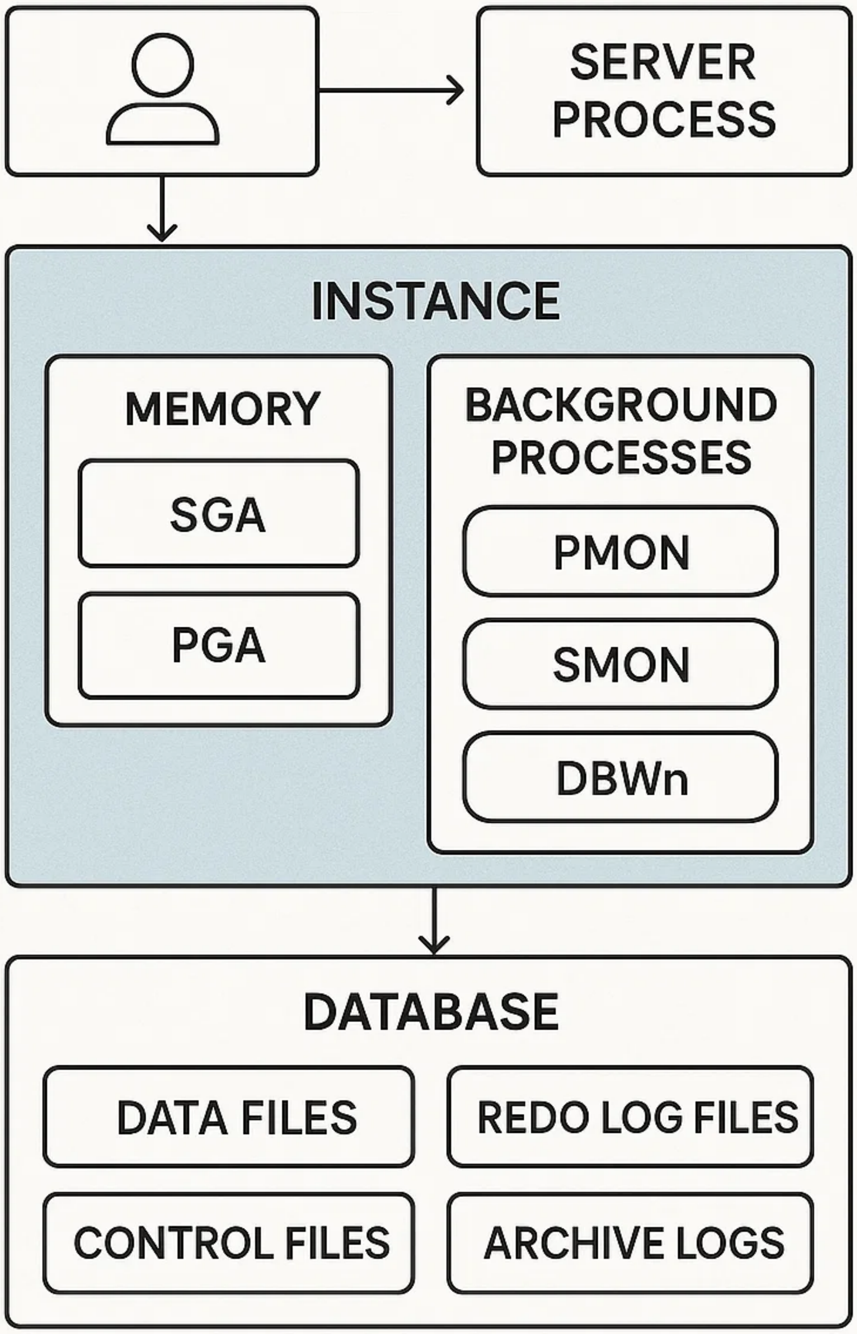
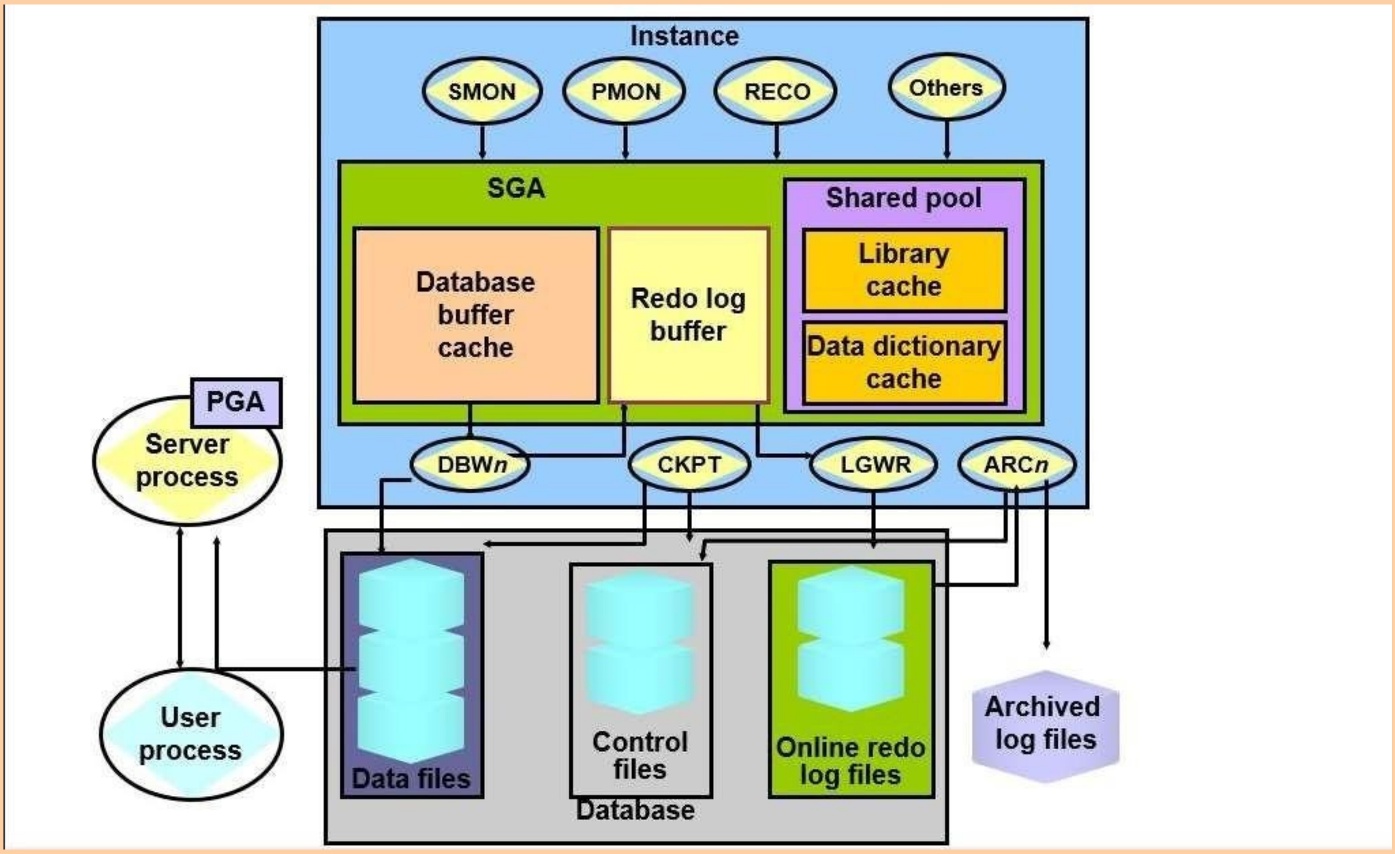
ORACLE DATABASE ARCHITECTURE

