LA-UR-15-21297
Approved for public release; distribution is unlimited.

Title: Trinity Burst Buffer - Architecture and Design

Author(s): Wright, Cornell G. Jr.

Intended for: ASC PI Meeting, 2015-02-10/2015-02-12 (Monterey, California, United States)

Issued: 2015-02-23

Trinity Burst Buffer

Architecture and Design

Cornell Wright

February 2015

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Agenda

- Acknowledgements
- Technology Trends
- Trinity Burst Buffer Hardware
- Trinity System Software
- Application Software

Acknowledgements

- Many people and groups have contributed to the concept, design and development of burst buffer, Trinity and the materials in this talk.
- Including:
  - Gary Grider
  - Josip Loncaric
  - Doug Doerfler
  - Nick Wright
  - Nathan Hjelm
  - Dave Henseler
  - Bob Pearson
  - Bronis de Supinski
  - Adam Moody
  - John Bent
  - Cray

Agenda

- Acknowledgements
- ☑ Technology Trends
- Trinity Burst Buffer Hardware
- Trinity System Software
- Application Software

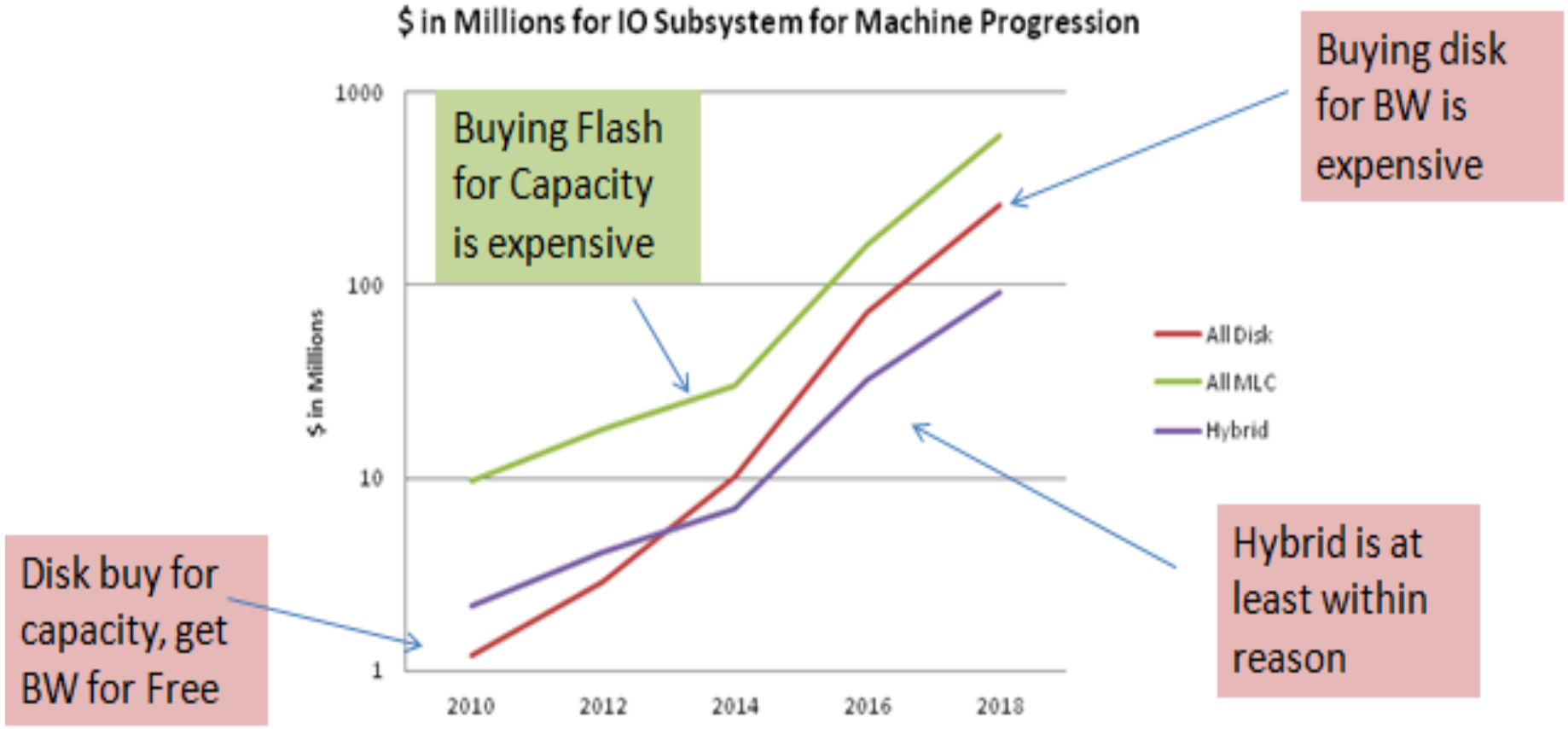
Memory and Storage Trends 2020 and beyond

Trend	Technology	Reasons
↗	On Package Memory	High BW needed
↘	DIMM Memory	Doesn't meet CPU BW needs
↗	Solid state non-volatile memory	Cost effective BW Decreasing capacity prices Decreasing latency Improving endurance
↘	High speed PFS	BW increasing more slowly than than capacity
↗	Near-line storage	Likely still cheaper per capacity than non-volatile memory
?	Tape Archive	Unknown (unlikely ?) if it can keep up with disk

Economic Analysis Results (circa 2009)

