Accelerating and Securing GPU Accesses to Large Datasets

GTC '24 S62559 Tuesday Mar 19, 8am

CJ Newburn, Distinguished Engineer, NVIDIA GPU Cloud

Oren Duer, Director of Storage Architecture, Networking Group

Vikram Mailthody, Sr. Research, NVIDIA Research

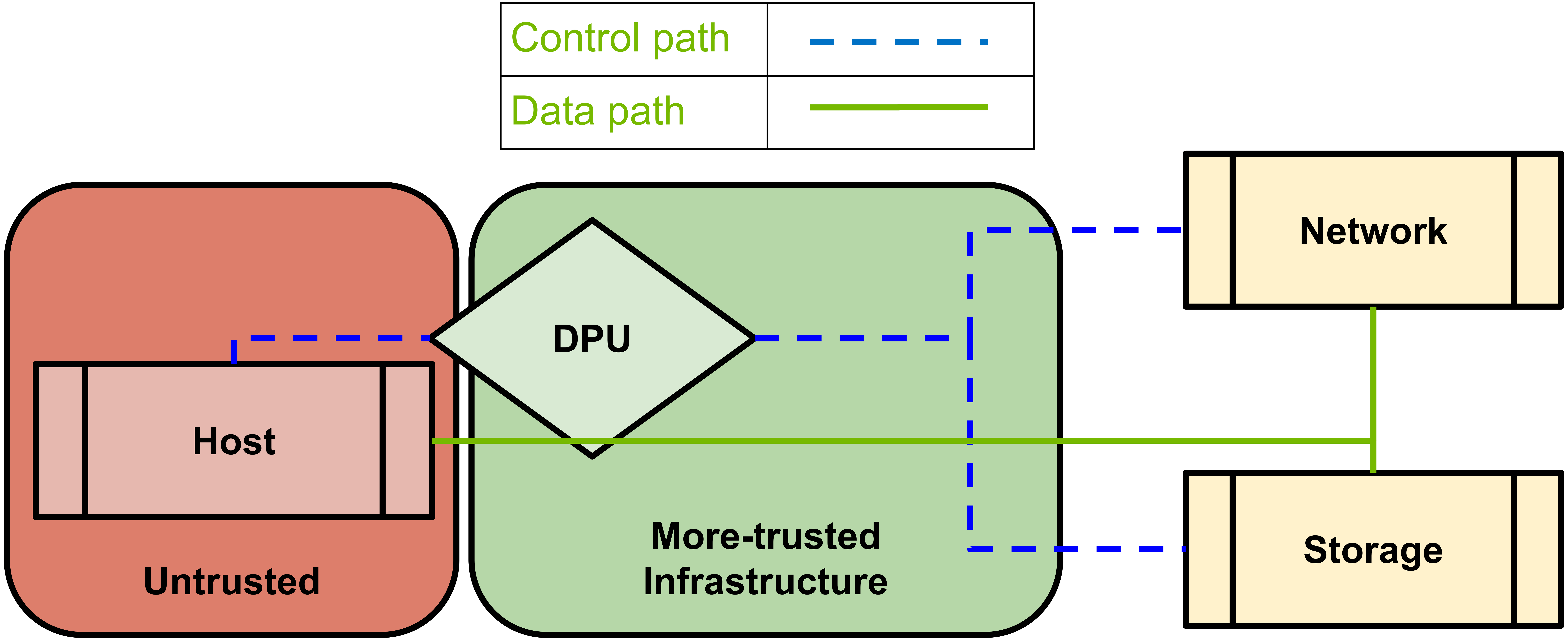
Trends

New considerations as we scale up and out

- Scaled data
  - Scaled data sets won't fit in the memory of one GPU or even of many nodes → use NVMe
  - Can't reach all data via loads and stores → need new API
  - Key-value/object stores are gaining traction as a way to access data → custom APIs for objects
  - Too much data for apps to track → serverless, with dataset services, orchestration
- Scaled clusters
  - Inefficient to statically partition resources → shared compute and storage resources with security
  - Don't trust computing resources → shift critical operations to trusted infrastructure control plane

Using a more-secure proxy

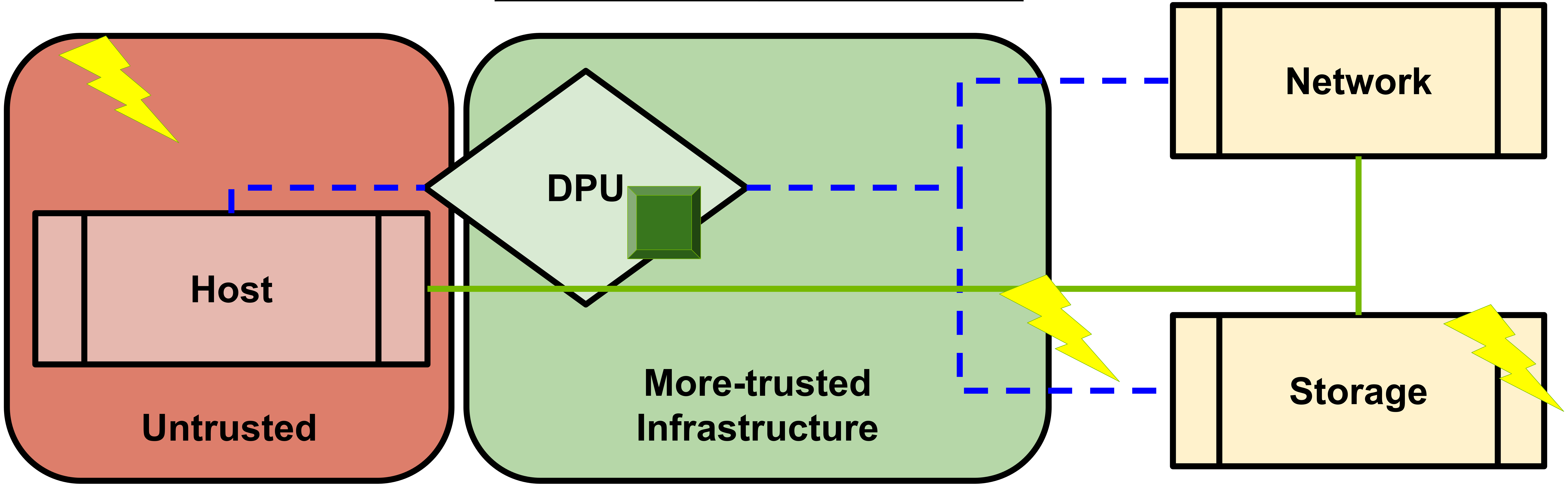
Control path enabled by trusted infrastructure, uninhibited data path (RDMA)



Attacks

Mitigated by putting a proxy agent on the DPU between the untrusted host and backend filer

Control path	- - - - -
Data path	———



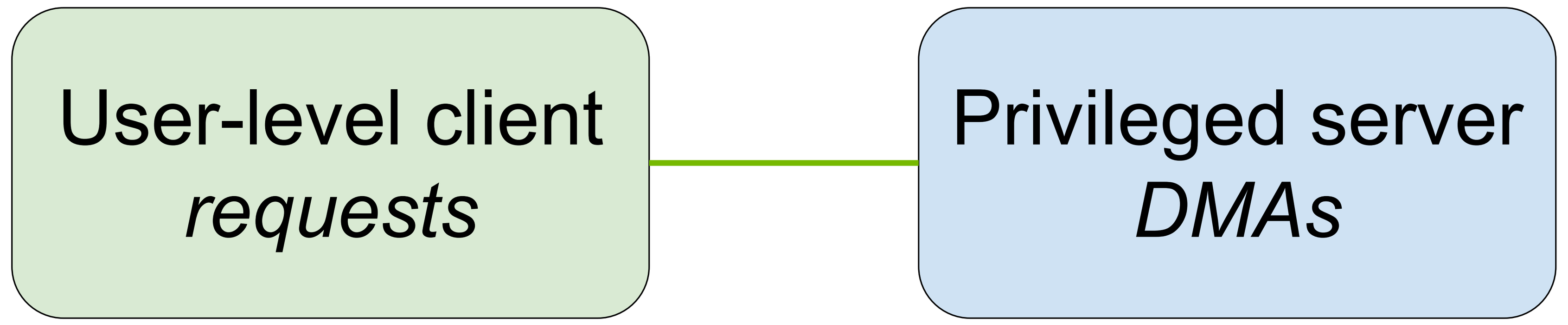
- Denial of service - new connections
- Mounting
- Bogus requests for blocks/keys
- Requests from wrong nodes

Storage client

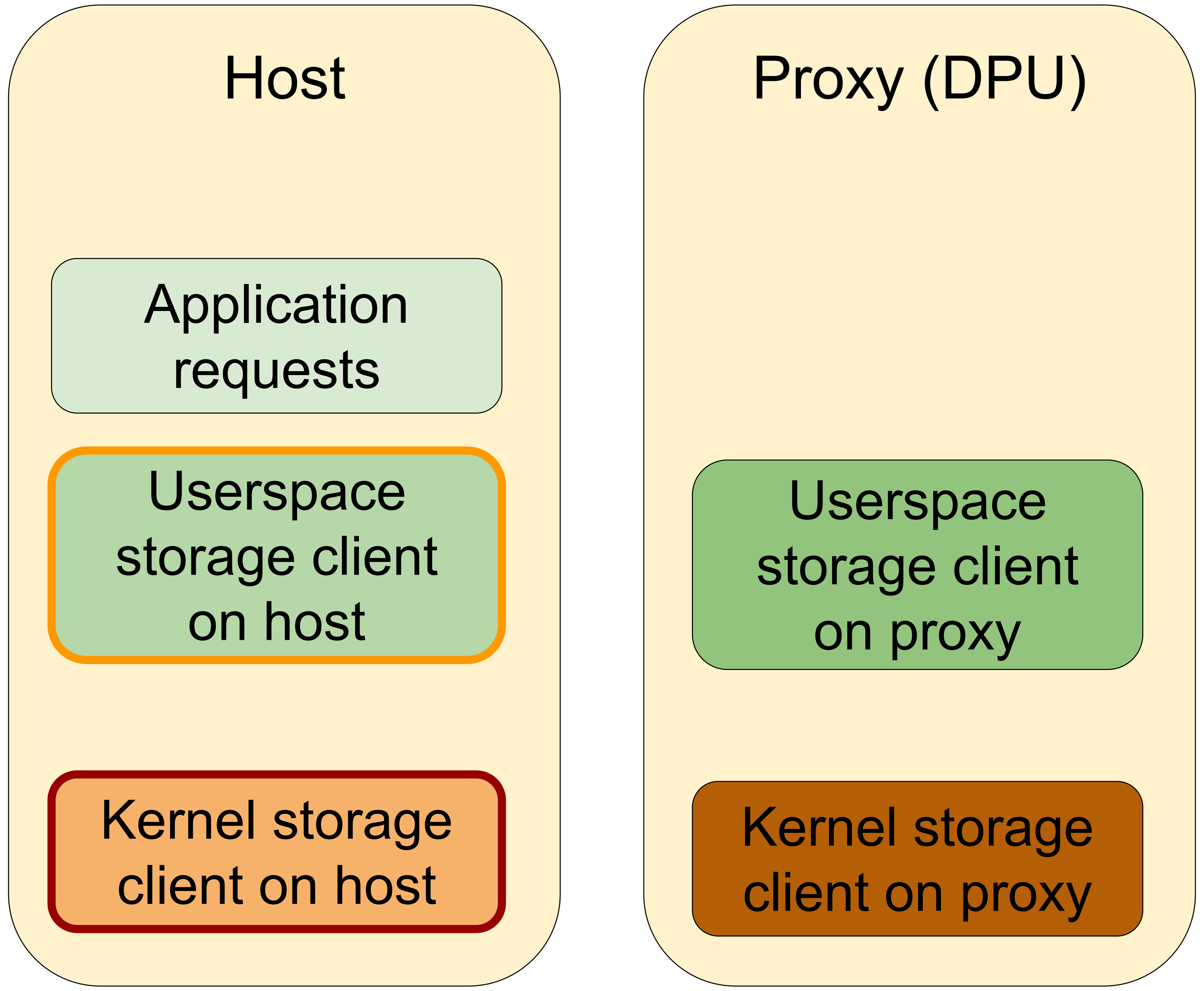
Shift from untrusted host to more-trusted proxy on DPU

