Redo Log Removal Mechanism for NVRAM Log Buffer

2009/10/20

Kyoung-Gu Woo, Heegyu Jin
Advanced Software Research Laboratories
SAIT, Samsung Electronics

SAMSUNG

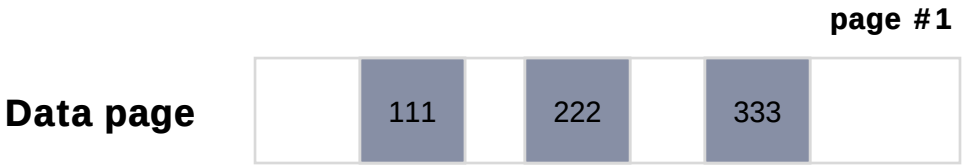
Contents

SAMSUNG

- Background
- Basic Philosophy
- Proof of Correctness
- Conclusion

Background

- Database Log
  - History of actions executed by DBMS
  - Contains REDO and UNDO information for every update
  - Typically written to log disk in a sequential manner
- Log record
  - <TxnID, pageID, offset, length, old image, new image>



Log page

prevLSN	TxnID	pageID	offset	length	Before img	After img
null	T1	#1	10	3	000	111
	T2	#1	20	3	000	222
	T3	#1	30	3	000	333

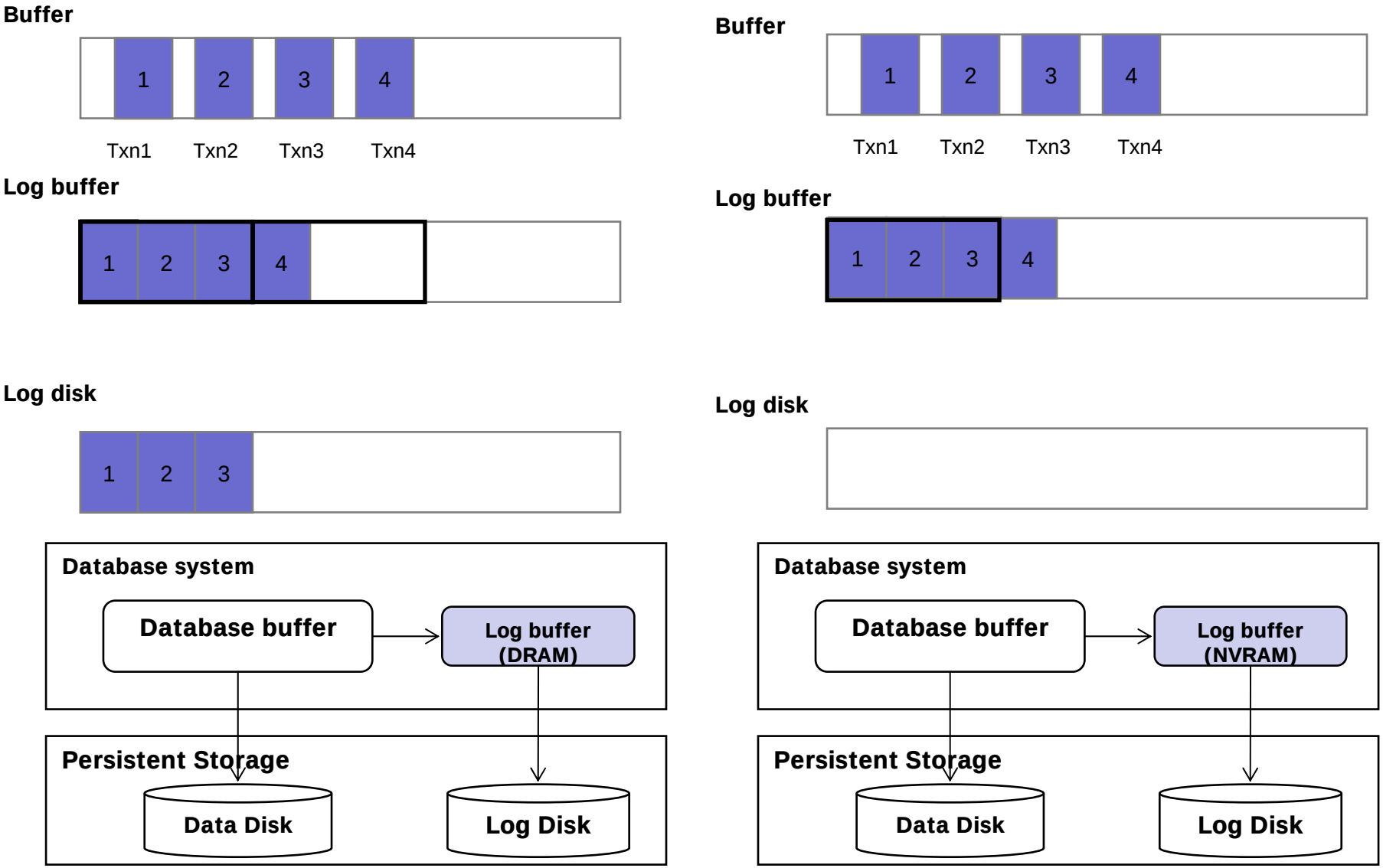
Background

- Write Ahead Logging (WAL) Policy
  - Log records must be written to disk prior to its corresponding data page
  - All log records should be written to disk when Transaction commits
- WAL + steal/no-force combination
  - Updated pages can be written to disk before commit
  - Updated pages are not necessarily written to disk at commit time