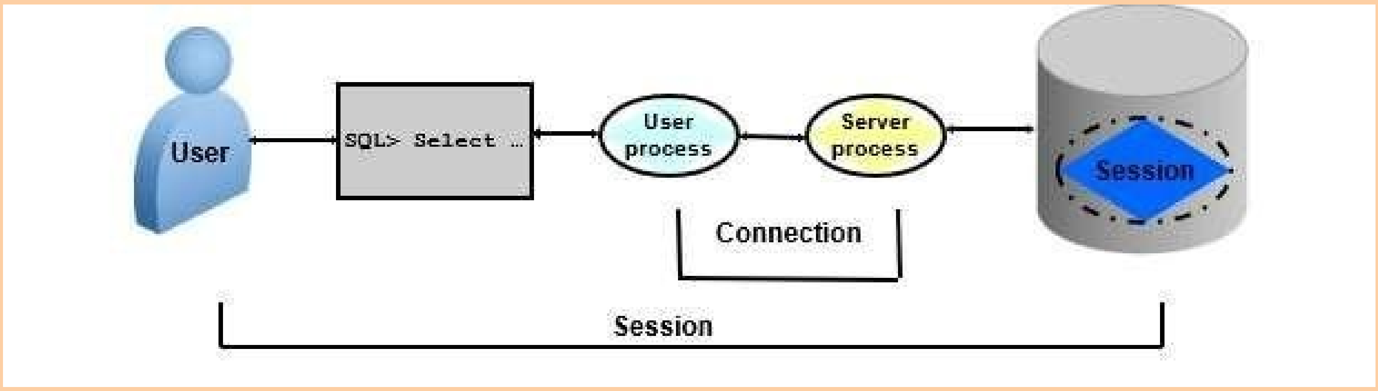


Oracle Database Architecture : Overview



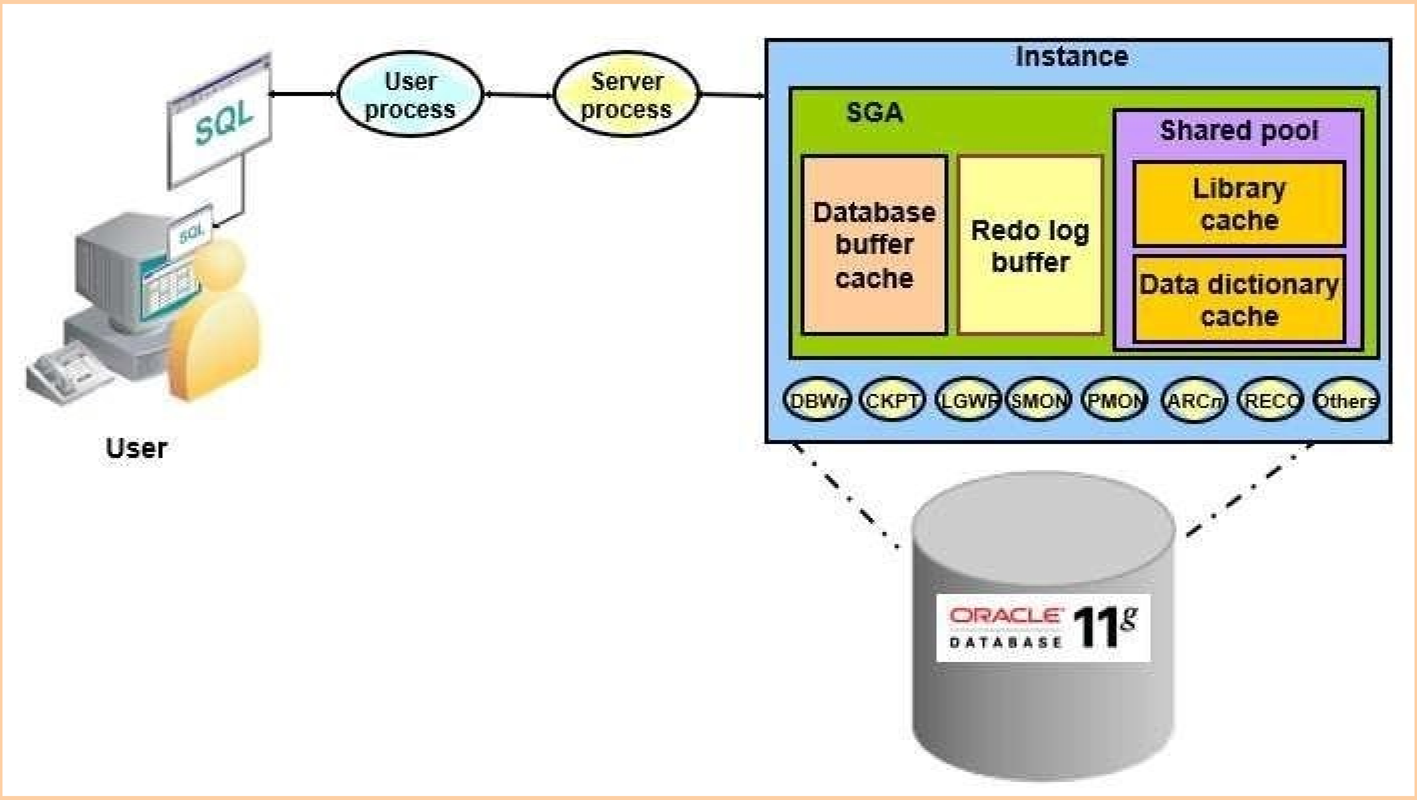
- It consists of an instance and its associated databases.
- The instance contains memory structures and background processes.
- Whenever an instance is started, the shared memory area ,System Global Area (SGA) is allocated, and the background processes are started.
- The database has physical structures and logical structures.
- The physical and logical structures are separate. So, the physical storage of data can be managed without affecting access to logical storage structures.

Connecting to the Database



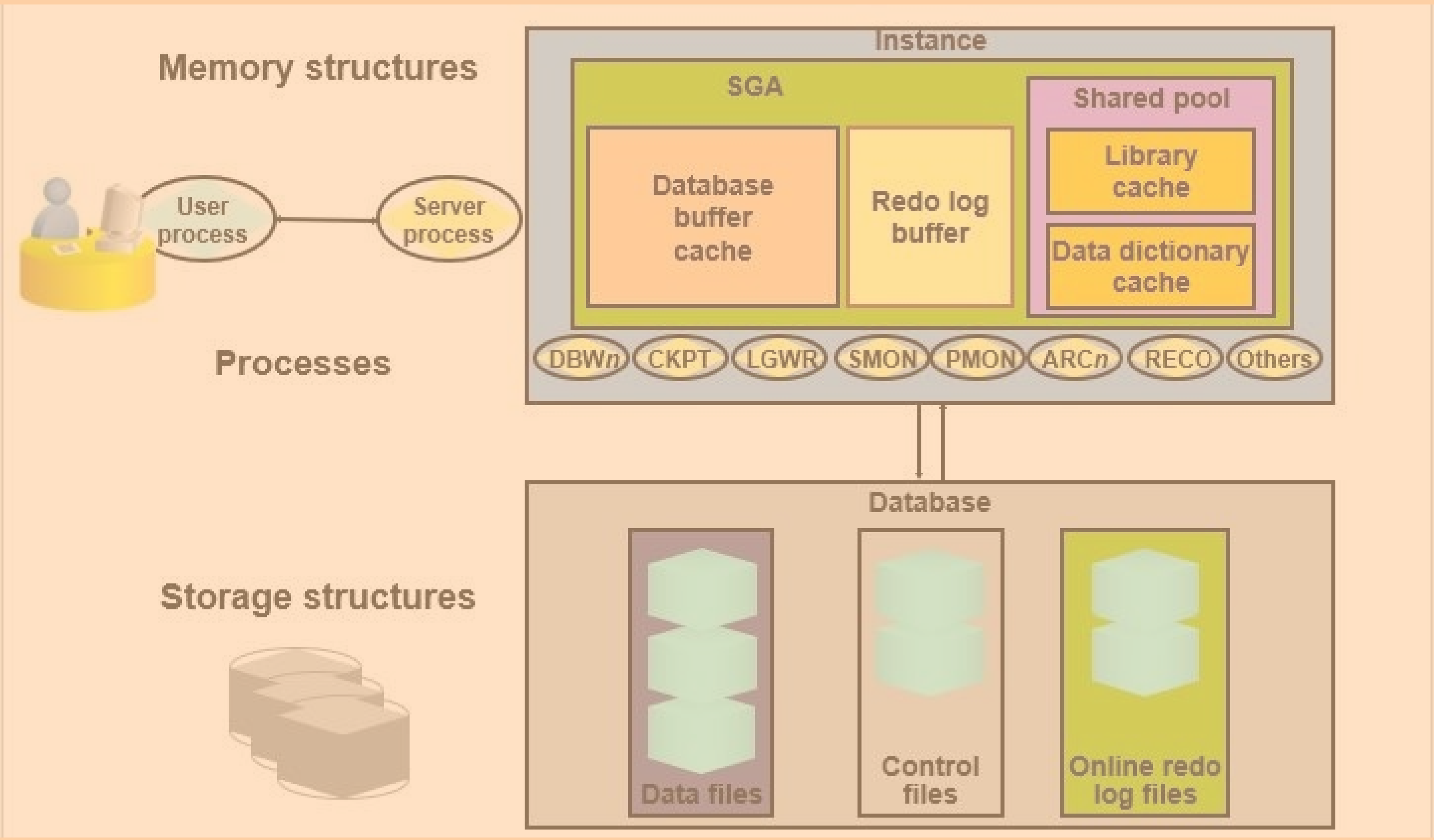
- **Connections and sessions** are related to user processes but are not the same.
- A **connection** is the communication link between a user process and the Oracle instance.
- This link uses inter-process communication or network software.
- A **session** is the user's active login state with the database.
- For example, logging into SQL*Plus with credentials starts a session that lasts until logout or exit.
- In a **dedicated connection**, the session is handled by a permanent server process; in **shared architecture**, it's handled by a server from a pool.
- A user can have **multiple sessions** using the same credentials (e.g., HR/HR) connected to the same instance at once.

