NVMe® TPAR 4217 LBA Range Access Control

Overview:- Chaitanya Kulkarni, NVIDIA

Workload Requirements

Workload Requirements

- Modern HPC, AI/ML training, and GPU clusters require fine-grained, concurrent LBA access within a single namespace to ensure efficient parallel I/O
  - This shift to finer-grained accesses boosts concurrent throughput. As a result, highly parallel, GPU-initiated access becomes more practical.
  - GPU-based LBA access is inherently user-level and should be selectively isolated from protected metadata. (More on this later)
- LBA access patterns within a namespace evolve over time, driven by:
  - Dynamic workload behavior.
  - Shifting application demands.

Workload Requirements

- Traditional apps make a modest number of predictable, coarse-grained (dense) accesses. CPUs can easily saturate connections to storage.

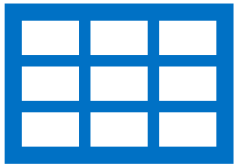
- A subset of emerging AI apps dominate large number of dynamic, fine-grained (sparse) accesses. For those apps only GPUs have enough threads to saturate connections to storage.

- Data set size is trending well beyond what fits in memory of many nodes.

Workload Requirements: Applications

Graph Neural Networks / Relational Graphs for Predictive AI / eCommerce applications

Each thread traverses to the next node based on its own access to fine-grained node embedding data.



Vector Database Search Index

Each thread performs an independent access to fine-grained data.*

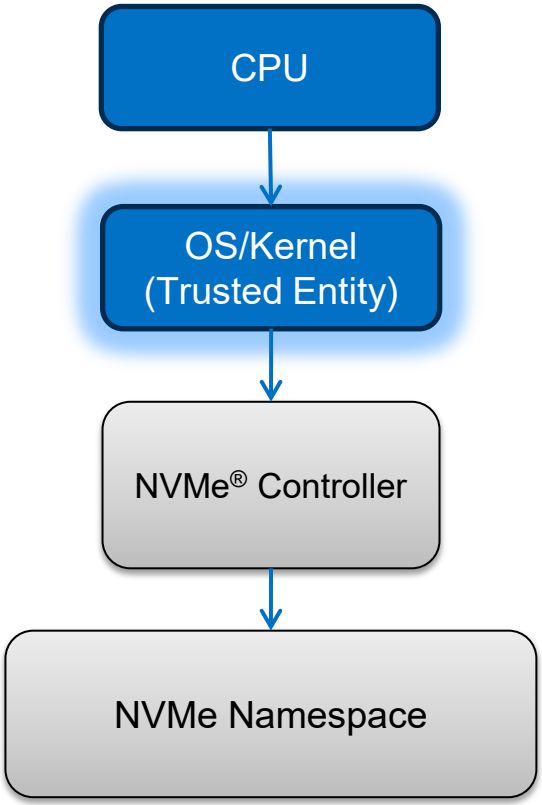
(*Unless temporal and spatial locality within coarse-grained IVF indexes is very strong, RAF/WAF will be prohibitive, outside of this scope.)

LBA Range Characteristics

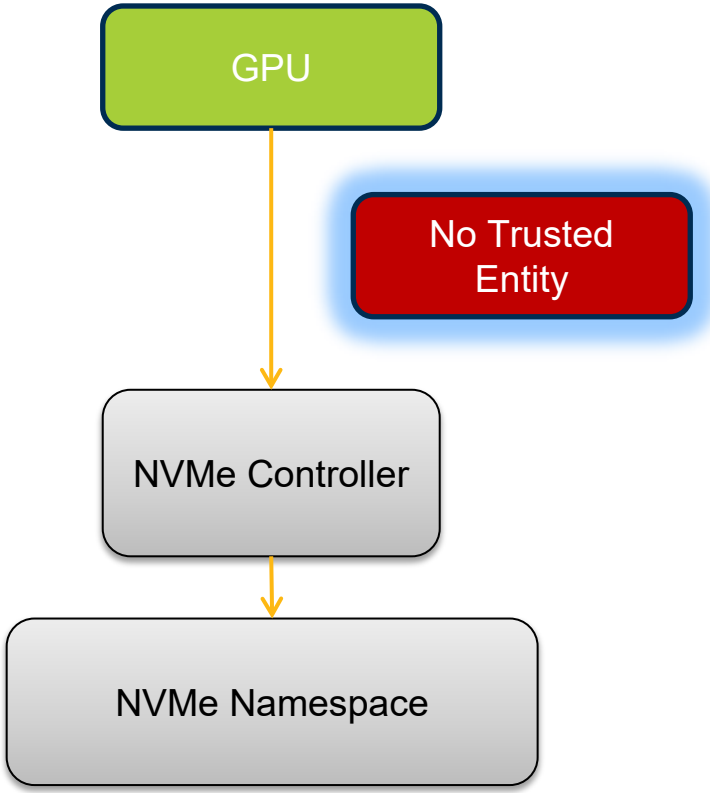
📐 **Size**	:	MBs to TBs (Never in KBs).
⚡ **Latency**	:	Acquire/Release Microsecond.
⏱️ **Duration**	:	Seconds to months (workload-dependent).
🔢 **Scale**	:	Active range count varies by deployment.
🔄 **Dynamism**	:	Rapid reallocation for starting GPU job.

Access Control Gap

CPU Architecture



GPU Architecture



Access Control Gap

OS-provided protections vs what SCADA* bypasses

✓ What CPU/OS Provides	✕ What SCADA Bypasses
✓ Per-process file descriptor table	✕ Per-operation validation
✓ UID/GID permission checks	✕ Kernel-mediated I/O path
✓ SELinux/AppArmor security policies	✕ VFS permission checks on each DMA transfer
✓ File system quota enforcement	✕ DMA Address Validation

⚡ **RESULT:** Storage device becomes the natural enforcement point

* *SCADA (Scaled Accelerated Data Access)* :- See Appendix A for more information.

