SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

SAMSUNG

Rethinking Distributed Storage System Architecture for Fast Storage Devices

Myoungwon Oh (myoungwon.oh@samsung.com)
Samsung Electronics

AGENDA

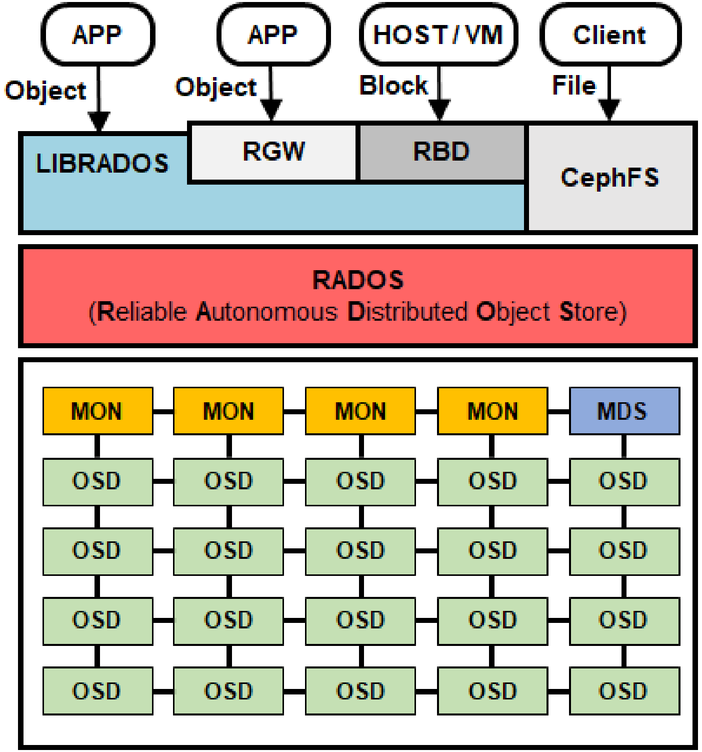
- Background and Motivation
- Proposed Design for Fast Storage Devices
  - Lightweight Data Store
  - Thread Control
  - Replication Offloading using NVMe-oF
- Summary

Background and Motivation

Ceph Architecture

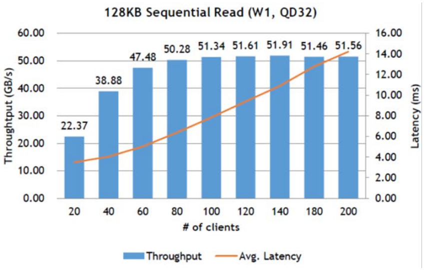
- 3 in 1 Interfaces
  - Object (RGW): Amazon S3 & OpenStack Swift
  - Block (RBD): Amazon EBS
  - File (CephFS): Lustre & GlusterFS
- RADOS
  - Heart of Ceph
  - Favor consistency and correctness over perfor mance
  - Serve I/O request, Protect data, and Check the consistency and integrity of data

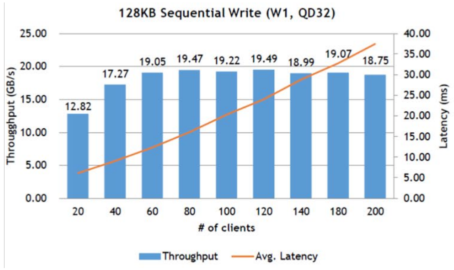
① **OSD**: Serve I/O, Replication/EC, Rebalance, Cohering Data
② **MON**: Maintain a master copy of the cluster map and state
③ **MGR**: Collect the statistics within the cluster
④ **MDS**: Manage the metadata (only for CephFS)



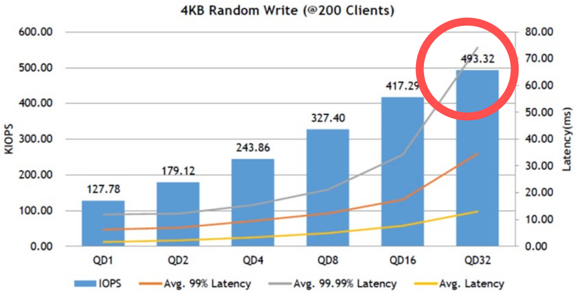
Challenging Issue: Performance

- Sequential I/O is good
  - Can fully saturate the network bandwidth



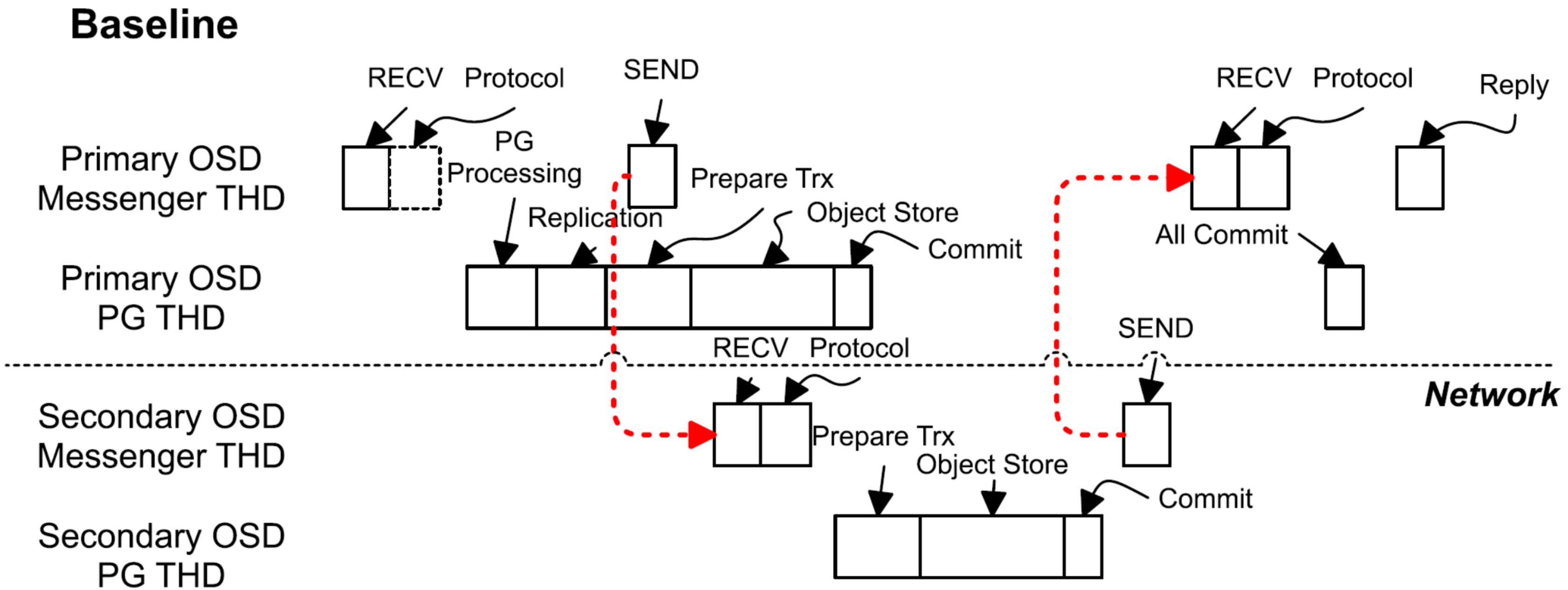


- Random small write is bad
  - CPU can become a bottleneck



Reference: https://www.samsung.com/semiconductor/global.semi/file/resource/2020/05/redhat-ceph-whitepaper-0521.pdf

