

# LADDER: Architecting Content and Location-aware Writes for Crossbar Resistive Memories

**Md Hafizul Islam Chowdhury**<sup>1</sup>, Muhammad Rashedul Haq Rashed<sup>1</sup>,  
Amro Awad<sup>2</sup>, Ricard Ewetz<sup>1</sup> and Fan Yao<sup>1</sup>

<sup>1</sup>Department of ECE  
University of Central Florida  
Florida, USA

<sup>2</sup>Department of ECE  
North Carolina State University  
North Carolina, USA



IEEE/ACM International Symposium on Microarchitecture (**MICRO'21**)  
October 18–22, 2021

# ReRAM: Emerging Memory Technology

---

- ❖ Increasing need for efficient and scalable memory systems.
- ❖ DRAM technologies do not match the capacity demand.
- ❖ **Resistive memory (ReRAM) is promising for main-memory integration.**
  - High cell-density and scalability
  - Low static power

ReRAM suffers from varying latency requirements for write, which could limit system performance if not utilized properly.

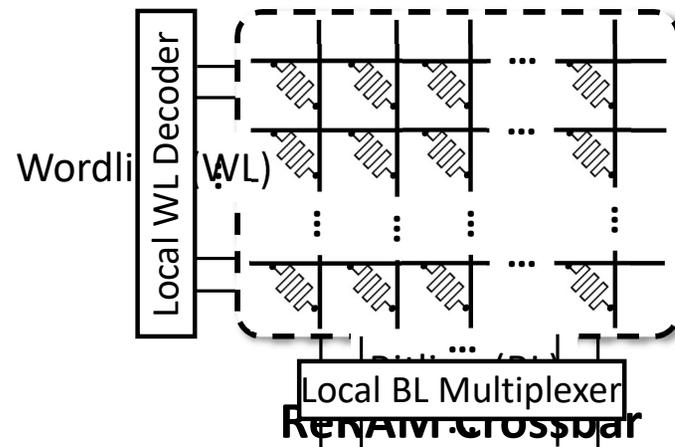
- ❖ Increasing demand for efficient and scalable memory system.
- ❖ Traditional memory (i.e., DRAM) **This work** match the capacity demand.
- ❖ ReRAM suffer from location/content-dependent variable write latency.

**LADDER - Processor-side low-overhead framework to improve ReRAM performance by exploiting variable write latency.**

ReRAM suffer from location/content-dependent variable write latency, limiting system performance.

# ReRAM Memory Organization

- ❖ ReRAM cells are arranged in dense array structures (called **Crossbar**).
  - Increase area efficiency
- ❖ ReRAM cell arrays form a **MAT** (i.e., crossbar size of 512x512 cells).
  - Contains peripheral circuitry to support read/write
  - The basic unit to form *banks, ranks and channels*

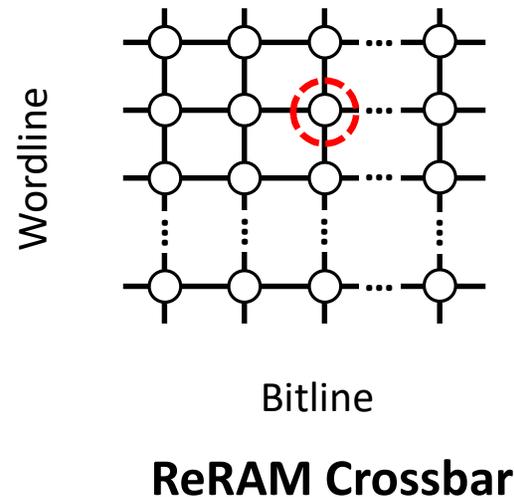


ReRAM MAT

# ReRAM Write Operations

---

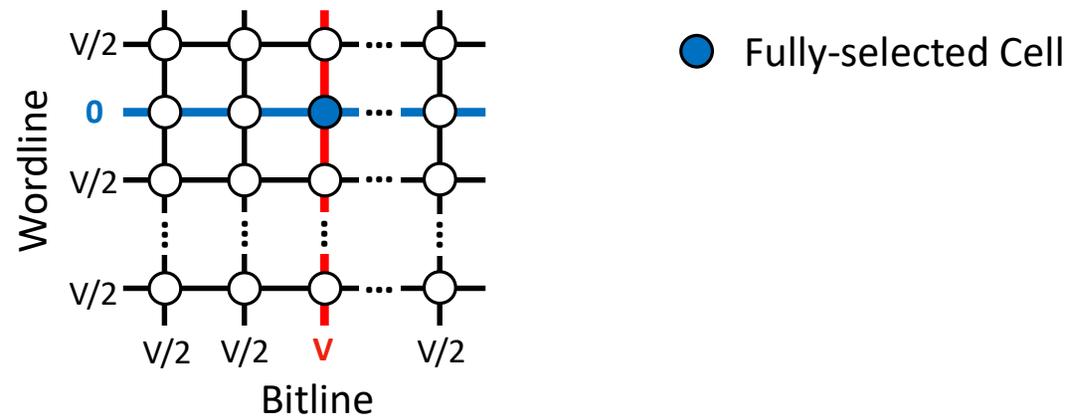
- ❖ ReRAM write operation involves two phases.
  - **RESET:** transition from *low-resistive state* to *high-resistive state* (i.e., '1' → '0')
  - **SET:** transition from *high-resistive state* to *low-resistive state* (i.e., '0' → '1')



# ReRAM Write Operations

❖ ReRAM write operation involves two phases.

- **RESET:** transition from *low-resistive state* to *high-resistive state* (i.e., '1' → '0')
- **SET:** transition from *high-resistive state* to *low-resistive state* (i.e., '0' → '1')

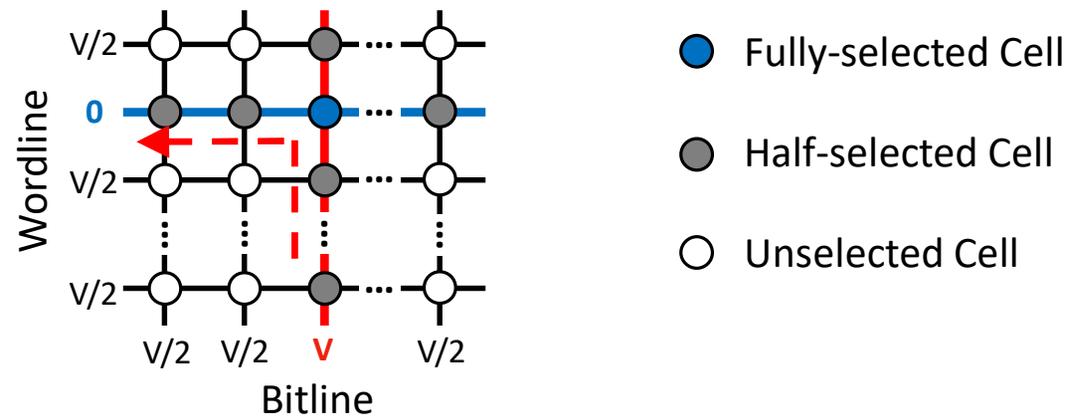


ReRAM Crossbar

# ReRAM Write Operations

❖ ReRAM write operation involves two phases.

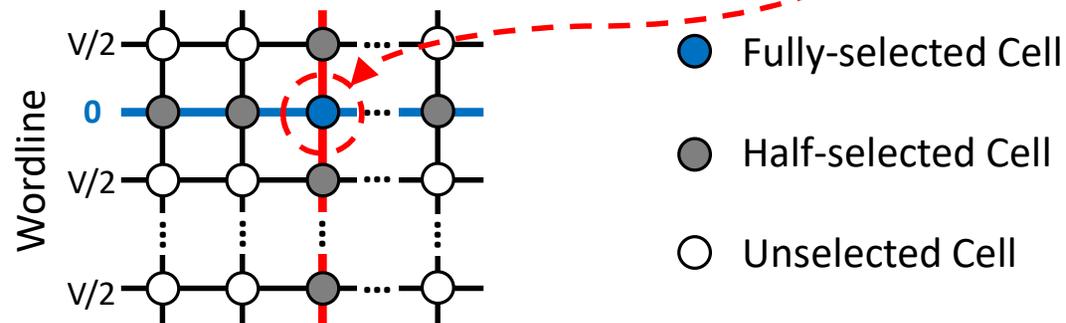
- **RESET:** transition from *low-resistive state* to *high-resistive state* (i.e., '1' → '0')
- **SET:** transition from *high-resistive state* to *low-resistive state* (i.e., '0' → '1')



ReRAM Crossbar

# ReRAM Write Operations

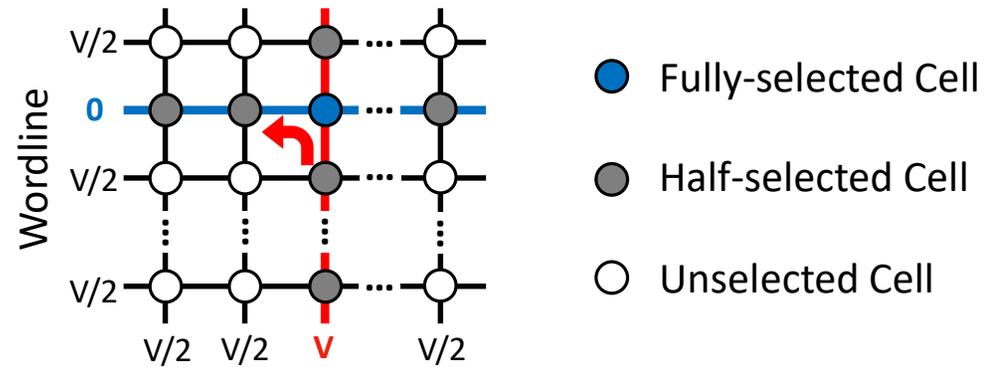
- ❖ ReRAM write operation involves two phases.
  - **RESET:** transition from *low-resistive state* to *high-resistive state* (i.e., '1' → '0')
  - **SET:** transition from *high-resistive state* to *low-resistive state* (i.e., '0' → '1')
- ❖ RESET latency,  $t = C * e^{-k|V_d|}$   $V_d =$  Voltage drop across target cell



RESET latency of a ReRAM cell is exponentially proportional to the voltage drop across the target cell.

