SNIA DEVELOPER CONFERENCE
SDC 24
BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

Storage Acceleration via Decoupled SDS Architecture

Arun Raghunath (Principal Engineer)
Intel Corporation

Current Cloud Storage Architecture

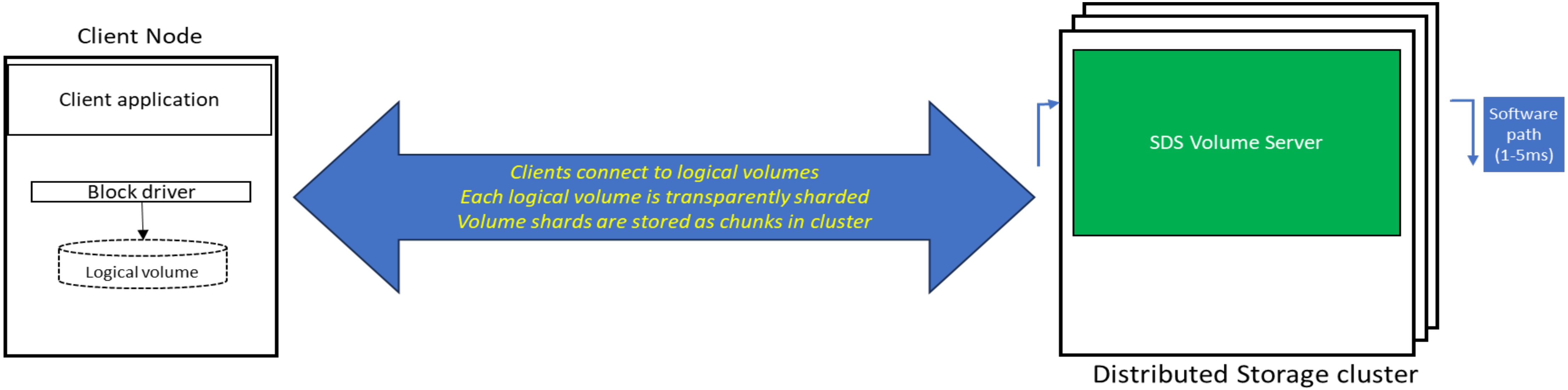


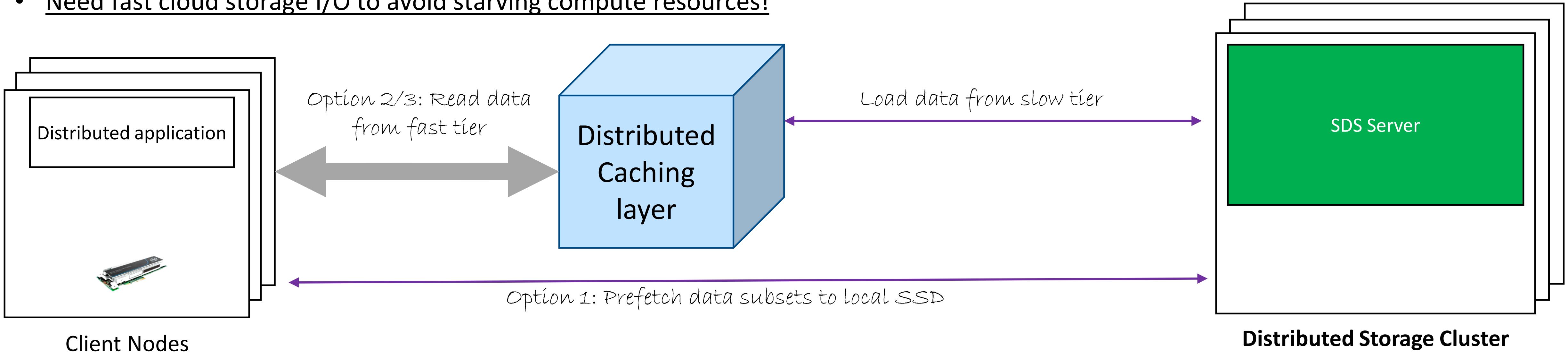
Figure: Cloud Block Storage service provided by Software Defined Storage (SDS) middleware

- ✓ Cloud storage services provide scalability, high availability, durability, reliability
- ✓ Internally, a cluster of servers is used, with each server offering up its drives to implement a given service
- ✓ Middleware aka Software Defined Storage (SDS) running on each server work together to provide storage services

- Cloud storage is slow (millisecond latency, 10K-100K IOPS throughput)
  - Underlying drive performance: microsecond latency, throughput in the millions of IOPS

