

Revisiting Data Models for Data-Intensive Scientific Computing

Dries Kimpe dkimpe@mcs.anl.gov

Argonne National Laboratory

Overview

- Introduction & Context
- Short introduction to
 - Trident
 - Container Abstraction
- Triton Data Model
 - Overview
 - Operations
 - Examples
- Conclusion
 - Open Questions
 - Future work

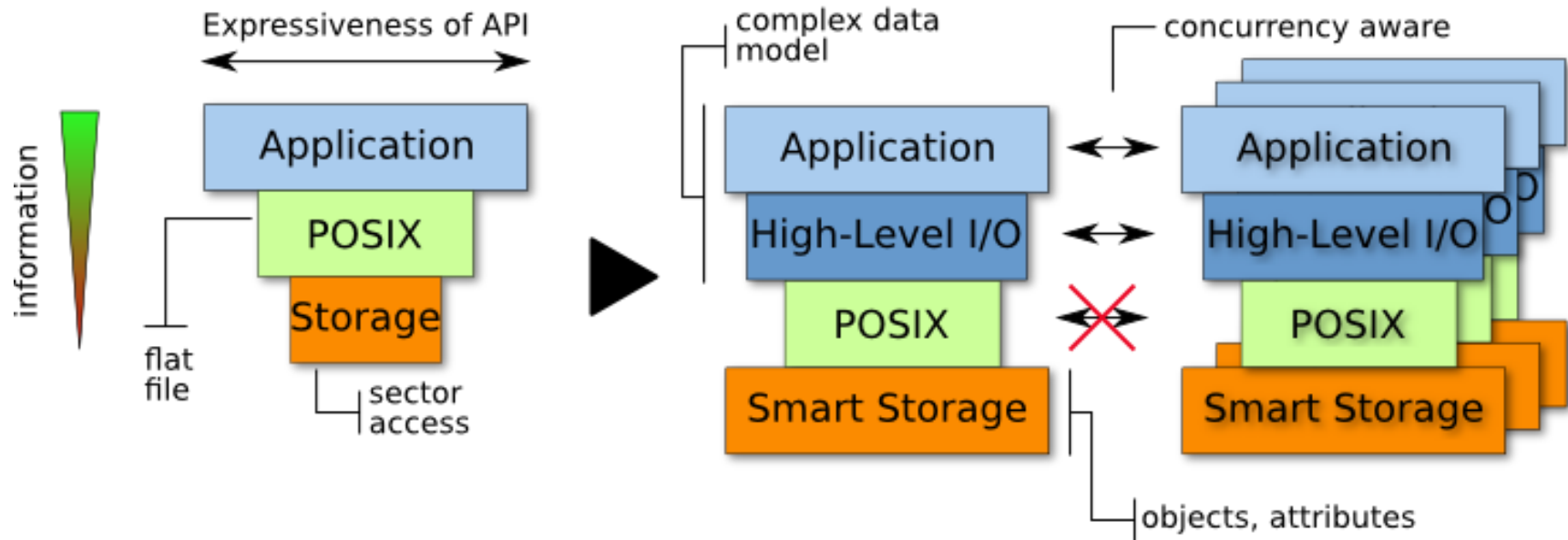


Context

- Many novel data-intensive storage models have emerged (mainly from the cloud computing and internet services community)
 - NoSQL, Column-Store, Map-Reduce, Object-Storage, Key-Value
- Why? Limitations of legacy storage models with respect to scalability, resiliency and expressiveness.
- This talk:
 - Present some alternative data models under exploration at Argonne.
 - Goal: Interact with audience to assess interest and usability.
 - Feel free to interrupt or ask questions at any time!
 - Target audience: applications/library not afraid to experiment
 - Ongoing work: things might change



Why a new data model?



- Model? (!= API, != Implementation)
- POSIX I/O API dates from ~1970: Plenty of research above and below POSIX but relatively little changes to POSIX (POSIX HPC extensions?)
- High-Level Libraries adapt to the application's data model but are more and more restricted by the POSIX API.
- The landscape changed: smart (object) distributed storage, application concurrency (need for scalable synchronization primitives and metadata operations)

Known Problems

Model Issues



Implementation Issues

- File locking & synchronization (inter-node synchronization)
 - Implementing MPI-IO shared file pointer
 - Manipulate meta data in high level I/O data formats (HDF5)
- Mapping application model to the file model (flat file)
 - Chunking, space efficiency, unlimited dimensions, ...
- Scalable metadata operations
 - Readdir + stat (`readdirplus`)
 - Generic namespace support
 - POSIX HPC Extensions (open by handle now in linux kernel)
- File partitioning
 - N-N / N-1 / N-M writing
- File Provenance

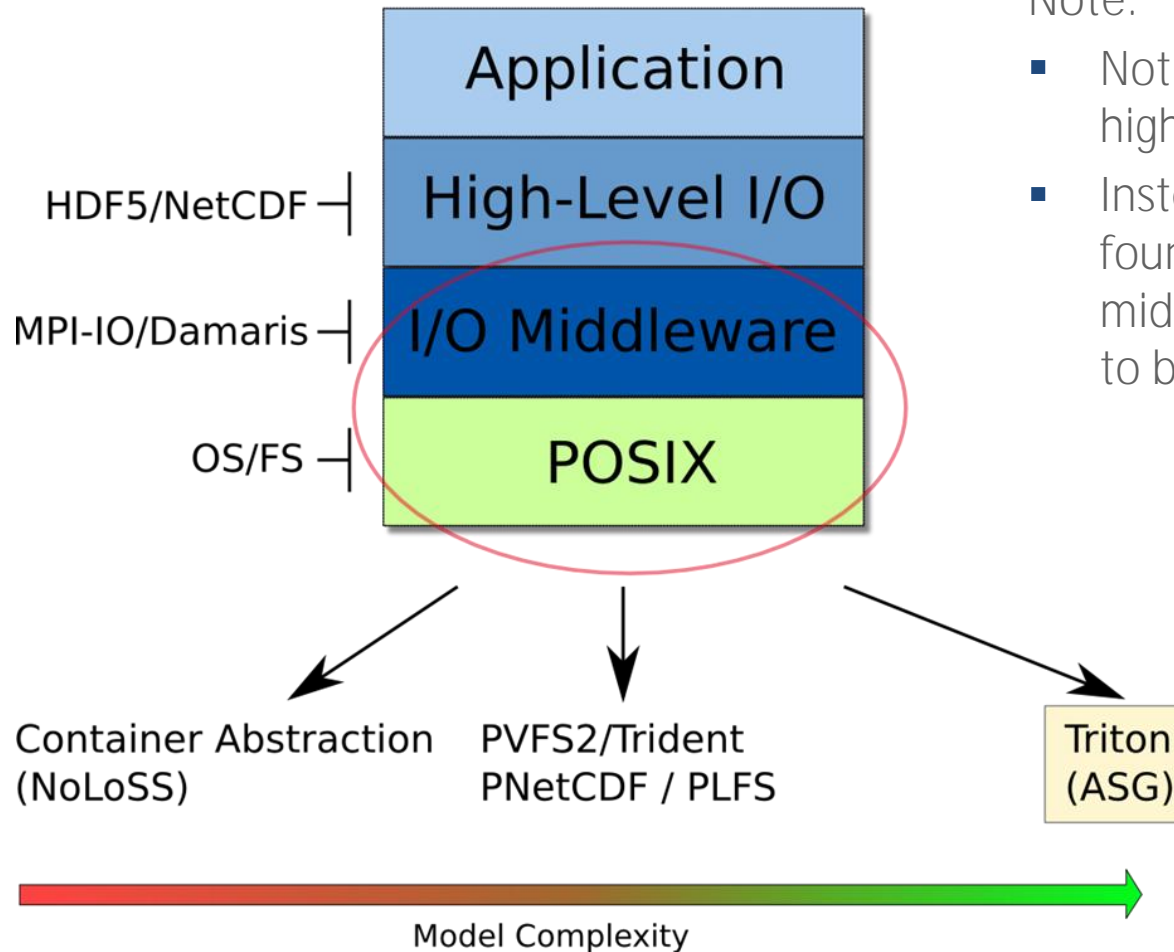


Situation of this work

Related Research at Argonne

Note:

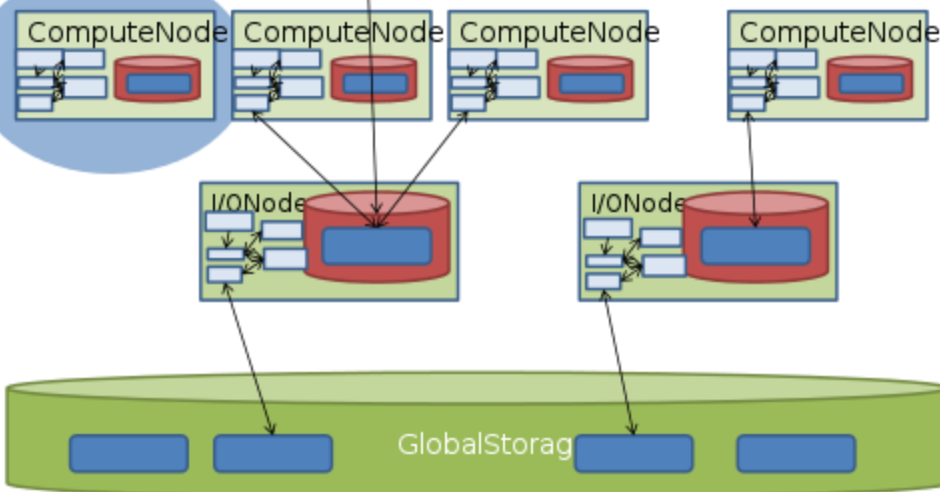
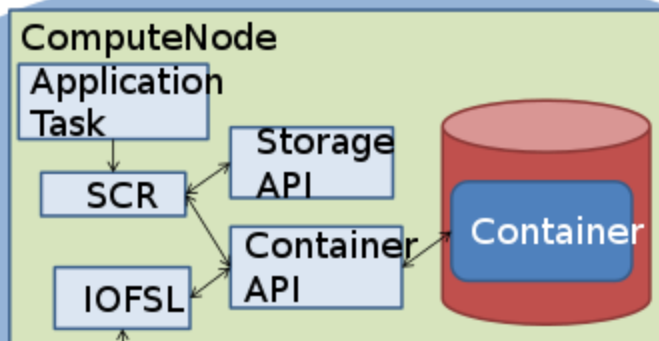
- Not trying to create another high level I/O library
- Instead provide new foundation for I/O middleware and high level I/O to build on.



In-System Storage Abstraction: Storage Containers



Container Abstraction (NoLoSS Project)



- Designed for **in-system** storage
- Expects **memory mapped** storage hardware.
- Targets **checkpointing, staging, in-situ** analysis
- Currently porting **SCR**
- People:
 - **LLNL**: Maya Gokhale, Kathryn Mohror, Brian Van Essen, Adam Moody, Bronis de Supinski
 - **ANL**: Kamil Iskra, Dries Kimpe, Rob Ross

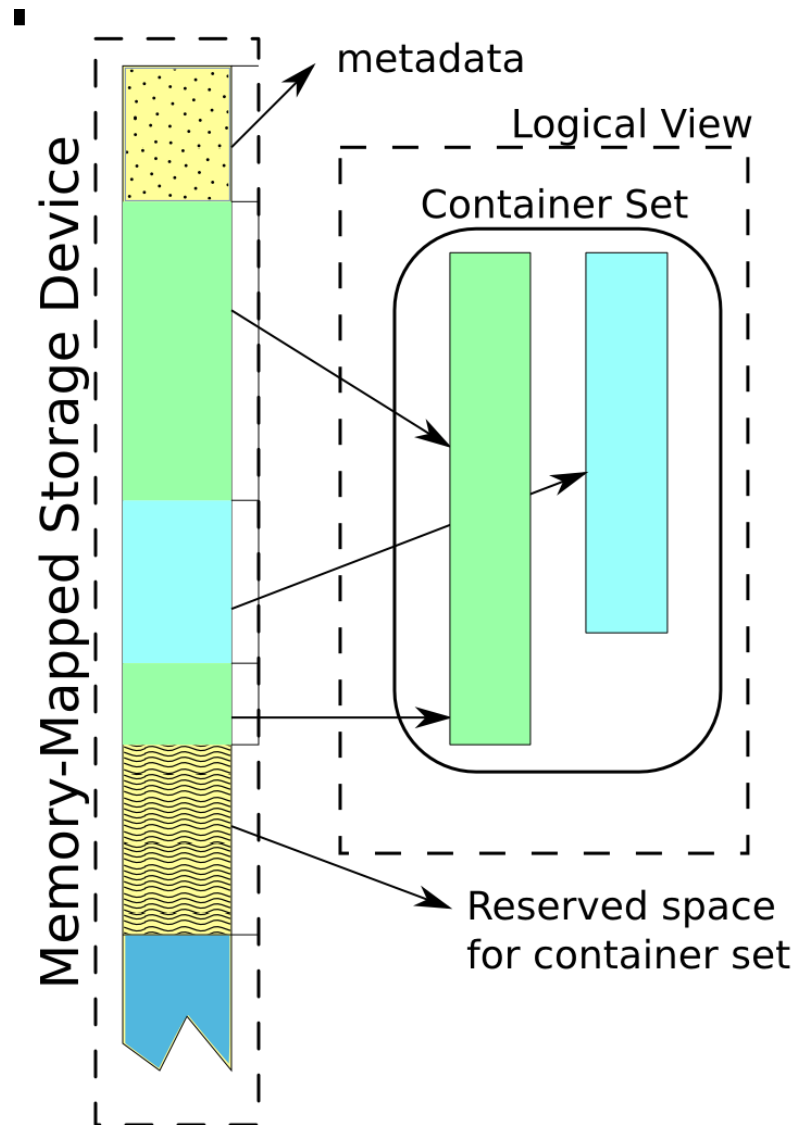
Integrated In-System Storage Architecture for High Performance Computing (ROSS 2012)

Dries Kimpe, Kathryn Mohror, Adam Moody, Brian Van Essen, Maya Gokhale, Rob Ross and Bronis R. de Supinski



