Hewlett Packard
Enterprise

DAOS INCITE Training

Agenda

- DAOS on Aurora: speeds and feeds and use cases – 45min
- DAOS with standard interface/middleware – 45min
- DAOS tuning and best practices – 45min

- Lunch break – 45min

- Advanced DAOS APIs – 20min
- Monitoring with darshan – 30min
- Q&A – 30 min - all

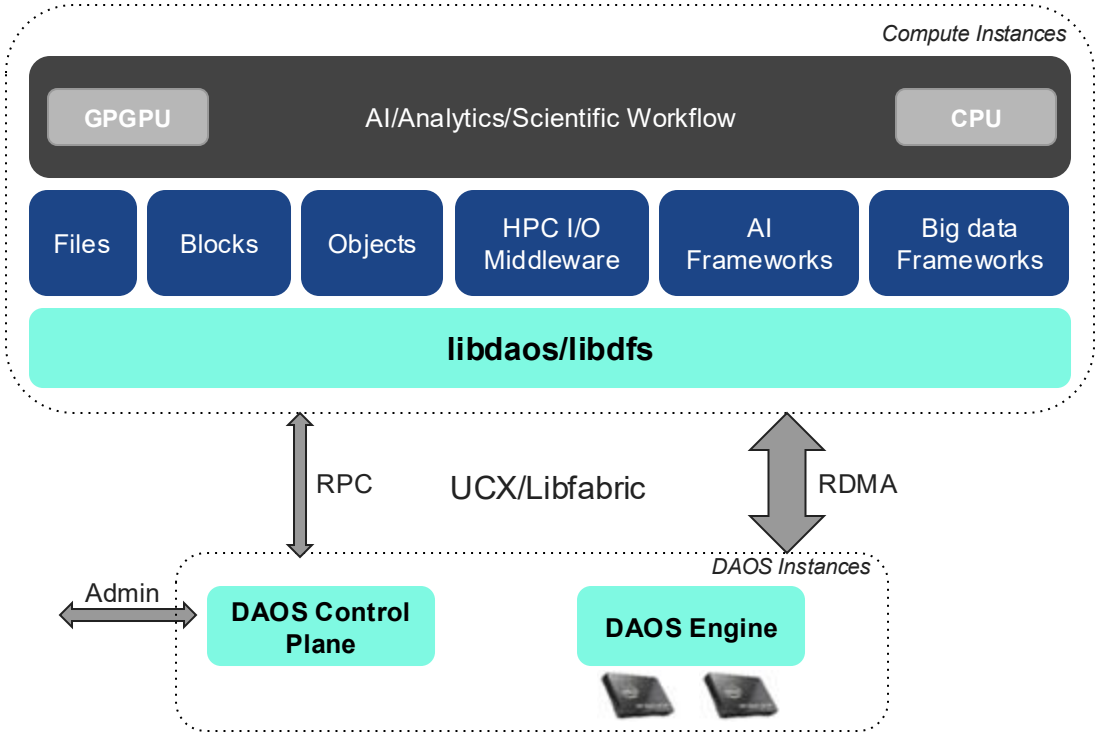
Hewlett Packard
Enterprise

DAOS on Aurora: Speeds and Feeds and Use Cases

Johann Lombardi, Senior Distinguished Technologist, HPE

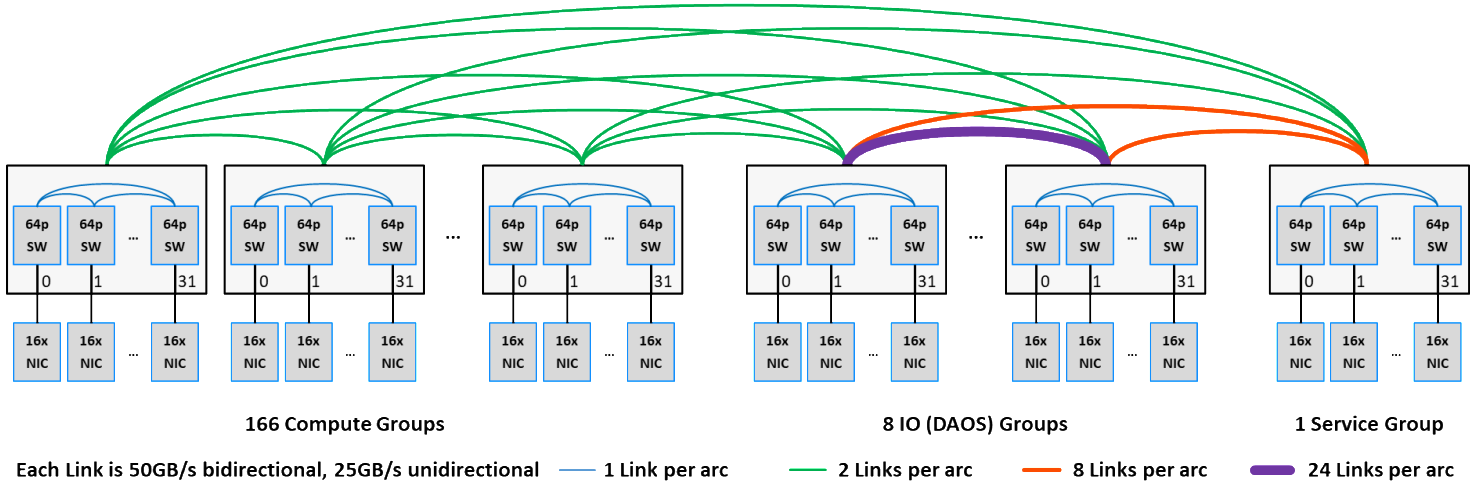
DAOS: Nextgen Open Storage Platform

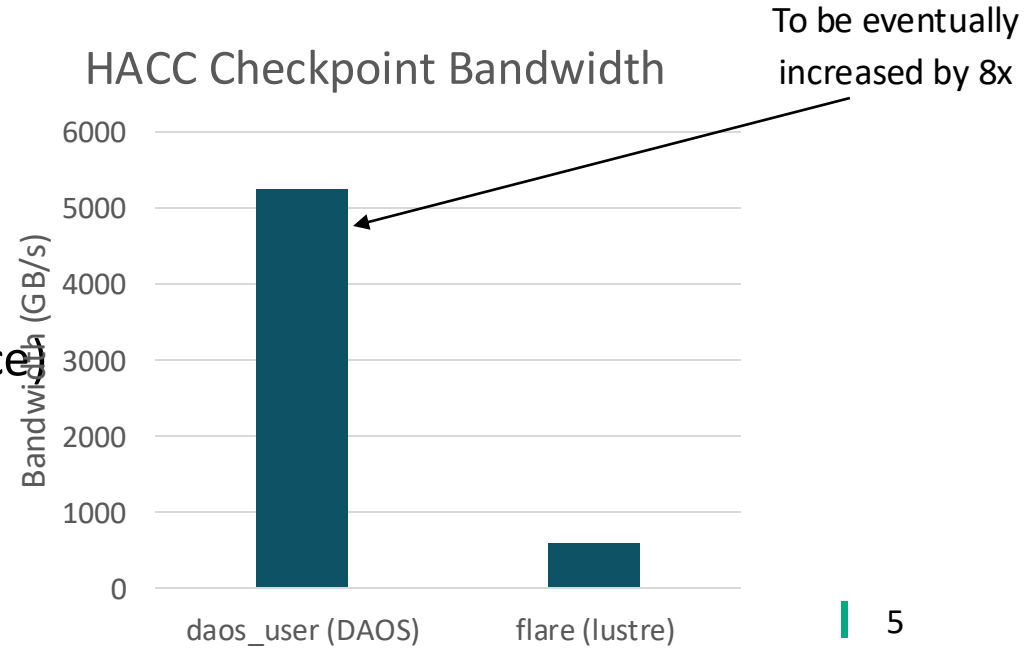
- Open platform for innovation
- Files, blocks, objects and more
- Full end-to-end userspace
- Flexible built-in data protection
  - EC/replication with self-healing
  - No performance impact on reads
- Flexible network layer
- Strong distributed consistency
  - Version-based with MVCC
- Efficient single server
  - O(100)GB/s and O(1M) IOPS per server
- Highly scalable
  - TB/s and billions IOPS of aggregated performance
  - O(1M) client processes
- Time to first byte in O(10) µs



Aurora DAOS System

- 1k storage nodes
  - Namely “aurora-daos-xxxx”
  - Dual NICs/Sockets running 2 DAOS engine
  - Raw capacity of 244TB per node
  - Raw read/write@~40GB/s per node
  - 1k storage nodes spread over 64 racks
  - >220PB and >31TB/s when at full capacity (vs 600GB/s for flare/Lustre)
- Directly connected to the Slingshot fabric
  - 8 DragonFly I/O groups (8 racks per group)
  - Adaptive routing enabled
  - More links between I/O groups for failure handling/rebuild
- daos_users
  - 128 storage nodes (1/8th of eventual total capacity/performance)
  - More nodes to be added gradually
  - read@~5TB/s and write@4.5TB/s





DAOS Use Cases

- Checkpoint/restart
  - High bandwidth and large capacity
  - Integration with HPC frameworks (HDF5, MPIIO, …)
- Out-of-core data / irregular accesses
  - Input/output data for simulation
  - High IOPS/bandwidth and low latency
  - Native POSIX and integration with HPC frameworks (HDF5, MPIIO, …)
- Analytics
  - High IOPS and low latency
  - Integration with Apache ecosystem (Spark)
- AI/ML
  - Optimizations for read-only workloads
  - High write bandwidth for checkpointing
  - Low latency KVCache for inference
  - Large capacity
  - Integration with AI framework (PyTorch)

DAOS 101 Summary

```
$ module use /soft/modulefiles/
$ module load daos
```

Set up Aurora DAOS environment

```
$ daos pool list
  Pool                 Size     State          Used  Imbalance  Disabled
  ----                 ----     -----          ----  ---------  --------
  HPE_test             47 TB    Degraded       3%    0%.        112/4096
```

Check out the pool allocated to your project

```
$ daos cont create HPE_test test_cont
    --type=POSIX --dir-oclass=RP_3G1 --file-oclass=EC_16P2GX --chunk-size=4MiB
    --properties=rd_fac:2,cksum:crc32,srv_cksum:on,ec_cell_sz:128kib
  Successfully created container e1870f74-609f-4ea0-9852-ef4a3de7b5af
    Container UUID : e1870f74-609f-4ea0-9852-ef4a3de7b5af
    Container Label: test_cont
    Container Type : POSIX
```

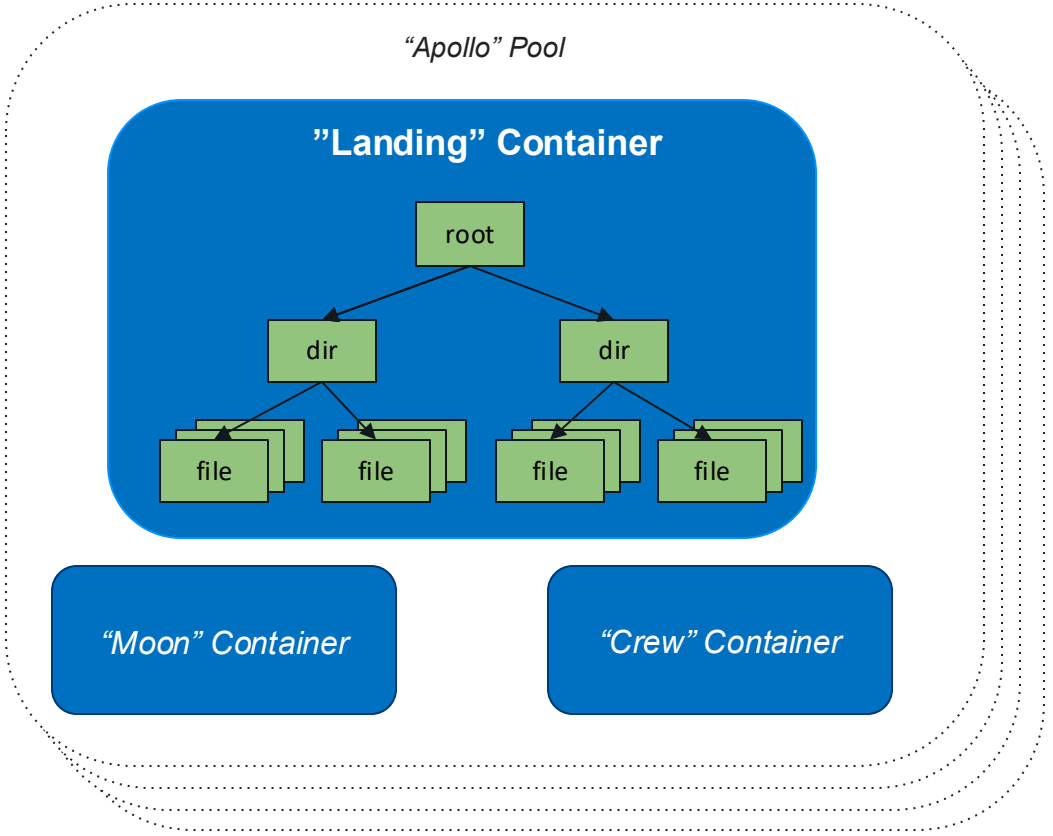
Create a POSIX container namespace

```
$ mkdir -p ~/daos; start-dfuse.sh -m ~/daos --pool HPE_test --cont test_cont

$ cd ~/daos ... cd

$ fusermount3 -u ~/daos
```

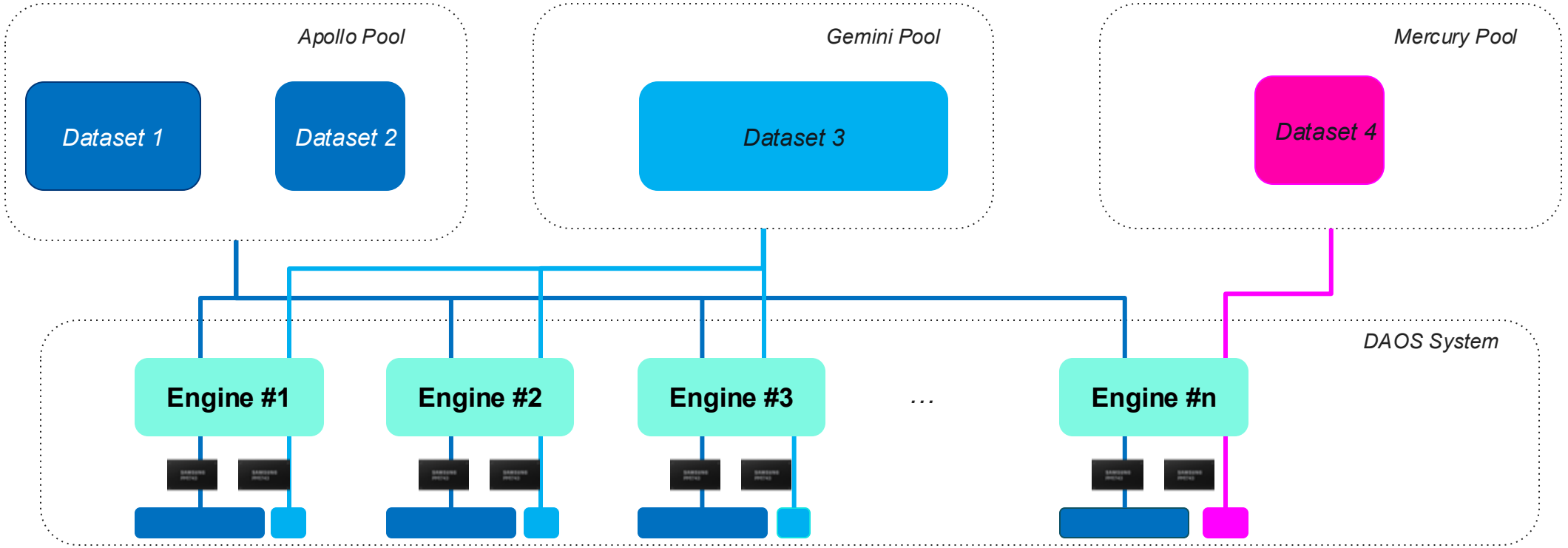
Mount the namespace, Access it … and Unmount it

DAOS Concepts

- Pool
  - private partition/tenant allocated to a project
  - distributed across all the storage nodes
    - Maximum BW/IOPS regardless of size
  - O(100) pools in a system
- Container
  - Dataset/bucket inside a pool
  - O(100) containers in a pool
  - e.g. checkpoints from May campaign
- Object
  - Array or key-value store
  - O(1T) objects in a container
  - e.g. files and directories in a POSIX container



Storage Pooling And Multi-tenancy



Pool 1	Apollo Pool	100PB	20TB/s	200M IOPS
Pool 2	Gemini Pool	10PB	2TB/s	20M IOPS
Pool 3	Mercury Pool	30TB	80GB/s	2M IOPS

DAOS Pool 101

- Find my pool

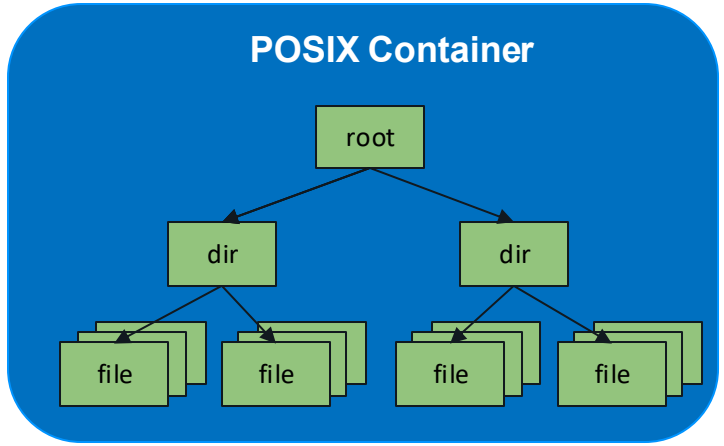
```
$ module use /soft/modulefiles/
$ module load daos
$ daos pool list
Pool                 Size     State          Used Imbalance Disabled
----                 ----     -----          ---- --------- --------
HPE_test             47 TB    Degraded       3%   0%.          112/4096
```

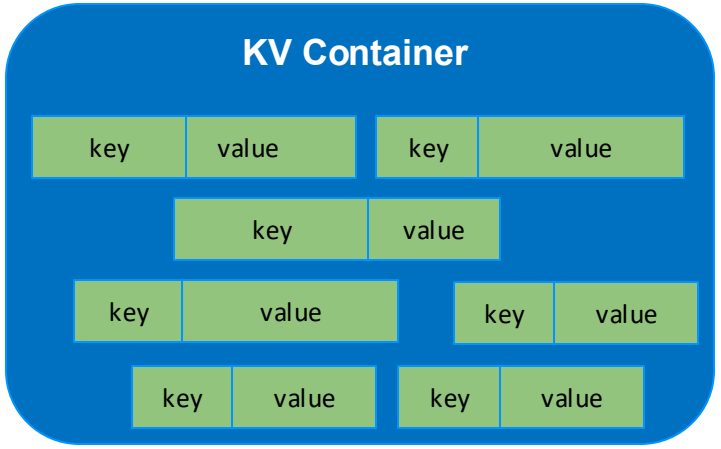
- Query info about my pool

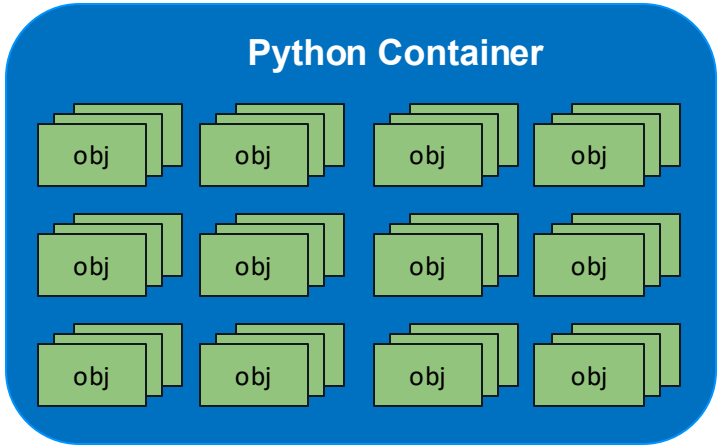
```
$ daos pool query HPE_test
Pool 7d5b9907-1b31-4b65-b308-03947b0cbbc7, ntarget=4096, disabled=112, leader=21, version=634,
state=Degraded
Pool health info:
- Rebuild done, 0 objs, 0 recs
Pool space info:
- Target(VOS) count:3984
- Storage tier 0 (SCM):
  Total size: 146 GB
  Free: 144 GB, min:36 MB, max:36 MB, mean:36 MB
- Storage tier 1 (NVMe):
  Total size: 4.7 TB
  Free: 4.6 TB, min:1.1 GB, max:1.1 GB, mean:1.1 GB
```

