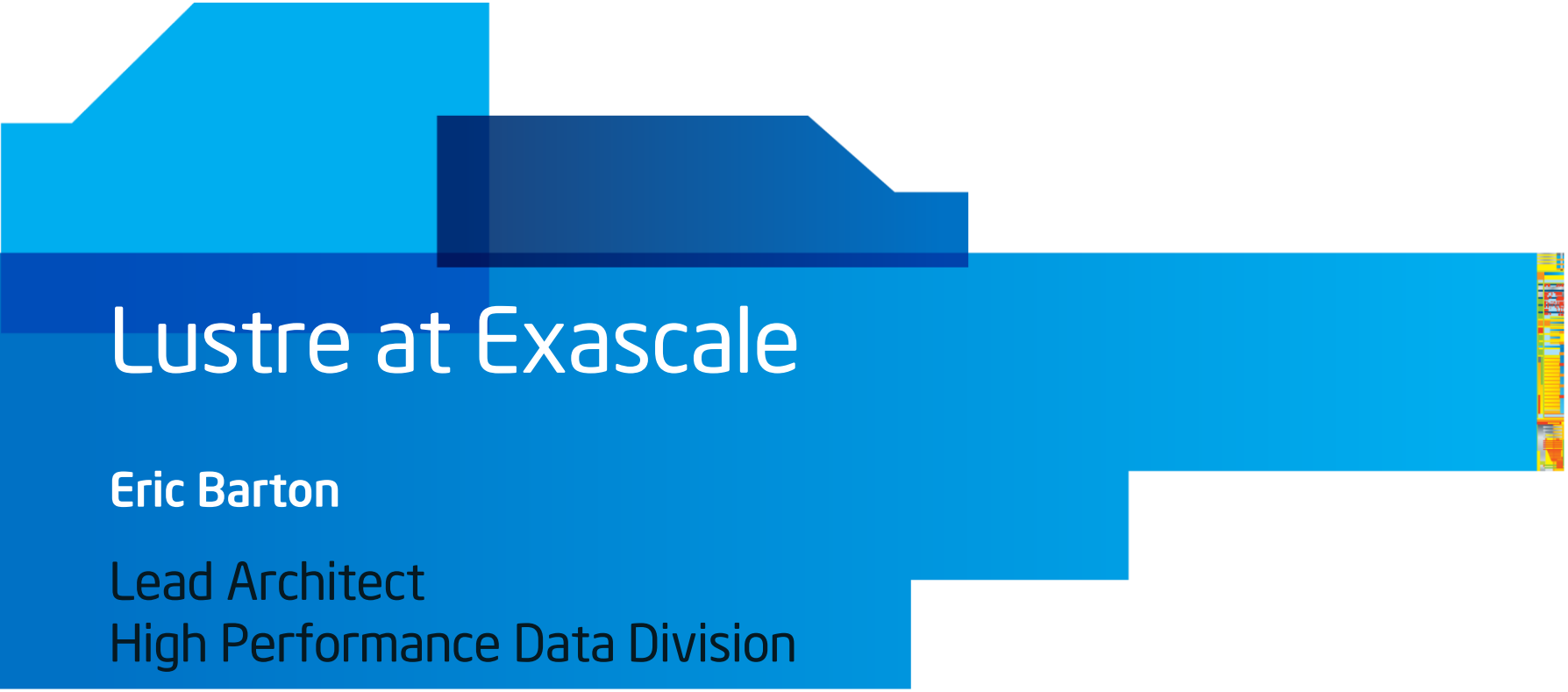


Lustre at Exascale

Eric Barton

Lead Architect
High Performance Data Division

Department of Energy - Fast Forward Challenge

FastForward RFP provided US Government funding for exascale research and development

Sponsored by 7 leading US national labs

Aims to solve the currently intractable problems of Exascale to meet the 2020 goal of an exascale machine