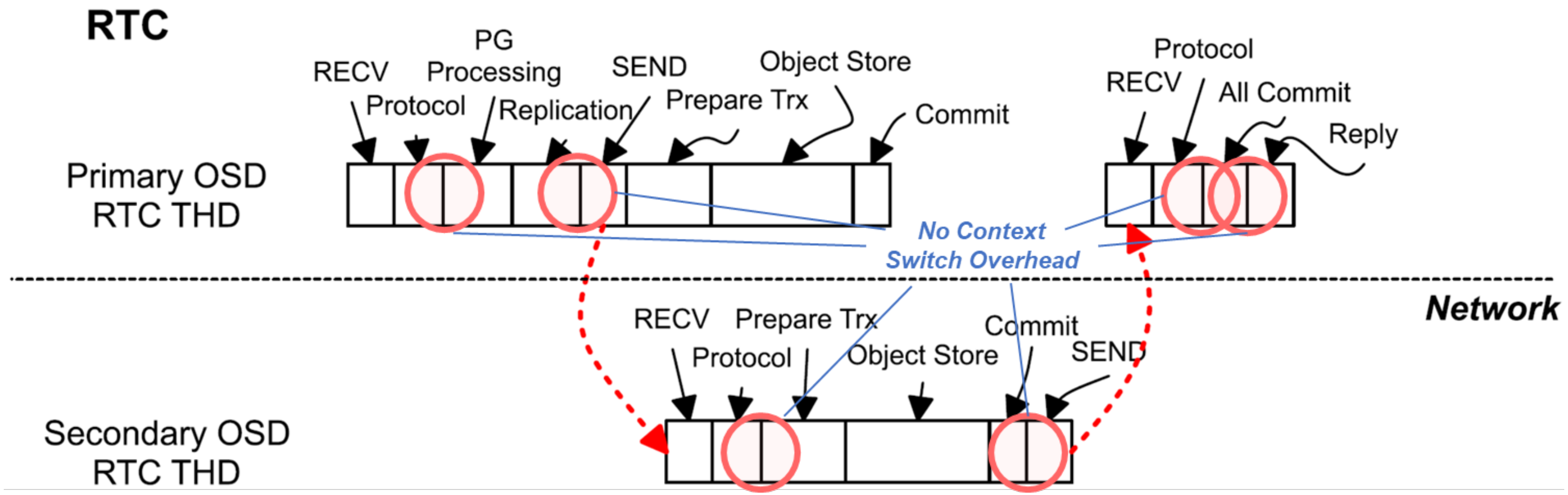
The I/O path



Baseline
RECV
Protocol
SEND
RECV
Protocol
Reply
Primary OSD Messenger THD
PG Processing
Prepare Trx
Replication
Object Store
Commit
All Commit
Primary OSD PG THD
SEND
RECV
Protocol
Network
Secondary OSD Messenger THD
Prepare Trx
Object Store
Commit
Secondary OSD PG THD

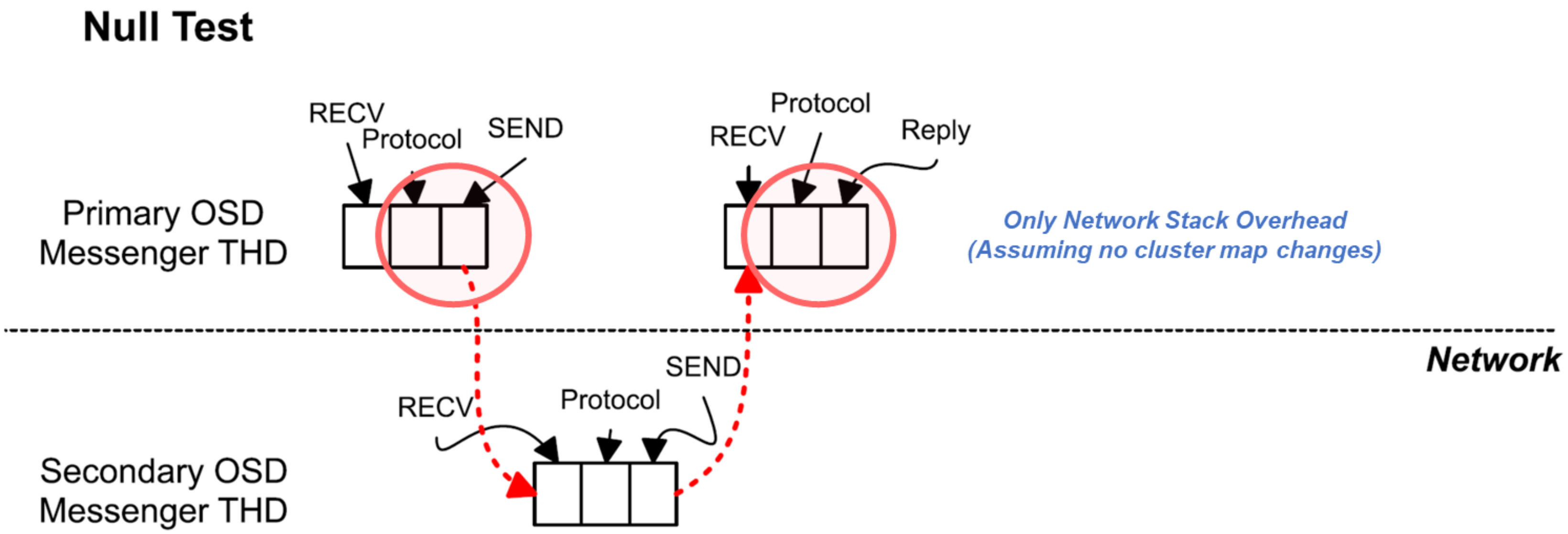
Experiment - RTC

- Write I/O Path in "*RTC*"
  - Run-to-completion model
  - Can eliminate the context switching overhead



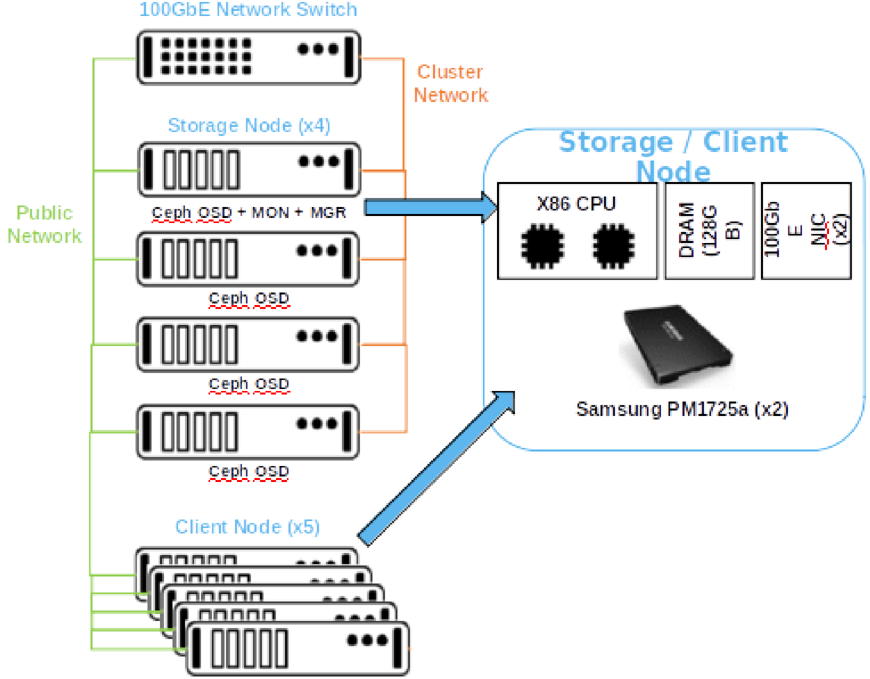
Experiment - Null Test

- Write I/O Path in "*Null Test*"
  - Hypothetical maximum performance when only network stack overhead exists