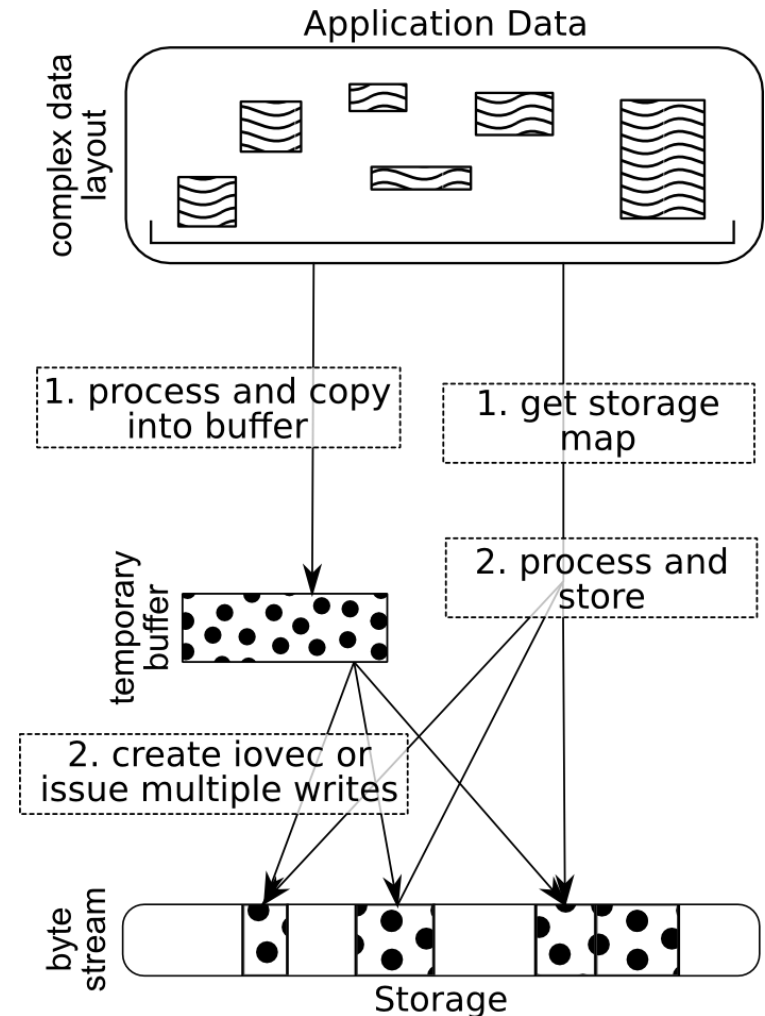
Container Abstraction

- Explicit location (also remote)
- More restrictive than POSIX
 - Drop costly (unused?) features
- Restricted model enables some new features
 - 'Direct Storage Access' (True zero-copy)
 - Space reservation (!= preallocation)
 - 3rd party transfers
- Status: Early evaluation
Implementation Available

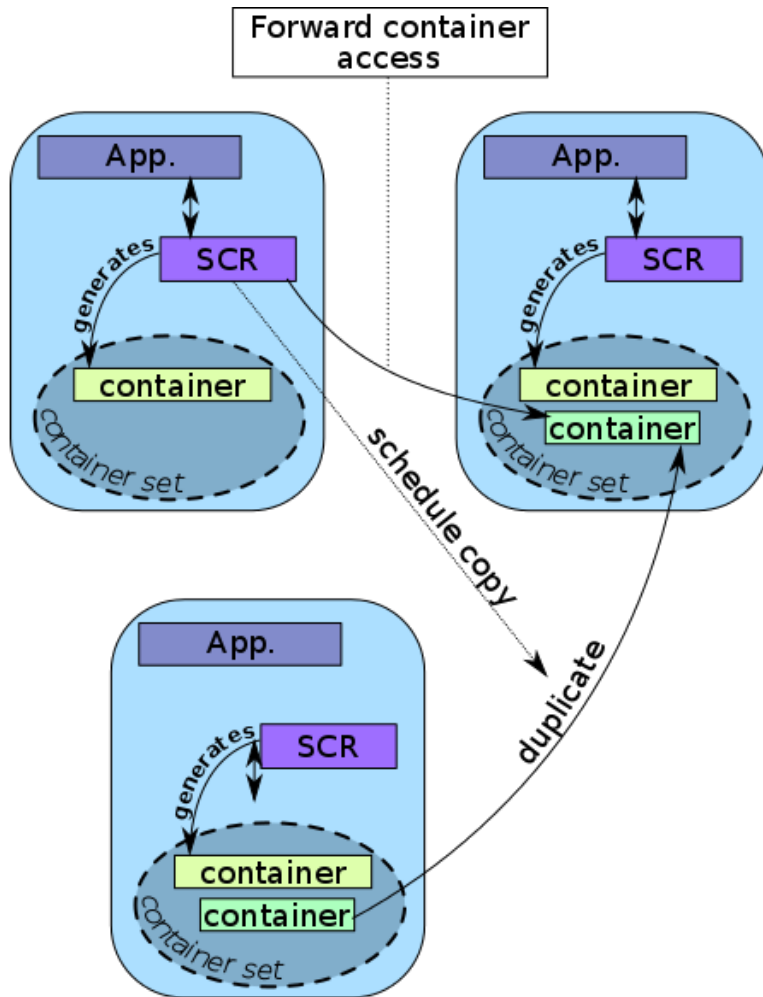


Direct Storage Access

- 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.



