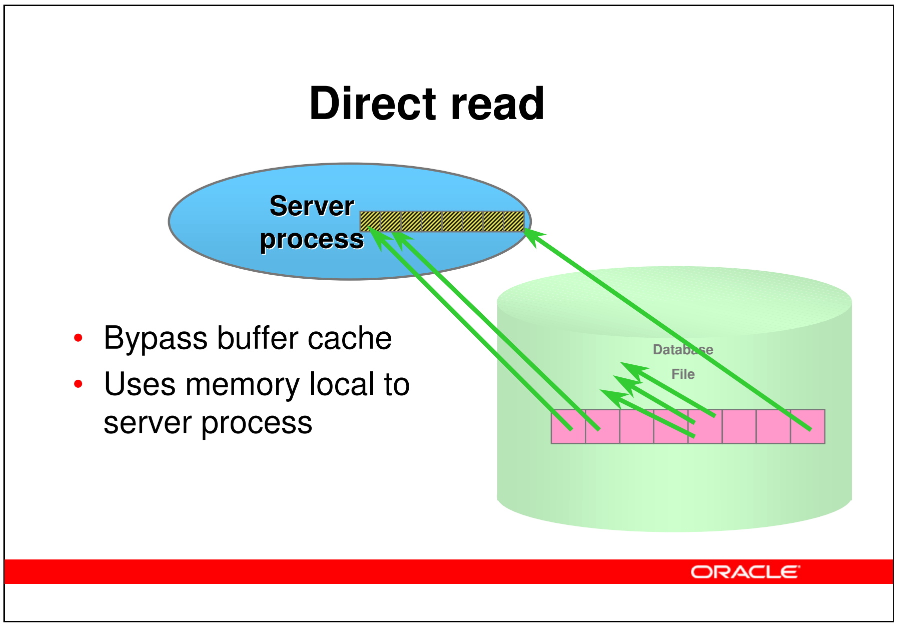
Buffer Cache

Server process

Database File

The process described earlier for reading blocks into the buffer cache is very efficient for typical OLTP applications, where it is likely that many concurrent users will benefit from the cache shared by all. As an example, branch blocks of indexes are very frequently read.

If you need to *scan* data in the database – which is what data warehouse applications quite typically do, reading one block at a time is inefficient. In stead, you should read multiple blocks at the time. This is what Oracle calls a “scattered” read and the term comes from the fact that the blocks (which are consecutive on disk) are read into *scattered* blocks in the buffer cache.



