Efficient Coding Schemes for Flash Memories

Eitan Yaakobi, Laura Grupp

Steven Swanson, Paul H. Siegel, and Jack K. Wolf

UCSD

University of California San Diego

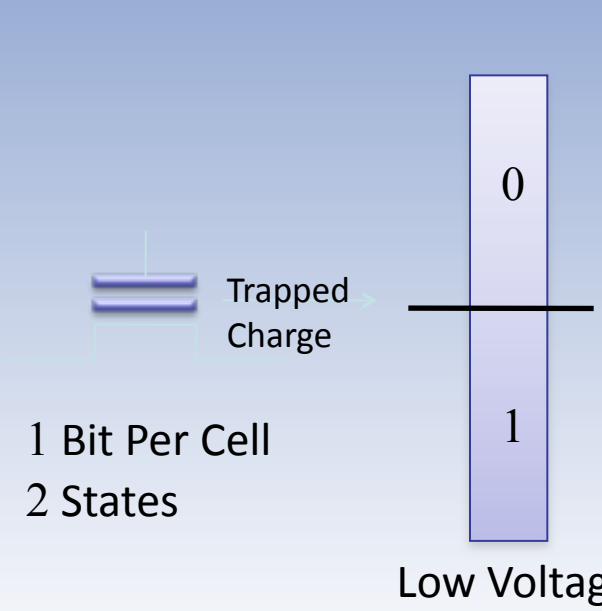
Flash Memory Summit, August 2010

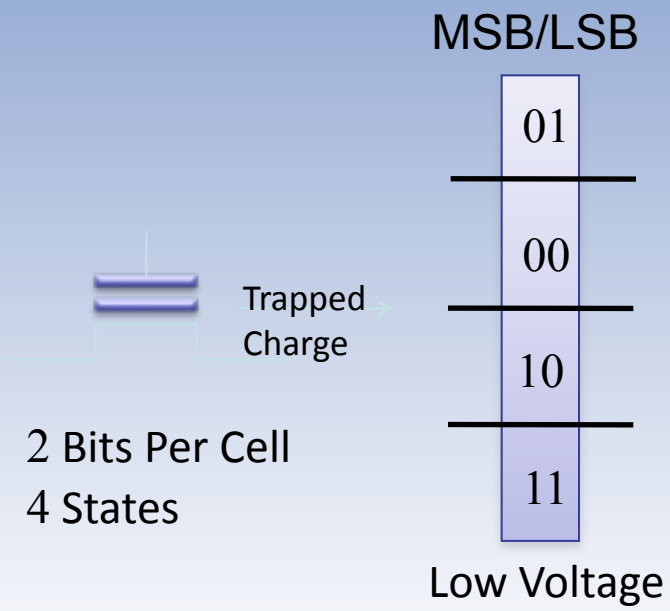
Outline

- Flash Memory Structure
- Single Bit Representation in MLC Flash
- New ECC Scheme for MLC Flash
- WOM-Codes

SLC Vs. MLC Flash

- In SLC flash, each cell stores a single bit
- In MLC, each cell can store multiple bits (typically 2 bits)





Flash Memory Structure

Flash Memory Structure

- A group of cells consist of a page

Flash Memory Structure

- A group of cells consist of a page
- A group of pages consist of a block

Flash Memory Structure

- A group of cells consist of a page
- A group of pages consist of a block
  - In SLC flash, a typical block layout is as follows

Flash Memory Structure

- A group of cells consist of a page
- A group of pages consist of a block
  - In SLC flash, a typical block layout is as follows

page 0	page 1
page 2	page 3
page 4	page 5
. . .	. . .
page 62	page 63

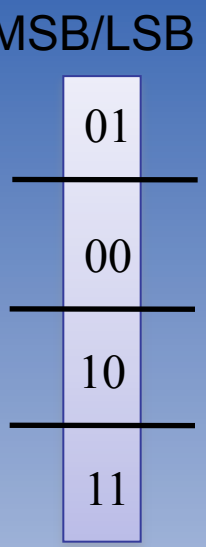
Flash Memory Structure

Flash Memory Structure

- In MLC flash the two bits within a cell **DO NOT** belong to the same page – **MSB page** and **LSB page**

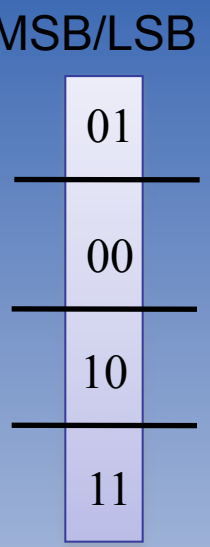
Flash Memory Structure

- In MLC flash the two bits within a cell **DO NOT** belong to the same page – **MSB page** and **LSB page**

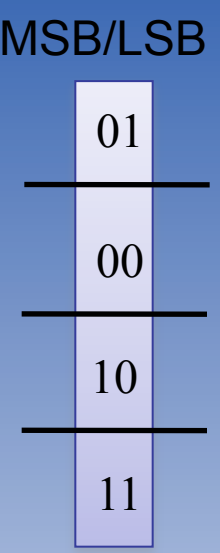


Flash Memory Structure

- In MLC flash the two bits within a cell **DO NOT** belong to the same page – **MSB page** and **LSB page**
- Given a group of cells, all the MSB's consist of one page and all the LSB's consist of another page



Flash Memory Structure



- In MLC flash the two bits within a cell **DO NOT** belong to the same page – **MSB page** and **LSB page**
- Given a group of cells, all the MSB's consist of one page and all the LSB's consist of another page

Row index	**MSB** of **first** 2^{14} cells	**LSB** of **first** 2^{14} cells	**MSB** of **last** 2^{14} cells	**LSB** of **last** 2^{14} cells
1	page 0	page 4	page 1	page 5
2	page 2	page 8	page 3	page 9
3	page 6	page 12	page 7	page 13
4	page 10	page 16	page 11	page 17
.	.	.	.	.
31	page 118	page 124	page 119	page 125
32	page 122	page 126	page 123	page 127

Experiment Description

Experiment Description

- We checked several flash memory **MLC** blocks

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**
  - A pseudo-random data is **written** to the block

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**
  - A pseudo-random data is **written** to the block
  - The data is **read** and **compared** to find errors

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**
  - A pseudo-random data is **written** to the block
  - The data is **read** and **compared** to find errors
- **Remarks**:

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**
  - A pseudo-random data is **written** to the block
  - The data is **read** and **compared** to find errors
- **Remarks**:
  - We measured many more iterations than the manufacturer's guaranteed number of erasures

Experiment Description

- We checked several flash memory **MLC** blocks
- For each block the following steps are repeated
  - The block is **erased**
  - A pseudo-random data is **written** to the block
  - The data is **read** and **compared** to find errors
- **Remarks**:
  - We measured many more iterations than the manufacturer's guaranteed number of erasures
  - The experiment was done in a lab conditions and related factors such as temperature change, intervals between erasures or multiple readings before erasures were not considered

Single Bit Representation in MLC Flash

Single Bit Representation in MLC Flash

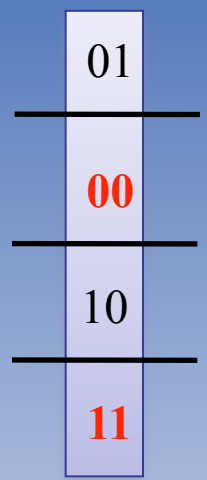
- How to store a single bit in MLC flash?

Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:

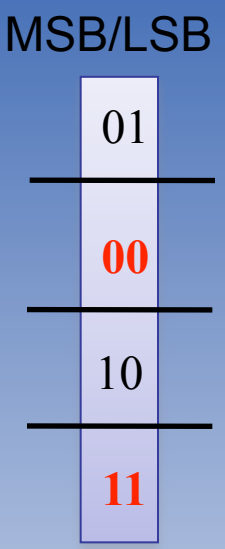
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:



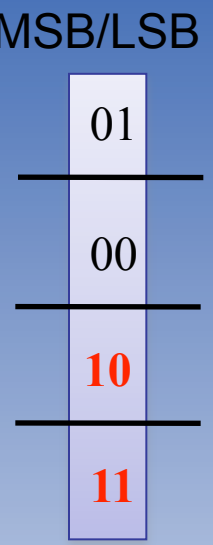
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**



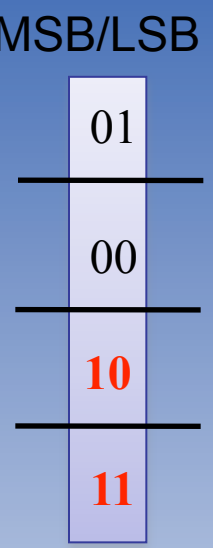
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**



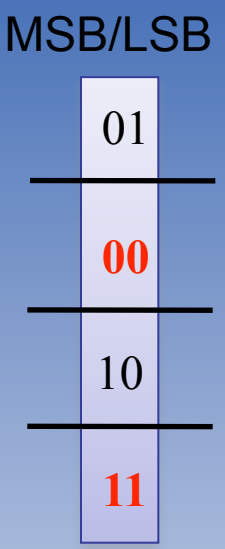
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**
  - Program only the **LSB pages**



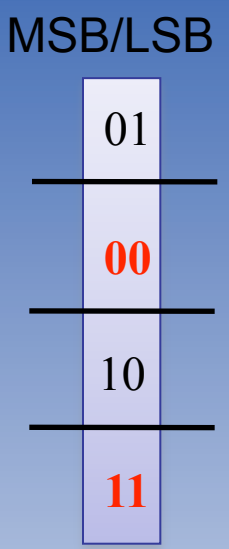
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**
  - Program only the **LSB pages**



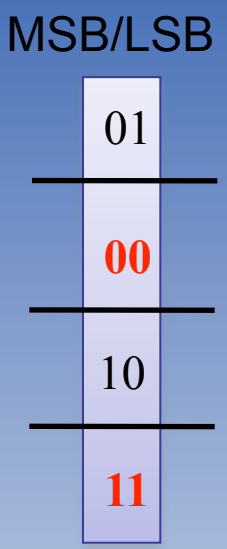
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**
  - Program only the **LSB pages**
  - Program the LSB and MSB pages **with the same values** (cells can be in state **11** or **00**)



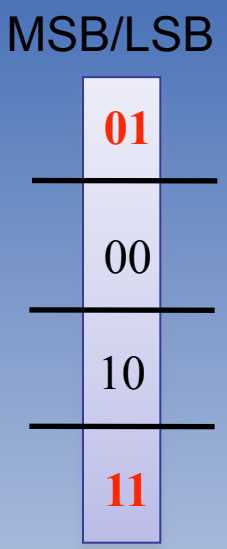
Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**
  - Program only the **LSB pages**
  - Program the LSB and MSB pages **with the same values** (cells can be in state **11** or **00**)
  - Program the data in the MSB pages, and program all LSB pages to **all-1** bit values (cells can be in state **00** or **01**)

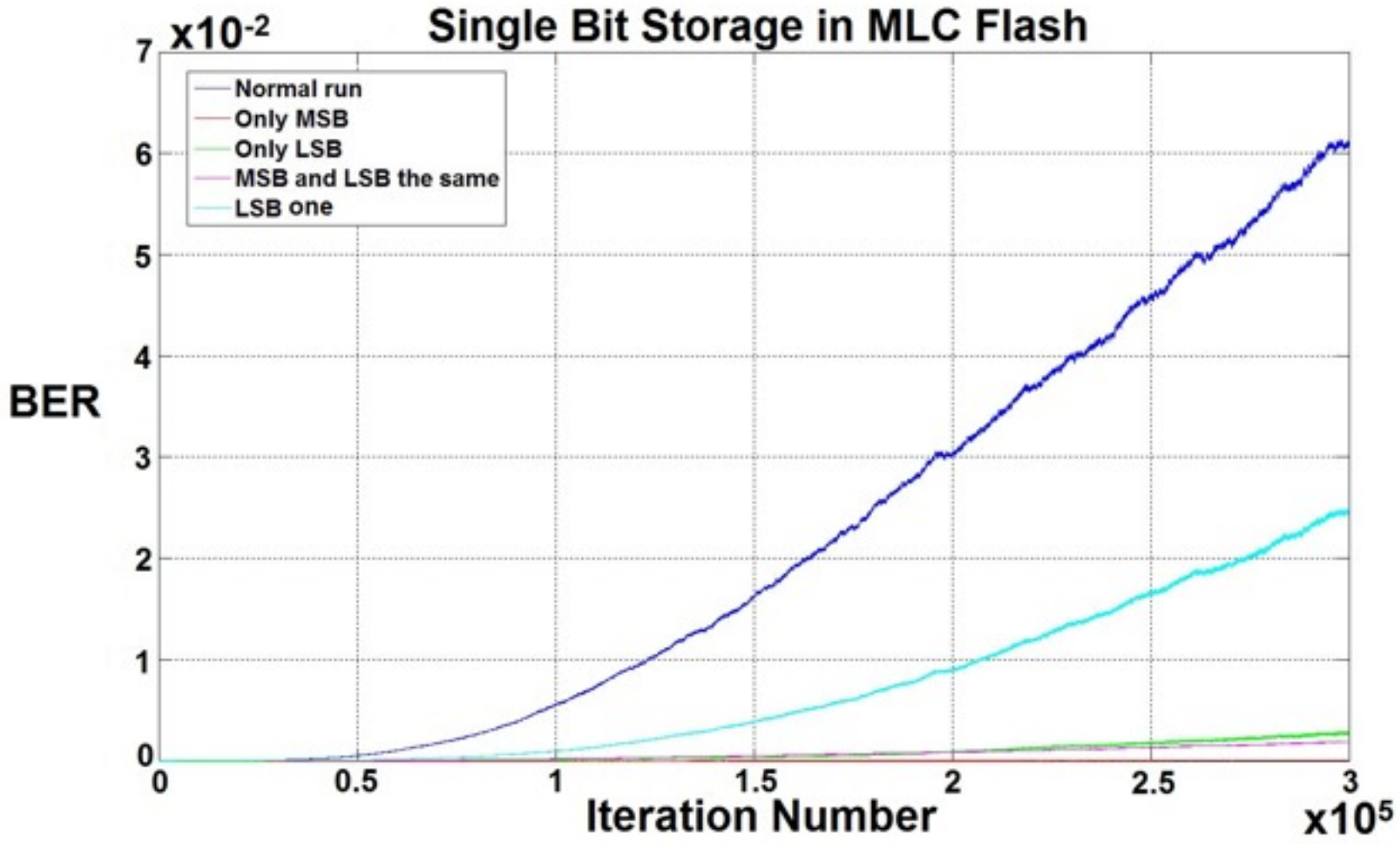


Single Bit Representation in MLC Flash

- How to store a single bit in MLC flash?
- There are several ways:
  - Program only the **MSB pages**
  - Program only the **LSB pages**
  - Program the LSB and MSB pages **with the same values** (cells can be in state **11** or **00**)
  - Program the data in the MSB pages, and program all LSB pages to **all-1** bit values (cells can be in state **00** or **01**)



