Milestone 7.2 - Burst Buffer & Data Integrity Demonstration

Milestone 7.3 - End-to-End Epoch Recovery Demonstration

Q7 Highlights

- Updates since Q6:
  - Integration with IOD on top of DAOS Lustre
  - Data Movement Functionality (Persist/Evict/Prefetch)
    - February monthly stakeholder meeting
  - VPIC I/O Kernel & H5Part integration
- Demonstration system is Intel's Lola Cluster:
  - 4 CNs: lola-[20,25-27]
  - 4 IONs: lola-[12-15] using flash & Lustre cross-mounts to share data stored in burst buffers
  - 4 OSSs: lola-[16-19] using flash
  - 1 MDS: lola-2

7.2/7.3 Demo on DAOS Lustre

- HDF5 SA 1.1.3 [Mandatory] HDF5 support for transactional I/O
- HDF5 SA 1.1.4 [Mandatory] HDF5 support for end-to-end data integrity
- HDF5 SA 1.2.3 [Mandatory] Isolating operations within transactions
- HDF5SA 1.2.4 [Mandatory] Support for end-to-end data integrity
- IOD SA 4.4 [Mandatory] Group transactions and versioned objects
- IOD SA 4.8 [Mandatory] End-to-end data integrity
- DAOS SA 1.4 [Mandatory] Transaction
- DAOS SA 3.3 [Mandatory] Epoch & Recovery

6.2 Demo on DAOS Lustre

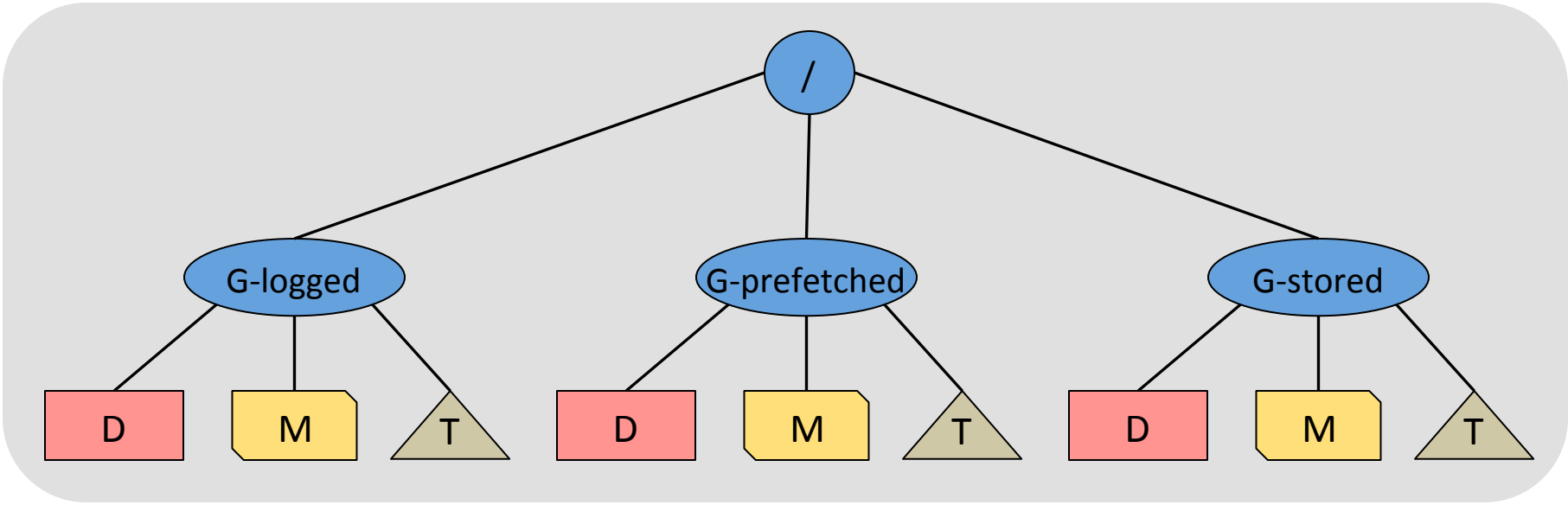
- Re-running HDF5 Integration test on new stack
- Requirements Covered:
  - HDF5 SA 1.1.3 [Mandatory] HDF5 support for transactional I/O
  - HDF5 SA 1.2.3 [Mandatory] Isolating operations within transactions
  - IOD SA 4.4 [Mandatory] Group transactions and versioned objects

