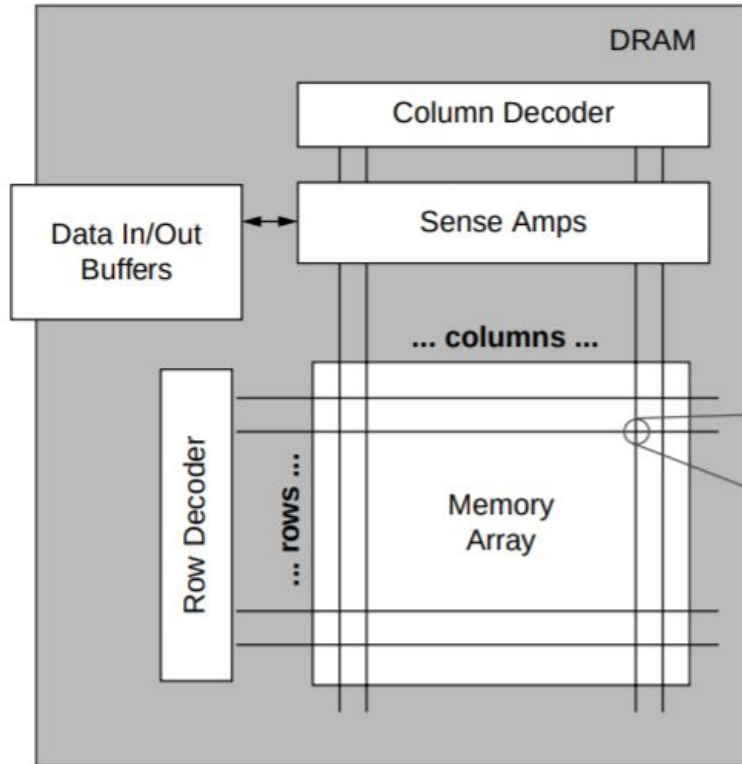


Flipping Bits in Memory Without Accessing Them

An Experimental Study of DRAM Disturbance Errors

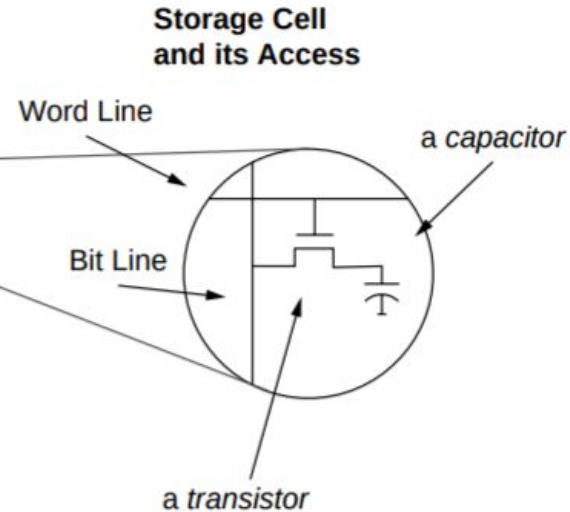
Onur Mutlu et al.

Basic DRAM overview

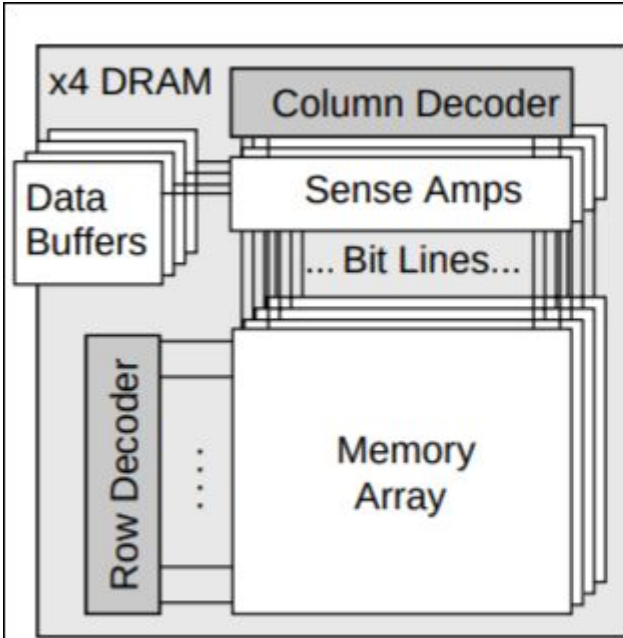


Row and column identify single bit.

Sense Amps also called *row buffers*.

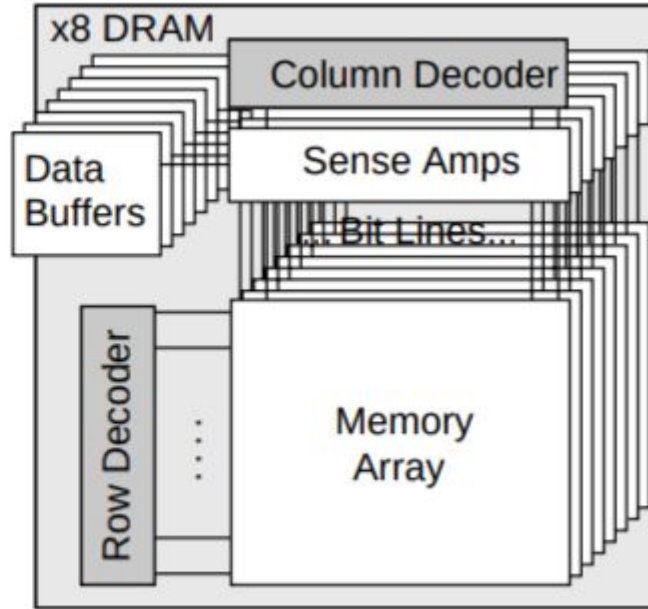


Multiple memory arrays - x4 DRAM, x8 DRAM



x4 DRAM

single bank



x8 DRAM

single bank

DRAM outputs as many bits as it has arrays.

Shared **column decoder** and **row decoder**.

