

# RowHammer

Mengjia Yan

Spring 2024



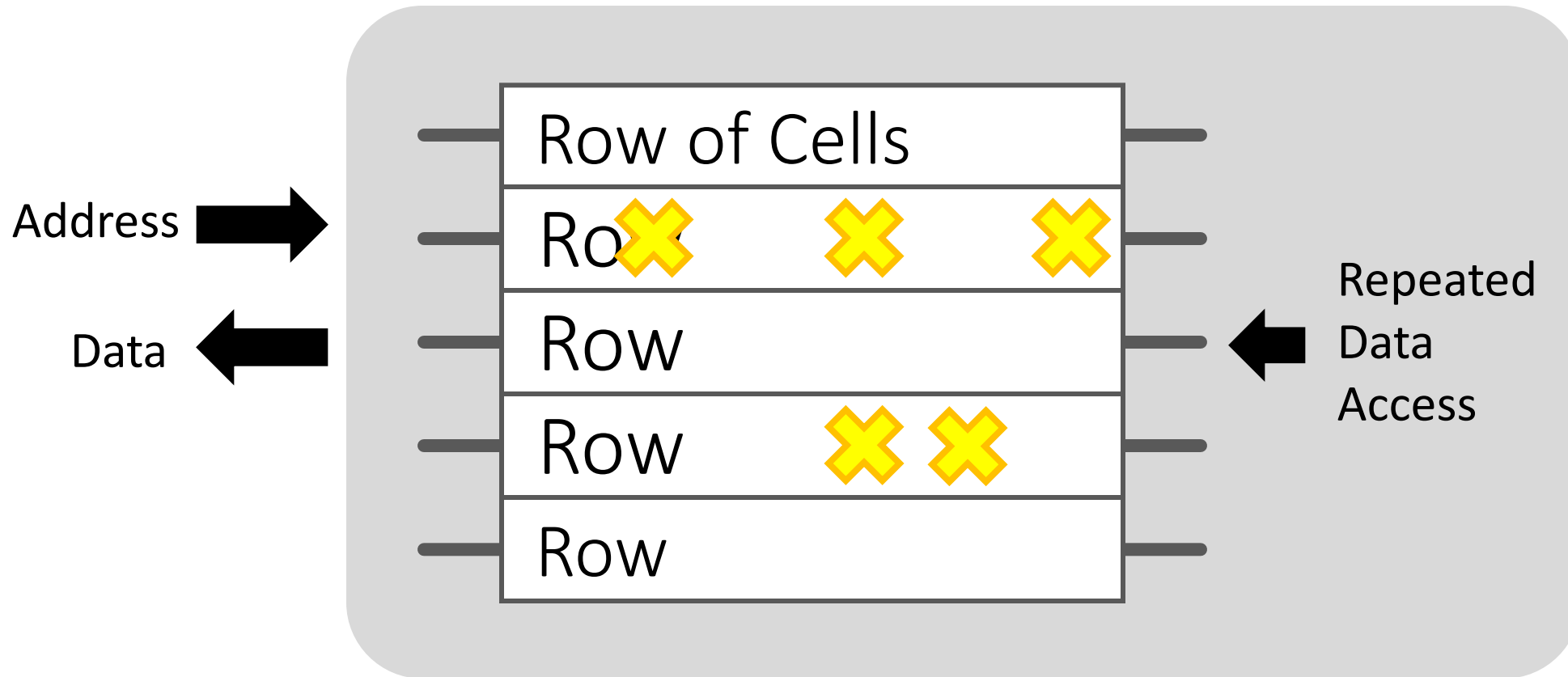
# RowHammer In One Sentence

Address 

Data 

DRAM

# RowHammer In One Sentence



**Observation:** Repeatedly accessing a row enough times can cause disturbance errors in nearby rows

# Outline

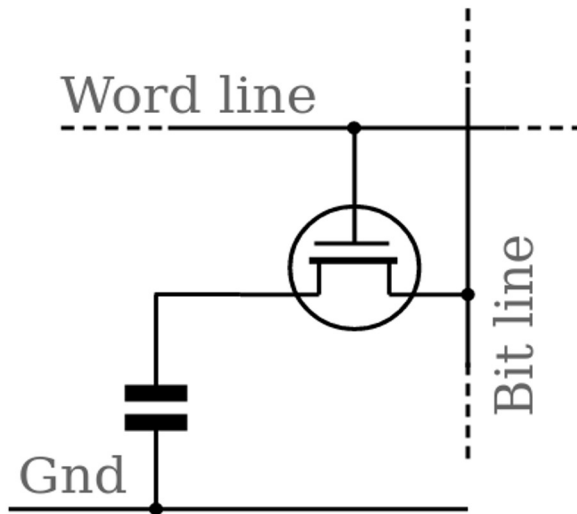
- Why does RowHammer happen? What is its working mechanism?
- How to trigger RowHammer in practice? Any challenges? --> Lab 4
- If you are an attacker, what do you do with it?
- If you are a defender, what can you do? --> Next Lecture

# DRAM Basics



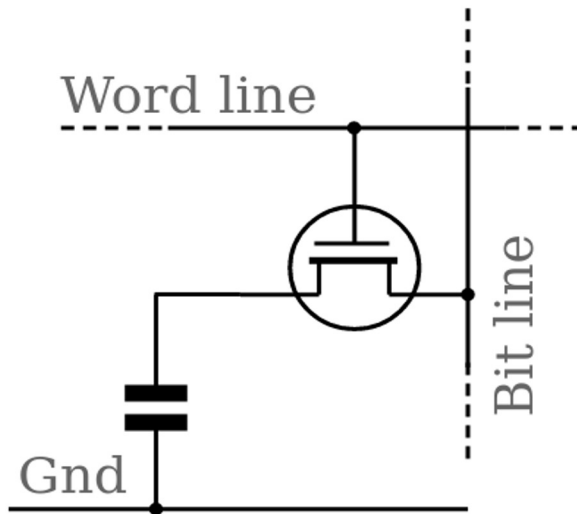
# DRAM Basics

- Each bit in DRAM is stored in a “cell” using a *capacitor*
- *Read is destructive*
- DRAM cells lose their state over time (hence **Dynamic** RAM)
- Data stored in DRAM cells needs to be “**refreshed**” at a regular interval



# DRAM Basics

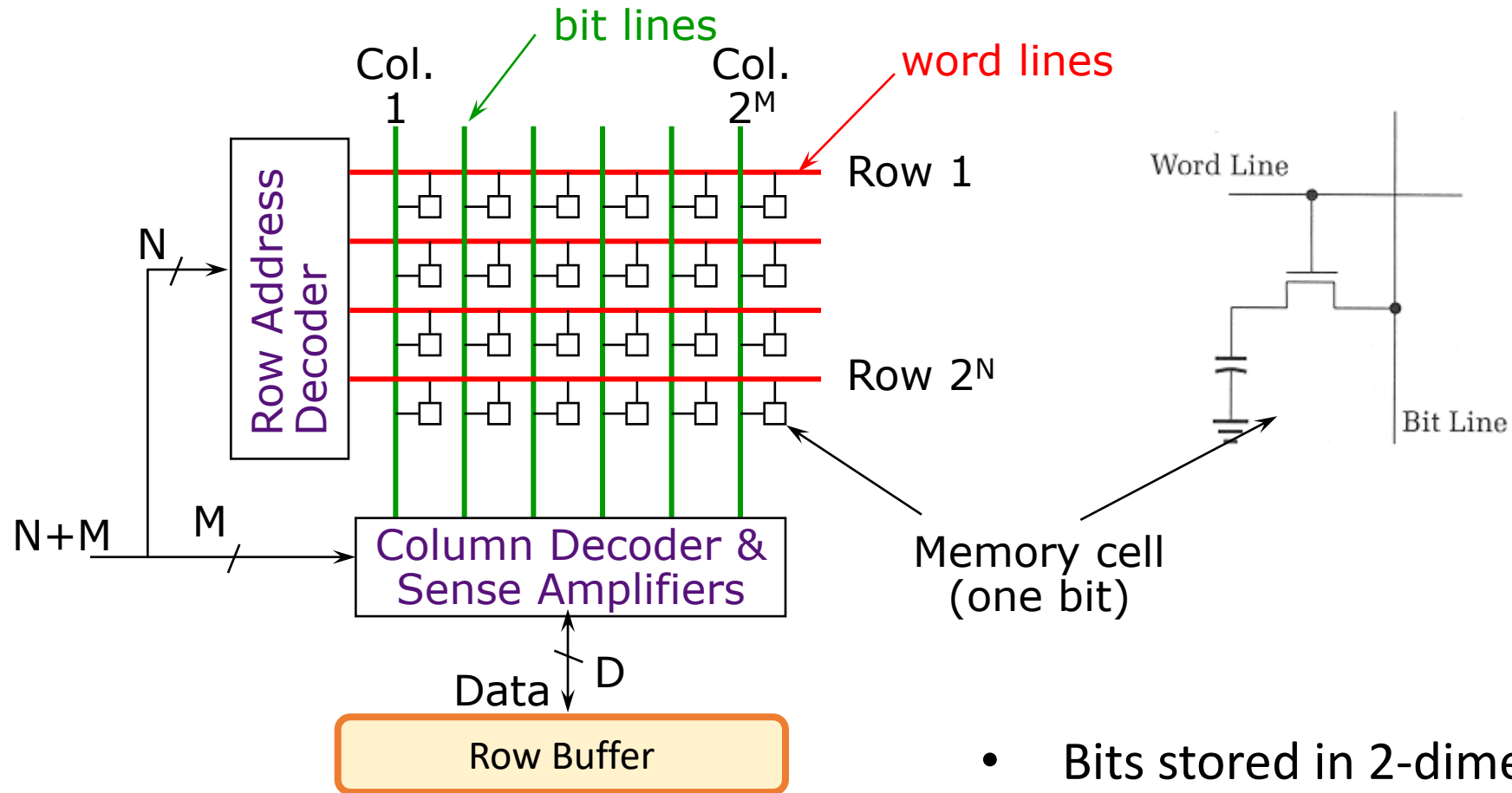
- Each bit in DRAM is stored in a “cell” using a *capacitor*
- *Read is destructive*
- DRAM cells lose their state over time (hence **Dynamic** RAM)
- Data stored in DRAM cells needs to be “**refreshed**” at a regular interval