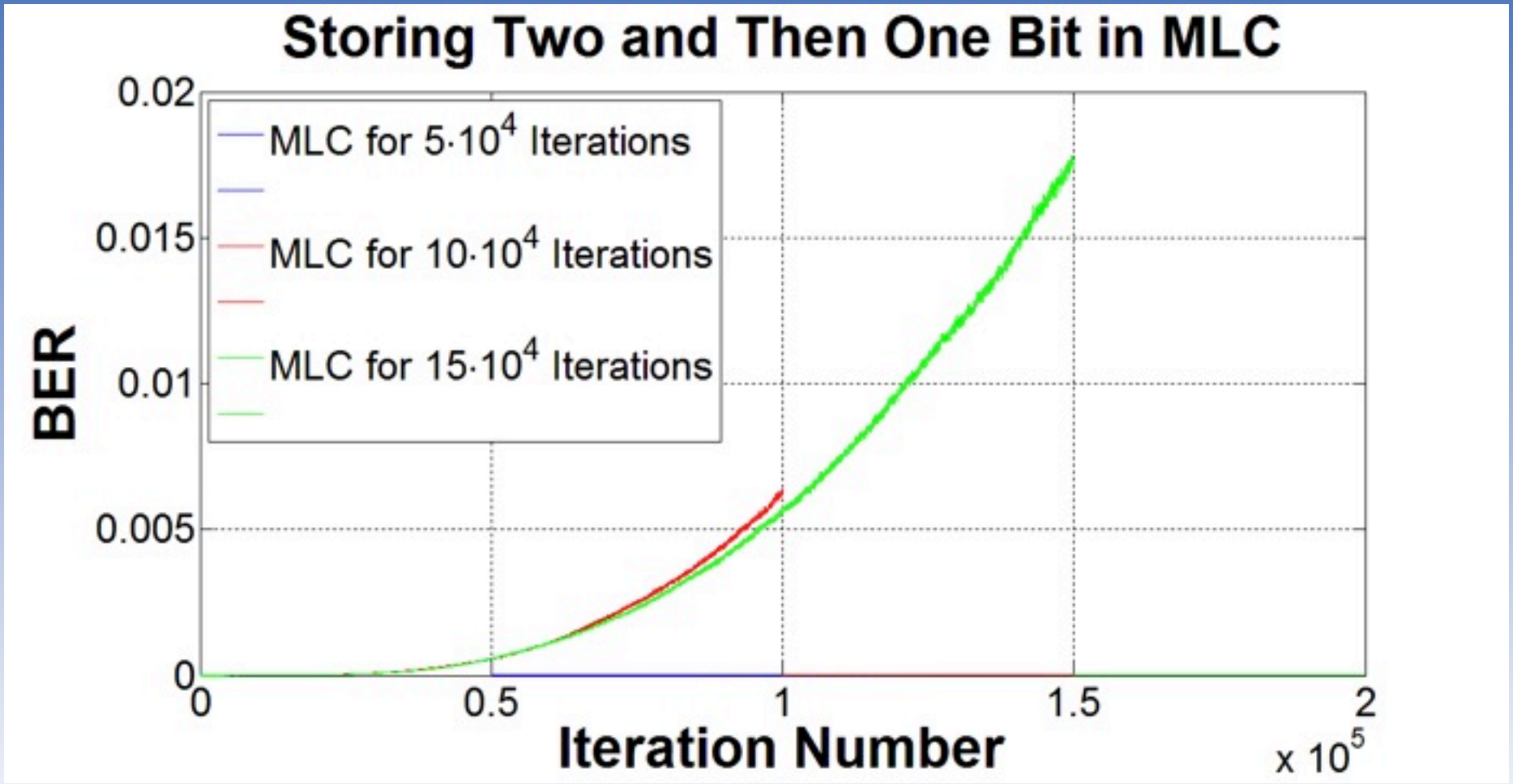
Single Bit Representation in MLC Flash



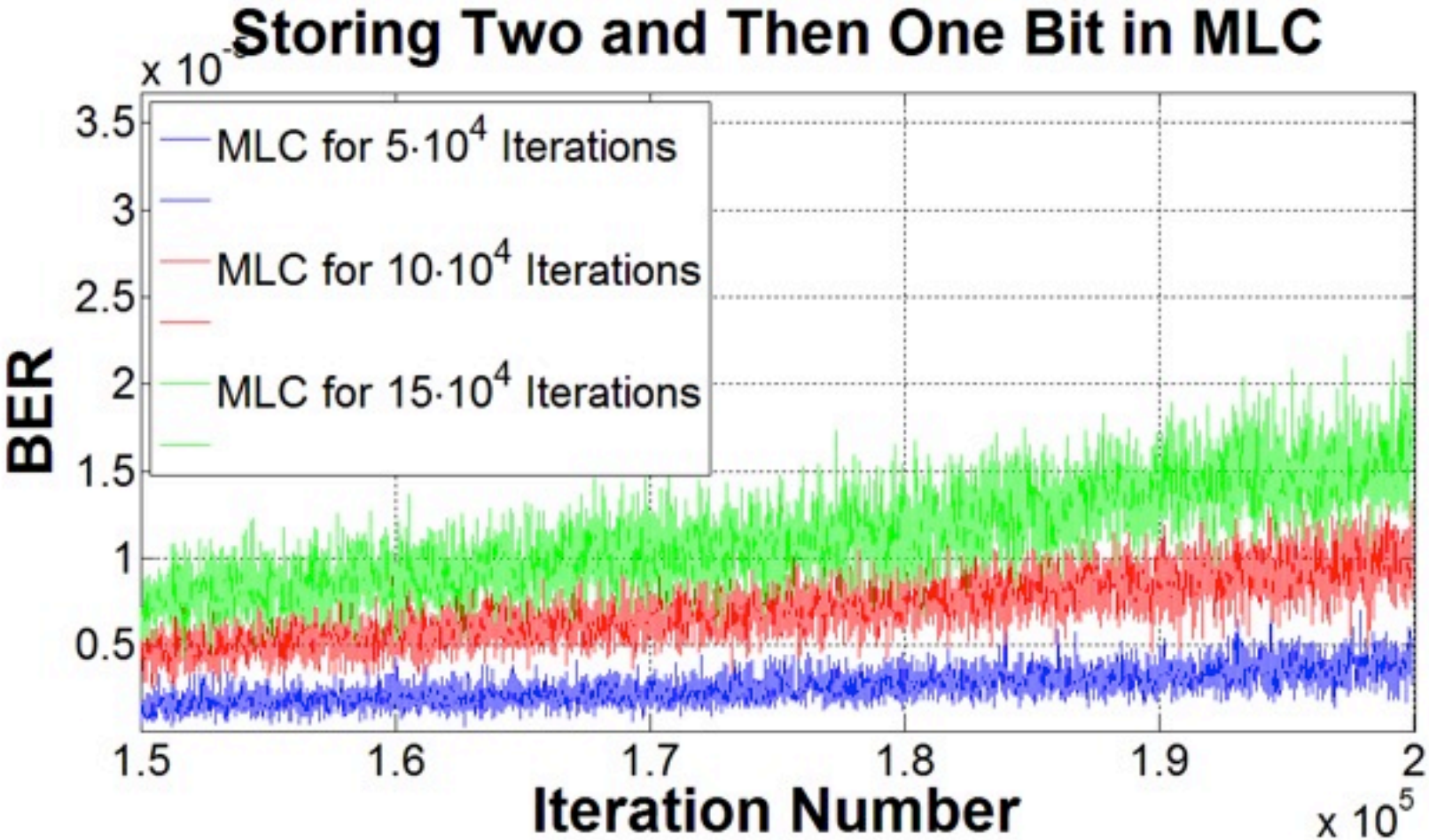
Single Bit Representation in MLC Flash

- What happens when the chip is first used as an **MLC** and then switched to be used as an **SLC**?
- We ran the following experiments:
  - Use the chip for **50,000** iterations as an **MLC** and **150,000** iterations as an **SLC**
  - Use the chip for **100,000** iterations as an **MLC** and **100,000** iterations as an **SLC**
  - Use the chip for **150,000** iterations as an **MLC** and **50,000** iterations as an **SLC**

Single Bit Representation in MLC Flash



Single Bit Representation in MLC Flash



ECC scheme for MLC flash

ECC scheme for MLC flash

- A common ECC in flash today is a **BCH code**

ECC scheme for MLC flash

- A common ECC in flash today is a **BCH code**
- Errors are corrected in each page **independently**

ECC scheme for MLC flash

- A common ECC in flash today is a **BCH code**
- Errors are corrected in each page **independently**
- In particular, in a pair of SLC and MLC pages sharing the same group of cells, errors are still corrected independently

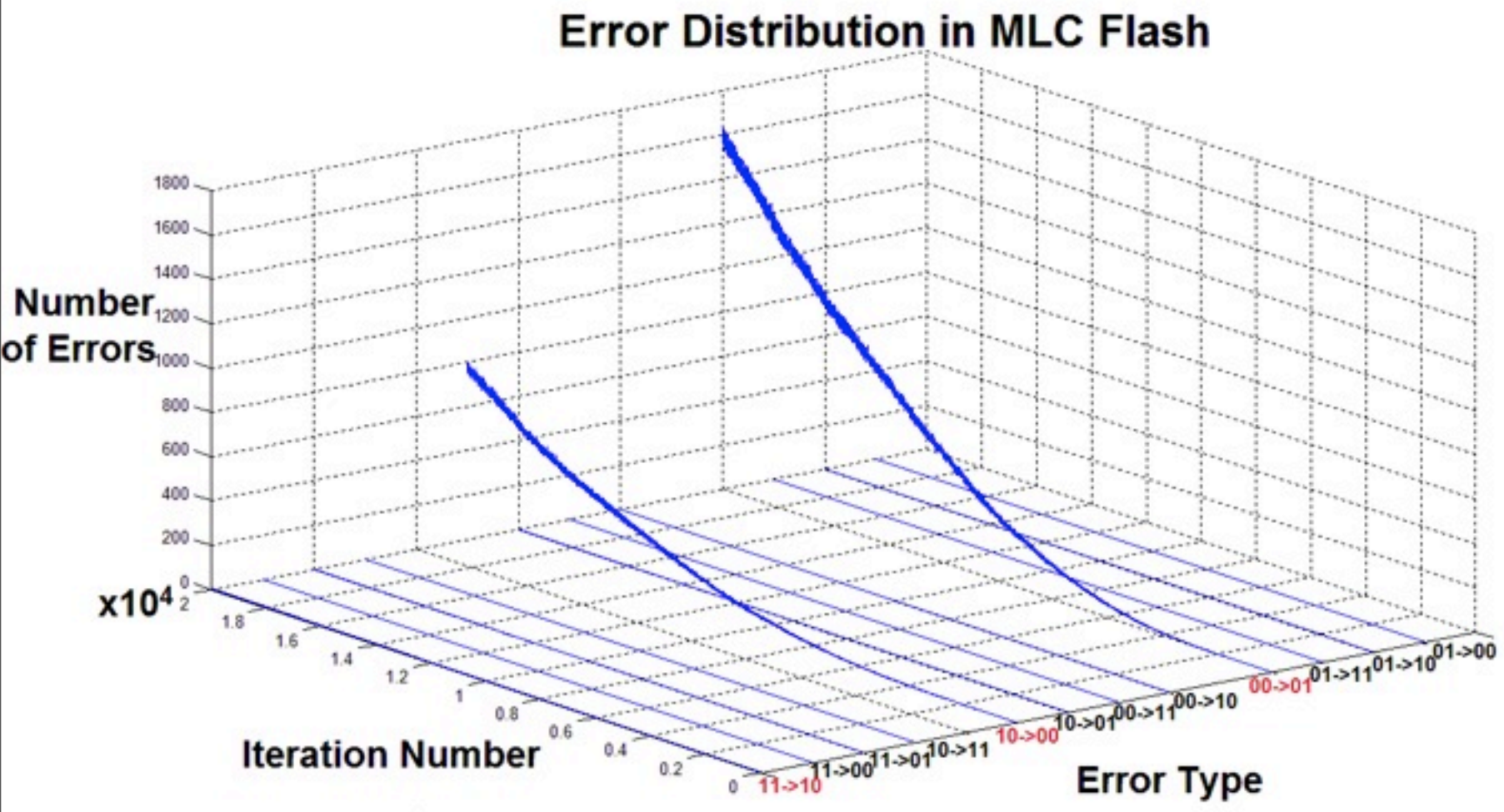
ECC scheme for MLC flash

- A common ECC in flash today is a **BCH code**
- Errors are corrected in each page **independently**
- In particular, in a pair of SLC and MLC pages sharing the same group of cells, errors are still corrected independently
- **Our goal**: to correct errors in a pair of pages **together**

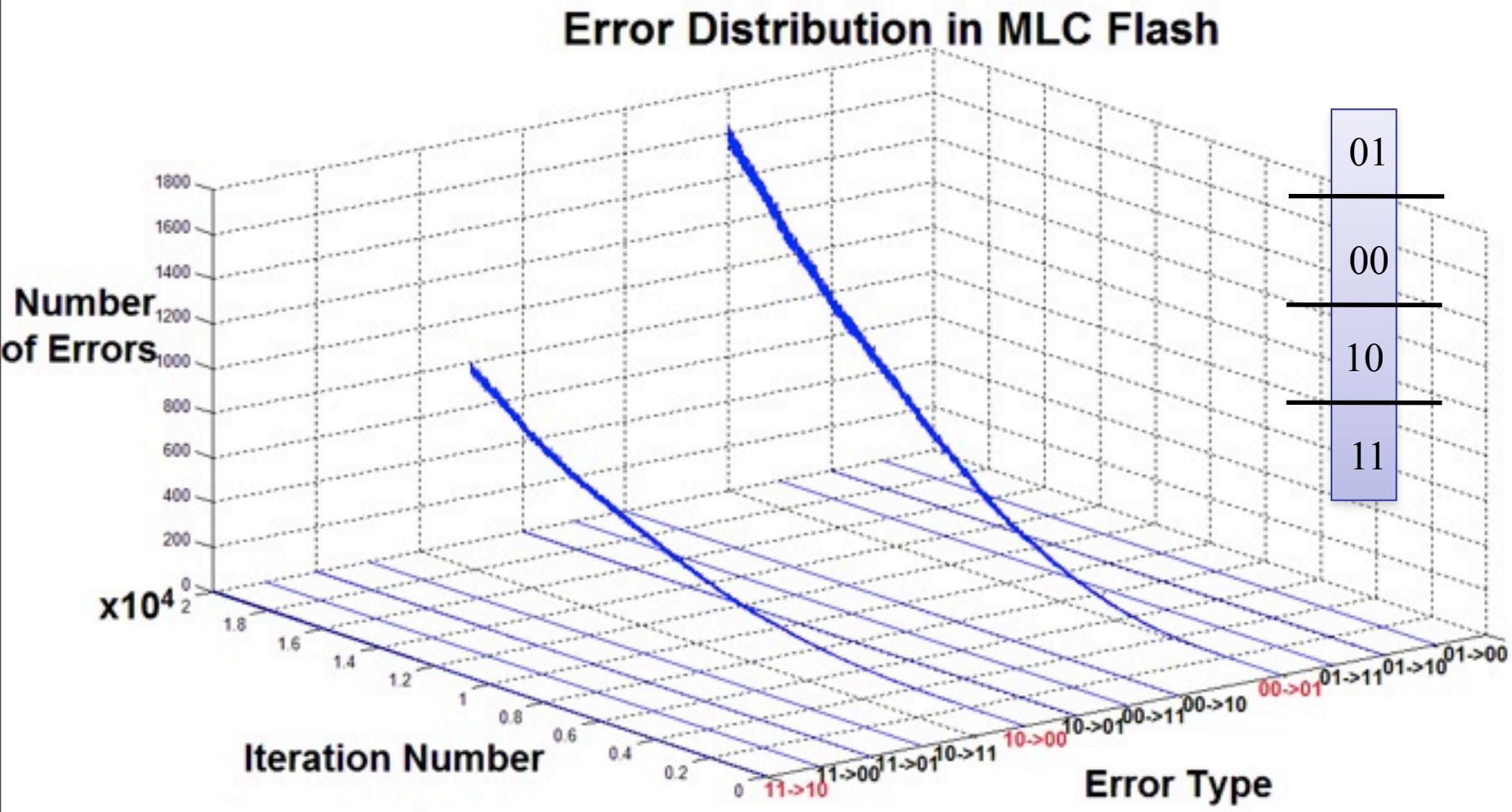
ECC scheme for MLC flash

- A common ECC in flash today is a **BCH code**
- Errors are corrected in each page **independently**
- In particular, in a pair of SLC and MLC pages sharing the same group of cells, errors are still corrected independently
- **Our goal**: to correct errors in a pair of pages **together**
- If a cell is in error, its level will typically increase by **one level**

