Durable Transactional Memory Can Scale With TimeStone

R. Madhava Krishnan, Jaeho Kim[*],
Ajit Mathew, Xinwei Fu, Anthony Demeri,
Changwoo Min, Sudarsun Kannan[+]

VIRGINIA TECH

HUAWEI[*]

RUTGERS[+]
UNIVERSITY | NEW BRUNSWICK

Executive Summary

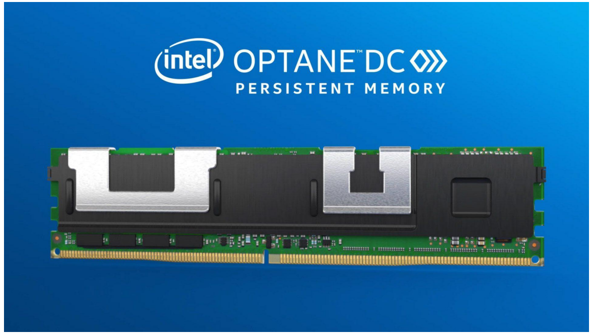
- TimeStone is a *highly scalable* Durable Transaction Memory (DTM)
  - Goals: High scalability, performance and low write amplification
  - Technique: Hybrid DRAM-NVMM logging and MVCC
- A novel Hybrid DRAM-NVMM logging approach for
  - High performance and low write amplification
- TimeStone adopts Multi-Version Concurrency Control (MVCC) model
  - For high scalability and support multiple isolation levels
- ***Scales upto 112 cores and has write amplification <= 1***

Talk Outline

- **Motivation**
- Overview
- Design
- Evaluation

Non-Volatile Main Memory (NVMM)

- NVMM has arrived!
- Storage like characteristics
  - Data persistence
  - Large capacity
- Memory like performance
  - ~100x faster than SSDs
  - Offers byte-addressability

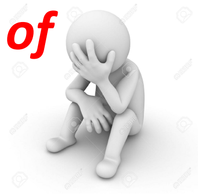


Durable Transactional Memory (DTM)

- DTMs are software framework supporting ACID properties
- DTMs makes NVMM programming easier
- Relieves the burden on NVMM application developers
- *There are some serious problems that needs immediate attention*
  - Poor Scalability
  - High Write Amplification (up to 6x)

Review of Existing DTMs

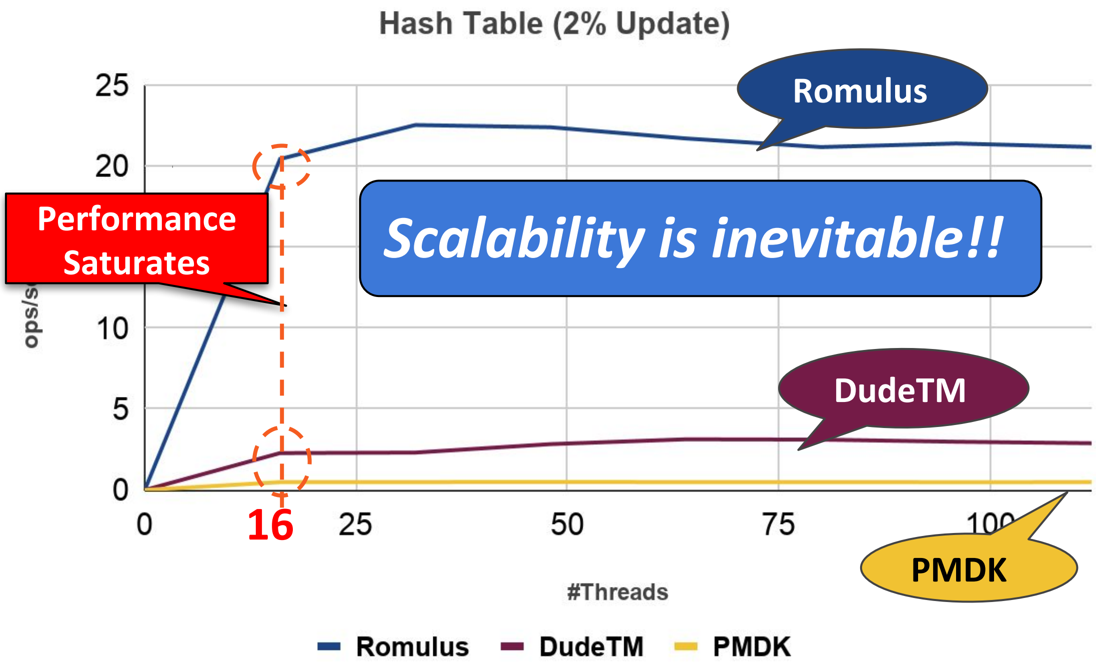
- State-of-art DTMs focuses on reducing the crash consistency cost
  - DudeTM [ASPLOS-17]
  - Romulus [SPAA-18]
- To reduce the crash consistency overhead
  - DudeTM keeps logging operations out of critical path
  - Romulus maintains a backup heap to eliminate logging operations
- ***Existing DTMs incurs high Write Amplification in the course of reducing the crash consistency cost***



Review of Existing DTMs

- *What is Write Amplification (WA)?*
  - Additional bytes written to NVMM for each user requested bytes
- *Why is it a serious problem?*
  - Low write endurance of NVMM
  - Additional writes generates unnecessary traffic at the NVMM
- ***Hence critical path latency increases and performance drops***
- ***None of the DTMs considers Many-core Scalability***

Existing DTMs Are Not Scalable



Performance Saturates

Scalability is inevitable!!

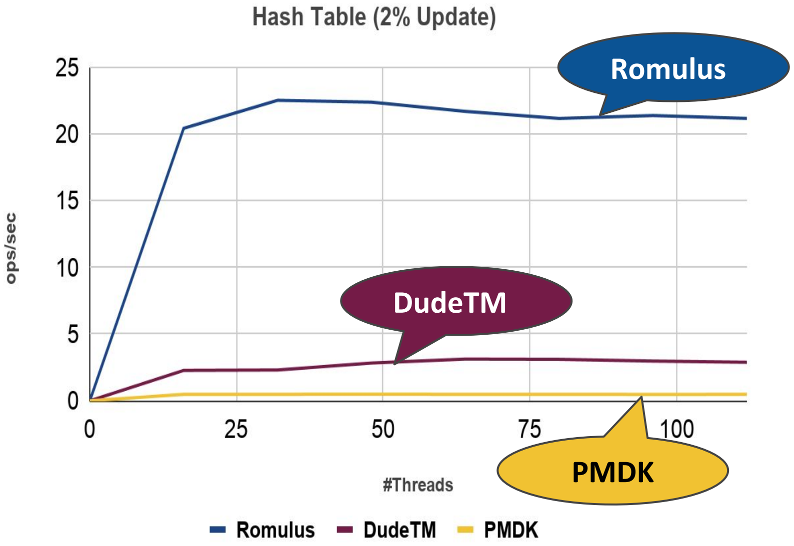
Poor Scalability

None of the DTMs scale beyond 16 cores!!!

The Reasons for Poor Scalability

1. Low RW Parallelism

- ➢ Poor scalability of the underlying STM
  - ○ eg) **DudeTM[ASPLOS-17]**
- ➢ Supports only single Writer
  - ○ eg) **Romulus[SPAA-18],**
  - ○ **PMDK[Intel]**



The Reasons for Poor Scalability

2. High Write Amplification

- Additional bytes written to NVMM
- Crash Consistency Overhead
- Metadata Overhead
- **High WA in the critical path**
  - ***Impacts the system throughput***

DTM Systems	Write Amplification(WA)
Libpmemobj	**70x**
Romulus	**2x**
DudeTM	**4-6x**
KaminoTx	**2x**
Mnemosyne	**4-7x**

So What Do We Need Now?