Fast cloud storage matters

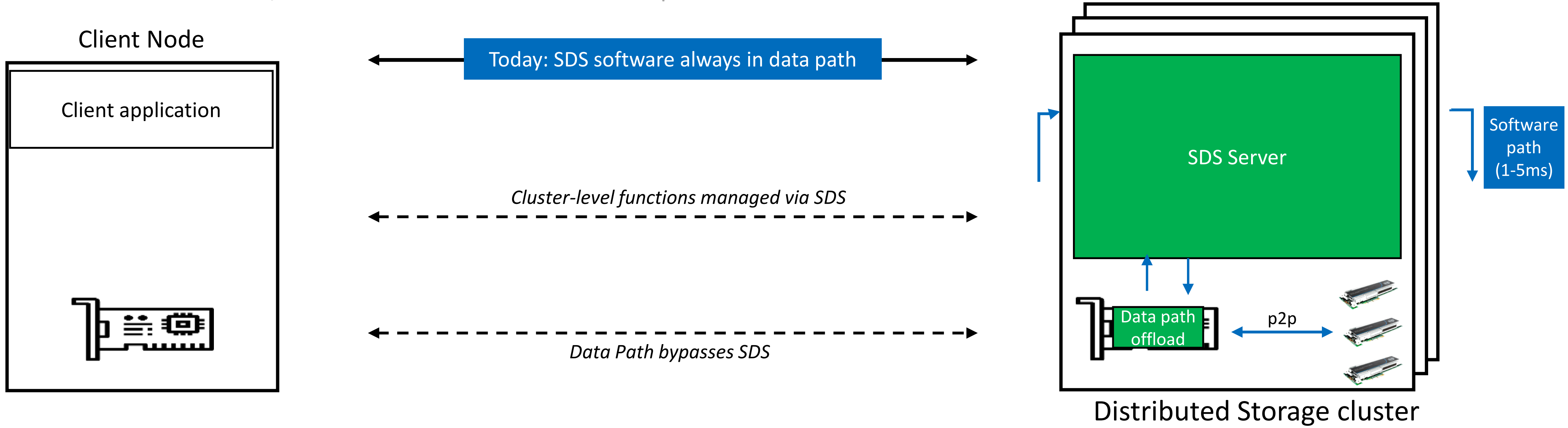
- Many important usages have read-only/read-heavy IO patterns over massive datasets
  - ✓ Deep Learning Training
  - ✓ Big data analytics
  - ✓ web search
- These usages are compute-intensive, hence their jobs are scaled out across many nodes
- Huge data footprint (multi-GB/TB) + multi-node access → persist data in cloud storage
- Need fast cloud storage I/O to avoid starving compute resources!



- Current practice adds significant cost and complexity
  - ❖ Option 1: Provision instances with local SSD. Prefetch and manage data subsets
  - ❖ Option 2: Pay for cache product/service
  - ❖ Option 3: Create/manage custom distributed caching layer

Solution approach: Decouple data path

- Question: Can we bypass the bottleneck SDS software path for data?
  - SDS software must handle cluster-level functions..
  - But does actual data read/write have to traverse the same IO path?



Basic idea: Skip SDS stack completely for IO requests!
A companion module handles IO requests.

On the backend storage target server,

- ✓ SDS software "shares" on-disk location of data with an offload
- ✓ **The data path offload caches logical-to-physical mappings** *under SDS control*
- ✓ On receiving an IO request, the offload directly accesses data on disk

Can we use hardware offloads to create a direct hardware path to media on storage target?

Software PoC design goals

- SDS choice: Ceph is open-source, widely deployed in the industry.
  - ✓ Offers various modes, interfaces, deployment options.
  - ✓ Multiple levels of indirections & translations. Block-based logical volumes created over objects
  - ✓ PoC with Ceph lends credibility to design

- SDS architecture changes: Decoupling requires datapath to be in sync with SDS control plane
  - ✓ Devised protocol to share mappings, evict on updates.
  - ✓ Modeled on PCIe-ATS with an eye towards future hardware IP

- Memory footprint: hardware offload memory is typically a scarce resource
  - ✓ We started with a compression-based approach trying to fit all mappings in memory
  - ✓ Design has evolved to a mapping table cache in offload memory holding a subset of overall mappings

- Characterize and understand interactions between data plane and cluster-level operations
  - ✓ Identify bottleneck paths between offload and host CPU
  - ✓ Interface recommendations for offload and for SDS

