RAPIDS
Reconciling Availability, Accuracy, and Performance in Managing Geo-Distributed Scientific Data

Lipeng Wan, Jieyang Chen, Xin Liang, Ana Gainaru, and others

Muhammad Raees

mr2714@rit.edu

Objective

Increase the performance of geo-distributed data (scientific) storage systems by

- Reducing storage and network overhead
- Improving data access/latency

While maintaining the same level of data

- Availability
- Accuracy

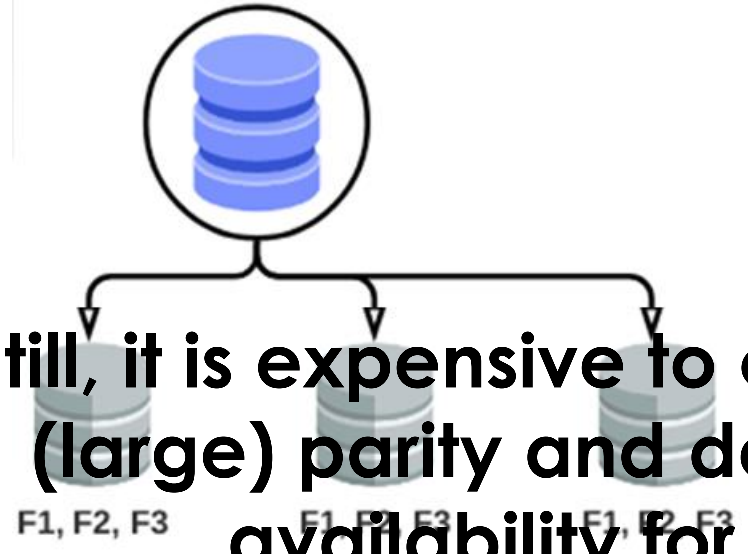
Data Size Grows

- Large scientific datasets
  - XGC gyrokinetic particle-in-cell code
  - Square Kilometer Array (SKA)
- High-Performance Storage Needs
- Managing failures is critical
  - Availability, accuracy, and performance

Improving Resilience

Data Duplication

- Availability Improvement
- Storage and Network Overhead

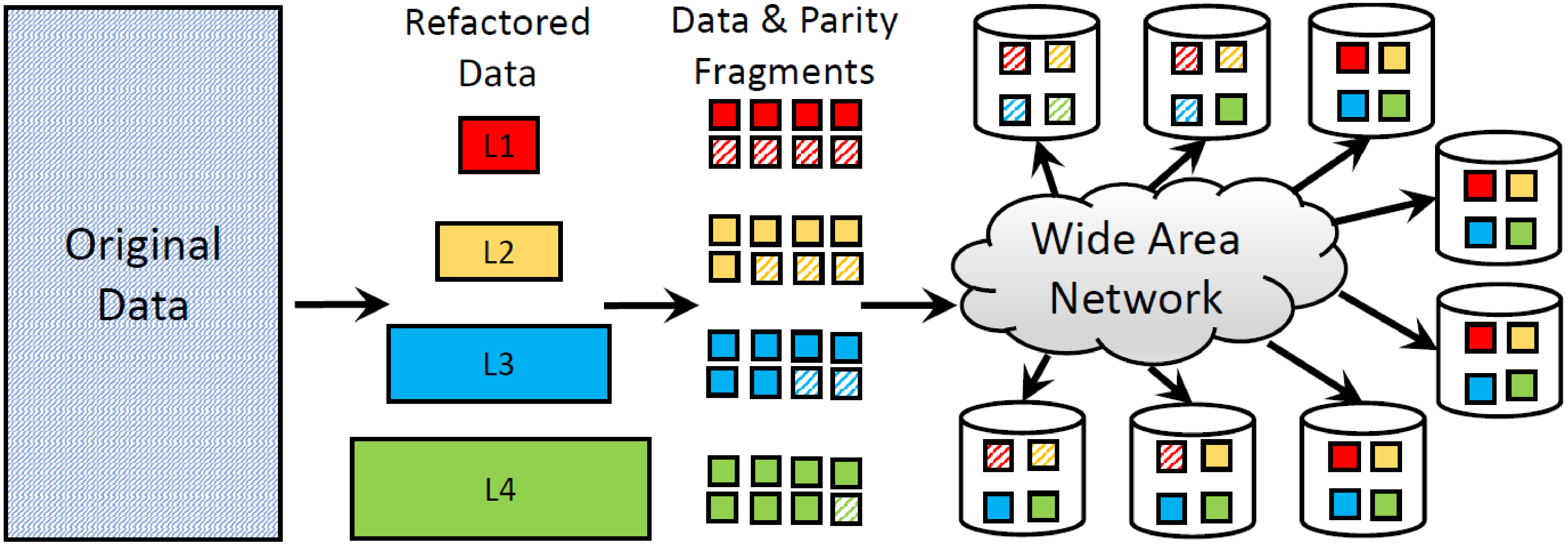


Erasure Coding

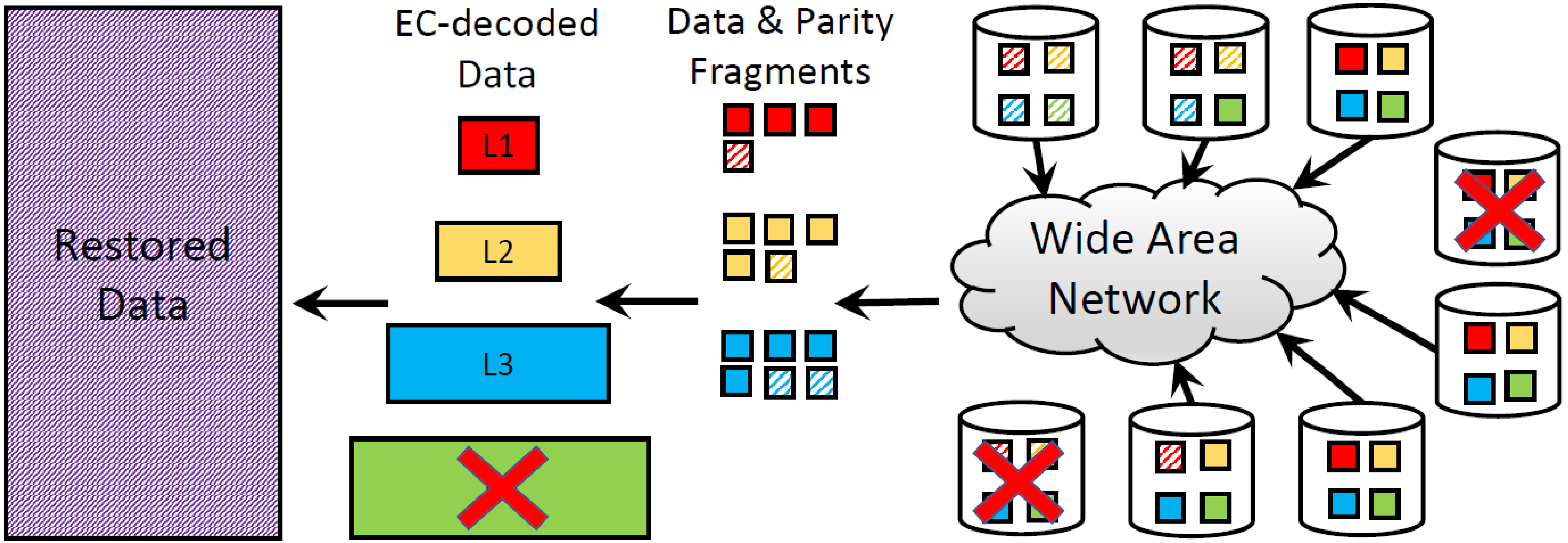
- Splitting data into data and parity fragments
- Data restoration from a subset of fragments

