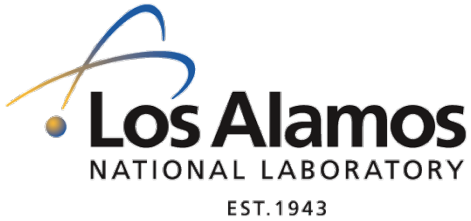
Abstract

The US DOE Office of Science and National Nuclear Security Administration Exascale activities leading to Exascale class computing in the next decade involves a number of initiatives including the Fast Forward industry technology concepts funding activity. The current Exascale activities and coordination mechanisms for those activities will be explained including the Fast Forward initiative. Also, the Storage and IO Fast Forward project will be described, including schedules, project management, and technical aspects.

DOE Fast Forward Storage and IO Project



Gary Grider Los Alamos National Laboratory

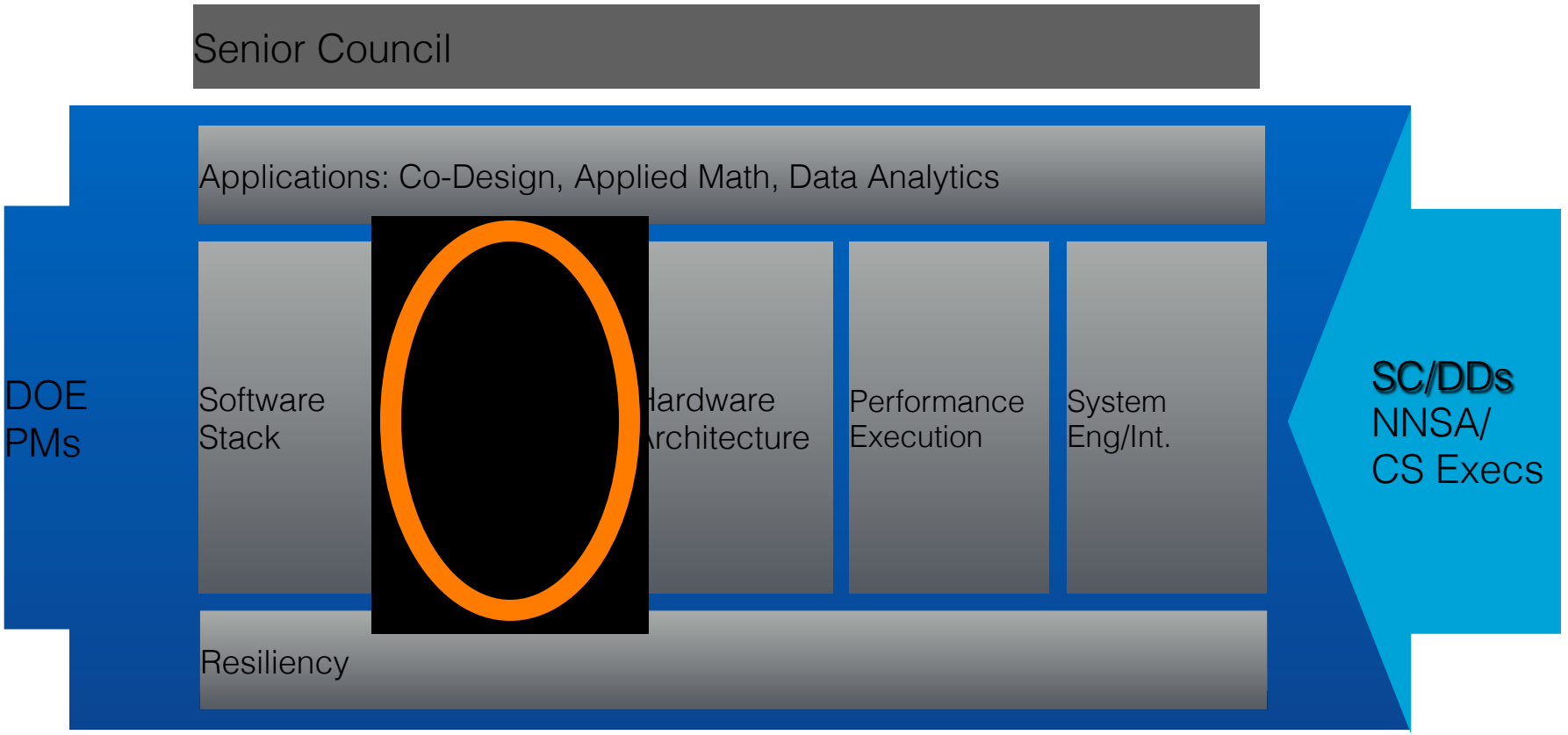
04/2013

LA-UR-13-22139

DOE Exascale Computing

