University of BRISTOL

# Introduction to Rowhammer

Sanjay Rawat

# Effect of Rowhammer

- Memory corruption but not via software bugs!
- This changes the dynamics of software defence mechanism
- Not easy to fix

- Organization (main take-away)
  - General introduction to rowhammer
  - DRAM design (only parts that facilitate rowhammering)
  - Attack Example

The slides are based on:

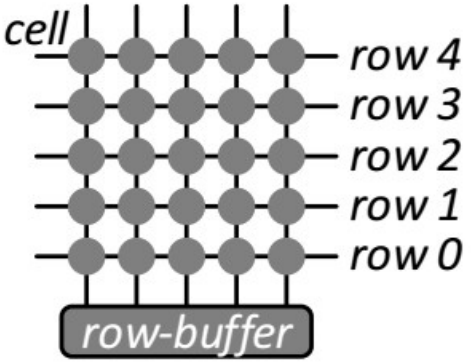
1. *Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors*, **Kim et al**.
2. Various talks/articles by Ander Fogh.
3. *Exploiting the DRAM rowhammer bug to gain kernel privileges*, **Seaborn and Dullien**

# What is Rowhammer

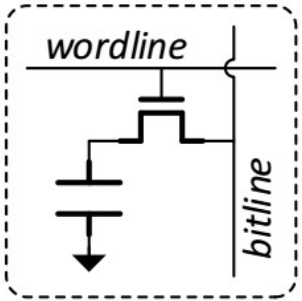
- DRAM is organized as rows/cells, which are densely populated
- Activation of a row interferes with the adjacent rows
- This results in discharge, thus change in the value they represent!
- Rowhammer is an exploitation of DRAM design to flip bit
- Flipped bits may correspond to control-bits, used for several checks

# Design of DRAM

- DRAM *module*, consists of DRAM *ranks,* which in turn are groups of DRAM *chips.*
- DRAM is a two dimensional array of *cells.*
- Each cell consists of a *capacitor* and an *access-transistor*.
- Accessing a cell involved *wordline and bitline*




**a.** Rows of cells

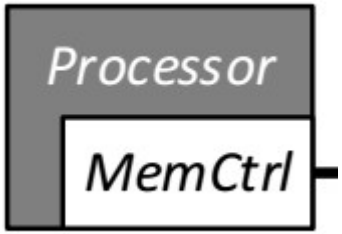


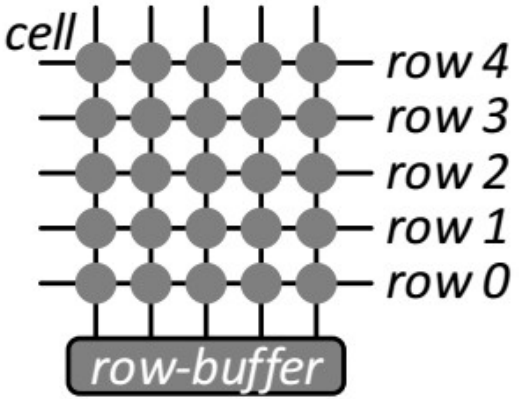

**b.** A single cell

# Accessing a row

1) Open Row. A row is opened by raising its wordline (`ACT row_addr`). This connects the row to the bitlines, transferring all of its data into the bank's row-buffer.
2) Read/Write Columns. The row-buffer's data is accessed by reading or writing any of its columns as needed (`READ/WRITE col_addr`).
3) Close Row. Before a different row in the same bank can be opened, the original row must be closed by lowering its wordline (`PRE bank_addr`). In addition, the row-buffer is cleared.

# illustration



# Rowhammer Primitive

- Capacitor charge/discharge maps to binary data.
- And it leaks charge, which requires *refresh*
- DDR3 DRAM specifications guarantee a retention time of at least 64 milliseconds

- 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.
- Ultimately, this leads to the cells losing data and experiencing disturbance errors.

