# A NUMA-aware NVM File System Design for Manycore Server Applications

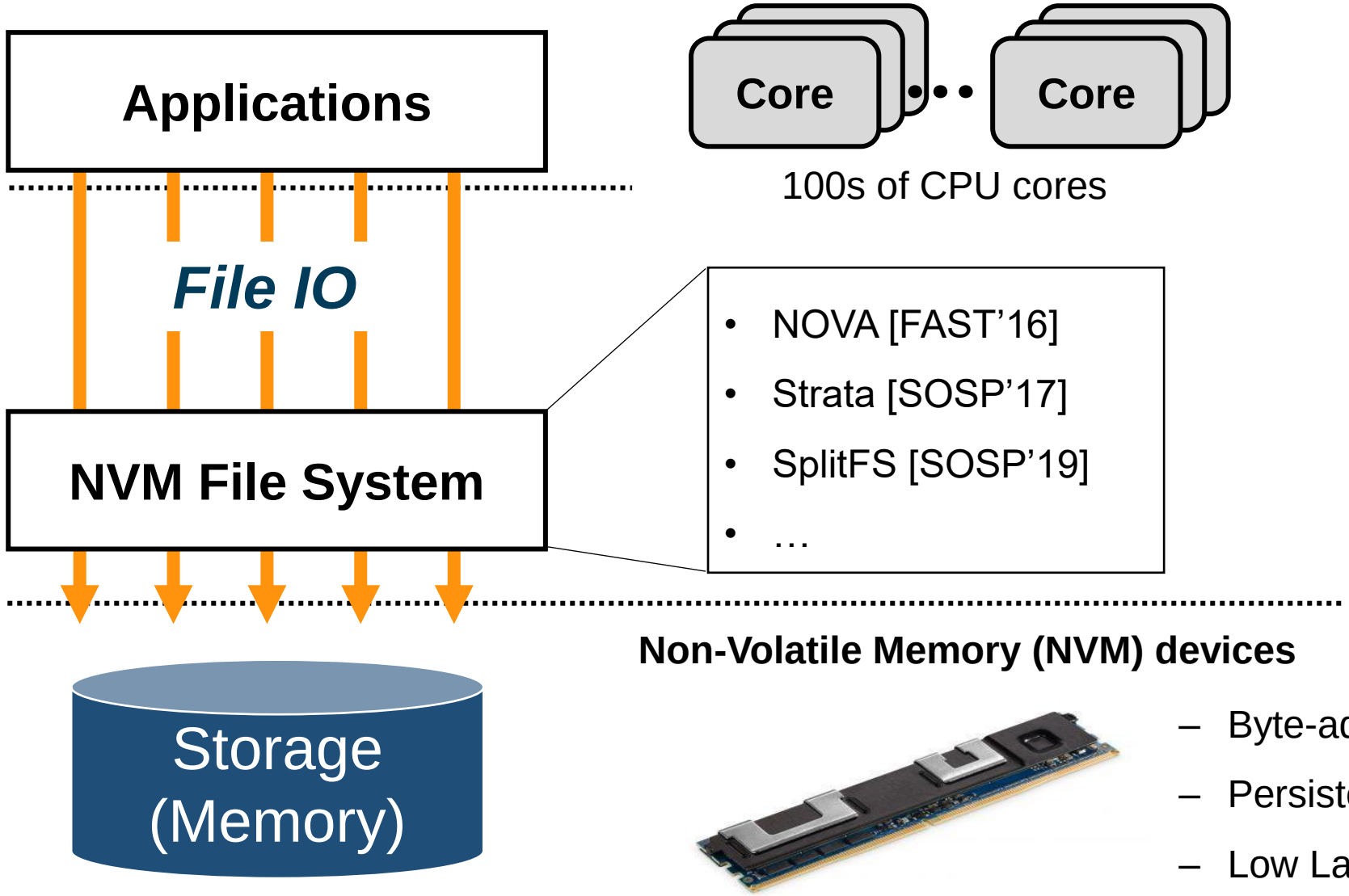
June-Hyung Kim, Youngjae Kim, Safdar Jamil, and Sungyong Park

**Nov 17, 2020**

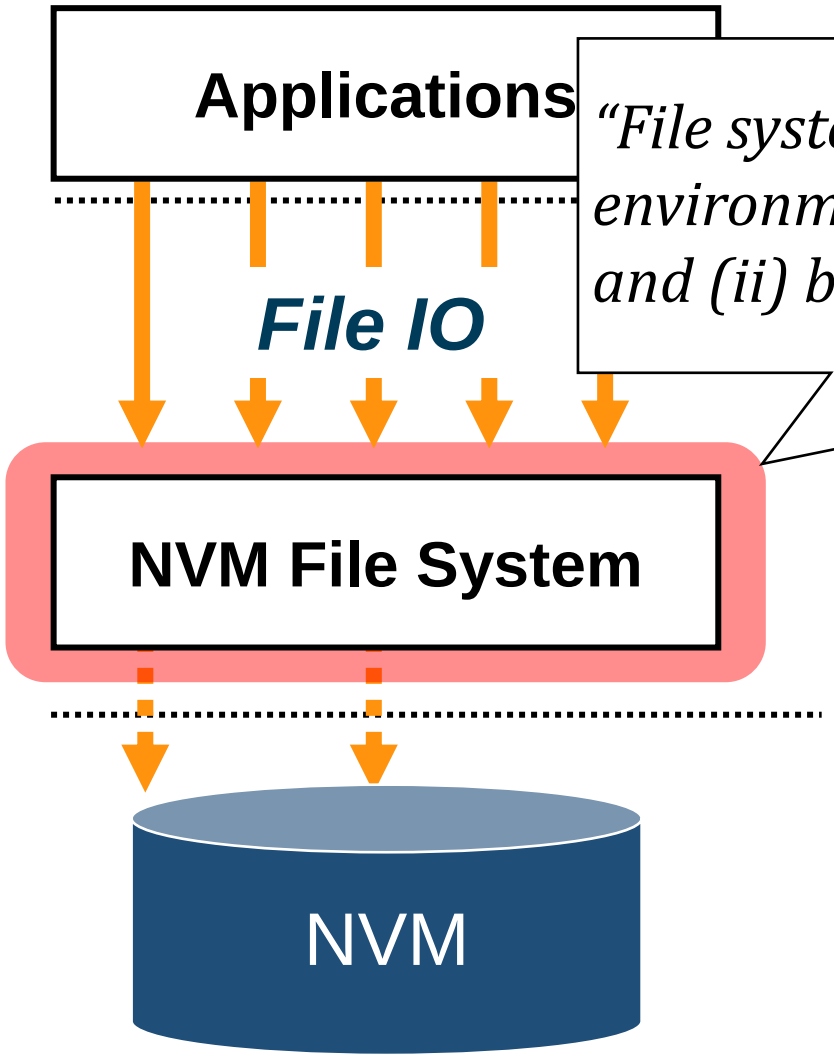
**MASCOTS'20**


SOGANG UNIVERSITY

# NVM File System on Manycore Server



# Parallel I/O Bottleneck in Non-Scalable File System




*"File systems that are not intended for a many-core environment cannot actively utilize (i) I/O parallelism and (ii) bandwidths of high-speed storage devices."*