  → The log write overhead is important to transaction performances

  → Logging performance becomes critical in MMDB
- Group commit
  - Flushes log buffer when a group of transactions have committed
  - Reduce disk I/O → increase throughput
  - At the cost of response time penalty to some transactions

Introduction of NVRAM as the Log Buffer



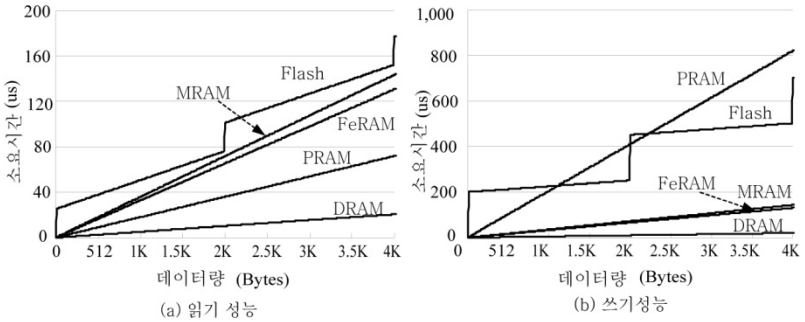
Benefits and Challenges of NVRAM Log Buffer

SAMSUNG

- Benefits
  - Unnecessary writes
    - Usually, the amount of log produced by on OLTP transaction is very small
    - The portion of unnecessary log writes = (A+C)/(A+B+C)

    → Byte addressability of NVRAM log buffer gives you better response time
  - Interesting question:

    What if size (B) = page size ?



- Problem
  - NVRAM log buffer is expected to be smaller in its capacity than log disks for a while
  - Periodic log flush from NVRAM log buffer to disk can be a new bottleneck point when the transaction rate is very high

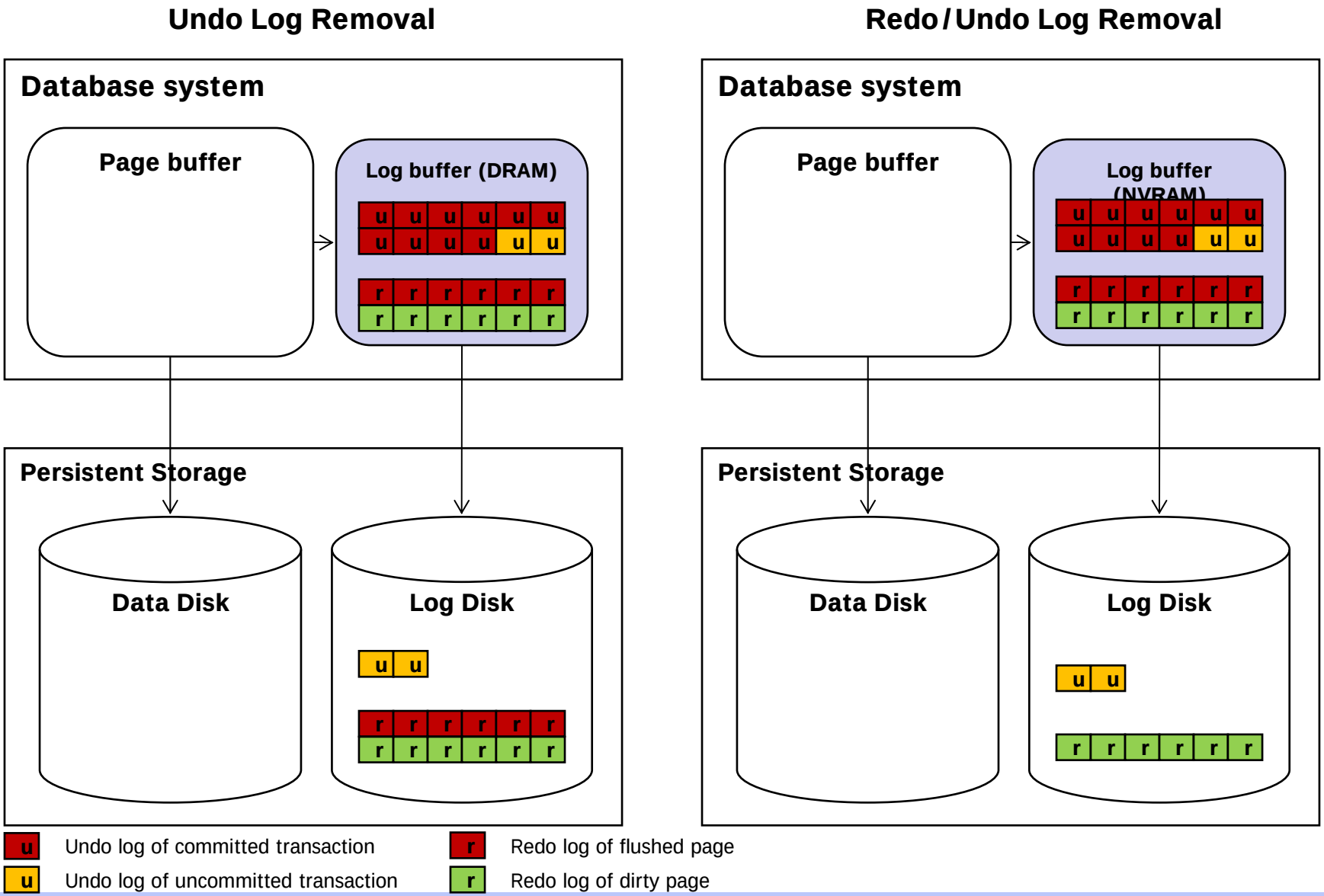
Possible Solutions against New Bottleneck with NVRAM log Buffer

