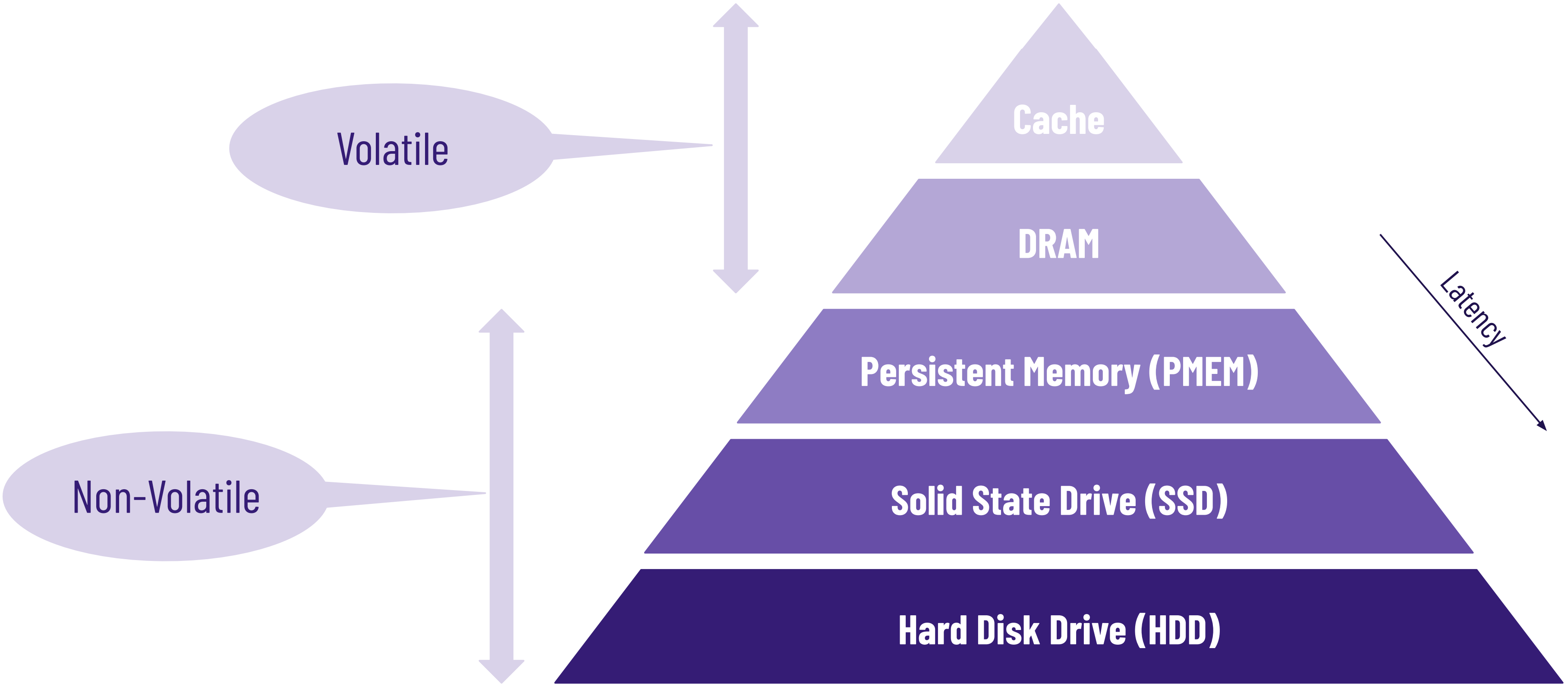
Snapshotting Mechanisms
for
Persistent Memory-Mapped Files

Mohammad Moridi and Wojciech Golab

moridi@uwaterloo.ca, wgolab@uwaterloo.ca

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Persistent Memory in Memory Hierarchy



Background

- **Persistent memory** keeps data in memory after the power is lost.

Background

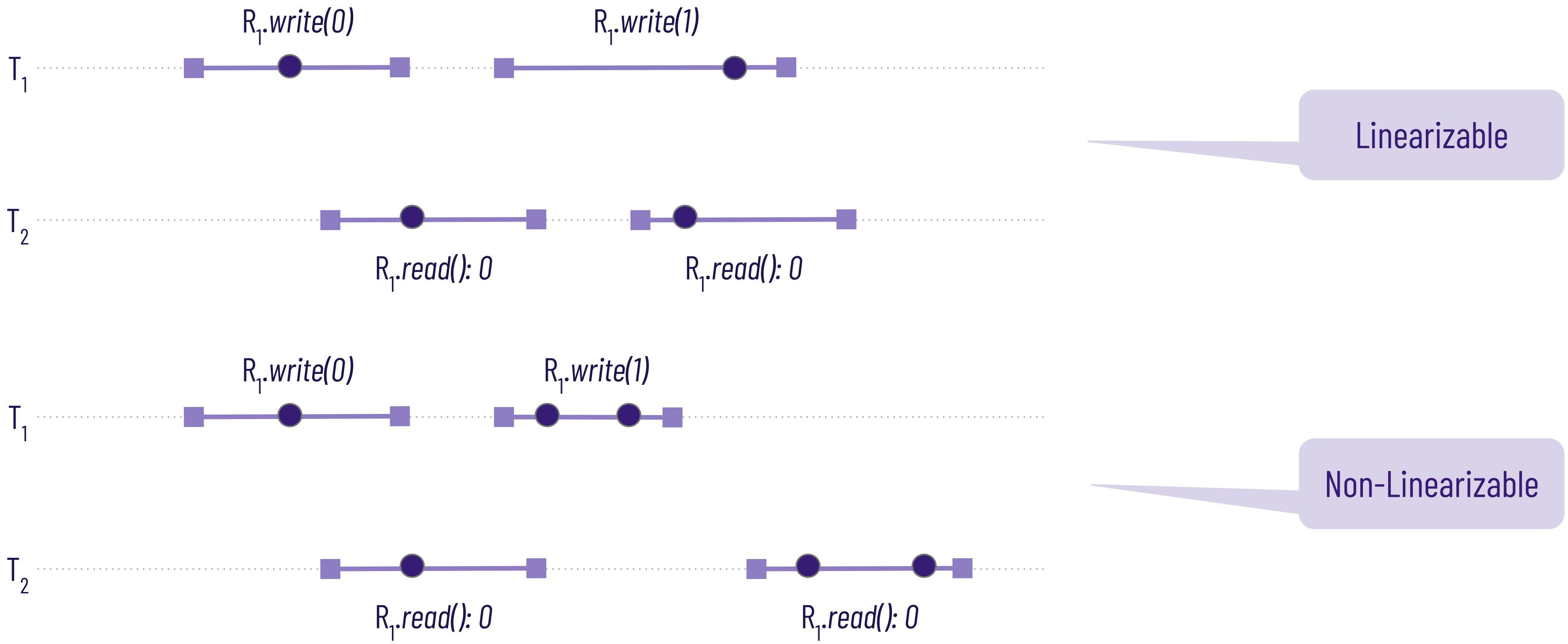
- Correctness Properties
  - **Linearizability**
  - **Durable Linearizability**
  - **Buffered Durable Linearizability**

Model

- **Threads**: The system consists of asynchronous entities, threads, that communicate using shared objects
- **Invocation** Event: It marks the initiation of an operation.
- **Response** Event: It signifies the conclusion of an operation.
- **Crash** Event: It signifies an unexpected system failure.
- **History**: A history models the system's execution as a **finite sequence** of *invocation*, *response*, and *crash* events.