SCADA I/O Path Analysis

TRADITIONAL **CPU Path**

Kernel validates & enforces access control at CPU memory stage

VS

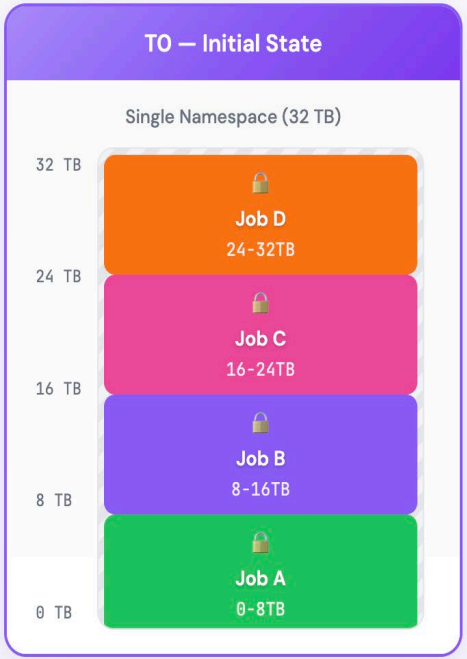
SCADA **Direct GPU-to-Storage Path**

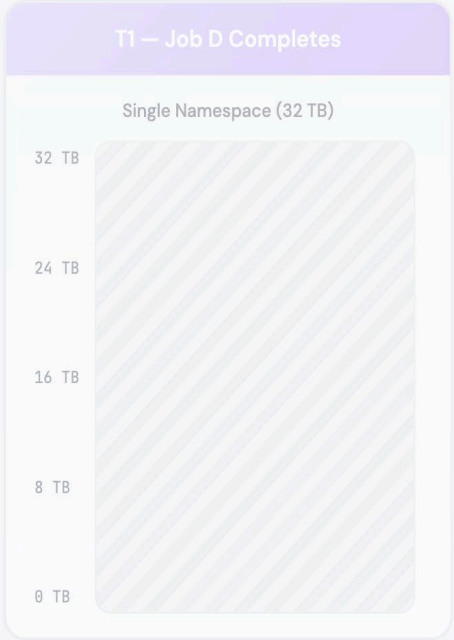
No kernel in path! — Where is access control enforced?

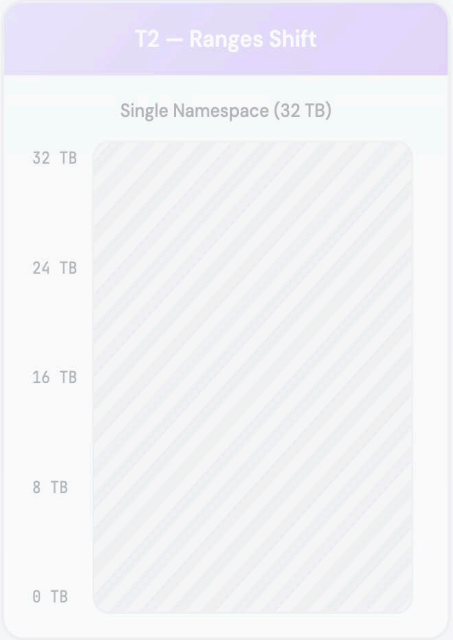
Access Control must be enforced at the storage level.

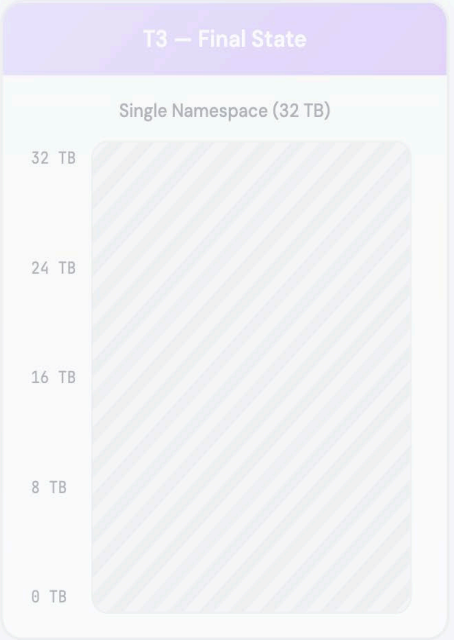
LBA Range Access Control Timeline

T0 — Initial State







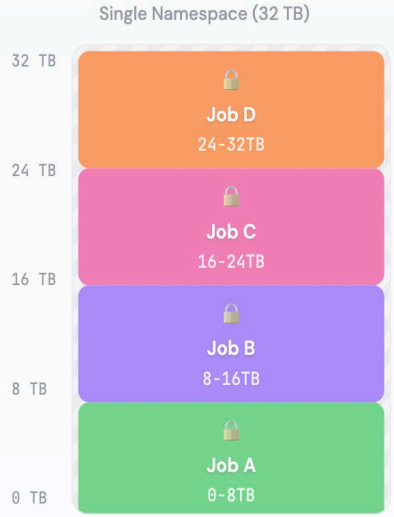


What's Happening

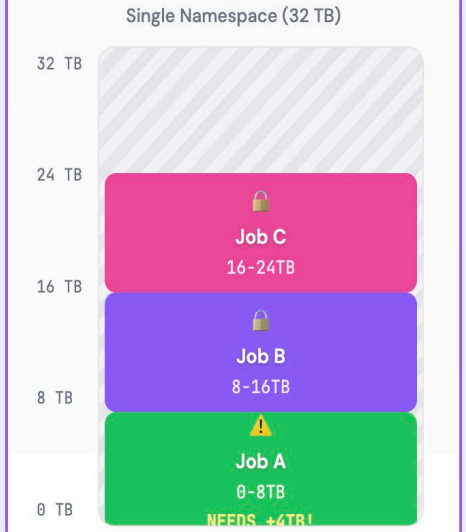
Initial allocation: Four AI training jobs start simultaneously, each assigned an 8TB checkpoint region within the shared 32TB namespace. Each job has hardware-enforced write protection for its assigned LBA range.

