



ColumnKeeper: Efficient Solutions to the ColumnDisturb Vulnerability in DRAM-based Systems

Andreas Kosmas Kakolyris

F. Nisa Bostancı

Ataberk Olgun

İsmail Emir Yüksel

Harsh Songara

Konstantinos Marios Sgouras

Umut Başer

Konstantinos Kanellopoulos

A. Giray Yağlıkçı

Onur Mutlu

SAFARI

ETH zürich



Executive Summary

Problem: DRAM chips are **vulnerable** to **ColumnDisturb**, a new, **column-based** read disturbance phenomenon.

Naively mitigating ColumnDisturb incurs **significant overheads**.

Goal: Design mechanisms that **mitigate ColumnDisturb bitflips** with **low performance, energy and area** overheads

Key Idea: Track row activations **at subarray granularity**.

Iteratively **refresh all rows**.

ColumnKeeper-D (Deterministic): Uses **separate** activation counters for **odd** and **even bitlines** of each subarray and issues a preventive refresh when **either** counter reaches a **predetermined threshold**

ColumnKeeper-P (Probabilistic): **Probabilistically** issues refreshes to one row in **three consecutive subarrays** upon a row activation in the middle subarray

Evaluation

- ColumnKeeper **mitigates ColumnDisturb** bitflips with **low overheads** at **current (1M)** and **“near-future” (128K)** thresholds
- ColumnKeeper **can mitigate** ColumnDisturb bitflips with **low overheads** even at **low thresholds (16K)** with **changes to DRAM microarchitecture**

Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

Conclusion

Outline

Background

Motivation

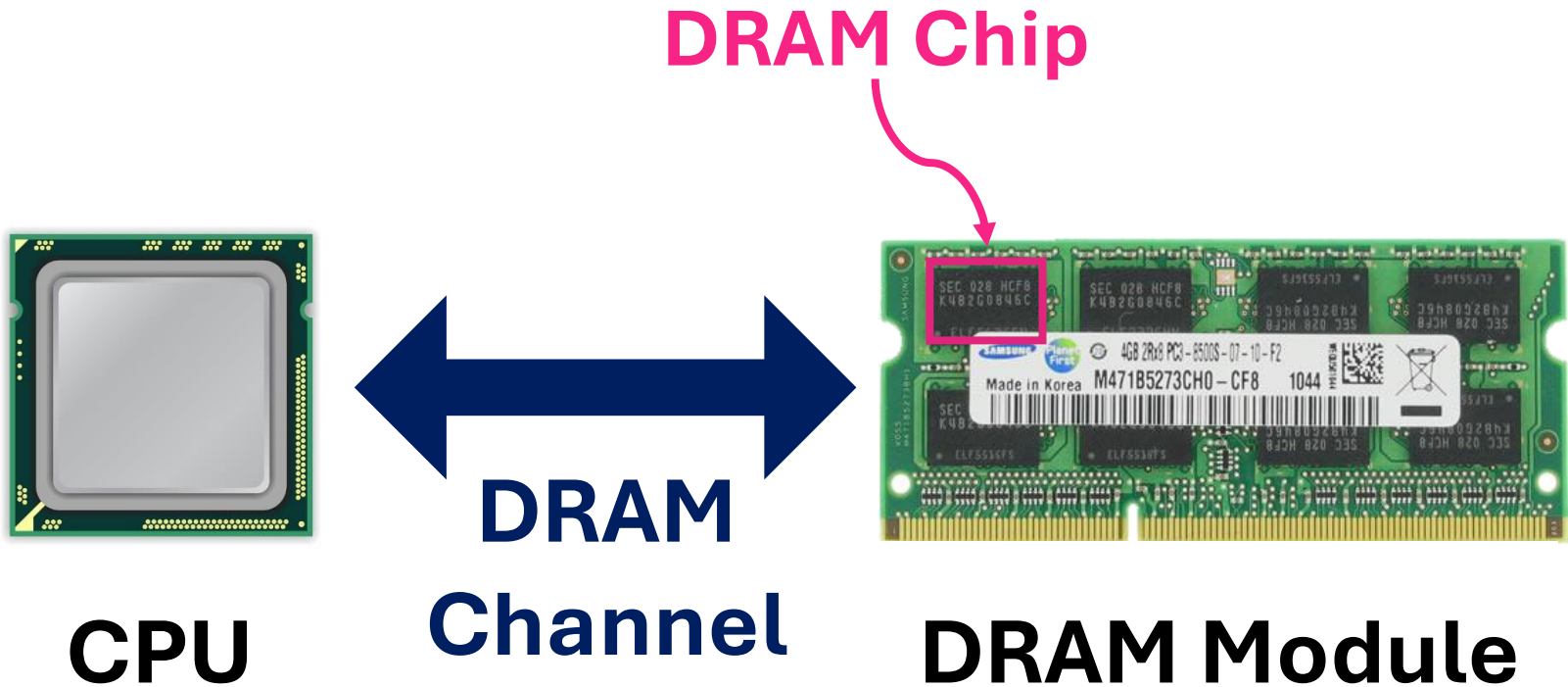
ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

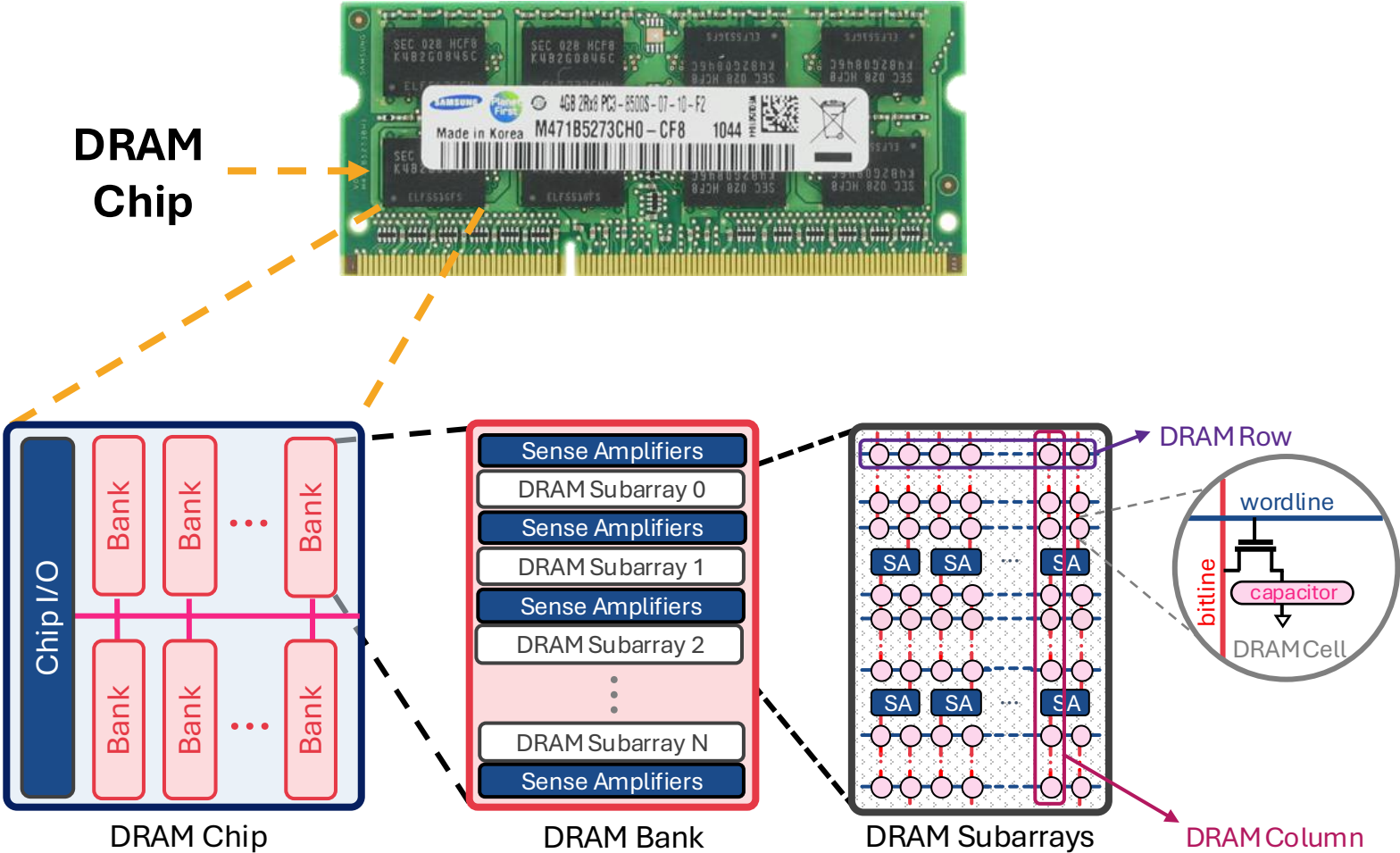
Evaluation

Conclusion

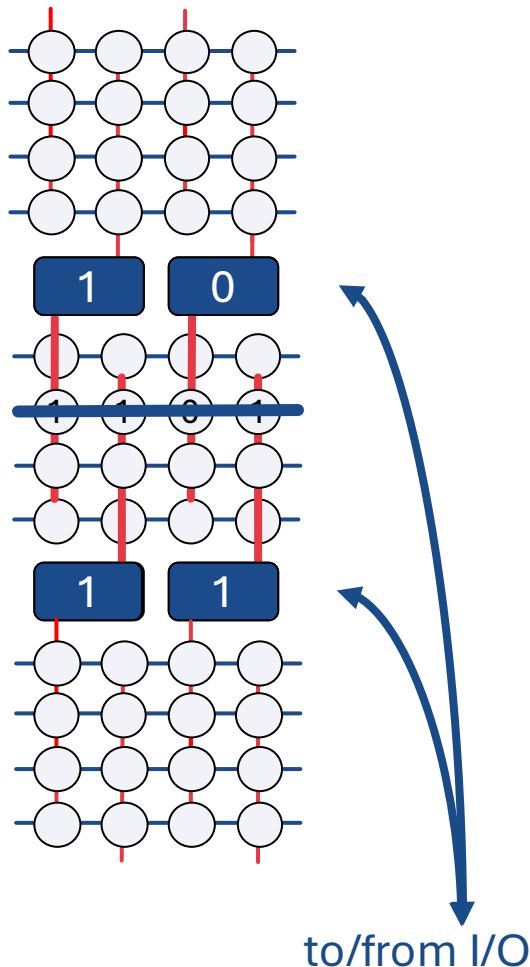
DRAM Organization



DRAM Organization



DRAM Organization

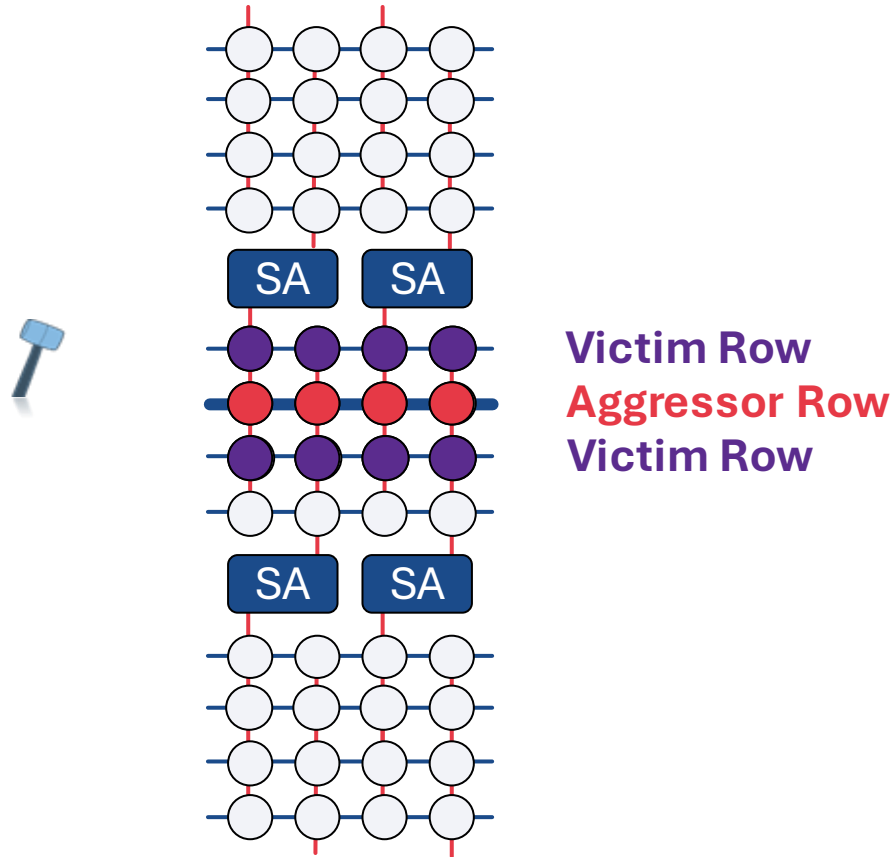


- 1 **ACTIVATE:** Fetch the row into the **sense amplifiers**
- 2 **READ/WRITE:** Retrieve or update data
- 3 **PRECHARGE:** Prepare the bank for a new ACTIVATE

DRAM Read Disturbance: RowHammer

Read disturbance in DRAM **breaks memory isolation**

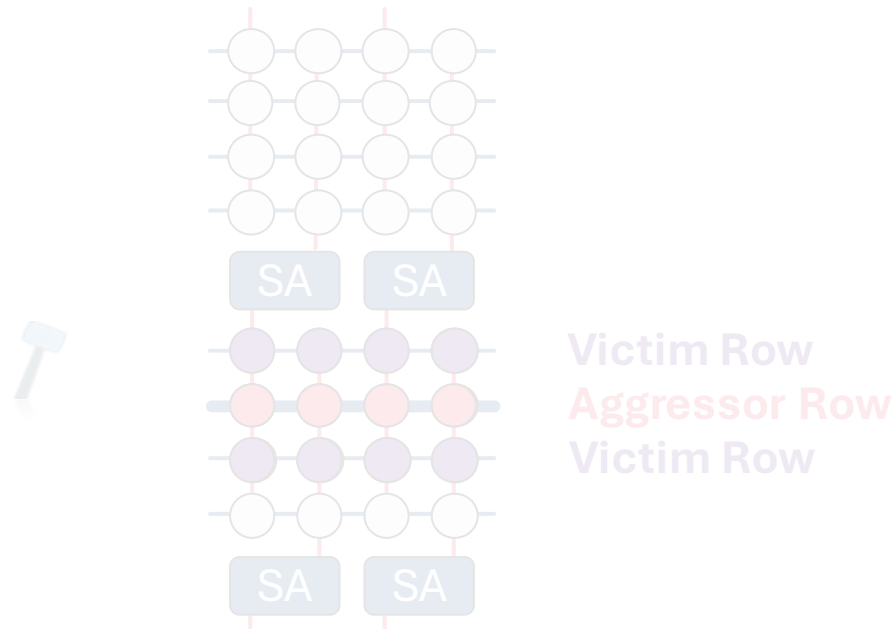
Prominent example: **RowHammer**



DRAM Read Disturbance: RowHammer

Read disturbance in DRAM **breaks memory isolation**

Prominent example: **RowHammer**



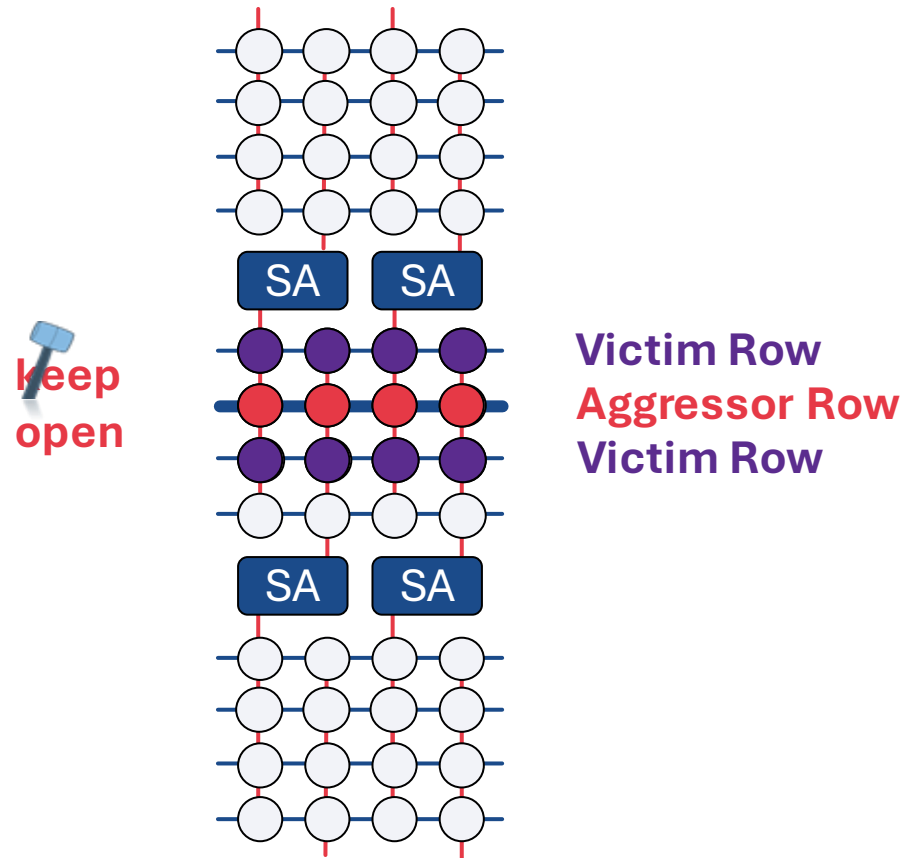
RowHammer causes bitflips in adjacent rows in the same subarray



