TIPS: Making Volatile Indexes Persistent With DRAM-NVMM Tiering

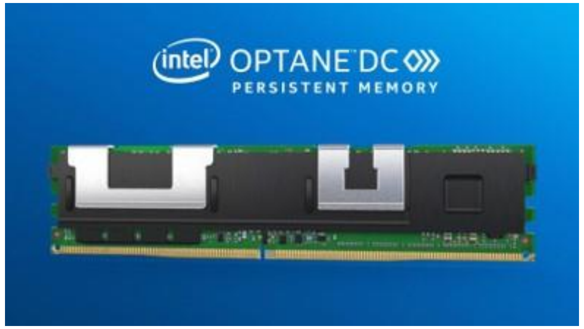
R. Madhava Krishnan,
Wook-hee Kim, Hee Won Lee[+*], Minsung Jang[†*],
Sumit Monga, Ajith Mathew, Changwoo Min

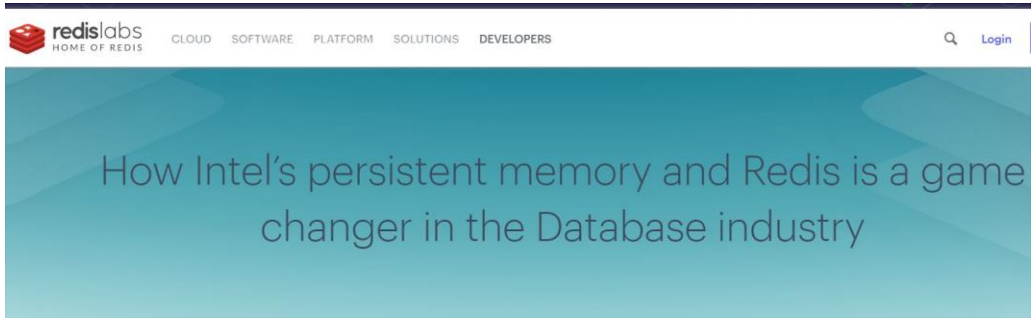
VIRGINIA TECH
[+] *Samsung Electronics*
[†] *Peraton Labs*

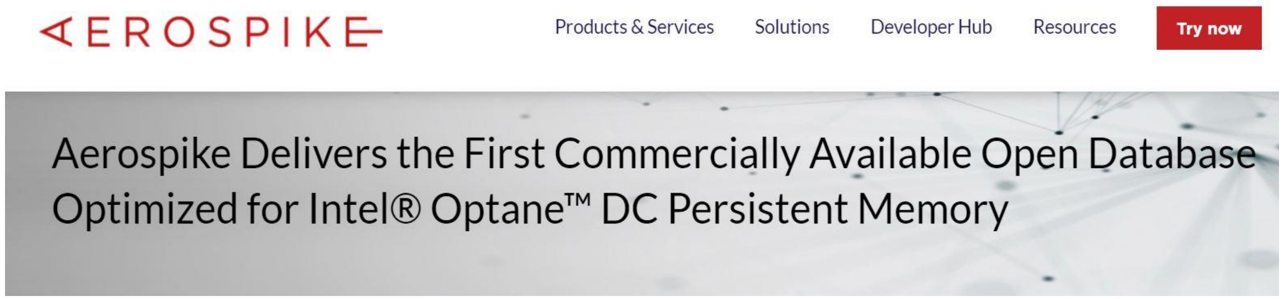
[*] The authors contributed to this work while they were at AT&T Labs Research

NVMM is Gaining Traction in Real-world Systems!

- Byte addressable Non-Volatile Main Memory (NVMM) has high capacity, low latency and durability
- Lots of interest in extending support for in-memory databases and key-value stores







Porting Volatile Indexes for NVMM is Crucial!

