Dynamic Selective Protection of Sparse Iterative Solvers via ML Prediction of Soft Error Impacts

Zizhao Chen[1], Thomas Verrecchia[2], Hongyang Sun[1] (speaker),
Joshua D. Booth[3], Padma Raghavan[4]

[1] University of Kansas, USA

[2] ENSTA Paris, France

[3] University of Alabama in Huntsville, USA

[4] Vanderbilt University, USA

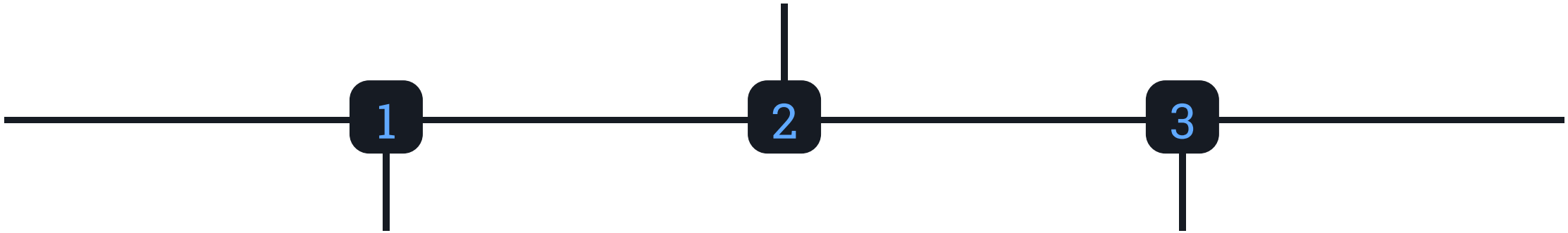
Fault Tolerance for HPC at eXtreme Scales (FTXS) Workshop

Nov. 12, 2023 @ Denver, CO, USA

Sparse iterative solvers are critical in scientific computing for solving sparse systems of linear equations. However, these methods are vulnerable to **hard failures** and **soft errors**. To ensure the effective use of HPC systems, **resilience techniques** must be deployed.

