# Log-Structured Non-Volatile Main Memory

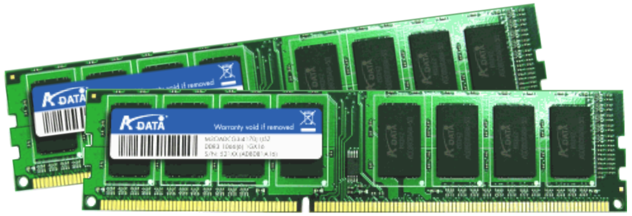
Qingda Hu*, **Jinglei Ren**, Anirudh Badam, and Thomas Moscibroda
**Microsoft Research** *Tsinghua University

USENIX ATC '17
JULY 12–14, 2017
SANTA CLARA, CA

Microsoft Research
微软亚洲研究院

# Non-volatile memory is coming…

- **Data storage**



Read: ~50ns
Write: ~10GB/s

Read: ~10µs
Write: ~100MB/s

intel Micron

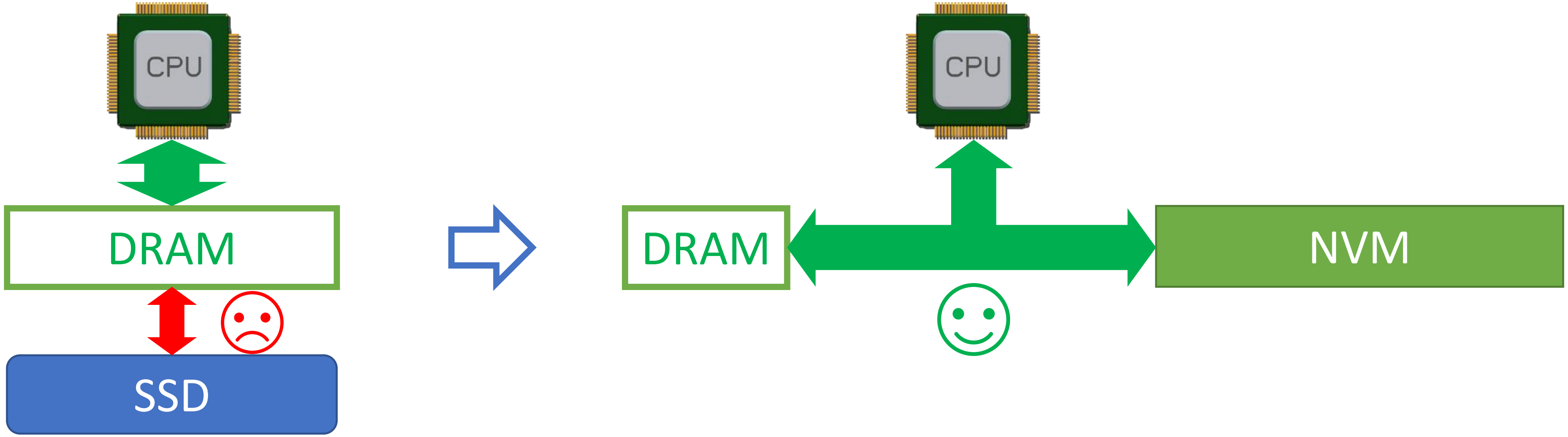
*3D XPoint/Optane (2015 - )*

PCM

Read: ~100ns
Write: ~1GB/s

# Background: Impact of NVM

- **Architecture**: Non-Volatile Main Memory (NVMM)