Interacting with an Oracle Database



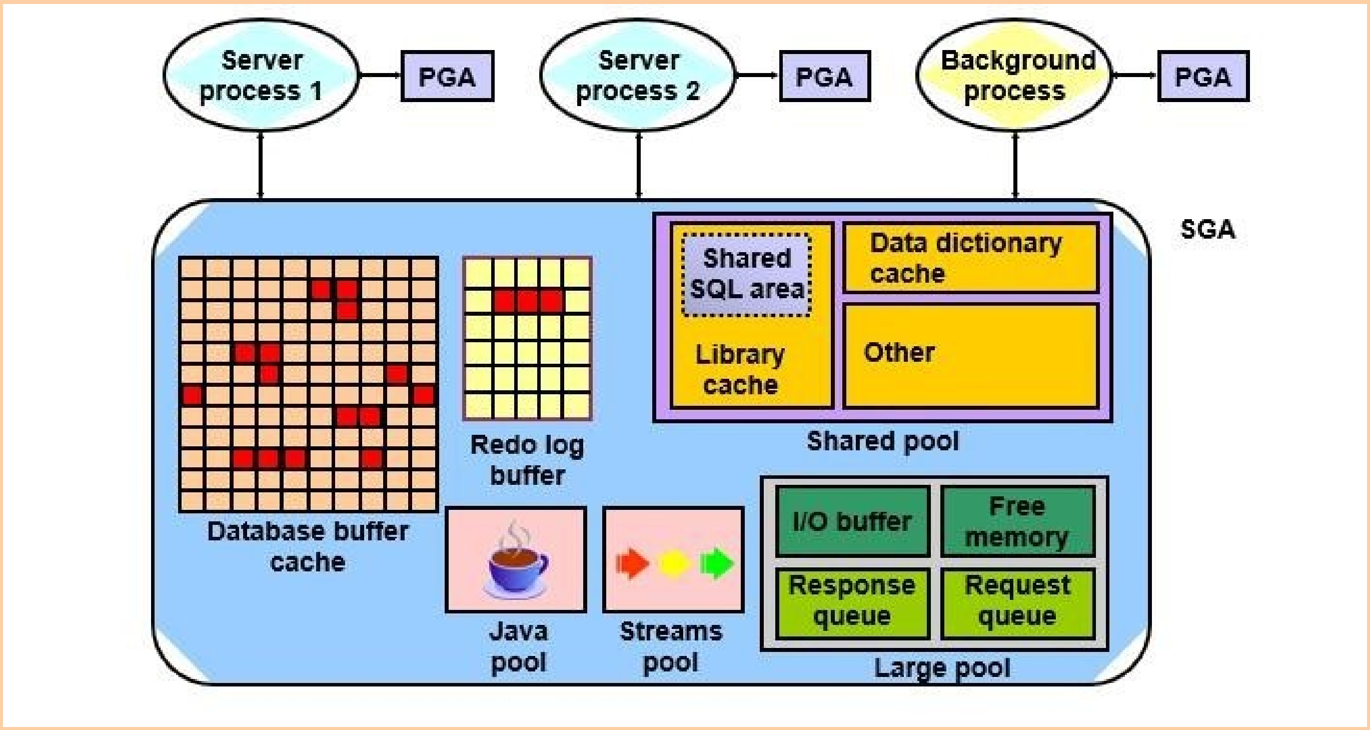
- This setup shows an Oracle database where the user and server process are on separate machines, connected via a network.
- An instance starts with the host/server where Oracle Database is installed.
- A user application starts a user process and tries to connect to the server.
- The server listener detects the request and spawns a dedicated server process for the user.
- The Oracle Listener is a server-side process that handles incoming client connection requests.
- It's part of Oracle Net Services and acts as the main communication gateway between clients and the database.

Oracle Database Server Structures



- After an instance starts, it mounts a specific database, linking it to the instance.
- The database is then opened and is accessible to authorized users.
- Multiple instances can run on the same machine, each using its own database.
- Oracle database architecture contains interrelated structural components.
- An instance uses memory and processes to manage and access the database.
- All memory structures reside in the main memory of the database server.
- Processes are tasks that operate within this memory.
- The process is a "thread of control" that runs a sequence of steps in the OS.

Oracle Database Memory Structures



- Oracle Database creates and uses memory structures for various purposes.
- For example, memory stores program code being run, data that is shared among users, and private data areas for each connected user.
- Two key memory structures in an instance are:

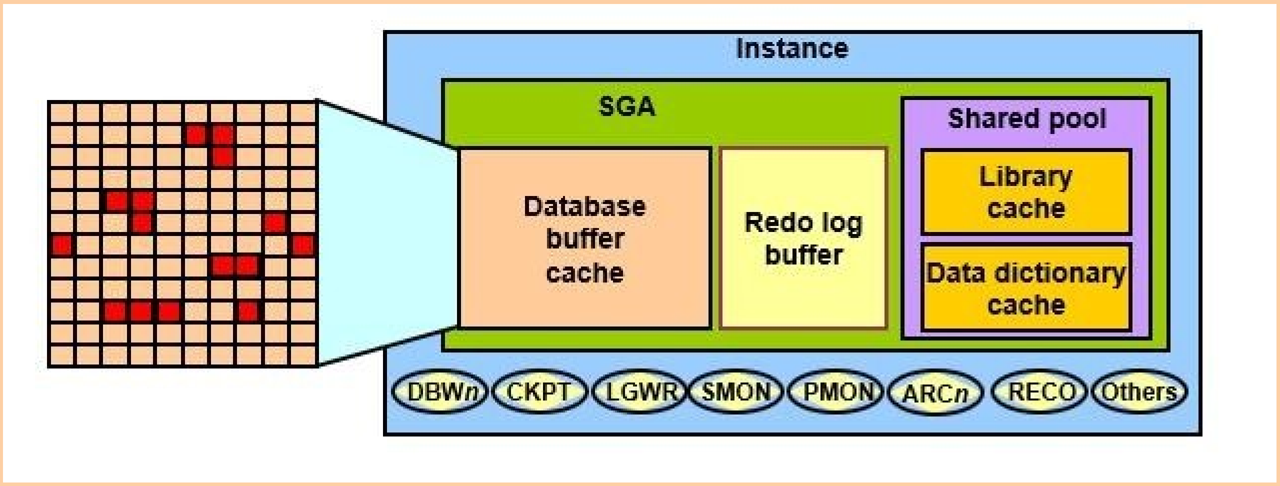
1.System Global Area (SGA): Shared memory is used by all servers and background processes.

- Holds data like cached blocks and shared SQL areas.
- Examples of data stored in the SGA include cached data blocks and shared SQL areas.

2.Program Global Areas (PGA): Non-shared memory for individual server or background processes.