Experiment - Setup

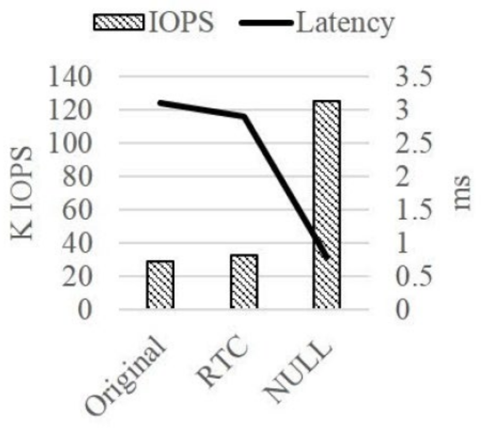
Use limited resource to eliminate other interference factors
- Each OSD has two messenger threads
- Each OSD has two storage threads



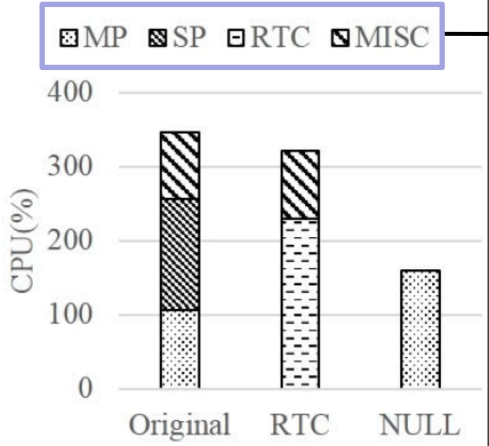
Ceph Configuration	
Ceph Version	15.02 (Octopus)
Ceph Storage Type	RBD
OSD Backend Storage	Bluestore
# of OSD daemon	32
OSDs per NVMe SSD	4
Replication Factor	x2
# of PG (Placement Group)	1024
Storage Nodes (x4)	
Processor (x2)	Intel® Xeon® Gold 6152 CPU @ 2.10GHz (22-core 44-thread)
DRAM (x8)	Samsung 16GB DDR4-2400 MT/s (128GB per node)
NVMe SSD (x2)	Samsung PM1725a NVMe SSD, 1.6TB, 2.5 inch form factor
NIC	Mellanox Connect X®-5 MCX516A-CCAT Dual-Port adapter (100GbE)
Network devices	
Network Switch (x1)	Mellanox MSN2700-CS2F 1U Ethernet switch (100GbE)
NIC	Mellanox Connect X®-5 MCX516A-CCAT Dual-Port adapter (100GbE)

Experiment - Results

- FIO 4KB Random Write – RBD
  - Use limited resource to eliminate other interference factors



IOPS / Latency



CPU usage

MP: Message Processing
SP: Storage Processing
RTC: Run-to-completion
MISC: Compaction, sync, etc.

	User	Data	Misc	Total
Original (GB)	21	42	78	120

Write Amplification

Improvement Points

- High CPU Consumption of Object Store
  - Write Amplification / Data Partition / Log-structured Data Layout
- Inefficient Threading Architecture
  - Conventional thread pool design (network and storage) caused performance deg radation
  - Strong consistency service needs an ordering and persistency
- Lots of Works in a Server Node
  - Network I/O Dispatching, Replication I/O Handling, Fault and Meta Managing, …

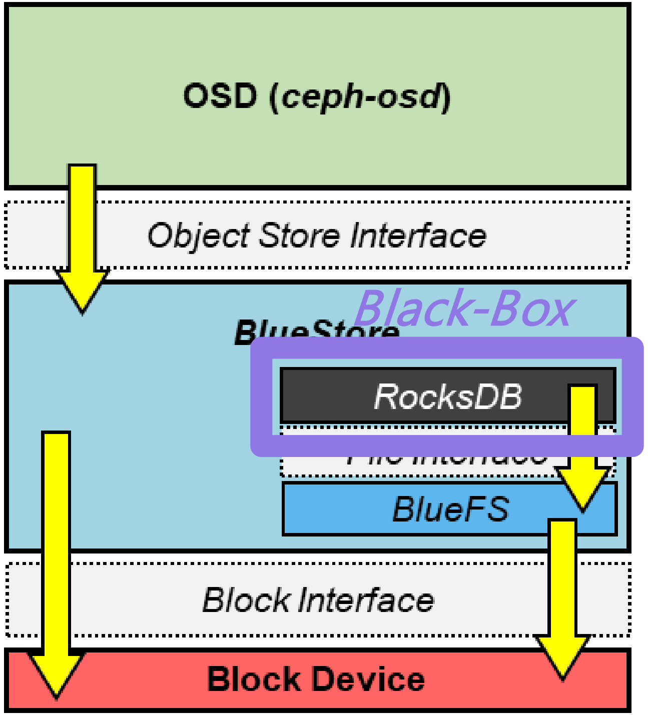
Proposed Design

Key Ideas

- Hybrid Update (out-of-place & in-place) based Application-managed Store
  - to minimize CPU consumption of data store
- Locality-aware Prioritized Thread Control
  - to minimize the context switching overhead while prioritizing network processing
- Offload Replication Processing using NVMe-oF Techniques
  - to minimize CPU consumption

① Lightweight Data Store

- Drawbacks of *BlueStore*
  - LSM-tree based store incurs **high write amplification** and **CPU overhead** due to the compaction process
  - Unnecessary **data copy** and **serializatio n/deserialization** overhead
  - **Parallelism** (single partition)

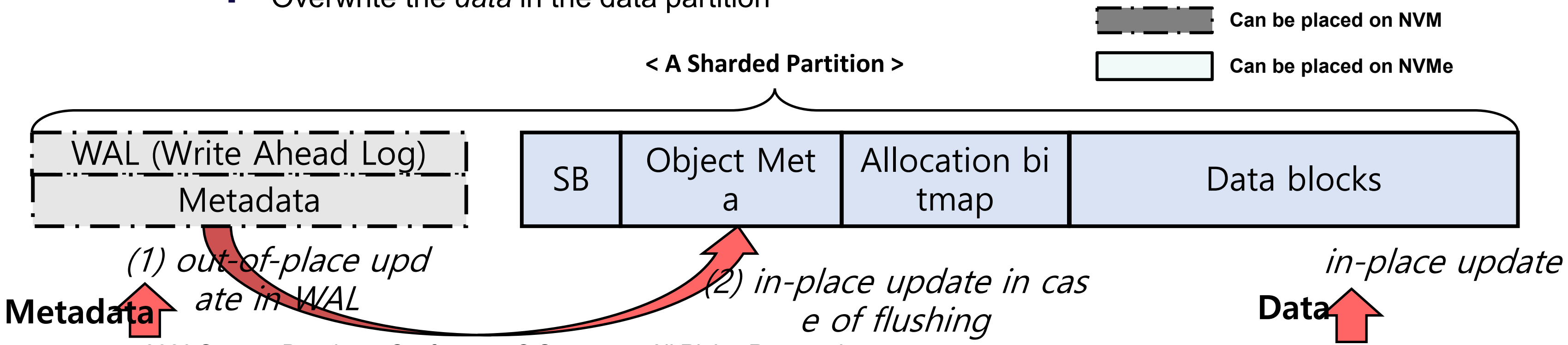


① Lightweight Data Store

- Overview
  - Target **only high-end NVMe SSDs** and **NVM**
  - **Hybrid update strategies** for different data types (in-place, out-of-place)
    - to minimize CPU consumption by reducing host-side GC
  - Utilize **NVMe feature** (atomic large write command)
  - **Sharded** data/processing model
  - Support **transactions**

