FIRST IN CHANGE

NVWAL: Exploiting NVRAM in Write-Ahead Logging

남범석

(Beomseok Nam)

UNIST (울산과기원)

Outline

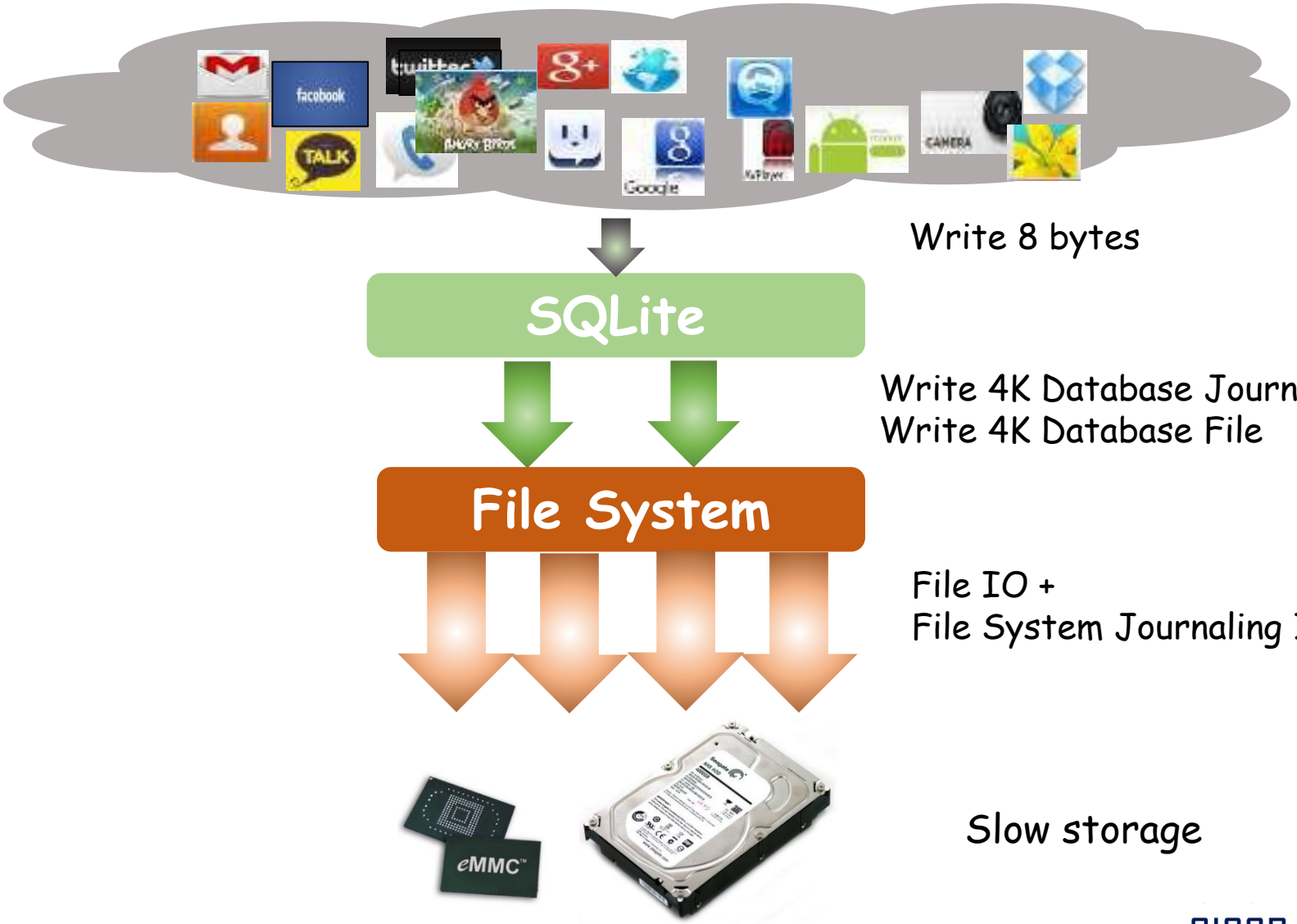
- Motivation
  - Write Amplification Problem in SQLite
- NVWAL: Write-Ahead-Logging on NVRAM [ASPLOS'16]
  - Byte-granularity differential logging
  - User-level NVRAM management for WAL
  - Transaction-aware lazy synchronization
- On-going Works:
  - Failure-atomic Slotted Paging for Persistent Memory
- Conclusion

