IXPUG
INTEL EXTREME PERFORMANCE USERS GROUP
CONFERENCE

Recovery of Distributed Iterative Solvers for Linear Systems Using Non-Volatile RAM

Yehonatan Fridman, Yaniv Snir, Harel Levin, Danny Hendler, Hagit Attiya, and Gal Oren*

*galoren@cs.technion.ac.il

הטכניון מכון טכנולוגי לישראל TECHNION Israel Institute of Technology

שורק

הוועדה לאנרגיה אטומית Israel Atomic Energy Commission

Acknowledgements

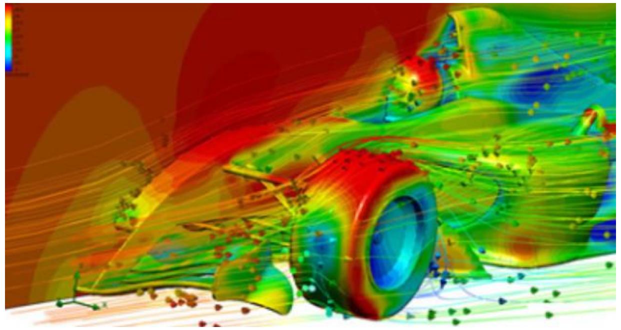
- This work was supported by:
  - Pazy grant 226/20,
  - ISF grant 1425/22,
  - Lynn and William Frankel Center for Computer Science,
  - Intel Corporation (oneAPI Center of Excellence program),
- Computational support was provided by the NegevHPC project.

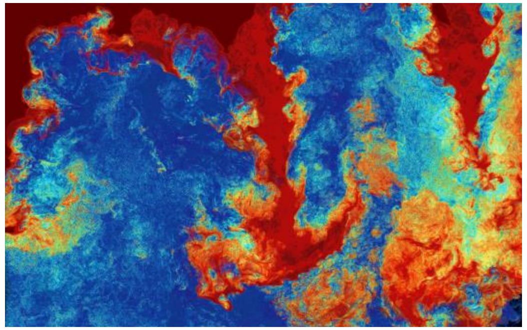
הטכניון
מכון טכנולוגי
לישראל
TECHNION
Israel Institute
of Technology

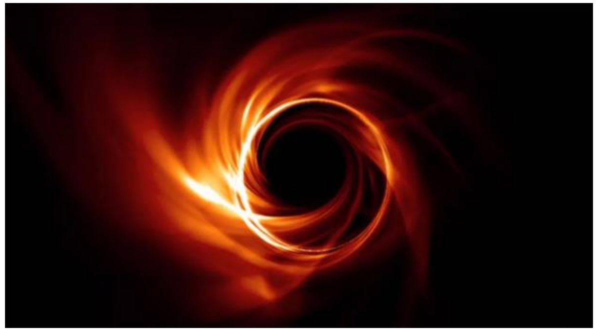
הוועדה לאנרגיה אטומית
Israel Atomic Energy Commission

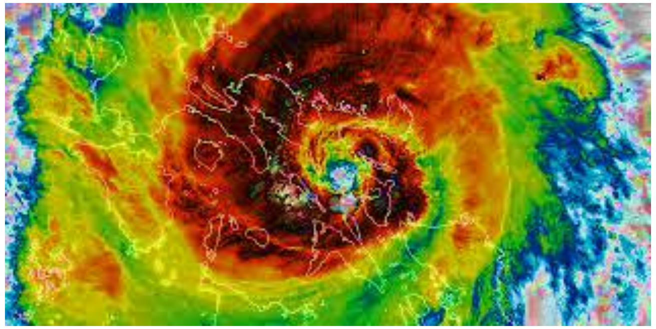
Exascale Challenges

HPC systems are a critical resource for scientific research and advanced industries:







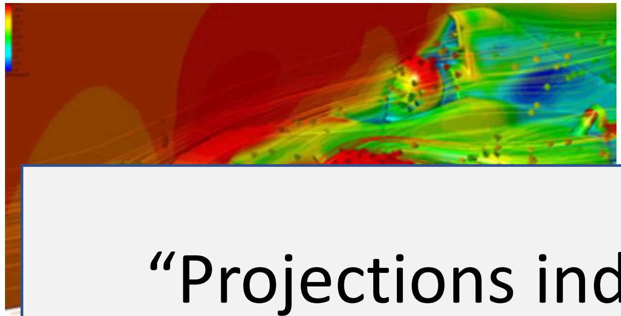


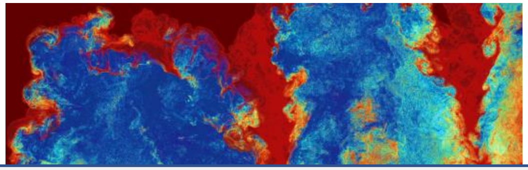
Exascale supercomputers → numerous compute nodes → frequent faults and crashes

The Aurora Exascale supercomputer

Exascale Challenges

HPC systems are a critical resource for scientific research and advanced industries:



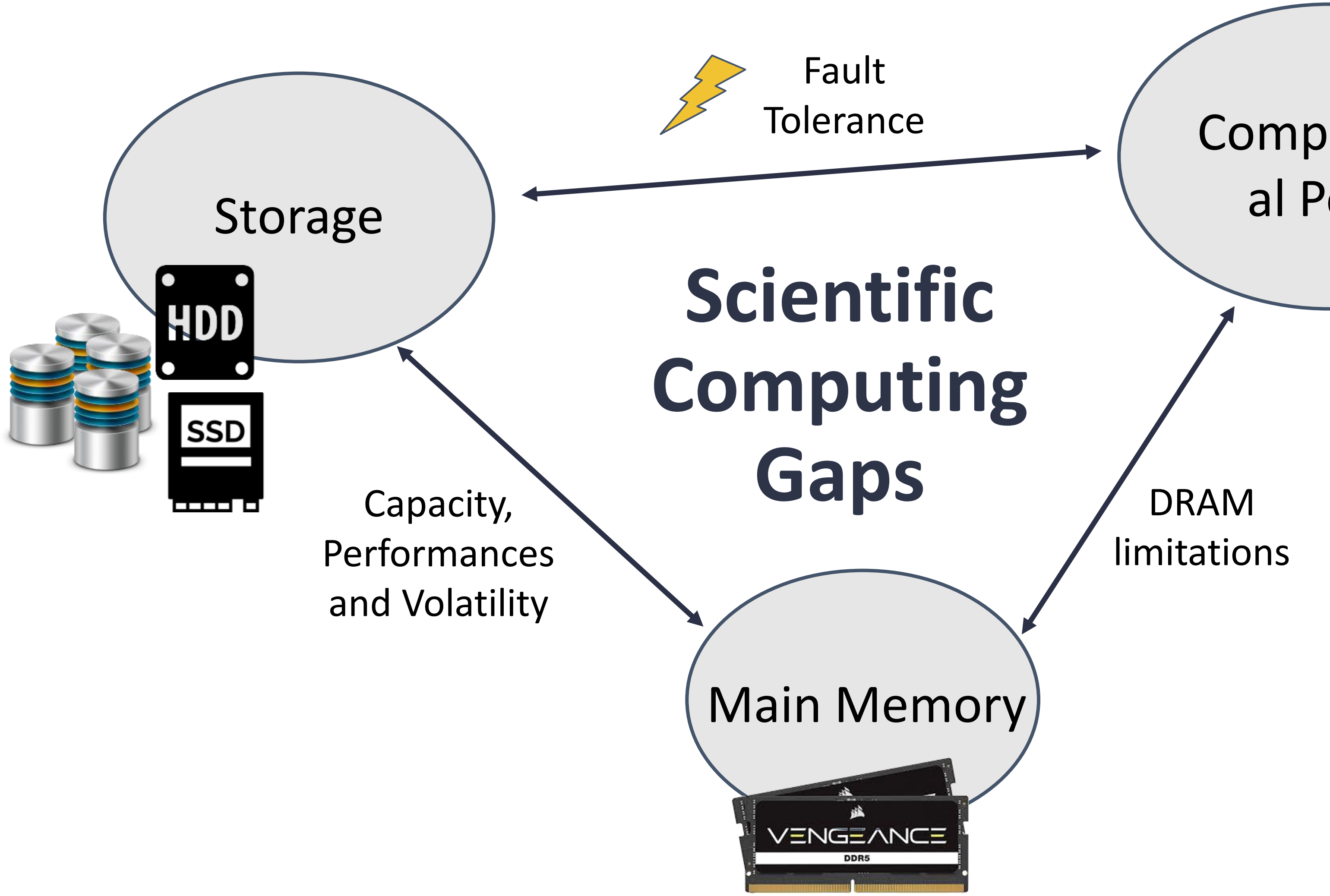


"Projections indicate that the next generation **Exascale** machines will experience **failures** up to **several times an hour**. "

Dauwe, Daniel, et al. "A performance and energy comparison of fault tolerance techniques for exascale computing systems." *2016 IEEE International Conference on Computer and Information Technology (CIT)*. IEEE, 2016.

The Aurora Exascale supercomputer

Scientific Computing Gaps



Storage

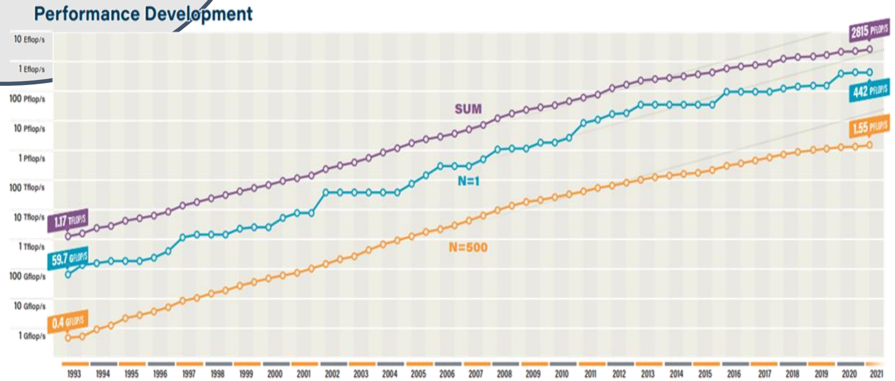
Computational Power

Main Memory

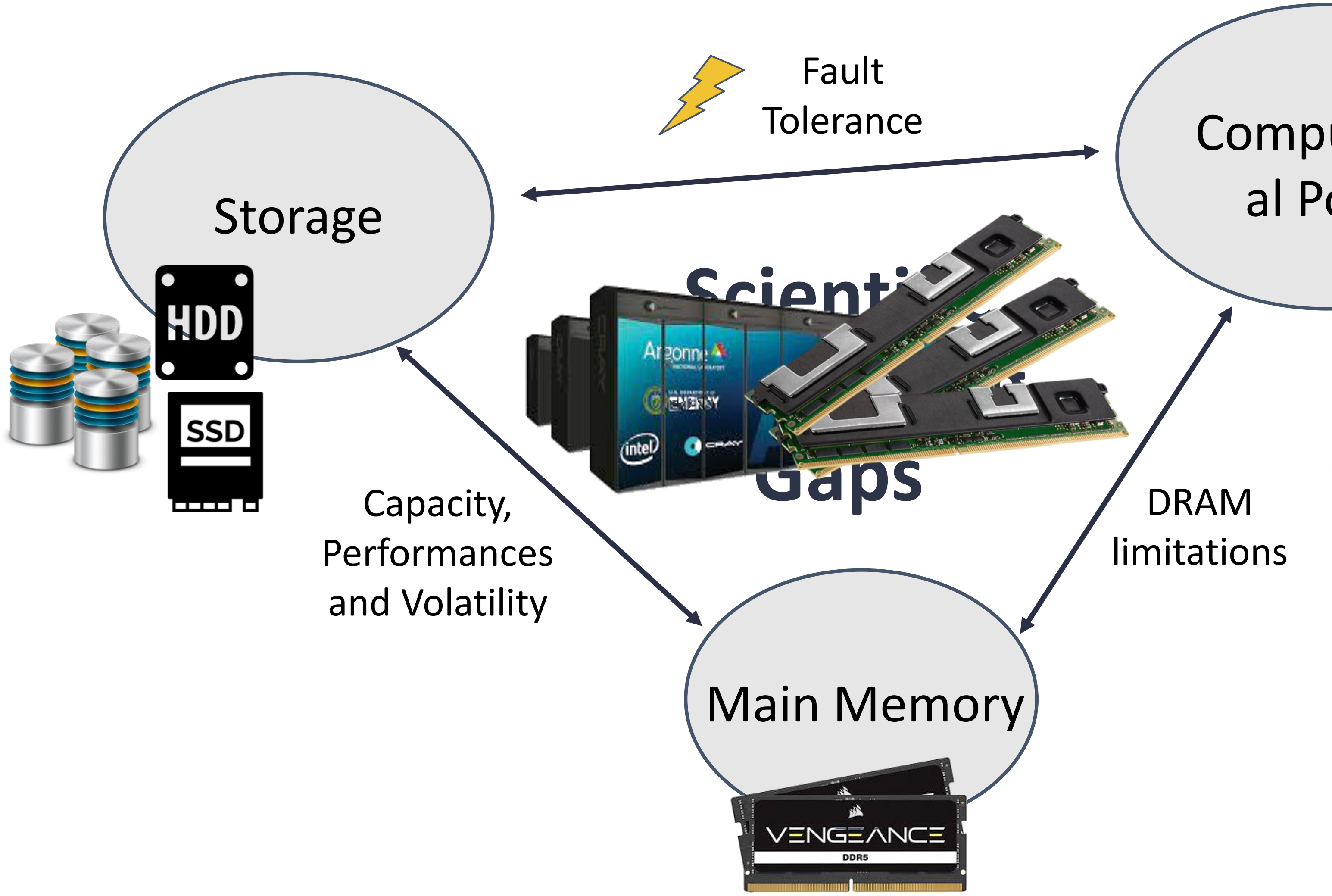
Fault Tolerance

Capacity, Performances and Volatility

DRAM limitations



Computational power of the **top 500** supercomputers for each year, SC, June 2021