SpanFS [ATC'15], FxMark [ATC'16], ScaleFS [SOSP'17], FLEX [ASPLOS'19] …

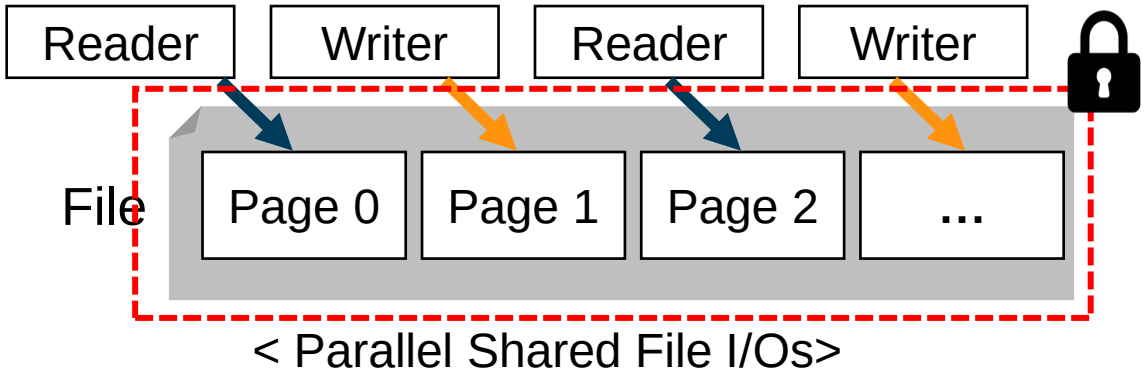
***Mainly due to..***

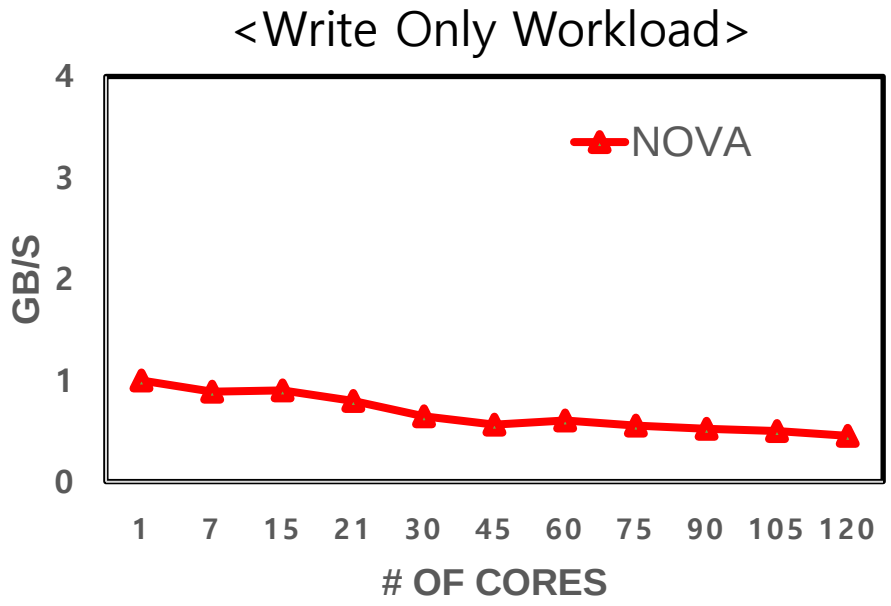
- ***File System Lock***
- ***Non-scalable File System Data Structure***

# Motivation

NOVA, a representative NVM file system, also has a scalability bottleneck.

- The performance does not scale when parallel shared file I/Os perform, due to
- coarse-grained Readers-Writer(RW) lock on inode.




*Preliminary result based on NVM emulation on DRAM [APSYS'19]*

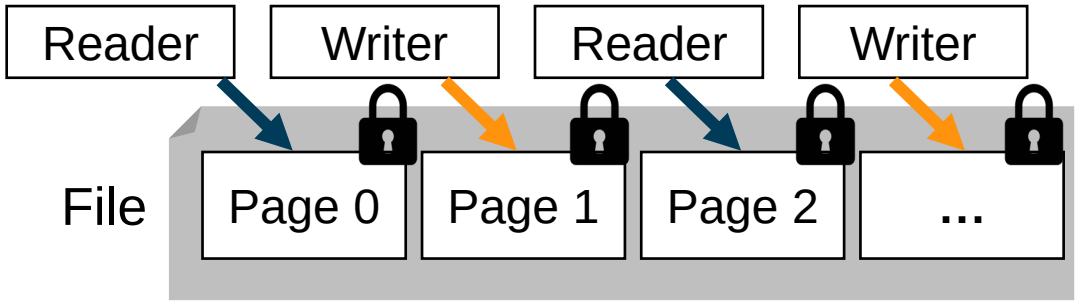
# Motivation

NOVA, a representative NVM file system, also has a scalability bottleneck.

- The performance does not scale when parallel shared file I/Os perform, due to
- coarse-grained Readers-Writer(RW) lock on inode.

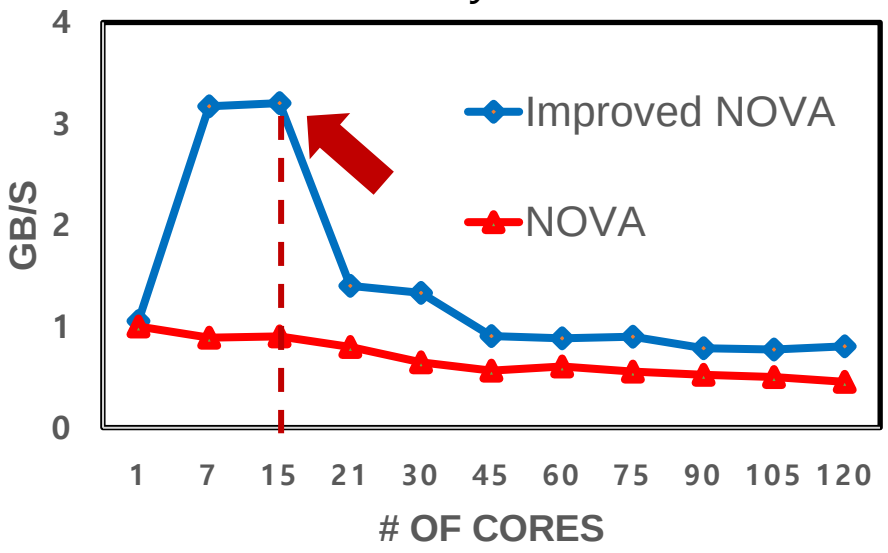
In our previous work, a fine-grained range-based RW lock was proposed and applied to NOVA.

- This solution increases the max I/O throughput, but still does not scale after 15 cores(NUMA boundary).




< Parallel Shared File I/Os>

<Write Only Workload>




*Preliminary result based on NVM emulation on DRAM [APSYS'19]*

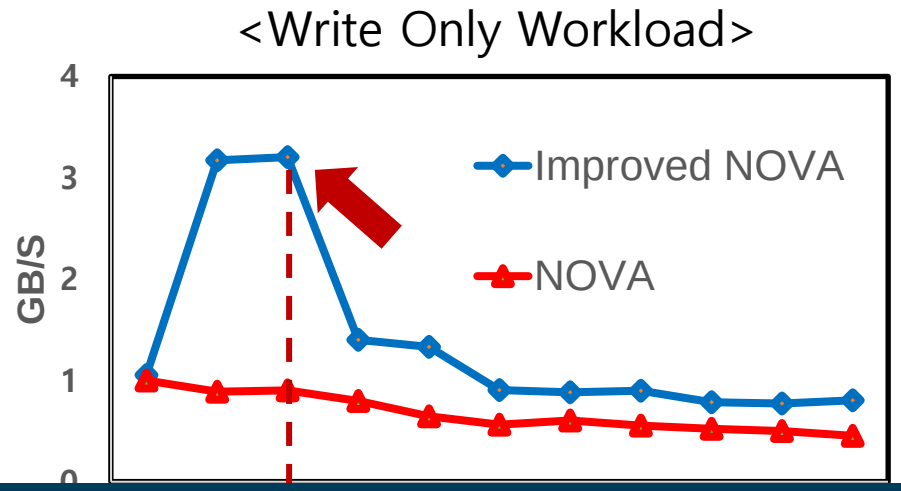
# Motivation

NOVA, a representative NVM file system, also has a scalability bottleneck.

- The performance does not scale when parallel shared file I/Os perform, due to
- coarse-grained Readers-Writer(RW) lock on inode.

In our previous work, a fine-grained range-based RW lock was proposed and applied to NOVA.

- This solution increases the max I/O throughput, but still does not scale after 15 cores(NUMA boundary).




***The fine-grained lock solution does not scale on Non-Uniform Memory Access (NUMA) architecture.***

***<u>This is because NOVA is not design for NUMA.</u>***

# Content

- Introduction and Motivation
- **Background and Problem Definition**
  - NVM File System: NOVA
  - NVM File System’s Scalability Limitation on NUMA Environment
- NUMA-aware NVM File System Design
- Evaluation Result
- Conclusion