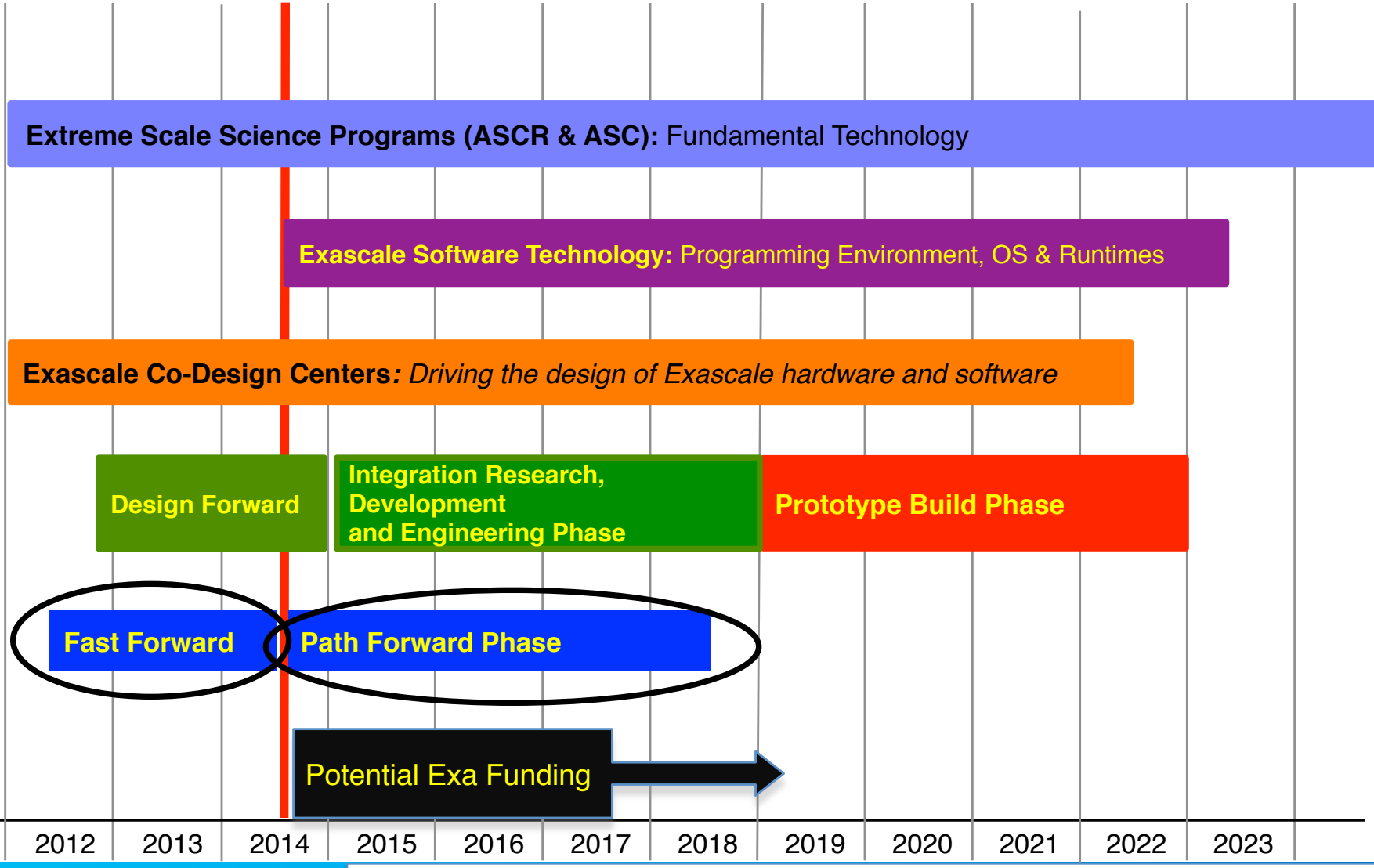
- Perform research, development and integration required to deploy Exascale computers in 2020+
- Partnership involving:
  - Government
  - Computer industry
  - DOE laboratories
  - Academia
- Target System Characteristics
  - 1,000 times more **performance** than a Petaflops system
  - **1 Billion** degrees of concurrency
  - **20 MW** Power requirement
  - **200** cabinets
  - **Development and execution time productivity improvements**
- The Exascale timeframe will vary with funding profiles

Current DOE Exascale Coordination



- DOE National Nuclear Security Administration NNSA/ASC program – LANL, LLNL, Sandia
- DOE Office of Science SC/ASCR program – PNNL, ORNL, LBNL, ANL

