EMC Office of the CTO
Fast Data Group

2 TIER STORAGE ARCHITECTURE

SORIN FAIBISH - DE, PRESENTER
JOHN BENT, UDAY GUPTA, DENNIS TING, PERCY TZELNIC
FAST DATA GROUP, CTO OFFICE - EMC

EMC²

Motivation for 2 tiers

Big Data Adds Big Challenges to HPC

- **Performance**: unable to complete analytics in a reasonable period of time
- **Scalability**: data set sizes limit their ability to perform analytics
- **Storage capacity**: need to support multiple uses and accommodate peak usage periods without over-provisioning resources
- **Reliability**: critical
- **Easy deployment and manageability at scale**: no skills to manage large data sets so they derive value

"Big Data Storage: Benefits from the Lessons Learned in High Performance Computing"
ESG, December 2014

Problem Solved by 2 tiers

As The 2nd Platform Evolves Towards The 3rd…

- Storage Disk Arrays become obsolete as:
  - Flash replaces disk for +100x performance (flash array faster than disk array)
  - Cloud replaces disk for +100x capacity (object store scales unlimited)
- Moving older/cold data to cloud is becoming today's reality
  - Cloud approaching $0 for data at rest
  - Capacity disks from arrays move to the cloud, leaving the Array as a Flash only Fast Tier, on-premise
- We can no longer package Performance and Capacity in one Storage Array
  - Split the two, hence 2 Tiers (Fast Tier and Capacity Tier)

➔ *Game Changer!*

How did we get here?

- HPC Burst Buffers and PLFS
- DOE Fast Forward Project
- Emerging Technologies and Evolving Workloads

Burst buffer overview

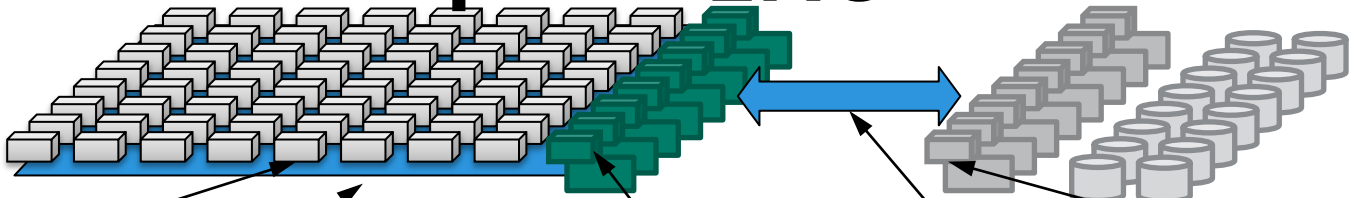
- Fast flash storage tier between CN's and scratch
  - On CN, ION, or storage array
  - Our focus has been on ION
    - Simple, hardware async, aggregation, separation of failure domains, co-processing of data streams
  - But our software allows BB to be anywhere
- Purpose
  - Checkpointing
    - Dump memory in 3-15 mins
    - N-N or N-1 using PLFS
  - Out of core
  - Staging ground
  - Pipelining data between disparate process groups

DOE - Fast Forward Project overview

- Sponsored by 7 leading US national labs
- Exascale goal by 2020
- RFP elements CPU, Memory and Filesystem
- One storage project was selected
  - Intel/Whamcloud, EMC and HDF5
    - With support for Graph analytics ACG application
  - PLFS Burst Buffer – renamed IOD - essential for the project done by EMC
- Completed June 2014

EXASCALE FAST FORWARD ARCHITECTURE

The HDF Group **EMC** **Intel**



Compute Nodes

Interconnect Fabric MPI/Portals

I/O Nodes Burst Buffer

SAN Fabric OFED

Storage Servers

EMC²

IOD: A non-POSIX* interface to persistent data

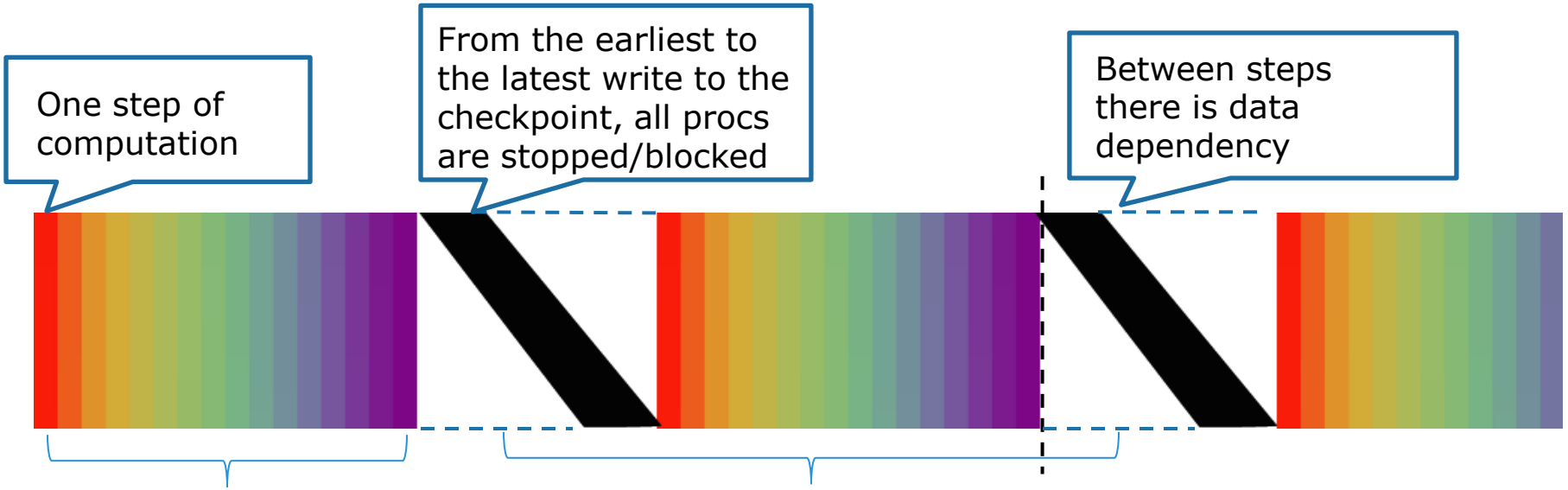
- Burst Buffer space management on IO Nodes (IONs)
  - Persist, purge, pre-stage, reorganize
  - Co-processing analysis on in-transit data
  - Query and reorganize data placement
- Object storage
  - Arrays for semantic storage of multi-dimensional structured data
  - Blobs for traditional sequences of bytes
  - Key-value stores for smaller get/put operations
  - Containers instead of directories
- Transactions: atomic operations across sets of objects
- Asynchronous operations operates in background (optional)
- List IO all the way through the stack
  - Reduce trips across network
  - Everything fully asynchronous with distributed transactions
  - Reads, writes, commits, unlink, etc across sets of objects