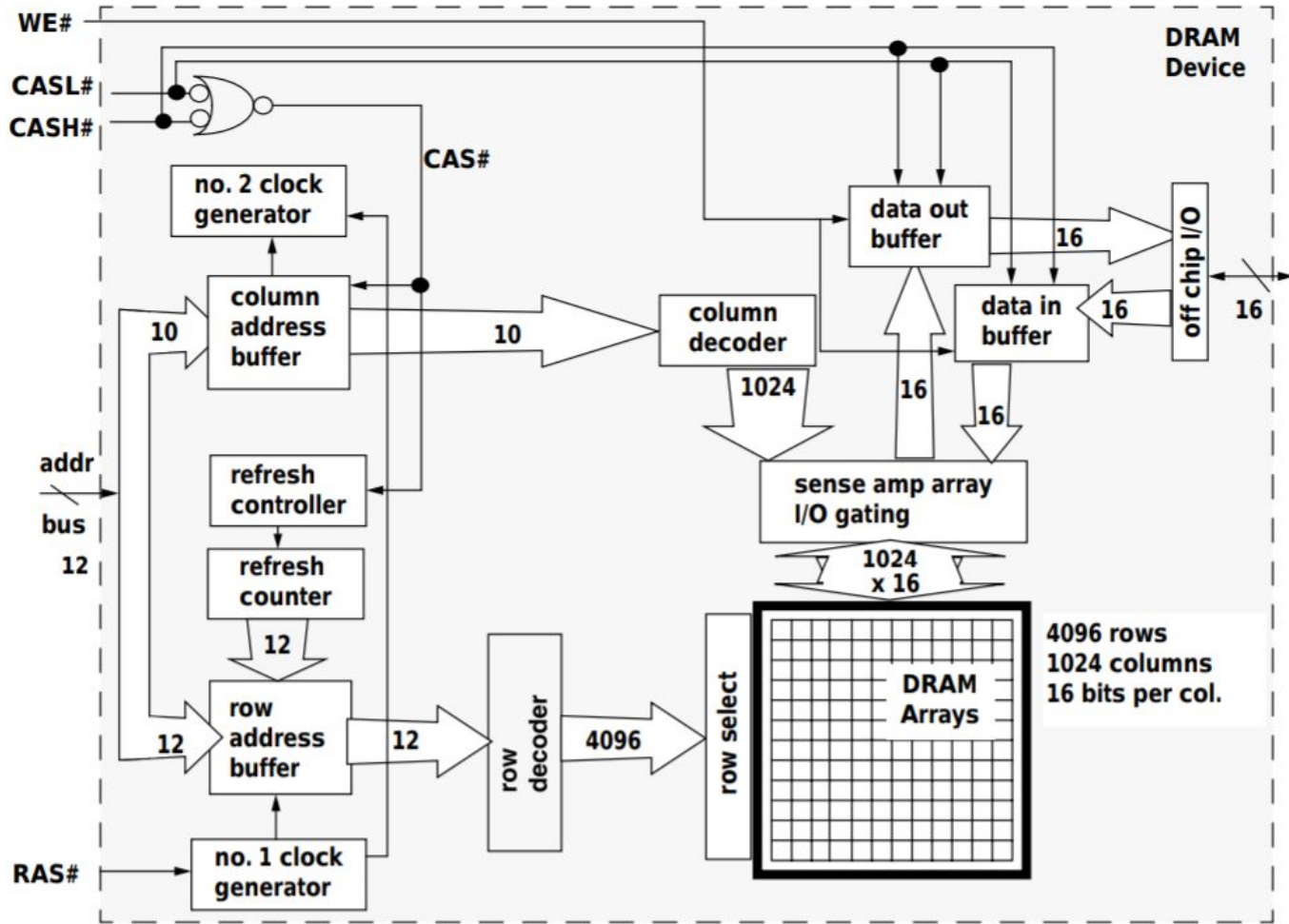


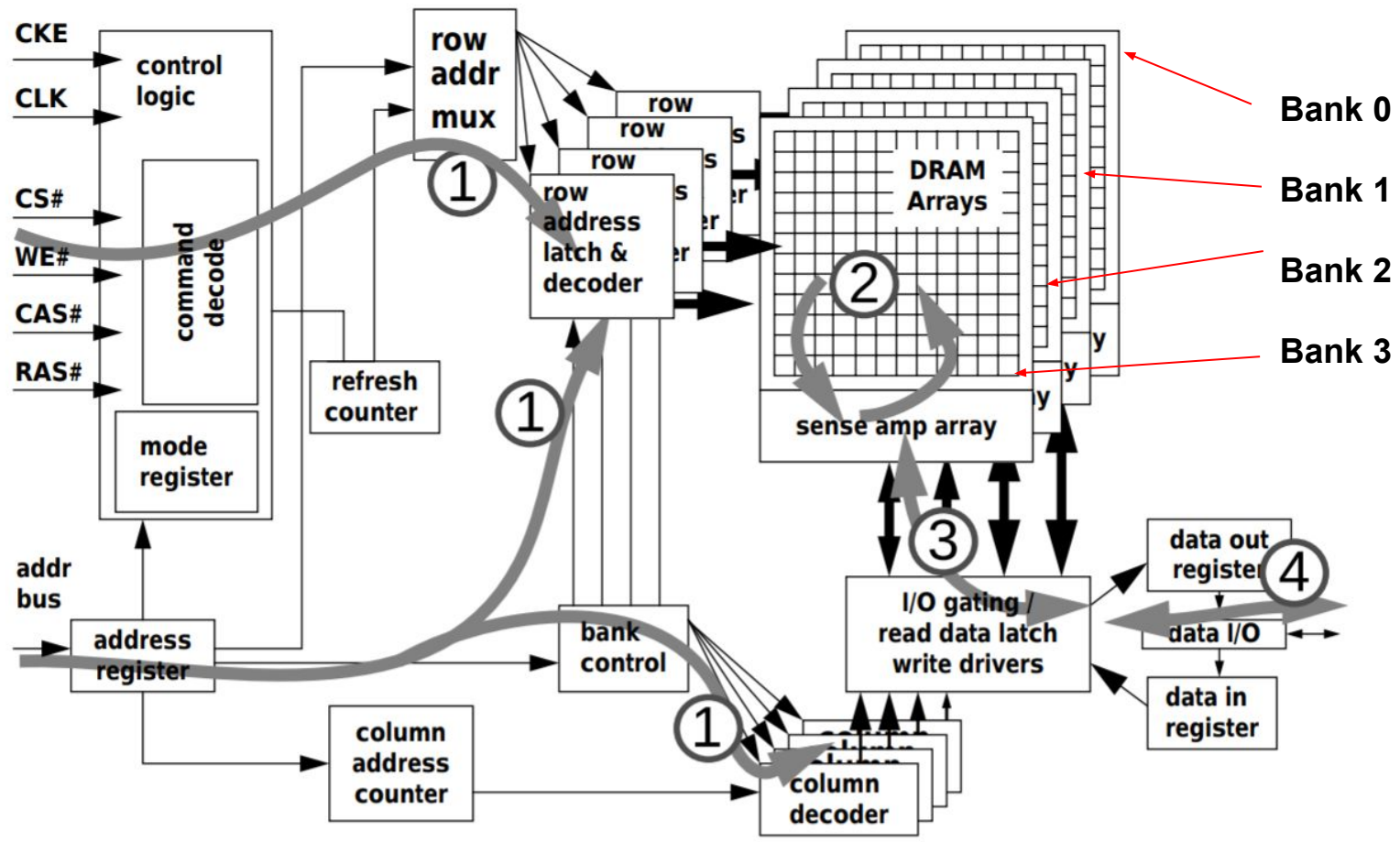
FIGURE 8.1: A 64-Mbit Fast Page Mode DRAM device (4096 x 1024 x 16). One bank

Bank - independent group of memory arrays

- DRAM chips on previous slide had multiple arrays but single bank
- Banks can be activated, precharged, read etc. at the same time as other banks*. Share command, address, data busses.

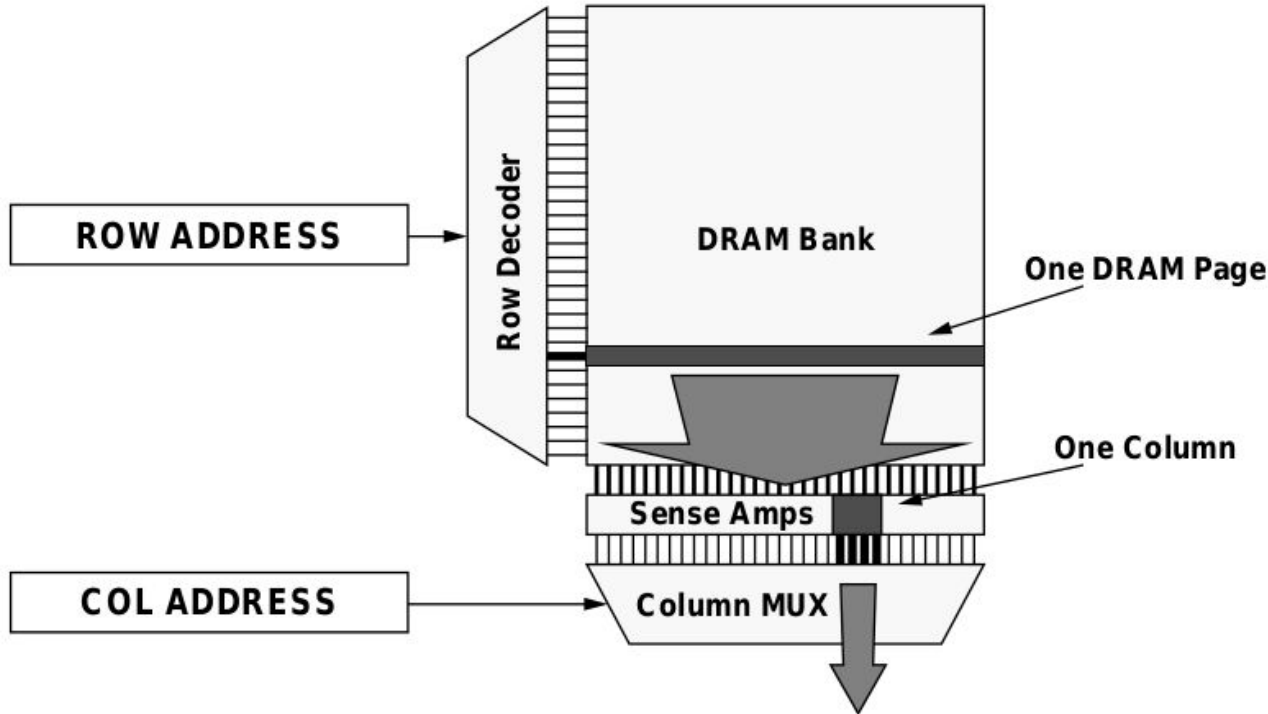
Why?

Interleave memory accesses to achieve high-bandwidth busses using low-bandwidth devices.



- 1
command transport and decode
- 2
in bank data movement
- 3
in device data movement
- 4
system data transport

Banks are (sort of) 3D



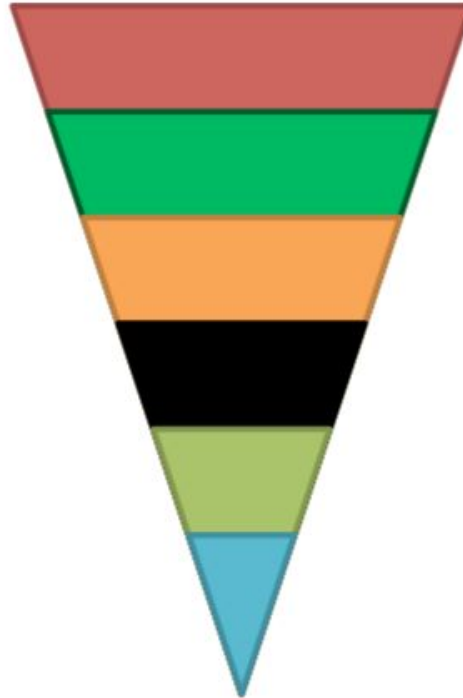
3D = 2D arrays with
#memory_arrays bits
in each cell.

Cell ~ column

DRAM page also
called DRAM row.

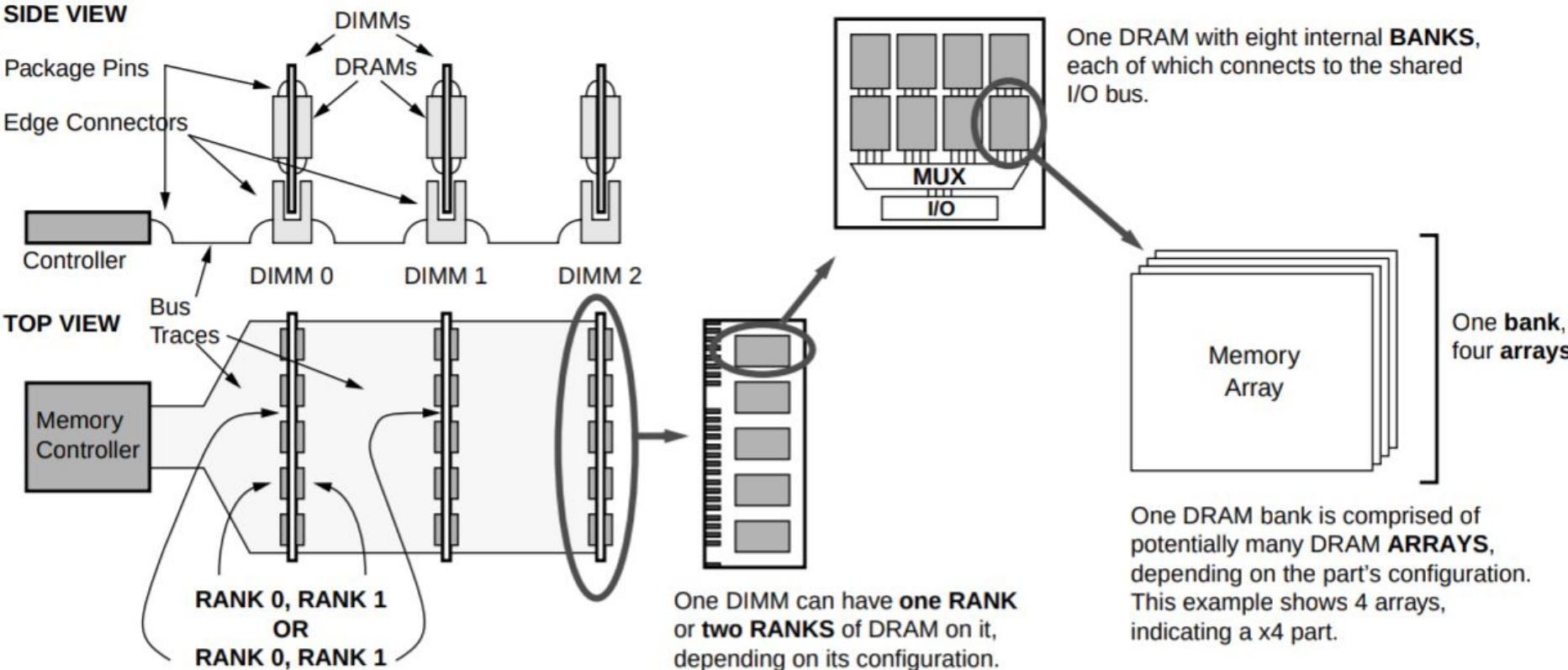
DRAM subsystem organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



- Multiple channels - multiple memory controllers. Each covers multiple ranks
- Multiple DIMMs containing multiple ranks
- Multiple ranks spanning multiple chips
- Chips within a rank accept the same commands.