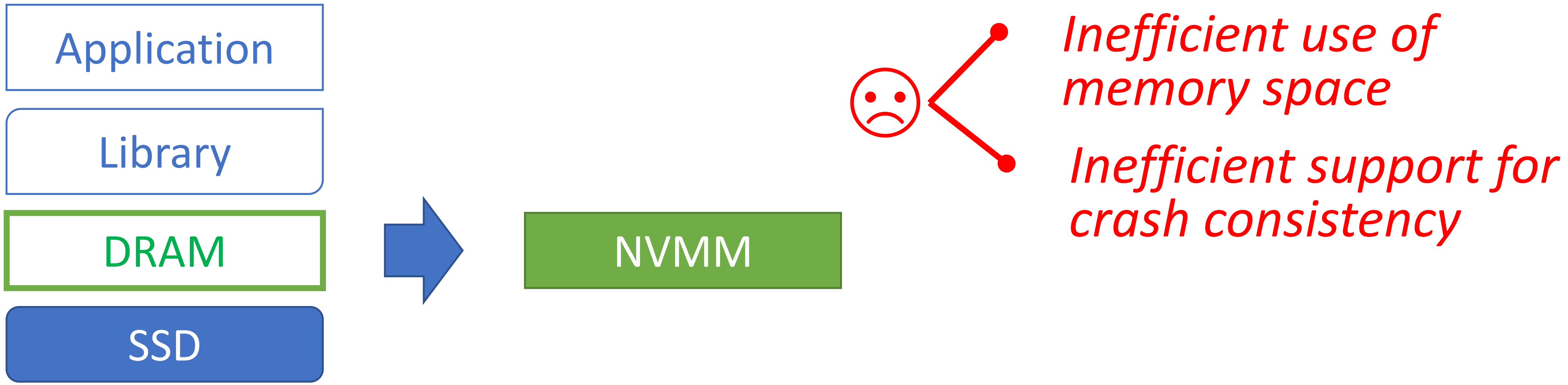



- ~~Data persistence as a bottleneck~~

➔ 10+x application performance improvement

# Executive Summary

- **Motivation**



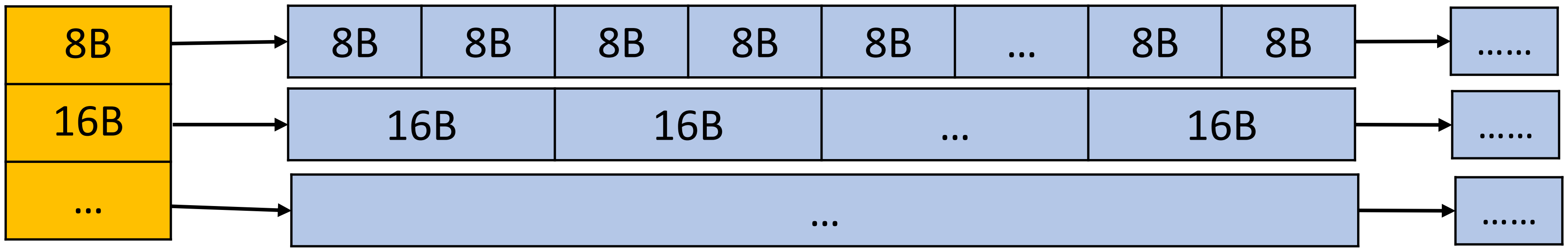

- **Solution**: *Log-structured* memory management for NVMM.
- **Evaluation**: 7x less memory waste; 90% higher write throughput.

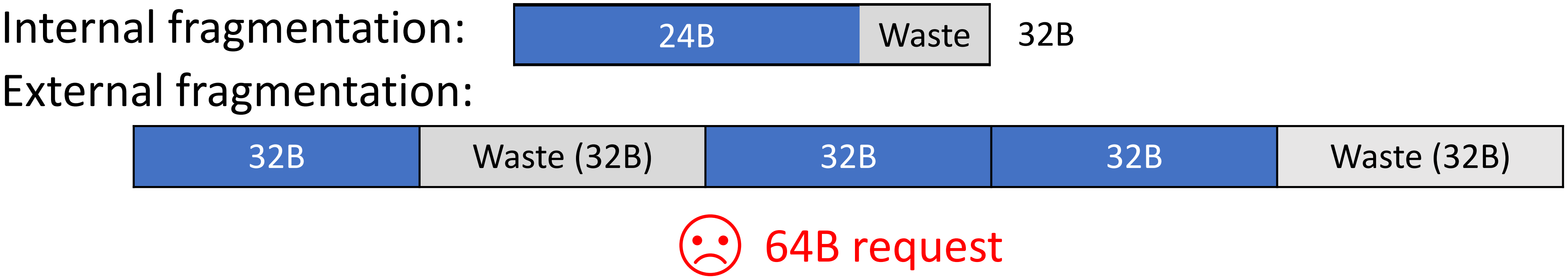
# Outline

- ***Motivation***
- Log-Structured NVMM
- Tree-Based Address Mapping
- Evaluation

# Motivation I

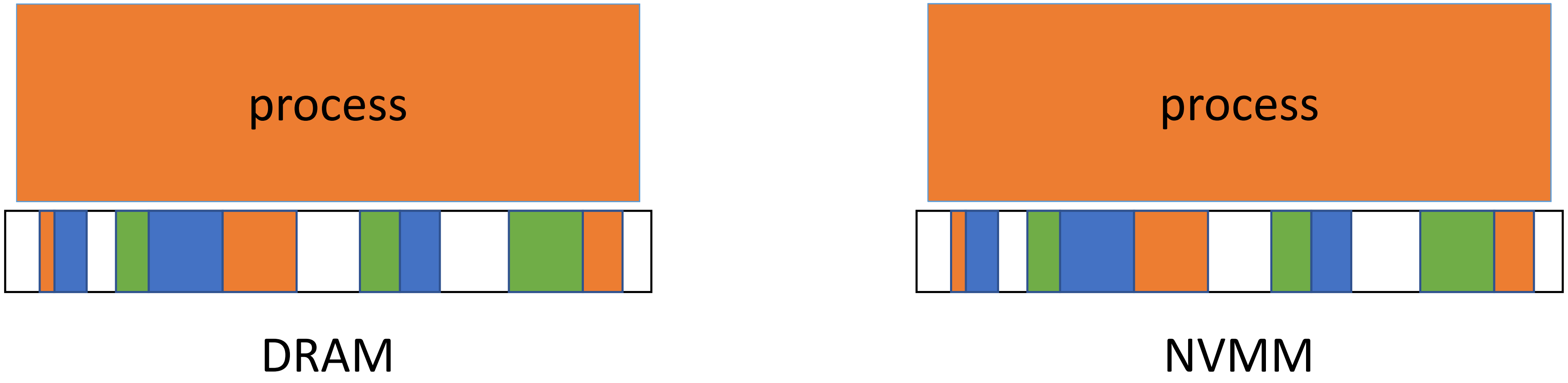
- **Inefficient use of memory space**
  - **Reason**: Traditional DRAM allocators incur *high memory fragmentation*.
  - **Explanation**:



# Motivation I

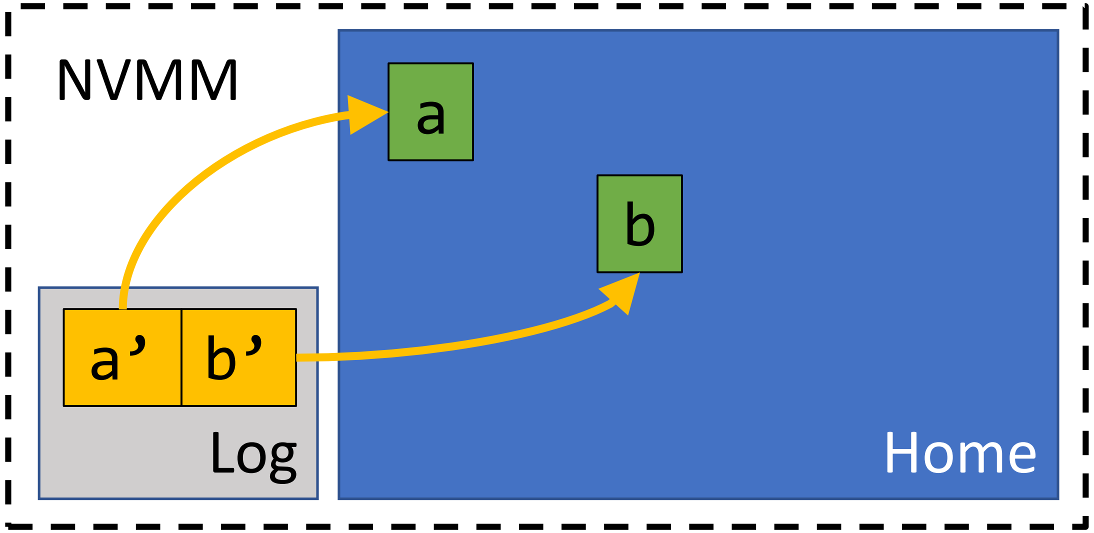
- **Inefficient use of memory space** (cont.)
  - *Fragmentation is a more severe issue for NVM!*



# Motivation II

- **Inefficient support for crash consistency**
  - **Reason**: *Write-twice in log and home*.
  - **Explanation**: Redo logging for example.

```
transaction {
  a += 1;
  b -= 1;
}
```

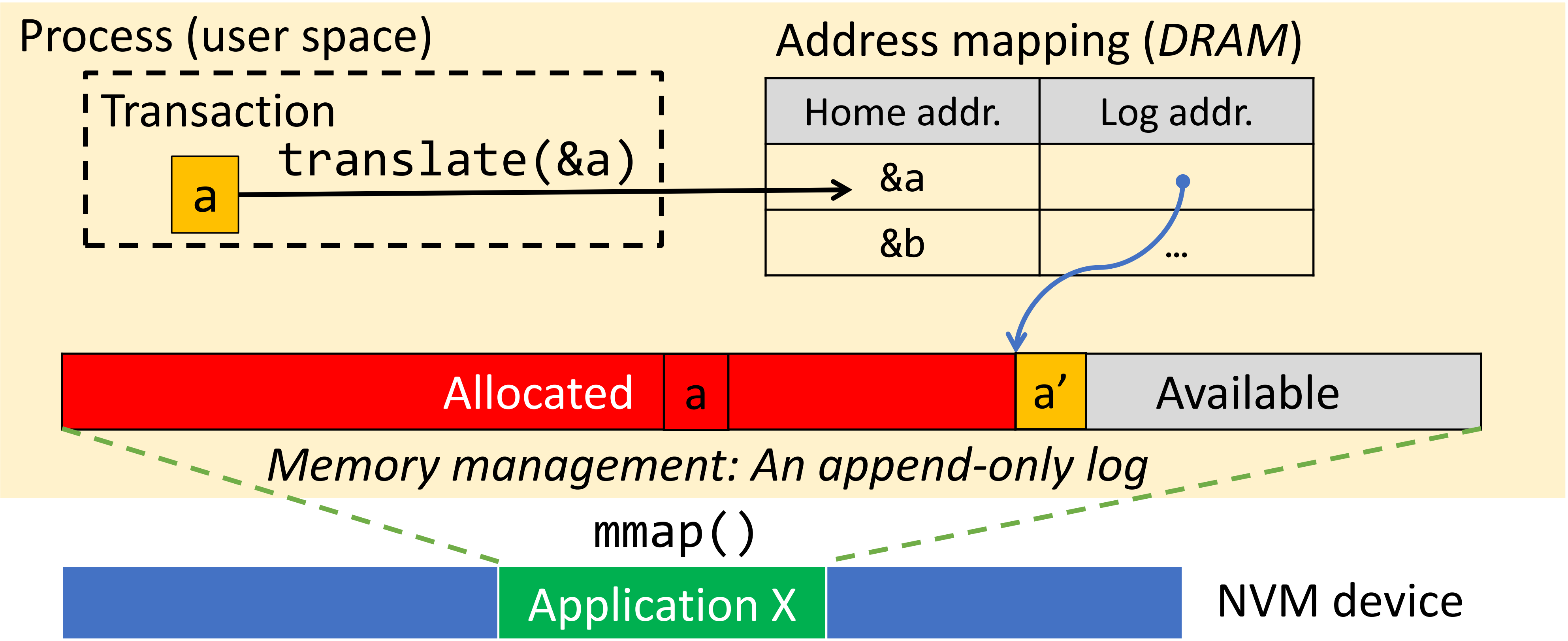


# Outline

- Motivation
- ***Log-Structured NVMM***
- Tree-Based Address Mapping
- Evaluation