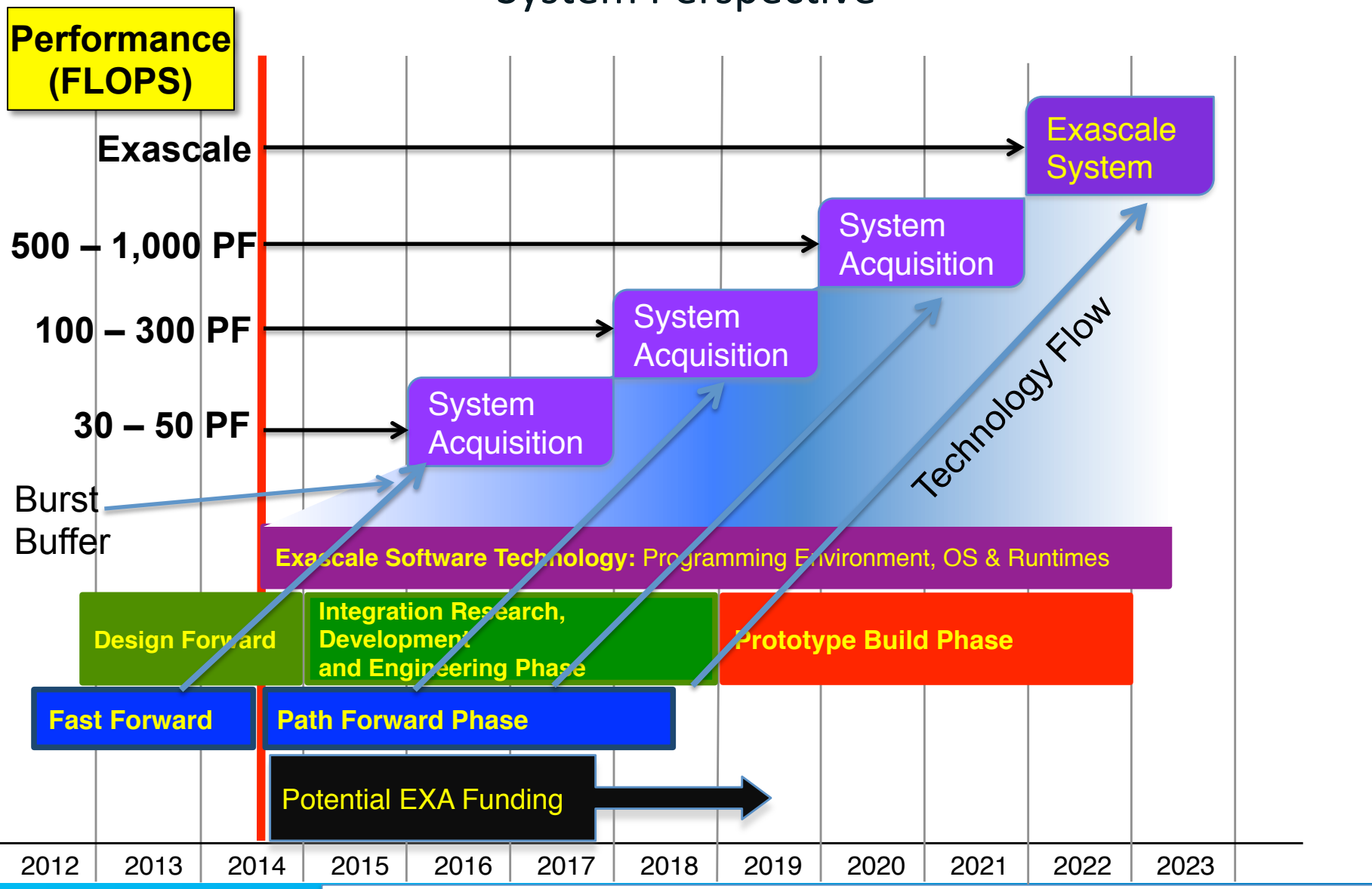
DOE Exascale Computing Timeline
(Anticipated)



U.S. DEPARTMENT OF ENERGY | Office of Science

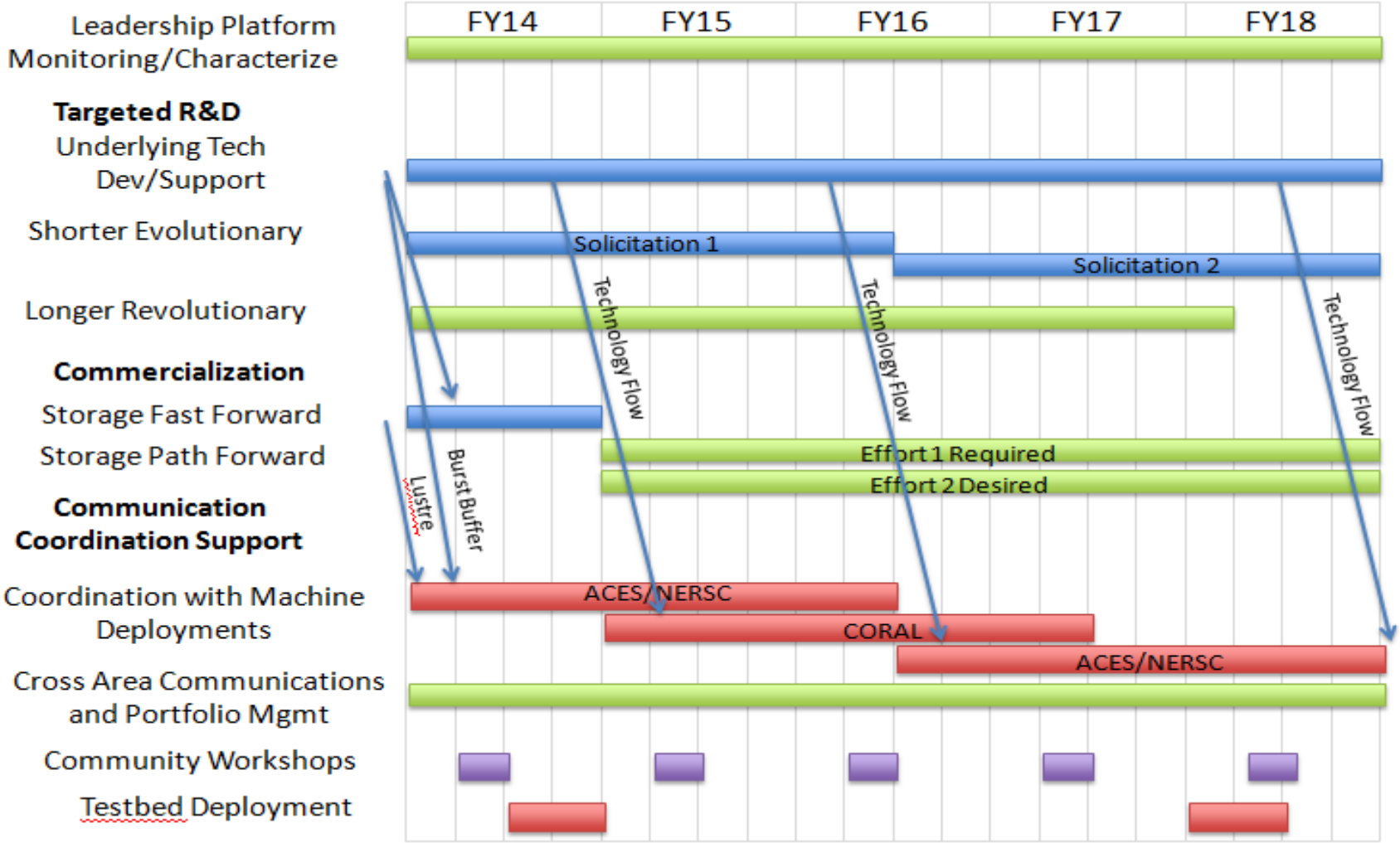
National Nuclear Security Administration