Motivation

SQLite

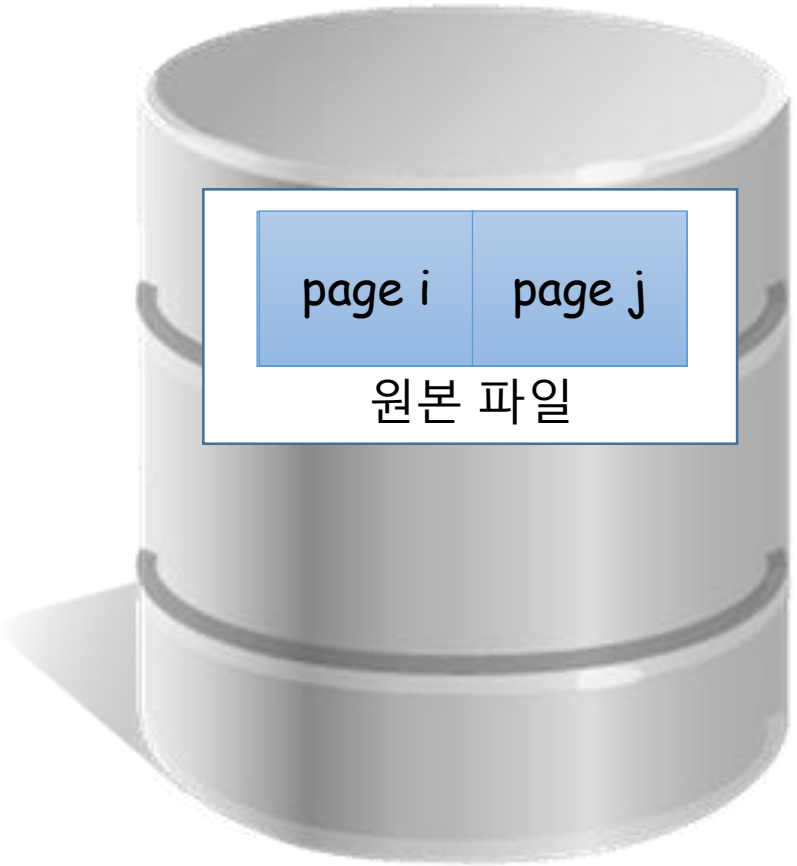
data

Write Amplification in SQLite

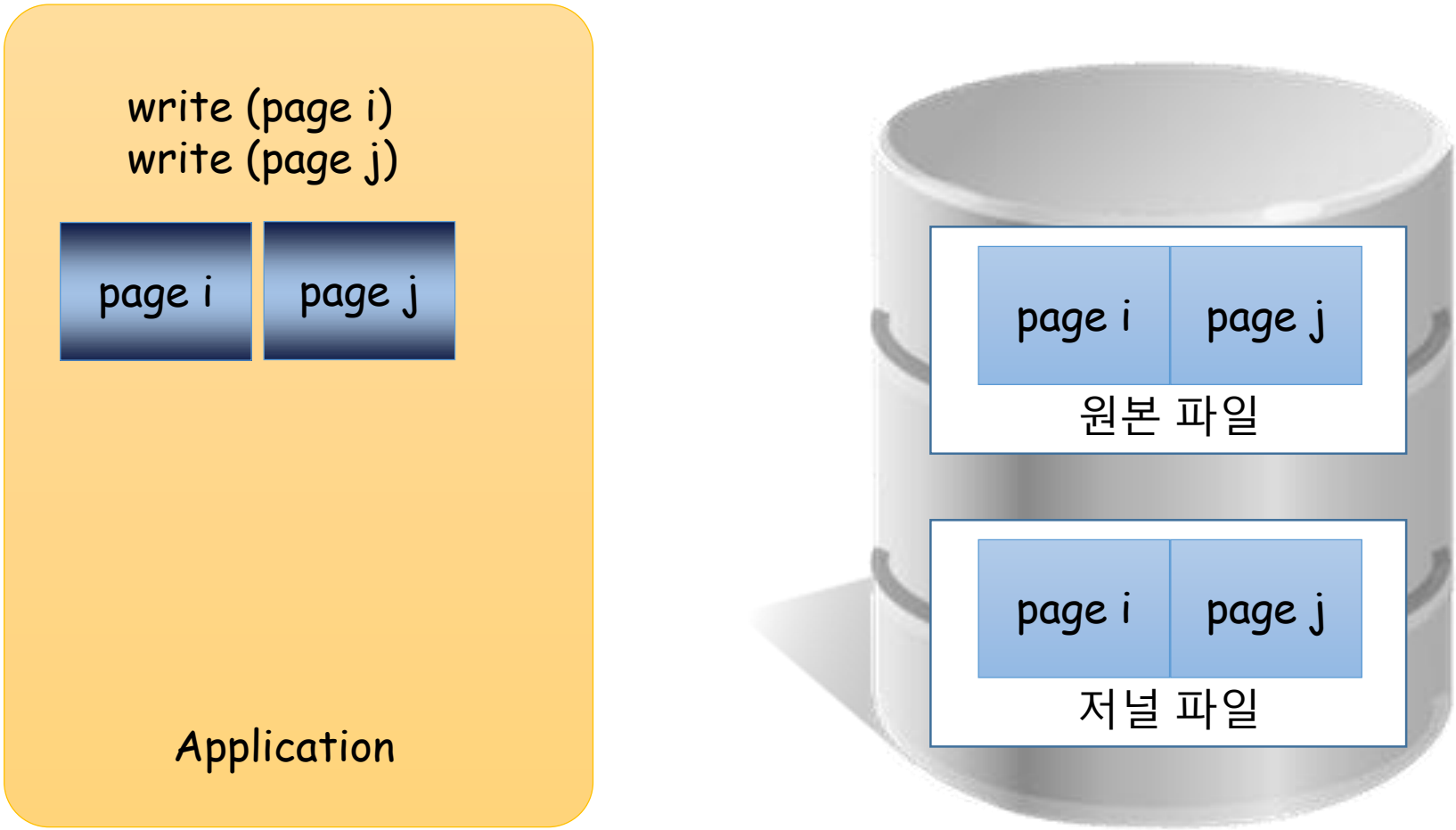


Rollback Journal

Application



Rollback Journal



Write-Ahead Logging in SQLite

Write 8 bytes

SQLite

Write 4K WAL File

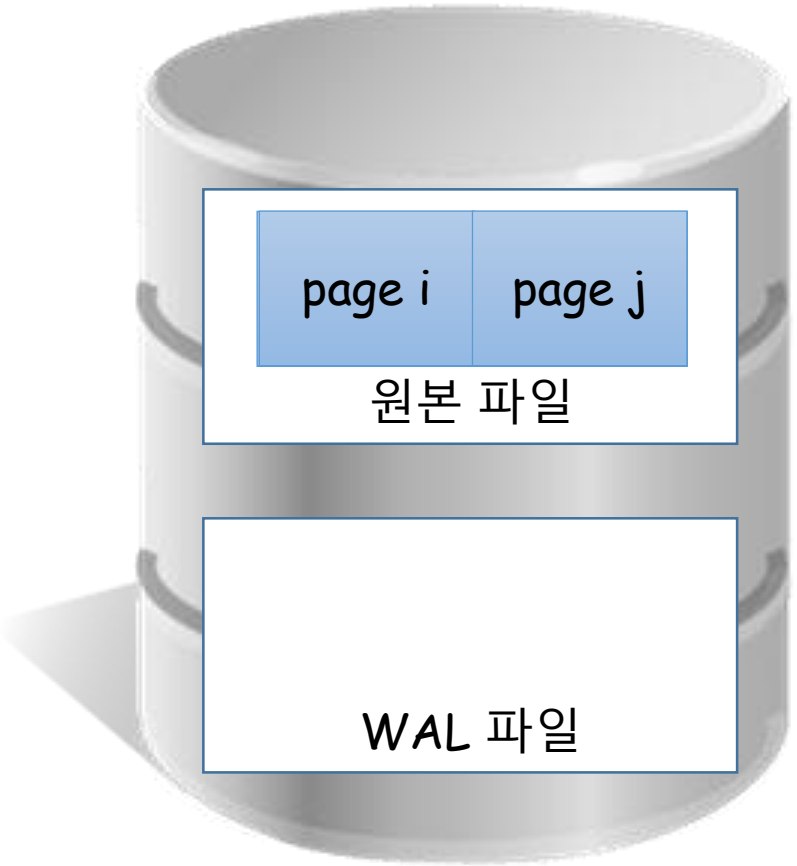
File System

File IO +
File System Journaling IO

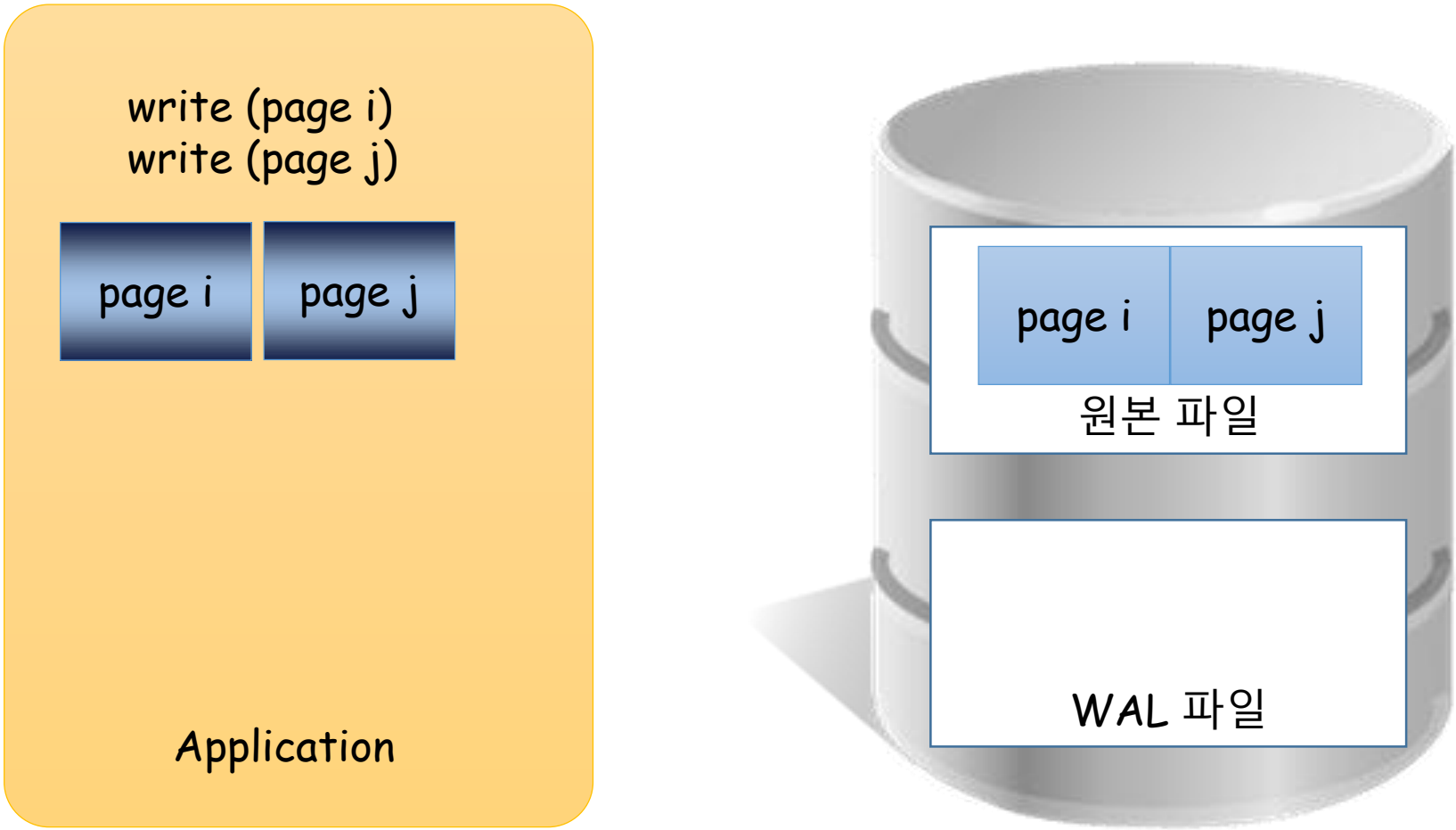
Slow storage

Write-Ahead Logging

Application



Write-Ahead Logging



Write-Ahead Logging in SQLite

Write 8 bytes

SQLite

Write 4K WAL File

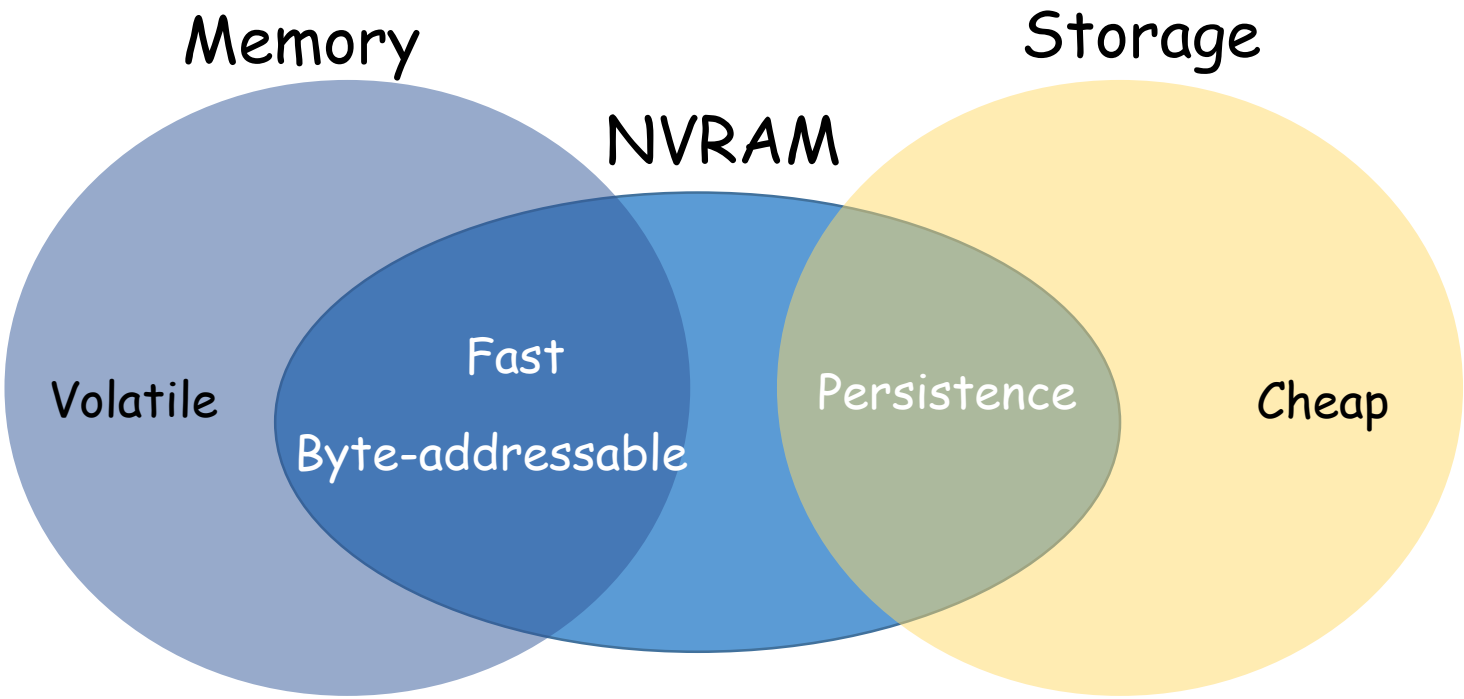
File System

File IO +
File System Journaling IO

Emerging Non Volatile Memory

Fast Storage

NVRAM



How?