Memory system organization



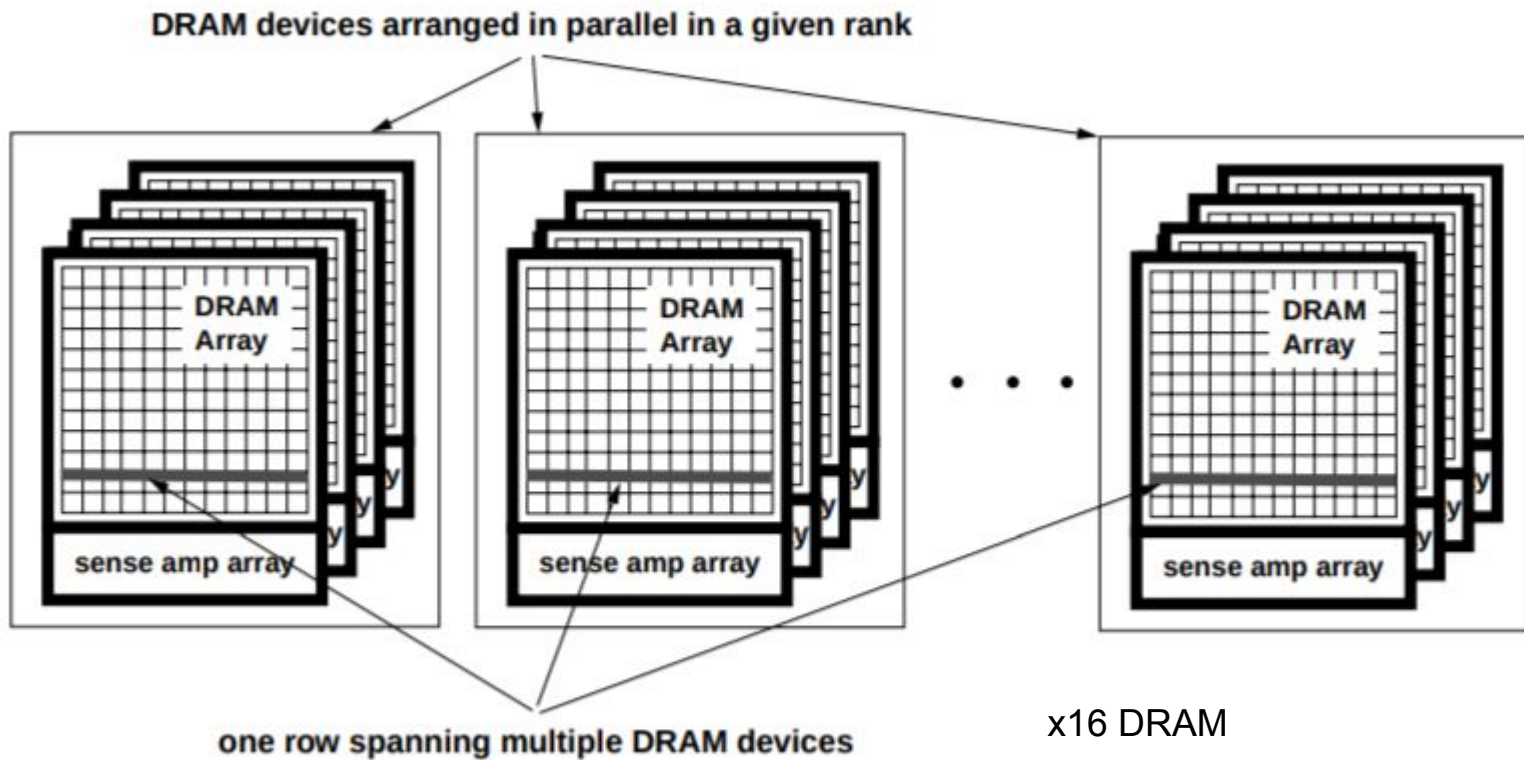
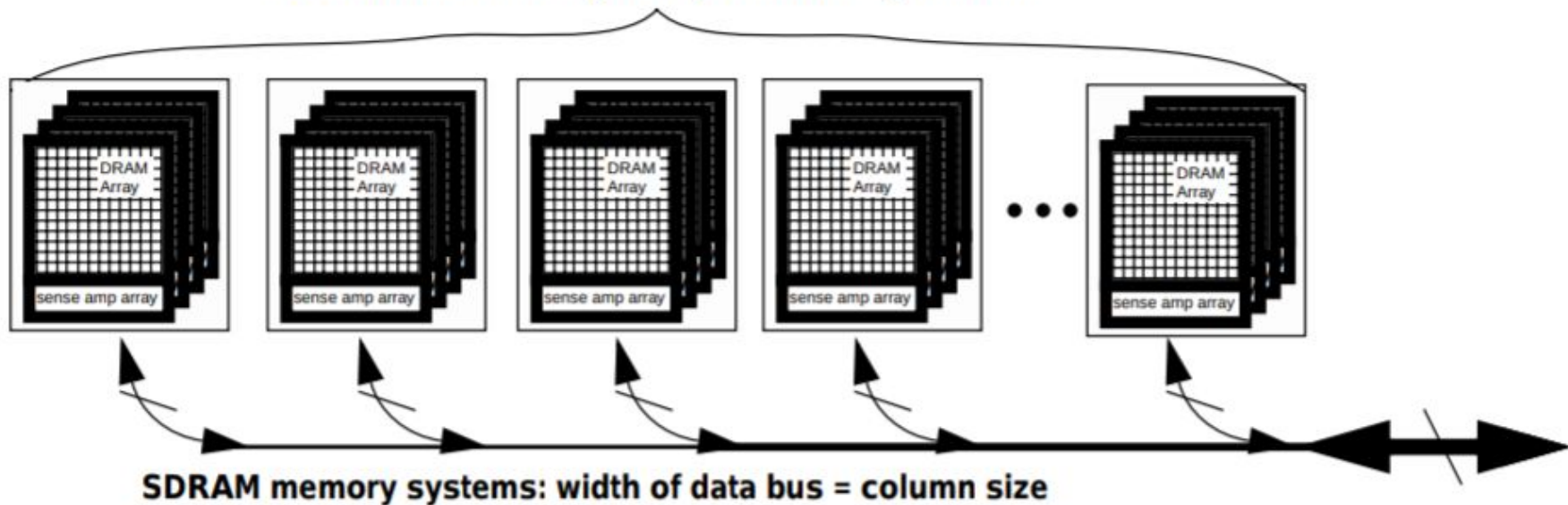


FIGURE 10.7: Generic DRAM devices with 4 banks, 8196 rows, 512 columns per row, and 16 data bits per column.

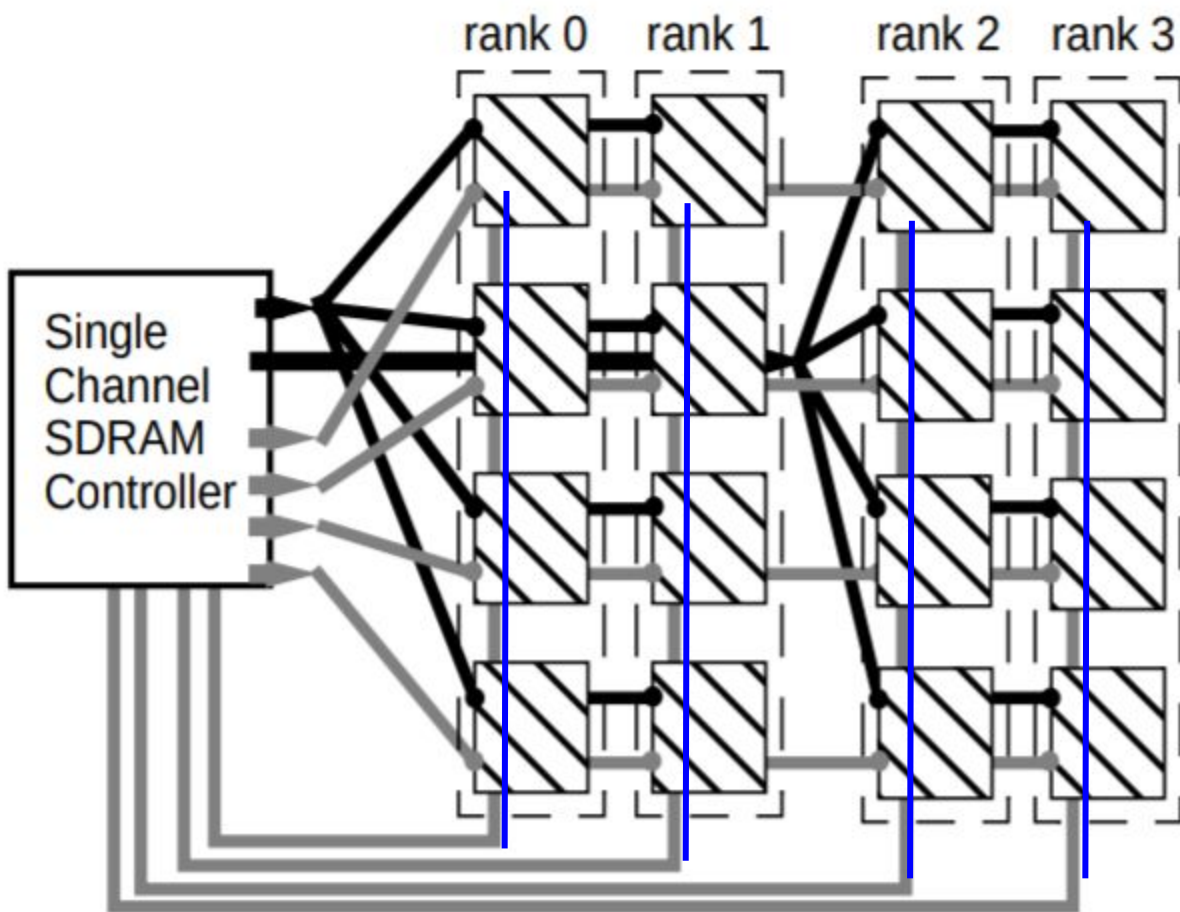
Column, **row**, and **bank** in different contexts (and publications) mean different things.

