Multi-Dimensional Hashed Indexed Metadata/Middleware (MDHIM) Project

James Nunez
Ultrascale Systems Research Center
High Performance Computing Systems Integration
May 10, 2012

LA-UR 12-10467 and LA-UR-11-11964

Los Alamos NATIONAL LABORATORY EST. 1943

Agenda

- Project Personnel
- Motivation - PLFS Problem
- MDHIM - Description
- MDHIM - How It Works
- Communication
- Data Stores
- Future Work

MDHIM Project Personnel

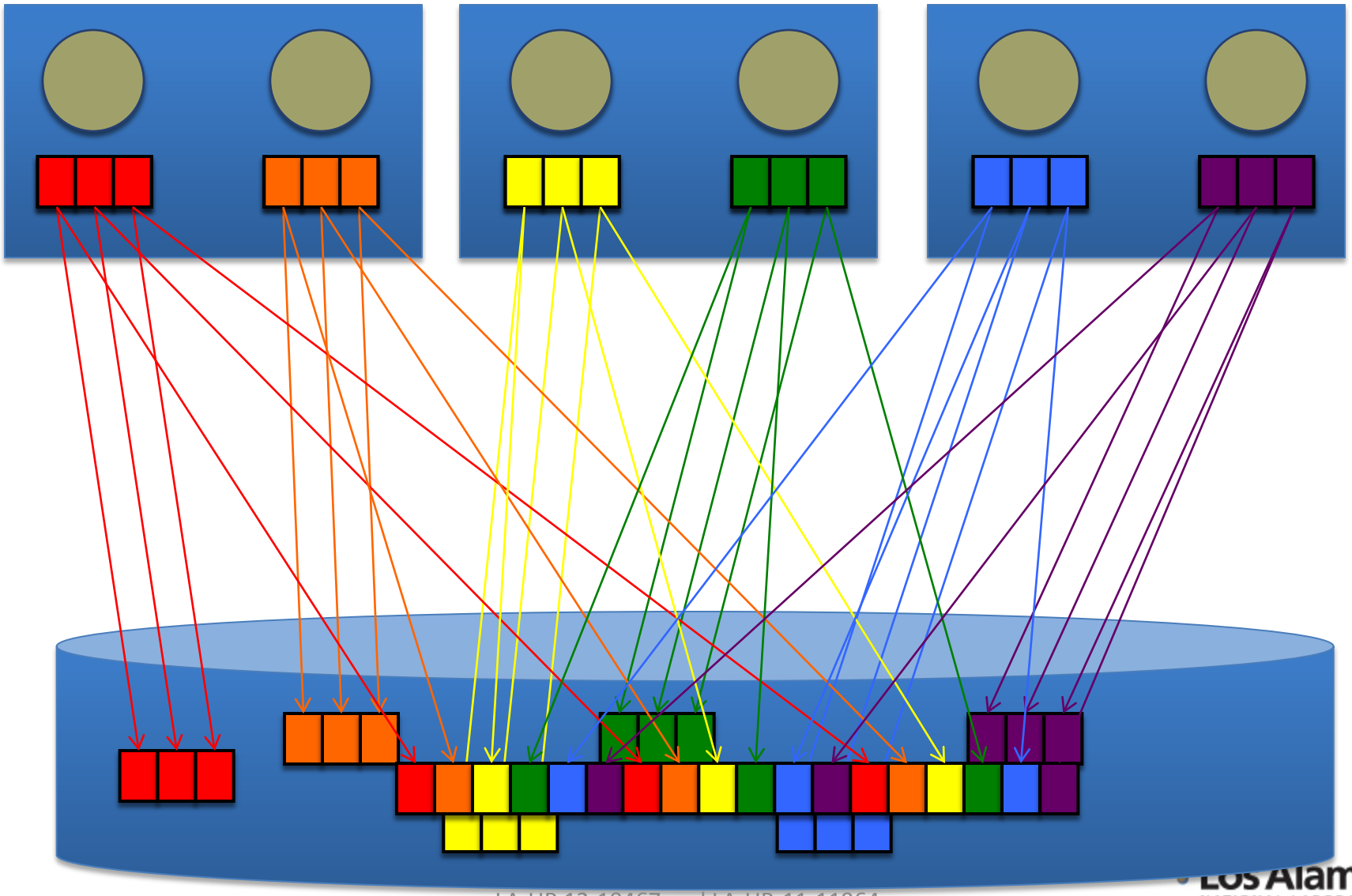
- Team Members
  - Gary Grider - Project PI
  - James Nunez - Developer
  - Hugh Greenberg - Developer
- Interested Parties
  - CMU: Garth Gibson, Chuck Cranor, and students(?)

MOTIVATION – PLFS: REVIEW AND PROBLEMS

DOE Parallel Apps "should" do Parallel IO

- Periodic checkpoint writes because the thousands of compute notes are not reliable
- Visualization writes
- Writes are synchronized
- Hundreds of thousands to millions of synchronized writes can be difficult for the file system
- Two most common write patterns
  - N-1 where N procs write to 1 shared file
  - N-N where N procs write to N non-shared files

