2020 OFA Virtual Workshop

DISTRIBUTED ASYNCHRONOUS OBJECT STORAGE (DAOS)

Kenneth Cain, Johann Lombardi, Alexander Oganezov

Intel

NOTICES AND DISCLAIMERS

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.

No product or component can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

Intel Advanced Vector Extensions (Intel AVX) provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

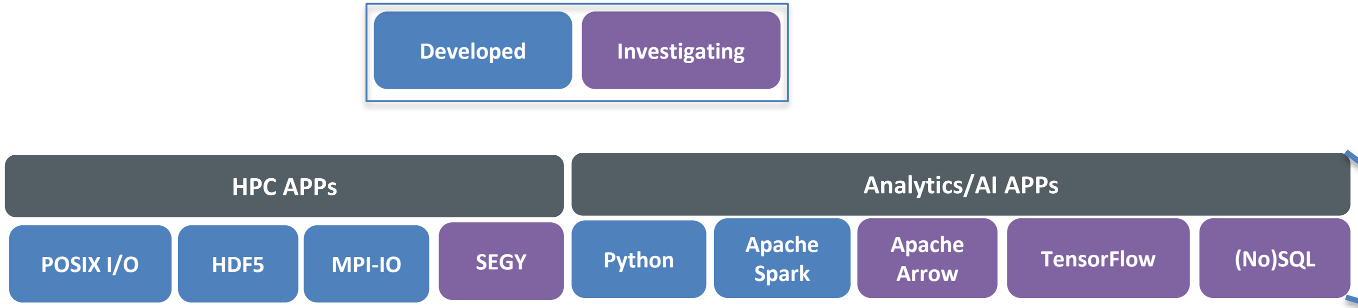
Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

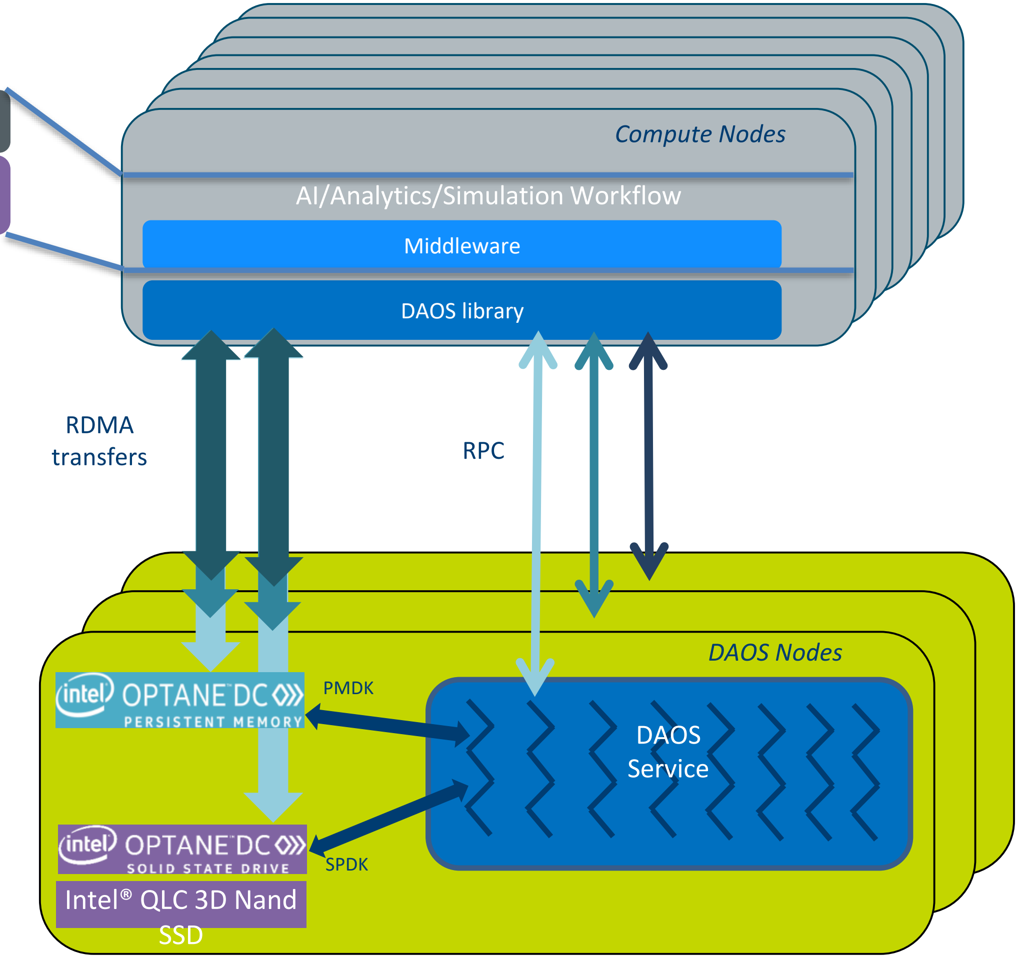
OUTLINE

- **DAOS overview**
- **Lessons learned building DAOS using OFI / libfabric**
- **Brainstorm – opportunities to further leverage networks/fabrics**

DAOS ARCHITECTURE OVERVIEW

DAOS ARCHITECTURE:
CLIENT LIBRARY AND INTERFACES

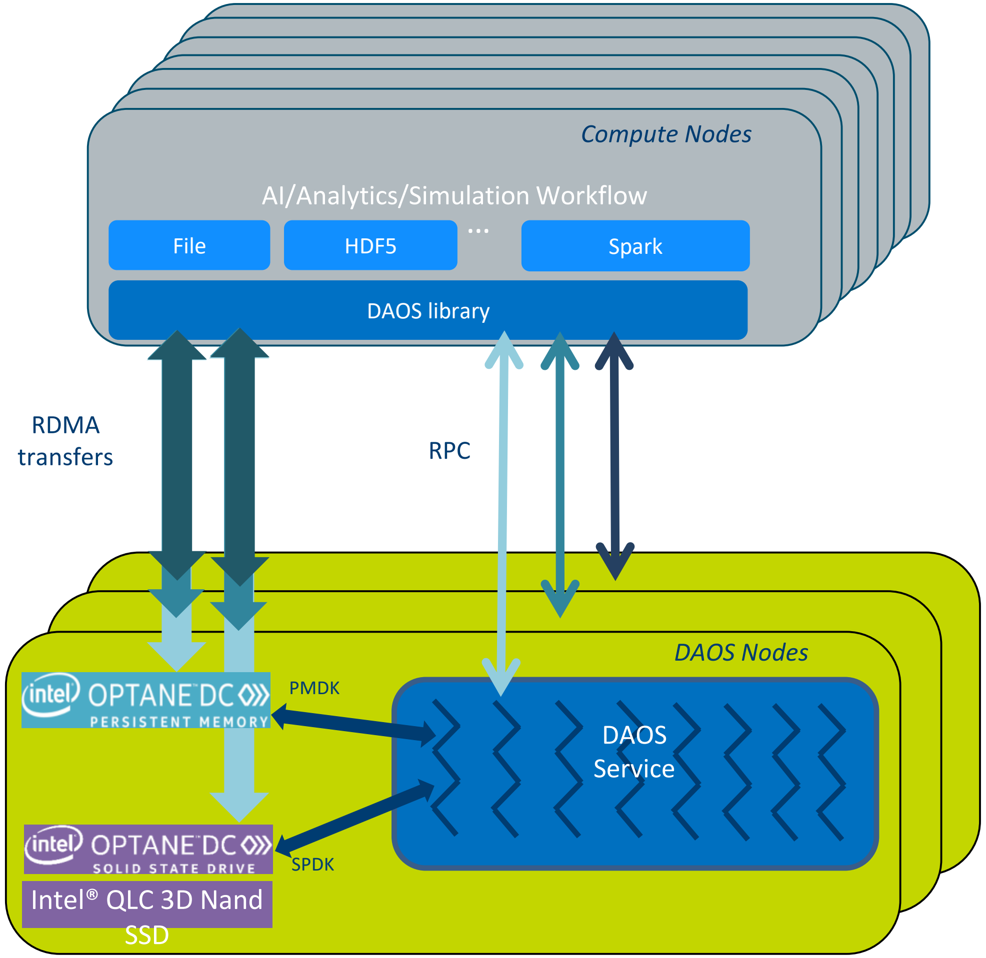




- **POSIX I/O – namespace distributed over servers**
  - DAOS Filesystem (libdfs) – apps / frameworks may link directly
  - FUSE Daemon – transparent access to DAOS, involves syscalls
  - I/O Interception Library – OS bypass for read/write operations
- **MPI-IO Support**
  - MPI-IO Driver uses DAOS array API (+ libdfs for collective open)
- **Python Bindings**
  - Export key-value store objects
  - Integrate with dictionaries: iterator, direct assignment, etc.