Why we widely use DRAM given some of its unappealing properties? (compare it with SRAM and SSD)

- Speed (2-10x slower than SRAM)
- Density (20x denser than SRAM)
- Cost (~100x cheaper per MB)

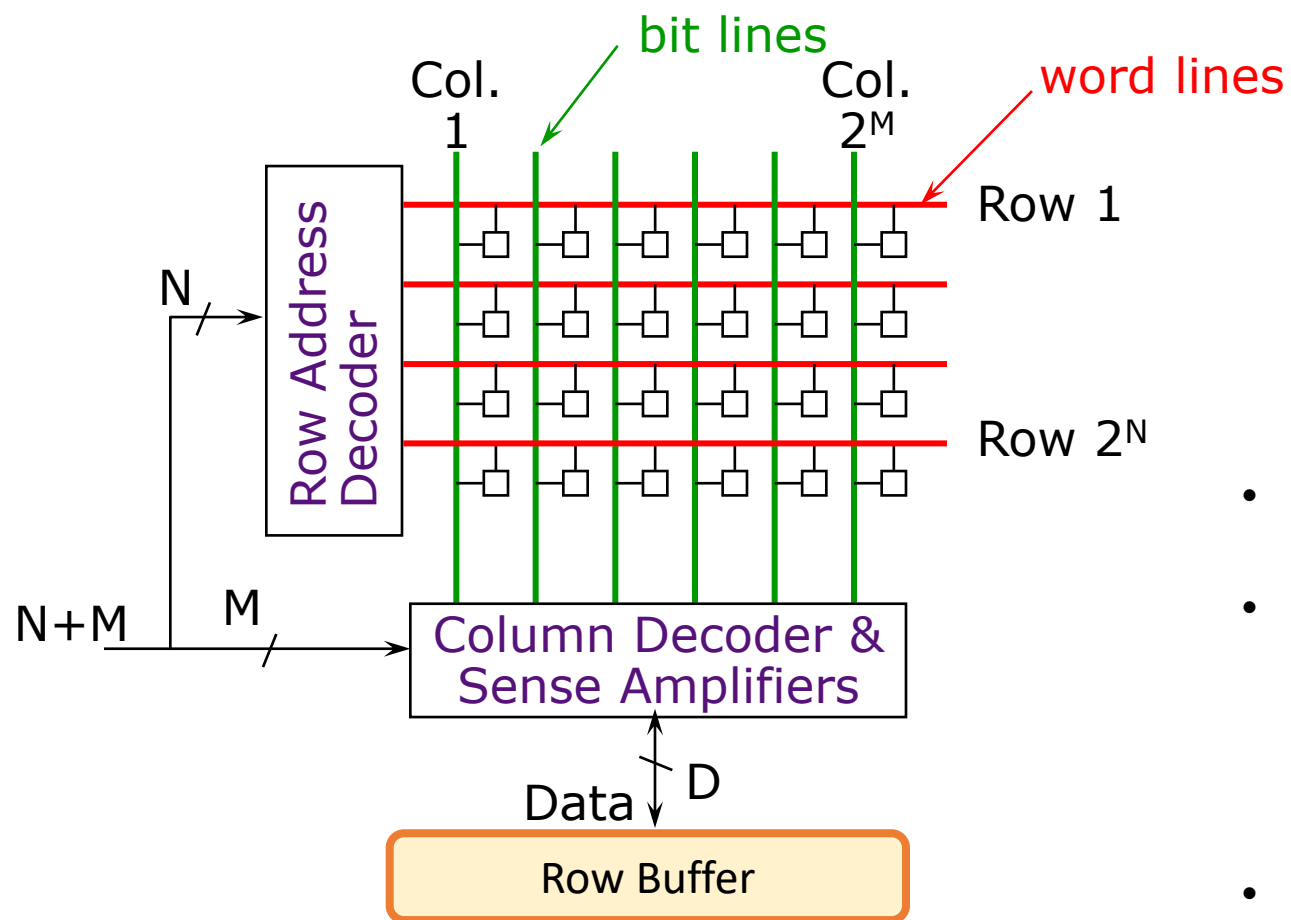
# DRAM Architecture



- Bits stored in 2-dimensional arrays on chip
- Question: why read the entire row?



# DRAM Refresh



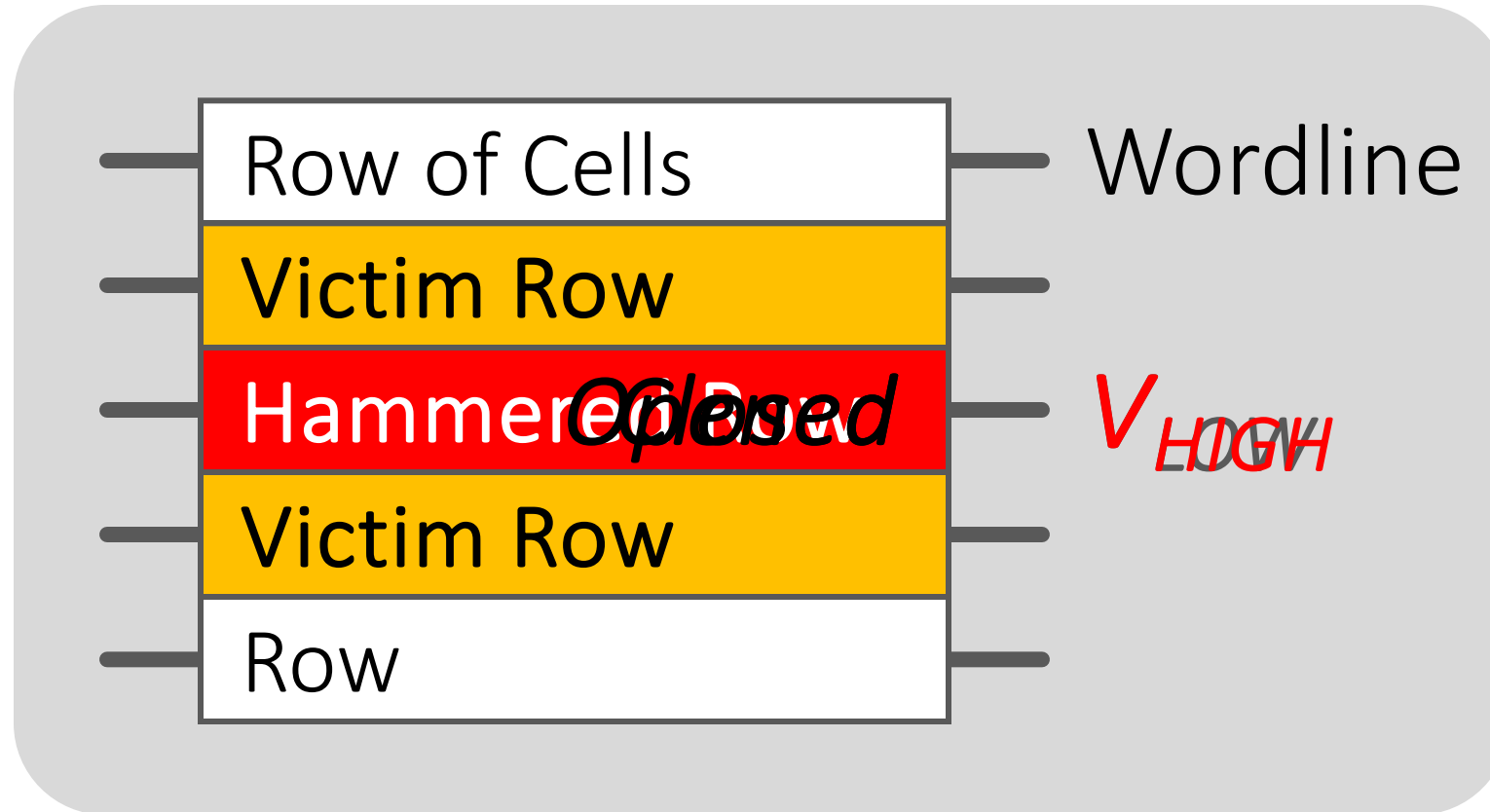
- How to do refresh?
- Performance penalty of refresh
  - In an 8Gb memory, upwards of 10% of time is spent in refresh!
- The common refresh interval: **64ms**

# Aside: Cold Boot Attacks

	Seconds w/o power	Error % at operating temp.	Error % at $-50^{\circ}\text{C}$
<b>SDRAM (1999)</b>	60	41	(no errors)
	300	50	0.000095
<b>DDR (2001)</b>	360	50	(no errors)
	600	50	0.000036
<b>DDR (2003)</b>	120	41	0.00105
	360	42	0.00144
<b>DDR2 (2007)</b>	40	50	0.025
	80	50	0.18