- More frequent flushing to log disk (Background process)
  - Sequential write speed is the key
    - Works for PRAM (problem solved!!!)
    - Does not work for MRAM or FeRAM (other solutions are needed)
- Deploying parallelism
  - Use log disk array
  - Works good but at additional cost
- Reduce the amount of log flushed to log disk from log buffer
  - Remove unnecessary log records in NVRAM
    - Undo Log Removal [Dewit84]
      - Remove undo log records after the transaction is committed
      - Already proposed for main memory DBMS
    - How about Redo Log Removal ??
      - We are going to talk about this now !!!

Contents

- Background
- Basic Philosophy
- Proof of Correctness
- Conclusion

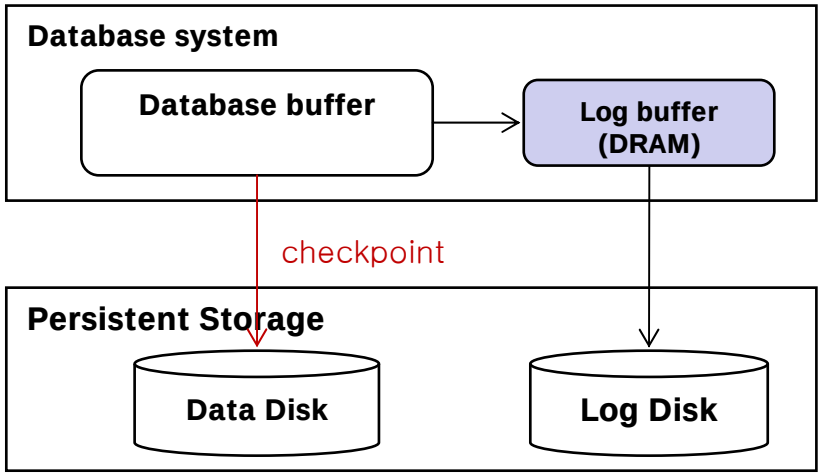
Log Removal in Log Buffer



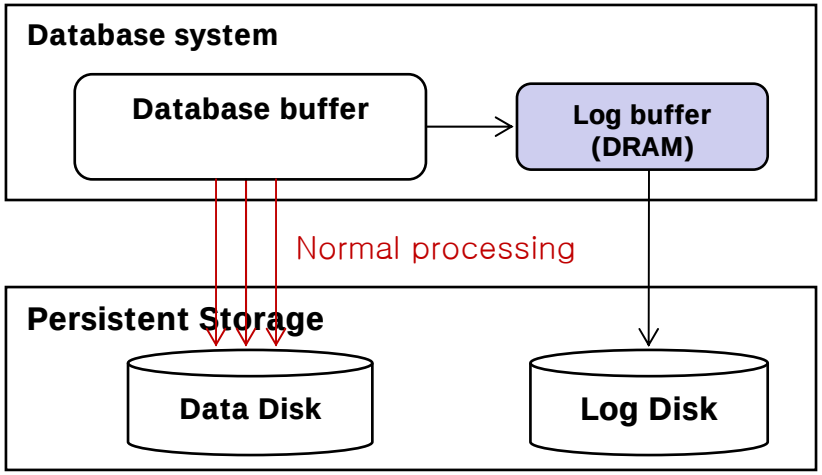
Redo Log Removal?

SAMSUNG

- What is Redo log for?
  - Basically Redo log is used to preserve information in dirty data pages in buffer
    - Unlike Undo log, Redo log is only used in failure recovery
  - Once a dirty data page is forced to disk, the page's corresponding redo logs have no use
    - If that simple, why didn't they try it in MMDB?