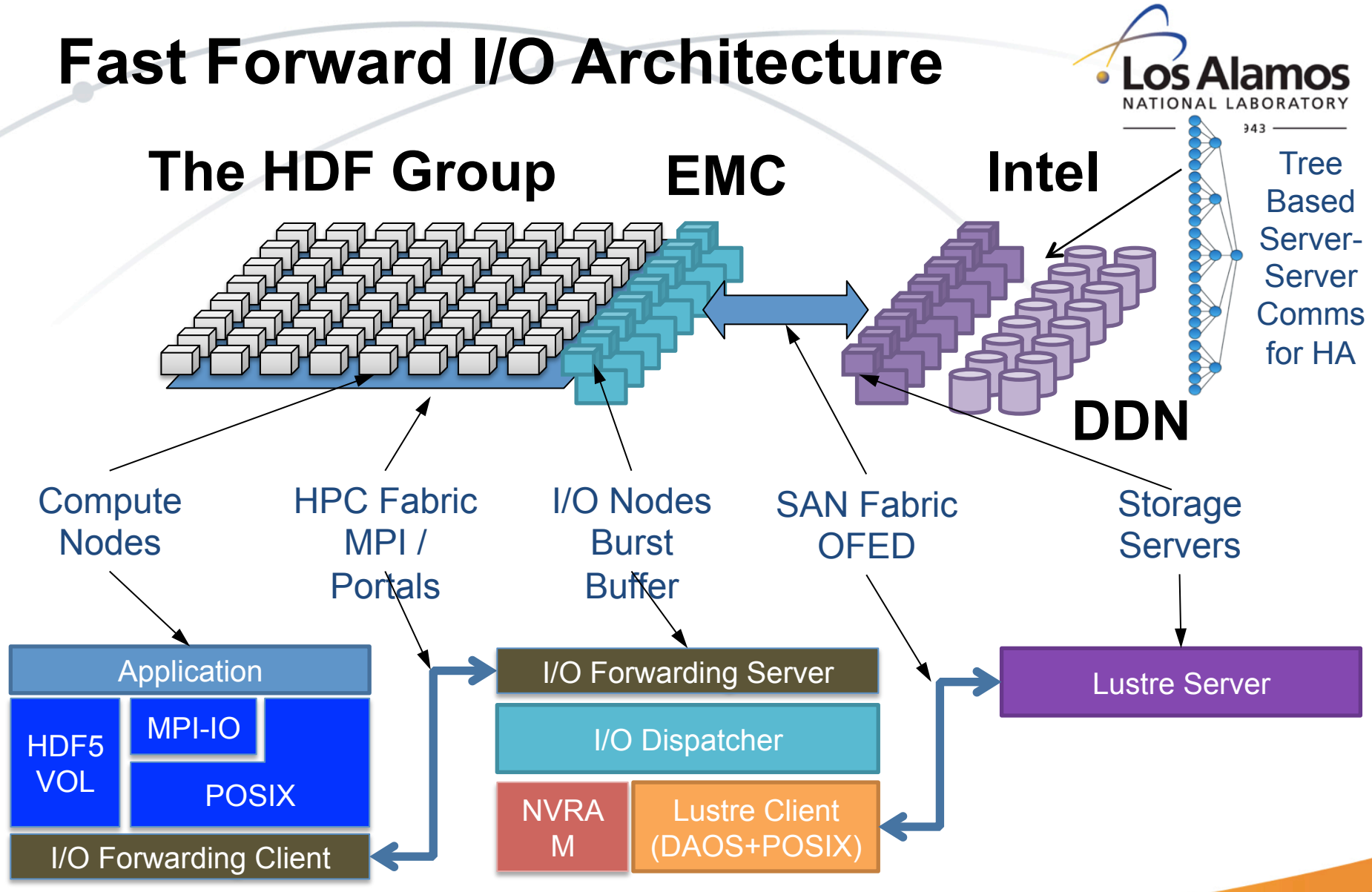
Los Alamos NATIONAL LABORATORY EST. 1943

Must Meet Two Requirements: **1 EB Capacity and 100 TB/sec**



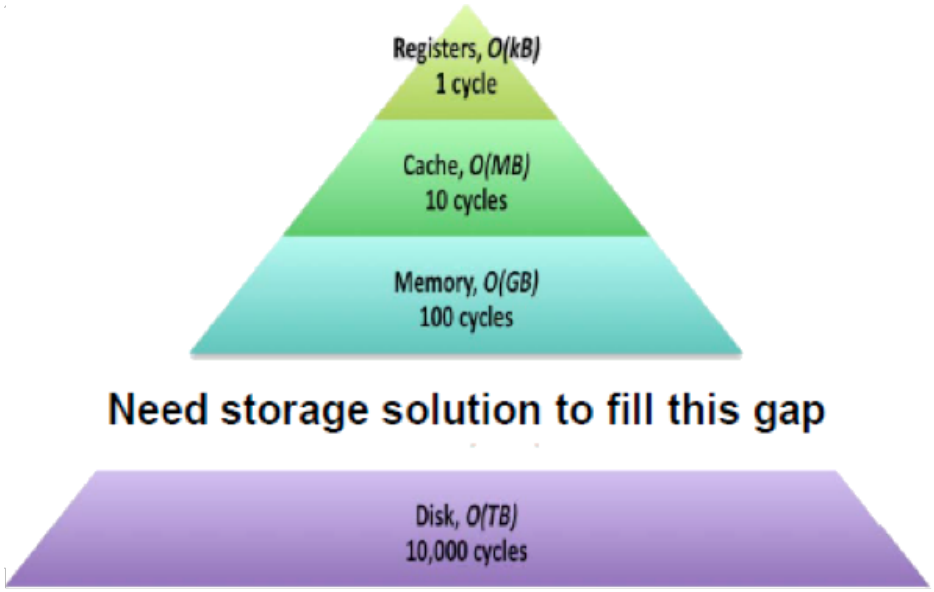
Take away: $M in Log Scale means you have hit the big time!

Fast Forward I/O Architecture



Gap in the Storage Hierarchy

- Burst Buffer is flash storage which would act as a cache to improve peak performance of the PFS.



- Flash is currently as little as 1/6 the cost of disk per GB/s bandwidth and has better random access characteristics (no seek penalty).
- How can we exploit this technology for HPC ?