App and DMA engine must be separated

- Otherwise app can spray all over host physical memory

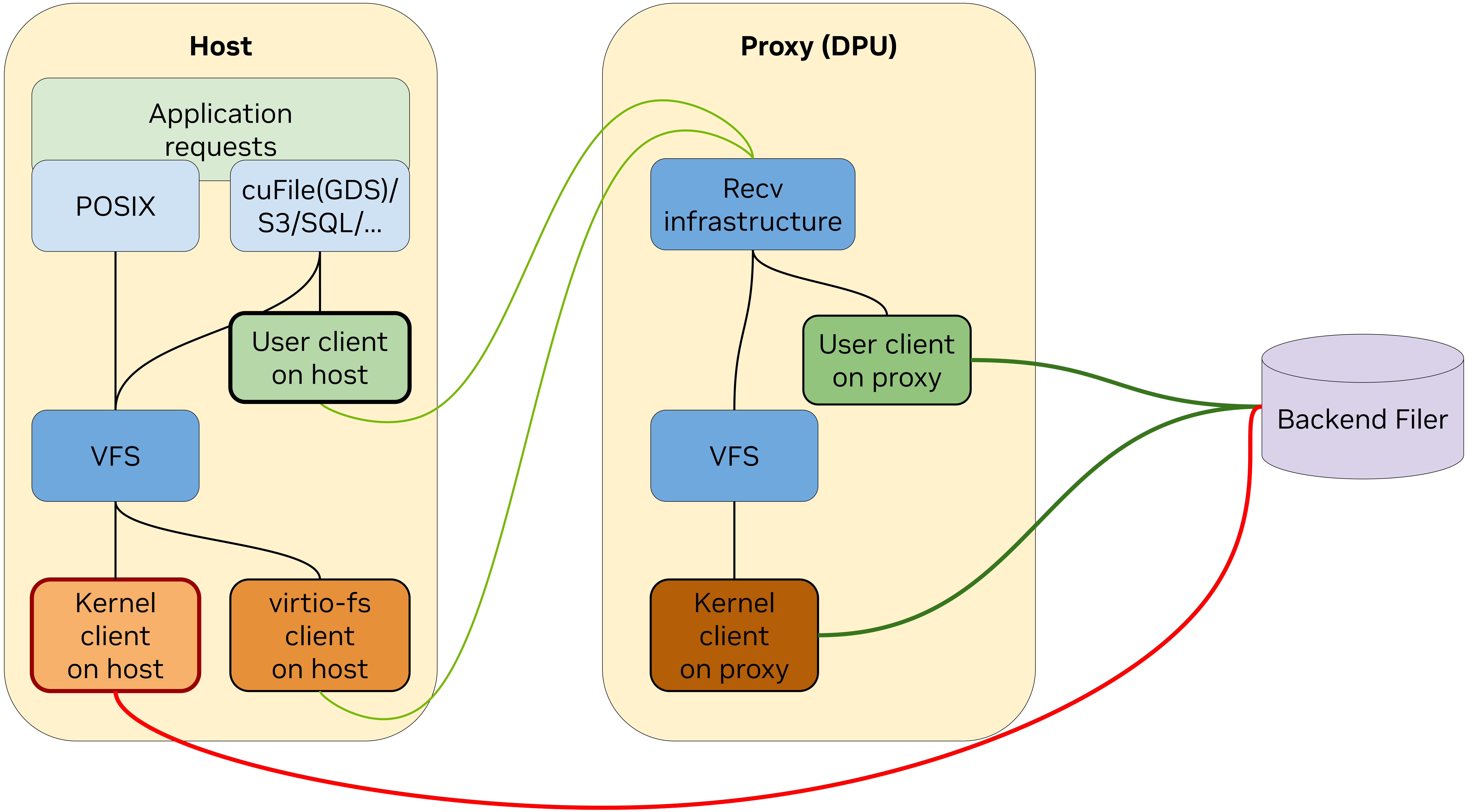


Host app makes a request; 4 choices for storage client



Paths from application to storage client

Shift from untrusted host to more-trusted proxy on DPU



RDMA steps, with a host app target

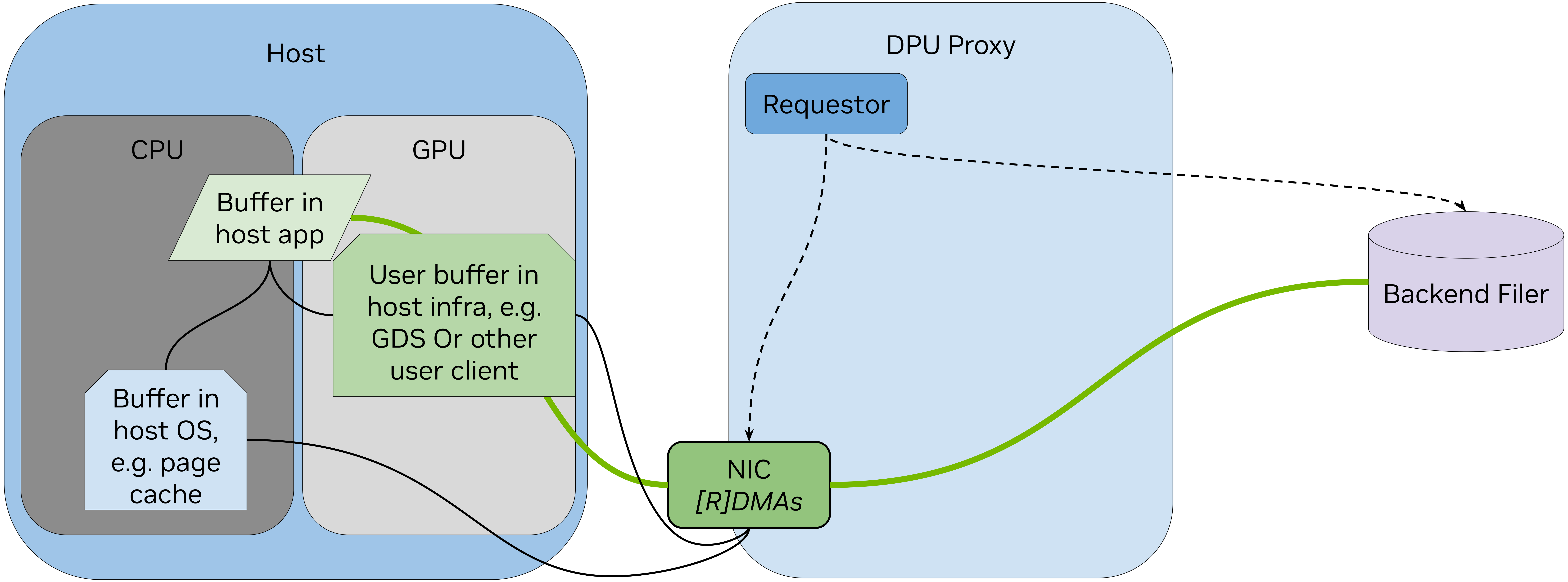
Create, communicate, and apply the remote key (rkey); zero copy

RDMA: bounce buffers on the DPU proxy

Sysadmins and storage clients make different trade-offs, e.g. data integrity vs. performance

RDMA: bounce buffers on the host

Client/infrastructure, page cache, or zero copy into the host application



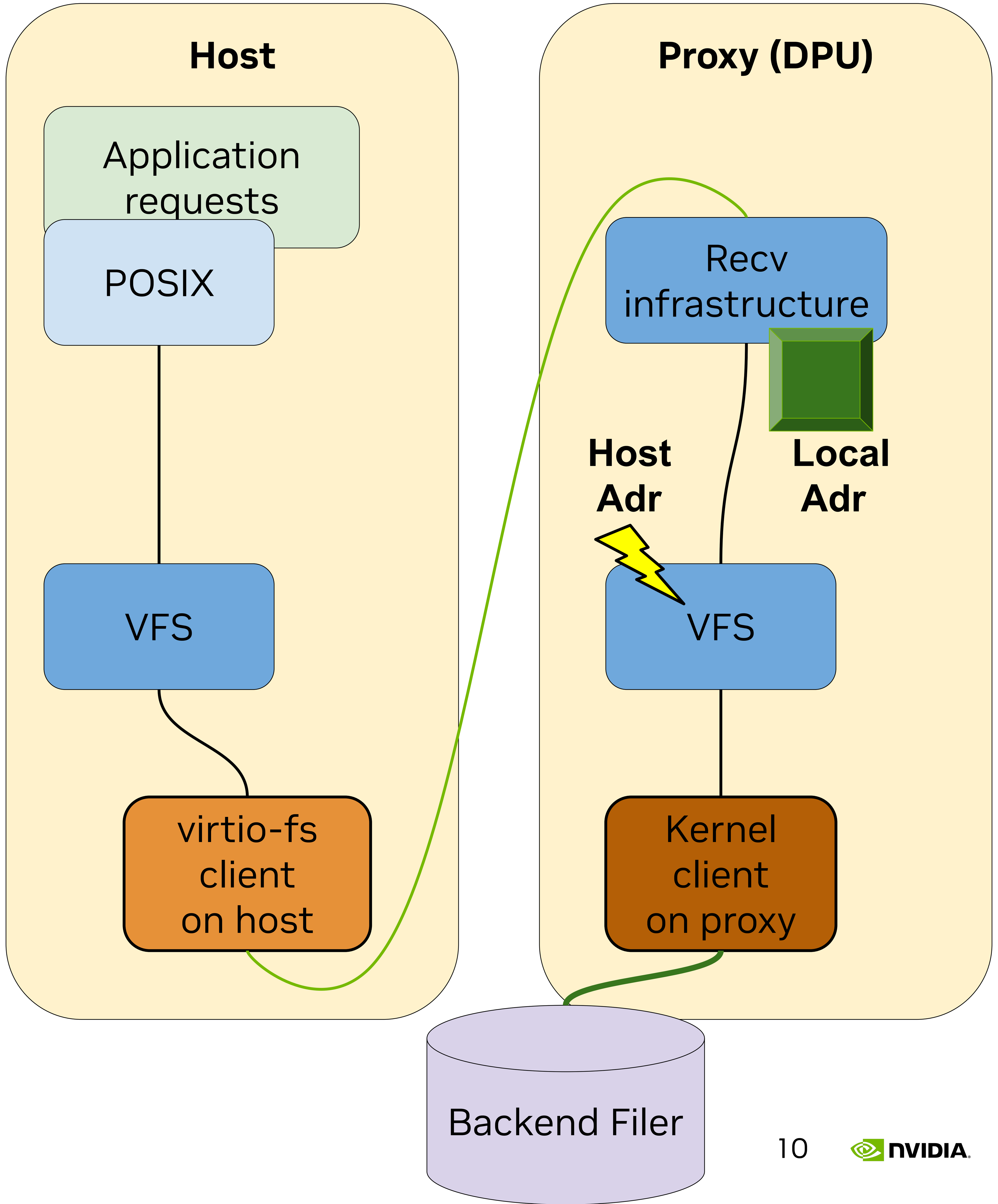
Suboptimal performance with proxy bounce buffers