Still, it is expensive to create, transfer, and store many (large) parity and data fragments to achieve high availability for large scientific datasets.

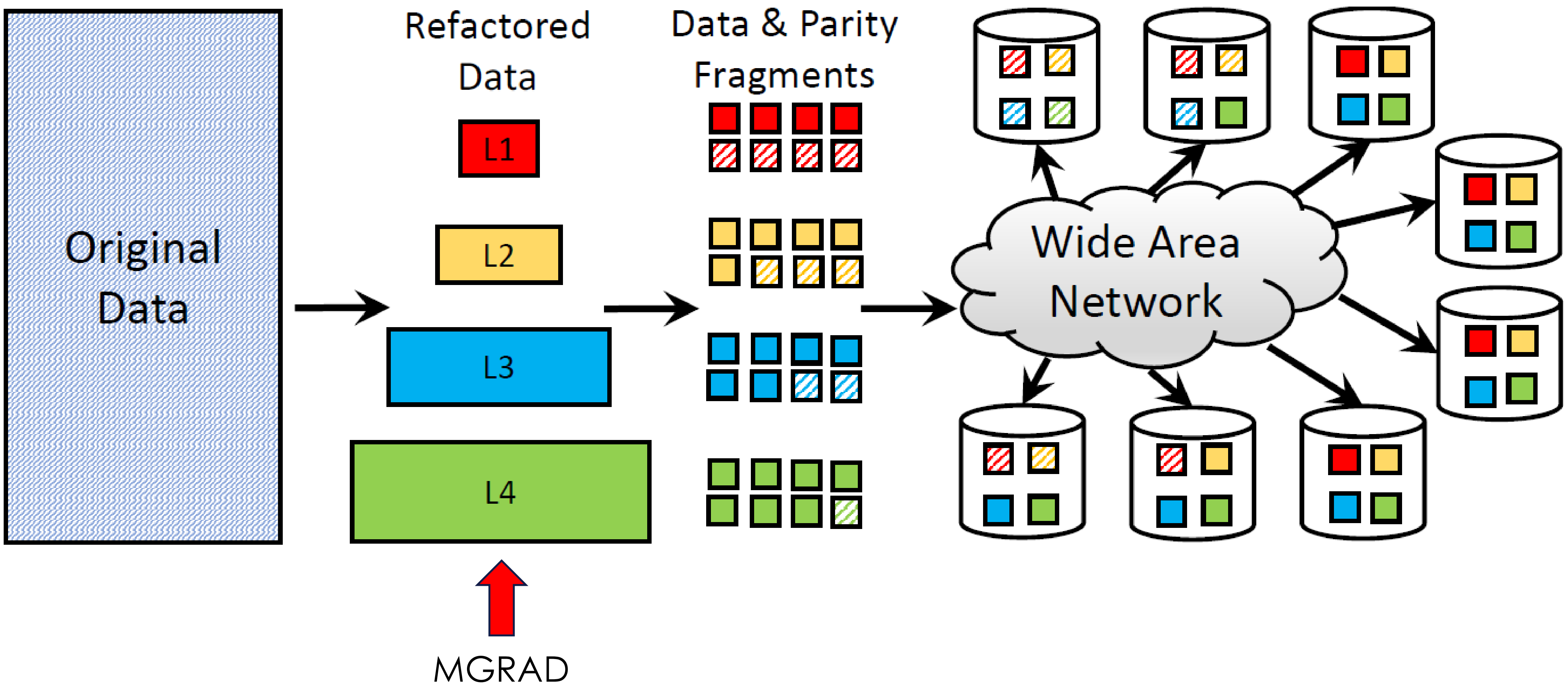
RAPIDS – Store Data



RAPIDS – Get Data



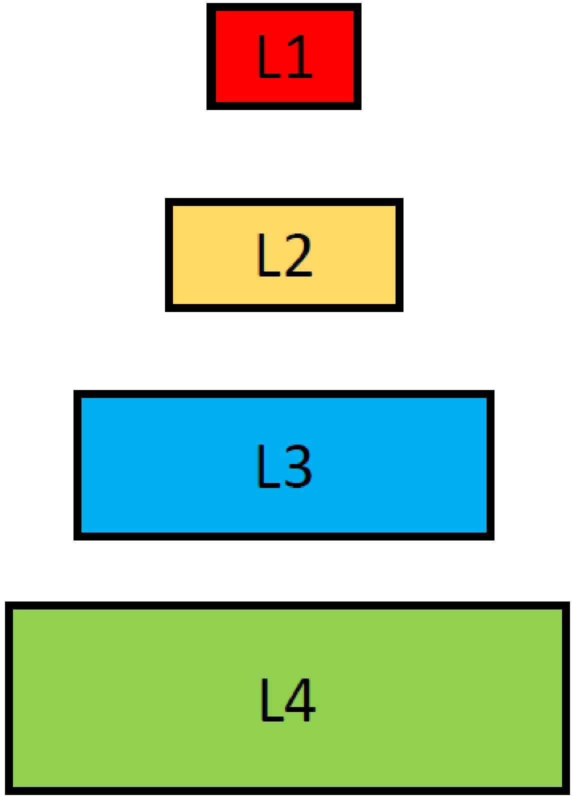
Refactor & Compress



Refactor & Compress

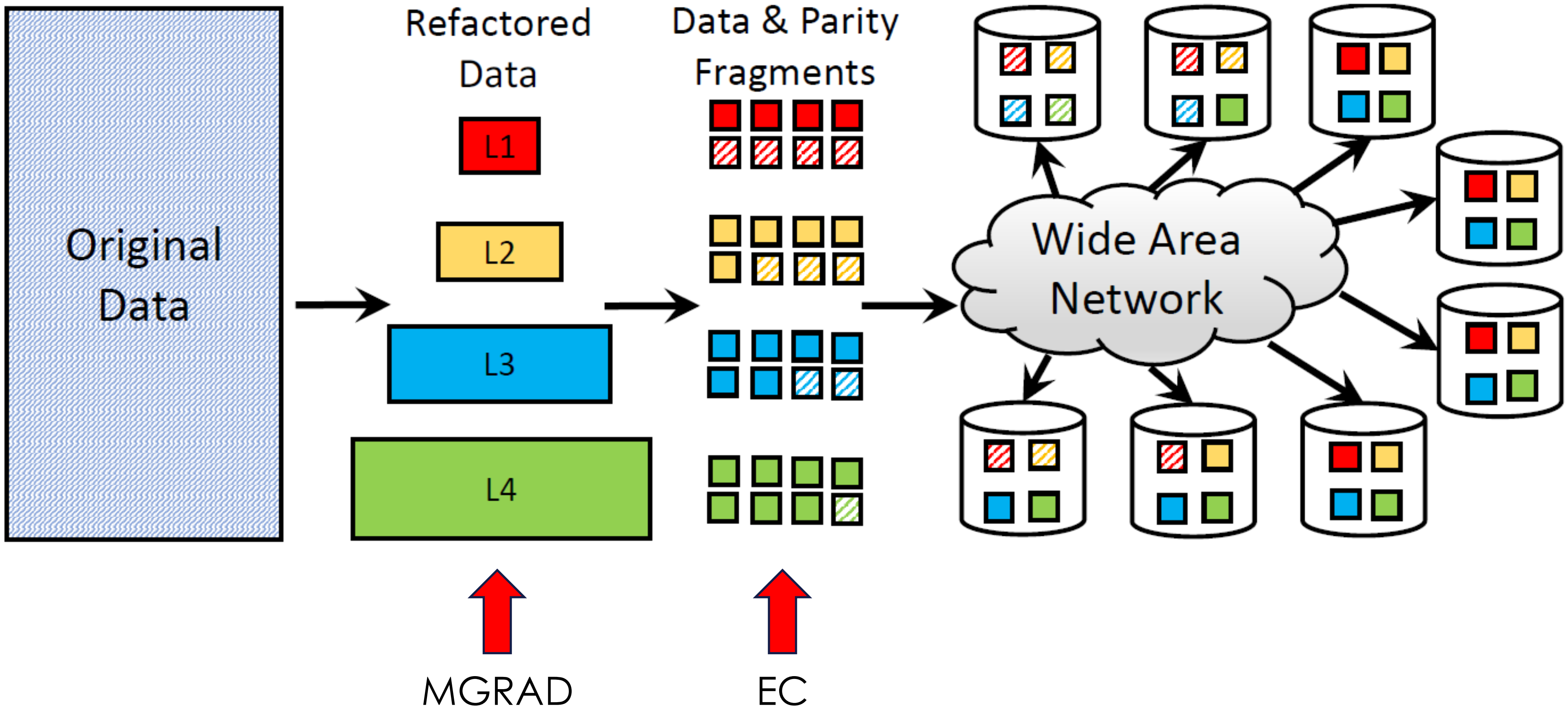
- Error-bounded lossy compression
- Refactor the original data into a hierarchical representation
  - E.g., 4 levels
  - Size: s1 < s2 < s3 < s4
- Data reduction is applied
  - Error in the reduced representation
  - Size: S > s1 + s2 + s3 + s4

