# Chapter 9
# Log Manager

Jong-Hyeok Park
akindo19@gmail.com


VLDB Lab.

# Log Manager

- ***Size*** and ***Performance*** issue : KEY POINT of system performance
- Log is temporal databases.
- Log manager provide interface to **Log Table** – sequence of log records

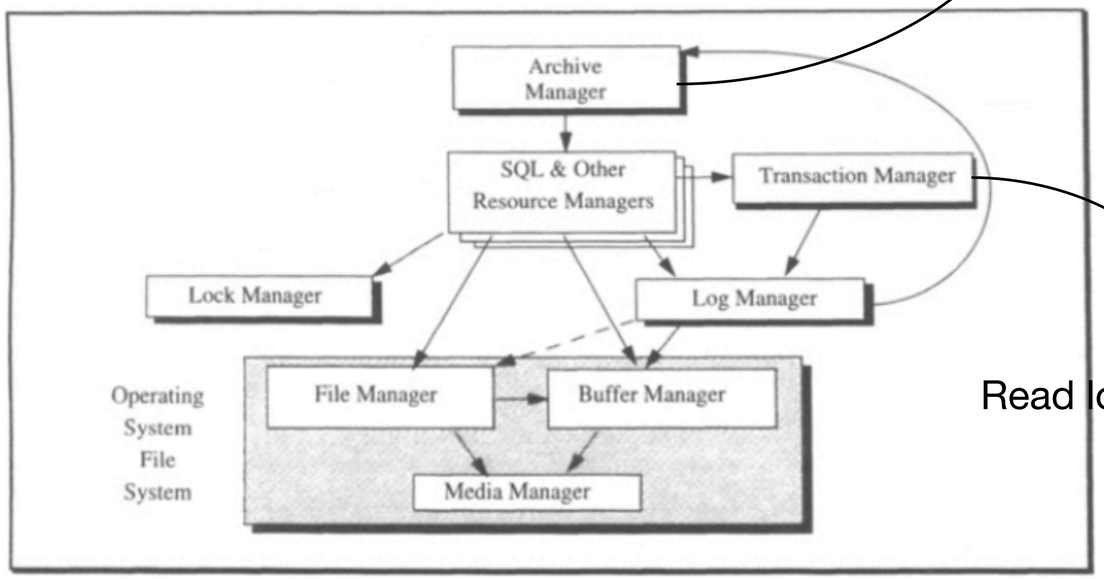
Scanning backward via only access header ; No forward lsn needed

For transaction UNDO

```
create domain LSN                unsigned integer(64);  -- log sequence number (file #, rba)
create domain RMID               unsigned integer;      -- resource manager identifier
create domain TRID               char(12);              -- transaction identifier
create table log_table (
        lsn                      LSN,                   -- the record's log sequence number
        prev_lsn                 LSN,                   -- the lsn of the previous record in log
        timestamp                TIMESTAMP,             -- time log record was created
        resource_manager         RMID,                  -- resource manager that wrote this record
        trid                     TRID,                  -- id of transaction that wrote this record
        tran_prev_lsn            LSN,                   -- prev log record of this transaction (or 0)
        body                     varchar,               -- log data: rm understands it
        primary key (lsn)                               -- lsn is primary key
        foreign key (prev_lsn)                          -- previous log record in this table
            references log_table(lsn),                  --
        foreign key (tran_prev_lsn)                     -- transaction's prev log rec is also in table
            references log_table(lsn),                  --
    )   entry sequenced;                                -- inserts go at end of file
```