* Currently Morphed as basis of 2 tiers architecture

Transactions: Key for FF Project

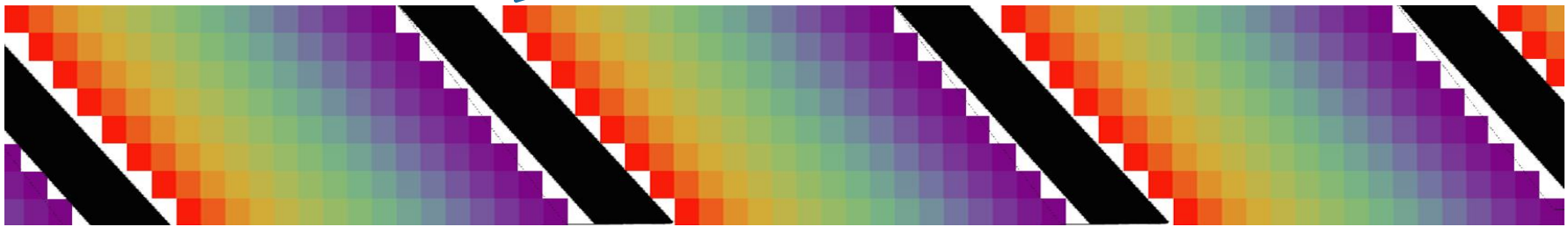
- Group sets of operations across sets of objects into one atomic operation
- Allow versions ("views") for readers
- Lend themselves well to time-series analysis
- Provide eventual consistency for time-skewed distributed processes
- Allow efficient incremental overwrite

TYPICAL MPI IO CHECKPOINT



TRANSACTIONAL CHECKPOINT

When one CN finish writing its part of checkpoint, it can continue computing.

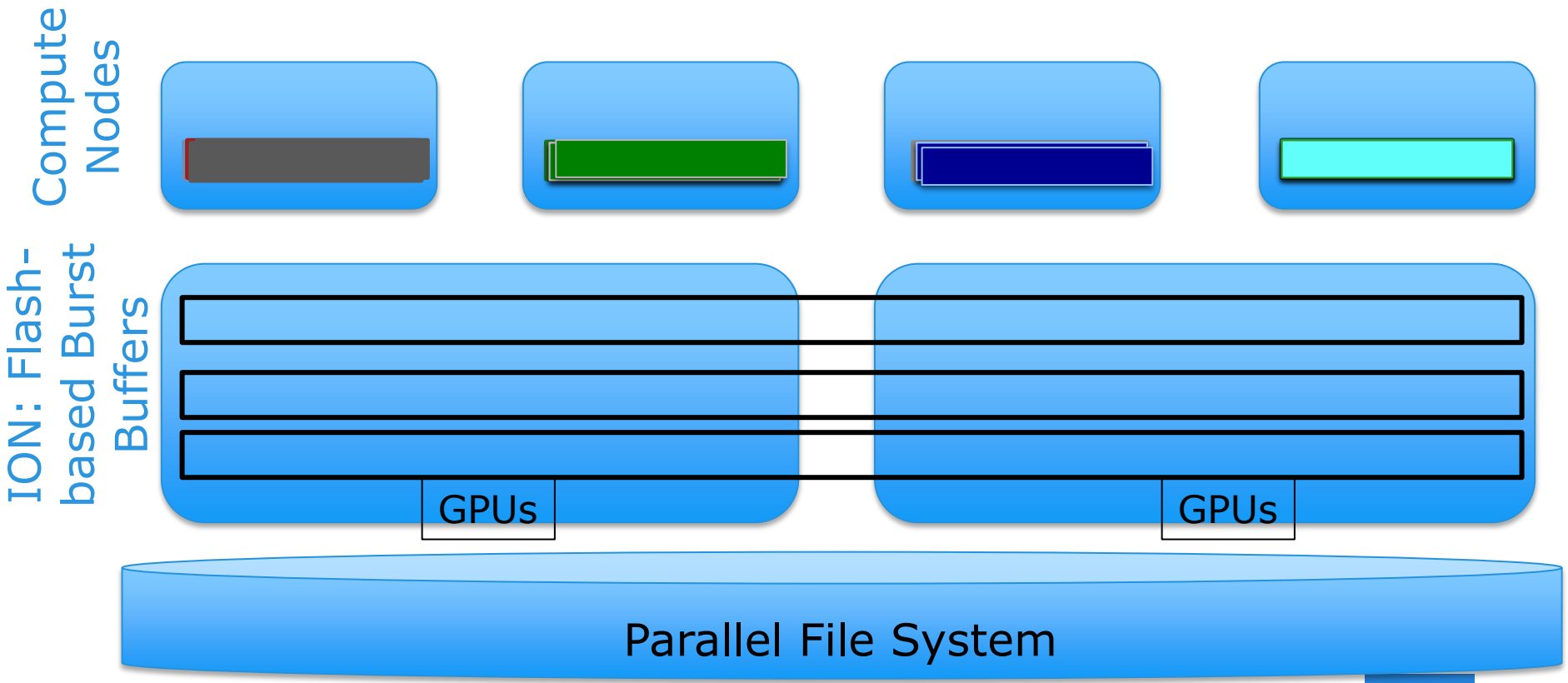


No “idle” triangle.

