Błędy pamięci i ich korekcja

Rafał Łasocha

Błędy

- mają znaczenie w zależności od systemu
  - zły bit w systemie bankowym
  - zły bit w zdjęciu z wakacji
  - jest wiele poziomów, na których może pojawić się błąd: HDD, DRAM, cache, rejestry
- typy błędów
  - soft failures
    - cząsteczki alfa
    - wtórne promieniowanie kosmiczne
  - hard failures

Cząsteczki alfa

- fakt że cząsteczki alpha będą wpływać na pamięć DRAM był znany od początku
- <coolstory>
  - w 1978 Intel produkował pamięci które miały niespodziewanie dużo błędów
  - okazało się, że kupowali część materiałów od firmy położonej na starej kopalni uranu
  - od tamtej pory już ostrożnie wybierają skąd kupują części
- </coolstory>
- od tamtej pory już wybierają ostrożnie skąd kupują części i cząsteczki alpha są względnie mało prawdopodobnym powodem błędów w kościach DRAM

Wtórne promieniowanie kosmiczne

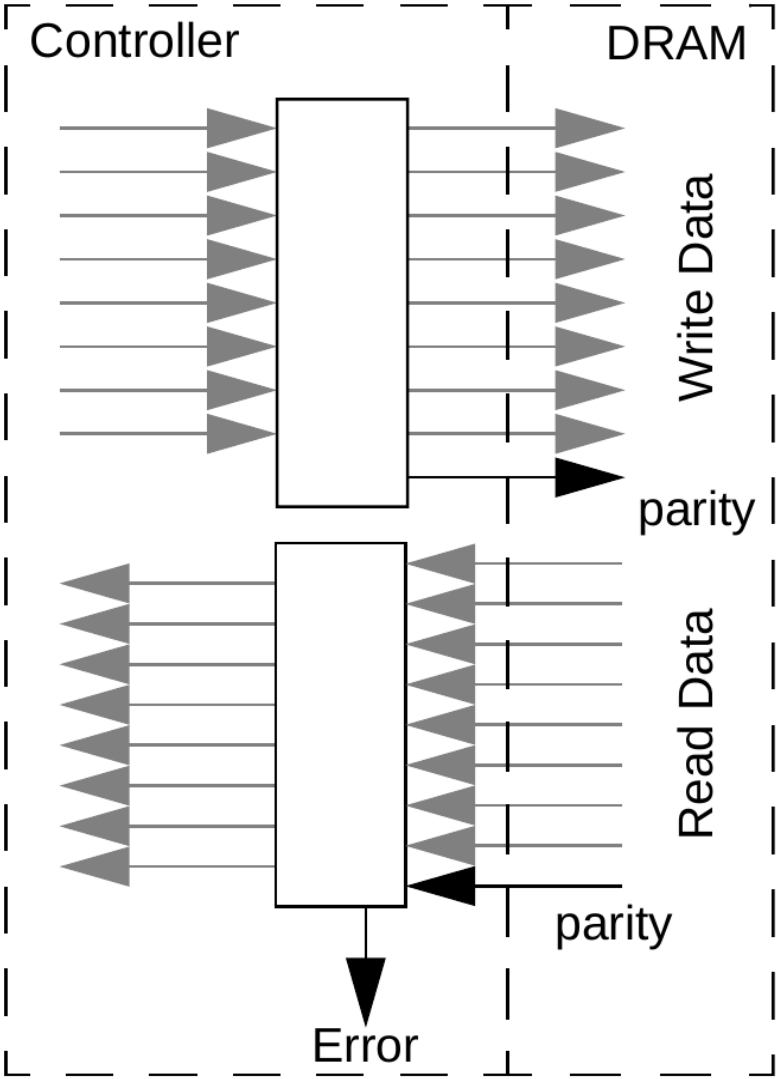
- promieniowanie kosmiczne → zderzenie z atmosferą → wysokoenergetyczne cząsteczki
- liczba tych cząsteczek zależy od wysokości bezwzględnej
- w momencie pisania książki „*Memory Systems: Cache, DRAM, Disk*” były głównym źródłem błędów (soft failures)
- SER (soft error rates) proporcjonalne do liczby cząsteczek w danym miejscu