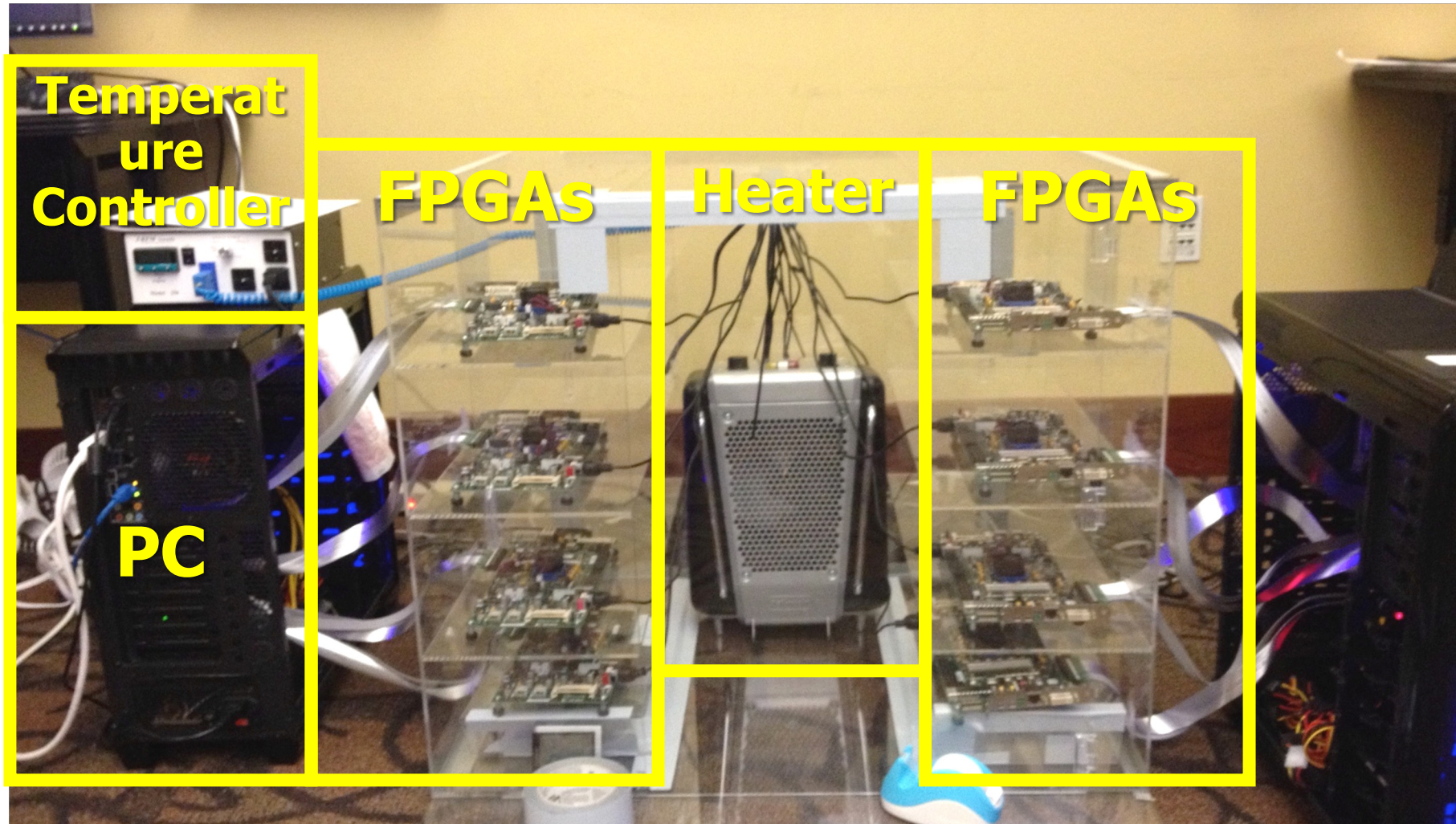


# See RowHammer Again



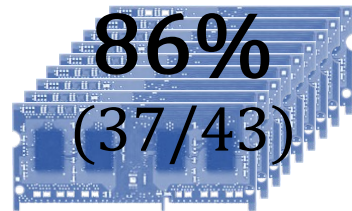
**Observation:** Repeatedly accessing a row enough times **between refreshes** can cause disturbance errors in nearby rows

# Infrastructures to Understand Rowhammer



# Most DRAM Modules Are Vulnerable

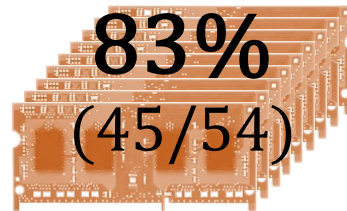
A company



Up to  
 $1.0 \times 10^7$

errors

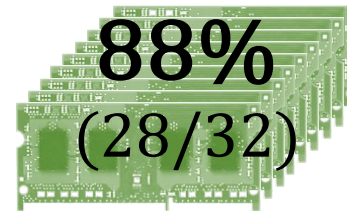
B company



Up to  
 $2.7 \times 10^6$

errors

C company

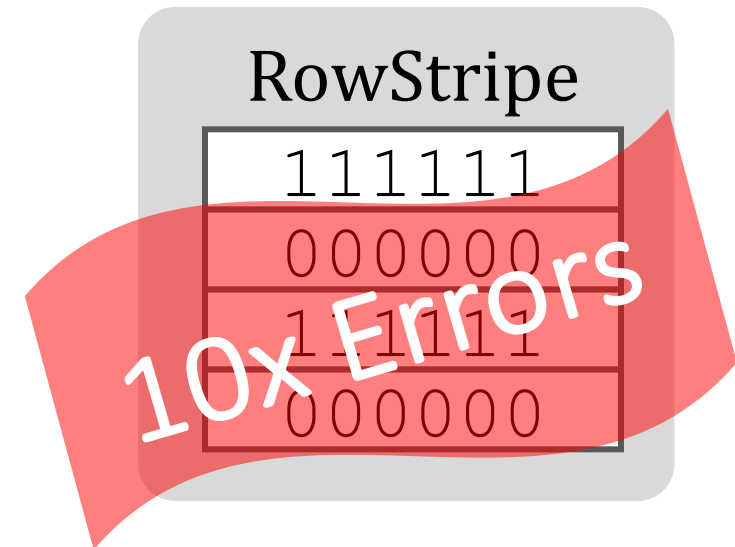
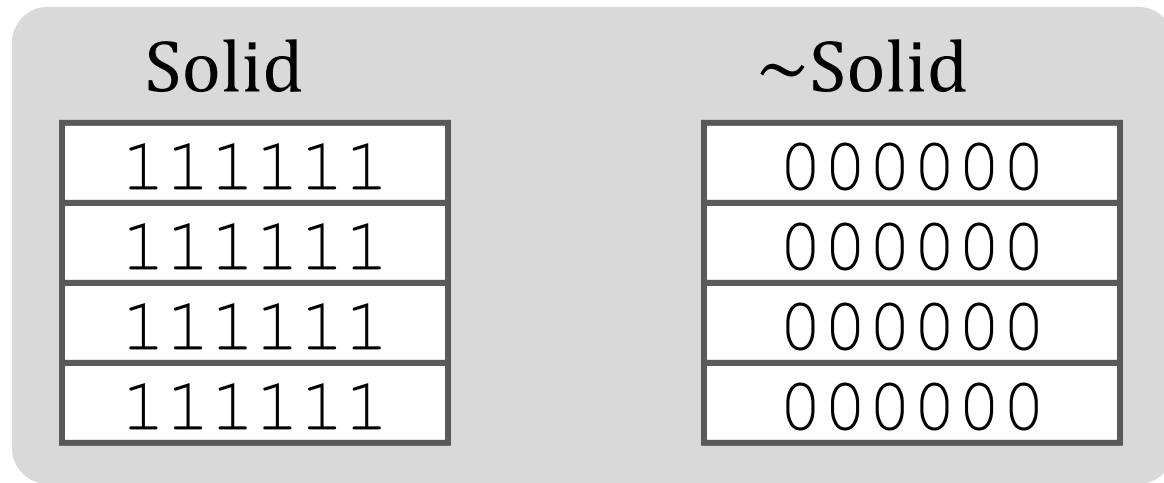


Up to  
 $3.3 \times 10^5$

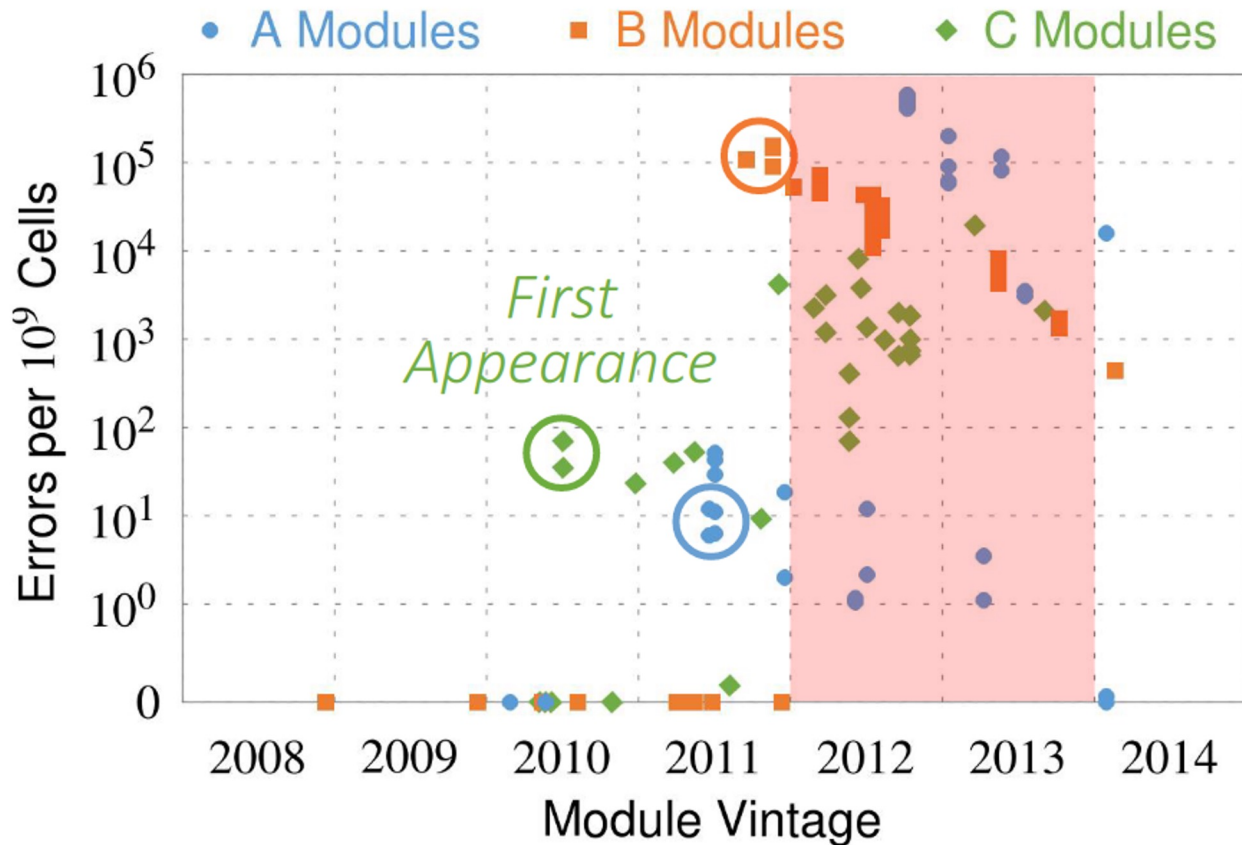
errors

# Study RowHammer Characteristics

- Highly local nature of the bit-flipping capability
- The probability of bitflips are data-dependent



