SecPM: a Secure and Persistent Memory System for Non-volatile Memory

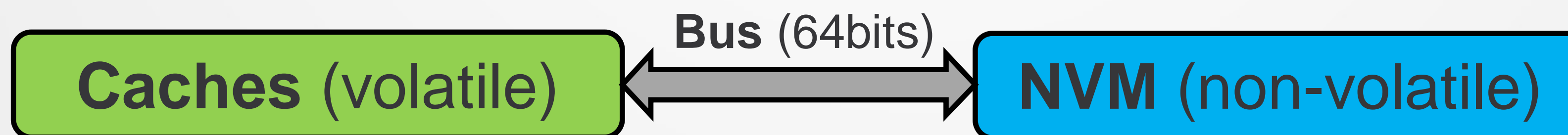
Pengfei Zuo, **Yu Hua**

Huazhong University of Science and Technology, China

10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'18)

Persistence Issue

- The non-volatility of NVM enables data to be persistently stored into NVM
- Data may be incorrectly persisted due to crash inconsistency
  - Modern processors and caches usually reorder memory writes
  - Volatile caches cause partial update



Consistency Guarantee for Persistence

- Durable transaction: a commonly used solution
  - NV-Heaps (ASPLOS'11), Mnemosyne (ASPLOS'11), DCT (ASPLOS'16), DudeTM (ASPLOS'17), NVML (Intel)
  - Enable a group of memory updates to be performed in an atomic manner
- Enforce write ordering
  - Cache line flush and memory barrier instructions
- Avoid partial update
  - Logging

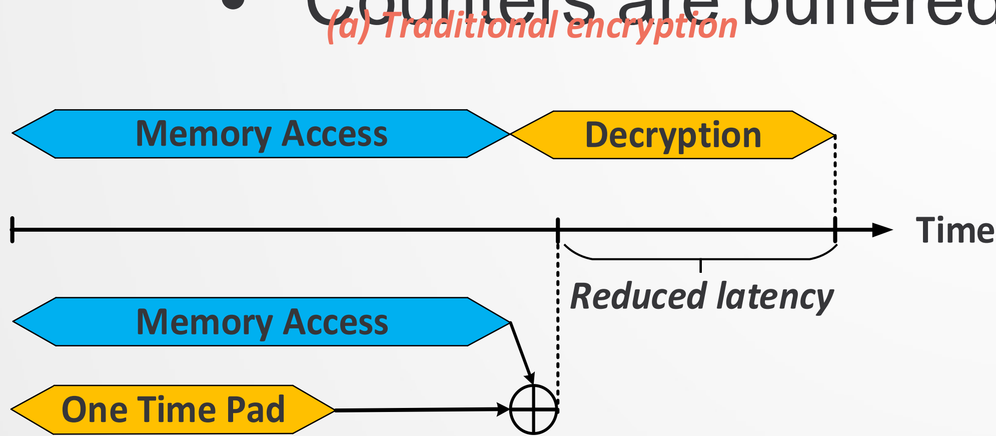
```
TX_BEGIN
    do some computation;
    // Prepare stage: backing up the data in log
    write undo log;
    flush log;
    memory_barrier();
    // Mutate stage: updating the data in place
    write data;
    flush data;
    memory_barrier();
    // Commit stage: invalidating the log
    log->valid = false;
    flush log->valid;
    memory_barrier();
TX_END
```

Security Issue

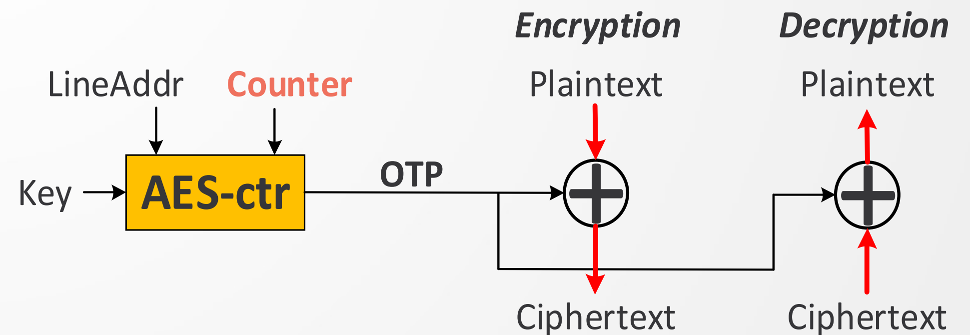
- Traditional DRAM: volatile
  - If a DRAM DIMM is removed from a computer
    - Data are quickly lost
- NVM: non-volatile
  - If an NVM DIMM is removed
    - An attacker can directly stream out the data from the DIMM
    - Unsecure