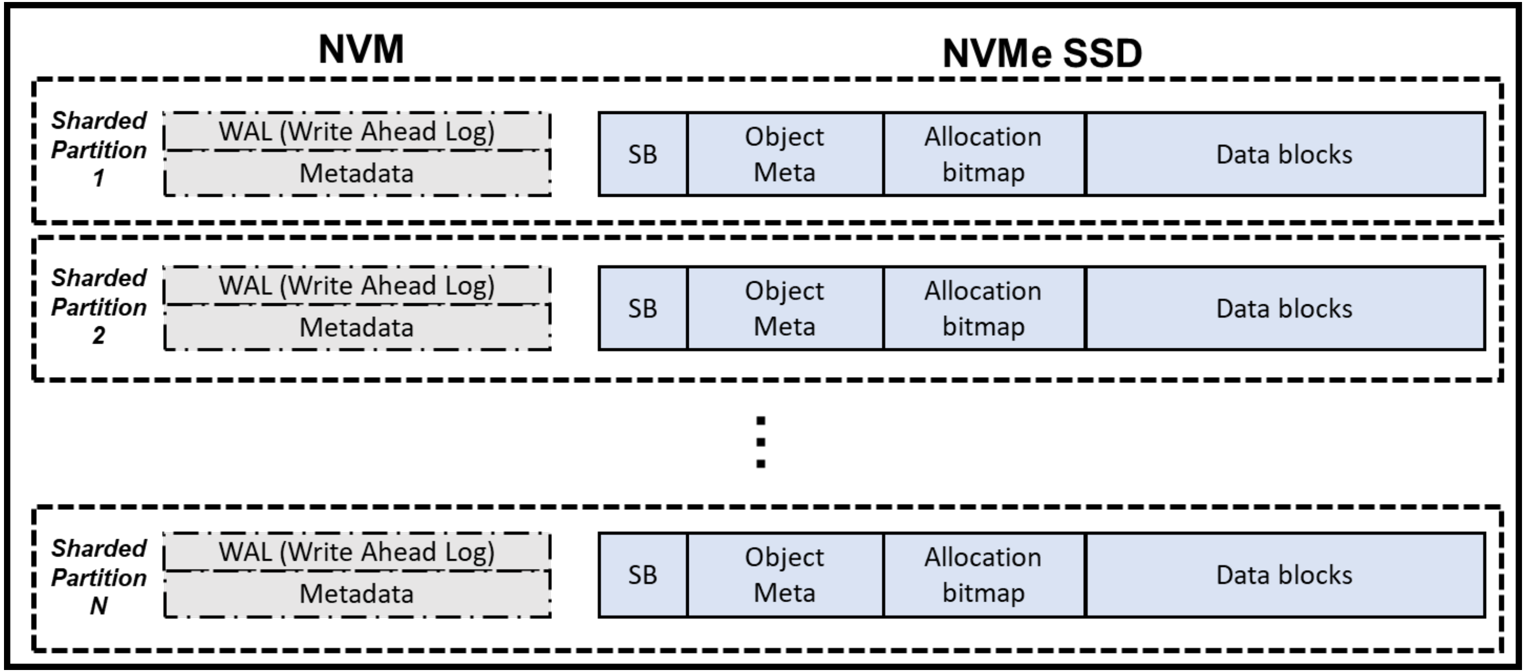
① Lightweight Data Store

- Overview
  - WAL Partition
    - *Log* and frequently updated *metadata* are stored as a WAL entry in the WAL partition
    - Space within the WAL partition is continually reused in a circular manner
    - Flush the *metadata* if necessary
  - Write procedure for *Metadata*
    - Appended at the WAL first -> Overwrite the *metadata* in the data partition in case of flushing
  - Write procedure for *Data*
    - Overwrite the *data* in the data partition



① Lightweight Data Store

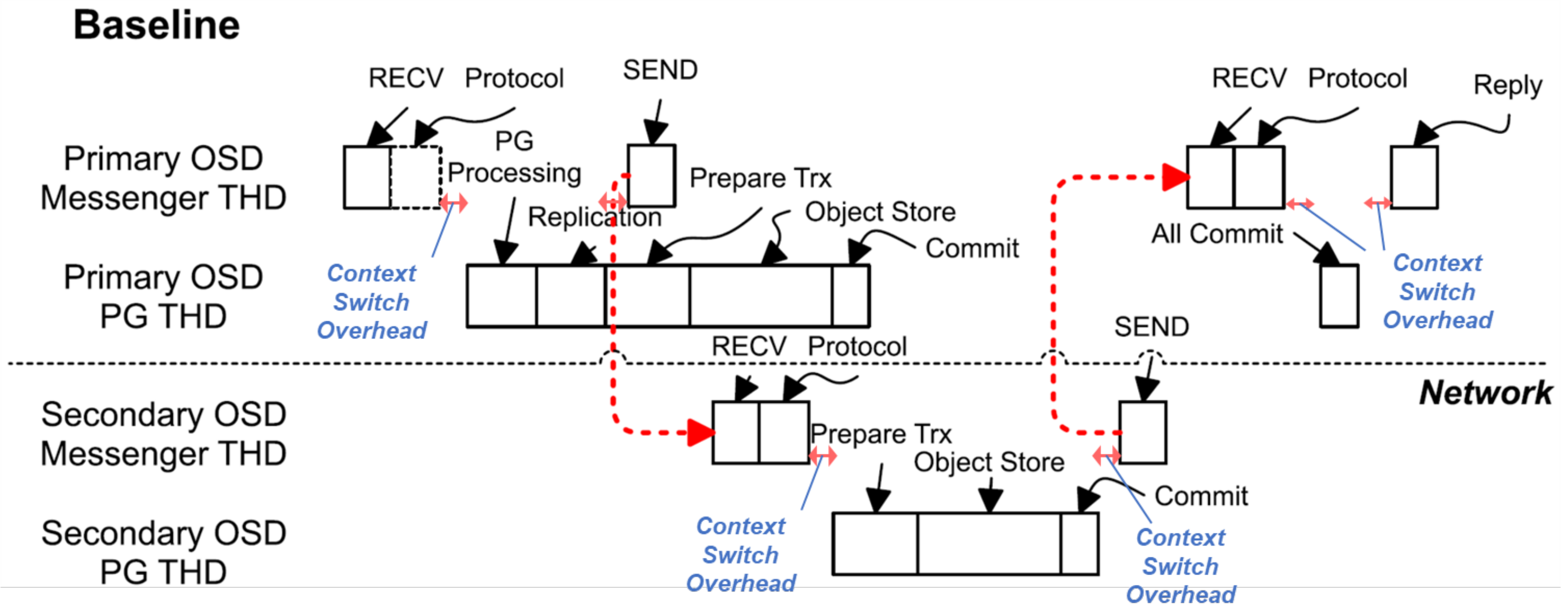
- Overview



① Lightweight Data Store

- Best case vs. Worst case
  - Best case (pre-allocation):
    - **One** write for **WAL** + **One** write for **data**
    - With pre-allocation, object meta and data bitmap aren't required to be updated
  - Worst case (flush happens):
    - **One** write for **WAL** + **One** write for **object meta** + **One** write for **allocation bitmap** + **One** writes for **data**
  - In either cases, it **doesn't** produce **unpredictable severe write amp lification** (constant rate) unlike LSM-tree DB
  - Frequent updated data can be placed in the NVM