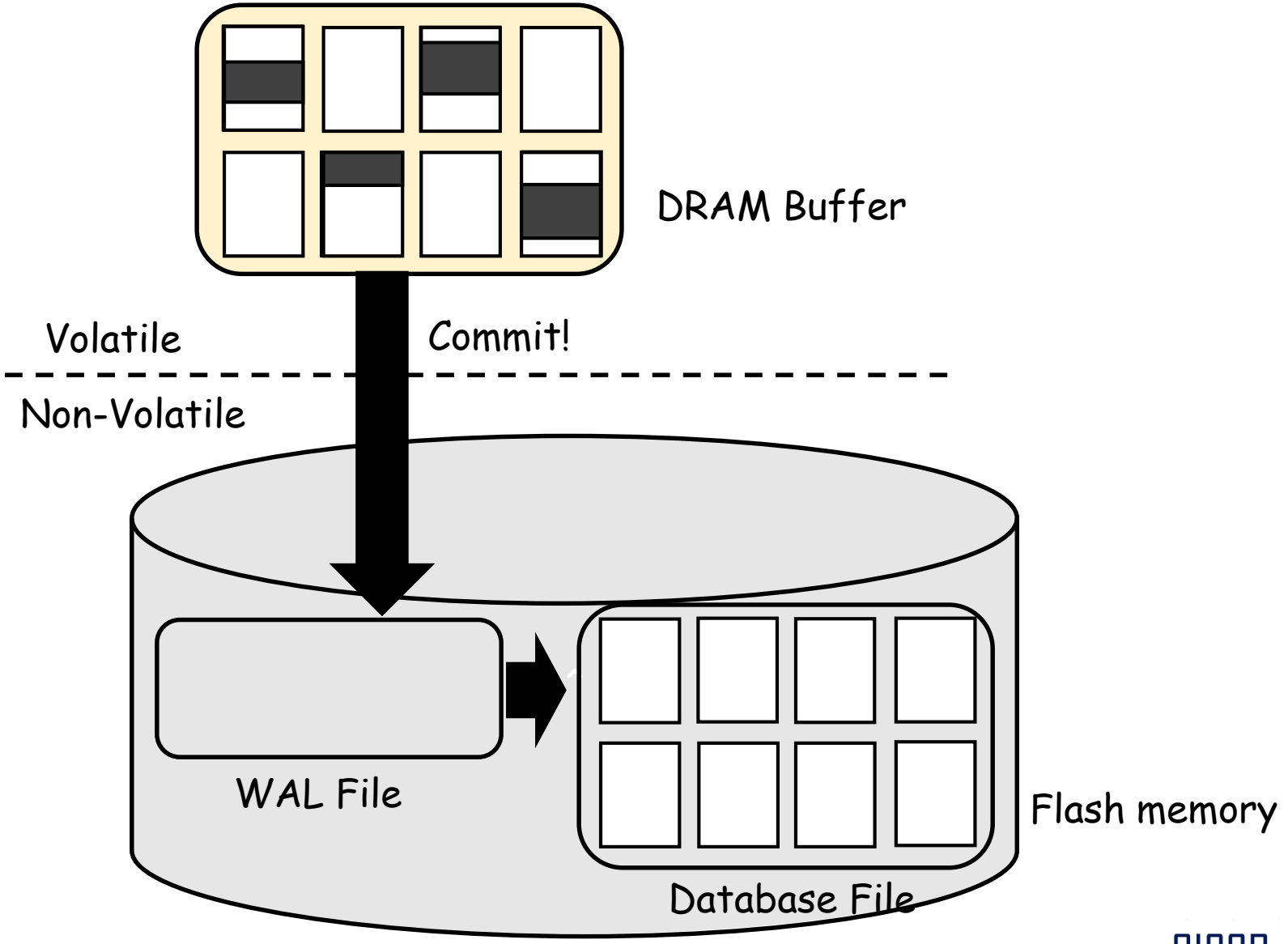
NVWAL: Exploiting NVRAM in Write-Ahead Logging

NVWAL (NVRAM Write-Ahead Logging)

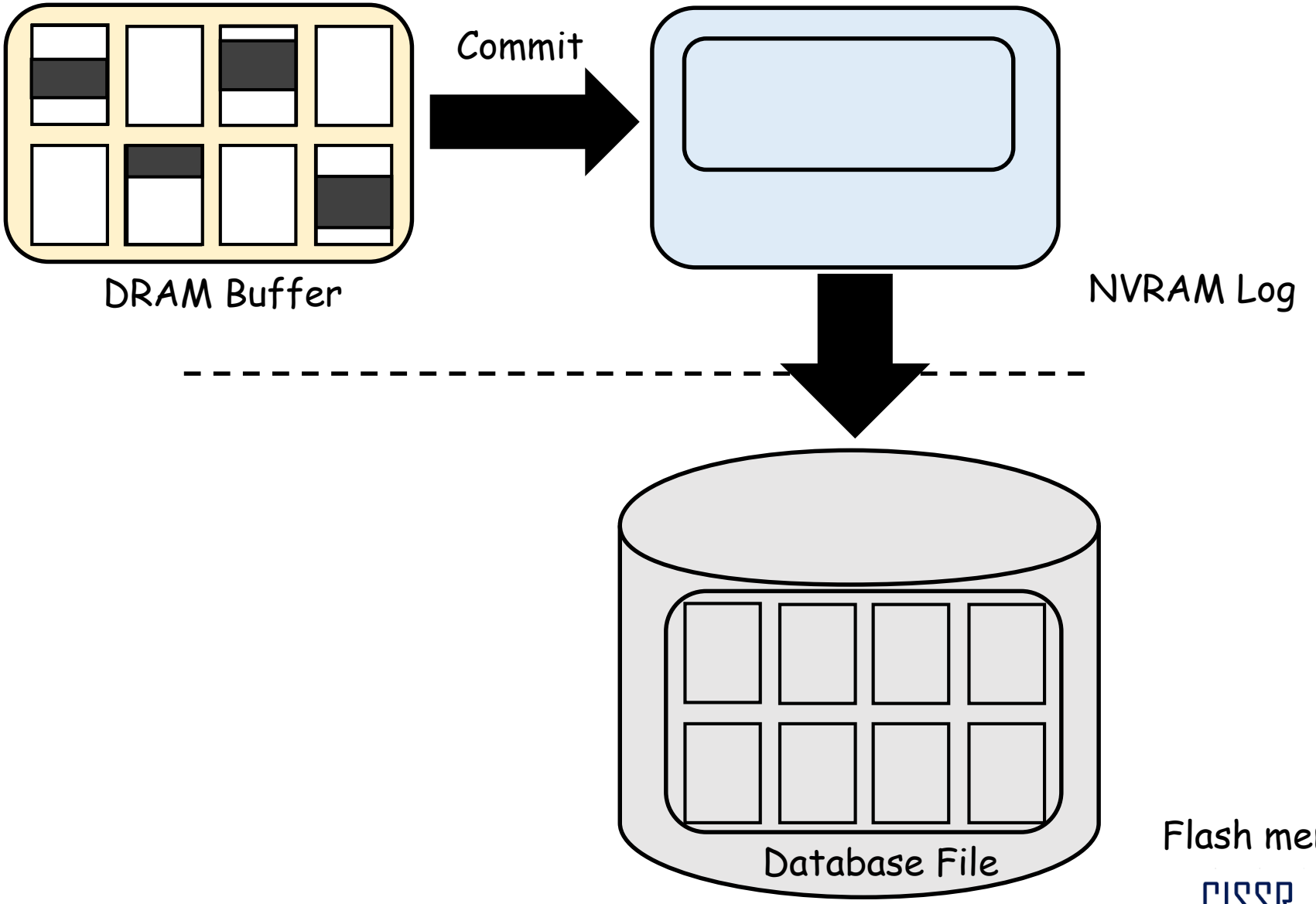
- Byte-granularity Differential Logging
- User-level Heap Management for WAL
- Transaction-aware Lazy Synchronization

