Pangolin: A Fault-tolerant Persistent Memory Programming Library

Lu Zhang and Steven Swanson

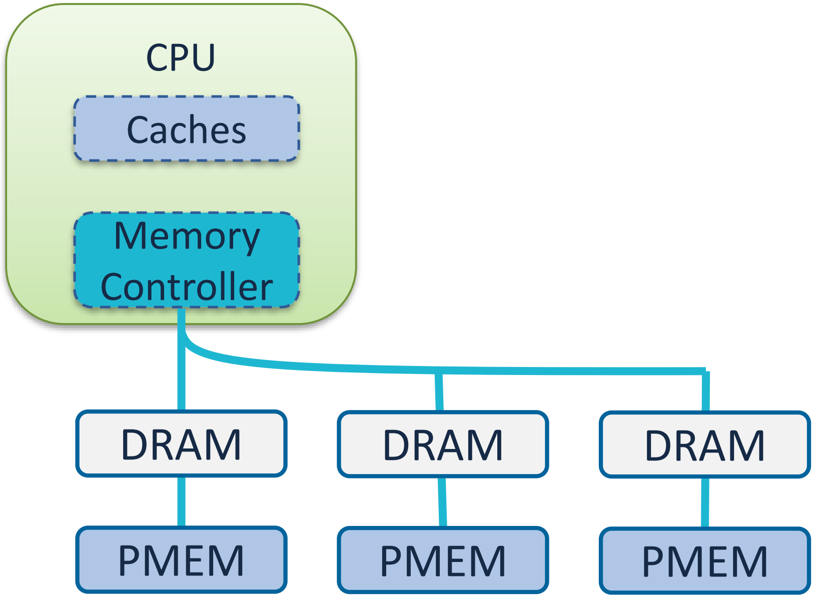
Non-Volatile Systems Laboratory
Department of Computer Science & Engineering
University of California, San Diego

NVSL

UCSDCSE
Computer Science and Engineering

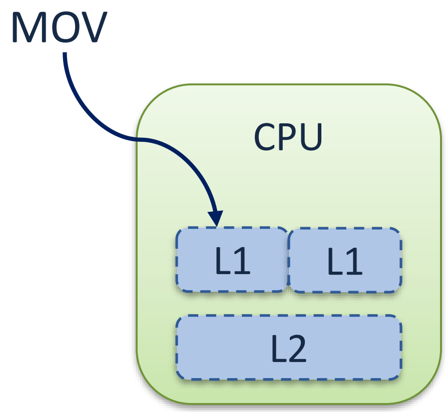
Persistent memory (PMEM) finally arrives

- Working alongside DRAM
- New programming model
  - Byte addressability
  - Memory semantics
  - Direct access (DAX)



Challenges with PMEM programming

- Crash consistency
  - Volatile CPU caches
  - 8-byte store atomicity

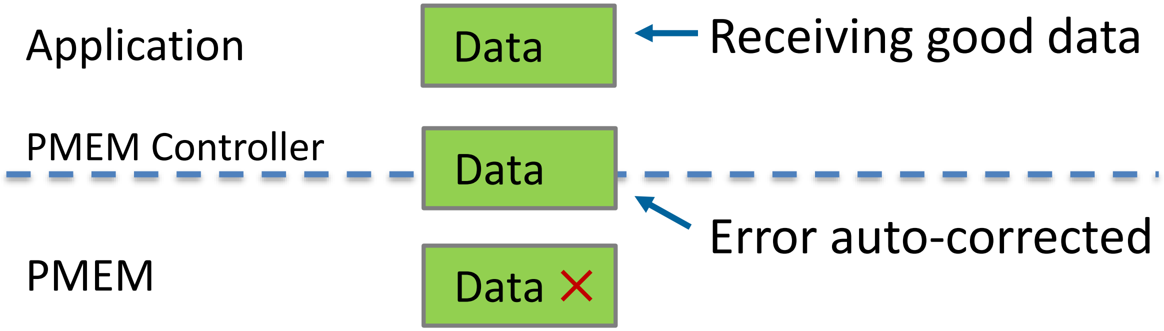


- Fault tolerance
  - Media errors
  - Software bugs

NVSL

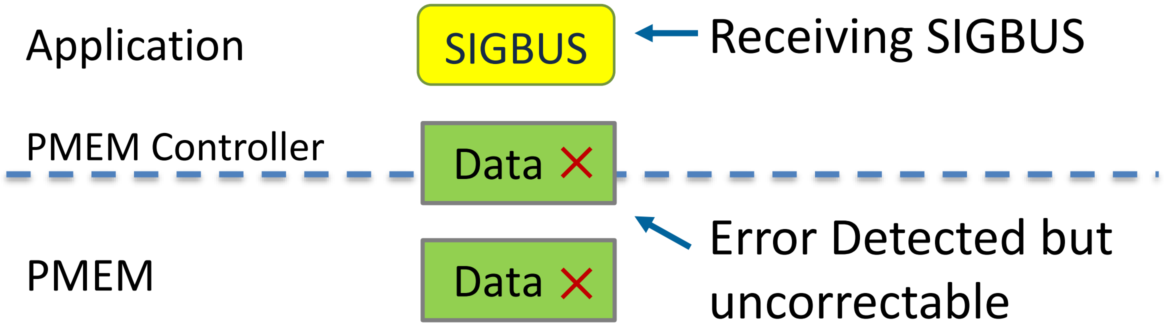
Persistent memory error types

- Persistent memory and its controller implement ECC
  - ECC-detectable & correctable errors do not need software intervention



Persistent memory error types

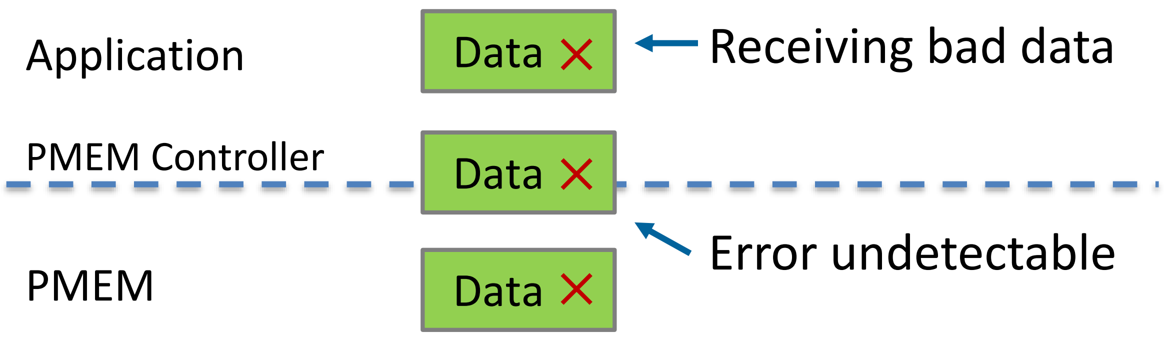
- Persistent memory and its controller implement ECC
  - ECC-detectable & correctable errors do not need software intervention
  - ECC-detectable but uncorrectable ones require signal handling