Linearizability Example

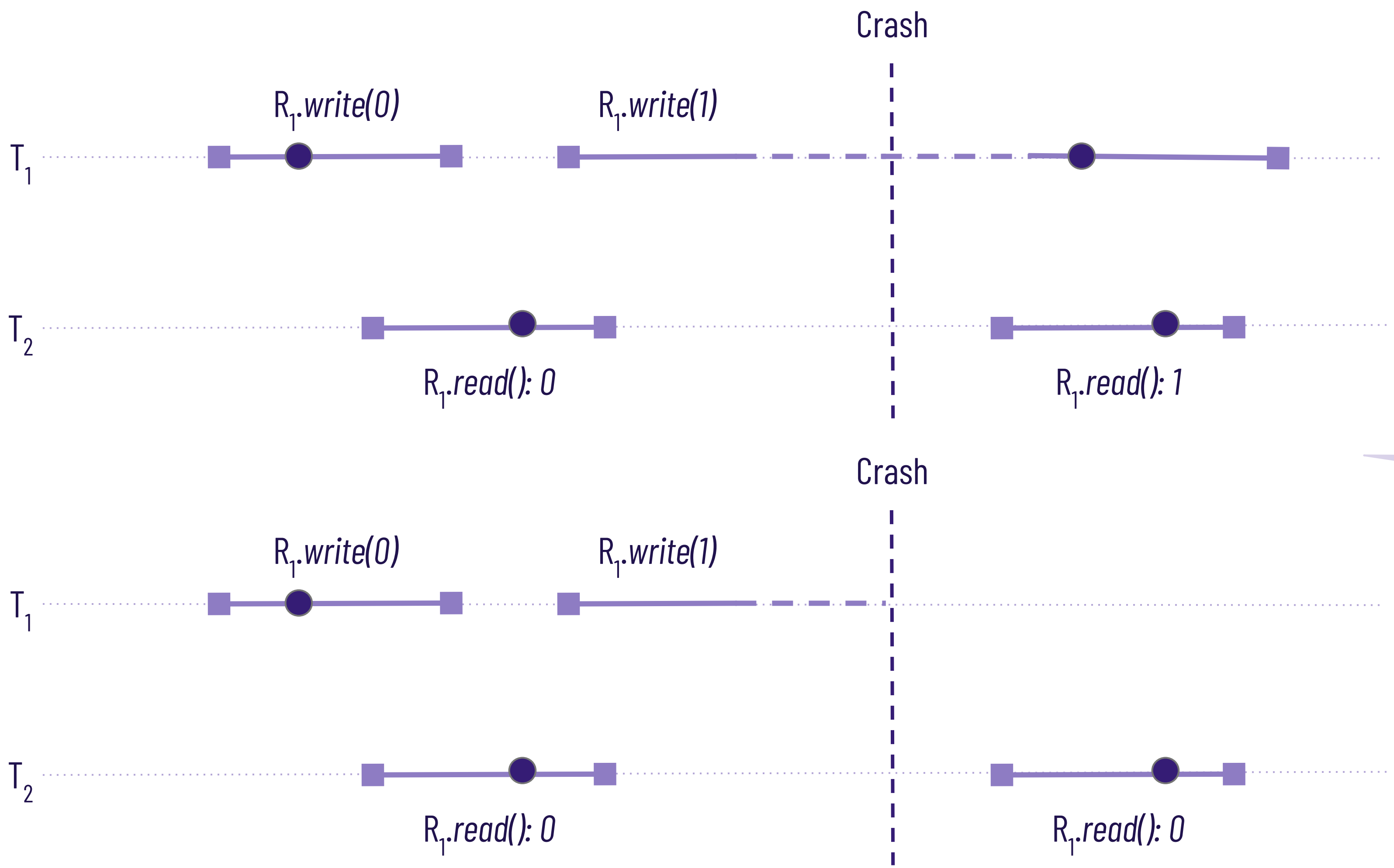
- Linearizability was originally defined by Herlihy and Wing [2]



Durable Linearizability (Definition)

- Durable Linearizability, as defined by Izraelevitz et al. [1]
  - **all threads** fail together
  - on recovery, **new threads** are created (no immediate reuse of thread IDs)
- A history is *durably linearizable* if it is *well-formed* (i.e., the projection onto each thread is sequential) and removing all crash events yields a *linearizable* history.

Durable Linearizability Example (Cont.)