N-1 File IO

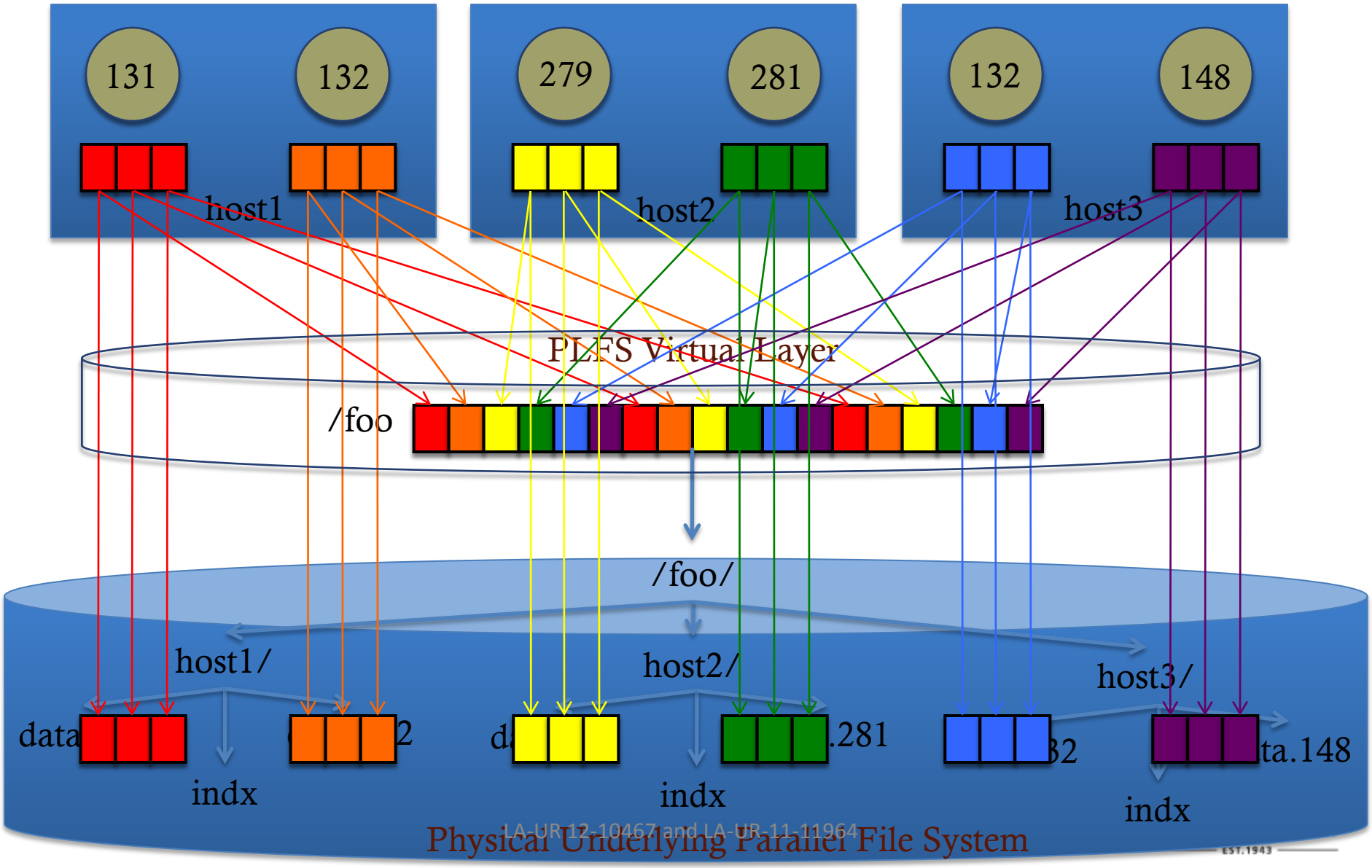


N to 1 is Really Better for the Long Run If Issues Can Be Addressed

- N to N at a billion way will just be a non starter.
- Further, even N to M seems silly for the user to have to manage, we have to solve the same problems for N to M as we do for N to 1
- N to 1 really looks like the best long run answer but we have to fix
  - Concurrence Management
  - Metadata Management
  - Etc.
- Lets take the best of both N to N and N to 1