DAOS Containers = Datasets/Buckets

- New data model to unwind 30+y of file-based management
- Introduce notion of dataset = container
- Basic unit of storage
- Containers have a type (e.g. POSIX, pyDAOS, …)
- POSIX containers can include trillions of files/directories
- Advanced dataset query capabilities
- Unit of snapshots
- ACLs
- Best practices:
  - 100's containers







DAOS Container 101

- Listing containers in your pool

```
$ daos cont list HPE_test
UUID                                 Label
----                                 -----
ebcbc7f8-db33-4637-951b-dd449b950ead myPOSIXcont
```

- Creating a (POSIX) container

```
$ daos cont create HPE_test test_cont --type=POSIX --dir-oclass=RP_3G1 --file-oclass=EC_16P2GX --
properties=rd_fac:2,cksum:crc32,srv_cksum:on,ec_cell_sz:128kib --chunk-size=4MiB
Successfully created container e1870f74-609f-4ea0-9852-ef4a3de7b5af
  Container UUID : e1870f74-609f-4ea0-9852-ef4a3de7b5af
  Container Label: test_cont
  Container Type : POSIX

$ daos cont list HPE_test
UUID                                 Label
----                                 -----
e1870f74-609f-4ea0-9852-ef4a3de7b5af test_cont
ebcbc7f8-db33-4637-951b-dd449b950ead myPOSIXcont
```

DAOS Container 101

- Querying your container

```
$ daos cont query HPE_test test_cont
  Container UUID              : e1870f74-609f-4ea0-9852-ef4a3de7b5af
  Container Label             : test_cont
  Container Type              : POSIX
  Pool UUID                   : 7d5b9907-1b31-4b65-b308-03947b0cbbc7
  Container redundancy factor : 2
  Number of open handles      : 1
  Latest open time            : 0x1e7ed8192c200000 (2025-05-09 13:50:54.426841088 +0000 UTC)
  Latest close/modify time    : 0x1e7ed8193a240003 (2025-05-09 13:50:54.441537536 +0000 UTC)
  Number of snapshots         : 0
  Object Class                : UNKNOWN
  Dir Object Class            : RP_3G1
  File Object Class           : EC_16P2GX
  Chunk Size                  : 4.0 MiB
```

DAOS Container 101

- Checking container properties

```
$ daos cont get-prop HPE_test test_cont
Properties for container test_cont
Name                                            Value
----                                            -----
Highest Allocated OID (alloc_oid)               0
Checksum (cksum)                                crc32
Checksum Chunk Size (cksum_size)                32 KiB
Compression (compression)                       off
Deduplication (dedup)                           off
Dedupe Threshold (dedup_threshold)              4.0 KiB
EC Cell Size (ec_cell_sz)                       128 KiB
Performance domain affinity level of EC (ec_pda) 1
Encryption (encryption)                         off
Global Version (global_version)                 3
Group (group)                                   users@
Label (label)                                   test_cont
Layout Type (layout_type)                       POSIX (1)
Layout Version (layout_version)                 1
Max Snapshot (max_snapshot)                     0
Object Version (obj_version)                    1
Owner (owner)                                   jlo@
Performance domain level (perf_domain)          root (255)
Redundancy Factor (rd_fac)                      rd_fac2
Redundancy Level (rd_lvl)                       node (2)
Performance domain affinity level of RP (rp_pda) 4294967295
Server Checksumming (srv_cksum)                 on
Health (status)                                 HEALTHY
Access Control List (acl)                       A::OWNER@:rwdtTaAo, A:G:GROUP@:rwtT
```

DAOS Container 101

- Storing user metadata for a container

```
$ daos cont list-attr HPE_test test_cont -V
Attributes for container test_cont:
  No attributes found.

$ daos cont set-attr HPE_test test_cont description:"dataset including pressure and
temperature”

$ daos cont list-attr HPE_test test_cont -V
Attributes for container test_cont:
Name         Value
----         -----
description dataset including pressure and temperature

$ daos cont del-attr HPE_test test_cont description

$ daos cont list-attr HPE_test test_cont -V
Attributes for container test_cont:
  No attributes found.
```

DAOS Container 101

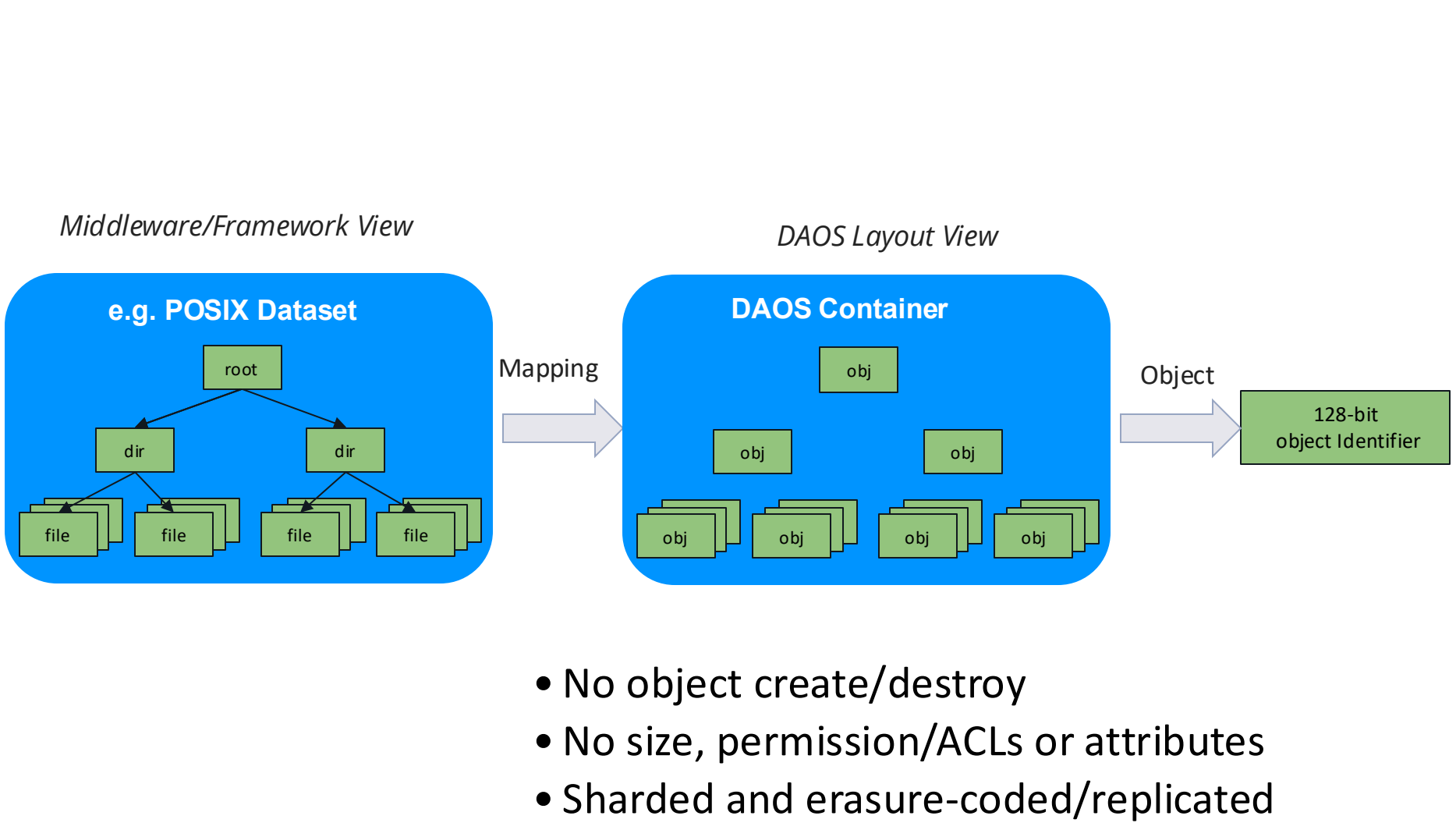
- Renaming a container

```
$ daos cont set-prop HPE_test test_cont label:new_cont
Properties were successfully set

$ daos cont list HPE_test
UUID                                 Label
----                                 -----
e1870f74-609f-4ea0-9852-ef4a3de7b5af new_cont
ebcbc7f8-db33-4637-951b-dd449b950ead myPOSIXcont
```

- Destroying a container

```
$ daos cont destroy HPE_test new_cont
Successfully destroyed container new_cont
```

Object Interface



- No object create/destroy
- No size, permission/ACLs or attributes
- Sharded and erasure-coded/replicated
- Algorithmic object placement
- Very short Time To First Byte (TTFB)

DAOS Object 101

Retrieving information about your POSIX container (single process)

```
$ daos fs scan HPE_test new_cont
DFS scanner: Start (2025-05-10-13:59:22)
DFS scanner: Scanned 7236 files/directories (runtime: 30 sec)
DFS scanner: Scanned 14315 files/directories (runtime: 60 sec)
DFS scanner: Done! (runtime: 87 sec)
DFS scanner: 21461 scanned objects
DFS scanner: 19201 files
DFS scanner: 101 symlinks
DFS scanner: 2159 directories
DFS scanner: 15 max tree depth
DFS scanner: 809438484 bytes of total data
DFS scanner: 42156 bytes per file on average
DFS scanner: 170912509 bytes is largest file size
DFS scanner: 665 entries in the largest directory
```

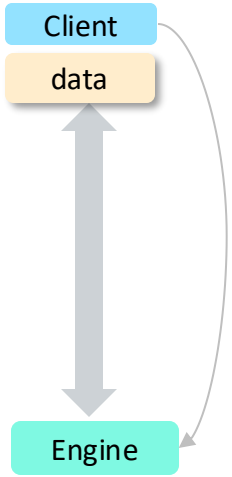
DAOS Object 101

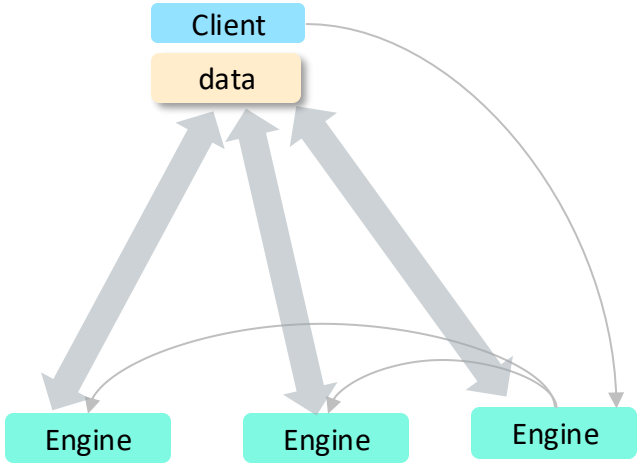
Retrieving information about an object (advanced)

```
$ daos fs  get-attr -H /daos/src HPE_test new_cont
OID = 2533280060991858.0
Object Class = RP_3G1
Directory Creation Object Class = RP_3G1
File Creation Object Class = EC_16P2GX
File Creation Chunk Size = 4194304

$ daos obj query -i 2533280060991858.0 HPE_test new_cont
oid: 2533280060991858.0 ver 0 grp_nr: 1
grp: 0
replica 0 204:10
replica 1 209:9
replica 2 56:5
```

Data Protection and Performance Trade-offs

No Data Protection
Full bandwidth/IOPS on
both read & write

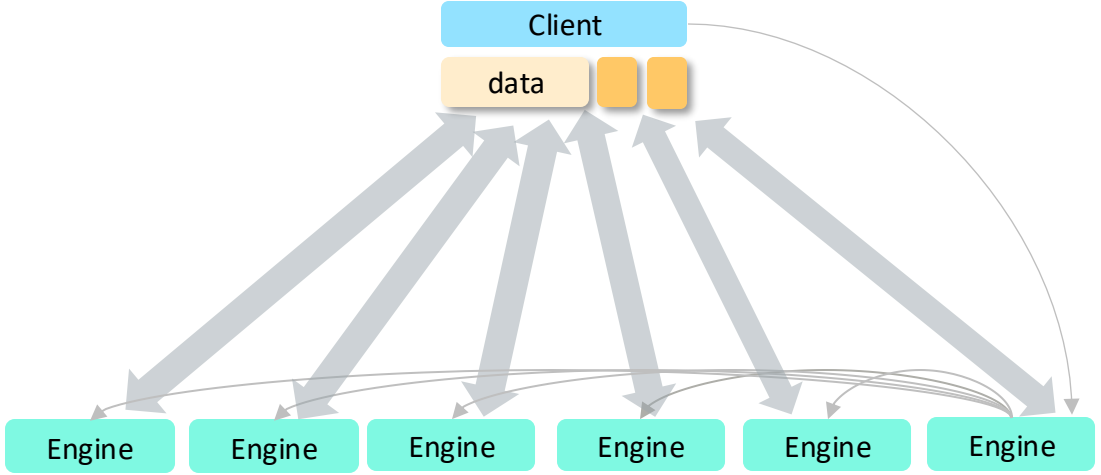




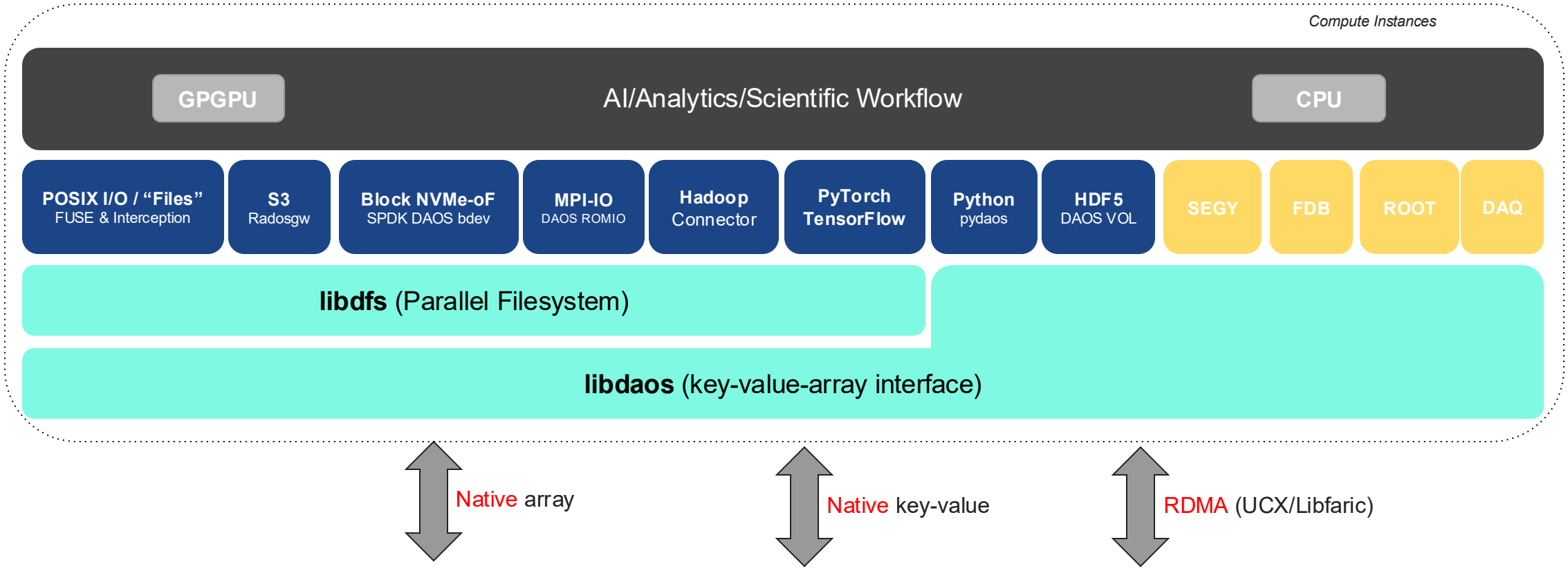
3-way Replication
Full bandwidth/IOPS on read
66% loss on write

16+2 Erasure Code
Full bandwidth/IOPS on read
12% loss on full stripe write
66% loss on partial write

4+2 Erasure Code
Full bandwidth/IOPS on read
33% loss on full stripe write
66% loss on partial write
(replicated turned into EC in background)



Software Ecosystem



Generic I/O Middleware/frameworks

Domain-specific data models under development in co-design with partners

Data Mover

- Quick single-node copy: daos utility

```
$ daos fs copy -s ~/daos/src/include -d daos://HPE_test/new_cont/
Successfully copied to DAOS: daos://HPE_test/new_cont/
    Directories: 5
    Files:       113
    Links:       0
```

- Parallel multi-node copy: mpiFileUtils
  - 100 gateway nodes for data movement (not configured yet)
  - Parallel copy (splitting both large dirs/files) using DAOS-aware mpiFileUtils

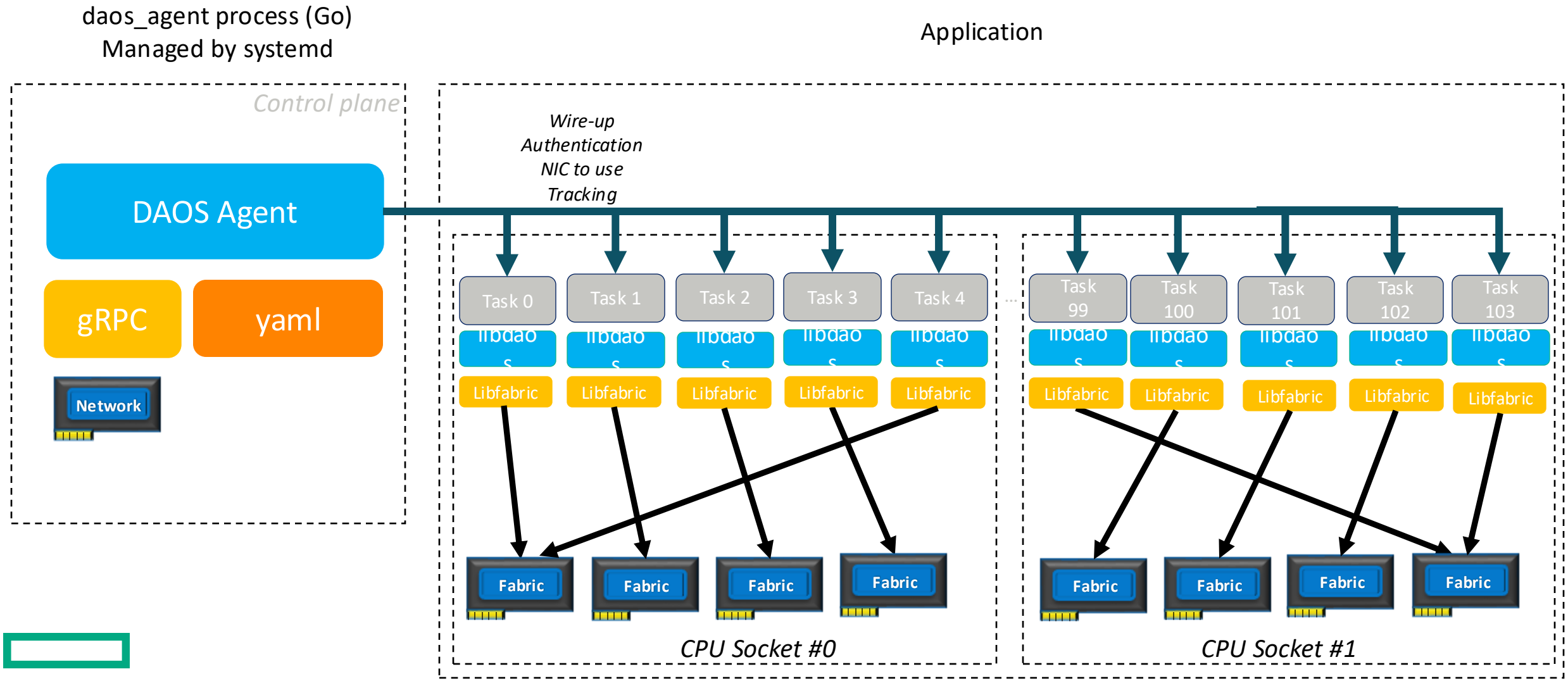
```
$ mpirun -np 100 dcp -v ~/daos/src/include daos://HPE_test/new_cont
```