Burst Buffer is a fast checkpoint store

- Hard drive capacity growing faster than speed
- Parallel File Systems (PFS) now sized on bandwidth
  - Previously sized on capacity
- Flash memory cost per bandwidth declining
- Flash still more expensive than disk for capacity
- Flash less expensive than disk for bandwidth
- Hybrid approach most economical
- Burst Buffer is:
  - High speed checkpoint store
  - Capacity 2 – 3x memory
  - Flash (only cost-effective option right now)
- Allows PFS to be sized for capacity

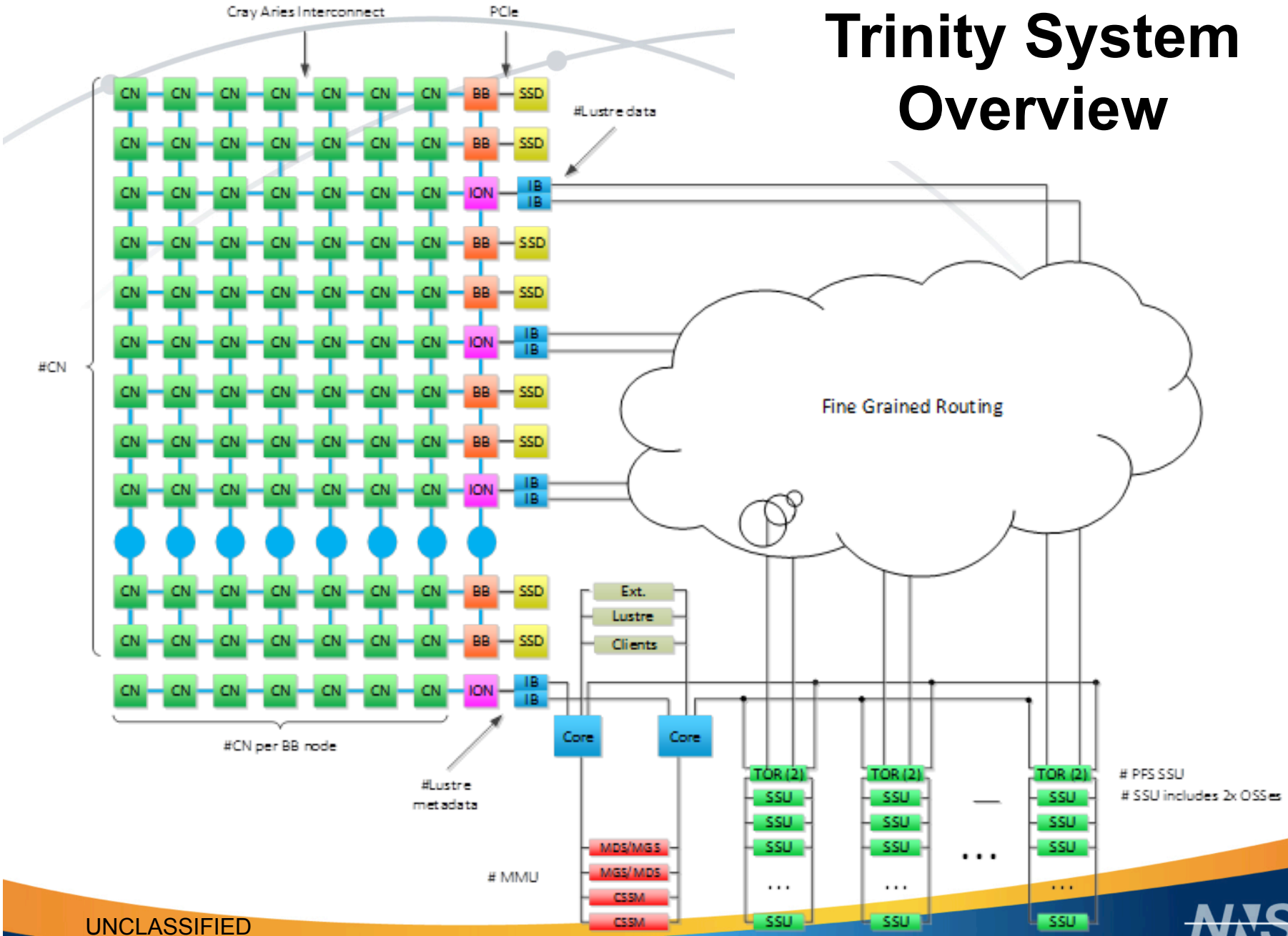
Burst Buffer – more than checkpoint

- Use Cases:
  - Checkpoint
    - In-job drain, pre-job stage, post-job drain
  - Data analysis and visualization
    - In-transit
    - Post-processing
    - Ensembles of data
  - Data Cache
    - Demand load
    - Data staged
  - Out of core data
    - Data intensive workloads that exceed memory capacity

Agenda

- Acknowledgements
- Technology Trends
- ☑ Trinity Burst Buffer Hardware
- Trinity System Software
- Application Software

