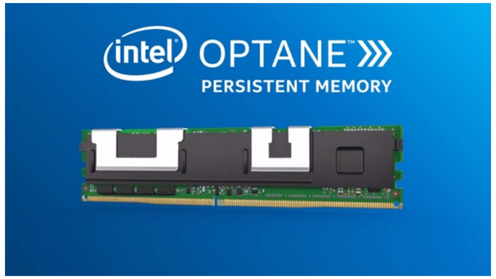
ARCHTM: ARCHITECTURE-AWARE, HIGH PERFORMANCE TRANSACTION FOR PERSISTENT MEMORY

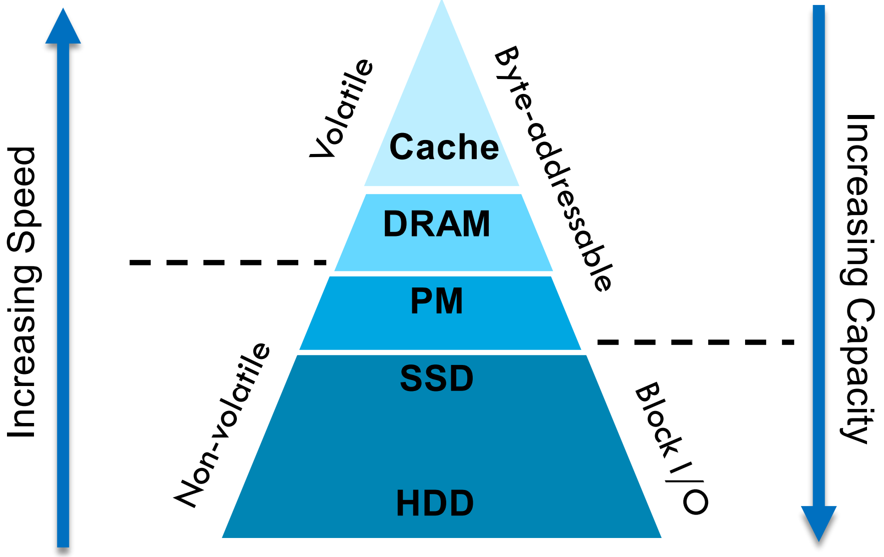
Kai Wu, Jie Ren, Ivy Peng[+], Dong Li

University of California, Merced
Lawrence Livermore National Laboratory[+]

Persistent Memory (PM) Has Arrived

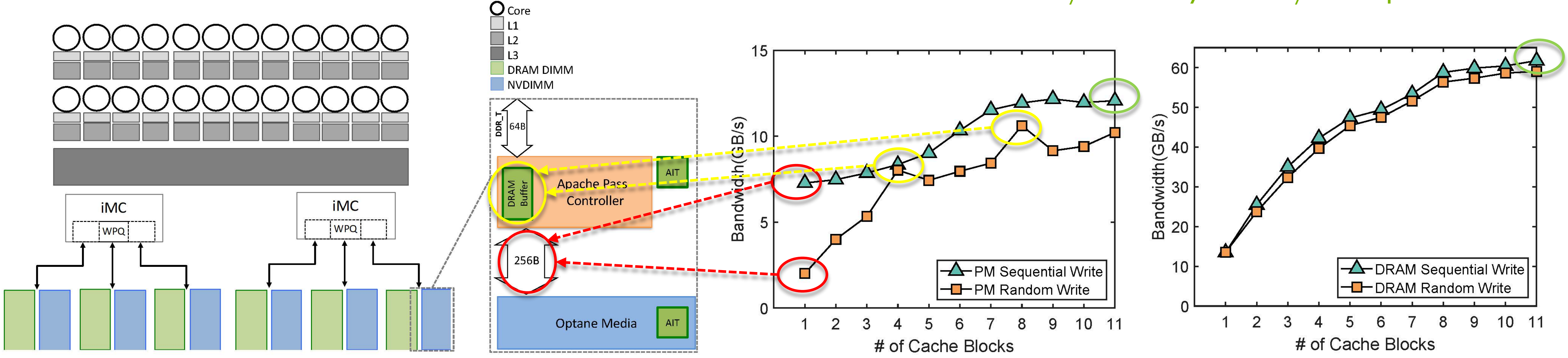


- Memory-like performance
  - ~100x faster than SSDs
  - Byte-addressability
- Storage-like characteristics
  - Non-volatility
  - High density
    - Each socket can have as much as 4.5 TB