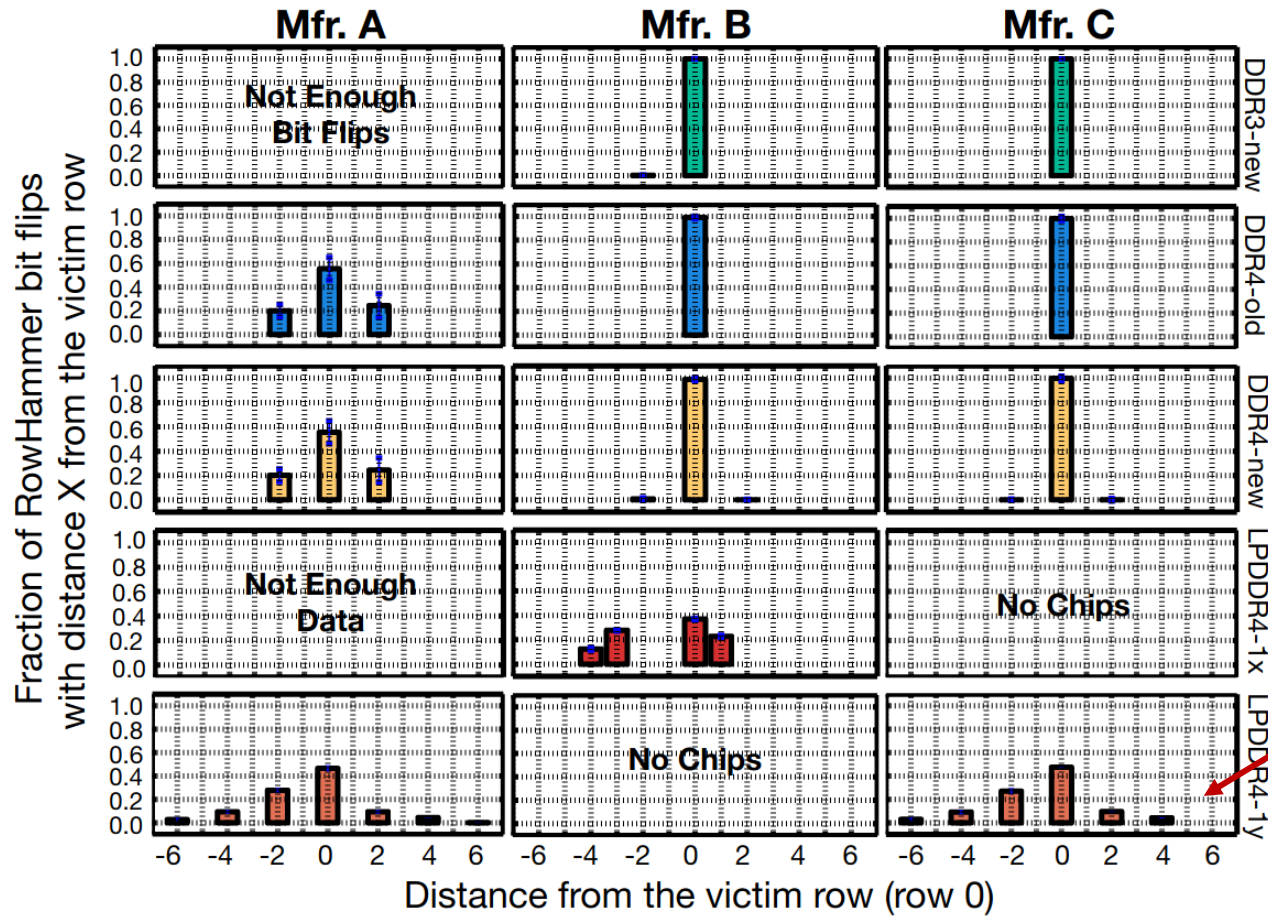
# Density Trends



- As DRAM gets physically denser, it becomes even **more vulnerable!**
- Only a few thousand hammer iterations are required on modern DRAM to cause a bit-flip



# Density Trends



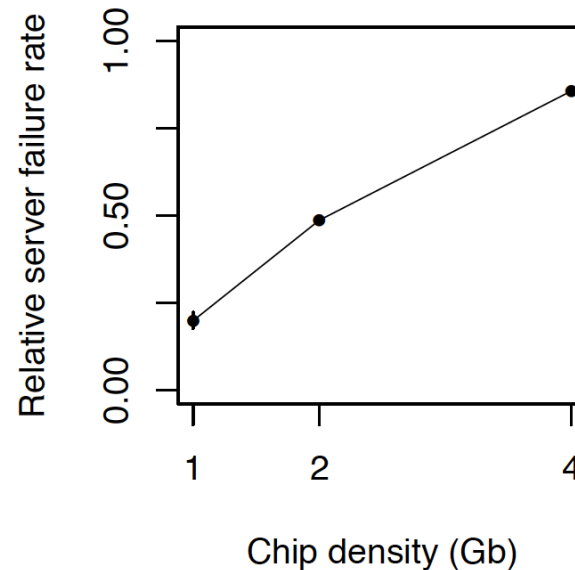
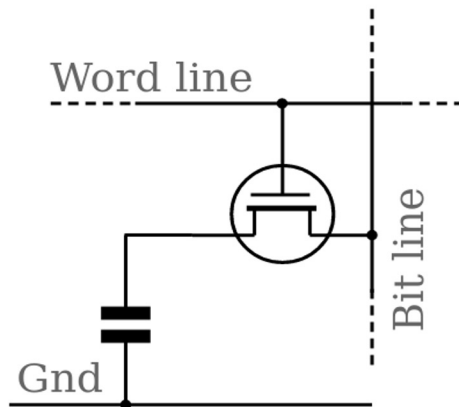
Denser DRAM also can result in flips in rows which are *not directly adjacent* to the attacker





# Technology Scaling

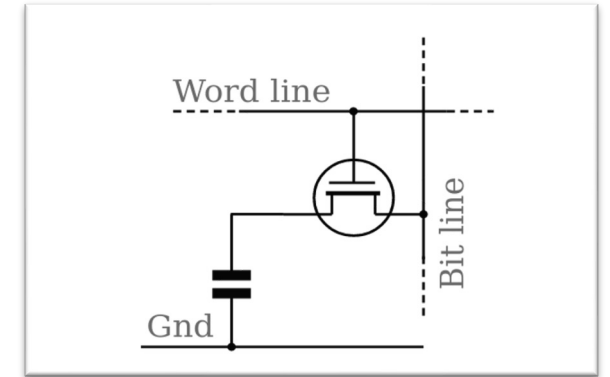
- Capacitor must be large enough for reliable sensing
- The access transistor should be large enough for low leakage and high retention time
- Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]



Data from all of Facebook's servers worldwide

# Why Is RowHammer Happening?

- DRAM cells are too close to each other
  - They are not electrically isolated from each other
- Access to one cell affects the value in nearby cells
  - Due to **electrical interference** between the cells and wires used for accessing the cells
  - Also called cell-to-cell coupling/interference
  - Other hypothesis exists
- Example: When we activate (apply high voltage) to a row, an adjacent row gets slightly activated as well
  - Vulnerable cells in that slightly-activated row lose a little bit of charge
  - If row hammer happens enough times, charge in such cells gets drained

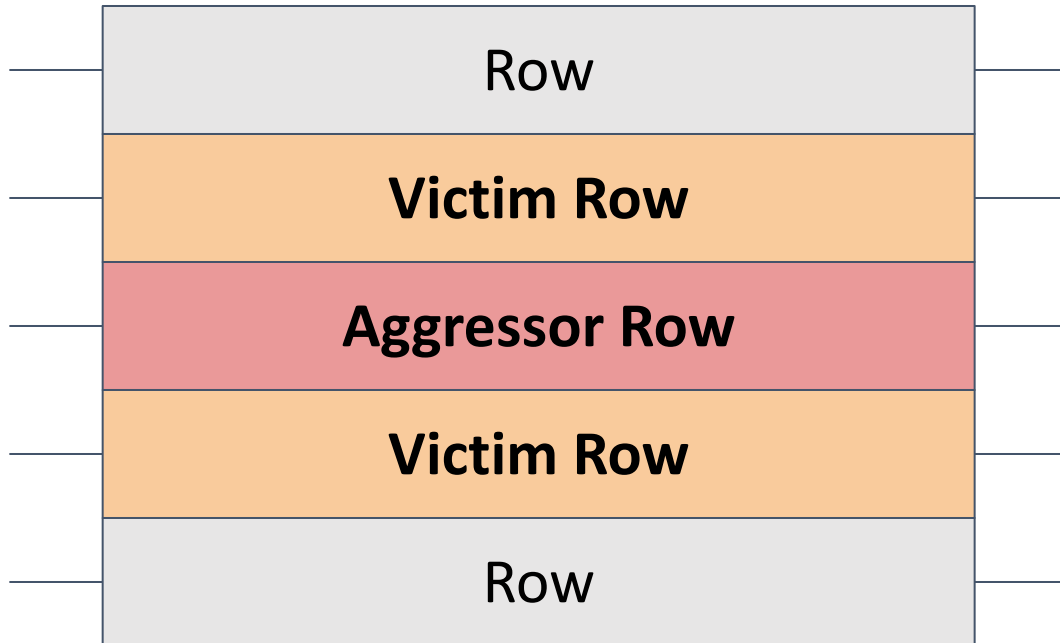


# RowHammer Attacks in Action



# RowHammer Attacks in Practice

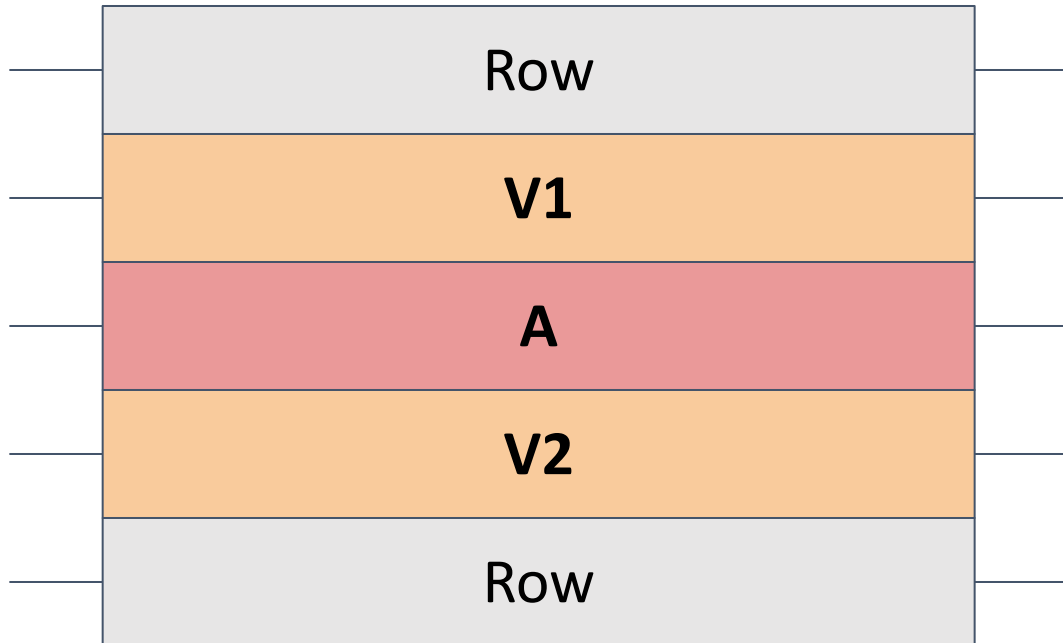
- Aggressor Row = Hammered Row



## Challenges:

1. How to hammer? Need to access aggressor row enough times between refreshes.
2. Address mapping. How to find addresses map to neighboring rows?
3. How to make victim's data to be located inside vulnerable cells?