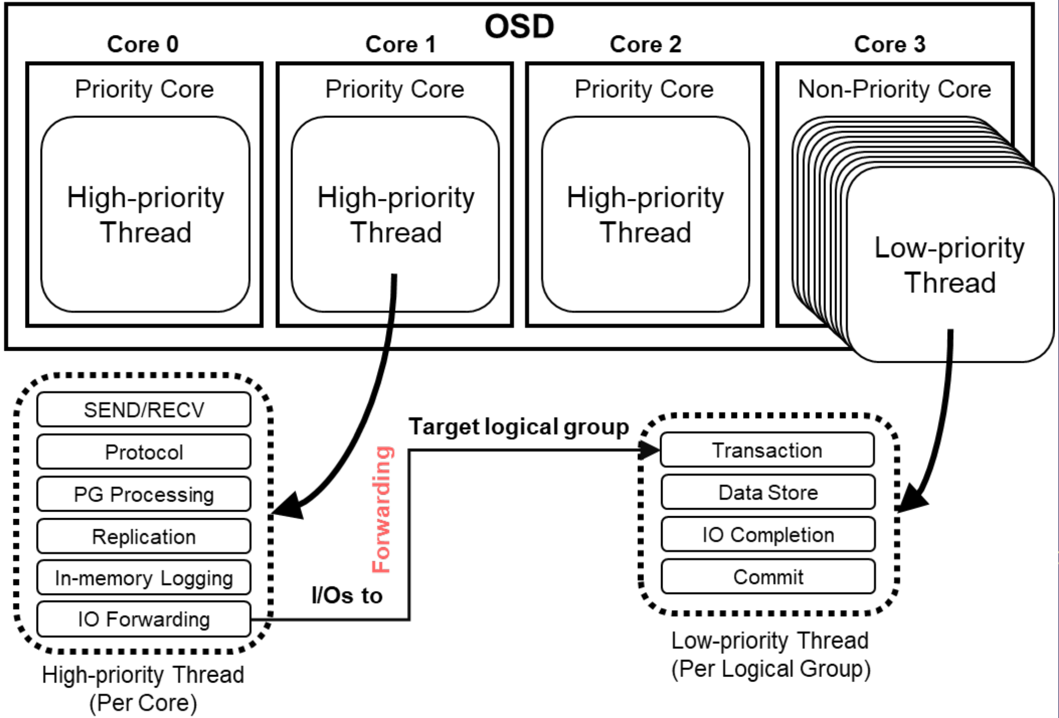
② Locality-aware Prioritized Thread Control



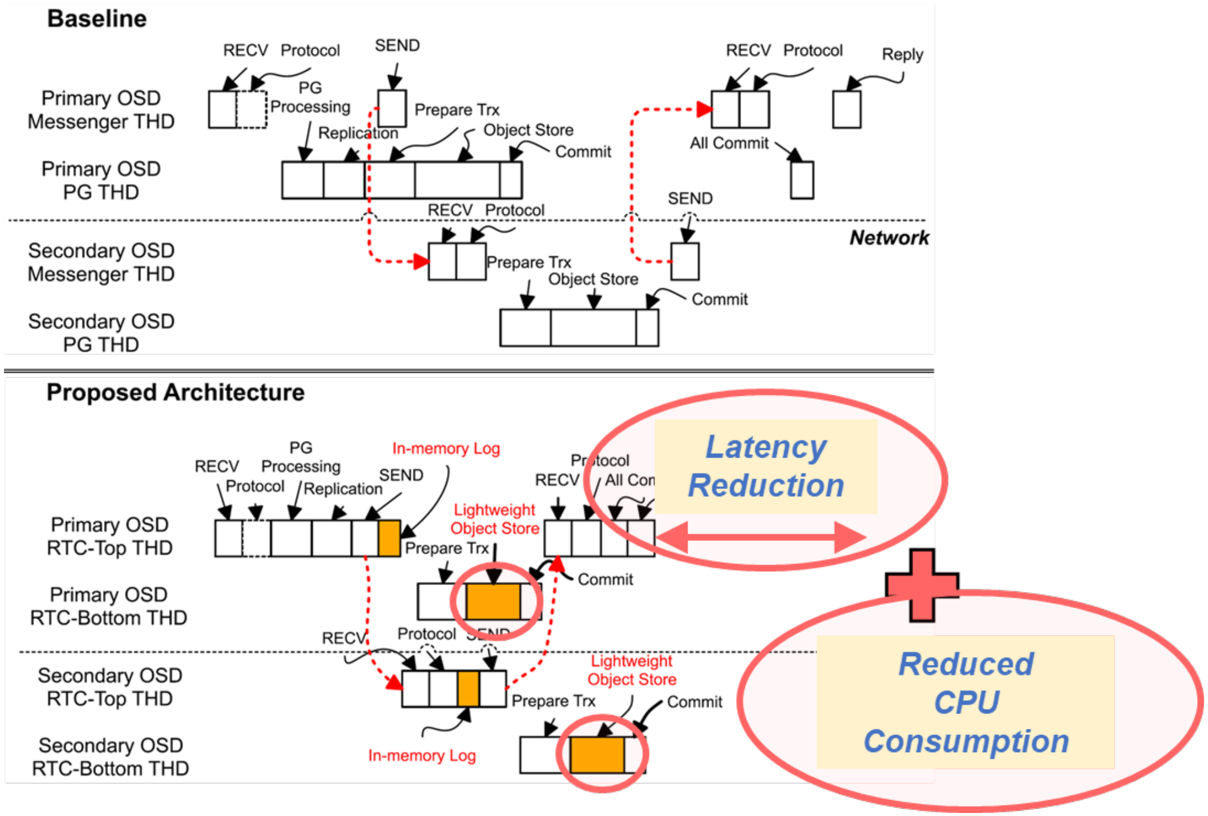
Need an order !!!

② Locality-aware Prioritized Thread Control

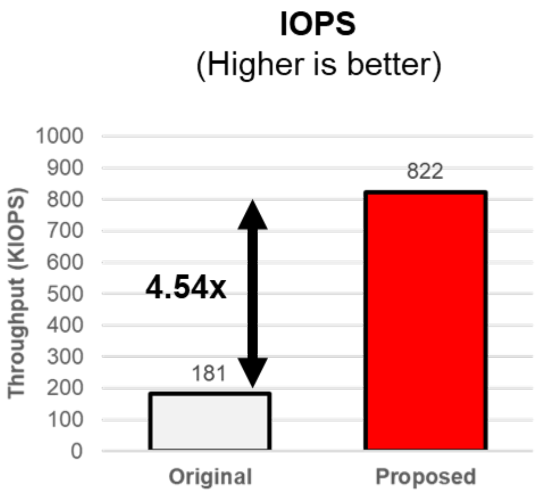
- ***High-priority*** *thread*
  - Network processing and I/O forwarding (**Latency critical job**)
- ***Low-priority*** *thread*
  - Storage processing (**Best-effort batch job**)

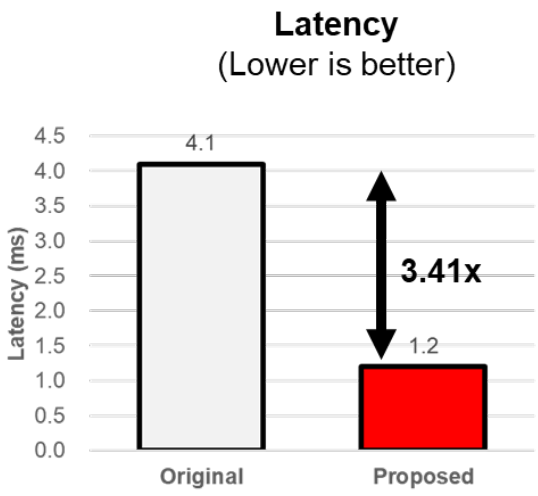


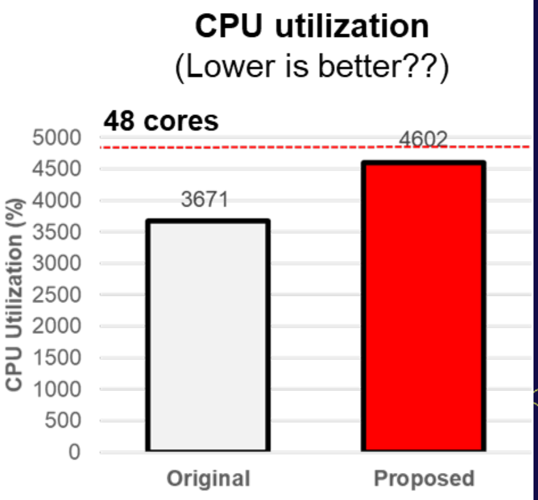
② Locality-aware Prioritized Thread Control