Storage

Fault
Tolerance

Computation
al Power

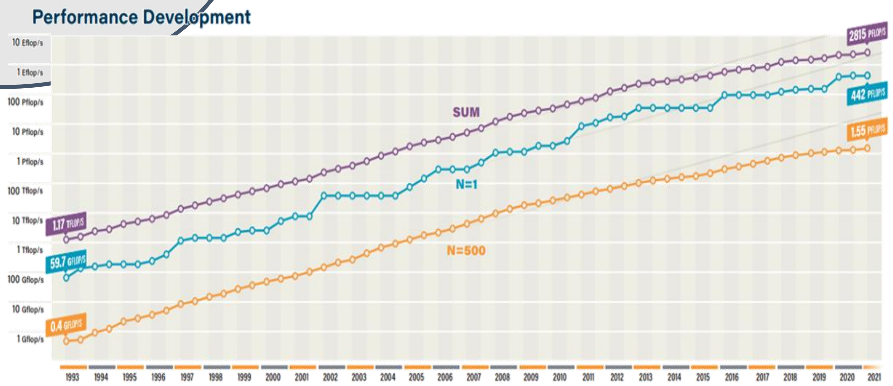
Scienti

Gaps

Capacity,
Performances
and Volatility

DRAM
limitations

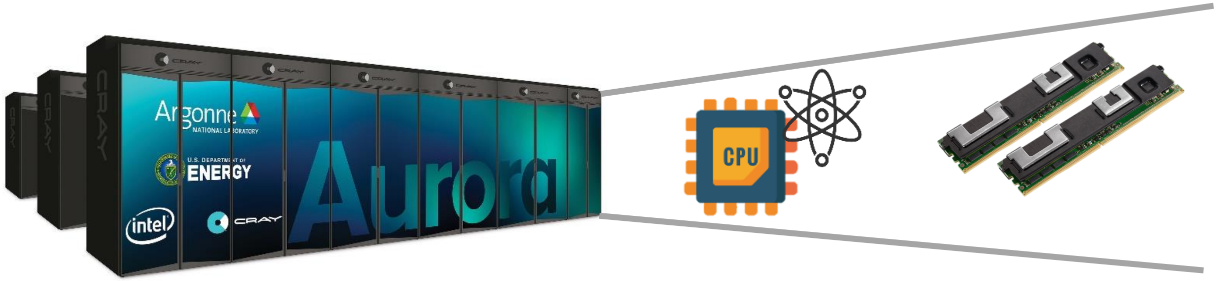
Main Memory



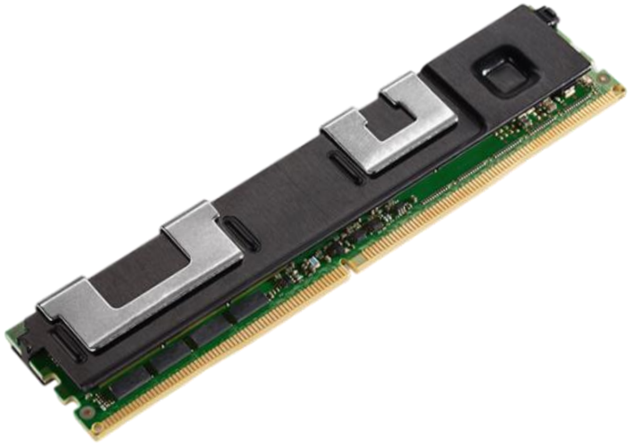
Computational power of the **top 500** supercomputers for each year, SC, June 2021

Non-Volatile RAM (**NVRAM**) in HPC

- **NVRAM** have emerged as compelling storage technologies to augment traditional DRAM with DRAM-like performance.
- Modern designs of supercomputers will integrate **NVRAM**.
- NVRAM can be exploited to increase scientific computations’ fault tolerance by persisting the state of computations and reading it back for recovery.



The Aurora supercomputer is planned to integrate Intel Optane™ DCPMM

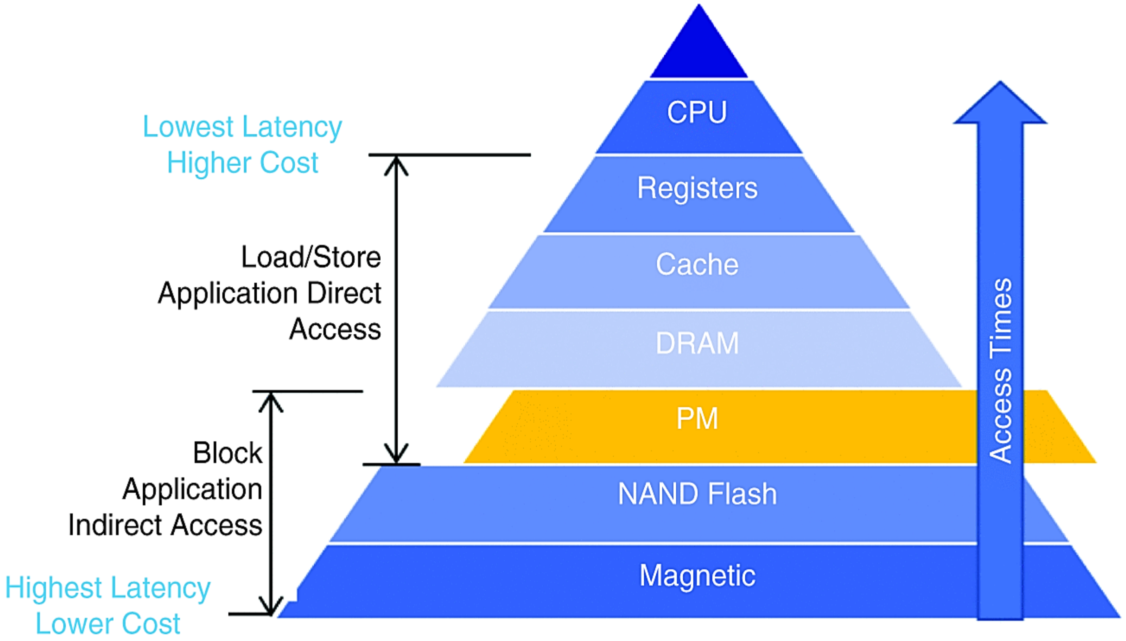


Intel Optane™ DCPMM

Non-Volatile RAM (**NVRAM**) in HPC

NVRAM (Persistent Memory) Characteristics

- Persistent, large capacity, directly connected to the CPU and byte addressable.
- **Direct access (DAX)** allows applications to directly access the persistent media from the CPU.
- 128GB, 256GB, 512GB per **NVDIMM**, and up to 6TB per two-socket system.
- **DDR4** Socket Compatible.
- Cheaper than **DRAM**.
- Relatively small bandwidth on write operations (~8% of **DRAM**).



Persistent Memory in the memory and storage hierarchy

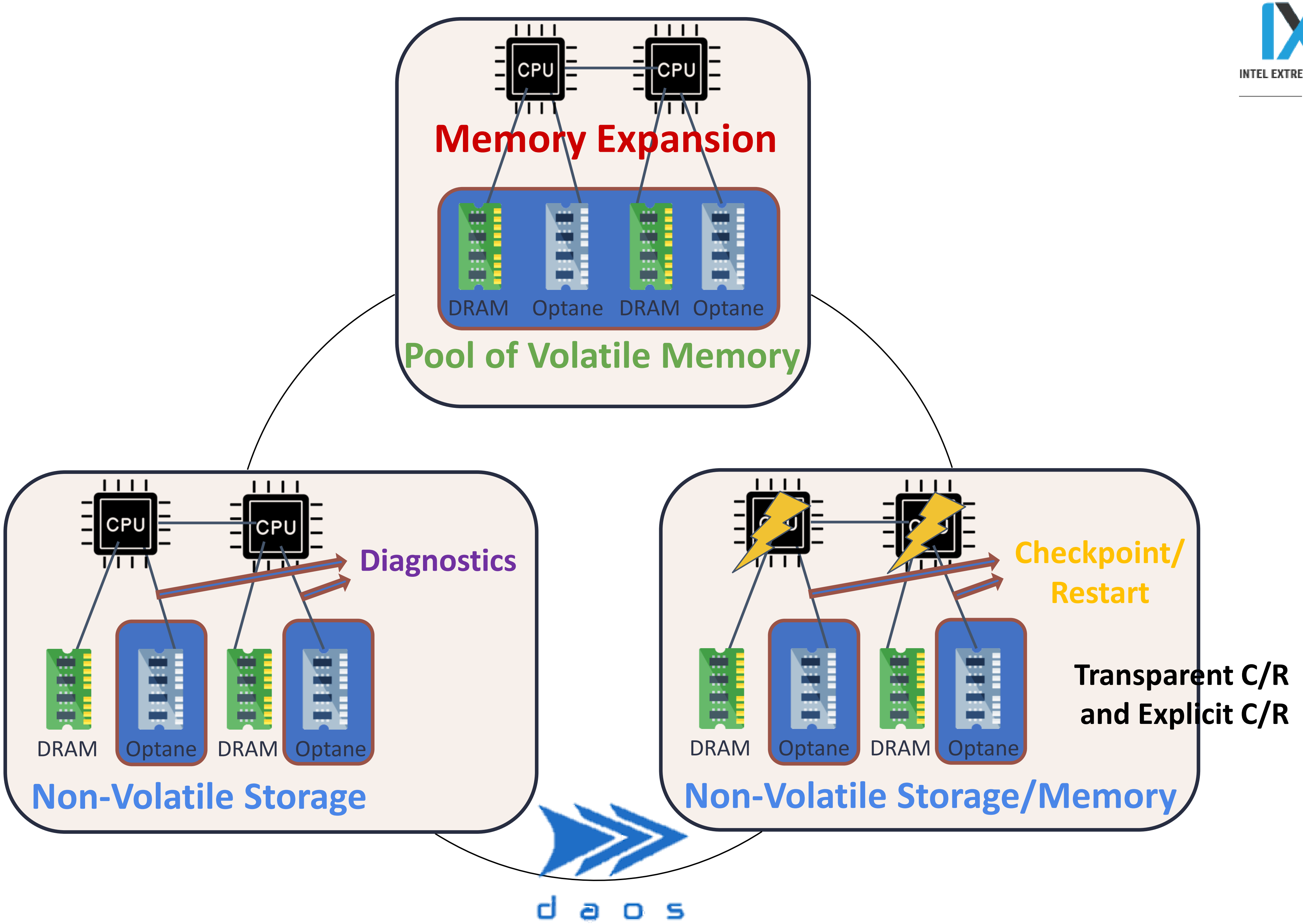
Non-Volatile RAM (**NVRAM**) in HPC

Preliminary Results of NVRAM in HPC

In our previous work “*Assessing the Use Cases of Persistent Memory in High-Performance Scientific Computing*”, we show the capabilities of **Intel Optane™ DCPMM** to **improve** scientific computations’ **performance** and **fault tolerance** by:

1) serving as an **expansion to the DRAM** (DCPMM in memory mode).
2) replacing standard storage devices as a **fast local storage area** for diagnostics and C/R (DCPMM in app-direct mode).
   - We utilized NVRAM both in **transparent C/R** (e.g., DMTCP) and **explicit C/R** (e.g., the SCR library).

Fridman Yehonatan, et al. "Assessing the Use Cases of Persistent Memory in High-Performance Scientific Computing." *2021 IEEE/ACM 11th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS).* IEEE, 2021.



Memory Expansion

Pool of Volatile Memory

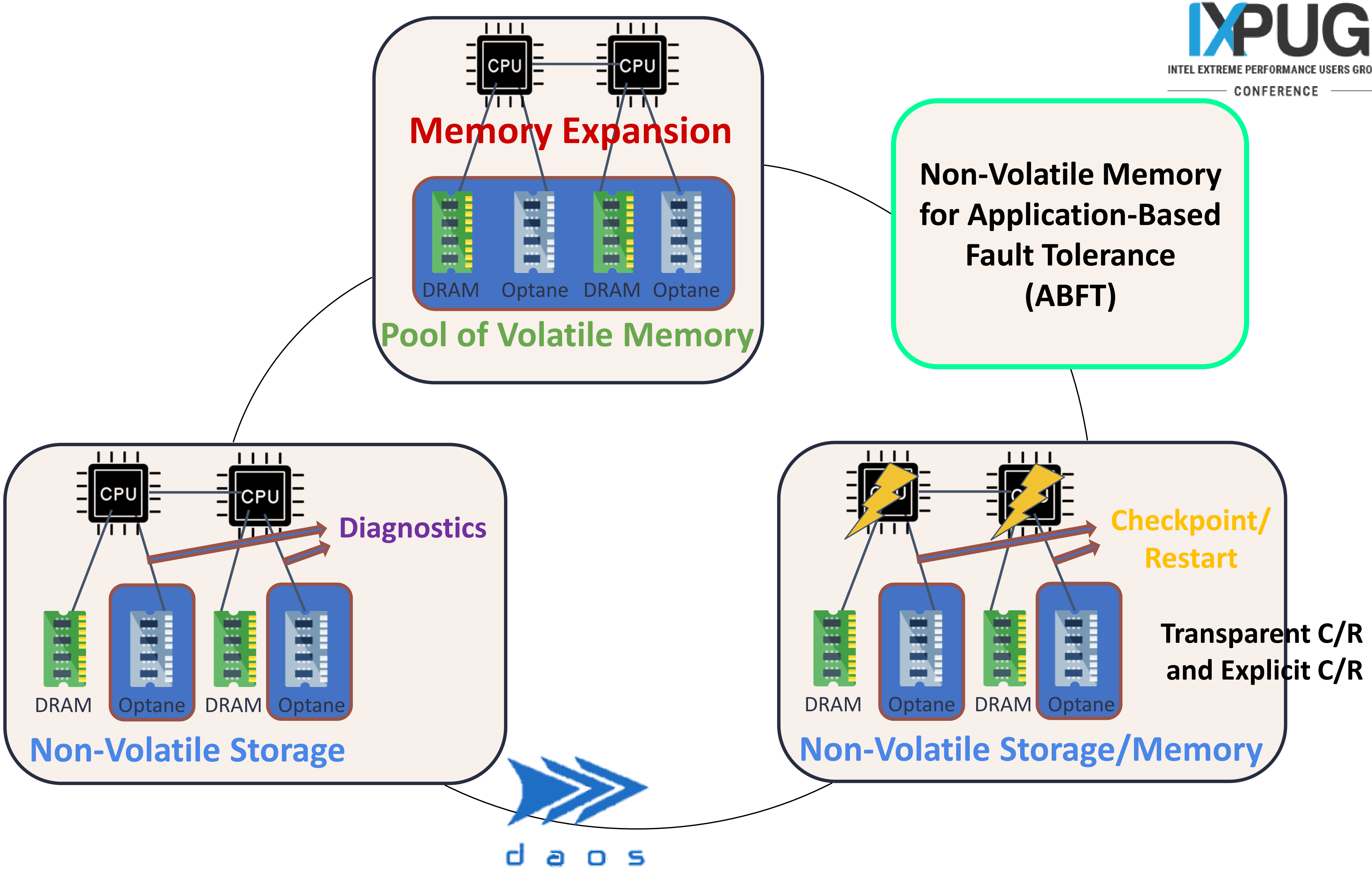
Diagnostics

Non-Volatile Storage

Checkpoint/ Restart

Transparent C/R and Explicit C/R

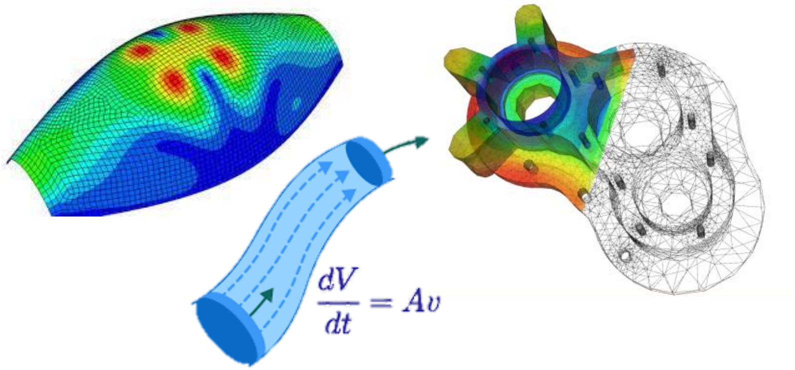
Non-Volatile Storage/Memory



Non-Volatile Memory for Application-Based Fault Tolerance (ABFT)

Distributed Linear Iterative Solvers

- Mathematical solvers account for 50-90% of scientific applications operations.
- Linear iterative solvers are main building blocks in scientific computations.