# Hammer Attempt #1: repeat accesses

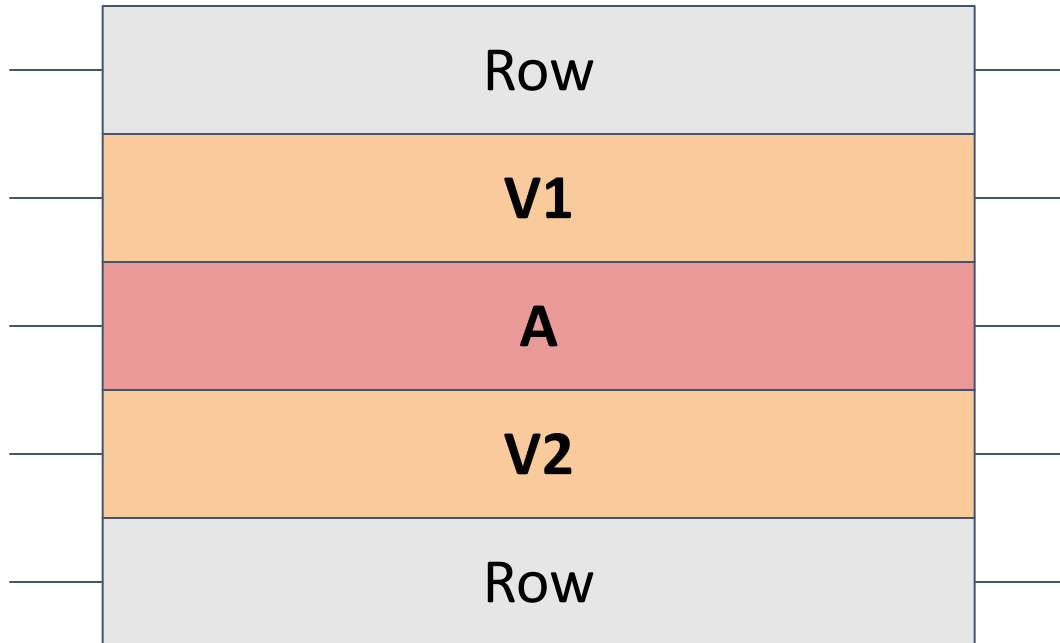


```
loop:  
  mov (A), %eax  
  
  mfence  
  jmp loop
```

Will this work?  
Why?

No. Because we will hit the cache.

# Hammer Attempt #2: use cflush

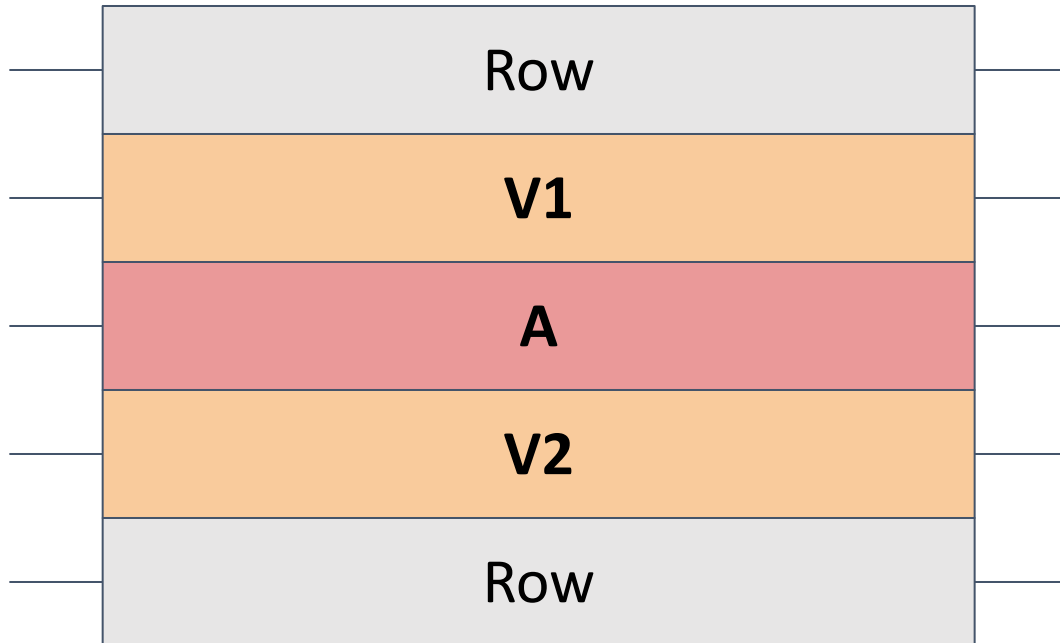


```
loop:  
  mov (A), %eax  
  cflush (A)  
  mfence  
  jmp loop
```

Will this work?  
Why?

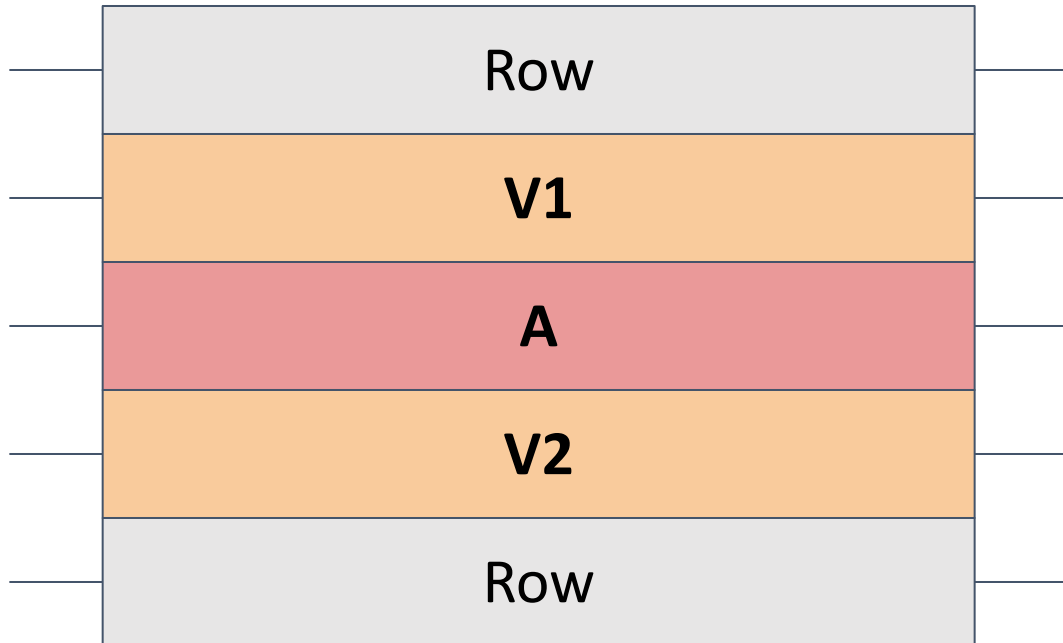
No. Because we will hit the row buffer.

# Hammer Attempt #3: force row open/close



```
loop:  
  mov (A), %eax  
  mov (A_dummy), %ecx  
  
  clflush (A)  
  clflush (A_dummy)  
  
  mfence  
  jmp loop
```

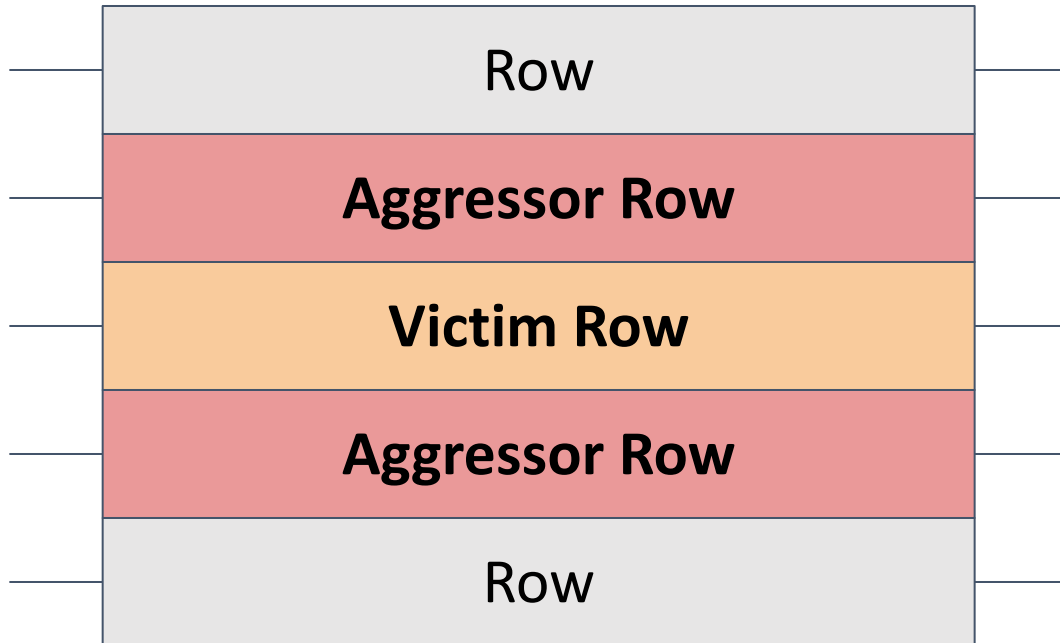
# “Single-Sided” Rowhammer



```
loop:  
  mov (A), %eax  
  mov (A_dummy), %ecx  
  
  clflush (A)  
  clflush (A_dummy)  
  
  mfence  
  jmp loop
```