DAOS Fault Handling

- Automatic online self-healing
  - Storage nodes monitor each other (SWIM protocol)
  - “Rebuild” is triggered to restore data redundancy/protection on surviving nodes when a node fails
  - The rebuild process is done online and should complete quickly
  - Failed storage node to be reintegrated by administrator once issue is fixed
- Checking rebuild status

```
$ daos pool query HPE_test
Pool 7d5b9907-1b31-4b65-b308-03947b0cbbc7, ntarget=4096, disabled=112, leader=21, version=634,
state=Degraded
Pool health info:
- Rebuild busy, 45 objs, 412318398 recs
Pool space info:
- Target(VOS) count:3984
- Storage tier 0 (SCM):
  Total size: 146 GB
  Free: 144 GB, min:36 MB, max:36 MB, mean:36 MB
- Storage tier 1 (NVMe):
  Total size: 4.7 TB
  Free: 4.6 TB, min:1.1 GB, max:1.1 GB, mean:1.1 GB
```

DAOS Client Structure & NIC Assignment

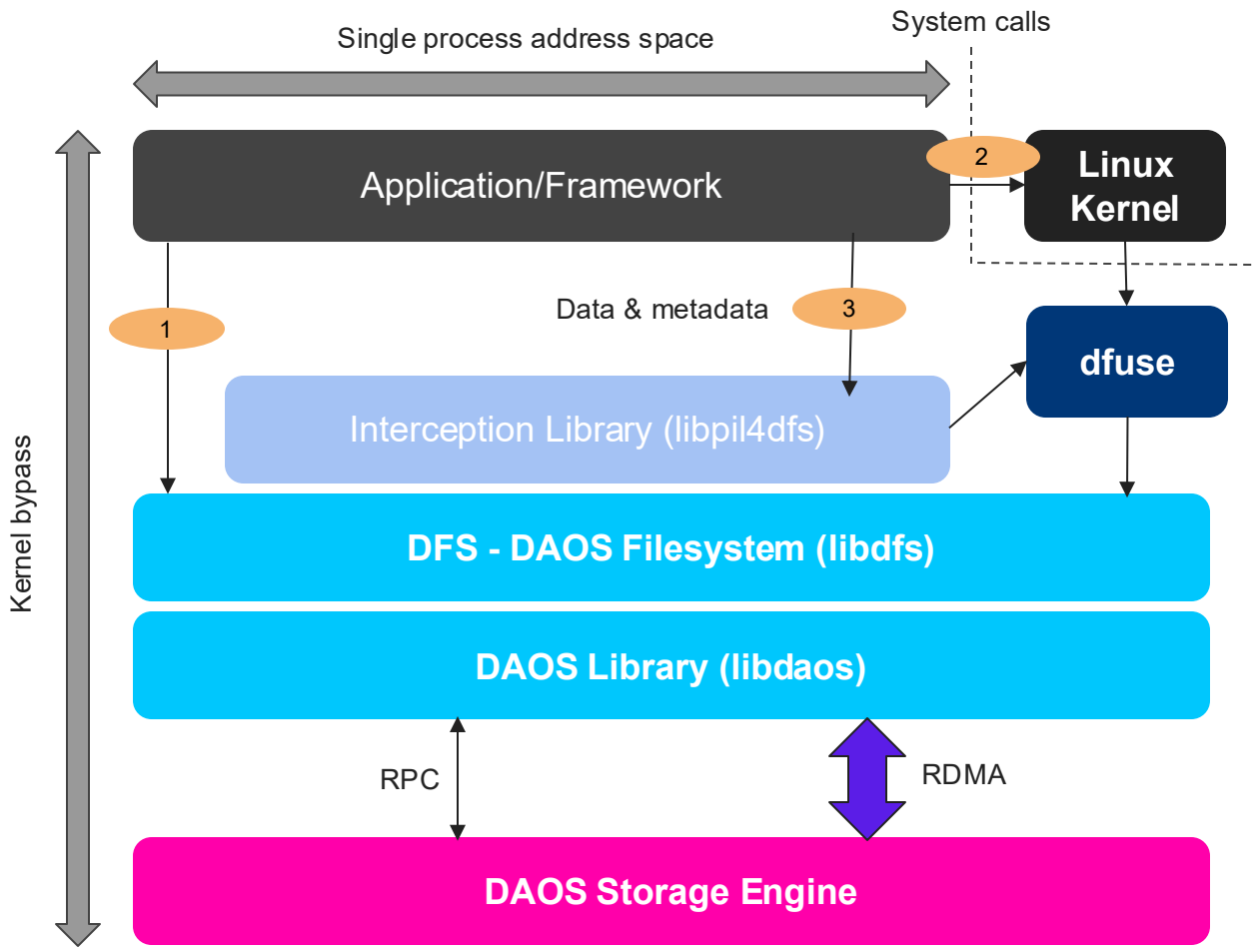


Hewlett Packard
Enterprise

DAOS with Standard Interface/Middleware

Mohamad Chaarawi, Distinguished Technologist, HPE
Johann Lombardi, Senior Distinguished Technologist, HPE

POSIX Containers



1. Userspace DFS library with API like POSIX
   - **Require** application changes
   - Low latency & high concurrency
   - No caching
2. DFUSE daemon to support POSIX API
   - **No** application changes
   - VFS mount point & high latency
   - Caching by Linux kernel
3. DFUSE + Interception library
   - Aim at delivering same performance as #1
   - **No** application changes
   - Using LD_PRELOAD intercepting:
     - Data & metadata interception
     - Mmap & binary execution via fuse
   - Compatibility to offload fd creation to fuse

dfuse

DAOS Fuse

- DAOS provides a dfuse daemon where you can mount a POSIX on all clients at a specific path:
  - DAOS Container becomes a parallel file system
  - Must be mounted by user on a specific path on every client node
  - Does not require any elevated privileges
  - Any user with access to the pool/container can mount the same containers on their nodes concurrently
  - User unmounts dfuse when they are done (PBS epilogue on aurora does it automatically)
- How to mount and umount? From the compute node:

```
launch-dfuse.sh ${DAOS_POOL}:${CONT}
clean-dfuse.sh ${DAOS_POOL}:${DAOS_CONT}
Mount point: /tmp/$USER/$DAOS_POOL/$DAOS_CONT
```

- Manually:
  - `clush --hostfile $PBS_NODEFILE "dfuse --pool=pool --container=cont -m /tmp/$USER/dfuse/ --disable-caching"`
  - `clush --hostfile $PBS_NODEFILE "fusermount -u /tmp/mschaara/dfuse/"`

- Mount Options:
  - --disable-caching: required if doing read/write from multiple nodes
    – Can remove that if running on a single node or doing read only

DFUSE – Use Cases

- Normal file system operations
  - Navigate through your namespace
  - Use as a path for your app / IO
- Node local apps:
  - Single node, caching enabled
  - Might need tuning of container attributes to extend caching time
- Performance:
  - Very limited – not intended for large BW and IOPS
  - Use in combination with interception or MPIIO driver (next)
- Multi-user dfuse:
  - Access to multiple users on a single node at a single mount point
  - Does not make sense to use on ECB, but only on login nodes
    - Not to be confused with users mounting the same container themselves
  - Sys admin creates a dedicated unix account for this to mount the container

Interception Library

DAOS Interception Libraries

- DAOS provides 2 interception libraries that can be loaded at runtime to intercept application calls for optimizing file system operations:
  - libioil: intercepts basic IO calls (read/write, pread/pwrite, fread/fwrite, etc.)
  - libpil4dfs: intercepts all glibc calls and handles most file operation (data and metadata) through the DFS API
- Why 2?
  - Original implementation (ioil) proved to be insufficient for several use cases
  - pil4dfs addresses those issues, but still can be challenging in its own way (glibc interception with trampoline)
- How to use interception:

```
mpiexec -genv LD_PRELOAD=/usr/lib64/libpil4dfs.so … ./a.out
```

- Avoid such usage:

```
export LD_PRELOAD=/usr/lib64/libpil4dfs.so
…
mpiexec … ./a.out

or
LD_PRELOAD=/usr/lib64/libpil4dfs.so mpiexec … ./a.out
```

Some Popular Questions/Concerns

- Do I need dfuse mounted if im using interception?
  - Yes.
- I don’t know if interception is working or not, can I tell?
  - Yes. add env variable: `export D_IL_REPORT=1`
  - Prints a report at the end of your program execution (per process)
- I get an error when using pil4dfs:
  - `[Errno 95] Operation not supported`
  - `export D_IL_COMPATIBLE=1`
  - Contact any DAOS admin or post on the aurora channel so we can investigate the operation causing this.

MPI-IO

MPIIO – Parallel IO

- Part of MPI Standard
- Collective IO optimizations:
  - Aggregation of small IO across all your processes
  - Collective File Create/Open (if file system supports it)
- Supports multitude of access patterns
  - Derived MPI datatype and File Views
- Building block for parallel IO libraries
  - HDF5, PnetCDF, etc.
- Chances are if you are doing parallel IO on Aurora, your App has an MPIIO backend

MPIIO with DAOS

- MPICH ROMIO (MPIIO implementation) has ADIO drivers
  - Abstraction layer over different file systems
  - UFS driver uses POSIX IO for all POSIX file systems
  - UFS driver works with DAOS through dfuse
- DAOS provides an ADIO driver
  - Available on aurora
  - Bypasses dfuse/POSIX API and uses the DFS API directly (faster)
  - Provides Collective File Create/Open (only 1 rank talks to the file system and Bcast the handle)
  - Has a list-io interface (no read-modify-write)
- How do I use the DAOS driver?
  - Used by default when path of a file is on a dfuse file system (DAOS container mounted with dfuse)
  - MPICH detects that the file system is DAOS and uses the driver by default
  - No need to set the env "`ROMIO_FSTYPE_FORCE=daos`:" anymore
  - No application code changes needed

MPIIO w/ DAOS Optimizations

- If the DAOS ADIO driver bypasses dfuse, why do I need to mount dfuse?
  - Application still needs a path to the file: (/mnt/daos/dir1/file)
  - Path is queried from the dfuse daemon to acquire the pool and container information, so the driver connects to them
  - *But is there really no way but to have dfuse mounted?*
    - There is with direct pool/container lookup using an env variable
    - Not straightforward always so let's discuss that offline if that is something needed.
- Tunable:
  - Collective buffering: a very good idea.
    - Avoid some pathological access patterns that can cause slowdowns
    - One way to set those is through env variable:
  - CB buffer size
  - Aggregators per node
  - Striping unit (usually == DFS chunk size)

Create a file in flare or dfuse and add:

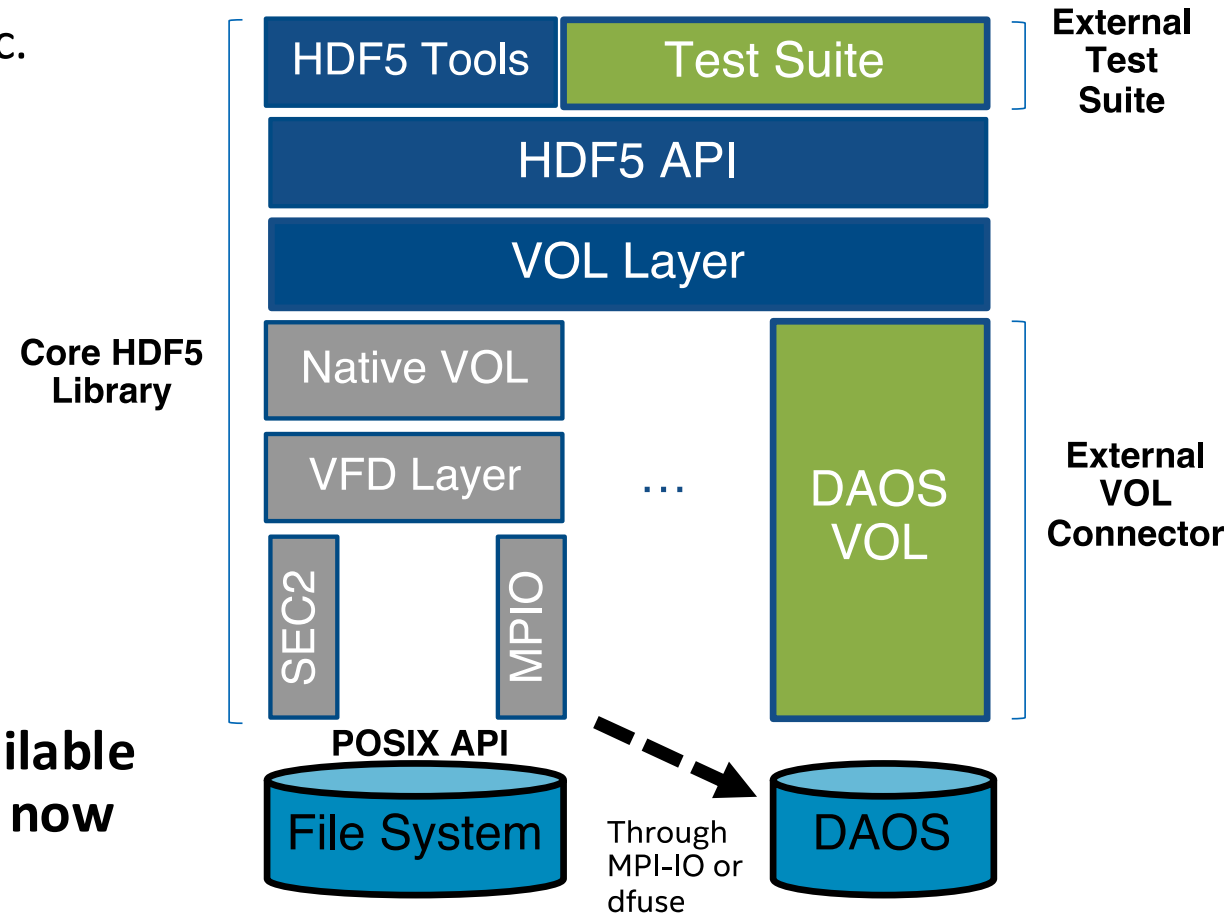
```
romio_cb_write enable
romio_cb_read enable
cb_buffer_size 16777216 cb_config_list *:8
striping_unit 2097152
export ROMIO_HINTS=/path/to/file
```

HDF5

HDF5

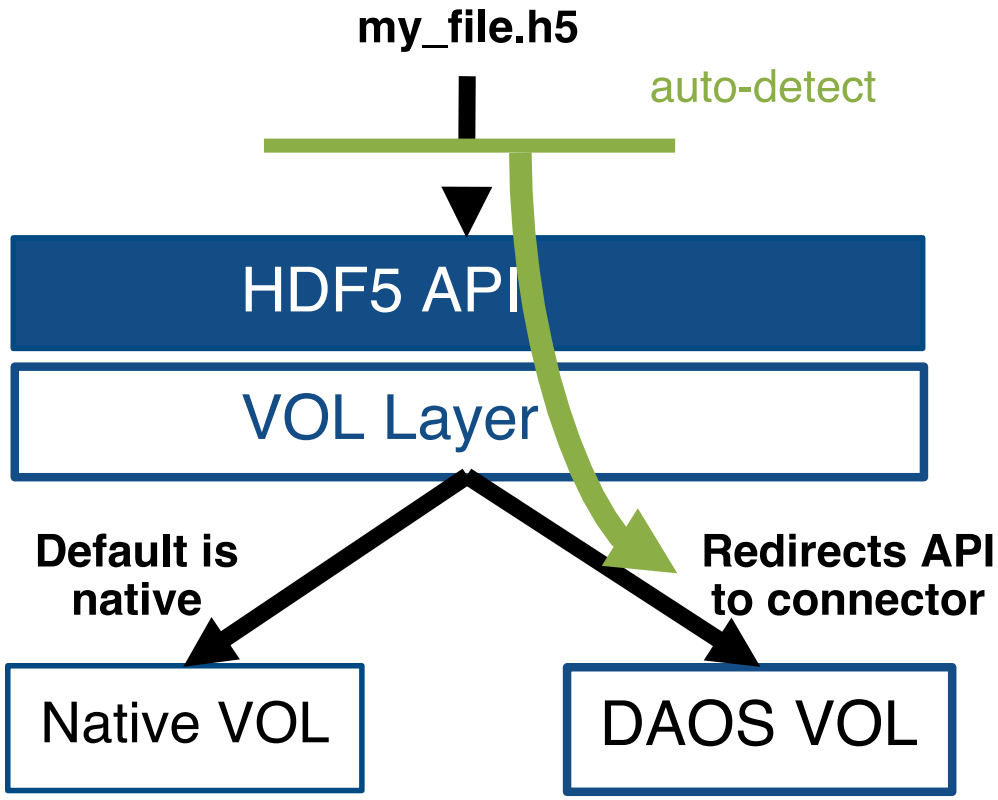
- Structured Data organization
  - Hierarchical groups, datasets, attributes, etc.
  - Good for scientific data
- Portable, self-describing
- Widely adopted
  - Most widely used at ANL?
- Supports Parallel IO
  - MPI based parallelism (MPIIO backend)
  - Allows collective operation
- Useful tools:
  - h5dump, h5ls, h5diff, h5copy, etc.
- **Using HDF5 through MPIIO is what is available on aurora and what is recommended for now**
  - **Same advice applies as for previous slides.**

- HDF5 Virtual Object Layer Framework:



HDF5 VOL Connector Selection

- Two ways to tell which connector to use
  - HDF5 file access property list:
    - `H5Pset_fapl_daos()`
  - Environment variable:
    - `HDF5_VOL_CONNECTOR=daos`
    - `HDF5_PLUGIN_PATH=/path/to/connector/folder`
- Path to file needs to be in a dfuse (POSIX mounted container)
  - HDF5 DAOS VOL creates a new container for every file and links it in the dfuse namespace
- HDF5 tools can be used normally on that file/container
  - Environment variables above need to be set



Hewlett Packard
Enterprise

DAOS Tuning and Best Practices

Mohamad Chaarawi, Distinguished Technologist, HPE

DAOS Containers

- What is a container?
  - Entire dataset, collection of objects
  - POSIX Namespace **(File/Directory != Container)**
    - A POSIX container can be mounted by different users individually
  - Unit of snapshot
- Access permissions:
  - Controlled with ACLs (same as pools)
    - https://docs.daos.io/latest/user/container/#access-control-lists
- Container types:
  - POSIX : 99% of container types today on Aurora
  - Python: PyDAOS object store for dictionaries
  - Pytorch: discussed later
  - HDF5: DAOS VOL connector – not utilized much so far
- Best Practices with Containers:
  - Have few containers as possible in your pool (10s, not 1000s)

Container Properties (Redundancy)

- What is a fault domain?
  - Unit of failure: ssd, engine, server, rack, etc.
  - On aurora the FD is set to the daos server node (2 engines, 32 targets/SSDs).
- Properties are set on container creation (most properties are immutable):

```
daos cont create --properties=rd_fac:2
```

- Redundancy factor property describes the number of **concurrent** fault domains (servers) that containers are protected against.
- Redundancy factor on Aurora today defaults to 0 (will be changed to 2 soon):
  - Mostly used to measure system performance, but not for production workloads
  - Data in container is non redundant; if a DAOS engines goes down, data loss / corruption will be observed by the user
  - **set the rd_fac to 2 as shown above**
- Why 2, not 1?
  - Aurora DAOS systems are set to halt after more than 2 concurrent fault domains failures.
  - So 1 does not guarantee full redundancy if you loose 2 fault domains concurrently.

Container Properties (Checksums)

- DAOS provide end-to-end data integrity feature to ensure data is intact from application to storage.
- Disabled by default on containers (should be enabled by default soon)
- On Aurora, we have seen data corruptions over the network (with memhooks cache monitor)
  - We are using kdreg2 in our case now, but
  - Cost of enabling, performance wise, is minimal; so, no reason not to enabled it.

```
daos cont create --properties=srv_cksum:on
```

- Checksum types supported: crc16/32/64, sha1/256/512,
- This ensures no silent data corruptions can occur.