Refactored Data



https://github.com/CODARcode/MGARD

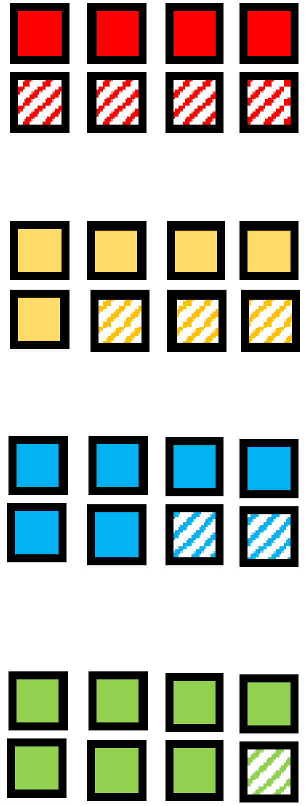
Erasure Coding



Erasure Coding

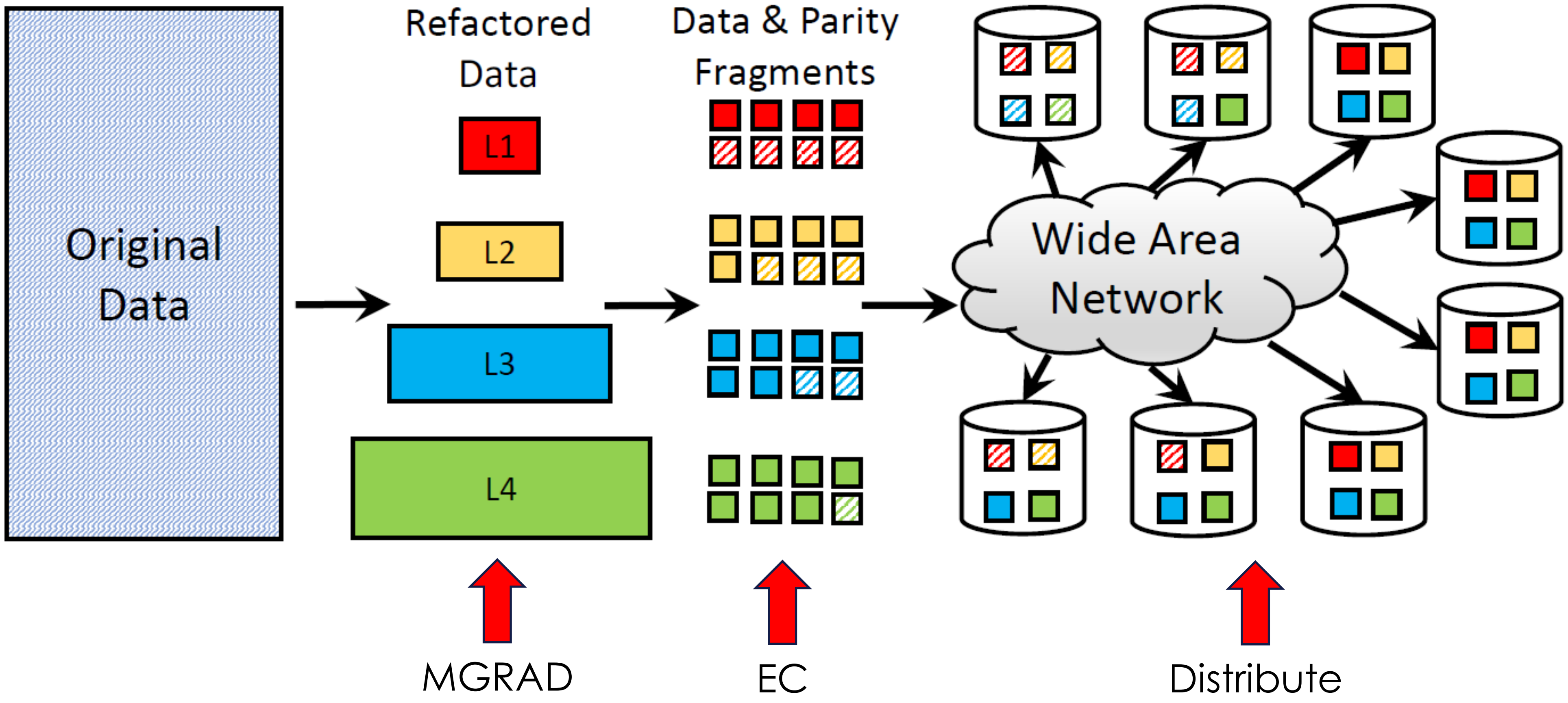
- Erasure coding (each level) with <u>fault tolerance</u>
  - Encoded into $n{-}mj$ data and mj parity
  - Parity $m1 > m2 > m3 > m4$
- Reduces storage and network overhead compared to whole data compression
  - Data s1 < s2 < s3 < s4
    - Smaller duplication overhead.

Data & Parity Fragments



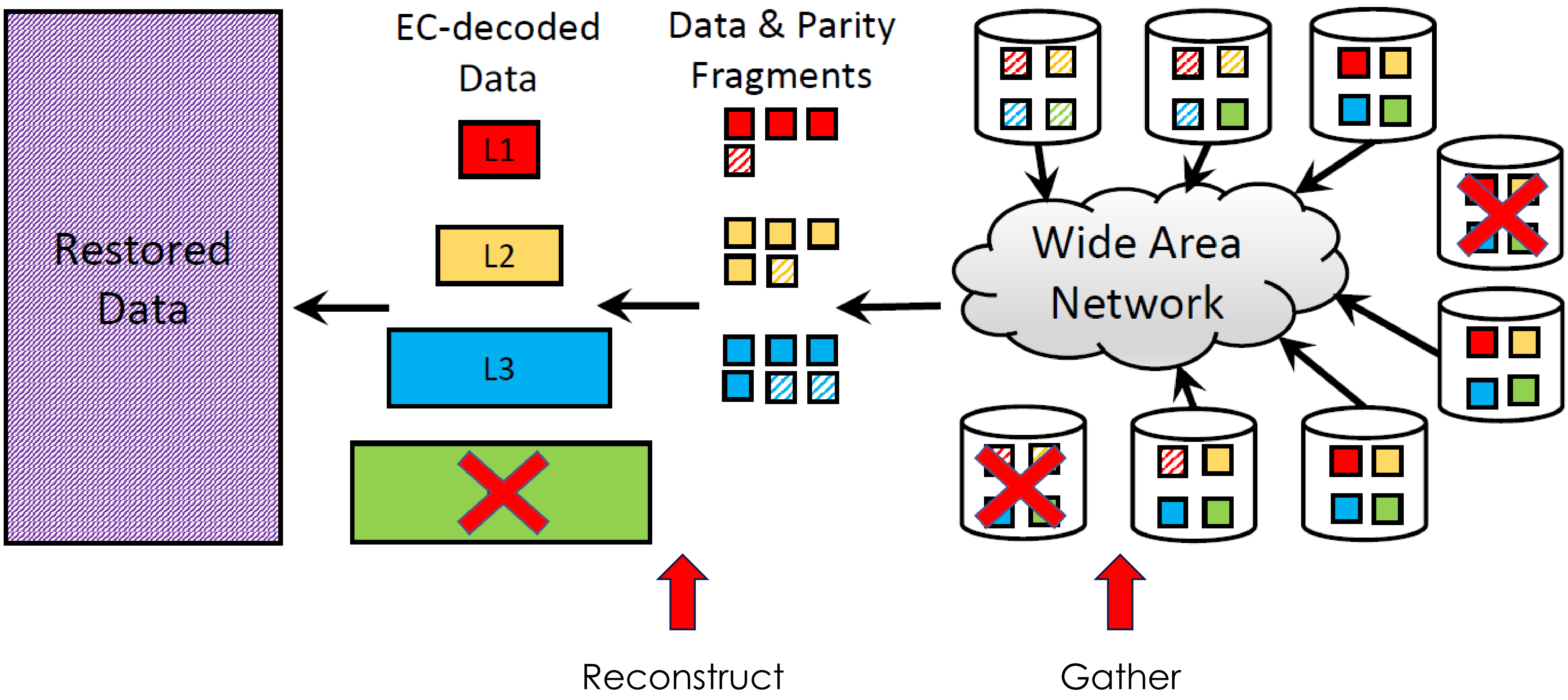
https://github.com/openstack/liberasurecode

