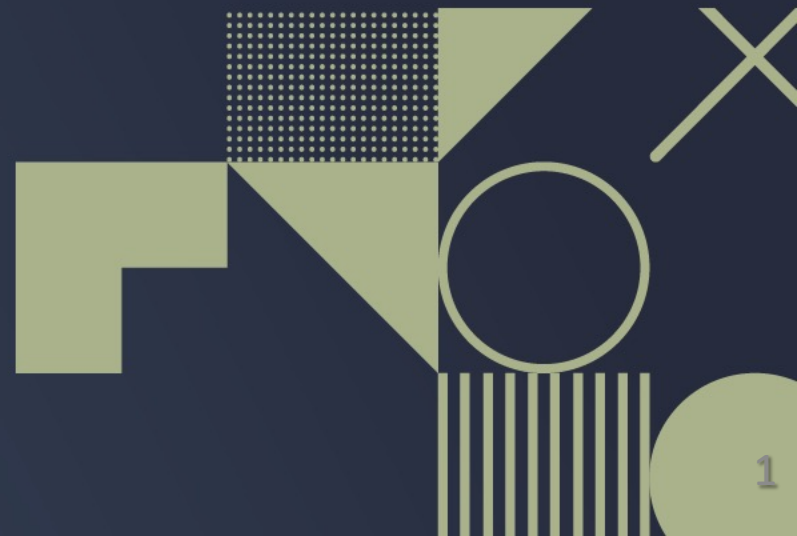
Simurgh: A fully Decentralized and Secure NVMM User Space File System

Nafiseh Moti, Fredric Schimmelpfennig, Reza Salkhordeh, David Klopp, Toni Cortes, Ulrich Rückert, André Brinkmann



Non Volatile Main Memory (NVMM)

- Persistent
  - ➢ Recoverable
- Byte-addressable
  - ➢ Pointers
- Fast
  - ➢ Efficient local storage

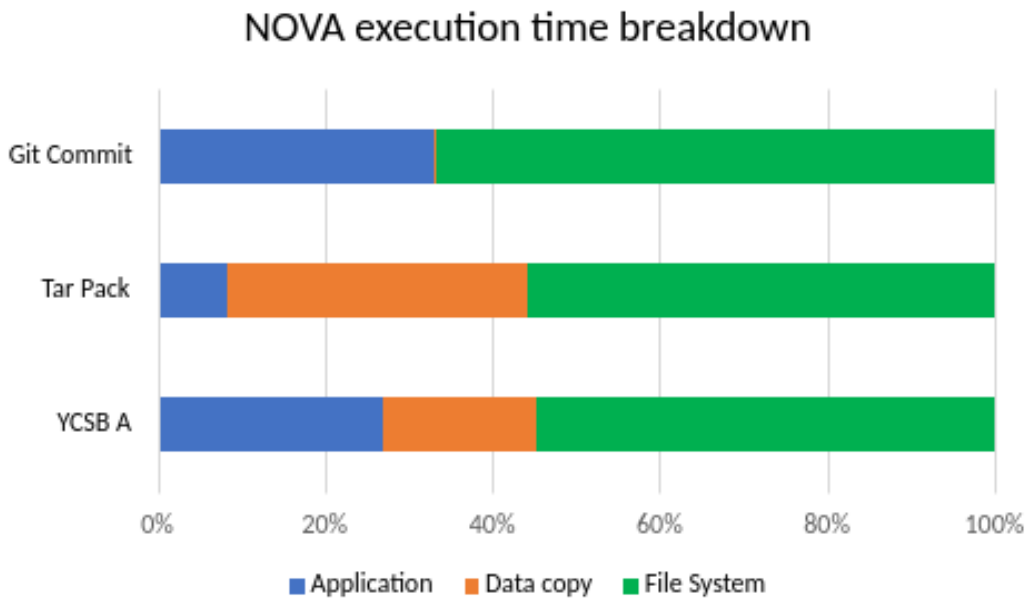
NVMM in HPC

- Large capacity, low power consumption
- Backend:
  - Classical distributed file systems
  - Burst buffers
- Disaggregated memory, RDMA

NVMM File Systems

Kernel level file systems:

- PMFS [EuroSys '14]
- NOVA [FAST '16]
- EXT4-DAX



OS not made for NVMM's:

- Extra copies
- Unnecessary journals/logs
- System call overhead
- VFS overhead
- Locks, poor scalability

NVMM File Systems

User level file systems:

- Hybrid
  - Arrakis [OSDI '14], Strata [SOSP '17], SplitFS [SOSP ‘19]

  - Metadata handled by kernel
  - VFS bottleneck

- Fully user level
  - Direct updates: ZoFS [SOSP '19]
  - Indirect updates: KucoFS [FAST '21]

  - Coarse-grained permission handled by kernel

Our Solution

- Decentralized, user level local **file system library**
  - For both data and metadata
  - Applications map the complete file system in their own address space