Exascale Computing Initiative Timeline
System Perspective



Data Management Notional Strategy

Storage and I/O Timeline



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 Processor, Memory and Storage
- IBM, Intel, AMD, and nVidia won Memory/Storage elements
- Whamcloud won the Storage (filesystem) component
    - HDF Group – HDF5 modifications and extensions
    - EMC – Burst Buffer manager and I/O Dispatcher
    - Cray - Test
- Contract renegotiated on Intel acquisition of Whamcloud
    - Intel - Arbitrary Connected Graph Computation
    - DDN - Versioning OSD

FF Storage Team and Uber-High Level Requirements

Constant failures expected at exascale

Storage System must guarantee data **and** metadata consistency

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

Storage System must guarantee data integrity

- Required end-to-end

Storage System must always be available

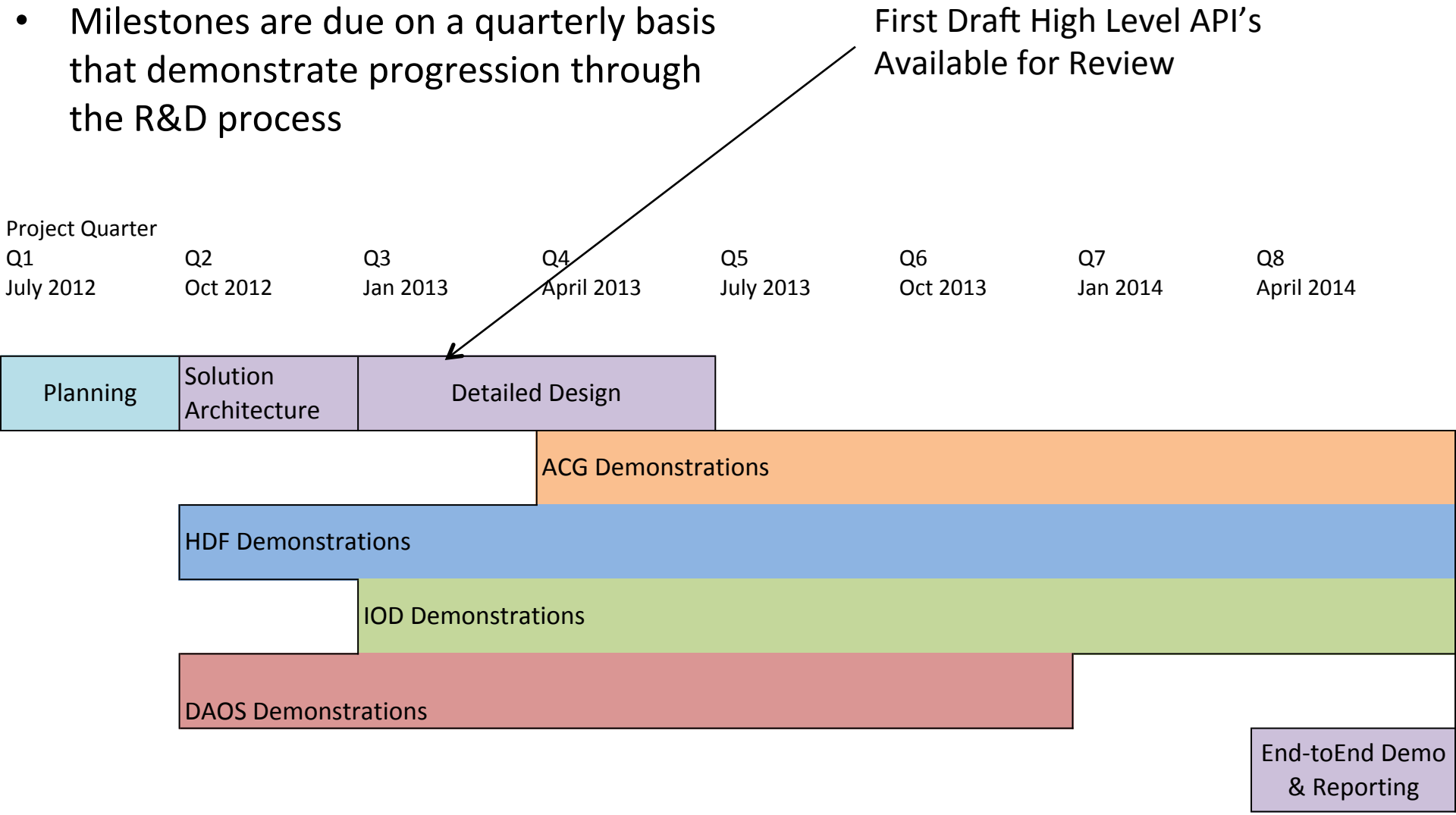
- Balanced recovery strategies
- Transactional models for fast cleanup on failure
- Scrubbing for repair / resource recovery ok to take days-weeks

