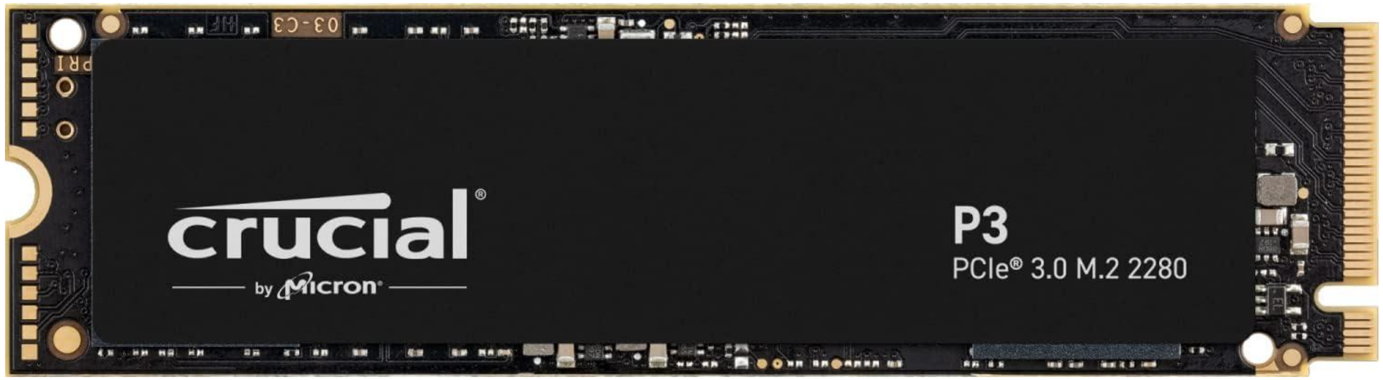
OBJECT STORES AND I/O APPROACHES

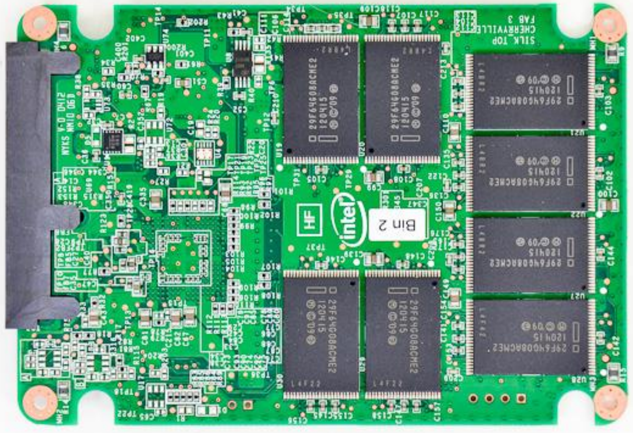
Hardware and Software interfaces

Hewlett Packard Enterprise

epcc

Storage

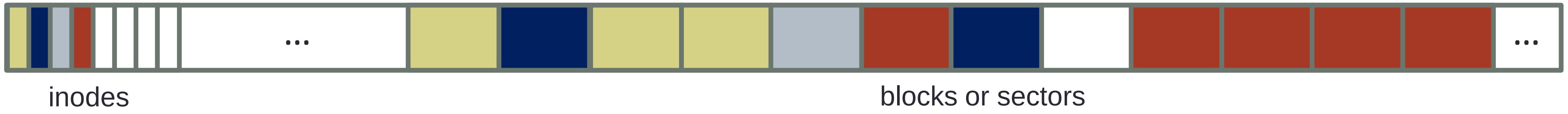

crucial
by Micron
P3
PCIe® 3.0 M.2 2280



crucial
by Micron
2.5-inch SOLID STATE DRIVE

Filesystems

- Lots of ways to store data on storage devices
- Filesystems have two components:
  - Data storage
  - Indexing
- Data stored in blocks
  - Chunks of data physically stored on hardware somewhere
- Indexing is used to associate names with blocks



  - File names are the index
  - Files may consist of many blocks
- Variable sized nature of files makes this a hard problem to solve

Filesystems

- Different ways of defining how inodes → blocks, and how directories, filenames, etc… are structured
  - As well as alternative approaches (i.e. log-structured filesystem)
  - Extended functionality (replication, distribution, backup, erasure coding, etc…)
- These are what differentiate filesystems, i.e.:
  - ext*: ext3, ext4
  - xfs
  - zfs
  - btrfs
  - etc…
- Maybe be important for performance or required functionality but the default can be used by most

Parallel filesystems

- Build on local filesystem but provide
  - Aggregated distributed local filesystem
  - Custom approach to define how inodes → blocks, and how directories, filenames, etc... are structured
  - Relaxed consistency (potentially) for concurrent writing
- i.e. Lustre:
  - Open-source parallel file system
  - Three main parts
    - Object Storage Servers (OSS)
      - Store data on one or more Object Storage Targets (OST)
      - The OST handles interaction between client data request and underlying physical storage
      - OSS typically serves 2-8 targets, each target a local disk system
      - Capacity of the file system is the sum of the capacities provided by the targets (roughly)
      - The OSS operate in parallel, independent of one another
    - Metadata Target (MDT)
      - One(ish) per filesystem
      - Storing all metadata: filenames, directories, permissions, file layout
      - Stored on Metadata Server (MDS)
  - Clients
    - Supports standard POSIX access

POSIX I/O

- Standard interface to files
  - Linux approach
  - Based on systems with single filesystem
  - open, close, write, read, etc...
- Does not support parallel or HPC I/O well
  - Designed for one active writer
  - Consistency requirements hamper performance
  - Has a bunch of functions that can impact performance, i.e. locking (flock, etc...)
- Some filesystems/approaches relax POSIX semantics to improve performance
  - Moving beyond filesystems allows other semantics to be targeted

Object storage

- Filesystems use Files
    - container for blocks of data
    - lowest level of metadata granularity (not quite true)
- Object stores use Objects
    - container for data elements
    - lowest level of metadata granularity
- Allows individual pieces of data to be:
    - Stored
    - Indexed
    - Accessed separately
- Allows independent read/write access to “blocks” of data

Object storage

- Generally restricted interface
  - Put: Create a new object
  - Get: Retrieve the object
- Removes the requirements for lots of functionality r.e. POSIX style I/O
- Traditionally objects are immutable
  - Once created cannot be changed
  - This removes the locking requirement seen for file writes
  - Makes updates similar to log-append filesystems, i.e. copy and update
  - Can cause capacity issues (although objects can be deleted)
- Object ID generated when created
  - Used for access
  - Can be used for location purposes in some systems

Object stores

- Often helper services and interfaces
  - Manage metadata
  - Permissions
  - Querying
  - Etc…
- Distribution and redundancy etc… part of the complexity
  - Often eventual consistency
- Lots of complexity in implementations
- Commonly use web interfaces as part of the Put/Get interface

S3 – Simple Storage Service

- AWS storage service/interface
  - Defacto storage interface for a range of object stores
- Uses a container model
  - Buckets contain objects
  - Buckets are the location point for data
  - Defined access control, accounting, logging, etc…
  - Bucket names have to be globally unique
- Buckets can be unlimited in size
  - Maximum object size is 5TB
  - Maximum single upload is 5GB
- A bucket has no object structure/hierarchy
  - User needs to define the logic of storage layout themselves (if there is any)
- Fundamental operations corresponding to HTTP actions:
  - http://bucket.s3.amazonaws.com/object
  - POST a new object or update an existing object.
  - GET an existing object from a bucket.
  - DELETE an object from the bucket
  - LIST keys present in a bucket, with a filter.