NVSL

Persistent memory error types

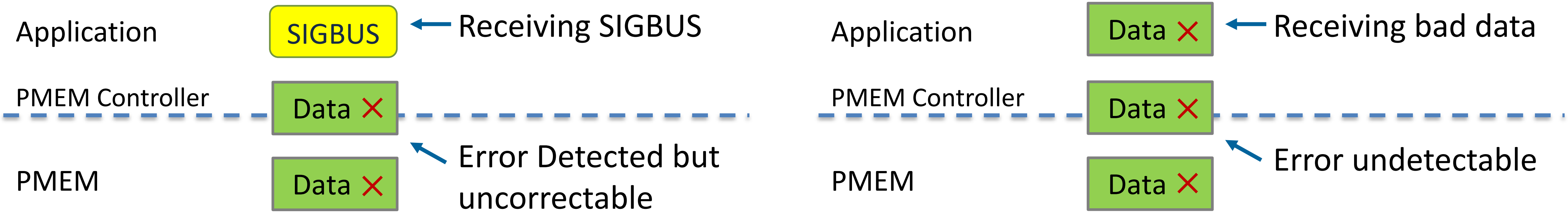
- Persistent memory and its controller implement ECC
  - ECC-detectable & correctable errors do not need software intervention
  - ECC-detectable but uncorrectable ones require signal handling
  - ECC-undetectable errors demand software detection and correction



NVSL

Handle uncorrectable & undetectable errors

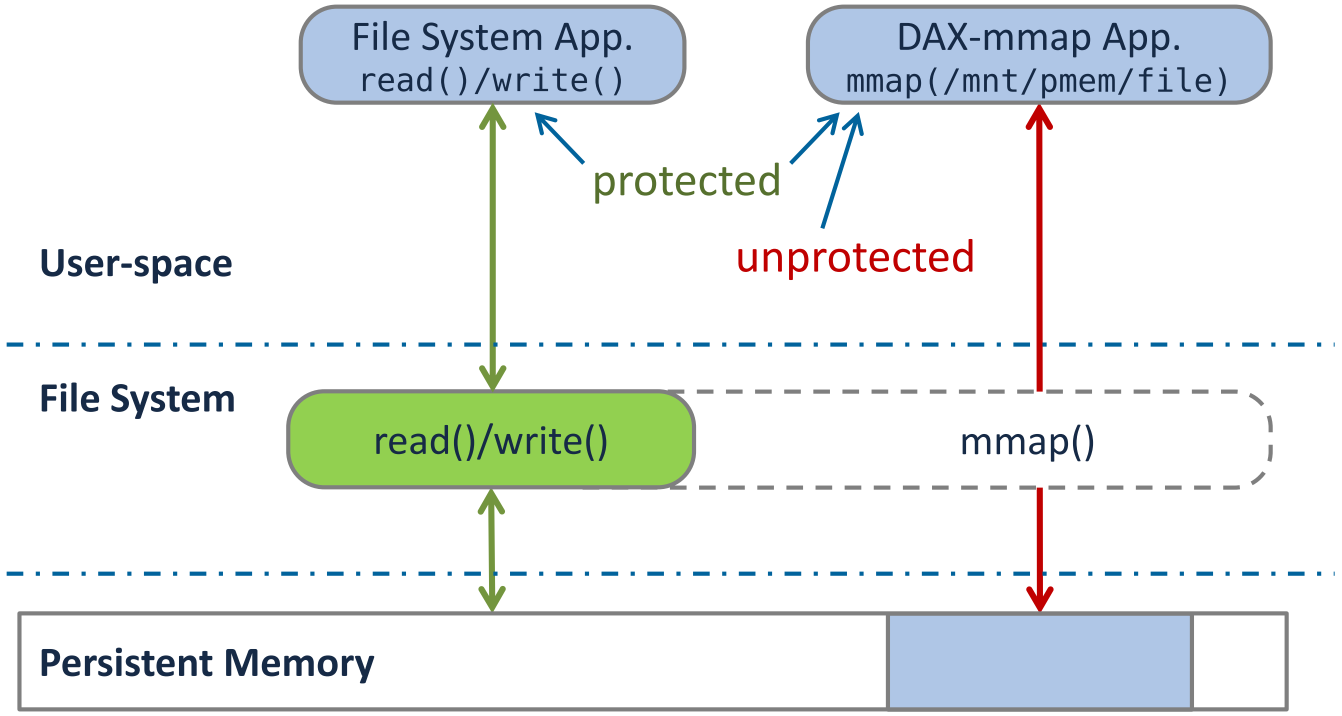
- Prepare some redundancy for recovery
- Implement software-based error detection and correction



NVSL

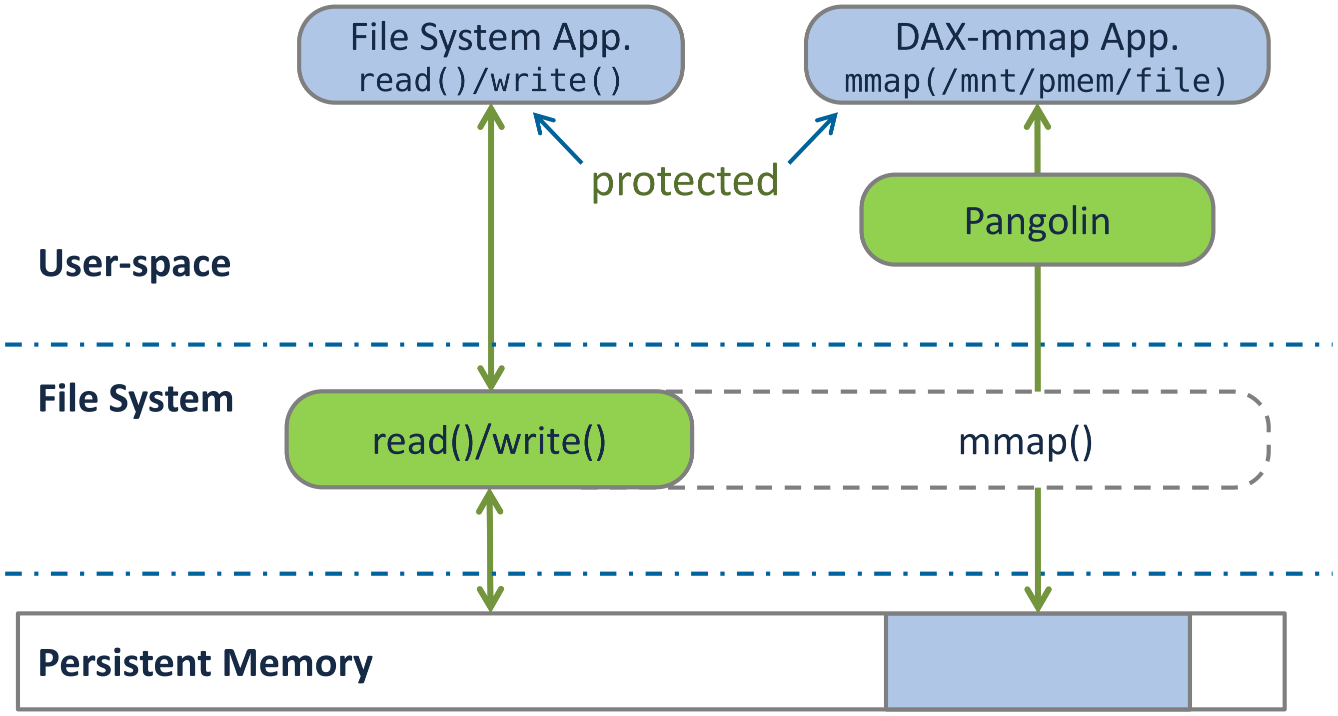
DAX-filesystem cannot protect mmap'ed data