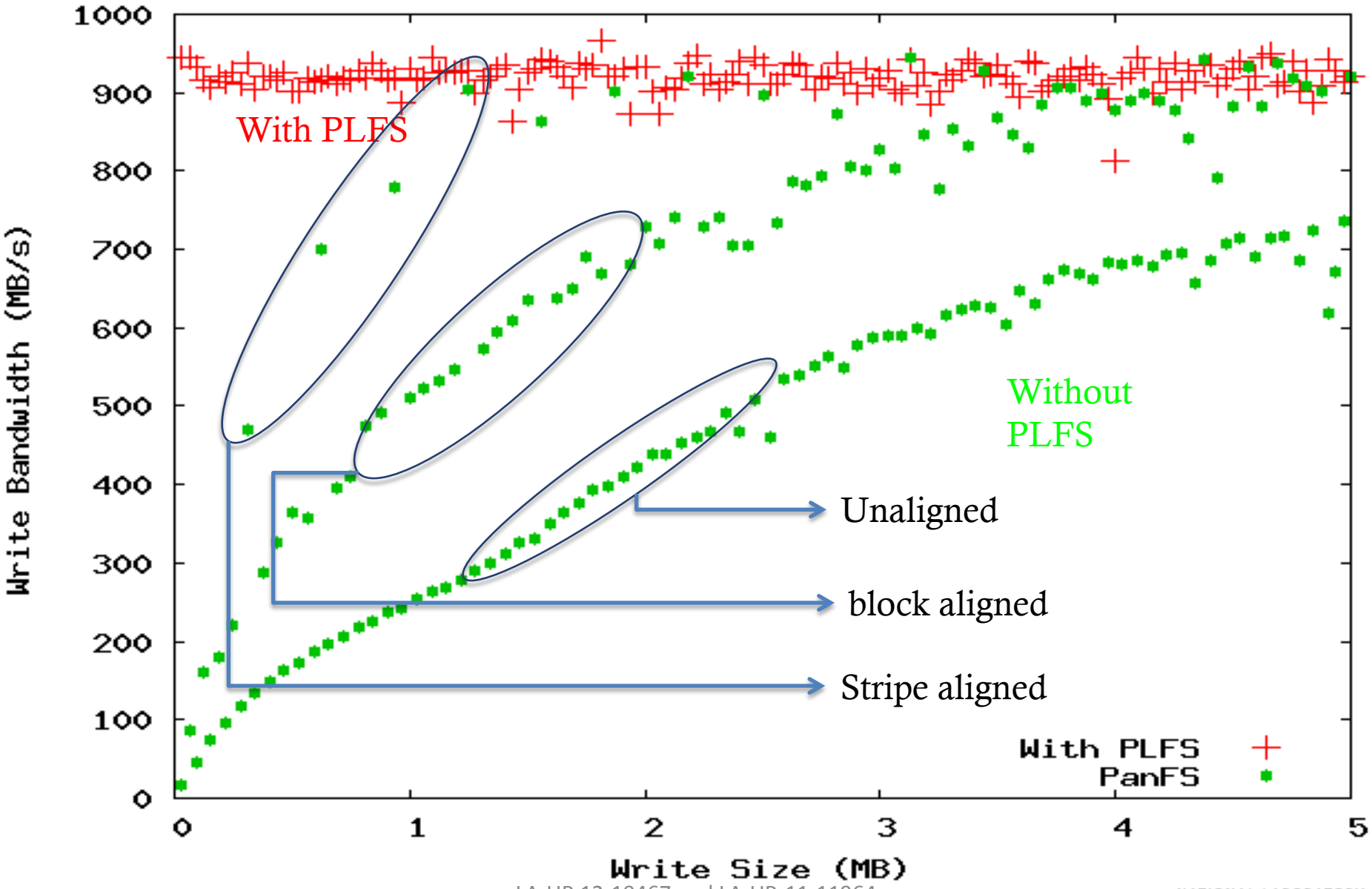
→ PLFS

Decouples Logical from Physical



LA-UR-12-10467 and LA-UR-11-11964

LBNL PatternIO Benchmark

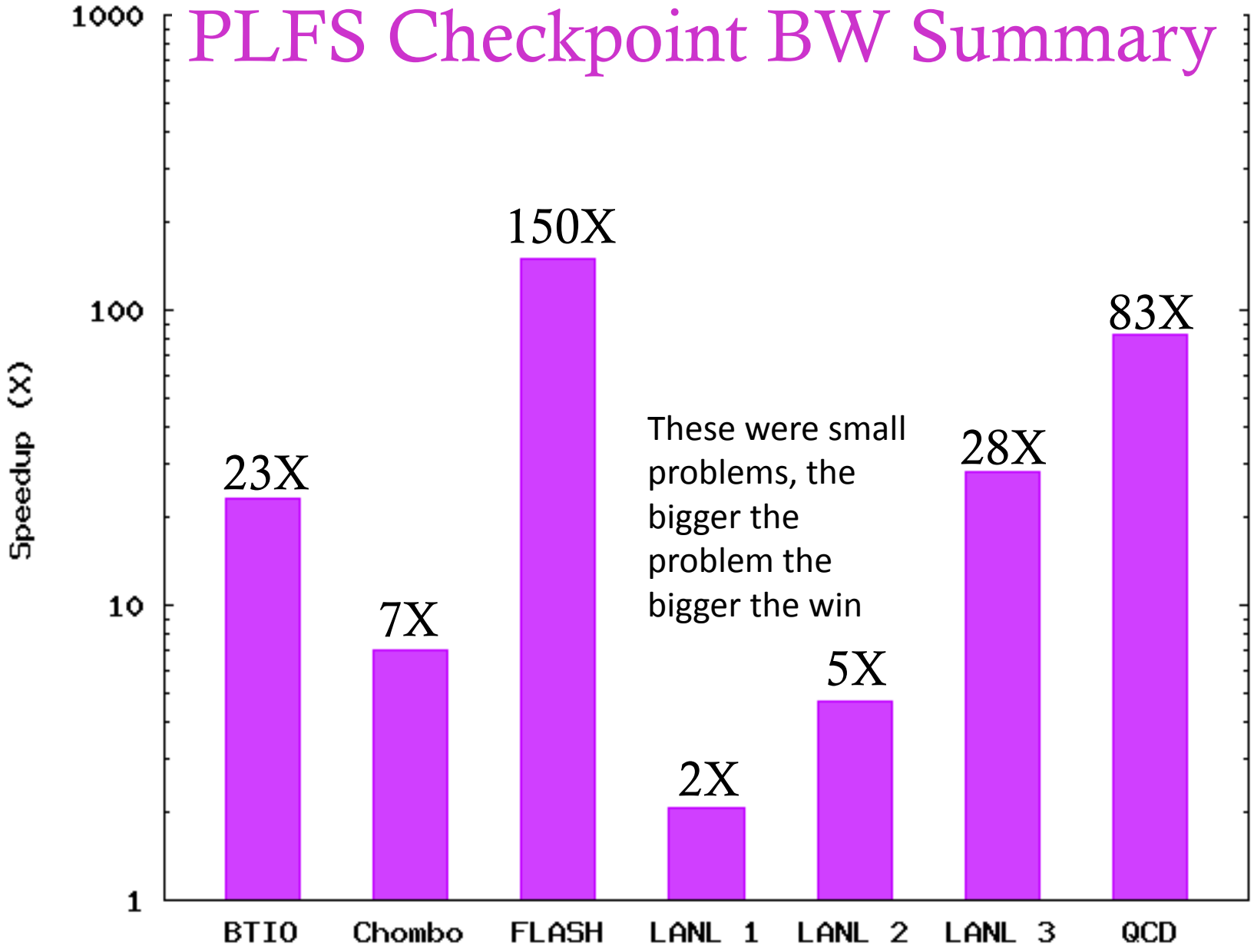


LA-UR 12-10467 and LA-UR-11-11964

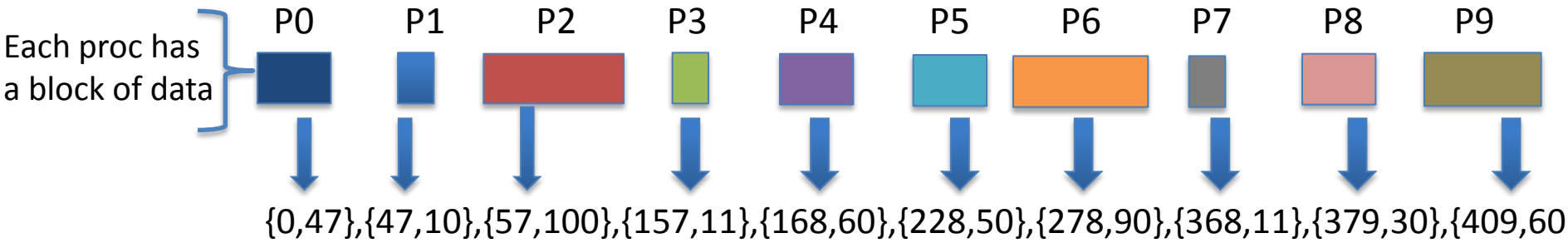
PLFS makes alignment and blocksize irrelevant!

NATIONAL LABORATORY EST.1943

PLFS Checkpoint BW Summary



Recall: PLFS Independent Writing

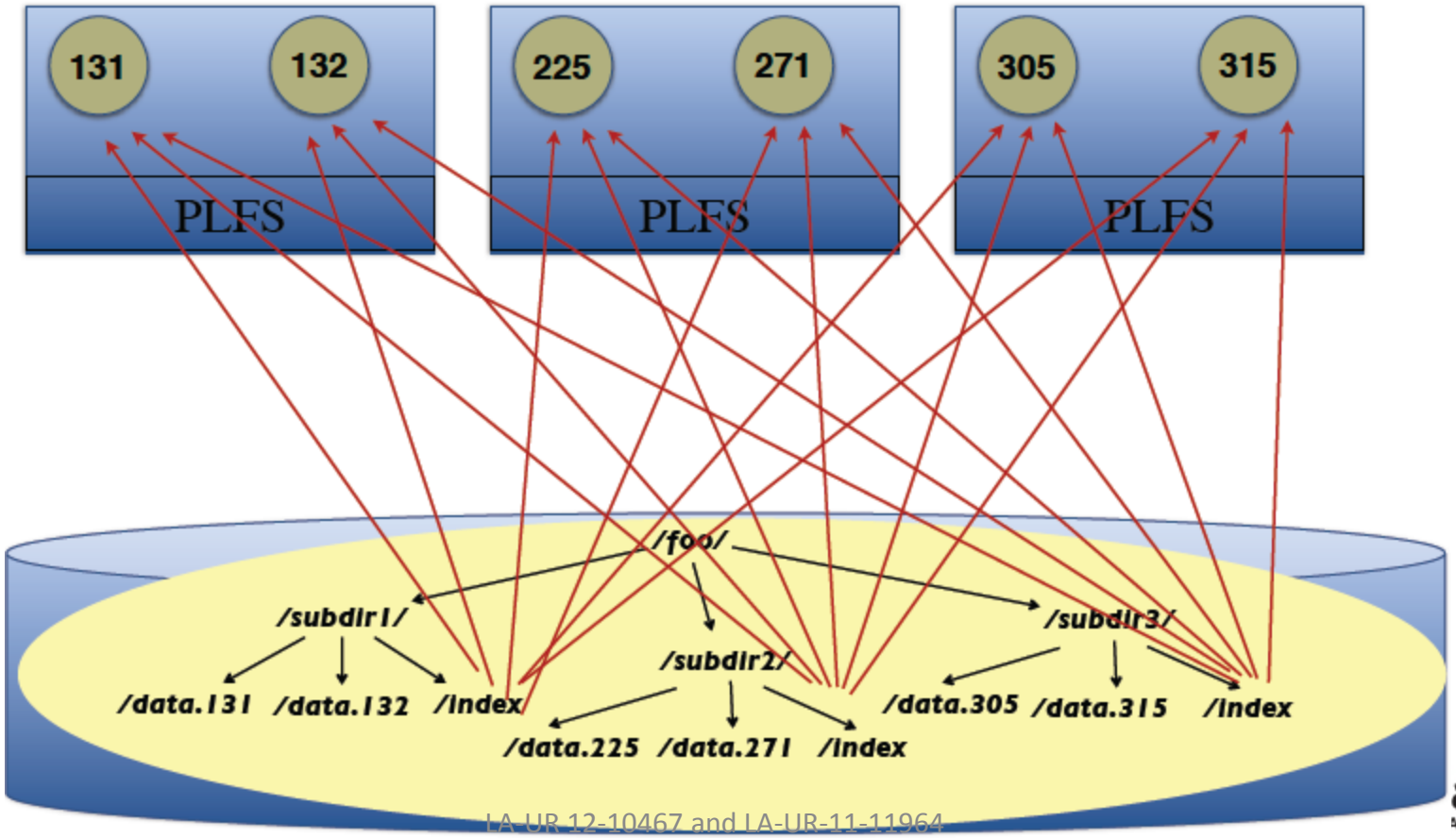


PLFS writes each block to a unique physical file (one log structured file per proc) and remembers where data is with an index.