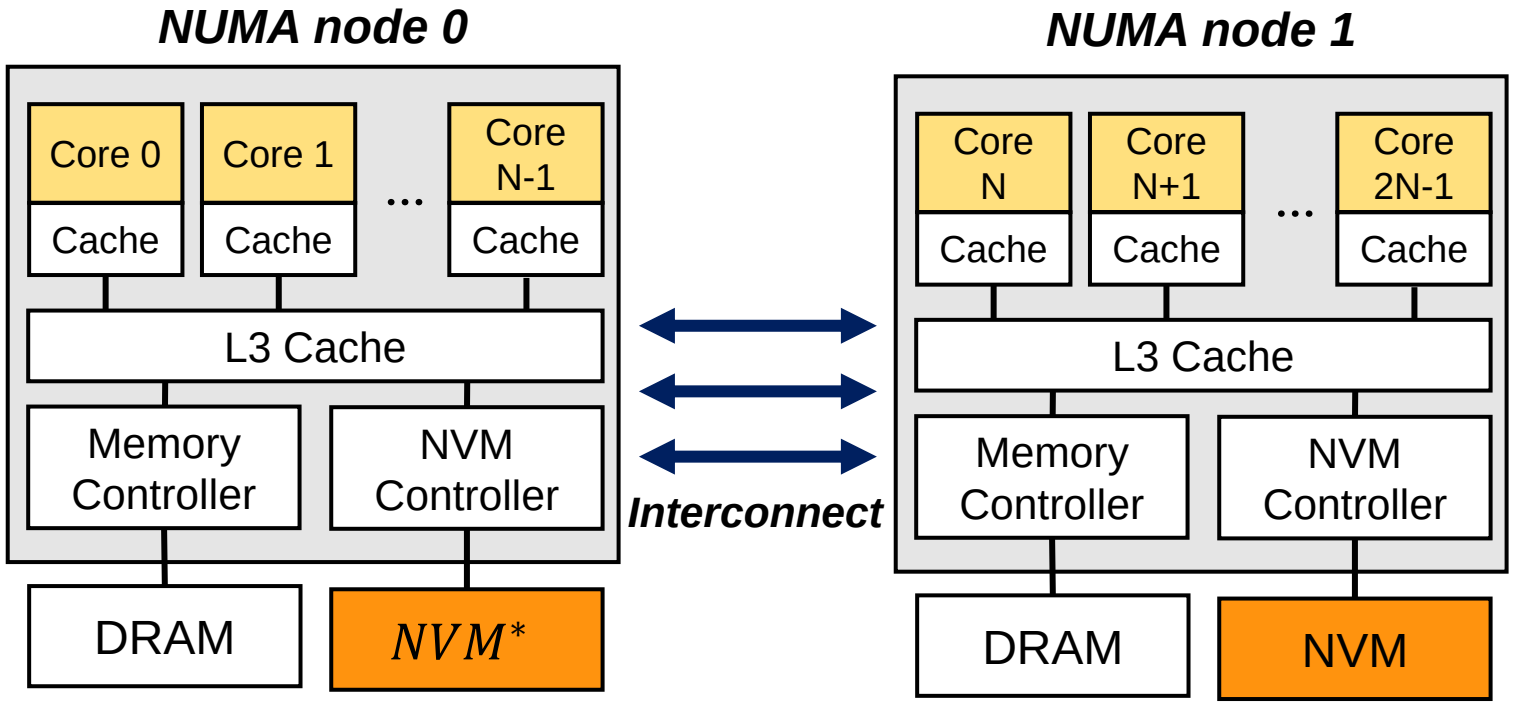
# Non-Uniform Memory Access (NUMA) architecture

Manycore servers are mostly based on NUMA architecture.

- Local memory(DRAM or NVM) access is faster than remote memory access.

System applications need to be aware of the NUMA architecture.

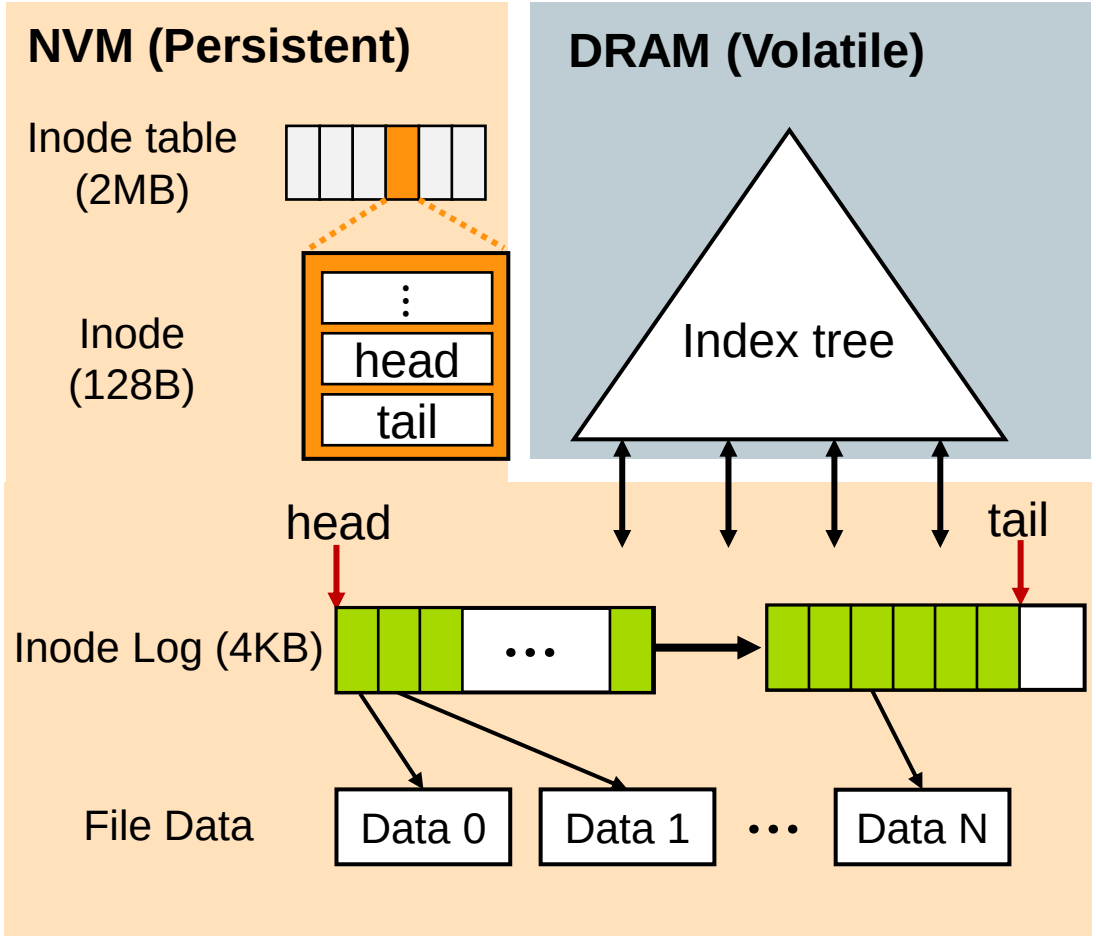
- But, existing NVM file systems are unaware of the NUMA architecture.
- Ex) NOVA place all file data and metadata on a single NUMA node.




* In 2019, Intel Optane Direct Connect (Optane DC) Persistent Memory became commercially available.

# NVM File System: NOVA [FAST'16]

- NOVA is a well-designed NVM file system.
- Log-structure design to guarantee consistency
  - Per-inode logging for high concurrency
  - Linked-list log only contains metadata
  - Index tree in DRAM to quickly search corresponding log entry

