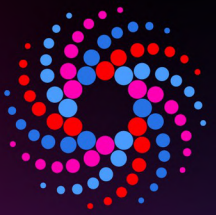


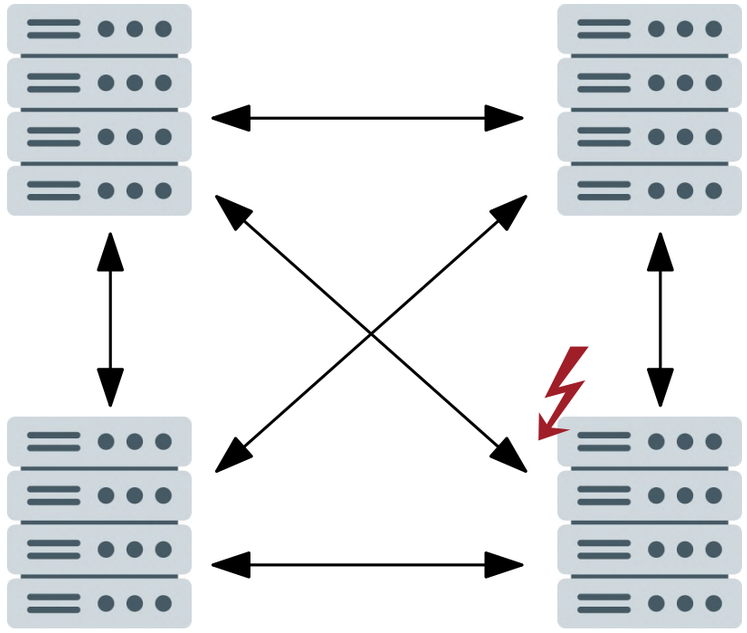
SC22
Dallas, TX | hpc accelerates.

ReStore: In-Memory REplicated STORagE for Rapid Recovery

Lukas Hübner, Demian Hespe, Peter Sanders, Alexandros Stamatakis

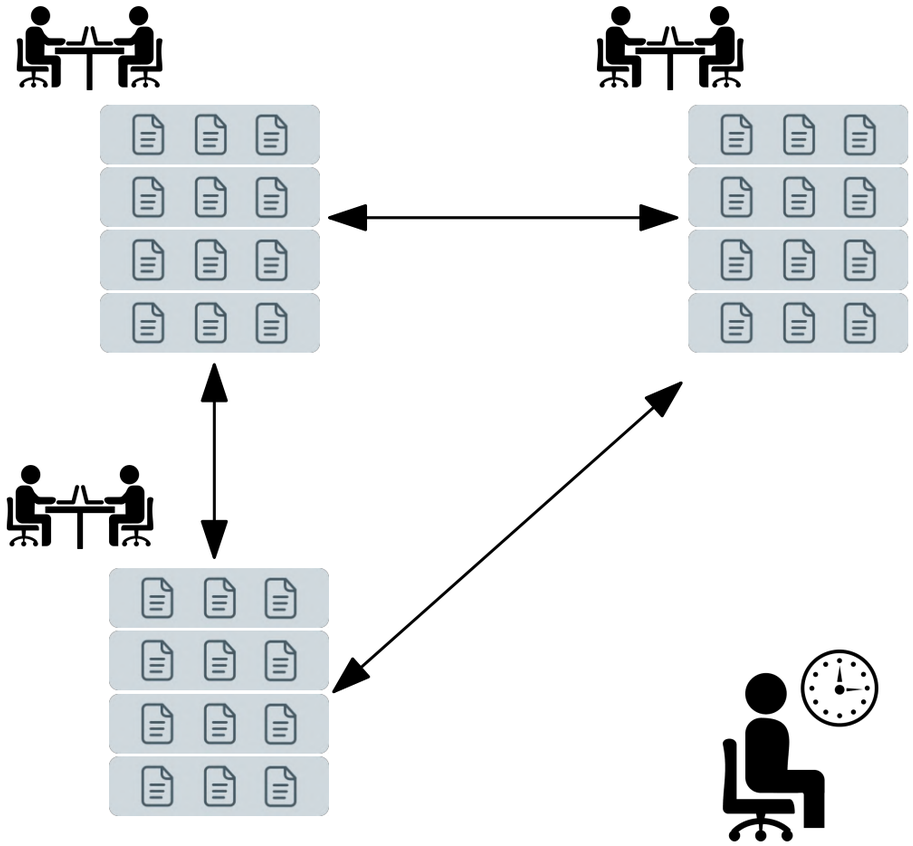
Motivation

- fail-stop faults
- More CPUs→ more faults → more recoveries
- Lower operational voltage of CPUs → less energy used, more faults
- Node failure → reload *dynamic* program state and *static* (e.g., input) data
- The parallel file system is a bottleneck