S3

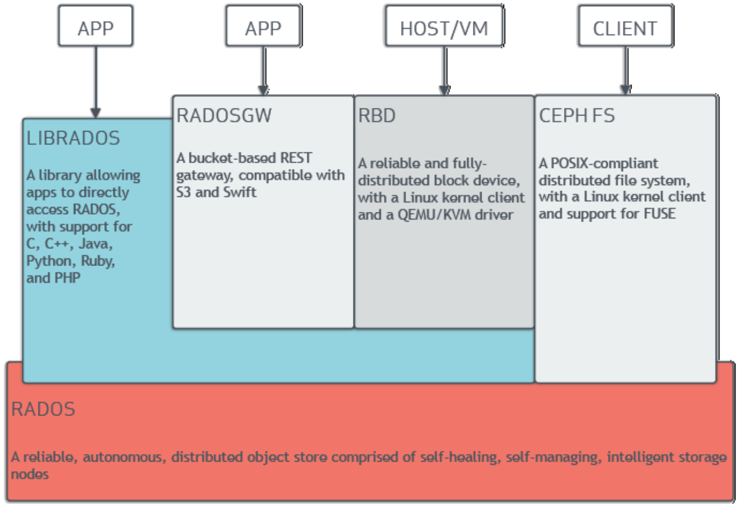
- Objects are combination of data and metadata
- Metadata is name – value pair (key) identifying the object
  - Default has some other information as well:
    - Date last modified
    - HTTP Content-Type
    - Version (if enabled)
    - Access Control List (if configured)
  - Can add custom metadata
- Data
  - An object value can be any sequence of bytes (up to 5TB)
  - Multi-part upload to create/update objects larger than 5GB (recommended over 100MB)

S3 Consistency Model

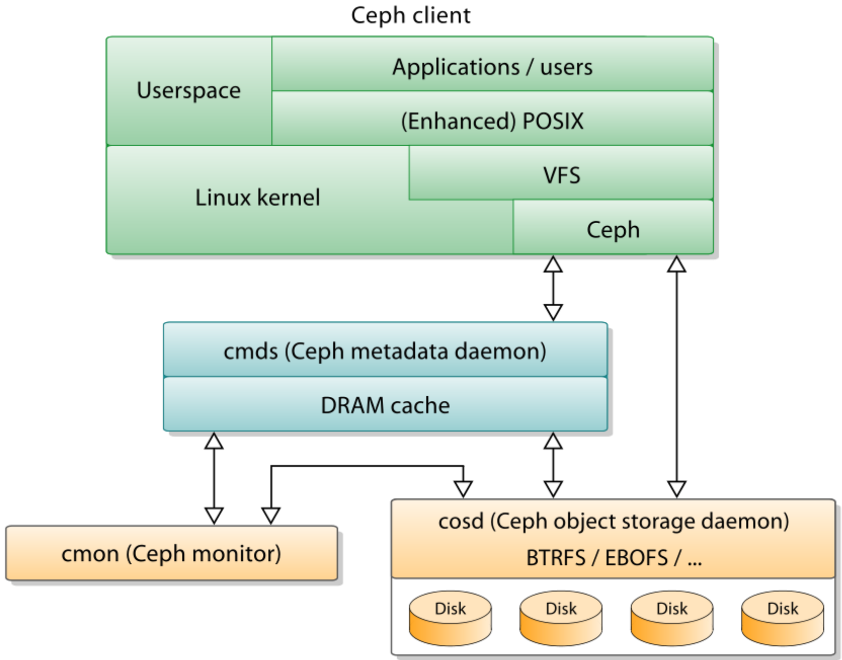
- Strong RAW (read after write) consistency
  - PUT (new and overwrite) and DELETE operations
  - READ on metadata also strong consistency
  - Across all AWS regions
- Single object updates are atomic
  - GET will either get fully old data or fully new data after update
  - Can’t link (at the S3 level) key updates to make them atomic
- Concurrent writers are *racy*
  - No automatic locking
- Bucket operations are eventually consistent
  - Deleted buckets may still appear after the delete has occurred
  - Versioned buckets may take some time to setup up initially (15 minutes)

Ceph

- Widely used object store from academic storage project
- Designed to support multiple targets
  - Traditional object store: RadosGW → S3 or Swift
  - Block interface: RBD
  - Filesystem: Ceph FS
  - Lower-level object store: LibRados
- Distributed/replicated functionality
  - Scale out by adding more Ceph servers
  - Automatic replication/consistency
    - replication, erasure coding, snapshots and clones
- Supports striping
  - Has to be done manually if using librados
- Supports tiering
- Lacking production RDMA support



APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes



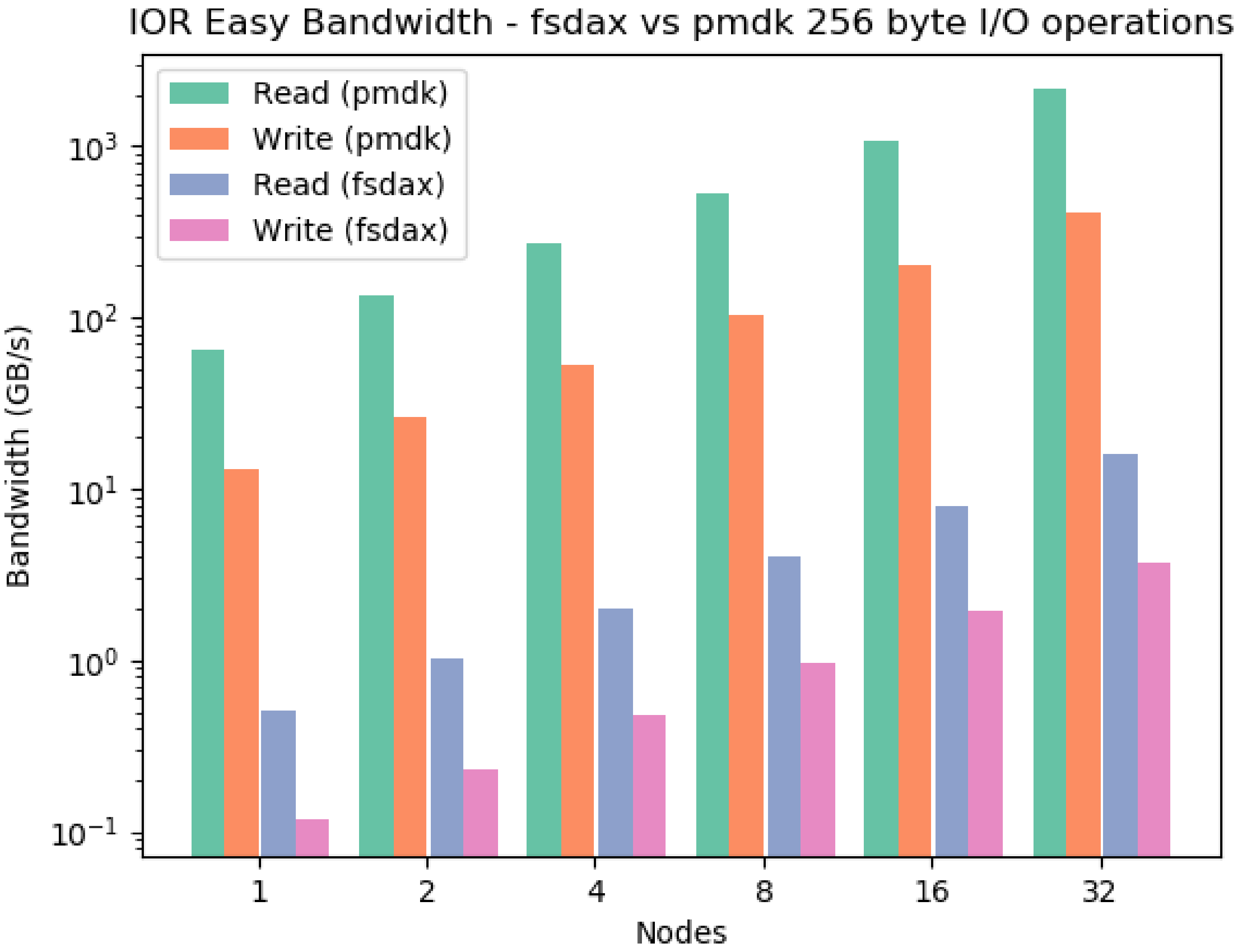
Hewlett Packard Enterprise

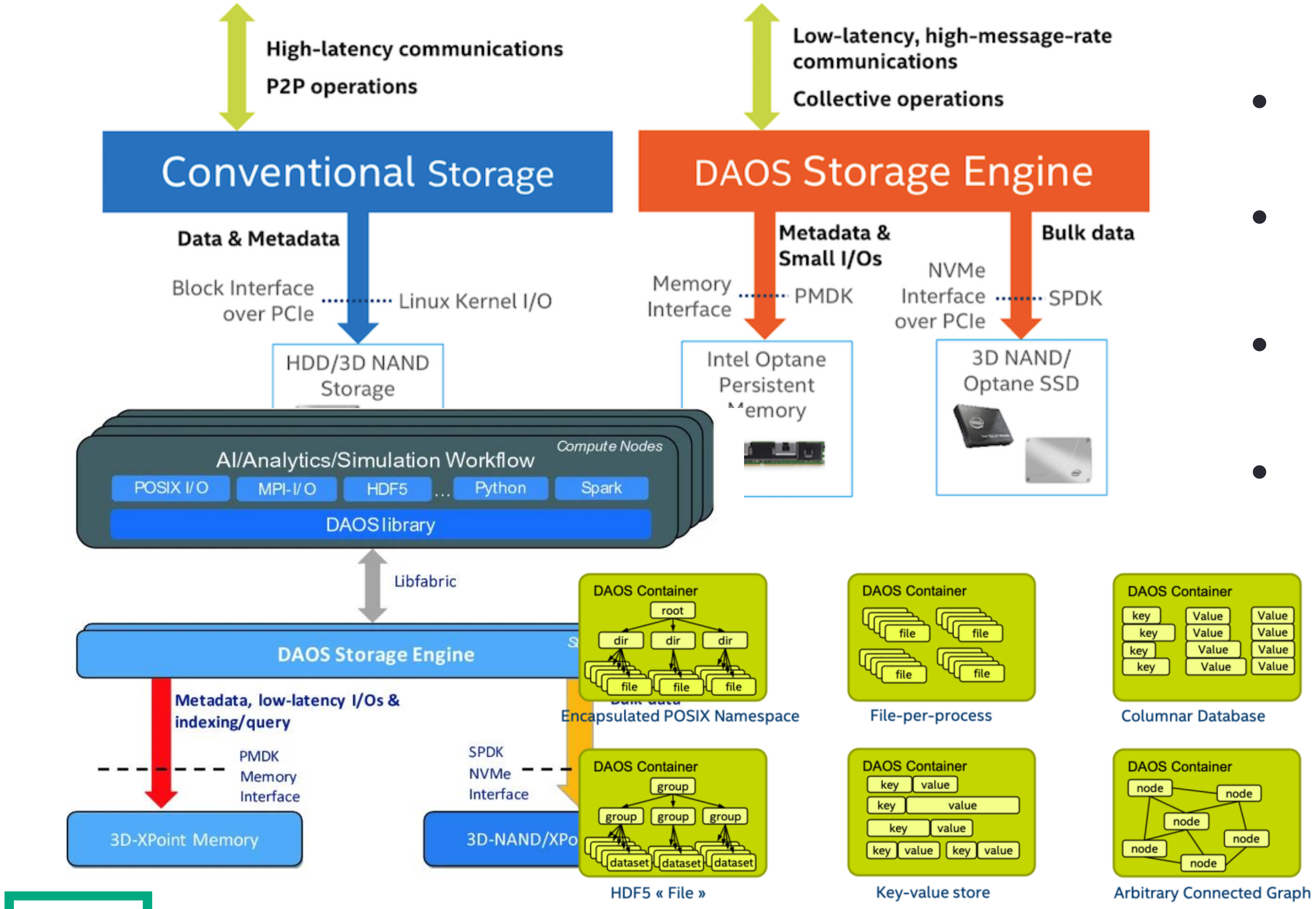
epcc

NVRAM

OPTANE
PERSISTENT MEMORY
Assembled in Taiwan
8089-A2-2210-0000293D
NMA1XXD128GPS
128GB
9424010
K18849-001
Warranty Void if Label Removed
SN:8089A2221000000293D
PN:100-586-613-00
H42898-004 FXC211110

Optane


IOR Easy Bandwidth - fsdax vs pmdk 256 byte I/O operations
Read (pmdk)
Write (pmdk)
Read (fsdax)
Write (fsdax)
Bandwidth (GB/s)
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
1
2
4
8
16
32
Nodes



High-latency communications

P2P operations

Low-latency, high-message-rate communications

Collective operations

Conventional Storage

DAOS Storage Engine

Data & Metadata

Block Interface over PCIe

Linux Kernel I/O

HDD/3D NAND Storage

Metadata & Small I/Os

Memory Interface

PMDK

Bulk data

NVMe Interface over PCIe

SPDK

Intel Optane Persistent Memory

3D NAND/ Optane SSD

Compute Nodes

AI/Analytics/Simulation Workflow

POSIX I/O | MPI-I/O | HDF5 | ... | Python | Spark

DAOS library

Libfabric

DAOS Storage Engine

Metadata, low-latency I/Os & indexing/query

PMDK Memory Interface

SPDK NVMe Interface

3D-XPoint Memory

3D-NAND/XPo

DAOS Container

root

dir

file

Encapsulated POSIX Namespace

DAOS Container

file

File-per-process

DAOS Container

key

Value

Columnar Database

DAOS Container

group

dataset

HDF5 « File »

