Revisiting Residue Codes for Modern Memories

Evgeny Manzhosov

Adam Hastings, Meghna Pancholi, Ryan Piersma, Mohamed Tarek Ibn Ziad, and Simha Sethumadhavan

Department of Computer Science
Columbia University

CS@CU COMPUTER SCIENCE

A New Way to Store Metadata

PERFORMANCE

A New Way to Store Metadata

PERFORMANCE

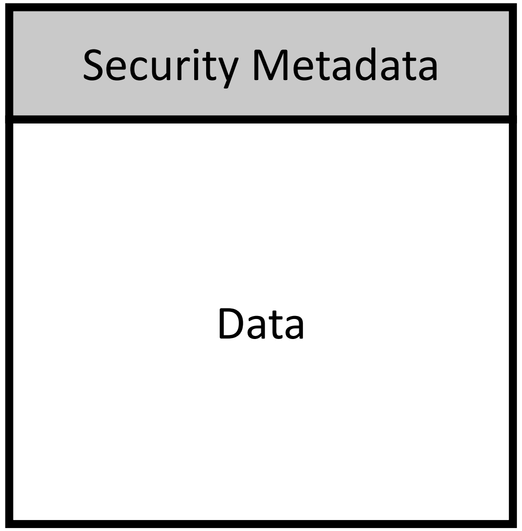
SECURITY
RELIABILITY

A New Way to Store Metadata

- Metadata is widely used in computer architecture

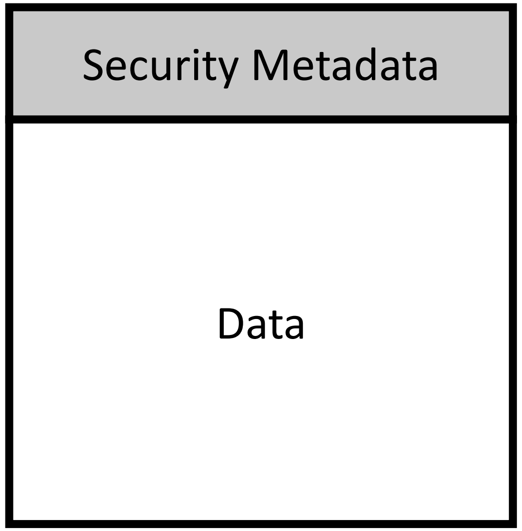
A New Way to Store Metadata

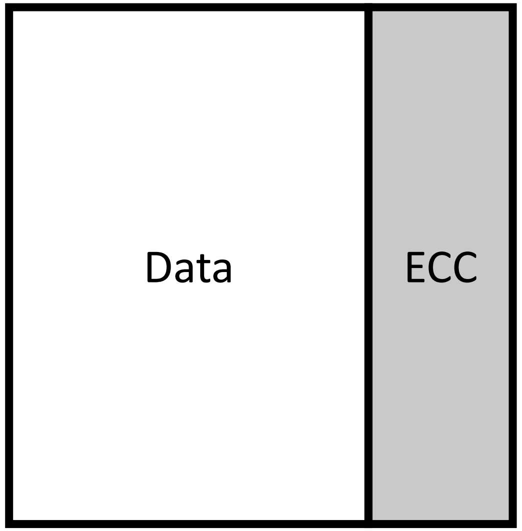
- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead



A New Way to Store Metadata

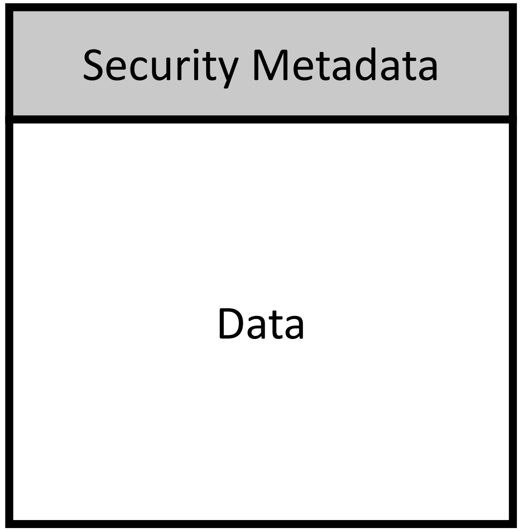
- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead

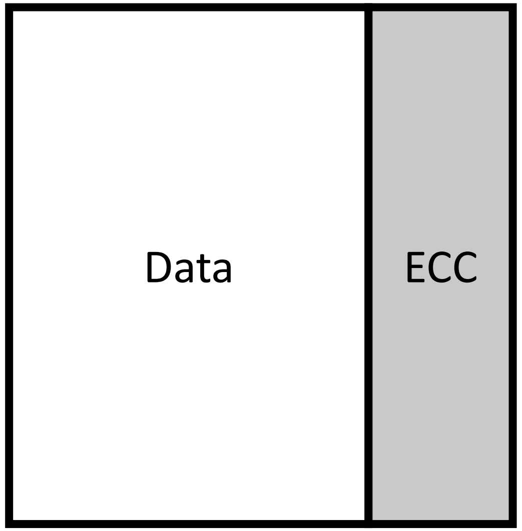




A New Way to Store Metadata

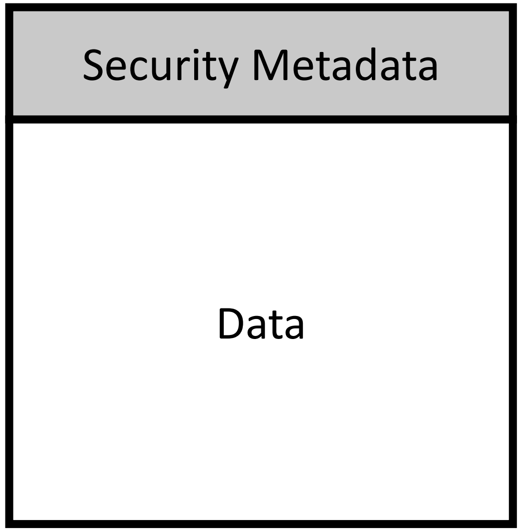
- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost