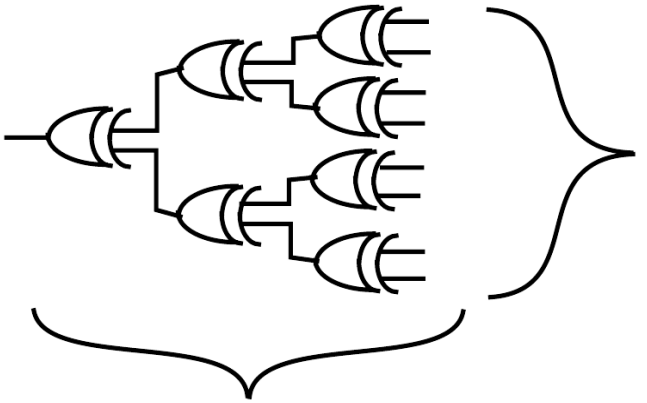
Błędy jednobitowe/wielobitowe

TABLE 30.1 Frequency of multi-bit and single-bit errors

DRAM Size-Vendor	x4	x8	x16
4 MB-A	7%	7%	—
4 MB-B	16%	16%	24%
16 MB-C	13%	19%	19%
64 MB-D	3%	12%	12%
64 MB-E	13%	18%	18%

Kontrola parzystości





data width
N bit wide

logic depth $= \log_2 N$

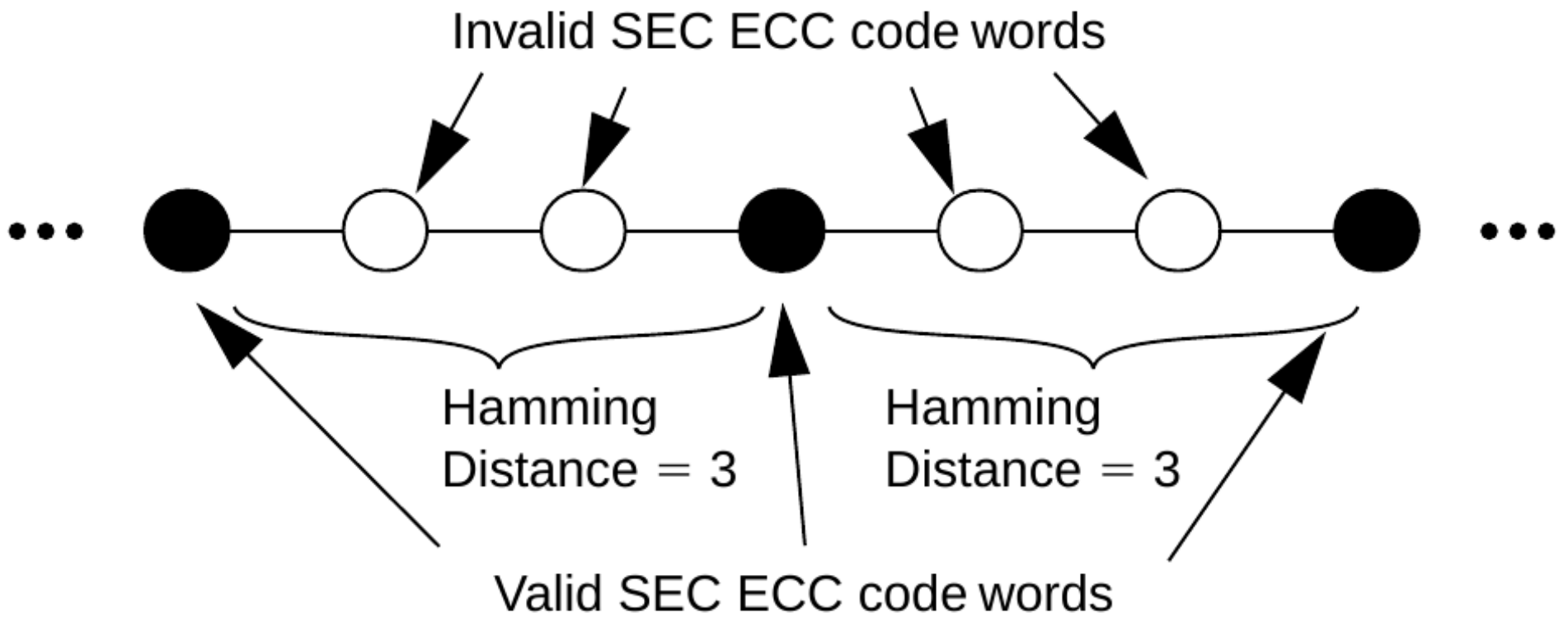
Kontrola parzystości

- zaleta – znajduje, że w słowie jest błąd
- wada – nie znajduje gdzie

SEC ECC

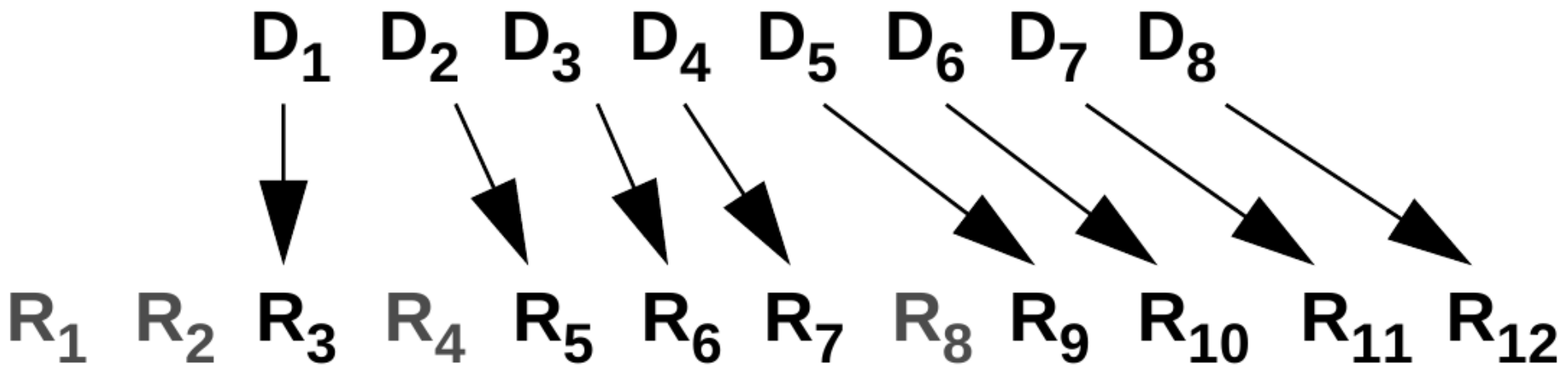