DAOS Container

key

value

Key-value store

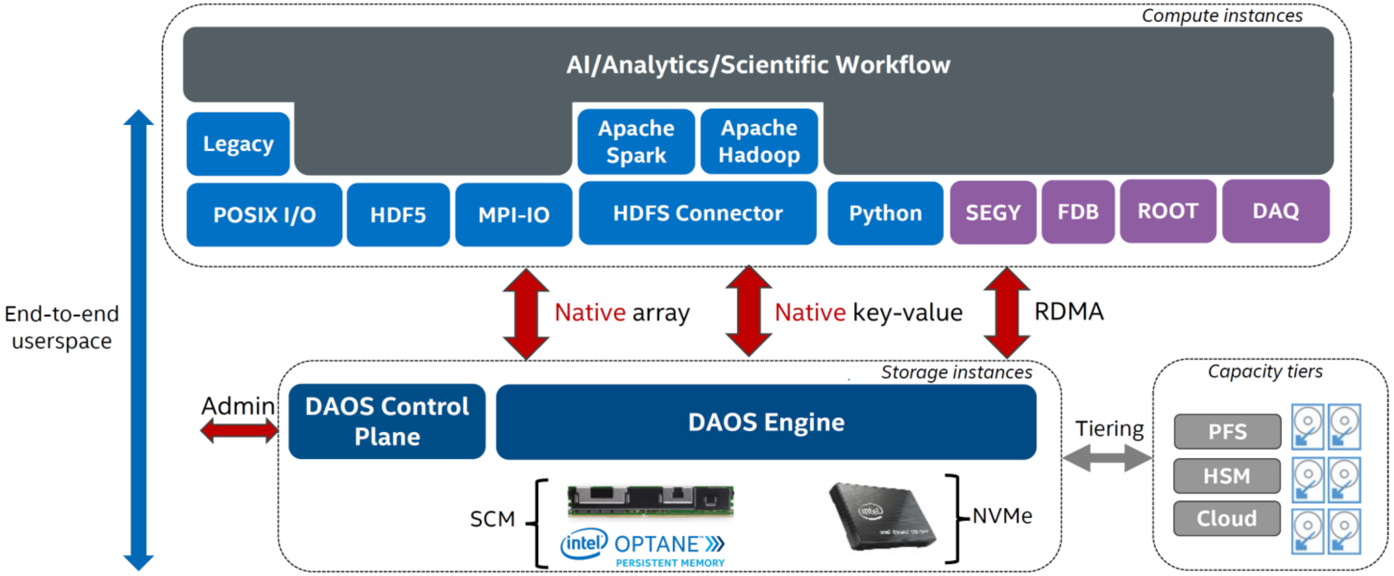
DAOS Container

node

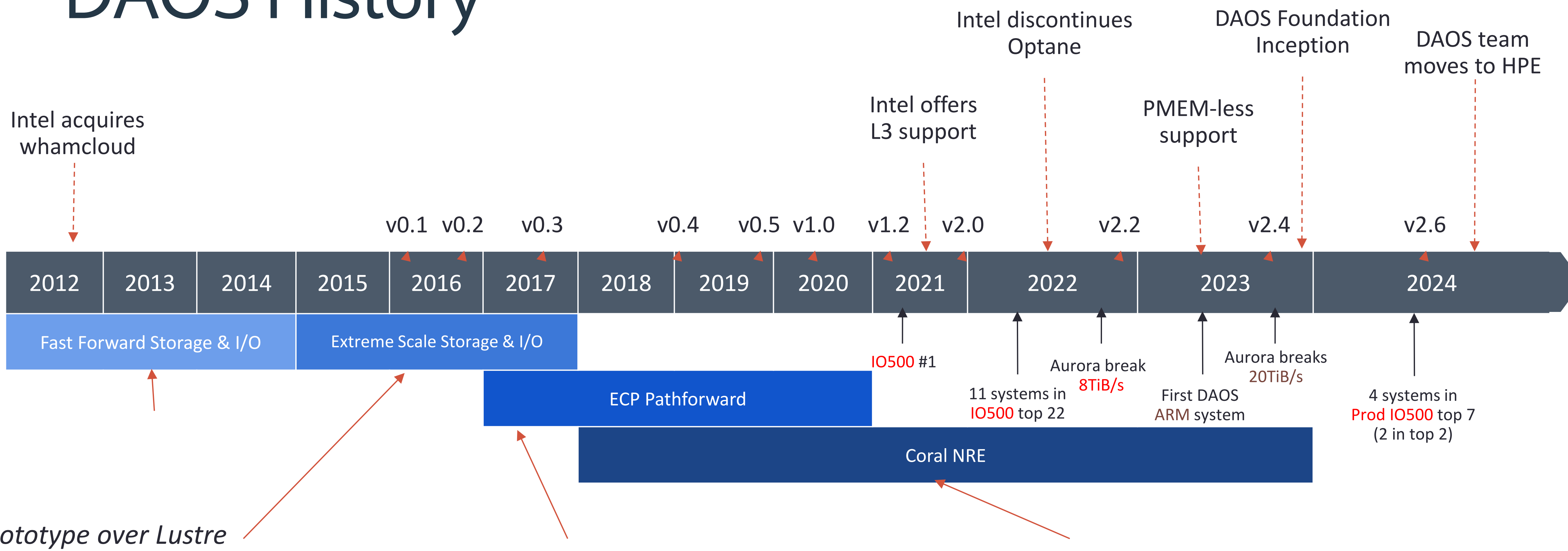
Arbitrary Connected Graph

- Native object store on non-volatile memory and NVMe devices and designed for HPC
- Pools
  - Define hardware range of data
- Containers
  - User space and data configuration definitions
- Objects
  - **Multi-level key-array** API is the native object interface with locality
  - **Key-value** API provides a simple key and variable-length value interface. It supports the traditional put, get, remove and list operations.
  - **Array API** implements a one-dimensional array of fixed-size elements addressed by a 64-bit offset. A DAOS array supports arbitrary extent read, write and punch operations.

- Range of storage interfaces
    - Native object store (libdaos)
    - Filesystem (various approaches)
    - Raw block device
    - MPI-I/O (ROMIO)
    - HDF5
    - PyDAOS
    - Spark/Hadoop
    - TensorFlow I/O
- DAOS systems built from DAOS servers
    - One per socket, has own NVMe and NVRAM
    - Scale system by adding more servers (in node or across nodes)
    - Metadata and data entirely distributed/replicated (no metadata centralisation)
    - RAFT-approach used for consensus across servers



DAOS History



Intel acquires whamcloud

Intel offers L3 support

Intel discontinues Optane

PMEM-less support

DAOS Foundation Inception

DAOS team moves to HPE

v0.1 v0.2 v0.3 v0.4 v0.5 v1.0 v1.2 v2.0 v2.2 v2.4 v2.6

2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024

Fast Forward Storage & I/O

Extreme Scale Storage & I/O

ECP Pathforward

Coral NRE

IO500 #1

11 systems in IO500 top 22

Aurora break 8TiB/s

First DAOS ARM system

Aurora breaks 20TiB/s

4 systems in Prod IO500 top 7 (2 in top 2)

Prototype over Lustre

- Build over ZFS OSD
- DAOS API over Lustre

Standalone prototype

- OS-bypass
- Persistent memory via PMDK
- Replication & self healing

DAOS embedded on FPGA

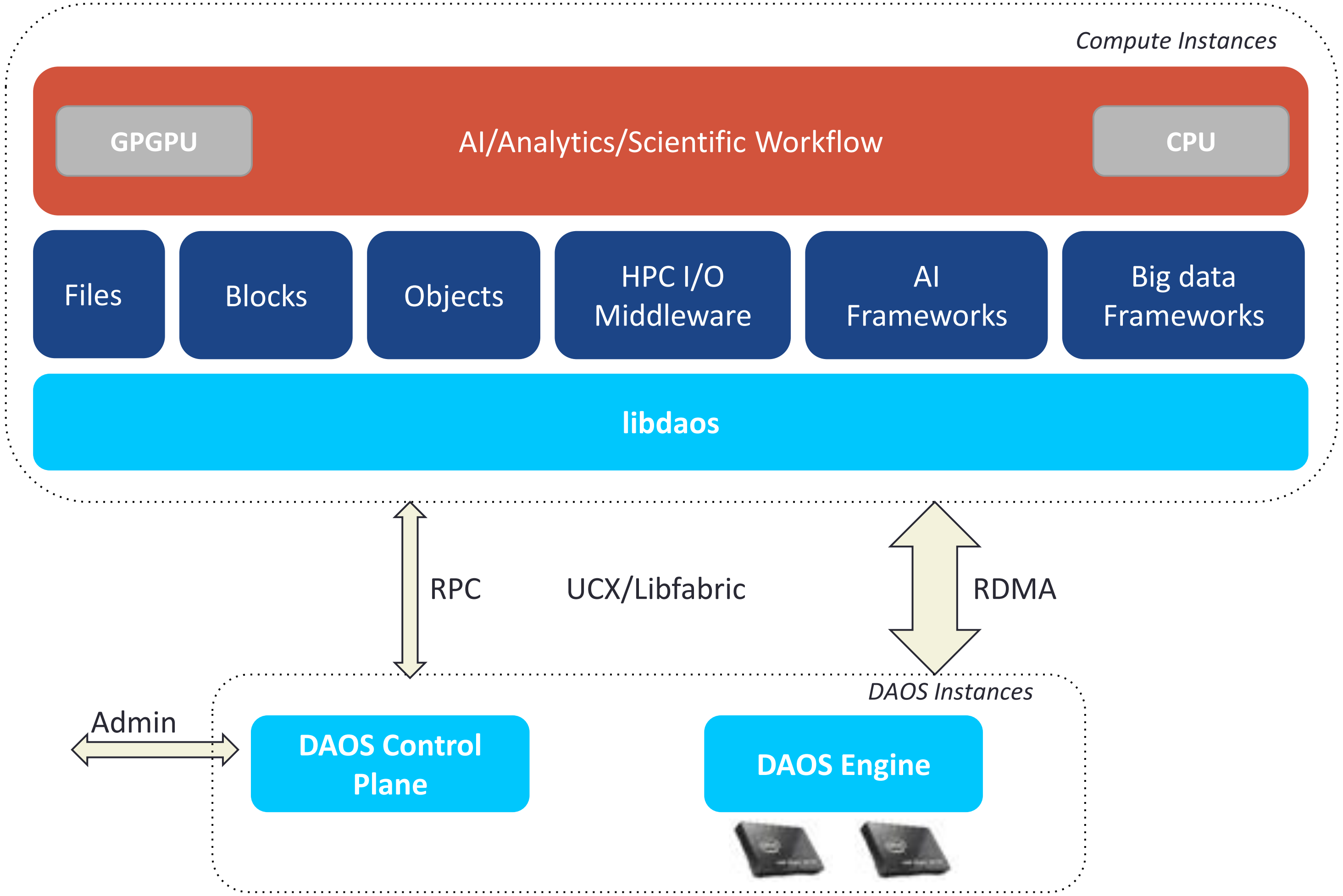
- Disaggregated I/O
- Monitoring
- NVMe SSD support via SPDK

DAOS Productization for Aurora

- Hardening
- 10+ new features
- Support for extra AI/Big data frameworks

DAOS: Nextgen Open Storage Platform

- Platform for innovation
- Files, blocks, objects and more
- Full end-to-end userspace
- Flexible built-in data protection
  - EC/replication with self-healing
- Flexible network layer
- 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