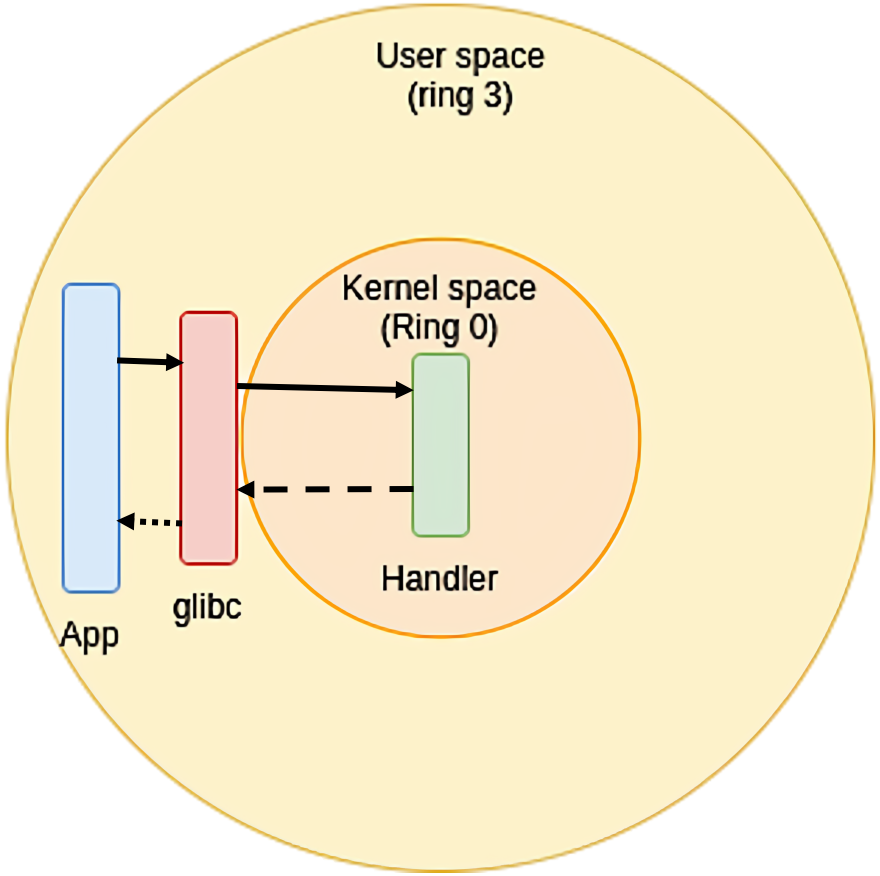
- Fine grained, light-weight **user level security mechanism**
  - Introducing protected user space functions
  - Proposing two new instructions
  - ➢ Similar guarantees as kernel level file systems

Protected User Level Functions

- **Security goal:** Provide a **trusted user level** execution
  - File system data protection
  - File system code isolation
  - Safe privilege transition

Traditional Kernel Secutity

- System call view:
  1. Glibc routine call
  2. Syscall
  3. Syscall return
  4. Glibc wrapper return

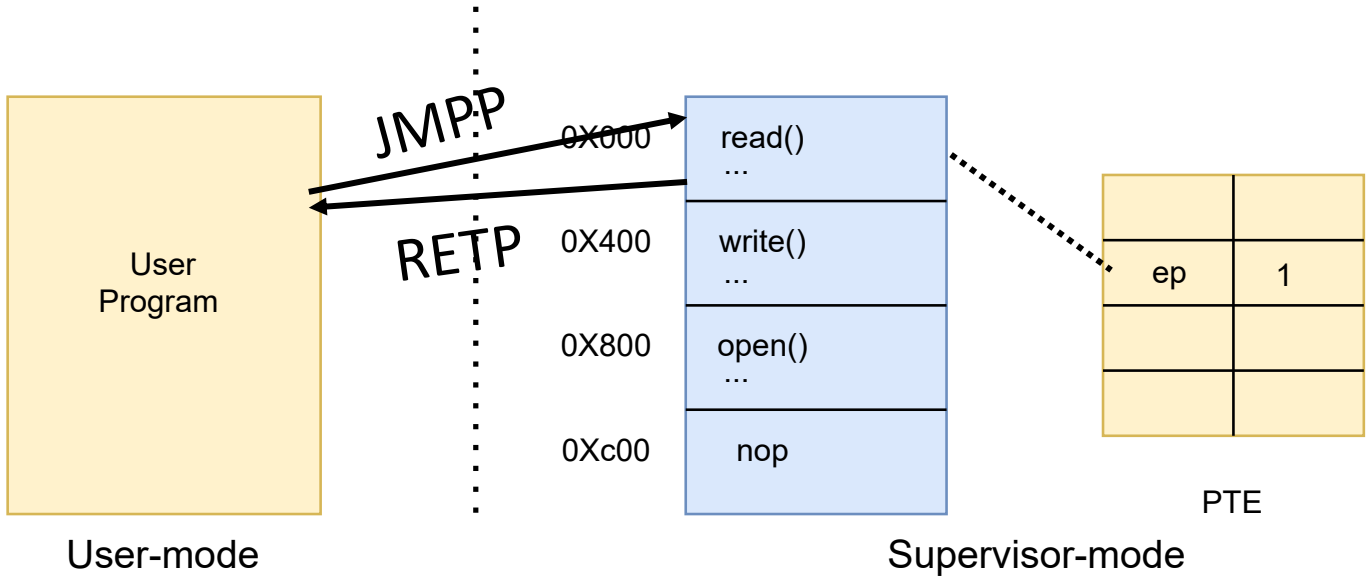


JMPP/RETP

- PTE protection bit
  - Code within protected pages execute in higher privilege mode
- JMP *protected* / RET *protected*
  - Safe transition