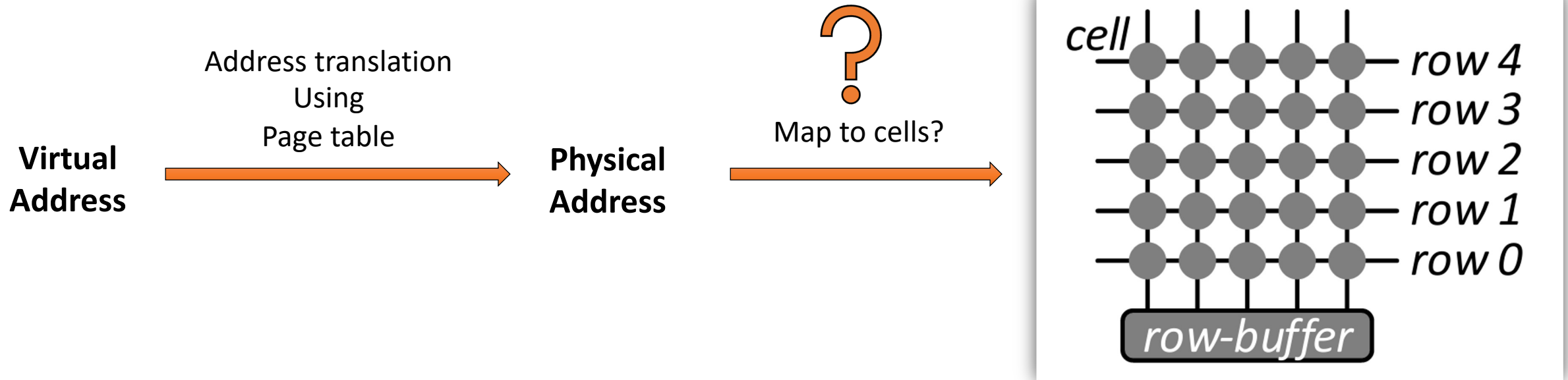
# “Double-Sided” Rowhammer

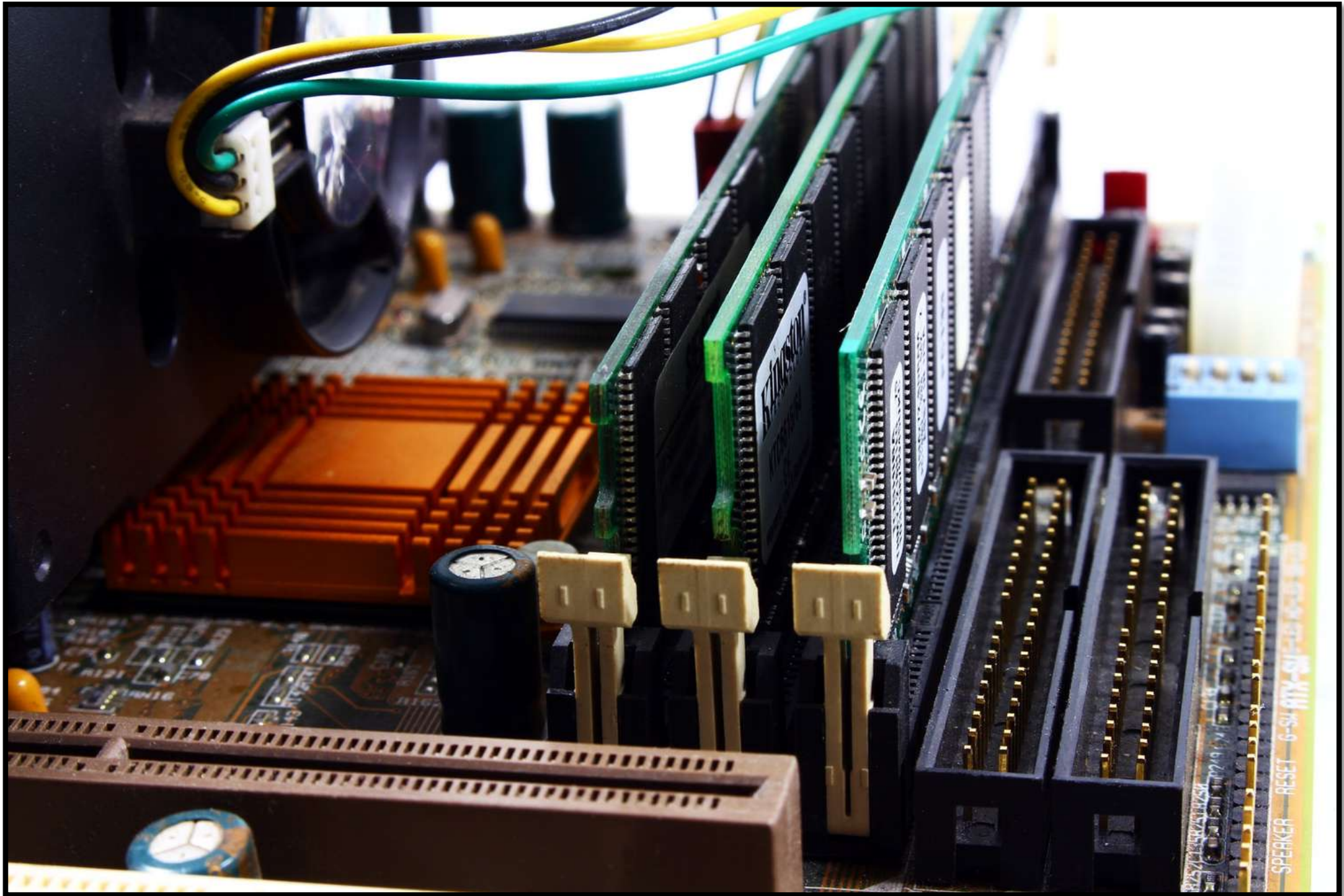


## **Increase the stress:**

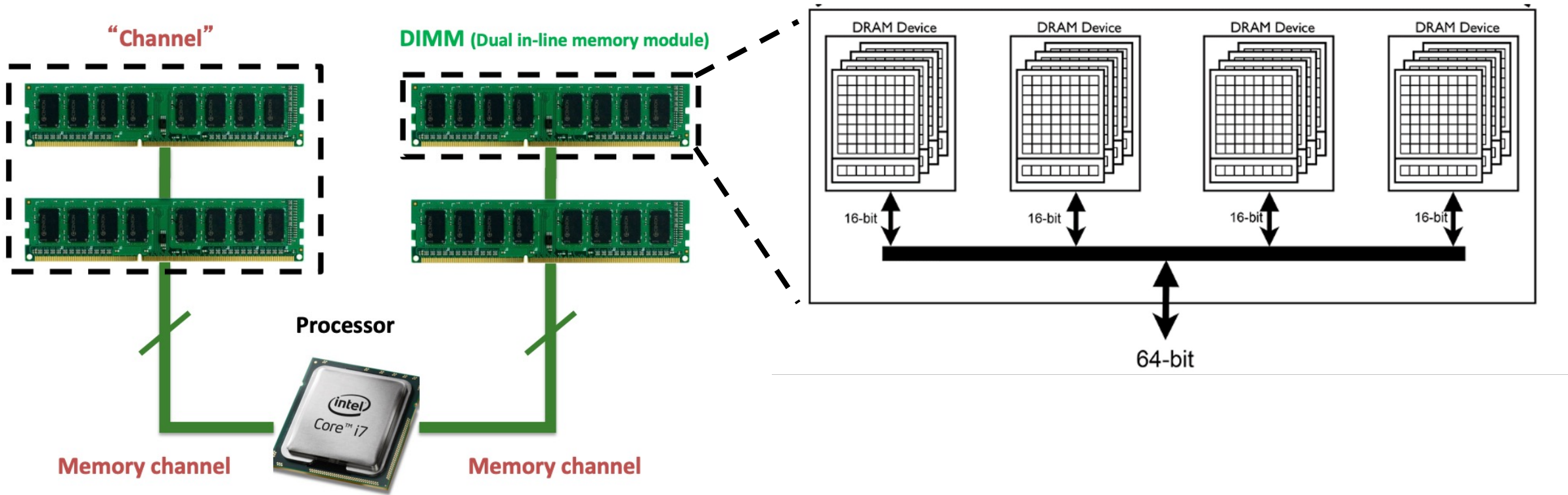
Repeatedly accessing both adjacent rows significantly increases the error rate of the victim row