② Locality-aware Prioritized Thread Control

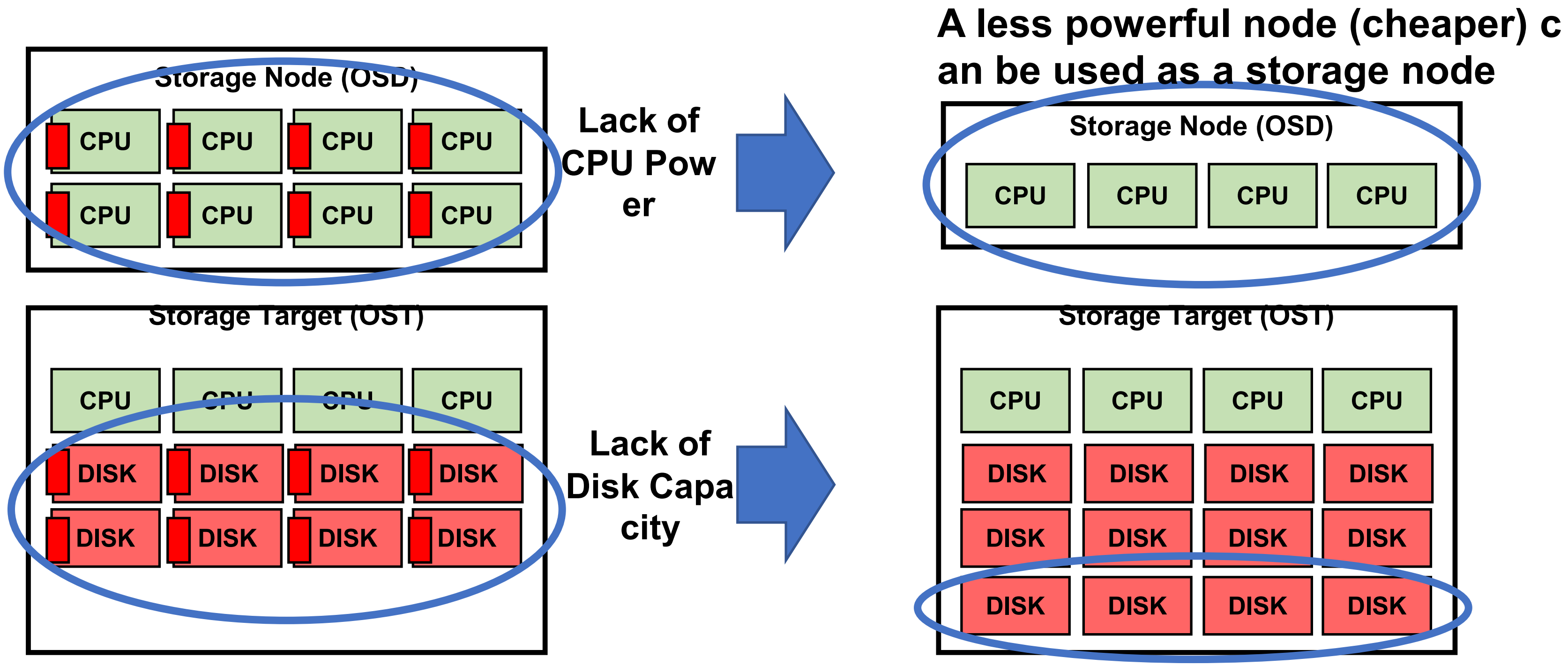






③ Replication Offloading using NVMe-oF

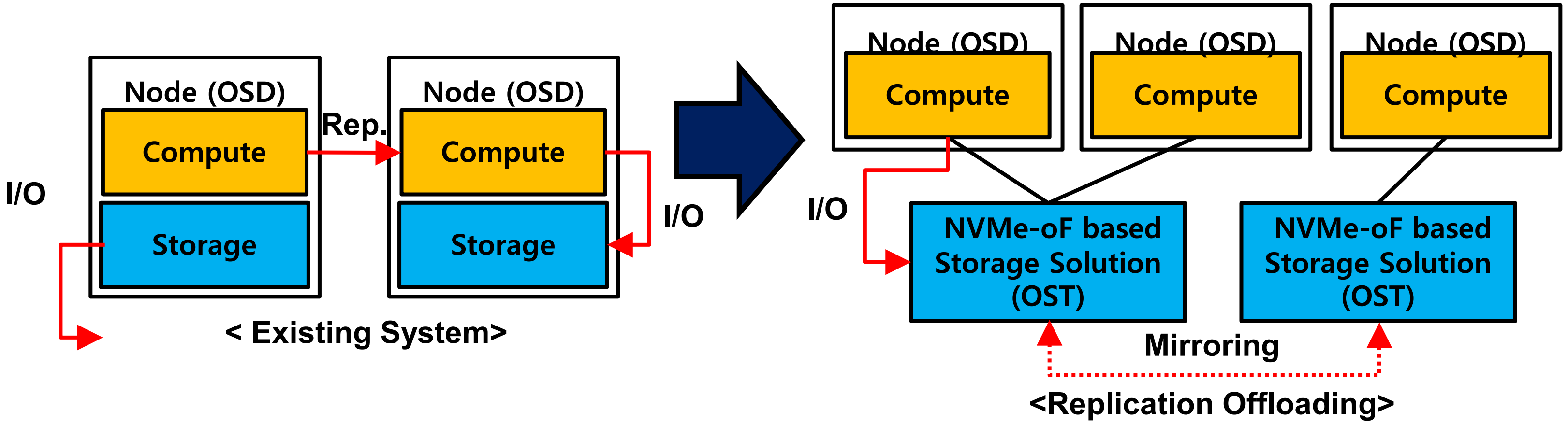
- Storage Disaggregation with NVMe-oF



③ Replication Offloading using NVMe-oF

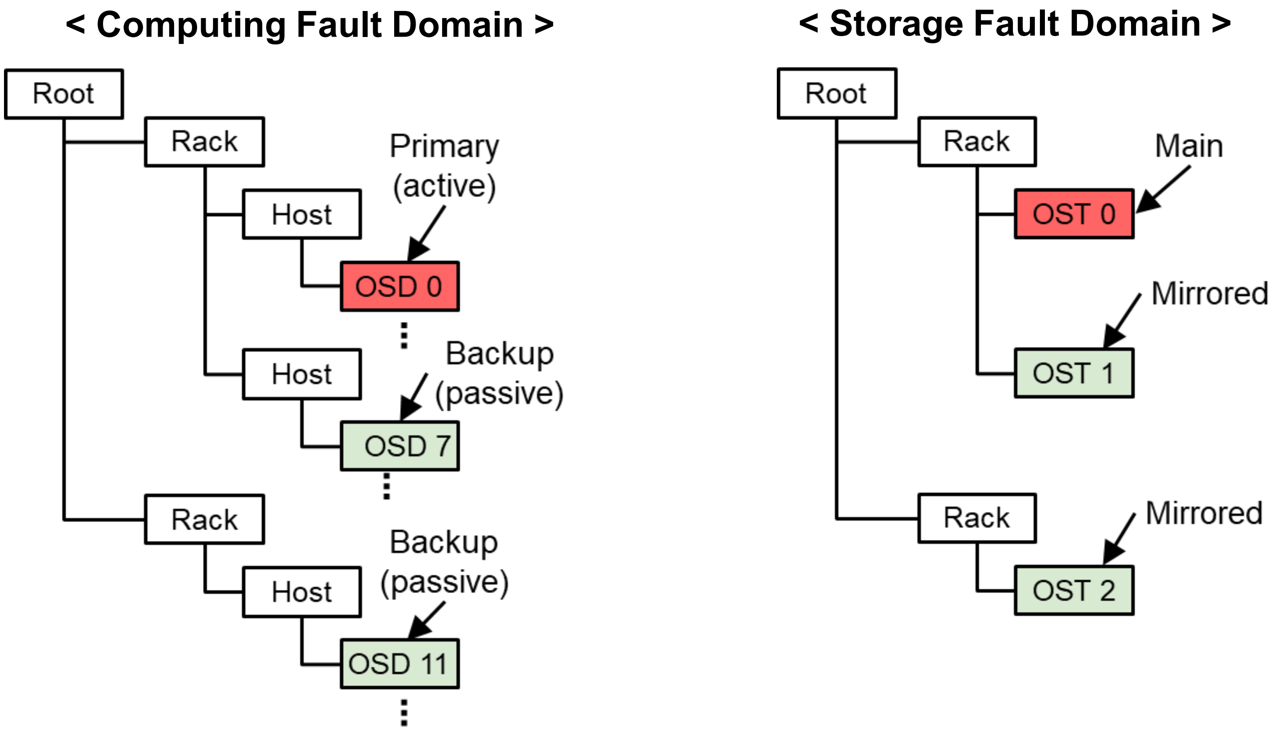
- Idea
  - Replication is **offloaded** to **NVMe-oF based storage solution** that support s mirroring
  - OSD does **not** process **replication** but is **aware** of **where** data is mirrored

Storage node-side CPU usage is reduced without losing data redundancy