Memory Encryption for Security

- Counter mode encryption
  - Hide the decryption latency
  - Generate One Time Pad (OTP) using a per-line counter
    - Counters are buffered in an on-chip counter cache



(b) Counter mode encryption



The Gap between Persistence and Security

- Ensuring both security and persistence
  - Simply combining existing persistence schemes with memory encryption is inefficient
  - Each write in the secure NVM has to persist two data
    - Including the data itself and the counter
- Crash inconsistency
  - Cache line flush instruction cannot operate the counter cache
  - Memory barrier instruction fails to ensure the ordering of counter writes
- Performance degradation
  - Double write requests

Durable Transaction in Secure NVM

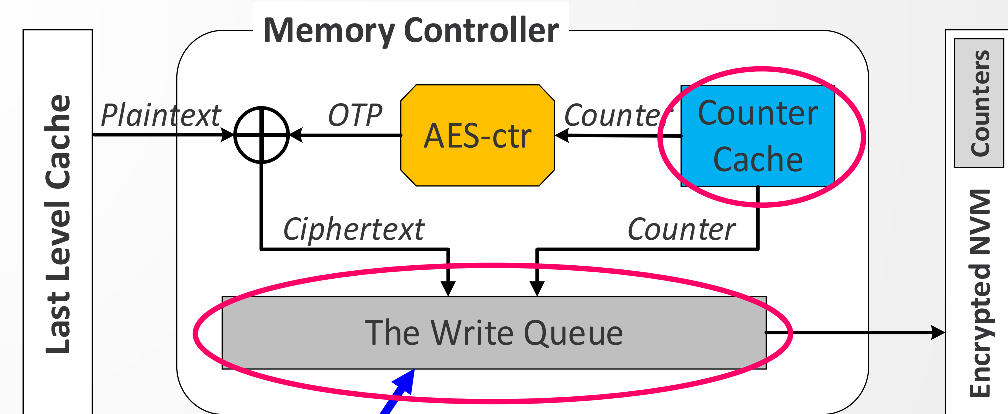
Stage	Log content	Log counter	Data content	Data counter	Recoverable?
Prepare	Wrong	Wrong	Correct	Correct	Yes
Mutate	Correct	Unknown	Wrong	Wrong	No
Commit	Correct	Unknown	Correct	Unknown	No

- Selective counter-atomicity (HPCA'18): modifications in software & hardware layers
  - Programming language
    - Add CounterAtomic variable and counter_cache_writeback() function
  - Compiler
    - Support the new primitives
  - Memory controller
    - Add a counter write queue

```
TX_BEGIN
    do some computation;
    // Prepare stage: backing up the data in log
    write undo log;
    flush log;
    memory_barrier();
    // Mutate stage: updating the data in place
    write data;
    flush data;
    memory_barrier();
    // Commit stage: invalidating the log
    log->valid = false;
    flush log->valid;
    memory_barrier();
TX_END
```

SecPM: a Secure and Persistent Memory System

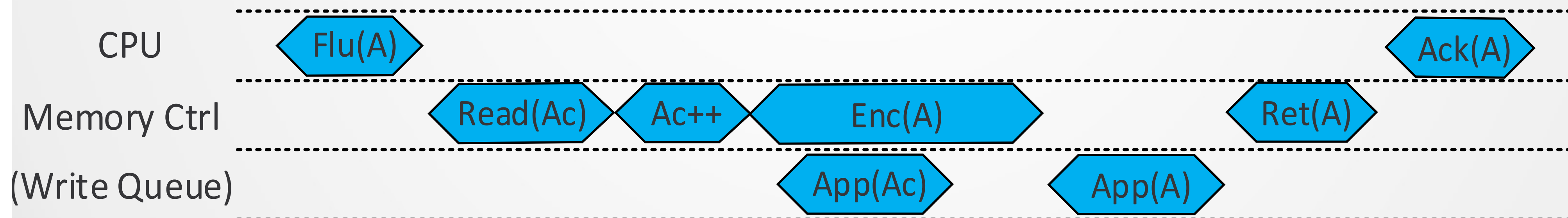
- Perform only slight modifications on the memory controller, being transparent for programmers
  - Programs running on an un-encrypted NVM can be directly executed on a secure NVM with SecPM
- Consistency guarantee
  - A counter cache write-through (CWT) scheme
- Performance improvement
  - A locality-aware counter write reduction (CWR) scheme