Differential Logging

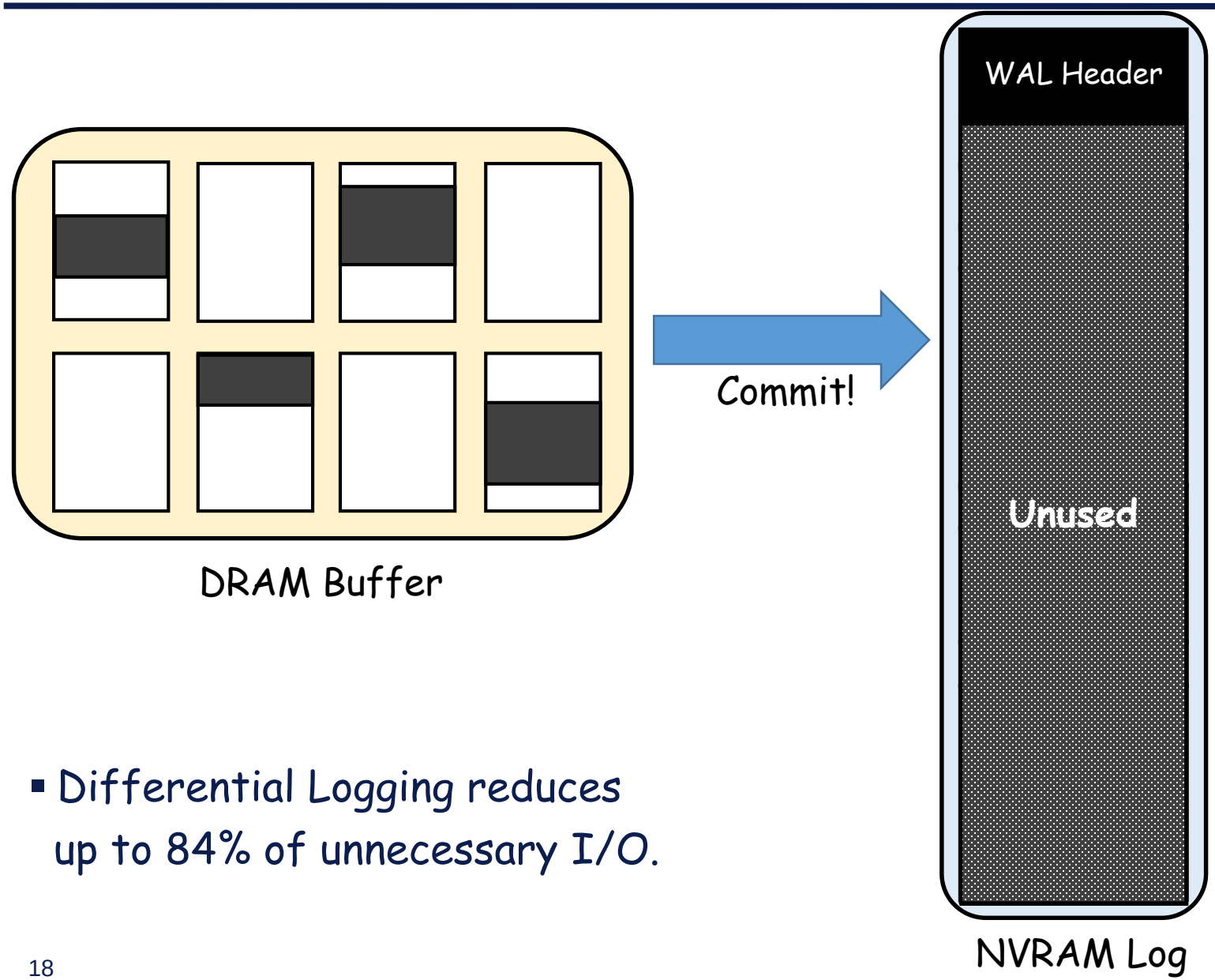
Write-Ahead Logging



Write-Ahead Logging in NVRAM



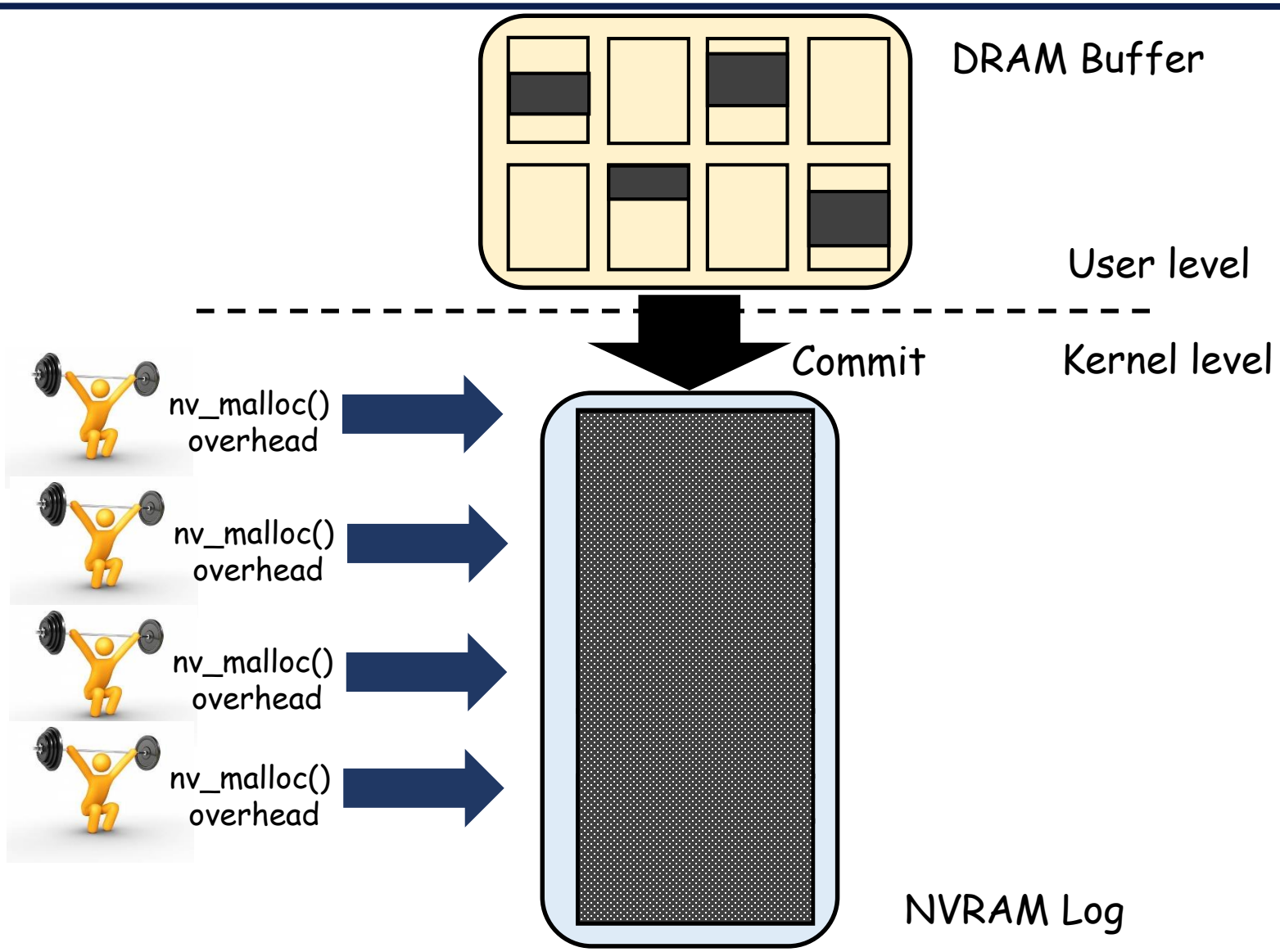
Differential Logging



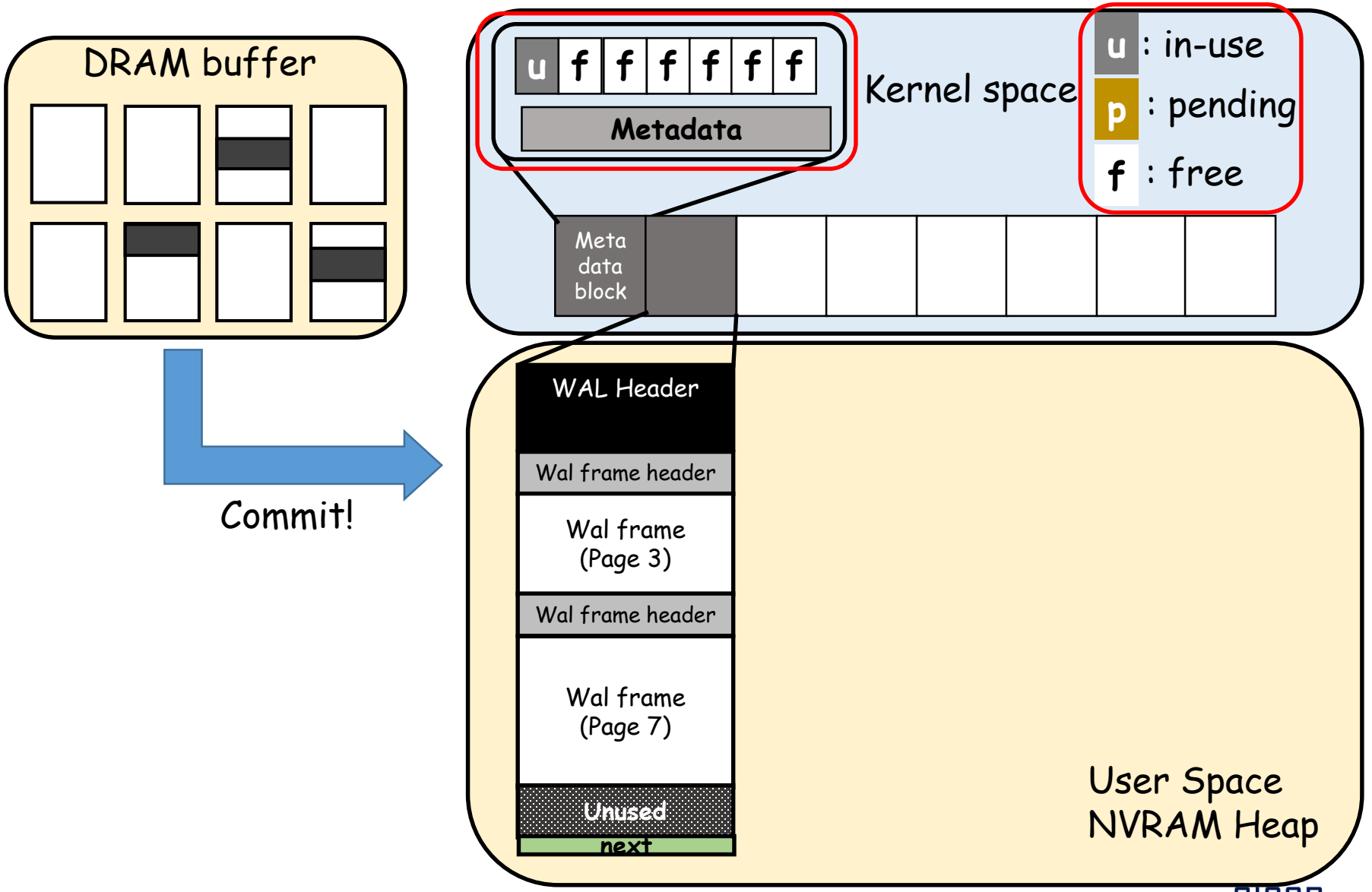
- Differential Logging reduces up to 84% of unnecessary I/O.

