Argonne NATIONAL LABORATORY

Thinking Past POSIX: Persistent Storage in Extreme Scale Systems

Rob Ross, Phil Carns, Kevin Harms, Dries Kimpe,
Rob Latham, and Sumit Narayan
Argonne National Laboratory
rross@mcs.anl.gov

(Some) Collaborators:

Lee Ward (SNL)
Anthony Skjellum (UAB)
Narasimha Reddy (TAMU)
Dave Goodell (now at Cisco)
Brad Settlemyer, Steve Poole (ORNL)
Richard Brooks (Clemson)
C. Karakoyunlu, J. Chandy (UConn)

U.S. DEPARTMENT OF ENERGY

What's wrong with POSIX?

- Data Model – difficult to map complex, distributed data sets into single "stream of bytes"
- Storage Model – no notion of locality
- Consistency – very strong consistency forces heavy-weight algorithms, makes it difficult to extract high performance

What's wrong with today's parallel file systems?

- Expensive – typically rely on expensive underlying hardware
- Inflexible – not built to support a variety of application abstractions
- Fragile – poor fault handling
- Inefficient – poorly utilize new fast devices, heavy-weight consistency management (arguably this last isn't their fault)

What are we doing?

- Investigating arch. and models for extreme scale storage
  - Applicable across many domains
  - Learning what other communities have already learned
- Building some things
  - Tools for observing I/O behavior
  - Simulations of critical components
  - A prototype and upper layers for demonstration of concepts
- Releasing code as open source
  - Some components available already
  - More as things mature

- Today I'm going to just talk about our work in progress extreme scale storage system

Common Capabilities in Data/Storage Products

	Scalability	Fault Tolerance	Concurrent Reads	Concurrent Writes	Synchronization Primitives	Atomicity	Layout Control	Record Oriented Access	Data Affinity
Parallel File System *Lustre, GPFS, Panasas, PVFS, Ceph*	✓	~	✓	✓	✓	✓			
Cloud Object Storage *Amazon S3, Rados Gateway*	✓	✓	✓			✓			
MapReduce *Hadoop HDFS, Google GFS*	✓	✓	✓				✓		
Key/Value Store *Dynamo, Cassandra, Hyperdex*	✓	✓	✓	✓	✓	✓		✓	
NoSQL Database *MongoDB, BigQuery*	✓	✓	✓		✓	✓		✓	✓

With help from C. Karakoyunlu (UConn).

A Clean Slate…

- User View
  - Data model – what is stored
  - Storage model – view of the system and components
  - Consistency model – what is guaranteed
- Architecture
  - Communication
  - Distribution mechanism(s) – enabling concurrency
  - Resilience mechanism(s) – ensuring data integrity and accessibility
  - Consistency mechanism(s) – enforcing consistency model
  - Security mechanism(s) – protecting data from undesired access
  - Local storage management – mapping data to devices
- Lots of viable design points…

