Argonne NATIONAL LABORATORY

Lawrence Livermore National Laboratory

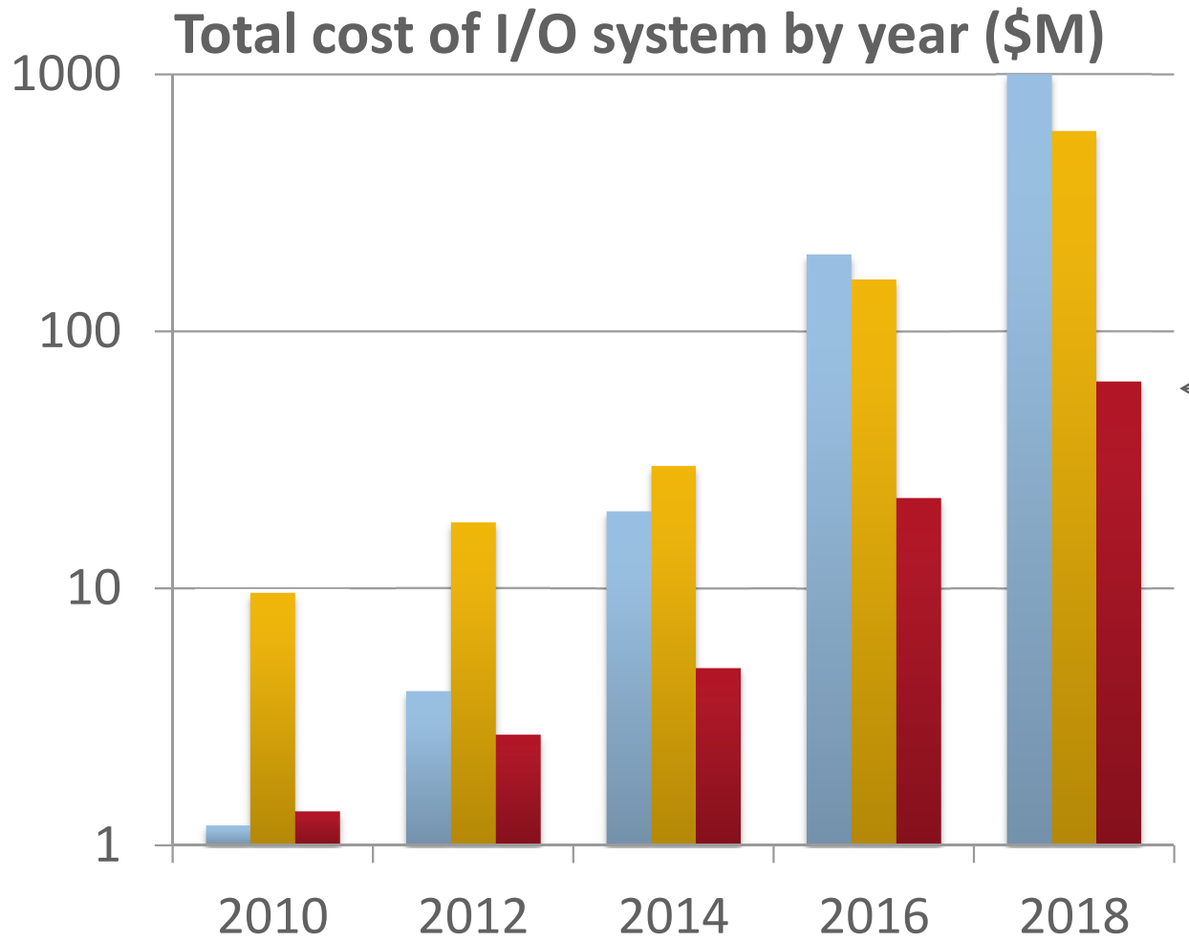
Integrated In-System Storage Architecture for High Performance Computing

Dries Kimpe, Rob Ross
Argonne National Laboratory

Kathryn Mohror, Adam Moody, Brian Van Essen, Bronis R. de Supinski, Maya Gokhale
Lawrence Livermore National Laboratory

U.S. DEPARTMENT OF ENERGY

Motivation



Assumptions: Require 900 PB capacity @ 100 TB/sec peak in 2018. Associated HW 4x raw device costs. No revolutionary solid state developments. Enterprise storage.

Thanks to Gary Grider (LANL) for this data.

- Exascale storage will be a hybrid of disk and solid state, unless we dramatically reduce capacity requirements.

Motivation

- HPC systems are quickly moving into a realm where hybrid SSD/HDD systems are necessary to hit performance and capacity targets.
- Yet, we do not understand how SSDs can best serve our science needs.