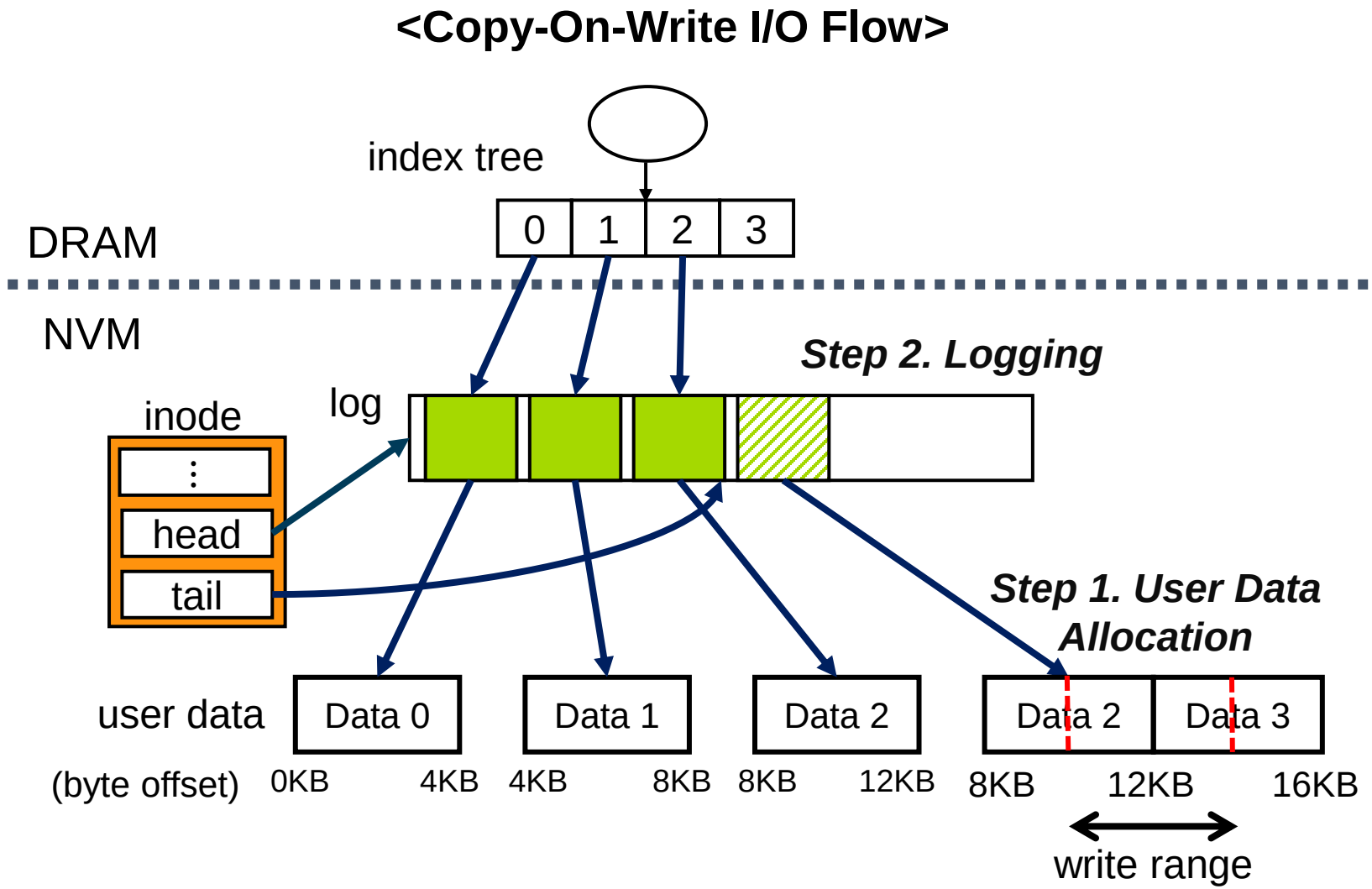


# Example: Write I/O Flow in NOVA

Write 4KB (10KB~14KB):

```
int fd;
char buf [4096];
…
fd=open(foo.txt);

pwrite(fd, buf, 4096, 10240)
```

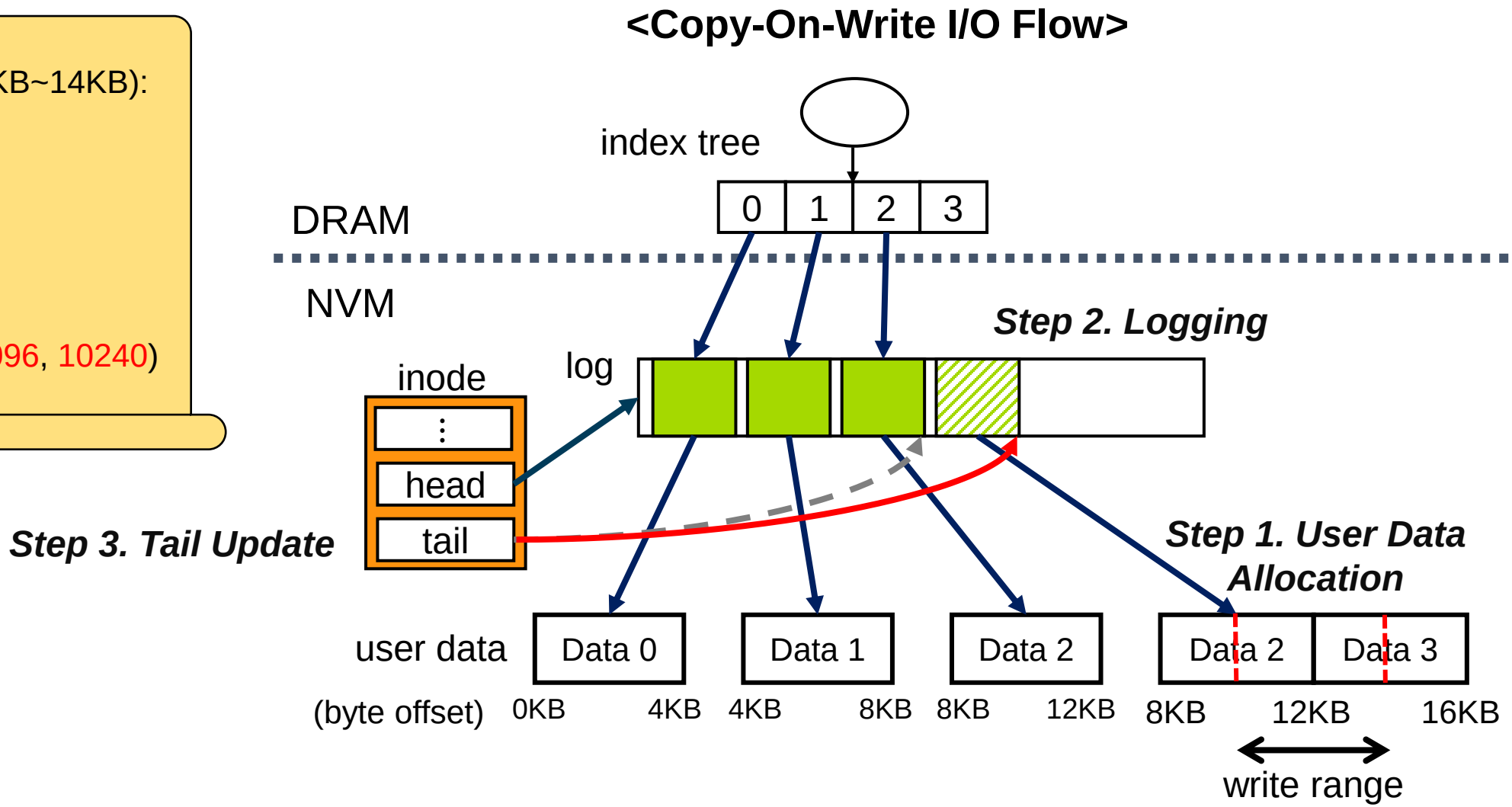


# Example: Write I/O Flow in NOVA

Write 4KB (10KB~14KB):

```
int fd;
char buf [4096];
…
fd=open(foo.txt);

pwrite(fd, buf, 4096, 10240)
```