Memory & Storage Hierarchy

PM Architecture & Performance Characterization

Write bandwidth to DRAM reaches 60 GB/s but only 13 GB/s to Optane PM



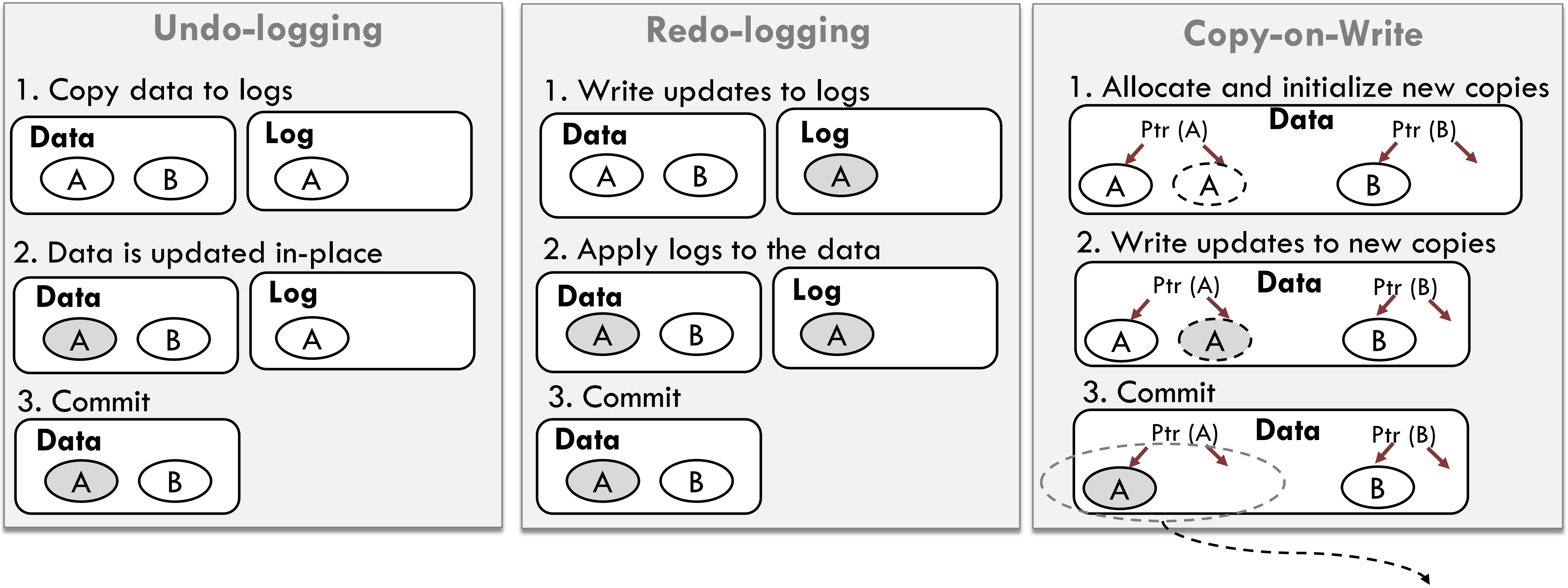
- ☐ Reducing write traffic on PM is critical
- ☐ PM microarchitecture (e.g., internal buffer and data block size) has a significant impact on the write performance of PM
  - Avoid small random writes
  - Leverage the combining buffer hardware to coalesce writes inside PM