Distributed Storage



https://github.com/globus/globus-cli

RAPIDS – Get Data



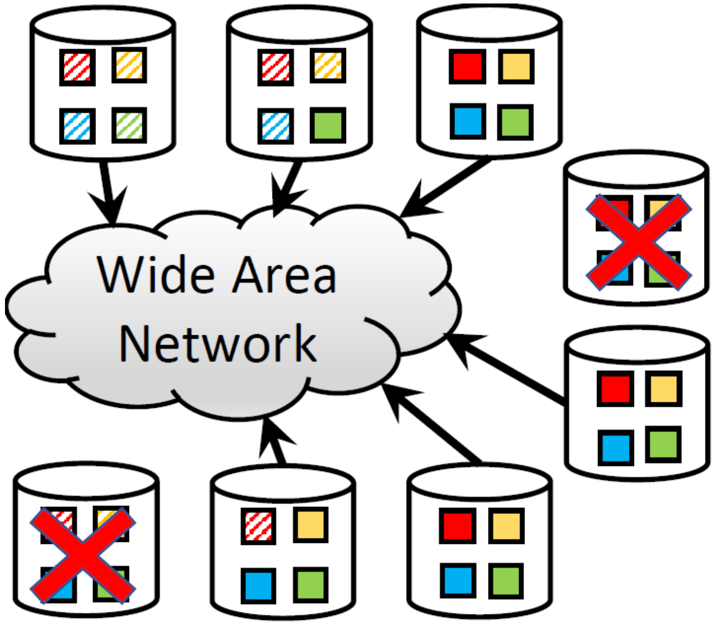
RAPIDS – Get Data

- Data gathering optimization
  - Which fragment fetched from which source
    - Network overhead / latency

- Reconstruction – Fault tolerance configuration is critical for availability and accuracy
  - Minimize the expected relative error

Data Gathering

- Find a set with values of enough fragments (fragments, systems) to minimize data transfer time
  - Find time for each transfer request (including multiple request sources)
  - Reduce average time for all requests

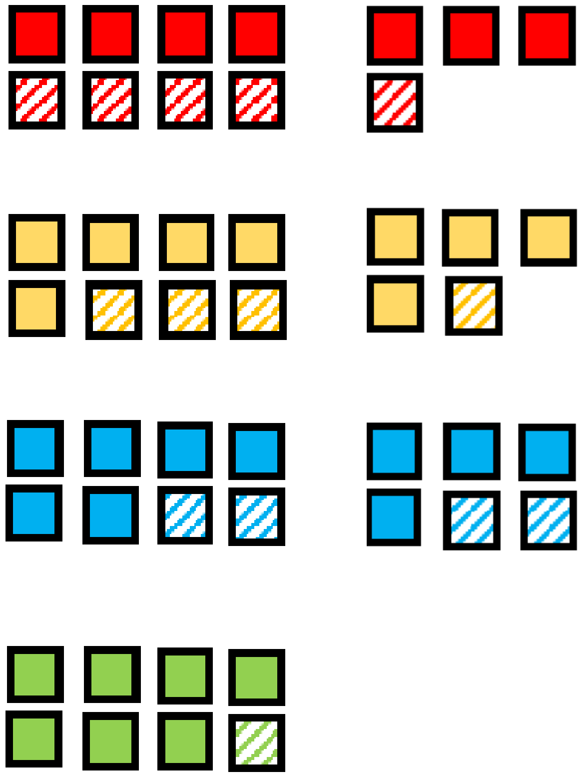


- Fetch a subset of fragments
  - Prioritize high-bandwidth
  - Reduce concurrency

Fault Tolerance

- Erasure coding defines data restructuring
  - how many failures each level can sustain
- If failure (N) > m1,
  - Data will be unavailable for reconstruction
  - Max error (1), data is useless
- If $m_{j+1} < N <= m_j$,
  - Data will be available for reconstruction
  - Error ej

Data & Parity Fragments



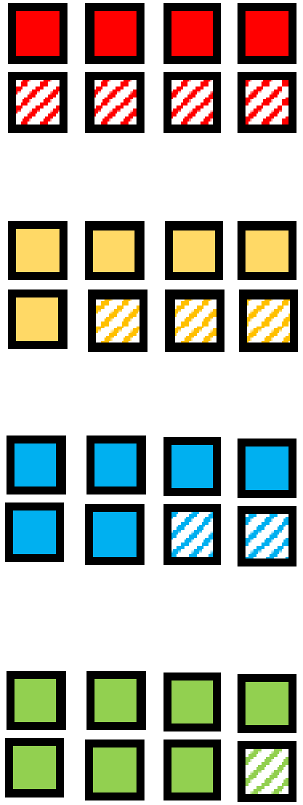
Fault Tolerance

- Minimize faults (each level) to reduce error under constraints (optimization)
  - Storage overhead should not exceed the threshold (defined)
  - Number of failures at each level can tolerate should follow $m1 > m2 > . . . > ml \geq 1$ as $(s1 < s2 < . . . < sl\ )$

- Small solution space – Brute-force search
  - $O(\text{Storage systems} - \text{refactor levels})^4$
- Large solution space – Heuristic search