- Index structures are core part of in-memory databases
- Recent research works focuses on converting volatile indexes to work on NVMM
- Manual porting is complex and error-prone
- Provides framework or guidelines to facilitate the porting
- State-of-the-art index conversion techniques
  - NVTraverse [PLDI-20], PRONTO[ASPLOS-20], RECIPE[SOSP19]

Existing Techniques Have a Narrow Scope

- Existing conversion techniques are proposed based on the concurrency control
  - NVTraverse [PLDI-20] for lock-free indexes, e.g., Atomic CAS
  - PRONTO [ASPLOS-20] for blocking indexes, e.g., Mutex
  - RECIPE [SOSP-19] for fine-grained and lock-free indexes

Existing Conversion Techniques Have Limited Applicability

Existing Techniques Have Other Critical Limitations

- ➢ Support only Buffered Durable Linearizability [RECIPE]
- ➢ Not handling persistent memory leaks [RECIPE, NVTraverse]
- ➢ In-depth knowledge on the volatile index [RECIPE, NVTraverse]
- ➢ Can not scale beyond the DRAM capacity [PRONTO]
- ➢ High crash consistency overhead [PRONTO]

We propose TIPS to solve these problems and make the overall conversion process simple, intuitive and less error prone

Talk Outline

- Motivation
- Overview
- Evaluation
- Conclusion

VIRGINIA TECH.