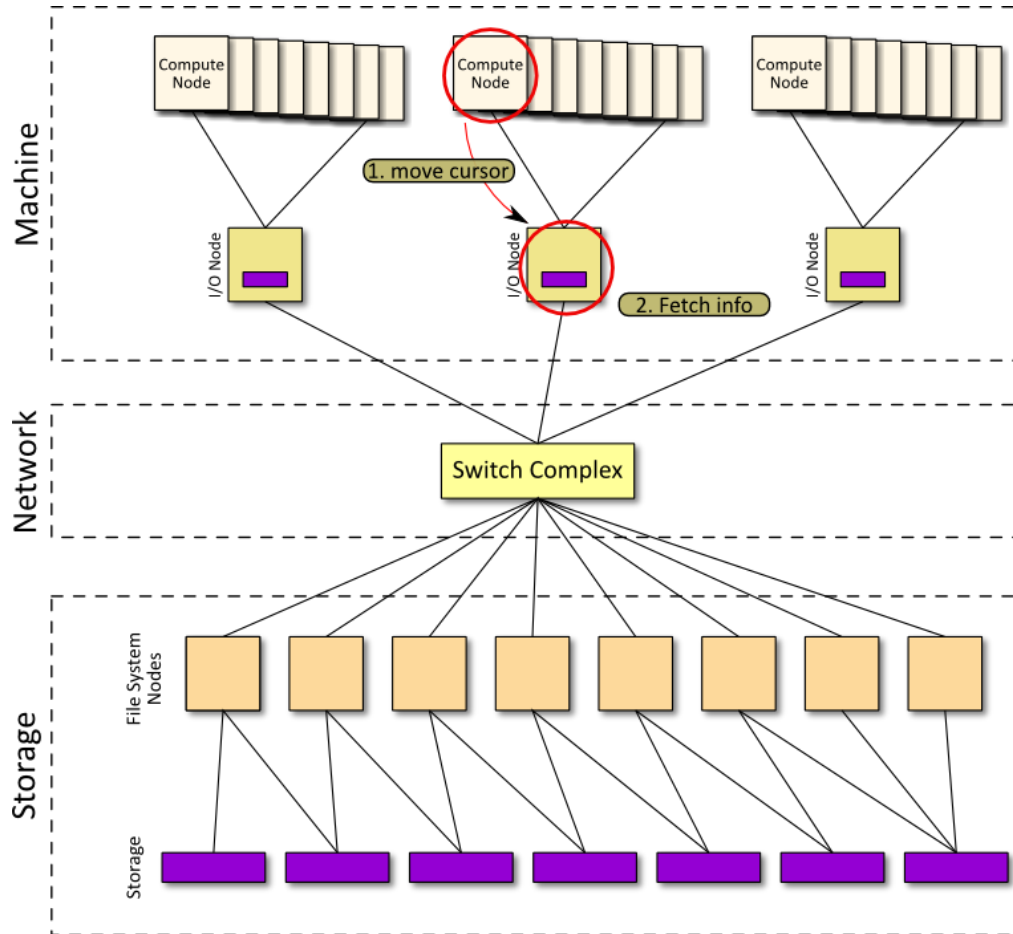
Remote Container Access & Transport



- 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



System Information Library



- Exposing node local storage forces the application/library to decide which (where) storage to use.
- For optimal performance, topology of system needs to be taken into account.
- Library allows enumeration of storage (and other?) properties for a node.
- Provides actions to move to the next or previous level in the hierarchy.
- General problem very difficult!

Exposing The True Nature of PVFS: Trident



High-Level Data Models over Object Storage

- Investigating object storage as a **more natural fit** for high level libraries
- Objects are independently accessed byte streams, **with attributes**
- Objects grouped into “containers”, roughly similar to traditional “file names”
- Experimenting with modified versions of PnetCDF and PVFS2
 - Early results show **complexity reduction** for PnetCDF
 - Explicit control over variable striping (downside: explicit control over variable striping)

PnetCDF

MPI-IO

“trident”
object API

parallel FS

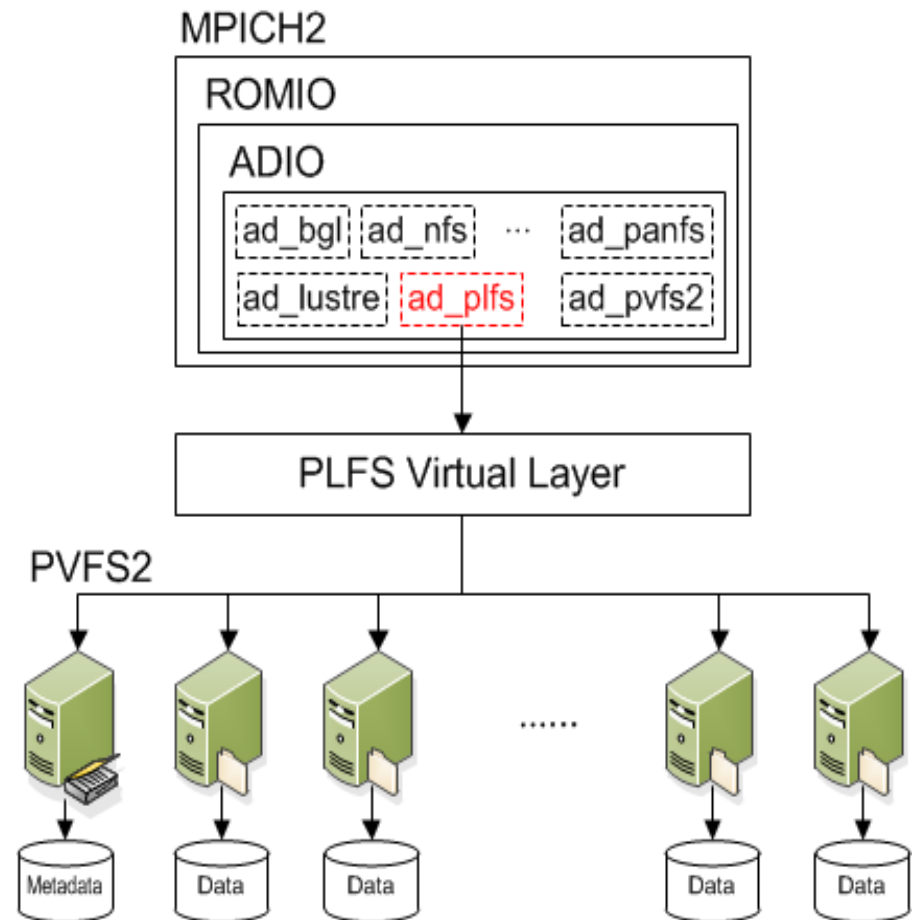
Modified
PVFS2

Dave Goodell (ANL)



PLFS on Trident

- Port of PLFS to Trident
 - PLFS is an I/O library which transforms I/O into a set of contiguous log writes
- Research directions:
 - Control placement
 - Reduce metadata overhead
- Status:
 - ad_plfs (ROMIO driver) complete
 - Starting work on PLFS port
- Shawn Kim (Penn State)
[summer internship @ ANL]

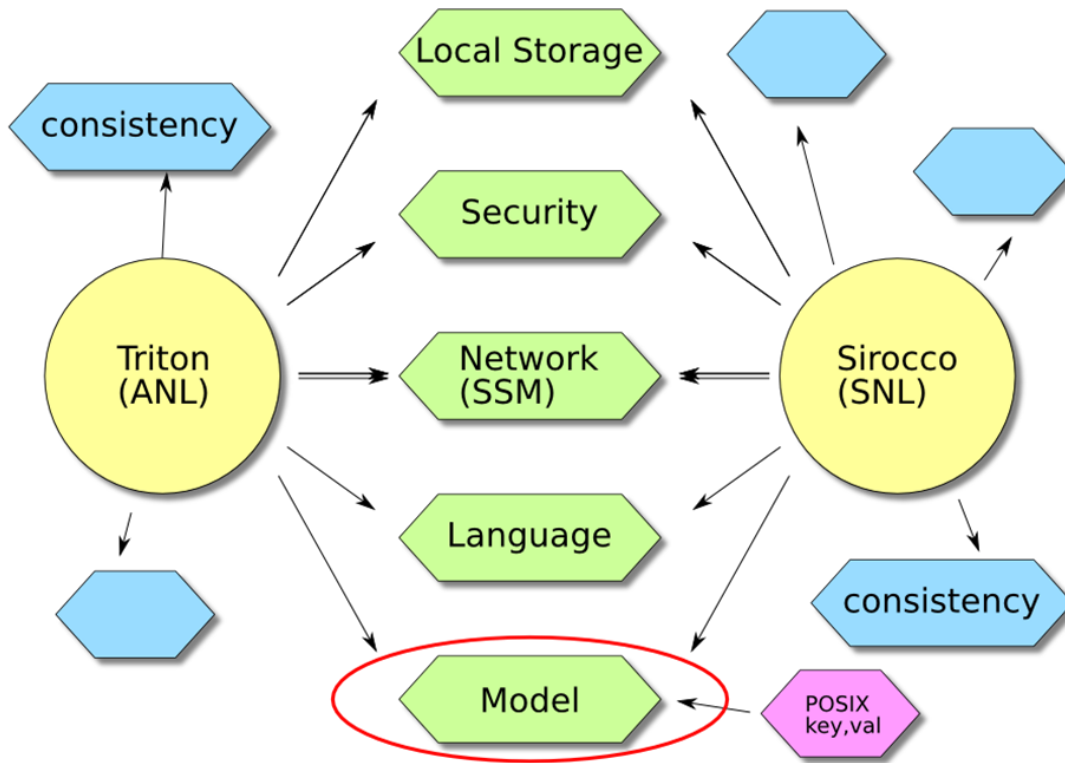


The Advanced Storage Group: Triton



Situation of this work

Advanced Storage Group (ASG)