③ Replication Offloading using NVMe-oF

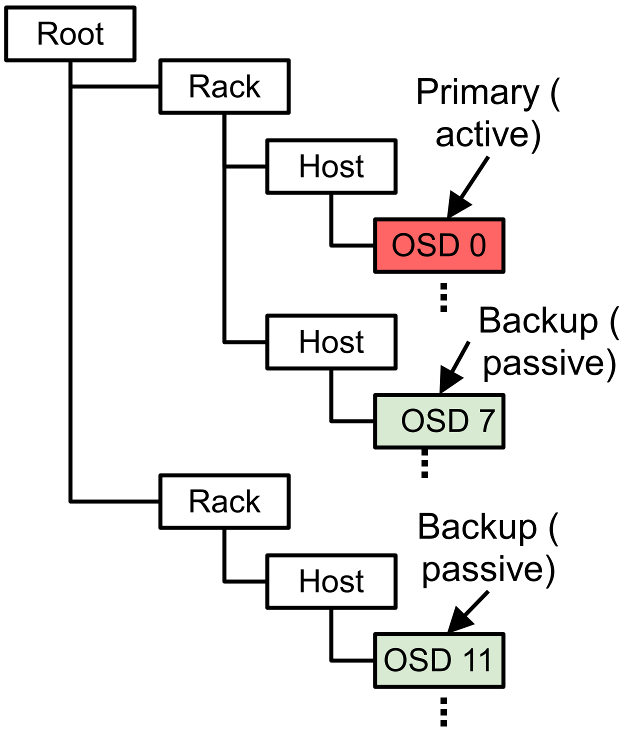
- Separate Fault Domain



③ Replication Offloading using NVMe-oF

- Computing Fault Domain

< CRUSH >
Input : Pool ID, Placement Group #
Output : A set of OSDs



{**<u>OSD 0</u>**, OSD 7, OSD 11}

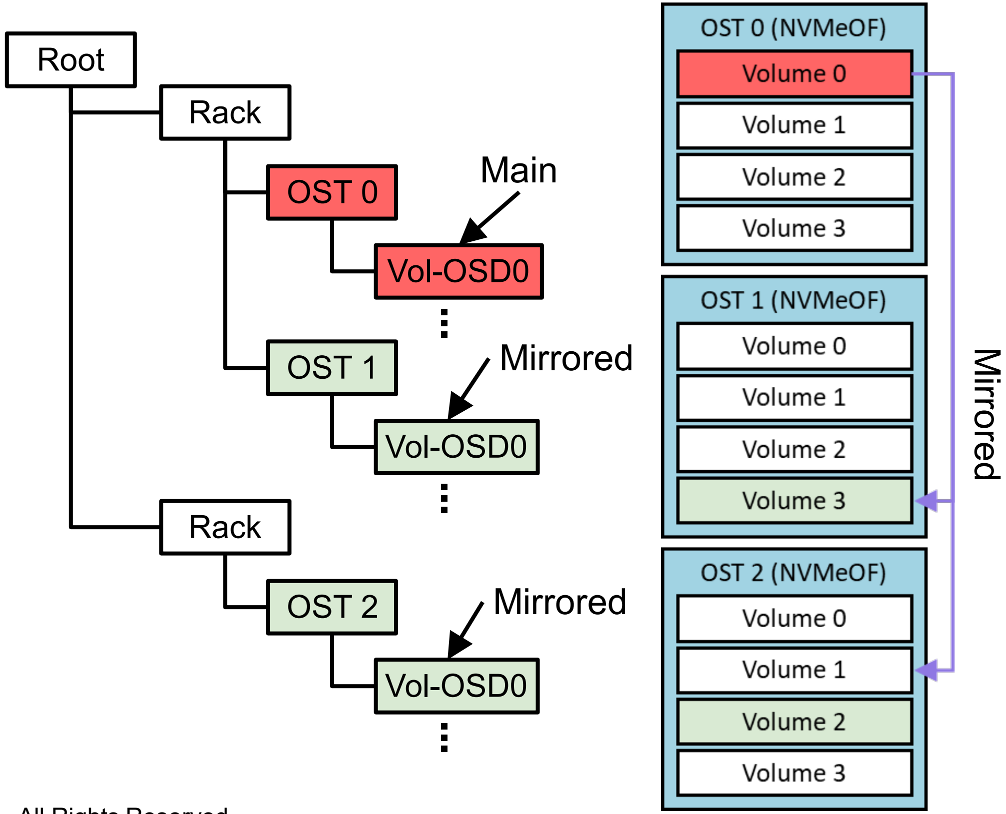
Primary (active)

③ Replication Offloading using NVMe-oF

- Storage Fault Domain

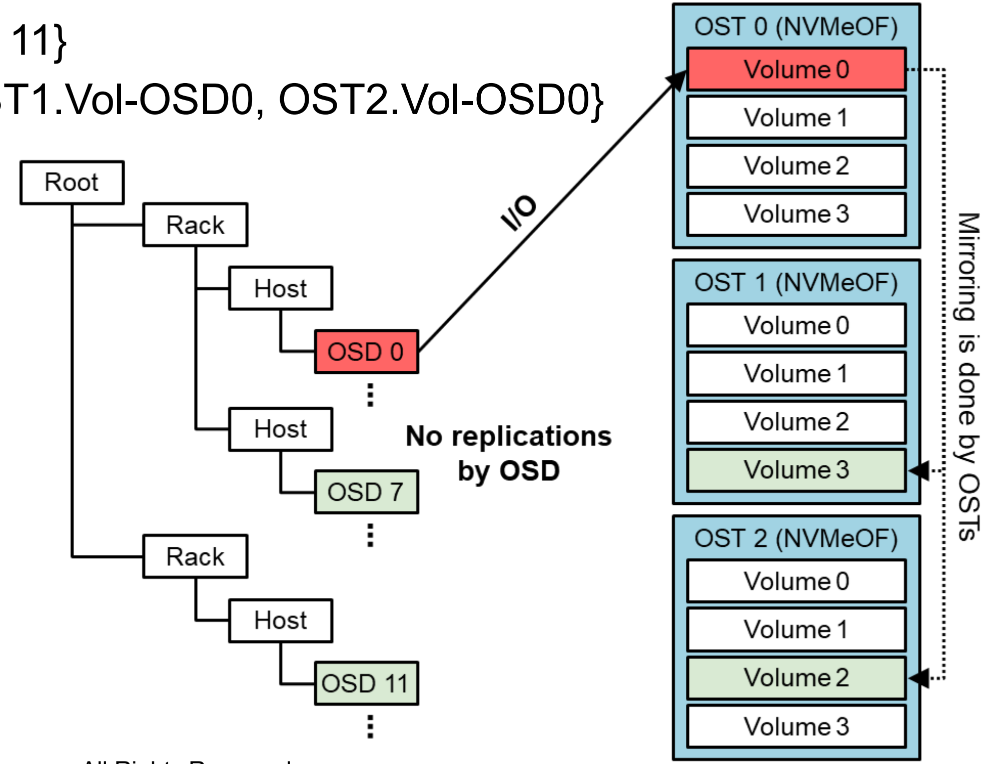
< CRUSH >
Input : Primary OSD ID
Output : A set of OSTs