TRANSACTIONAL COMPARED TO MPI IO

	Transactional	MPI File IO
Open	Independently	Global barrier implied
Write	Simple POSIX like API	Complicated, maybe optimized by shuffling data
Close	Committed independently	Global barrier implied
Read	Readable TID means data is complete	Data integrity ensured by sync() and execution order

DISTRIBUTED ASYNCHRONOUS TRANSACTIONS



THREE IOD OBJECT TYPES

- Blobs
  - Data Files will be stored as such
- Arrays
  - When stored, they are “flattened” into a blob
- KV Stores
  - Namespace metadata may be stored here as well such as value length

EMERGING TECHNOLOGIES

- New PCIe flash arrays
- New flash technology close to memory latency
- New faster interconnect fabrics & RDMA
- Improved HDFS performance
- Myriad of new Open Source Object Stores: Swift, S3
- New Data analytics and other in-memory apps
- New Analytics engines: Spark

EVOLVING WORKLOADS

- More and more storage closer and closer to apps
  - PCIe attached FLASH, FLASH on chip, etc.
- New storage interfaces
  - KV and object store
- Workloads adapting to the above
  - Or forcing it . . .
- Increased market pressures for open source and SDS

EVOLUTION OF HPC LEADS TO EMC 2 TIERS

- File systems aren't scalable!
  - > Object storage are
- Object systems aren't usable!
  - > File system interfaces on top of object stores

- What have we gained?
  - Decoupled storage systems (2Tier)
  - Performance of flash, scalability of object
  - Flexible interfaces
  - Relaxed protocols for applications not needing POSIX semantics
  - Evolutionary on-ramp for slow 2nd platform transformation into 3rd
  - An architecture that allows "Dynamically Loaded Namespaces"

2 Tiers Was Designed For New SW/HW Platforms

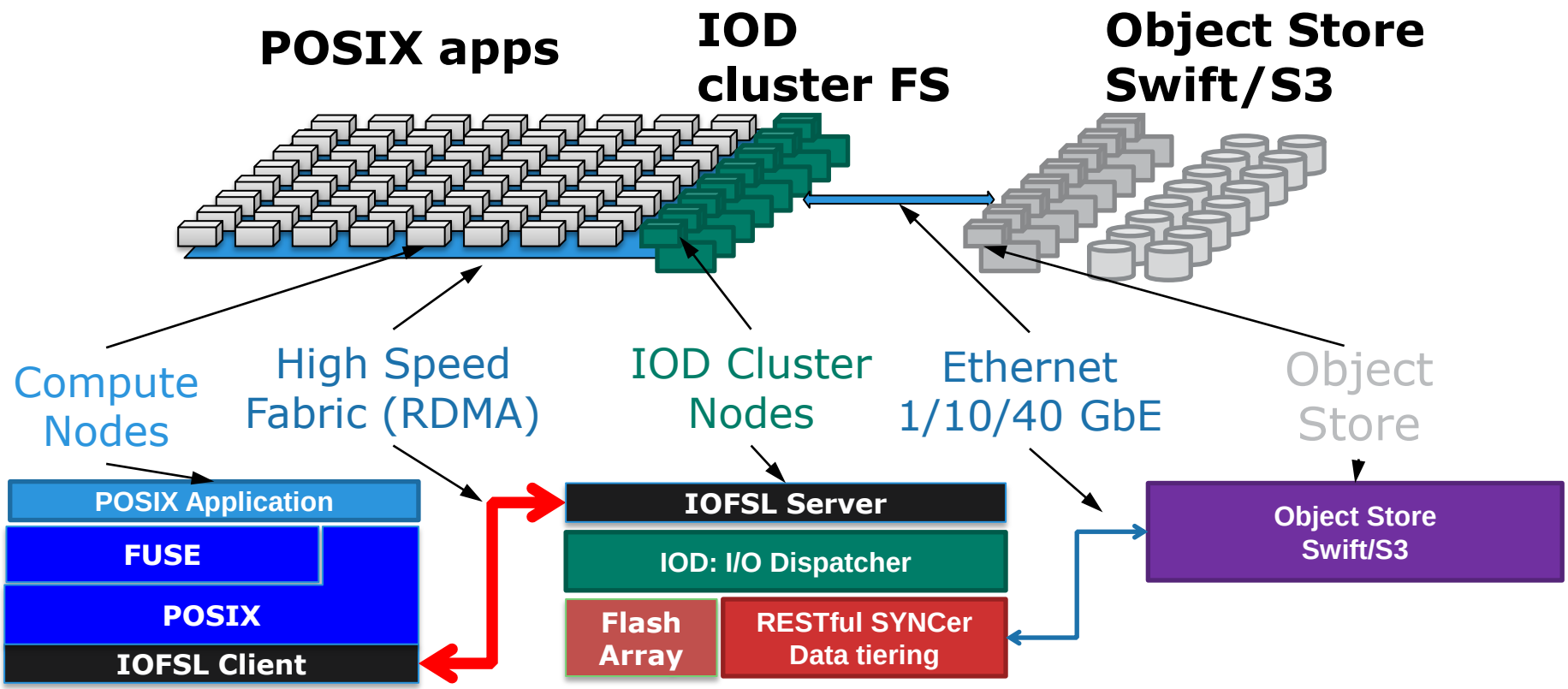
- Disaggregate the monolithic memory / storage / IO Stack and recast it into loosely coupled "Fast Tier" and "Capacity Tier"
- 2 Tiers is **Software Defined Storage**
- 2 Tiers is all about independent Scaling:
  - Scale-out Performance for Fast Tier, O(1,000)
  - Hyperscale Size for Capacity Tier, O(100,000)
- 2 Tiers on New Platforms Infrastructure → Perfect fit for New Big Data Apps
  - Seamless support for legacy POSIX Apps