# Log Manager

Only recent log records should be kept online




Read log records when Trx or System fail

# Log Manger

**Q. Why have a Log Manager ?**

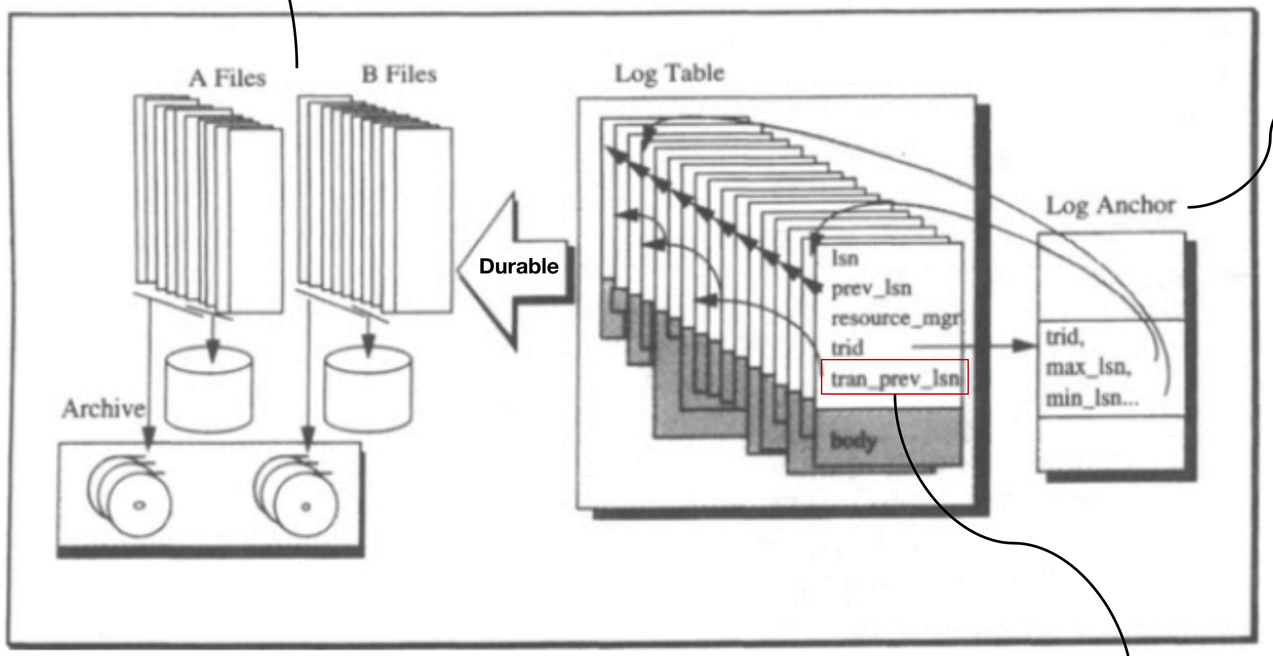
**A. (Just) entry-sequenced SQL table is NOT enough !!!**

- **Encapsulation :** Fault isolation + Provide an interface for new resource manager.
- **Startup :** Reconstruct durable system at system restart; SQL compiler and catalog are not runnable.
- **Careful Writes :** Log is duplexed ; the log is **the only durable copy** of committed transaction updates until the data is copied to durable storage.

# Log Tables

**LOGA0000000** **LOGB0000000**

File index in the log directory




Cached in main memory

Speed transaction backout

# LSN

- ***LSN (Log Sequence Number)*** *: file number + relative byte offset within that file*
  - **Unique**
  - **Monotonicity : WAL protocol**

$$LSN(A) > LSN(B).$$

If log record A is created **after** log record B

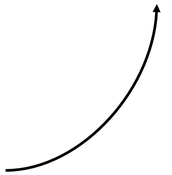
```
typedef struct {    long    file;       /* number of log file in log directory                */
                    long    rba;        /* relative byte address of (first byte) record in file */
                }   LSN ;               /* Definition of Log Sequence Numbers (LSN)           */
LSN NullLSN ={0,0};                     /* null LSN is used to end pointer chains.            */
```

# Log Table Interface

- ***Log Table Open & Close***
  - Authorization to access log table
- ***Log Table Read***
  - Func1. Copy log record body and return log header
  - Func2. Return current maximum LSN
- ***Log Insert***
  - 1) Allocate space for log record at the end of log; 2) fill header ; 3) fill body
  - Return LSN of resulting log record
- ***Log Flush***
  - All log records up to and including the designated LSN will have been copied durable storage
  - Lazy option – defers log writes ; most records have already been written async. by buffer manager.

Group commit

- Batching , Box-carrying
- Amortize I/O cost vs. Delay in commit



# Log Table Interface

**Record-oriented read + insert/flush interface**

- ***Data Copy***
  - Don’t provide direct pointer to the data area in the log buffer

- ***Increment Insert***
  - Caller provide the entire log body on a single insert call
    (vs. provide it one part at a time.)