Three Main Goals of TIPS

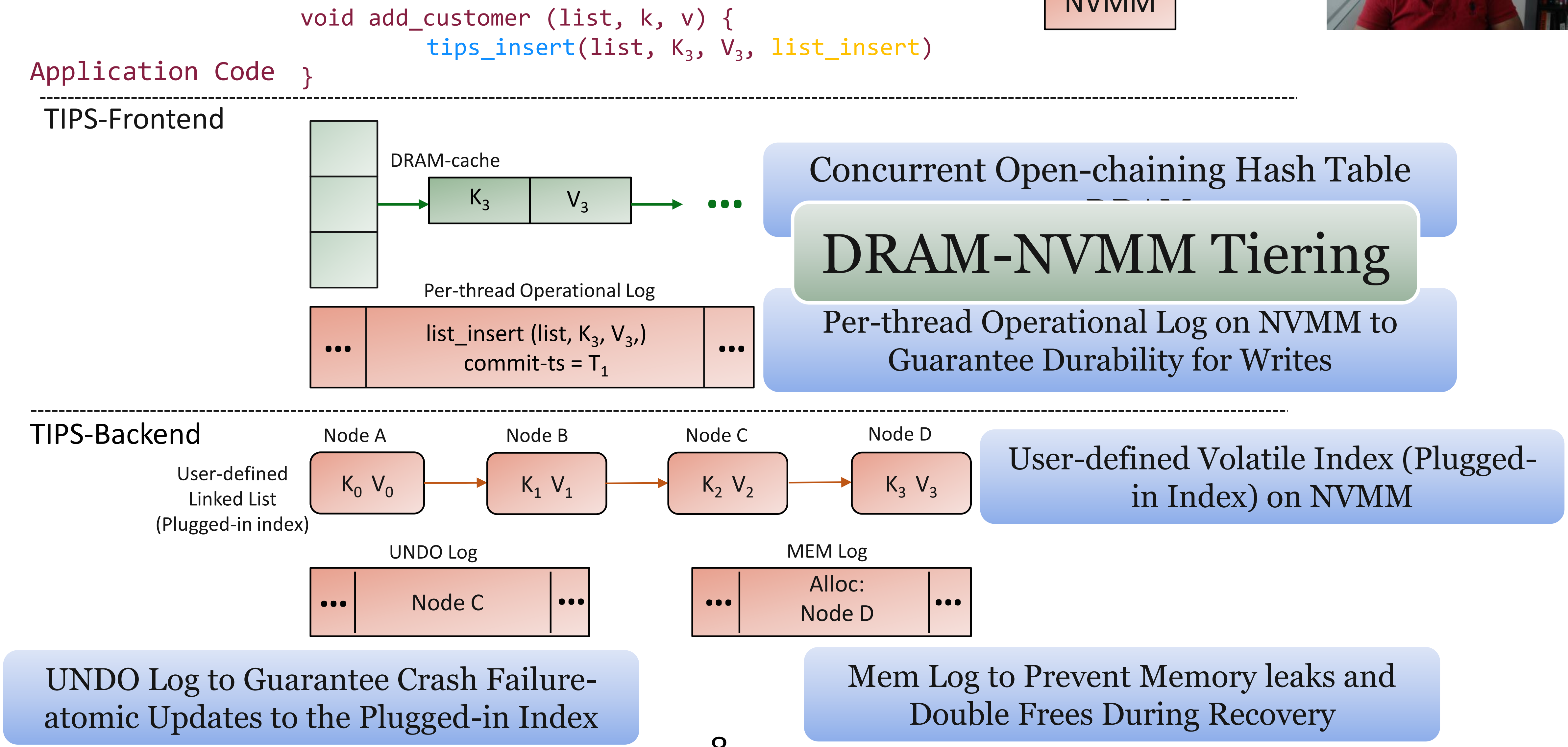
1) Support an Index-agnostic Conversion

2) Guarantee Durable Linearizability for Correctness

3) Provide High-Performance and Scalability

TIPS Architecture

DRAM
NVMM



Application Writes are Absorbed in TIPS-Frontend

TIPS Facade API

```
Void add_customer (list, k, v) {
    tips_insert(list, K3, V3, list_insert)
}
```

