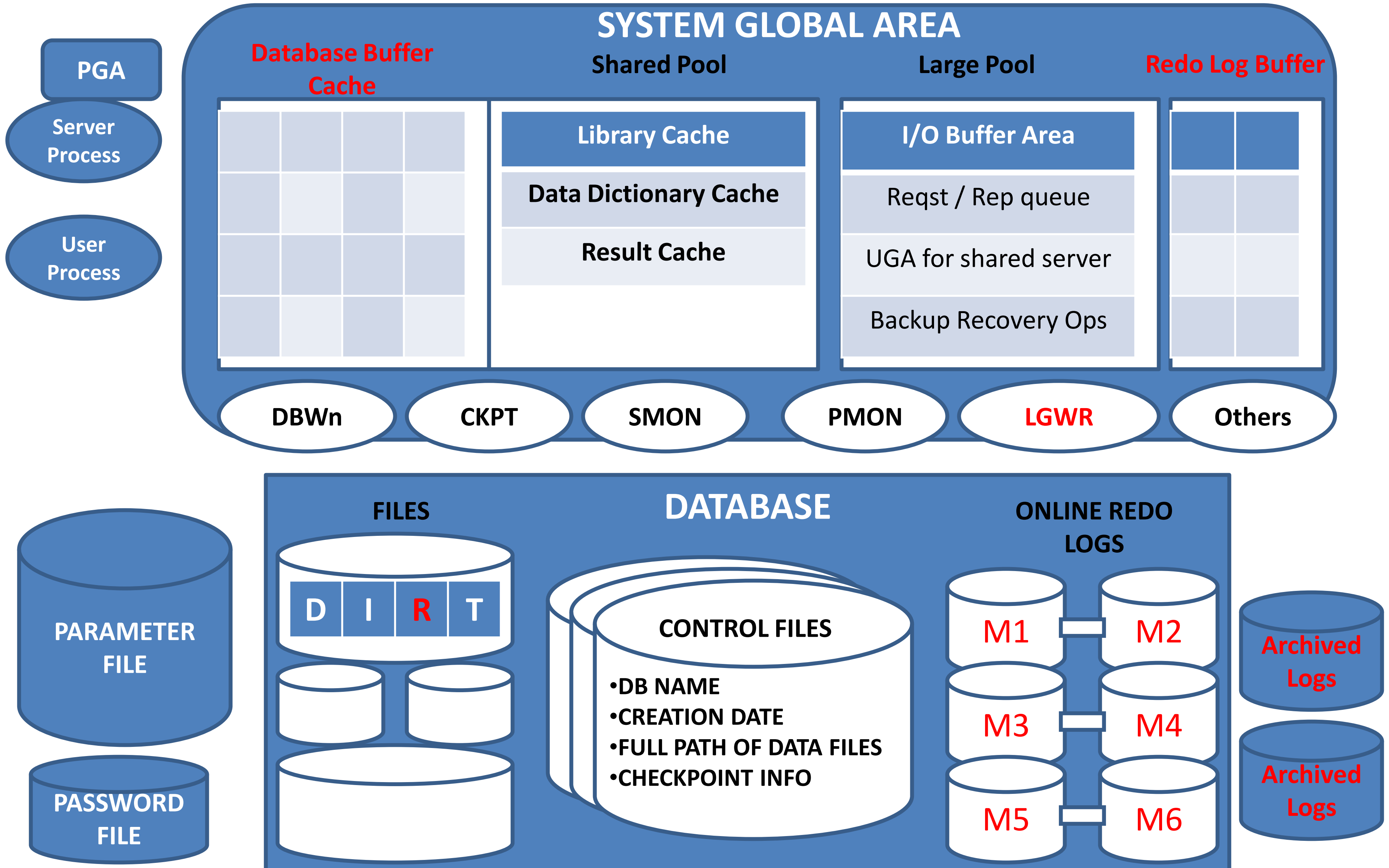
Oracle Architecture: Redo and Undo

Anuj Parashar

Session Agenda

- ❑ Recap: Overview of Oracle Architecture
- ❑ Understanding Undo
- ❑ Understanding Redo
- ❑ Undo vs. Redo
- ❑ Redo and Undo working together
- ❑ What does a COMMIT do?
- ❑ What does a ROLLBACK do?
- ❑ How big a transaction should be?
- ❑ Temporary Tables, where they fit?
- ❑ Can the redo generation be turned off?
- ❑ What Generates the most and the least undo?
- ❑ ORA-01555 Snapshot Too Old
- ❑ How Oracle creates a read consistent view?
- ❑ References
- ❑ Q & A