Goal:

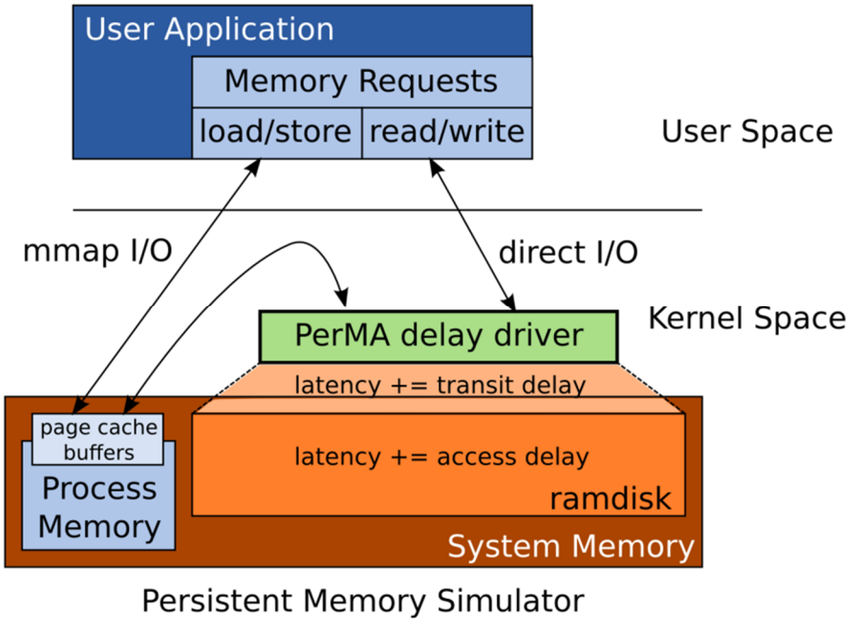
Conduct a detailed assessment of the potential roles and benefits of in-system storage in exascale computational science.

Thrusts:

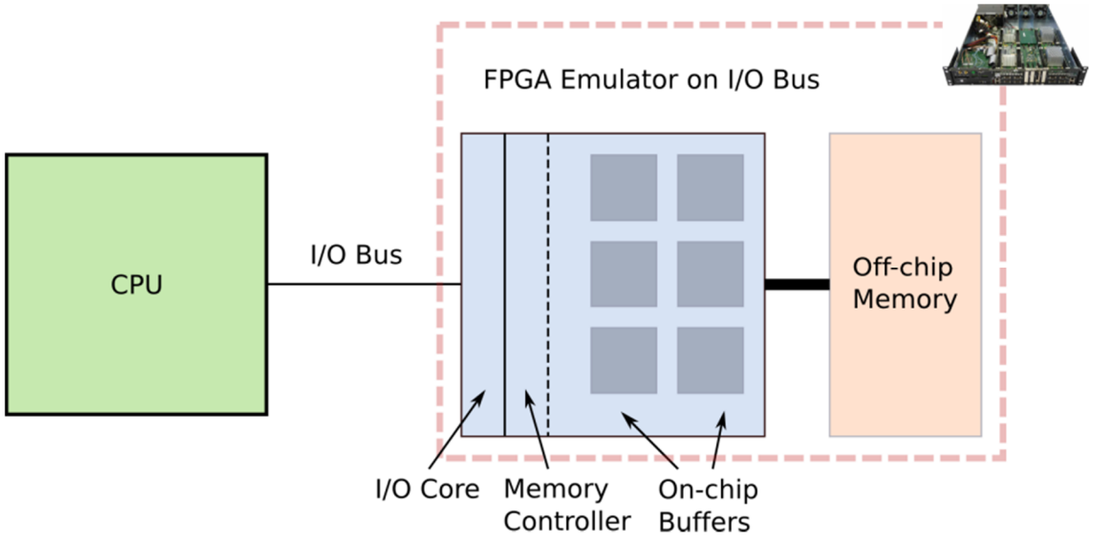
- Explore and evaluate existing/emerging hardware options
- Assess software mechanisms that best exploit them, implement examples, provide feedback
  - Checkpoint and restart
  - Data management