Shrinking vs Substituting Recovery

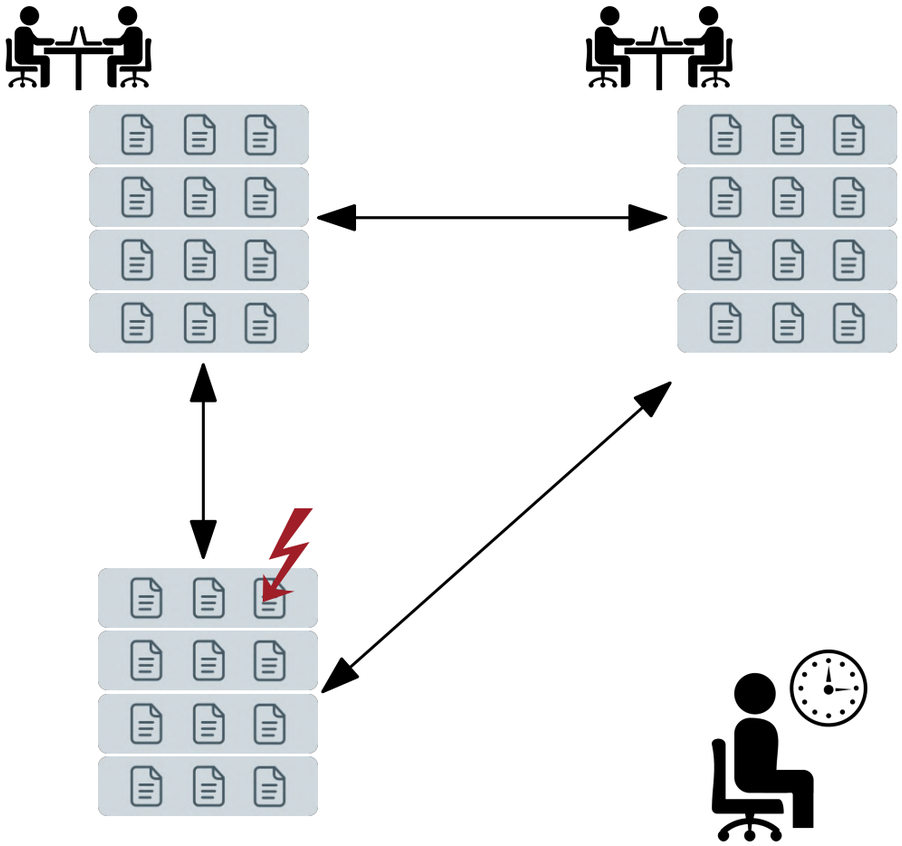
Substituting Recovery



Shrinking Recovery

Shrinking vs Substituting Recovery

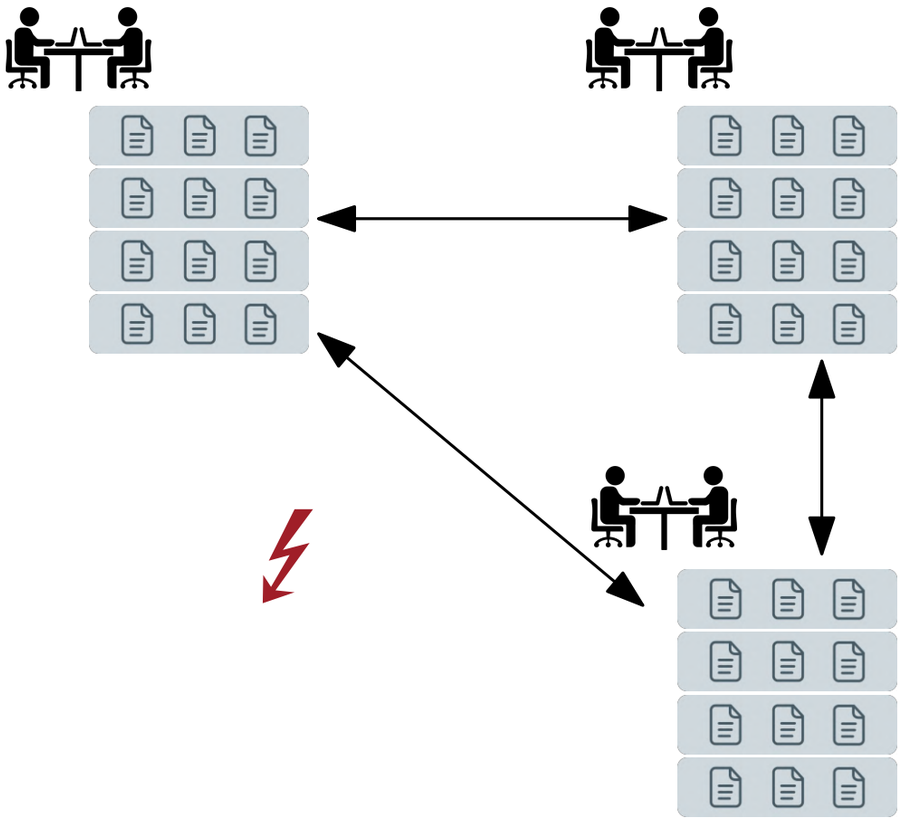
Substituting Recovery



Shrinking Recovery

Shrinking vs Substituting Recovery

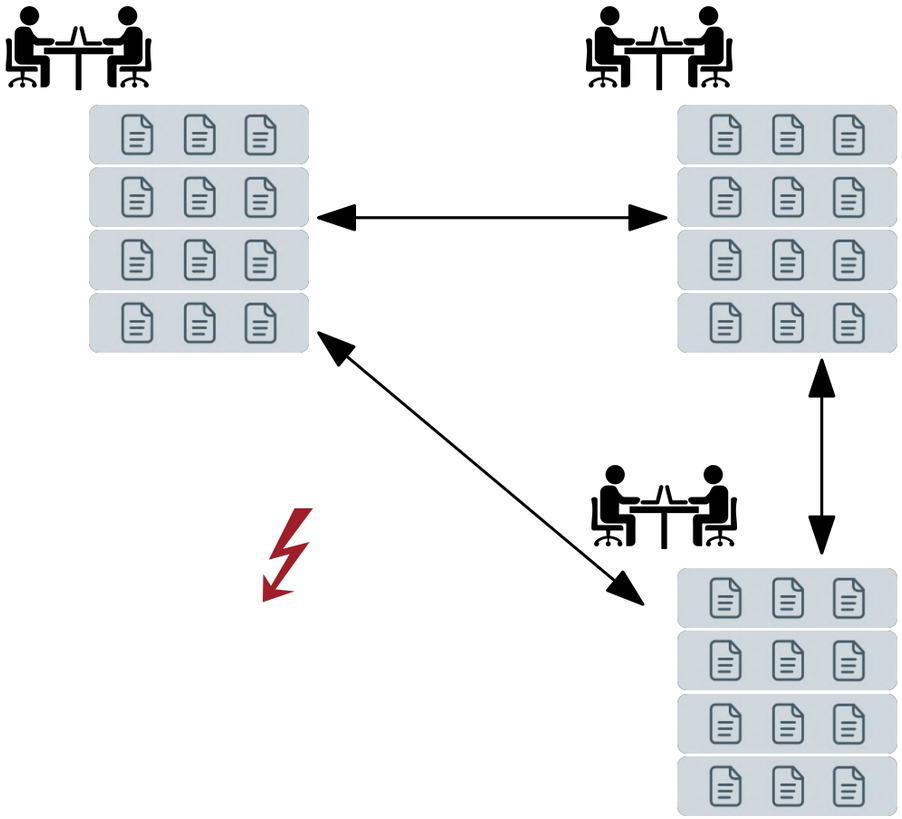
Substituting Recovery



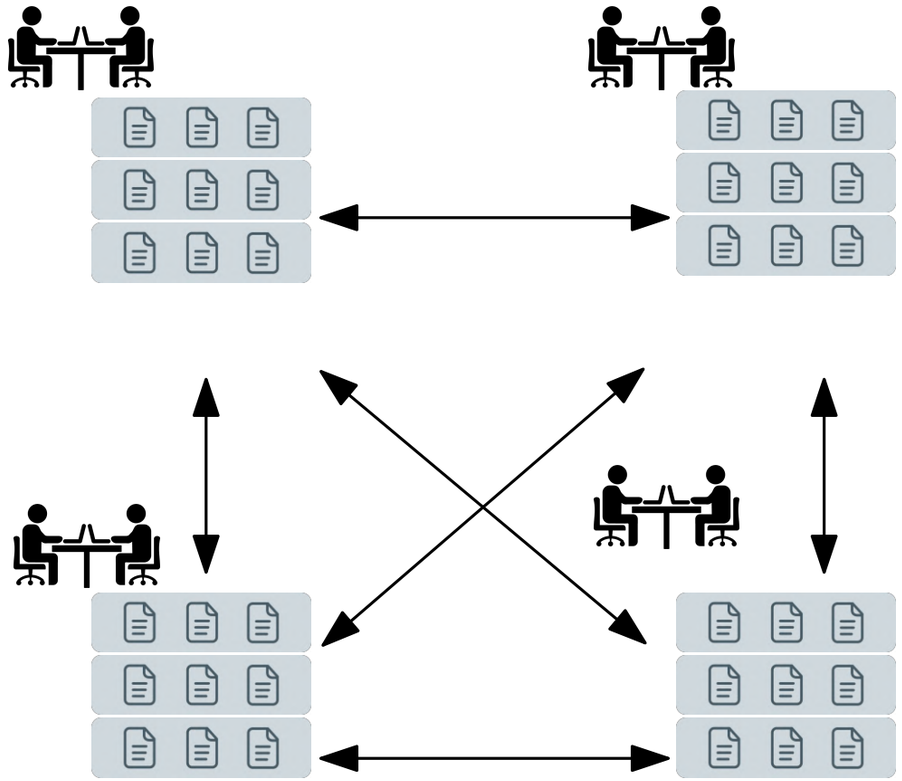
Shrinking Recovery

Shrinking vs Substituting Recovery

Substituting Recovery

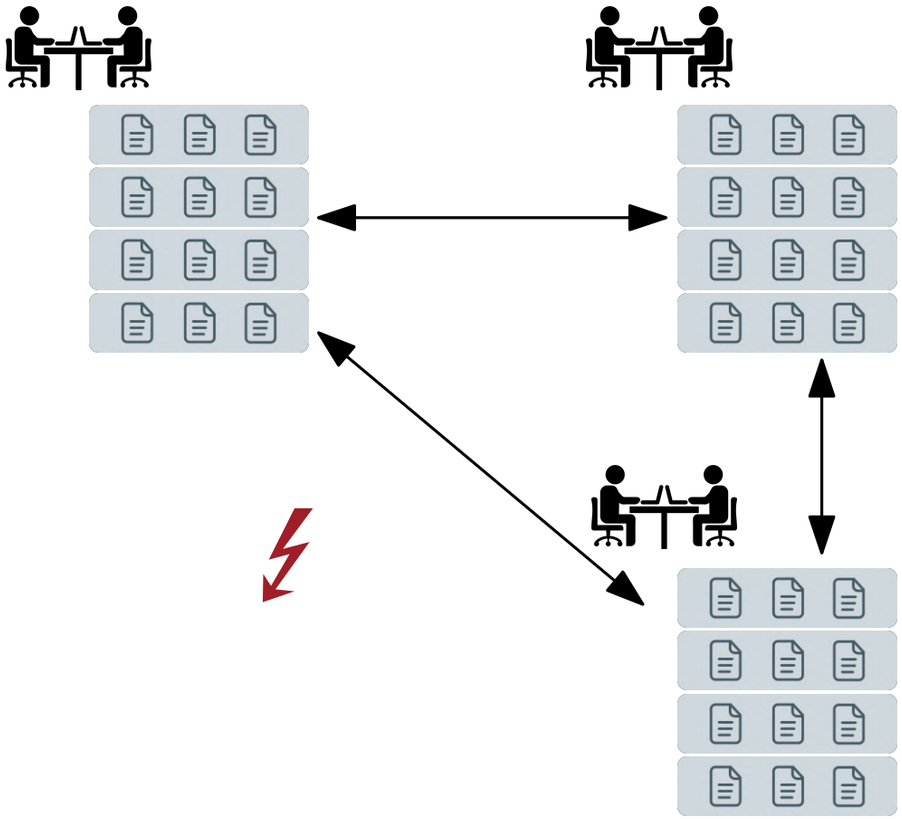


Shrinking Recovery

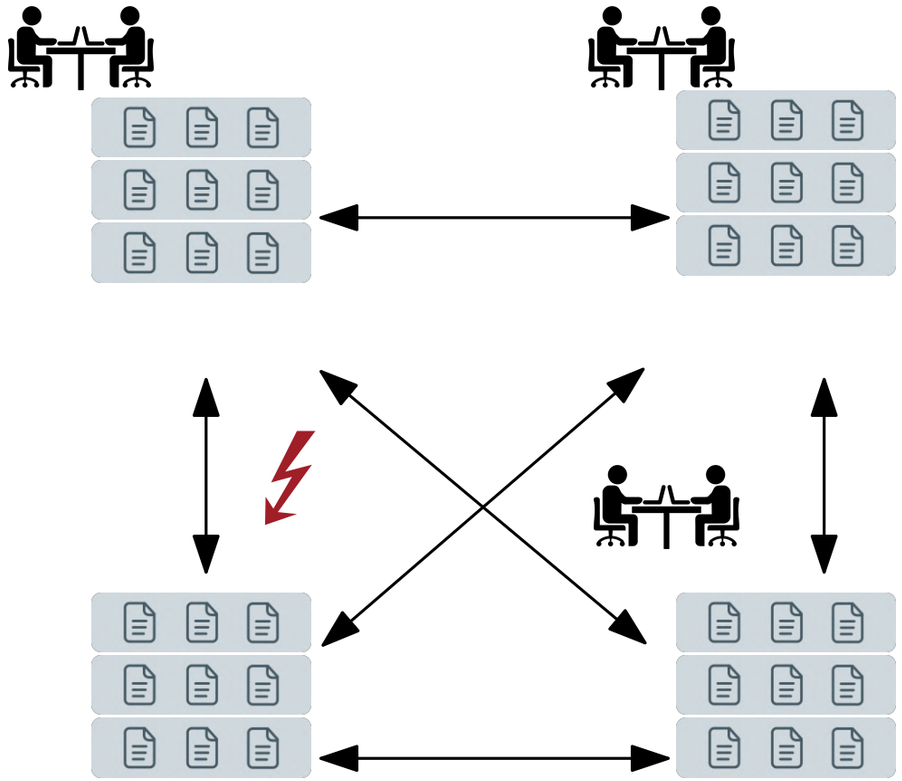


Shrinking vs Substituting Recovery

Substituting Recovery

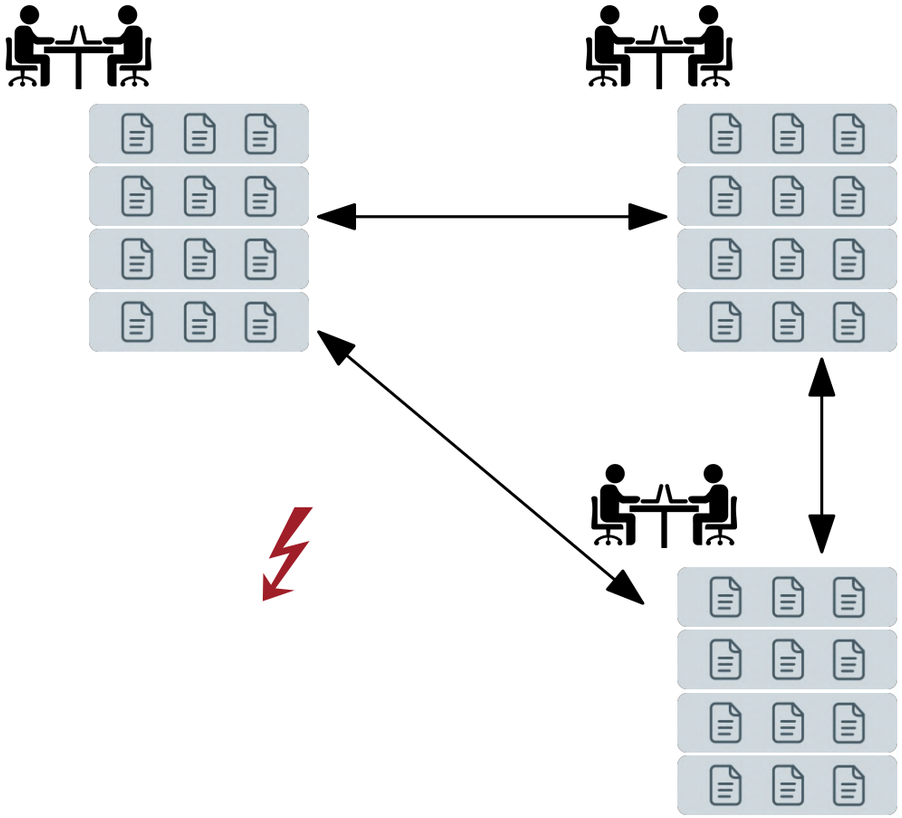


Shrinking Recovery

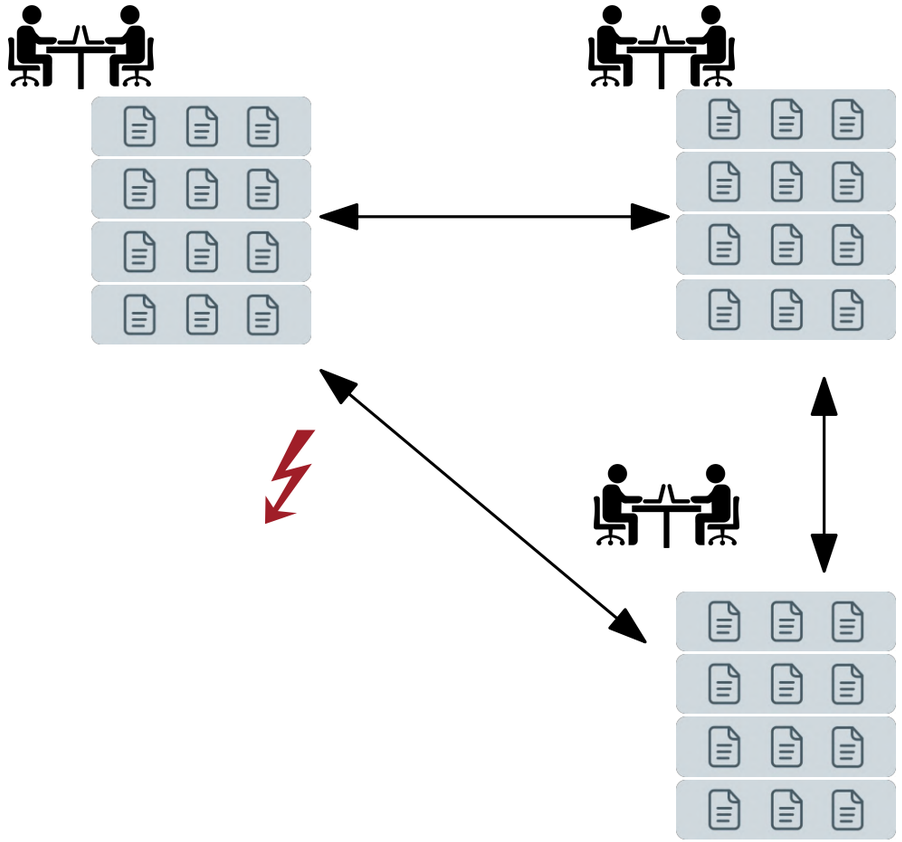


Shrinking vs Substituting Recovery

Substituting Recovery

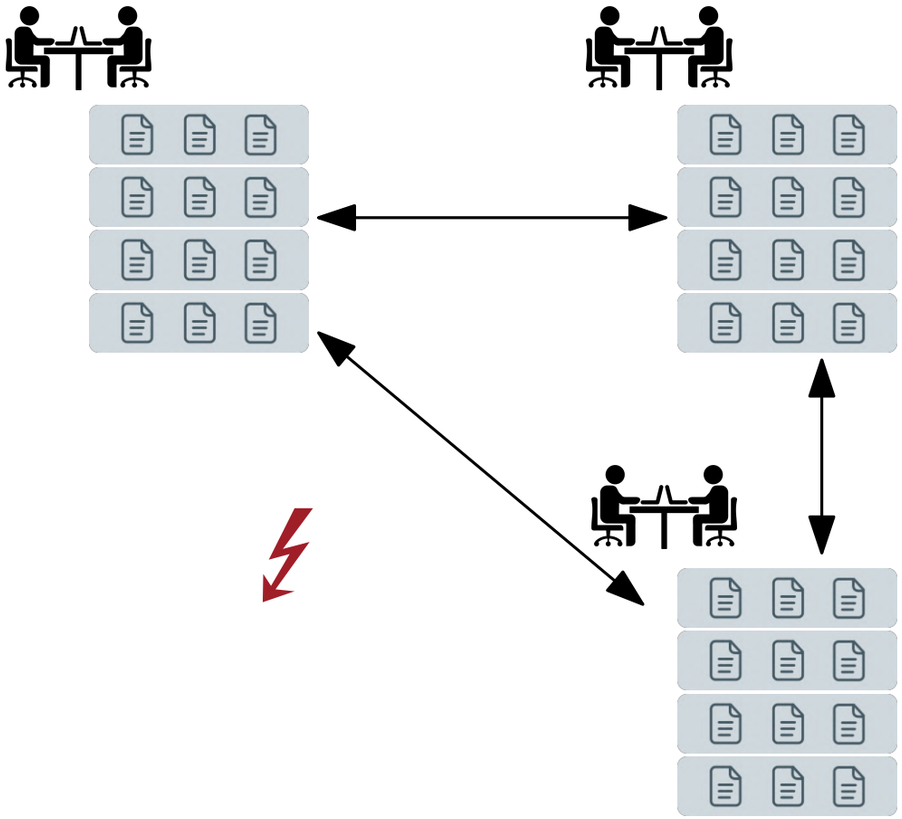


Shrinking Recovery



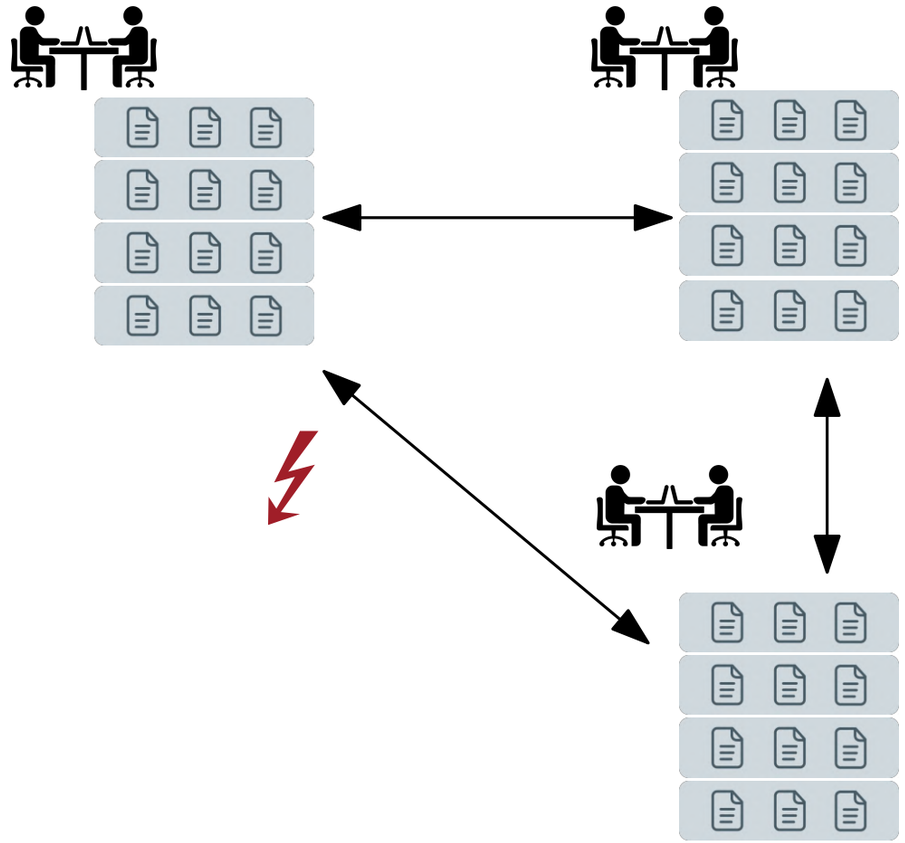
Shrinking vs Substituting Recovery

Substituting Recovery



- Up to $5\,\%$ of nodes idling
- Limited number of failures supported
- Recovery time does not scale

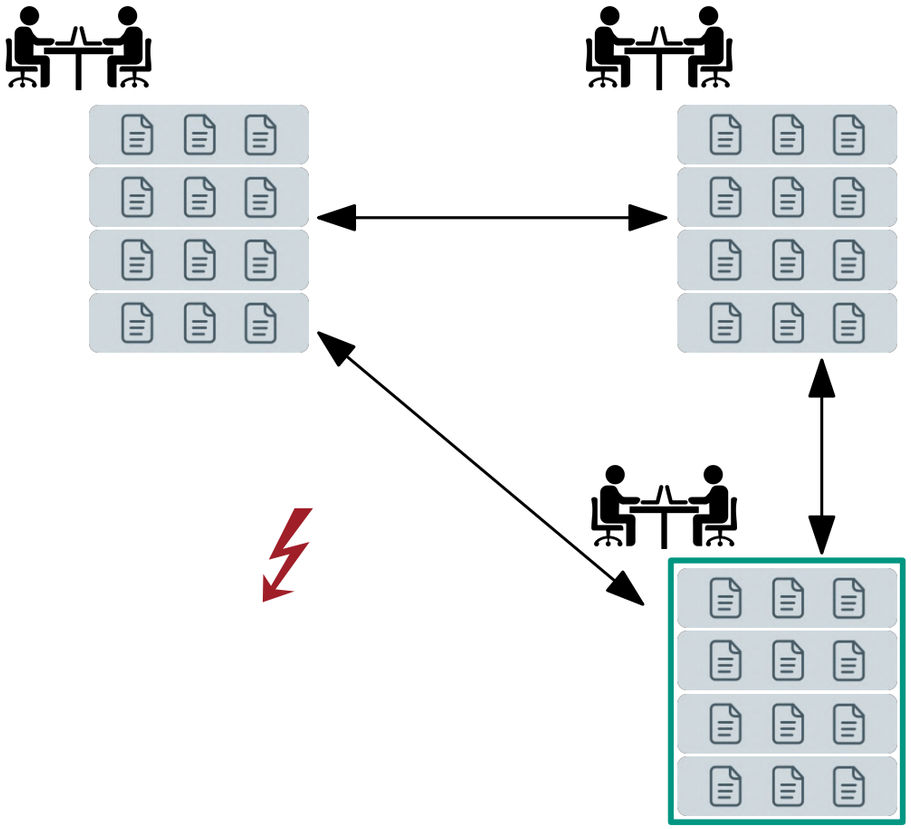
Shrinking Recovery



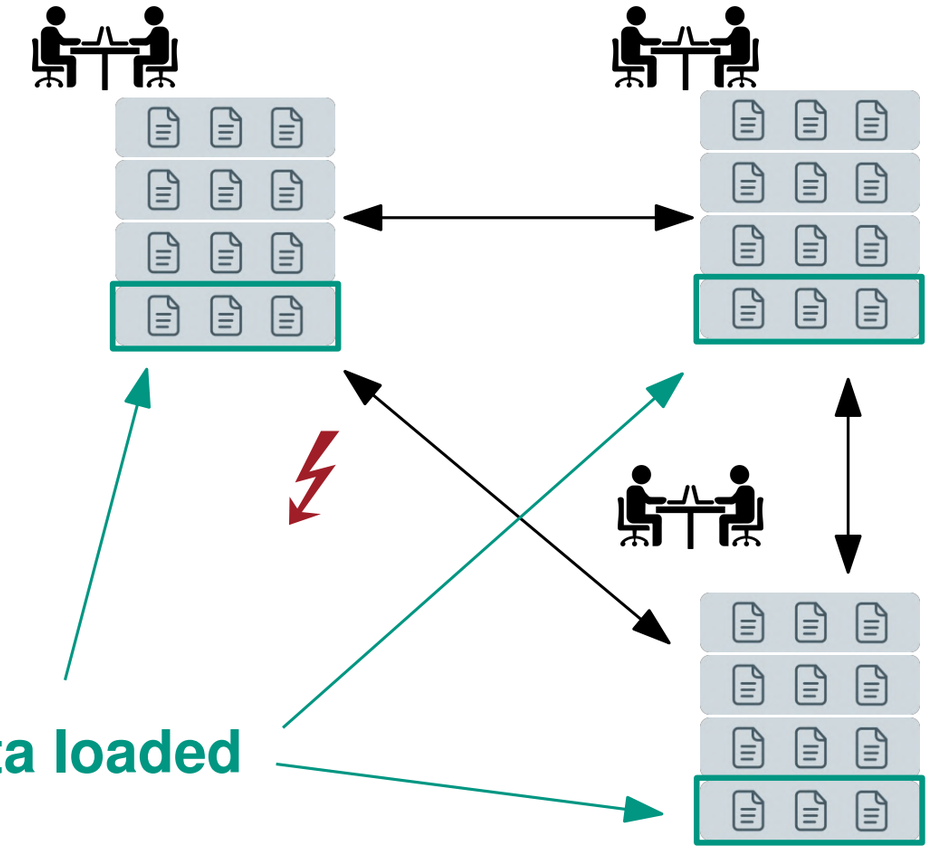