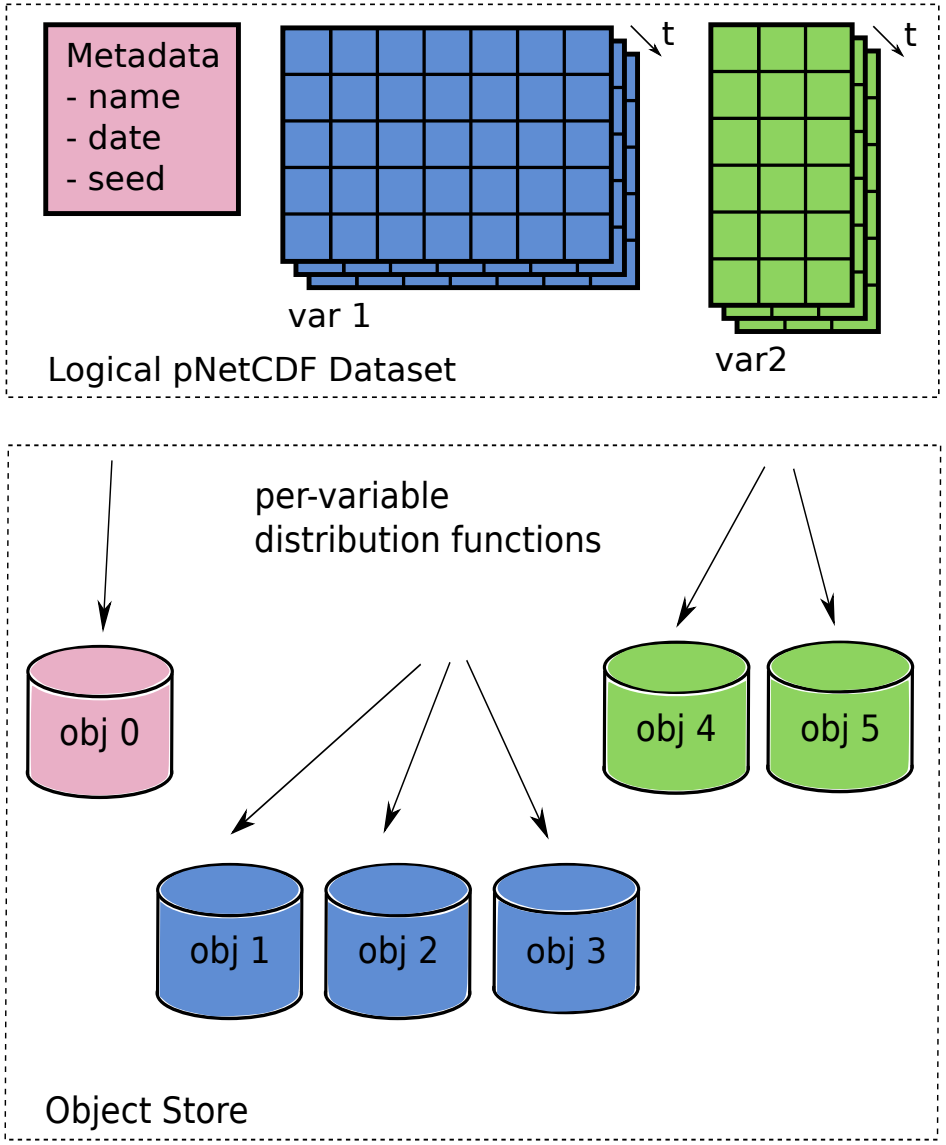
“User” View

User View

- Provide a flexible “substrate” for development of extreme scale data services
  - Not particularly things that look like file systems
  - Keyword/value
  - Bulk data
  - Multiple streams from single application
  - Consistency model that rewards good behavior
- “User” isn’t the end user
  - Envision software layers atop that map to domain of interest
- Pursuing an object-based storage approach
  - Alternative to block-based (or file-based) storage
  - Objects are logical collections of bytes with identifiers
  - Component that exposes object API is responsible for mapping to HW

PnetCDF Mapping to Objects

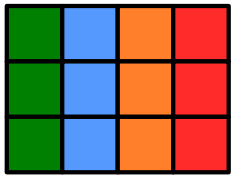
Variables mapped into distinct objects. Resizing of one variable has no impact on others.