# Variable RESET Latency

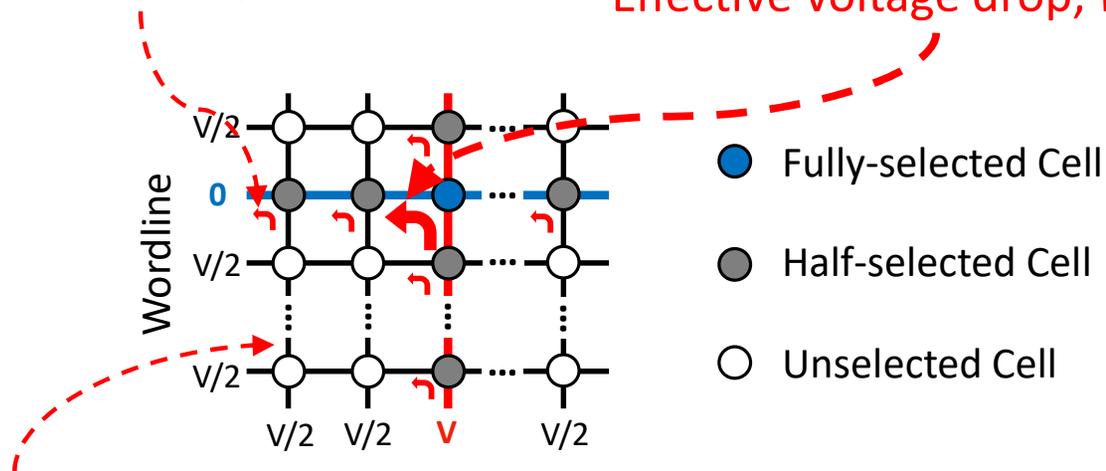
---



# Variable RESET Latency

*Sneak current through half-selected cells*

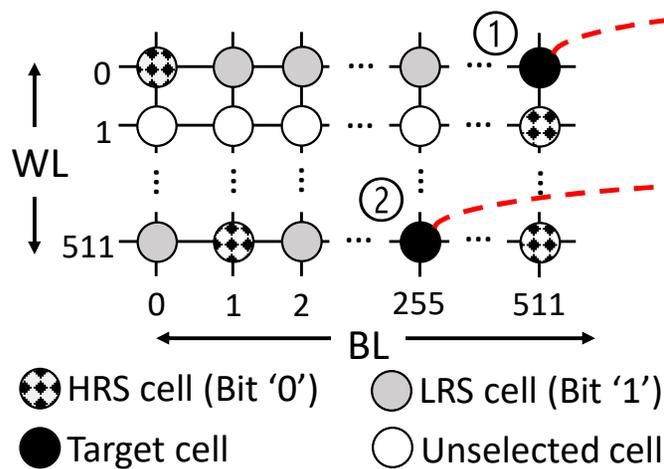
*Effective voltage drop,  $V_d < V$*



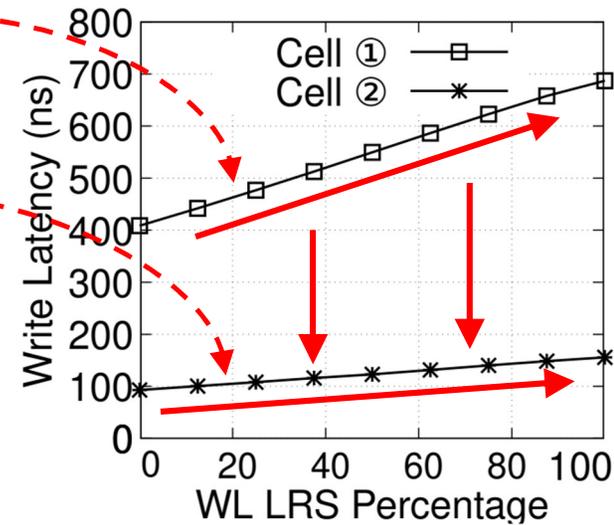
*Non-zero wire resistance*

$$\text{RESET latency, } t = C * e^{-k|V_d|}$$

# Location/Content Dependence of RESET



Crossbar showing two RESETs



Location/Content dependency of RESET latency

- ❖ RESET latency varies significantly based on **data-pattern and location**.
- ❖ Using worst case reset latency can significantly degrade system performance.

# Prior Works and Motivation

---

## ❖ Architecture and circuit-level techniques:

- Dynamic reset voltage to compensate the IR-drop (e.g., **Zokae et al. HPCA'20**)
- Track *bitline data patterns* with custom profiling circuitry (e.g., **Wen et al. TCAD'19**)
- Model *location dependent* RESET latency (e.g., **Zhang et al. DATE'16**)
- Limit sneak current at crossbar level (e.g., **Xu et al. HPCA'15**)

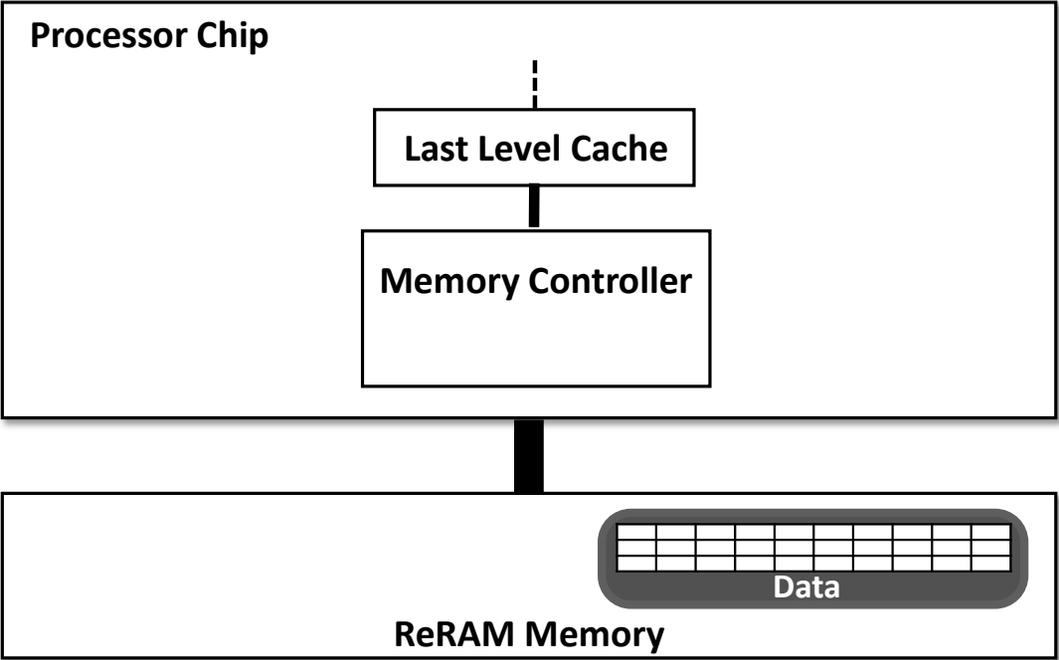
## ❖ Issues:

- Additional circuitry increases design complexity of memory subsystems
- Do not harness the full potential gains

A processor-side scheme that utilizes variable RESET latency, without adversely impacting the complexity of commodity ReRAM devices.