- Concept: Friendly competition in designing an exascale storage system
- Different design choices, but shared building blocks simplifying exchange (codes, ideas)
- Periodical evaluation of design decisions with adoption of the best one.

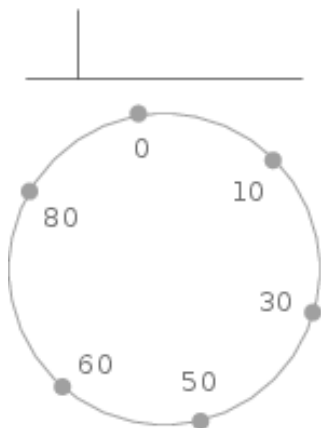
Triton

Introduction

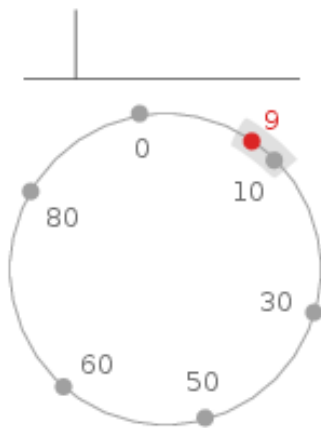
- The model presented in this talk is one of the models implemented by Triton.
 - (key,val), POSIX, variants
 - Self-healing, resilient
- Triton: ANL effort towards development of an exascale storage system
 - Comparison to T10 (object standard) OSD:
 - Triton is more like PanFS
 - Own local storage abstraction

Detail of Triton Resiliency Scheme

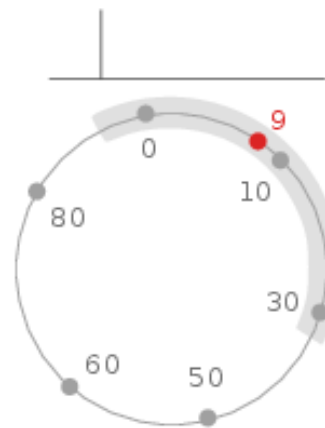
Servers (grey) are arranged in an n-dimensional address space and referenced by an ID in that space. Here, $n=1$.



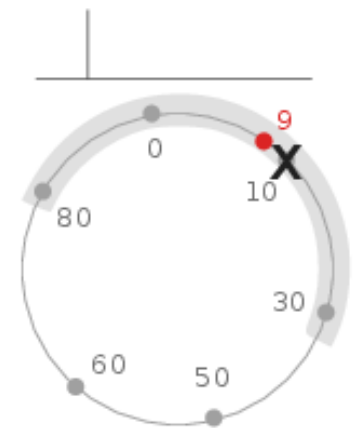
Objects (red) are likewise addressed by an ID in this space. The primary for an object is located at the server with the closest ID in the address space.



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.