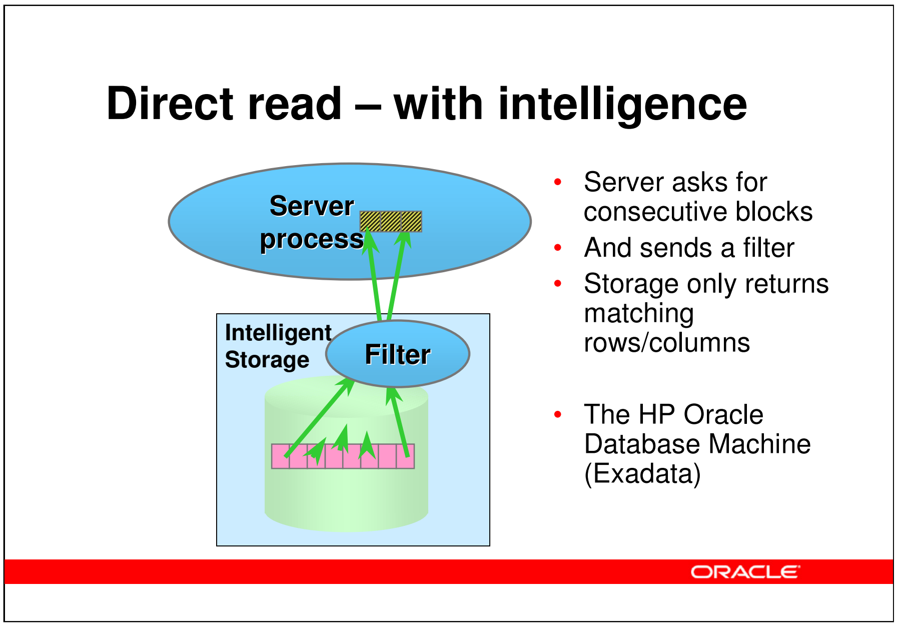
Direct read

Server process

Database File

- Bypass buffer cache
- Uses memory local to server process

The next step in scanning is to avoid the buffer cache completely. As you rarely have benefit from the sharing in the buffer cache with data warehouse type applications, a much more efficient *direct* read can be done from the database file to buffers allocated in the server process. Not only do you bypass the buffer cache overhead, you can also make the actual read more efficient by read consecutive blocks from disk to consecutive blocks in memory.

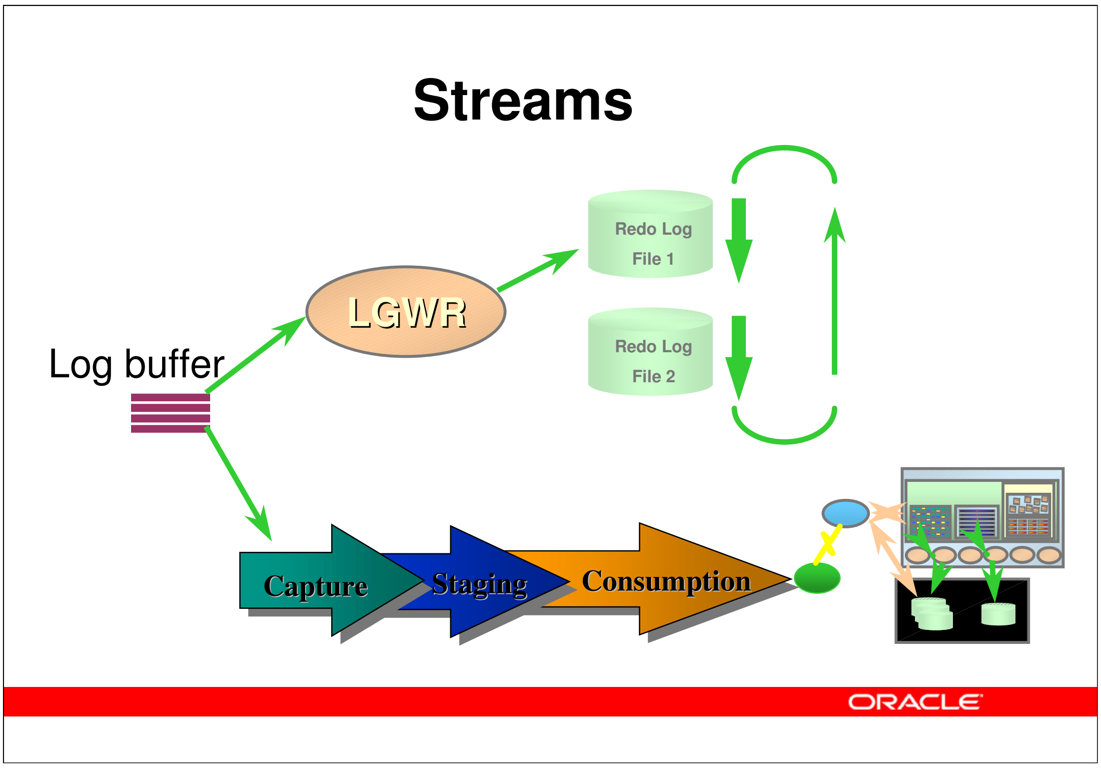


Direct read – with intelligence

- Server asks for consecutive blocks
- And sends a filter
- Storage only returns matching rows/columns
- The HP Oracle Database Machine (Exadata)

In the HP Oracle database machine (Exadata), the storage has been made intelligent by making it understand part of the SQL language. In particular it can filter rows by typical criteria (such as "columnA = 123") and it can select only a subset of columns from each row. The net result is that the server process receives blocks that contain exactly the information needed. On the previous two slides, the blocks processed by the server will typically include both rows and columns that are not needed by the application request.