Trinity System Overview



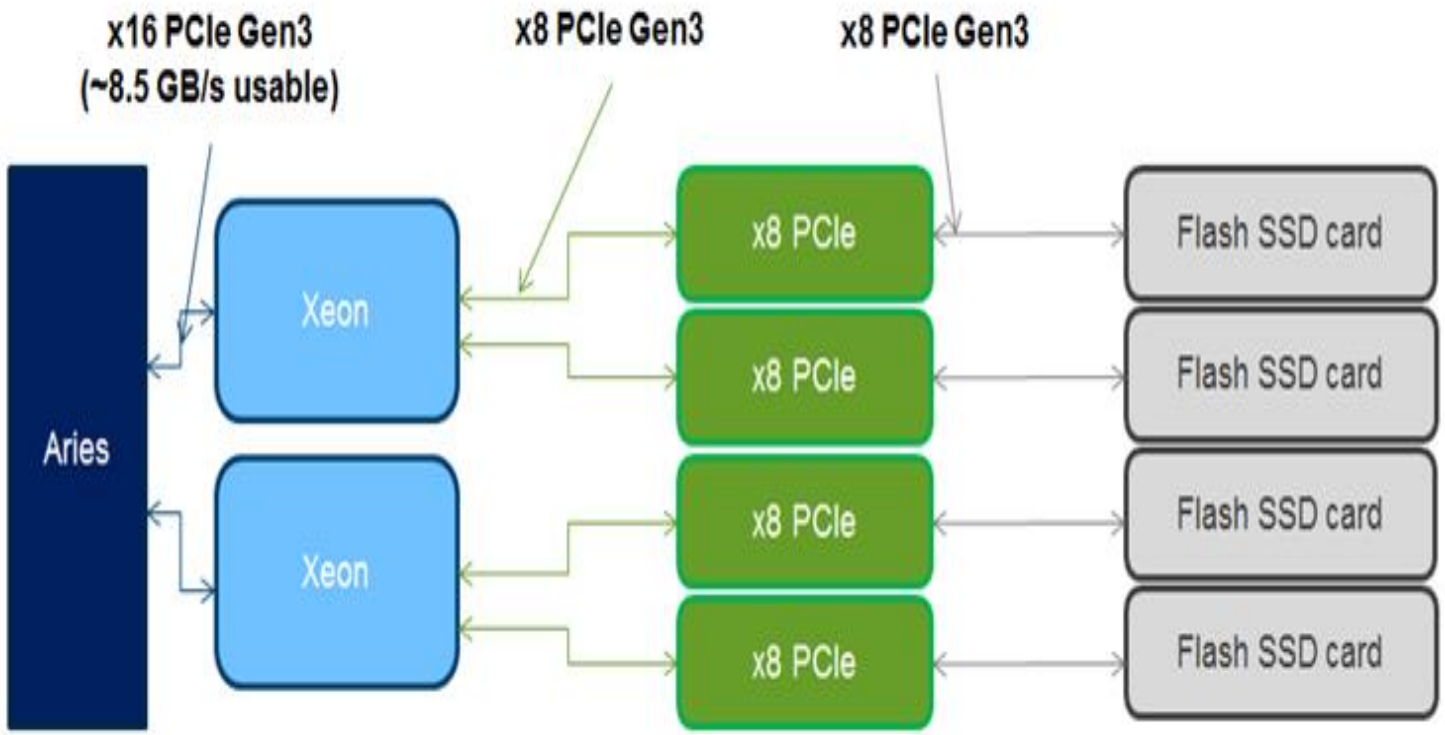
Trinity Burst Buffer Hardware

- Trinity:
  - ~10K Haswell + ~10K KNL nodes
  - 2.1 PB memory
- 576 Burst Buffer Nodes
  - Announced as Cray DataWarp™
  - On high speed interconnect
  - Globally accessible
  - Trinity IO Node + PCIe SSD Cards
  - Distributed throughout cabinets

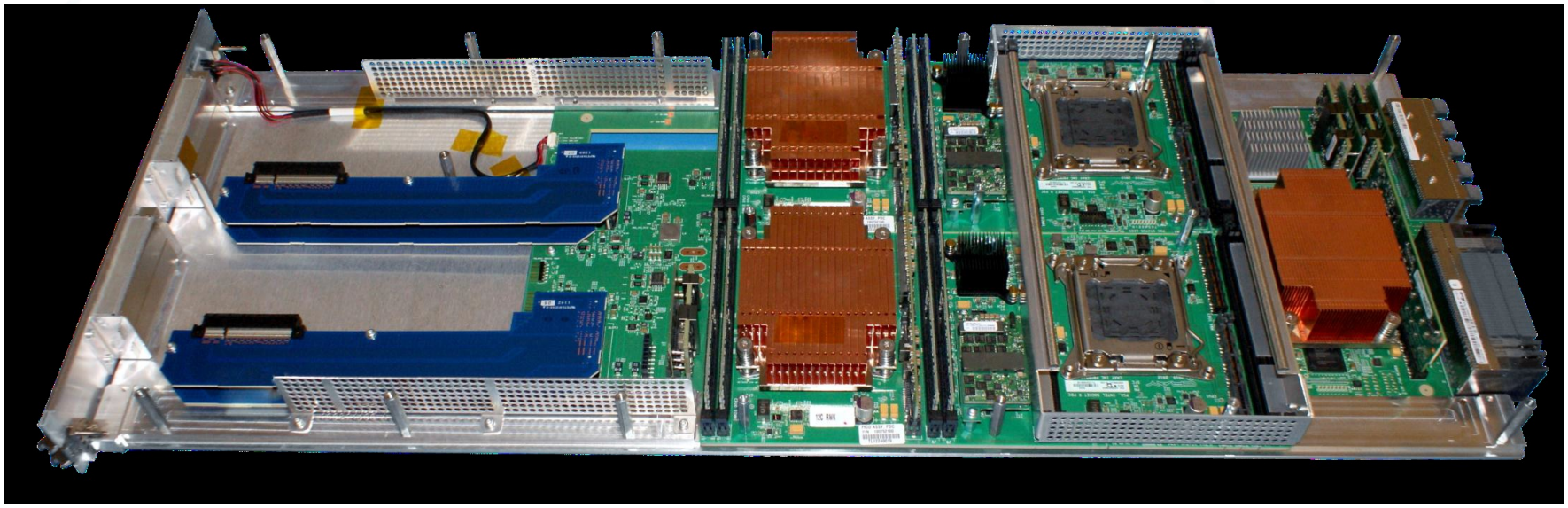
Metric	Burst Buffer	PFS
Nodes	576 BB Nodes	234 LNET Routers
Bandwidth	3.3 TB/S	1.45 TB/S
Capacity	3.7 PB	82 PB
Memory Multiple	1.75 X	39 X
Application Efficiency	88%	79%