Recap: Overview of Oracle Architecture



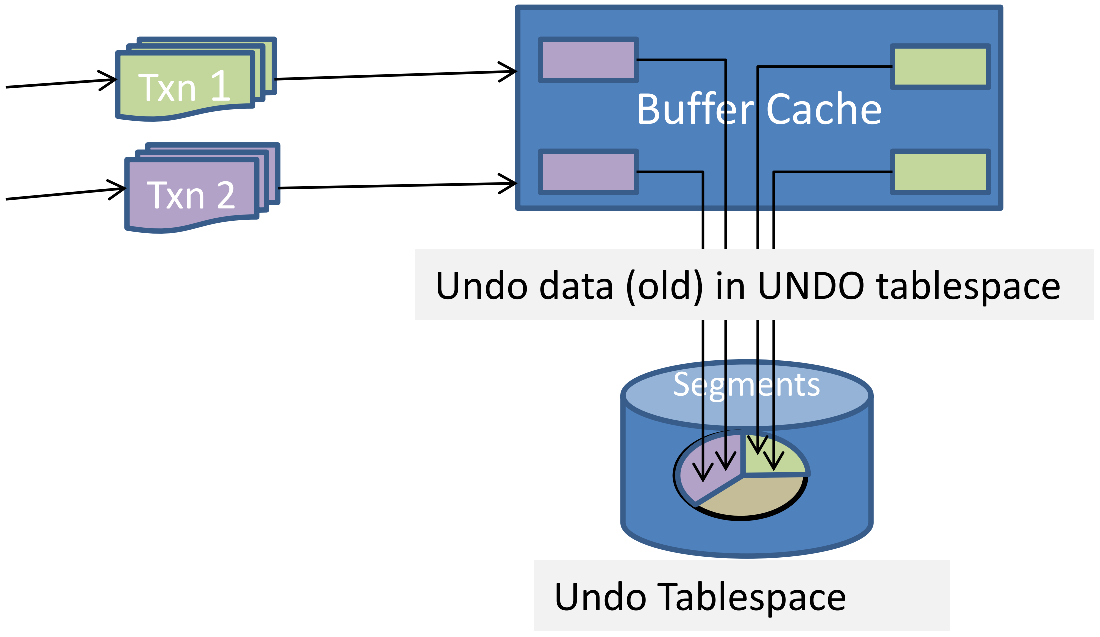
Understanding Undo

- **DML always executes as part of a transaction, which can be:**
  - Rolled back using the ROLLBACK command
  - Committed using the COMMIT command

- **Undo data is:**
  - A copy of original, premodified data
  - Captured for every transaction that changes data
  - Retained at least until the transaction is ended
  - Used to support:
    - Rollback operations
    - Read-consistent queries
    - Flashback Query, Flashback Transaction, and Flashback Table
    - Recovery from failed transactions

Understanding Undo

- **Transactions and UNDO Data (for automatic undo management):**
  - Undo is only generated by a transaction and is only relevant in the context of that transaction.
  - Whenever a transaction starts, it's assigned a transaction identifier. A corresponding entry can be seen in V$TRANSACTION view.
  - Each transaction would be allocated only one segment.
  - One undo segment can serve more than one transaction at a time.



Understanding Undo

- **Managing Undo: Automatic Undo Management:**
  - Fully automated management of undo data and space in a dedicated undo tablespace
  - For all sessions
  - Self-tuning in AUTOEXTEND tablespaces to satisfy long-running queries
  - Self-tuning in fixed-size tablespaces for best retention (ignore UNDO_RETENTION).

- **DBA tasks in support of Flashback operations:**
  - Configuring undo retention: UNDO_RETENTION specifies (in seconds) how long already committed undo information is to be retained.
  - Changing undo tablespace to a fixed size
  - Avoiding space and “snapshot too old” errors

Understanding Undo

- **Managing Undo: Manual undo management:**
  - DBA determines how many undo segment (Rollback Segments) to be manually created, based on the estimated or observed workloads.
  - No undo tablespace is used.
  - The manually managed undo segments never grows because of a long running query.
  - Can be used for limiting tablespace growth or supporting Flashback operations.
  - Long running queries could fail with a "Snapshot too old" error.
  - DML could fail because there is not enough space to accommodate undo for new transactions.
  - Not recommended anymore. From 11G onwards, AUTO is the default undo management mode