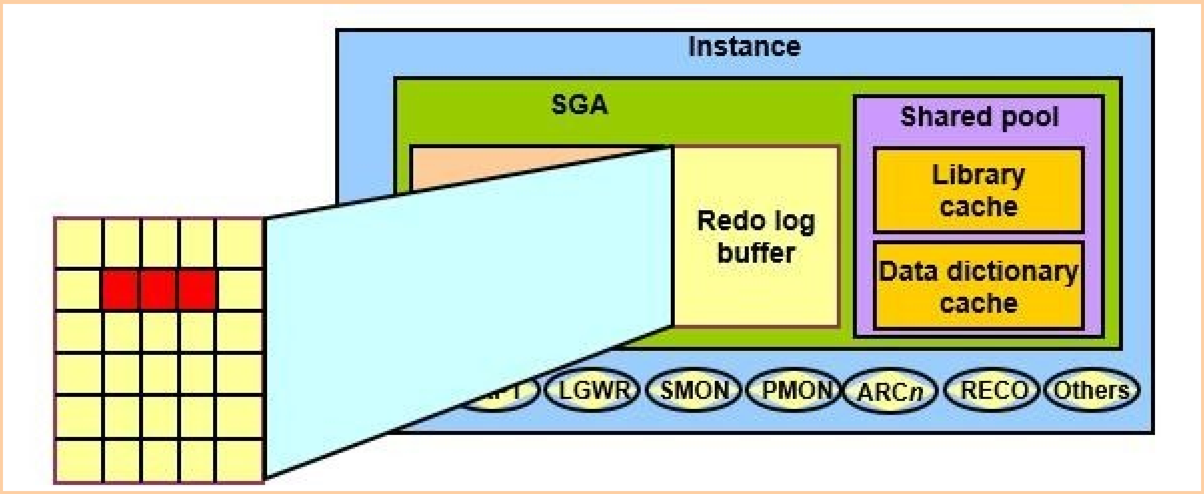
- Each process has its own exclusive PGA for control and session-specific data.

Database Buffer Cache



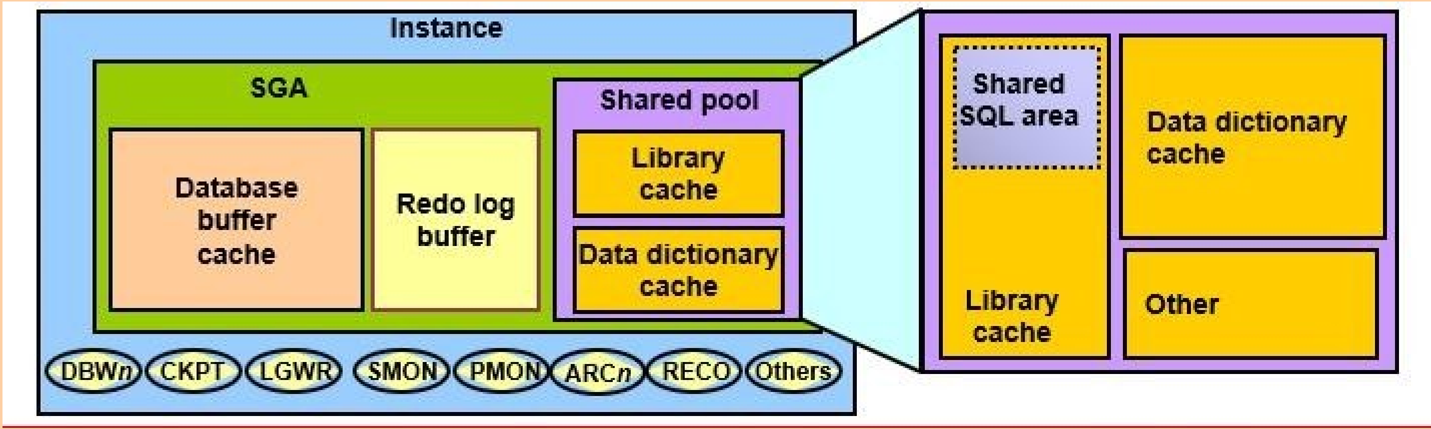
- The database buffer cache is part of the SGA that stores copies of data blocks read from disk.
- It is shared by all users connected to the instance.
- When data is needed, Oracle first checks the cache:
  - If found (cache hit), the data is read from memory.
  - If not found (cache miss), the data is loaded from disk into the cache.
- Cache hits are faster than cache misses.
- Oracle uses an LRU(**Least Recently Used**) -based algorithm with touch counts to manage the cache efficiently.
- LRU is a memory management algorithm used to decide **which data blocks to keep in memory** and **which to evict** when new blocks need to be loaded.
- Frequently accessed (recently used) blocks stay in memory.
- Less-used (least recently used) blocks are moved to the end of a list.
- When space is needed, Oracle removes the least recently used blocks to make room for new ones.

Redo Log Buffer



- The redo log buffer is a circular buffer in the SGA that holds information about changes made to the database which is stored in redo entries.
- Redo entries contain the information necessary to reconstruct (or redo) changes that are made to the database by DML, DDL, or internal operations.
- Redo entries are used for database recovery if necessary.
- Redo entries are copied by Oracle Database processes from the user's memory space to the redo log buffer in the SGA.
- The redo entries take up continuous, sequential space in the buffer.
- The LGWR(log writer) background process writes the redo log buffer to the active redo log file (or group of files) on disk.

Shared Pool

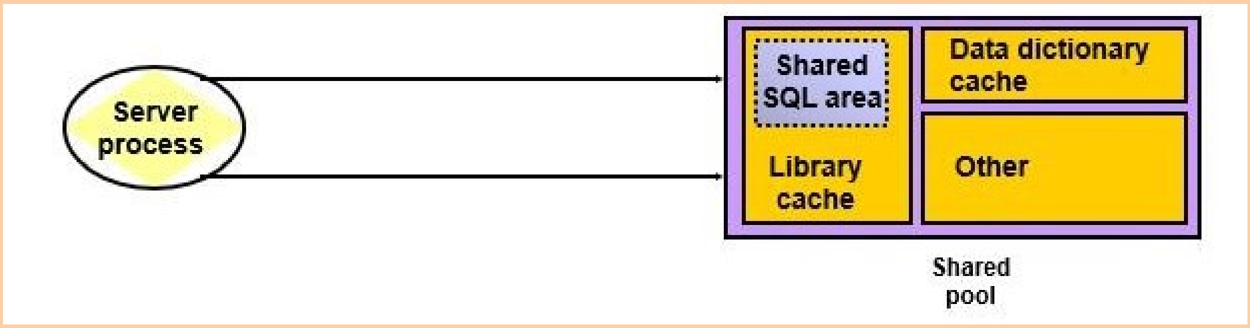


- The shared pool portion of the SGA contains the library cache, the data dictionary cache, the SQL query result cache, the PL/SQL function result cache, buffers for parallel execution messages, and control structures.
- The *data dictionary* is a collection of database tables and views containing reference information about the database, its structures, and its users.
- Oracle Database accesses the data dictionary frequently during SQL statement parsing.
- This access is essential to the continuing operation of Oracle Database.
- The data dictionary is accessed so often by Oracle Database that two special locations in memory are designated to hold dictionary data.
- One area is called the *data dictionary cache*, also known as the row cache, because it holds data as rows instead of buffers (which hold entire blocks of data).
- The other area in memory to hold dictionary data is the *library*

cache.

- All Oracle Database user processes share these two caches for access to data dictionary information.
- Oracle Database represents each SQL statement that it runs with a shared SQL area (as well as a private SQL area kept in the PGA).
- Oracle Database recognizes when two users are executing the same SQL statement and reuses the shared SQL area for those users.

Allocation and Reuse of Memory in the Shared Pool



- In general, any item (shared SQL area or dictionary row) in the shared pool remains until it is flushed according to a modified LRU (least recently used) algorithm.
- The memory for items that are not being used regularly is freed if space is required for new items that must be given some space in the shared pool.
- A modified LRU algorithm allows shared pool items that are used by many sessions to remain in memory if they are useful, even if the process that originally created the item terminates.
- As a result, the overhead and processing of SQL statements associated with a multiuser Oracle Database system are minimized.
- When a SQL statement is submitted to Oracle Database for execution, the following memory allocation steps are automatically performed:

1. Oracle Database checks the shared pool to see if a shared SQL area already exists for an identical statement.

- If so, that shared SQL area is used for the execution of the subsequent new instances of the statement.