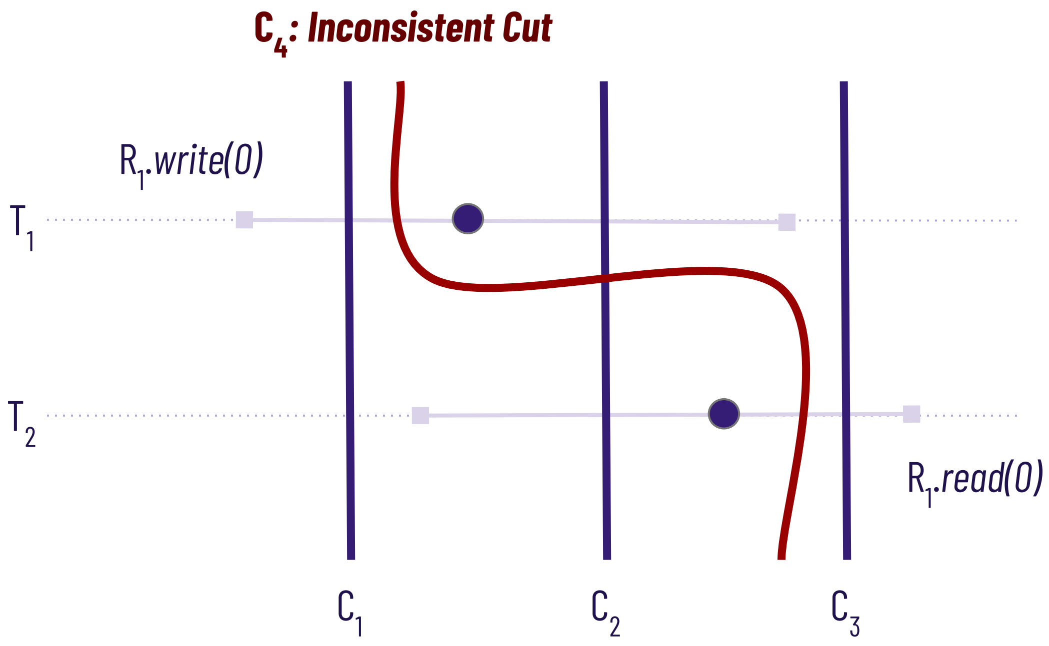


Both are Durably Linearizable

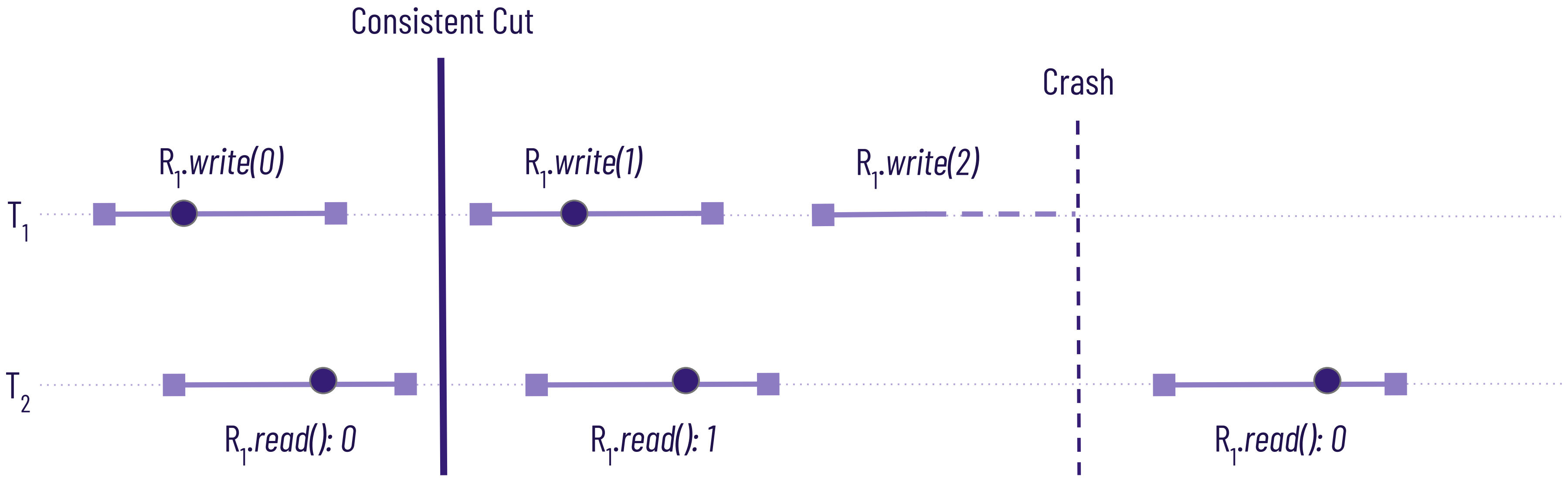
Buffered Durable Linearizability

- Buffered Durable Linearizability, as defined by Izraelevitz et al. [1]
- It is a more **relaxed** concept than durable linearizability.
- It allows for the **removal** of operations that have already been **completed**.
- The modified history should be a "**consistent cut**" of the original history.

Consistent Cut Example



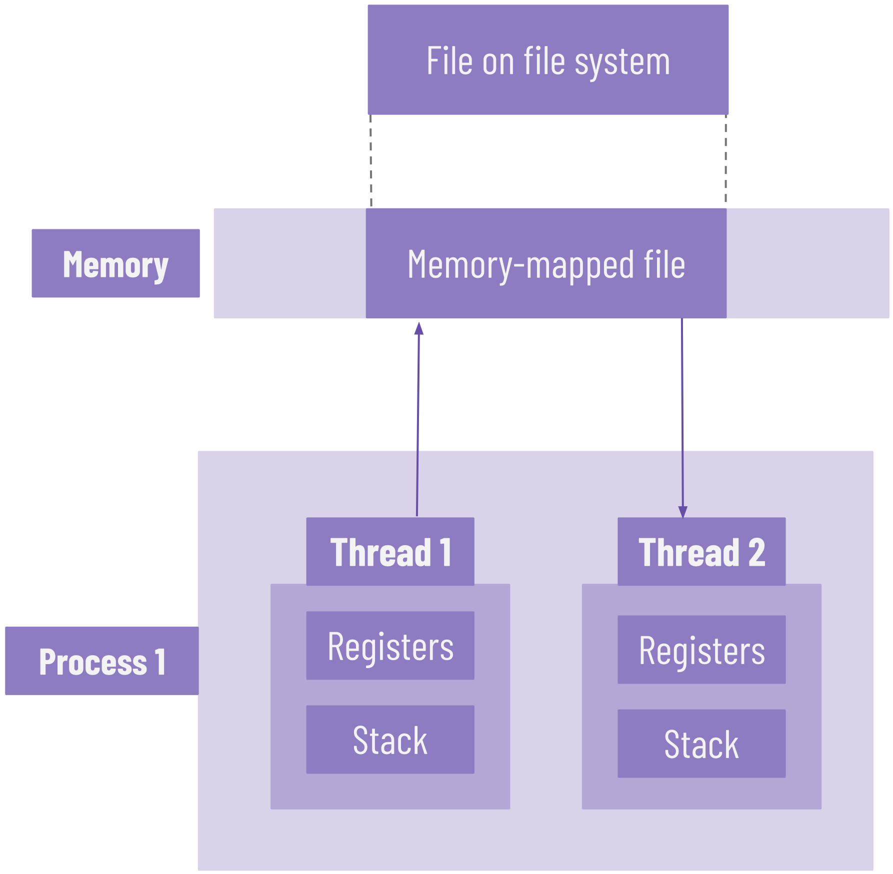
Buffered Durable Linearizability Example



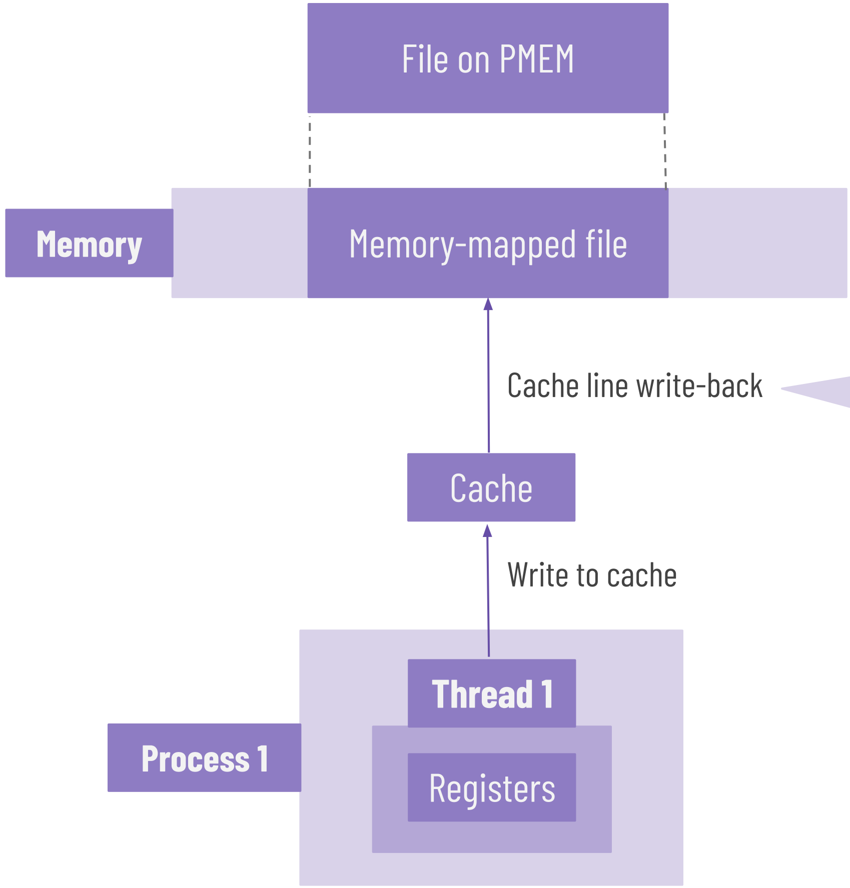
Background

- **Memory-Mapped Files**
- **Montage** is a buffered durable platform.