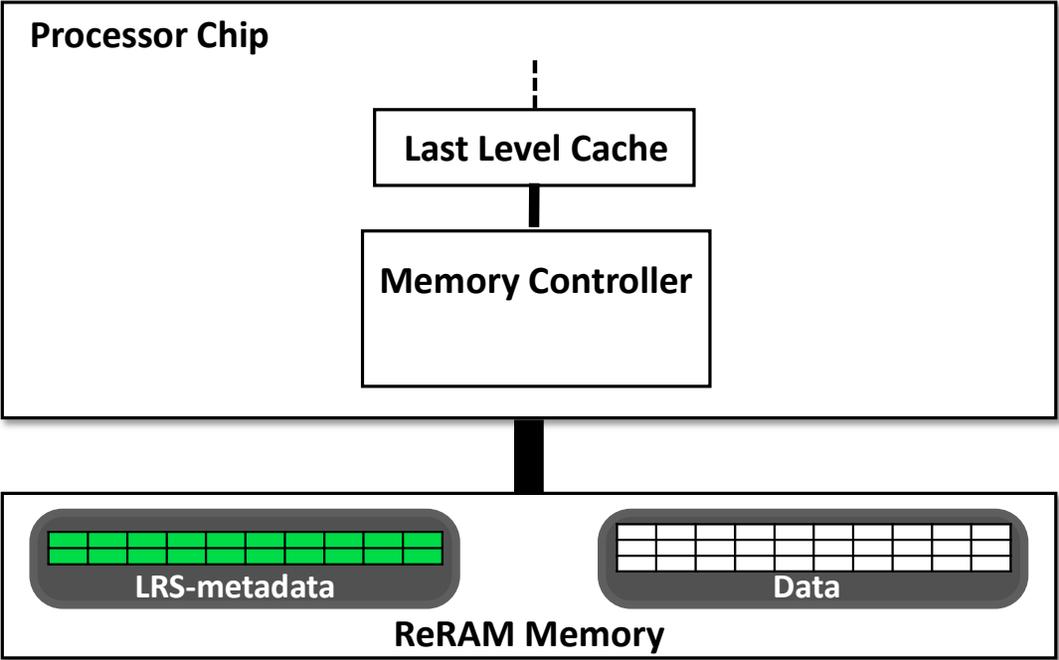
# Overview of LADDER

---



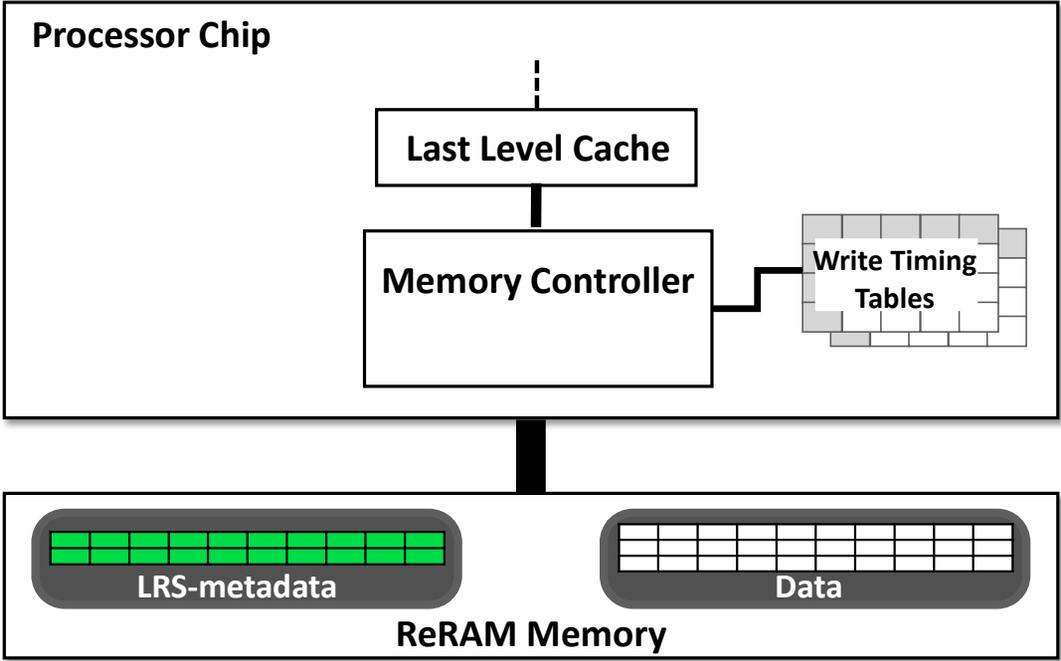
# Overview of LADDER

---



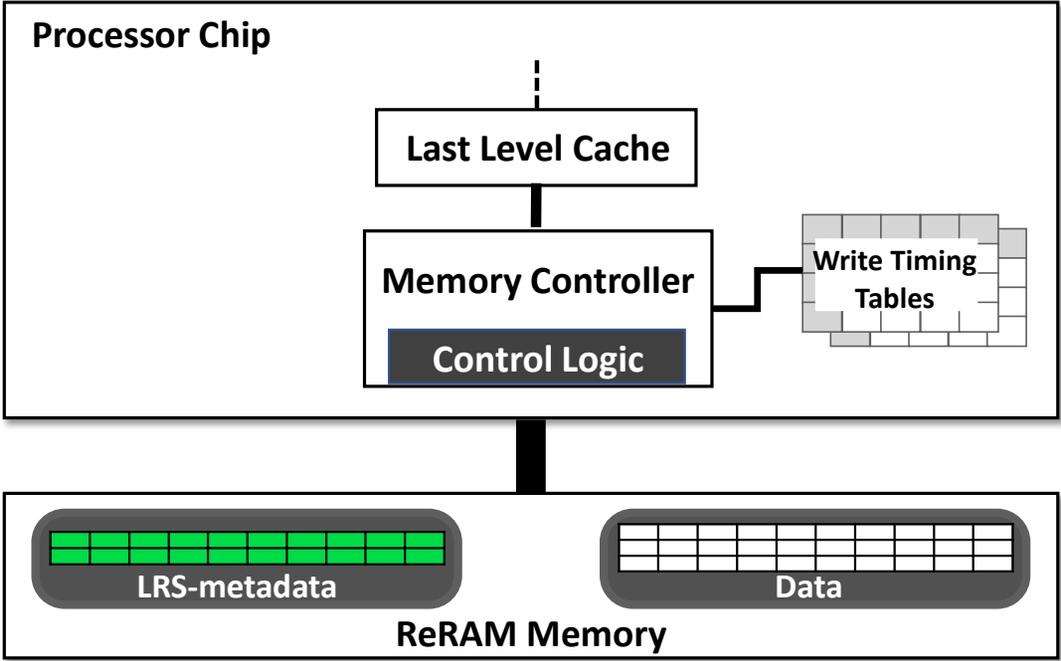
# Overview of LADDER

---



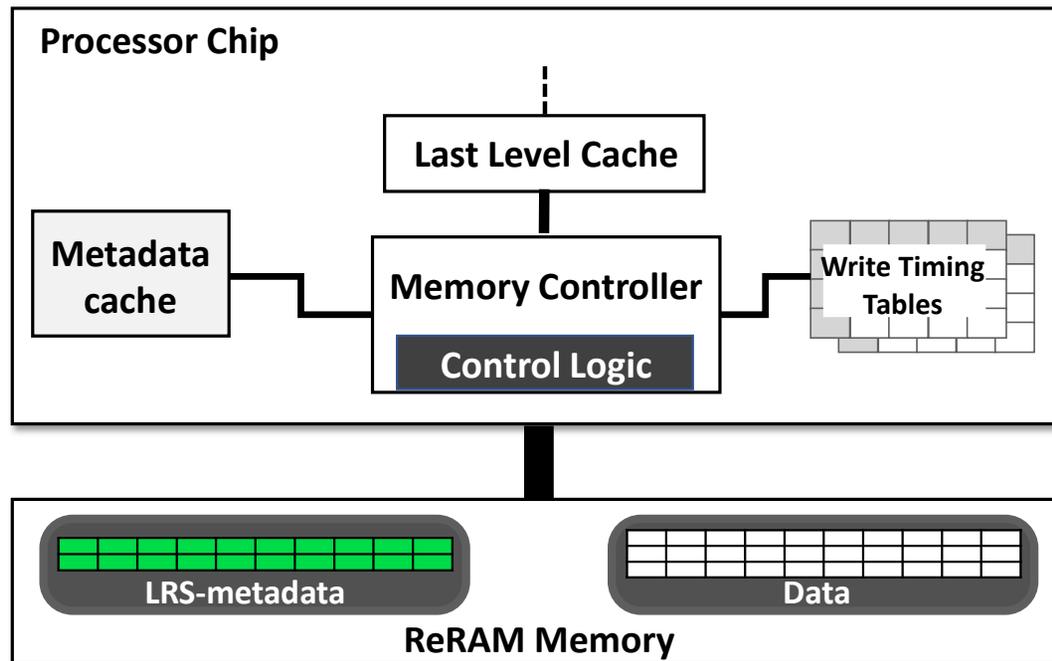
# Overview of LADDER

---

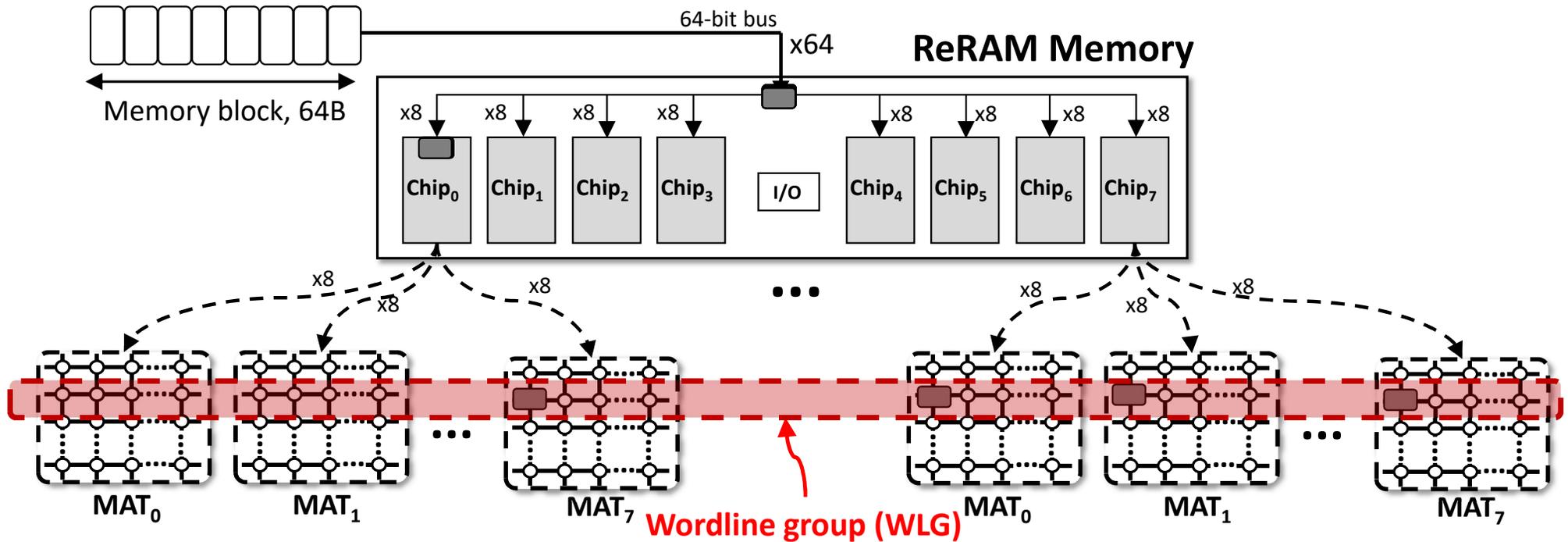


# Overview of LADDER

---



# Location & Data-Content Aware Latency Model

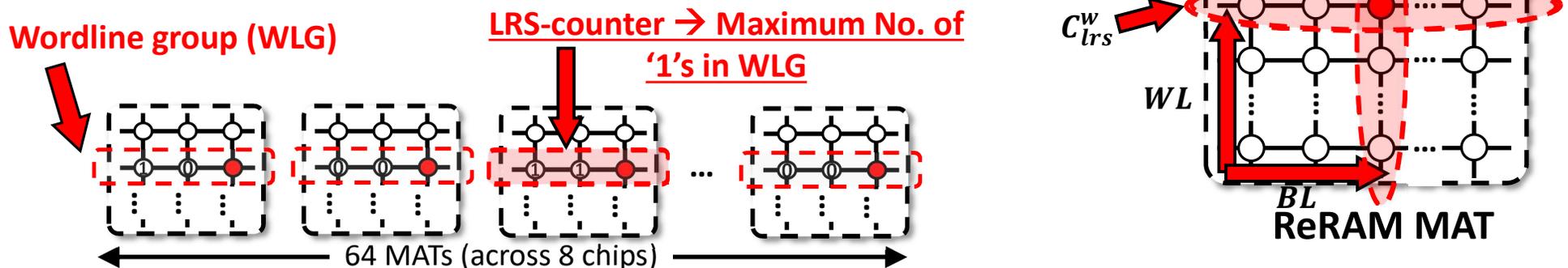


- ❖ Each MAT stores 1 byte (8-bit) per memory block.
- ❖ 8 MATs in each chip are used to store one memory block (64 WLGs total).

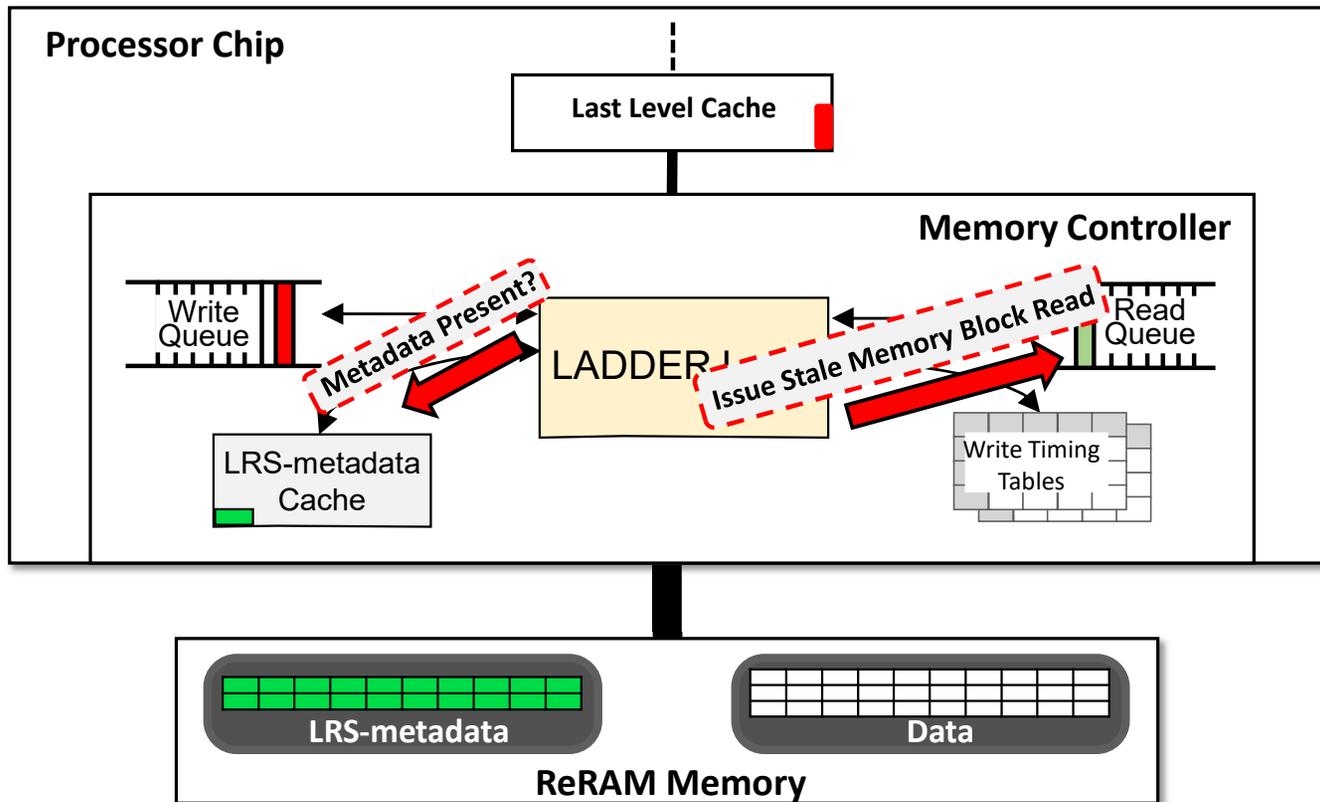