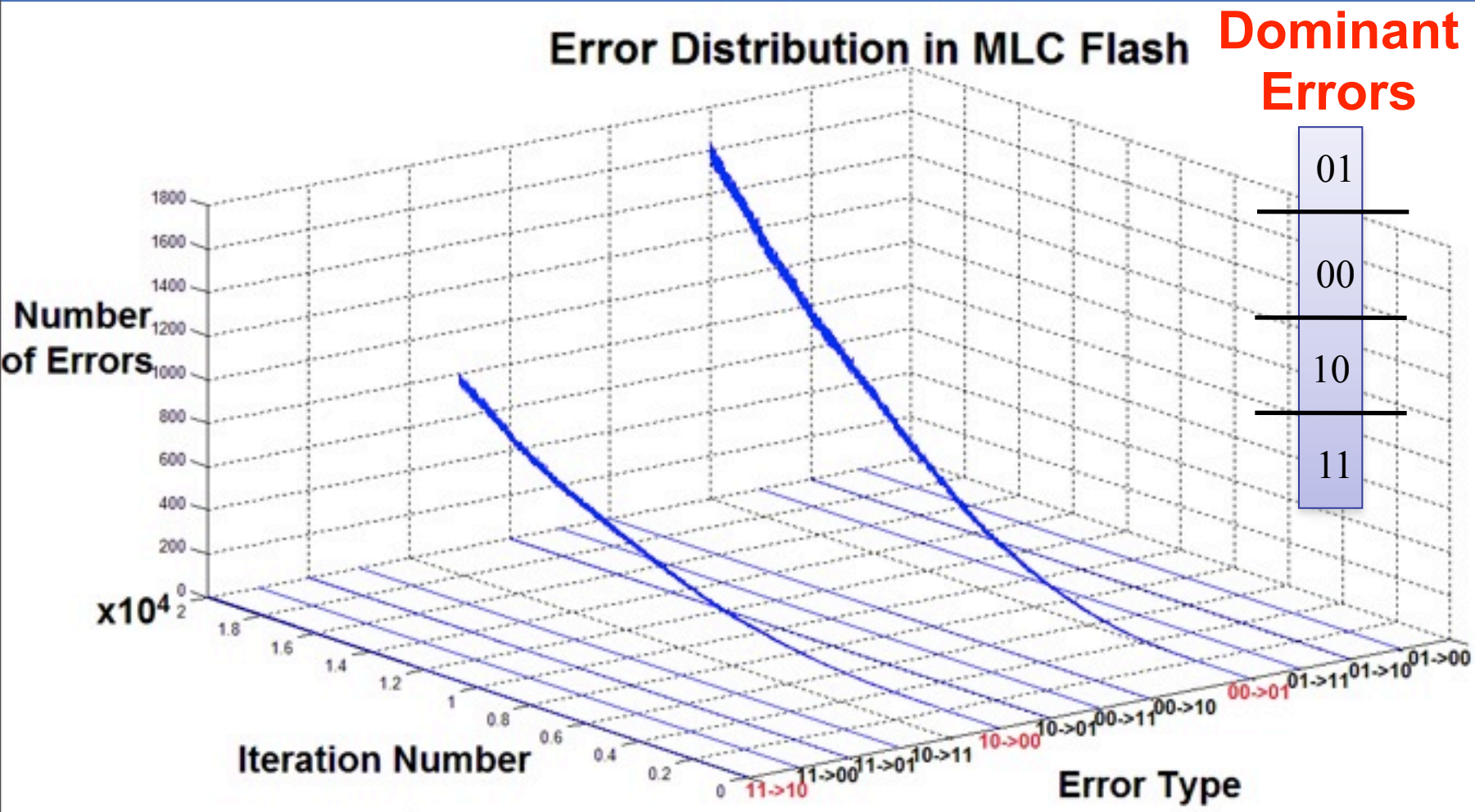
ECC scheme for MLC flash



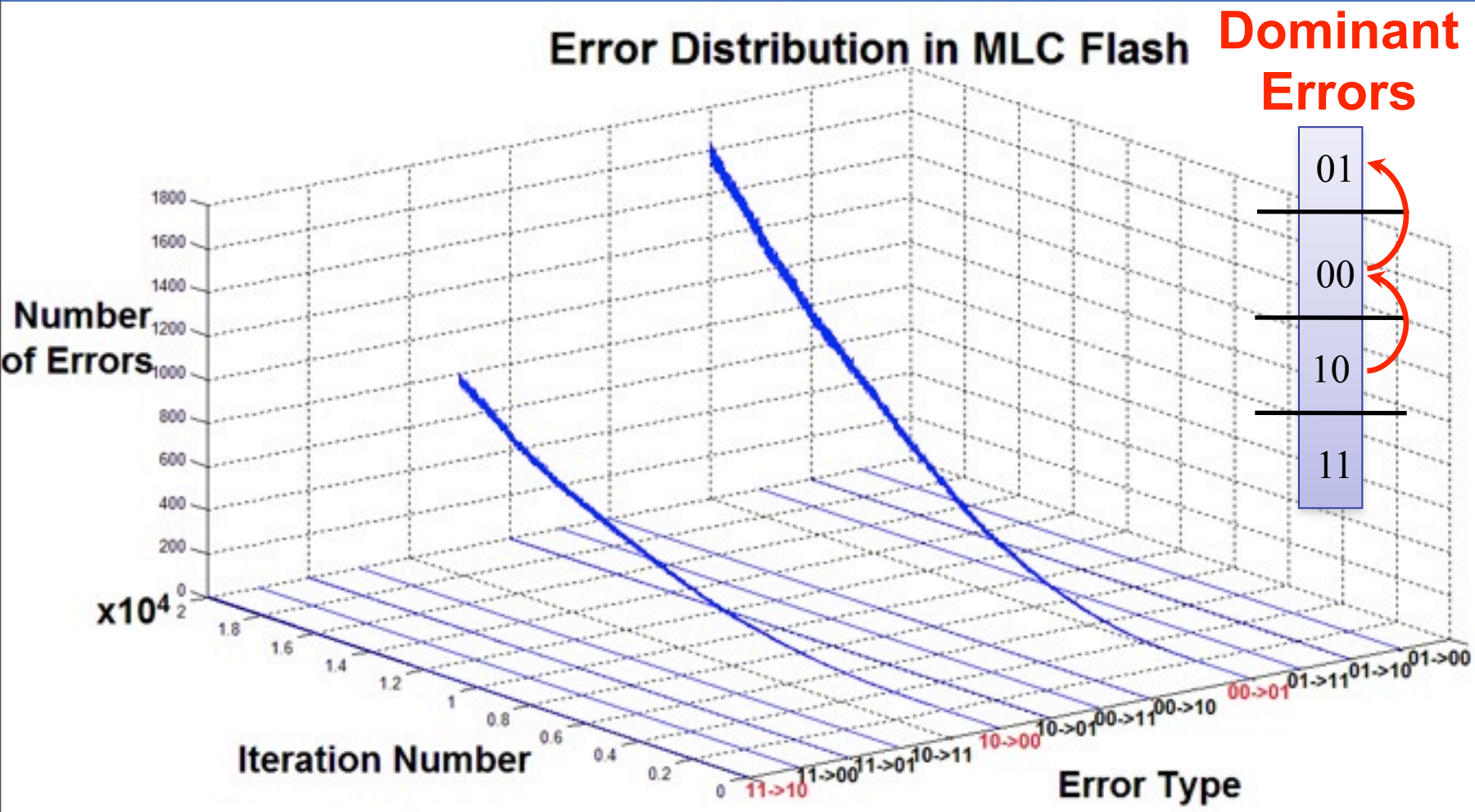
ECC scheme for MLC flash



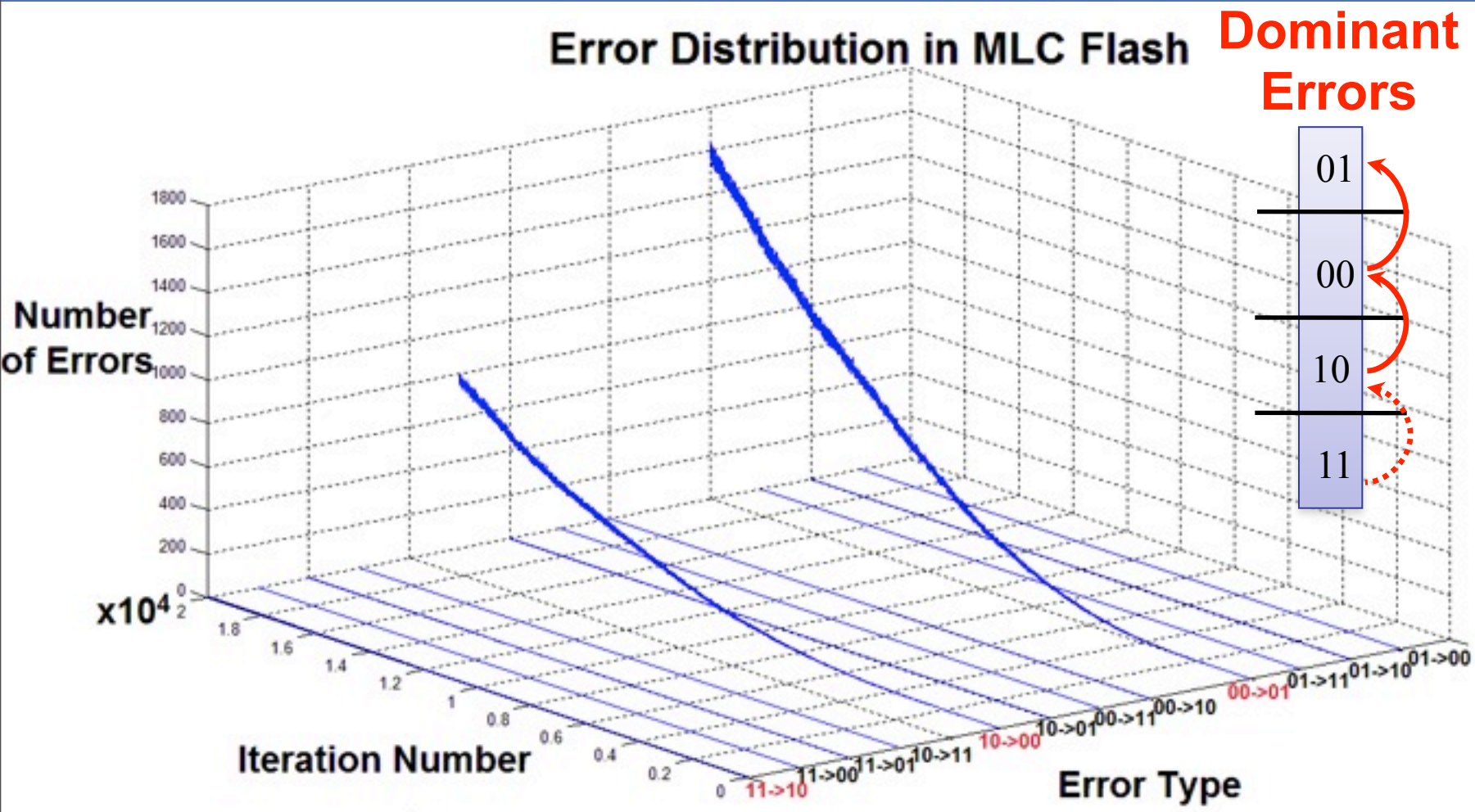
ECC scheme for MLC flash



ECC scheme for MLC flash



ECC scheme for MLC flash



ECC scheme for MLC flash

s_2

ECC scheme for MLC flash

- How to correct errors in a pair of pages together?

s_2

ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors

s_2

ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:

s_2

ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code
    C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$

s_2

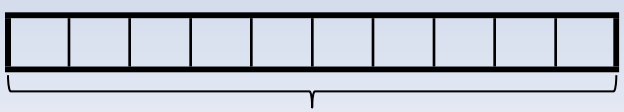
ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code
    C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$
  - The codes are "**compatible**" –
    For the same information word, **the r_1 redundancy bits** generated by the encoder of C_1 are **identical** to **the first r_1 redundancy bits** generated by the encoder of C_2

s_2

ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code
    C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$
  - The codes are "**compatible**" –
    For the same information word, **the r_1 redundancy bits** generated by the encoder of C_1 are **identical** to **the first r_1 redundancy bits** generated by the encoder of C_2

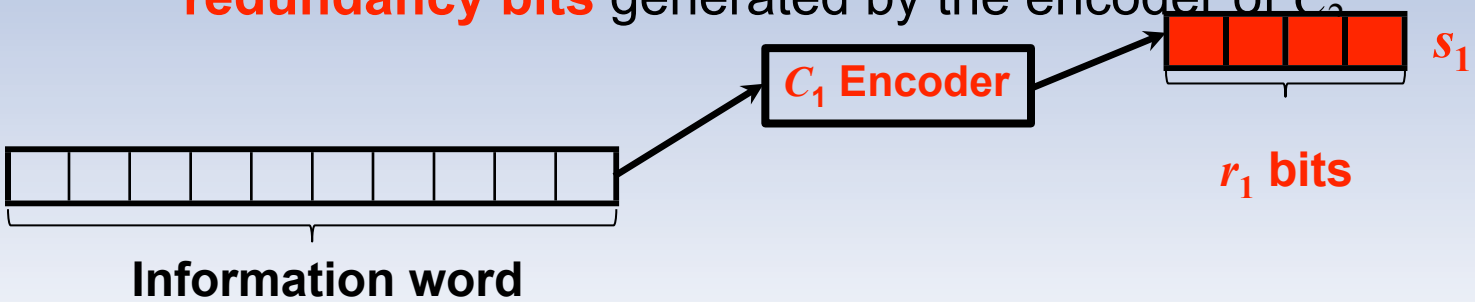


Information word

s_2

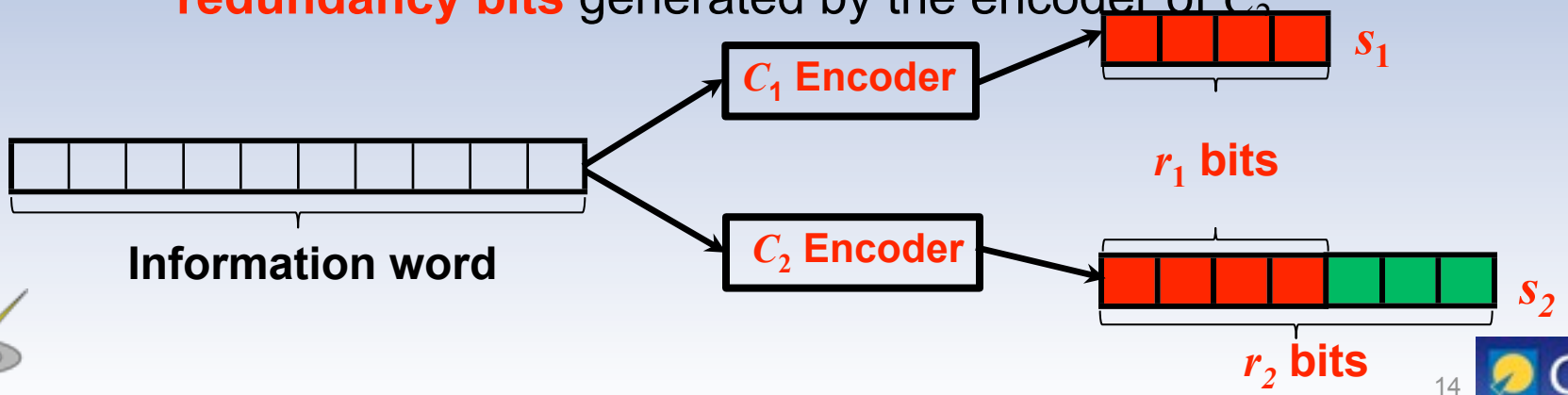
ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code
    C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$
  - The codes are "**compatible**" –
    For the same information word, **the r_1 redundancy bits** generated by the encoder of C_1 are **identical** to **the first r_1 redundancy bits** generated by the encoder of C_2



ECC scheme for MLC flash

- How to correct errors in a pair of pages together?
  - First, **one level errors** are corrected and then the other errors
- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code
    C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$
  - The codes are "**compatible**" –
    For the same information word, **the r_1 redundancy bits** generated by the encoder of C_1 are **identical** to **the first r_1 redundancy bits** generated by the encoder of C_2



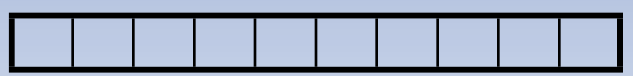
ECC scheme for MLC flash

ECC scheme for MLC flash

- **Code construction**:

ECC scheme for MLC flash

- **Code construction**:

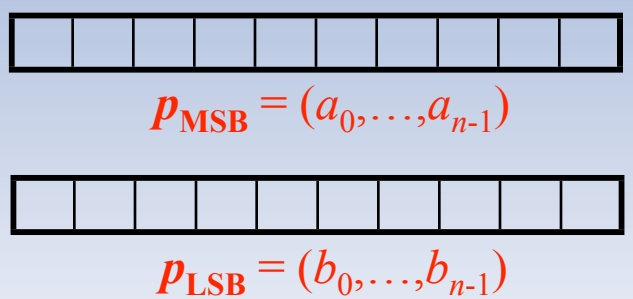


$p_{\mathrm{MSB}} = (a_0,\ldots,a_{n-1})$

$p_{\mathrm{LSB}} = (b_0,\ldots,b_{n-1})$

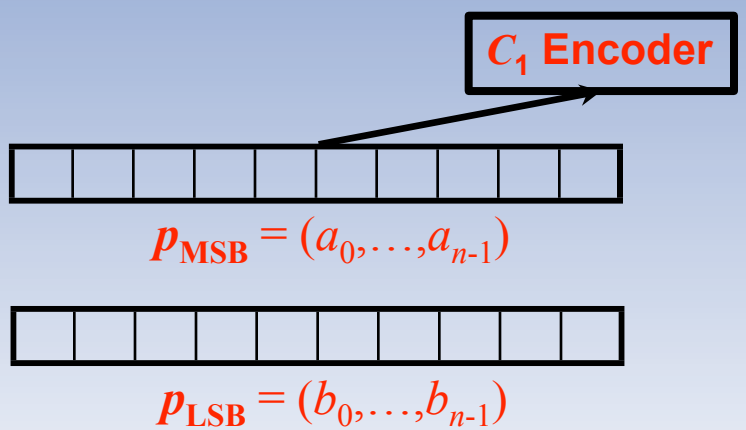
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible



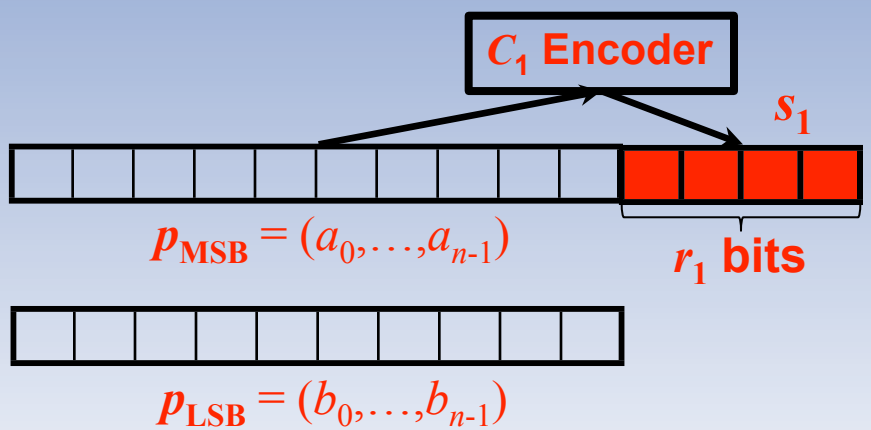
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible



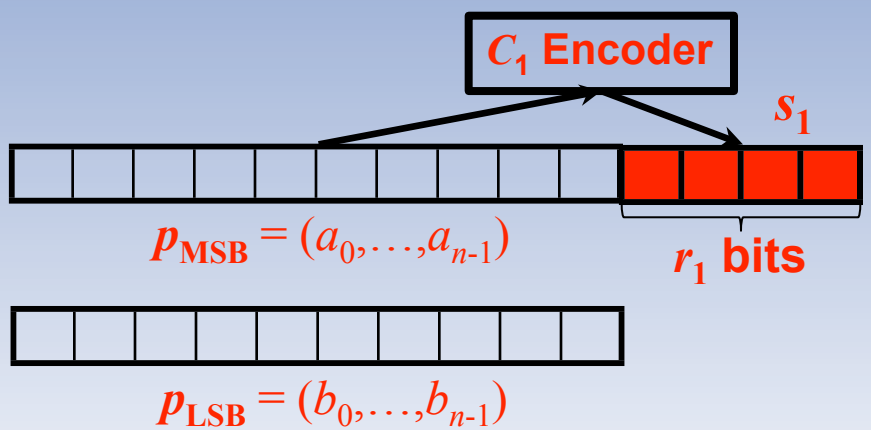
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible



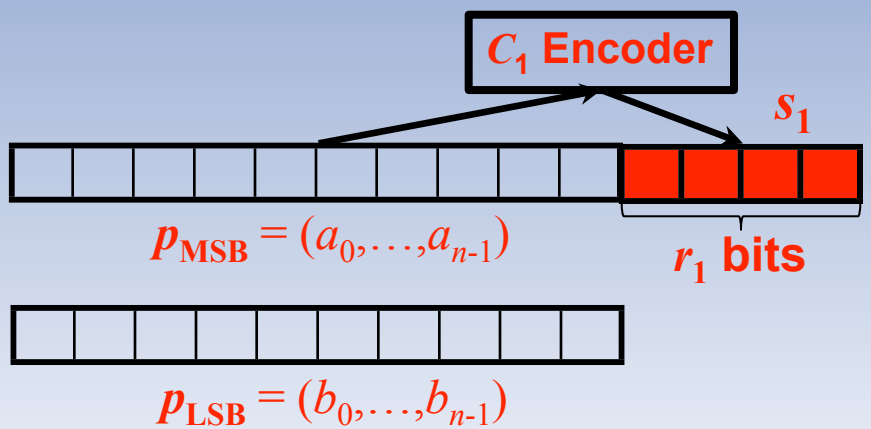
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:



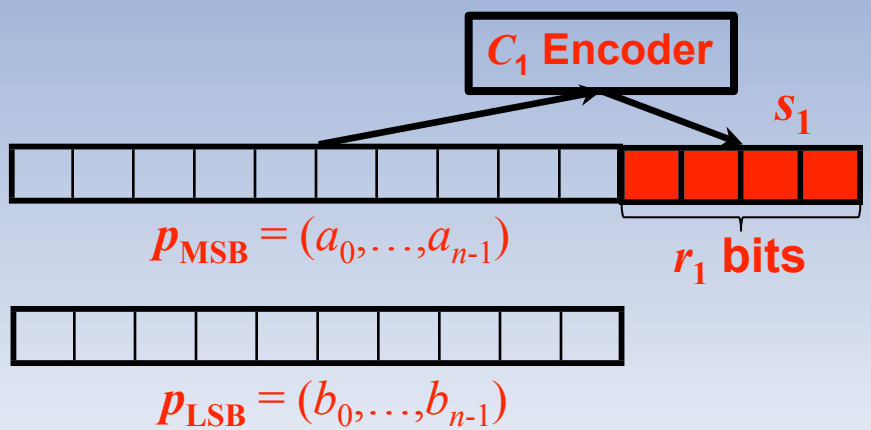
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:
  - $p_{\mathrm{MSB}} = (a_0,\ldots,a_{n-1})$ and $p_{\mathrm{LSB}} = (b_0,\ldots,b_{n-1})$ share the same group of cells.



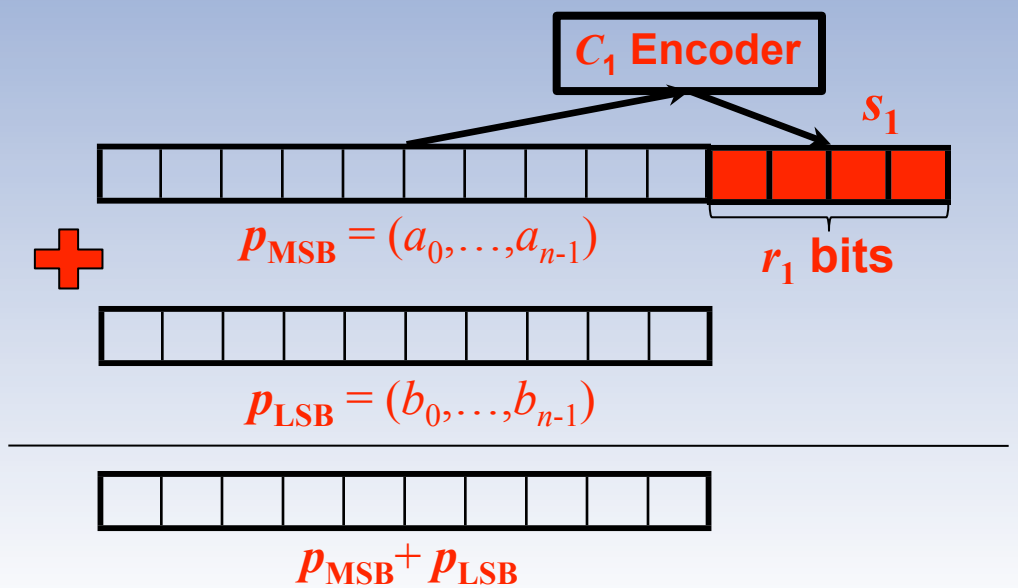
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:
  - $p_{\mathrm{MSB}} = (a_0,\ldots,a_{n-1})$ and $p_{\mathrm{LSB}} = (b_0,\ldots,b_{n-1})$ share the same group of cells.
  - Calculate s_1, the r_1 redundancy bits of C_1 corresponding to p_{MSB}