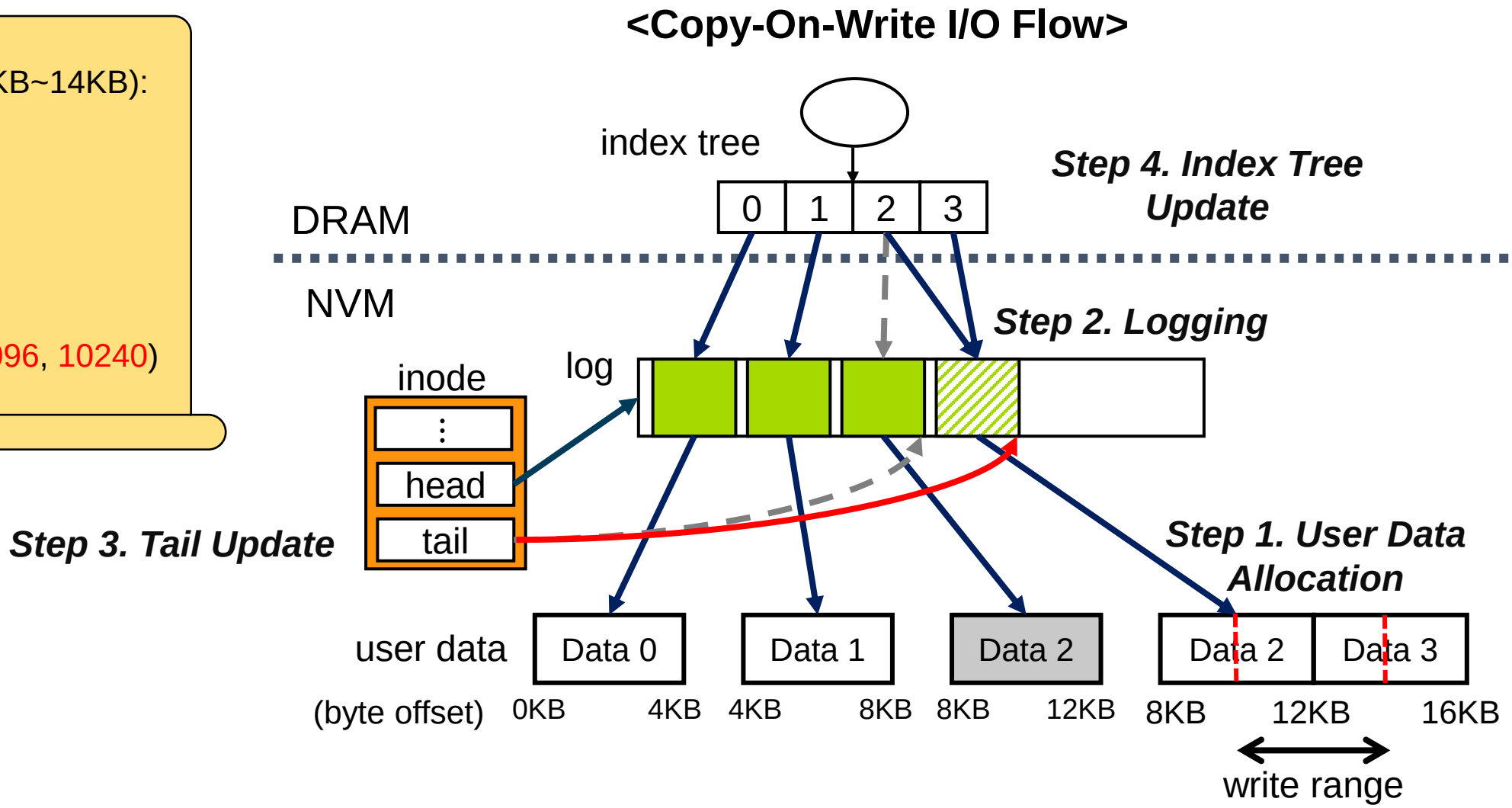


# Example: Write I/O Flow in NOVA

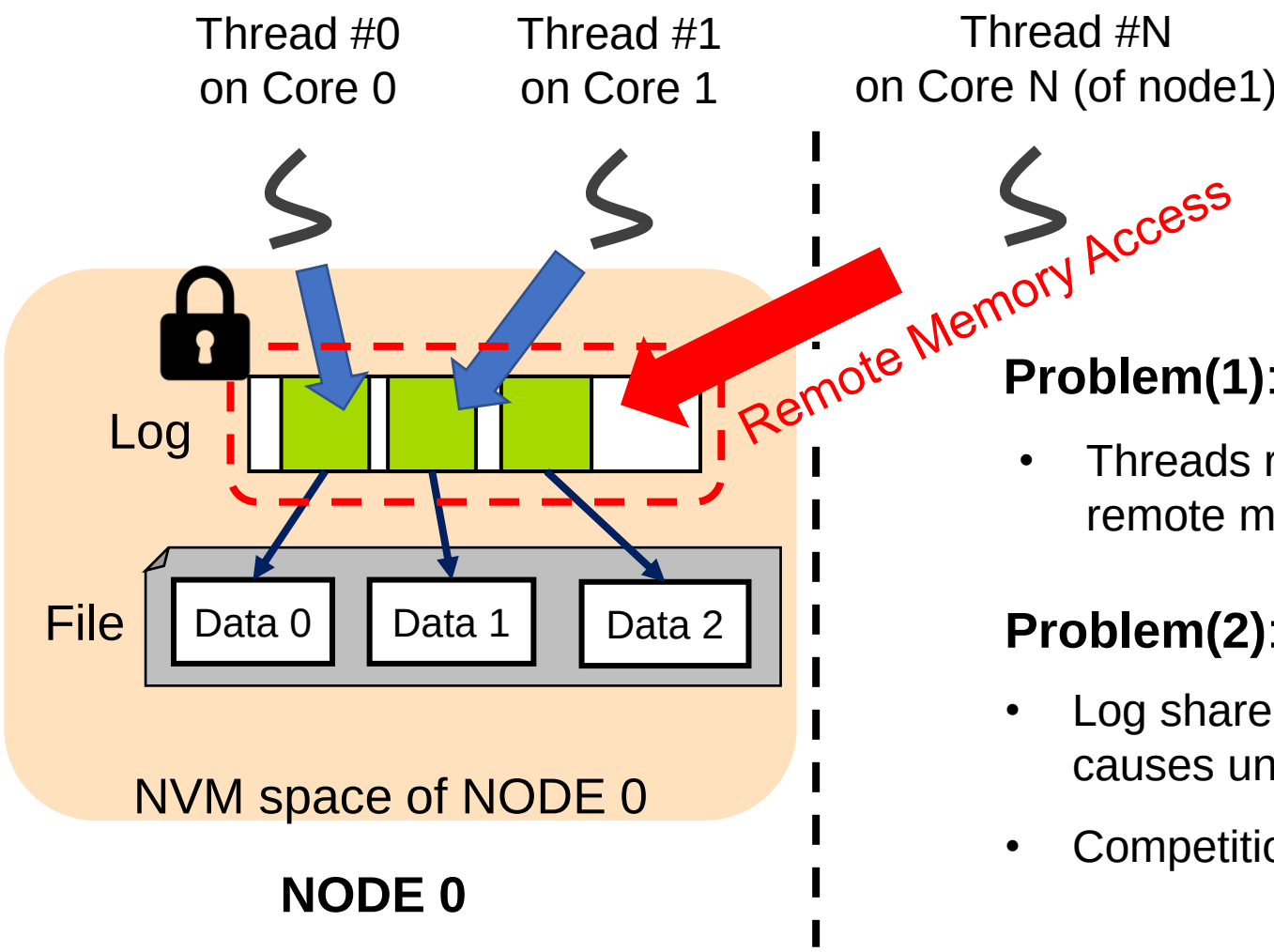
Write 4KB (10KB~14KB):

```
int fd;
char buf [4096];
…
fd=open(foo.txt);

pwrite(fd, buf, 4096, 10240)
```



# NOVA Scalability Limitation on NUMA Environment



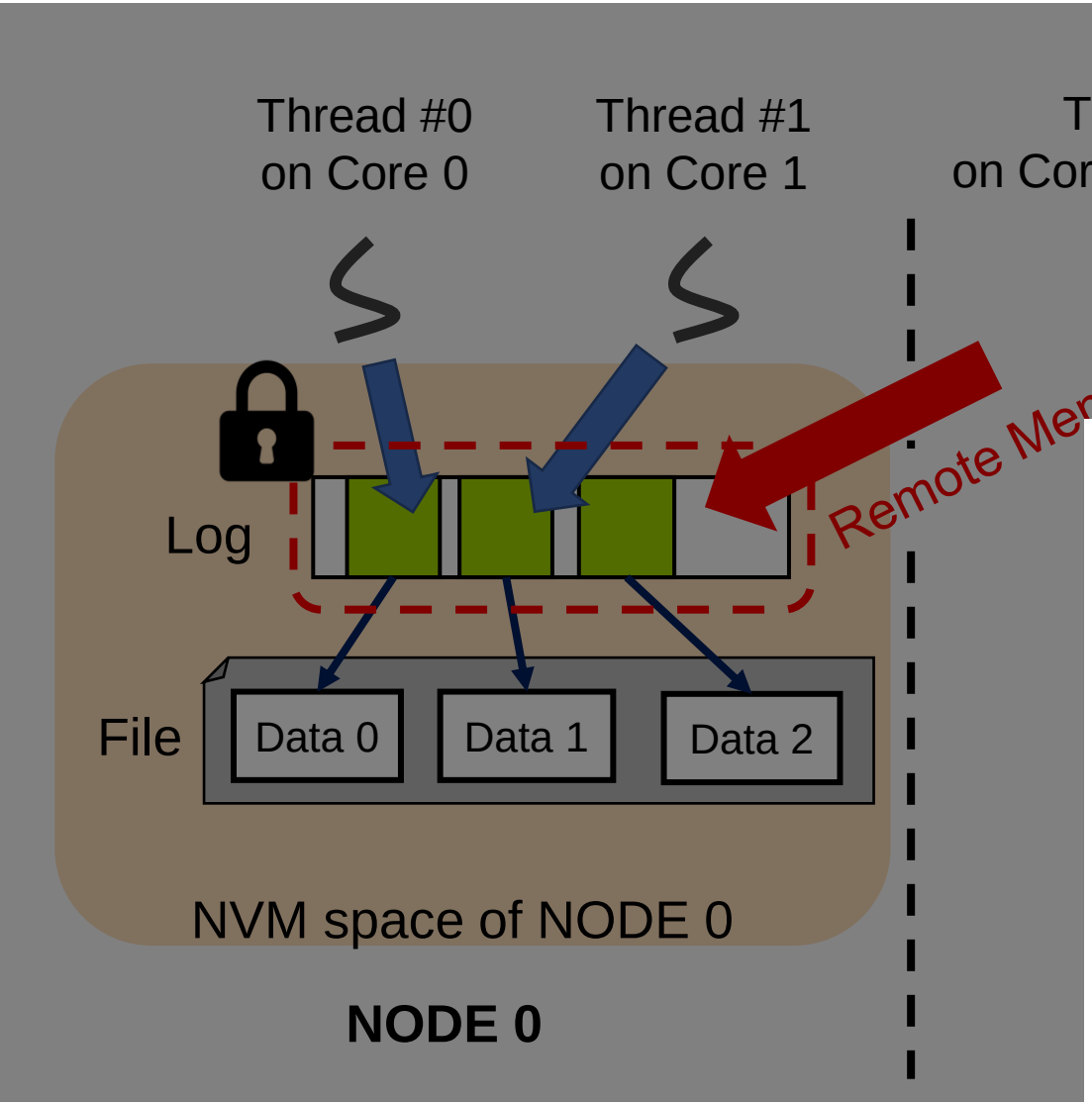

**Problem(1)**: Initial data placement on a single NUMA node

- Threads running on other nodes must access the file via remote memory access.