DAOS ARCHITECTURE:
HIGH PERFORMANCE COMMUNICATIONS

- **RDMA (iWARP, RoCE, IB, OPA) + scalable collectives**
- **User-space networking, libfabric (via CART / Mercury)**
  - RPC via tagged messages: **fi_tsend / fi_trecv**
  - Bulk transfer via RDMA: **fi_readmsg / fi_writemsg**
  - End-to-end OS bypass: low-latency, high-message-rate in I/O path
- **Clients / applications link with DAOS lib**
  - No: context switch, locking, caching, or data copy
  - No need for dedicated cores
- **Servers initiate RDMA**
  - PM access over fabric:
    - Zero-copy RDMA to PM ; mmap'ed via PMDK
    - Memory consistency / flush done in server *code* after RDMA
  - NVMe SSD access over fabric:
    - RDMA into DRAM ; then SPDK for I/O to device

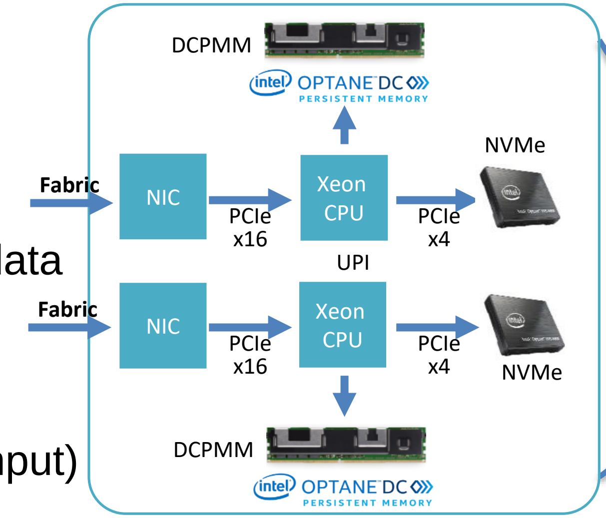


DAOS ARCHITECTURE:

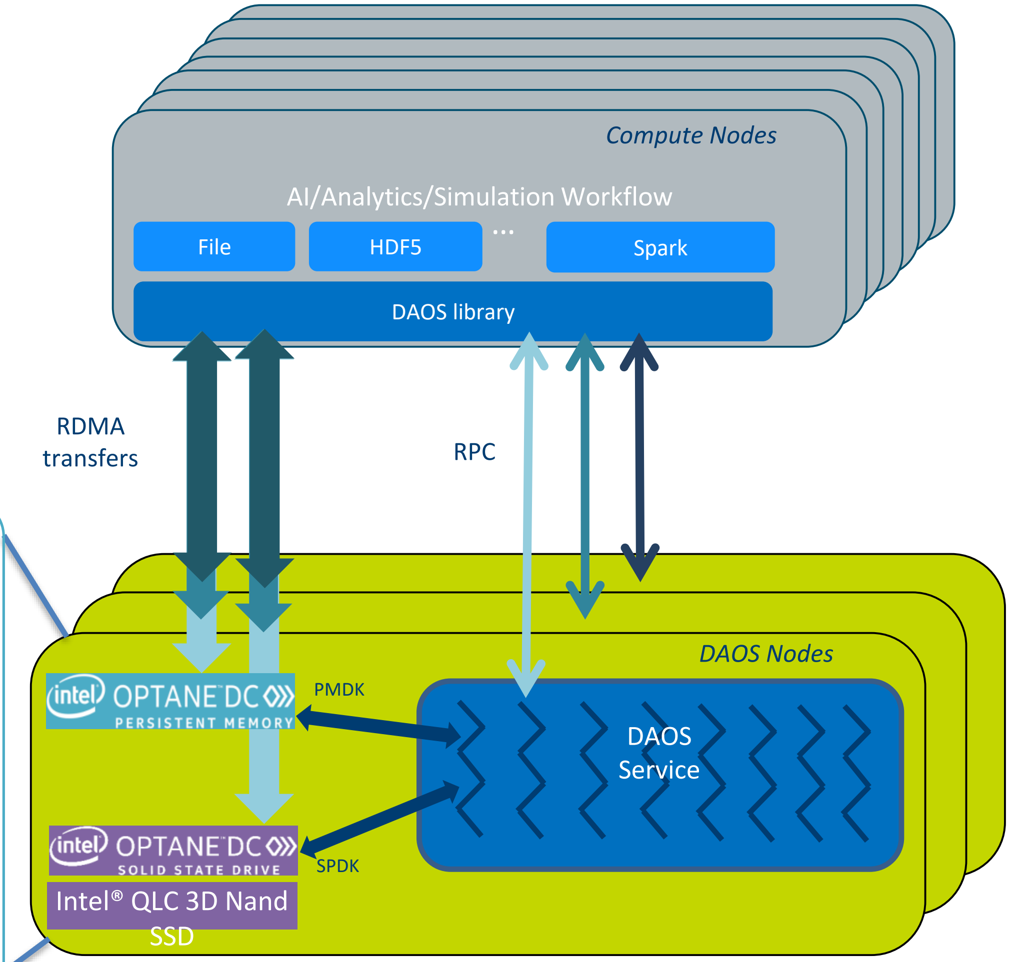
STORAGE SERVERS – TWO-LEVEL DATA PLACEMENT

- **Algorithmic placement**
  - Identify servers to store data replicas or shards
  - **Client-calculated** jump consistent hash based on key, object class (e.g., replicated, striped)
  - Fault domains taken into account – reduce impact of server loss (e.g., if a whole domain fails)

- **One tier, two media types:**
  - **Server-selected** media
  - Data Center Persistent Memory (DCPMM)
    - PM ← app small, byte-granular data, metadata
    - PM ← DAOS metadata
  - NVMe (*NAND, Intel® Optane™) SSD
    - SSD ← app-only bulk data (for high throughput)
    - SSD ← (aggregation of small data in PM)



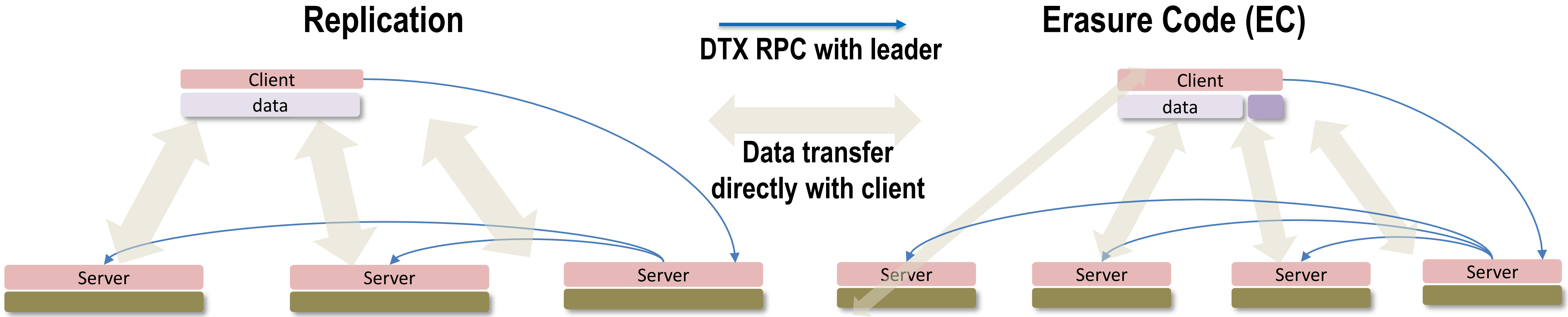
Dual-Socket DAOS Nodes (DNs)
Intel® Xeon servers with DCPMM & NVMe SSDs



DATA PROTECTION AND SELF-HEALING / REBUILD

COMMON PROPERTIES OF REPLICATION AND ERASURE CODE (EC)

- **Leader server chosen to manage distributed transaction protocol (DTX)**
  - Chosen algorithmically based on key – no single leader node bottleneck
- **Degraded mode – client I/O satisfied by surviving servers**
  - Non-blocking protocol for server fail-out
- **Self-healing / rebuild (online recovery)**
  - Declustered – per object, select alternate server storage to restore original degree of replication
    - Many alternate nodes in parallel – pull object data from surviving servers
  - Throttled – to control impact to serving ongoing client I/O requests