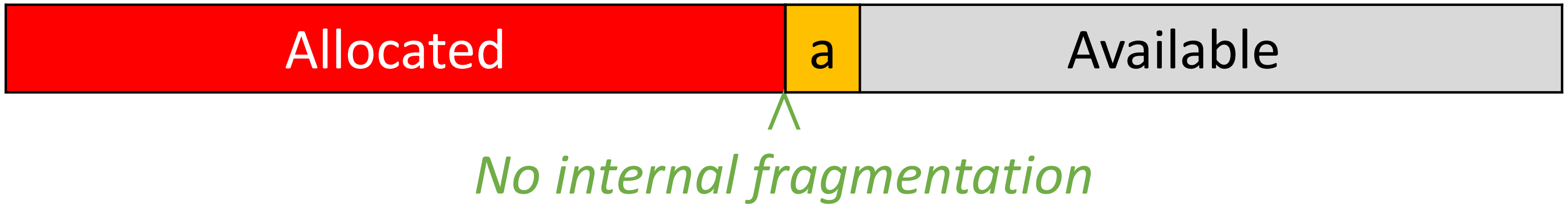
# Log-Structured NVMM

- Library and architecture

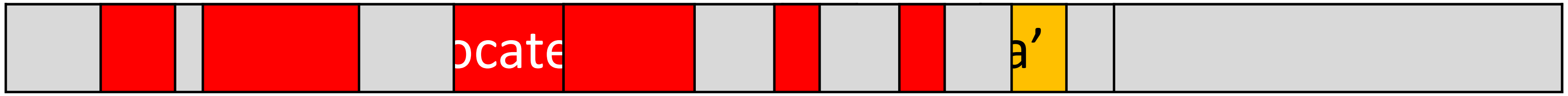


# Log-Structured NVMM

- Low fragmentation
  - For internal fragmentation: *Compact append*



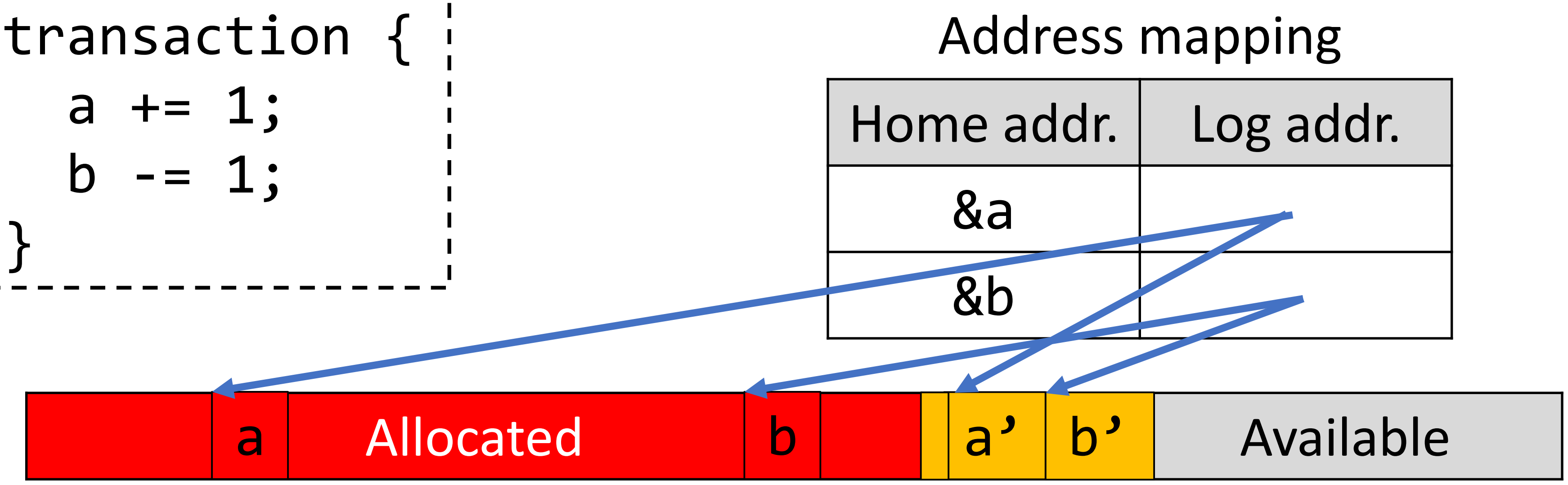

  - For external fragmentation: *Log cleaning*



# Log-Structured NVMM

- Efficient crash-consistent update
  - No separate areas. Write only *once*.

```
transaction {
  a += 1;
  b -= 1;
}
```



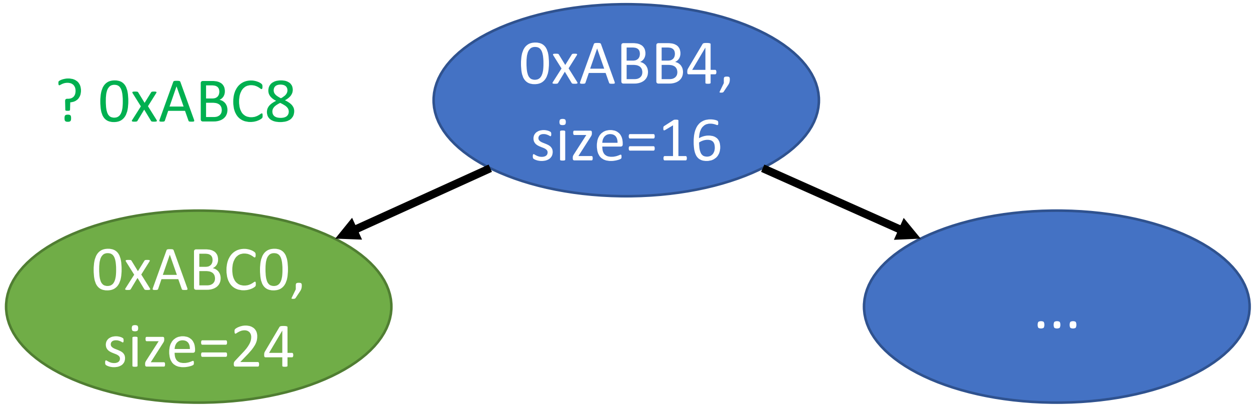

  - Header: size, *checksum*, etc.