- Single-Bit Error Correction ECC
- korzystamy z odległości Hamminga pomiędzy bajtami
- rozszerzamy nasze słowa do takiej długości, żeby pomiędzy każdymi dwoma poprawnymi słowami, były dokładnie dwa błędne

SEC ECC


Invalid SEC ECC code words
Hamming
Distance = 3
Hamming
Distance = 3
Valid SEC ECC code words

SEC ECC

start with 8 data bits
(do not use D_0)



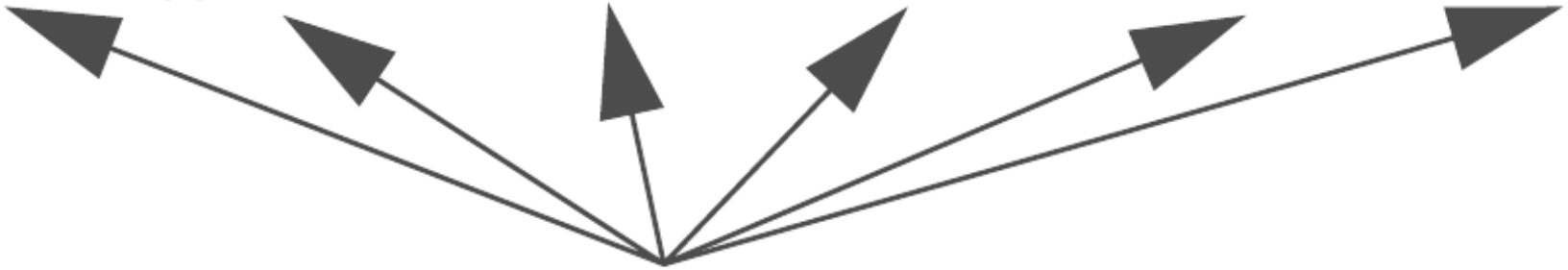
Reserve R_m bit positions where m is a power of 2.

Move data bits into available bit positions. (skip R_0)