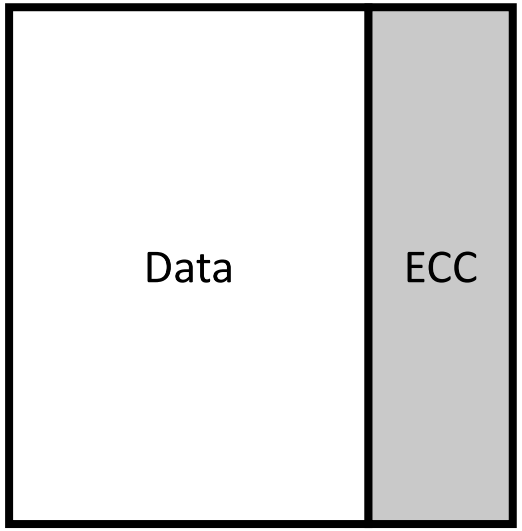




A New Way to Store Metadata

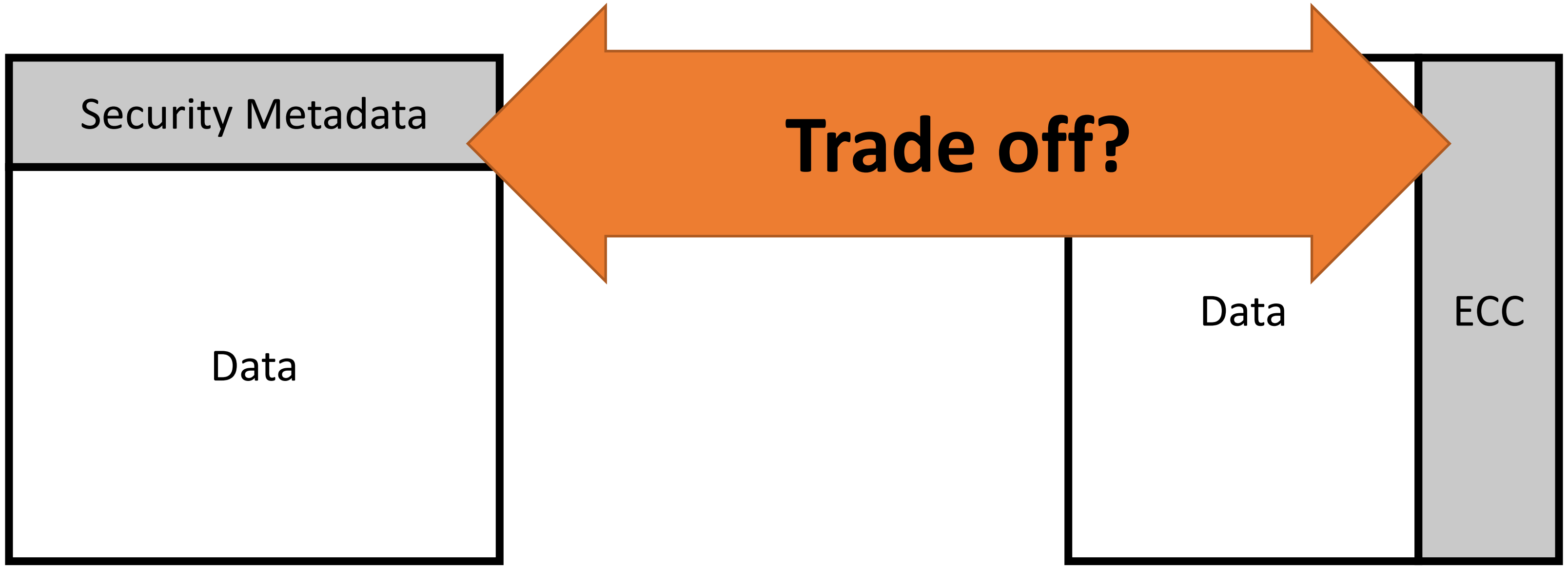
- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques





A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques



A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques
  - Complicated system design to manage metadata

A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques
  - Complicated system design to manage metadata
- Solution:

A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques
  - Complicated system design to manage metadata
- Solution:
  - MUSE (Multi-Use) ECC

A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques
  - Complicated system design to manage metadata
- Solution:
  - MUSE (Multi-Use) ECC w/ 30% less metadata overhead for Chip Kill ECC

A New Way to Store Metadata

- Metadata is widely used in computer architecture
  - Modern hardware security solutions: ~1 to 21% overhead
  - Reliability techniques: e.g., ChipKill ECC w/ 12.5% overhead
- Problem: metadata increases cost
  - Forces architects to choose among multiple important techniques
  - Complicated system design to manage metadata
- Solution:
  - MUSE (Multi-Use) ECC w/ 30% less metadata overhead for Chip Kill ECC
  - Rowhammer defense w/ 40b hash w/o giving up on reliability

Talk Outline

- ChipKill with MUSE ECC

Talk Outline

- ChipKill with MUSE ECC
- Use Cases:
  - Rowhammer defenses
  - PIM Reliability

Talk Outline

- ChipKill with MUSE ECC
- Use Cases:
  - Rowhammer defenses
  - PIM Reliability
- Paper Contents Overview

MUSE ECC

MUSE ECC

$data$

MUSE ECC

$$codeword = data \times m$$

MUSE ECC

$$codeword = data \times m$$

Store To Memory

MUSE ECC

$$codeword = data\times m$$

Store To Memory

Read From Memory

MUSE ECC

$$codeword = data \times m$$

Store To Memory

$$remainder = codeword \bmod m$$

Read From Memory

MUSE ECC

$$codeword = data \times m$$

Store To Memory

$$remainder = codeword \bmod m$$

$$data = \begin{cases} codeword/m & remainder = 0 \\ & \end{cases}$$

Read From Memory

MUSE ECC

$$codeword = data \times m$$

Store To Memory

$$remainder = codeword \bmod m$$

$$data = \begin{cases} codeword/m & remainder = 0 \\ error & else \end{cases}$$

Read From Memory

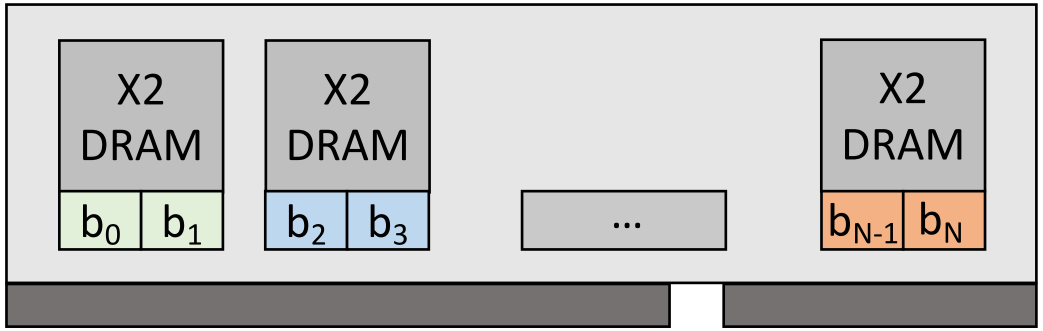
MUSE ECC

$$\#(unique\ remainders) \equiv \#(all\ errors)$$

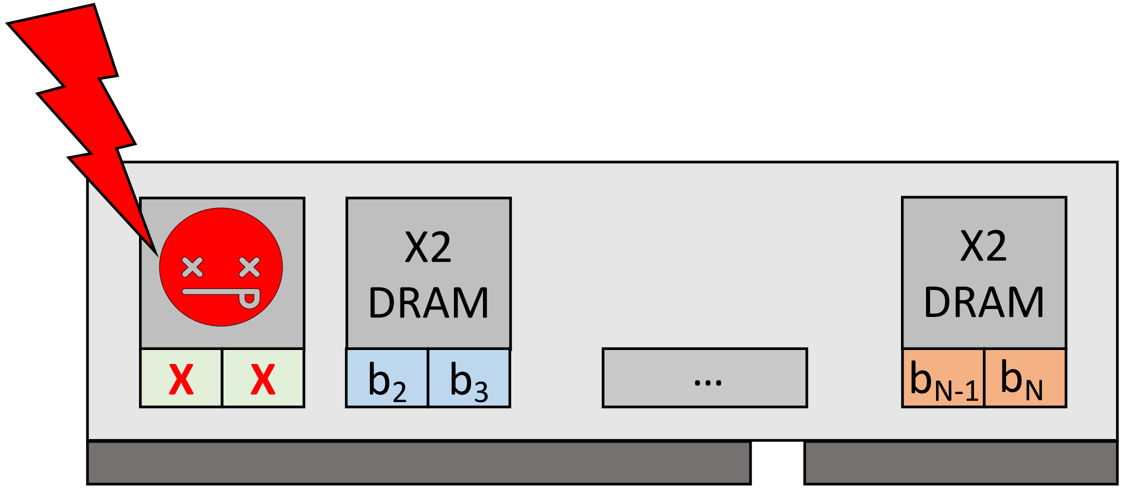
$$Single\ Error\ Correction$$

What is ChipKill?