- All nodes participate in computation
- Unlimited number of failures supported
- Recovery time scales with $1/p$

Shrinking vs Substituting Recovery

Substituting Recovery



Shrinking Recovery



Single PE receives all messages
→ **bottleneck**

Design Goals

ReStore

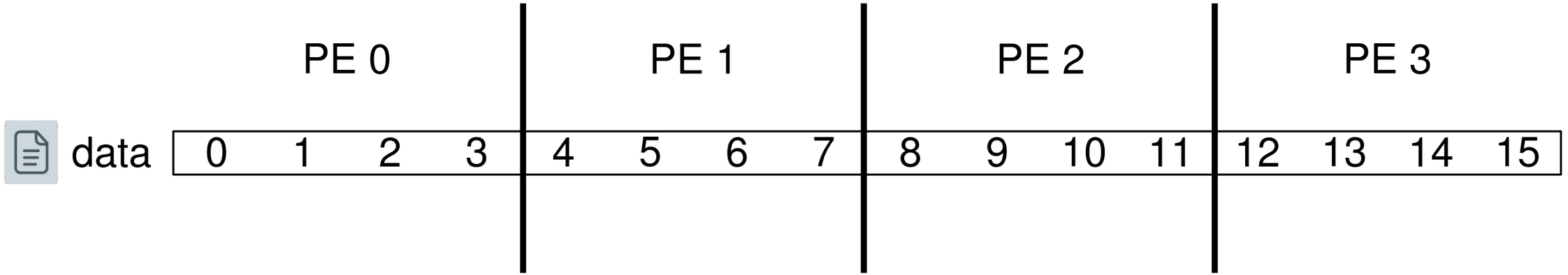
- **in-memory** access to the parallel file system is a bottleneck
- **no spare nodes required** spare nodes are wasted resources
- **no checkpointing nodes required** checkpoint nodes are wasted resources
- **scalable recovery** $\in \mathcal{O}(1/p)$ time per failure
- **arbitrary replication level** more flexibility and robustness
- **rapid recovery** that's what we needed for our application

Related Work

	ftRMA	Fenix	SCR	Lu	GPI_CP	*ReStore*
Features						
in-memory checkpointing	✓	✓	✗	✓	✓	✓
substituting recovery	✓	✓	✓	✓	✓	✓
shrinking recovery	✗	✗	✗	✗	✗	✓
all nodes participate in computation	✗[2]	(✓)[1]	(✓)[1]	✗[2]	(✓)[1]	✓
scaleable recovery	✗	✗	✗	✗	✗	✓
programming model	MPI RDMA	MPI	MPI	MPI	PGAS/GPI	MPI

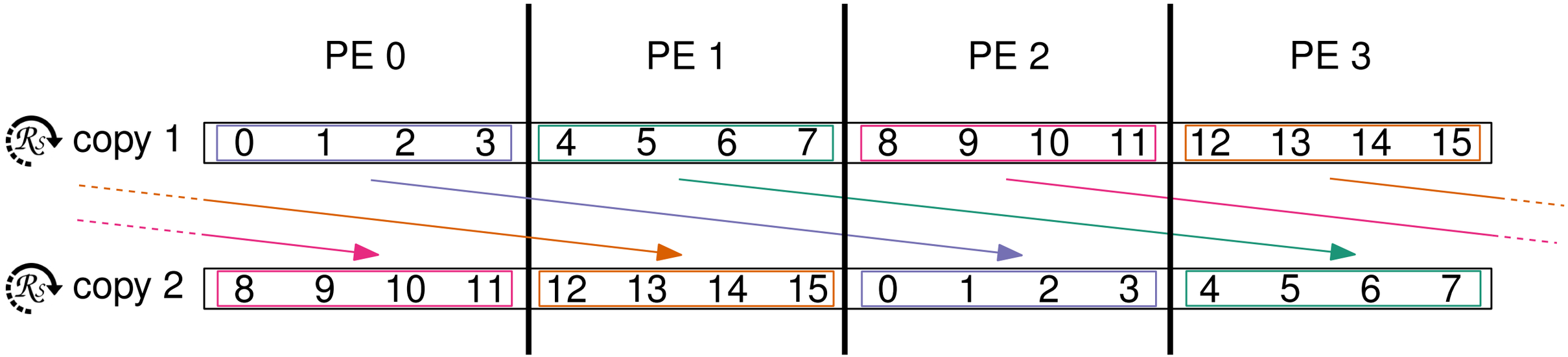
[1] Need for nodes idling until they replace a failing node
[2] Additionally, some nodes used solely to store checkpoints