T1 — Job D Completes

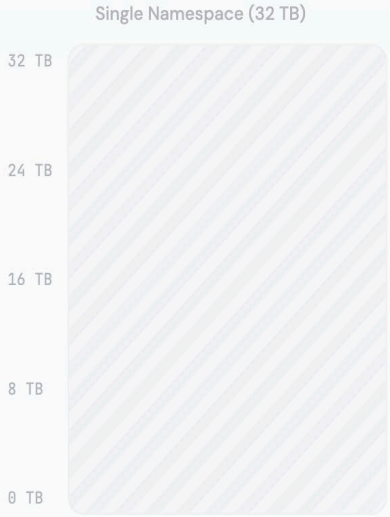
T0 — Initial State



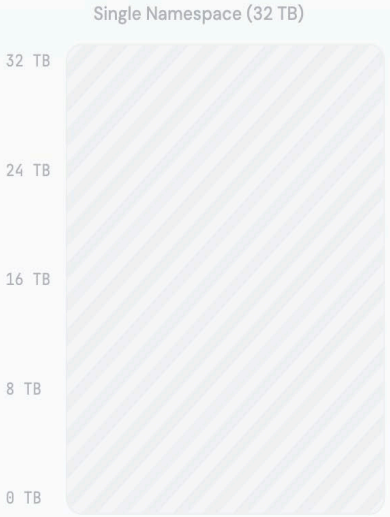
T1 — Job D Completes



T2 — Ranges Shift



T3 — Final State



What's Happening

Job D finishes training and releases its 24-32TB range. Meanwhile, Job A's model has grown and now requires **12TB** for checkpointing, but it only has 8TB allocated. The freed space from Job D creates an opportunity for reallocation.

T2 — Ranges Shift

What's Happening

Dynamic reallocation: Job A expands from 0-8TB to **0-12TB**. Job C shifts to 12-20TB. Job B moves to 20-28TB. All changes happen atomically with `zero I/O interruption` — no namespace recreation needed.

T3 — Final State

What's Happening

Stable configuration: Job A now has the 12TB it needs (0–12TB). Job B occupies 20–28TB. Job C sits at 12–20TB. 4TB remains free (28–32TB) for future growth. Each job's LBA range is `hardware-protected` against writes from other jobs.

Proposed Solution

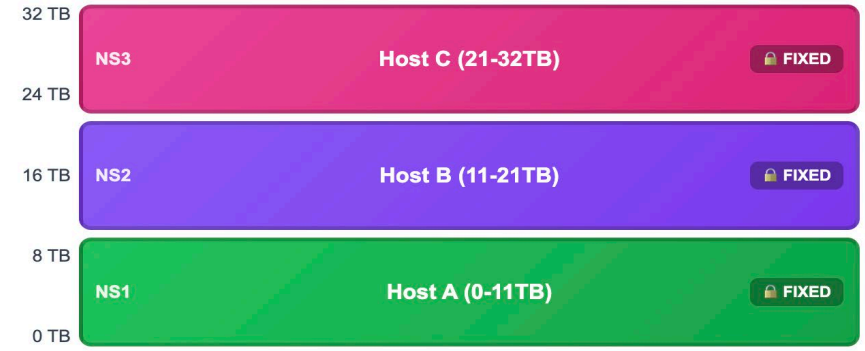
- **Multi-Host LBA Range Access Control**

  - Multi-Host LBA Range Access Control enables precise and coordinated access to distinct LBA ranges within a single Namespace.

- **Key Benefits:**
  - Maximizes parallel host access by eliminating namespace-level bottlenecks.
  - Enables addressable and range-specific access enforcement.
  - Provides guaranteed, native multi-host synchronization directly at the storage protocol level—without external orchestration.

Current: Namespace-Level Reservations — Why It Fails

Option A: Static Namespaces

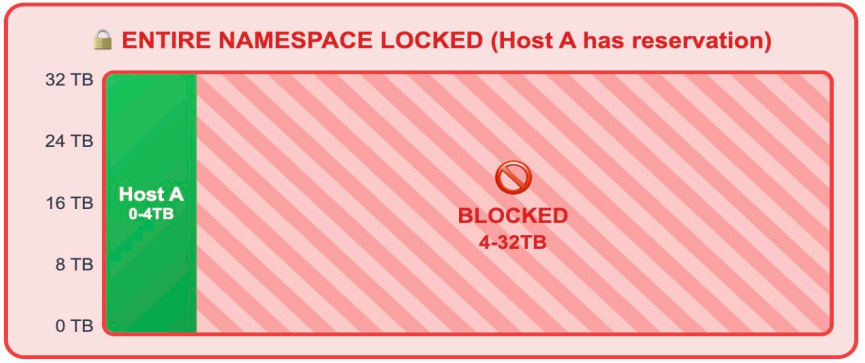


T0: ✓ Works	T1: ✗ Can't shift	T2: ✗ Can't expand	T3: ✗ Can't handoff

⚠ Key Problem

Boundaries fixed at creation time. Adapting to changing requirements needs destroy + recreate cycle with downtime.

Option B: Namespace-Level Reservations



✓ Host A	✗ Host B	✗ Host C
Needs: 4TB | Range: 0-4TB	Needs: 4TB | Range: 16-20TB	Needs: 4TB | Range: 23-27TB
HAS ACCESS	BLOCKED!	BLOCKED!

⚠ Key Problem

Entire 32TB namespace locked even though Host A only needs 4TB (12.5%).
Result: 28TB (87.5%) blocked unnecessarily — massive waste and contention.