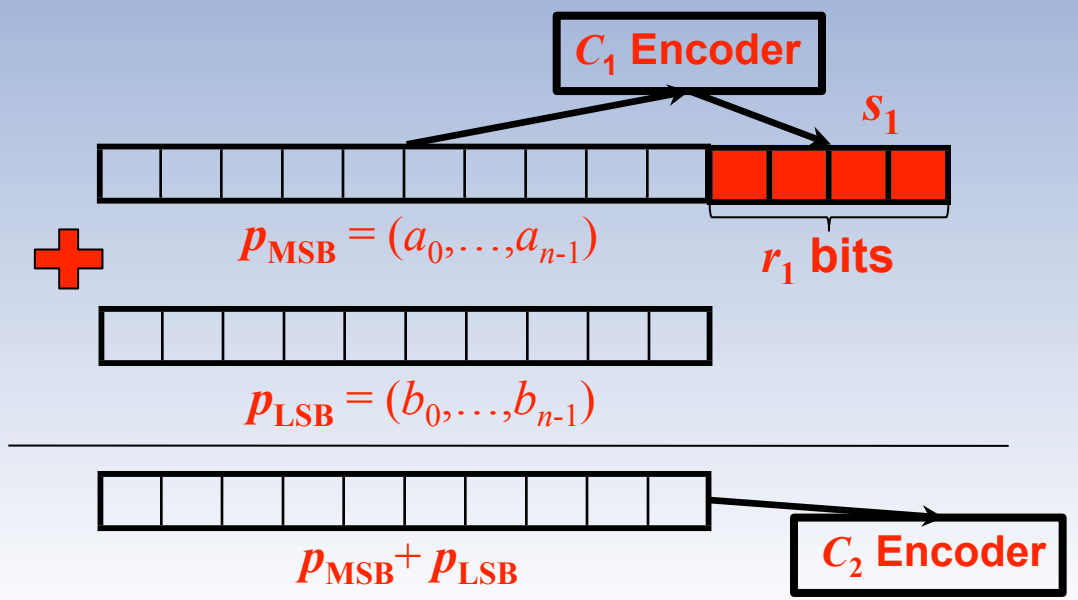
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:
  - $p_{\mathrm{MSB}} = (a_0,\ldots,a_{n-1})$ and $p_{\mathrm{LSB}} = (b_0,\ldots,b_{n-1})$ share the same group of cells.
  - Calculate s_1, the r_1 redundancy bits of C_1 corresponding to p_{MSB}



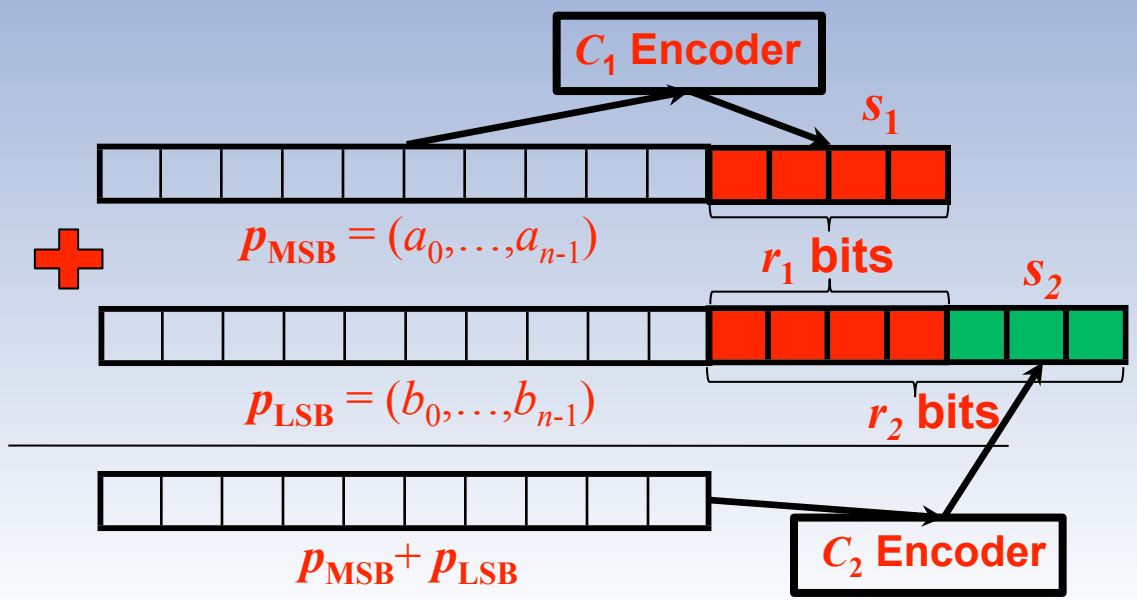
ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:
  - $p_{\text{MSB}} = (a_0,\ldots,a_{n-1})$ and $p_{\text{LSB}} = (b_0,\ldots,b_{n-1})$ share the same group of cells.
  - Calculate s_1, the r_1 redundancy bits of C_1 corresponding to p_{MSB}

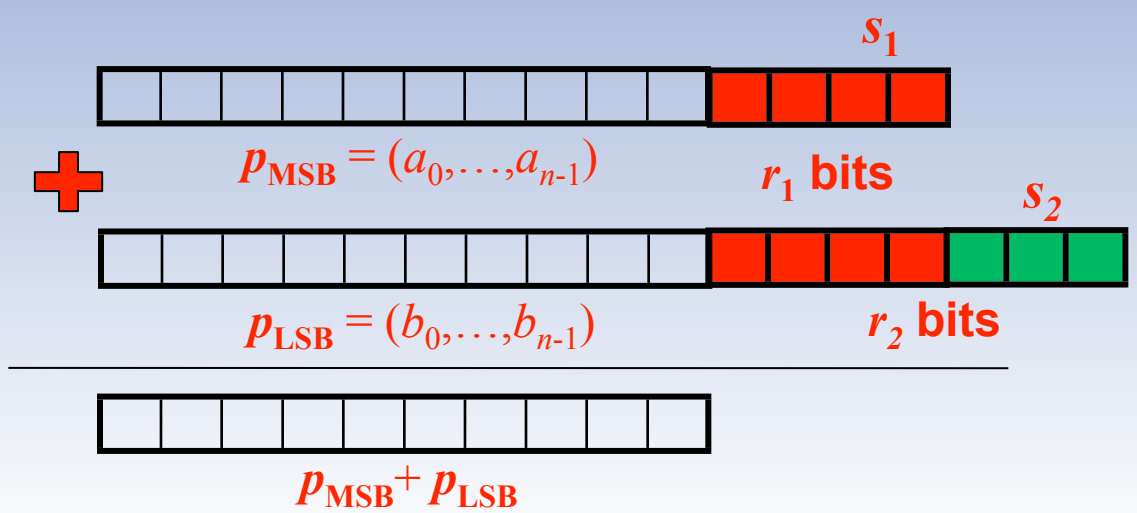


ECC scheme for MLC flash

- **Code construction**:
  - C_1 is a t_1-error-correcting BCH code, C_2 is a t_2-error-correcting BCH code, where $t_2 > t_1$, and the codes are compatible
- **Encoding**:
  - $p_{\text{MSB}} = (a_0,\ldots,a_{n-1})$ and $p_{\text{LSB}} = (b_0,\ldots,b_{n-1})$ share the same group of cells.
  - Calculate s_1, the r_1 redundancy bits of C_1 corresponding to p_{MSB}

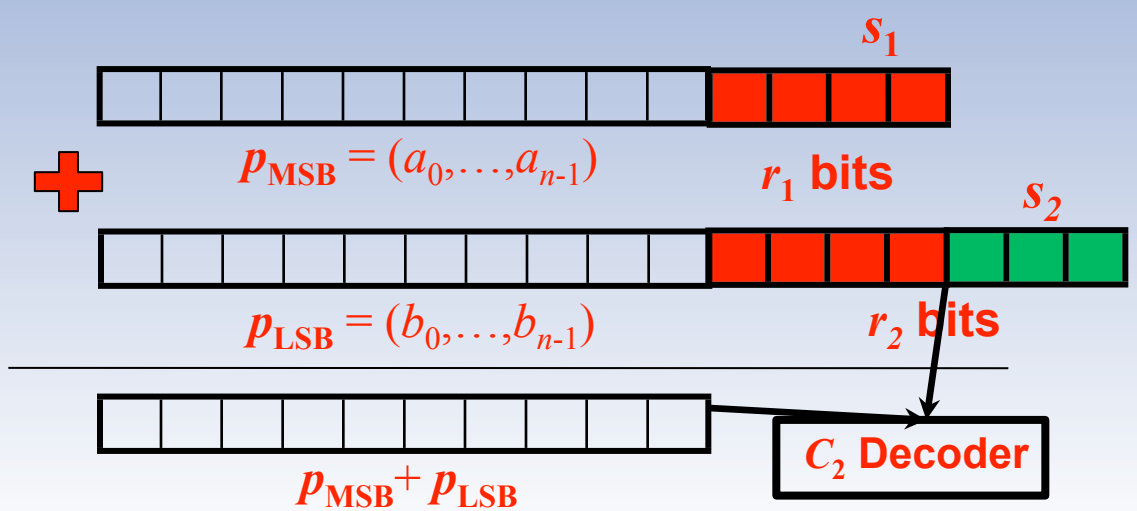


ECC scheme for MLC flash



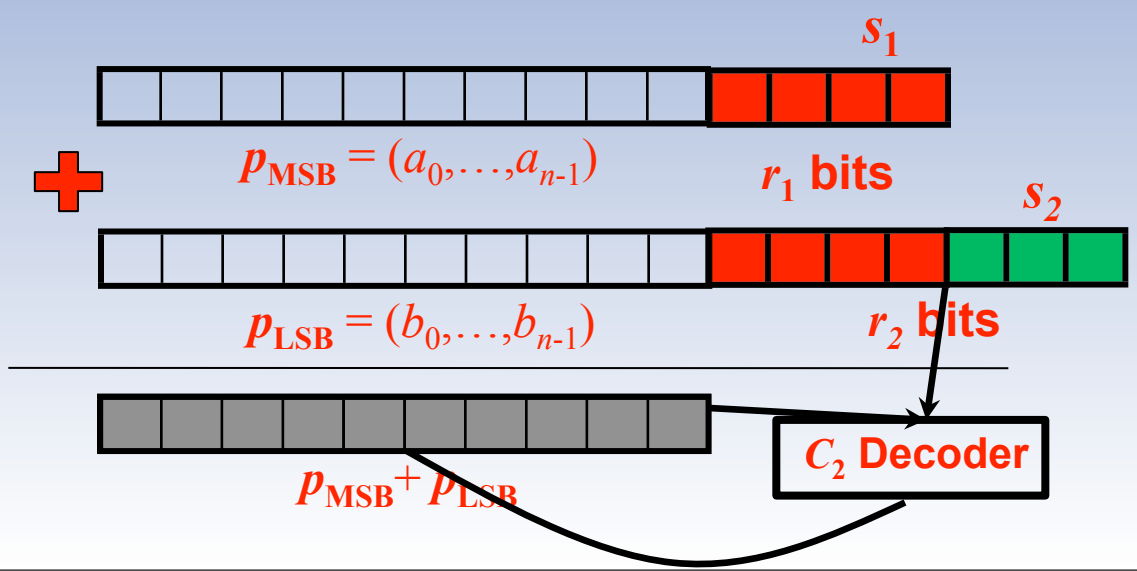
ECC scheme for MLC flash

- **Decoding**:



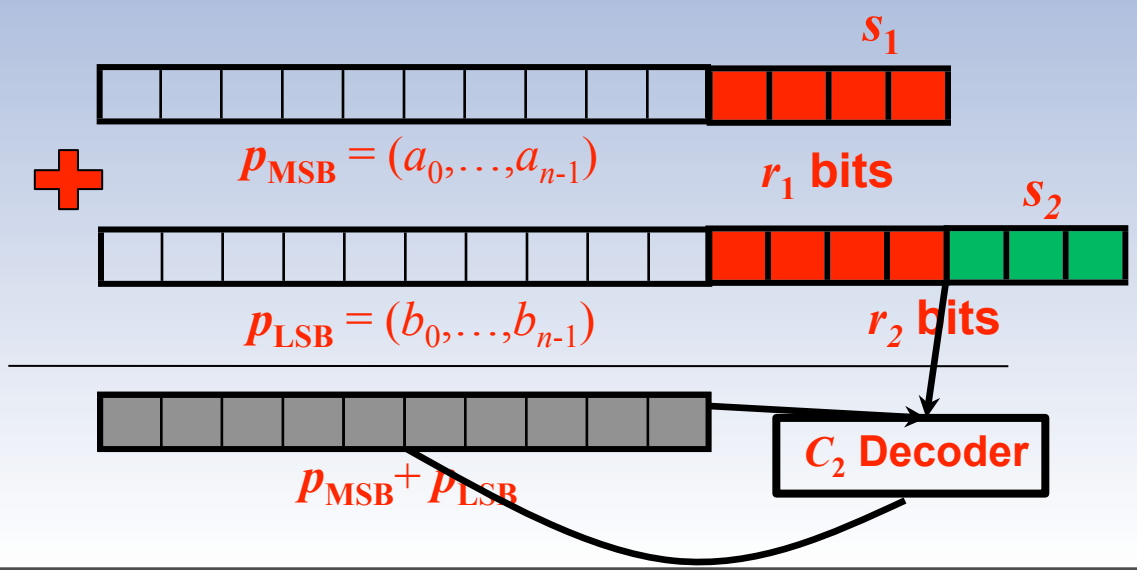
ECC scheme for MLC flash

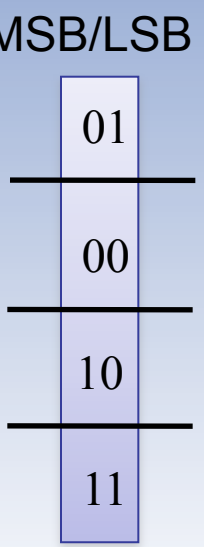
- **Decoding**:



ECC scheme for MLC flash

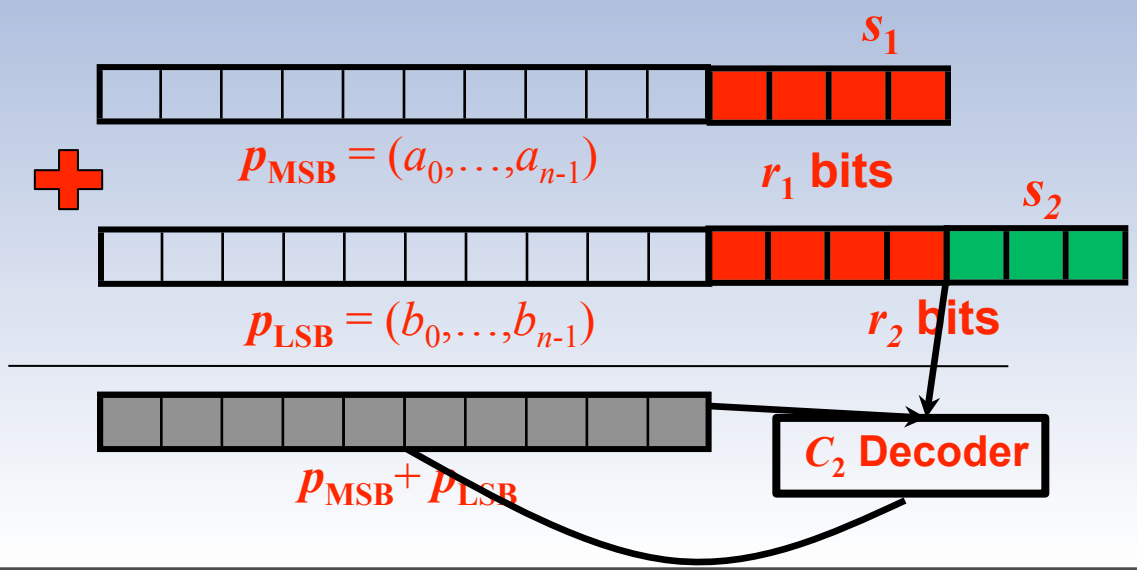
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$

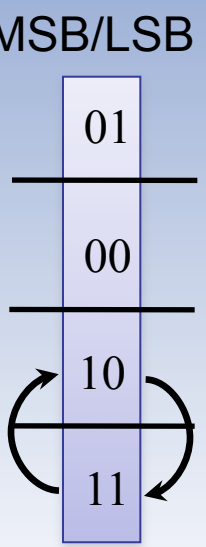




ECC scheme for MLC flash

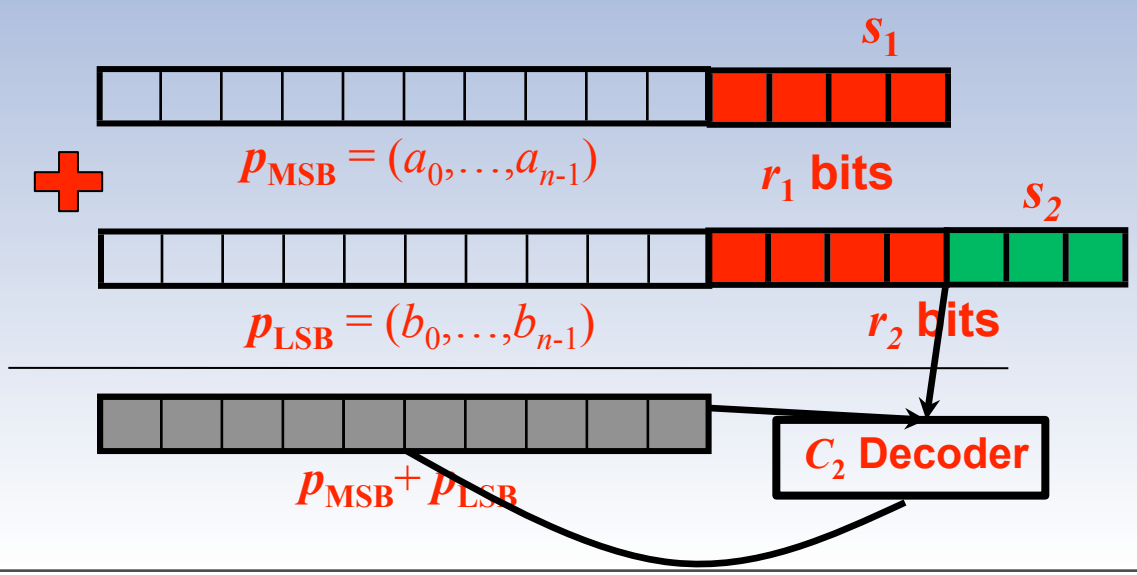
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:

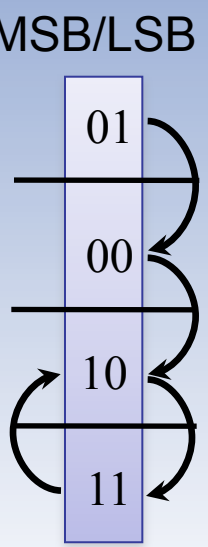




ECC scheme for MLC flash

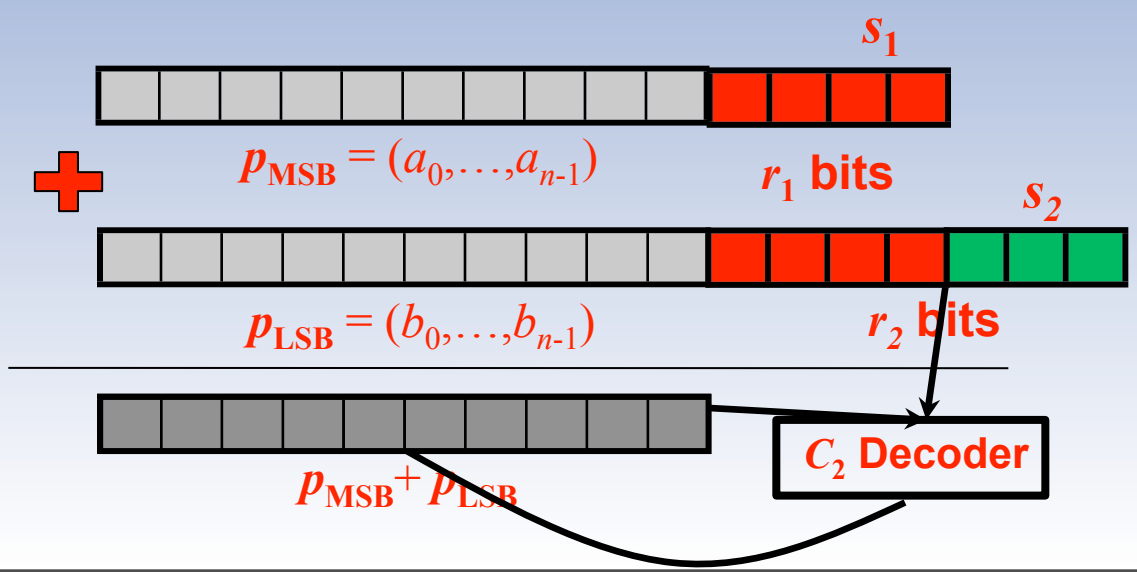
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa

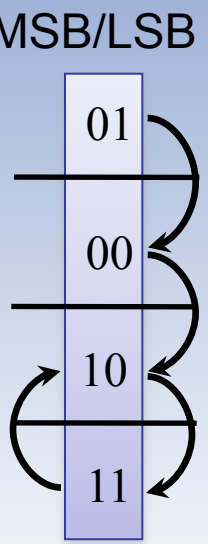




ECC scheme for MLC flash

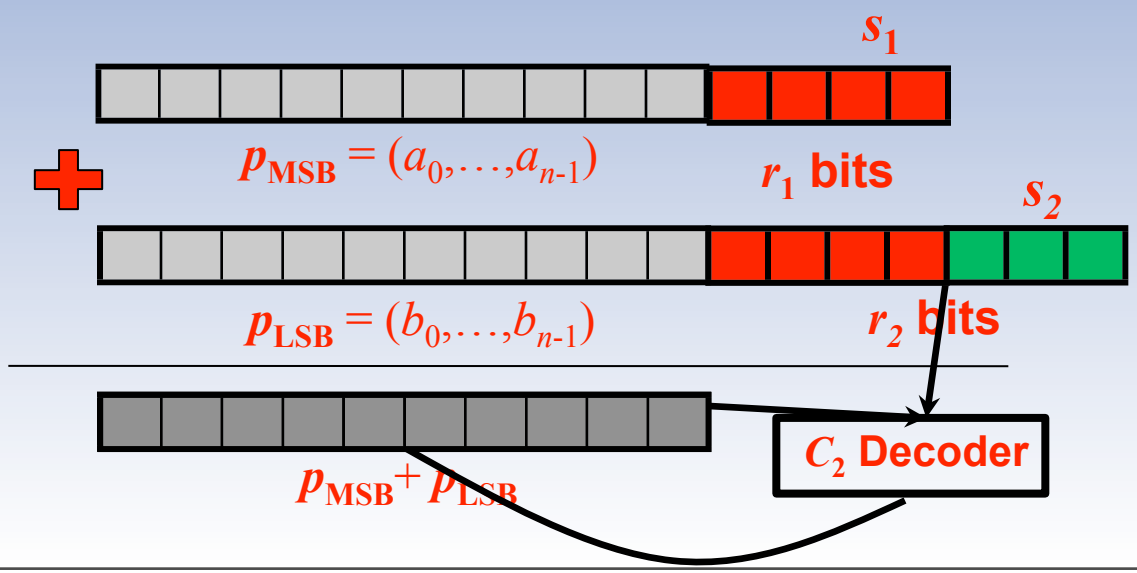
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa

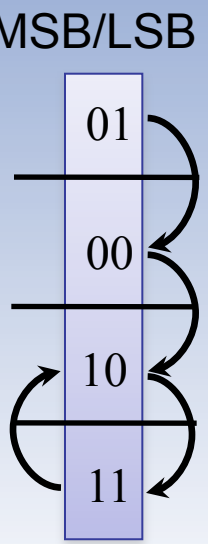




ECC scheme for MLC flash

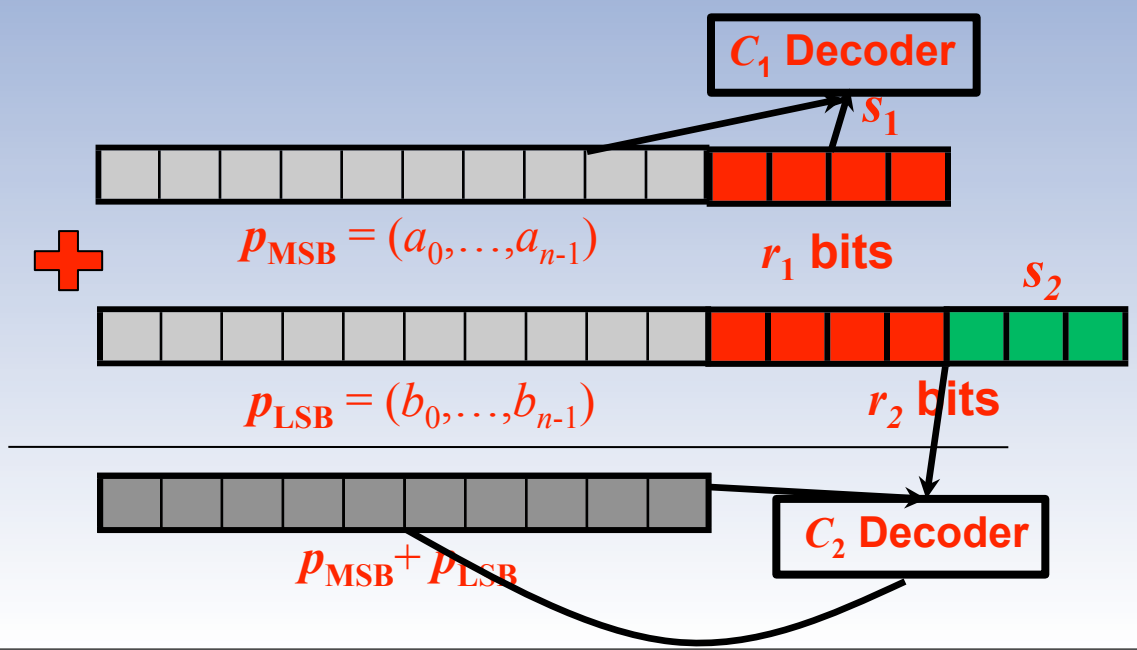
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**

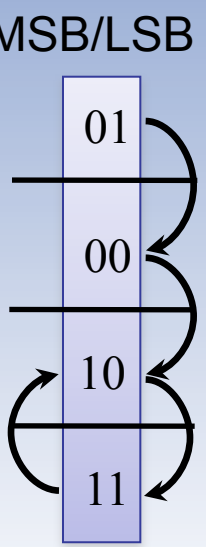




ECC scheme for MLC flash

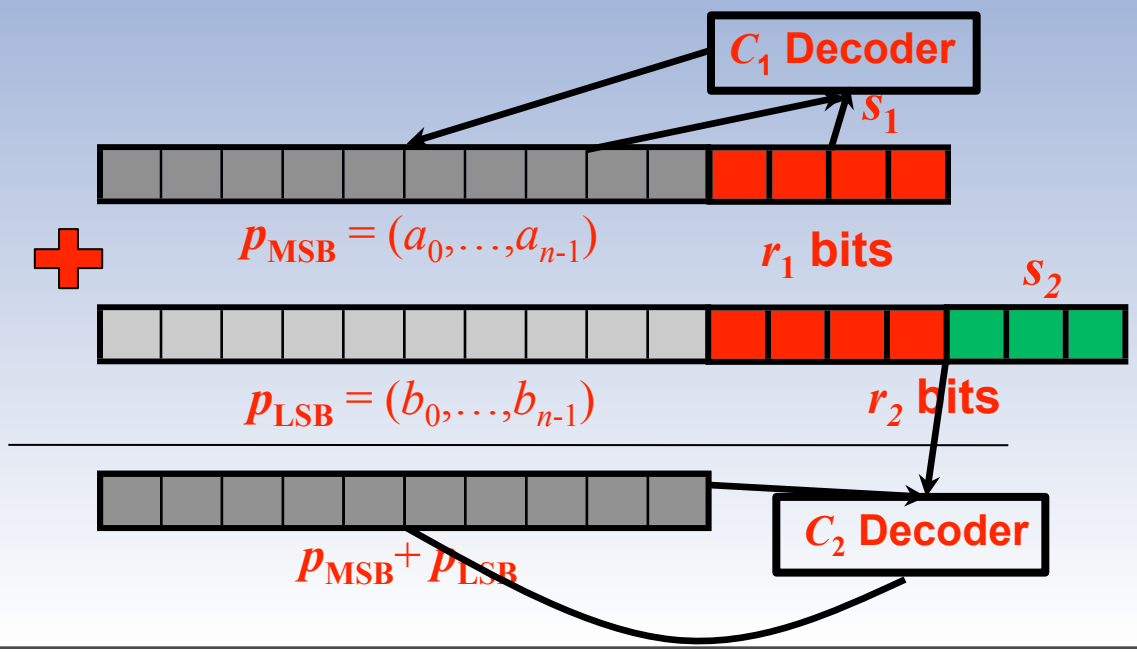
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**

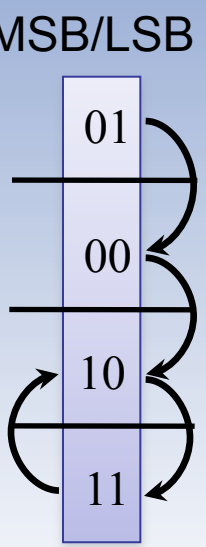




ECC scheme for MLC flash

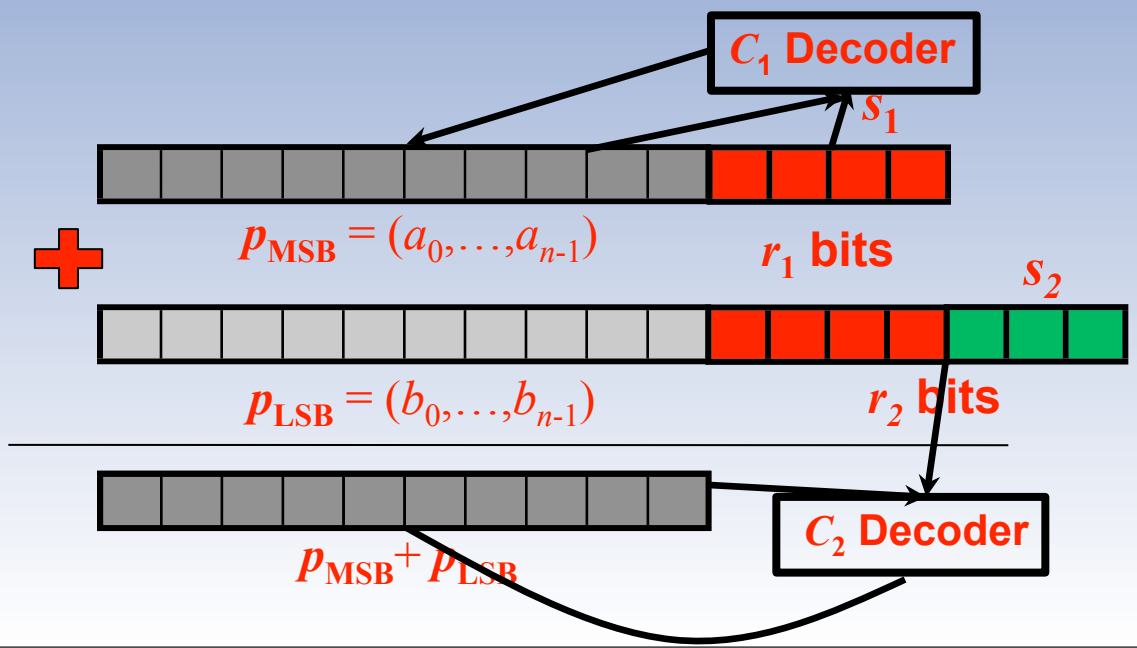
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**

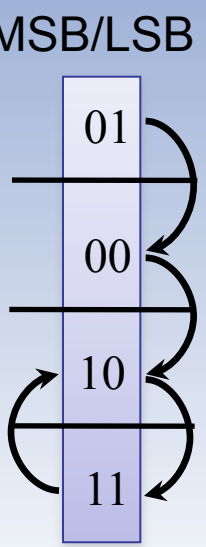




ECC scheme for MLC flash

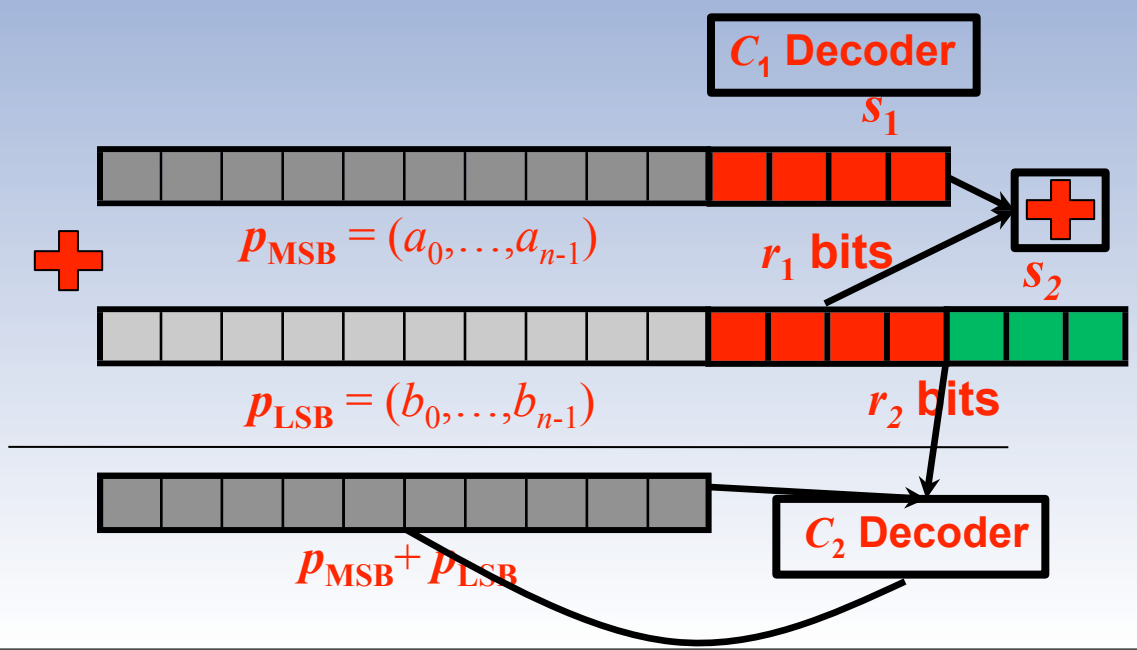
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**
  - Using the r_1 bits of s_1, find up to t_1 errors in p_{MSB}

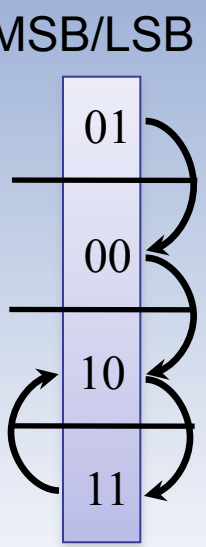




ECC scheme for MLC flash

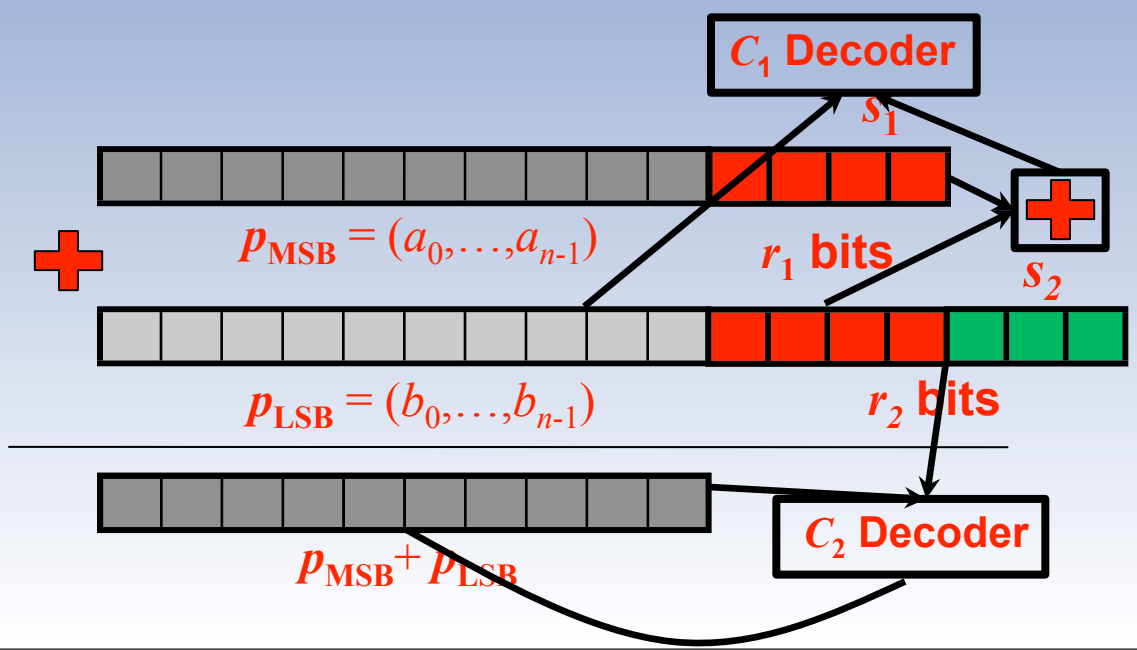
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**
  - Using the r_1 bits of s_1, find up to t_1 errors in p_{MSB}

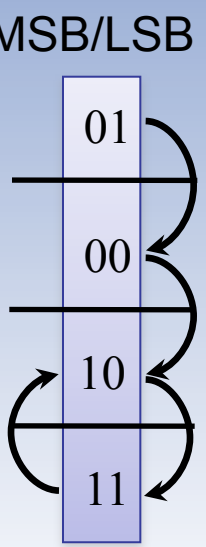




ECC scheme for MLC flash

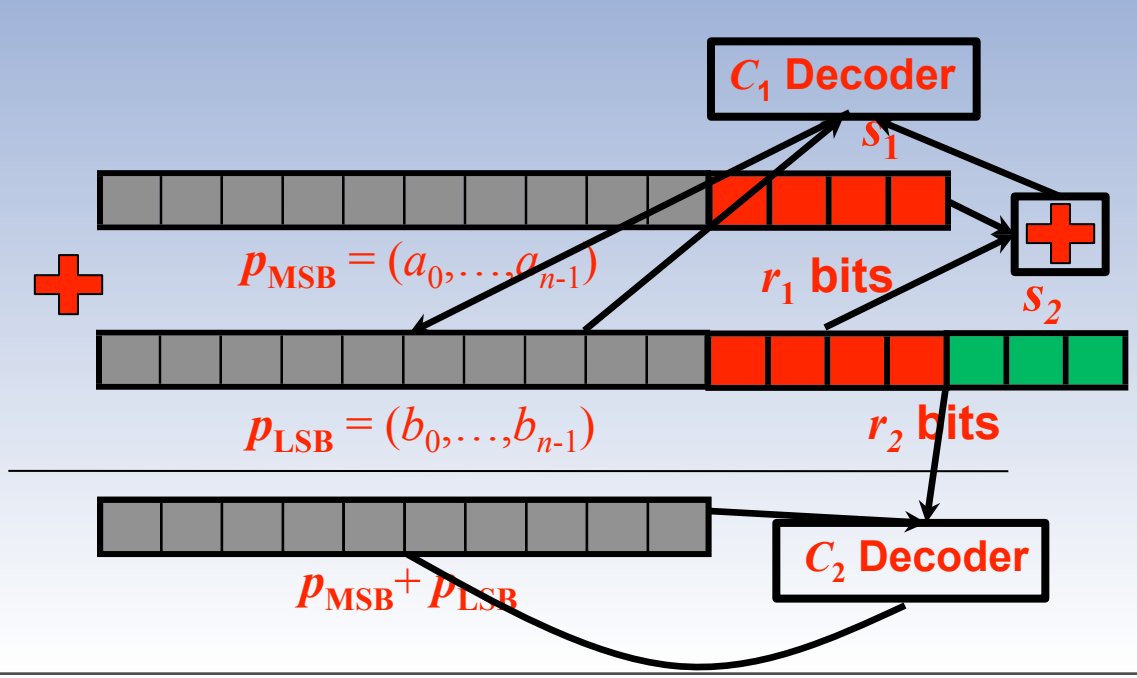
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**
  - Using the r_1 bits of s_1, find up to t_1 errors in p_{MSB}