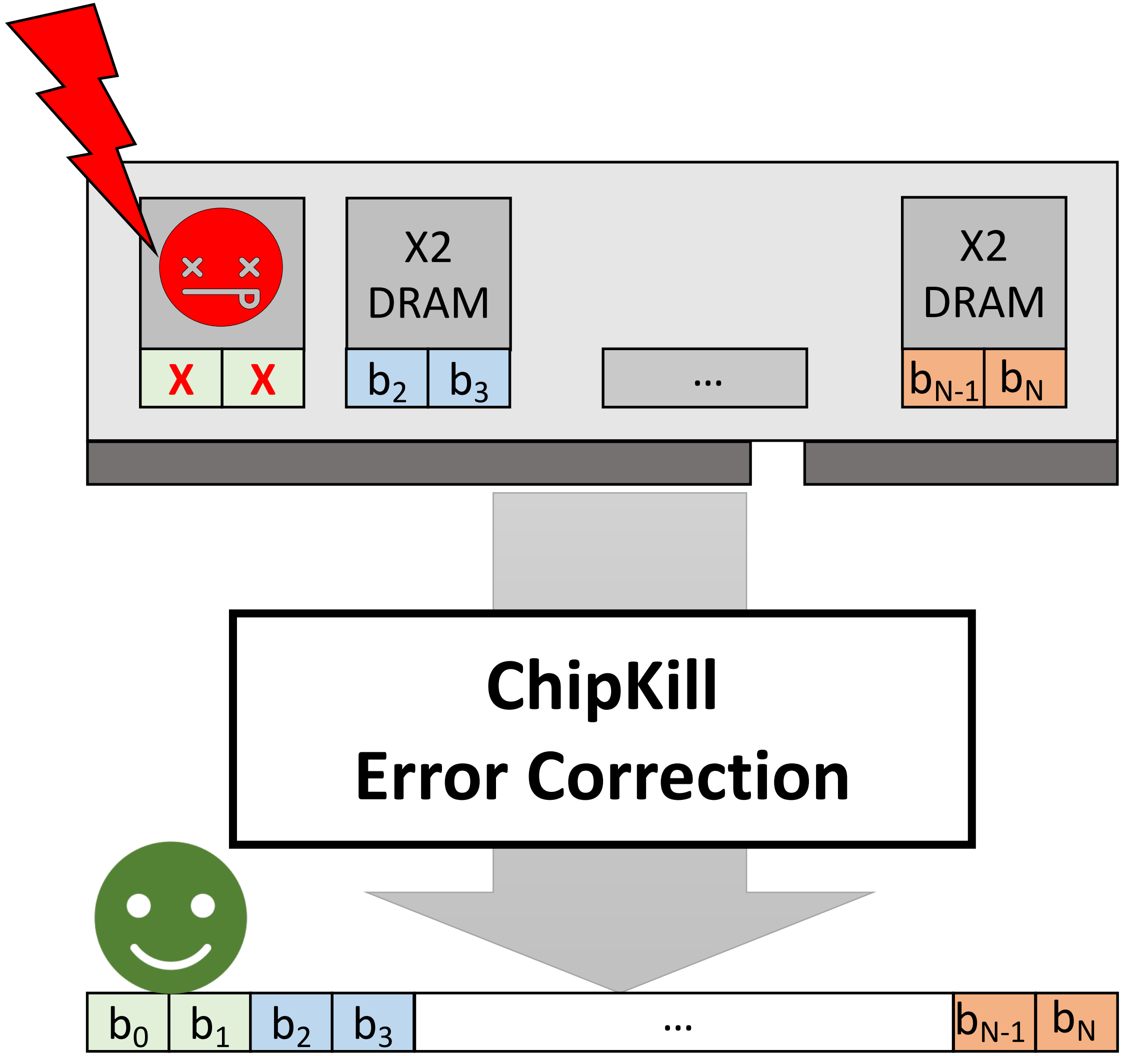
What is ChipKill?



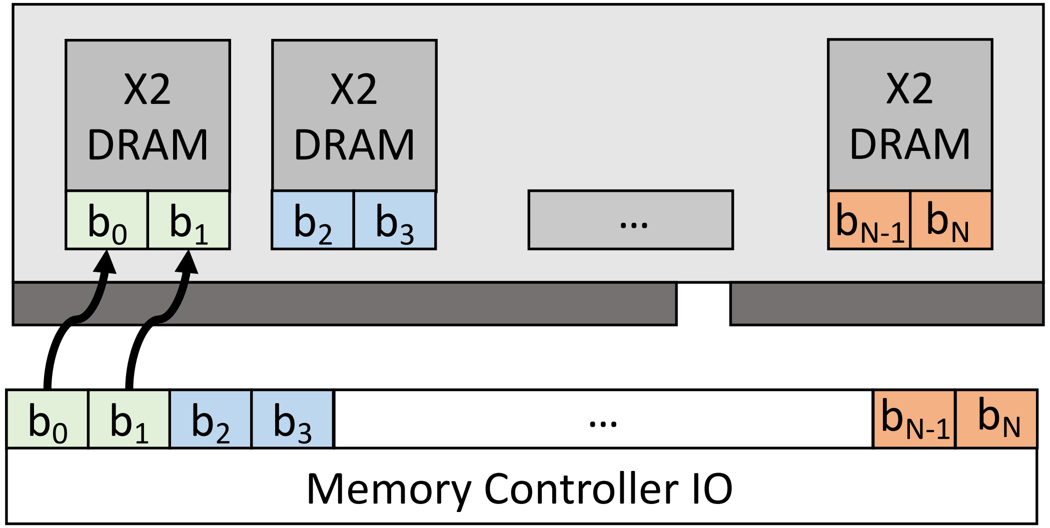
What is ChipKill?



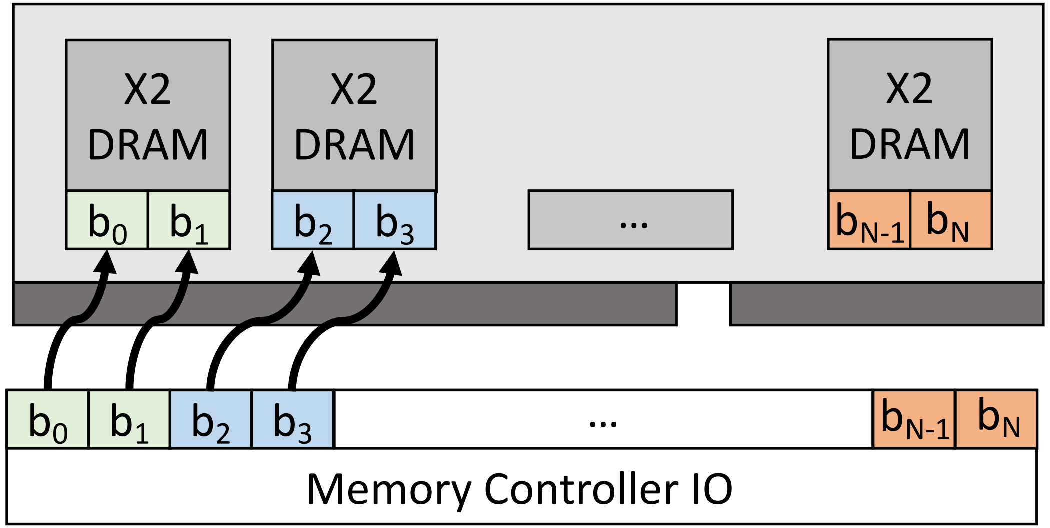
What is ChipKill?