- A scalable and high performance DTM

Our Solution:

TimeStone

Talk Outline

- Motivation
- **Overview**
- Design
- Evaluation

Two Main Goals of TimeStone

1) Achieve High Scalability and Performance

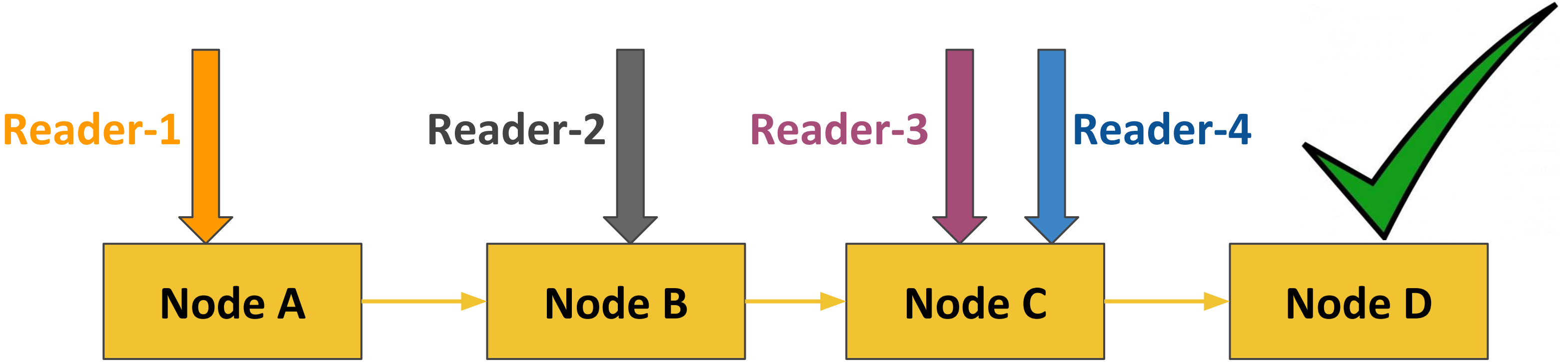
2) Reduce Write Amplification significantly

Goal 1 - To Achieve High Scalability

- TimeStone adopts Multi-Version Concurrency Control (MVCC)
- Supports non-blocking reads and concurrent disjoint writes
- MVCC provides better RW parallelism
- Let’s illustrate how MVCC works!

Illustration - MVCC Programming Model

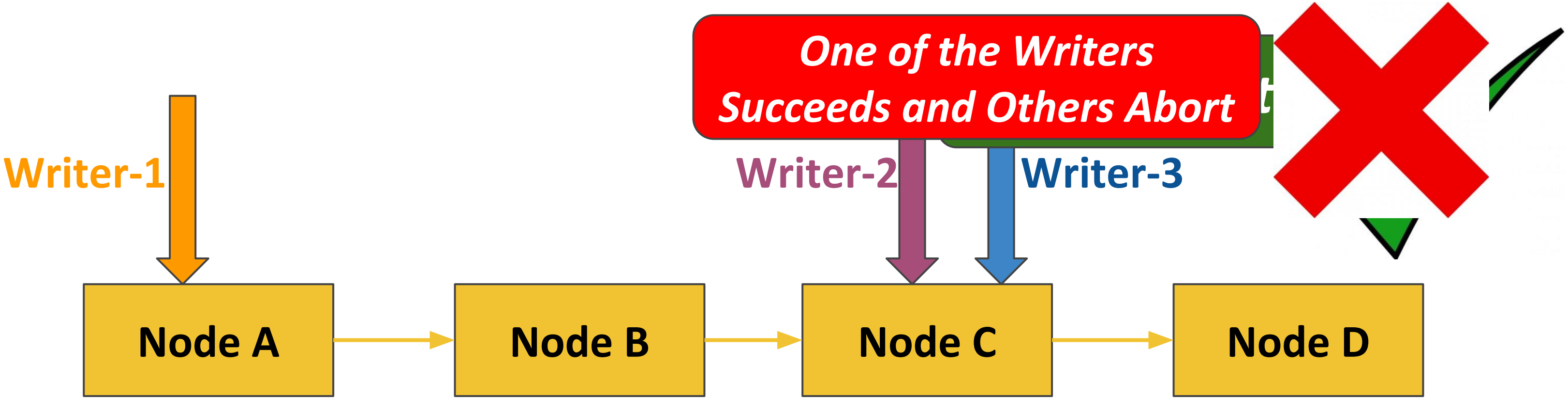
CASE 1: Concurrent Readers



Timestone Supports Non-Blocking Reads

Illustration - MVCC Programming Model

CASE 2: Concurrent Writers



Timestone Supports Disjoint Writes

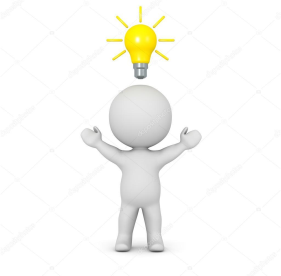
Goal 1 - To Achieve High Scalability

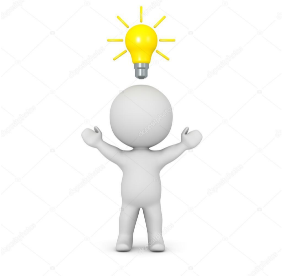
- ➢ MVCC provides better RW Parallelism
- ➢ ***But that's not just enough for better scalability!***

- ➢ Two reasons for poor scalability
  - ○ Low RW Parallelism ⇒ **solved by adopting MVCC** ✓
  - ○ High Write Amplification
- ➢ ***MVCC can incur very high Write Amplification***

Goal 1 - To Achieve High Scalability

- We optimize MVCC for NVMM to achieve better Scalability
  - ***MVCC for better RW parallelism***
  - ***Optimize MVCC for NVMM***