Transactions on Persistent Memory

- Failure-atomic transaction is a critical mechanism for accessing and manipulating data on PM
- Existing PM transaction systems are implemented into two major paradigms – logging (undo & redo) and copy-on-write
- Both paradigms do not consider the performance impact of PM architecture characteristics

Issues of Existing PM Transactions

X Data from last commit
X Newly written data
X The new copy of data



Undo-logging

1. Copy data to logs
2. Data is updated in-place
3. Commit

Redo-logging

1. Write updates to logs
2. Apply logs to the data
3. Commit

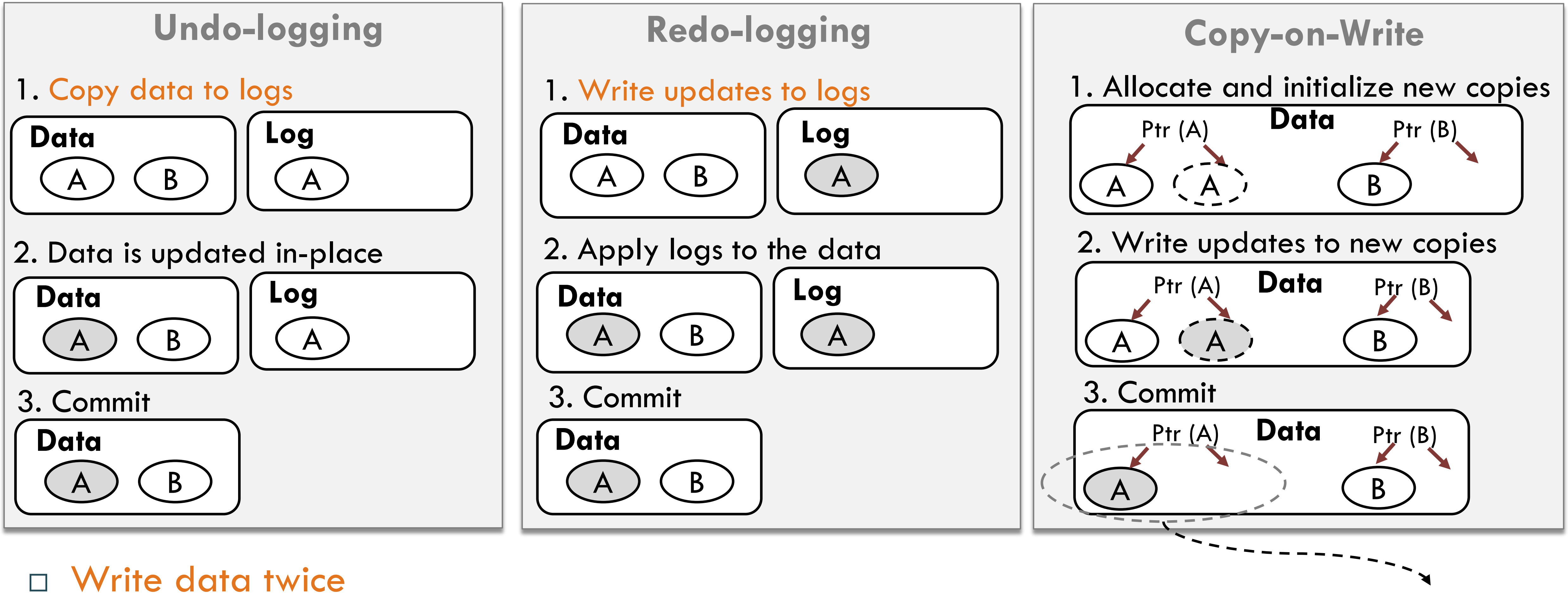