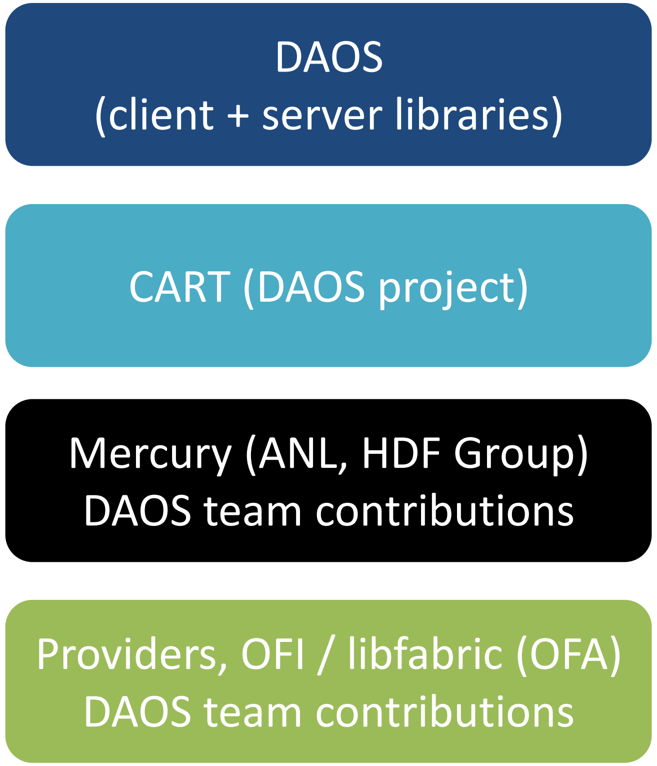


DAOS TO LIBFABRIC

VIA CART, MERCURY RPC MIDDLEWARES

- **CART: Collective Adaptive Reliable Transport**
  - P2P RPC reliability, built over Mercury RPC:
    - Timeout detection / retry
    - Flow control
    - SWIM protocol for fault detection / reaction
  - Collective RPC – broadcast, barrier, incast variable (IV)
    - Reliability: group versioning as membership expands or contracts

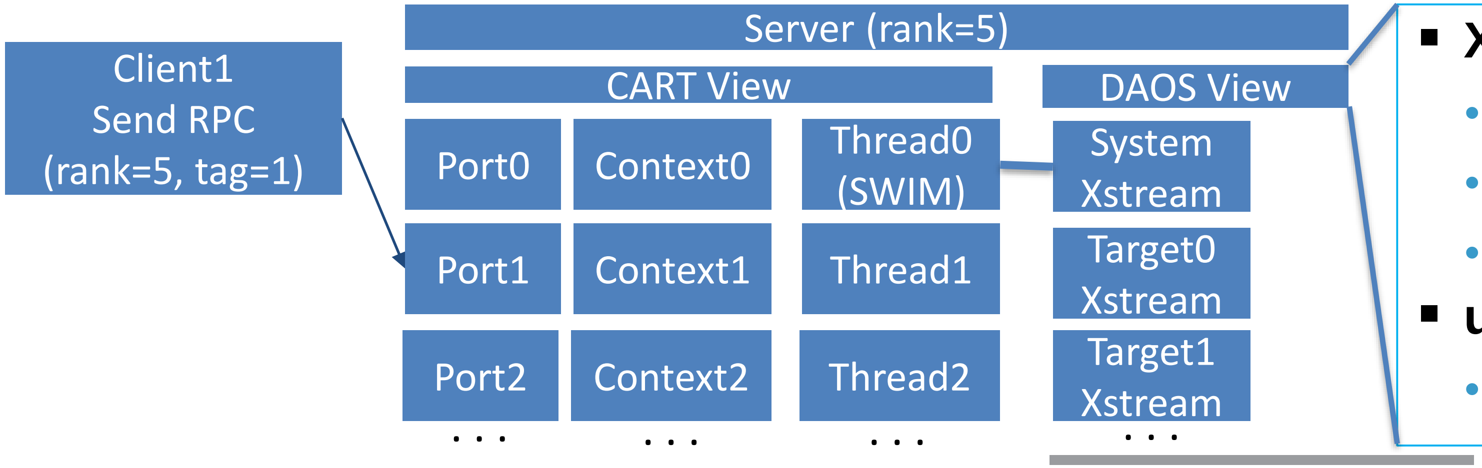
- **Mercury: P2P RPC between targets, pluggable OFI providers**



CART RANKS AND TAGS/CONTEXTS

AND ASSOCIATION WITH DAOS SERVER NODE RESOURCES

- **Rank: Unique 32-bit identifier assigned to each process in a ‘set’ (e.g., DAOS server)**
- **Tag: Number identifying a context/port on the node**
- **Server can create multiple CART contexts, process them independently.**
- **Allows different priorities of executions depending on context receiving a given RPC**



- **Xstream: execution stream (pthread)**
  - System xstream (e.g., for DAOS metadata tasks)
  - 1 Thread / target (often configured based on #cores)
  - Helper threads for target background tasks e.g., rebuild
- **user-level threads (argobots)**
  - Context switching in user-space

CART RPCS

POINT-TO-POINT (P2P)

Timeouts – 3 levels

- All RPCs, per context, per RPC
- Can resend on timeout

Flow Control (Max RPCs in flight to a target)

- Initiated RPCs > limit put on a queue, processed after others complete or timeout

Inline vs. Bulk Transfers

- Messages < ‘eager size’ use send, inline with RPC
- Messages > ‘eager size’ use RDMA either internally (CART/HG) or explicitly (bulk API)

Progress-based model

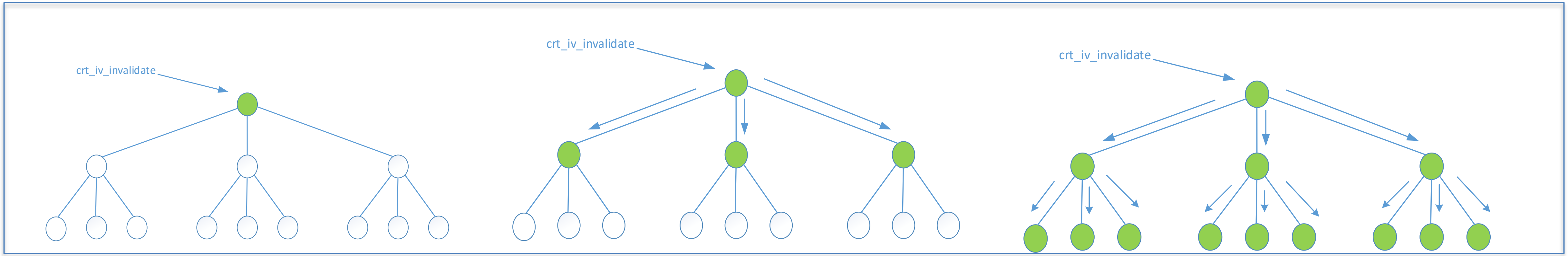
- No actions occur until crt_progress() called (near other calls, or in separate thread)
  - RPC create / send does not perform communication – happens when crt_progress() invoked on the CART context.
- Callback functions invoked from the context of the crt_progress() function
  - E.g., RPC handling (server), RPC reply received (client), bulk transfer completion

CART RPCS
COLLECTIVE

- **Barrier synchronization**
- **Broadcast**
- **Shared Incast Variables (IV)**
  - Scalable Fetch, Update, Invalidate ops
  - Example use: scalable read cache

- **Optional rank filtering for collective over (sub)group**
- **Request propagated through K-ary tree**
- **Reply aggregation**
- **Chained bulk transfer support**
- **Group versioning, error on receiver / sender mismatch**

<u>Example: broadcast used within IV invalidate</u>



LESSONS LEARNED: BUILDING DAOS UPON OFI / LIBFABRIC

CLIENT/SERVER VS MPI MODEL OF USAGE

- **DAOS client/server model, different application lifecycles**
  - Client endpoint addresses can end up being reused – differentiate between 2 client runs reusing same endpoint address
  - Server resources dedicated to a client need to be released on client disconnect
    - Examples: Address Vector (AV) table entry, bounce buffers, memory registrations (MR), …
    - Additional challenges on abrupt disconnect (client failure)

- **MPI usage tends to be more of "launch everything at once, do a job and exit"**
  - Servers: dynamically expand / contract system ; and members of sub-groups associated with storage pools

- **DAOS utilizes multiple contexts, with different type of workloads**
  - Background fault detection protocols e.g., SWIM
  - DAOS metadata activities and request handling
  - I/O processing and background activity (e.g., self-healing / rebuild)

- **DAOS requires multi-tenancy support:**
  - Server can run as a different user (possibly root) compared to clients – some providers do not support this model

SCALABILITY

- **Number of issues found during scalability testing**
  - MR cache: seeing some buffer overwrites / corruption (that do not occur when disabling MR cache)
  - Race conditions: DAOS multi-threaded request processing. Have seen issues in some providers
  - Resource leaks: AV table entries, mem registrations, buffers
    - Additional challenge, concurrent cleanup with MPI-based clients (near simultaneous disconnects)