# Outline

- Motivation
- Log-Structured NVMM
- ***Tree-Based Address Mapping***
- Evaluation

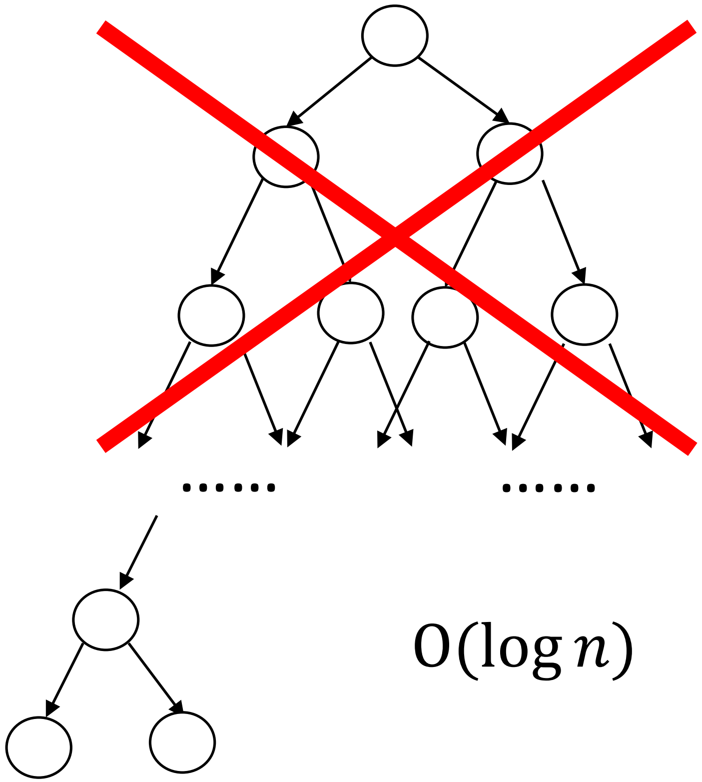
# Tree-Based Address Mapping

- **Unique challenges to NVMM**
  - *Pervasive* and *highly frequent* memory accesses.
  - *Allocation granularity ≠ access granularity* → No O(1) lookup.
    - Filesystems: hash(*block number*) as the index. ☺
    - Databases: hash(*key* or *tuple ID*) as the index. ☺
    - Main memory: hash(address)? That maps *every* address! ☹
- *Tree-based mapping made performant.*