Memory-Mapped Files



Cache Line Write-Back (CLWB)

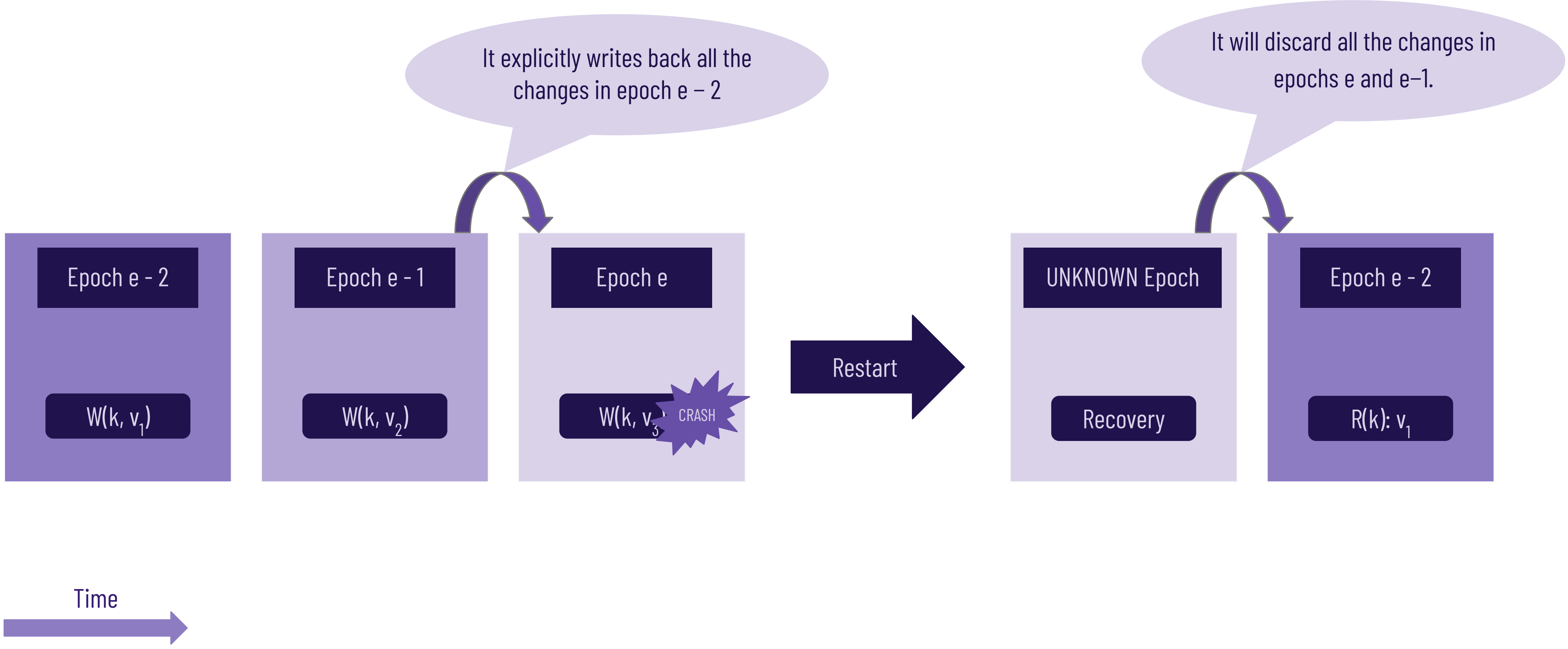


- **Implicit**
  - Automated by the processor's cache controller
- **Explicit**
  - Specific software components issue cache write-backs

Montage

- Montage is a **general-purpose** system for building *buffered durably linearizable* persistent data structures.
- Montage only persists changes to semantically essential *payloads*.
- Montage divides time into “**epochs**” to keep track of operations.
- **Periodic persistence:** changes applied in one epoch persist all together at the end of an epoch.
- On a crash, Montage’s recovery routine reverts state to an epoch boundary.

Periodic Persistence in Montage



Snapshotting

- A **point-in-time copy** of the state of a system.
- Create **recoverable views** of the memory-mapped files in the presence of persistent memory failures.

Snapshotting

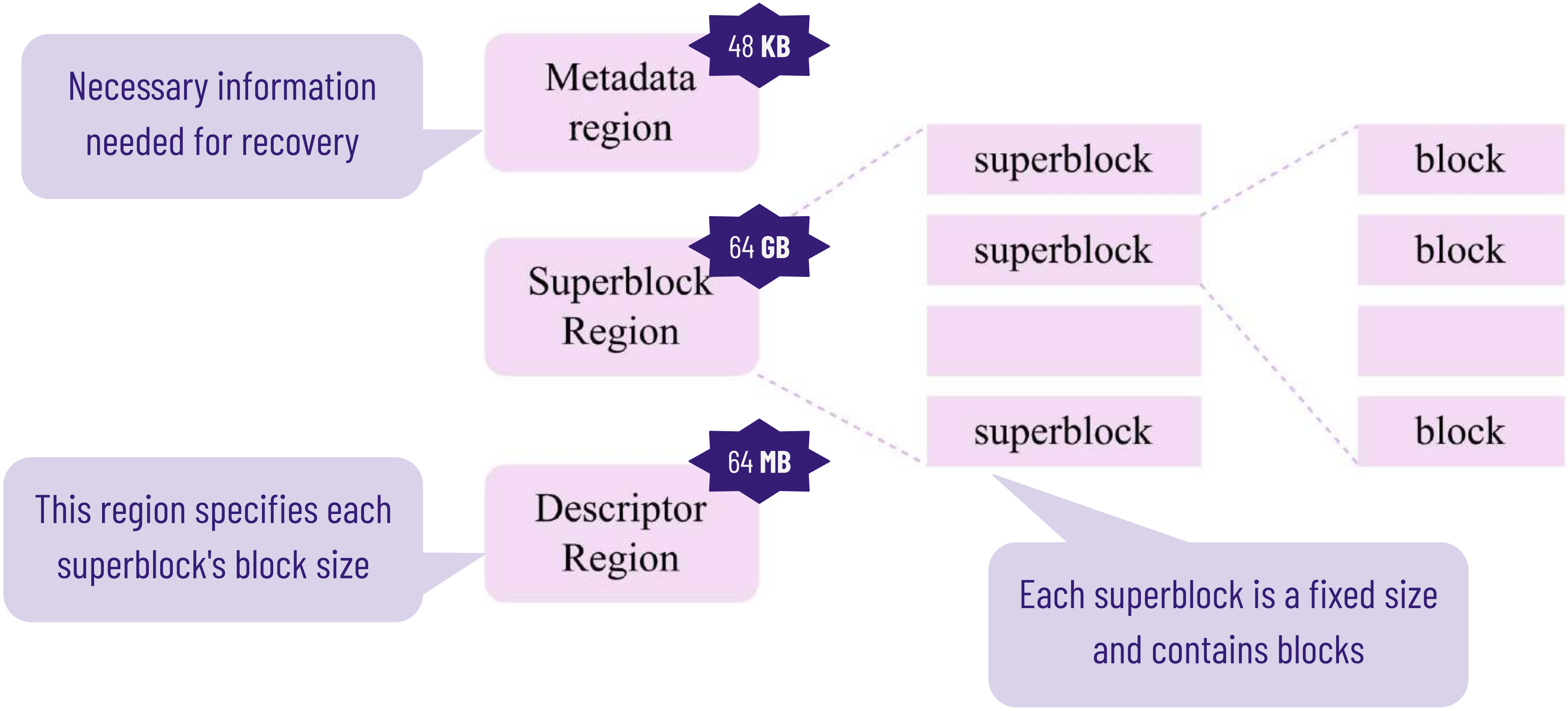
- Montage is vulnerable to the persistent memory failure due to hardware failures.
- Persistent memory challenges are not exclusive to Montage.
- Snapshotting provides a recoverable system state at specific moments.
- The process captures and stores the system's current state as a snapshot.
- After a failure, system can use these snapshots to recover from a persistent memory failure.

