PuDGhost

Experimental Analysis of Computation Result Corruption in Processing-using-DRAM Operations on Real DRAM Chips and Implications for Future Systems

Daichi Tokuda

I. E. Yuksel T. Kubo A. Olgun H. Luo

F. N. Bostanci J. Wang A. G. Yağlıkçı

S. Takamaeda-Yamazaki O. Mutlu

Executive Summary

Motivation: Processing-using-DRAM (PuD) alleviates data movement bottlenecks by performing computation within DRAM

- PuD activates multiple DRAM rows simultaneously (SiMRA) to compute in parallel across columns
- Ideally, each column's result is determined *solely* by the activated cells used as operands

Problem: No prior work studies how data *not* intended to participate in PuD operations (non-operand data) affects PuD reliability

Experimental Study: 96 off-the-shelf DDR4 chips to study *PuDGhost*, interference from non-operand data that can corrupt PuD operations

- Non-activated rows affect SiMRA outputs by up to 10%
- Concurrently computing columns affects SiMRA outputs by up to 48%

Mitigation: We propose robust column screenings that account for PuDGhost Our solutions provide 413× lower GEMV NMSE vs. PuDGhost-unaware systems

Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

Mitigating PuDGhost

Conclusion

Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

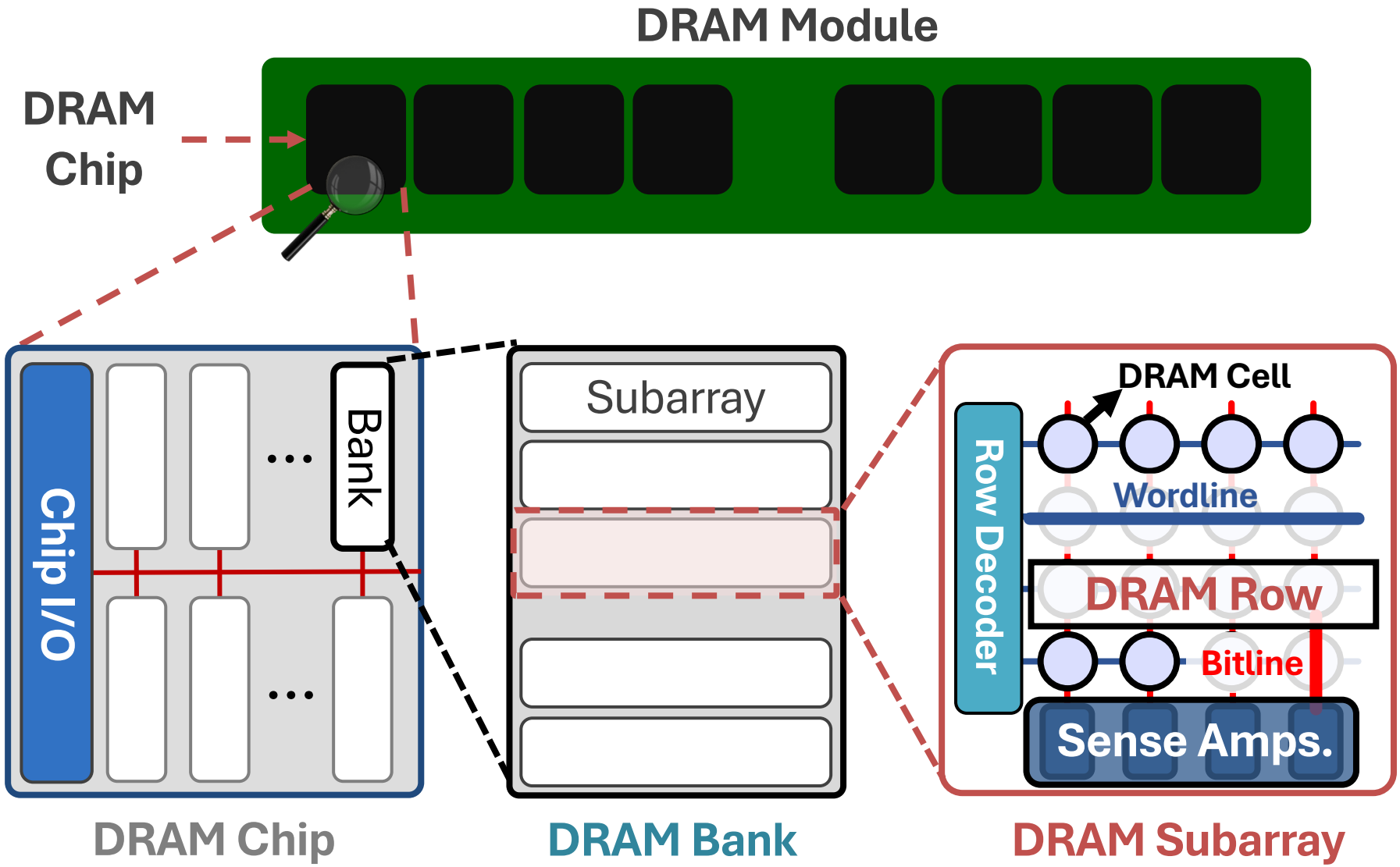
Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

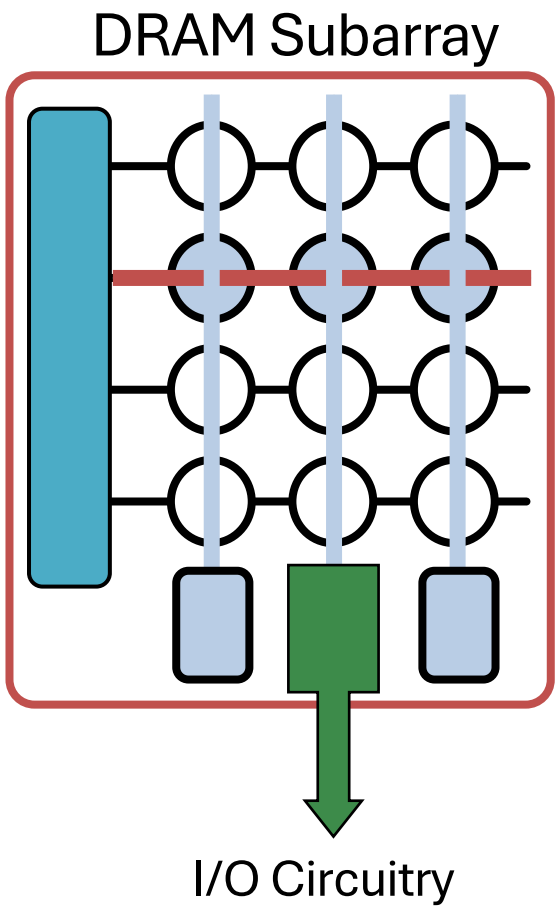
Mitigating PuDGhost

Conclusion

DRAM Organization



DRAM Operation



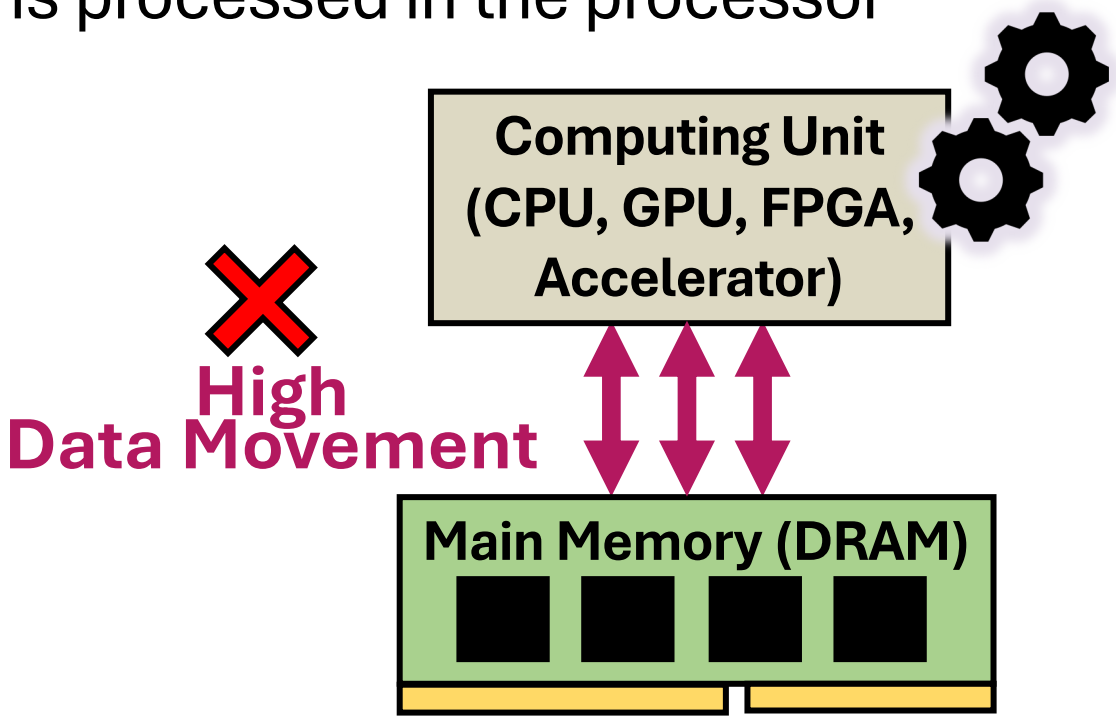
1 **ACTIVATE (ACT):**
Fetch the row's content into the **sense amplifiers**

2 **Column Access (RD/WR):**
Read/Write the target column and drive to I/O

3 **PRECHARGE (PRE):**
Prepare the bank for a new ACTIVATE

Data Movement Bottleneck

- Today's computing systems are **processor centric**
- All data is processed in the processor

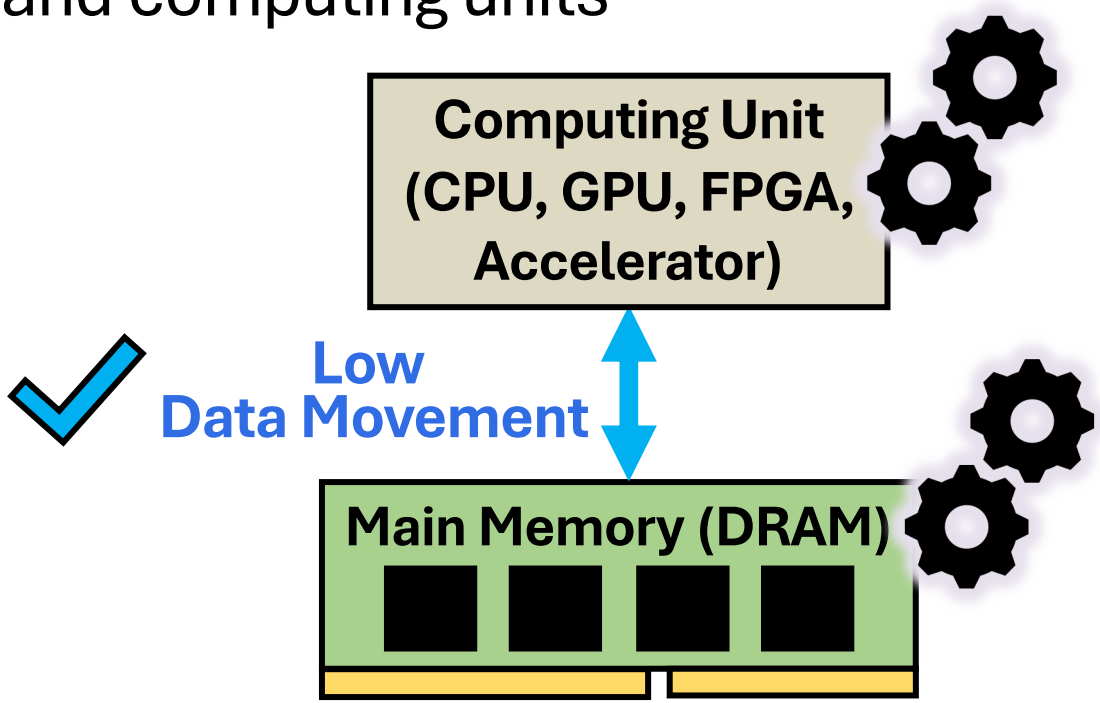


More than 60% of the total system energy is spent on data movement[1]

1 A. Boroumand et al., "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS, 2018

Processing-using-DRAM (PuD)

- Processing-using-DRAM (PuD) uses the analog operational principles of DRAM cells to perform computation
- PuD alleviates the frequent data movement between main memory and computing units

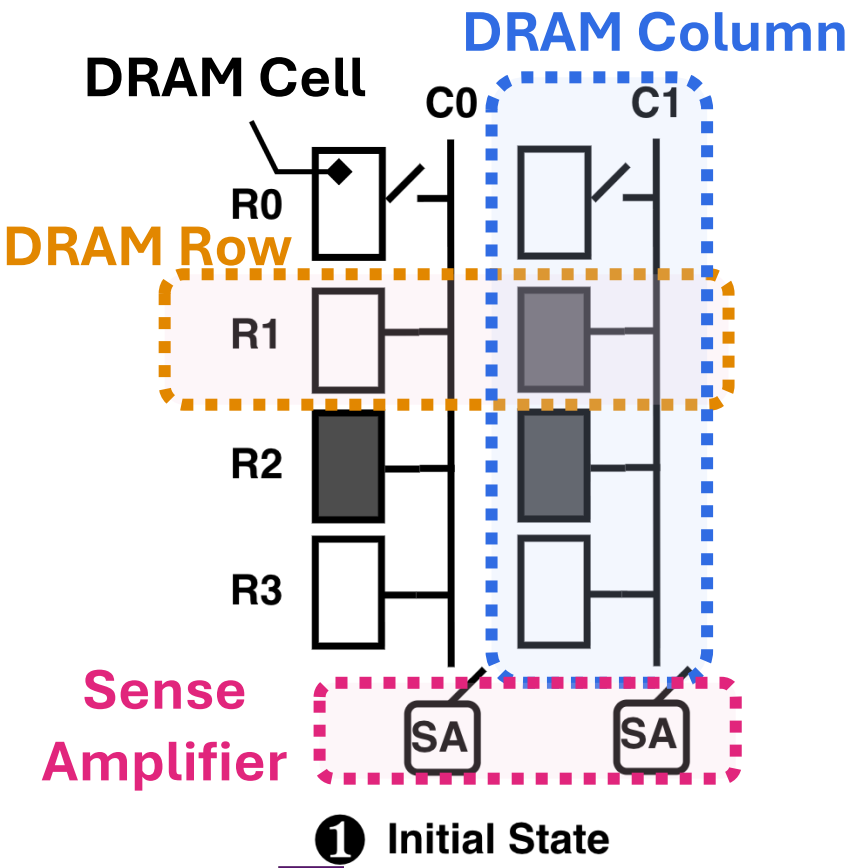


Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)

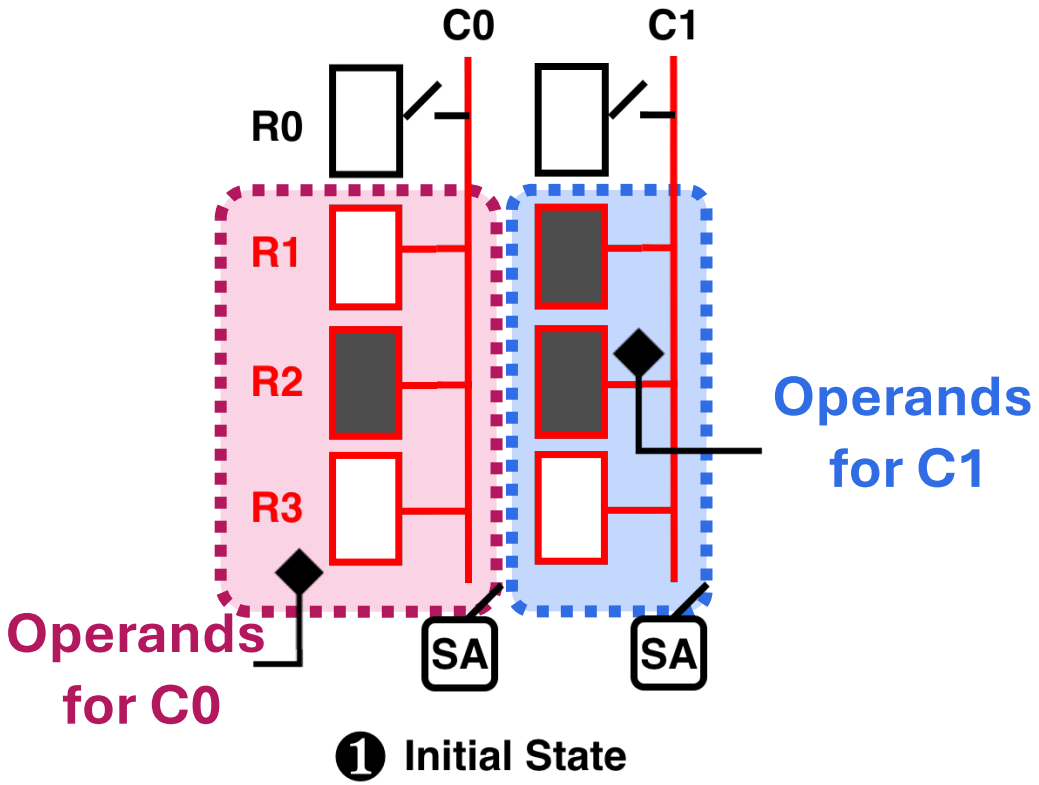
Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



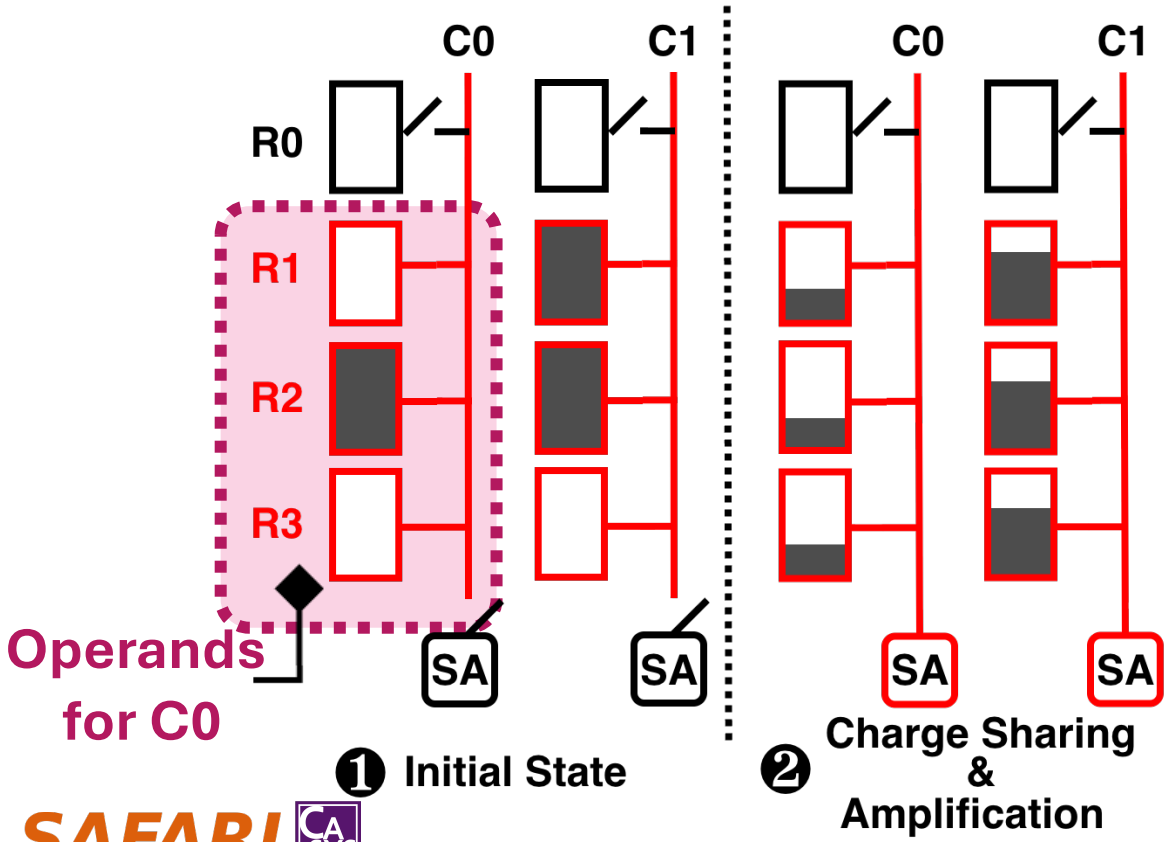
Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