Basic Data Distribution

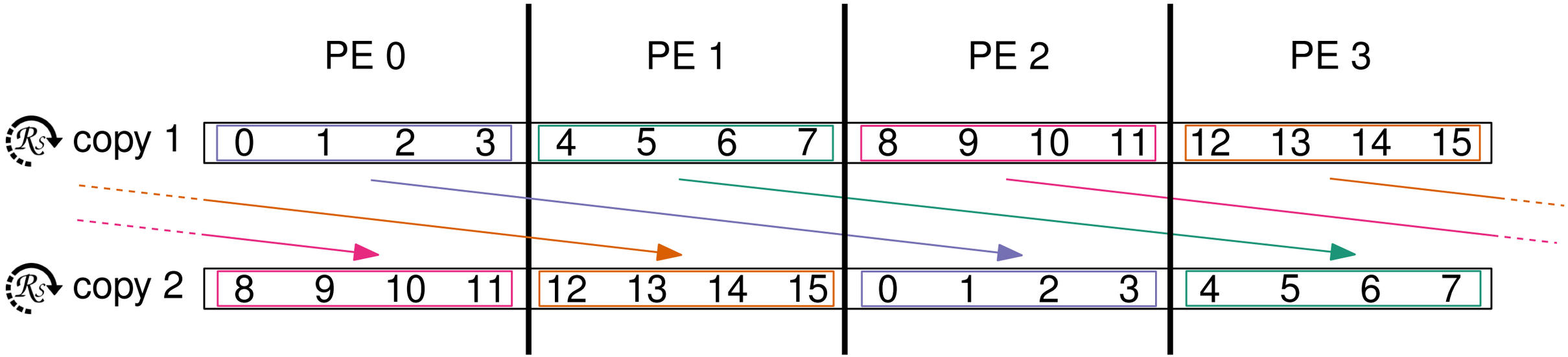


- Data distributed across PEs
- Data divided into blocks
- Blocks addressable via IDs

Basic Data Distribution



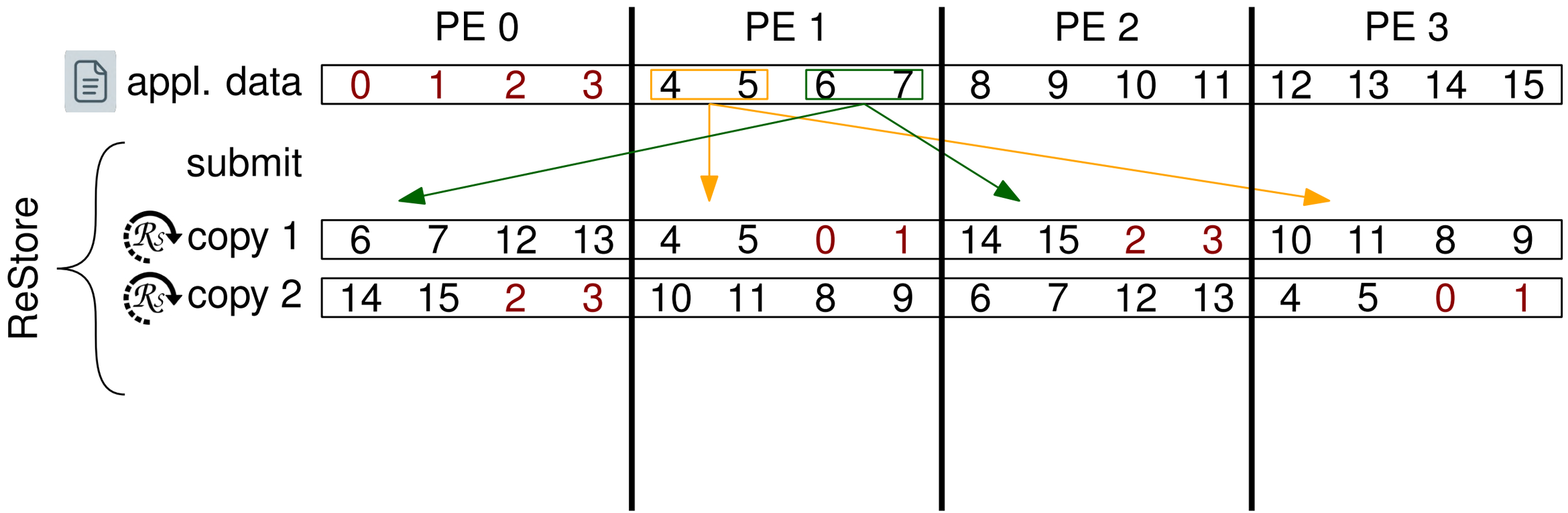
Basic Data Distribution



On recovery: $r = 2$ sending nodes $\rightarrow$ bottleneck

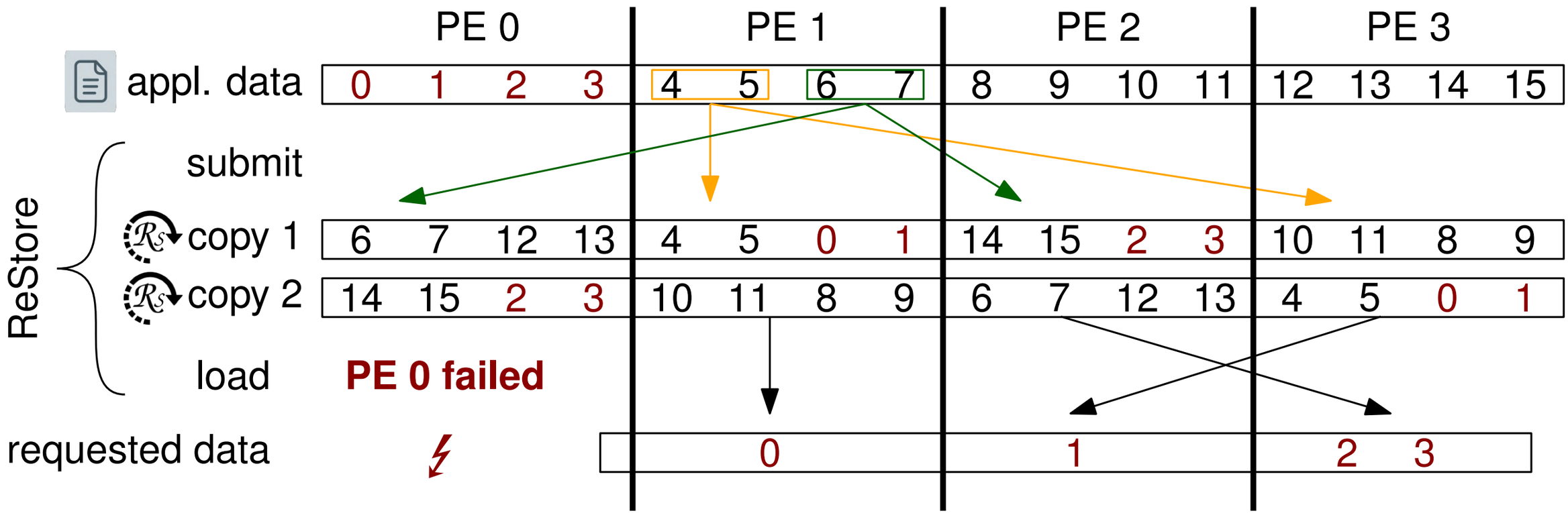
Data Distribution for Faster Recovery

- **Idea:** Break up access pattern using a random permutation for the block IDs
- More PEs serving data after failure
- Too many PEs serving data $\rightarrow$ messages too small
- Empirical optimum: Permute $256\,\mathrm{KiB}$ together



Data Distribution for Faster Recovery

- **Idea:** Break up access pattern using a random permutation for the block IDs
- More PEs serving data after failure
- Too many PEs serving data $\rightarrow$ messages too small
- Empirical optimum: Permute $256\,\mathrm{KiB}$ together



Lukas Hübner, Demian Hespe, Peter Sanders, Alexandros Stamatakis
ReStore: In-Memory REplicated STORagE for Rapid Recovery
HITS • Computational Molecular Evolution
KIT • Institute of Theoretical Informatics

Implementation and Experimental Setup

Implementation

- C++; header-only; modern CMake
- https://github.com/ReStoreCpp/ReStore

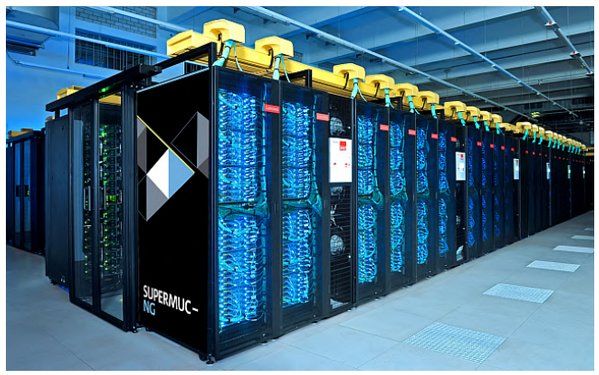
ReStore

Experimental Setup

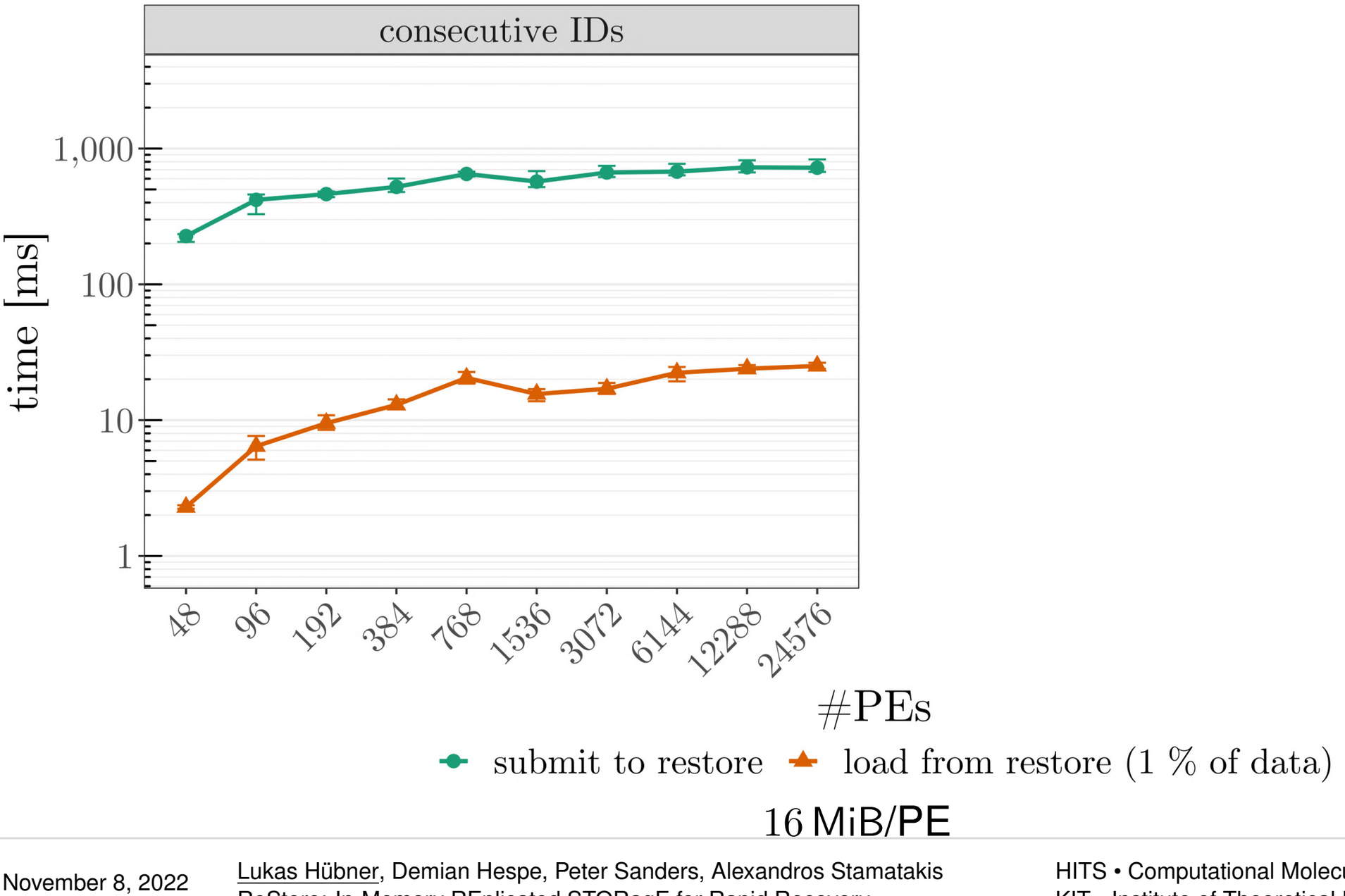
- SuperMUC-NG
- 10 repetitions per experiment
- MPI Implementation: OpenMPI during experiments, ULFM during unit tests