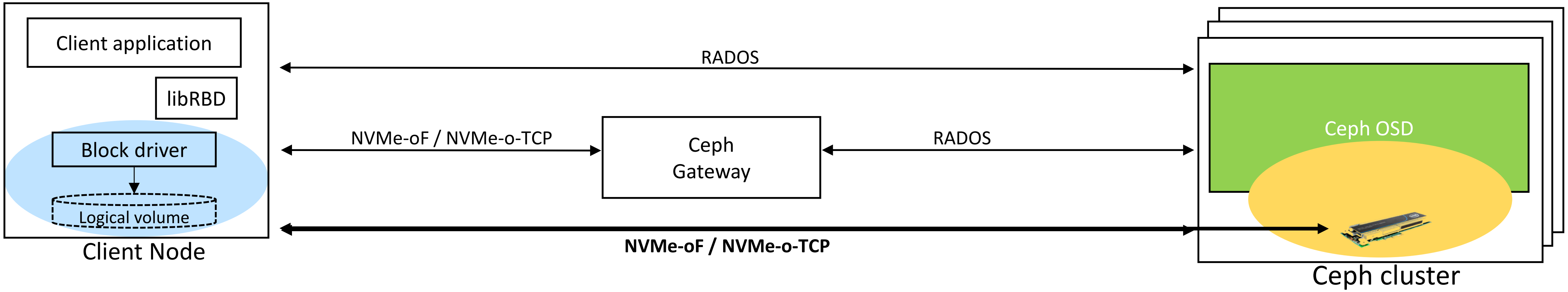
PCIe-ATS: PCIe Address Translation Services

NVMe-oF paths into Ceph

- Goal of Ceph NVMe-oF gateway project,
  - ✓ Enable bare metal clients to connect to Ceph
  - ✓ Reduce client CPU overheads to access Ceph
- What about performance?
  - ✓ NVMe drive performance in order of 10s of microseconds
  - ✓ Remote access over NVMe-oF in order of 10-100 microseconds

Separate control and data paths in Ceph architecture to improve performance

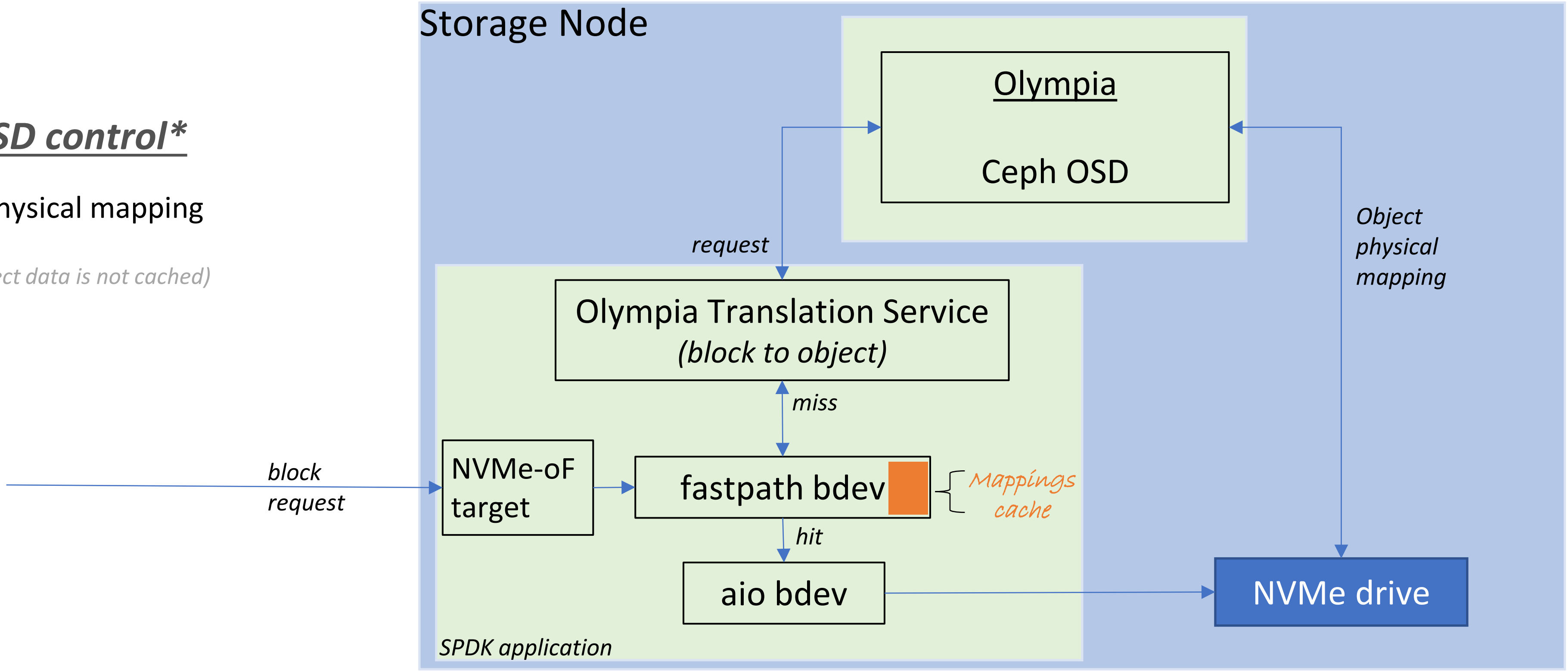
- Extend NVMe-oF path to the target server running a Ceph Object Storage Daemon (OSD)
  - ✓ Initiator side: How to locate the server within the cluster responsible for the addressed logical extent
    - ADNN (Adaptive Distributed NVMe-oF Namespaces) standardization ongoing
    - Custom out-of-band mechanisms being used by a few tier-1 cloud providers for their storage services
- Extend NVMe-oF path all the way to the drive
  - ✓ Target side: Data path within target server. *Focus of the software PoC*