Hewlett Packard Enterprise

epcc

DAOS Design Fundamentals

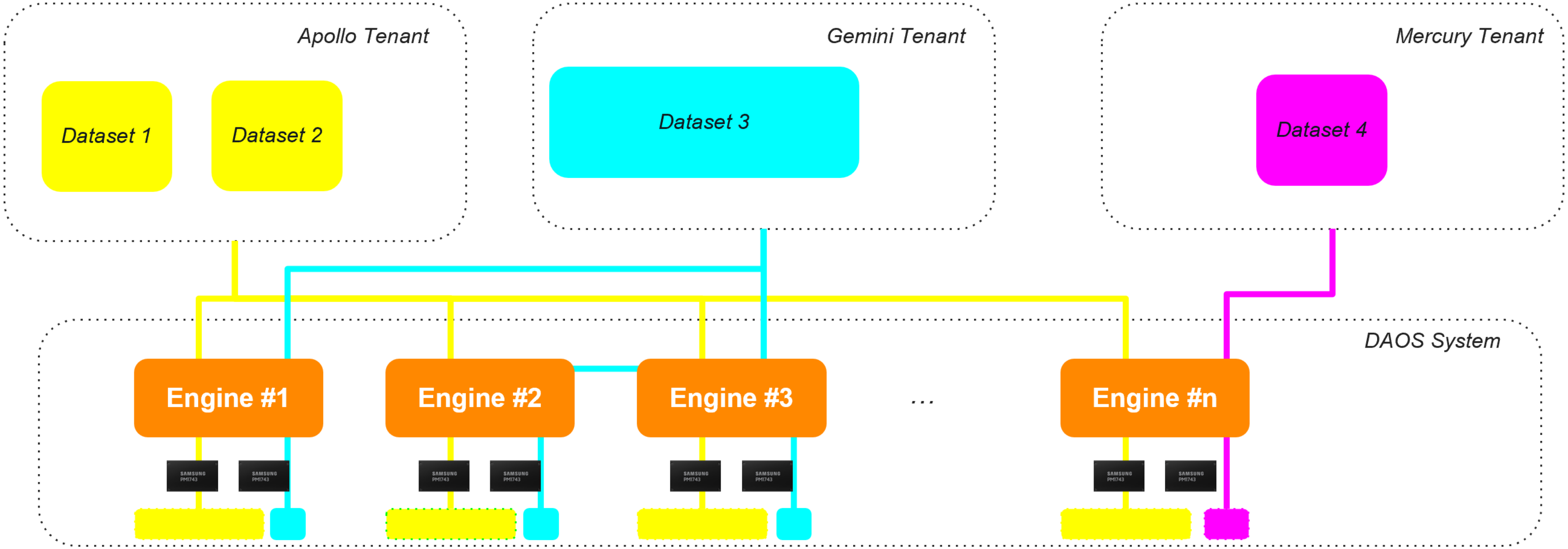
- No read-modify-write on I/O path (use versioning)
- No locking/DLM
- No client tracking or client recovery
- No centralized (meta)data server
- No global object table
- Non-blocking I/O processing (futures & promises)
- Serializable distributed transactions
- Built-in multi-tenancy
- User snapshot

Scalability & Performance

High IOPS

Unique Capabilities

Flexibility in data space creation and operation



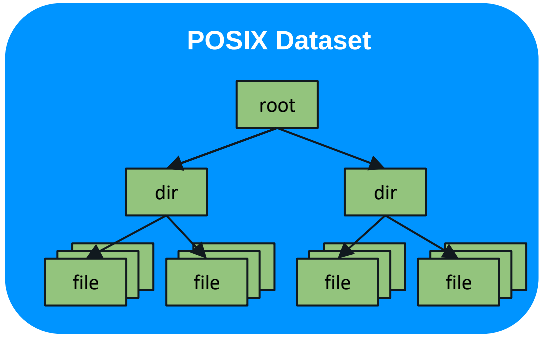
Pool 1 🟨	Apollo Tenant	100PB	20TB/s	200M IOPS
Pool 2 🟦	Gemini Tenant	10PB	2TB/s	20M IOPS
Pool 3 🟪	Mercury Tenant	30TB	80GB/s	2M IOPS

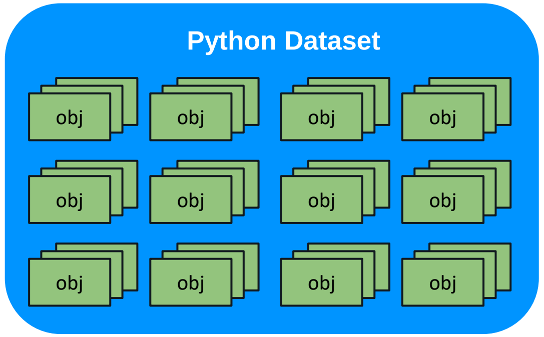
Hewlett Packard Enterprise

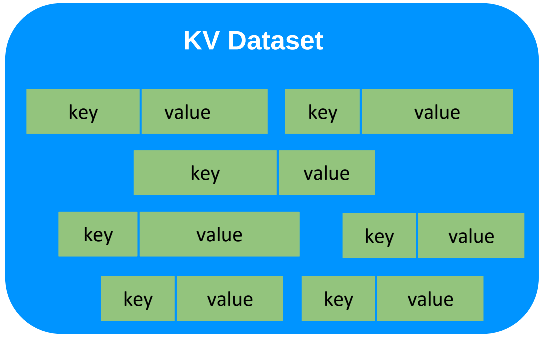
epcc

Beyond files

- New data model not based on files and file based approaches
- Introduce notion of dataset
- Basic unit of storage
- Datasets have a type
- POSIX datasets can include trillions of files/directories
- Advanced dataset query capabilities
- Unit of snapshots
- ACLs/IAM per dataset

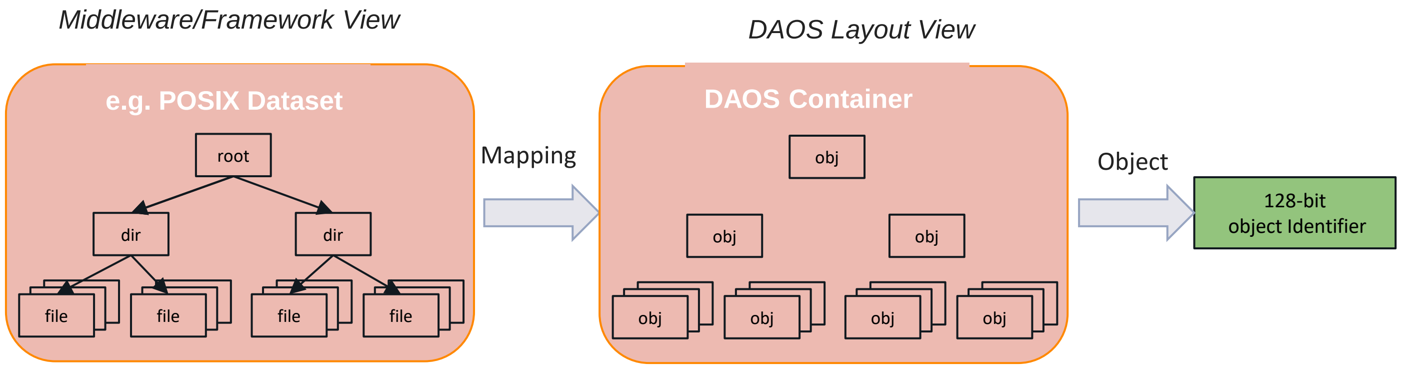


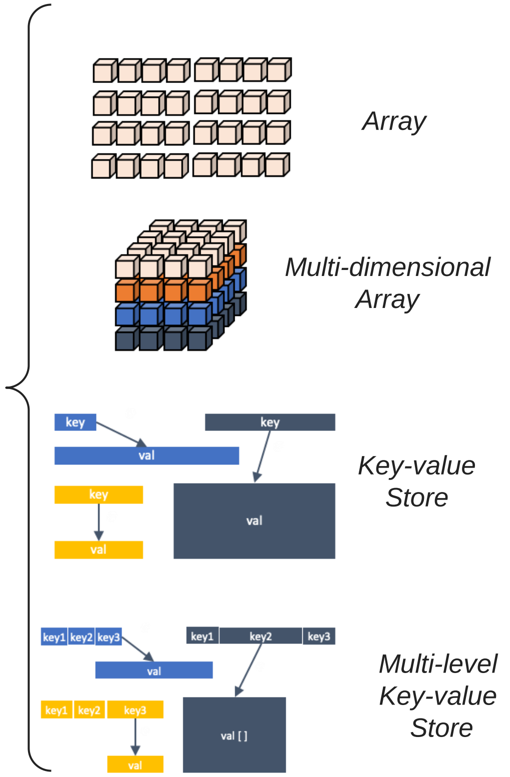




Objects

- Objects are the final level of storage for DAOS

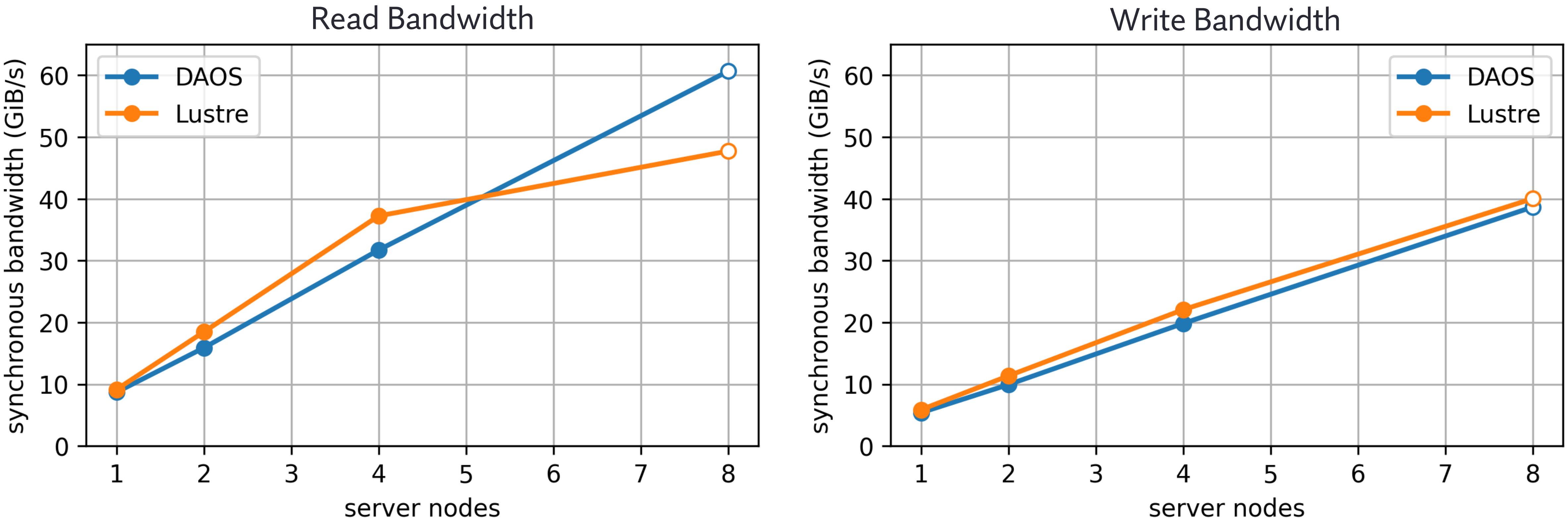




- Metadata operations are with the container
  - No size, permission/ACLs or attributes
- Sharded and erasure-coded/replicated
- Algorithmic object placement
- Very short Time To First Byte (TTFB)

DAOS Performance

- Comparing Lustre and DAOS on the same hardware
  - IOR bulk synchronous I/O

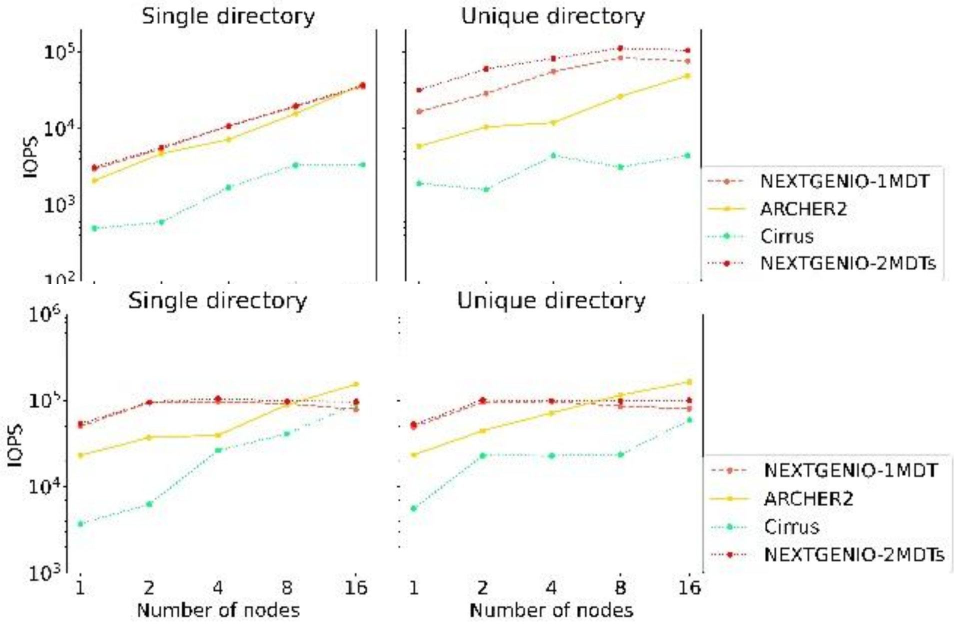


Hewlett Packard Enterprise

epcc

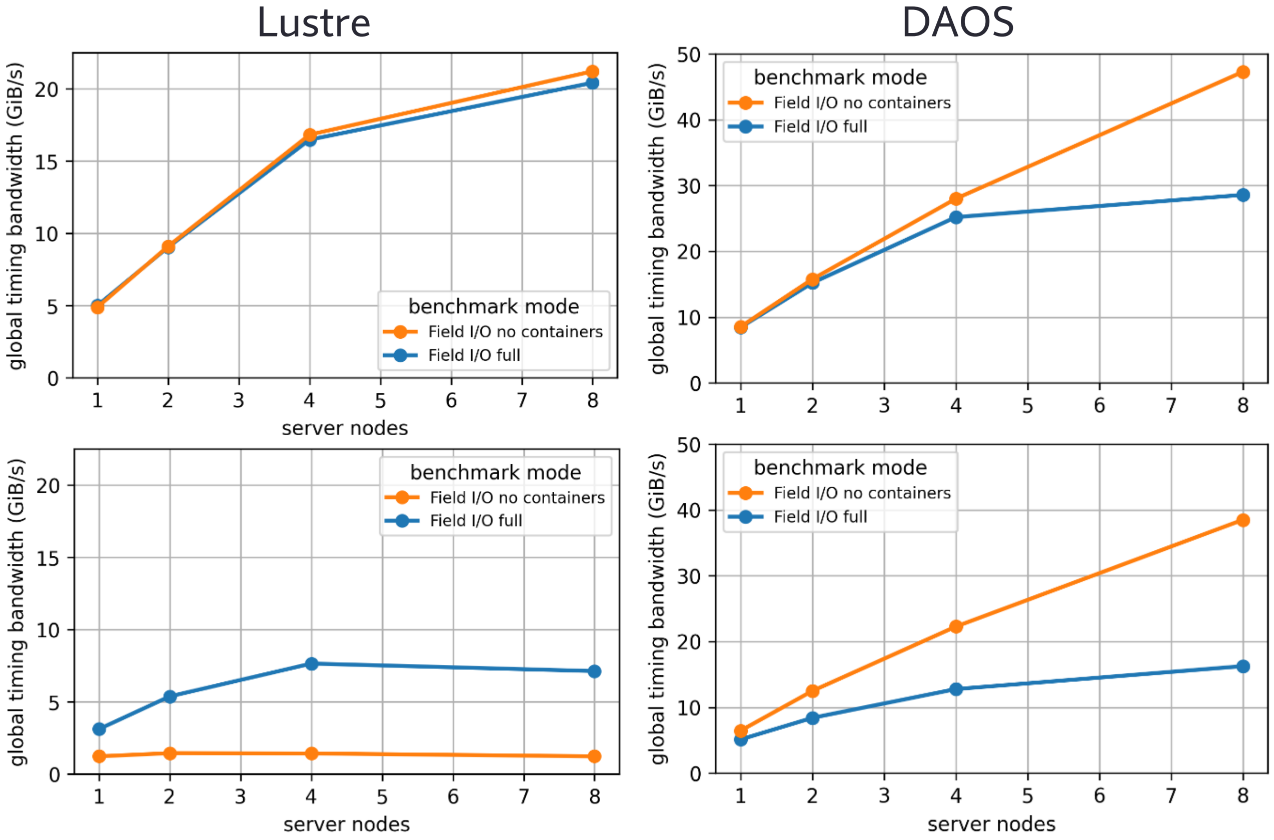
DAOS performance

- Separate read and write steps
  - More “object like” access patterns
  - Weather field -> Object or file