Fault Tolerance Heuristic

Data & Parity Fragments



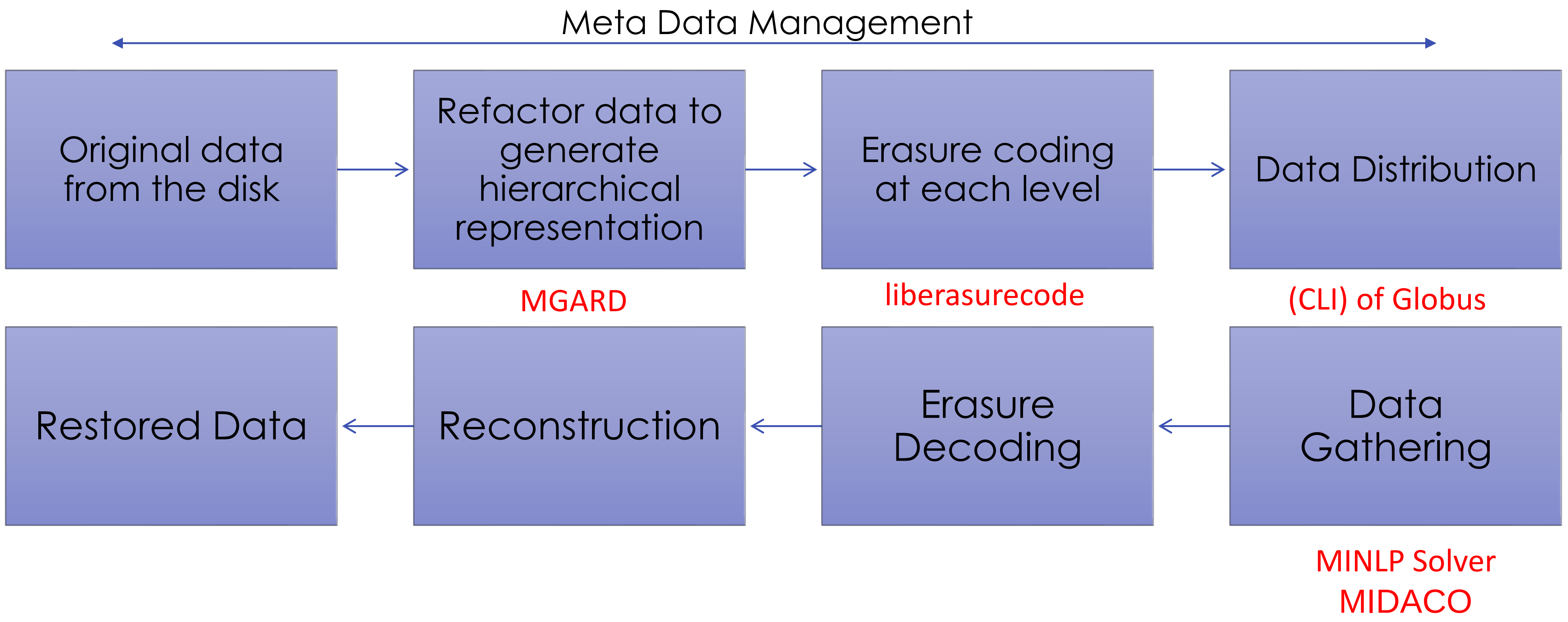
1. Set initial fault tolerance configuration w.r.t. overall storage overhead – reducing search space
2. Increment parity on the level (bottom start)
3. If storage overhead is violated
   i. Reduce/use the last stable configuration
   ii. Move to the level above in the hierarchy and go to step 2

Assumptions

i. $s1 \ll s2 \ll s3 \ll s4$

ii. $e1 \gg e2 \gg e3 \gg e4$

Recap


Meta Data Management
Original data from the disk
Refactor data to generate hierarchical representation
Erasure coding at each level
Data Distribution
MGARD
liberasurecode
(CLI) of Globus
Restored Data
Reconstruction
Erasure Decoding
Data Gathering
MINLP Solver
MIDACO

Meta Data Management

- Track all the data and parity fragments
  - key-value database (fragment: system)
  - Restoring lost fragments – EC (fragment: system)
  - Reconstruction information
  - Central – Prone to failure

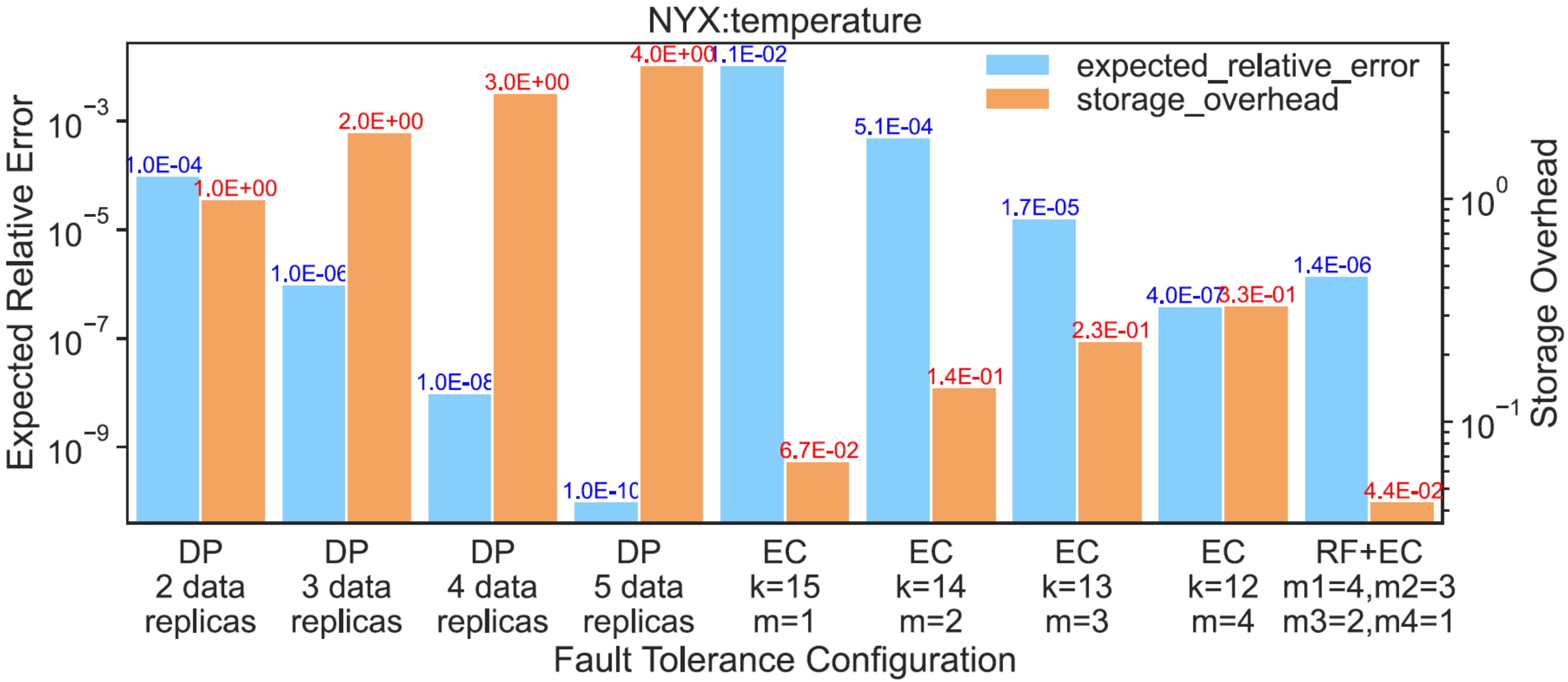
Evaluation – Data Sets

- NYX cosmology simulation
- SCALE-LETKF weather simulation
- Hurricane Isabel climate simulation