Asynchronous DRAM refresh (ADR): cache lines reaching the write queue can be considered durable.

Counter Cache Write-through (CWT) Scheme

- CWT ensures the crash consistency of both data and counter
  - Append the counter of the data in the write queue during encrypting the data
  - Ensure the counter is durable before the data flush complet



Durable Transaction in SecPM

Stage	Log content	Log counter	Data content	Data counter	Recoverable?
Prepare	Wrong	Wrong	Correct	Correct	Yes
Mutate	Correct	Correct	Wrong	Wrong	Yes
Commit	Correct	Correct	Correct	Correct	Yes

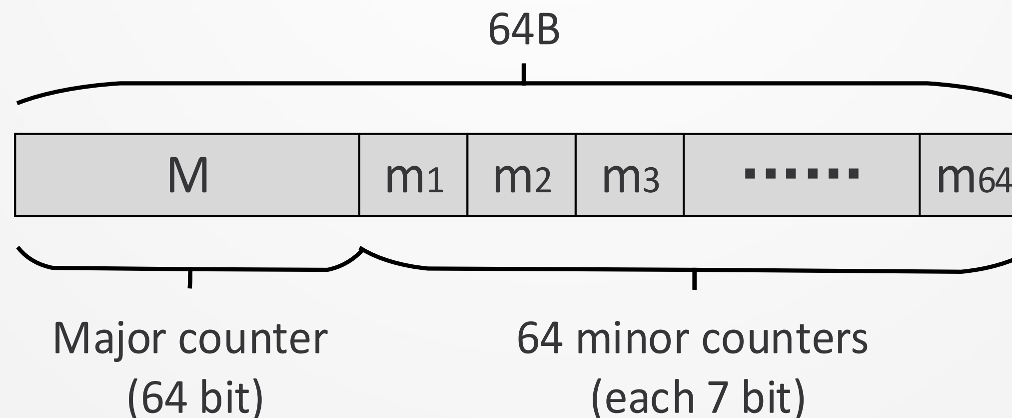
At least one of log and data is correct in whichever stage a system failure occurs

The system can be recoverable in a consistent state in SecPM

```
TX_BEGIN
    do some computation;
    // Prepare stage: backing up the data in log
    write undo log;
    flush log;
    memory_barrier();
    // Mutate stage: updating the data in place
    write data;
    flush data;
    memory_barrier();
    // Commit stage: invalidating the log
    log->valid = false;
    flush log->valid;
    memory_barrier();
TX_END
```

Counter Write Reduction (CWR) Scheme

- leveraging the spatial locality of counter storage, log and data writes
  - The spatial locality of counter storage
    - The counters of all memory lines in a page are stored in one memory line
    - Each memory line is encrypted by the major counter concatenated with a minor counter