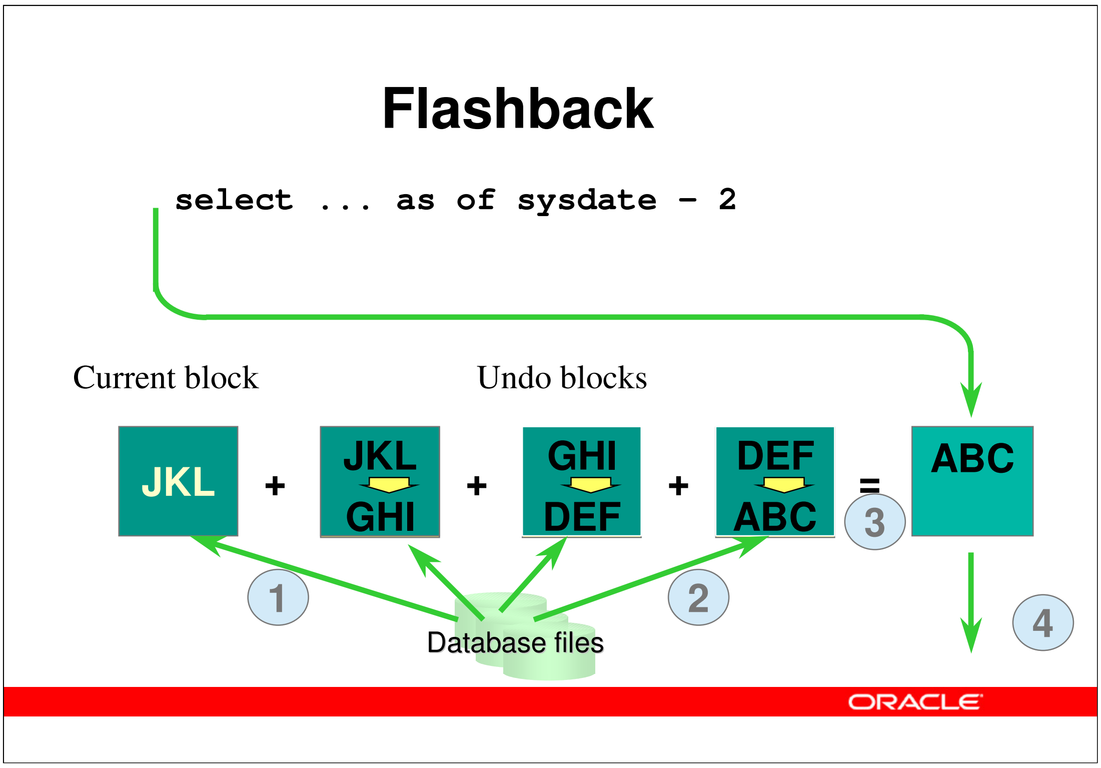
Effectively, this means data warehouses measured in TB's become fully practically useful as queries previously running in minutes or even hours now run in seconds or minutes.



Streams

As we have seen earlier, the log buffer contains a detailed record of all changes to the database, and is necessary for recovery. Oracle allows you to use the log for other purposes as well – namely using the streams feature.

This is another example of how the basic architecture is augmented with extra features. With Oracle Streams, the contents of the log buffer is captured, possibly staged, and forwarded for consumption at some other place. In it's most simple case, this will implement replication, as all changes done to one database can be copied to another database and applied at that other database.