- If there is no shared SQL area for a statement, Oracle Database allocates a new shared SQL area in the shared pool.
- In either case, the user's private SQL area is associated with the shared SQL area that contains the statement.

2. Oracle Database allocates a private SQL area on behalf of the session.

- The location of the private SQL area depends on the type of connection established for the session.

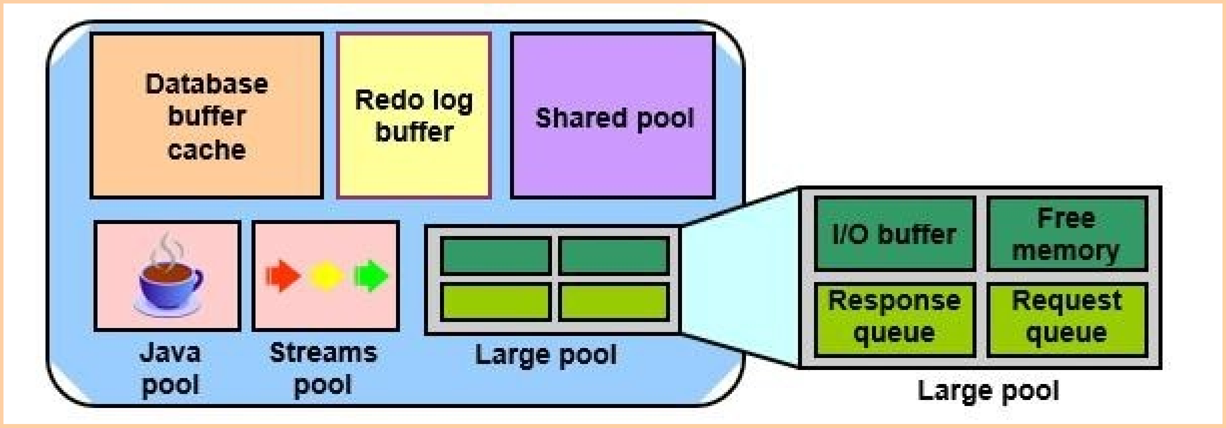
- **Note:** A shared SQL area can be flushed from the shared pool even if the shared SQL area corresponds to an open cursor that has not been used for some time.

- If the open cursor is subsequently used to run its statement, Oracle Database reparses the statement and a new shared SQL area is allocated in the shared pool.
- Oracle Database also flushes a shared SQL area from the shared pool in these circumstances:
- When the DBMS_STATS package is used to update or delete the statistics of a table, cluster, or index, all shared SQL areas that contain statements referencing the analyzed schema object are flushed from the shared pool.
- The next time a flushed statement is run, the statement is parsed in

a new shared SQL area to reflect the new statistics for the schema object.

- If a schema object is referenced in a SQL statement and that object is later modified in any way, the shared SQL area is invalidated (marked invalid) and the statement must be reparsed the next time it is run.
- If you change a database's global database name, all information is flushed from the shared pool.
- The administrator can manually flush all information in the shared pool to assess the performance (with respect to the shared pool, not the data buffer cache) that can be expected after instance startup without shutting down the current instance.
- The ALTER SYSTEM FLUSH SHARED_POOL statement is used to do this.

Large Pool

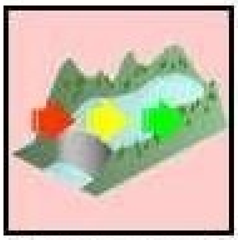


- The database administrator can configure an optional memory area called the *large pool* to provide large memory allocations for:

- Session memory for the shared server and the Oracle XA interface (used where transactions interact with more than one database):
  - I/O server processes
  - Oracle Database backup and restore operations
- By allocating session memory from the large pool for shared server, Oracle XA, or parallel query buffers, Oracle Database can use the shared pool primarily for caching shared SQL and avoid the performance overhead that is caused by shrinking the shared SQL cache.
- In addition, the memory for Oracle Database backup and restore operations, for I/O server processes, and for parallel buffers is allocated in buffers of a few hundred kilobytes.
- The large pool is better able to satisfy such large memory requests than the shared pool.

- The large pool does not have an LRU list. It is different from reserved space in the shared pool, which uses the same LRU list as other memory allocated from the shared pool.

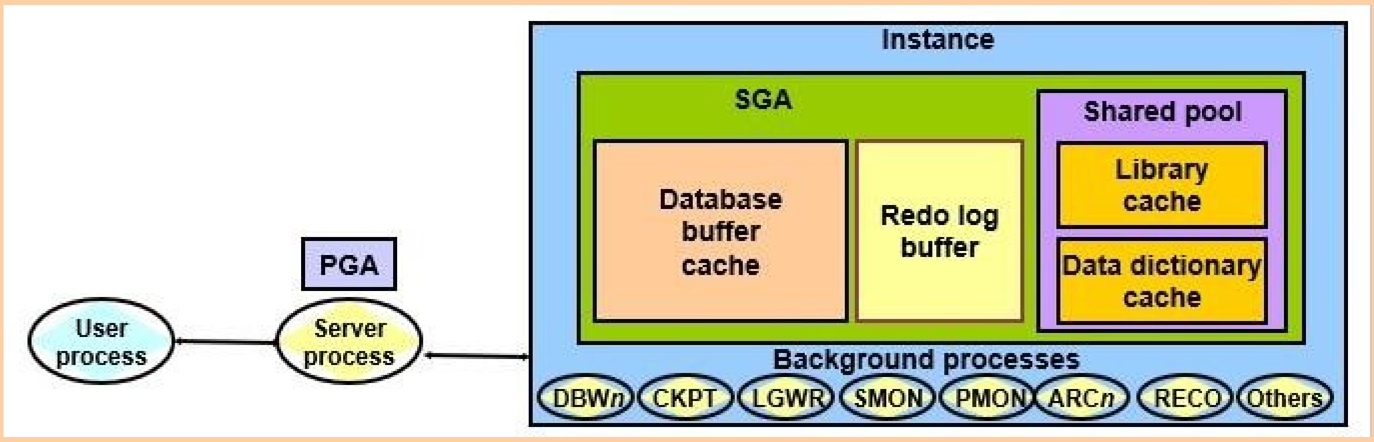
Java Pool and Streams Pool



- Java pool memory is used in server memory for all session-specific Java code and data in the JVM.
- Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.
- The Java Pool Advisor statistics provide information about library cache memory used for Java and predict how changes in the size of the Java pool can affect the parse rate.
- The Java Pool Advisor is internally turned on when statistics_level is set to TYPICAL or higher.
- These statistics reset when the advisor is turned off.
- The Streams pool is used exclusively by Oracle Streams.
- The Streams pool stores buffered queue messages, and it provides memory for Oracle Streams capture processes and apply processes.
- Unless you specifically configure it, the size of the Streams pool starts at zero.
- The pool size grows dynamically as needed when Oracle Streams is used.

- **Note:** A detailed discussion of Java programming and Oracle Streams is beyond the scope of this class