Create an end-to-end data path from remote client over NVMe-oF to the drive managed by a Ceph OSD

RADOS: Ceph client protocol (Reliable Autonomic Distributed Object Store)
RBD: RADOS Block Device

Software PoC Details



Direct read path to drive ***under OSD control***

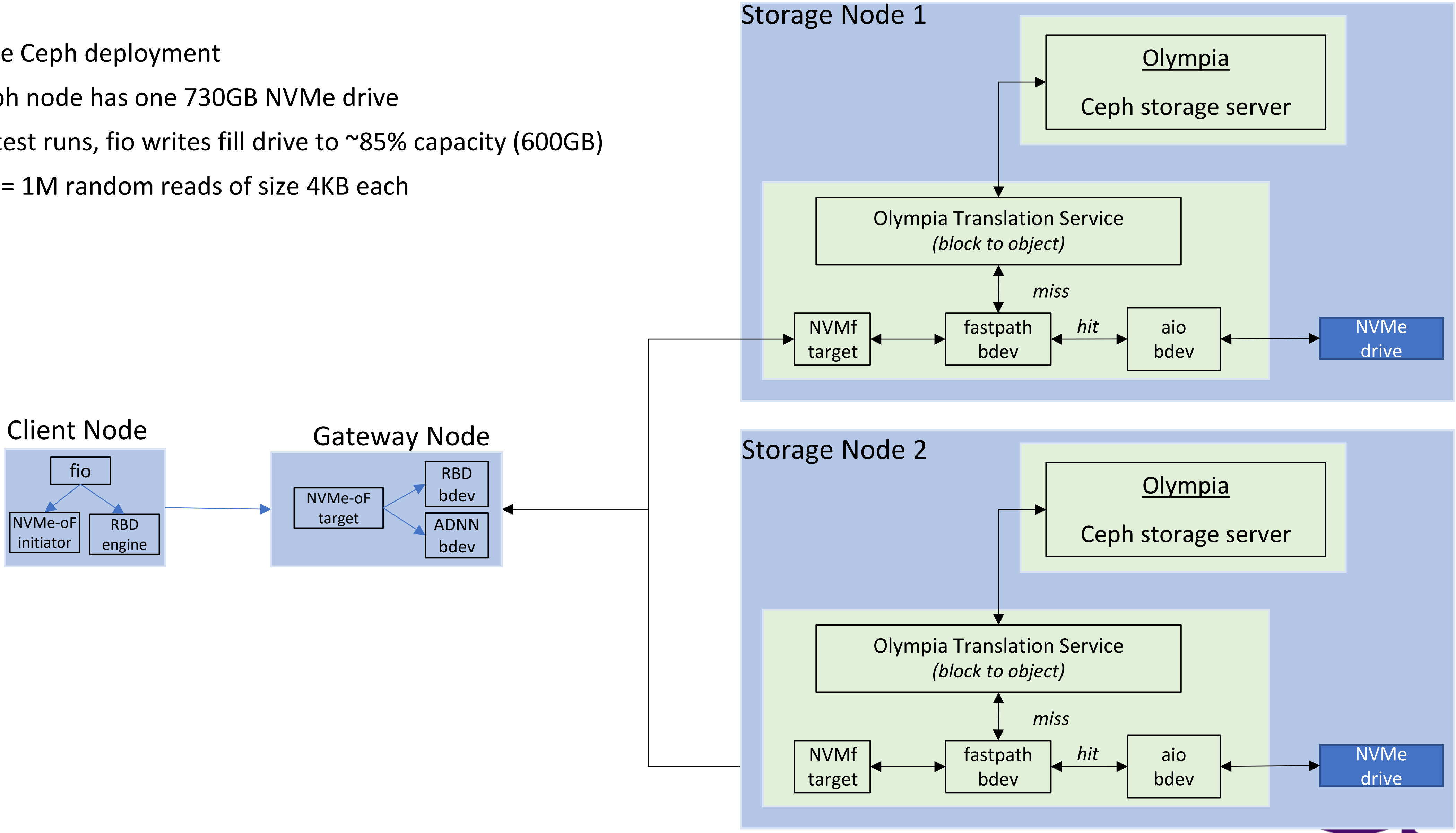
- Patches to Ceph OSD to share logical-to-physical mapping
- Mappings cached by SPDK application *(object data is not cached)*