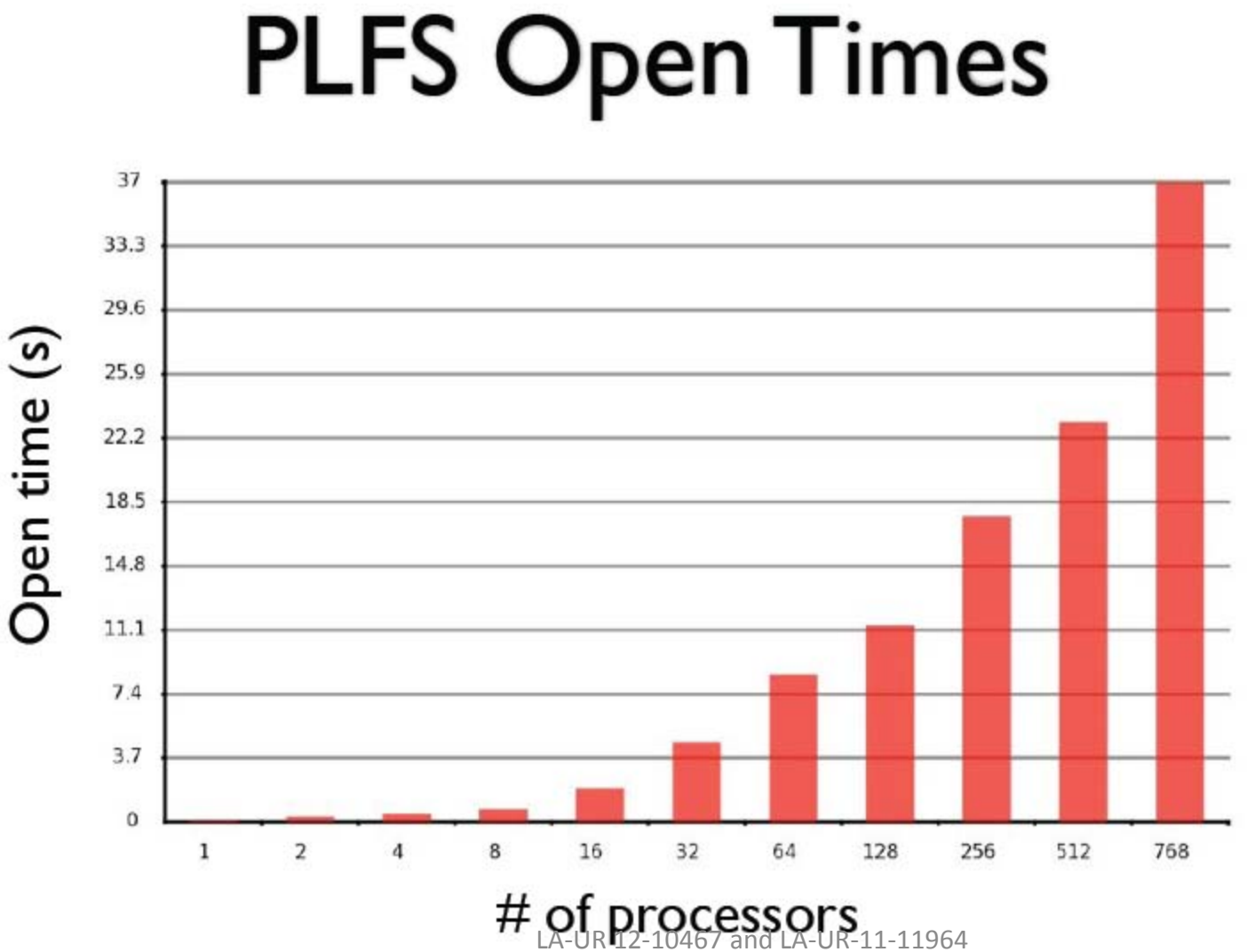
Notice that if every write were the same size (i.e. it was structured), then PLFS could save just one index entry identifying the pattern instead of one entry per write.

Logical Offset	Physical Offset	Length	Physical Block ID
0	0	47	0
47	0	10	1
57	0	100	2
157	0	11	3
168	0	60	4
228	0	50	5
278	0	90	6
368	0	11	7
379	0	30	8
409	0	60	9

Naïve Approach: Every Process Reads All Indexes to have a Complete Index in Each Proc's Memory to Honor any Unknown Read Request (N Squared!)



Effects of Everyone Reading All PLFS Indexes



MDHIM – WHAT IS IT?

LA-UR 12-10467 and LA-UR-11-11964

The MDHIM Project

- The Multi-Dimensional Hierarchical Indexing Metadata/Middleware (MDHIM) project aims to develop a research prototype infrastructure capable of managing massive amounts of index information representing even larger amounts of scientific data to enable data exploration at enormous scale
- Another way to think about the utility of what we are proposing is to think of a highly scalable/parallel multi-dimensional Virtual Storage Access Method (VSAM) or key/value store (Bigtable/Hbase).

Key/Value Stores Already Exist, Why Another?

- What about the new indexing technology like Bigtable, Hbase …
  - Written in languages not suited for extreme scientific computing environments (Java, Erlang, etc.)
  - Built on top of immature infrastructures like HDFS etc.
  - Even Hypertable which runs on standard file systems does not leverage the real power of parallel file system technology
  - Don't really take advantage of supercomputer interconnects
- We can make data operations scale on commercial parallel file systems today by utilizing shared nothing/append only concepts just like Google has done with lessoned semantics
  - Our premise is that GFS/HDFS recreated many of the wheels that parallel file systems have had for years and just went unnoticed.
- Shouldn't there be a way to make needed metadata/indexing operations scale on top of existing commercial file systems?
- Can we leverage supercomputer style interconnects to go beyond shared nothing style indexing techniques

LA-UR 12-10467 and LA-UR-11-11964

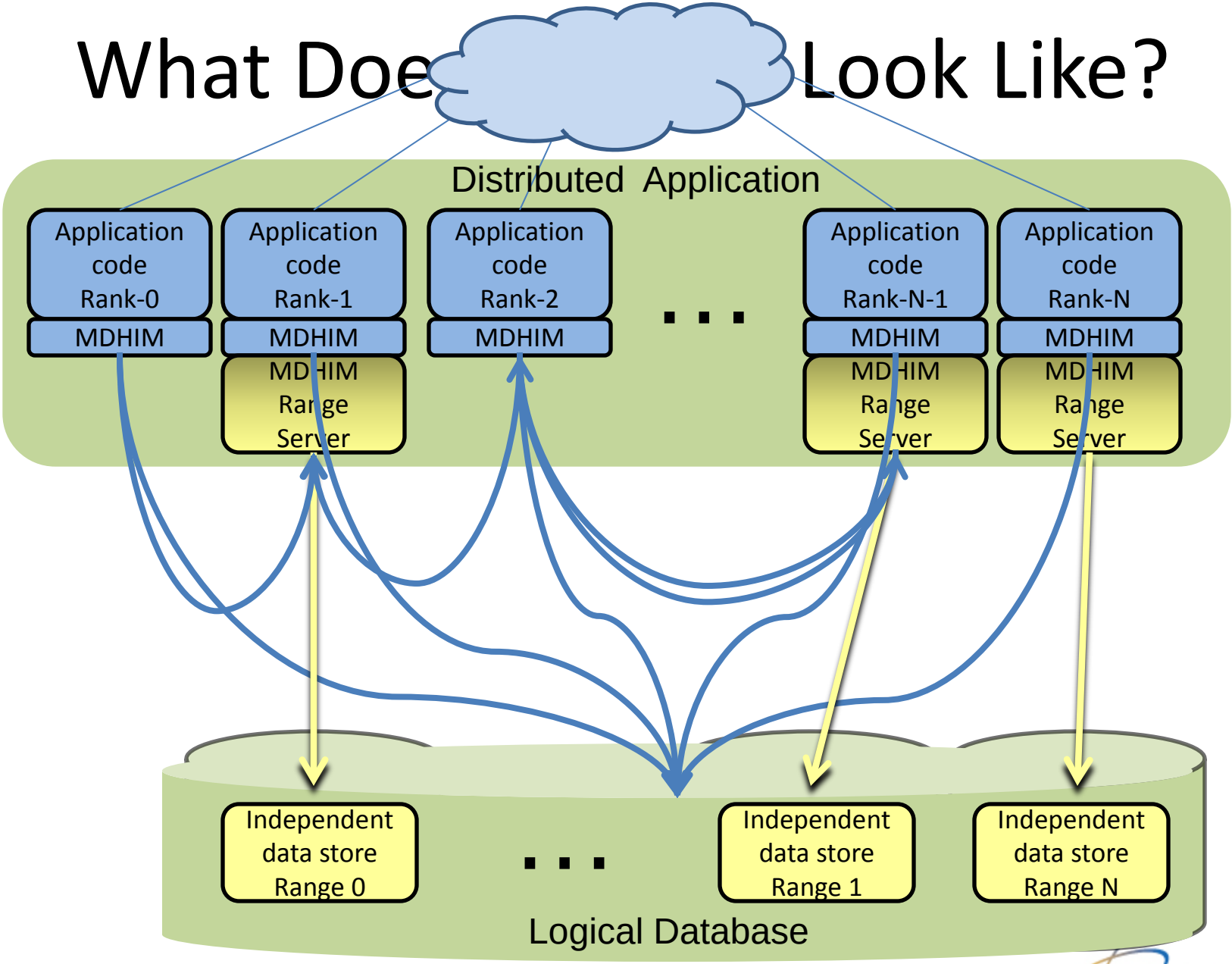
How? Leverage

- Giga+ and its algorithms for achieving very high concurrency to very large numbers of entries within a single file system directory.
- Commercial Parallel File System files, directories, tree indexing
- Bigtable/Hypertable have reasonable key/value api's
- Public ISAM and other index methods exists but most are not parallel (PBL ISAM, Tokutek, Fastbits, etc.)
- These leveraged capabilities are the required ingredients for building a scalable Parallel ISAM:
  - append only methods for data and metadata
  - shared nothing to everything
  - tree indexing, ranging, etc.
  - light weight embeddable single machine ISAM
- Supercomputing class interconnects, enabling shared some

MDHIM – HOW IT WORKS

LA-UR 12-10467 and LA-UR-11-11964

What Does It Look Like?