Main
{<u>**OST0.Vol-OSD0**</u>,
OST1.Vol-OSD0,
OST2.Vol-OSD0}



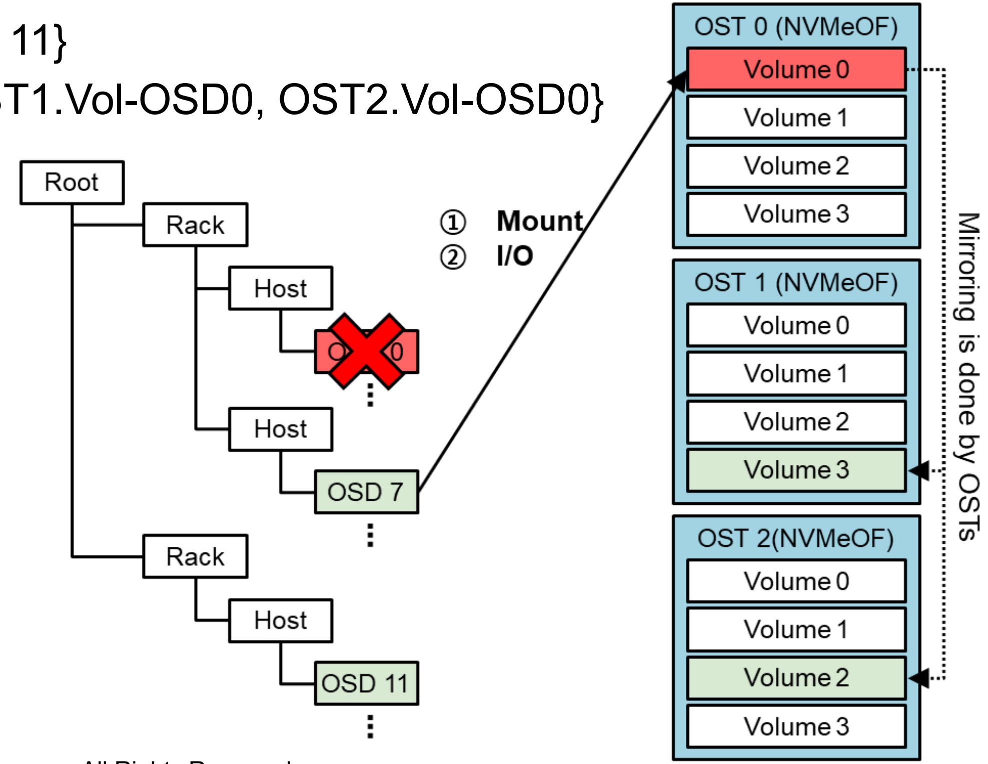
③ Replication Offloading using NVMe-oF

- I/O Path
  - {**OSD 0**, OSD 7, OSD 11}
  - {**OST0.Vol-OSD0**, OST1.Vol-OSD0, OST2.Vol-OSD0}



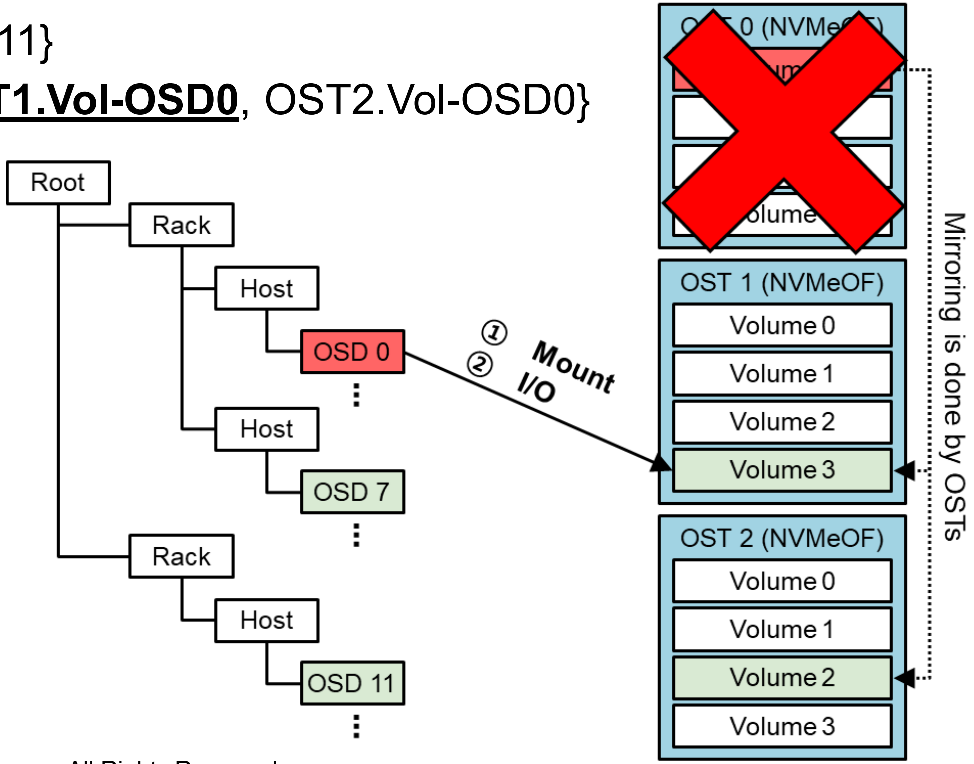
③ Replication Offloading using NVMe-oF

- OSD (Computing) Failure
  - {~~OSD 0~~, **<u>OSD 7</u>**, OSD 11}
  - {**<u>OST0.Vol-OSD0</u>**, OST1.Vol-OSD0, OST2.Vol-OSD0}



③ Replication Offloading using NVMe-oF

- OST (Storage) Failure
  - {**<u>OSD 0</u>**, OSD 7, OSD 11}
  - {~~OST0.Vol-OSD0~~, **<u>OST1.Vol-OSD0</u>**, OST2.Vol-OSD0}



Summary

Conclusion

- Reliable distributed storage systems face the challenge to fully exploit NVMe performance
  - Excessive CPU usage is the main problem
- It's time to rethink the conventional I/O SW stack
- We propose a lightweight I/O architecture for distributed storage systems

Upstream Status

- Efforts to make flash devices more attractive to the world
  - Tiering / Global deduplication Improvement
    - https://github.com/ceph/ceph/pull/29283 (Merged)
    - https://github.com/ceph/ceph/pull/35899
    - https://github.com/ceph/ceph/pull/34684
    - https://github.com/ceph/ceph/pull/35112
  - New Store Design
    - https://github.com/ceph/ceph/pull/36343

Question