DRAM devices arranged in parallel in a given rank



DRAM devices in a rank respond to the same command & addr but send different data.

Together they form wider data bus. $64\text{bit data bus} = 8 * \text{x8 DRAM} = 4 * \text{x16 DRAM}$

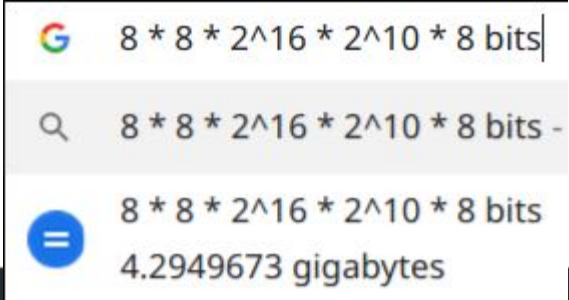


“Mesh Topology”

- Addr & Cmd
- Data Bus
- Chip (DIMM) Select

decode-dimms

```
---=== Memory Characteristics ===---
Maximum module speed          1600 MHz (PC3-12800)
Size                          4096 MB
Banks x Rows x Columns x Bits 8 x 16 x 10 x 64
Ranks                         1
SDRAM Device Width            8 bits
(...)
Number of SDRAM DIMMs detected and decoded: 1
```



8 * 8 * 2¹⁶ * 2¹⁰ * 8 bits

8 * 8 * 2¹⁶ * 2¹⁰ * 8 bits -

8 * 8 * 2¹⁶ * 2¹⁰ * 8 bits

= 4.2949673 gigabytes

Bits - probably rank width. So $64 / \text{SDRAM Device Width} = 8$ DRAM chips within rank.

$8 \text{ DRAM chips} * \text{DRAM width} * 2^{16} \text{ rows} * 2^{10} \text{ columns} * 8 \text{ banks} = 4 \text{ GB of memory.}$

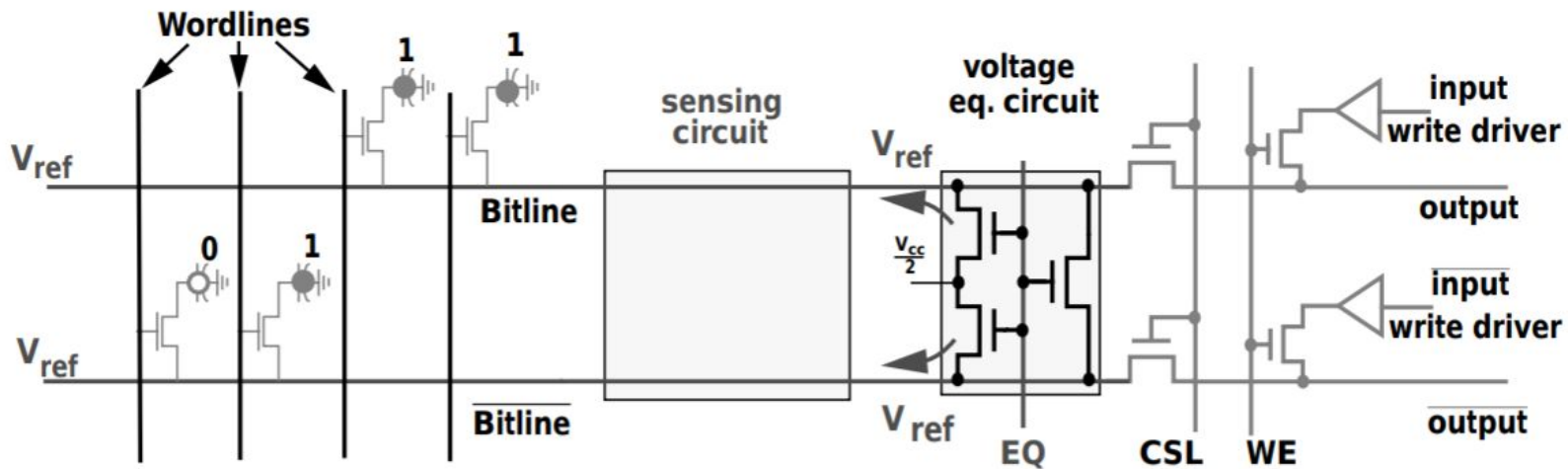
Install i2c-tools; modprobe eeprom; decode-dimms

Access sequence (simplified; to single DRAM)

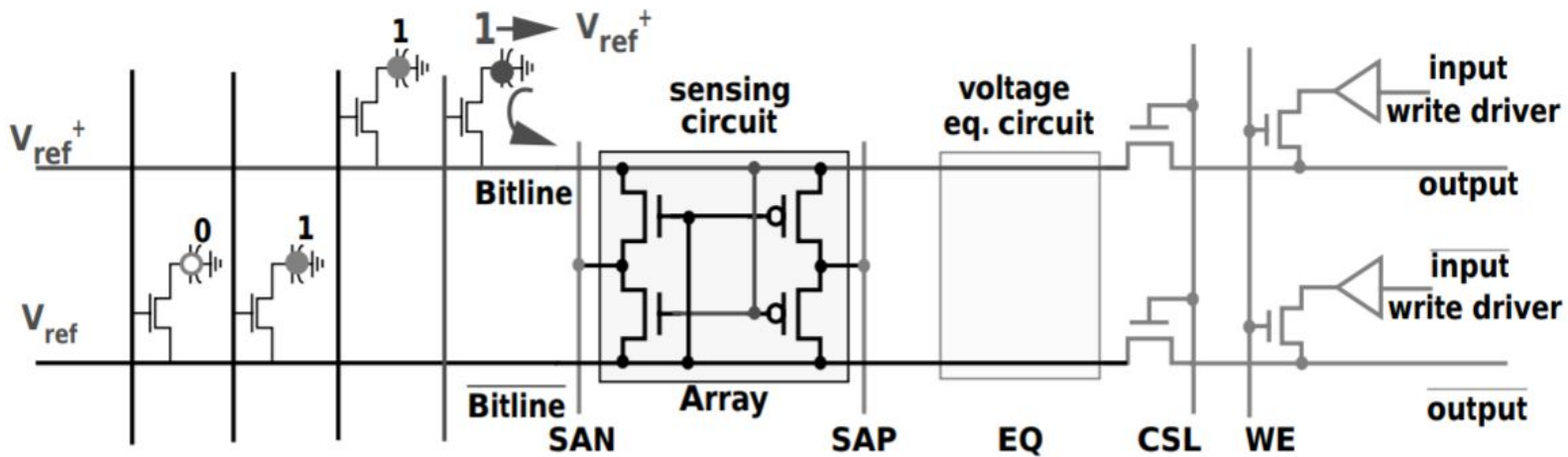
1. Precharge bitlines in appropriate bank.
2. Open/Activate row by raising appropriate wordline. This connects rows to bitlines, transferring data into the bank's row buffer (it's also amplified).
3. Read/write columns. Row buffer's data is accessed.
4. Close row. In order to access another row, current row's wordline must be lowered. Row-buffer is cleared. Charge is restored.