    user code -> trusted code

    trusted code -> user code

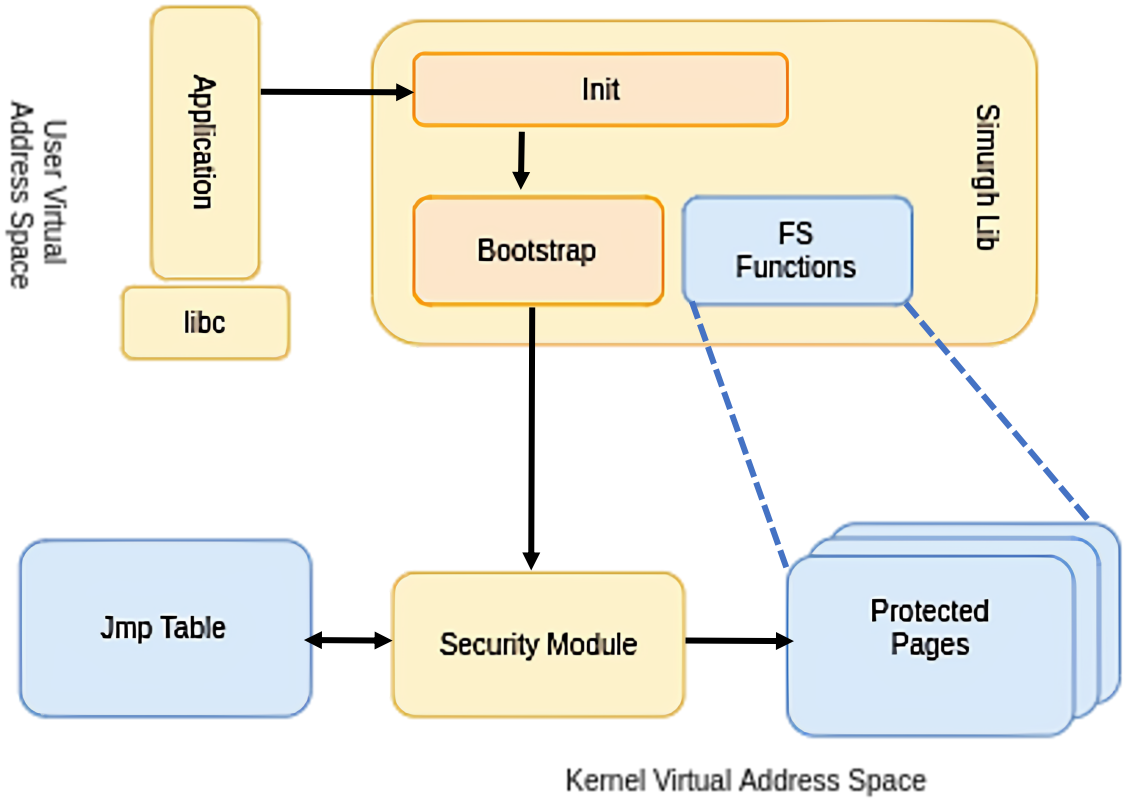


- Crucial functions are executed within in a supervisor mode ring
  - No manipulations of FS code

User Level File System Security

Setup of protected functions:

- Only kernel can set *protection bit*
  - ➢ One time bootstrapping process
- General purpose bootstrap module



Security Evaluation

Goals:

- Test the functionality of JMPP/RETP
- Comparison to the standard function call

Gem5 implementation:

- Combined overhead of ~70 cycles for JMPP/RETP
- Overhead consists of changing ring, return address writing, checking the protected bit, and returning from the call

➢**The same order of magnitude as normal function call**

Simurgh File System Library

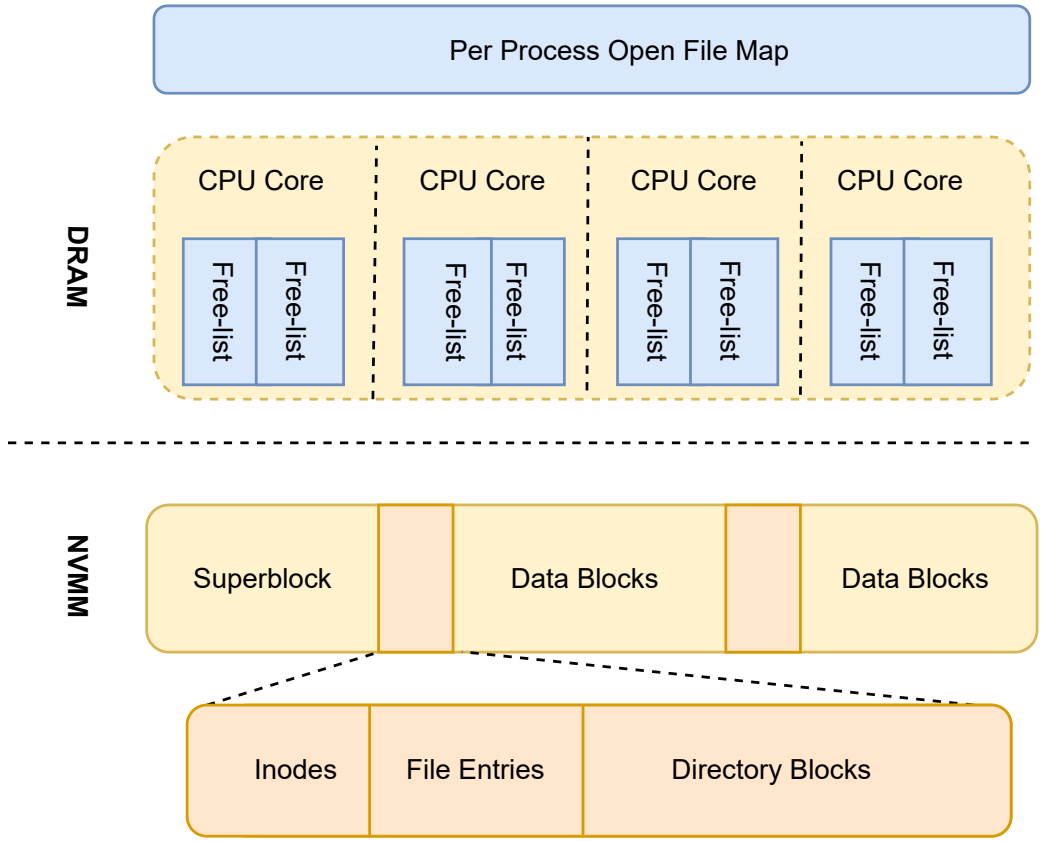
- File System Design **Goals**:
  - **General purpose**, drop in solution
  - Persist and share from **independent processes**
  - POSIX API, **no application changes needed**
  - **Scalability and efficiency** in data and meta data operations
  - **Protection** equal to kernel file system

High Level Design

- Interposing library (LD_PRELOAD)
  - Serving glibc calls without kernel
- NVMM file system mapped into application
  - Shared NVMM (persistent data)
  - Shared DRAM (runtime data structures)