DAOS Objects

- DAOS provides a multi level Key value/array object API.
- POSIX Files and Directories are built on top of the low-level API.
- Objects can be created with different redundancy types and levels:
  - No redundancy (non-production)
  - Replication (good for metadata / small objects)
  - Erasure Coding (good for files / large objects)
- If not set by the user, DAOS automatically selects an object class (oclass) for files and directories using replication for directories and EC for files:

	RF0	RF1	RF2
File	SX	EC_16P1GX	EC_16P2GX
Dir	S1	RP_2G1	RP_3G1

- The default object class for files and directories can be modified on container creation with the `--file-oclass` and `--dir-oclass` settings, respectively.
  - These default object classes are immutable. If desired, objects inside the container can be created with an object class other than the container default.
  - **The default setting for object class on the container works well when files are large (>= GBs) and directories are relatively lean (< 10k entries).**

Object Classes (Redundancy)

- Object class controls redundancy and striping/sharding of the object (file, directory).
- Redundancy considerations:
- Erasure Code:
  - Best for large IO access patterns.
  - Full stripe write: 12%-33% performance hit.
  - Partial stripe write: 66% performance hit.
  - Read performance should be the same.
  - Not supported for directory objects
- Replication:
  - Best for metadata objects (directories) and small files (<= 16k).
  - Write IOPS: slower by the number of replicas created
  - Read IOPS: equal or better – more shards to serve concurrent requests

Object Classes (Sharding)

- Widely striped objects (SX, RP_GX, or EC_GX):
  - Best IO bandwidth (read/write) for single shared file, highest IOPS for entry creation in single shared directory.
  - Inefficient for File Per Process access especially with thousands of clients and clients having multiple threads and processes.
  - Slow file stat (file size), ls (directory listing)
    - Because all targets are queried
  - Caching with dfuse helps in this case – limited to single node or read-only.
- Small stripe objects (S1/2/4/16/32, RP_ G1/2/4/16/32, or EC_G1/2/4/16/32):
  - Limits size and bandwidth to the number of targets (1/2/4/16/32).
  - Good for File per process or directory per process.
  - Fast stat, ls, rm; because of the limited groups size in the object class, such operation do not query all the DAOS targets on the storage system.

Object Class Recommendations

- This table shows the recommendation for file object class settings:

	RF0	RF1	RF2
Large files, Single shared access, high BW required	SX	EC_16P1GX	EC_16P2GX
Tiny files, more IOPS required	S1	RP_2G1	RP_3G1
Something in between, and File per process to large files	S32	EC_16P1G32	EC_16P2G32

- Object class for files or directories can be set on file creation time through different ways:
  - DFS API: requires application change to use the DFS API directly.
  - MPIIO: using MPI Info hints: `romio_daos_obj_class`
  - daos tool: `daos fs set-attr --path=/mnt/dfuse/d1 --oclass=EC_16P1G32`
    - In this case, d1 must exist in the daos container mount at the dfuse mountpoint, and anything created under d1 will now have object class of `EC_16P1G32`.

Erasure Code Settings

- EC property settings on the container can be tricky to set right to get the best performance.
- Main parameters to consider:
  - File pbject class
  - ec_cell_size
  - DFS chunk size
  - Application IO size
- Ideally most of your writes should be "full stripe":
  - IO size be a multiple of DFS chunk size
  - DFS chunk size be a multiple (at least 1x) of ec_cell_size x EC data blocks (16 in EC_16P2)
- Recommendation on Aurora is to set:
  - File object class = EC_16P2GX (default)
  - **ec_cell_size = 128KiB** (default is 64KiB which proved to be a but inefficient in some cases performance wise)
  - **DFS Chunk size = at least 2MiB** (default is 1MiB, but newer version of DAOS will set this automatically to be ideal depending on the ec_cell_size and default object class)
  - Application IO size to be a multiple of DFS chunk size (2, 4, 8, 16, 32, etc. MiB).
- ***This does not mean other IO patterns are not supported but will not perform as well.***

So, How do I Create a Container?

```
> daos cont create mypool mycont --type=POSIX --dir-oclass=RP_3G1 --file-oclass=EC_16P2GX --
properties=rd_fac:2,cksum:crc32,srv_cksum:on,ec_cell_sz:128kib --chunk-size=4MiB
```

- Red items are non-default as of today but will be default in the future.

- How do I change the object class for one dir once the container is created and mounted?

```
> mkdir /mnt/dfuse/dir_with_tiny_files
> daos fs set-attr --path=/mnt/dfuse/dir_with_tiny_files --oclass=RP_3G1
```

- If it is not possible for the application to write files in separate directories, the advice is to give priority to the large file settings which will be inefficient to small files. So, there will be some performance hit in that case.

MPIIO Settings

- MPICH includes a DAOS ROMIO driver to avoid going through the kernel
  - dfuse is still required to be mounted to access files through a path and query the pool and container information from that path.
- MPIIO uses a POSIX container, so the same advice applies in this case.
- Other libraries build on top of MPICH (HDF5, PnetCDF, ADIOS, etc.)
  - Using those libraries also follow the same advise.
- MPIIO recommendations:
  - The DAOS adio driver is optimized for Single Shared File access (i.e. when the file is not opened with MPI_COMM_SELF).
    - Collective file open shares the underlying DAOS pool and container handles (expensive operations).
    - As long the file is open collectively, IO access whether independent or collective is OK.
    - Avoid File Per Process access with the driver. If not possible, set it to use the UFS driver instead: *export ROMIO_FSTYPE_FORCE=ufs* and use the Interception library.
  - Enable ROMIO collective buffering when doing collective IO especially when using access patterns that are extremely non-contiguous in the file (thousands of small bytes offsets).

DAOS Troubleshooting

DAOS[8644/8644/0] mgmt ERR src/mgmt/cli_mgmt.c:318 get_attach_info() failed to connect to /var/run/daos_agent/daos_agent.sock DER_NONEXIST(-1005): 'The specified entity does not exist'

- Did you: `module use /soft/modulefiles/; ml daos_perf;` ?
- Sometimes environment does not carry over / inherited in scripts, etc.

object ERR src/object/cli_shard.c:791 dc_rw_cb() 937047578338908502.1857.2323.1 (non-EC) RPC 0 to 53/5, flags 0/100000, task 0x86231c0 failed, DMA: DER_TIMEDOUT(-1011): 'Time out'

object ERR src/object/cli_shard.c:854 dc_rw_cb() ... 0xe23fdf0 opc 0 to rank 10 tag 3: DER_HG(-1020): 'Transport layer mercury error'

- Network transfer errors (busy server/target, IO is retried automatically by the client)
- Artifact of user space IO and error logging to stderr

mercury->mem [error] /builddir/build/BUILD/mercury-2.4.0/src/na/na_ofi.c:8870 na_ofi_mem_register() fi_mr_enable() failed, rc: -28 (No space left on device), mr_reg_count: 15711

- IO buffers from memory is too segmented in a single IO. Hit a network limitation (HW limit) of number of Memory Registrations. Usually seen with a some HDF5/PnetCDF apps.
- Resolve: enable collective buffering in MPIIO
- Resolve: Next version of DAOS will combine small memory buffers into a contiguous buffer

Hewlett Packard
Enterprise

Advanced DAOS APIs

Mohamad Chaarawi, Distinguished Technologist, HPE
Johann Lombardi, Senior Distinguished Technologist, HPE

PyDAOS

PyDAOS Overview

- Special type of container implementing DAOS Dictionary (DDict)
  - Multiple DDicts can be created in a PyDAOS container
  - A DDict is identified by a name/string
- DAOS Dictionary (DDict) mimicking the python dict interface
  - Built over DAOS key-value stores
  - Support all the methods of the regular python dictionary class
  - But:
    - are persistent
    - can be bigger than the amount of DRAM available on the compute node
    - support parallel and bulk insertion from difference compute nodes
- https://docs.daos.io/latest/user/python/#daos-dictionaries

Create a PyDAOS Container and a Dictionary

```
$ daos cont create HPE_test mypydaoscont --type PYTHON
Successfully created container b3f71dd3-1884-40ae-a2c5-fd1229a5cb4b
  Container UUID : b3f71dd3-1884-40ae-a2c5-fd1229a5cb4b
  Container Label: mypydaoscont
  Container Type : PYTHON

$ python3
Python 3.6.15 (default, Sep 23 2021, 15:41:43) [GCC] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pydaos
>>> dcont = pydaos.DCont("HPE_test", "mypydaoscont")
>>> print (dcont)
HPE_test/mypydaoscont
>>> dd = dcont.dict("stadium", {"Barcelona" : "Camp Nou", "London" : "Wembley"})
>>> print(dd)
stadium
```

Adding Entries to a DDict

```
>>> dd["Milano"] = "San Siro"
>>> dd["Rio"] = "Maracanã"
>>> print(dd["Milano"])
b'San Siro'
>>> print(len(dd))
4
>>> dd.bput({"Madrid" : "Santiago-Bernabéu", "Manchester" : "Old Trafford"})
>>> print(len(dd))
6
>>> for key in dd: print(key, dd[key])
...
Milano b'San Siro'
London b'Wembley'
Madrid b'Santiago-Bernabéu'
Rio b'Maracanã'
Barcelona b'Camp Nou'
Manchester b'Old Trafford’
>>> "Rio" in dd
True
>>> "Paris" in dd
False
```

Exporting a DDict to a regular Python Dictionary

```
>>> d = dd.dump()
>>> d == dd
True
```

PyTorch Integration

PyTorch DAOS Modules

- Development effort from Enakta Labs and Google
- Self-contained modules integrated with PyTorch
  - pydaos.torch modules
  - Link directly with libdfs (no need to dfuse or pil4dfs)
  - Highly parallel and use multiple network contexts / event queues
- Forthcoming optimizations
  - Use multiple client NICs
  - p2p transfer between GPU HBM and network
- https://docs.daos.io/v2.7/user/pytorch/

PyTorch DAOS Data Loader Module

	Time to scan 1.1M Files
Regular scan	**291s**
Optimized scan	**32s**

- Support for both:
  - Map-style dataset
    - –torch.utils.data.Dataset
    - –implement `__len__/__getitem__/__getitems__()`
    - –requires files scanning which is done in parallel thanks to dfs parallel dir scanning capability
  - Iterable datasets
    - –torch.utils.data.IterableDataset
    - – `__iter__()` over samples
- PyTorch multi-processing supported
  - Provide `worker_init()` method
  - Share pool/container handle across all the workers

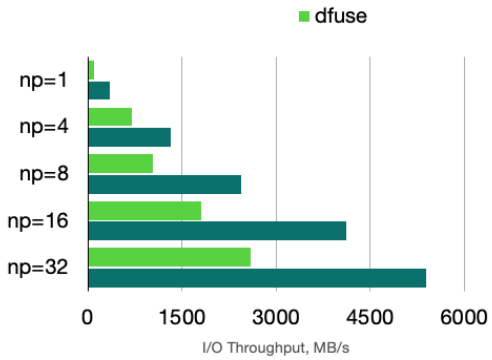
```
import torch
from torch.utils.data import DataLoader
from pydaos.torch import Dataset

dataset = Dataset(pool='pool', container='container', path='/training/samples')
# That's it, when the Dataset is created, it will connect to DAOS, scan the namaspace of the container
# and will be ready to load data from it.

for i, sample in enumerate(dataset):
    print(f"Sample {i} size: {len(sample)}")
```

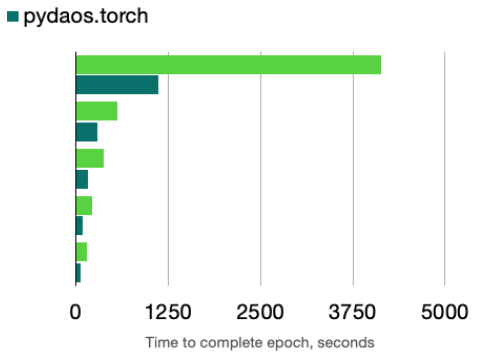
```
dataloader = DataLoader(dataset,
                        batch_size=4,
                        shuffle=True,
                        num_workers=4,
                        worker_init_fn=dataset.worker_init)

# and use DataLoader as usual
for batch in dataloader:
    print(f"Batch size: {len(batch)}")
```

PyTorch DAOS Data Loader Module

Test results

Large IO: 100,000 of 4MB samples, batch size=32, 1 reader

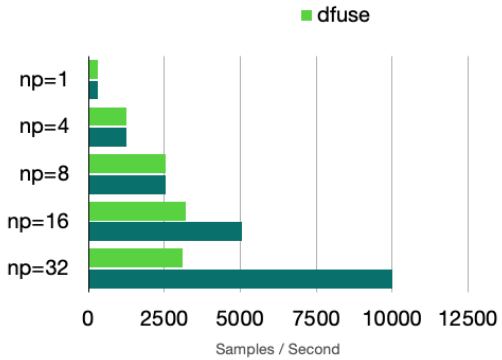


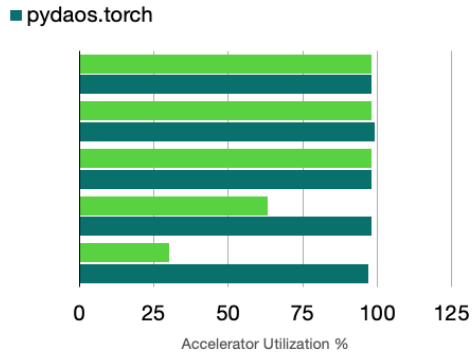


ENAKTA LABS

Test results

Small IO: 1,000,000 of 5000 bytes samples, batch size=32, 4 readers





ENAKTA LABS

source: DUG'24 / PyTorch Update / Denis Barkhtanov, Enakta

PyTorch DAOS Checkpoint Module

- Support torch.save and torch.load function
  - checkpoints directly to/from DAOS container
  - could be same or different container than the one used for data loading
- pydoas.torch.Checkpoint class
  - manages the DAOS connections
  - provides reader and writer methods.

```
import torch
from pydaos.torch import Checkpoint

# ...

chkp = Checkpoint(pool, cont, prefix='/training/checkpoints')

with chkp.writer('model.pt') as w:
    torch.save(model.state_dict(), w)

# Later, to load the model

with chkp.reader('model.pt') as r:
    torch.load(r)
```

```
import logging
import torch
from pydaos.torch import Checkpoint as DaosCheckpoint

from dlio_benchmark.checkpointing.base_checkpointing import BaseCheckpointing
from dlio_benchmark.utils.utility import Profile
from dlio_benchmark.utils.config import ConfigArguments

from dlio_benchmark.common.constants import MODULE_CHECKPOINT

dlp = Profile(MODULE_CHECKPOINT)


class PyDaosTorchCheckpointing(BaseCheckpointing):
    __instance = None

    @staticmethod
    def get_instance():
        """ Static access method. """
        if PyDaosTorchCheckpointing.__instance is None:
            logging.basicConfig(level=logging.INFO)
            PyDaosTorchCheckpointing.__instance = PyDaosTorchCheckpointing()
        return PyDaosTorchCheckpointing.__instance

    @dlp.log_init
    def __init__(self):
        super().__init__("pt")

        args = ConfigArguments.get_instance()
        prefix = args.checkpoint_folder
        pool = args.checkpoint_daos_pool
        cont = args.checkpoint_daos_cont

        logging.info(f"Checkpointing is set to DAOS pool: {pool}, container: {cont} with prefix: {prefix}")
        self.ckpt = DaosCheckpoint(pool, cont, prefix)

    @dlp.log
    def get_tensor(self, size):
        return torch.randint(high=1, size=(size,), dtype=torch.int8)

    @dlp.log
    def save_state(self, suffix, state):
        name = self.get_name(suffix)
        with self.ckpt.writer(name) as f:
            torch.save(state, f)

    @dlp.log
    def checkpoint(self, epoch, step_number):
        super().checkpoint(epoch, step_number)

    @dlp.log
    def finalize(self):
        super().finalize()
```

Hewlett Packard
Enterprise

Monitoring with Darshan

Kaushik Velusamy

Monitoring with DAOS interception library report

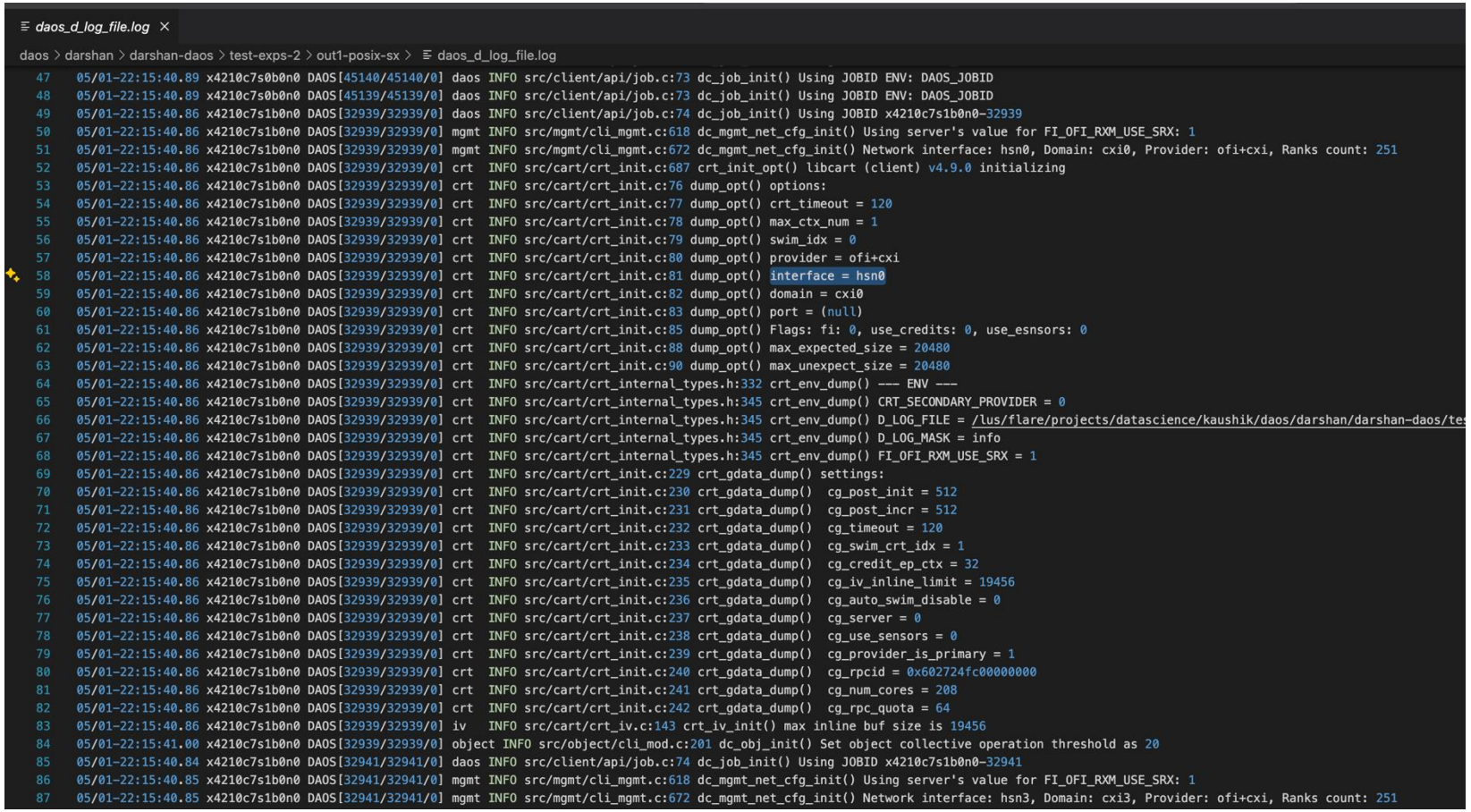
export D_IL_REPORT=1

mpiexec --env LD_PRELOAD=/usr/lib64/libpil4dfs.so
-np ${NRANKS}..