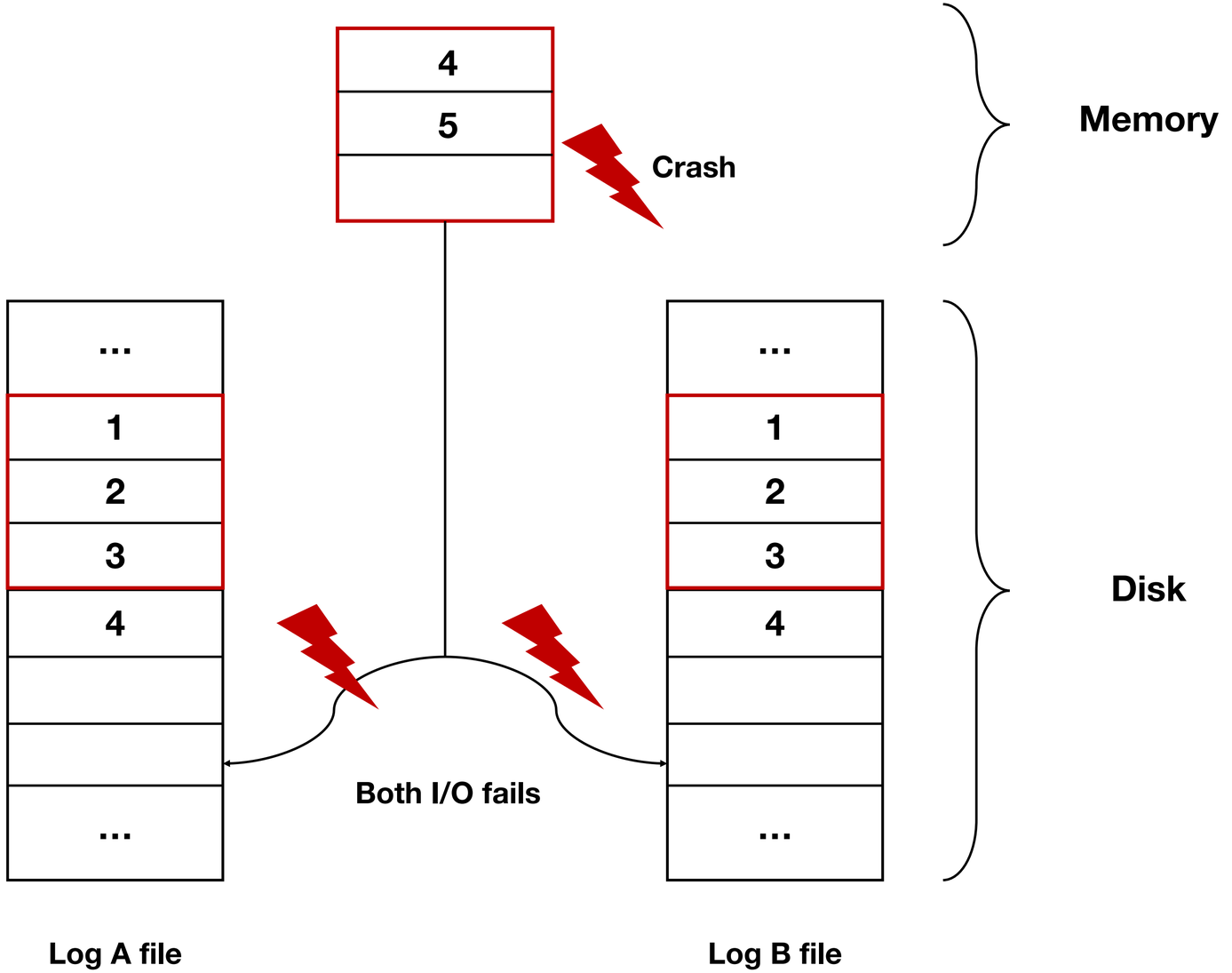
- ***SQL representation***
  - Allow SQL interface to access log table

# Log Anchor

- LSN of **most recently written** records
- LSN of next records
- Max LSN in stable storage
- Checkpoint information

- Making Anchor stable
  - Ping-Pong writes
  - Serial writes (Not parallel write)
  - Can not write log anchor to stable storage after every update **TOO slow !!!**
  - Use relative log anchor from storage ; use it as a hint