- 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.
- Ultimately, this leads to the cells losing data and experiencing disturbance errors.

```
1 Attack_Loop:
2   mov  (addr_X), %rax // read the row X
3   mov  (addr_Y), %rbx // read the row Y
4   clflush  (addr_X) // flush X from cache
5   clflush  (addr_Y) // flush Y from cache
6   jmp  Attack_Loop
```

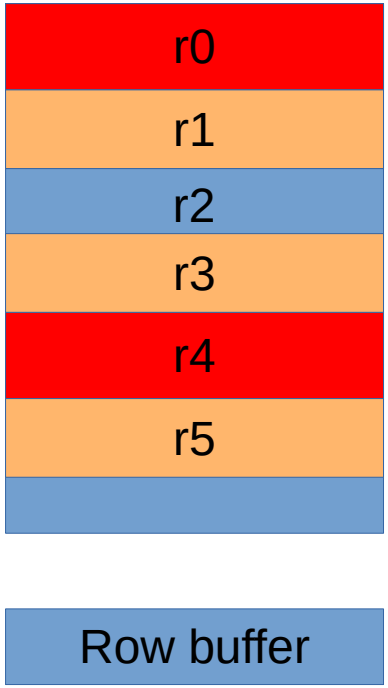
# How bits can be flipped

Single sided hammering

Double sided hammering

# How bits can be flipped

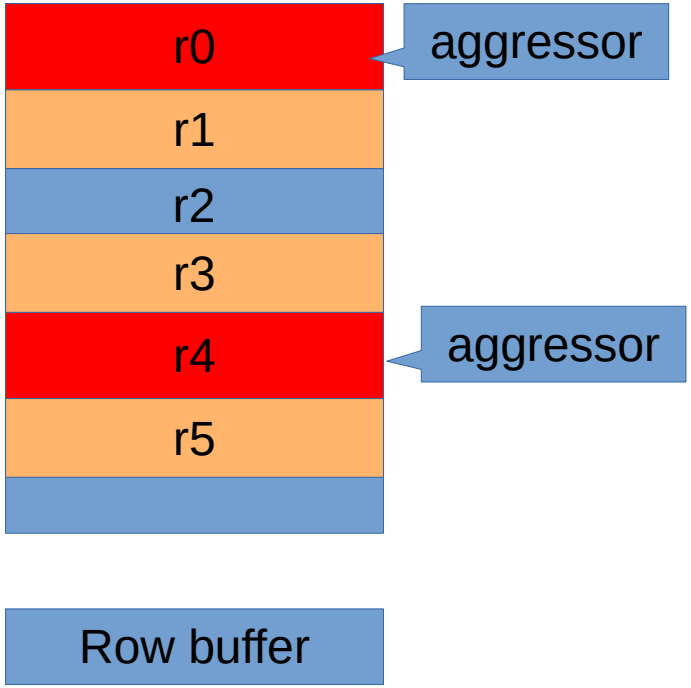
Single sided hammering




Double sided hammering

# How bits can be flipped

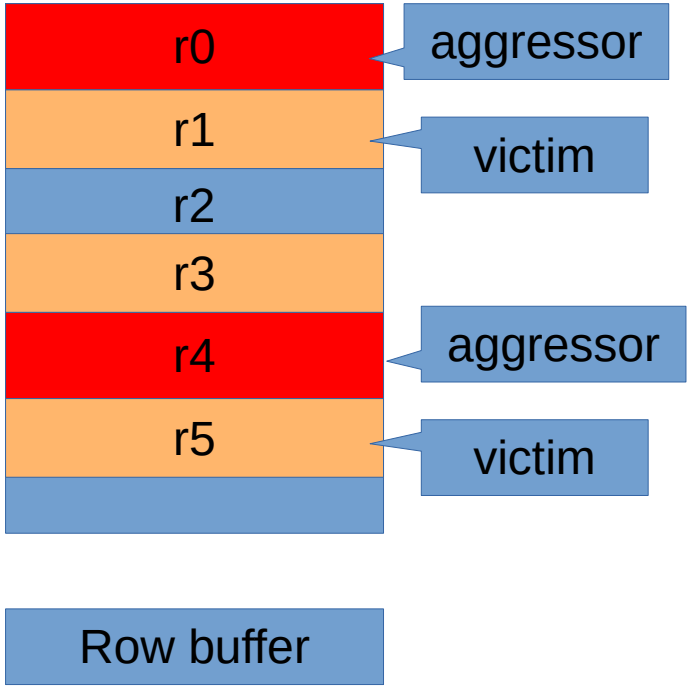
Single sided hammering




Double sided hammering

# How bits can be flipped

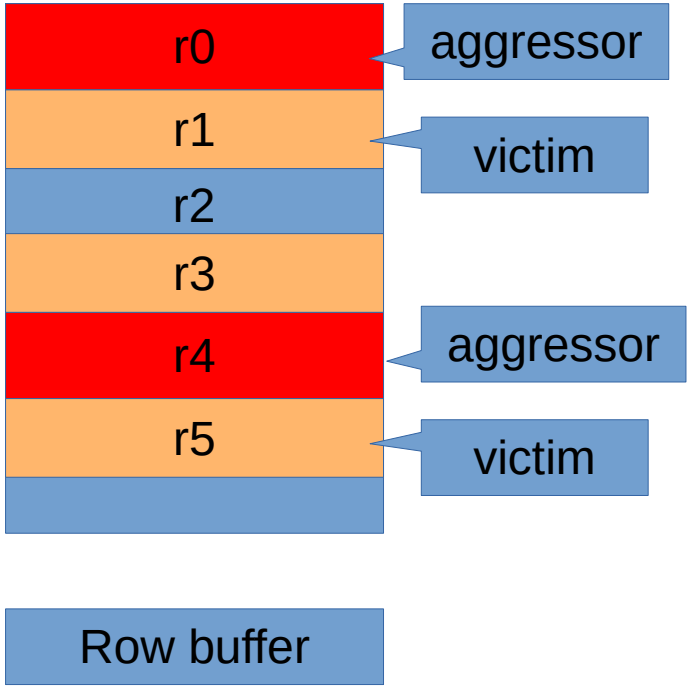
Single sided hammering