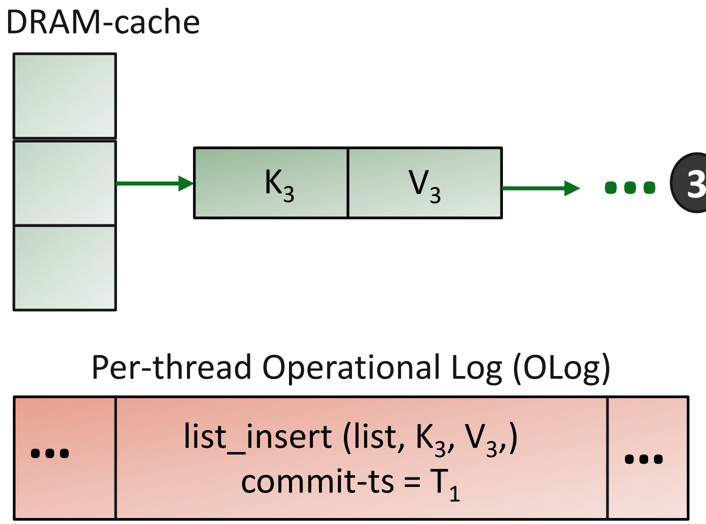
Application Code

4 Exit and return success to the application

Function Pointer to the Volatile Index Implementation

1 Execute the TIPS facade API with a pointer to the volatile index implementation

TIPS-Frontend



Durable Linearizability

3 Update the DRAM-cache to make the write visible ***(Linearization Point)***

2 Commit the write operation in the OLog and Persist the OLog record ***(Durability Point)***

DRAM

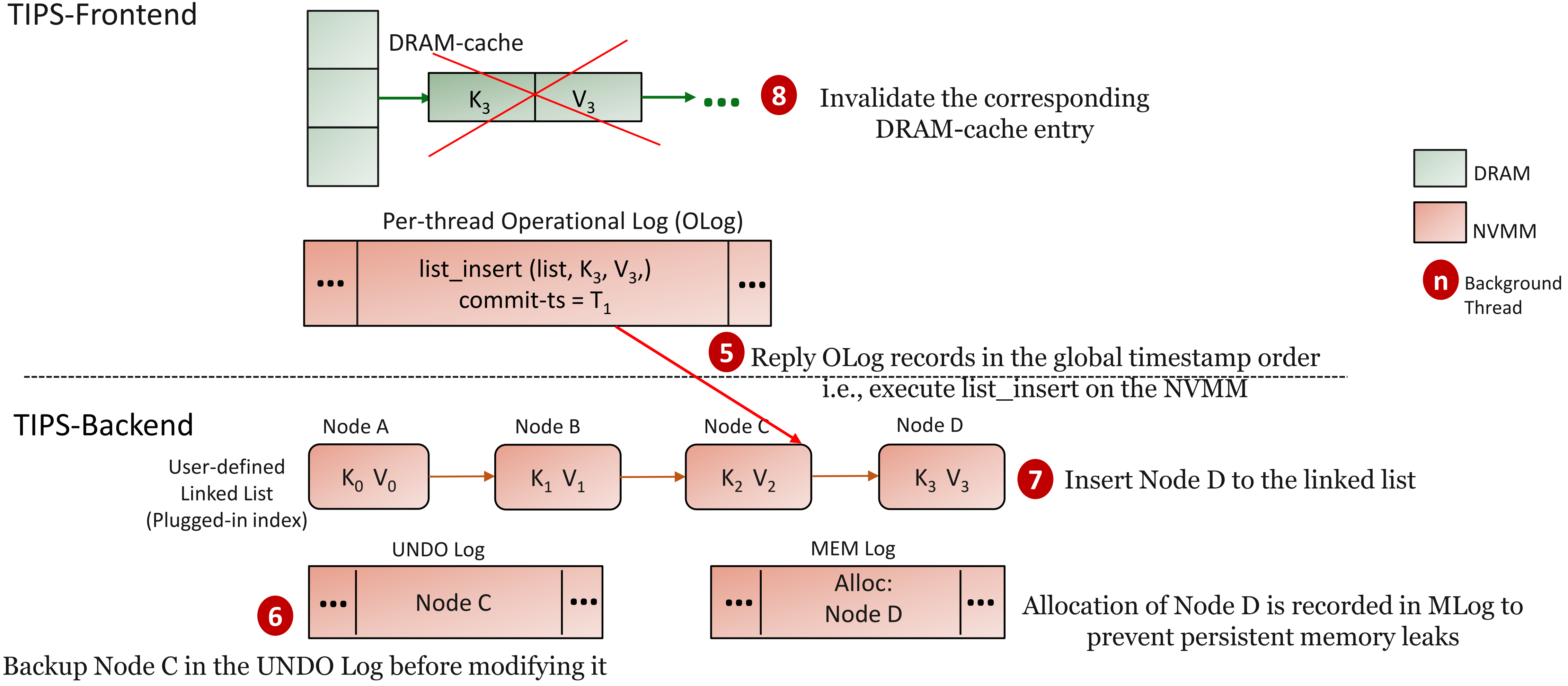
NVMM

n Application Thread

Writes always happen at the fast TIPS-Frontend; Parallel disjoint writes regardless of concurrency model supported by the plugged-in index

VIRGINIA TECH.

Plugged-in Index is Updated Using Background Threads



Key Benefits of DRAM-NVMM Tiering

- Support index-agnostic conversion
  - Allows plugged-in index to co-exist with the DRAM-cache
  - No restrictions on the concurrency model of the volatile index
- Two different levels of concurrency (Tiered Concurrency Model)
  - Concurrency model of DRAM-cache + Plugged-in index
  - DRAM-Cache supports concurrent lock-free reads and disjoint writes
  - Index with blocking concurrency (e.g., Mutex) can benefit from DRAM-cache
- Support Durable Linearizability agnostic of volatile index

VIRGINIA TECH.

Can the TIPS-Backend Become a Scalability Bottleneck?

- TIPS-Frontend is fast and scalable with concurrent DRAM-cache and per-thread operational logging
- Backend writes are inherently slower because of
  - Writes happening in the NVMM
  - Notorious UNDO logging overhead
- Slower backend can easily bottleneck the frontend
- How do we make the TIPS-backend scalable?

VIRGINIA TECH.

How TIPS Makes its Backend Scalable?

- ## A Key Intuition
  - Real-world workloads are rarely 100% writes
- We introduce two more techniques
  - UNO Logging Protocol to Reduce the UNDO Logging Overhead
  - Adaptive Scaling for Concurrent Background Writes

VIRGINIA TECH.

UNO Logging Protocol

- All three logs (OLog, ULog, MLog) in TIPS works synergistically
- Not all modified addresses are required to be UNDO logged
  - Selectively log only the addresses required for the correct recovery
- Perform UNDO logging only when the requested address
  - is not previously UNDO-logged i.e., avoid redundant UNDO logging
  - is not present in the OLog i.e., addresses that can not be recreated by OLog replay
- Significantly reduces the number of UNDO loggings performed

VIRGINIA TECH.

Benefits of UNO Logging

- Makes the backend writes fast
  - Number of UNDO logging is significantly reduced
  - Enables write coalescing in the UNDO log
- Reduces crash consistency overhead in the write critical path
  - Using OLog requires only 2 persist barriers
- Prevents persistent memory leaks
  - Addresses in the MLog can be freed upon recovery
- UNO logging is index-agnostic
  - applicable to any index irrespective of type or concurrency control

Adaptive Scaling of Background Writers

- TIPS uses Adaptive Scaling to concurrently update the plugged-in index
  - Carefully orders the operations for a faster concurrent reply
- Adaptive scaling has some very nice properties
  - Automatically adjusts the worker count based on workload nature
  - Optimizes worker count based on the write-scalability
  - Prevents wastage of CPU cycles and other hardware resources
- Refer to the paper for more details and correctness