Flashback

```
select ... as of sysdate - 2
```

Current block

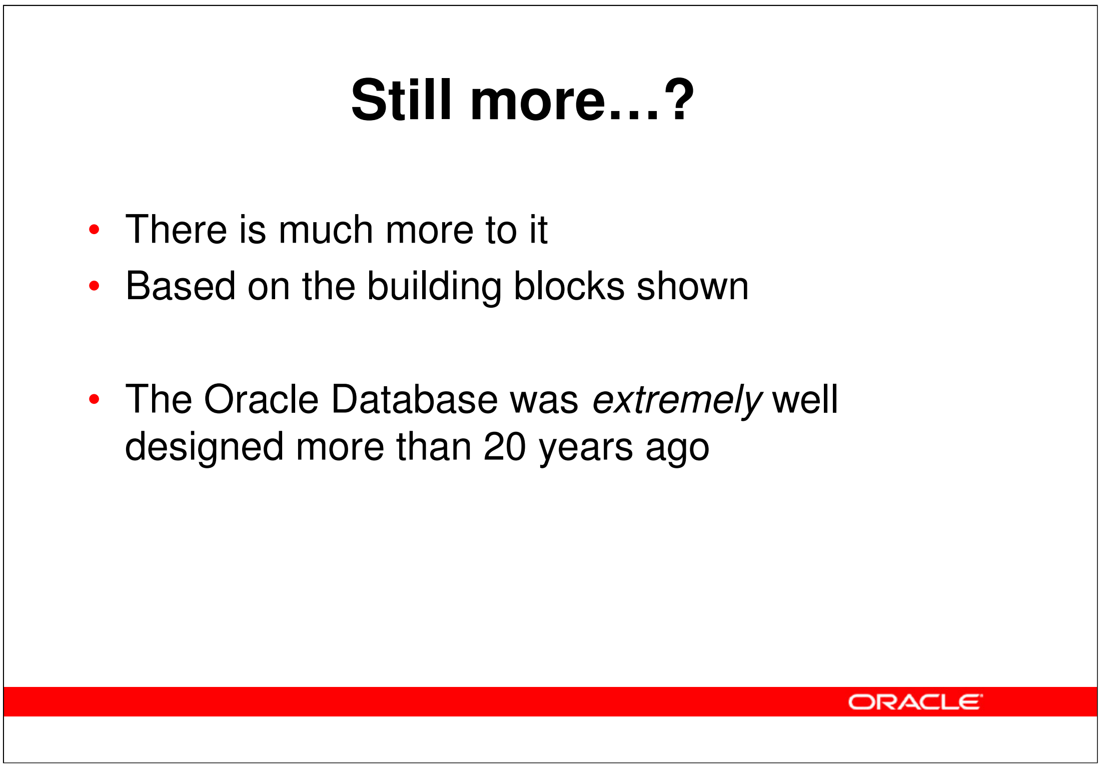
Undo blocks

JKL + JKL → GHI + GHI → DEF + DEF → ABC = ABC

The undo information that has been stored in the undo tablespaces on disk can also be used to construct (very) old views of data – this is what called *flashback.* Assume you would like to see what a row looked like to days ago, and you issue the query shown.

1. The most recent version of the block is read from disk (if not already in the buffer cache)
2. The sequence of undo blocks – in reverse order of generation – are being read from the undo tablespaces and are being applied
3. After applying all undo, the block as it looked like at a previous point in time is found

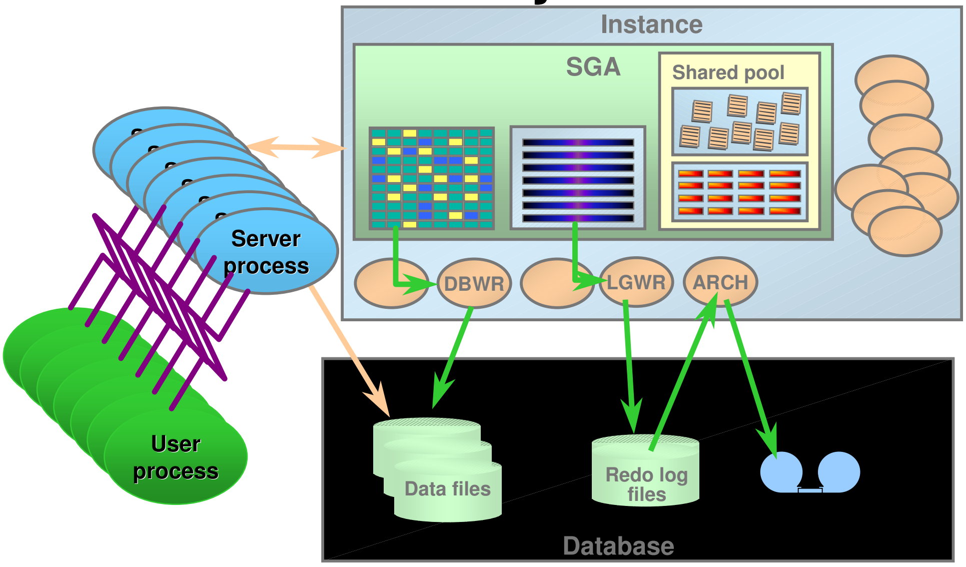
4. The query uses this consistent read copy of the block and continues.