Understanding Redo

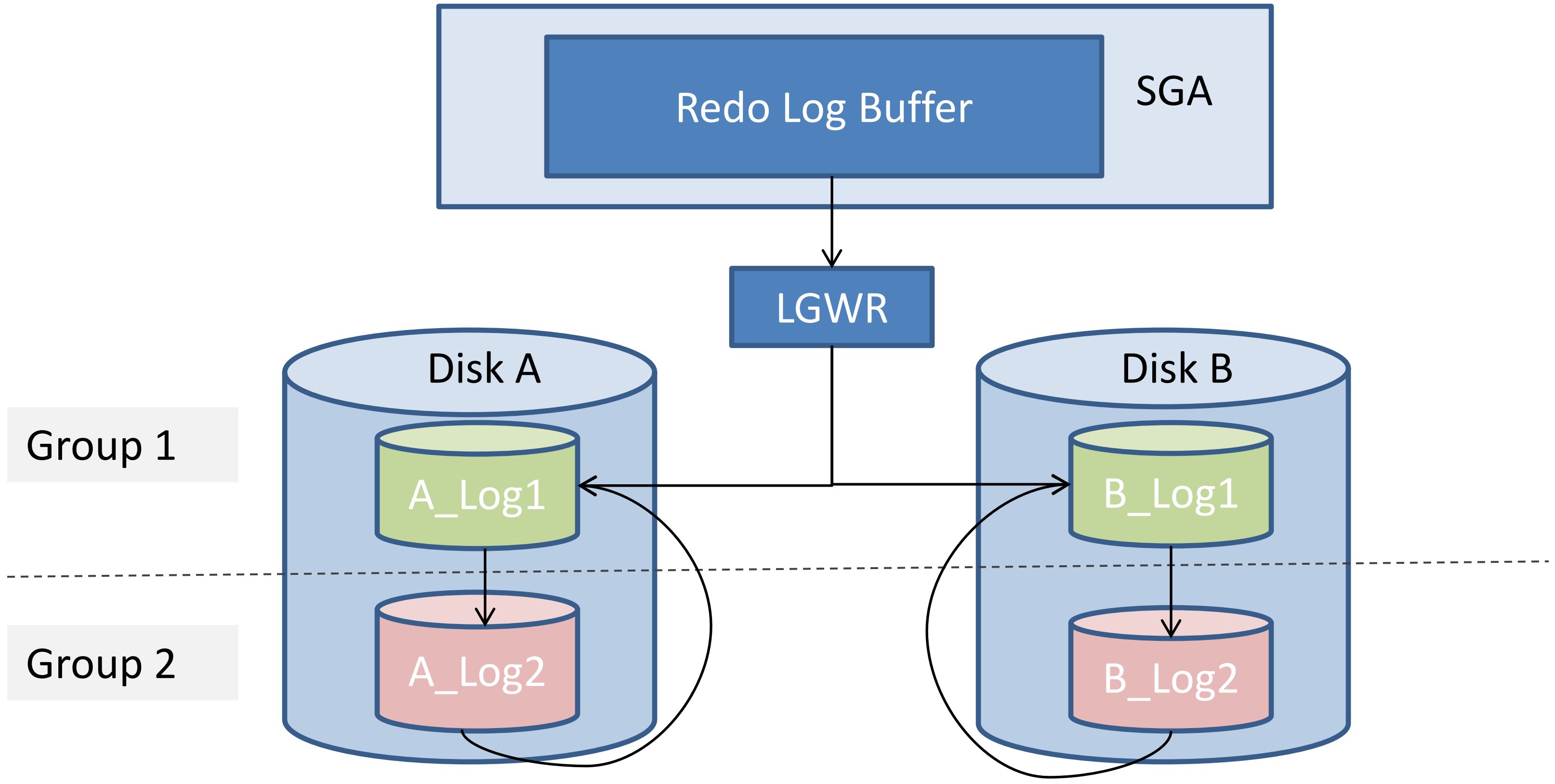
- **Characteristics of Redo (logs)**
  - These are the transaction logs for Oracle Database.
  - Redo is used to replay a transaction in the event of failure.
  - The most crucial structure for recovery operations
  - Two types:
    - Online Redo Logs: For instance Failure
    - Archived Redo Logs: For media Failure
  - Redo log buffers are flushed to disk:
    - Every 3rd Second
    - When it’s 1/3rd full
    - 1 MB of buffered data
    - Whenever COMMIT takes place
    - When a DBWn process writes modified buffers to disk, if necessary
  - UNDO data is also protected by Redo

Understanding Redo

- **Contents of Redo Log**
  - Change Vectors: It is a description of a change made to a single block in the database.
  - Redo Records / Redo Entries: It is made up of group of change vectors.
  - Ex: Update the salary of an employee:
    This would generate redo records with the following change vectors:
    - CV describing changes to the data segment block for the table.
    - CV describing changes to the **UNDO segment data block**.
    - CV describing changes to the transaction table of the UNDO segment.
  - REDO logs are written to. Oracle doesn’t read them during normal process.