Converting a Volatile Hash Table Using TIPS

```
void hash_insert(hash_t *hash, key_t key, val_t value)
{
    node_t **pprev_next, *node, *new_node;
    int bucket_idx;
    pthread_rwlock_wrlock(&hash->lock);

    // Find a node in a collision list
    bucket_idx = get_bucket(key);
    node       = hash->buckets[bucket_idx]->head;
    pprev_next = &hash->buckets[bucket_idx]->head;
    while (node && node->key < key) {
        pprev_next = &node->next;
        node = node->next;
    }
    // Case 1: update an existing key
    if (node->key == key) {
        node->value = value;
        goto unlock_out;
    }
    // Before modifying the value, backup the old value
       tips_ulog_add(&node->value, sizeof(node->value))
       node->value = value; // then update then the node
       goto unlock_out;
    }
    // Case 2: add a new key
    // Allocate a new node using tips_alloc
    new_node = malloc (sizeof(*new_node));
    new_node = tips_alloc (sizeof(*new_node));
    new_node->key   = key;
    new_node->value = value;
    new_node->next  = node;
    *pprev_next = new_node;
    // Before modifying the value, backup the old value
    tips_ulog_add(pprev_next, sizeof(*pprev_next))
    *pprev_next = new_node; // then update then the node

unlock_out:
    pthread_rwlock_unlock(&hash->lock);
}
```

M2 (Case 1 block)

M1 (Case 2 block)

➢ Two simple guidelines for the conversion

- ❑ Replace the memory allocation/free with tips_alloc or tips_free
- ❑ Add tips_undo_add before modifying any NVMM address

➢ Key Benefits

- ❑ No need to manually insert flush/fence
- ❑ Makes the conversion simple and trivial
- ❑ Developers need not worry persistence and visibility ordering

VIRGINIA TECH.

Other Interesting Designs

- Concurrency model and epoch-based GC in DRAM-cache
- Scan operation
- Adaptive Scaling
- UNO logging reclamation
- Recovery algorithm
- Detailed correctness section

Talk Outline

- Motivation
- Overview
- Evaluation
- Conclusion

VIRGINIA TECH.

Evaluation Questions

- How much LoC are required to convert an index using TIPS?
- How does TIPS perform against the prior index-specific conversion techniques?
- How does TIPS perform against the NVMM-optimized indexes?

Evaluation Settings

- 2 socket server with Intel DCPMM
  - 512GB NVMM and 64GB DRAM
  - 2.4 GHZ 64 core Intel Xeon Gold CPU
- We evaluate 7 Indexes with different concurrency model
- YCSB with 32M keys for both integer and string type keys

Workload Name	Read/Write/Scan Ratio	Workload Nature
Workload A	50/50/0	Write intensive
Workload B	95/5/0	Read intensive
Workload C	100/0/0	Read only
Workload D	95/5/0	Read Latest
Workload E	0/5/95	Short Range Scan

Evaluation Settings

- DRAM-cache size is set to 25% (300 MB)
- Compared against the state-of-the-art index conversion techniques
  - PRONTO [ASPLOS-20]
  - NVTraverse [PLDI-20]
  - RECIPE [SOSP-19]
- And against NVMM-optimized indexes
  - Hash Indexes- CCEH [FAST-19], LevelHashing [OSDI-18],
  - B+Tree Indexes- FastFair [Fast-18], BzTree[VLDB-18]
  - Radix Tree Indexes- WOART [FAST-17]