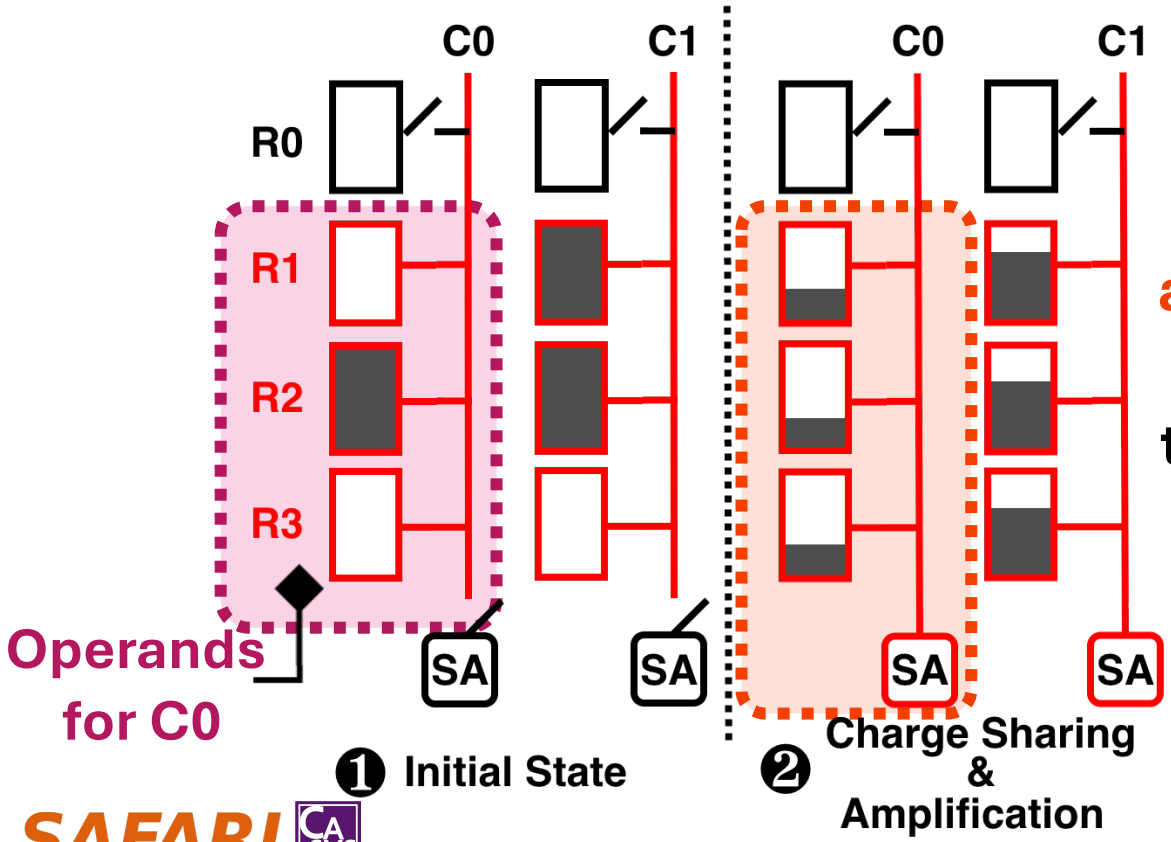
Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



Simultaneous Multiple-Row Activation (SiMRA)

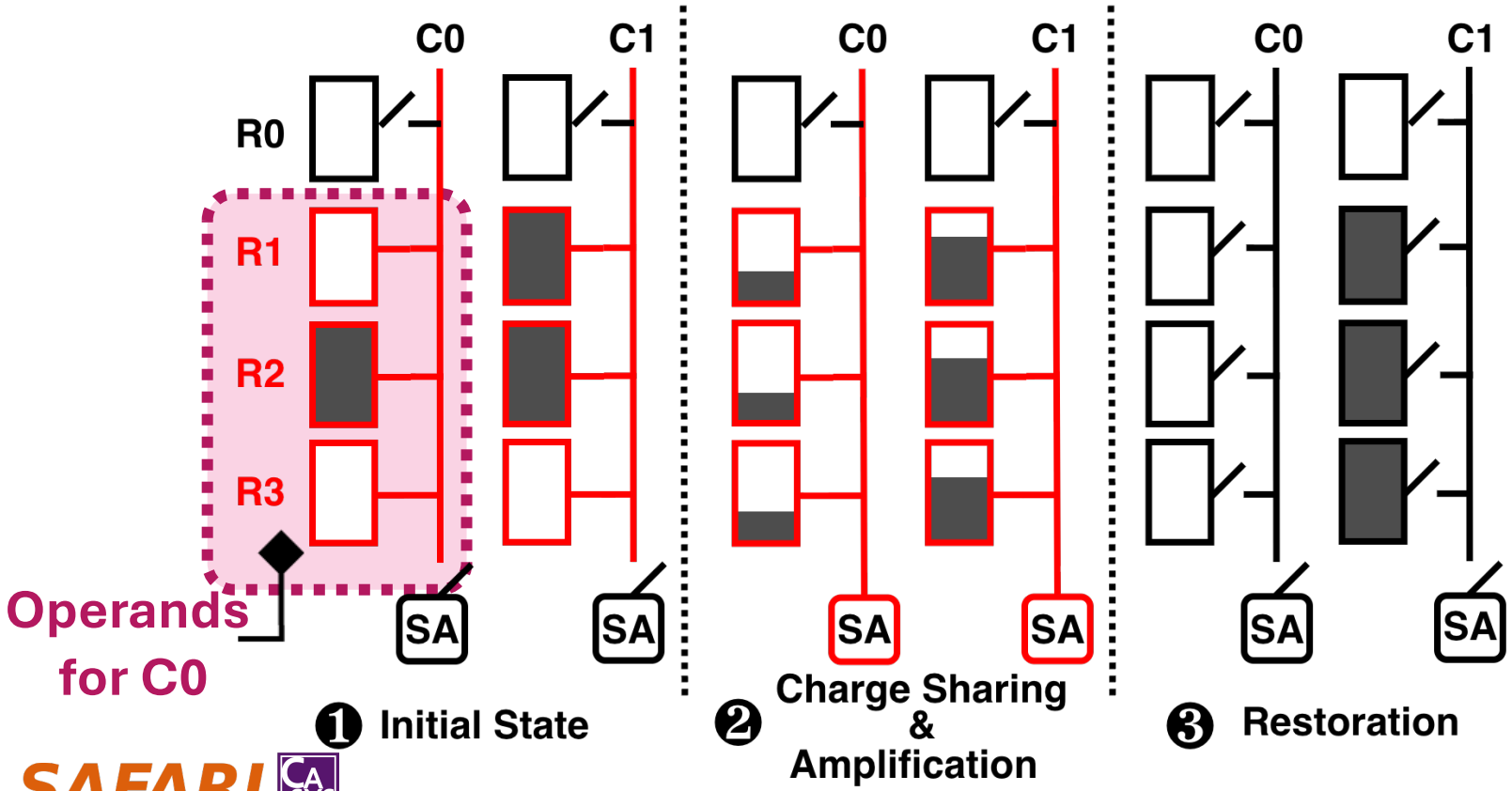
- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



Enabling charge sharing across the activated cells in each column through the shared bitline

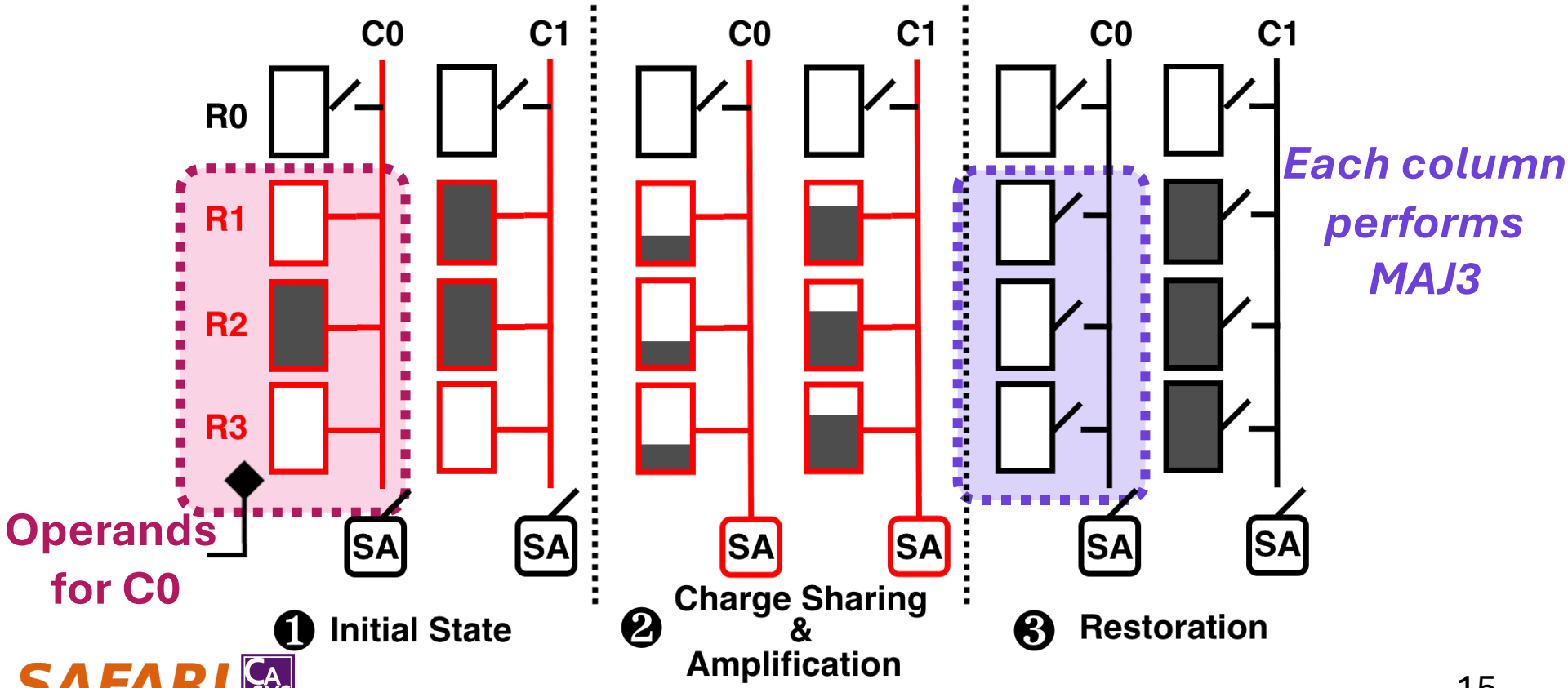
Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



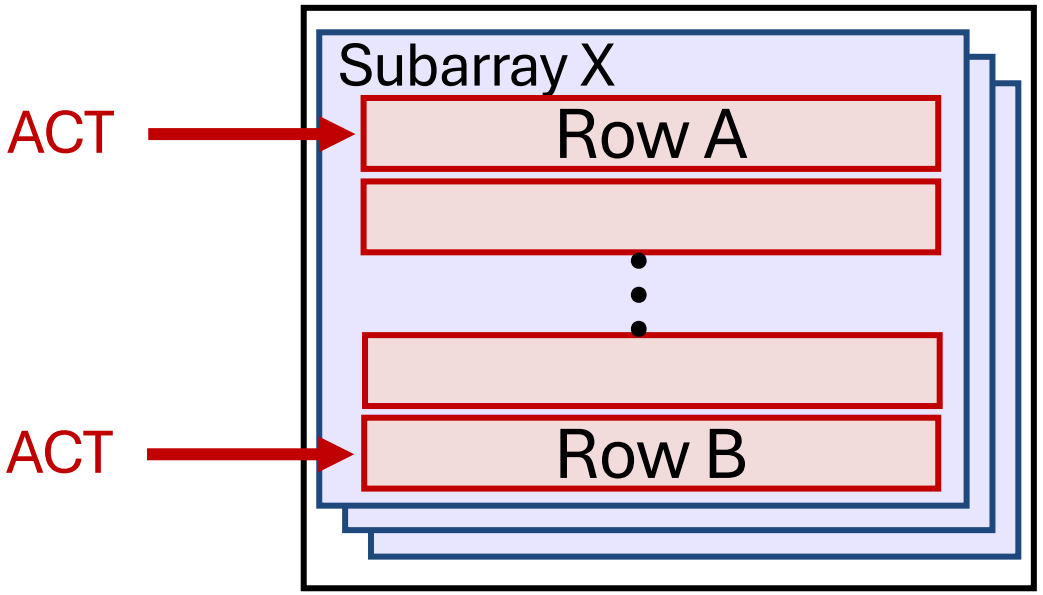
Simultaneous Multiple-Row Activation (SiMRA)

- Simultaneously activating X rows in a subarray enables a majority-of-X operation (MAJX)



SiMRA in COTS DRAM Chips

Activating two rows in **quick succession** in **COTS** DRAM chips **simultaneously** activates **multiple rows** in a subarray



DRAM Bank

Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

Mitigating PuDGhost

Conclusion

Problem

A Key Expectation of PuD:

Each column's SiMRA result depends ***solely*** **on its own operand data**

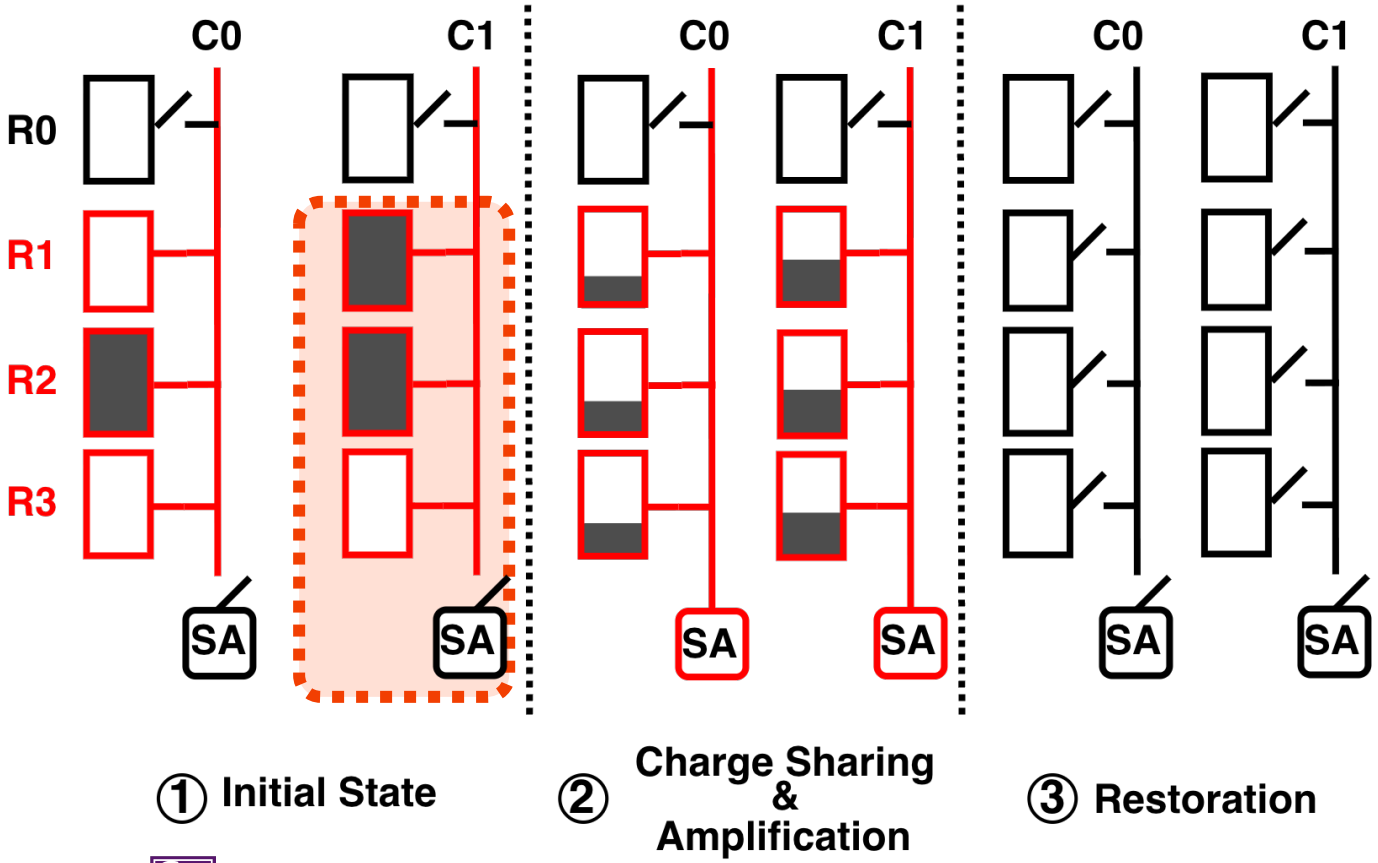
Our Hypothesis:

This key expectation of PuD might be **challenged by ongoing DRAM density scaling**

Does this key expectation hold on real DRAM chips?

Unfortunately, no.