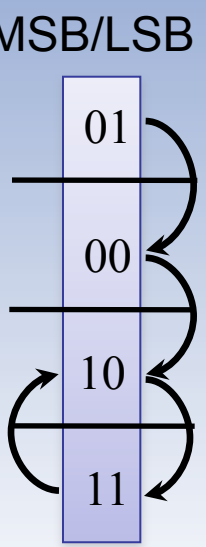


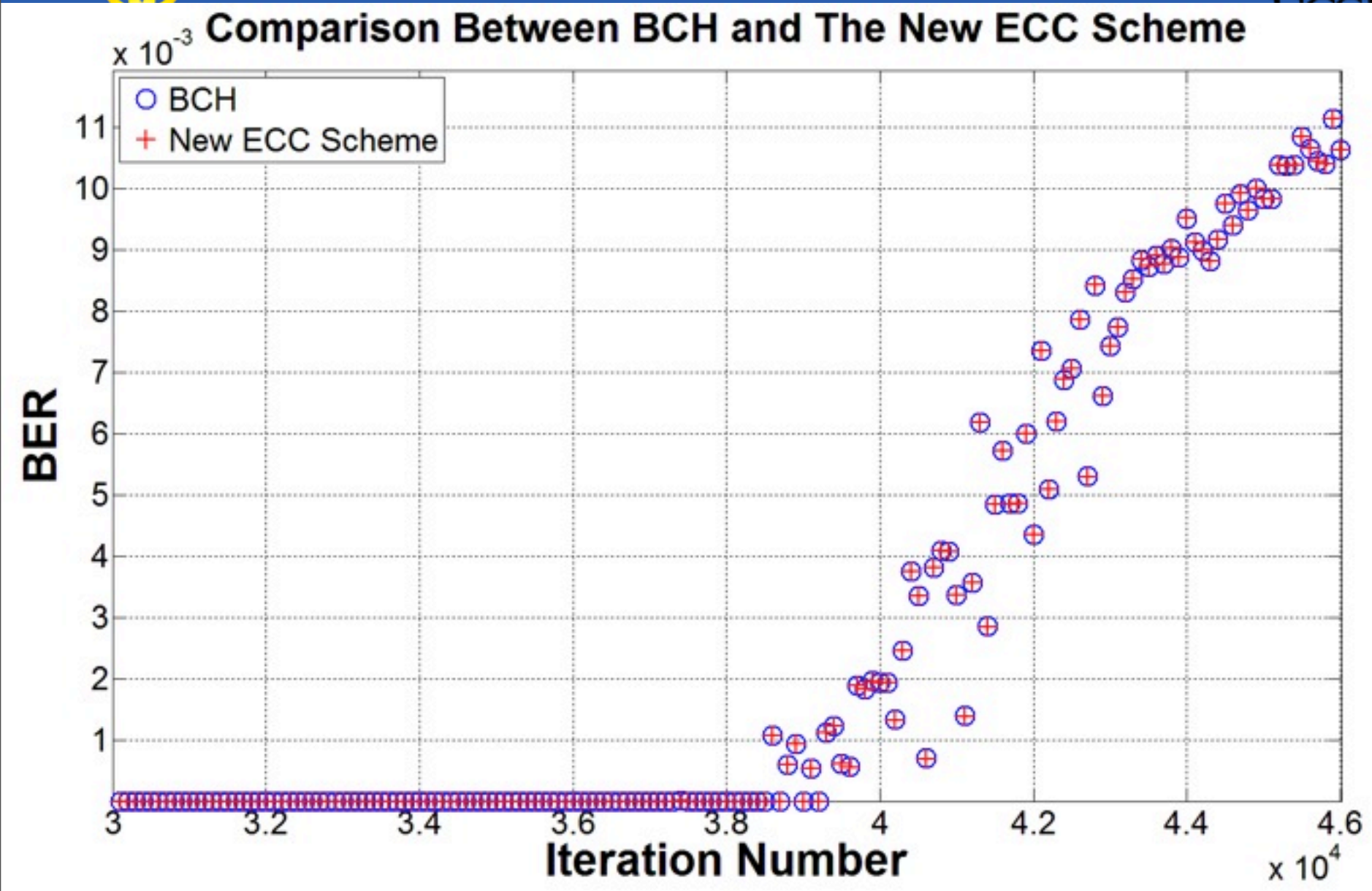


ECC scheme for MLC flash

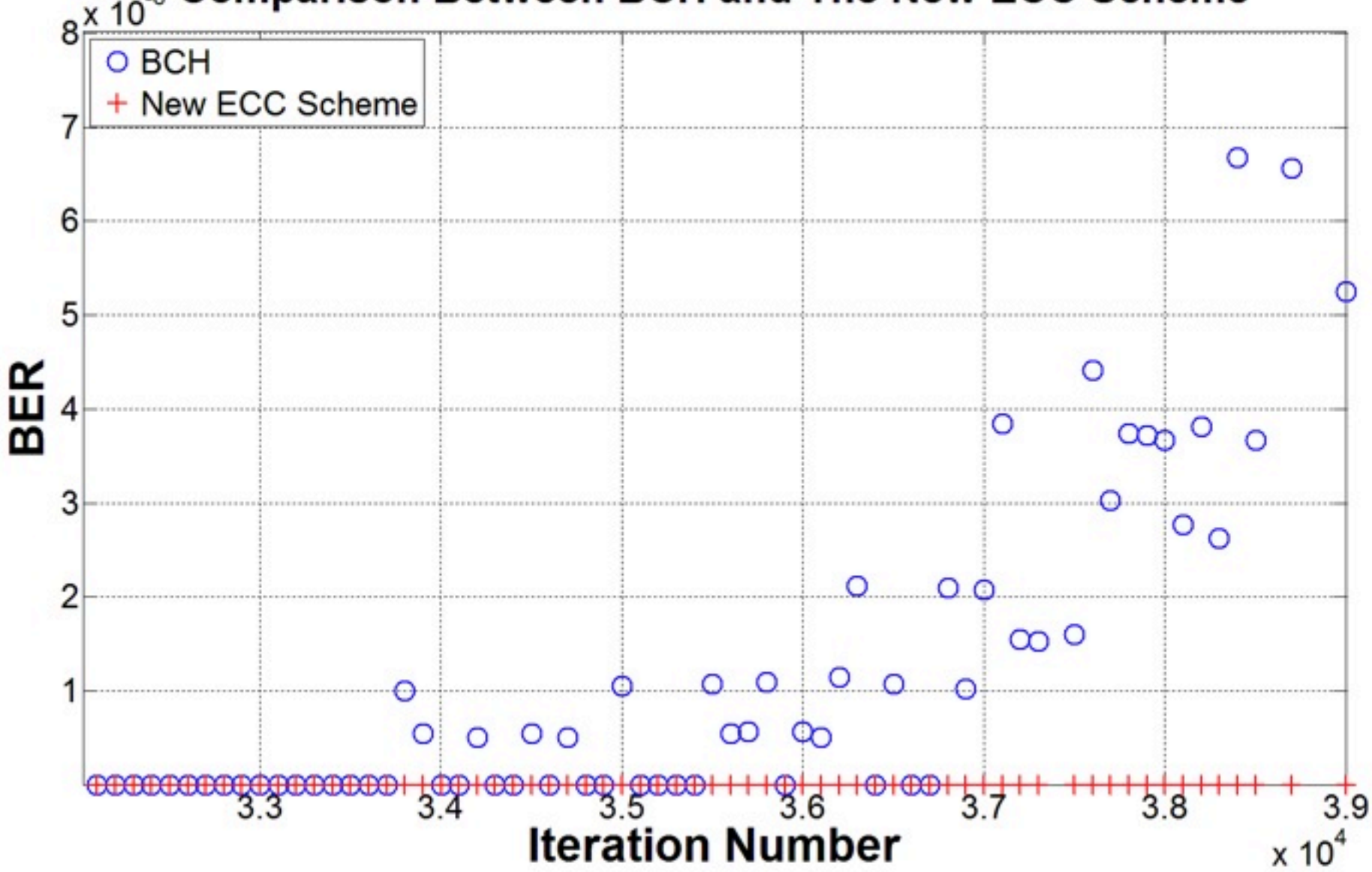
- **Decoding**:
  - Using the r_2 bits of s_2 find up to t_2 errors in $p_{MSB} + p_{LSB}$
  - Change the state of erroneous cells as follows:
    - Level **11** is changed to level **10** and vice versa
    - Level **00** is changed to level **10** and level **01** is changed to level **00**
  - Using the r_1 bits of s_1, find up to t_1 errors in p_{MSB}






Comparison Between BCH and The New ECC Scheme
x 10^-3
BCH
New ECC Scheme
BER
11
10
9
8
7
6
5
4
3
2
1
3
3.2
3.4
3.6
3.8
4
4.2
4.4
4.6
Iteration Number
x 10^4

Comparison Between BCH and The New ECC Scheme



Write Once Memory (WOM) Codes for SLC

Write Once Memory (WOM) Codes for SLC

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

	1st write	2nd write
data		
cells		

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

	1st write	2nd write
data	01	
cells	100	

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

	1st write	2nd write
data	01	11
cells	100	110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

	1st write	2nd write
data	01	11
cells	100	110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

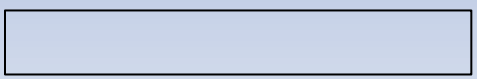
	1st write	2nd write
data	01	11
cells	100	110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

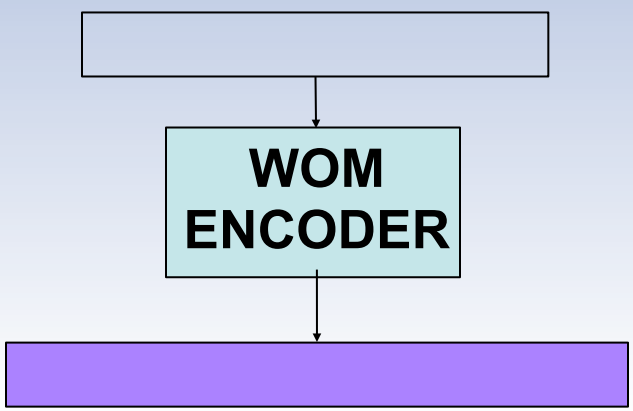


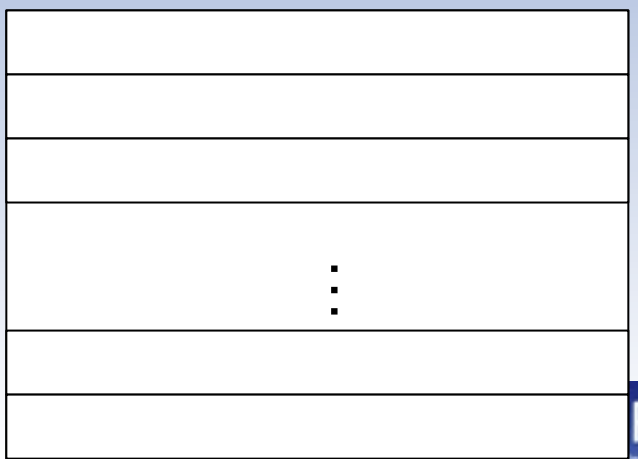
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



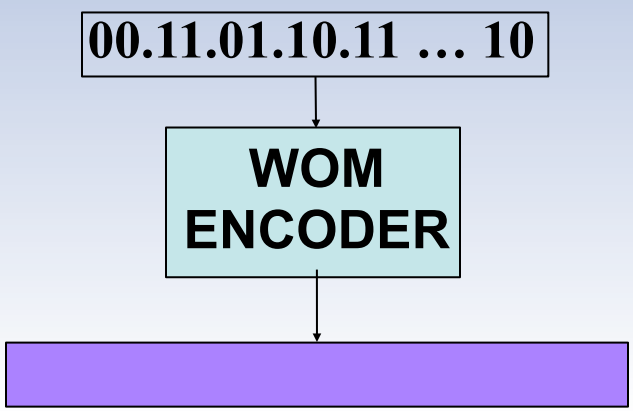


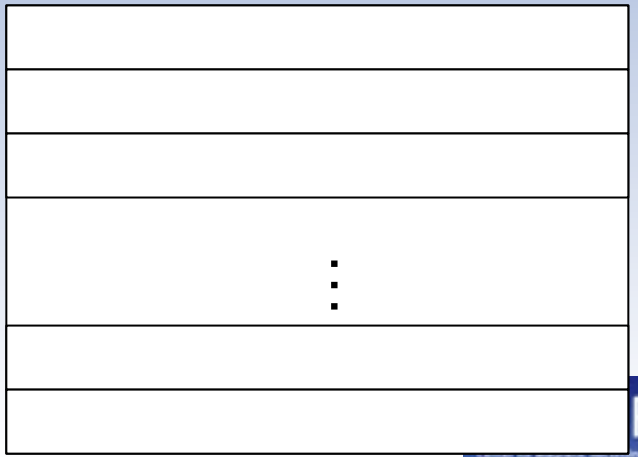
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



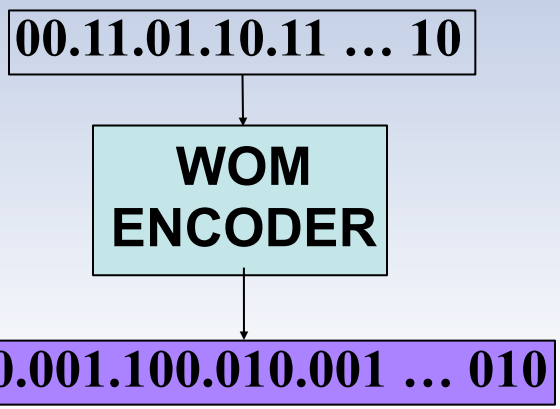


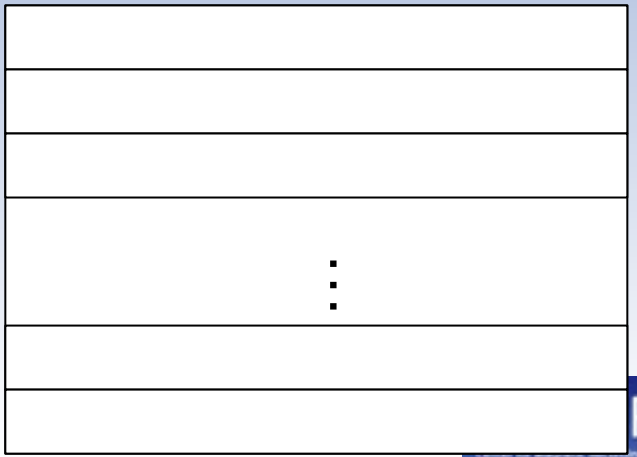
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



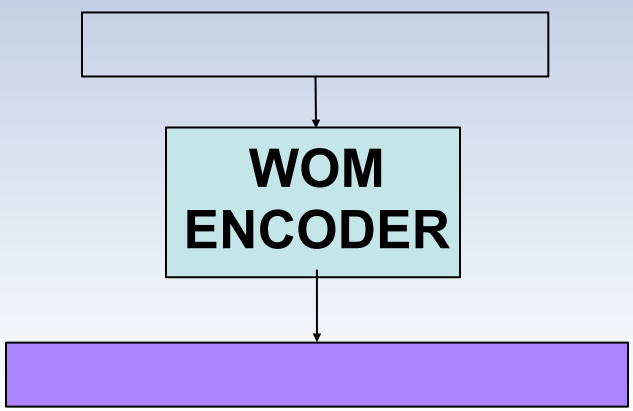


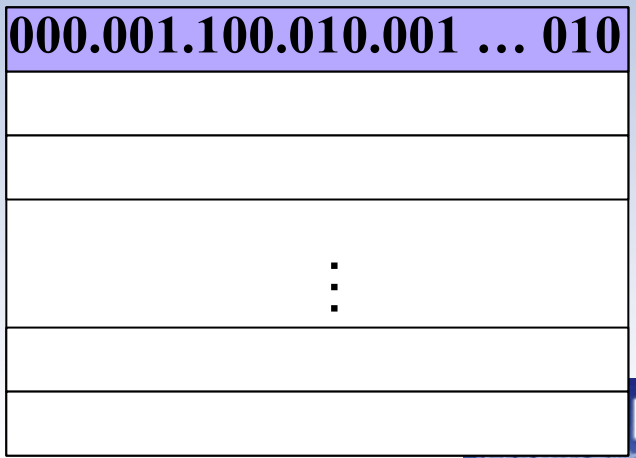
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



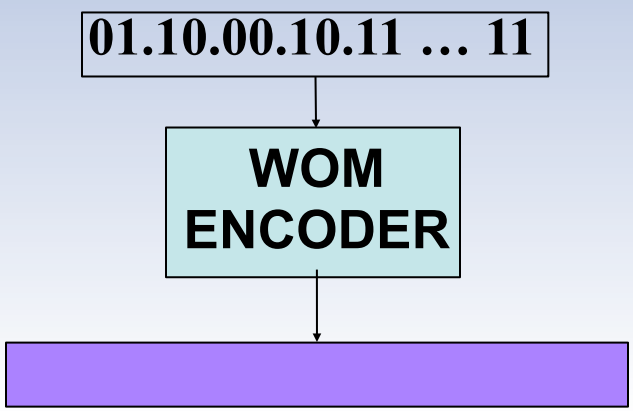


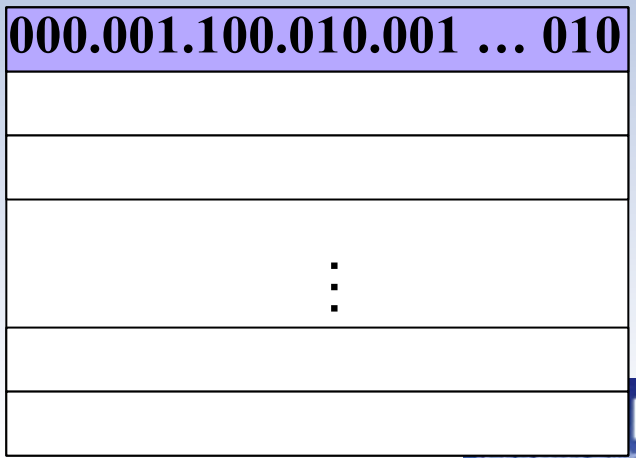
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