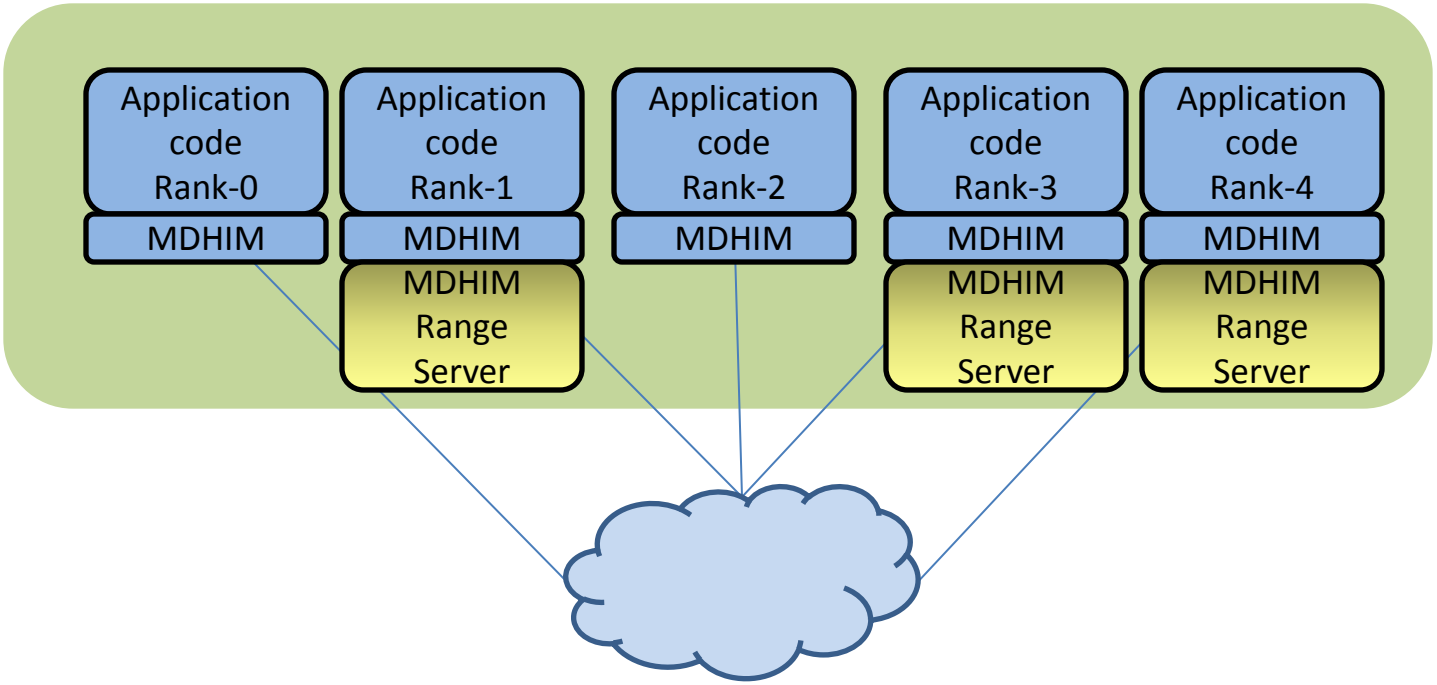
MDHIM API

- MDHIM init
  - Initializes MDHIM structures and creates range server threads.
- MDHIM open
  - Creates directories for data and key files.
- MDHIM insert
  - A list function that inserts new records with key and record data.
- MDHIM flush
  - Makes key distribution information available to the clients.
- MDHIM find
  - find a record using primary key (match, best higher or lower) and set the absolute record number.
- MDHIM finalize
  - Shut down range server threads.

MDHIM API

- MDHIM close
  - close a MDHIM file
- MDHIM get
  - get previous/next/first/last record
- MDHIM read
  - a list function that read records (key and data), using absolute record numbers
- MDHIM readkey
  - a list function that read the keys, using absolute record numbers
- MDHIM delete
  - Delete the current (last read) record
- MDHIM update
  - a list function that update data (not key) in records, using absolute record numbers

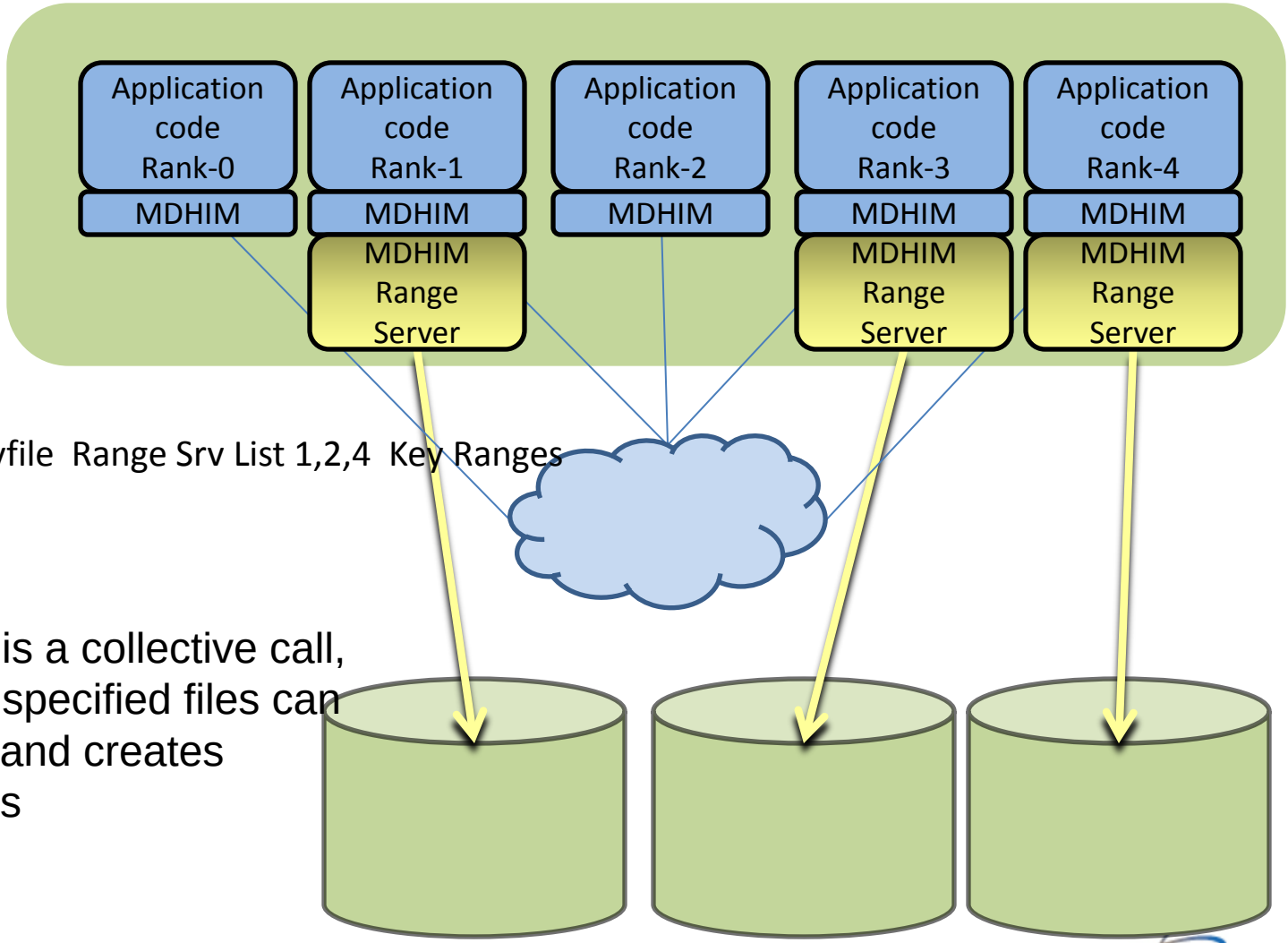
MDHIM Operation: Initialization



Initialization

Initialization is a collective call, it fires up index range services on specified nodes, set up range computation and ensure everyone knows how to compute range