SEC ECC

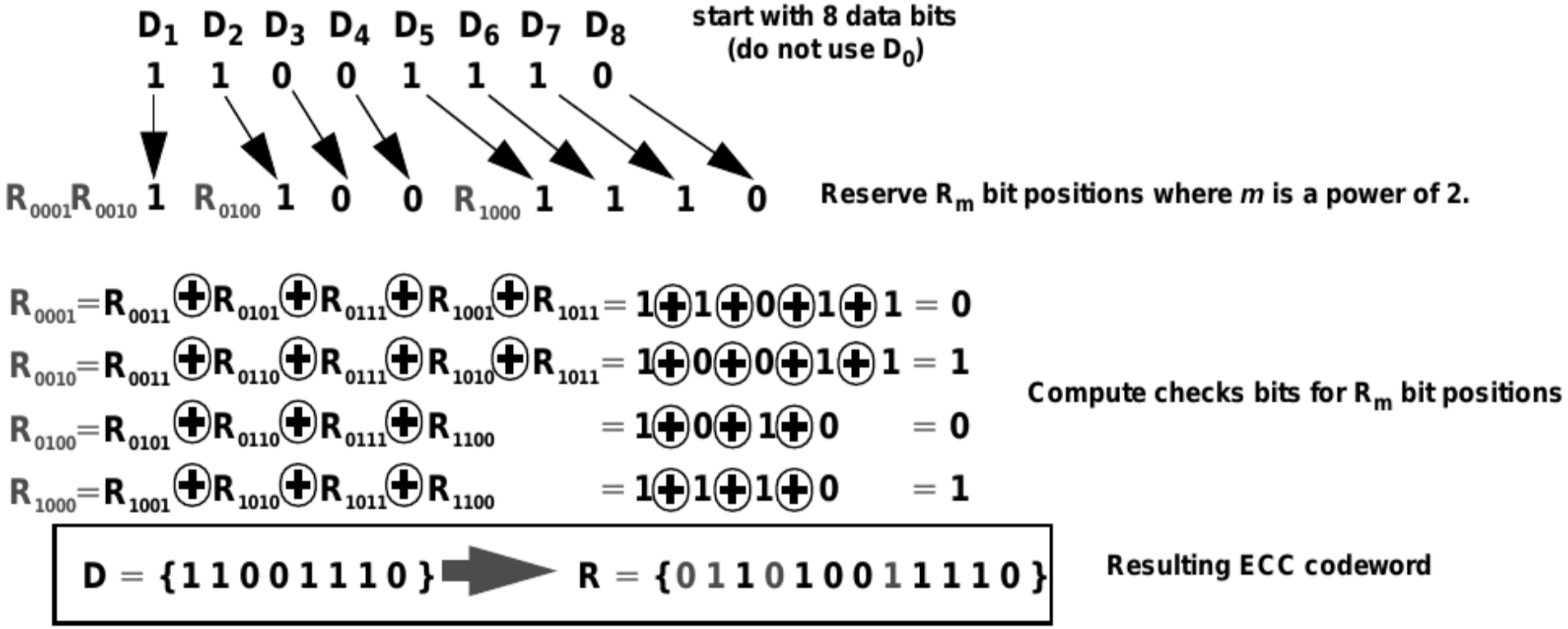
$$R_1 = R_3 \oplus R_5 \oplus R_7 \oplus R_9 \oplus R_{11}$$

$$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011}$$



Compute R_m from the data vector.

SEC ECC - przykład



SEC ECC - przykład

$R = \{0\,1\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\}$
$R = \{0\,1\,1\,0\,1\,0\,0\,1\,1\,1\,0\,0\}$

One bit error. Can it be detected and corrected?

Recompute check bits

$$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$$
$$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$
$$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$
$$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

XOR old check bits against new check bits

	R_{1000}	R_{0100}	R_{0010}	R_{0001}	
	1	0	1	0	Old
$\oplus$	0	0	0	1	New
	1	0	1	1	ECC Syndrome $= 1011_2 = 11_{10}$

Syndrome != 0000

Bit position 11 is suspect

SEC ECC – przykład 2

$R = \{ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \}$

$R = \{ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \}$

Two bit error. Can we detect and correct?

Recompute check bits

$$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$$

$$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 0$$

$$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

XOR old check bits against new check bits

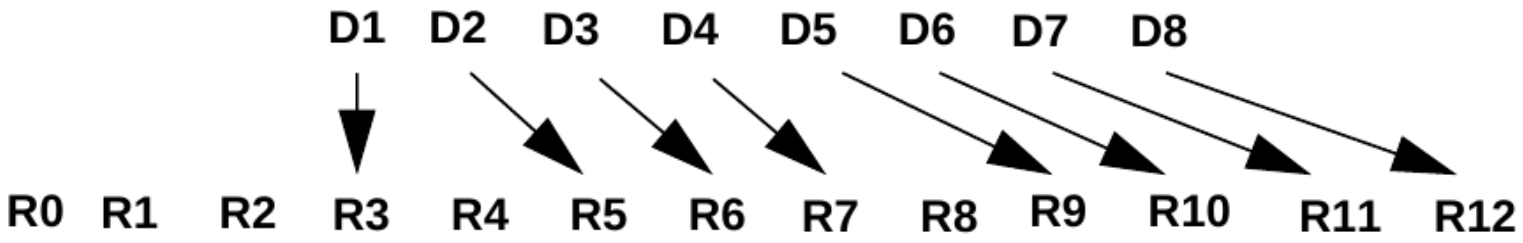
	R_{1000}	R_{0100}	R_{0010}	R_{0001}	
	1	0	1	0	Old
$\oplus$	1	0	0	1	New
	0	0	1	1	Suggests bit 3 is corrupt

SECDED ECC

- Single-Bit Error Correction Double Error Detection ECC

SECDED ECC

- Single-Bit Error Correction Double Error Detection ECC
- SEC + Parzystość = SEC DED



$R0 = D1 \oplus D2 \oplus \bullet\bullet\bullet \oplus Dn$

Basic Idea: Use R_0 to check parity of original bit vector

Bossen's b-adjacent Algorithm

- zabezpiecza pary bitów obok siebie
- taki sam narzut na szerokość słowa

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Two-bit vectors in 2-adjacent code.