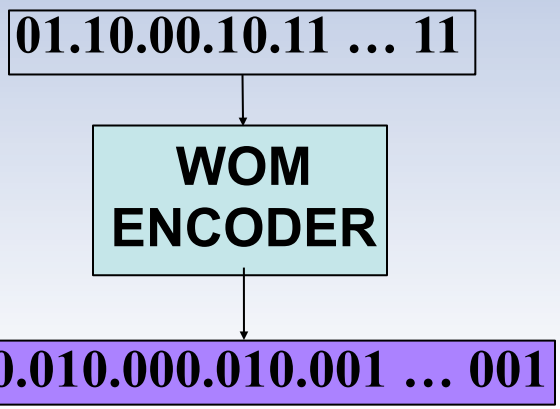


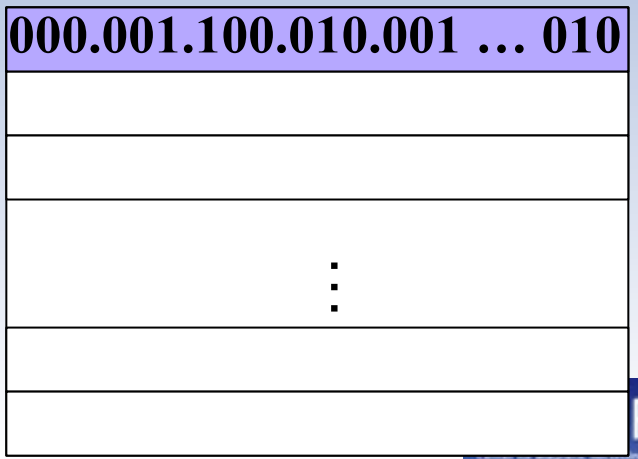
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



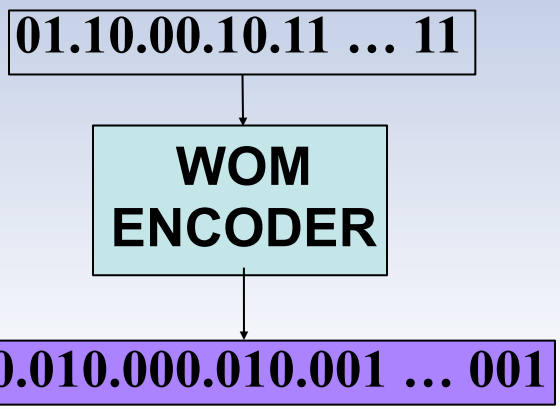


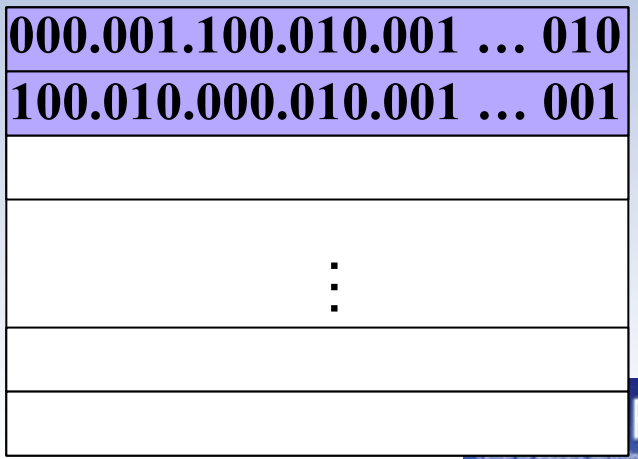
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



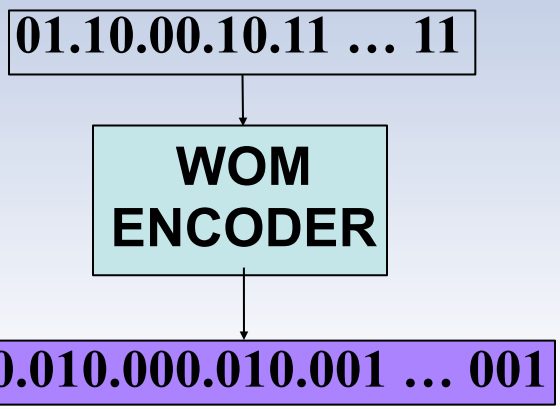


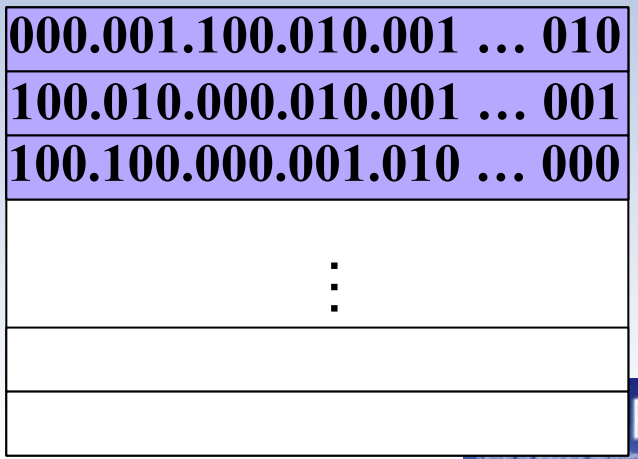
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



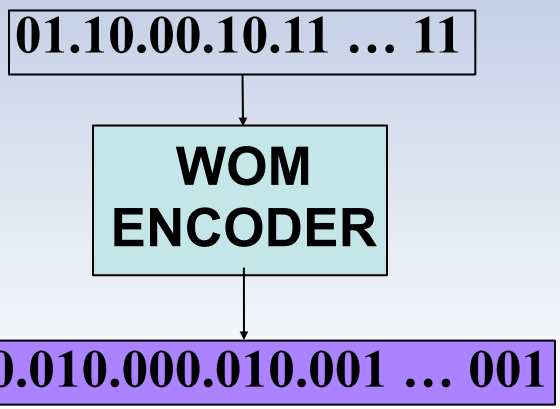


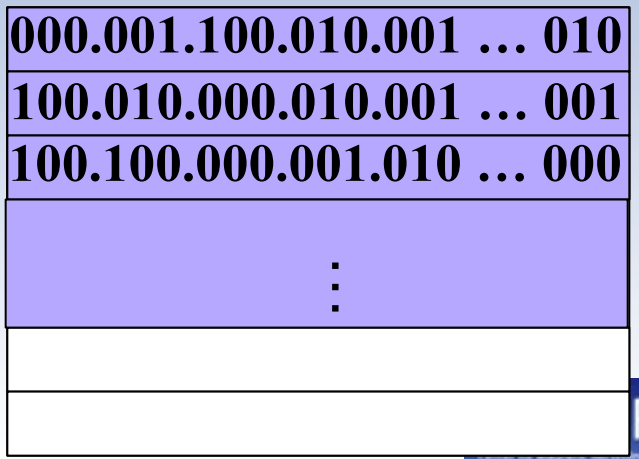
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



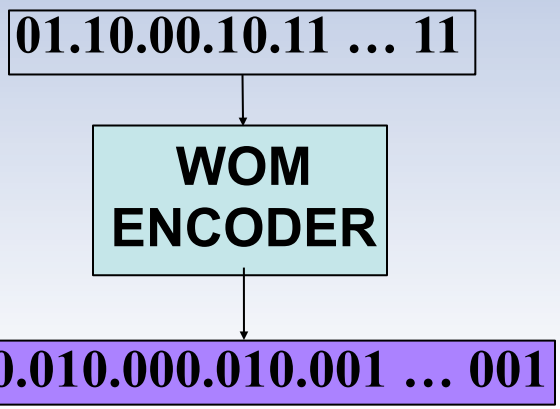


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



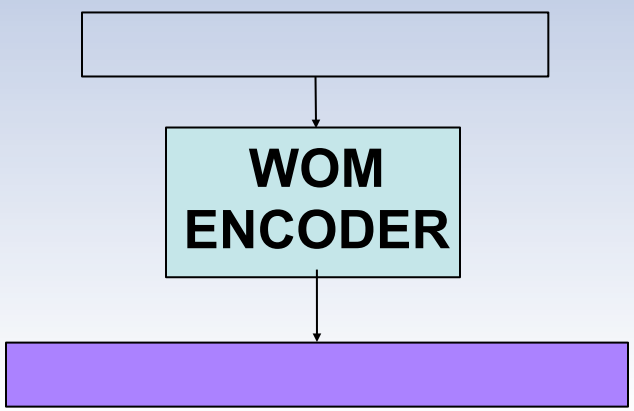
000.001.100.010.001 … 010
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



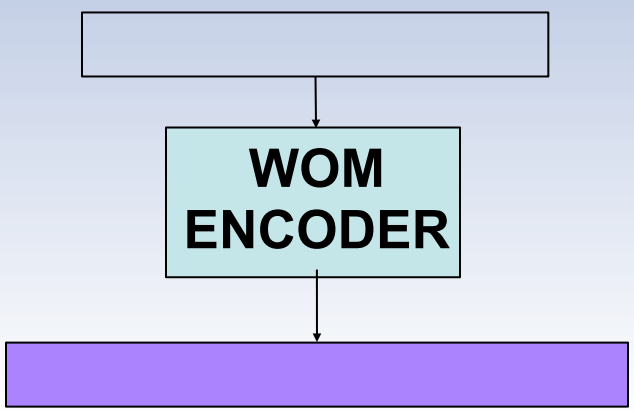
000.001.100.010.001 … 010
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



000.001.100.010.001 … 010
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

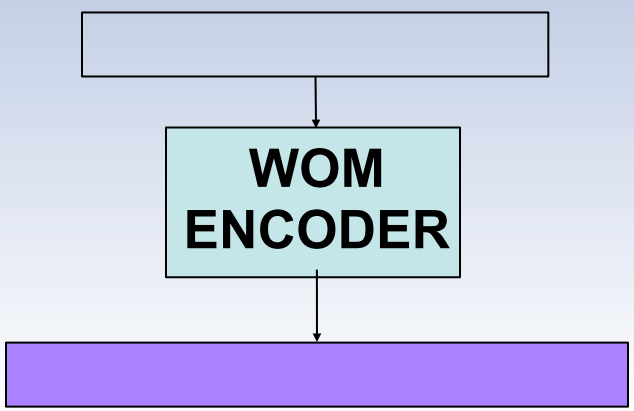
INVALID

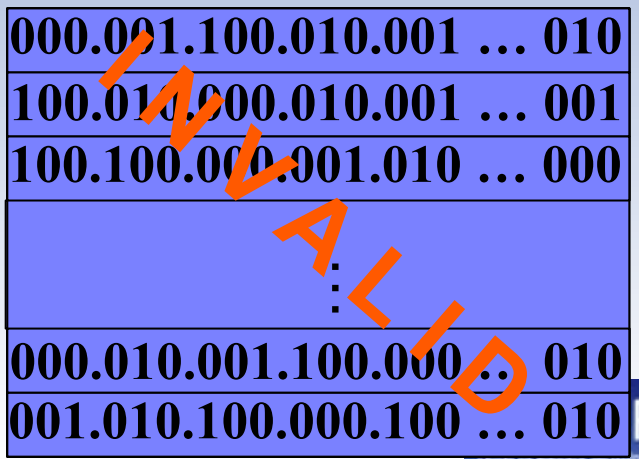
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



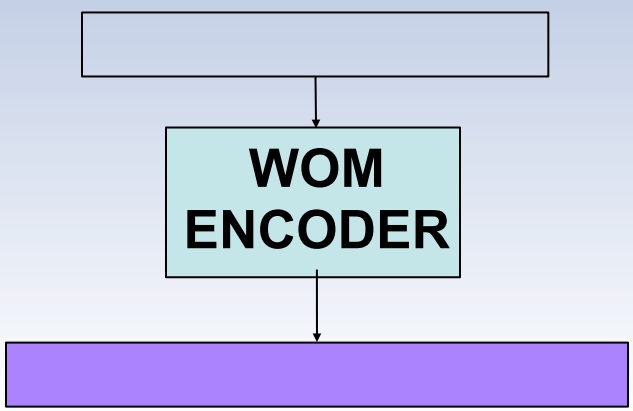


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



000.001.100.010.001 … 010
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

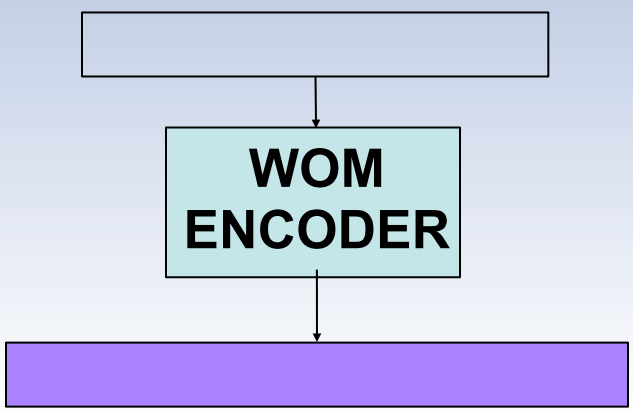
INVALID

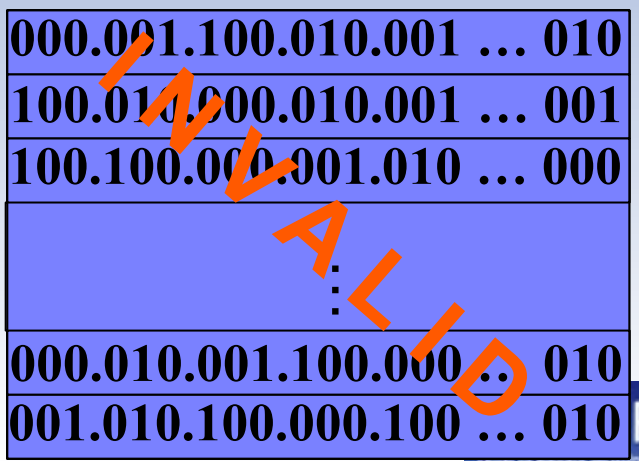
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



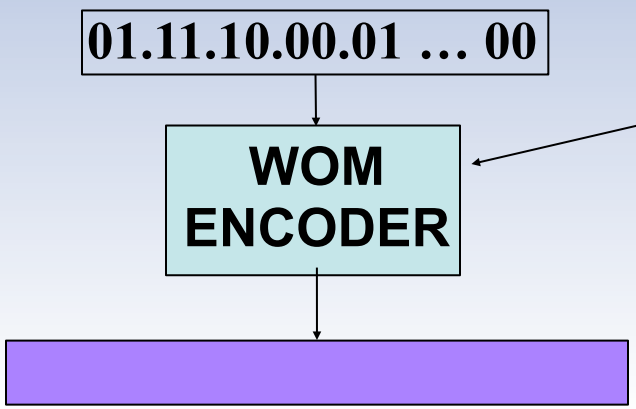


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

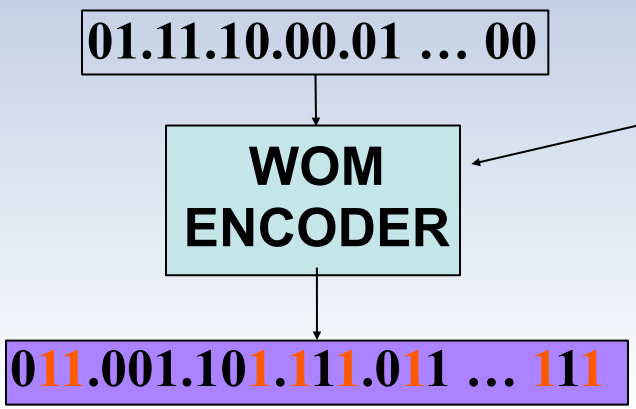


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

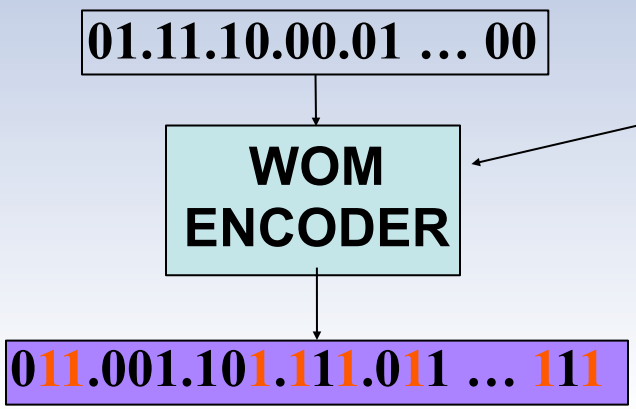


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

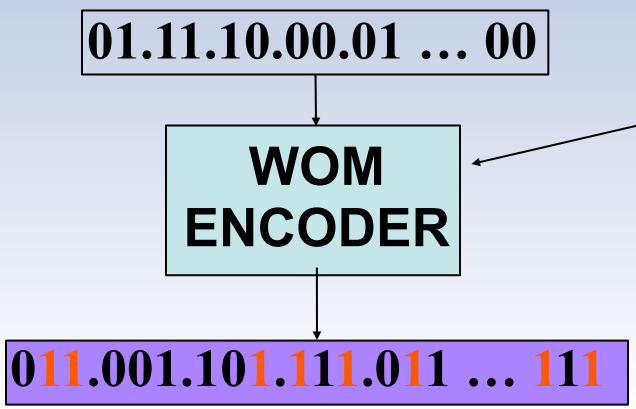


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

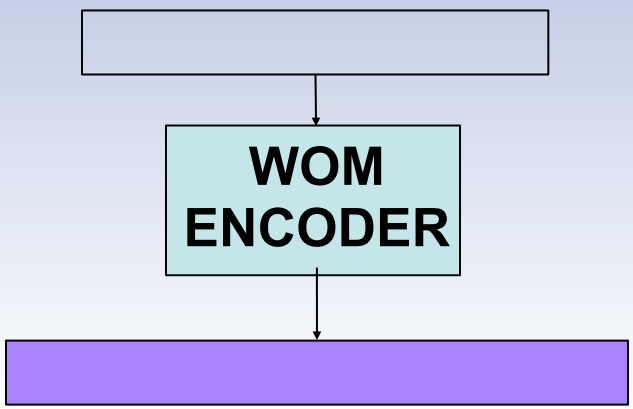


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



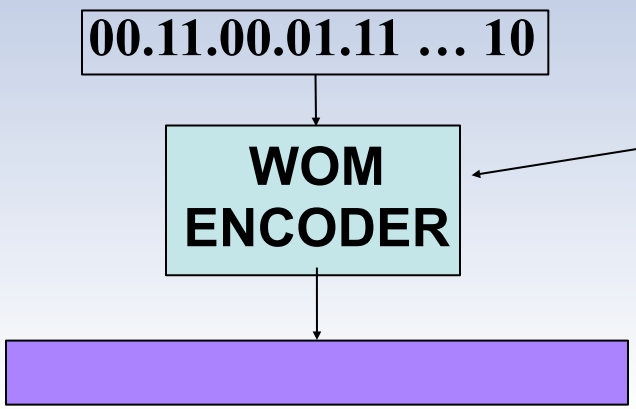
011.001.101.111.011 … 111
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



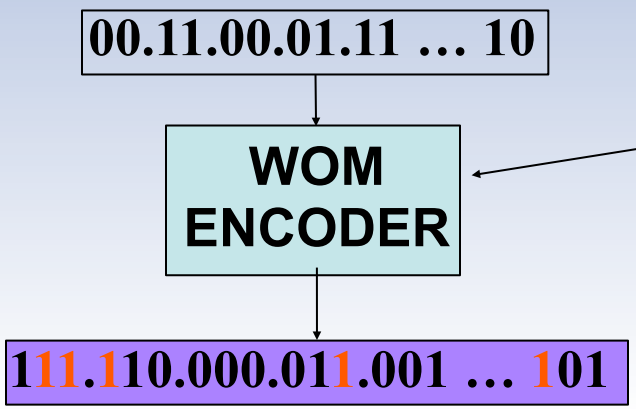
011.001.101.111.011 … 111
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