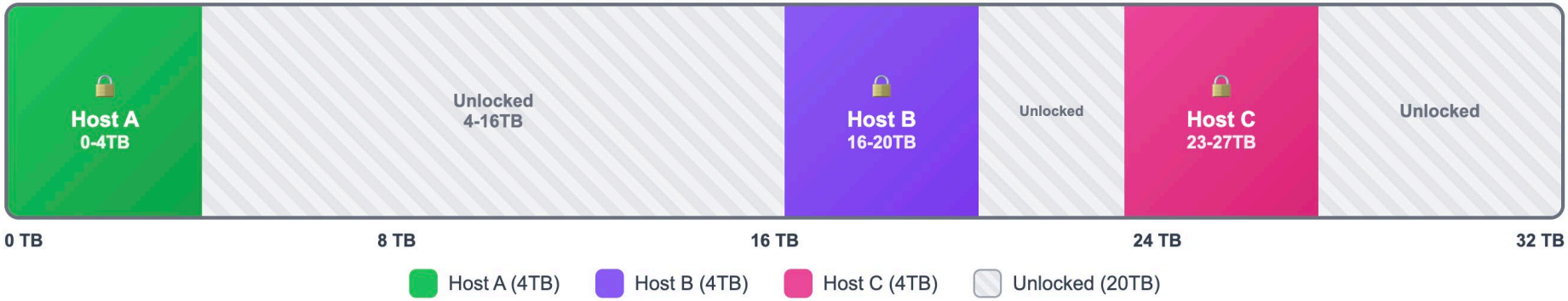
Host A Active (4TB - 12.5%) | Blocked/Inaccessible (28TB - 87.5%)

SDC | StorageAI™

nvm EXPRESS®

Proposed: LBA Access Control

Single Namespace (32 TB) with LBA Access Control



✓ Host A ✓ Host B ✓ Host C **All 3 Hosts Access Concurrently - No Blocking!**

Benefits

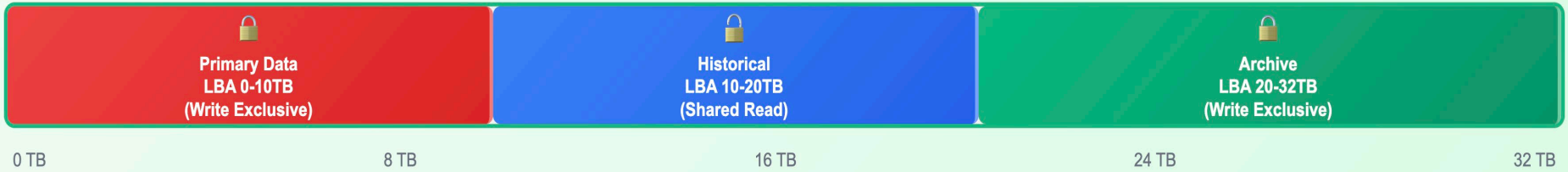
- ✓ **Fine-grained locks:** Each host locks only 4TB (12.5%) instead of entire 32TB
- ✓ **True parallelism:** 3 hosts work simultaneously on different ranges
- ✓ **Efficient utilization:** 62.5% of namespace remains available for other hosts
- ✓ **No contention:** Non-overlapping ranges = no blocking

USECASE : Database Partitioning

Use Case: Database Partitioning

Distributed database partitions storage across primary write node, read replicas, and archive logs - all sharing the same high-performance namespace

Single Namespace (32 TB) with Range Locks



Host A	Host B	Host C	Host D	Host E
Primary Writer	Read Replica	Read Replica	Read Replica	Archive
Real-time transactions exclusive access	Analytics queries parallel access	Analytics queries parallel access	Analytics queries parallel access	Log archival append-only

✓ Concurrent Operations Without Lock Contention

Primary writes | Replica reads | Archive writes - all simultaneously on different LBA ranges

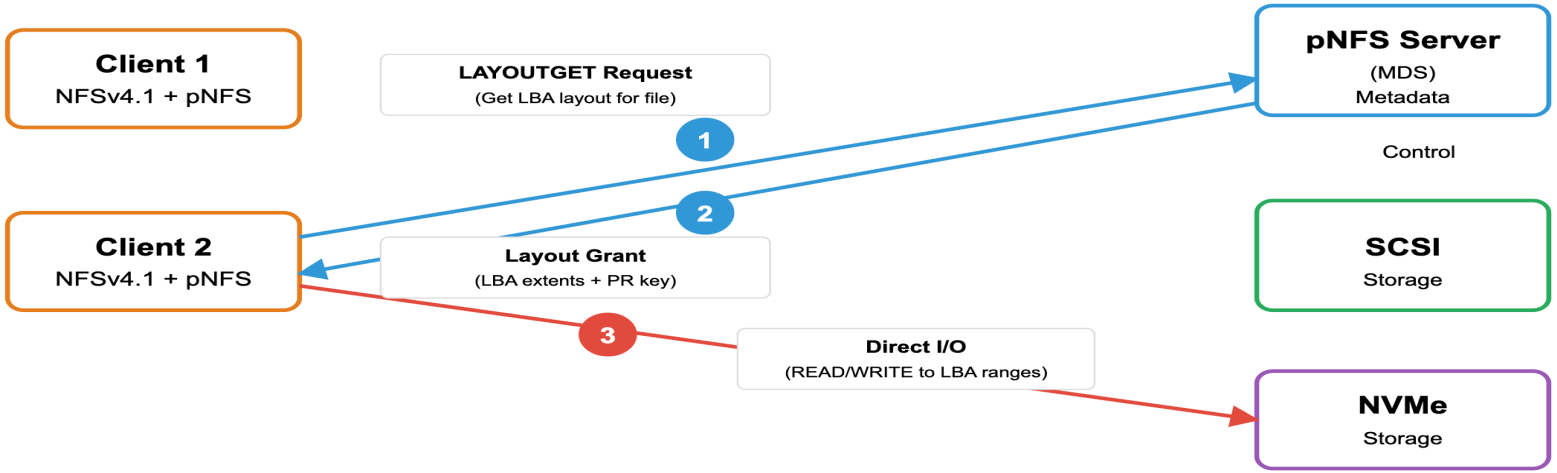
Benefits

Parallelism	Performance Isolation	Resource Efficiency	Failover Capability
Concurrent ops without contention	Workloads don't interfere	Single namespace, multiple access patterns	PREEMPT for primary takeover