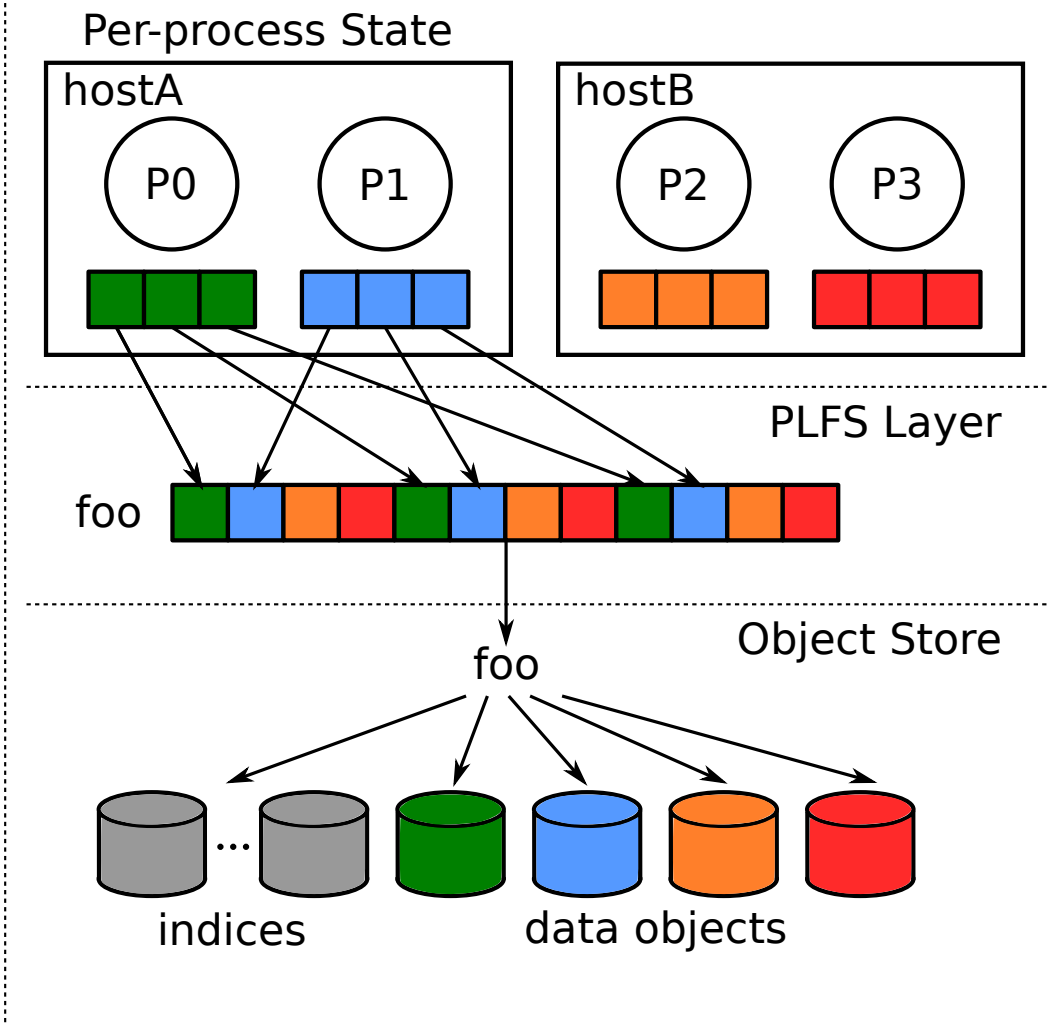


PLFS Mapping to Objects

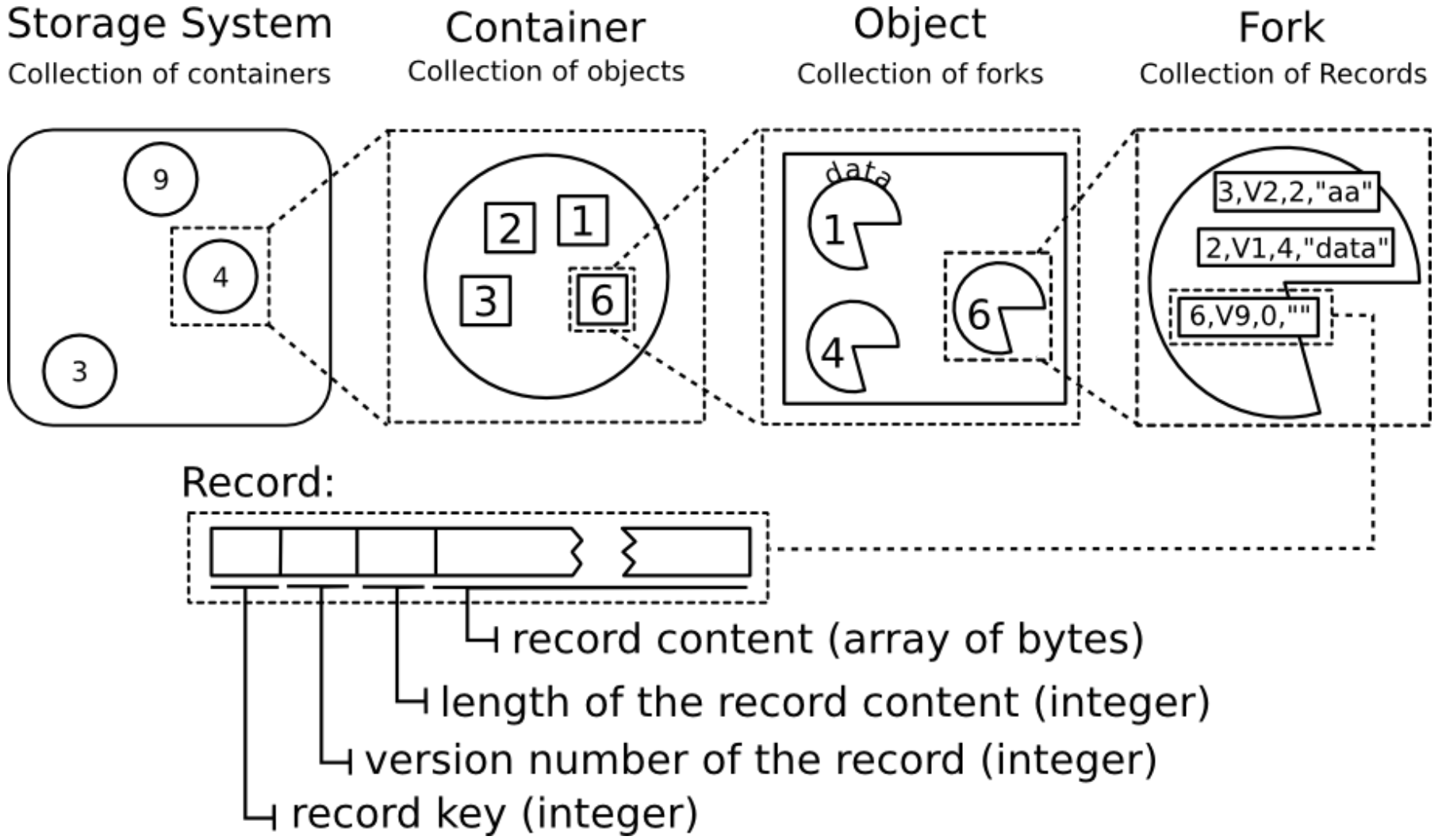
Data from a process lands in a unique object; overhead to create objects much lower than overhead to create files.

Global Application State





Data Model



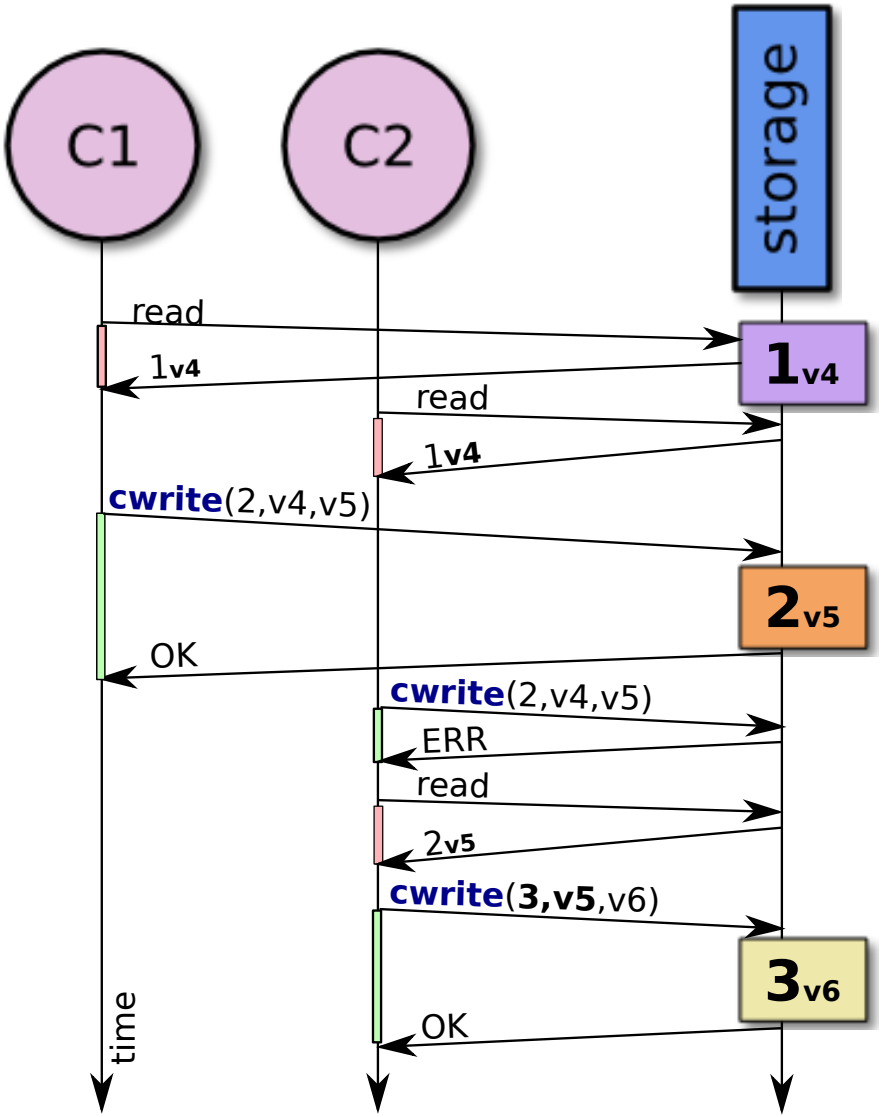
Note: This is a *somewhat* arbitrary number of “levels”…

