Blurred Persistence in Transactional Persistent Memory

Youyou Lu, Jiwu Shu, Long Sun

Tsinghua University

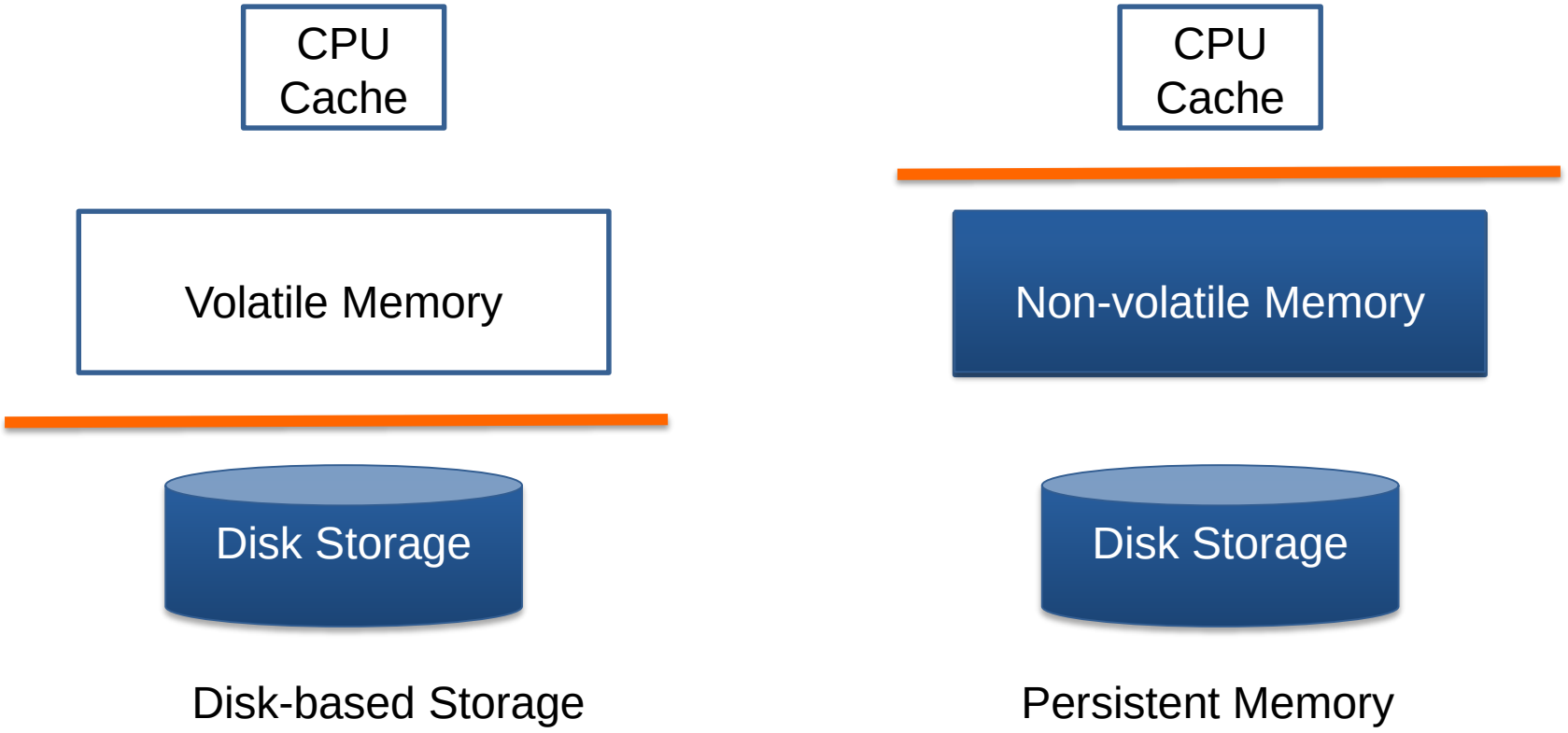
Overview

- Problem: high performance overhead in ensuring storage consistency of persistent memory
- Our Goal: to propose a software solution to reduce transactional overhead for persistent memory
- Key Idea: Blurred Persistence
  - Allow the volatile (uncommitted) data to be persisted **–-> XIL (Execution in Log)**
    - By reorganizing the memory log
  - Allow the to-be-persisted (checkpointed) data to stay volatile **–-> VCBP (Volatile Checkpoint and Bulk Persistence)**
    - Leveraging the persistent copies
    - Maintaining the overwrite order
- Results: improves system performance by 56.3% to 143.7%

Outline

- Introduction and Background
- Opportunities and Challenges
- Our Approach: Blurred Persistence
- Evaluation
- Conclusion