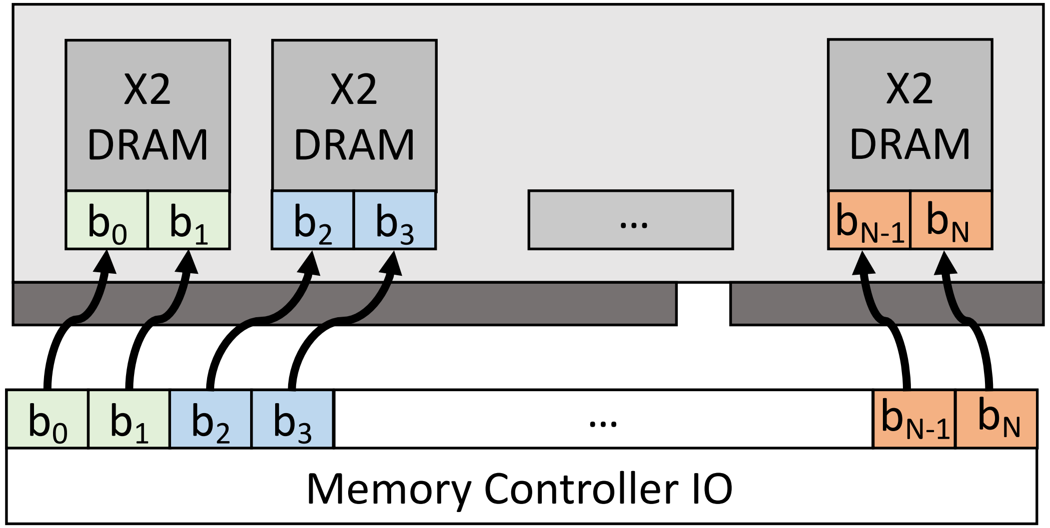
ChipKill with MUSE ECC



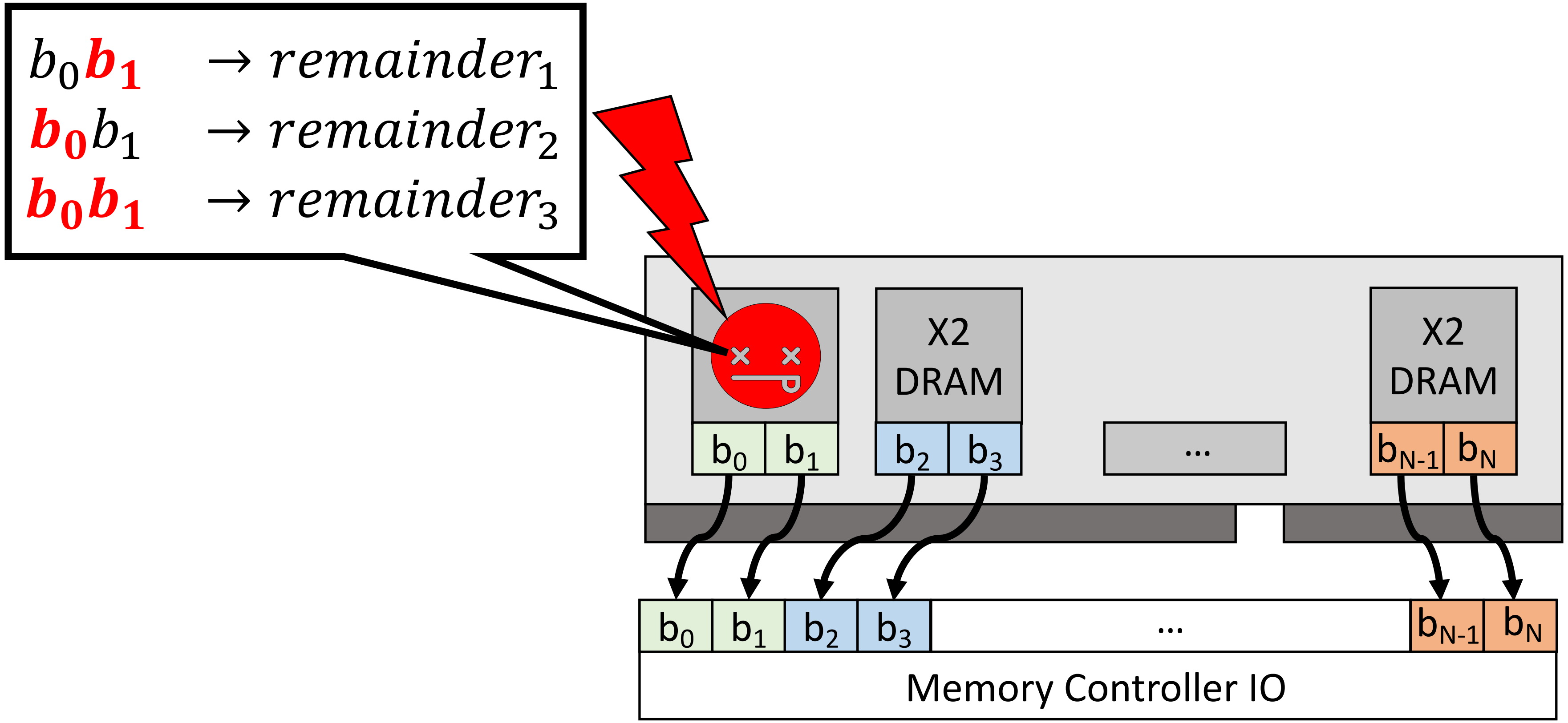
ChipKill with MUSE ECC



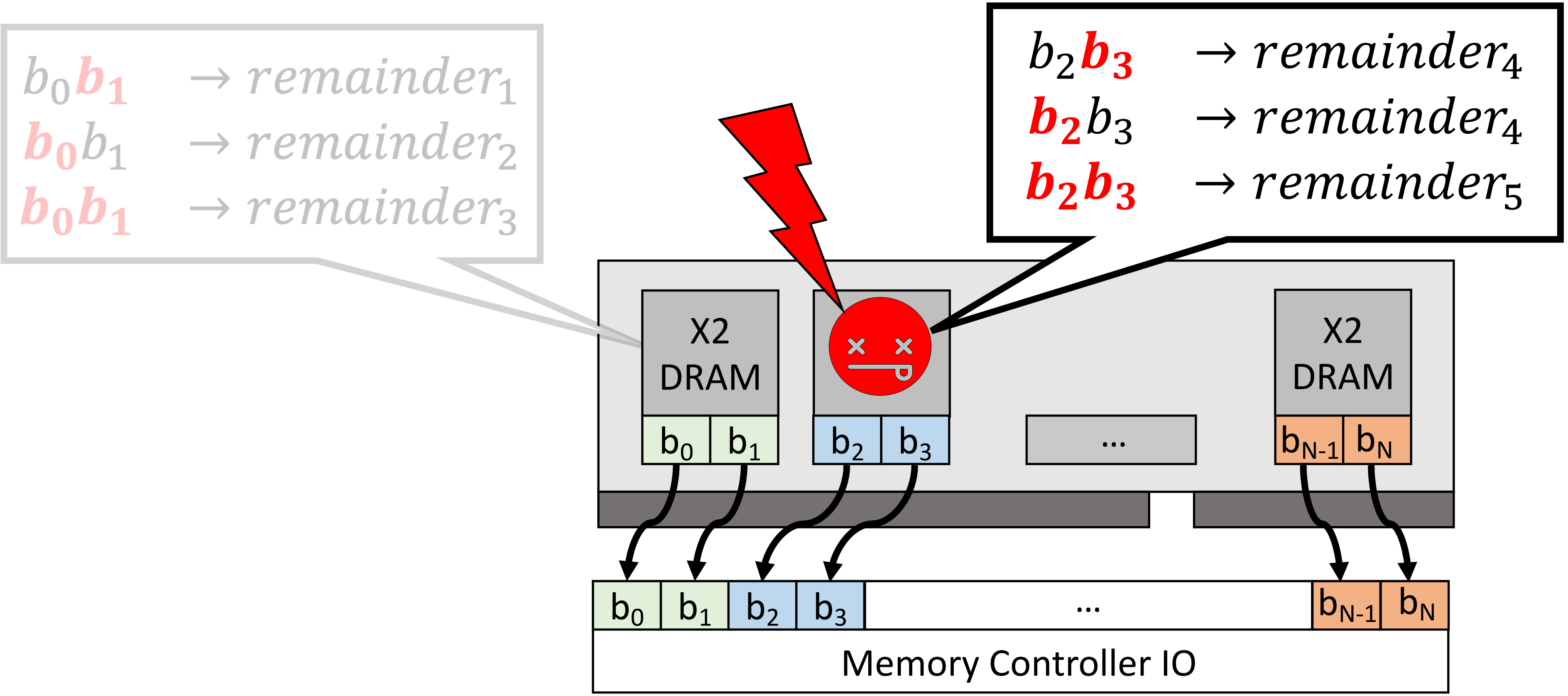
ChipKill with MUSE ECC



ChipKill with MUSE ECC



ChipKill with MUSE ECC



ChipKill with MUSE ECC

$$\#(unique\ remainders) \equiv \#(all\ symbol\ errors)$$

$$ChipKill$$

ChipKill with MUSE ECC

$$\#(unique\ remainders) \equiv \#(all\ symbol\ errors)$$

12b instead of 16b

DDR4 MUSE: 25% fewer ECC bits

MUSE ECC: Shuffling

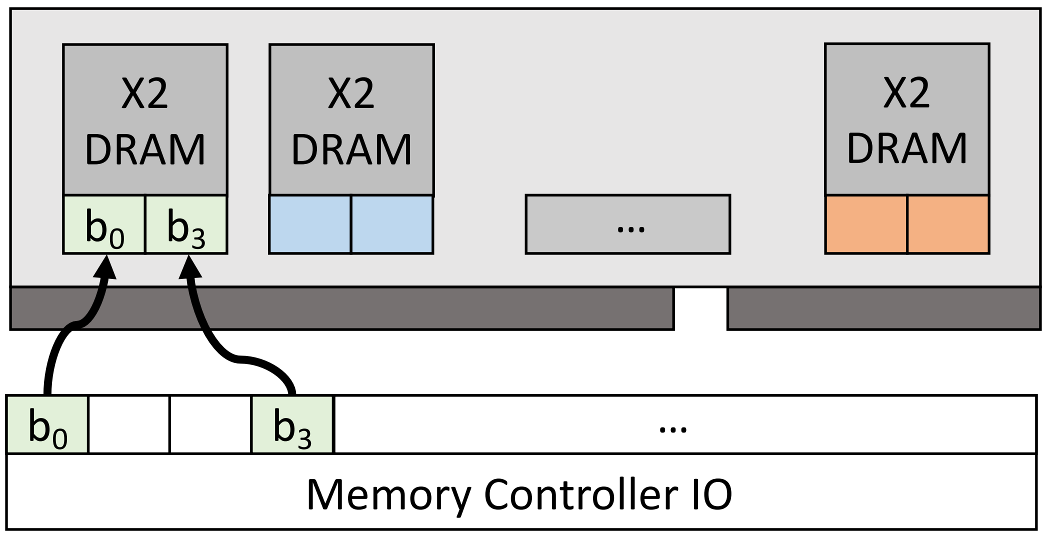
MUSE ECC: Shuffling

$$remainder(\boldsymbol{error_1}) = remainder(\boldsymbol{error_2})$$

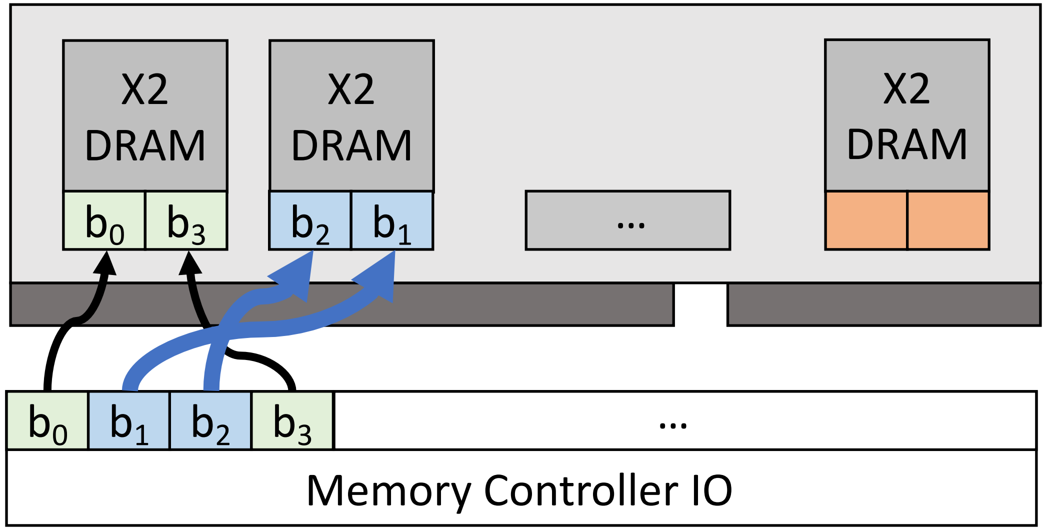