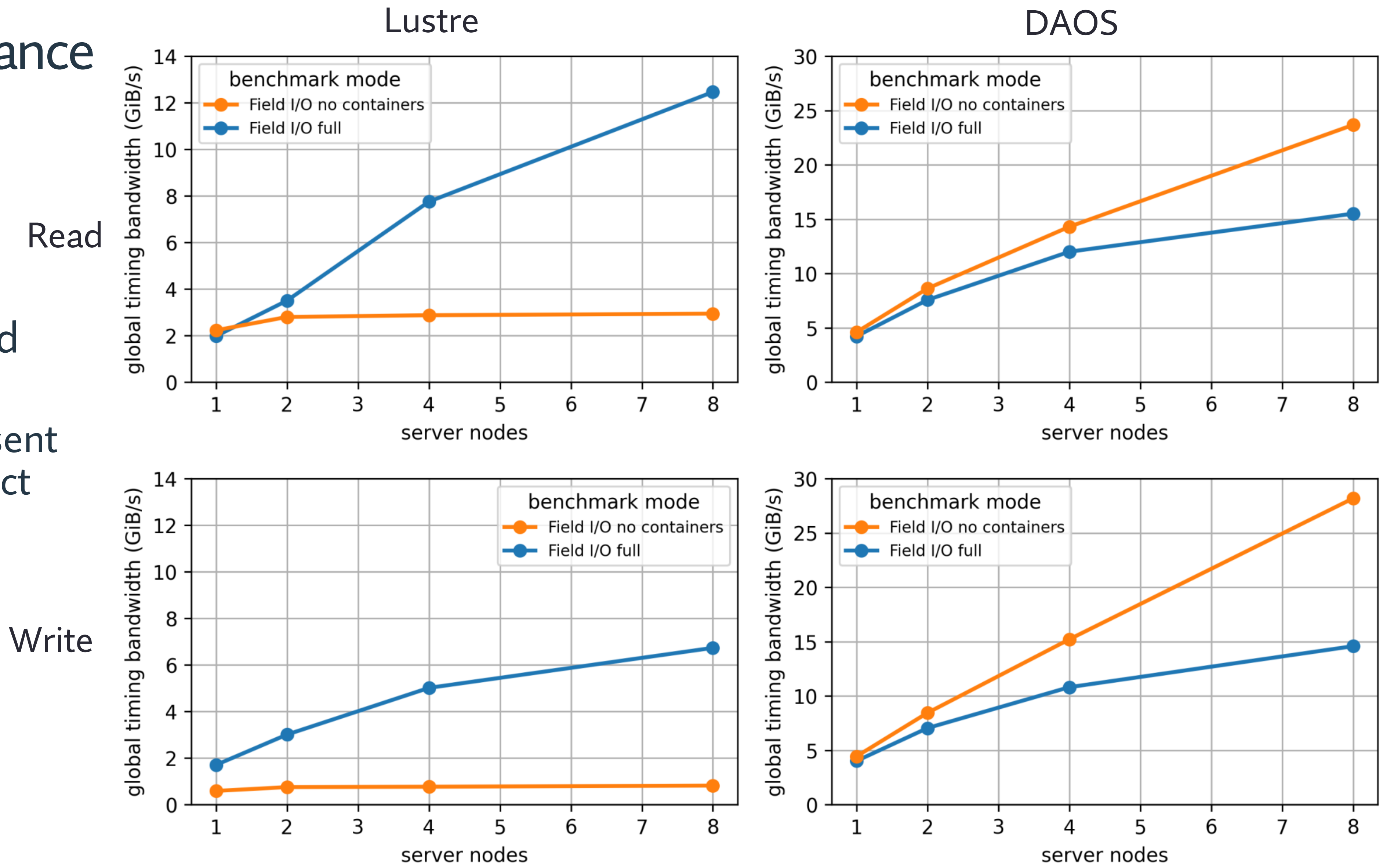
Read

Write



DAOS performance

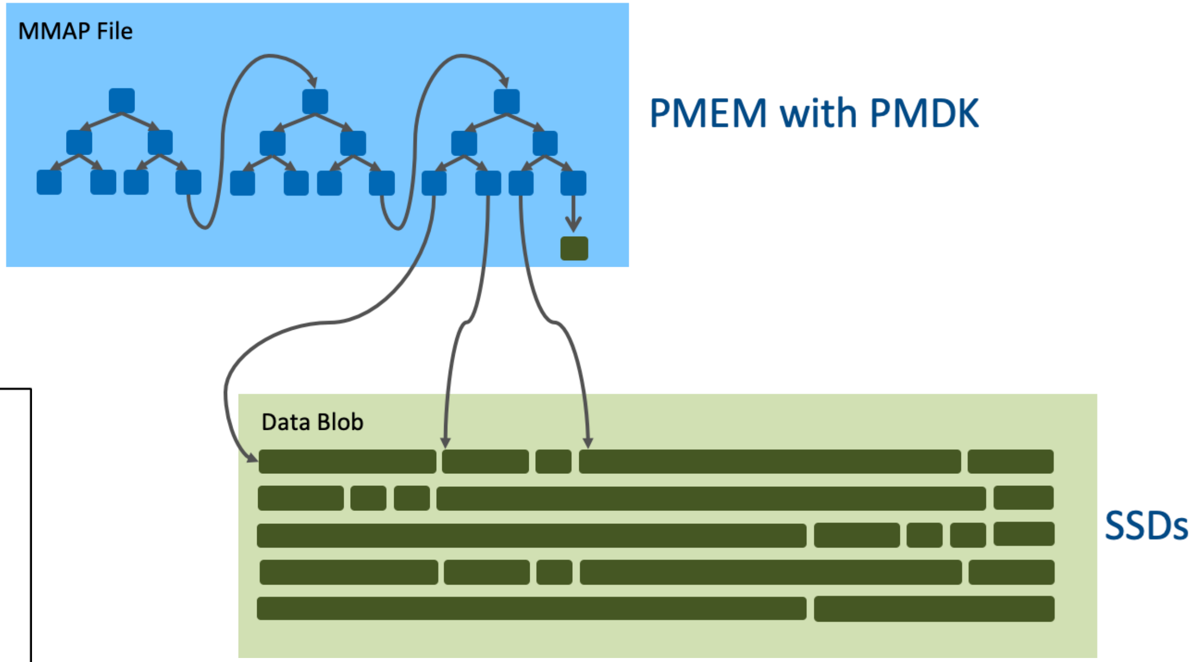
- Contending read and write workers
  - Containers represent filesystem or object store structure