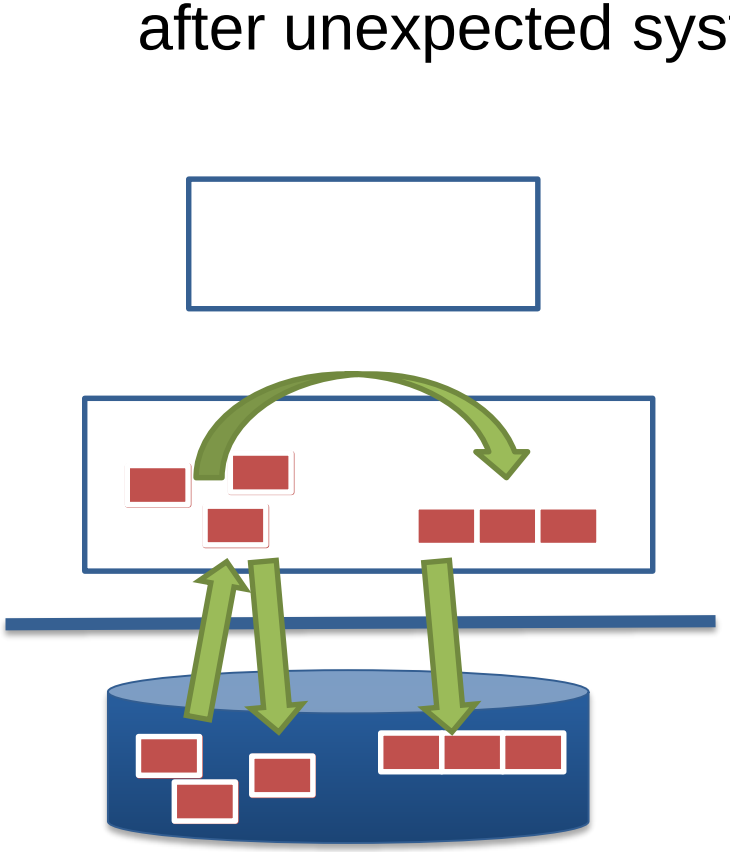
Persistent Memory



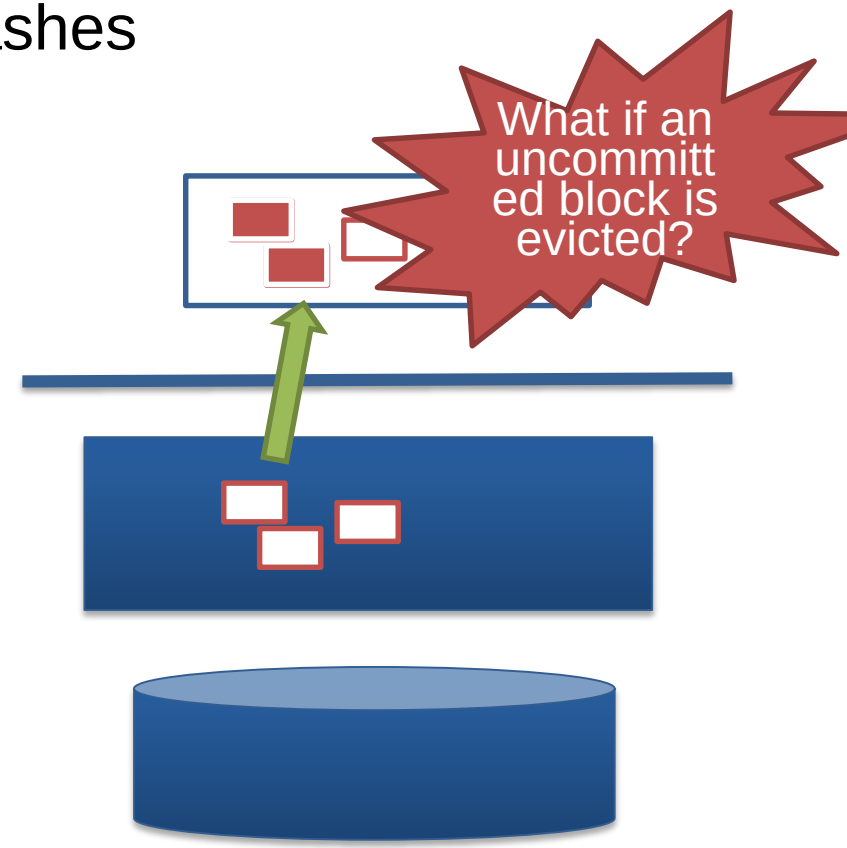
- The volatility-persistence boundary moves up

Storage consistency

- Atomicity and durability
- A storage system can recover to a consistent state after unexpected system crashes