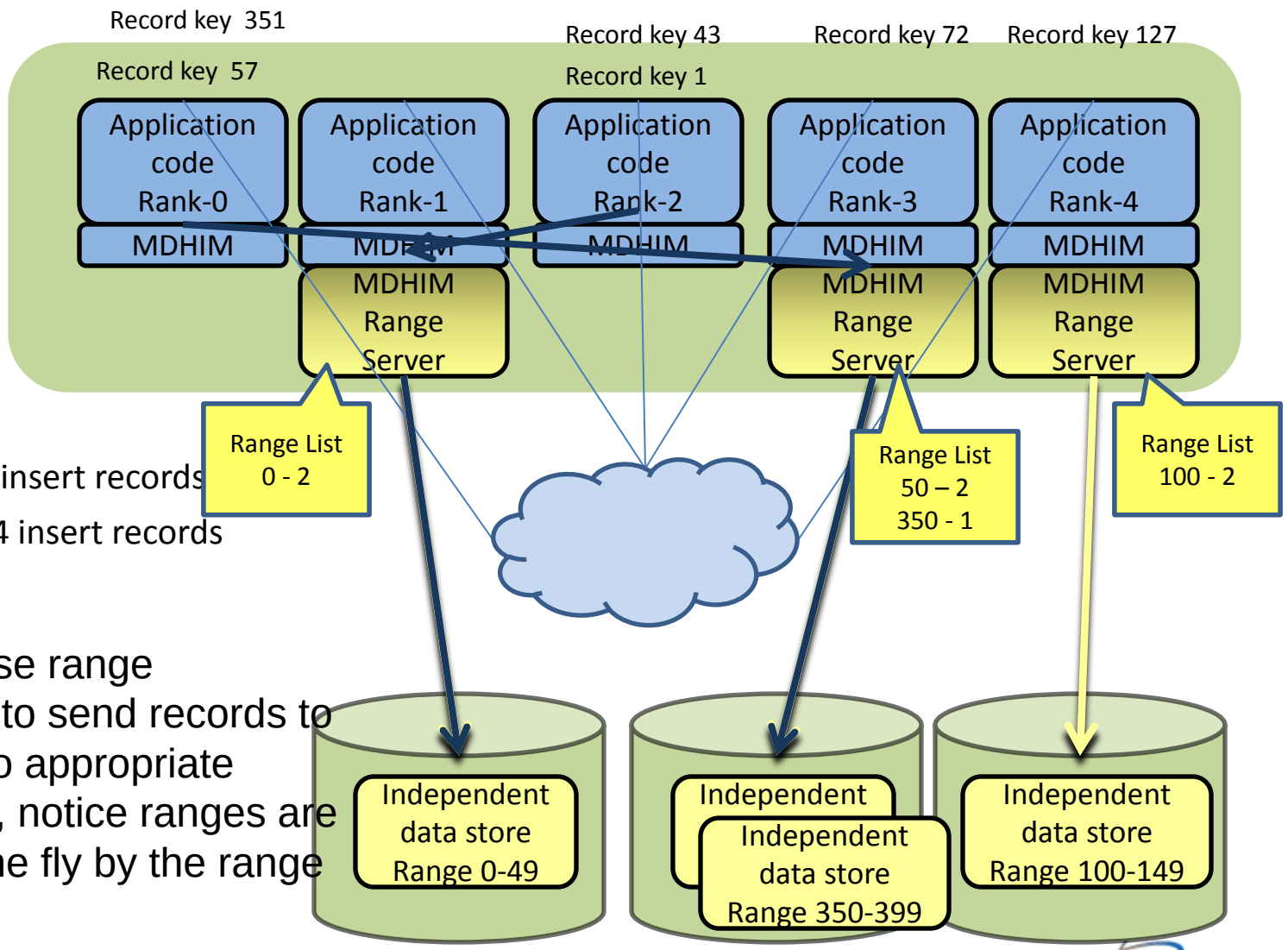
MDHIM Operation: Open



Open /tmp/keyfile Range Srv List 1,2,4 Key Ranges Every 50

Open/create is a collective call, that ensures specified files can be written to and creates subdirectories

MDHIM Operation: Insert



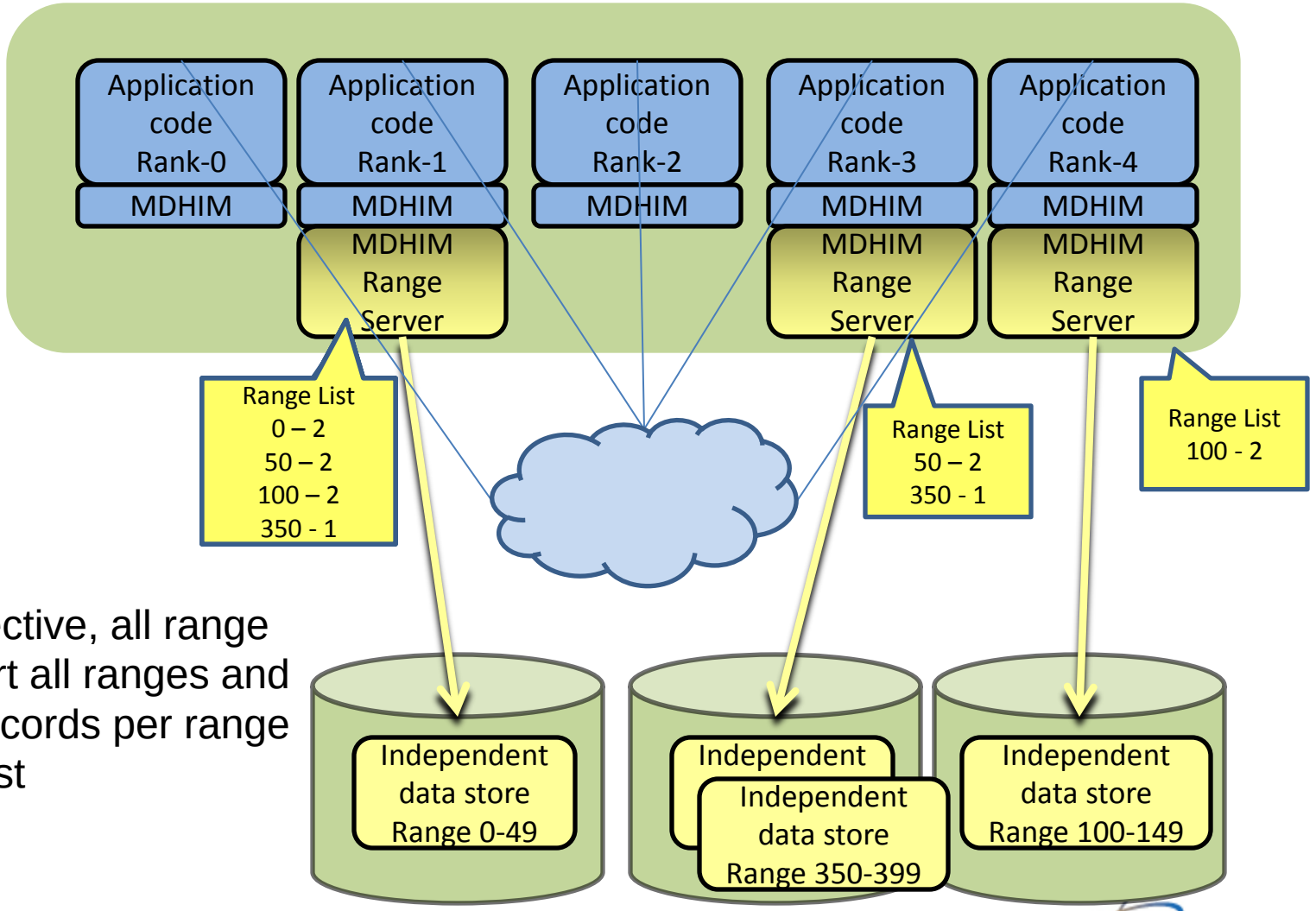
Processes use range computation to send records to be indexed to appropriate range server, notice ranges are created on the fly by the range servers