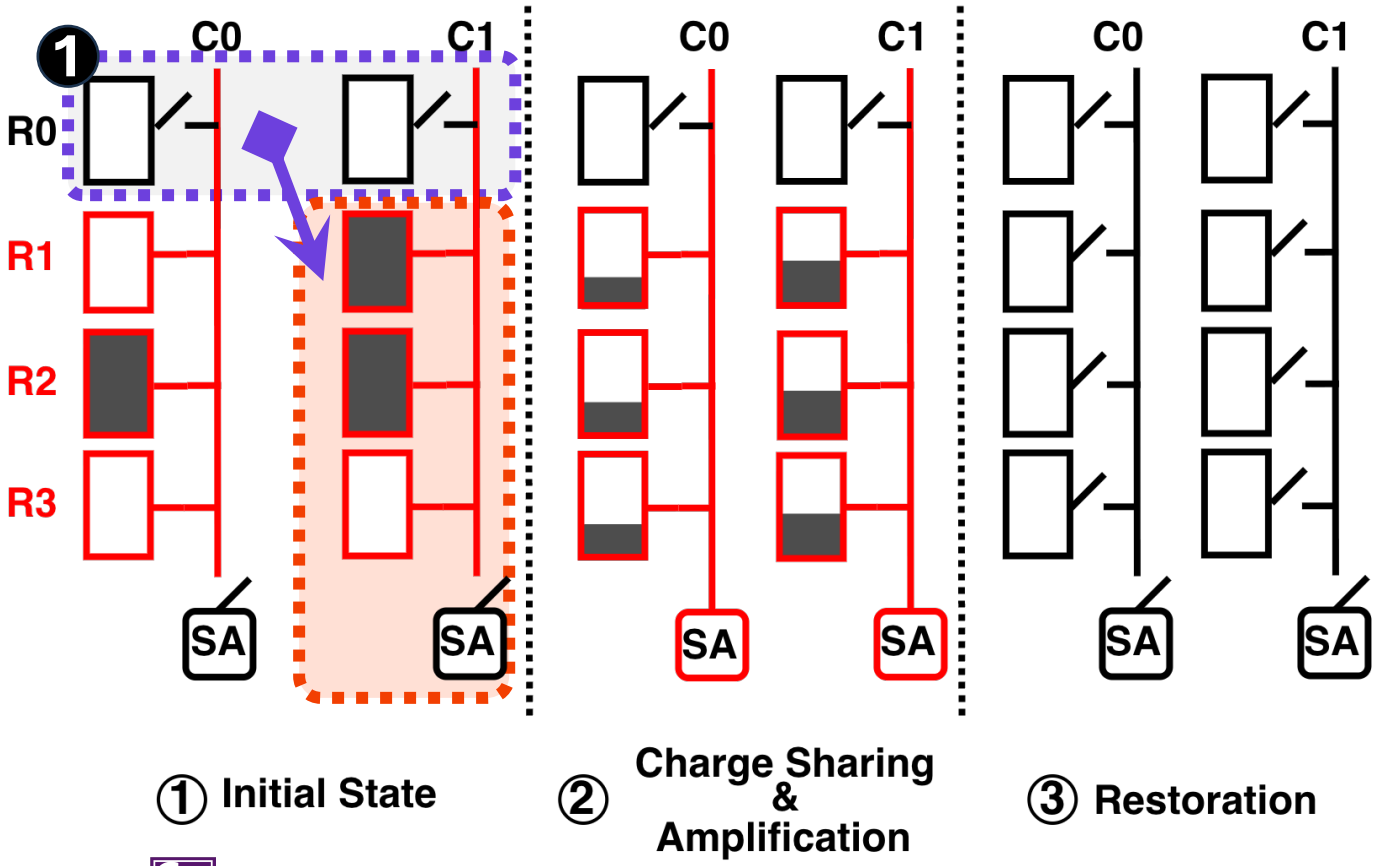
What is PuDGhost?



What is PuDGhost?

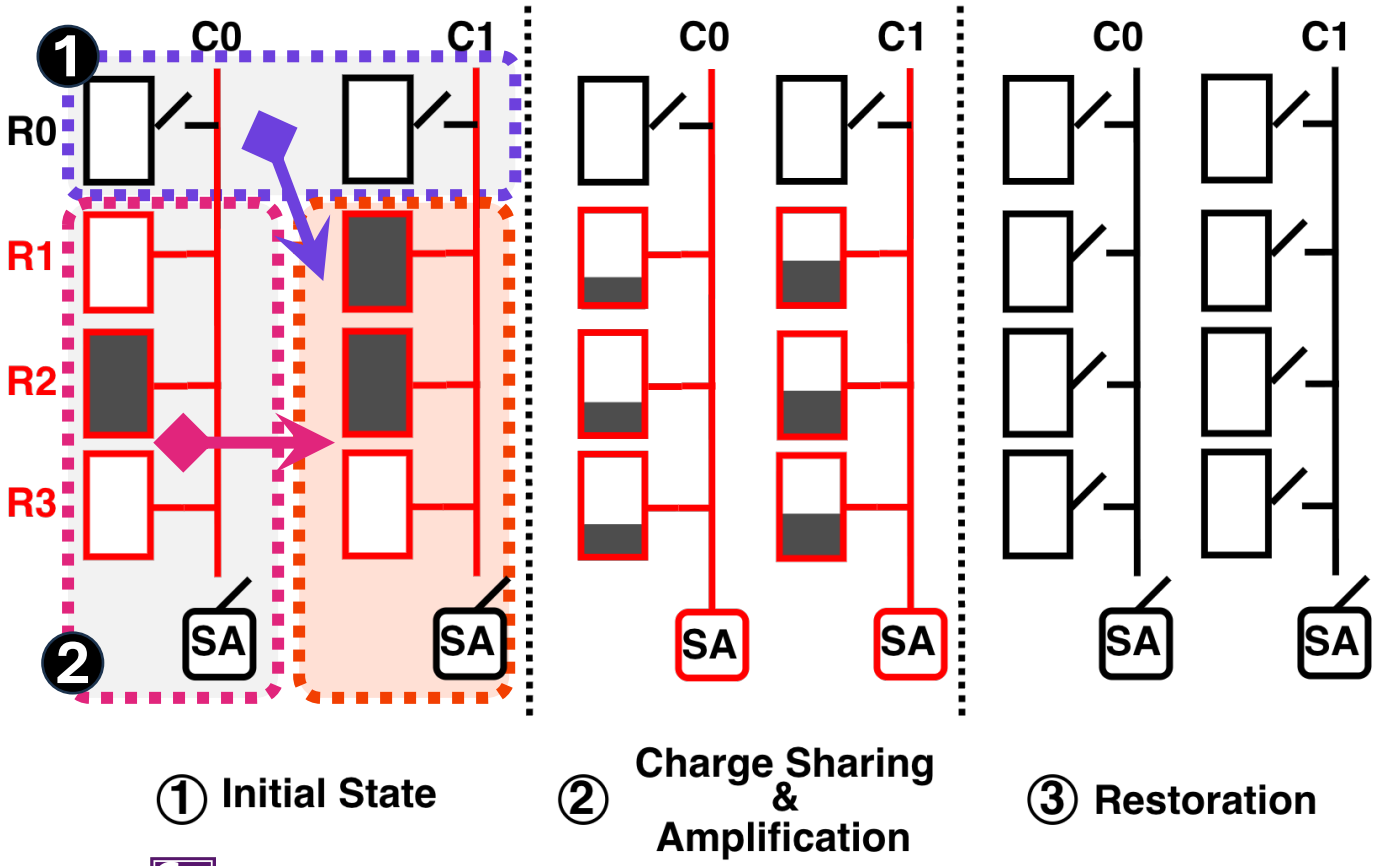
Interference from

❶ **Non-Activated Rows**

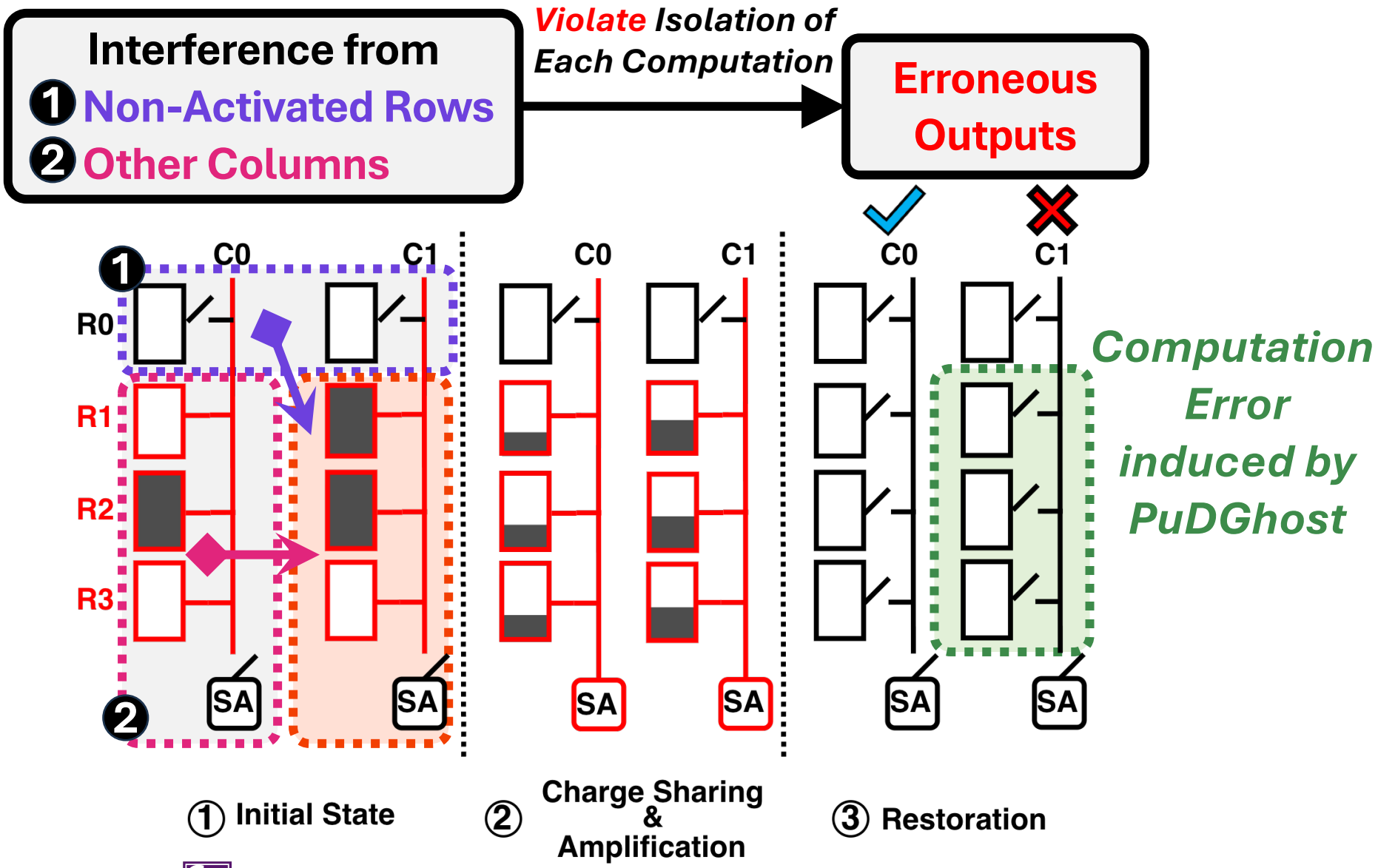


What is PuDGhost?

Interference from
1. **Non-Activated Rows**
2. **Other Columns**



What is PuDGhost?



Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

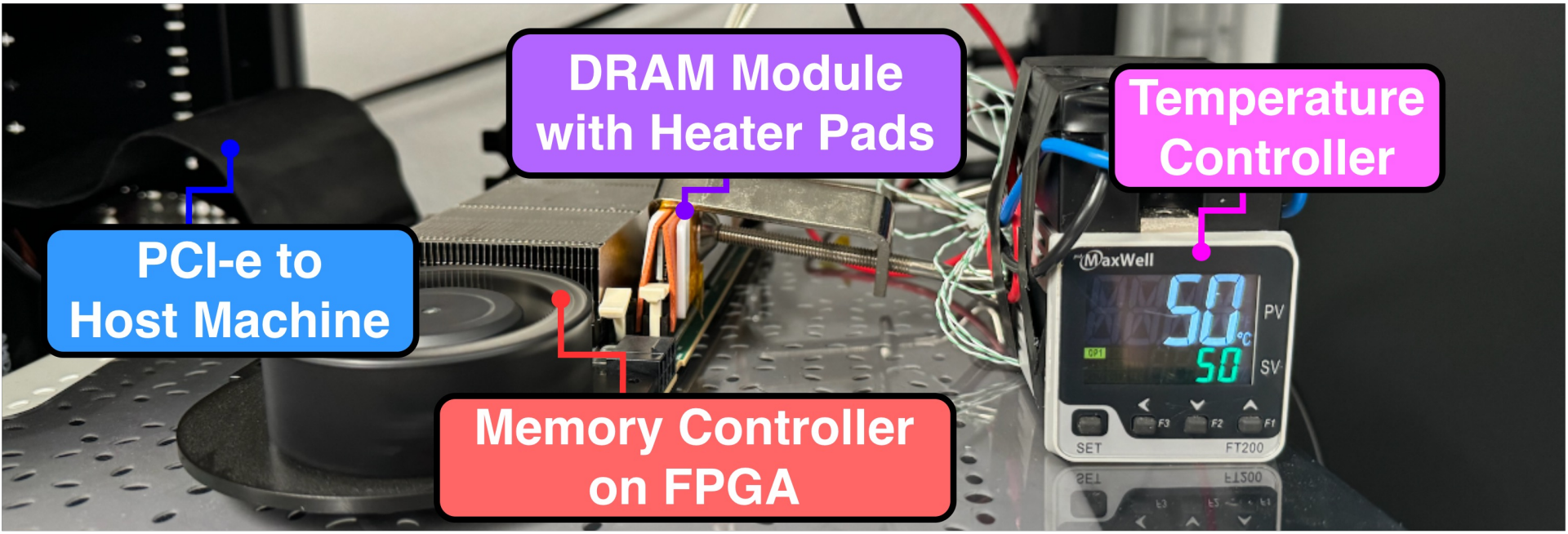
Interference from Concurrently Computing Columns

Mitigating PuDGhost

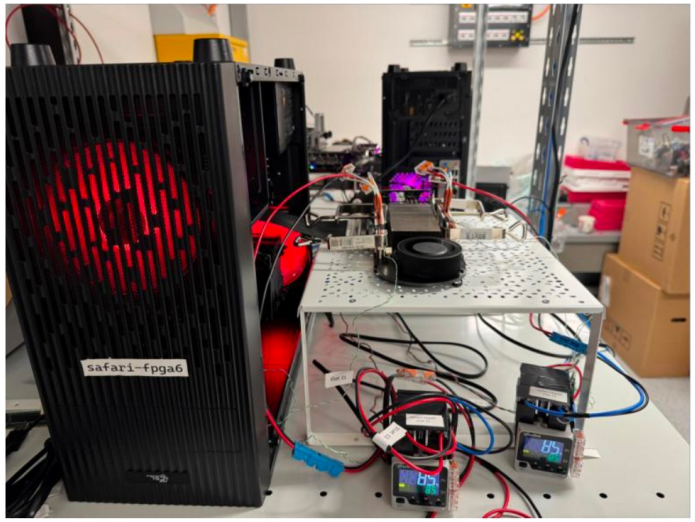
Conclusion

DRAM Bender: DRAM Testing Infrastructure

Fine-grained control over DRAM commands, timings, and temperature



Laboratory for Understanding Memory



DRAM Chips Tested

- **96 DDR4 chips** from SK Hynix
- Covering three different die-revision and chip-density pairs

Chip Mfr.	Module Mfr.	#Modules (#Chips)	Die Rev.	Chip Density	Chip Org.
SK Hynix	TimeTec	3 (24)	A	4Gb	x8
	TeamGroup	7 (56)	M	4Gb	x8
	SK Hynix	2 (16)	A	8Gb	x8

Characterization Methodology - I

- Number of Simultaneously Activated Rows:
  - 2 / 4 / 8 / 16 / 32 Rows
- Tested Instances:
  - We select **three subarrays** for each bank and each module

How to Quantify Interference from Non-Operand Data:

- We vary *data not intended to participate* in SiMRA
- We then measure the resulting change in SiMRA outputs

Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

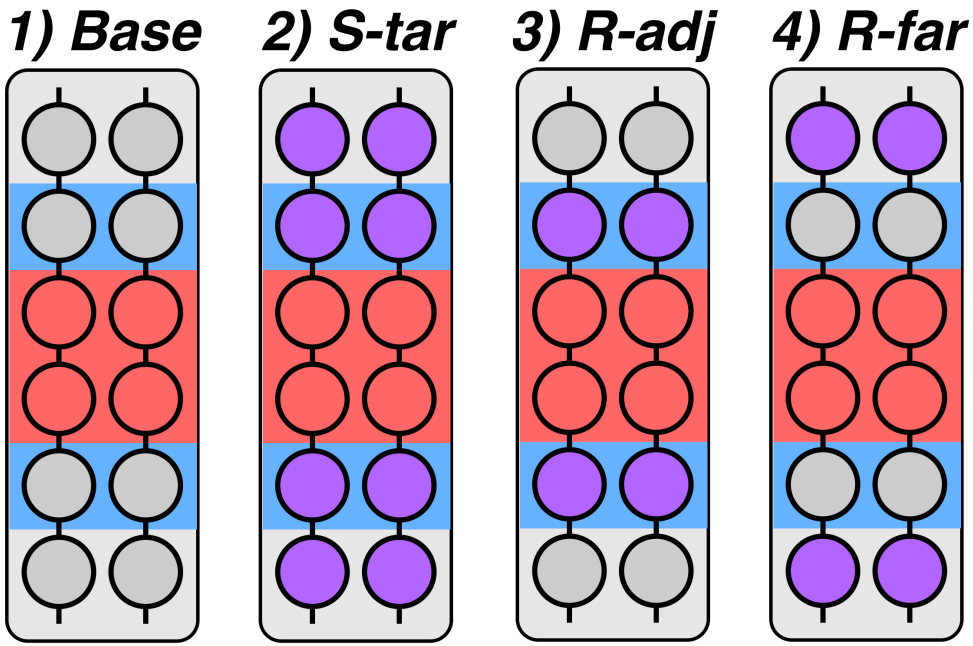
Interference from Concurrently Computing Columns