Data Model Operations and Consistency

- A limited set of operations:
  - **Write**: overwrite one or more records (**atomic**)
  - **Read**: retrieve one or more records (including metadata)
  - **Probe**: only retrieve metadata (version and length etc.)
  - **Reset**: Sets the entity back to the default state (i.e. `erase')

- Versions:
  - Client generally provides (record granular) version number; API also supports auto increment
  - Version is used to order transactions (no retrieval of obsolete versions)
  - Write, read and punch support **conditional execution** based on the expected version

Updating with Conditionals



Algorithm:

- Augment storage to provide **write conditional**
- Read returns version associated with record
- **Write is conditional on match of version at server**

Notes:

- **Optimistic: No significant overhead when there is no contention** (other than storing versions)
- No state shared between clients and servers, simplifies fault handling
- Not necessarily fair when contention occurs

User View Wrapup

- Object model isn't all that controversial, really
- Optimizations for "strings of one byte records" make this efficient for byte streams
- Records make it possible to map key:value stores naturally
- **Goal is a single model that is broadly applicable**

- Open questions:
  - Do we really need forks?
    - Or, do we need more flexibility in depth of hierarchy?
  - Will we need "privileged" forks or records to store things like security information?
  - Can breadth of use cases be supported with only one or two implementations?
    - Or will there be a performance portability problem when moving between?
  - Do we need distributed transactions?

Architecture

Storage Service Architectures

- Communication
- Data distribution
- Resilience
- Consistency
- Security
- Local storage management

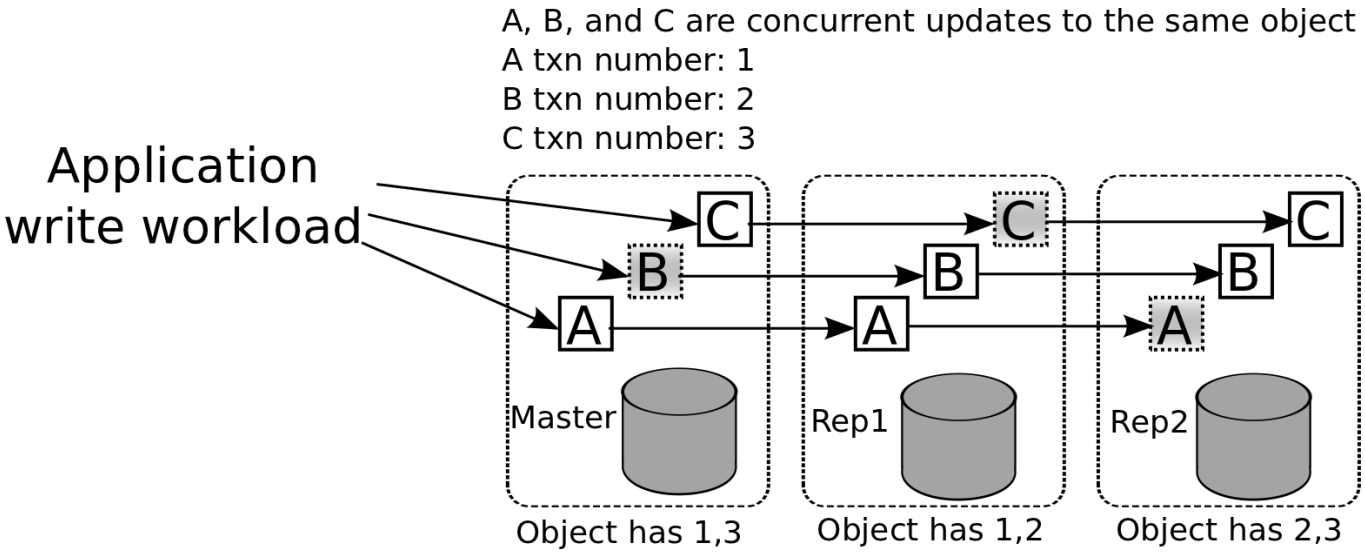
- Assumptions:
  - For cost reasons, we have a set of servers with local storage attached
  - Object data model described previously

Communication

- Most persistent services are built using RPC
- Most RPC systems weren't designed for HPC (or even modern) interconnects
- Mercury RPC project addressing this
  - http://trac.mcs.anl.gov/projects/mercury
  - Jerome Soumagne (The HDF Group) leading development
- More on this tomorrow from Dries!