Hardware Exploration and Evaluation

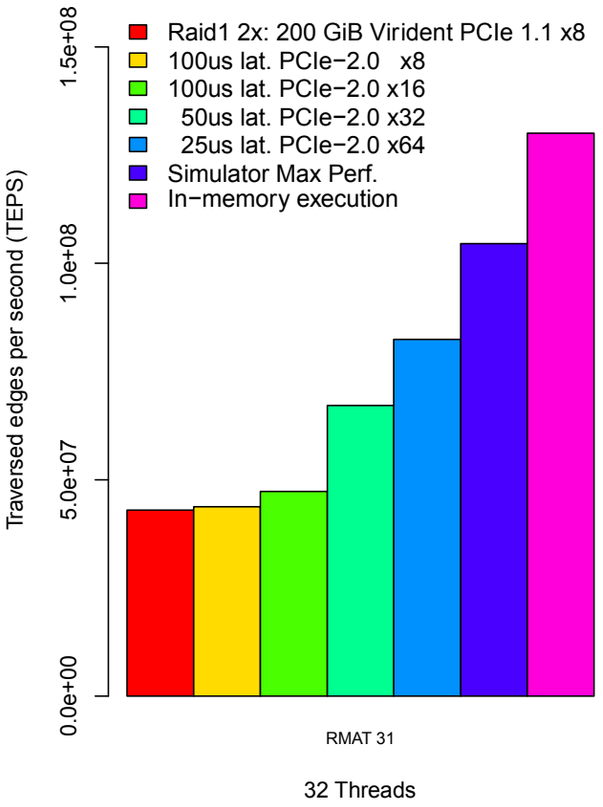
- **Simulation** and **emulation**
- Memory API to persistent memory
- Models latencies and bandwidth
  - Different memory technologies (NAND Flash, PCM, memristor, MRAM)
  - Different interconnects (SATA, PCIe, DIMM)