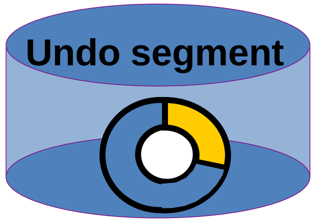
Understanding Redo

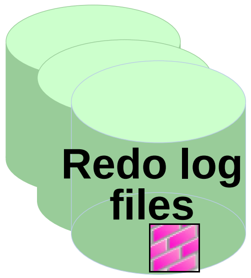
- **How Oracle Database Writes to the Redo Log?**
  - Redo records are buffered in a circular fashion in the redo log buffer of the SGA.
  - They are written to one of the redo log files by the Log Writer (LGWR) database background process.



Undo vs. Redo

	Undo	Redo
Record of	How to undo a change	How to reproduce a change
Used for	Rollback, read consistency, flashback	Rolling forward database changes
Stored in	Undo segments	Redo log files
Protects against	Inconsistent reads in multiuser systems	Data loss





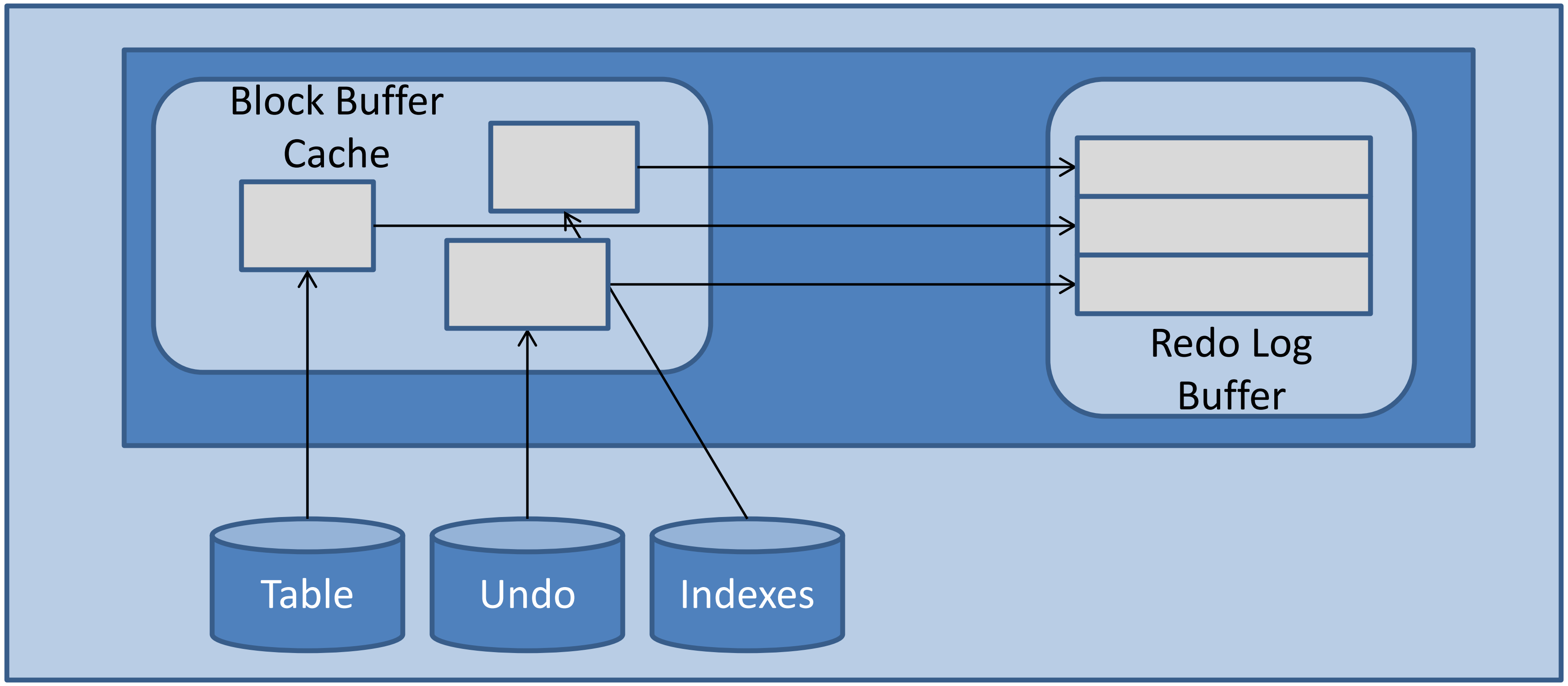
Redo and Undo working together

- **INSERT-UPDATE-DELETE Scenario**
  - insert into t (x, y) values (1, 1);
  - Update t set x = x + 1 where x = 1;
  - delete from t where x = 2;

- **What would happen?**
  - If the system fails at various points in the processing of these statements.
  - If we ROLLBACK at any point
  - If we succeed and COMMIT