**Problem(2)**: Shared data structure between different nodes

- Log shared among threads running on different nodes causes unavoidable remote memory access.
- Competition for the lock leads to huge performance loss.

# Our Approaches




**Problem(1)**: Initial data placement on a single NUMA node

**IDEA(1):** NUMA-aware data placement

**Problem(2)**: Shared data structure between different nodes

**IDEA(2):** Lock-free per-core data structure

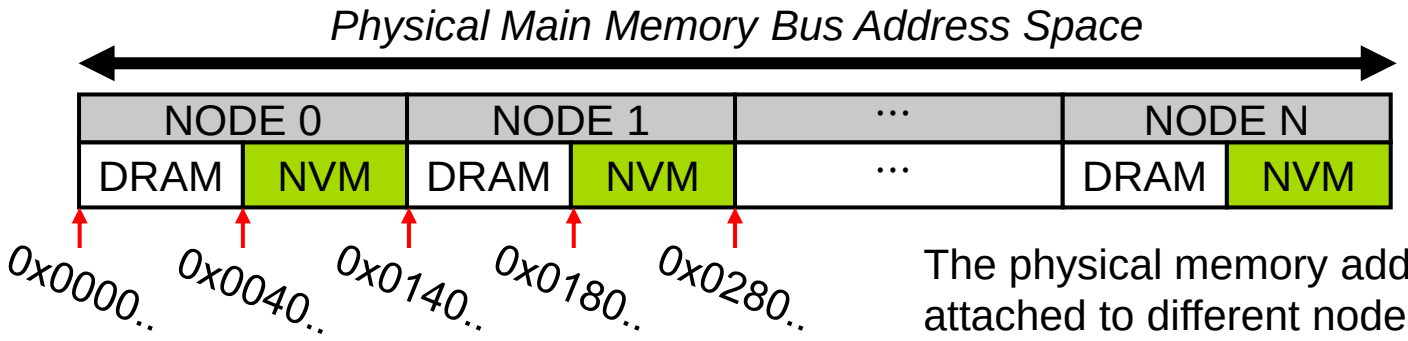
# Content

- Introduction and Motivation
- Background and Problem Definition
- **NUMA-aware NVM File System Design**
  - IDEA (1): NUMA-aware Data Placement
  - IDEA (2): Lock-Free Per-Core Data Structure
- Evaluation Result
- Conclusion

# IDEA(1): Virtualizing NVM Devices

The NUMA-aware file system can reduce remote memory accesses by distributing files over multiple NUMA nodes.

- First of all, we have virtualized the non-contiguous NVM address space from multiple NUMA nodes to a single contiguous virtual address space.

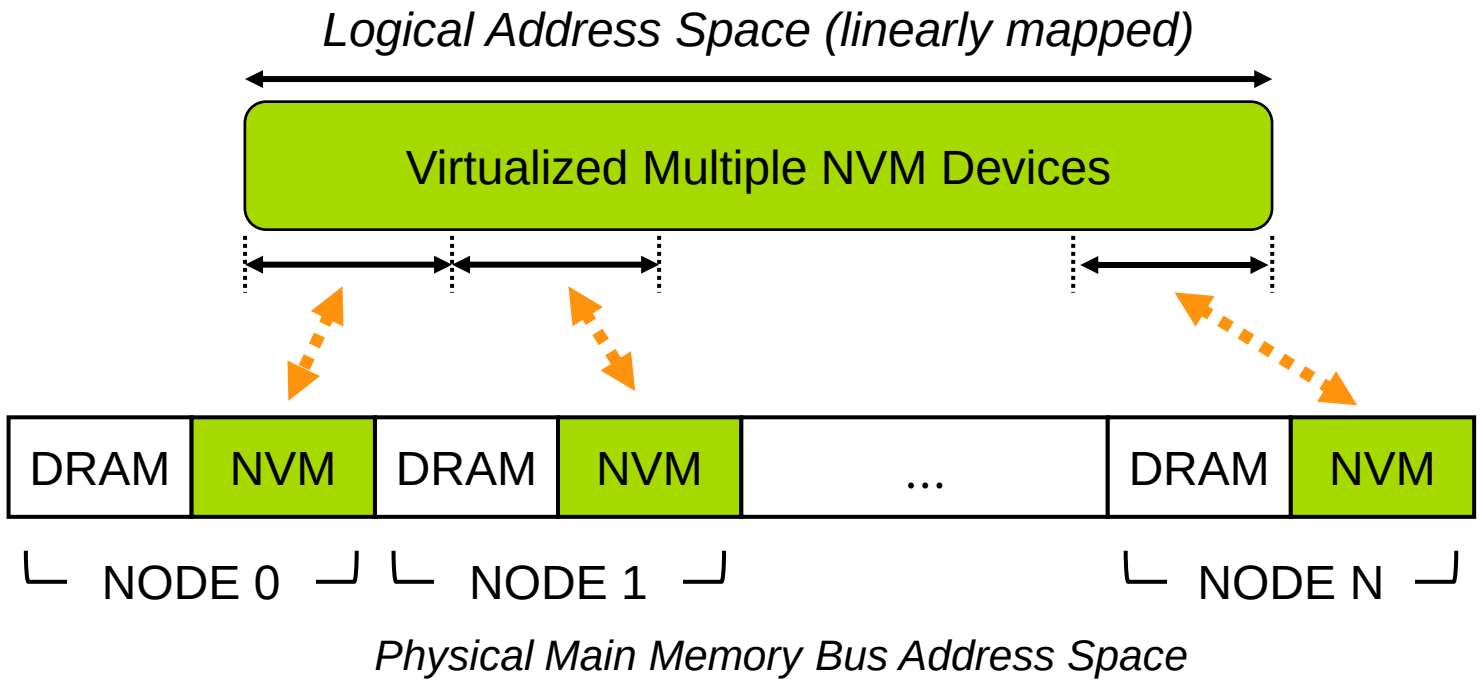



The physical memory address spaces of NVM devices attached to different nodes are not contiguous.

# IDEA(1): Virtualizing NVM Devices

The NUMA-aware file system can reduce remote memory accesses by distributing files over multiple NUMA nodes.

- First of all, we have virtualized the non-contiguous NVM address space from multiple NUMA nodes to a single contiguous virtual address space.



# IDEA(1): Memory Allocation Policy "*Local Write First*"

- To mitigate the remote memory access, we adopt a "*Local Write First*" policy, where threads give preference to write files to a local NVM device.

  1. An entire logical address space is divided into the number of CPU cores.

  2. Data and log pages for each thread are allocated from the region allocated to the core on which the thread is running.

  ***Each I/O thread allocates data or log pages in local NVM.***

# IDEA(2): Lock-Free Per-Core Data Structure

File data structures shared among various threads become a major cause of scalability bottleneck for the file system in a NUMA-based system.

We propose per-core data structures for scalable file system.

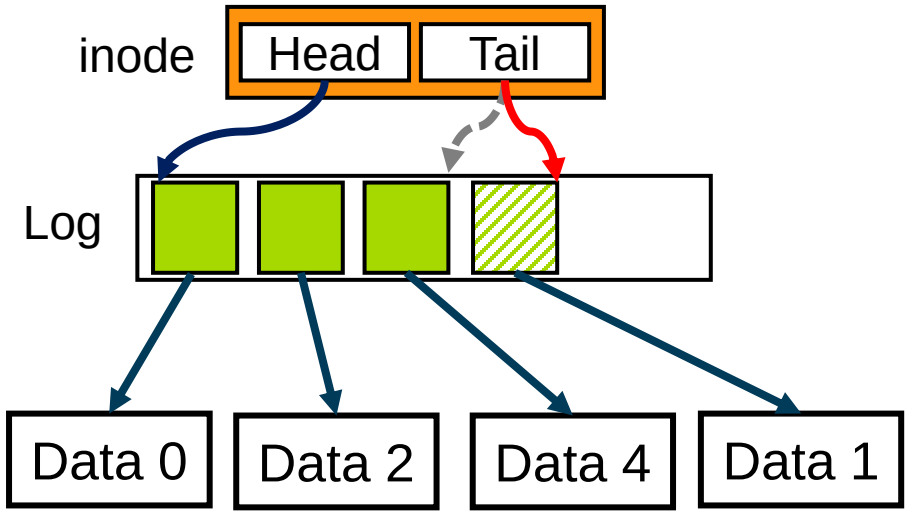
- We extend the NOVA's per-inode log structure to be a lock-free per-core log.