Cray XC40 DataWarp Blade

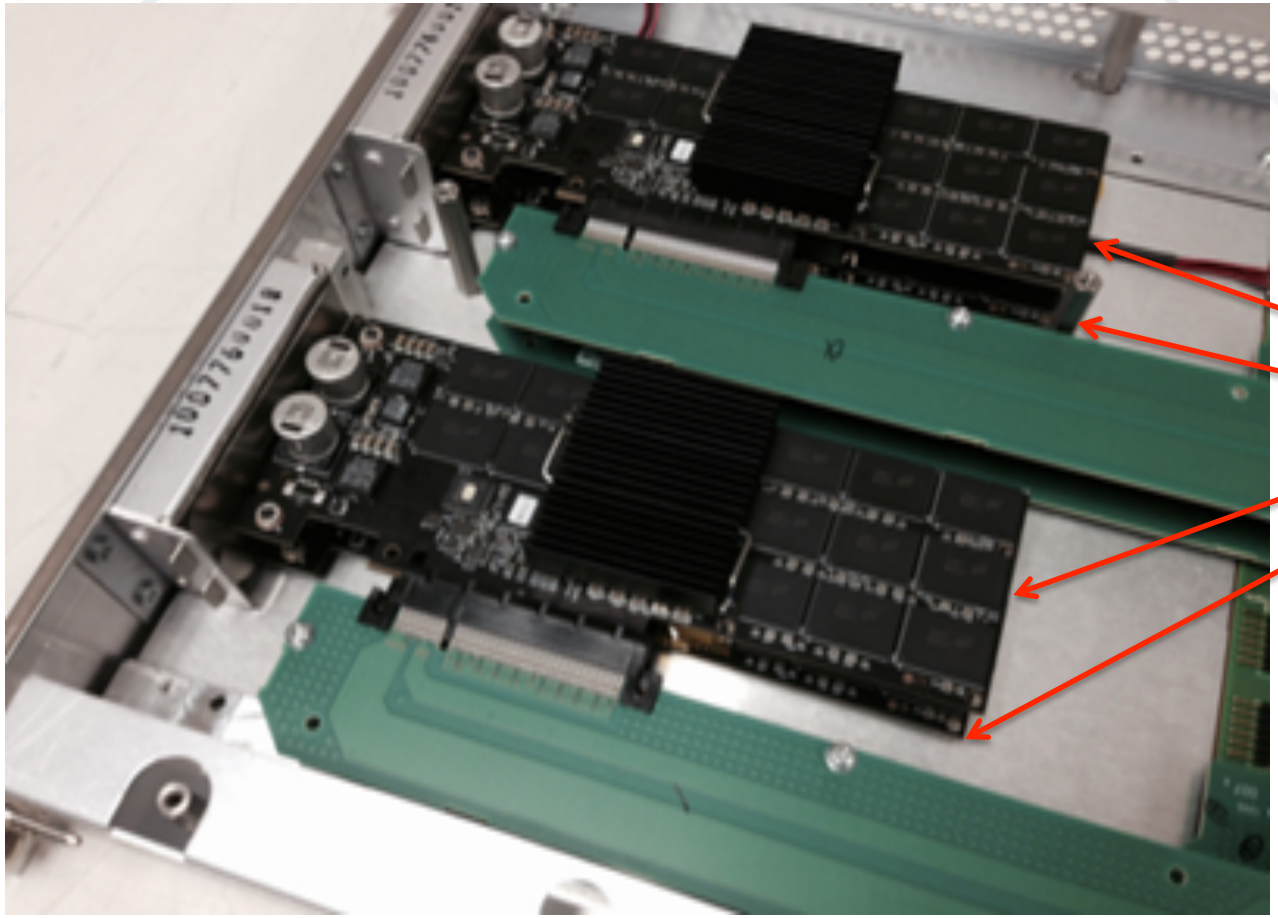
(2 Burst buffer Nodes)