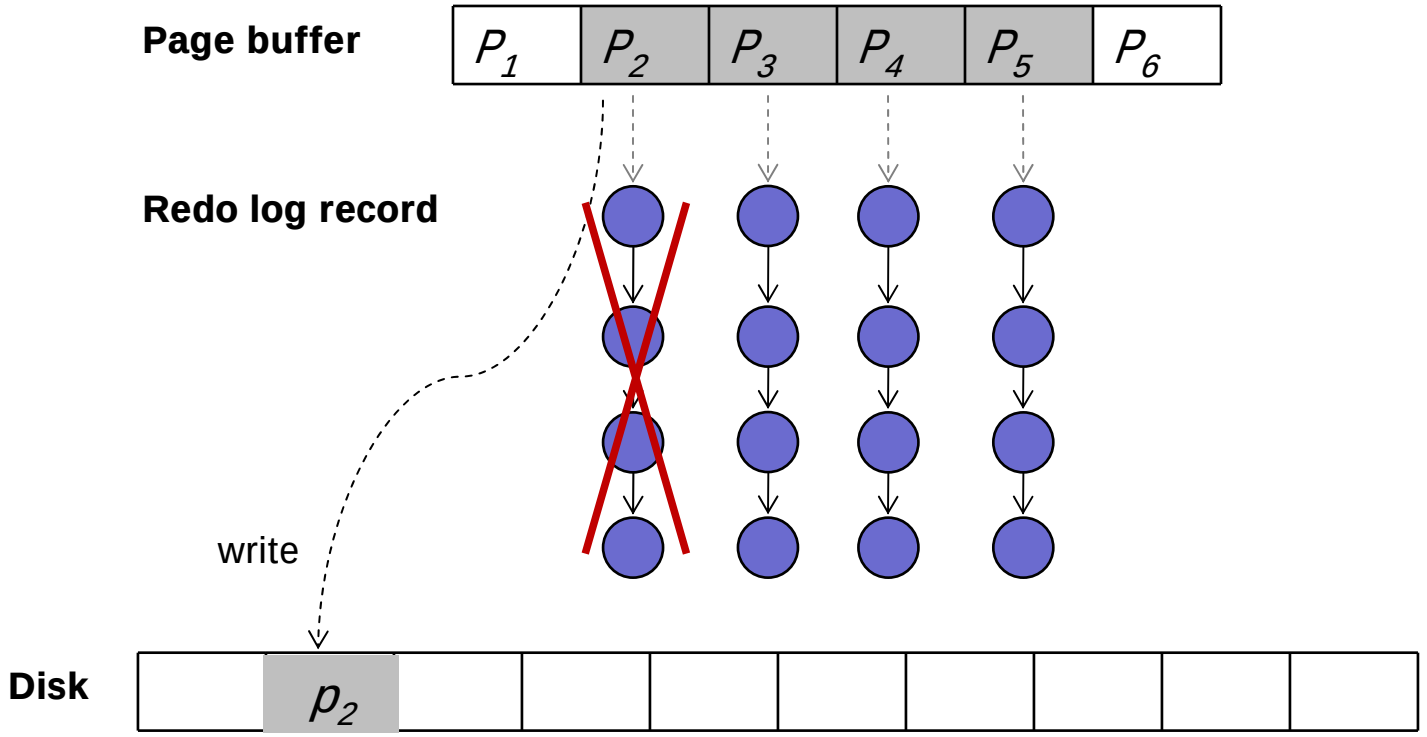
Typical MMDB:
Except checkpoint, data pages are not written to disk



DRDB:
data pages are written to disk during normal processing

Redo Log Removal: Base Philosophy(1/2)

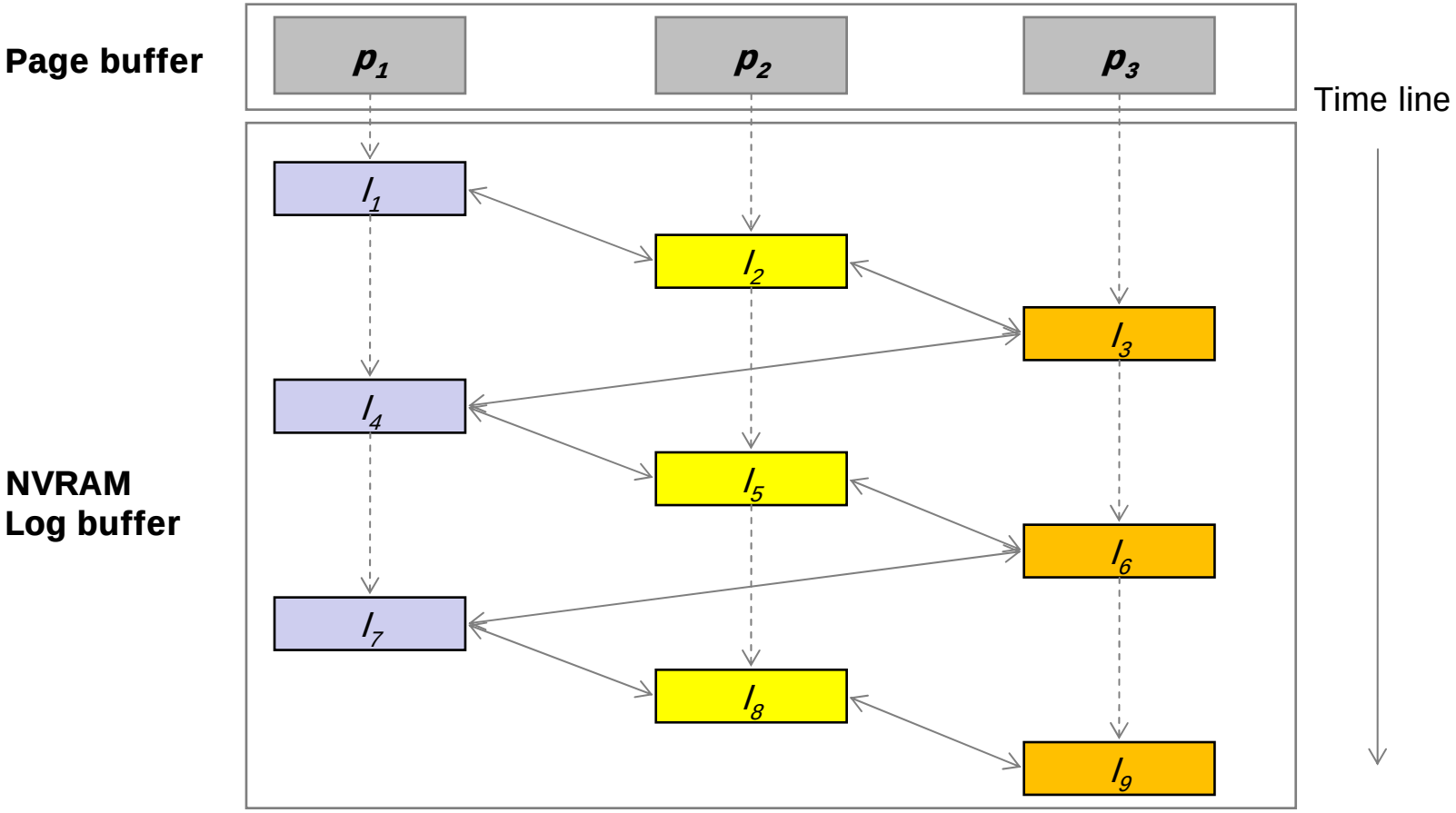
- Redo Log Removal
  - When a data page is flushed to disk, redo log records for the data page are removed.