Bossen's b-adjacent Algorithm

$$\begin{bmatrix} C_{\alpha^2} \\ C_{\alpha} \\ C_1 \\ C_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \alpha^2 & \alpha & 1 & 1 & 1 & \alpha^2 & \alpha & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha & 1 \\ \alpha^2 & \alpha & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \alpha^2 & \alpha & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & \alpha^2 & \alpha & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & \alpha & \alpha^2 & 1 & 1 & 1 & 1 & \alpha^2 & \alpha & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \alpha^2 & \alpha & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha & 0 & 0 & 1 & 1 & 1 & 1 & \alpha^2 & \alpha & 1 & 0 & 1 & 1 & \alpha^2 & \alpha & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Parity check matrix in GF(2^2)

Apply transform matrices

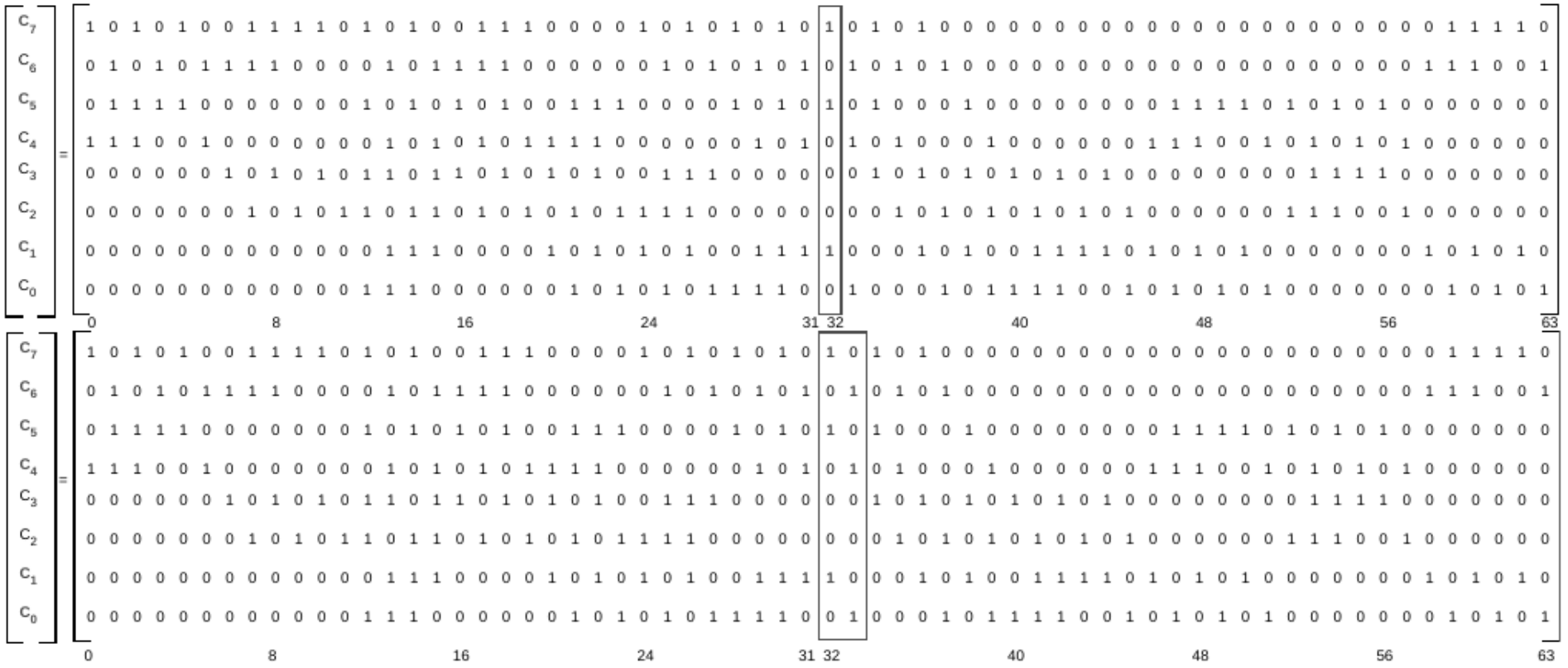
$$T_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad T_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad T_{\alpha} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad T_{\alpha^2} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Parity check matrix in binary field

$$\begin{bmatrix} C_7 \\ C_6 \\ C_5 \\ C_4 \\ C_3 \\ C_2 \\ C_1 \\ C_0 \end{bmatrix} = \begin{bmatrix} 1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0 \\ 0\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,1 \\ 0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\ 1\,1\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\,0\,0\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,1\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1 \end{bmatrix}$$