Mitigating PuDGhost

Conclusion

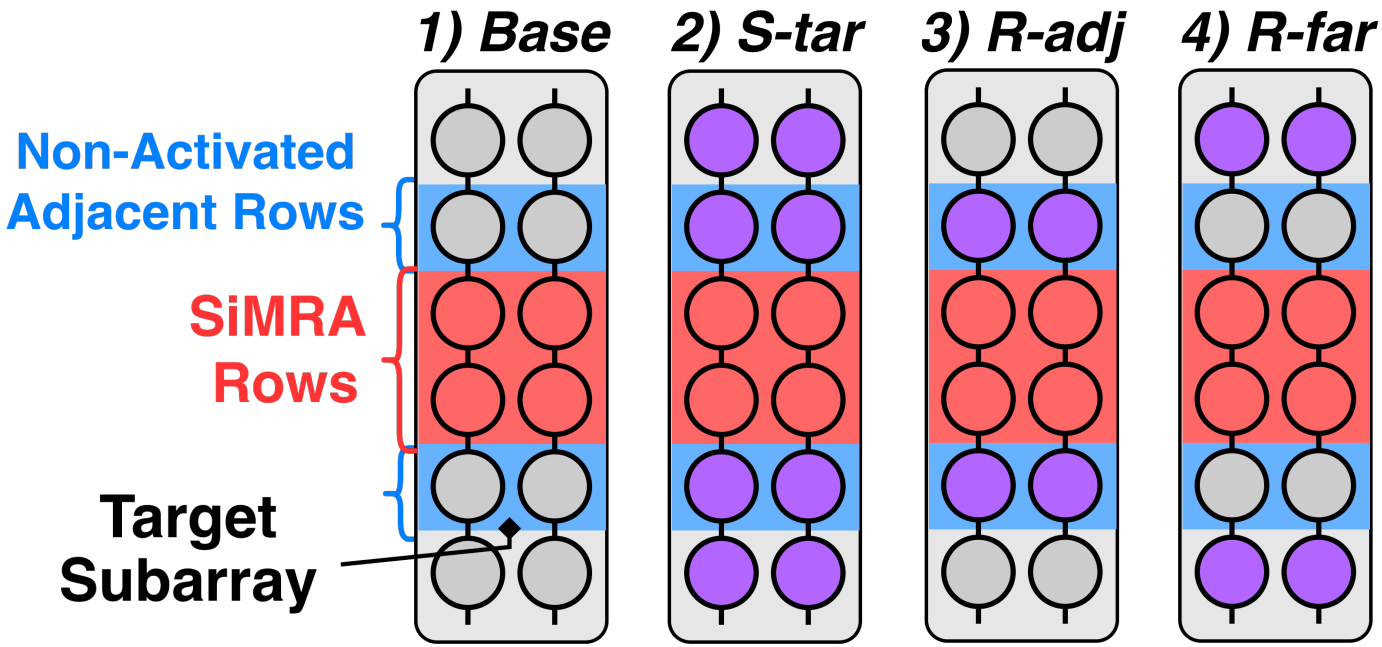
Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- **Controlled cells**: cells whose data pattern is intentionally varied to observe how SiMRA outputs change
- We compare four configurations



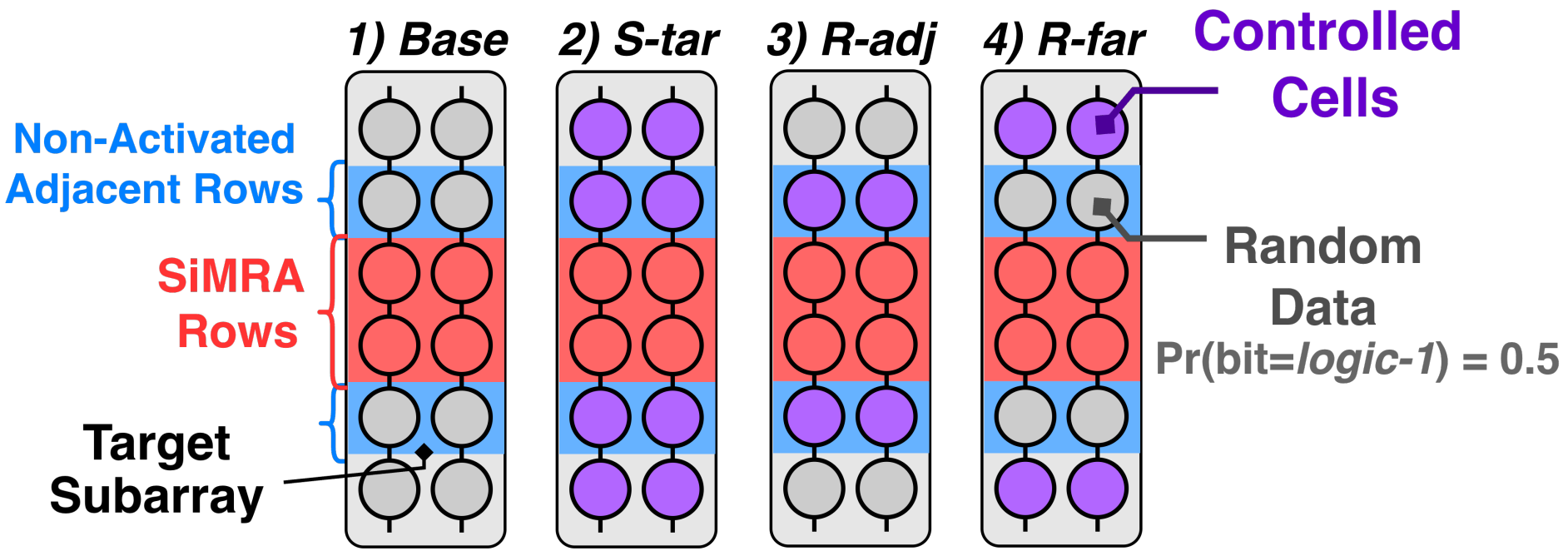
Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- **Controlled cells**: cells whose data pattern is intentionally varied to observe how SiMRA outputs change
- We compare four configurations



Interference from Non-Activated Rows

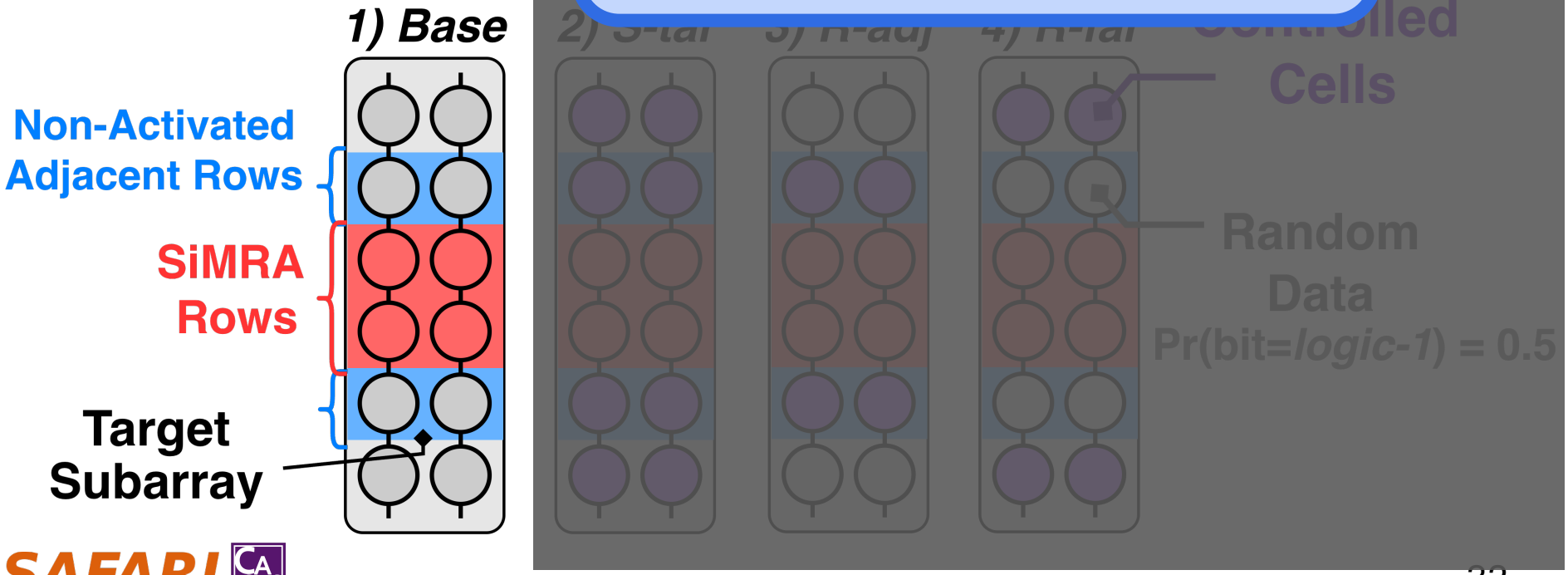
- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- **Controlled cells**: cells whose data pattern is intentionally varied to observe how SiMRA outputs change
- We compare four configurations



Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- **Controlled cells**: cells whose data pattern is intentionally varied to observe how SiMRA
- We compare four confi

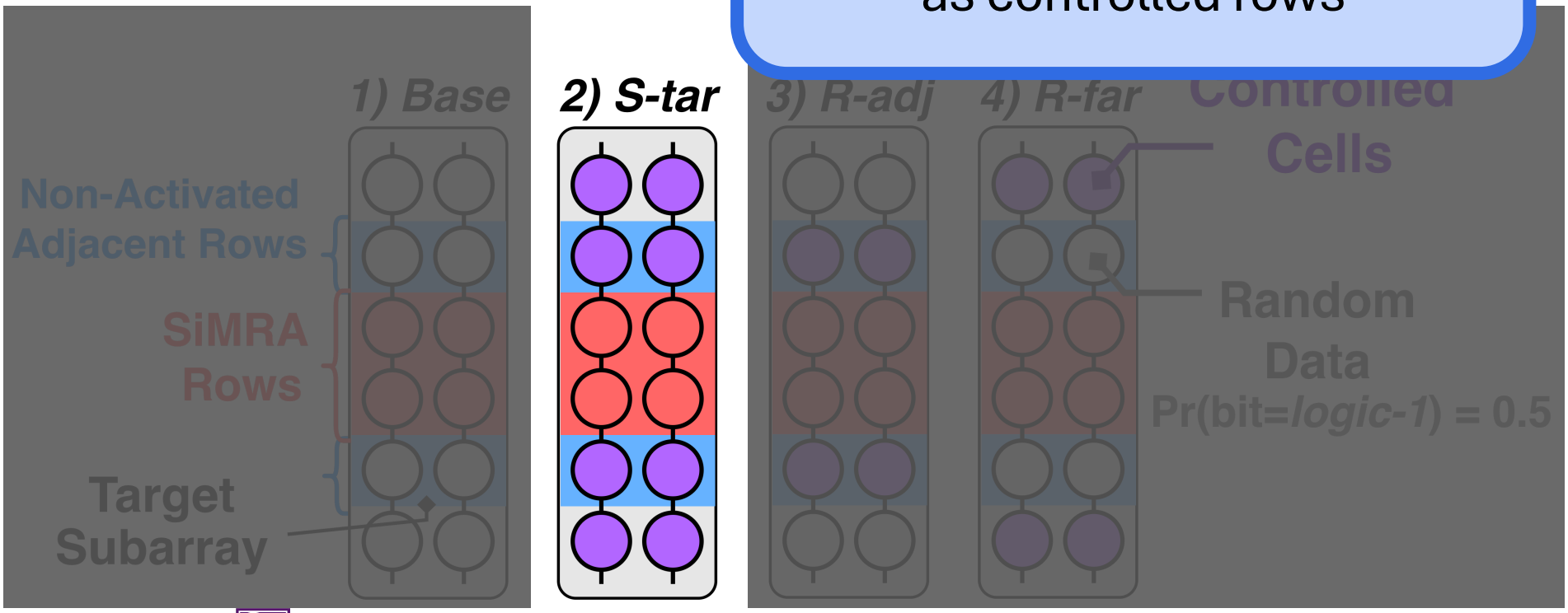
1) Base fills all rows with random data, (with no controlled cells)



Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- **Controlled cells**: cells whose
to observe how SiMRA output
- We compare four configuratio

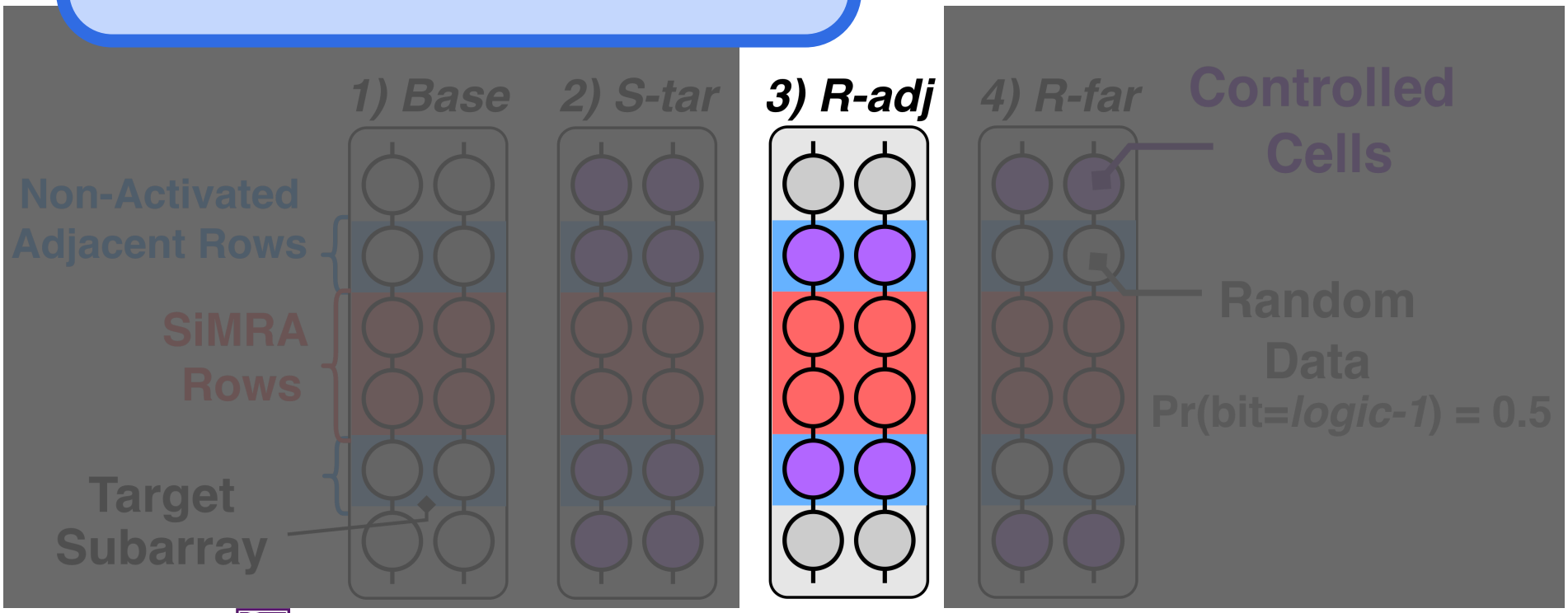
2) S-tar designates all rows except the SiMRA rows as controlled rows



Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that equal logic-1 when SiMRA is executed with random inputs
- pattern is intentionally varied
- nge
- as controlled rows

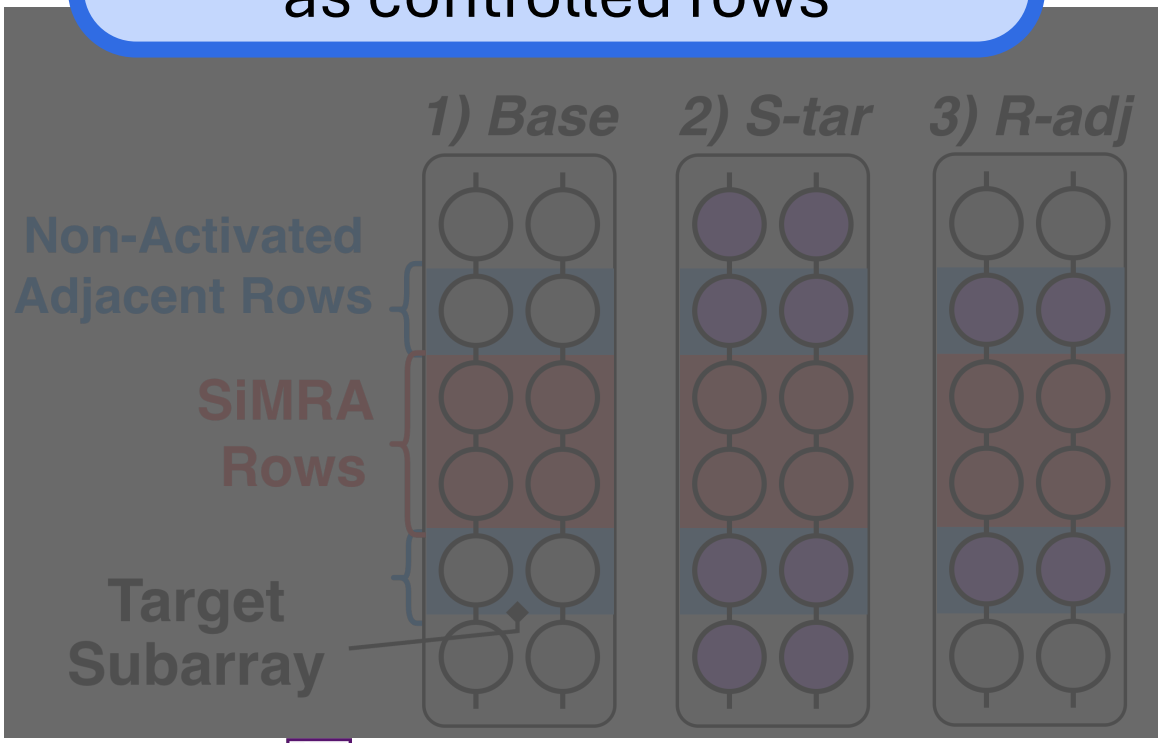
3) R-adj designates all non-activated adjacent rows

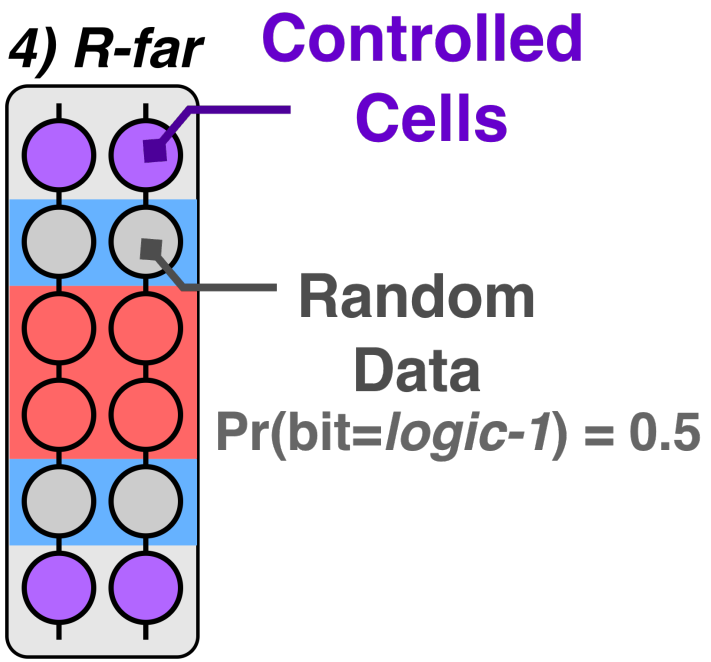


Interference from Non-Activated Rows

- Main metric (p_{o1}): fraction of output bits across all columns that
- pattern is intentionally varied
- nge
- as controlled rows