Copy-on-Write

1. Allocate and initialize new copies
2. Write updates to new copies
3. Commit

Reset pointers and free old copy

Issues of Existing PM Transactions

X Data from last commit
X Newly written data
X The new copy of data



Undo-logging

1. Copy data to logs
2. Data is updated in-place
3. Commit

Redo-logging

1. Write updates to logs
2. Apply logs to the data
3. Commit

Copy-on-Write

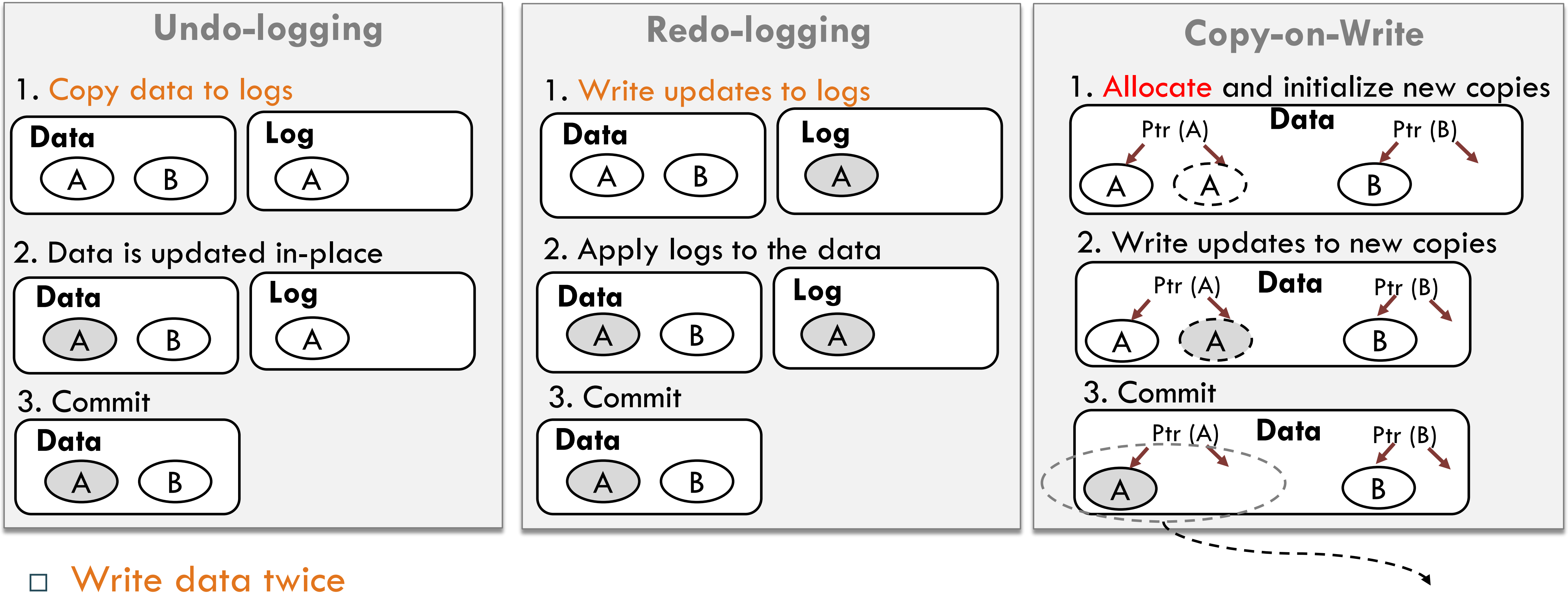
1. Allocate and initialize new copies
2. Write updates to new copies
3. Commit

Reset pointers and
free old copy

- ☐ Write data twice
- ☐ In-place update to the data could cause concurrent random writes