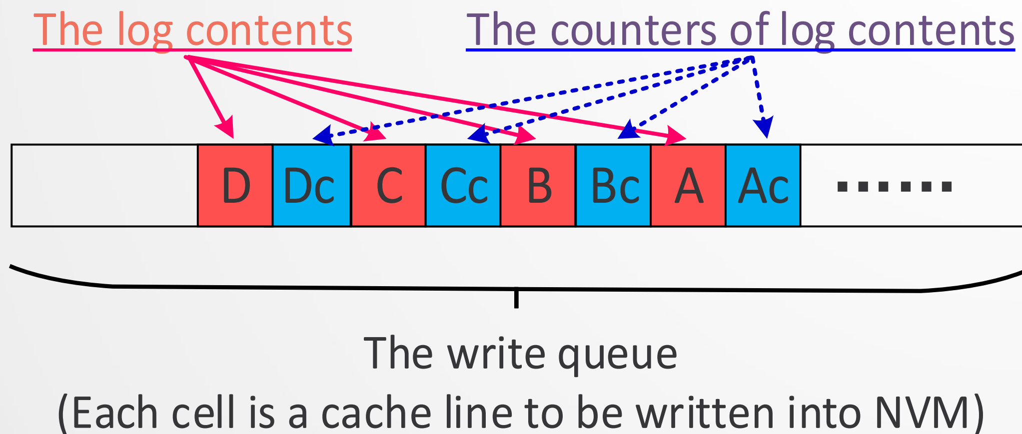


Counter Write Reduction (CWR) Scheme

- leveraging the spatial locality of counter storage, log and data writes
  - The spatial locality of counter storage
    - The counters of all memory lines in a page are stored in one memory line
    - Each memory line is encrypted by the major counter concatenated with a minor counter
  - The spatial locality of log and data writes
    - A log is stored in a contiguous region
    - Programs usually allocate a contiguous memory region for a transaction

Counter Write Reduction (CWR) Scheme

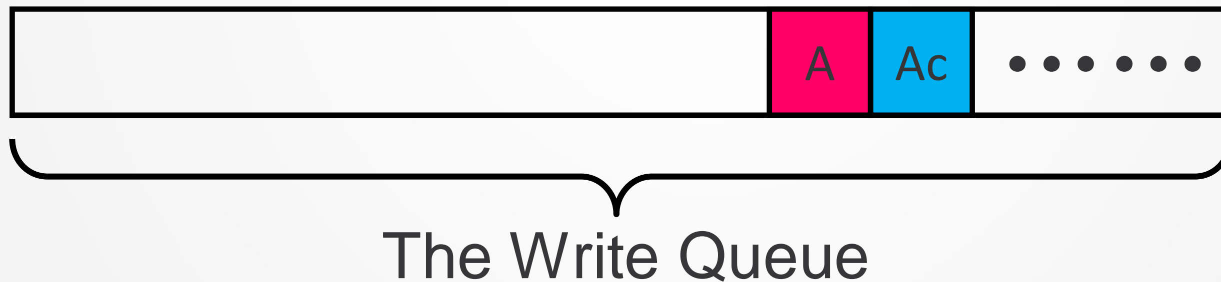
- An illustration of the write queue when writing a log
  - The counters **Ac**, **Bc**, **Cc**, and **Dc** are written into the same memory line
  - The latter cache lines contain the updated contents of the former ones (**Ac** ∈ **Bc** ∈ **Cc** ∈ **Dc**)
    - They are evicted from the write-through counter cache



Ac:	M	m1'	m2	m3	m4	⋯	m64
Bc:	M	m1'	m2'	m3	m4	⋯	m64
Cc:	M	m1'	m2'	m3'	m4	⋯	m64
Dc:	M	m1'	m2'	m3'	m4'	⋯	m64

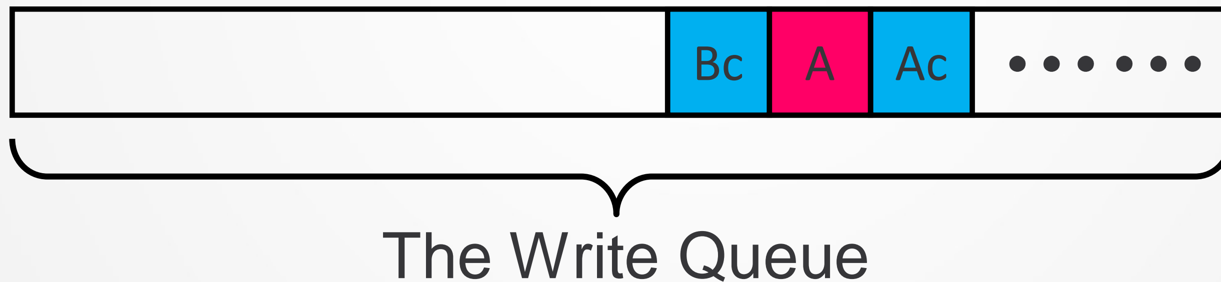
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