0 8 16 24 31 32 40 48 56 63

Bossen's b-adjacent Algorithm



Bossen's b-adjacent Algorithm

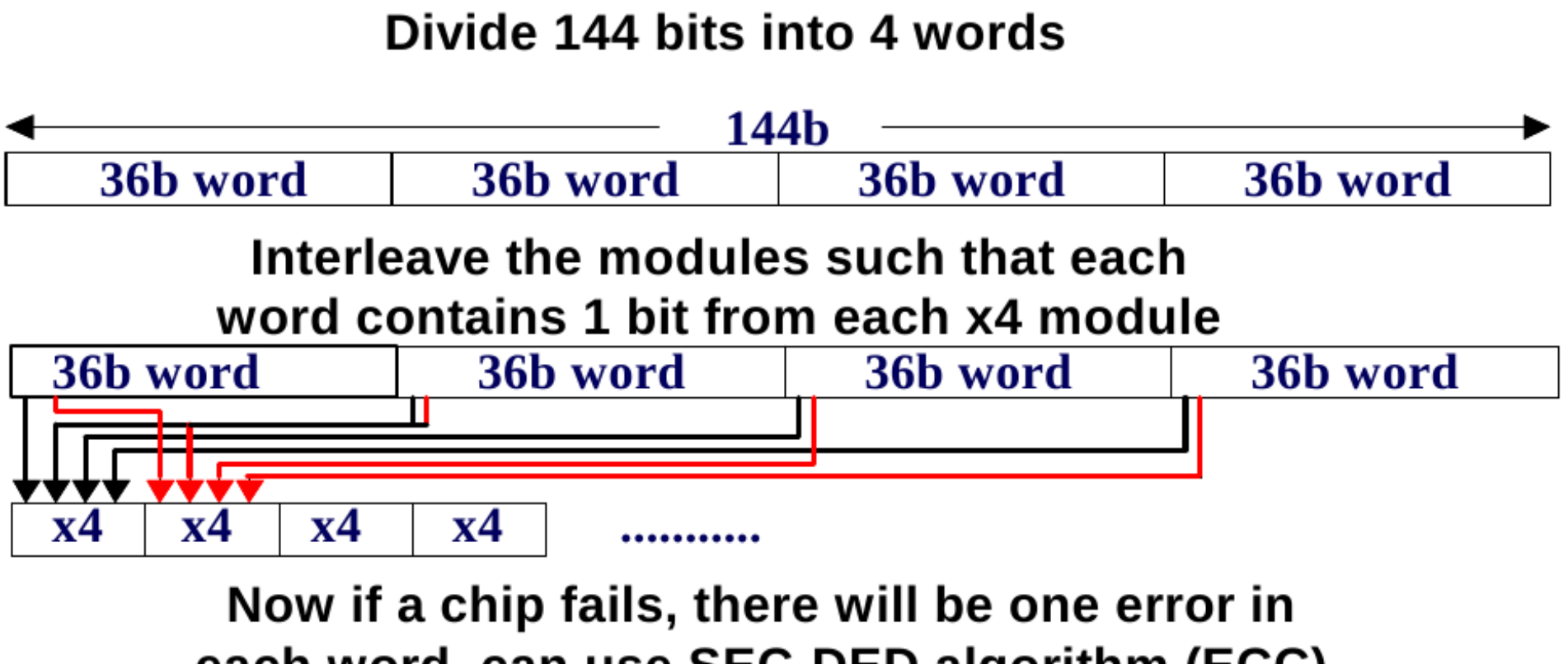
TABLE 30.3 Error location table for the 2-adjacent error correction algorithm, taken from US Patent #5,490,155 (Compaq's Advanced ECC implementation)

				S7:	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				S6:	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
s	s	s	s	s5:	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
3	2	1	0	s4:	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0			C4	C5		C6	5	3	1	C7		4	2		2,3	0.1	4,5
0	0	0	1		C0	51	49	47	63	33			61		28		59			30,31
0	0	1	0		C1	46	50	48	58	31			62		32		60			28,29
0	0	1	1			48,49	46,47	50,51	60,61	29			58,59				62,63			32,33
0	1	0	0		C2	57	52	54,55	11	35			9	19			7	17		
0	1	0	1		45	39	23	21	37											
0	1	1	0		43								24							12,13
0	1	1	1		41										14		26,27			
1	0	0	0		C3	55	56	52,53	6		16		10		34		8		18	
1	0	0	1		40				27											14,15
1	0	1	0		44	20	38	22					36							
1	0	1	1		42												24,25			
1	1	0	0			53	54	56,57	8,9			18,19	6,7			16,17	10,11			34,35
1	1	0	1		42,43				25						12					
1	1	1	0		40,41					15			26							
1	1	1	1		44,45	22,23	20,21	38,39									36,37			