Cray XC30 IO Blade



Cray XC40 DataWarp Blade



4 PCIe 3.2 TB SSD Cards

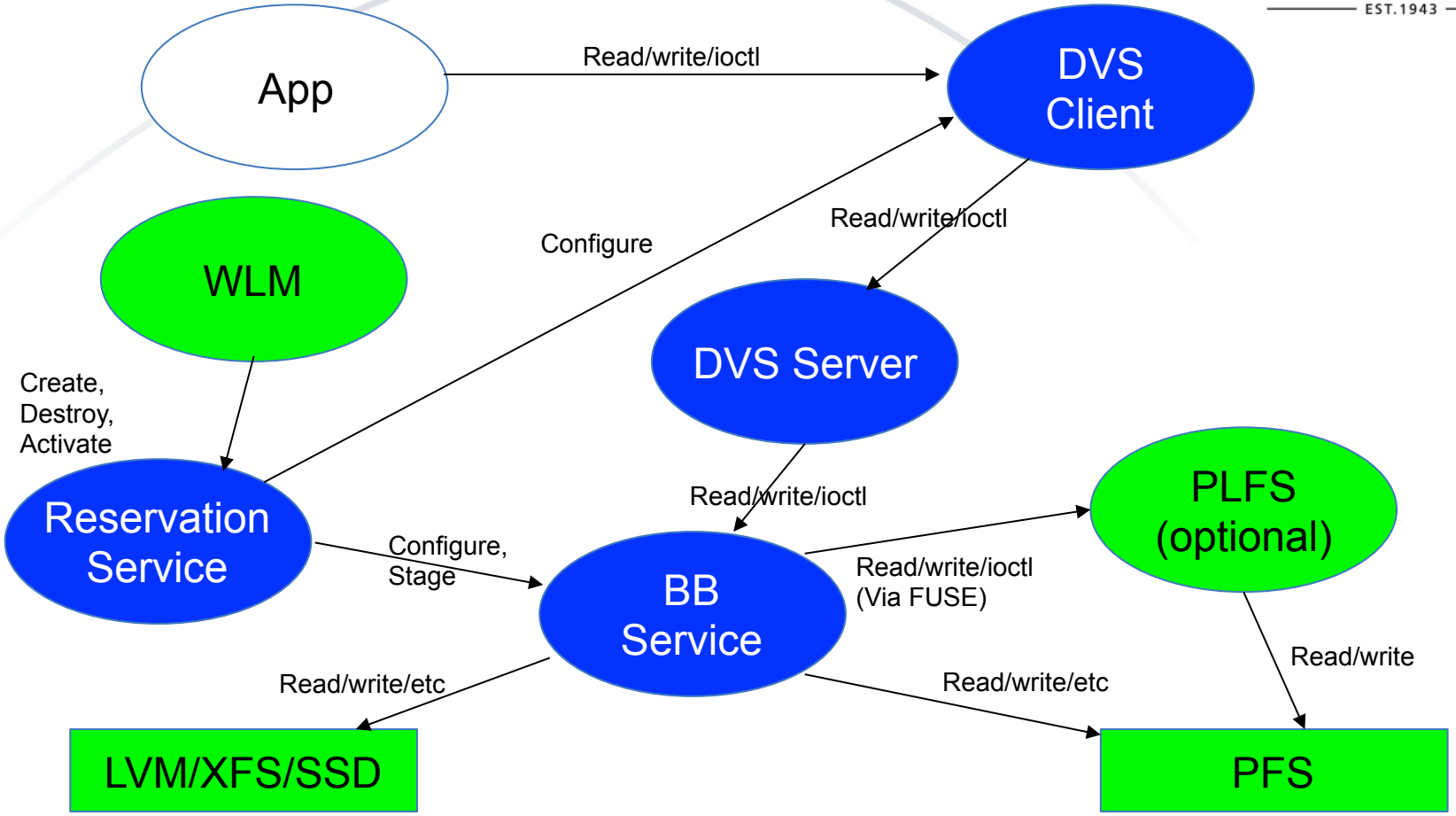
Agenda

- Acknowledgements
- Technology Trends
- Trinity Burst Buffer Hardware
- ☑ Trinity System Software
- Application Software

Burst Buffer System Software

- Burst Buffer SSD partitioned into allocation units
  - Allocation units belong to LVM volume group
- Workload manager
  - Job submission requests BB capacity
  - Starts job when capacity available
- Burst Buffer registration service
  - Selects allocation units
  - Creates XFS logical volumes on SSD
  - Mounts via DVS on compute nodes
- Multiple Access Modes
  - Scratch / Cache
  - Striped / Private / Load Balanced (RO)
- Trinity BB Checkpoints will use Striped Scratch
- Automated stage/drain of specified directories from/to PFS
- Per job write limits (endurance management)
- PLFS file structure on PFS (Optional)
- Administrative Functions – configuration, monitoring, repair