A completely redesigned IO stack for Exascale

- Objects instead of files
  - Array objects for semantic storage of multi-dimensional data
  - Blob objects for traditional sequences of bytes
  - Key-value stores for smaller get/put operations
- Containers instead of directories
  - Snapshots for efficient COW across sets of objects (with provenance)
  - Transactions for atomic operations across sets of objects
- List IO all the way through the stack
  - Reduce trips across network
- Everything fully asynchronous with Transactions
  - Reads, writes, commits, unlink, etc
- Explicit Burst Buffer management exposed to app
  - Migrate, purge, pre-stage, multi-format replicas, semantic resharding
- End-to-end data integrity
  - Checksums stored with data, app can detect silent data corruption
- Structure preserving and Analysis Shipping for In-Transit/In-Storage processing

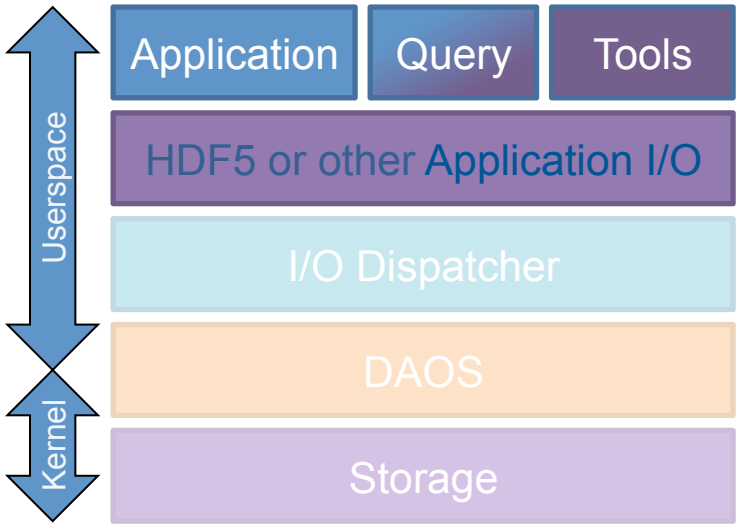
Storage Fast Forward Timeline

- Milestones are due on a quarterly basis that demonstrate progression through the R&D process



Application I/O: HDF5 or Other

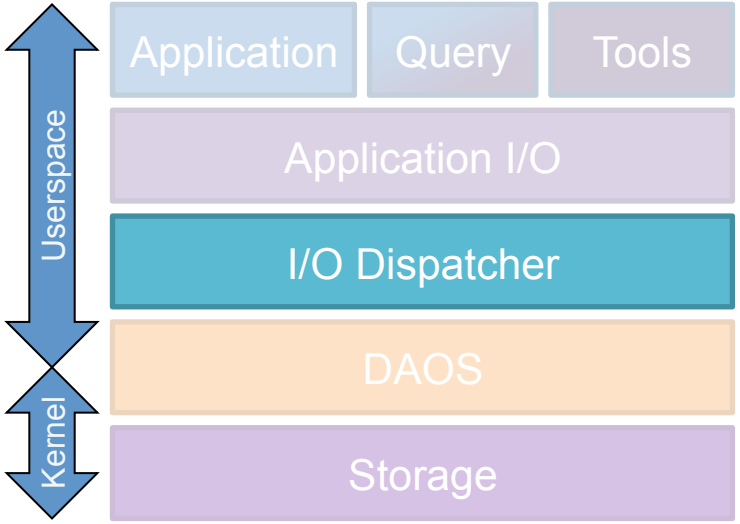
- New Application Capabilities
  - Asynchronous I/O
    - Create/modify/delete HDF5 objects
    - Read/write HDF5 Dataset elements
  - Transactions
    - Group many HDF5 API operations into single transaction
- HDF5 Data Model Extensions
  - Pluggable Indexing + Query Language
  - Pointer datatypes
- New Storage Format
  - Leverage I/O Dispatcher/DAOS capabilities
  - End-to-end metadata+data integrity
  - Built-for-HPC storage containers