- Run m6.2_demo.sh

7.2 Persist / Evict / Prefetch Demo

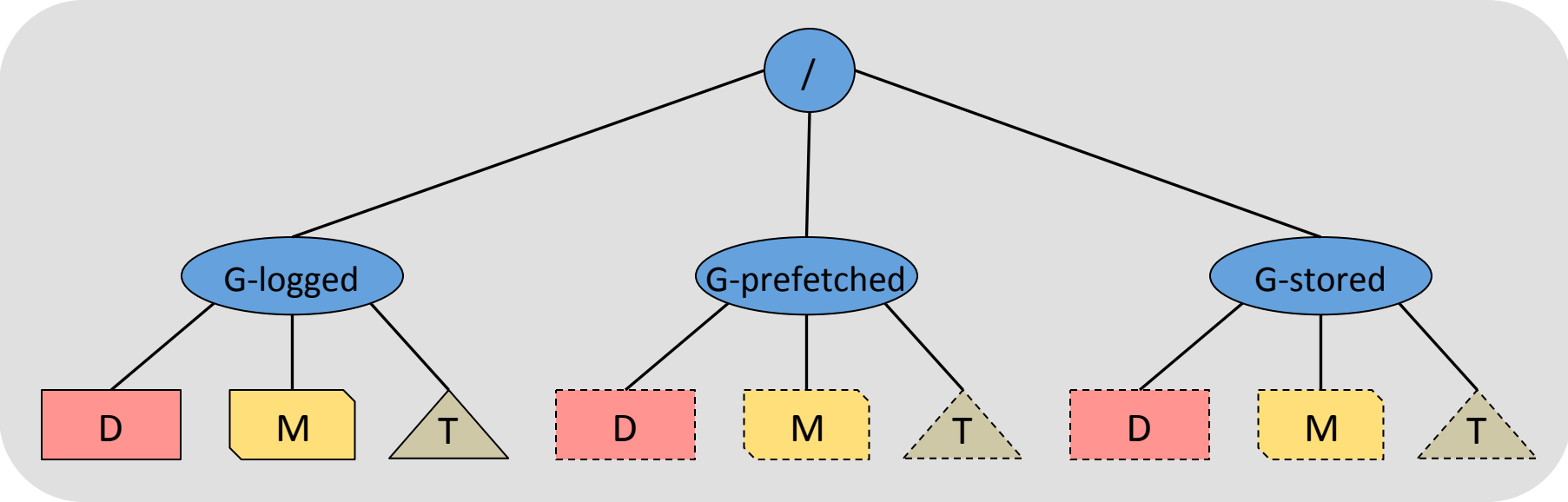
- Testing Expectations Covered
  - Create multiple transactions; persist from BB to DAOS
  - Create multiple transactions; persist from BB to DAOS; evict from BB, releasing space
    - IOD KV evict a no-op in Q7
  - Close container; reopen; asynchronously prefetch multiple objects from DAOS to BB
- Requirements Covered
  - HDF5 SA 1.1.3 [Mandatory] HDF5 support for transactional I/O
  - HDF5 SA 1.2.3 [Mandatory] Isolating operations within transactions
  - IOD SA 4.4 [Mandatory] Group transactions and versioned objects
  - DAOS SA 1.4 [Mandatory] Transaction