A bounce buffer on the DPU becomes the RDMA destination, requiring an extra hop to the host

Downsides - avoidable with a userspace client on DPU

- VFS requires a local address, mandates a bounce buffer in the DPU proxy
- Throughput implications: contention in system cache
- Latency implications: store and forward, interrupts
- CPU utilization implications: host polling

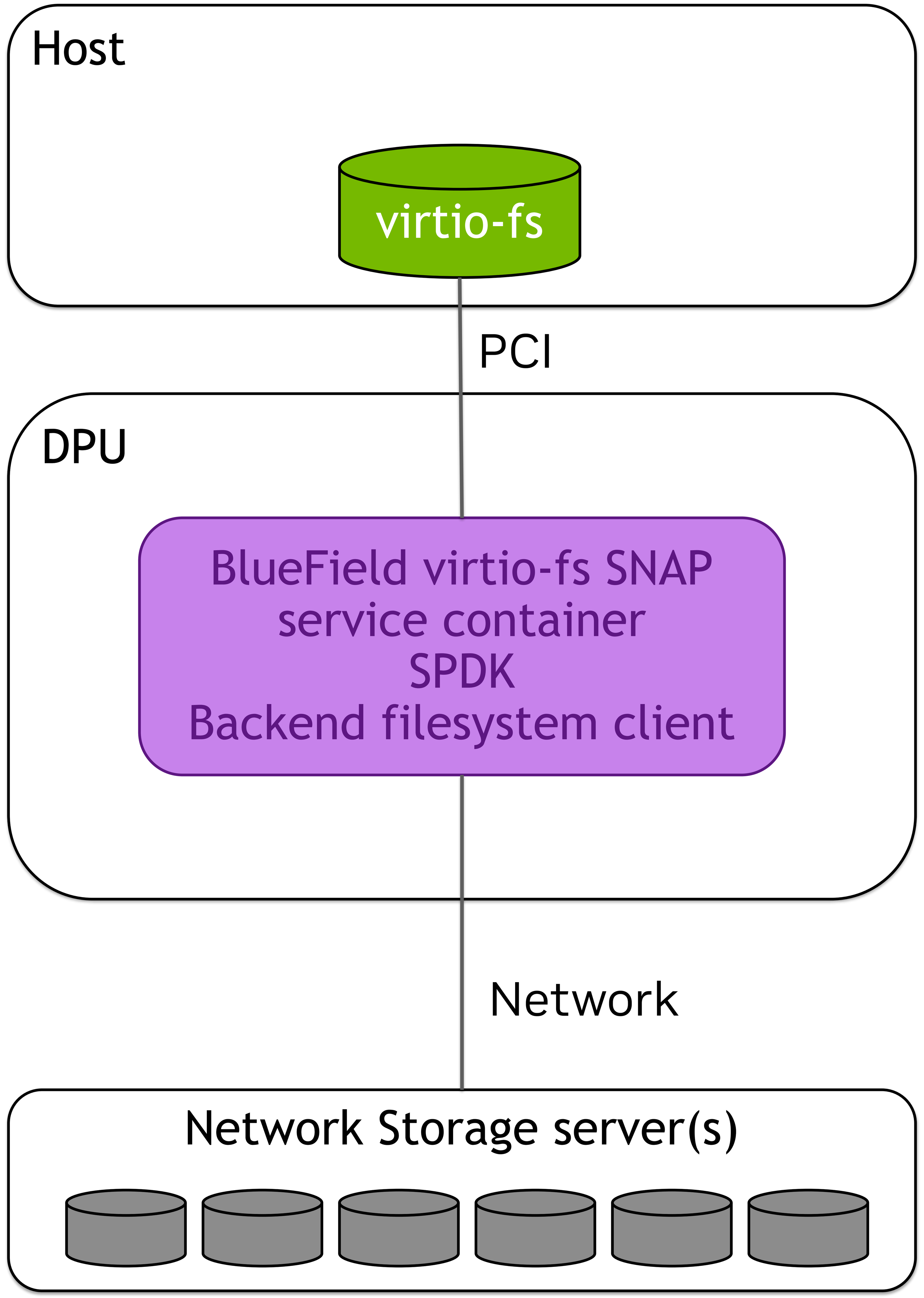
But it may be necessary to materialize data at the client



DOCA SNAP virtio-fs with SPDK

First filesystem implementation for DPU Secure Storage

- DOCA SNAP virtio-fs is a runtime DOCA service and a DOCA library SDK
  - Research project with a POC, not yet on product roadmap
- Presents a secured PCI level file storage end point to the host
  - Mount initiated from DPU side and presented to the host - more secure
  - Hides the actual files system within the DPU
- A new member in the SNAP offering of storage emulated PCI devices
  - DOCA SNAP NVMe and DOCA SNAP Virtio-BLK already exist
- Supports any file storage as backend
  - Standard Linux kernel such as NFS, SMB
  - Internally developed
  - Partners, vendors
- Based on SPDK framework and storage stack
- GDS (GPUDirect Storage) on host side
- DOCA security, zero trust, isolation



DOCA SNAP virtio-fs proposed benefits

Security

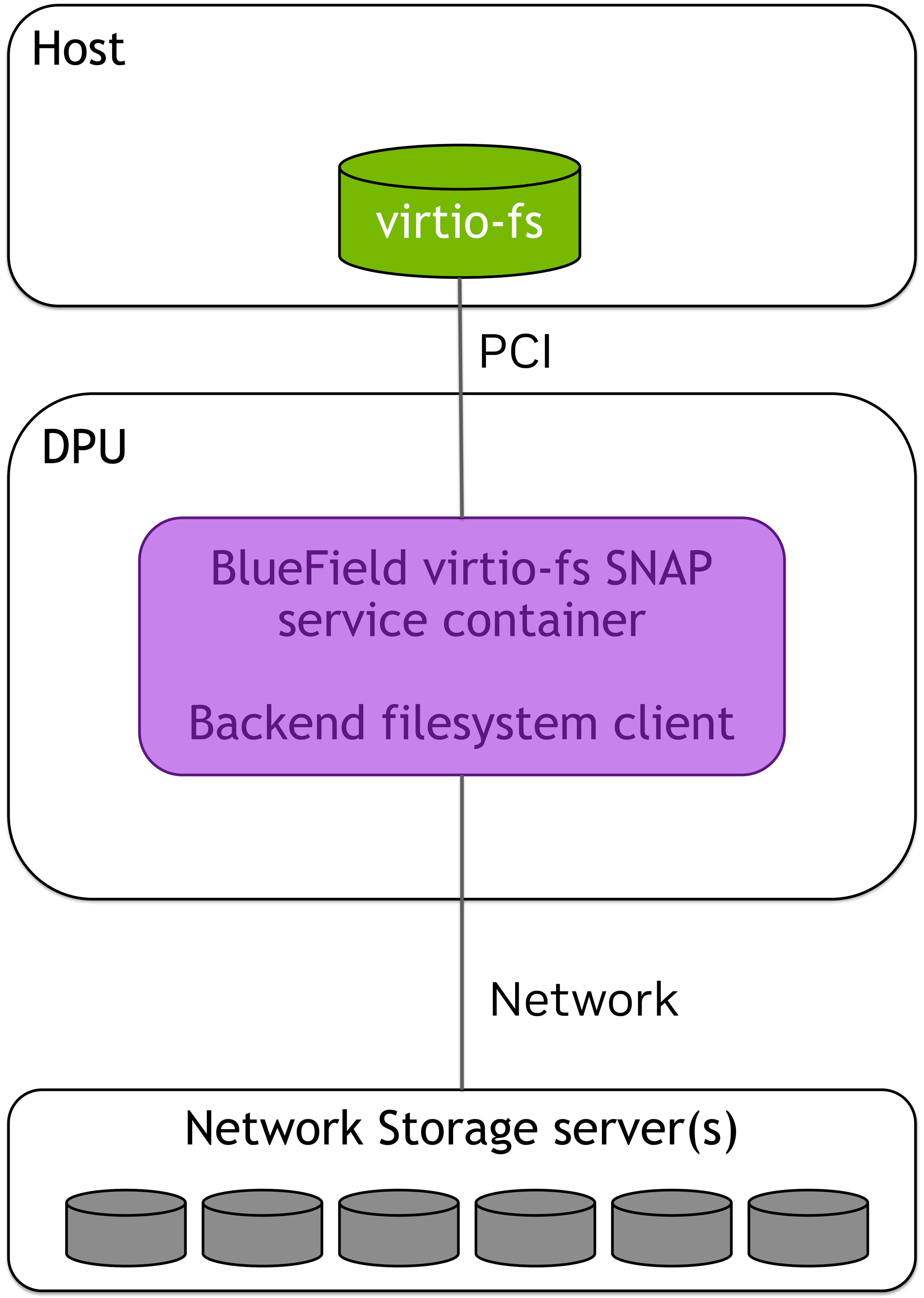
- Zero Trust, centrally enforced to be less subject to supply chain attacks
- Host is not exposed directly to storage network
- Reduced attack surface by providing local storage interface
- Tenant isolation by exposing volumes at PCI level to PFs and VFs
- Cloud provider in full control: exposed volumes, monitoring

Performance

- Offload storage related CPU cycles to DPU
- Use DPU accelerator engines (CRC, Erasure Coding, AES-XTS, …)
- Reduce noise on host
- Full zero copy support