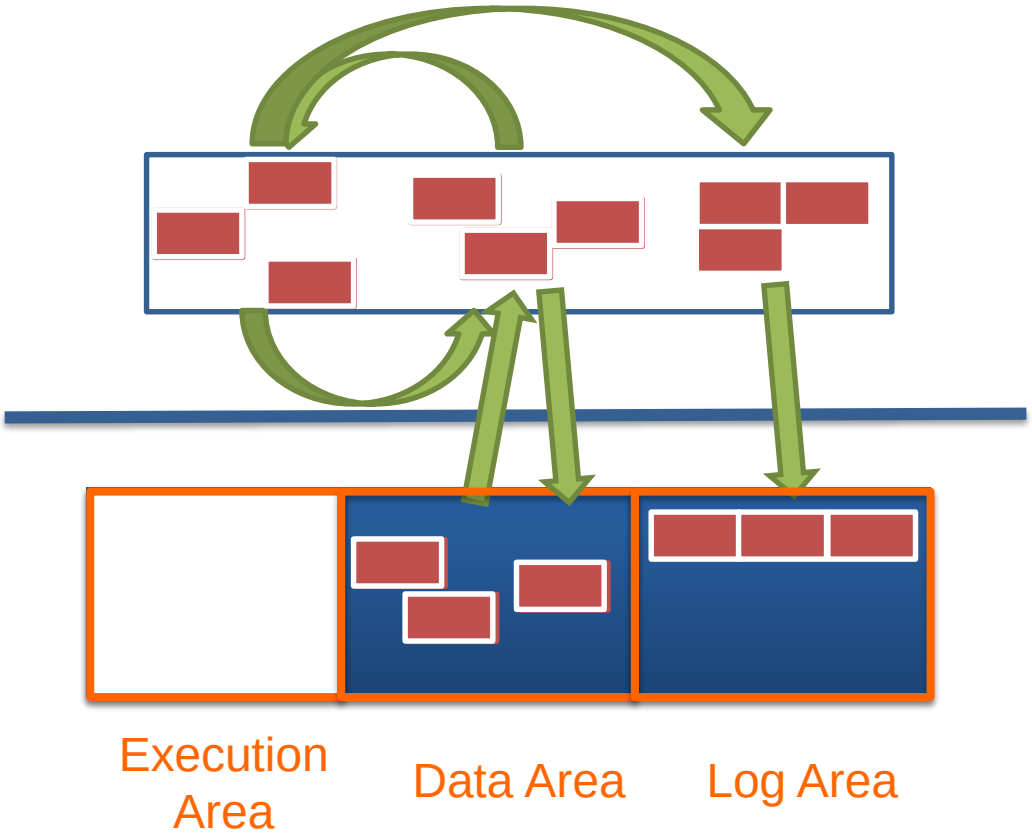
Disk-based Storage



Persistent Memory

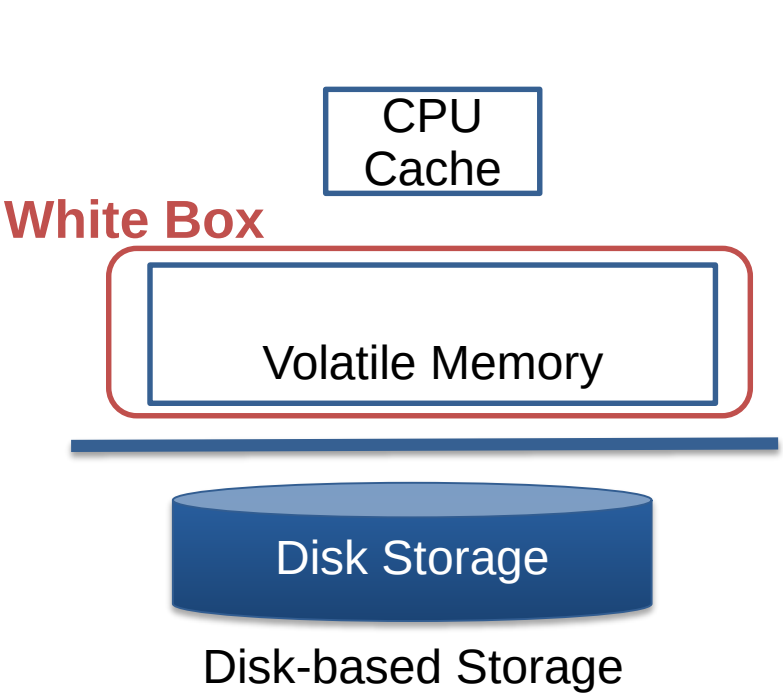
Transactional Persistent Memory

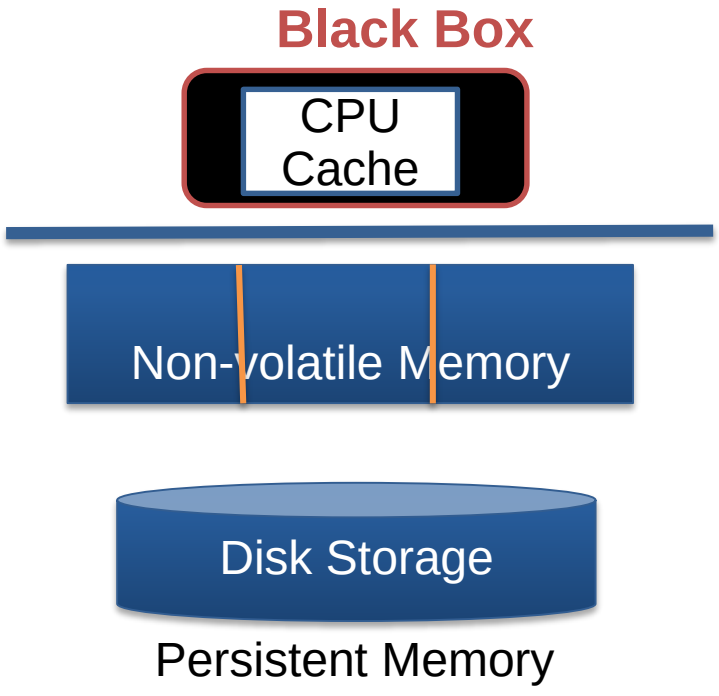
- Separated memory areas



Problems

- High transactional overhead
  - Data copies between memory areas
  - Forced persistence using flush and barrier commands
    - Clflush and mfence can add extra 250ns latency
- Root cause: white box vs. black box