Chipkill Memory

- Chipkill Memory (IBM), Extended ECC (Sun), Advanced ECC (HP), Chipspare (HP)
- pamięć, która nastawia się na działanie pomimo całkowitej utraty jednego z chipów
- często korzysta z *bit steeringu*

Bit steering



Chipkill memory & Bit steering

- wymaga szerszego wejścia danych
- często są używane lepsze algorytmy niż SEC DED, w publikacji Google wspominali o kościach z korekcją typu 4-adjacent
- nieraz chipkill memory potrafi sam się przerzucić na wolny chip, jak widzi że jakiś strasznie sypie błędami

Inne sposoby

- Memory scrubbing
  - prosty sposób: kolejno przepychać wszystkie słowa przez obwód ECC
  - znacznie zwiększa pobór mocy
- „DRAM RAID”

Statystyki

- ciężko zebrać dobre informacje, bo trzeba wielu urządzeń działających przez długi czas, żeby otrzymać realne dane
- większość statystyk jest w symulowanych warunkach (np. podwyższona temperatura)

Statystyki od Google (2009)

- liczba maszyn – w dziesiątkach tysięcy
- testowane DDR1, DDR2, FBDIMM

Statystyki od Google (2009)

- ~20% CE / rok / kość
- ~1.5% UE / rok / maszyna
- nie zależy od typu
- może zależeć od producenta
- zdecydowanie zależy od konkretnego urządzenia
- liczba CE i UE (ppb) rośnie z czasem
- niekoniecznie zależy od wielkości pamięci
- raczej nie zależy od temperatury
- zdecydowanie zależy od obciążenia
- raczej większość błędów to hard errors

„Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors”

- doświadczenia na kościach głównie z 2012 / 2013 roku
- podatne było 110 ze 129 przetestowanych kości