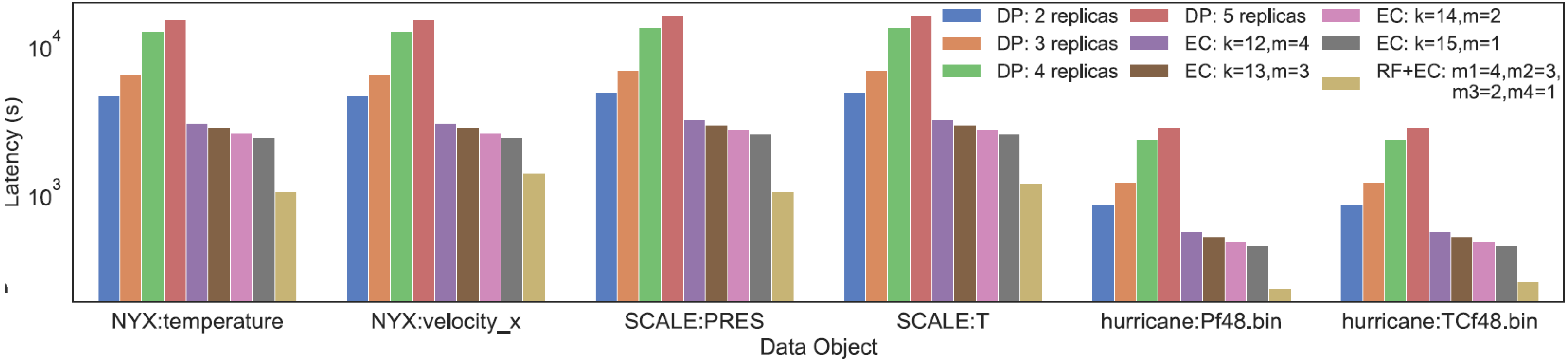
Dataset	Object name	Size/object
NYX	temperature	16TB
NYX	velocity_x	16TB
SCALE-LETKF	PRES	16.82TB
SCALE-LETKF	T	16.82TB
Hurricane Isabel	Pf48.bin	2.98TB
Hurricane Isabel	TCf48.bin	2.98TB

- Oak Ridge Leadership Computing Facility (OLCF)
  - 704-compute node cluster; where each node has
    - 2 16-core Processors (32 cores)
    - 256GB of memory
  - 9 GPU nodes; where each node has
    - 2 NVIDIA K80 GPUs, 2 Intel Xeon 14-core processors
    - 1TB memory