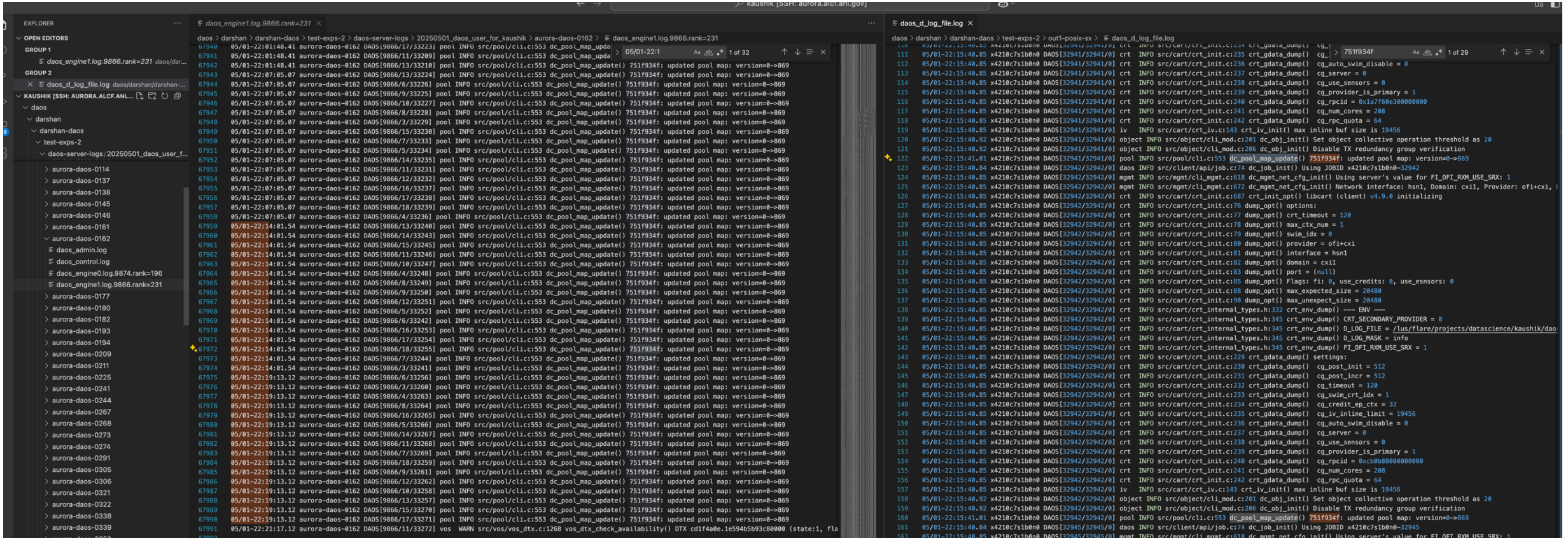
```
Options:
api                 : POSIX
apiVersion          :
test filename       : /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
access              : single-shared-file
type                : independent
segments            : 1
ordering in a file  : sequential
ordering inter file : constant task offset
task offset         : 1
nodes               : 2
tasks               : 24
clients per node    : 12
memoryBuffer        : CPU
dataAccess          : CPU
GPUDirect           : 0
repetitions         : 1
xfersize            : 1 MiB
blocksize           : 10 MiB
aggregate filesize  : 240 MiB
verbose             : 1
stonewallingTime    : 100
stoneWallingWearOut : 0

libpil4dfs intercepting summary for ops on DFS:
[read   ]  10
[write  ]  10

[open   ]  2
[stat   ]  2
[opendir]  0
[readdir]  0
[link   ]  0
[unlink ]  0
[rdlink ]  0
[seek   ]  20
[mkdir  ]  0
[rmdir  ]  0
[rename ]  0
[mmap   ]  0

[op_sum ]  44
libpil4dfs intercepting summary for ops on DFS:
[read   ]  10
[write  ]  10
```

Monitoring with DAOS DEBUG LOGs Client

```
export D_LOG_FILE="${PBS_O_WORKDIR}/${PBS_JOBID}-${NNODES}/daos_d_log_file.log"
export D_LOG_MASK=info
export D_LOG_FILE_APPEND_PID=1
export D_LOG_STDERR_IN_LOG=1
```

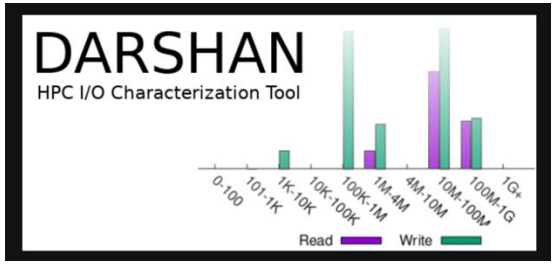


Monitoring with DAOS DEBUG LOGs Server



** Server logs can be collected on need basis.*

Monitoring with Darshan instrumentation



- Darshan, a scalable HPC I/O characterization tool.
- Designed to capture an accurate picture of application I/O behavior.
- Properties such as patterns of access within files.
- Investigate and tune the I/O behavior of complex HPC applications.
- Minimum overhead lightweight design
- Can help uncover critical insights into applications' usage of this novel storage technology.

A shared library your application preloads at runtime, which generates a binary file at program termination, and a suite of utilities to analyze this file.

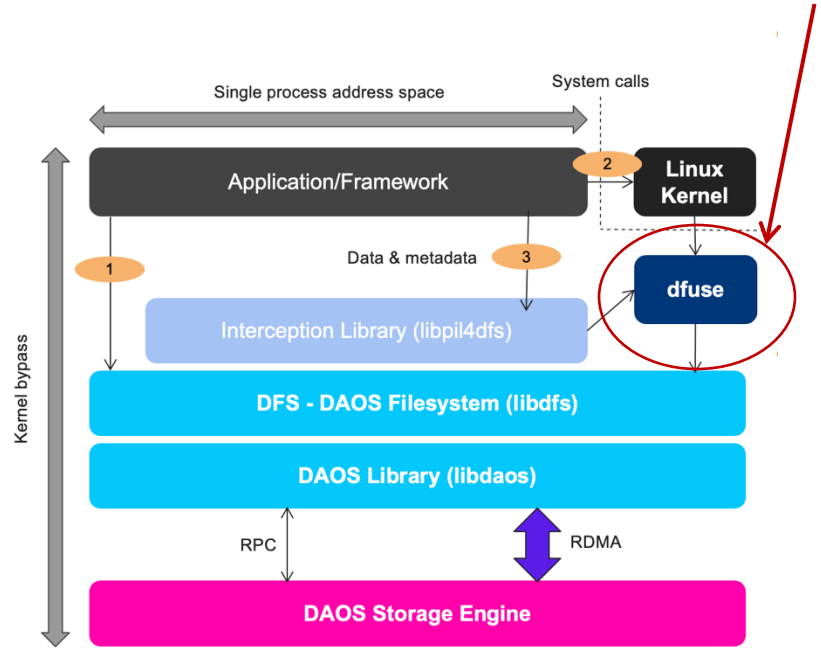
https://github.com/darshan-hpc/darshan

S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood and N. J. Wright, "Modular HPC I/O Characterization with Darshan," 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT), Salt Lake City, UT, USA, 2016, pp. 9-17, doi: 10.1109/ESPT.2016.006

Darshan + Legacy Posix access Via Fuse

- Users have multiple options for accessing DAOS storage, each with varying performance, ease of adoption, etc.

➢ No app modifications required
➢ Performance limited by FUSE architecture, single process, etc.
➢ Darshan support:
  - Instrumentation via Darshan's POSIX module
  - No insight into DAOS usage by FUSE daemon



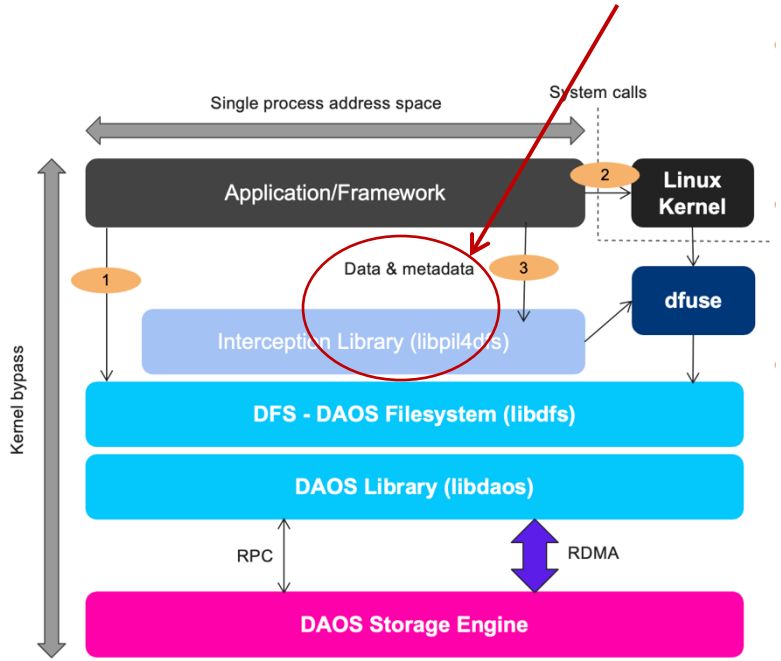
Darshan + Legacy POSIX access via FUSE + interception libs

➢ No app modifications required

➢ Performance improved by bypassing FUSE for some or all I/O operations

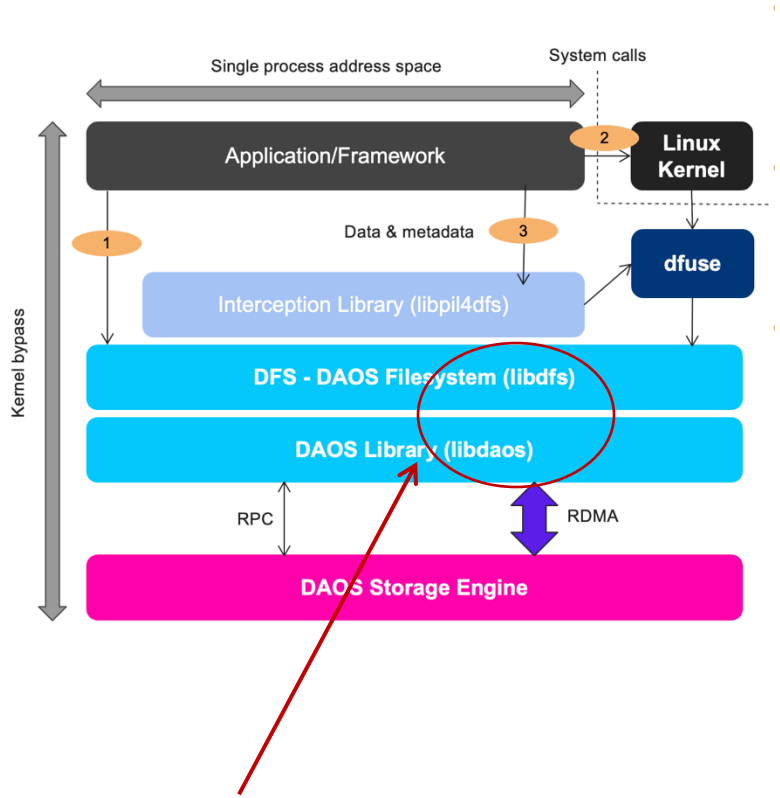
➢ Darshan support:

■ POSIX, DFS, and DAOS level instrumentation



Darshan + Direct access to DFS (file) and DAOS (object) APIs

- App or middleware modifications required
- Direct usage of DAOS APIs provides most control over performance, redundancy, etc.
- Darshan support:
  - Full instrumentation of DFS and DAOS APIs



To install your own setup of darshan profiler – on Aurora

```
module use /soft/modulefiles
module load daos
git clone https://github.com/darshan-hpc/darshan.git
./prepare.sh

./configure --enable-daos-mod \
            --enable-hdf5-mod --with-hdf5= /opt/aurora/24.347.0/spack/unified/0.9.0/install/linux-sles15-x86_64/oneapi-2025.0.5/hdf5-1.14.5-drswwe2
            --enable-pnetcdf-mod  --with-pnetcdf=/opt/aurora/24.347.0/spack/unified/0.9.0/install/linux-sles15-x86_64/oneapi-2025.0.5/parallel-
netcdf-1.12.3-jy2mhag \
            --with-log-path=/lus/flare/projects/Aurora_deployment/pkcoff/darshanlog \
            --with-jobid-env=PBS_JOBID \
            --prefix=/lus/flare/projects/Aurora_deployment/pkcoff/lib/darshan \
            'CFLAGS=-O2 -g -Wall' CC=mpicc
make install

/lus/flare/projects/datascience/kaushik/daos/darshan/darshan-daos/install-daos-darshan/bin/darshan-config --log-path

/lus/flare/projects/datascience/kaushik/daos/darshan/darshan-daos/install-daos-darshan/bin/darshan-mk-log.pl

(should only need to do this one time)  that script just sets up that year/month/day hierarchy within the log dir
This is where your final darshan logs would go

Then to save to a specific file location: export DARSHAN_LOGFILE=<fullpathtodarshanlogfile>

To test if darshan is working
• LD_PRELOAD=/path/to/libdarshan.so DARSHAN_ENABLE_NONMPI=1 cat /some/file
• LD_PRELOAD=/path/to/libdarshan.so mpiexec ...
```

To install pydarshan on Macbook
Python utilities to interact with Darshan log records

```
brew update
brew install --cask anaconda # restart terminal

conda create --name env-mac-darshan --clone base
conda  activate env-mac-darshan

git clone https://github.com/daos-stack/daos.git
cd darshan
git submodule update --init
./prepare
cd darshan-util
../configure --disable-darshan-runtime --prefix=/Users/kvelusamy/Desktop/projects/tools/darshan-util-install \
             --enable-apmpi-mod --enable-apxc-mod CFLAGS='-g -O0 -Wall'

make & make install

cd pydarshan
pip install  .

export LD_LIBRARY_PATH=/Users/kvelusamy/Desktop/projects/tools/darshan-util-install/lib/:$LD_LIBRARY_PATH
export PATH=/Users/kvelusamy/Desktop/projects/tools/darshan-util-install/:$PATH
export DYLD_FALLBACK_LIBRARY_PATH=/Users/kvelusamy/Desktop/projects/tools/darshan-util-install/lib/

$ LD_PRELOAD=/Users/kvelusamy/Desktop/projects/tools/darshan-util-install/lib/libdarshan-util.a
  python -m darshan summary ~/Desktop/pyDAOS/kaushikv_ior_id4576025-45130_5-1-80140-6343233643831684606_1.darshan

ls ~/Desktop/pyDAOS/kaushikv_ior_id4576025-45130_5-1-80140-6343233643831684606_1.html
```

Darshan module on aurora and LD_PRELOAD ORDER

```
module use /soft/perftools/darshan/darshan-3.4.7/share/craype-2.x/modulefiles
module load darshan
```

1

```
LD_PRELOAD=/soft/perftools/darshan/darshan-3.4.7/lib/libdarshan.so:
           /opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-
x86_64/oneapi-2025.0.5/hdf5-1.14.5-zrlo32i/lib/libhdf5.so:
           /opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-
x86_64/oneapi-2025.0.5/parallel-netcdf-1.12.3-cszcp66/lib/libpnetcdf.so:
           /usr/lib64/libpil4dfs.so
```

2

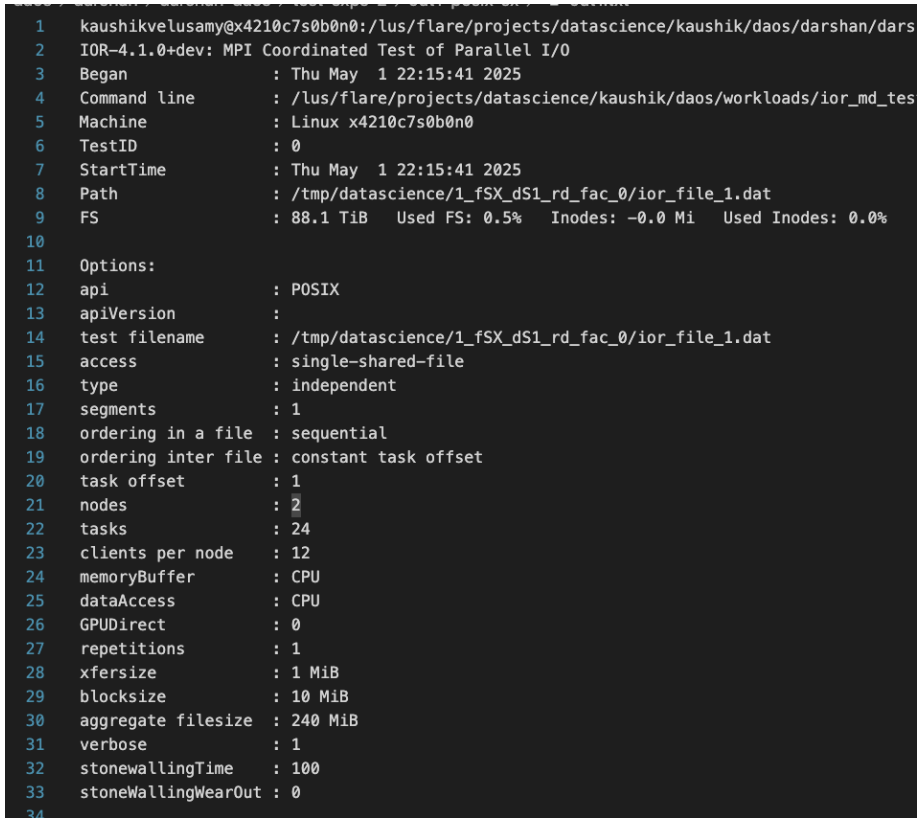
3

4

If your application is using gpu_tile_compact.sh then this whole LD_PRELOAD will go in your personal copy of the bash script via export.

Demo

```
mpiexec  ${EXT_ENV1} –np 24 -ppn 12 --cpu-bind list:4:9:14:19:20:25:30:35:56:61:66:71:72:77:82:87 --no-vni –genvall \
../ior_mdtest_install_bin/bin/ior -a POSIX -i 1 -b 10m -t 1m -D 100 -k -w -r -C -e -v -o /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
```



```
kaushikvelusamy@x4210c7s0b0n0:/lus/flare/projects/datascience/kaushik/daos/darshan/dars
IOR-4.1.0+dev: MPI Coordinated Test of Parallel I/O
Began               : Thu May  1 22:15:41 2025
Command line        : /lus/flare/projects/datascience/kaushik/daos/workloads/ior_md_tes
Machine             : Linux x4210c7s0b0n0
TestID              : 0
StartTime           : Thu May  1 22:15:41 2025
Path                : /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
FS                  : 88.1 TiB   Used FS: 0.5%   Inodes: -0.0 Mi   Used Inodes: 0.0%

Options:
api                 : POSIX
apiVersion          :
test filename       : /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
access              : single-shared-file
type                : independent
segments            : 1
ordering in a file  : sequential
ordering inter file : constant task offset
task offset         : 1
nodes               : 2
tasks               : 24
clients per node    : 12
memoryBuffer        : CPU
dataAccess          : CPU
GPUDirect           : 0
repetitions         : 1
xfersize            : 1 MiB
blocksize           : 10 MiB
aggregate filesize  : 240 MiB
verbose             : 1
stonewallingTime    : 100
stoneWallingWearOut : 0
```