Behavior Characteristics Of 2 Tiers

Key Features

- Software Defined Storage module running on any HW and Compute Fabric
- Fast Tier provides
  - POSIX API and POSIX semantics for single node jobs (read returns last write() data)
  - POSIX API and NFS semantics for multi-node jobs (read returns last flush() or close() data)
- Each Job runs in its own Dynamically Loadable Namespace
- Both Data and Metadata are tiered; Policy Engine:
  - Write Data and Metadata to the Object Store on Close and Flush
  - Encapsulate the Namespace and Evict it with the Data to the Object Store, on Job Completion (default)
  - Allow user to set the alternative Encapsulation and Eviction Policy
- Fast Tier provides High Availability
- Capacity Tier provides Data Protection Features

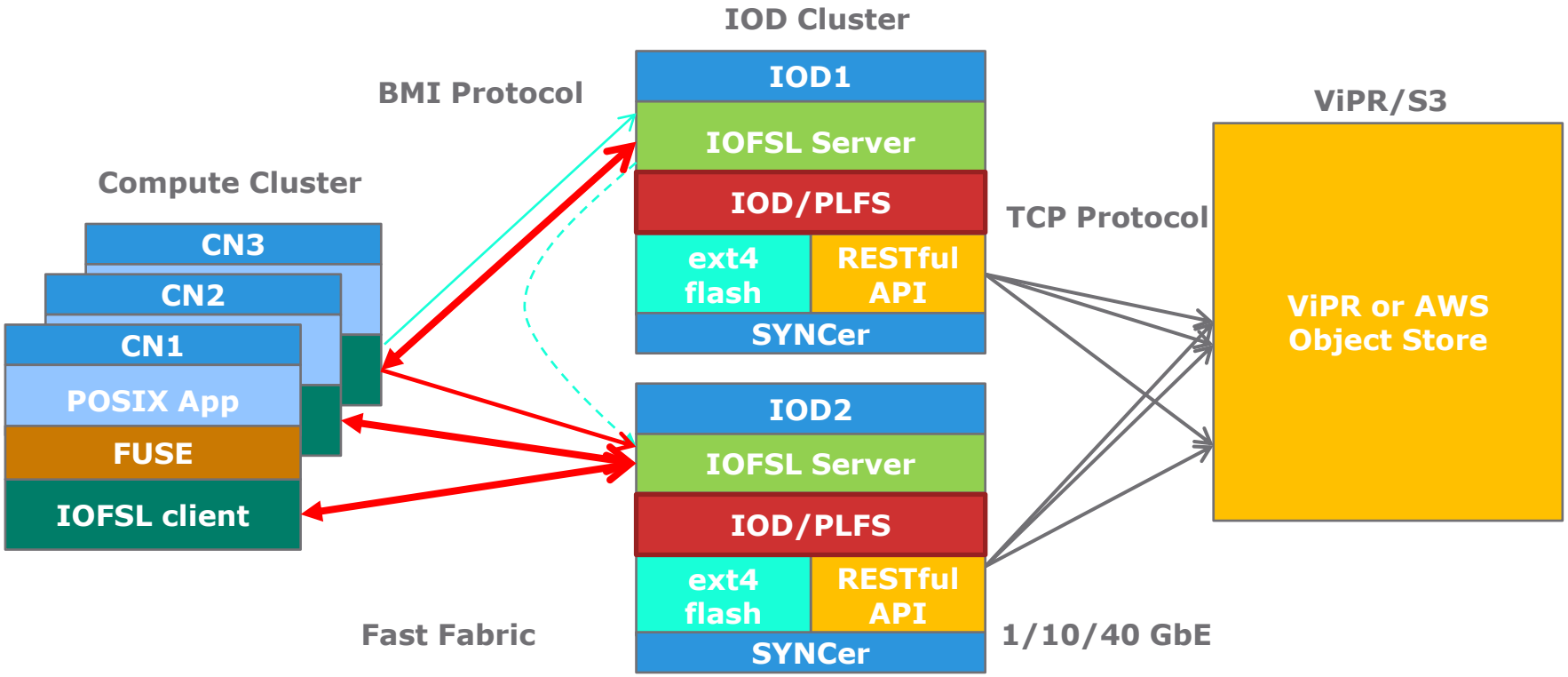
2 TIERS IOD ARCHITECTURE



Main Building Blocks Of 2 Tiers

- I/O Forwarding and Function Shipping
  - IOFSL client with FUSE as POSIX API on the CN's
  - IOFSL server with PLFS and direct IO to Linux local FS
  - Re-direction of IO calls from client to the server that owns the data
  - Use fast RDMA transport protocol for data transfers
  - Manages space on flash on behalf of application
- IOD service module running either on CN's locally or on ION's
  - Provide data and metadata tiering between FT and CT
  - Use containers for maximum availability of service on failures
  - Use MDHIM as KV store for namespace management on FT
  - Support transactions for POSIX semantic
- Dynamically Loadable Namespace Module based on PLFS
  - Loaded by Job Scheduler before job start from CT Object Store to FT Cluster FS
  - Stored as Job Object on CT at end of run and purged from FT
  - Registered in persistent storage Job History table
- Job Scheduling module for data and metadata scheduling and resource allocation