7.2 Persist / Evict / Prefetch Demo



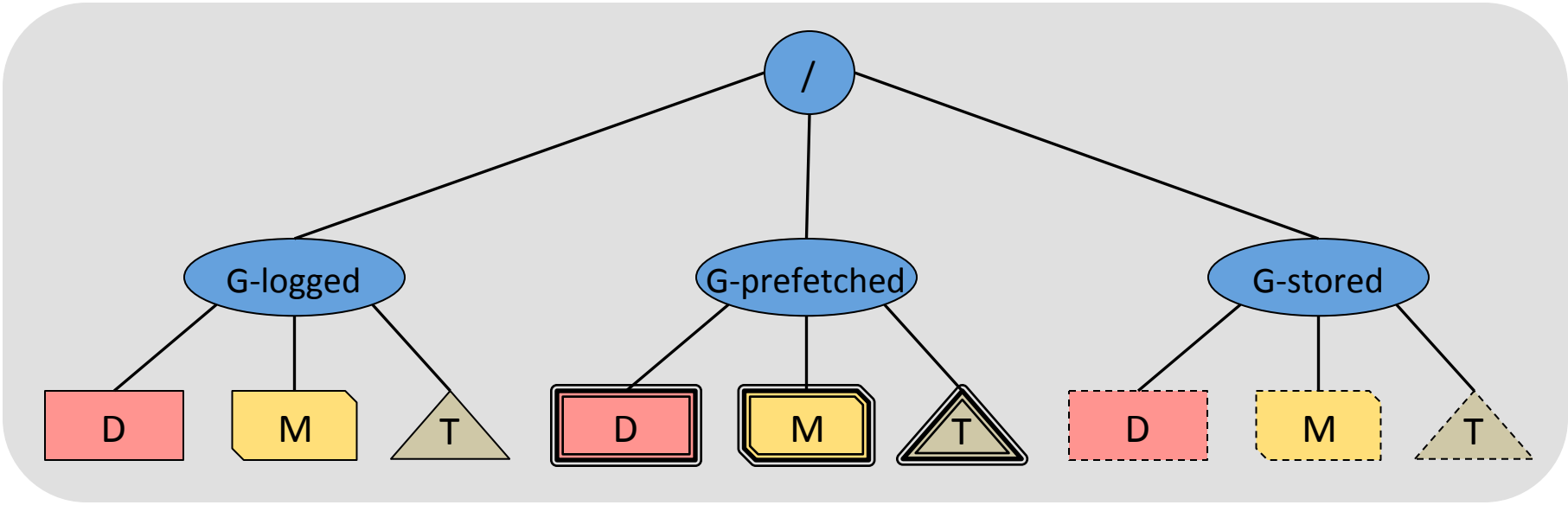
1. tr2: G-* created
2. tr3: D*, M*, T* created
3. **persist** cv 3; print contents
4. tr4 & tr5: D*, M*, T* updated
5. <show BB space #1>
6. **persist** cv 5; print contents

7.2 Persist / Evict / Prefetch Demo



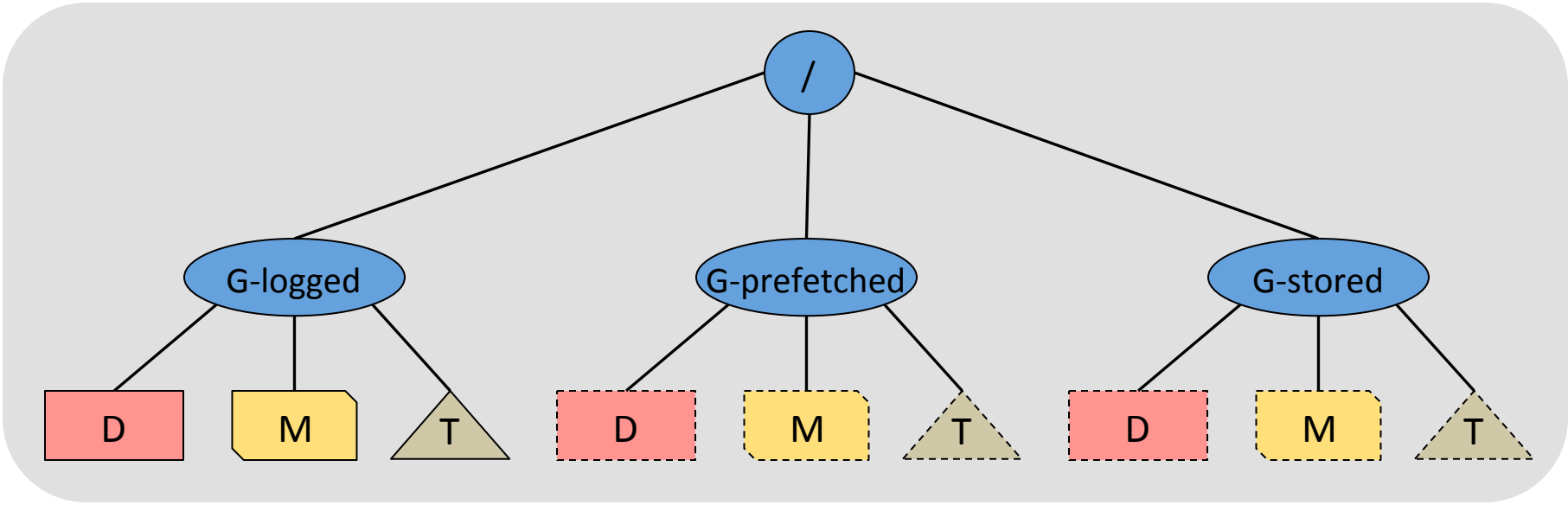
1. tr2: G-* created
2. tr3: D*, M*, T* created
3. persist cv 3; print contents
4. tr4 & tr5: D*, M*, T* updated
5. <show BB space #1>
6. persist cv 5; print contents
7. **evict** G-prefetched/*, G-stored/*
8. <show BB space #2>
9. close container
10. reopen container
11. <show BB space #3>