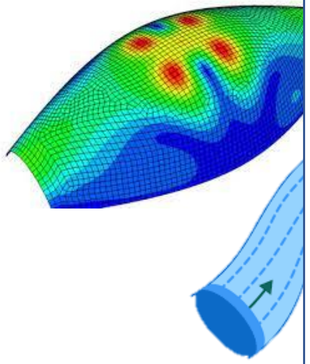
$$Ax = b$$

- Conjugate Gradient (CG), GMRES, Jacobi, SOR and more…
- The Trilinos Project.

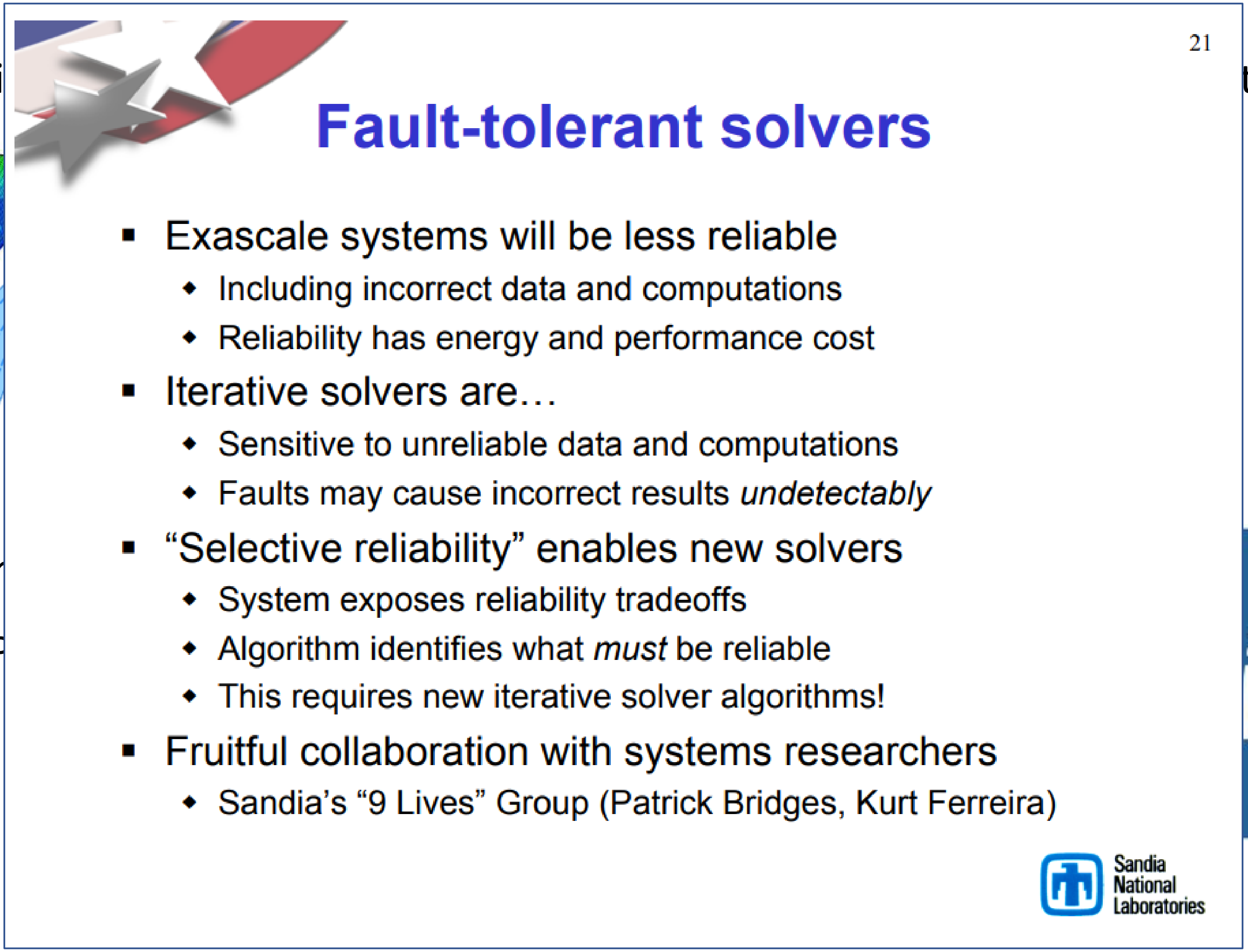
TRILINOS

Distributed Linear Iterative Solvers

- Distributed li tions.



- Conjugate Gr
- The Trilinos F



21

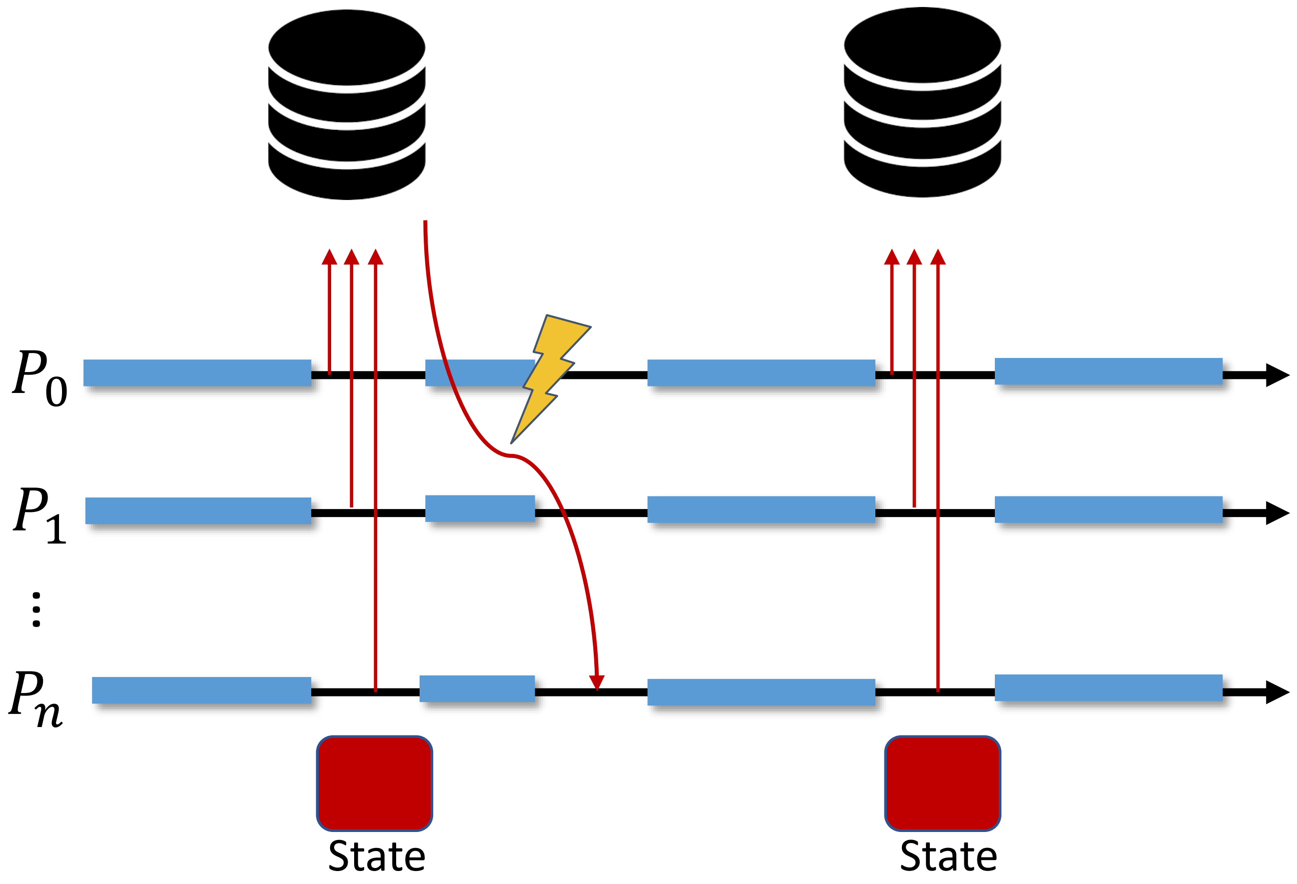
Fault-tolerant solvers

- Exascale systems will be less reliable
  - Including incorrect data and computations
  - Reliability has energy and performance cost
- Iterative solvers are…
  - Sensitive to unreliable data and computations
  - Faults may cause incorrect results *undetectably*
- “Selective reliability” enables new solvers
  - System exposes reliability tradeoffs
  - Algorithm identifies what *must* be reliable
  - This requires new iterative solver algorithms!
- Fruitful collaboration with systems researchers
  - Sandia’s “9 Lives” Group (Patrick Bridges, Kurt Ferreira)

Sandia National Laboratories

Recoverability of Linear Iterative Solvers

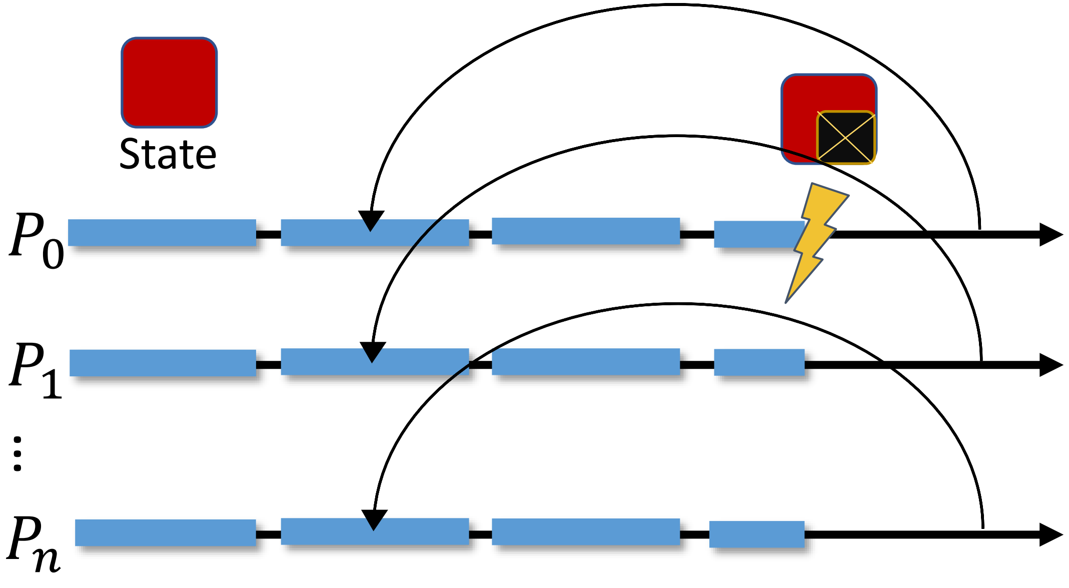
Checkpoint/Restart (C/R) to Storage



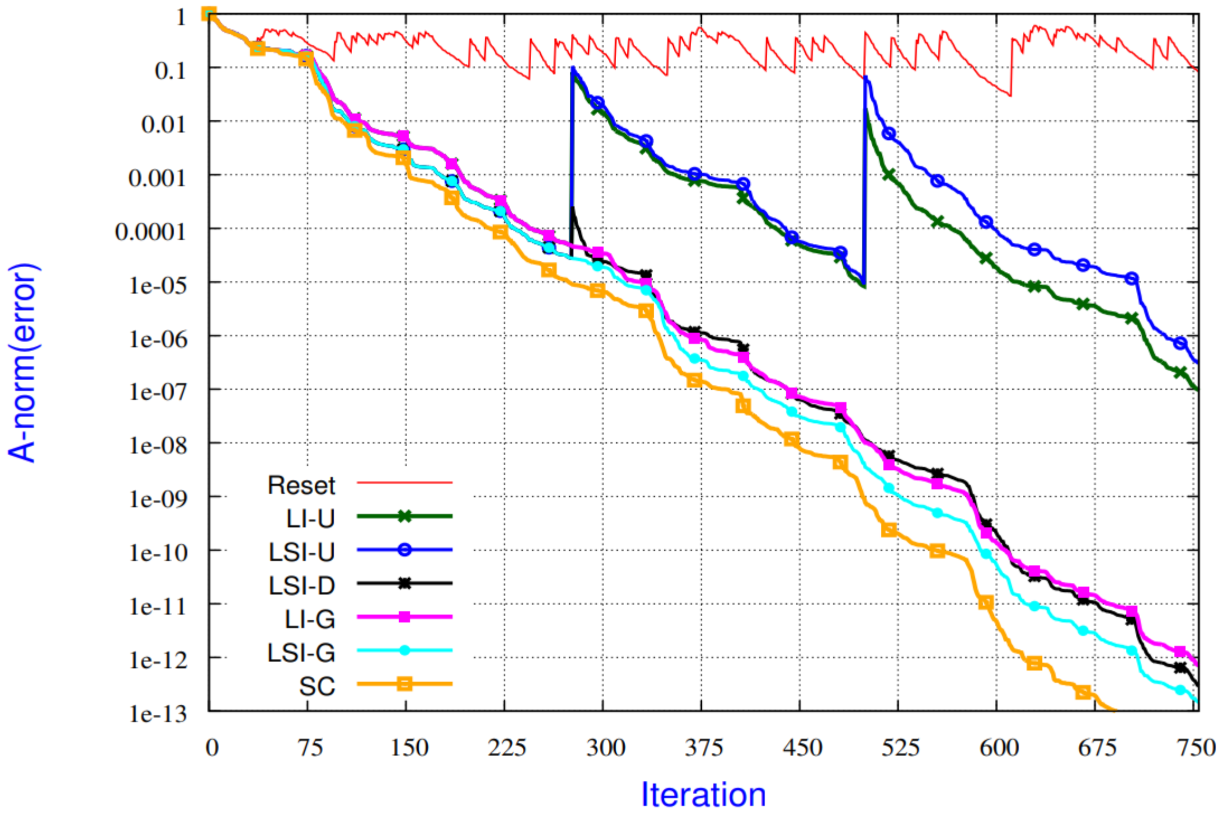
Pros	Cons
direct recovery, no extra calculations are needed to reconstruct the computation	Frequent checkpoints to standard HDD or SSD are **expensive** (in memory and time)