User-level Heap Management

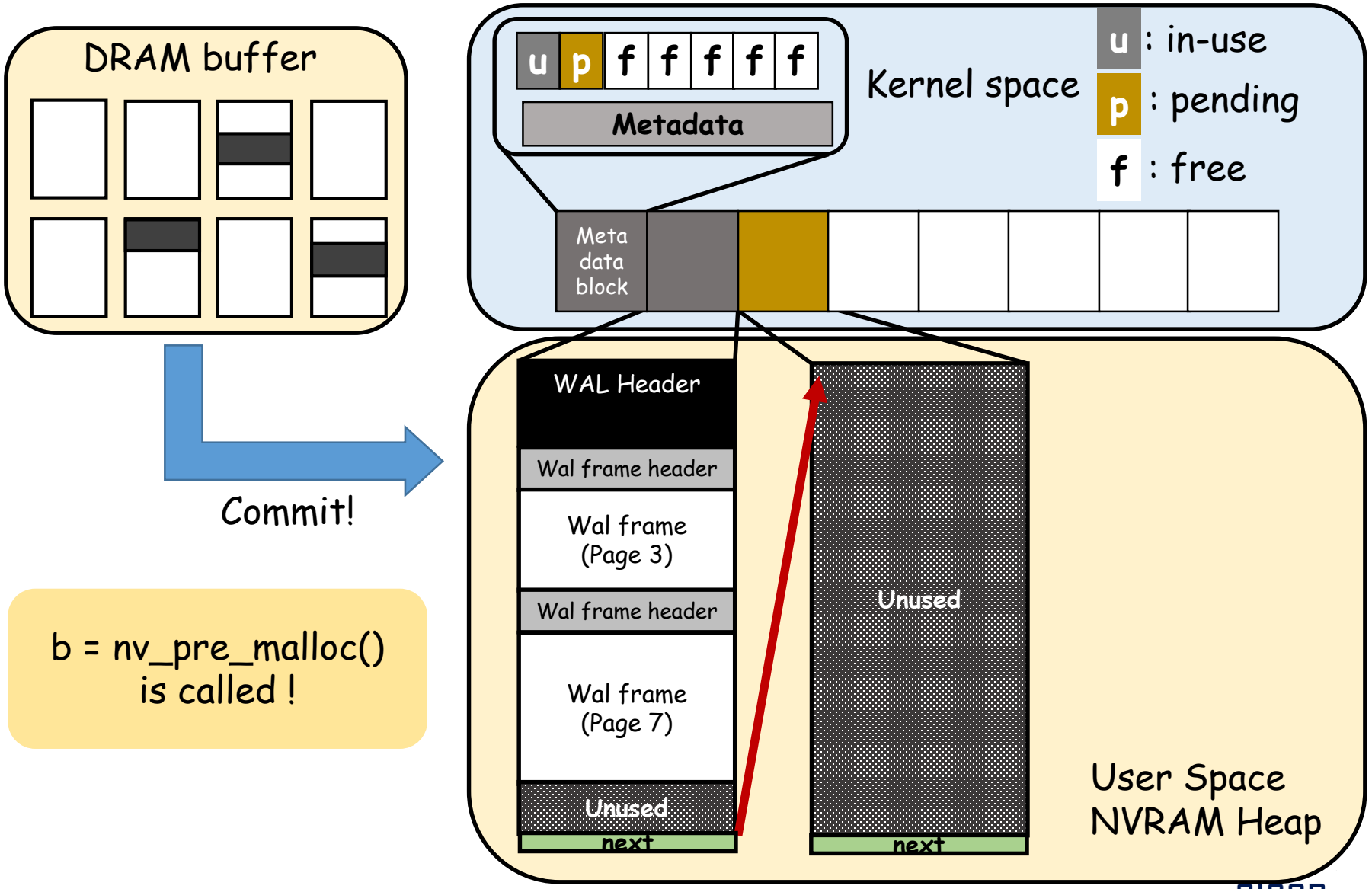
Block Management by NVRAM Heap Manager



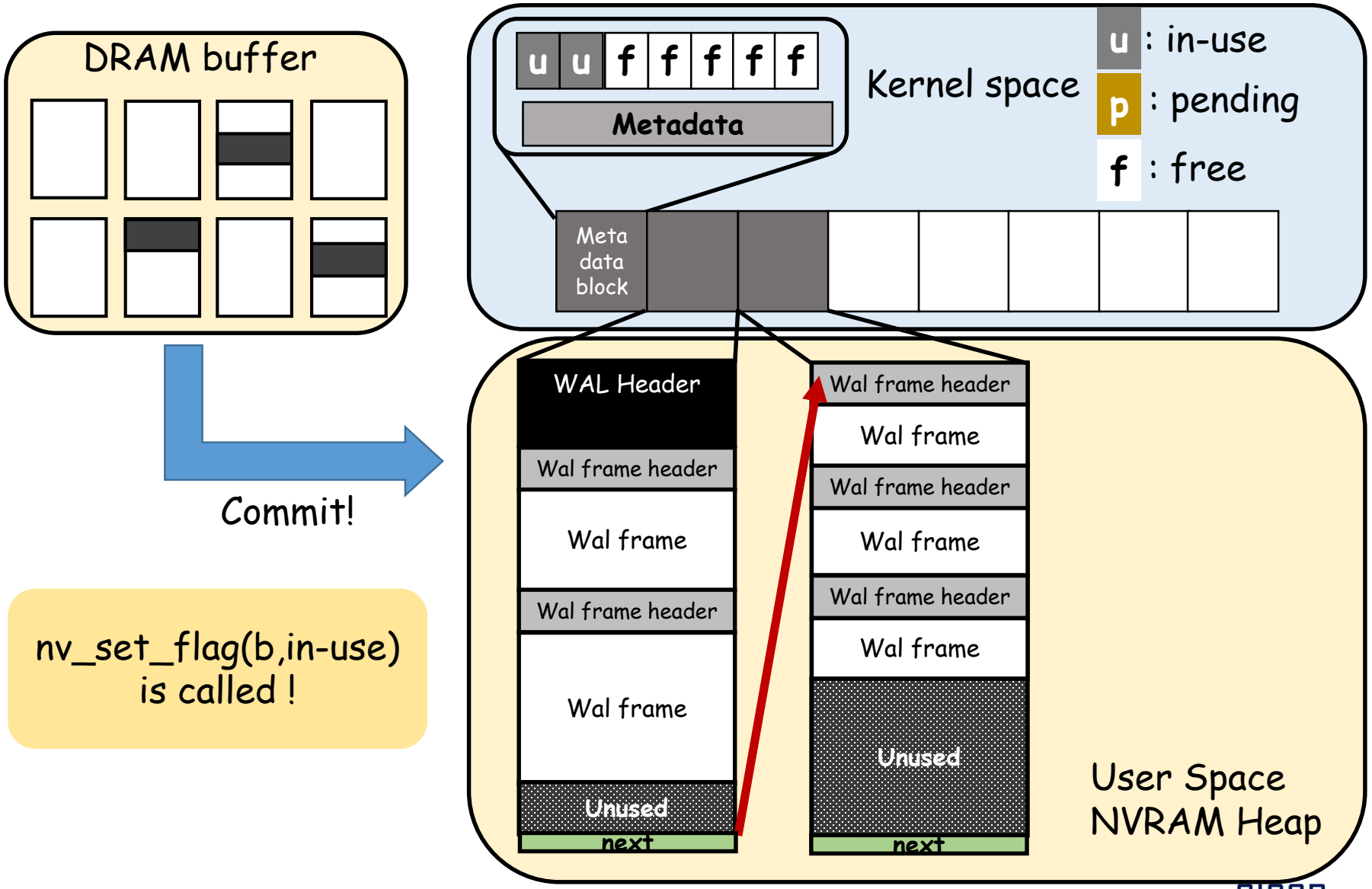
NVRAM User-level management



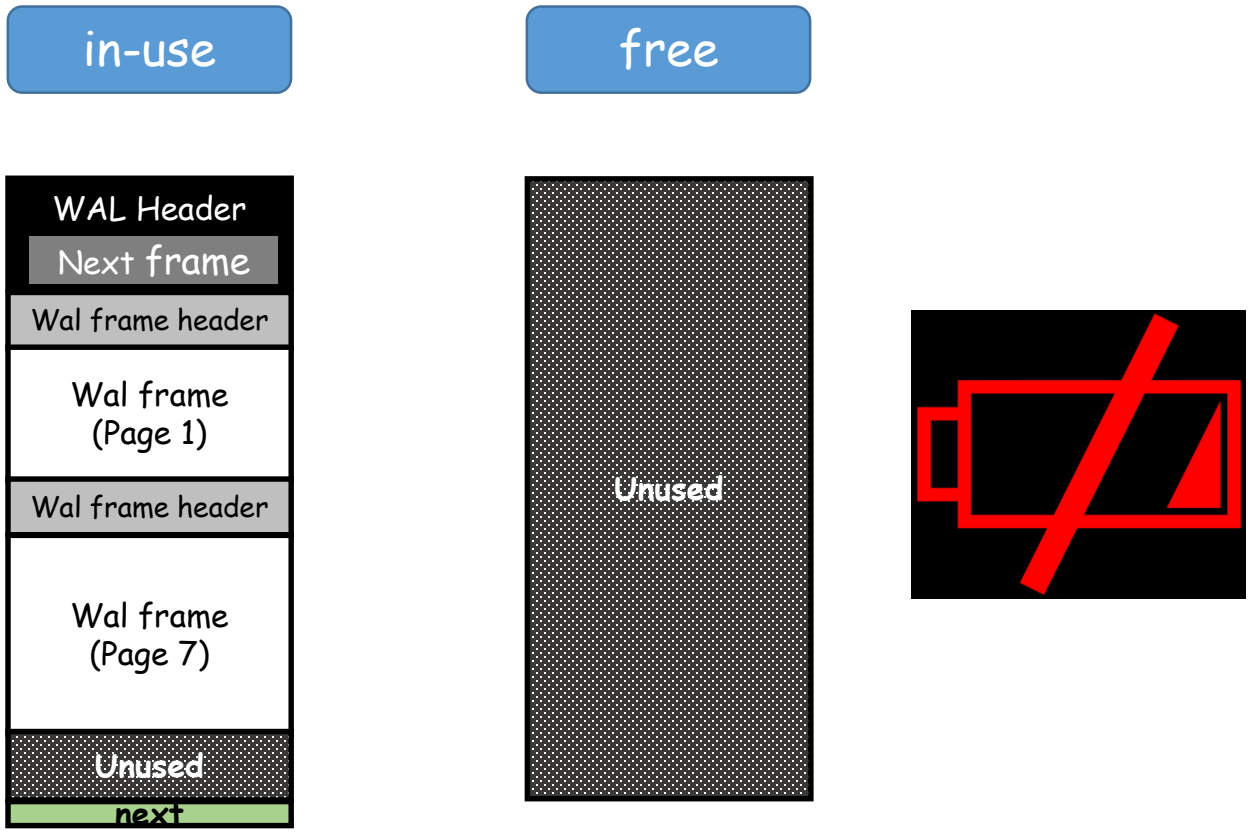
NVRAM User-level management



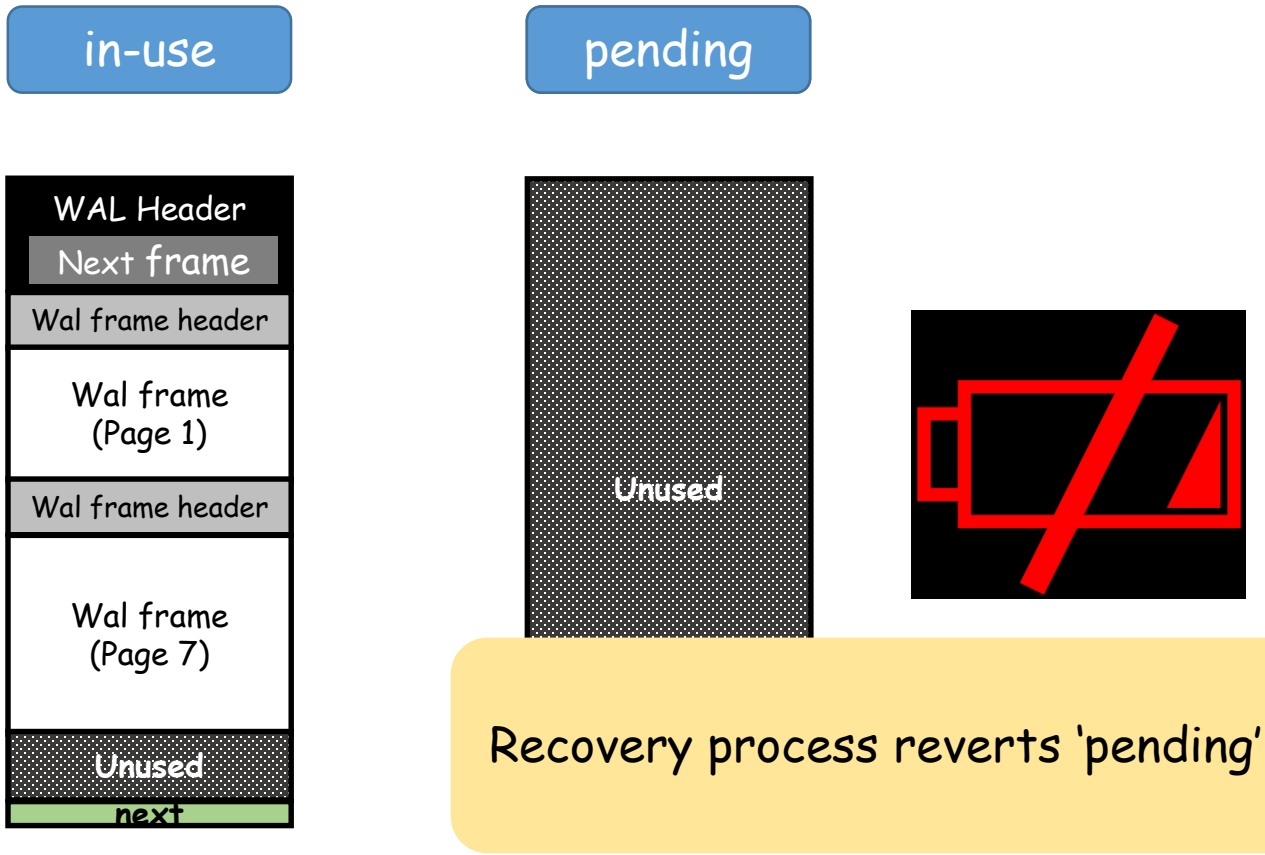
NVRAM User-level management



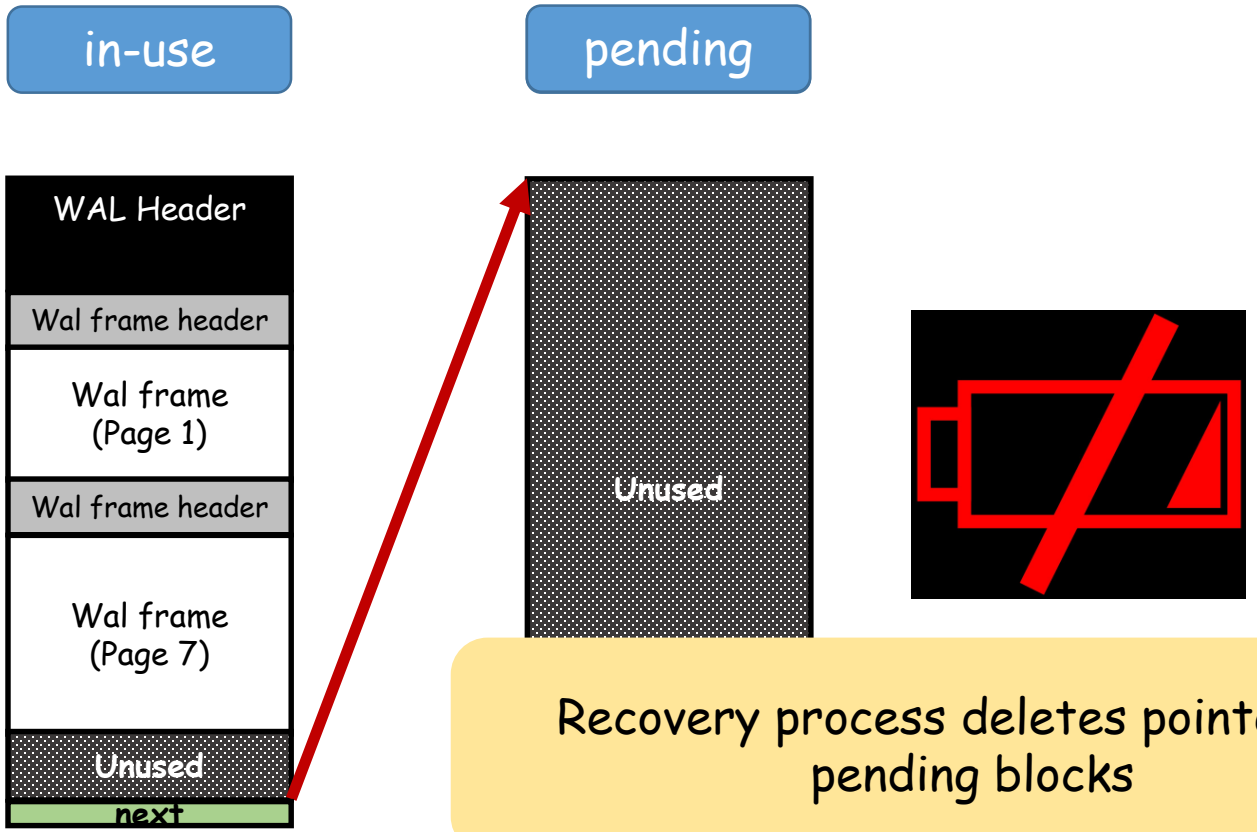
Recovery in NVWAL (1) – free



Recovery in NVWAL (2) – pending

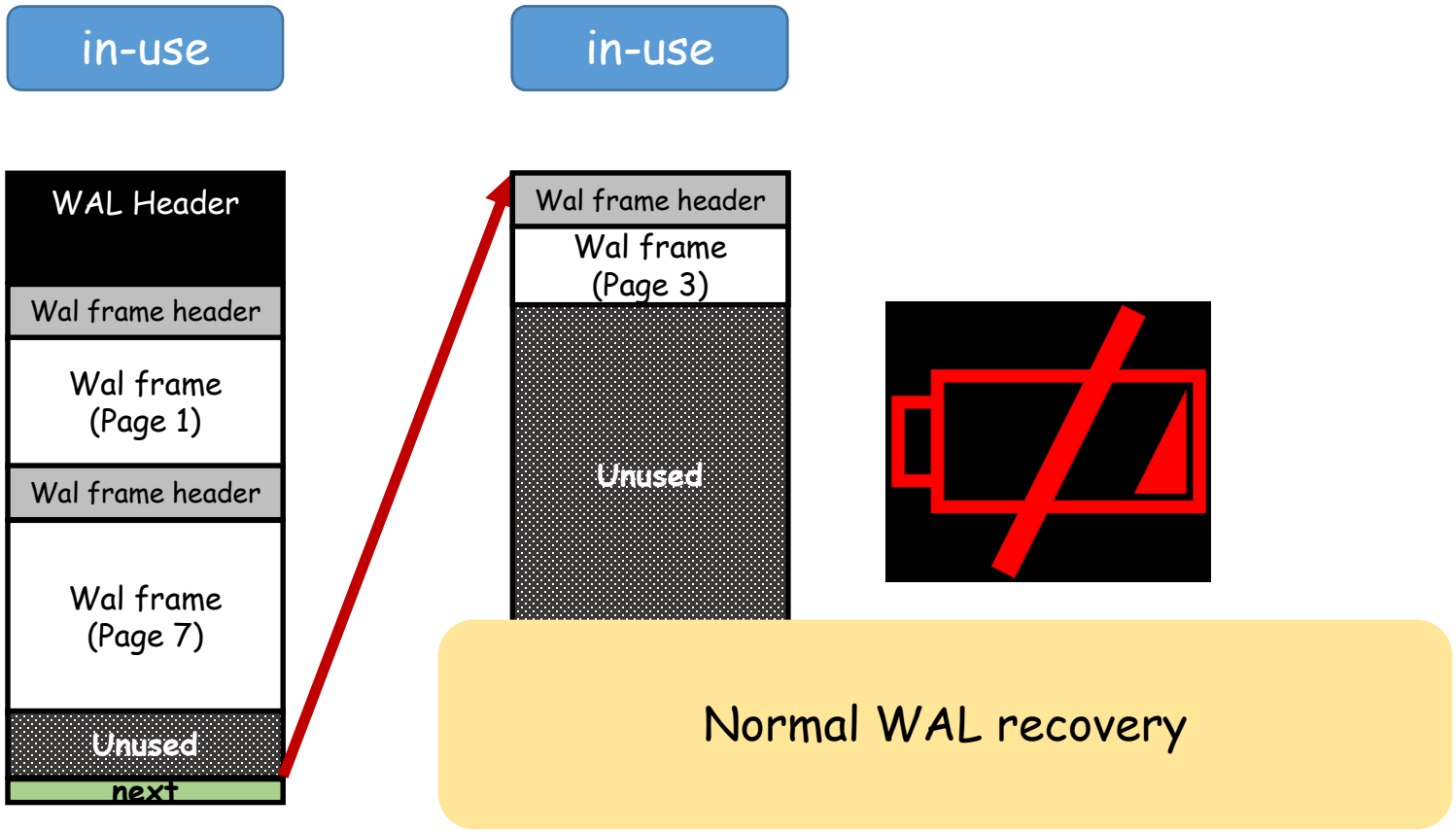


Recovery in NVWAL (3) – pending