$$\Downarrow$$

$$\cancel{ChipKill}$$

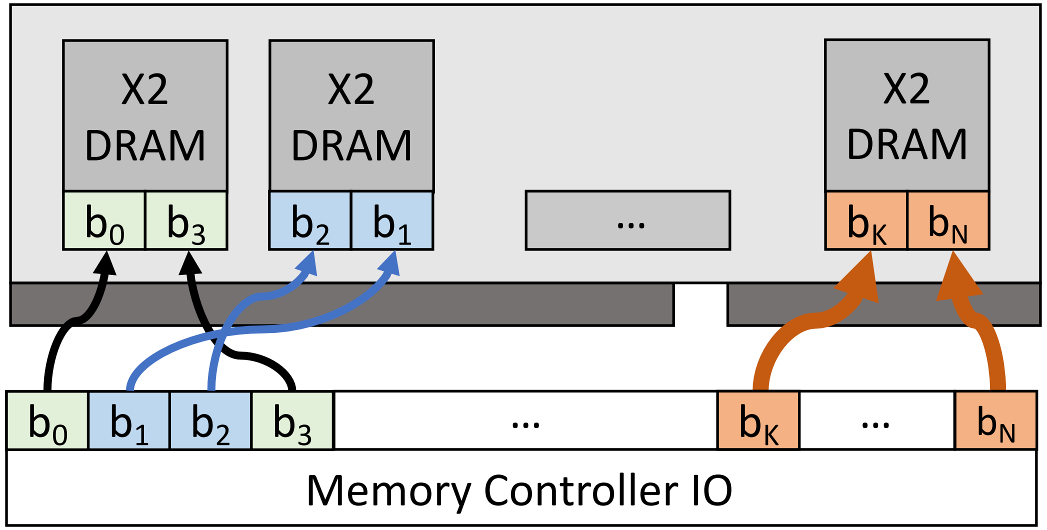
MUSE ECC: Shuffling



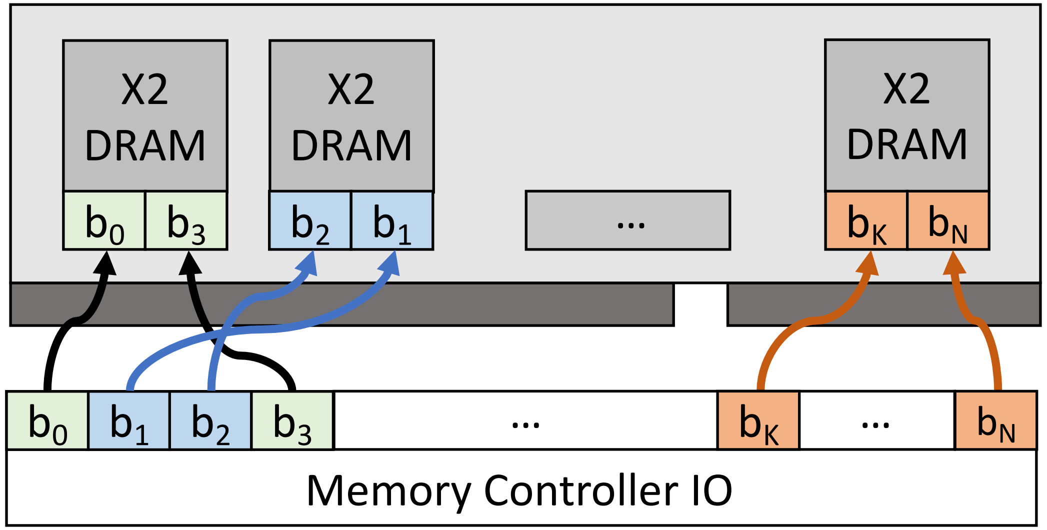
MUSE ECC: Shuffling



MUSE ECC: Shuffling

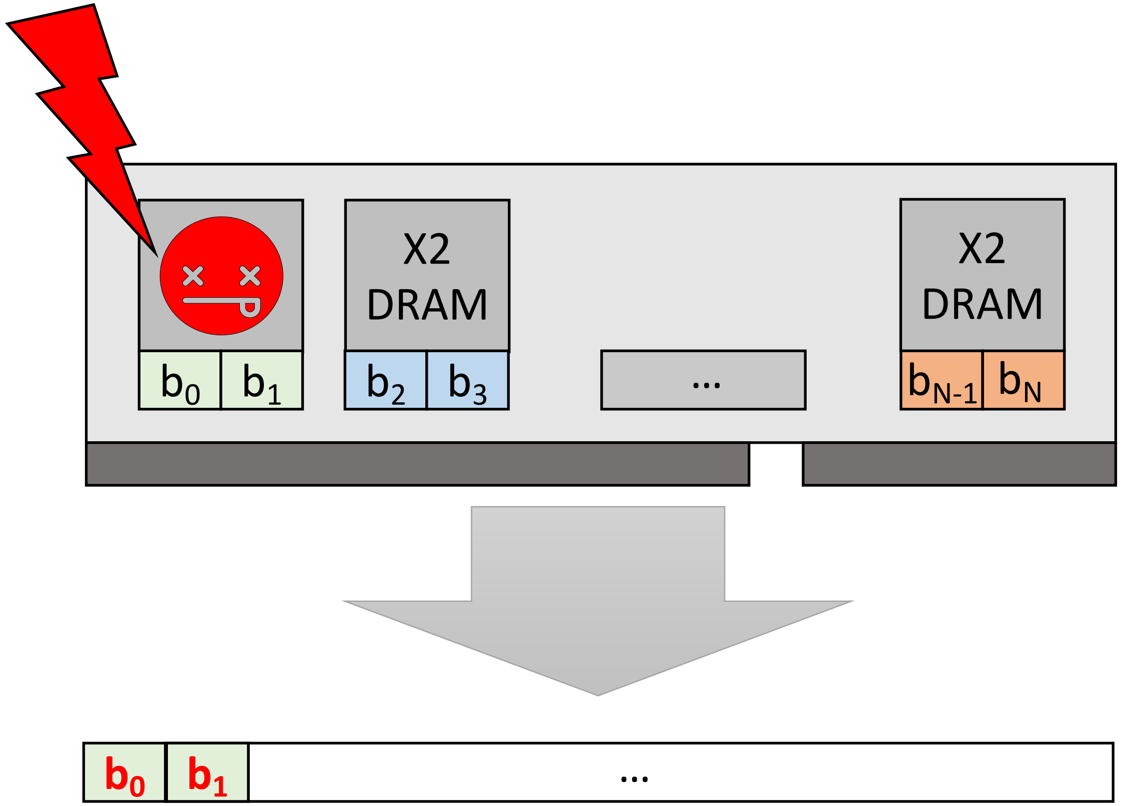


MUSE ECC: Shuffling



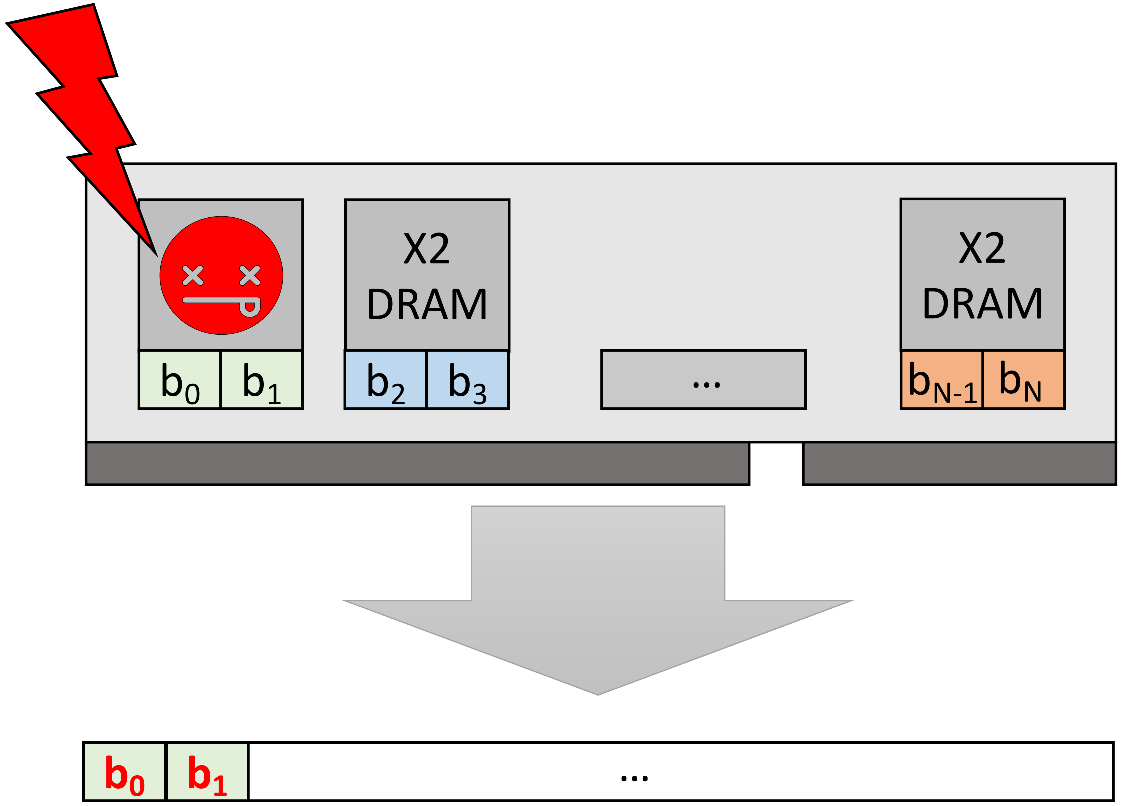
MUSE ECC: Shuffling

MUSE Chip Kill

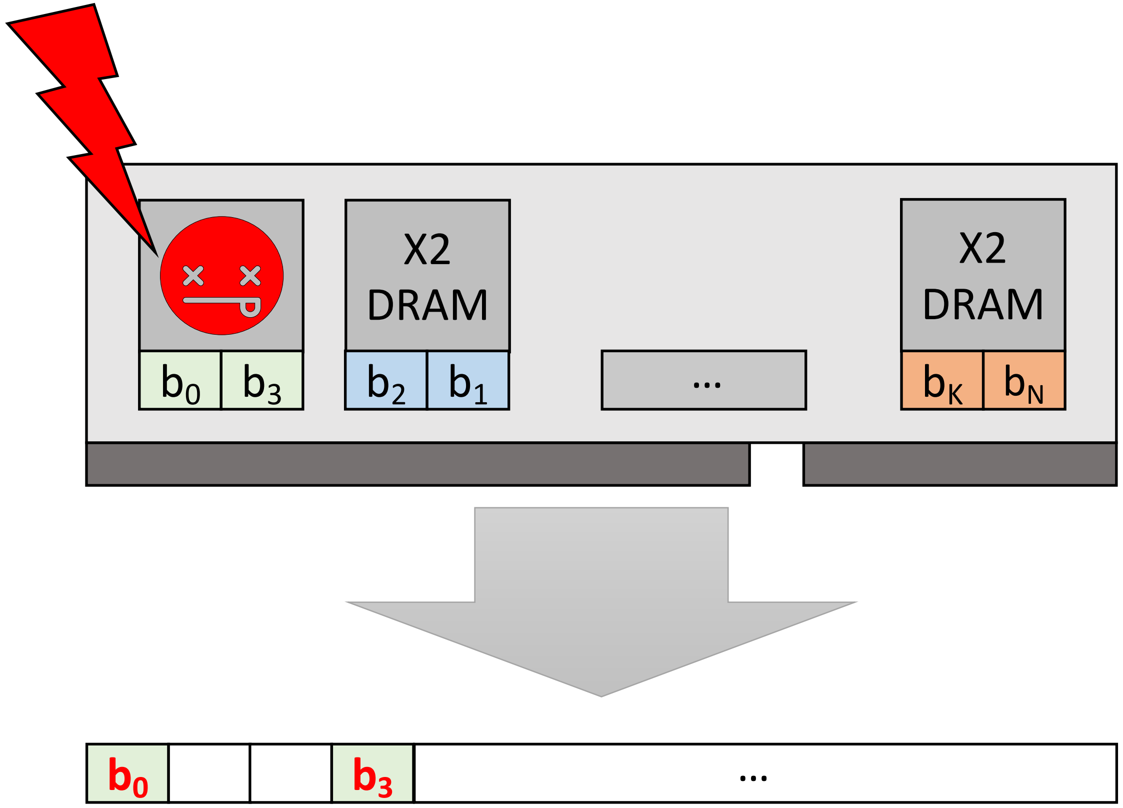


MUSE ECC: Shuffling

MUSE ChipKill



MUSE ChipKill w/ Shuffling



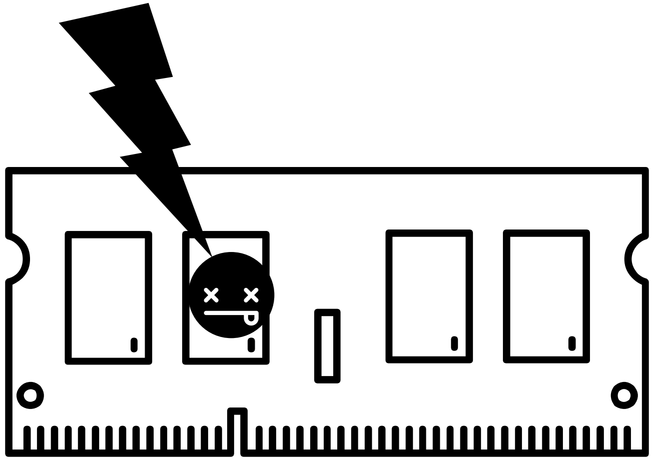
MUSE ECC: Shuffling

$$\#(unique\ remainders) \equiv \#(all\ symbol\ errors)$$

$$ChipKill$$

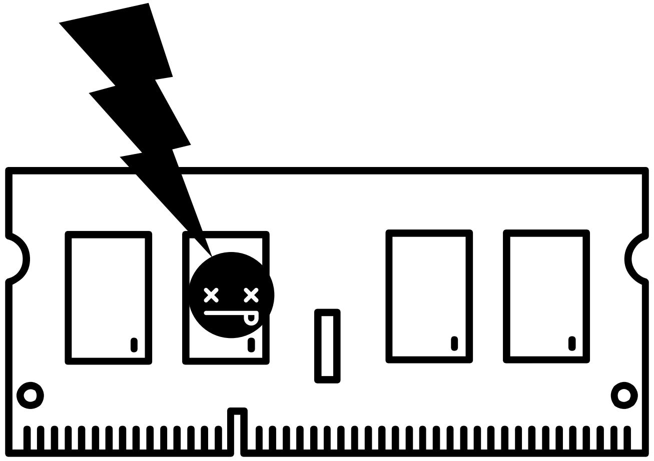
MUSE (Multi-Use) ECC

ChipKill Guarantees

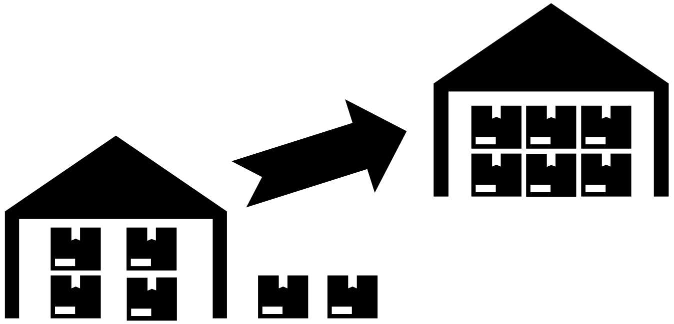