RFP elements were CPU, Memory and Filesystem

Whamcloud won the Filesystem component

- 3 subcontractors – EMC, HDF Group, Cray

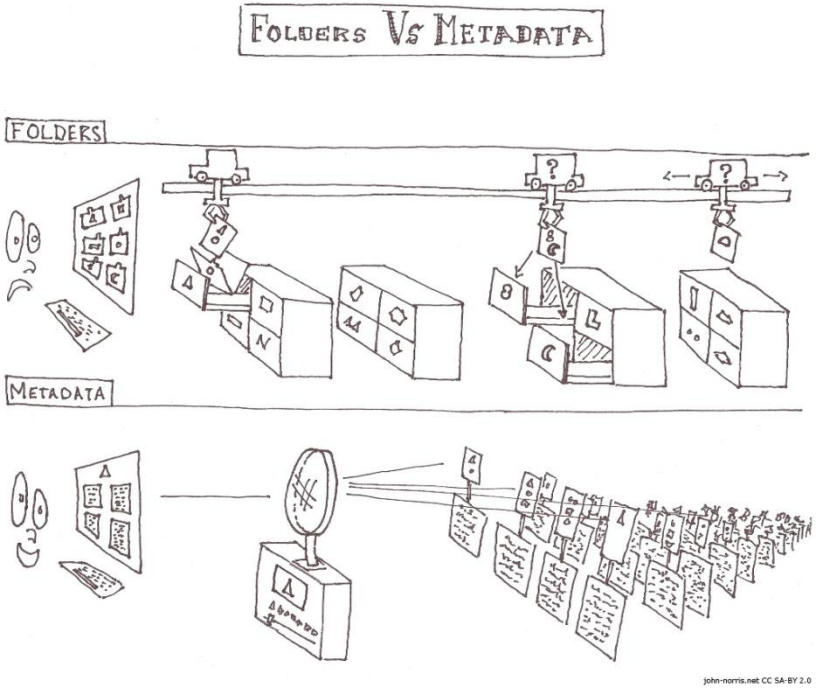
Exascale I/O technology drivers

	2012	2020
Nodes	10-100K	100K-1M
Threads/node	~10	~1000
Total concurrency	100K-1M	100M-1B
Object create	**100K/s**	**100M/s**
Memory	1-4PB	30-60PB
FS Size	10-100PB	600-3000PB
MTTI	1-5 Days	6 Hours
Memory Dump	< 2000s	< 300s
Peak I/O BW	1-2TB/s	100-200TB/s
Sustained I/O BW	**10-200GB/s**	**20TB/s**

Exascale I/O technology drivers

(Meta)data explosion

- Many billions of entities
  - Mesh elements / graph nodes
- Complex relationships
- UQ ensemble runs
  - Data provenance + quality



OODB

- Read/Write -> Instantiate/Persist
- Fast / ad-hoc search: “Where’s the 100 year wave?”
  - Multiple indexes
  - Analysis shipping