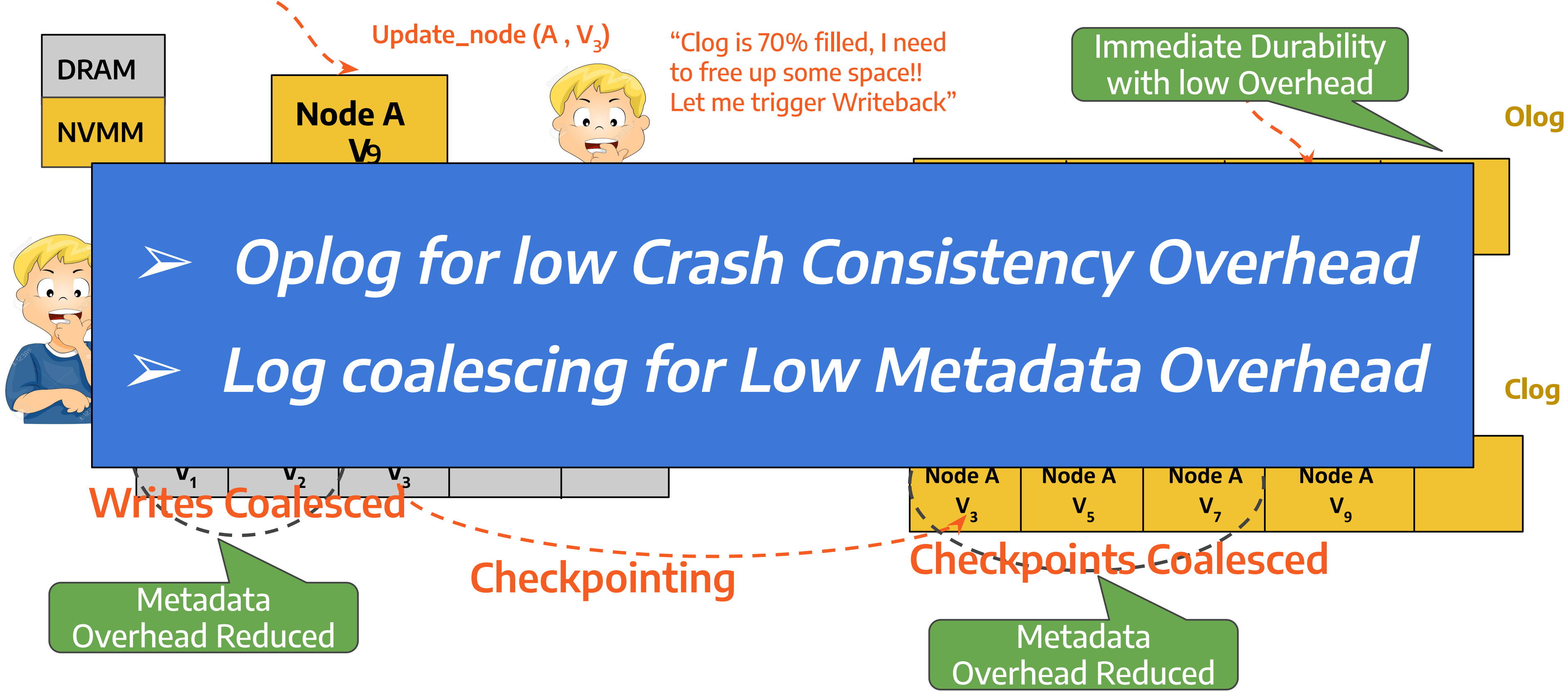




Goal 2 - Low Write Amplification

- TOC logging is a multilayered hybrid DRAM-NVMM logging
  - **T**ransient Version log in DRAM (Tlog)
    - *To leverage faster DRAM for better coalescing*
  - **O**perational log in NVMM (Olog)
    - *To Guarantee Immediate Durability*
  - **C**heckpoint log in NVMM (Clog)
    - *To Guarantee Correct Recovery*
- TOC logging is key to achieve low write amplification

Reducing Write Amplification in TimeStone



- ***Oplog for low Crash Consistency Overhead***
- ***Log coalescing for Low Metadata Overhead***

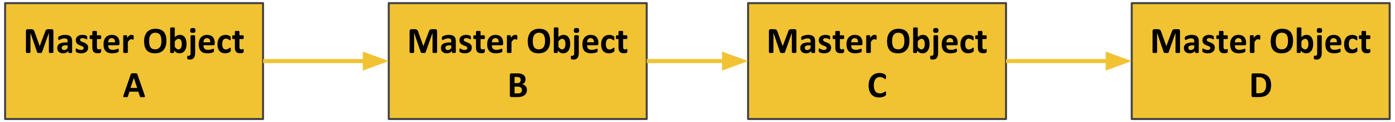
Talk Outline

- Motivation
- Overview
- **Design**
- Evaluation

Object Structure In TimeStone: Master Object

DRAM
NVMM

- TimeStone is an object based DTM
- User defined persistent structure called the master object
- For eg., a simple linked list

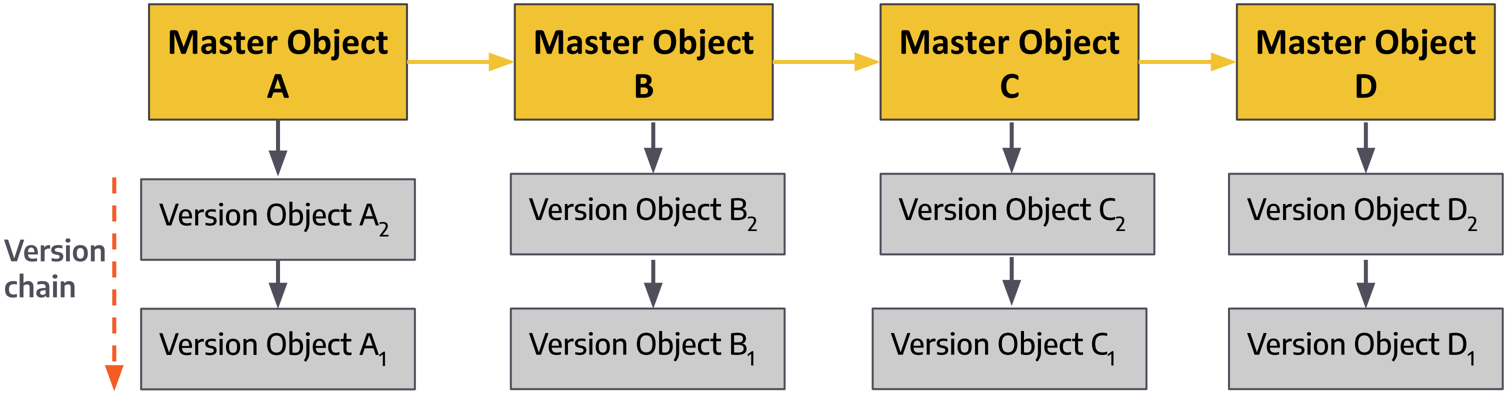


Object Structure in TimeStone: Version Object

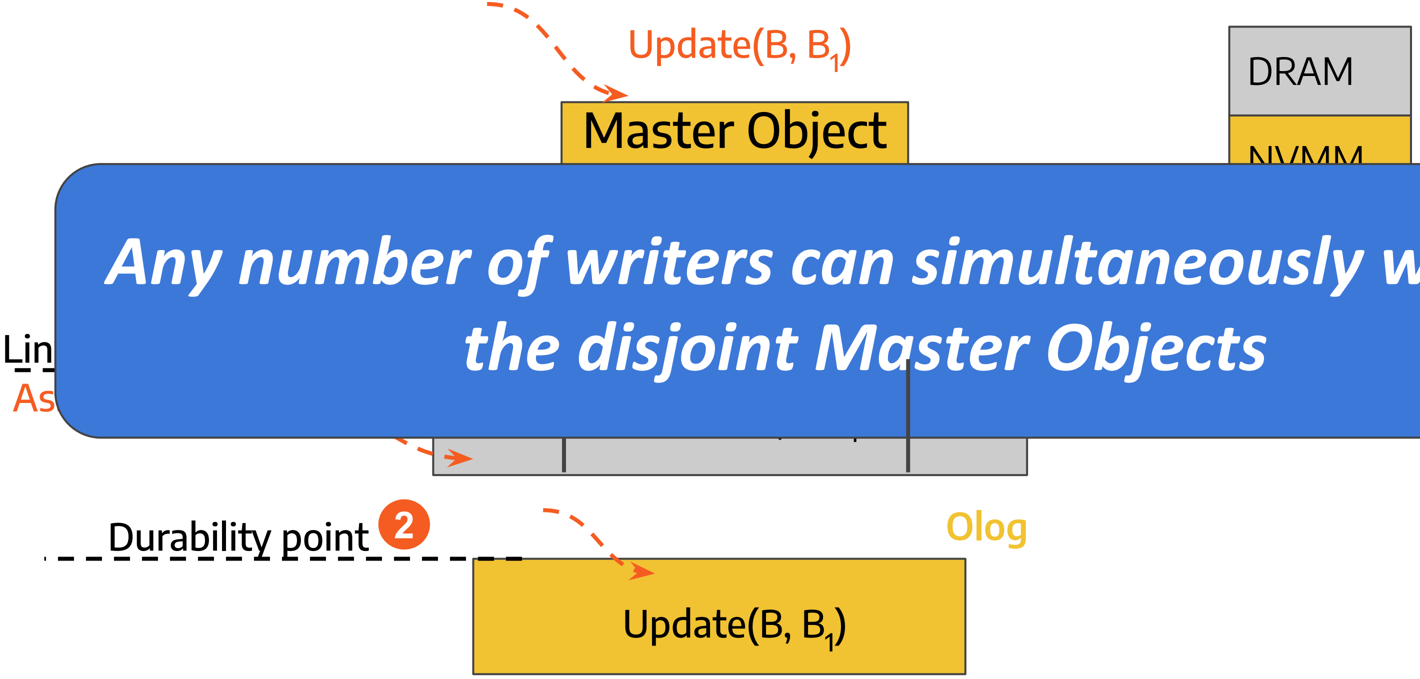
- Different versions of one master object called the Version object