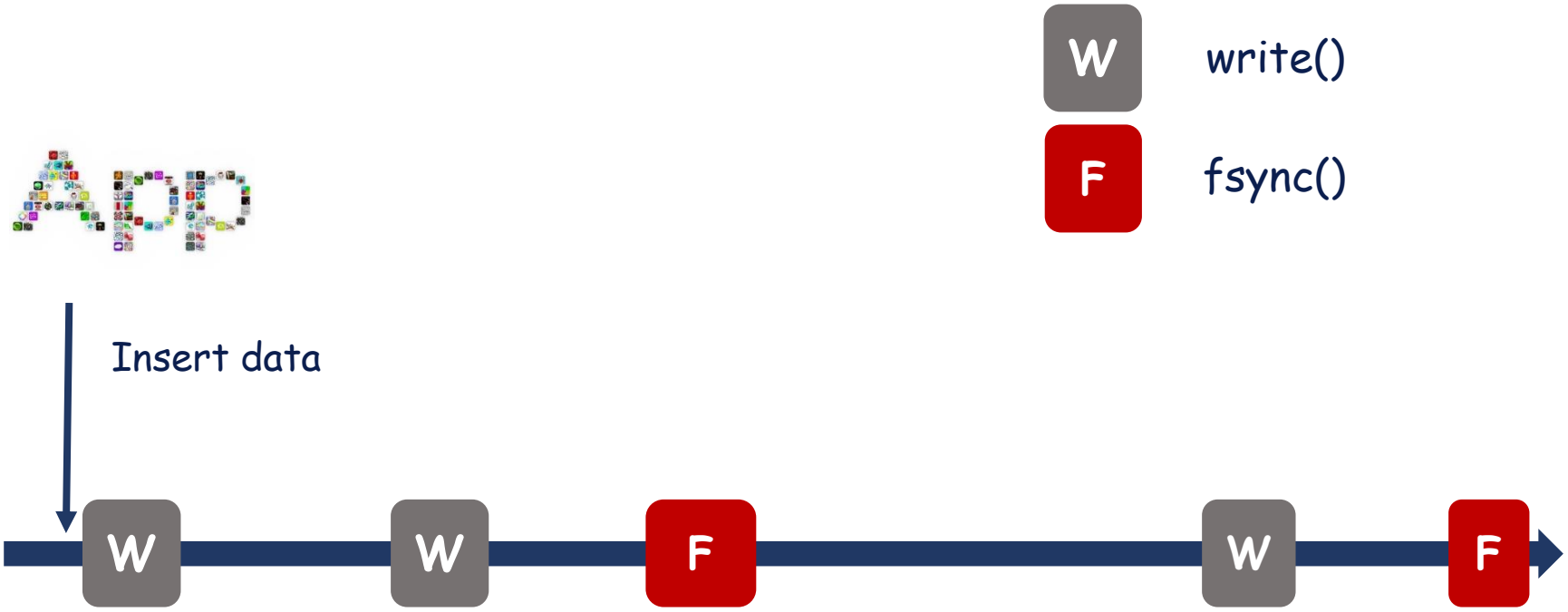
Recovery process deletes pointers to pending blocks

Recovery in NVWAL (4) in-use



Transaction-aware Lazy Synchronization

Persistency Guarantee in Flash



Eager Synchronization in NVRAM

Insert data

m memcpy

mb memory barrier

CL flush cache line flush

pb persist barrier

m mb CL flush mb m mb CL flush mb pb C mb CL flush mb pb

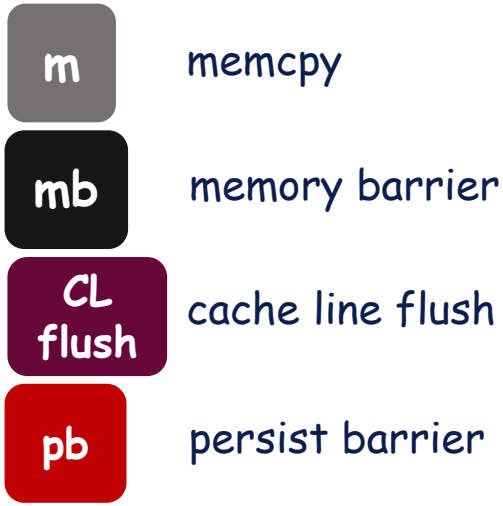
Enforce ordering

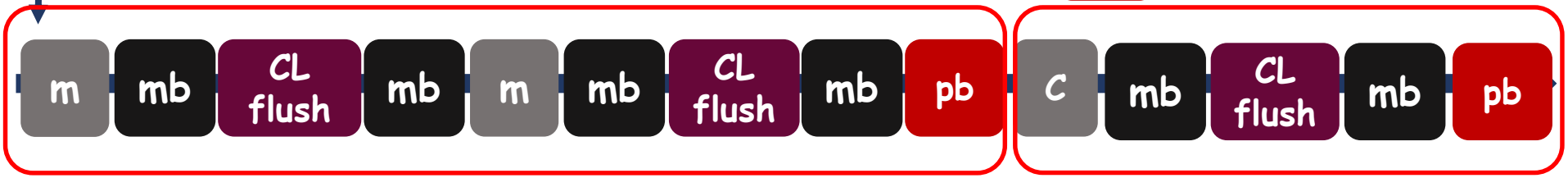
Enforce ordering

Enforce ordering

Transaction-Aware Persistency Guarantee in NVRAM

Insert data





Logging phases

Commit phases

Asynchronous Commit in NVRAM

m — memcpy

mb — memory barrier

CL flush — cache line flush

pb — persist barrier

Chk sum — checksum

Insert data

CISSR

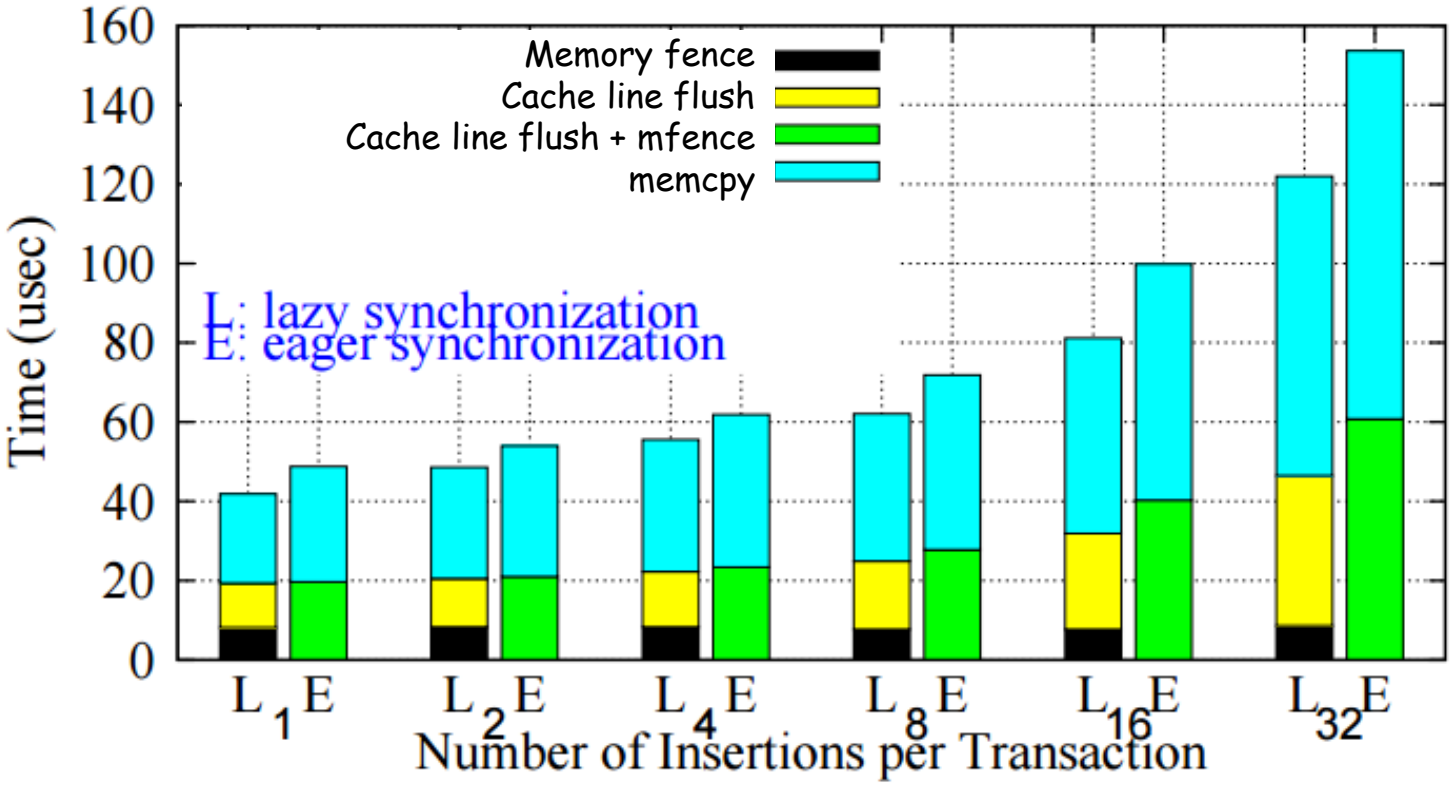
Center for Innovative System Software Research

Evaluation

- Implement NVWAL in SQLite 3.7.11
  - Used in Android 4.4
- Tuna
  - NVRAM emulation board with ARM Cortex-A9
  - DDR3-SDRAM DRAM
  - DDR3-SDRAM NVRAM (Xilinx Zynq SOC)

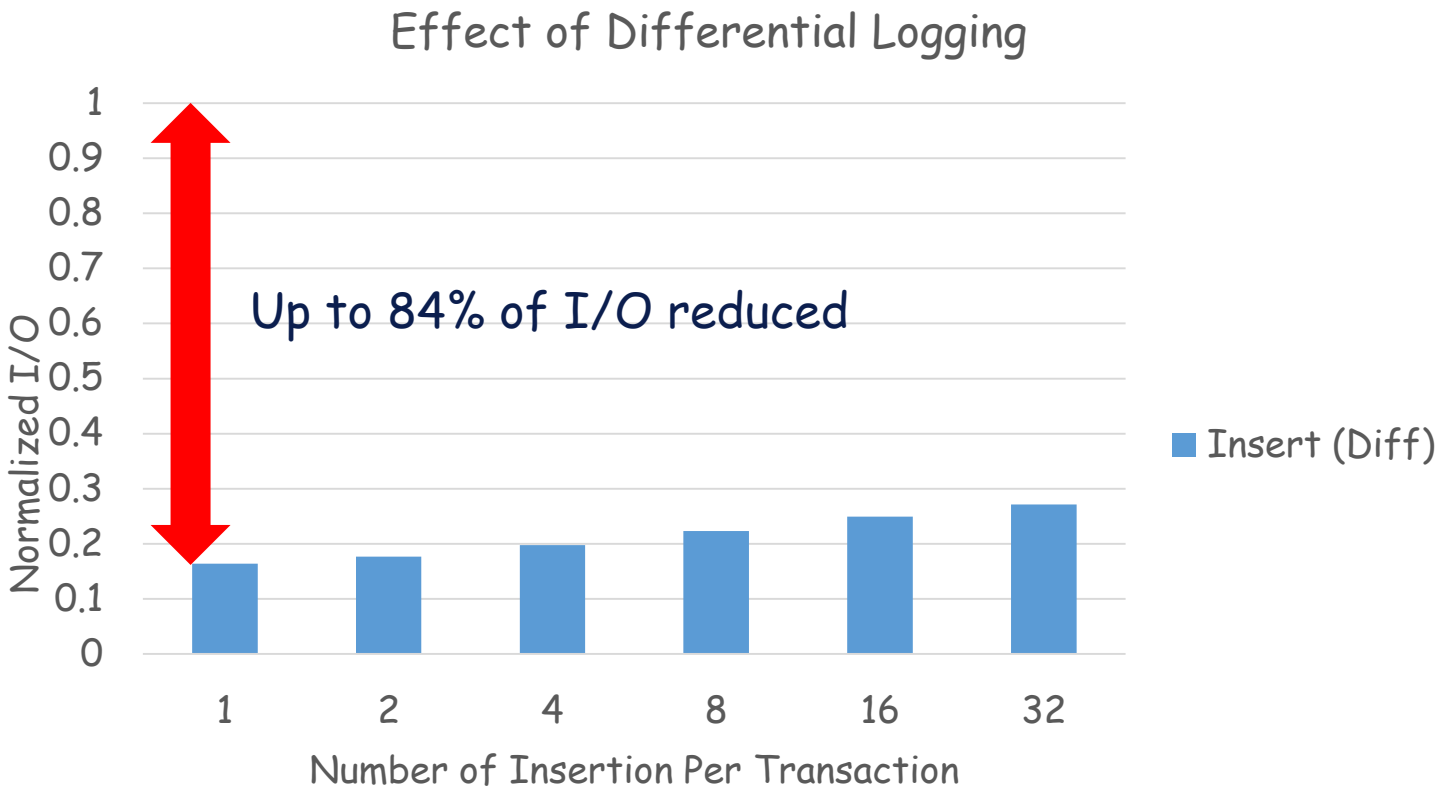
- Nexus5
  - 2.26 GHz Snapdragon 800 processor
  - DDR memory
    - we assume that specific address range of DRAM is NVRAM
    - NVRAM latency is emulated by nop operations.
- Performance Analysis Tools
  - Mobibench

Overhead of Ordering Constraints



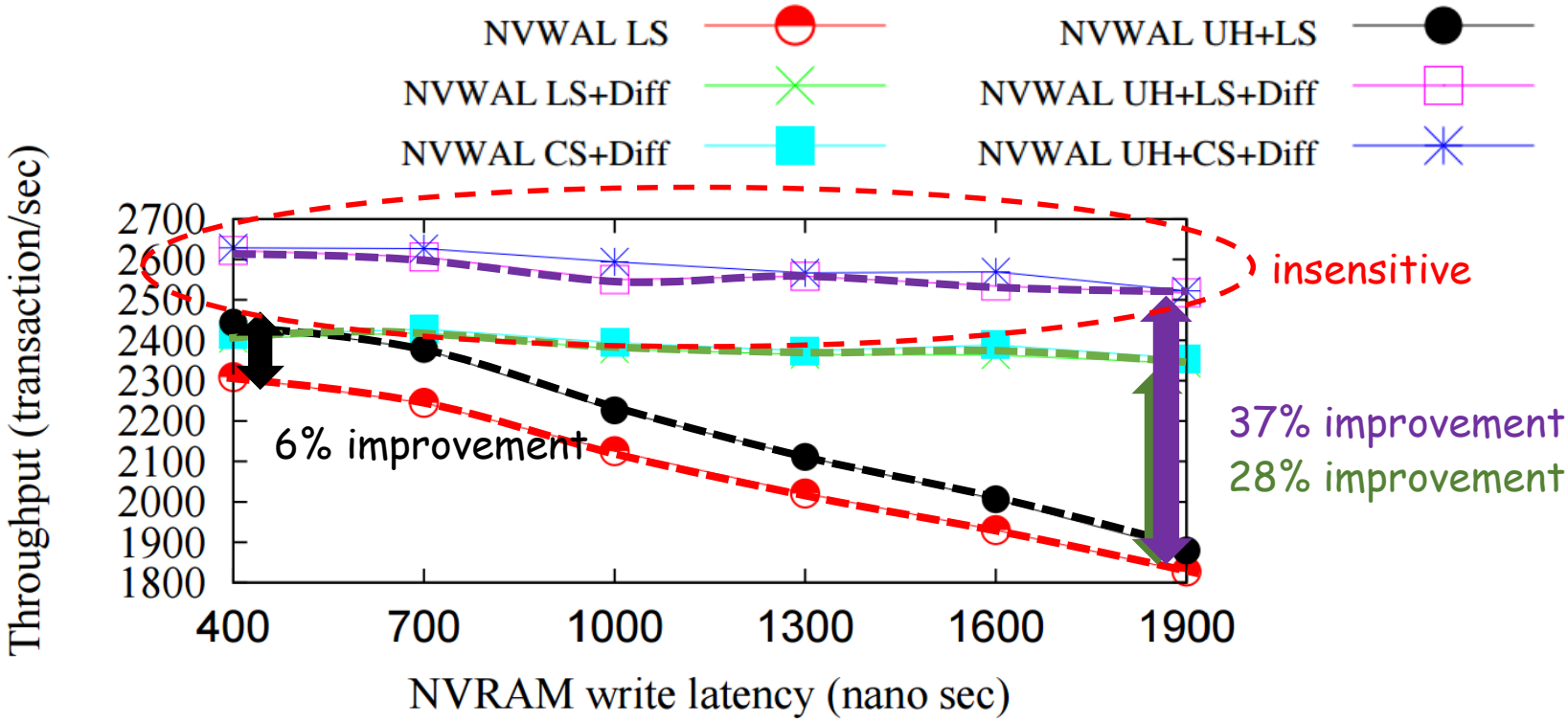
- Lazy synchronization eliminates up to **23%** of persistency overhead.

Differential Logging and I/O



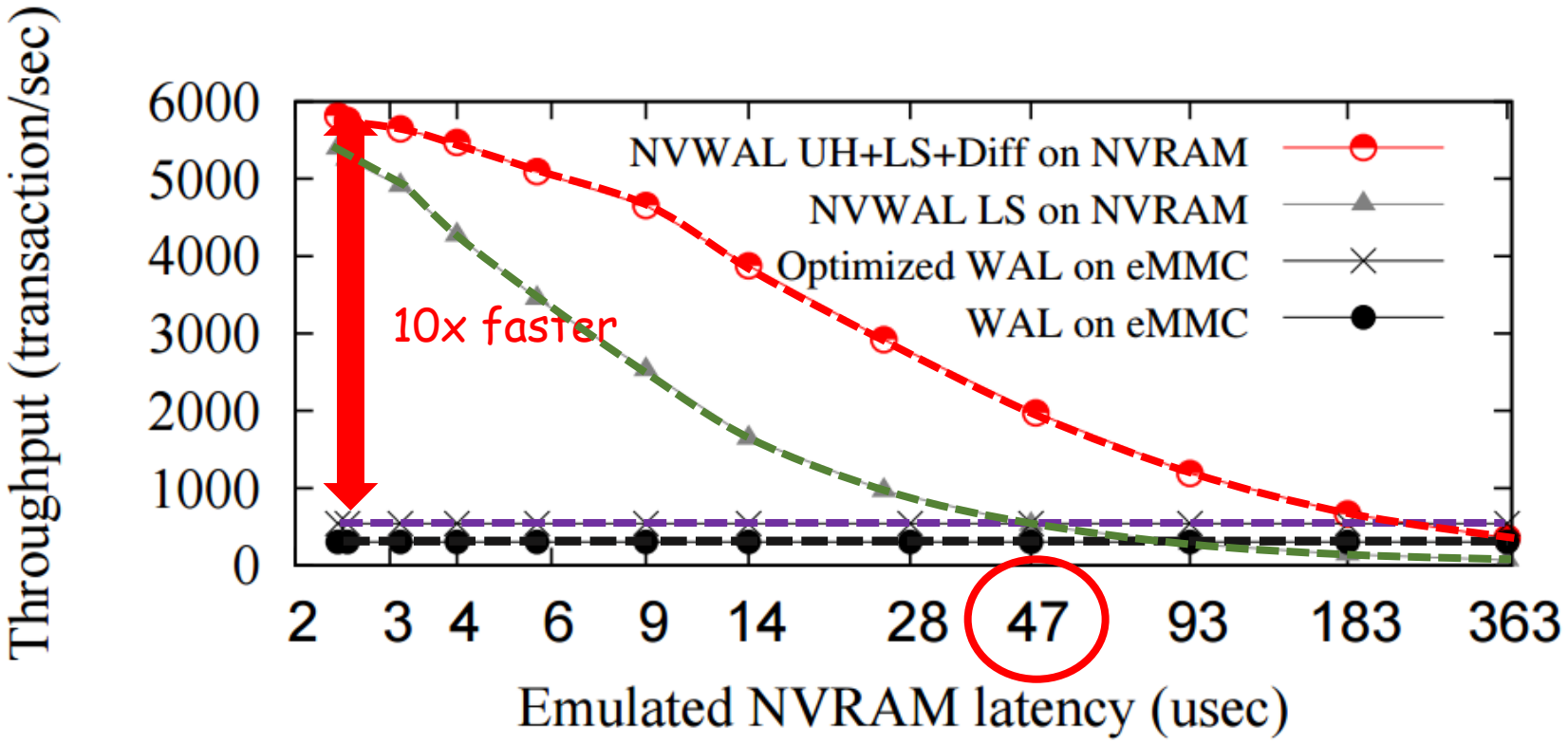
- Differential logging eliminates up to 84% of the unnecessary I/O.

Transaction Throughput and NVRAM Latency



- User-Level Heap improves 6% of performance.
- Differential logging yields up to 28% higher throughput.
- Combining all, we can get up to 37% higher performance.

Transaction Throughput of NVWAL on Nexus5



- Combining all of optimization performs at least 10 times faster when NVRAM latency is smaller than 3us.
- With our optimization, NVWAL shows performance similar to that of WAL on flash memory when the write latency is set to 230 usec.

Conclusion

- Strict ordering of memory writes causes unnecessary overhead.
  - Transaction-aware lazy synchronization
- Leveraging byte-addressability of NVRAM
  - Byte-granularity differential logging
  - User-level NVRAM heap manager
- Via the optimizations, we make application performance insensitive to the NVRAM write latency.
  - 400 nsec → 1900 nsec NVRAM latency results in only 4% performance degradation

Thank You