IOFSL IO Forwarding Solution



Changes to IOFSL for 2 Tiers

Performance Optimizations

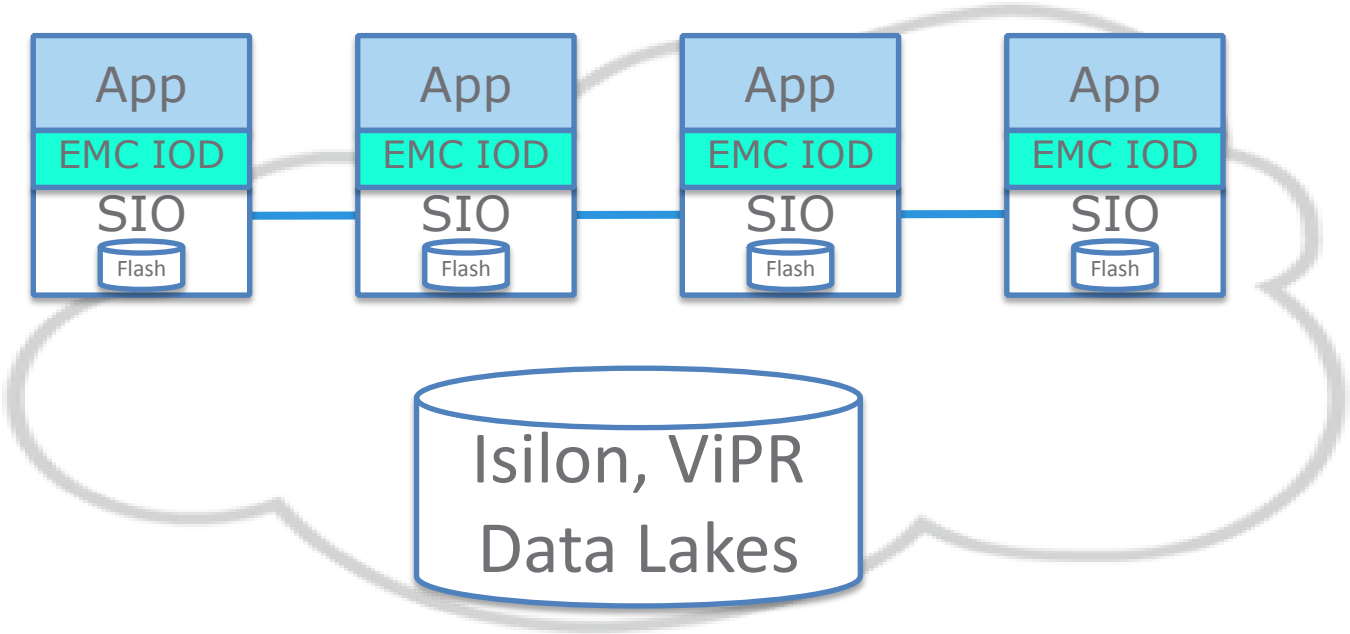
FUSE & IOFSL Client

- Optimized FUSE for adaptive operation for small/large IO's
  - Achieve 98% of native performance
- IOFSL changes
  - RDMA layer optimizations
  - State Machine additions
  - ZOIDFS API for POSIX (~94% compliance)
  - Multi-server support
  - Added purge and data management functions transparent to app
  - Support IOD and KV store API

IOFSL server

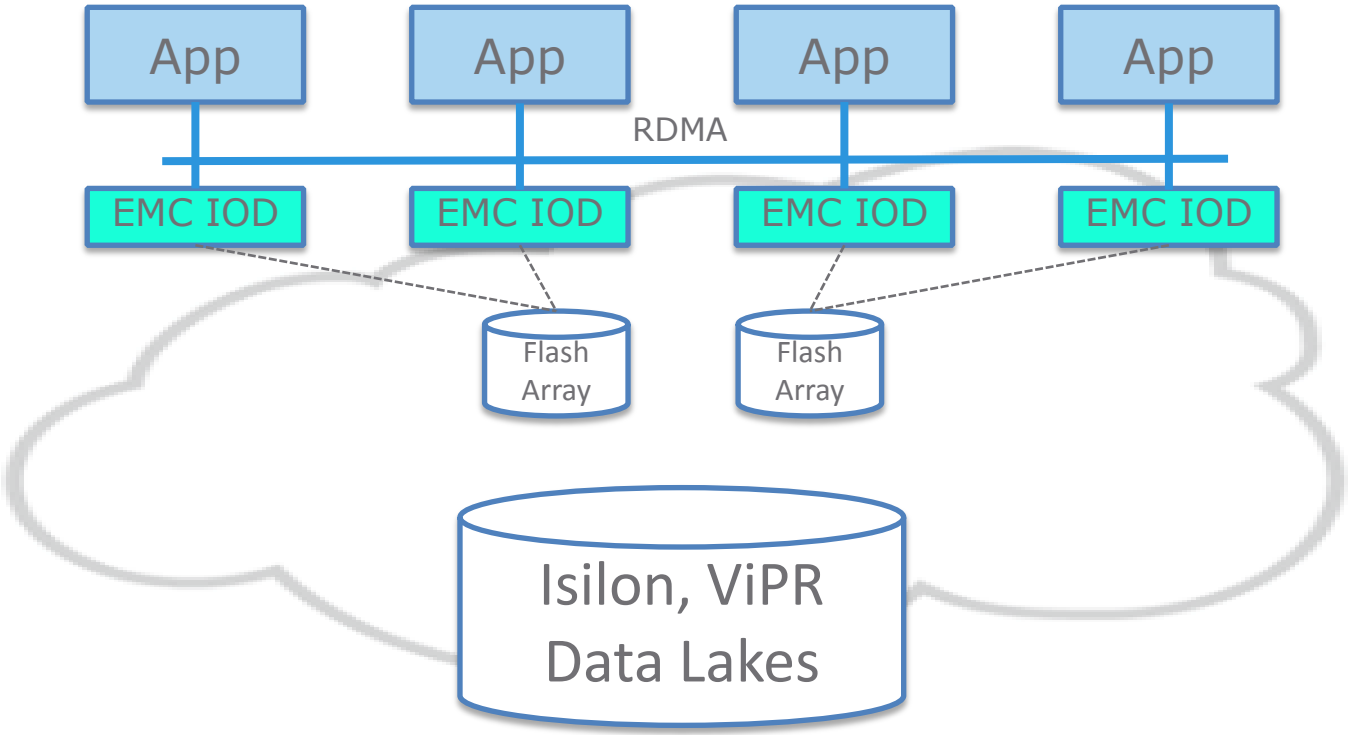
- IOFSL IOstore support for PLFS
- I/O redirect to other IOD nodes
- POSIX Function Shipping
- Pipelining reordering IO request for read consistency