USECASE : LBA Range Aware pNFS layout for NVMe® Technology

Very high level pNFS Architecture

Parallel Network File System - SCSI & NVMe Layouts



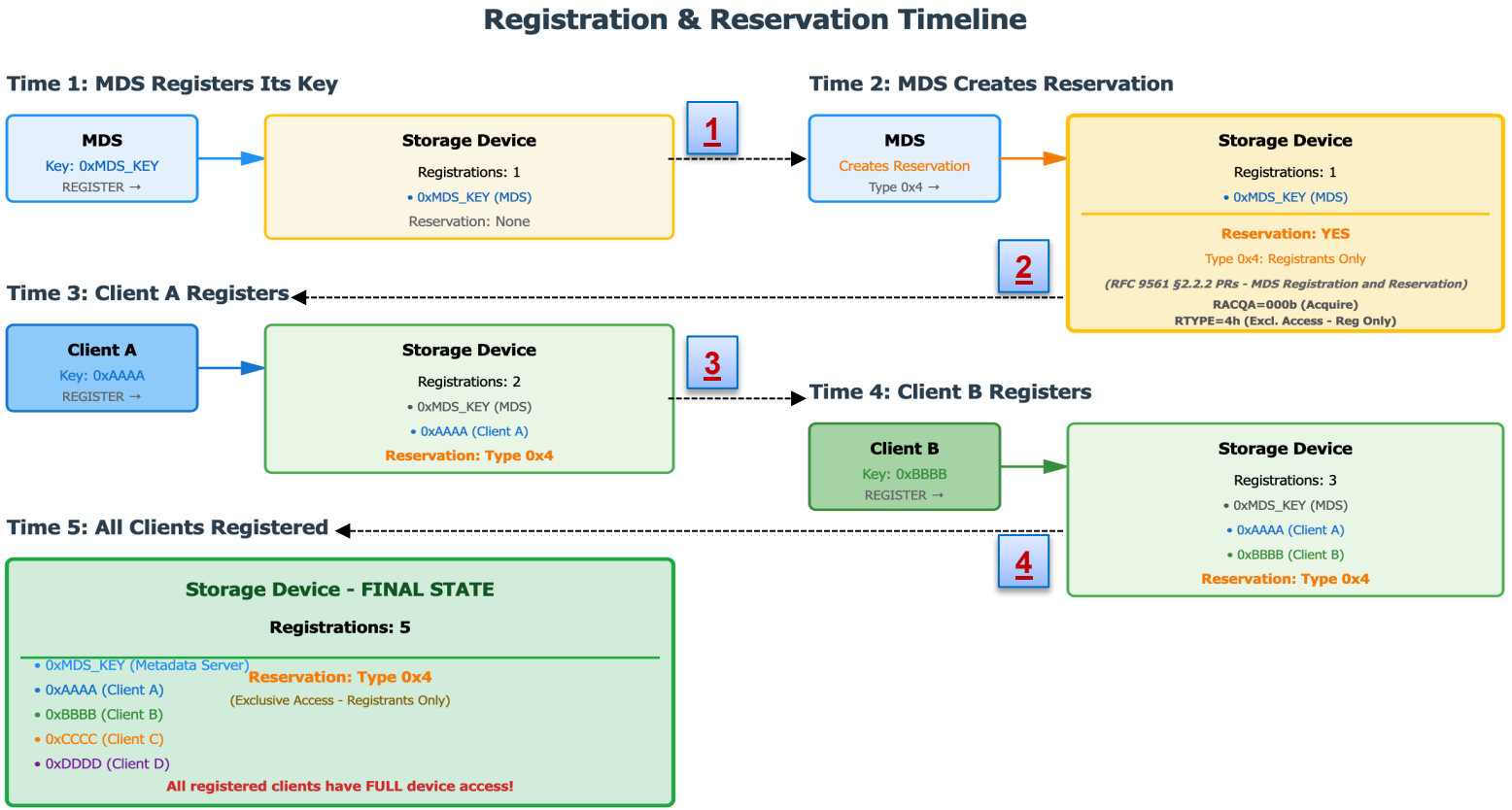
Flow:

① Client sends LAYOUTGET (requests LBA layout)
② Server grants layout (LBA extents + PR key)
③ Client performs direct I/O

Reference: RFC 8154 (pNFS SCSI) | RFC 9561 (pNFS NVMe) | TPAR4217 (LBA Range Locking)

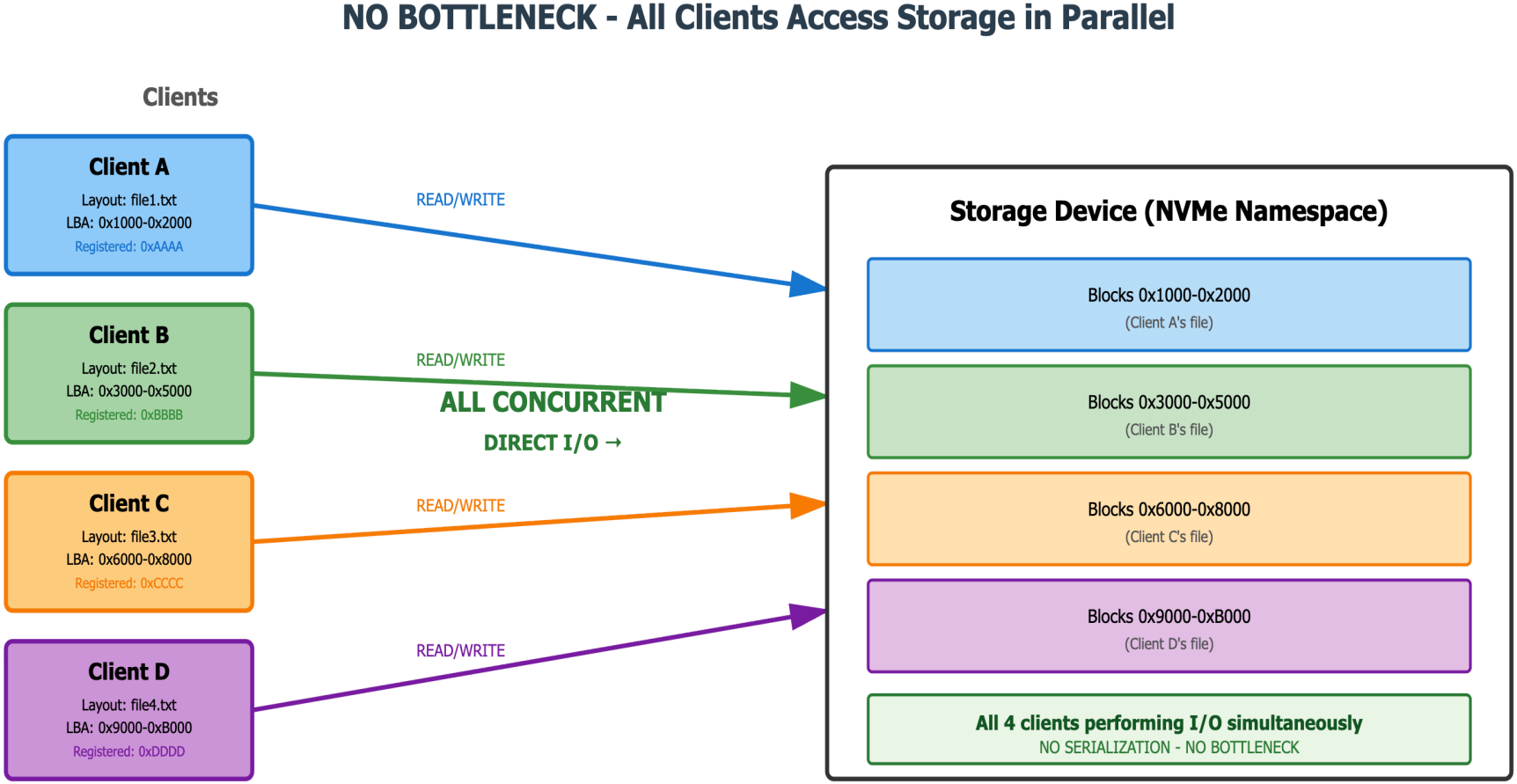
How pNFS NVMe Registration and Reservation Works