Issues of Existing PM Transactions

X Data from last commit
X Newly written data
X The new copy of data



Undo-logging

1. Copy data to logs
2. Data is updated in-place
3. Commit

Redo-logging

1. Write updates to logs
2. Apply logs to the data
3. Commit

Copy-on-Write

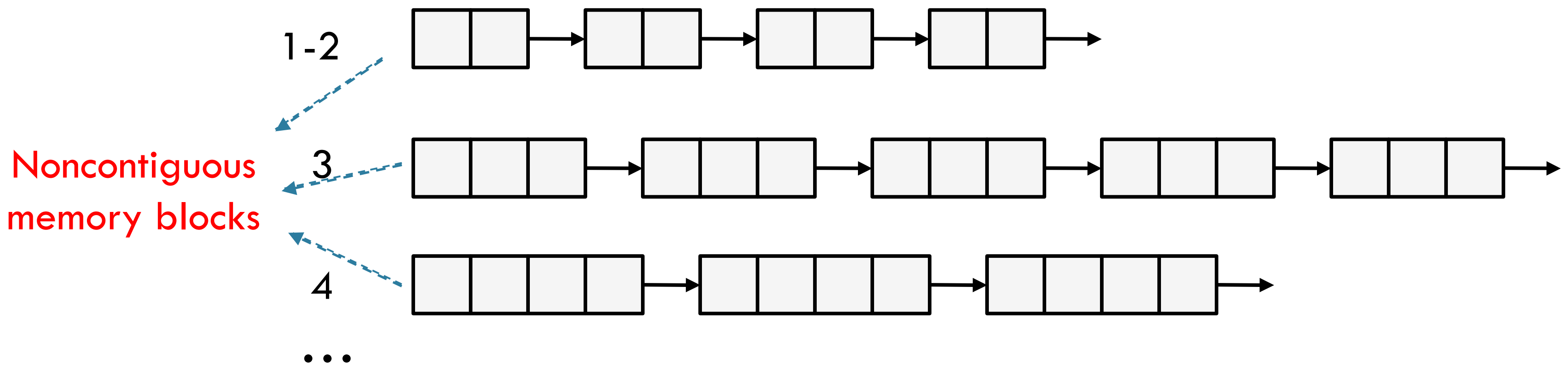
1. Allocate and initialize new copies
2. Write updates to new copies
3. Commit

Reset pointers and free old copy

- ☐ Write data twice
- ☐ In-place update to the data could cause concurrent random writes
- ☐ Frequent metadata updates causes many small random writes

Issues of Memory Allocation for PM Transactions

- Existing memory allocation implementations use multiple free lists, each for a different allocation size



  - Multiple free lists could cause consecutive allocation requests of different sizes to go to different free lists
  - Return freed memory blocks to thread-local free lists for reuse

Reduce the opportunity to leverage the combining buffer hardware to coalesce writes inside PM

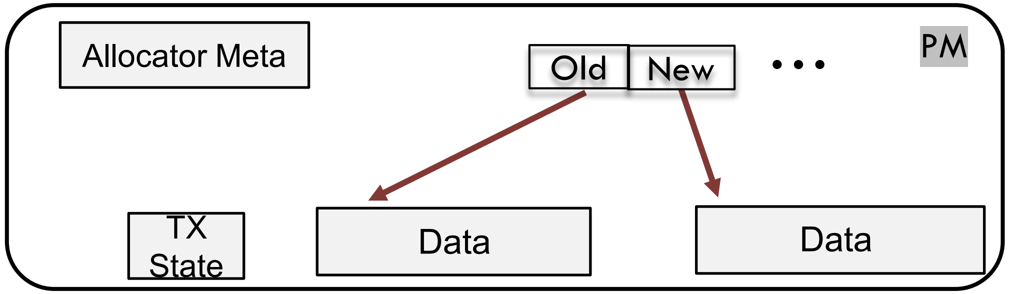
Design Goals of ArchTM

- ArchTM: an architecture-aware PM transaction system
  - Reduce write traffic on PM

    Logless → Use copy-on-write
  - Avoid small writes on PM
  - Encourage coalescable writes on PM