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

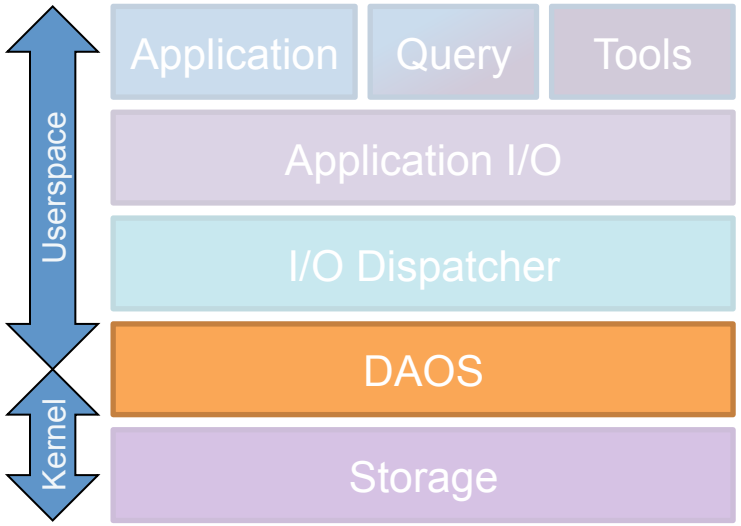
I/O Dispatcher

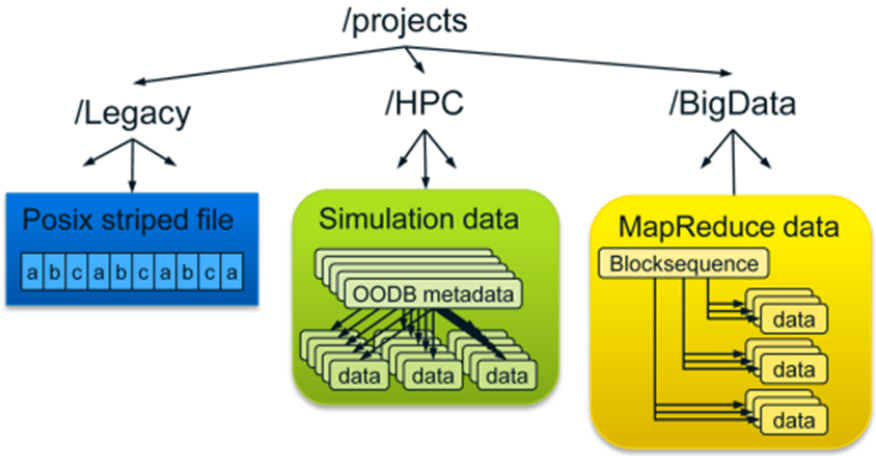
- Based on PLFS (parallel log structure) and MDHIM (parallel KVS)
- I/O rate/latency/bandwidth matching
  - Burst buffer / pre-fetch cache
  - Absorb peak application load
  - Sustain global storage performance
- Layout optimization
  - Application object aggregation / sharding
  - Upper layers provide expected usage
- Higher-level resilience models
  - Exploit redundancy across storage objects
- Scheduler integration
  - Pre-staging / Post flushing
- In-Transit Analysis processing (capable of global communications across IO Nodes / BB Nodes)



DAOS Containers

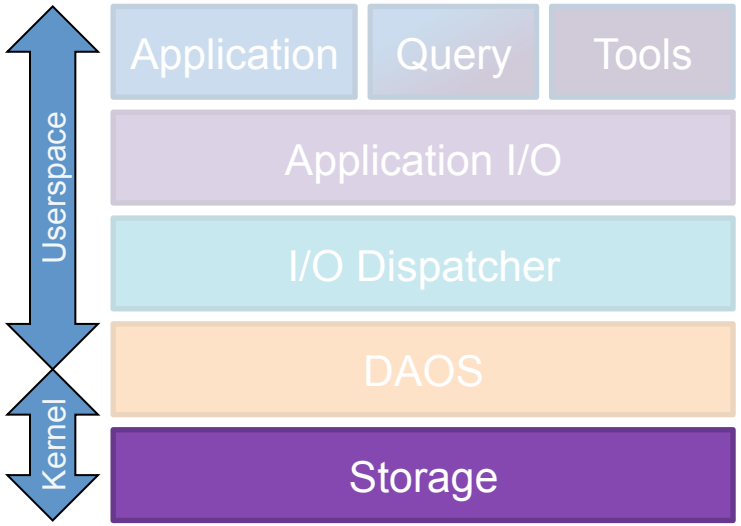
- Distributed Application Object Storage
  - Sharded transactional object storage
  - Virtualizes underlying object storage
  - Private object namespace / schema
- Share-nothing create/destroy, read/write
  - 10s of billions of objects
  - Distributed over thousands of servers
  - Accessed by millions of application threads
- ACID transactions
  - Defined state on any/all combinations of failures
  - No scanning on recovery
- In-Transit/Storage Analysis Processing





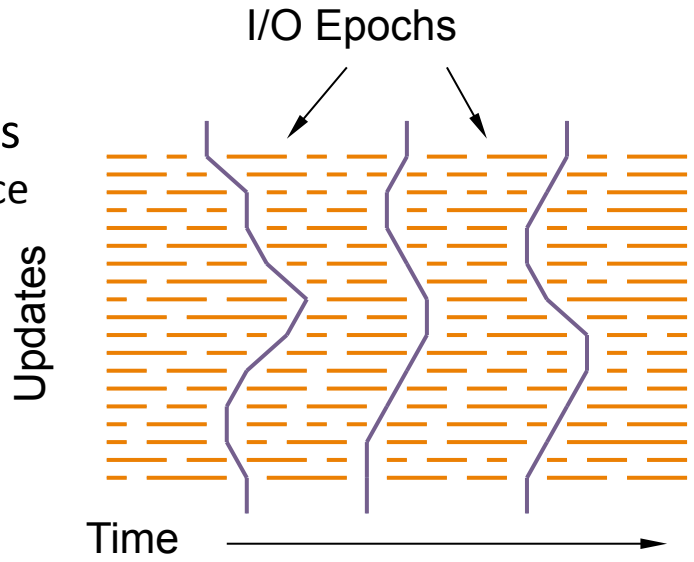
Versioning OSD

- DAOS container shards
  - Space accounting & quota
  - Shard objects
- Transactions
  - Container shard versioned by epoch
  - Writes ordered by epoch, not time
  - Commit
    - Integrates writes into extent metadata
    - Epoch durable once all shards commit
  - Abort
    - Rollback to last globally committed container version