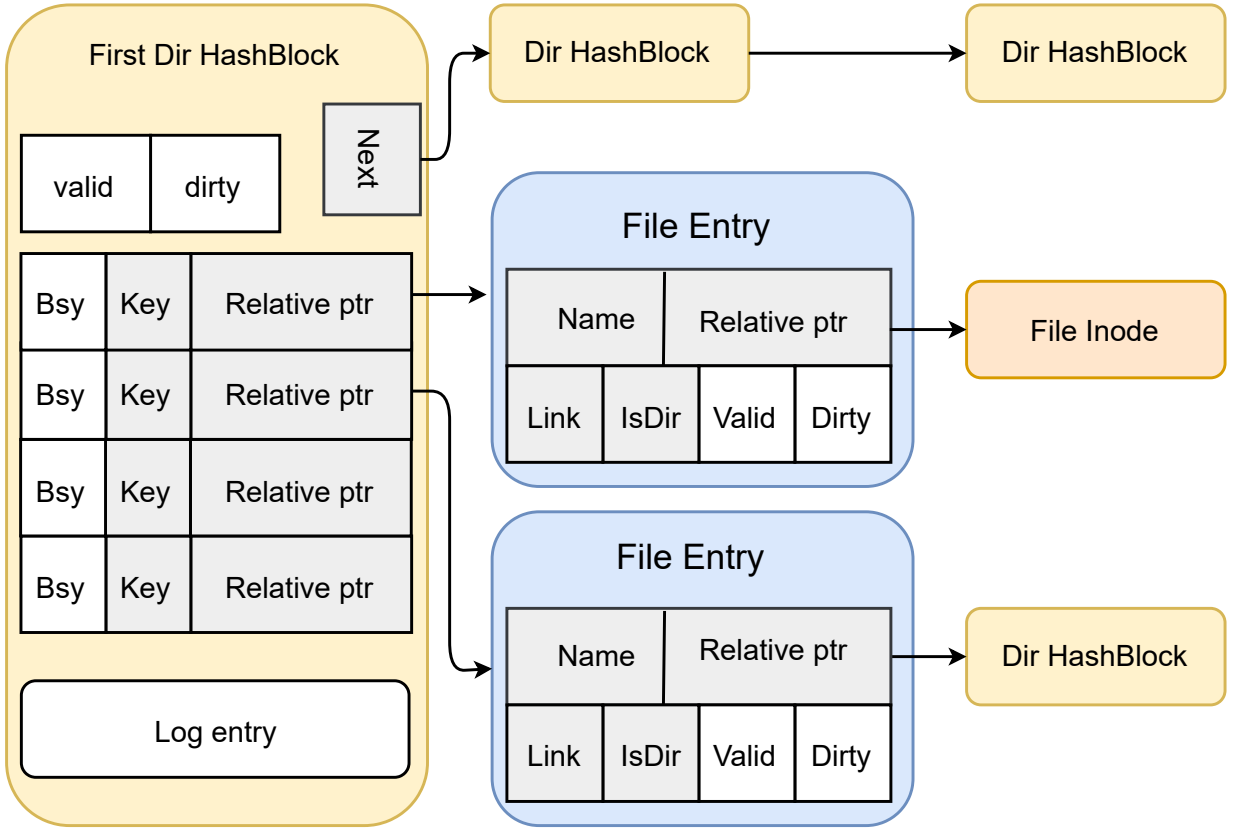
Specific implementations:

- Persistent pointers
- Separate allocators:
  - Data blocks
  - Meta data objects
- …



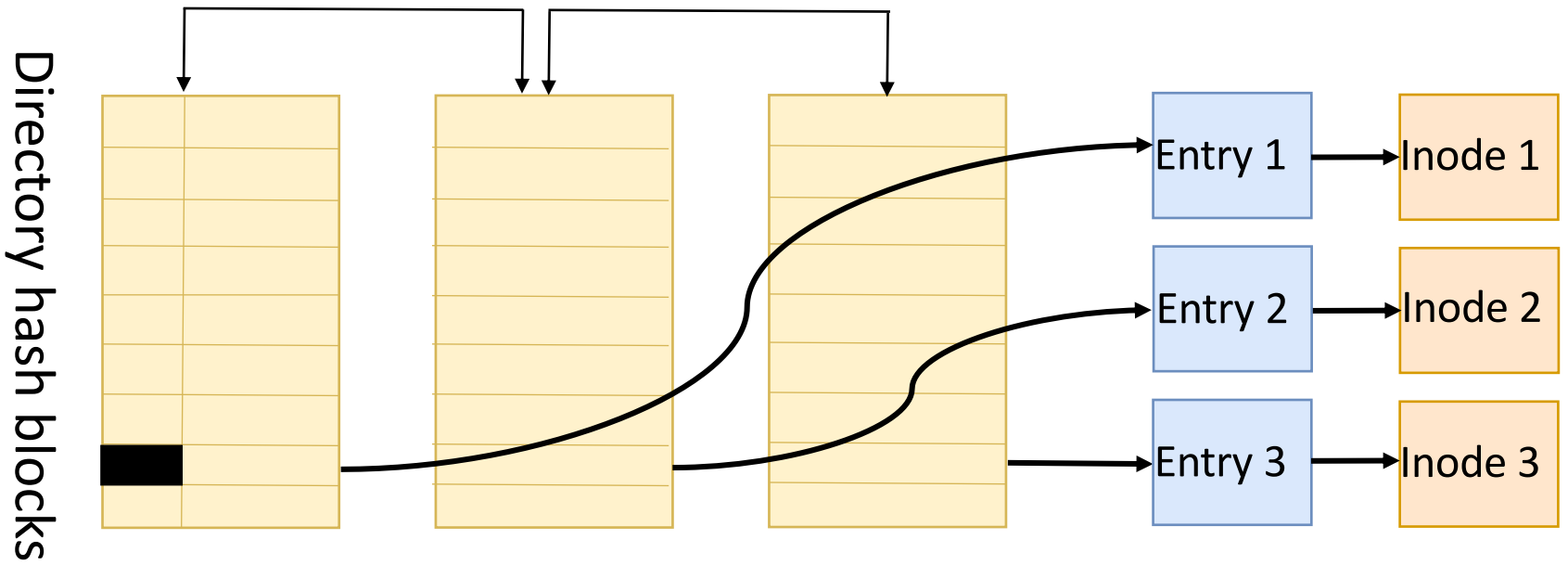
Directory Structure

- Challenges:
  - consistency
  - scalability
  - NVMM properties
- Hash based directory structure
- No metadata cache
- Fixed size objects
- All time consistent:
  - valid bits, dirty bits, connections