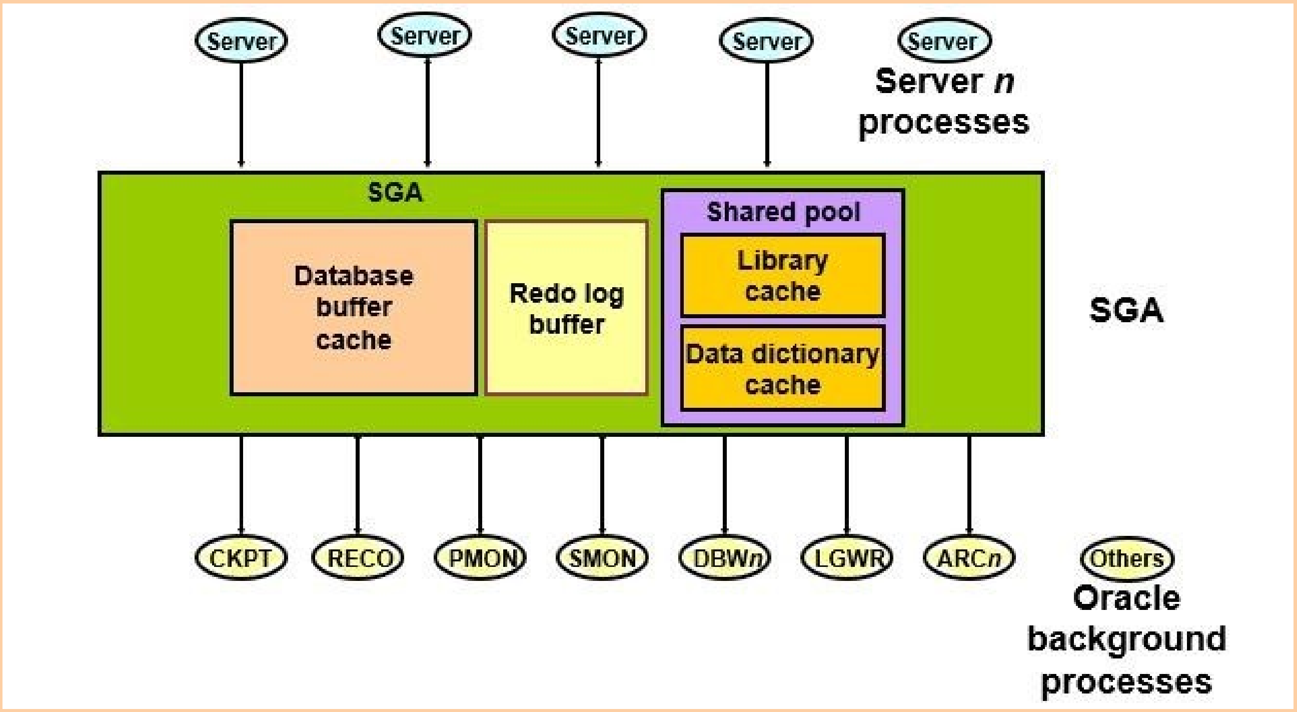
Process Architecture



- The processes in an Oracle Database system can be divided into two major groups:
- User processes that run the application or Oracle tool code Oracle Database processes that run the Oracle database server code (including server processes and background processes)
- When a user runs an application program or an Oracle tool such as SQL*Plus, Oracle Database creates a *user process* to run the user's application.
- Oracle Database also creates a *server process* to execute the commands issued by the user process.
- In addition, the Oracle server also has a set of *background processes* for an instance that interact with each other and with the operating system to manage the memory structures, asynchronously perform I/O to write data to disk, and perform other required tasks.
- The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options.

- The code for connected users can be configured as a dedicated server or a shared server.
- **Dedicated server:** For each user, the database application is run by a user process that is served by a dedicated server process that executes Oracle database server code.
- **Shared server:** Eliminates the need for a dedicated server process for each connection.
- A dispatcher directs multiple incoming network session requests to a pool of shared server processes.
- A shared server process serves any client request.

Process Structures



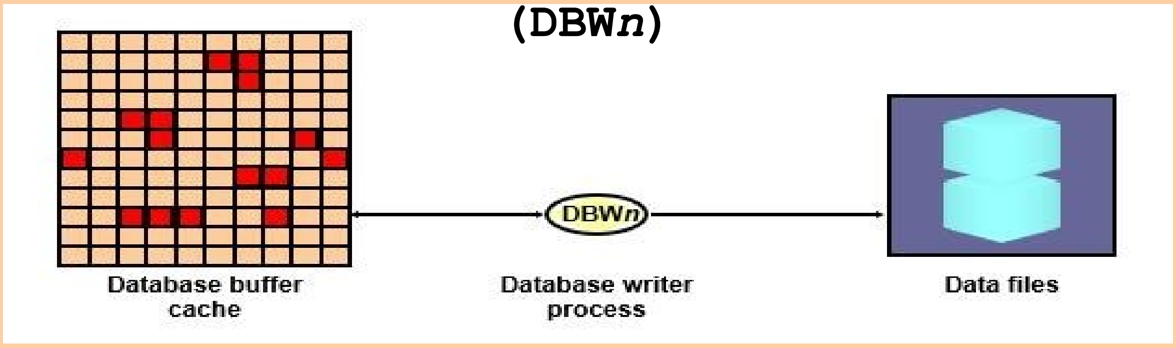
Server Processes

- Oracle Database creates server processes to handle the requests of user processes connected to the instance.
- Sometimes, when the application and Oracle Database operate on the same computer, it is possible to combine the user process and corresponding server process into a single process to reduce system overhead.
- However, when the application and Oracle Database operate on different computers, a user process always communicates with Oracle Database through a separate server process.
- Server processes created on behalf of each user's application can perform one or more of the following:
- Parse and run SQL statements issued through the application Read necessary data blocks from data files on disk into the shared database buffers of the SGA (if the blocks are not already present in the SGA) return results in such a way that the application can process the information

Background Processes

- To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called *background processes*.
- An Oracle Database instance can have many background processes.
- The background processes commonly seen in non-RAC, non-ASM environments can include the following:
    - Database writer process (DBW*n*)
    - Log writer process (LGWR)
    - Checkpoint process (CKPT)
    - System Monitor process (SMON)
    - Process monitor process (PMON)
    - Recoverer process (RECO)
    - Job queue processes Archiver processes (ARC*n*)
    - Queue monitor processes (QMN*n*)
- Other background processes may be found in more advanced configurations such as RAC.
- See the V$BGPROCESS view for more information on the background processes.
- On many operating systems, background processes are created automatically when an instance is started.

Database Writer Process (DBW*n*)



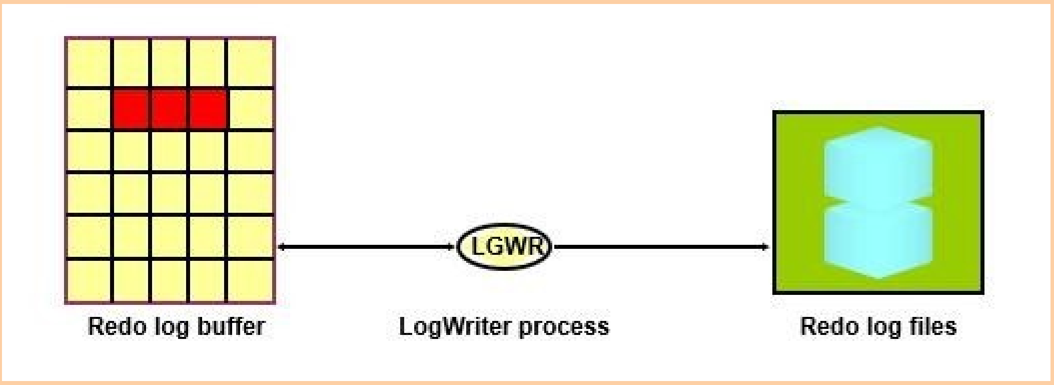
- The Database Writer process (DBW*n*) writes the contents of buffers to data files.
- The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk.
- Although one Database Writer process (DBW*0*) is adequate for most systems, you can configure additional processes (DBW*1* through DBW*9* and DBW*a* through DBW*j*) to improve write performance if your system modifies data heavily.
- These additional DBW*n* processes are not useful on uniprocessor systems.
- When a buffer in the database buffer cache is modified, it is marked dirty and is added to the LRUW (LRU write) list of dirty buffers that is kept in SCN order.
- This order therefore matches the order of redo that is written to the redo logs for these changed buffers.
- When the number of available buffers in the buffer cache falls below an internal threshold (to the extent that server processes find it difficult to obtain available buffers), DBW*n* writes dirty buffers to the data files in the order that they were modified by following

the order of the LRUW list.

- The SGA contains a memory structure that has the redo byte address (RBA) of the position in the redo stream where recovery should begin in the case of an instance failure.
- This structure acts as a pointer into the redo and is written to the control file by the CKPT process once every three seconds.
- Because the DBW*n* writes dirty buffers in SCN order, and because the redo is in SCN order, every time DBW*n* writes dirty buffers from the LRUW list.
- It also advances the pointer held in the SGA memory structure so that instance recovery (if required) begins reading the redo from approximately the correct location and avoids unnecessary I/O. This is known as *incremental checkpointing*.
- **Note:** There are other cases when DBW*n* may write (for example, when tablespaces are made read-only or are placed offline).
- In such cases, no incremental checkpoint occurs because dirty buffers belonging only to the corresponding data files are written to the database unrelated to the SCN order.
- The LRU algorithm keeps more frequently accessed blocks in the buffer cache so that, when a buffer is written to disk, it is unlikely to contain data that will soon be useful.
- The DB_WRITER_PROCESSES initialization parameter specifies the number of DBW*n* processes.