With Darshan-Parser

install-daos-darshan/bin/darshan-parser

kaushikv_ior_id4576025-45130_5-1-80140-6343233643831684606_1.darshan

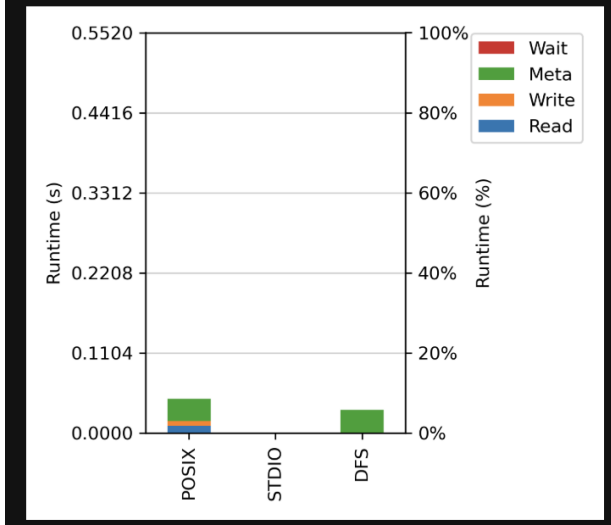
```
# ****************************************************
# DFS module data
# ****************************************************

# description of DFS counters:
#   DFS_*: DFS operation counts.
#   OPENS,GLOBAL_OPENS,LOOKUPS,DUPS,READS,READXS,NB_READS,WRITES,WRITEXS,NB_WRITES,GET_SIZES,PUNCHES,REMOVES,STATS are types of operations.
#   DFS_BYTES_*: total bytes read and written.
#   DFS_RW_SWITCHES: number of times access alternated between read and write.
#   DFS_MAX_*_TIME_SIZE: size of the slowest read and write operations.
#   DFS_SIZE_*_*: histogram of read and write access sizes.
#   DFS_ACCESS*_ACCESS: the four most common access sizes.
#   DFS_ACCESS*_COUNT: count of the four most common access sizes.
#   DFS_CHUNK_SIZE: DFS file chunk size.
#   DFS_*_RANK: rank of the processes that were the fastest and slowest at I/O (for shared files).
#   DFS_*_RANK_BYTES: bytes transferred by the fastest and slowest ranks (for shared files).
#   DFS_F_*_START_TIMESTAMP: timestamp of first open/read/write/close.
#   DFS_F_*_END_TIMESTAMP: timestamp of last open/read/write/close.
#   DFS_F_READ/WRITE/META_TIME: cumulative time spent in read, write, or metadata operations.
#   DFS_F_MAX_*_TIME: duration of the slowest read and write operations.
#   DFS_F_*_RANK_TIME: fastest and slowest I/O time for a single rank (for shared files).

#<module>   <rank>  <record id> <counter>   <value> <file name> <mount pt>  <fs type>
DFS -1  13273173857004836666 DFS_OPENS   24  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_GLOBAL_OPENS    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_LOOKUPS 72  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_DUPS    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_READS   240 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_READXS  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_WRITES  240 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_WRITEXS 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_NB_READS    240 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_NB_WRITES   240 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_GET_SIZES   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_PUNCHES 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_REMOVES 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_STATS   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_BYTES_READ  201351360   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_BYTES_WRITTEN   251658240   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_RW_SWITCHES 24  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_MAX_READ_TIME_SIZE  1032    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_MAX_WRITE_TIME_SIZE 1048576 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_0_100 24  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_100_1K    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_1K_10K    24  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_10K_100K  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_100K_1M   192 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_1M_4M 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_4M_10M    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_10M_100M  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_100M_1G   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_READ_1G_PLUS   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_0_100    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_100_1K   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_1K_10K   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_10K_100K 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_100K_1M  240 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_1M_4M    0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_4M_10M   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_10M_100M 0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_100M_1G  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SIZE_WRITE_1G_PLUS  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS1_ACCESS  1048576 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS2_ACCESS  1032    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS3_ACCESS  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS4_ACCESS  0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS1_COUNT   432 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS2_COUNT   24  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS3_COUNT   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_ACCESS4_COUNT   0   ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_CHUNK_SIZE  1048576 ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_FASTEST_RANK    19  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_FASTEST_RANK_BYTES  18875400    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SLOWEST_RANK    17  ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_SLOWEST_RANK_BYTES  18875400    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_OPEN_START_TIMESTAMP  0.495613    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_READ_START_TIMESTAMP  0.538058    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_WRITE_START_TIMESTAMP 0.499866    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_CLOSE_START_TIMESTAMP 0.506562    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_OPEN_END_TIMESTAMP    0.554160    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_READ_END_TIMESTAMP    0.548369    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_WRITE_END_TIMESTAMP   0.522338    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_CLOSE_END_TIMESTAMP   0.554198    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_READ_TIME 0.005253    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_WRITE_TIME    0.007477    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_META_TIME 0.743890    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_MAX_READ_TIME 0.000057    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_MAX_WRITE_TIME    0.000076    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_FASTEST_RANK_TIME 0.023989    ior_file_1.dat  UNKNOWN N/A
DFS -1  13273173857004836666 DFS_F_SLOWEST_RANK_TIME 0.041173    ior_file_1.dat  UNKNOWN N/A
```

```
mpiexec  ${EXT_ENV1} –np 24 -ppn 12 --cpu-bind list:4:9:14:19:20:25:30:35:56:61:66:71:72:77:82:87 --no-vni –genvall \
../ior_mdtest_install_bin/bin/ior -a POSIX -i 1 -b 10m -t 1m -D 100 -k -w -r -C -e -v -o /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
```

```
kaushikvelusamy@x4210c7s0b0n0:/lus/flare/projects/datascience/kaushik/daos/darshan/dars
IOR-4.1.0+dev: MPI Coordinated Test of Parallel I/O
Began               : Thu May  1 22:15:41 2025
Command line        : /lus/flare/projects/datascience/kaushik/daos/workloads/ior_md_tes
Machine             : Linux x4210c7s0b0n0
TestID              : 0
StartTime           : Thu May  1 22:15:41 2025
Path                : /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
FS                  : 88.1 TiB   Used FS: 0.5%   Inodes: -0.0 Mi   Used Inodes: 0.0%

Options:
api                 : POSIX
apiVersion          :
test filename       : /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
access              : single-shared-file
type                : independent
segments            : 1
ordering in a file  : sequential
ordering inter file : constant task offset
task offset         : 1
nodes               : 2
tasks               : 24
clients per node    : 12
memoryBuffer        : CPU
dataAccess          : CPU
GPUDirect           : 0
repetitions         : 1
xfersize            : 1 MiB
blocksize           : 10 MiB
aggregate filesize  : 240 MiB
verbose             : 1
stonewallingTime    : 100
stoneWallingWearOut : 0
```

Cross-Module Comparisons

I/O Cost

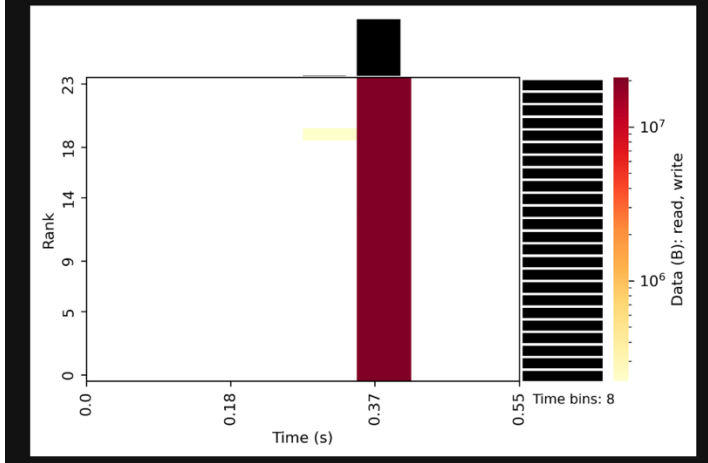


Average (across all ranks) amount of run time that each process spent performing I/O, broken down by access type. See the right edge bar graph on heat maps in preceding section to indicate if I/O activity was balanced across processes. The 'Wait' category is only meaningful for PNETCDF asynchronous I/O operations.

Heat Maps

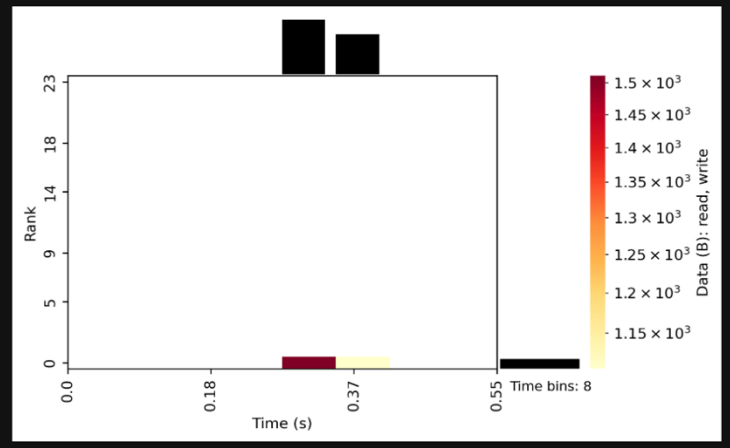
```
mpiexec  ${EXT_ENV1} –np 24 -ppn 12 --cpu-bind list:4:9:14:19:20:25:30:35:56:61:66:71:72:77:82:87 --no-vni –genvall \
../ior_mdtest_install_bin/bin/ior -a POSIX -i 1 -b 10m -t 1m -D 100 -k -w -r -C -e -v -o /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
```

Heat Map: HEATMAP POSIX



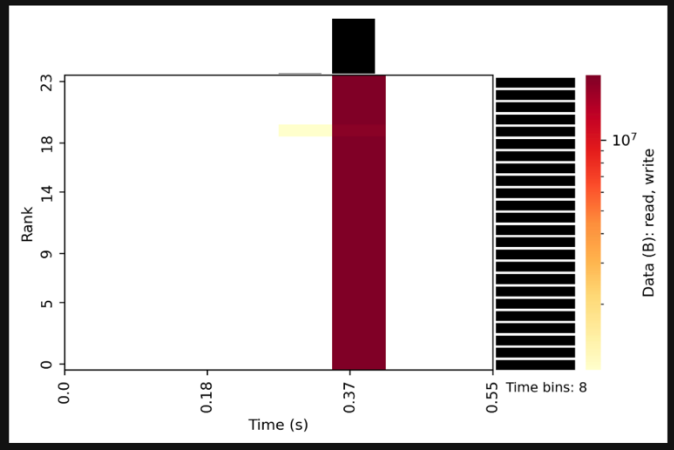
Heat map of I/O (in bytes) over time broken down by MPI rank. Bins are populated based on the number of bytes read/written in the given time interval. The top edge bar graph sums each time slice across ranks to show aggregate I/O volume over time, while the right edge bar graph sums each rank across time slices to show I/O distribution across ranks.

Heat Map: HEATMAP STDIO



Heat map of I/O (in bytes) over time broken down by MPI rank. Bins are populated based on the number of bytes read/written in the given time interval. The top edge bar graph sums each time slice across ranks to show aggregate I/O volume over time, while the right edge bar graph sums each rank across time slices to show I/O distribution across ranks.

Heat Map: HEATMAP DFS



Heat map of I/O (in bytes) over time broken down by MPI rank. Bins are populated based on the number of bytes read/written in the given time interval. The top edge bar graph sums each time slice across ranks to show aggregate I/O volume over time, while the right edge bar graph sums each rank across time slices to show I/O distribution across ranks.

Per Module Statistics

```
mpiexec  ${EXT_ENV1} –np 24 -ppn 12 --cpu-bind list:4:9:14:19:20:25:30:35:56:61:66:71:72:77:82:87 --no-vni –genvall \
../ior_mdtest_install_bin/bin/ior -a POSIX -i 1 -b 10m -t 1m -D 100 -k -w -r -C -e -v -o /tmp/datascience/1_fSX_dS1_rd_fac_0/ior_file_1.dat
```

Per-Module Statistics: POSIX

Overview

files accessed	1
bytes read	240.00 MiB
bytes written	240.00 MiB
I/O performance estimate	8567.13 MiB/s (average)

File Count Summary (estimated by POSIX I/O access offsets)

	number of files	avg. size	max size
total files	1	240.00 MiB	240.00 MiB
read-only files	0	0	0
write-only files	0	0	0
read/write files	1	240.00 MiB	240.00 MiB

Per-Module Statistics: STDIO

Overview

files accessed	1
bytes read	0 Bytes
bytes written	2.56 KiB
I/O performance estimate	39.07 MiB/s (average)

File Count Summary (estimated by STDIO I/O access offsets)

	number of files	avg. size	max size
total files	1	2.56 KiB	2.56 KiB
read-only files	0	0	0
write-only files	1	2.56 KiB	2.56 KiB
read/write files	0	0	0

Per-Module Statistics: DFS

Overview

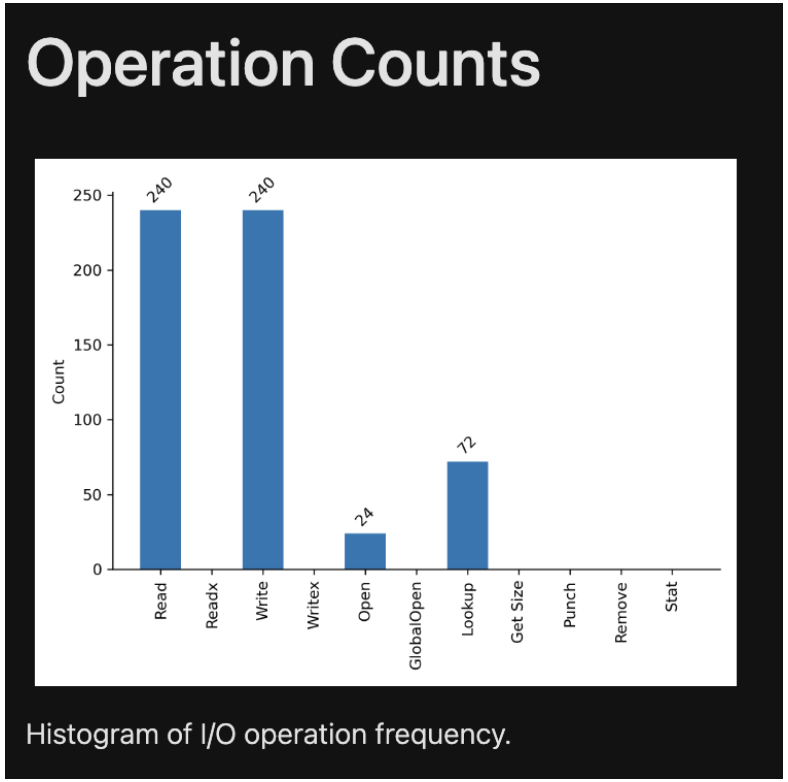
files accessed	1
bytes read	192.02 MiB
bytes written	240.00 MiB
I/O performance estimate	10492.77 MiB/s (average)

File Count Summary (estimated by DFS I/O access offsets)

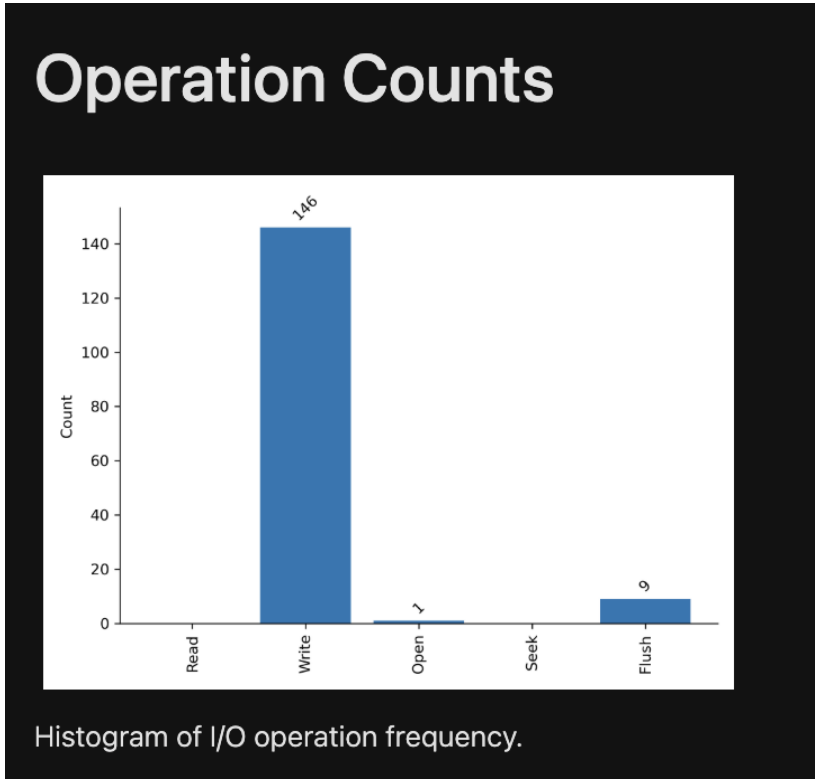
	number of files	avg. size	max size
total files	1	0	0
read-only files	0	0	0
write-only files	0	0	0
read/write files	1	0	0

Operation Counts

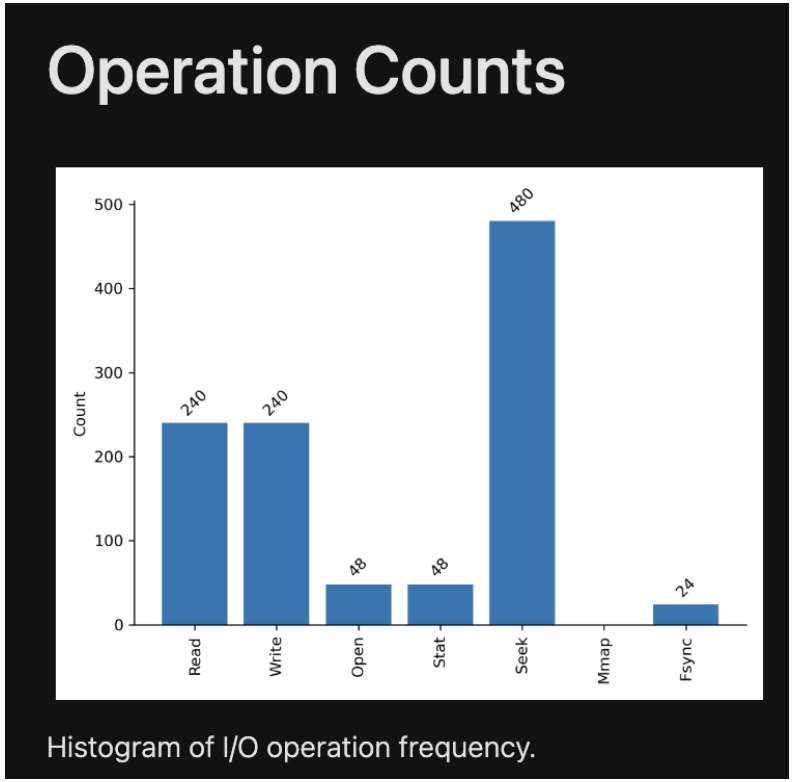
- POSIX



- STDIO



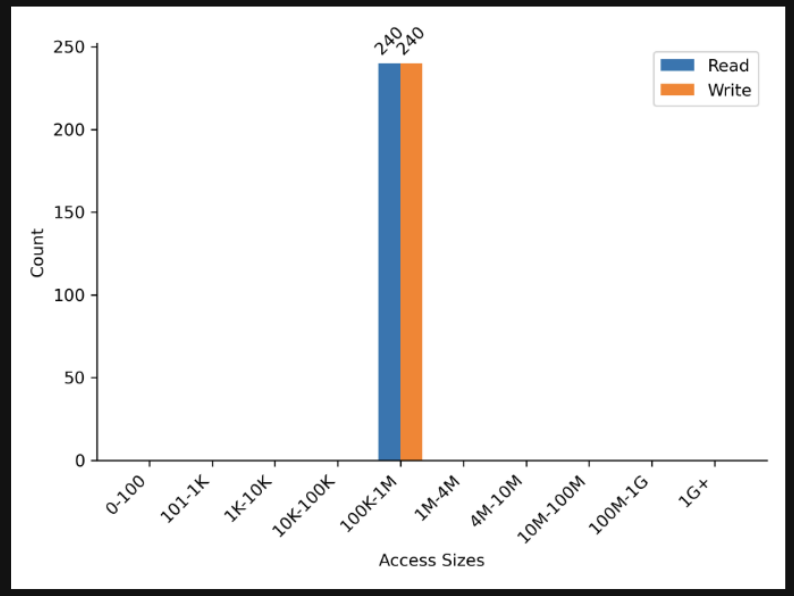
- DFS



Access Sizes

- POSIX

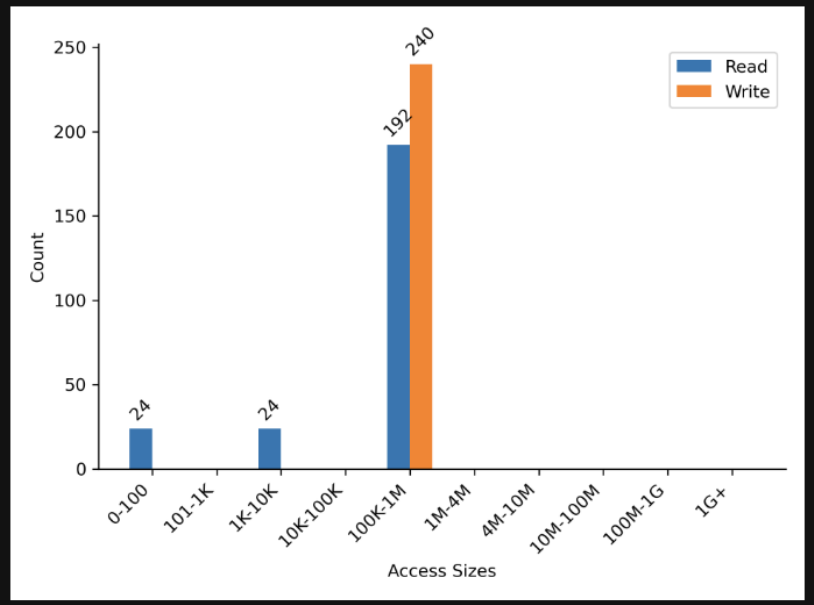
Access Sizes



Histogram of read and write access sizes. The specific values of the most frequently occurring access sizes can be found in the *Common Access Sizes* table.

- DFS

Access Sizes

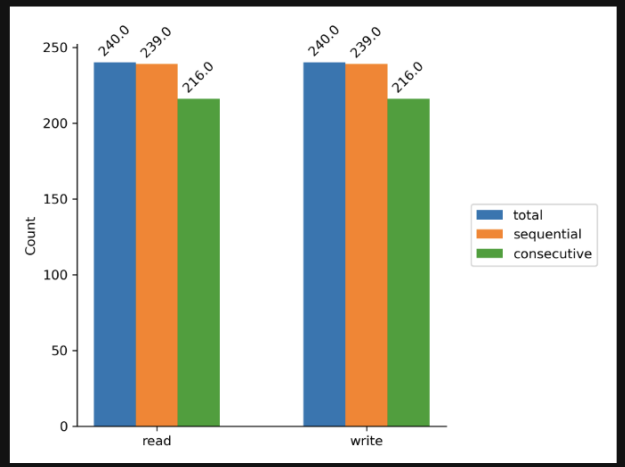


Histogram of read and write access sizes. The specific values of the most frequently occurring access sizes can be found in the *Common Access Sizes* table.