4) R-far designates all rows *except* the SiMRA rows and the adjacent rows



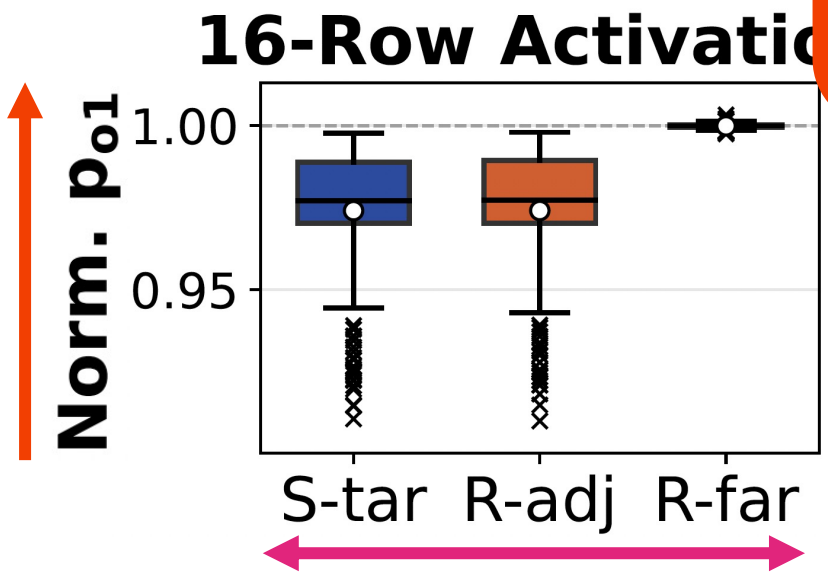


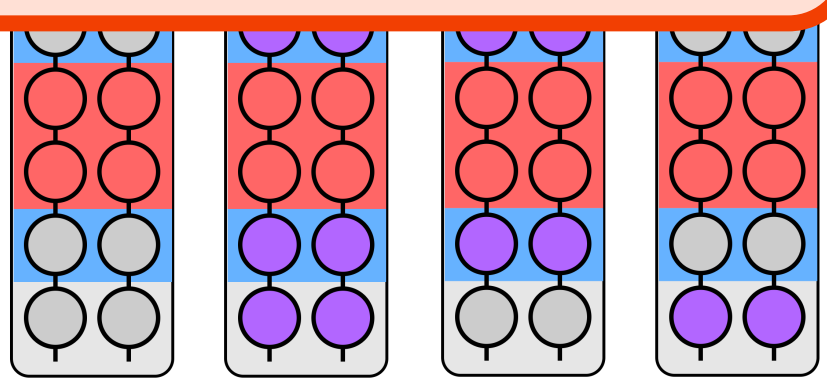
Results: Non-Activated Rows - I

Results: Controlled cells store **all zeros**

p_{o1} under each configuration normalized to Base

Recall: p_{o1} is the fraction of SiMRA output bits that equal logic-1

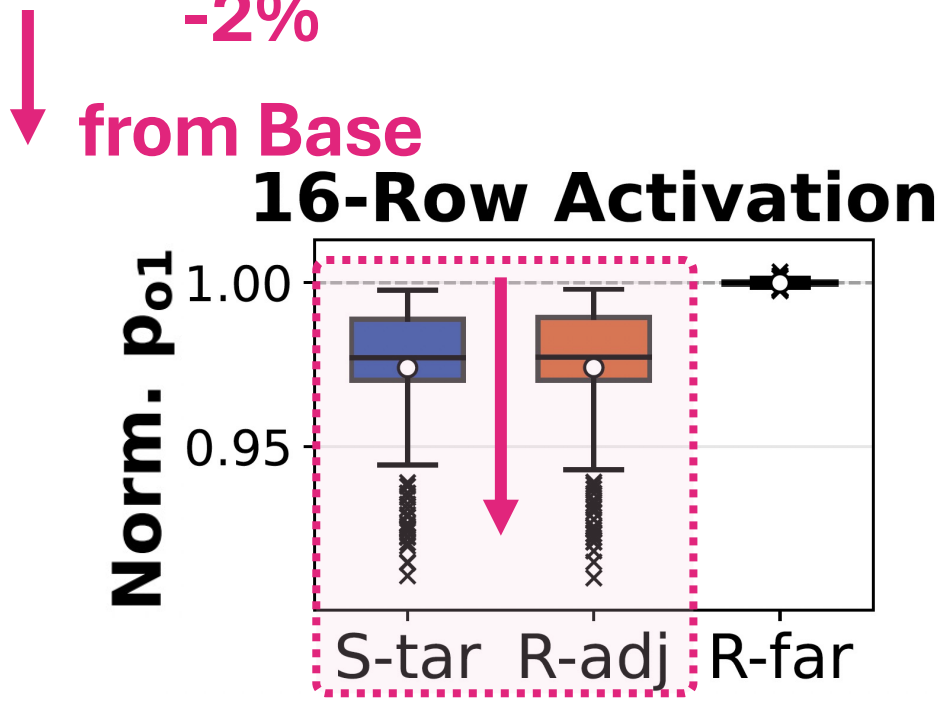


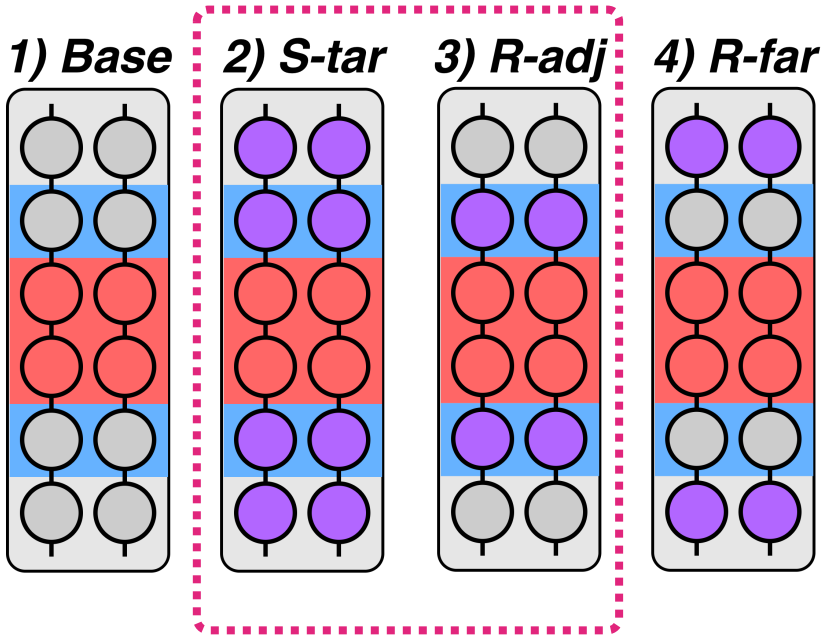


Three different non-baseline configurations

Results: Non-Activated Rows - I

Results: Controlled cells store **all zeros**





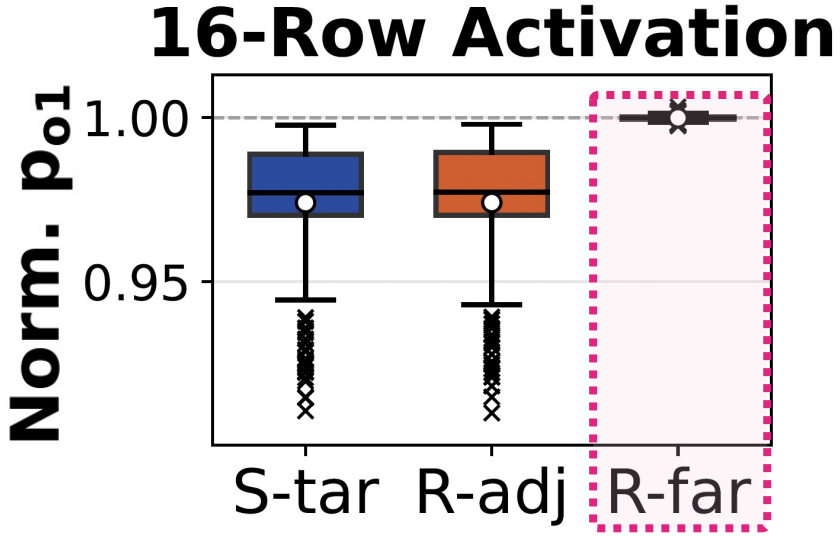
Non-activated rows bias SiMRA outputs **toward logic-0**
when the rows store **logic-0**
by **2%** on average and **up to 10%**

Results: Non-Activated Rows - I

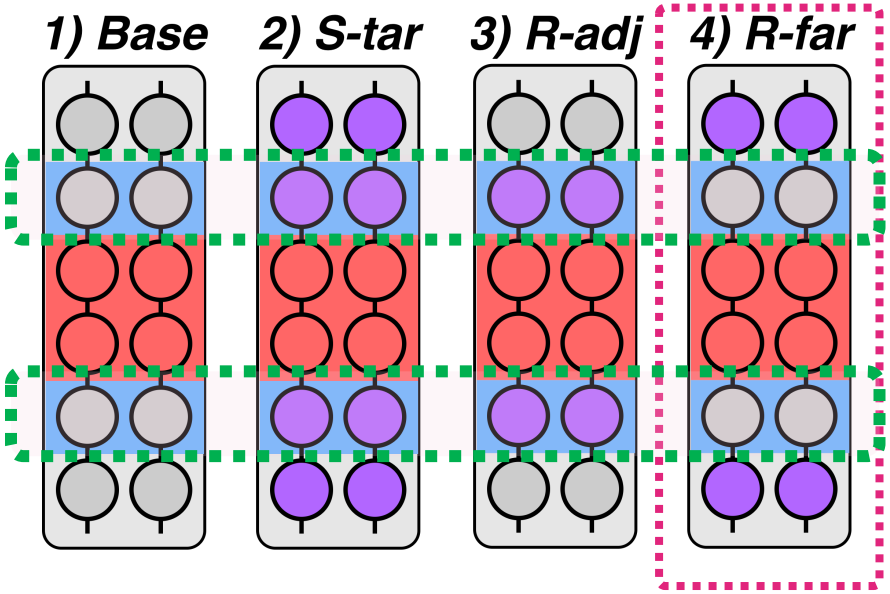
Results: Controlled cells store **all zeros**

±~0 %

from Base



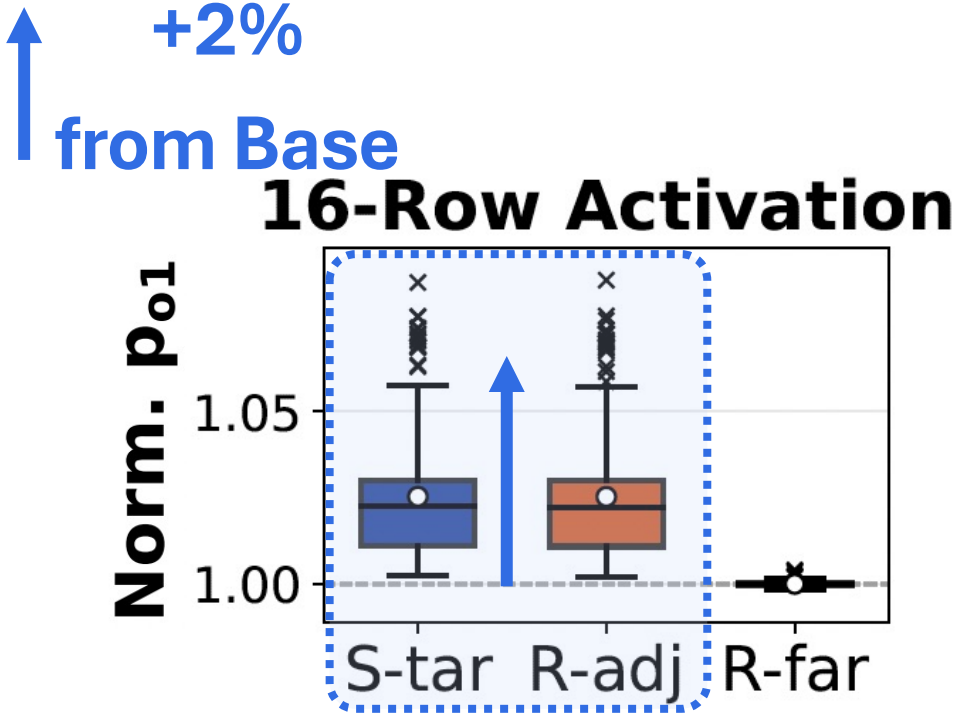
Non-Activated Adjacent Rows

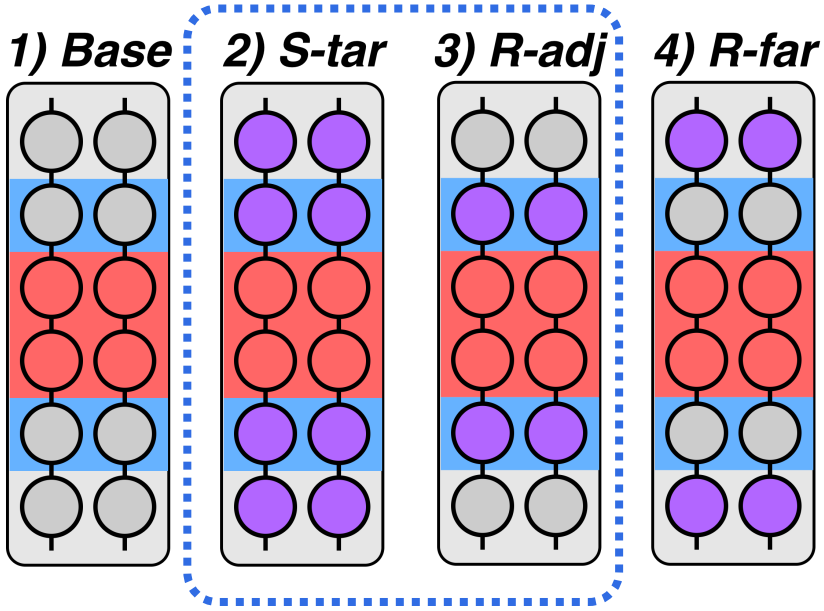


Interference from non-activated rows is
highly localized to physically adjacent rows

Results: Non-Activated Rows - II

Results: Controlled cells store **all ones**





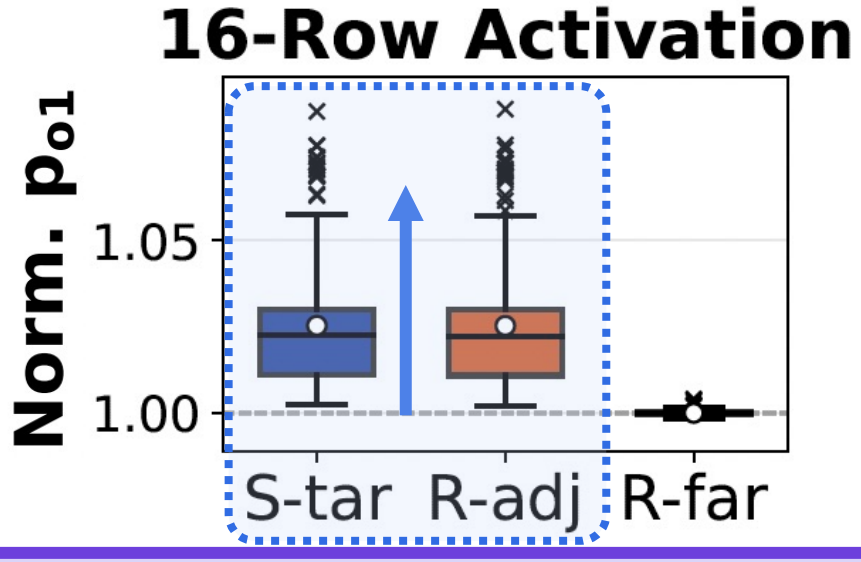
Non-activated rows bias SiMRA outputs **toward logic-1**
when the rows store **logic-1**
by **2%** on average and **up to 10%**

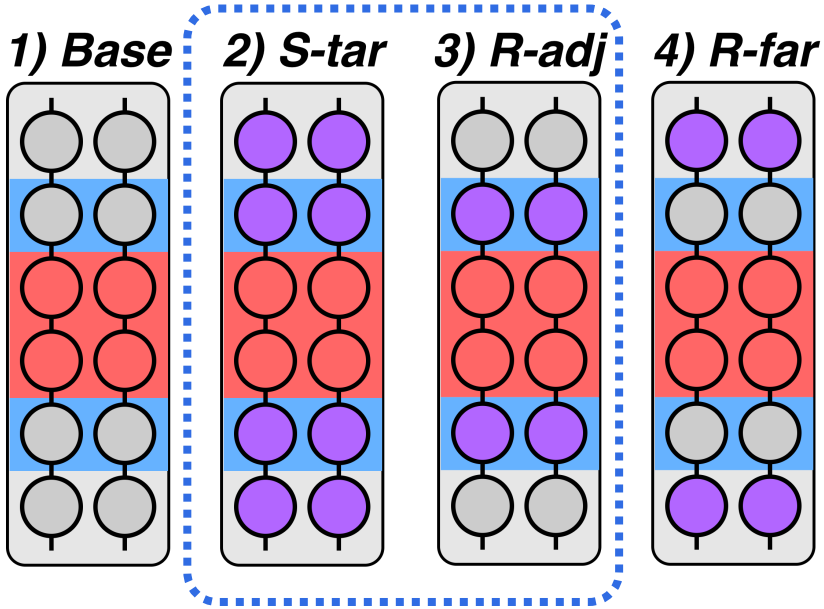
Results: Non-Activated Rows - II

Results: Controlled cells store **all ones**

+2%

from Base



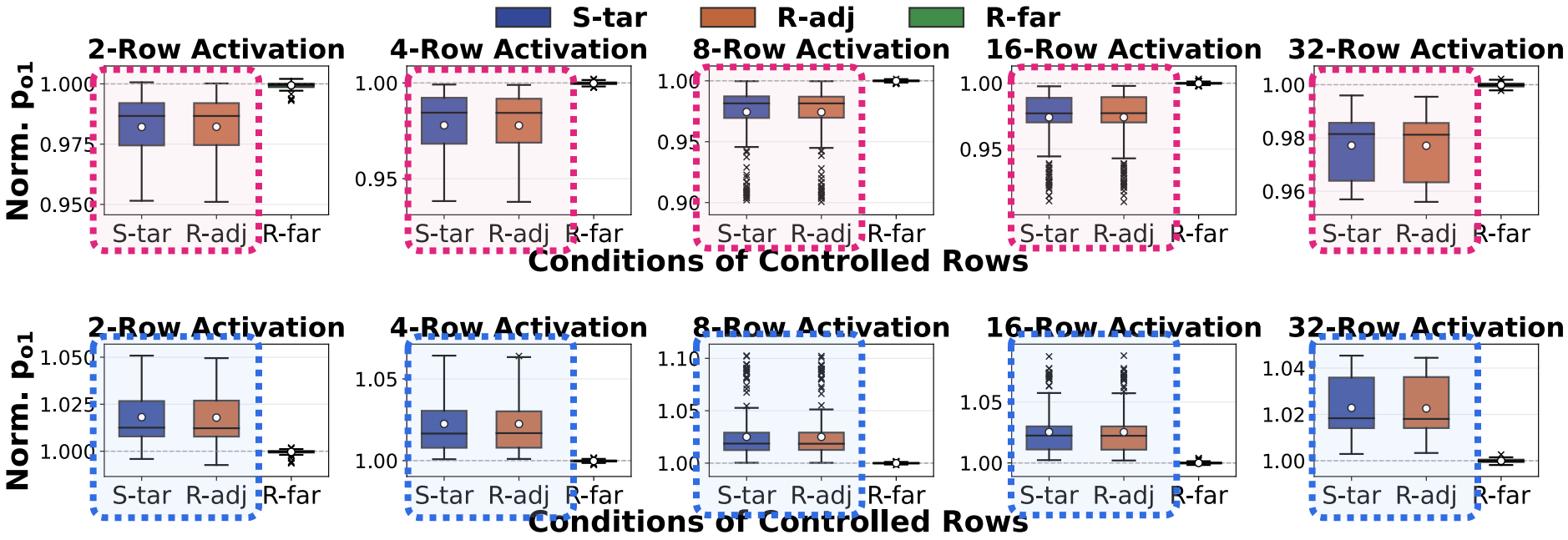


Key Takeaway

Non-activated adjacent rows bias SiMRA outputs toward logic-0 (logic-1) when the rows store logic-0 (logic-1)