Resilience Techniques

Protect scientific applications from failures or errors at different levels



Soft Errors

Common on large computing platforms, leading to **performance degradation**

Selective Protection

Reduce resilience overheads by **selectively protecting most critical elements**

PCG Algorithm: an iterative solver for sparse linear systems

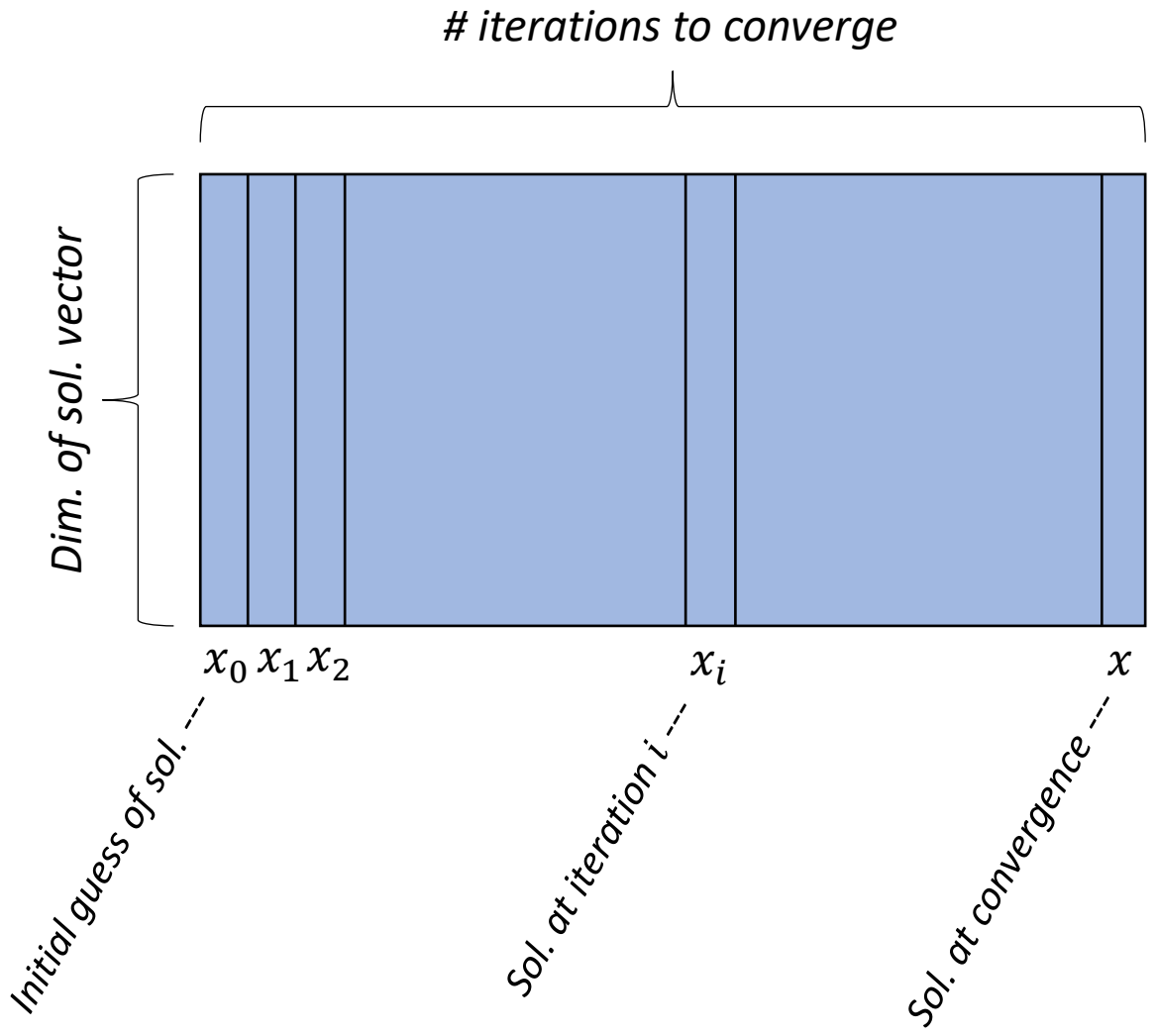
$$\boldsymbol{Ax = b}$$

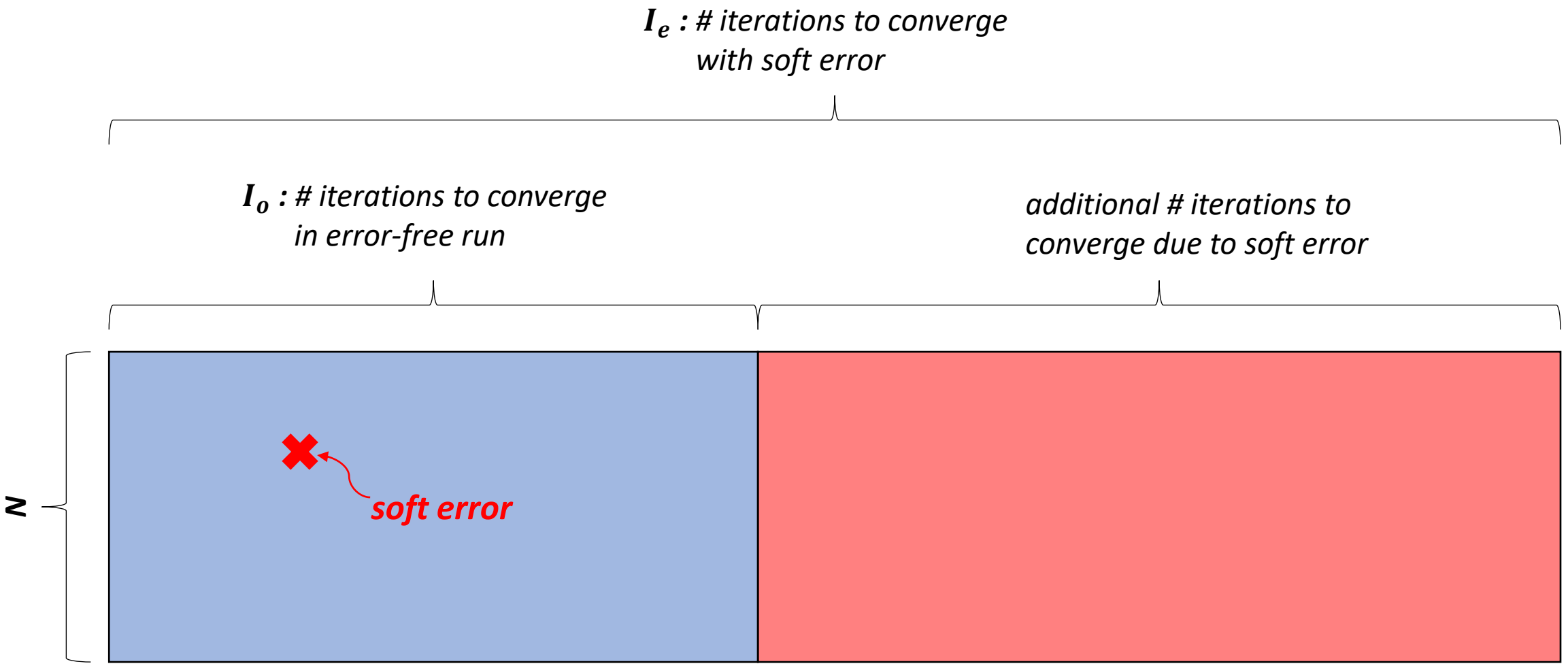
Algorithm 1: Preconditioned Conjugate Gradient (PCG)