RAxML-NG

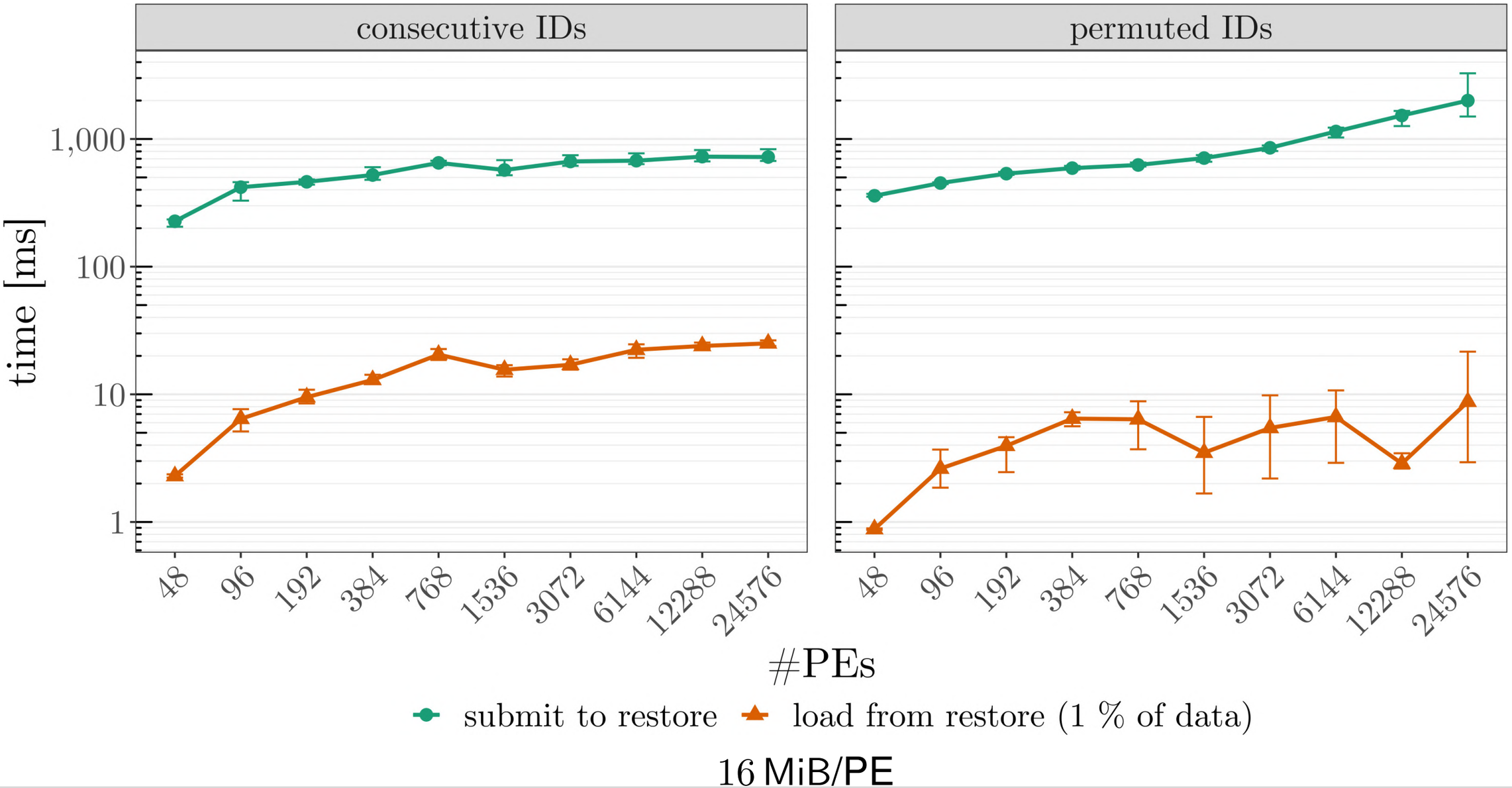
- Existing bioinformatics tool
- Real-world application, cited $50\,000$+ times
- Checkpointing dynamic data part of previous work
- Slow loading of input data from parallel file system $\rightarrow$ ReStore



Evaluating ID Randomization

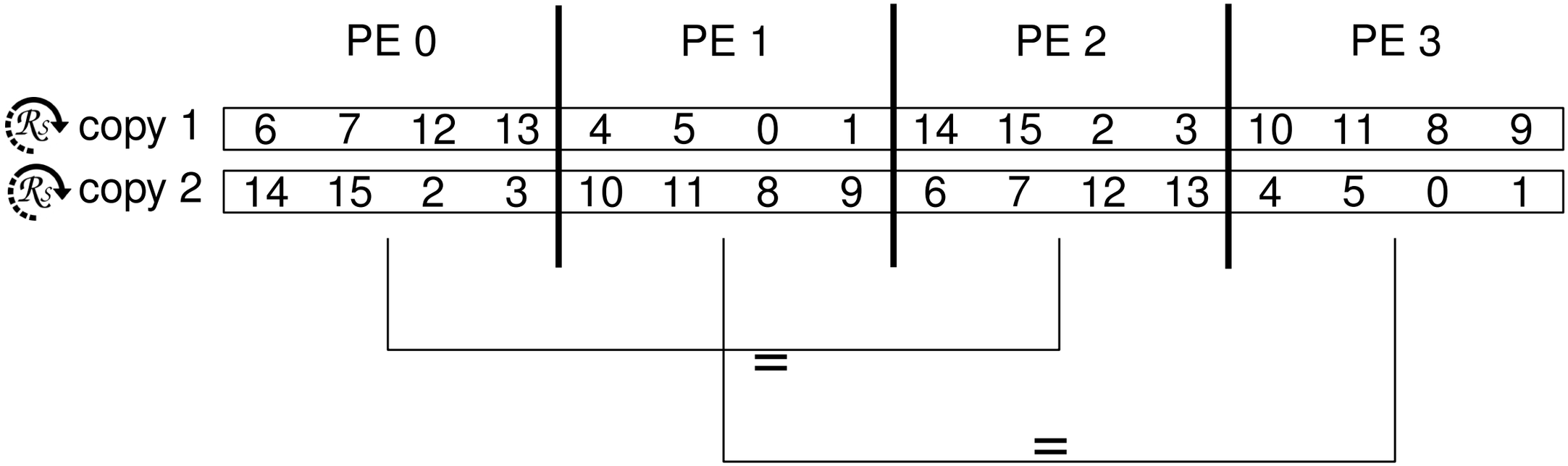


Evaluating ID Randomization



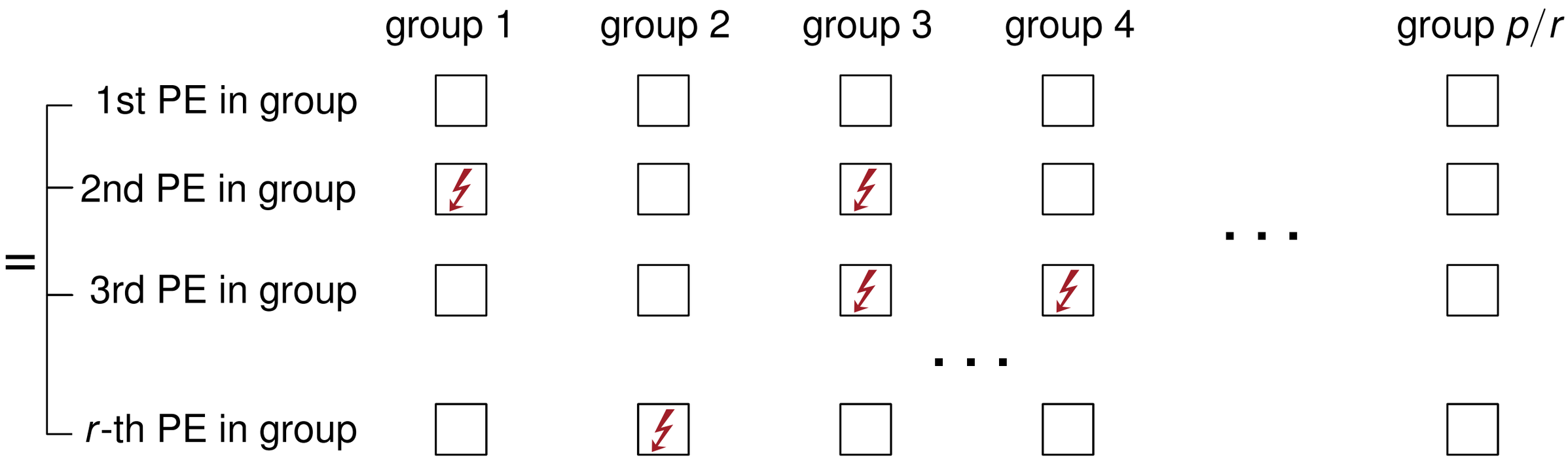
Probability of Irrecoverable Data Loss

Number of replicas r divides number of PEs p
$\rightarrow$ *groups* of PEs storing the same data



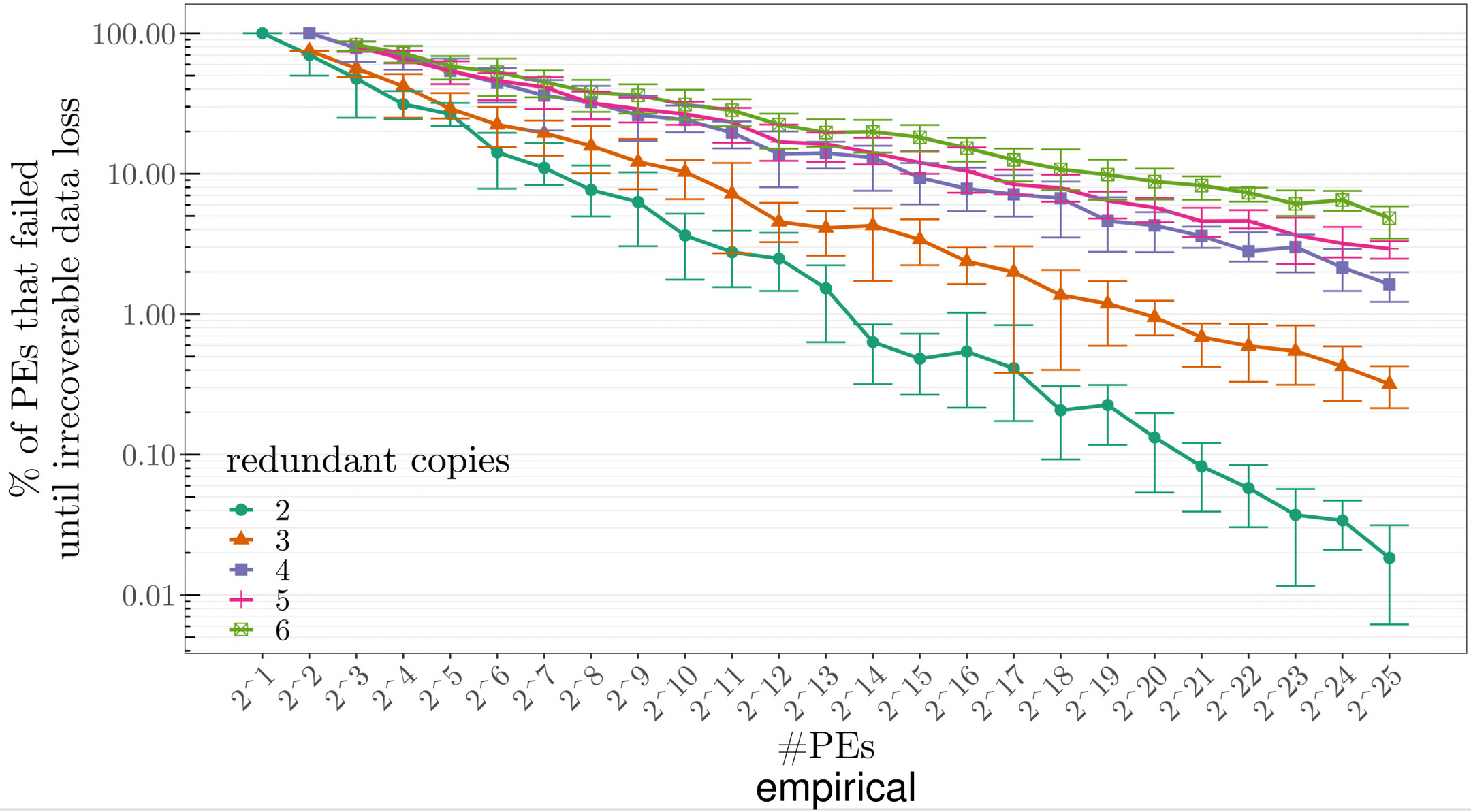
Probability of Irrecoverable Data Loss

Given f failures, what is the probability, that all PEs of any group failed?