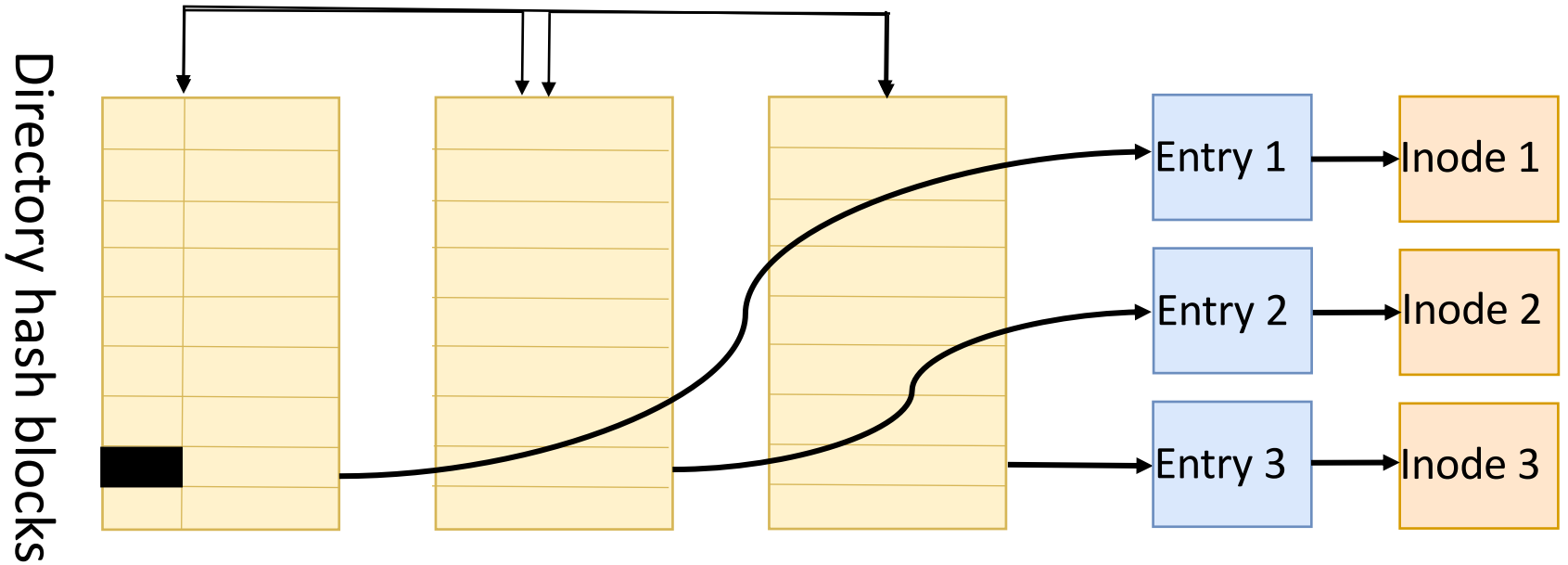
File Creation

Creating file nr. 3
on the same *hash line*



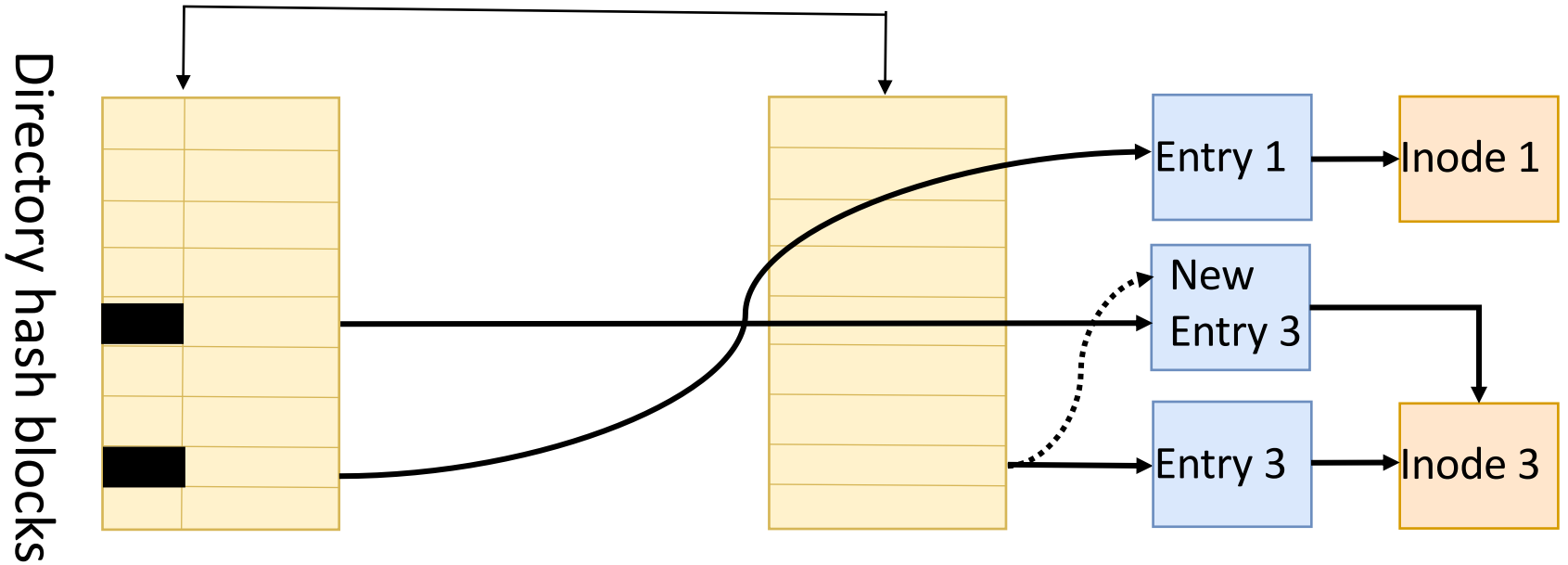
File Deletion

Deleting file nr. 2



File Rename

Renaming file nr. 3



Evaluation

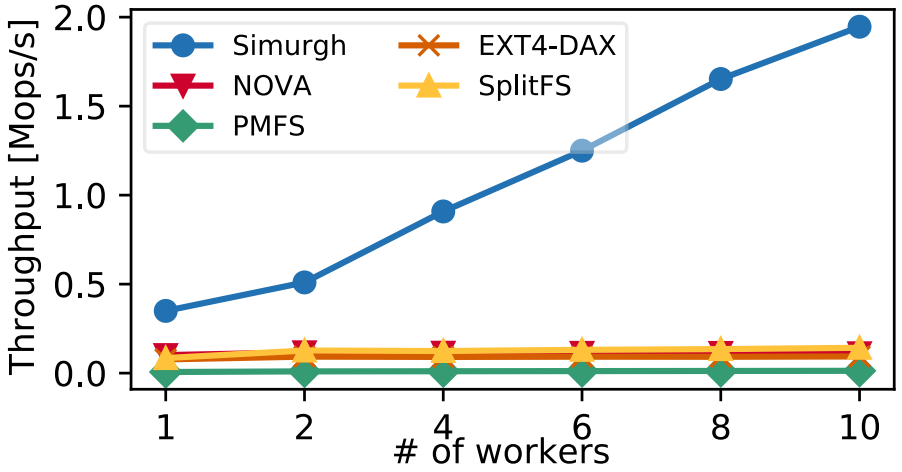
Setup:

- 10 core Xeon Gold 5212, 2.5GHz
- 192 GByte DRAM
- 746 GByte Intel Optane DC persistent memory, 6 DIMMs
- Linux kernel 4.18 on CentOS 8.2
- File Systems: Simurgh, NOVA, PMFS, EXT4-DAX, SplitFS
- Benchmarks: FXMark, Filebench, YCSN, Tar, Git

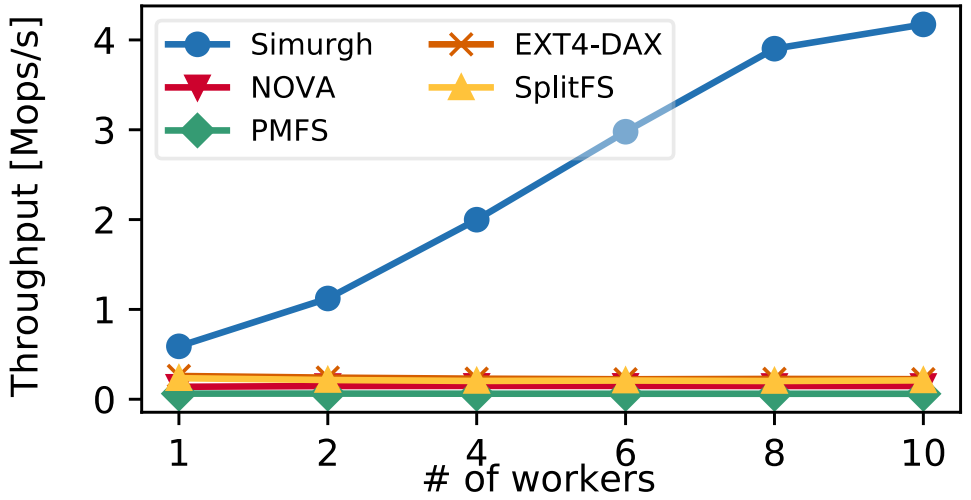
Overhead of JMPP/RETP is artificially added to all Simurgh functions

Shared Directory Operations

Create



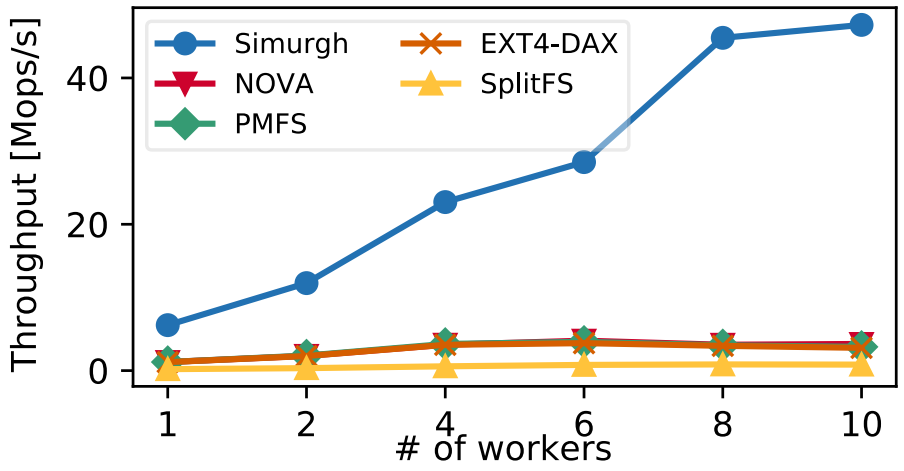
Rename



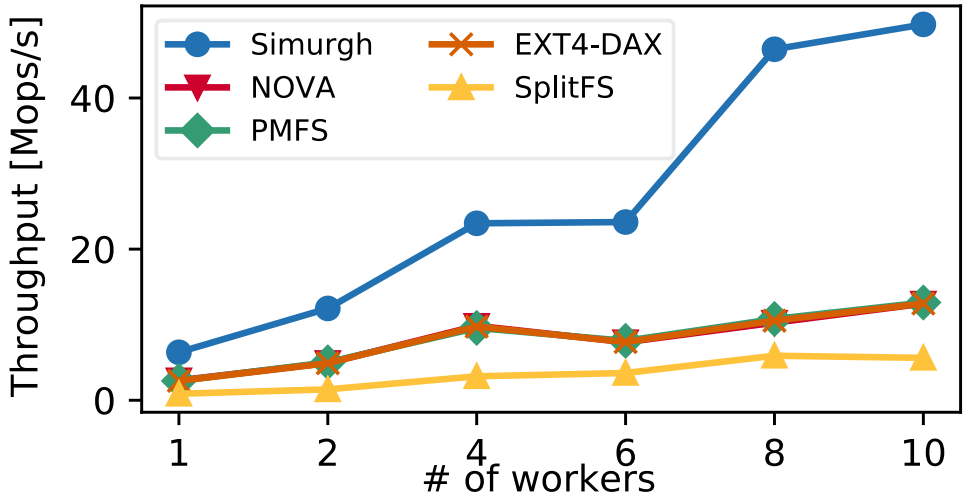
- No competition over VFS locks.
- Fast handling of metadata objects.