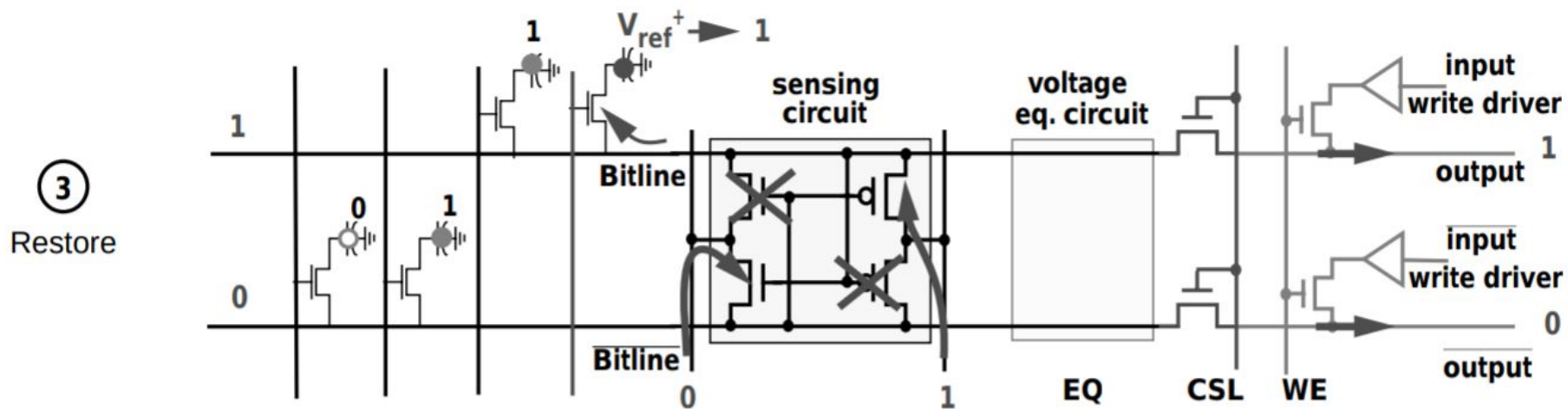
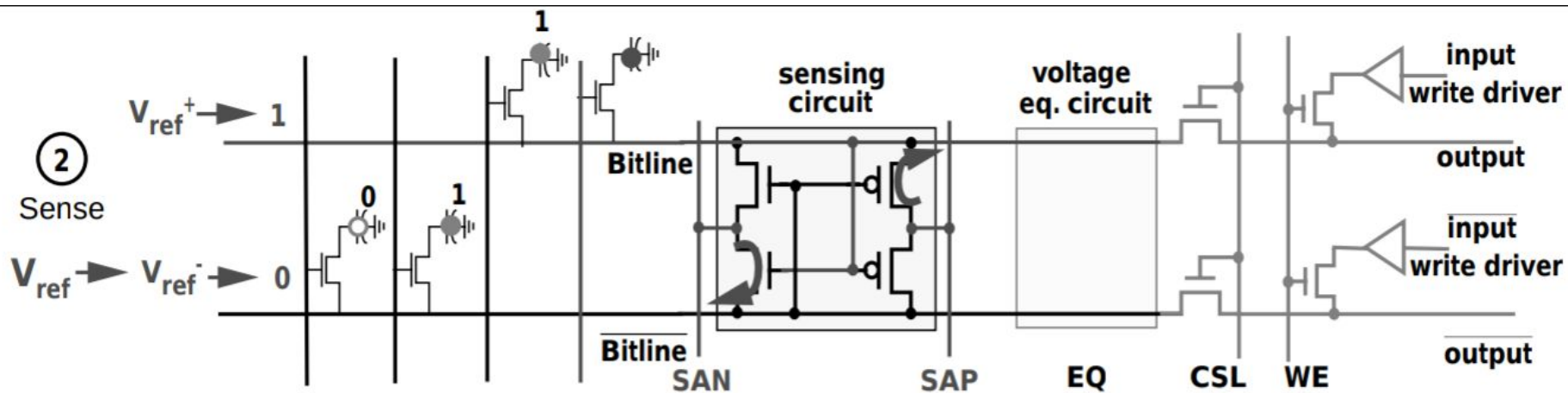
t_{RC} - row cycle time - timing constraint defined between a pair of ACTIVATES to the same row (in the same bank). Usually $\sim 55\text{ns}$.

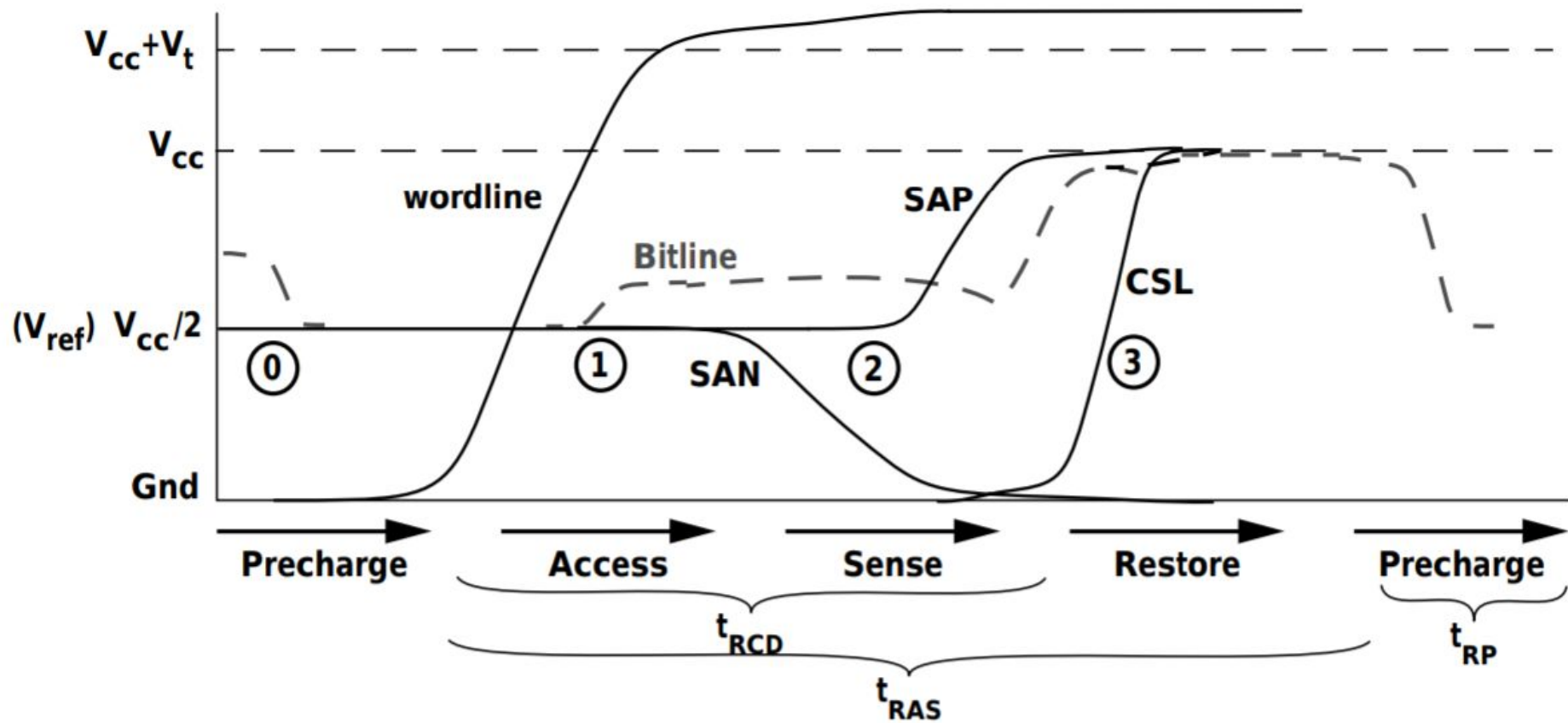
0
Precharge



1
Access







DRAM commands & refreshing

<i>Operation</i>	<i>Command</i>	<i>Address(es)</i>
1. Open Row	ACTIVATE (ACT)	Bank, Row
2. Read/Write Column	READ/WRITE	Bank, Column
3. Close Row	PRECHARGE (PRE)	Bank
Refresh (Section 2.4)	REFRESH (REF)	—

REFRESH == ACTIVATE

When row is activated, charges are amplified by sense-amps.

The DDR3 DRAM specifications guarantee a retention time of at least 64 milliseconds, meaning that all cells within a rank need to be refreshed at least once during this time window.

REF command refreshes many rows at a time. When a rank receives a REF, it automatically refreshes several of its least-recently-refreshed rows by internally generating ACT and PRE pairs to them.

Disturbance errors

When two circuit components interact with each other in unintended way.

Some errors can be detected by manufactures (DRAM is packed, it's hard to make it precise => remapping).

Hypothesis:

When a wordline's voltage is toggled repeatedly, some cells in nearby rows leak charge at a much faster rate. Such cells cannot retain charge for even 64ms, the time interval at which they are refreshed.

“Among 129 DRAM modules we analyzed (comprising 972 DRAM chips), we discovered disturbance errors in 110 modules (836 chips).”

Test code

```
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

clflush - evicts data from the cache.

mfence - ensures data is fully flushed*.

Physical addr to bank mapping disclosed on AMD, reverse-engineered on Intel.

X and Y map to the same bank, but to different rows within the bank. This forces the memory controller to open and close the two rows repeatedly: (ACT(X), READ(X), PRE(X), ACT(Y), READ(Y), PRE(Y), ...).

Execute with different (X,Y) until every row is opened/closed millions of times.

Bit-flips induced by disturbance on a 2GB module

Bit-Flip	Sandy Bridge	Ivy Bridge	Haswell	Piledriver
'0' → '1'	7,992	10,273	11,404	47
'1' → '0'	8,125	10,449	11,467	12

The faster a processor accesses DRAM, the more bit-flips it has.

Code1b:

Access pattern: (reqX , reqX , reqX , ...). In this case, the memory controller minimizes the number of DRAM commands by opening and closing the row just once, while issuing many column reads in between: (ACTX, READX, READX, READX, ... , PREX).

No disturbance errors detected.

FPGA-based test environment

```
1 TESTBULK(AI, RI, DP)
2   setAI(AI)
3   setRI(RI)
4    $N \leftarrow (2 \times RI)/AI$ 
5
6   writeAll(DP)
7   for  $r \leftarrow 0 \dots ROW_{MAX}$ 
8     for  $i \leftarrow 0 \dots N$ 
9       ACT  $r^{th}$  row
10      READ  $0^{th}$  col.
11      PRE  $r^{th}$  row
12   readAll()
13   findErrors()
```

a. Test all rows at once