# Location & Data-Content Aware Latency Model

- ❖ Ideally, both WL and BL data pattern should be monitored.
  - i.e., keep the counter of '1's in each row and column
  - Profiling BL content (i.e., column) in memory controller is prohibitively expensive
- ❖ Tradeoff: Track number of '1's in **WL (i.e., row) only** (LRS metadata)
- ❖ WL and BL location and WL content of target cell,  $\langle WL, BL, C_{lrs}^w \rangle$ .

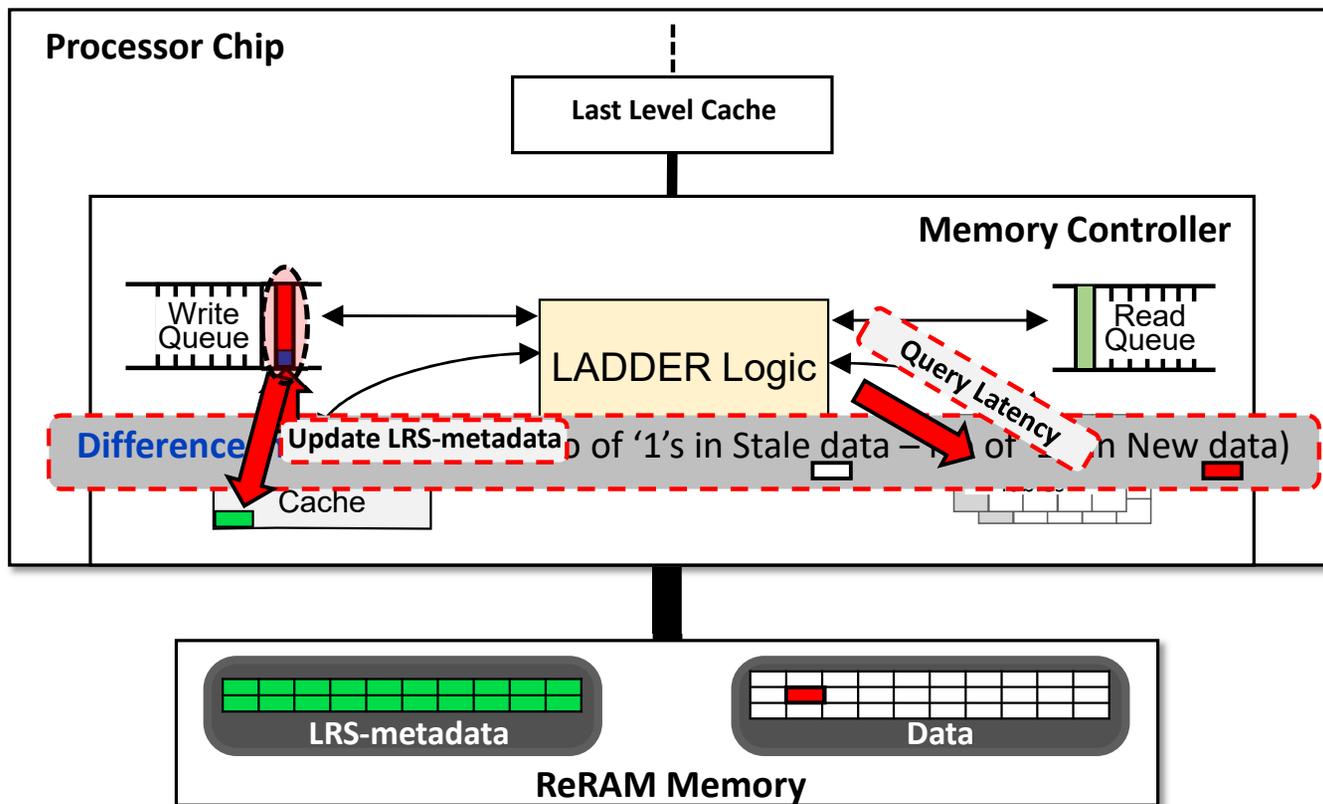
- $WL, BL$ : Wordline and bitline locations of a target cell
- LRS-counter,  $C_{lrs}^w$ : Maximum number of '1's across WLG