Maintenance

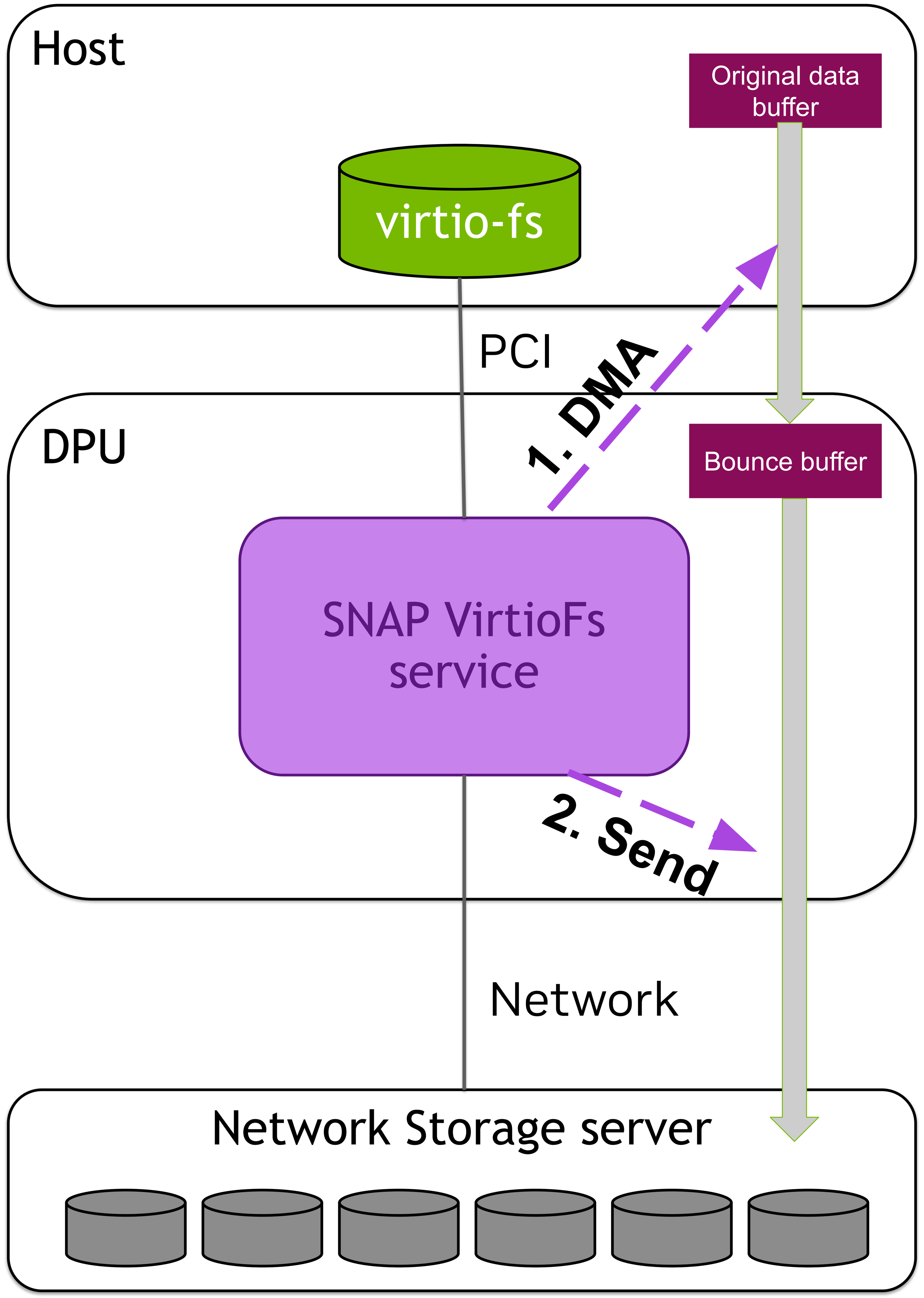
- Easily update/upgrade storage services
- Easily change storage vendors
- Flexibility in kernel dependencies
- Transparent to tenant

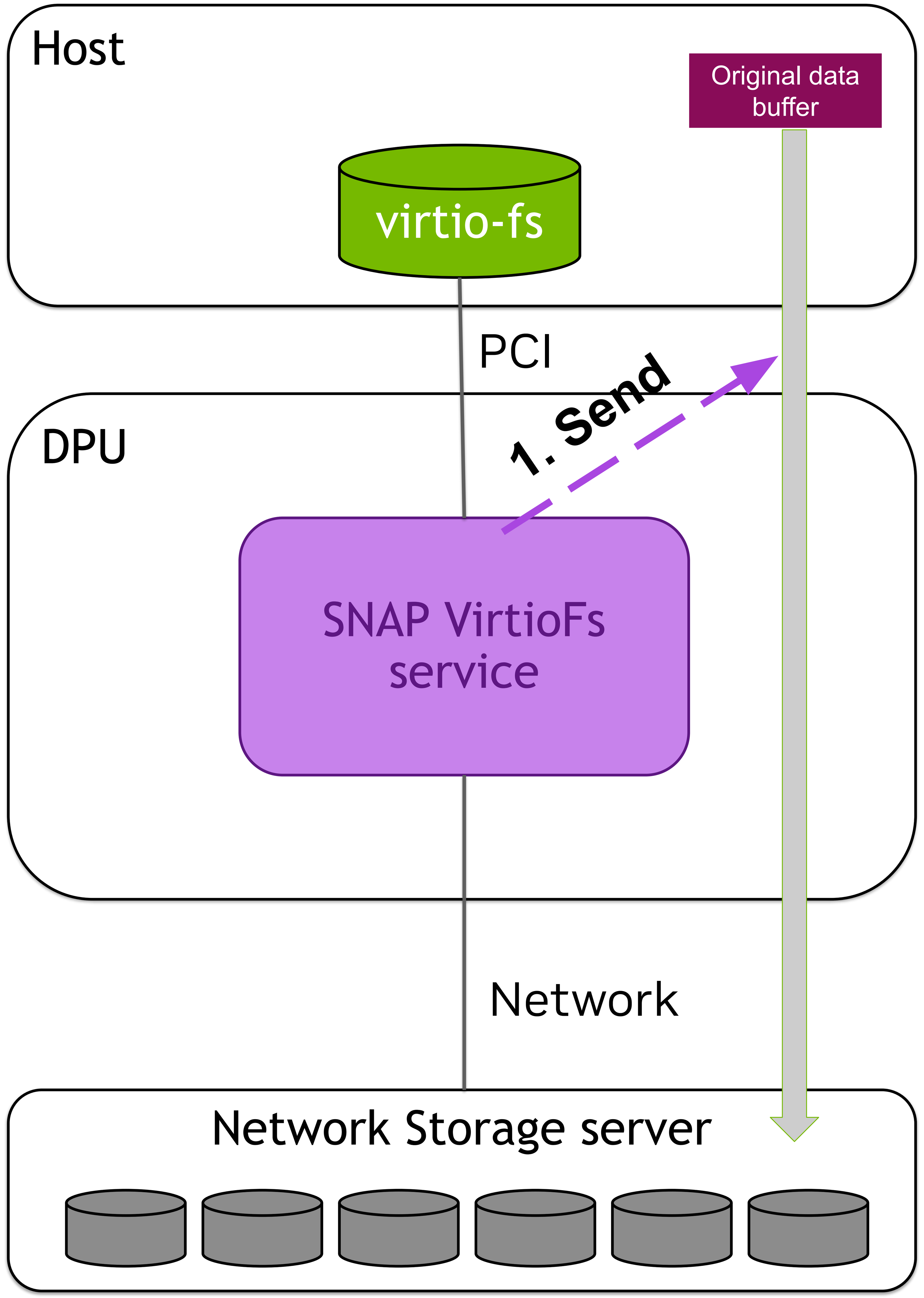


DPU Secure Storage Zero Copy research project

Example WRITE flow that avoids bounce buffer overheads

- Simple case: bounce buffer
  - Same as executing a request originating from the DPU
- Cross-function mkey
  - Enables DPU to use an address from the host's address space when DPU issues RDMA
  - No bounce buffer
  - Transfer is still initiated by the DPU service
  - Only trusted DPU services can gain access to a cross-function mkey
- Zero copy for kernel based DPU filesystems
  - Need a way to pass mkey through VFS for POSIX read()/write() syscalls
- mkey supplied by trusted provider on DPU
  - Consumed by storage clients, smaller attack surface, more reliable
  - Per-transaction, storage server is never exposed to entire host memory
  - Key is invalidated when the transaction ends