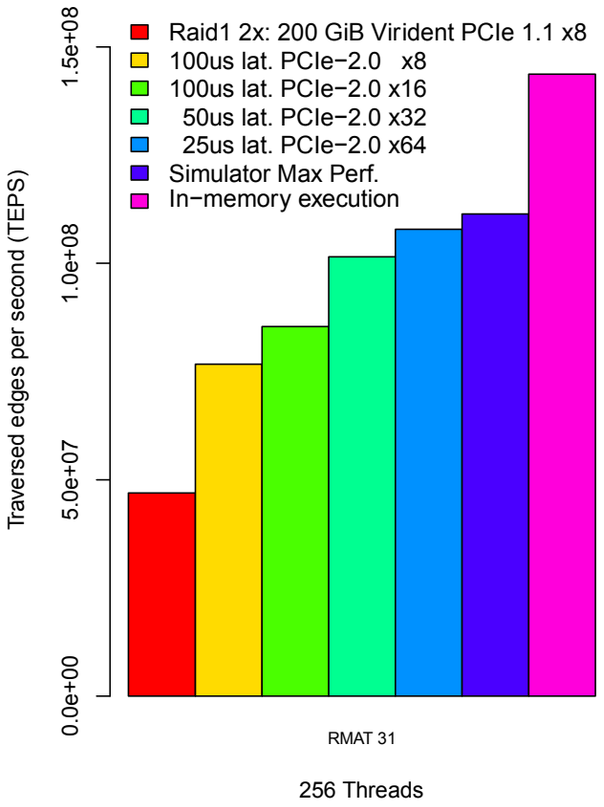


Persistent Memory Simulator



Hardware Exploration and Evaluation

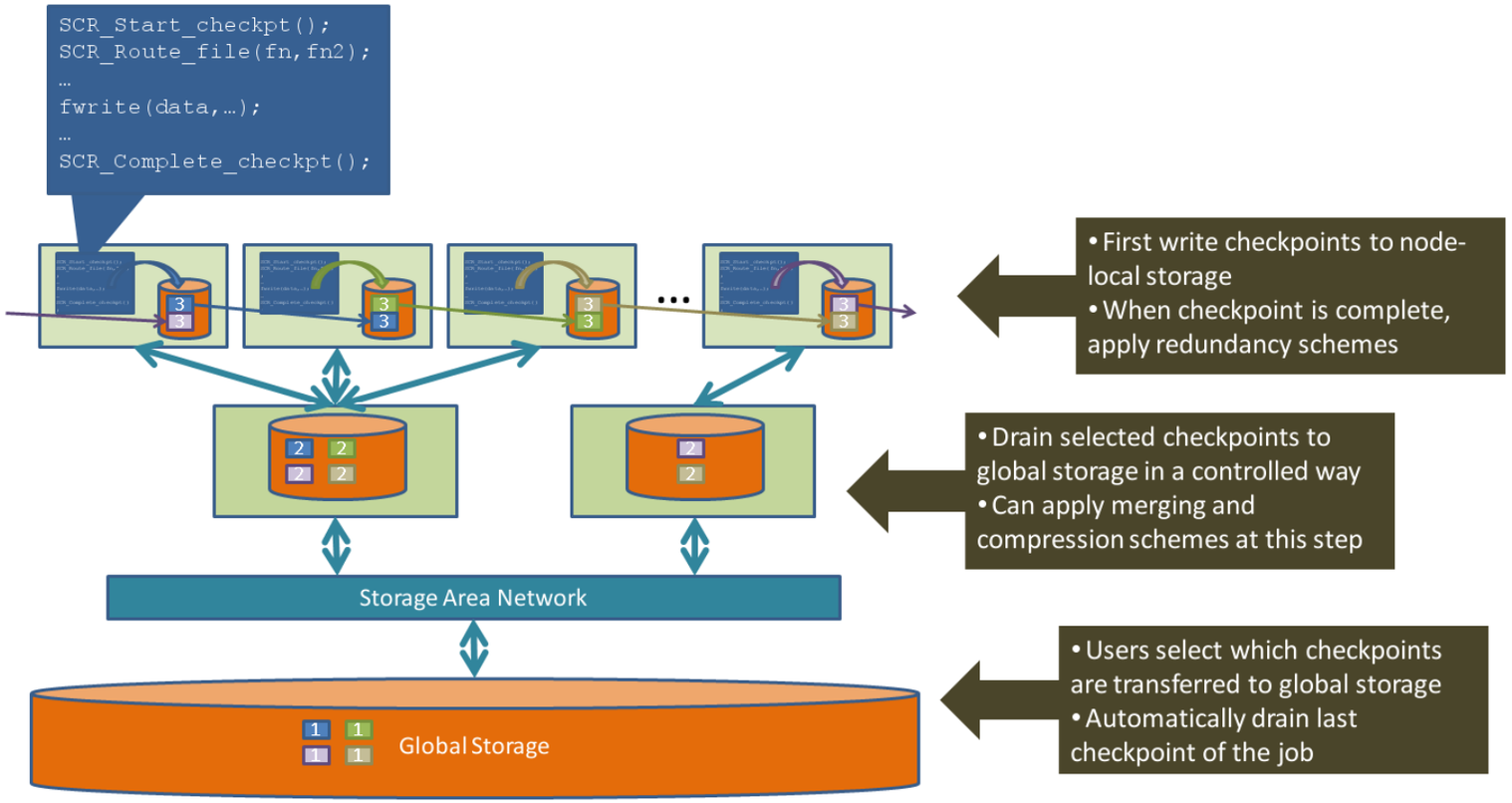




- Breadth first search graph traversal
- 170 GiB of data
- One and eight thread/core
- Comparison of
  - In-memory
  - PCIe-attached flash
  - Four generations of simulated NVRAM

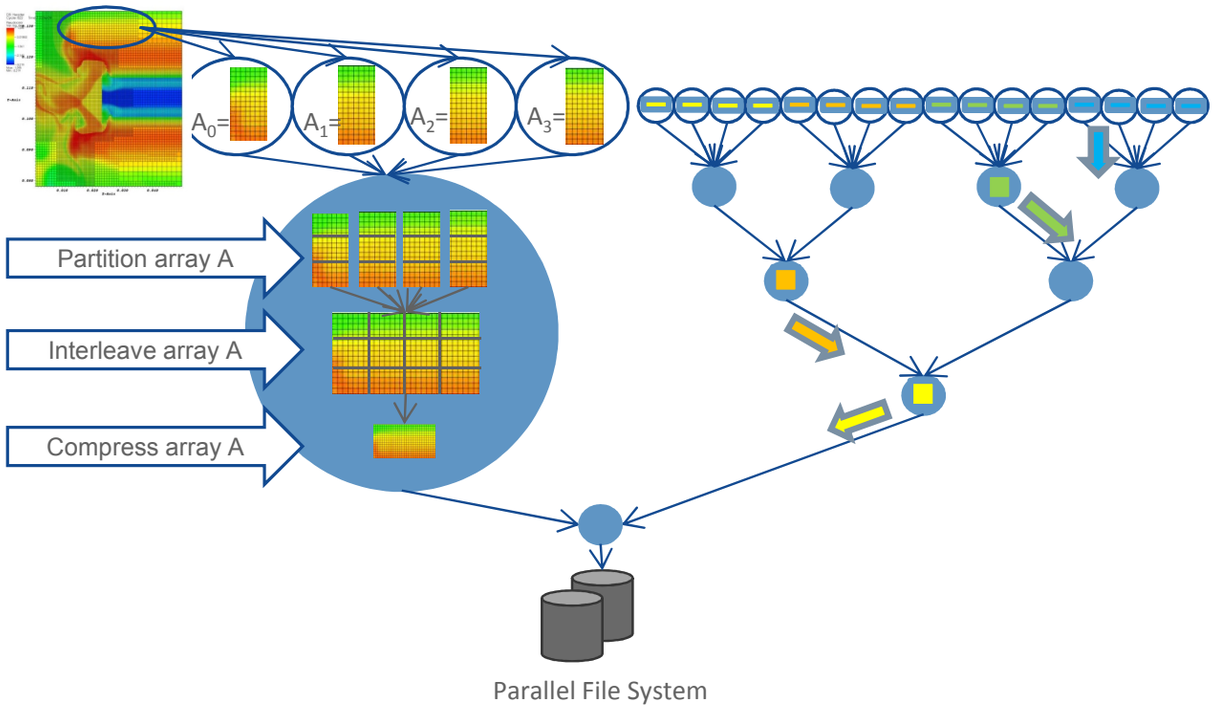
- Future NVRAM should achieve 75% performance of fully in-memory run