Step-by-step process showing how the MDS creates a reservation and clients register their keys with the storage device.



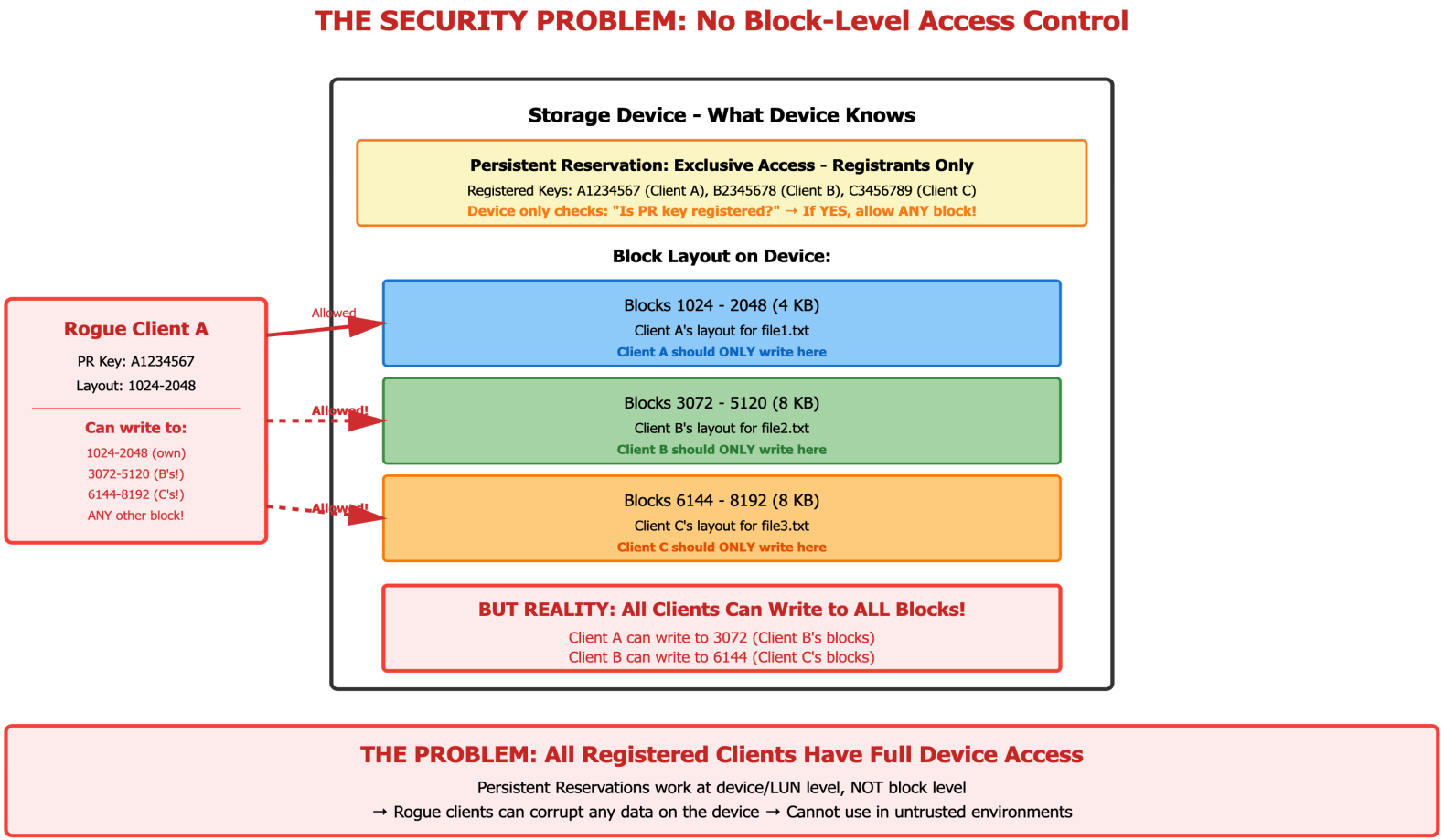
pNFS NVMe: Excellent Concurrency

All registered clients can read/write to the device simultaneously. There is NO lack of concurrency - this is the major performance advantage of pNFS NVMe.