```
1 TESTEACH(AI, RI, DP)
2   setAI(AI)
3   setRI(RI)
4    $N \leftarrow (2 \times RI)/AI$ 
5
6   for  $r \leftarrow 0 \dots ROW_{MAX}$ 
7     writeAll(DP)
8     for  $i \leftarrow 0 \dots N$ 
9       ACT  $r^{th}$  row
10      READ  $0^{th}$  col.
11      PRE  $r^{th}$  row
12   readAll()
13   findErrors()
```

b. Test one row at a time

AI - activation interval;
time it takes to
execute one iteration
of the inner for loop.
RI - refresh interval;
how frequently module
is refreshed
DP - data pattern

AI - 55ns - t_{RC} min
RI - 64ms - default

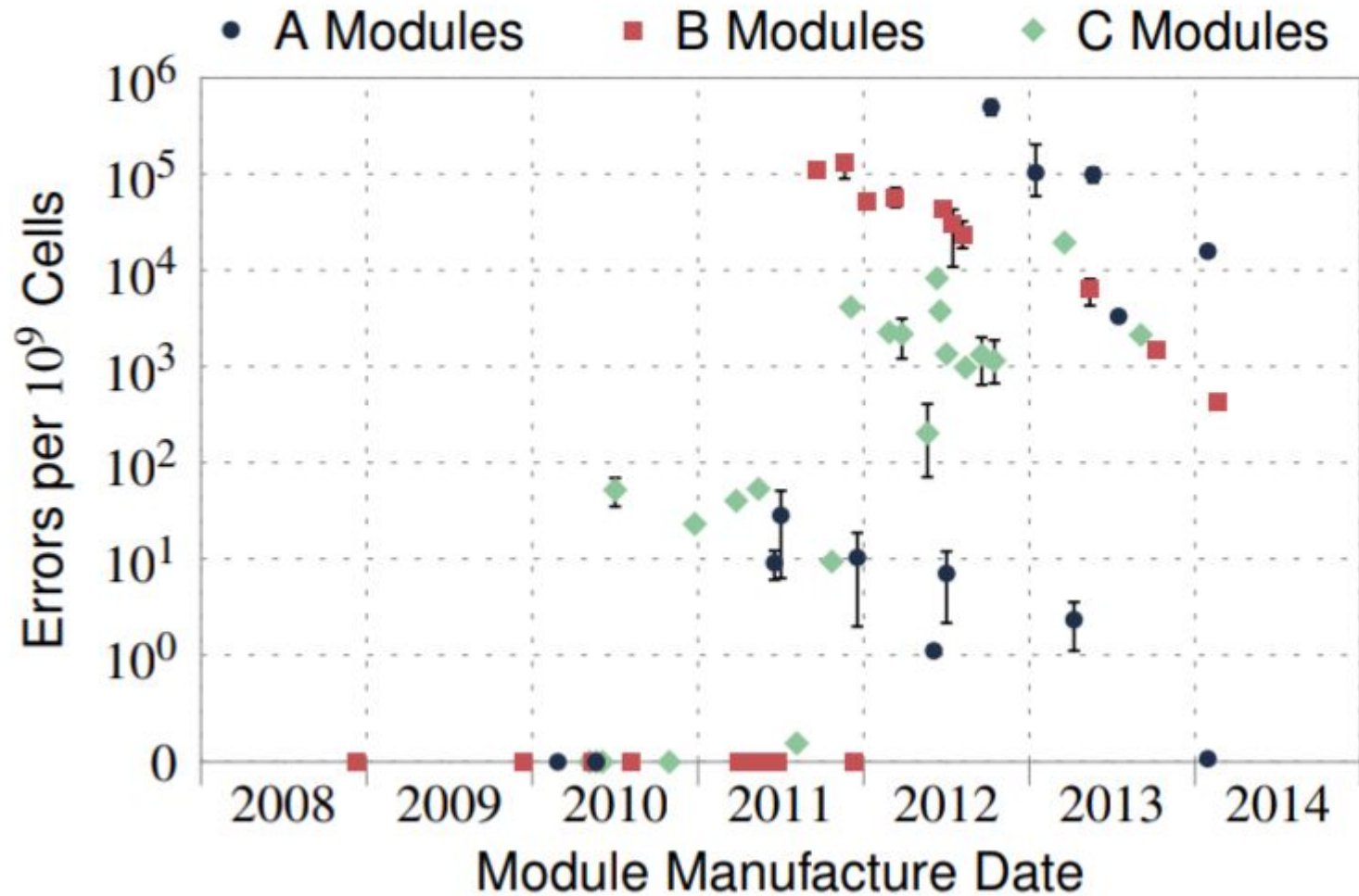
Nomenclature

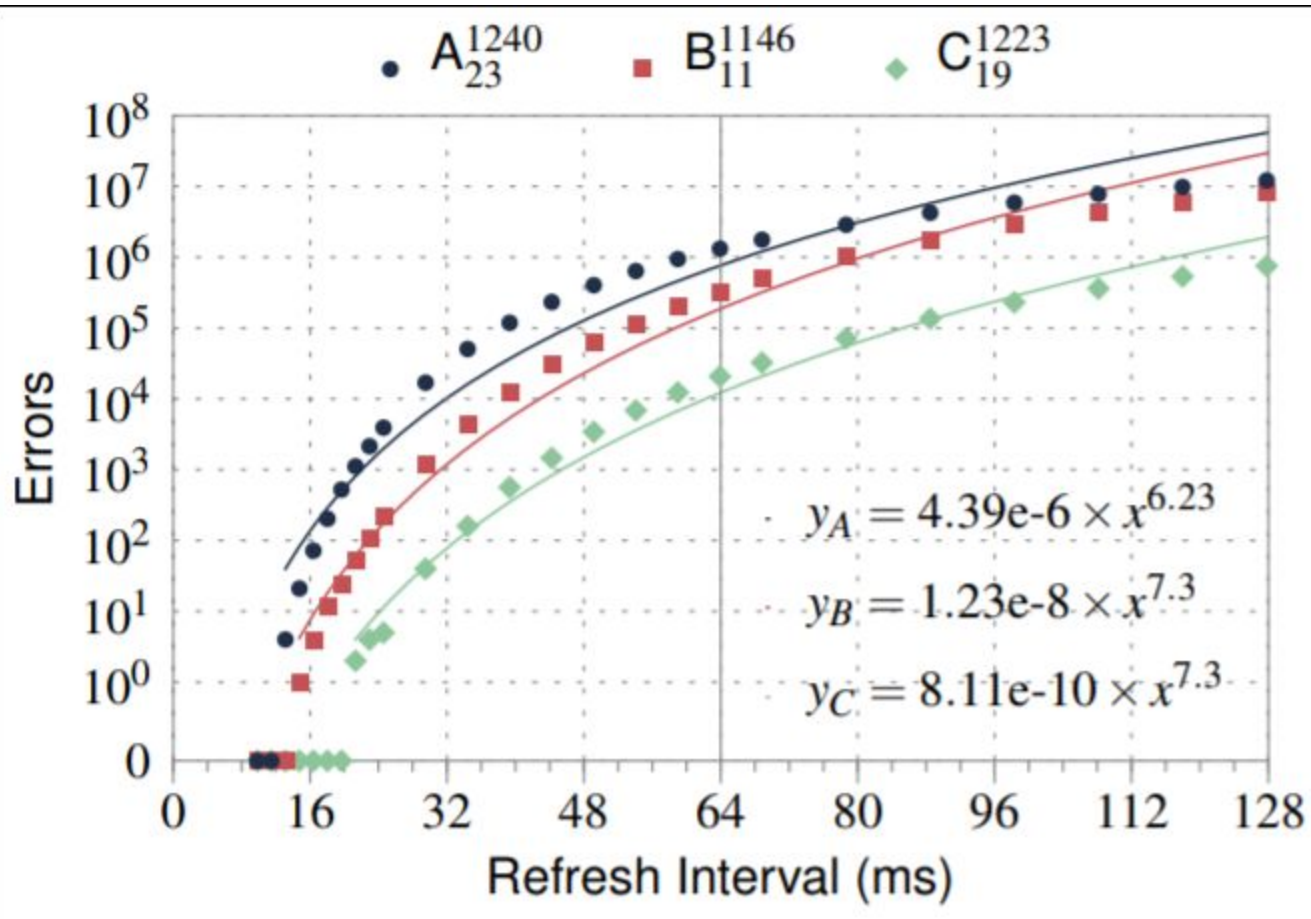
- Victim row
- Aggressor row

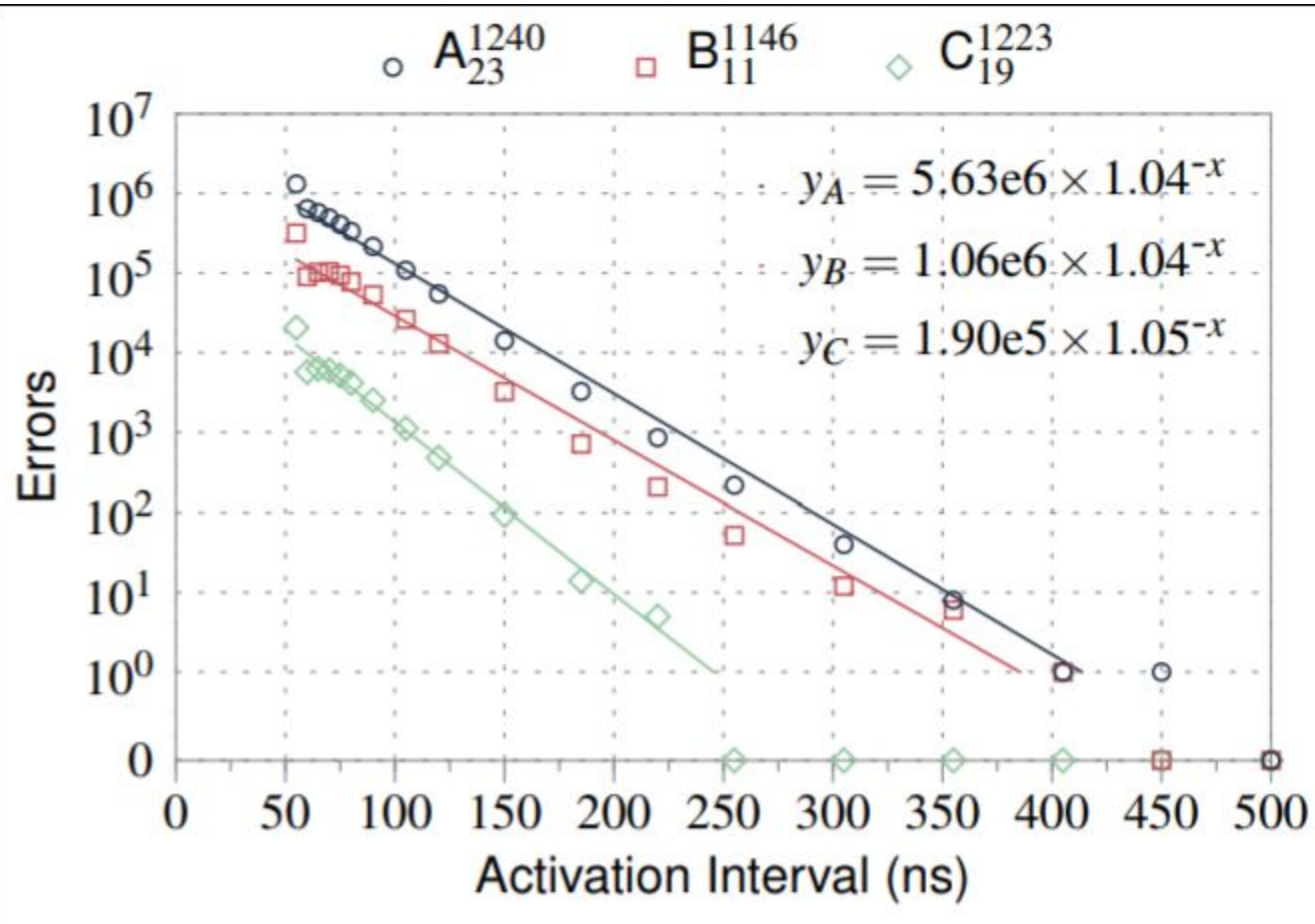
If a cell experienced an error in either of the runs, we refer to it as a victim cell for that module.

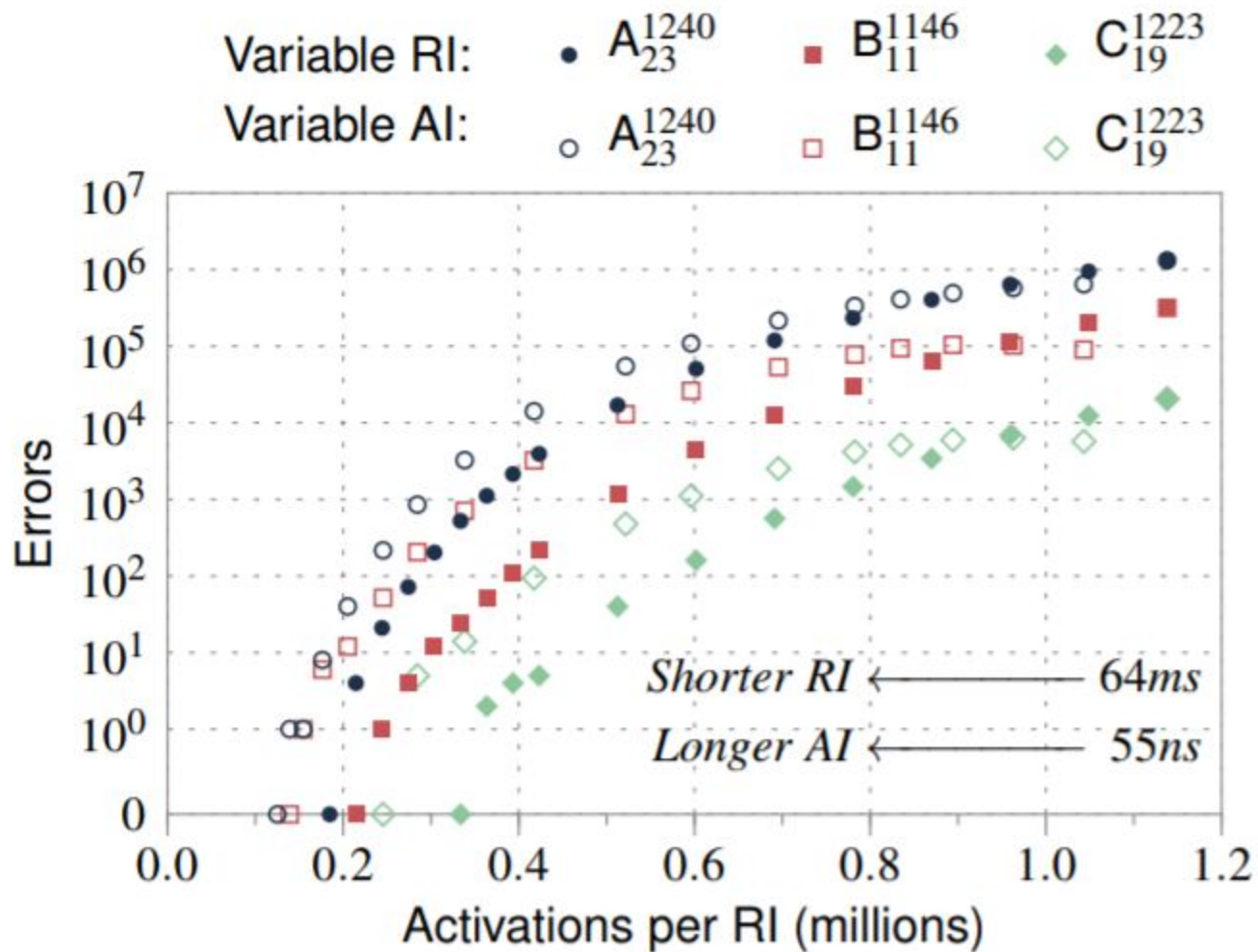
In most of the tests, $AI=55\text{ns}$ and $RI=64\text{ms}$, for which the corresponding value of N is $2.33 \cdot 10^6$.

A







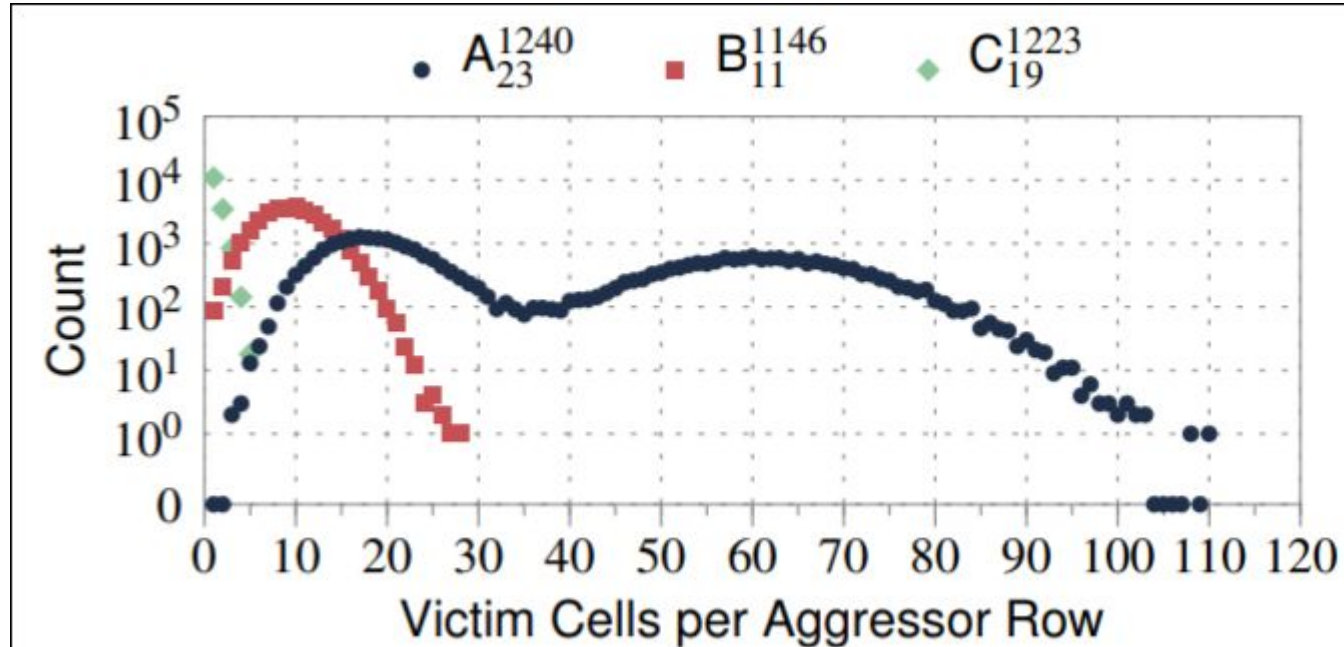


ECC?

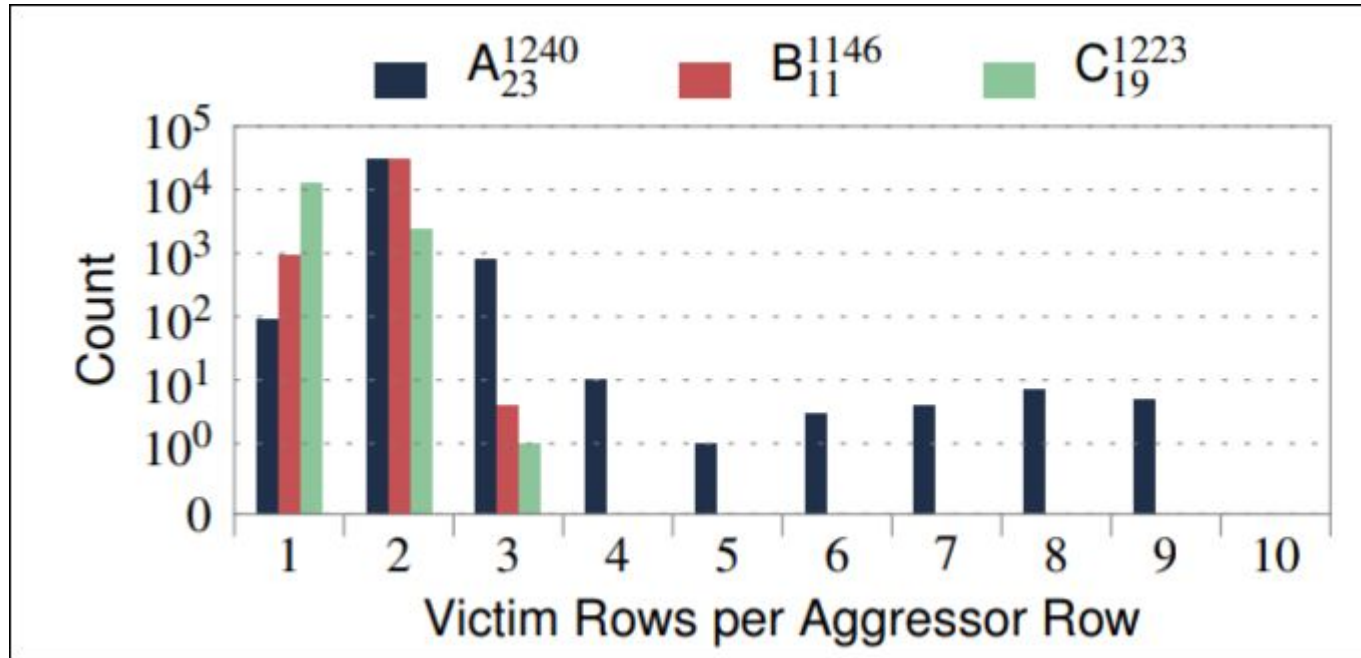
<i>Module</i>	<i>Number of 64-bit words with X errors</i>			
	<i>X = 1</i>	<i>X = 2</i>	<i>X = 3</i>	<i>X = 4</i>
A₂₃	9,709,721	181,856	2,248	18
B₁₁	2,632,280	13,638	47	0
C₁₉	141,821	42	0	0

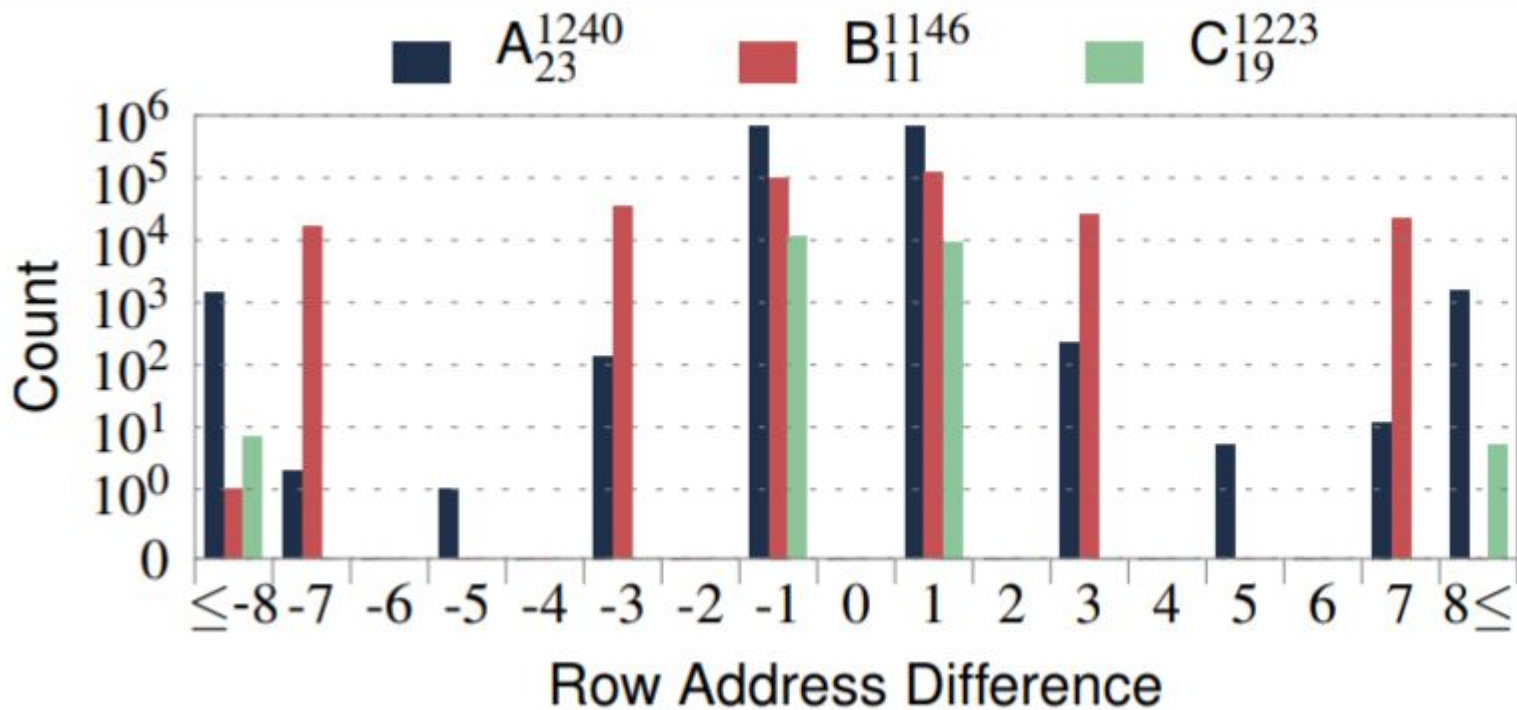
ECDED (single error-correction, double error-detection) can correct only a single-bit error within a 64-bit word. If a word contains two victims, however, SECDED cannot correct the resulting double-bit error. And for three or more victims, SECDED cannot even detect the multi-bit error, leading to silent data corruption.

How many times aggressor row flips X cells



How many times aggressor row affects X rows





- Aggressor is refreshing its data => 0 errors.
- Errors in non-adjacent rows => re-mapping
- Logical / physical adjacency

1 -> 0 & 0 -> 1 flips

- Some cells might represent logical value 1 as charged and some as discharged.