# LADDER-Basic Data Write Operation



# LADDER-Basic Data Write Operation



# Optimization Opportunities with LADDER-Basic

---

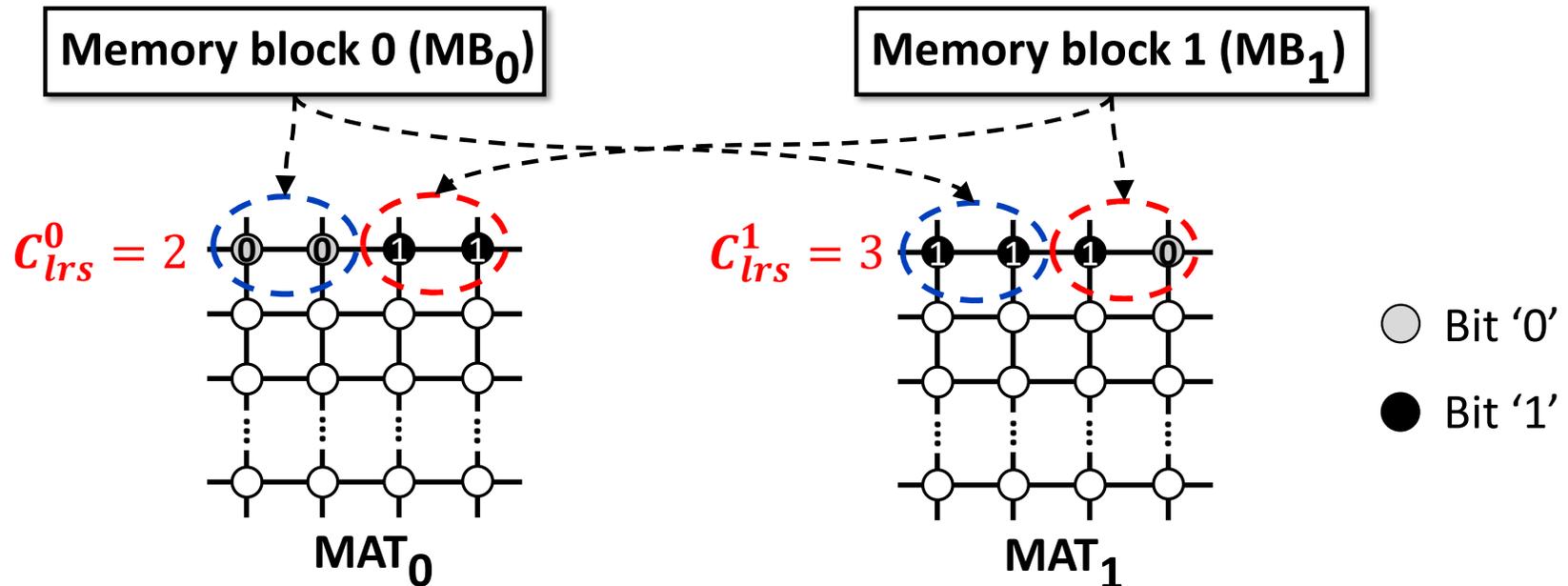
- ❖ Each data write requires a Stale Memory Block (SMB) read.
- ❖ Non-trivial metadata maintenance overhead.
  - Additional reads and writes for LRS-metadata
  - Metadata storage overhead
- ❖ How to get rid of Stale Memory Block (SMB) Reads?
- ❖ How to reduce LRS-metadata maintenance overhead?

# LADDER-Est: Eliminating SMB Reads

---

- ❖ RESET latency depends only on maximum no. of '1's in WLG.
- ❖ LRS-metadata estimation scheme:
  - Tracks only the worst-case byte in a newly-written data as metadata
  - Memory controller can update no. of '1's in the worst-case byte without old data

# LRS-metadata Estimation Principle

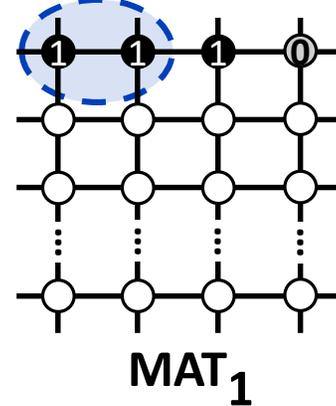
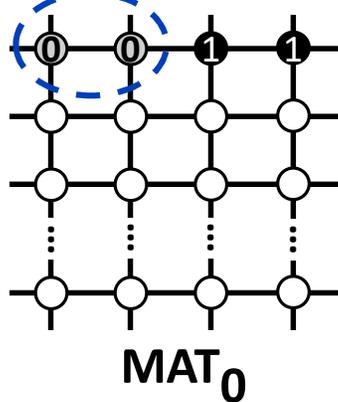


Accurate LRS-counter (LADDER-Basic),  $C_{lrs}^w = 3$

# LRS-metadata Estimation Principle

Memory block 0 ( $MB_0$ )

Worst case byte,  $S_0^M = 2$



○ Bit '0'  
● Bit '1'

Accurate LRS-counter (LADDER-Basic),  $C_{lrs}^w = 3$

# LRS-metadata Estimation Principle

Estimated counter ( $\sum S_i^M$ )  $\geq$  Actual counter ( $C_{lrs}^w$ )

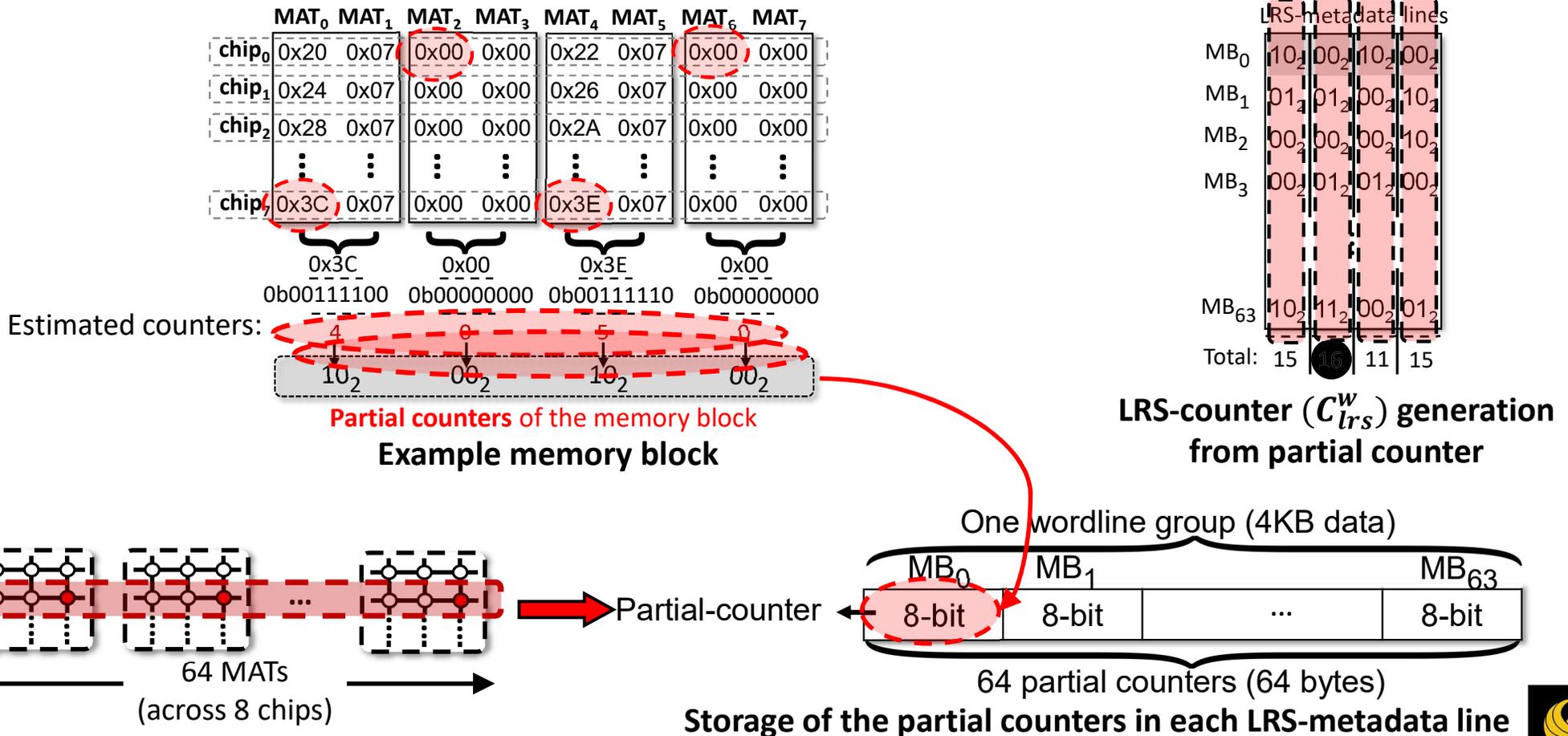
RESET Latency corresponding to estimated counter is equal or higher than minimally required latency.