7.2 Persist / Evict / Prefetch Demo



1. tr2: G-* created
2. tr3: D*, M*, T* created
3. persist cv 3; print contents
4. tr4 & tr5: D*, M*, T* updated
5. <show BB space #1>
6. persist cv 5; print contents
7. evict G-prefetched/*, G-stored/*
8. <show BB space #2>
9. close container
10. reopen container
11. <show BB space #3>
12. **prefetch replicas** G-prefetched/* (asynch)
13. <show BB space #4>
14. read & print G*/D*, G*/M*

7.2 Persist / Evict / Prefetch Demo



1. tr2: G-* created
2. tr3: D*, M*, T* created
3. persist cv 3; print contents
4. tr4 & tr5: D*, M*, T* updated
5. <show BB space #1>
6. persist cv 5; print contents
7. evict G-prefetched/*, G-stored/*
8. <show BB space #2>
9. close container
10. reopen container
11. <show BB space #3>
12. prefetch replicas G-prefetched/* (asynch)
13. <show BB space #4>
14. read & print G*/D*, G*/M*
15. **evict replicas** G-prefetched/*
16. <show BB space #5>

7.2 PEP Demo – BB space usage

```
[root@lola-3 ff_output]# grep Creatin
Creating Group ID 3000000000000020 (C
Creating Group ID 3000000000000038 (C
Creating Group ID 3000000000000050 (C
Creating Dataset ID 7000000000000000
Creating Datatype ID 5000000000000002
Creating Map ID 3000000000000001 (CV
Creating Datatype ID 500000000000000a
Creating Dataset ID 7000000000000008
Creating Map ID 3000000000000019 (CV
Creating Datatype ID 5000000000000012
Creating Dataset ID 7000000000000010
```

```
<...>
echo  "Summary usage - Arrays and
du -bs /tmp/iod_plfs/eff_pep.h5

<...>

echo "Per-Array usage for H5Datas
echo -n "/G-logged/D      "
du -bs /tmp/iod_plfs/eff_pep.h5/7000000000000000
echo -n "/G-prefetched/D "
du -bs /tmp/iod_plfs/eff_pep.h5/7000000000000008
echo -n "/G-stored/D      "
du -bs /tmp/iod_plfs/eff_pep.h5/7000000000000010

< ... >
```