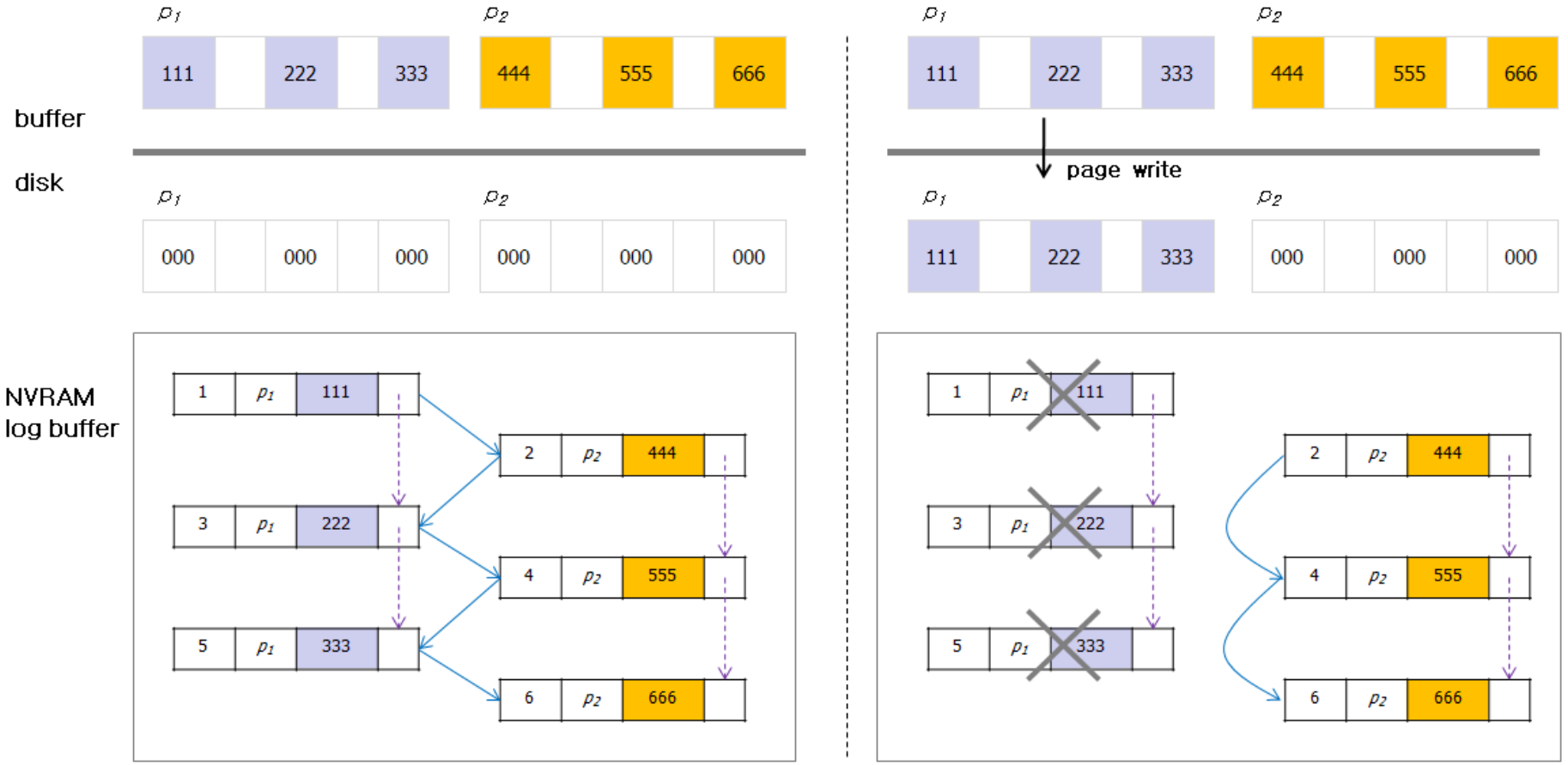
Redo Log Removal: Base Philosophy(2/2)

SAMSUNG

- Log Records are stored in NVRAM not in a sequential manner
  - Log records are connected by pointers
  - for the purpose of removing log record efficiently