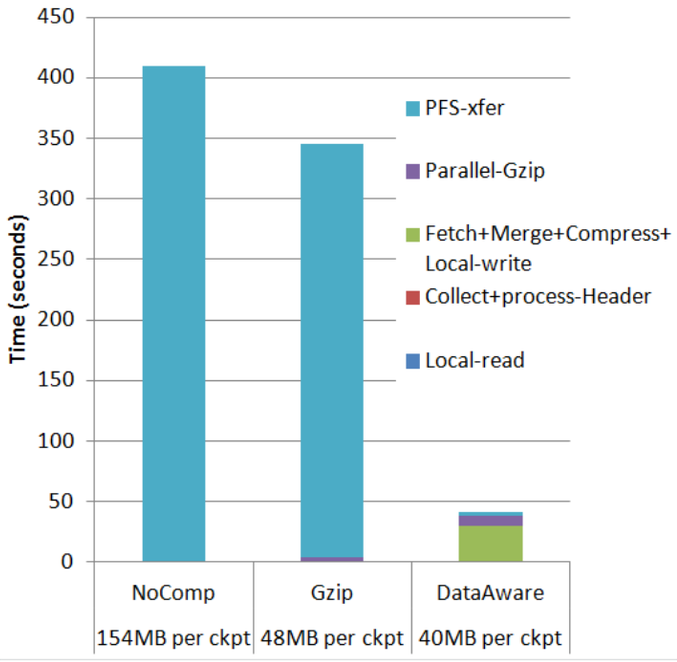
Checkpoint and Restart



- Based on **SCR**
- Utilize in-system storage and multiple checkpoint strategies for scalable, highly resilient checkpoint and restart