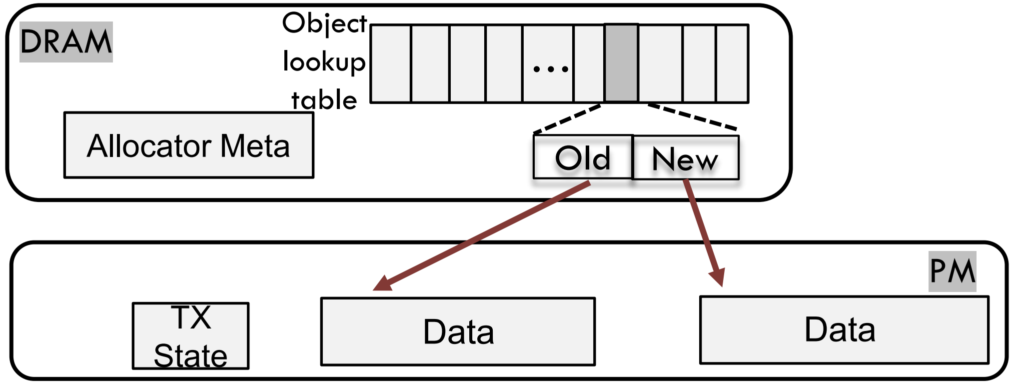
Avoid Small Writes on PM

- Minimize metadata modifications on PM with guaranteed crash consistency



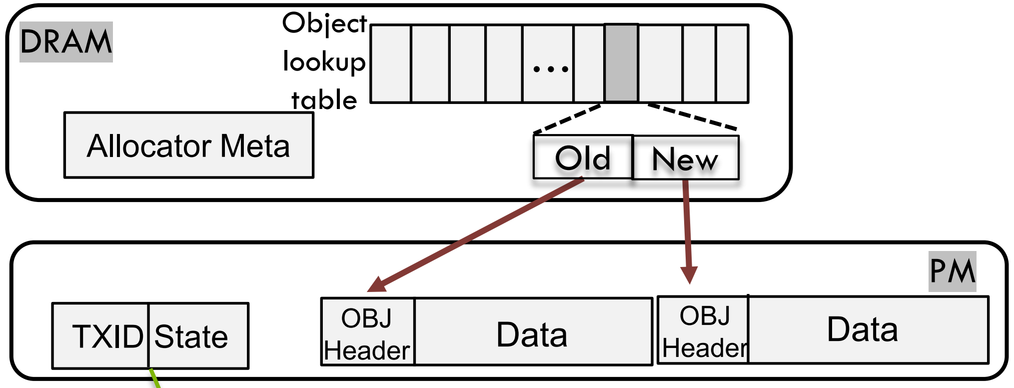
Avoid Small Writes on PM

- Minimize metadata modifications on PM with guaranteed crash consistency
  - Buffer metadata on DRAM
    - Allocator metadata
    - Object mapping metadata
      - Object lookup table



Avoid Small Writes on PM

- Minimize metadata modifications on PM with guaranteed crash consistency
  - Buffer metadata on DRAM
  - Annotation
    - Add transaction ID into the transaction state variable
    - Add object metadata (e.g, Object ID, size, and transaction ID) into the object header