# Bad scenario for Parallel writes

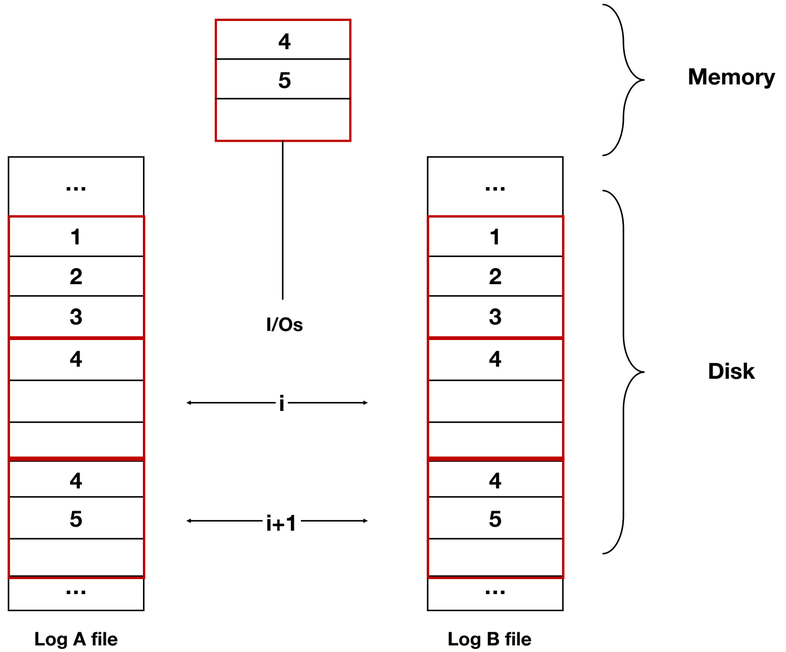



It’s ok to lose “4” but “3” is not

# Bad scenario for Parallel writes

- Solution #1. Allow only full page write
  - ➢ Wait all transaction committing in log record entry

- Solution #2. Pin—Pong write
  - ➢ Write two pages alternatively
  - ➢ Only “one” lost in bad scenario



# WADS

- Write Ahead Data Set
- High-transaction rate system requires minimal latency in writing a disk-based log
- Key point : Ignore track switches when writing logs
  - Dedicated cylinder : Assign a whole cylinder to log tail
  - Write anywhere on next track : On flush write to closest sector on next free track
  - Write log when cylinder fills : Write log tail to end of a log
  - Read cylinder at restart : Reconstruct log

# WADS

$\boldsymbol{b = time\ to\ write\ block\ (\approx 0.1ms)}$

$\boldsymbol{r = expected\ rotational\ delay\ (\approx 20ms)}$

$\boldsymbol{n = number\ of\ WADS\ track\ (10)}$

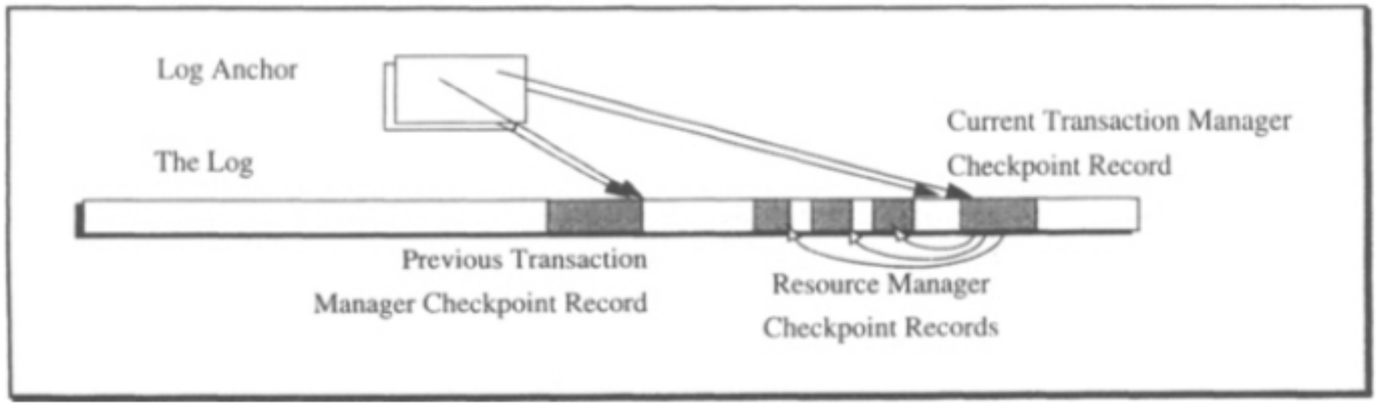
- For each of the n writes in WADS track : $\boldsymbol{t = \frac{b}{2} + b\ \approx 0.1\ ms}$
- For the 1 write of all n+1 blocks at the log tail : $\boldsymbol{t = r + (n+1)b\ \approx 20\ ms}$
- Vanilla scheme (per block) : $\boldsymbol{t = r + b\ \approx 20\ ms}$

# Log Restart

$log\_write\_anchor(LSN\ anchor)$

$log\_read\_anchor(void)$

- **Update trx manager LSN in the log anchor**
- **Read actual anchor (checkpoint record) from the log**