# IDEA(2): Per-Core Log Structure in NOVA

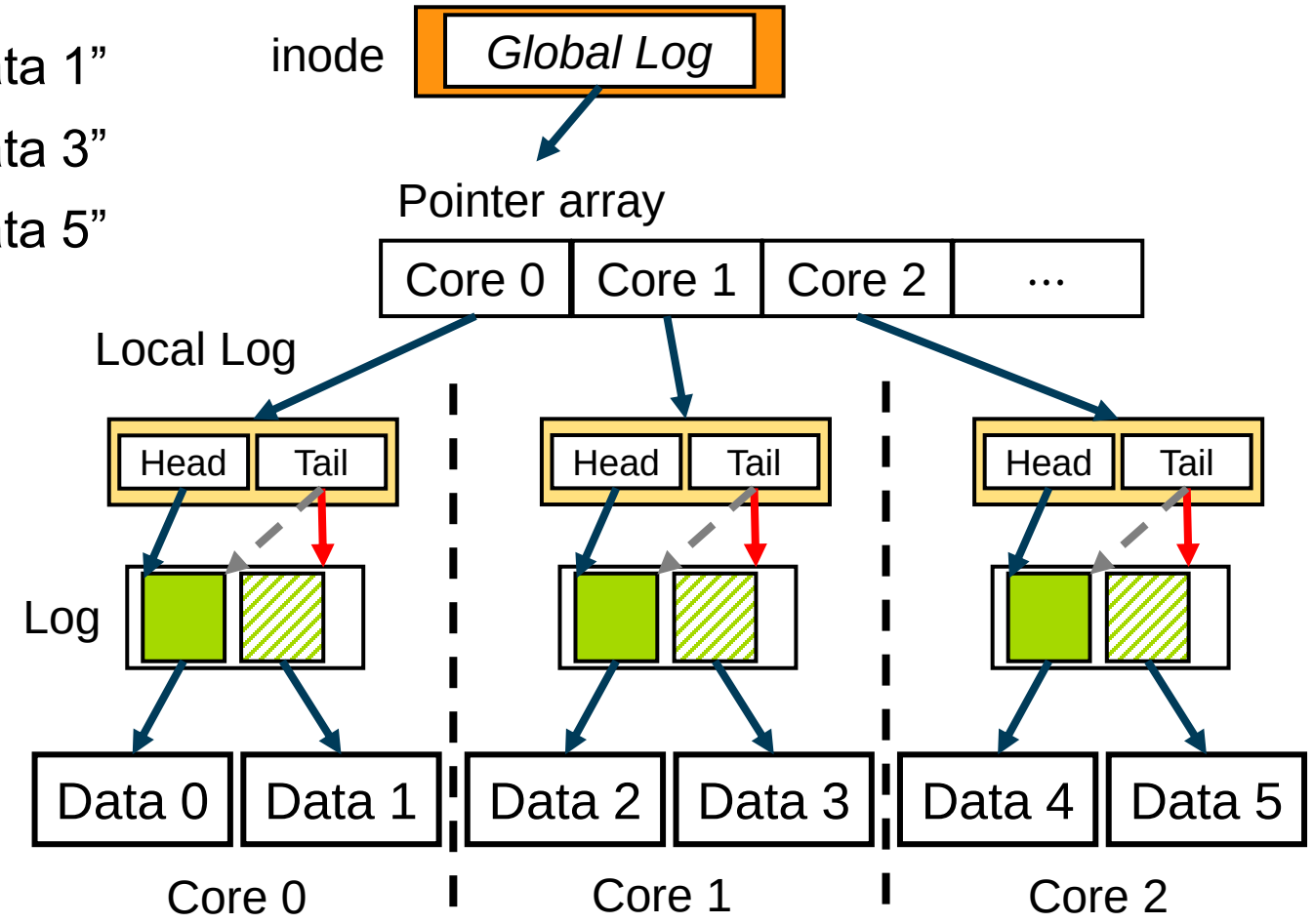
*thread* #0 on core 0: Write “data 0” then “data 1”

*thread* #1 on core 1: Write “data 2” then “data 3”

*thread* #2 on core 2: Write “data 4” then “data 5”




Per-inode Log structure in NOVA




Per-Core Log structure

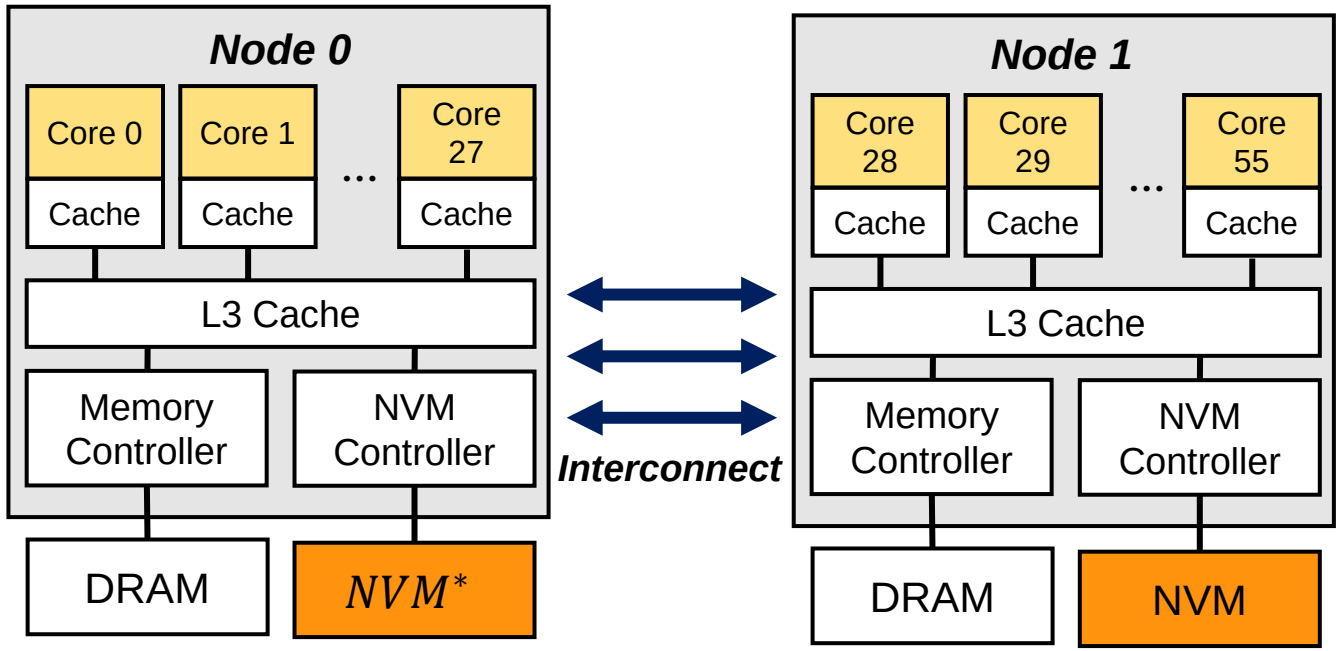
# Content

- Introduction and Motivation
- Background and Problem Definition
- NUMA-aware NVM File System Design
- **Evaluation**
- **Conclusion**

# Performance Evaluation

- Testbed

| | | |
|---|---|---|
| CPU | | Intel(R) Xeon(R) Platinum 8280M v2 2.70GHz<br>CPU Nodes (#): 2, Cores per Node (#): 28 |
| Memory | | DRAMs per Node (#): 6, DDR4, 64 GB * 12 (=768GB) |
| NVM | | Intel Optane DC Persistent Memory<br>NVMs per Node (#): 6, 128 GB * 12 (=1.5TB) |




<Testbed NUMA-architecture>

# Performance Evaluation

- Comparing Schemes

| Scheme | | Description |
|---|---|---|
| NOVA(V) | | Vanilla NOVA. |
| NOVA(FL) | | NOVA using a fine-grained lock. |
| NOVA(FL+NUMA) | | NOVA using a fine-grained lock with NUMA-aware design. |

- Parallel I/O workloads in FxMark benchmark

This talk contains only the result of this workload.