Local Fast Tier SDS Example

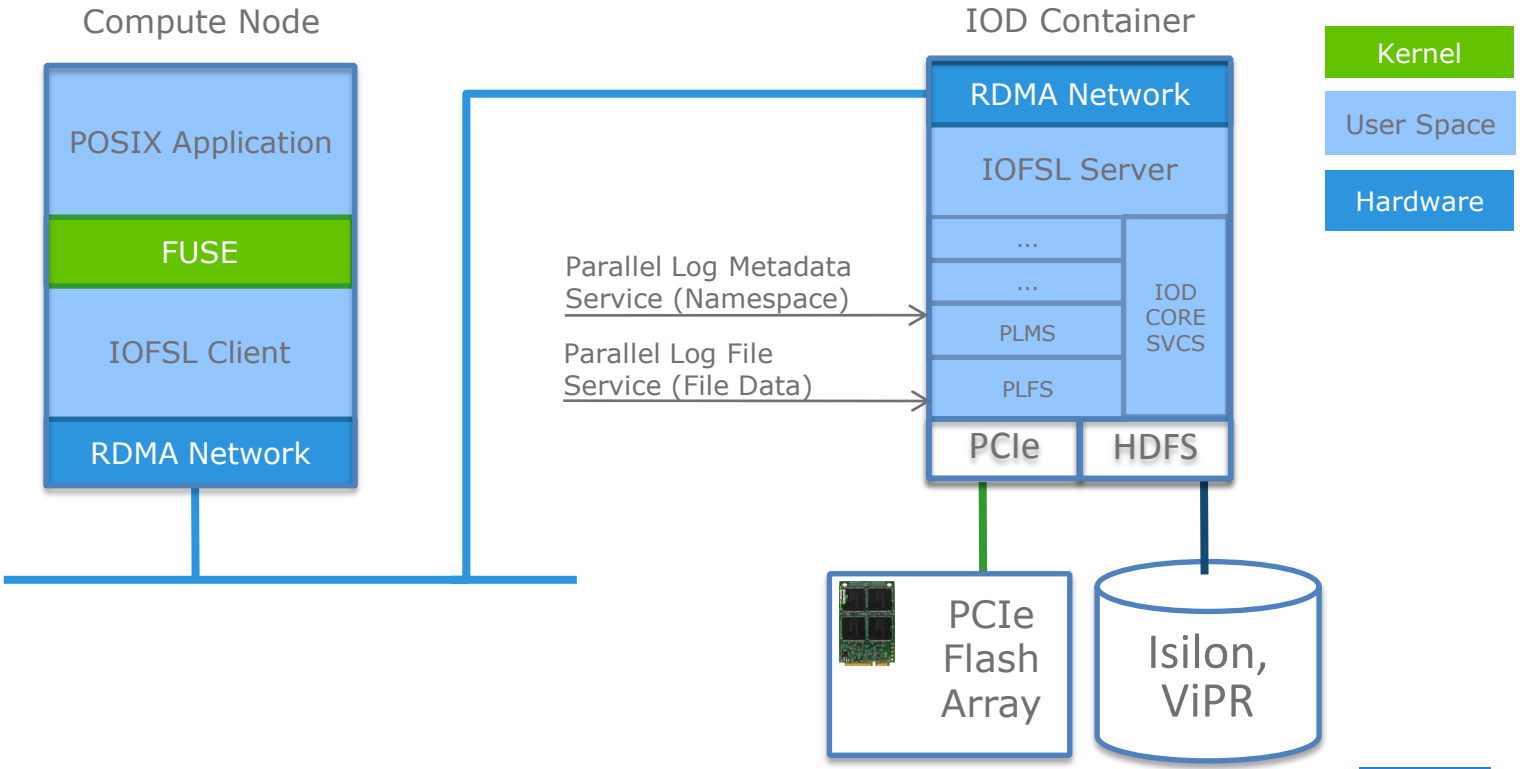


Note: Compute Server interconnect should be RDMA, for best performance

Network Fast Tier SDS Example



EMC IOD Software Layers - POC



Dynamically Loaded Namespace

- Describe Metadata Only!
- Allow FT is not cluttered by Metadata for all Data stored in CT (unlike HSM solutions)
- Show Two Different Options for Splitting SN into DLN's
- Maintain a single gigantic namespace on capacity tier
- "Page" sub-trees into fast tier as required for application working sets
- Paging means between storage tiers; not between memory and storage

How dynamically loaded namespace

IndexFS/BatchFS pioneered techniques!