Recoverability of Linear Iterative Solvers

Intrinsic Recovery of iterative solvers



<span style="color:green">Pros</span>	<span style="color:red">Cons</span>
Minimal memory footprint and manipulation of recovery data	Reached state is **lost**, recovery requires extra calculations

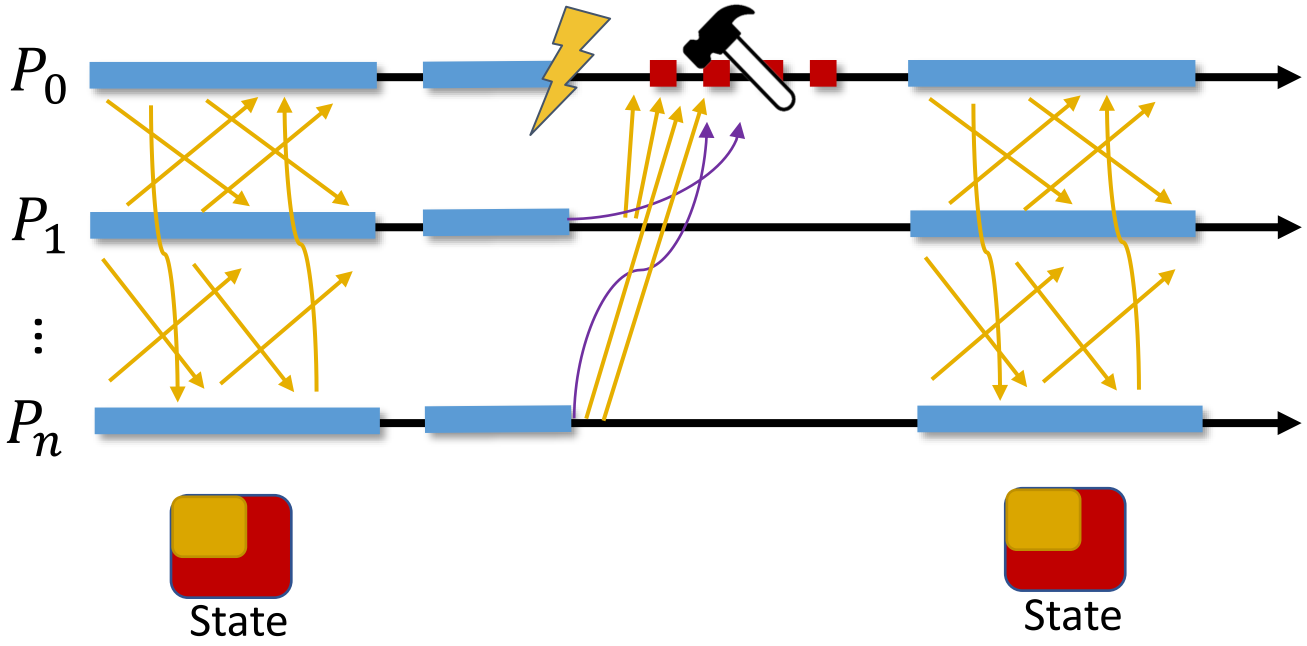


PCG on a 7-point stencil 3D Poisson equation using 32 processors with faults

Agullo, Emmanuel, et al. *Towards resilient parallel linear Krylov solvers: recover-restart strategies*. Diss. INRIA, 2013.

Recoverability of Linear Iterative Solvers

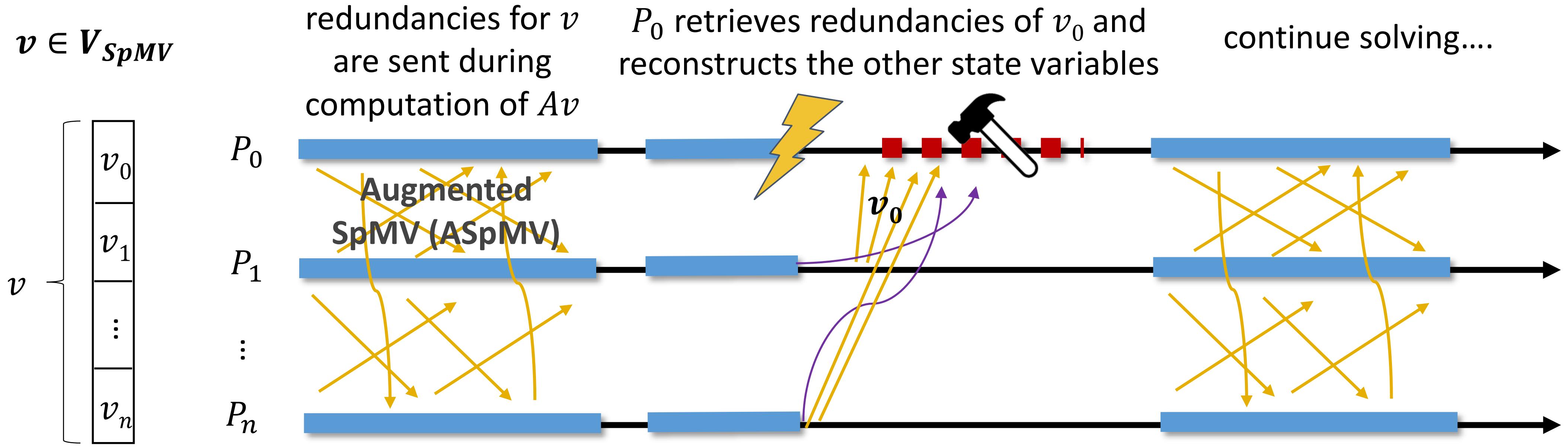
In-RAM Exact State Reconstruction (*ESR*)



Pros	Cons
Exact state reconstruction of failed processes	Huge memory and networking **overheads**

In-RAM Exact State Reconstruction (*ESR*)

- An *ESR* technique for **iterative linear solvers** was developed by Chen and refined by Pachajoa.
- Exploiting redundancies on other nodes' **RAM** from **Sparse matrix-vector multiplication (SpMV)** operations.
- $c + 1$ copies to support the **failure** of c **simultaneous** nodes.

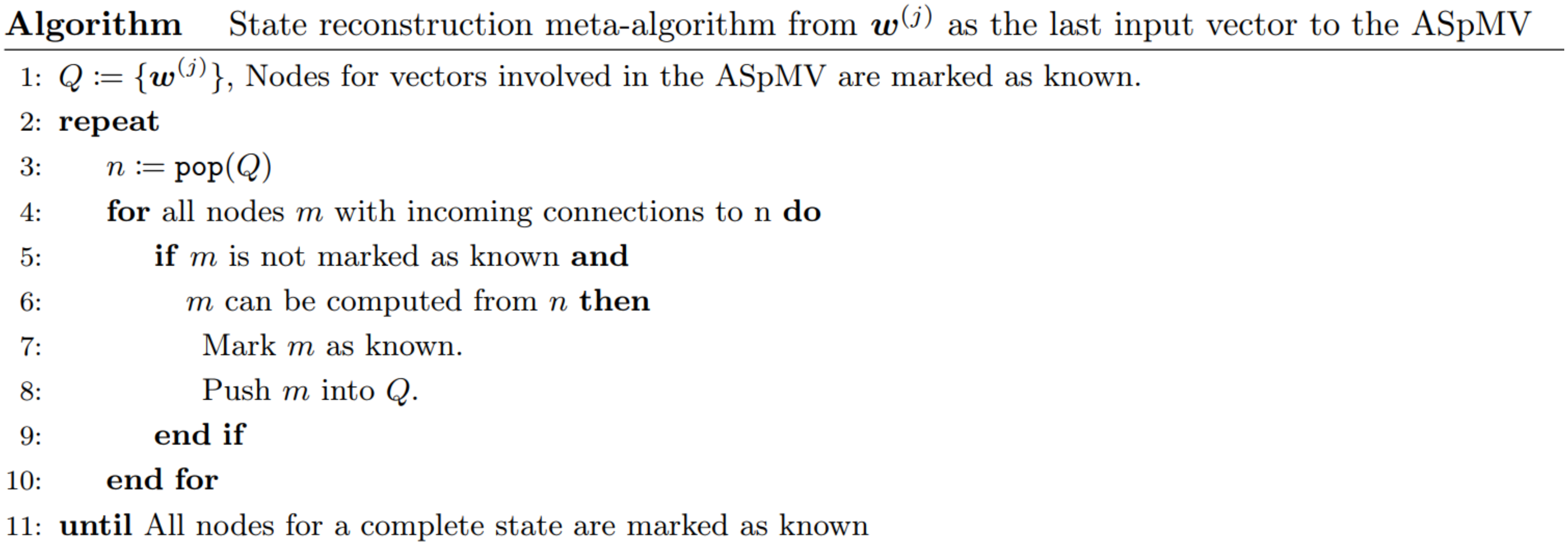


$\boldsymbol{v} \in \boldsymbol{V_{SpMV}}$
v
v_0
v_1
⋮
v_n
redundancies for v are sent during computation of Av
P_0 retrieves redundancies of v_0 and reconstructs the other state variables
continue solving....
P_0
P_1
⋮
P_n
Augmented SpMV (ASpMV)
$\boldsymbol{v_0}$

Zizhong Chen. "Algorithm-based Recovery for Iterative Methods Without Checkpointing". In: Proceedings of the 20th International Symposium on High Performance Distributed Computing. 2011, pp. 73–84.

In-RAM Exact State Reconstruction (*ESR*)

- A generic strategy was developed to implement *ESR* techniques for linear iterative solvers.



Algorithm State reconstruction meta-algorithm from $\boldsymbol{w}^{(j)}$ as the last input vector to the ASpMV

```
$Q := \{\boldsymbol{w}^{(j)}\}$, Nodes for vectors involved in the ASpMV are marked as known.
repeat
    $n :=$ pop($Q$)
    for all nodes $m$ with incoming connections to n do
        if $m$ is not marked as known and
          $m$ can be computed from $n$ then
            Mark $m$ as known.
            Push $m$ into $Q$.
        end if
    end for
until All nodes for a complete state are marked as known
```

Pachajoa, C. et al., “A generic strategy for node-failure resilience for certain iterative linear algebra methods,” in 2020 IEEE/ACM 10th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS). IEEE, 2020, pp. 41–50.

In-RAM Exact State Reconstruction (*ESR*)

In-RAM *ESR* Limitations:

- **Memory overheads:** A dramatic increase in memory overheads is required to support multiple processes' simultaneous failure.

Suppose keeping $c = \alpha \cdot N$ copies for each $v \in V_{SpMV}$ (when N is number of compute nodes).

Memory overhead for recovery increases **quadratically** with N.

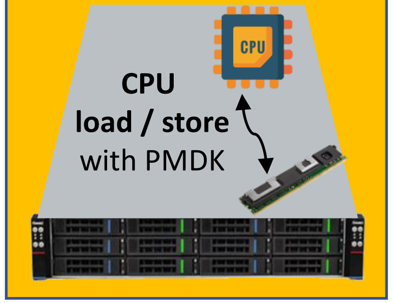
- **Networking & time overheads:** redundancies are passed to neighboring processes via Message Passing Interface, adding time overhead.
- **Hardware limitation:** *ESR* is not adjusted to work with NVRAM and exploit its persistence capabilities, which is the main disadvantage for the next supercomputers.

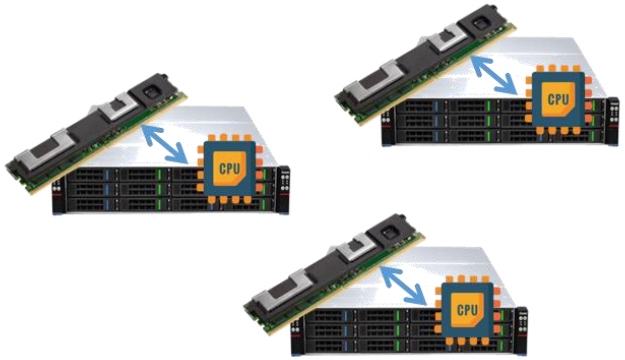
NVRAM Cluster Architectures

Two possible **cluster architectures** which involve **NVRAM** devices:

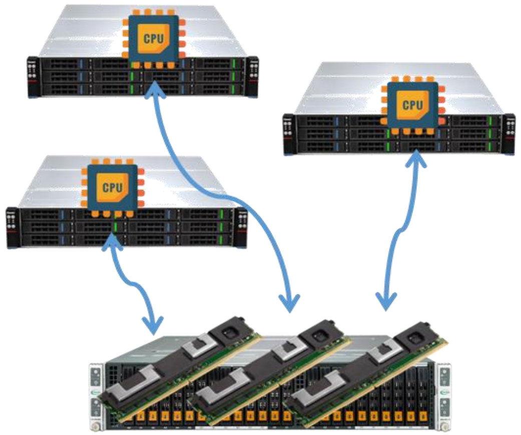
Homogeneous NVRAM cluster

Local persistent operations with **PMDK** libraries

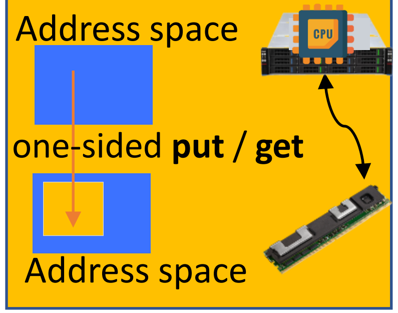




Persistent recovery data (PRD) NVRAM sub-cluster

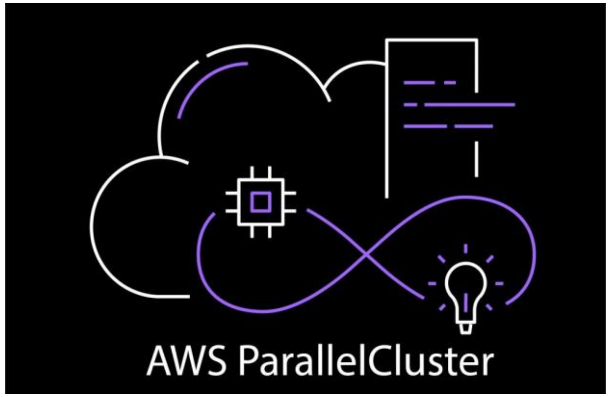


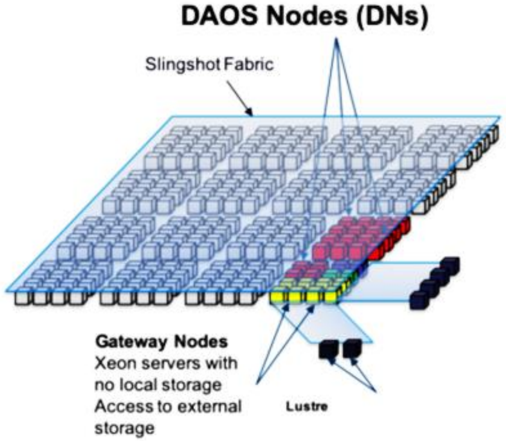
Persistent **RDMA** operations with **One-Sided MPI**