ASG Data Model

Overview

Storage System

Collection of containers

Container

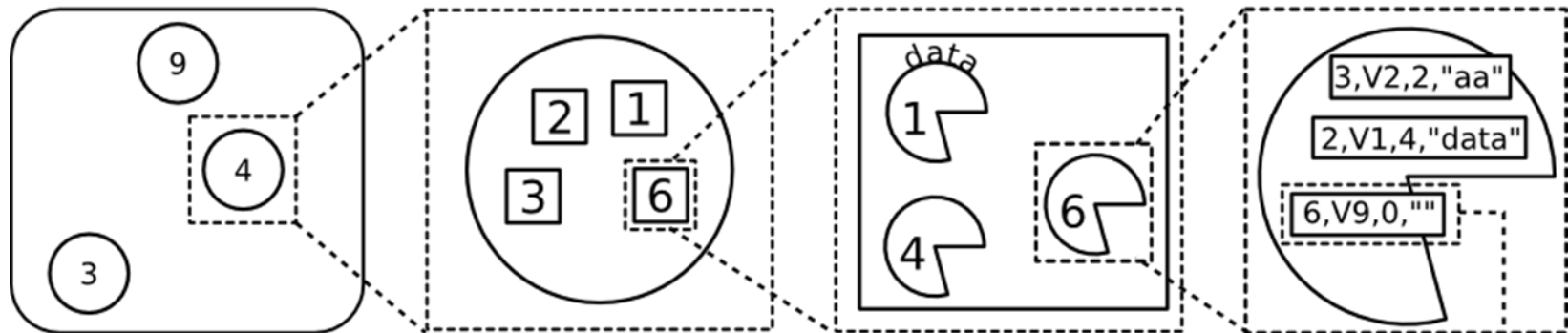
Collection of objects

Object

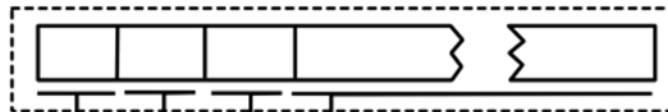
Collection of forks

Fork

Collection of Records



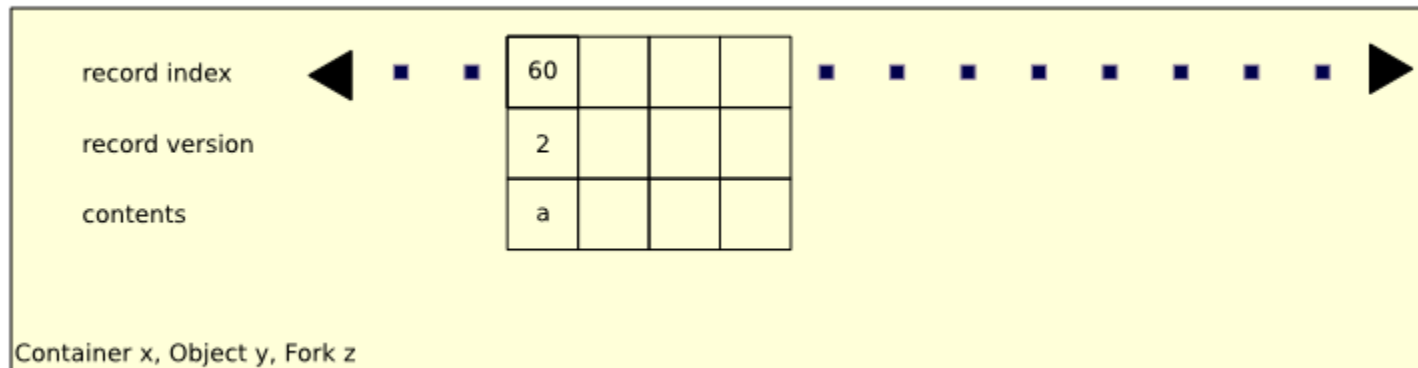
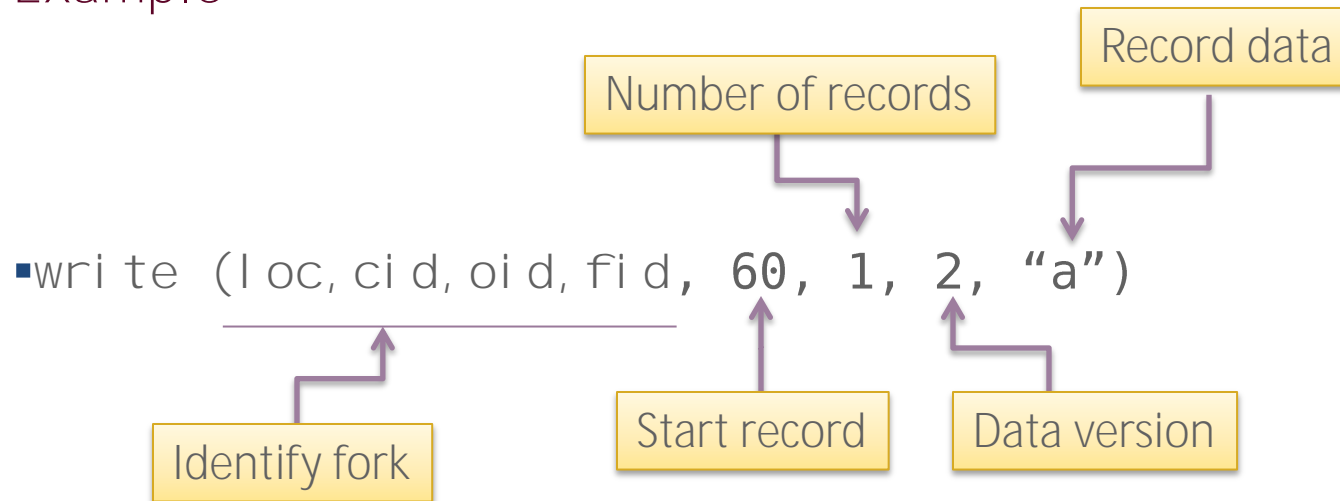
Record:



- record content (array of bytes)
- length of the record content (integer)
- version number of the record (integer)
- record key (integer)

ASG Data Model

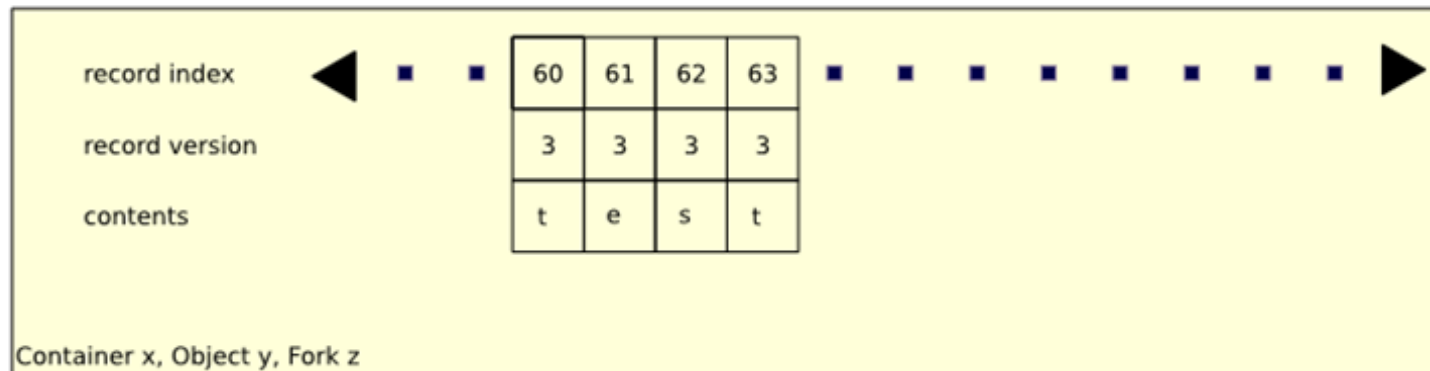
Example



ASG Data Model

Example

- write (l oc, ci d, oi d, fi d, 60, 1, 2, "a")
 - write (l oc, ci d, oi d, fi d, 60, 4, 3, "test")
- Writing 4 records with version number 3

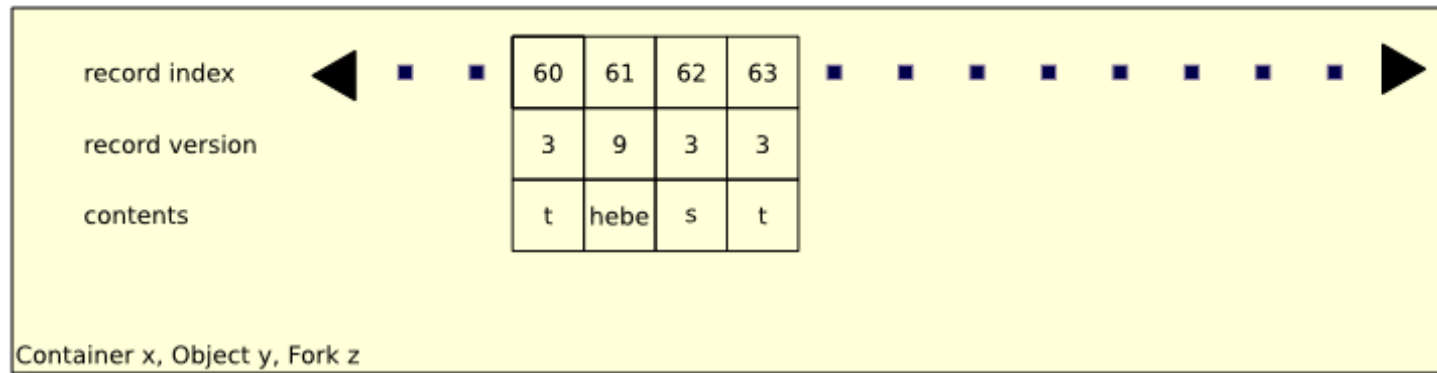


ASG Data Model

Example

- `write (loc,cid,oid,fid, 60, 1, 2, "a")`
- `write (loc,cid,oid,fid, 60, 4, 3, "test")`
- `write (loc,cid,oid,fid, 61, 1, 9, "hebe")`

–Writing 1 record of length 4 with version 9



Data Model: Operations

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.); No data
- Punch: Like write, but writes zero-length records

- Reset: **Sets the entity back to the default state (i.e. 'erase')**
 - Note: no 'create'

- Write, read and punch support conditional execution based on the expected version (more about this later).
- Client generally provides version number; API also supports auto increment.
- Write, read, punch operate on records
- Probe and reset operate on records, forks, objects and containers
- Version: Used to order transactions; No retrieval of obsolete versions



Conditional Operations

- Enables the user to provide a condition on the version of one or more of the specified records.
- If the condition is not satisfied, the operation does not retrieve or update record contents; However, information is returned.
- Currently:
 - COND_UNTIL: Transfer (read or write) records as long as the existing version (if any) is strictly smaller than the specified version.
 - COND_ALL: Only transfer data if all existing records in the range have a version number strictly smaller than the specified version.