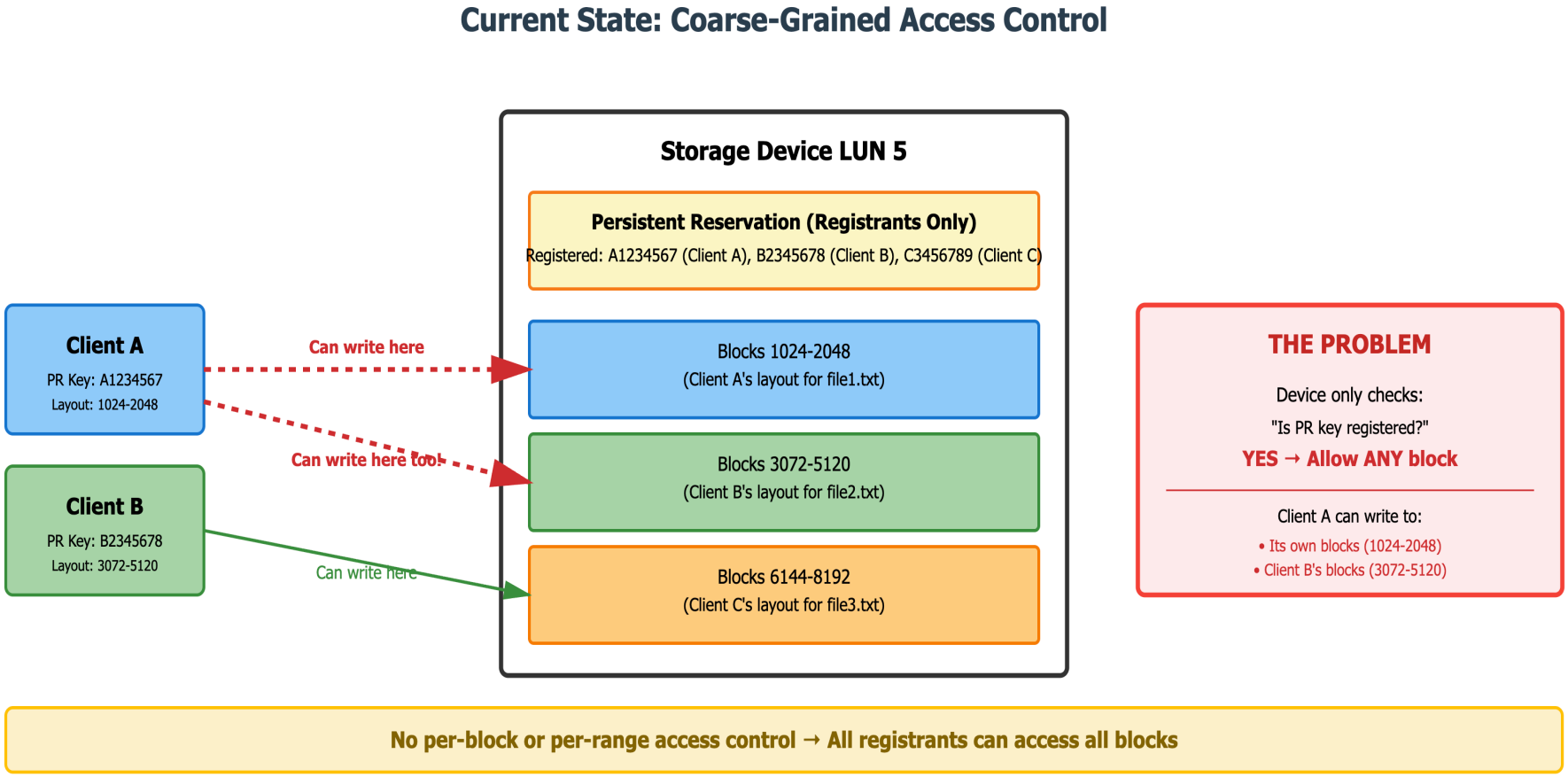
BUT: The Security Problem - Rogue Clients Can Write Anywhere

The problem is NOT concurrency - the problem is that ALL registered clients have FULL device access. A malicious or buggy client can write to ANY block on the device, not just its assigned ranges.



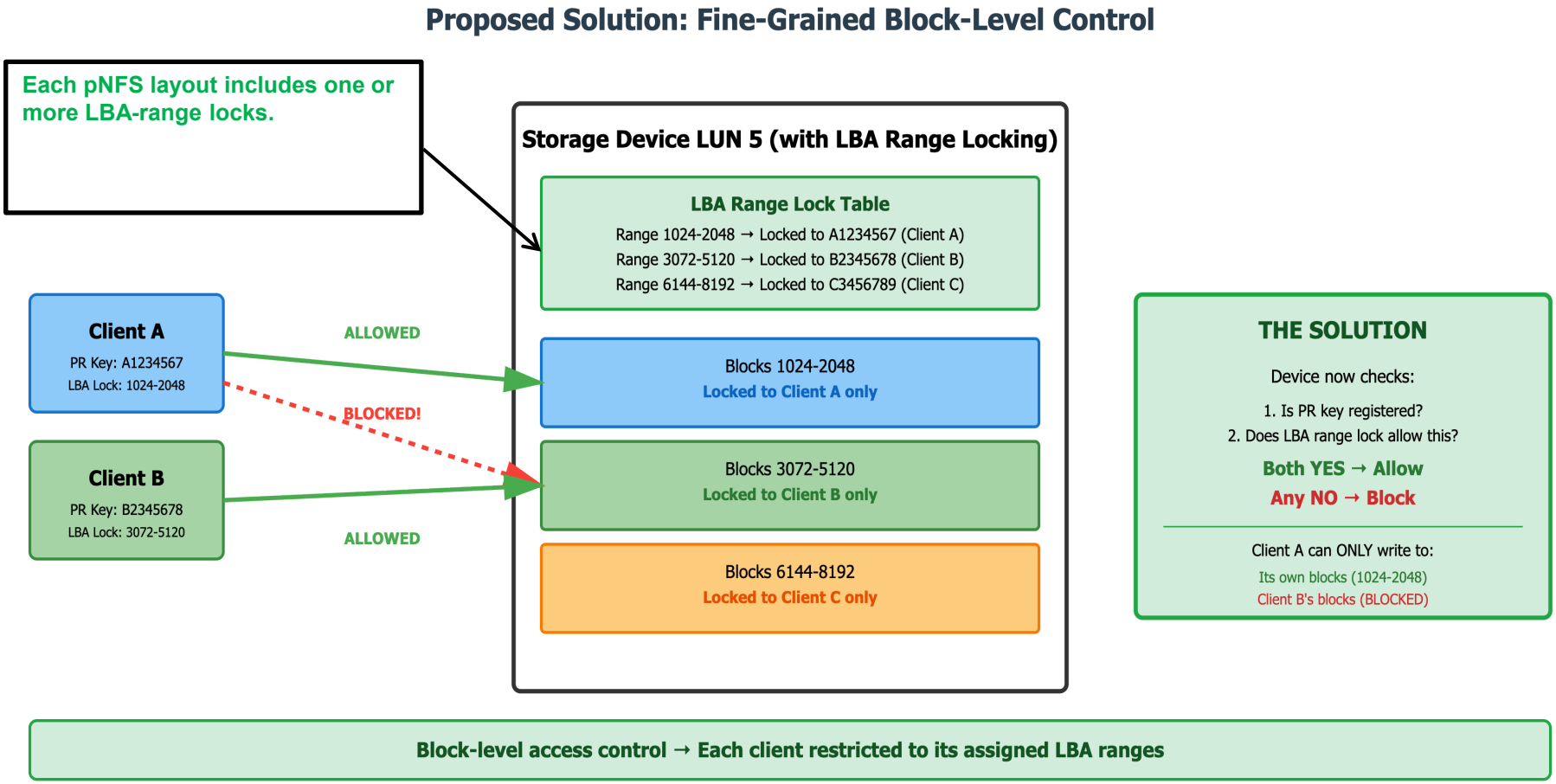
Current Problem: No Block-Level Access Control