Redo Log Removal: Example Case



Before redo logs are removed

After redo logs are removed

Contents

SAMSUNG

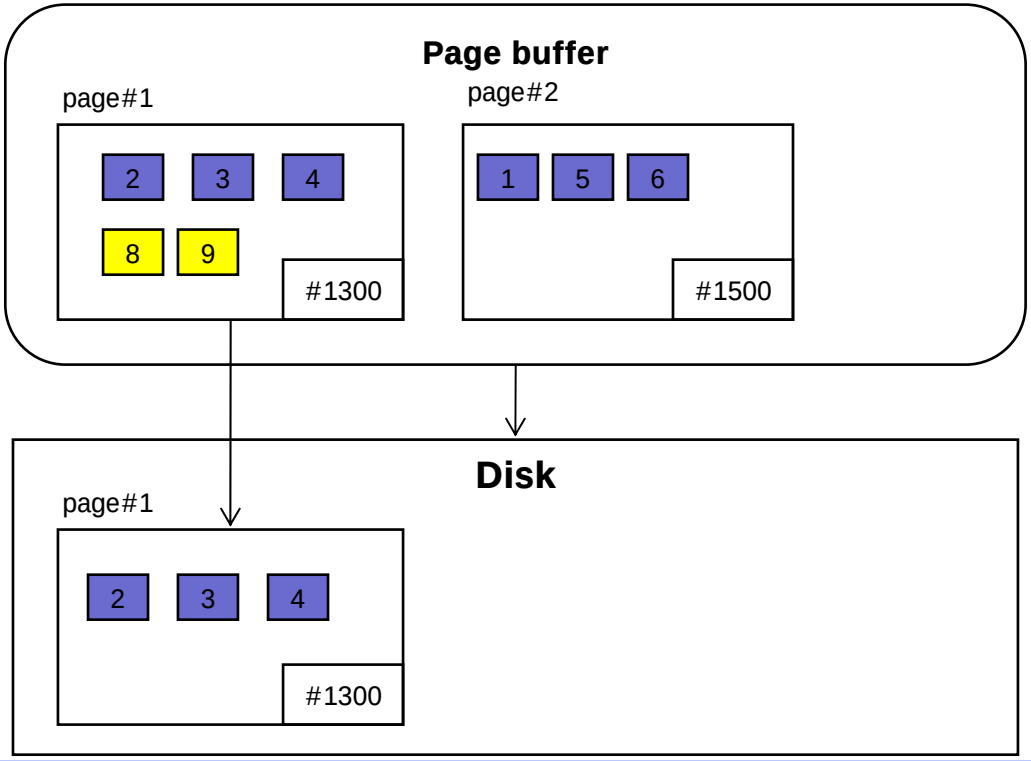
- Background
- Basic Philosophy
- Proof of Correctness
- Conclusion

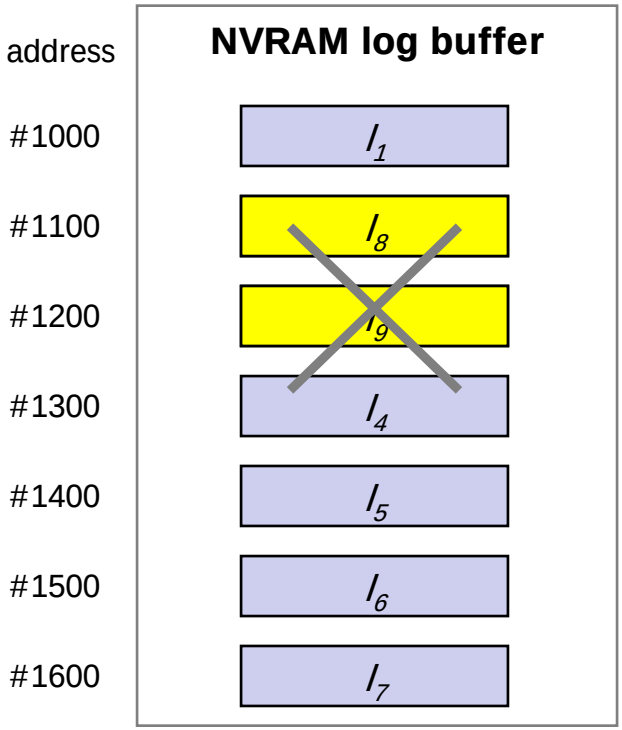
Proof of Concept: Two Key Issues in Redo Log Removal

- Unordered LSN
  - LSN is usually the start address of log records in physical layout
    - i.e., physical LSN is commonly used
  - In NVRAM log buffer, the start address of log record is random
    - i.e., physical LSN is unordered in NVRAM log buffer
- Log duplication
  - Crash can occur during periodic log flush
  - As a result, same log records can be existing both in log buffer and disk
  - The problem is to stitch two different log sequences seamlessly