Resolve Path

Resolve shared path

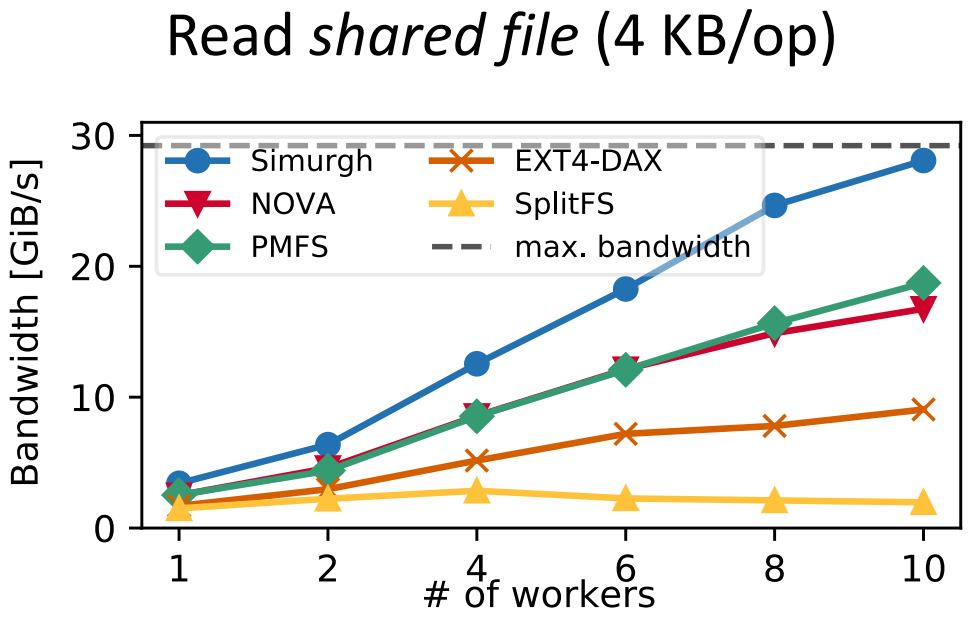


Resolve private path



- File listing scalable. Shared path resolving in VFS not scalable.
- Meta data layout and implementation allow fast traversal.

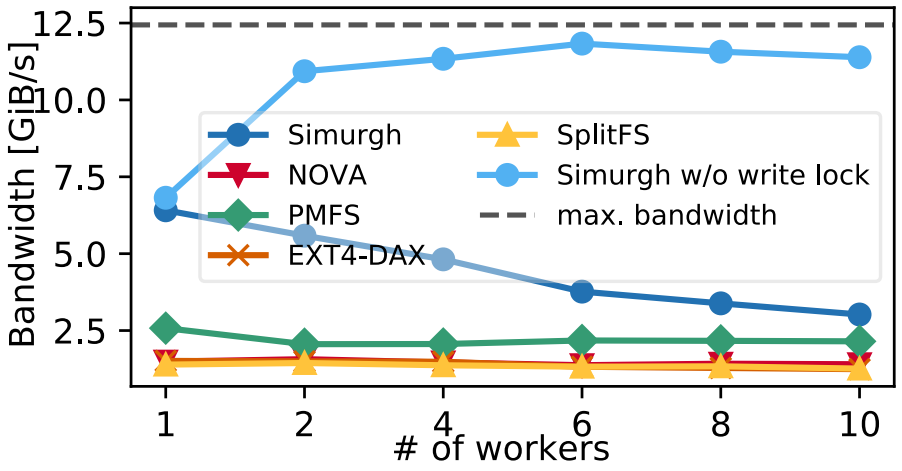
Read Operation



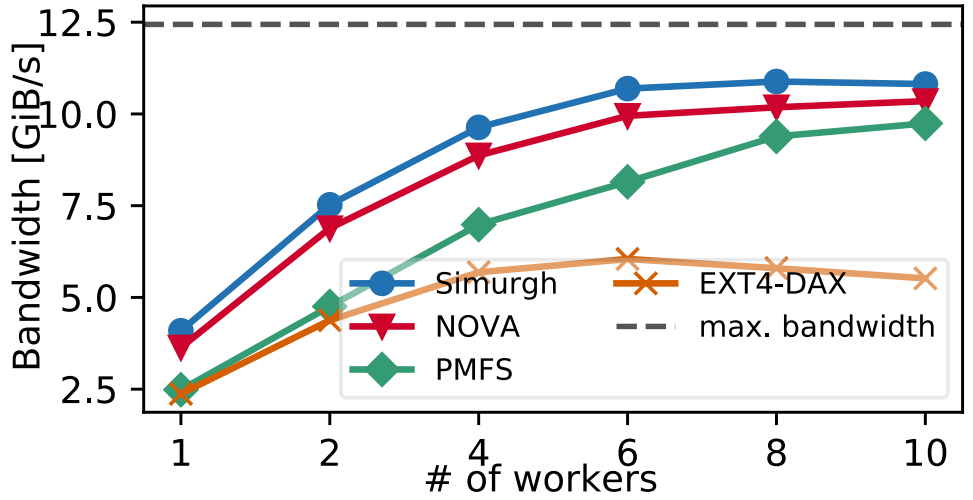
- Simurgh shows lowest overhead.
- No struggle with atomically updates of read/write semaphore.

Write Operation

Overwrite shared file (4 KB/op)

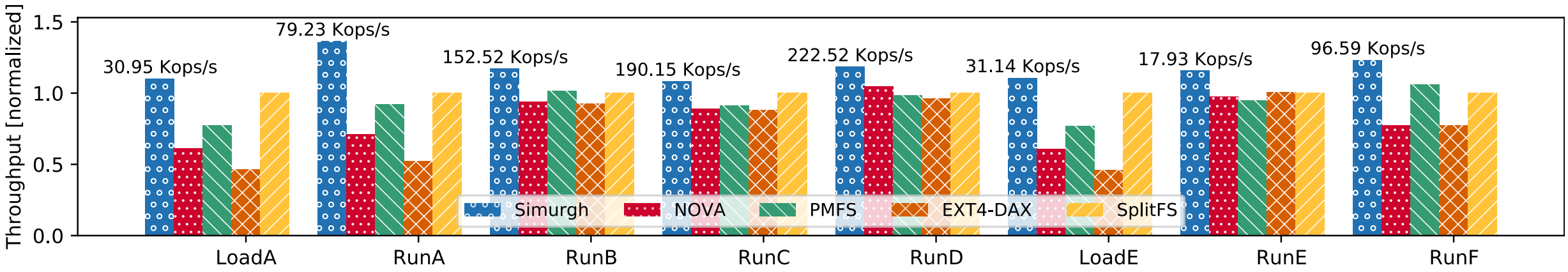


Write continuous (512 MB/op)