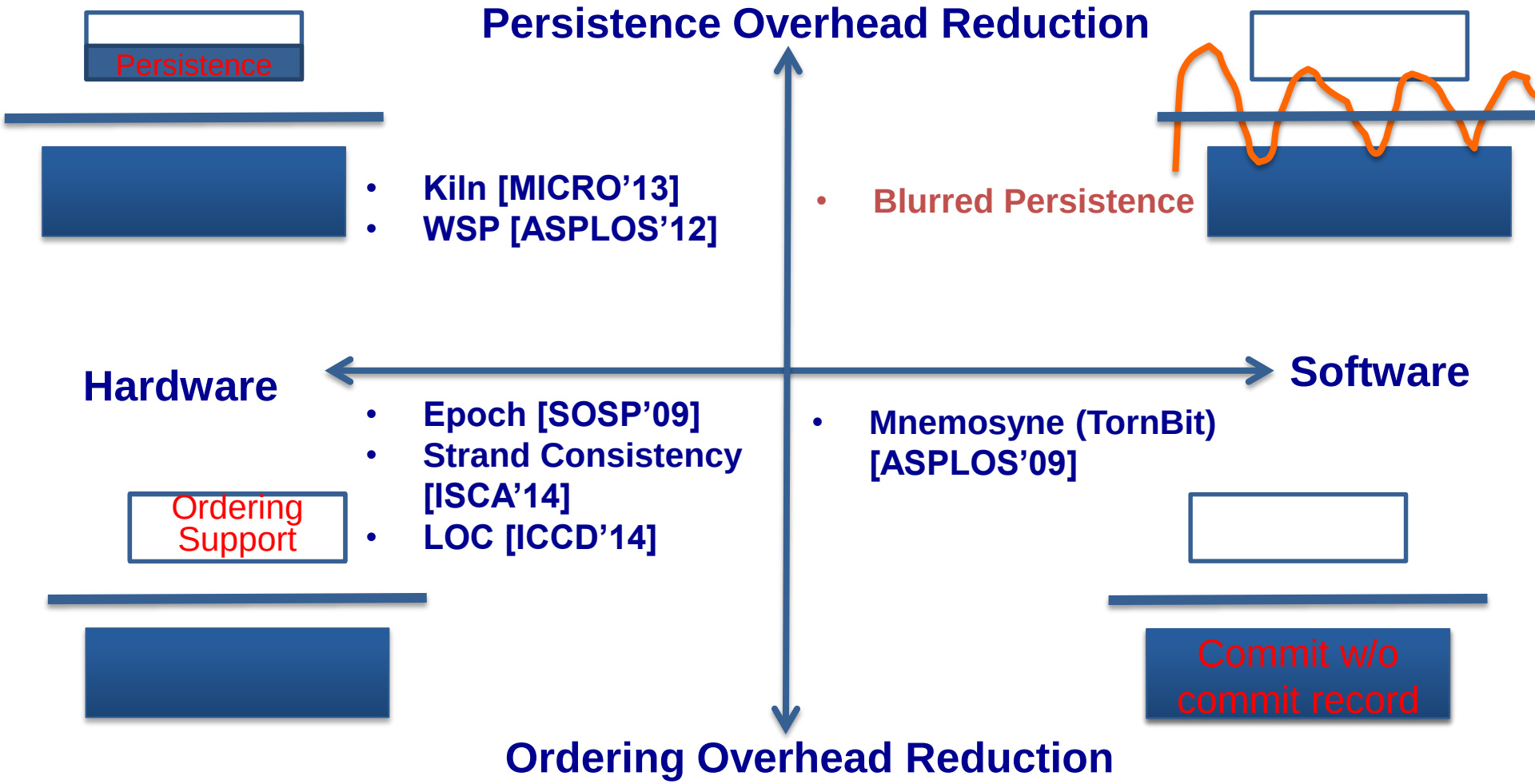




Outline

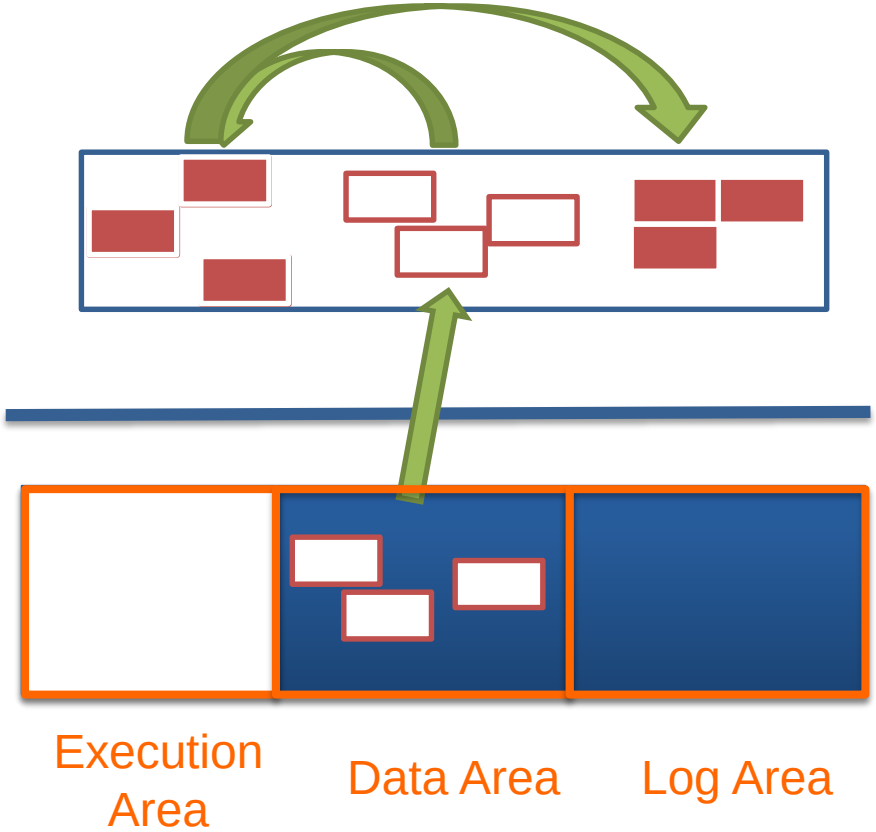
- Introduction and Background
- Opportunities and Challenges
- Our Approach: Blurred Persistence
- Evaluation
- Conclusion

Existing Solutions

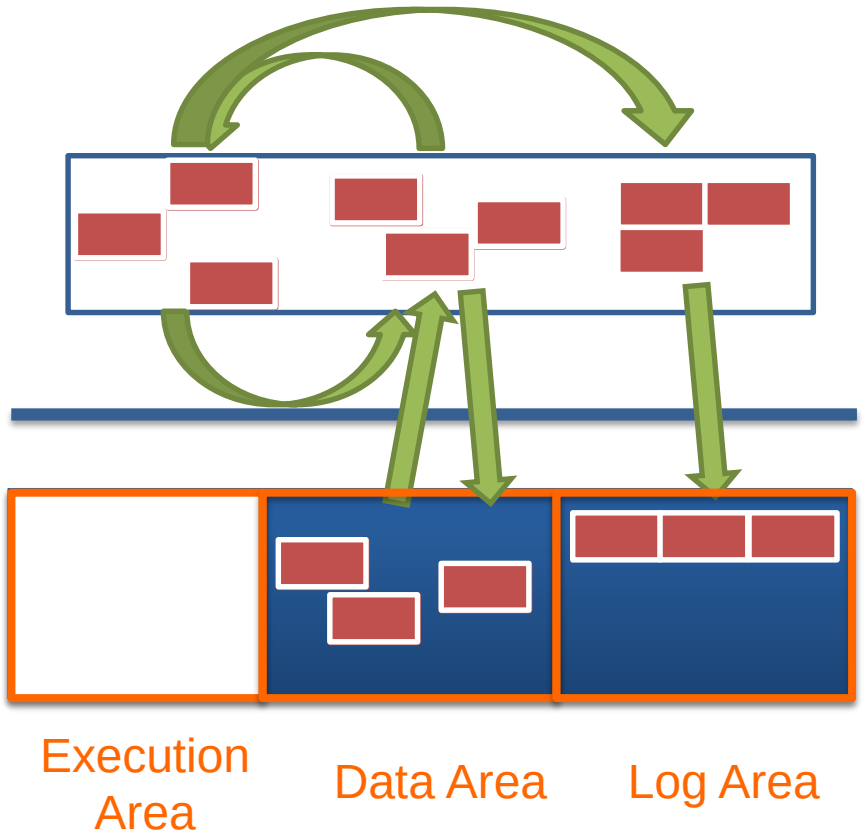


Observations and Opportunities

- Volatile copies
  - Remove duplicated in the execution area
  - > execution in log



- Persistent copies
  - No need to force persistence if the data block has another persistent copy
  - > volatile checkpoint



Key Ideas

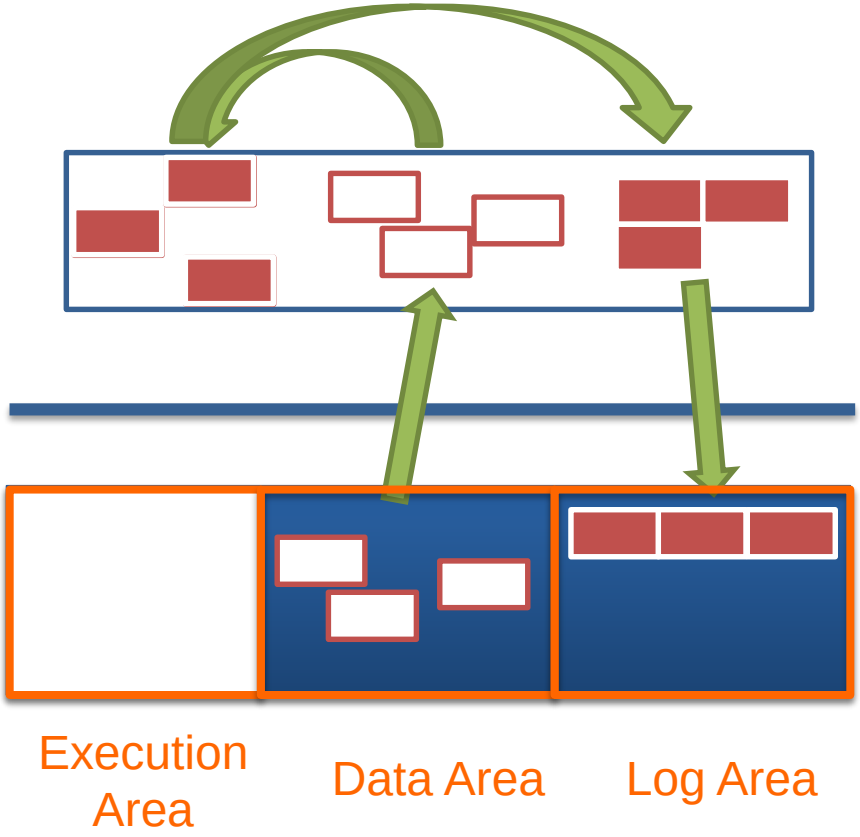
- Blurred Persistence
  - Allow the volatile (uncommitted) data to be persisted
  - –> XIL (Execution in Log)
    - What if a system crash? How to identify the uncommitted data?
  - Allow the to-be-persisted (checkpointed) data to stay volatile
  - –> VCBP (Volatile Checkpoint with Bulk Persistence)
    - How to provide durability? How to identify the not-committed data?
    - How to keep the write order?