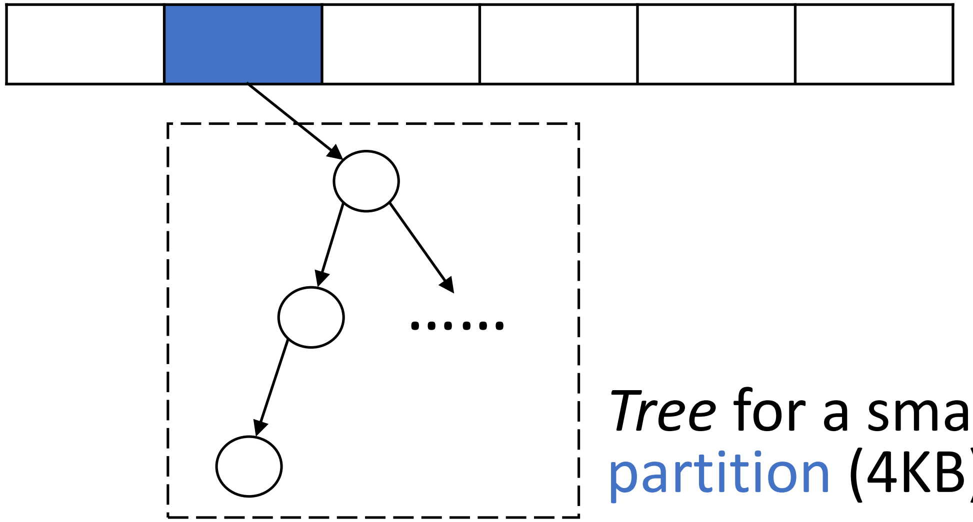
# Tree-Based Address Mapping

- Two-layer mapping




$O(\log n)$

*Partition index*: $O(1)$

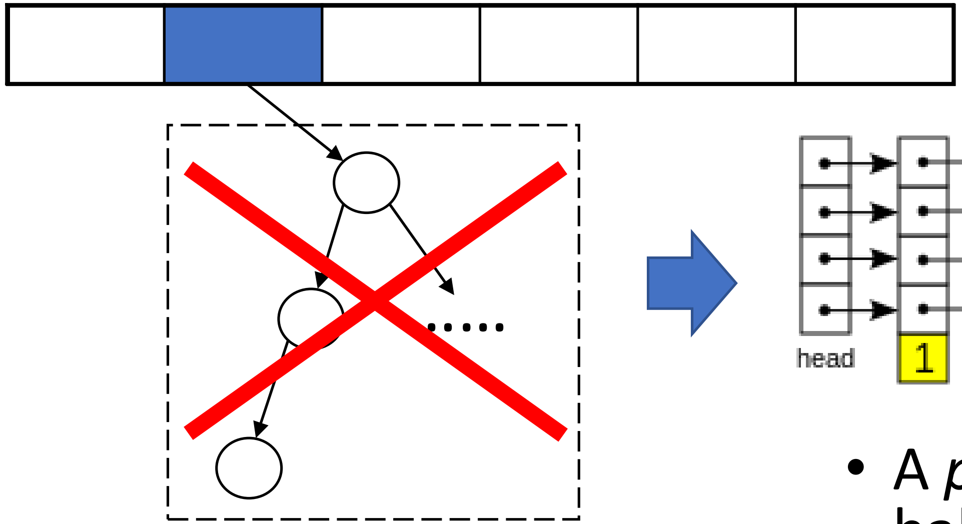


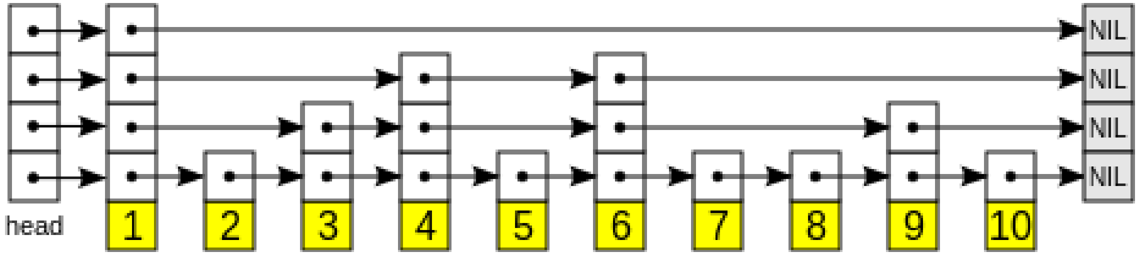

*Tree* for a small partition (4KB)

- *Improves transaction throughput by 39.6% on average.*

# Tree-Based Address Mapping

- Skip list





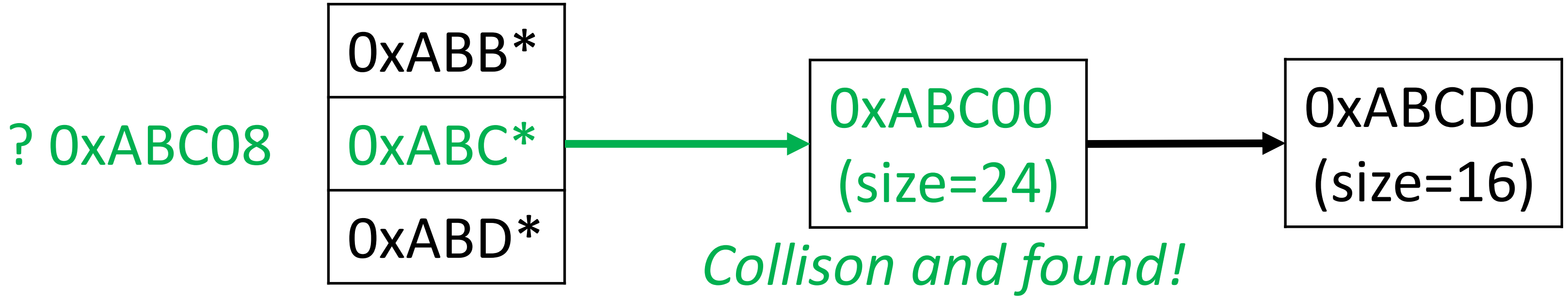

- A *probabilistically* balanced tree. No complex balancing operations ➔ No locking for read-only operations.
- *Improves transaction throughput by 48.9% with four threads.*

# Tree-Based Address Mapping

- Group update
  - Within each transaction, all writes are first *buffered in DRAM*.
  - Writes with contiguous addresses are combined on transaction commit.
  - *Improves transaction throughput by 42.3% on average.*

# Tree-Based Address Mapping

- Hot tree node cache
  - A *thread-local* cache that references recently accessed nodes of the trees.
  - A special hash table design: *Deliberately high collision.*
    - **Motivation**: Addresses within a cached node are not hit due to random distribution of their hash values.
    - **Solution**: Use *high-order bits* of an address as its hash value.




  - *Improves transaction throughput by 30.1% on average.*

# Outline

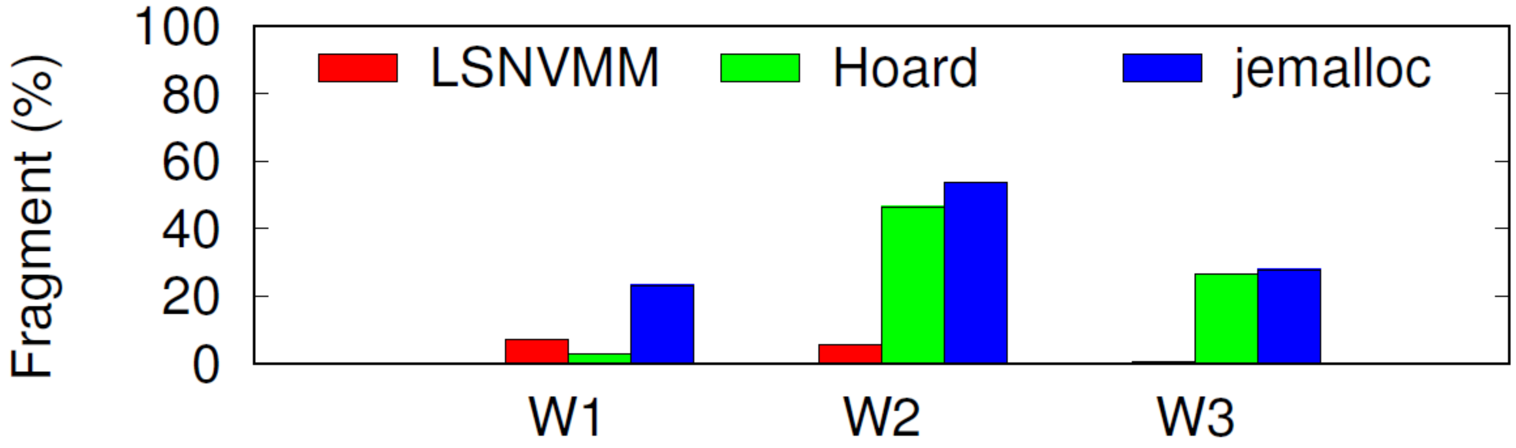
- Motivation
- Log-Structured NVMM
- Tree-Based Address Mapping
- ***Evaluation***

# Evaluation

- Environment:
  - 8-core Intel Xeon CPU E5-2637 v3 (3.5 GHz), 64 GB DRAM
  - 64-bit Linux kernel version 4.2.3
  - *NVM emulation*: write latency = $\max\{500\text{ns}, \frac{write\_size}{1\text{GB}/s}\}$
- Part I: How effective are *individual* optimizations? – Already shown.
- Part II: How does LSNVMM perform against traditional systems?
- Part III: What are the *inherent costs* of the log-structured approach?