Address space

one-sided **put** / **get**

Address space

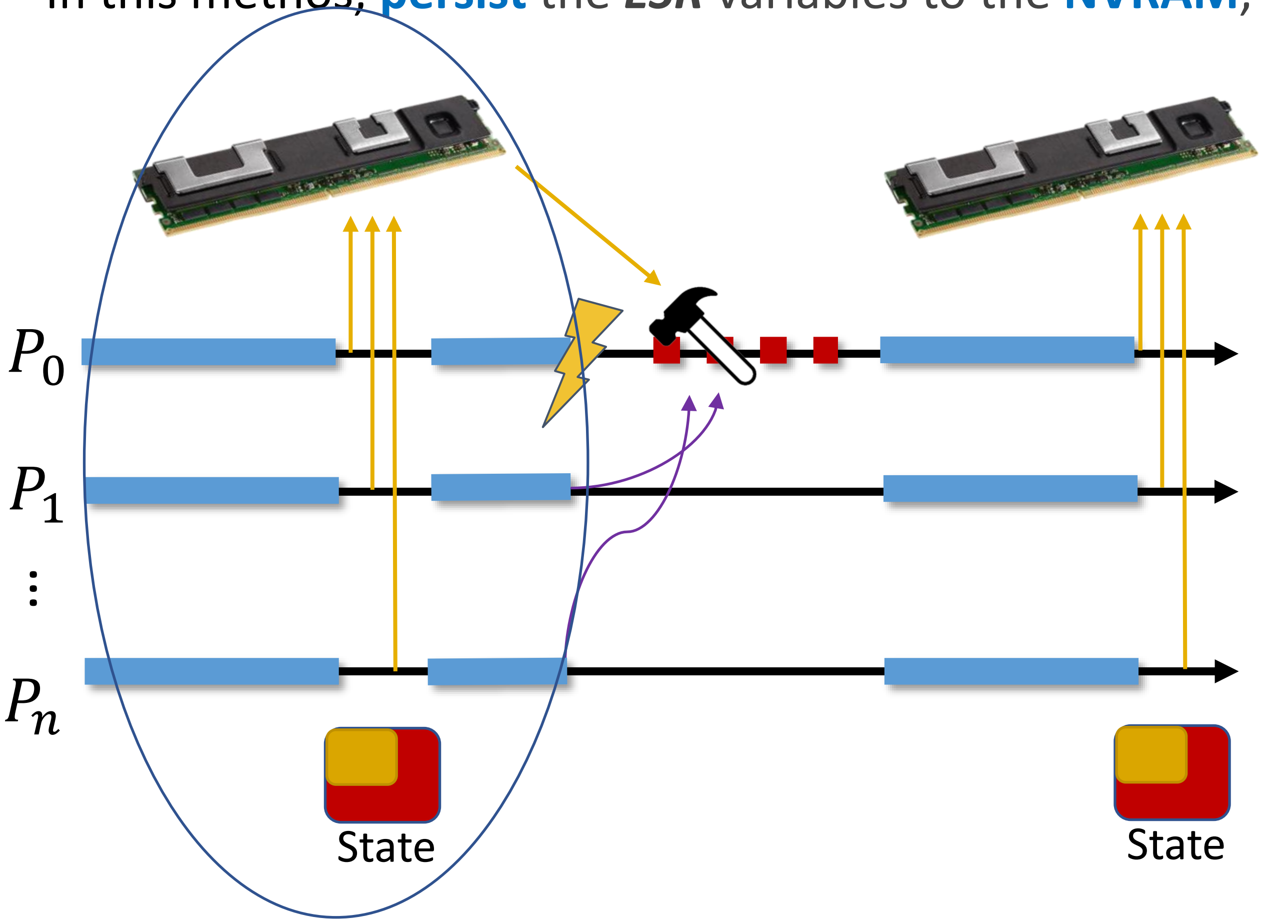




Recoverability of linear iterative solvers with **NVRAM**

Our Solution: **In-NVRAM** Exact State Reconstruction (*ESR*)

In this methos, **persist** the ***ESR*** variables to the **NVRAM**, and retrieve them in case of a failure.



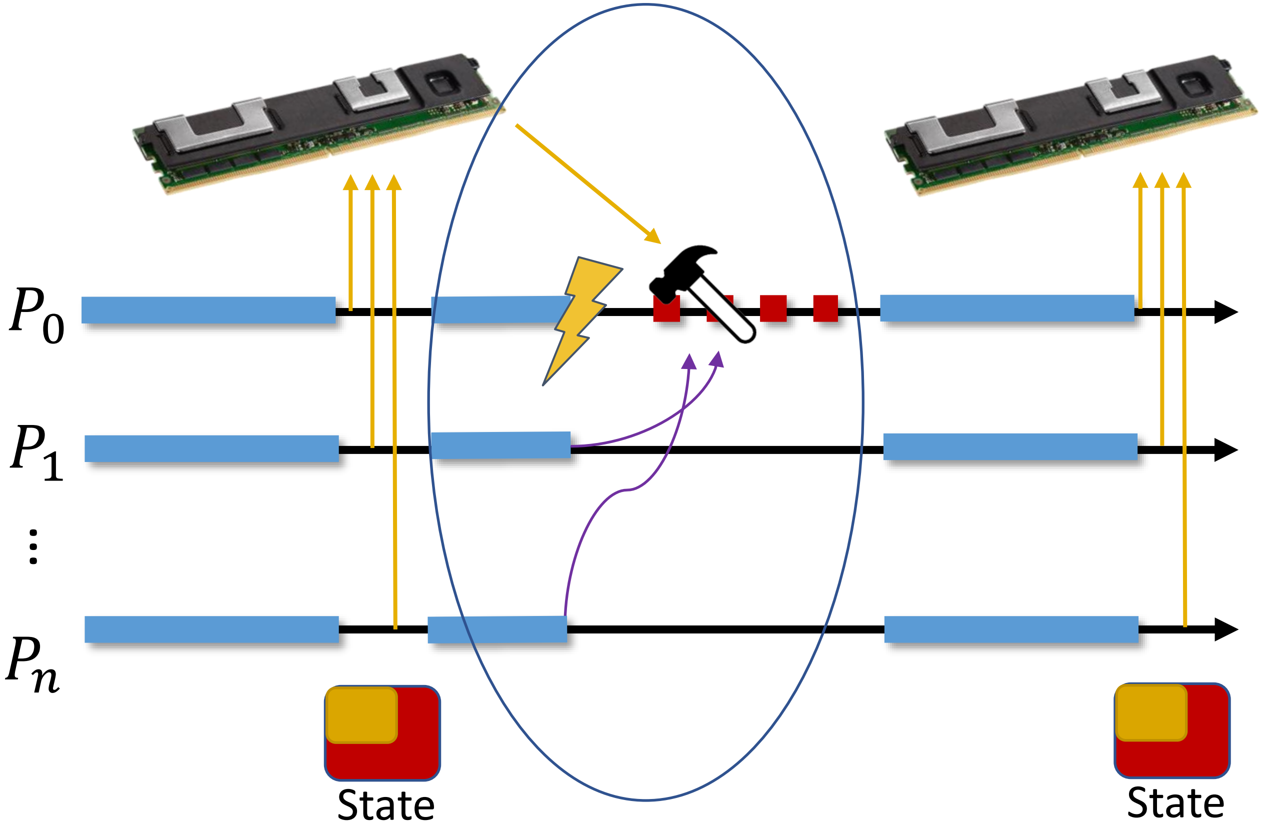
Algorithm *In-NVRAM* ESR for a persistence iteration j of process p

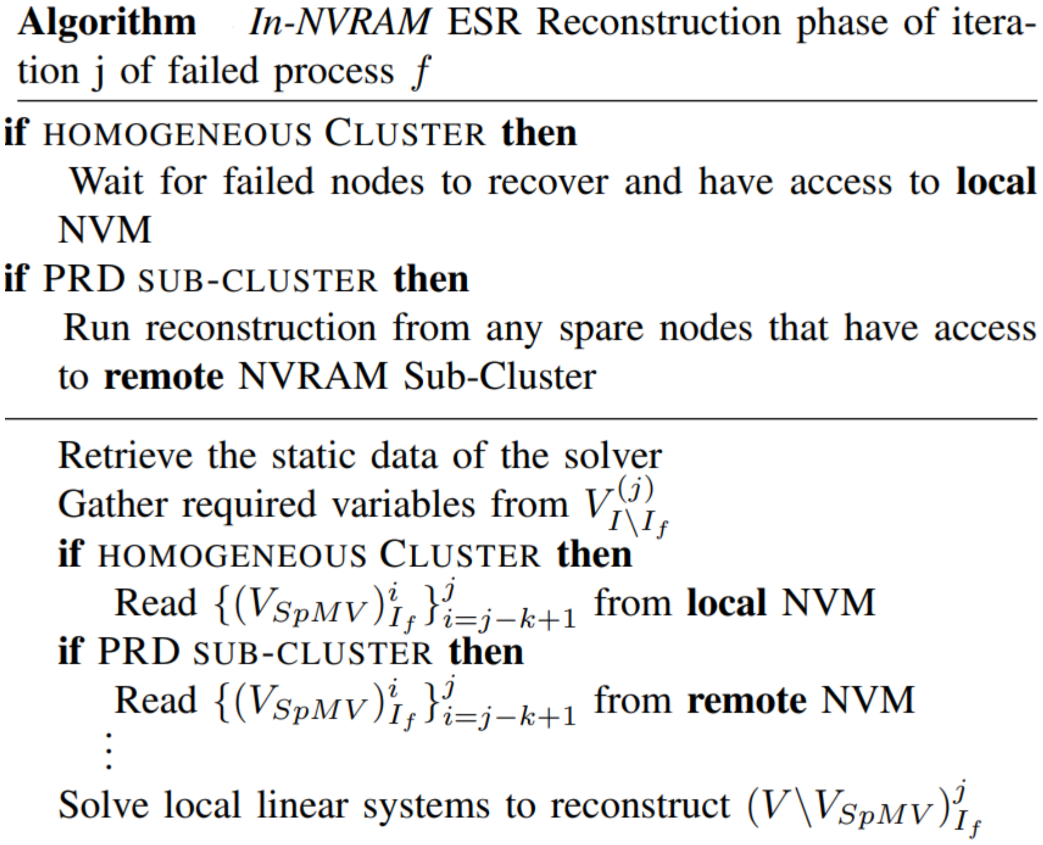
```
Compute $j^{th}$ iteration of the solver
  ⋮
if HOMOGENEOUS CLUSTER then
    persist $(V_{SpMV})^j_{I_p}$ to local NVM
if PRD SUB-CLUSTER then
    persist $(V_{SpMV})^j_{I_p}$ to remote NVM
```

Recoverability of linear iterative solvers with **NVRAM**

Our Solution: **In-NVRAM** Exact State Reconstruction (*ESR*)

Reconstruction stage:





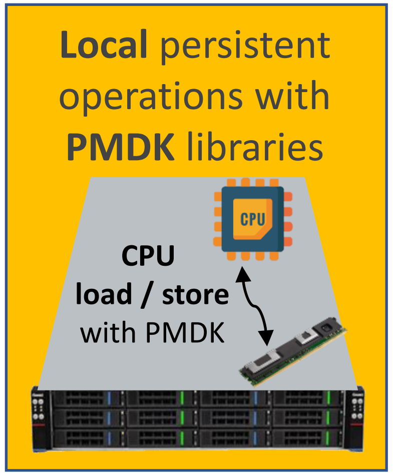
Algorithm *In-NVRAM* ESR Reconstruction phase of iteration j of failed process f

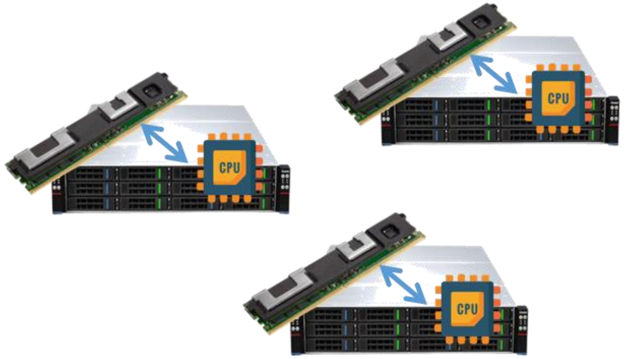
```
if HOMOGENEOUS CLUSTER then
    Wait for failed nodes to recover and have access to local NVM
if PRD SUB-CLUSTER then
    Run reconstruction from any spare nodes that have access to remote NVRAM Sub-Cluster

Retrieve the static data of the solver
Gather required variables from $V^{(j)}_{I\backslash I_f}$
if HOMOGENEOUS CLUSTER then
    Read $\{(V_{SpMV})^{i}_{I_f}\}^{j}_{i=j-k+1}$ from local NVM
if PRD SUB-CLUSTER then
    Read $\{(V_{SpMV})^{i}_{I_f}\}^{j}_{i=j-k+1}$ from remote NVM
  ⋮
Solve local linear systems to reconstruct $(V\backslash V_{SpMV})^{j}_{I_f}$
```