Redo and Undo working together

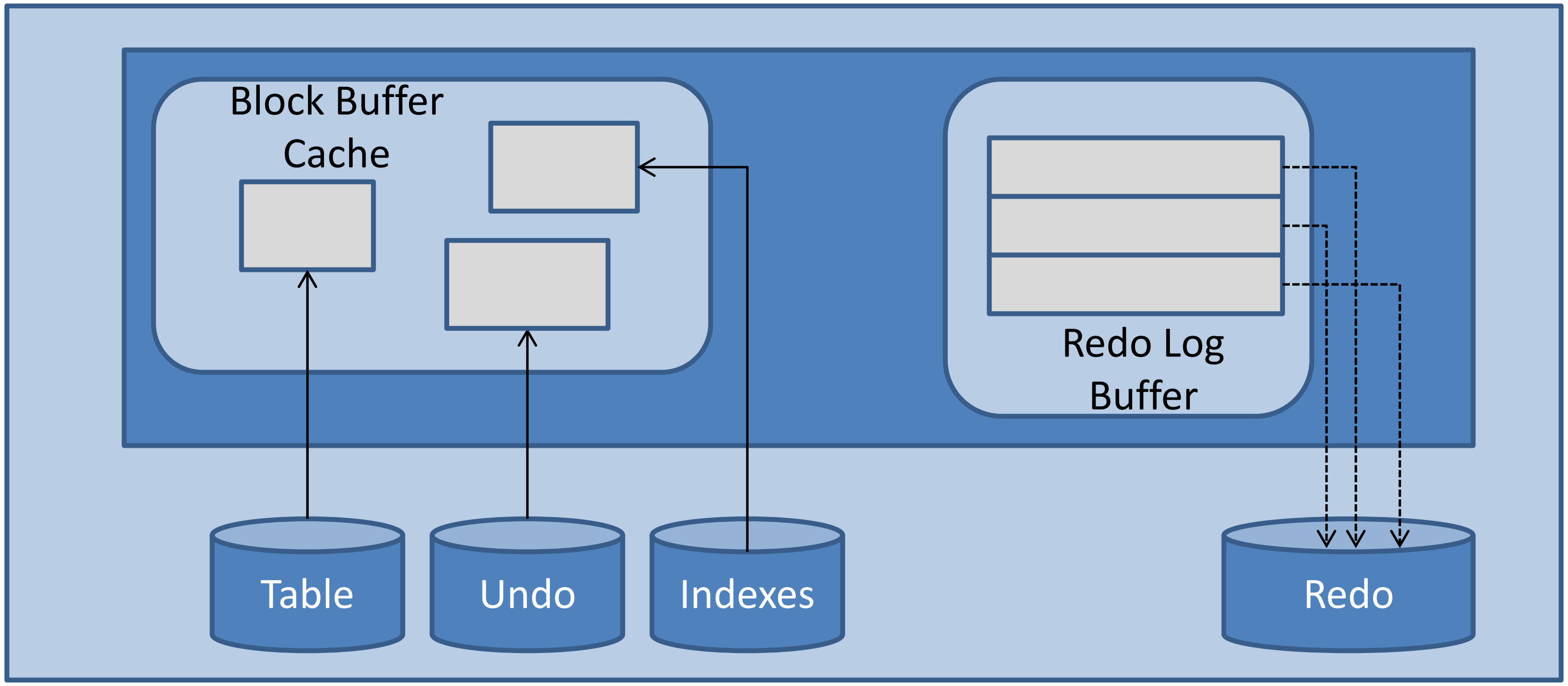
- After the initial INSERT statement



- Scenario: The system crashes just now:
  - Everything is OK. SGA is wiped out.
  - No undo or redo is required during the recovery.