$\sum S_i^M$  is updated using the newly written block  $\rightarrow$  No more SMB reads

Accurate LRS-counter (LADDER-Basic),  $C_{lrs}^w = 3$

Estimated LRS-counter,  $\sum S_i^M = 4$

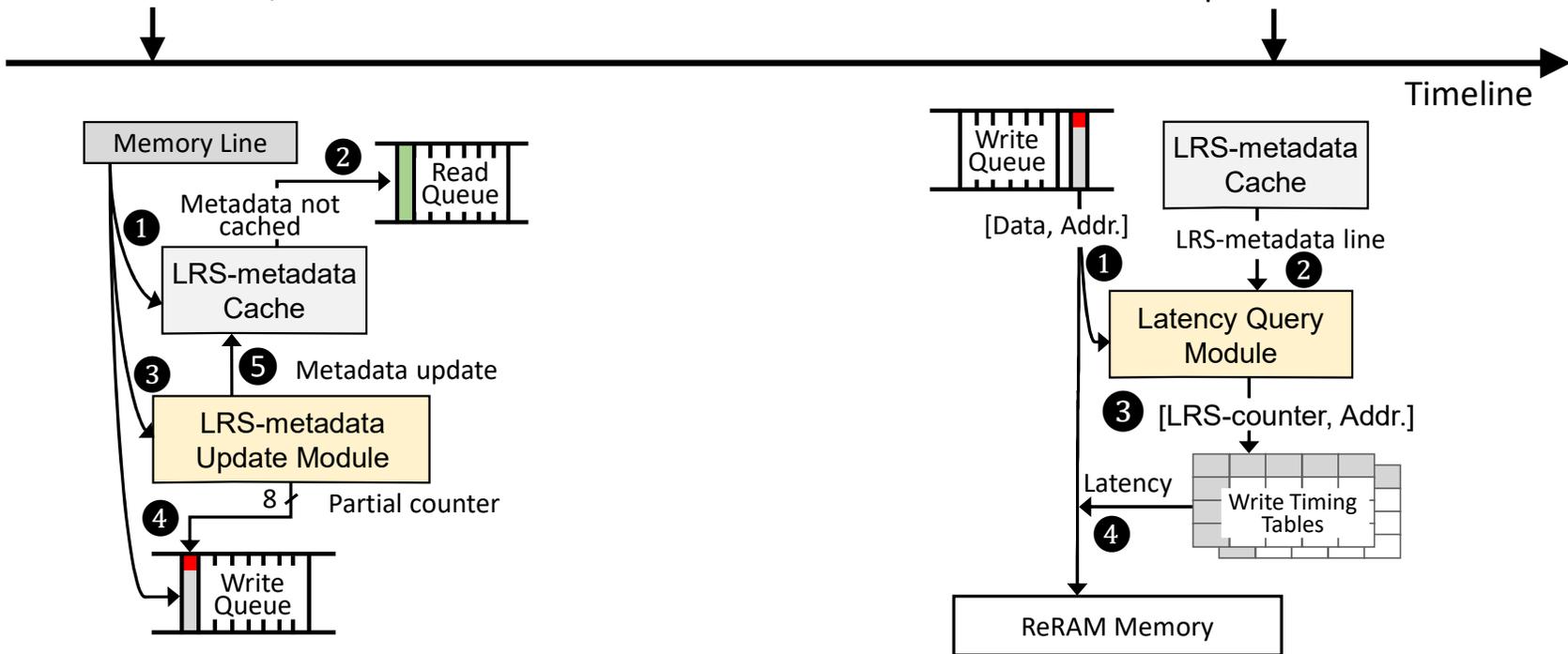
# LADDER-Estimate Scheme



# LADDER-Est: Optimized LADDER Logic

Write arrives at memory controller

Write is dispatched to ReRAM memory



Partial counter generation and LRS-metadata line update

LRS-counter generation and the RESET latency determination

# Improving Estimation Performance with Shifting

- ❖ **Observation:** Logical value '1's are clustered together in a few mats.
- ❖ Each data block in the WLG leverages a distinct shift based on its position in WL.
- ❖ During read, reverse shift is performed to restore original bit order.

	MAT <sub>0</sub>	MAT <sub>1</sub>	MAT <sub>2</sub>	MAT <sub>3</sub>
Memory Block <sub>0</sub>	1	3	0	1
Memory Block <sub>1</sub>	2	3	0	0
Memory Block <sub>2</sub>	1	3	0	1
	⋮	⋮	⋮	⋮

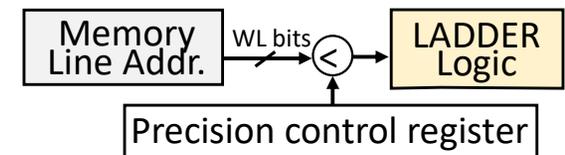
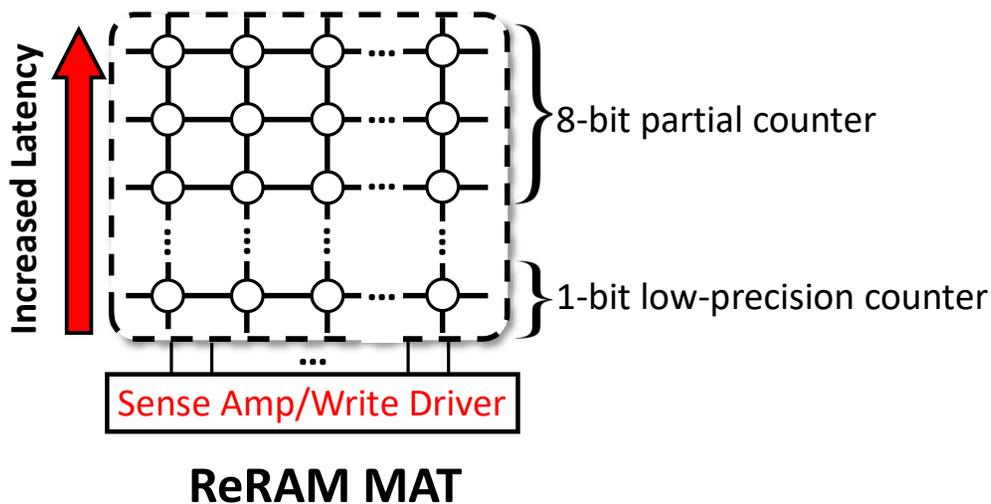
LRS cell count in each MAT corresponding to different memory blocks

	MAT <sub>0</sub>	MAT <sub>1</sub>	MAT <sub>2</sub>	MAT <sub>3</sub>
Memory Block <sub>0</sub>	1	3	0	1
Memory Block <sub>1</sub>	0	2	3	0
Memory Block <sub>2</sub>	0	1	1	3
	⋮	⋮	⋮	⋮

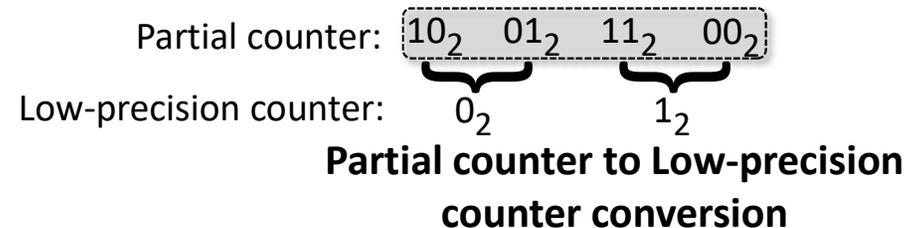
Intra-line bit-level shifting in data lines

# LADDER-Hybrid: Reducing LRS-metadata Overhead

- ❖ WLS closer to the write driver are **relatively insensitive** to WL-contents.
- ❖ **Multi-Granularity LADDER Counters:** Selectively reduce LRS-counter precision.