Common Access Sizes

- POSIX

Access Pattern



Sequential (offset greater than previous offset) vs. consecutive (offset immediately following previous offset) file operations. Note that, by definition, the sequential operations are inclusive of consecutive operations.

Common Access Sizes

Access Size	Count
1048576	480

- DFS

Common Access Sizes

Access Size	Count
1048576	432
1032	24

Data Access by Category

Summary of data access volume categorized by storage target (e.g., file system mount point) and sorted by volume.

Hewlett Packard
Enterprise

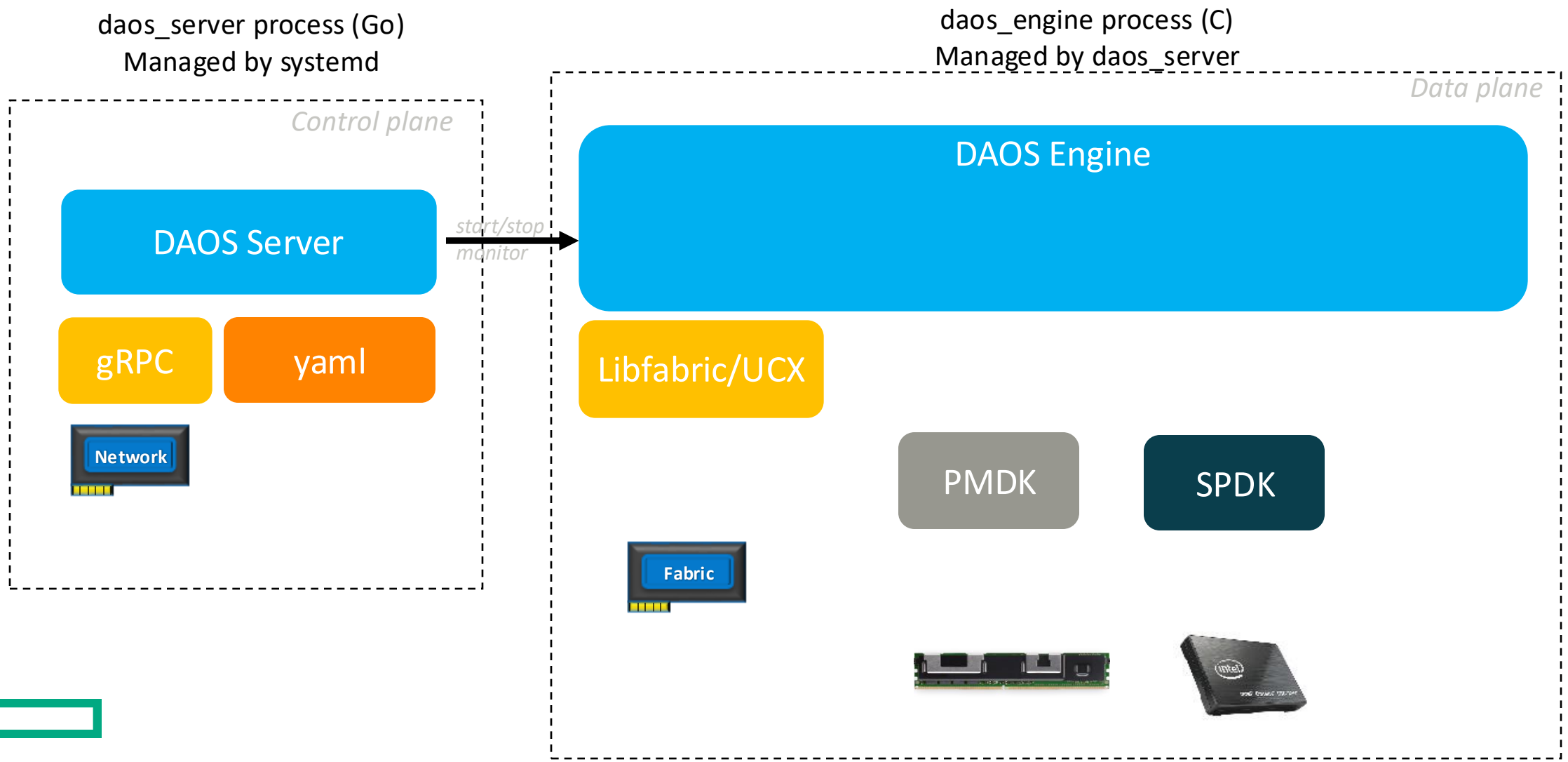
Q&A

Questions/Topics

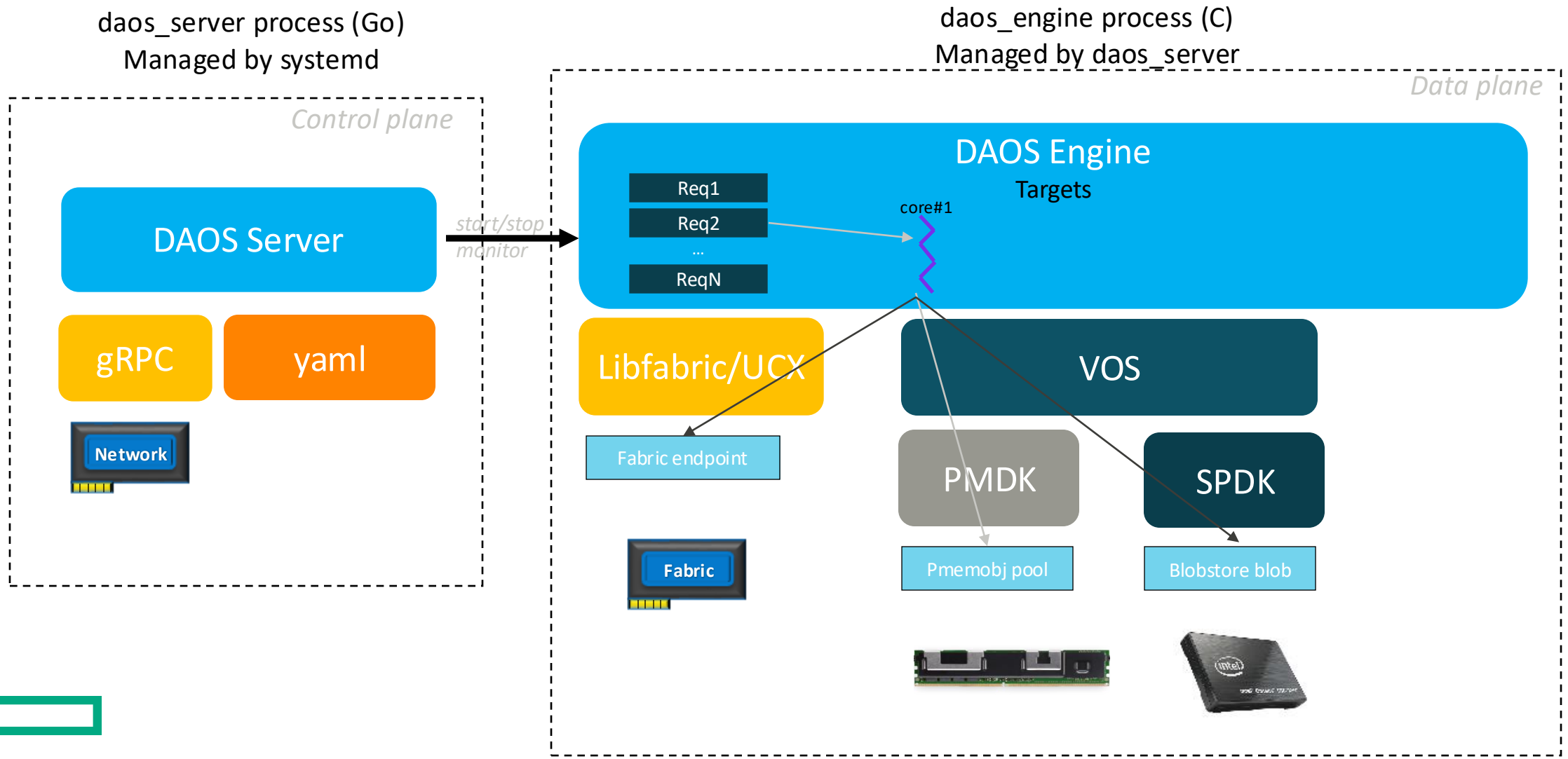
- How to make DAOS easier to use?
  - Mimic the behavior of a parallel filesystem
    - Pre-mount a “root” POSIX container with dfuse on the compute node
    - Rely on multi-user dfuse
  - Simplify access to parallel data mover

Thank You

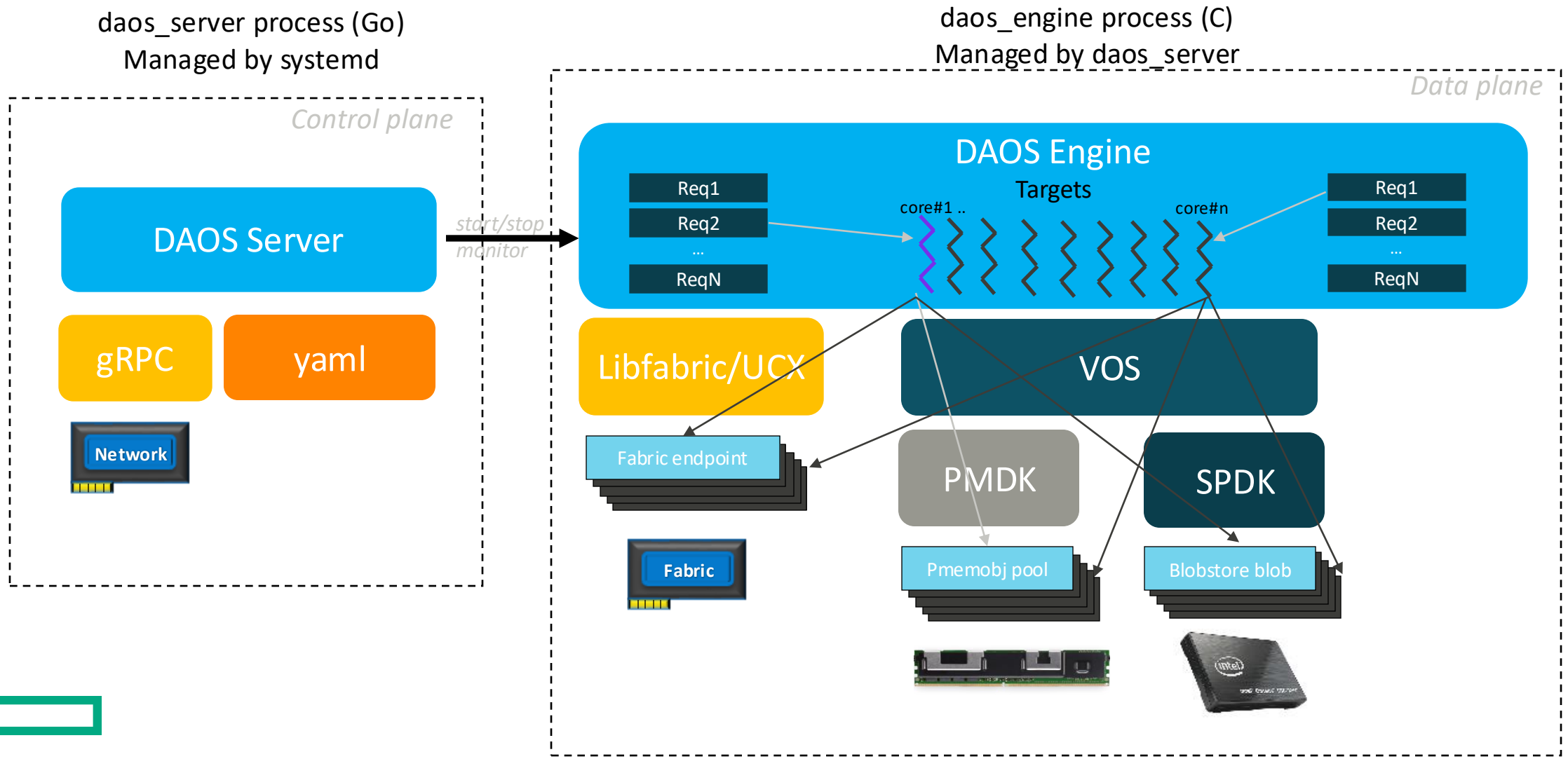
DAOS Service Structure



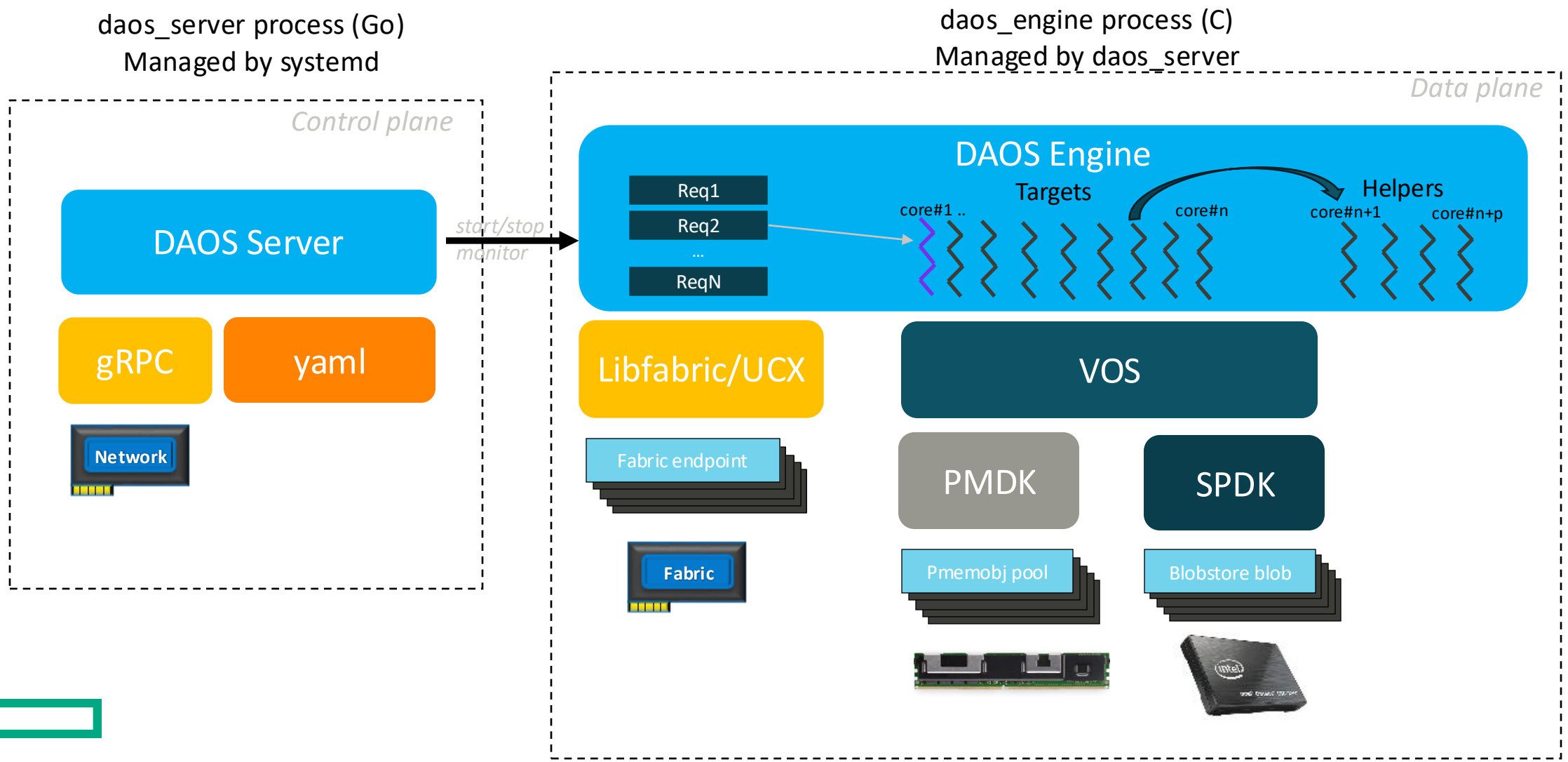
DAOS Service Structure



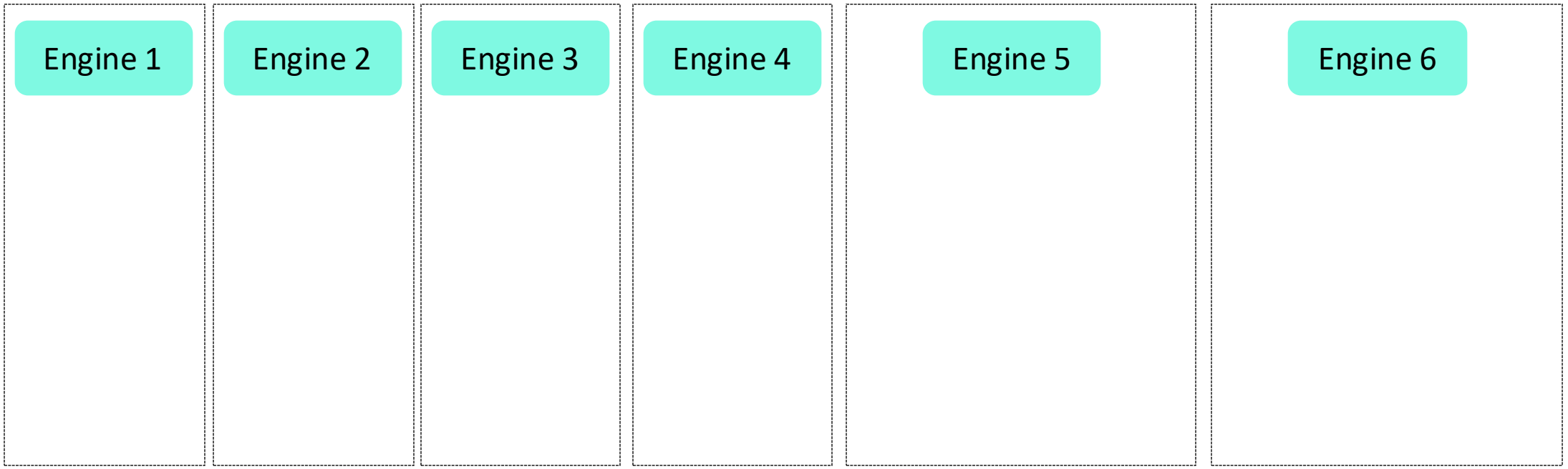
DAOS Service Structure



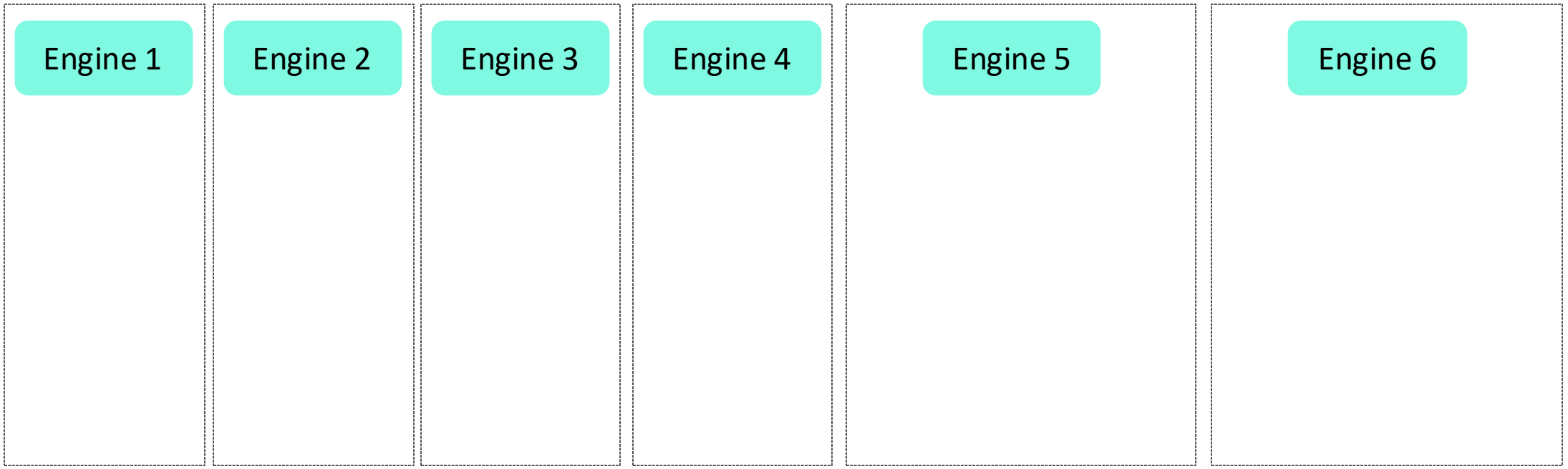
DAOS Service Structure



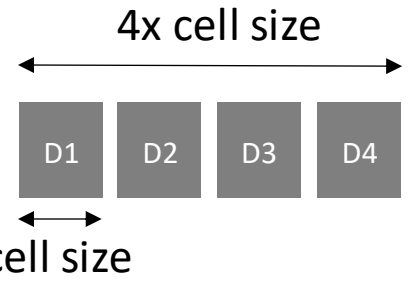
Erasure Code Full Stripe I/O (4+2)