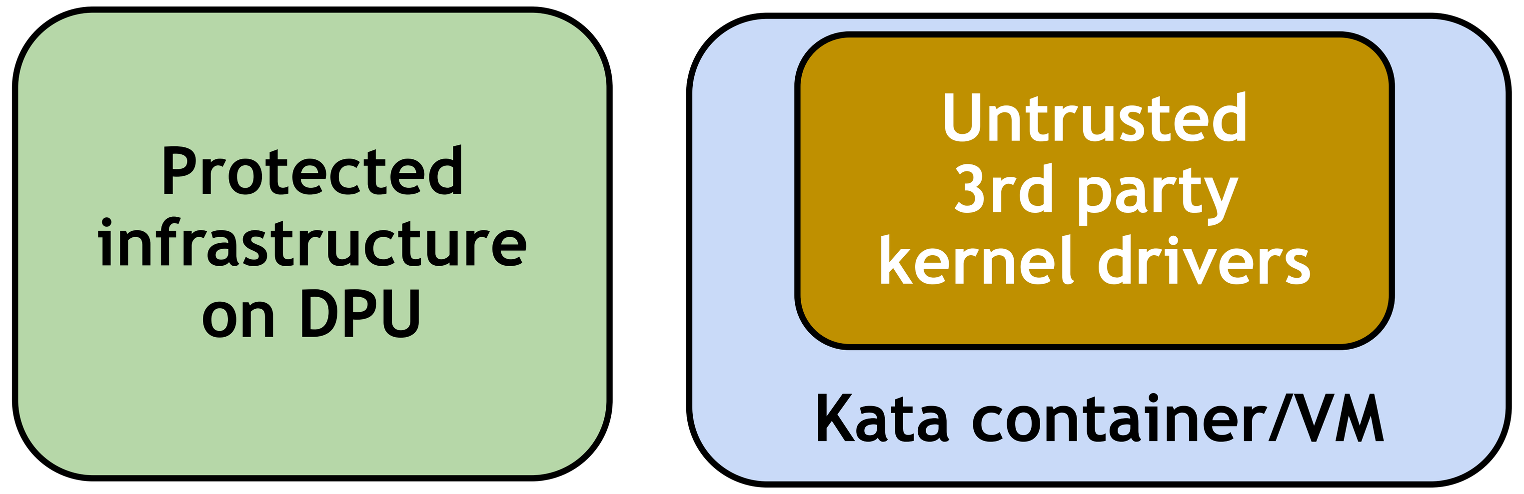


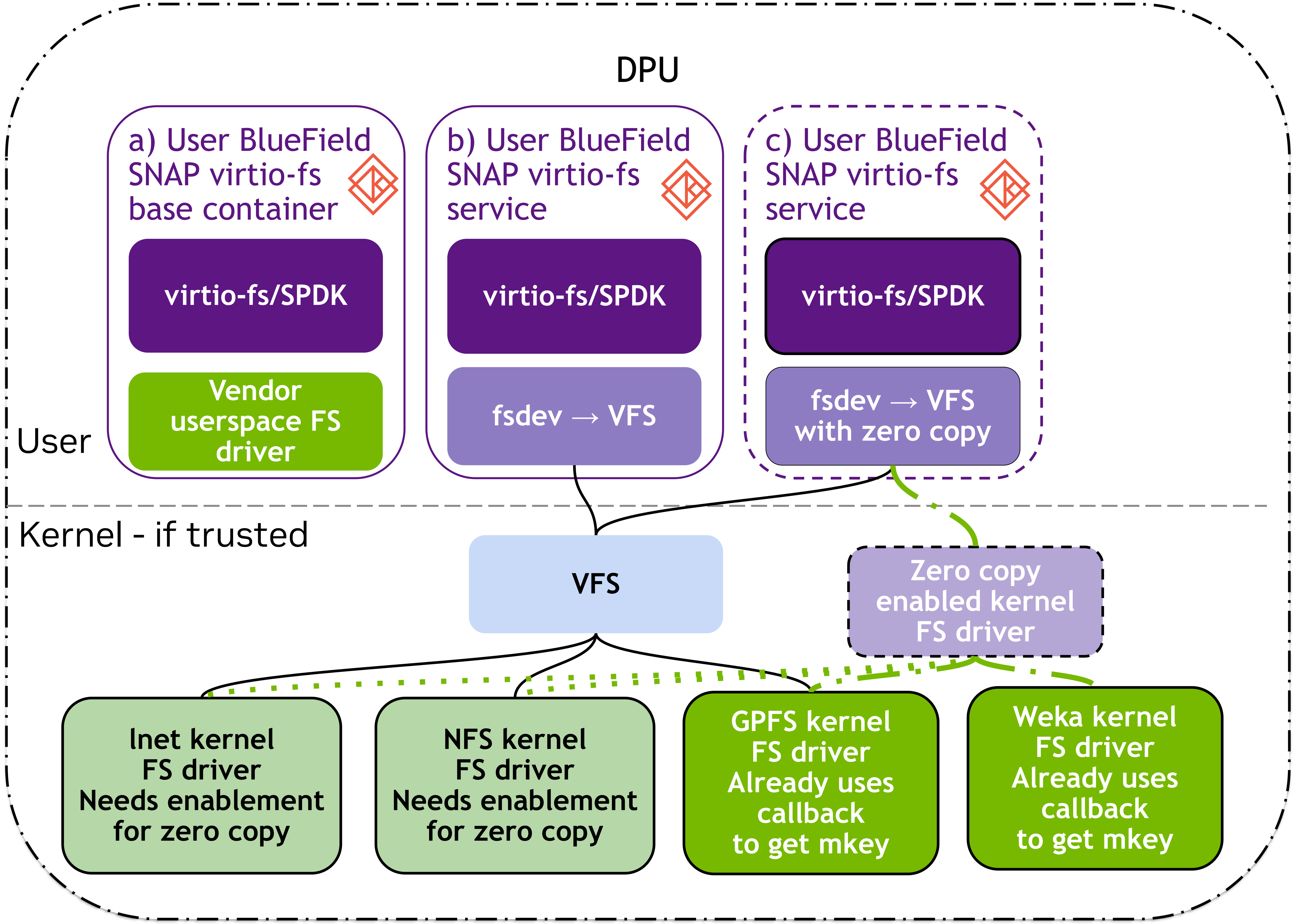


DOCA SNAP virtio-fs SDK for DPU Secure Storage

Research POC in consideration from partners

- DOCA virtio-fs SDK
- Possible approaches over time include
  a. DOCA SNAP virtio-fs base container + vendor storage client at user level
     i. SOL: polling, easy zero copy, no kernel syscalls
     ii. Via SPDK APIs
     iii. Starting to investigate: Weka, VAST, DDN Infinia and others
     iv. Dell, NetApp, Pure have interest in this model for NFS
     v. Zero copy is a performance enhancement
  b. DOCA SNAP virtio-fs base container + DPU kernel driver
     i. Base container forwards to DPU's VFS
     ii. Works for legacy kernel drivers
     iii. No zero copy
  c. NV user-level container connecting to VFS + NV library that links to enabled kernel driver
     i. Can call NV library to get keys for zero copy
     ii. IBM GPFS and Weka already had callbacks to get mkey for GDS, investigating this soln
- DPU could be protected from whole client stack by using Kata containers or KubeVirt





SPDK – FS stack

Parallel to BLOCK stack

SPDK fsdev API

bdev-like but for the file ops

Module management/Control API

- `const char *name`
- `int (*module_init)(void)`
- `void (*module_fini)(void)`
- `int (*config_json)(struct spdk_json_write_ctx *w)`
- `int (*get_ctx_size)(void)`

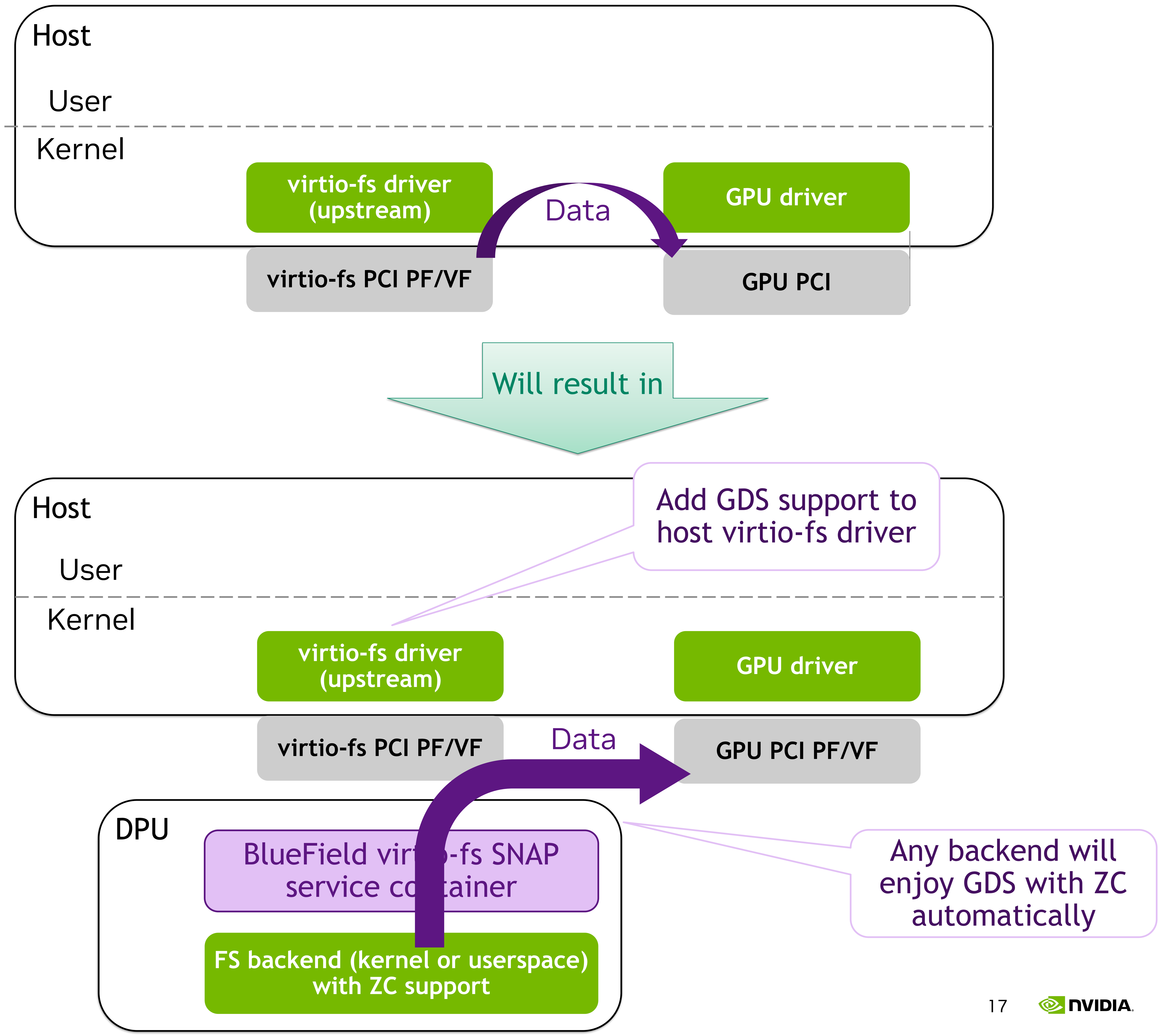
Device ops API

- `int (*destruct)(void *ctx)`
- `void (*submit_request)(struct spdk_io_channel *ch, struct spdk_fsdev_io *)`
- `struct spdk_io_channel *(*get_io_channel)(void *ctx)`
- `int (*negotiate_opts)(void *ctx, struct spdk_fsdev_instance_opts *opts)`
- `void (*write_config_json)(struct spdk_fsdev *fsdev, struct spdk_json_write_ctx *w)`
- `int (*get_memory_domains)(void *ctx, struct spdk_memory_domain **domains, int array_size)`

virtio-fs and GPU with GPUDirect Storage (GDS) enabling

- virtio-fs (host driver) lacks support for GDS
  - Will be added
  - Modifications similar to other kernel storage drivers are needed

- Adding GDS support to virtio-fs host driver to automatically benefits all advantages above
  - Zero copy between GPU and FS backend
  - Storage service security
  - GDS enabling is localized to the host virtio-fs driver and storage clients on the DPU are unaffected



Call to Action for DPU Secure Storage

Watch for more announcements on progress and partnerships!

- File systems vendors
  - Port to and tune for Arm/DPU
  - Model as SPDK driver, register to SPDK FSDEV interface
  - Make use of SPDK “memory domain” APIs to get mkeys representing host buffers for zero copy RDMA transfers
  - Kernel-level solutions are invited to leverage our zero-copy kernel solution as it ripens

- Upstreamed virtio-fs and FUSE enabling
  - Tune host virtio-fs driver, e.g. multi-queue support to relieve a performance bottleneck
  - Explore cache invalidation to alleviate reliance on users opening files in O_DIRECT mode
  - Add support for GPUDirect Storage (GDS) to enable targeting of GPU buffers

- Data centers/customers
  - Evaluate security risks
  - Select a more secure option
  - Elect to reduce OpEx with modest CapEx investment for DPUs