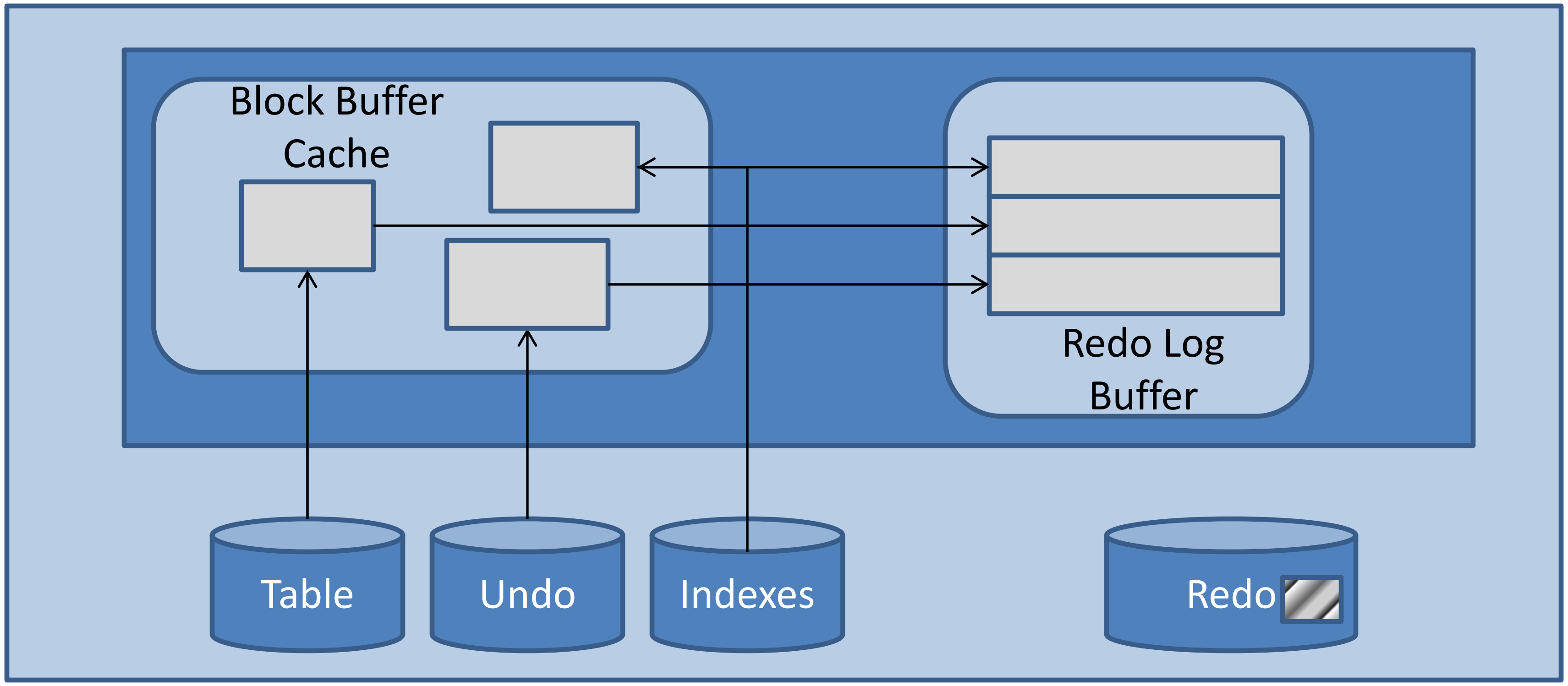
Redo and Undo working together

- **Scenario: The Buffer Cache Fills up right now**
  - DBWR must make room, the modified buffers are to be flushed from the cache to disk.
  - DBWR asks LGWR to flush the corresponding redo logs in the buffer to the disk.



Redo and Undo working together

- **The Update Statement**
  - Same thing as in case of INSERT, except the redo generated this time is more.
  - We have some redo in the buffer and some flushed to disk.