DRAM

NVMM

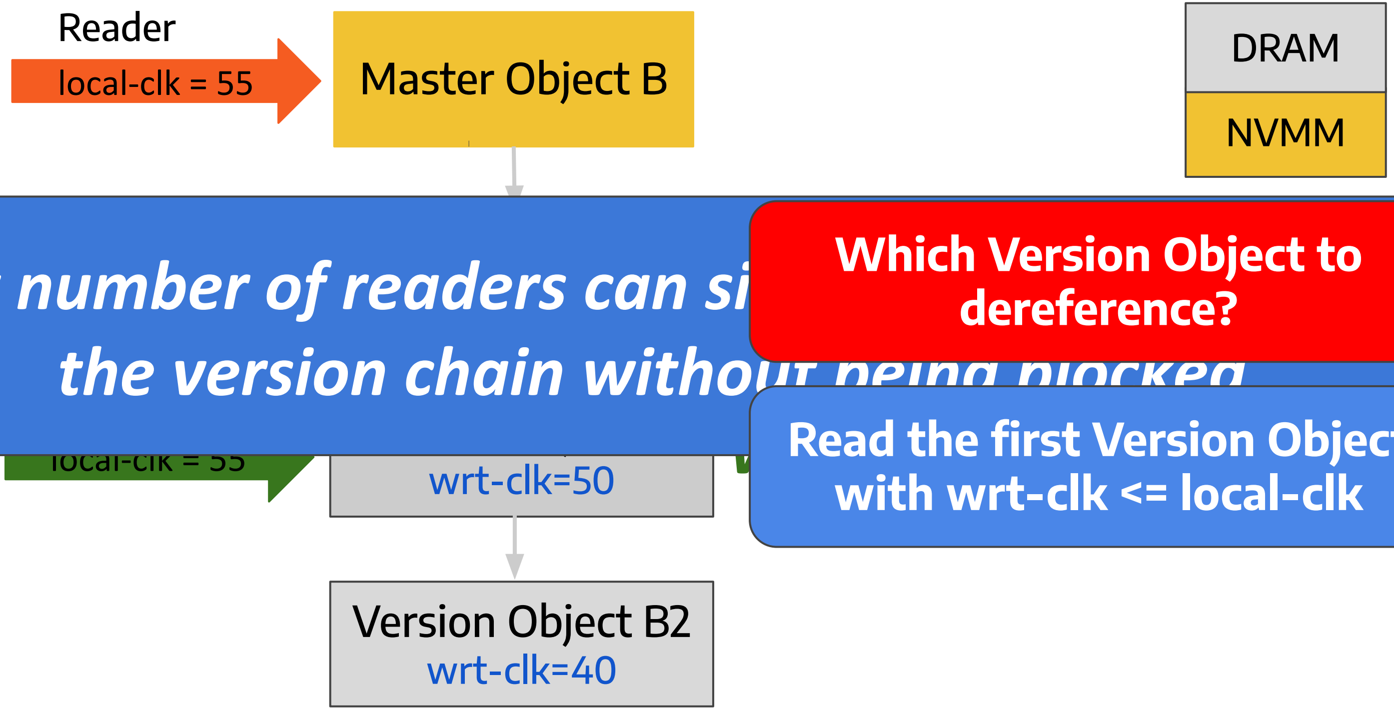


Writes in TimeStone



Any number of writers can simultaneously work on the disjoint Master Objects

Dereferencing - Finding the Right Version



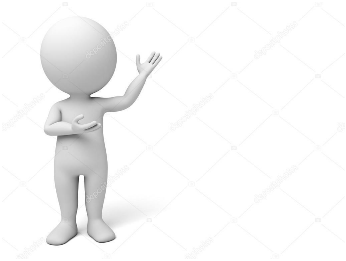
Any number of readers can si
the version chain without being blocked

Which Version Object to dereference?

Read the first Version Object with wrt-clk <= local-clk

Other Interesting Features in TimeStone

- Mixed isolation support
- Asynchronous time based garbage collection
- More details on the design



Talk Outline

- Motivation
- Overview
- Design
- **Evaluation**

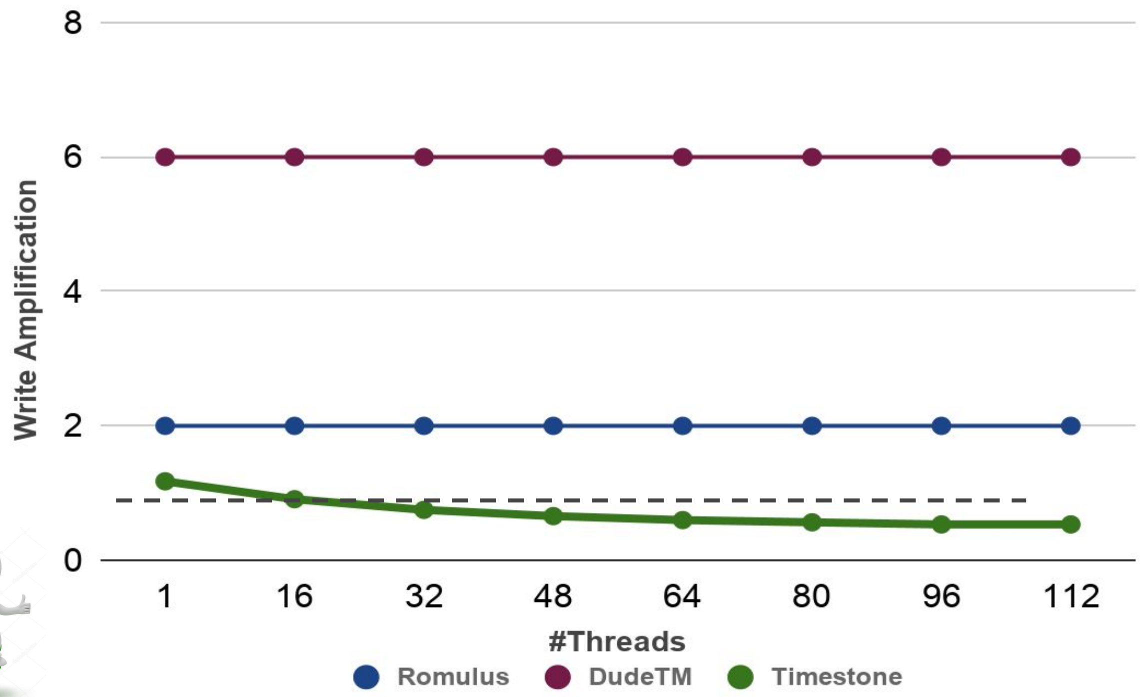
Evaluation Questions

- What is the write amplification in TimeStone?
- Is log coalescing beneficial?
- Does TimeStone scale?
- What is the impact on real-world workload?

Evaluation Settings

- Real NVMM server (Intel DCPMEM)
  - 1TB NVMM and 337GB DRAM
  - 2.5 GHZ 112 core Intel Cascade Lake processor
- Benchmarks
  - Microbenchmarks - List, Hash Table, BST
  - Application Benchmarks - Kyotocabinet and YCSB
- Workloads
  - Different update ratios, access patterns and data set size
- Compared against state-of-art DTM systems