- **Connection-oriented providers and resource pre-allocation**
  - RXM-based providers pre-allocate resources per client based on FI_UNIVERSE_SIZE
    - CQ size, bounce buffers
  - Address entry in AV table (populated by fi_av_insert())
  - Some resources such as AV table size can be implemented in HW with strict limits on maximum size
  - As a result server has to manage ‘clients’ by evicting stale/old/LRU entries

- **Connection-less RXD provider considered:**
  - Does not require persistent connection – scales higher than connection-oriented providers
  - DAOS cannot use it for now due to no RDMA support - poor performance

TESTING

- **Providers tend to be in various states of stability**

- **Currently sockets provider is main one in DAOS CI**
  - Sockets provider is not performant ; and not actively maintained
  - DAOS would like to move to OFI_RXM;TCP provider, however facing stability issues for now

Wishlist items

- **More automated tests using available providers**
  - With combinations of common / provider-specific variables – e.g., with/without shared rx buffers (FI_OFI_RXM_USE_SRX)

- **Longevity tests**

- **Performance tests**

- **Valgrind/Thread/memory sanitizer tests:**
  - DAOS has a few valgrind memory suppressions for issues seen in providers

POTENTIALLY INTERESTING FABRIC FEATURES

POTENTIAL AREAS OF EXPLORATION - BRAINSTORM

- **Multiple interface solutions on client – e.g., single virtual / bonded interface**
  - A process uses one interface today – with many processes / node (e.g., MPI ranks) interface use can be distributed
  - A single, highly threaded process needs a mechanism to use all interfaces (e.g., for performance and/or fault resilience)
  - Each system (e.g., MPI, DAOS) needs to solve the problem on its own – possible wish list item for OFI / libfabric

- **Lower-latency client-initiated RDMA to server persistent memory:**
  - Smaller scale, special cases assumed
  - Array + Map Conceptual API: preallocated, registered PM

- **Network e.g., switch configured and enforced traffic classes/QoS, traffic management.**
  - current approach: enforce proportion of normal vs. background I/O (e.g., self-healing) through user-level thread scheduling.

- **Network telemetry to diagnose / optimize DAOS service performance**

- **Network offloaded checksum, erasure code, etc.**

RESOURCES

Resource	URL
Source Code on GitHub	https://github.com/daos-stack/daos
Documentation	https://daos-stack.github.io/
Community Mailing List	https://daos.groups.io/
DAOS Solution Brief	https://www.intel.com/content/www/us/en/high-performance-computing/overview.html

OPENFABRICS ALLIANCE

2020 OFA Virtual Workshop

THANK YOU

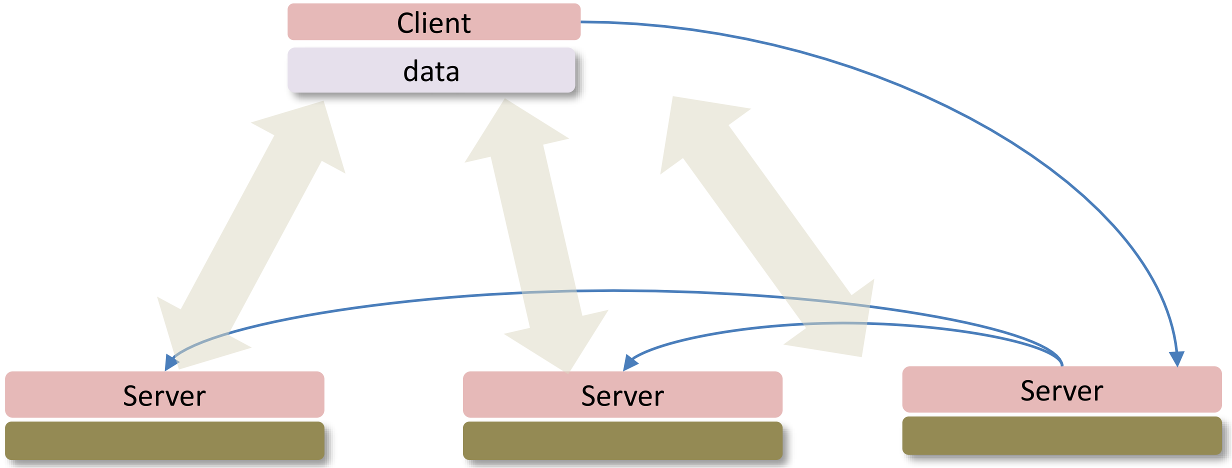
Kenneth Cain, Software Engineer

Intel

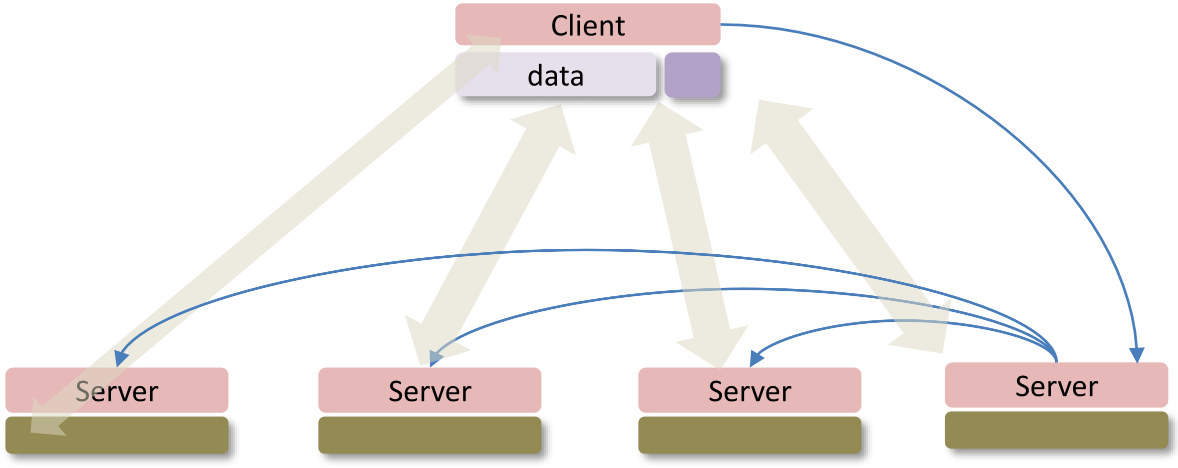
BACKUP SLIDES

DATA PROTECTION AND SELF-HEALING / REBUILD

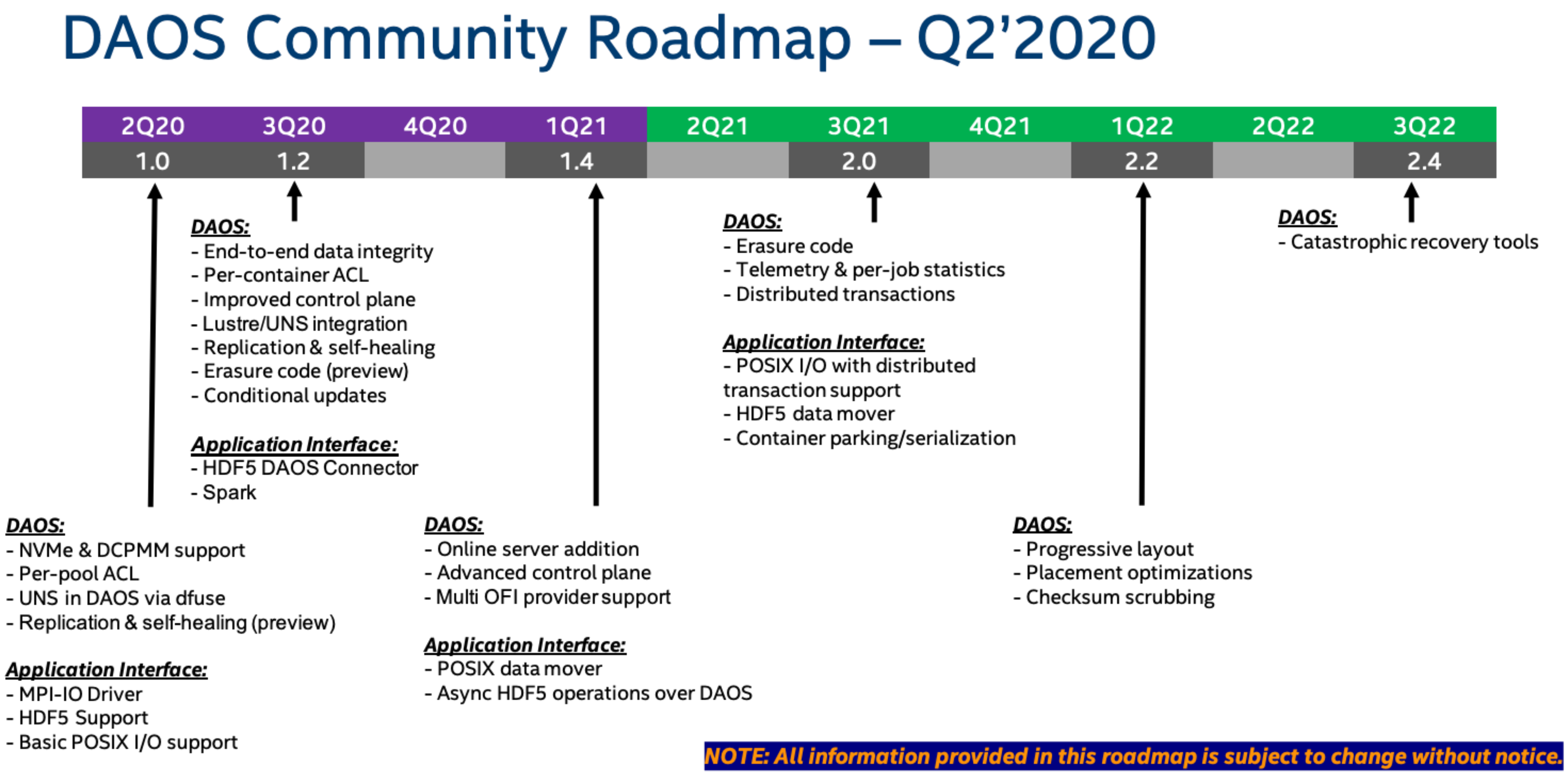
- **Data replication in DAOS**
  - Primary-slave
  - Distributed transaction for atomicity
- **Degraded mode**
  - Client issue I/O requests to surviving server(s)
- **Self-healing / rebuild (online recovery)**
  - Server-side exchange / data reconstruction