Unordered LSN issue

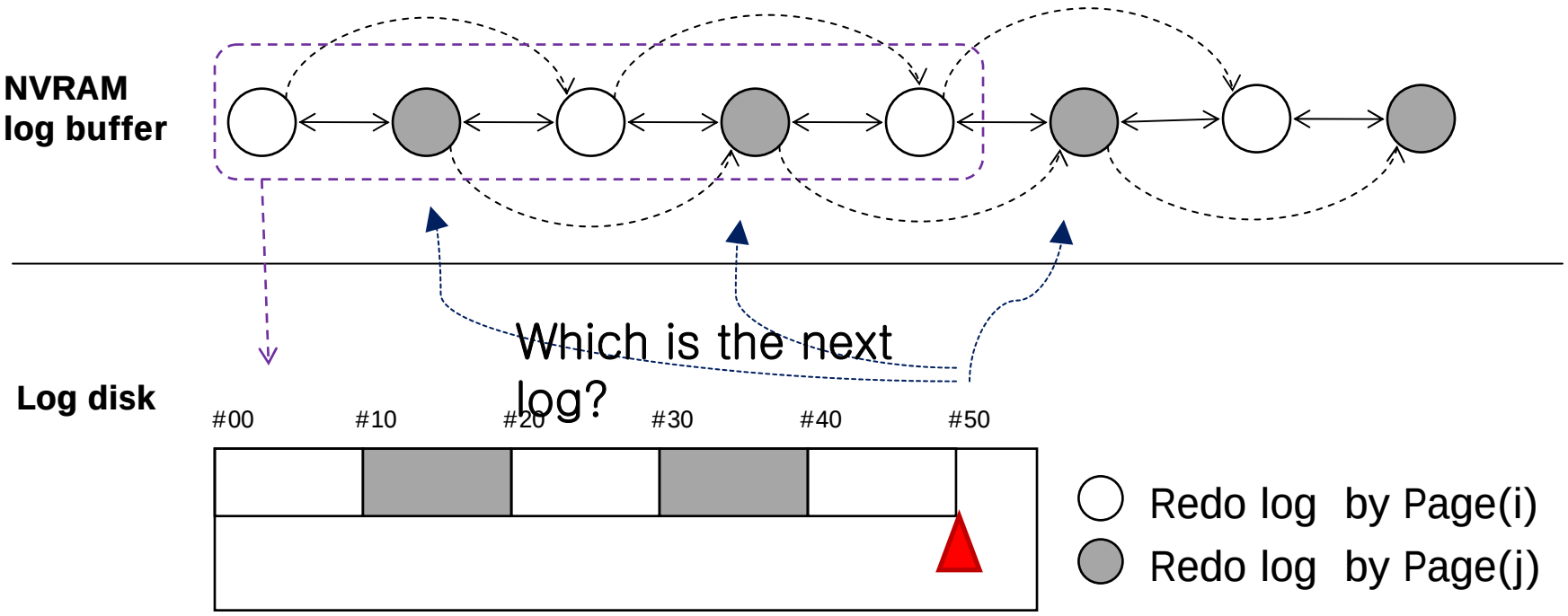
- In failure recovery, a data page's LSN(pageLSN) is compared to log records' LSNs
  - When a Log record's LSN is larger than the pageLSN, redo is required
- However, Address of log record in NVRAM doesn't follow chronological orders!





Log duplication issue

- System crashes before it removes duplicate log records
- The junction point between log buffer and log disk is lost
  - Physical LSN(Log Sequence Number) is used in Disk
  - Random Memory Address is used to identify log records in NVRAM