Results: Non-Activated Rows - I & II

We consistently observe the similar trend across all tested SiMRA row counts



Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

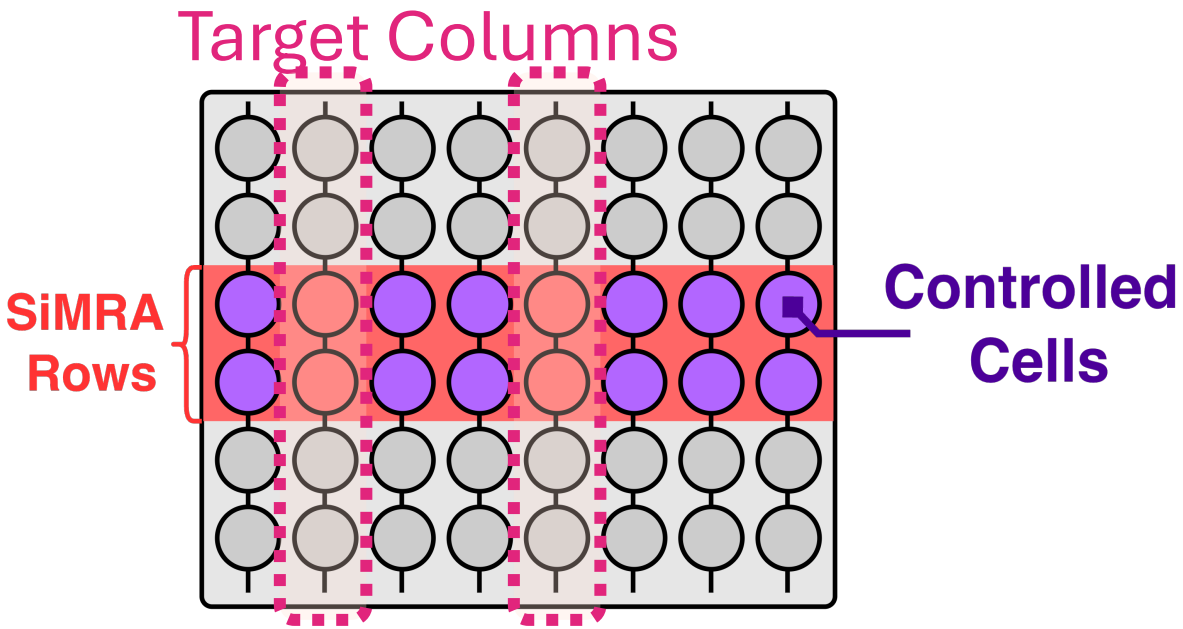
Mitigating PuDGhost

Conclusion

Interference from Concurrently Computing Columns

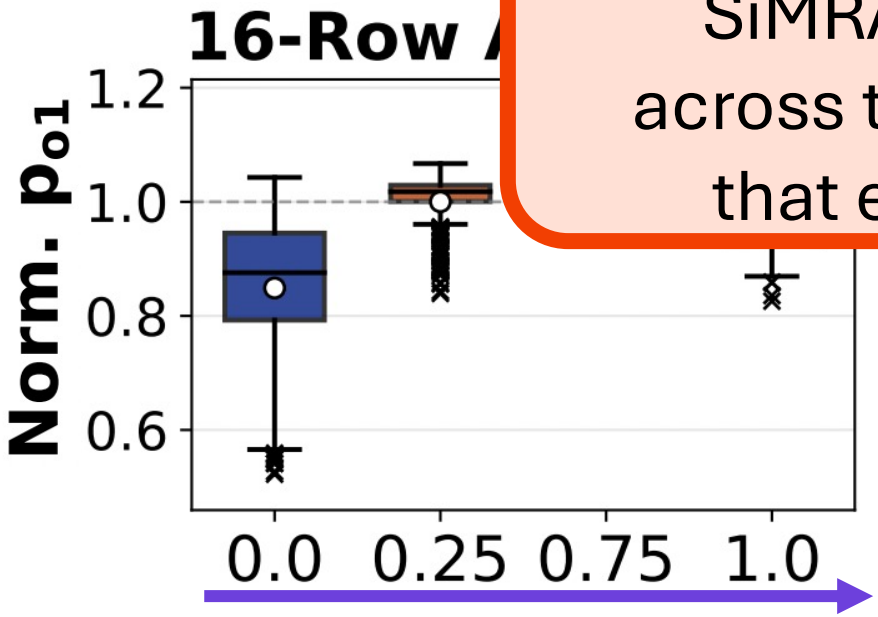
- Target Columns: Randomly Select 1/8th of All Columns
- Main metric (p_{o1}): fraction of output bits across target columns that equal logic-1 when SiMRA is executed with random inputs
- We sweep the fraction of logic-1 in controlled cells across {0.0, 0.25, 0.50, 0.75, 1.0}

Baseline



Results: Concurrently Computing Columns

p_{o1} normalized to the p_c = 0.5 baseline

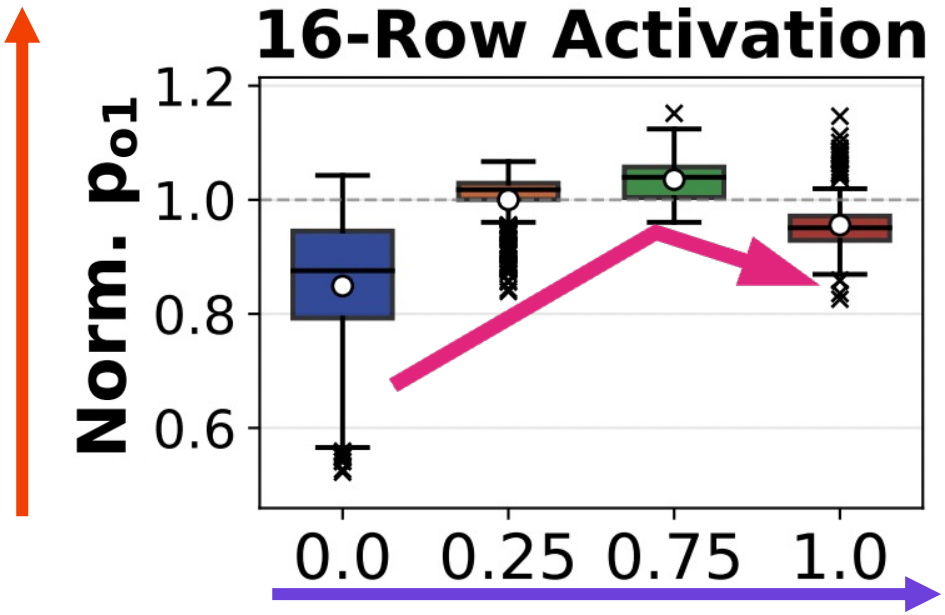


Recall: p_{o1} is the fraction of SiMRA output bits across target columns that equal logic-1

Fraction of logic-1 in controlled cells:
{0.0, 0.25, 0.75, 1.0}

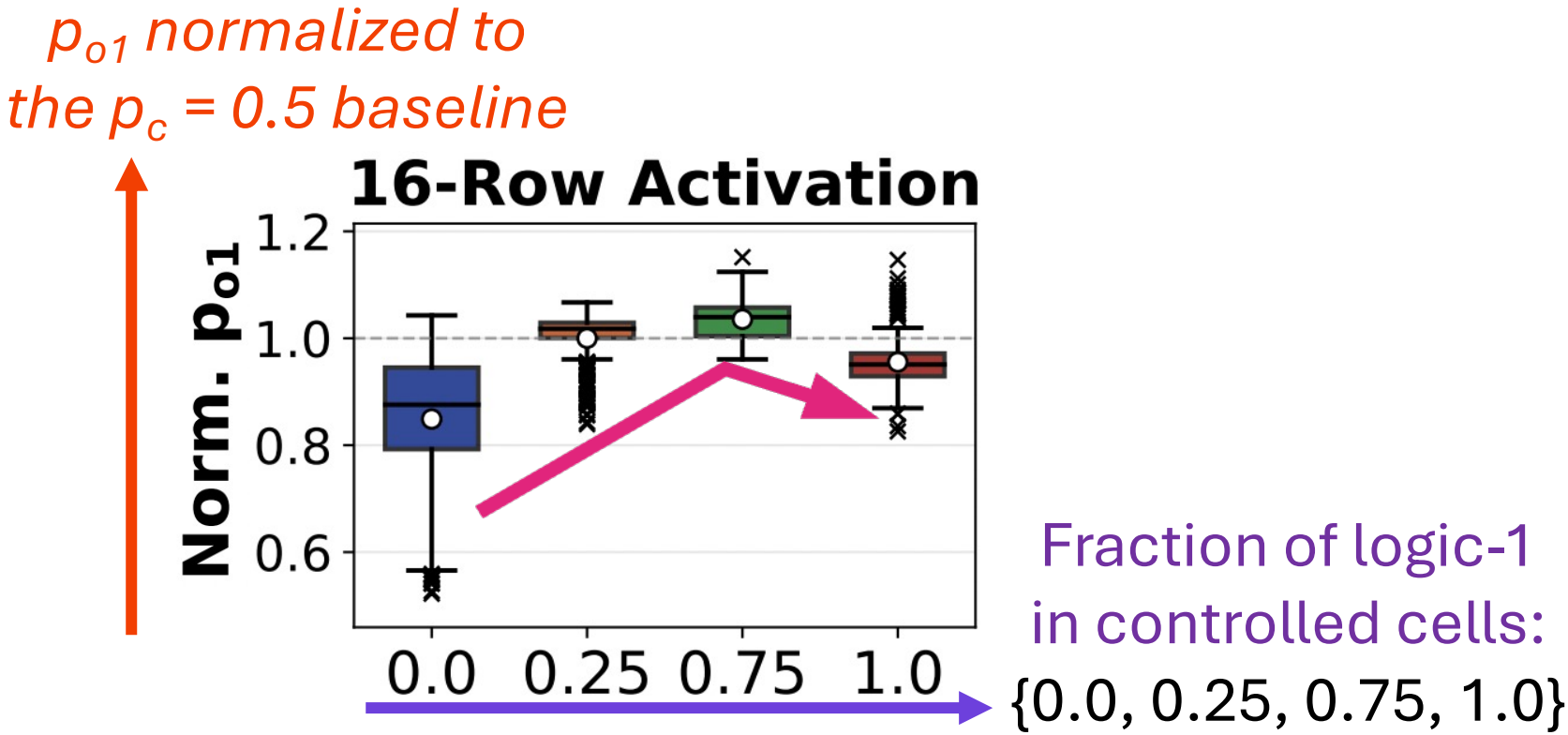
Results: Concurrently Computing Columns

p_{o1} normalized to the p_c = 0.5 baseline



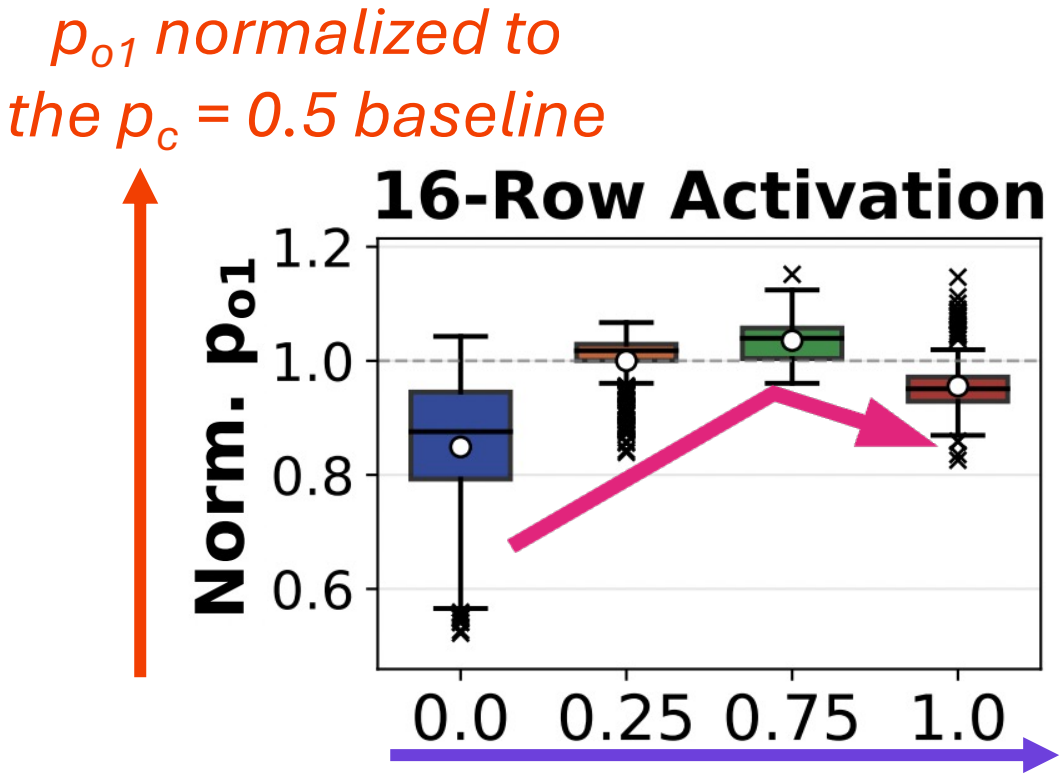
Fraction of logic-1 in controlled cells: {0.0, 0.25, 0.75, 1.0}

Results: Concurrently Computing Columns



Columns that concurrently perform computations affect SiMRA output by **6%** on average and **up to 48%**

Results: Concurrently Computing Columns

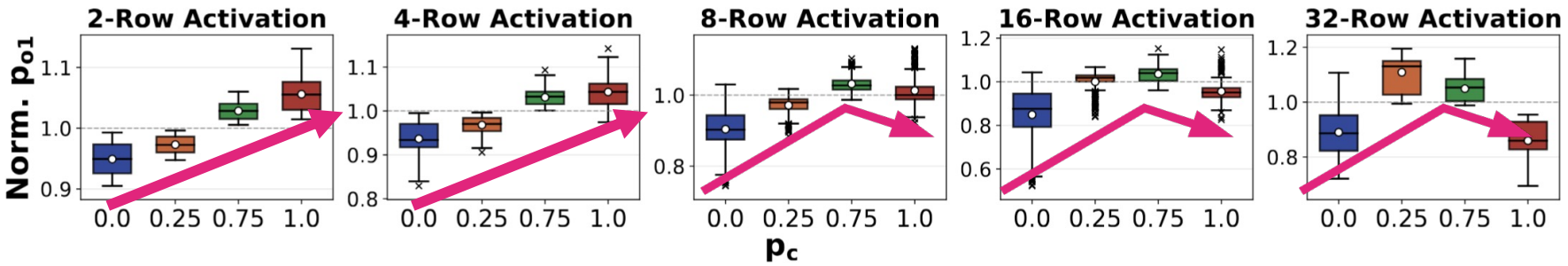


Key Takeaway

Concurrently computing columns more strongly affect SiMRA outputs than adjacent rows

Results: Concurrently Computing Columns

We consistently observe the substantial interference across all tested SiMRA row counts



Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

Mitigating PuDGhost

Conclusion

Reliability Challenges Induced by Non-Operand Data

PuDGhost violates a key expectation of PuD: each column's computation should depend *solely* on its own operand data

Key Challenge: PuDGhost-induced Interference is ***Dynamic***

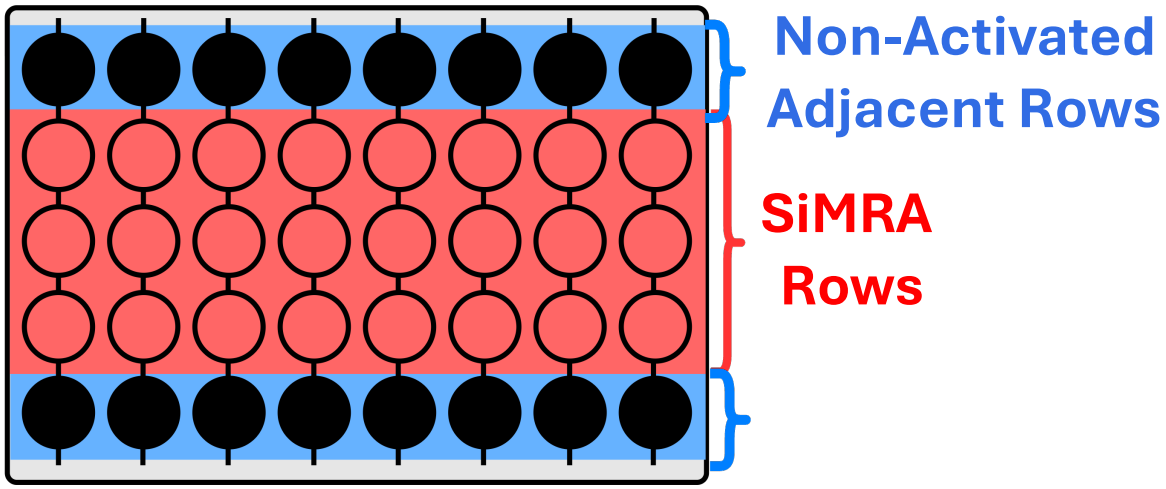
- Columns that appear reliable under one non-operand data configuration may produce errors under another

Challenges of Naive Column Screening

- **Column Screening**:
  - Post-manufacturing screening identifies reliable columns by testing with large sets of random inputs
  - Only identified reliable columns are used during execution

- Screening that does *not* account for PuDGhost can **fail:**
- Columns labeled as reliable during column screening can **later become unreliable** during PuD execution due to dynamic non-operand data