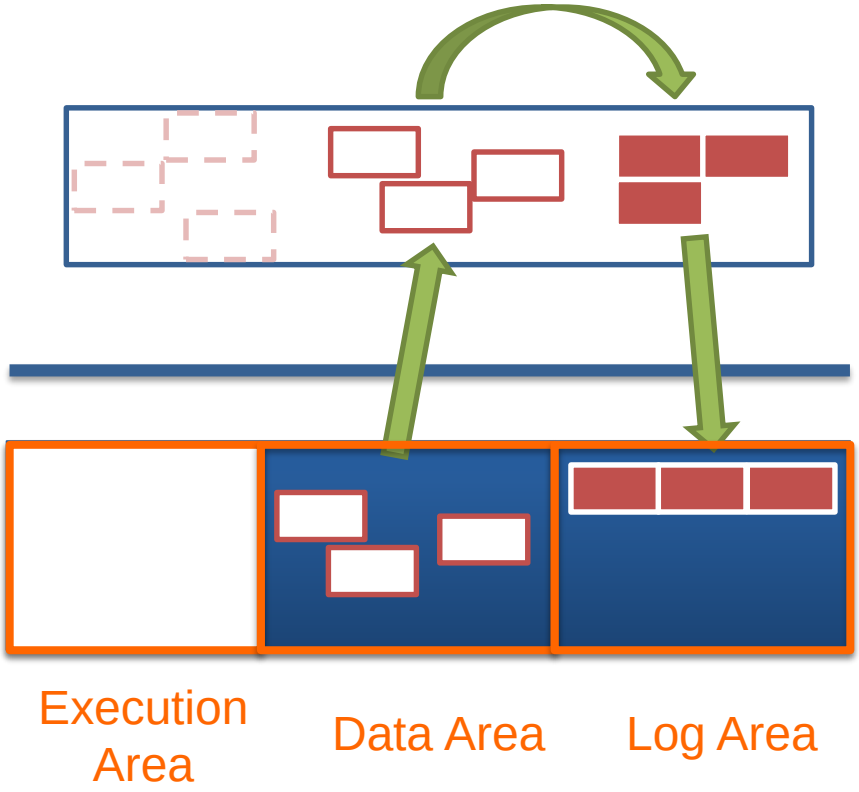
Outline

- Introduction and Background
- Opportunities and Challenges
- Our Approach: Blurred Persistence
- Evaluation
- Conclusion

1. Execution in Log (XIL)

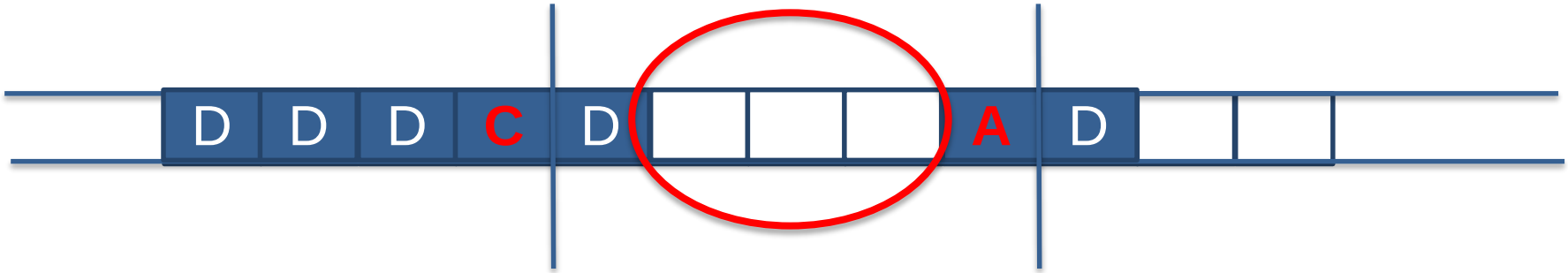
- Execution in Log
  - Reduce data copies in the execution area



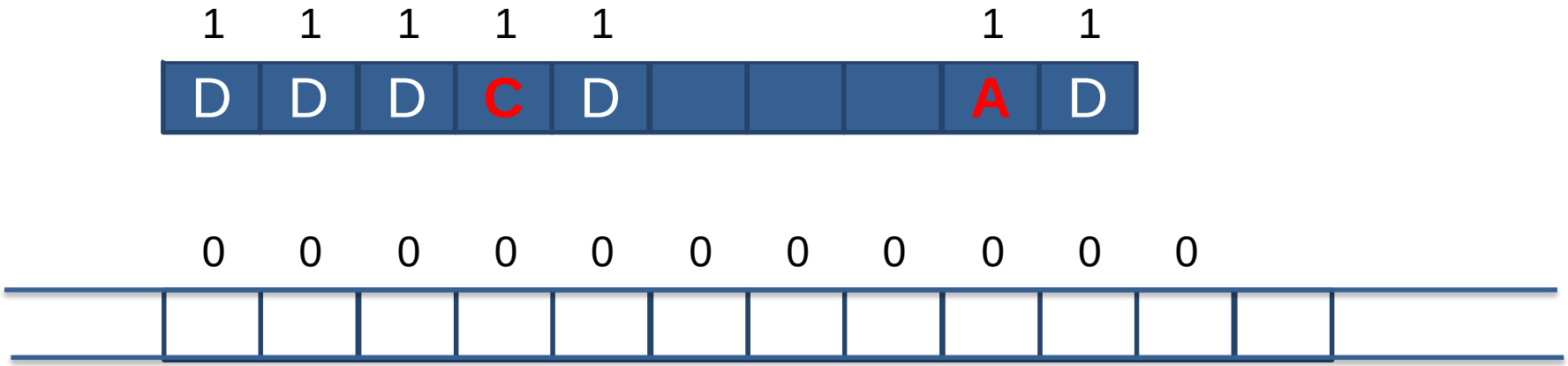


- Challenge: How to identify the uncommitted data that are persisted after system crashes?
  - Cause: hardware cache eviction of the CPU cache