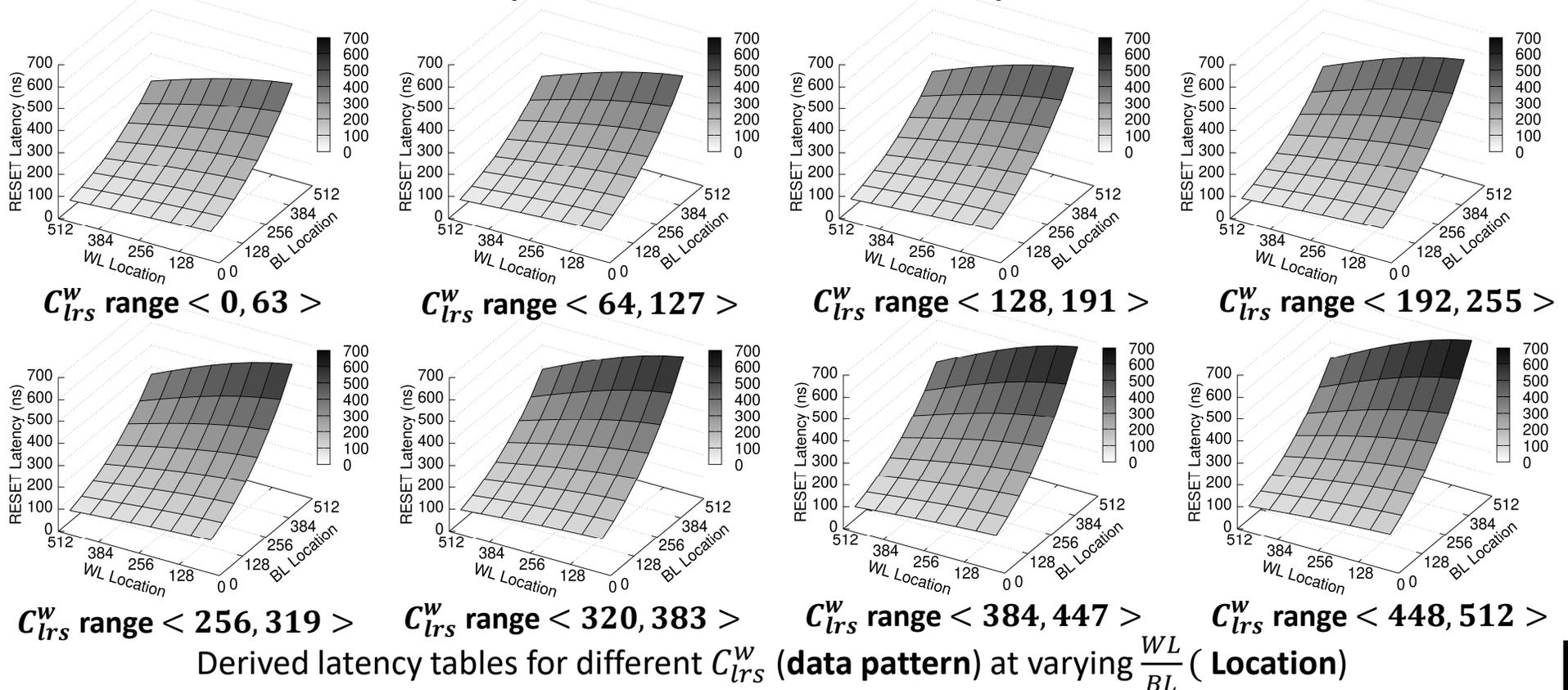


**LADDER architecture with multi-granularity counter**



# LADDER Latency Model

❖ Modified nodal analysis to obtain the latency model.



# Evaluation Methodology

---

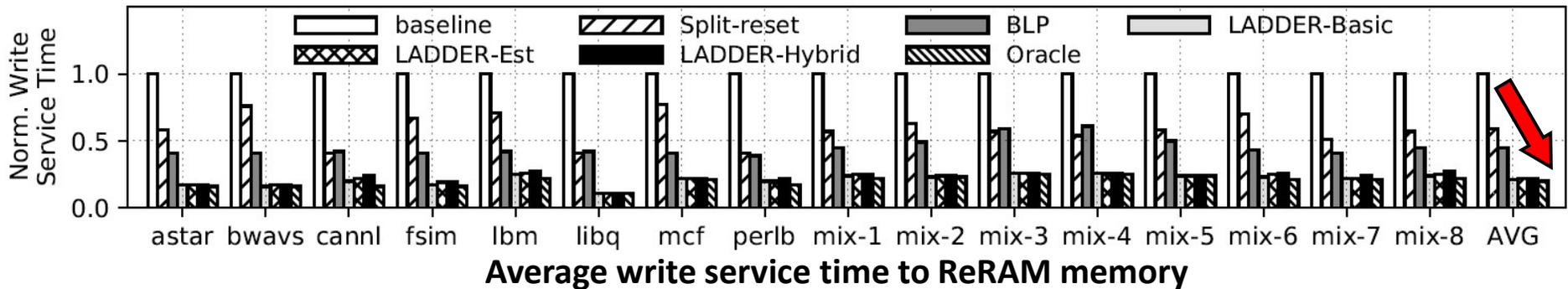
- ❖ **Simulator:** Gem5 full-system simulation (**Kernel:** Linux 3.4).
  - ❖ **CPU:** 4-core, Out-of-order, x86
  - ❖ **Cache:** Private L1/L2 cache, shared L3 cache
  - ❖ **Memory:** 16GB dual channel ReRAM memory, 256 MATs/bank, 512x512 crossbar
- ❖ **Workloads:** SPEC2006 (reference input), PARSEC2 (sim-large input).
- ❖ **Schemes:**
  - **Baseline:** Worst case fixed latency
  - **Prior state-of-the-art schemes:** Bitline data pattern (BLP)<sup>1</sup>, Split-reset<sup>2</sup>
  - **LADDER:** LADDER-Basic, LADDER-Estimation with shifting, LADDER-Hybrid
  - **Oracle:** Location and content-aware writes with no overhead (theoretical)

1. *Exploiting In-memory Data Patterns for Performance Improvement on Crossbar Resistive Memory*, IEEE TCAD'19

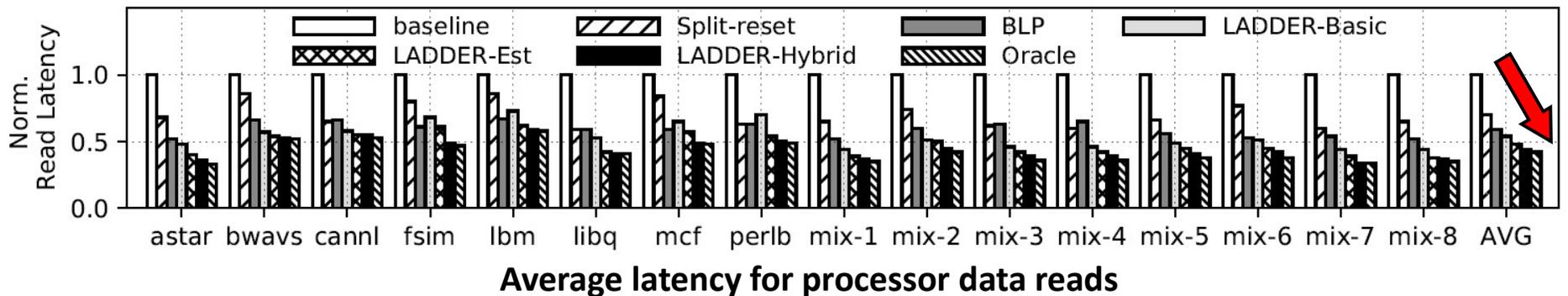
2. *Overcoming the challenges of crossbar resistive memory architectures*, HPCA'15

# Evaluation: Read/Write Performance

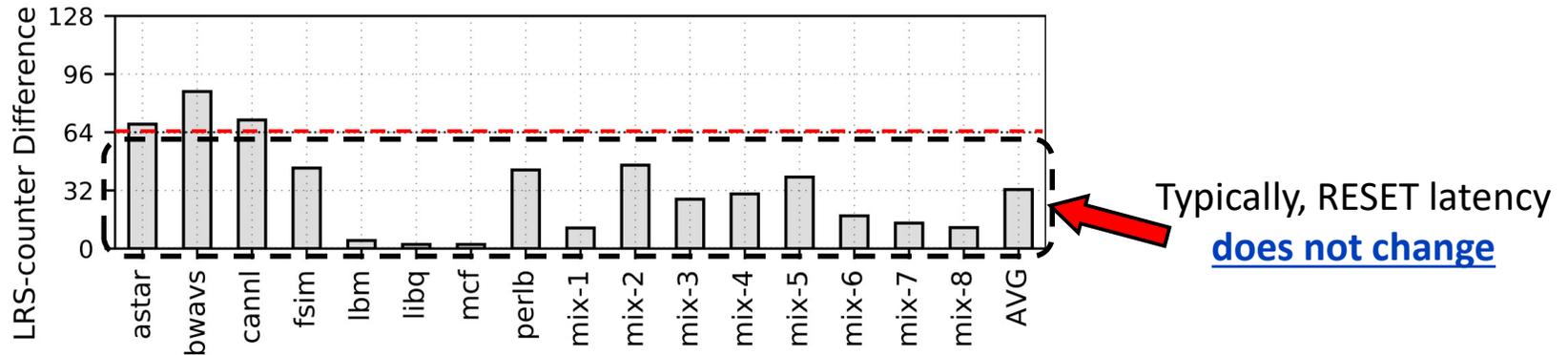
**79% write latency reduction compared to baseline**    **37% and 23% reduction compared to Split-reset and BLP**