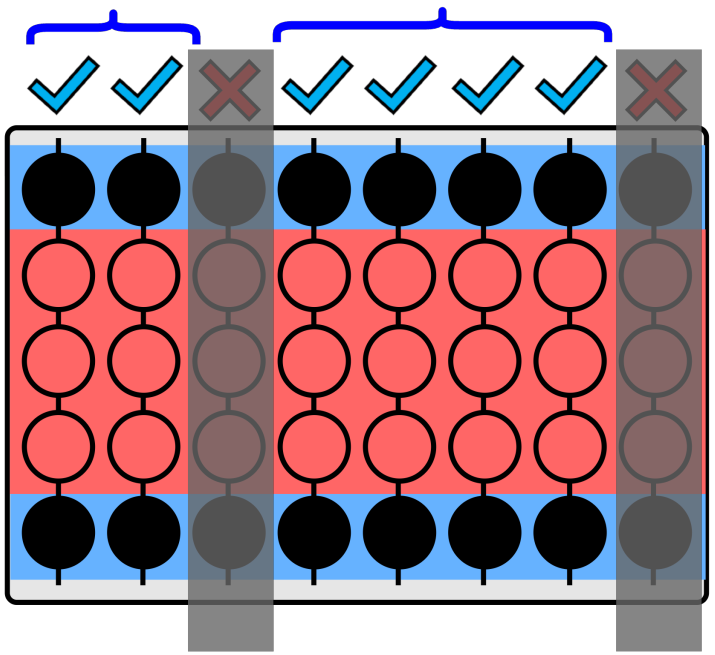
Challenges of Naive Column Screening



Stage 1:
Column Screening

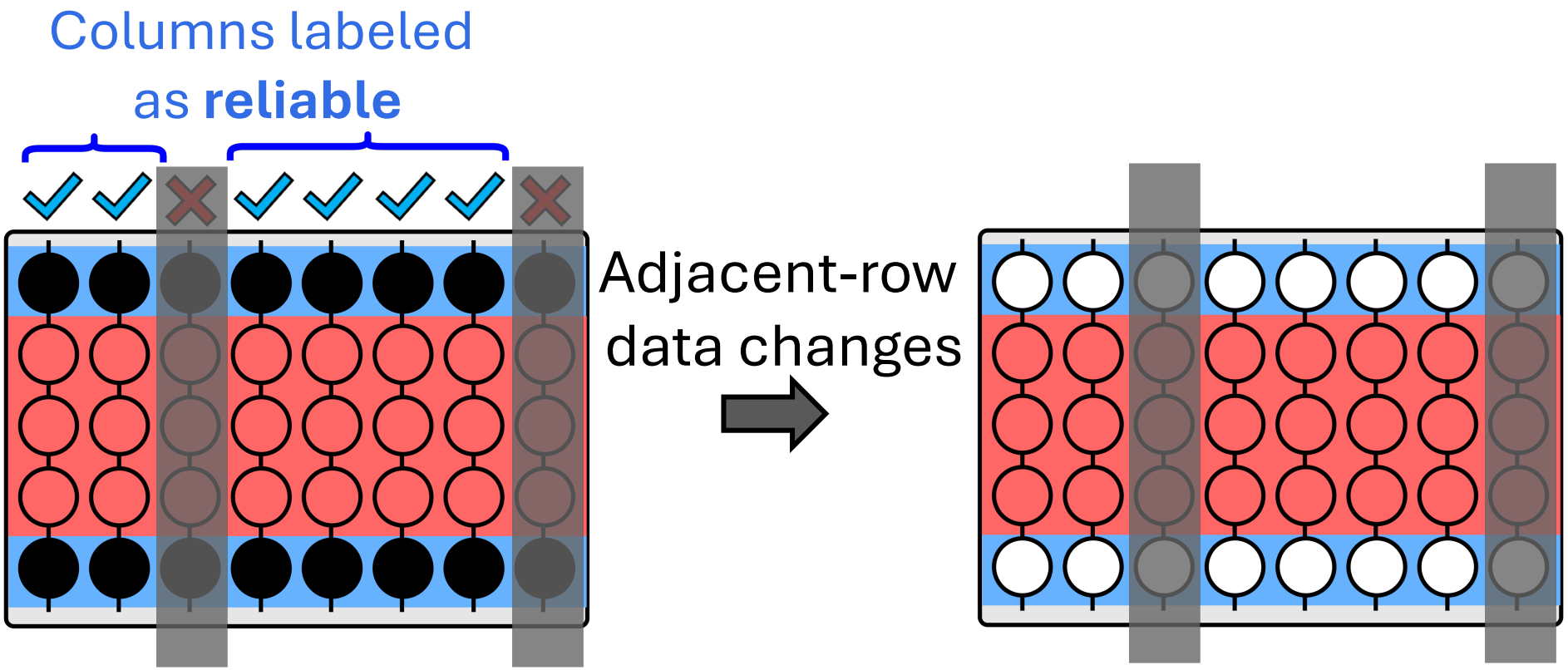
Challenges of Naive Column Screening

Columns labeled as **reliable**



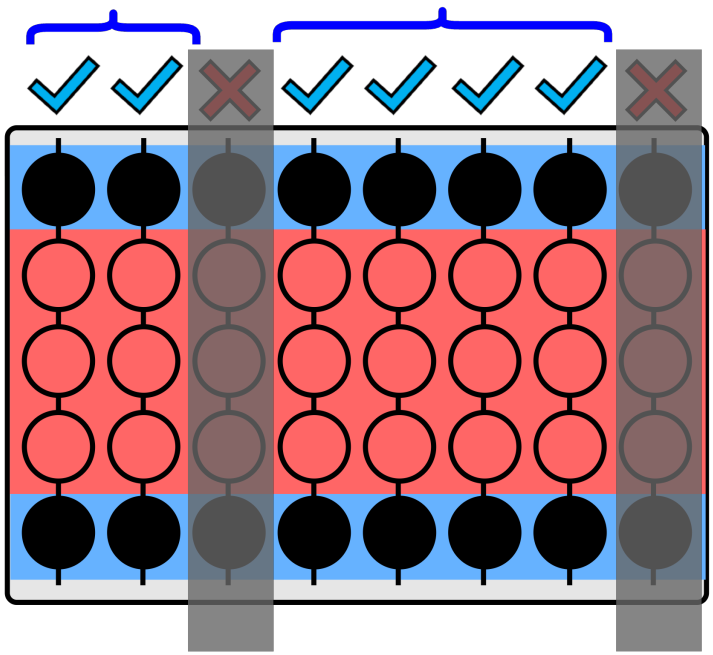
Stage 1:
Column Screening

Challenges of Naive Column Screening



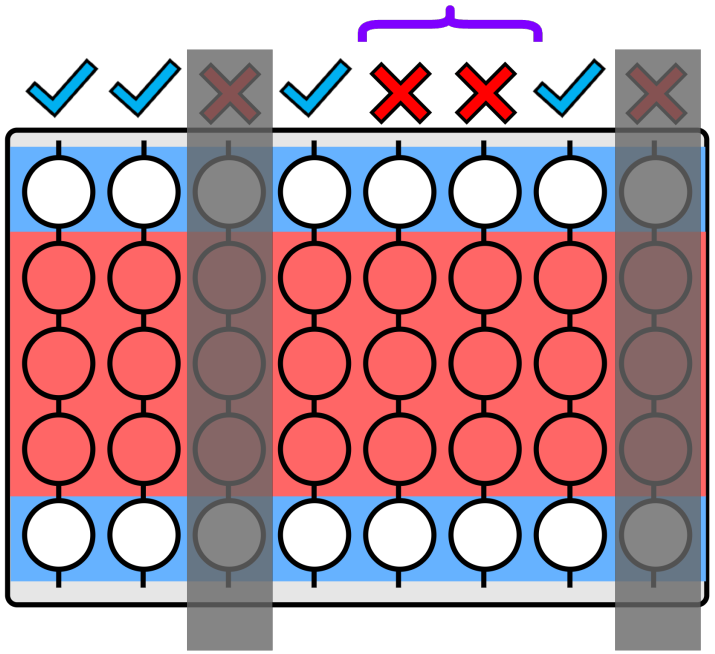
Challenges of Naive Column Screening

Columns labeled as **reliable**



Adjacent-row data changes

Columns labeled as **reliable** **later become unreliable**



Stage 1: Column Screening

Stage 2: PuD Execution

Robust Column Screening Technique

- We propose a new *robust* column screening
- **Our approach:** Fix adjacent-row data across column screening and PuD execution
  - Use the same data pattern in adjacent rows during both column screening and PuD execution

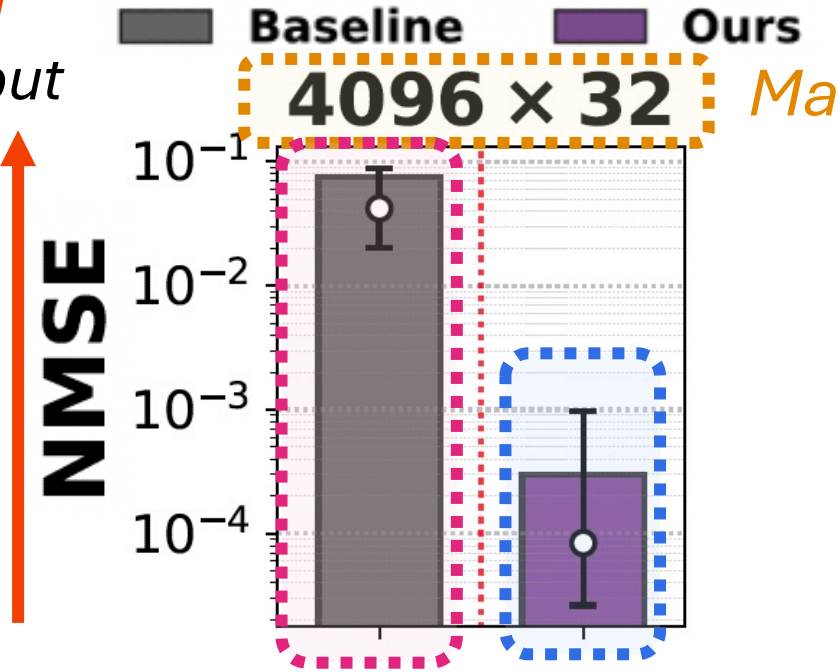
Evaluation of Robust Column Screening

- Setup: Execute Matrix-Vector Multiplication (GEMV) with 8-bit precision on COTS DRAM chips using MAJ3
  - ***Stage 1 (Screening)***: Screen columns using 8,192 random inputs; columns with *no* errors across all inputs are labeled as reliable
  - ***Stage 2 (Execution)***: Execute GEMV using the 32K columns labeled as reliable in Stage 1
- *Configurations:*

	Data in Adjacent Rows		
Configurations	**Stage 1 (Screening)**	**Stage 2 (Execution)**	
Baseline	**All Ones** ⟷	**All Zeros**	**Mismatch**
Ours	**All Ones** ⟷	**All Ones**	**Match**

Results: Column Screening

Normalized Mean Squared Error of GEMV output



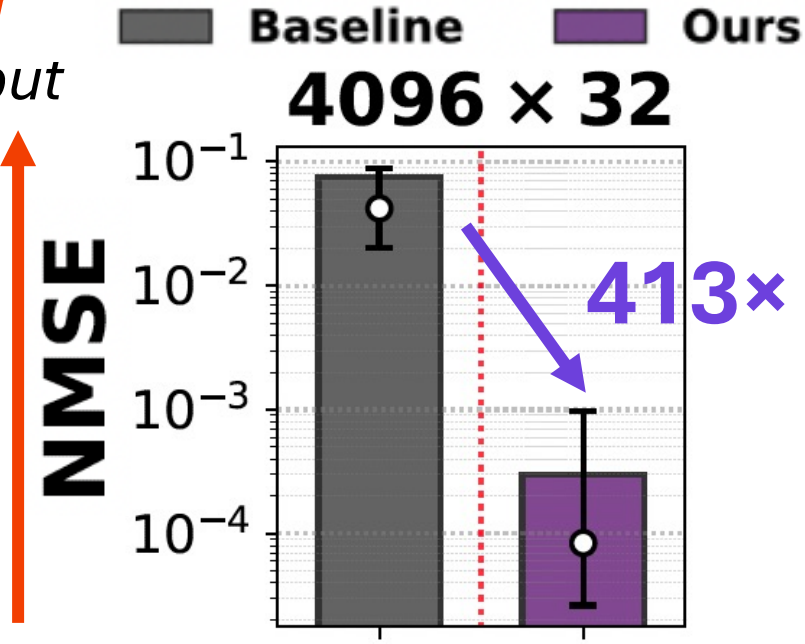
Matrix dimension

PuDGhost-Unaware Column Screening

Our PuDGhost-Aware Column Screening

Results: Column Screening

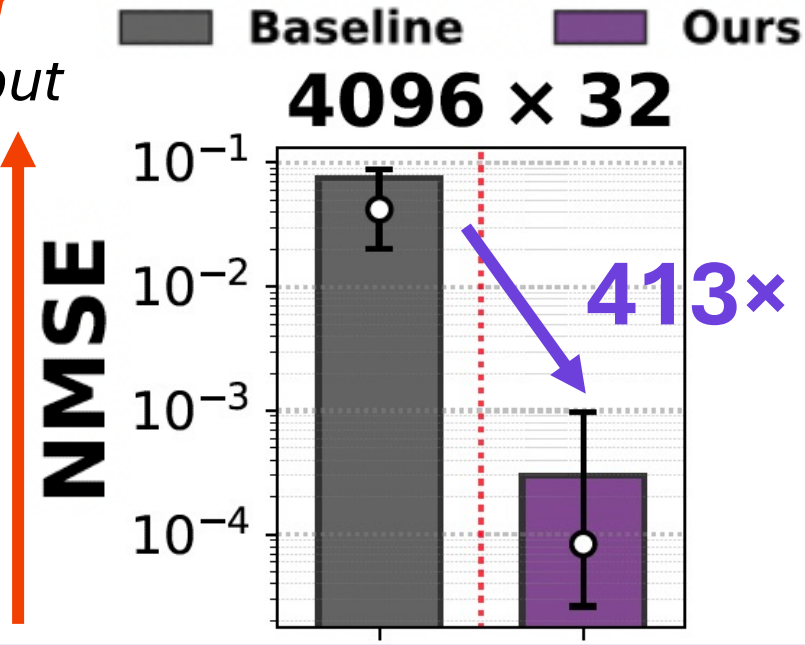
Normalized Mean Squared Error of GEMV output



Our robust column screening **reduces NMSE of GEMV outputs by 413×** compared to PuDGhost-unaware baseline

Results: Column Screening

Normalized Mean Squared Error of GEMV output



Key Takeaway

Our solution significantly reduces application-level errors

Available on arXiv

PuDGhost: Experimental Analysis of Computation Result Corruption in Processing-using-DRAM Operations on Real DRAM Chips and Implications for Future Systems

Daichi Tokuda[1] İsmail Emir Yüksel[2] Tatsuya Kubo[1,4] Ataberk Olgun[2] Haocong Luo[2] Nisa Bostanci[2]
Jikun Wang[2] A. Giray Yağlıkçı[3] Shinya Takamaeda-Yamazaki[1,4] Onur Mutlu[2]
[1]The University of Tokyo [2]ETH Zurich [3]CISPA [4]RIKEN

Processing-using-DRAM (PuD) is a promising computation paradigm to alleviate the frequent data movement between main memory and processing units. The PuD paradigm provides a substrate for highly parallel computation by using each DRAM column as a computation engine via simultaneous multiple-row activation (SiMRA). Unfortunately, DRAM density scaling might hinder PuD's benefits. This is because denser cell arrays bring rows and columns closer, making even regular DRAM operations susceptible to noise and interference from neighboring cells. PuD repurposes DRAM from a storage device into a parallel computing substrate, yet no *prior work investigates whether interference from rows or columns that are not intended to participate in the computation can compromise PuD robustness.*

existing operational principles of DRAM to realize massively parallel computation within DRAM. Prior work demonstrates the potential of this approach to substantially improve throughput and energy efficiency compared to conventional systems for a wide range of applications, including databases, web search, data analytics, graph processing, genome analysis, cryptography, optimization solvers, hyperdimensional computing, and LLMs [26, 33, 35, 37, 38, 40–42, 48, 49, 51, 55–59, 61, 72–75].