Exascale I/O requirements

Constant failures expected at exascale

Filesystem must guarantee data **and** metadata consistency

- Metadata at one level of abstraction is data to the level below

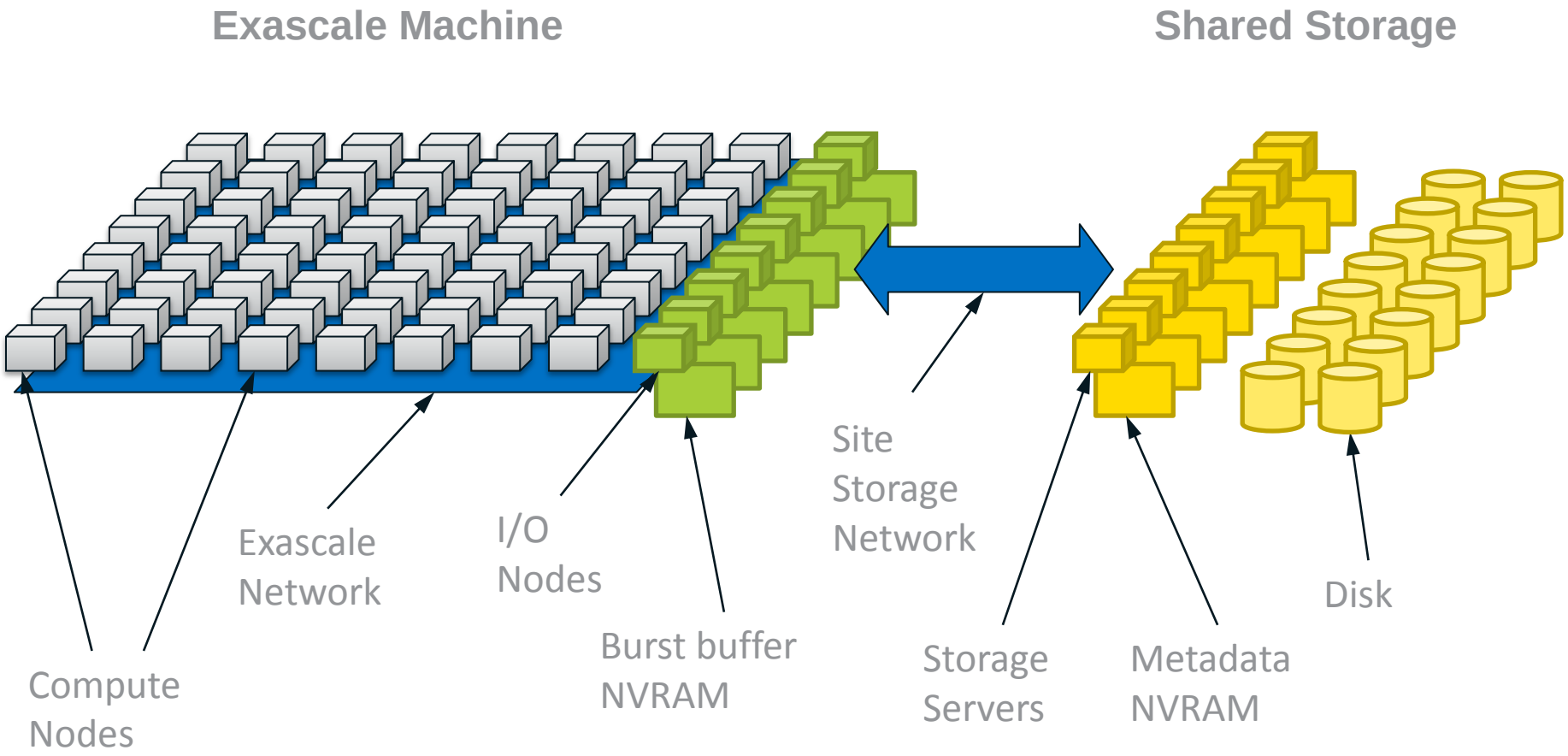
Filesystem must guarantee data integrity

- Required end-to-end

Filesystem must always be available

- Balanced recovery strategies
  - Transactional models
    - Fast cleanup up failure
  - Scrubbing
    - Repair / resource recovery that may take days-weeks