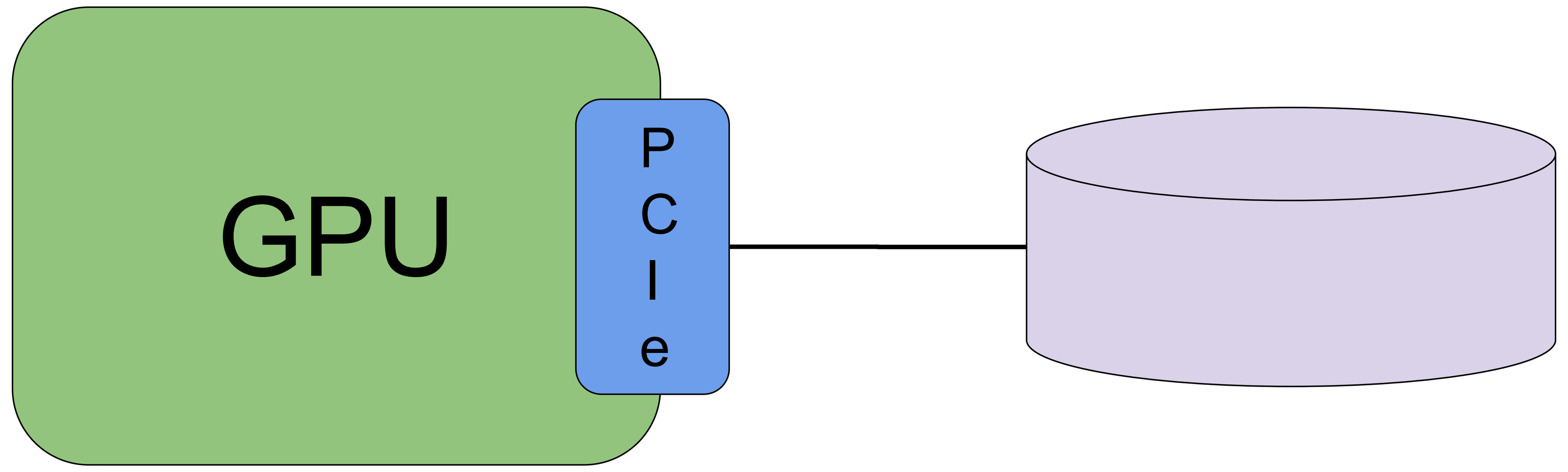
A new class of problems on scaled data

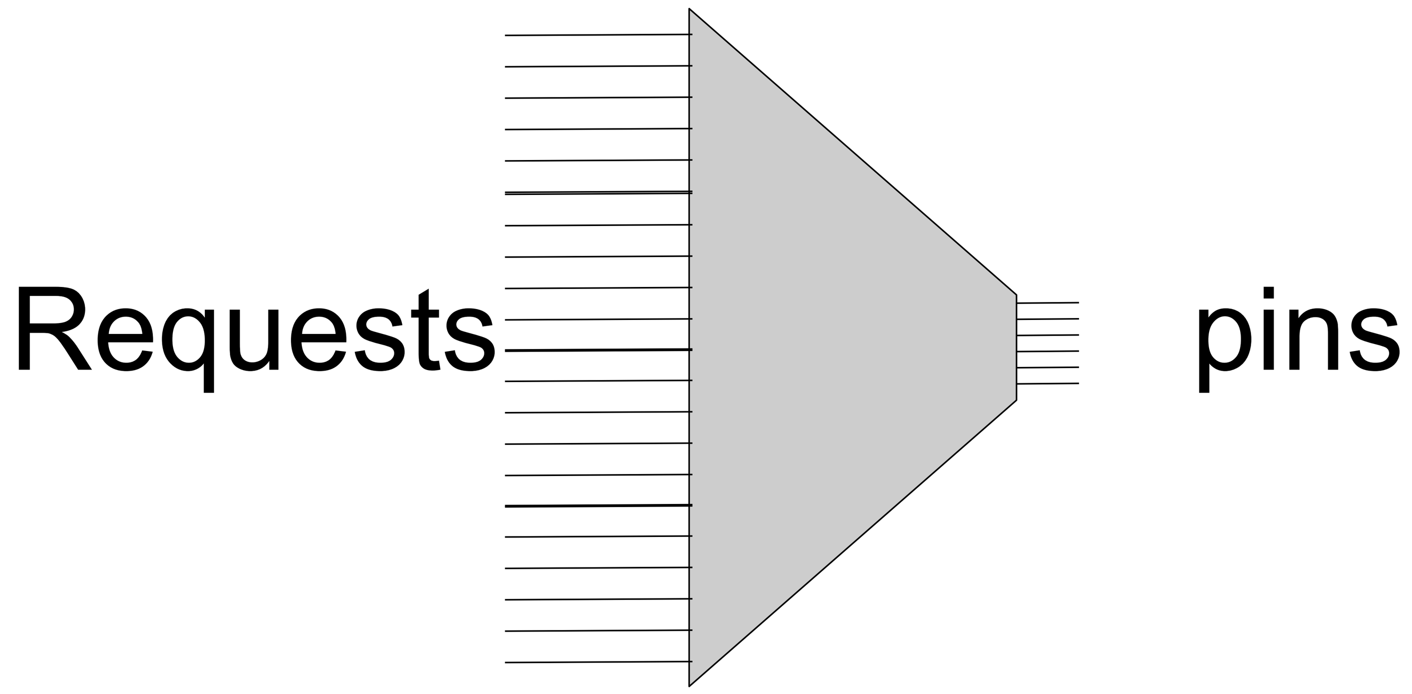
GPU becomes not only a compute monster, but also a fine-grained data access engine
Both use O(100K) threads to accelerate, compute or IO

- Huge data that are too big to reach with loads and stores
  - Partitioning, caching, communication complexity
  - Error handling at scale is problematic
  - NVSHMEM for memory; something more for mem+storage

  → new API family that covers data anywhere
- Accesses are initiated from the GPU (or CPU)
  - GPUDirect Async Kernel Initiated Storage, not just GDS
  - Example: graph traversal based on reading node data
- Vast volume of accesses, 1+ per GPU thread

  → greatest benefit with fine granularity





How do we most efficiently squirt
O(100K) requests/responses through the PCIe pins?

Emerging application domains that motivate a new programming model

Applications

- Graph Neural Networks (GNNs) – graph + feature store
  - Neither the graph nor the data fit into a GPU for 1T edges
  - High-value embeddings for entities and relationships
  - Key parts of recommendation and bad-actor detection systems
  - GNNs improve accuracy over other embedding types
- Vector search/vectorDB – vector store
  - NeMo Retriever, NVIDIA RAFT in RAG-LLM
  - Data deduplication to prep for foundational training of trillion-token LLMs
- LLM fine-tuning joint with GNN embeddings benefits from huge key value service
- Graph analytics available in cuGraph:
  - Personalized pagerank, community detection on huge graphs
  - Distributed sampling and partitioning for GNN models

Common need: simple management of data larger than physical memory of host + device

- Avoid OOM (Out of Memory) errors
- Typically requires caches, partitioning, multi-GPU/multi-node communication
- Needs to be re-created for each application unless we have a common solution

Characteristics and usages across scale

system scale	1 GPU discrete 1-10TB, tabular data Local NVMe	8 GPU HGX 20-40TB, 3D proteins Local or TOR NVMes	256 GPU SuperPOD 100+TB, transaction graph TOR NVMes or RDMA filers
app domain	Data science	Molecular generative AI BioNemo, Pharma	Anomaly detection, RecSys FSI, cybersecurity, retail
usage model	Exploratory data analysis, Model creation, Train a couple of models overnight	Input knowledge graph Build hetero molecular graphs Molecular diffusion inference Docking analysis	Load and build graph Sample and train Inference to create embeddings

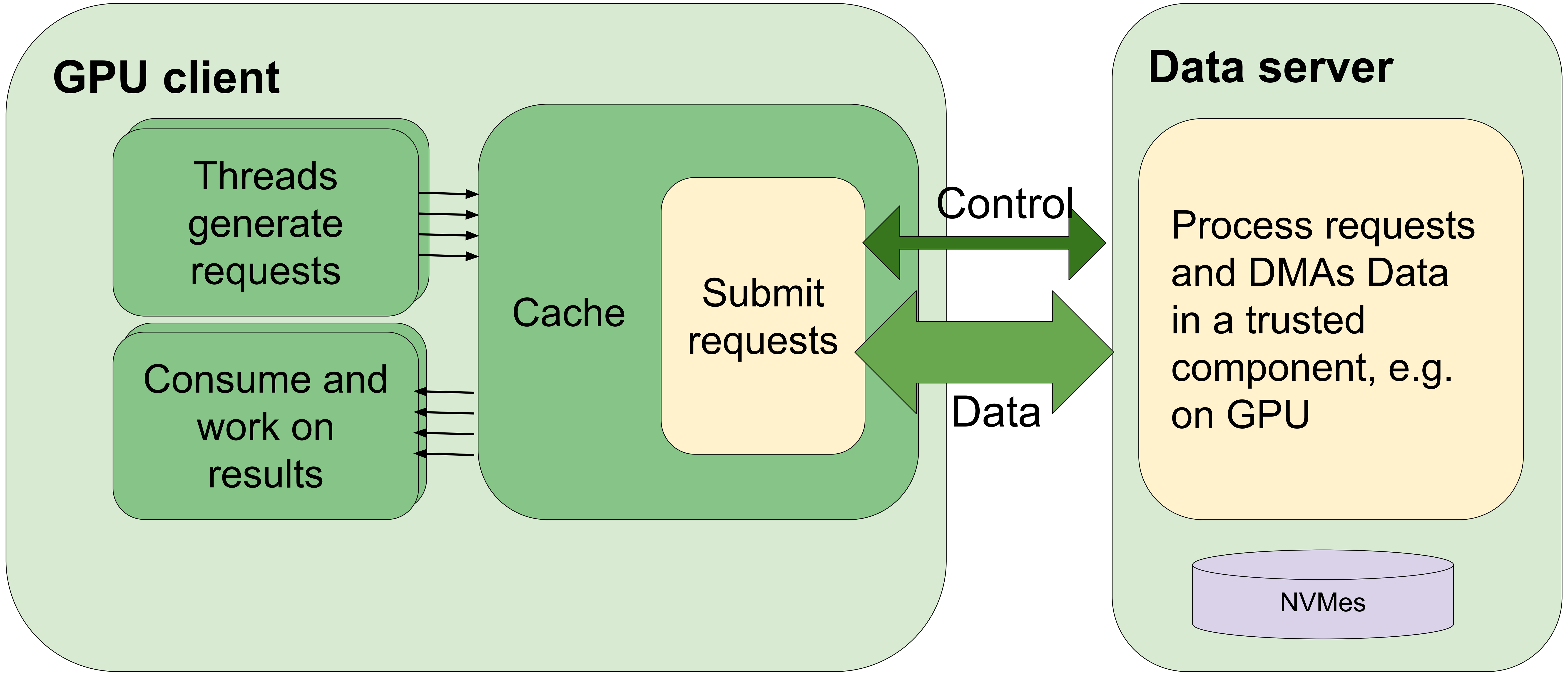
Application layering example

Delivering new capabilities through existing stack

- cuGraph service implementation changes, application does not
- Now training can proceed independent from data size
- No need to manage memory system, caching, partitioning, big improvement in maintainability for GNN training at scale

GPU-initiated scaled data architecture

GPU becomes an autonomous highly parallel data access engine



- Request, initiation, service, and consumption all happen within a GPU kernel
- GDA KI Storage enables data IO accesses that are both initiated and triggered by GPU
- Requests are processed in a trusted, privileged server with access to storage
- Features a key pillar of Magnum IO: flexible abstraction

Rethinking interfaces for the modern data center

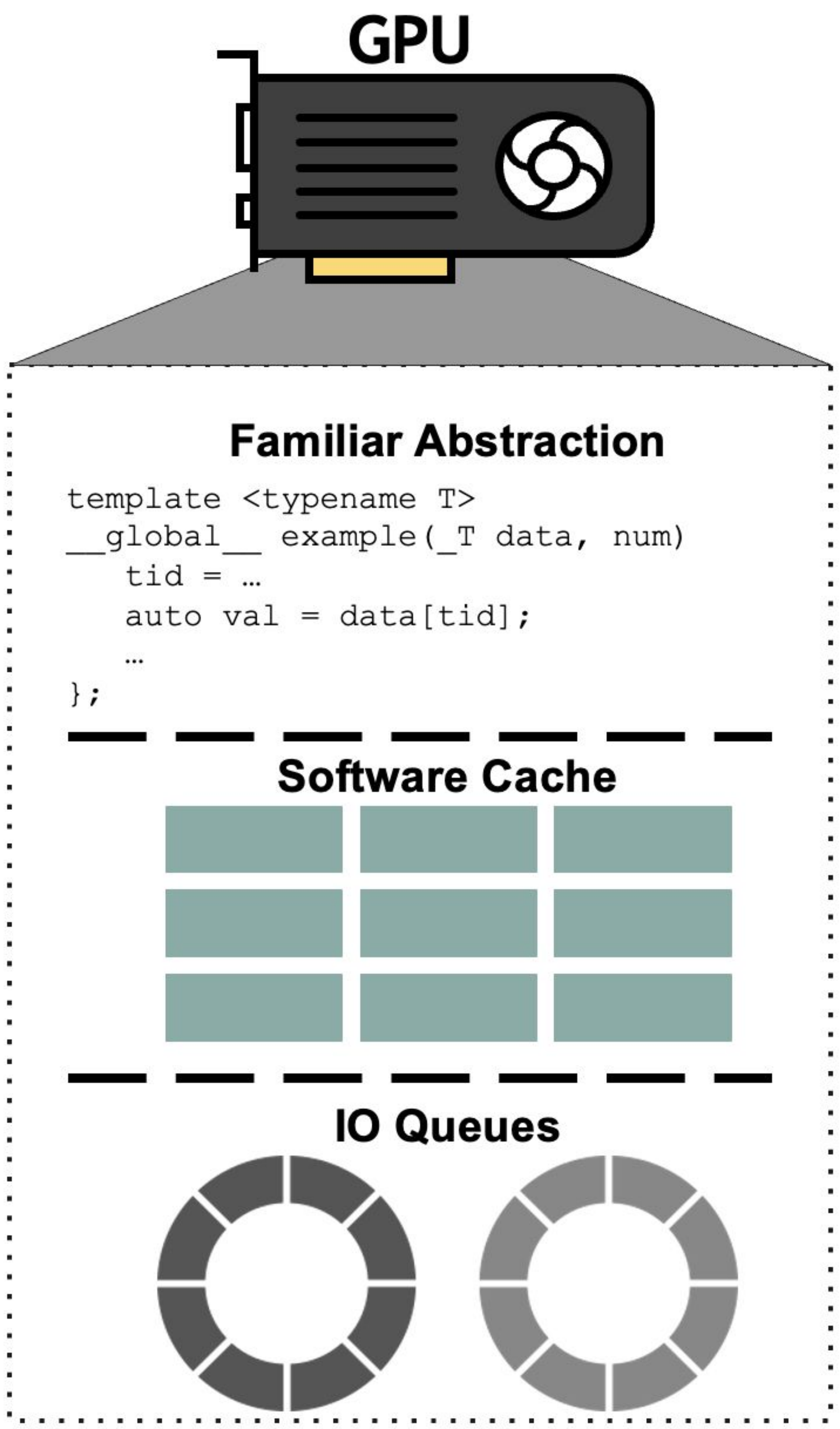
Internal name: **SCADA for scaled accelerated data access**