Data-Aware Compression

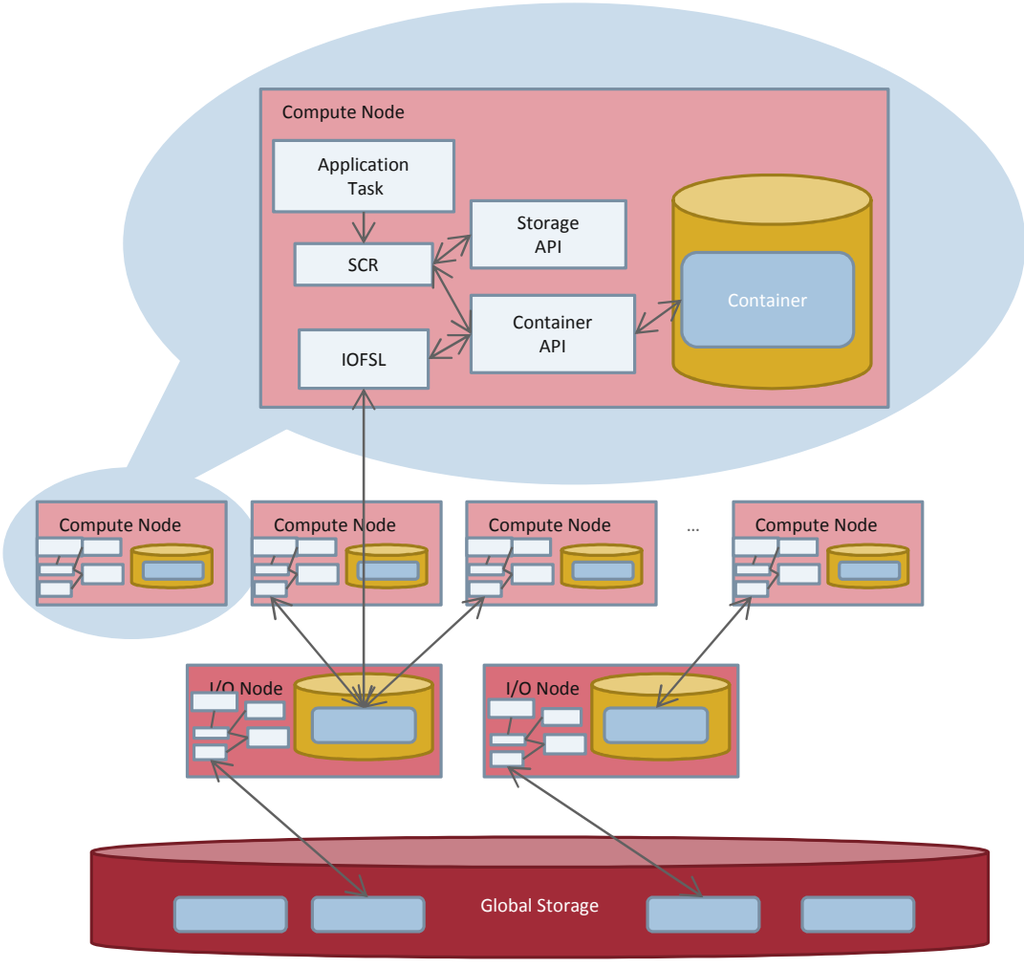




- Interleave like with like data from distinct processes before compression
- Move checkpoints asynchronously to minimize application overhead

Integration
(Overview)

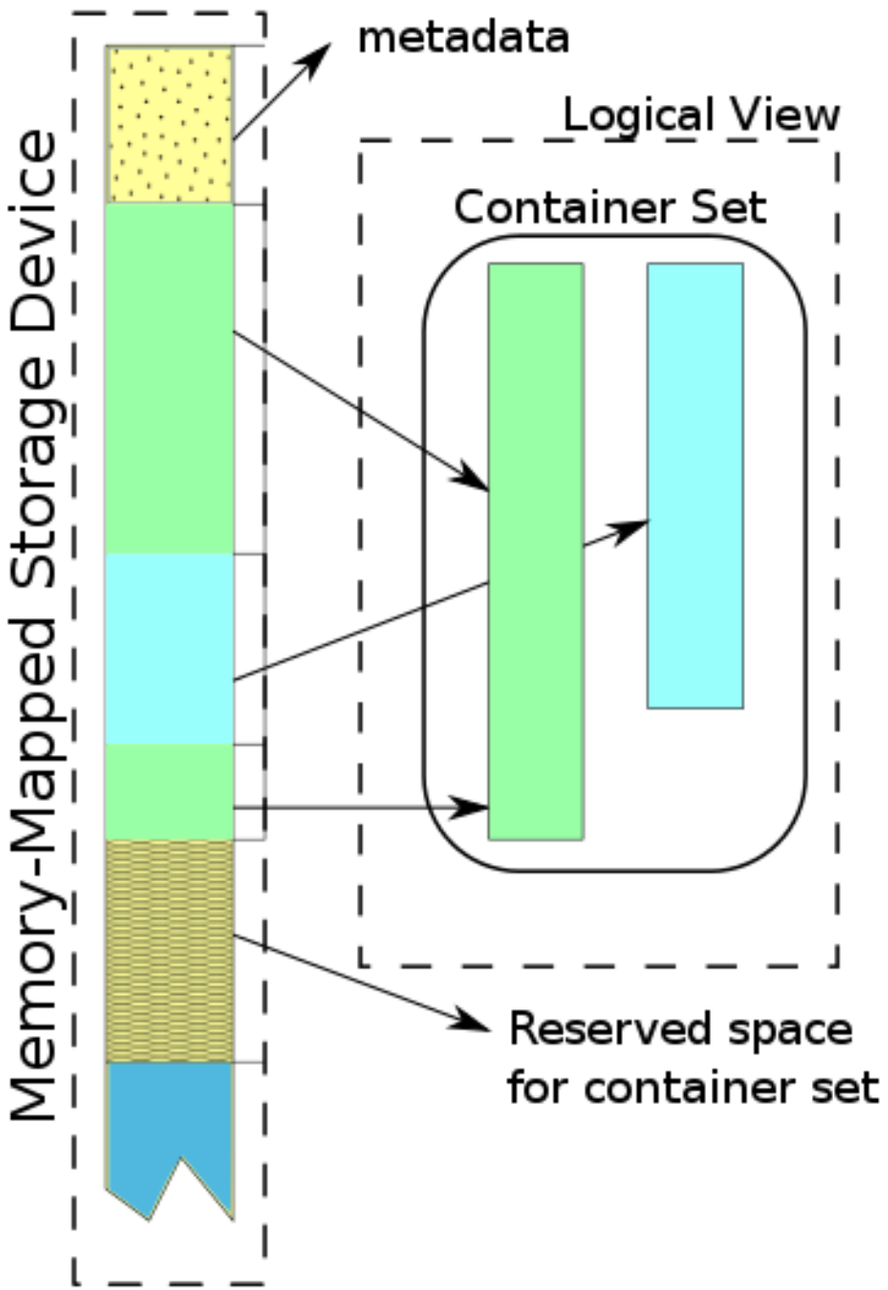
- Emulated NVRAM for now
- Storage abstraction API provides information on storage devices available in the system
- Container API provides lightweight management abstraction for in-system storage