Example:

`write (... , 60, 3, 2, "abc") = OK`




Conditional Operations

- Enables the user to provide a condition on the version of one or more of the specified records.
- If the condition is not satisfied, the operation does not retrieve or update record contents; However, information is returned.
- Currently:
 - COND_UNTIL: Transfer (read or write) records as long as the existing version (if any) is strictly smaller than the specified version.
 - COND_ALL: Only transfer data if all existing records in the range have a version number strictly smaller than the specified version.

Example:

`write (... , 62, 1, 4, "d") = OK`

60	61	62
2	2	2
a	b	c



60	61	62
2	2	4
a	b	d



Conditional Operations

- Enables the user to provide a condition on the version of one or more of the specified records.
- If the condition is not satisfied, the operation does not retrieve or update record contents; However, information is returned.
- Currently:
 - COND_UNTIL: Transfer (read or write) records as long as the existing version (if any) is strictly smaller than the specified version.
 - COND_ALL: Only transfer data if all existing records in the range have a version number strictly smaller than the specified version.

Example:

```
write (... , 60, 2, 3, "efgh", COND_ALL) = OK
```

60	61	62
2	2	4
a	b	d



60	61	62
3	3	4
ef	gh	d



Conditional Operations

- Enables the user to provide a condition on the version of one or more of the specified records.
- If the condition is not satisfied, the operation does not retrieve or update record contents; However, information is returned.
- Currently:
 - COND_UNTIL: Transfer (read or write) records as long as the existing version (if any) is strictly smaller than the specified version.
 - COND_ALL: Only transfer data if all existing records in the range have a version number strictly smaller than the specified version.

Example:

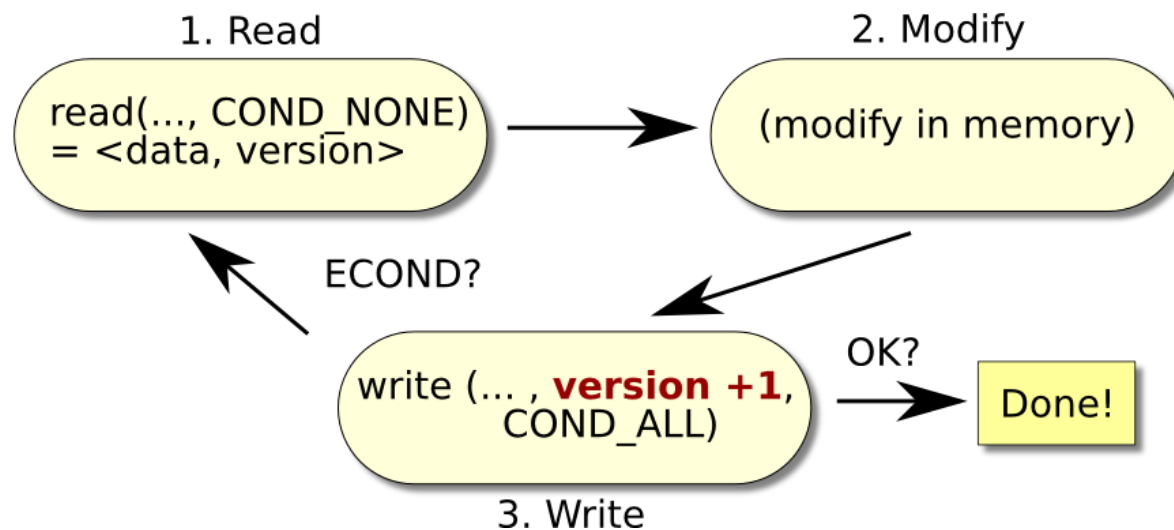
```
write (... , 60, 3, 4, "abc", COND_ALL) = ECOND
```



Example

Synchronization: R-M-W using versioning

- The model does not support locking
 - Read and write are atomic
 - However: what about Read-Modify-Write?
- Conditional operations can be used to implement R-M-W

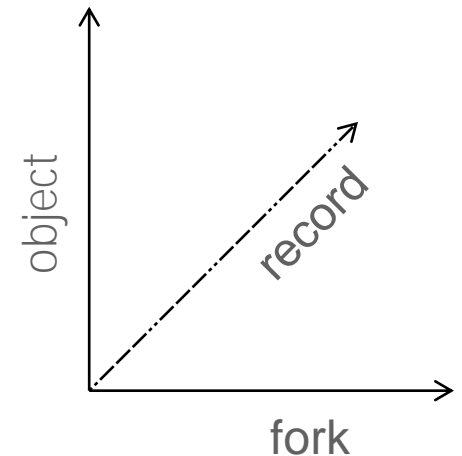
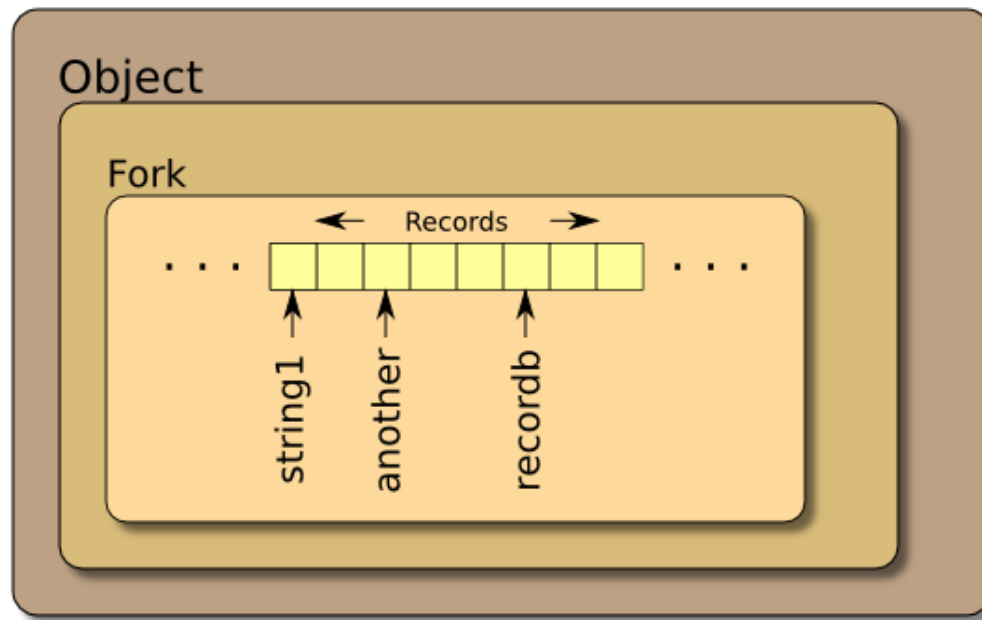


Example

Exploiting Object Structure

- Performance of preliminary implementation is not affected by choice of fork
- Fork + record can be used as 2-dimensional record space
 - Record contents additional dimension (access granularity)
- Example: (key,value) structures