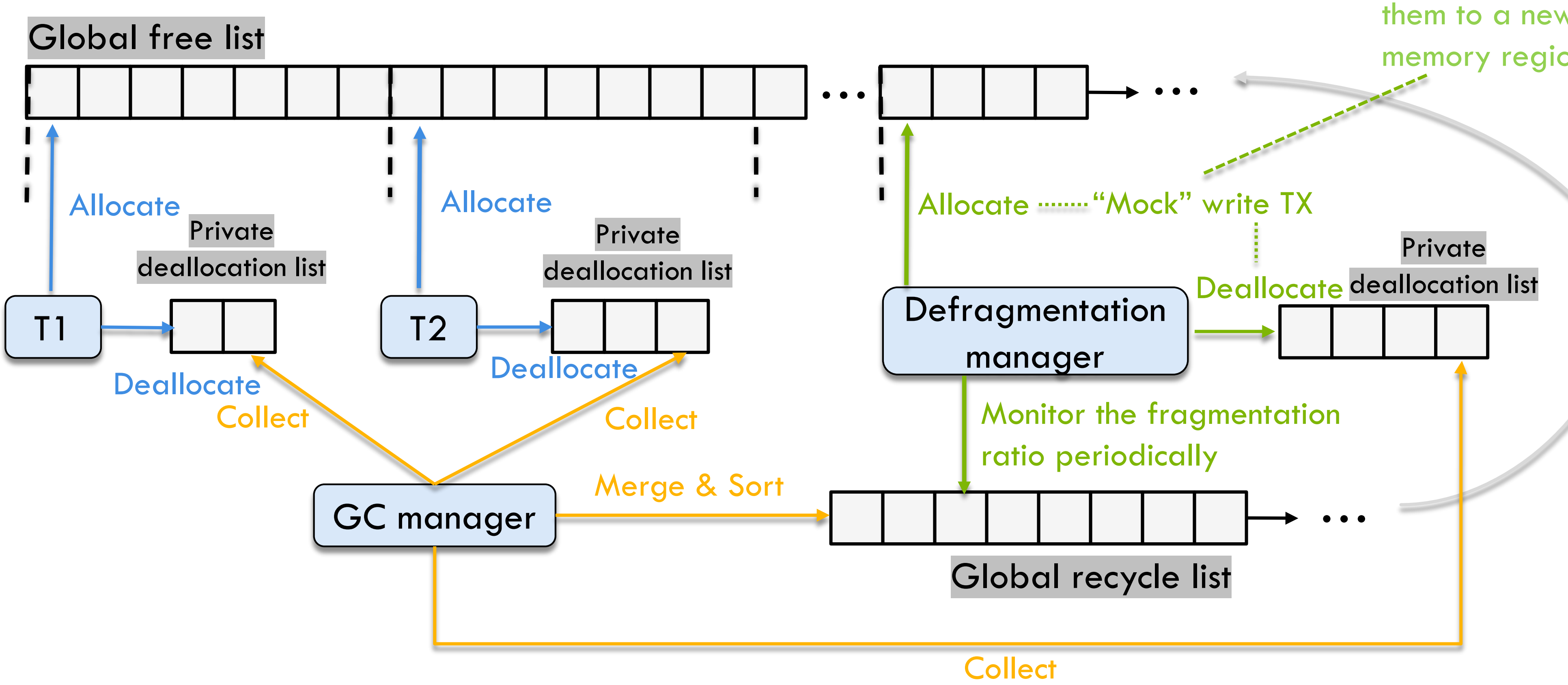
Persist together

Encourage Coalescable Writes on PM

- Consecutive allocation requests get contiguous memory blocks but minimize memory fragmentation
  - Contiguous memory allocation
    - Use a regular data path for large allocations and reclamations
    - Use a locality-aware data path for small allocations and reclamations to encourage sequential writes in transactions
      - A single free list
      - Global recycling
  - Online memory defragmentation
    - Examines memory usage by regions and reduces fragmentation on PM during the runtime

Encourage Coalescable Writes on PM

- Locality-aware data path & online memory defragmentation



Aggregate persistent objects in underutilized regions and migrates them to a newly allocated memory region

Recovery Management

- Step 1: detect uncommitted transactions
  - Check the state of each transaction state variable on PM

- Step 2: rebuild object lookup table
  - Scan persistent object pool on PM to find persistent objects
  - Insert the location information (i.e., pointers to the object on PM) into the lookup table
    - Discard the object copies in uncommitted transactions (collected from Step 1)
    - Only keep the latest object copy by comparing the transaction ID annotated in the object copies

Other Optimization Techniques

- Scalable object referencing
- Non-blocking read
- Reduce recovery time by incorporating an incremental checkpoint

Please find more details in our paper!