- Log Holes
  - Uncommitted data blocks: allocated but not written

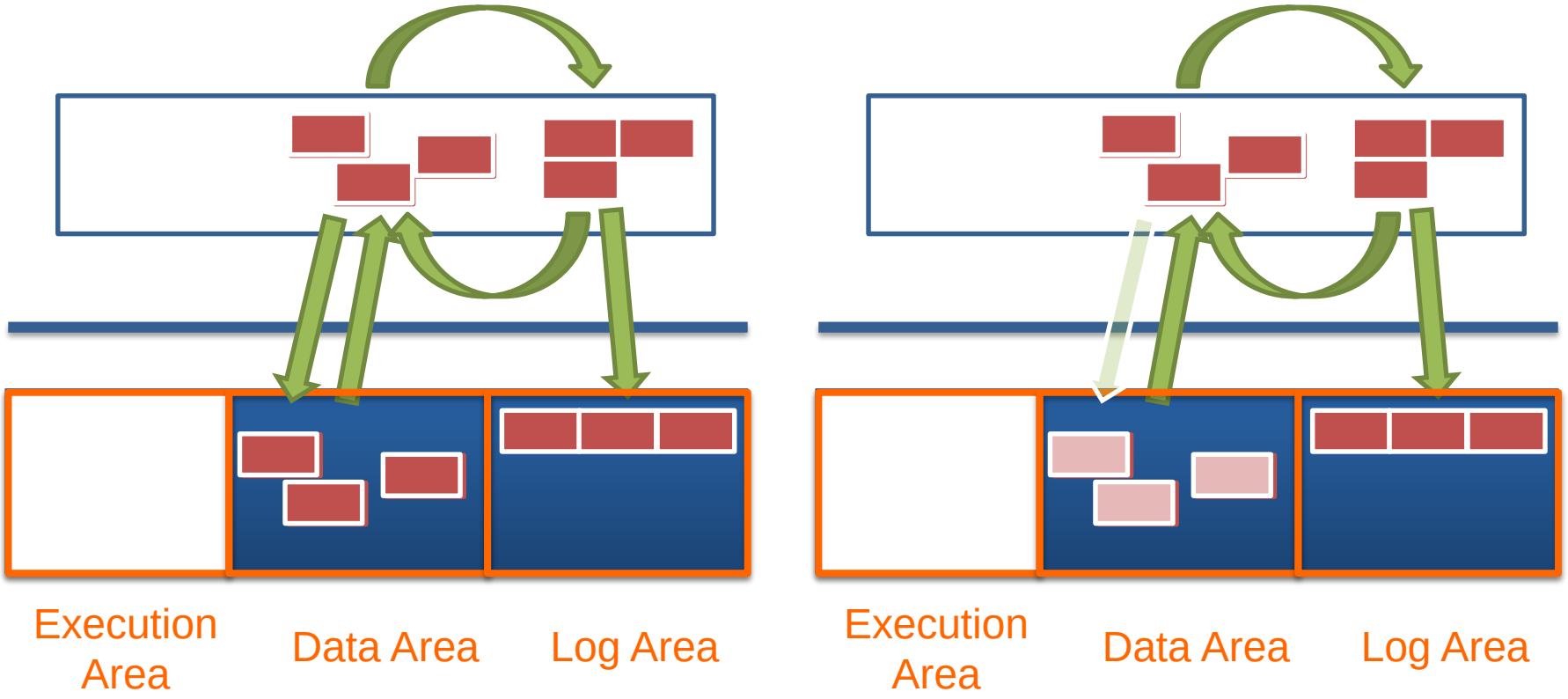


- Solution: Memory Log Reorganization
  - Identify the non-written blocks
    - TornBit technique borrowed from Mnemosyne[ASPLOS'11]
  - Identify the uncommitted blocks that are written
    - Consecutively allocate log records for each transaction
      - For multi-thread applications, each thread is allocated with a private log, but the head of the private log is globally visible
    - Add descriptive metadata in the commit/abort record
      - e.g., a backpointer to the commit record of the last committed transaction



2. Volatile Checkpoint with Bulk Persistence (VCBP)

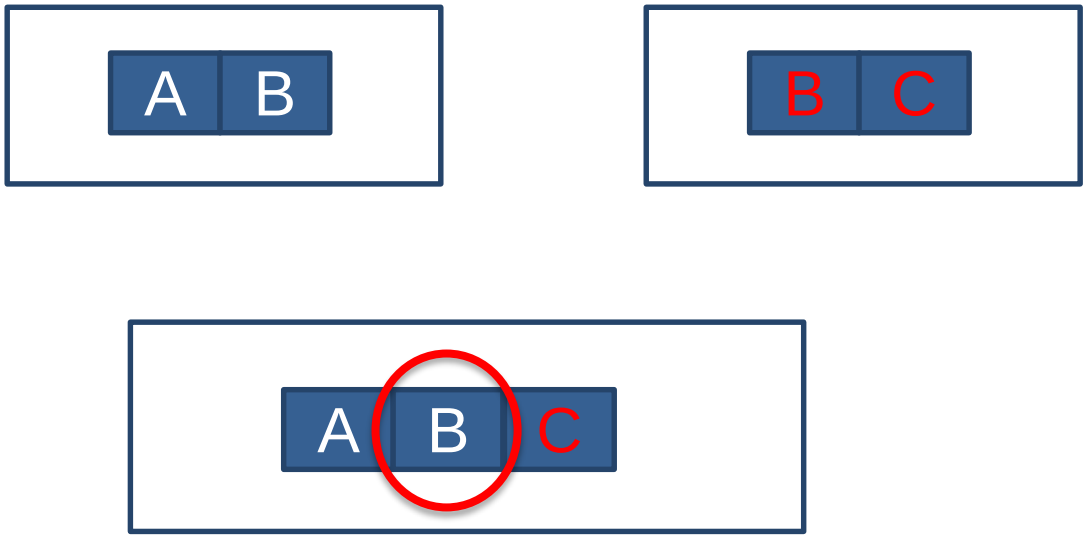
- Volatile Checkpoint
  - make the committed data visible
- Bulk Persistence
  - ensure the durability property (after log truncation)



- Challenges:
  - (1) Volatile checkpoint: committed data are volatile?
  - (2) Bulk persistence: uncommitted data are forced persisted

- Solutions:
  - (1) leverage the persistent copies in the log area
  - (2) make the uncommitted data identifiable in persistent memory
    - If in execution area, it is OK (all data in execution area are discarded even if they are evicted to memory)
    - If in log area, using XIL techniques
    - Data area does not have uncommitted data, but need to keep the persistence order of a concurrently-updated block