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

a. Induces errors

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

b. Does not induce errors

Code 1. Assembly code executed on Intel/AMD machines

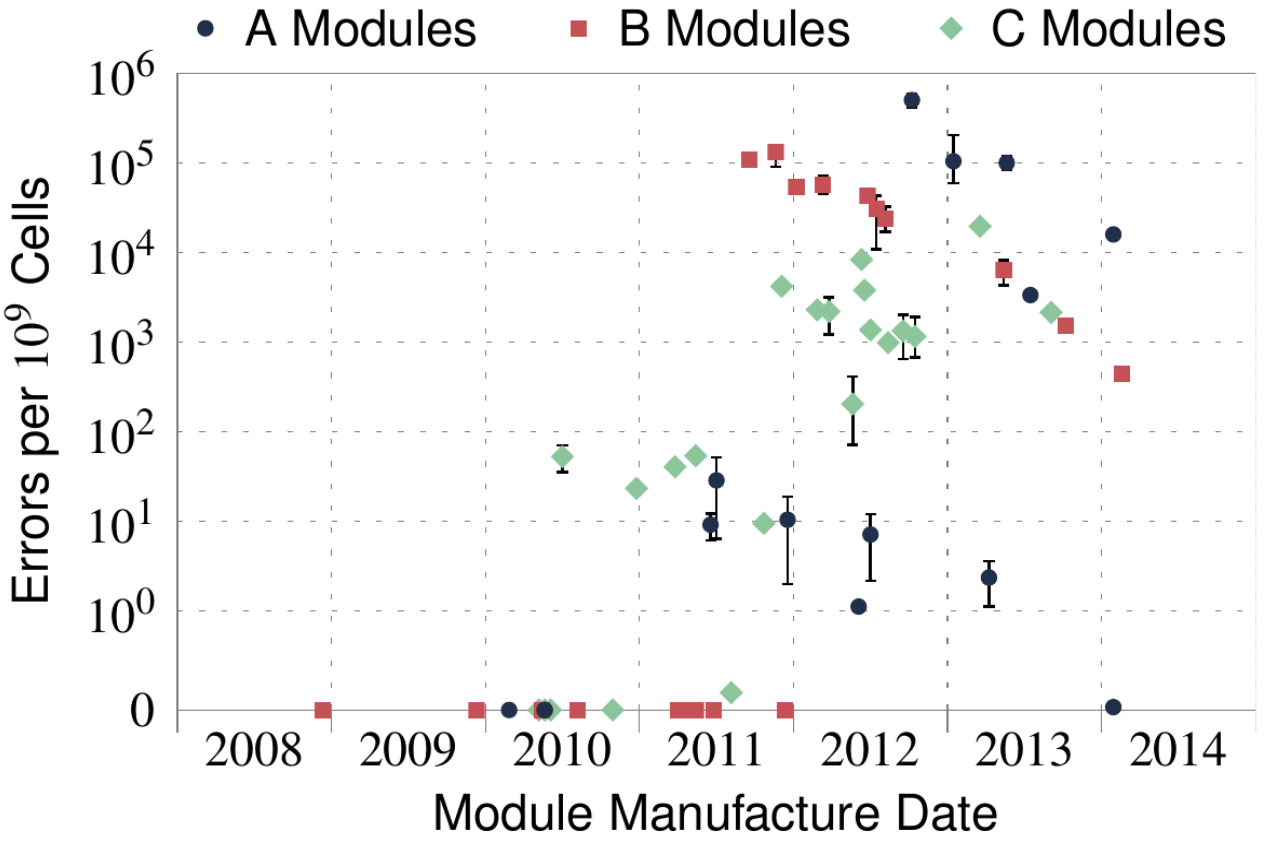


Figure 3. Normalized number of errors vs. manufacture date

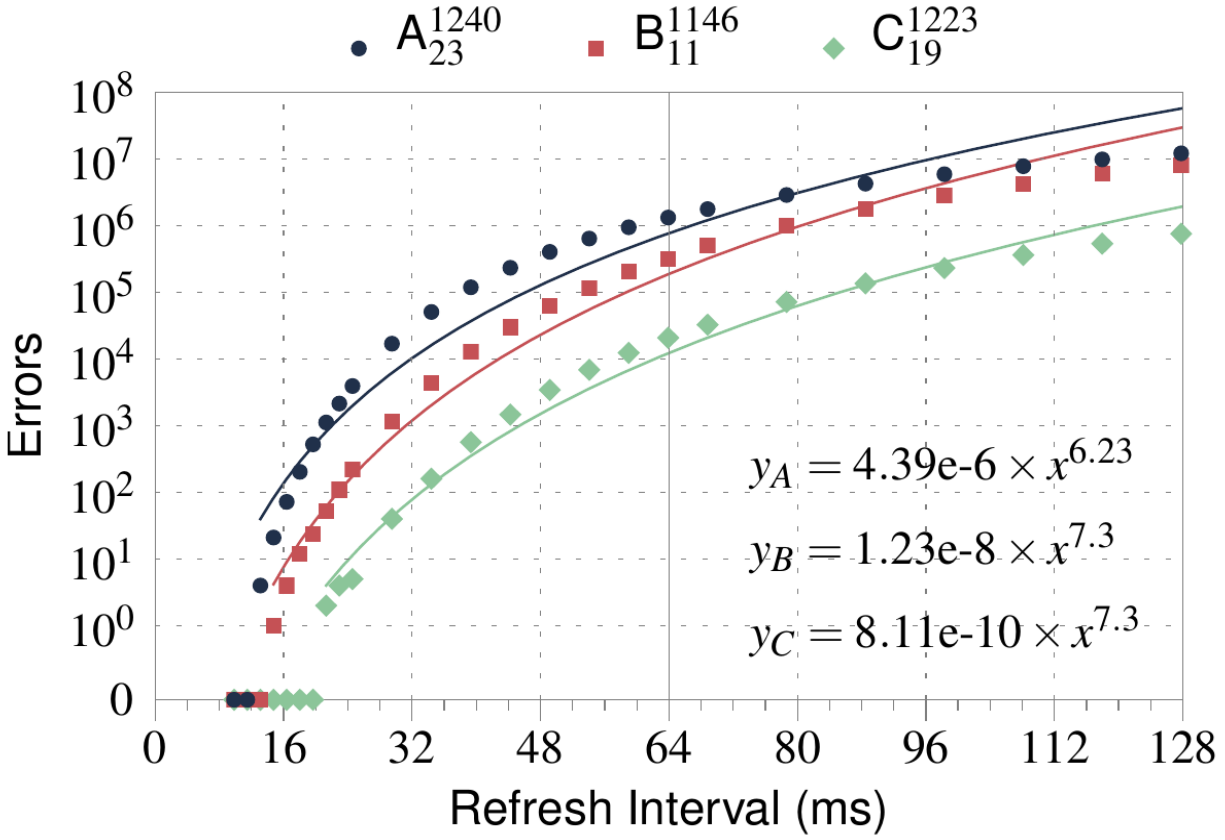


Figure 4. Number of errors as the refresh interval is varied

Potencjalne rozwiązania

1)zwiększyć częstotliwość odświeżania

2)mapowanie niedziałających komórek do zapasowych chipów

3) jeśli widzimy że jakiś wiersz jest często otwierany, odświeżmy jego sąsiadów

4) j. w., ale nie zawsze (ppb)

Źródła

- *„Memory Systems: Cache, DRAM, Disk”*
- http://www.intel.com/content/www/us/en/chipsets/e7500-chipset-mch-x4-single-device-data-correction-note.html
- http://static.googleusercontent.com/media/research.google.com/pl//pubs/archive/35162.pdf
- *„Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors”*