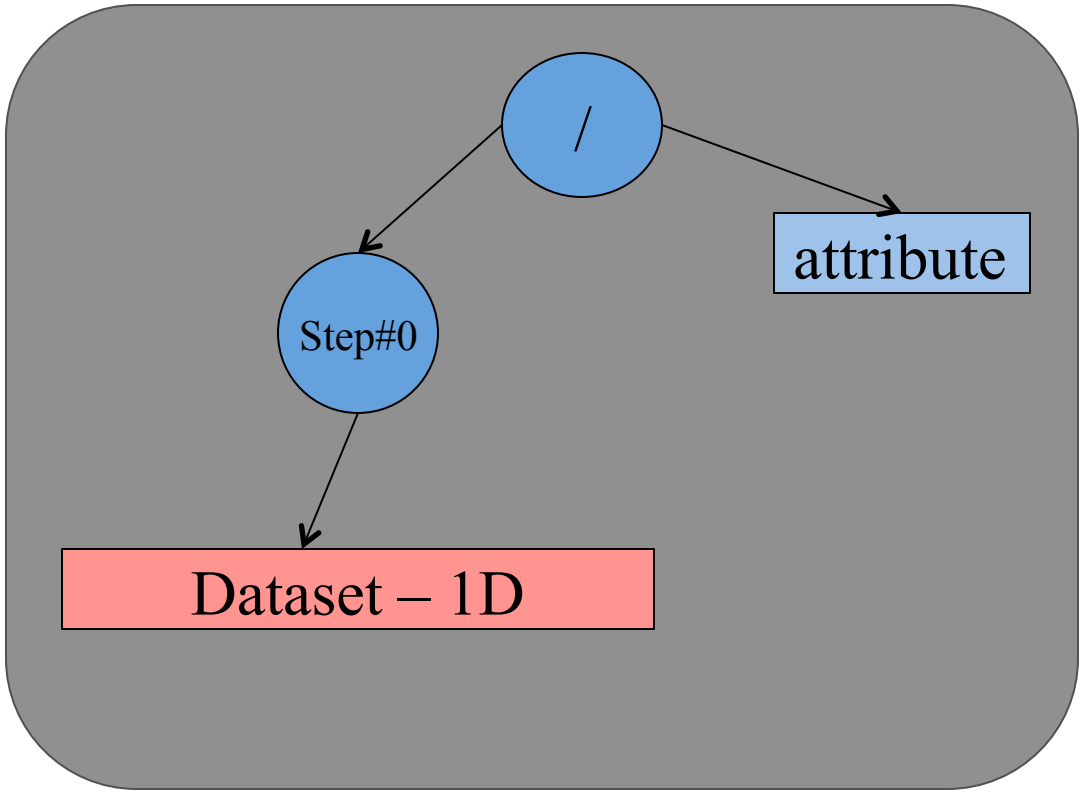
```
======= BB space after prefetch of replicas
=======
Summary usage - All KVs (including H5Maps):
17031564        /tmp/iod_root/0/eff_pep.h5
Summary usage - Arrays and Blobs (H5Datasets and H5NamedDatatypes)
28764    /tmp/iod_plfs/eff_pep.h5

Per-KV usage for H5Map objects
/G-logged/M      528470  /tmp/iod_root/0/eff_pep.h5/3000000000000001
/G-prefetched/M 1056854  /tmp/iod_root/0/eff_pep.h5/3000000000000019
/G-stored/M      528470  /tmp/iod_root/0/eff_pep.h5/3000000000000031

Per-Array usage for H5Datasets
/G-logged/D      4144    /tmp/iod_plfs/eff_pep.h5/7000000000000000
/G-prefetched/D  4112    /tmp/iod_plfs/eff_pep.h5/7000000000000008
/G-stored/D      4096    /tmp/iod_plfs/eff_pep.h5/7000000000000010

Per-Blob usage for H5NamedDatatypes
/G-logged/T      4110    /tmp/iod_plfs/eff_pep.h5/5000000000000002
/G-prefetched/T  4110    /tmp/iod_plfs/eff_pep.h5/500000000000000a
/G-stored/T      4096    /tmp/iod_plfs/eff_pep.h5/5000000000000012
```

Run Demo

VPIC I/O Kernel

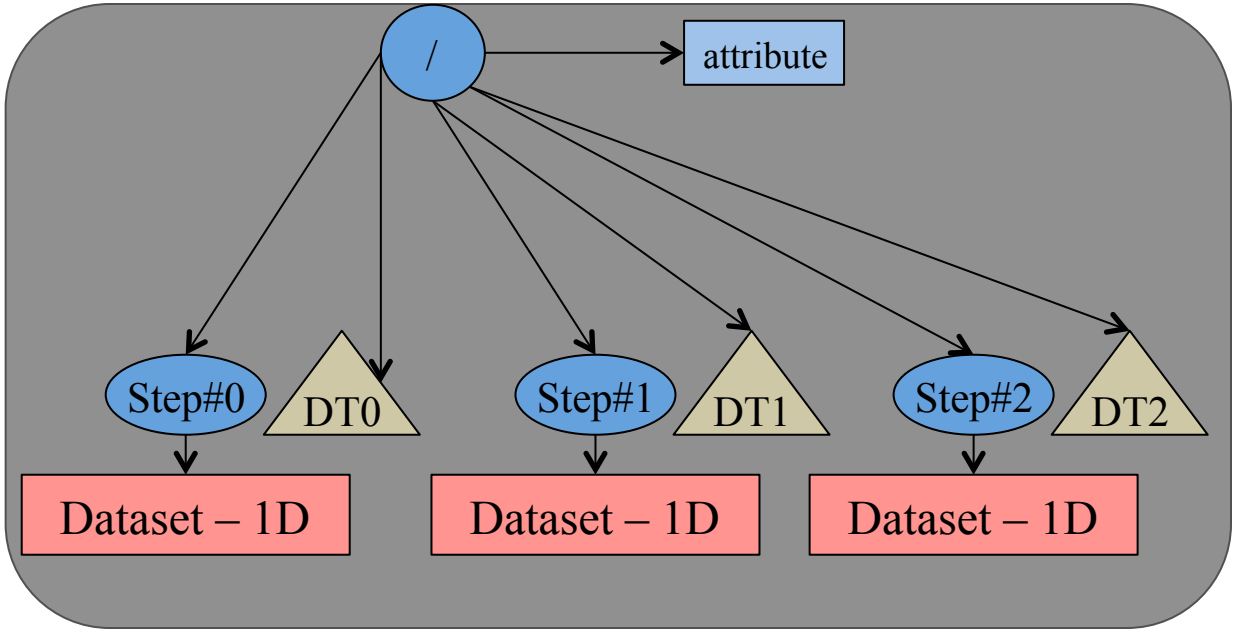
- VPIC (vector particle-in-cell) is a computer code that simulates plasma behavior efficiently. The I/O access pattern of VPIC was extracted into an I/O kernel code.
  - Suren Byna (LBL)
  - We modified the kernel code to run on the FastForward I/O stack.
- VPIC uses H5Part which is an I/O library built on top of HDF5.
  - Add new H5Part API to access the HDF5 file using the new HDF5 FastForward API routines.
  - Prefetch, Evict, and Asynchronous functionality will be demonstrated in a separate test program.