Evaluation – Error and Storage

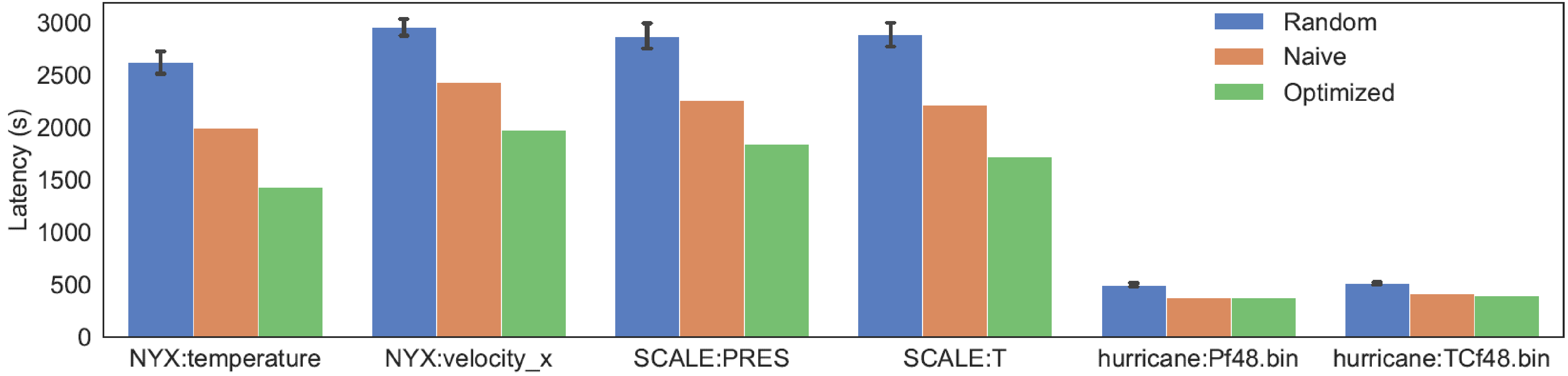


Evaluation – Latency



Evaluation – Data Gathering

Average over 50 iterations

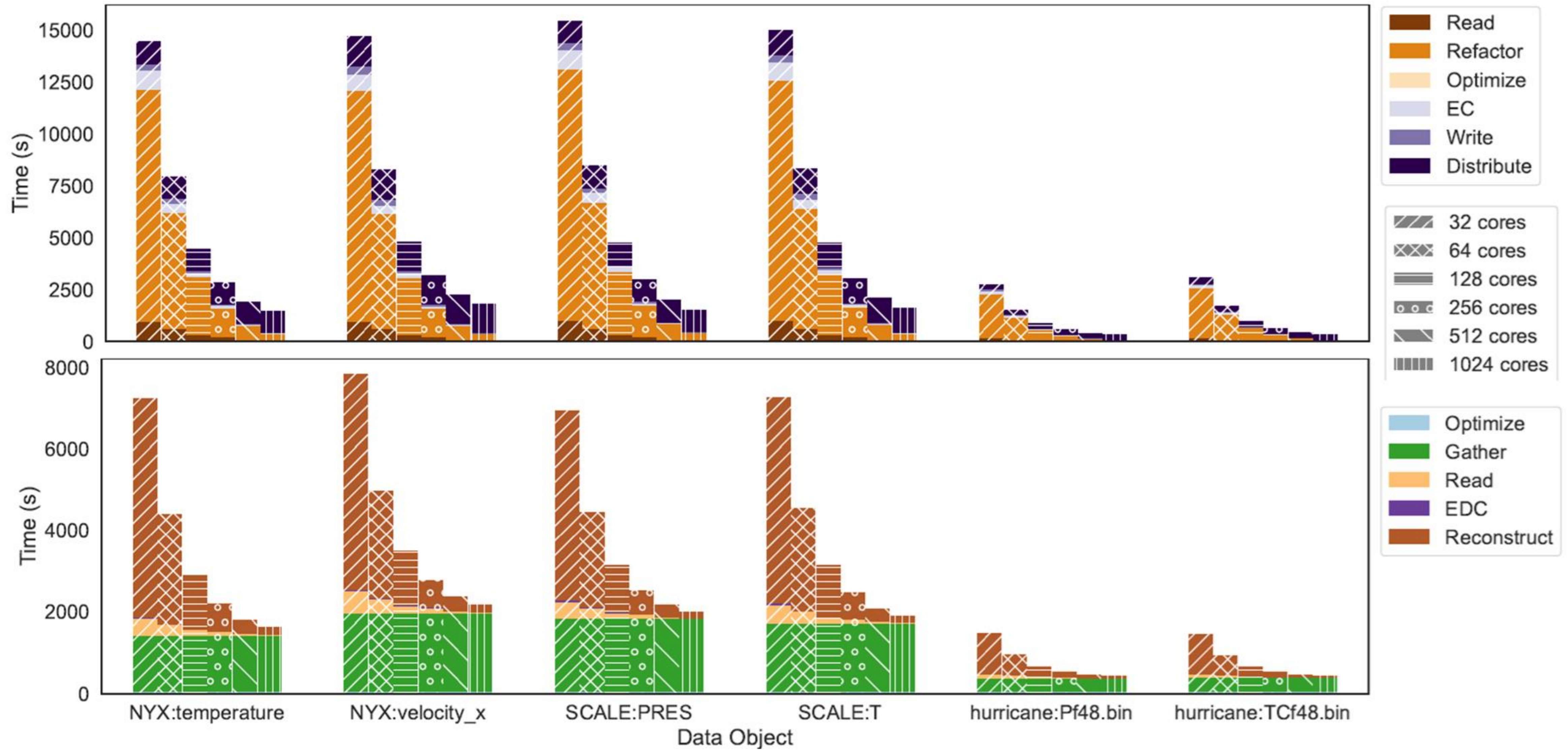


Random: varying the seed
Naïve: Sort by bandwidth and select from top-bottom
Optimized: Naïve + MIDACO

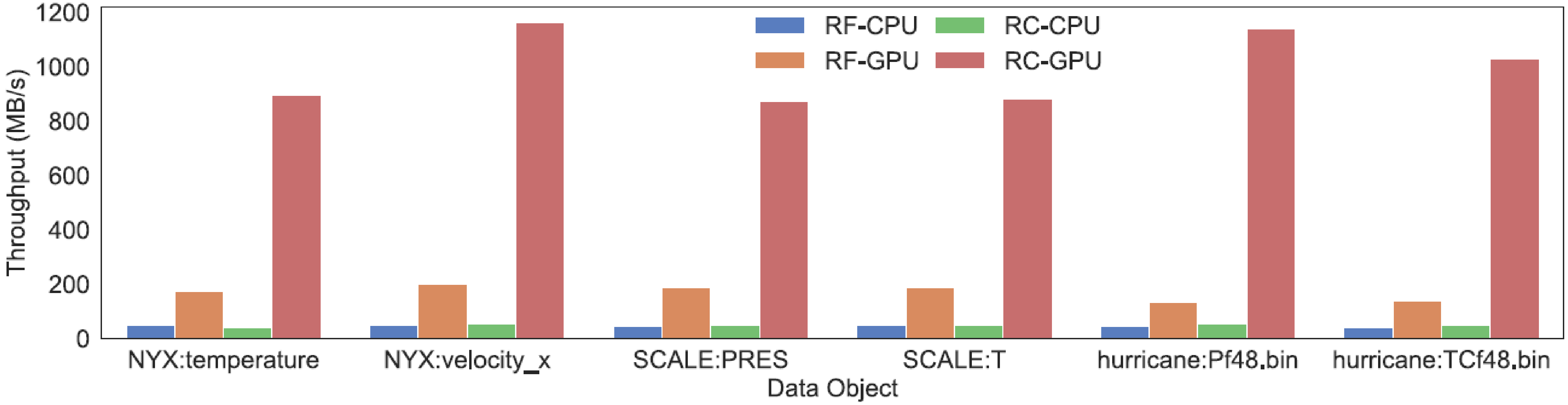
Evaluation – FT Heuristic

Data object	Optimal FT Configuration		Speedup (t_{BF}/t_H)
	Brute-Force	Heuristic	
NYX:temperature	[8,5,4,2]	[8,5,4,2]	124
NYX:velocity_x	[5,4,3,1]	[5,4,3,1]	152
SCALE:PRES	[9,6,4,2]	[9,6,4,2]	114
SCALE:T	[7,4,3,2]	[7,4,3,2]	137
hurricane:Pf48.bin	[11,10,8,7]	[11,10,8,7]	152
hurricane:TCf48.bin	[11,9,8,6]	[11,9,8,6]	137

Evaluation – Performance



Evaluation – Performance



RAPIDS

- Two optimization models.
  - A mixed integer nonlinear programming (MINLP) solver for gathering data – MIDACO
  - A heuristic algorithm compared to the brute-force search for optimizing fault tolerance
- Evaluation results show
  - Minimum expected relative error
  - Reduced storage overhead
  - Higer data latency

Thank You