Multi-Use (MUSE) ECC

ChipKill Guarantees

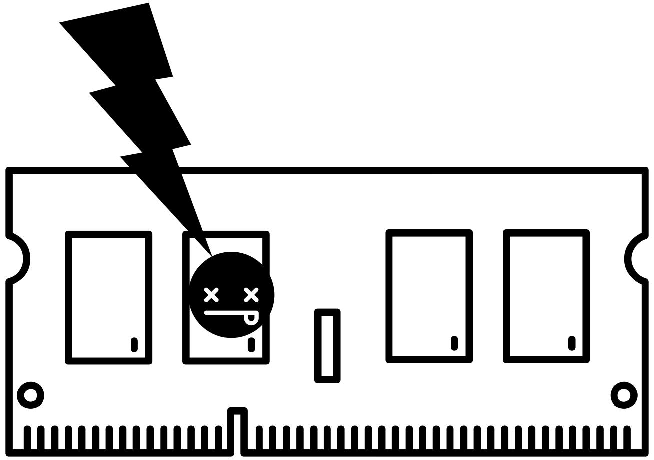


Storage Efficient

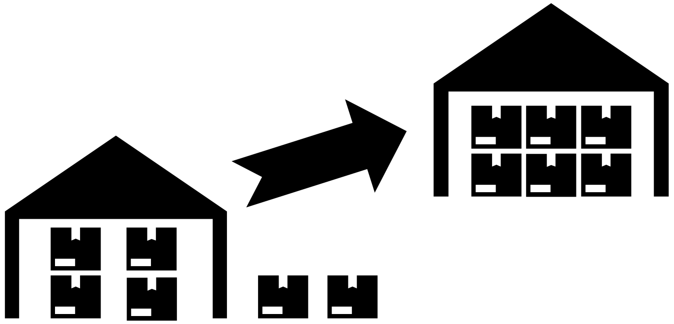


Multi-Use (MUSE) ECC

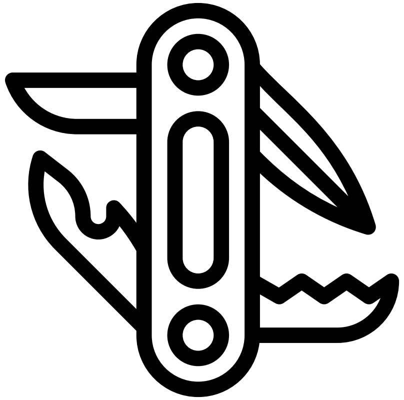
ChipKill Guarantees



Storage Efficient



Flexible

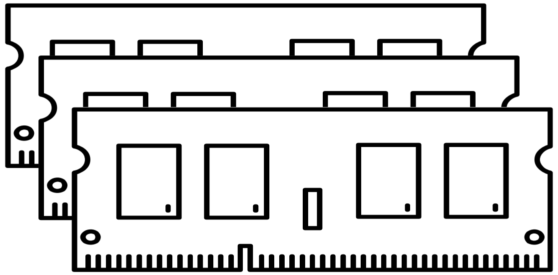


Outline

- ~~ChipKill with MUSE ECC~~
- Use Cases:
  - Rowhammer defenses
  - PIM Reliability
- Paper Contents Overview

Use Cases

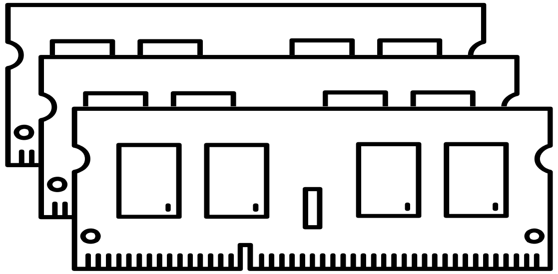
Use Case 1: Rowhammer Defense



DDR5 w/
128B Cache Lines

80b channel

Use Case 1: Rowhammer Defense



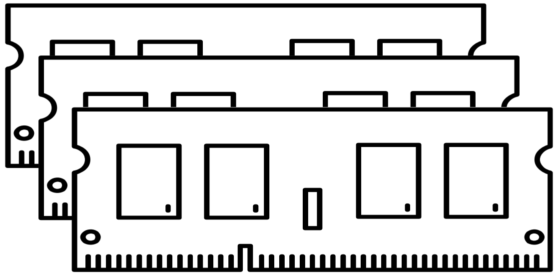
**DDR5 w/
128B Cache Lines**

+

MUSE(80,69) ECC