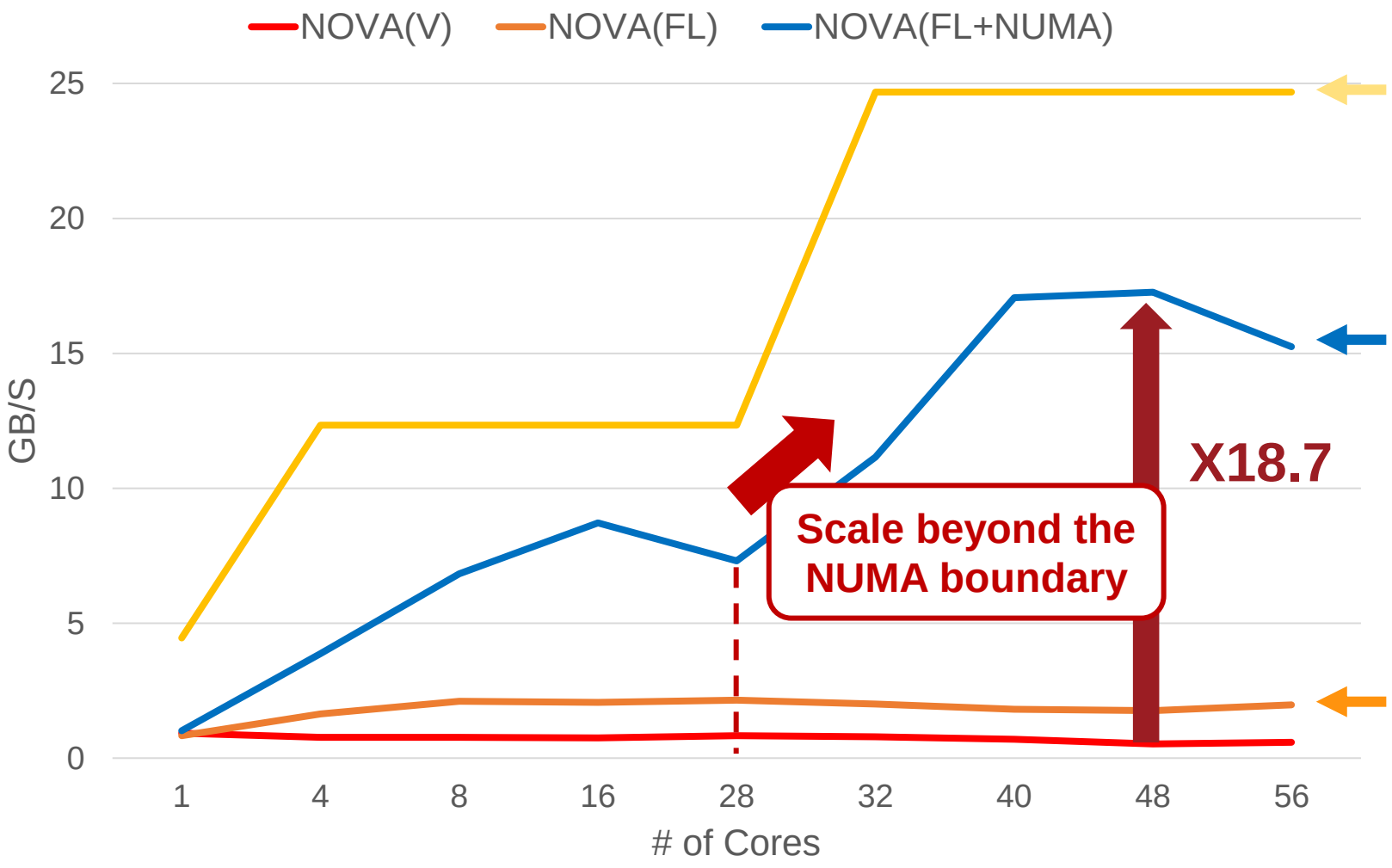
**Shared File Write (N-to-1 write):** Multiple threads write private regions on a single file.

**Private File Write (N-to-N write):** Multiple threads write to their private files.

**Private File Read (N-to-N read):** Multiple threads read the files after writing the files.

FxMark[ATC'16]: C. Min. et. al., "Understanding Manycore Scalability of File Systems", USENIX ATC 2016

# Evaluation: Shared File Write (N-to-1 Write)

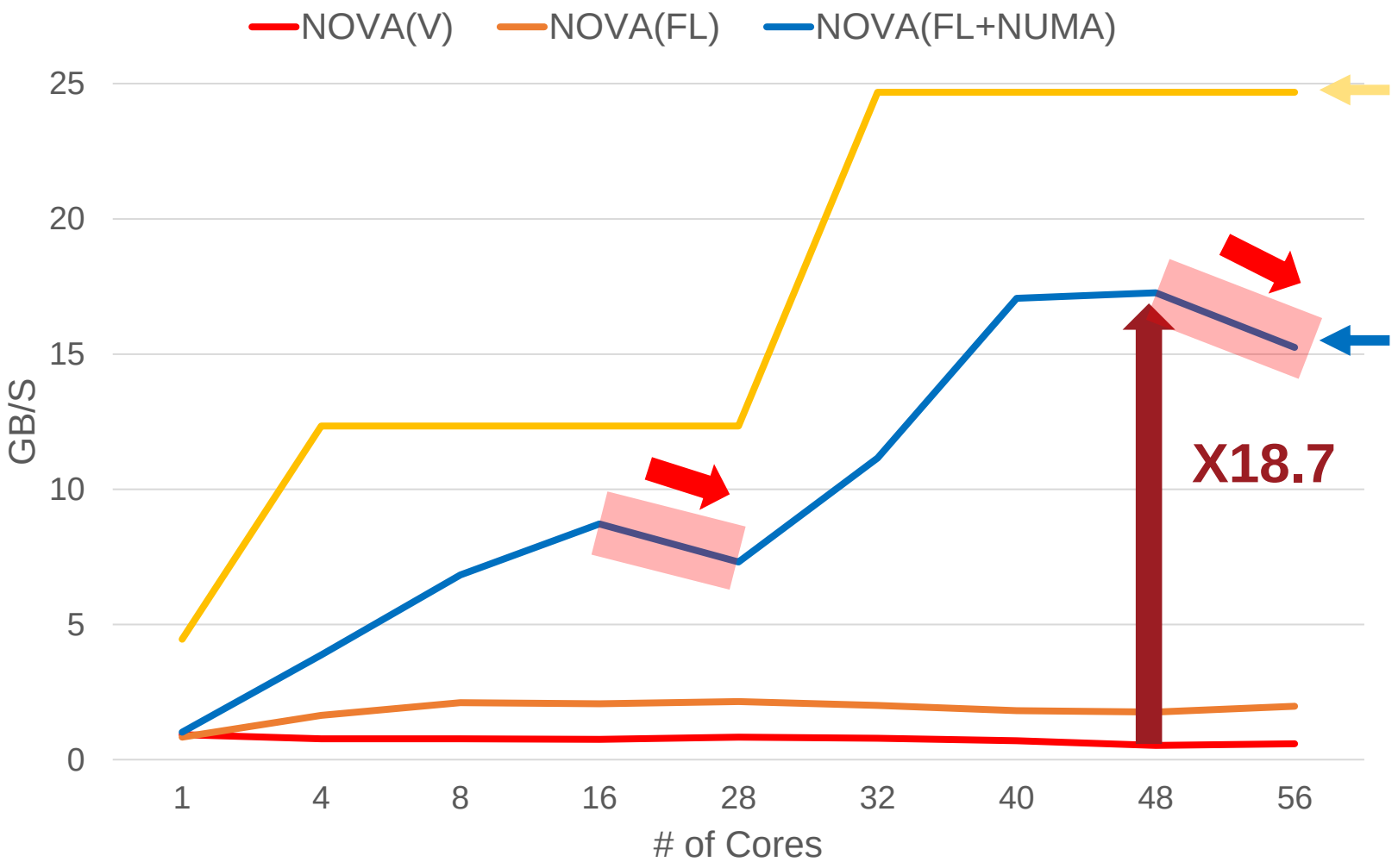



NVM Max Bandwidth (ideal)

- The “local write first“ policy and lock-free data structure improved the scalability.

Fine-grained lock slightly improved performance but high bandwidth loss.

# Evaluation: Shared File Write (N-to-1 Write)




NVM Max Bandwidth (ideal)

- The “local write first“ policy and lock-free data structure improved the scalability.
- We suspect the Optane DC memory controller be highly congested.

# Conclusion

- NVM file system need to be aware of the NUMA architecture for scalability.

- We proposed a NUMA-aware NVM file system design.
    1. Virtualized the NVM modules of several NUMA nodes
    2. “Local Write First” Memory allocation policy to reduce remote memory access
    3. Lock-free per-core file system data structure

# Thank you!
# QnA


- **June-Hyung Kim**
- junehyung@sogang.ac.kr
- Sogang University, South Korea