# Challenge #2: DRAM Addressing

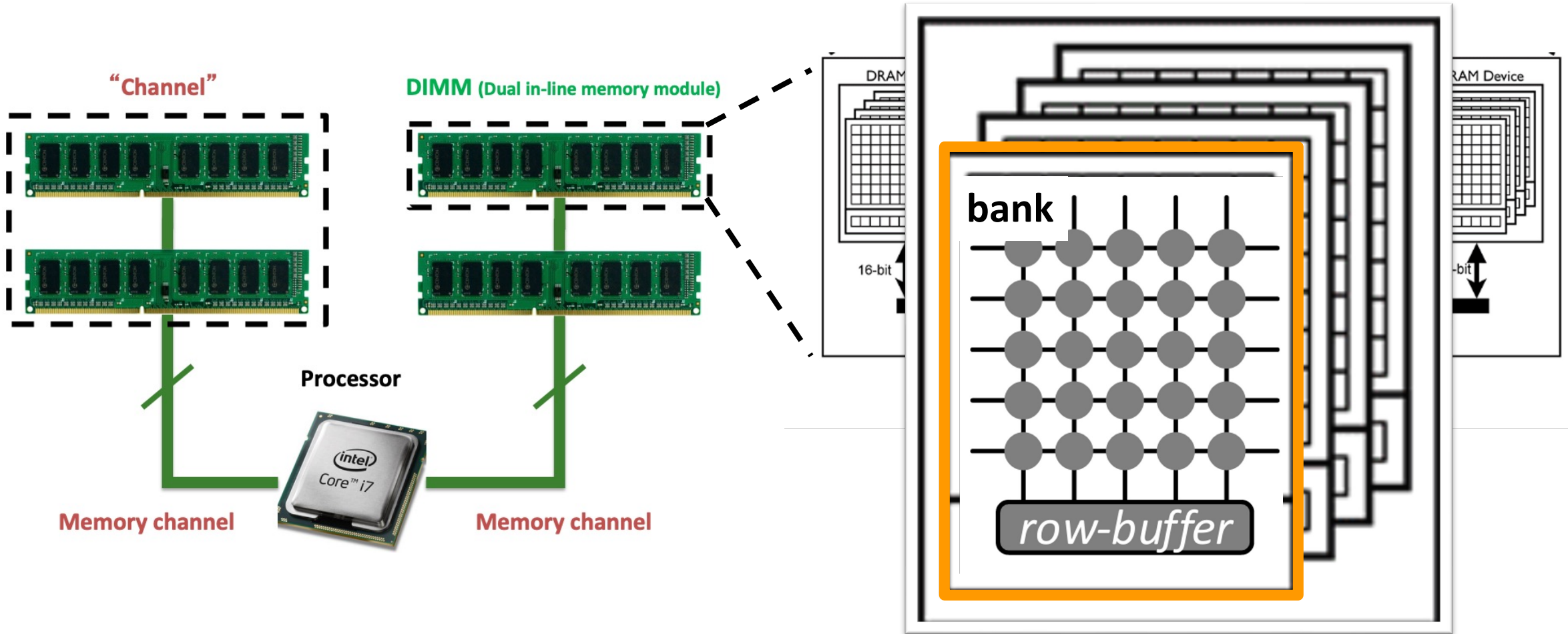




# DRAM Organization: Top-down View

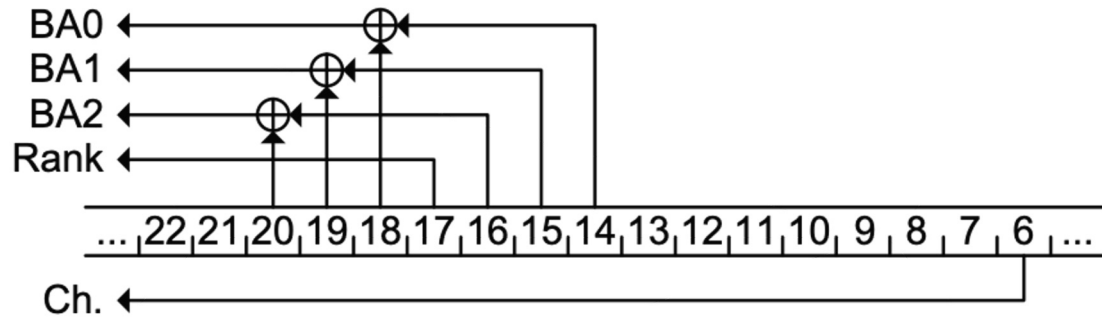


# DRAM Organization: Top-down View

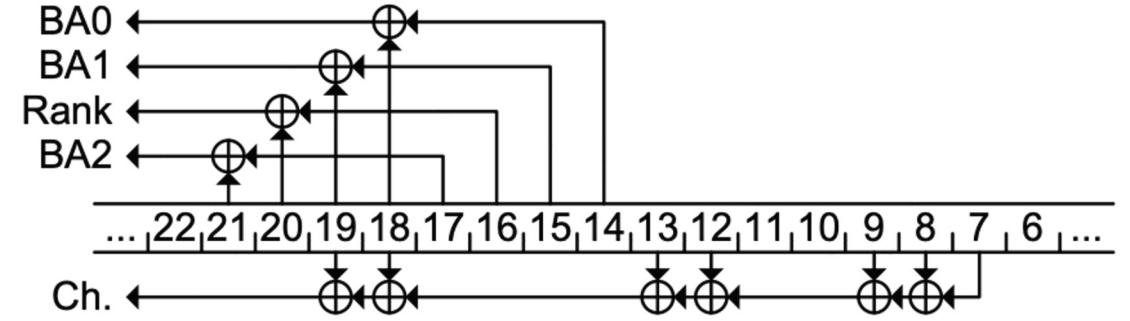


Channel -> DIMM -> Rank -> Chip -> Bank -> Row/Column

# Address Mapping Examples



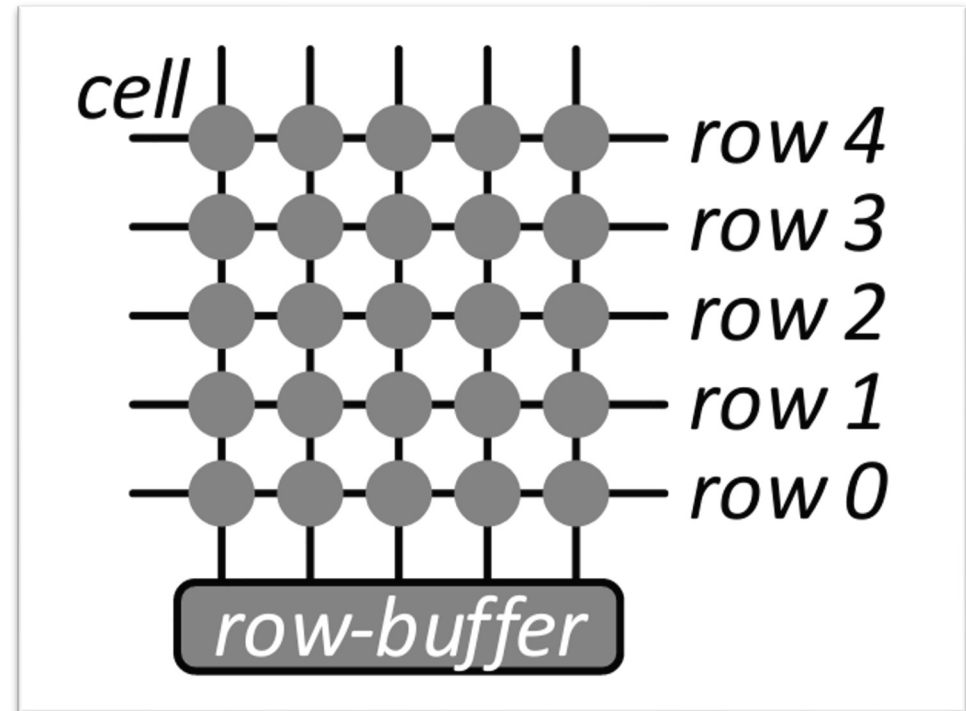
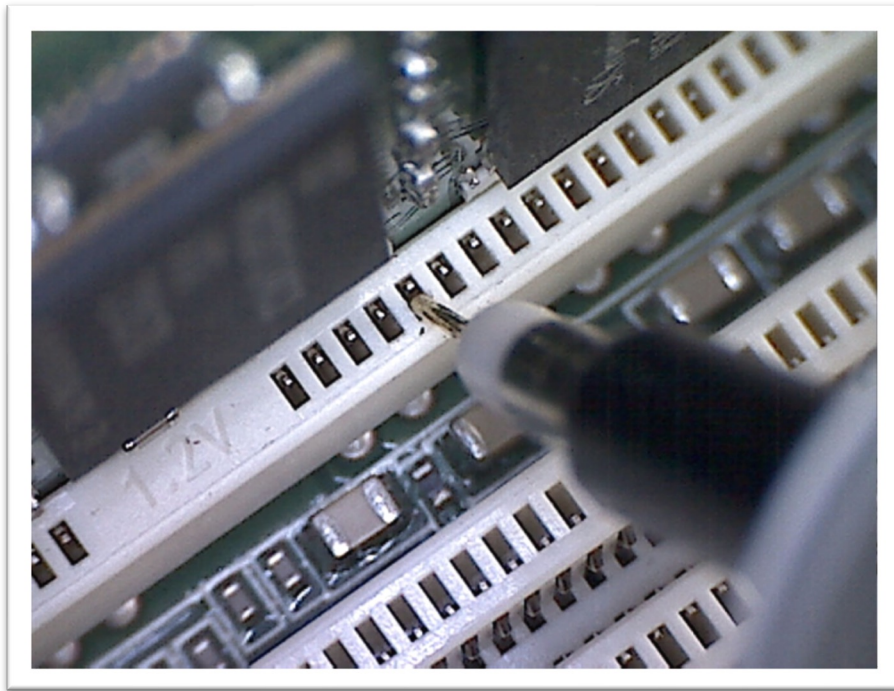
(a) Sandy Bridge – DDR3 [23].



(b) Ivy Bridge / Haswell – DDR3.

# Reverse Engineer the Mapping

- Approach #1: Physical Probe
- Approach #2: Timing Side Channel via Row Buffer



# Attacks Built Upon RowHammer





# Native Client (NaCl) Sandbox Escape

- NaCl is a sandbox for running native code (C/C++)
- Runs a “safe” subset of x86, statically verifying an executable
- Use bit flips to make an instruction sequence unsafe

## Example “Safe” Code:

```
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.
```

# Native Client (NaCl) Sandbox Escape

We can flip bits to allow for (unsafe) non 32-byte-aligned jumps

## Exploited “Safe” Code:

```
andl $~31, %ecx // 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.
```

# Kernel Privilege Escalation

What could happen if a user could gain direct write access to a page table?

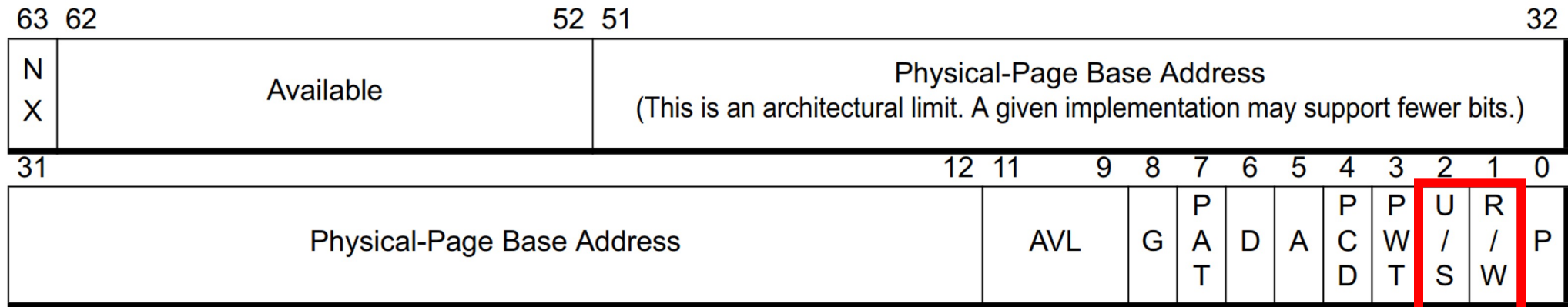


Figure 5-21. 4-Kbyte PTE—Long Mode

# Steel SSH Keys

Observation: the likelihood of Rowhammer bitflips can be influenced by the data pattern.