The core computational capability of various PuD architectures relies on Simultaneous Multiple-Row Activation (SiMRA), a DRAM operation that simultaneously activates multiple DRAM rows within a subarray [23, 26, 33, 38, 40–43, 46–48, 50–52, 54–58, 61, 63, 64, 67, 68, 76]. Figure 1a shows how performing

https://arxiv.org/pdf/2606.19119

More in the Paper

- **More Characterization Results**
  - Temperature, spatial variation, data patterns
- **Details and Analyses on Mitigating PuDGhost**
  - Discussion on how modifying DRAM chips could help mitigate PuDGhost
  - Interference-aware compute row layout
  - Column screening tradeoffs across reliability, throughput, and capacity overhead
- **More Application-level Evaluation**
  - PuDGhost's impact on SiMRA-based true random number generation (TRNG)

Outline

Background

Problem & Goal

Testing Infrastructure & Methodology

Interference from Non-Activated Rows

Interference from Concurrently Computing Columns

Mitigating PuDGhost

Conclusion

Conclusion

We extensively study the impact of interference from non-operand data in Processing-using-DRAM operations

We characterize 96 DDR4 chips:

- Non-activated rows affect SiMRA outputs by up to 10% for random inputs
- Concurrently computing columns affect SiMRA outputs by up to 48% for random inputs

We propose PuDGhost-aware column screening techniques:

- Our solutions reduce GEMV NMSE by 413× compared to PuDGhost-unaware systems

We hope our work motivates and guides solutions to enable future PuD systems that are robust

PuDGhost:

Experimental Analysis of Computation Result Corruption in Processing-using-DRAM Operations on Real DRAM Chips and Implications for Future Systems

Daichi Tokuda

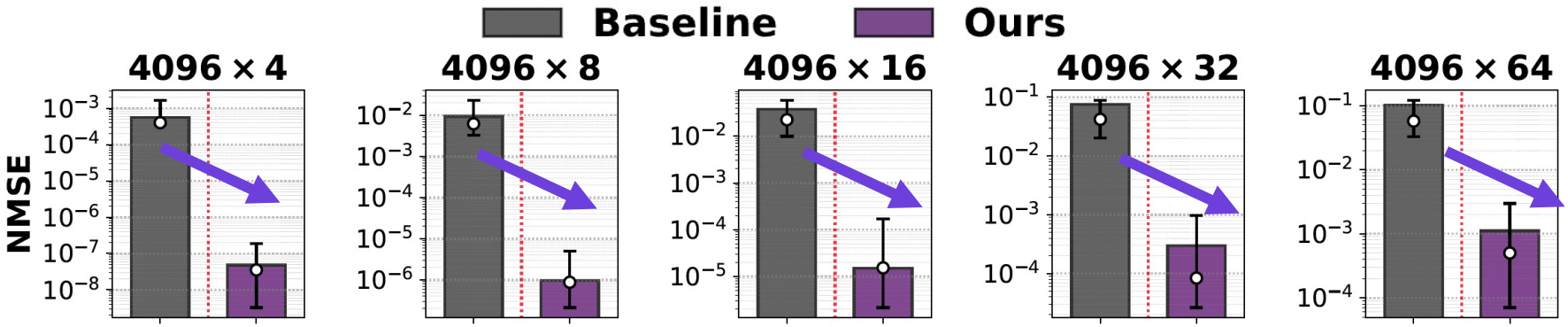
E. Yuksel T. Kubo A. Olgun H. Luo

N. Bostanci J. Wang A. G. Yağlıkçı

S. Takamaeda-Yamazaki O. Mutlu

Backup Slides

Results: Column Screening



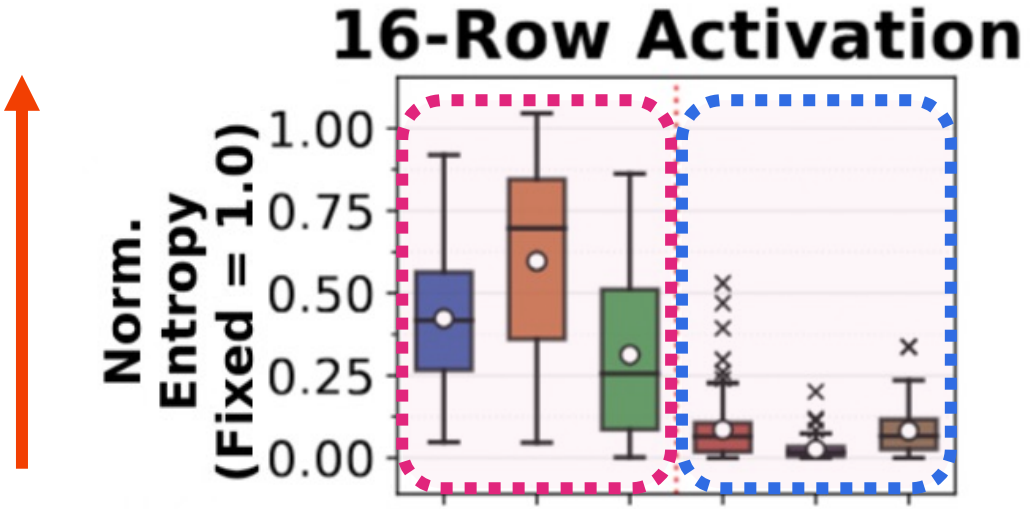
PuDGhost's Impact on SiMRA-based TRNG

- We evaluate PuDGhost's impact on SiMRA-based true random number generation (TRNG) on COTS DRAM chips
- Setup:
  - Step 1: Find input pattern that maximizes entropy; identify source columns with randomness
  - Step 2: **Fix the input pattern of source columns** from Step 1 and **measure entropy while varying other data**
- Conditions:
  - **Fixed (baseline)**: All data remains **identical to Step 1**
  - Adjacent-row interference: Vary **adjacent-row data**
  - Inter-column interference: Vary **non-source columns' inputs**

PuDGhost's Impact on SiMRA-based TRNG

- We evaluate PuDGhost's impact on SiMRA-based true random number generation (TRNG) on COTS DRAM chips

Entropy normalized to Fixed (baseline)

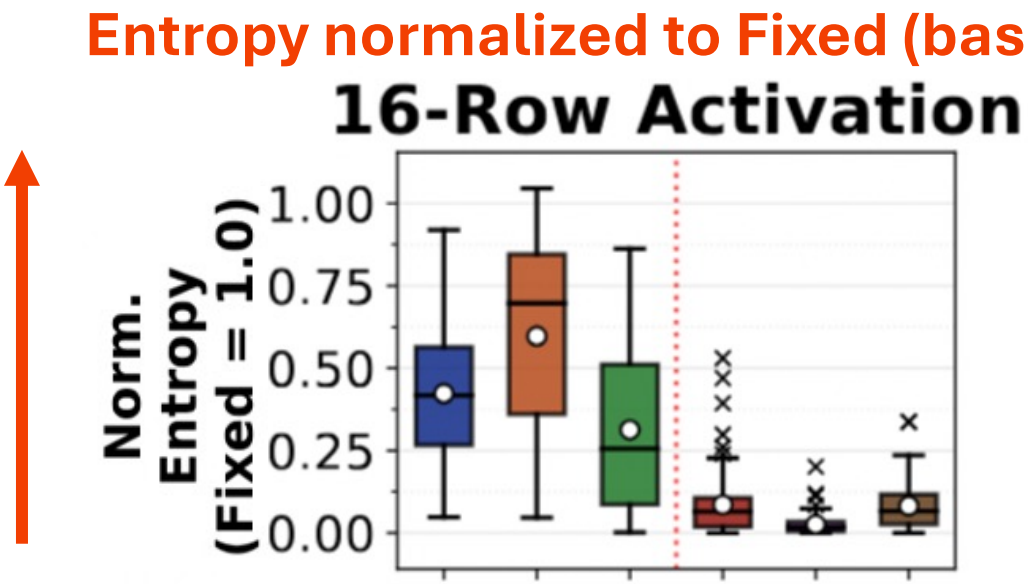


Adjacent-Row Interference

Inter-Column Interference

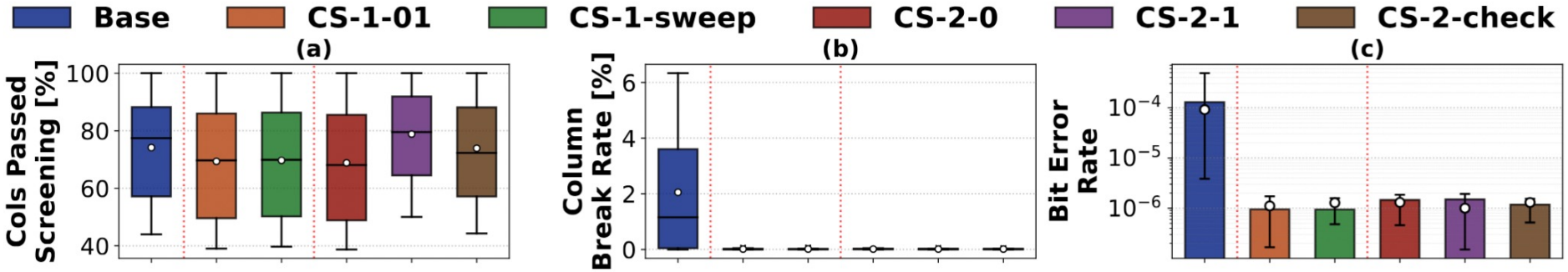
PuDGhost's Impact on SiMRA-based TRNG

- We evaluate PuDGhost's impact on SiMRA-based true random number generation (TRNG) on COTS DRAM chips



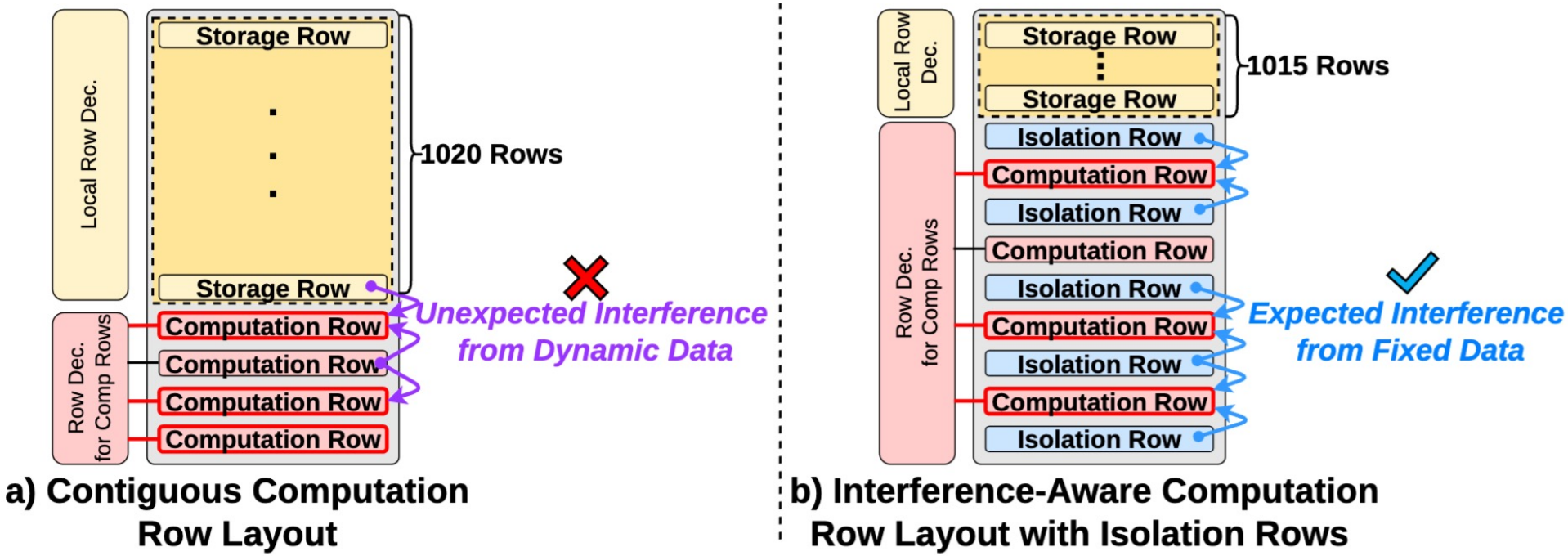
SiMRA-based TRNG designs that do not account for PuDGhost could unknowingly lose up to 93% of the entropy

Column Screening Techniques



Our column screening techniques reduce MAJ3 bit error rate by 91× compared to PuDGhost-unaware systems

PuDGhost-Aware Compute Row Layout



Device-Level PuDGhost Mechanism

- Hypothesis: PuDGhost arises from **electrical coupling during SiMRA charge sharing and sensing**
- **Interference from non-activated rows**
  - Electrical coupling from adjacent rows
- **Interference from concurrently computing columns**
  - Coupling through **bitlines**
  - Interactions through **sense amplifier circuitry**
  - **The combination of these two effects** leads to the complex interference behavior

We need device-level studies to fundamentally understand PuDGhost, and our study can inspire such future work and be a good starting point

PuDGhost vs. Read Disturbance Phenomena

- Interference source:
  - Read disturbance: activated aggressor row
  - PuDGhost: ***non-activated*** adjacent rows and concurrently computing **columns**
- Triggering mechanism:
  - Read disturbance: repeated activation (RowHammer) or prolonged activation (RowPress)
  - PuDGhost: **a single SiMRA operation**
- Manifestation:
  - Read disturbance: persistent bitflips through gradual charge leakage
  - PuDGhost: **transient errors** during charge sharing and sensing

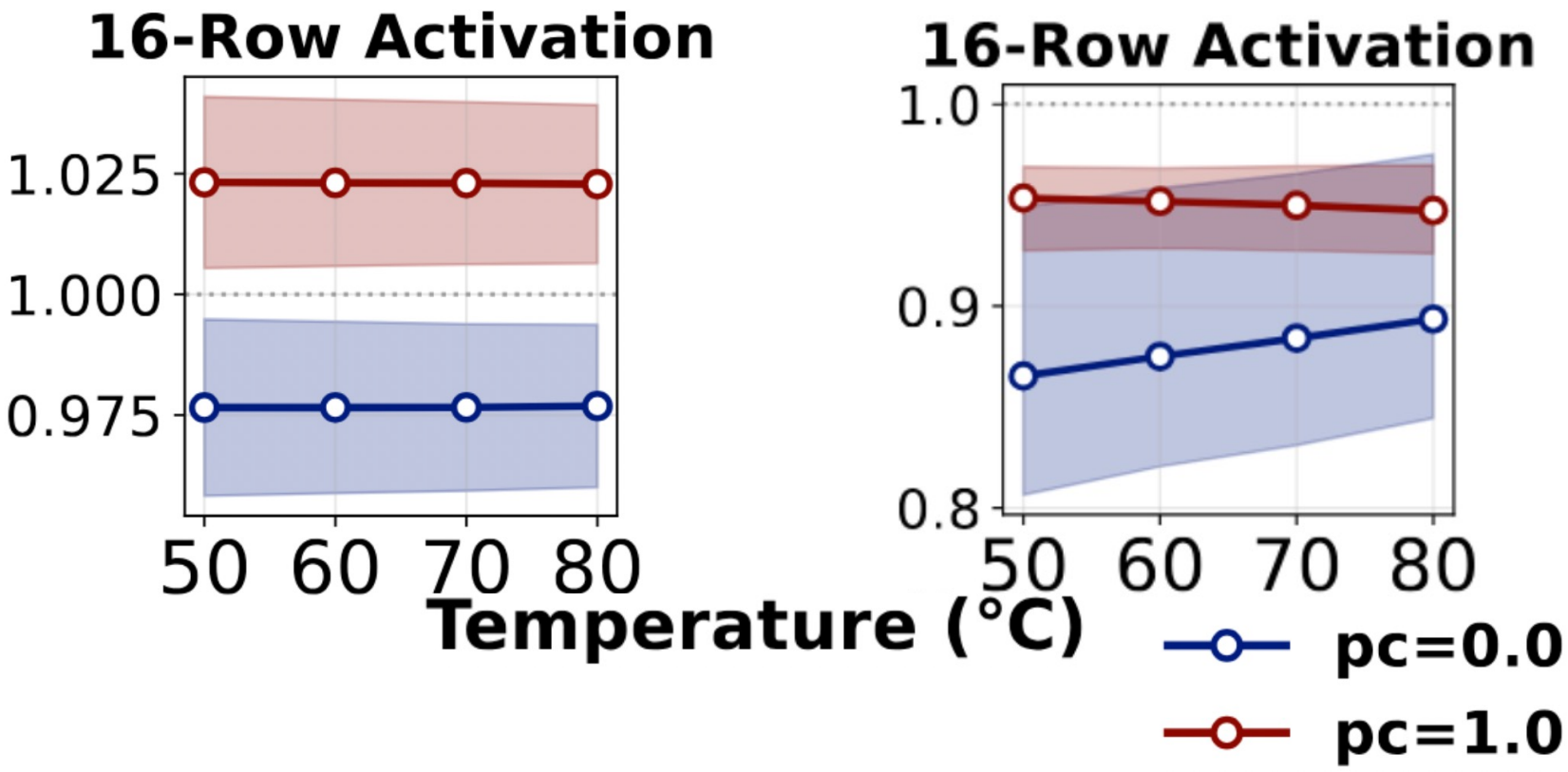
Toward Relibale PuD Systems

- Full DRAM Redesign:
  - Design DRAM from the ground up for reliable PuD; a promising but long-term direction
- Cell Array-Level Modifications:
  - Potential modifications to DRAM cell array design to reduce interference (e.g., isolation transistors)
- PuD ECC Schemes
  - Prior work proposes ECC schemes tailored for PuD to detect and correct computation errors
  - (e.g., Reed-Solomon-like approach)
- Error-Tolerant Applications
  - Some applications can tolerate a certain level of computation errors without significant quality degradation (e.g. machine learning, genomic sequence search)

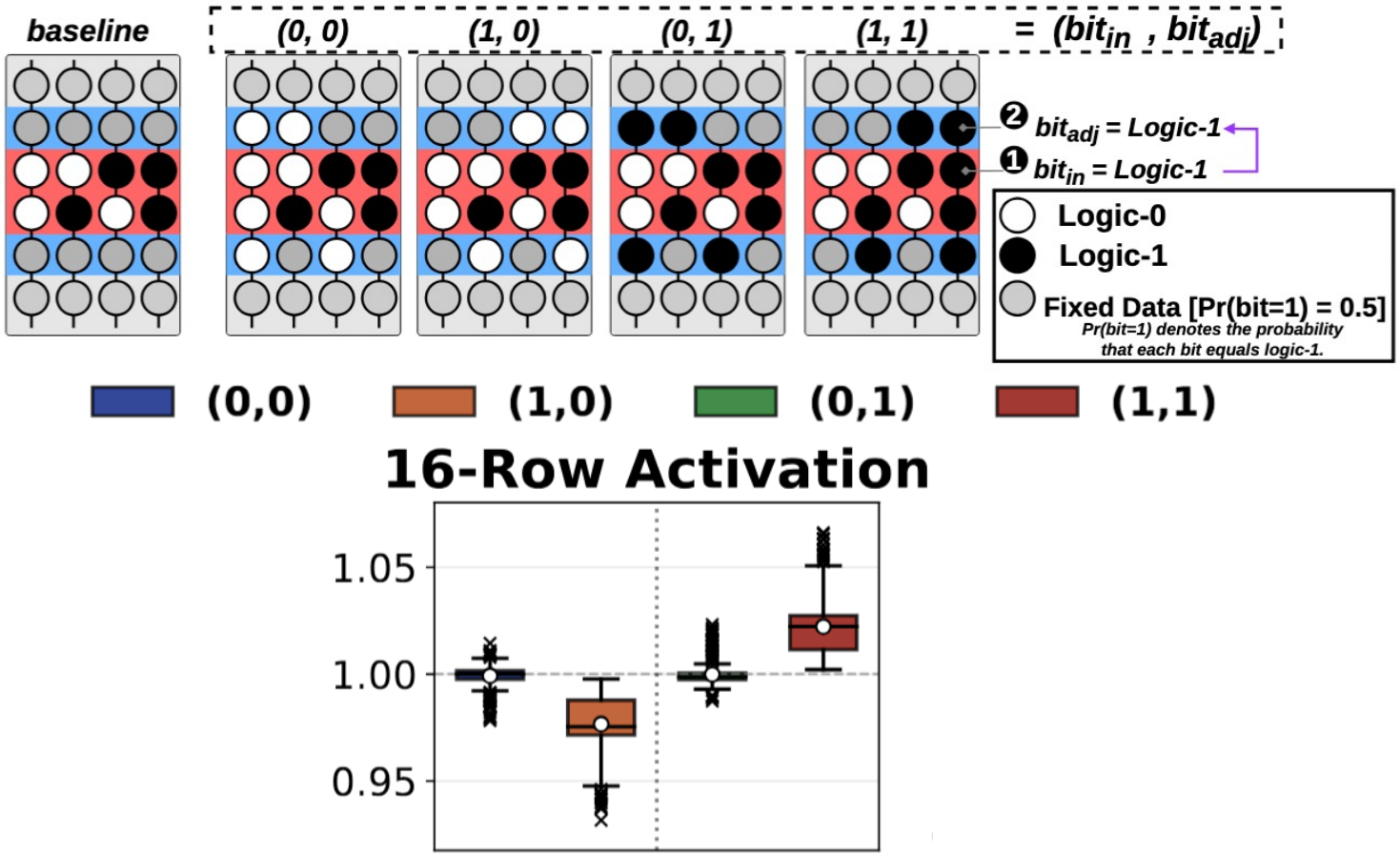
Temperature Sensitivities

Adjacent Rows

Conucurrently Computing Columns



Input-Dependent Adjacent Row Patterns



Cells in SiMRA rows storing logic-1 are **substantially more susceptible** to interference from non-activated adjacent rows