# Log Restart

**A. Preparing for Restart : Careful Write of Log Anchor**

- ***Log anchor*** points log files and the current log end ( ***log_anchor.lsn*** )
- Careful writes:
  - **Ping-Pong :** Don’t overwrite most recent anchor record
  - **Independent Updates :** Put records in different disk block
  - **Independent Failures :** Place two files on different media
  - **Accept most recent :** When reading file, read one with the most recent timestamp

**B. Finding the Anchor and Log end at Restart**

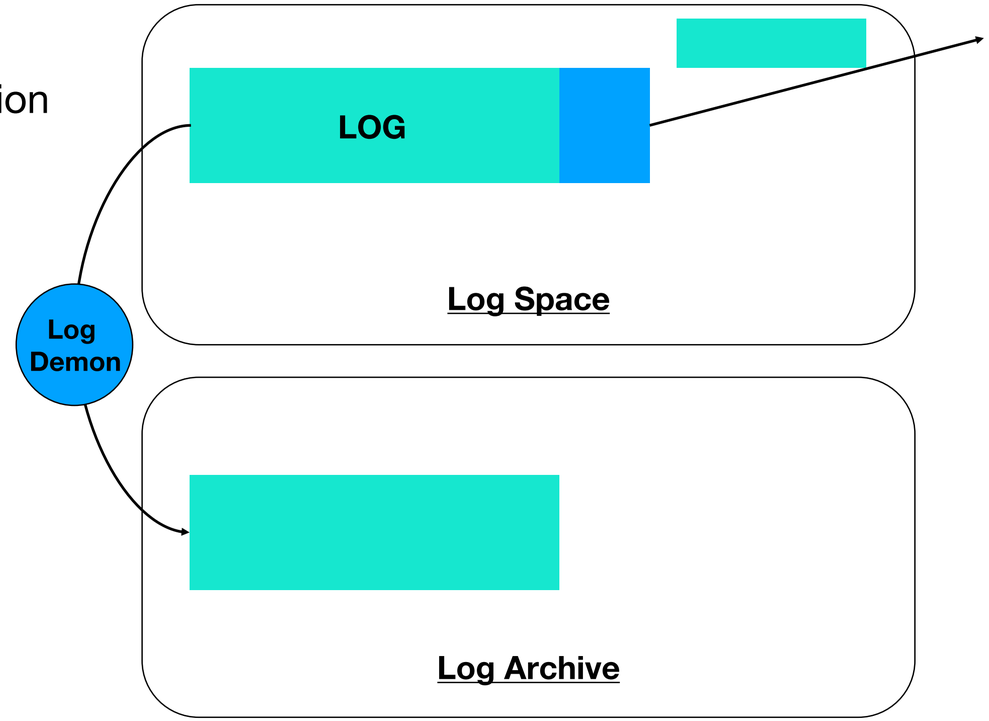
- ***Log anchor :*** this data might be stale - maximum LSN (log_anchor.lsn) must be recomputed
- Fragmented log records : invalidate fragmentary records

# Archiving the Log

**Q. How much of the Log Table should be online?**

**A. *Low-water mark***

1. Estimate log space utilization
2. Stop to begin transaction
3. Copy log to archive
4. Free log




**Log runs out of space**

- UNDO records involved in aborting transaction
- System restart

# Low water mark LSN

UNDO log records of live transactions must be kept online

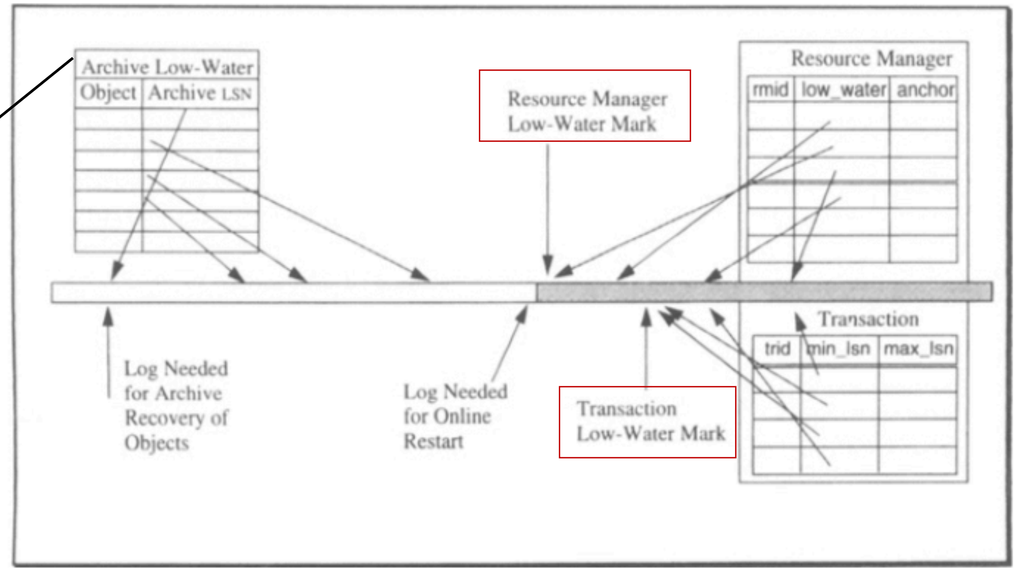
**Transaction Low-water mark =** $min(Trx \rightarrow min\_lsn)$