- Some filesystems (e.g. NOVA) provide protection only via read()/write()
- No known filesystem can protect DAX-mmap'ed PMEM data



DAX-filesystem cannot protect mmap'ed data

- Some filesystems (e.g. NOVA) provide protection only via read()/write()
- No known filesystem can protect DAX-mmap'ed PMEM data



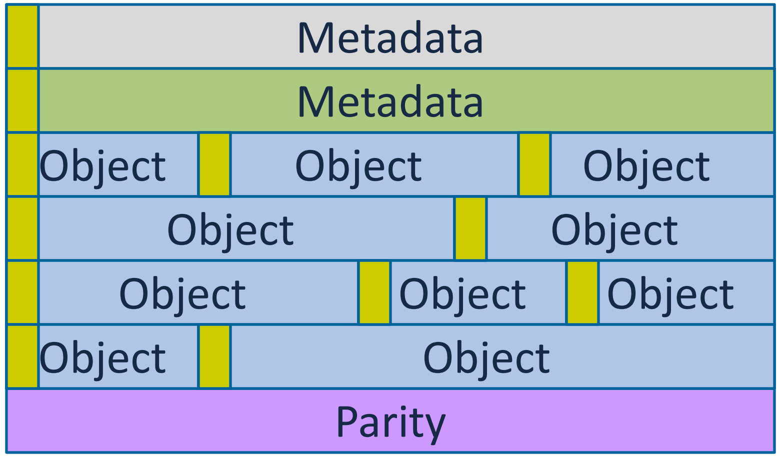
Pangolin design goals

- Ensure crash consistency
- Protect application data against media and software errors
- Require very low storage overhead (1%) for fault tolerance

NVSL

Pangolin – Replication, parity, and checksums

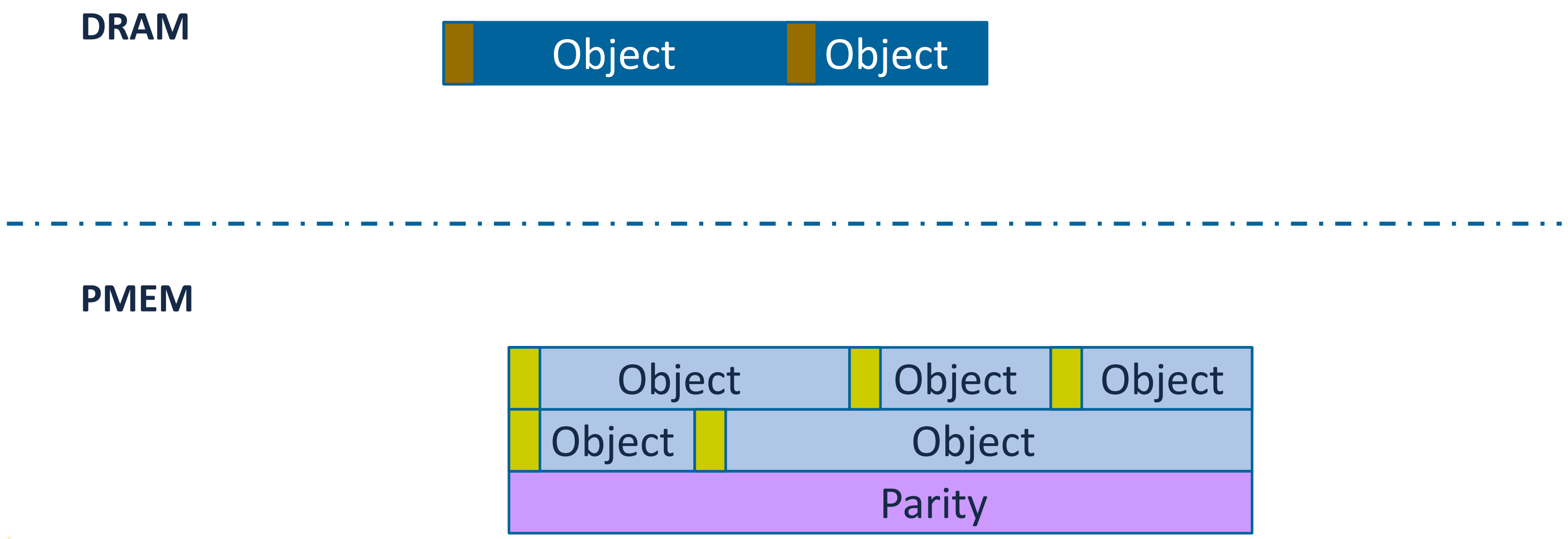
- Combines replication and parity as redundancy
  - Similar performance compared to replication
  - Low space overhead (**1%** of gigabyte-sized object store)