- **Scale:** Single API for data access independent of scale
    - Fit where you couldn't before, e.g. 10 TB in one node, avoid OOM worries
    - Transparently scale both data set size and size of compute cluster
- **Higher abstraction:** "Serverless access" is the way of the modern data center
    - Front end: handle caching, avoid partitioning, communicate among multi-GPU/multi-node
    - Back end: app accesses dataset X, relegates details of where/how data is stored
    - Data platform tools could manage curation, locality, sharding, staging
    - Acceleration with best use of GPU threads, memory management, and topology-tuned communication
- **Easy enablement:** Low-level interface that leaves application layers unchanged
- **Fundamentally-low TCO:** Reduce the cost of storage data
    - Huge data → huge memory → huge cost
    - Applications of low computational complexity use HBM only for memory vs. compute
    - Cheap NVMes make datacenters more efficient

Two SCADA research prototypes: BaM, GIDS

Preparing for trial integration into production stacks

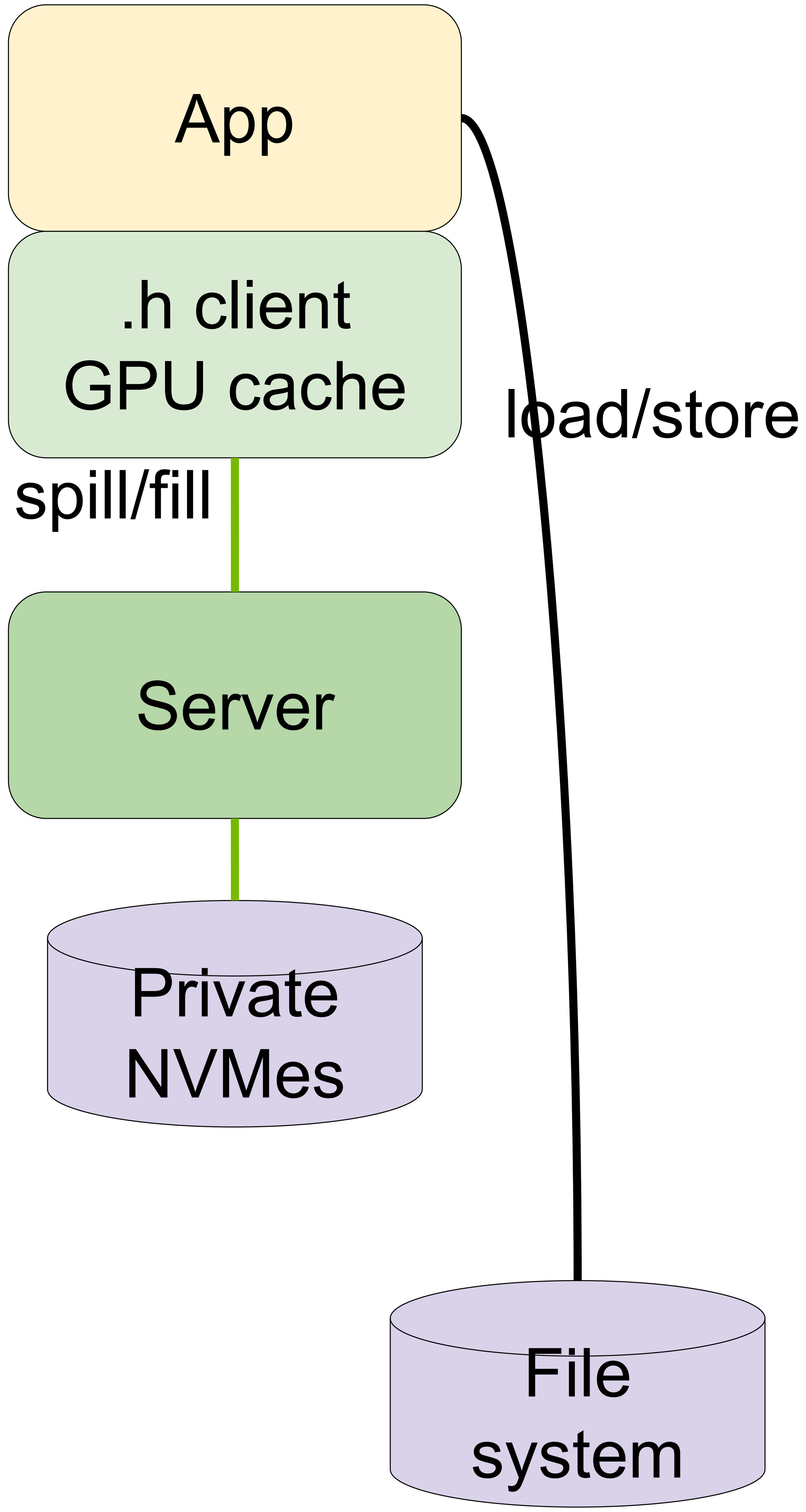
- Follow-on to an earlier OSS academic prototype
  - **Big accelerator memory, BaM**: “GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture”, ASPLOS 2023: https://doi.org/10.1145/3575693.3575748
  - **GIDS**: “Accelerating Sampling and Aggregation Operations in GNN Frameworks with GPU-Initiated Direct Storage Accesses”, VLDB’24: https://arxiv.org/abs/2306.16384
- Currently a functional prototype, first used by cuGraph
  - Easy integration into widely used package manager
- Templated C++ header library, specialized for app objects
  - Familiar programmer abstractions
- GPU cache aggregates to a smaller number of IOs
- Optimizing IO queue interactions for O(100K) GPU threads



Progression of SCADA services

Start with simpler cases, grow over time with your input

- Common infrastructure
  - Header-centric client library in front of opaque implementation
  - Memory managed by app, SCADA provides APIs to allocate and free
- Start with a simple but critical service like swap, then extend
  - App on CPU reads all data from storage into GPU, as it always has
  - GPU threads write data into SCADA and read it back later
  - Relieve out of memory (OOM) avoidance with unbounded capacity
  - API for contiguous arrays
- We'd love your feedback for the next APIs and services