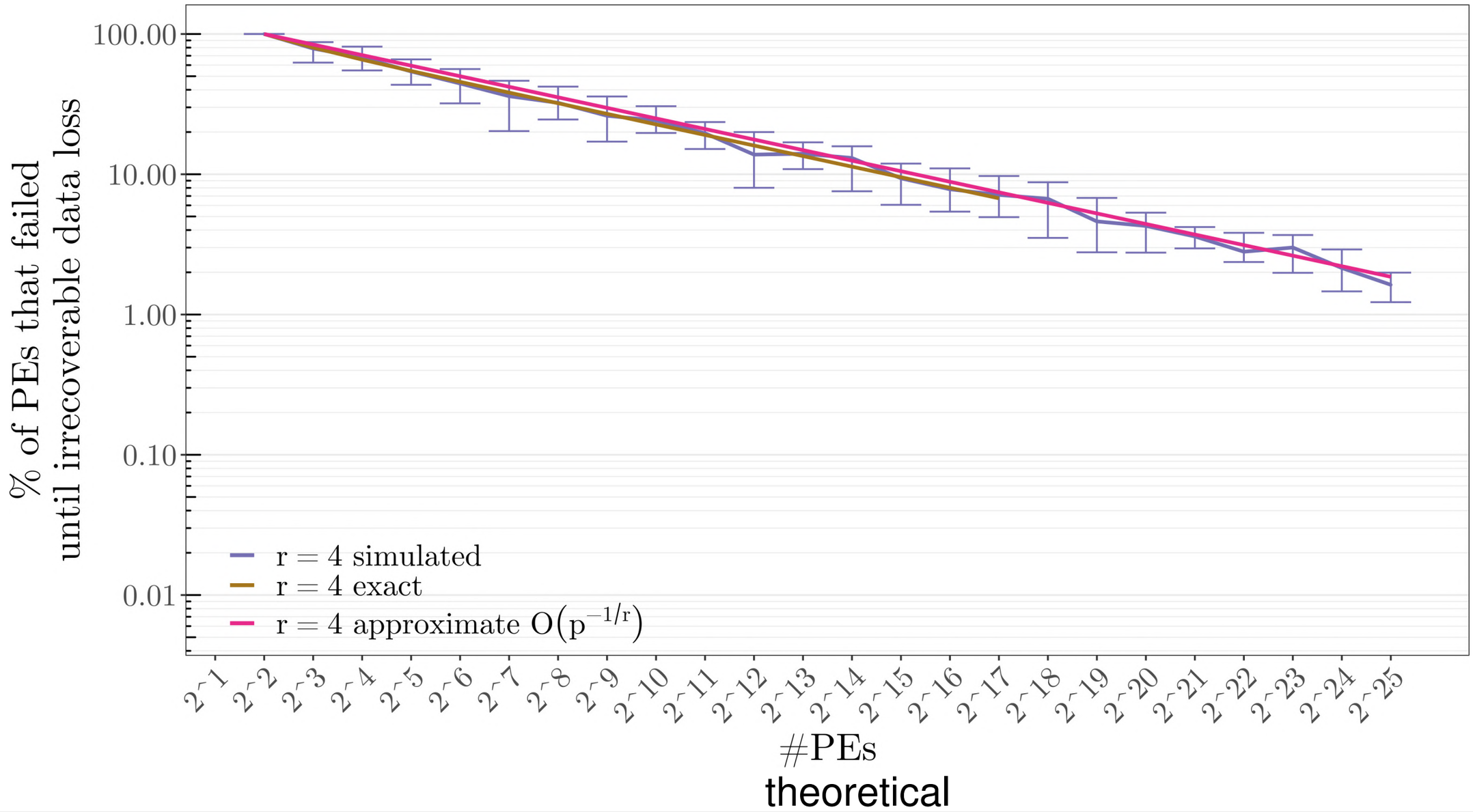


Lukas Hübner, Demian Hespe, Peter Sanders, Alexandros Stamatakis
ReStore: In-Memory REplicated STORagE for Rapid Recovery
HITS • Computational Molecular Evolution
KIT • Institute of Theoretical Informatics