Stop-the-World Snapshotting

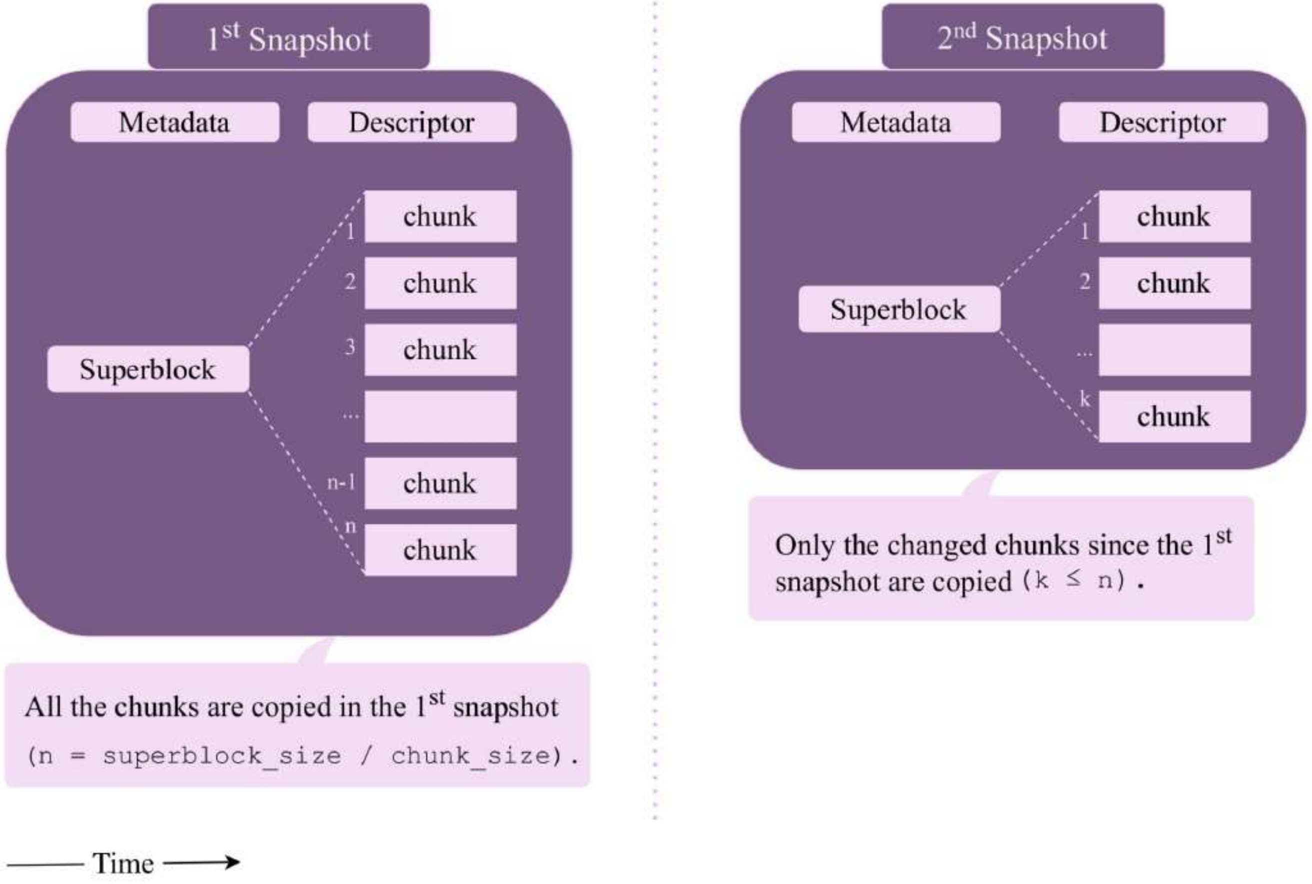
- This method requires pausing all changes to the memory-mapped files while snapshotting is in progress.

Ralloc

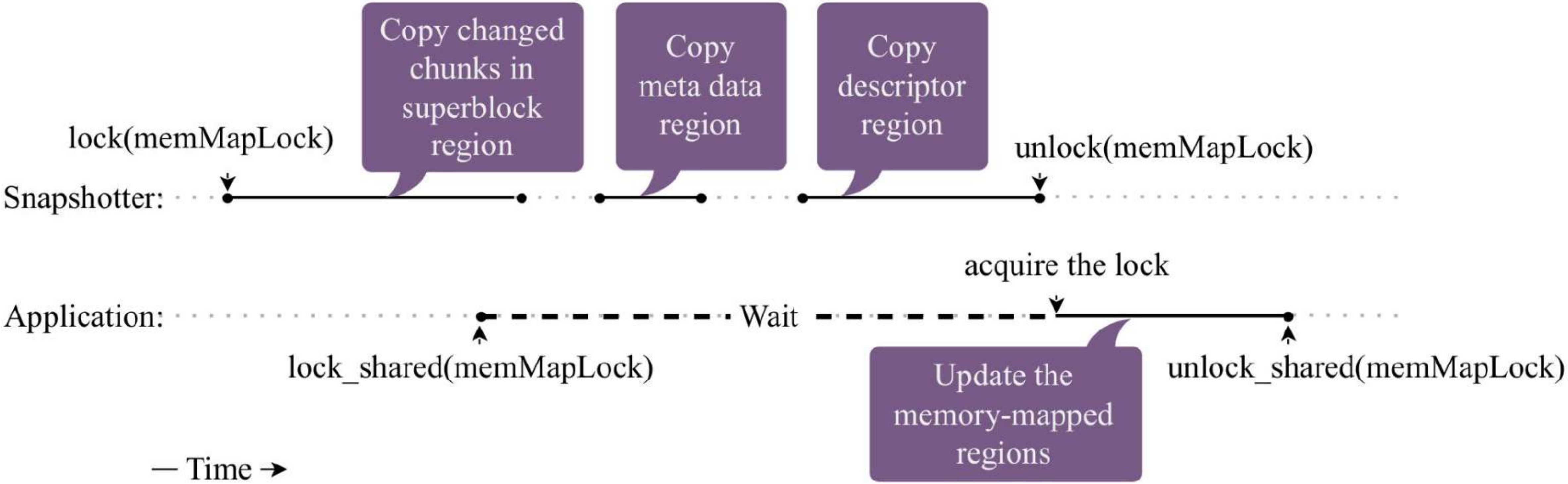
- A persistent memory **allocator** designed for applications that use large amounts of persistent memory like Montage [4].