Evaluation Setup

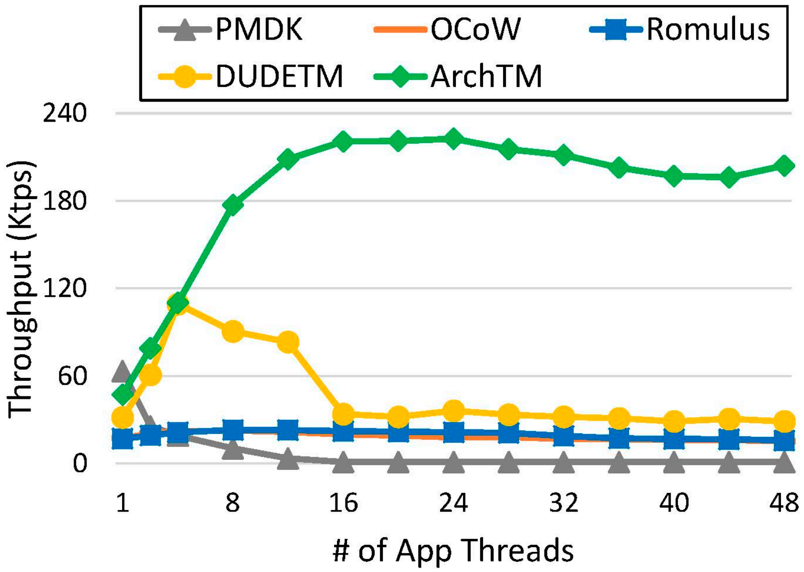
- Real PM platform (Intel Optane DC PMM)
  - 2nd Gen Intel Xeon Scabble processor (24 cores on each socket)
  - 192 GB DRAM and 1.5 TB PM

- Run TPC-C and TATP against PMEMKV (from Intel)

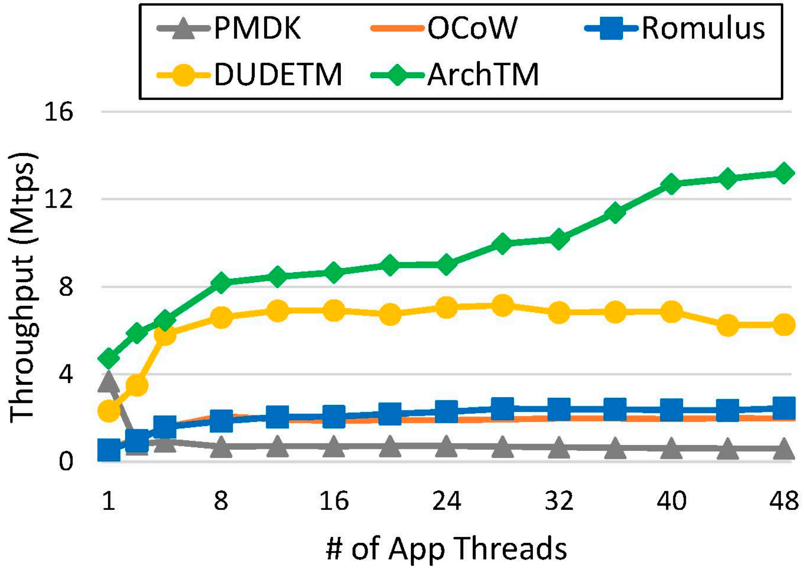
- Comparison: PMDK [Intel], Romulus [SPAA'18], DUDETM [ASPLOS'17] and the Oracle system (copy-on-write-based, OCoW)

Evaluation: TPC-C & TATP

TPC-C 100% update rate



TATP



- On average, ArchTM significantly outperforms DUDETM, Romulus, OCoW and PMDK by 3x, 7x, 8x and 75x, respectively

Please find more evaluation in our paper!

Conclusion

- Pinpoint performance problems in common transaction implementations on real PM hardware
- Highlight the importance of considering PM architecture characteristics for transaction performance
- ArchTM: an architecture-aware PM transaction system
  - Avoid small writes on PM
  - Encourage coalescable writes on PM
  - Outperform the four state-of-the-art PM transaction systems