VPICIO Original



- Create the HDF5 file.
- Create and write an Attribute on the root group of the HDF5 file.
- Write a timestep:
  - Create timestep Group under the root group.
  - Select particles (HDF5 hyperslab) for each MPI process to write to the dataset in this timestep.
  - Create a dataset.
  - Write to that dataset.
  - Close the dataset.
- Close the file.

VPICIO EFF Version



- Create N timesteps instead of just 1 (runtime argument -s).
- Each timestep is done in 1 transaction.
- Add a Named Datatype object at each timestep (to exercise IOD BLOBs).
- Persist every R timesteps (runtime argument -r).
- Add a read/verification phase
  - Reads back data at every timestep and verifies correctness.
  - Runtime argument (-v) to skip to this phase.

Demo

- Run VPICIO with no corruption injection or Node failure
- Rerun in Verify Mode with different number of processes reading.

End-To-End Data Integrity

- HDF5 SA 1.1.4 [Mandatory] HDF5 support for end-to-end data integrity
- HDF5 SA 1.2.4 [Mandatory] Support for end-to-end data integrity
- IOD SA 4.8 [Mandatory] End-to-end data integrity
- Run VPIC with environment flags to inject corruptions on all 3 object types (KV, ARRAY, BLOB) at different timesteps for:
  - ION resident data
  - Checksums of ION resident data
  - DAOS resident data
  - Checksums of DAOS resident data
- Show Corruption detection in verification phase.
- Possibly re-run verification phase with different number of processes (if time permits).

Corruption Injection

- IOD corruption tool used to corrupt data
  - iod/bin/iod_corrupt_tool
  - API routines corrupt_data() & kv_corrupt()
- ION-resident data and metadata for blobs and arrays can be corrupted in-place
  - iod_corrupt_tool [--checksum] –oid OID –tid TID
  - Subsequent reads on OID@TID will fail checksum check
  - Subsequent persists on OID@TID *may* fail
  - Will not fail for KV's since IOD does not currently reorganize KV's during persist
  - Will typically fail for blobs/arrays except when the data in IONs already matches the checksum chunk size that IOD uses on DAOS
- DAOS-resident data cannot be corrupted without changing the epoch
- Therefore we must corrupt DAOS objects before the epoch commit
  - For blobs/arrays, we pass a hint to the persist call
  - For KV's, we corrupt them in ION's before doing the persist

Epoch Recovery with VPIC

- DAOS SA 3.3 [Mandatory] Epoch & Recovery
  - Scenario 1: compute node (CN) failure
  - Scenario 2: I/O node (ION) failure
  - Scenario 3: transient failure of a Lustre server, e.g.: OST/MDT failover/reboot
  - Scenario 4: permanent failure of a Lustre server
- Demo Scenario 1 & 2
  - Run VPICIO and power cycle CN (1) and ION (2) after a persist.
  - Wait for node to come back then clear the Burst Buffer.
  - Re-launch VPICIO.
  - Expected behavior is that VPICIO will resume from the latest persisted transaction on DAOS.