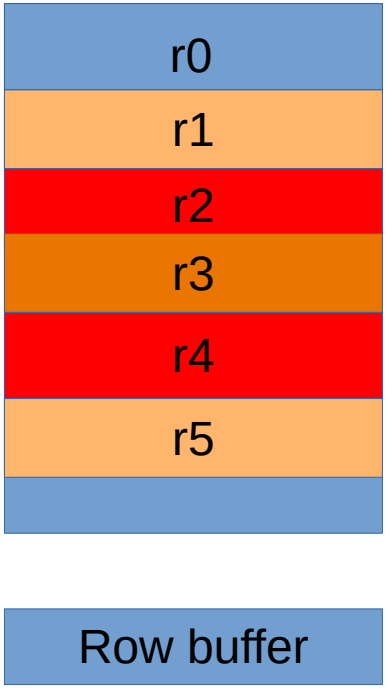

Double sided hammering

# How bits can be flipped

Single sided hammering

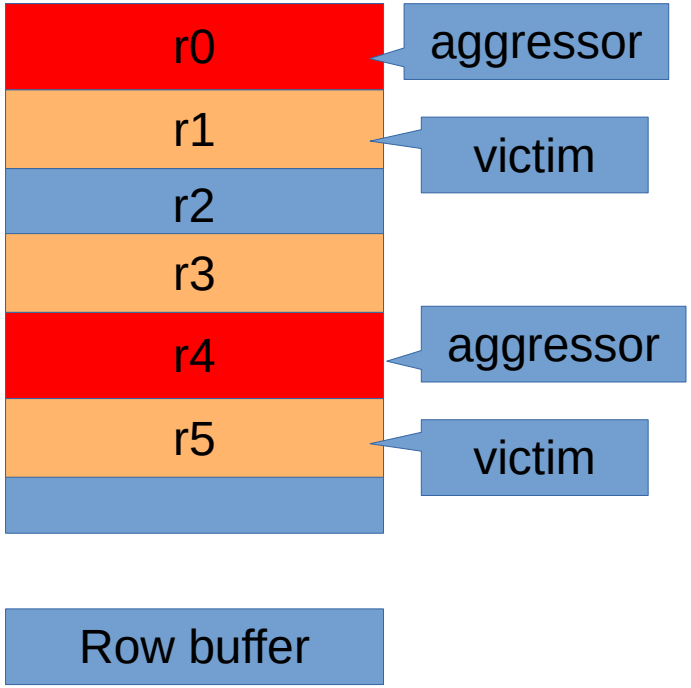



Double sided hammering

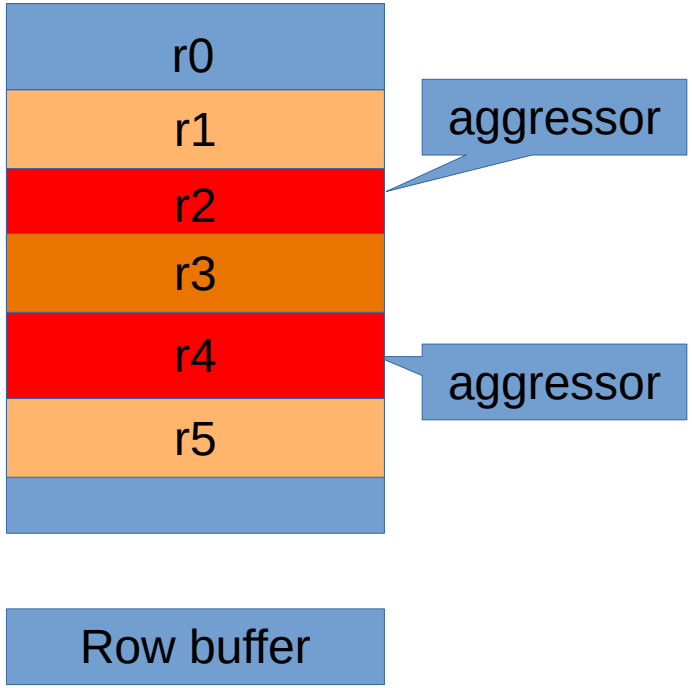


# How bits can be flipped

Single sided hammering

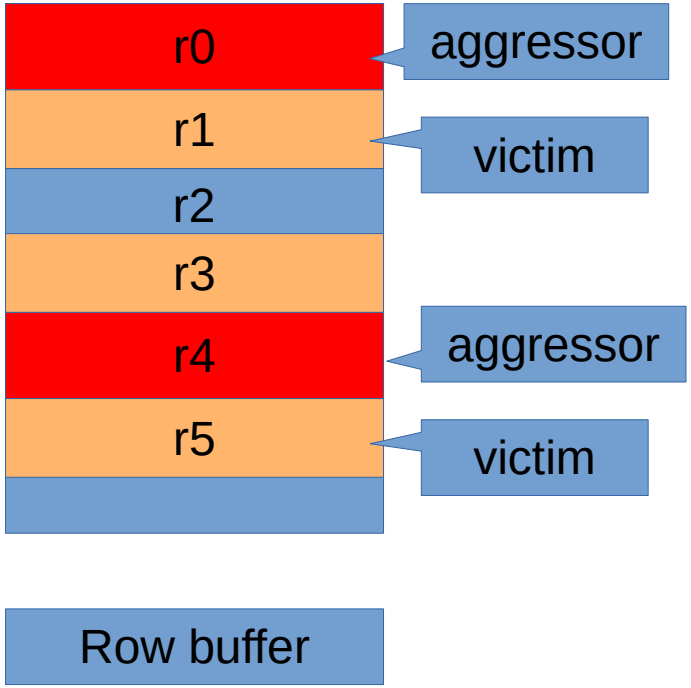



Double sided hammering

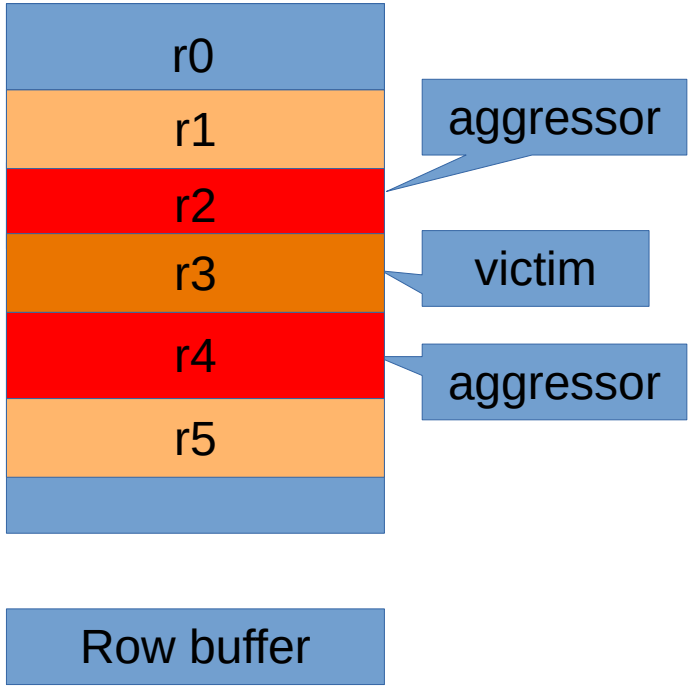


# How bits can be flipped

Single sided hammering




Double sided hammering



# A 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

# Sandbox escape Example

- Sandbox for running native code (C/C++)
- Applications can only access addresses inside the sandbox

# Sandbox escape Example

- Sandbox for running native code (C/C++)
- Applications can only access addresses inside the sandbox
- Safe instruction sequence:

```