80b codeword with 64b data and 5b metadata

Use Case 1: Rowhammer Defense

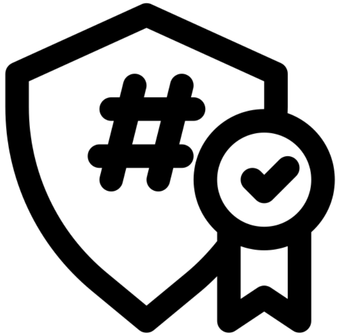


DDR5 w/
128B Cache Lines

+

MUSE(80,69) ECC

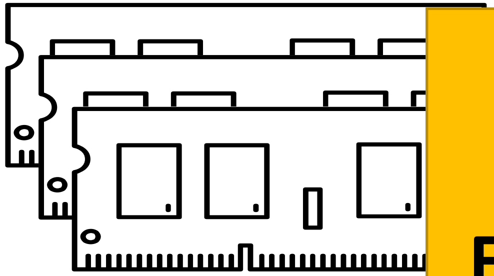
=



40b hash for
64B data

40b aggregated across 8 DRAM transactions

Use Case 1: Rowhammer Defense

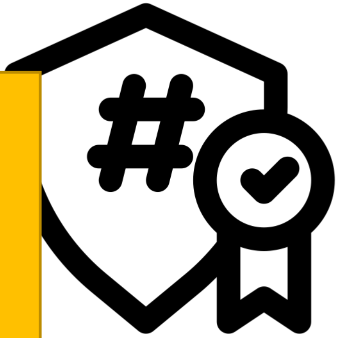


DDR5 w/

128B Cache Lines

Chance of successful attack is 2^{-40}

Faking hash value will take a lot of time [1]



b hash for

64B data

[1]: Exploiting correcting codes: On the effectiveness of ECC memory against rowhammer attacks.
Cojocar, L., Razavi, K., Giuffrida, C. and Bos, H., In 2019 IEEE Symposium on Security and Privacy.