Two Issues, One Solution

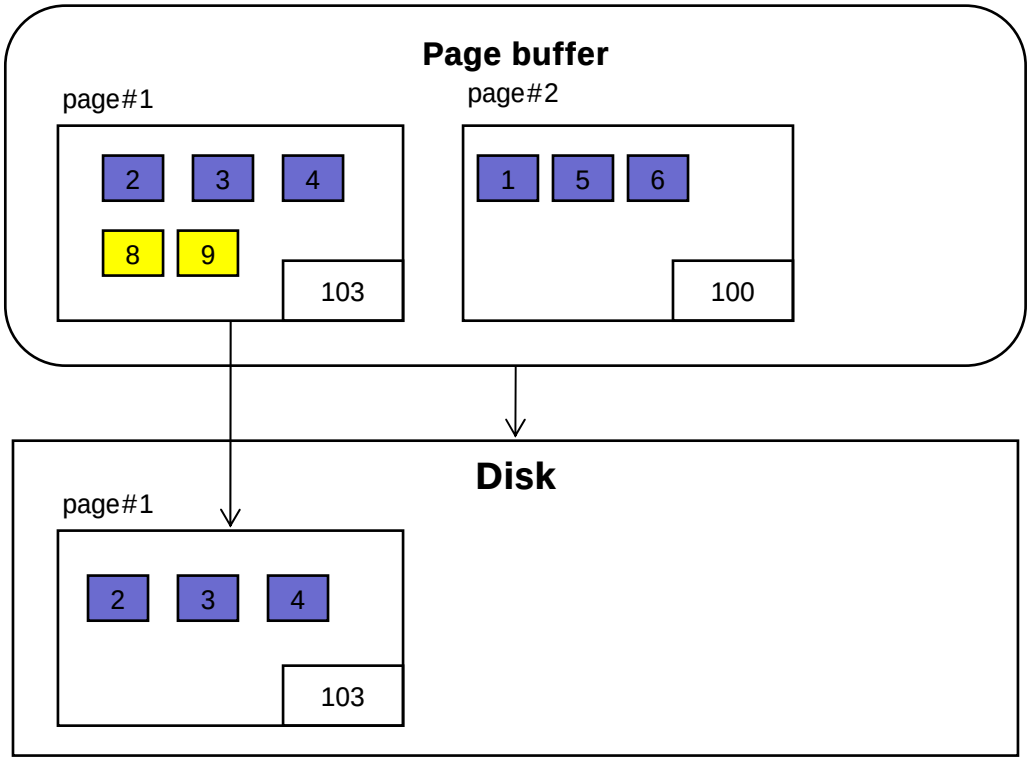
SAMSUNG

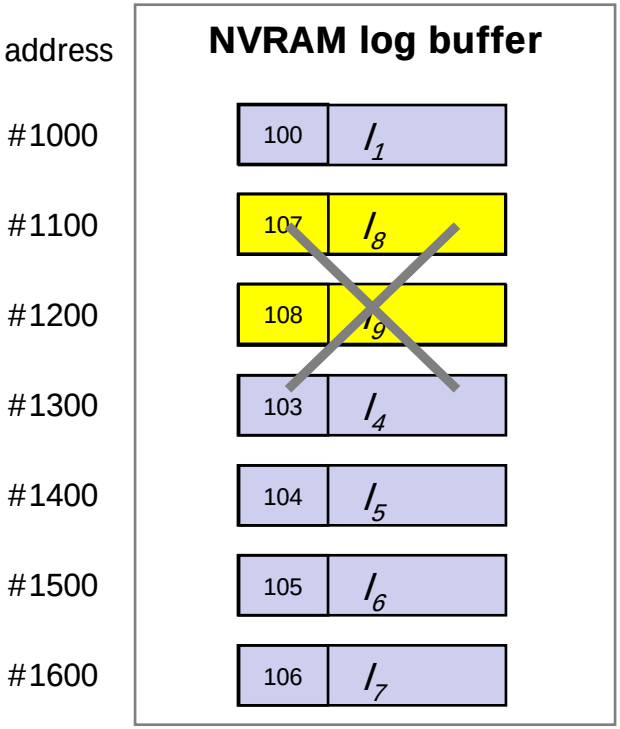
- Let's use Logical LSN than Physical LSN
  - Physical LSN in Disk is increasing monotonically
  - We embed a logically increasing number inside each log record

So Simple!! In fact, Too Simple...

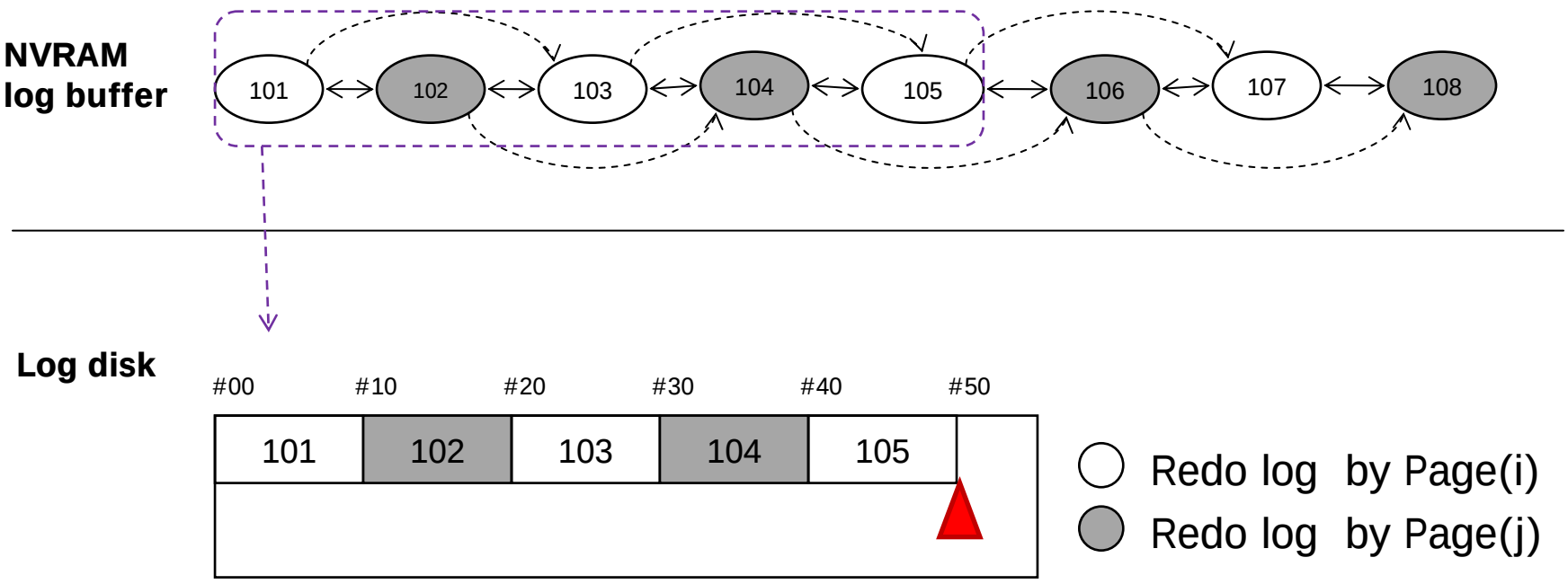
Unordered Physical LSN → Ordered Logical LSN

- Same Logic with one more comparison(physical LSN) works





Duplicated Log Identified by Logical LSN



Contents

SAMSUNG

- Background
- Basic Idea
- Proof of Correctness
- Conclusion

Conclusion

SAMSUNG

- We proposed a redo log removal mechanism
  - It is not a complete research yet
  - But, it seems to be working
  - And, we think that the performance of NVRAM log buffer will be enhanced for OLTP applications

- Limitations for now
  - Correctness is not proven for record-level locking,,, sadly
    - Which is commonly used in heavy OLTP systems
  - Overhead & Gain is not experimentally proven
    - Is periodic flushing really a bottleneck in real applications?
    - Does memory management overhead small enough?
      - Is disk speed slow enough? (especially sequential write throughput)

Appendix

SAMSUNG

Reference

- [DeWitt84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro,M. R. Stonebraker, and D. Wood. “Implementation Techniques for Main Memory Database Systems.” In Proceedings of the 1984 ACM SIGMOD International Conference on anagement of Data, pages 1–8, June 1984