First Two Snapshots



Stop-the-World Snapshotting



Online Snapshotting

- Online snapshotting is a technique used to create a **point-in-time copy** of data **while** it is still in use.

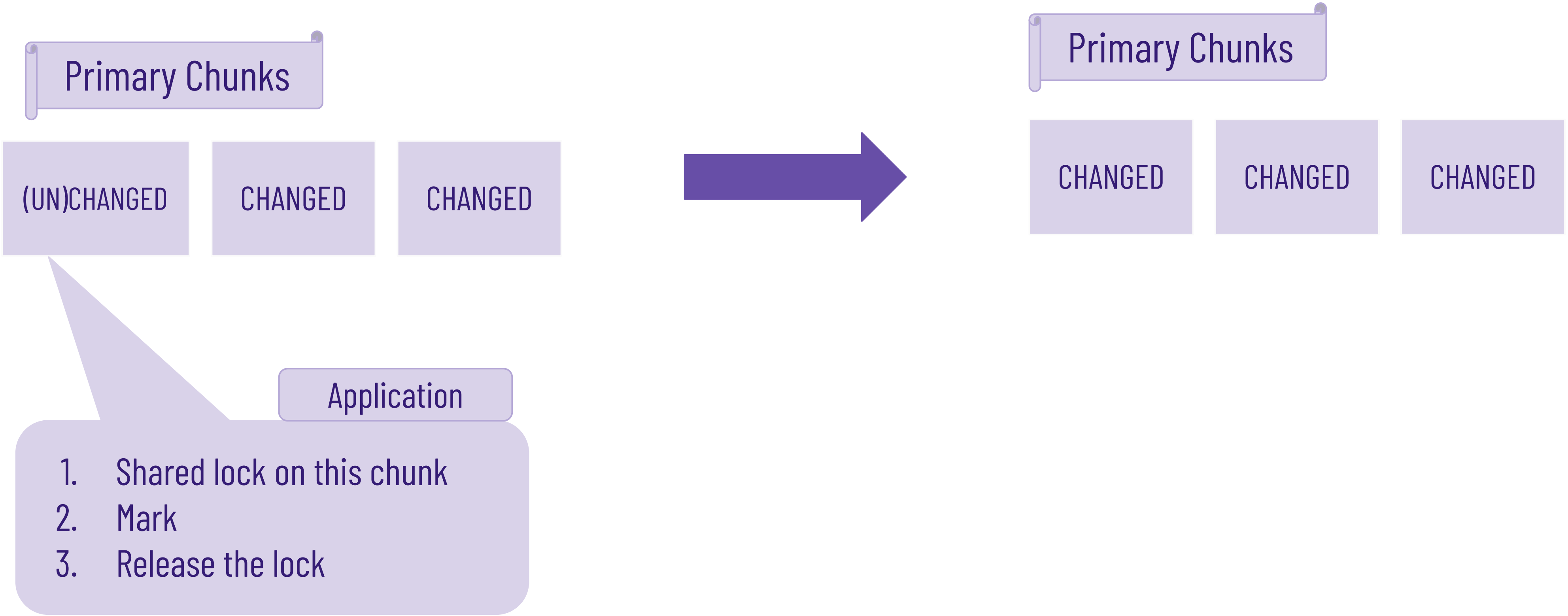
Online Snapshotting on Montage

There are four scenarios for an application thread trying to change a chunk:

1. There is **no snapshotting** is going on
2. There is an **ongoing snapshotting** process:
   - The chunk **has not been copied** yet.
   - The chunk **has already been copied**.
   - The chunk **is being** copied.