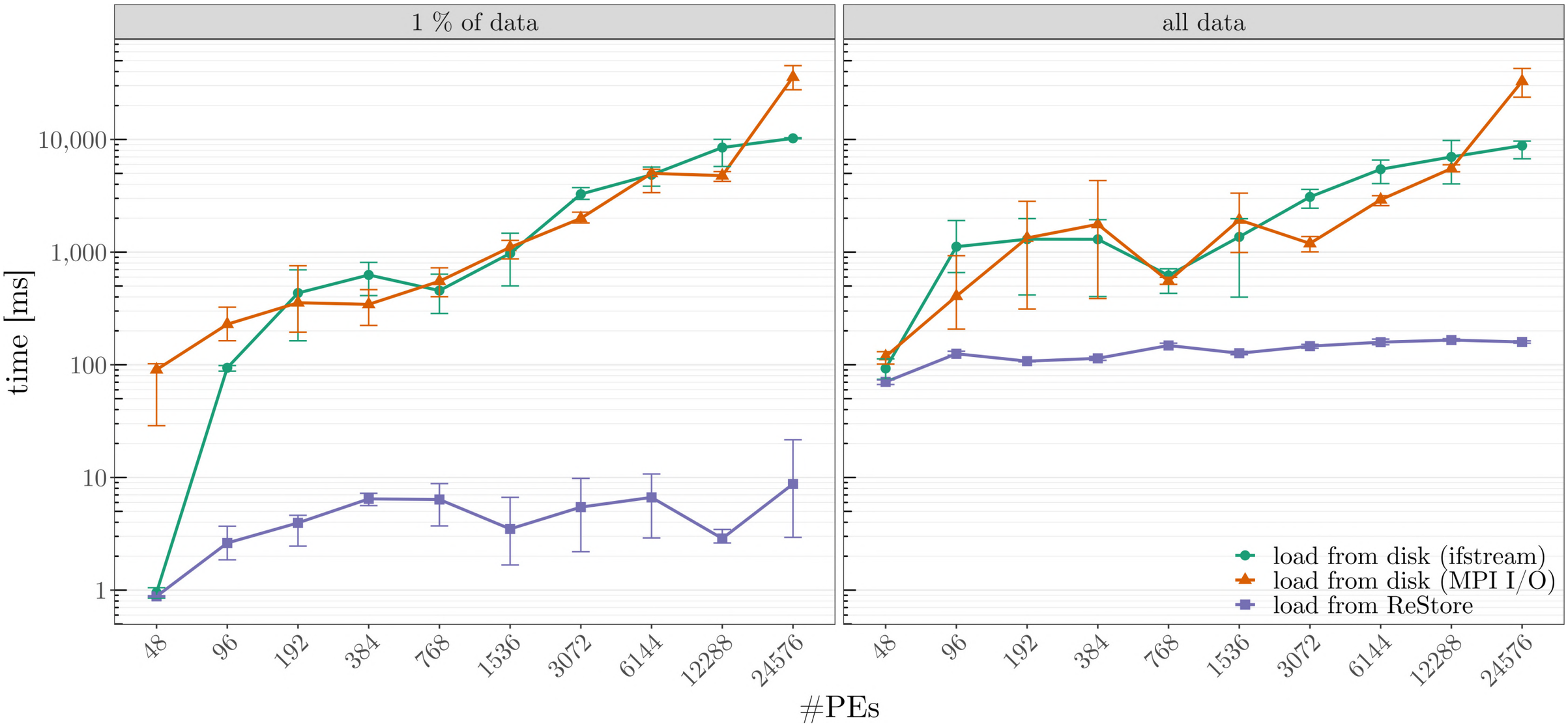
Probability of Irrecoverable Data Loss



Probability of Irrecoverable Data Loss

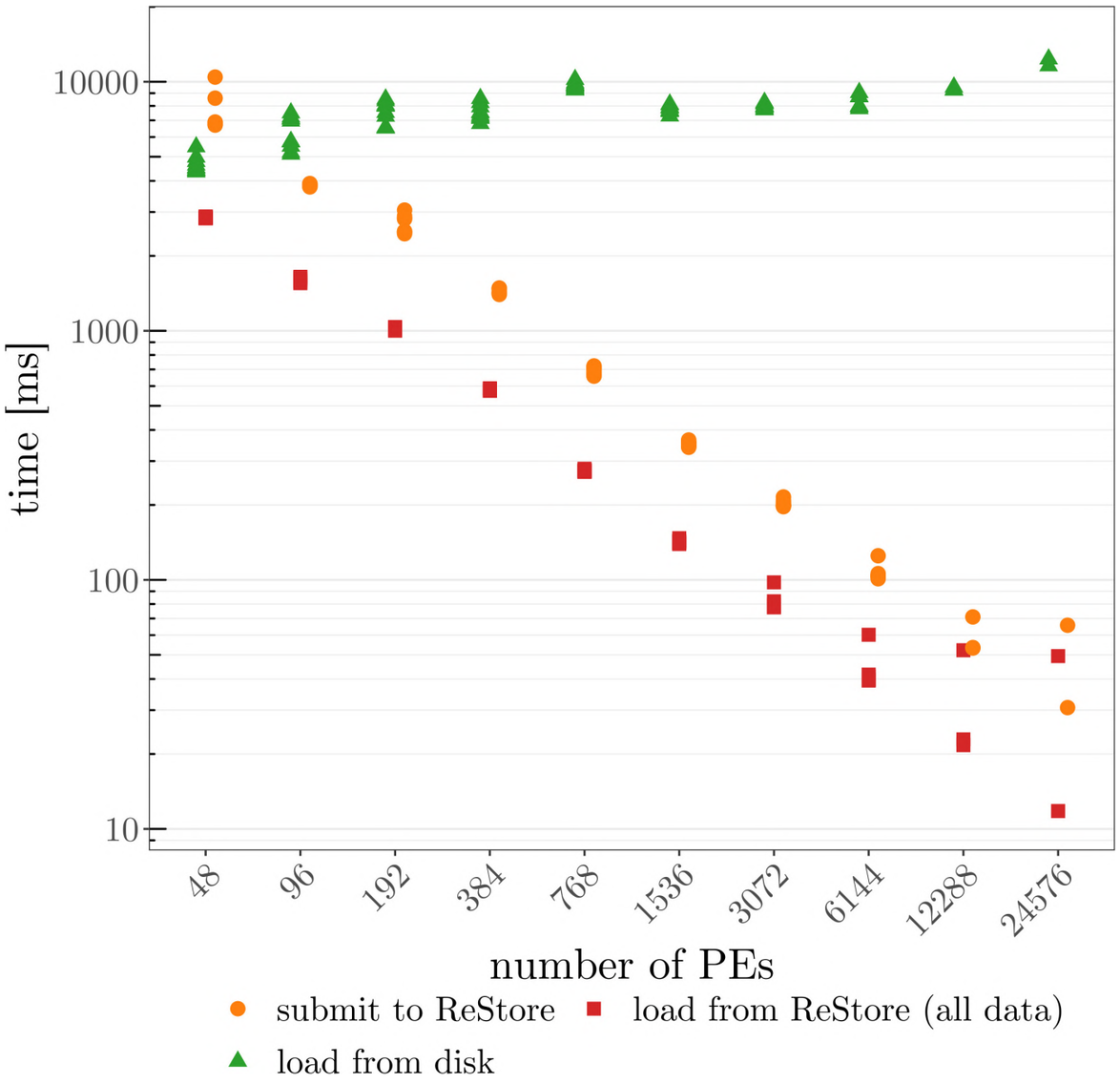


In-Memory vs. Parallel File System



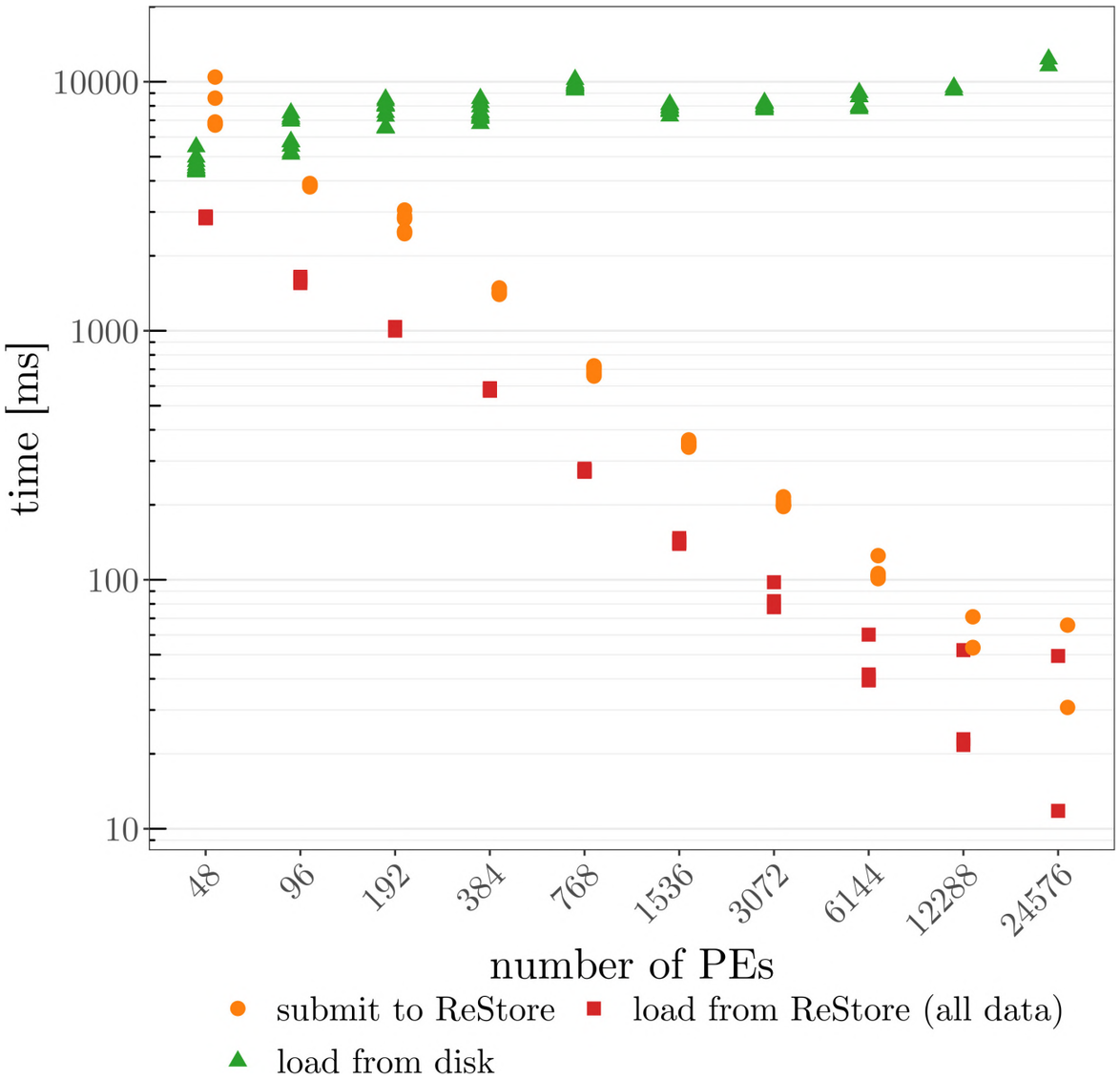
16 MiB data per PE

Overhead of ReStore in RAxML-NG

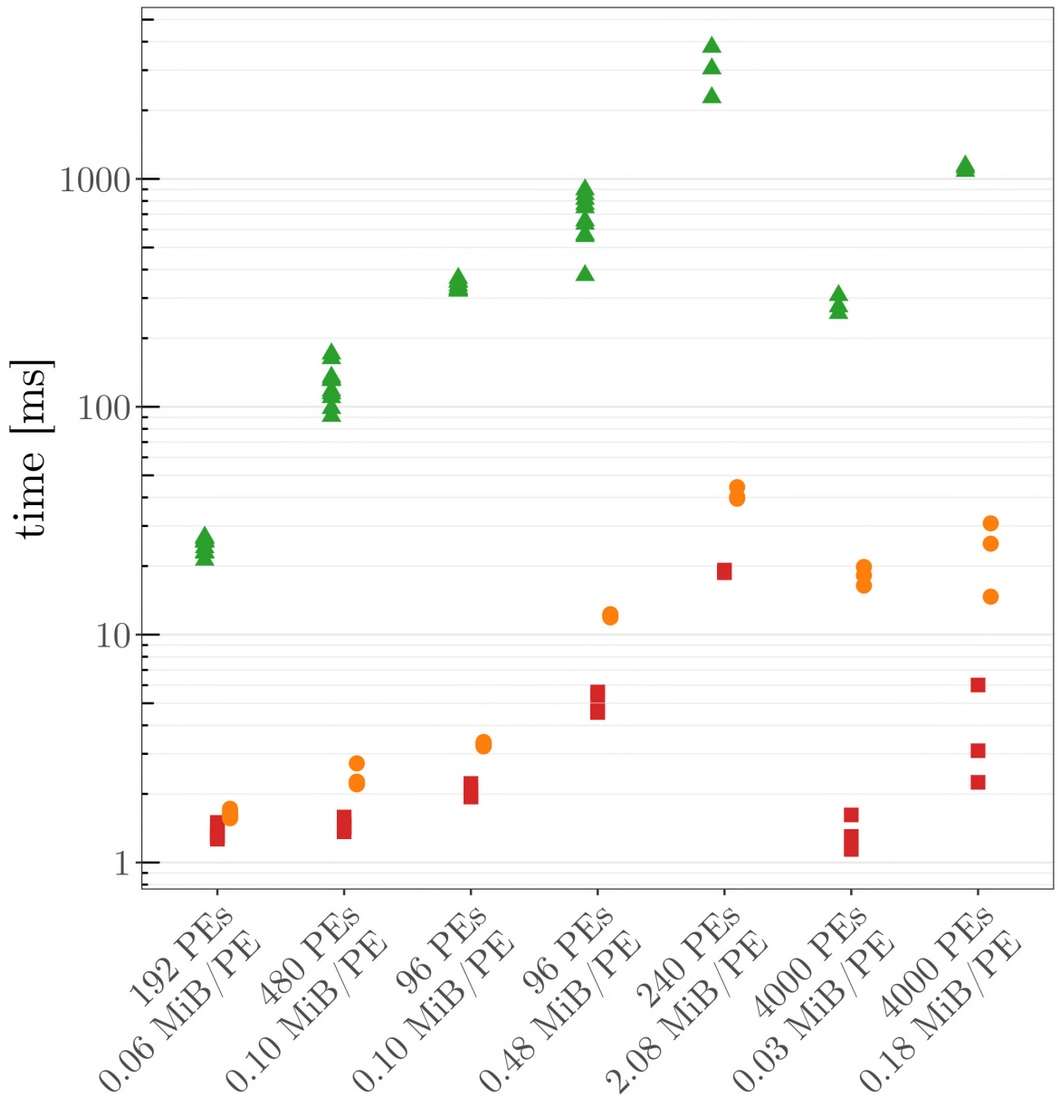


19.1 GiB synthetic dataset

Overhead of ReStore in RAxML-NG

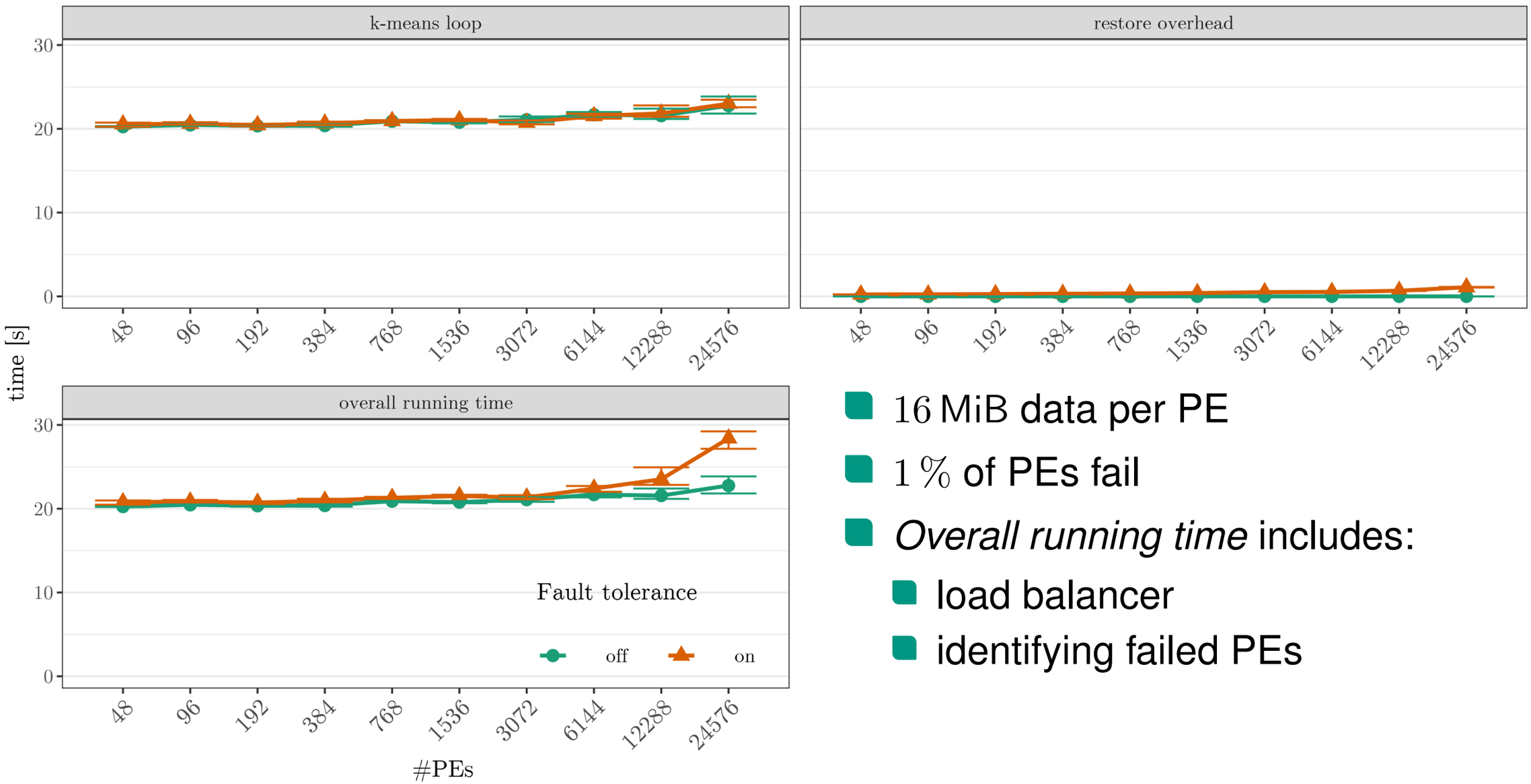


19.1 GiB synthetic dataset



empirical datasets

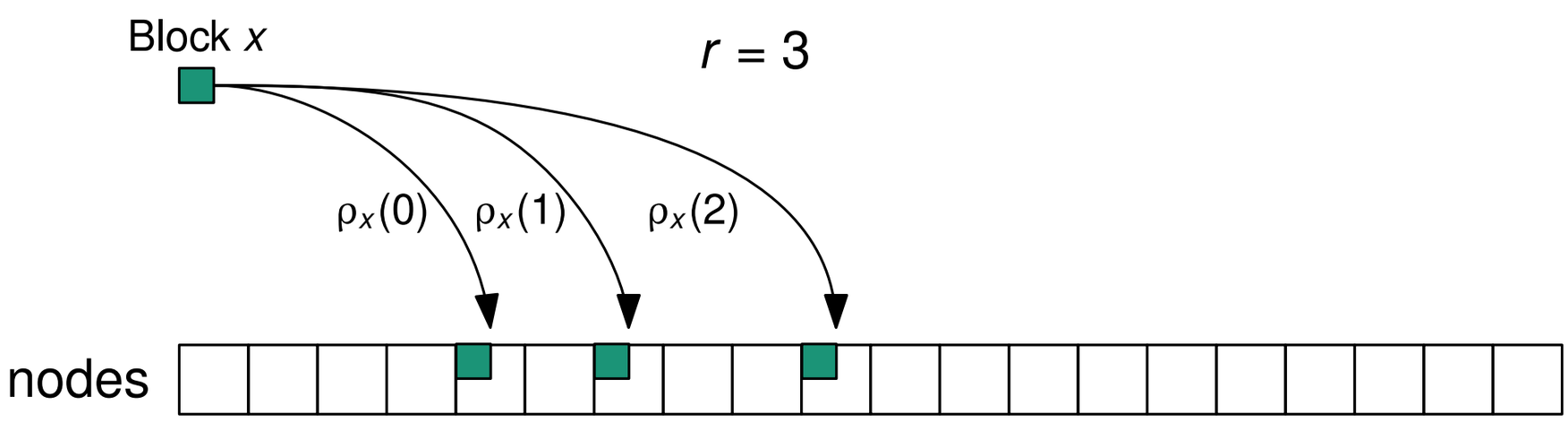
Overhead of ReStore in k-means



- 16 MiB data per PE
- 1 % of PEs fail
- *Overall running time* includes:
  - load balancer
  - identifying failed PEs

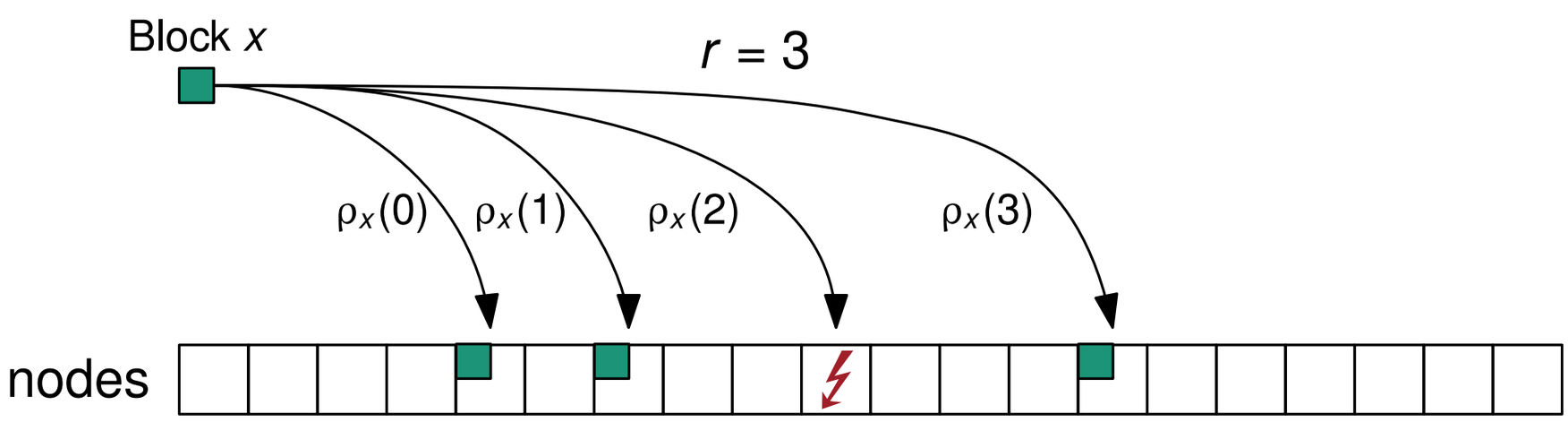
Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$
- Nodes on which this block is stored? $\mathcal{O}(r+f)$ time, $\mathcal{O}(1)$ space