- SPDK fastpath bdev consults cached mappings.
  - On a hit, fastpath bdev reads directly from drive
  - On a miss, SPDK fastpath bdev requests Olympia Translation service (OTS) for physical mappings
- Olympia Translation Service (OTS) identifies the object backing a client block request
- Olympia modifications to Ceph OSD locates object on the drive and shares the physical mapping with the SPDK application
- OSD evicts mappings when the object is moved on drive

SPDK: Storage Performance Development Kit

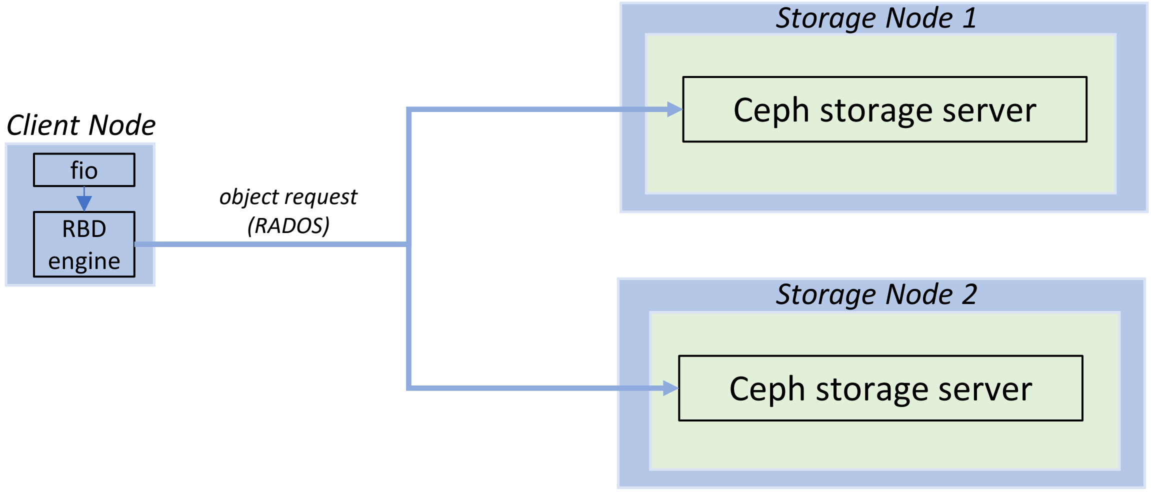
Software PoC experiment setup

- Two node Ceph deployment
- Each Ceph node has one 730GB NVMe drive
- Prior to test runs, fio writes fill drive to ~85% capacity (600GB)
- Test run = 1M random reads of size 4KB each

