

CROW

A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability

Presented at ISCA 2019

Hasan Hassan

*Minesh Patel Jeremie S. Kim A. Giray Yaglikci Nandita Vijaykumar
Nika Mansouri Ghiasi Saugata Ghose Onur Mutlu*

ETH zürich

**Carnegie
Mellon
University**

Summary

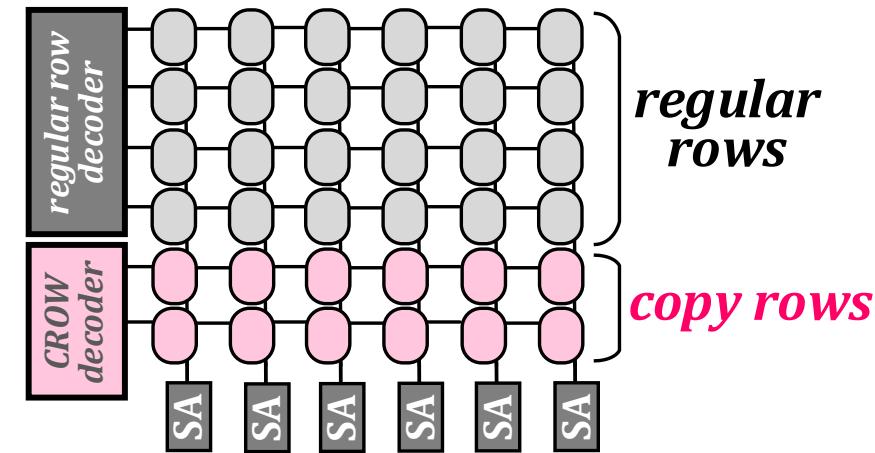
Source code available in July:
github.com/CMU-SAFARI/CROW

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces **copy rows** into a subarray
- The benefits of a **copy row**:
 - Efficiently duplicating data from regular row to a **copy row**
 - Quick access to a duplicated row
 - Remapping a regular row to a **copy row**



CROW is a flexible substrate with many use cases:

- **CROW-cache & CROW-ref** (**20% speedup** and consumes **22% less DRAM energy**)
- Mitigating RowHammer
- We hope CROW enables many other use cases going forward

Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

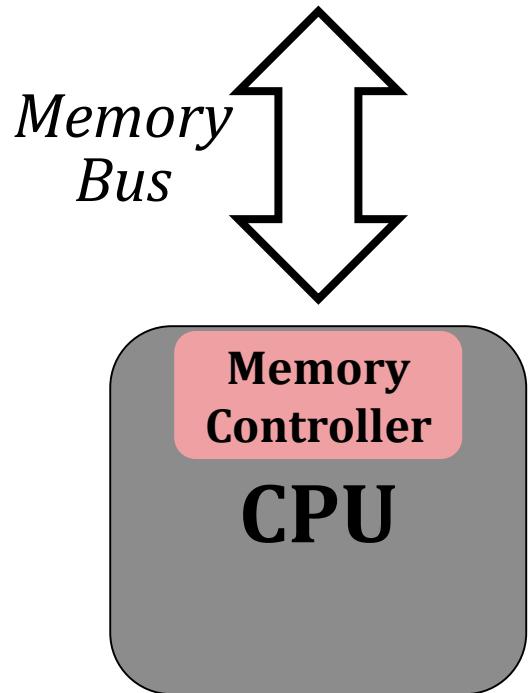
CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

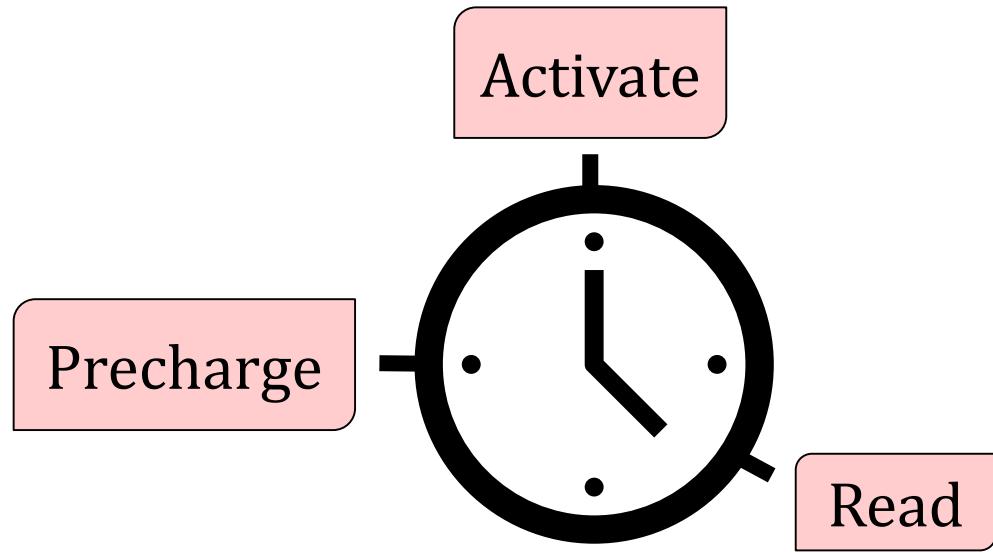
3. Evaluation

4. Conclusion

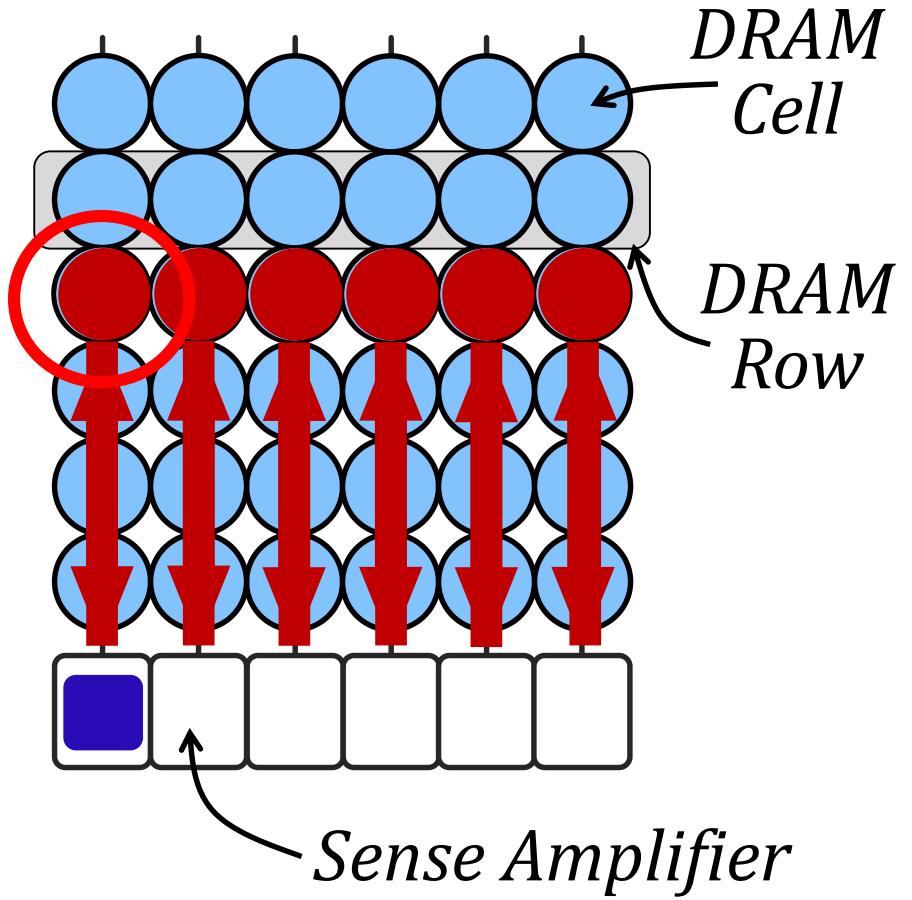
DRAM Organization



Accessing DRAM



DRAM Subarray



Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

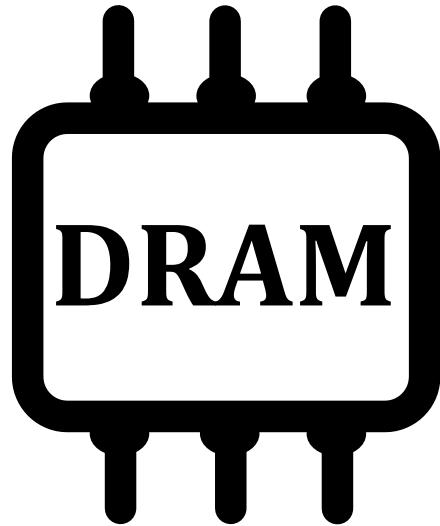
CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

Challenges of DRAM Scaling

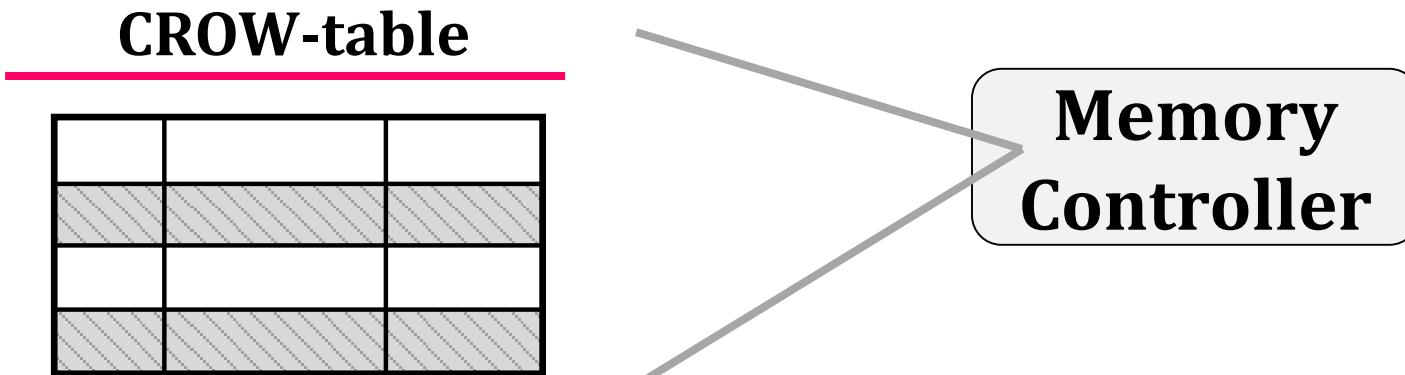
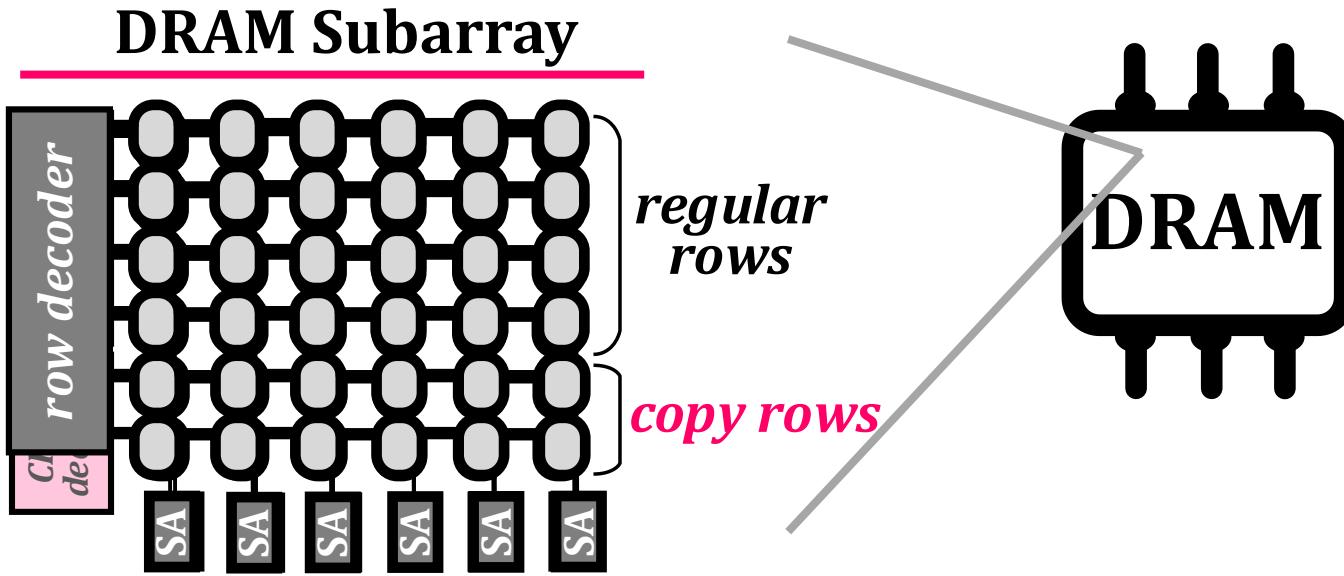


- 1 access latency
- 2 refresh overhead
- 3 exposure to vulnerabilities

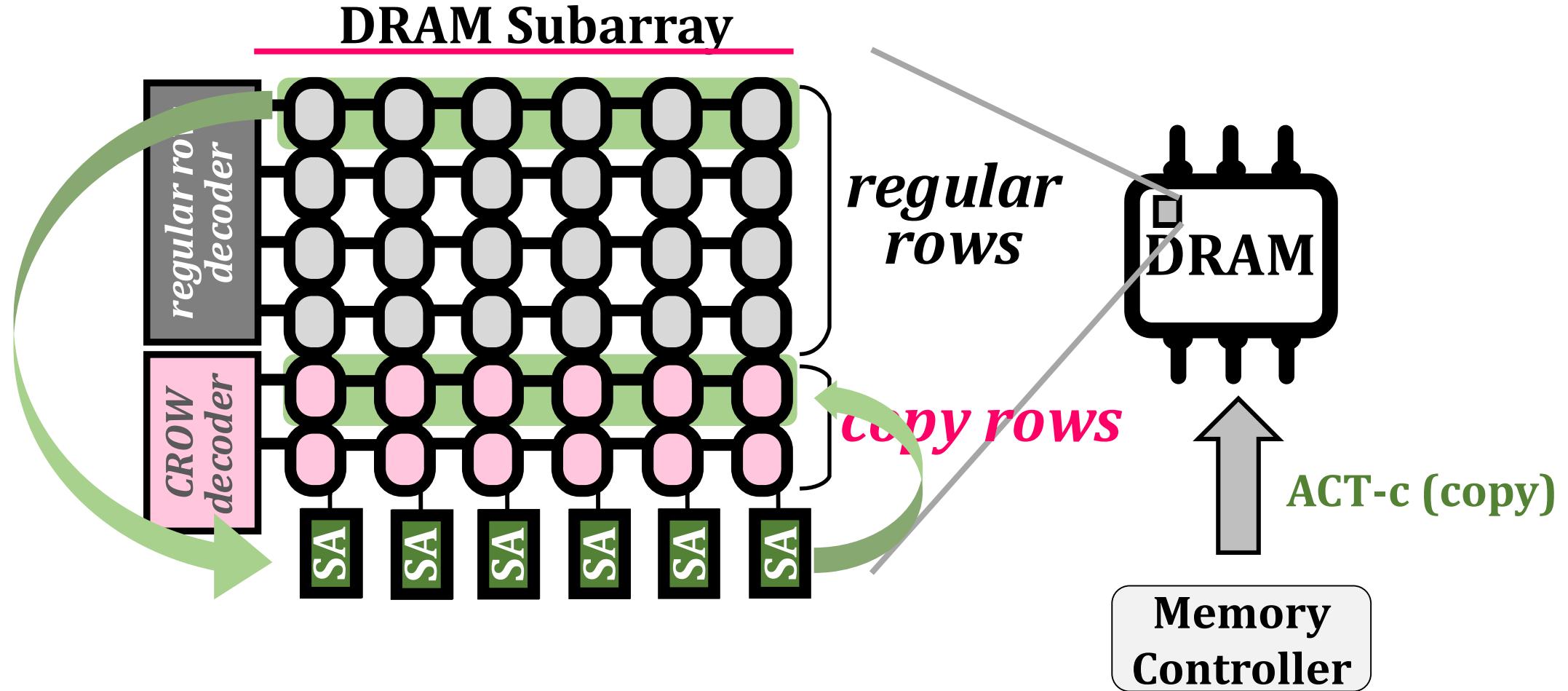
Our Goal

We want a **substrate** that enables
the **duplication** and **remapping** of data within a subarray

The Components of CROW

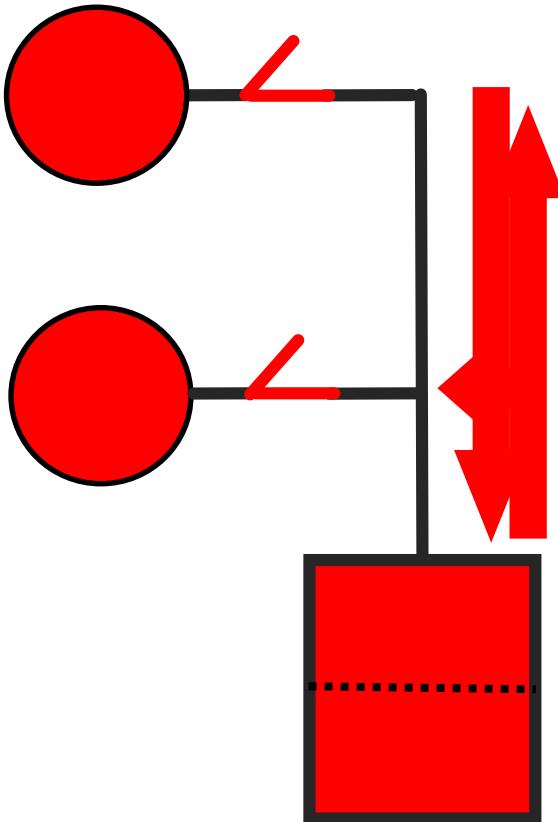


CROW Operation 1: Row Copy



Row Copy: Steps

source row:



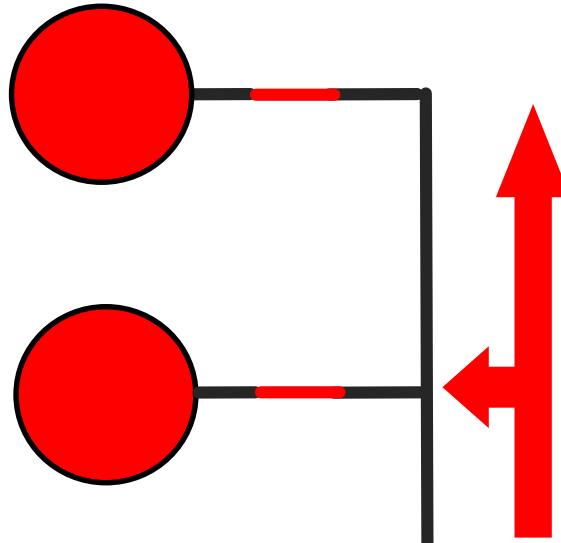
destination row:

- 1 Activation of the source row
- 2 Charge sharing
- 3 Beginning of restoration
- 4 Activation of the destination row
- 5 Restoration of both rows to source data

*Sense
Amplifier*

Row Copy: Steps

source row:



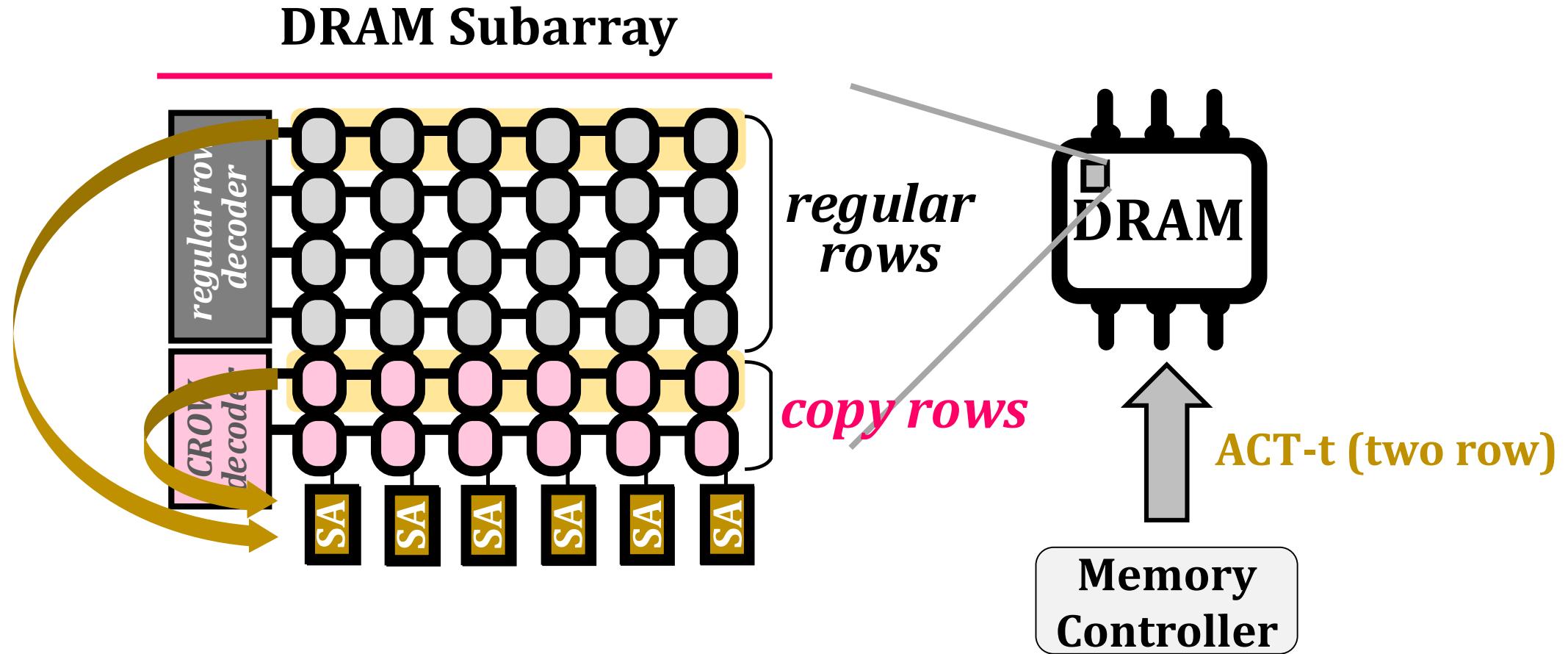
destination row:

- ① Activation of the source row
- ② Charge sharing
- ③ Beginning of restoration

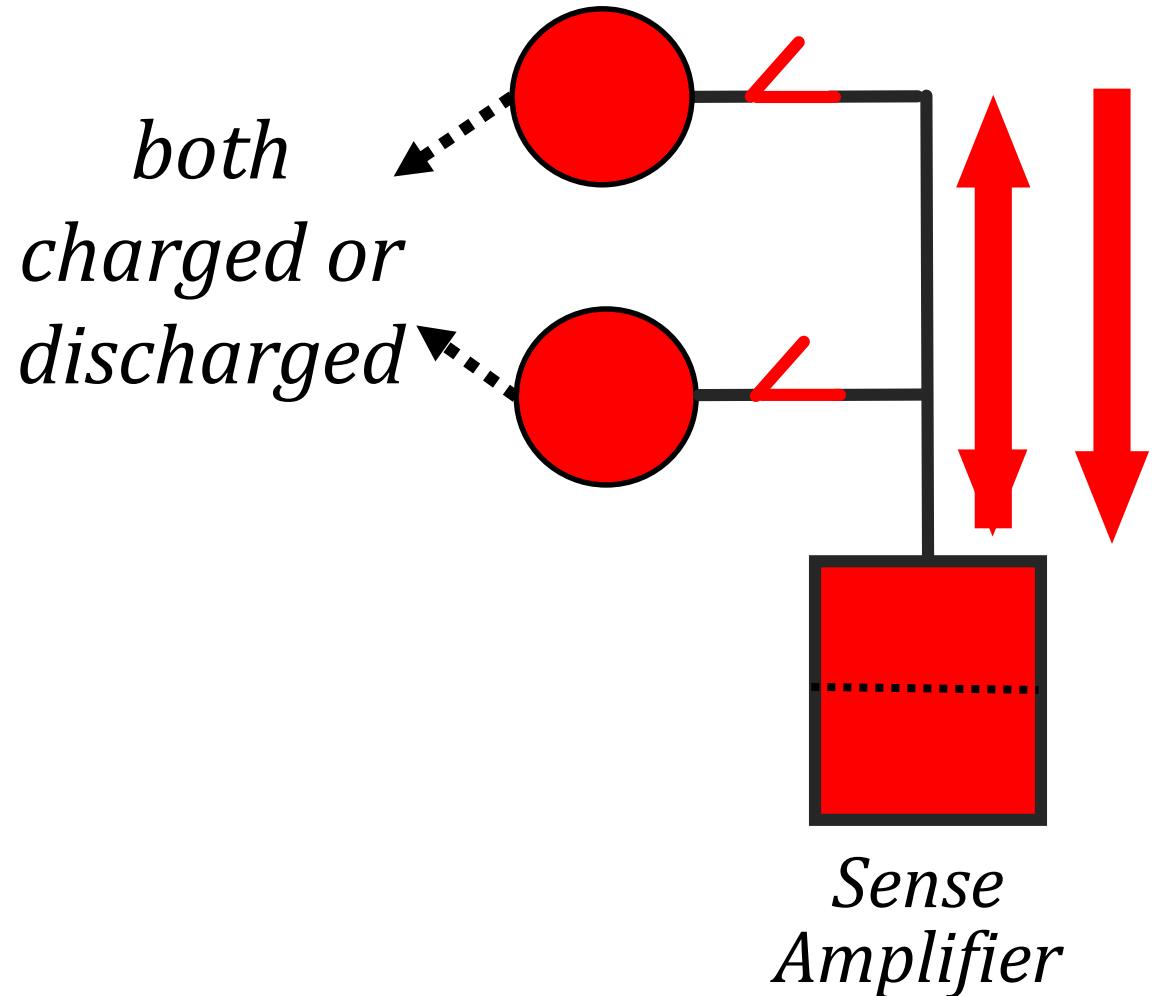
Enables quickly copying a regular row
into a **copy row**

Amplifier

CROW Operation 2: Two-Row Activation

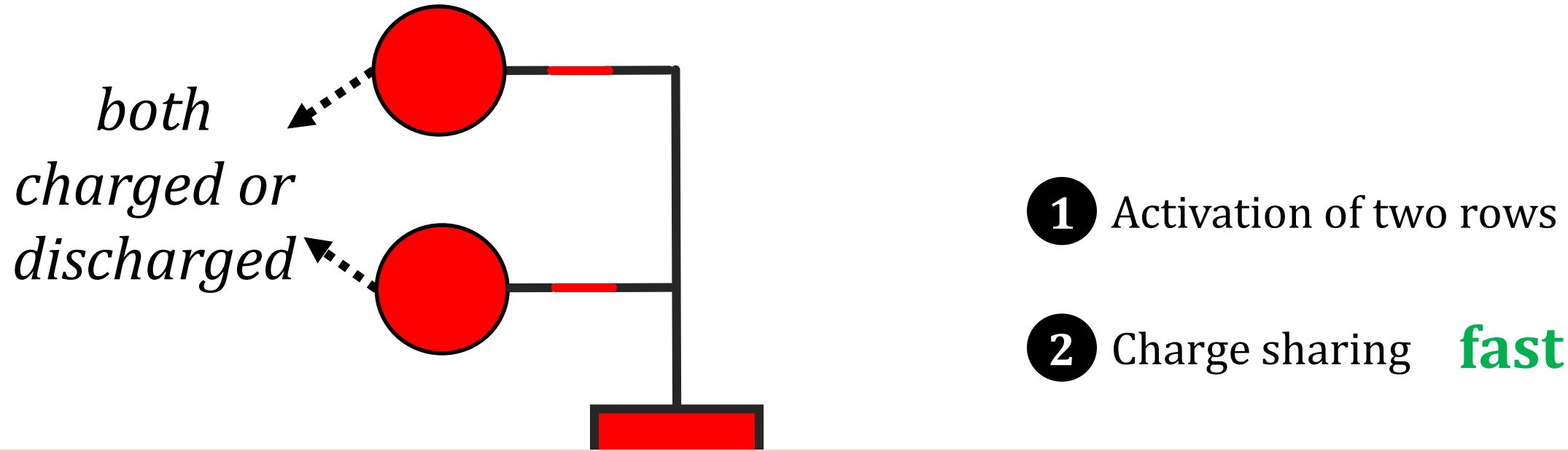


Two-Row Activation: Steps



- 1 Activation of two rows
- 2 Charge sharing **fast**
- 3 Restoration

Two-Row Activation: Steps



Enables fast access to data that is duplicated across a regular row and a **copy row**

Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

CROW-cache

Problem: High access latency

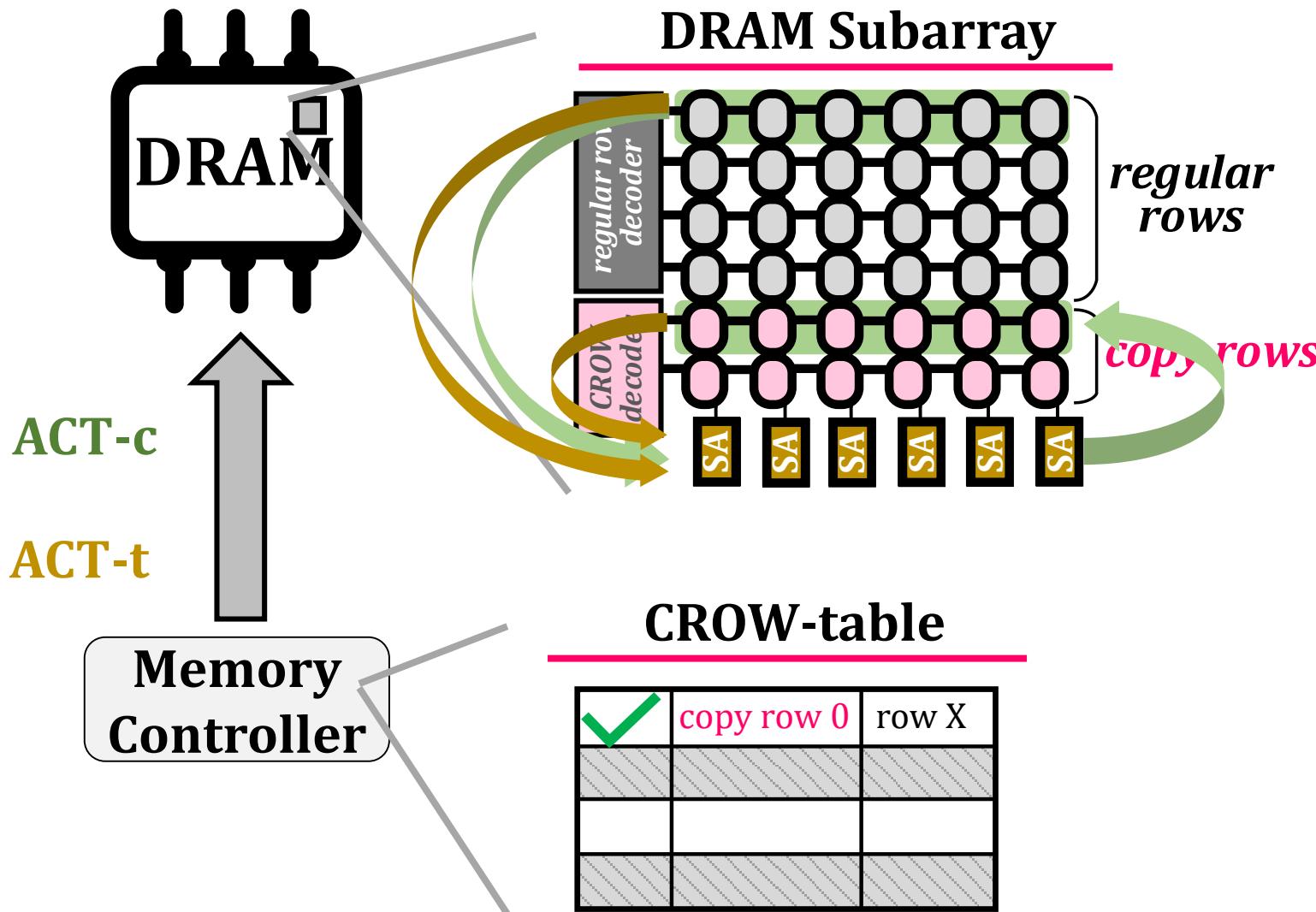
Key idea: Use **copy rows** to enable low-latency access to most-recently-activated regular rows in a subarray

CROW-cache combines:

- row copy → copy a newly activated regular row into a **copy row**
- **two-row activation** → activate the regular row and **copy row** together on the next access

Reduces activation latency by **38%**

CROW-cache Operation



Request Queue

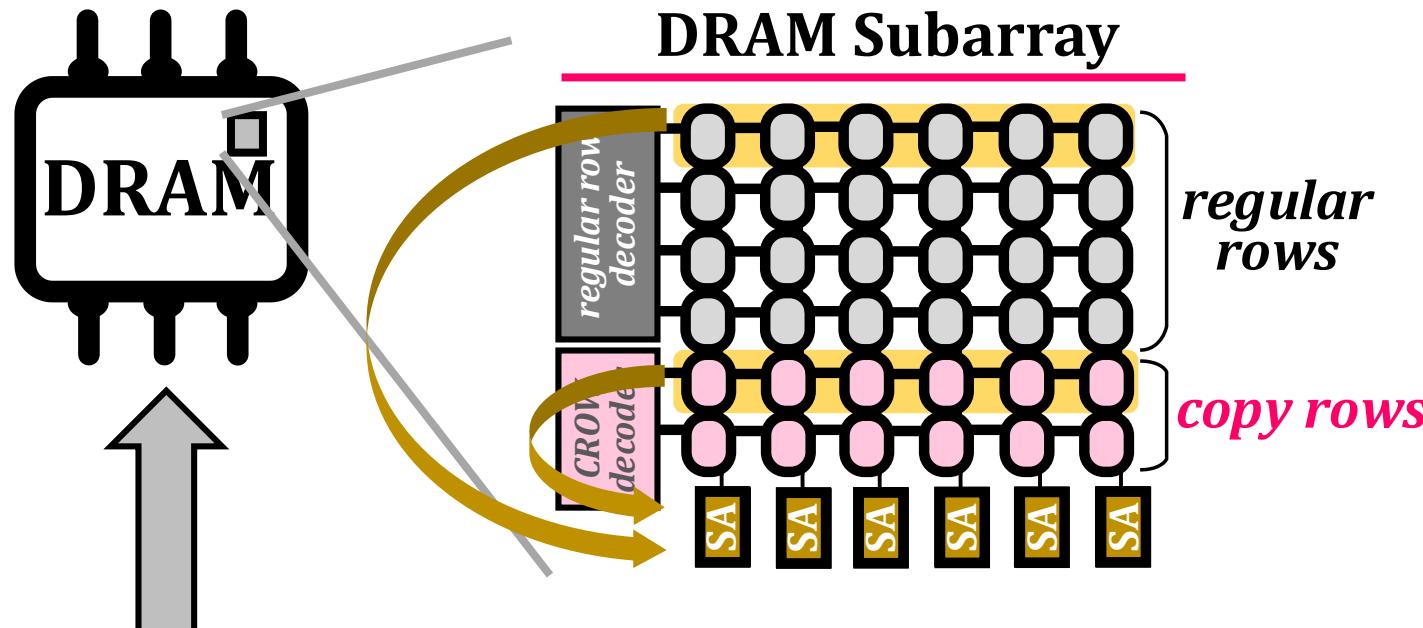
load row X

[bank conflict]

load row X

- 1 CROW-table miss
 - 2 Allocate a copy row
 - 3 Issue ACT-c (copy)
-
- 1 CROW-table hit
 - 2 Issue ACT-t (two row)

CROW-cache Operation



Request Queue

load row X

[bank conflict]

load row X

- 1 CROW-table miss
- 2 Allocate a copy row

Second activation of row X is faster



Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

CROW-ref

Problem: Refresh has high overheads. Weak rows lead to high refresh rate

- **weak row:** at least one of the row's cells cannot retain data correctly when refresh rate is decreased

Key idea: Safely reduce refresh rate by remapping a **weak** regular row to a **strong copy row**

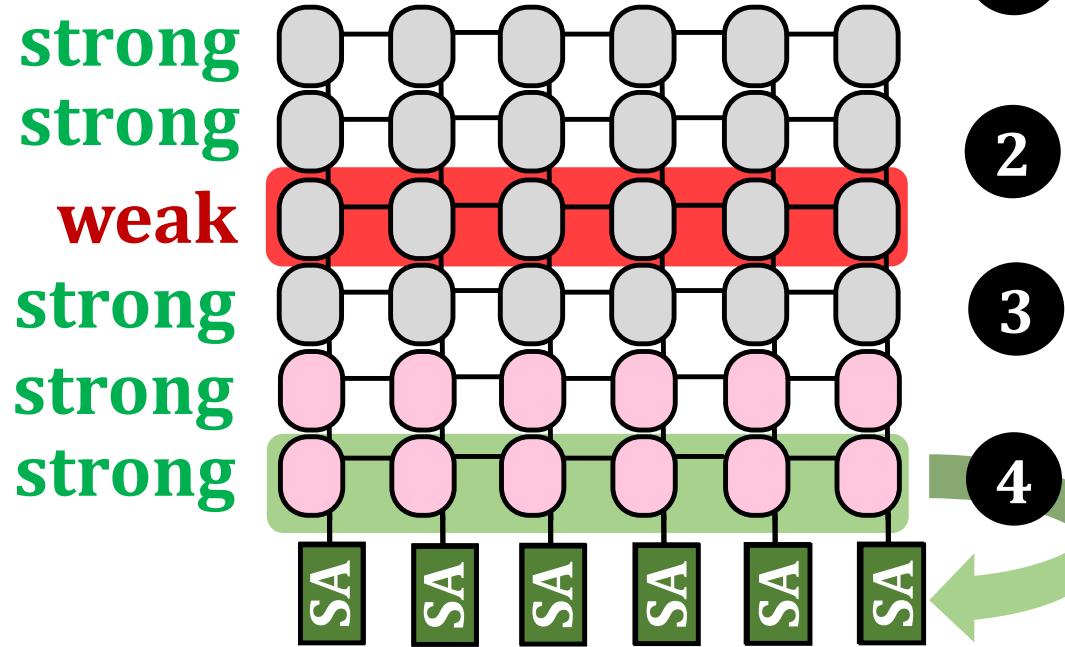
CROW-ref uses:

- **row copy** → copy a weak regular row to a strong **copy row**

CROW-ref eliminates more than half of the refresh requests

CROW-ref Operation

Retention Time Profiler



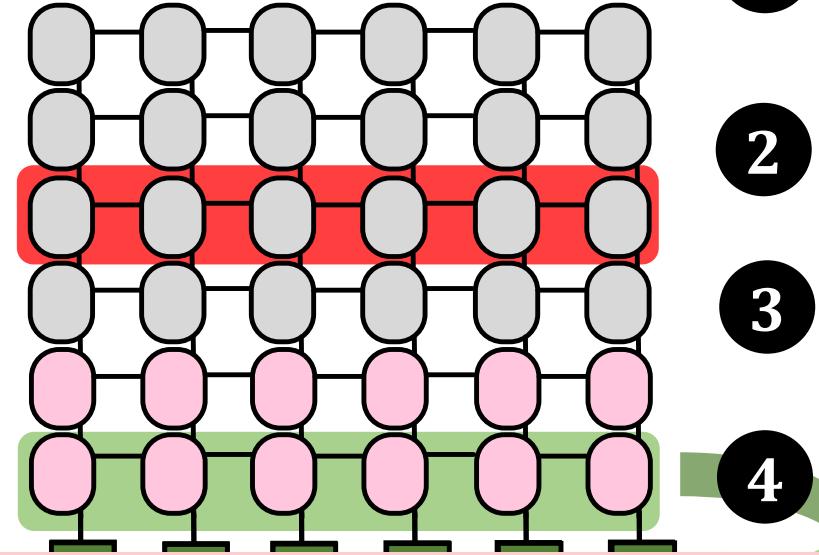
- 1 Perform retention time profiling
- 2 Remap weak rows to strong copy rows
- 3 On ACT, check the CROW-table
- 4 If remapped, activate a copy row

CROW-ref Operation



Retention Time

strong
strong
weak
strong
strong
strong



- 1 Perform retention time profiling
- 2 Remap weak rows to strong copy rows
- 3 On ACT, check the CROW-table
- 4 If remapped, activate a copy row

How many weak rows exist
in a DRAM chip?

Identifying Weak Rows

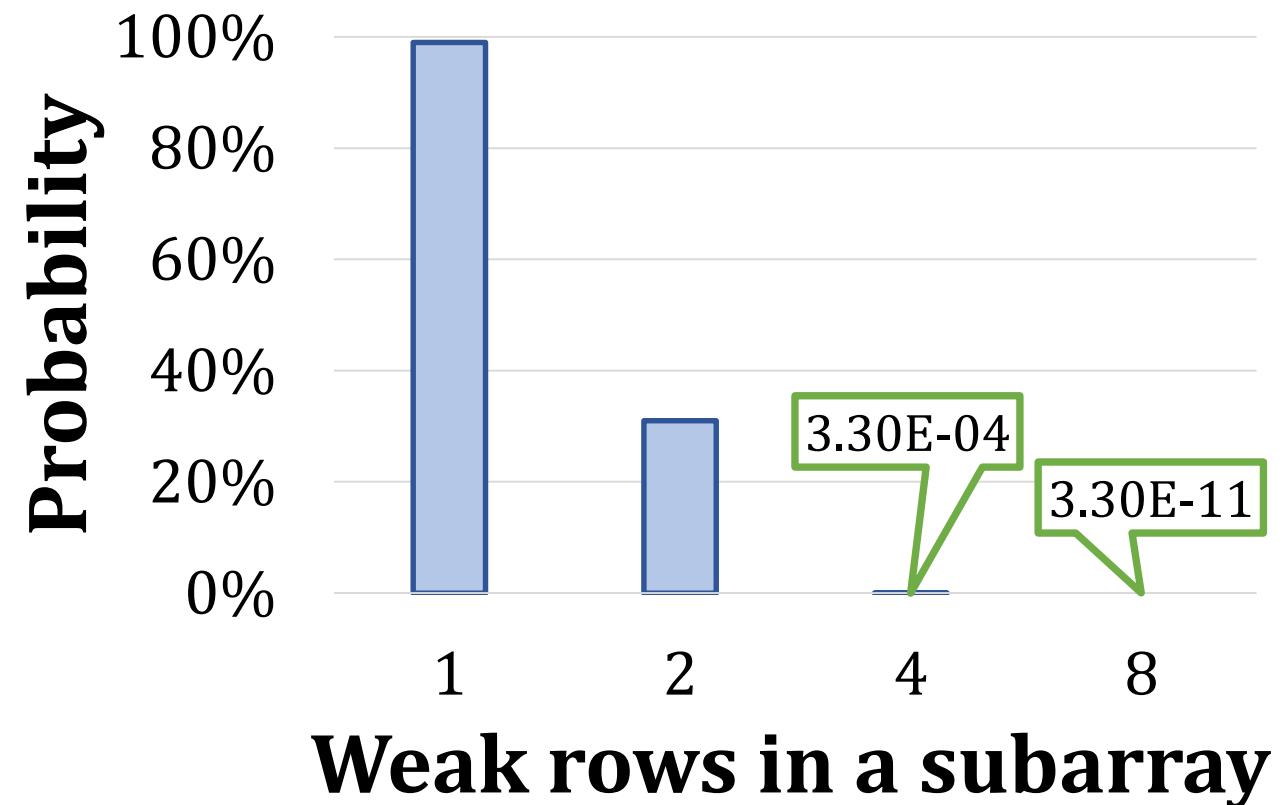
Weak cells are rare [*Liu+, ISCA'13*]

weak cell: retention < 256ms

~ $1000/2^{38}$ (32 GiB) failing cells

DRAM Retention Time Profiler

- REAPER [*Patel+, ISCA'17*]
- PARBOR [*Khan+, DSN'16*]
- AVATAR [*Qureshi+, DSN'15*]
- At system *boot* or during *runtime*



Identifying Weak Rows

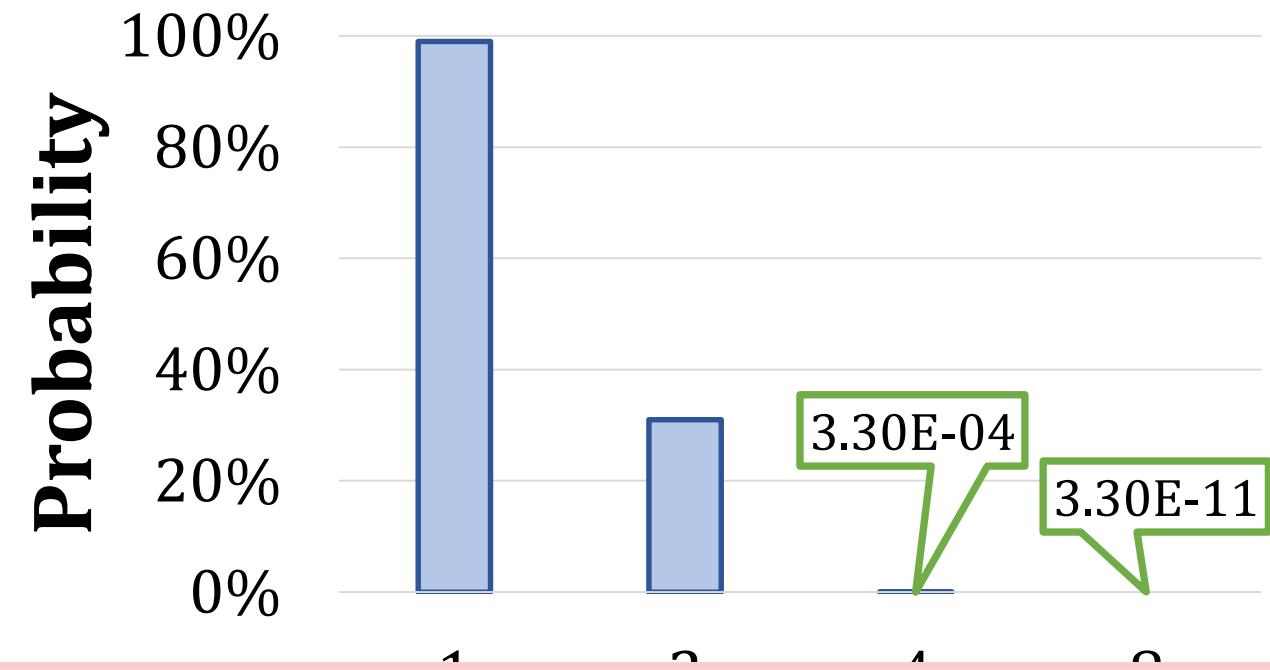
Weak cells are rare [*Liu+, ISCA'13*]

weak cell: retention < 256ms

~ $1000/2^{38}$ (32 GiB) failing cells

DRAM Retention Time Profiler

- REAPER [*Patel+, ISCA'17*]



A few **copy rows** are sufficient
to halve the refresh rate

Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

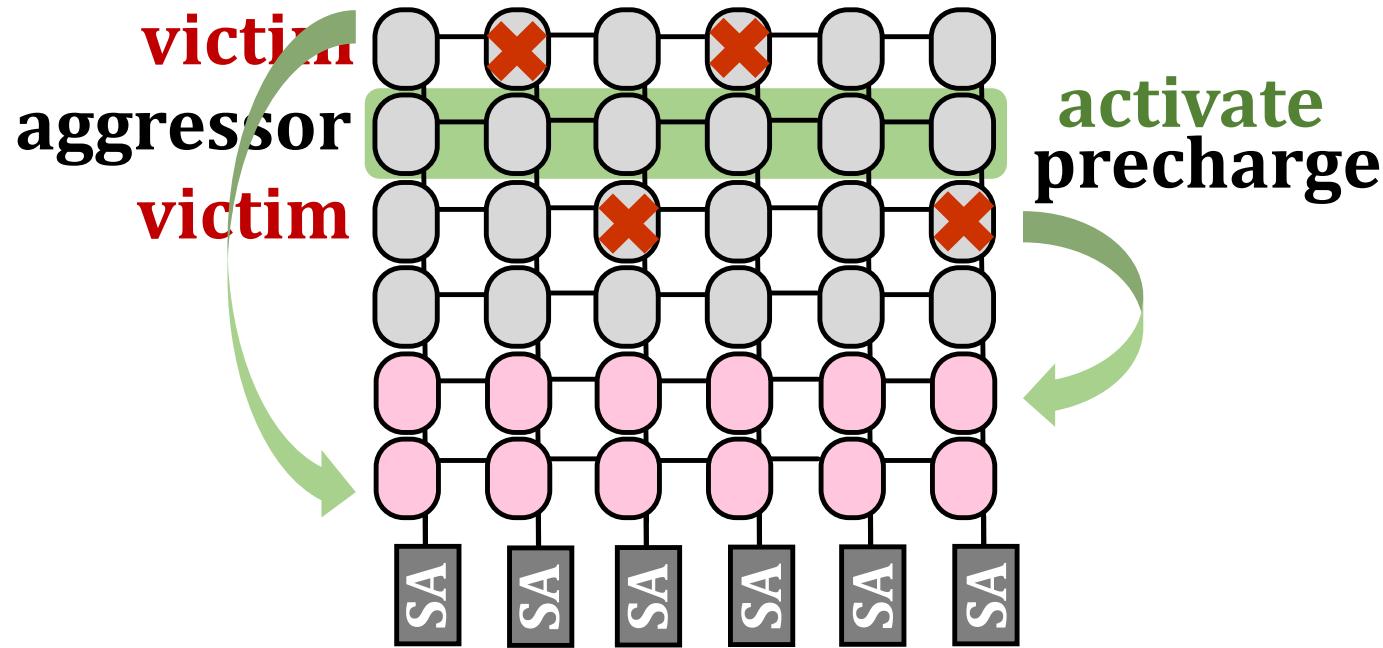
CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

Mitigating RowHammer



Key idea: remap victim rows to copy rows

Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

Methodology

- **Simulator**

- DRAM Simulator (Ramulator [*Kim+, CAL'15*])
<https://github.com/CMU-SAFARI/ramulator>

Source code available in July:
github.com/CMU-SAFARI/CROW

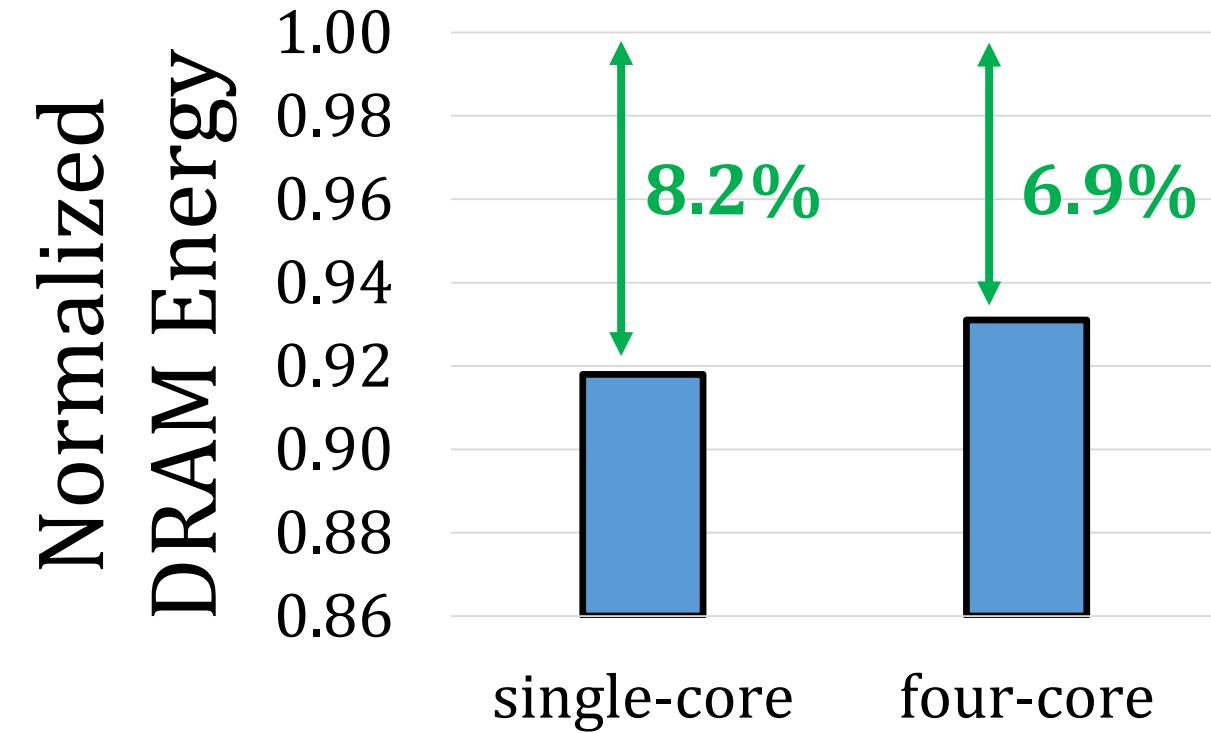
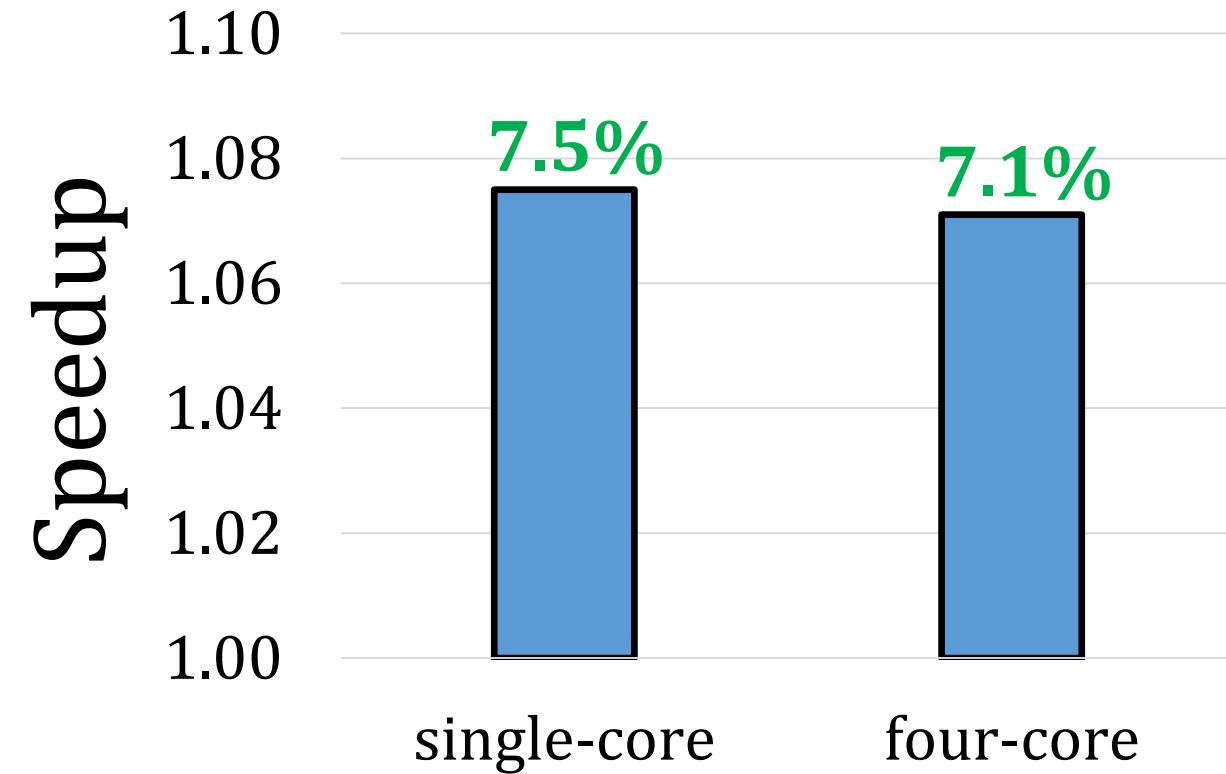
- **Workloads**

- 44 single-core workloads
 - *SPEC CPU2006, TPC, STREAM, MediaBench*
- 160 multi-programmed four-core workloads
 - *By randomly choosing from single-core workloads*
- Execute at least 200 million representative instructions per core

- **System Parameters**

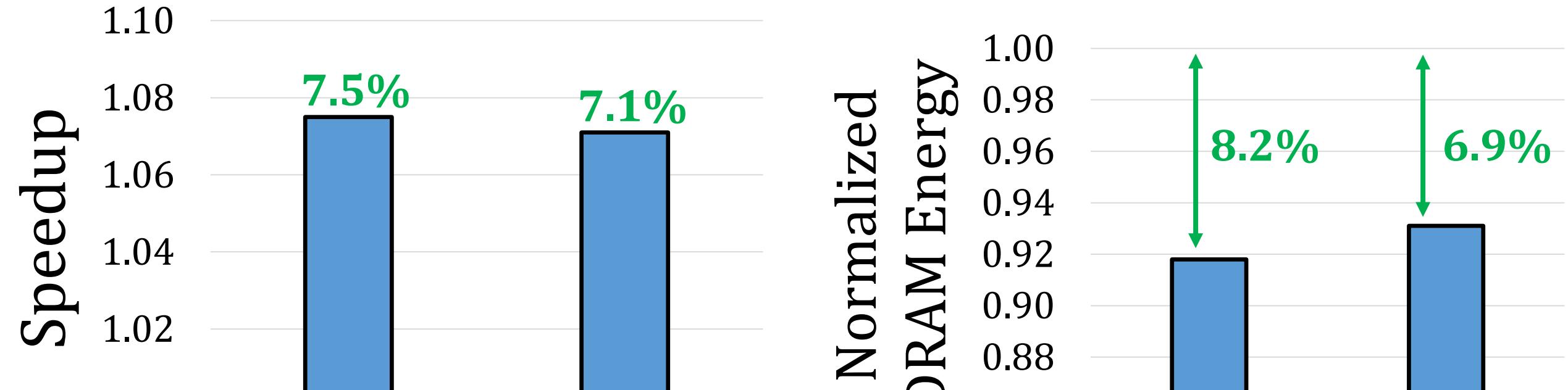
- 1/4 core system with 8 MiB LLC
- LPDDR4 main memory
- 8 **copy rows** per 512-row subarray

CROW-cache Results



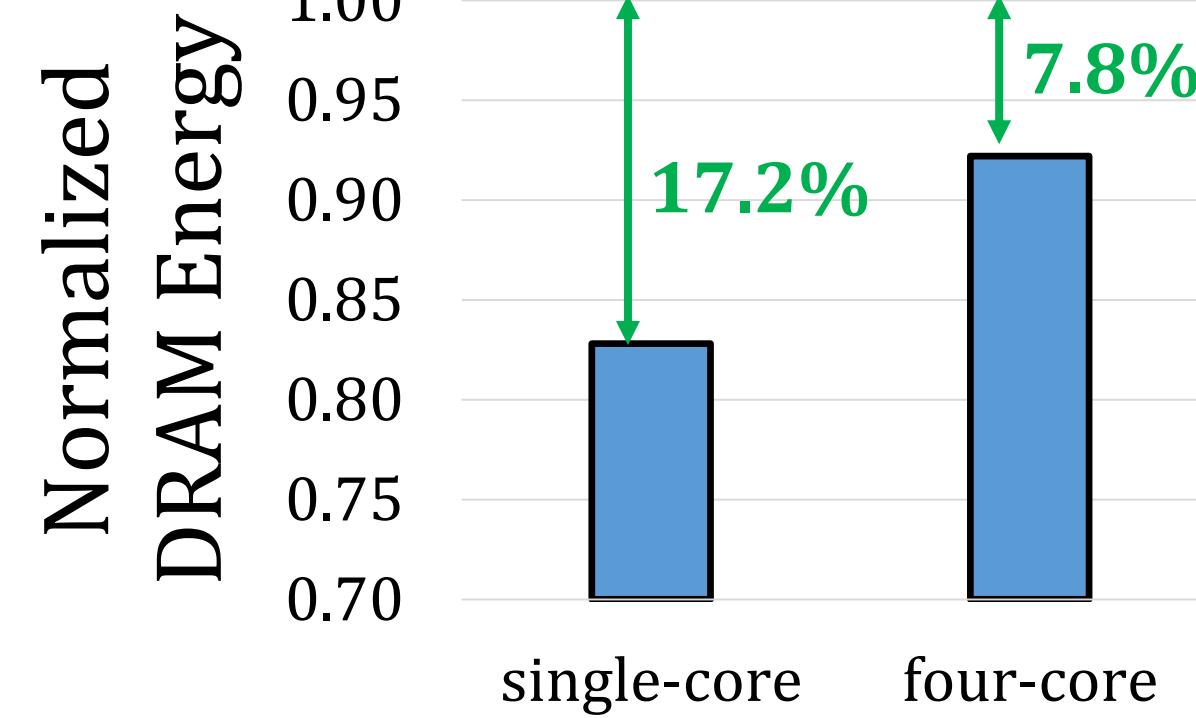
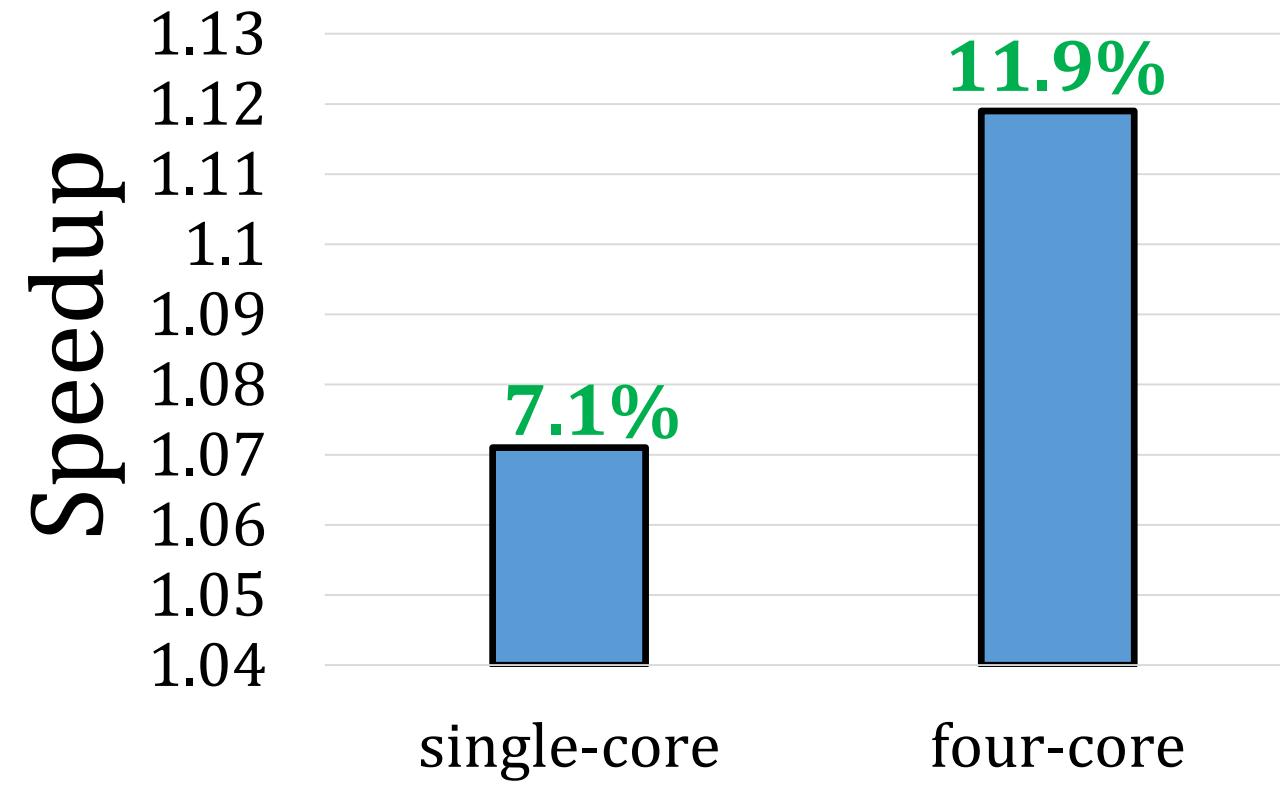
* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)

CROW-cache Results



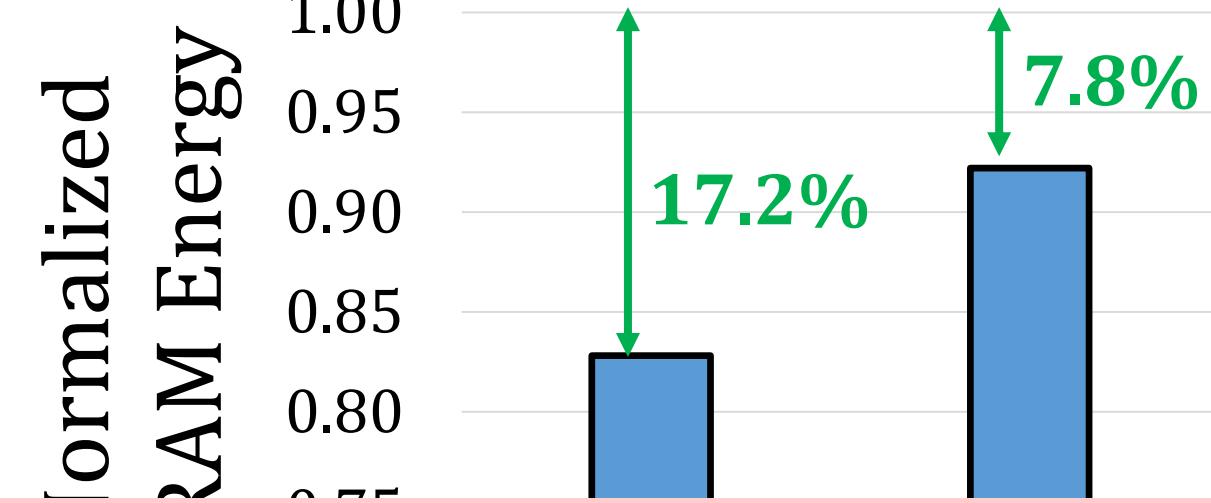
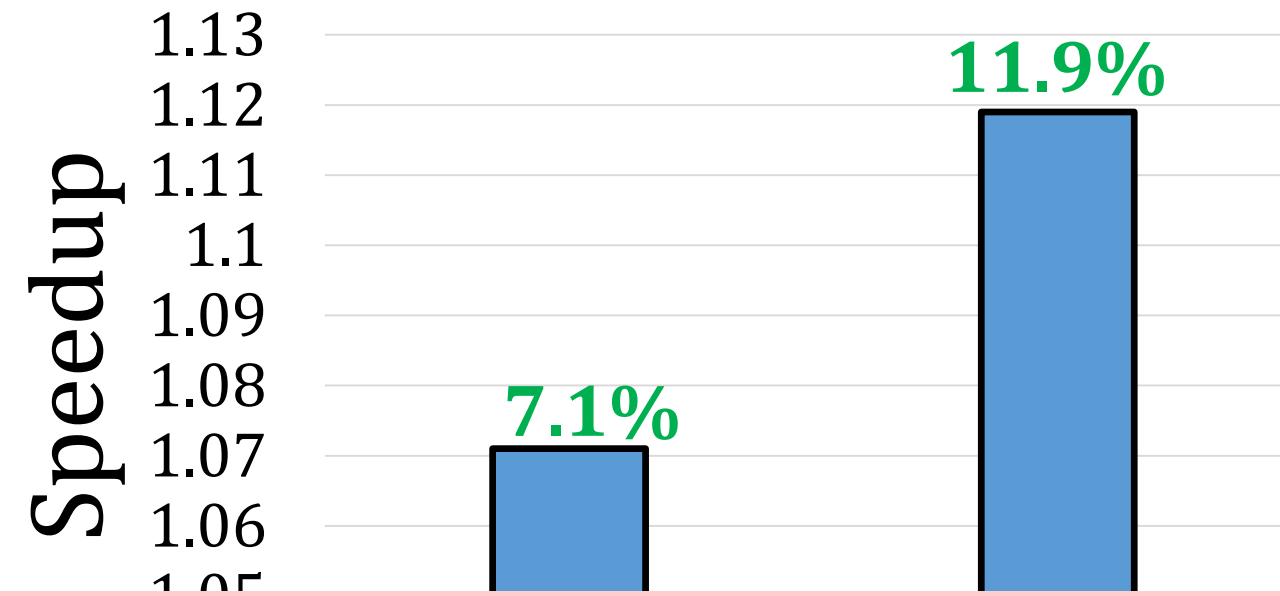
CROW-cache **improves** single-/four-core
performance and energy

CROW-ref Results



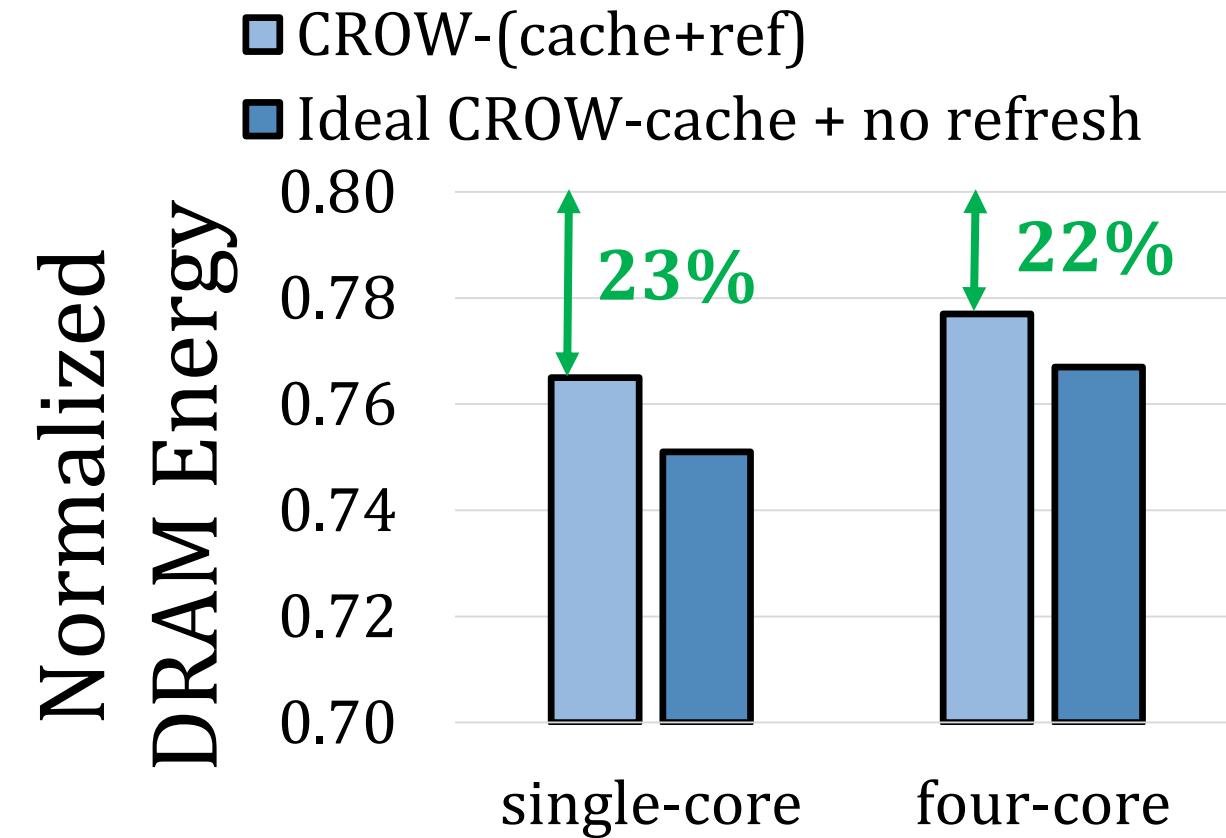
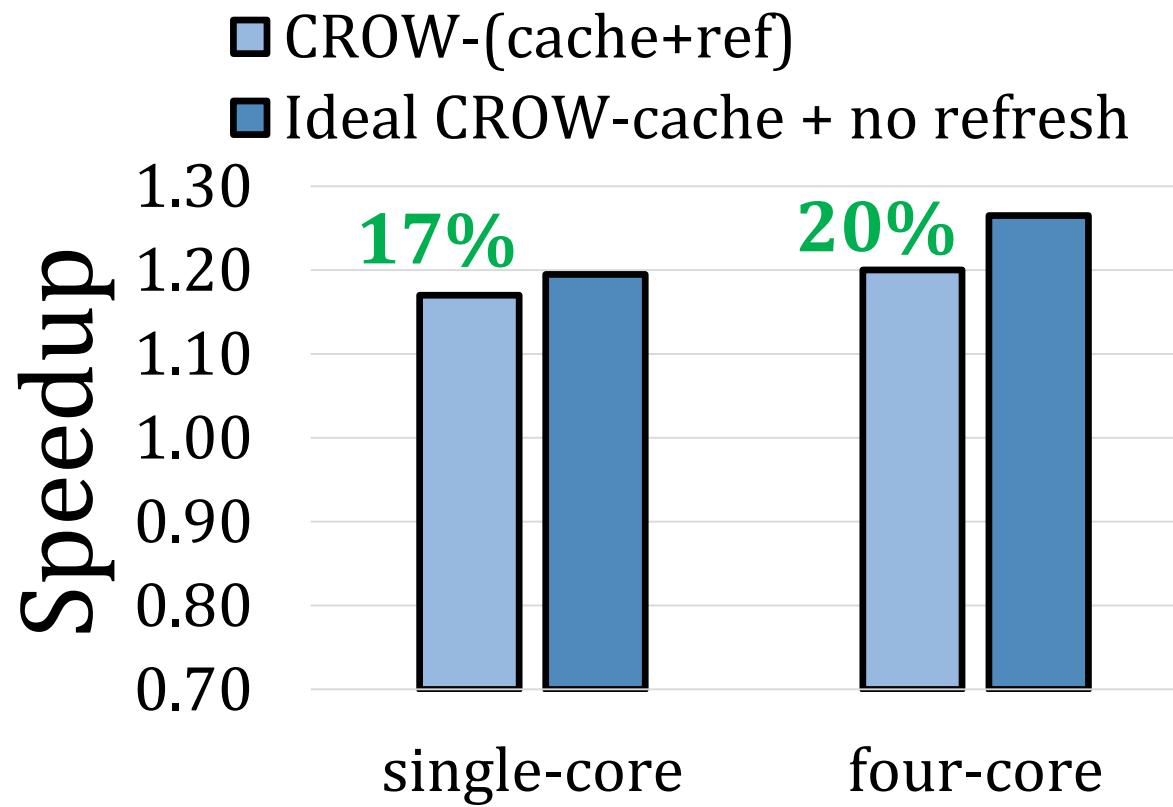
* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)

CROW-ref Results

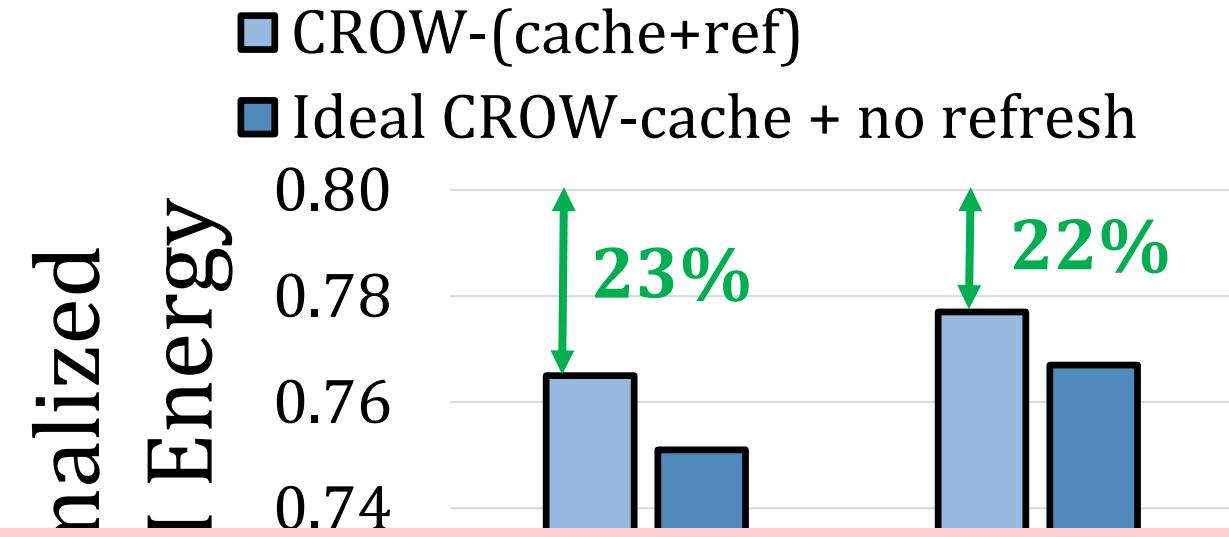
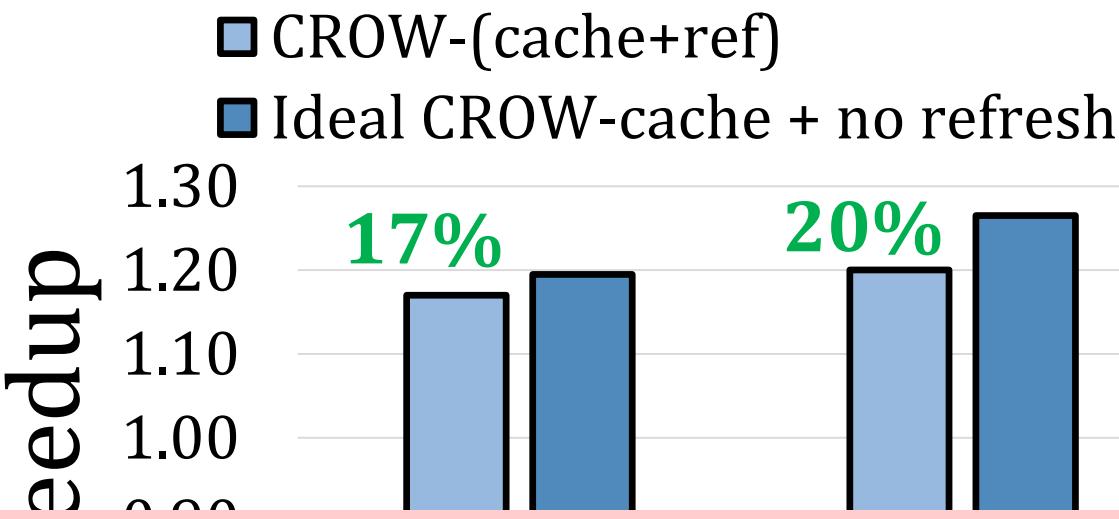


CROW-ref **significantly reduces** the performance
and energy overhead of DRAM refresh

Combining CROW-cache and CROW-ref



Combining CROW-cache and CROW-ref



CROW-(cache+ref) provides more performance and DRAM energy benefits than each mechanism alone

Hardware Overhead

For 8 copy rows and 16 GiB DRAM:

- 0.5% DRAM chip area
- 1.6% DRAM capacity
- 11.3 KiB memory controller storage

CROW is a low-cost substrate

Other Results in the Paper

- Performance and energy sensitivity to:
 - Number of copy-rows per subarray
 - DRAM chip density
 - Last-level cache capacity
- CROW-cache with prefetching
- CROW-cache compared to other in-DRAM caching mechanisms:
 - TL-DRAM [*Lee+, HPCA'13*]
 - SALP [*Kim+, ISCA'12*]

Outline

1. DRAM Operation Basics

2. The CROW Substrate

CROW-cache: Reducing DRAM Latency

CROW-ref: Reducing DRAM Refresh

Mitigating RowHammer

3. Evaluation

4. Conclusion

Conclusion

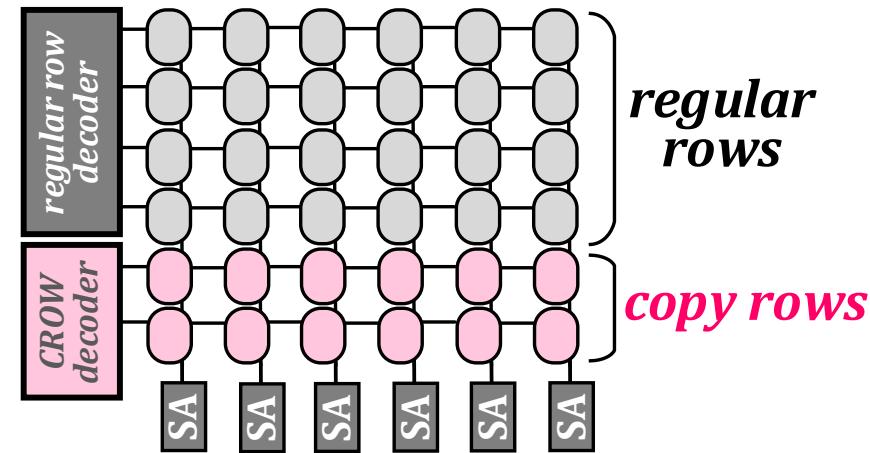
Source code available in July:
github.com/CMU-SAFARI/CROW

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces **copy rows** into a subarray
- The benefits of a **copy row**:
 - Efficiently duplicating data from regular row to a **copy row**
 - Quick access to a duplicated row
 - Remapping a regular row to a **copy row**



CROW is a flexible substrate with many use cases:

- **CROW-cache & CROW-ref** (**20% speedup** and consumes **22% less DRAM energy**)
- Mitigating RowHammer
- We hope CROW enables many other use cases going forward

CROW

A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability

Hasan Hassan

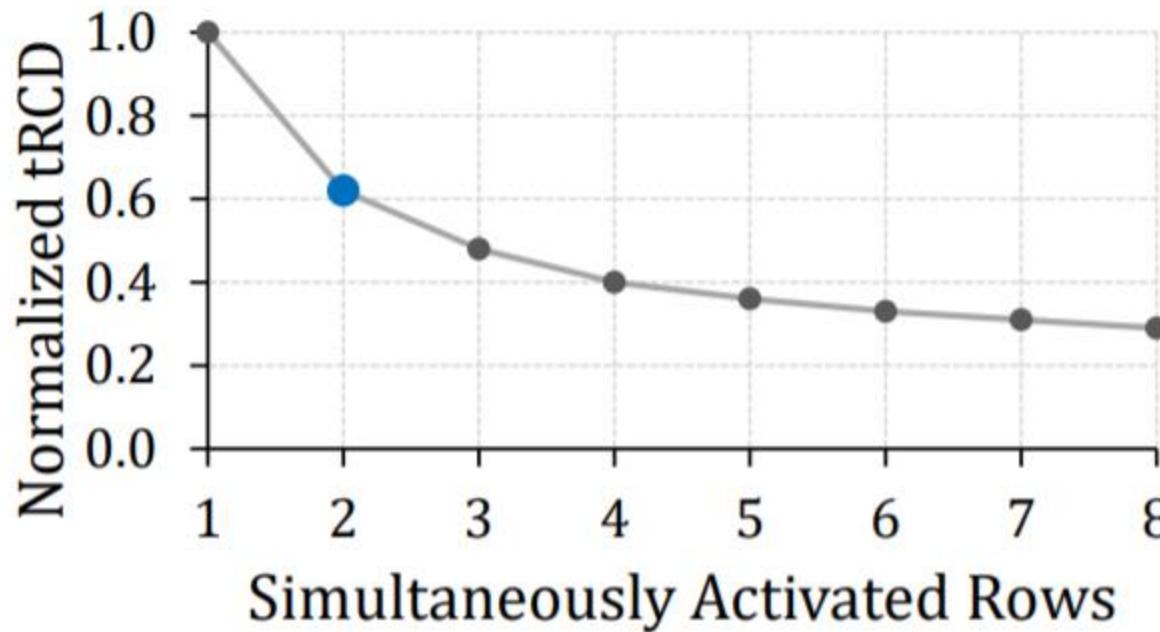
*Minesh Patel Jeremie S. Kim A. Giray Yaglikci Nandita Vijaykumar
Nika Mansouri Ghiasi Saugata Ghose Onur Mutlu*

ETH zürich

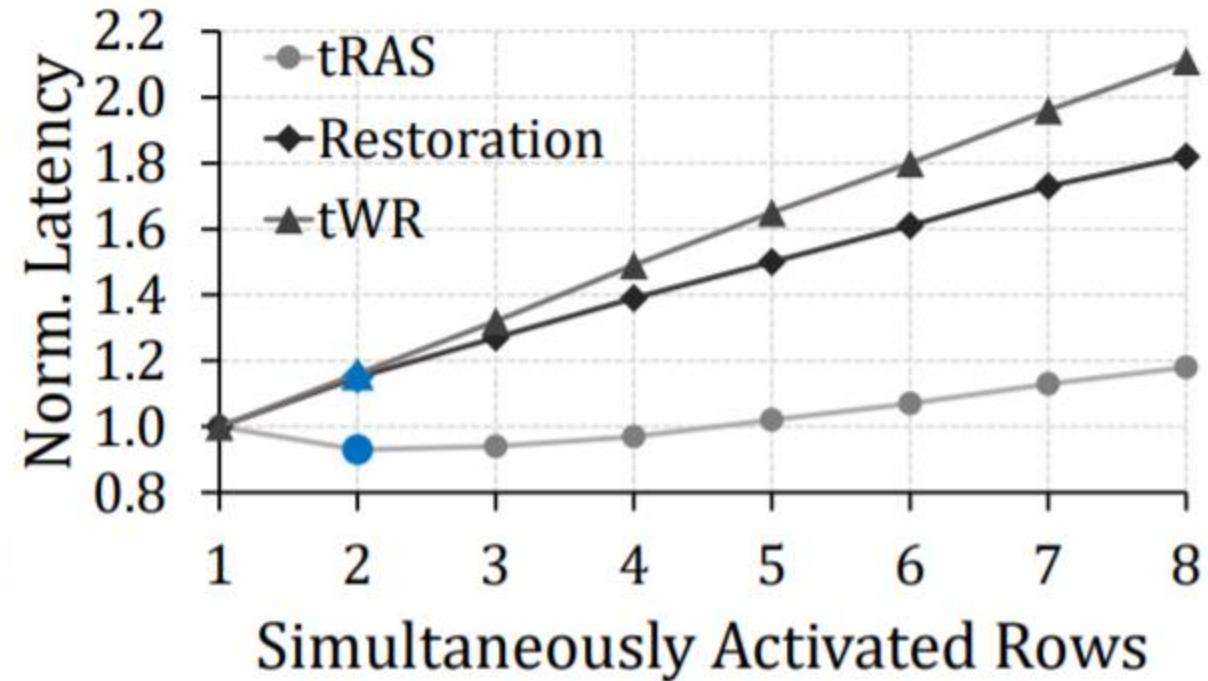
**Carnegie
Mellon
University**

Backup Slides

Latency Reduction with MRA

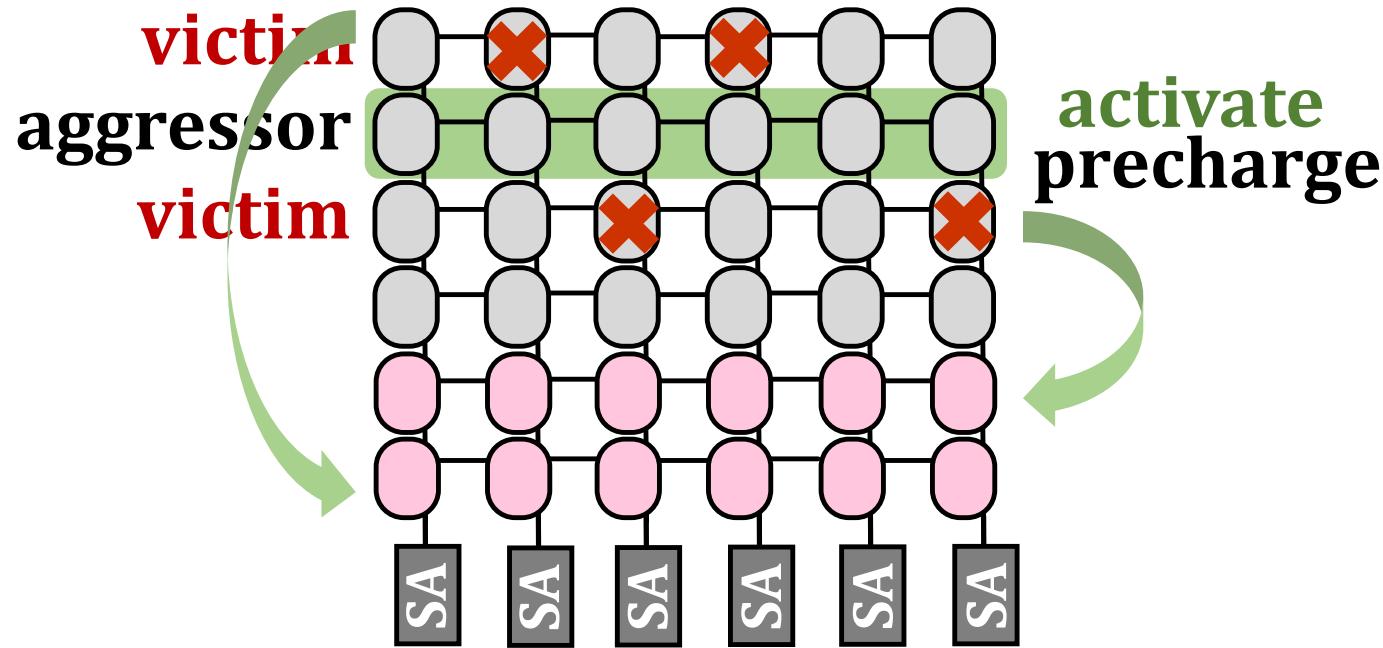


(a) tRCD (18 ns)



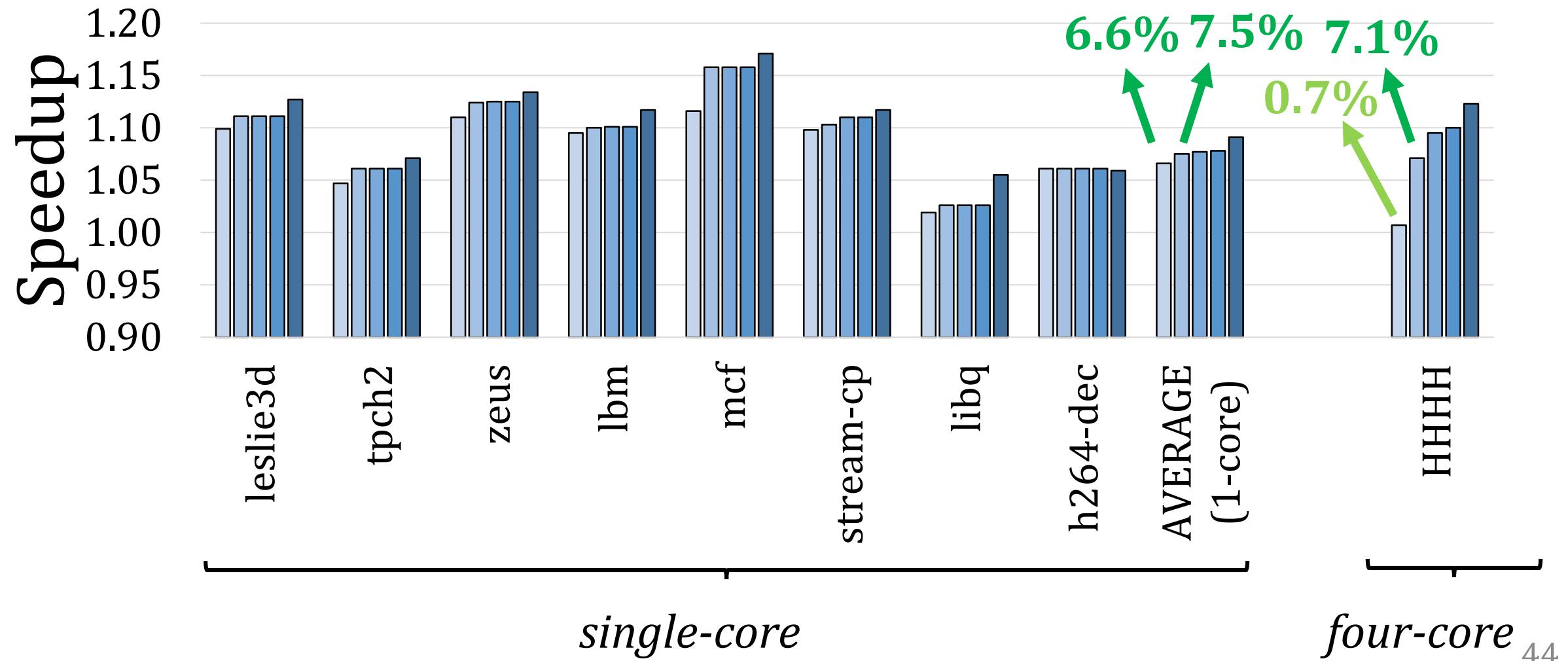
(b) tRAS (42 ns), tWR (18 ns), and restoration (24 ns)

Mitigating RowHammer

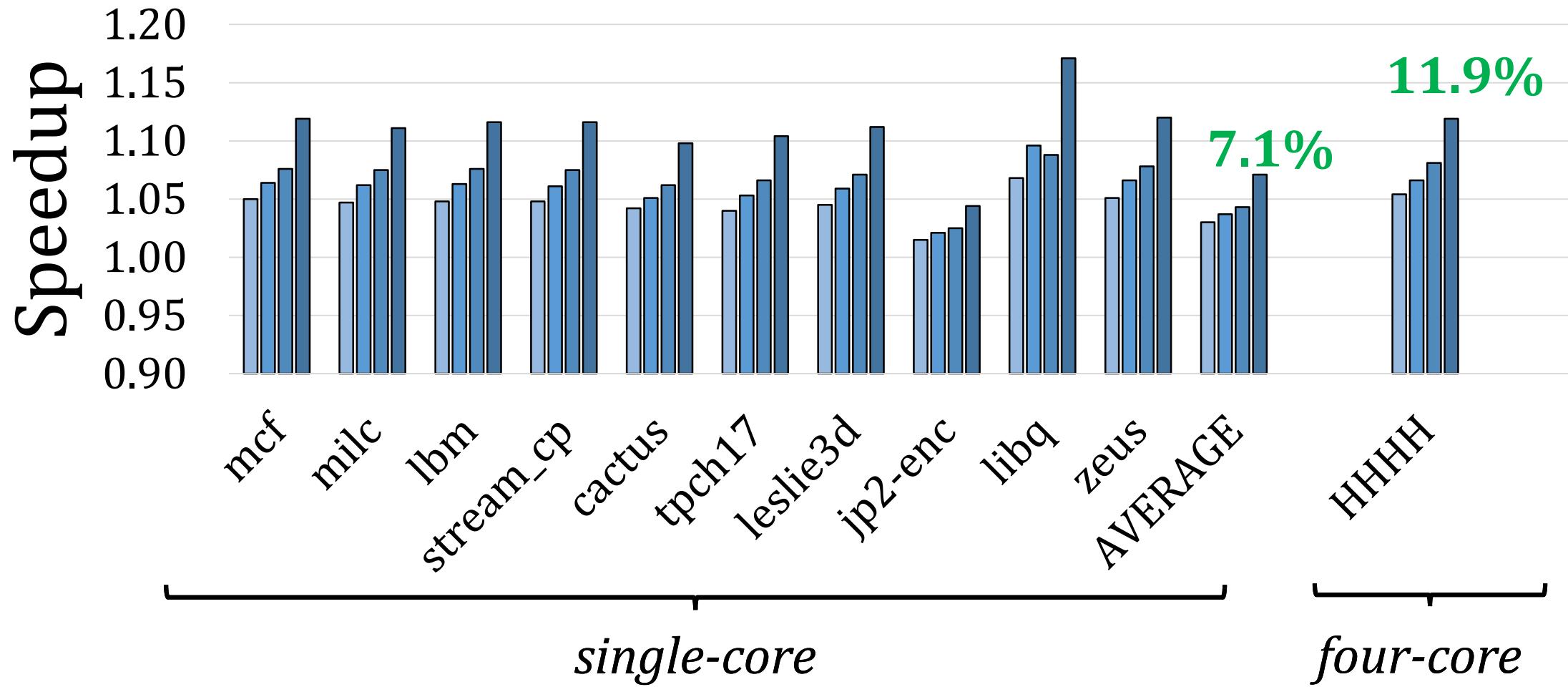


Key idea: remap victim rows to copy rows

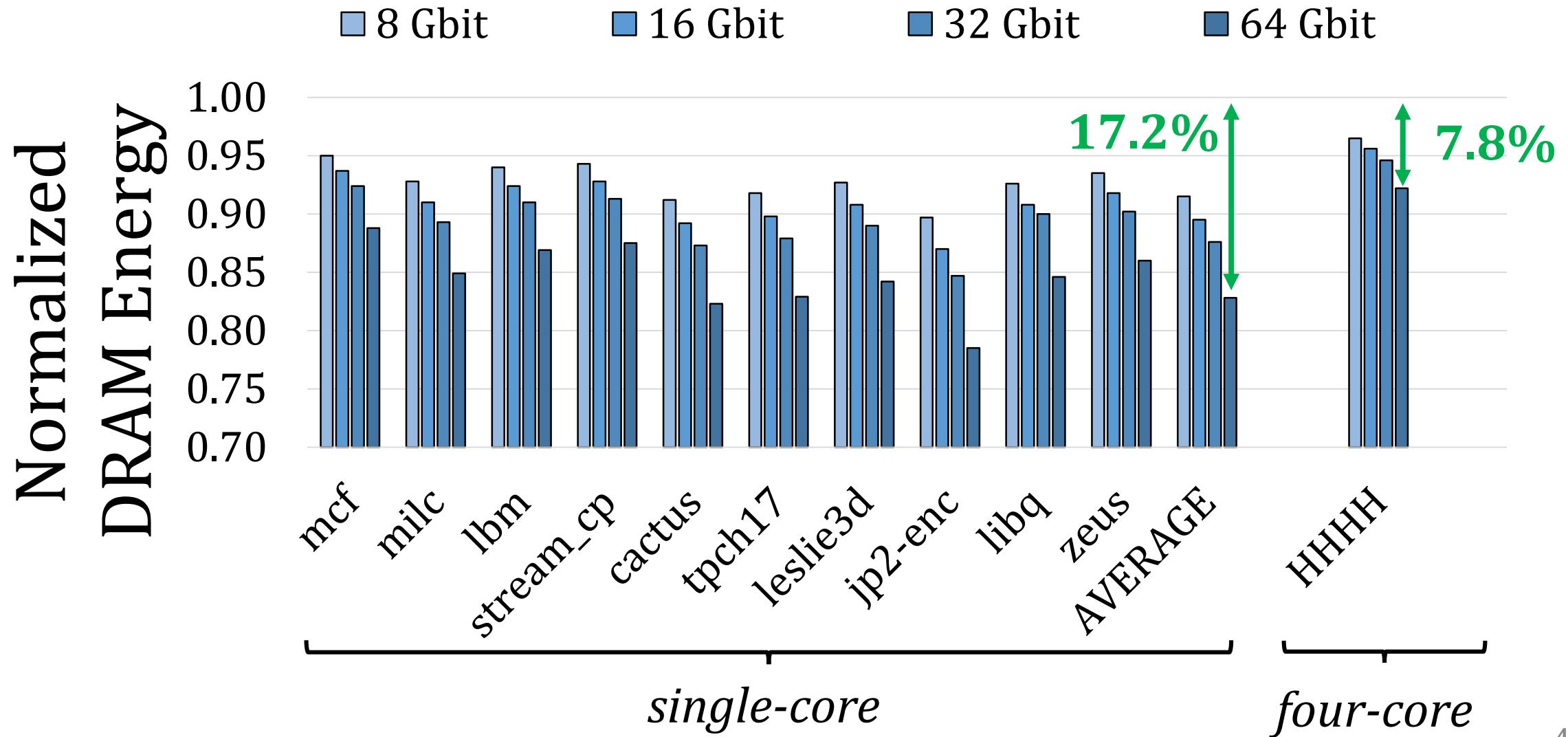
CROW-cache Performance



CROW-ref Performance

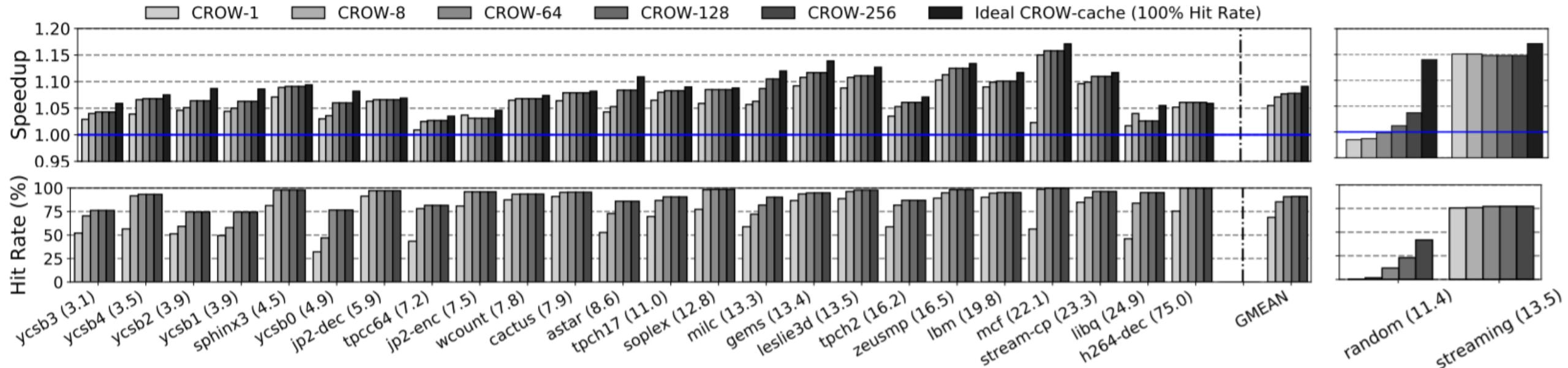


CROW-ref Energy Savings

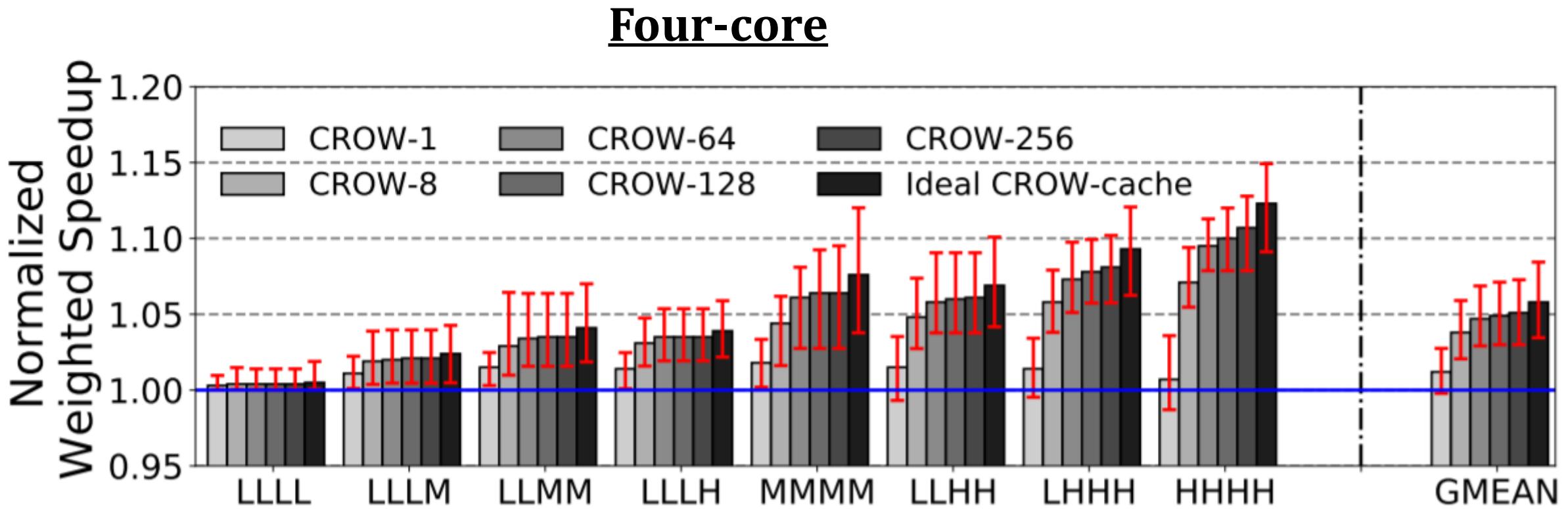


Speedup - CROW-cache

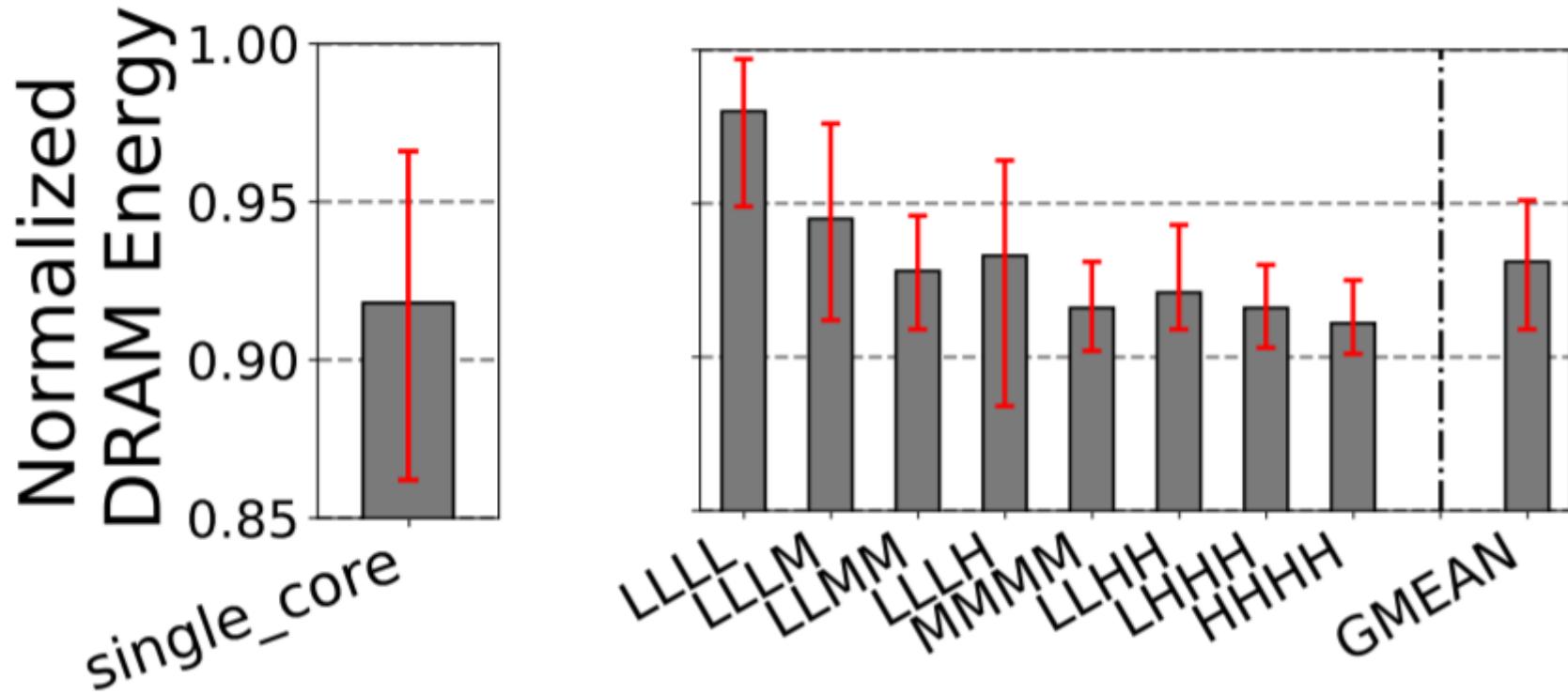
Single-core



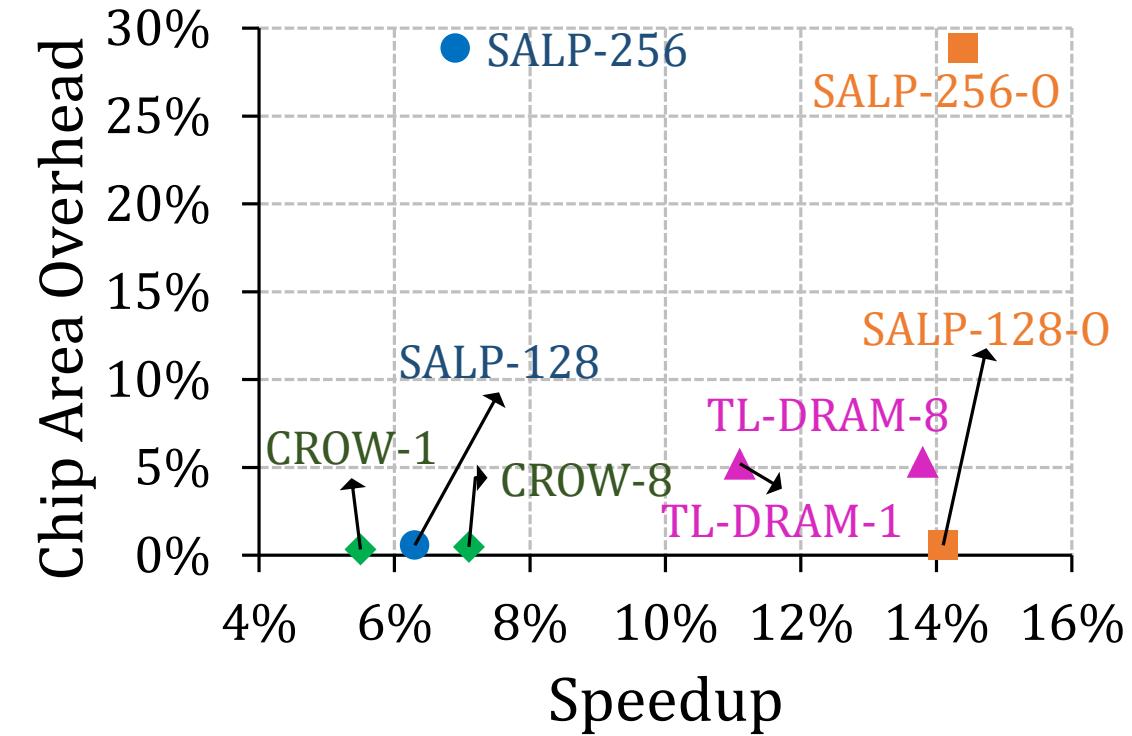
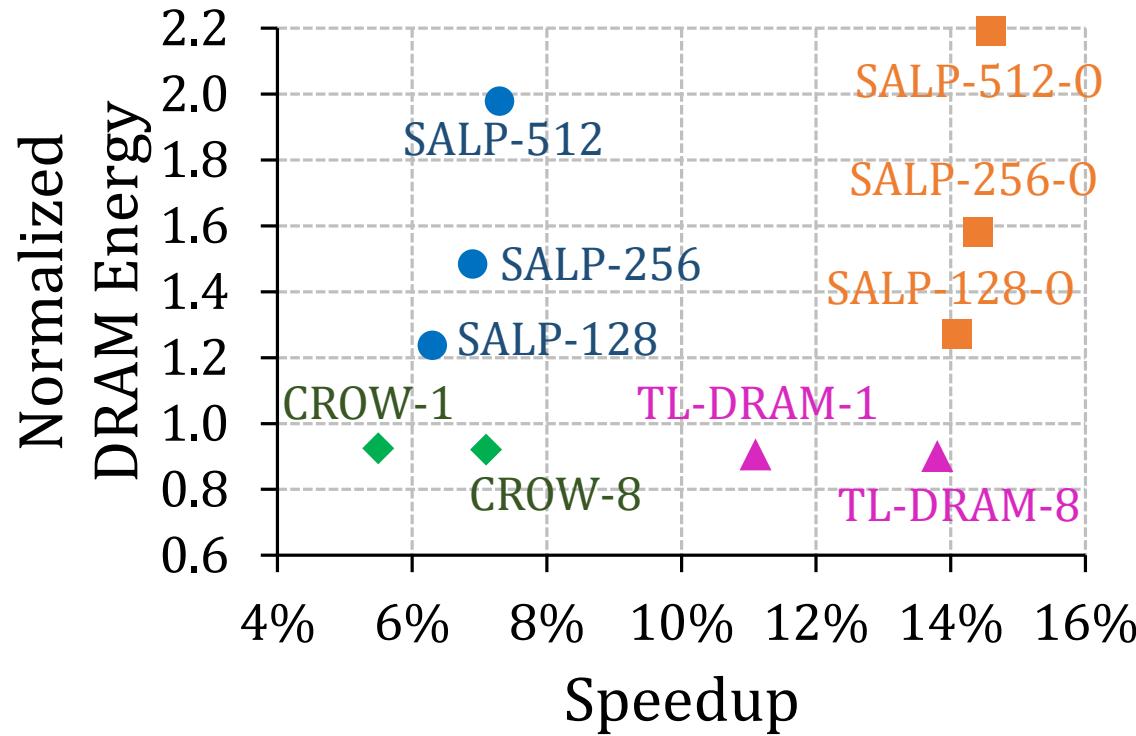
Speedup - CROW-cache



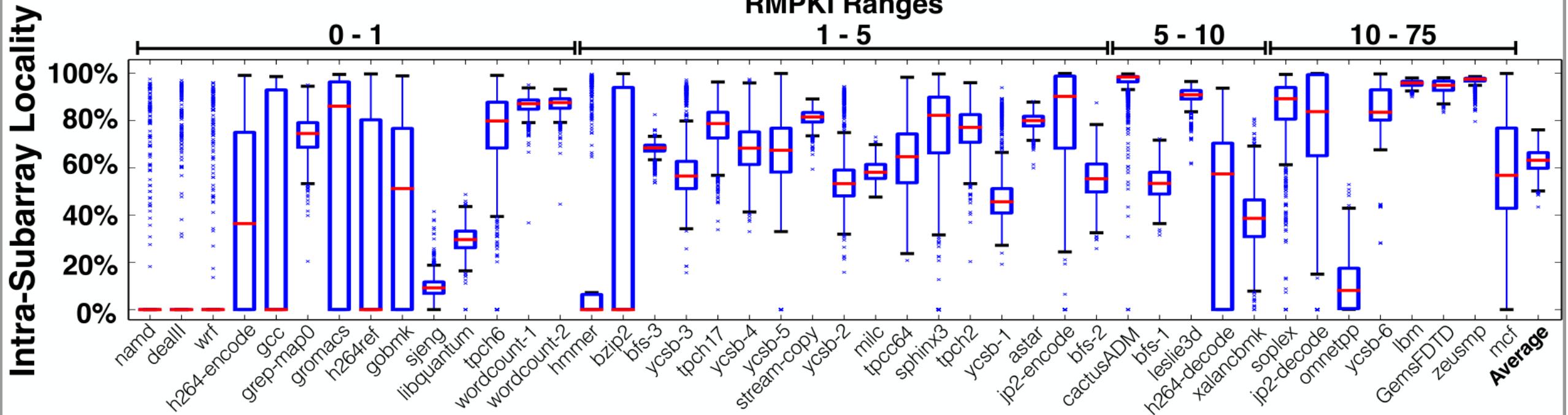
Energy - CROW-cache



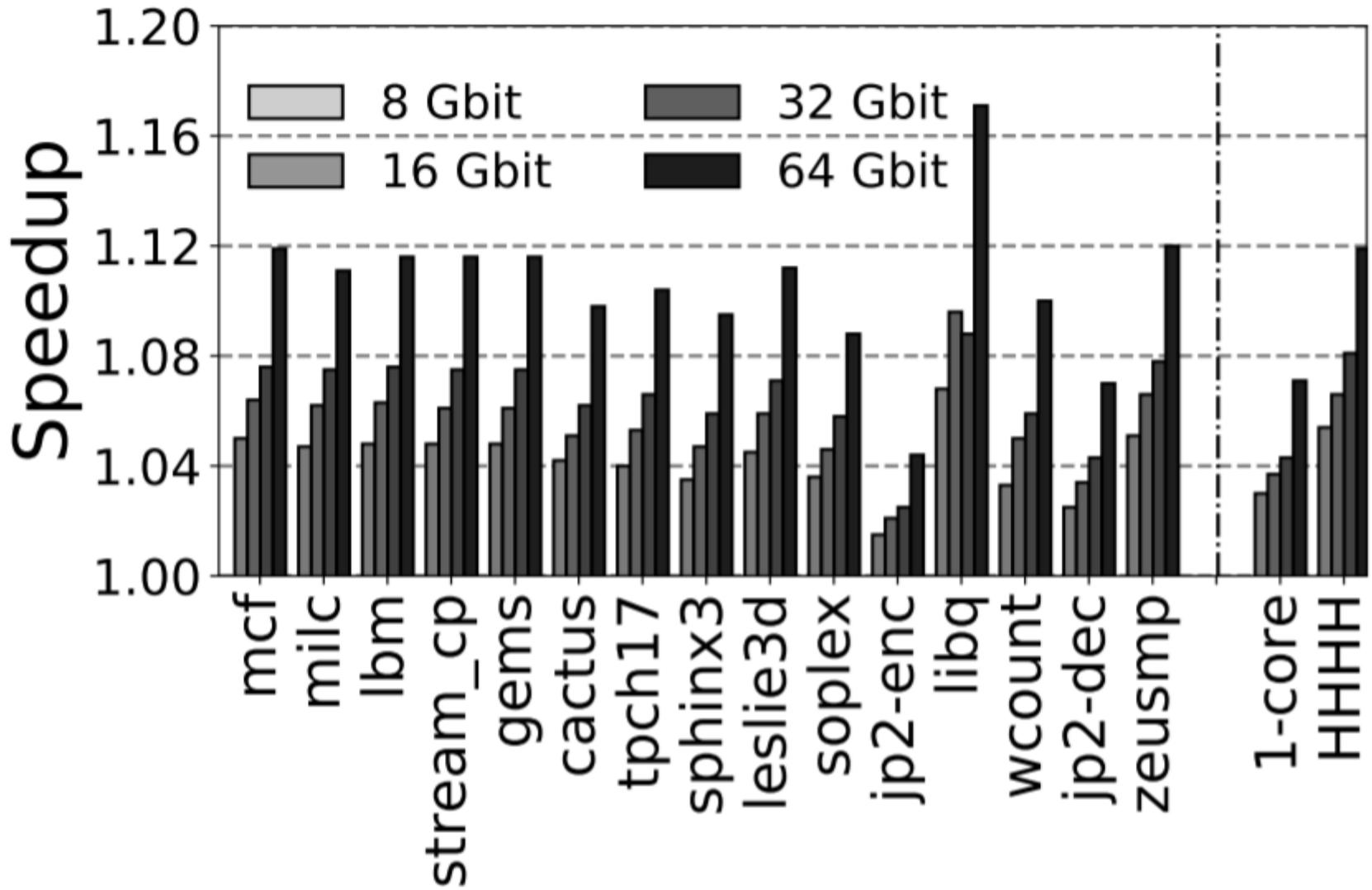
Comparison to TL-DRAM and SALP



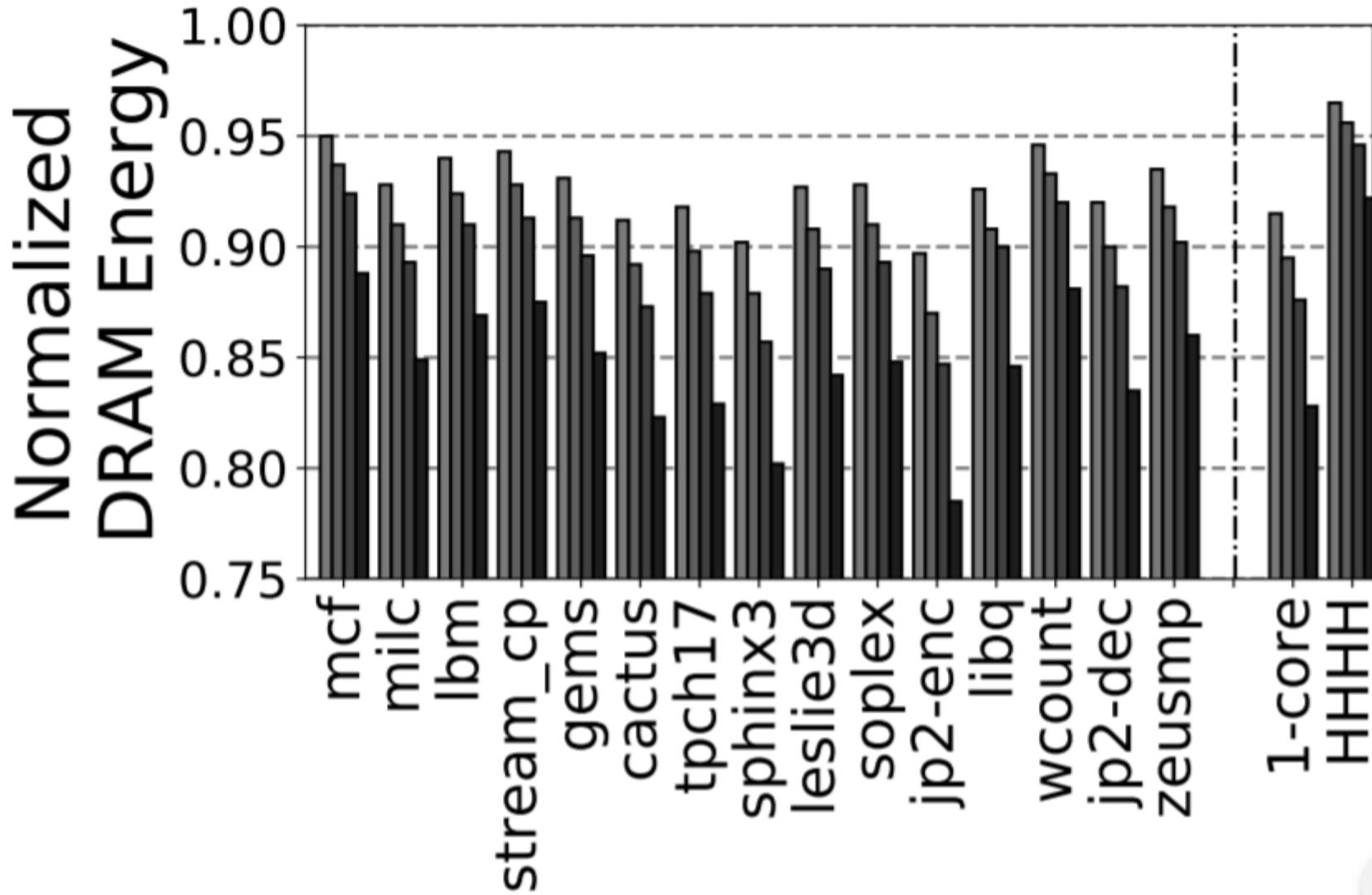
Slide on RLT



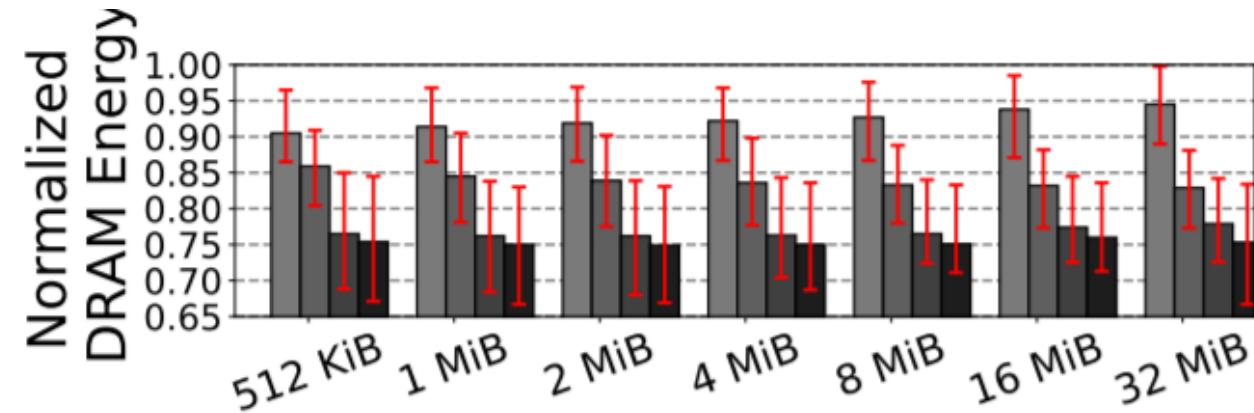
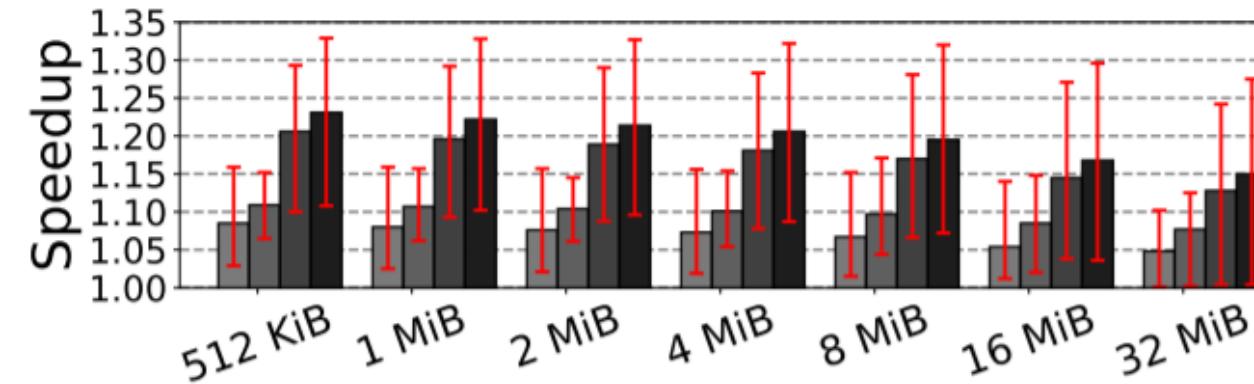
Speedup - CROW-ref



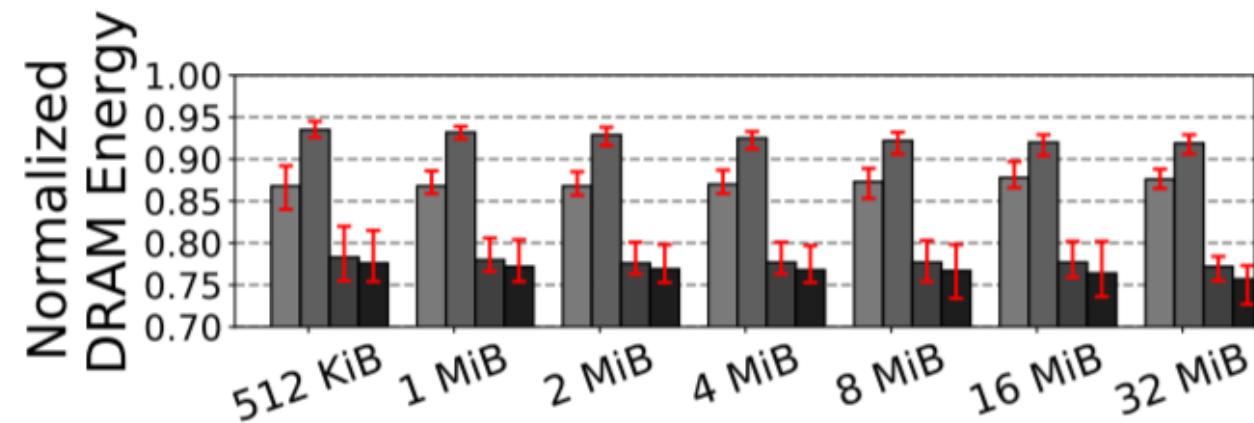
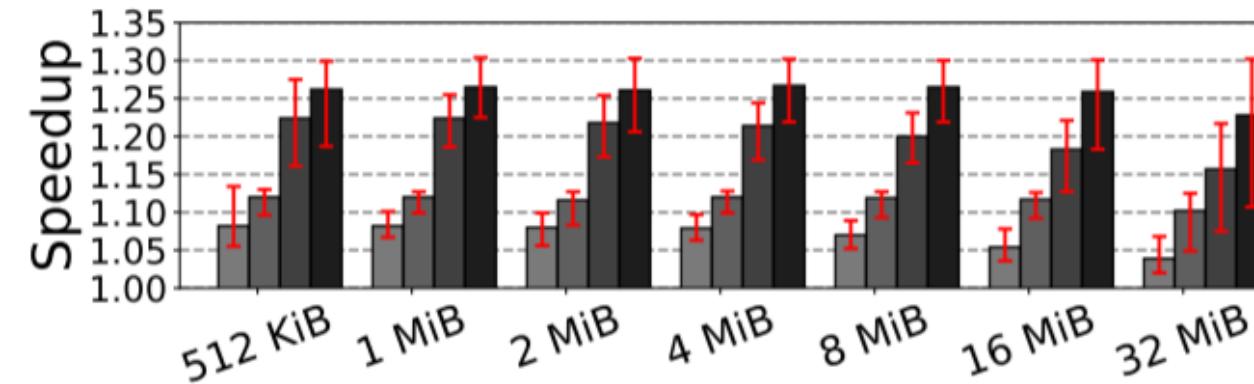
Energy - CROW-ref



CROW-cache + ref



(a) Single-core workloads



(b) Four-core workloads

CROW-table Organization

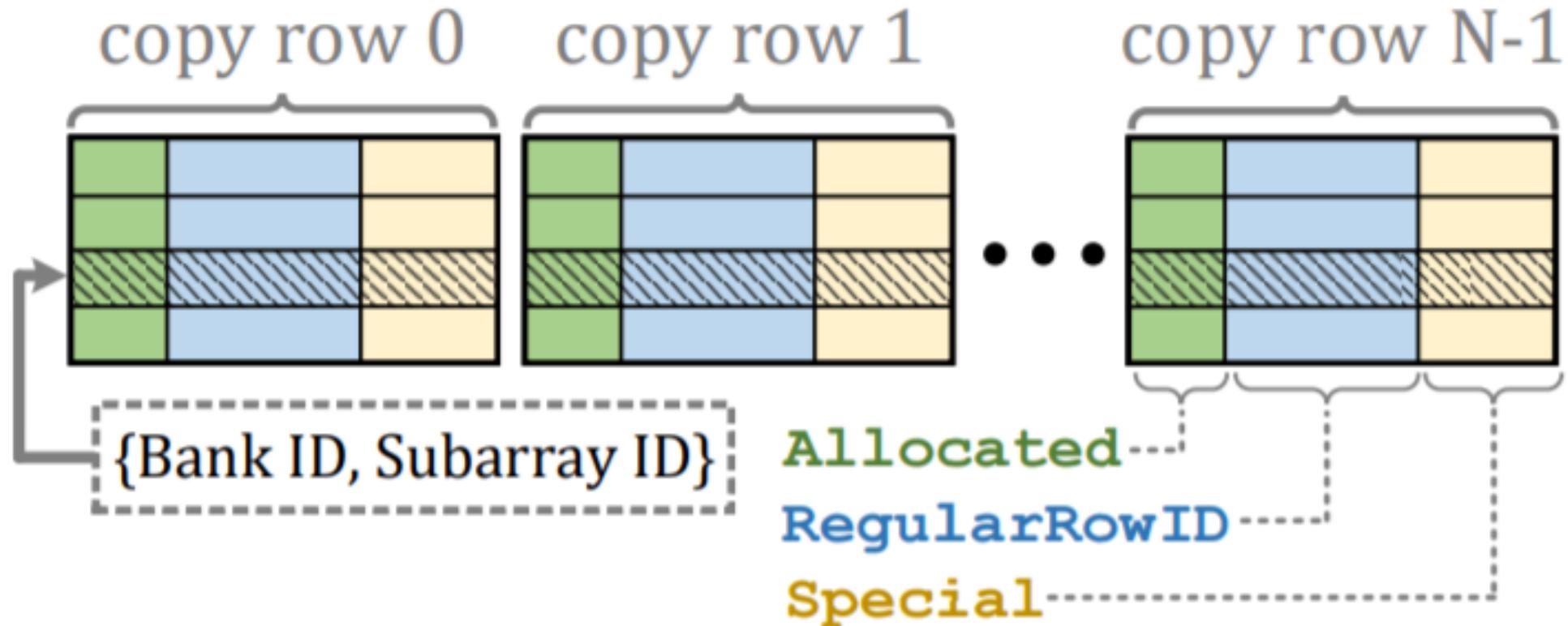


Figure 4: Organization of the CROW-table.

tRCD vs tRAS

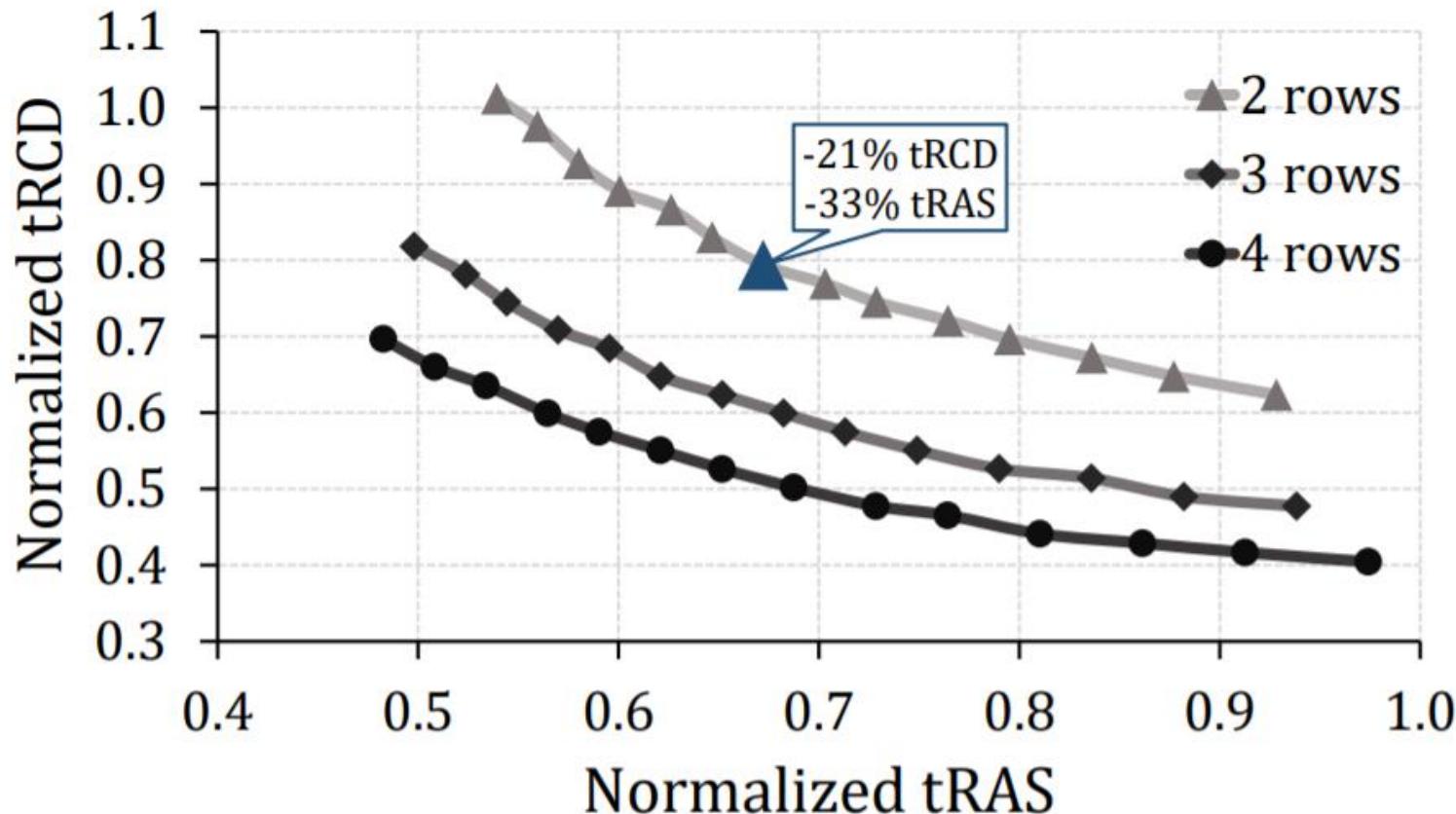


Figure 6: Normalized tRCD latency as a function of normalized tRAS latency for different number of simultaneously activated DRAM rows.

MRA Area Overhead

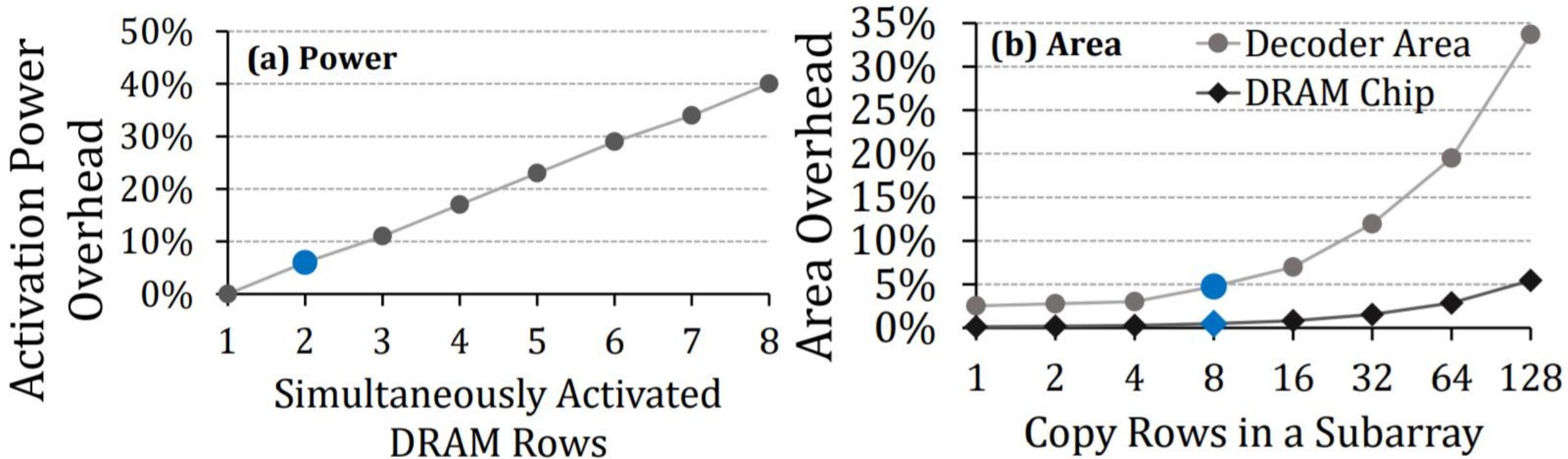


Figure 7: Power consumption and area overhead of MRA.

DRAM Charge over Time