andl $~31, %eax // Truncate address to 32 bits
// and mask to be 32-byte-aligned.
addq %r15, %rax // Add %r15, the sandbox base address.
jmp *%rax // Indirect jump.
```

# Sandbox escape Example

- Sandbox for running native code (C/C++)
- Applications can only access addresses inside the sandbox
- Safe instruction sequence:

```
andl $~31, %eax // Truncate address to 32 bits
// and mask to be 32-byte-aligned.
addq %r15, %rax // Add %r15, the sandbox base address.
jmp *%rax // Indirect jump.
```

What if `rax → rcx` (which points to outside address!)

# Sandbox escape Example

- Sandbox for running native code (C/C++)
- Applications can only access addresses inside the sandbox
- Safe instruction sequence:

```
andl $~31, %eax // Truncate address to 32 bits
// and mask to be 32-byte-aligned.
addq %r15, %rax // Add %r15, the sandbox base address.
jmp *%rax // Indirect jump.
```

What if `rax → rcx` (which points to outside address!)
`Jmpq %*rax` code `ff e0`; `jmpq %*rcx` code `ff e1`

# Sandbox escape Example

- Sandbox for running native code (C/C++)
- Applications can only access addresses inside the sandbox
- Safe instruction sequence:

```
andl $~31, %eax // Truncate address to 32 bits
// and mask to be 32-byte-aligned.
addq %r15, %rax // Add %r15, the sandbox base address.
jmp *%rax // Indirect jump.
```

What if `rax → rcx` (which points to outside address!)
`Jmpq %*rax` code `ff e0`; `jmpq %*rcx` code `ff e1`
1111111111000**0** → 1111111111000**1**

- If interested more attacks, like page table entries hammering, read:
- *Exploiting the DRAM rowhammer bug to gain kernel privileges*, **Seaborn and Dullien**