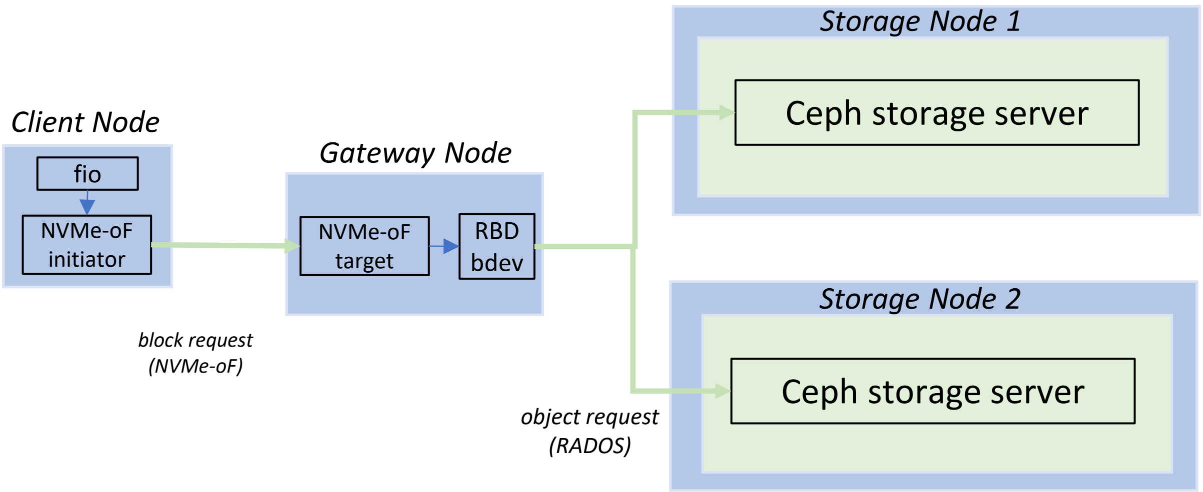


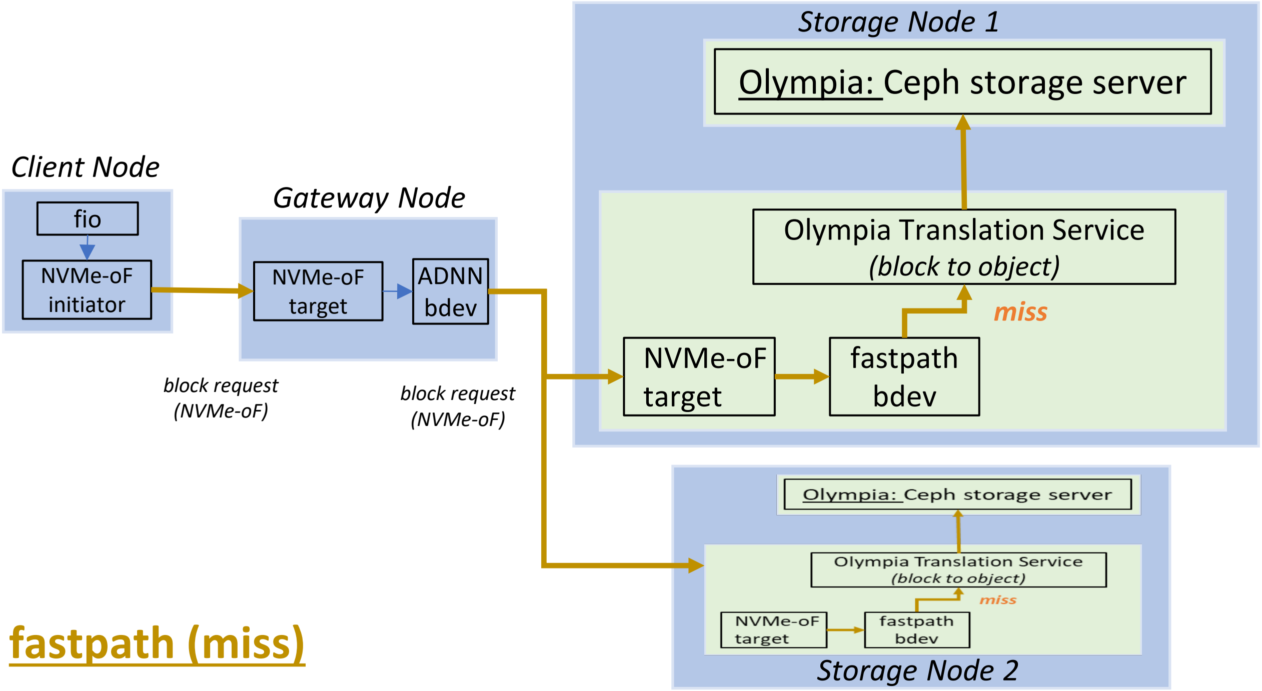
IO paths measured in experiments

Baseline (Ceph RBD)