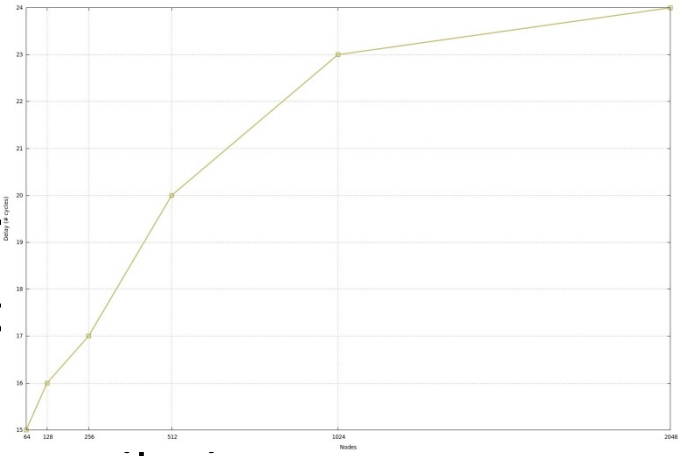
Transactions

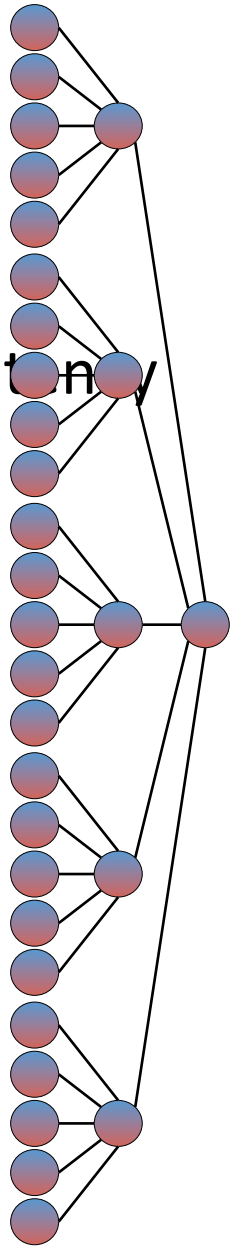
- Consistency and Integrity
  - Guarantee required on any and all failures
    - Foundational component of system resilience
  - Required at all levels of the I/O stack
    - Metadata at one level is data to the level below
- No blocking protocols
  - Non-blocking on each OSD
  - Non-blocking across 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



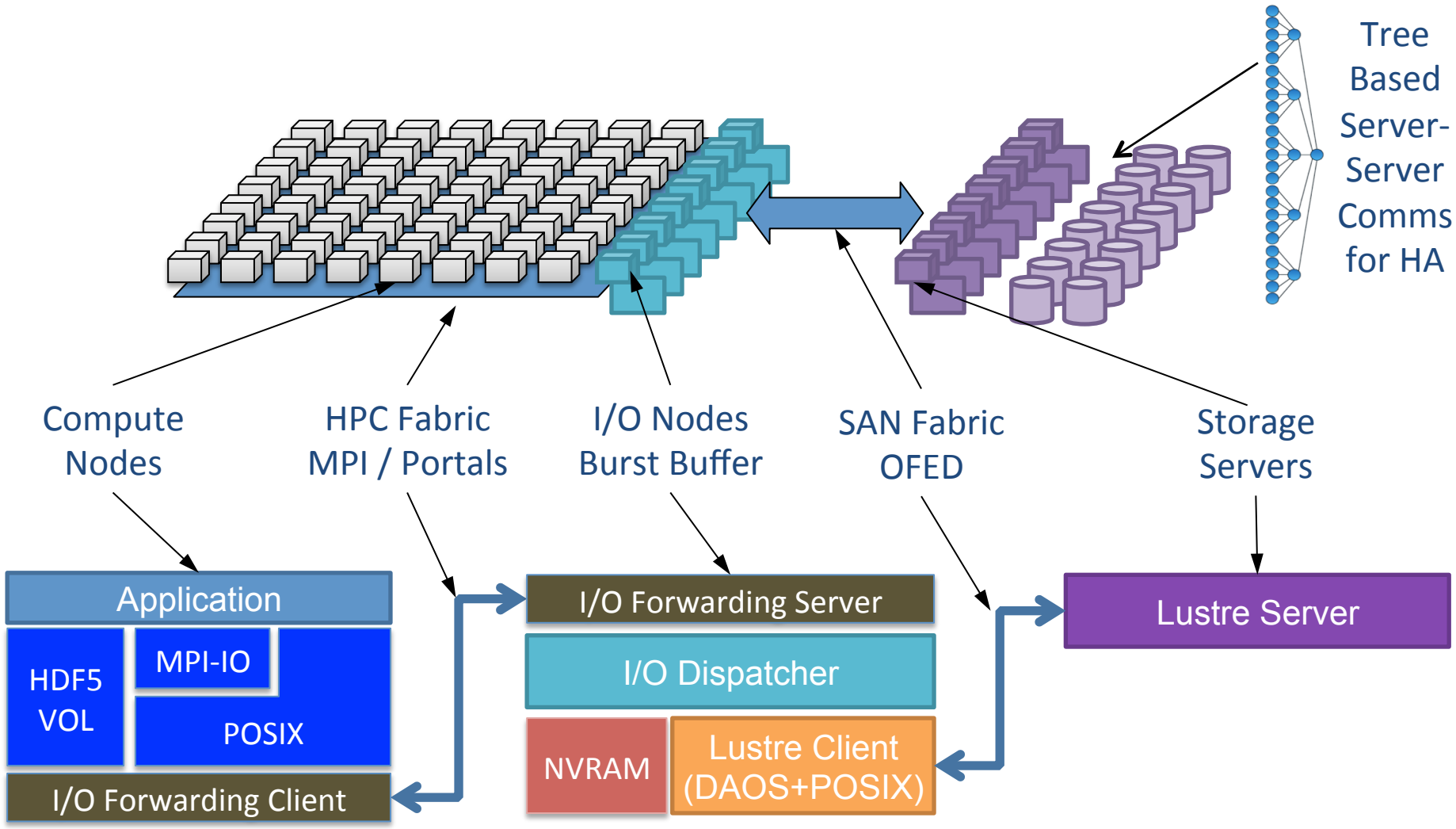
Server Collectives

- Gossip protocols
  - Fault tolerant O(log n) global state distribution latency
  - Peer Discovery
- Tree overlay networks
  - Fault tolerant
    - Collective completes wit
      on quorum change
  - Scalable server commun
    - DAOS transaction collect
    - Collective client eviction
    - Distributed client health monitoring