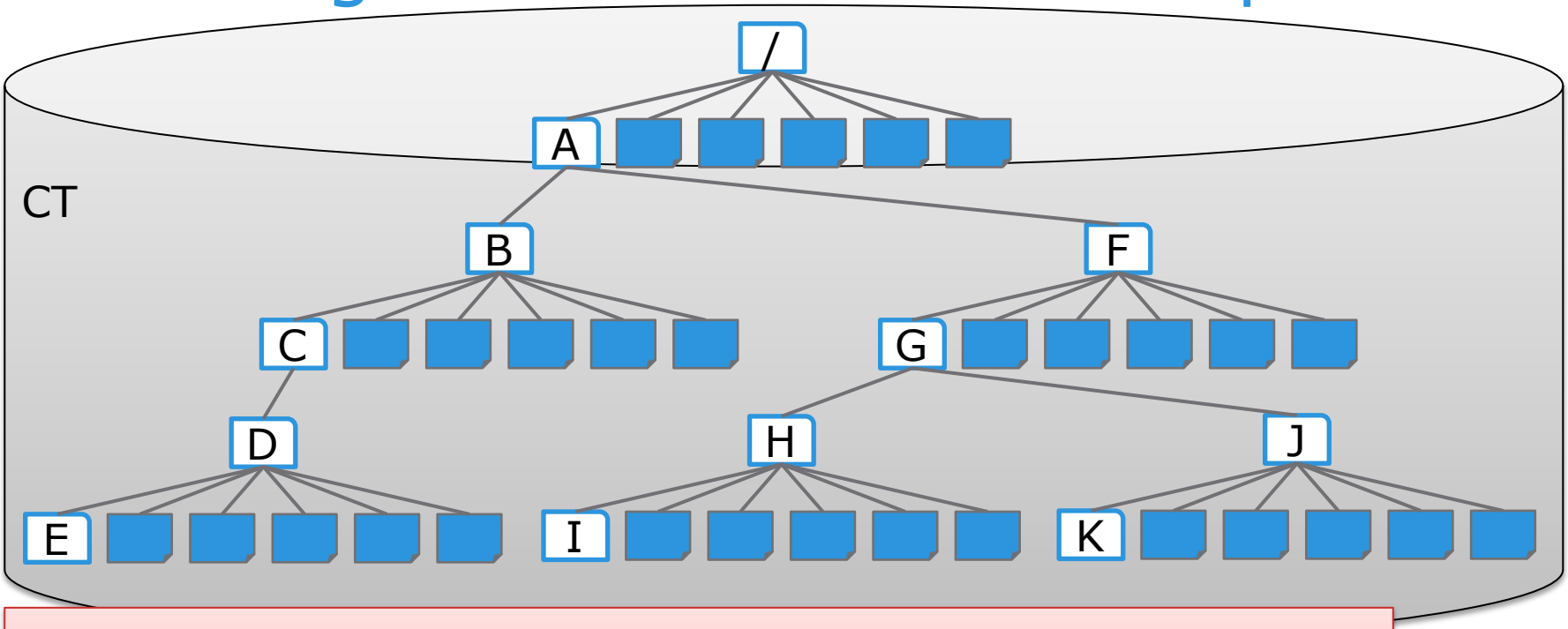
SSTable Optimization

```
function import_dln(char *path) {
  foreach dln in ct->indexfs->path_to_dlns(path) {
    dln = ct->read(dln)
    ft->write(dln)
    ft->indexfs->insert_dln(dln)
  }
}
```

Filesystem Tiering

```
function import_dln(char *path) {
  for dir in ct->traverse_path(path) {
    ft->mkdir(dir)
    for file in ct->readdir(dir) {
      copy_file(file,ct,ft)
  }
}
```

Logical View of the Namespace

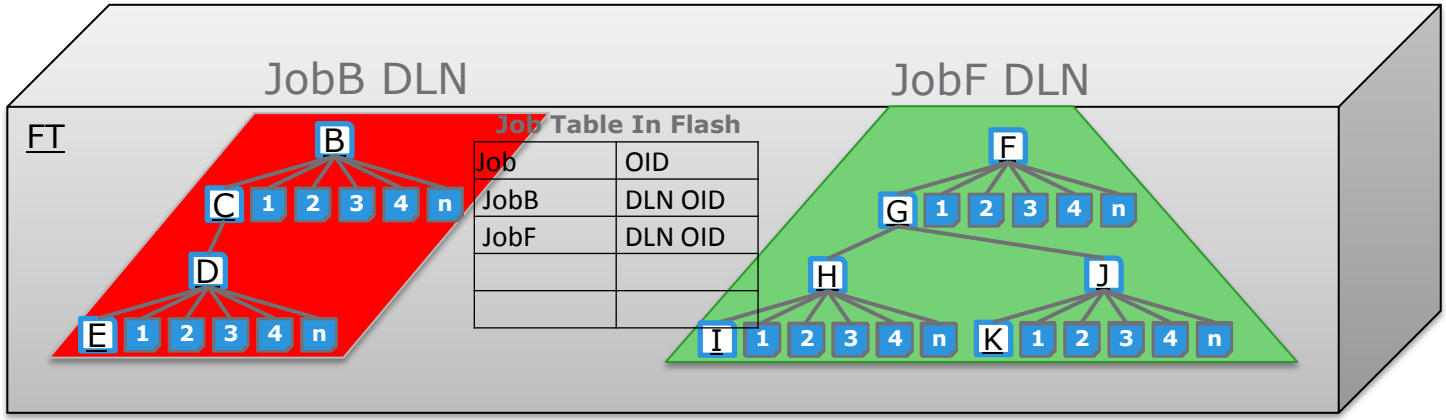


How is it physically stored?

1. In flattened compressed objects that IOD serializes/pickles?
2. In a KV store like MDHIM/LevelDB/ShashwatKV?
3. In one or more actual FS's like OneFS/ext.n/zfs/xfs?