At restart, REDO changes committed + UNDO uncommitted changes

**Resource Low-water mark =** $min(Resource\_manager \rightarrow low\_water\_lsn)$

**Restart Low-water mark =** $min(Resource\ manager\ low-water\ mark\ , Trx\ low-water\ mark)$

Trx low-water mark lsn @ most recent archive copy of object began



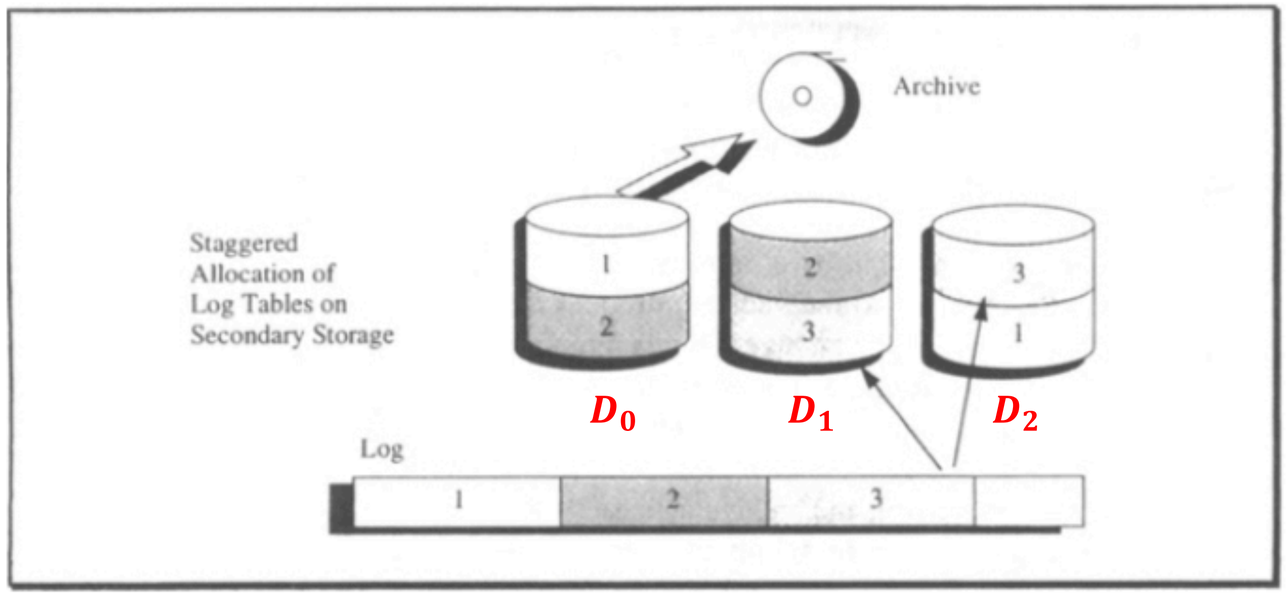

**Various Low-water mark**

# Staggered Allocation

**Log bottleneck :** If the file being archived resides on same disk as the current LSN

- Read & Write to current log file vs. Archive transfer

$$Log\ Table \leftrightarrow i^{th}\ Log\ file \qquad (i+1) mod\ Num_{disk}$$



# References

[1] Jim Gray and Andreas Reuter, “Transaction Processing: Concepts and Techniques”, Morgan Kaufmann, San Mateo, CA (1993)

[2] Stanford CS Theory CS 361A handout (http://theory.stanford.edu/~rajeev/cs361.html)