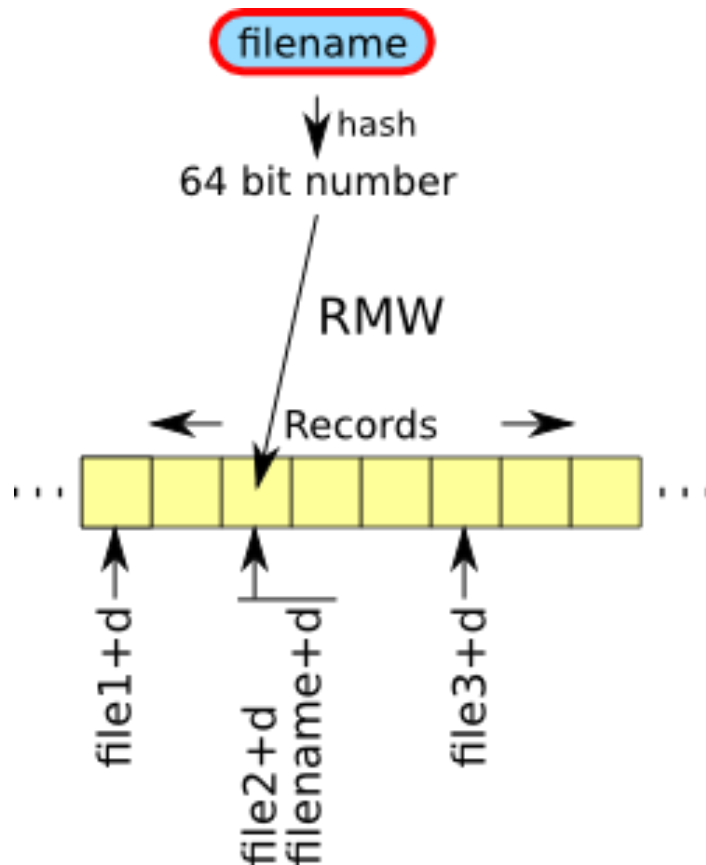
Container



Example

Implementing extended attributes and directories

- (key,value) mapping (with key a *string*) data structure which supports **atomic insert, overwrite, lookup and remove (rename?)**

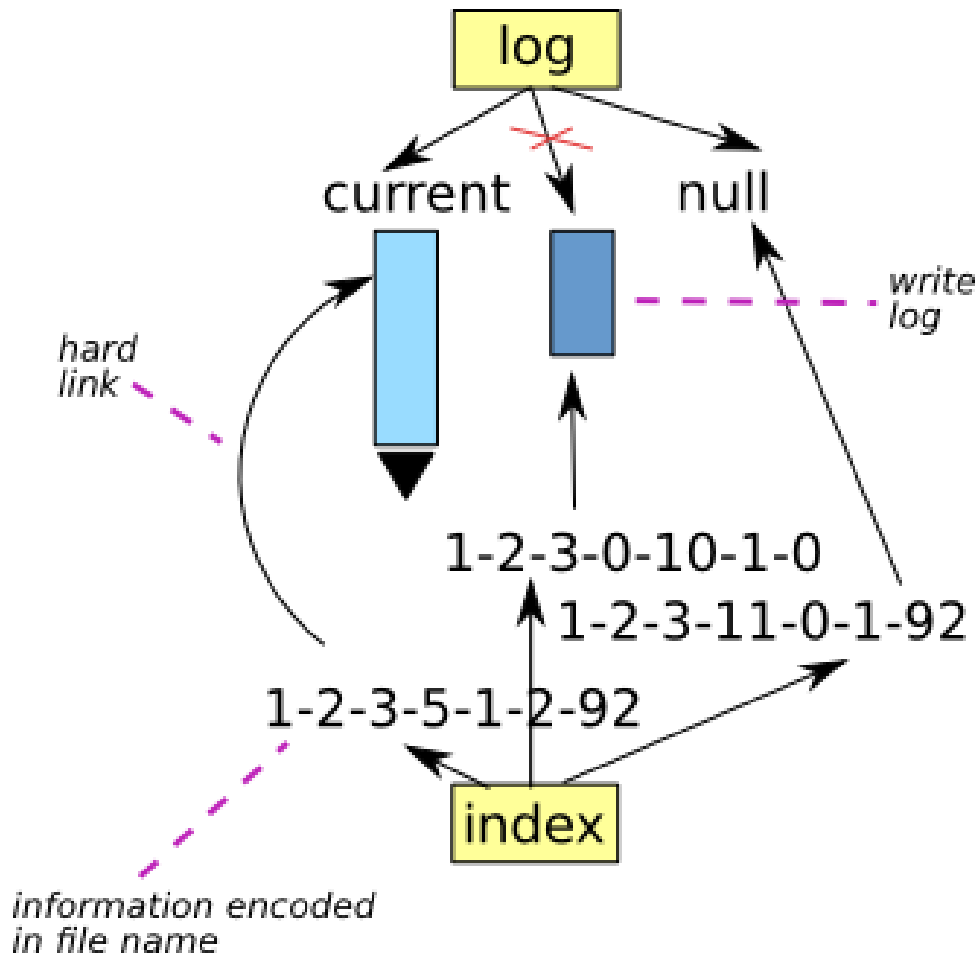


Preparation: Hash the string key, use as destination record number.

- Insert: write-conditional with default version
- Overwrite: R-M-W
- Remove: R-M-W with empty data
- Lookup: unconditional read (is atomic)

Note: each object can support 2^{64} of these data structures!

Reference Implementation



- Implements the model focusing on functionality and usability, not performance, resilience or scalability.
- No external dependencies
- Uses underlying FS
- Hardlink support required
- Write logging
- Uses directory as DB, filename to encode data
- Code available at [git://git.mcs.anl.gov/asg/reference](https://git.mcs.anl.gov/asg/reference)



Open Questions

(ongoing work)

- Namespaces
 - Reddy Narasimha & students (Texas A&M Uni): Legacy support (POSIX)
 - Cengiz Karakoyunlu (UConn) summer project @ ANL
- Location-Awareness
 - Do we need to expose location in the model?
 - If not: how do we offer placement control?
- Auditing & Security
 - Collaboration with Richard Brooks & Jill Gemmill (Clemson)
 - Building on LWFS work (validation, simulation)
- Provenance
 - Bradley Settlemeier (ORNL)



Acknowledgements

- Team at Argonne
 - Phil Carns, Dave Goodell, Kevin Harms, Dries Kimpe, Rob Ross, Justin Wozniak
- Collaborators (ASG)
 - ORNL: Stephen Poole, Bradley Settlemyer
 - SNL: Lee Ward, Matthew Curry, Ruth Klundt, Geoffrey Danielson
 - Clemson: Jill Gemmil, Richard Brooks, Haiying Shen
 - UAB: Anthony Skjellum, Matthew Farmer
 - ... and people I forgot to mention here!
- More information about Triton:
 - Triton: <http://trac.mcs.anl.gov/projects/triton>
 - Object storage semantics for replicated concurrent-writer file systems
Philip Carns, Robert Ross and Samuel Lang
- Questions? dkimpe@mcs.anl.gov