# Evaluation

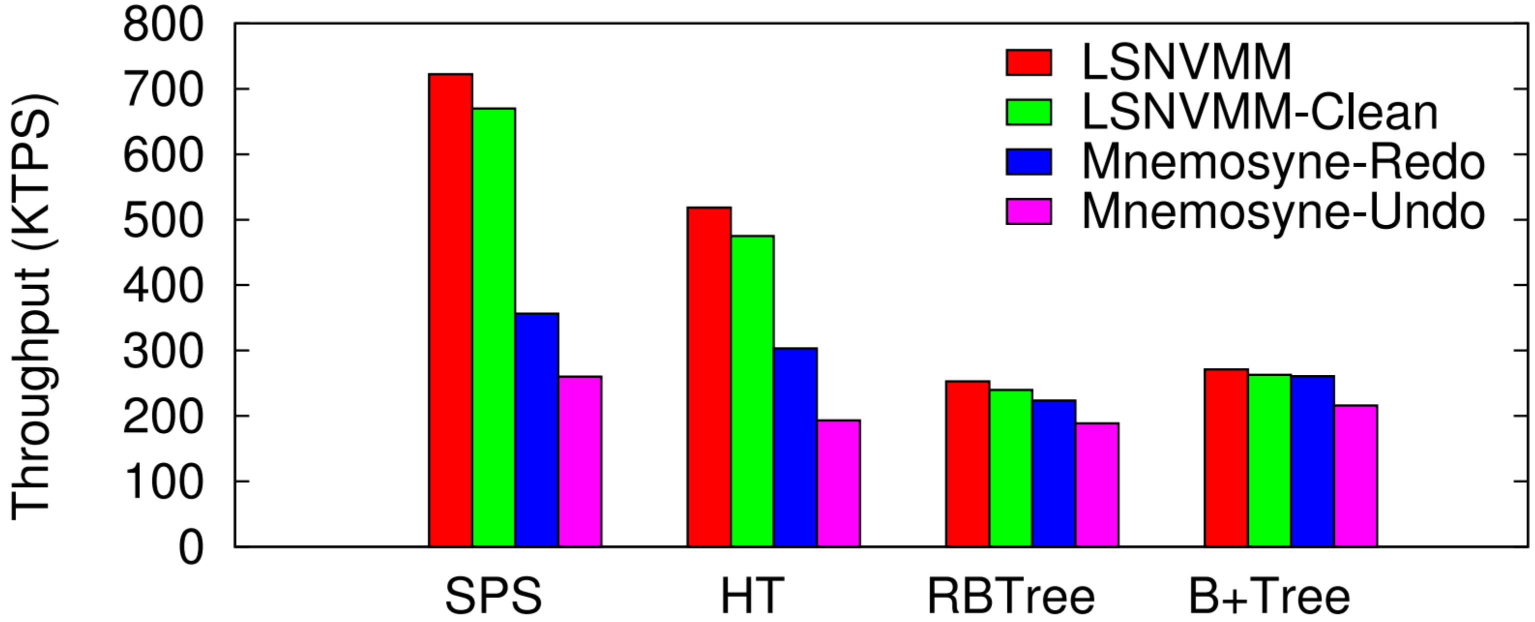
- Fragmentation: Compared to Hoard and jemalloc




  - Workloads 1 ~ 3 collected from [S. Rumble, FAST ’14].
  - Hoard/jemalloc produces *25.3%/35.0%* fragmentation on average.
  - ➢Log-structured NVM (LSNVMM) produces 4.5% fragmentation on average.