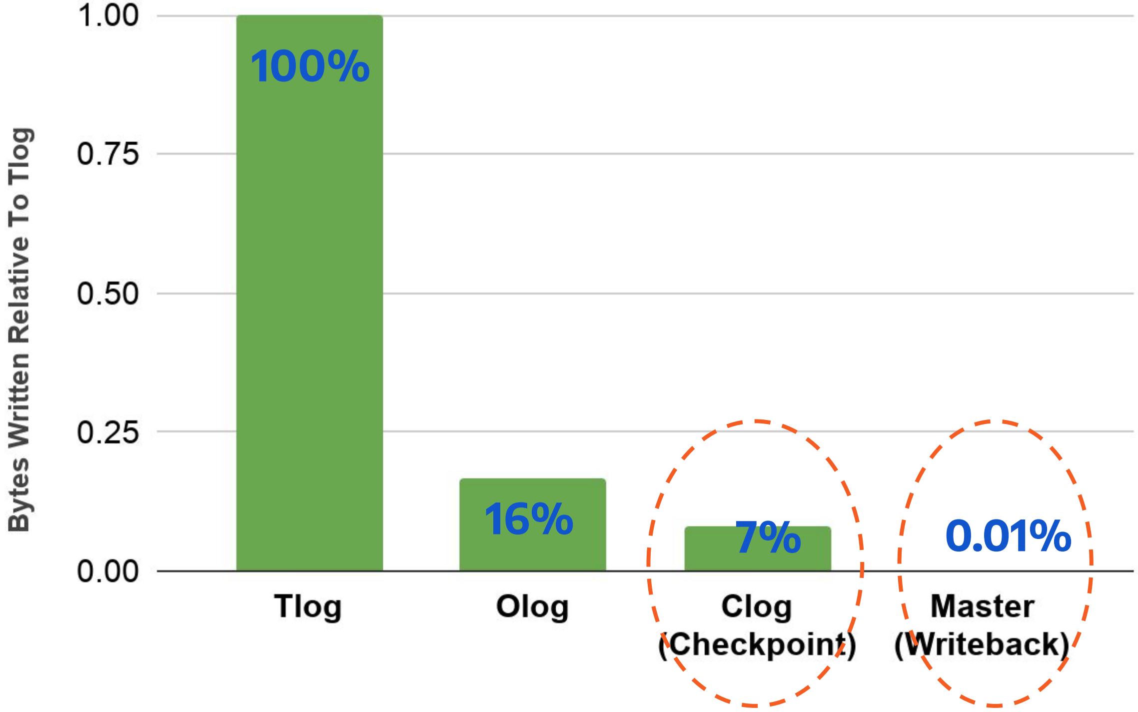
Write Amplification for Write-intensive (80% Update) Hash Table



Write Amplification of PMDK is 70 even for 2% Update case

Write Amplification of TimeStone is always <= 1

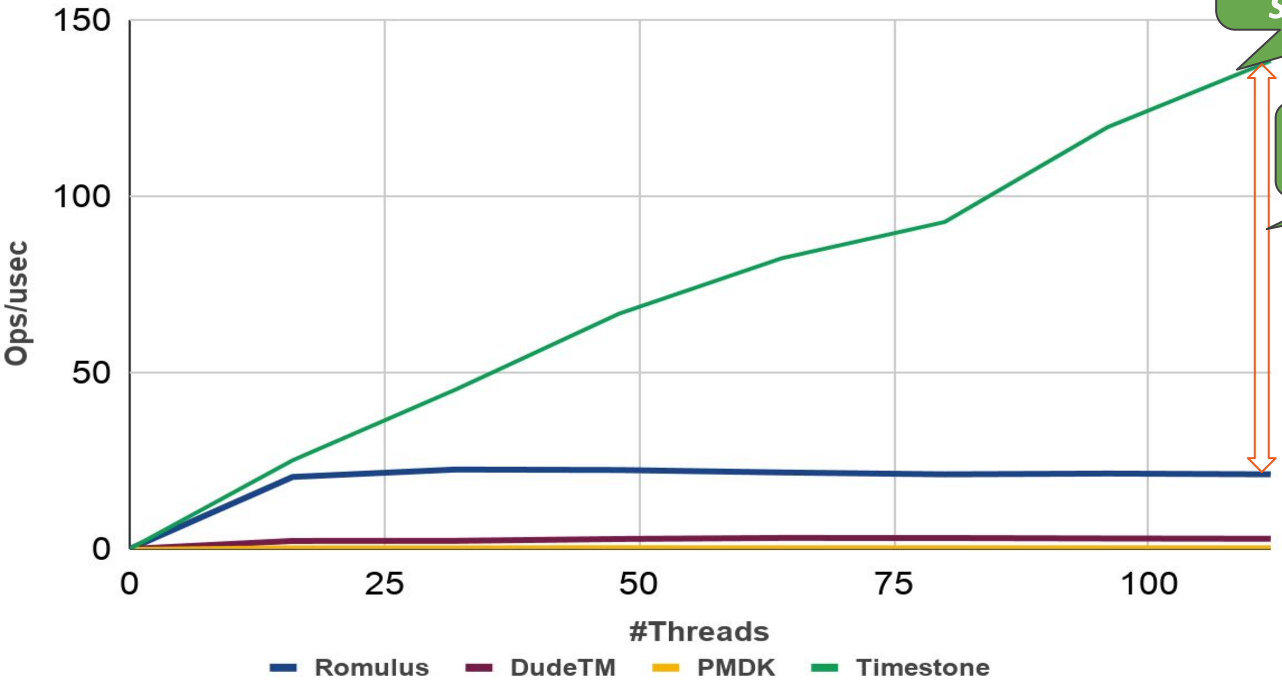
Write Coalescing in TOC Logging



- ➢ Only 7% of writes are checkpointed from Tlog
- ➢ The rest are coalesced in the Tlog

- ➢ Only 0.01% of writes are written back to master
- ➢ The rest are coalesced in the Tlog and Clog

Scalability for Read-Mostly Hash Table (2% Update)



TimeStone scales linearly

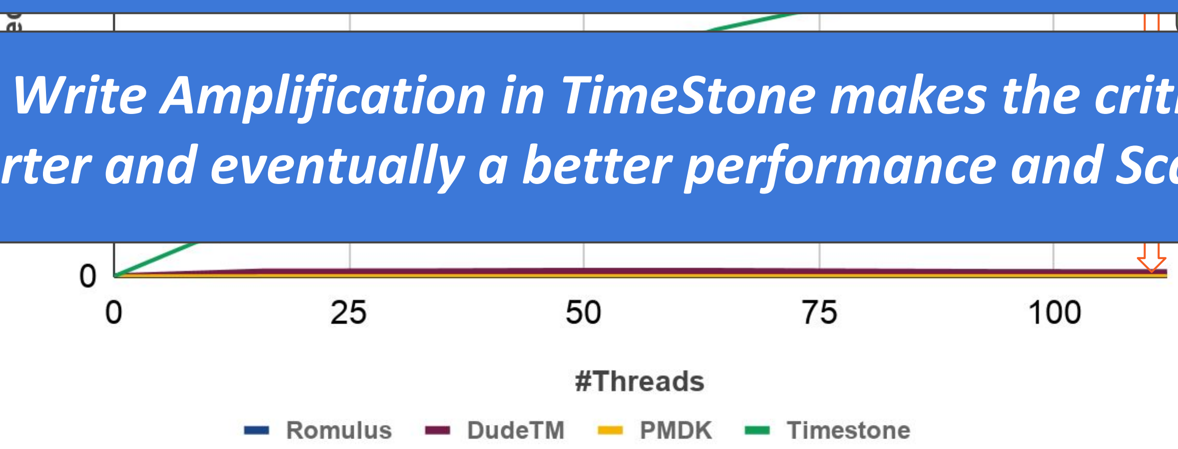
TimeStone is 70x faster than Romulus

Scalability for Write-Intensive Hash Table (80% Update)