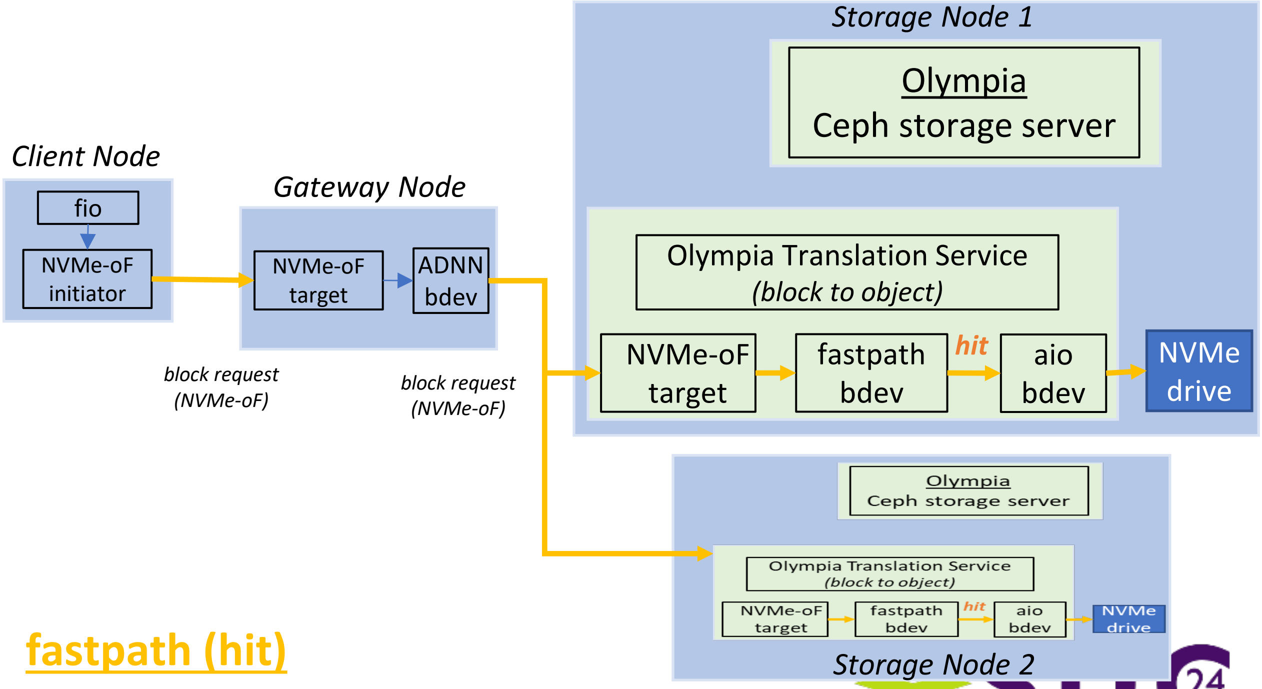


NVMf Gateway-RBD





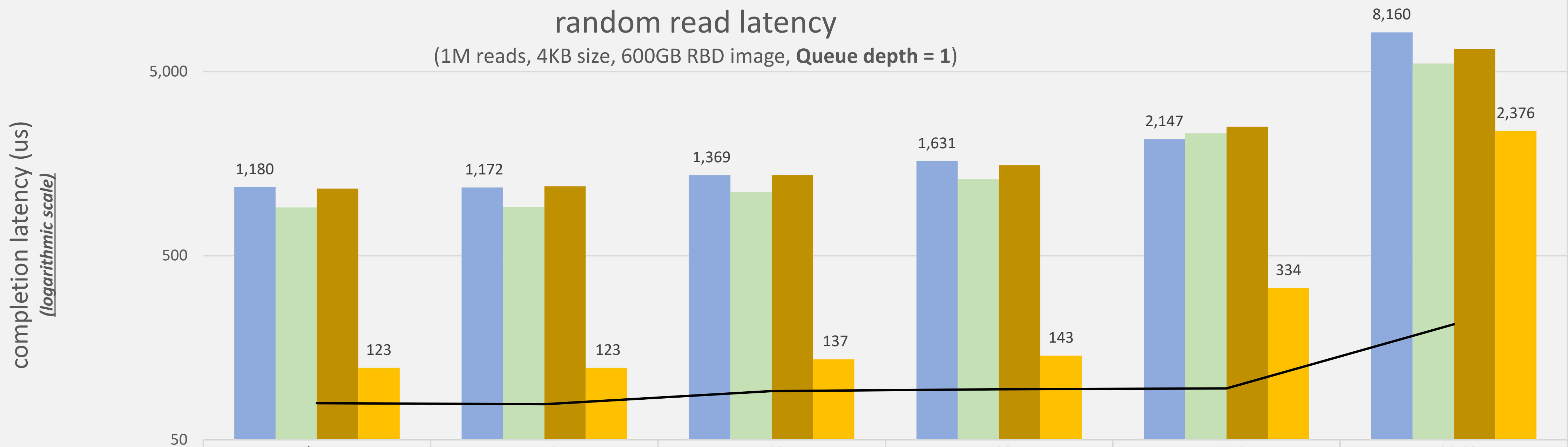
fastpath (miss)



fastpath (hit)