Replication and Consistency

- Primary-copy with a twist
- All writes are assigned a unique transaction number (scoped to an object, distinct from versions in data model)
  - Txn numbers are stored persistently with object
  - Defines an ordering to determine which write “wins” if there is overlap
- All writes are atomic
  - The txn number defines the scope of the atomic write operation
  - An entire write is visible to readers completely or not at all
- Data can be applied or relayed in any order



Semantics Across Replicas (Bonus Material)

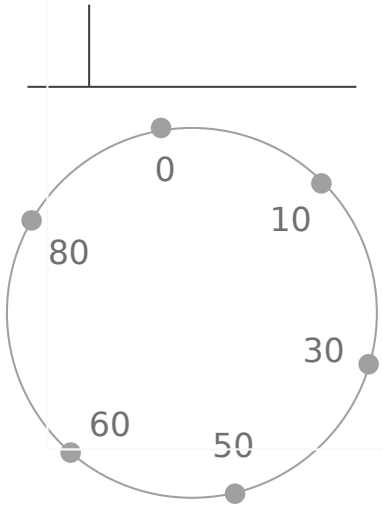
- Previous slide looks like eventual consistency (plus some features)
- But we are going one step further, based on Pu's Epsilon Serializability ESR rules
  - COMMU or RITU replica control
  - Writes can be applied in any order at each replica
  - Eventual consistency (determined by txn number)
  - But with strict rules bounding divergence!
- What are the bounds in Triton? By default:
  - Replicas can diverge as long as concurrent operations are in progress from clients
  - Once a write completes from client perspective, it is guaranteed to be visible and in correct order relative to any subsequent reads or writes
- Strong enough for MPI-IO and PVFS semantics, but loose enough to prevent serialization during write bursts
- All replicas consistent if client does "writes(); barrier(); read();"
- Tunable semantics with minor tweaks to protocol:
  - Tighten by serializing reads and writes at master server (more expensive)
  - Loosen by allowing different (or no) bounds on eventual consistency (less expensive)

Data Placement

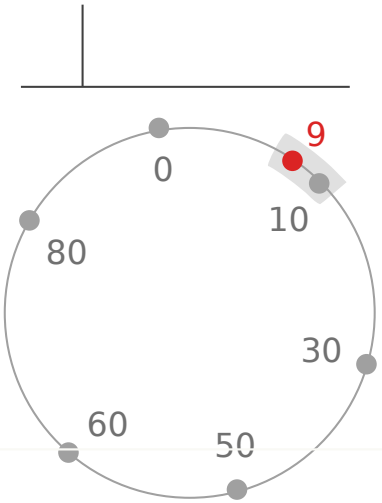
- At a high level, we are either:
  - algorithmically placing data (objects and ECC/replicas), or
  - Placing data in an unstructured manner
- Algorithmic distribution
  - Simplifies metadata, makes things easy to find
  - Limits flexibility of placement
    (although you can perhaps pick IDs that go where you want)
  - *Can* simplify triage (determining impact of a fault)
    - Need to be able to reason about what could have been on a server
- Unstructured distribution
  - Provides great flexibility in placement (put things wherever you want)
  - Requires either indexing or searching to find things
  - Similarly requires indexing for triage, or routine scanning

Resilience and Algorithmic Data Placement

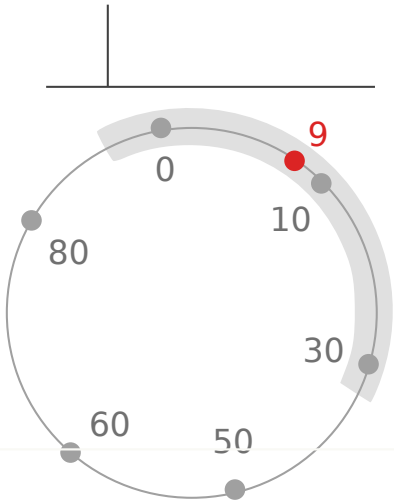
Servers (grey) are arranged in an n-dimensional address space and referenced by an ID in that space. Here, n=1.



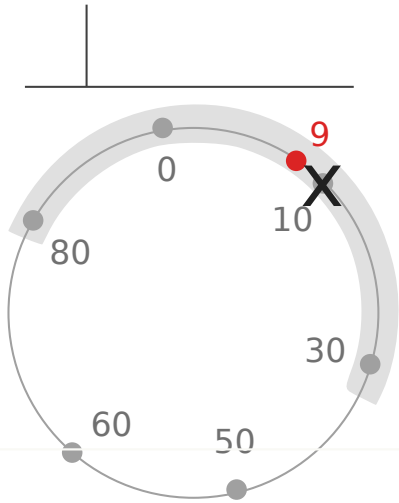
Objects (red) are likewise addressed by an ID in this space. The primary for an object is located at the server with the closest ID in the address space.



For a replicated object, replicas are placed on the k-1 next closest servers in the address space.