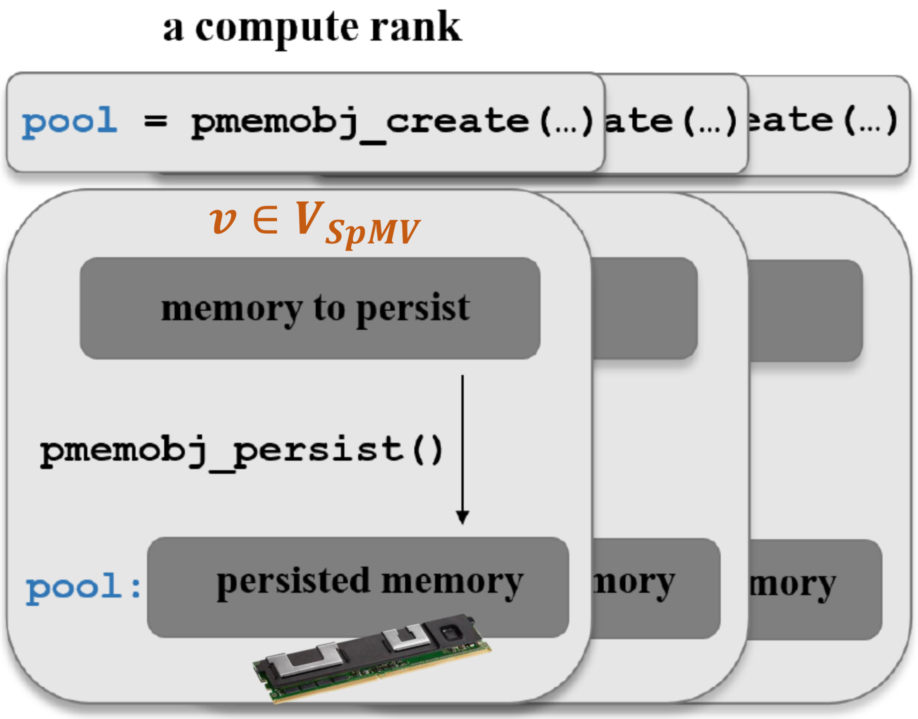
NVRAM Cluster Architectures

Implementation details:

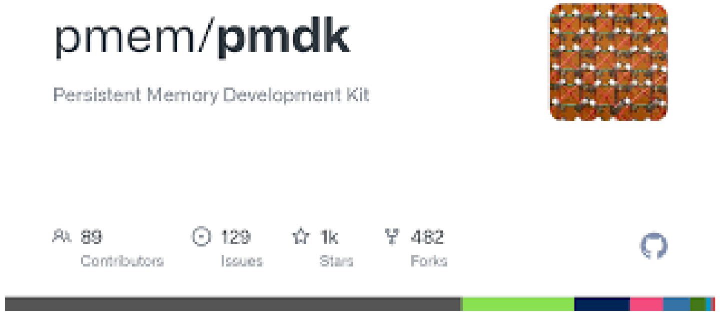
Homogeneous NVRAM cluster


Local persistent operations with **PMDK** libraries




a compute rank

```
pool = pmemobj_create(…)
```

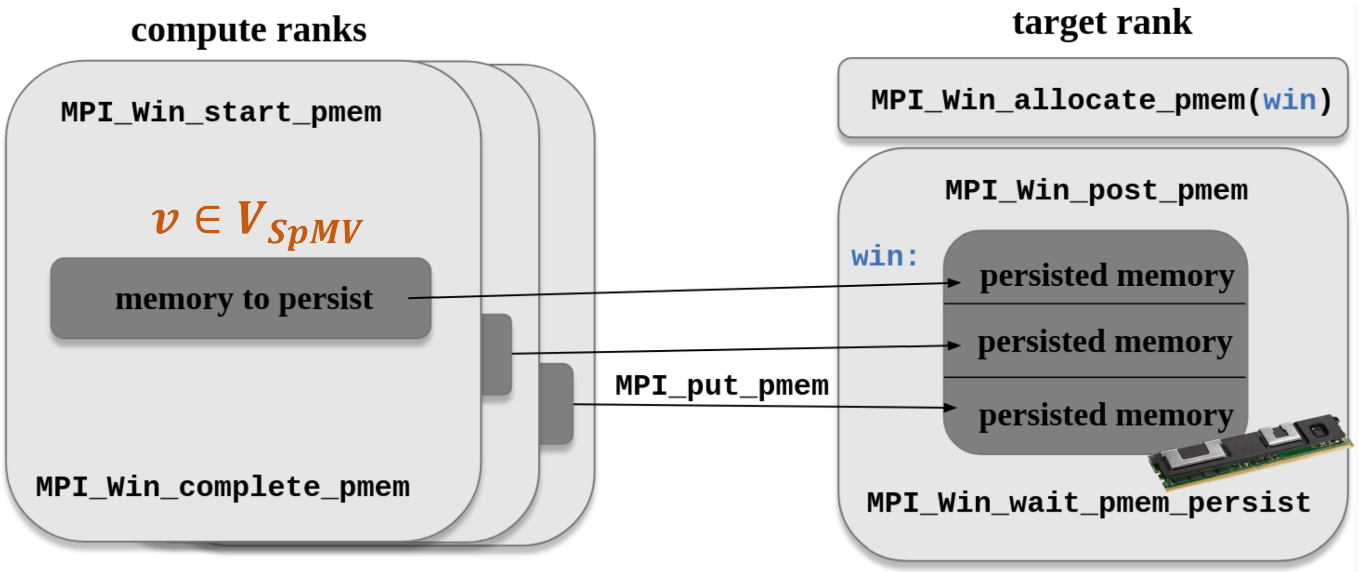

pmem/**pmdk**

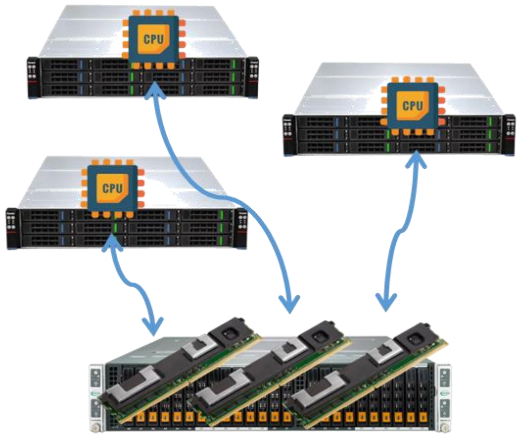
Persistent Memory Development Kit

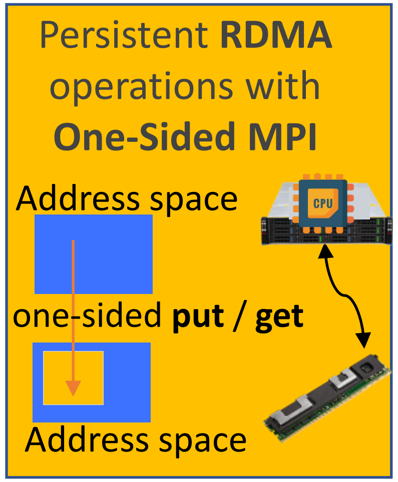
NVRAM Cluster Architectures

Implementation details:

Persistent recovery data (PRD) NVRAM sub-cluster





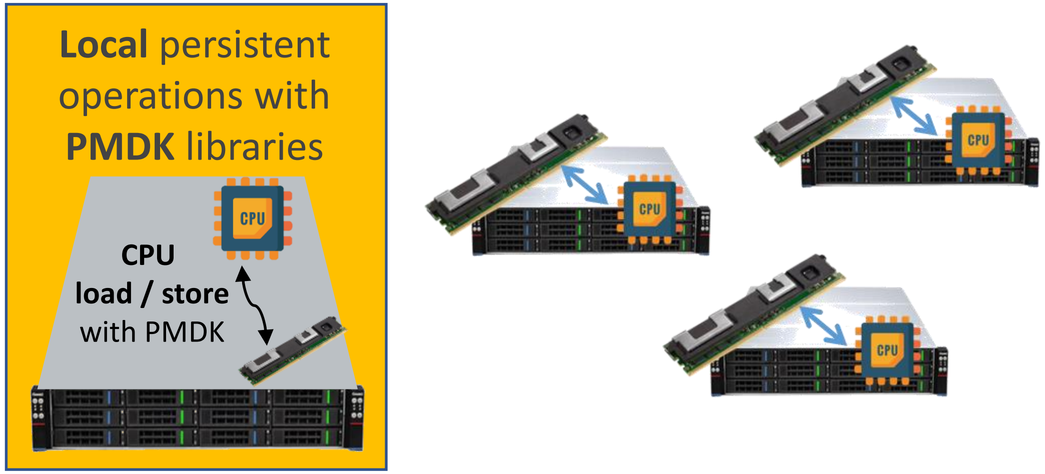

Persistent **RDMA** operations with **One-Sided MPI**
Address space
one-sided **put** / **get**
Address space

Post-Start-Complete-Wait synchronization protocol

Dorozy˙ nski, P. ´ et al., “Checkpointing of parallel mpi applications using mpi one-sided api with support for byte-addressable non-volatile ram,” Procedia Computer Science, vol. 80, pp. 30–40, 2016.

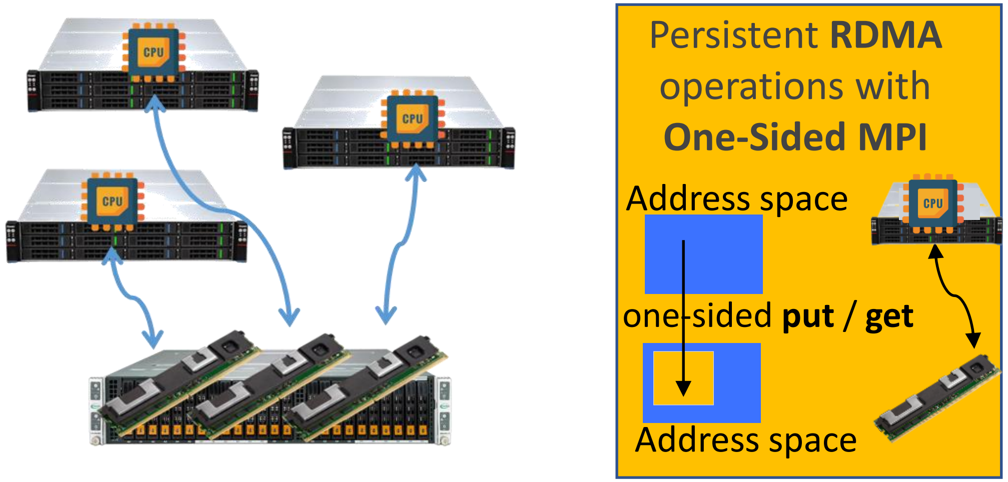
NVRAM Cluster Architectures

Homogeneous NVRAM cluster



Local persistent operations with **PMDK** libraries

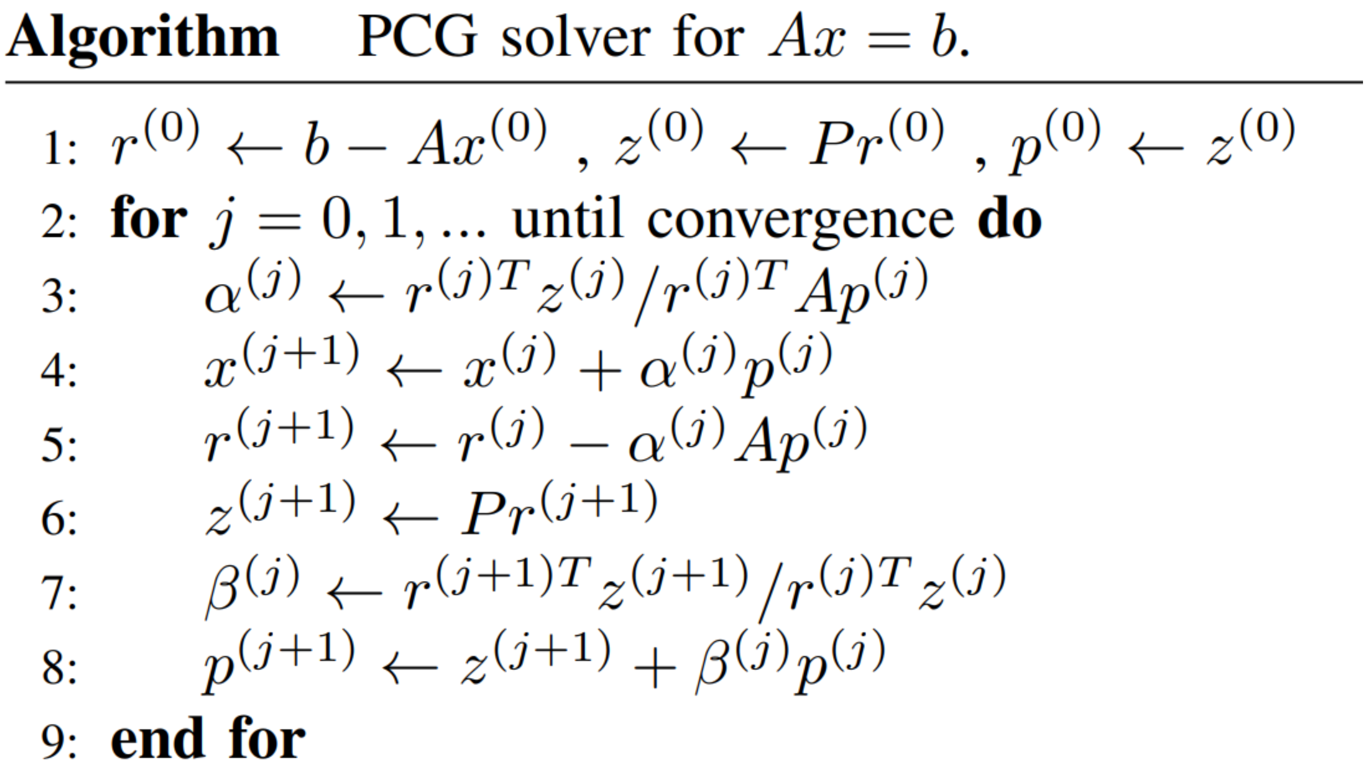
Persistent recovery data (PRD) NVRAM sub-cluster



Persistent **RDMA** operations with **One-Sided MPI**

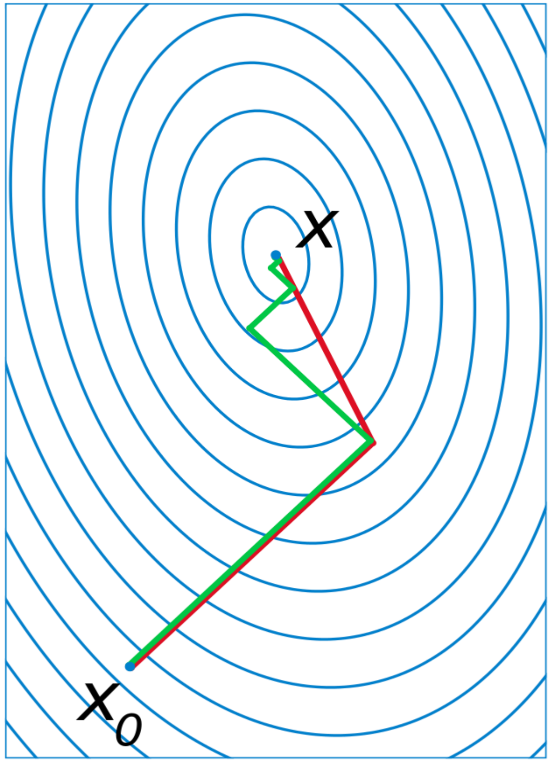
	Homogeneous NVRAM cluster	PRD NVRAM sub-cluster
Pros	persistency is local, so **no network overheads**	when node fails, can reconstruct its state from a **replacement node** accessing the PRD sub-cluster
Cons	when node fails, **need to wait for its recovery** to be able to access its NVRAM	(1) **networking overheads** for persisting to the NVRAM sub-cluster (2) the PRD sub-cluster is a **single point of failure**