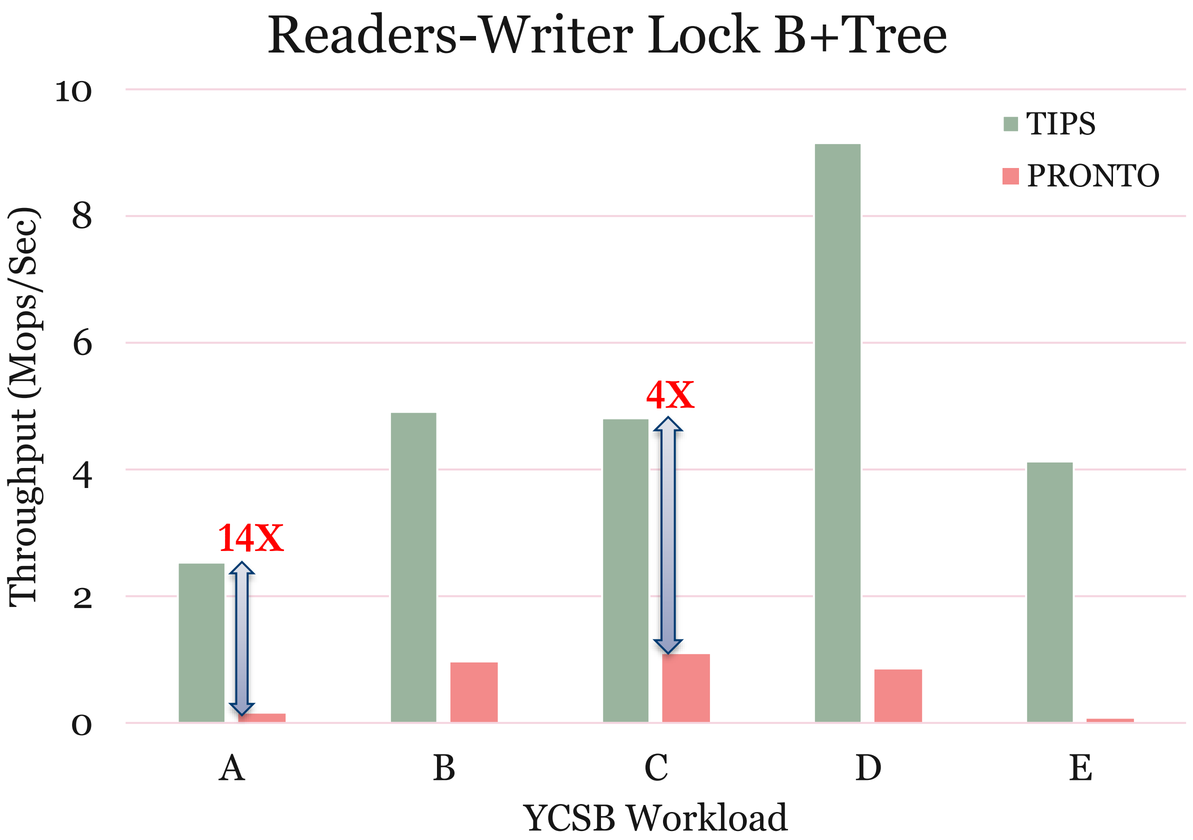
VIRGINIA TECH.

LoC Required for Conversion

Indexes	Concurrency Control	LoC change/ original LoC
Hash Table (HT)	Readers-Writer Lock	5/211
Lock-Free Hash Table (LFHT)	Non-blocking reads and writes	5/199
Binary Search Tree (BST)	Readers-Writer Lock	5/203
Lock-Free Binary Search Tree (LFBST)	Non-blocking reads and writes	5/194
B+Tree	Readers-Writer Lock	8/711
Adaptive Radix Tree (ART)	Non-blocking reads and blocking writes	9/1.5k
Cache-Line Extensible Hash Table (CLHT)	Non-blocking reads and blocking writes	8/2.8k
Redis Key-value Store	Blocking reads and writes	18/10k

TIPS has better applicability and requires minimal code changes in the original codebase