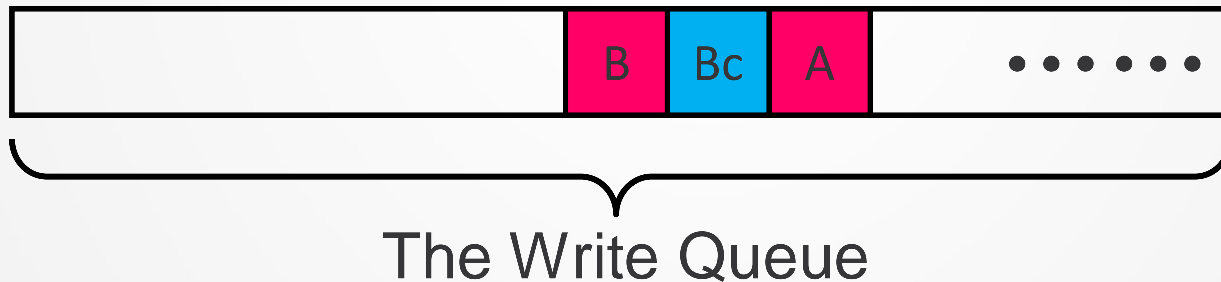
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



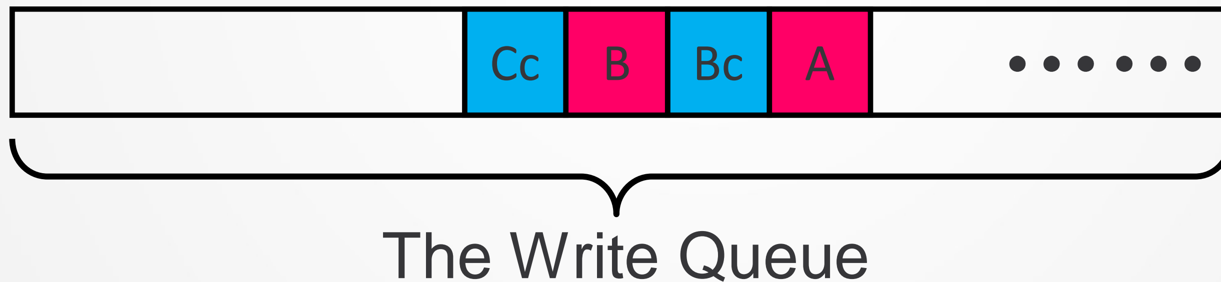
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



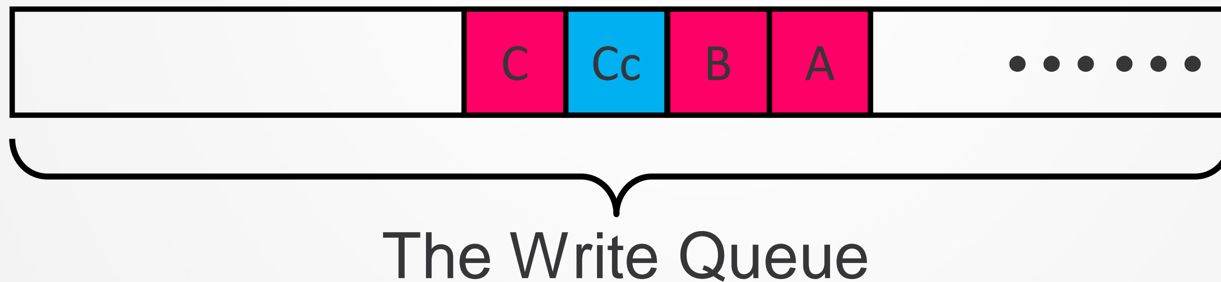
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



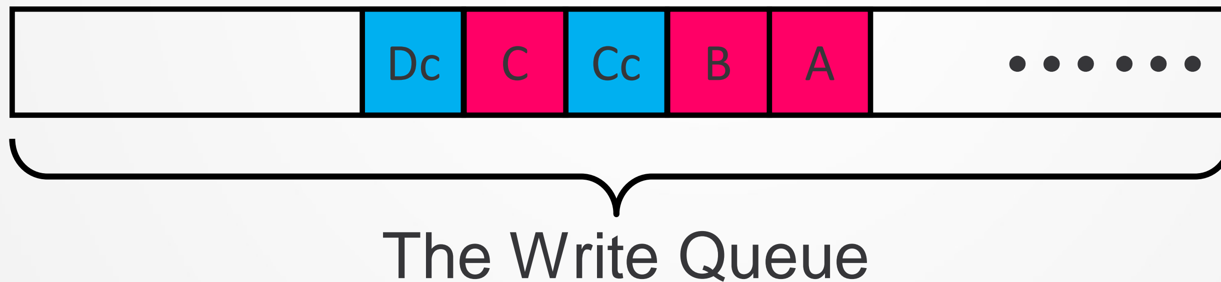
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