Example: Preconditioned Conjugate Gradient (PCG)



Algorithm PCG solver for $Ax = b$.

```
$r^{(0)} \leftarrow b - Ax^{(0)}$ , $z^{(0)} \leftarrow Pr^{(0)}$ , $p^{(0)} \leftarrow z^{(0)}$
for $j = 0, 1, \ldots$ until convergence do
    $\alpha^{(j)} \leftarrow r^{(j)T} z^{(j)} / r^{(j)T} A p^{(j)}$
    $x^{(j+1)} \leftarrow x^{(j)} + \alpha^{(j)} p^{(j)}$
    $r^{(j+1)} \leftarrow r^{(j)} - \alpha^{(j)} A p^{(j)}$
    $z^{(j+1)} \leftarrow P r^{(j+1)}$
    $\beta^{(j)} \leftarrow r^{(j+1)T} z^{(j+1)} / r^{(j)T} z^{(j)}$
    $p^{(j+1)} \leftarrow z^{(j+1)} + \beta^{(j)} p^{(j)}$
end for
```

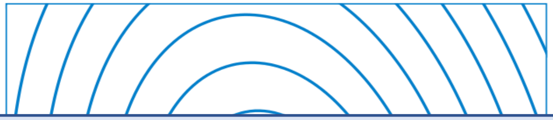


According to the *ESR* technique, redundancies are kept for the variable p

Example: Preconditioned Conjugate Gradient (PCG)

Algorithm PCG solver for $Ax = b$.

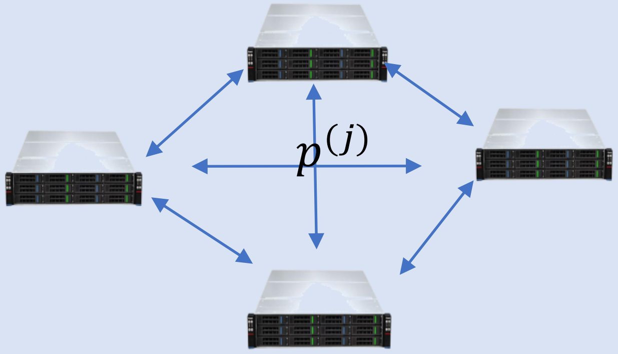
```
$r^{(0)} \leftarrow b - Ax^{(0)}$ , $z^{(0)} \leftarrow Pr^{(0)}$ , $p^{(0)} \leftarrow z^{(0)}$
for $j = 0, 1, \ldots$ until convergence do
    $\alpha^{(j)} \leftarrow r^{(j)T} z^{(j)} / r^{(j)T} \cancel{Ap}^{(j)}$
    $x^{(j+1)} \leftarrow x^{(j)} + \alpha^{(j)} p^{(j)}$
    $r^{(j+1)} \leftarrow r^{(j)} - \alpha^{(j)} Ap^{(j)}$
    $z^{(j+1)} \leftarrow Pr^{(j+1)}$
    $\beta^{(j)} \leftarrow r^{(j+1)T} z^{(j+1)} / r^{(j)T} z^{(j)}$
    $p^{(j+1)} \leftarrow z^{(j+1)} + \beta^{(j)} p^{(j)}$
end for
```



In-RAM *ESR*

$ASpMV(A, p^{(j)})$

to create redundancies on other nodes' RAM

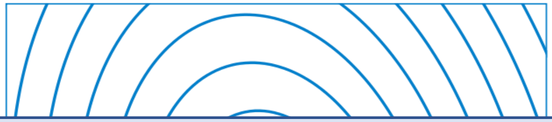


According to the *ESR* technique, redundancies are kept for the variable p

Example: Preconditioned Conjugate Gradient (PCG)

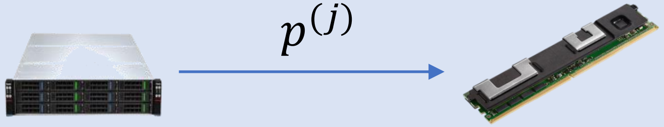
Algorithm PCG solver for $Ax = b$.

```
$r^{(0)} \leftarrow b - Ax^{(0)}$ , $z^{(0)} \leftarrow Pr^{(0)}$ , $p^{(0)} \leftarrow z^{(0)}$
for $j = 0, 1, \ldots$ until convergence do
    $\alpha^{(j)} \leftarrow r^{(j)T} z^{(j)} / r^{(j)T} Ap^{(j)}$
    $x^{(j+1)} \leftarrow x^{(j)} + \alpha^{(j)} p^{(j)}$
    $r^{(j+1)} \leftarrow r^{(j)} - \alpha^{(j)} Ap^{(j)}$
    $z^{(j+1)} \leftarrow Pr^{(j+1)}$
    $\beta^{(j)} \leftarrow r^{(j+1)T} z^{(j+1)} / r^{(j)T} z^{(j)}$
    $p^{(j+1)} \leftarrow z^{(j+1)} + \beta^{(j)} p^{(j)}$
end for
```



In-NVRAM *ESR*

Persist $p^{(j)}$ to NVRAM (locally or remotely)