# Evaluation

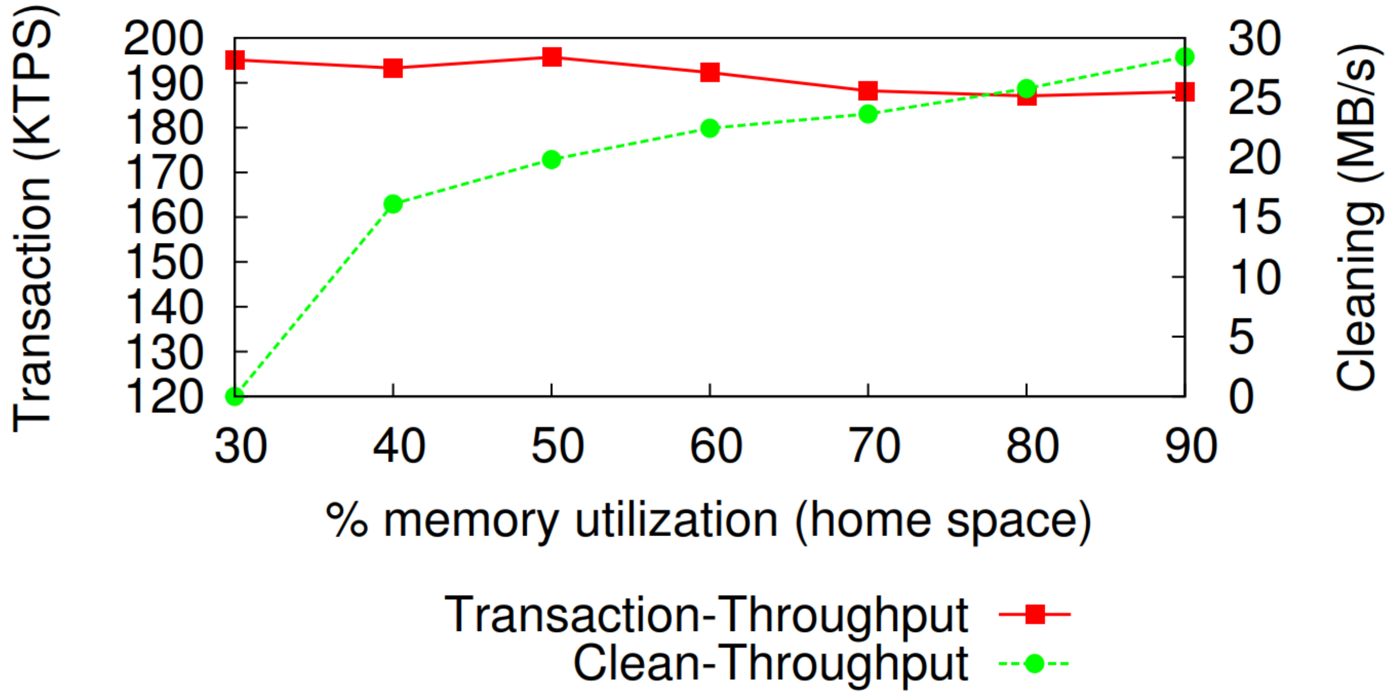
- Transaction throughput compared to Mnemosyne




  - With 4 threads, log-structured NVMM performs 44.7% and 80.8% better than Mnemosyne and Mnemosyne-Undo, respectively, on average.

# Evaluation

- Cost of log cleaning




  - The performance degradation due to log cleaning is 8% at *90% memory utilization*.

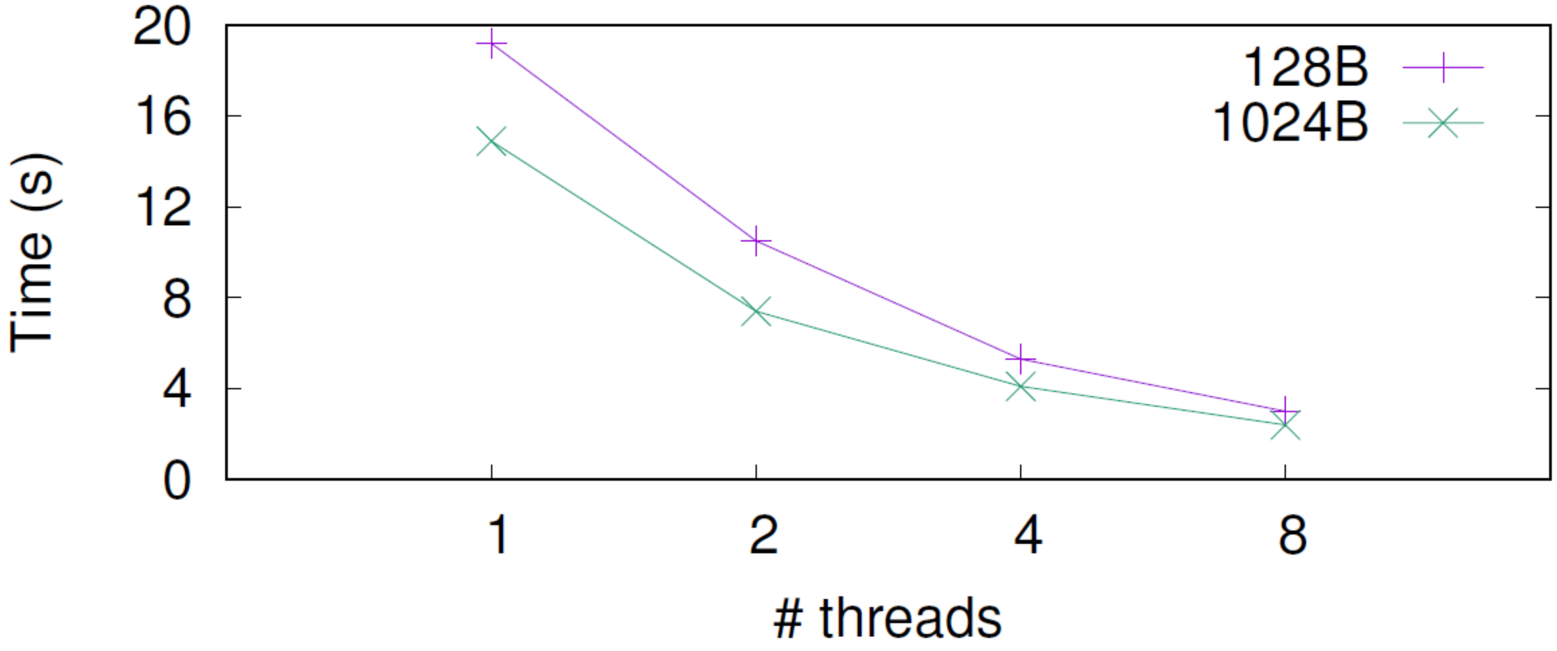
# Conclusion

- **Takeaway I**: Applying the *log-structured* approach to NVMM can largely reduce memory fragmentation and improve system performance.
- **Takeaway II**: A *tree*-based address mapping mechanism can be made efficient to serve log-structured NVMM.

- **Thank you!**
- Q & A

# Backup

- Recovery time (10GB logs)



# Backup

- DRAM footprint (1GB data)