Failure Classification :

1. Transaction failure
    a. Logical error : Transaction incomplete due to internal error condition
    b. System error : DBMS must terminate active transaction due to errors like deadlock
2. System crash : Power failure and hw/sw failure causing system crash
   Fail-stop assumption : Non volatile contents assumed to be not corrupted by system crash
3. Disk failure : Head crash/disk crash causing data storage loss. This failure can be detected using checksums

Storage classification

1. Non-volatile storage : Survive system crash (most of the times)
   Disk, tape, flash memory
2. Volatile storage : doesn't survive system crash
   Main and cache memory
3. Stable storage : Mythical storage that always survives : replication sets at local and remote sites

Data Access :

1. Physical blocks
2. Buffer blocks

Logs are kept in stable storage
Immediate database modification :
Write == commit directly to buffer and database
Immediate DB modification failure recovery system :
Logs:
<T1 start>
<T1, var, old val, new val>
<T1 commit>

1. Failure after T started and committed? → Redo
2. Failure after T started and not committed? → Undo + log record <T, var, old val> written + <T abort>

Deterred DB modification failure recovery :
<T1 start>
<T1, var, new val>
Write only in local buffer
Commit will lead to DB modification

3. Failure after T started and committed? → Redo
4. Failure after T started and not committed? → Nothing

SCR start commit redo

Check points :
Unnecessary redoing of transactions which have been committed and started. Rather, only redo those operations after a checkpoint.
A log record of <checkpoint L> was added where L is the active transaction list at that time

Redo phase:
1. Find last <checkpoint L> record, and set undo-list to L.
2. Scan forward from above <checkpoint L> record
1. Whenever a record <T i , X j , V 1 , V 2 > is found, redo it by writing V 2 to X j
2. Whenever a log record <T i start> is found, add T i to undo-list
3. Whenever a log record <T i commit> or <T i abort> is found, remove T i from undo-list

Undo phase:
Scan log backwards from end
1. Whenever a log record <T i , X j , V 1 , V 2 > is found where T i is in undo-list perform same actions as for transaction rollback:
1. perform undo by writing V 1 to X j .
2. write a log record <T i , X j , V 1 >
2. Whenever a log record <T i start> is found where T i is in undo-list,
1. Write a log record <T i abort>
2. Remove T i from undo-list
3. Stop when undo-list is empty
● i.e. <T i start> has been found for every transaction in undo-list
● After undo phase completes, normal transaction processing can commence

Shadow paging : Alternative to log based recovery
-Useful if transactions execute serially
- 2 page tables during lifetime of the transaction : current page table and shadow page table
Current Page table points to a page in the disk and is in the main memory.
Shadow page table is in the non volatile memory(disk).
Working :

1. Consider updation of a page :

a. CPT helps in accessing that particular page.
b. Shadow page is already pointing at this particular page since earlier write.
c. The states of all data items are copied into an unused page and updations are carried out on this unused page.
d. The CPT now points to this particular previously unused page.
e. Now suppose a failure occurs at this point. Now, CPT can just point to the page where SPT is pointing.

2. Commit
a. Modify pages in disk.
b. Copy CPT in disk and make it a SPT. Update pointer to SPT
c. Free memory not needed

Properties	Shadow Paging	Log based recovery
Maintain	CPT and SPT	Log record
Recovery	Trivial, only memory references changes	Complex Undo and redo operations
Overhead	No overhead of maintaining a log record Overhead of copying CPT	
Garbage collection	Necessary	Not needed