- Another Challenge: overwrite order of a concurrently-updated data block from multiple transactions



Outline

- Introduction and Background
- Opportunities and Challenges
- Our Approach: Blurred Persistence
- Evaluation
- Conclusion

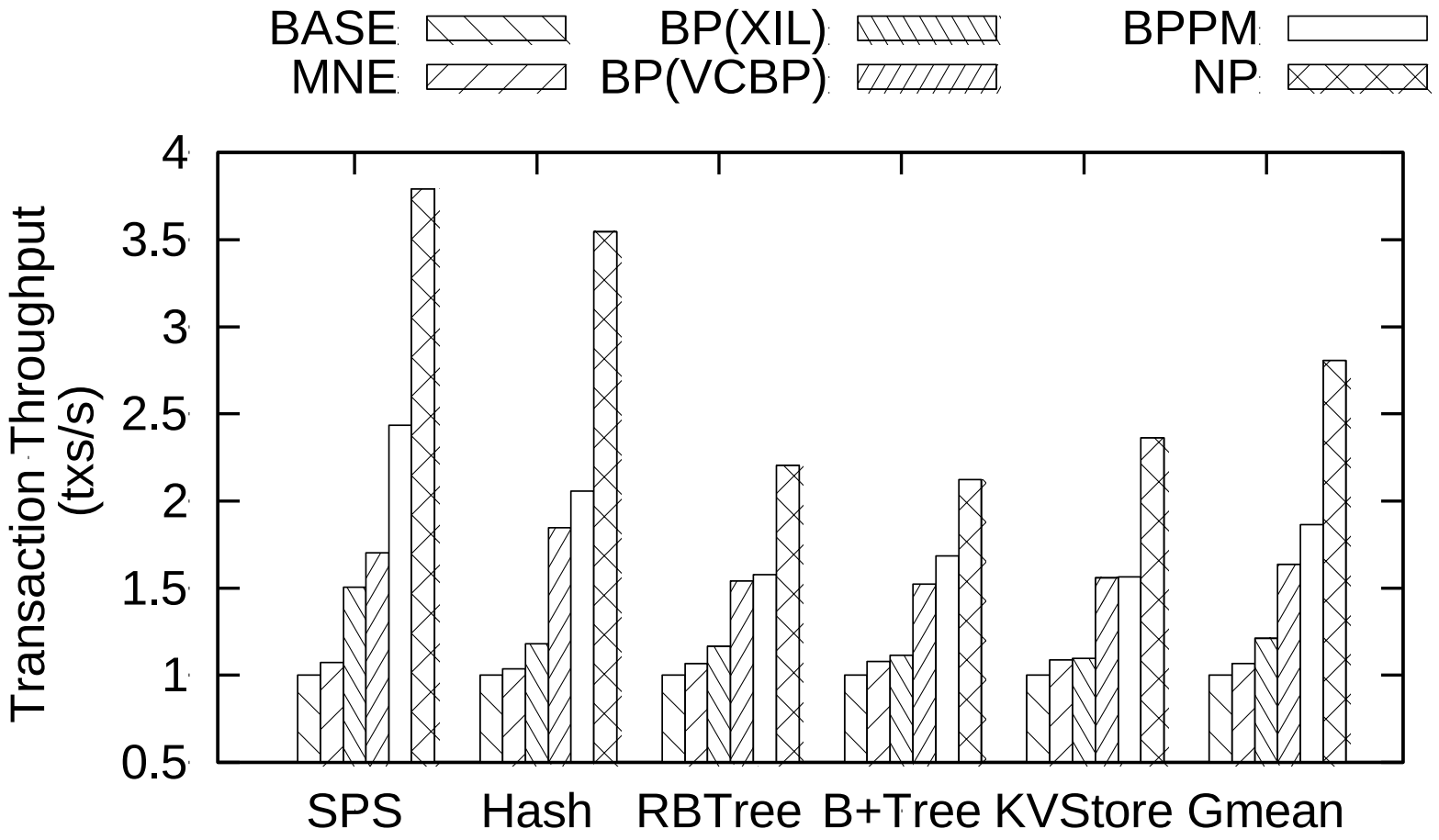
Experimental Setup

- Blurred-Persistence Persistent Memory (BPPM)
  - Software transactional memory (TinySTM) + persistence support
  - Intel STM compiler

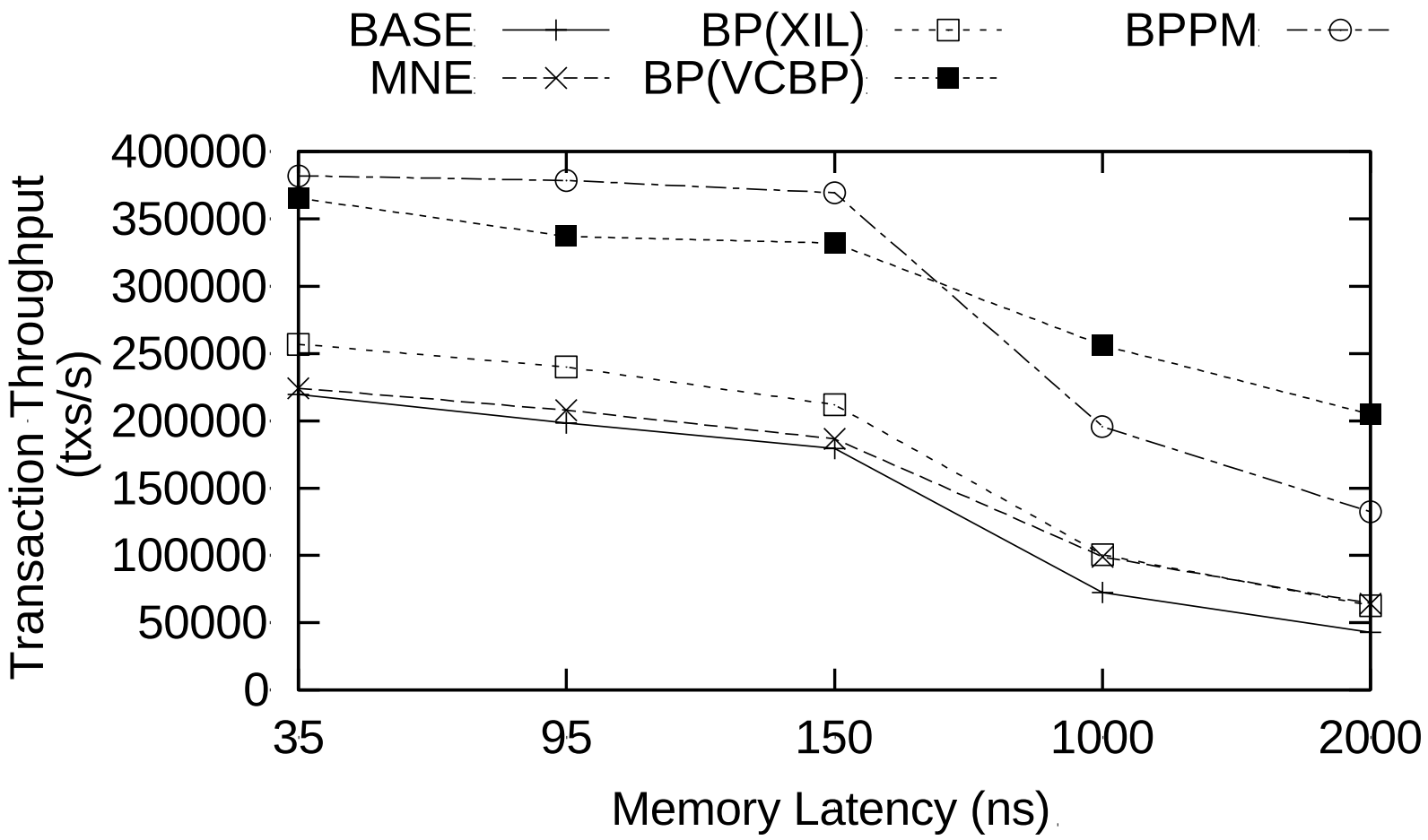
- Evaluated Systems
  - Baseline (BASE), Mnemosyne (MNE), No Persistence (NP)
  - *BP(XIL)*, *BP(VCBP)*, *BPPM*