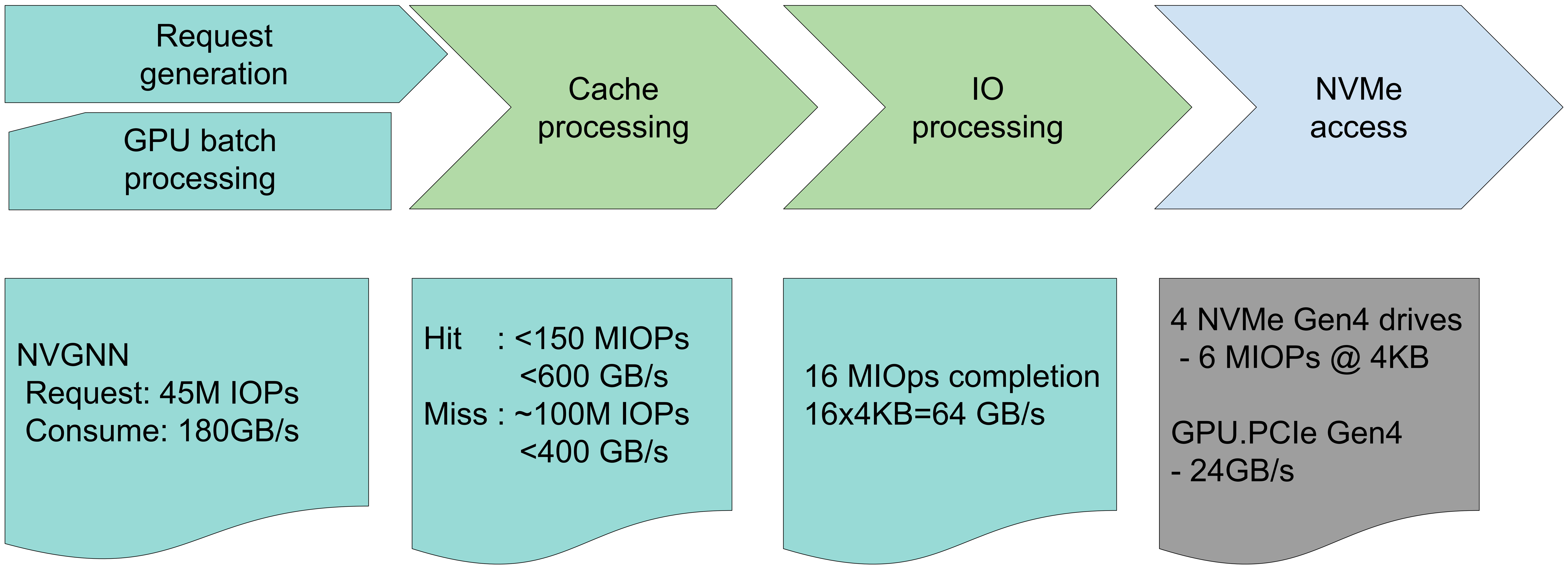


Performance results: the GPU as a data access monster

Bottleneck is NVMe and pin bandwidth, not GPU code

GPU tput on a single A100
6910 CUDA cores @1.41GHz

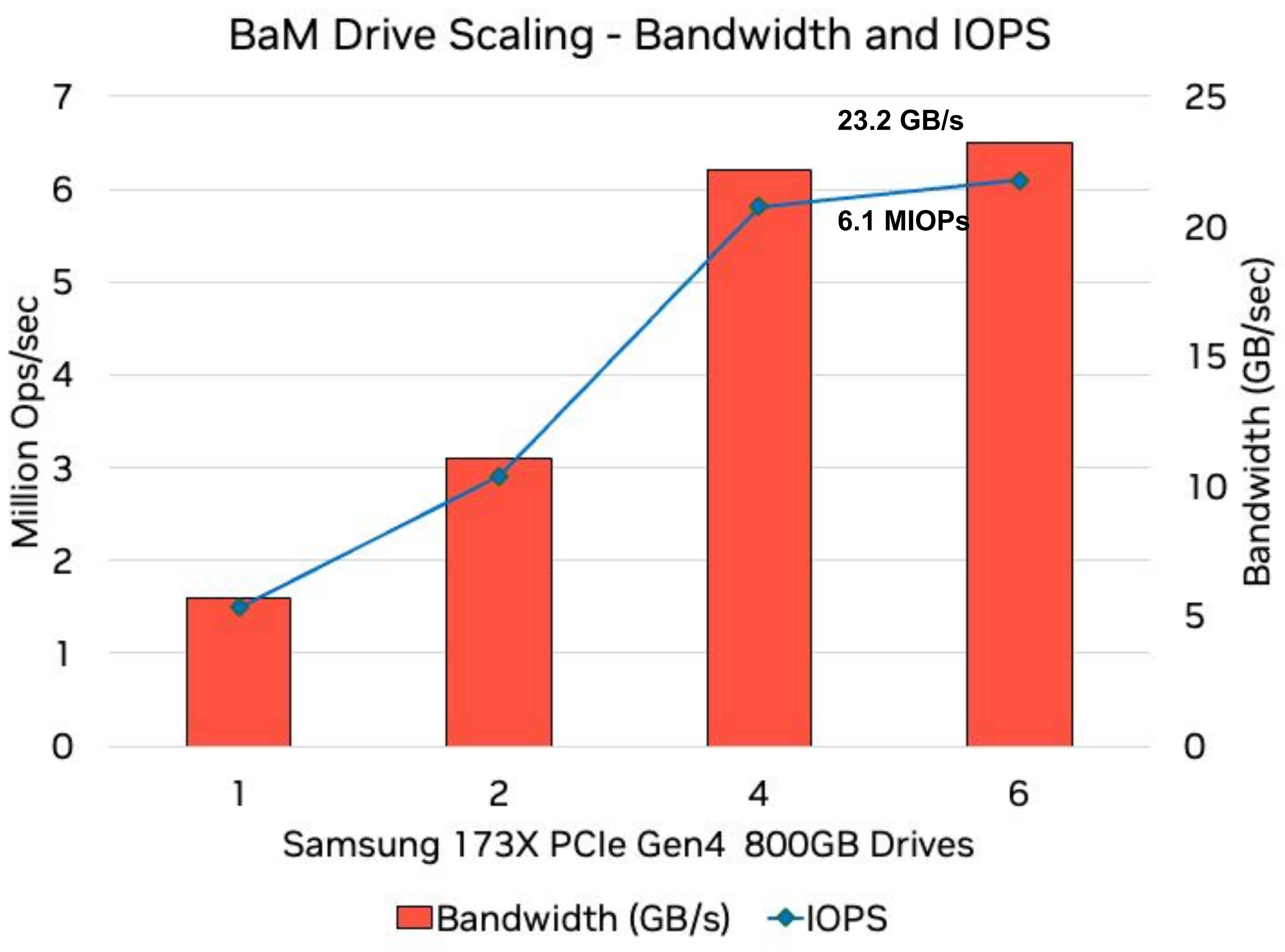
Transfer size = 4KB



- Data lookup acceleration enables higher throughput by reducing the IO bottleneck to (feature) data
  - Transparent data reuse benefit: cache bw (400-600 GB/s) >> PCIe into the GPU (24 GB/s for Gen4)
  - IO processing (16 MIOPs) keeps up with PCIe-saturating NVMe IOPs rates (6 MIOPs for Gen4)
- GPUs are latency tolerant - HW context switching covers miss latency

Random reads mostly saturate PCIe Gen4 with 4 Gen4 drives

Initial Big Accelerator Memory (BaM)* research prototype validates perf trends



DGX A100 GPU

- UIUC-NVIDIA BaM replaces the NVMe driver to enable GPU-initiated IO transfers to/from NVMe
    - 6+ Million IOPs, 23+ GB/s on 4KB random reads
    - 0% CPU utilization
- BaM NVMe Block Bench
    - Microbenchmark to stress storage through BaM
    - Scales GPU requests at 4KB against storage devices and measures operations/s and GB/s in 4KB xfers.
    - 6 drives vs. 4 bumps up MIOPs and GB/s slightly

BaM and GIDS are UIUC-NVIDIA Research prototype projects and not intended for general release.

Explicit storage IO is 25x of mmap, faster media is better

Direct GPU access vs. faulting through CPU to storage with BaM and high-performance Gen5 NVMe™ brings 25+x for GNN Training

Feature Aggregation depends on SSD performance
It's 99% of execution time in the baseline, 80% of tuned
Sampling and training depend on GPU performance

Workload Execution Time (seconds)	Baseline (mmap) Gen4/A100	BaM Enabled Micron Gen5/H100	BaM Enabled Micron Gen4	Gen5 v Gen4 Performance
Feature Aggregation	1,130 (99%)	18.6 (83%)	35.0	**2x**
Training	2.1 (0.2%)	0.73 (3%)	3.6	**5x**
End-to-End	1,137	22.4	43.2	**2x**
E2E Improvement over Baseline		**50x**	**26x**	
Feature Aggregation Improvement		**61x**	**32x**	

GIDS with IGBH-Full training. NVMe performance results measured by Micron's Data Center Workload Engineering team, baseline (mmap) performance results measured by NVIDIA's Storage Software team on a similar system.
Systems under test: Gen4: 2x AMD EPYC 7713, 64-core, 1TB DDR4, Micron 9400 PRO 8TB, NVIDIA A100-80GB GPU, Ubuntu 20.04 LTS (5.4.0-144), NVIDIA Driver 535.113.01, CUDA 12.2, DGL 1.1.2
Gen5: Dell R7625, 2x AMD EPYC 9274F, 24-core, 1TB DDR5, Micron Gen5 SSD, NVIDIA H100-80GB GPU, Ubuntu 20.04 LTS (5.4.0-144), NVIDIA Driver 535.113.01, CUDA 12.2, DGL 1.1.2
Work based on paper "GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture" https://arxiv.org/abs/2203.04910 and using https://github.com/ZaidQureshi/bam

GNN on GPU induces queue depths 10-100x of CPU

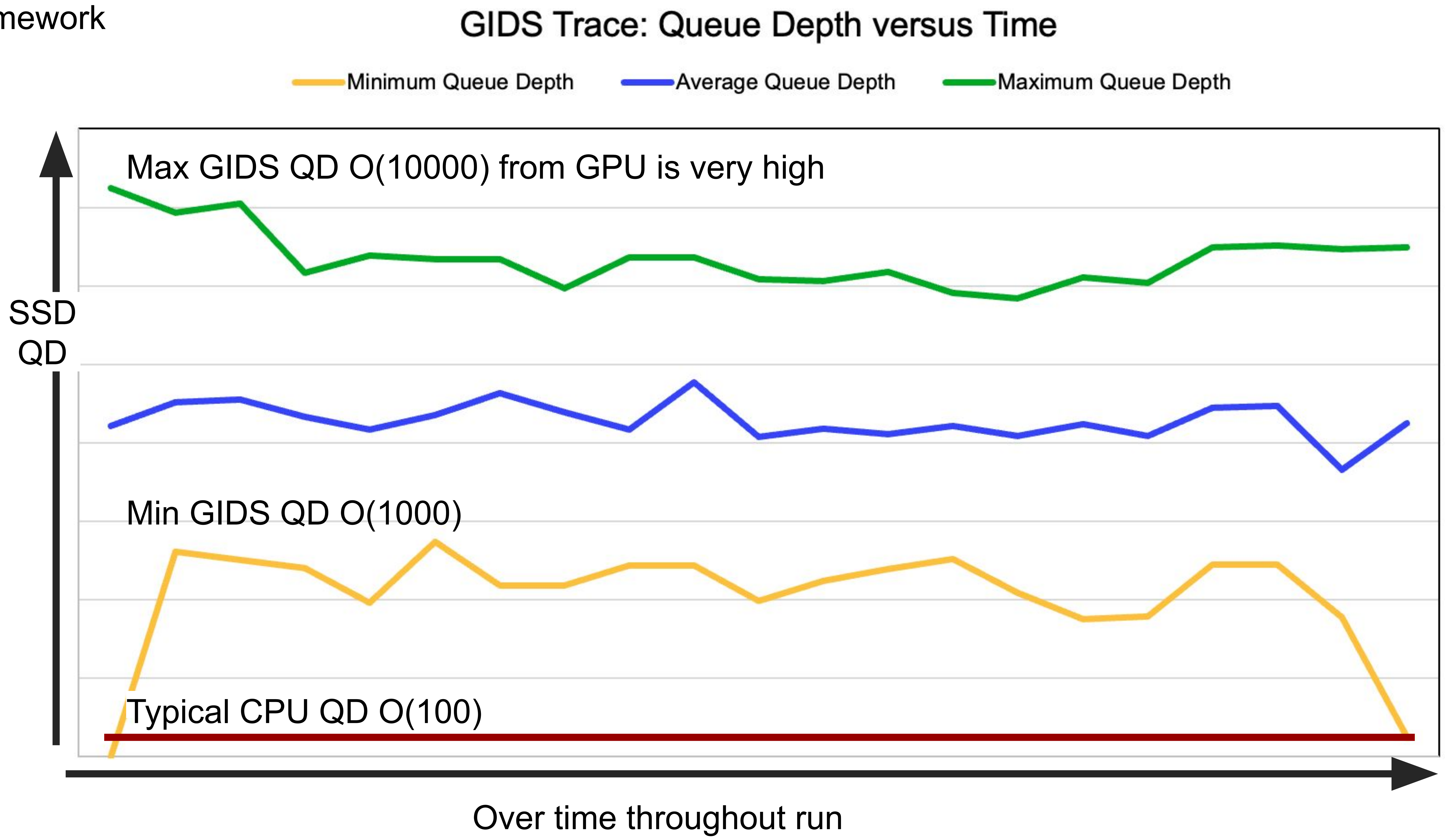
Investigated with Micron NVMe™ IO Trace tool

Using **GPU-initiated direct storage (GIDS)** framework

A trace of the IO pattern at the SSD level shows interesting behavior:

- Near drive's max IO performance
- 10-100x SSD queue depth wrt CPU
- 99% small block reads

GIDS with BaM presents a challenging SSD workload:
High-Performance NVMe is Required



GIDS with IGBH-Full training. NVMe IO trace measured by Micron's Data Center Workload Engineering team.
System under test: 2x AMD EPYC 7713, 64-core, 1TB DDR4, 4 Micron 9400 PRO 8TB, 1x NVIDIA A100-80GB GPU, Ubuntu 20.04 LTS (5.4.0-144), NVIDIA Driver 535.113.01, CUDA 12.2, DGL 1.1.2
Work based on paper "GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture" https://arxiv.org/abs/2203.04910 and using https://github.com/ZaidQureshi/bam

Call to action for SCADA

Come help chart the future of turning the GPU into a data access engine

- App developers and users
  - Share need for more data capacity than will fit in GPU-CPU memory for compute
  - Specify kinds of services of interest, e.g. array, swap, key-value, VectorDB, dataframe?
  - Specify details on product stack support, deployment models
- Infrastructure developers
  - Layer on SCADA as has been done for NVSHMEM, e.g. Kokkos perf-portable framework
  - Look at new venues for fine-grained interleaving of compute and communication, e.g. LLNL
- Storage vendors
  - Support higher IOPs on finer-grained transactions
  - Participate in implementing and showcasing SCADA
  - Special thanks to Micron, Kioxia, Samsung, Western Digital who've shared drives with us to experiment with in our experimental clusters like ForMIO (for Magnum IO)