Performance Evaluation

Note: Lower is better



	Avg latency	P50	P90	P99	P99.9	P99.99
Baseline (Ceph RBD)	1,180	1,172	1,369	1,631	2,147	8,160
NVMf Gateway-RBD	913	922	1,106	1,303	2,311	5,538
fastpath (miss)	1,155	1,188	1,369	1,549	2,507	6,652
fastpath (hit)	123	123	137	143	334	2,376
local drive (s/w direct)	79	78	92	94	95	212

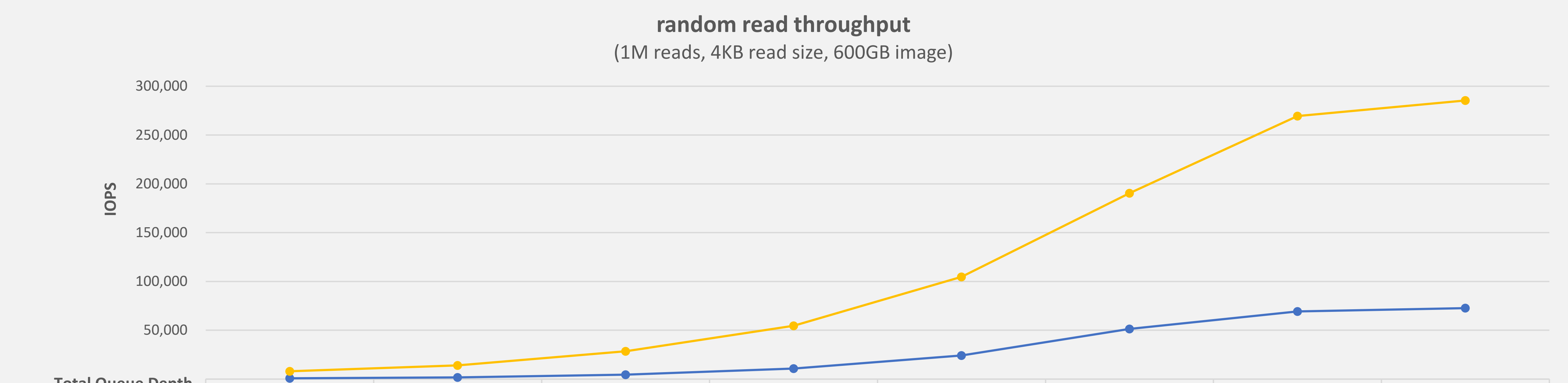
Test Setup

- Two node Ceph deployment with each node having one 730GB NVMe drive
- Baseline: client reads via Ceph native protocol (RADOS)
- Performance bar: Local drive software read latency
- fastpath hit columns → All the reads find cached mapping *(note: data is still read from drive)*

Hardware specifications
- P3700 NVMe drive latency = 20us

fastpath (hit) provides order-of-magnitude reduction in average latency

Performance Evaluation



Total Queue Depth	1	2	4	8	16	32	64	128
Baseline (Ceph RBD)	819	1,698	4,596	10,734	24,145	51,421	69,273	72,690
fastpath (hit)	8,022	14,015	28,481	54,590	104,800	190,500	269,400	285,400

*Total queue depth = number of jobs * I/O depth per job*

Test Setup

- Two node Ceph deployment with each node having one 730GB NVMe drive
- Baseline: client reads via Ceph native protocol (RADOS)
- Queue depth varied from 1 to 128

4-10X increase in throughput across varying queue depths

Summary and Future Work

Summary

- ✓ Decouple SDS architecture for fast data access
  - SDS continues to manage cluster-level functions to provide durability, availability, reliability guarantees
  - Companion module enables direct data path to drives
  - SDS controls mappings shared with data path module
- ✓ Software proof-of-concept implemented in Ceph demonstrates 10X speedups for random reads

Future work

- ✓ Test PoC at scale (increase number of OSD/drives/logical volumes/clients)
- ✓ Expand scope beyond block storage, to object and file interface
- ✓ Run fastpath module on accelerator for end-to-end hardware path for cloud storage!

Please take a moment to rate this session.

Your feedback is important to us.