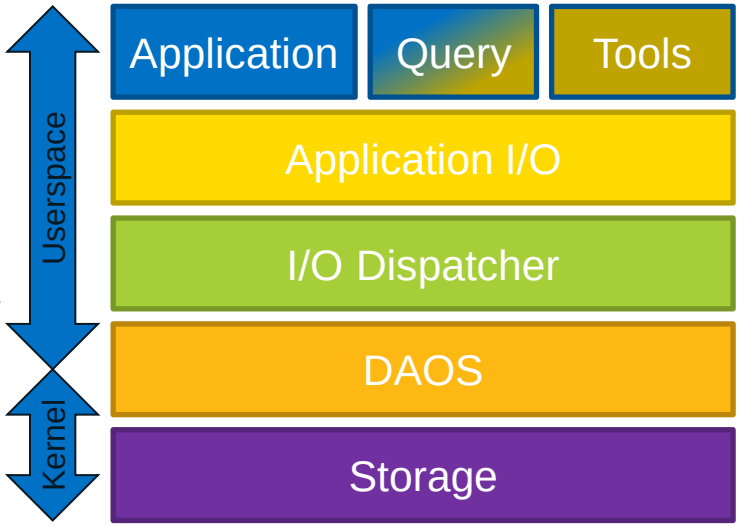
Exascale I/O Architecture



I/O stack

Features & requirements

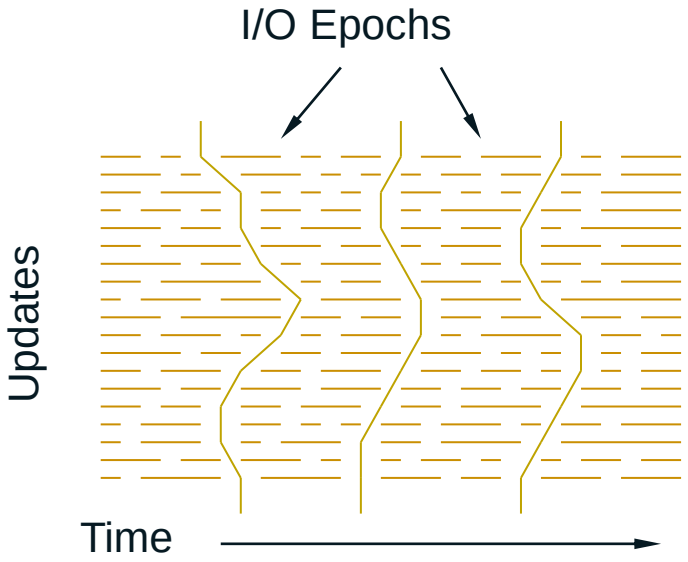
- Non-blocking APIs
  - Asynchronous programming models
- Transactional == consistent thru failure
  - End-to-end application data & metadata integrity
- Low latency / OS bypass
  - Fragmented / Irregular data



Layered Stack

- Application I/O
  - Multiple top-level APIs to support general purpose or application-specific I/O models
- I/O Dispatcher
  - Match conflicting application and storage object models
  - Manage NVRAM burst buffer / cache
- DAOS
  - Scalable, transactional global shared object storage

Transactions



Consistency and Integrity

- Guarantee required on any and all failures
  - Foundational component of system resilience
- Required at all levels of the I/O stack
  - Metadata at one level is data to the level below

No blocking protocols

- Non-blocking on each OSD
- Non-blocking across OSDs

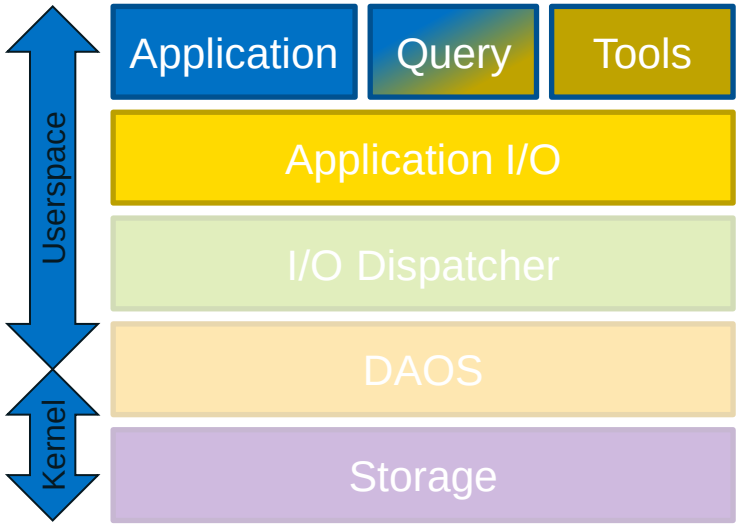
I/O Epochs demark globally consistent snapshots

- Guarantee all updates in one epoch are atomic
- Recovery == roll back to last globally persistent epoch
  - Roll forward using client replay logs for transparent fault handling
- Cull old epochs when next epoch persistent on all OSDs

I/O stack

Applications and tools

- Query, search and analysis
  - Index maintenance
- Data browsers, visualizers, editors
- Analysis shipping
  - Move I/O intensive operations to data



Application I/O

- Non-blocking APIs
- Function shipping CN/ION
- End-to-end application data/metadata integrity
- Domain-specific API styles
  - HDFS, Posix, …
  - OODB, HDF5, …
    - Complex data models

I/O Dispatcher

I/O rate/latency/bandwidth matching

- Burst buffer / prefetch cache
- Absorb peak application load
- Sustain global storage performance