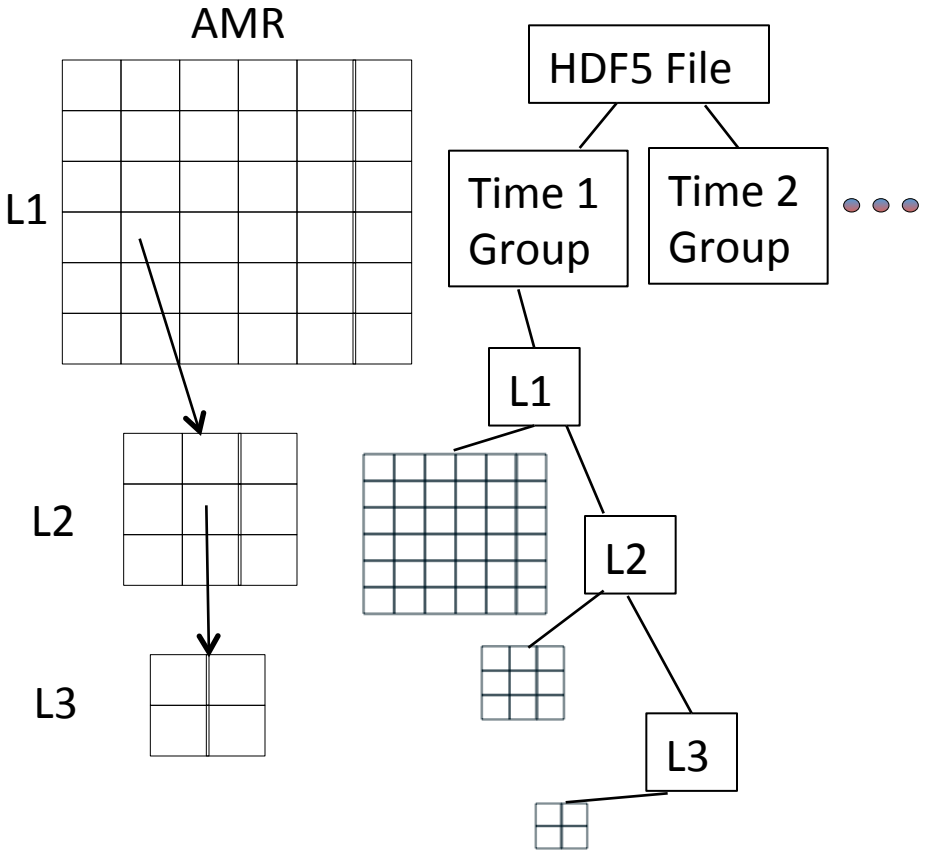




Fast Forward I/O Architecture



HDF5 (the current example of a high level API to this new IO stack)



- H5TRBegin-a (trans1, req1)
- H5Create-file-a (… trans, req2))
- H5Creaet-group-a(… trans, req3)
- H5TRCommit-a (trans1)
- Go do other work
- H5TRCheck/wait (trans1) or HTReqCheck/ wait(reqN)
- H5TRBegin-a(trans2,req4)
- H5Dwrite (object/array,trans2,rew5) (write all you want)
- H5TRCommit (trans2)
- Go do other work,
- You can even start a new transaction to do metadata or data ops with trans3++ and overlap as much IO and computation, including abort.
- You cant be sure anything made it to storage until H5TRCheck/wait say that transaction is secure.
- You can control structure, async behavior, rollback, etc.

IOD API

- OBJECT FUNCTIONS
- iod_obj_create, create_list, open, close
- iod_array_write, array_write_list, blob_write, blob_write_list
- iod_obj_open_read read_list, blob_read, blob_read_list, array_read, array_readlist
- iod_array_extend, query
- iod_obj_set_layout and get_layout
- iod_obj_unlink and unlink_listiod_obj_unlink_list
- iod_obj_set_scratch, get_scratch
-
- DATA CONSISTENCY
- iod_trans_query, trans_start, trans_slip, trans_finish, trans_persist, trans purge,
- trans_fetch, trans_replica, trans_snapshot
-
- Key Value functions
- iod_kv_set, set_list, get_num, get_list, list_key, get_value, unlink_keys
-
- EVENT FUNCTIONS
- iod_eq_create, destroy, poll, abort, query, init, finish

DAOS

- Event Queue Mgmt
- daos_eq_create, destroy, poll, query, init, finish, next, abort
-
- DAOS Storage Layout Mgmt
- daos_sys_open **cage/rack/node/target**, close, listen, query
-
- Container Structure Mgmt
- daos_container_ope, unlink, snapshot, query, listen
-
- Collective operation APIs
- daos_local2global and global2local
-
- Shard API
- daos_shard_add, disable, query, flush
-
- Object API
- daos_object_open, close, read, write, flush, punch
-
- Epoch & Epoch functions
- daos_epoch_scope_close, catchup, commit

Finish

- The remaining part of the draft api set that hasn't been added yet is analysis/function shipping.
- We do want people to look at our API's and even designs which are on the open web as we pay for them http://wiki.whamcloud.com
- We are interested in a variety of uses, library call, function/ analysis shipping, HDF5, DSL, structured, AMR, etc. We can set up a limited set of direct interaction with the subsystem owners
- Later when we have prototype libs/system, a limited set of app developers may want to do some simple trials
- Remember, this is a prototyping activity and ends in FY14.