Indexes Types

- Primary Global – Global index with key order same as record order and Primary Global keys point at data records
  - There must be exactly one of these per PISAM file
  - If there is a Secondary Global index, this the Primary Global Index must be unique
  - If there is no Secondary Global index, the Primary Global Index doesn't have to be unique
  - If you don't want the Primary Global index functionality, then setting the Primary Global keys to random and possible random but unique is a way to accomplish this fuctionality (which implies you only want to use Secondary Local indexes)
- Secondary Local
  - There can be multiples of these and they do not have to be unique
  - There will be one instance of each Secondary Local index for each Global index (Primary and Secondary's)
  - There can be more than one type of Secondary Local index, to enable multiple statistical representations of the Secondary Local key values
- Secondary Global – Global index where key order is not same as record order. A Secondary Global key does not point at a data record, it points at the Primary Global key for the record which does point at a data record
  - There can be multiples of these and they do not have to be unique
  - If there is a Secondary Global index, the Primary Global keys must be unique
  - The Global Primary index is used to gain access to the data records

MDHIM Operation: Flush

Flush



Flush is collective, all range servers report all ranges and how many records per range and broadcast

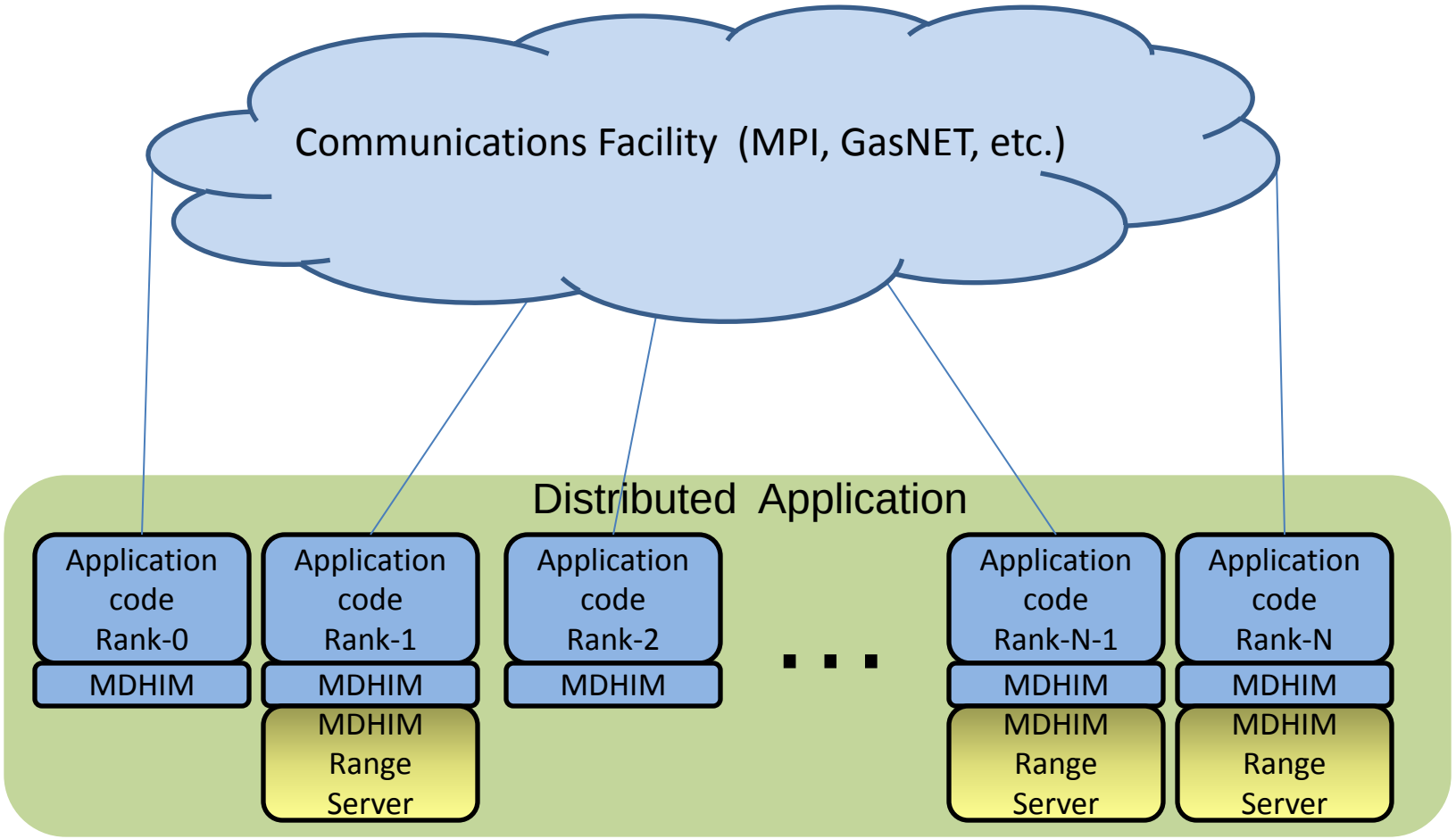
Key Distribution Methods

- For all indexes, one of the functions of MDHIM on flush is to share key distribution. For every range, for every index (Primary Global, Secondary Global, and Secondary Local) key distribution information can be kept as records are inserted and then this key distribution information can be provided to the query client for use in query optimization.
- The amount of key distribution information that can be kept by the range servers for each key can be specified. The minimum information would be number of records per range for the Primary Global Key which would result in knowing minimal key distribution information and global order for the records across ranges.
- Key distribution information can be ignored and this would imply basically a map/reduce or Hive like functionality. This minimizes the inter machine communications required and behaves as if we were not really on a supercomputer with a high performance interconnect
- Key distribution can be extremely rich and could provide statistics on each index for each range, for example quartiles or distribution type, mean, standard deviation, etc. The exact number of these statistical key distribution representations is not known at this time and will need to be flexible to allow the user to specify key distribution type information for each index for each range.
- This key distribution capability is one of the things that shows how adding the high performance interconnect on a parallel machine can benefit, more extensive statistical models per range or subrange per index versus costs to store the records.
- This is part of the power of the MDHIM approach, to provide the user the flexibility to have as many dimensions of indices as desired with as much or little key distribution information as desired. We are trying to provide a tool to allow people to represent enormous volumes of record oriented data with very small amounts of indices/statistical distribution information.