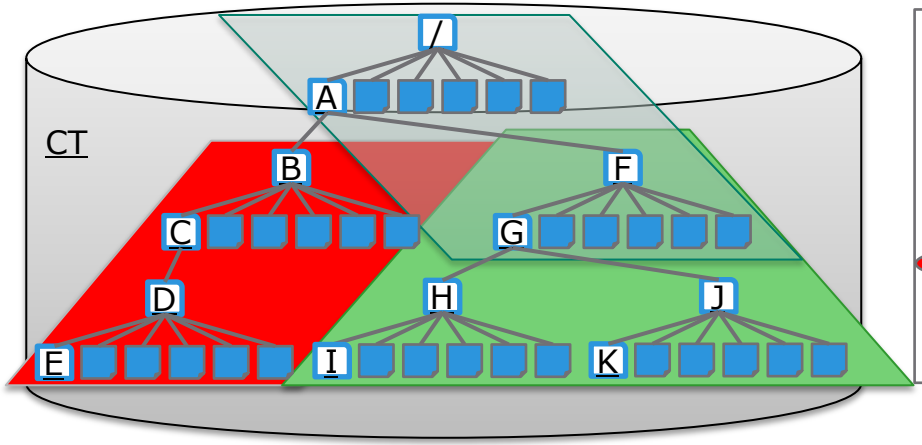
EMC²

DLN's into Fast Tier

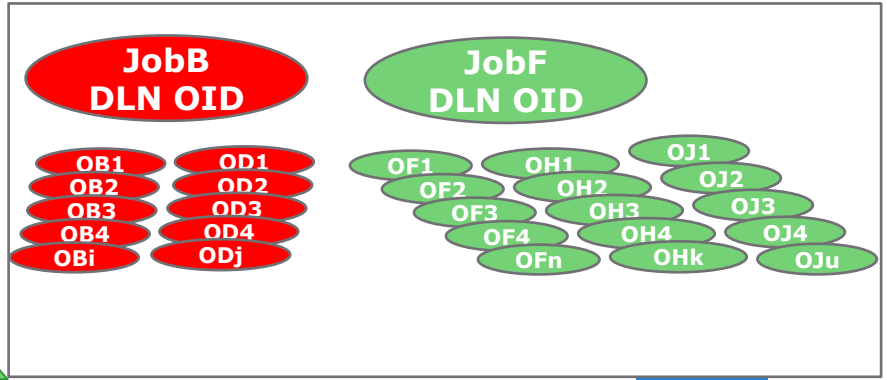


Fast Tier
Expanded
FS View

Capacity Tier Logical NS View



Object Store View of DLN objects

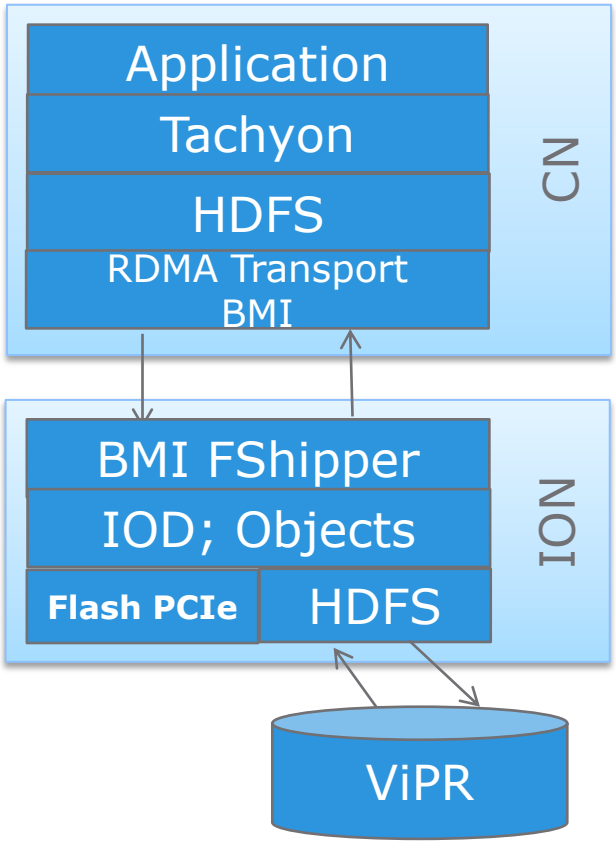


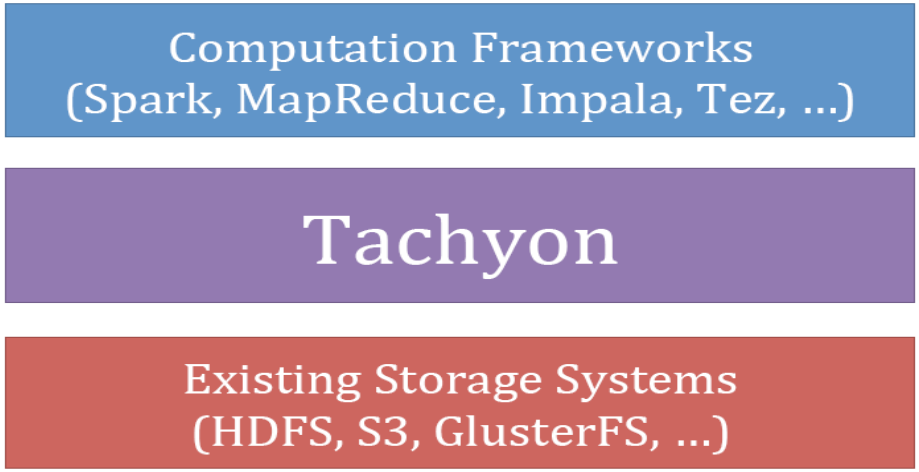
COMBINING TACHYON & 2 TIERS IOD

FUTURE RESEARCH

- Tachyon is already a Tiering file system
  - It attempts to serve the file system from memory
  - Tiering with a disk file system as necessary
- EMC IOD is already a Tiering file system
  - It attempts to serve the file system from Flash
  - Tiering with a cloud object store/ViPR as necessary
- Main research opportunities
  - Extend Tachyon memory with IOD on FT
    - Tachyon no longer tiers at all. IOD does Tiering as directed by scheduler.
  - Replace Tachyon interface to file system with IOD/HDFS
    - This creates three tiers: memory, flash, object
  - Add IOD transactional IO to Tachyon

Spark-Tachyon-HDFS/RDMA





IOD will control data movement. App will run unchanged like on a "faster" HDFS.
See HDFS to Orange FS JNI (Clemson University).

FUTURE RESEARCH WORK

- IO Forwarding between ION/IOD
- MDHIM extension and performance optimizations for namespace and ACL
- PLFS modes complete implementation
  - ✓ Small files read and POSIX operations completion
  - ✓ Extension for flat file support (N-N) for non-MPI apps
  - ✓ Merge all the modes under a single umbrella of PLFS
- Memory management for IOD using multi-tiered memory architectures (Intel DDR4)
- Implementation of full POSIX support for Burst Buffers using IOFSL server semantics
- Integration of IndexFS with 2 tiers (unimplemented functions and PLFS small files)
- Selection and Integration with Job Scheduler SW


EMC²