Hewlett Packard Enterprise

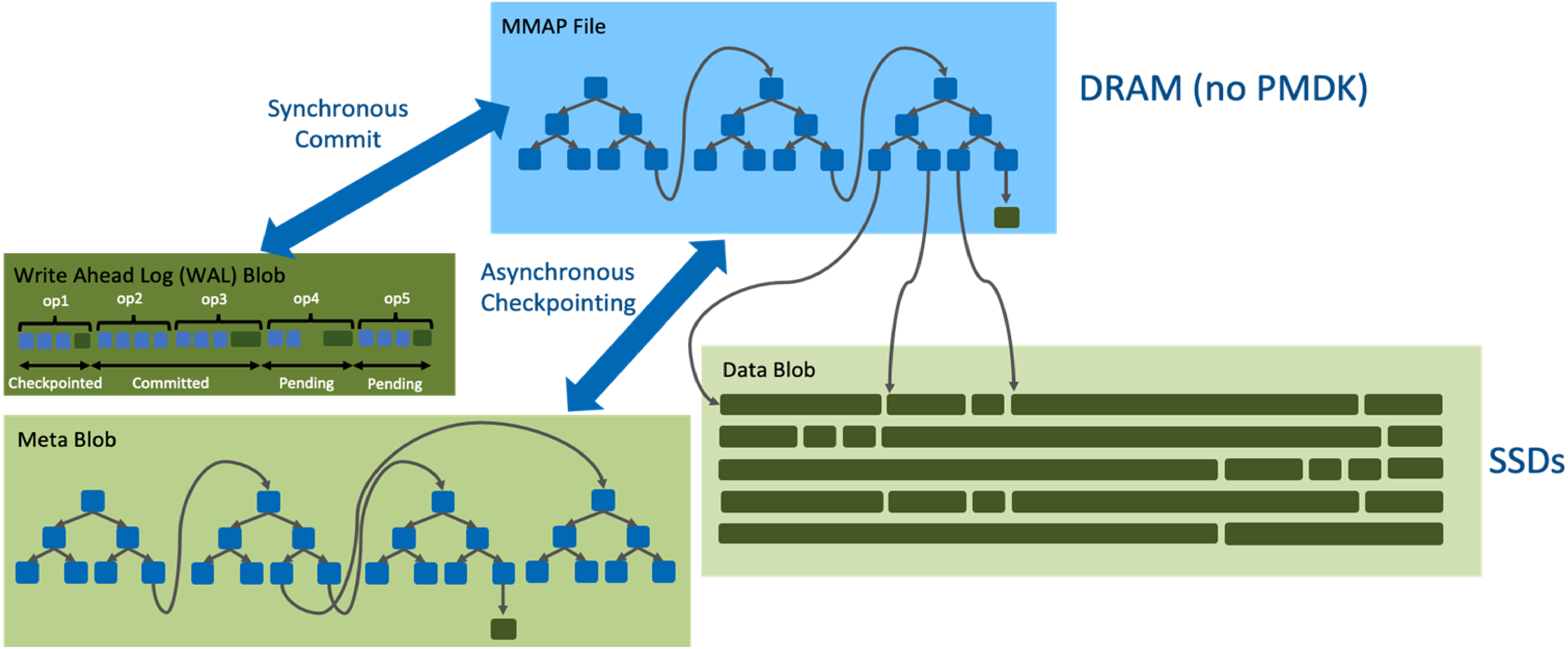
epcc

DAOS beyond NVRAM/Optane/PMem

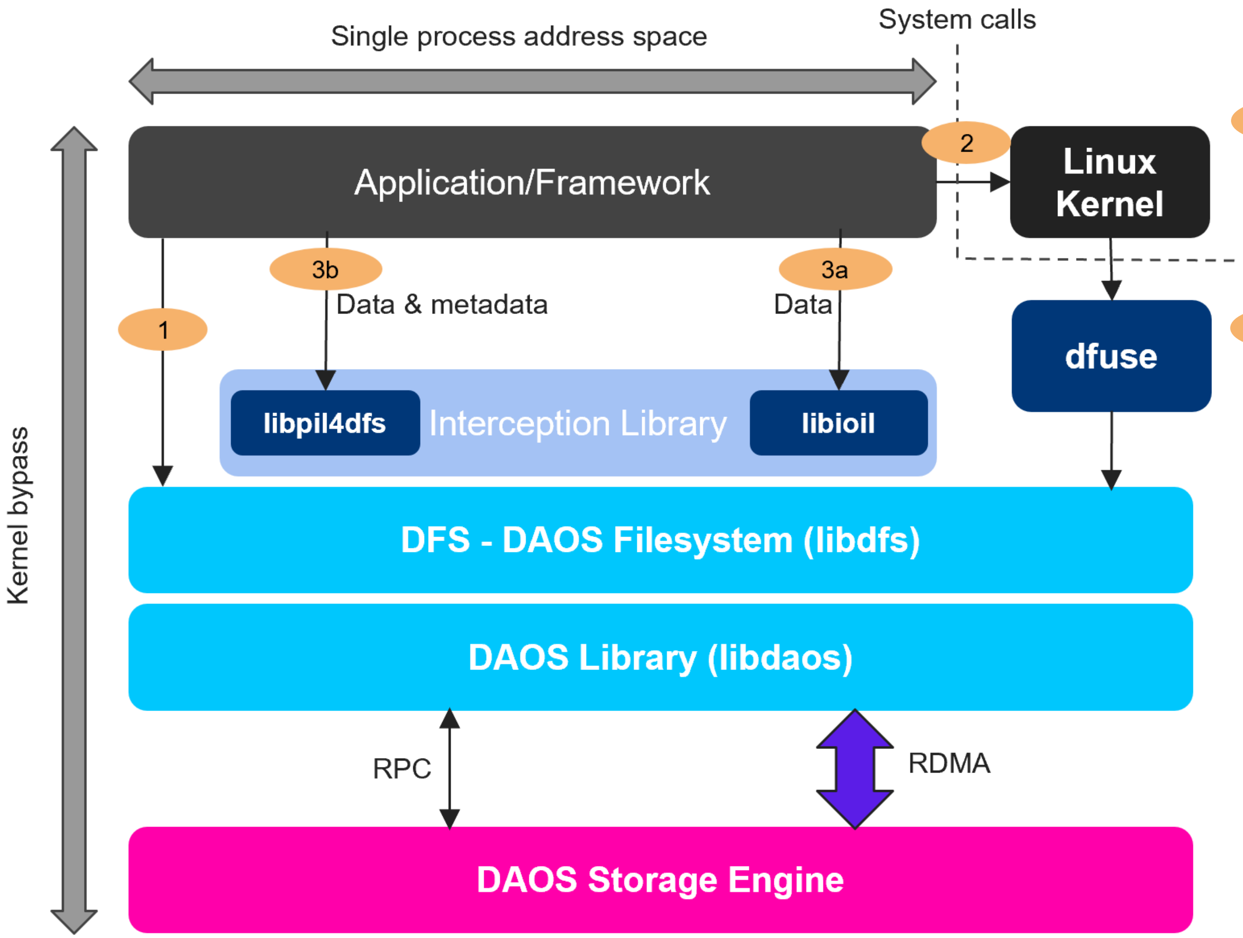


- Persistent metadata
- Require Intel Optane PMEM (or NVDIMM-N)
- App Direct mode
- Mode used on Aurora

DAOS beyond NVRAM/Optane/PMem



DAOS access approaches



1 1. Userspace DFS library with API like POSIX
   - **Require** application changes
   - Low latency & high concurrency
   - No caching

2 2. DFUSE daemon to support POSIX API
   - **No** application changes
   - VFS mount point & high latency
   - Caching by Linux kernel

3. DFUSE + Interception library
   - **No** application changes
   - 2 flavours using LD_PRELOAD
   - 3a libioil
     - (f)read/write interception
     - Metadata via dfuse
   - 3b libpil4dfs
     - Data & metadata interception
     - Aim at delivering same performance as #1 w/o any application change
     - mmap & binary execution via fuse

Hewlett Packard Enterprise

epcc

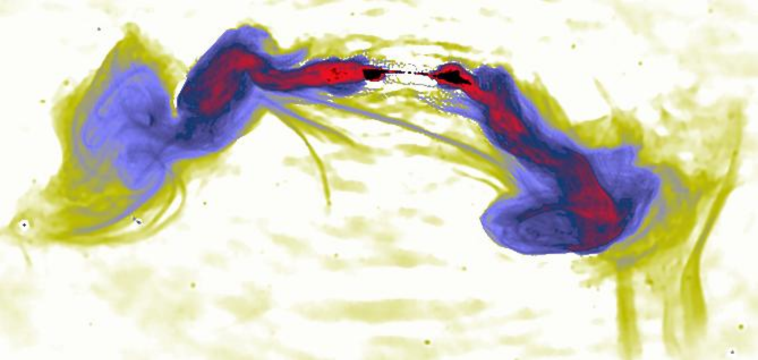
Object stores

- High performance object stores offer:
    - Server-side consistency by default
        - reducing round trip messaging for some operations
    - Distributed metadata functionality
        - no single performance bottleneck
    - Small object size performance
        - non-kernel space I/O operations so don’t have interrupt/context switch performance issues
        - designed for faster hardware and for large scale operation
    - Decoupled metadata from data
    - In-built redundancy control/configuration
    - Multi-versioning and transactions to reduce contention/provide consistency tools
    - Scaling across storage resources
    - Searching/discovery across varying data dimensions

Object stores

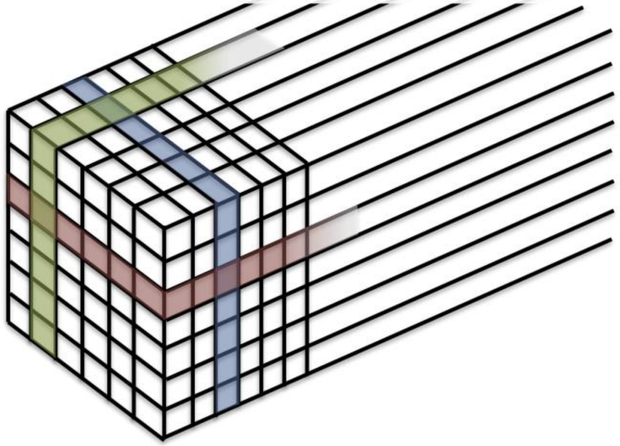
- Object can have as much or as little complexity as you want
  - Single array
  - Single key-value
  - Nested object containing table of entries
  - etc..
- High performance object stores can't....
  - Beat filesystems for bulk I/O with low metadata overheads
  - Support high performance alternative functionality without porting effort
  - Eliminate server side contention
  - Fix poor storage design
  - Create your data layout and indexing for you
  - Fix configuration/resource issues

- Object Stores can unlock previously expensive I/O patterns



- Enable discovery as well as storage

Clients want to do **different** analytics across **multiple** axis



Practical Setup

- https://github.com/adrianjhpc/ObjectStoreTutorial/Exercises/exercisesheet.pdf
- Take IOR source code
- Run on the EPCC system
- SSH details will be sent
- You will get a username
  - ngguestXX
  - Password and key