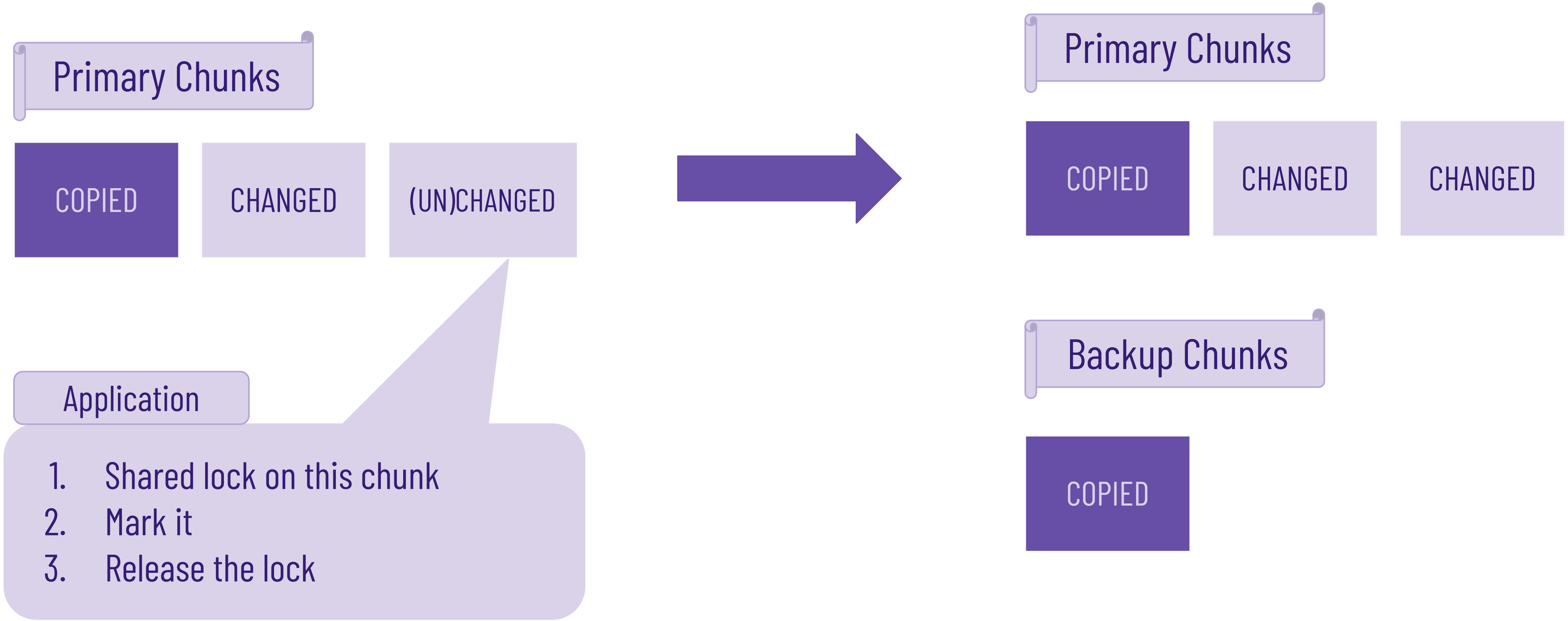
Online Snapshotting

1. **No snapshotting** is going on



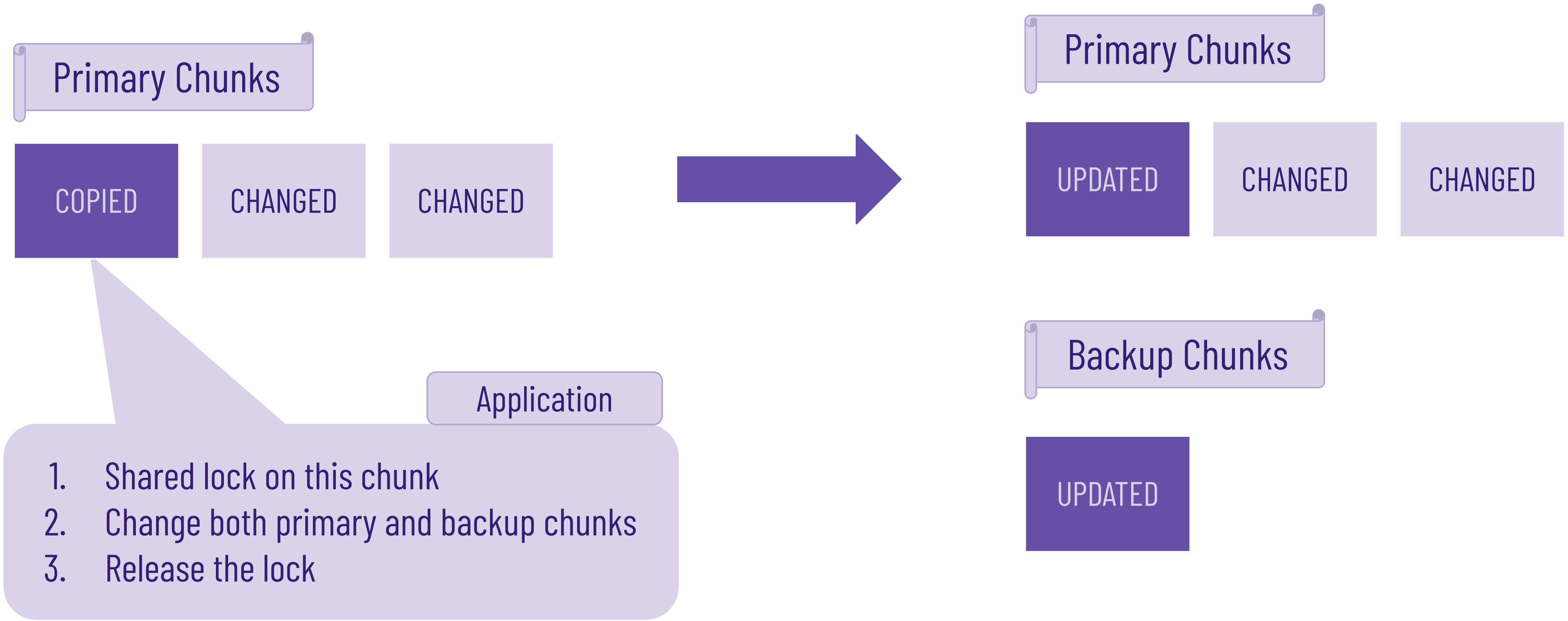
Online Snapshotting

2. The chunk **has not been copied** yet



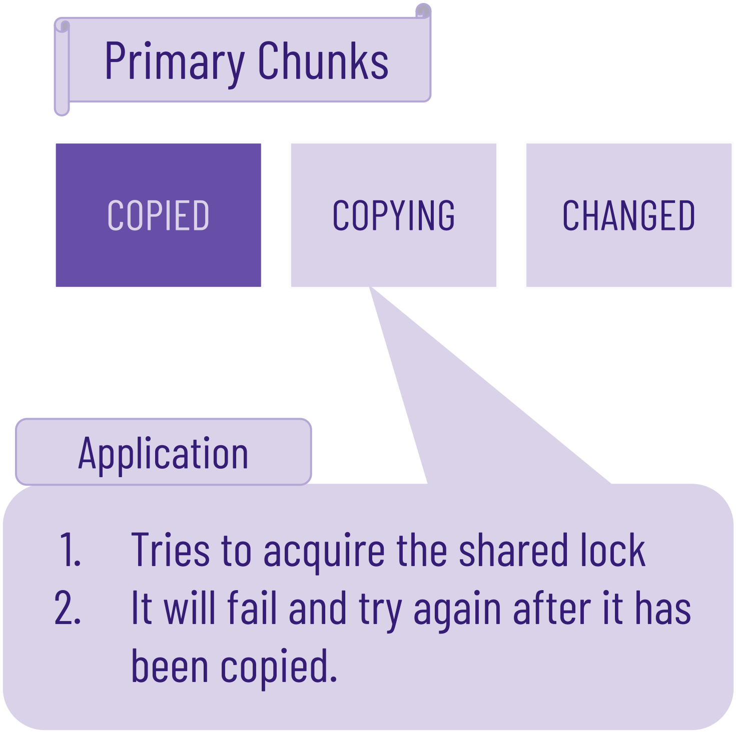
Online Snapshotting

3. The chunk **has already been copied**.



Online Snapshotting

4. The chunk **is being** copied.



UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Double-Checked Locking (DCL)

- Minimize lock overhead by ensuring lock acquisition only when necessary.

Experimental Evaluation

- A detailed examination of snapshotting algorithms is conducted, in different experimental conditions.