- Workloads
  - Data array swaps, hash table, red-black tree, B+ tree,
  - Key-value store

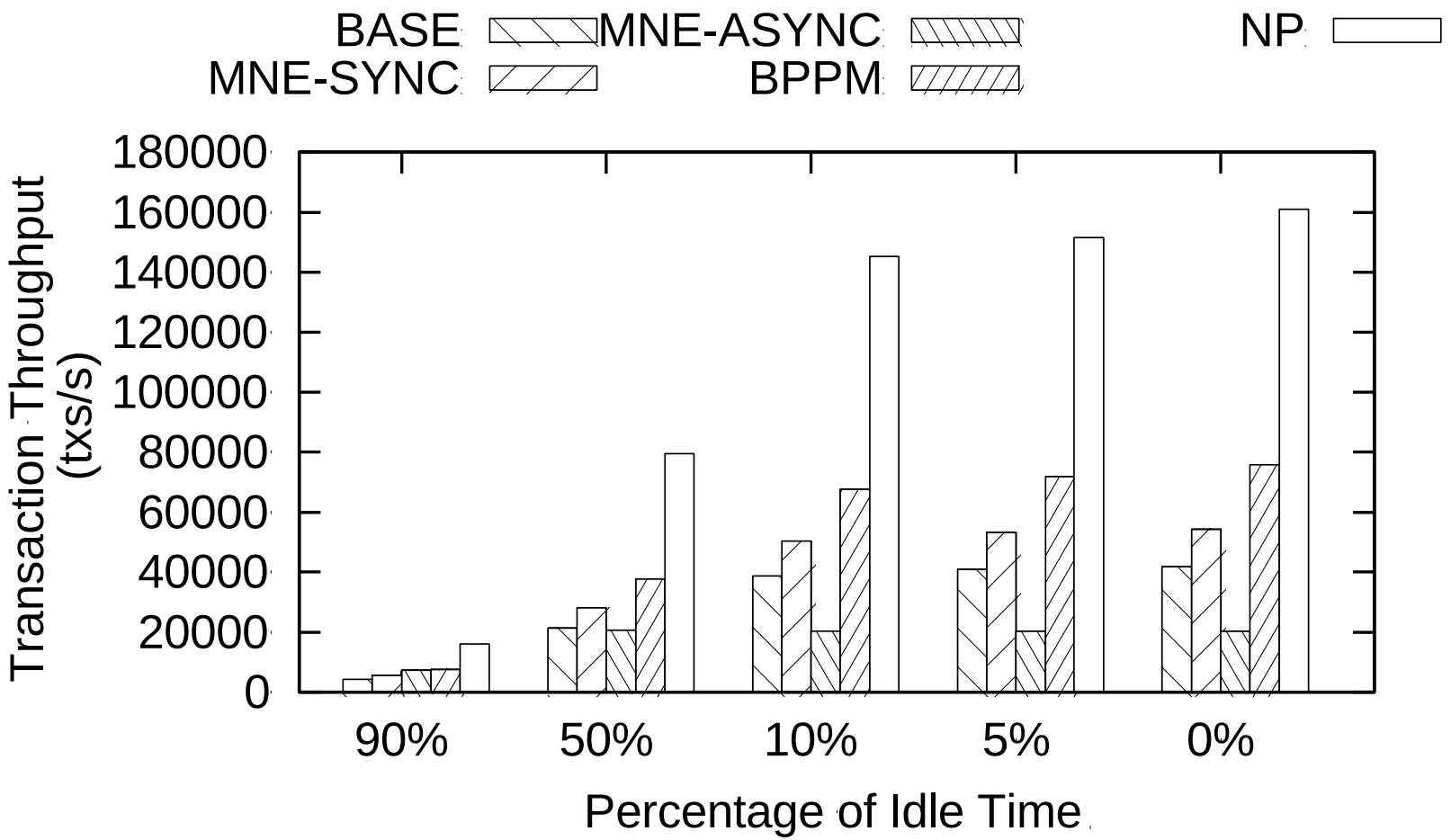
Overall Performance



Sensitivity to Memory Latency



Sensitivity to Transaction Idle Time



Outline

- Introduction and Background
- Opportunities and Challenges
- Our Approach: Blurred Persistence
- Evaluation
- Conclusion

Overview

- Blurred Persistence: a general software solution to reduce transactional overhead in persistent memory
- Two Techniques:
  - Execution in Log (XIL): Allow the volatile (uncommitted) data to be persisted
    - By reorganizing the memory log
  - Volatile Checkpoint with Bulk Persistence (VCBP): Allow the to-be-persisted (checkpointed) data to stay volatile
    - Leveraging the persistent copies
    - Maintaining the overwrite order
- Results: improves system performance by 56.3% to 143.7%

Blurred Persistence in Transactional Persistent Memory

Youyou Lu

Tsinghua University

Email: luyouyou@tsinghua.edu.cn