- The maximum number of DBW*n* processes is 20.
- If it is not specified by the user during startup, Oracle Database determines how to set DB_WRITER_PROCESSES based on the number of CPUs and processor groups.
- The DBW*n* process writes dirty buffers to disk under the following conditions:
- When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBW*n* to write.
- DBW*n* writes dirty buffers to disk asynchronously while performing other processing.
- DBW*n* periodically writes buffers to advance the checkpoint, which is the position in the redo thread (log) from which instance recovery begins.
- This log position is determined by the oldest dirty buffer in the buffer cache.
- In all cases, DBW*n* performs batched (multiblock) writes to improve efficiency.
- The number of blocks written in a multiblock write varies by operating system.

LogWriter Process (LGWR)



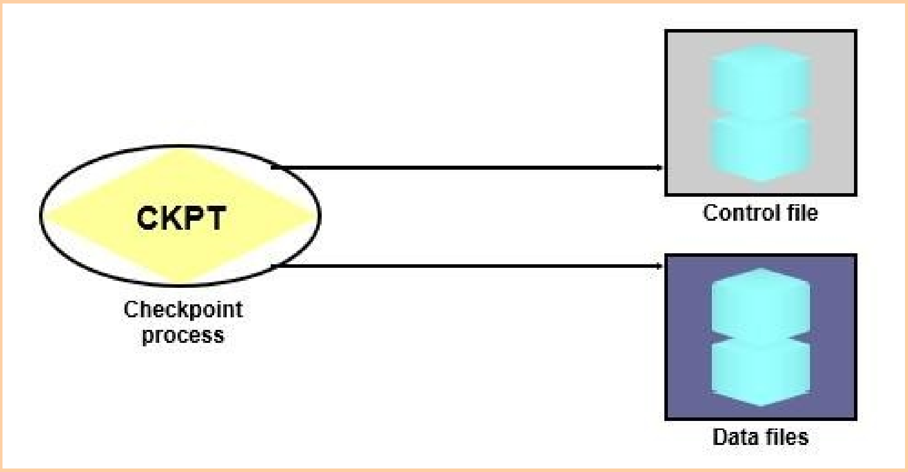
- The LogWriter process (LGWR) is responsible for redo log buffer management by writing the redo log buffer entries to a redo log file on disk.
- LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.
- The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk.
- LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.
- LGWR writes one contiguous portion of the buffer to disk.
- LGWR writes:
- When a user pLogWriter process commits a transaction
- When the redo log buffer is one-third full
- Before a DBW*n* process writes modified buffers to disk (if necessary)

- Before DBW*n* can write a modified buffer, all redo records that are associated with the changes to the buffer must be written to disk (the write-ahead protocol).
- If DBW*n* finds that some redo records have not been written, it signals LGWR to write the redo records to disk and waits for LGWR to complete writing the redo log buffer before it can write out the data buffers.
- LGWR writes to the current log group. If one of the files in the group is damaged or unavailable, LGWR continues writing to other files in the group and logs an error in the LGWR trace file and in the system alert log.
- If all files in a group are damaged, or if the group is unavailable because it has not been archived, LGWR cannot continue to function.
- When a user issues a COMMIT statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries.
- The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a *fast commit mechanism*.
- The atomic write of the redo entry containing the transaction's commit record is the single event that determines whether the transaction has been committed.
- Oracle Database returns a success code to the committing transaction, although the data buffers have not yet been written to disk.
- If more buffer space is needed, LGWR sometimes writes redo log entries before a transaction is committed.
- These entries become permanent only if the transaction is later

committed.

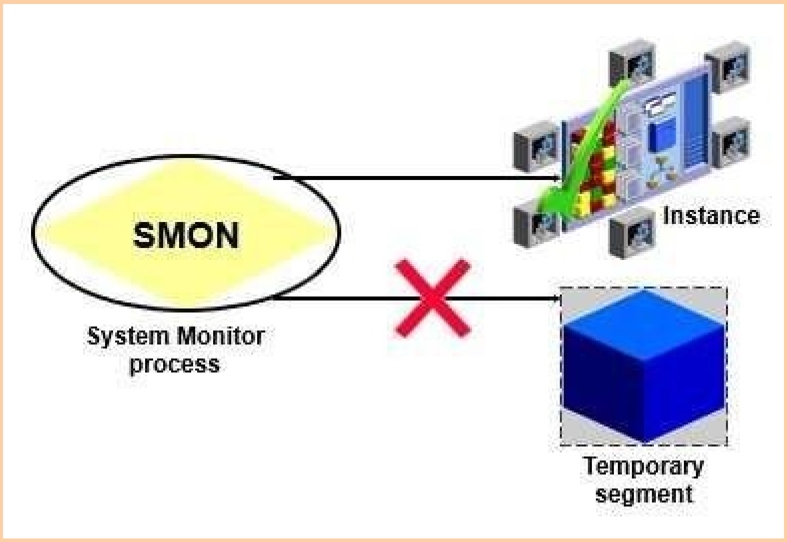
- When a user commits a transaction, the transaction is assigned to a system change number (SCN), which Oracle Database records along with the transaction's redo entries in the redo log.
- SCNs are recorded in the redo log so that recovery operations can be synchronized in Real Application Clusters and distributed databases.
- In times of high activity, LGWR can write to the redo log file by using group commits.
- For example, if a user commits a transaction, LGWR must write the transaction's redo entries to disk.
- As this happens, other users issue COMMIT statements.
- However, LGWR cannot write to the redo log file to commit these transactions until it has completed its previous write operation.
- After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually.
- Therefore, Oracle Database minimizes disk I/O and maximizes the performance of LGWR.
- If requests to commit continue at a high rate, every write (by LGWR) from the redo log buffer can contain multiple commit records.

Checkpoint Process (CKPT)



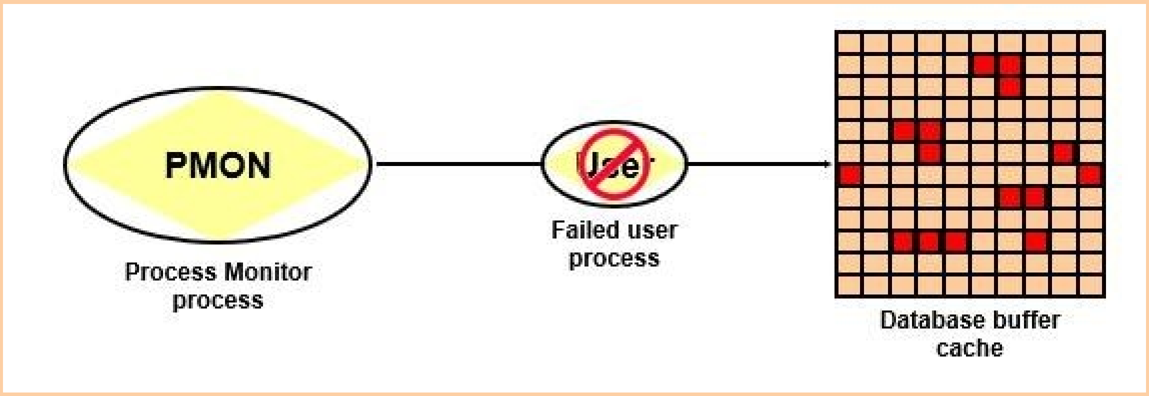
- A *checkpoint* is a data structure that defines a system change number (SCN) in the redo thread of a database.
- Checkpoints are recorded in the control file and in each data file header. They are a crucial element of recovery.
- When a checkpoint occurs, Oracle Database must update the headers of all data files to record the details of the checkpoint. This is done by the CKPT process.
- The CKPT process does not write blocks to disk; DBW*n* always performs that work.
- The SCNs recorded in the file headers guarantee that all changes made to database blocks prior to that SCN have been written to disk.
- The statistic DBWR checkpoints displayed by the SYSTEM_STATISTICS monitor in Oracle Enterprise Manager indicate the number of checkpoint requests that have been completed.