TimeStone still

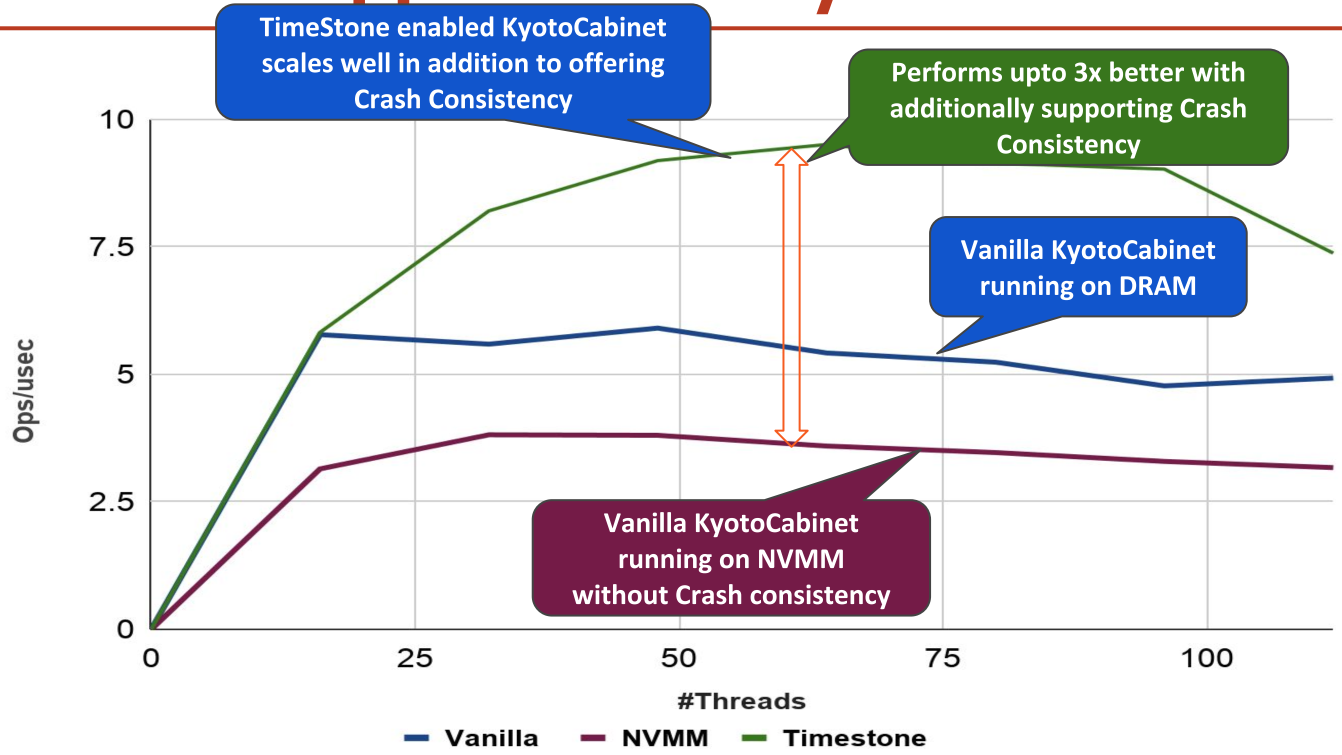
With MVCC TimeStone supports better RW parallelism than existing DTMs and hence it Scales better

rms 100x faster than DudeTM



Low Write Amplification in TimeStone makes the critical path shorter and eventually a better performance and Scalability

Real-World Application - KyotoCabinet



TimeStone enabled KyotoCabinet scales well in addition to offering Crash Consistency

Performs upto 3x better with additionally supporting Crash Consistency

Vanilla KyotoCabinet running on DRAM

Vanilla KyotoCabinet running on NVMM without Crash consistency

Discussion

- Durable Transactional Memory Systems
  - *Romulus[SPAA-18], DudeTM[ASPLOS-17], PMDK, Mnemosyne[ASPLOS-11]*
- Inspired from in-memory databases
  - *Ermia[SIGMOD-16], Cicada[SIGMOD-17]*
- Also non-linearizable synchronization algorithms
  - *RCU[OLS-02], RLU[SOSP-15], MV-RLU[ASPLOS-19]*
- Future work
  - *Provide memory safety and reliability in TimeStone*
  - *Extend TimeStone to support distributed transactions*

Conclusion

- Current DTMs:
  - Do not scale beyond 16 cores
  - High write amplification
- **TimeStone:**
  - Adopts and optimizes *MVCC* for better *multi-core Scalability*
  - Proposes *TOC Logging* to reduce the *Write Amplification*
- ***Scales upto 112 cores***
- ***Has Write Amplification <=1***
- ***Performs Upto 100x better than the state-of-art DTMs***

BACKUP SLIDES

R. Madhava Krishnan

Advisor : Dr. Changwoo Min

Conclusion

- Current DTMs:
  - Do not scale beyond 16 cores
  - High write amplification
- **TimeStone:**
  - Adopts and optimizes *MVCC* for better *multi-core Scalability*
  - Proposes *TOC Logging* to reduce the *Write Amplification*
- ***Scales upto 112 cores***
- ***Has Write Amplification <=1***

Thank You!

Problems In The Existing DTMs

High Storage Overhead

- ➢ DudeTM
  - ○ requires DRAM == NVMM
- ➢ Romulus, KamnioTX
  - ○ Only half of the available NVMM is used
- ➢ Curtails the cost effectiveness of NVMM

DTM Systems	Storage over
Libpmemobj	Min al
Romulus	Very High
DudeTM	Very High
KaminoTx	Very High
Mnemosyne	Minimal

2x the size of NVMM

Minimal Storage Overhead in Timestone

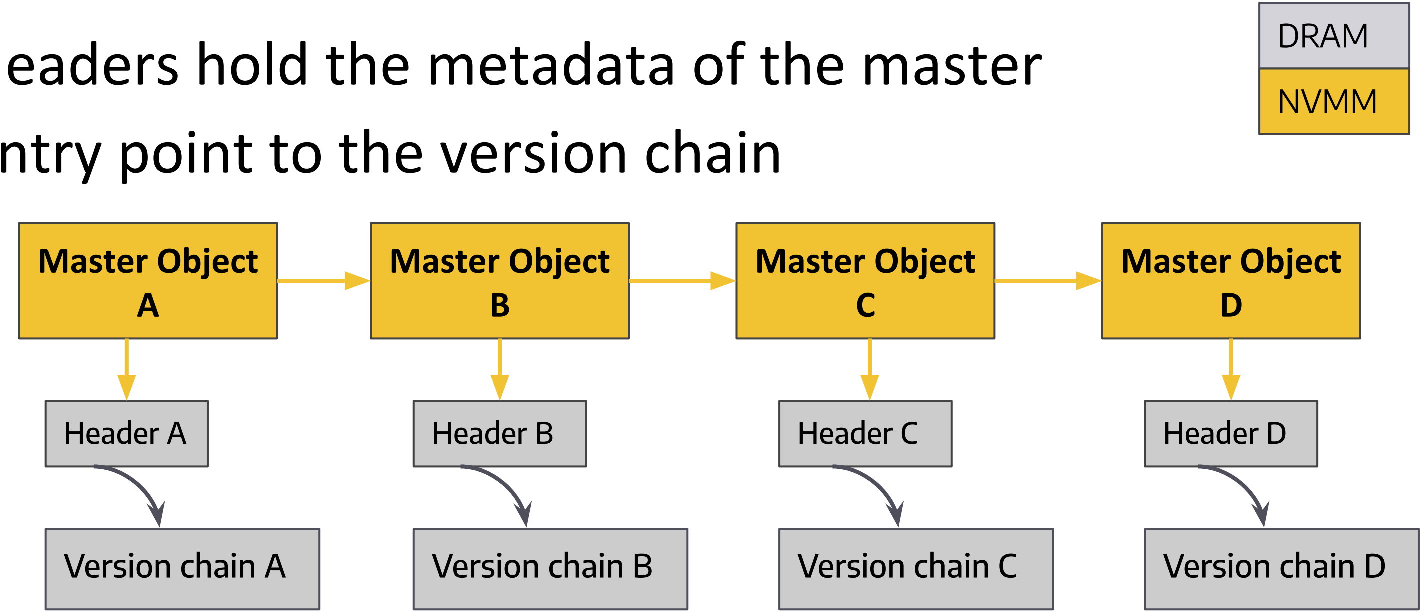
- Additional storage required only for the logs
- All Logs in Timestone are finite (4MB)
- Asynchronous time based garbage collection mechanism
  - Does not become a scalability bottleneck
  - Does not block writers
  - Enables better log write coalescing

Design of Timestone

- Timestone follows the MVCC programming model
- Object organization in Timestone
- How writes are handled in Timestone?
- How reads (object dereferencing) are handled?