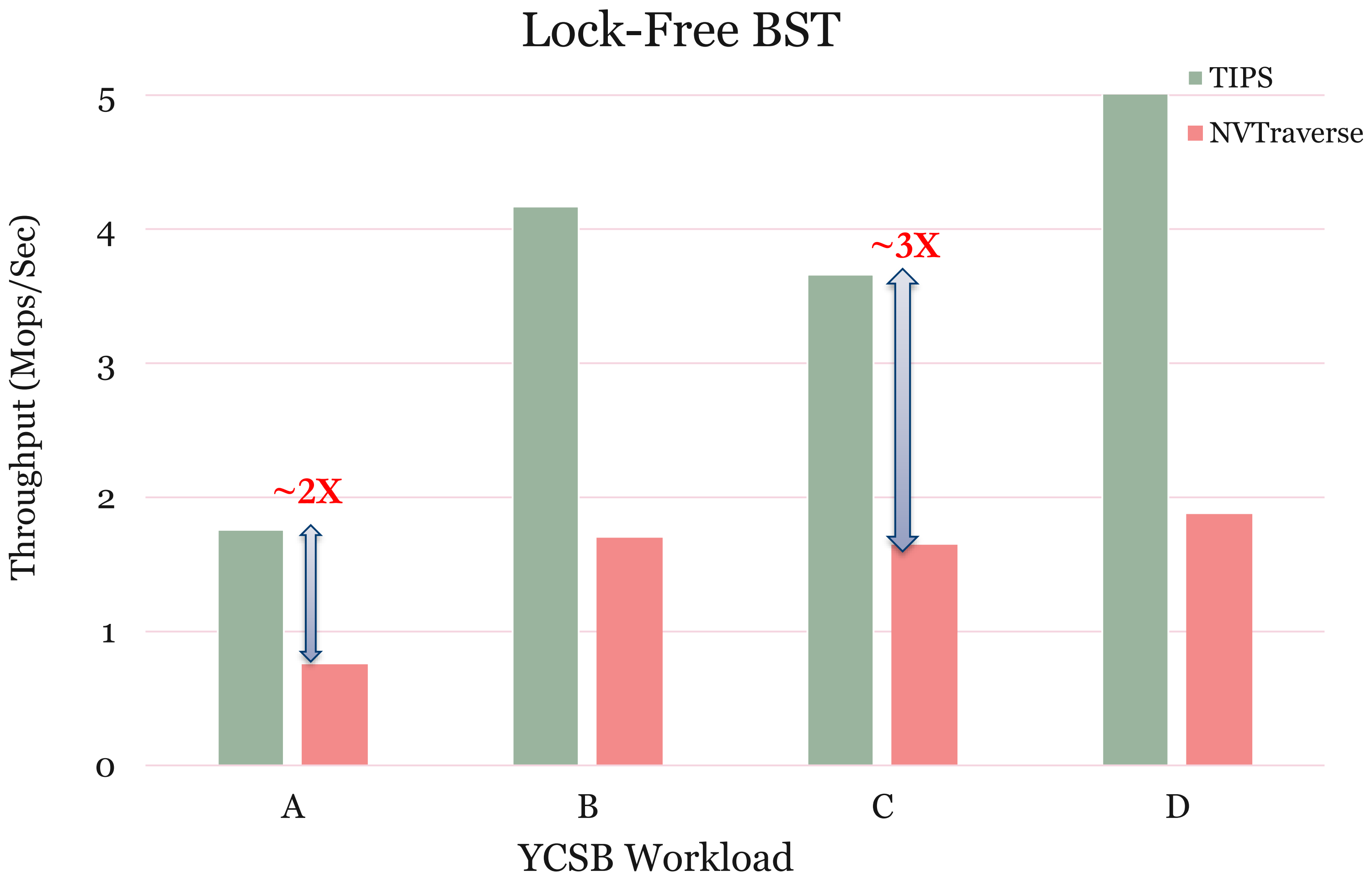
TIPS vs PRONTO for Blocking Indexes



- ❑ TIPS outperforms PRONTO by up to 14X
- ❑ TIPS can support concurrent reads/writes with its DRAM-cache

Pronto: Easy and Fast Persistence for Volatile Data Structures [ASPLOS-2020]

TIPS vs NVTraverse for Lock-Free Indexes



- NVTraverse incurs 6 and 17 p-barriers for reads and writes
- TIPS incurs 2 p-barriers in the write critical path
- No p-barriers required for reads in TIPS

NVTraverse: In NVRAM Data Structures, the Destination Is More Important Than the Journey [PLDI-2020]

Other Interesting Evaluations

- Performance comparison with the NVMM-optimized indexes
- Empirical analysis of TIPS design
- Scalability, skewness, large datasets etc.
- Sensitivity analysis
- Real-world application Redis
- More information on our conversion experience

Conclusion

- Current Index conversion techniques
  - Limited applicability
  - Weak consistency guarantee
  - Not address persistent memory leak
- **TIPS**
  - No restrictions on concurrency model
  - Offers strong consistency i.e., Durable Linearizability
  - In addition to providing outstanding performance and scalability
  - https://github.com/cosmoss-vt/tips

Thank You