- **Erasure code in DAOS**
  - Client compute EC on full stripe write
  - Replication, server merge/encode for partial write
- **Degraded mode**
  - Client-side inflight data reconstruction
- **Self-healing / rebuild (online recovery)**
  - Server-side exchange / data reconstruction



DAOS COMMUNITY ROADMAP



DAOS PERFORMANCE

IO-500 BENCHMARKS

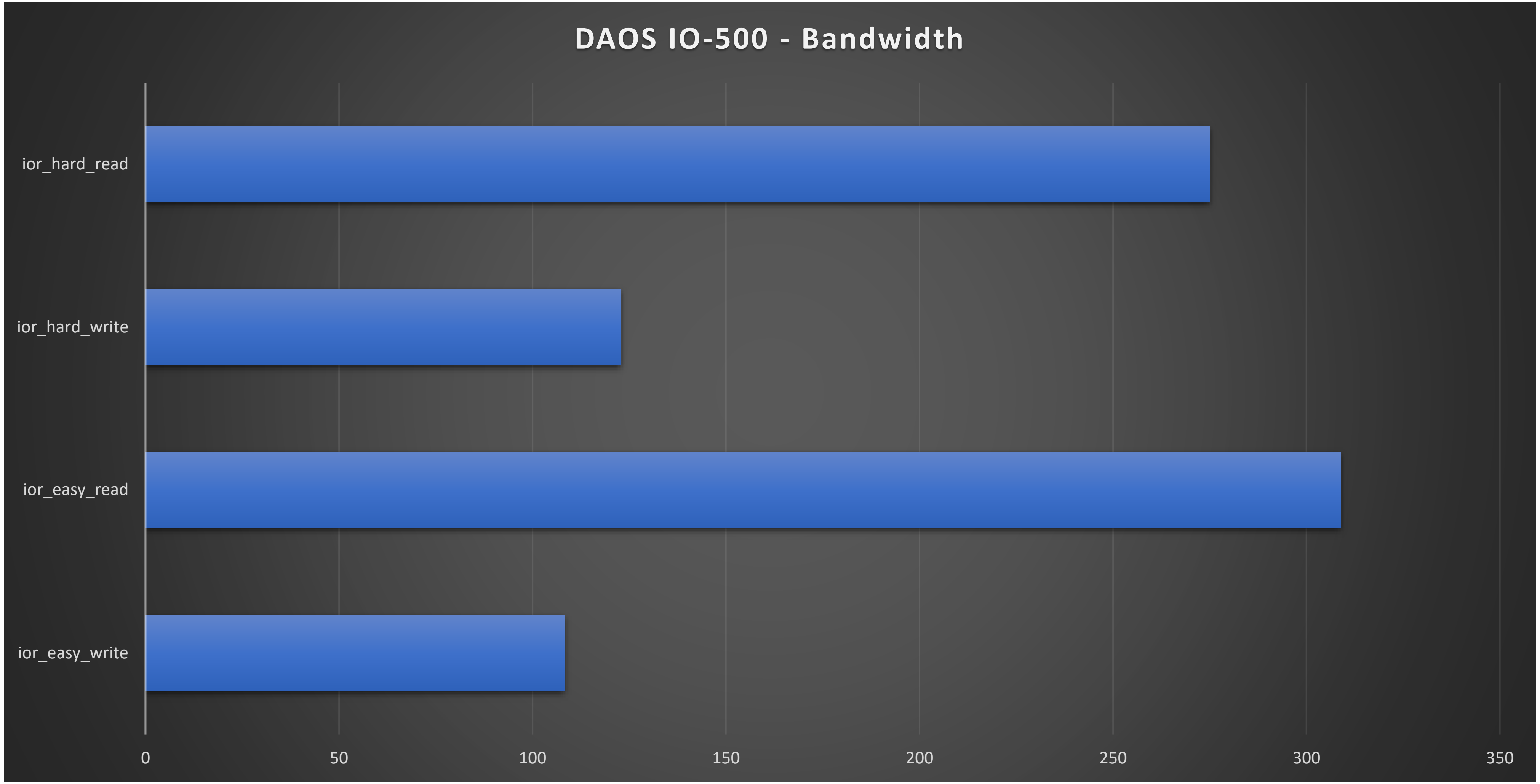
IO-500 Benchmarks

- **IOR**
  - Easy: any IOR pattern to show best-case performance without any explicit caching
  - Hard: single shared file with transfer 47008 bytes!
  - Separate Write and Read/verify runs.
- **mdtest**
  - Easy: private directory per process with empty files
  - Hard: shared directory with 3901-byte files
  - Separate write, read, stat, and delete runs
- **Find**
  - scan namespace created with IOR and mdtest