DRAM Read Disturbance: RowPress

Read disturbance in DRAM **breaks memory isolation**

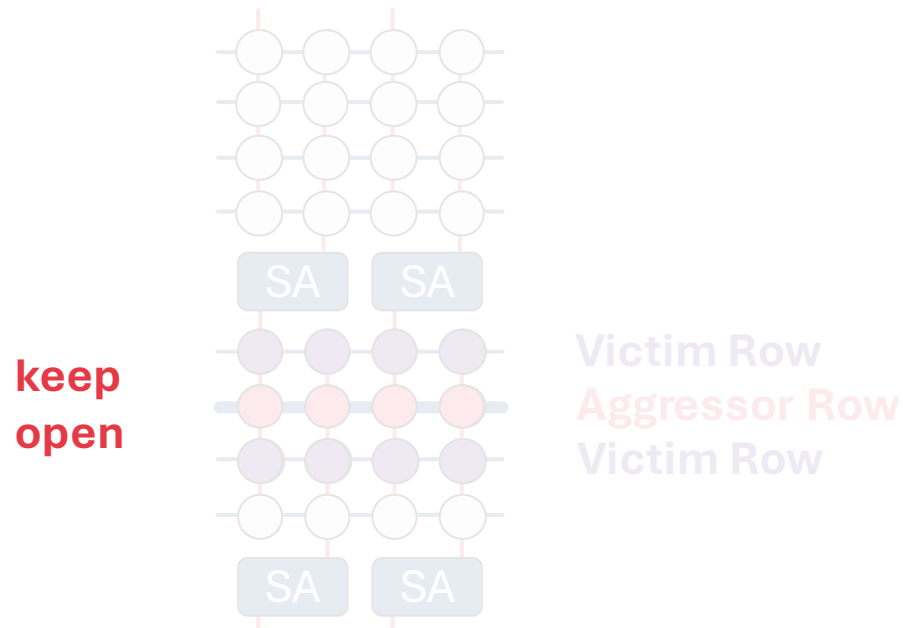
Another prominent example: **RowPress**



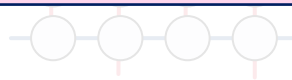
DRAM Read Disturbance: RowPress

Read disturbance in DRAM **breaks memory isolation**

Another prominent example: **RowPress**

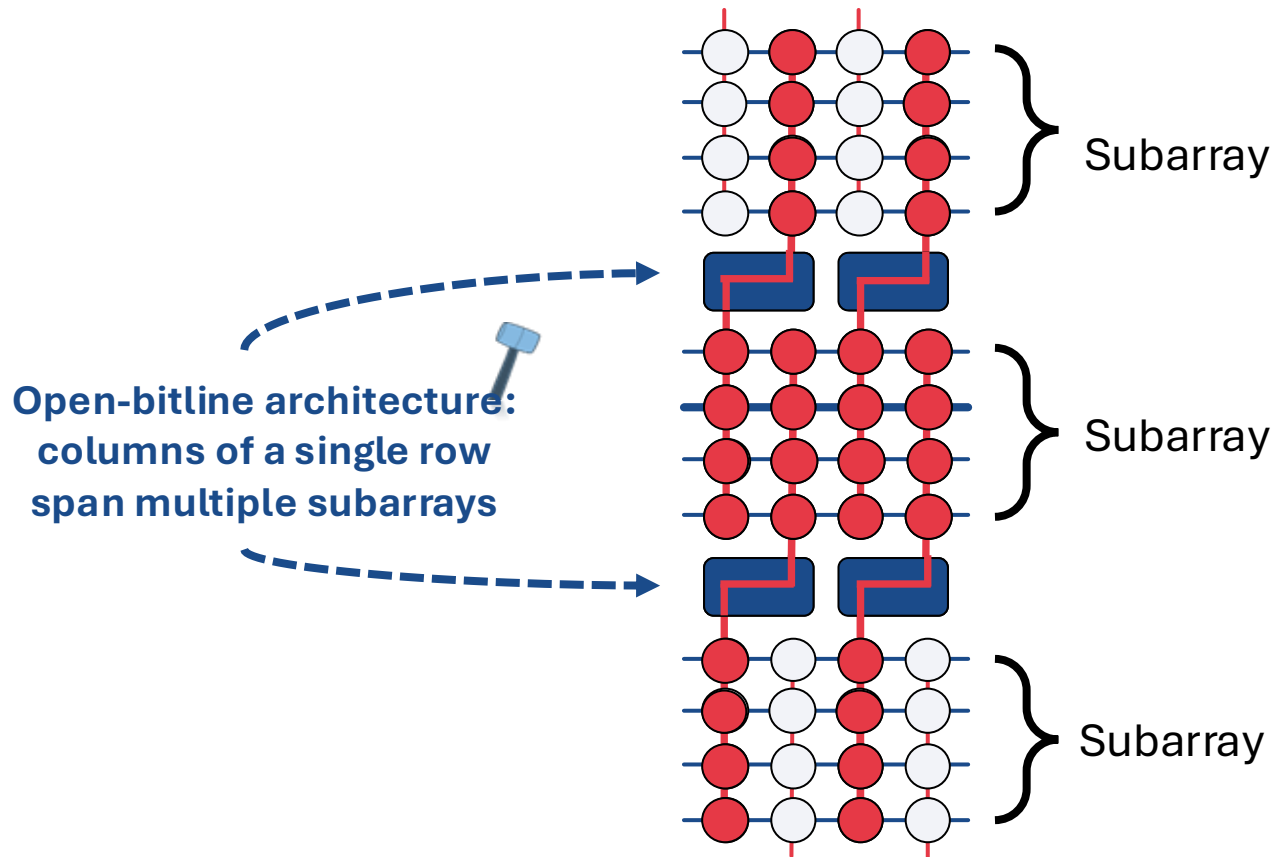


RowPress causes more bitflips in adjacent rows
in the **same subarray**



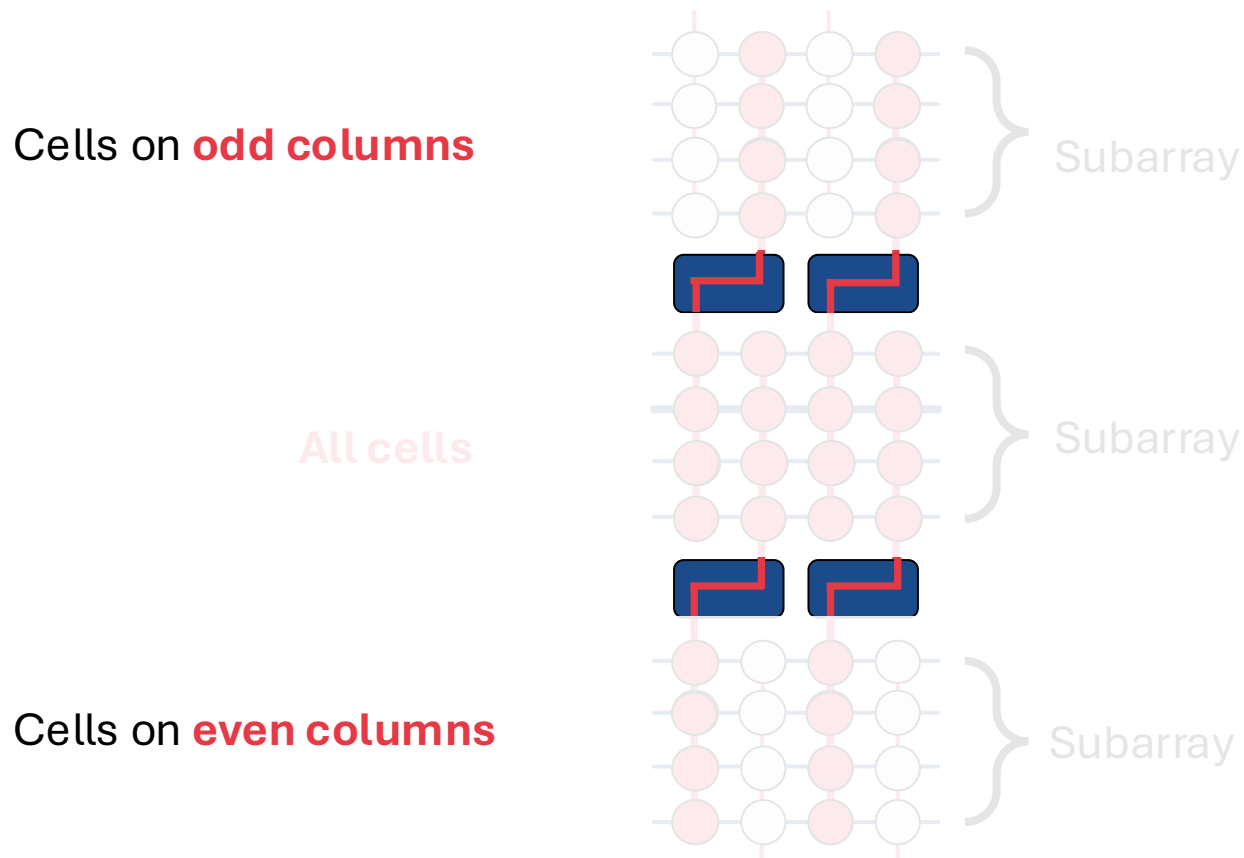
DRAM Read Disturbance: ColumnDisturb

ColumnDisturb: Repeatedly **hammering** or **pressing** a DRAM row **disturbs many** DRAM cells **on the same DRAM column**



DRAM Read Disturbance: ColumnDisturb

ColumnDisturb: Repeatedly **hammering** or **pressing** a DRAM row **disturbs many** DRAM cells **on the same DRAM column**



DRAM Read Disturbance: ColumnDisturb

ColumnDisturb: Repeatedly **hammering** or **pressing** a DRAM row **disturbs many** DRAM cells **on the same DRAM column**

ColumnDisturb affects DRAM cells across **three consecutive** DRAM **subarrays** (up to **3072** DRAM rows)

Cells on **odd columns**



Subarray

ColumnDisturb induces bitflips within **the DDR4 refresh window**

All cells



Subarray

The **number of activations** required to **induce ColumnDisturb bitflips (N_{CD})** is currently $\approx 1M$



Scaling studies show **ColumnDisturb** is **getting worse** **in newer chips**

Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

Conclusion

Mitigating ColumnDisturb (I)

The **easiest way** of mitigating ColumnDisturb is to **reduce** the DRAM refresh window so that **all rows** are refreshed **before bitflips** can **occur**

Significant performance overheads at near-future ColumnDisturb **thresholds:**
More than 50% for workloads with **high memory intensity**

Significant DRAM energy overheads at near-future ColumnDisturb **thresholds:**
More than 6x increase in **energy consumption**

Mitigating ColumnDisturb by increasing the **DRAM** refresh rate
incurs **prohibitive overheads**

Mitigating ColumnDisturb (II)

Alternative method: Modifying **RowHammer mitigation mechanisms** to protect against ColumnDisturb

RowHammer mitigations such as **Per-Row Activation Counting (PRAC)**:

- Track activations **at the row granularity**
- Only refresh **a few neighboring victim rows**

Using PRAC to mitigate ColumnDisturb **requires**:

- **Aggressively configuring PRAC** to fire at a low threshold
- **Refreshing** all **3072 rows** across three subarrays

Modified PRAC incurs **prohibitive overheads**

Our Goal

Design mechanisms
that **prevent ColumnDisturb bitflips**
at low performance, energy, and area overheads

Our Overarching Key Idea

Track activations at **subarray granularity**

Iteratively refresh all rows within each subarray

Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

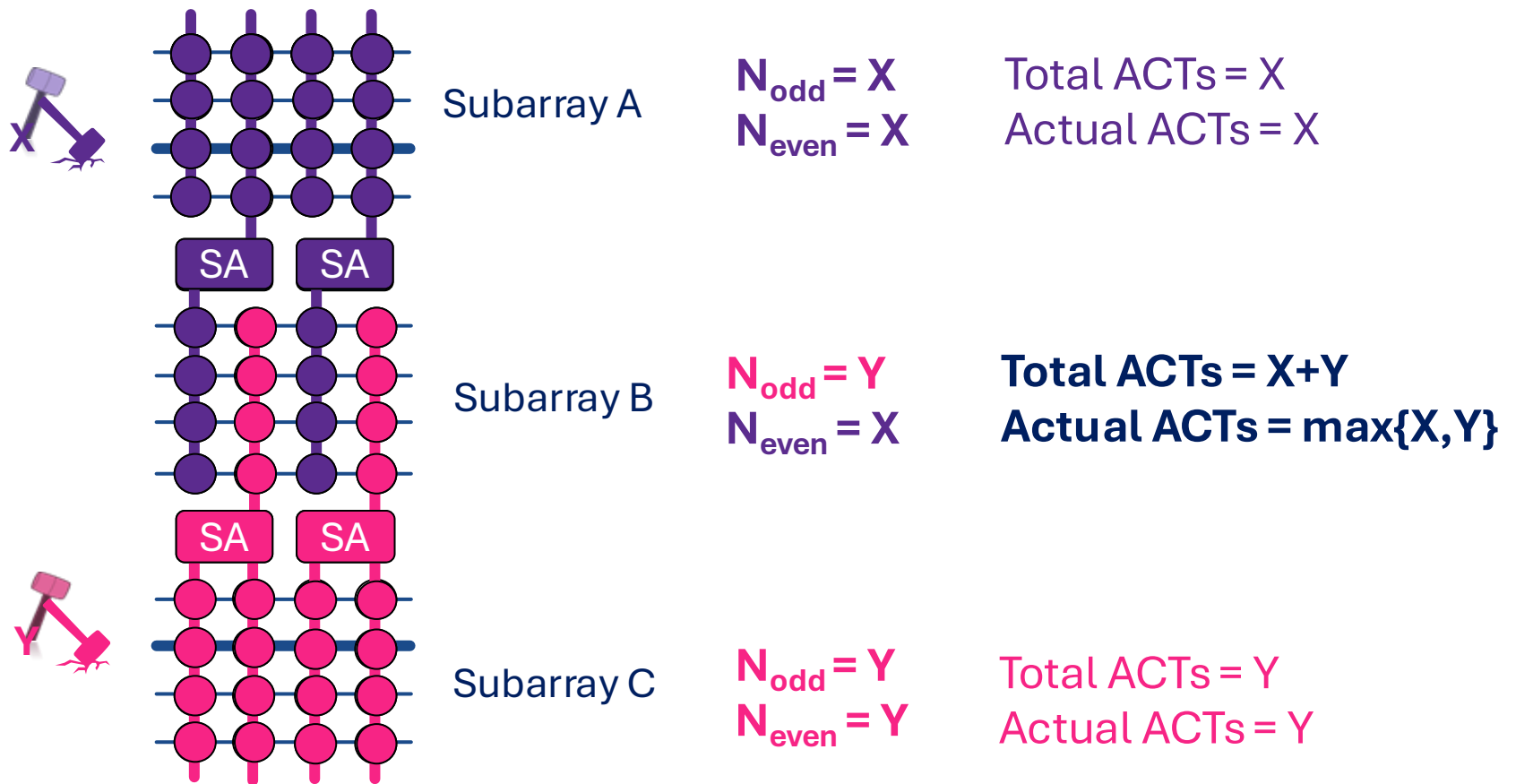
Conclusion

Key Observation

Activations in one subarray affect all columns in that subarray but either odd or even columns in neighboring subarrays

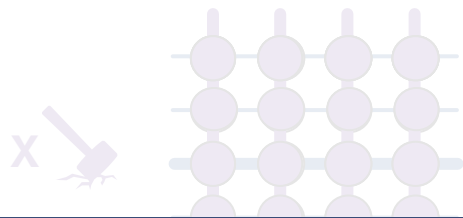
Key Observation

Activations in one subarray affect all columns in that subarray but either odd or even columns in neighboring subarrays



Key Observation

Activations in one subarray affect **all columns** in that subarray but either **odd** or **even** columns in neighboring subarrays



Subarray A

$$N_{\text{odd}} = X$$
$$N_{\text{even}} = X$$

$$\text{Total ACTs} = X$$
$$\text{Actual ACTs} = X$$

Naively counting the activations affecting a subarray **overestimates the activation count** of cells and **increases overheads**

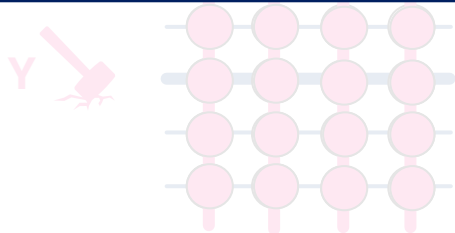


Subarray B

$$N_{\text{odd}} = Y$$
$$N_{\text{even}} = X$$

$$\text{Total ACTs} = X + Y$$
$$\text{Actual ACTs} = X + Y$$

In the **worst case** when $X = Y$
Total ACTs = $2 \times \text{Actual ACTs}$



Subarray C

$$N_{\text{odd}} = Y$$
$$N_{\text{even}} = Y$$

$$\text{Total ACTs} = Y$$
$$\text{Actual ACTs} = Y$$

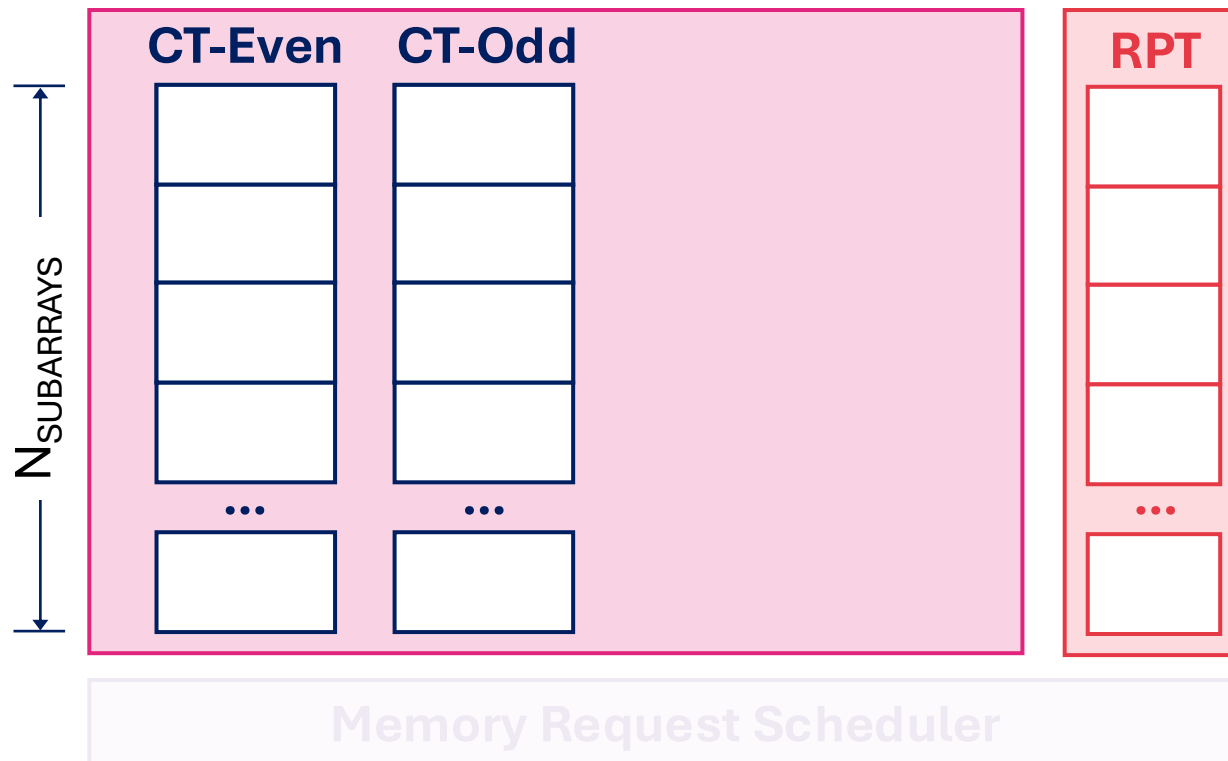
ColumnKeeper-D Key Idea

Separately count activations in odd and even columns
of each subarray to **mitigate ColumnDisturb**
at **low performance** and **energy overheads**

ColumnKeeper-D Overview

ColumnKeeper-D consists of:

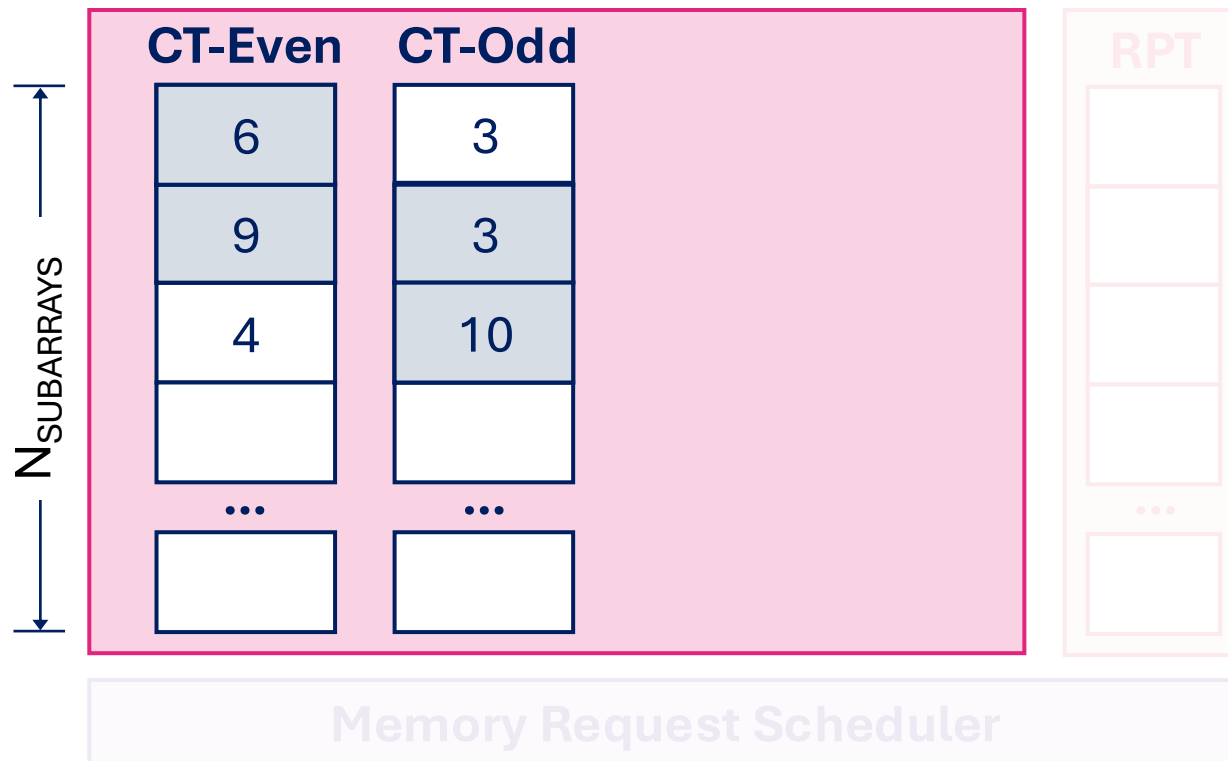
- **Counter Table – Even (CT-Even)**: Counts activations in even columns
- **Counter Table – Odd (CT-Odd)**: Counts activations in odd columns
- **Row Pointer Table (RPT)**: Points to the next row to be preventively refreshed in each subarray



ColumnKeeper-D Overview

Upon a row activation in subarray k, ColumnKeeper-D:

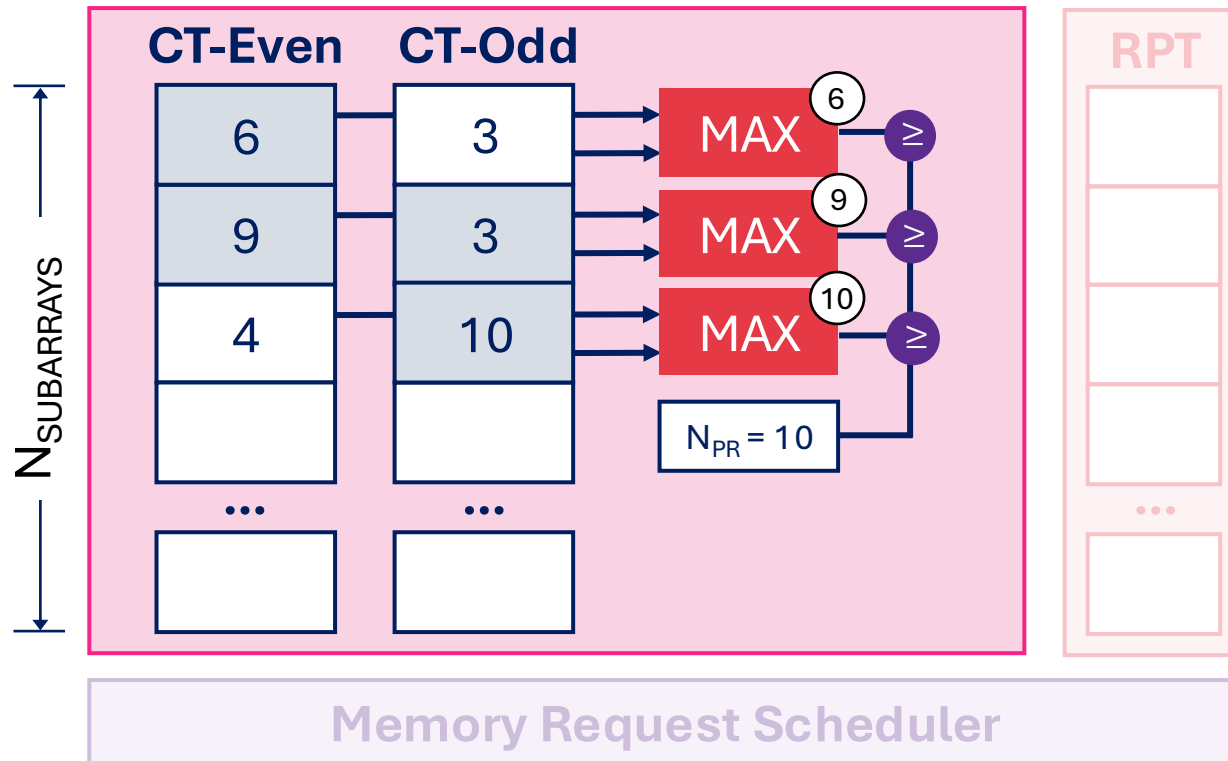
- Increments **both** the **CT-Even** and **CT-Odd** entries for subarray k
- Increments the **CT-Even** entry for the **previous** subarray
- Increments the **CT-Odd** entry for the **next** subarray



ColumnKeeper-D Overview

Upon a row activation in subarray k , ColumnKeeper-D:

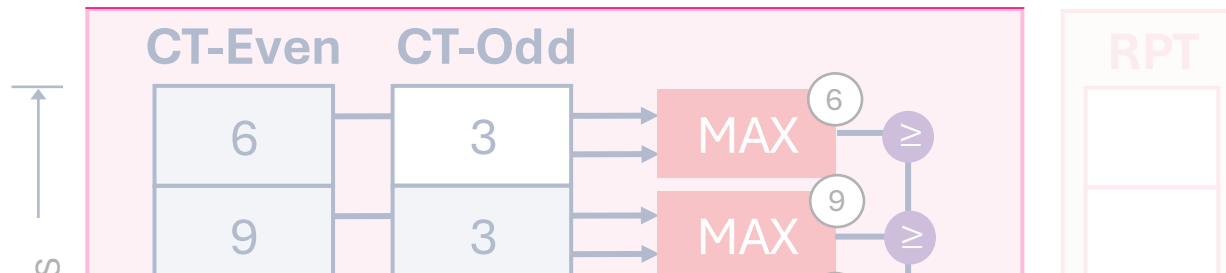
- Calculates the **maximum value** of the two counters for the three subarrays
- **Compares** the calculated value to a **predetermined threshold (N_{PR})**



ColumnKeeper-D Overview

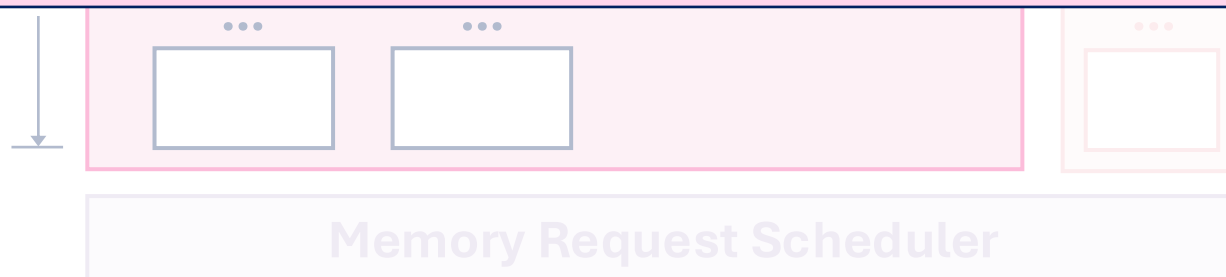
Row activation in subarray k:

- Calculates the **maximum value** of the two counters for the three subarrays
- **Compares** the calculated value to a **predetermined threshold (N_{PR})**



We set N_{PR} to N_{CD} divided by the subarray size

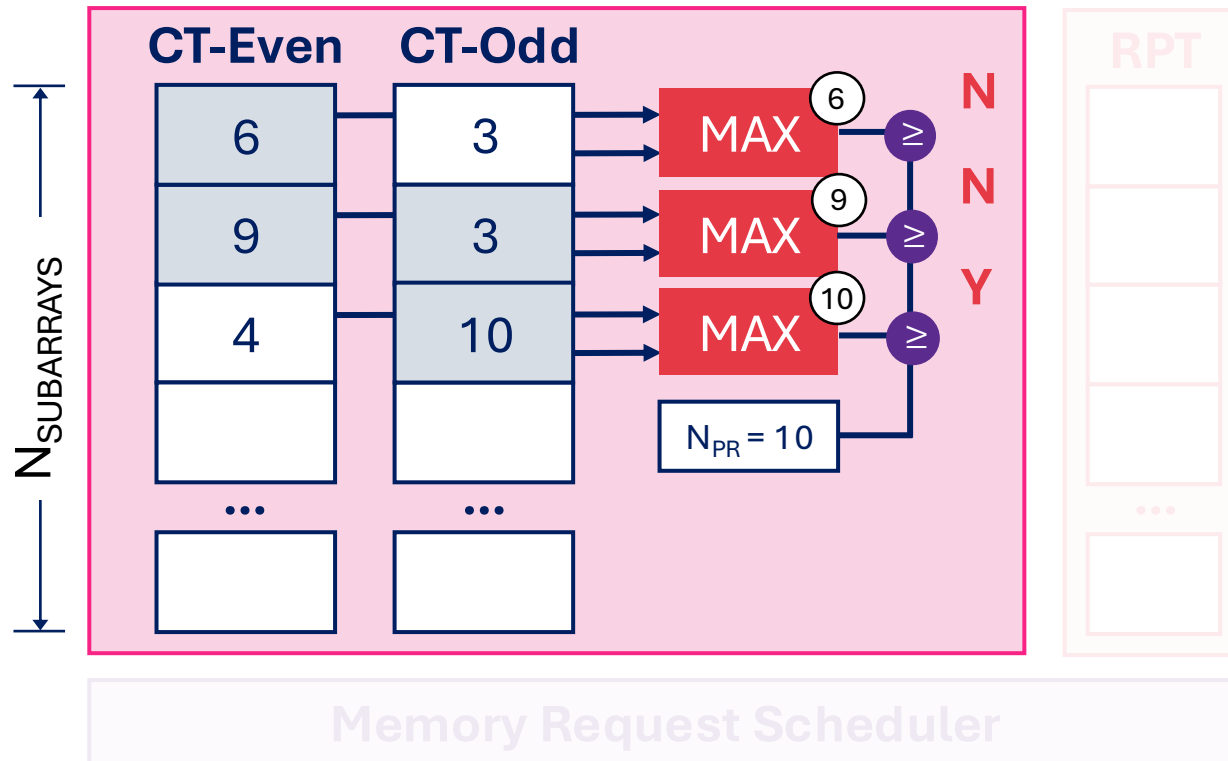
This ensures **all rows** are iteratively refreshed **before reaching N_{CD}**



ColumnKeeper-D Overview

Row activation in subarray k:

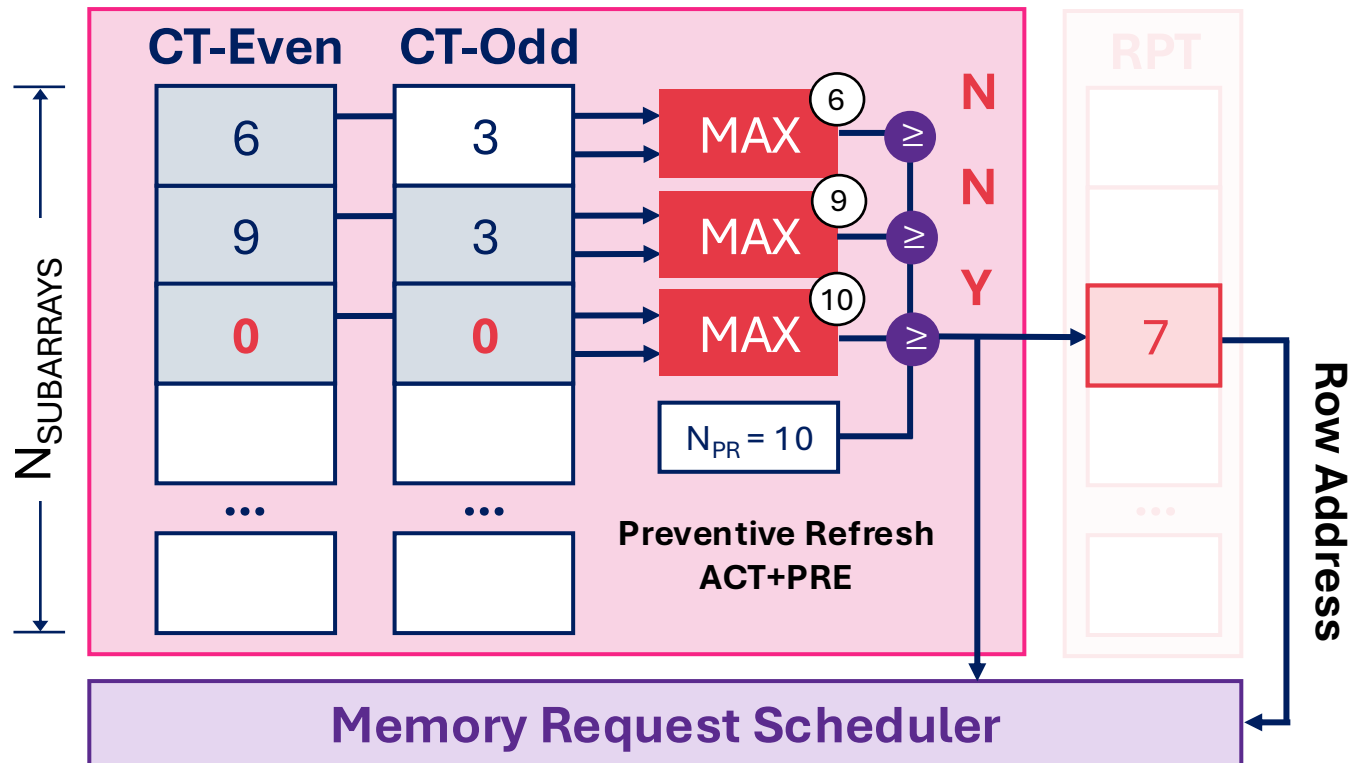
- Calculates the **maximum value** of the two counters for the three subarrays
- **Compares** the calculated value to a **predetermined threshold (N_{PR})**
- If the **maximum** counter value for a subarray **exceeds N_{PR}**
ColumnKeeper-D decides to **issue a preventive refresh** for **that subarray**



ColumnKeeper-D Overview

ColumnKeeper-D:

- **Issues** an **ACT** command followed by a **PRE** command to the **Row Address** pointed by the corresponding **RPT** entry
- **Increments** the **RPT** entry to point to the **next row** in that subarray
- **Zeroes** the **CT-Even** and **CT-Odd** entries for that subarray



Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

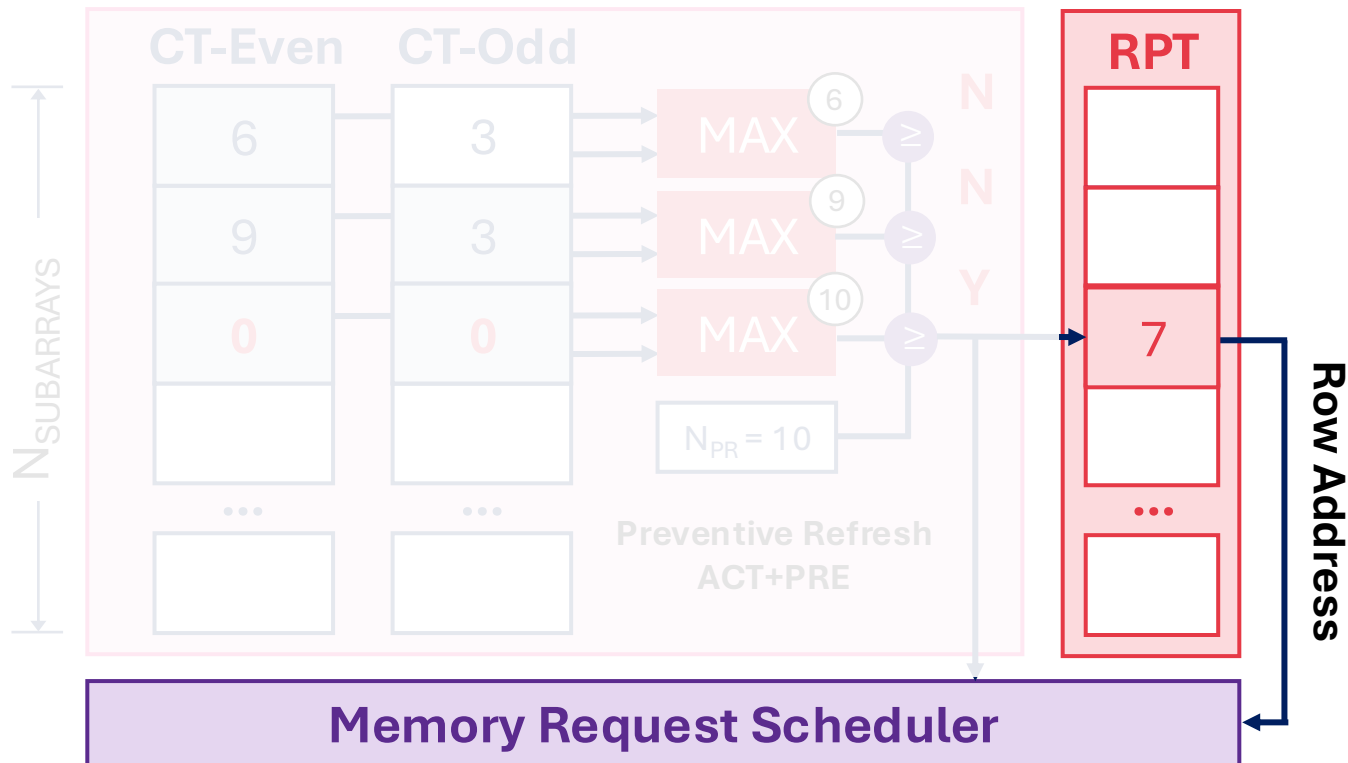
Conclusion

ColumnKeeper-P Key Idea

Probabilistically issue **preventive refreshes** to
avoid activation counting and
mitigate ColumnDisturb at **lower area overhead**

ColumnKeeper-P Overview

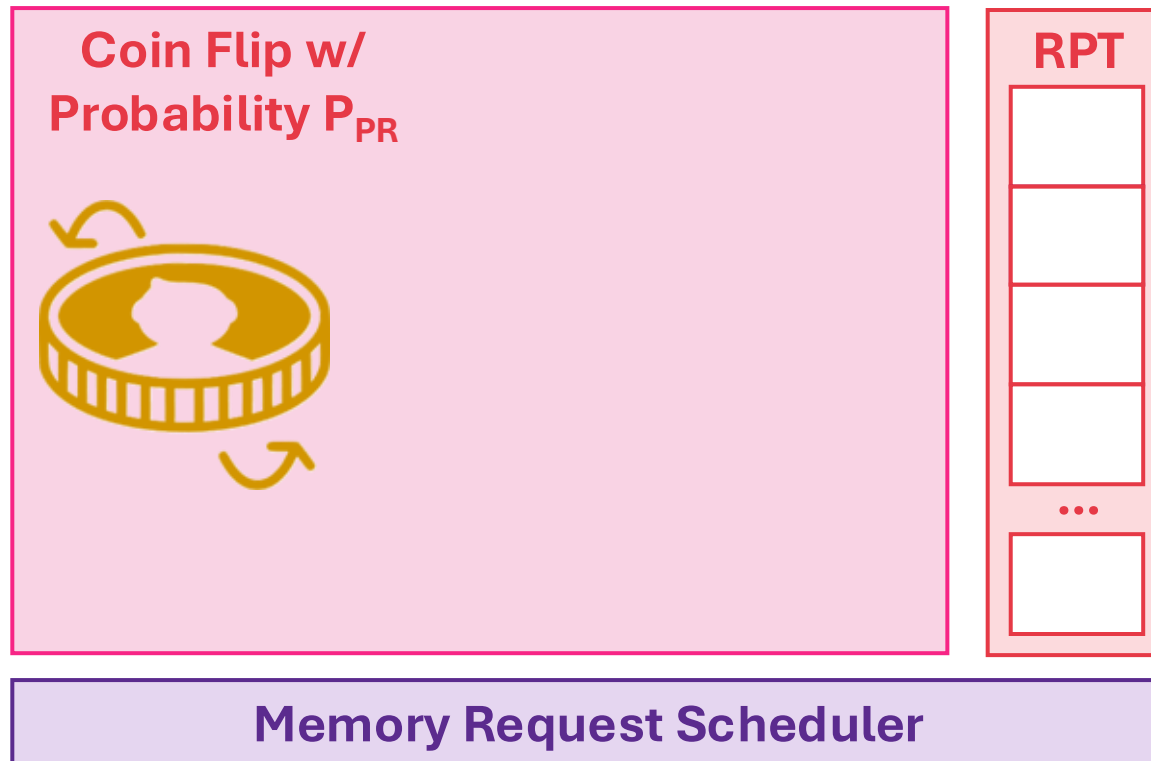
ColumnKeeper-P substitutes ColumnKeeper-D's **deterministic activation counting mechanism** with a **probabilistic one**



ColumnKeeper-P Overview

Upon a row activation in one subarray:

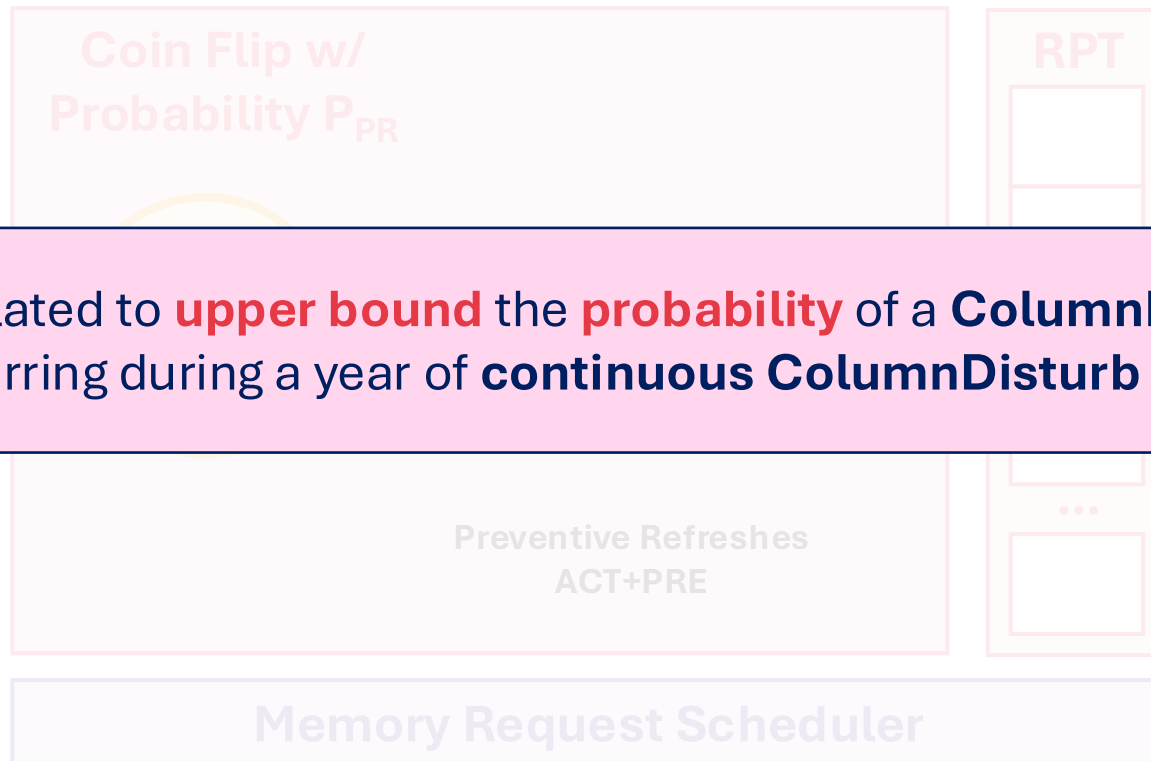
- **ColumnKeeper-P** flips a coin that lands heads up with a **predetermined probability (P_{PR})**



ColumnKeeper-P Overview

Upon a row activation in one subarray:

- **ColumnKeeper-P** flips a coin that lands heads up with a **predetermined probability (P_{PR})**

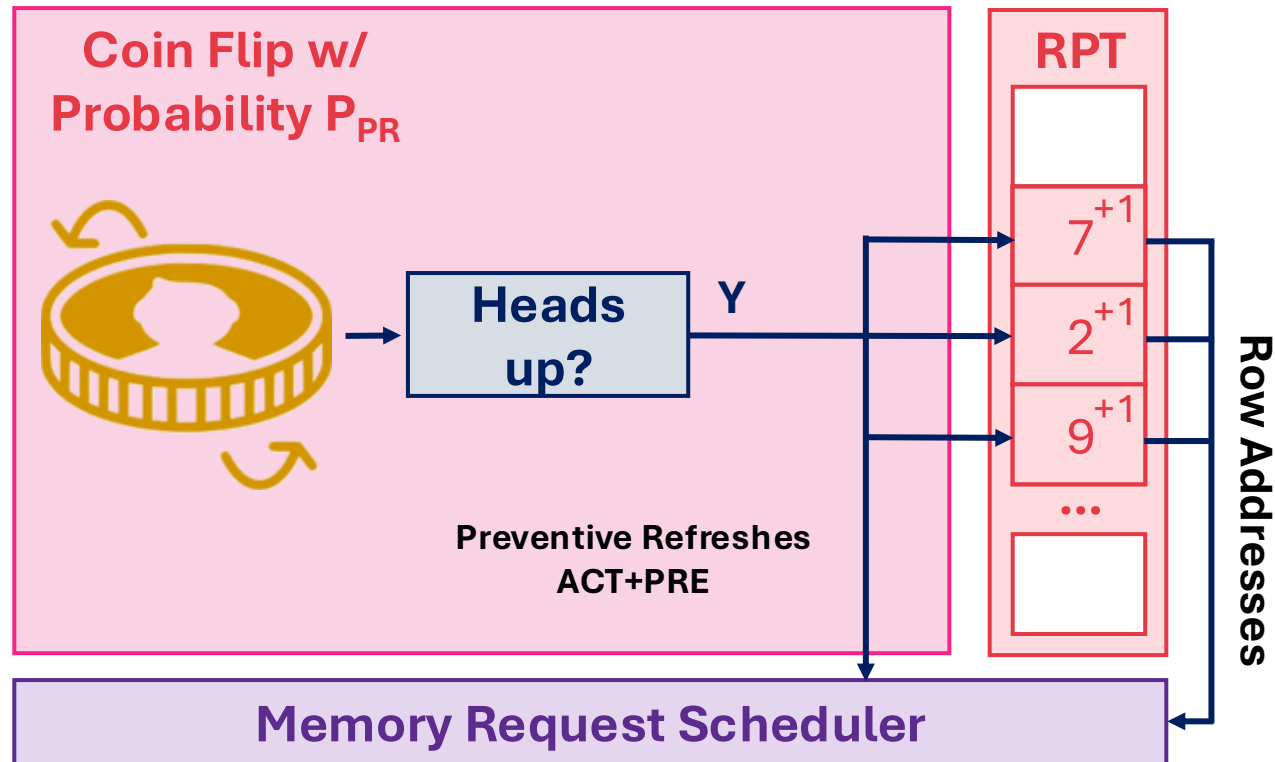


P_{PR} is calculated to **upper bound** the **probability** of a **ColumnDisturb** bitflip occurring during a year of **continuous ColumnDisturb** attack

ColumnKeeper-P Overview

Upon a row activation in one subarray:

- **ColumnKeeper-P** flips a coin that lands heads up with a **predetermined probability (P_{PR})**
- If the coin lands **heads up**, **ColumnKeeper-P** issues **preventive refreshes** for **that subarray and its neighbors** and **increments all three RPT entries**



More Details in the Paper

- Detailed **calculation of N_{PR}** for **ColumnKeeper-D**
- **Security analysis** of **ColumnKeeper-D**
- Detailed **calculation of P_{PR}** for **ColumnKeeper-P**
- **Monte-Carlo security analysis** of **ColumnKeeper-P**

Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

Conclusion

Methodology

Performance and energy consumption evaluation: cycle-level simulations using **Ramulator2 [Luo+, CAL 2023]** and **DRAMPower [Chandrasekar+, DATE 2013]**

System Configuration:

Processor 1 or 4 cores, Last-Level Cache: 2 MiB/core

DRAM 16GB DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 64K rows/bank, 1K rows/subarray

Mem. Ctrl. 64-entry read/write request queue,
Scheduling policy: FR-FCFS, Open-Row Policy
Address mapping: RoBaRaCoCh

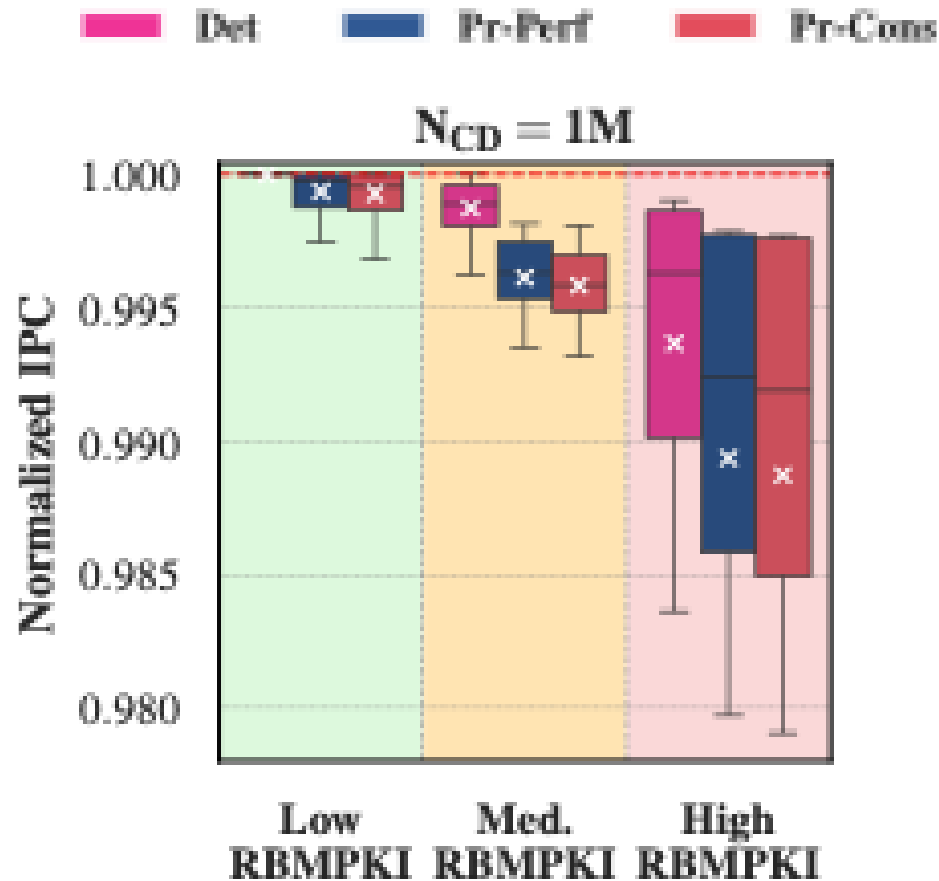
ColumnKeeper-D (*Det*) or

ColumnKeeper-P w/ yearly failure probability of 10^{-3} (*Pr-Perf*) or 10^{-12} (*Pr-Cons*)

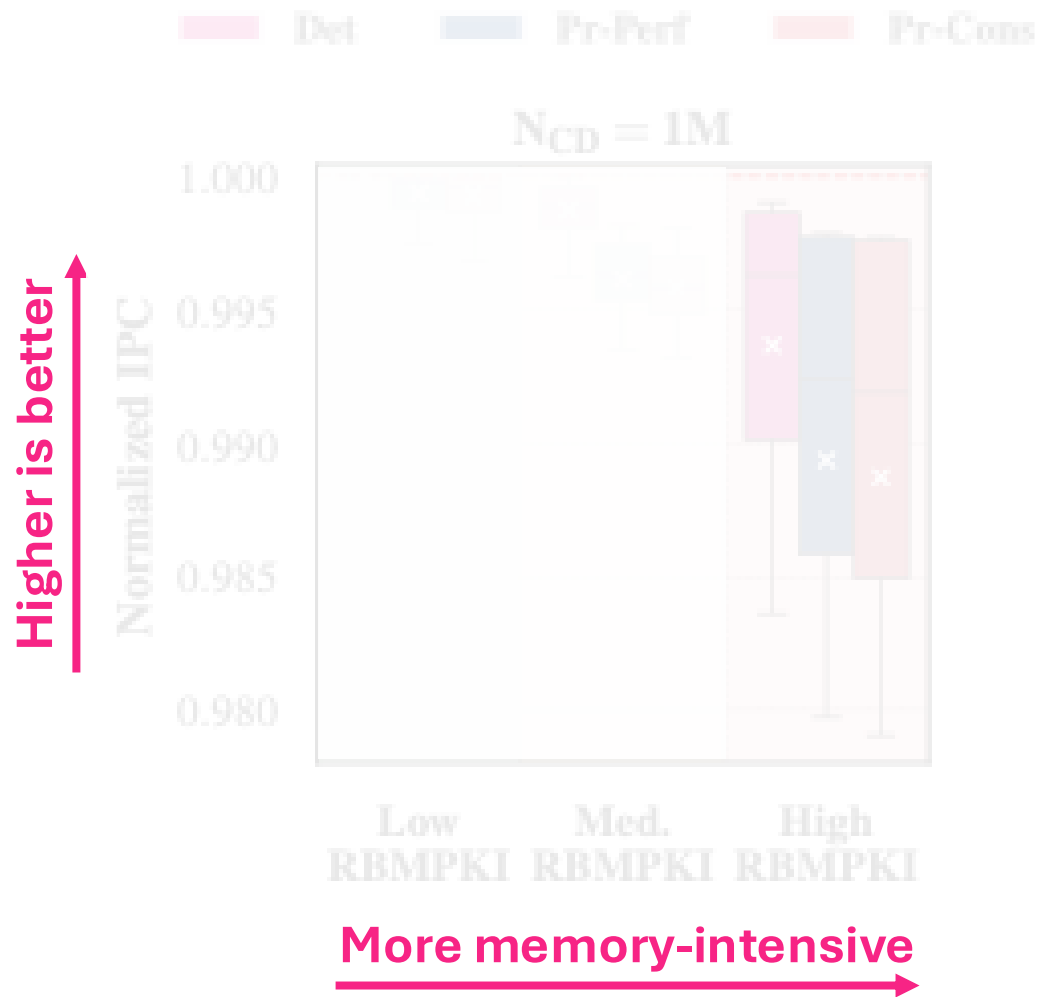
Workloads:

- **62 single-core workloads** from SPEC CPU2006, SPEC CPU2017, TPC, MediaBench, and YCSB categorized by **Row Buffer Misses per-kilo-instructions (RBMPKI)**
- **60 quad-core workload mixes** synthesized from the resulting RBMPKI categories

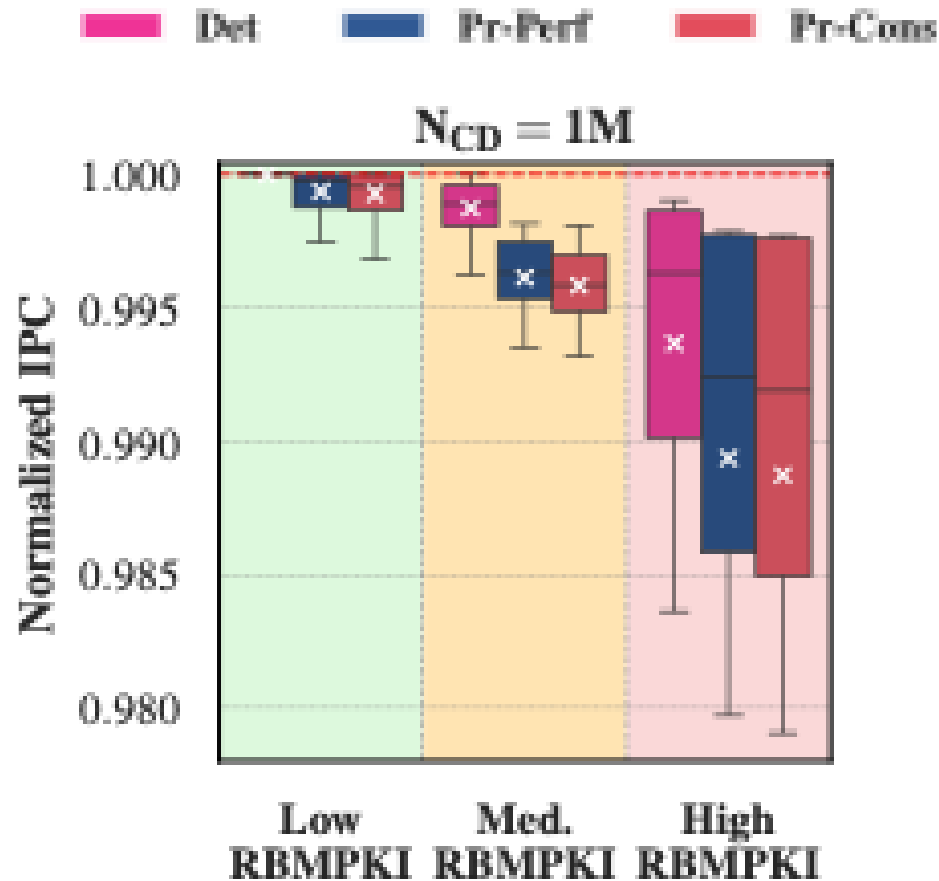
Evaluation: Single-Core Performance



Evaluation: Single-Core Performance

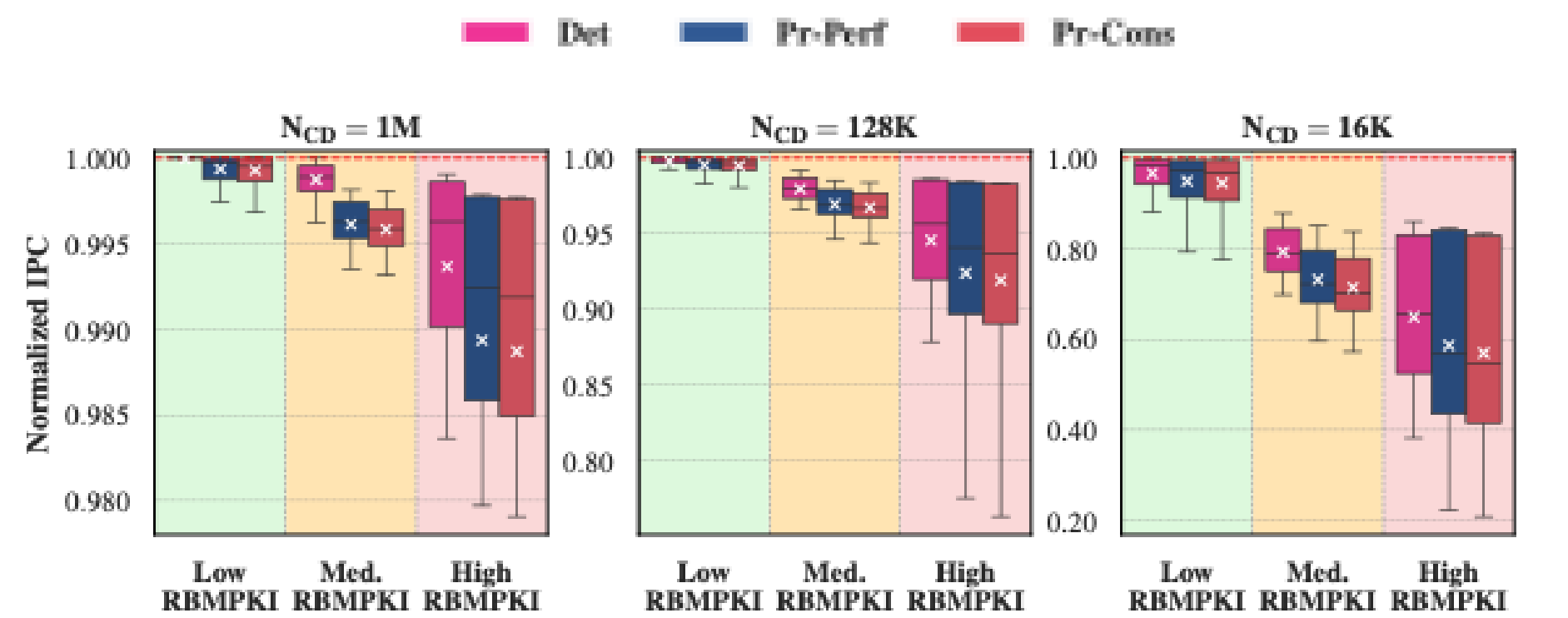


Evaluation: Single-Core Performance

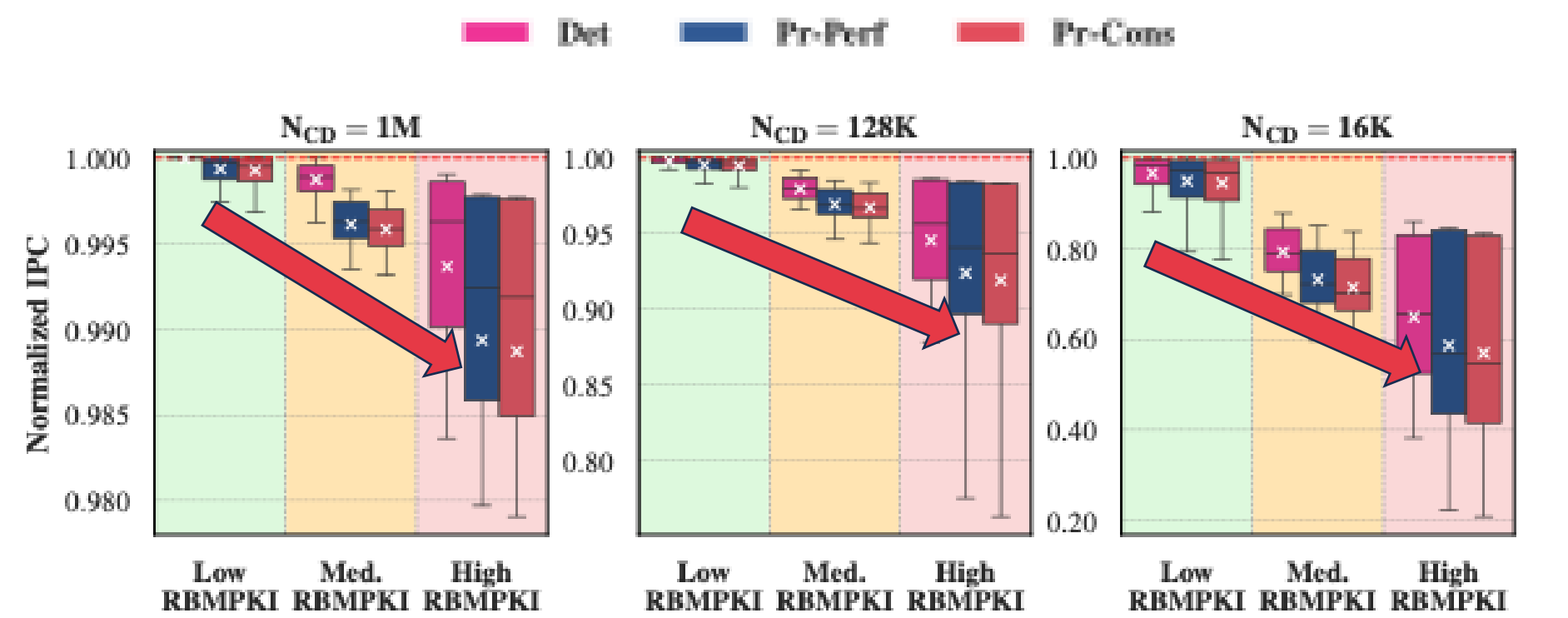


ColumnKeeper-D and -P incur low (<1%) average performance overheads for current ColumnDisturb thresholds (1M)

Evaluation: Single-Core Performance



Evaluation: Single-Core Performance



Performance overheads increase for more memory-intensive workloads

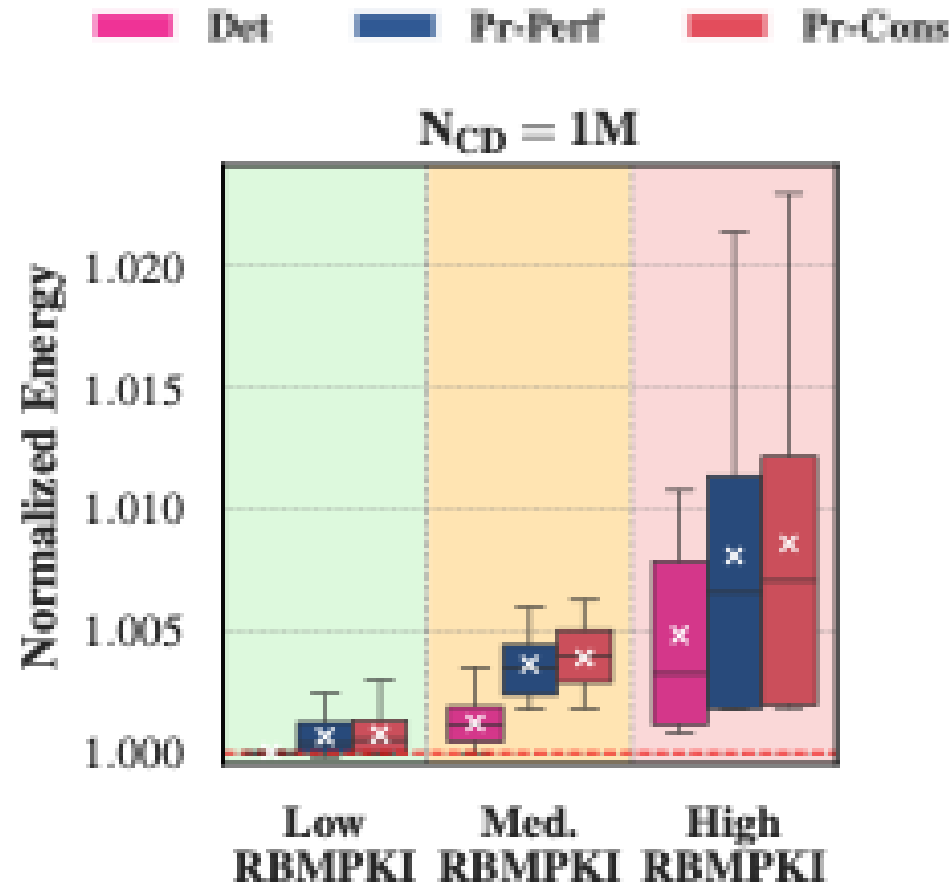
Evaluation: Single-Core Performance



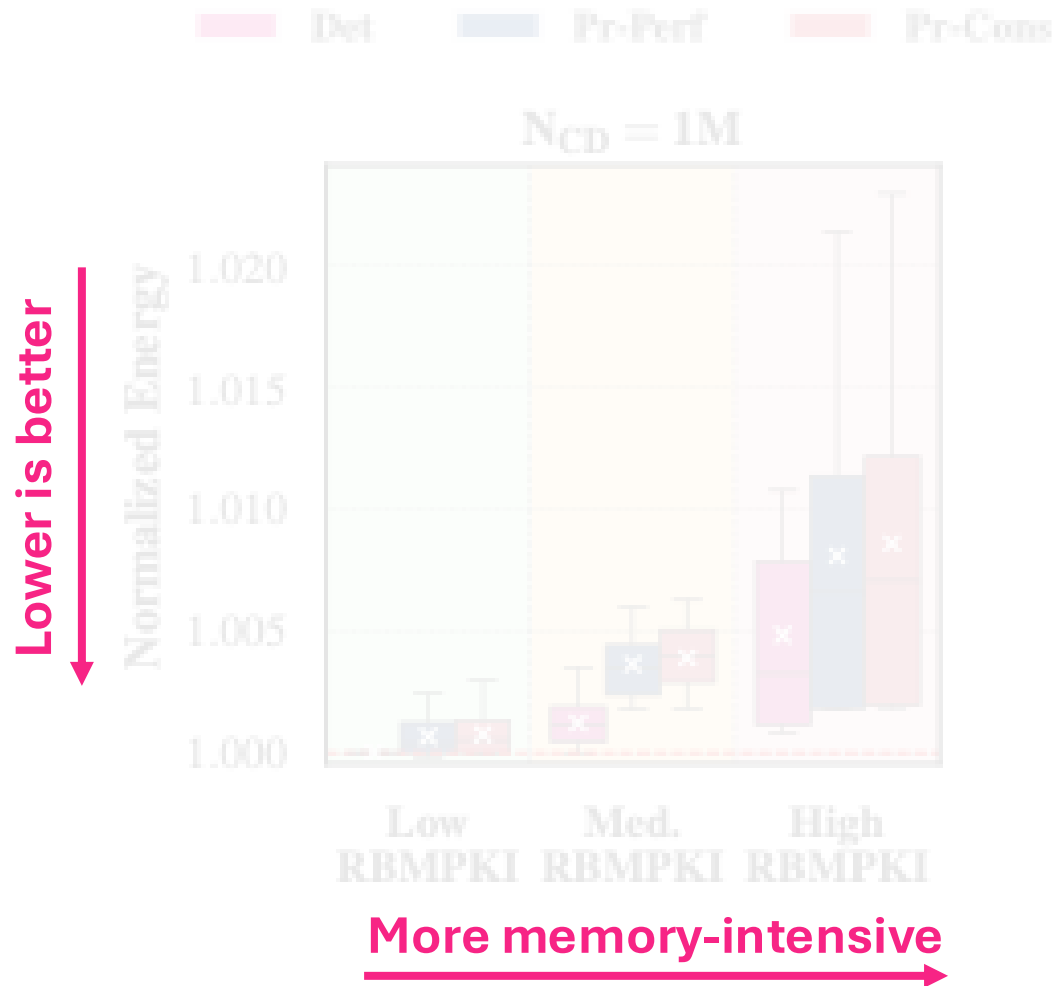
Performance overheads increase for more **memory-intensive** workloads

All ColumnKeeper mechanisms incur significant performance overheads
at low **ColumnDisturb** thresholds of $N_{CD}=16K$

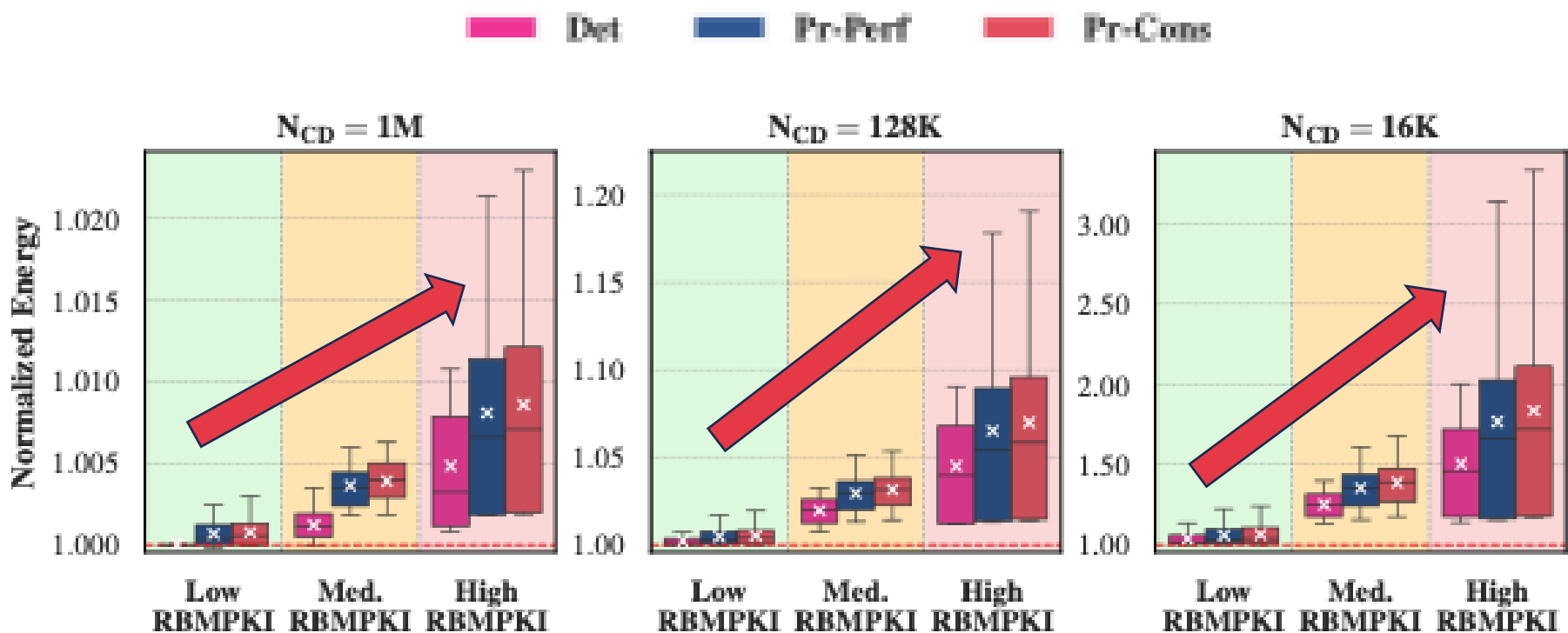
Evaluation: DRAM Energy (Single-Core)



Evaluation: DRAM Energy (Single-Core)



Evaluation: DRAM Energy (Single-Core)

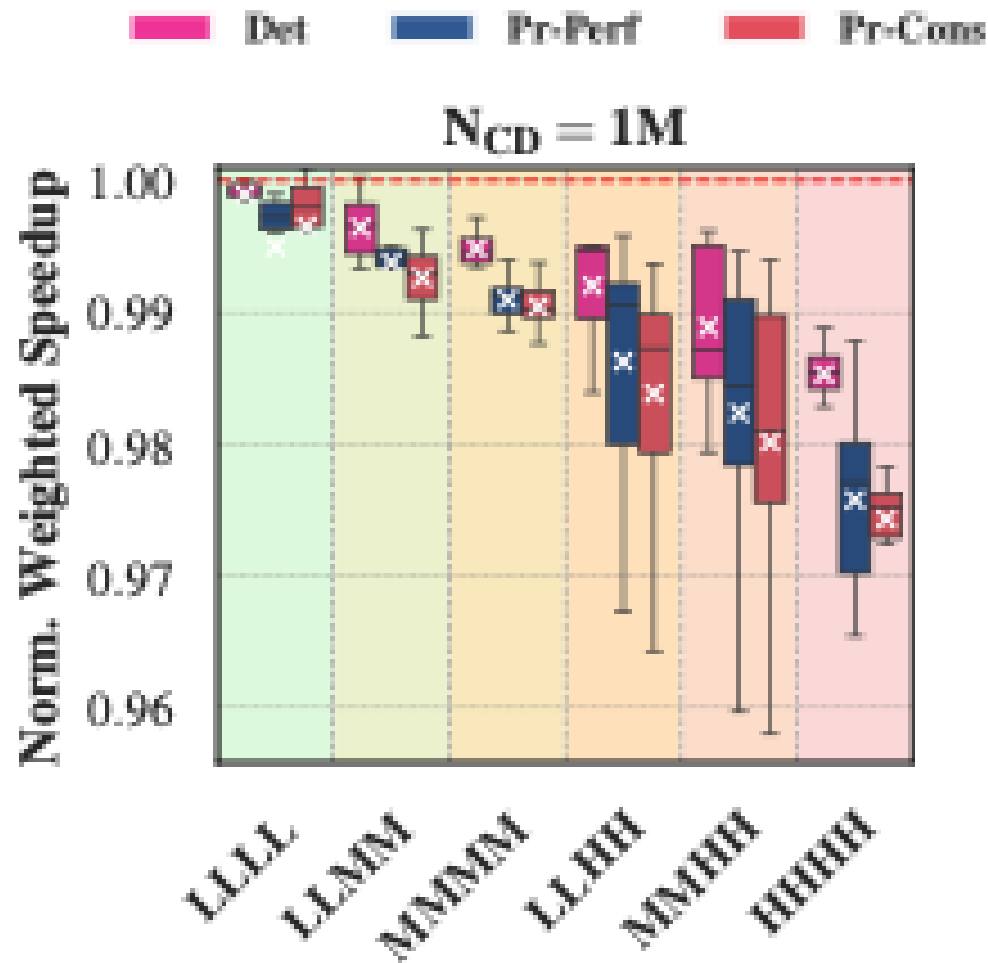


Energy overheads increase for more **memory-intensive** workloads

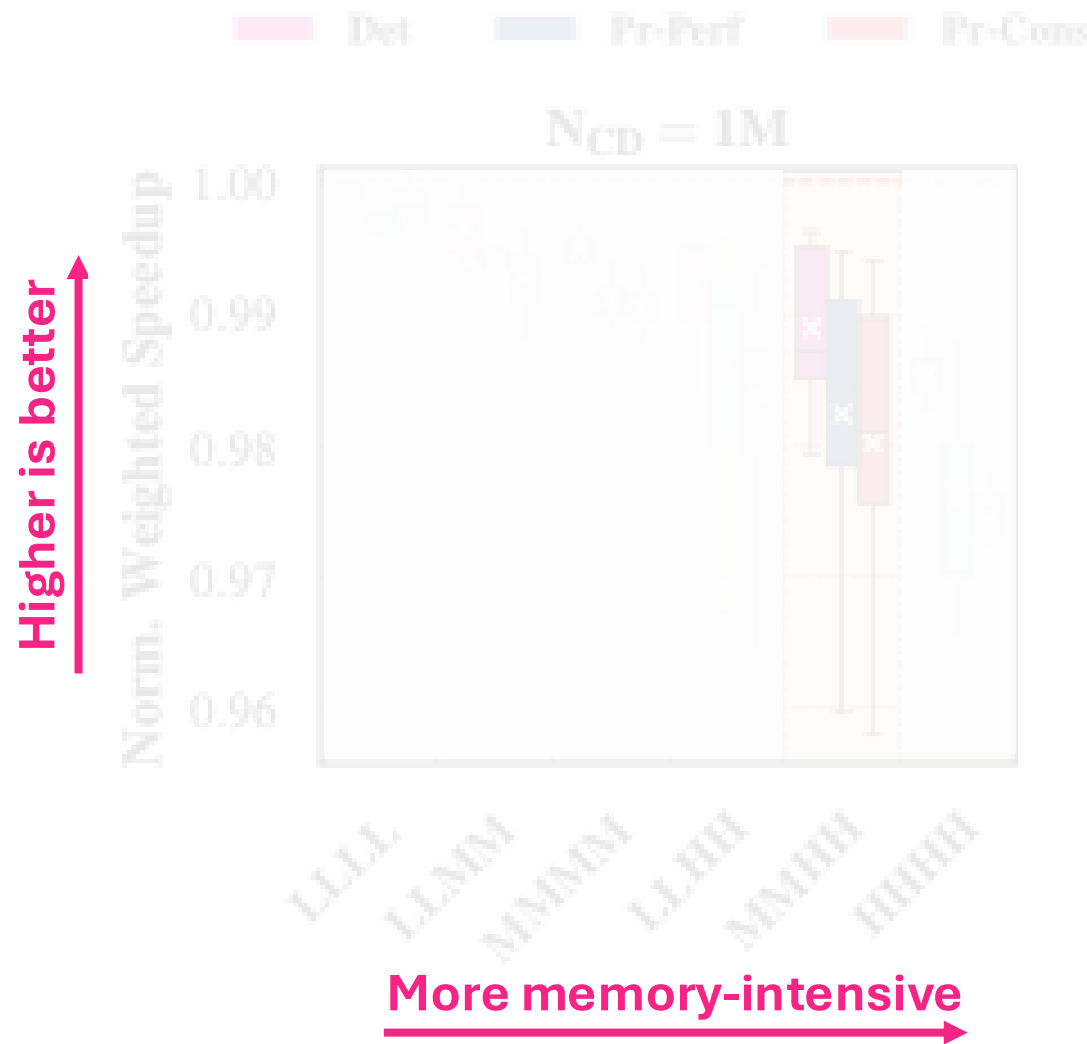
Low energy overheads for **current** thresholds

Significant energy overheads for **future** low thresholds

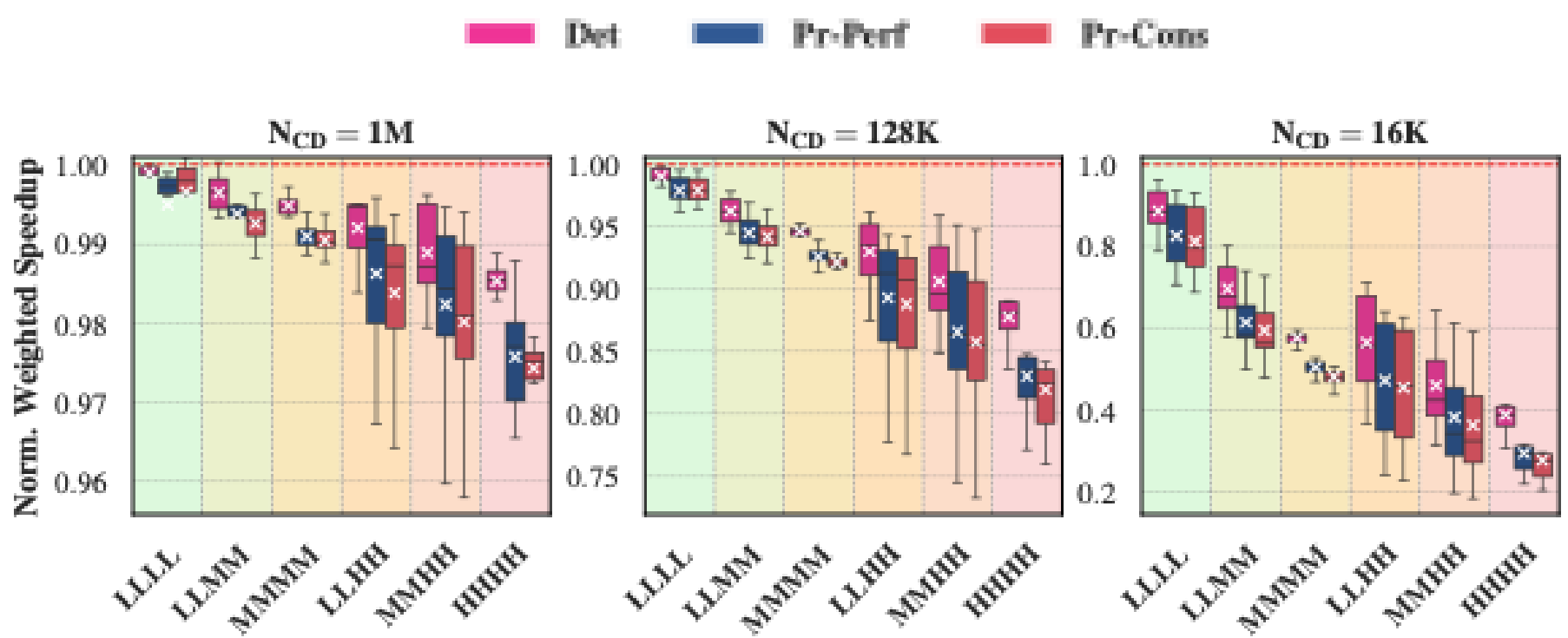
Evaluation: Multi-Core Performance



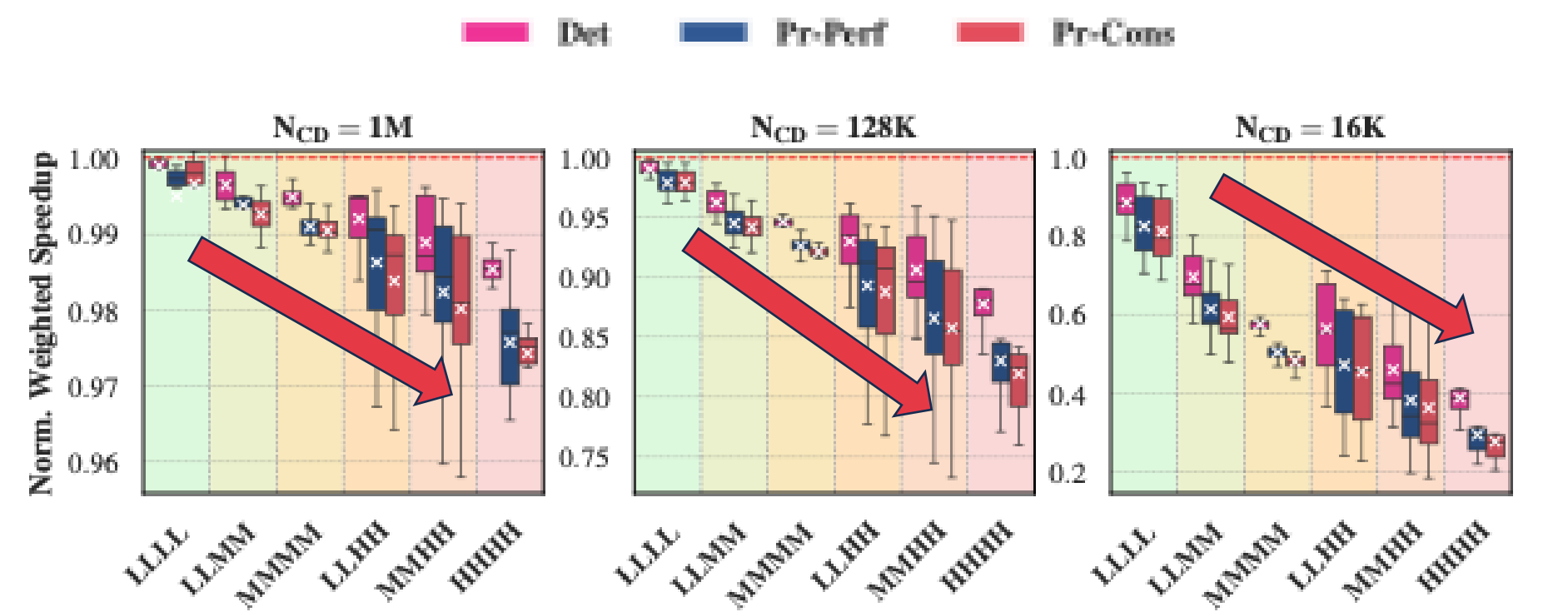
Evaluation: Multi-Core Performance



Evaluation: Multi-Core Performance



Evaluation: Multi-Core Performance

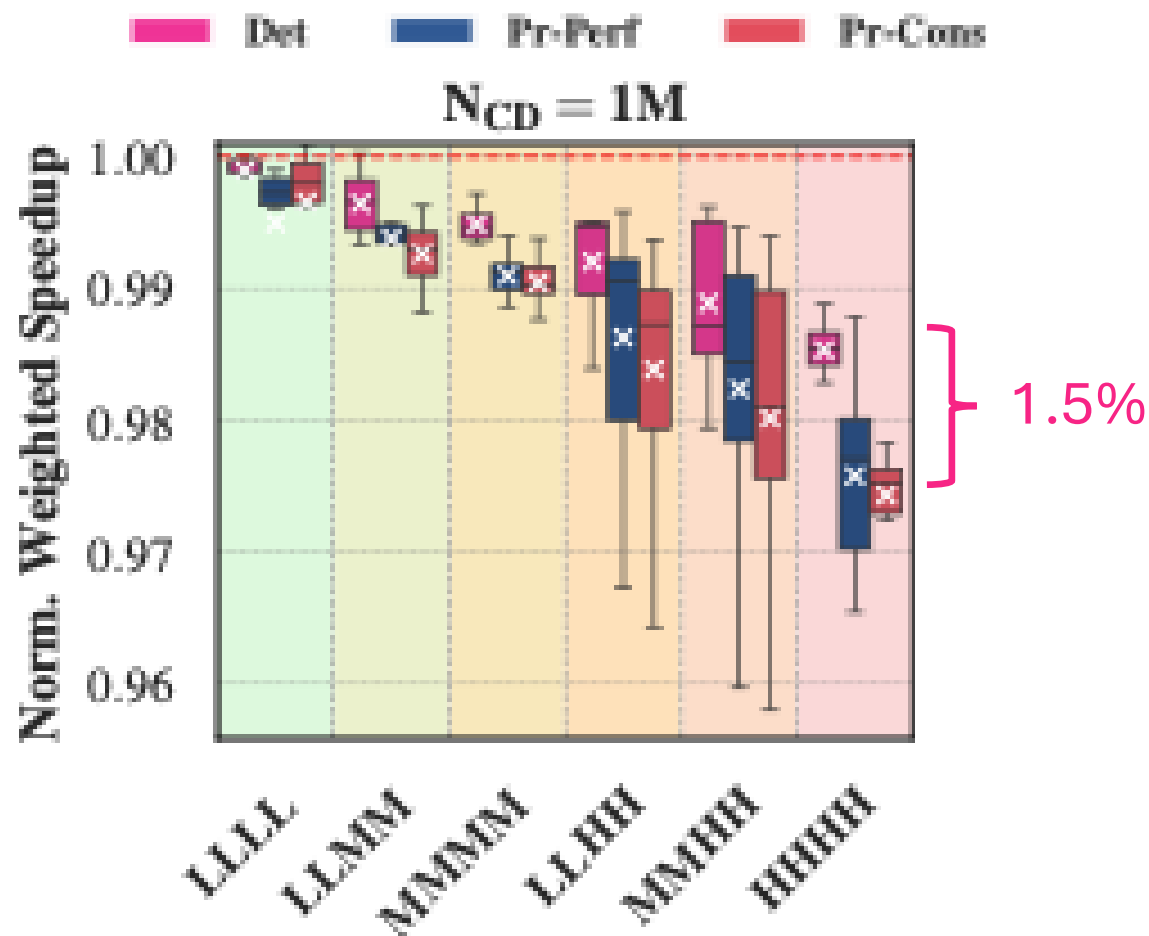


Performance overheads increase for memory-intensive workload mixes

Low performance overheads for current thresholds

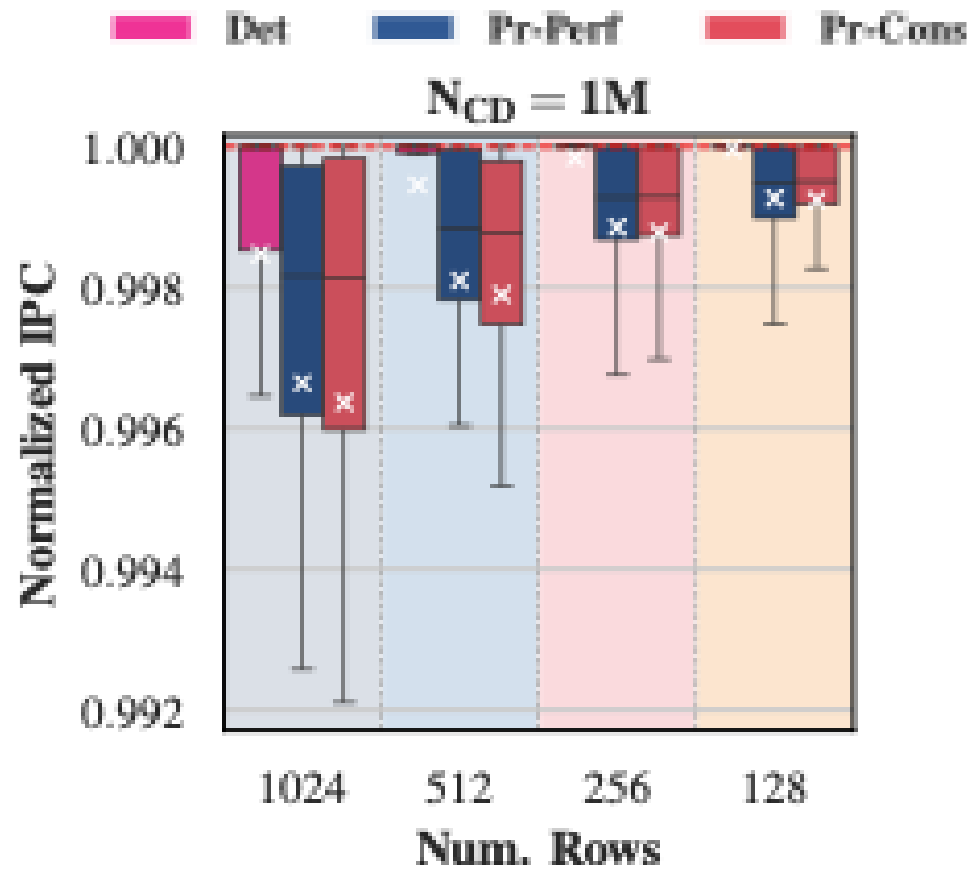
Significant performance overheads for future low thresholds

Evaluation: Multi-Core Performance

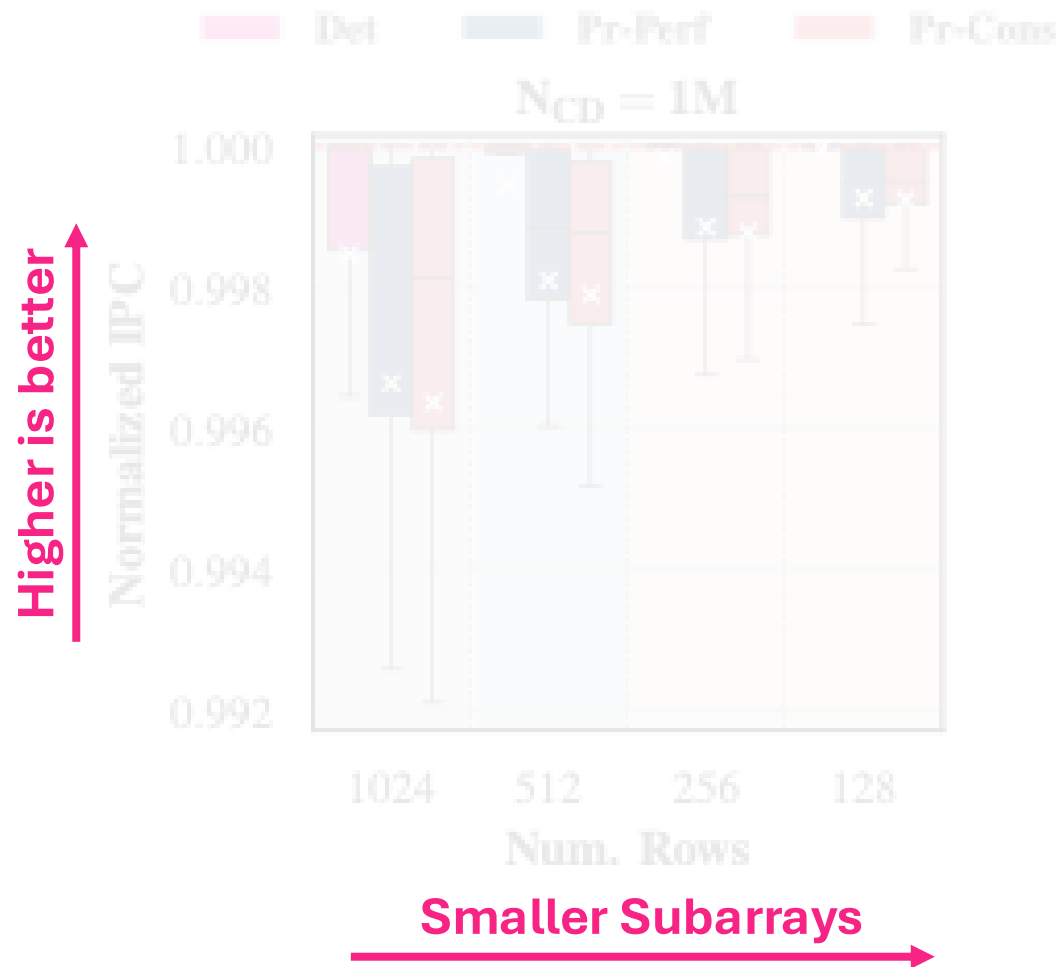


More pronounced difference between the deterministic and the probabilistic mechanisms than single-core

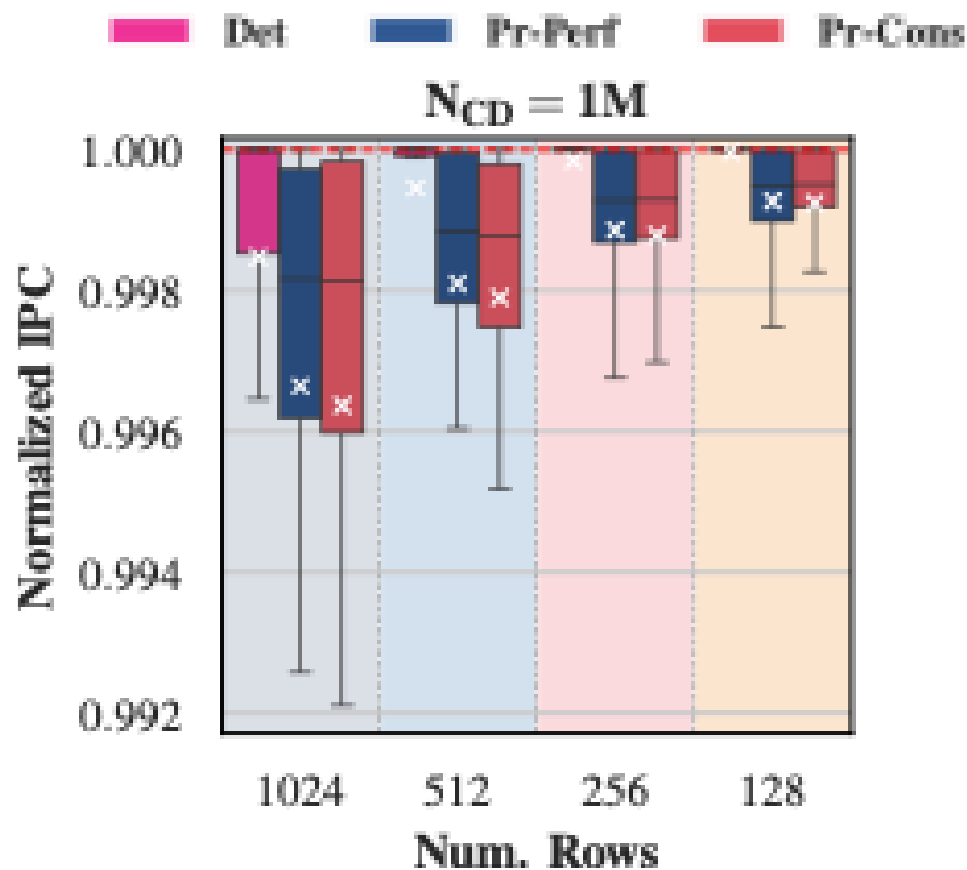
Evaluation: Sensitivity to Subarray Size



Evaluation: Sensitivity to Subarray Size

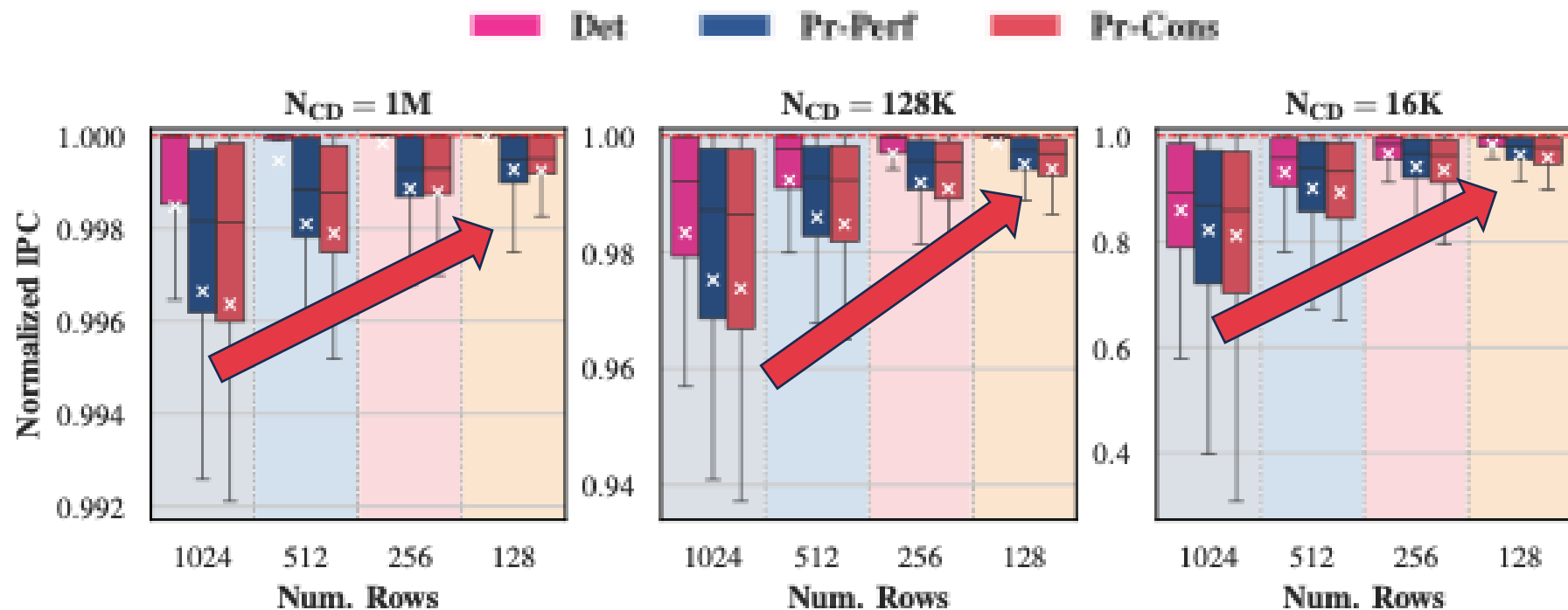


Evaluation: Sensitivity to Subarray Size



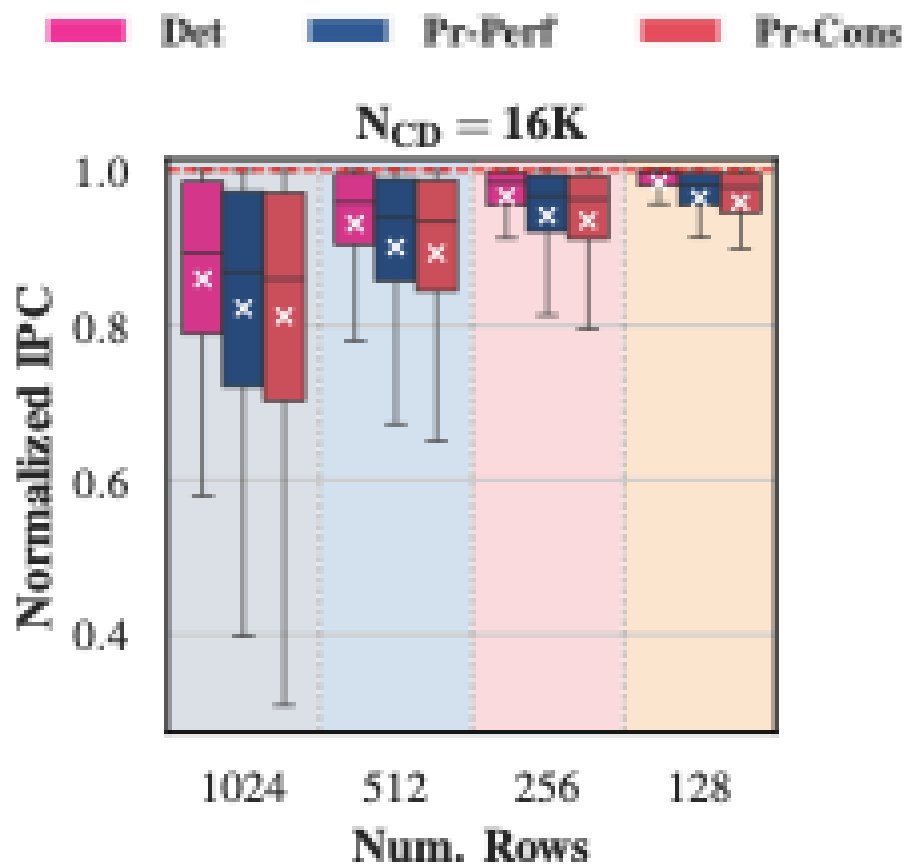
Smaller subarrays (128 rows) eliminate the overheads of ColumnKeeper for current ColumnDisturb thresholds (1M)

Evaluation: Sensitivity to Subarray Size

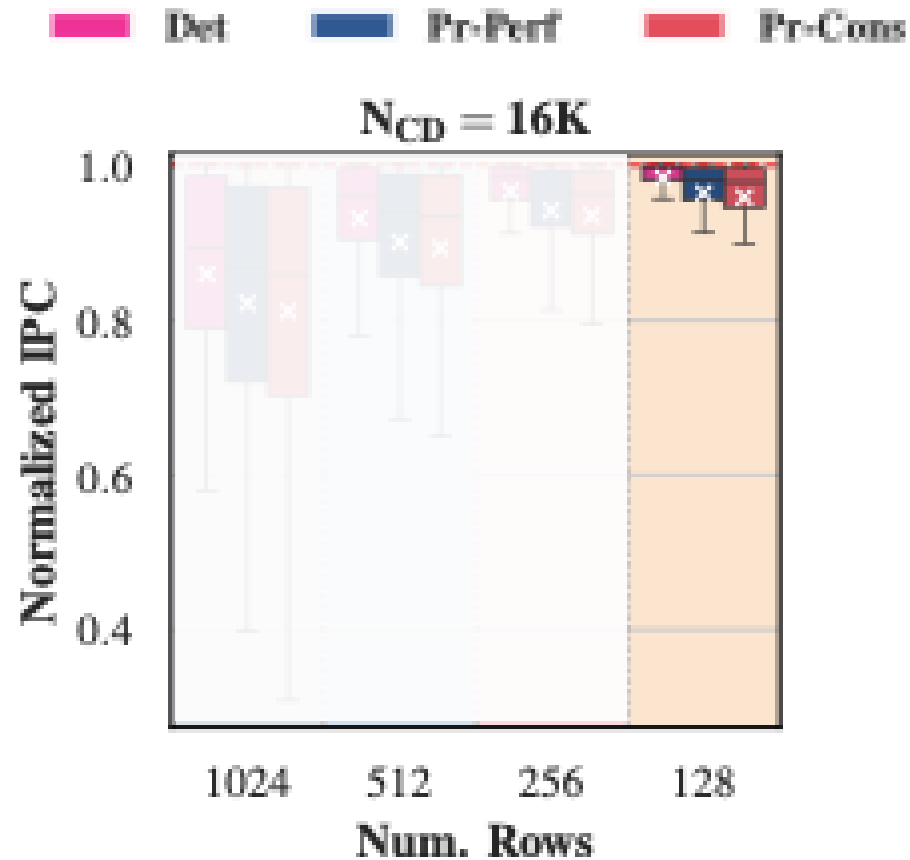


Smaller subarrays reduce the overheads of ColumnKeeper for all ColumnDisturb thresholds

Evaluation: Sensitivity to Subarray Size



Evaluation: Sensitivity to Subarray Size



Smaller subarrays reduce the overheads of ColumnKeeper even at very low thresholds

Evaluation: Area and Power Overheads

ColumnKeeper requires:

- **7.5 KB** of SRAM for **ColumnKeeper-D**
- **2.5 KB** of SRAM for **ColumnKeeper-P**

Hardware Implementation & Synthesis:

- **ColumnKeeper** implemented in **Verilog**
- **OpenRoad + NanGate 45nm** Open-Source Cell Library

Area Overheads:

- **0.1 mm²** for **ColumnKeeper-D**
- **0.03 mm²** for **ColumnKeeper-P**

Power Overheads:

- **50 mW** for **ColumnKeeper-D**
- **15 mW** for **ColumnKeeper-P**

More in the Paper

- Evaluation of ColumnKeeper with **Subarray-level Parallelism** [Kim+, ISCA 2012]
- **Impact** of physical page allocation policy on the **performance overheads** of ColumnKeeper
- **Performance evaluation** of ColumnKeeper integrated with **RowHammer mitigation mechanisms**
- **Performance evaluation** under **adversarial workloads**
- An **in-DRAM implementation** of our design

ColumnKeeper: Efficient Solutions to the ColumnDisturb Vulnerability in DRAM-based Systems

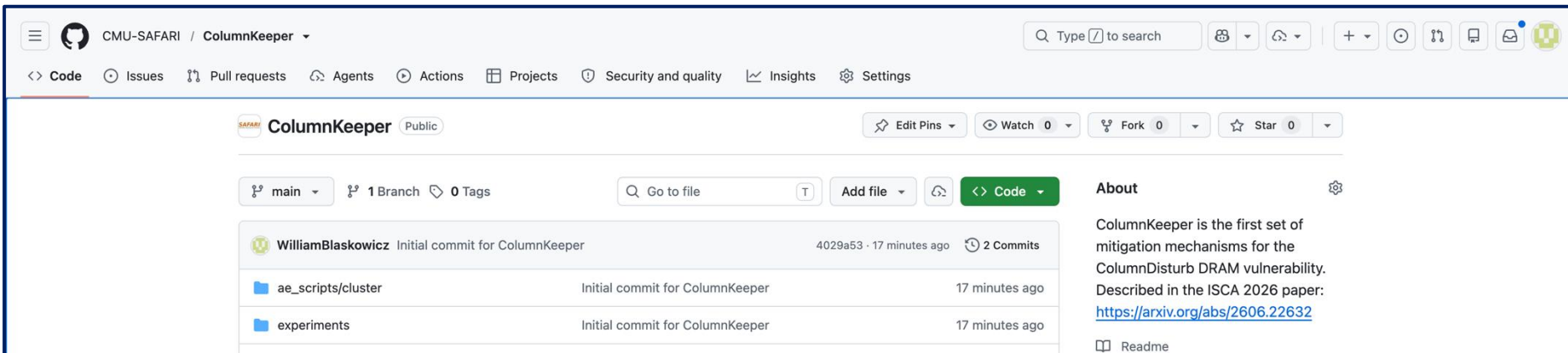
Andreas Kosmas Kakolyris¹ F. Nisa Bostancı¹ Ataberk Olgun¹ İsmail Emir Yüksel¹
Harsh Songara¹ Konstantinos Marios Sgouras¹ Umut Başer^{1,2}
Konstantinos Kanellopoulos¹ A. Giray Yağlıkçı³ Onur Mutlu¹

¹ETH Zürich ²TOBB ETÜ ³CISPA



<https://arxiv.org/pdf/2606.22632>

Artifact Evaluated & Open-Source



<https://github.com/CMU-SAFARI/ColumnKeeper>

Outline

Background

Motivation

ColumnKeeper-D (Deterministic)

ColumnKeeper-P (Probabilistic)

Evaluation

Conclusion

Conclusion

Goal: Mitigate ColumnDisturb bitflips at low overhead

Key Idea: Track row activations **at subarray granularity** and iteratively refresh **all affected rows** in a subarray

ColumnKeeper-D: Uses **separate** activation counters for **odd** and **even columns** of each subarray

ColumnKeeper-P: **Probabilistically issues** refreshes to one row in **three consecutive subarrays** upon a row activation in the middle subarray

Key Results:

- **ColumnKeeper prevents ColumnDisturb bitflips with low overheads for current and near-future thresholds**
- **ColumnKeeper can prevent ColumnDisturb bitflips at low overheads for low thresholds with changes to DRAM microarchitecture**

ColumnKeeper: Efficient Solutions to the ColumnDisturb Vulnerability in DRAM-based Systems

Paper



Code





ColumnKeeper: Efficient Solutions to the ColumnDisturb Vulnerability in DRAM-based Systems

Backup Slides

Andreas Kosmas Kakolyris

F. Nisa Bostancı

Ataberk Olgun

İsmail Emir Yüksel

Harsh Songara

Konstantinos Marios Sgouras

Umut Başer

Konstantinos Kanellopoulos

A. Giray Yağlıkçı

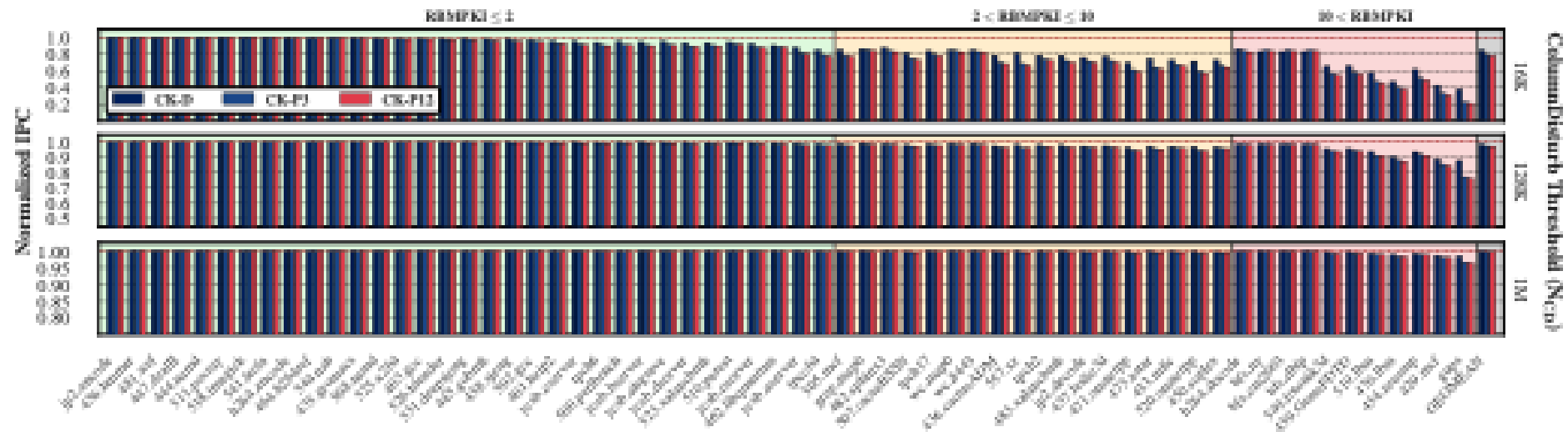
Onur Mutlu

SAFARI

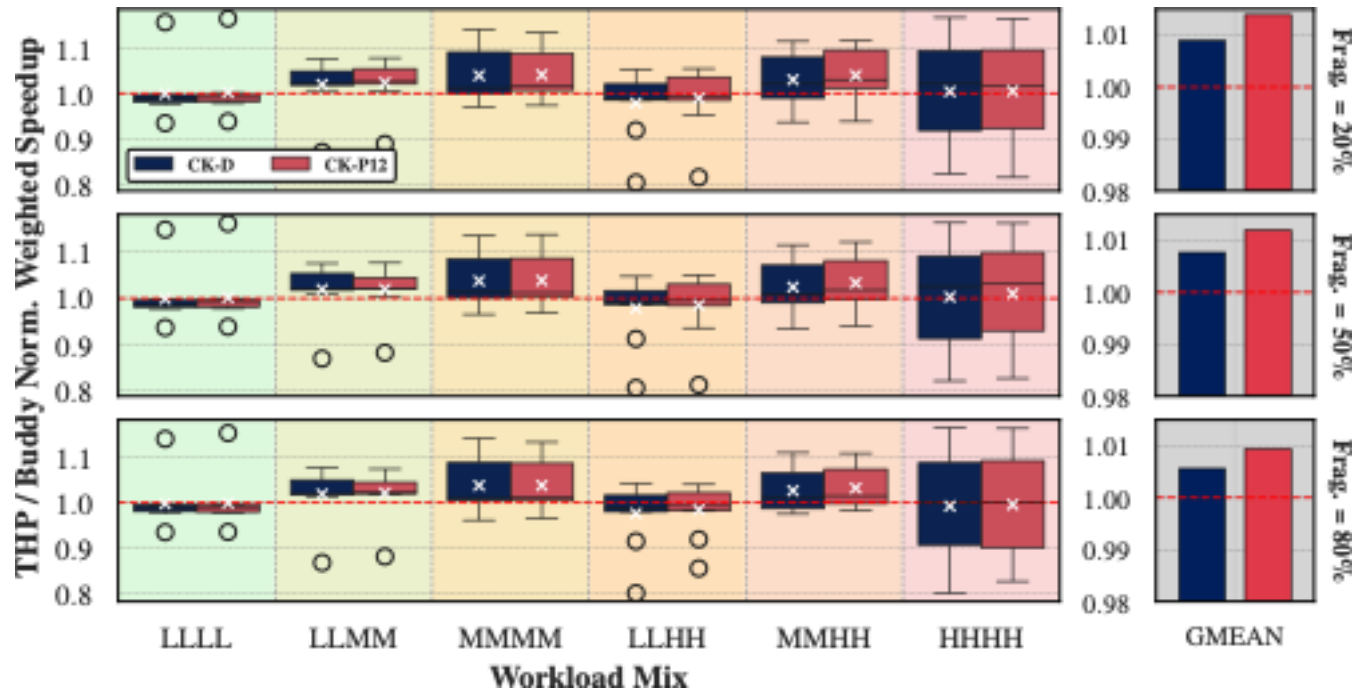
ETH zürich



Evaluation: Detailed Single-Core Results

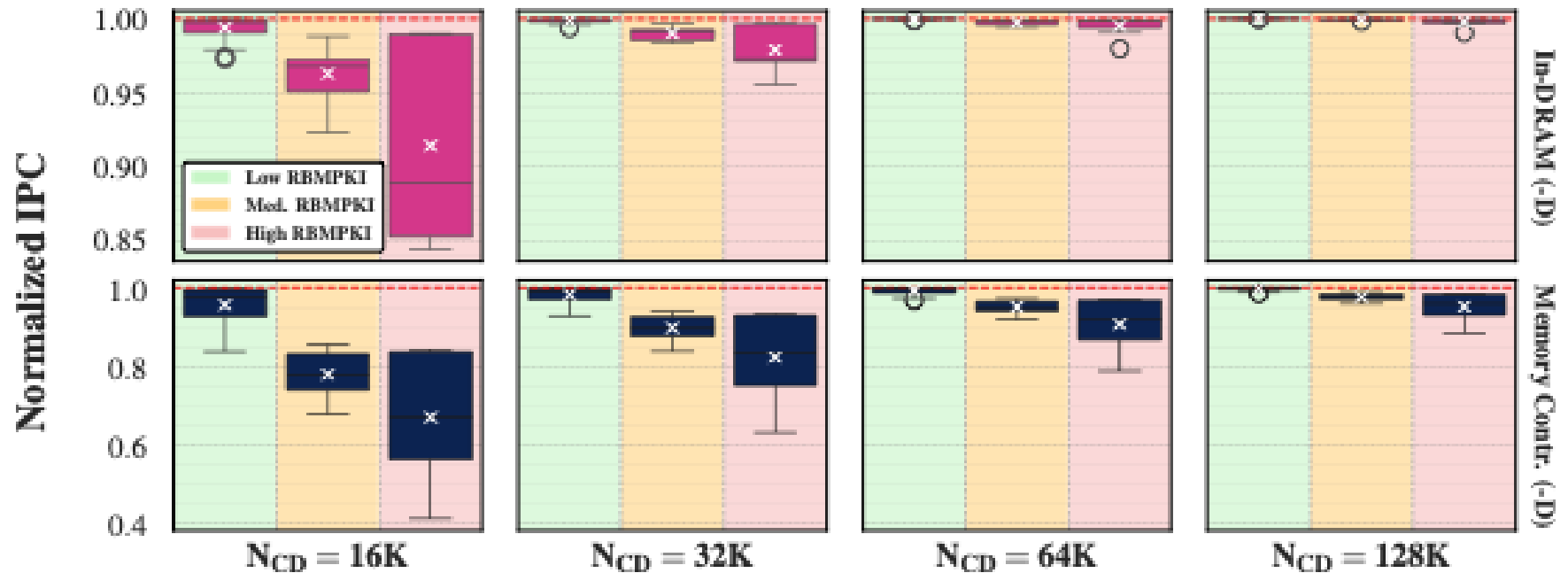


Evaluation: Sensitivity to Page Allocation



No significant difference in performance between the two allocators

Evaluation: In-DRAM Implementation



The in-DRAM implementation of ColumnKeeper-D reduces its overheads at low ColumnDisturb thresholds

ColumnKeeper-P: P_{PR} Calculation

Methodology

- Similar to the one proposed in HiRA [Yaglikci+, MICRO 2022]
- Considers the maximum number of attacks that can happen in a refresh window
- At least one of these attacks has to succeed

To induce bitflips: The number of preventive issues issued to a subarray must be lower than the subarray size (S) when the ColumnDisturb threshold is reached

Total number of preventive refreshes issued after N activations (M_N):

- Sum of Bernoulli random variables
- M_N follows a binomial distribution

$$P(M_N \leq k) = I_q(N - k, 1 + k), q = 1 - P_{PR}$$

ColumnKeeper-P: P_{PR} Calculation

We substitute:

- $N = N_{CD} - 1$
- $k = S - 1$

A single ColumnDisturb attack succeeds with probability P_1 :

$$P_1 = P(M_{N_{CD}} < S) = P(M_{N_{CD}} \leq S - 1) = I_q(N_{CD} - S, S)$$

Probability of a successful attack within a refresh window based on maximum number of attacks:

$$P_{REFW} = 1 - (1 - P_1)^A, A = \frac{t_{REFW}}{t_F} + 1, t_F = (1 + 3S) \cdot t_{RC}$$

Probability of a successful attack within a year (P_Y):

$$P_Y = 1 - (1 - P_{REFW})^{492.7 \cdot 10^6}$$

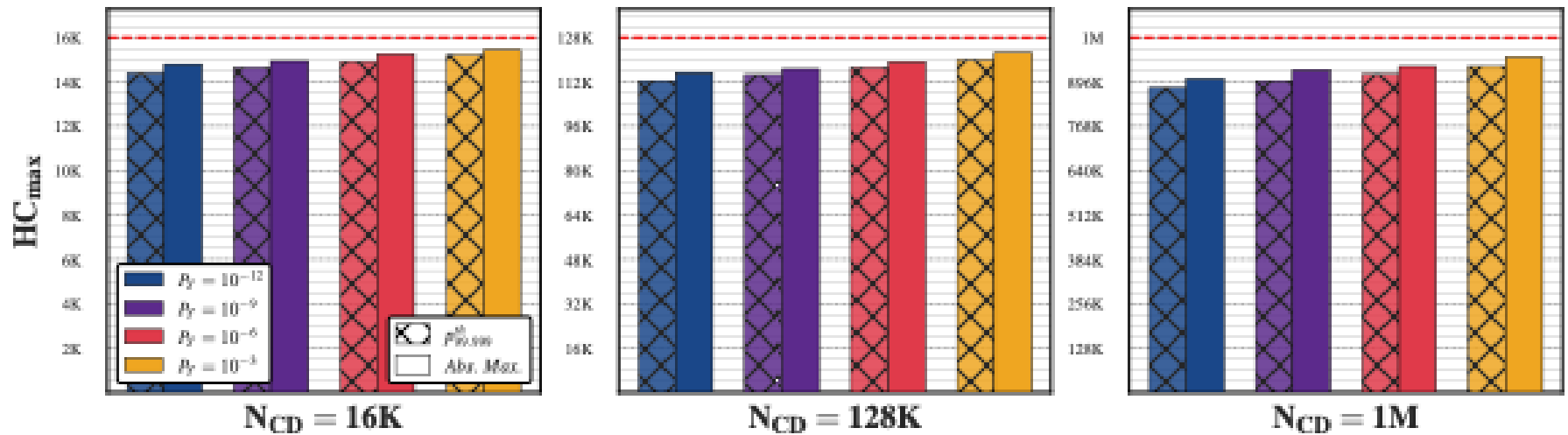
ColumnKeeper-P Configuration

Table 1: Configuration parameters of ColumnKeeper-P.

P_Y	10^{-3}	10^{-6}	10^{-9}	10^{-12}
P_{REFW}	2.03×10^{-12}	2.03×10^{-15}	2.03×10^{-18}	2.03×10^{-21}
P_1	4.39×10^{-15}	4.38×10^{-18}	4.38×10^{-21}	4.38×10^{-24}
N_{CD}	Preventive Refresh Probability (P_{PR})			
1M	1.23×10^{-3}	1.26×10^{-3}	1.29×10^{-3}	1.32×10^{-3}
128K	1.00×10^{-2}	1.02×10^{-2}	1.05×10^{-2}	1.07×10^{-2}
16K	8.92×10^{-2}	9.13×10^{-2}	9.32×10^{-2}	9.50×10^{-2}

Small increases in P_{PR} significantly increase security for the same N_{CD}

ColumnKeeper-P: Monte-Carlo Simulation



Lower P_Y leads to more preventive refreshes and a lower HC_{max}

Evaluation: Integration w/ RH Mitigations

ColumnKeeper issues ACT+PRE commands:

- ACTs and PREs are transparent to RowHammer mitigation mechanisms

RowHammer mitigation mechanisms issue:

- ACT + PRE commands
- RFM commands

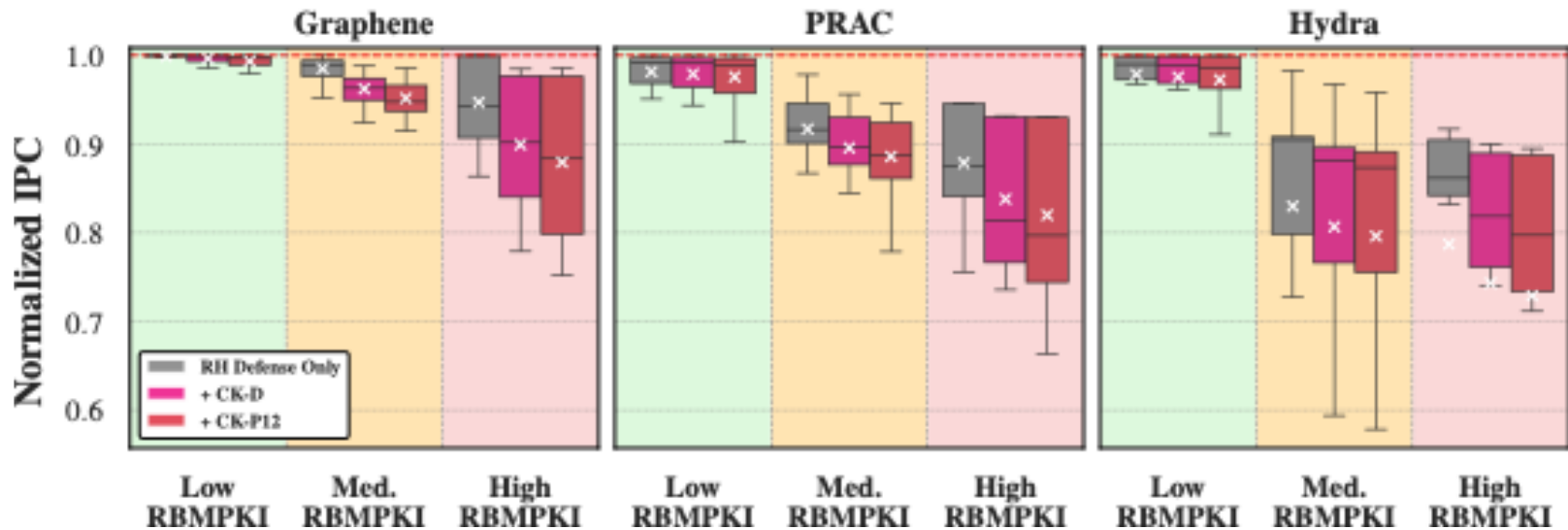
**RFM commands accounted by
incrementing all activation counters in affected banks**

**ColumnKeeper operates transparently to
RowHammer mitigation mechanisms**

**RowHammer mitigation mechanisms operate
transparently to ColumnKeeper**

Evaluation: Integration w/ RH Mitigations

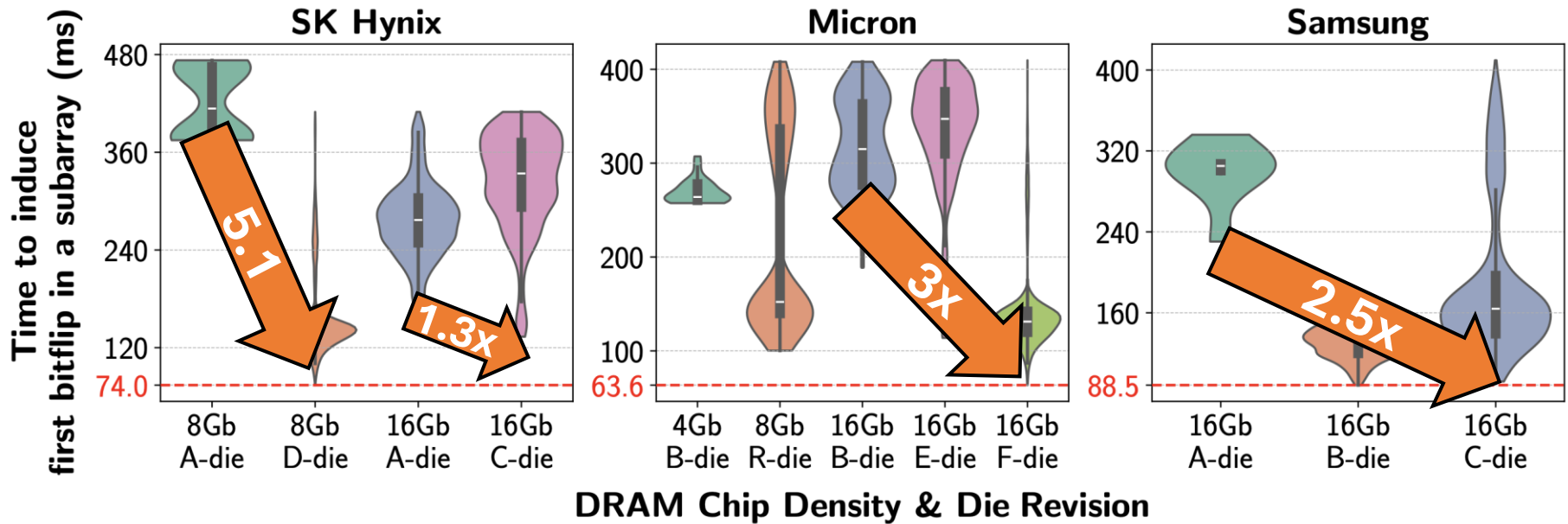
$N_{CD} = 128K, N_{RH} = 128$



No destructive interference between RowHammer mitigation mechanisms and ColumnKeeper

ColumnDisturb:

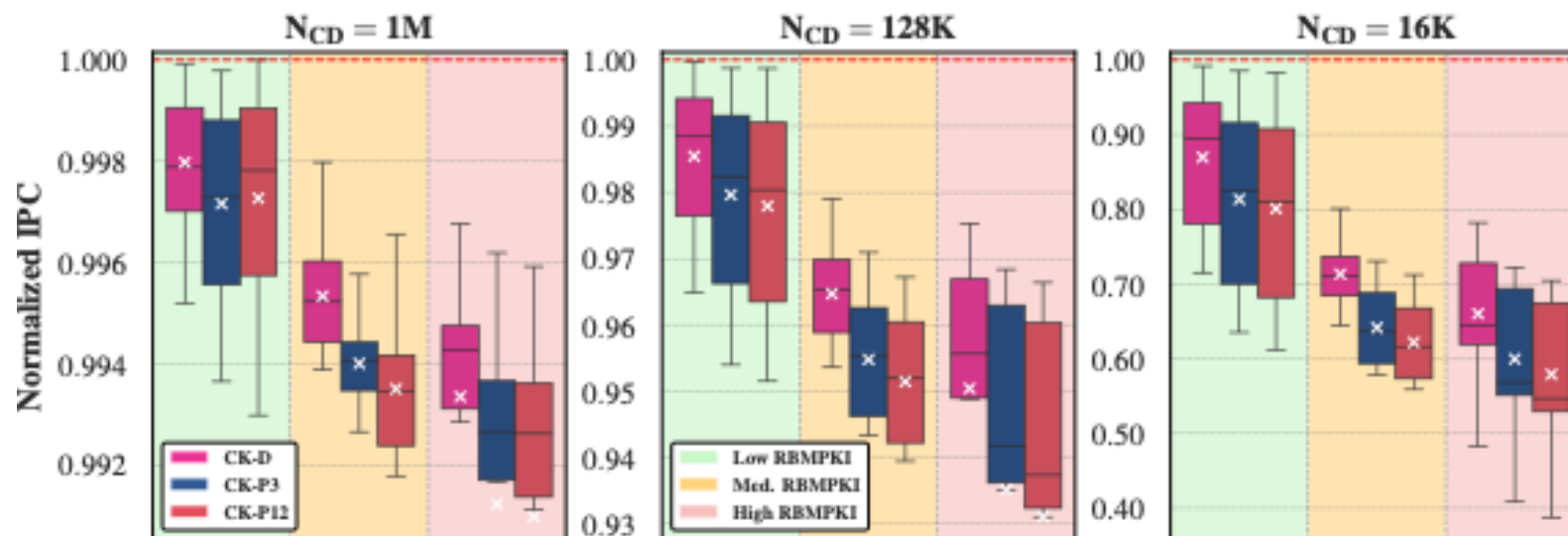
Newer Chips are More Vulnerable



Newer chips tend to be more vulnerable to **ColumnDisturb** bitflips

This trend is **consistent across all tested manufacturers**

Evaluation: Adversarial Workloads



Low performance overheads for **current** and **near-future** thresholds

Significant performance overheads for **future low** thresholds