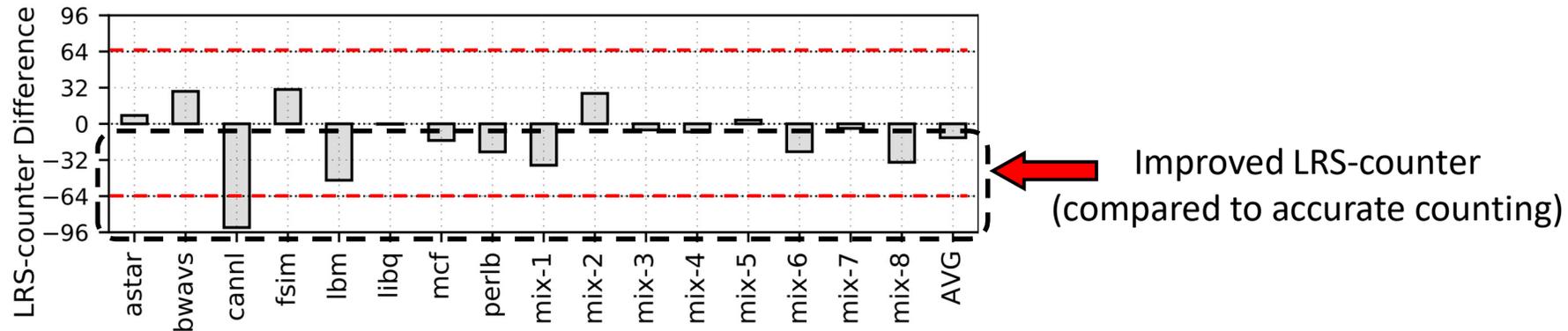
**56% read latency reduction compared to baseline**    **37% and 16% reduction compared to Split-reset and BLP**



# Evaluation: LRS-metadata Estimation Effectiveness



LRS-counter differences between LADDER-basic and LADDER-Est **without** Shifting

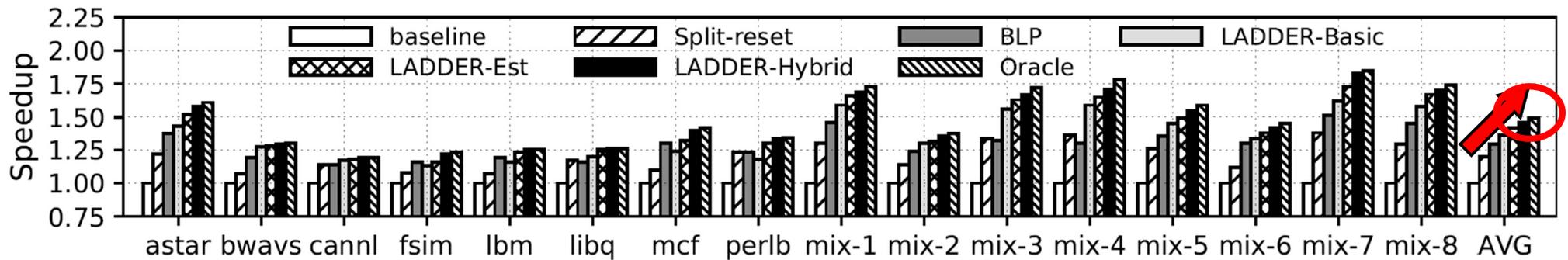


LRS-counter differences between LADDER-basic and LADDER-Est **with** Shifting

# Evaluation: Overall Speedup

On average, 22% and 13% speedup over Split-reset and BLP respectively

98% of the performance of the ideal scheme



# Evaluation: Hardware Overhead Analysis

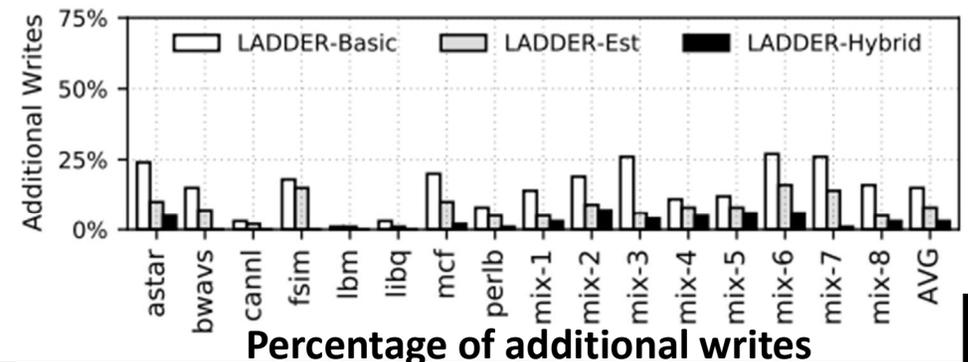
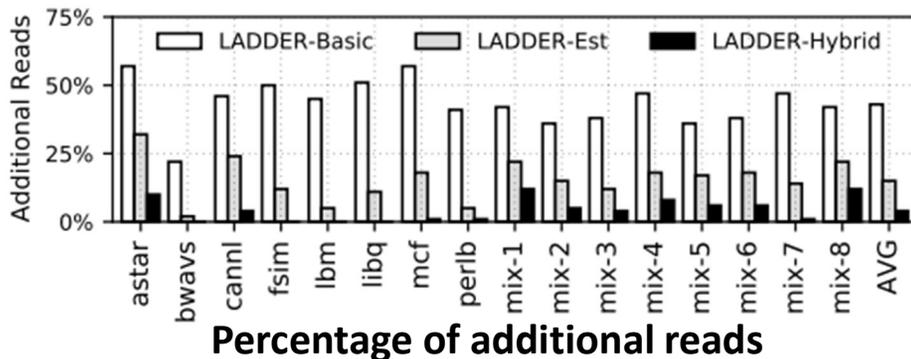
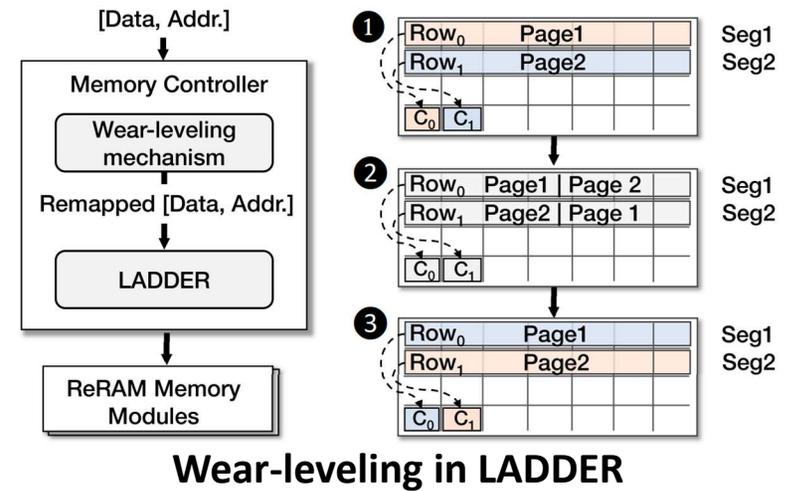
---

- ❖ ReRAM storage overhead: **1.56%** for LADDER-Est and **0.97%** for LADDER-Hybrid.
- ❖ On-chip storage for LADDER latency table: **512 Bytes** (8x8x8).
- ❖ LADDER logic and LRS-metadata cache overhead.
  - Implemented LADDER logic using Verilog
  - Synthesized using Synopsis Design Compiler with 45nm
  - Used CACTI7 to *LRS-metadata Cache*

<i>Module</i>	<i>Area (mm<sup>2</sup>)</i>	<i>Power (mW)</i>	<i>Latency (ns)</i>
<i>LRS-metadata Update Module</i>	0.0061	3.71	0.17
<i>Latency Query Module</i>	0.0047	6.57	0.32
<i>LRS-metadata Cache (64KB)</i>	0.2442	48.83	0.81

# More on Paper

- ❖ Additional details of LADDER designs.
- ❖ Overhead analysis of LADDER:
  - Overhead of metadata maintenance
  - Dynamic energy consumption overheads
- ❖ LADDER with wear-leveling techniques.
- ❖ Crash-consistency of LADDER.
- ❖ And more...



# Conclusions

---

- ❖ **LADDER** – a *processor-side* scheme improving write performance for ReRAM crossbar main memories.
- ❖ A counter-based mechanism (LRS-metadata) to track WL data pattern.
- ❖ Several novel optimizations to further enhance LADDER.
- ❖ Achieves considerable performance and energy efficiency improvements over state-of-the-art techniques with minimal overheads.

# Thanks! Questions?

Md Hafizul Islam Chowdhuryy

Email: [reyad@knights.ucf.edu](mailto:reyad@knights.ucf.edu)



UCF