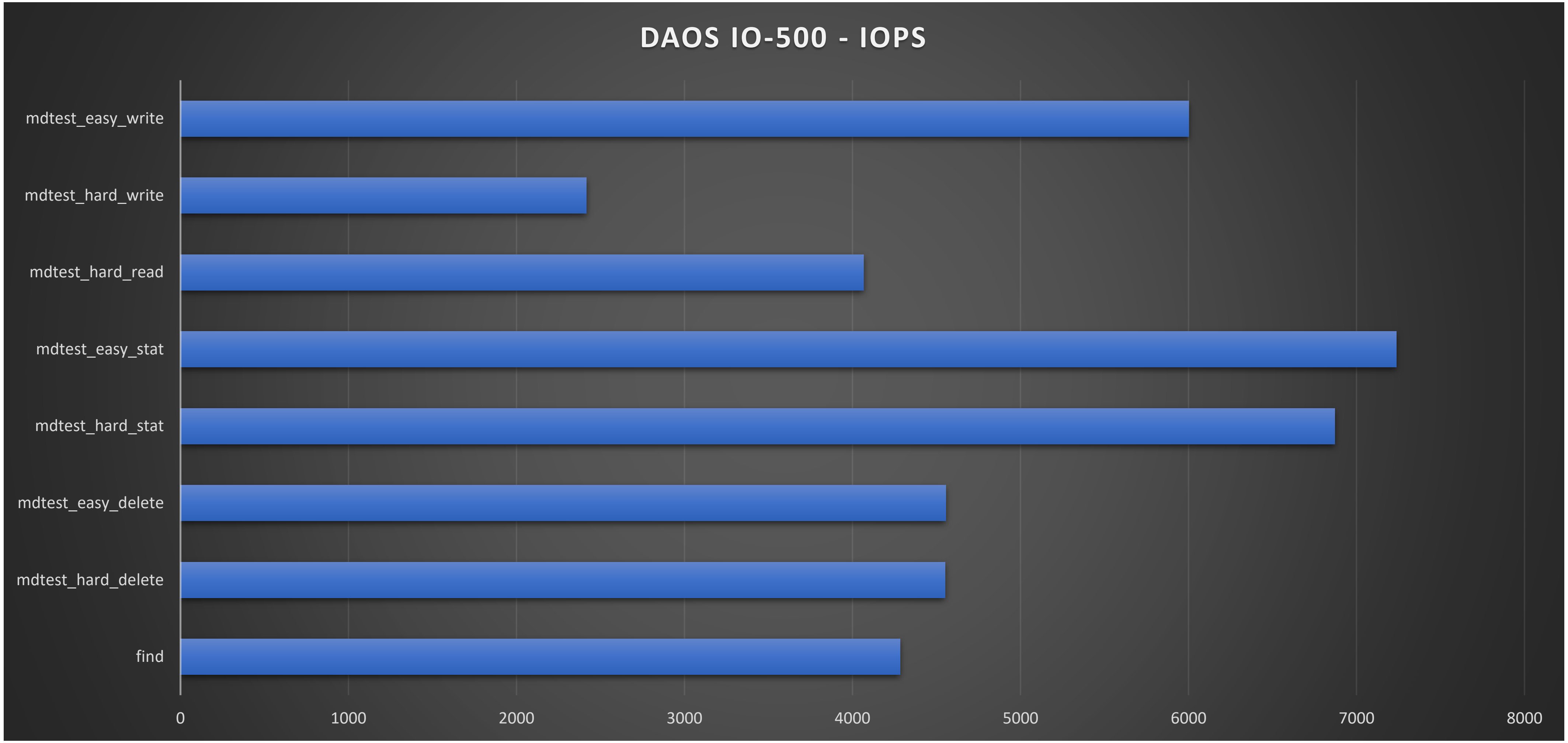
DAOS Testbed

- **10, 26 Compute nodes**
  - 10 node x 31 ranks/node (10 node challenge)
  - 26 node x 28 ranks/node (open challenge)
  - 2x BDW CPU
    - Xeon® E5-2699 v4 @2.2GHz
    - 22 cores per CPU
  - 2x Intel® Omni-Path® 100 adaptors
- **24x Storage nodes**
  - 2x CLX CPU
    - Xeon® Platinum 8260L @ 2.4GHz
    - 24 cores per CPU
  - 12x Optane® DC Persistent Memory DIMMs
    - 500GB each for a total of 3TB
    - Configured in app-direct/interleaved mode
  - 2x Intel® Omni-Path® 100 adaptors

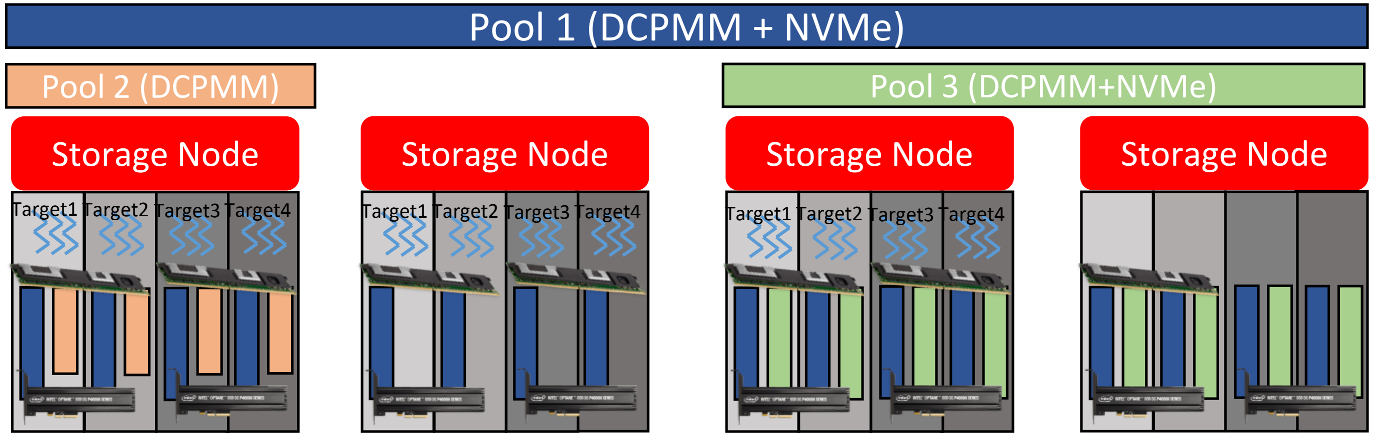
DAOS & IO-500: BANDWIDTH

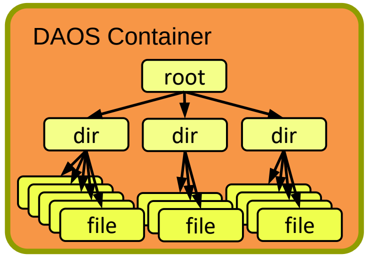


DAOS & IO-500: IOPS

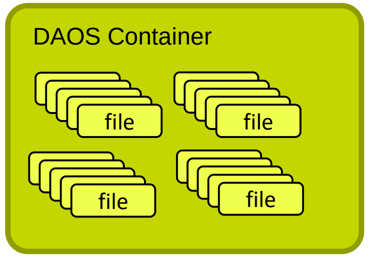


POOLS AND CONTAINERS

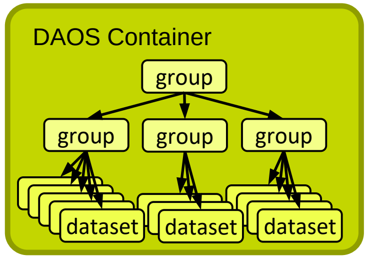




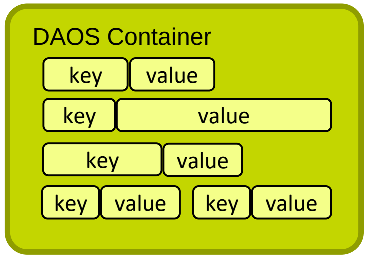
Encapsulated POSIX Namespace



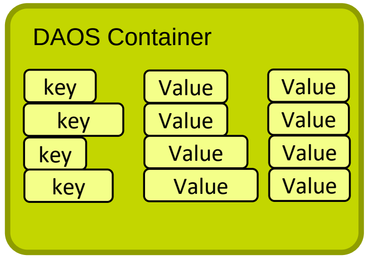
File-per-process



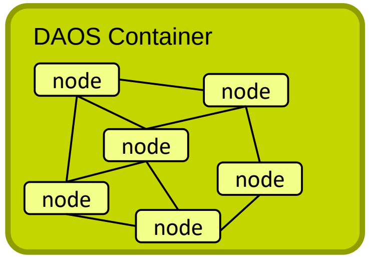
HDF5 « File »