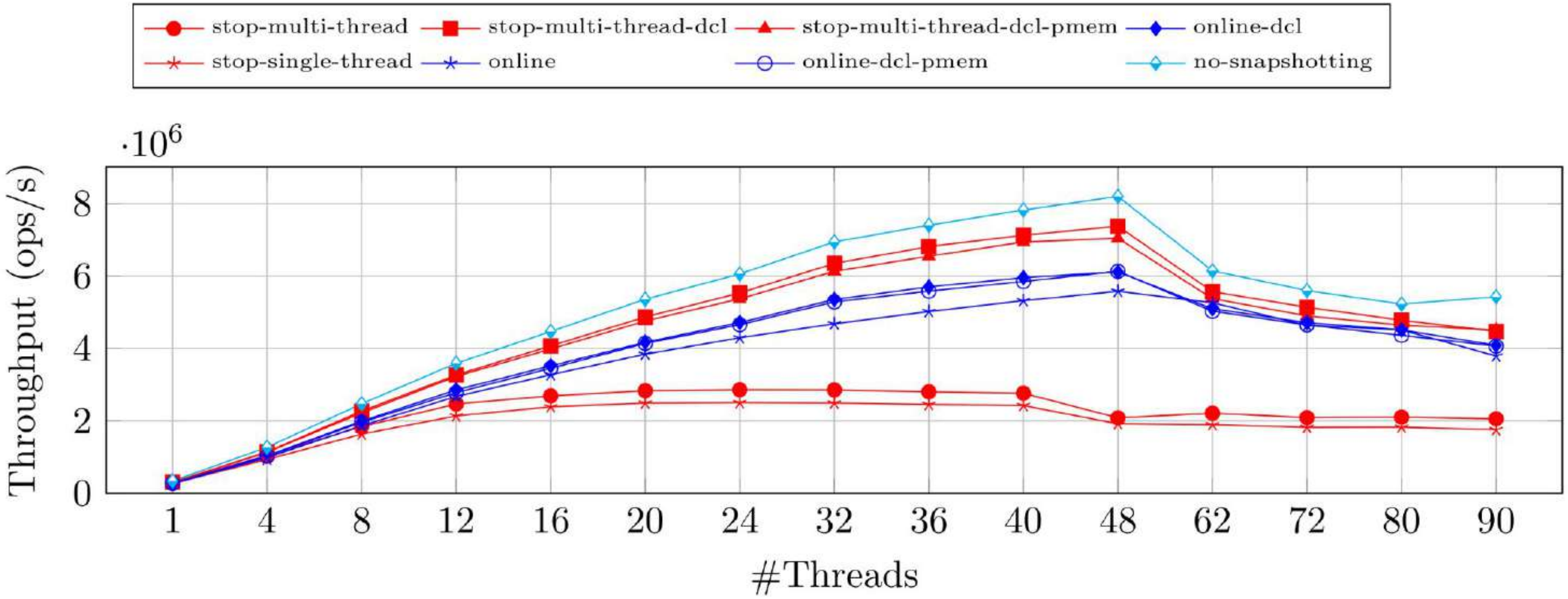
Working Environment

- Testing Environment:
  - Server Specs: Four Intel(R) Xeon(R) Gold 6230 processors with **Optane persistent memory** on Ubuntu 20.04.6 LTS.
- Thread Management:
  - Up to 20 threads: One socket, no hyperthreading.
  - 20-40 threads: Single socket with hyperthreading.
  - 40-80 threads: Two sockets.
  - 80-90 threads: Oversubscription on the first socket.

- Benchmarked Data Structure: Hashmap.
- Experimental Setup:
  - Hashmap: 0.5 million elements into 1 million hash buckets.
- Evaluation Details:
  - Averages over five trials
  - Standard deviation: Less than 2% of the mean

Throughput Analysis

Hashmap with 0% get, 50% insert, and 50% remove



Conclusion

- **Enhanced Fault Tolerance**: We've significantly improved Montage's fault tolerance through rigorous strategies.
- **Buffered-Durable Consistency**: Introduced a robust definition to ensure snapshot correctness.
- **Snapshotting Mechanisms**: Developed both **stop-the-world** and **online** methods.
- **Double-Checked Locking**: Minimized reader lock acquisitions, enhancing system performance.
- **Consistency and Recovery**: Our strategies ensure data consistency, durability, and efficient system recovery.

Future Work

- **Online Snapshotting for Large Allocations**: Address the current limitation in our online snapshotting mechanism for larger allocations.
- **PMEM vs. SSD**: Understand the performance differences between SSD-based and PMEM-based backup media.
- **NUMA-Aware Algorithms**: Optimize snapshotting algorithms to be aware of Non-Uniform Memory Access (NUMA) configurations for performance improvements.
- **Buffered Durably Linearizable Verifier**: Design and implement a verifier to efficiently test program correctness on buffered durably linearizable platforms.

References

1. Izraelevitz, J., Mendes, H., and Scott, M. L. 2016. Linearizability of Persistent Memory Objects Under a Full-System-Crash Failure Model. In DISC 2016: Distributed Computing, 313-327. Springer, Berlin, Heidelberg.
2. Herlihy, M. P., and Wing, J. M. 1990. Linearizability: A correctness condition for concurrent objects. ACM Trans. Prog. Lang. Syst. 12, 3 (July), 463-492.
3. Wen, H., Cai, W., Du, M., Jenkins, L., Valpey, B., and Scott, M. L. A Fast, General System for Buffered Persistent Data Structures.
4. Cai, W., Wen, H., Beadle, H. A., Kjellqvist, C., Hedayati, M., and Scott, M. L. 2020. Understanding and optimizing persistent memory allocation. In Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management, 1-14.
5. Douglas C. Schmidt and Tim Harrison. Double-checked locking. In Robert Martin, Dirk Riehle, and Frank Buschmann, editors, Pattern Languages of Program Design 3. Addison-Wesley, 1998.
6. Amirsaman Memaripour, Joseph Izraelevitz, and Steven Swanson. Pronto: Easy and fast persistence for volatile data structures. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, page 789–806, 2020.
7. T. Cao. Fault Tolerance for Main-Memory Applications in the Cloud. PhD thesis, Cornell Univ., Ithaca, NY, USA, 2013.
8. T. Cao, M.A.V. Salles, B. Sowell, Y. Yue, A.J. Demers, J. Gehrke, and W.M. White. Fast checkpoint recovery algorithms for frequently consistent applications. In Proc. ACM SIGMOD Int. Conf. Manage. Data, pages 265–276, 2011.

UNIVERSITY OF WATERLOO

FACULTY OF ENGINEERING

Mohammad Moridi and Wojciech Golab

moridi@uwaterloo.ca, wgolab@uwaterloo.ca

Related Work

- Pronto, Developed by Memaripour et al. [6]
  - Pronto uses a volatile memory allocator for snapshot creation and keeps them on PMEM.
  - However, it lacks backup protection against persistent memory hardware failures.
- In-Memory Databases
  - **Copy-on-Update** [7]: This method uses an extra data structure to create a duplicate of the primary dataset and a bit array to log the row update statuses.
  - **Ping-Pong** [8]: This technique involves two versions of the data; one is used for updates and the other for noting the progressive snapshot.

Double-Checked Locking

- **Purpose**: Minimize lock overhead by ensuring lock acquisition only when necessary.
- Application in Snapshotting:
    - Snapshotting is infrequent compared to the duration of an epoch.
    - Improves efficiency by avoiding redundant reader locks during non-snapshotting periods.

Double-Checked Locking

Function takeSnapshotDCL():

$isSnapshotting \leftarrow true$

takeSnapshot()

$isSnapshotting \leftarrow false$

Function doPersistProtected($addr$):

Input: $addr$: The memory address of the chunk to persist