Object Structure in Timestone: Control Header

- Headers hold the metadata of the master
- Entry point to the version chain



DRAM

Update_node (A , V_3)

“Clog is 70% filled, I need to free up some space!! Let me trigger Writeback”

Key idea

3

Update_node (A , V_3)

➢

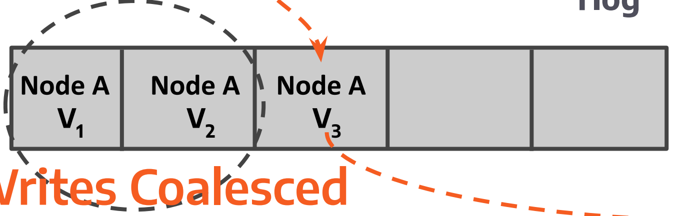
➢

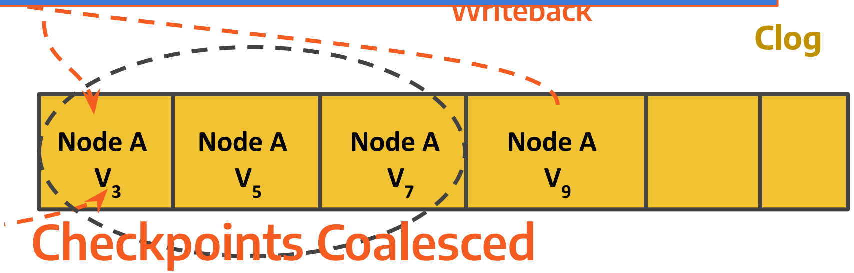
lates

log

Looks good, But what happens if there is a power failure before Tlog checkpoints its updates?

Let me trigger ch





Implementation

- Core library in C
- About 7000 LOC
- An additional C++ wrapper to hide the concurrency control and crash consistency.
- NVMM friendly design pattern
    - Logging writes are one sequential write + p-barrier

Mixed Isolation in Timestone

- Timestone supports different isolation levels on the same instance of the data structure
- By default it supports serializable SI
- Timestone supports stricter isolation levels by having read-set validation at the commit time
- Keeps track of the read set and write set if the transaction runs in a stricter isolation level
- Upon read set validation failure the transaction is aborted and the updates are not visible

How Timestone guarantees ACID?

- **Atomicity**
  - Upon transaction commit, updates are atomically visible
  - Upon abort, the copy does not make it to version chain
- **Consistency**
  - Both the link and data consistency as we make a complete copy of the object
- **Isolation**
  - Reader isolation using time as synchronization primitive
  - Writer isolation using try_lock
- **Durability**
  - Immediately durable after commit using the oplog.

Recovery Design in Timestone

- Tightly Coupled with our logging design
- Completely reclaim and destroy all the logs upon safe termination
- Upon starting Timestone, check if the nvlog heap is consistent
- If not trigger the recovery
- Recovery is essentially a two step process
    - Replay Clog to set the master object in a consistent state
    - Replay Olog to reach to the latest point before the crash occurred

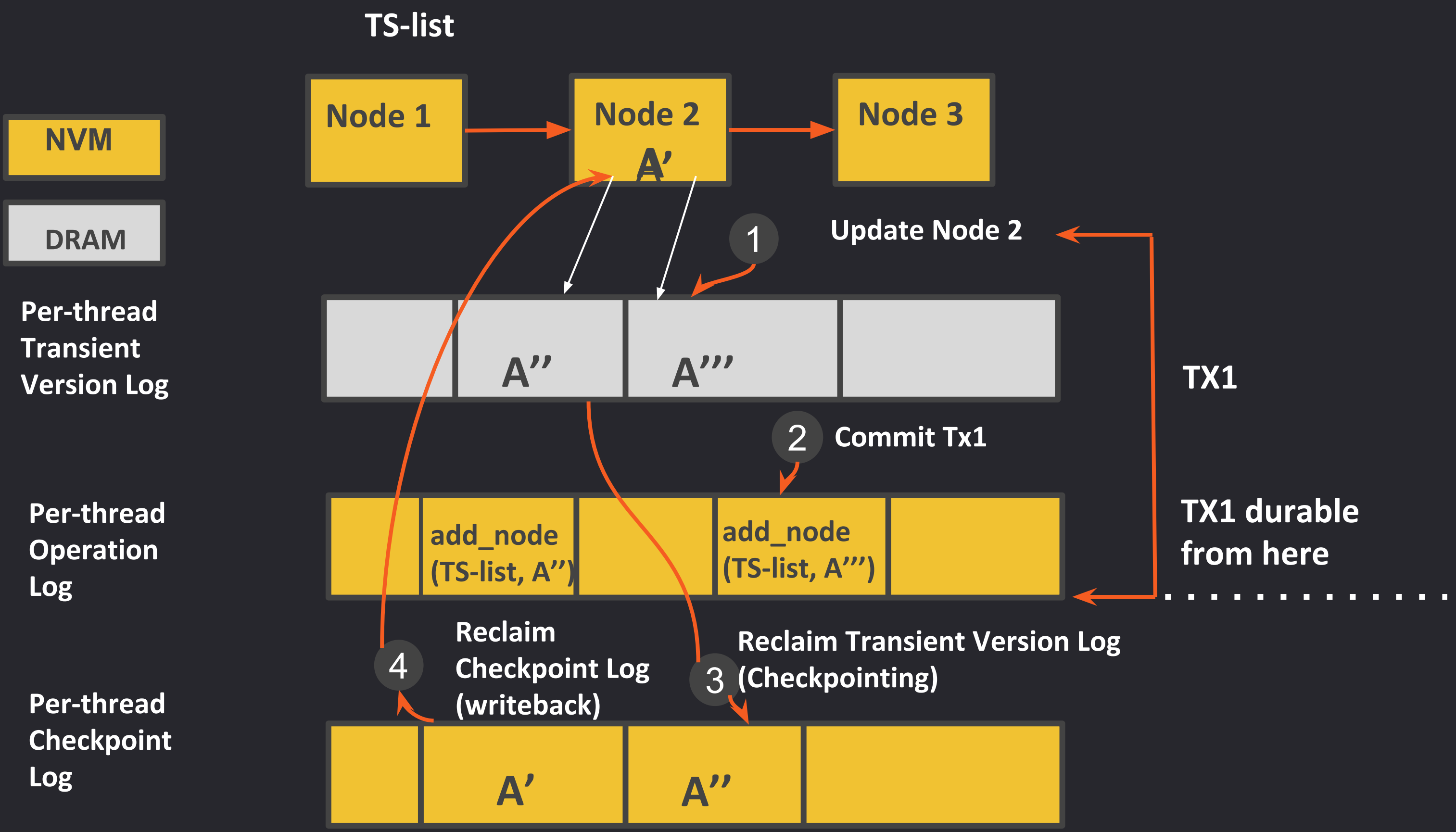
Recovery Design in Timestone

- Oplog replay executed in the order of start-ts and commits in the order of commit-ts
- Starts-ts order ensures similar view to that of live transaction
- Commit-ts order brings application to the last consistent state observed
- Using oplog reduces the NVM footprint.
- We achieve a deterministic and no-loss recovery.