- RowStripe = (even/odd rows '0's/'1's); Solid (all '0's')

<i>Module</i>	TESTBULK(<i>DP</i>) + TESTBULK(\sim <i>DP</i>)			
	Solid	RowStripe	ColStripe	Checkered
A ₂₃	112,123	1,318,603	763,763	934,536
B ₁₁	12,050	320,095	9,610	302,306
C ₁₉	57	20,770	130	29,283

Table 6. Number of errors for different data patterns



ROWhammer

It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after

Flipping bits in memory for fun (and profit)

- Generic strategy
 - Identify data structure that, if randomly bit-flipped, yields improved privileges
 - Fill as much memory as possible with this data structure
 - Wait for the bit flip to occur
- Apply this to JVM:
 - Spray memory with references
 - Bit flip causes reference to point to object of wrong type

Flipping bits in memory for fun (and profit)

<https://github.com/google/rowhammer-test>

Escape JVM, NaCl. Get access to kernel-memory:

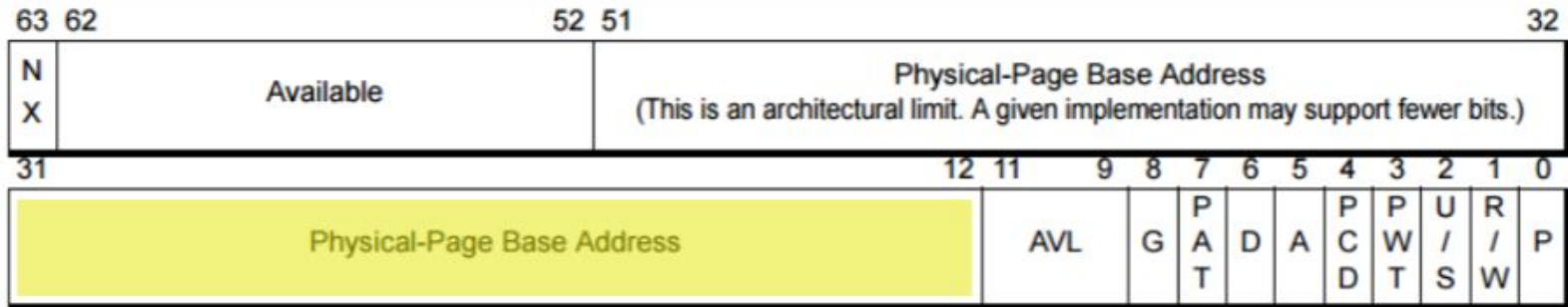


Figure 5-21. 4-Kbyte PTE—Long Mode

Could flip:

- “Writable” permission bit (RW): 1 bit → 2% chance
- Physical page number: 20 bits on 4GB system → 31% chance

Error correction mechanisms

- Parity - detection
- SEC ECC single-bit error correction
- SECDED ECC Single-bit error correction, double-bit error detection
- Multi-Bit Error Detection and Correction: Bossen's b-Adjacent Algorithm

More about that in “*Memory Systems: Cache, DRAM, Disk*” - chapter 30.

Consider State(full/less), Space-cost, Time-cost, energy-cost.

Solutions

1. **Make better chips** - improve circuit design (more precision). Problem could resurface when the process technology is upgraded.
2. **Correct errors** - ECC. Space inefficient, high cost, cannot correct multi-bit disturbance errors.
3. **Refresh all rows frequently** - energy and time inefficient.
4. **Retire cells (manufacturer)** - days to find faulty rows. Might not be enough healthy rows.