System Monitor Process (SMON)



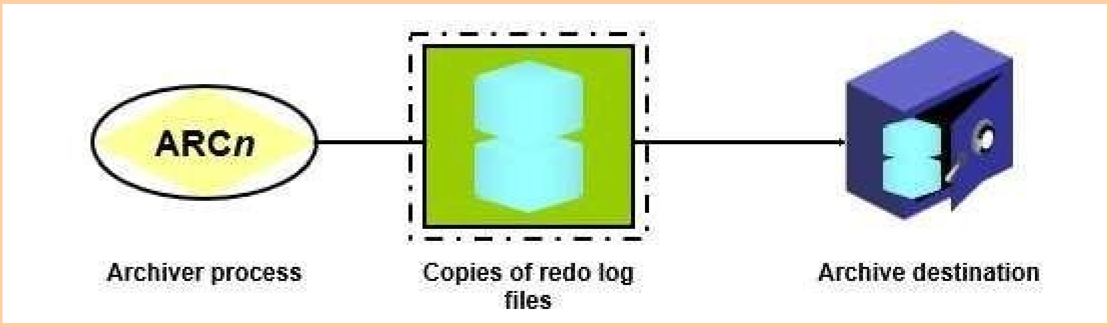
- The System Monitor process (SMON) performs recovery at instance startup if necessary.
- SMON is also responsible for cleaning up temporary segments that are no longer in use.
- If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online.
- SMON checks regularly to see whether the process is needed. Other processes can call SMON if they detect a need for it.

Process Monitor Process (PMON)



- The Process Monitor process (PMON) performs process recovery when a user process fails.
- PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using.
- For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.
- PMON periodically checks the status of dispatcher and server processes and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally).
- PMON also registers information about the instance and dispatcher processes with the network listener.
- Like SMON, PMON checks regularly to see whether it is needed; it can be called if another process detects the need for it.

Archiver Processes (ARC*n*)



- The archiver processes (ARC*n*) copy redo log files to a designated storage device after a log switch has occurred.
- ARC*n* processes are present only when the database is in ARCHIVELOG mode and automatic archiving is enabled.
- If you anticipate a heavy workload for archiving (such as during bulk loading of data), you can increase the maximum number of archiver processes with the LOG_ARCHIVE_MAX_PROCESSES initialization parameter. The ALTER SYSTEM statement can change the value of this parameter dynamically to increase or decrease the number of ARC*n* processes.

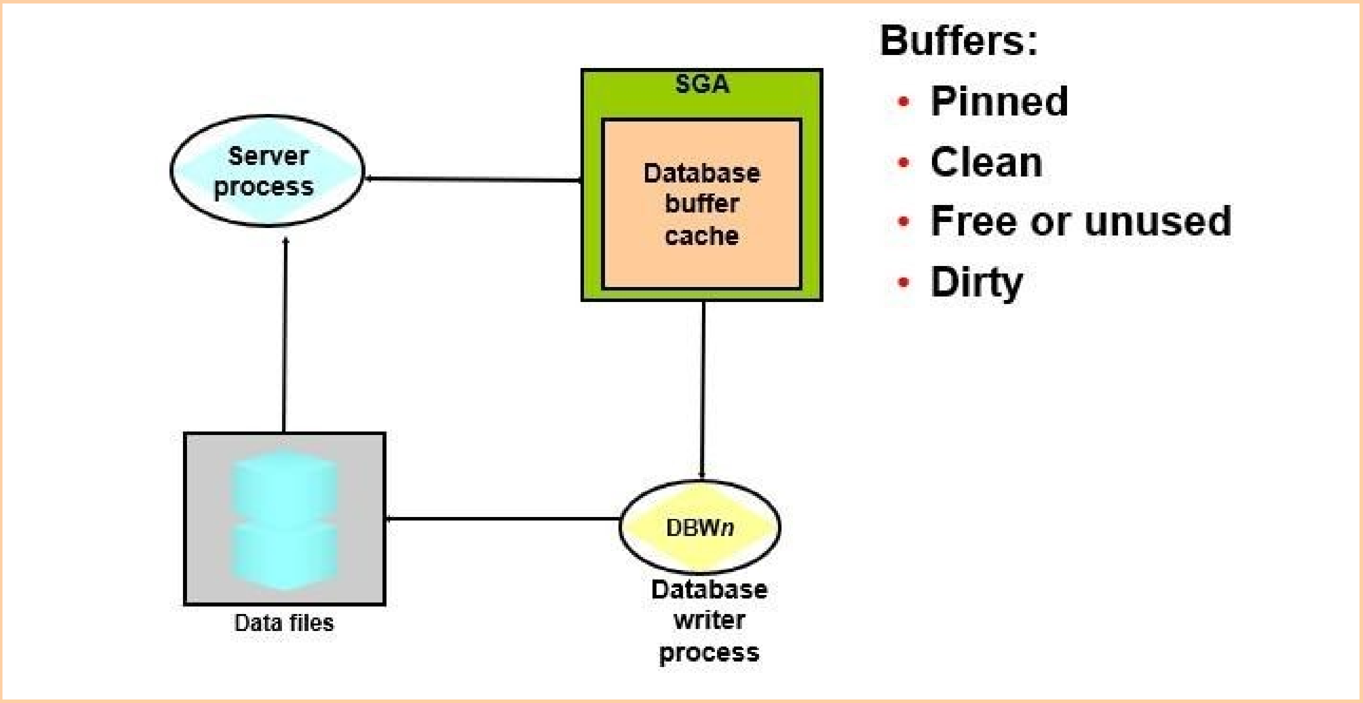
OTHER PROCESSES

- There are several other background processes that might be running. These can include the following:
- The Manageability Monitor process (MMON) performs various manageability-related background tasks.
- For example,
    - Issuing alerts whenever a given metric violates its threshold value,
    - Taking snapshots by spawning additional process (MMON slaves)
    - Capturing statistics value for SQL objects that have been recently modified
- The Lightweight Manageability Monitor process (MMNL) performs frequent tasks related to lightweight manageability, such as session history capture and metrics computation.
- The Memory Manager process (MMAN) is used for internal database tasks.
- It manages automatic memory management processing to help allocate memory where it is needed dynamically to avoid out- of-memory conditions or poor buffer cache performance.
- The Rebalance process (RBAL) coordinates rebalance activity for disk groups in an Automatic Storage Management instance.
- It performs a global open on Automatic Storage Management disks. ORBn performs the actual rebalance data extent movements

in an Automatic Storage Management instance.

- There can be many of these at a time, named ORB0, ORB1, etc.
- The Automatic Storage Management process (ASMB) is present in a database instance using an Automatic Storage Management disk group.
- It communicates with the Automatic Storage Management instance.
- Job queue processes are used for batch processing.
- They run user jobs and can be viewed as scheduler services that are used to schedule jobs as PL/SQL statements or procedures on an Oracle Database instance.
- The coordinator process, named CJQ0, periodically selects jobs that need to be run from the system JOB$ table.
- The CJQ0 process dynamically spawns job queue slave processes (J000 through J999) to run the jobs.
- The job queue process runs one of the jobs that was selected by the CJQ0 process for execution.
- The processes run one job at a time.
- The Queue Monitor process (QMNx) is an optional background process for Oracle Streams Advanced Queuing, which monitors the message queues.
- You can configure up to 10 queue monitor processes.

Server Process and Database Buffer Cache



- When a query is processed, the Oracle server process looks in the database buffer cache for images of any blocks that it needs.
- If the block image is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the database buffer cache.
- Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads.
- Buffers in the buffer cache can be in one of the following four states:
    - **Pinned:** Multiple sessions are kept from writing to the same block at the same time. Other sessions wait to access the block.
    - **Clean:** The buffer is now unpinned and is a candidate for

immediate aging out, if the current contents (data block) are not referenced again. Either the contents are in sync with the block contents stored on the disk, or the buffer contains a consistent read (CR) snapshot of a block.

- **Free or unused:** The buffer is empty because the instance has just started. This state is very similar to the clean state, except that the buffer has not been used.
- **Dirty:** The buffer is no longer pinned but the contents (data block) have changed and must be flushed to the disk by DBW*n* before it can be aged out.