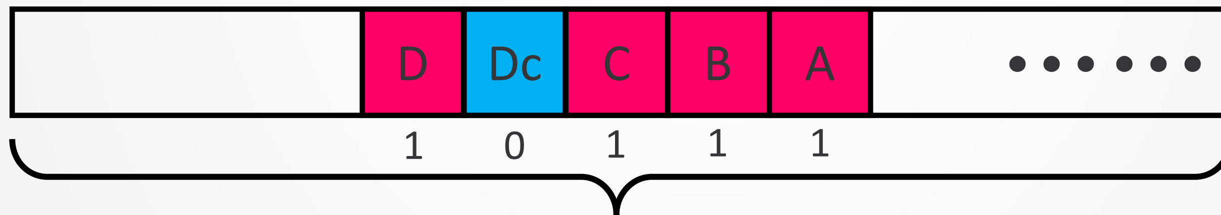
Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data



Counter Write Reduction (CWR) Scheme

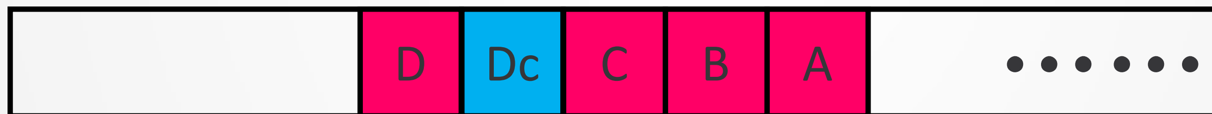
- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data
  - Using a flag to distinguish whether a cache line is from CPU caches or the counter cache



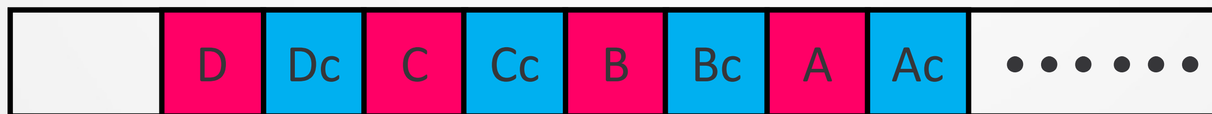
(1: from CPU caches; 0: from the counter cache)

Counter Write Reduction (CWR) Scheme

- When a new cache line arrives, remove the existing cache line with the same physical address in the write queue
  - Without causing any loss of data
  - Using a flag to distinguish whether a cache line is from CPU caches or the counter cache