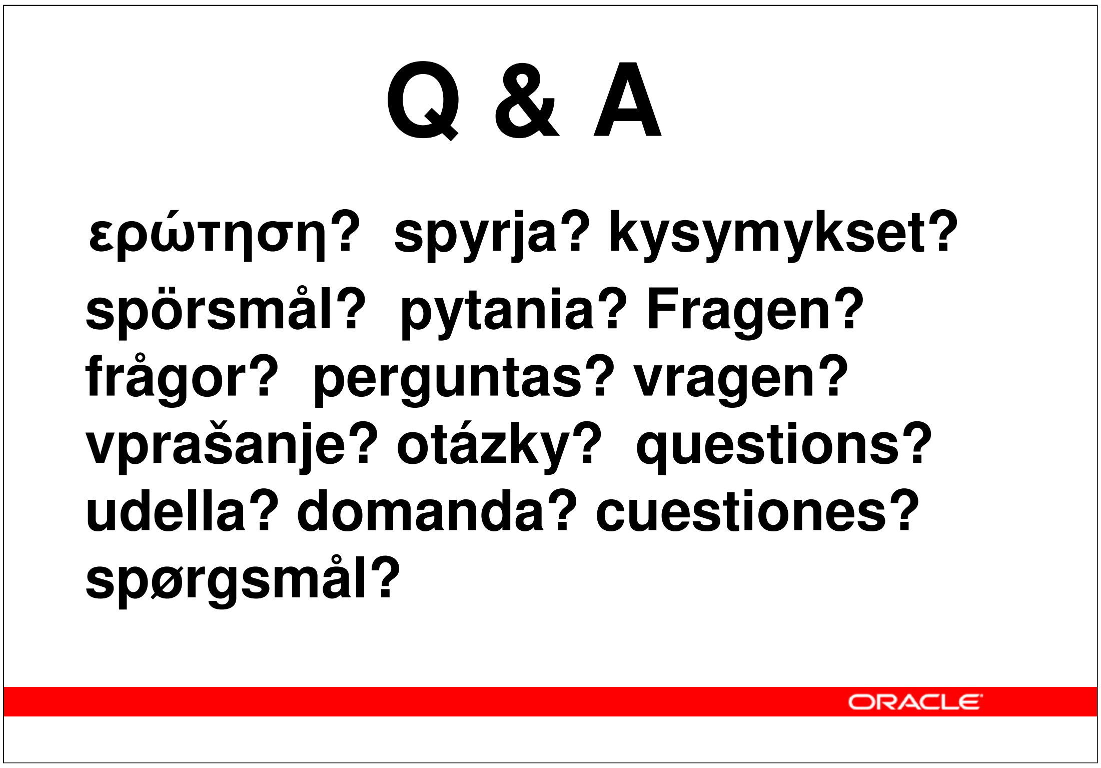


Still more…?

- There is much more to it
- Based on the building blocks shown
- The Oracle Database was *extremely* well designed more than 20 years ago

Summary


Instance
SGA
Shared pool
Server process
User process
DBWR
LGWR
ARCH
Data files
Redo log files
Database



Q & A

ερώτηση? spyrja? kysymykset?
spörsmål? pytania? Fragen?
frågor? perguntas? vragen?
vprašanje? otázky? questions?
udella? domanda? cuestiones?
spørgsmål?

If you find that the word “Questions?” on this slide in *your* language is spelled incorrectly or poorly chosen, of if the word in your language is missing, please email me at bjorn.engsig@oracle.com. And then next time you see one of my presentations, you may see the word correctly in your language!