Storage Hierarchy Abstraction

- Primarily static information on storage system configuration
- Designed for constant memory usage wrt the number of nodes
  - Query for resources as needed
  - Hierarchy of storage resources
    - Local, up, down
  - E.g., a Blue Gene I/O forwarding node can query for
    - local storage (nothing now; SSD in the future)
    - storage at the next level (parallel file system)
    - list of compute nodes that share the I/O node
- Each storage location has:
  - a URI for location, e.g., `storage://compute2/tmp`
  - a set of attribute-value properties, e.g., `capacity = 10E12`
- SCR can benefit from knowing
  - which processes share storage
  - persistency information
  - expected bandwidth

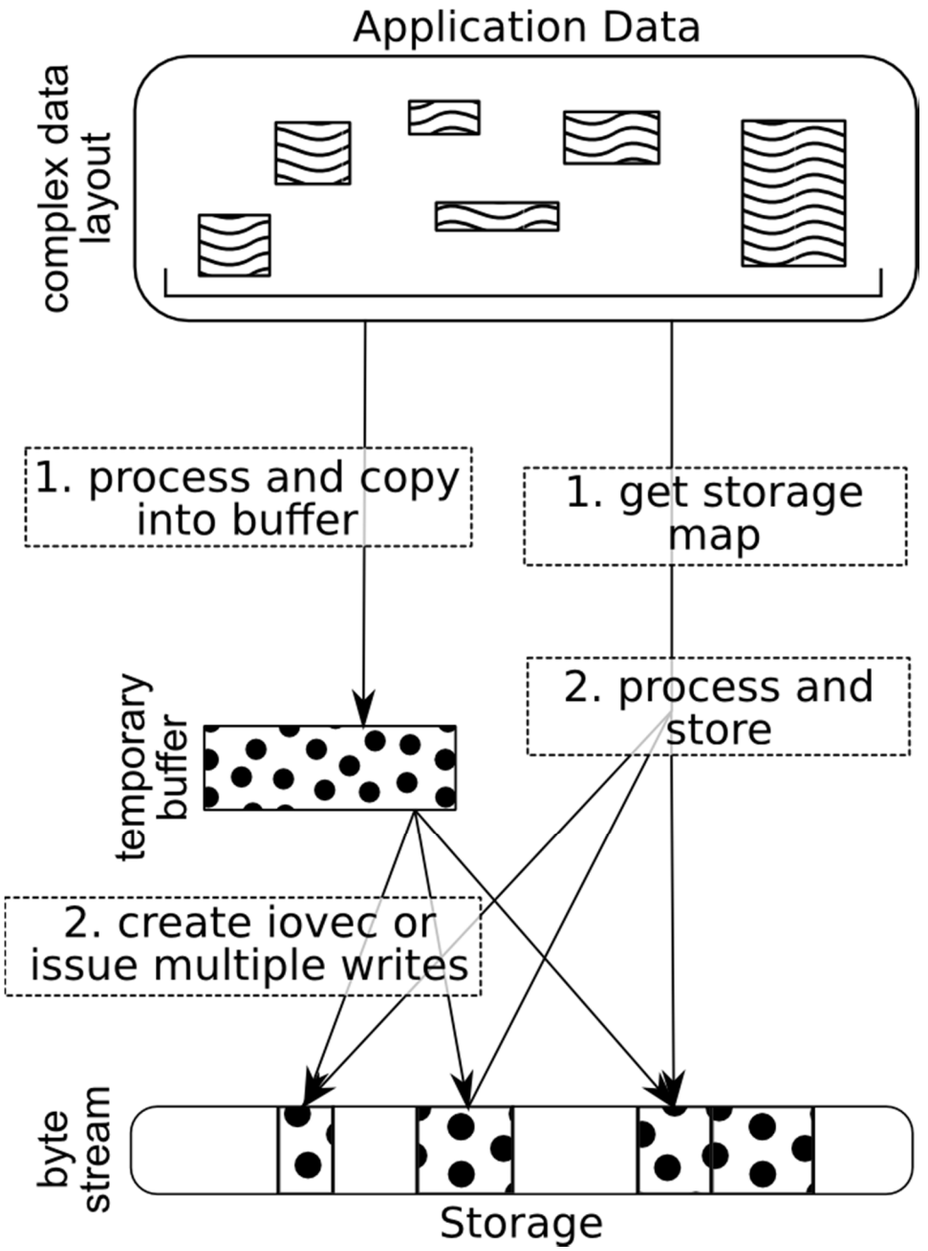
Container Abstraction



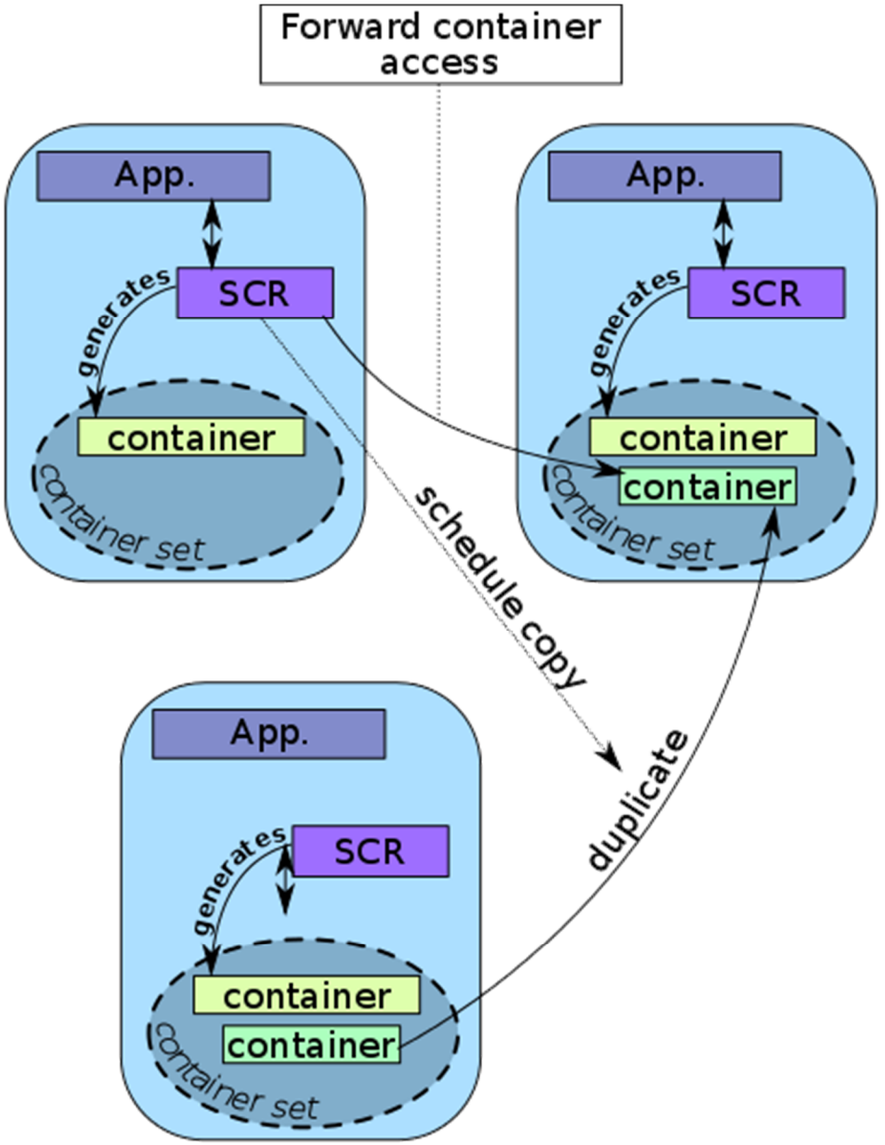
- Data model for in-system storage
  (designed for **memory mapped** storage)
- Why not a file on a local file system?
  - Functionality not needed for local temporary storage can come at a high cost.
  - Hard to add new capabilities
  - Require admin privileges
- User-space implementation simplifies **experimentation** and **deployment**
  (can run on flash emulator)
- Constructed for **direct storage access**
  (true zero-copy) – See next slide
- API overview:
  - `create`, `delete`, `rename`, get attributes, non-contiguous `read`, `write` and **explicit** `resize`.
  - Each container has a name
  - Containers grouped in sets for isolation, space reservation

Direct Storage Access

- Expose container storage layout
  - Storage format designed for direct access
  - **Application transfers** data
    - Avoid extra copy (processing data)
    - No complicated non-contiguous I/O description needed.
- Compare:
  - memory-mapped I/O (extra copy)
  - XIP (no write support, fs dependent)
  - direct-io (alignment restrictions, API bottleneck)
- Layout returned as set of pointers into storage.



Remote Container Access & Transport



- Containers are a purely **local concept**
  - **No global namespace**
- Some applications need **remote access**
  - **Use storage hierarchy abstraction to identify remote location.**
  - **Remote read/write**
- **Extension:** remote copy **operation**
  - **Request duplication of a container to another location**
  - **Remote source and dest (3rd party)**
  - **Global** scheduling **of data movement**
- Implementation using IOFSL (under construction)

Summary

- Provided an overview of a prototype, integrated in-system storage architecture
  - SCR, IOFSL, PerMA, abstraction layers
- Code under active development
  - Will be open sourced
  - Storage abstraction and container API developed as standalone libraries to encourage reuse

- This work is supported by US DOE ASCR within the NoLoSS project