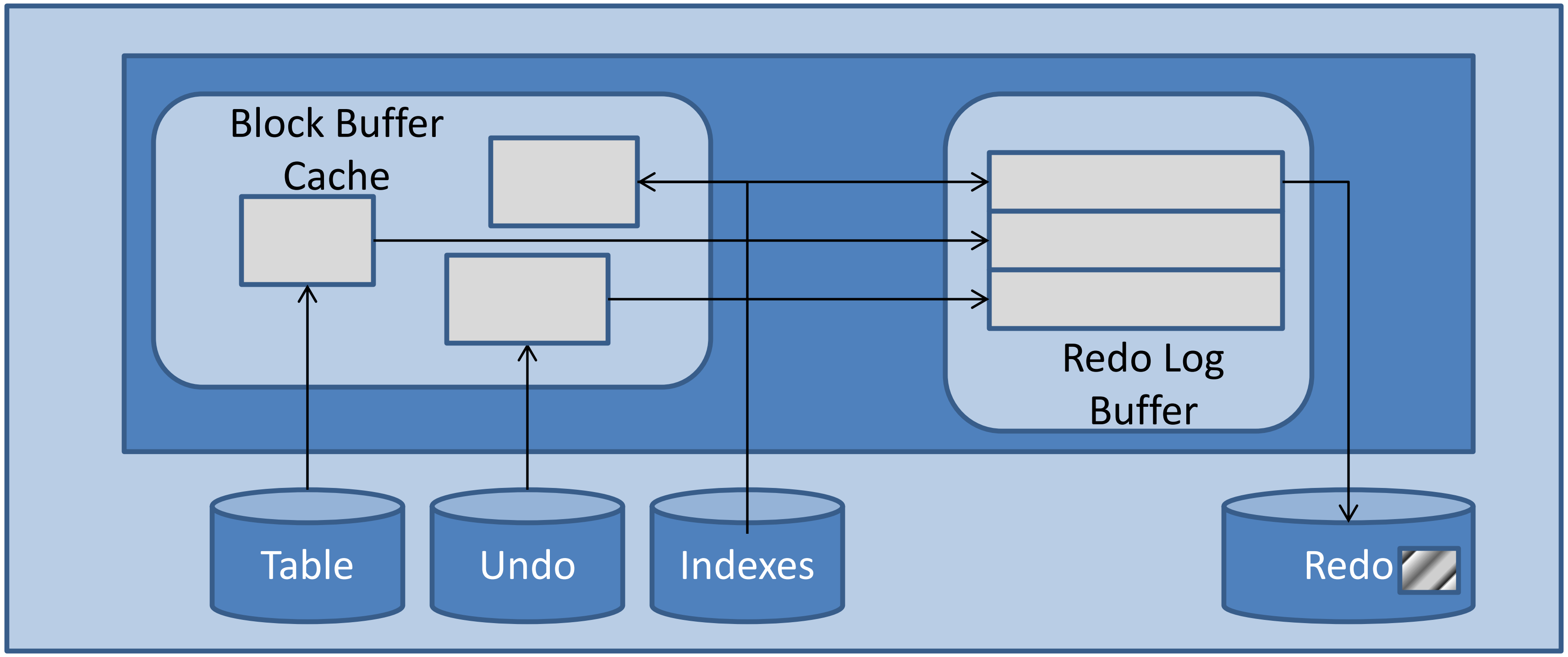
Redo and Undo working together

- **Scenario: The system Crashed right now**
  - The redo generated for the UNDO segment is now gone as it was in redo log buffer and never flushed to disk.
  - First read the redo logs and roll-forward the transaction.
  - On realizing that the transaction was never committed, rollback the transaction using the undo entries in redo logs.

- **Scenario: The application Rolls Back the Transaction**
  - Oracle would find the undo information for this transaction either in the cached undo segment blocks or on disk if they have been flushed.
  - It will apply the undo information to the data/index blocks in the buffer cache, or if they are no longer in the cache, they are read from disk into the cache to have the undo applied to them.
  - These blocks will later be flushed to the data files with their original row values restored.

Redo and Undo working together

- **The Delete Statement**
  - Same thing as in case of INSERT, except the redo generated this time is more.
- **The COMMIT**
  - The redo log buffer will be flushed to disk.
  - Changes are now permanent, though the DF might not reflect it.



What does a COMMIT do?

- **Before a COMMIT is performed**
  - UNDO blocks have been generated in SGA.
  - Modified DB blocks have been generated in SGA.
  - Buffered redo logs for above data and undo has already been generated in SGA.
  - Some combination of above data has already been flushed to disk.
  - All locks have been acquired.

- **When we COMMIT**
  - SCN (System Change Number) is generated for our transaction.
  - LGWR writes the remaining redo log buffers to disk and record the SCN in redo log files as well. This step is actual commit. The transaction entry is removed (V$TRANSACTION)
  - Locks are released (V$LOCK)
  - <u>BLOCK Cleanout</u>: Cleaning up the locking related information from the blocks header.
  - LGWR has been flushing the redo log buffers to disk on regular interval, so when it finally comes to commit, there is not much left.

What does a ROLLBACK do?

- **Before a ROLLBACK is performed**
  - UNDO blocks have been generated in SGA.
  - Modified DB blocks have been generated in SGA.
  - Buffered redo logs for above data and undo has already been generated in SGA.
  - Some combination of above data has already been flushed to disk.
  - All locks have been acquired.

- **When we ROLLBACK**
  - UNDO all the changes done by using undo segments and mark the undo entries as applied.
  - Release all the locks.

- **When should we rollback?**
  - You don’t want to rollback unless you have to.
  - People using real tables for reporting purpose and then rollback.

- **Do we need redo for rolling back the transaction?**

How big a transaction should be?

- **Philosophy**
  - As big as they should be, not any less, not any more
  - When a logical unit of work completes (Remember ACID)
  - It should be based on business need and not on available disk space.

- **What if we COMMIT frequently?**
  - Suppose you are committing x times during the processing, you spend x * COMMIT time instead (Thought it's very less)
  - COMMIT requires the LGWR to flush the redo logs buffers to online redo files, this creates a contention which could result into "LOG FILE SYNC" wait.

- **Why "Log File Sync" wait occurs?**
  - Redo management is a point of serialization within the database. There is only one LGWR process and all transactions end up at LGWR.

Temporary tables, where they fit?