5. **Dynamically identify “hot” rows and refresh neighbours** - needs too much hw to track hot rows; heuristics and filters sometimes cause multiple rows to be refreshed.

PARA - probabilistic adjacent row activation

Every time a row is opened and closed, one of its adjacent rows is also opened (i.e., refreshed) with some low probability p . Choose adjacent row with equal ppb.

Implemented in memory-controller. Stateless.

Probability of error on N accesses = $(1-p/2)^N$ (binomial distribution)

Mapping disclosure

To enable low-overhead solutions, manufacturers should disclose how they map logical rows onto physical rows. Along with other metadata about the module (e.g., capacity, and bus frequency), the mapping function could be stored in a small ROM (called the SPD) that exists on every DRAM module.

The manufacturers should also disclose how they re-map faulty physical rows. It also could be stored in SPD.

PARA results

Adversarial access pattern opens and closes a row just enough times (N_{th}) during a refresh interval but no more ($p=0.001$) .

Duration	$N_{th}=50K$	$N_{th}=100K$	$N_{th}=200K$
<i>64ms</i>	1.4×10^{-11}	1.9×10^{-22}	3.6×10^{-44}
<i>1 year</i>	6.8×10^{-3}	9.4×10^{-14}	1.8×10^{-35}

Table 7. Error probabilities for PARA when $p=0.001$

PARA performance impact

Impact on 29 single-threaded workloads from SPEC CPU2006, TPC, and memory-intensive microbenchmarks evaluated on a cycle-accurate DRAM simulator.

Averaged across all 29 benchmarks, there was only a 0.197% degradation in instruction throughput during the simulated duration of 100ms. In addition, the largest degradation in instruction throughput for any single benchmark was 0.745%.

Soooo it looks lightweight.



Bibliography

- Onur Mutlu et al. - *Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors*
- Bruce Jacob, David T. Wang, Spencer Ng - *Memory Systems: Cache, DRAM, Disk*
- Mark Seaborn, Thomas Dullien - *Exploiting the DRAM rowhammer bug to gain kernel privileges. How to cause and exploit single bit errors*