With only Persistent Reservations, all registered clients can access any block on the device.



Proposed Solution: LBA Range Locking

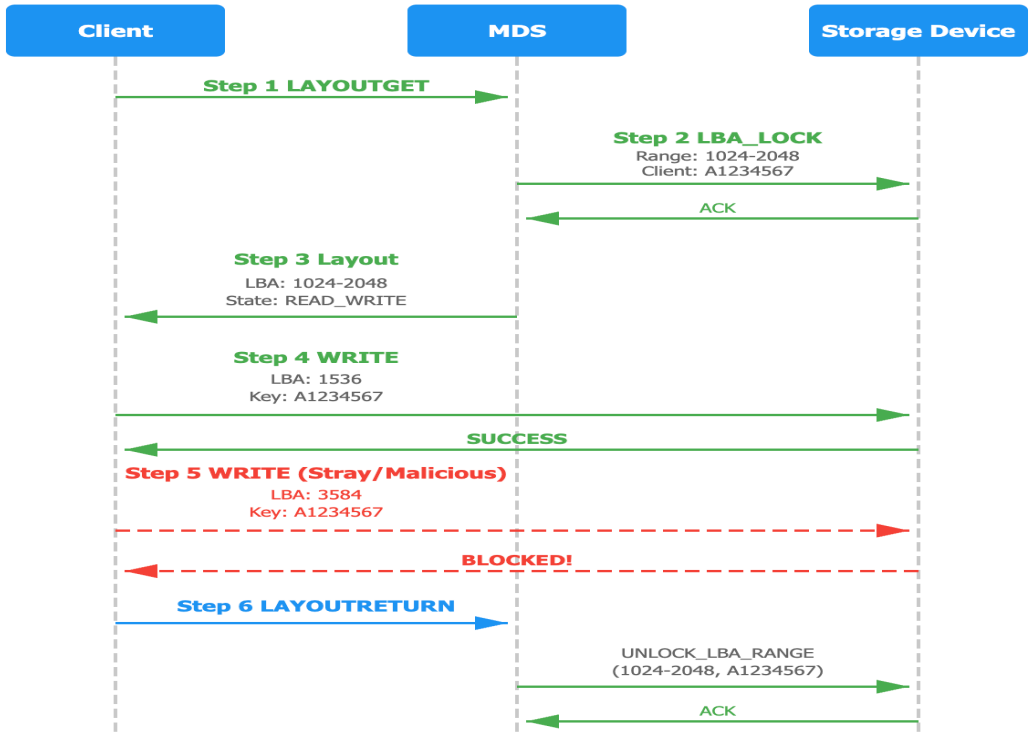
With LBA range locking, the storage device can enforce per-client, per-range access control at the block level.



How LBA Range Locking Works

Step-by-step process of setting up and using LBA range locks in pNFS NVMe.

LBA Range Locking Protocol Flow



Step 1: LAYOUTGET
Client requests layout
for file /data/file1.txt

Step 2: MDS Sets LBA Lock
MDS issues LBA range lock command to storage device
`LOCK_LBA_RANGE(1024–2048, A1234567)`

Step 3: Return Layout
MDS returns layout to client
with LBA range info

Step 4: Legitimate I/O
Device checks: PR + LBA Lock
Client writes to its range block 1536 (within 1024-2048)

Step 5: Stray/Malicious I/O
Device checks: PR LBA Lock
Client tries to write to another client's range block 3584 (NOT in 1024-2048)

Step 6: LAYOUTRETURN
Client returns layout
MDS releases LBA locks

Questions ?

APPENDIX

APPENDIX A

NVIDIA SCADA :-

Scaled Accelerated Data Access

NVIDIA SCADA

NVIDIA SCADA — Scaled Accelerated Data Access

SCADA stands for Scaled Accelerated Data Access and is a storage data I/O scheme in which GPUs directly initiate and control storage I/O. It is part of NVIDIA's "Storage-Next" architecture.

The Problem It Solves

New applications like GNNs and vector databases make fine-grained requests from every GPU thread to more data than can fit in the memory of many nodes. For AI inference workloads, the GPU is working on over one thousand data-intensive parallel threads, which typically requires smaller data sets. Not being able to get these data sets at the required speed leads to underutilization of expensive GPU cycles.