With CWR



Without CWR

Performance Evaluation

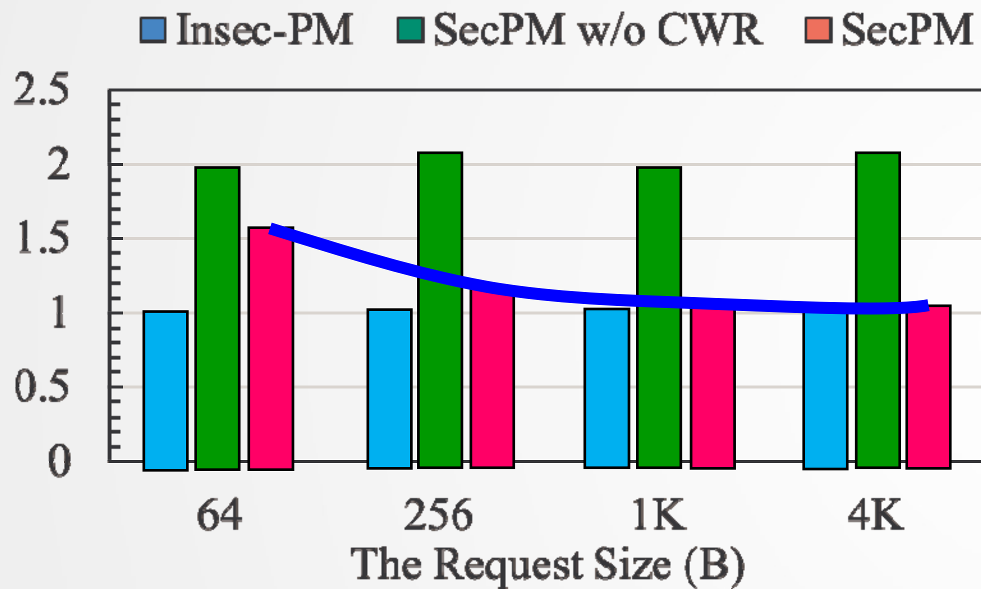
- Model NVM using gem5 and NVMain

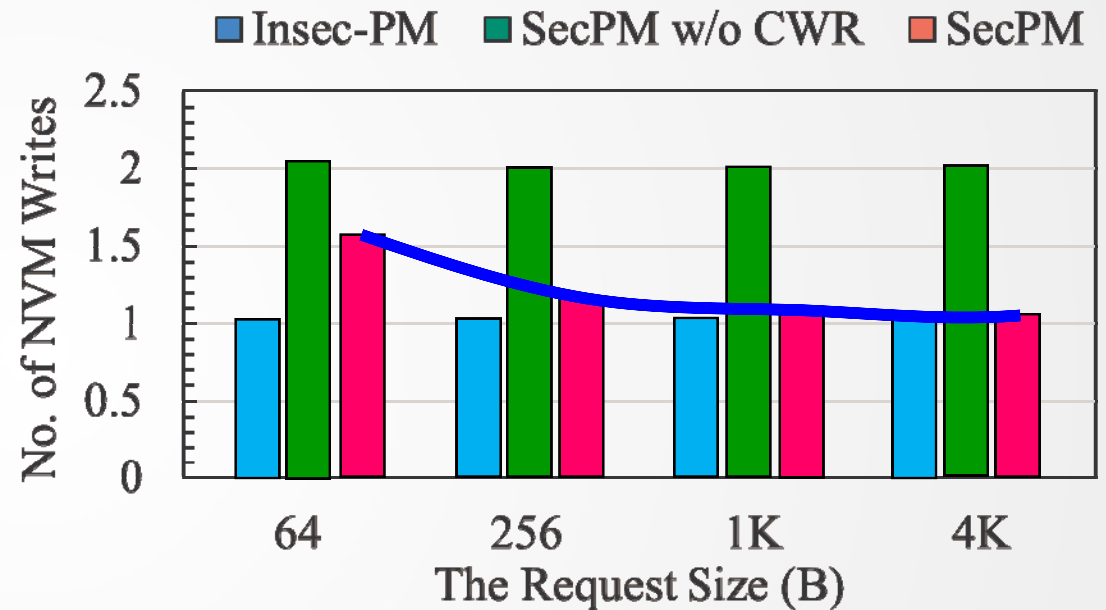
CPU and Caches
X86-64 CPU, at 2 GHz
32KB L1 data & instruction caches
2MB L2 cache
8MB shared L3 cache

Memory Using PCM
Capacity: 16GB
Read/write latency: 150/450ns
Encryption/decryption latency: 40ns
Counter cache: 1MB, 10ns latency

- Storage benchmarks
  - A hash table based key-value store
  - A B-tree based key-value store