- ❑ **Temporary Tables generates no REDO for their data blocks**
  - But they do generate undo and undo is logged (generates redo)
  - Undo is required for transaction management, save points, rollbacks.

- ❑ **Temporary tables would generate some REDO**
  - An INSERT will generate little or no undo/redo activity.
  - A DELETE will generate the same amount of redo as with a permanent table.
  - An UPDATE will generate about half the redo as with the permanent table.

- ❑ **Delete operations on GTT are not considered a good idea, why?**
  - DELETE generates maximum amount of UNDO and because undo gets logged, a lot of redo would be generated.
  - TRUNCATE instead, but remember it's a DDL and would commit your data.
  - Let the GTT empty themselves automatically after a commit or whenever session terminates.

Can the Redo generation be turned off?

- **NO**
  - This would leave your database unusable and unrecoverable in case of instance / media failures.
  - However some very specific operations can generate significantly less undo.

- **NOLOGGING**
  - Some amount of redo will always be generated to protect the data dictionary.
  - NOLOGGING does not prevent redo from being generated by all subsequent operations.
  - The only exceptions to the above statement is some special operations, such as direct path load using SQL *Loader, or a direct path INSERT using INSERT /*+APPEND*/.
  - After performing NOLOGGING operations in an ARCHIVELOG mode database, a new baseline backup of the affected data file must be taken as soon as possible.

What generates the most and the least undo?

- **In GENERAL, INSERT would generate the least amount of undo**
  - Need to store only the ROWID of the inserted rows to roll it back.
- **UPDATE comes after that**
  - Only the changes bytes need to be recorded along with the ROWID
- **DELETE would generate the most undo**
  - Oracle must records the entire row’s before image into the undo segment.
- **Exceptions**
  - Many factors, such as presence of INDEXES, IOT can change the story altogether. INDEXES are complex structures and might generate huge undo.

ORA-01555 Snapshot Too Old

- **Why do we get it?**
  - The undo segments are too small for the work you perform on your system.
  - Oracle’s multi-versioning model uses undo segment data to reconstruct blocks as they appeared at the beginning of your statement or transaction (depending on the isolation mode).
  - If the necessary undo information no longer exists, you will receive an ORA-01555: snapshot too old error message and your query will not complete.
  - Block cleanout

- **Contrary to the popular belief, frequent committing can result in this error.**

ORA-01555 Snapshot Too Old

- **How to avoid it?**
  - Don’t commit frequently to use a “limited resource” (undo segment) sparingly. In that case, we can get “ORA-30036: unable to extend segment by x in undo tablespace “y”.
  - This error can be dealt with by correctly sizing the undo.

- **Automatic undo management**
  - Set the UNDO_RETENTION parameter to allow your longest running query.

- **Manual undo management**
  - Let the DBA decide how many undo segments to have and how big each should be.

How Oracle creates a read consistent view

- Read the Data Block
- Read the Row Header
- Check the LockByte to see if there is an ITL entry.
- Read the ITL to determine the Transaction ID.
- Read the Transaction table. If the transaction has been committed and has a System Commit Number less than the query's SCN, cleanout the block and move on to the next data block (if required), go to Step 1.
- Read the last undo block indicated
- Compare the block transaction id with the transaction table transaction id. If the transaction id in the undo block does not equal the transaction id from the transaction table, then signal an Ora-01555, Snapshot too old.

How Oracle creates a read consistent view

- Compare the block transaction id with the transaction table transaction id. If the transaction id in the undo block does not equal the transaction id from the transaction table, then signal an Ora-01555, Snapshot too old.
- If the transaction ids are identical, clone the data block in memory, Starting with the head undo entry, apply the changes to the cloned block.
- If the tail undo entry (the last one read) indicates another data block address, read the indicated undo block into memory. Repeat 7 & 8 until the first record does not contain a value for the data block address.
- When there is no previous data block address, the transaction has been undone.
- If the undo entry contains
  - a pointer to a previous transaction undo block address, read the Transaction ID in the previous transaction undo block header and read the appropriate Transaction Table entry. Return to step 5.
  - an ITL record, restore the ITL record to the data block. Return to step 4.

References

- ❑ **Expert Oracle Database Architecture: by Thomas Kyte**
  http://www.apress.com/9781430229469

- ❑ **Oracle Documentation:**
  http://www.oracle.com/technetwork/database/enterprise-edition/documentation/database11gr1-087487.html

- ❑ **Arup Nanda's Blog:**
  http://arup.blogspot.in/2011/01/how-oracle-locking-works.html

- ❑ **Oracle Core: Essential Internals for DBAs and Developers**
  http://www.apress.com/9781430239543

- ❑ **Different Oracle whitepapers**