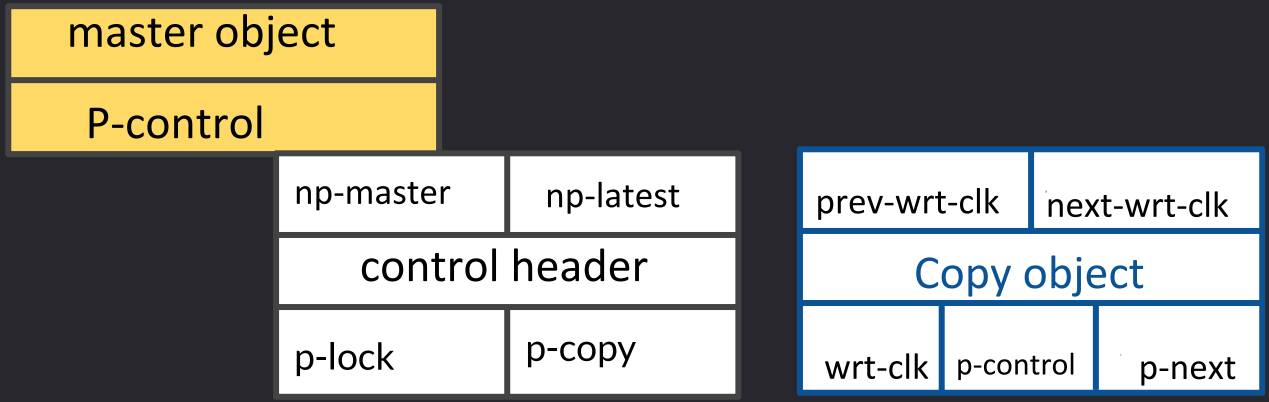
Scalable Garbage Collection

- Memory is finite!
- Writers are blocked if the log resources are full
- A non-scalable garbage collection will directly affect the write throughput
- We propose a asynchronous concurrent garbage collection scheme
- A thread itself is responsible for reclaiming its logs
- Reclamation are done according to the grace period semantics
- Cross log coordination is established without any centralized lookup or any dependency tracking
- We just use timestamps

- The Tlog and Clog are reclaimed in two different modes
    - Write back mode (when log_utilization > 75%)
    - Best effort mode (when log_utilization < 75% and > 30%)
- Thread checks for reclamation at the transaction boundary
- In write back mode the latest copy object is written back
    - All the other versions (belonging to same master) are ignored
- In best effort mode objects are reclaimed until the first writeback is required
    - Stopping at the first writeback allows to coalesce updates
- OLog entries can be discarded after Tlog writeback


TS-list
NVM
DRAM
Node 1
Node 2
A’
Node 3
Per-thread Transient Version Log
A’’
A’’’
① Update Node 2
TX1
② Commit Tx1
Per-thread Operation Log
add_node (TS-list, A’’)
add_node (TS-list, A’’’)
TX1 durable from here
④ Reclaim Checkpoint Log (writeback)
③ Reclaim Transient Version Log (Checkpointing)
Per-thread Checkpoint Log
A’
A’’

Object Structure in Timestone



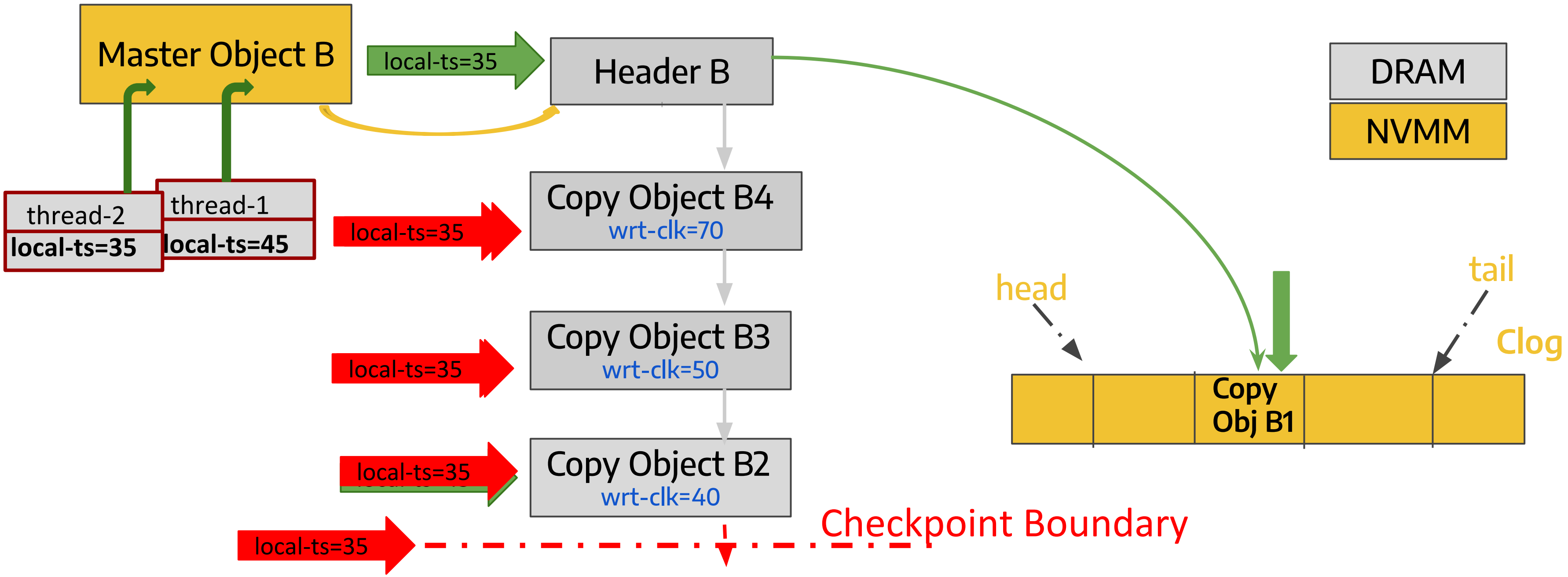
Principles Behind the Logging Design

- Per-thread logs to eliminate any scalability bottleneck
- Longer the object stays in the log better chance of absorbing redundant writes
- No two logs will have the same copy object at any given instant
- Effective use of QP clock boundary to decide the reclamation/writeback candidate
- On-fly construction of control header for all the non-volatile logs on DRAM
- NVM friendly access pattern design for nvlogs.

MVCC Transactional Model

- MVCC - Optimal design choice to achieve all features in one system
- Problems with MVCC
  - High version chain traversal cost
  - Global timestamp allocation bottleneck
- We employ a concurrent and asynchronous garbage collection scheme to solve version lookup cost
- We use hardware clock (RDTSCP in x86) for timestamp allocation
- A reader/writer will traverse the version chain to find the right version to dereference.
- The right copy is identified by timestamp lookup

Dereferencing - Finding the Right Version



DRAM

Update_node (A , V_3)

“Clog is 70% filled, I need to free up some space!! Let me trigger Writeback”

Key idea

3

Update_node (A , V_3)

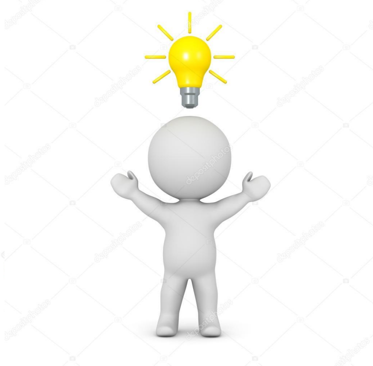
➢

➢

lates

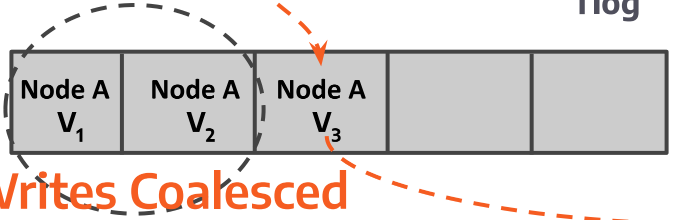
Looks good, But what happens if there is a power failure before Tlog checkpoints its updates?

log



Let me trigger ch

Tlog

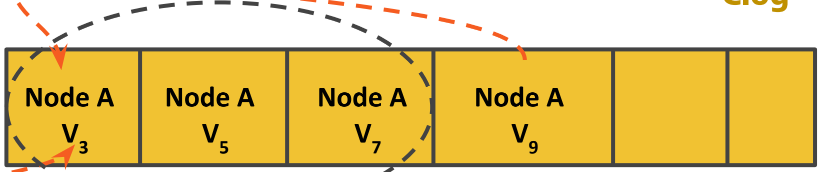


Writes Coalesced

Checkpointing

Writeback

Clog



Checkpoints Coalesced