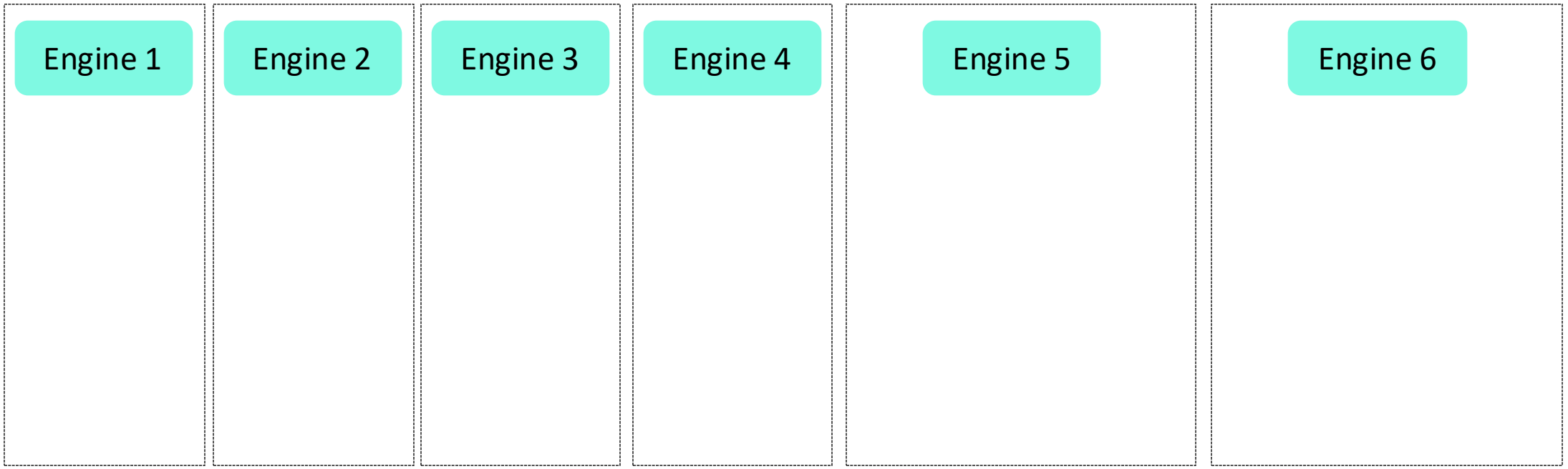
Erasure Code Full Stripe I/O (4+2)



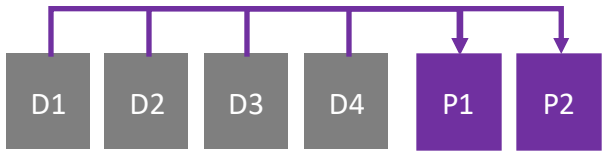
Application Buffer



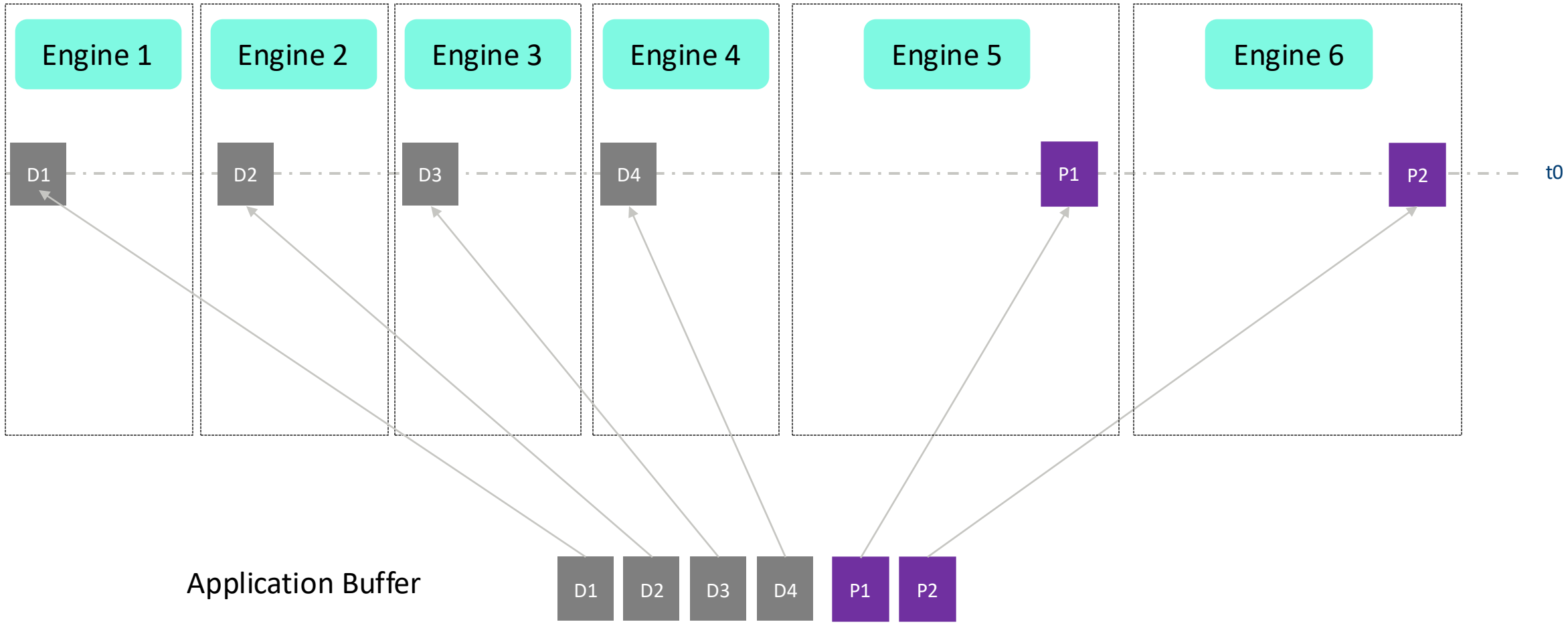
Erasure Code Full Stripe I/O (4+2)



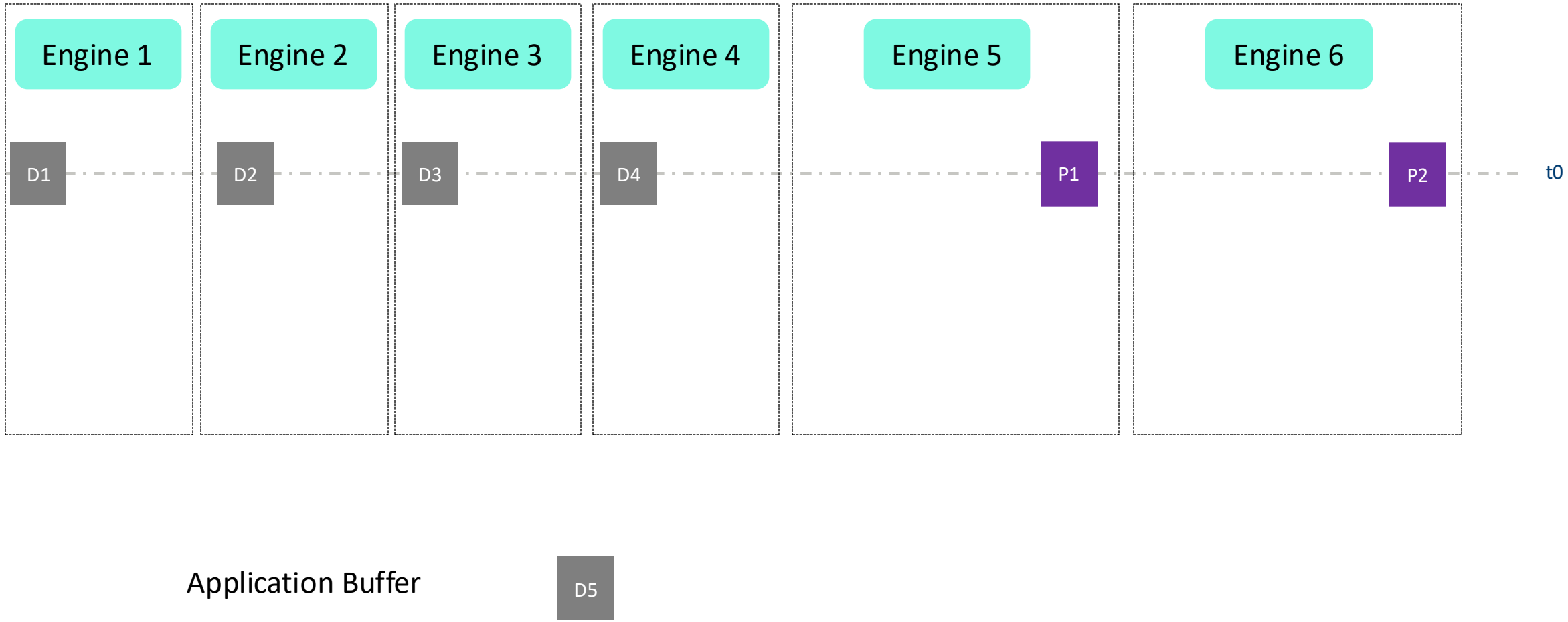
Application Buffer



Erasure Code Full Stripe I/O (4+2)



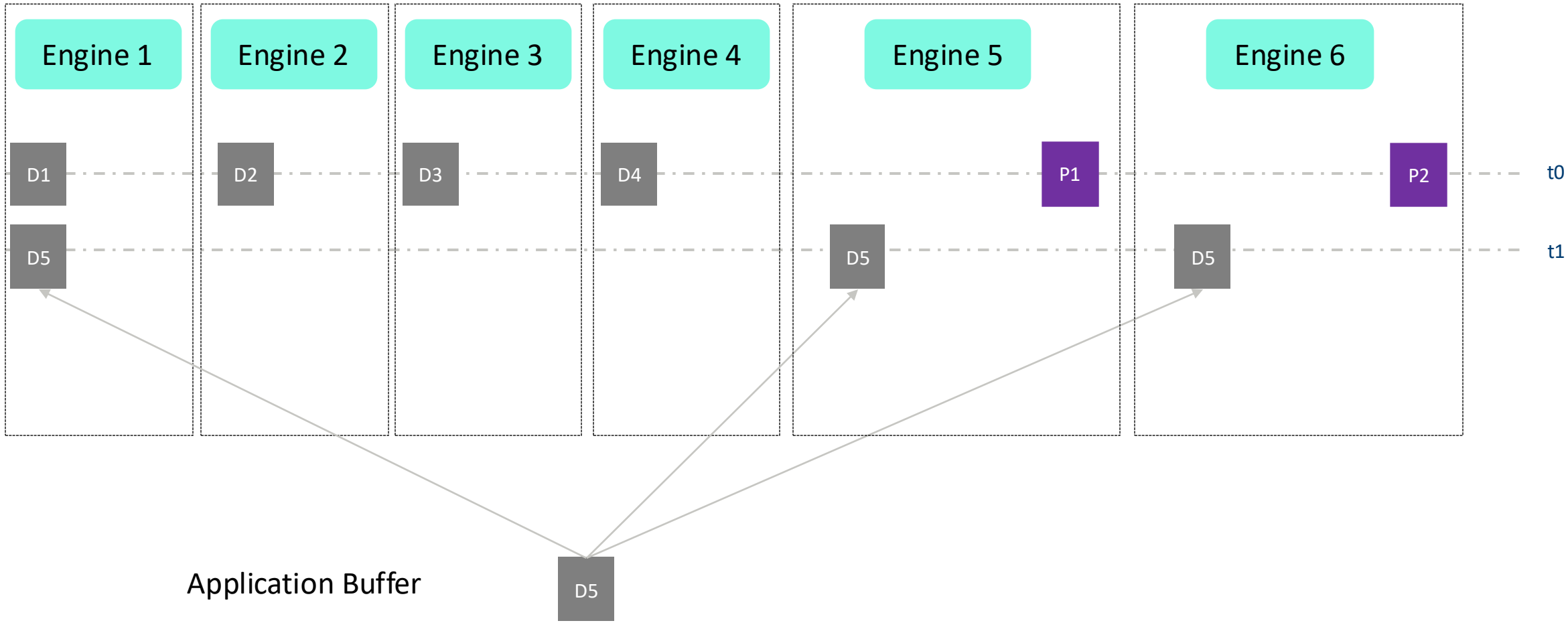
Erasure Code Partial I/O (4+2)



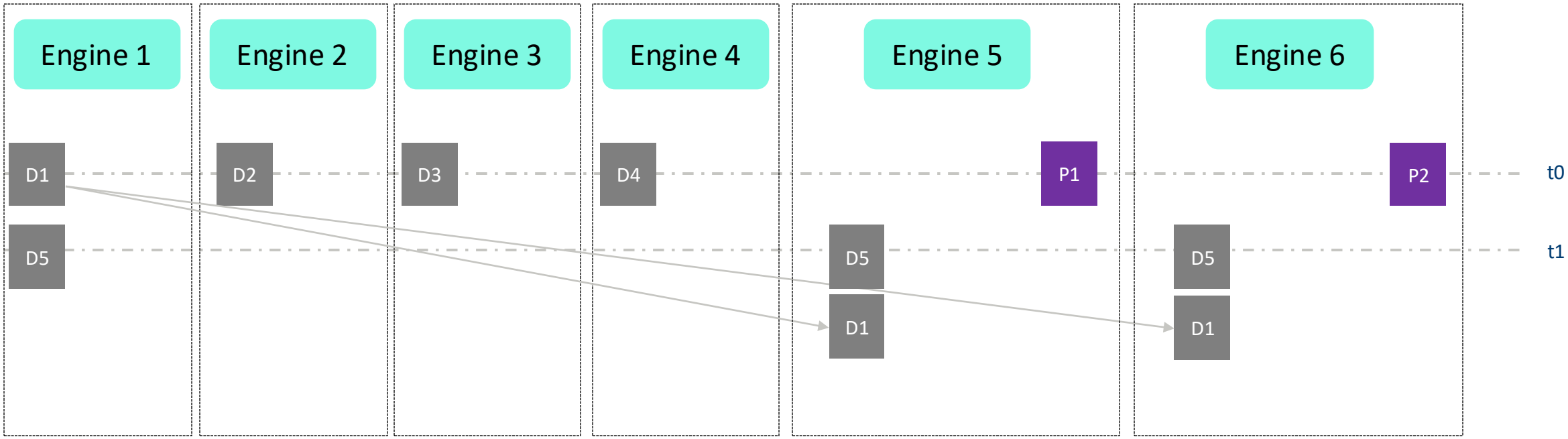
Application Buffer

D5

Erasure Code Partial I/O (4+2)



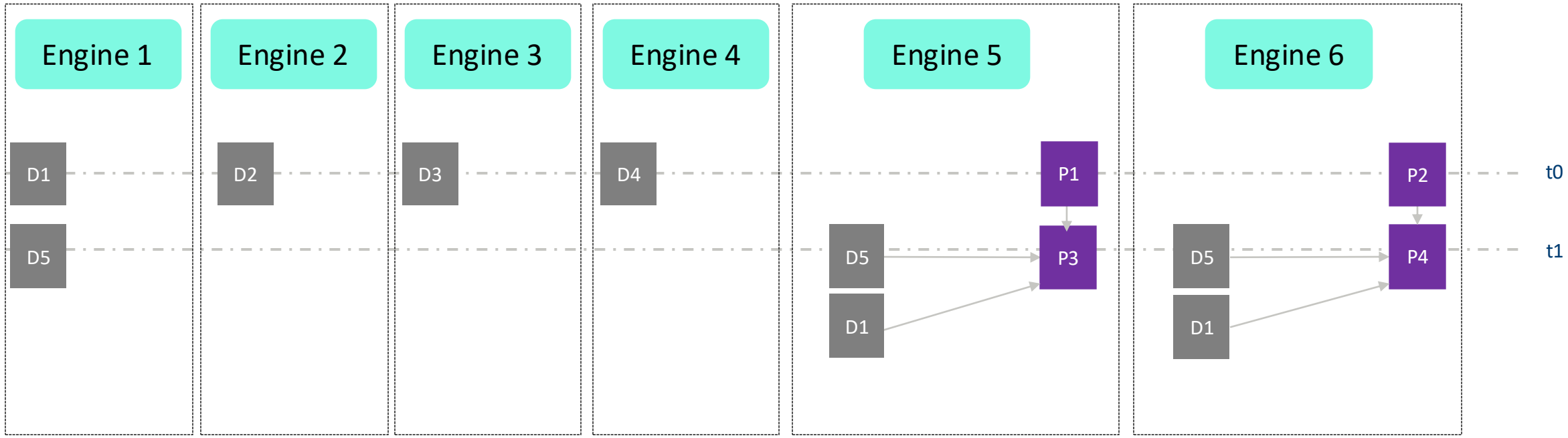
Erasure Code Aggregation (4+2)



Application Buffer

D5

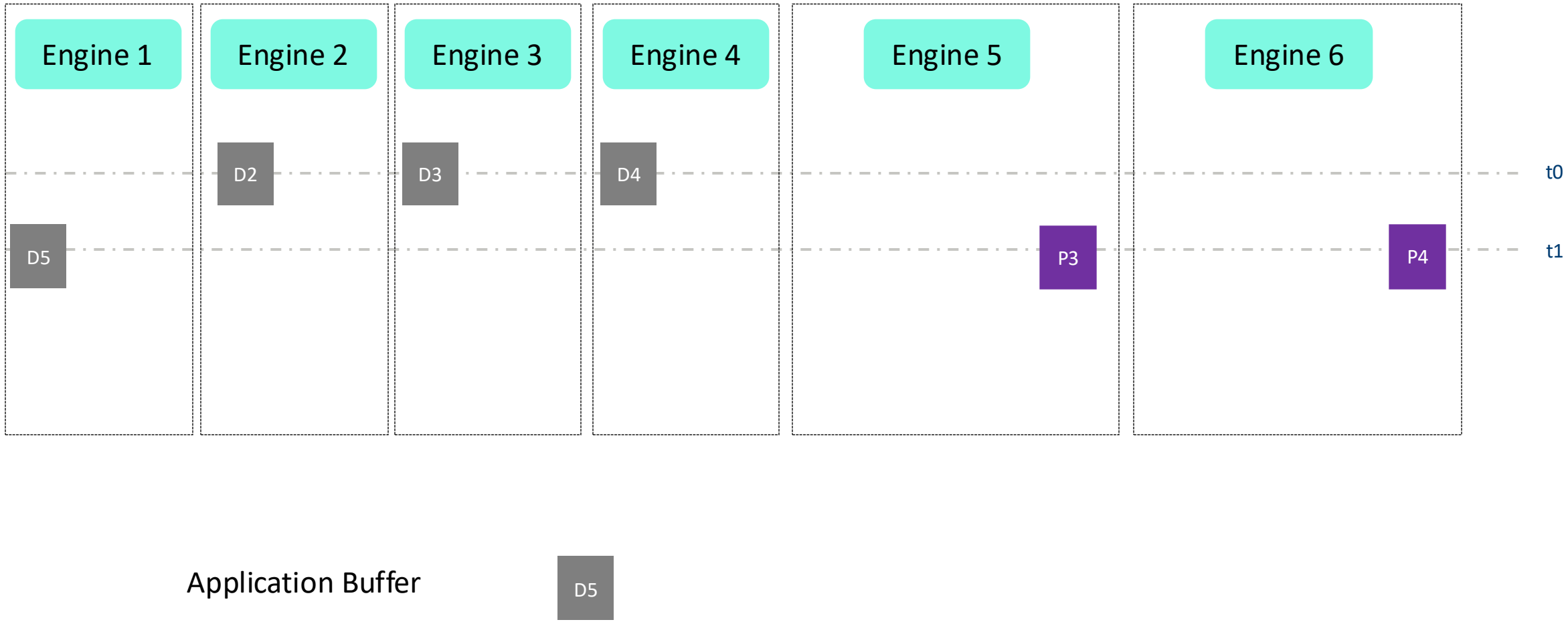
Erasure Code Aggregation (4+2)



Application Buffer

D5

Erasure Code Aggregation (4+2)



Erasure Code Read (4+2)