Layout optimization

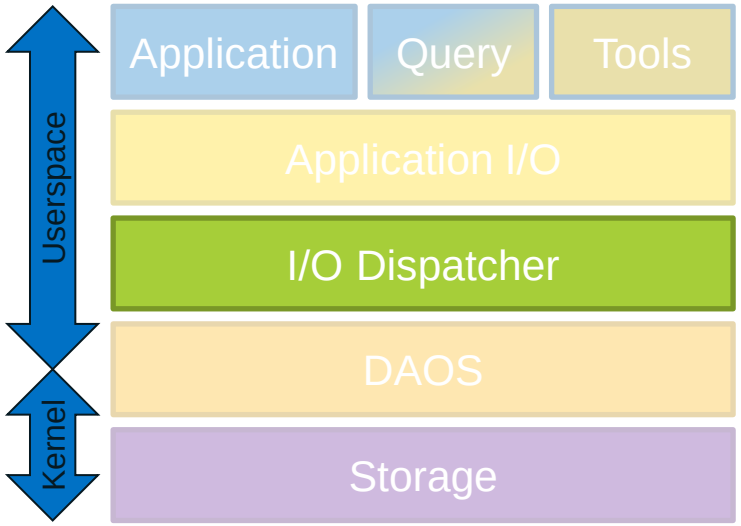
- Application object aggregation / sharding
- Upper layers provide expected usage

Higher-level resilience models

- Exploit redundancy across storage objects

Scheduler integration

- Pre-staging / Post flushing



DAOS Containers

Distributed Application Object Storage

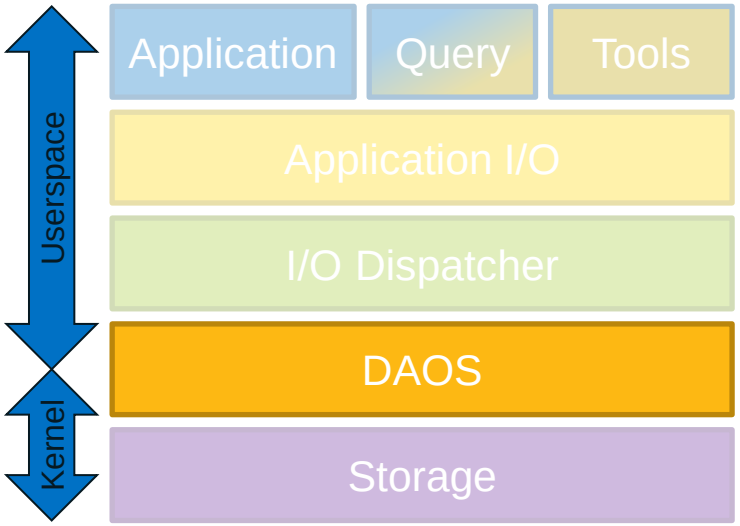
- Sharded transactional object storage
- Virtualizes underlying object storage
- Private object namespace / schema