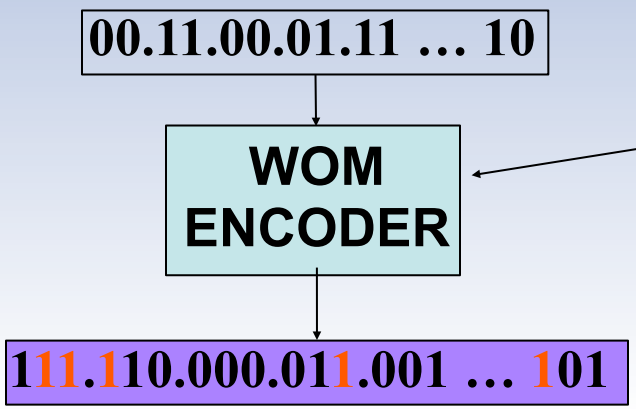
011.001.101.111.011 … 111
100.010.000.010.001 … 001
100.100.000.001.010 … 000
⋮
000.010.001.100.000 … 010
001.010.100.000.100 … 010

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

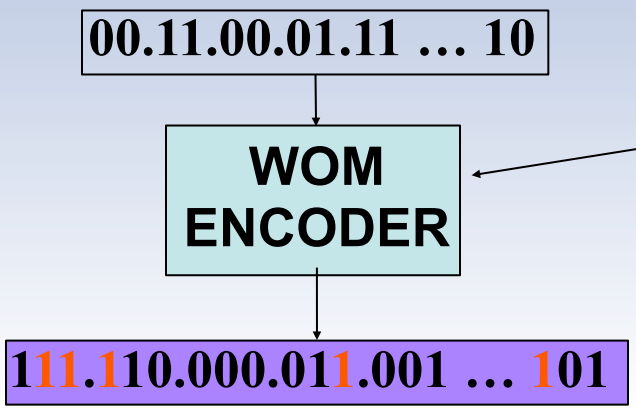


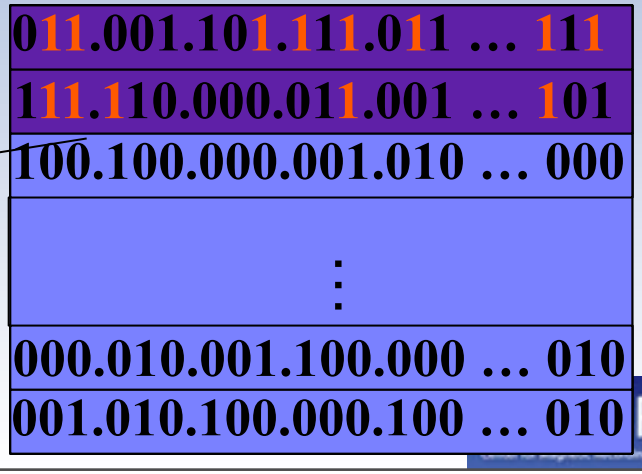
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



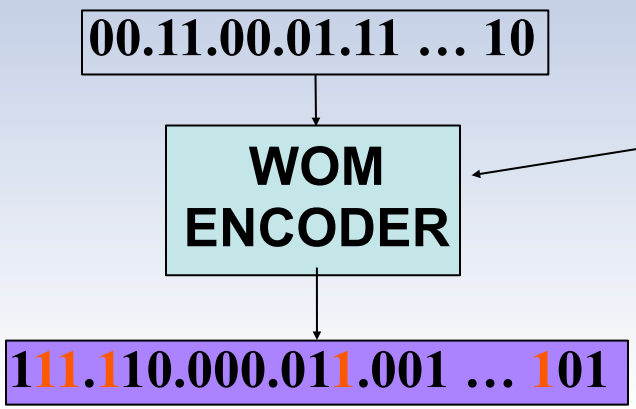


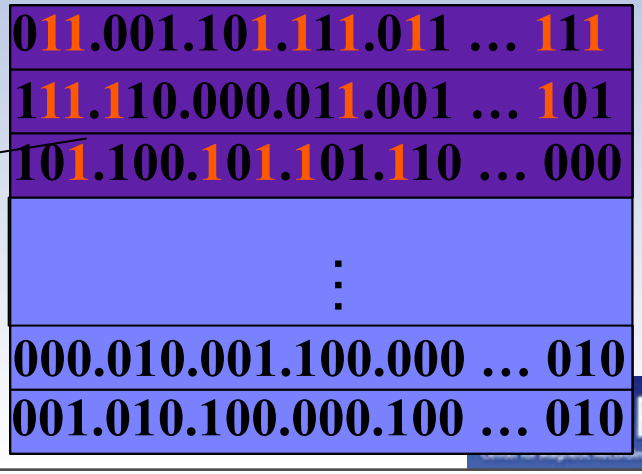
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



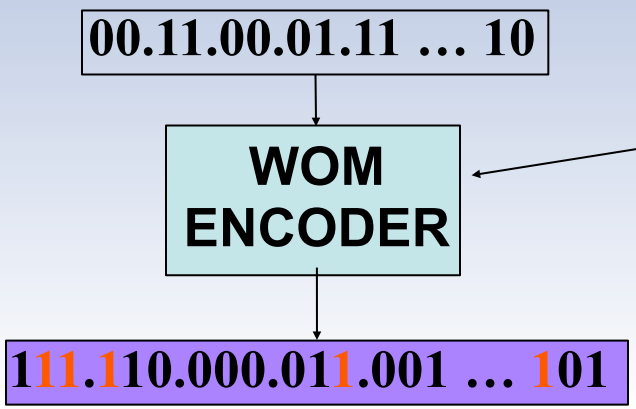


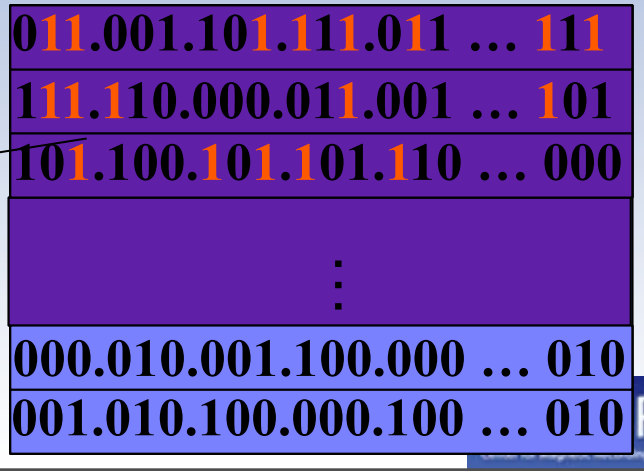
Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



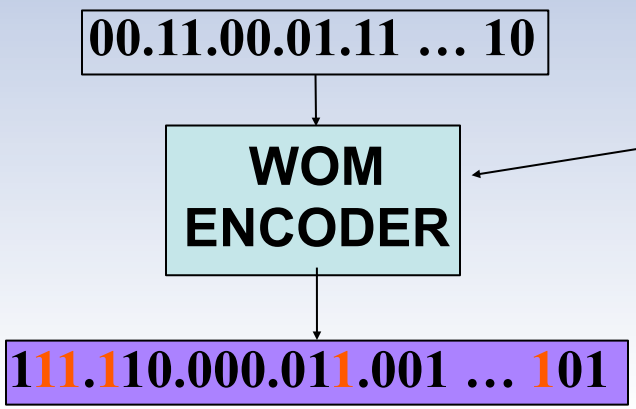


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

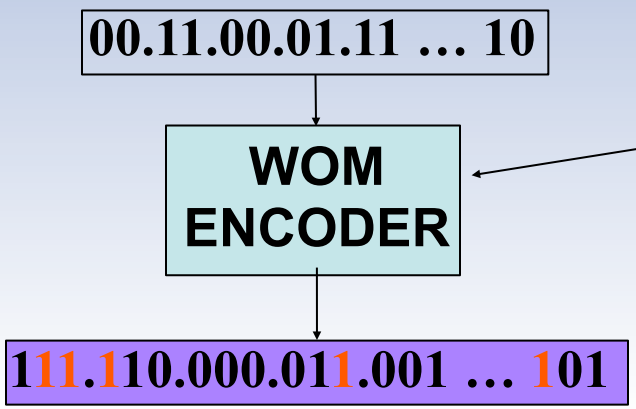


Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state



Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

011.001.101.111.011 … 111
111.110.000.011.001 … 101
101.100.101.101.110 … 000
⋮
000.110.111.111.110 … 010
111.110.100.101.101 … 110

Write Once Memory (WOM) Codes for SLC

- A scheme for storing two bits twice using only three cells **before erasing the cells**
- The cells only **increase** their level
- How to implement? (in **SLC** block)
  - Each page stores **2KB/1.5 = 4/3KB** per write
  - A page can be written twice before erasing
  - Pages are **encoded** using the **WOM code**
  - When the block has to be rewritten, mark its pages as **invalid**
  - Again write pages using the WOM code **without erasing**
  - **Read before write** at the second write

Advantages:

• The number of bits written per cell is **4/3**

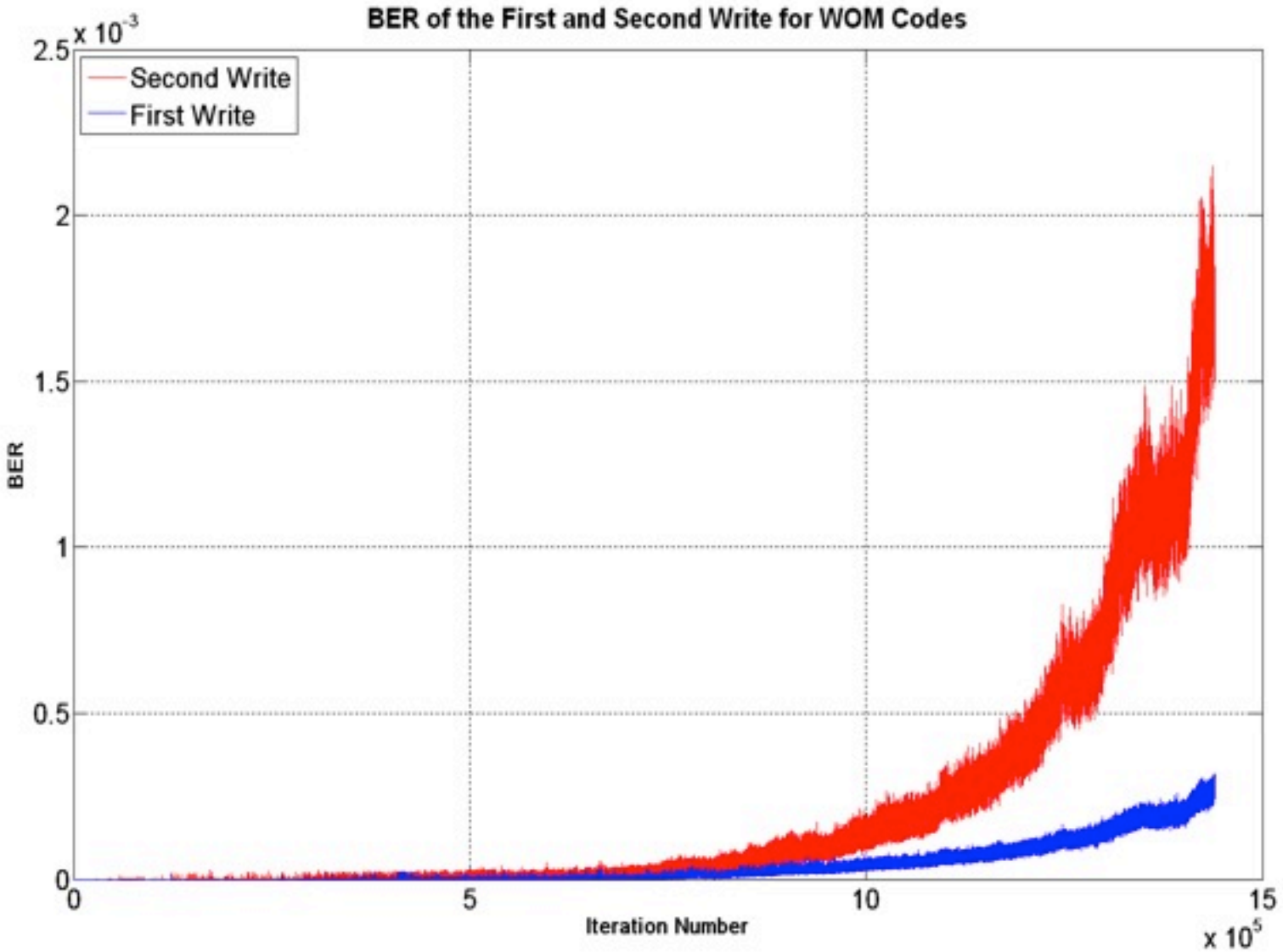
• Possible to write twice before a physical erasure

data	1st write	2nd write
00	000	111
01	100	011
10	010	101
11	001	110

Cells state

011.001.101.111.011 … 111
111.110.000.011.001 … 101
101.100.101.101.110 … 000
⋮
000.110.111.111.110 … 010
111.110.100.101.101 … 110

BER for the First and Second Writes



WOM-Codes with two writes

WOM-Codes with two writes

- Assume there are n cells and two writes, $t=2$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t=2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

    The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$
- **Rivest and Shamir** constructed WOM-codes of rates (2/3, 2/3) and (0.67, 0.67), $R = 1.34$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

    The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$
- **Rivest and Shamir** constructed WOM-codes of rates (2/3, 2/3) and (0.67, 0.67), $R = 1.34$
- We construct WOM-codes from any linear code:

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

    The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$
- **Rivest and Shamir** constructed WOM-codes of rates $(2/3, 2/3)$ and $(0.67, 0.67)$, $R = 1.34$
- We construct WOM-codes from any linear code:
  - The $[23,12,7]$ Golay code: $(0.9458, 0.5217)$ $R = 1.4632$

WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

    The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$
- **Rivest and Shamir** constructed WOM-codes of rates $(2/3, 2/3)$ and $(0.67, 0.67)$, $R = 1.34$
- We construct WOM-codes from any linear code:
  - The $[23,12,7]$ Golay code: $(0.9458, 0.5217)$ $R = 1.4632$
  - The $[16,11,4]$ extended Hamming code $(0.769, 0.6875)$, $R = 1.456$ and for the same rate we get $(0.6875, 0.6875)$, $R = 1.375$

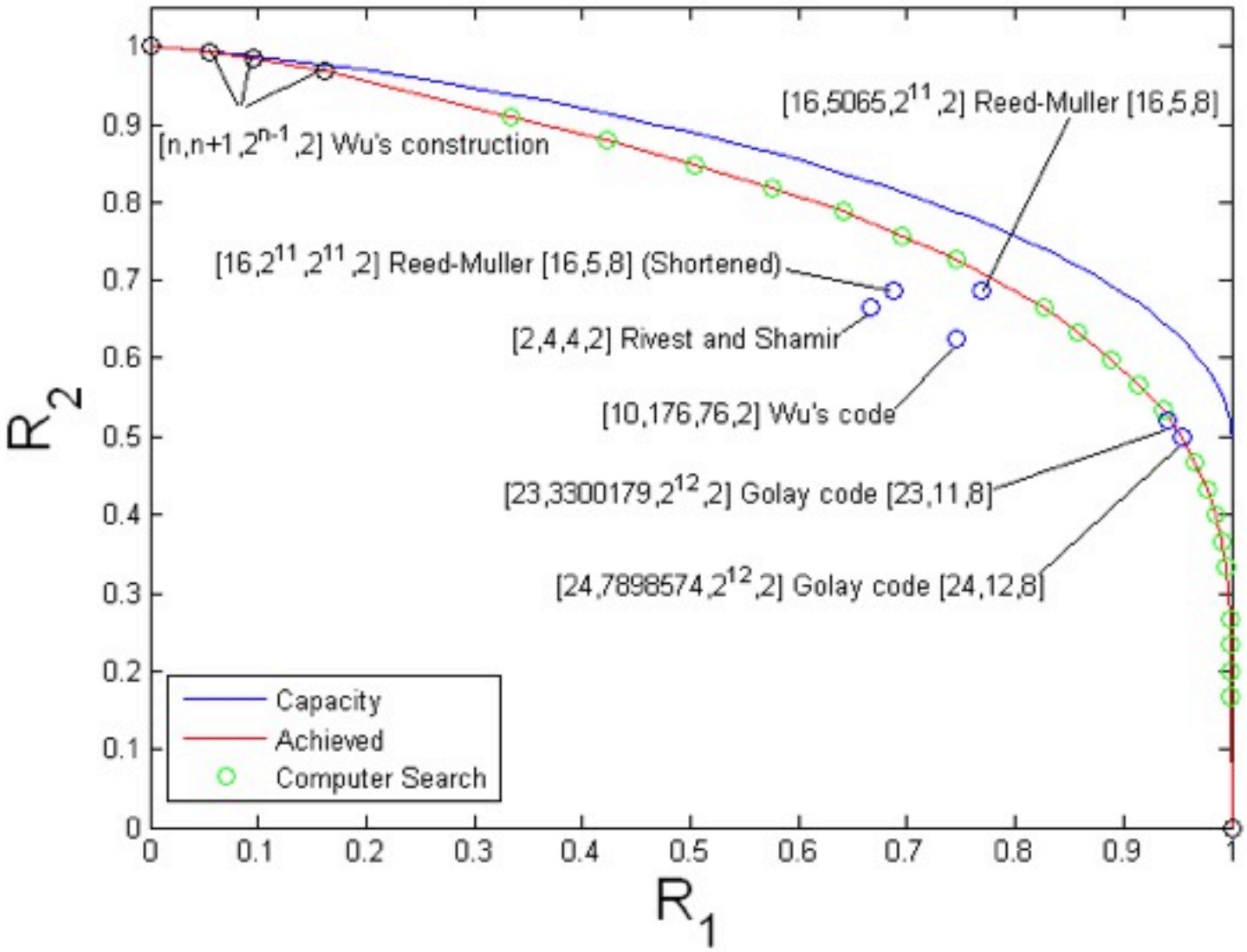
WOM-Codes with two writes

- Assume there are n cells and two writes, $t = 2$
  - First write: k_1 bits, $R_1 = k_1/n$, second write: k_2 bits, $R_2 = k_2/n$
  - Capacity region (**Heegard** 1986, **Fu and Han Vinck** 1999)

$$C = \{ (R_1, R_2) \mid \$p \in [0, 0.5], R_1 \leq h(p), R_2 \leq 1 - p \}$$

    The WOM-rate $R = R_1 + R_2 \leq \log_2(3) \approx 1.58$, achieved for $p = 1/3$
- **Rivest and Shamir** constructed WOM-codes of rates $(2/3, 2/3)$ and $(0.67, 0.67)$, $R = 1.34$
- We construct WOM-codes from any linear code:
  - The $[23,12,7]$ Golay code: $(0.9458, 0.5217)$ $R = 1.4632$
  - The $[16,11,4]$ extended Hamming code $(0.769, 0.6875)$, $R = 1.456$ and for the same rate we get $(0.6875, 0.6875)$, $R = 1.375$
  - By computer search we found rate $(0.7273, 0.7273)$, $R = 1.4546$

WOM Capacity and Achievable Rates



Summary

Summary

- Single Bit Representation in MLC Flash

Summary

- Single Bit Representation in MLC Flash
- New ECC Scheme for MLC Flash

Summary

- Single Bit Representation in MLC Flash
- New ECC Scheme for MLC Flash
- WOM-Codes

Summary

- Single Bit Representation in MLC Flash
- New ECC Scheme for MLC Flash
- WOM-Codes
- More analysis of codes and error behavior -

COME TO BOOTH #510!