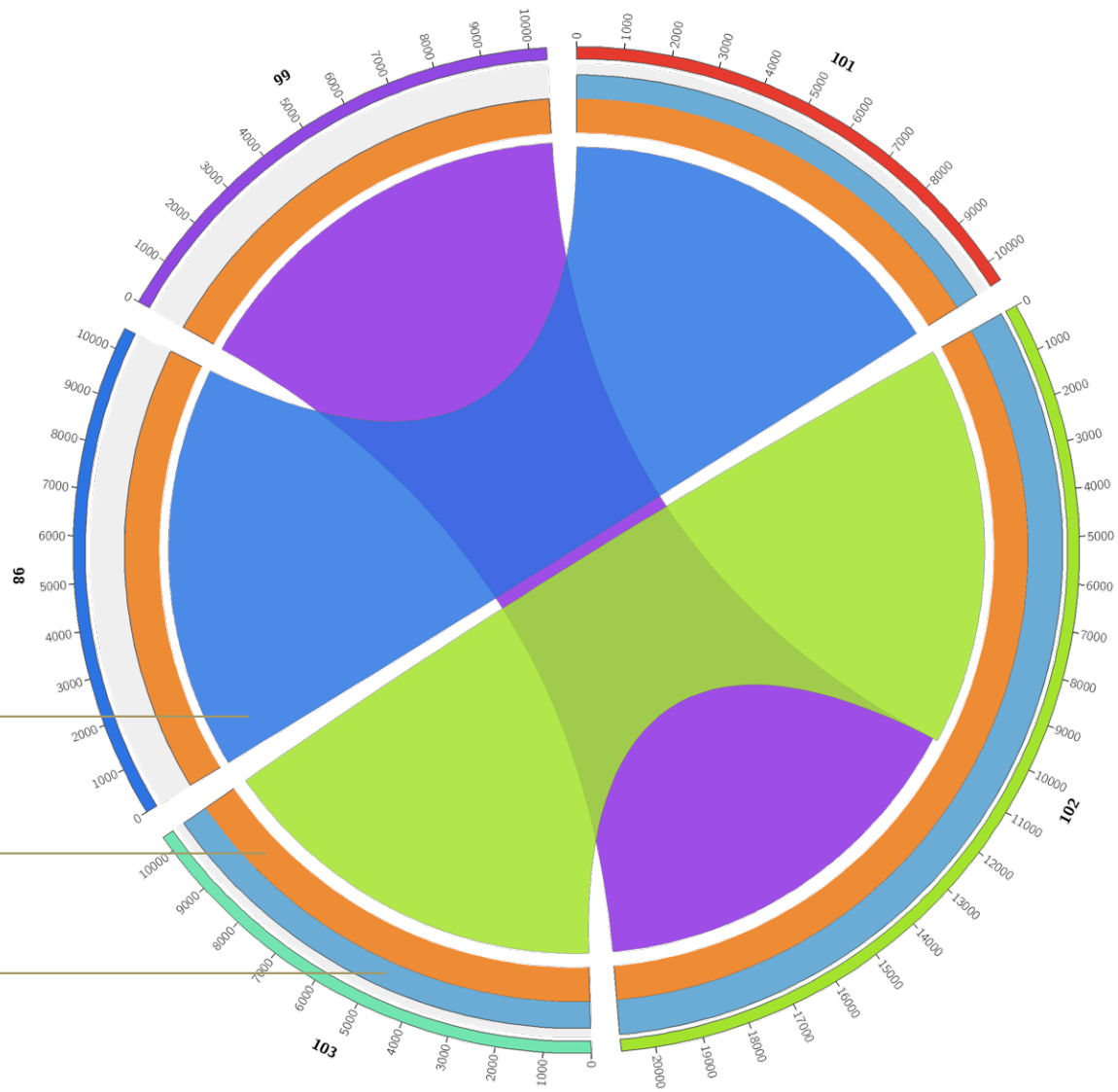
In the event of a server failure, the object will be re-replicated to the next closest server in the address space.



Data Distribution and Rebuild Behavior

Looking at how triage (what needs fixing) and rebuild (fixing it) proceed using different schemes for distributing replicas in distributed storage.

- 192 servers
- 1 server fails
- What happens?



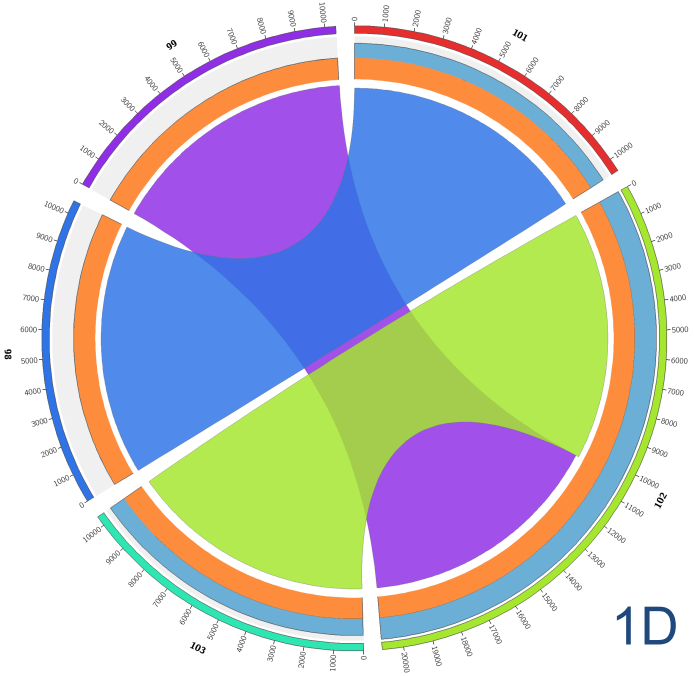
Arcs represent servers that sent/received data.