Burst Buffer System Software



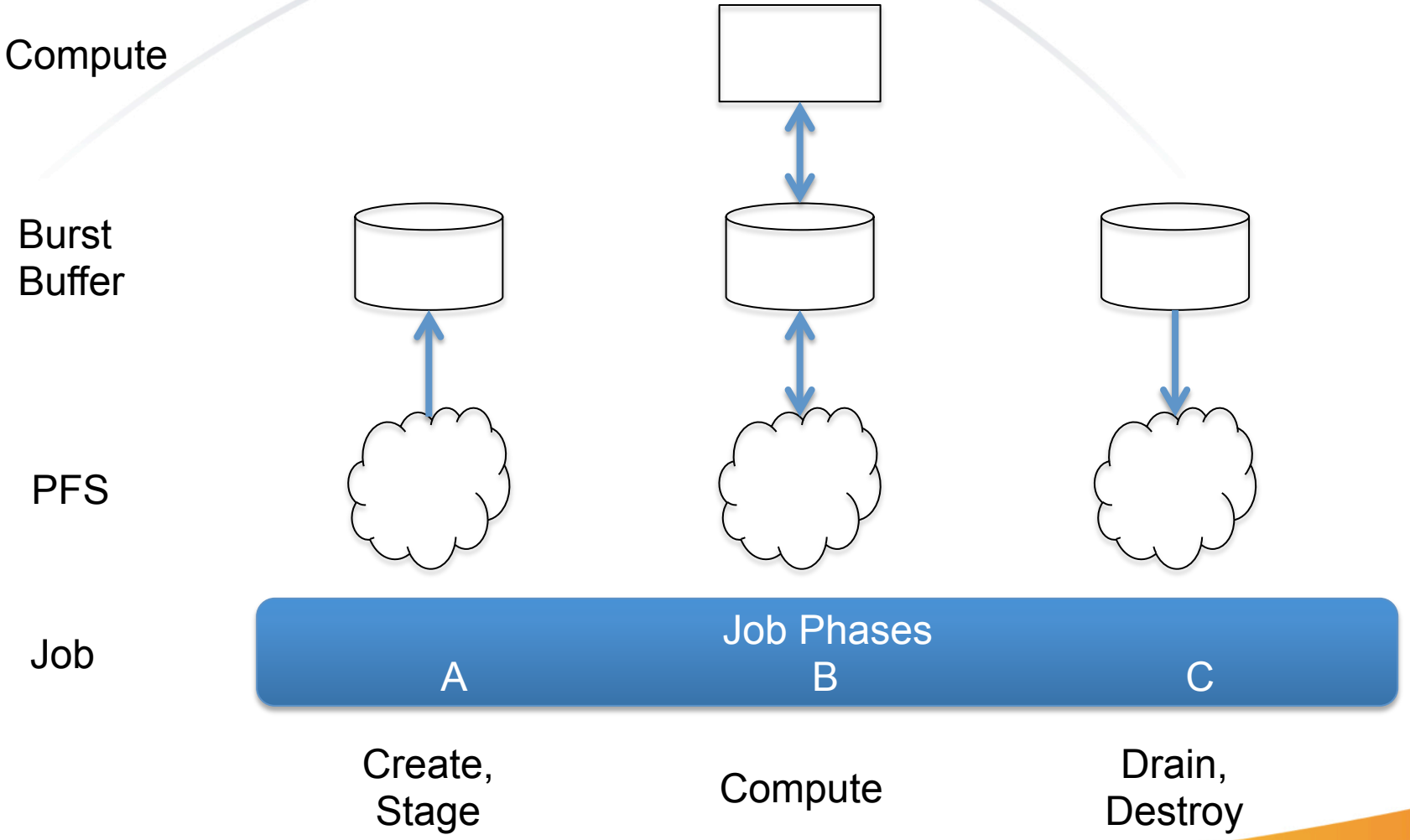
Cray

Third party

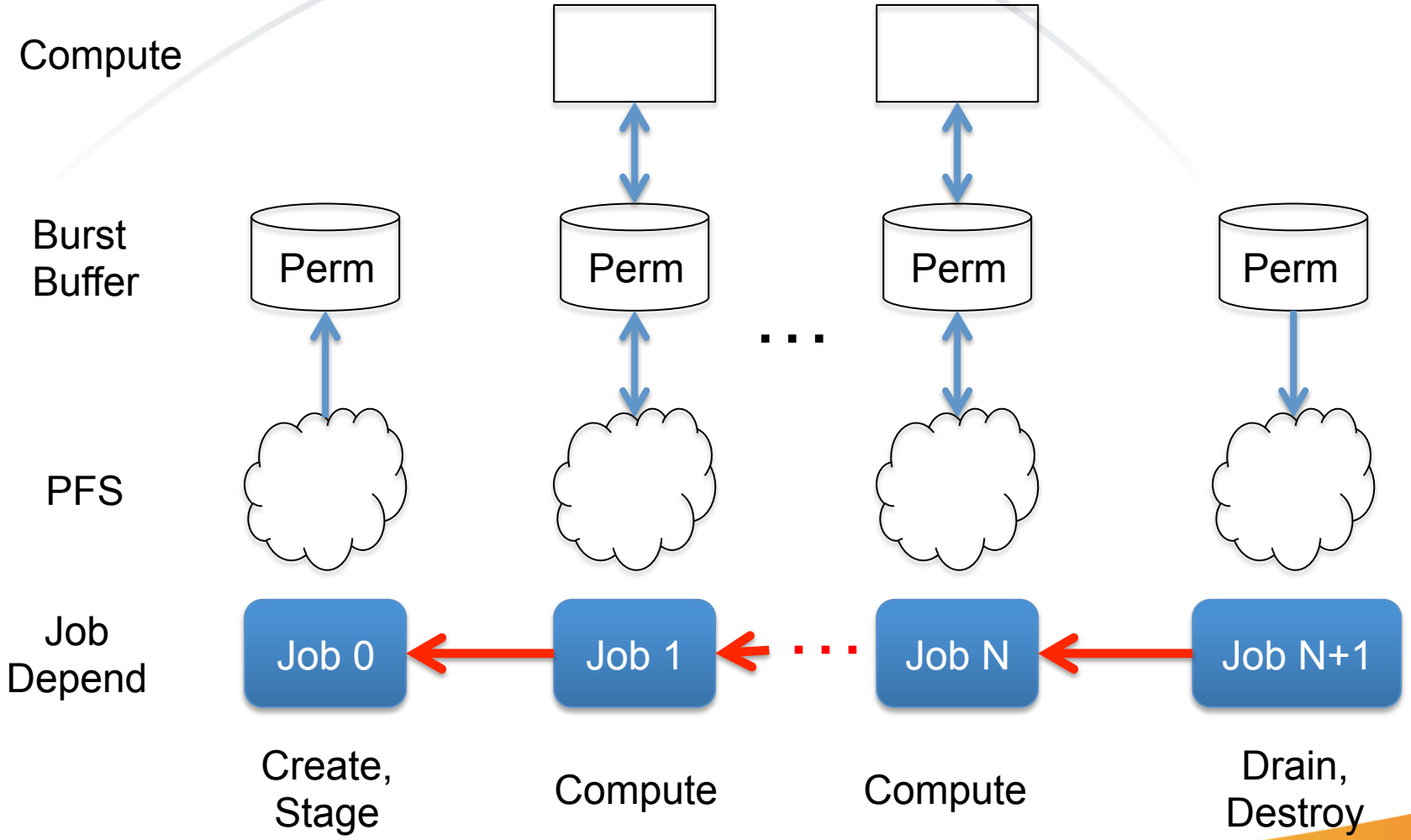
Burst Buffer Operating Modes

Mode	Description
Private Scratch	Per node burst buffer (BB) space
Shared Scratch	Shared space, files may be striped across all BB nodes.
Shared Cache	Parallel File System (PFS) cache. Transparent and explicit options
Load Balanced Read Only Cache	PFS files replicated into multiple BB nodes to speed up widely read files