Share-nothing create/destroy, read/write

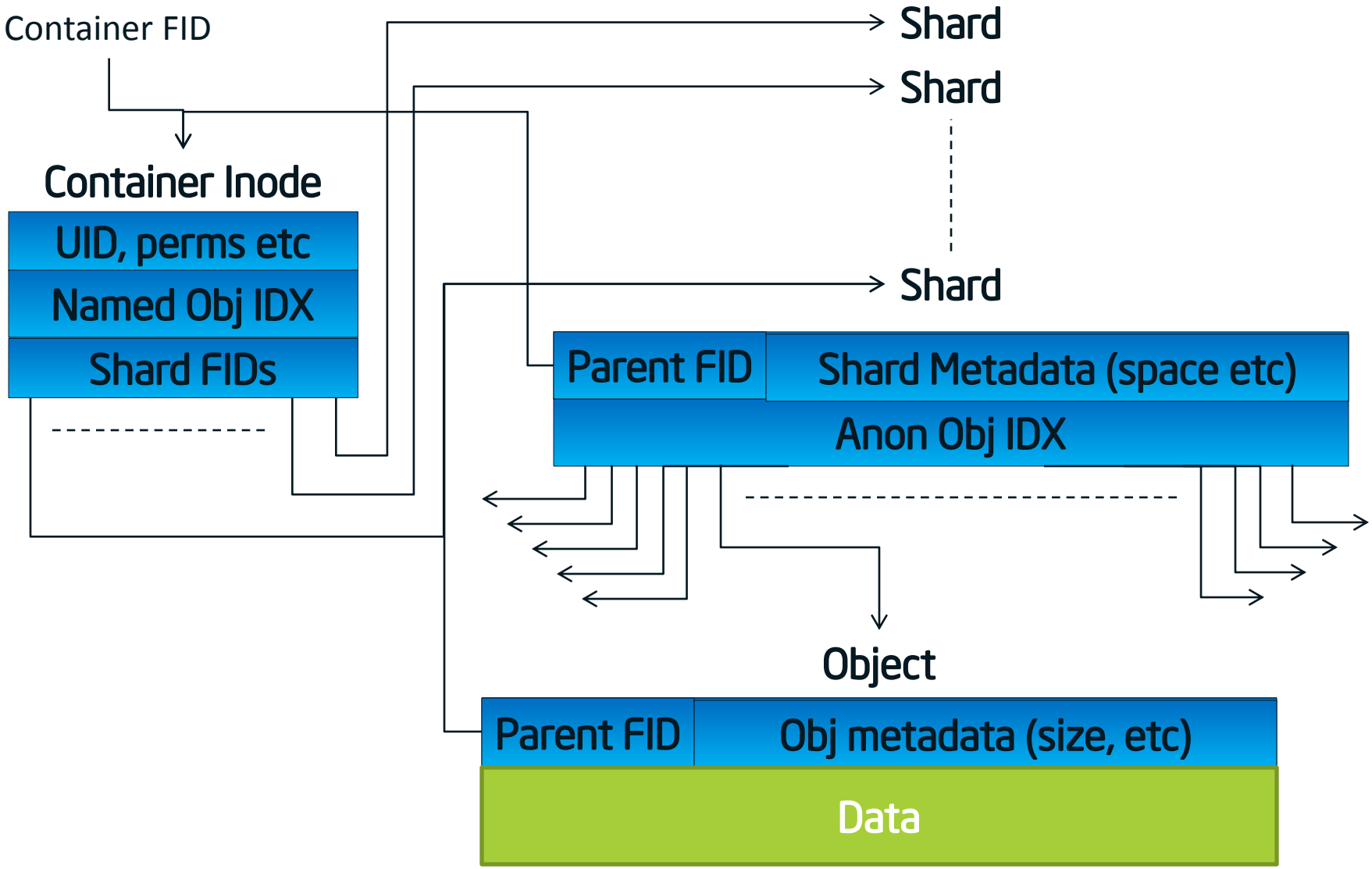
- 10s of billions of objects
- Distributed over thousands of servers
- Accessed by millions of application threads

ACID transactions on objects & containers

- Defined state on any/all combinations of failures
- No scanning on recovery



DAOS Container



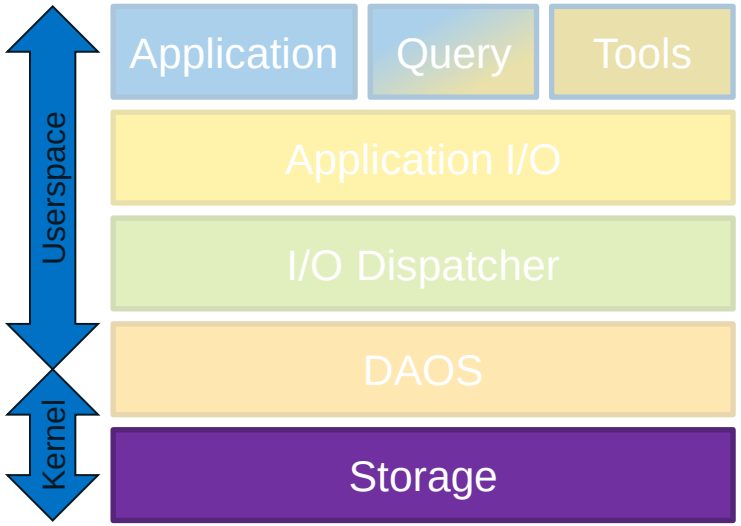
Transactional OSD

DAOS container shards

- Space accounting
- Quota
- Named & anonymous objects

Transactions

- Container shard versioned by epoch
  - Implicit commit
    - Epoch becomes durable when globally persistent
  - Explicit abort
    - Rollback to specific container version
- Out-of-epoch-order updates
- Version metadata aggregation



Server Collectives

Collective client eviction

- Enables non-local/derived attribute caching (e.g. SOM)

Collective client health monitoring

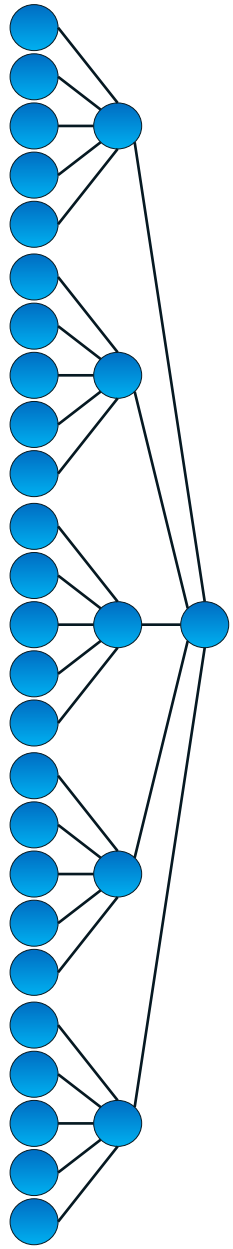
- Avoids “ping” storms

Global epoch persistence

- Enables distributed transactions (SNS)

Spanning Tree

- Scalable O(log n) latency
  - Collectives and notifications
- Discovery & Establishment
  - Gossip protocols
  - Accrual failure detection



Exascale filesystem

Conventional namespace

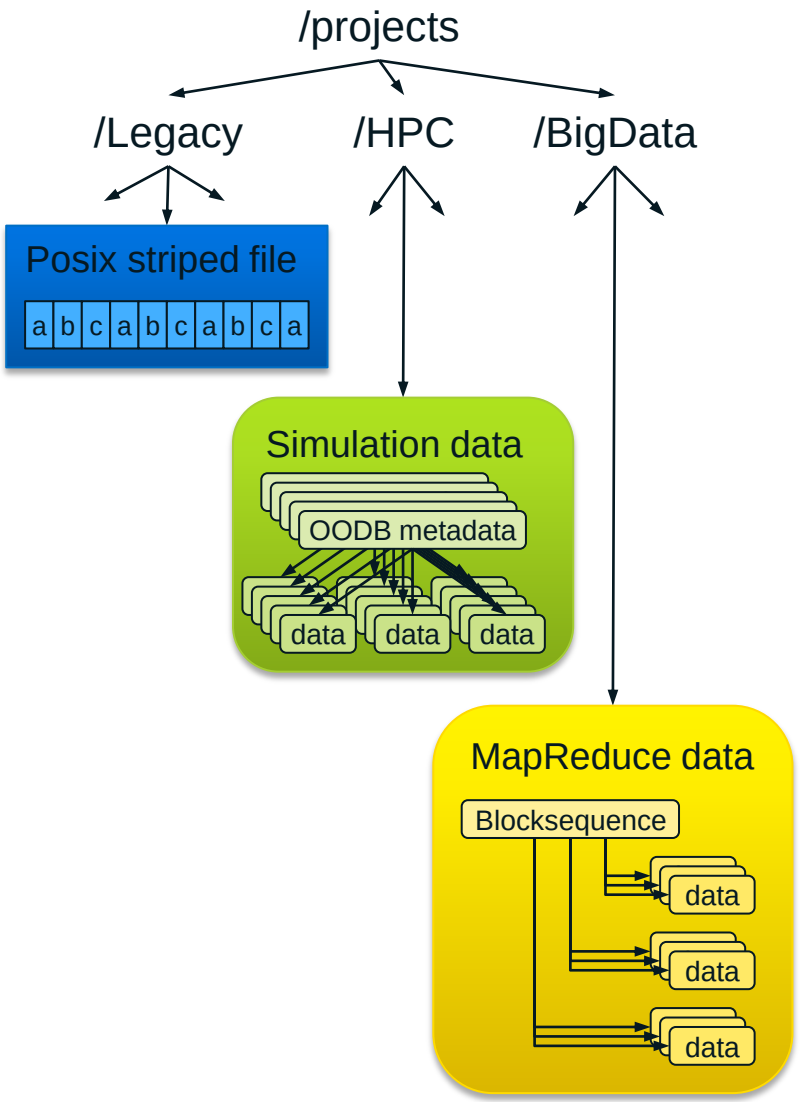
- Works at human scale
- Administration, security & accounting
- Legacy data and applications

DAOS Container files

- Works at exascale
- Application and middleware-specific schemas
- Consistency guaranteed

Storage pools

- Streaming v. IOPS
- Data management
  - Migration between storage pools
  - Rebalance within storage pools
- Pool quota accounting and enforcement



Thank You