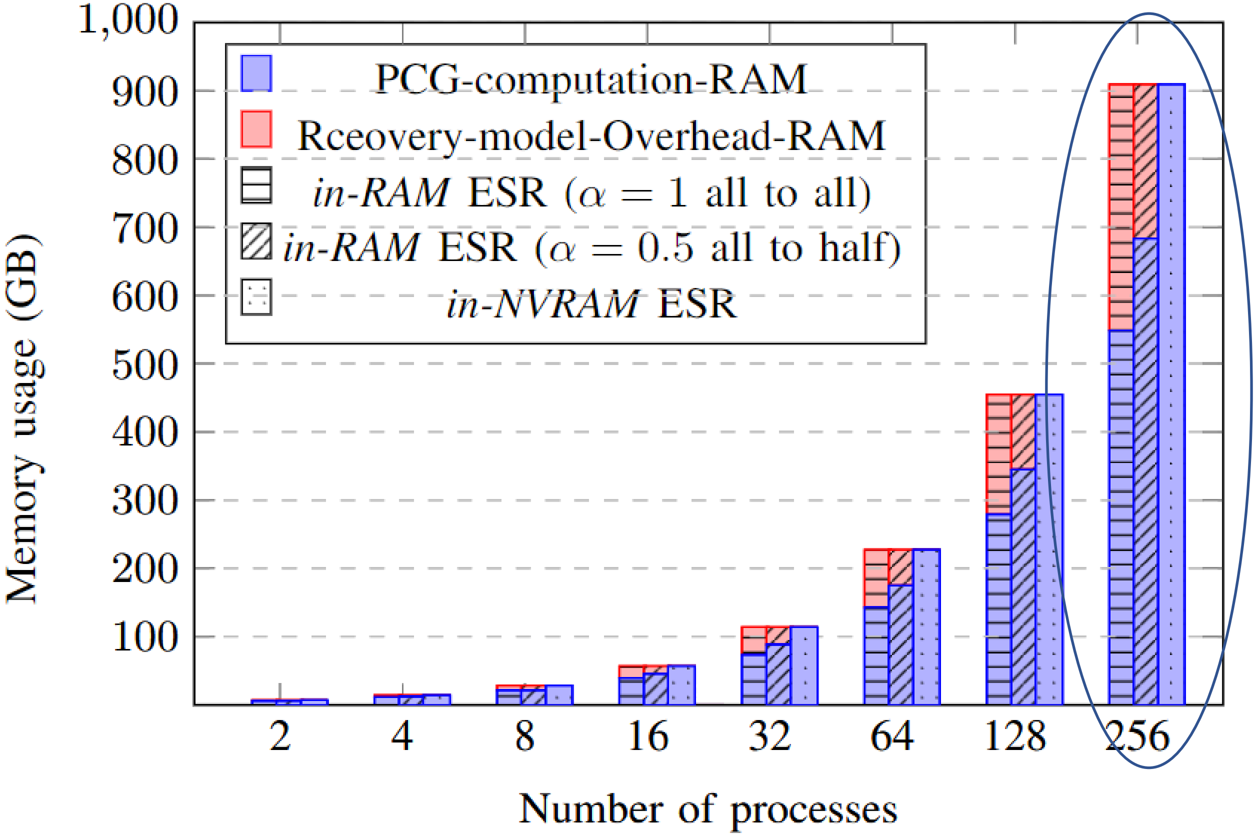


According to the *ESR* technique, redundancies are kept for the variable p

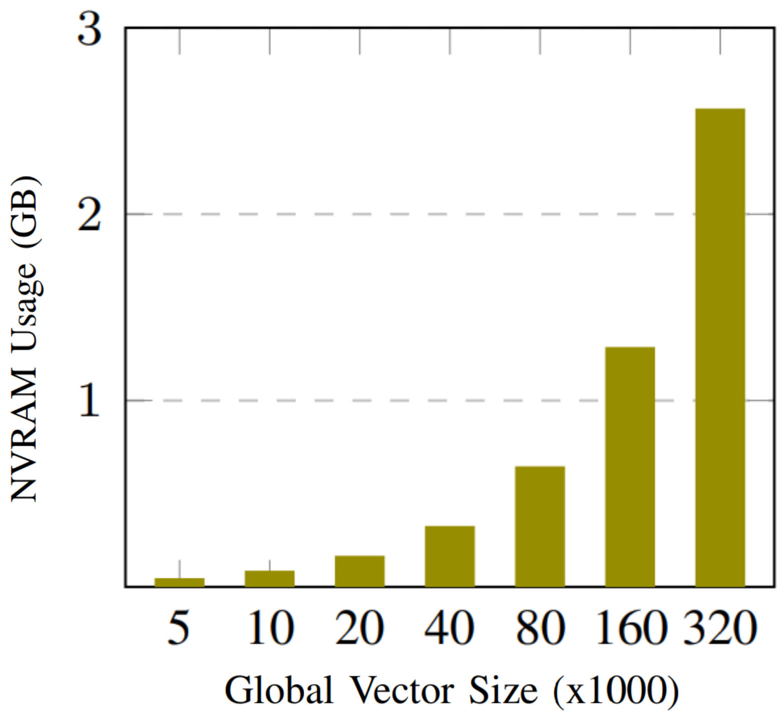
Results

Reducing **memory footprint:**

RAM usage for **in-NVRAM** and **in-RAM** *ESR*



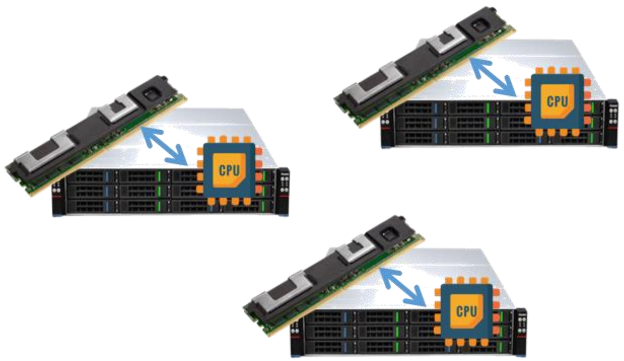
NVRAM usage in **in-NVRAM** *ESR*



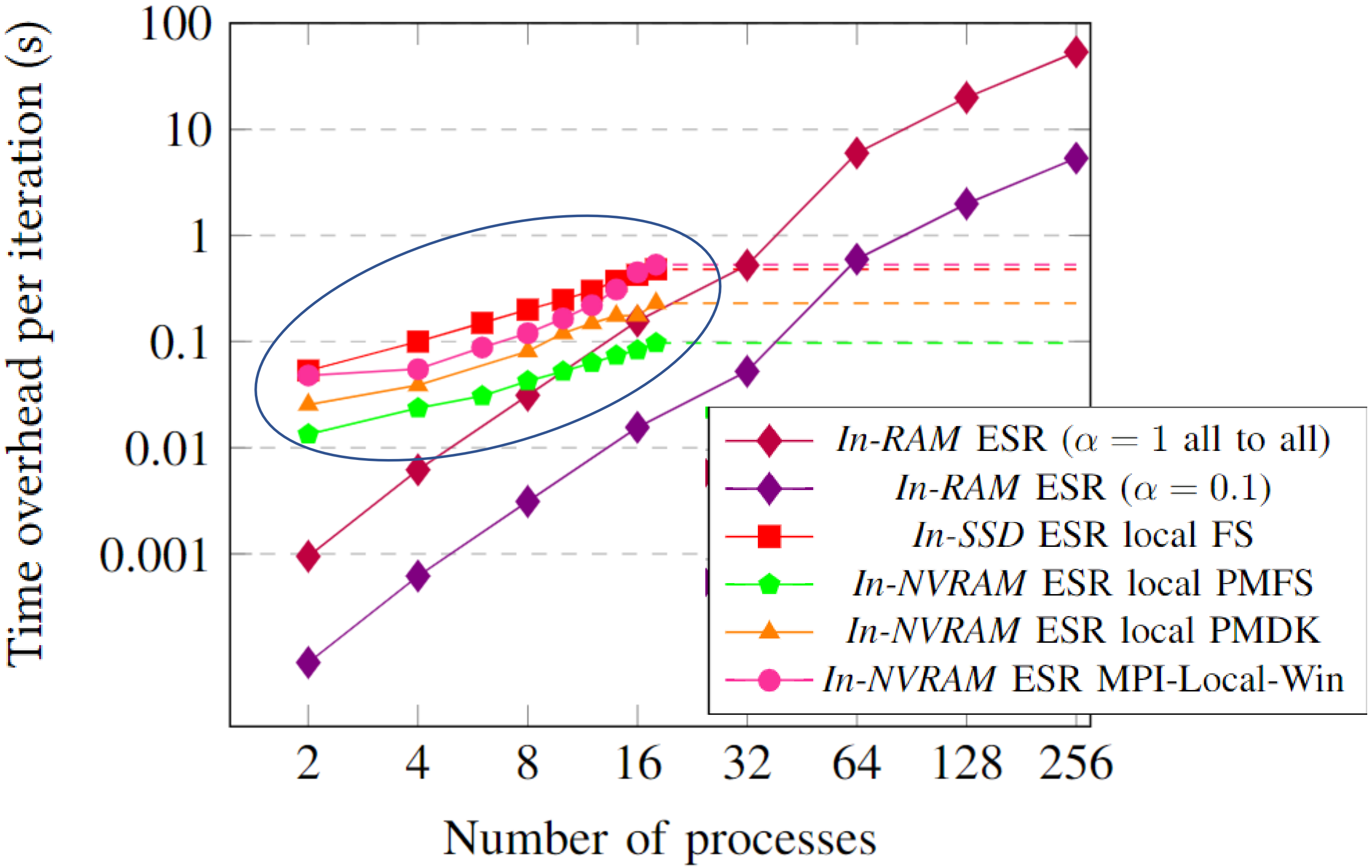
Results

Reducing **time overhead:**

Homogeneous NVRAM cluster



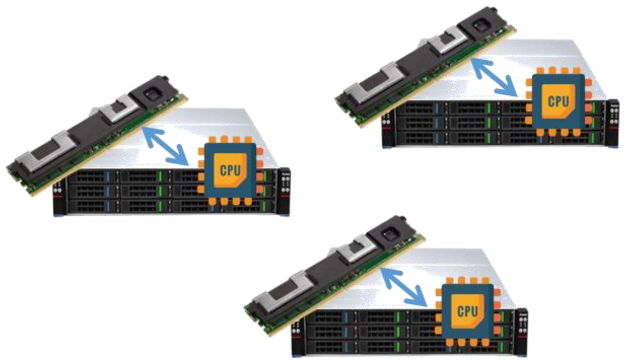
logscale



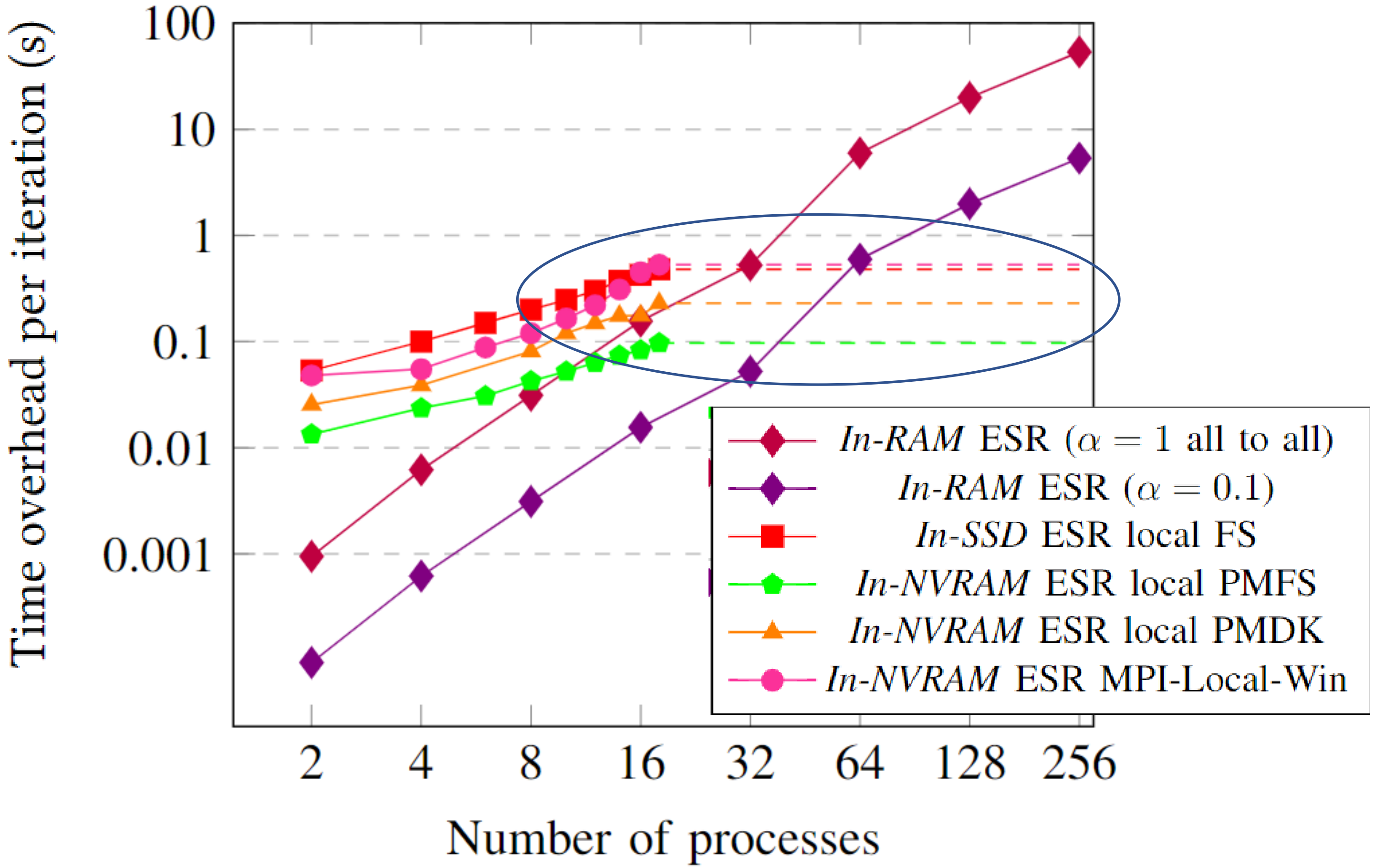
Results

Reducing **time overhead:**

Homogeneous NVRAM cluster



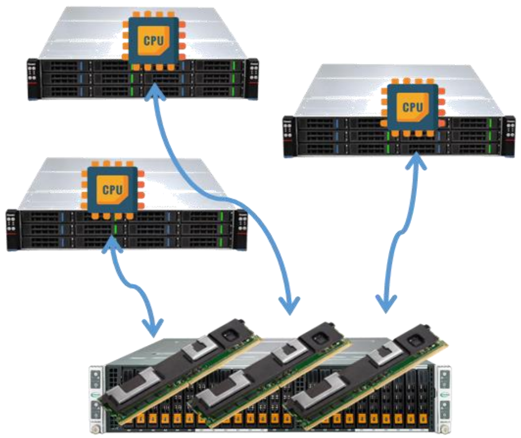
logscale

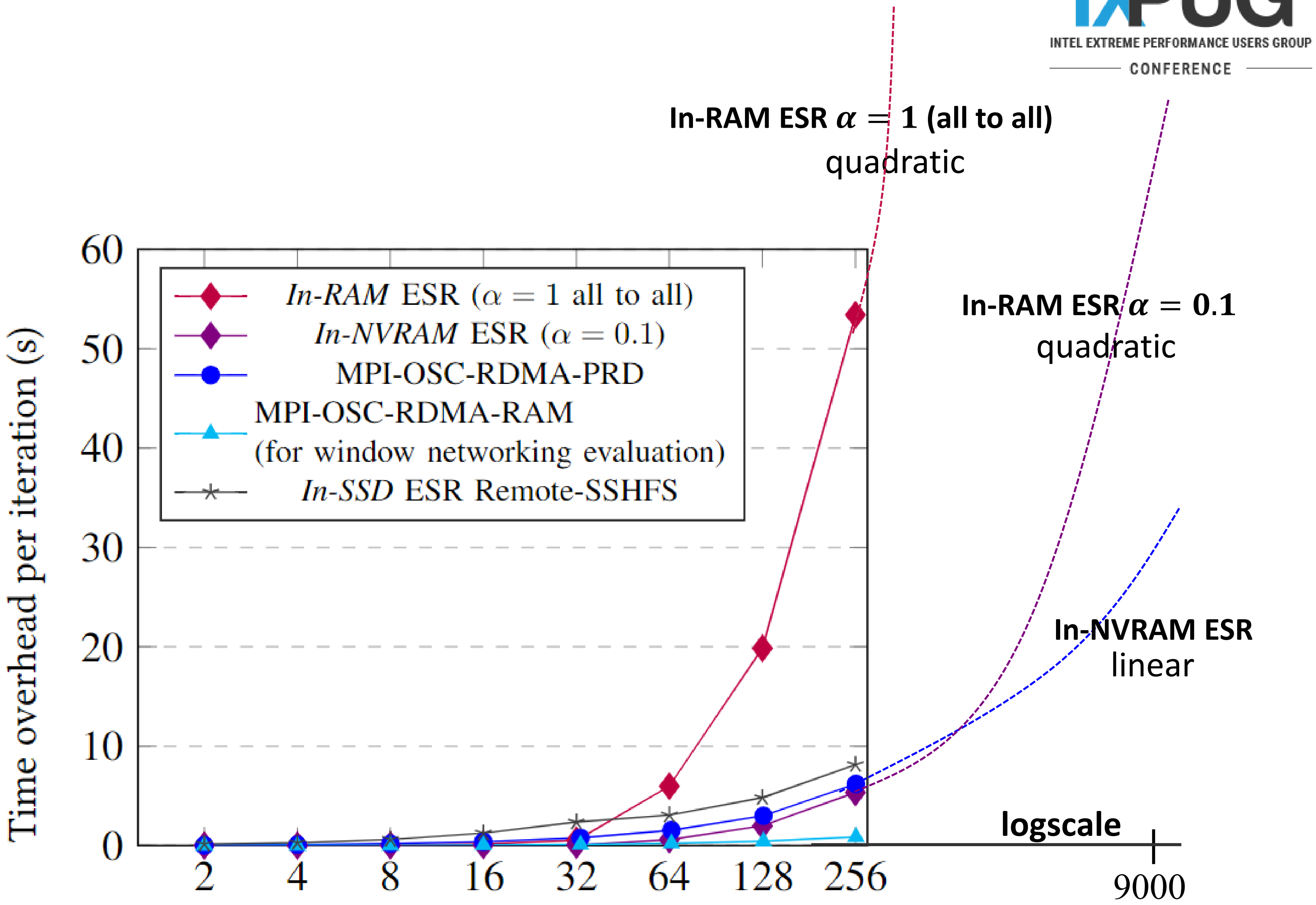


Results

Reducing **time overhead:**

PRD NVRAM sub-cluster





In-RAM ESR $\alpha = 1$ (all to all)
quadratic

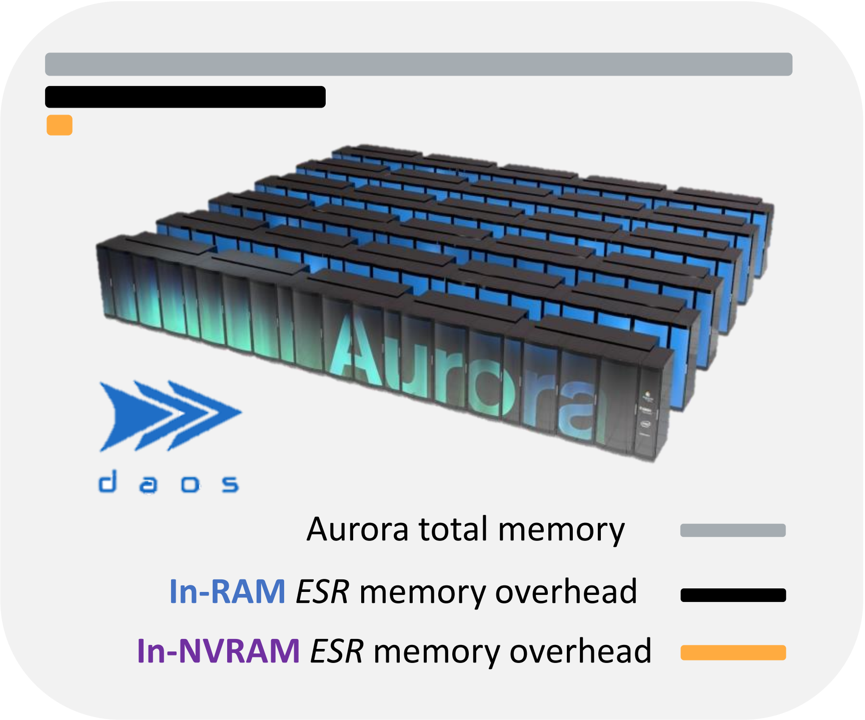
In-RAM ESR $\alpha = 0.1$
quadratic

In-NVRAM ESR
linear

Extrapolation to Aurora

Expected **memory benefits on the Aurora supercomputer**

- **9,000** nodes, total memory of the system ~**10PB.**
- ~**230PB** DAOS storage (Includes **DCPMM**).
- We extrapolate and estimate the **in-RAM** *ESR* memory overhead for a hero computation in **~3PB (**for $\alpha = 1$) and **~300TB (**for $\alpha = \mathbf{0.1}$).
- ***In-NVRAM*** *ESR* overhead can be reduced to ~0.3TB on NVRAM (up to **x9000** reduction).

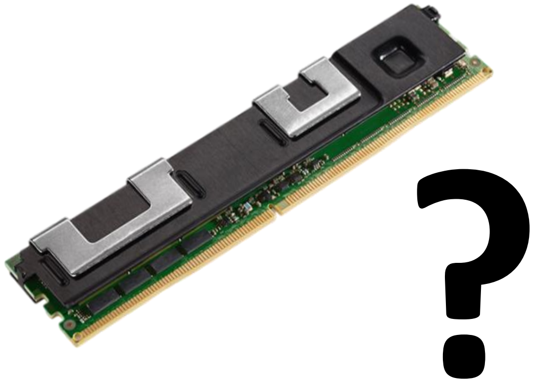


Conclusions

- In this work we investigate how to significantly improve the performance of **in-RAM** *ESR* using NVRAM, presenting the **In-NVRAM** *ESR* model.
- **In-NVRAM** *ESR* utilizes the recently enabled capabilities **of direct access (DAX)** to NVRAM and the access to such memory using MPI **One-Sided Communication (OSC) over RDMA**.
- **In-NVRAM** *ESR* significantly reduces memory footprint and time overheads for recoverability.
- We consider both on-node and remote sub-cluster architectures, as they both relevant.

Discussion

- Can we reduce programmer effort by automation?
- What about direct solvers?
- The Future of Intel Optane?



https://github.com/Scientific-Computing-Lab-NRCN/In-NVRAM-ESR.git

IXPUG
INTEL EXTREME PERFORMANCE USERS GROUP
CONFERENCE

Recovery of Distributed Iterative Solvers for Linear Systems Using Non-Volatile RAM

Yehonatan Fridman, Yaniv Snir, Harel Levin, Danny Hendler, Hagit Attiya, and Gal Oren*

*galoren@cs.technion.ac.il