Ribbons indicate data movement (color indicates source).

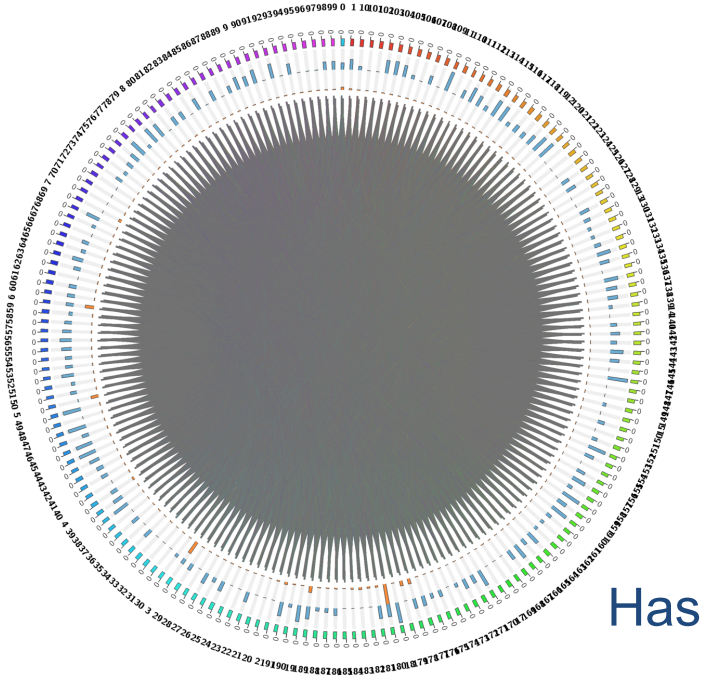
Orange bar depicts triage time.

Blue bar depicts rebuild time.

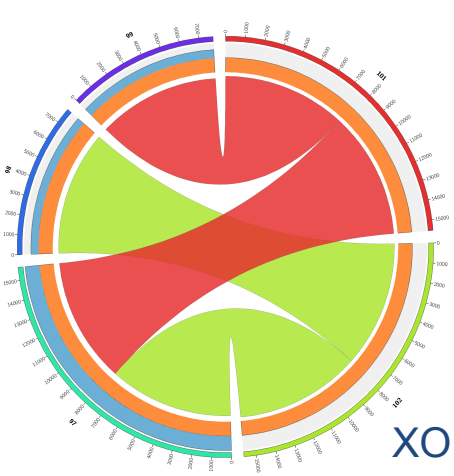
Algorithms and Distribution of Work



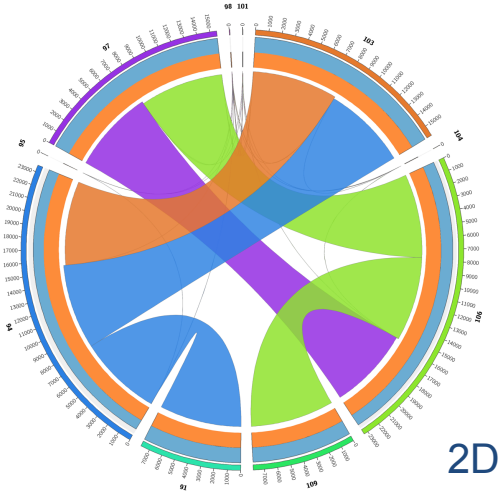
1D



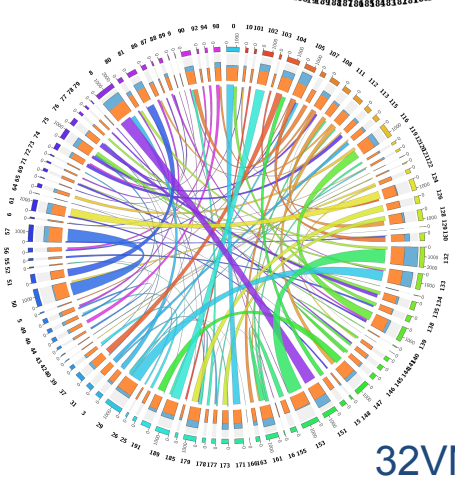
Hash



XOR



2D



32VNodes

Looking for balance of rebuild, limited rebuild time, limited perturbation of system, and also ability to place data and coordinate concurrent I/O.