- Checksums all metadata and object data

Pangolin – Transactions with micro-buffering

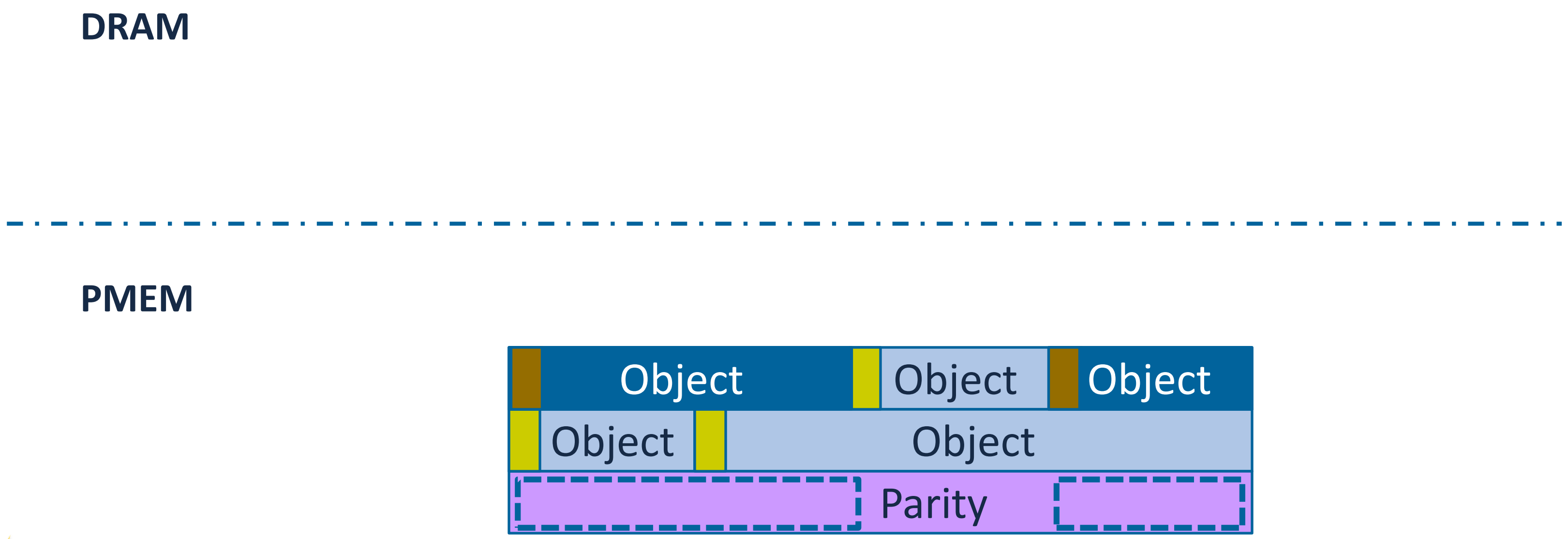
- Provides micro-buffering-based transactions
  - Buffers application changes in DRAM
  - Atomically updates objects, checksums, and parity



NVSL

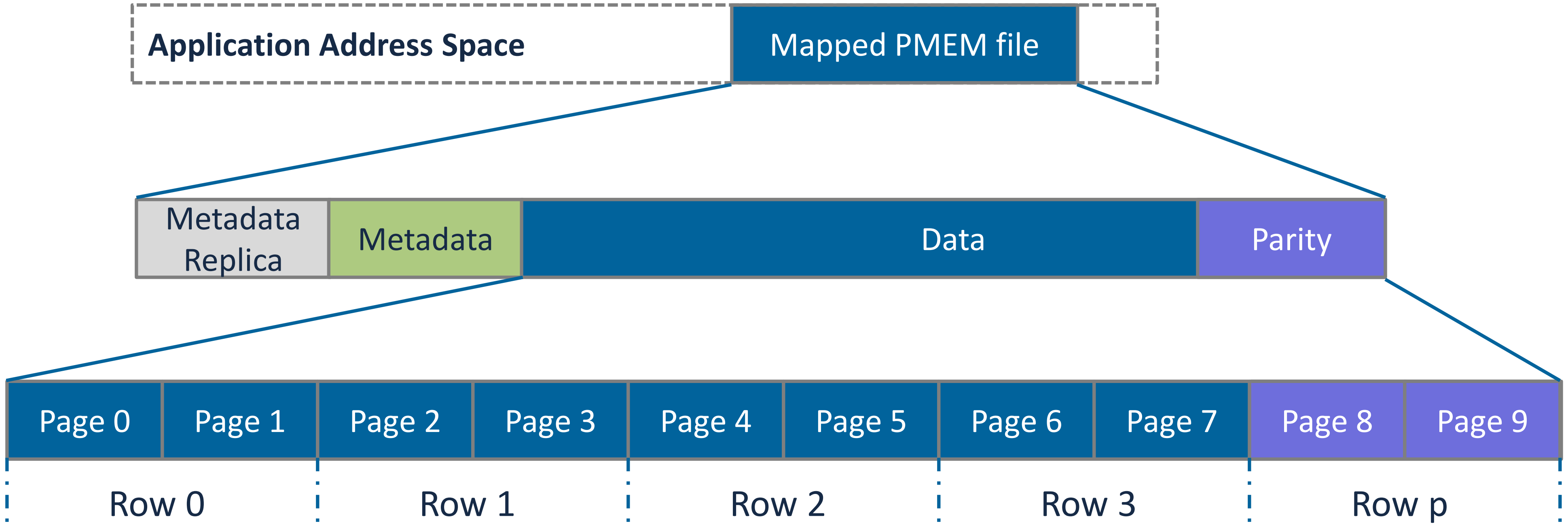
Pangolin – Transactions with micro-buffering

- Provides micro-buffering-based transactions
  - Buffers application changes in DRAM
  - Atomically updates objects, checksums, and parity



Pangolin's data redundancy

- Reserve space for metadata replication and object parity
- Organize object data pages into "rows"

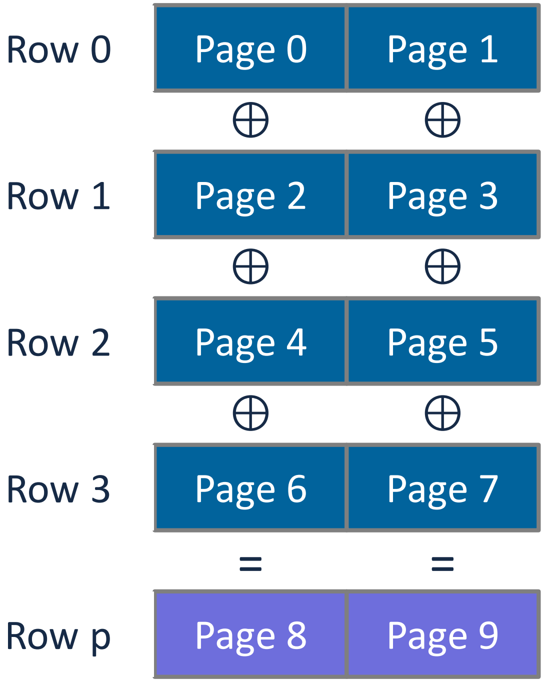


Row size: default 160 MB (1% of a data "zone")

NVSL

Pangolin's parity coding

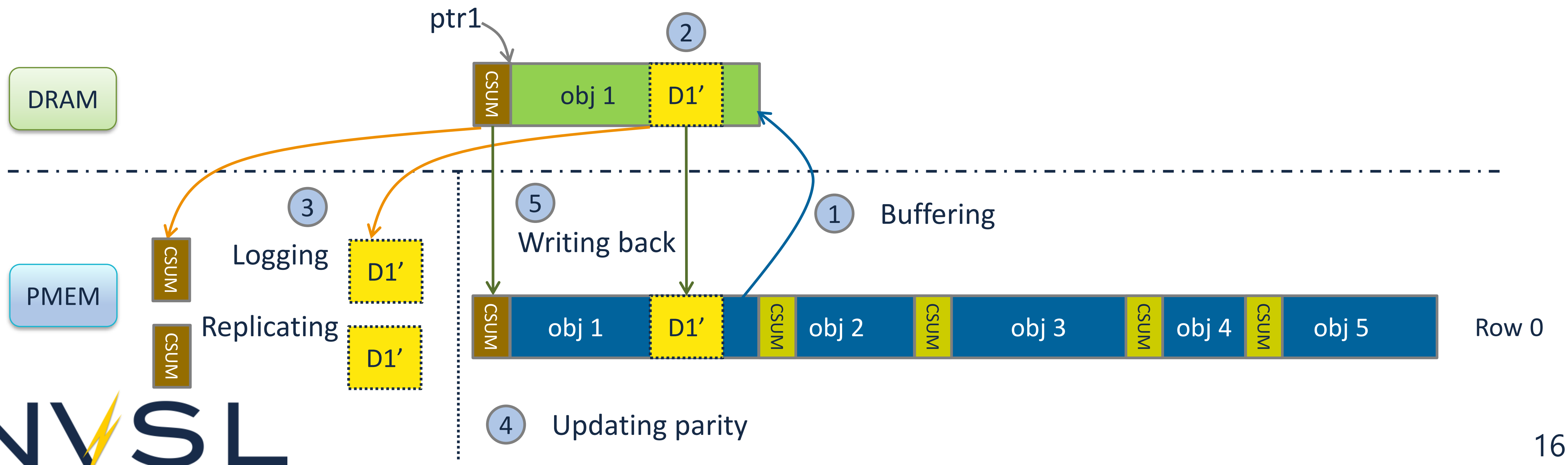
- Compute a parity page vertically across all rows
- Afford losing one whole row of data
- By default, Pangolin implements 100 rows per data zone



NVSL

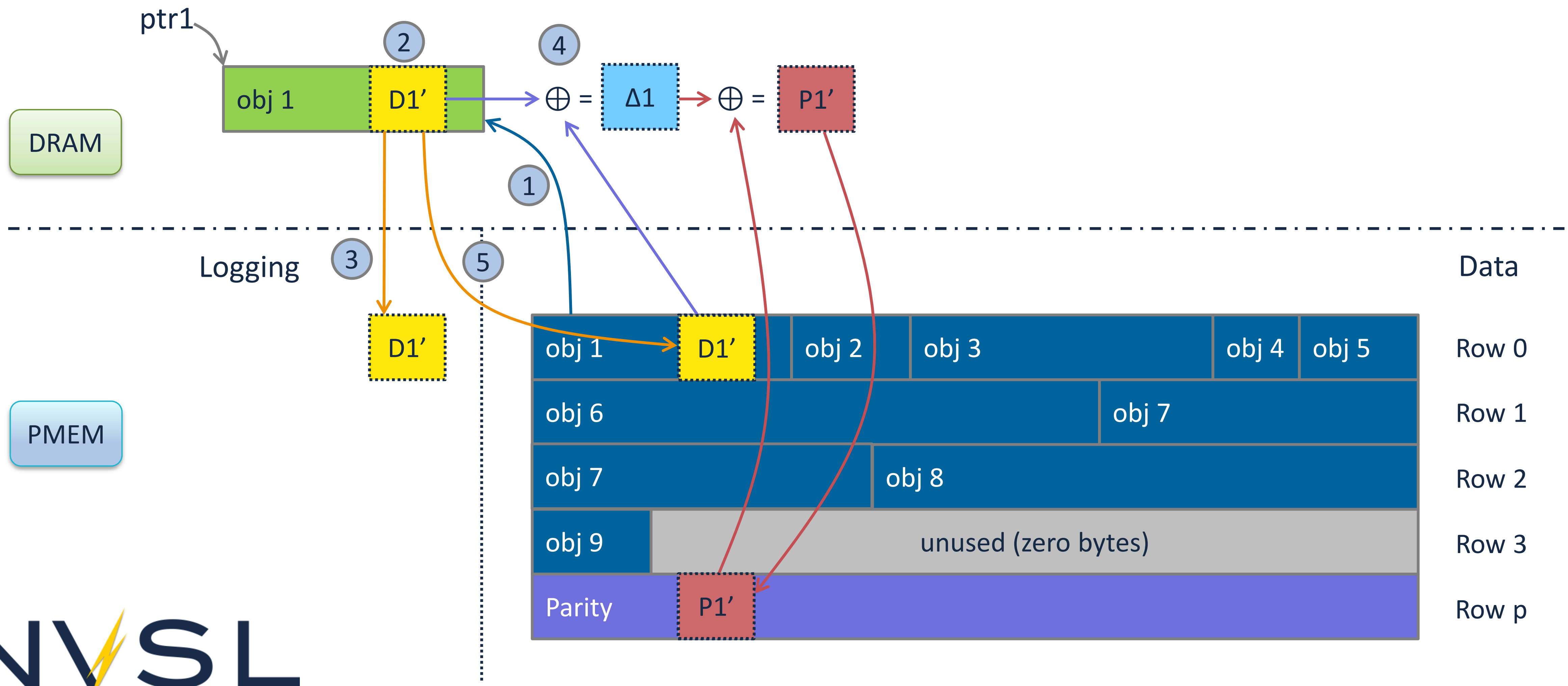
Micro-buffering provides transactions

- Move object data in DRAM and perform data integrity check
- Buffer writes to objects and write back to PMEM on commit
- Guarantee consistency with redo logging (replicated)



NVSL

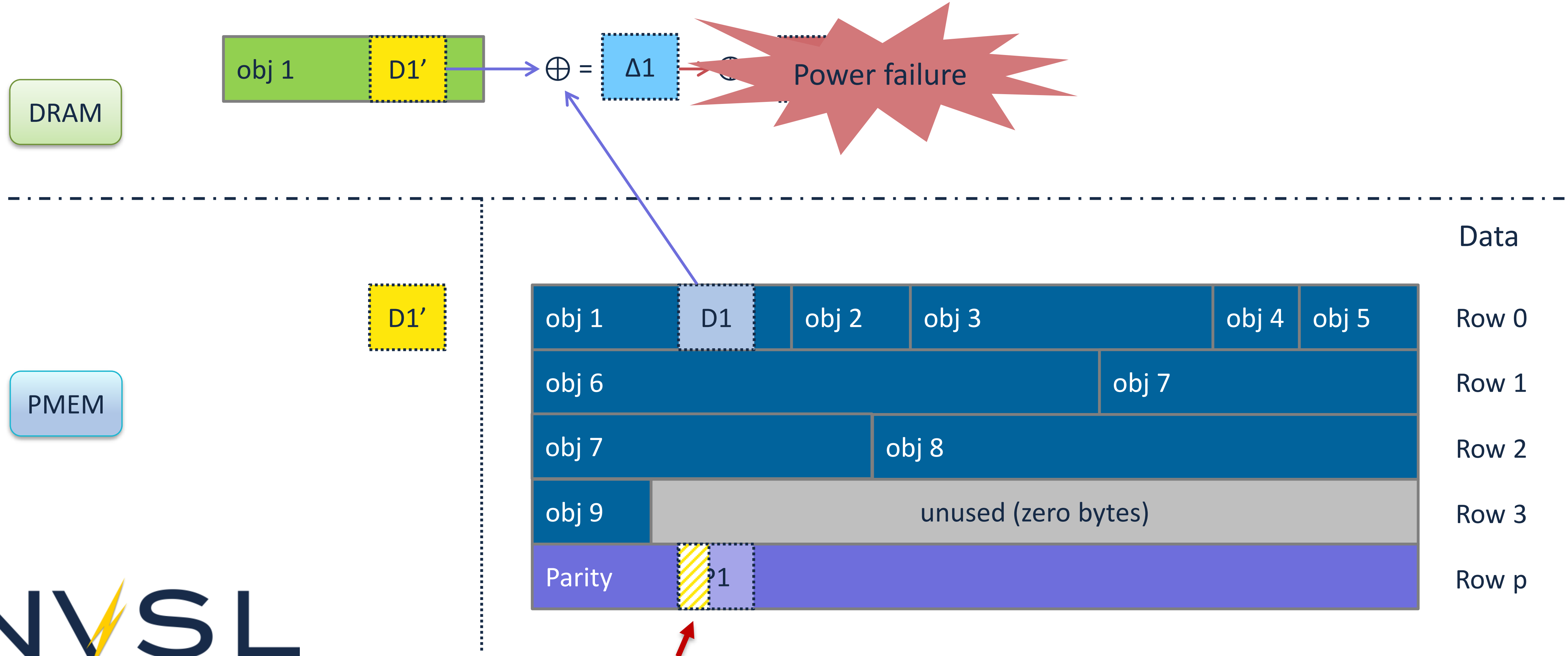
Updating parity using only modified ranges



NVSL

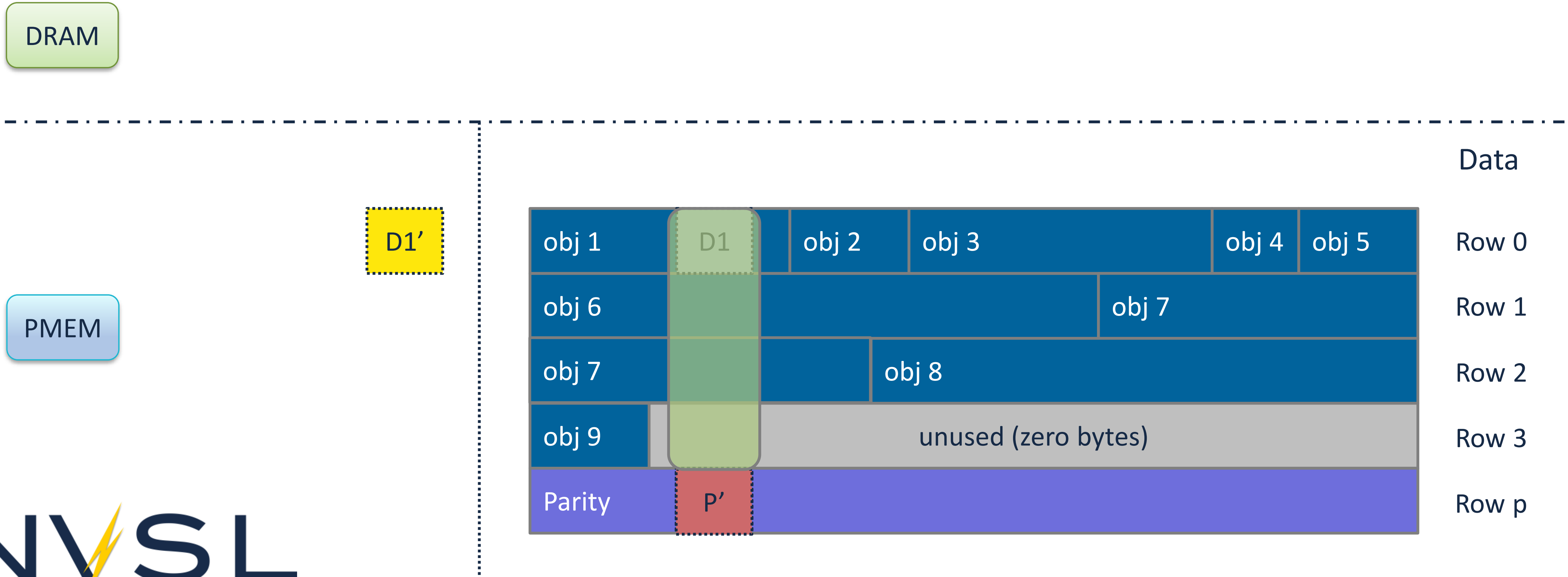
Parity's crash consistency depends on object logs

- Apply all redo-logs (if exist) and then re-compute parity



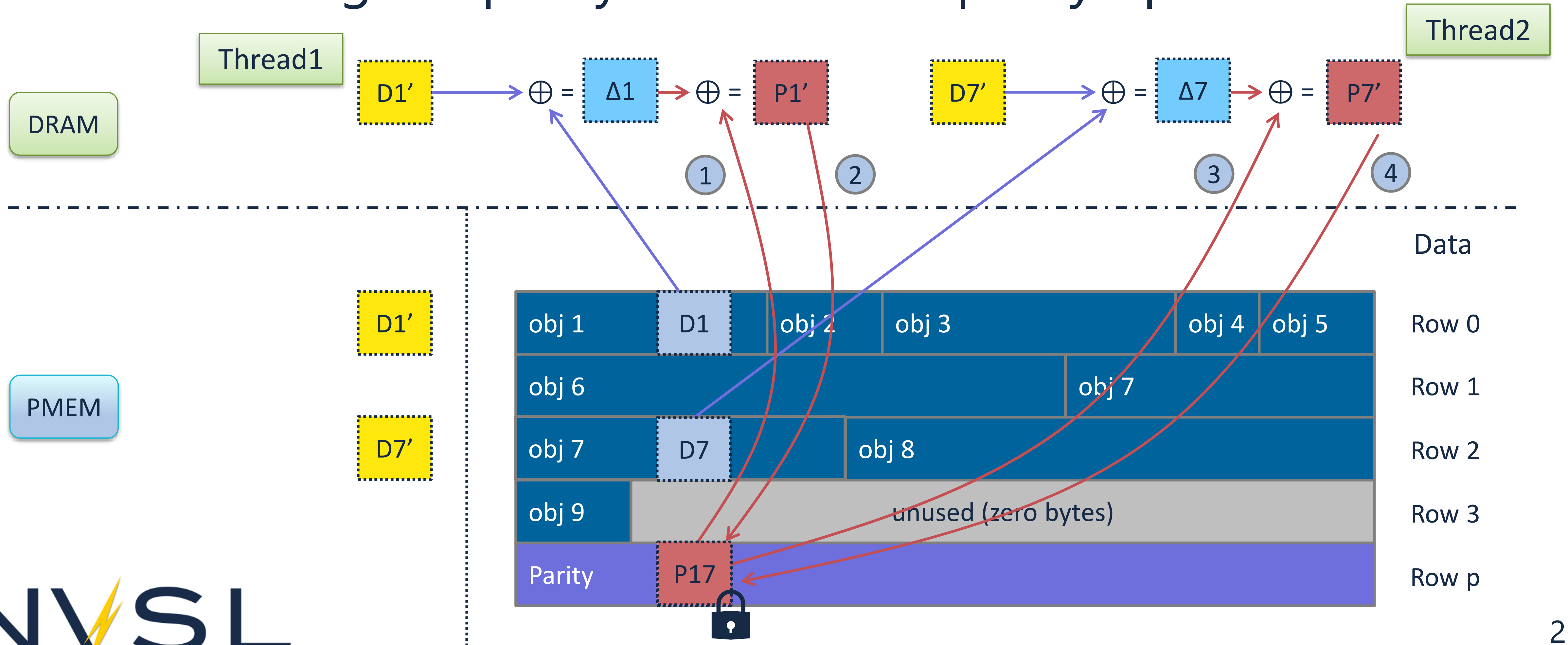
Parity's crash consistency depends on object logs

- Apply all redo-logs (if exist) and then re-compute parity



Multithreaded update – Lock parity ranges

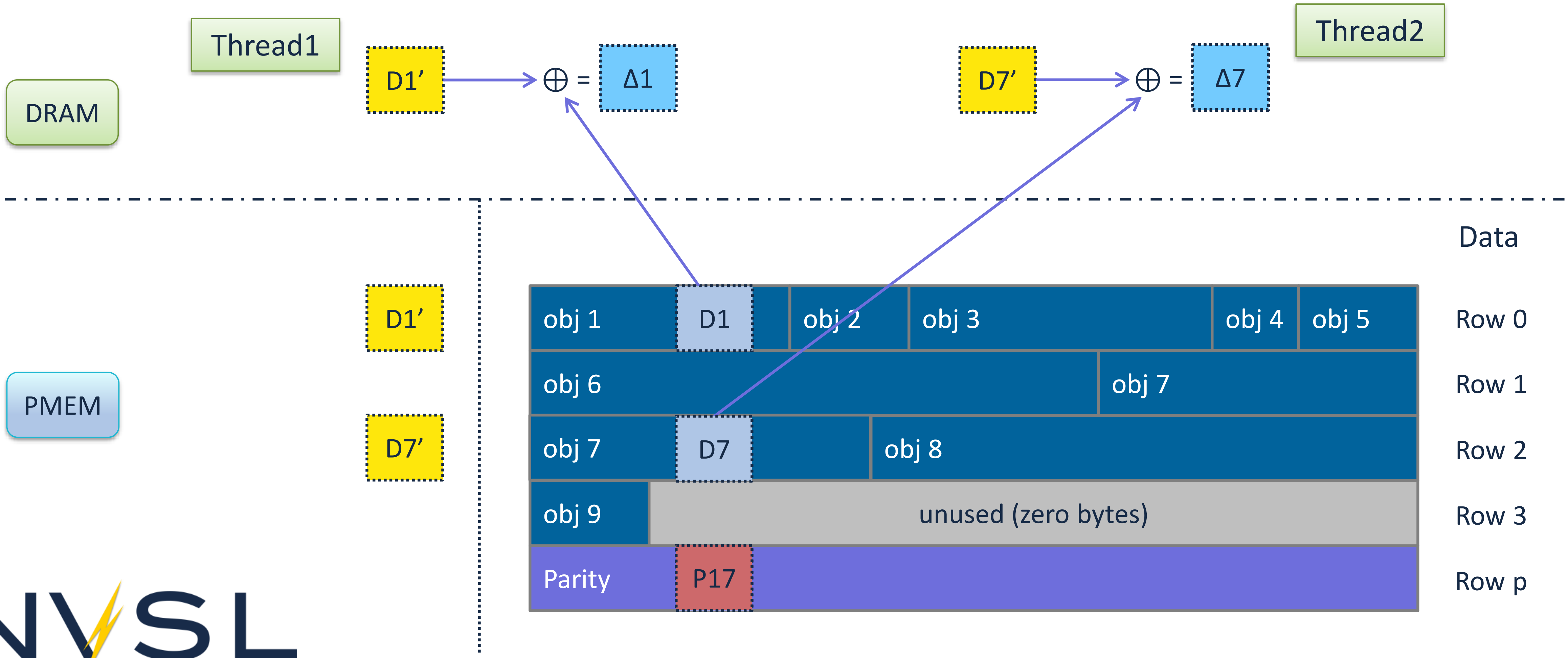
- Lock a range of parity and serialize parity updates



NVSL

Multithreaded update – Atomic XORs

- Parity range can update, lock-free, with atomic XORs



NVSL

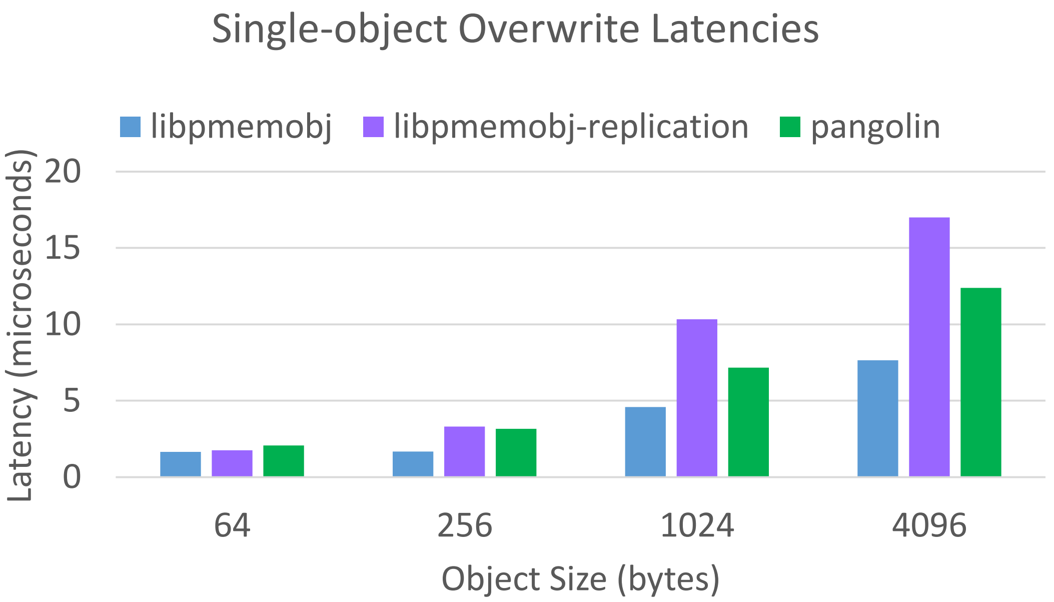
Multithreaded update – Hybrid scheme

- Atomic XORs can be slower than vectorized ones
- Use shared mutex to coordinate both methods
- Small updates (< 8KB)
  - Take shared lock of a parity range (8 KB)
  - Update parity concurrently with atomic XORs
- Large updates (≥ 8KB)
  - Take exclusive locks of parity ranges (8 KB each)
  - Update parity using vectorized XORs (non-atomic)

NVSL

Performance – Single-object transactions

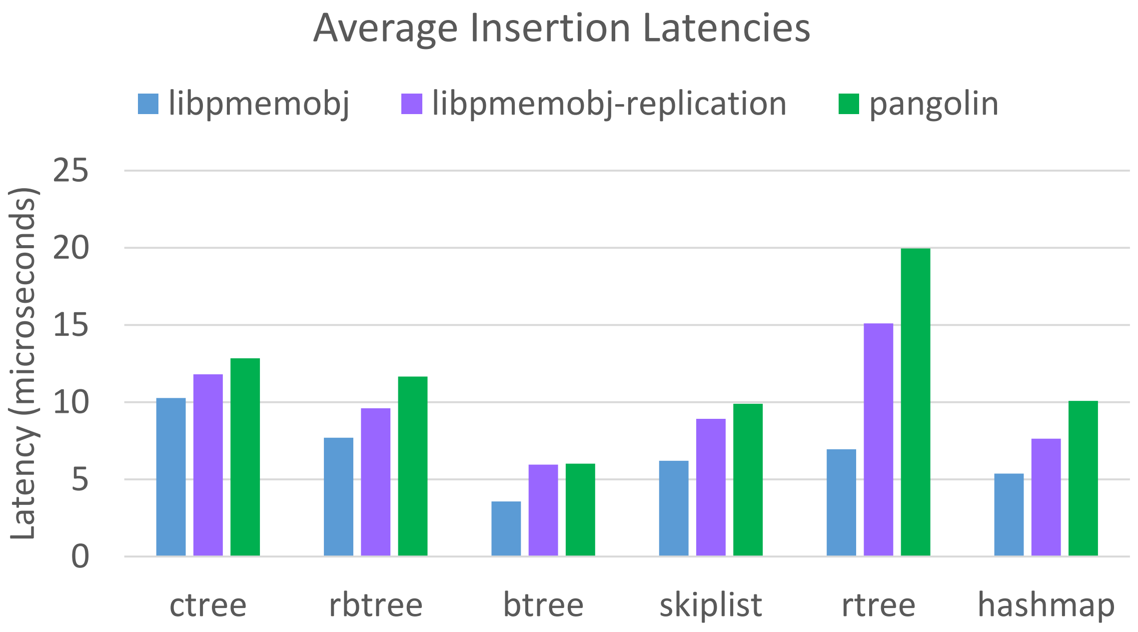
- Evaluation based on Intel's Optane DC persistent memory
- On average, Pangolin's latency is 11% lower than libpmemobj with replication.

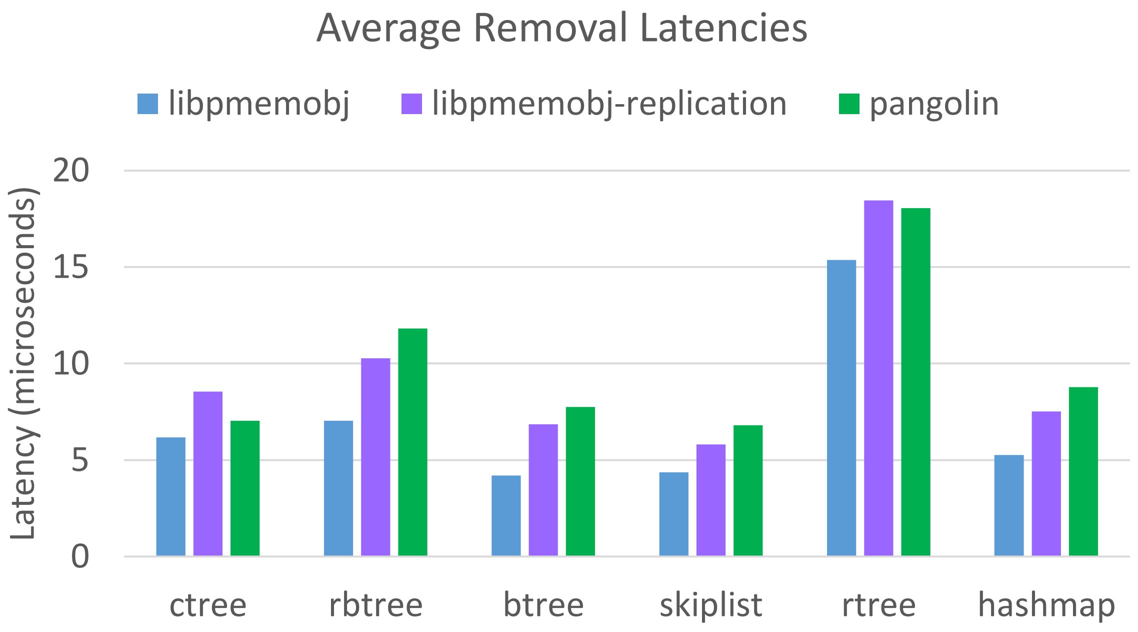


NVSL

Performance – Multi-object transactions

- Performance of Pangolin is 90% of libpmemobj's with replication
- Pangolin incurs about 100× less space overhead





Conclusion

- PMEM programming libraries should also consider fault tolerance for critical applications.
- Parity-based redundancy provides similar performance compared to replication and significantly reduces space overhead.
- Micro-buffering-based transactions can both support crash consistency and provide fault tolerance.

NVSL