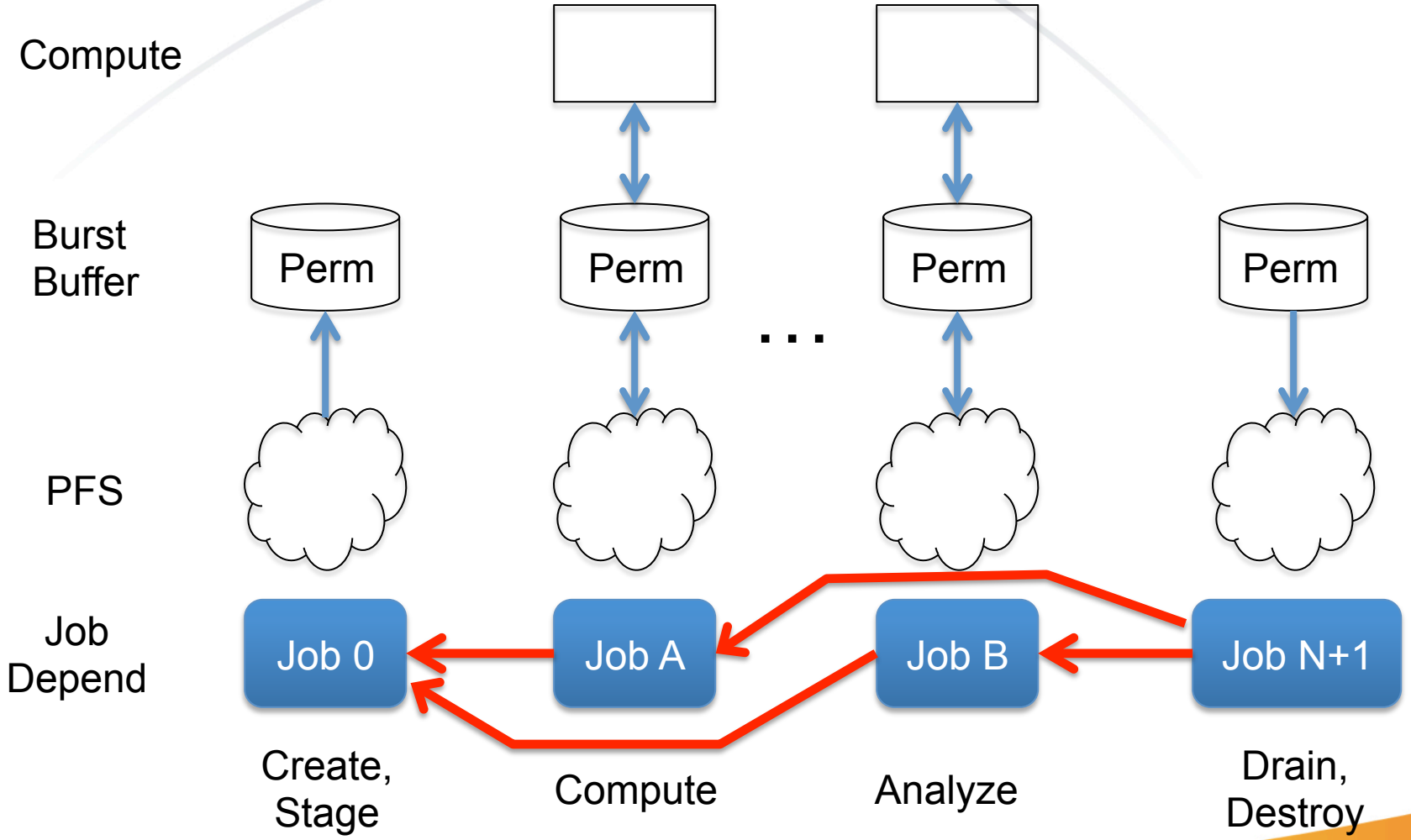
Single-Job Burst Buffer



Multi-Job Burst Buffer



Compute + Analysis Burst Buffer



Agenda

- Acknowledgements
- Technology Trends
- Trinity Burst Buffer Hardware
- Trinity System Software
- ☑ Application Software

Applications Have Options

- Direct POSIX calls from application
  - Will require Cray specific ioctl commands to exploit striping and stage / drain functions
- Hierarchical Input Output (HIO) Library
  - Hides vendor specific interface
  - Provides additional performance, reliability, management and diagnostic functions
- SCR
  - Adam Moody planning to support Trinity BB

HIO Design Goals

HIO, short for Hierarchical Input Output, is the burst buffer enablement library for Trinity and other systems.

Goals:

- Support Trinity and future Burst Buffer implementations
  - Isolate application from BB technology shifts
- Easy to incorporate into existing applications
  - Lightweight, simple to configure
- Improve checkpoint performance
- Improve job reliability and efficiency
- Support checkpoint and analysis scenarios (e.g., viz)
- Extend to bridge to future IO (e.g., Fast Forward IO)

Why implement HIO as a library ?

- Simplest packaging and delivery available
- Self contained, minimal external or system dependencies
- Easiest for applications to adopt
- Library approach facilitates rapid prototyping and deployment, responsiveness to application needs
- Library also provides a vehicle to provide (at no cost to appls):
  - Checkpointing best practices
  - Performance and functional diagnostics
  - Mitigation for system problems
- Why not provide via extensions to MPI-IO now ?
  - Existing implementation performs poorly
  - Unloved by users
- MPI-IO may be investigated in future
  - Will require standardization effort, possible code overhaul
  - HIO library would be largely reusable in that environment

HIO Project Features

- Full thread safety
- C/C++/Fortran support
- Configurable diagnostic and performance monitoring
- Header / Library packaging
- Open-source
- Support tri-lab ATS and CTS systems (more than Trinity)
- On-site (at LANL) prototype
- Intent to prototype EAP support for HIO as POC and test vehicle
- PFS-only version available before Trinity

HIO Design Features

- Flexible configuration capability
- Abstract view of IO namespace
- Open/Read/Write/Close data interfaces
- Checkpoint management
- Hardware error recovery
- (Future) Job management

Thank You !

Questions

HIO – Flexible Configuration

- Keyword=value format
- Multiple sources for flexibility
  - System file (optional)
  - Application file (entire or partial)
  - Application environment
  - API call
- Applied on rank 0 and propagated

HIO – Abstract Namespace

- Named Context / Dataset / ID / Element
  - Context: All data managed by an HIO Instance
  - Dataset: Particular type/format of data
  - ID: Instance of data, expected to be sequence
  - Element: Named section of dataset
- On Trinity, will map to BB & PFS directory structure
- Future system's BB may not have FS
- Limited guarantees on physical file structure
- Will provide export to POSIX file functionality

HIO – Data Interfaces

- Synchronous or asynchronous
- Open specific ID or highest ID
- Turnstile to limit concurrency
- Multiple destination directories
- Possible future directions:
  - POSIX read / write intercept
  - MPI-IO implementation
  - ADIOS interface

HIO – Checkpoint Management

- Advisory interface based on:
  - Node and system MTTI (system config)
  - Bandwidth (system config)
  - Job size
  - Checkpoint size
- Periodic BB checkpoint background drain to PFS
- BB checkpoint deletion for space management
- (Future) Schedule PFS traffic to reduce contention

HIO – Hardware Error Recovery

- Multiple data roots
  - e.g., BB; /scratch1; /scratch2
  - Errors on read handled by notification and subsequent fallback to secondary (or tertiary) root
  - Errors on write handled by notification, potential fallback to secondary (or tertiary) root and immediate rescheduling of checkpoint
  - Active data root's bandwidth will influence recommended checkpoint interval
- Any complete BB checkpoint will be marked eligible for post job drain to PFS (even if job subsequently fails.)
- Application level CRC on data (optional)

HIO – Job Management

- Potential future capability:
  - Enable graceful shutdown of jobs and system with final checkpoint to PFS