The Number of NVM Write Requests

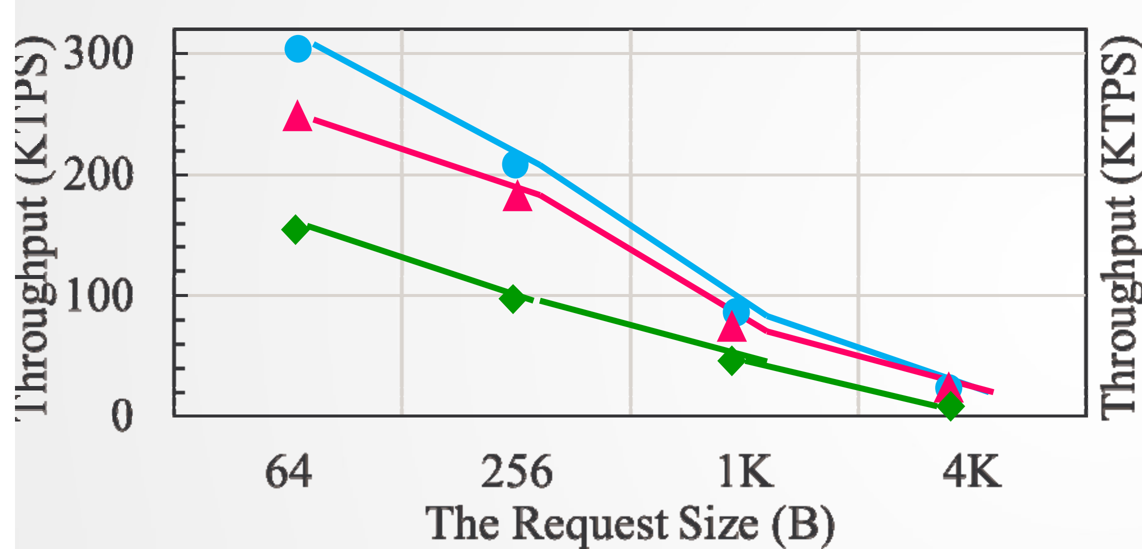


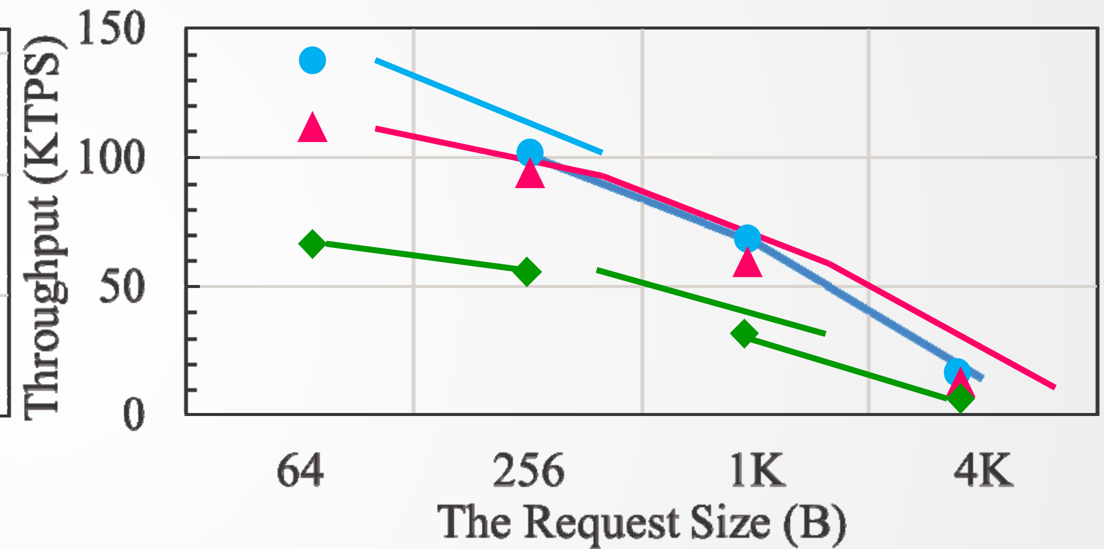


Compared with the SecPM w/o CWR, SecPM significantly reduces NVM writes

Compared with Insec-PM, SecPM only causes 13%, 5%, and 2% more writes when the request size is 256B, 1KB, and 4KB, respectively

Transaction Throughput





Compared with the SecPM w/o CWR, SecPM significantly increases the throughput by 1.4 ～ 2.1 times

Compared with InsecPM, SecPM incurs a little throughput reduction, due to the more NVM writes and the latency overhead of data encryption

Conclusion

- Both security and persistence of NVM are important
- Simply combining existing persistence schemes with memory encryption is inefficient
  - Crash inconsistency
  - Significant performance degradation
- This paper proposes SecPM to bridge the gap between security and persistence
  - Guarantee consistency via a counter cache write-through (CWT) scheme
  - Improve performance via a locality-aware counter write reduction (CWR) scheme

Thanks! Q&A