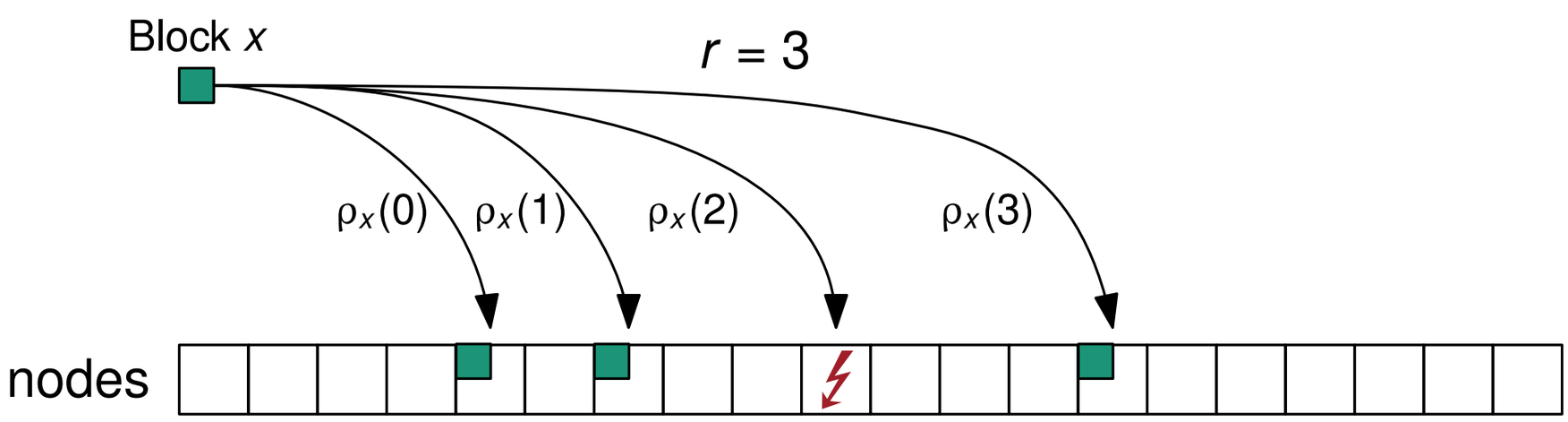
Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$
- Nodes on which this block is stored? $\mathcal{O}(r+f)$ time, $\mathcal{O}(1)$ space



Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$
- Nodes on which this block is stored? $\mathcal{O}(r+f)$ time, $\mathcal{O}(1)$ space



No need to redistribute any block that did not lose a replica!

Conclusion

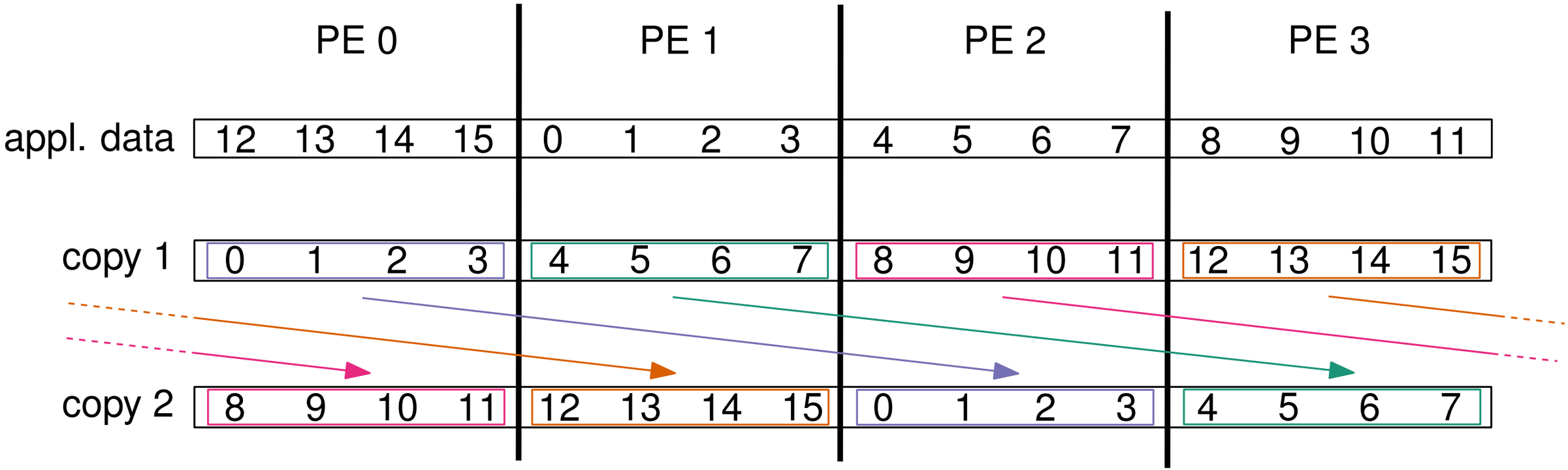
- Permutation-based data distribution enables recovery of lost data in milliseconds on tens of thousands of PEs
- First in-memory library to support shrinking recovery
- RAxML-NG's recovery performance improved by up to two orders of magnitude
- Extension to easily restore lost replicas after a failure
- Provably small probability of data loss

ReStore

https://github.com/ReStoreCpp/ReStore

Thanks for listening :-)

Basic Data Distribution



- Avoid storing the blocks needed after a failure of node *i* on node *i*
- No need to change the distribution of the application data; assigning different IDs when submitting to ReStore is sufficient

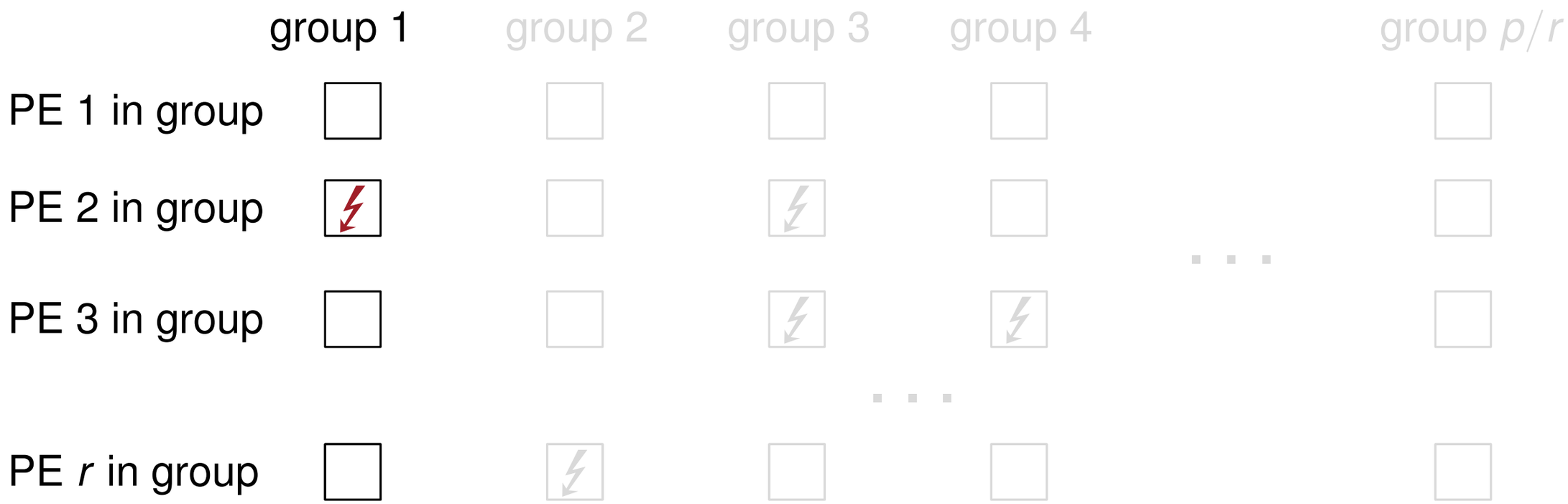
Implementation and Experimental Setup

Experimental Setup

- We benchmark on the SuperMUC-NG
- Two Intel Skylake Xeon 8174 processors with 24 cores each per node
- 96 GiB of RAM per node
- Omnipath interconnect with 100 Gbit s^{-1}
- OpenMPI as MPI implementation
- 10 repetitions per experiment

Probability of Irrecoverable Data Loss

Given f failures, what is the probability, that all copies of group 1 failed?

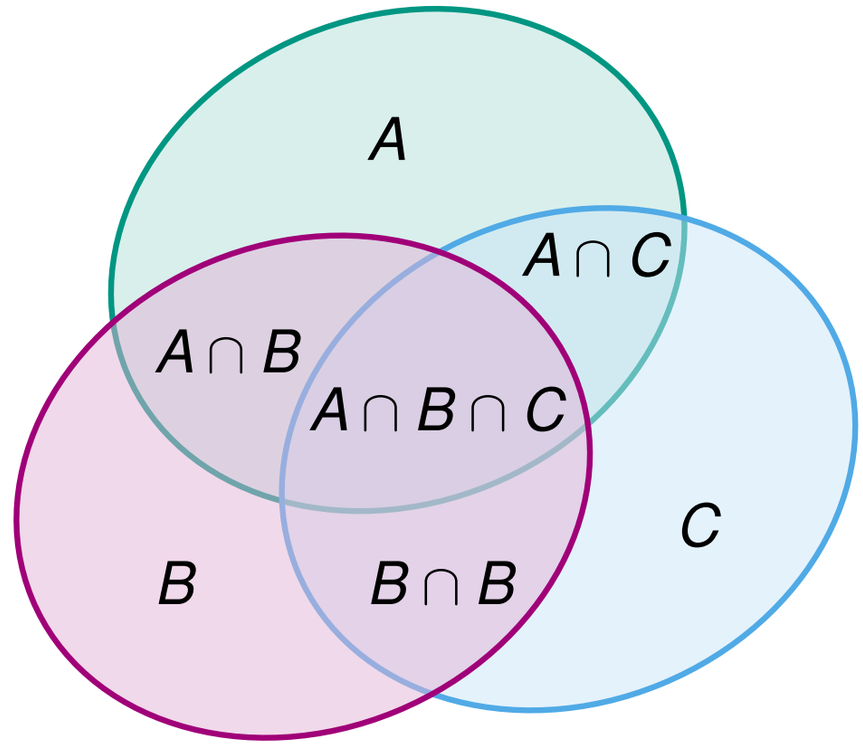


- Number of possibilities to draw f nodes from p nodes: $\binom{p}{f}$
- Number of possibilities to draw all r copies of group 1 plus $f - r$ other nodes: $\binom{p-r}{f-r}$
- $P(\text{All nodes of group 1 failed}) = \binom{p-r}{f-r} / \binom{p}{f}$

Probability of Irrecoverable Data Loss

Inclusion-exclusion principle

$$
\begin{aligned}
|A \cup B \cup C| = &|A| + |B| + |C| \\
&- |A \cap B| - |A \cap C| - |B \cap C| \\
&+ |A \cap B \cap C|
\end{aligned}
$$



Probability of Irrecoverable Data Loss

- Given f, there are $\binom{p-r}{f-r}$ configurations of failed nodes which lead to data loss
- Summing up over all groups would count certain states twice, trice, ...
- E.g., states in which *all* nodes of group 1 and group 2 failed would be counted twice

$$P_{\text{IDL}}^{\leq}(f) = \sum_{j=1}^{g} (-1)^{j+1} \binom{g}{j} \frac{\binom{p-jr}{f-jr}}{\binom{p}{f}}$$

Probability of Irrecoverable Data Loss

- Given f, there are $\binom{p-r}{f-r}$ configurations of failed nodes which lead to data loss
- Summing up over all groups would count certain states twice, trice, . . .
- E.g., states in which *all* nodes of group 1 and group 2 failed would be counted twice

probability of irrecoverable data loss at failure f or any failure before $\longrightarrow$

$$P^{\leq}_{\text{IDL}}(f) = \sum_{j=1}^{g} (-1)^{j+1} \binom{g}{j} \frac{\binom{p-jr}{f-jr}}{\binom{p}{f}}$$

Probability of Irrecoverable Data Loss

- Given f, there are $\binom{p-r}{f-r}$ configurations of failed nodes which lead to data loss
- Summing up over all groups would count certain states twice, trice, ...
- E.g., states in which *all* nodes of group 1 and group 2 failed would be counted twice

probability of irrecoverable data loss at failure f or any failure before

$$P^{\leq}_{\mathrm{IDL}}(f) = \sum_{j=1}^{g} (-1)^{j+1} \binom{g}{j} \frac{\binom{p-jr}{f-jr}}{\binom{p}{f}}$$

inclusion-exclusion principle

all combinations of $1, \ldots, j, \ldots, g$ groups in which all nodes failed

Probability of Irrecoverable Data Loss

- Given f, there are $\binom{p-r}{f-r}$ configurations of failed nodes which lead to data loss
- Summing up over all groups would count certain states twice, trice, . . .
- E.g., states in which *all* nodes of group 1 and group 2 failed would be counted twice

probability of irrecoverable data loss at failure f or any failure before

$$P_{\text{IDL}}^{\leq}(f) = \sum_{j=1}^{g} (-1)^{j+1} \binom{g}{j} \frac{\binom{p-jr}{f-jr}}{\binom{p}{f}}$$

inclusion-exclusion principle

all combinations of $1, \ldots, j, \ldots, g$ groups in which all nodes failed

number of configurations