Input: $A, M, b, x_0, tol, maxit$

begin
- $r_0 \leftarrow b - Ax_0$ // Initial residual
- $z_0 \leftarrow M^{-1}r_0$ // Preconditioning
- $p_0 \leftarrow z_0$
- $i \leftarrow 0$
- **while** $i < maxit$ and $\|r_i\|/\|b\| > tol$ **do**
  - $q_i \leftarrow Ap_i$
  - $v_i \leftarrow r_i^T z_i$
  - $\alpha \leftarrow v_i/(p_i^T q_i)$
  - $x_{i+1} \leftarrow x_i + \alpha p_i$ // Improve approximation
  - $r_{i+1} \leftarrow r_i - \alpha q_i$ // Update residual
  - $z_{i+1} \leftarrow M^{-1}r_{i+1}$ // Preconditioning
  - $v_{i+1} \leftarrow r_{i+1}^T z_{i+1}$
  - $\beta \leftarrow v_{i+1}/v_i$
  - $p_{i+1} \leftarrow z_{i+1} + \beta p_i$ // New search direction
  - $i \leftarrow i + 1$
- **end**

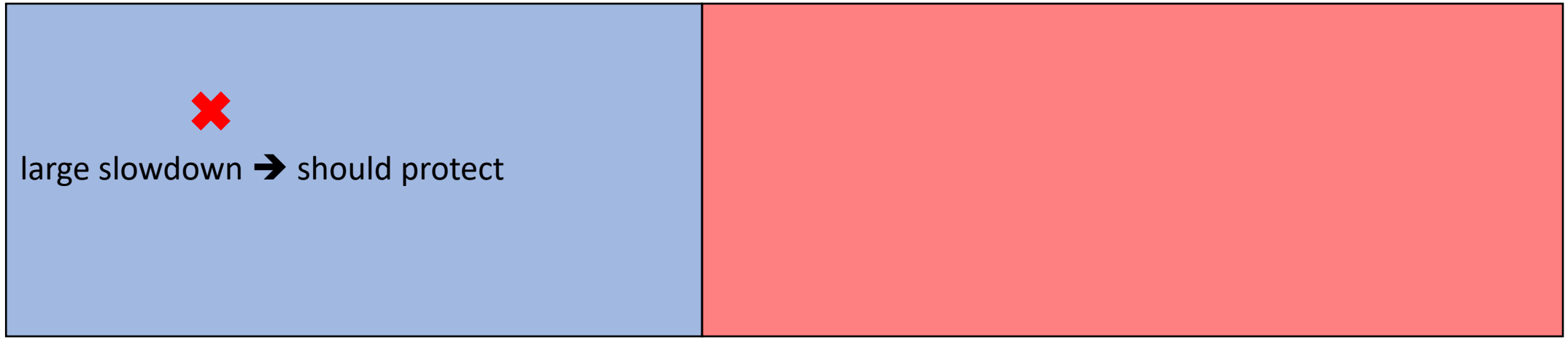
end

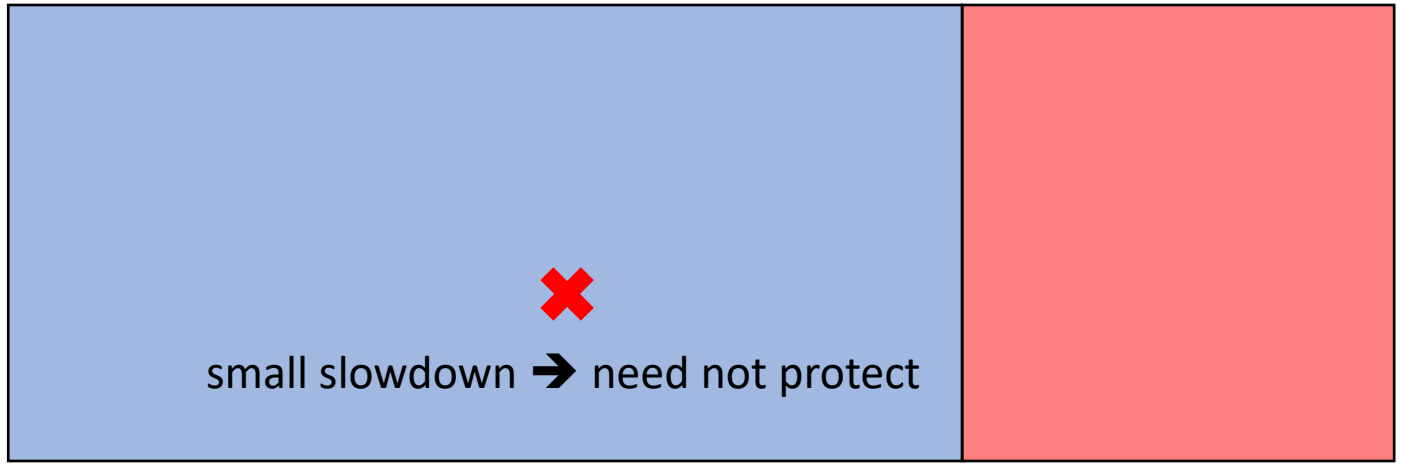




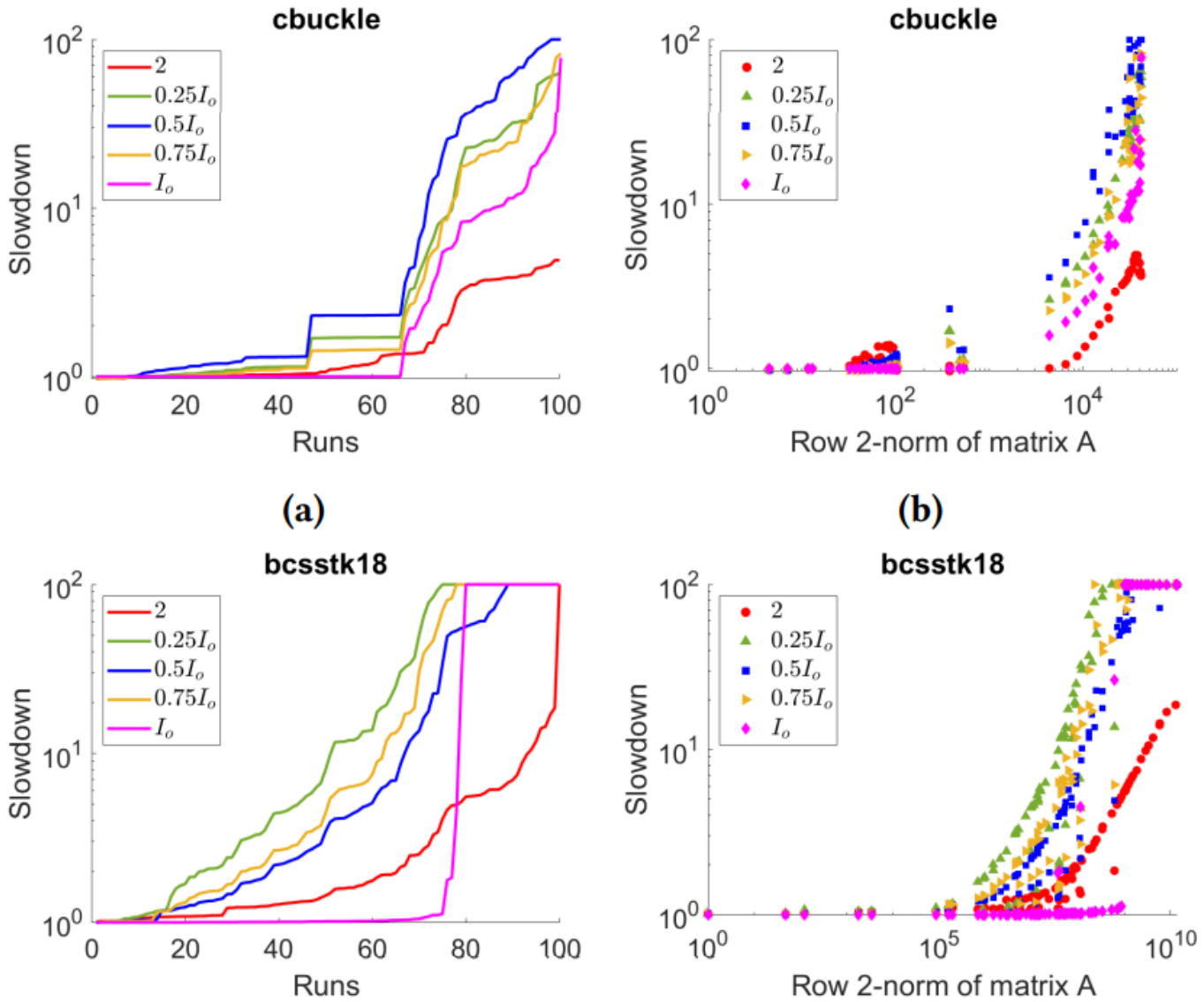
$$slowdown = \frac{I_e}{I_0}$$

Observation: not all errors are equal w.r.t. performance degradation





Profiling of soft error impact (injecting errors in 100 random locations in 5 different iterations)



Impact varies drastically:

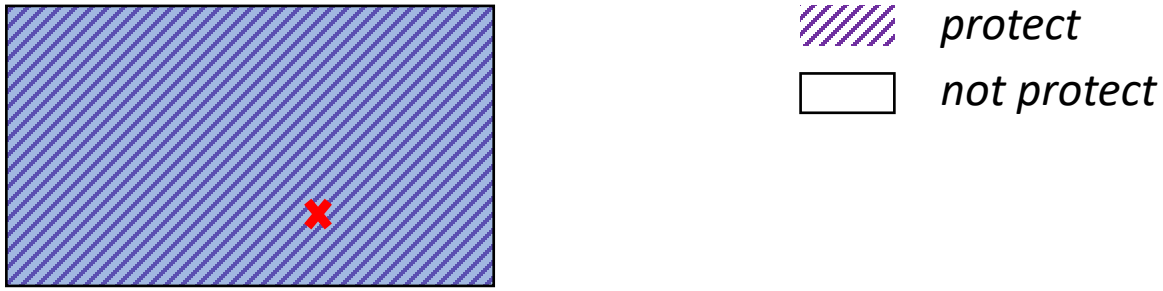
- Certain iterations (e.g., *middle ones*) suffer larger slowdowns than others;
- Location-wise, slowdowns are strongly correlated with *row 2-norm* of matrix:

$$\|A_{i*}\|_2 = \sqrt{\sum_j A_{i,j}^2}$$

Which elements to protect?

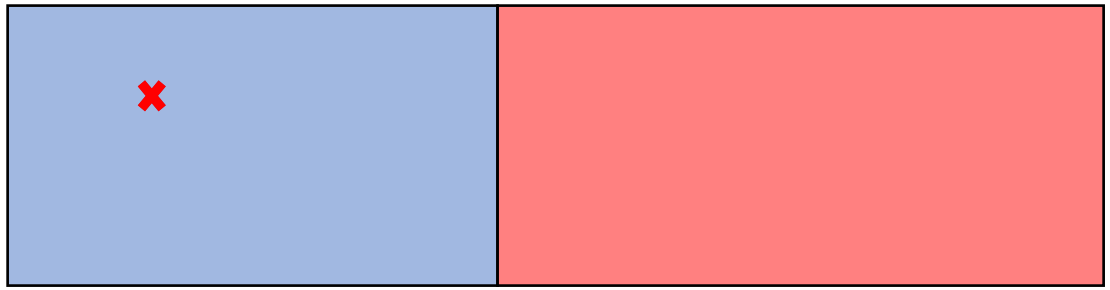
- **Full-protection:**

100% overhead (i.e., duplicate entire computation)



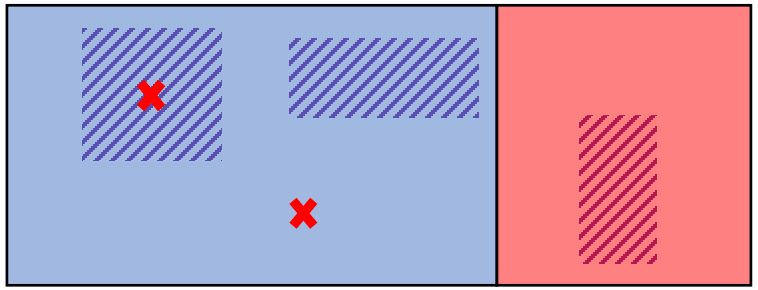
- **Zero-protection:**

Average overhead depends on expected slowdown (can be large)