COMMUNICATION

LA-UR 12-10467 and LA-UR-11-11964

MDHIM Communication



MDHIM Communication

- Objective
  - Make use of the Supercomputer's high speed interconnect for communication with range services and key statistics distribution
- Currently using MPI
- Future
  - Abstract layer for communication
  - MPI, shmem, UPC (GASnet), …

DATA STORES

LA-UR 12-10467 and LA-UR-11-11964

Data Store

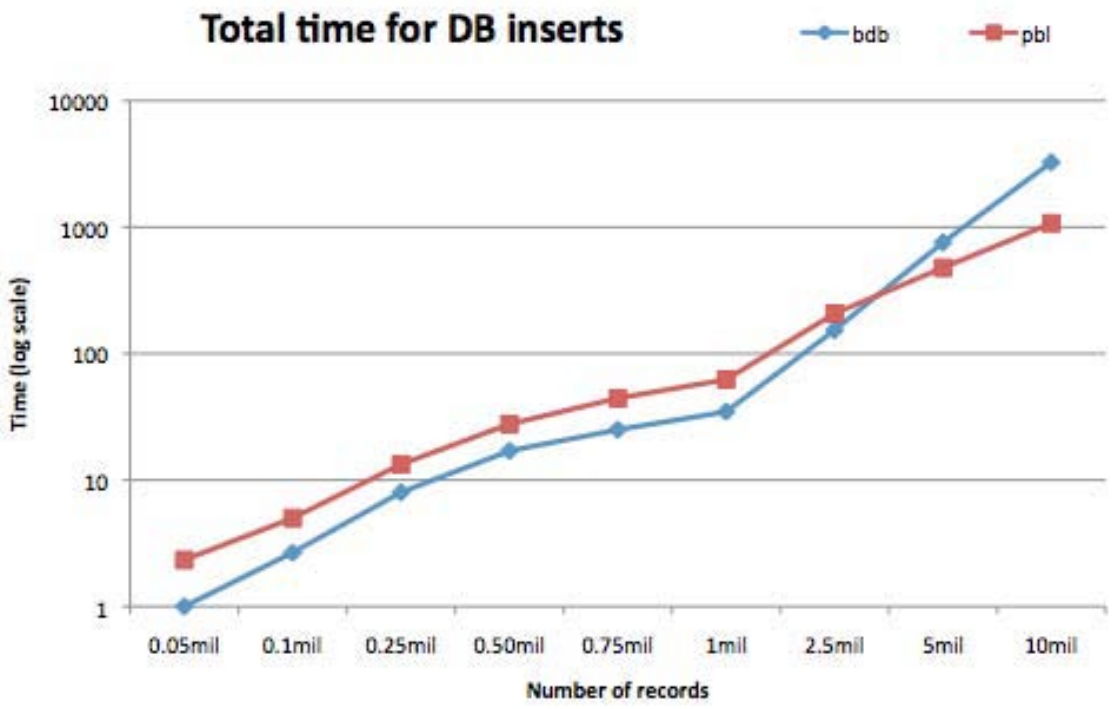
- Requirements
  - Embeddable
  - Language
  - No client-server model
  - Support range queries
- Currently using Program Base Library Functions Indexed Sequential Access Method (PBL ISAM)
- Future
  - Abstract layer for data stores
  - Berkeley DB, Level DB, Toukutek, B-Trees, …

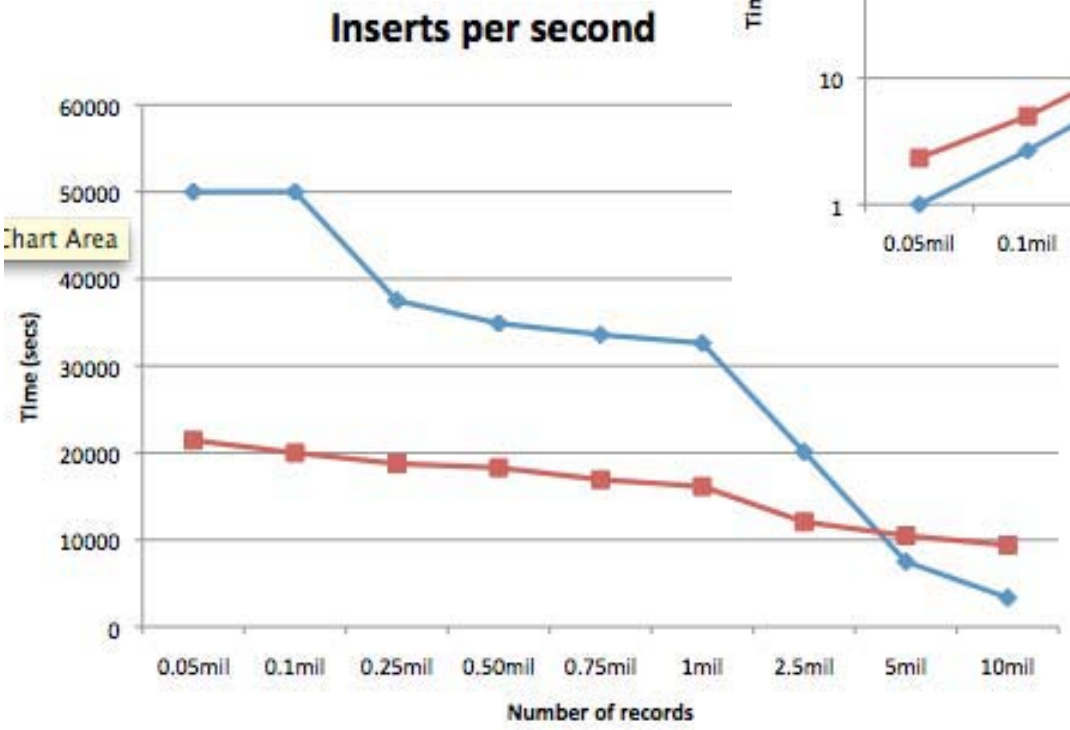
Candidate Data Stores

Storage Engine	Range Server Parallel	Lang.	Embed--dable	Server Process	SQL DB	Stab Queries	Range Queries	State--less
PBLISAM	No	C	✓	No	No	✓	✓	✓
Cassandra	✓	Java	No	✓	No	✓	✓	No
TokuDB	✓	C++	No	✓	✓	✓	✓	No
Fastbits	No	C++	✓	No	No	No	✓	✓
LevelDB	No	C++	✓	No	No	No	✓	✓
Hbase	✓	Java	No	✓	No	✓	✓	No
Hypertable	✓	C++	No	✓	No	✓	✓	No
SQLlite	No	C	✓	No	✓	✓	✓	No
BerkeleyDB	✓	C	✓	No	No	✓	✓	No
Voldermort	✓	Java	No	✓	No	No	No	✓
MonetDB	No	Java	No	✓	✓	✓	✓	No
MongoDB	✓	C++	No	✓	No	✓	✓	✓
Redis	No	C	No	✓	No	No	✓	No

Data Store Investigation

- Performance
- DB File size
- Features





Los Alamos NATIONAL LABORATORY EST. 1943

TESTING

Testing

- Testing
  - Data store routines vs. data store routines with wrapper code vs. MDHIM routines
  - Insert scaling study
  - Against Map Reduce style and other key-value data stores
- Test Harness
  - Simulates application inserting records in parallel
- Benchmarks
  - TPC-C, …?
  - Sort benchmarks

RESEARCH QUESTIONS AND FUTURE WORK

What are the Research Questions?

- Can massively parallel indexing be done on existing commercial file systems and not require new data intensive style file systems?
- Can we build a multi-dimensional indexing system that effectively represents petabytes with only mega-gigabytes of representation data?
- Can the application of “shared some” as apposed to shared nothing be done using supercomputing class interconnect technology and how would this “shared some” environment help over the current “shared nothing” fad (is there something in between map-reduce and transactional SQL)?

LA-UR-12-10467 and LA-UR-11-11964

Current and Future Work

- Current
  - Full functionality/tested
  - Work through scaling issues
  - Enhanced flush statistics
  - Test against competing distributed parallel data stores
- Future
  - Rebalance of range servers
  - Restart capability
  - User defined key comparison
  - User defined statistical function

Summary

- Multiple dimensions of global indices
- Multiple local indices
- Efficient queries
- “Stab” or range queries into global indices
- Range queries into local indices
- Range queries over combinations of global and local indices
- Can be treated as global index system
- Can be treated as only local index system (like Hive etc.)
- Utilizes interconnect present in HPC systems (not really present in classical data intensive systems) to provide:
  - Rich key distribution information to make queries optimized and parallel
  - Representing petabytes of record information with megabytes of key distribution information

Acknowledgements

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.

Questions?

James Nunez (jnunez@lanl.gov)

LA-UR 12-10467 and LA-UR-11-11964