Use Case 2: Processing-In-Memory

PIM with MUSE:

Use Case 2: Processing-In-Memory

PIM with MUSE:

✓**Single code for both storage and arithmetic reliability**

Use Case 2: Processing-In-Memory

PIM with MUSE:

✓**Single code for both storage and arithmetic reliability**

✓**ECC check is done in parallel to compute**

Use Case 2: Processing-In-Memory

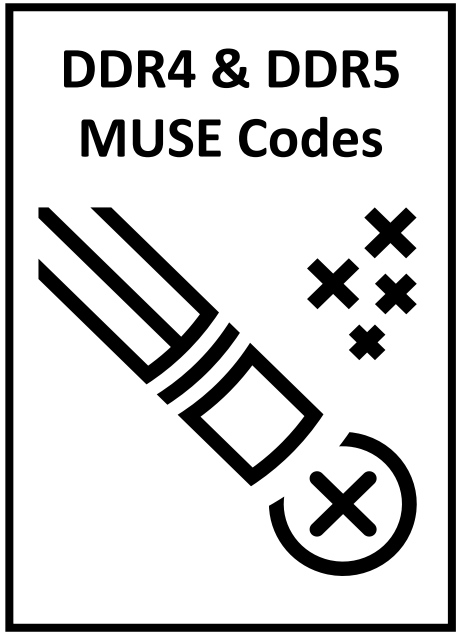
PIM with MUSE:

- ✓ **Single code for both storage and arithmetic reliability**
- ✓ **ECC check is done in parallel to compute**
- ✓ **Storage efficient: 256b data needs 12b ECC (out of 32b)**

Outline

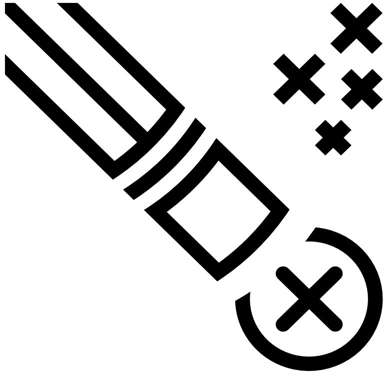
- ~~ChipKill with MUSE ECC~~
- ~~Use Cases:~~
  - ~~Rowhammer defenses~~
  - ~~PIM Reliability~~
- Paper Contents Overview

In the paper

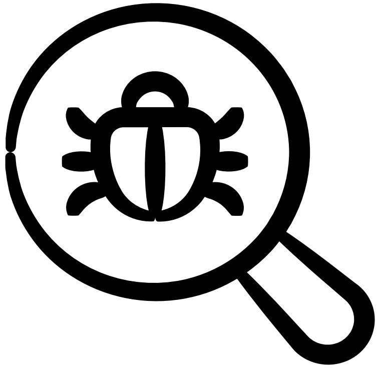


In the paper

DDR4 & DDR5 MUSE Codes

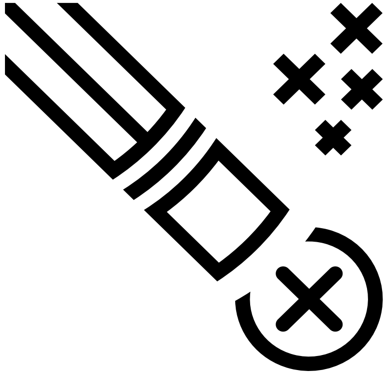


More Case Studies

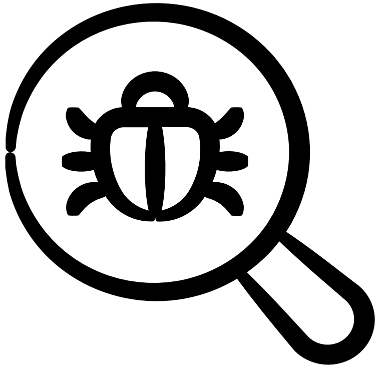


In the paper

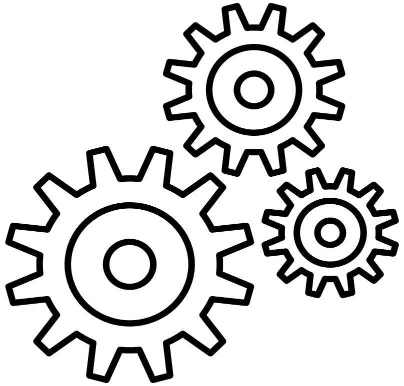
DDR4 & DDR5 MUSE Codes



More Case Studies

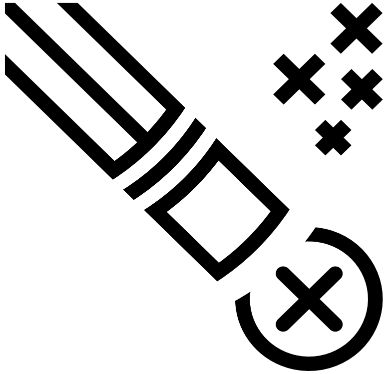


uArch details

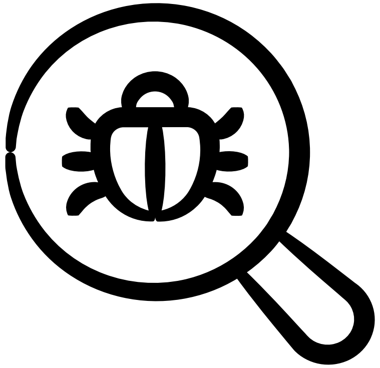


In the paper

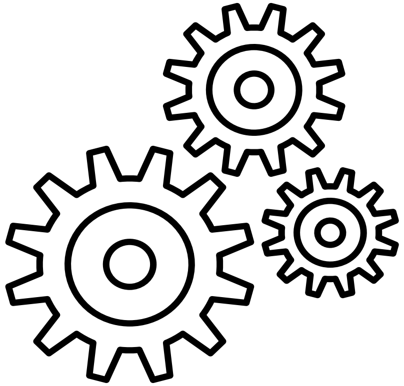
DDR4 & DDR5 MUSE Codes



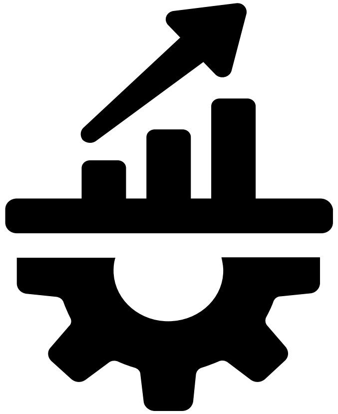
More Case Studies



uArch details



Performance Evaluation



Conclusion

MUSE ECC is the only ECC scheme that:

- Provides ChipKill with only **9.3%** storage overhead
- Offers in-lined metadata storage for any purpose
- Drop-in replacement for existing ECC schemes

Backup slides

Background: Residue Codes

$$codeword\text{: } 1101111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

Background: Residue Codes

LSB MSB

$$codeword': 11{\color{red}1}1111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

Background: Residue Codes

$$codeword': 11\mathbf{\color{red}1}1111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

$$codeword' = codeword + \mathbf{\color{red}2^2}$$

decoding

Background: Residue Codes

$$codeword': 111\mathbf{1}1111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

$$remainder = (codeword + \mathbf{2^2}) \bmod m$$

decoding

Background: Residue Codes

$$codeword': 111\mathbf{1}1111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

$$remainder = (codeword + \mathbf{2^2}) \bmod m = \mathbf{2^2} \bmod m$$

decoding

Background: Residue Codes

$$codeword': 11\mathbf{1}1111 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 11001010$$

$$remainder = \left(codeword + \mathbf{2^2}\right) \mathrm{mod}\ m = \mathbf{2^2} \mathrm{mod}\ m$$

$$remainder \neq 0 \Longrightarrow data = \frac{codeword - f_{err}\left(\mathbf{2^2} mod\ m\right)}{m}$$

decoding

Background: Linearity of Residue Codes

$$(x \ \mathbf{OP} \ y) \bmod M = (x \bmod M \ \mathbf{OP} \ y \bmod M) \bmod M$$

e.g., $(x + y) \bmod M = (x \bmod M + y \bmod M) \bmod M$