Key-value store



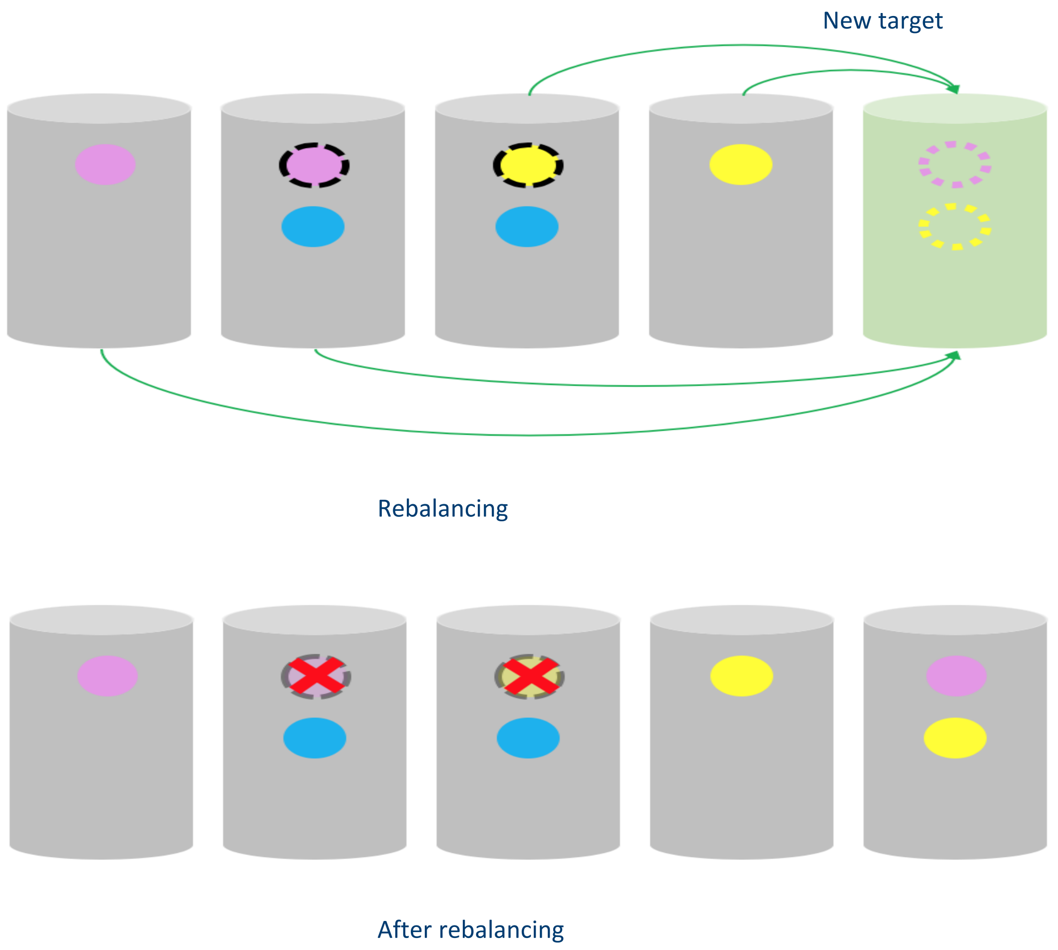
Columnar Database



Graph

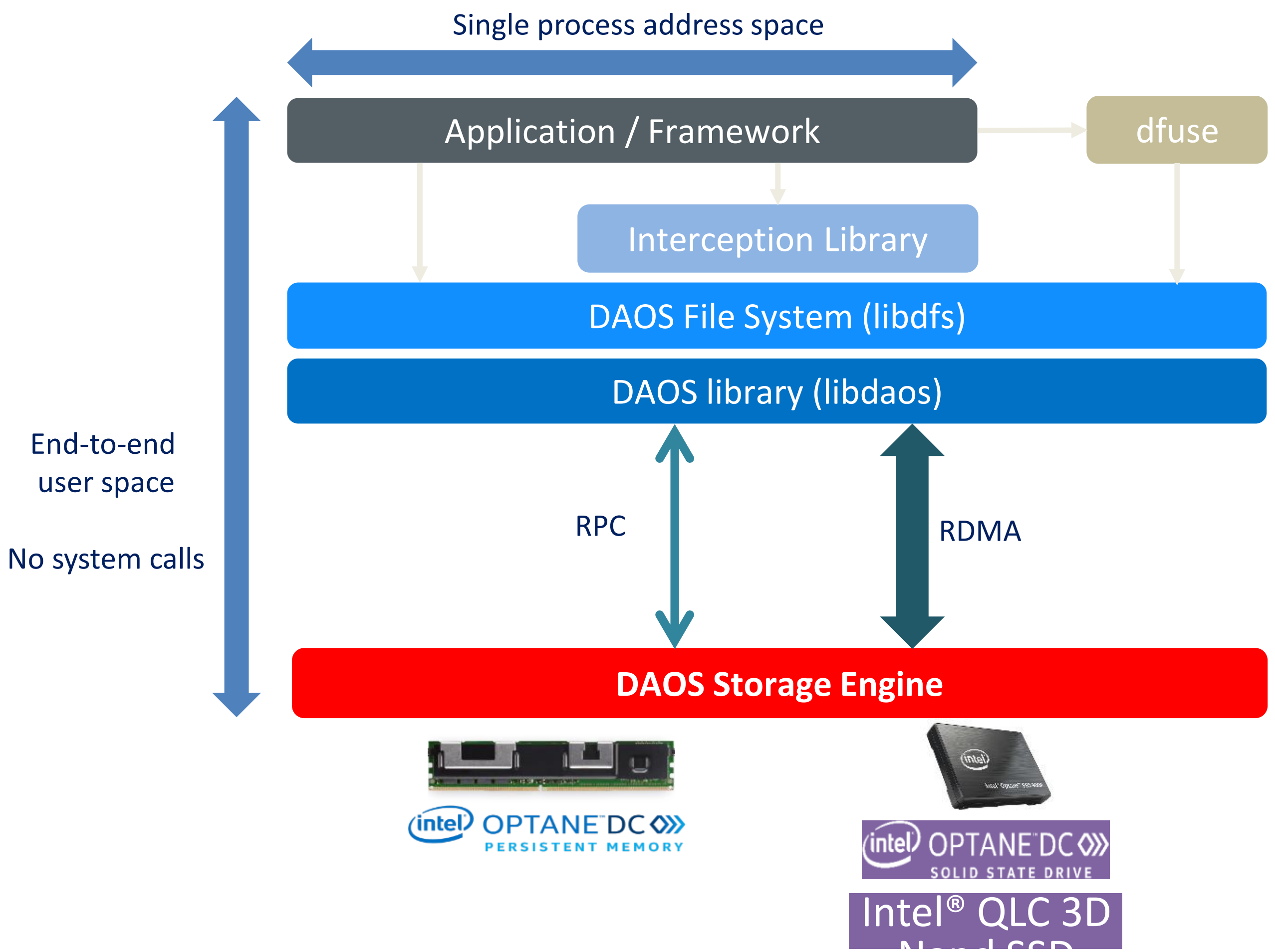
STORAGE TARGET REINTEGRATION / ADDITION

- **Reintegrate recovered target to the pool**
  - Add temporarily excluded storage targets back to the pool
    - Replaced: empty storage target
    - Not replaced: retained data but lagging behind
  - Migrate data back to the reintegrated targets
- **Expand the pool size**
  - Add more nodes/devices to the system
  - Rebalance data within the pool
- **Online data rebalance**



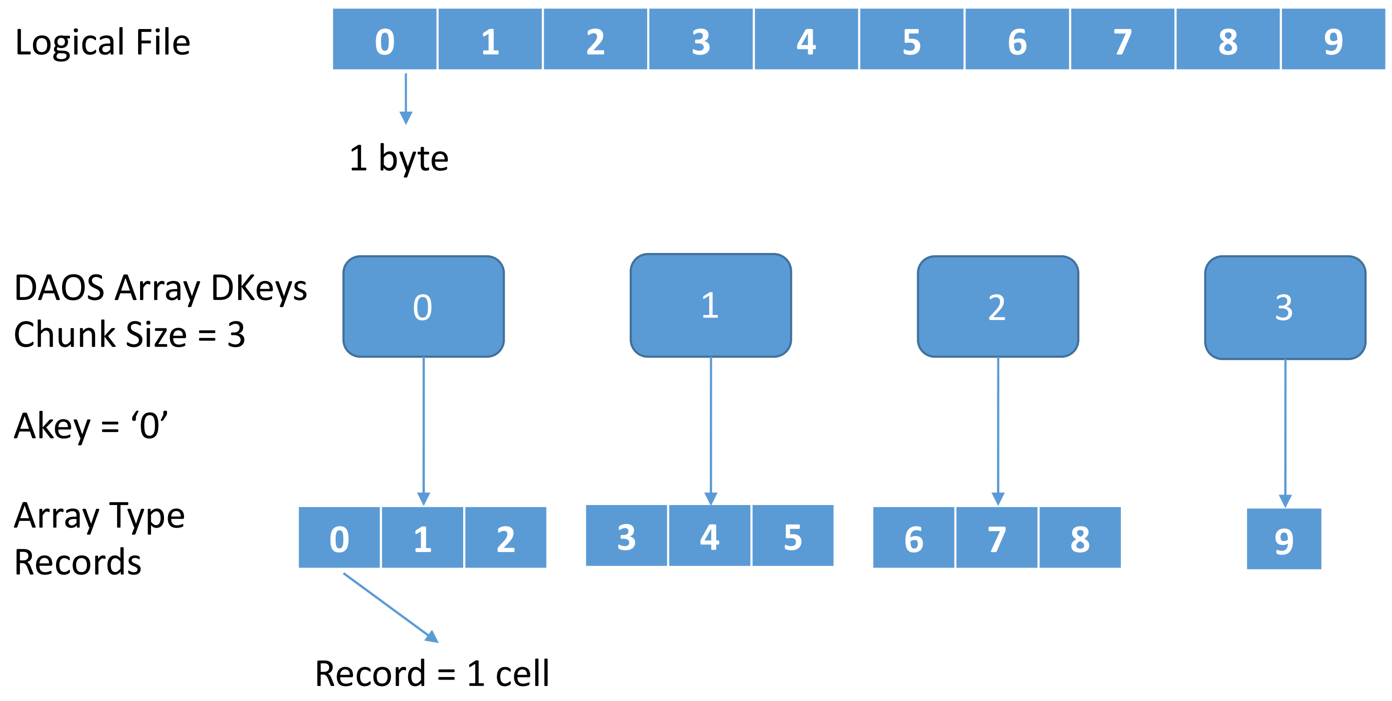
POSIX I/O SUPPORT

- **DAOS File System (libdfs)**
  - Encapsulated POSIX namespace
  - Application/framework can link directly with libdfs
    - ior/mdtest backend provided
    - MPI-IO driver leveraging collective open
    - TensorFlow, …
- **FUSE Daemon (dfuse)**
  - Transparent access to DAOS
  - Involves system calls
- **I/O interception library**
  - OS bypass for read/write operations