(b) Single-sided Rambleed. Here, the sampling page (A1) is neighbored by the secret-containing page (S) on a single side.

# Other Attacks

- Virtual machine takeover
  - Use page de-duplication to corrupt host machine
- OpenSSH attacks
  - Overwrite internal public key with attacker controlled one
- Drammer
  - Rowhammer privilege escalation on ARM
  - Utilizes determinism in page allocation to target vulnerable DRAM rows
- Rowhammer.js
  - Remote takeover of a server vulnerable to rowhammer

**Without memory integrity, *any* software-based security mechanism is insecure!**

# Rowhammer Mitigations?

- Manufacturing “better” chips
- Increasing refresh rate
- Error Correcting Codes
- Targeted row refresh (TRR) - Used in DDR4!
- Retiring vulnerable cells
- Static binary analysis
- User/kernel space isolation in physical memory

# Rowhammer Solutions?

- Manufacturing “better” chips
- Increasing refresh rate
- Error Correcting Codes
- Targeted row refresh (TRR) - Used in DDR4!
- Retiring vulnerable cells
- Static binary analysis
- User/kernel space isolation in physical memory

cost

Performance, power

cost, power

cost, power, complexity

cost, power, complexity

security

# Takeaways

Reliability  $\leftrightarrow$  Security Implications



Stop

(a) Normal



Yield

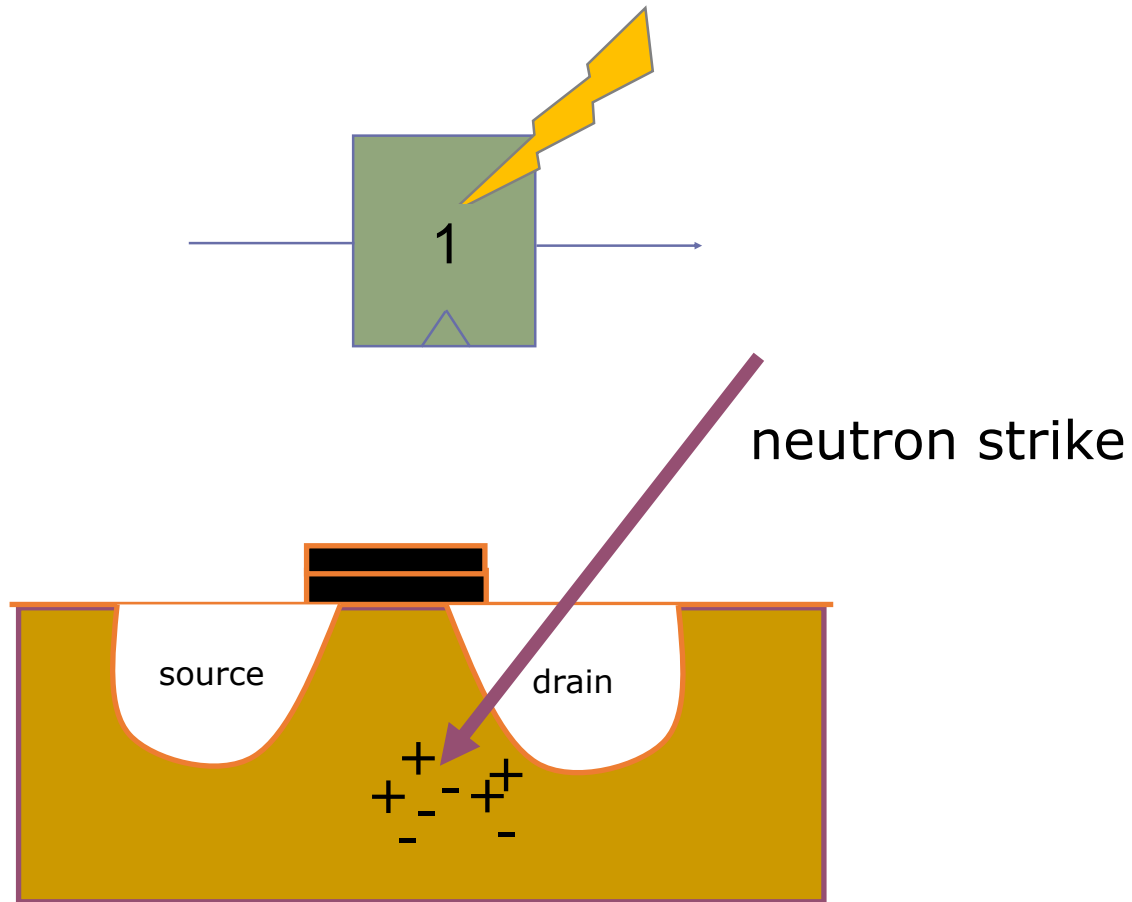


Speed Limit

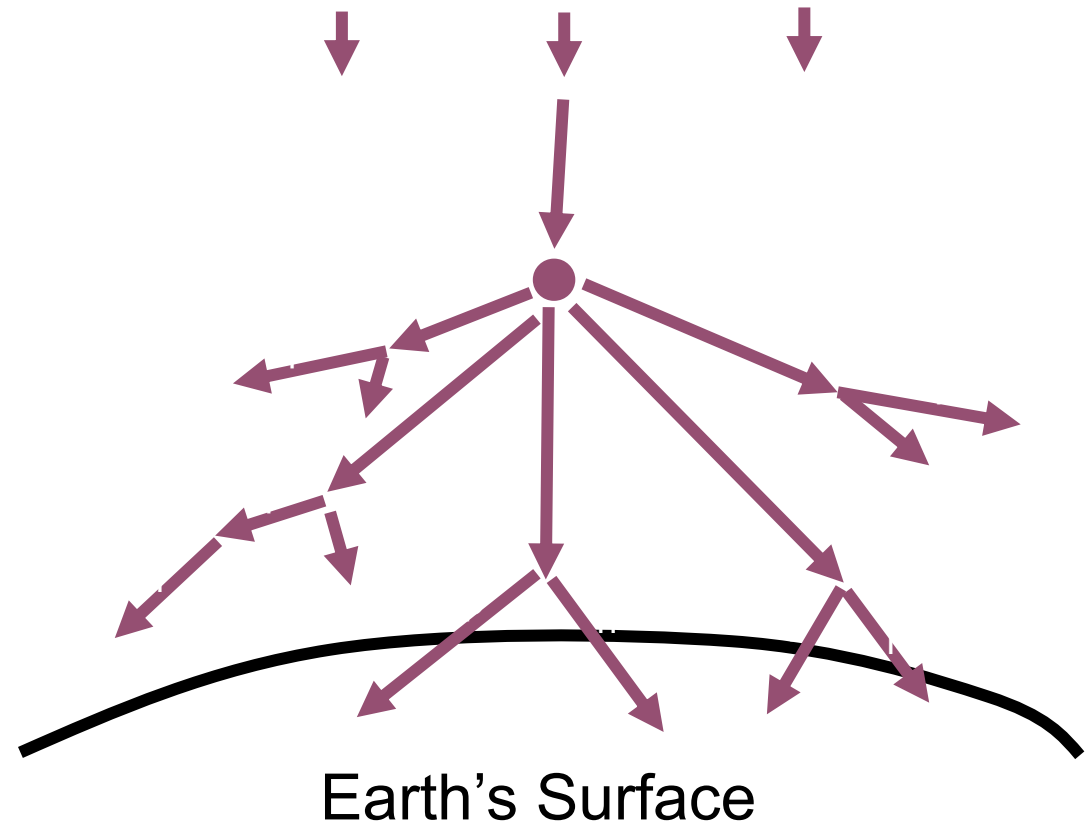
(b) Attack



# Reliability Problems in 2000s



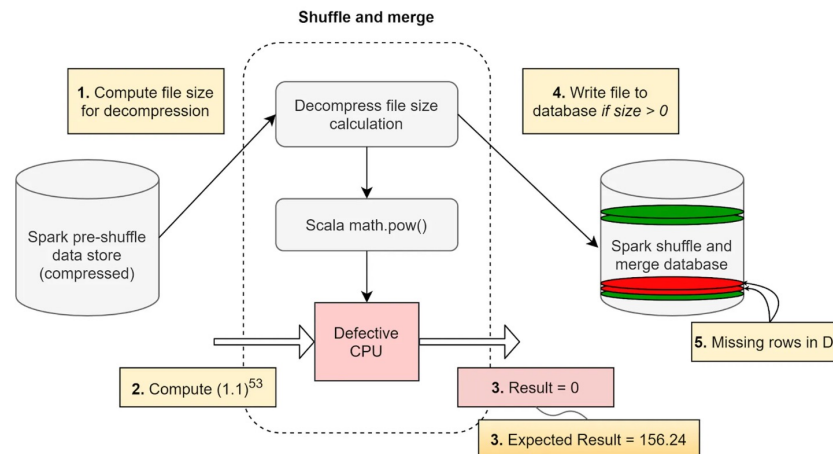
Transistor Device



Earth's Surface

# Other Reliability Problems in 2020s

- Silent Data Corruption (SDC)
  - Cloud companies noticed SDC is a widespread problem for large-scale infrastructure systems.
- Problems
  - Long error detection latencies: taking days to weeks
  - Scalability



Example errors:

$$\text{Int}[(1.1)^3] = 0, \text{ expected} = 1$$

$$\text{Int}[(1.1)^{107}] = 32809, \text{ expected} = 26854$$

$$\text{Int}[(1.1)^{-3}] = 1, \text{ expected} = 0$$

“Cores that don’t count” by Google, HotOS, 2021

“Silent data corruption at Scale” by Facebook, Arxiv, 2021

**Next:**

**Spring Break**