Optimal Fault Tolerance Configurations

Input: Initial fault tolerance configurations $M^i = [m_1^i, m_2^i, \ldots, m_l^i]$
Output: Optimal fault tolerance configurations $M^o = [m_1^o, m_2^o, \ldots, m_l^o]$
$M = [m_1, m_2, \ldots, m_l] = M^i$, $l_{curr} = l$, $M_{prev} = []$;
while *True* **do**
  $W = \frac{\sum_{j=1}^{l} \frac{m_j}{n-m_j} s_j}{S}$;
  if $W < \omega$ **then**
    foreach $1 \leq x < l_{curr}$ **do**
      $m_x = m_x + 1$;
    end
    $M = [m_1, m_2, \ldots, m_l]$;
  else
    foreach $1 \leq x < l_{curr}$ **do**
      $m_x = m_x - 1$;
    end
    $M = [m_1, m_2, \ldots, m_l]$;
    $l_{curr} = l_{curr} - 1$;
  end
  if $M == M_{prev}$ **then**
    break;
  end
  $M_{prev} = M$;
end
$M^o = M$;

PMGRAD

- PMGARD, multilevel decomposition approach, with an error-controlled data refactoring and reconstruction framework for scientific data.
  - Decomposing original data into multiple components
  - Multilevel coefficients, at relative importance to the precision of the reconstructed data, are used to reconstruct the approximation of the original data