- **Selective-protection:**

Protect only elements with *slowdown≥2* (i.e., *≥100% overhead*)



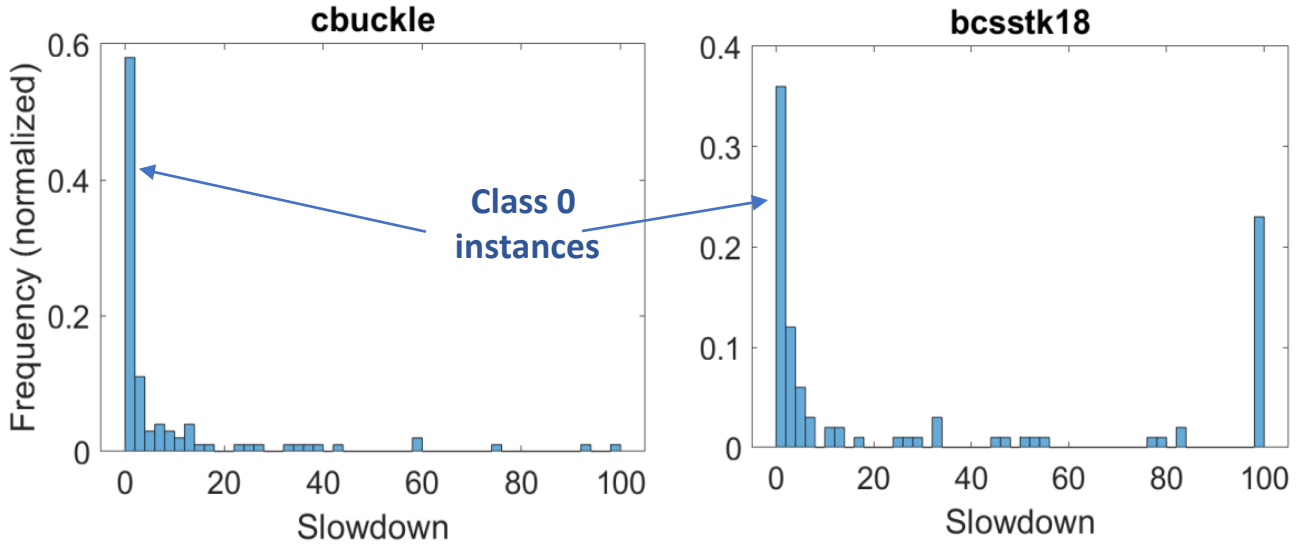
Predicting Error Impact via ML

Binary classification:

- **Class 1**: slowdown ≥ 2
- **Class 0**: slowdown < 2

with two features:

- Iteration number
- Row-2 norm of matrix *A*



	"cbuckle"				*"bcsstk18"*			
	Acc.	Recall	Prec.	F1	Acc.	Recall	Prec.	F1
LR	0.91	0.8	0.97	0.88	0.57	1	0.57	0.73
SVM	0.92	0.8	1	0.89	0.57	1	0.57	0.73
RF	0.93	0.9	0.93	0.91	0.8	0.9	0.78	0.84
KNN	0.94	0.95	0.91	0.93	0.76	0.84	0.76	0.80

LR: Logistic Regression
SVM: Support Vector Machine
RF: random Forest
KNN: K-Nearest Neighbor

* Total 300 runs (one random error per run): 200 for training and 100 for testing (results reported)

Our selective protection scheme based on **dynamic-ml** achieves the **lowest** average overhead

$$overhead = \frac{\sum_{t=1}^{I_e}(N + n_t) - I_o N}{I_o N}$$

Static-rand:

Protect certain percentage of *randomly selected* elements

Static-r2n:

Protect certain percentage of selected elements with *larger row 2-norms in matrix A*

	"cbuckle"	*"bcsstk18"*
zero-protection	816%	3143%
full-protection	100%	100%
static-rand (best)	91.4% (89%)	99% (99%)
static-r2n (best)	56.3% (30%)	88.4% (88%)
dynamic-ml	**43.7%**	**87.6%**

* Average **overhead** (in %) compared to error-free run

Future Work

- **Improve prediction accuracy** with more features and ML algorithms:
  - ✓ **For classification** (tune threshold considering error prob.)
  - ✓ **For regression** (i.e., predict exact slowdown)

- **Train a single ML model** that captures sparse matrix structures (currently, one model per matrix)

- **Further validation** with more matrices and solvers