MPI-IO DRIVER FOR DAOS

- **The DAOS MPI-IO driver is implemented within the I/O library in MPICH (ROMIO)**
  - Added as an ADIO driver
  - Portable to Open-MPI, Intel MPI, etc.
  - Merged in upstream mpich
  - 1 MPI File = 1 DAOS Array Object

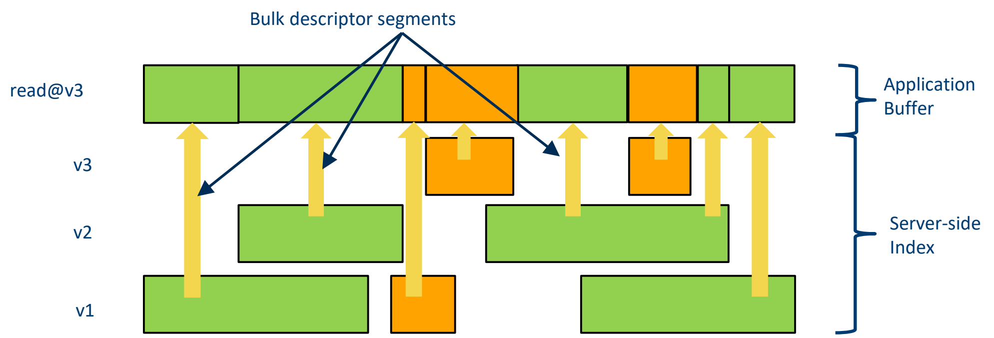


Application works seamlessly by just specifying the use of the driver by appending “daos:” to the path.

FINE-GRAINED I/O

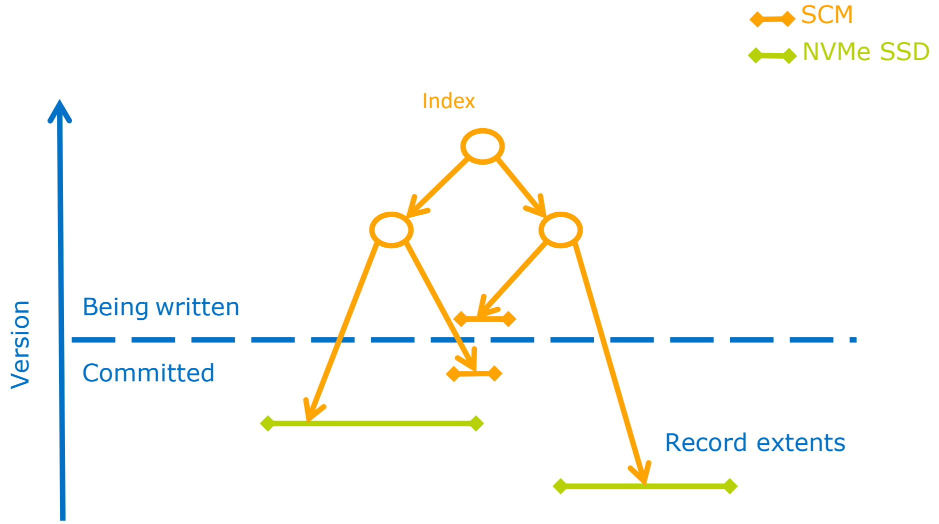
Mix of storage technologies

- **Storage Class Memory (AEP / Optane DC pmem)**
  - DAOS metadata & application metadata (6% min)
  - Byte-granular application data
- **NVMe SSD (*NAND, Optane SSDs)**
  - Cheaper storage for bulk data (e.g. checkpoints)
  - Multi-KB



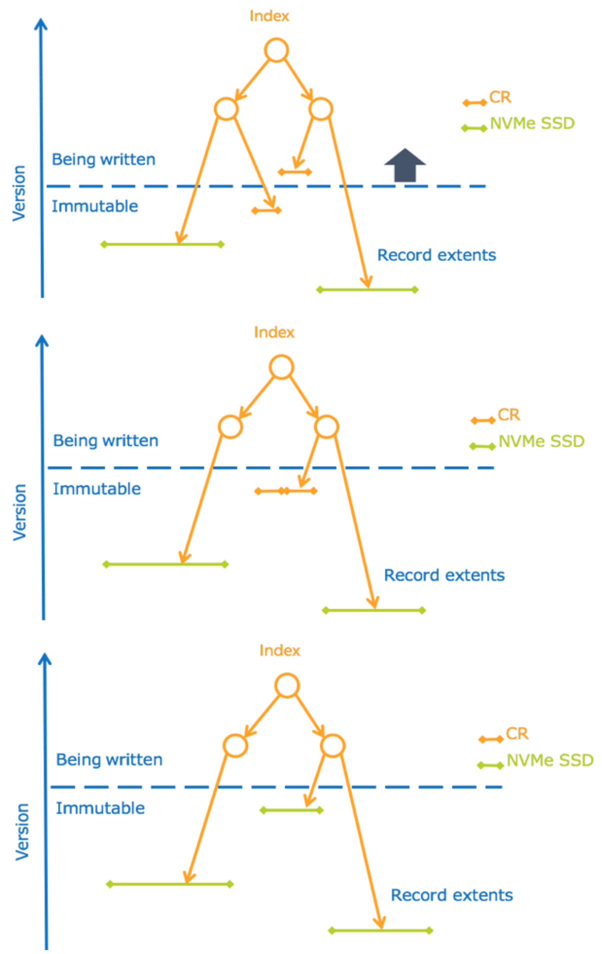
I/Os logged / inserted into persistent index

- **Non-destructive write & consistent read**
- **No alignment constraints**
- **No read-modify-write**

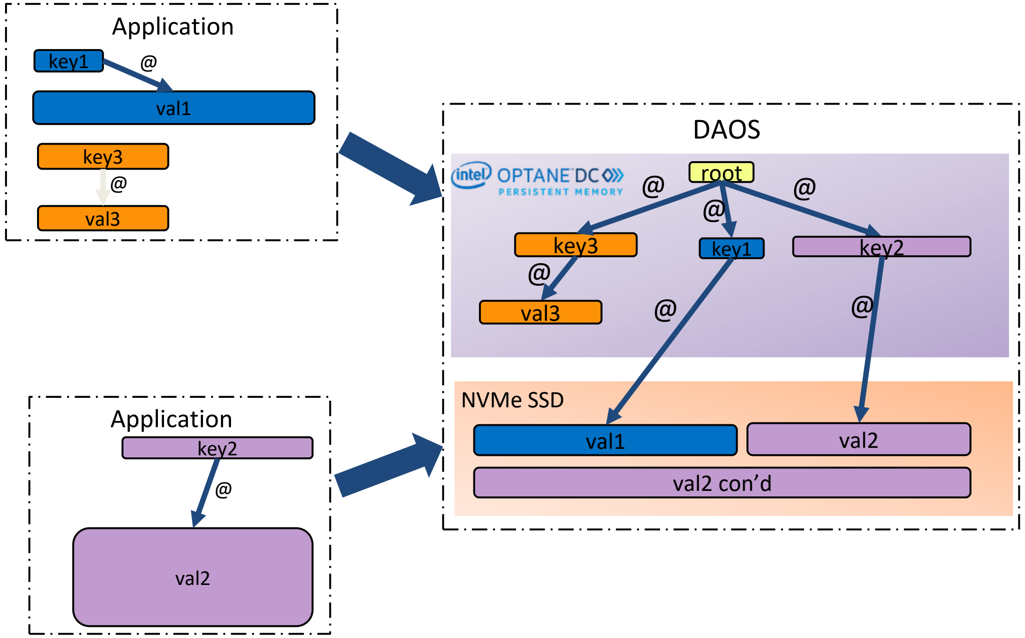


DATA AGGREGATION

- **Merge small extents in DCPMM, migrate to NVMe SSD**
- **Merge extents in NVMe SSD to larger extent**
- **Reclaim old snapshots**
  - Overwrites: delete old version
  - Punch/delete: delete whole subtree
- **EC aggregation**
  - Compute parities for partial writes



ADVANCED STORAGE MODEL



- **Native support for structured, semi-structured & unstructured data models**
  - Built on top of DCPMM
  - Unconstrained by POSIX serialization
  - Custom attributes
  - Data access time orders of magnitude faster (µs)
  - Scalable concurrent updates & high IOPS
  - Non-blocking
  - Enable in-storage computing