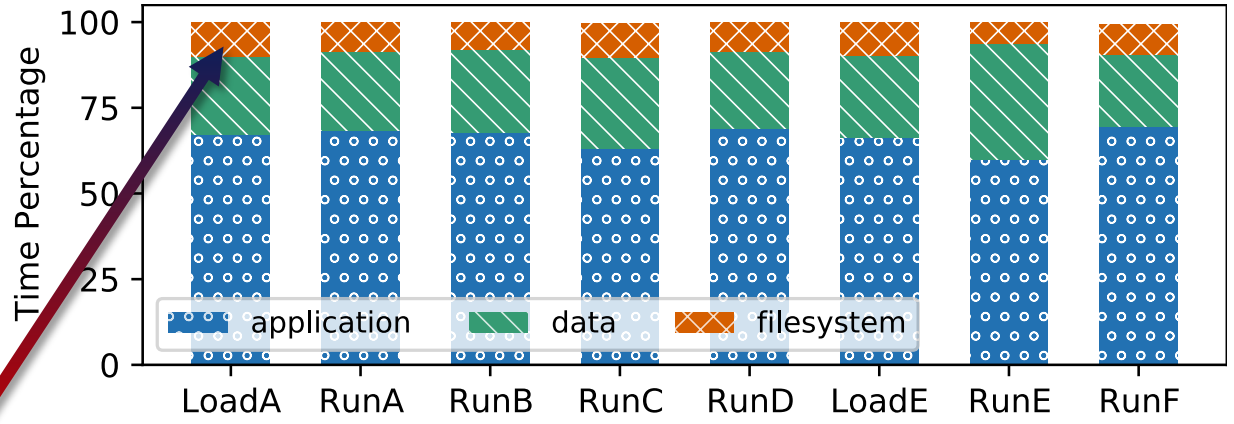


- Optional, lockless shared file writing.
- Close to device bandwidth.

YCSB Benchmarks



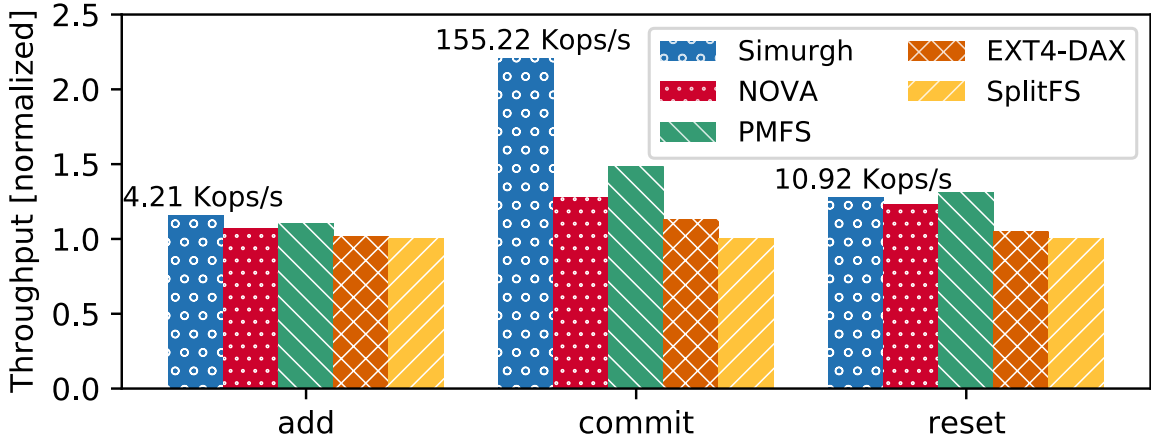
Execution time breakdown



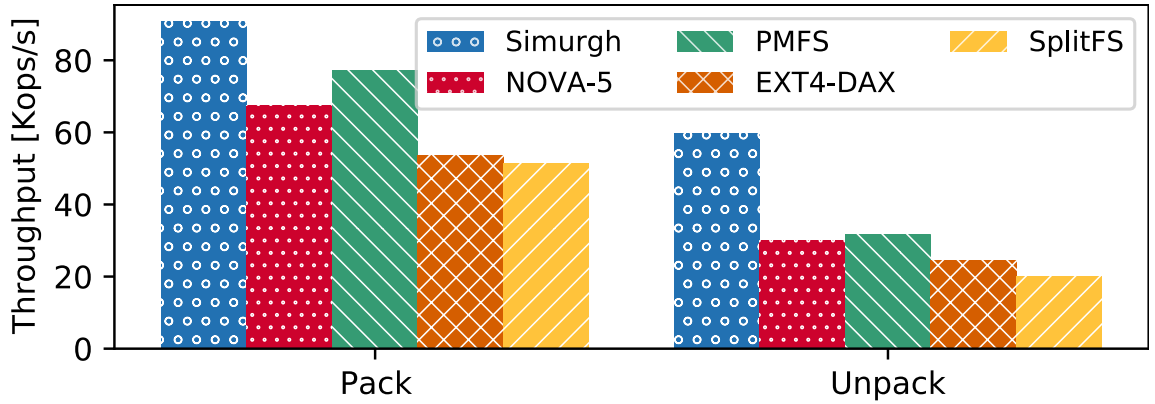
Nova > 50 % fs time

Tar & Git Throughput

Git



Tar



Even applications with high compute to I/O ratio can profit.

Runtime of I/O heavy applications can potentially be halved.

Conclusion

- Protected functions as a light-weight security mechanism.
- Simurgh is able to completely bypass OS‘s inefficient software stack.
- Simurgh offers the same protection and access rights as kernel level file systems.
- Simurgh guarantees consistency, durability and ordering.
- Evaluation shows outstanding data and meta data performance.

- Future work includes adopting Simurgh to distributed shared memory environments and multi-node HPC applications.

Thank you for your attention