Architecture Wrapup

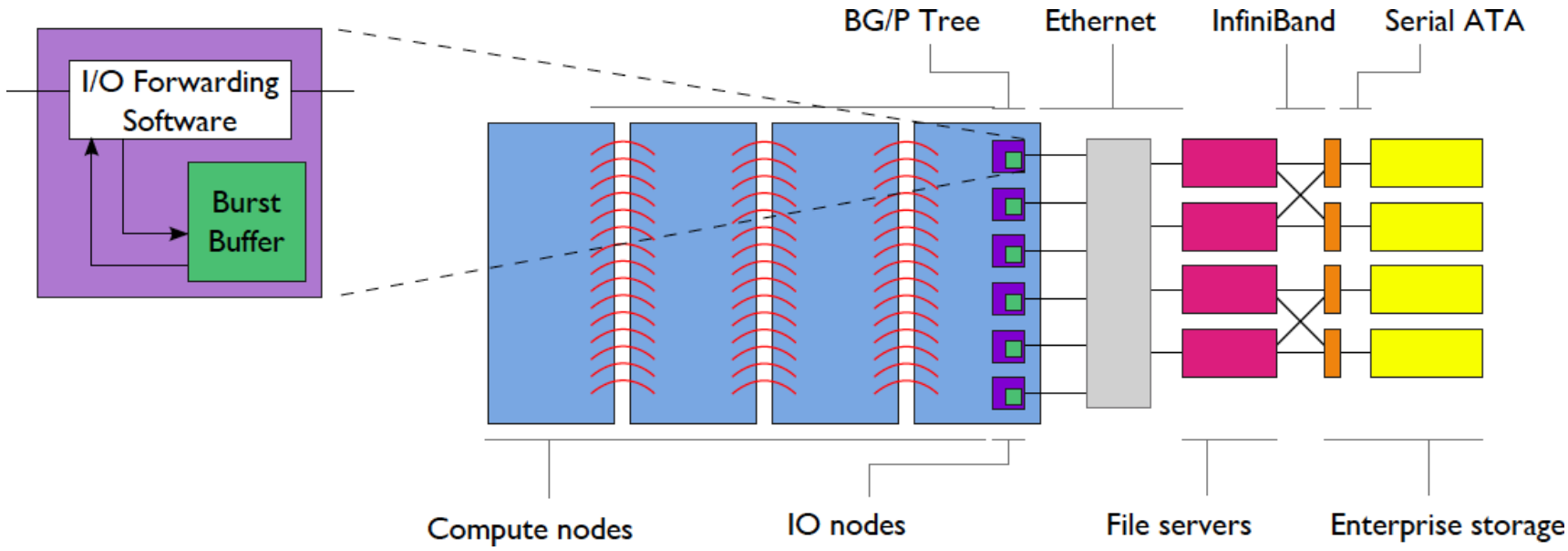
- I thought that I would be short on time by this slide, so I didn't discuss security or local storage
  - Security model looks like the LWFS security model
  - Local storage is log based:

    P. Carns, R. Ross, and S. Lang. Object Storage Semantics for Replicated Concurrent-Writer File Systems. In Proceedings of the Workshop on Interfaces and Abstractions for Scientific Data Storage. September 2010.
- Current work is focusing on finding the "right" distribution algorithms for expected system sizes, object characteristics, and fault rates
- Sort of a mash-up of Dynamo (Cassandra, Riak), Pu's Epsilon Serializability, and distributed object storage

Known Unknowns

Storage Model: Topology and Device Properties

- System must expose properties and allow for tuning
- Approaches like CRUSH incorporate some notion of topology and an ability to non-uniformly distribute data
  - But difficult to guarantee even placement when needed
  - And no notion of device performance in model



Burst buffers are just the beginning of more complex storage hierarchies.

Distributed Transactions

- Other teams seem convinced that distributed transactions are necessary.
- Are we missing something?

Transactions

Consistency and Integrity

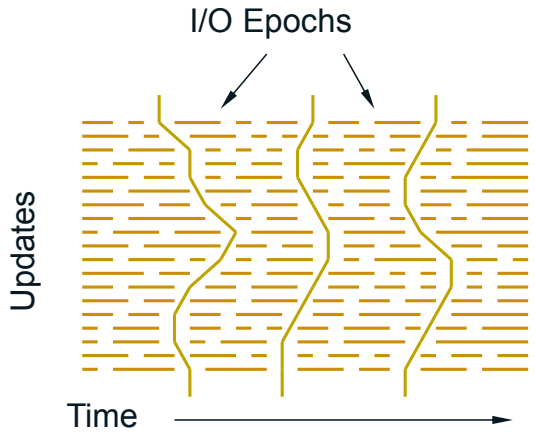
- Guarantee required on any and all failures
  - Foundational component of system resilience
- Required at all levels of the I/O stack
  - Metadata at one level is data to the level below

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



Eric Barton, "Fast Forward I/O & Storage," PDSW 2012, November 2012.

Big Picture

- Coordination
  - See Matthieu's talk tomorrow morning
- Optimization and adaptivity
  - See Babak's talk tomorrow morning, and
  - Emmanuel's talk on Wednesday morning
- Resilience
  - See mini workshop talks tomorrow

For more information …

- C. Karakoyunlu, D. Kimpe, P. Carns, K. Harms, R. Ross, and L. Ward. **Towards a unified object storage foundation for scalable storage systems**. In Proceedings of the Fifth Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS), September 2013.
- J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross. **Mercury: Enabling remote procedure call for high-performance computing**. In Proceedings of the IEEE Cluster Conference, September 2013.
- D. Goodell, S. J. Kim, R. Latham, M. Kandemir, and R. Ross**. An evolutionary path to object storage access**. In Proceedings of the 7th Parallel Data Storage Workshop (PDSW 2012), Salt Lake City, UT, November 2012.
- P. Carns, K. Harms, D. Kimpe, J. M. Wozniak, R. Ross, L. Ward, M. Curry, R. Klundt, G. Danielson, C. Karakoyunlu, et al. **A case for optimistic coordination in HPC storage systems**. PDSW 2012, November 2012.
- D. Kimpe, P. Carns, K. Harms, J. M. Wozniak, S. Lang, and R. Ross. **AESOP: Expressing concurrency in high-performance system software**. In Proceedings of the 7th International Conference on Networking, Architecture and Storage (NAS), pages 303–312, Fujian, China, June 2012.
- P. Carns, R. Ross, and S. Lang. **Object storage semantics for replicated concurrent-writer file systems**. In Proceedings of the Workshop on Interfaces and Abstractions for Scientific Data Storage, September 2010.

Acknowledgments

This research was in part supported by the United States Department of Defense and by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy.